Connecting streams and databases

Gian Merlino
Druid committer • Cofounder @ Imply

Overview

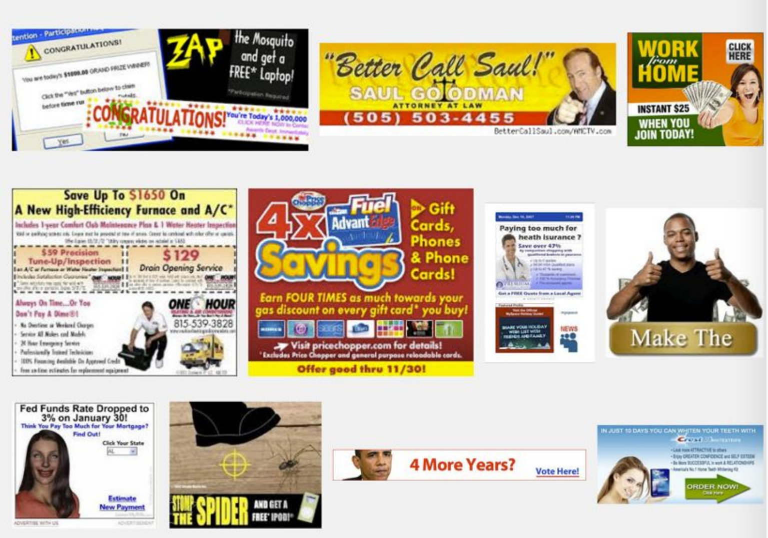
Why streams and why databases?

Things you may care about

How popular stream systems work

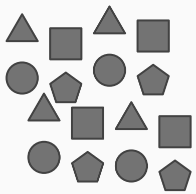
How we deal with streams in Druid

Stream processing



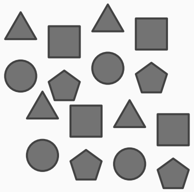
Streams

Stream processing

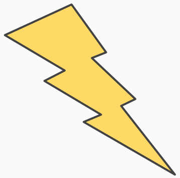


Streams

Stream processing

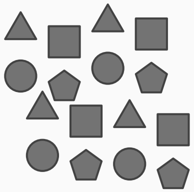


Streams



Actions

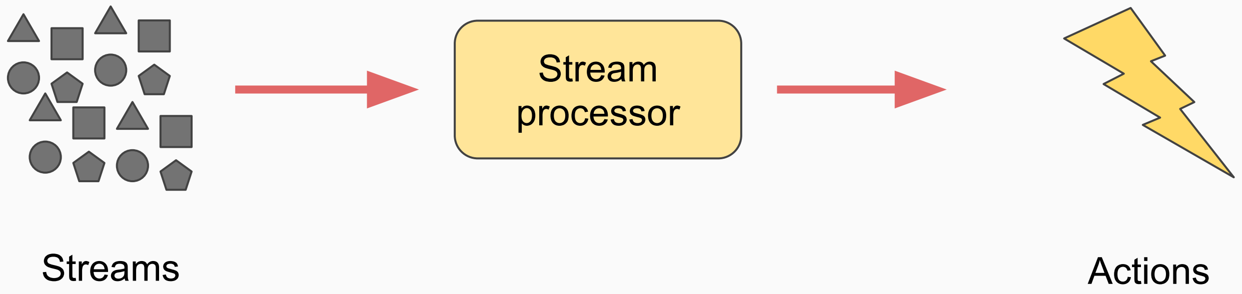
Stream processing



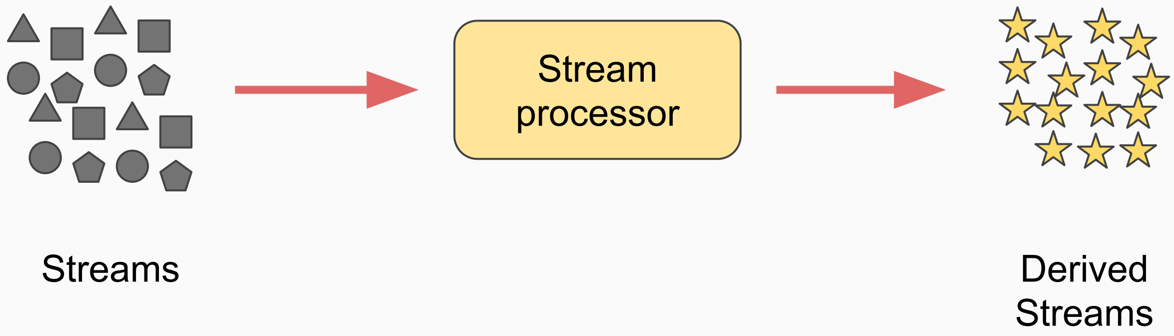
Streams

You

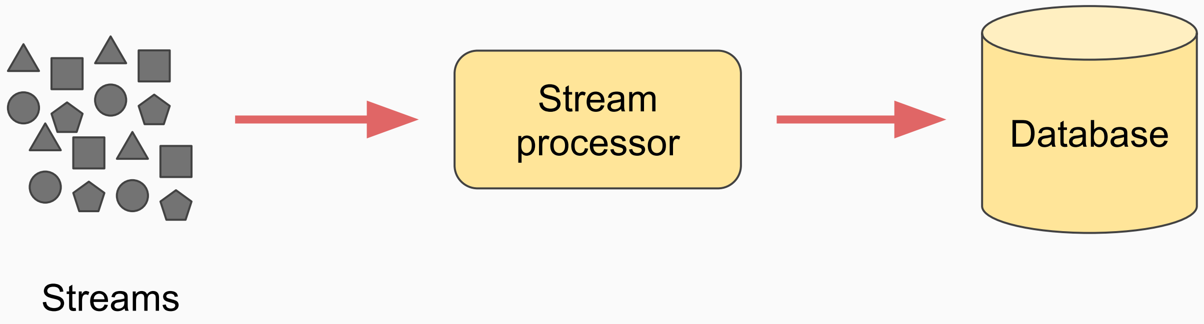
Stream processing



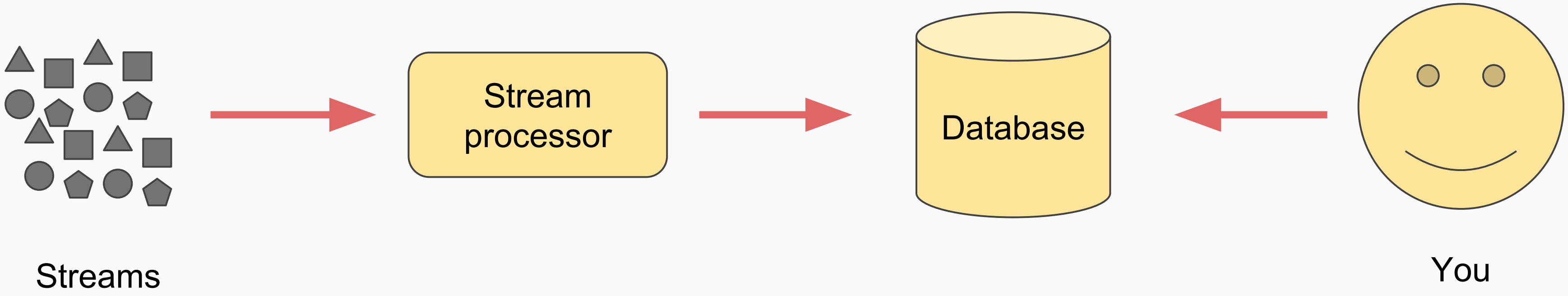
Stream processing



Stream processing



Stream processing



Why **streams?**

- Real-time monitoring
- Real-time response
- Shorter feedback loops
- Better user experience

Why **stream processing**?

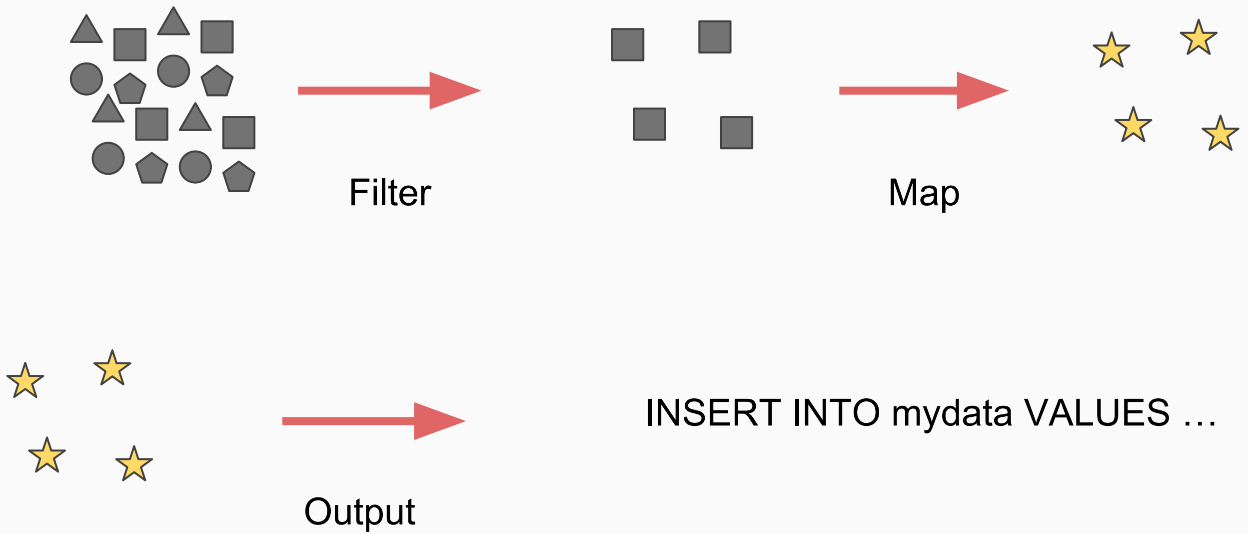
- Original streams are not exactly what we want
- Common things to want
    - Enhancement
    - Session reconstruction (and other joins)
    - Load into databases

Why **databases?**

- Lots of questions to ask
- Streaming through raw data, for every question, is slow
- Faster to load a derivative into a database, query it there
  - DB strengths: ad-hoc search, aggregation, key lookup
  - DB weaknesses: joining big distributed tables, joining external data

Stream operations

Basic operations



Grouping operations

Grouping operations

- Tricky!
- Data points for a window may come in “late”
- Windows may be aligned (e.g. aggregates)
- Windows may be unaligned (e.g. sessions)

Requirements

Requirements

- Correctness
- Latency
- Cost
- (Thanks, Akidau et. al)

Correctness

- Want accurate reflection of reality

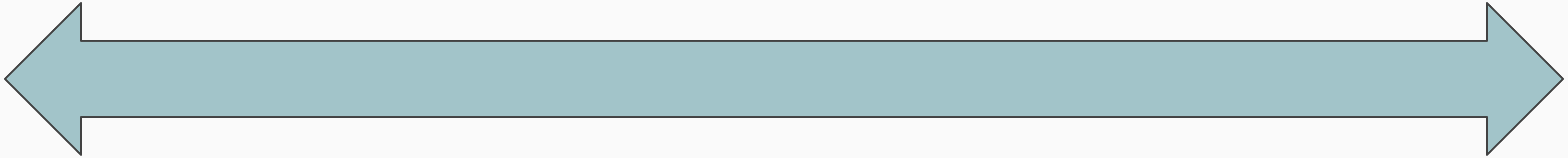
Correctness

- Message processing guarantees
    - None
    - At most once
    - At least once
    - Exactly once

Correctness

- Window emitting guarantees
  - Wait for “enough” data before emitting, and emit once
  - Emit periodic updates

Latency



Very low latency
subsecond

“Low” latency
seconds – minutes

High latency
hours – days

Data pipelines

Goals

- Low-latency results
- Strong correctness guarantees
- Ability to do backfills

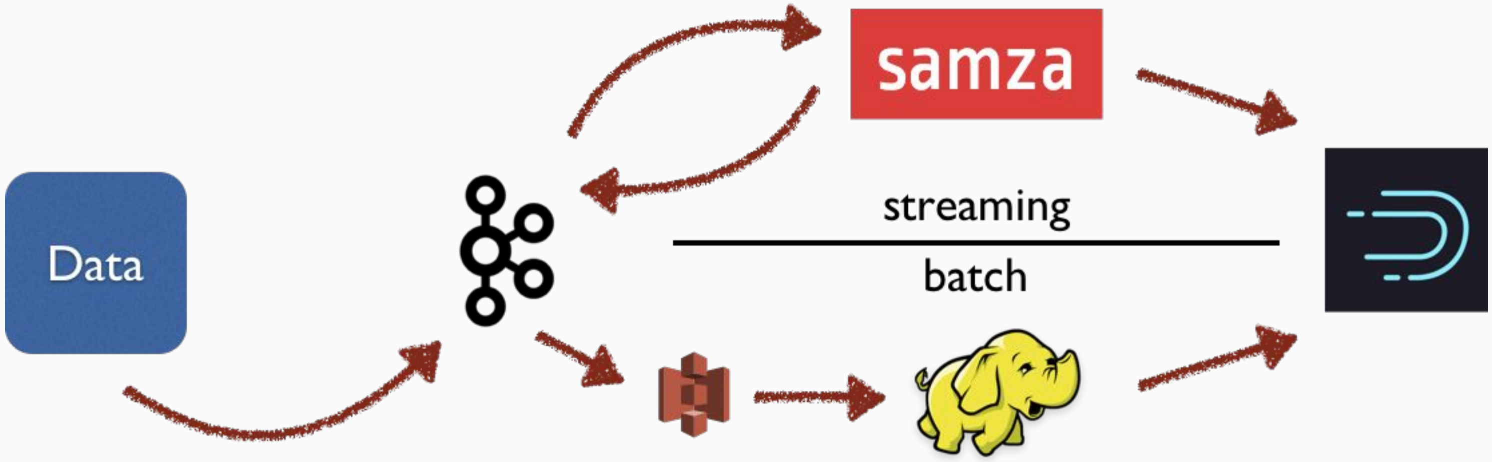
Why backfills?

- Bugs in processing code
- Need to restate existing data
- Limitations of some current streaming software

Backfills: Lambda

- Hybrid batch/realtime a.k.a. “lambda architecture”
- Backfills automated or on-demand
- Pros: Can achieve goals with a wide variety of OSS
- Cons: Operations and development are complex

Backfills: Lambda



Backfills: Lambda

- Batch technologies
    - Hadoop Map/Reduce
    - Spark
- Streaming technologies
    - Samza
    - Storm
    - Spark Streaming

Backfills: Lambda

- Software exists to simplify development
  - Summingbird
  - Google Cloud Dataflow
  - Starfire (internal tool)

Backfills: Stream replay

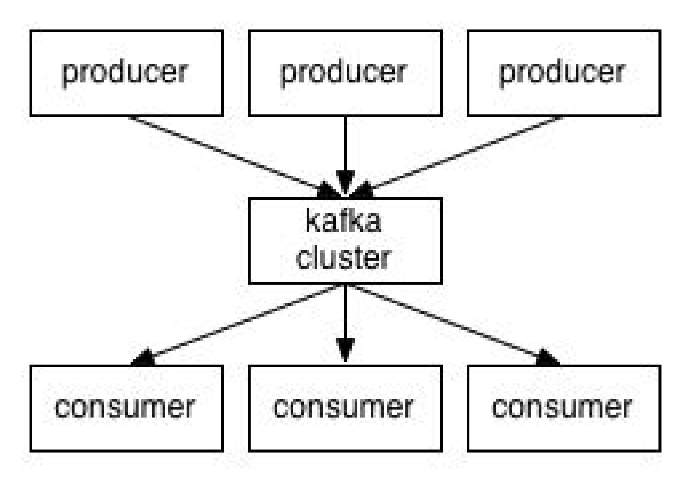
- Stream replay a.k.a. “kappa architecture”
- Backfills on demand
- Simpler development and operations
- Workable if stream processing guarantees are strong enough

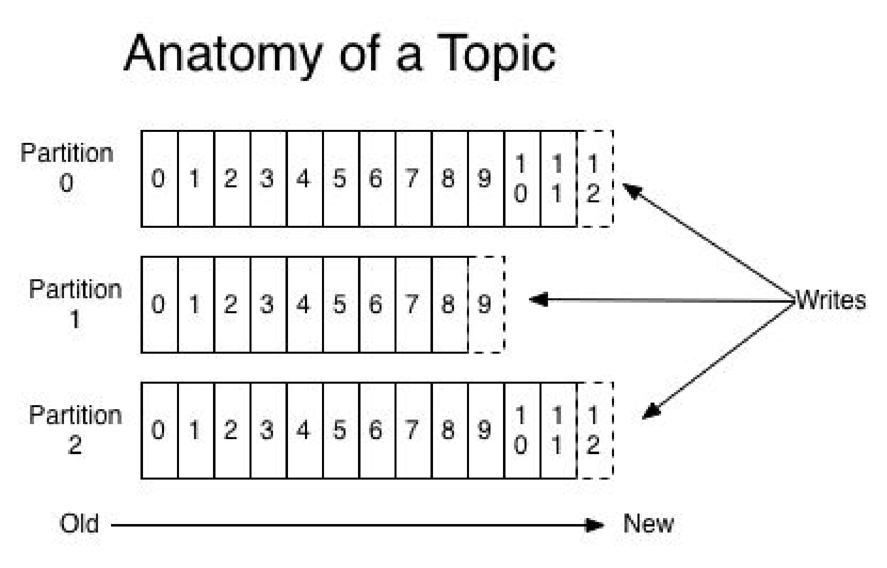
Side note on batch processing

- Stream and batch processing not too different on unbounded datasets
- Batch processors must still deal with late data

Streaming systems

Kafka





http://kafka.apache.org/documentation.html

Kafka: API

- Producer: Send message to a (topic, partition)
- Consumer: Read messages from a (topic, partition)
- Very low latency
- Can do simple operations directly with the Kafka API
- More complex processing is easier with a “real” stream processor

Kafka: API

- Possible to integrate closely, and efficiently, with databases
- Not an accident

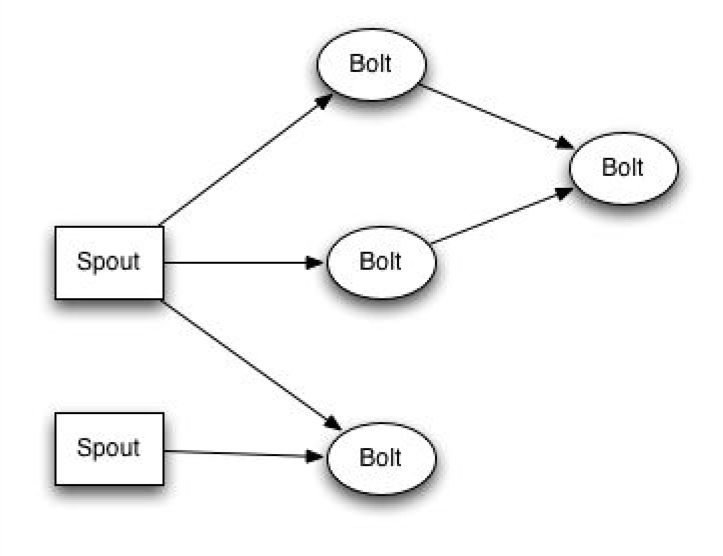
Kafka: Guarantees

- Producer: At-least-once, if configured appropriately
- Consumer
  - At-least-once straightforward with high-level consumer
  - Exactly-once (from Kafka data!) can be done with more work

Kafka: Guarantees

- Exactly-once strategies
- Naturally unique message IDs
  - Must assign outside of Kafka
  - De-duplicate messages while consuming
  - Must make sure to keep around enough de-duplication data
- Single-writer-per-partition
  - Duplicate messages will be adjacent; ignore them

Storm



http://storm.apache.org/tutorial.html

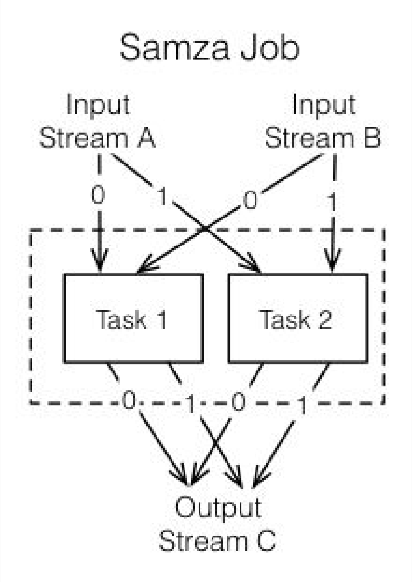
Storm

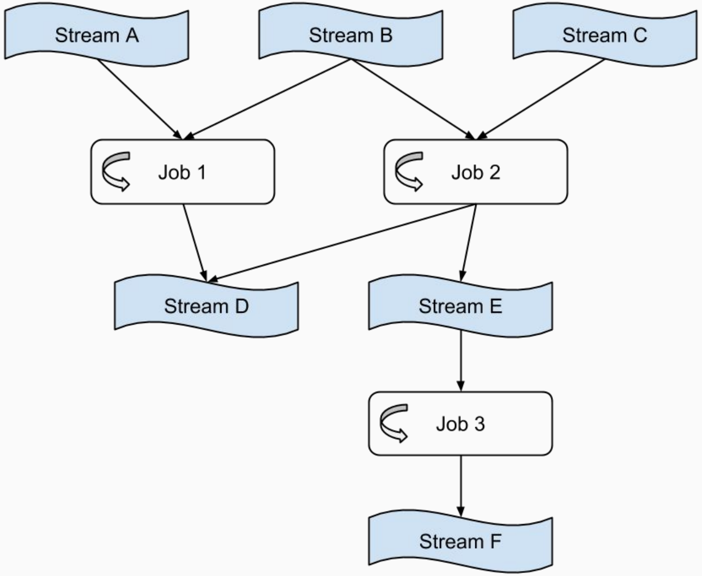
- Messages acked at spout when fully processed
- Spouts typically checkpoint after acks
- At-least-once if spouts are able to replay
- Exactly-once with idempotent operations
- No innate concept of state

Storm / Trident

- Does have concept of state
- Messages grouped into batches
- Each batch given a transaction id (txid)
- Txids globally ordered, meant to be stored in DB
- Skip DB update for stale txids
- Coordination overhead

Samza





http://samza.apache.org/learn/documentation/0.9/introduction/concepts.html

Samza

- Periodically flush output and checkpoint Kafka offsets
- At-least-once
- Exactly-once with idempotent operations

Druid

Druid

- Open source column store
- Designed for fast filtering and aggregations
- Unique optimizations for event data
- Data partitioning/sharding first done on time
- Data is partitioned into defined time buckets (hour/day/etc)

Druid Segments

Timestamp	Page	Views
2015-01-01T00	p1	1
2015-01-01T01	p2	1
2015-01-02T05	p3	1
2015-01-02T07	p4	1
2015-01-03T05	p5	1
2015-01-03T07	p6	1

Timestamp	Page	Views
2015-01-01T00	p1	1
2015-01-01T01	p2	1

Segment_2015-01-01/2014-01-02

Timestamp	Page	Views
2015-01-02T05	p3	1
2015-01-02T07	p4	1

Segment_2015-01-02/2014-01-03

Timestamp	Page	Views
2015-01-03T05	p5	1
2015-01-03T07	p6	1

Segment_2015-01-03/2014-01-04

Partition by time

Rollup on ingestion

timestamp	publisher	advertiser	gender	country	click	revenue
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	USA	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	USA	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	USA	1	0.45
…						
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	1	1.53

Rollup on ingestion

timestamp	publisher	advertiser	gender	country	impressions	clicks	revenue
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Male	USA	1800	25	15.70
2011-01-01T01:00:00Z	bieberfever.com	google.com	Male	USA	2912	42	29.18
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Male	UK	1953	17	17.31
2011-01-01T02:00:00Z	bieberfever.com	google.com	Male	UK	3194	170	34.01

Druid Segments

- Can be built from streams
- Immutable once built: no contention between reads and writes
- Simple parallelization: one thread scans one segment
- Streaming append + atomic batch replace
- Want to avoid having a unique key for messages

Druid: Batch ingestion

- Exactly-once, from Hadoop
- Uses atomic replacement

Druid: Stream ingestion

Events →

```
{time: 1440000000000, user: alice, page: /foo, count: 2}
{time: 1440000000000, user: alice, page: /bar, count: 1}
{time: 1440000000000, user: bob,   page: /bar, count: 1}
```

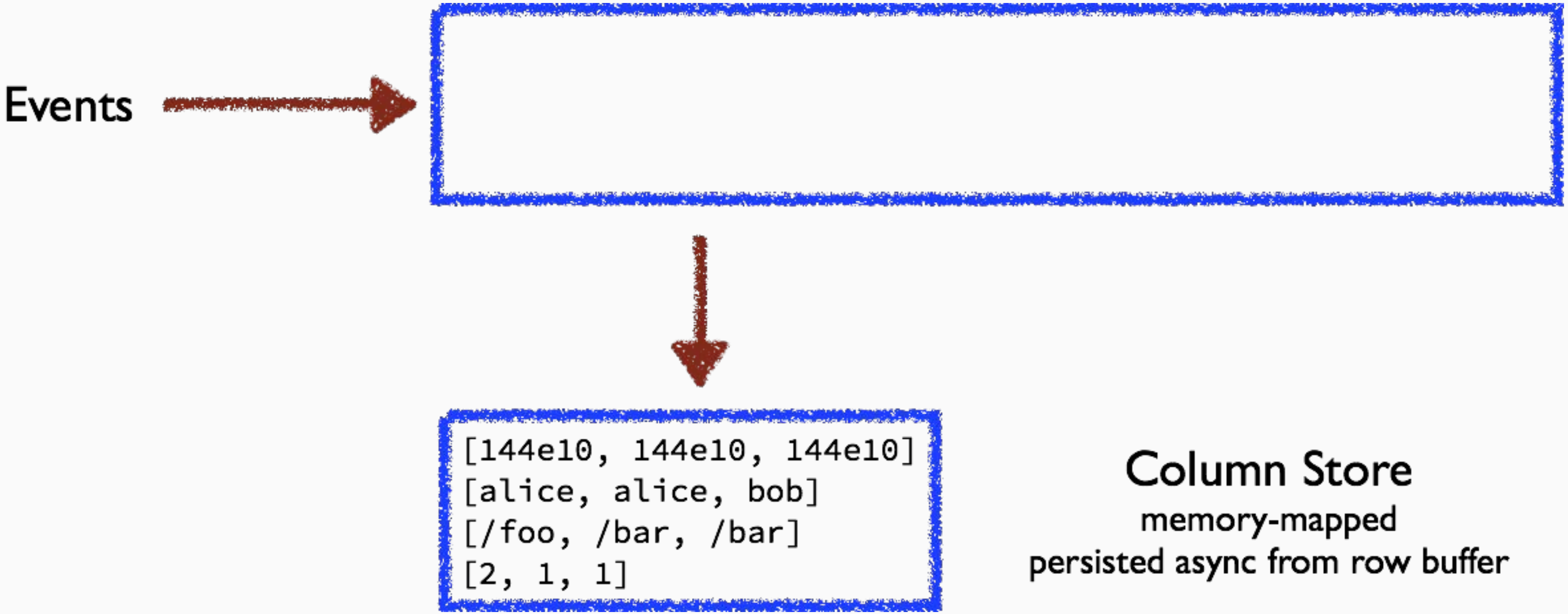
Row Buffer

in-memory

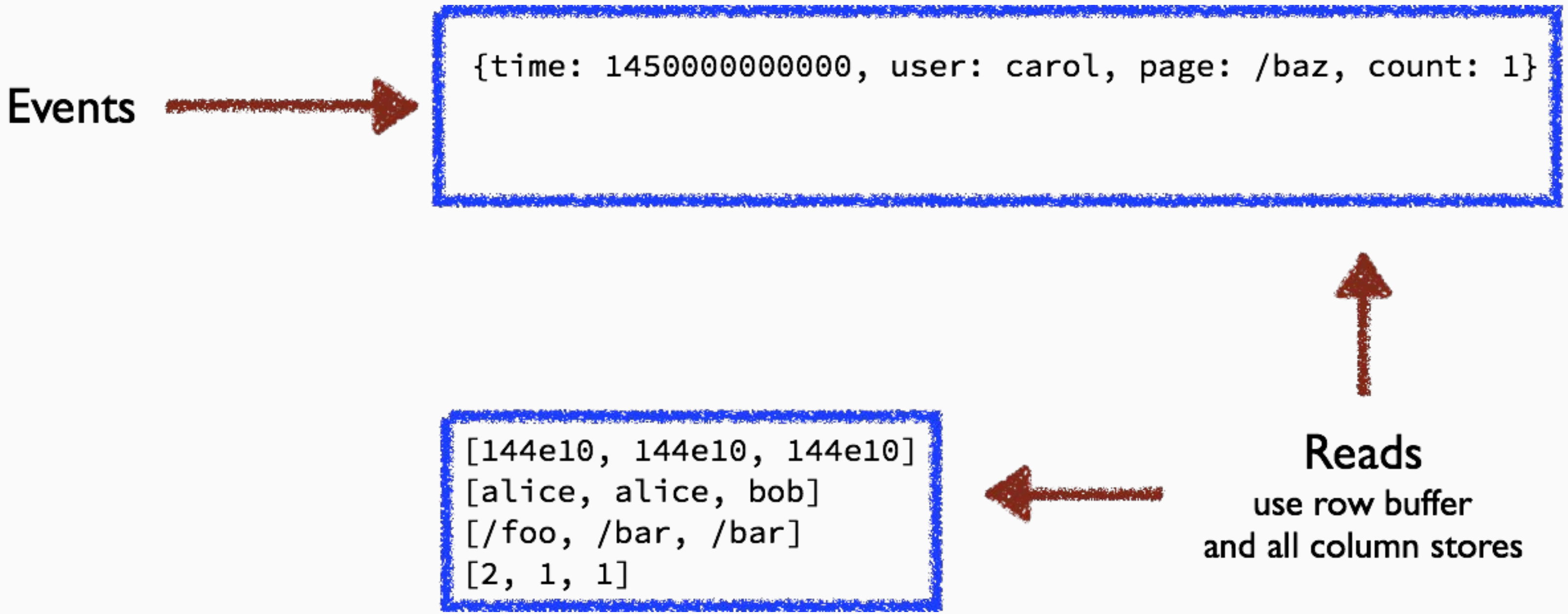
limited in size

grouped on dimensions

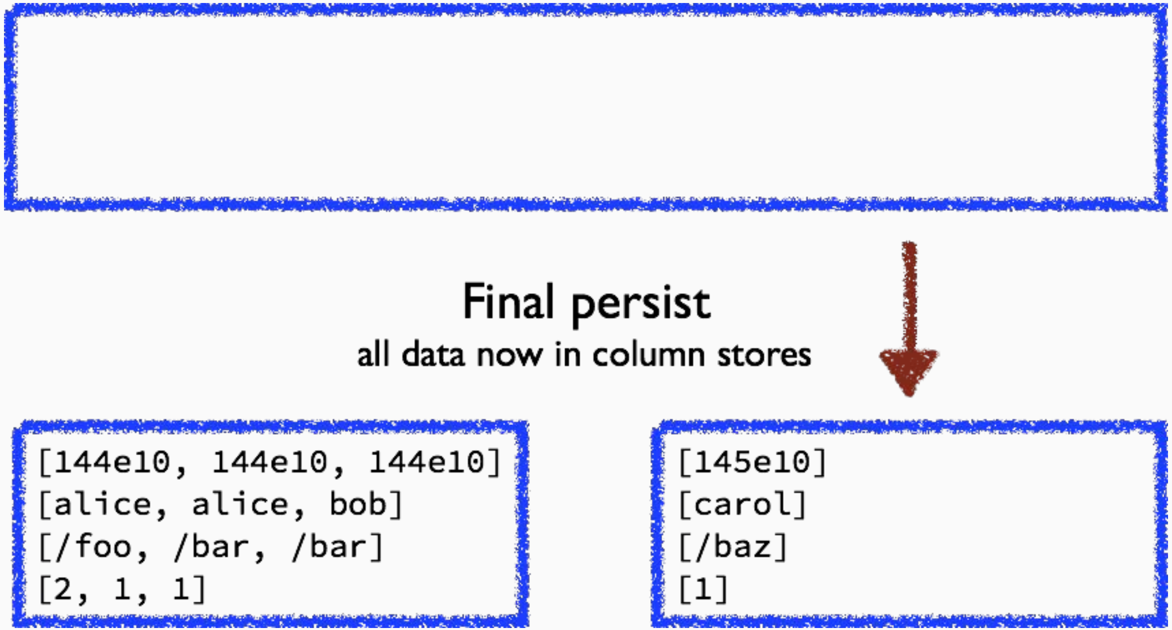
Druid: Stream ingestion



Druid: Stream ingestion



Druid: Stream ingestion



Druid: Stream ingestion

```
[144e10, 144e10, 144e10]
[alice, alice, bob]
[/foo, /bar, /bar]
[2, 1, 1]
```

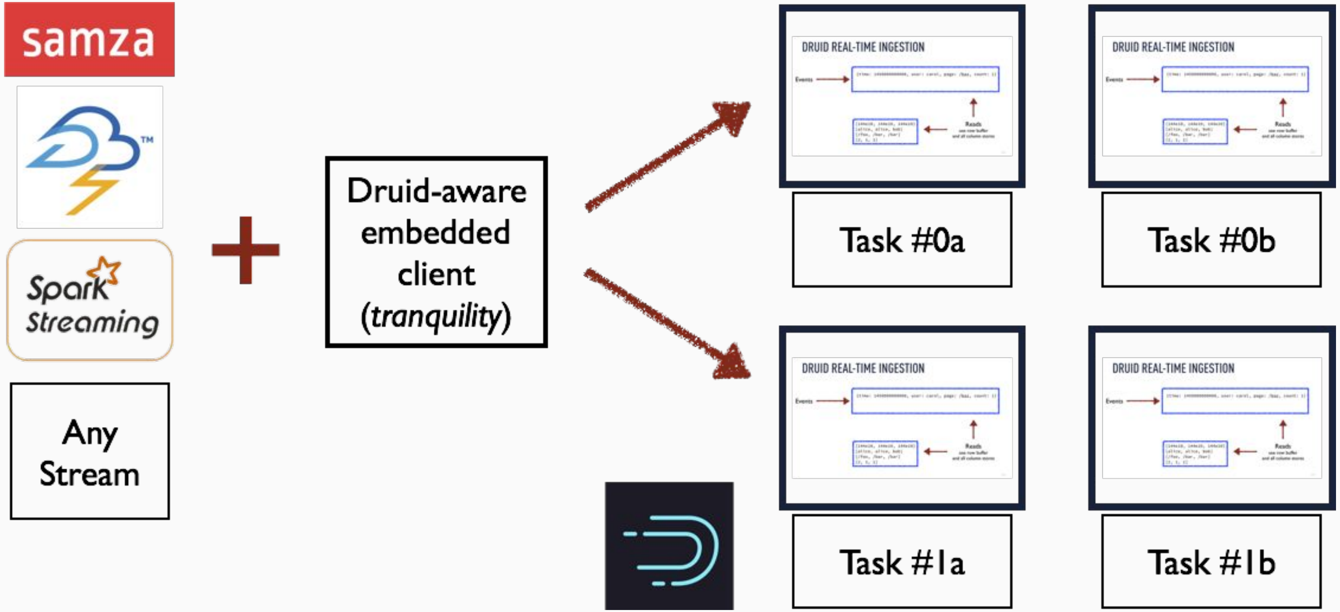
```
[145e10]
[carol]
[/baz]
[1]
```

Merge

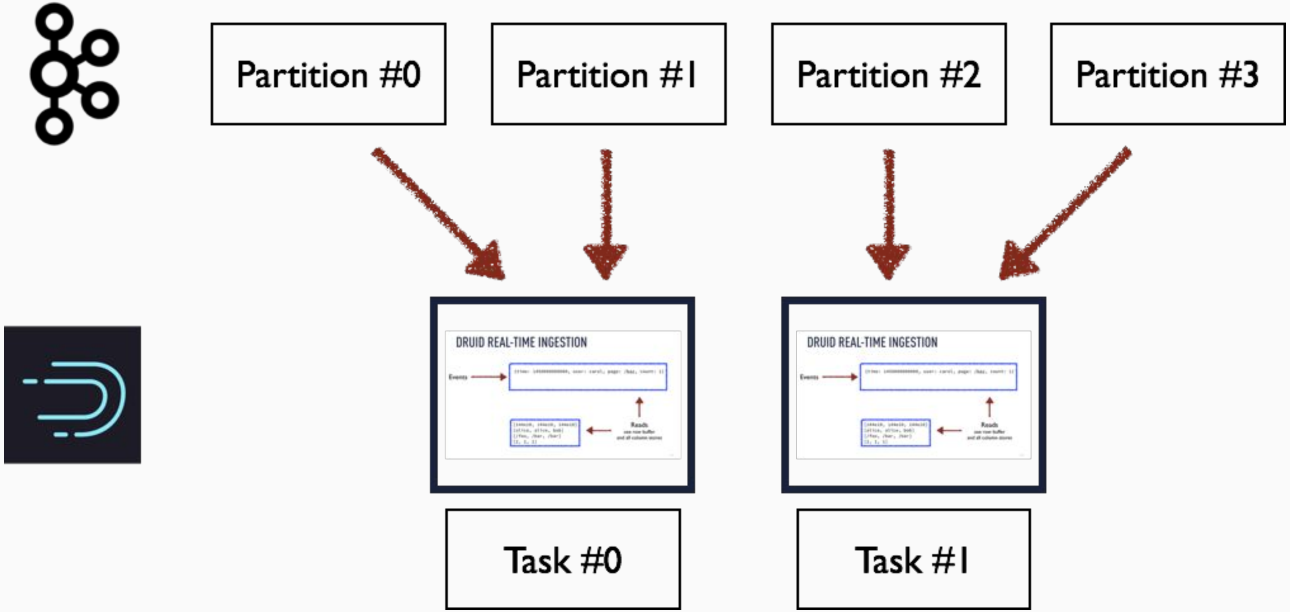
all data in a single segment
queried along with all existing data
target size 500MB–1GB

```
[144e10, 144e10, 144e10, 145e10]
[alice, alice, bob, carol]
[/foo, /bar, /bar, /baz]
[2, 1, 1, 1]
```

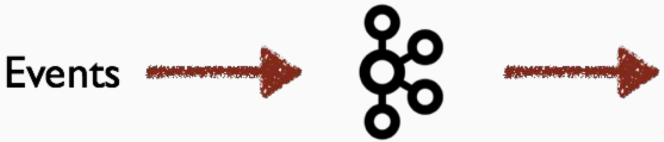
Druid: Stream push



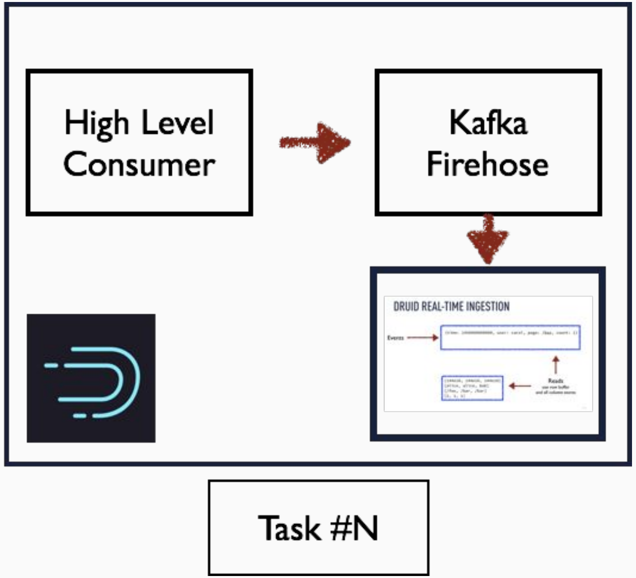
Druid: Kafka pull



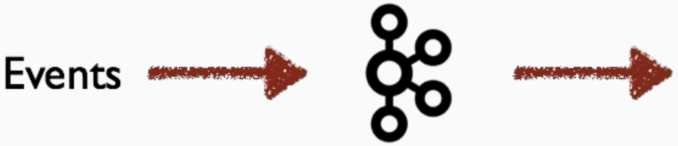
Druid: Kafka pull (current)



Kafka Firehose
uses Kafka high-level consumer
(+) commit offsets when persisted to disk
(+) easy to scale ingestion up and down
(-) not HA
(-) can generate duplicates during rebalances



Druid: Kafka pull (next-gen)



New Kafka Ingestion

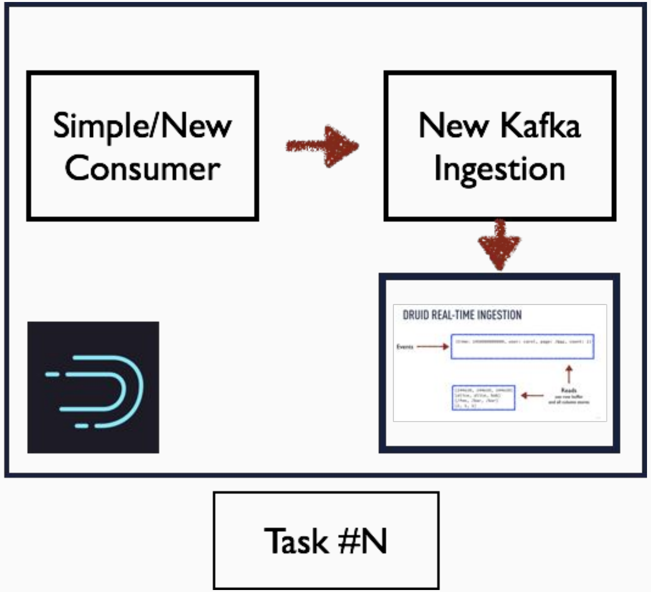
uses Kafka simple or new consumer

(+) store offsets along with Druid segments

(+) easy to scale ingestion up and down

(+) HA— control over who consumes what

(+) no duplicates during rebalances



Do try this at home

Software

- Druid - druid.io - @druidio
- Kafka - kafka.apache.org - @apachekafka
- Samza - samza.apache.org - @samzastream
- Storm - storm.apache.org - @stormprocessor

Take aways

- Databases and streams are best friends
- Consider latency and correctness in system design
- Know the guarantees provided by your tools
- Have a backfill strategy if you’re interested in historical data

Thanks!

@implydata
@druidio
@gianmerlino

imply.io
druid.io