Beyond the DSL - #process

Unlocking the power…
I'm here to make you PAPI! ;)
If you're PAPI and you know it, merge your streams!

antony@confluent.io

Kafka Streams DSL - the Easy Path

DSL - but eventually...

Quick Scientific™ survey

Who in the audience uses Kafka in prod?

- Stream processing frameworks?
- Kafka Streams?
- PAPI?

Antony Stubbs - New Zealand Made

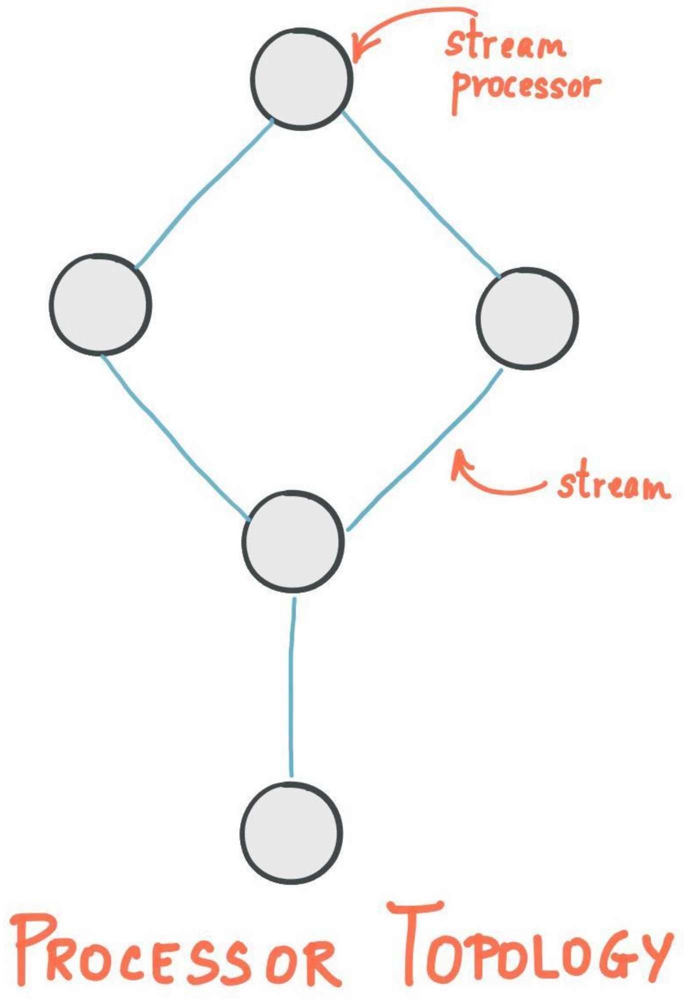
- @psynikal
- github.com/astubbs
- Confluent for ~2 years
- Consultant in EMEA
- Kafka Streams in my favourite

Agenda (eventually consistent)

A lot to get through...

- DSL intro
- #process?
- State stores
- Some brief user stories
- A lesson in punctuality
- Optimal Prime
- Secondary field querying on KS state
- Q&A - write down your questions...

Topologies, Trolls and Troglodytes



What is the DSL?

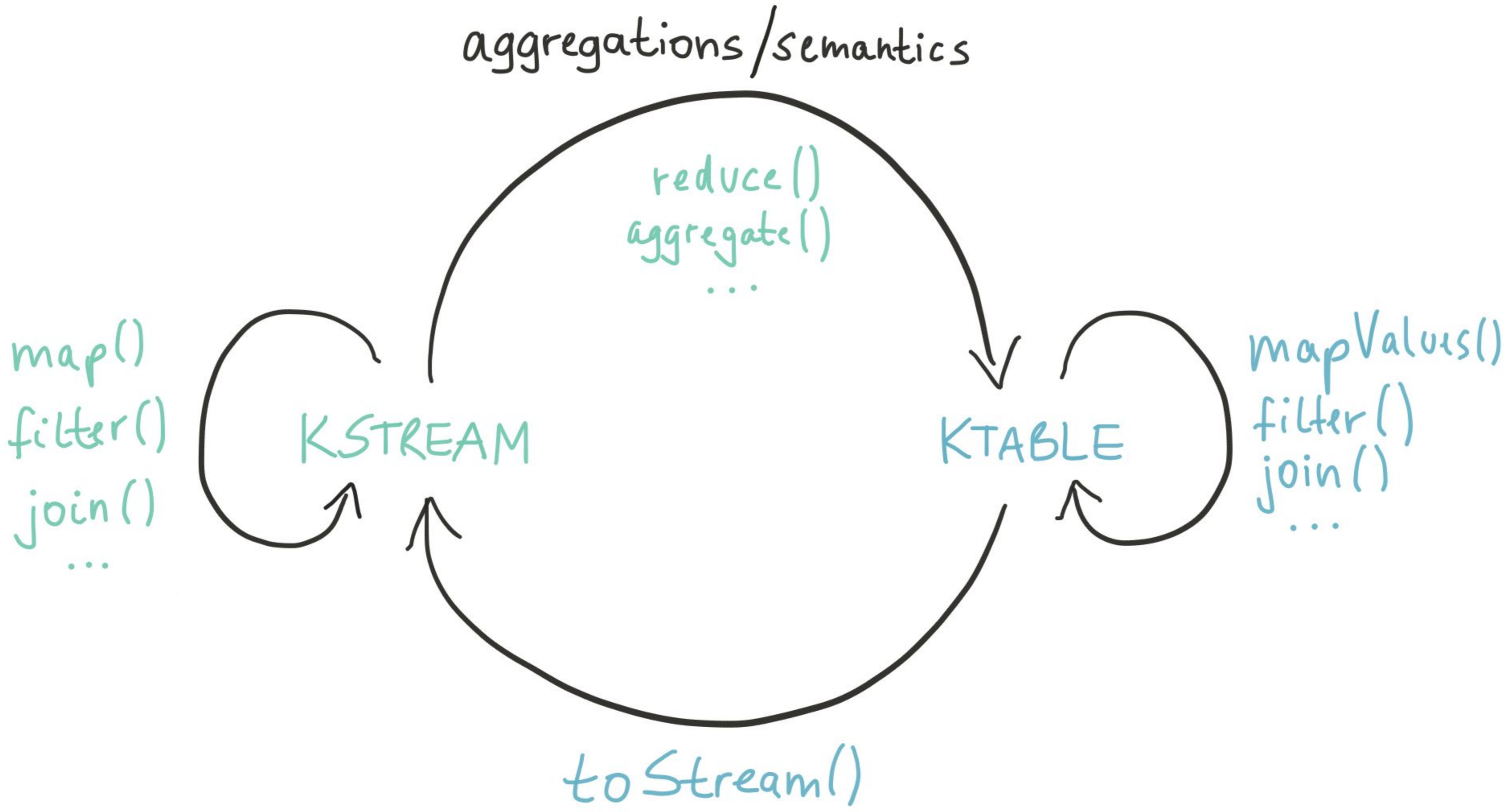
```
KStream<Integer, Integer> input =
    builder.stream("numbers-topic");

// Stateless computation
KStream<Integer, Integer> doubled =
    input.mapValues(v -> v * 2);

// Stateful computation
KTable<Integer, Integer> sumOfOdds = input
    .filter((k,v) -> v % 2 != 0)
    .selectKey((k, v) -> 1)
    .groupByKey()
    .reduce((v1, v2) -> v1 + v2, "sum-of-odds");
```

confluent

What is the DSL?



What is #process?

confluent

What is #process?

Freedom

confluent

What is #process?

Power

But with great power...
but not that much… :)

What is #process?

```
KStream#process({magic})
```

What is #process?

```
interface Processor<K, V> {

  void process(K key, V value)

}
```

What is #transform?

```
interface Transformer<K, V, R> {

  R transform(K key, V value)

}
```

What is #process?

```
interface Processor<K, V> {

  public void init(ProcessorContext context)

  void process(K key, V value)

  <... snip ...>
}
```

#process(Hello, World) - a bit more truth

```
KStream transform = inputStream.process({
  new Processor<Object, NewUserReq, Object>() {
    KeyValueStore state
    ProcessorContext context

    @Override
    void init(ProcessorContext context) {
      this.state = (KeyValueStore) context.getStateStore("used-emails-store")
      this.context = context
    }

    @Override
    Object Processor(Object key, NewUserReq value) {
      def get = state.get(key)
      if (get == null) {
        value.failed = false
        state.put(key)
      } else {
        value.failed = true
      }
      return KeyValue.pair(key, value)
    }

    @Override
    Object punctuate(long timestamp) {
      return null
    }

    @Override
    void close() {

    }
  }
})
```

PAPI vs DSL

When should you use which?

- “It depends”
- DSL
  - Easy
  - Can do a lot with the blocks
    - Clever with data structures
  - If it fits nicely, use it

- PAPI can be more advanced, but also super flexible
  - Build reusable processors for your company
  - Doesn’t have the “nice“ utility functions like count - but that’s the point
  - Can ”shoot your own foot”
  - Be responsible

- Don’t bend over backwards to fit your model to the DSL

A Combination - the Best of Both Worlds

```
ppvStream.groupByKey().reduce({ newState, ktableEntry ->
        <.... reduce function …>

}).toStream().transform({
    new Transformer<>((){
        <.... do something complicated…>

    }
}).mapValues({ v ->
    <.... map function …>
}).to("output-deltas-v2")
```

What is a State Store? IQ?

A local database - RocksDB

- K/V Store
- High speed
- Spills to disk

By nature of Kafka

- Distributed
- Redundant
- Backed onto Kafka (optionally)
- Highly available (Kafka + Standby tasks)

An optimisation?

- Moves the state to the processing

What are Interactive Queries (IQ)?

Working With Processors and State Stores

Simple:

- Deduplication (vs EOS)
- Secondary indices
- Need to do something periodically
- TTL Caches
- Synchronous state checking

Advanced:

- State recalculation
- Expiring data efficiently
- Global KTable triggered joins (DSL work around)
- Probabilistic counting with custom store implementations...

Globally Unique Email

KS microservice user registration

Need to make sure requested email is globally unique before accepting

DSL mechanism

- Construct a KTable (or global for optimisation) from used emails
- IQ against the KTable state to see if email is available
- However KTable state is asynchronous
  - May not populate before a duplicate request is processed (sub topologies <intermediate topic boundaries>, multi threading…)

PAPI mechanism

- Save used emails in a state store
- Remote IQ against the state store initially for async form UI validation
- Synchronously check the state store for used emails before emitting to the account created topic on command

But routing..?

State Subselects - Compound Keys

State stores have the #put, #get and *#range* method call - this brings some new magic...

Order Items

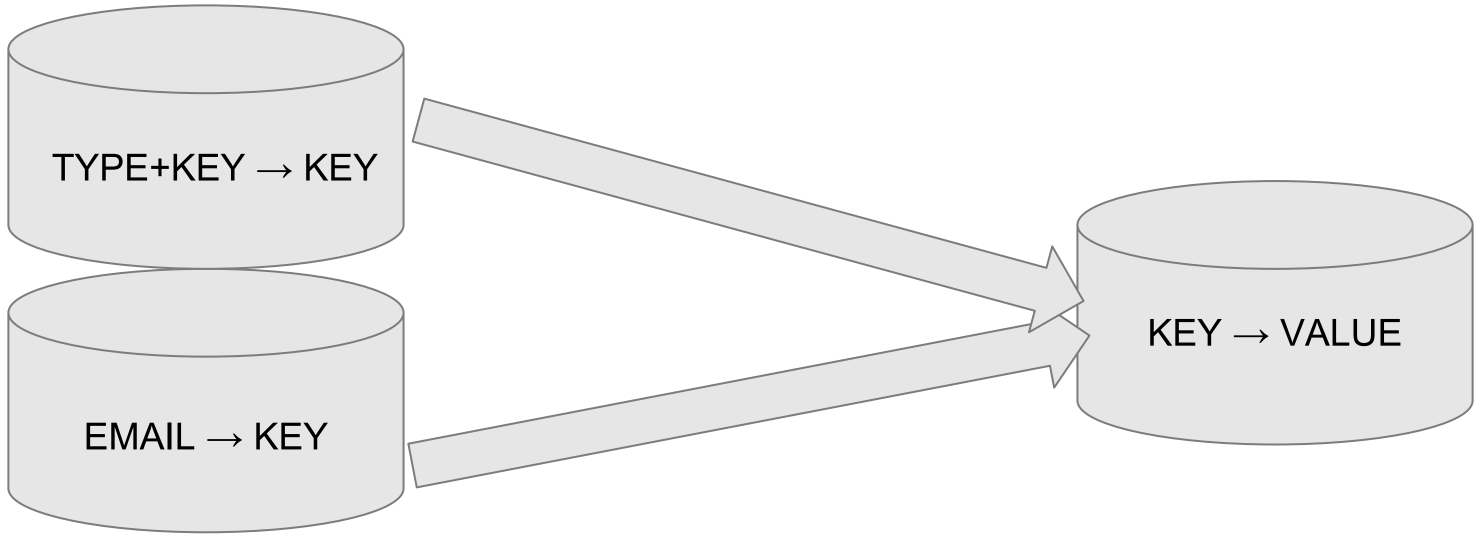
- Select all orders items from my state store, for this order key
- Avoids building larger and larger compounded *values*
    - Which will take longer and longer to update - using individual entries instead

Time

- Timestamp compounded with key
- Great for retrieving entries within computable time windows (hint hint)
- Great for scanning over a range of entries

State Subselects - Secondary Indices

- Think ~“tables”...
- Serving an “Interactive Query” for a few possible fields...
    - Can’t construct compound keys for multiple combinations
    - One State Store per combination
    - Upon inserting the primary key/value
        - Also insert into the extra stores the field<->key mappings
        - Upon query, query against the appropriate store that holds the mappings for the requested field
        - Collect the possibly many values (keys) and retrieve the entire object(s) from the primary store

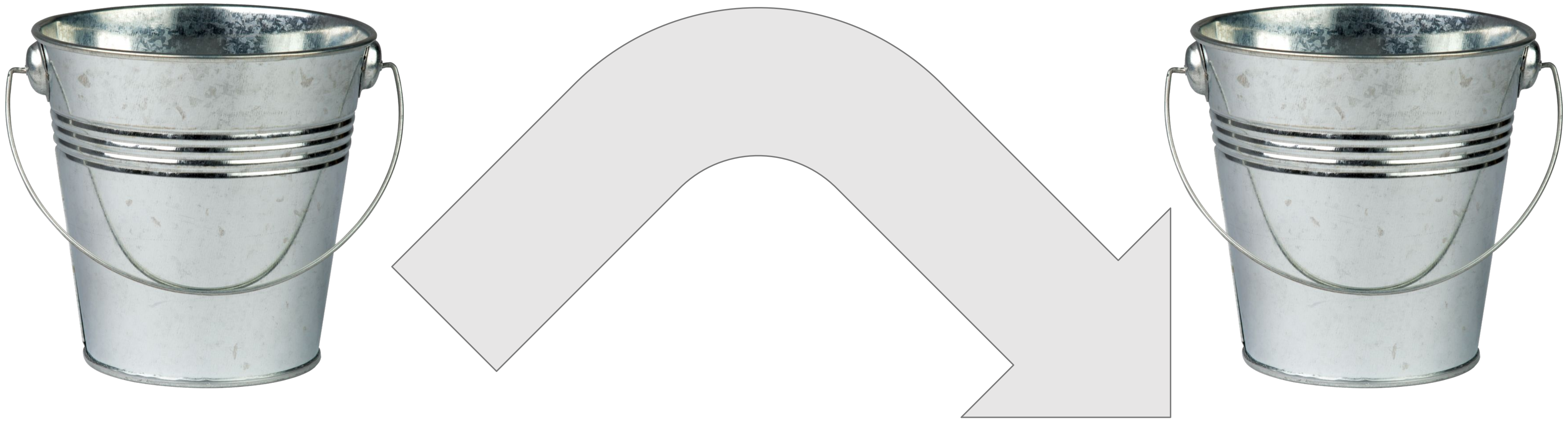


Window Recalculation

The use case is this:

- We have “devices” in the wild
- We want to compute aggregations based on the state of these devices
- Message from these devices arrive out of order
  - We can’t ensure ordering because there’s (a) proxy layer(s) out of our control
  - (No single writer, no orderIng guarantee)
- Messages arrive with state from the devices
- The state has some flags which are only present if they’ve changed

Case Study - leveraging state stores



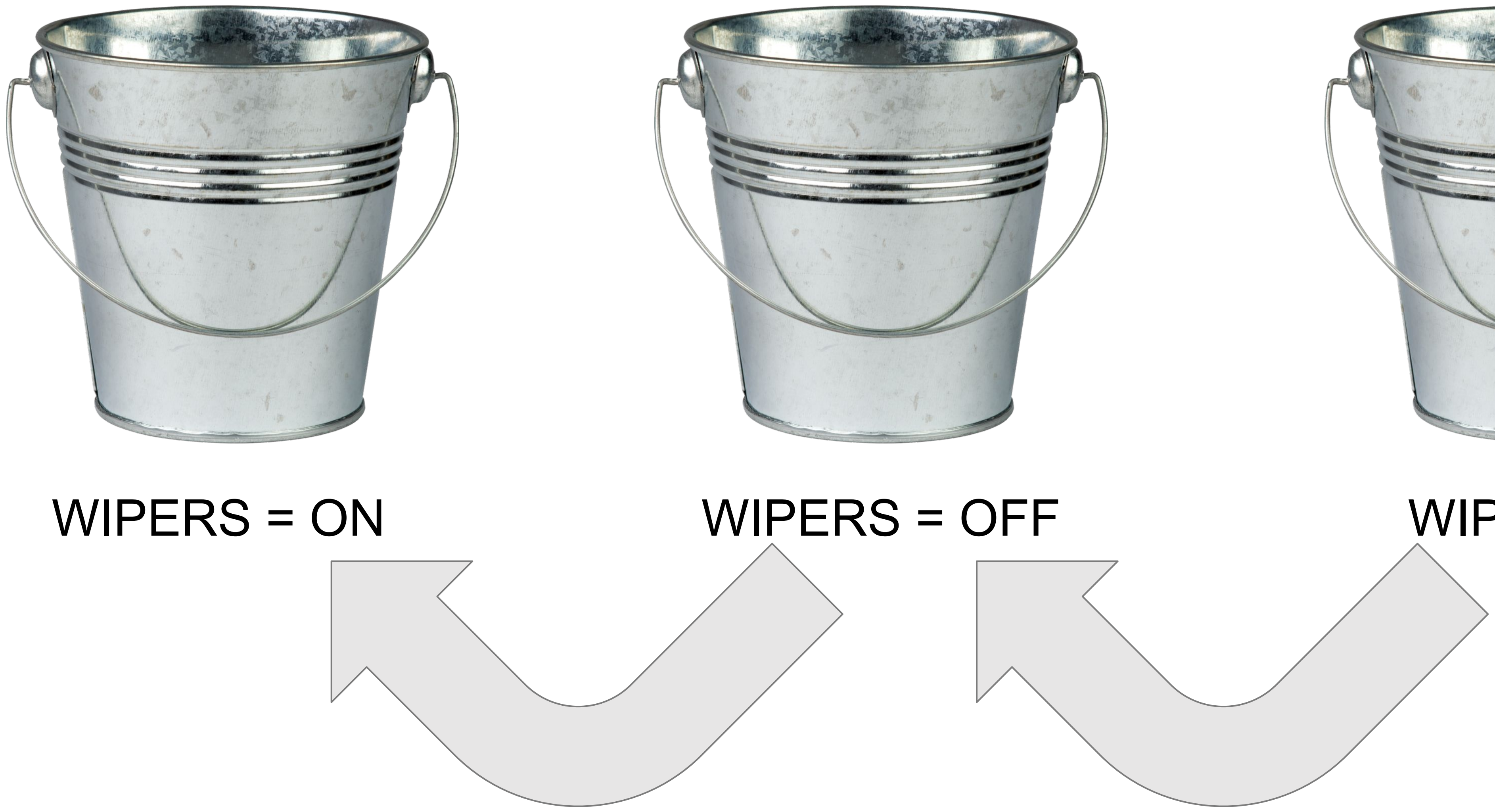
WIPERS = ON

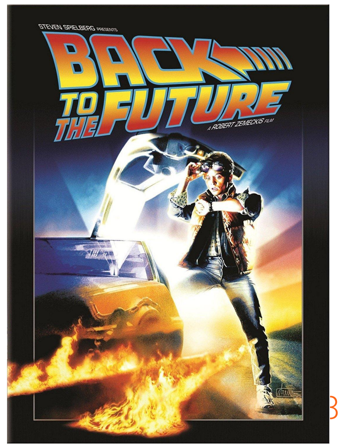
WIPERS = ?

So what's the problem you ask?

- What happens when the late message with the missing state arrives?
- Need to go back and update a past aggregate from data that landed in a new aggregate

Case Study - leveraging state stores





Dynamic,

aggregation

dependent,

recalculation,

with -

out of order arrival.

Dynamic,

dependent,

aggregate,

on demand recalculation,

from -

out of order data.

So, need to go back and recalculate...

DSL despair!

Can’t update aggregates outside of the aggregate that has been triggered…

Potential messy DSL solution (bend over backwards) - synthetic events!

- Publish an event back to the topic with the correct time stamp and information needed to retrigger the other aggregates
- You need to calculate / all / the correct time stamps
- Pollutes the stream with fake data
- Is unnatural / smells
- Breaks out of the KS transaction (EOS)

Enter #process()

Keep track of our aggregates ourselves

- Need to calculate our own time buckets
- Time query or store for possible future buckets
- All kept with in the KS context (no producer break out for synthetic events)

Case Study - leveraging state stores

Punc’tuation.?

- What are Punctuations?
- What is Wall Time?
- What is Event Time?

#process window retention period you may ask?

- DSL has window retention periods
  - We need state - but for how long?
  - KTable TTL? (Bounded vs unbounded keyset)

- #process TTL?
  - Using punctuation - scan periodically through the state store and delete all buckets that are beyond our retention period
  - Do TTL on “KTable” type data

- How? Compound keys...

Future Event Triggers

Expiring credit cards in real time or some other future timed action

- don’t want to poll all entries and check action times
- need to be able to expire tasks

Time as secondary index (either compound key or secondary state store)

- range select on all keys with time value before now
- take action
- emit action taken event (context.forward to specific node or emit)
- delete entry
- poll state store with range select every ~second,
- or schedule next punctuator to run at timestamp of next event
    - need to update

Database Changelog Derivation

Problem:

- DB CDC doesn’t emit deltas, only full state
- Can’t see what’s changed in the document

Solution:

- Derive deltas from full state, stored in a stateful stream processor
- Can use KTable tuples

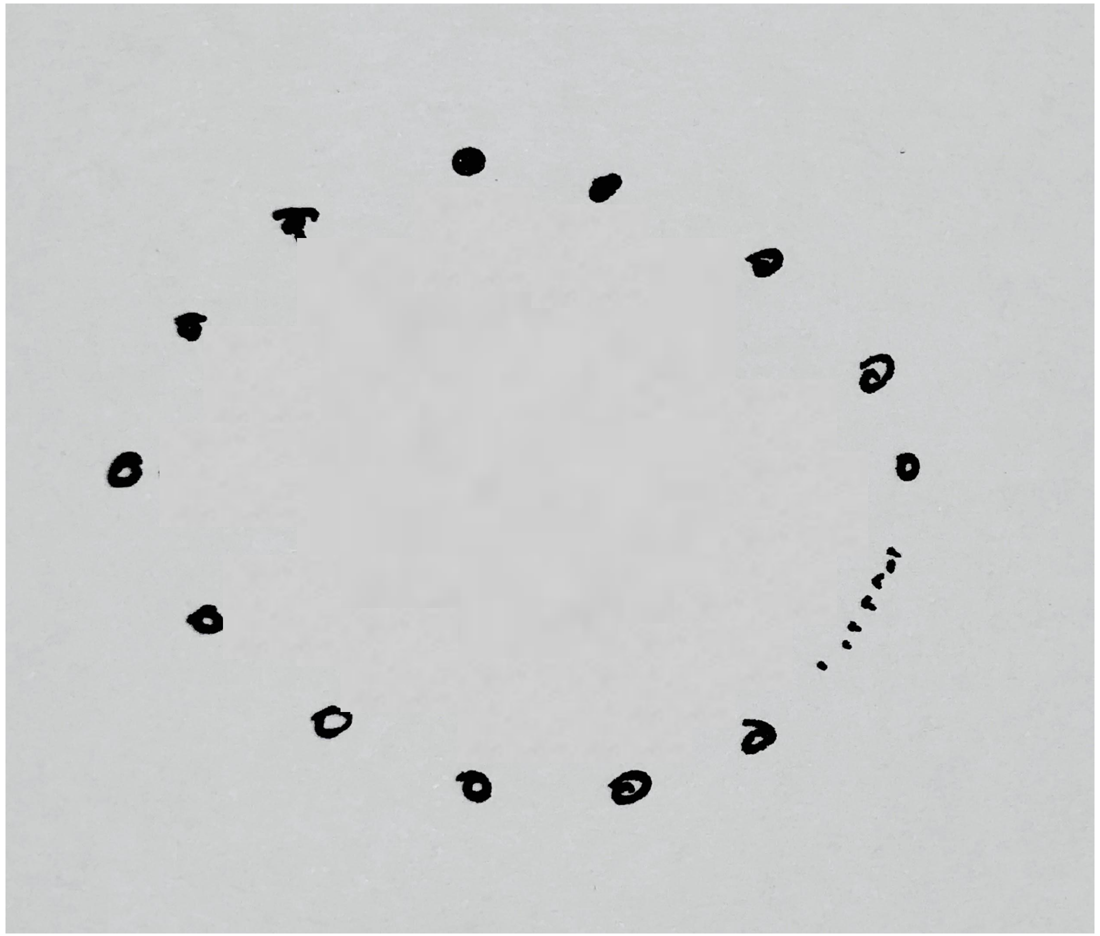
Issue:

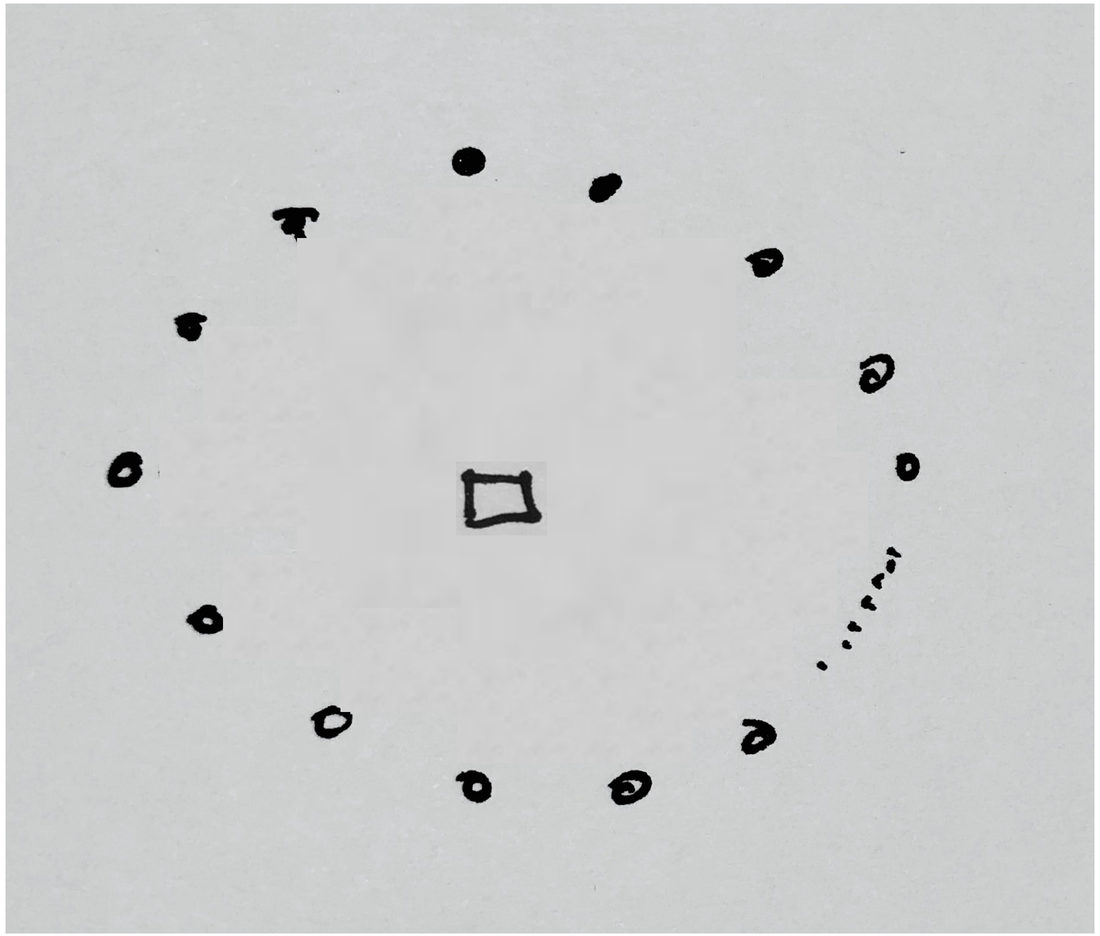
- No TTL - enter PAPI

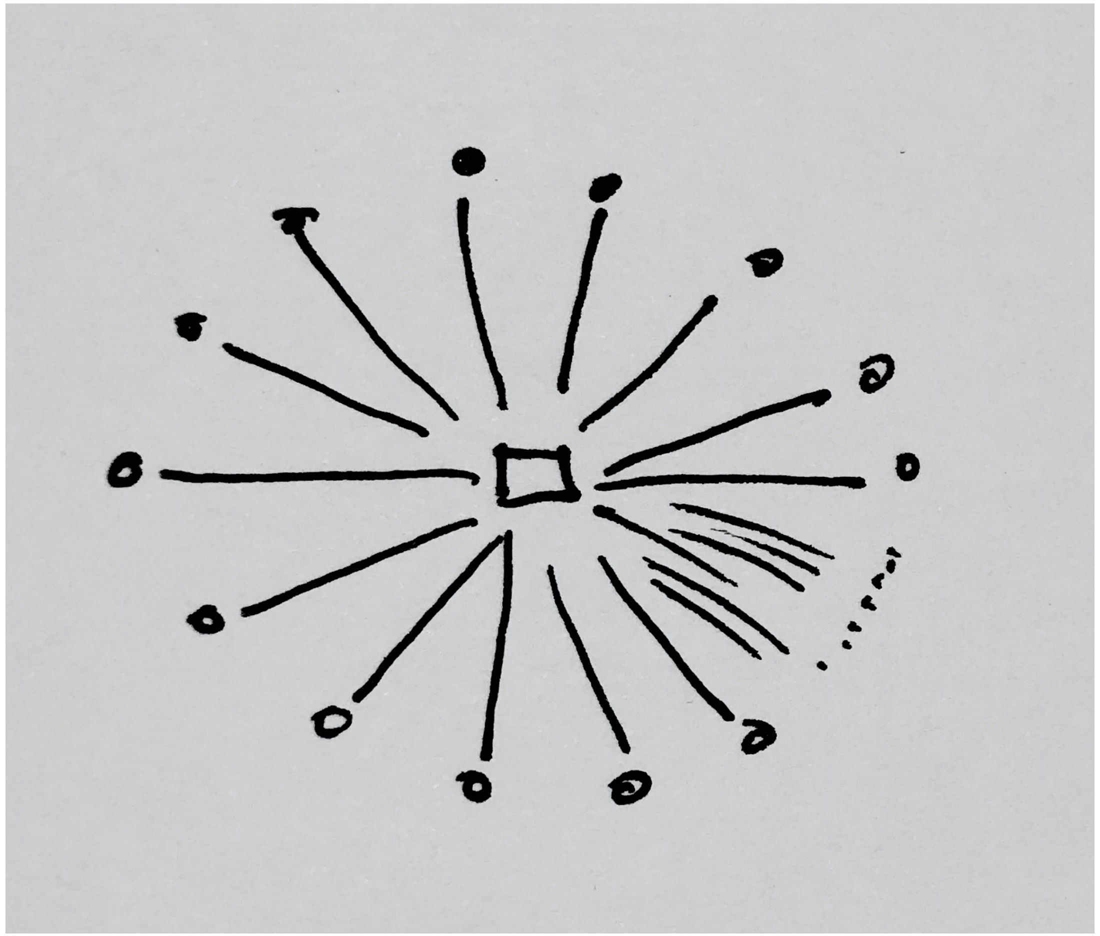
Distributed one to MANY MANY MANY late (maybe) joins

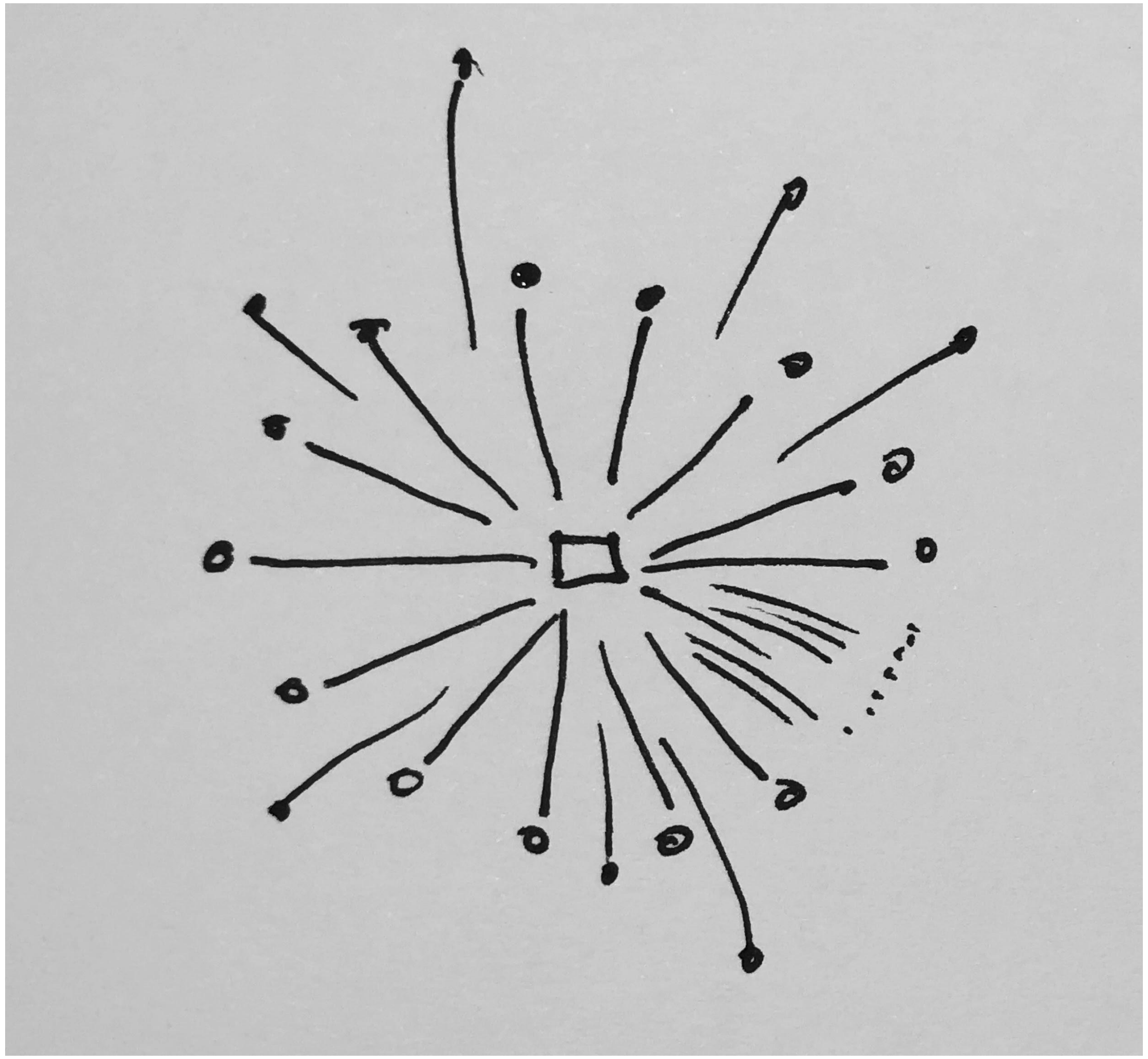
What’s the problem?

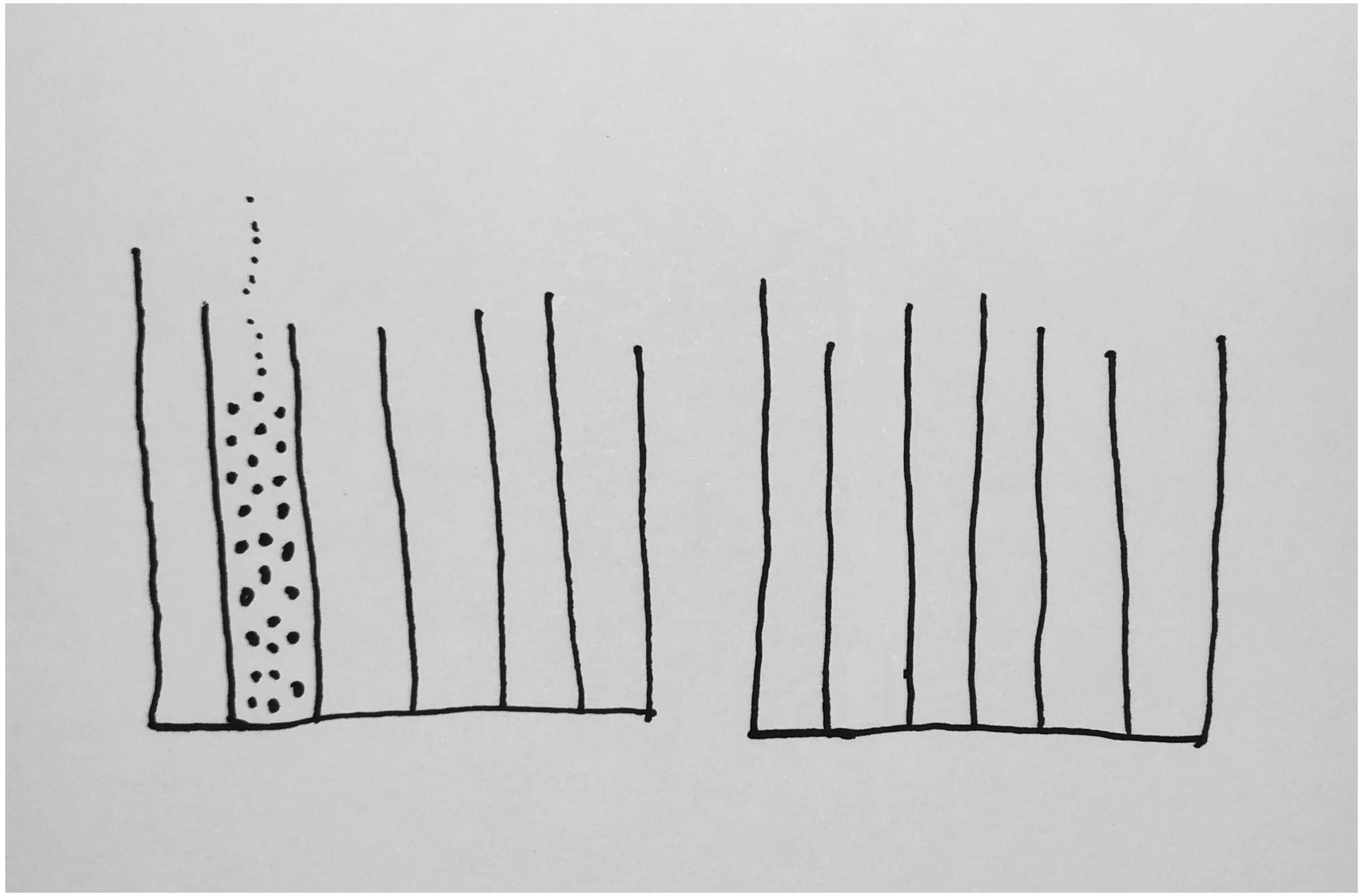
- Effectively re-joining missed joins once the right hand side arrives

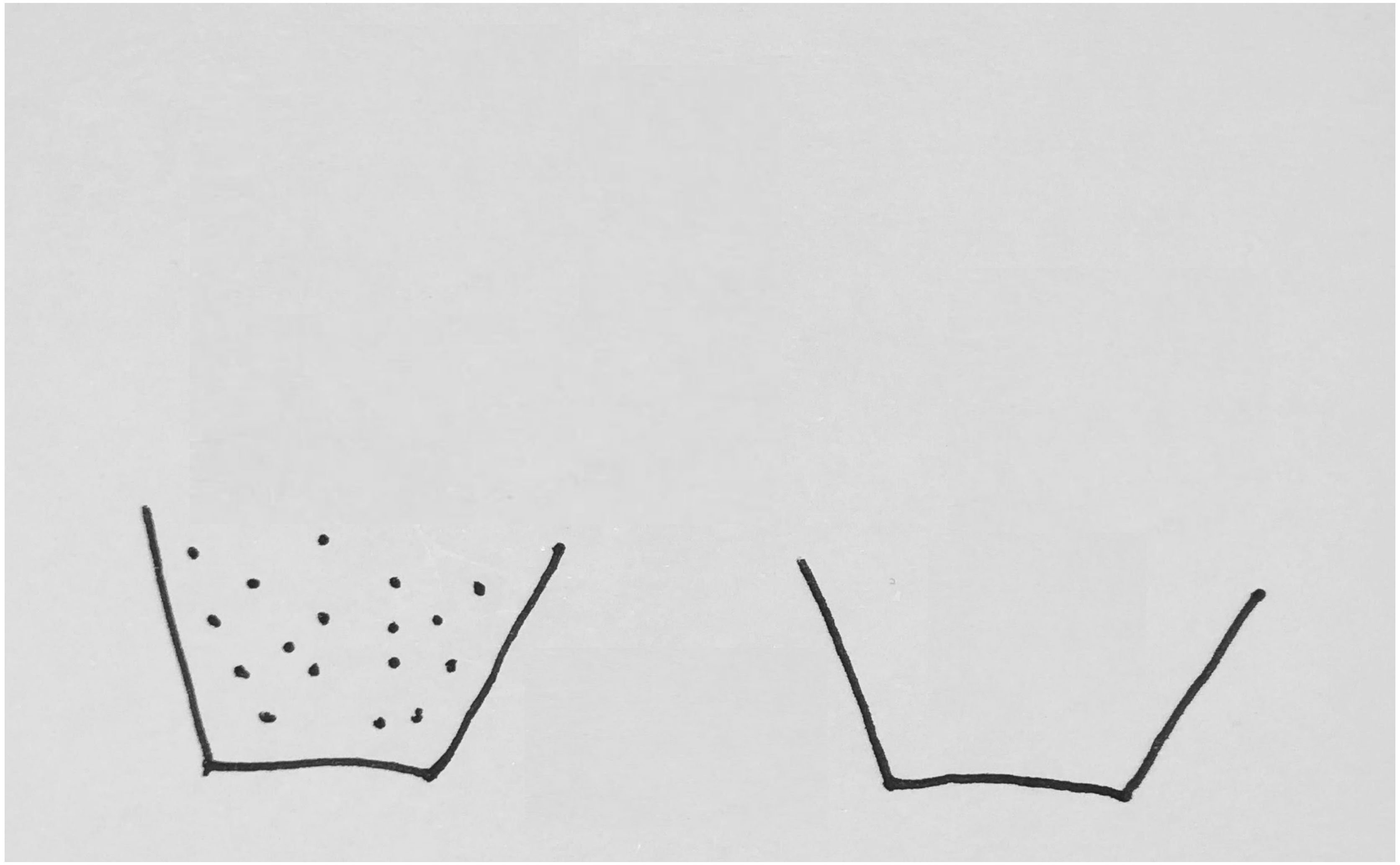


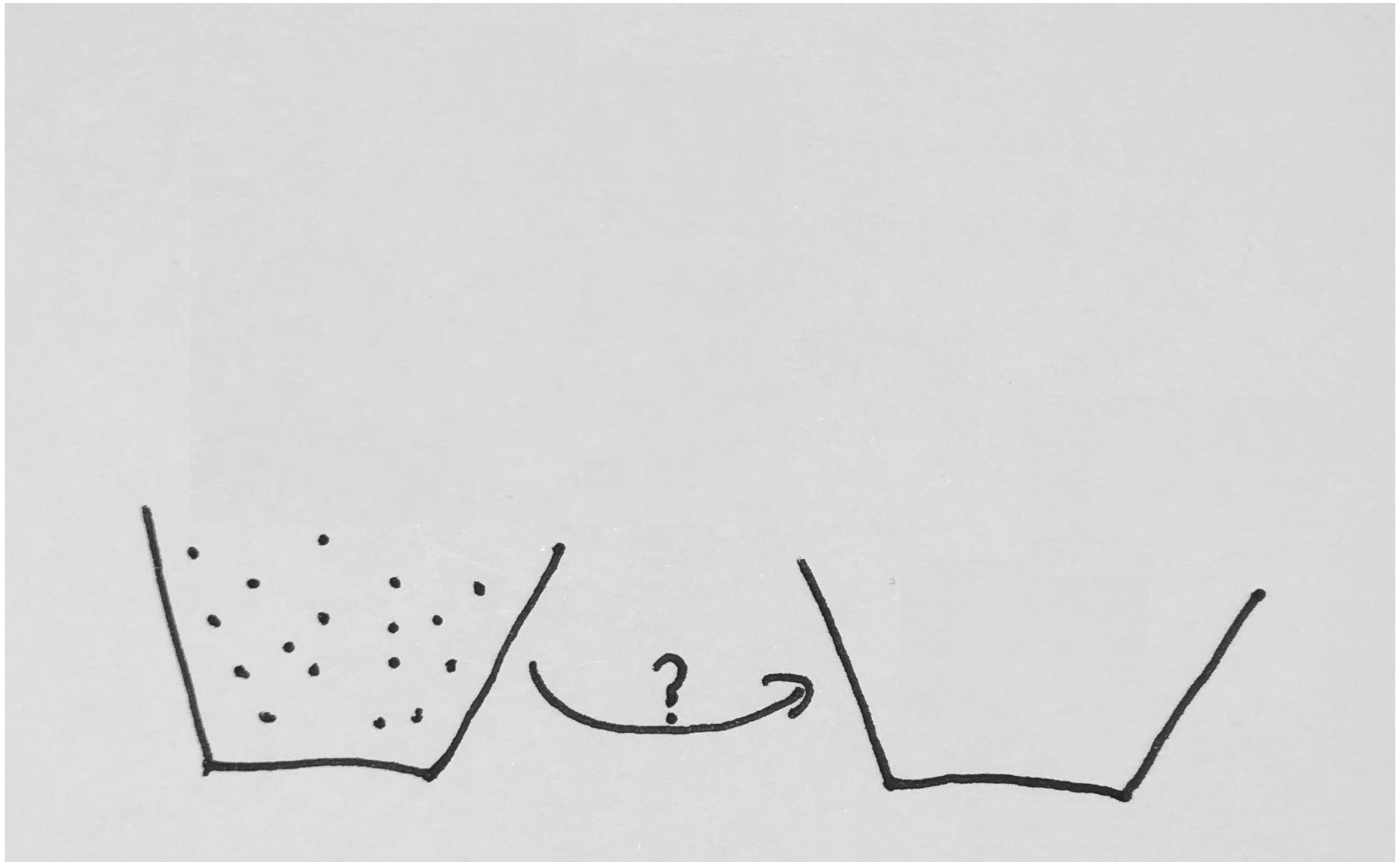


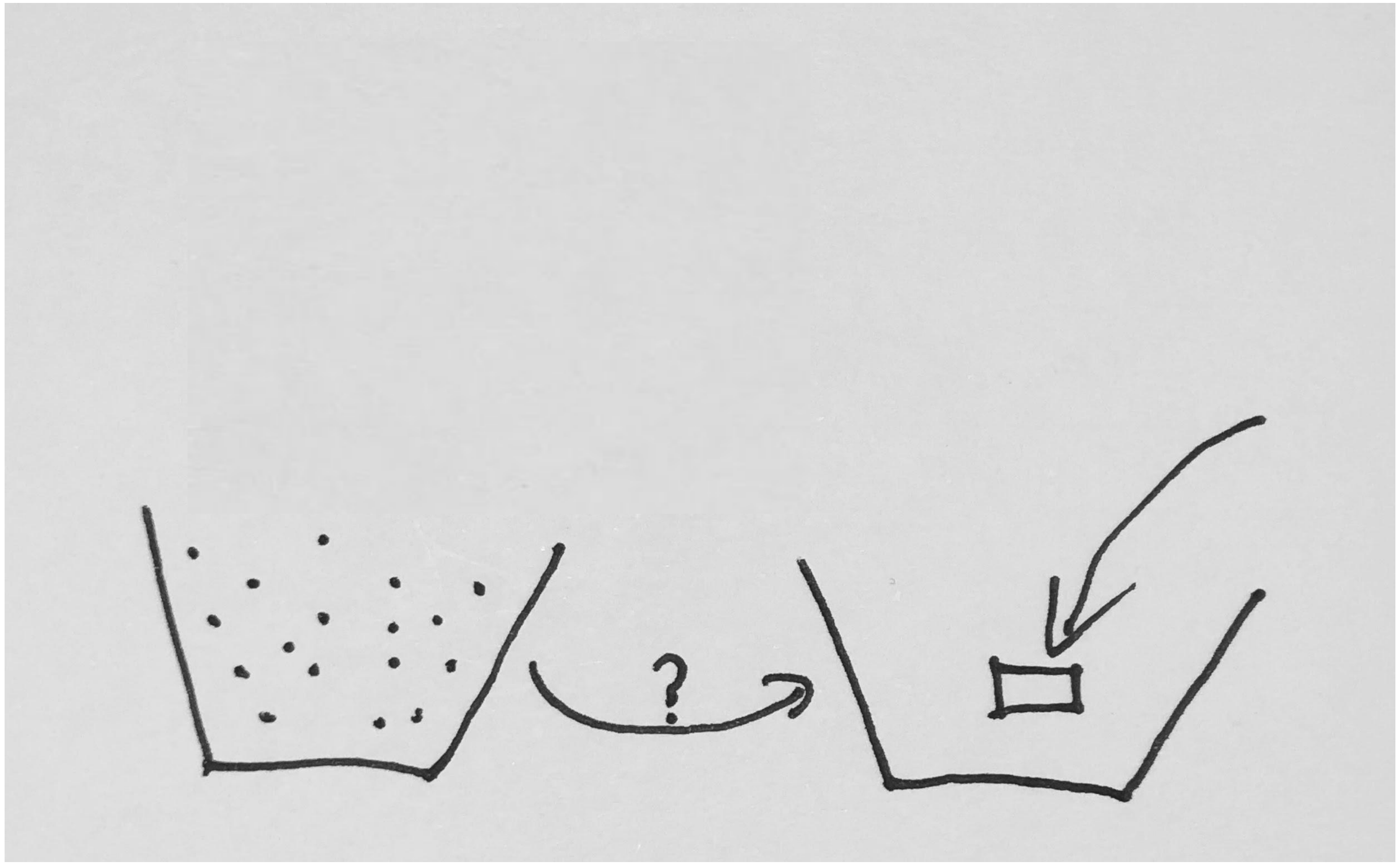












Probabilistic Data Structures

Probabilistic data structures such as Count-min sketches to perform e.g. counting at scale (https://github.com/confluentinc/kafka-streams-examples/blob/4.0.0-post/src/test/scala/io/confluent/examples/streams/ProbabilisticCountingScalaIntegrationTest.scala

Custom State Stores and Fault Tolerance

The previous example (probabilistic counting) also demonstrates how to implement custom state stores that are fault-tolerant etc.

Kafka vs doc store as source of truth

Doc store wasn’t good event source

Difficult to sync other systems off something that’s not a real / exposed event log

Bending over backwards to recreate lost data from writing to BD first.

Dual write issues in several places

Difficult retry mechanism with local caching

Trying to resolve out of order issues due to not using kafka log as truth

Optimising Topologies

Sometimes it’s useful to avoid some DSL overheads

- Combine operators
- Avoid repartitioning in some cases
- etc...

Beware inconsistent hashing...

Speaking of topology optimisation...

Global topology optimisation coming in 2.1

Two phase topology building

- First optimisation is reusing intermediate rekey topics
- Avoids branched on demand rekey further down the DAG by detecting the rekey and moving it up immediately
    - manually achievable by forcing rekey straight away with #through

Now I did ask about KSQL after all...

Check out it KSQL if you haven't already…

- Abstraction over Kafka Streams
- Languages outside of the JVM
- Non programmers
- Among others...

KSQL User Defined Functions in CP 5.0!

- Parallels with Processors combined with the DSL, you can now insert more complex functionality into ksql
    - Eg trained machine learning model and use as UDF in KSQL

Where to next? & We're hiring!

Don't be afraid of #process and do drop down from the DSL for some operations!

- github.com/confluentinc/kafka-streams-examples
- "Ben Stopford" on youtube.com
- Kafka Streams playlist on confluentinc youtube

- Consulting services? Contact sales@confluent.io

Further reading

- confluent.io/resources/
- docs.confluent.io/current/streams/
- confluentinc on Youtube
- github.com/astubbs
- @psynikal

Join us! https://www.confluent.io/careers/

Come find me for Q&A later...

confluent