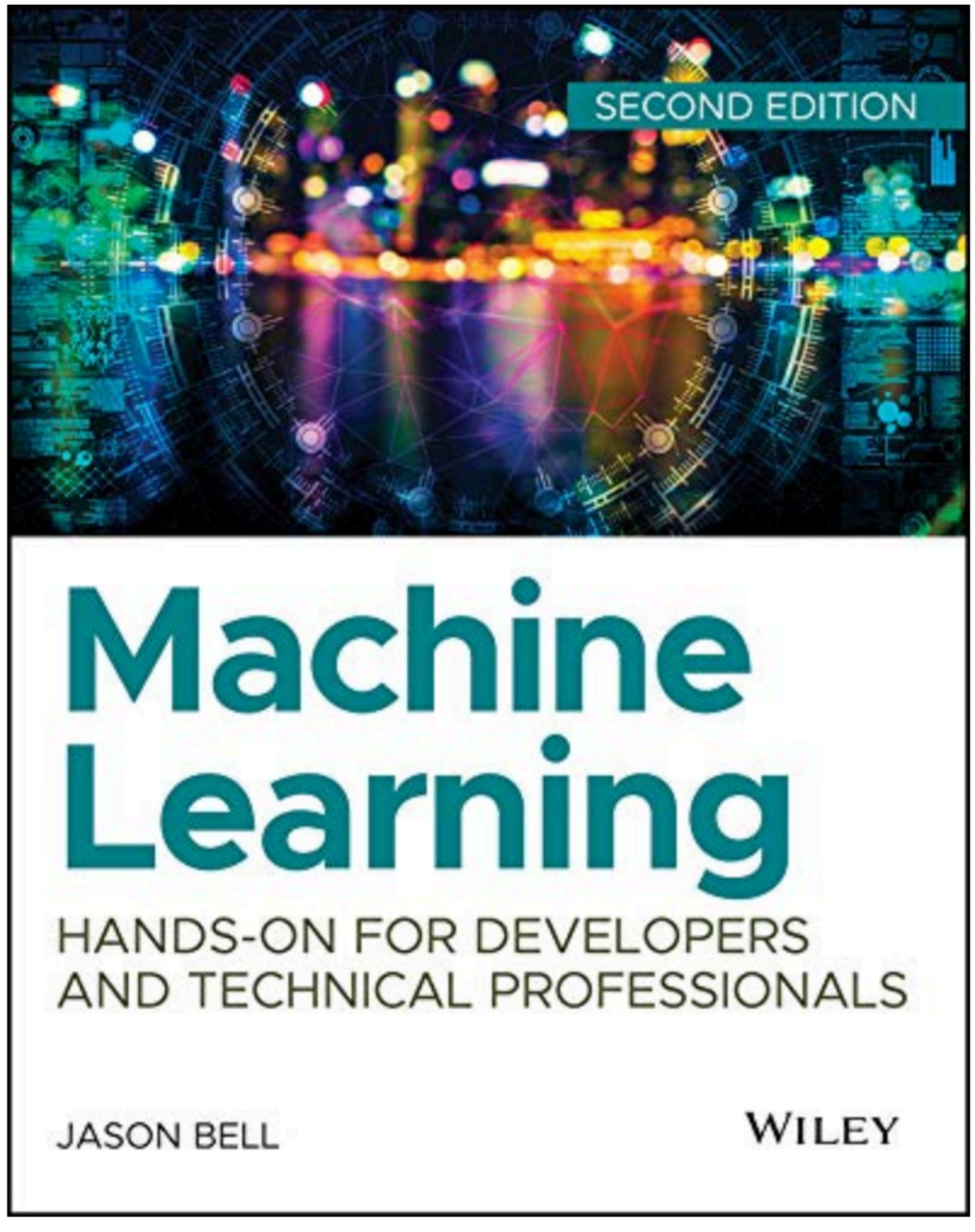
Hello Cleveland!

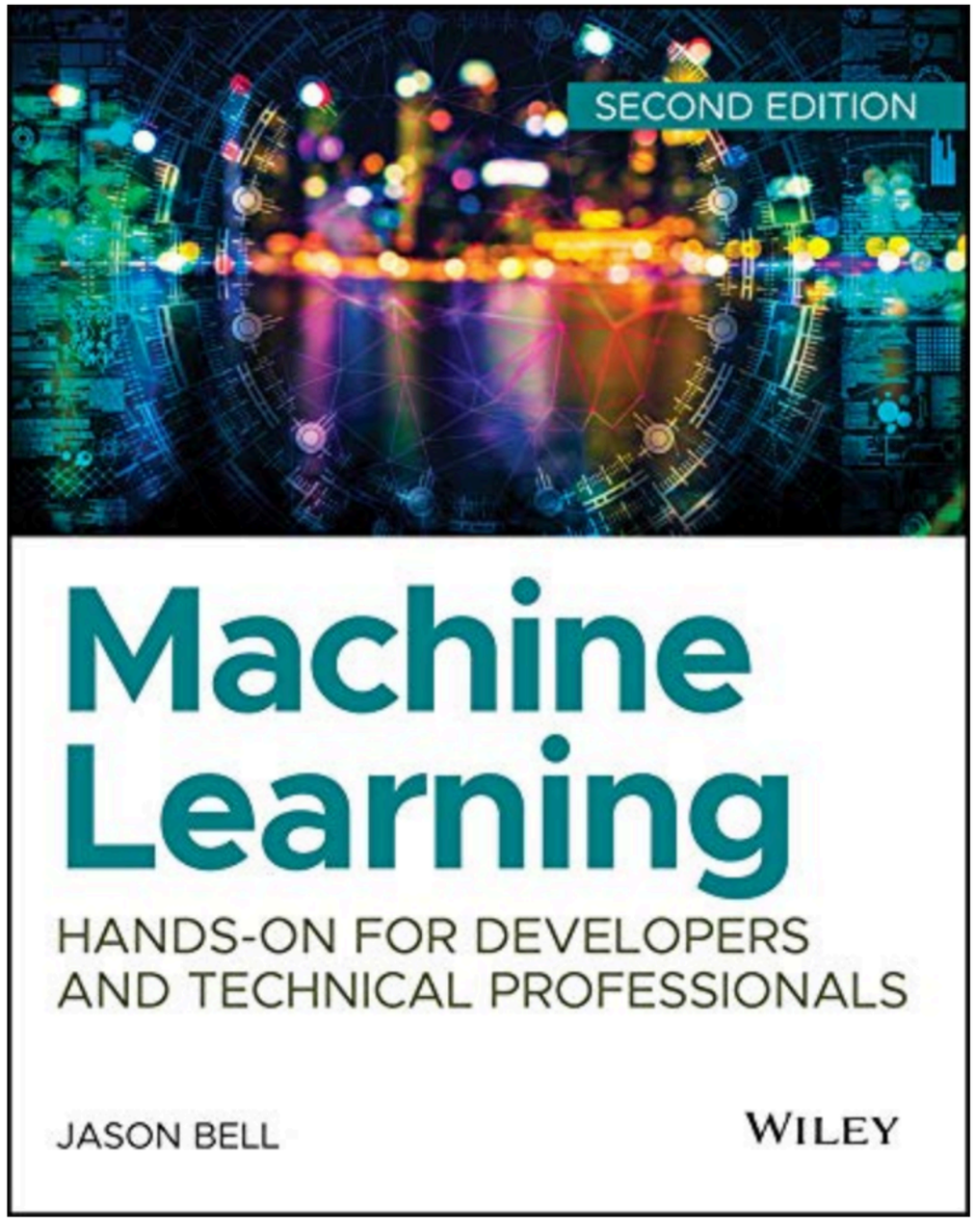
Work for

digitalis.io

Wrote this >>>



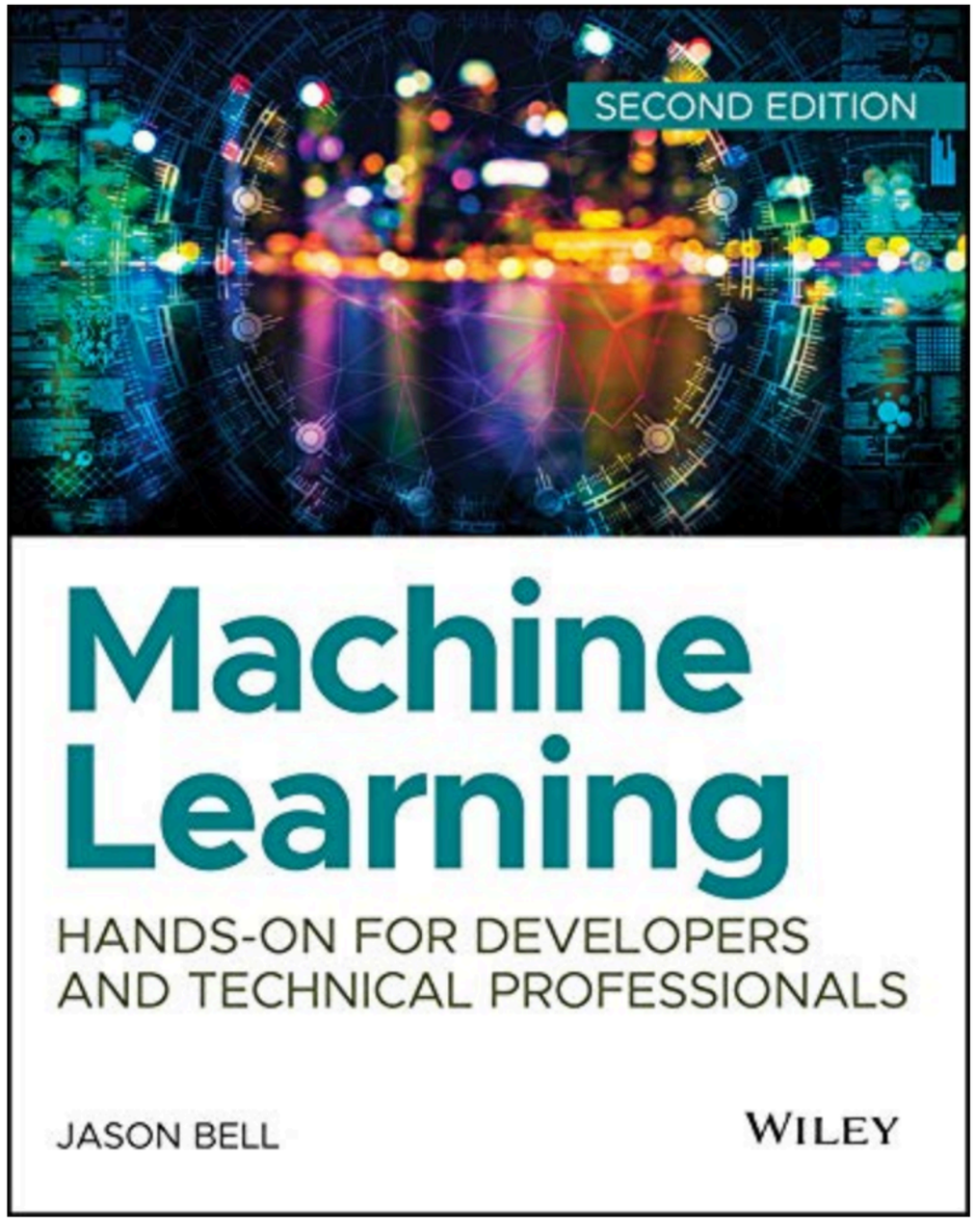
You can heckle on Twitter….
@jasonbelldata



The Plan

1. A quick overview of project and my original talk.
2. Look at how to make it better with Kafka's components.

Think of this talk as a conceptual how-to and thoughts about awkward data.



THIS IS A TRUE STORY.

The events depicted took place in London in 2016.

At the request of the client, the names have been changed.

Out of respect for the data,
the rest has been told exactly
as it occurred.

The Data

CSV Payloads are GZipped and sent to an endpoint via HTTP.

The data contains live flight searches and are batched together in chunks of a certain duration.

```
LHR,IST,2020-08-01,169.99,Y
```

There may be 3 rows or 20,000 rows.

It’s variable and we can’t control that.

I call this “The Fun Bit”

So this is what we did as a proof of concept.

And boy did I learn.

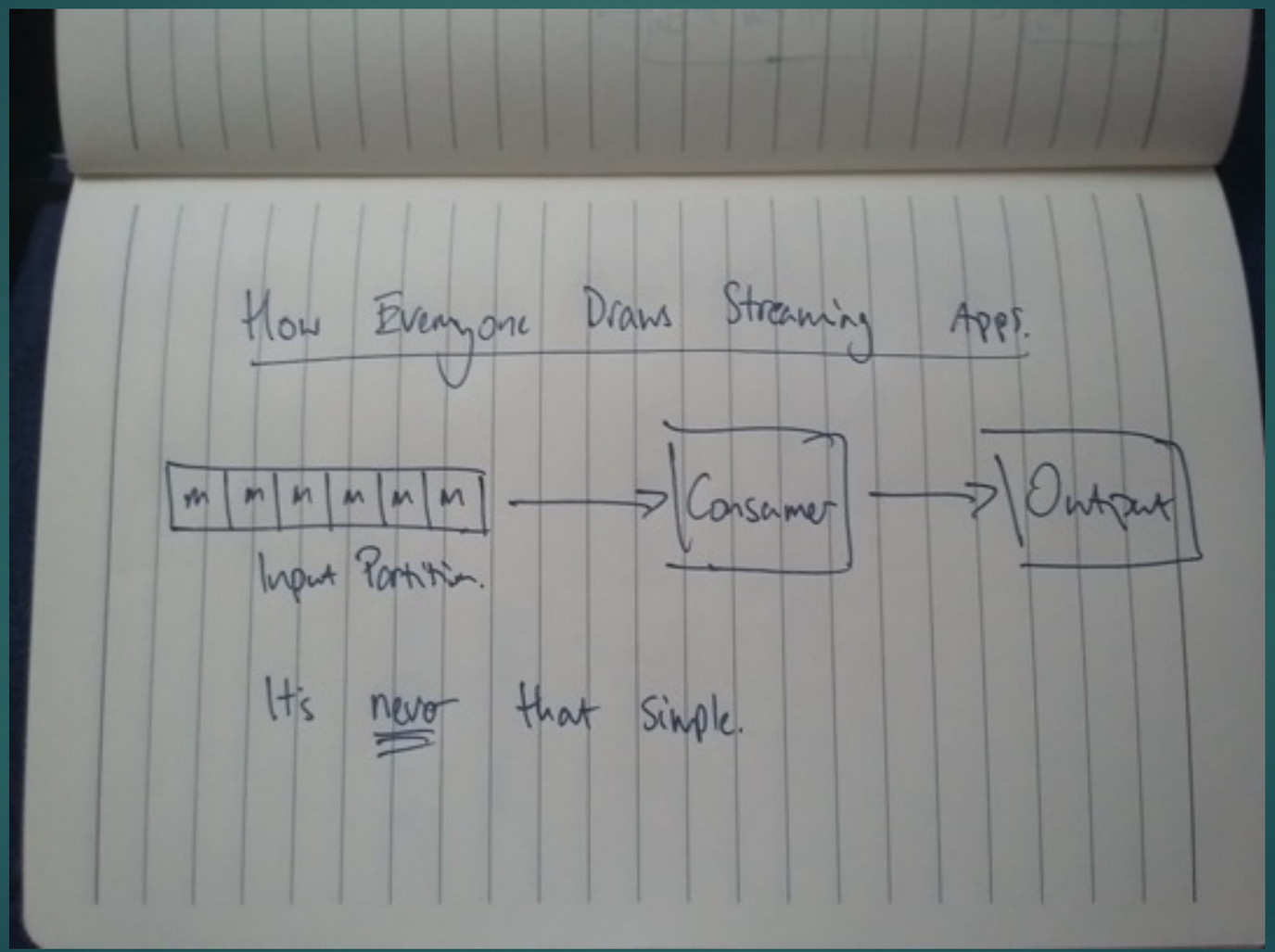
Thoughts, Experiences and Considerations with Throughput and Latency on High Volume Stream Processing Systems Using Cloud Based Infrastructures.

How I Bled All Over Onyx

One Day I Got a Question

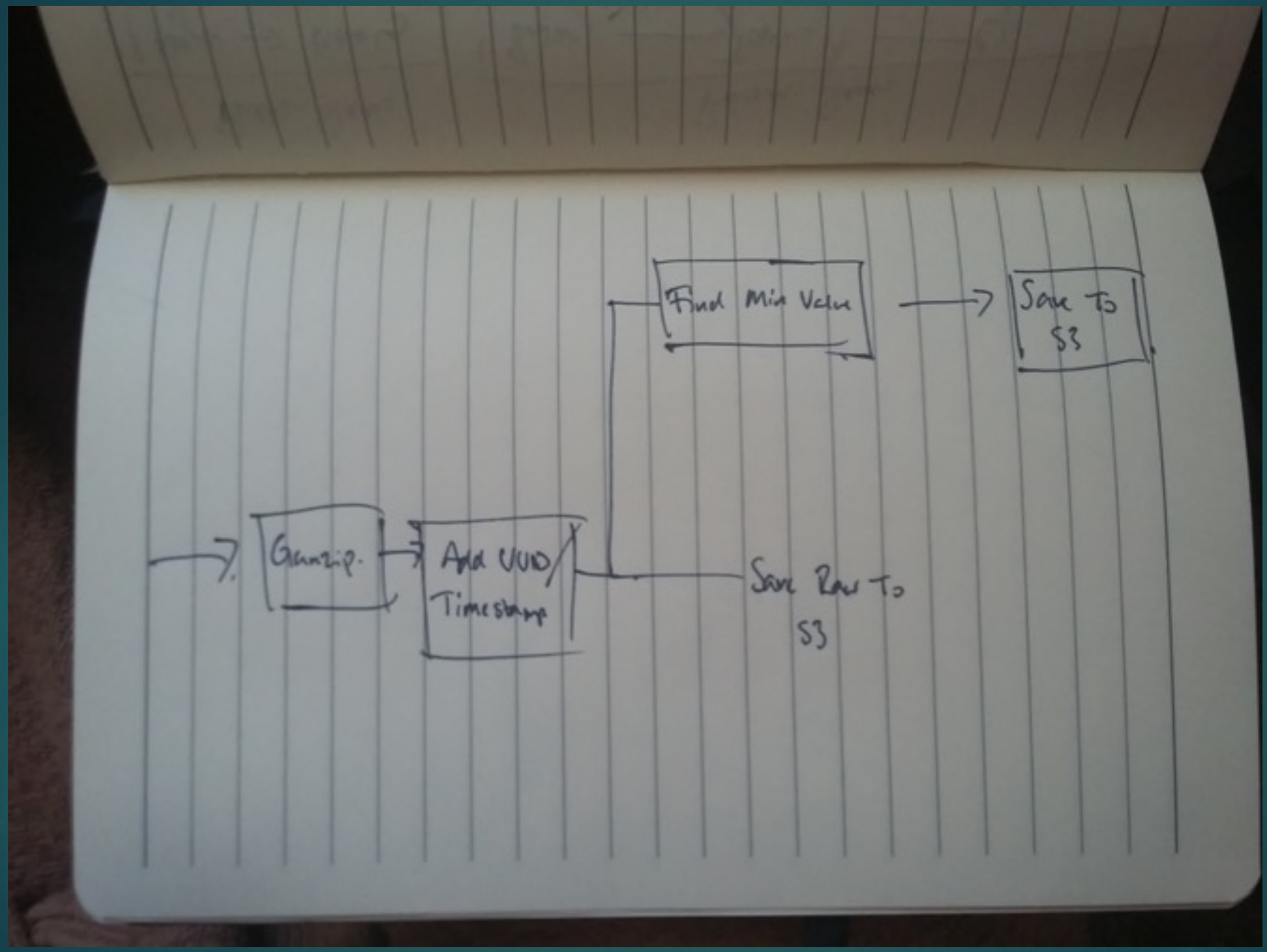
- "We want to stream in 12TB of data a day…… Can you do that?"

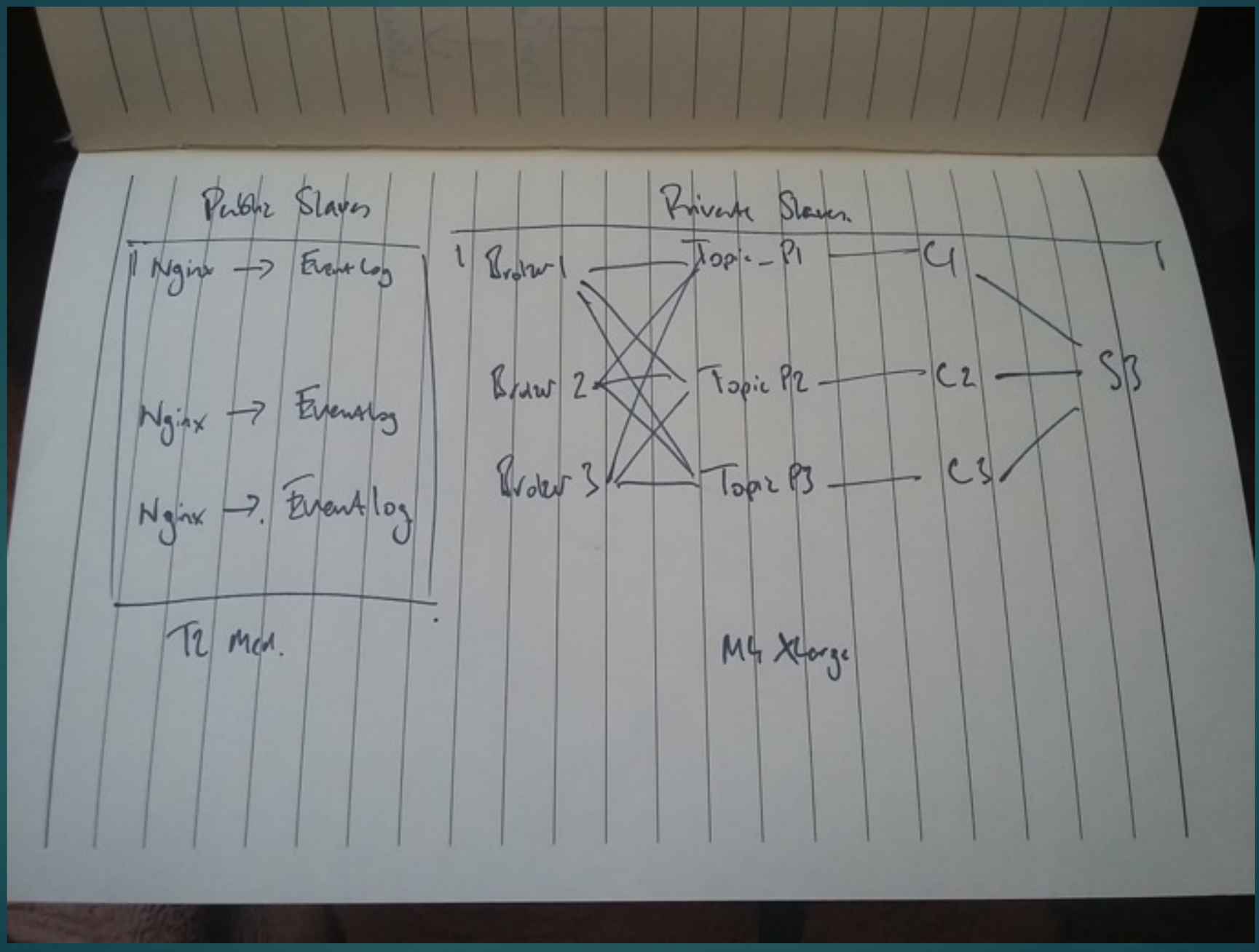
A Streaming Application… kind of.



What is Onyx?

- Distributed Computation Platform
- Written 100% in Clojure
- Great plugin architecture (inputs and outputs)
- Uses graph Direct Acyclic Graph workflow
- I spoke about it at ClojureX 2016
- It's very good….


Find Min Value
Save To S3
Gunzip.
Add UUID Timestamp
Save Raw To S3


Public Slaves
Nginx → Event Log
Nginx → Event Log
Nginx → Event Log
T2 Med.
Private Slaves
Broker 1
Broker 2
Broker 3
Topic - P1
Topic P2
Topic P3
C1
C2
C3
S3
M4 Xlarge

Phase 1 – Testing at 1% Volume

- It's working!
- Stuff's going to S3!
- Tell the client the good news!

Phase 2 – Testing at 2% Volume

- It's working!
- Stuff's going to S3!
- Tell the client the good news!

Phase 3 – Testing at 5% Volume

Know Thy Framework

- Aeron buffer size = 3 x (batch-size x segment size x connections)
  - (1 x 512k x 8) x 3 = 4mb
  - Aeron default buffer is 16mb but then the **twist**
    - Onyx segment size = aeron.term.buffer.size / 8
      - Max message size of 2mb

Onyx in Docker containers.

- --shm-size ended up being 10GB
  - OOM killers were frequent
  - Java logging is helpful but still hard to deal with.

```
CGROUPS_MEM=$(cat /sys/fs/cgroup/memory/memory.limit_in_bytes)
MEMINFO_MEM=$(($(awk '/MemTotal/ {print $2}' /proc/
meminfo)*1024)) MEM=$(($MEMINFO_MEM>$CGROUPS_MEM?$CGROUPS_MEM:
$MEMINFO_MEM)) JVM_PEER_HEAP_RATIO=${JVM_PEER_HEAP_RATIO:-0.6}
XMX=$(awk '{printf("%d",$1*$2/1024^2)}' <<< " ${MEM} $
{JVM_PEER_HEAP_RATIO} ") # Use the container memory limit to
set max heap size so that the GC # knows to collect before
it's hard-stopped by the container environment, # causing OOM
exception.
```

One Friday evening…

2. I did a rewrite in Kafka Streams.

- Moved the pure Clojure functions into the streams architecture.
- Used Amazonica to write to AWS S3.

Kafka Streams rewrite…

- Took 2 hours to write and deploy to DCOS.
- And, by this stage, was slightly tipsy.

Kafka Streams rewrite…

- In deployment Kafka Streams didn't touch more than 2% of the CPU/Memory load on the instance.
- Max memory was 790Mb compared to 8Gb Onyx jobs.

Monday Morning…

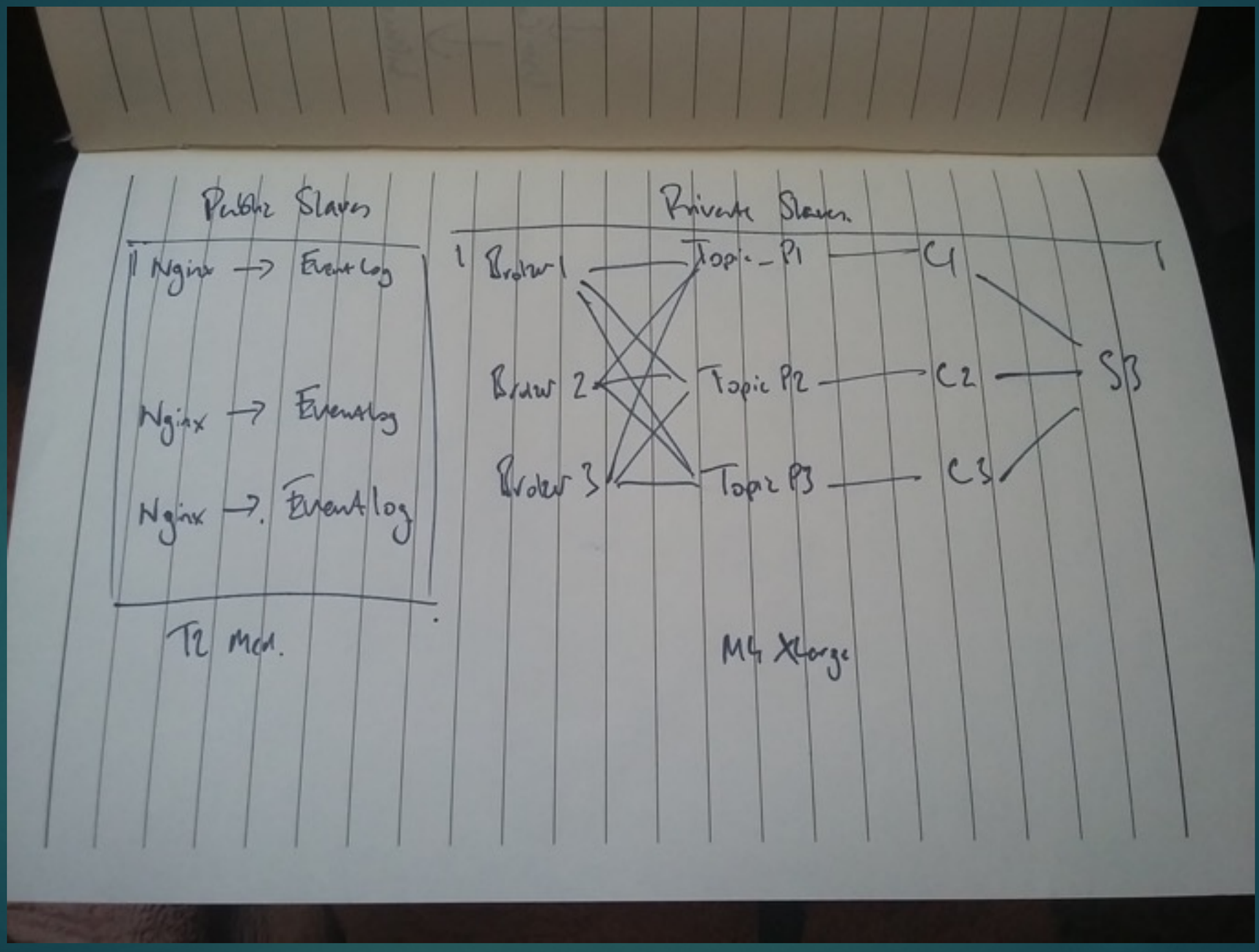
- Now might be a good time to tell the CTO. ;)

Now lets make it better.

I need to break down the process into separate components.

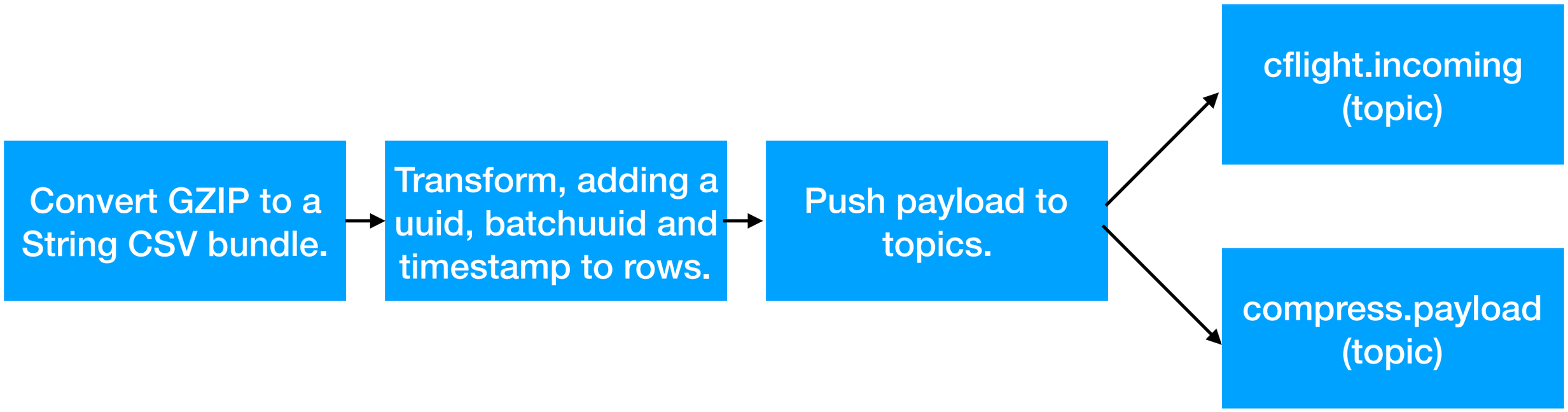
Doing everything in one streaming app is bad!

The Event Log Producer


Public Slaves
Nginx → Event Log
Nginx → Event Log
Nginx → Event Log
T2 Med.
Private Slaves
Broker 1
Broker 2
Broker 3
Topic - P1
Topic P2
Topic P3
C1
C2
C3
S3
M4 Xlarge

I'm going to make this into a streaming application.

Streaming Event Log




```
LHR,IST,2020-08-01,169.99,Y,b6e87fb96f73c119caed08d6e9509c66,39b6aba33172e379c763fcbd1b23aad7,2020-08-01T09:00:00
```

Deserialise Decompress Step

```
(defn deserialize-gzip-message [bytes]
  (try (-> bytes
           ByteArrayInputStream.
           GZIPInputStream.
           io/reader
           slurp)
       (catch Exception e {:error e})))
```

```
LHR,IST,2020-08-01,169.99,Y,b6e87fb96f73c119caed08d6e9509c66,39b6aba33172e379c763fcbd1b23aad7,2020-08-01T09:00:00
```

Transform Step

```
Define a batch UUID called batchuuid
Define a batch timestamp
Define a StringBuilder outputbundle.

Split bundle by newline:
For each CSV line:
  Define a uuid
  Append the uuid
  Append the batchuuid
  Append the timestamp
  Append complete line to outputbundle + newline
Push outputbundle string to compress.payload
Push outputbundle string to cflight.incoming
```

```
LHR,IST,2020-08-01,169.99,Y,b6e87fb96f73c119caed08d6e9509c66,39b6aba33172e379c763fcbd1b23aad7,2020-08-01T09:00:00
```

The Cheapest Price Streaming App

Option 1 - Compare each line of the CSV bundle.

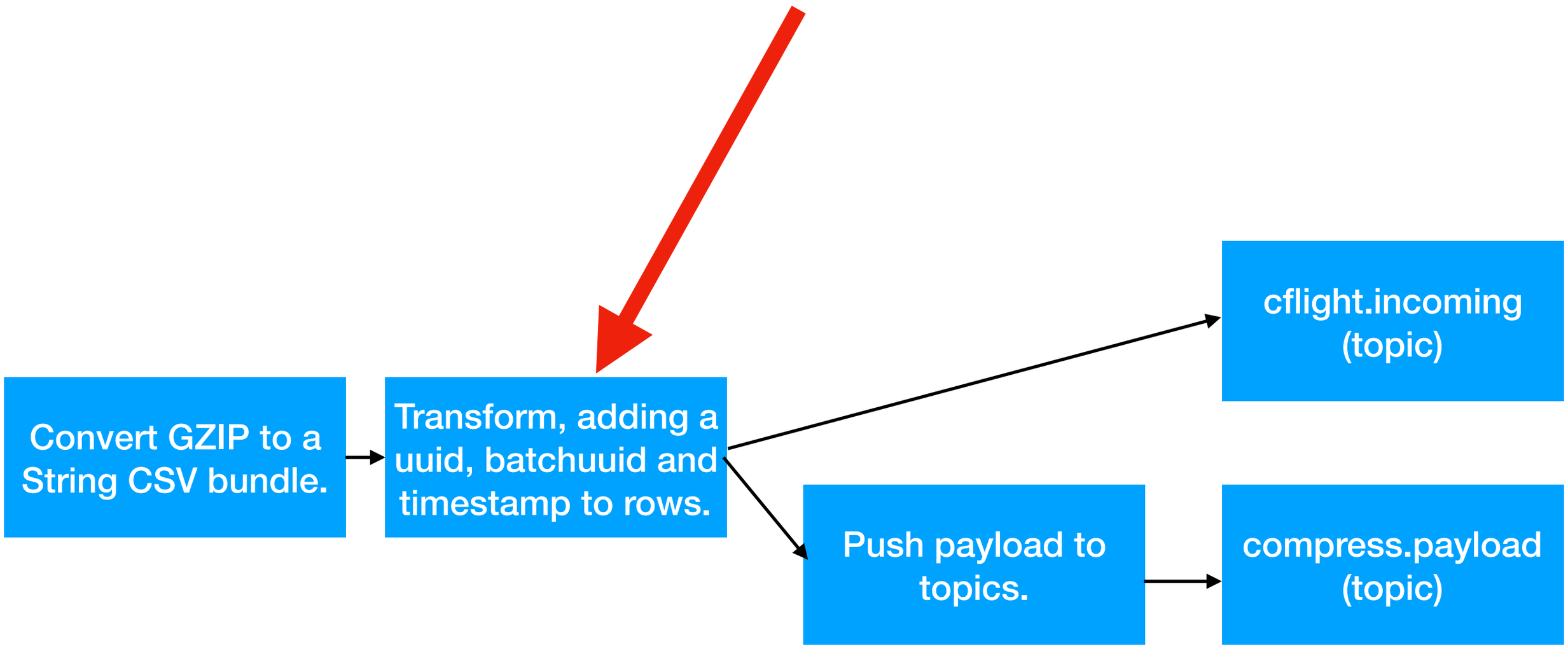
```
(defn get-price
  [row]
  (let [first-comma (inc (cstr/index-of row ","))
        last-comma (cstr/index-of row "," first-comma)
        price (subs row first-comma last-comma)]
    (Double/parseDouble price)))

;; Finds the cheapest flight on a row by row comparison.
;; Uses get-price function.
(defn find-cheapest-flight [data]
  (println "Finding cheapest flight")
  (let [rows (cstr/split data #"\n")
        [price cheapest-flight] (reduce (fn [[cheapest-price cheapest-row :as cheapest]
                                             [candidate-price candidate-row :as candidate]]
                                          (if (> cheapest-price
                                                 candidate-price)
                                            candidate
                                            cheapest))
                                        (map (fn [row-o]
                                               [(get-price row-o) row-o])
                                             rows))]
    (str cheapest-flight "\n"))) ;; put the EOL in before we serialise it again
```

OR????????

KSQL?

Option 2 - Step 1: Convert the CSV bundle to AVRO payload


```
{“depiata":"LHR",
 “arriata":"IST",
 “searchdate”:"2020-08-01",
 “gbpprice”:169.99,
 “paxclass":"Y",
 “uuid":"b6e87fb96f73c119caed08d6e9509c66",
 “batchuuid":"39b6aba33172e379c763fcbd1b23aad7",
 "timestamp":"2020-08-01T09:00:00"}
```




```
LHR,IST,2020-08-01,169.99,Y,b6e87fb96f73c119caed08d6e9509c66,39b6aba33172e379c763fcbd1b23aad7,2020-08-01T09:00:00
```

Event Log Transform Step Amended

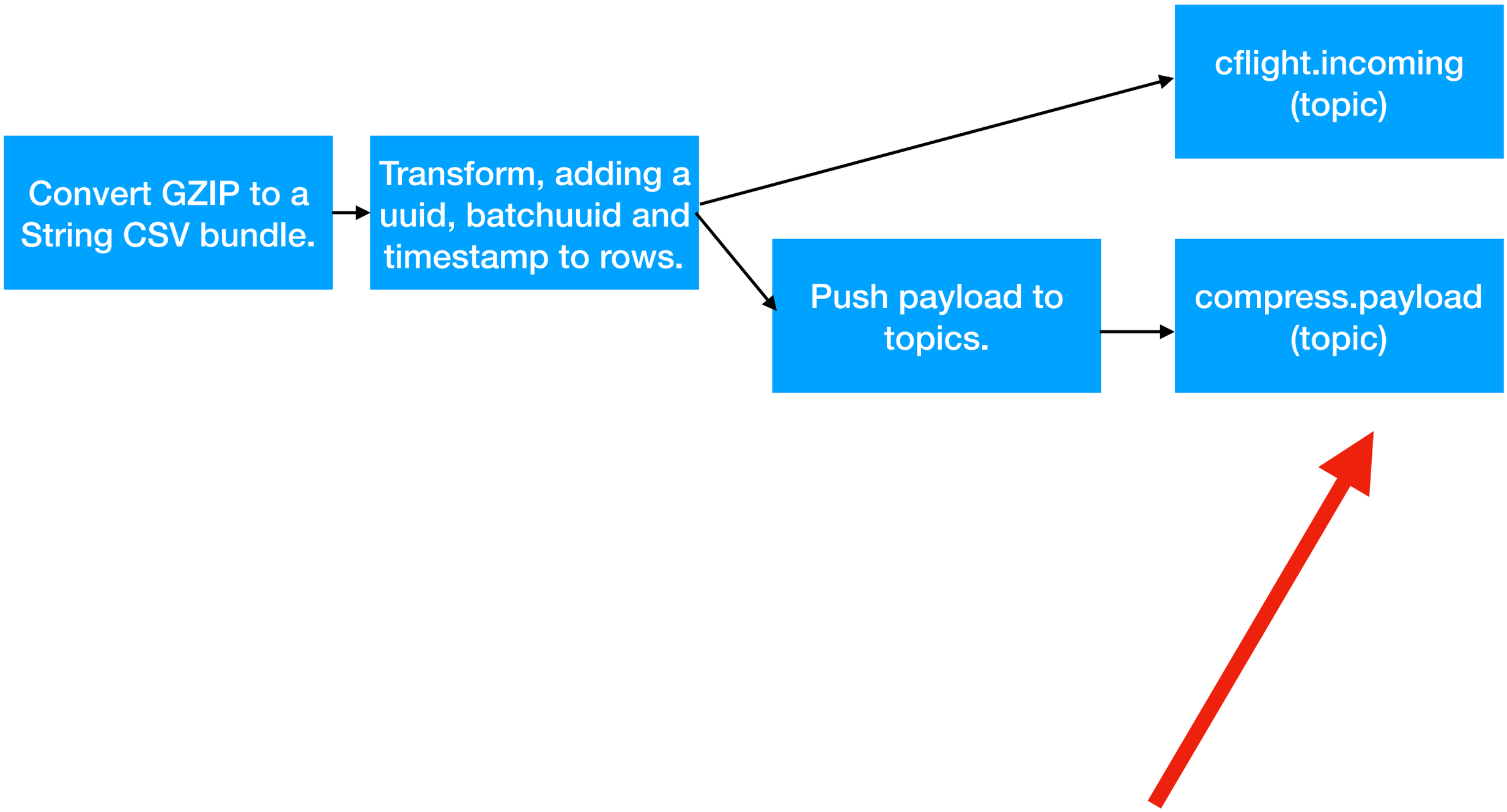
```
Define a batch UUID called batchuuid
Define a batch timestamp
Define a StringBuilder outputbundle.

Split bundle by newline:
For each CSV line:
  Define a uuid
  Append the uuid
  Append the batchuuid
  Append the timestamp
  Append complete line to outputbundle + newline
  Convert line to AVRO
  Push AVRO payload to cflight.incoming topic
Push outputbundle string to compress.payload
```

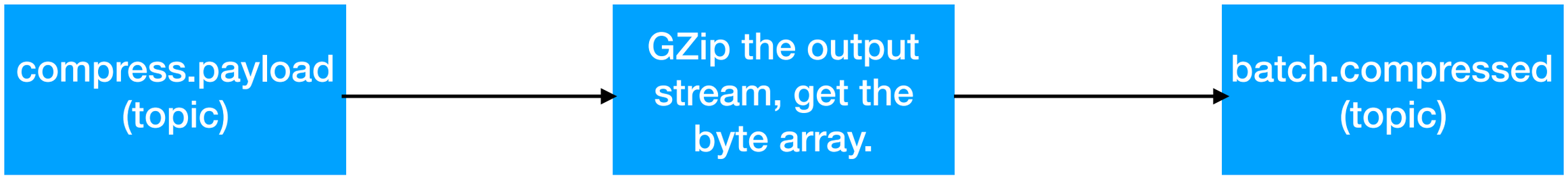
```
LHR,IST,2020-08-01,169.99,Y,b6e87fb96f73c119caed08d6e9509c66,39b6aba33172e379c763fcbd1b23aad7,2020-08-01T09:00:00
```

Option 2 - Step 2: Craft a KSQL Job

```
CREATE STREAM CFLIGHT (…….)
    WITH (KAFKA_TOPIC=‘cflight.incoming',
          PARTITIONS=1,
          VALUE_FORMAT='avro');

CREATE STREAM CFLIGHT_CHEAPEST AS
    SELECT UUID, BATCHUUID,
           MIN(GBPPRICE) AS CHEAPEST_QUOTE
    FROM CFLIGHT
    GROUP BY BATCHUUID
    EMIT CHANGES;
```

The Batch Compressor Streaming App


Convert GZIP to a String CSV bundle.
Transform, adding a uuid, batchuuid and timestamp to rows.
cflight.incoming (topic)
Push payload to topics.
compress.payload (topic)


compress.payload (topic)
GZip the output stream, get the byte array.
batch.compressed (topic)

Compress and Serialise Step

```
(def gzip-serializer-fn
  (fn [output-str]
    (let [out (java.io.ByteArrayOutputStream.)]
      (do (doto (java.io.BufferedOutputStream.
                  (java.util.zip.GZIPOutputStream. out))
        (.write (.getBytes output-str))
        (.close)))
(.toByteArray out))))
```

Persist to S3

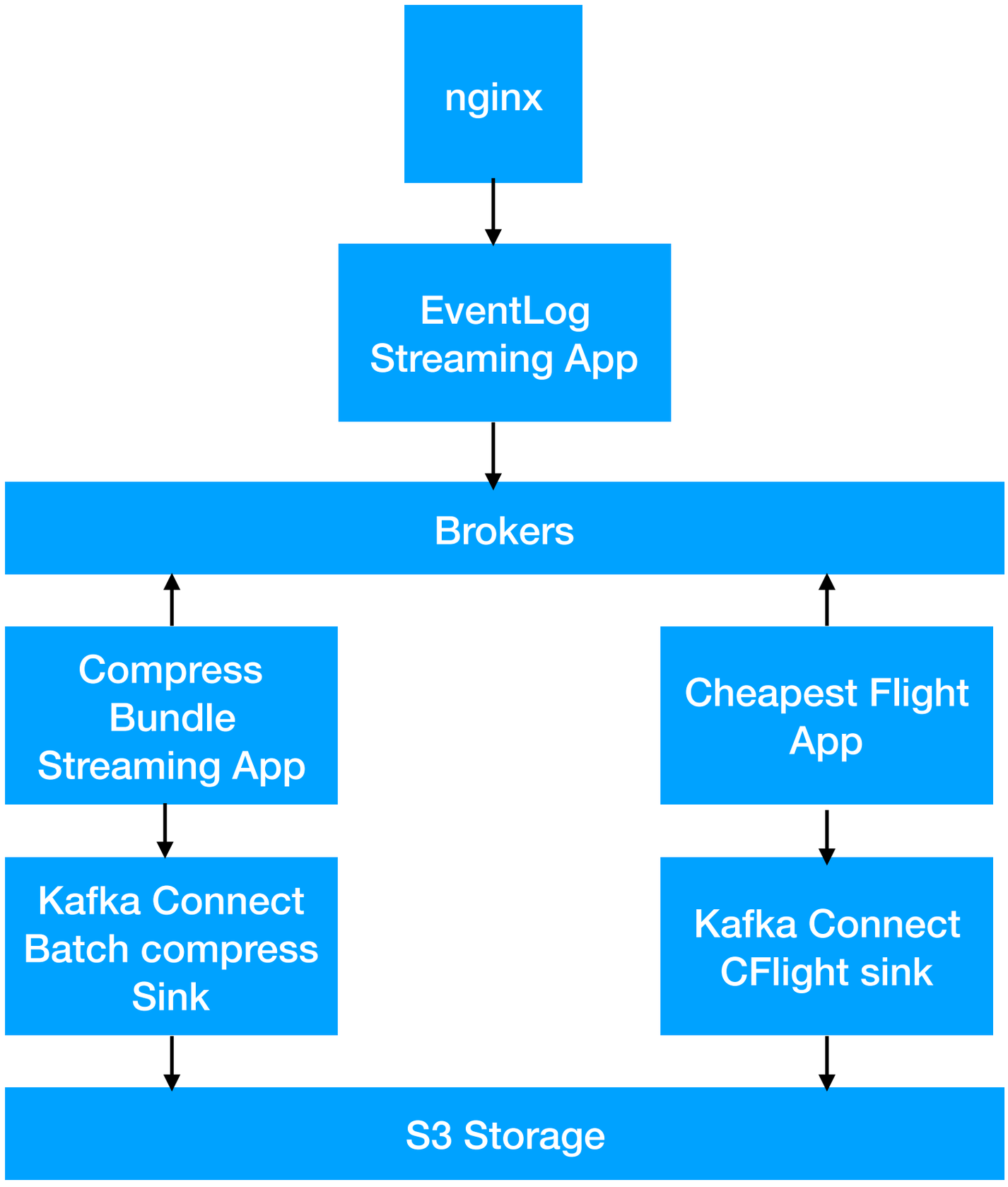
Kafka Connect

Use Prebundled connectors where you can.

Also means you have access to further filter and transform steps if required.

```
{"name": "cflight.sink",
 "config":
 {"connector.class":"io.confluent.connect.s3.S3SinkConnector",
  "tasks.max":"1",
  “topics”:"CHEAPEST_FLIGHT",
  "s3.region":"us-east-1",
"s3.bucket.name": "cflight.bucket",
"s3.part.size":"5242880",
"flush.size":"1",
"storage.class":"io.confluent.connect.s3.storage.S3Storage",
"format.class": "io.confluent.connect.s3.format.avro.AvroFormat",
"partitioner.class":"io.confluent.connect.storage.partitioner.DefaultPartitioner",
"schema.compatibility":"NONE"}}
```

```
{"name": "batchcompress.sink",
 "config":
 {"connector.class":"io.confluent.connect.s3.S3SinkConnector",
  "tasks.max":"1",
  “topics":"batch.compressed",
  "s3.region":"us-east-1",
"s3.bucket.name": "flightbatch.bucket",
"s3.part.size":"5242880",
"flush.size":"1",
"storage.class":"io.confluent.connect.s3.storage.S3Storage",
"format.class": "io.confluent.connect.s3.format.bytearray.ByteArrayFormat",
"partitioner.class":"io.confluent.connect.storage.partitioner.DefaultPartitioner",
"schema.compatibility":"NONE"}}
```


nginx
EventLog
Streaming App
Brokers
Compress
Bundle
Streaming App
Cheapest Flight
App
Kafka Connect
Batch compress
Sink
Kafka Connect
CFlight sink
S3 Storage

Thank you.

Many thanks to Shay and David for organising, everyone who attended and sent kind wishes. Lastly, a huge thank you to MeetupCat.

Photo supplied by @jbfletch_