Streaming Log Analytics with Kafka

Kresten Krab Thorup, Humio CTO

Log Everything,

Answer Anything,

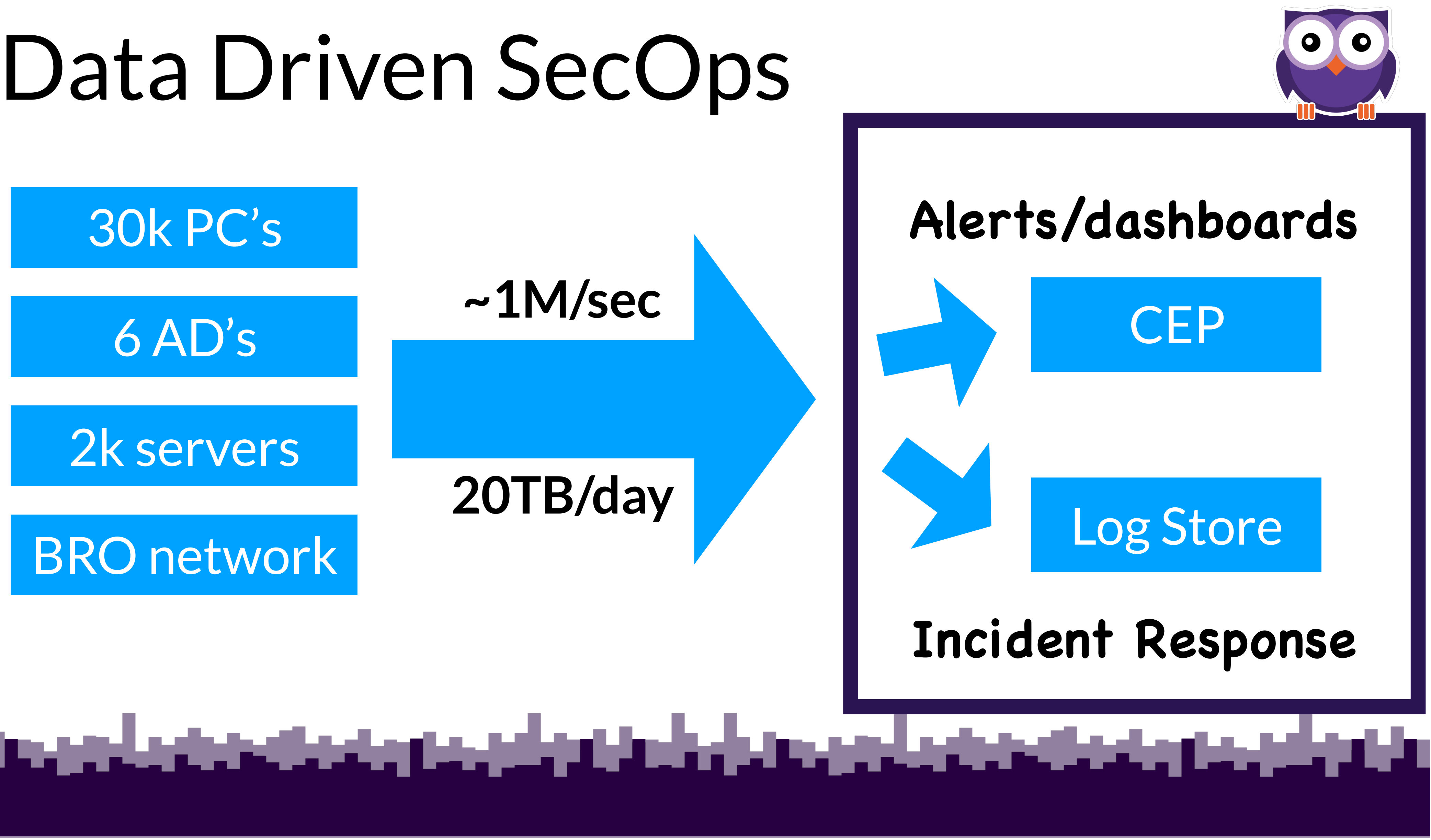
In Real-Time.

Why this talk?

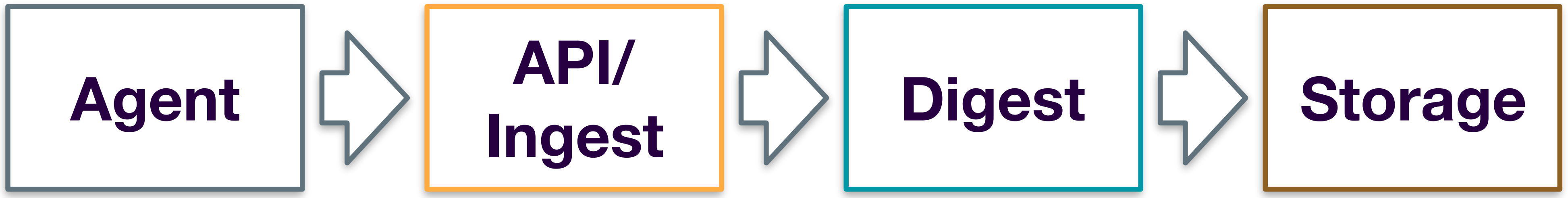
- Humio is a Log Analytics system
    - Designed to run "on-prem"
    - High volume, real time responsiveness.

- We decided to delegate the 'hard parts' of distributed systems to Kafka. This is a talk about our experiences.

Data Driven SecOps



Humio Ingest Data Flow



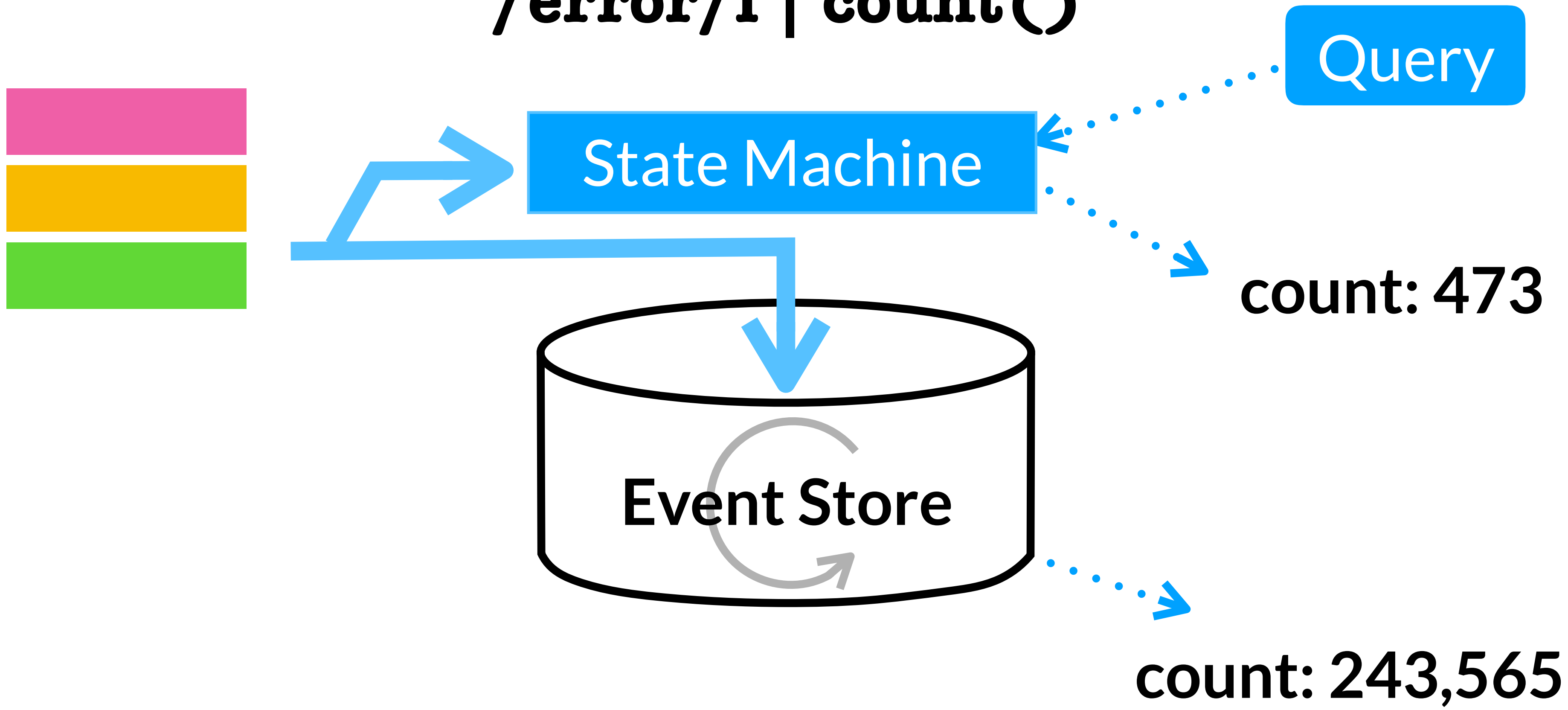
- Send data

- HTTP/TCP API
- Authenticate
- Field Extraction

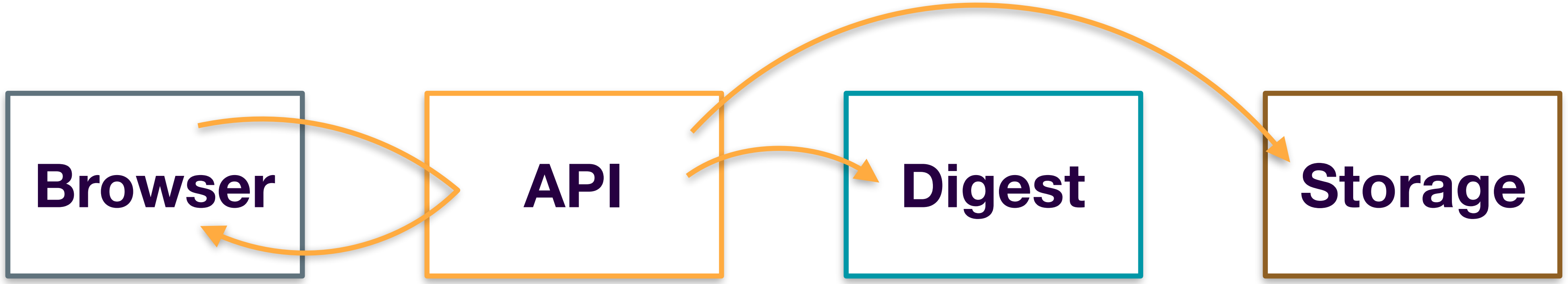
- Streaming queries
- Write segment files

- Replication

```
/error/i | count()
```



Humio Query Flow



- Start Query
- Poll Status

- Initiate Query
- Merge results
- Schedule polls

- Provide results for live data (materialized view)

- Provide results for historic data (ad-hoc query)

Real-time Processing

- “Materialized views” for dashboards/alerts.
- Processed when data is in-memory anyway.
- Fast response times for “known” queries.

Brute-Force Search

- Shift CPU load to query time
- Data compression
- Allows ad-hoc queries
- Requires “Full stack” ownership

Use Kafka for the ‘hard parts’

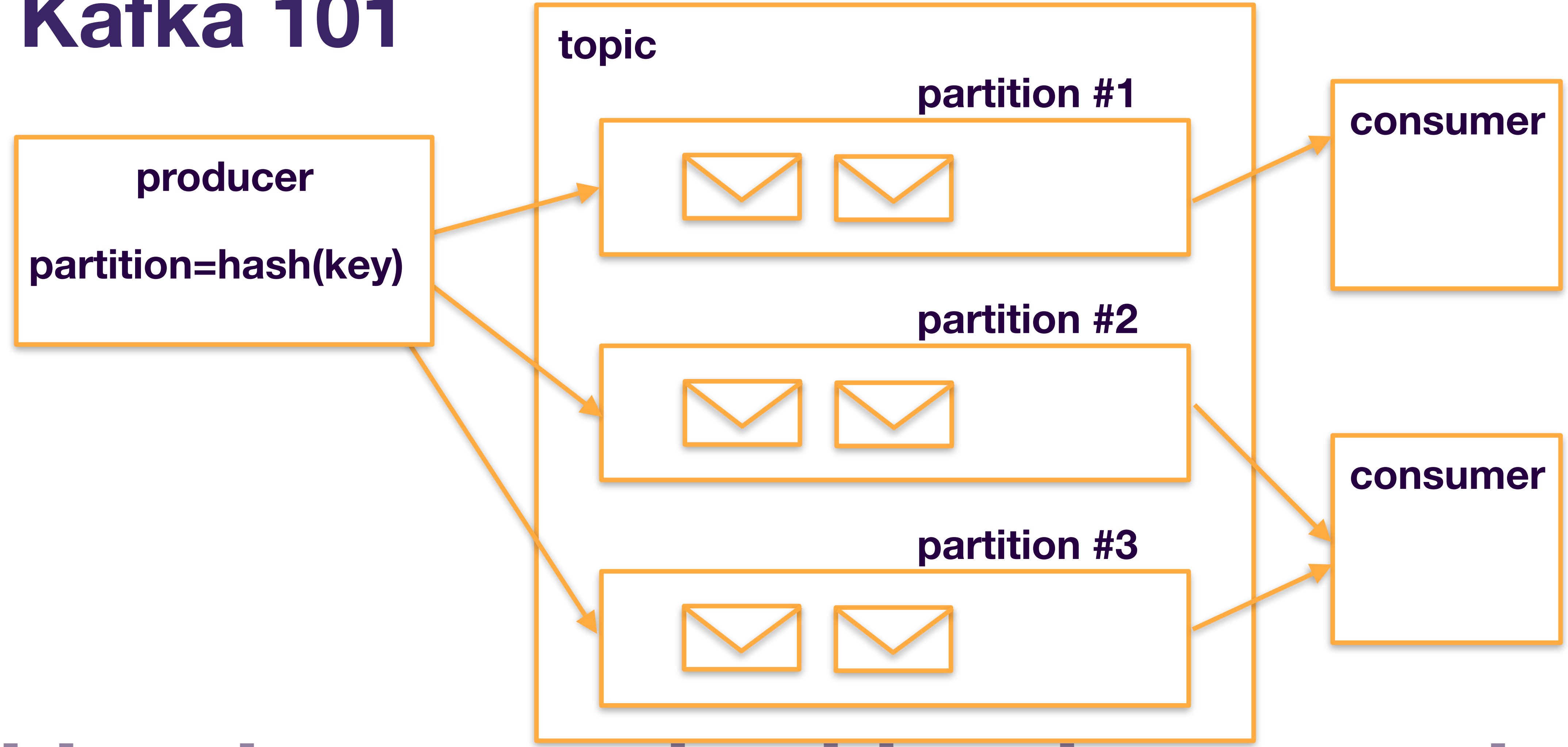
- Coordination
- Commit-log / ingest buffer
- Transient data

- No KSQL

Kafka 101

- Kafka is a <u>reliable</u> distributed log/queue system
- A Kafka queue consists of a number of partitions
- Messages within a partition are sequenced
- Partitions are replicated for durability
- Use ‘partition consumers’ to parallelise work

Kafka 101


producer
partition=hash(key)
topic
partition #1
partition #2
partition #3
consumer
consumer

Coordination ‘global data’

- Zookeeper-like system in-process
- Hierarchical key/value store
- Make decisions locally/fast without crossing a network boundary.
- Allows in-memory indexes of meta data.

Coordination ‘global data’

- Coordinated via single-partition Kafka queue
- Ops-based CRDT-style event sourcing
- Bootstrap from snapshot from any node
- Kafka config: low latency

Log Store Design

- **Build minimal index and compress data**

  Store order of magnitude more events

- **Fast “grep” for filtering events**

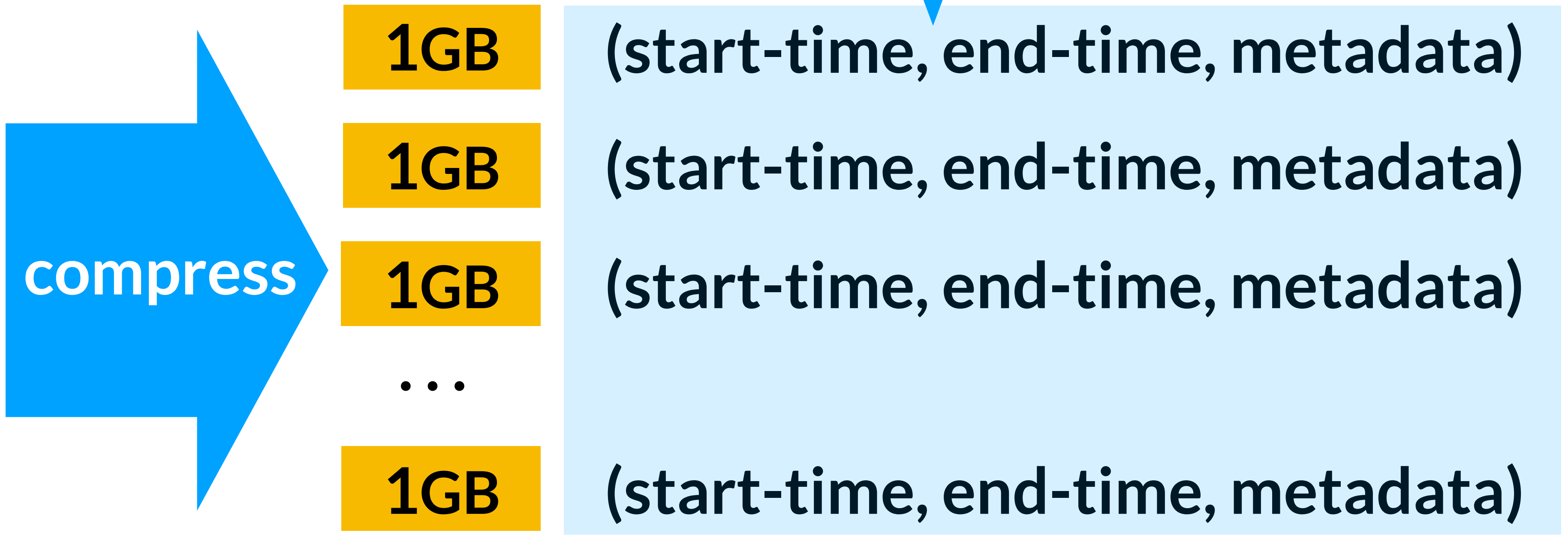
  Filtering and time/metadata selection reduces the problem space

Event Store

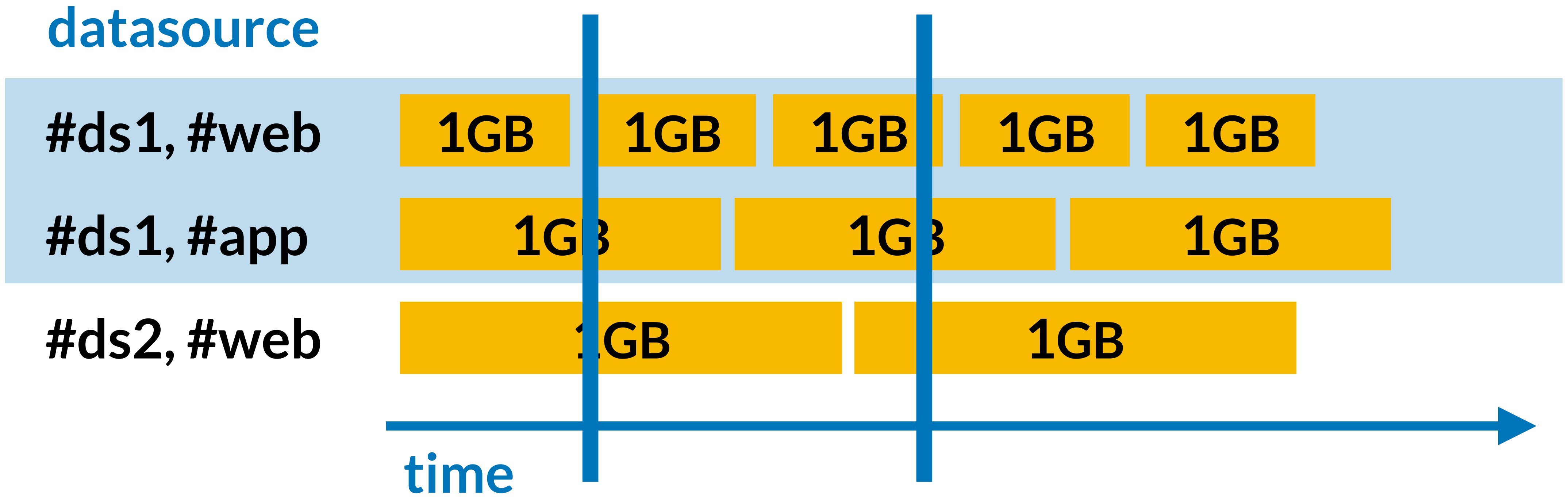
10GB	(start-time, end-time, metadata)
10GB	(start-time, end-time, metadata)
10GB	(start-time, end-time, metadata)
. . .	
10GB	(start-time, end-time, metadata)

Event Store

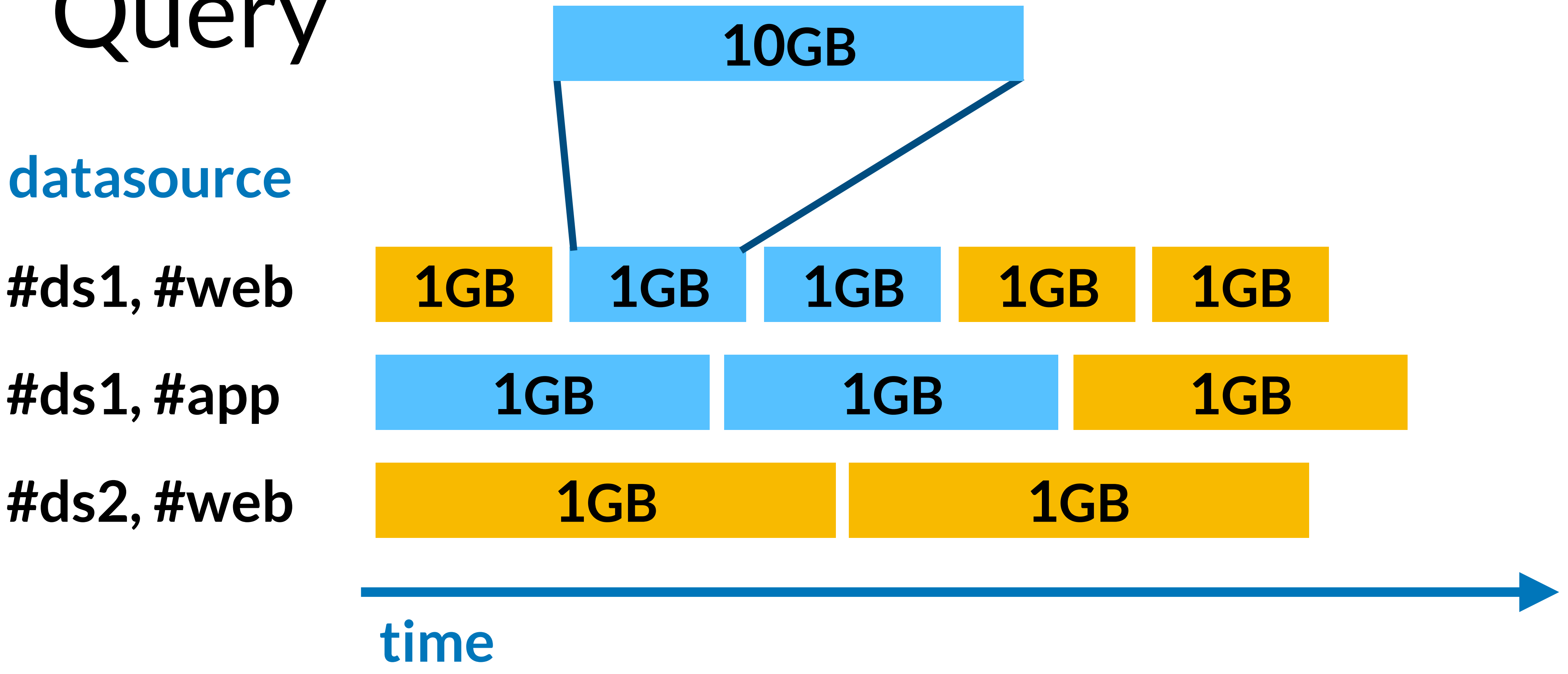
1 month x 1TB/day ingest
4TB data, <1MB index


compress
1GB
(start-time, end-time, metadata)
1GB
(start-time, end-time, metadata)
1GB
(start-time, end-time, metadata)
...
1GB
(start-time, end-time, metadata)

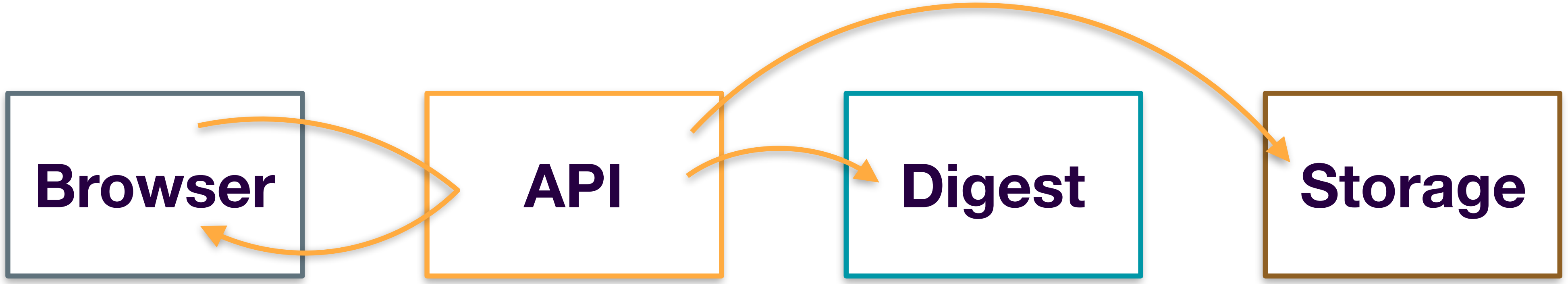
Query


datasource
#ds1, #web
#ds1, #app
#ds2, #web
1GB
1GB
1GB
1GB
1GB
1GB
1GB
1GB
1GB
1GB
time

Query


10GB
datasource
#ds1, #web
1GB
1GB
1GB
1GB
1GB
#ds1, #app
1GB
1GB
1GB
#ds2, #web
1GB
1GB
time

Humio Query Flow



- Start Query
- Poll Status

- Schedule Query
- Merge results

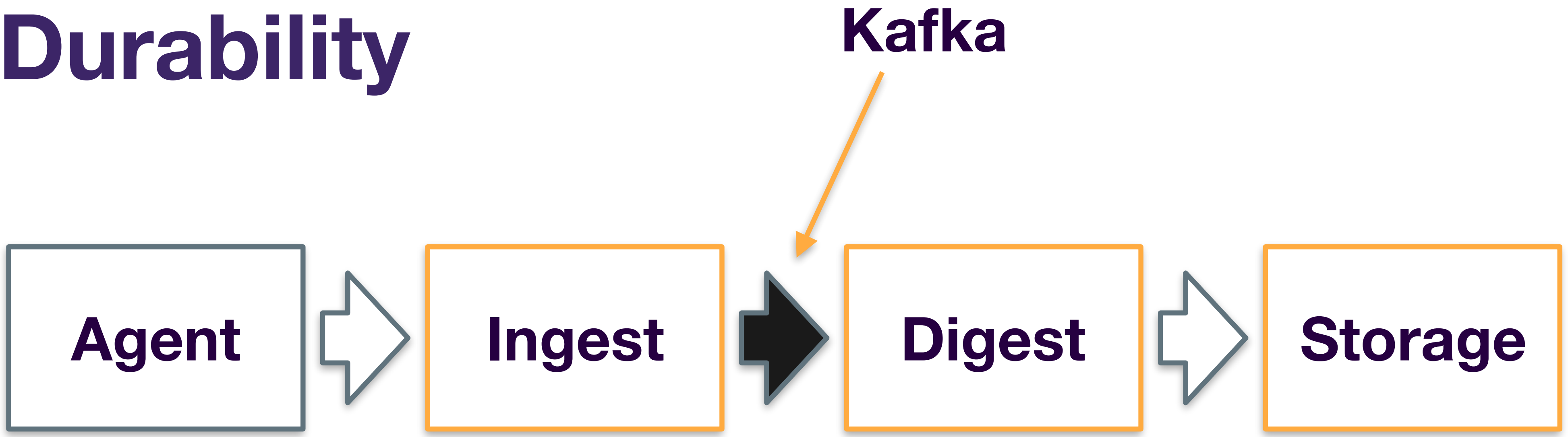
- Provide results for live data (materialized view)

- Provide results for historic data (ad-hoc query)

Durability

- Don’t loose people’s data.
- Control and manage data life expectancy
- Store, Replicate, Archive, Multi-tier Data storage

Durability



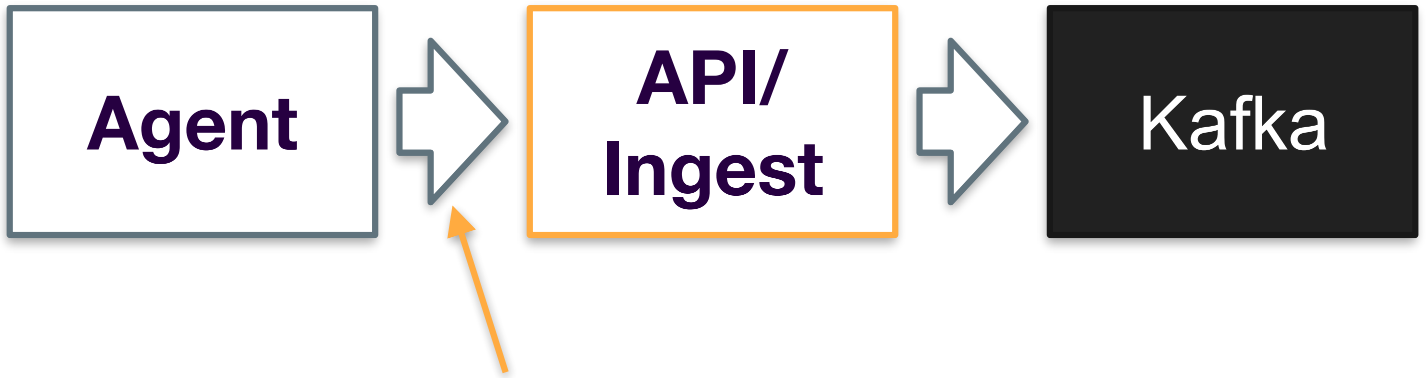
- Send data

- Authenticate
- Field Extraction

- Streaming queries
- Write segment files

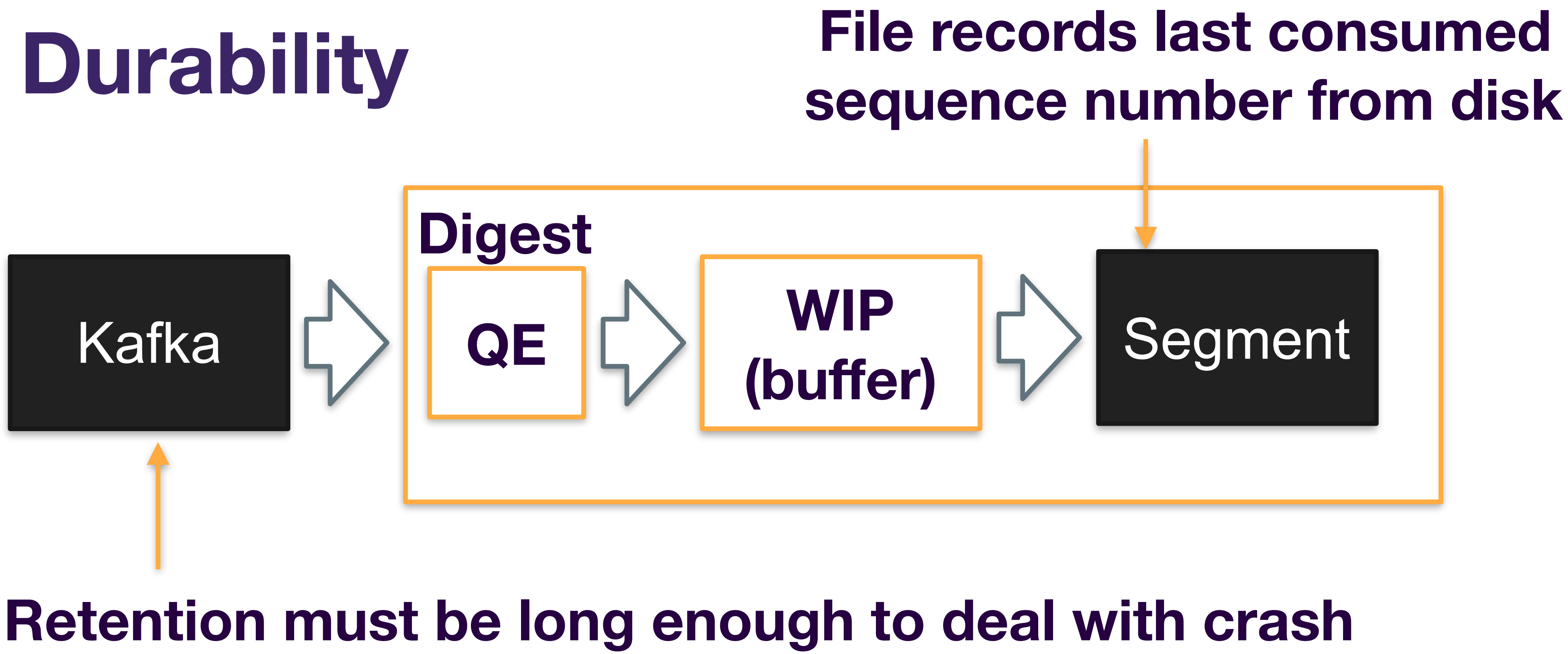
- Replication
- Queries on ‘old data’

Durability

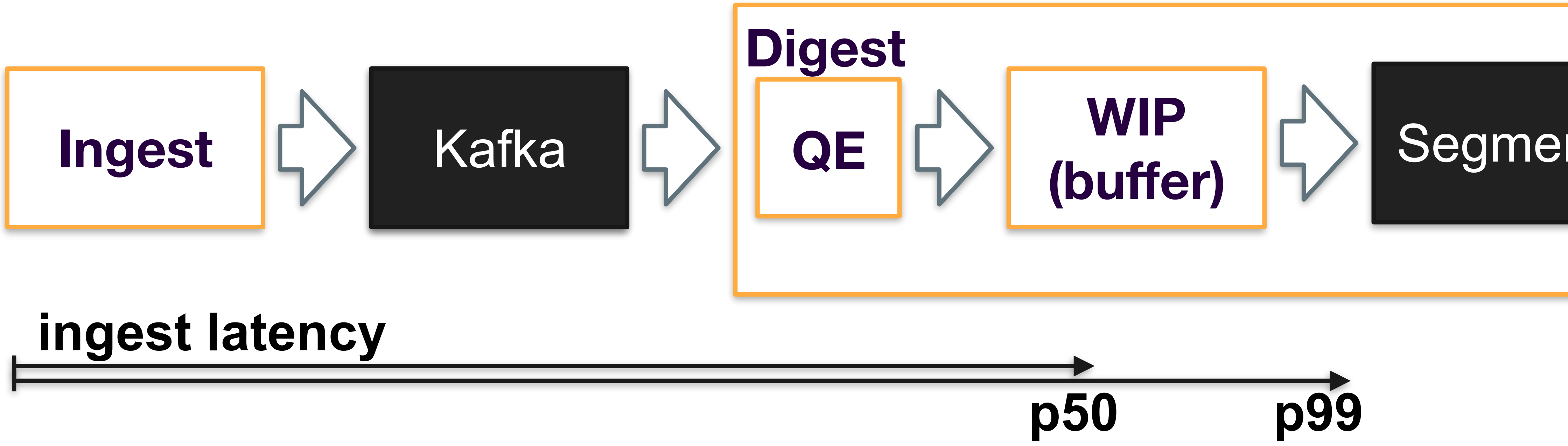


HTTP 200 response => Kafka ACK'ed the store

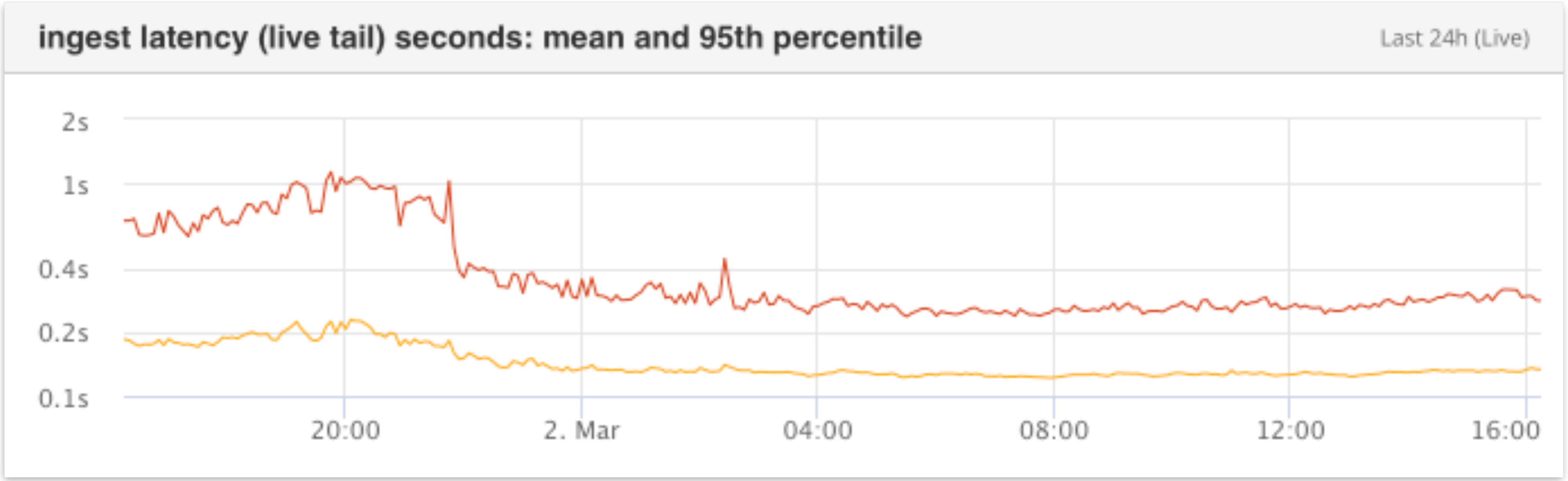
Durability

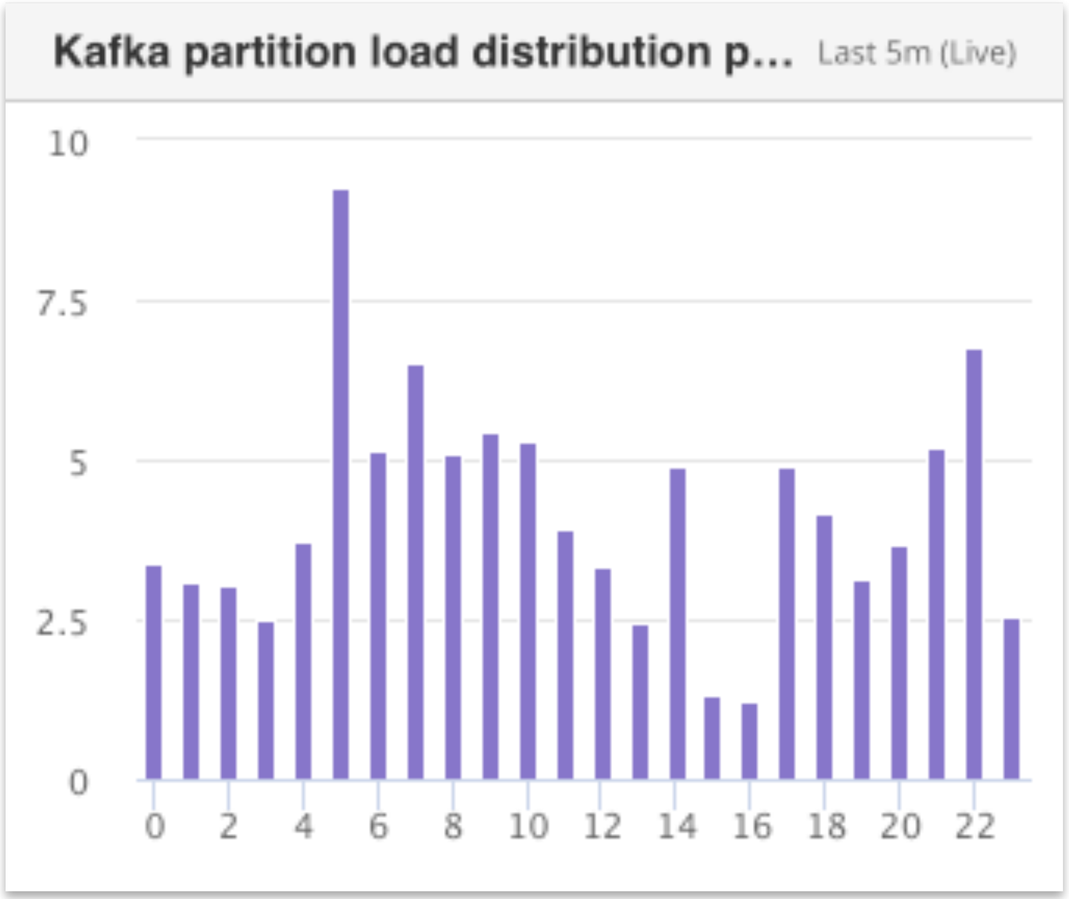

File records last consumed sequence number from disk
Digest
Kafka
QE
WIP (buffer)
Segment
Retention must be long enough to deal with crash

Durability

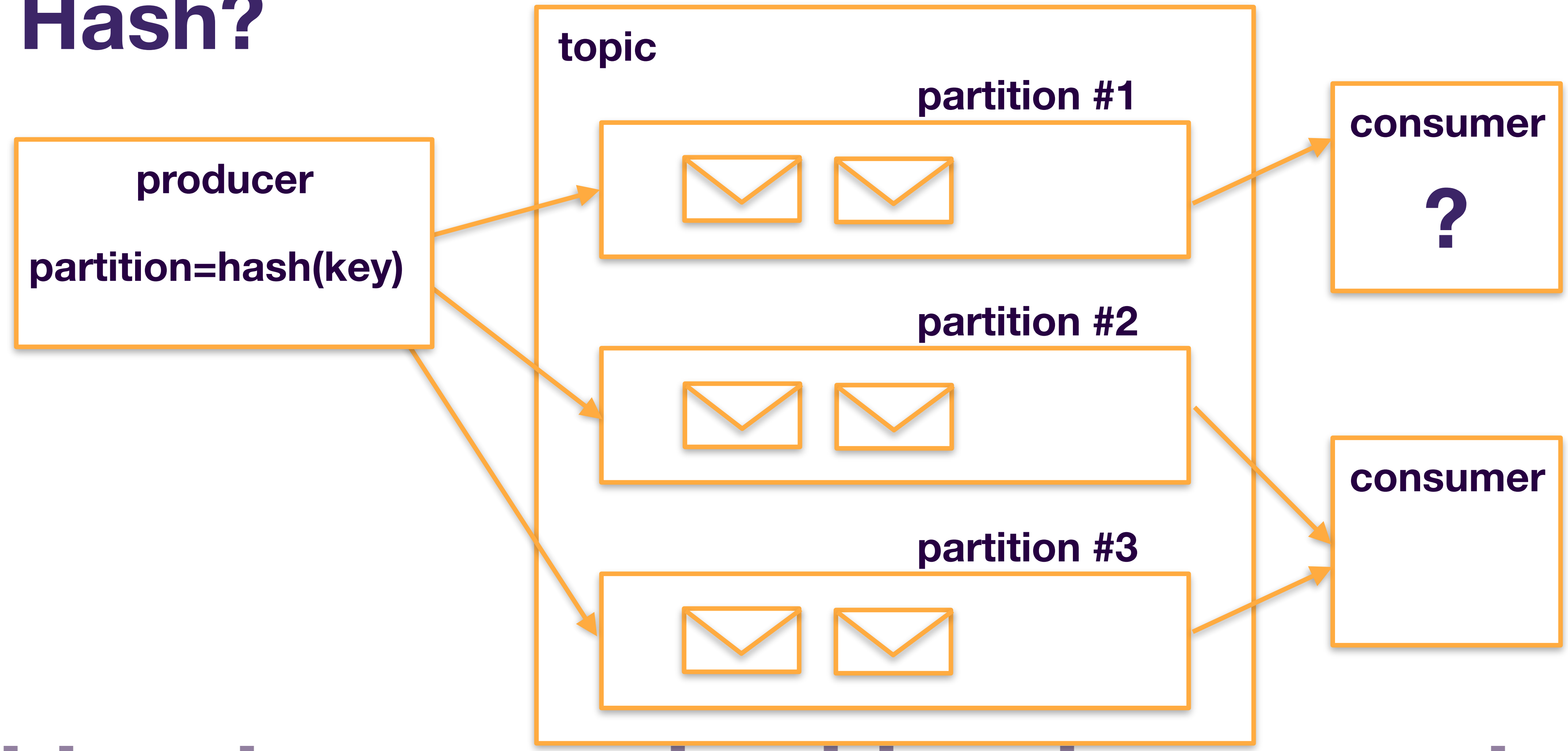


Ingest
Kafka
Digest
QE
WIP (buffer)
Segme
ingest latency
p50
p99


ingest latency (live tail) seconds: mean and 95th percentile
Last 24h (Live)
2s
1s
0.4s
0.2s
0.1s
20:00
2. Mar
04:00
08:00
12:00
16:00


Kafka partition load distribution p…
Last 5m (Live)
10
7.5
5
2.5
0
0
2
4
6
8
10
12
14
16
18
20
22

Hash?

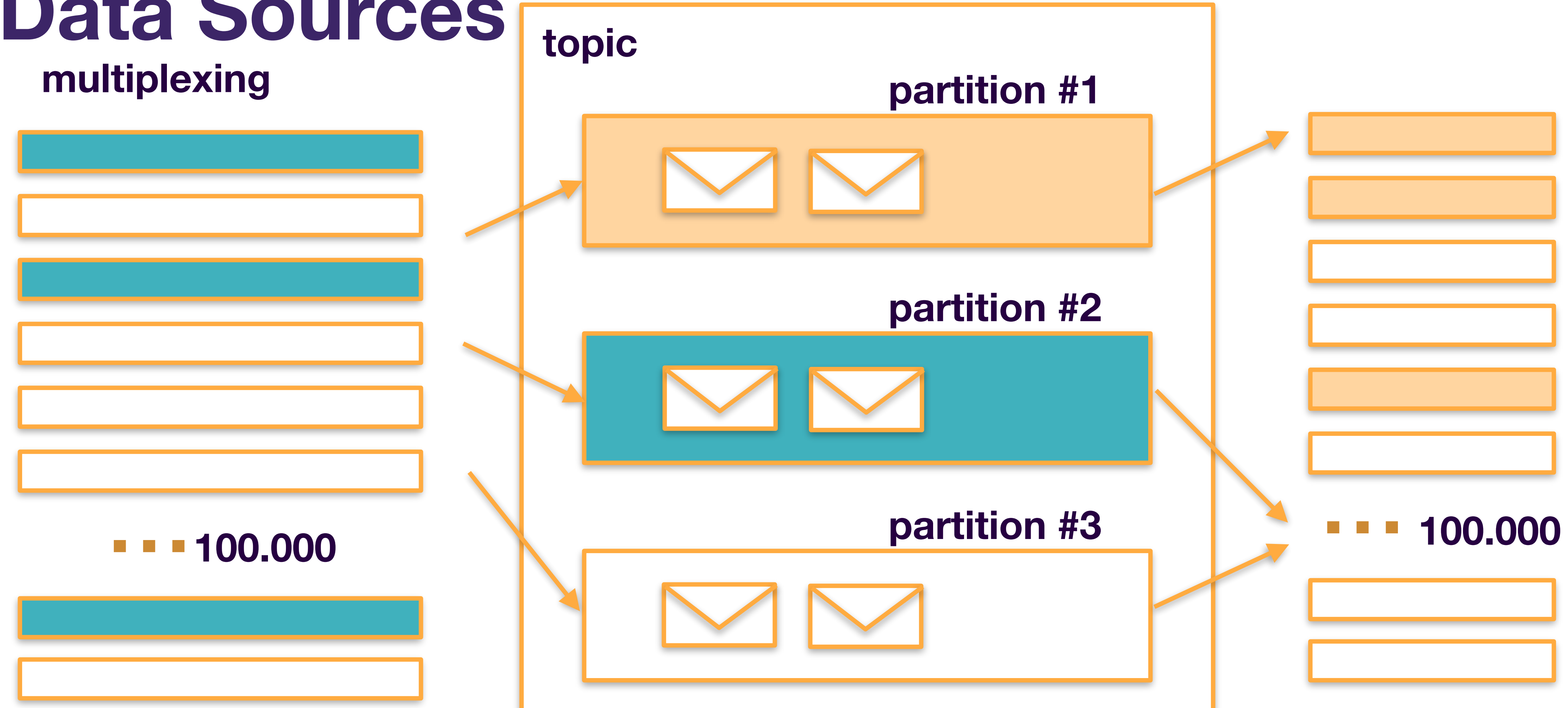

producer
partition=hash(key)
topic
partition #1
partition #2
partition #3
consumer
?
consumer

Partitions falling behind…

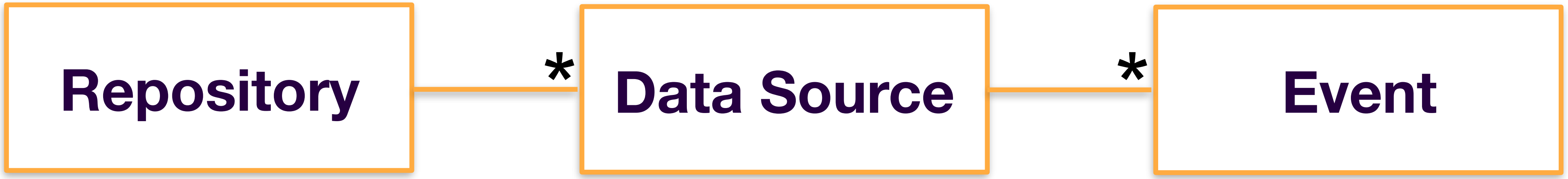
- Reasons:
    - Data volume
    - Processing time for real-time processing
- Measure ingest latency
- Increase parallelism when running 10s behind
    - Log scale (1, 2, 4, …) randomness added to key.

Data Sources

multiplexing



Data Model



- Storage limits
- User admin

- Time series identified by set of key-value ‘tags’

- Timestamp + Map[String,String]

```
Hash (#type=accesslog,#host=ops01 )
```

High variability tags ‘auto grouping’

- Tags (hash key) may be chosen with large value domain
    - User name
    - IP-address

- This causes many datasources => growth in metadata, resource issues.

High variability tags ‘auto grouping’

- Tags (hash key) may be chosen with large value domain
    - User name
    - IP-address

- Humio sees this and <u>hashes</u> tag value into a smaller value domain before the Kafka partition hash.

High variability tags ‘auto grouping’

- For example, before Kafka ingest hash(“kresten”)

  `#user=kresten   =>    #user=13`

  - Store the actual value ‘`kresten`’ in the event

- At query time, a search is then rewritten to search the data source `#user=13`, and re-filter based on values.

Multiplexing in Kafka

- Ideally, we would just have 100.000 dynamic topics that perform well and scales infinitely.
- In practice, you have to know your data, and control the sharding. Default Kafka configs work for many workloads, but for maximum utilisation you have to do go beyond defaults.

Using Kafka in an on-prem Product

- Leverage the stability and fault tolerance of Kafka
- Large customers often have Kafka knowledge
- We provide kafka/zookeeper docker images
- Only real issue is Zookeper dependency
    - Often runs out of disk space in small setups

Other Issues

- Observed GC pauses in the JVM
- Kafka and HTTP libraries compress data
- JNI/GC interactions with `byte[]` can block global GC.
- We replaced both with custom compression
  - JLibGzip (gzip in pure Java)
  - LZ4/JNI using DirectByteBuffer

Resetting Kafka/Zookeeper

- Kafka provides a ‘cluster id’ we can use as epoch
- All Kafka sequence numbers (offsets) are reset
- Recognise this situation, no replay beyond such a reset.

What about KSQL?

- Kafka now has KSQL which is in many ways similar to the engine we built
    - Humio moves computation to the data,
    - KSQL moves the data to the computation
- We provide interactive end-user friendly experience

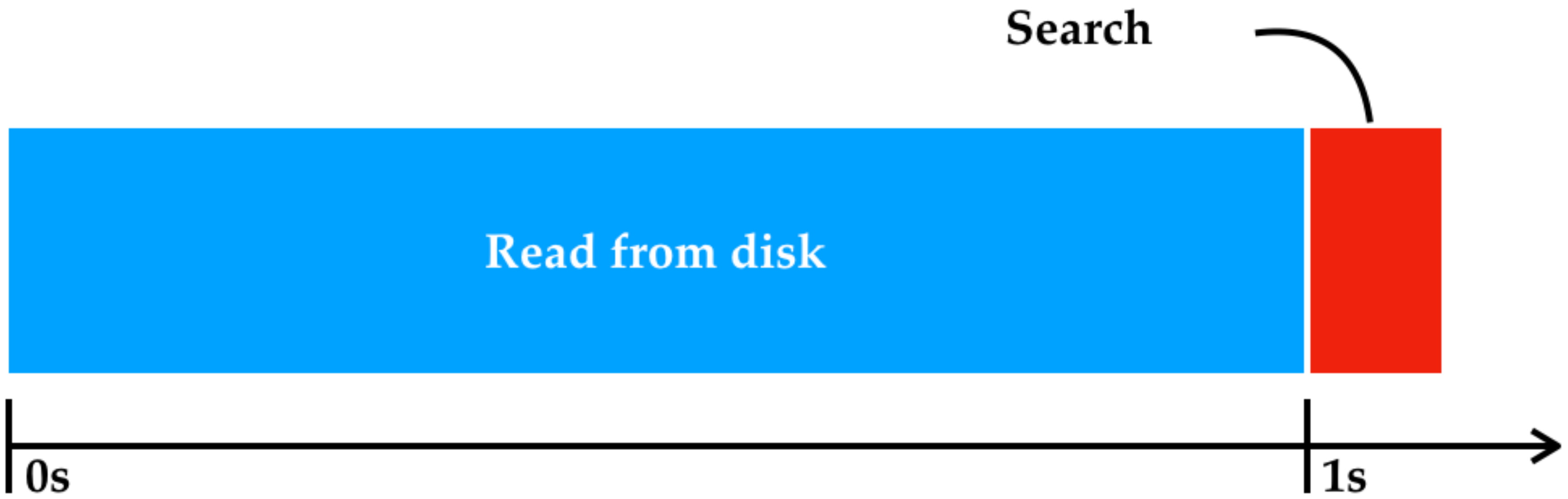
Final thoughts

- Many difficult problems go away by using Kafka.
- We’ve been happy with the decision to defer the ‘hard parts’ of distributed systems to Kafka.
- Some day we may build our own persistent commit log, but for how it is not worth the trouble.

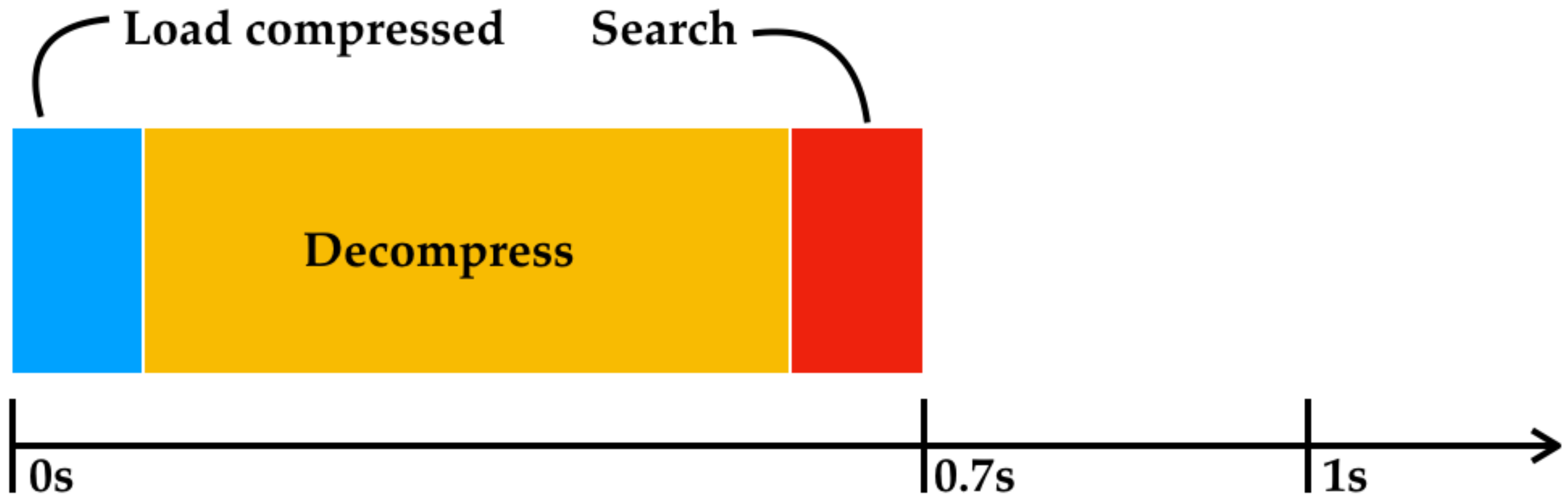
Thanks for your time.

Kresten Krab Thorup
Humio CTO

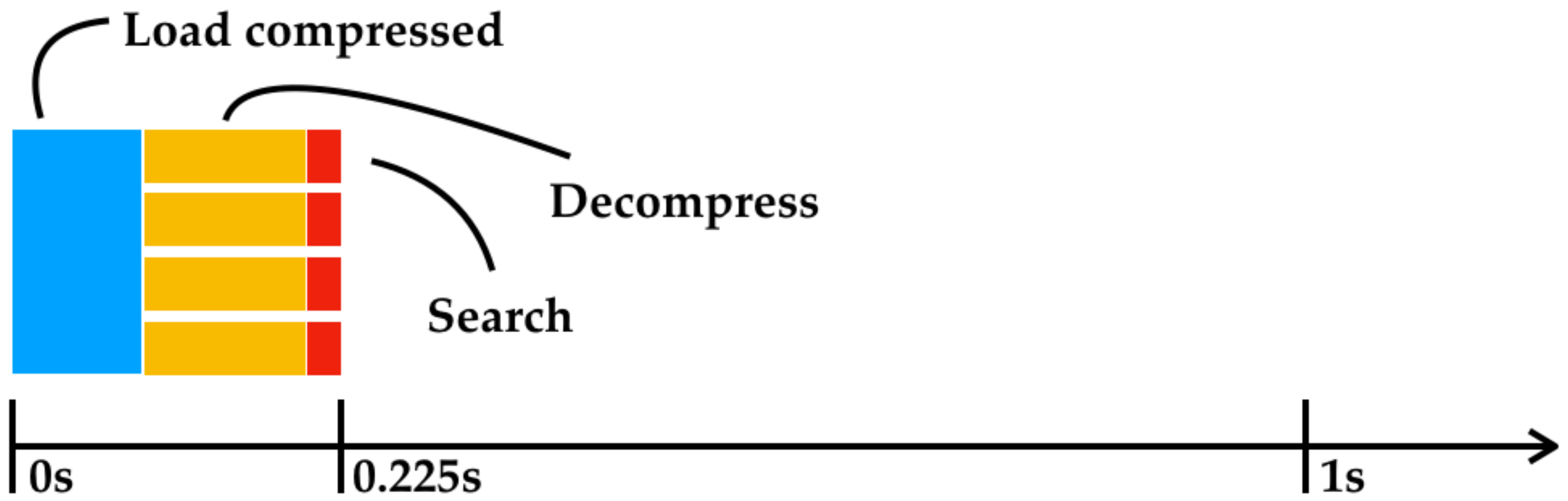
Filter 1GB data



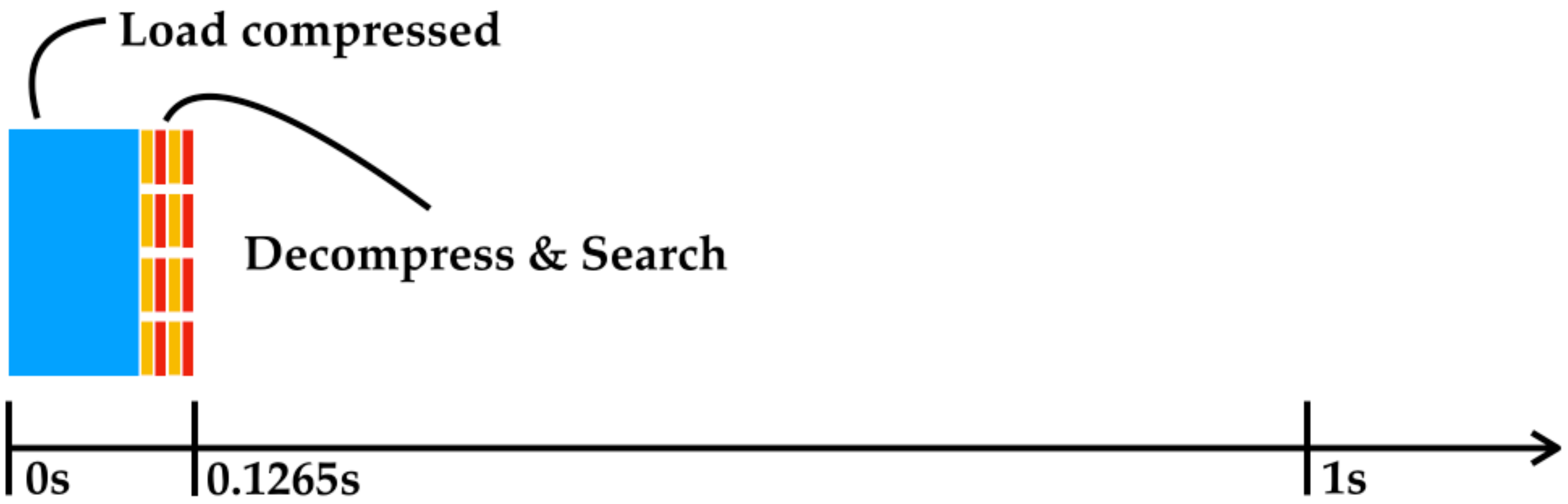
Filter 1GB data



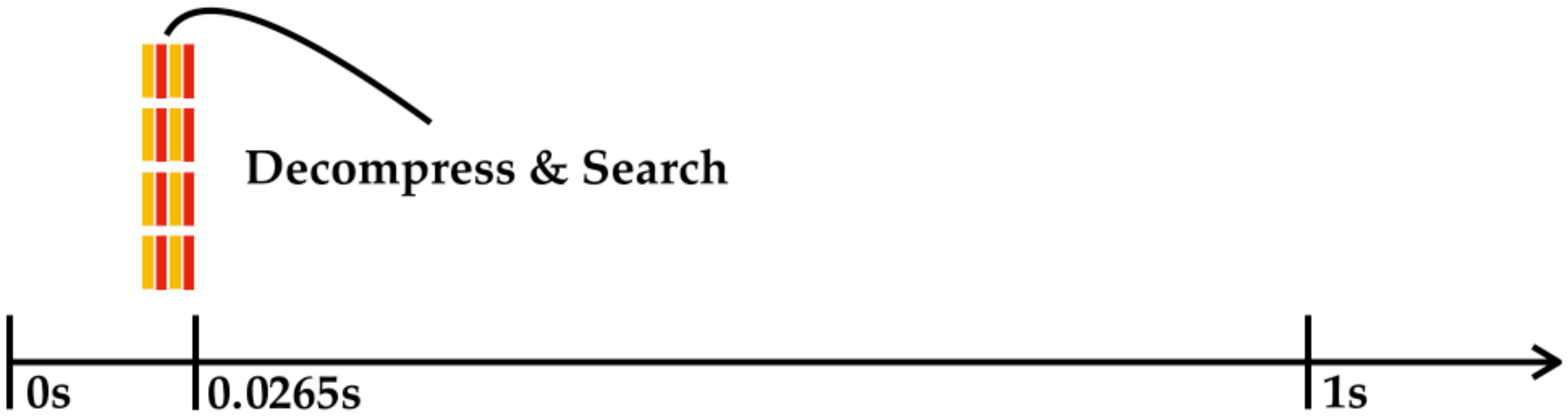
Filter 1GB data



Filter 1GB data



Filter 1GB data


Decompress & Search
0s
0.0265s
1s