Scaling near-realtime analytics with Kafka and HBase

OSCON 2012

Dave Revell & Nate Putnam

Urban Airship

About Us

- Nate Putnam
  - Team Lead, Core Data and Analytics (1 year)
  - Previously Engineer at Jive Software (4 years)
  - Contributor to HBase/Zookeeper
- Dave Revell
  - Database Engineer, Core Data and Analytics (1 year)
  - Previously Engineer at Meebo (1 year)
  - HBase contributor

In this Talk

- About Urban Airship
- Why near-realtime?
- About Kafka
- Data Consumption
- Scheduling
- HBase / High speed counting
- Questions

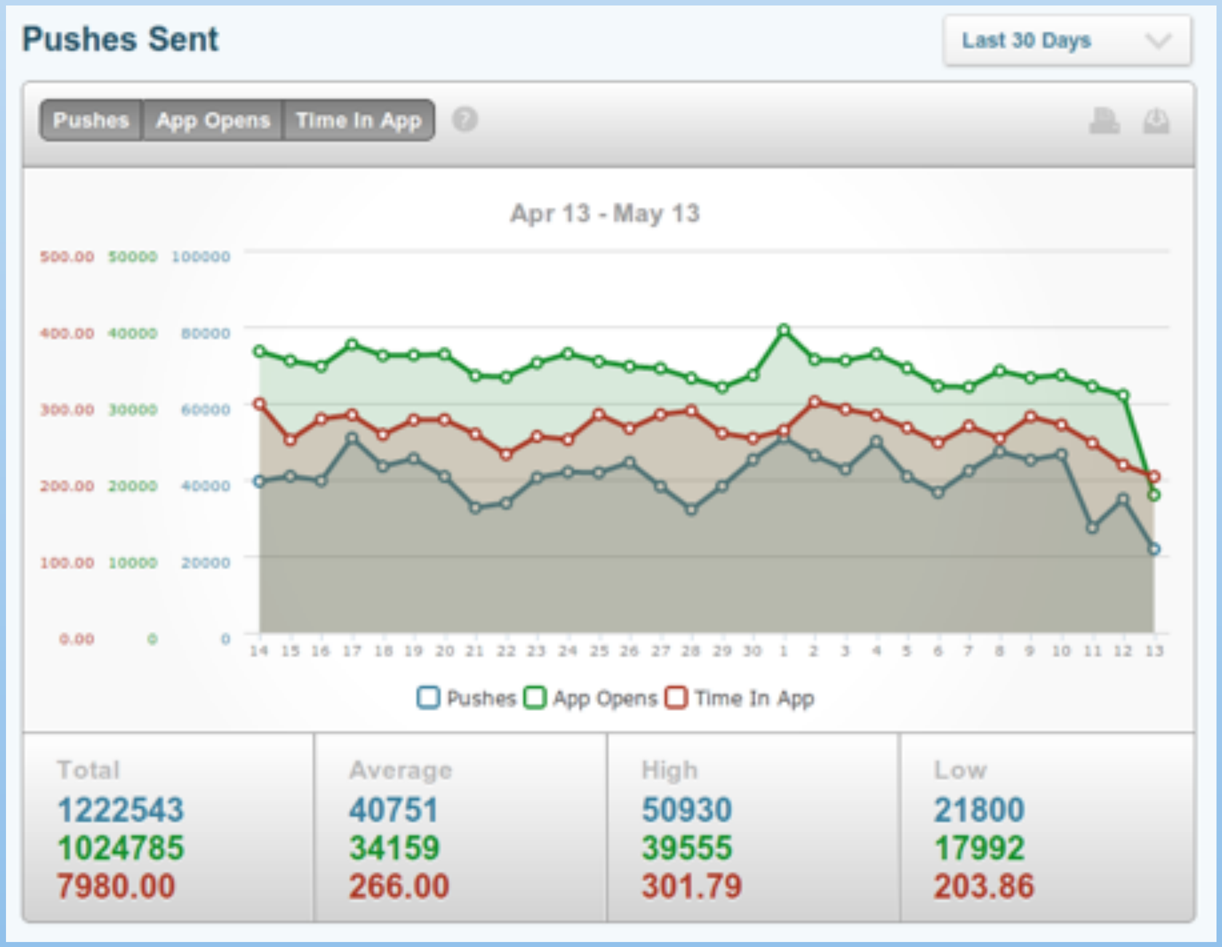
What is an Urban Airship?

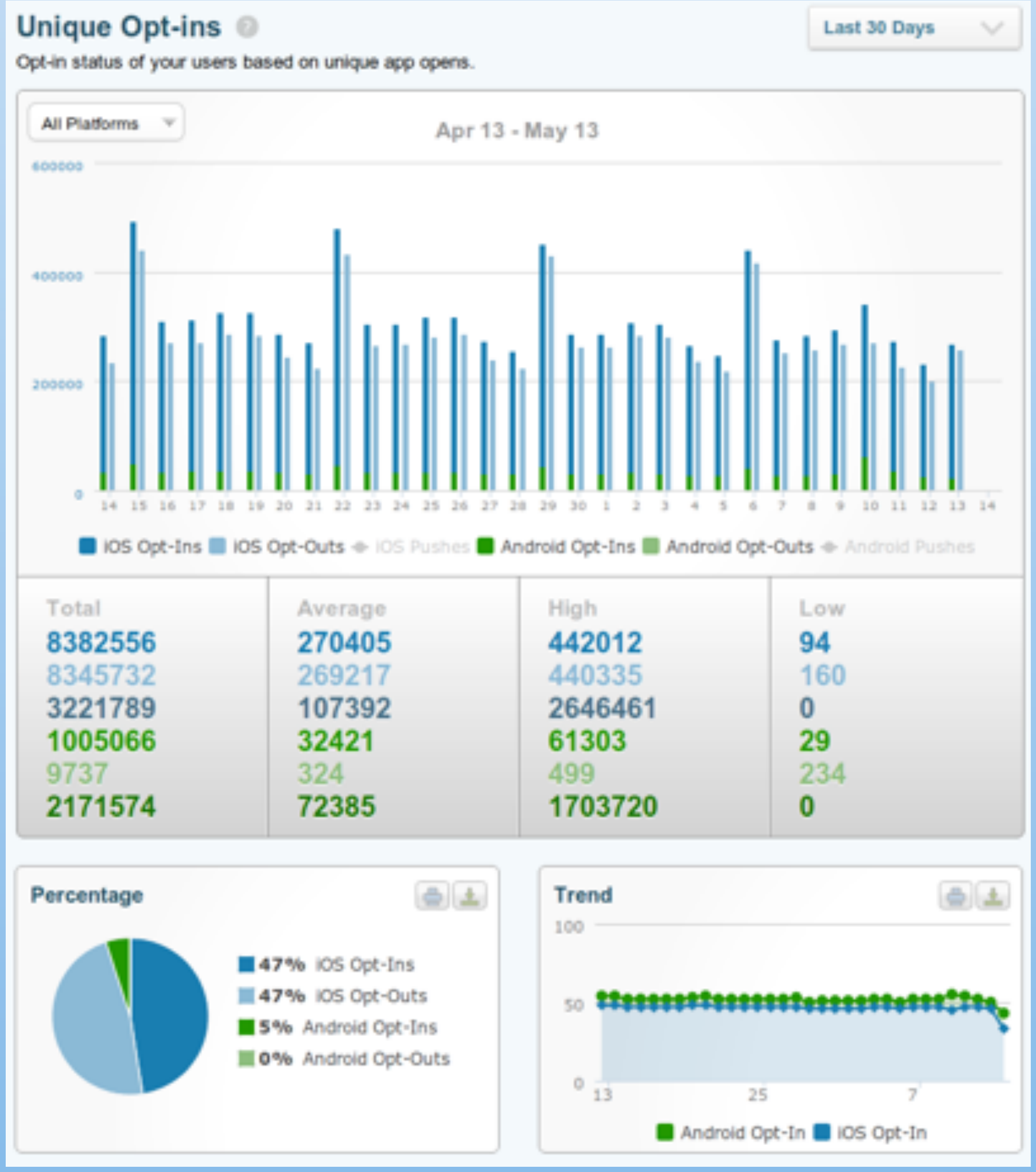
- Hosting for mobile services that developers should not build themselves
- Unified API for services across platforms
- SLAs for throughput, latency

By The Numbers

- Hundreds of millions devices
- Front end API sustains thousands of requests per second
- Millions of Android devices online all the time
- 6 months for the company to deliver 1M messages, hundred million plus a day now.

Pretty Graphs

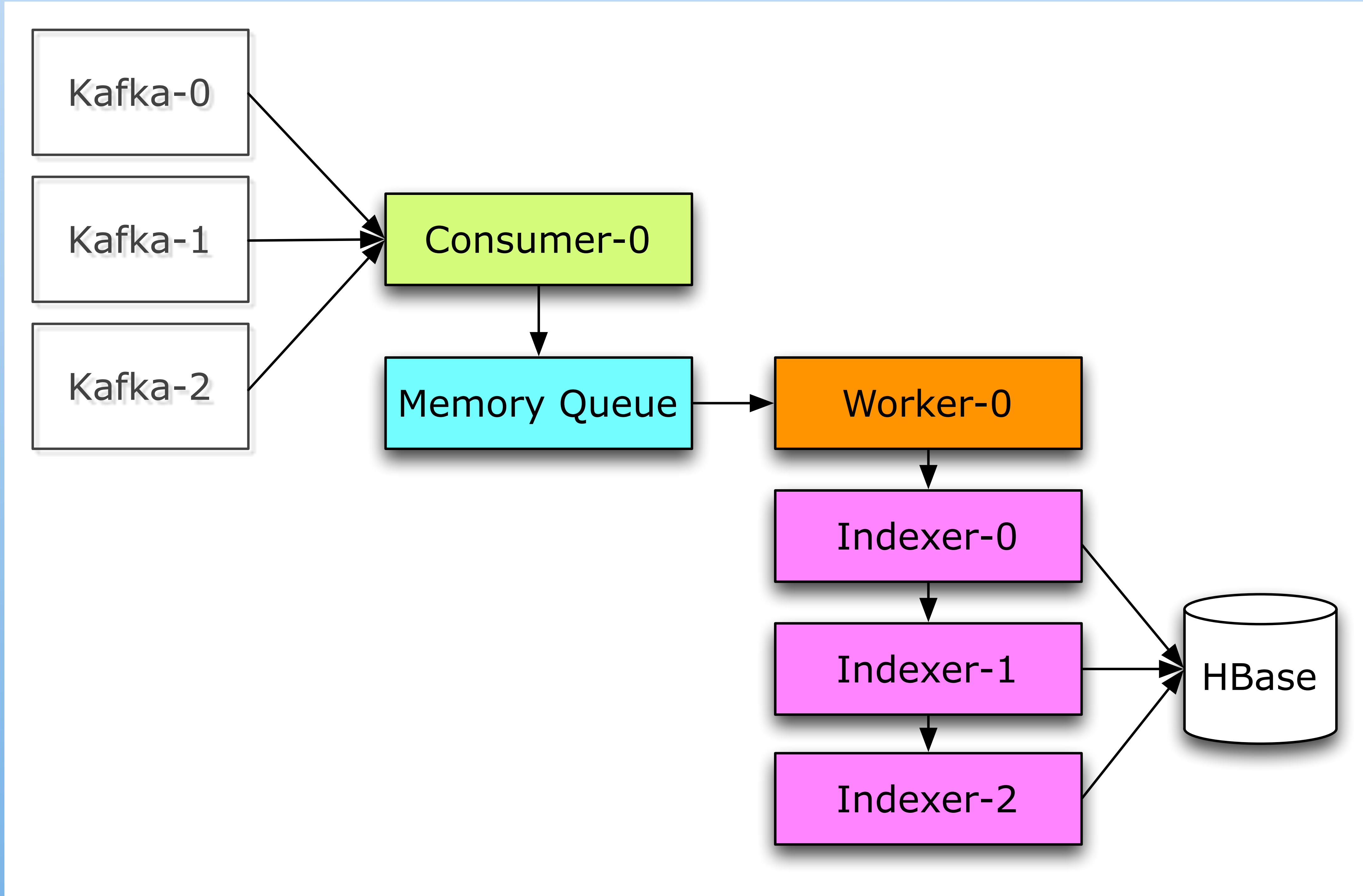




Near-Realtime?

- Realtime or Really fast?
- Events happen async
- Realtime failure scenarios are hard
- In practice a few minutes is all right for analytics

Overview

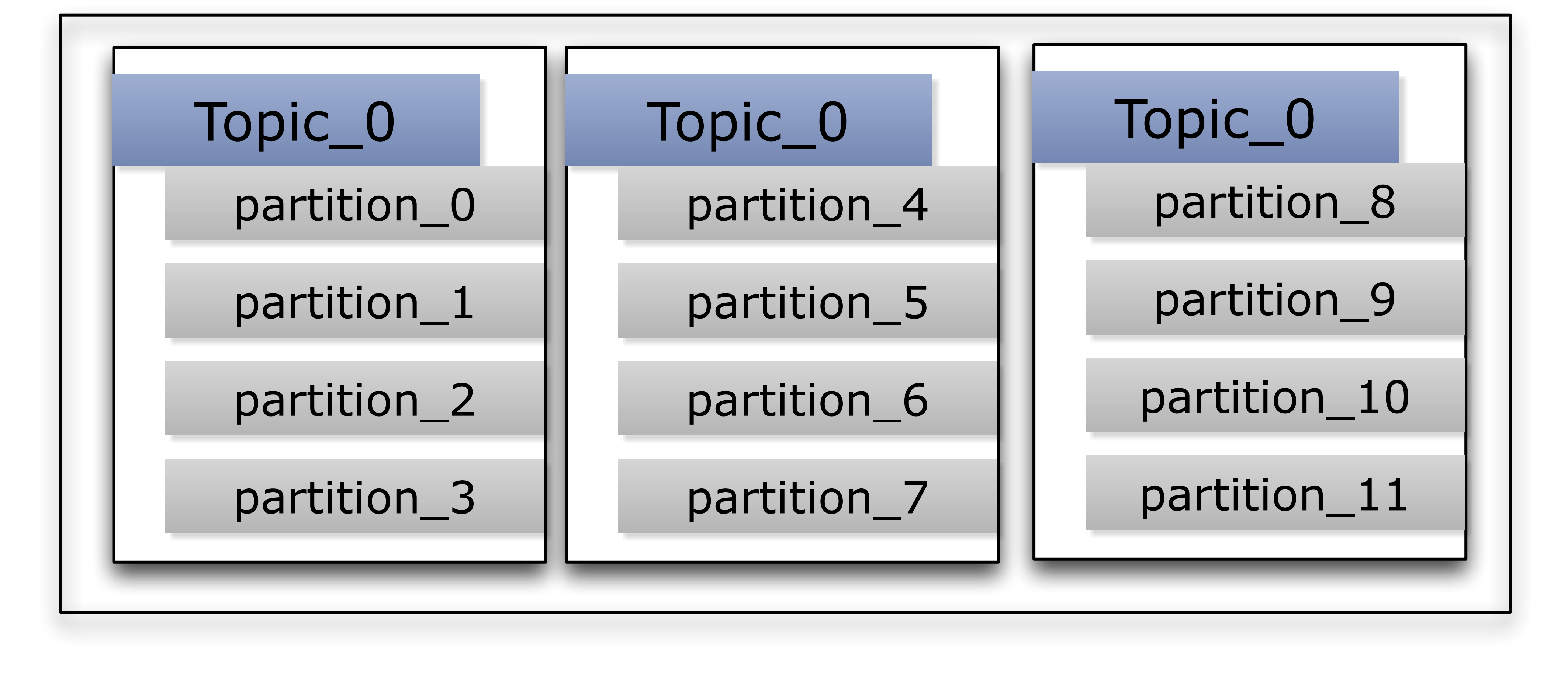

Kafka-0
Kafka-1
Kafka-2
Consumer-0
Memory Queue
Worker-0
Indexer-0
Indexer-1
Indexer-2
HBase

Kafka

- Created by the SNA Team @ LinkedIn
- Publish-Subscribe system with minimal features
- Can keep as many messages buffered as you have disk space
- Fast with some trade offs

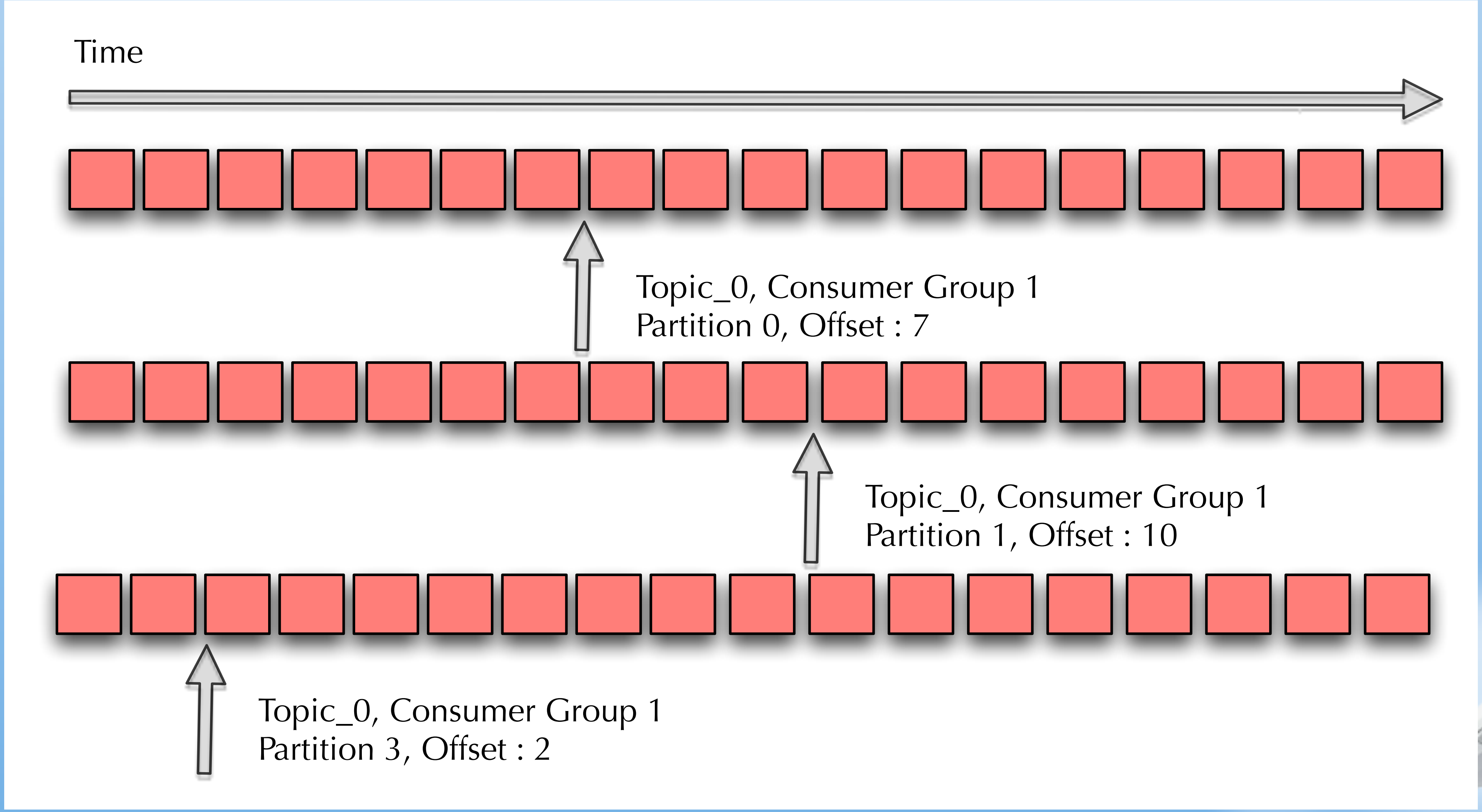
Kafka

- Topics are partitioned across servers
- Partitioning scheme is customizable



Kafka

- Consumption is 1 thread per distinct partition
- Consumers keep track of their own offsets



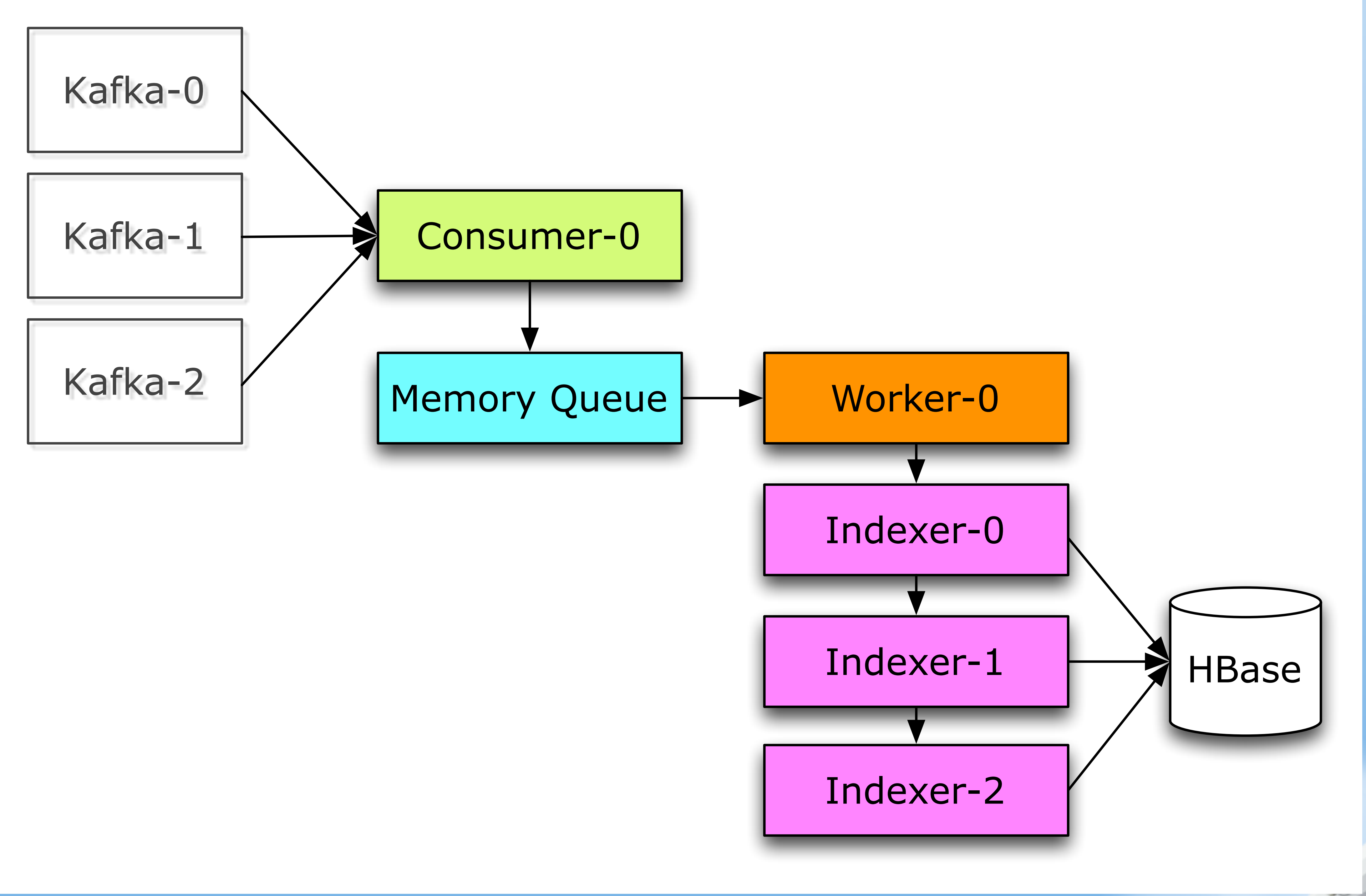
Kafka

- Manipulation of time based indexes is powerful
- Monitor offsets and lag
- Throw as much disk at this as you can
- http://incubator.apache.org/kafka/

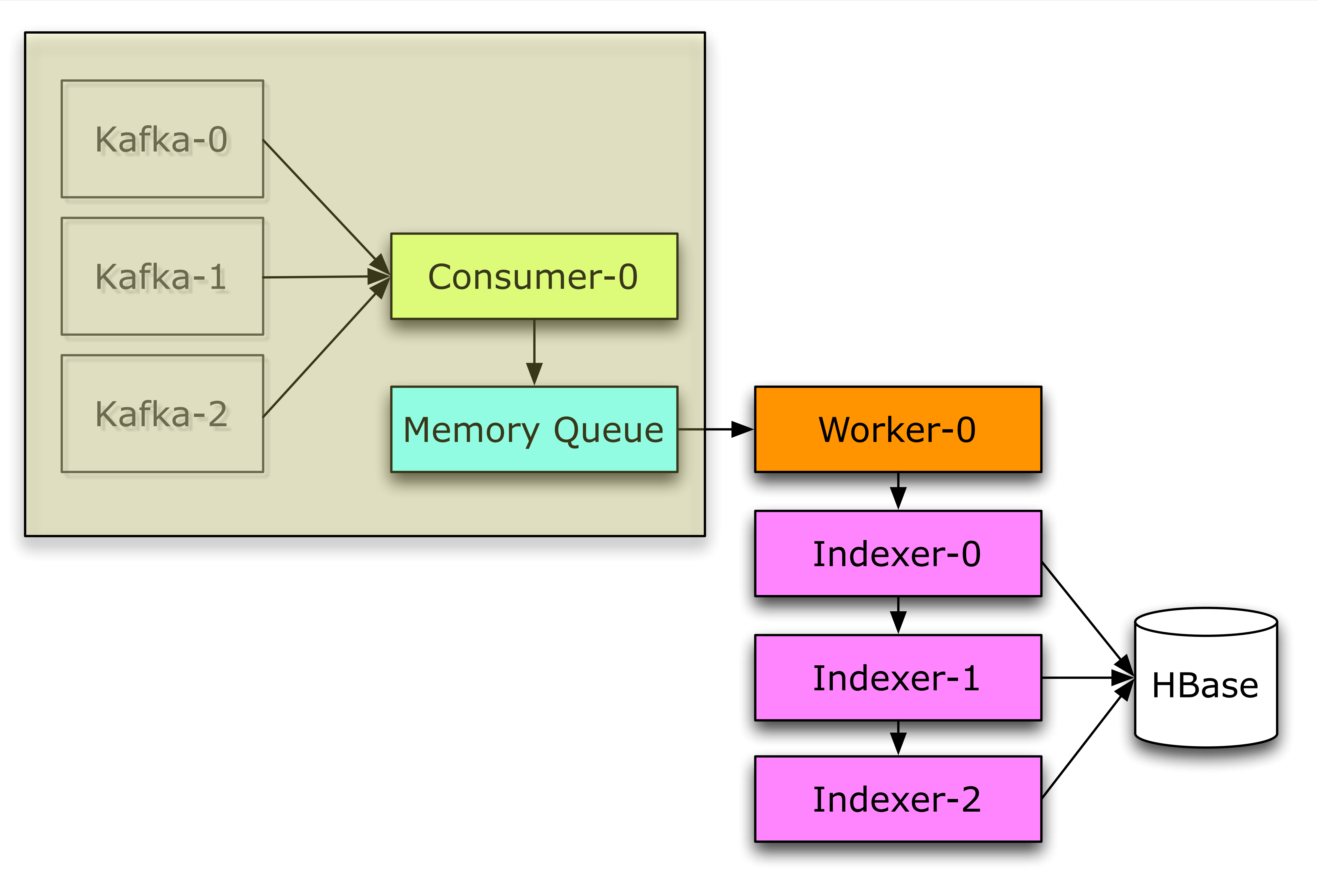
Consumers

- Mirror Kafka design
- Lots of parallelism to increase throughput
- Share nothing
- No ordering guarantees

Consumers

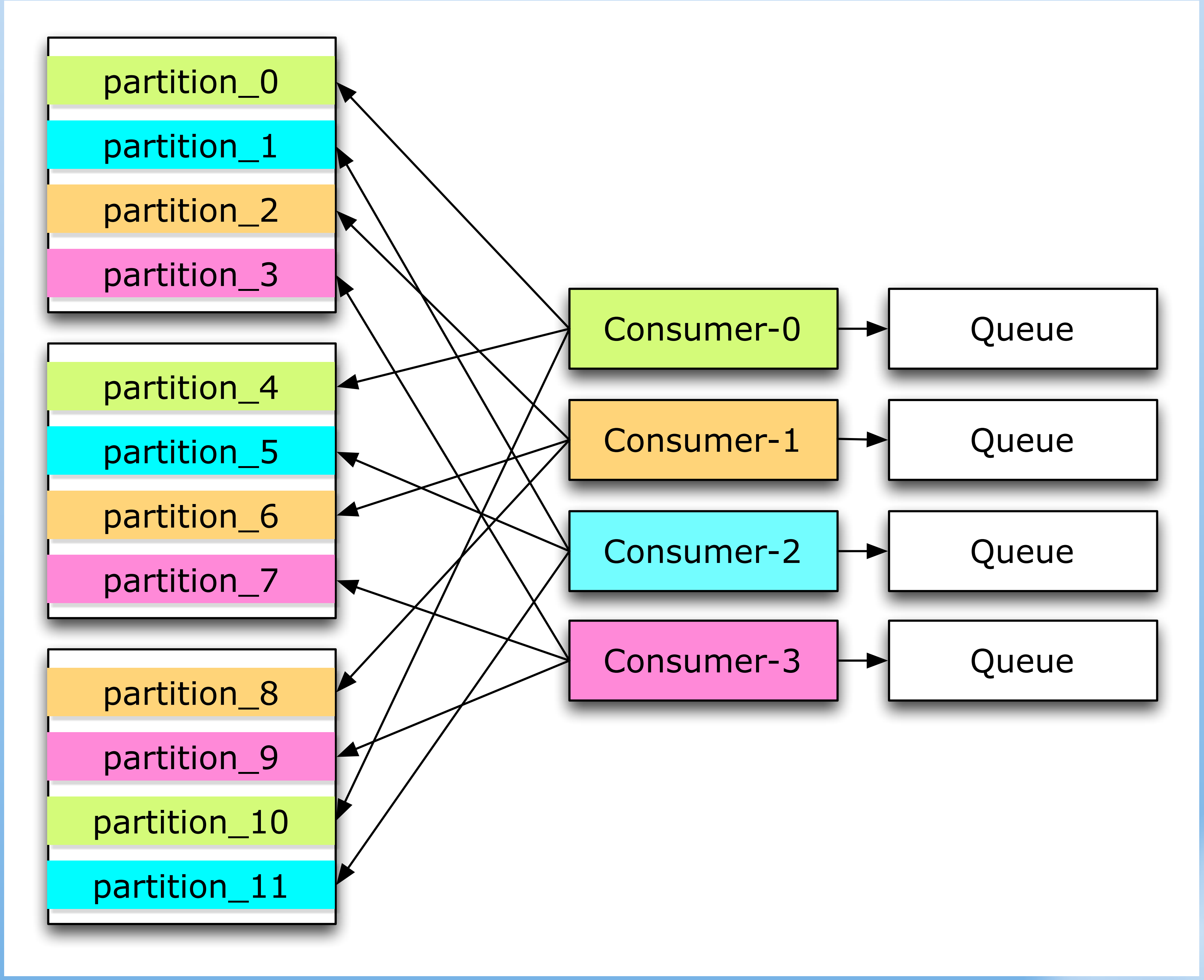

Kafka-0
Kafka-1
Kafka-2
Consumer-0
Memory Queue
Worker-0
Indexer-0
Indexer-1
Indexer-2
HBase

Consumers

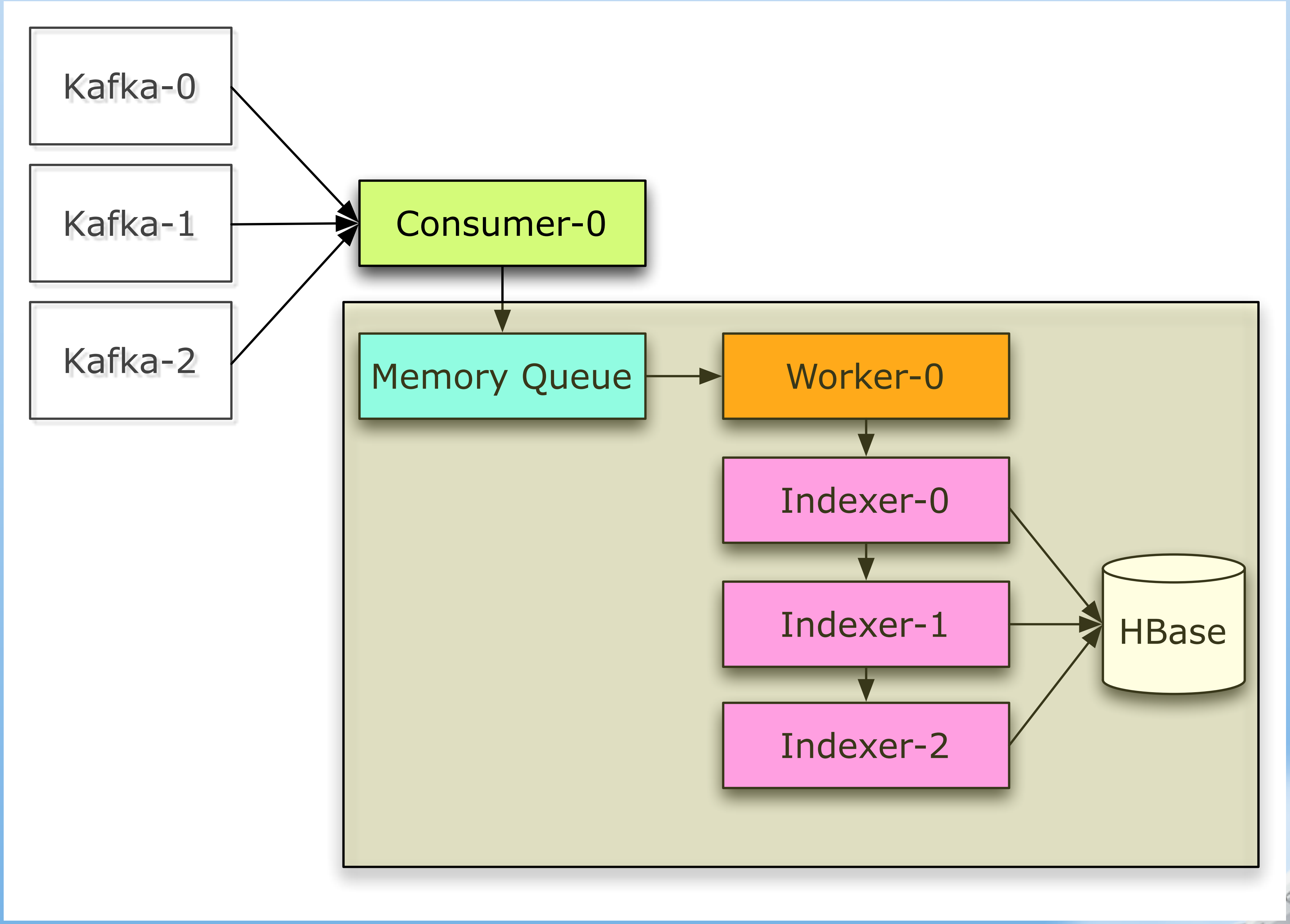


Kafka-0
Kafka-1
Kafka-2
Consumer-0
Memory Queue
Worker-0
Indexer-0
Indexer-1
Indexer-2
HBase

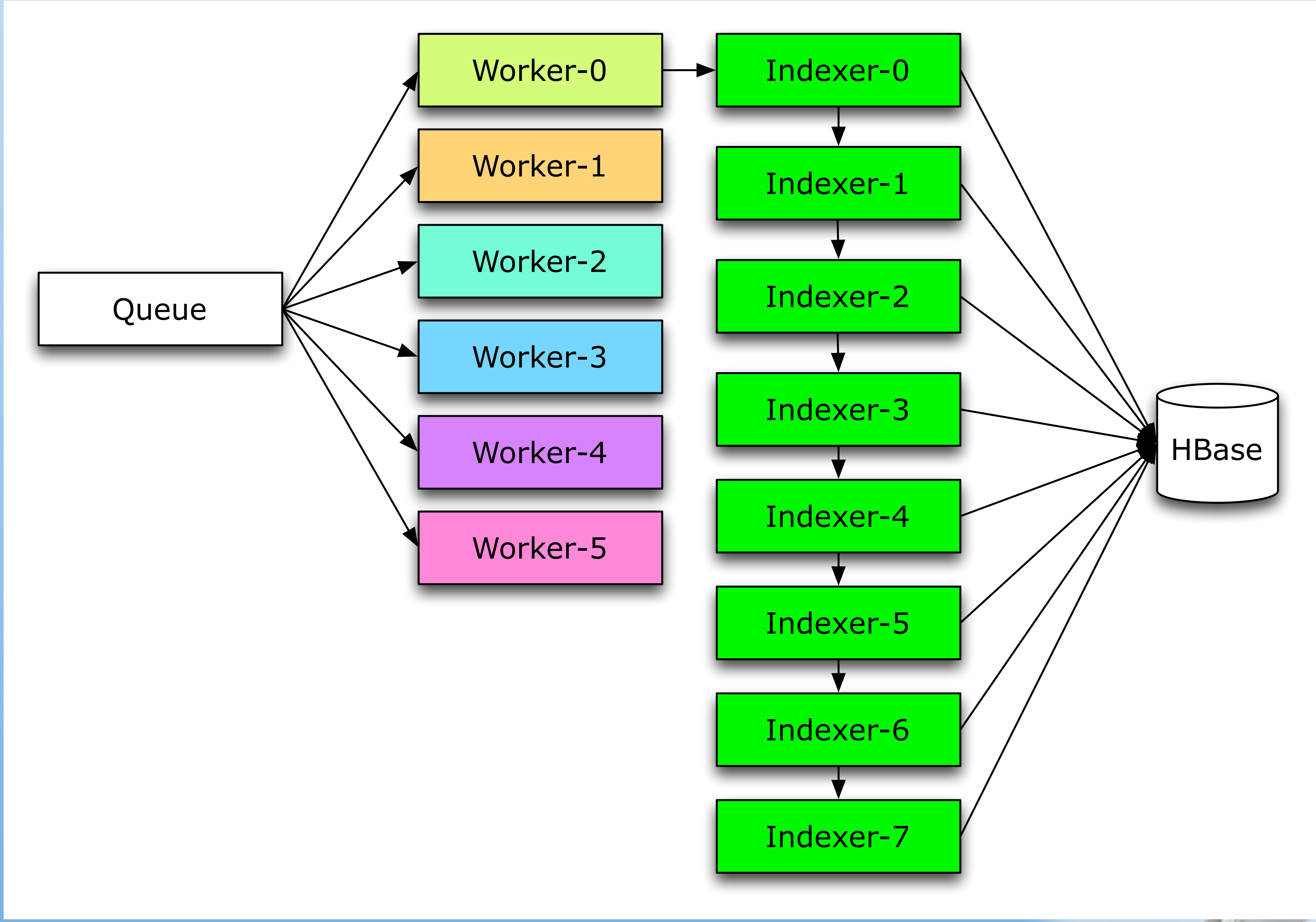
Consumers


partition_0
partition_1
partition_2
partition_3
partition_4
partition_5
partition_6
partition_7
partition_8
partition_9
partition_10
partition_11
Consumer-0
Consumer-1
Consumer-2
Consumer-3
Queue
Queue
Queue
Queue

Consumers


Kafka-0
Kafka-1
Kafka-2
Consumer-0
Memory Queue
Worker-0
Indexer-0
Indexer-1
Indexer-2
HBase

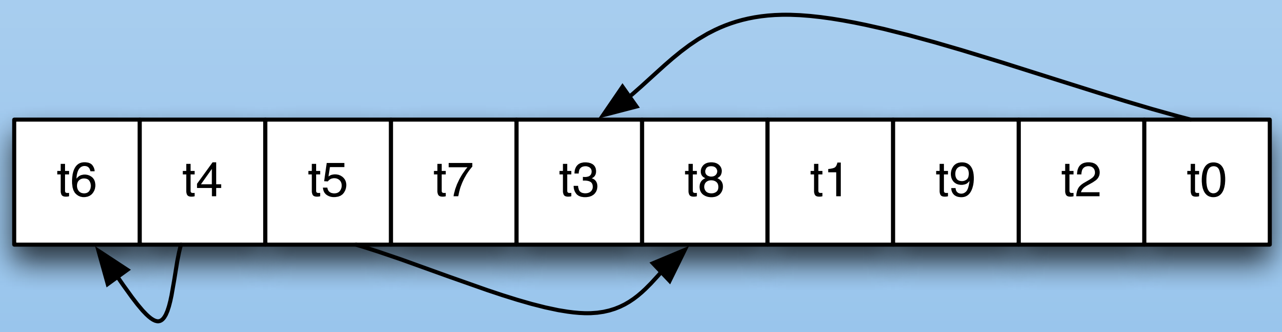
Consumers

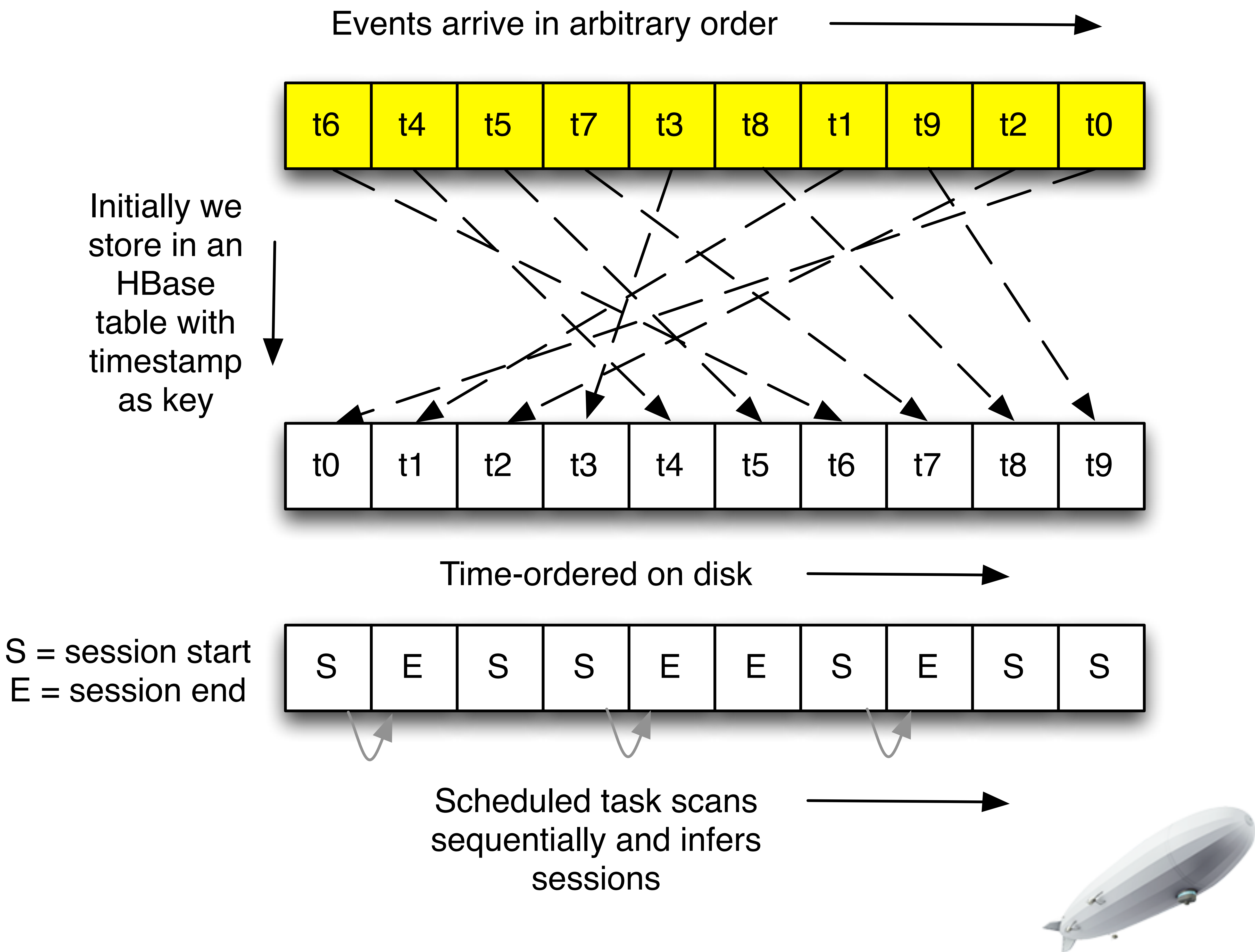


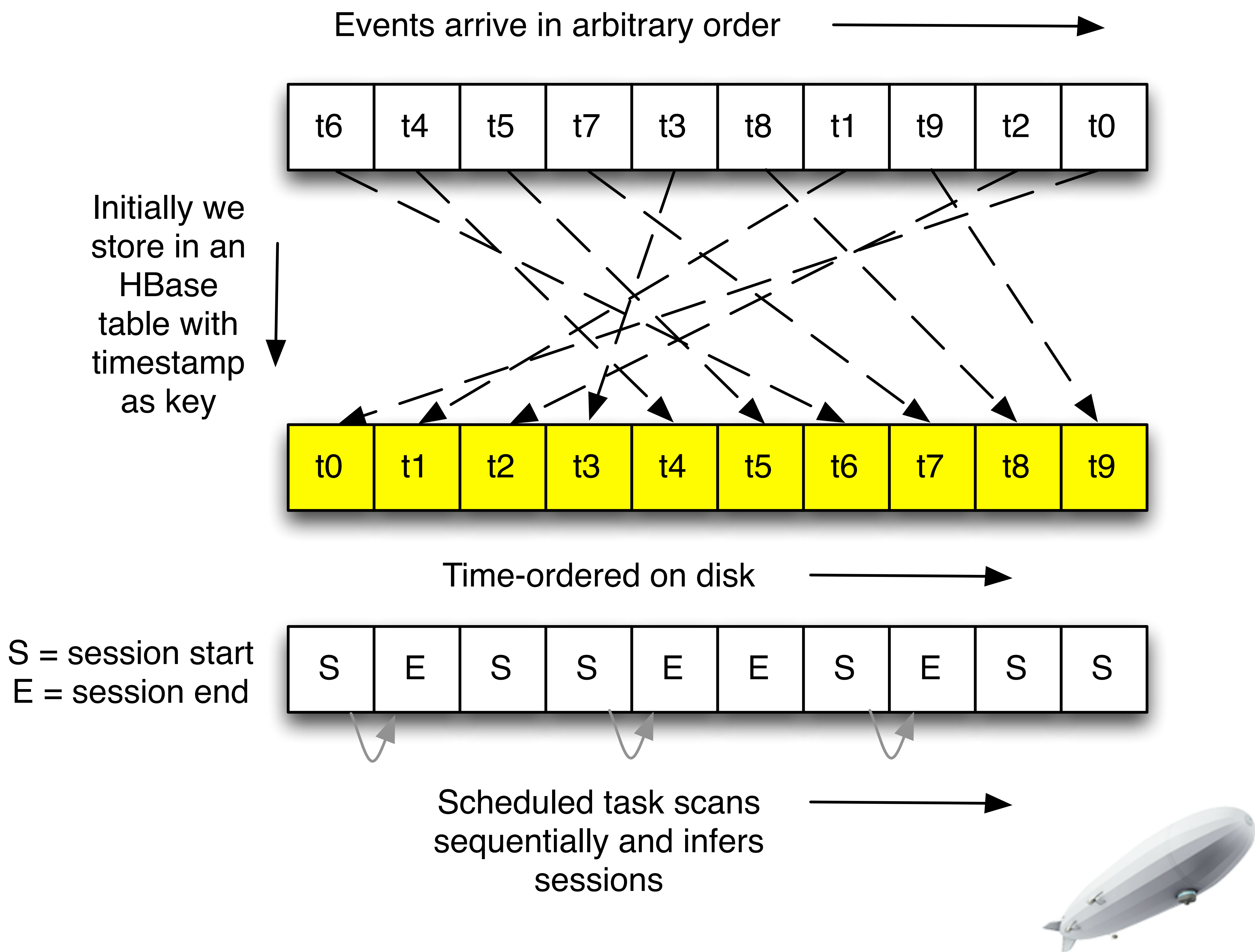
Queue
Worker-0
Worker-1
Worker-2
Worker-3
Worker-4
Worker-5
Indexer-0
Indexer-1
Indexer-2
Indexer-3
Indexer-4
Indexer-5
Indexer-6
Indexer-7
HBase

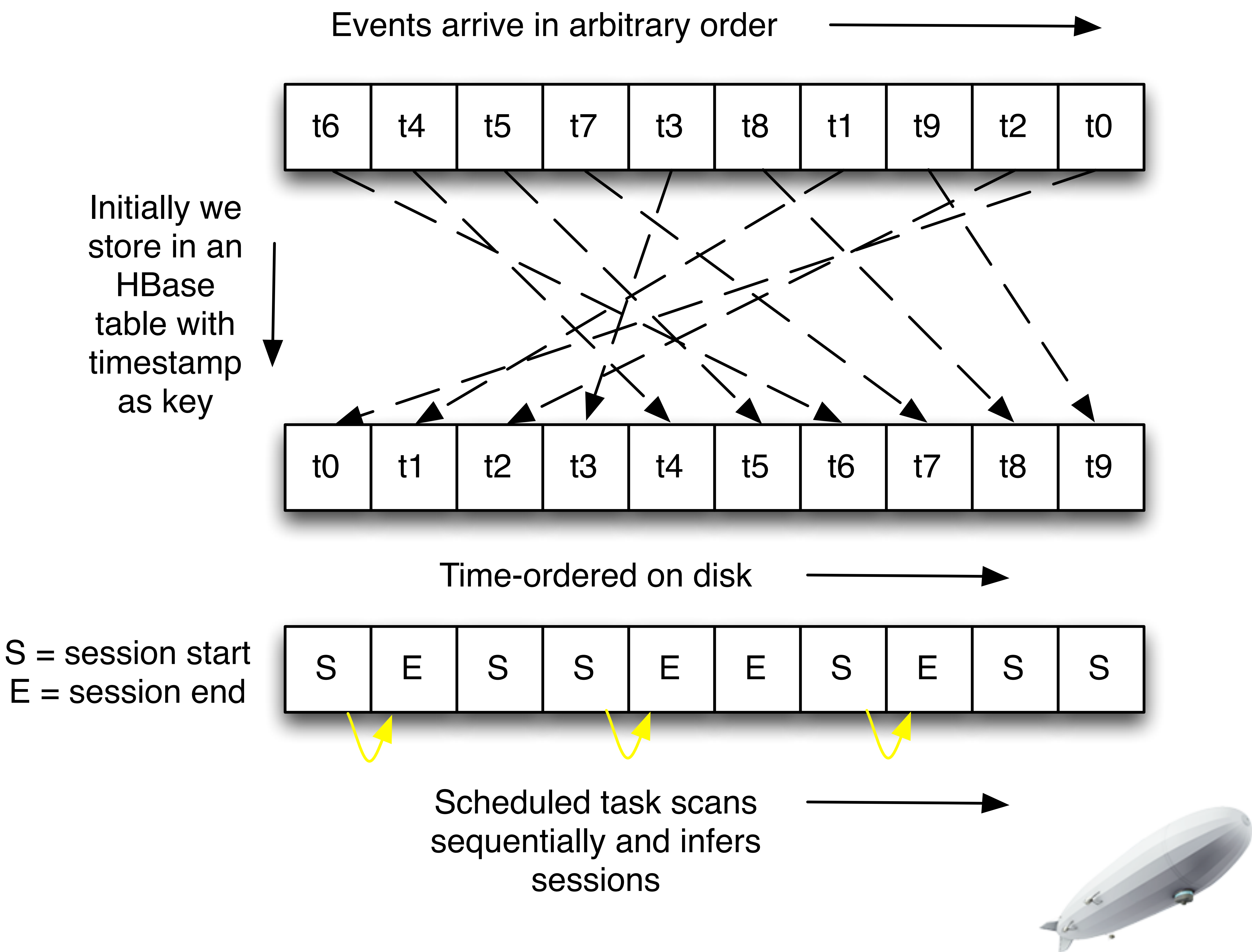
Scheduled aggregation tasks

- Challenge: aggregate values that arrive out of order
- Example: sessions/clickstream
- Two steps:
  - Quickly write into HBase
  - Periodically scan to calculate aggregates




Events arrive in arbitrary order
t6
t4
t5
t7
t3
t8
t1
t9
t2
t0
Initially we store in an HBase table with timestamp as key
t0
t1
t2
t3
t4
t5
t6
t7
t8
t9
Time-ordered on disk
S = session start
E = session end
S
E
S
S
E
E
S
E
S
S
Scheduled task scans sequentially and infers sessions


Events arrive in arbitrary order
t6 t4 t5 t7 t3 t8 t1 t9 t2 t0
Initially we store in an HBase table with timestamp as key
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9
Time-ordered on disk
S = session start
E = session end
S E S S E E S E S S
Scheduled task scans sequentially and infers sessions

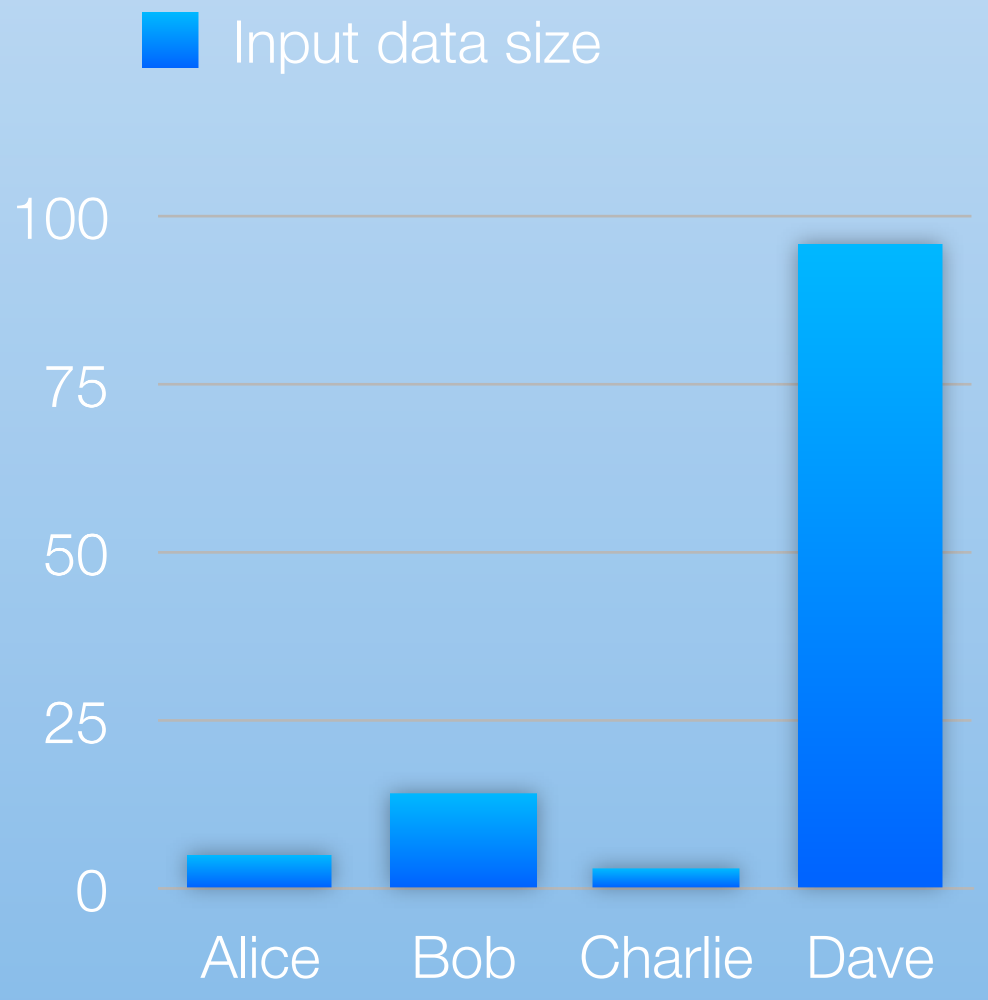

Events arrive in arbitrary order
t6
t4
t5
t7
t3
t8
t1
t9
t2
t0
Initially we store in an HBase table with timestamp as key
t0
t1
t2
t3
t4
t5
t6
t7
t8
t9
Time-ordered on disk
S = session start
E = session end
S
E
S
S
E
E
S
E
S
S
Scheduled task scans sequentially and infers sessions

Strengths

- Efficient with disk and memory
- Can tradeoff response time for disk usage
- Fine granularity, 10Ks of jobs

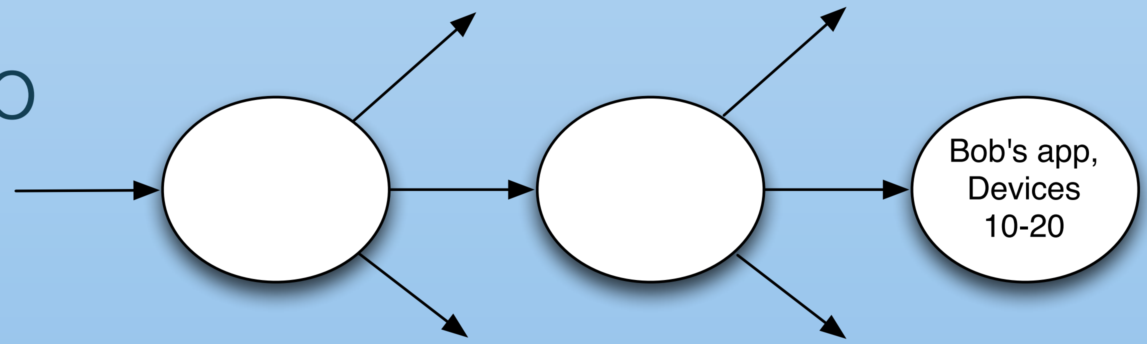
Compared to MapReduce

- Similar to MapReduce shuffle: sequential IO, external sort
- Fine grained failures, scheduling, resource allocation
- Can't do lots of jobs, can't do big jobs
- But MapReduce is easier to use



Pro/con vs. realtime streaming

- For example, a Storm topology
- Challenge: avoid random reads (disk seeks) without keeping too much state in RAM
- Sorting minimizes state
- But latency would be good



HBase

- What it is
- Why it's good for low-latency big data

HBase

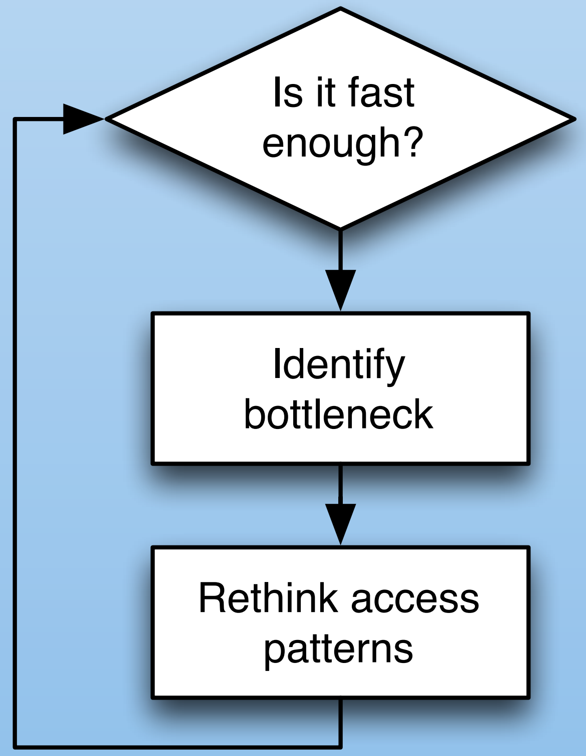
- A database that uses HDFS for storage
- Based on Google's BigTable
- Solves the problem "how do I query my Hadoop data?"
    - Operations typically measured in milliseconds
    - MapReduce is not suitable for real time queries
- Scales well by adding servers (if you do everything right)
- Not partition tolerant or eventually consistent

Why we like HBase

- Scalable
- Fast: millions of ops/sec
- Open source, ASF top-level project
- Strong community

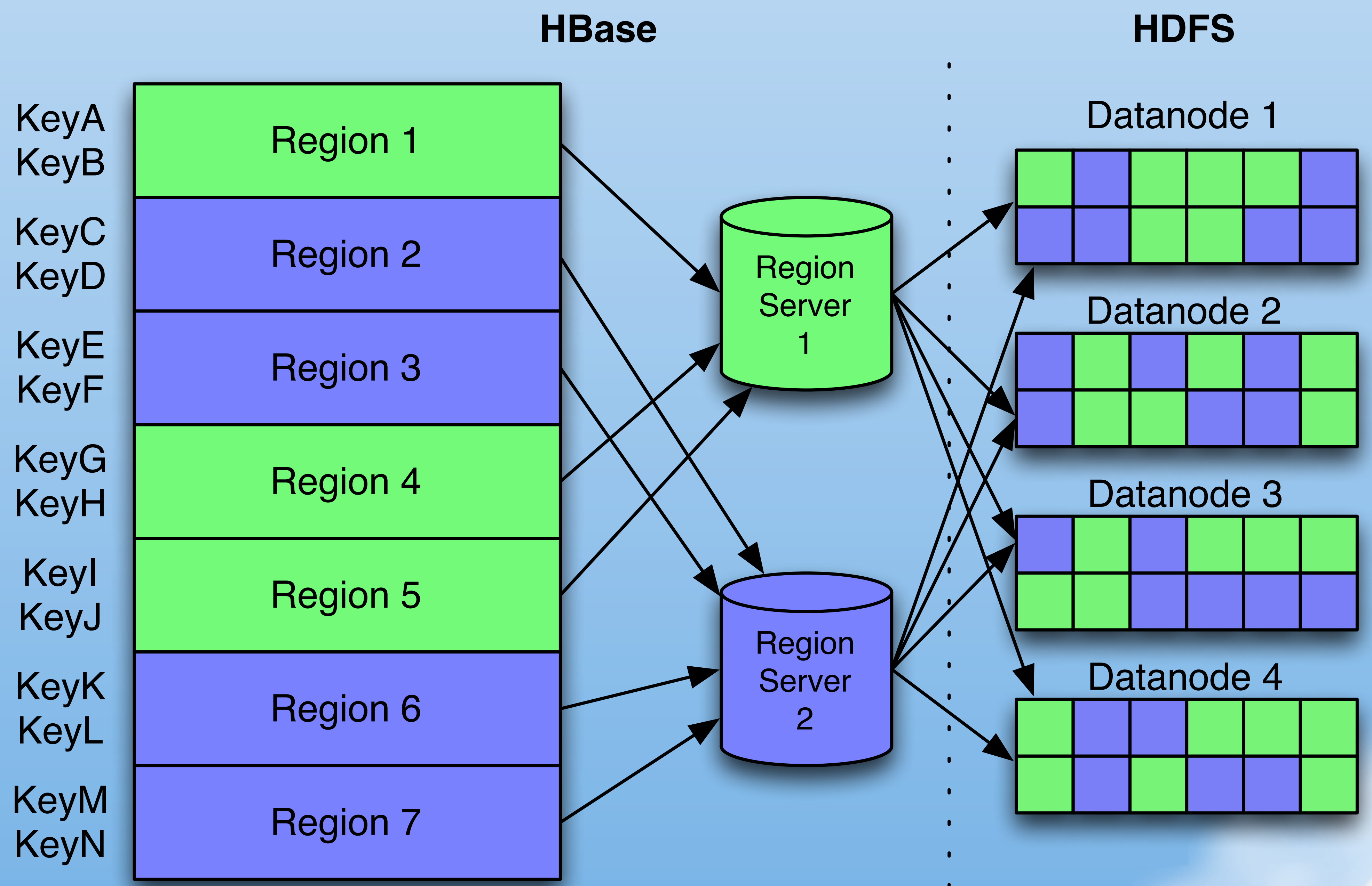
HBase difficulties

- Low level features, harder to use than RDBMS
- Hard to avoid accidentally introducing bottlenecks
- Garbage collection, JVM tuning
- HDFS



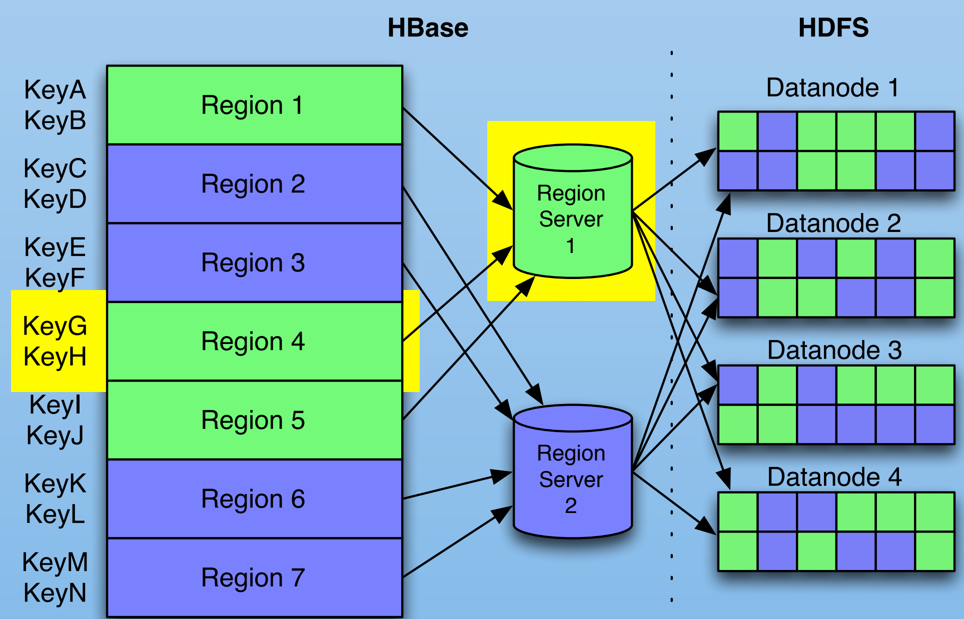
How to fail at HBase

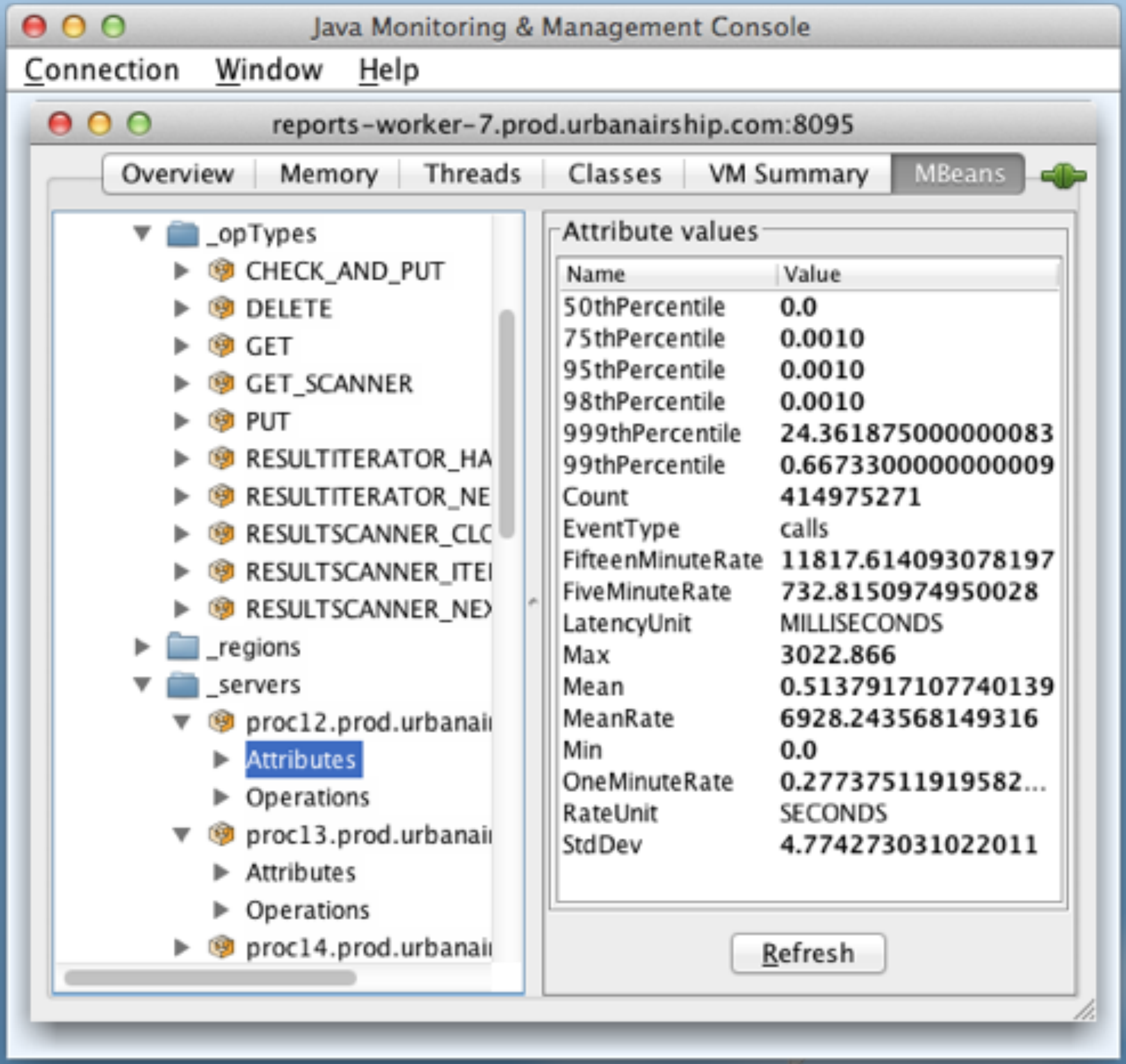
- Schema can limit scalability



Troubleshooting

- Isolate slow regions or servers with statshtable
  - http://github.com/urbanairship/statshtable





Counting

- The main thing that we do
- Scaling dimensions:
  - Many counters of interest per event
  - Many events
  - Many changes to counter definitions

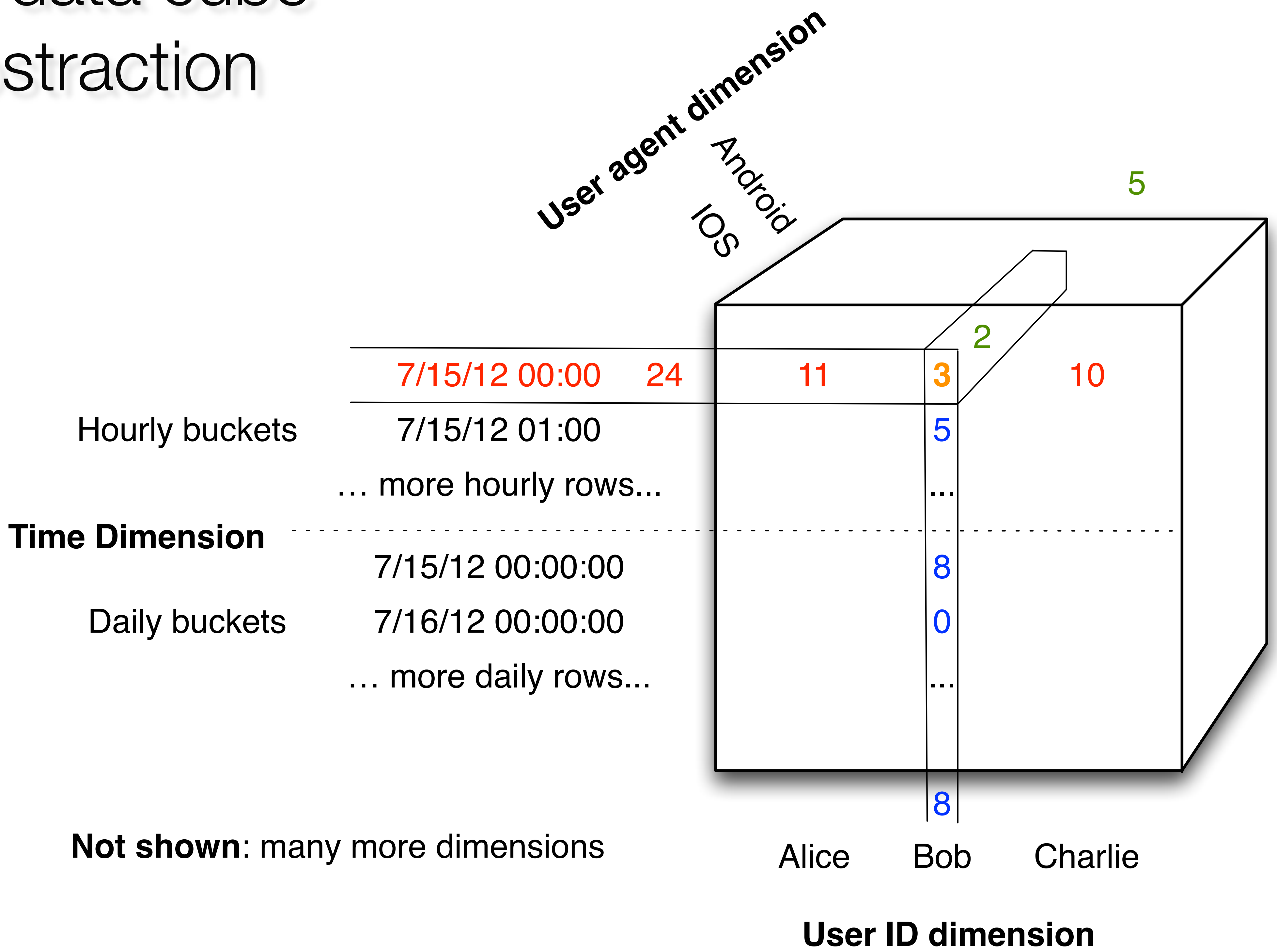
A naive attempt

```
for event in stream:
  user_id = extract_user_id(event)
  timestamp = extract_timestamp(event)
  event_type = extract_event_type(event)
  client_type = extract_client_type(event)
  location = extract_location(event)

  increment_event_type_count(event_type)
  increment_client_and_event_type_count(event_type, client_type)
  increment_user_id_and_event_type_count(user_id, event_type)
  increment_user_id_and_client_type_count(user_id, client_type)

  for time_precision in {HOURLY, DAILY, MONTHLY}:
    increment_time_count(time_precision, timestamp)
    increment_time_client_type_event_type_count(time_precision, ..)
    increment_(...)

    for location_precision in {CITY, STATE, COUNTRY}:
      increment_time_location_event_type_client_type_user_id(...)

      for bucket in yet_another_dimension: ....
```

Counting with datacubes

- Challenge: count items in a stream matching various criteria, when criteria may change
- github.com/urbanairship/datacube
- A Java library for turning streams into OLAP cubes
  - Especially multidimensional counters

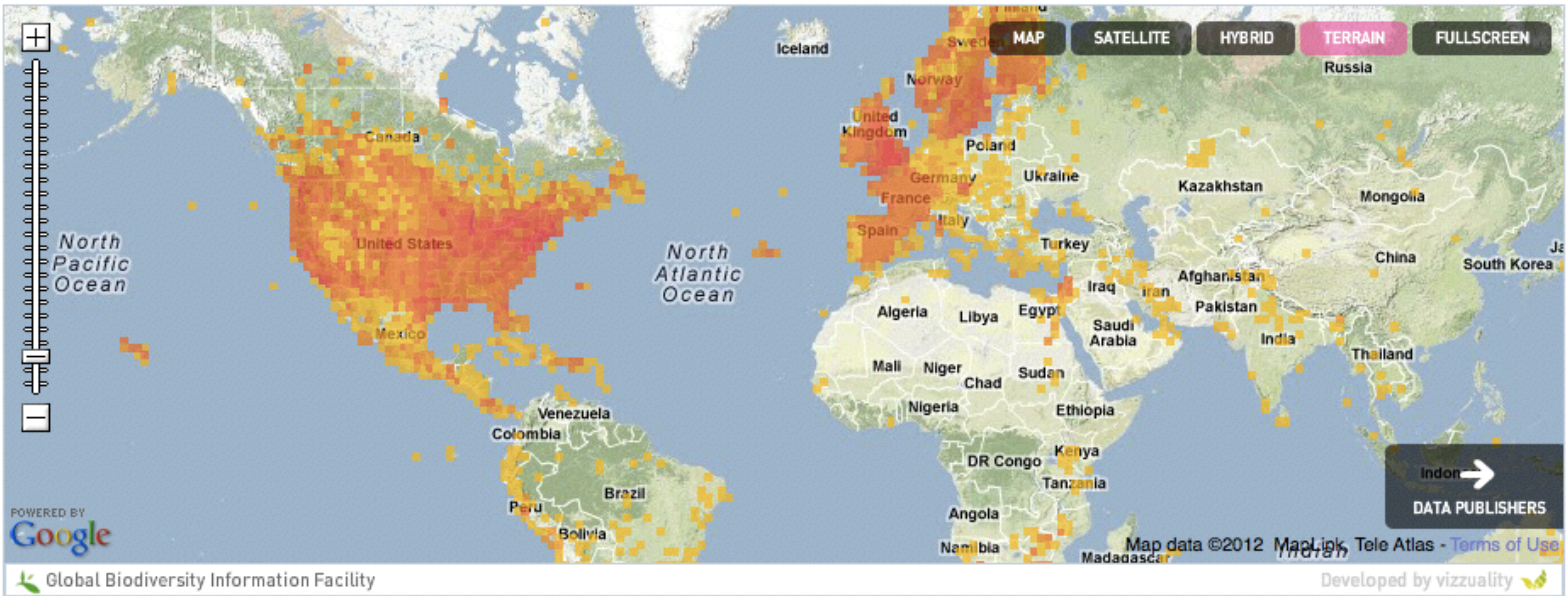
The data cube abstraction



Why datacube?

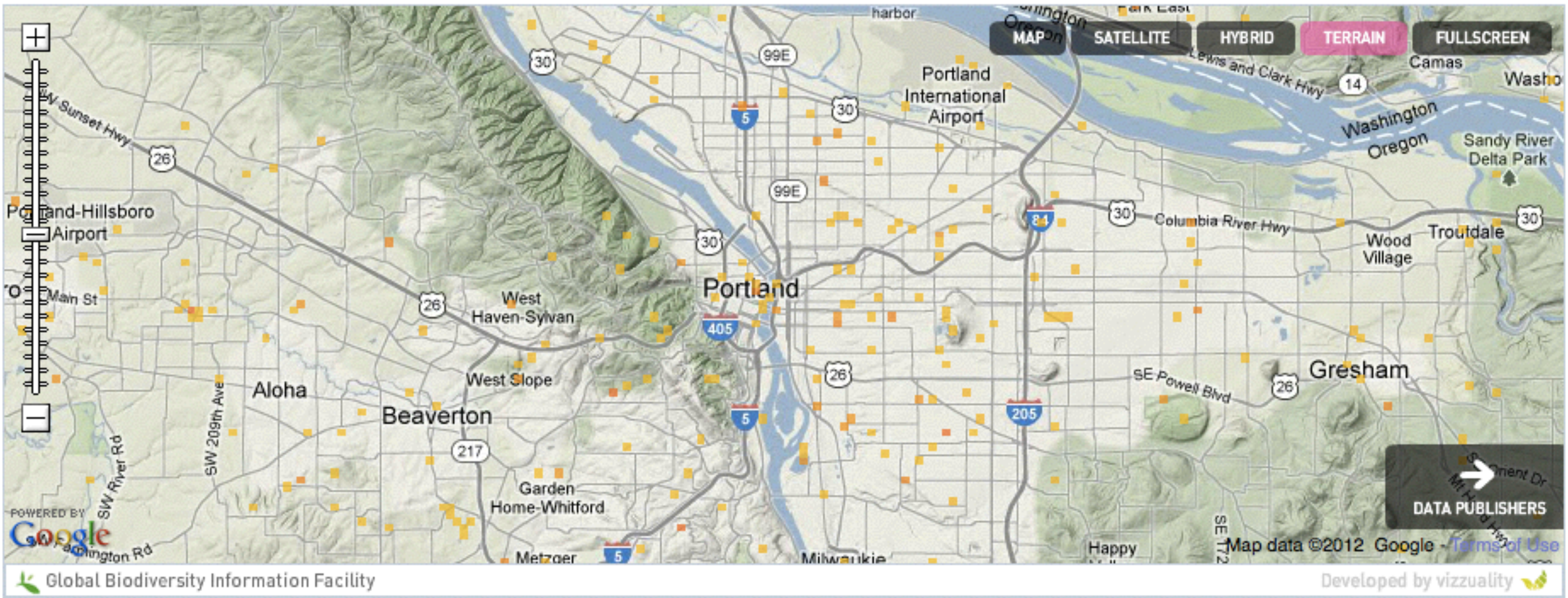
- Handles exponential number of writes
- Async IO with batching
- Declarative interface: say what to count, not how
- Pluggable database backend (currently HBase)
- Bulk loader
- Easily change/extend online cubes

Datacube isn't just for counters



Courtesy Tim Robertson, GBIF

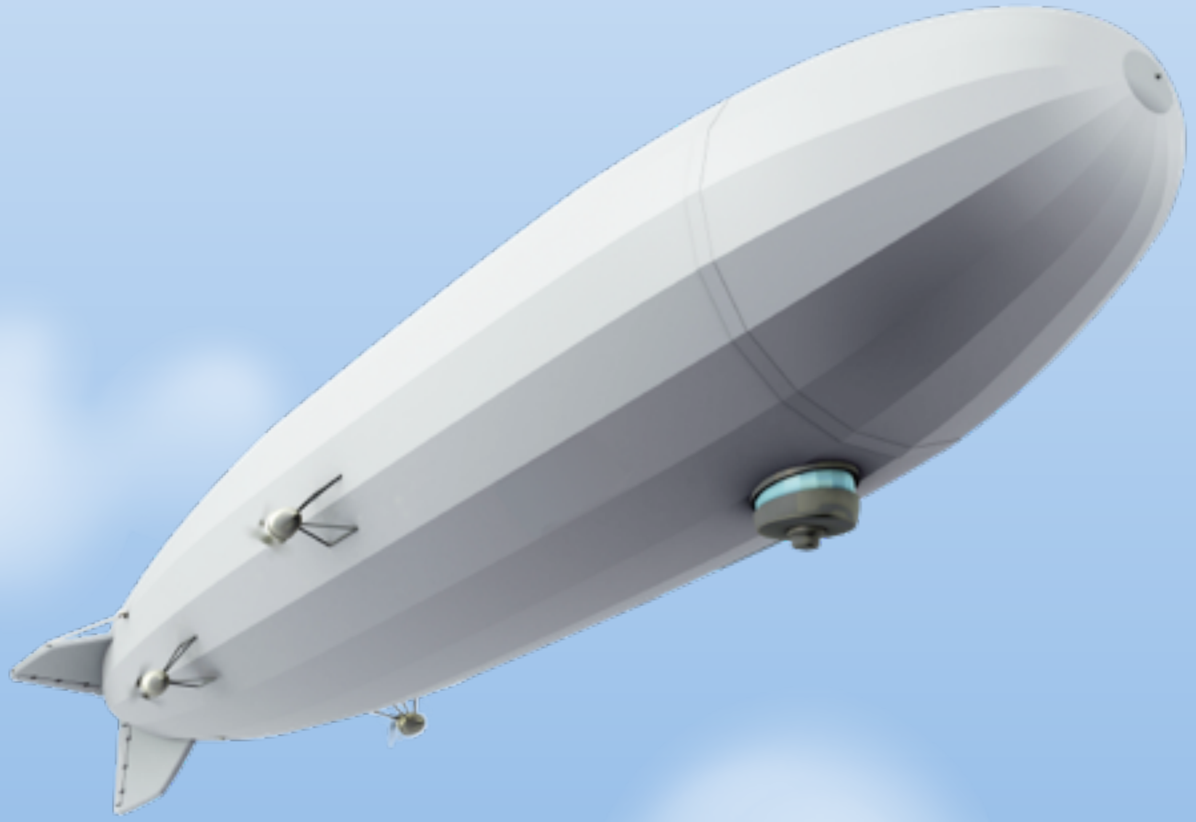
github.com/urbanairship/datacube



Courtesy Tim Robertson, GBIF

github.com/urbanairship/datacube

Questions?



Thanks!

- HBase and Kafka for being awesome
- We're hiring! urbanairship.com/jobs/
- @nateputnam @dave_revell