



Stream All the Things!

Architectures for Data that Never Ends

Dean Wampler, Ph.D.
dean@deanwampler.com
[@deanwampler](https://twitter.com/deanwampler)

Free as in 

lightbend.com/fast-data-platform
(2nd Edition coming soon!)

O'REILLY®

Fast Data Architectures for Streaming Applications

Getting Answers Now from
Data Sets that Never End

Dean Wampler

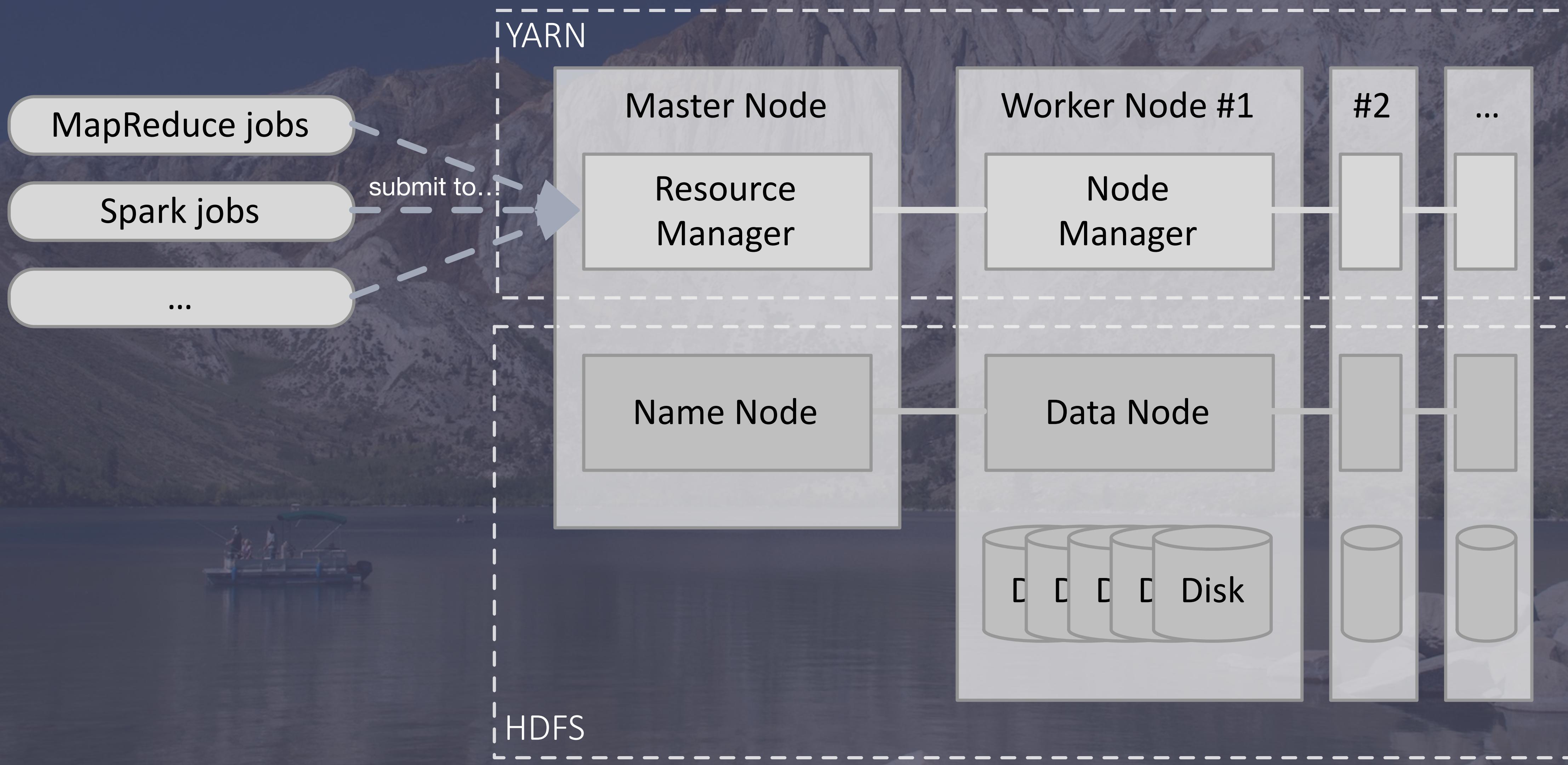
Compliments of
 Lightbend

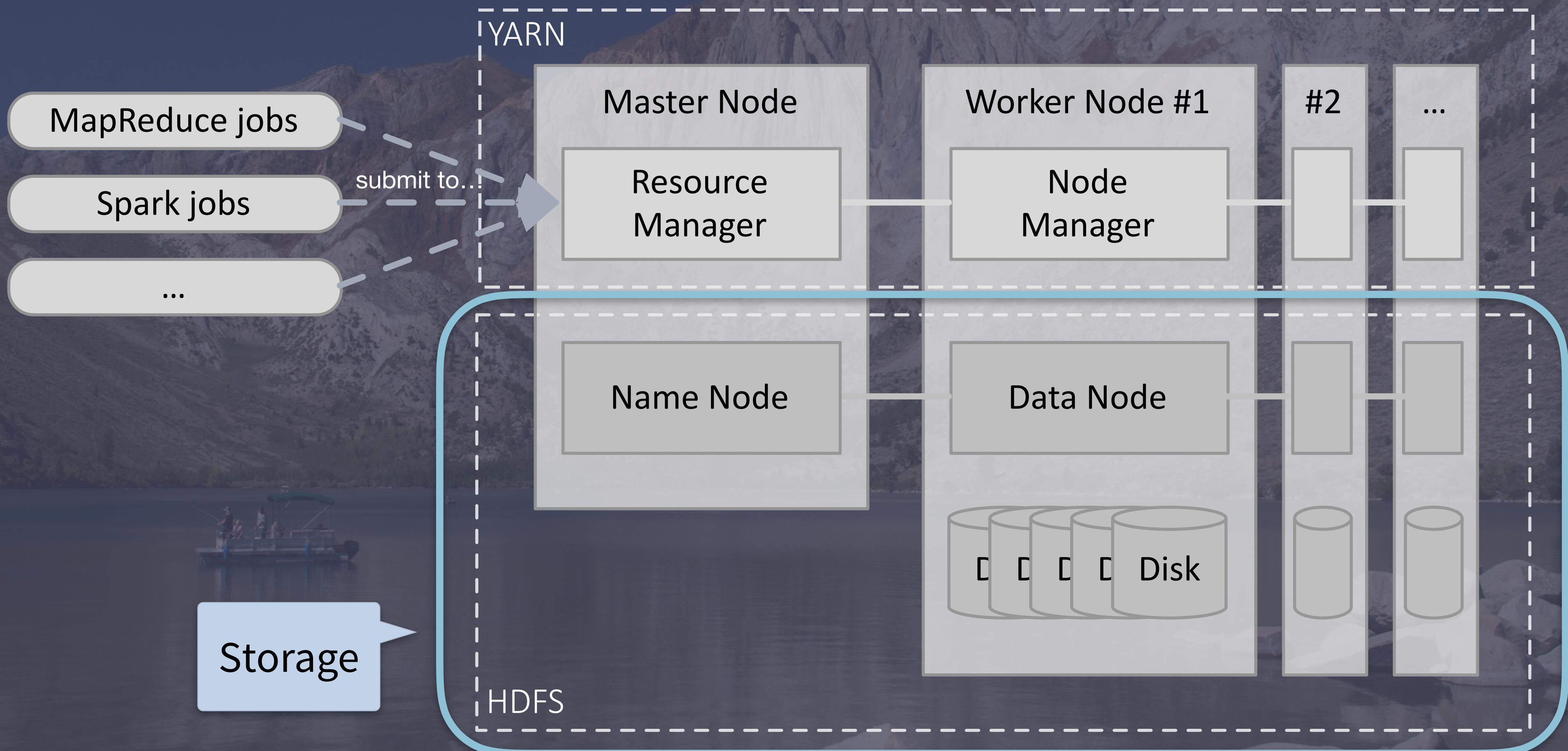
An aerial photograph showing a vast area of agricultural land. The fields are organized into a grid-like pattern, with various shades of green and brown indicating different crops or stages of cultivation. A network of blue lines, representing canals or rivers, cuts through the fields, creating a complex web of waterways. The perspective is from above, looking down at the agricultural landscape.

Streaming
in Context...

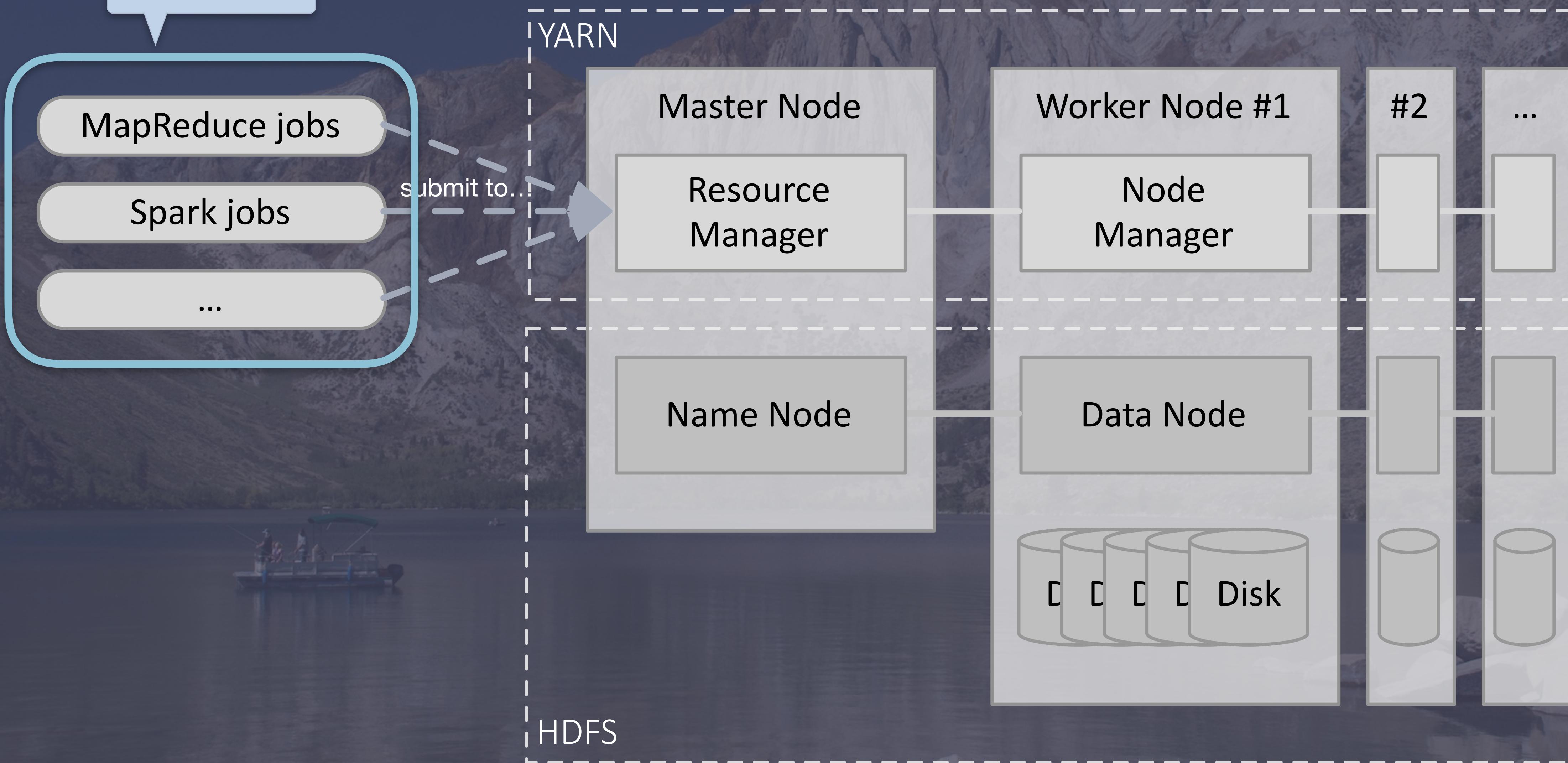
A scenic landscape featuring a large, rugged mountain range with sharp peaks and vertical rock faces. In the foreground, a calm lake reflects the surrounding mountains. A small boat with two people is visible on the left side of the lake. The sky is clear and blue.

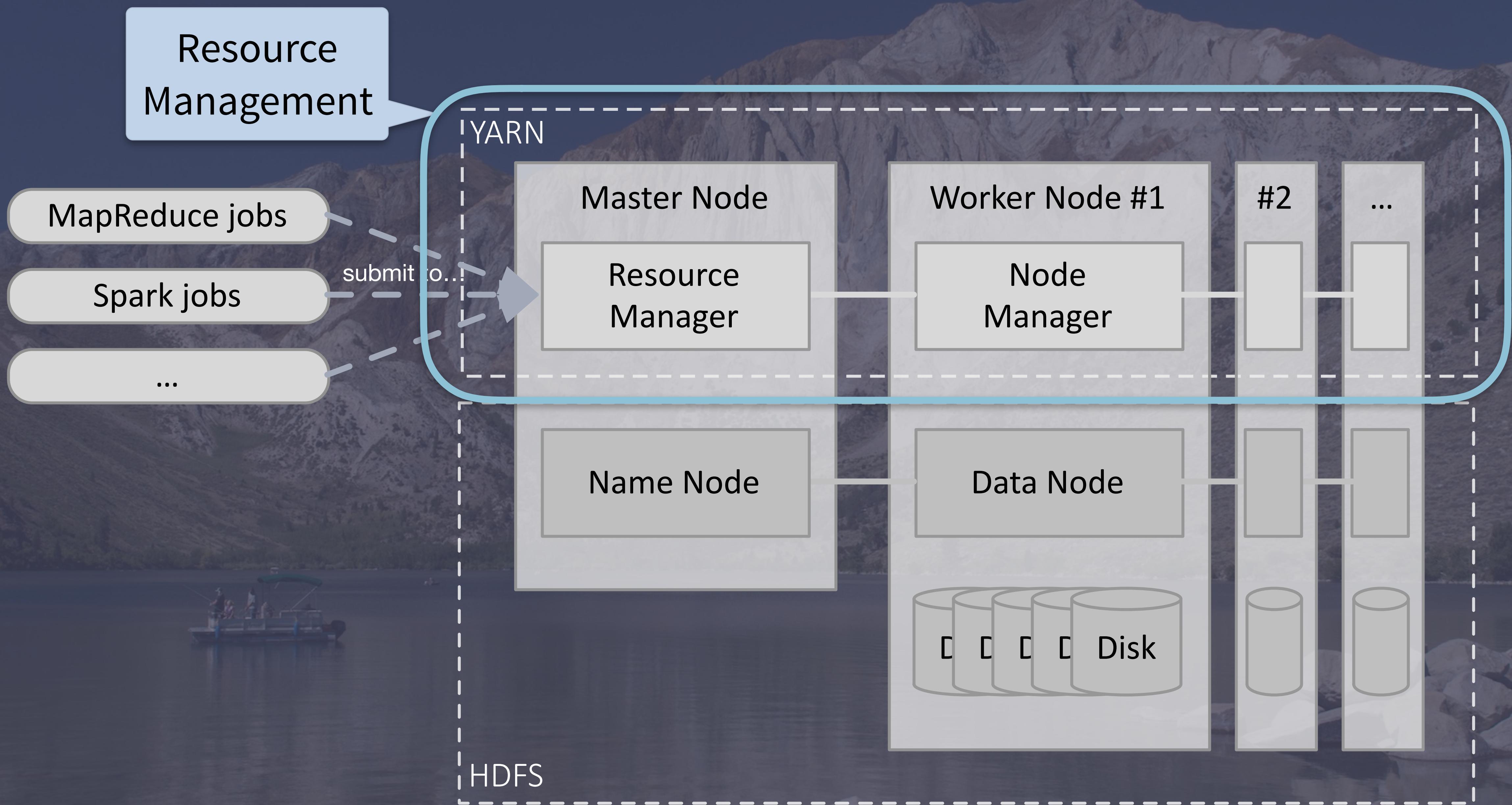
Hadoop: Classic Batch Architecture



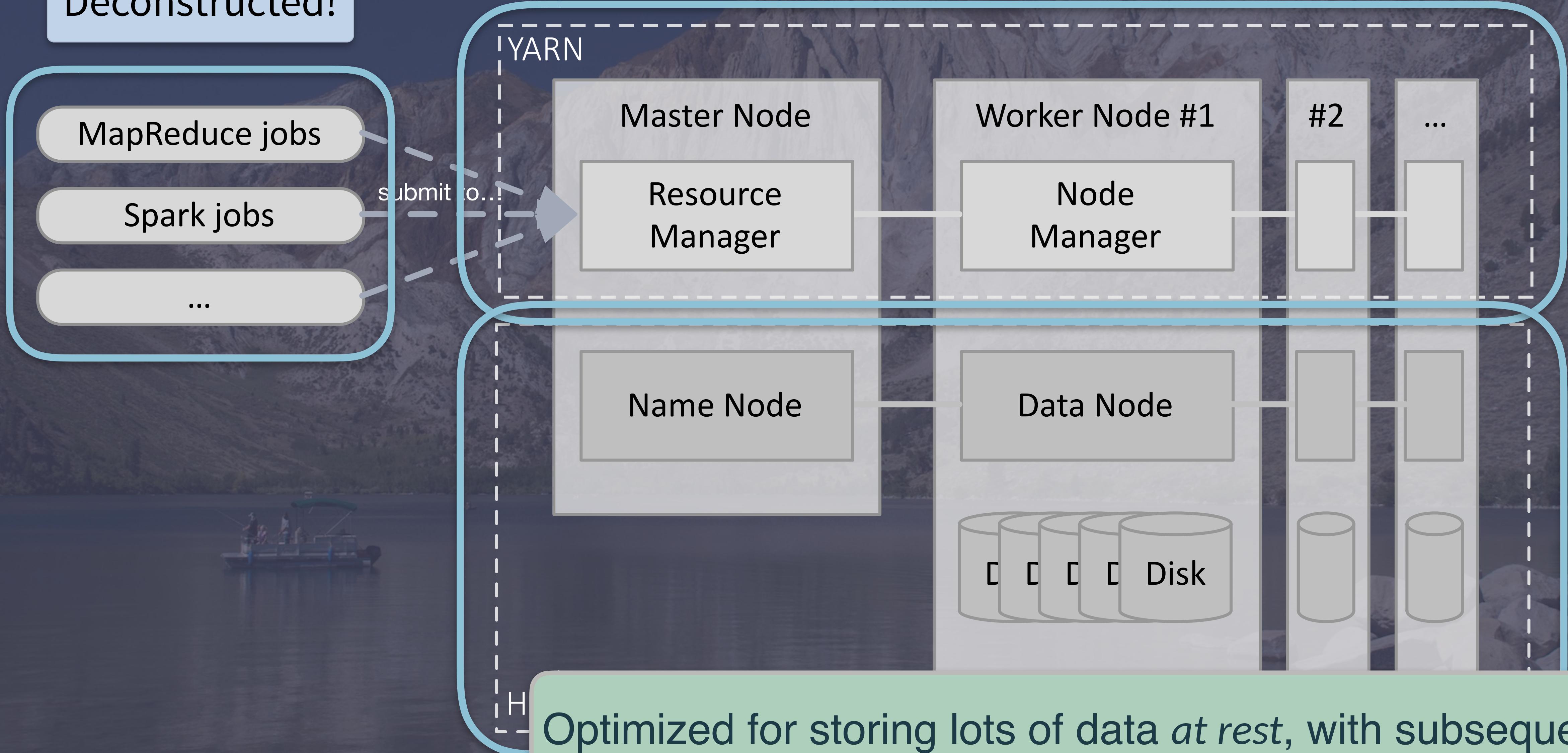


Compute



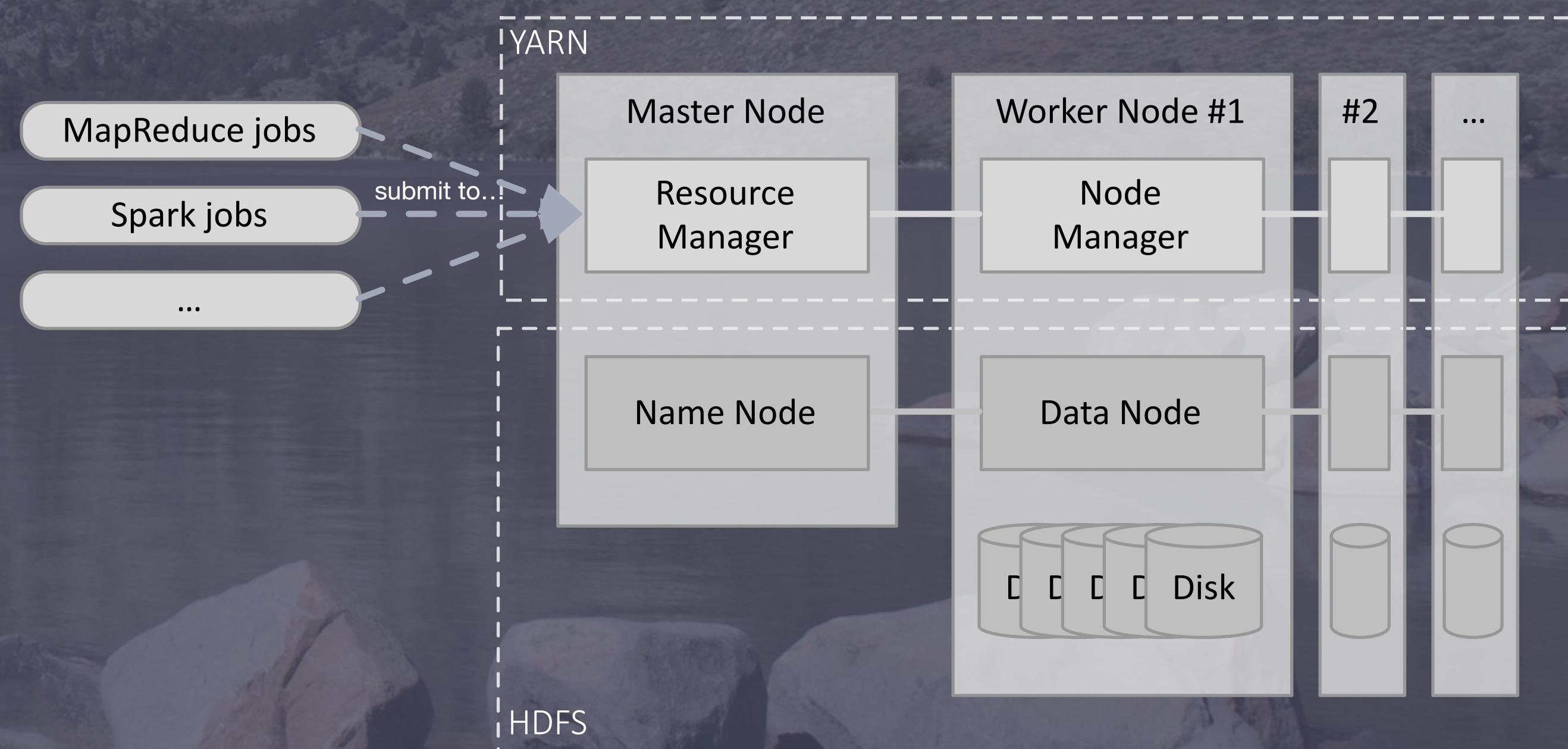


Database Deconstructed!

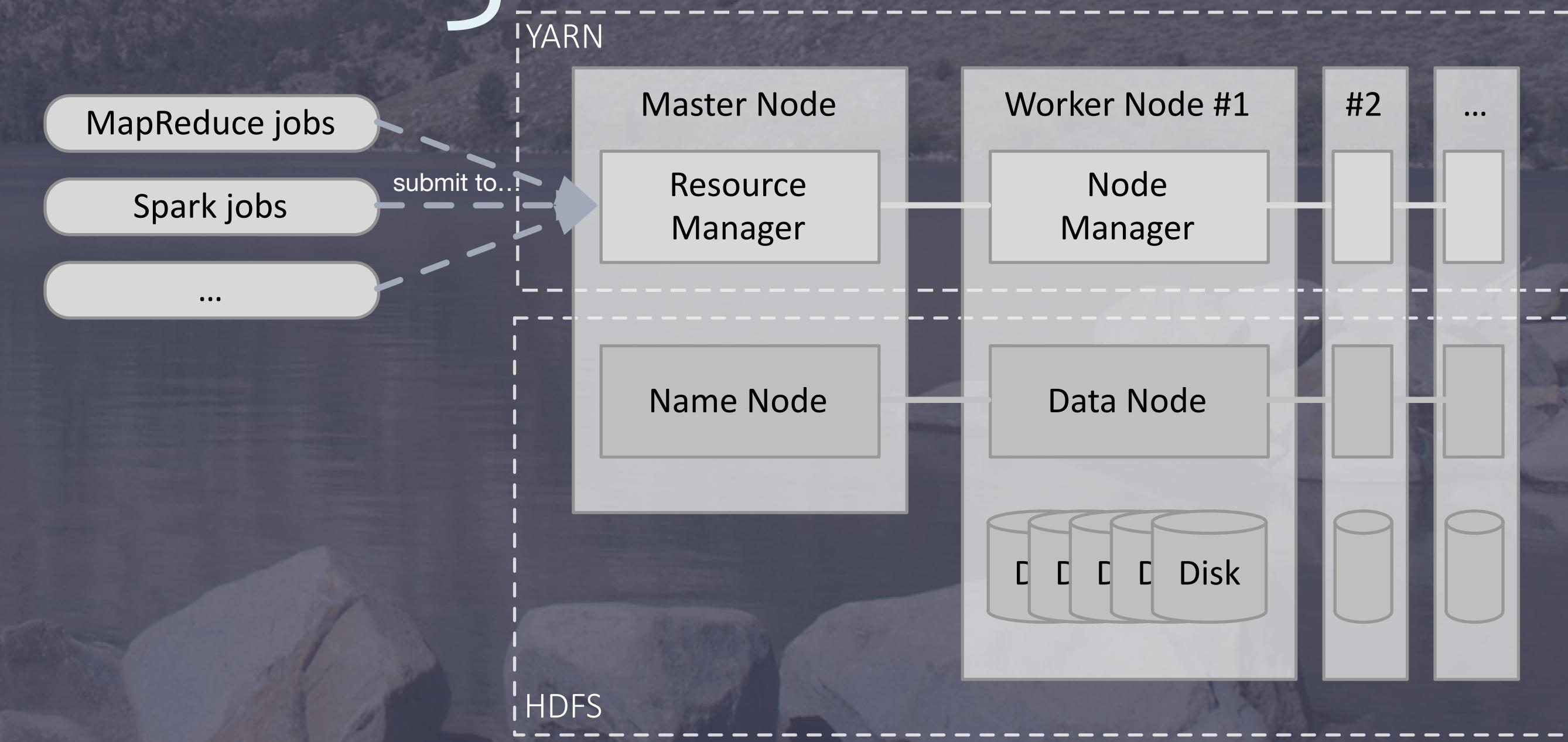


Optimized for storing lots of data *at rest*, with subsequent processing, but not optimized for data *in motion*.

- Characteristics
 - Batch and interactive queries
 - Massive storage - HDFS is the data “backplane”
- Integrate jobs through HDFS
- Multiuser jobs

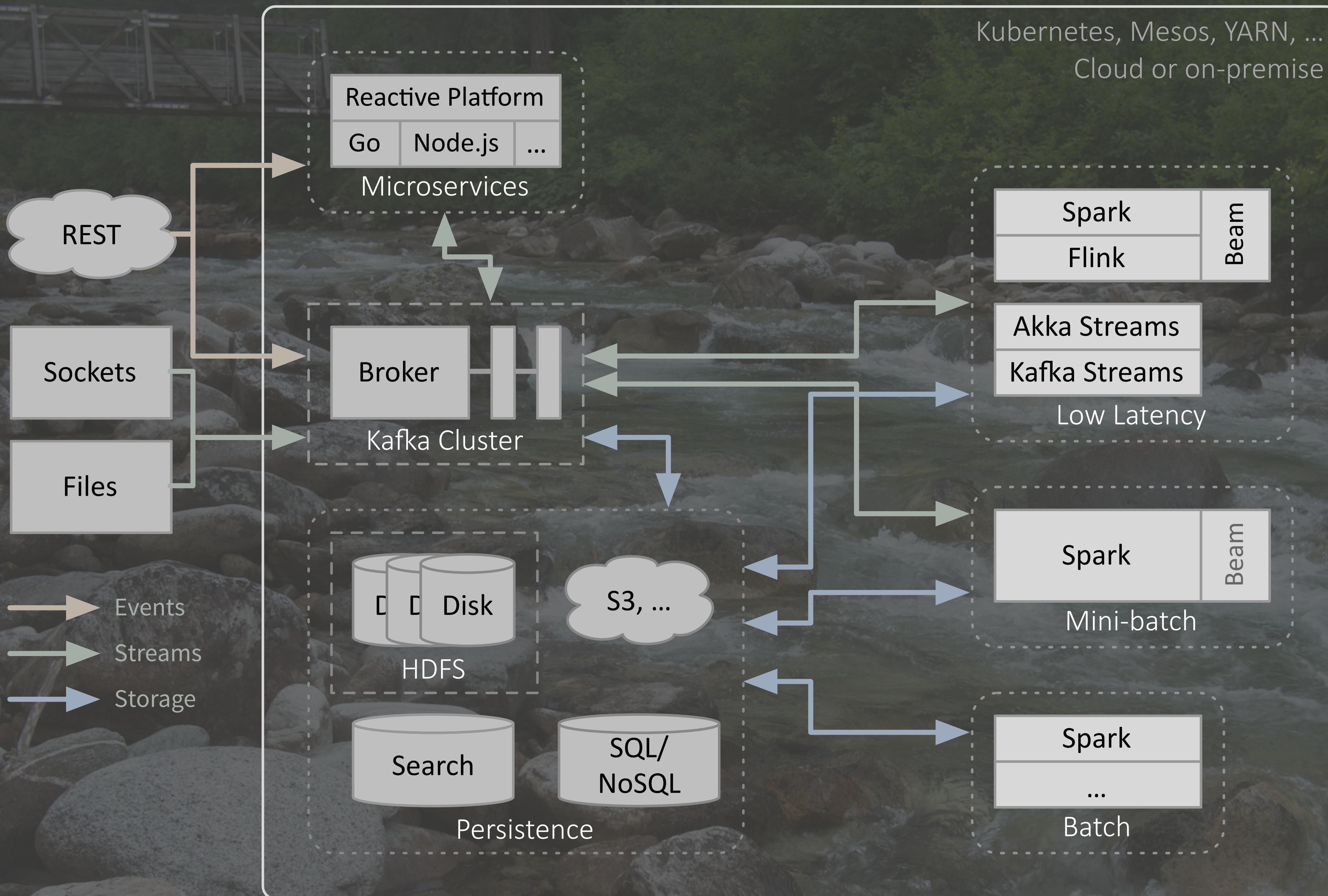


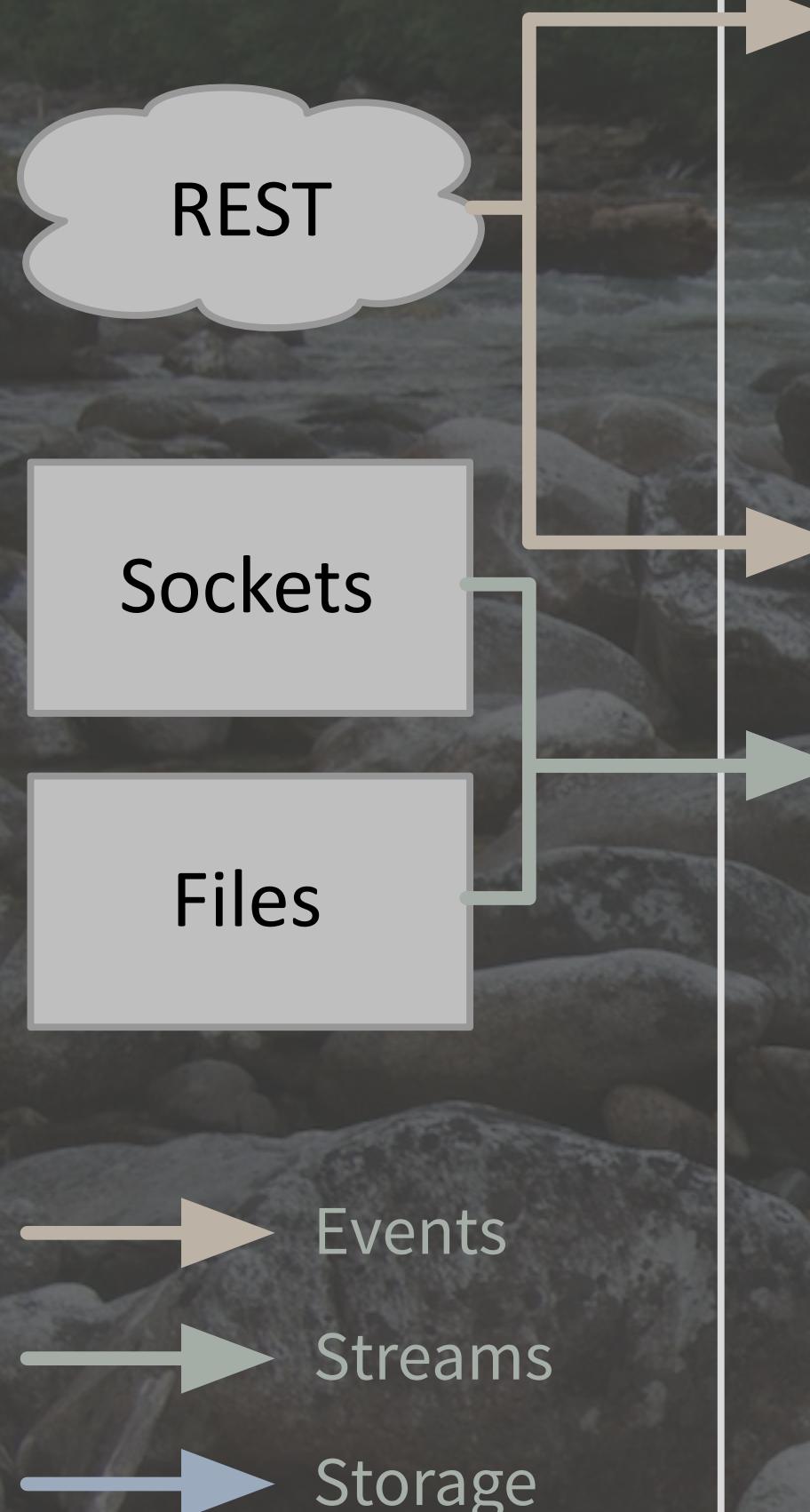
- Use Cases
 - Data warehouse replacement
 - Interactive exploration
 - Offline ML model training



A photograph of a rocky river flowing under a wooden bridge. The water is clear and greenish-blue, with white rapids. Large rocks are scattered along the riverbed. A wooden bridge is visible in the background.

New Streaming, “Fast Data” Architecture



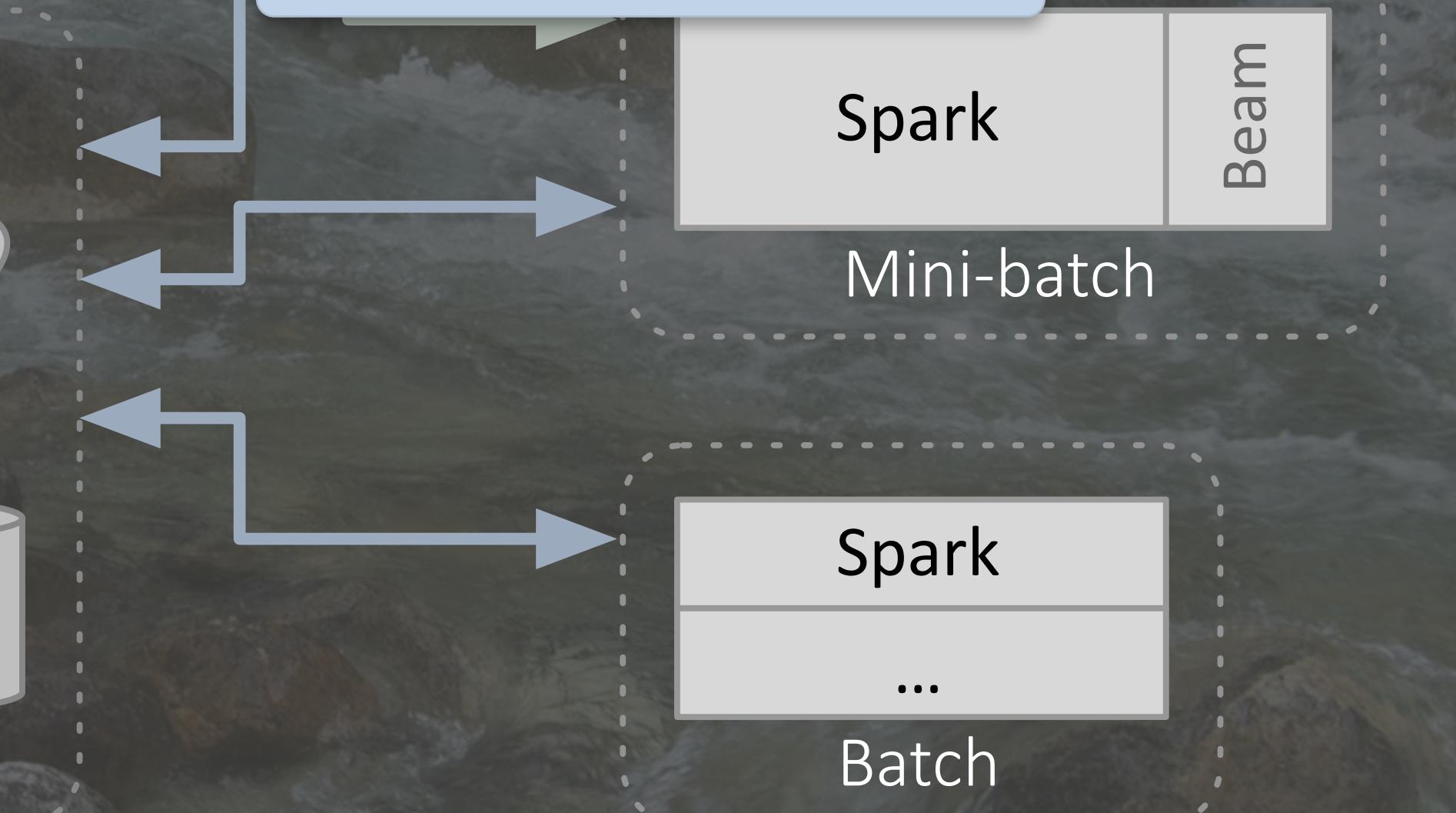


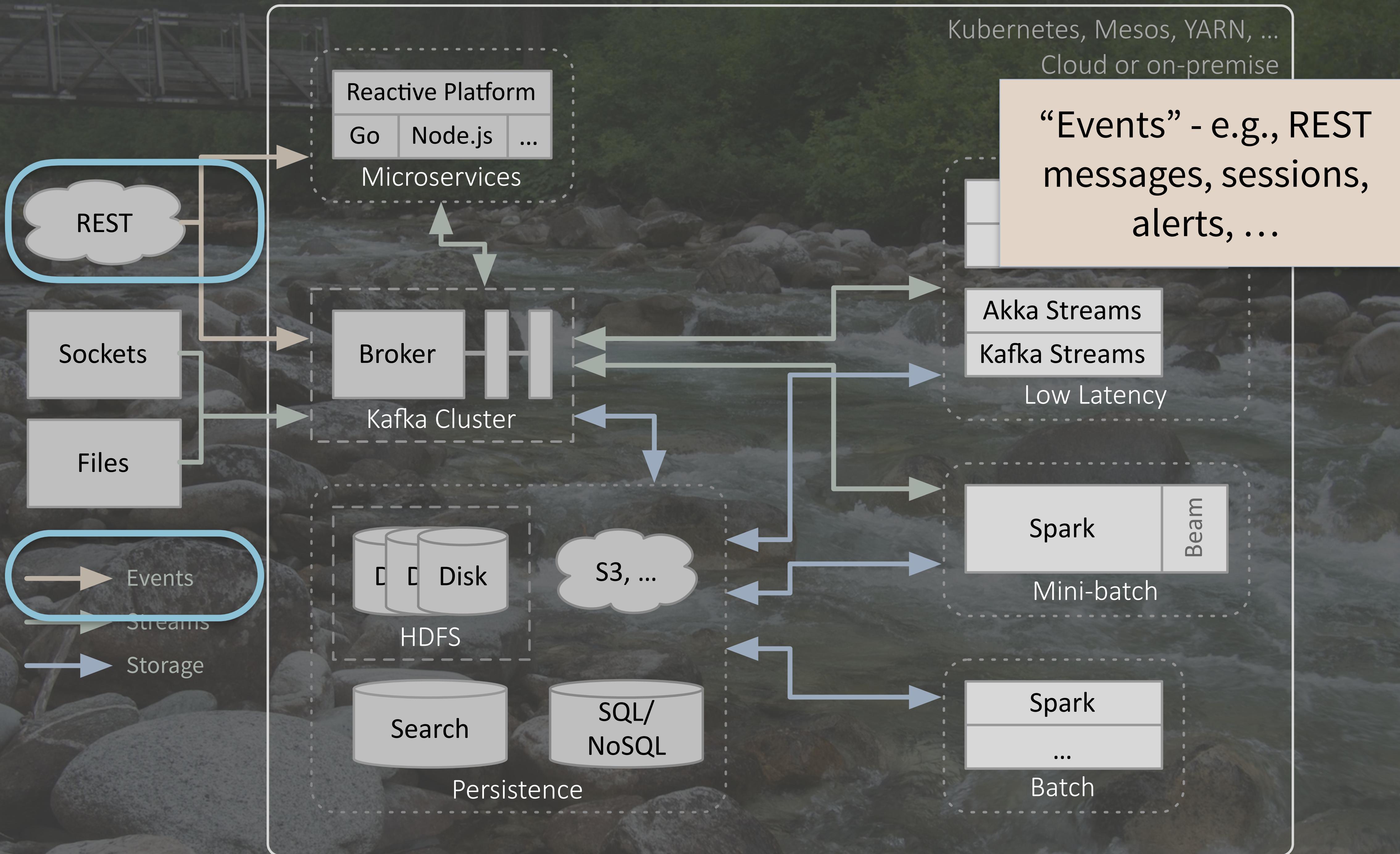
Kubernetes and Mesos provide the job and resource management needed for dynamic, heterogeneous work loads

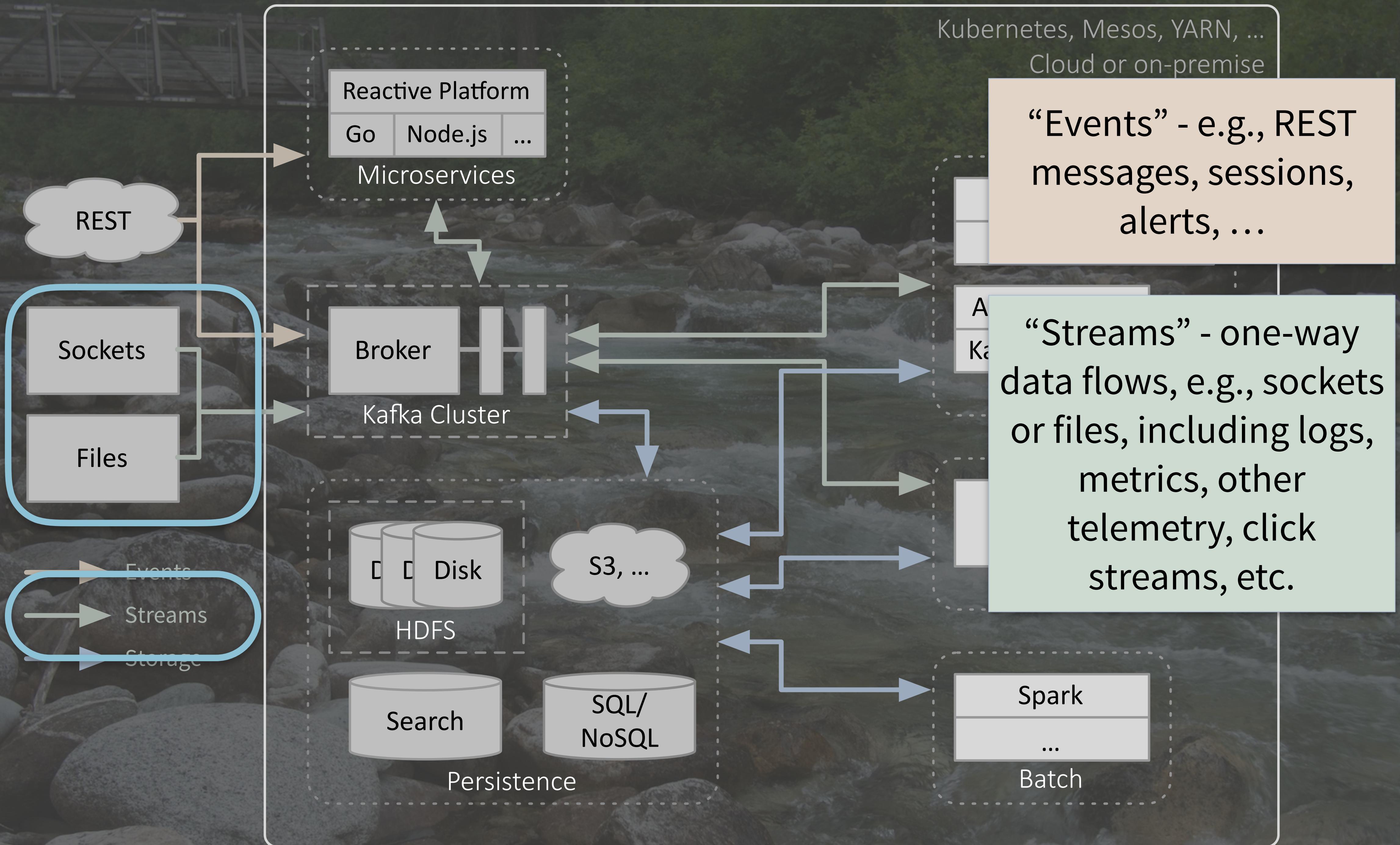
Kubernetes, Mesos, YARN, ...
Cloud or on-premise

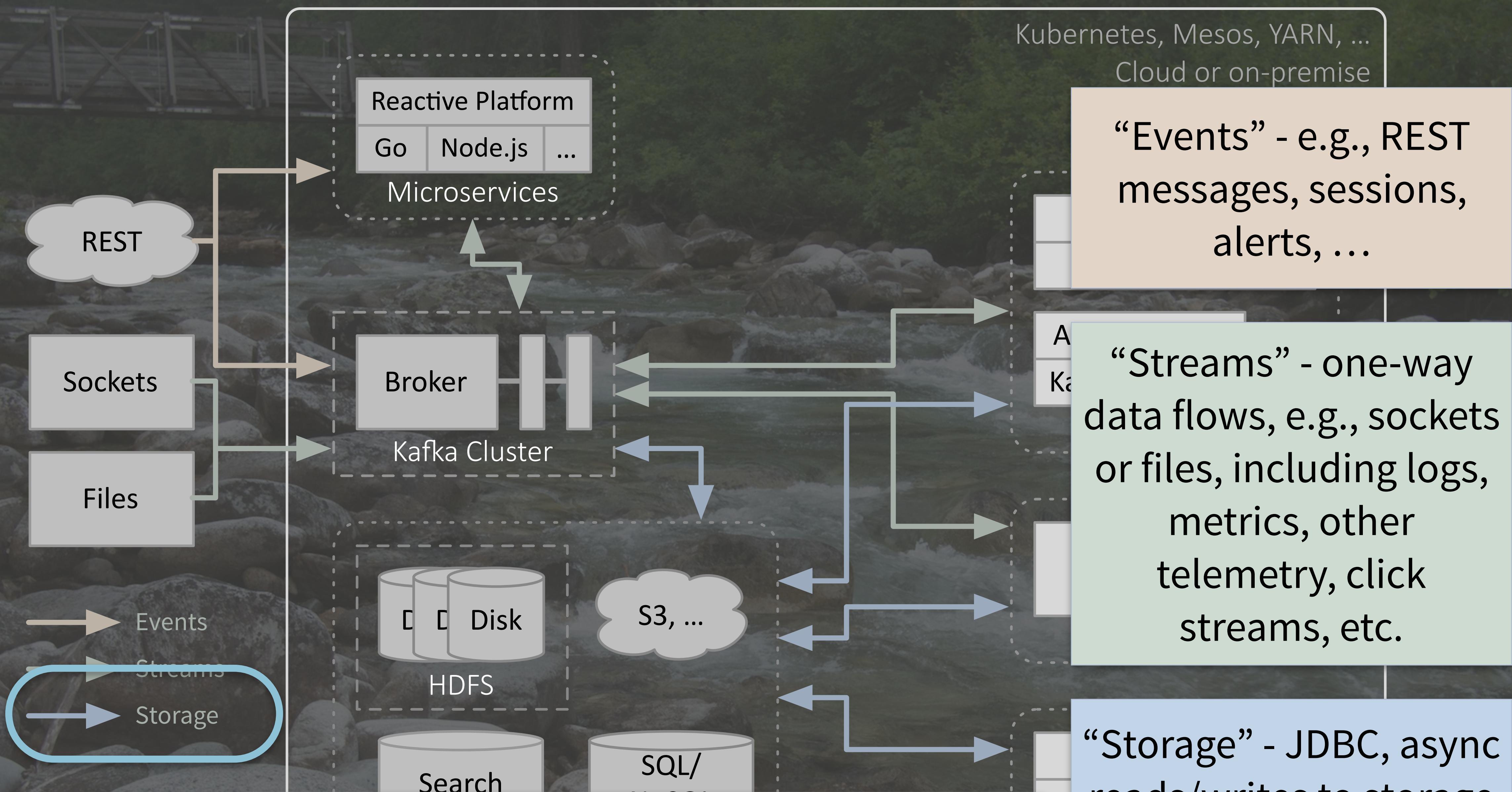
While YARN can be used, it's not flexible enough for today's dynamic workloads

Deploy in the cloud or on premise

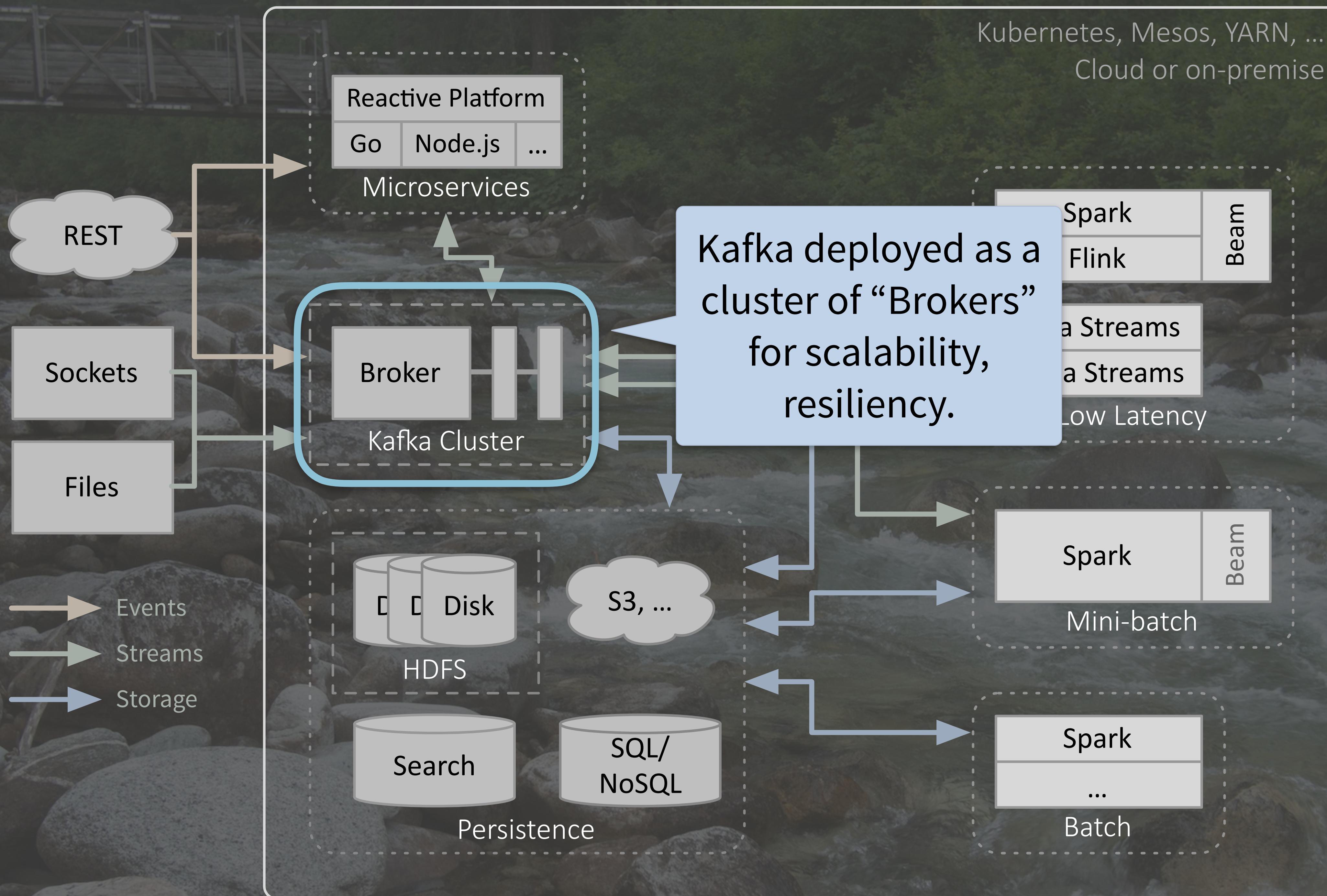


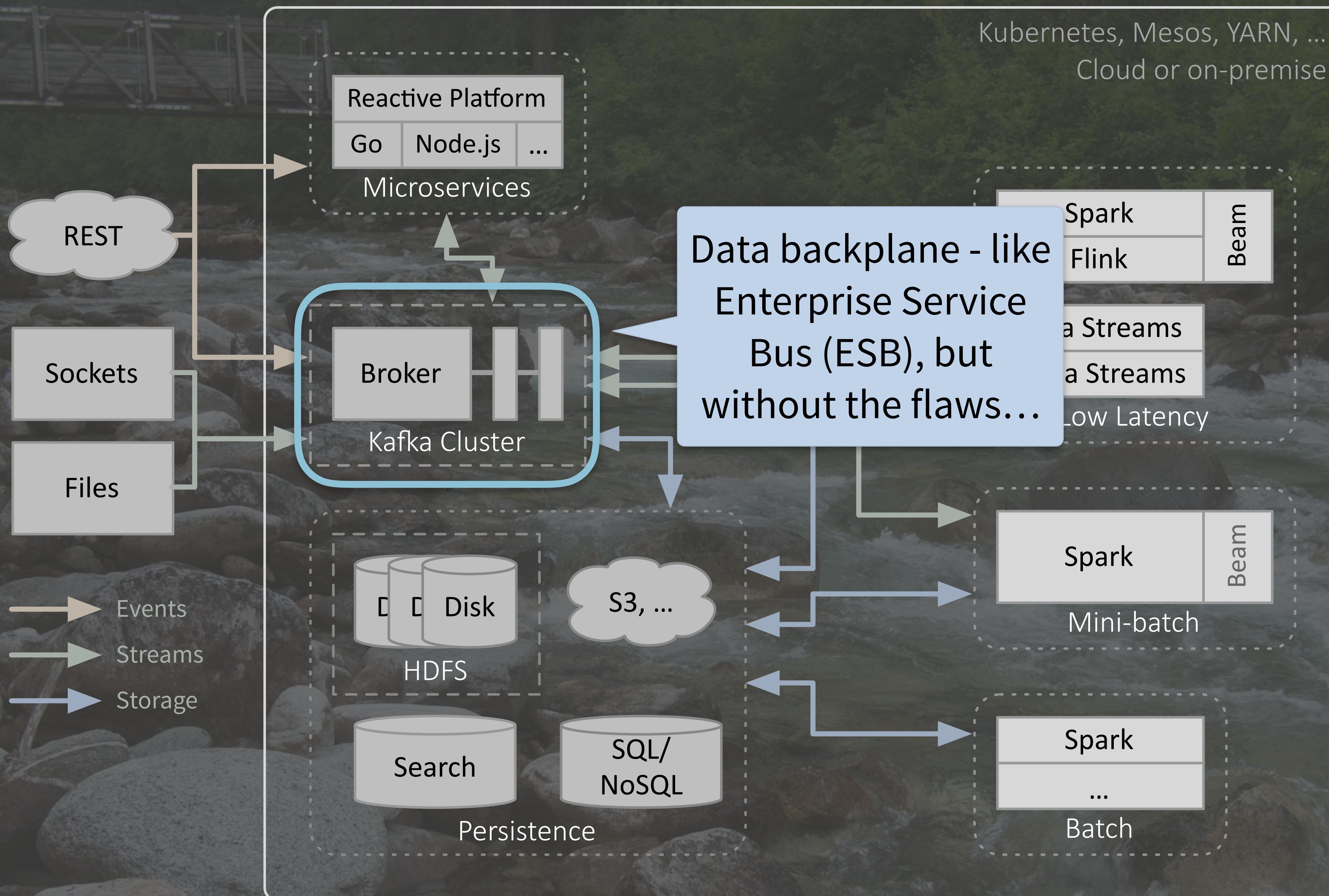






Each has different volumes, velocities, latency characteristics, protocols, etc.





Why Kafka?

Organized into topics

Topics are partitioned, replicated, and distributed

Kafka

Partition 1

Topic A

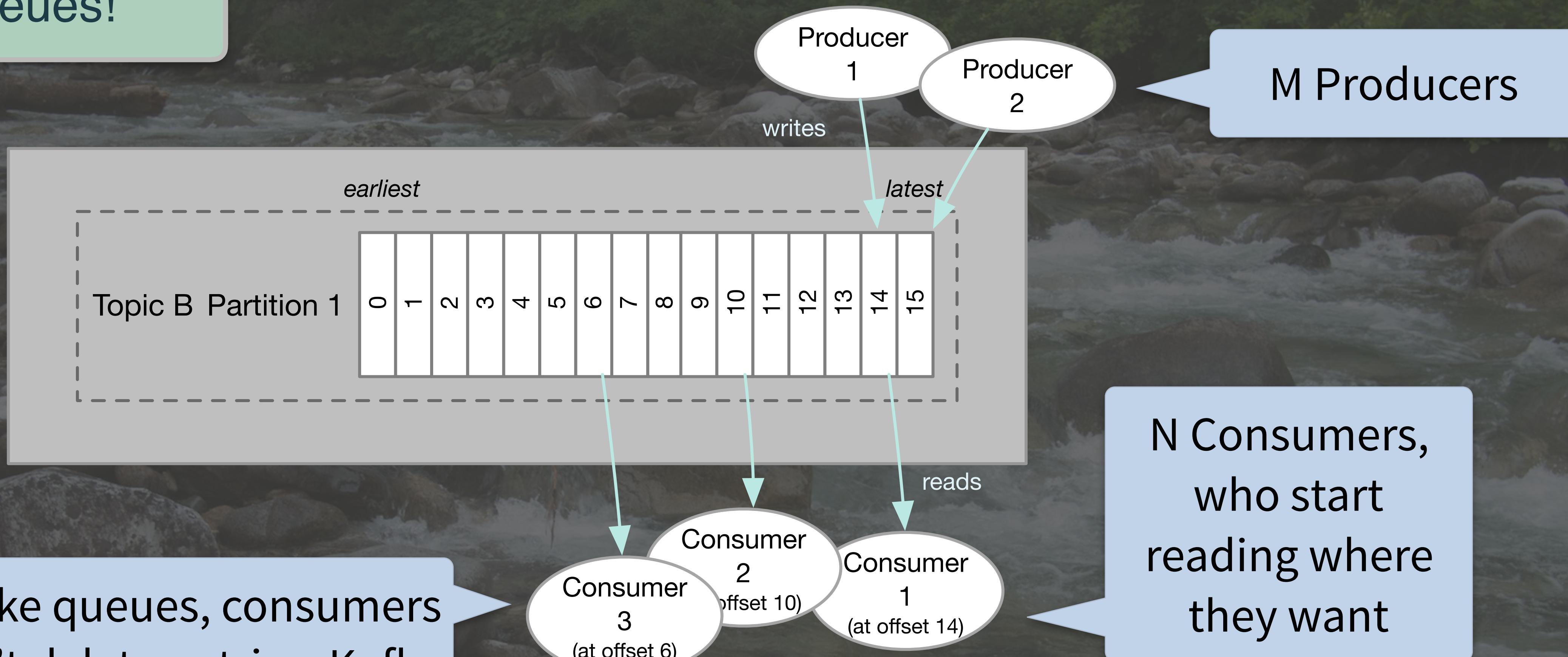
Partition 2

Topic B Partition 1



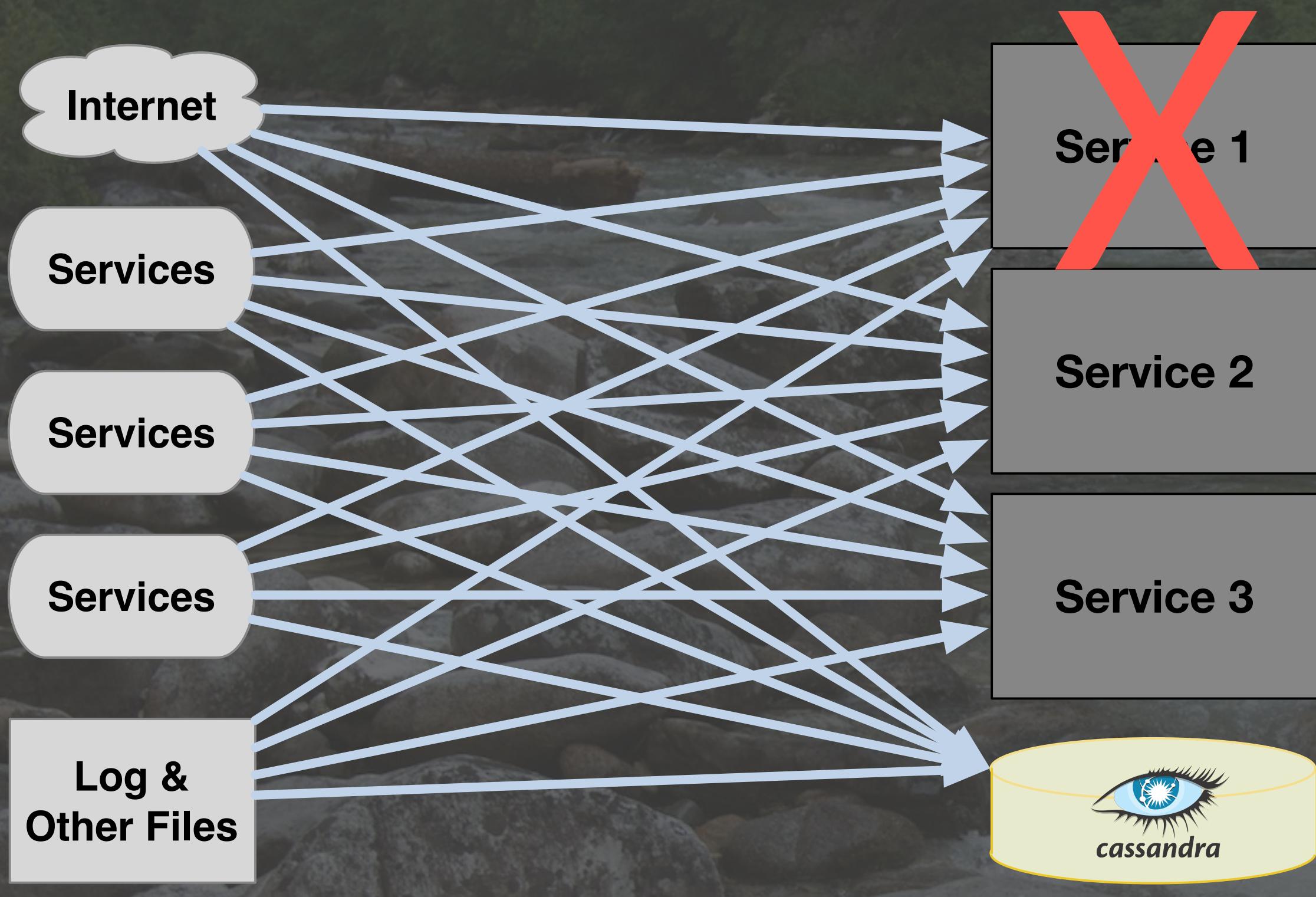
Why Kafka?

Logs, not
queues!



Using Kafka

Before:



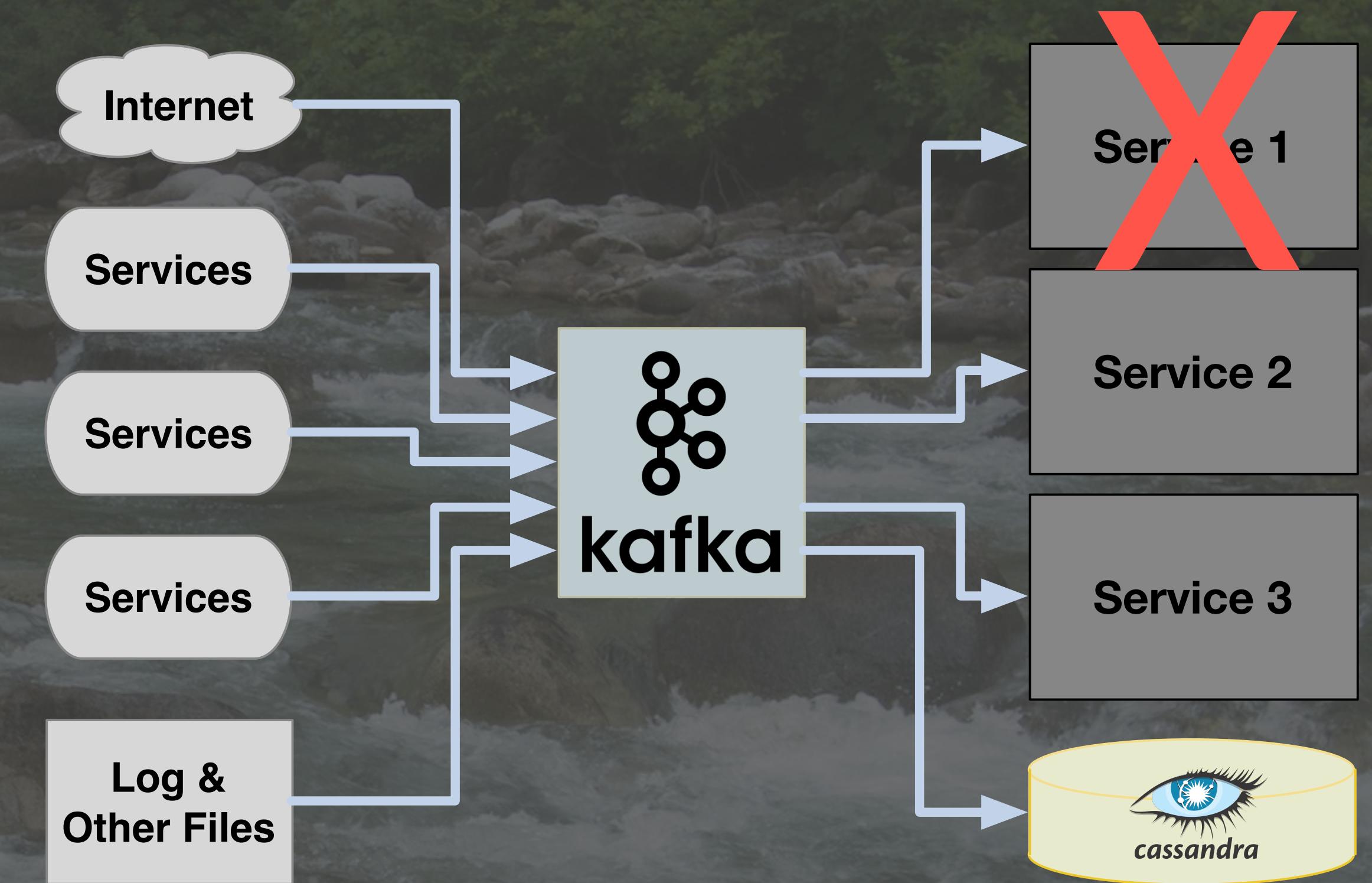
Producers

$N * M$ links

Consumers

 Messy and fragile;
what if “Service 1”
goes down?

After:

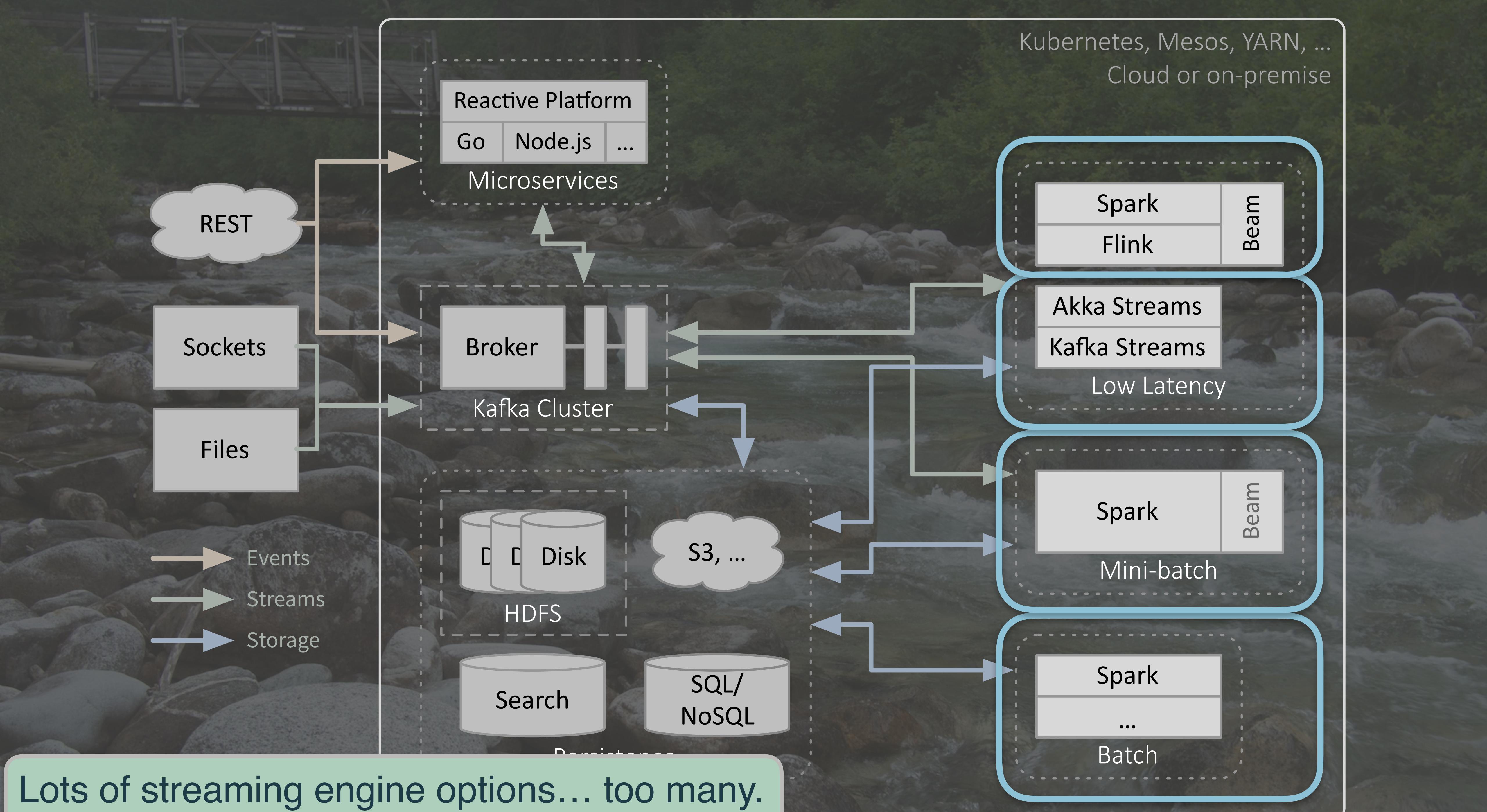


Producers

$N + M$ links

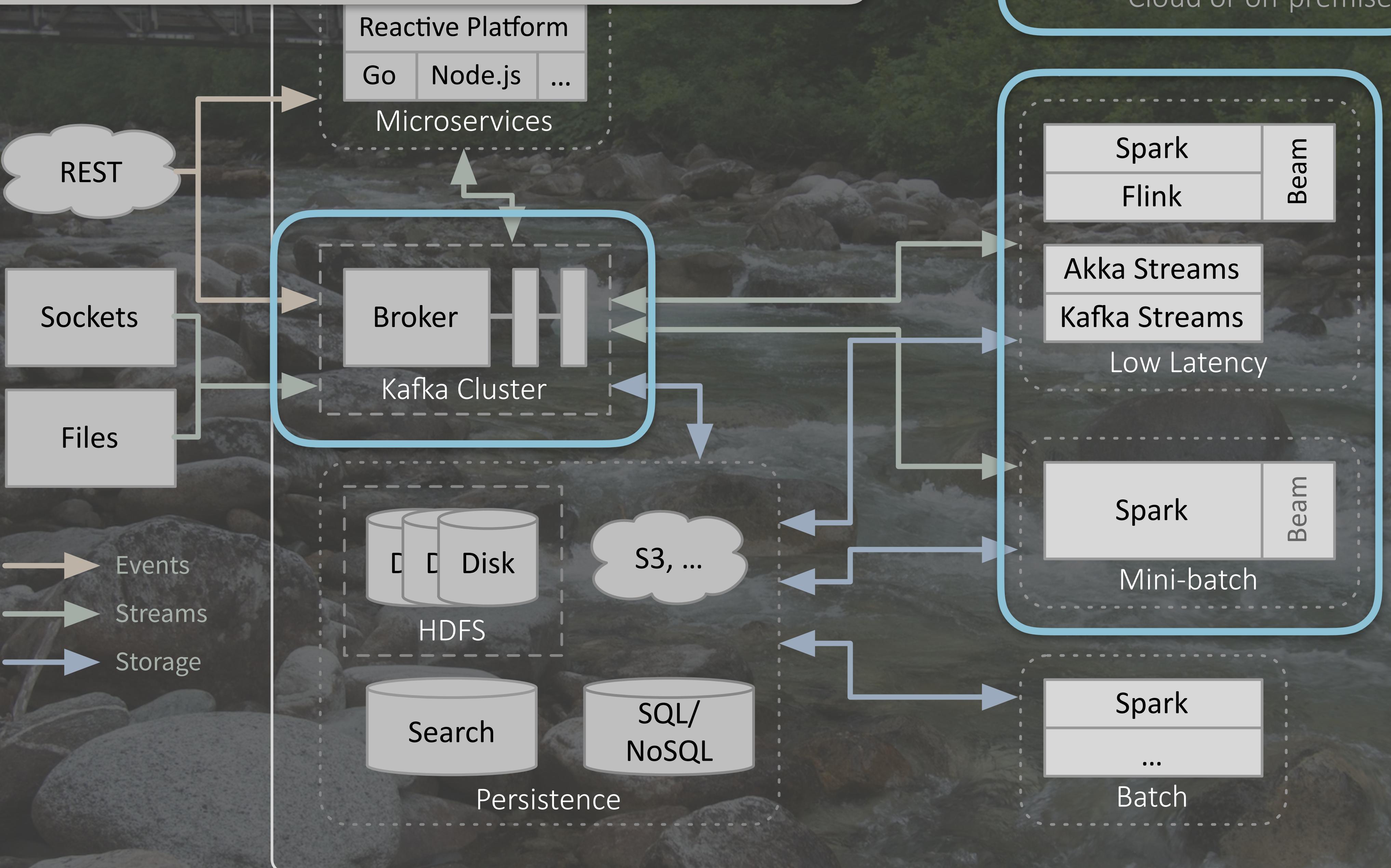
Consumers

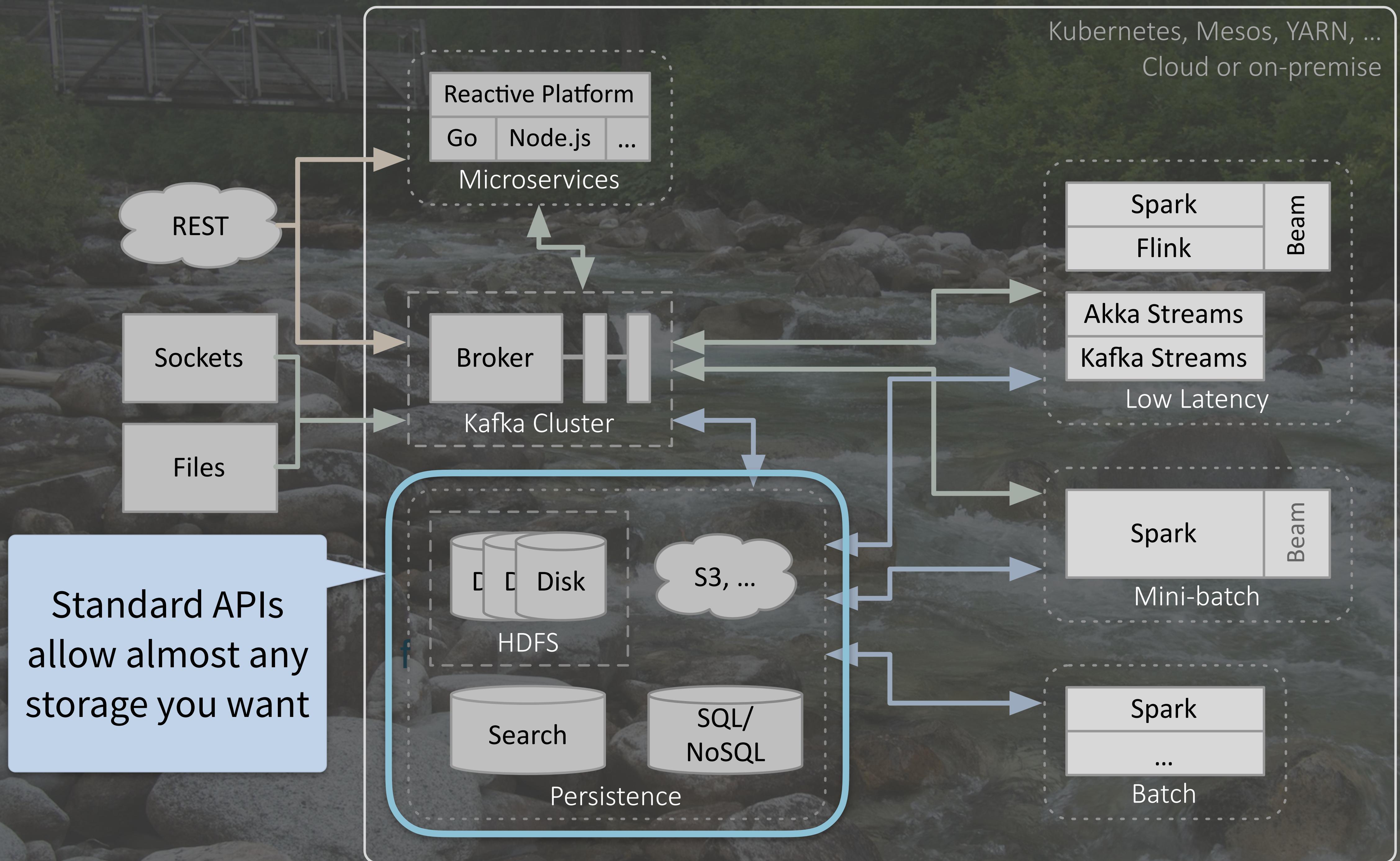
Simpler and more
robust! Loss of Service
1 means no data loss.

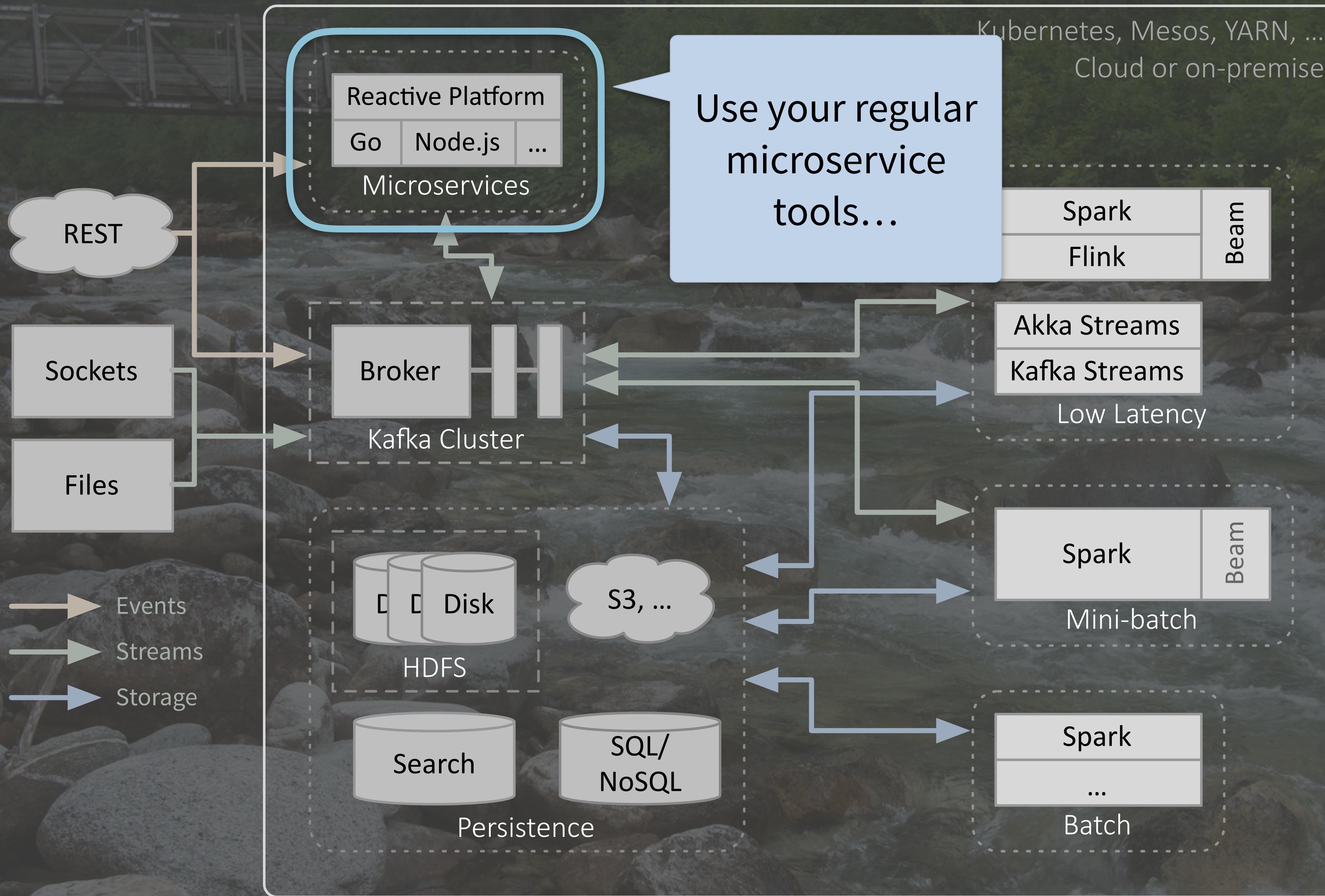


The streaming analog of a deconstructed database!

Kubernetes, Mesos, YARN, ...
Cloud or on-premise

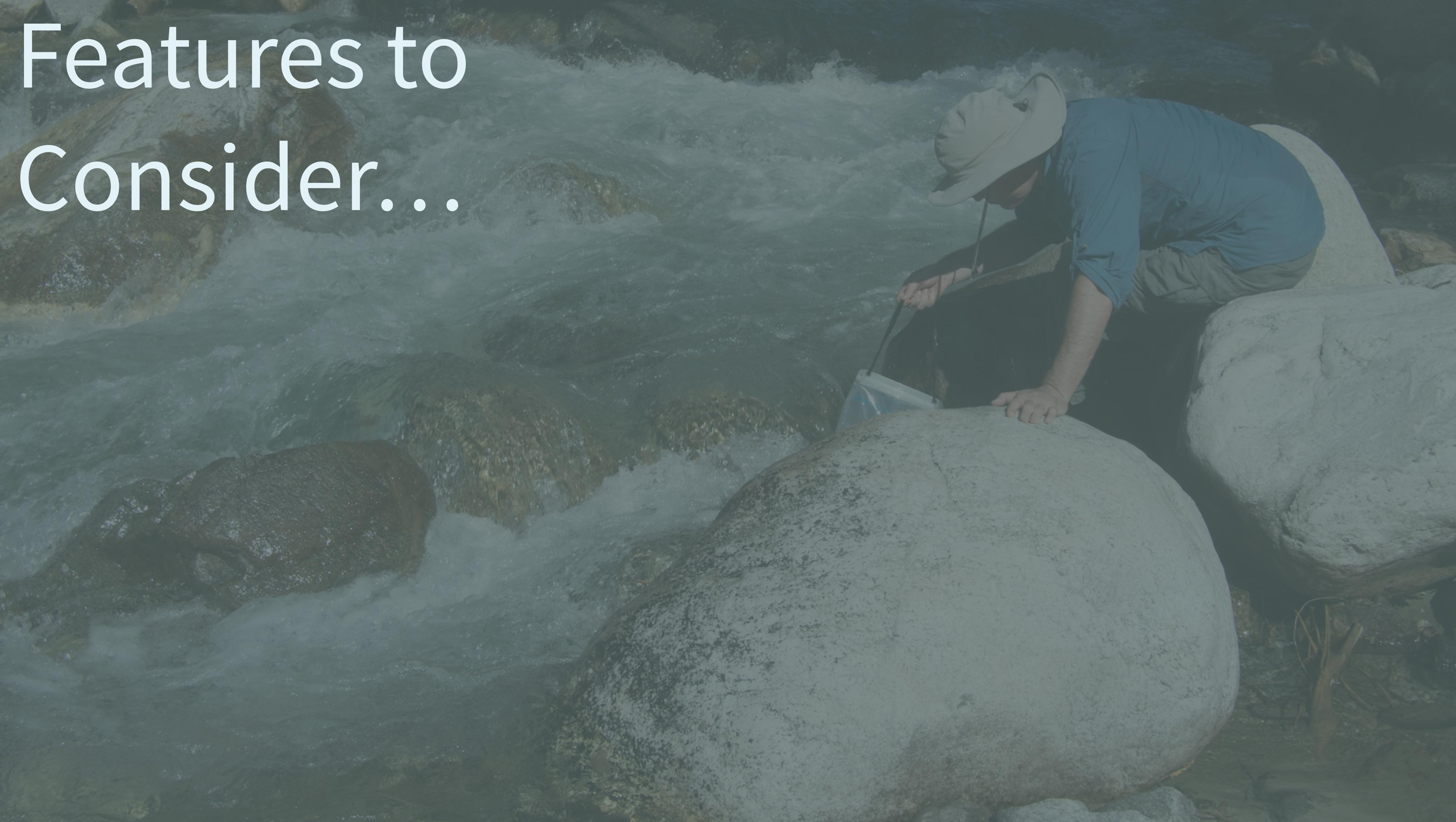








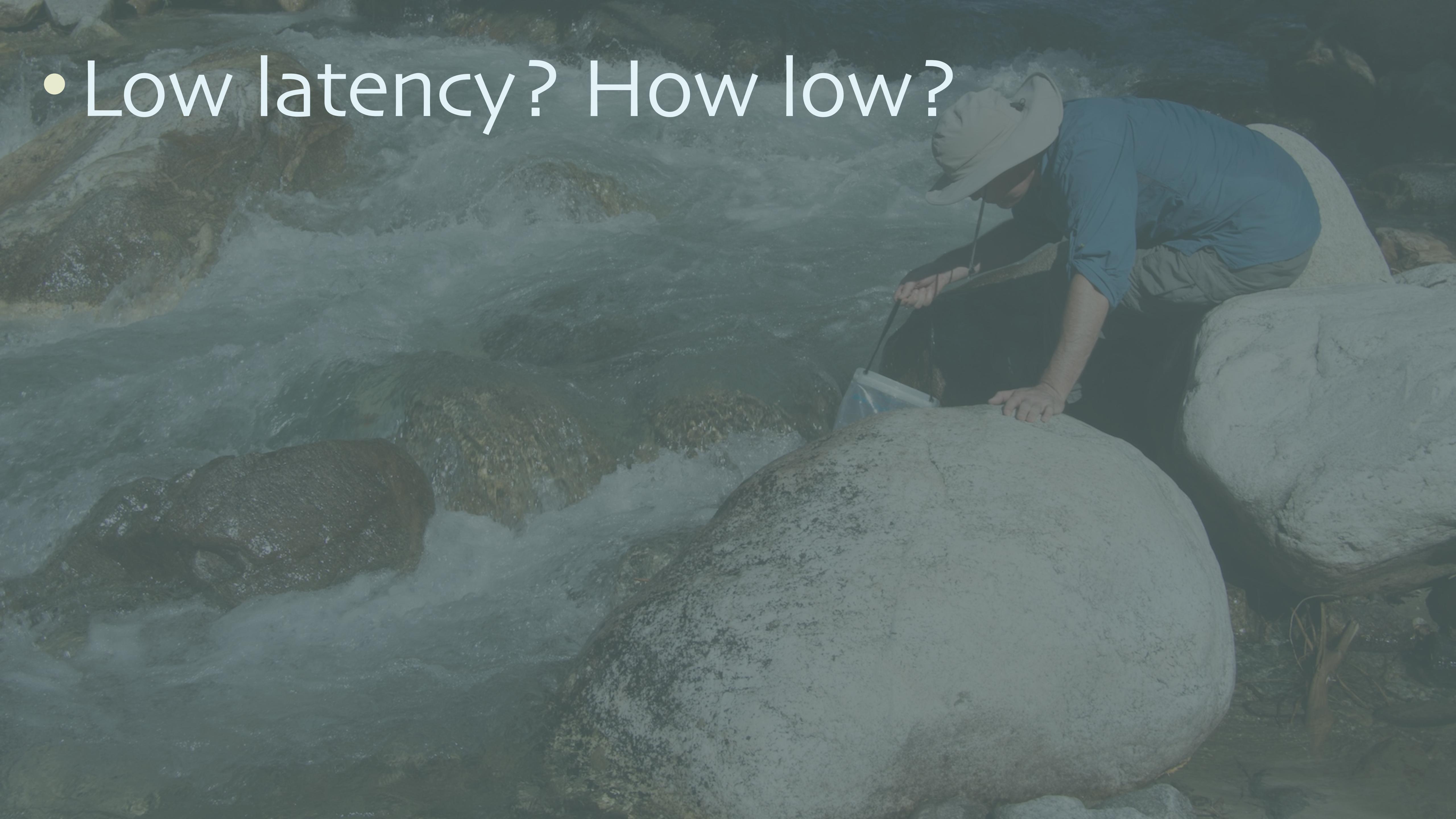
Streaming Engines



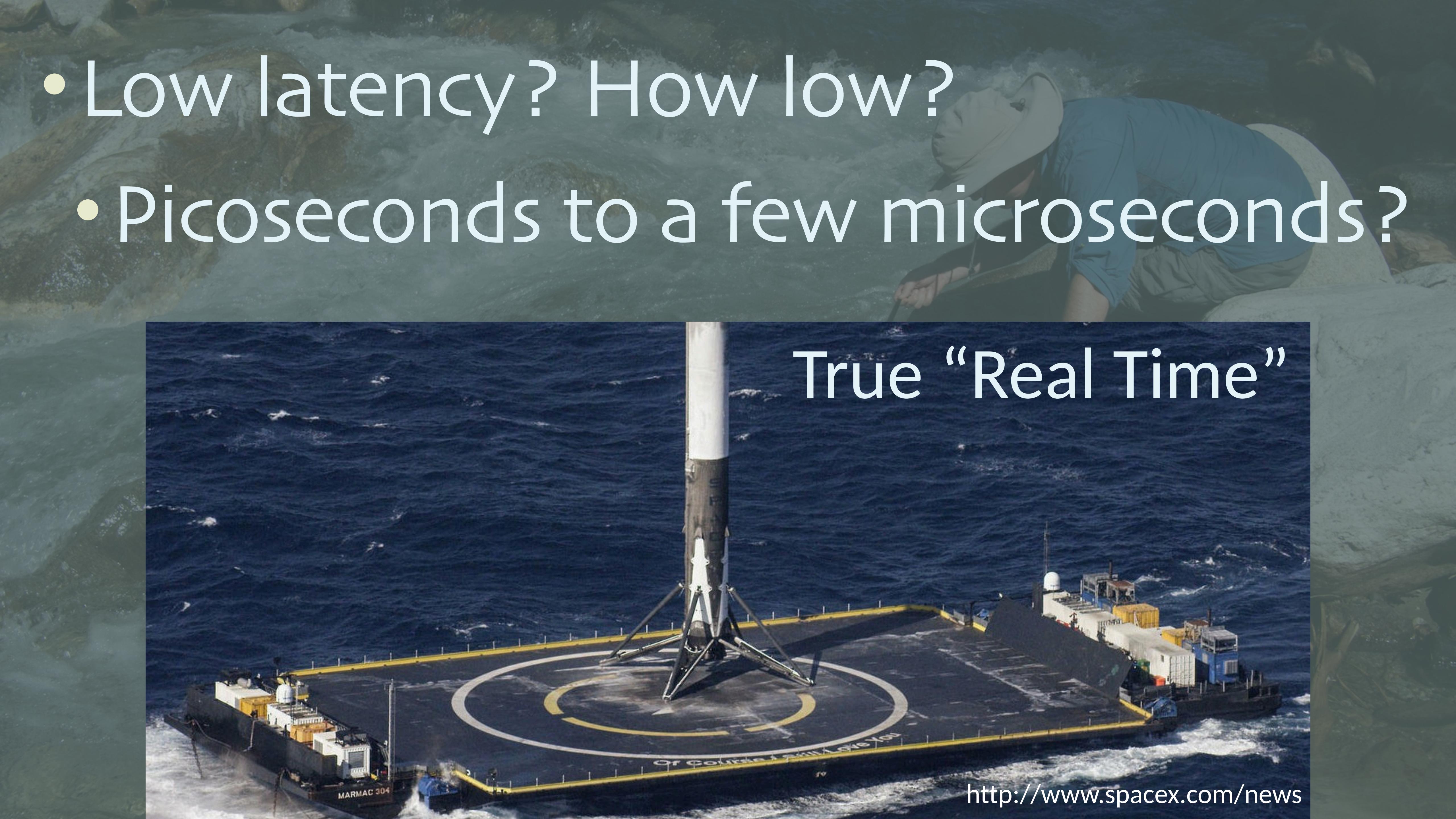
Features to Consider...

- Low latency? How low?
- High Volume: How high?
- Which kinds of data processing?
- Process data individually or in bulk?
- Preferred application architecture and DevOps processes?
- Integration with other services

- Low latency? How low?



- Low latency? How low?
- Picoseconds to a few microseconds?

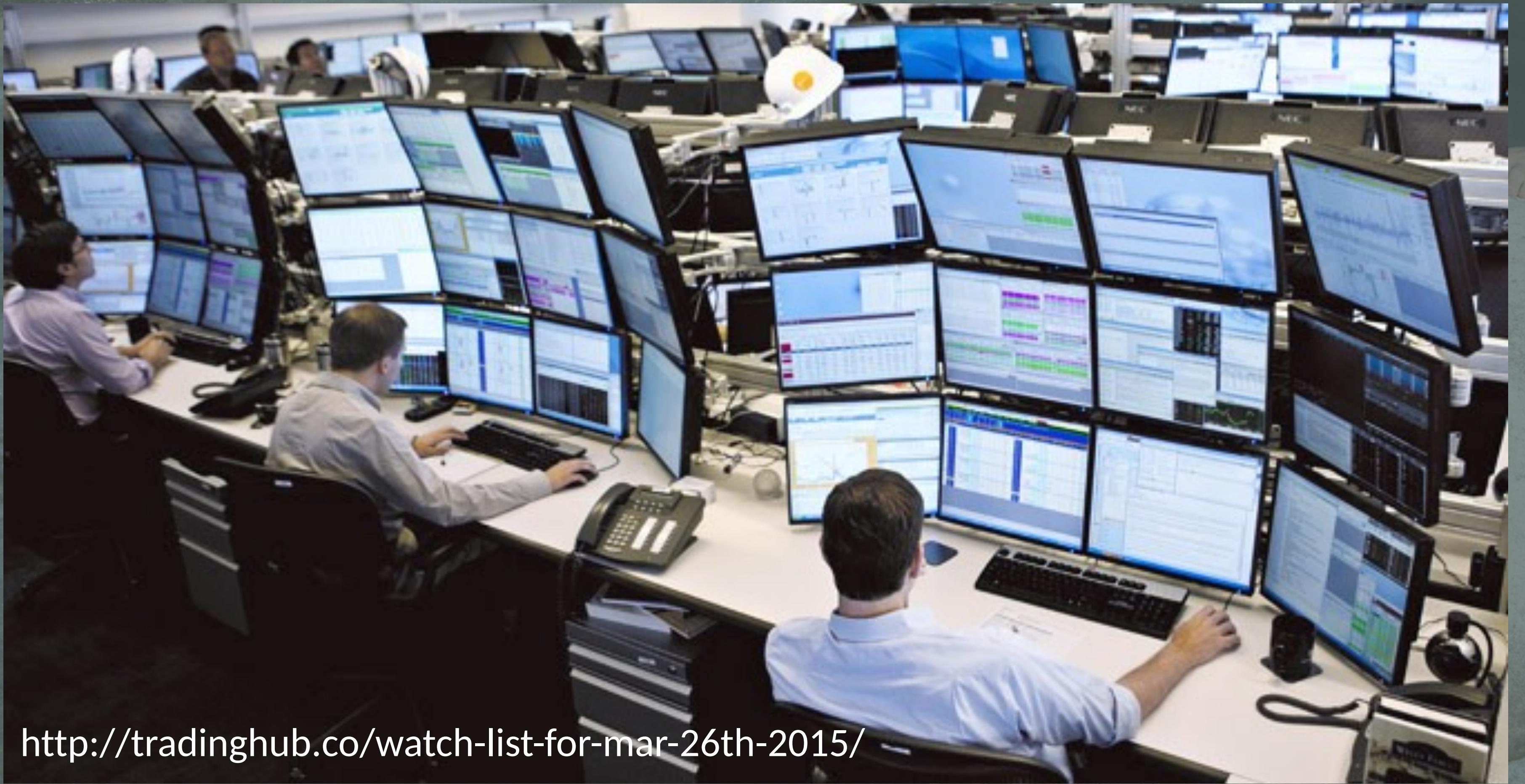


True “Real Time”



- Low latency? How low?
 - Picoseconds to a few microseconds?
 - Custom hardware (FPGAs).
 - “Kernel bypass” network HW/SW.
 - Custom C++ code.

- Low latency? How low?
- < 100 microseconds?



- Low latency? How low?
 - < 100 microseconds?
 - Fast JVM message handlers.
 - Akka Actors
 - LMAX Disruptor

- Low latency? How low?
- < 10 milliseconds?

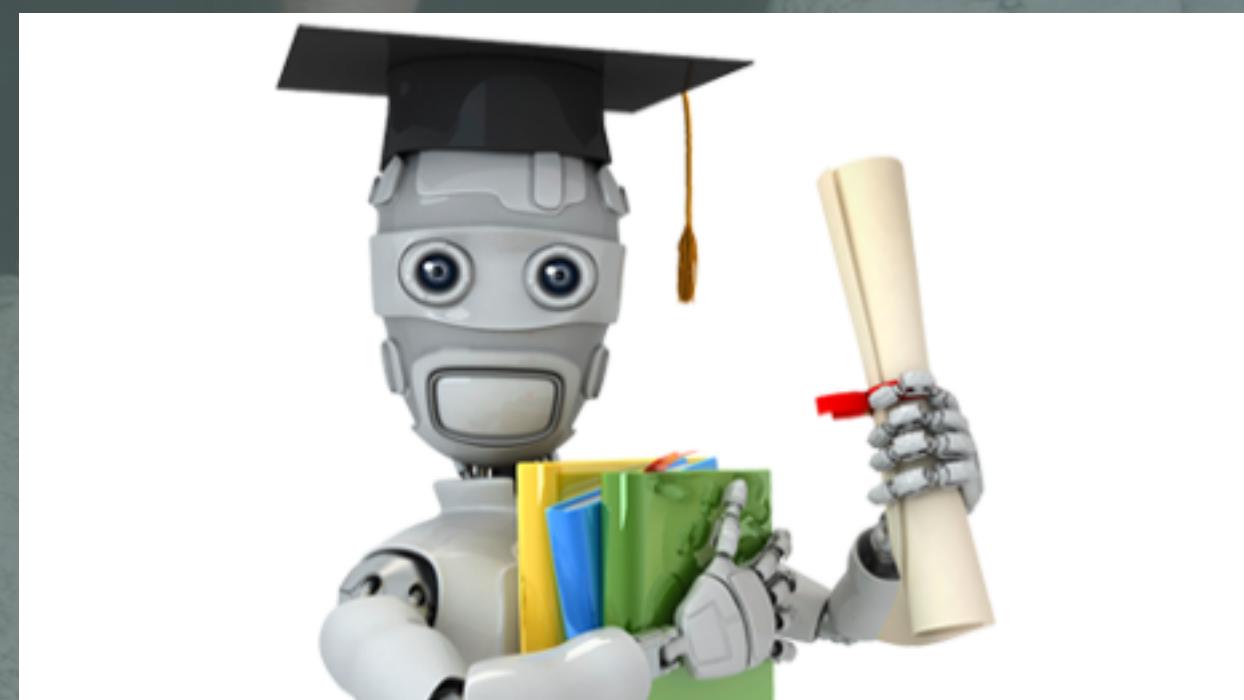


- Low latency? How low?
- < 10 milliseconds?
- Fast data streaming tools like Flink and more recently Spark, Akka (and Akka Streams), and Kafka Streams.

- Low latency? How low?
- < hundreds of milliseconds?



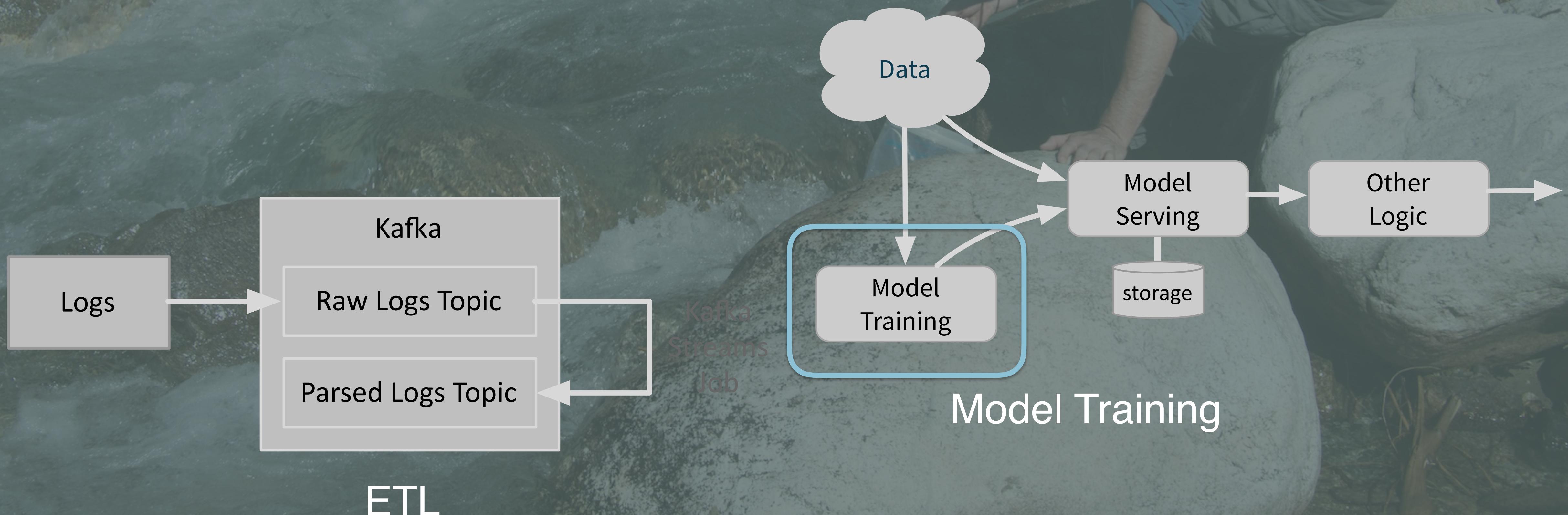
<https://github.com/keen/dashboards>



<https://www.coursera.org/learn/machine-learning>

- Low latency? How low?
- < hundreds of milliseconds?
- “micro-batches”
- Processing records in bulk, e.g.,
Spark’s micro-batch model and
“streaming SQL” over windows.

- Low latency? How low?
- < 1 second to minutes?

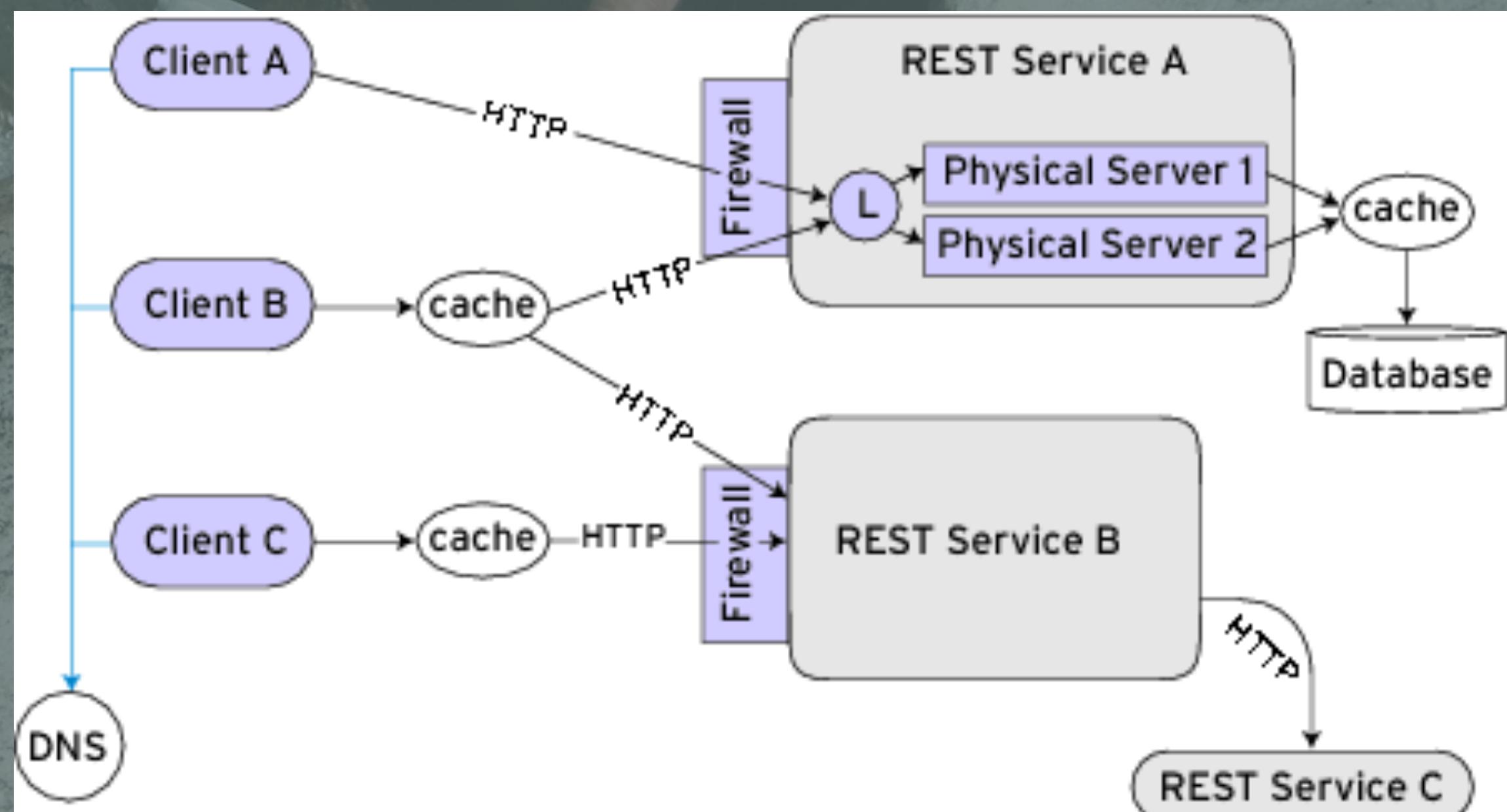


- Low latency? How low?
 - > 1 minute?
 - Consider periodic batch jobs!

- High Volume: How high?

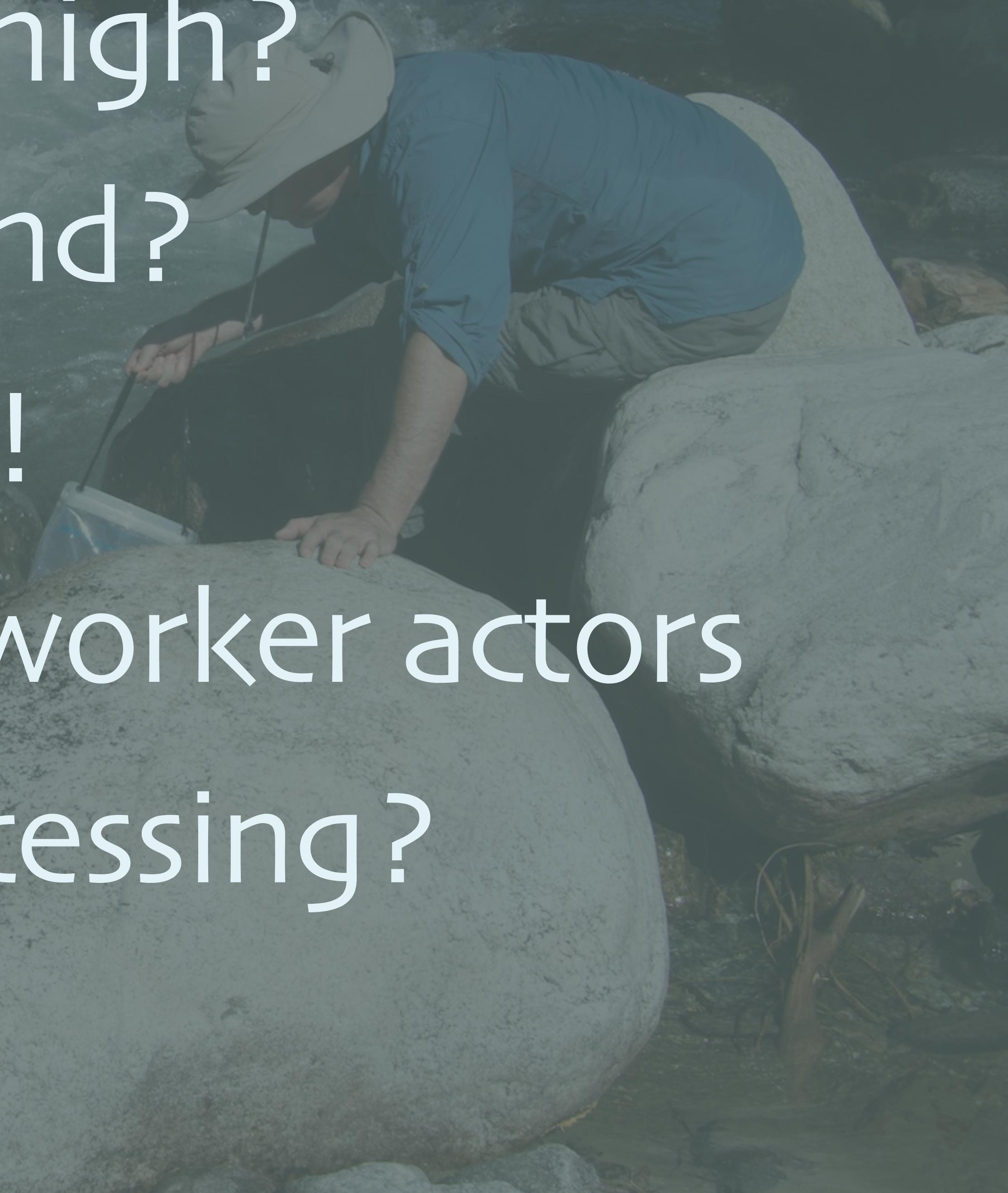


- High Volume: How high?
- < 10,000 events/second?
- REST
- One at a time...



<http://www.drdobbs.com/web-development/soa-web-services-and-restful-systems/199902676>

- High Volume: How high?
- < 100,000 per second?
- Nonblocking REST!
- Parallelism - Akka worker actors
- Switch to bulk processing?



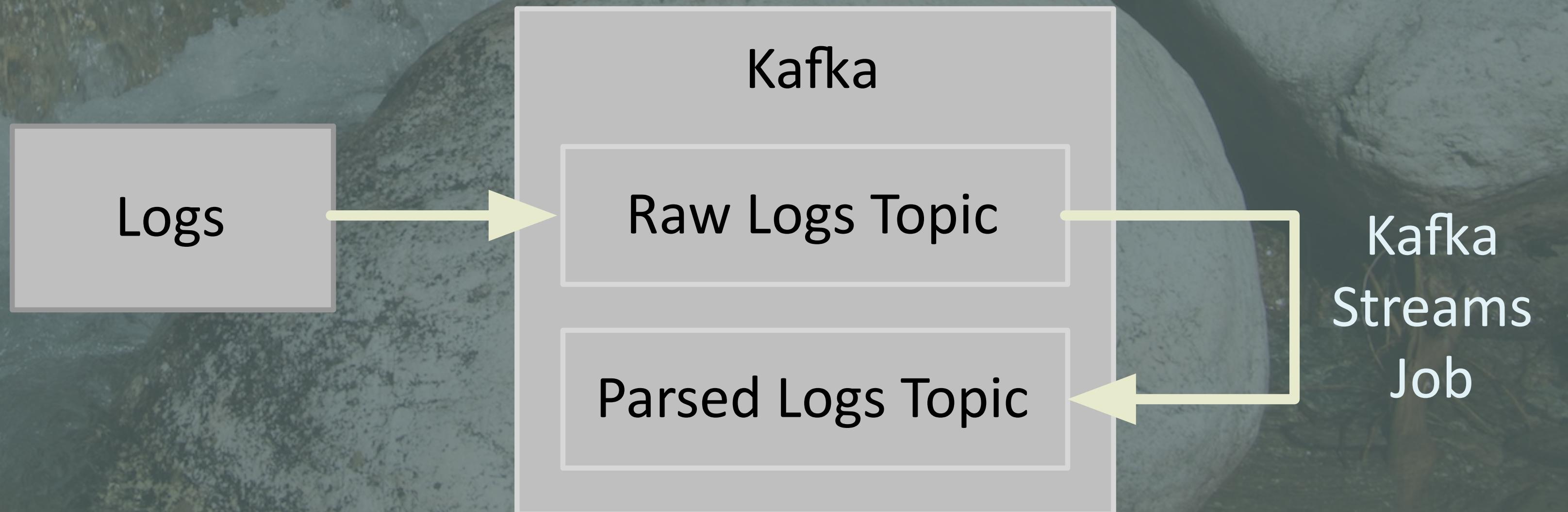
- High Volume: How high?
- 1,000,000s per second?
 - Flink or Spark Streaming
 - Process in bulk



- Which kinds of data processing?



- Which kinds of data processing?
- Extract, transform, and load (ETL)?



- Which kinds of data processing?
- “Dataflow” pipelines

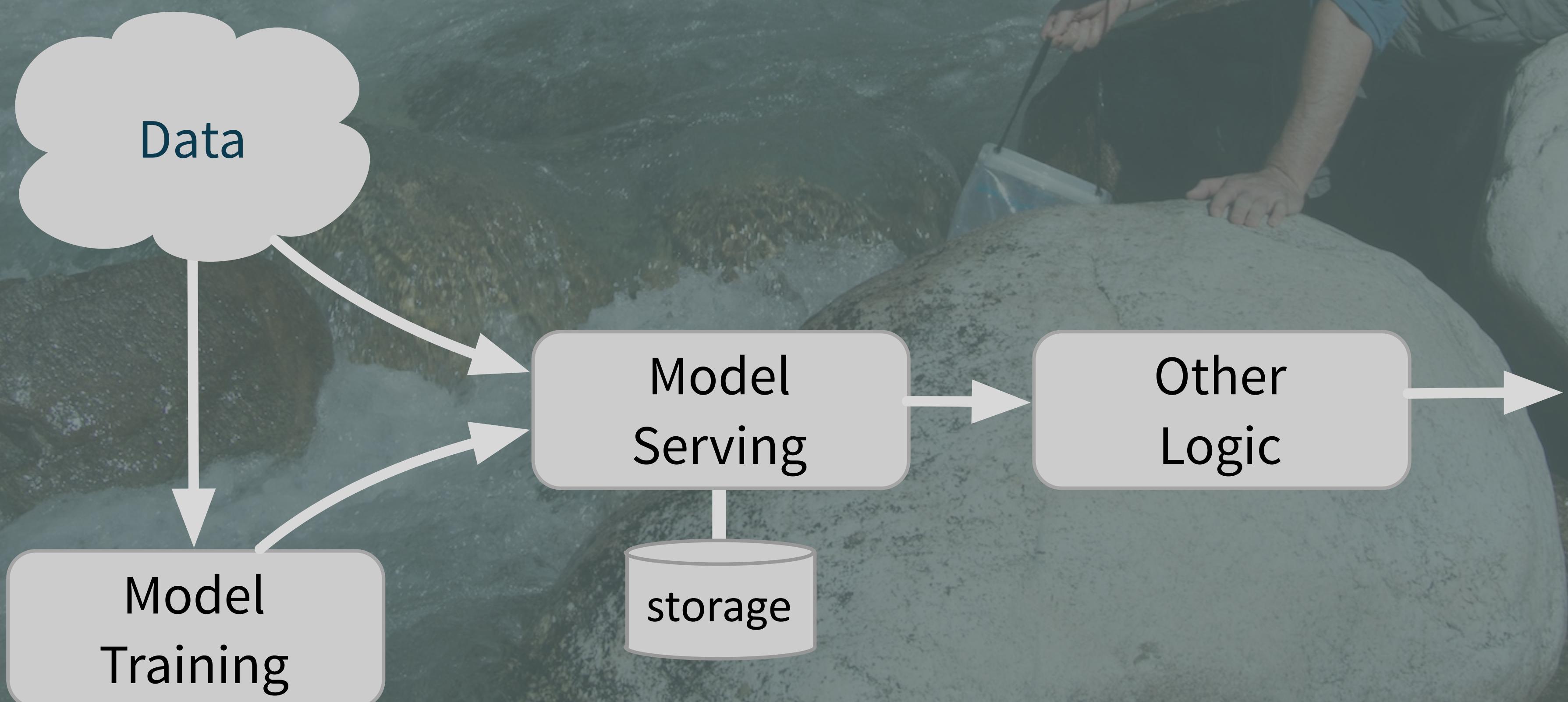
```
val sc = new SparkContext("local[*]", "Inverted Idx")
sc.textFile("data/crawl")
  .map { line => val Array(path, text) = line.split("\t",2);
(path, text)
} flatMap {
  case (path, text) => text.split("""\W+""").map((_, path))
} map {
  case (w, p) => ((w, p), 1)
} reduceByKey {
  case (n1, n2) => n1 + n2
}
```

- Which kinds of data processing?
- SQL?

```
SELECT COUNT(*)  
FROM my-iot-data  
GROUP BY zip-code
```

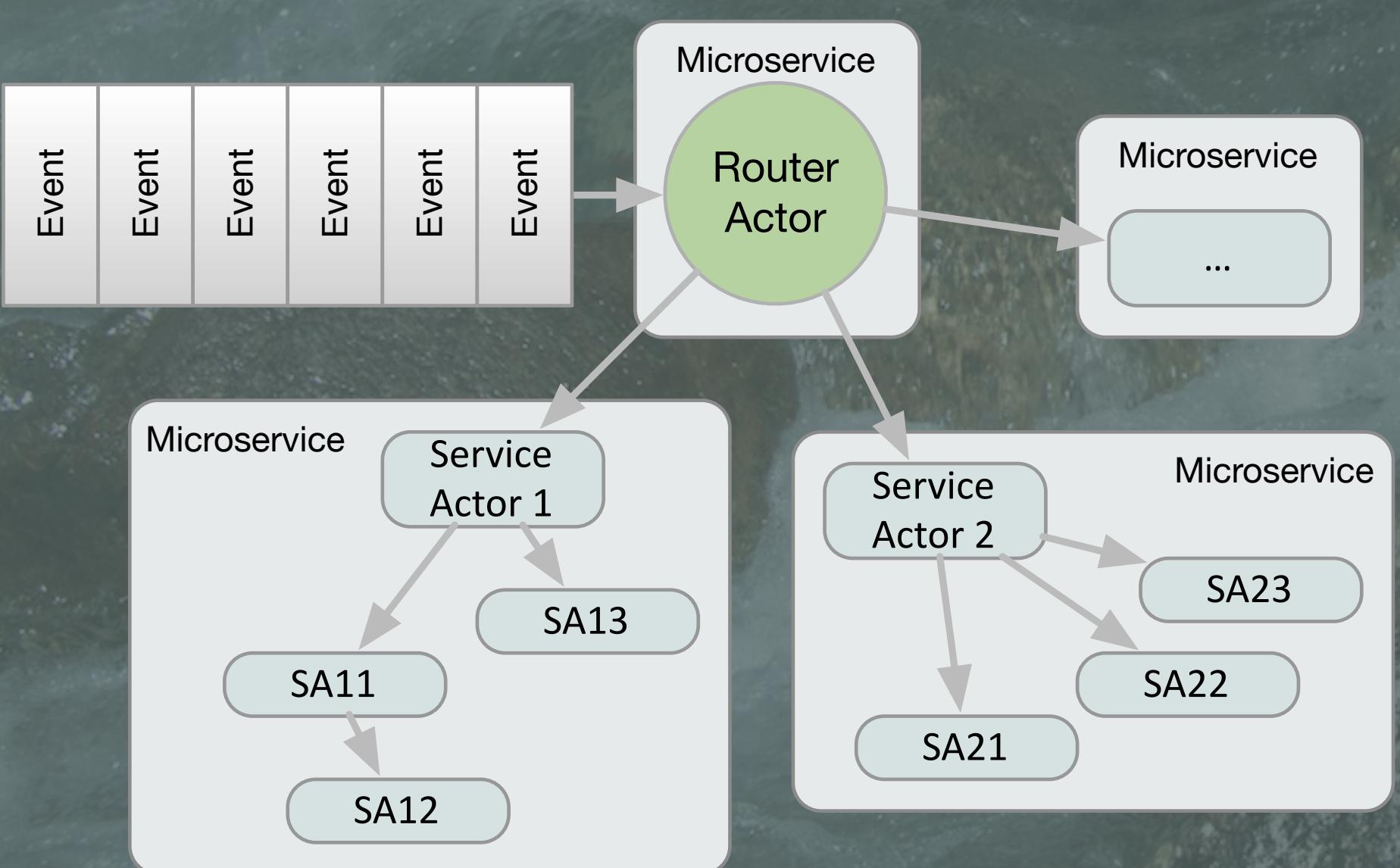
```
val input = spark.read.  
format("parquet").  
stream("my-iot-data")  
  
input.groupBy("zip-code").  
count()
```

- Which kinds of data processing?
- Train and serve ML models?



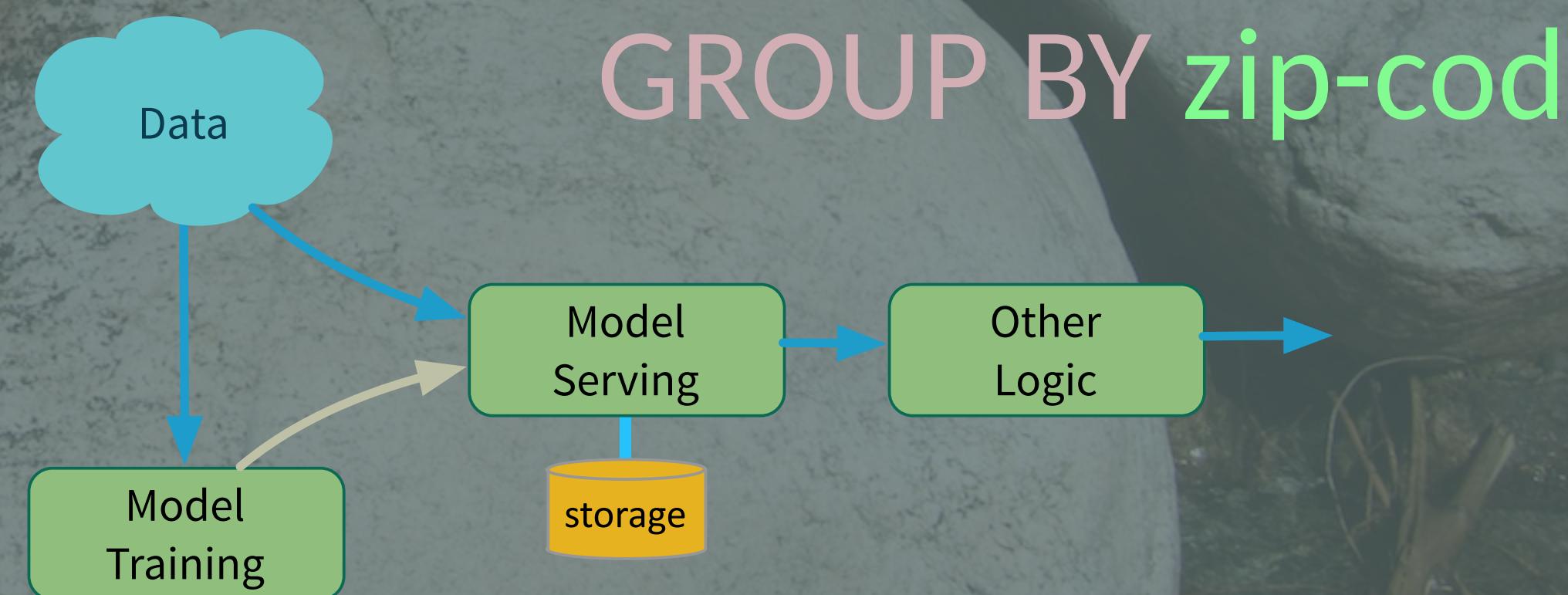
• Process data individually or in bulk?

Event-driven μ-services



Events

"Record-centric" μ-services
SELECT COUNT(*)
FROM my-iot-data
GROUP BY zip-code



Records

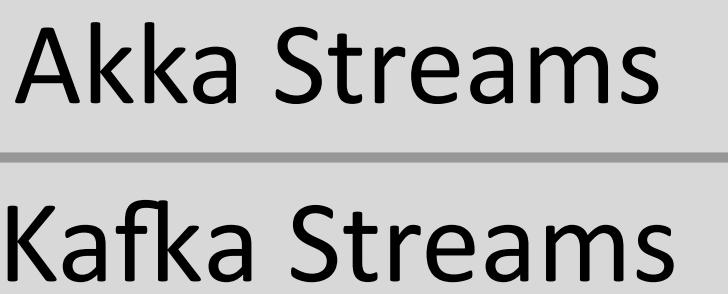
- Preferred application architecture?

- Streaming library in an app?

- or, distributed services

running your job?

Already have a microservices-based, DevOps CI/CD workflow? Stream processing with microservices may fit better into your environment!



Low Latency

Spark

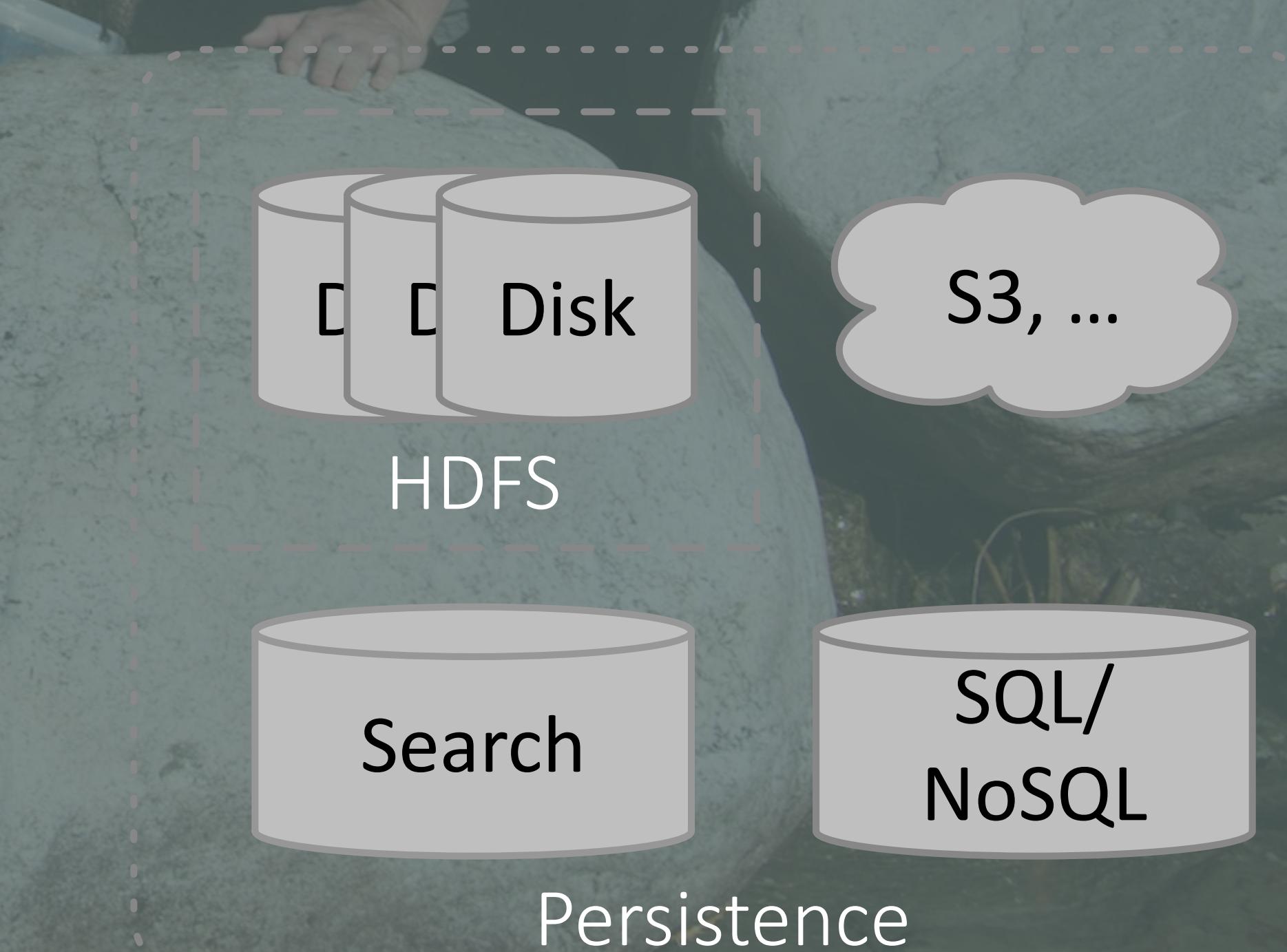
Mini-batch

Spark

...

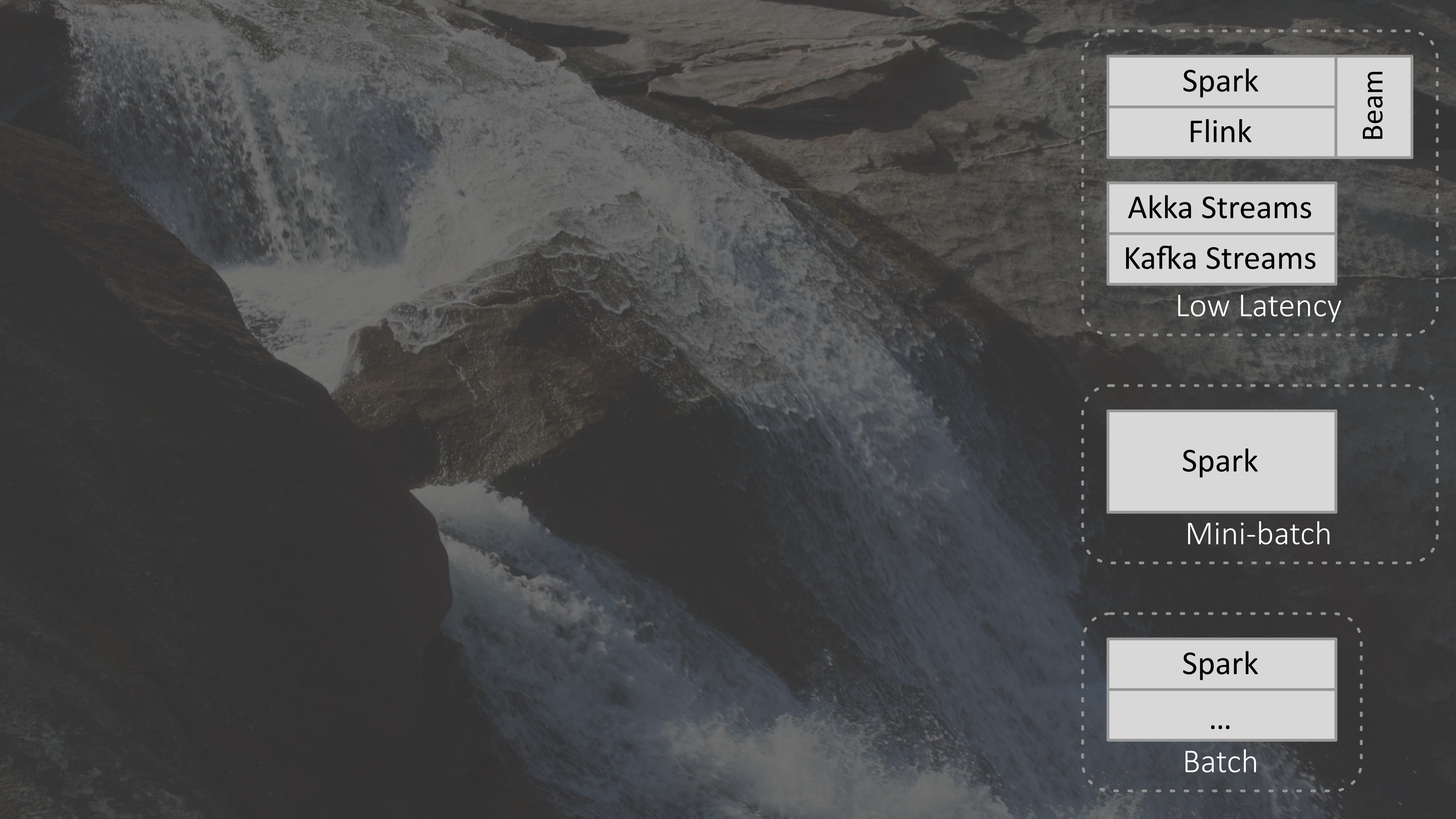
Batch

- Integration with other tools.
- Akka, Flink, & Spark integrate with Databases, Kafka, file systems, REST, ...
- Kafka Streams only read & write Kafka topics.



A close-up photograph of a waterfall cascading down a series of dark, layered rock formations. The water is white and turbulent as it falls and splashes against the rocks. The lighting highlights the texture of the rock and the spray of the water.

Best of Breed Streaming Engines

The background of the diagram is a dark, grainy photograph of a mountain range. The mountains are rugged with patches of snow on their peaks and ridges. The foreground is in deep shadow, while the middle ground shows more detail of the mountain slopes and valleys.

Spark

Flink

Beam

Akka Streams

Kafka Streams

Low Latency

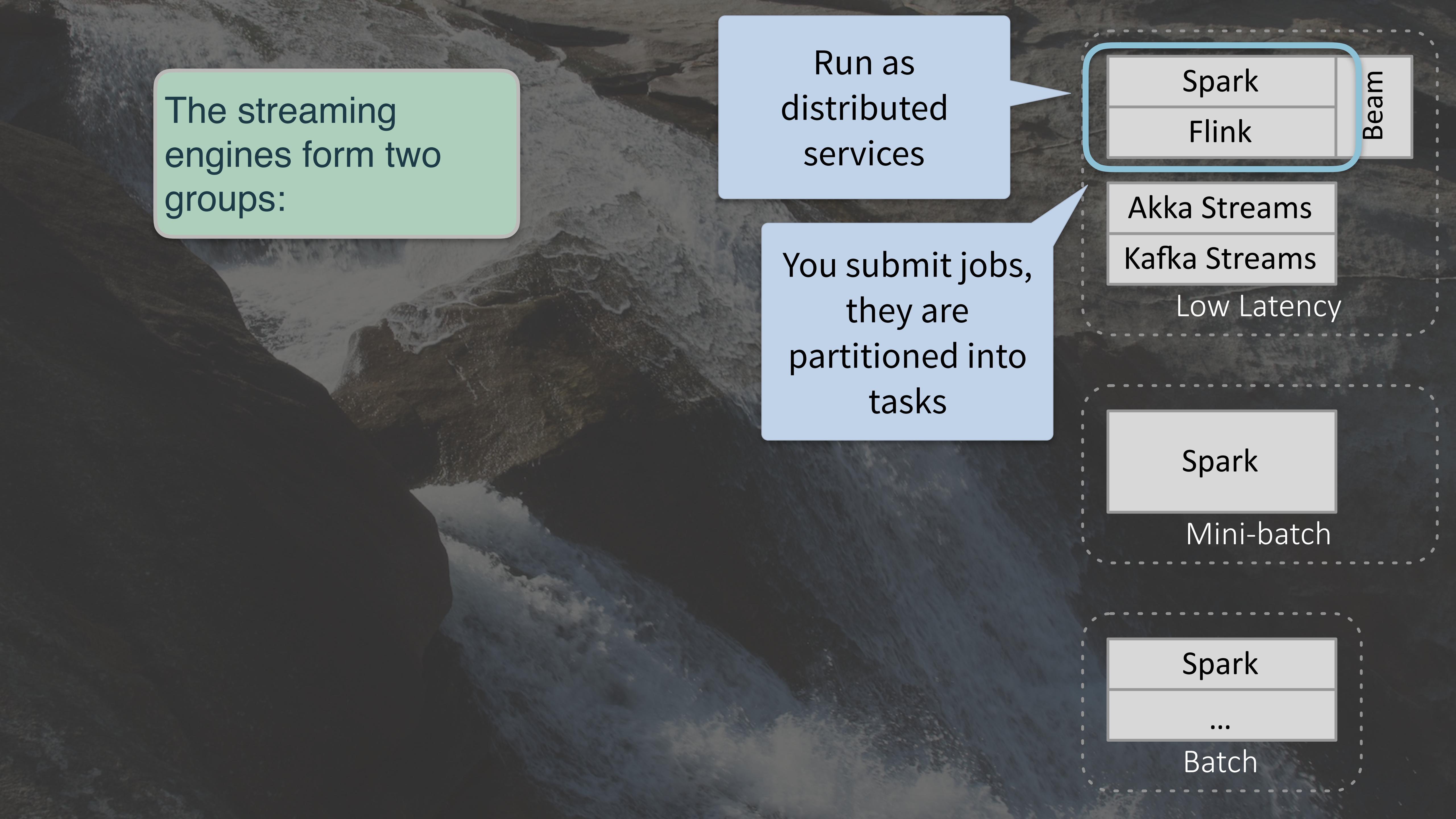
Spark

Mini-batch

Spark

...

Batch



The streaming engines form two groups:

Run as distributed services

You submit jobs,
they are partitioned into tasks

Spark
Flink

Akka Streams

Kafka Streams

Low Latency

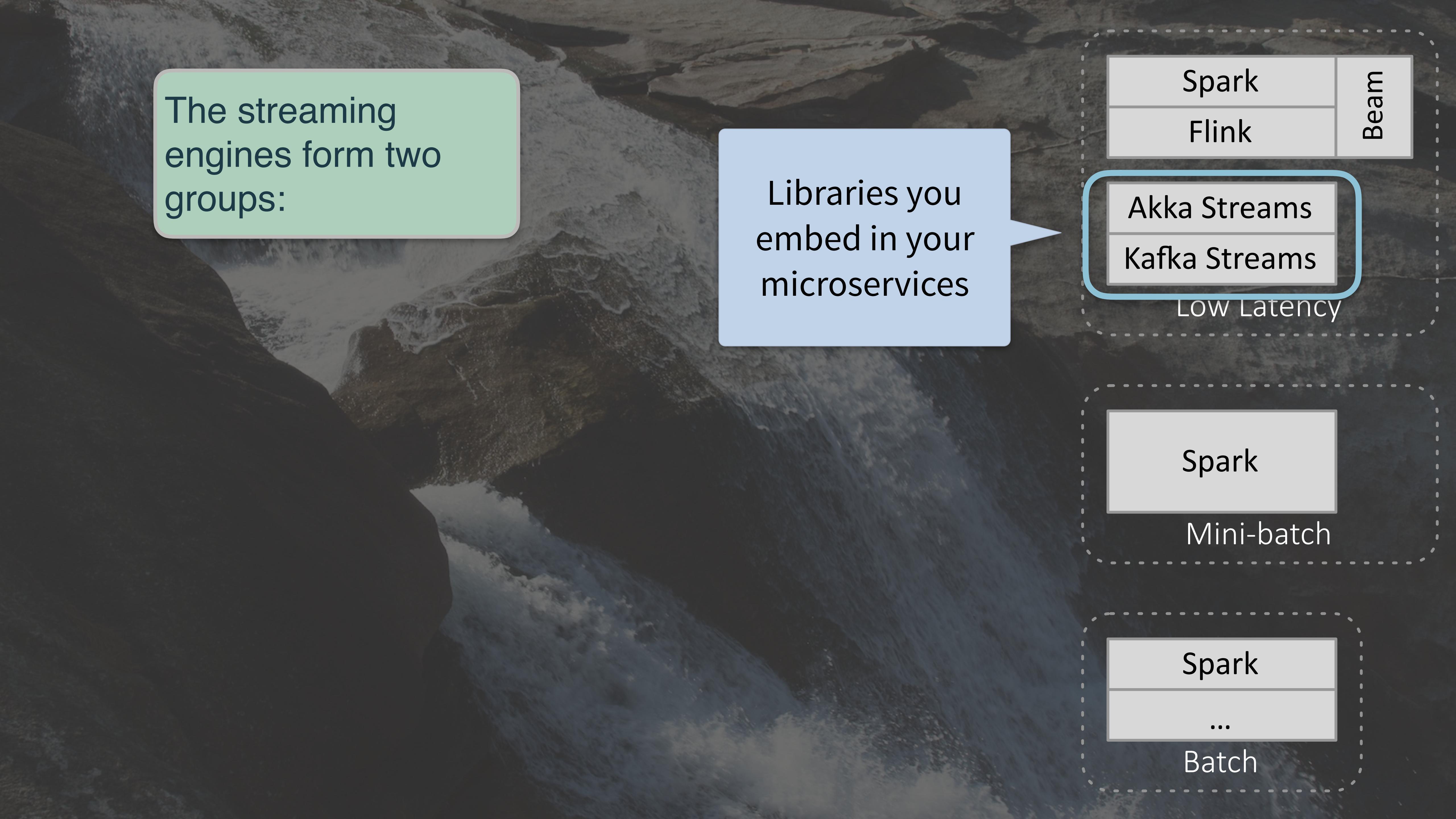
Spark

Mini-batch

Spark

...

Batch



The streaming engines form two groups:

Libraries you embed in your microservices

Spark

Flink

Akka Streams

Kafka Streams

Low Latency

Spark

Mini-batch

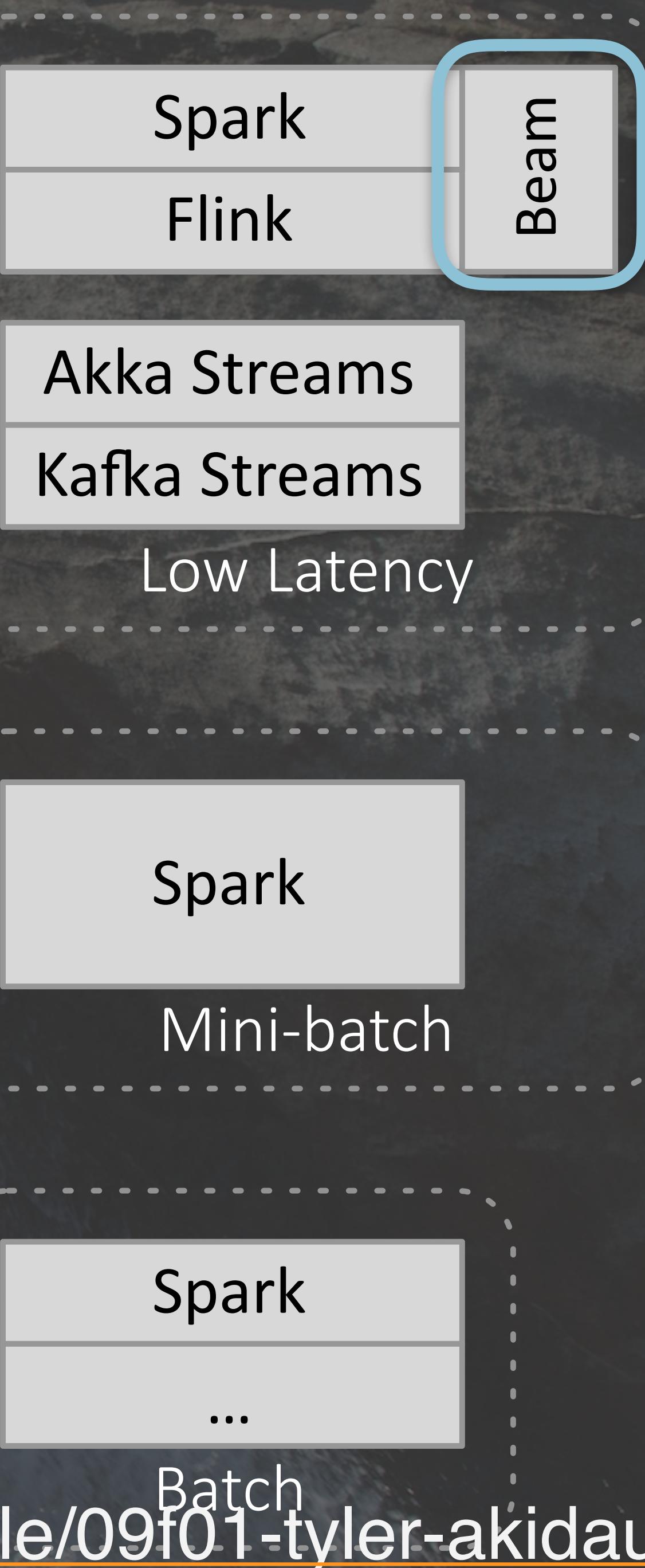
Spark

...

Batch

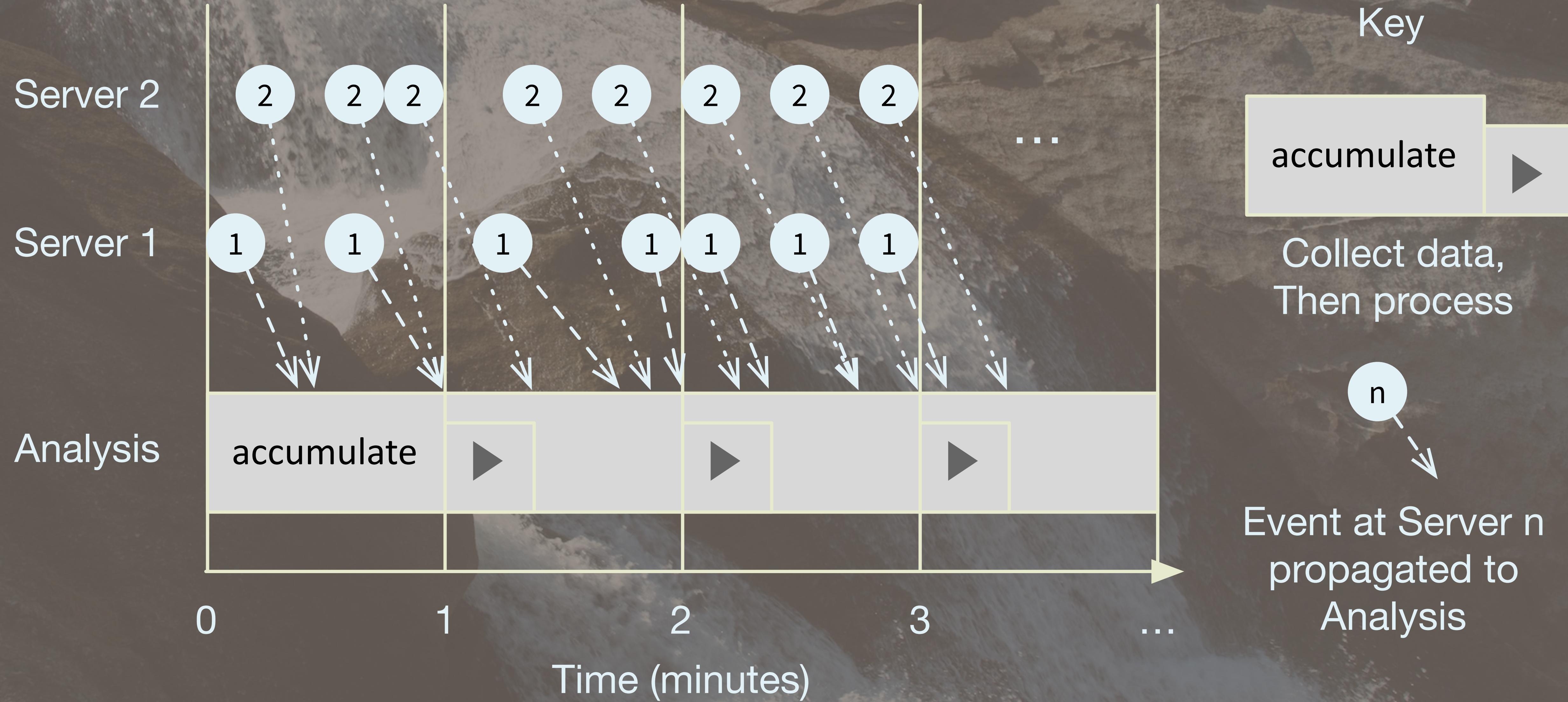
Beam

- Apache Beam
- (Google Dataflow)
- Requires a “runner”
- Most sophisticated streaming semantics

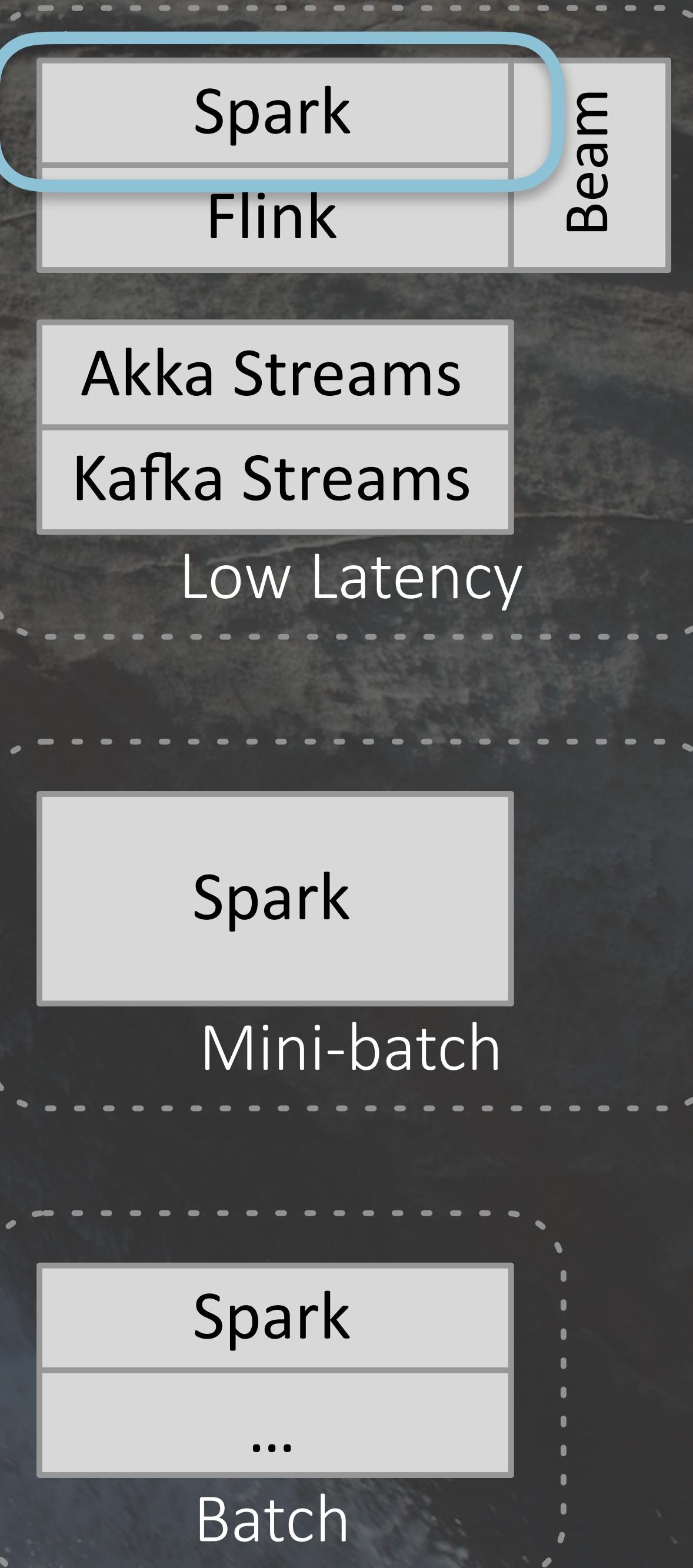


See these blog posts: <https://www.oreilly.com/people/09f01-tyler-akidau>





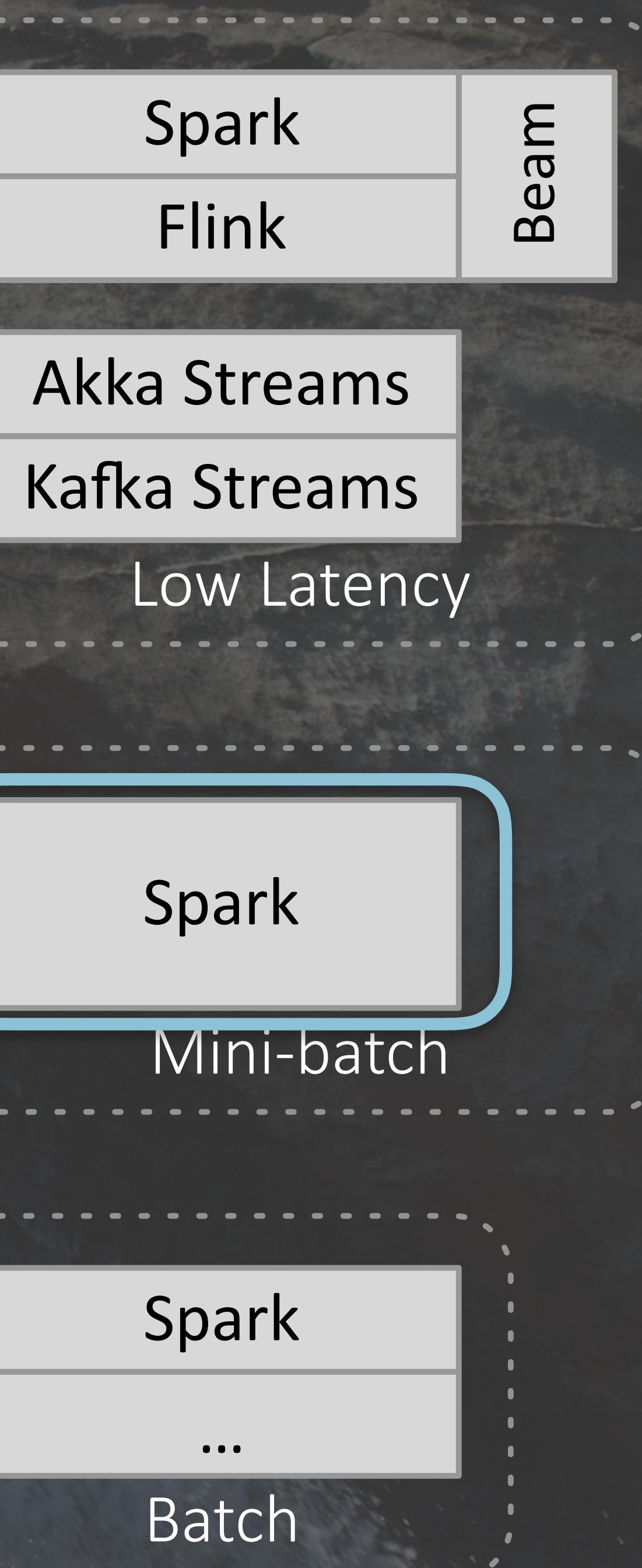
- Spark Structured Streaming
- “Dataset” - SQL
- Millisecond latency
- Ideal for Rich SQL, ML.



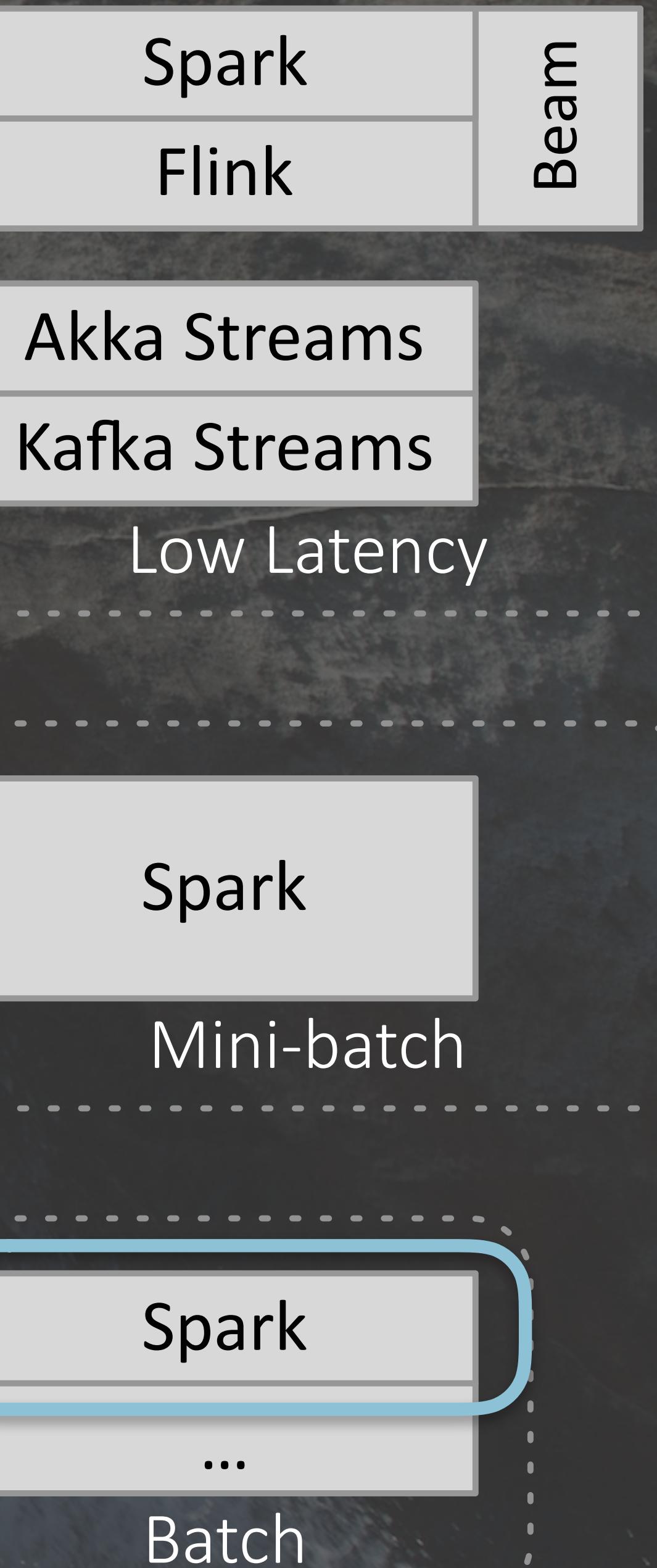
Spark

- Spark Streaming
 - Mini-batch model
 - “RDD” (dataflow) based
 - ~0.5 sec latency
- Original model - obsolete

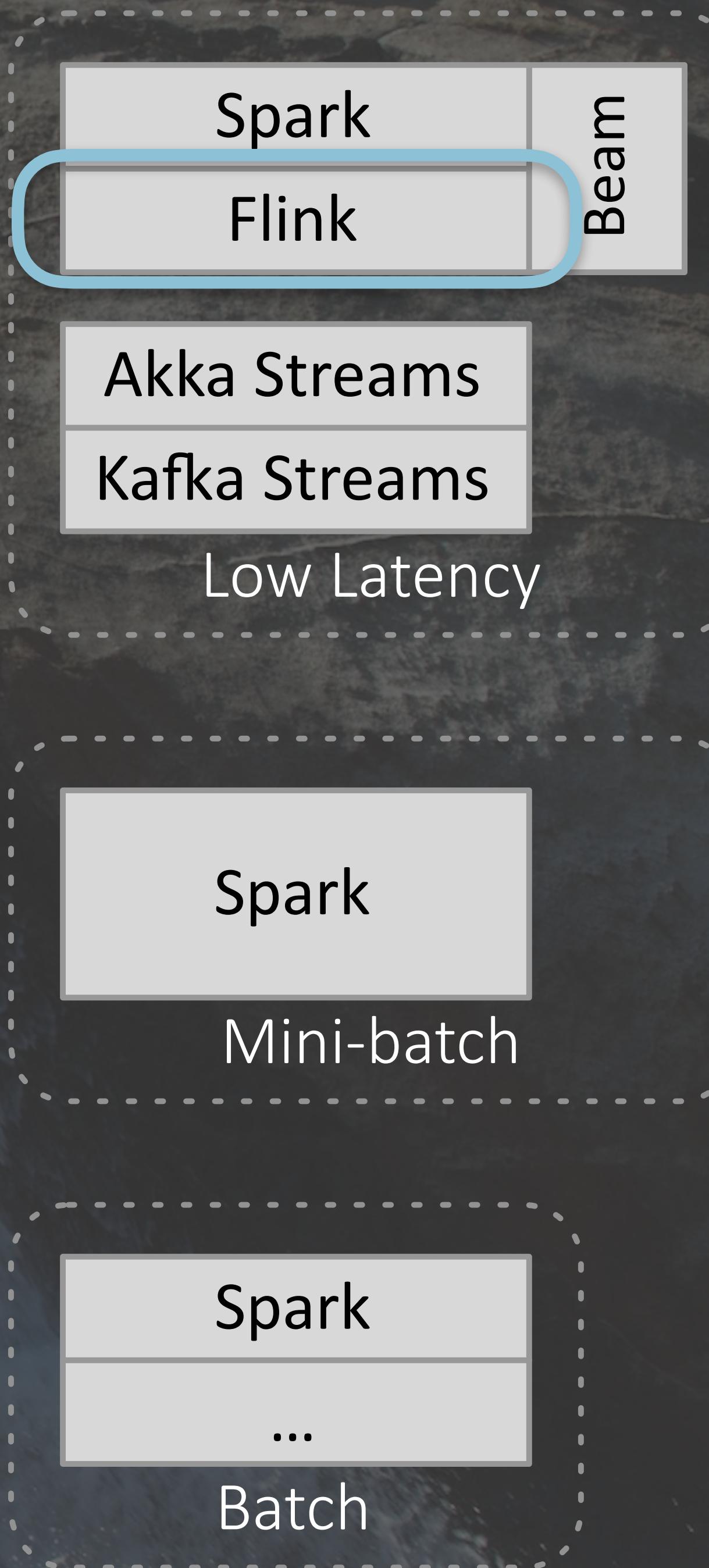
Spark 



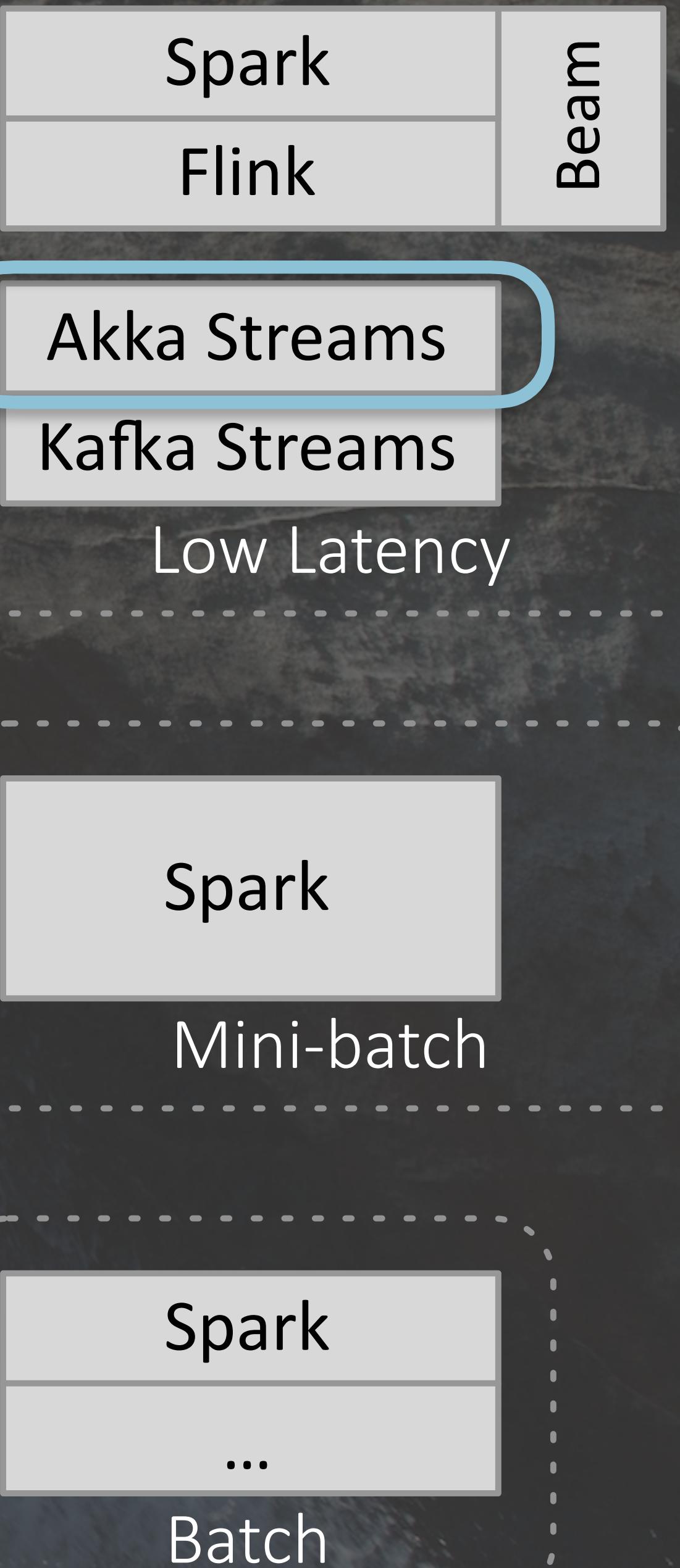
- Spark Batch
- Same Dataset and RDD features as streaming.
- Massive scalability
- Excellent performance



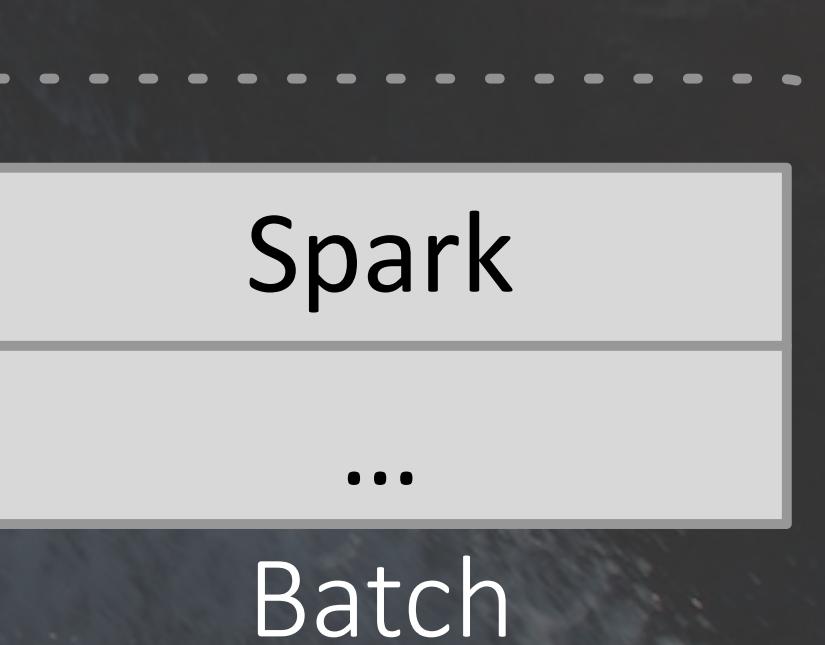
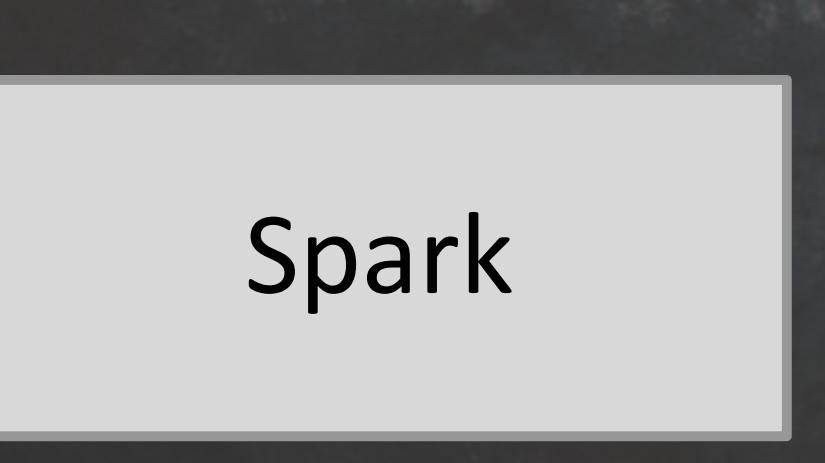
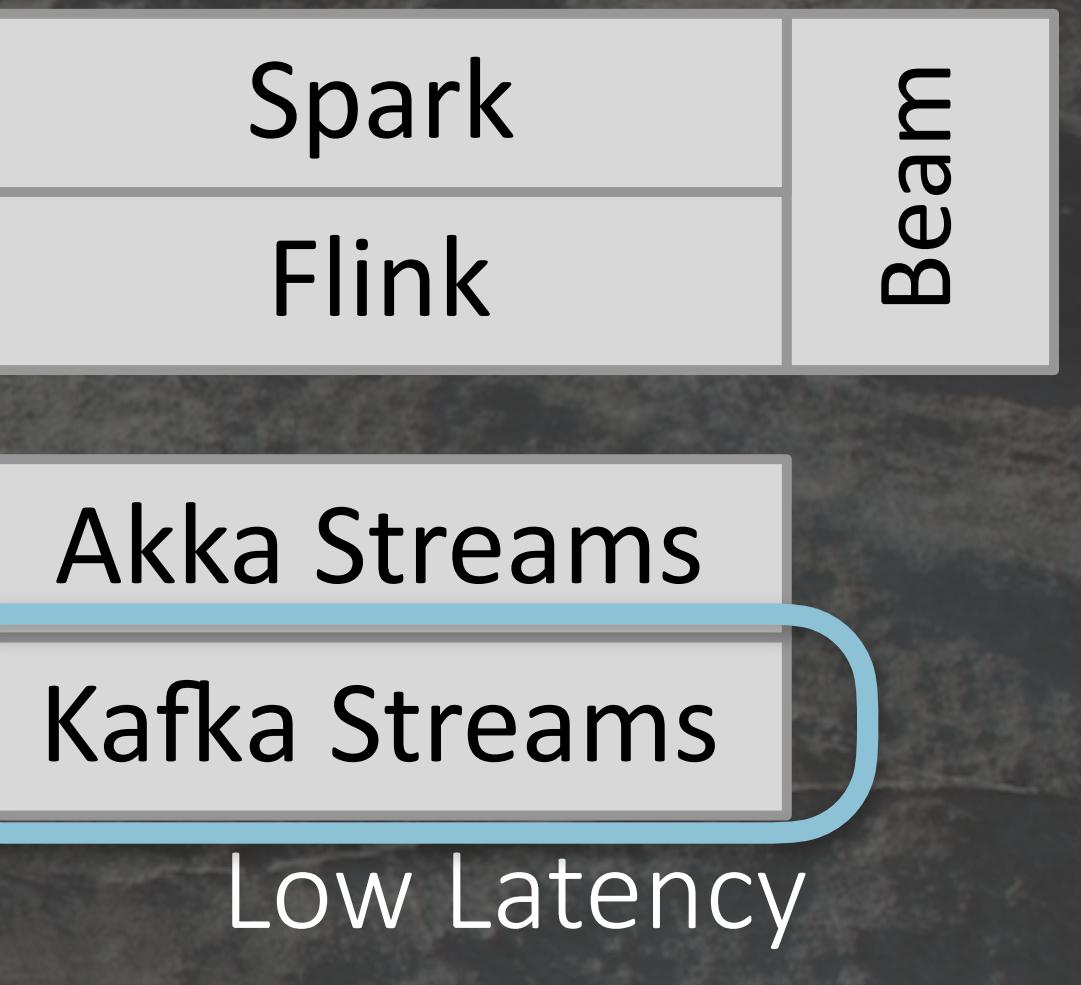
- Apache Flink
- High volume, low latency
- Sophisticated streaming (Beam) semantics
- SQL, evolving ML support



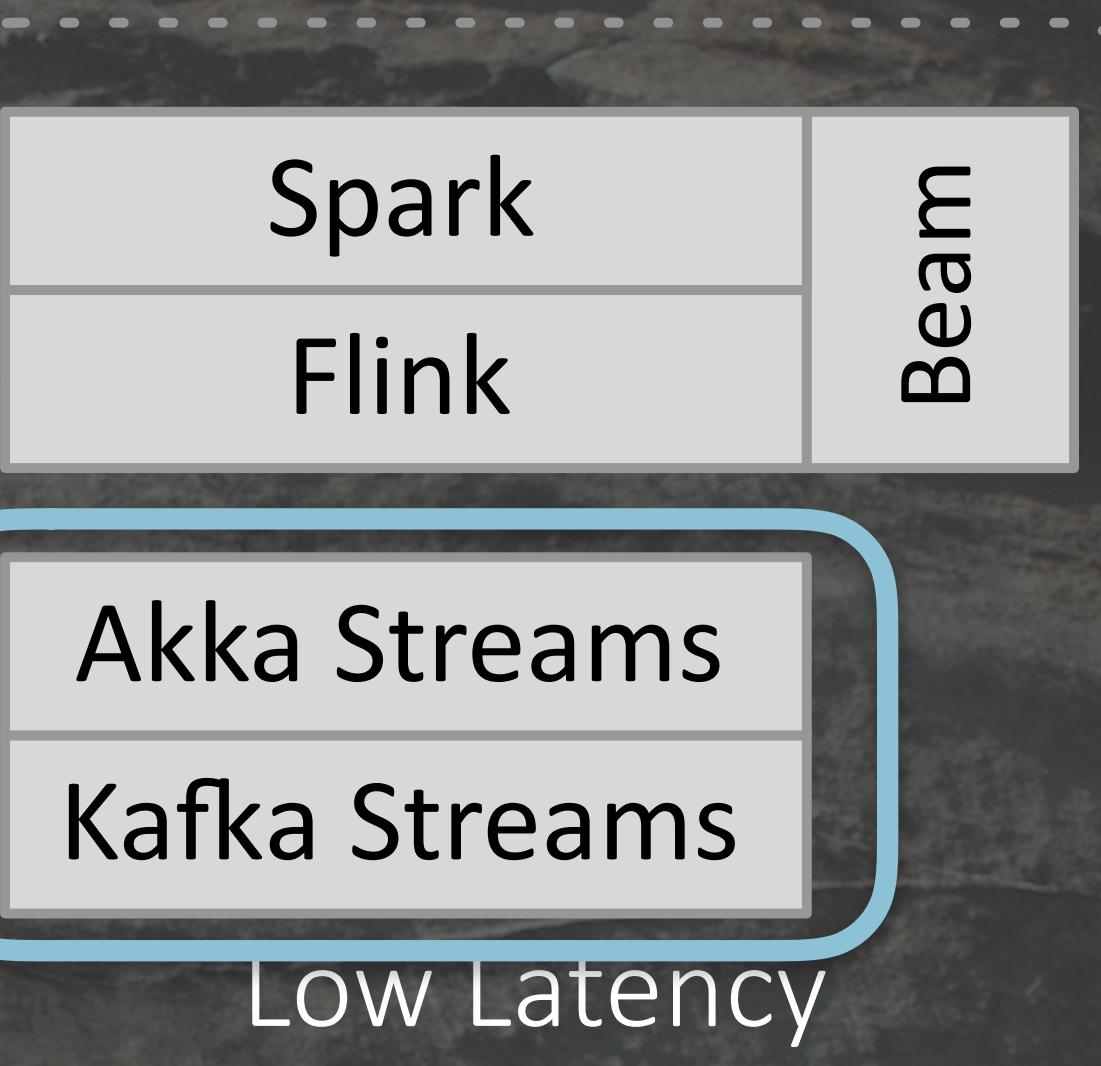
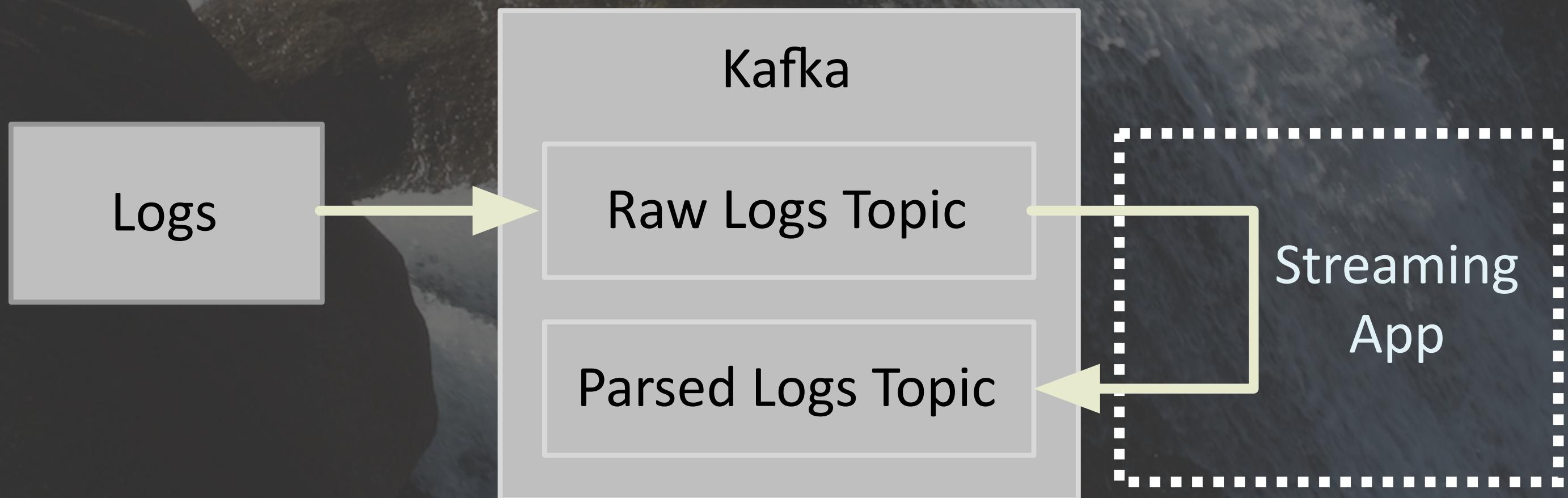
- Akka Streams
- Low latency
- Complex Event Processing
- Efficient, per event
- Mid-volume pipelines



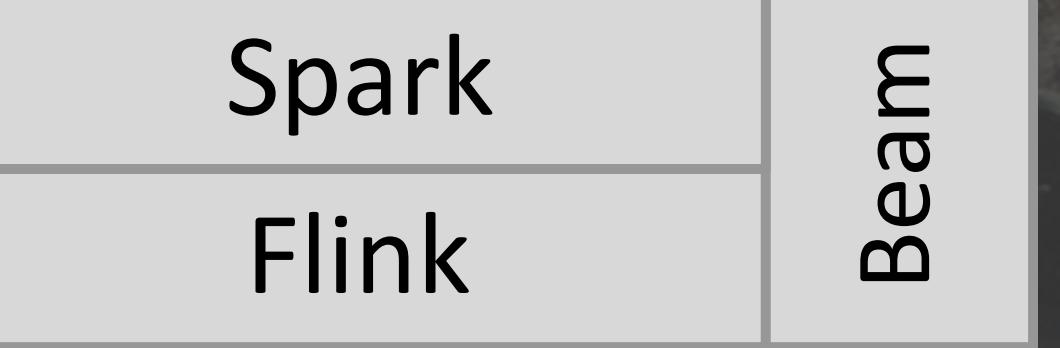
- Kafka Streams
- Low overhead Kafka topic processing
- Ideal for ETL and aggregations



- Akka and Kafka Streams
- “Exactly once” with transactions



- Akka and Kafka Streams
- Neither have built-in support for state checkpointing



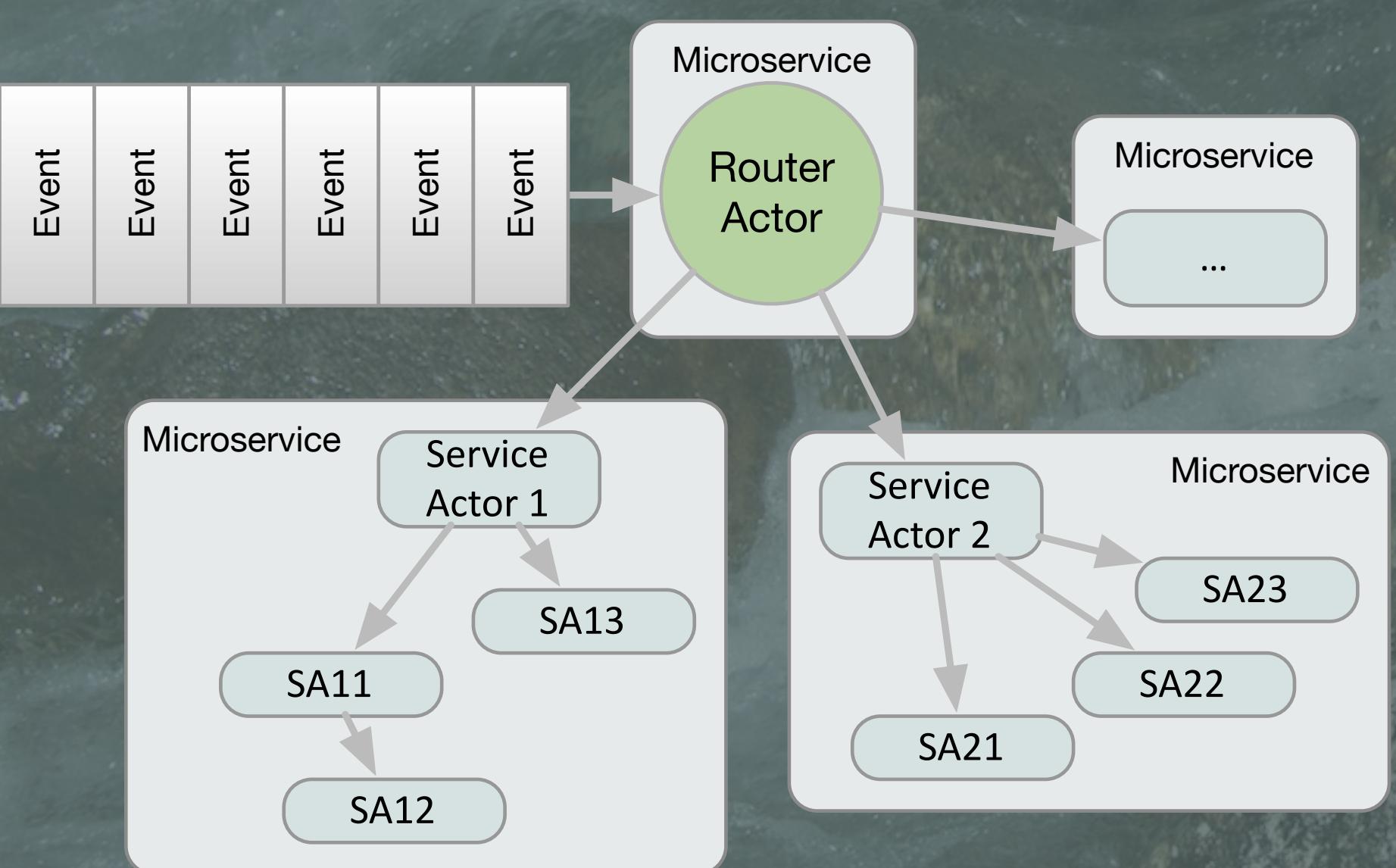
• Process data individually or in bulk?



Each grew out of one end of this

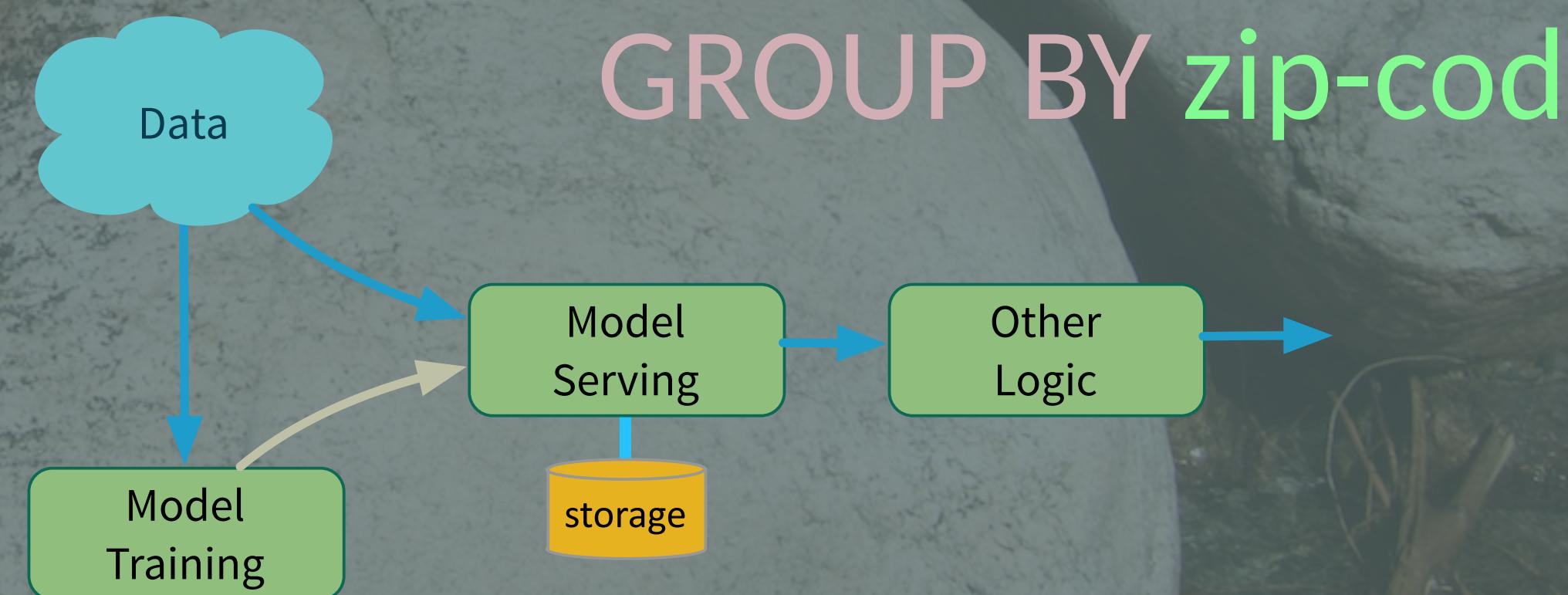


Event-driven μ -services



Events

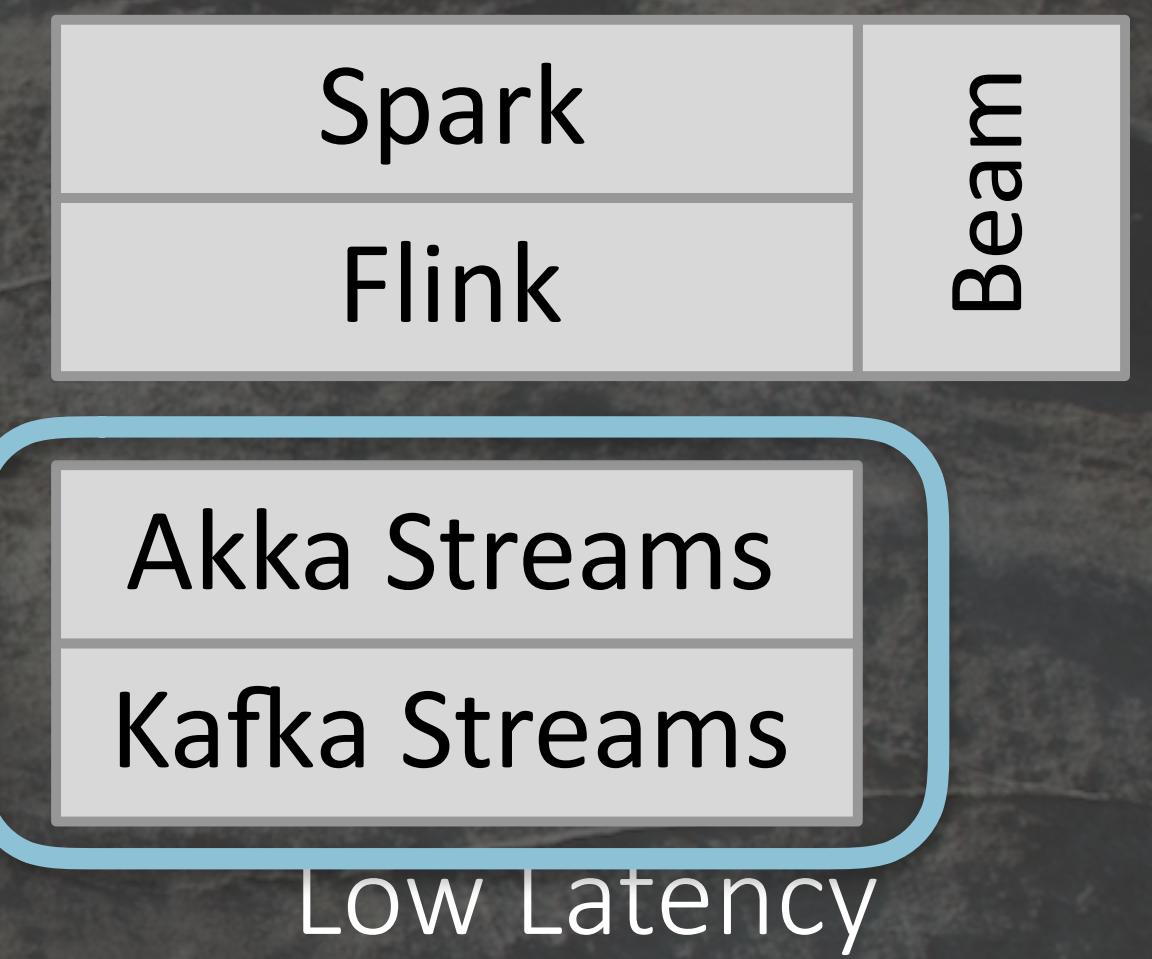
- “Record-centric” μ -services
- `SELECT COUNT(*)
FROM my-iot-data
GROUP BY zip-code`



Records

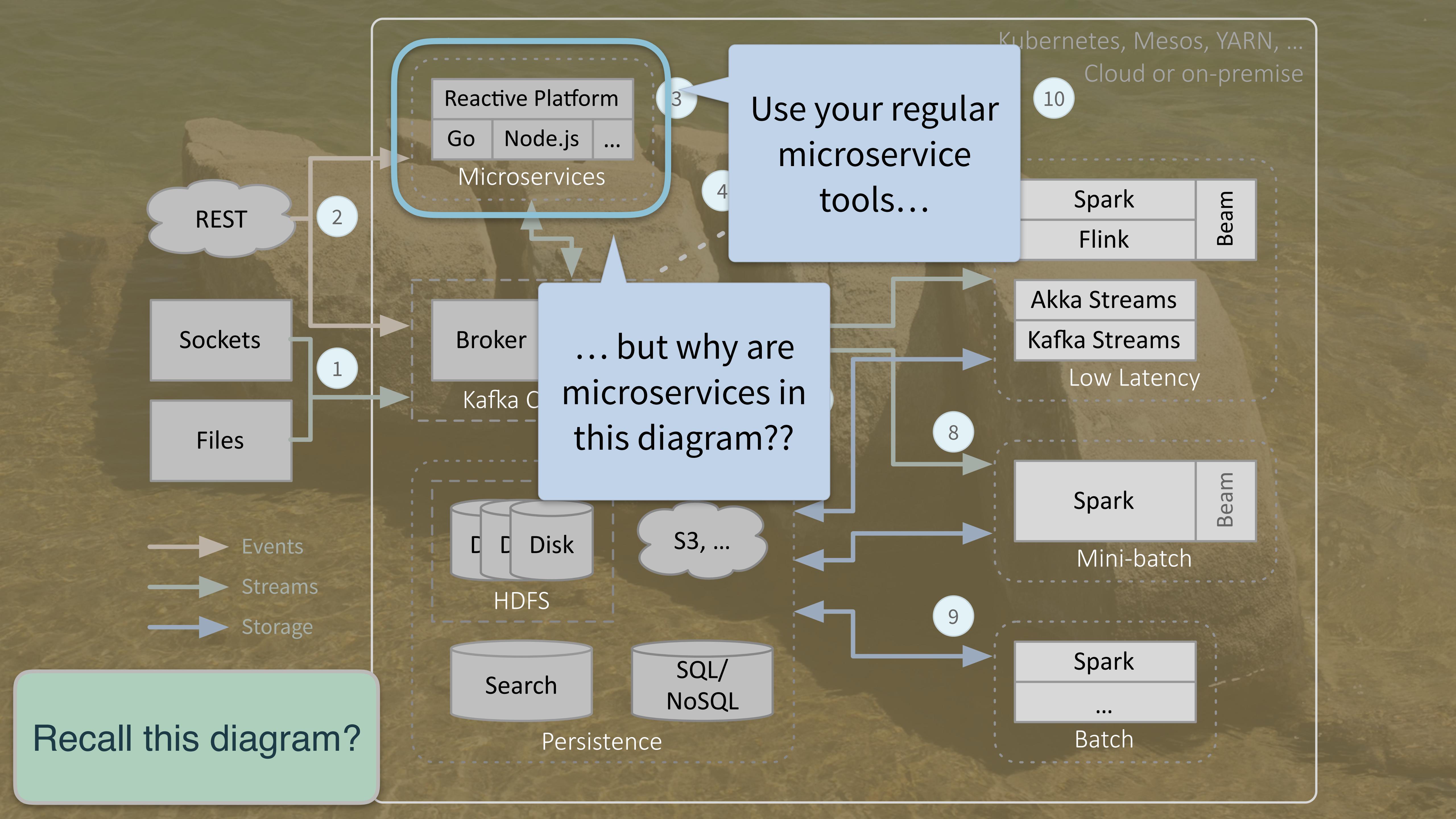
• Akka Streams vs. Kafka Streams talk

- Also at polyglotprogramming.com/talks/

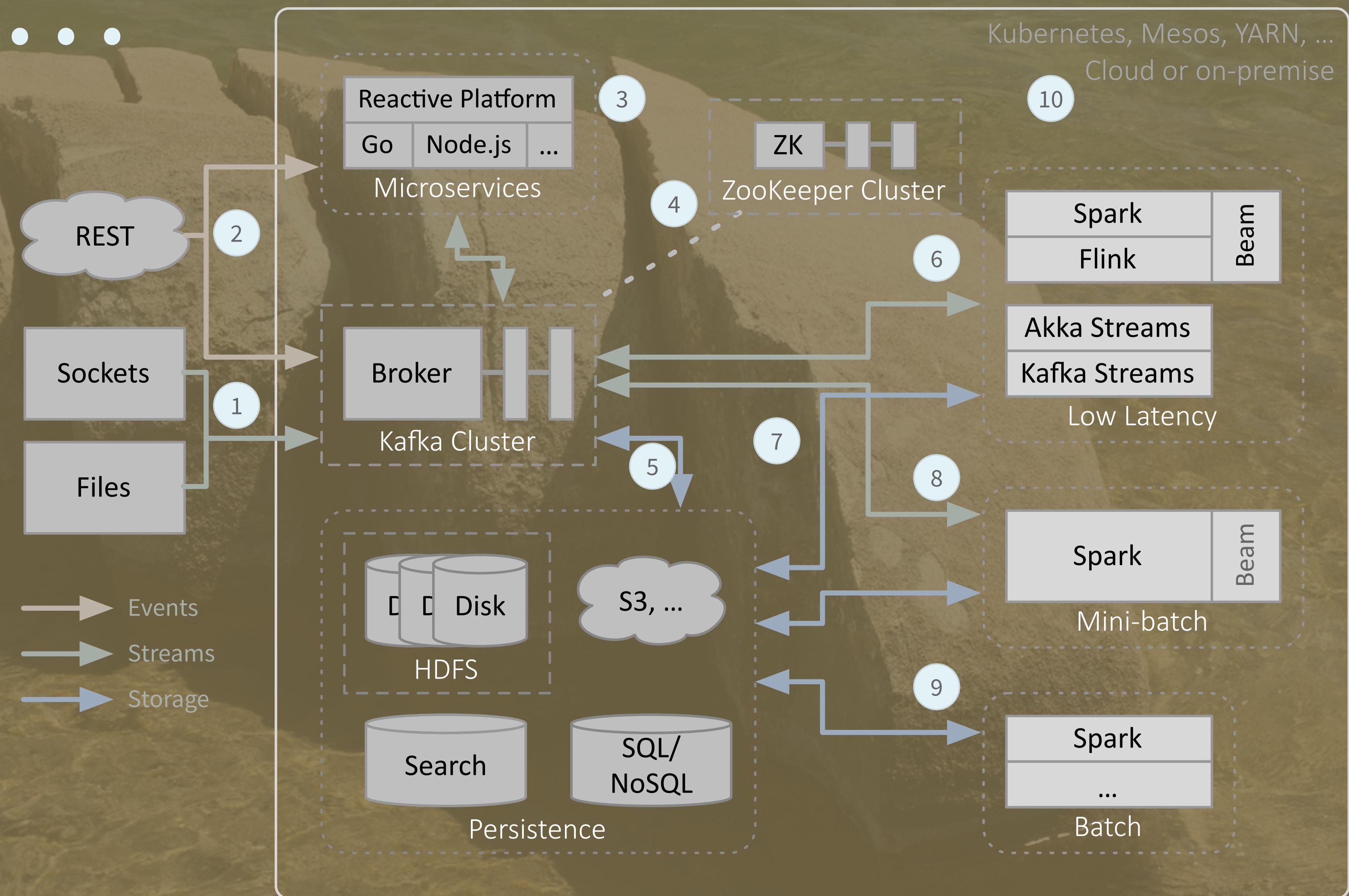


A large, light-colored rock formation, possibly granite, is partially submerged in clear, shallow water. The rocks are angular and have sharp edges. The water is very clear, allowing a view of the sandy bottom and the rocky bed below. The lighting suggests it's a sunny day.

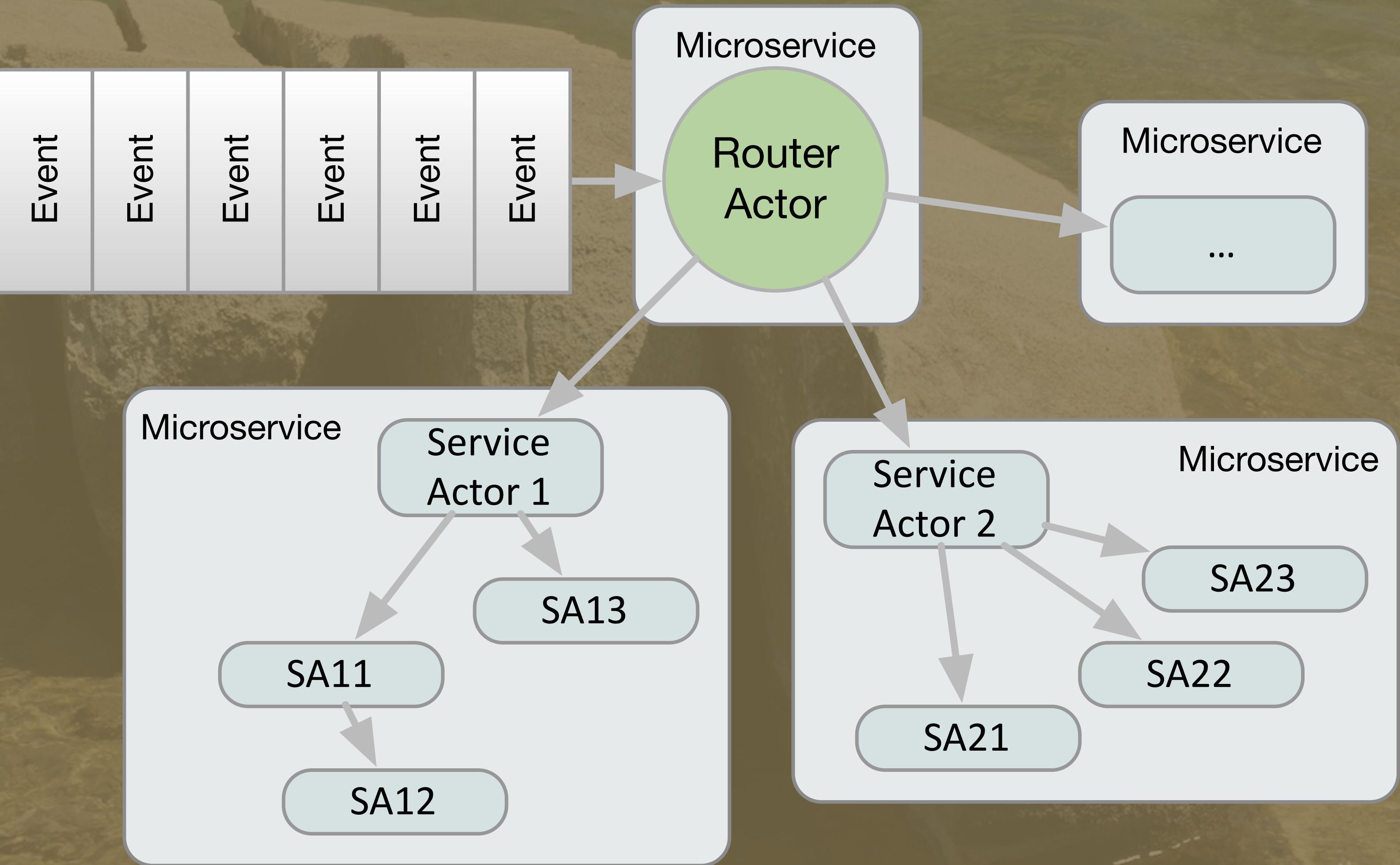
Microservices and Fast Data



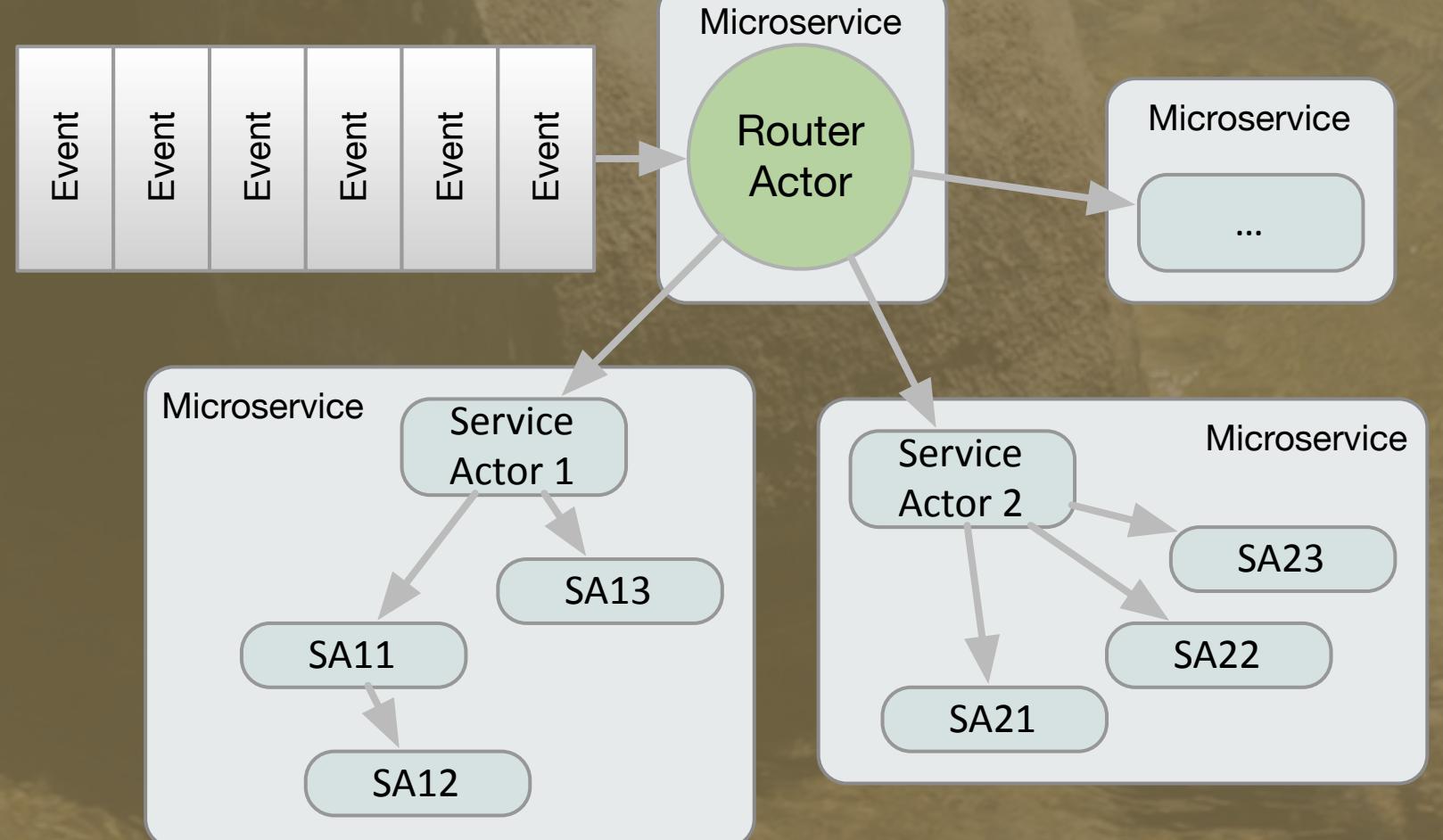
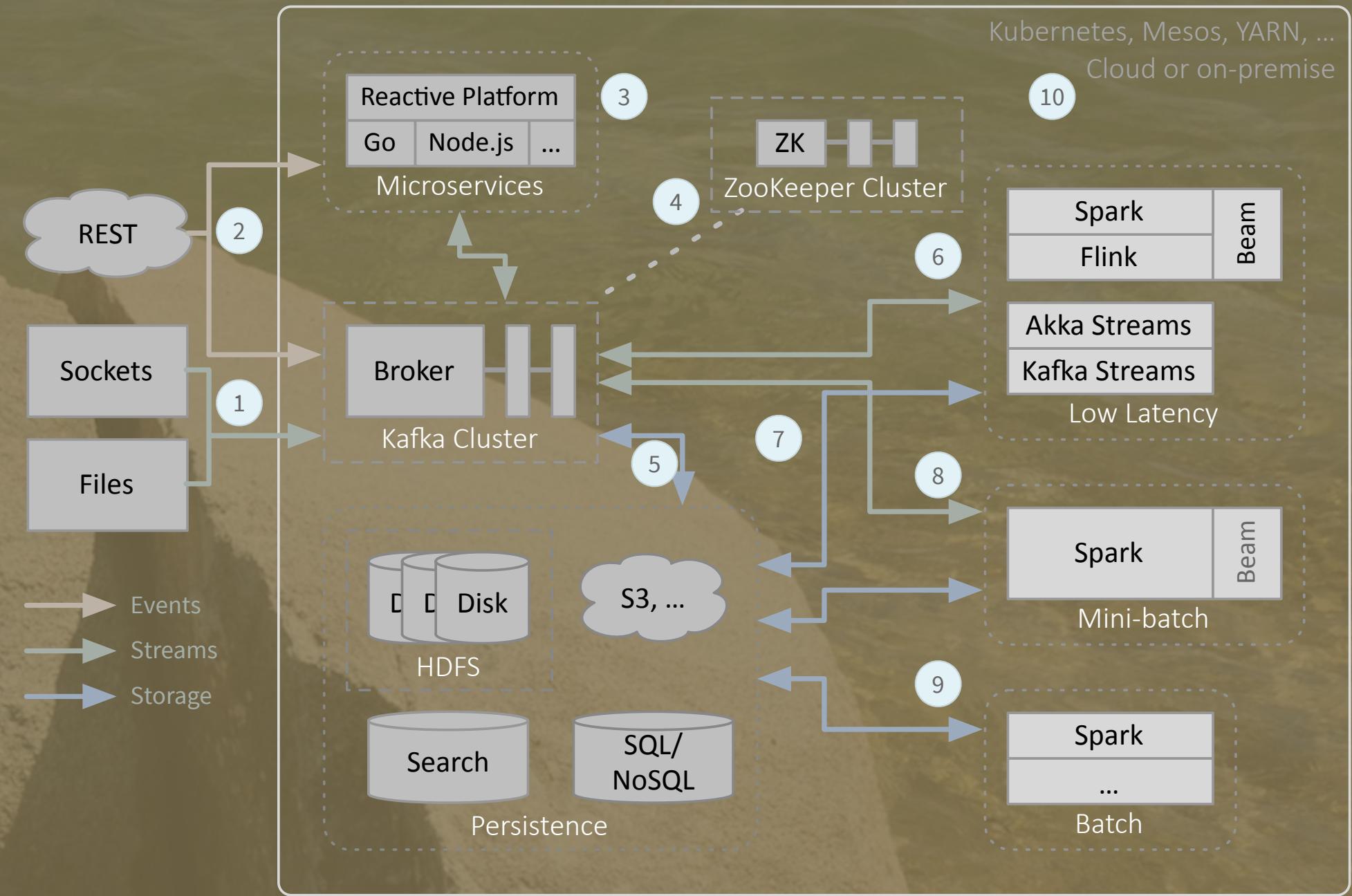
How is this...



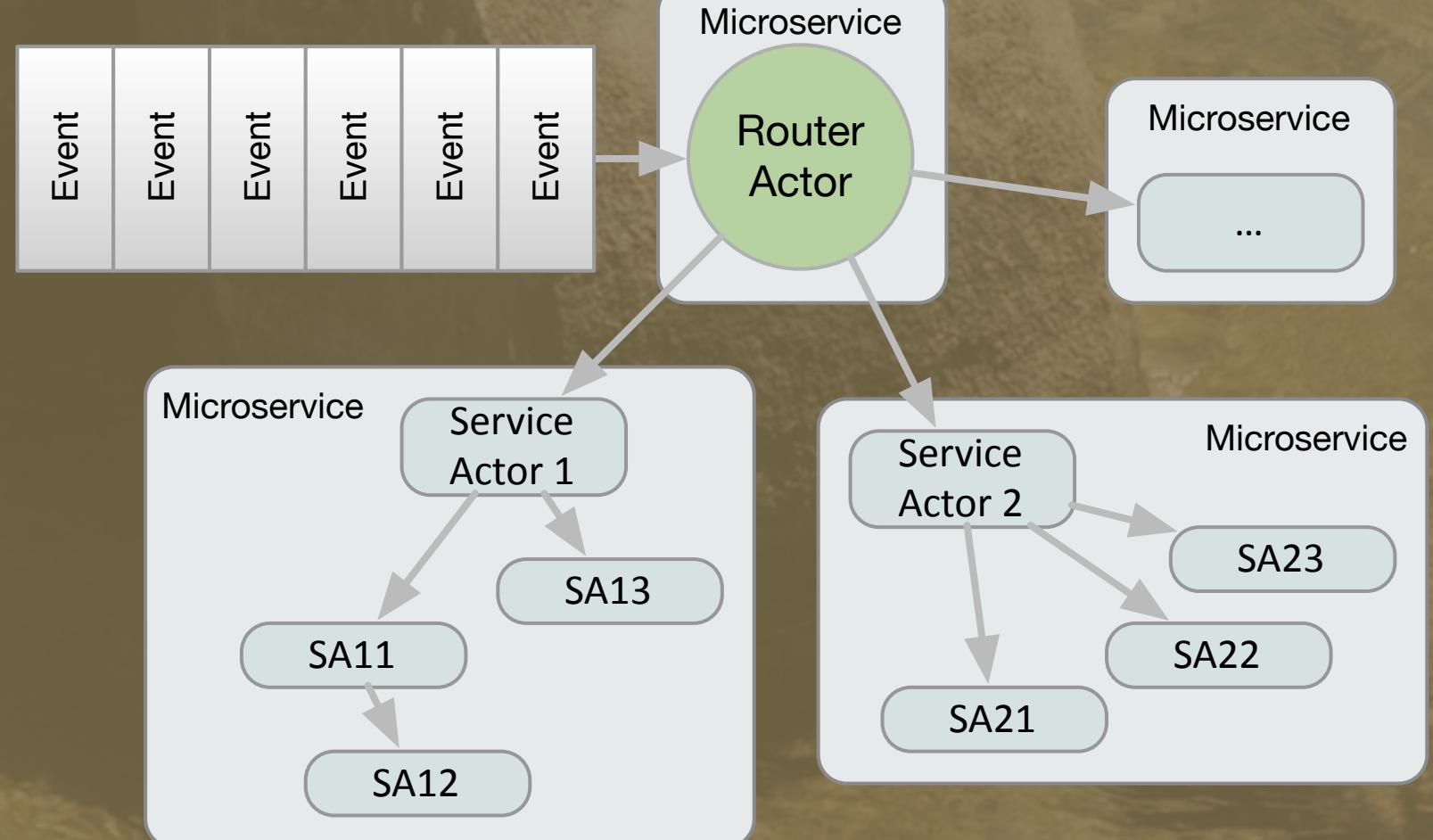
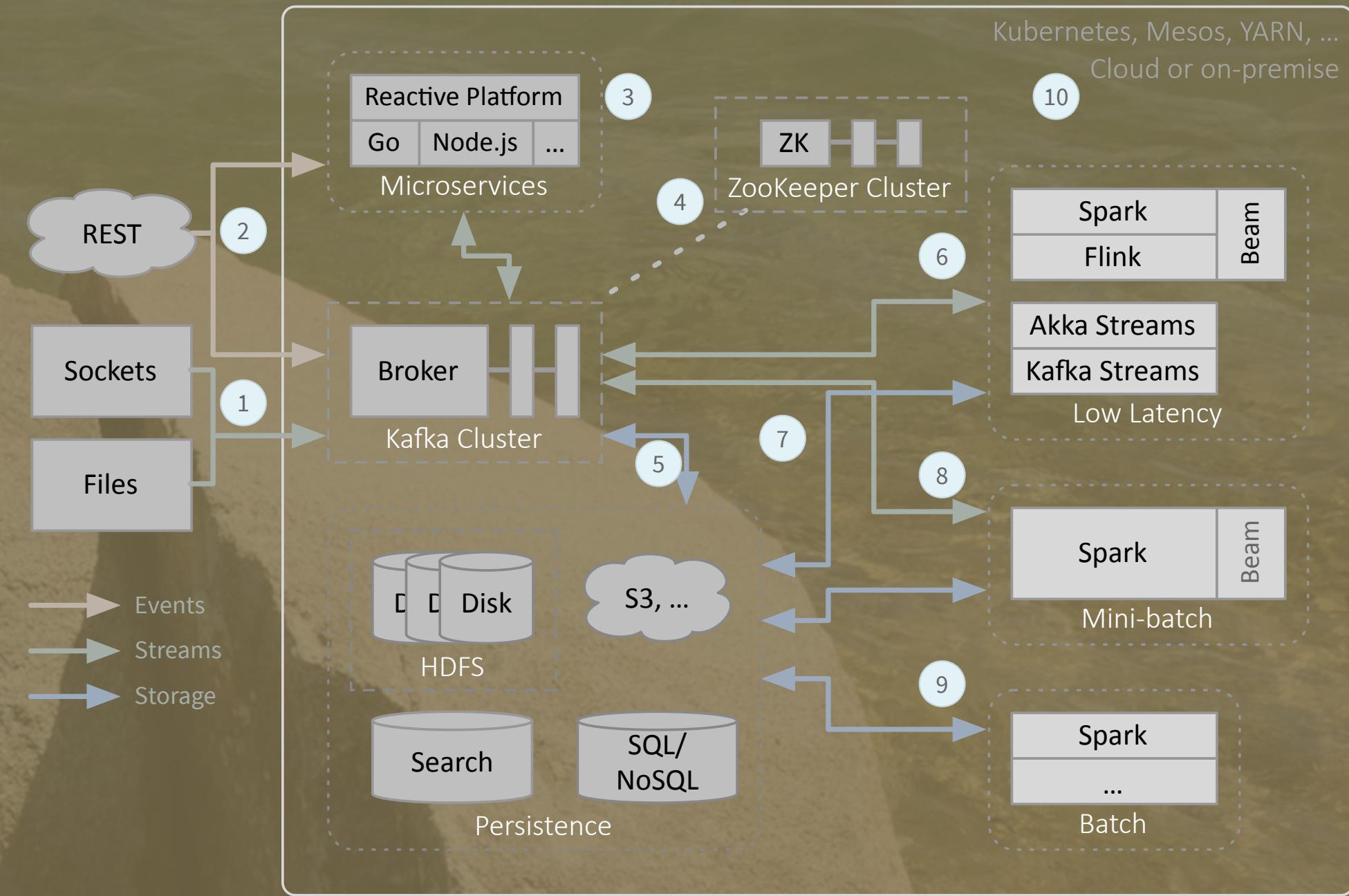
... like this?



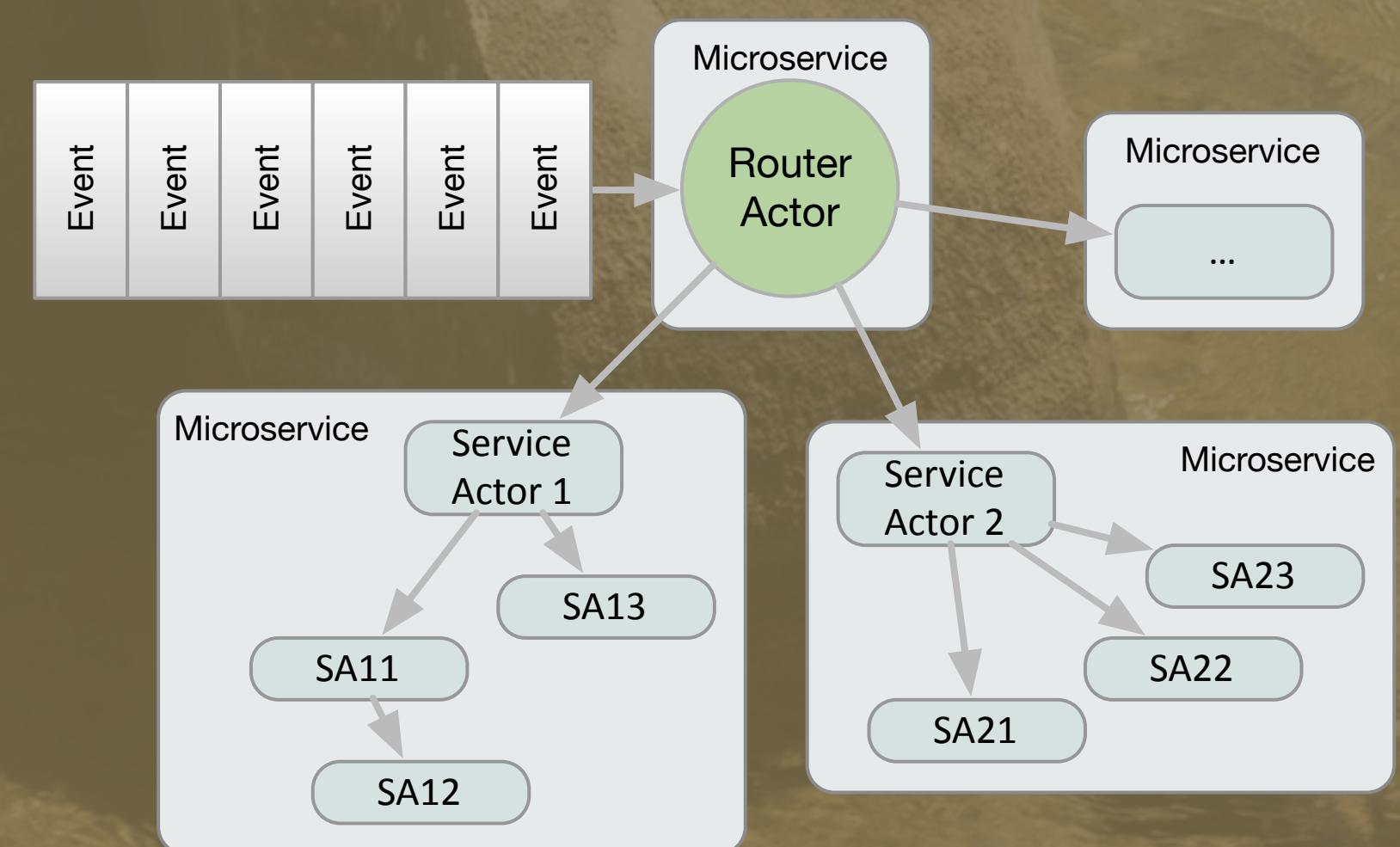
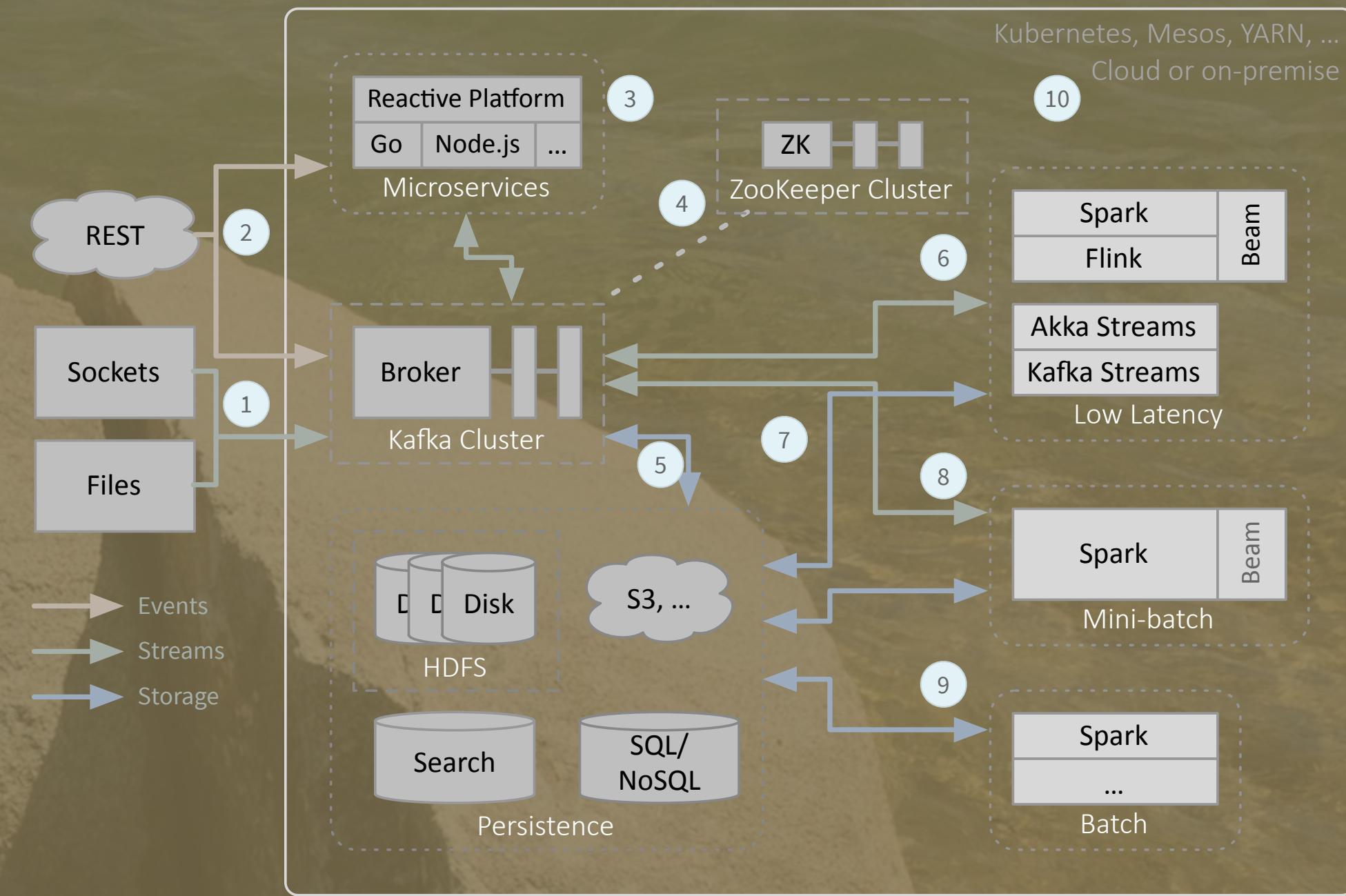
- A data app / microservice:
- A single responsibility.



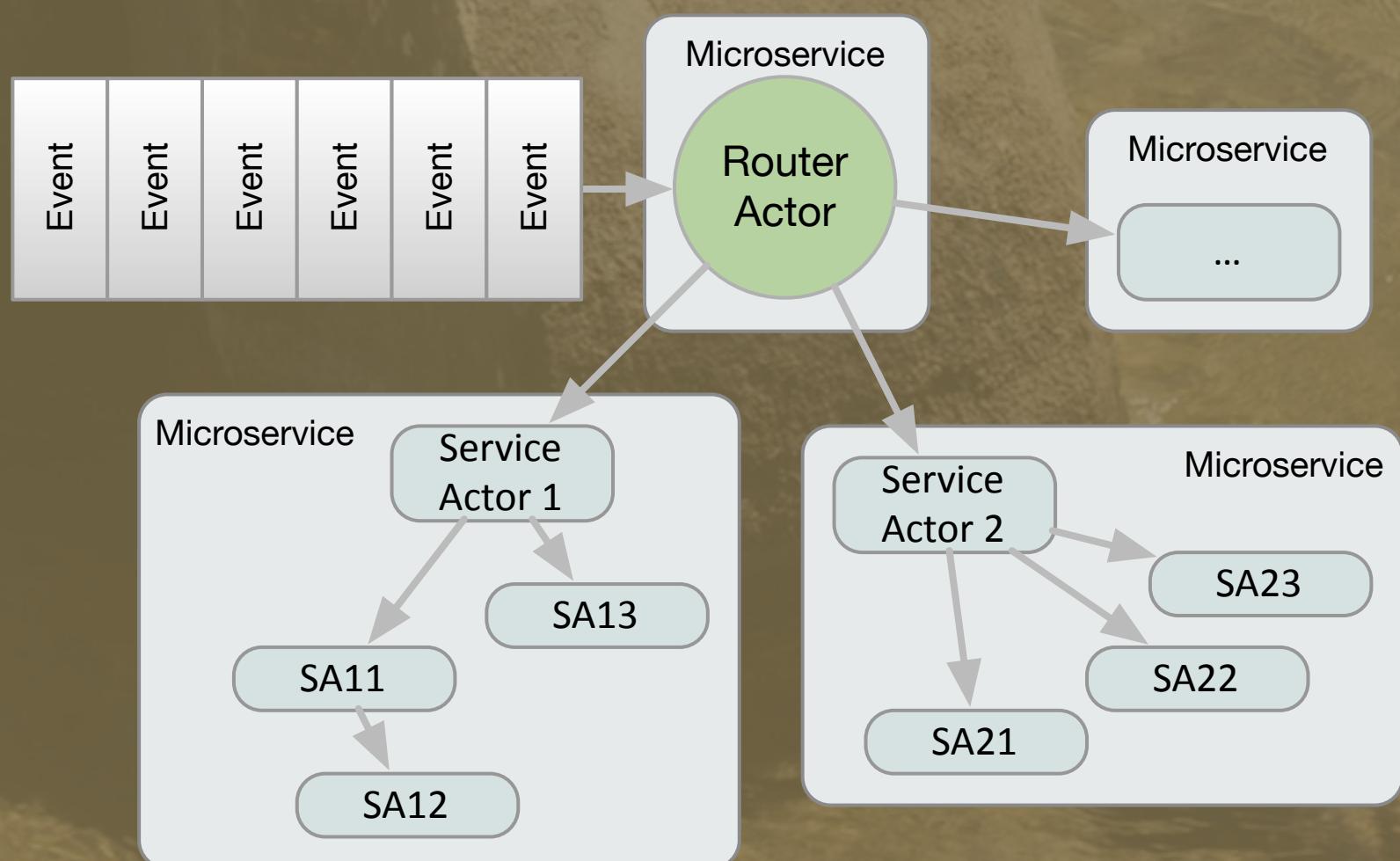
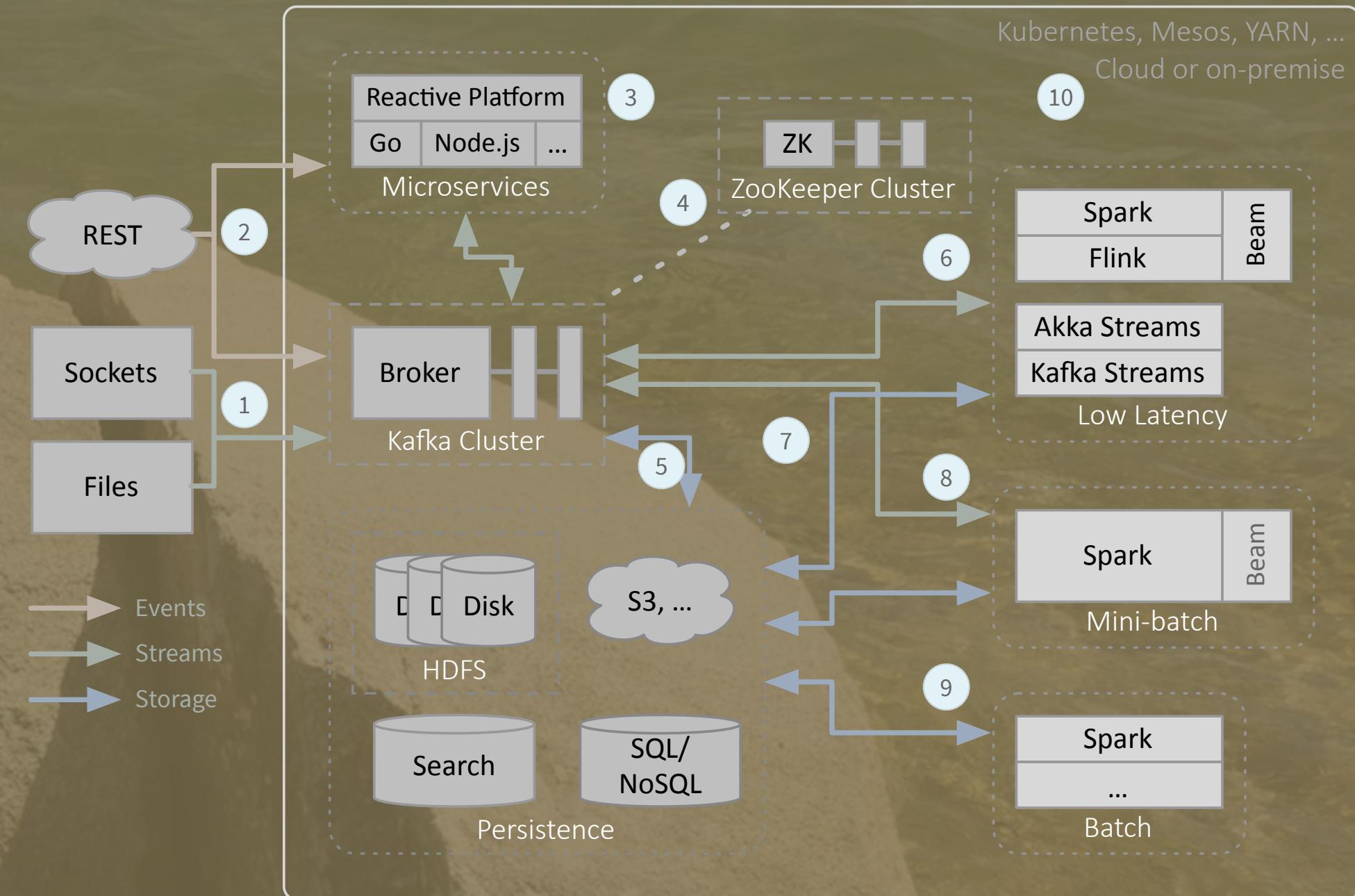
- A data app / microservice:
- A single responsibility.
- The input never ends.



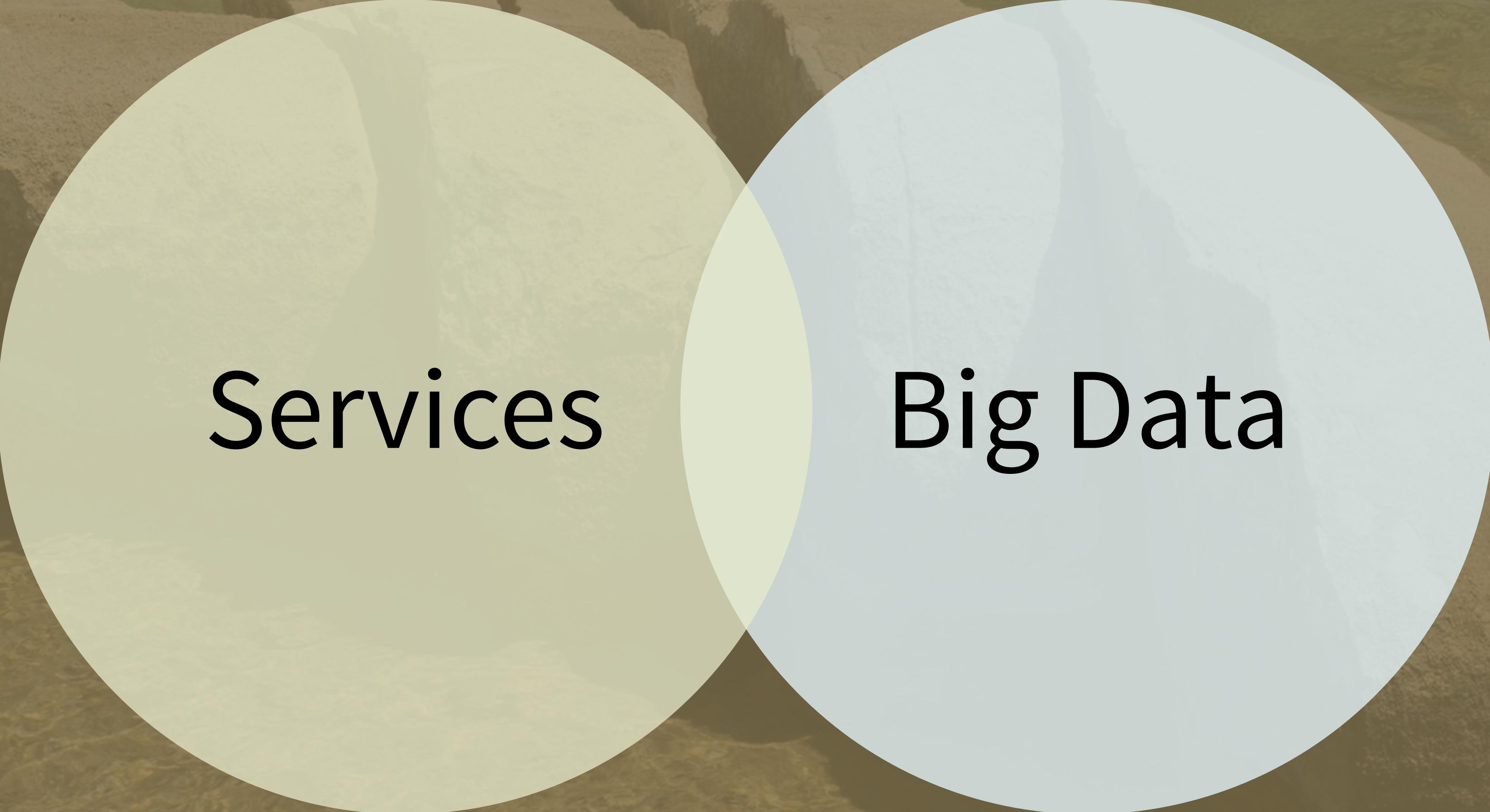
- A data app/microservice:
- A single responsibility.
- The input never ends.
- So, both must be available, responsive, resilient, & scalable. I.e., reactive



- Going the other way, “small” microservice architectures become data-centric, as the data grows.



The Recent Past



Services

Big Data

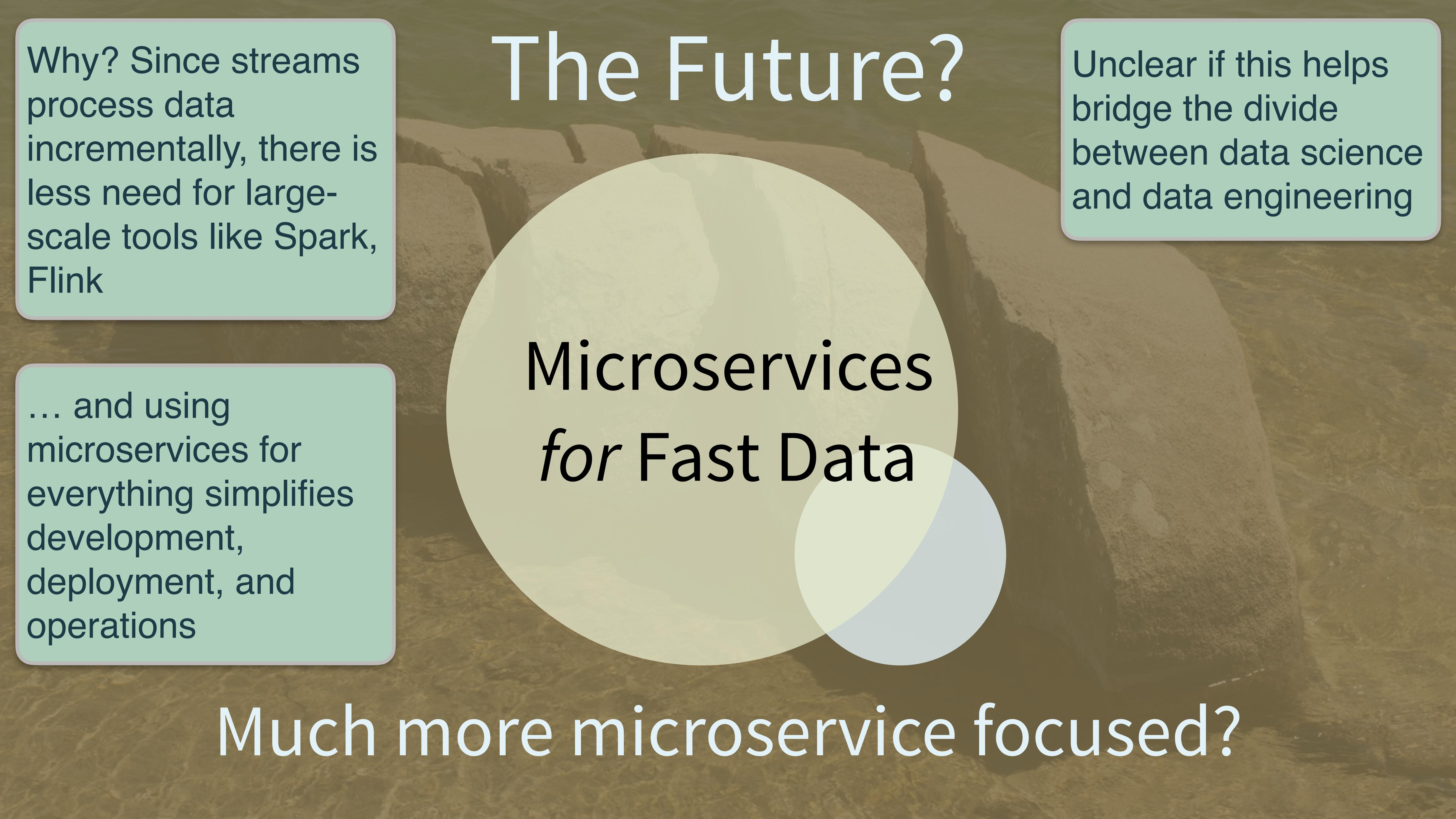
Some Overlap: Concerns, Architecture

The Present



Microservices
& *Fast* Data

Much More Overlap



Why? Since streams process data incrementally, there is less need for large-scale tools like Spark, Flink

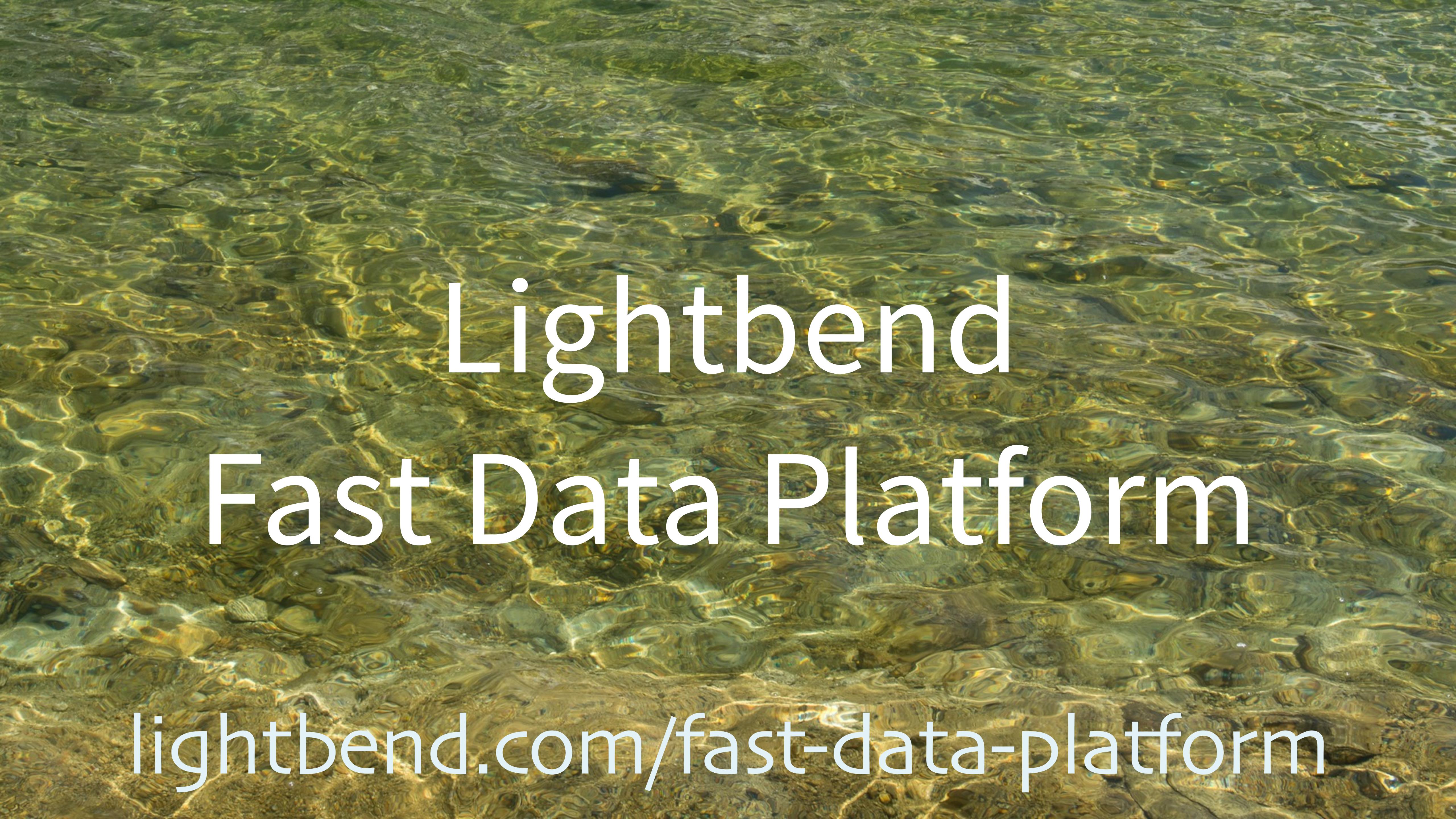
... and using microservices for everything simplifies development, deployment, and operations

The Future?

Unclear if this helps bridge the divide between data science and data engineering

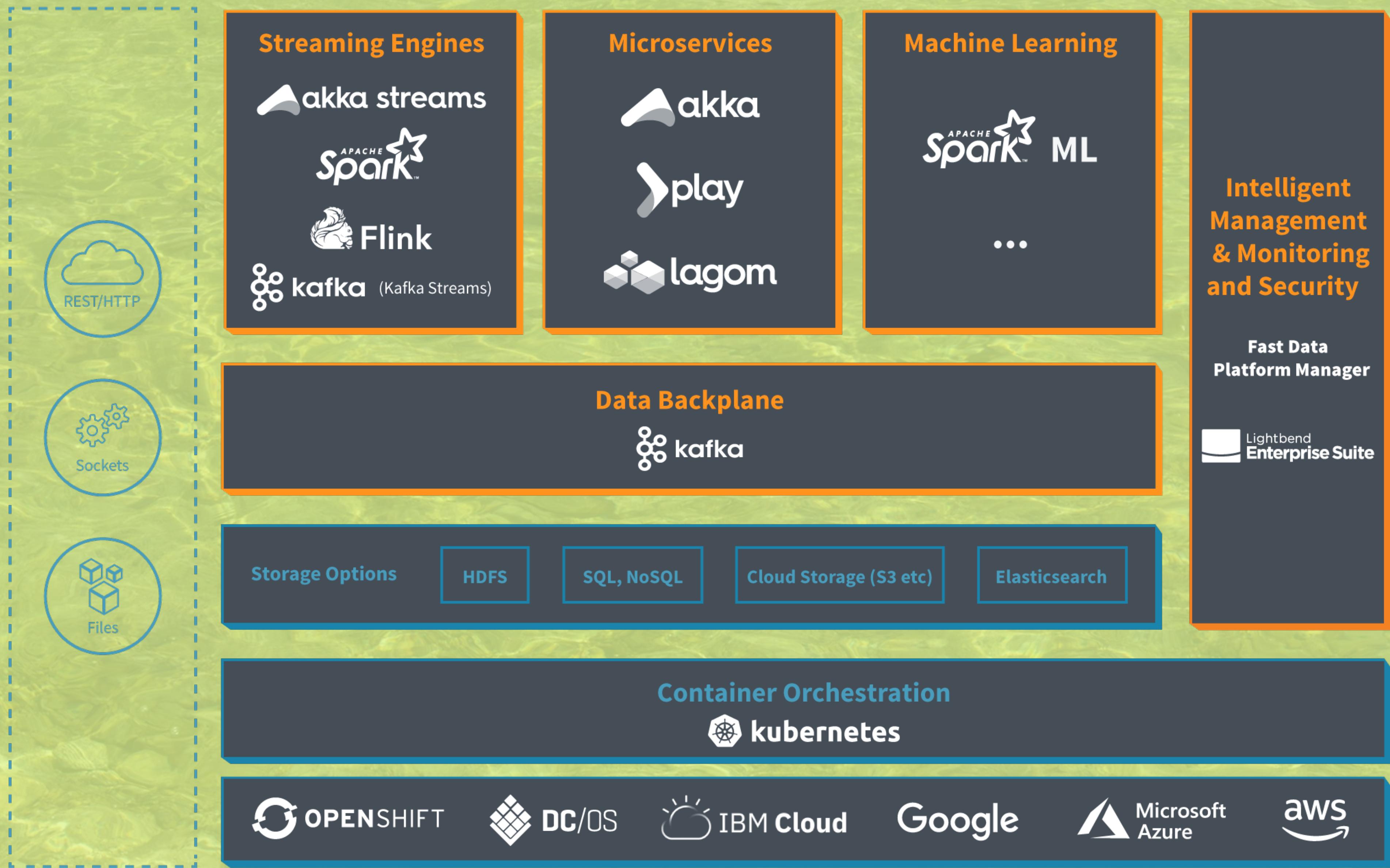
Microservices
for Fast Data

Much more microservice focused?



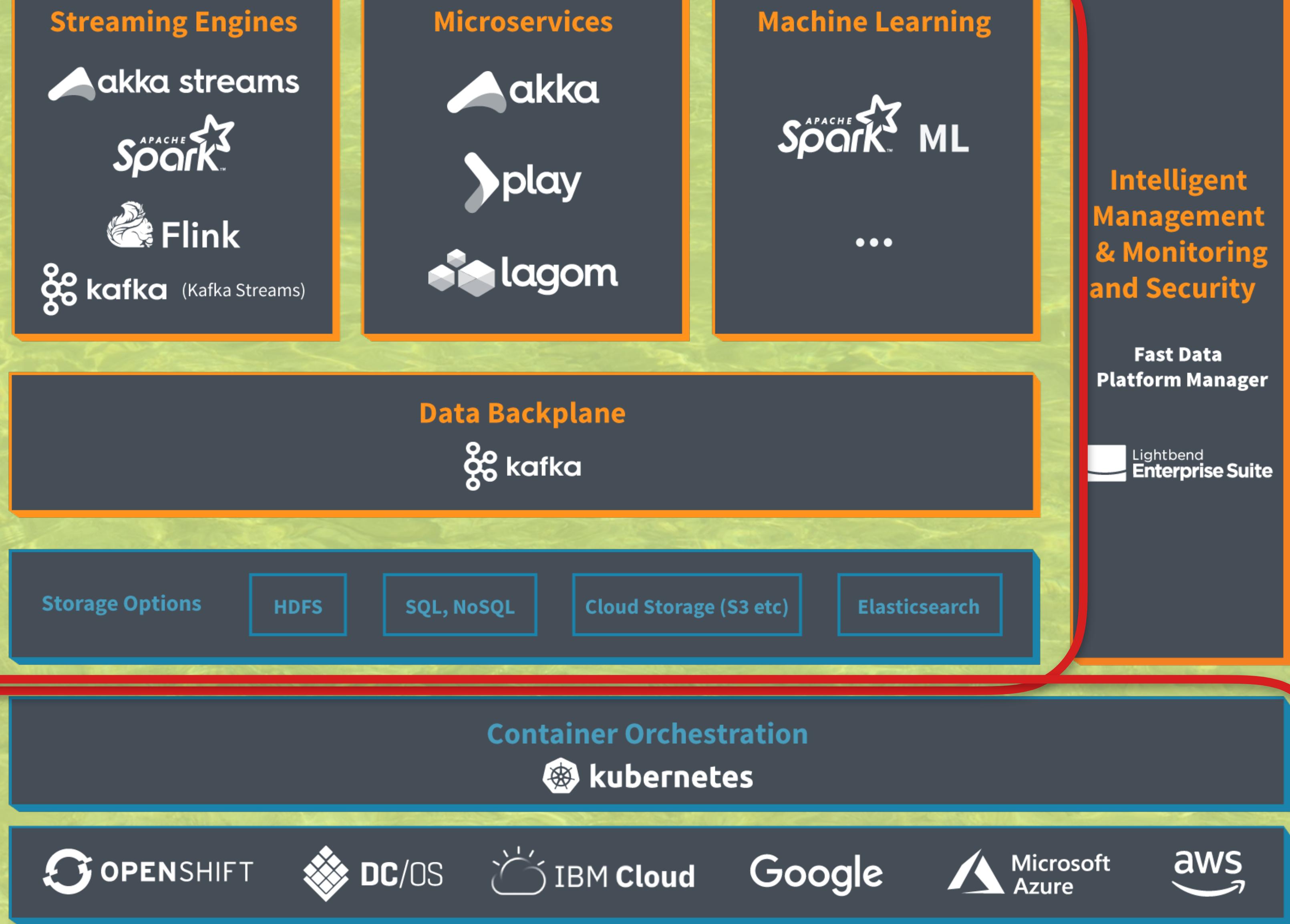
Lightbend Fast Data Platform

lightbend.com/fast-data-platform

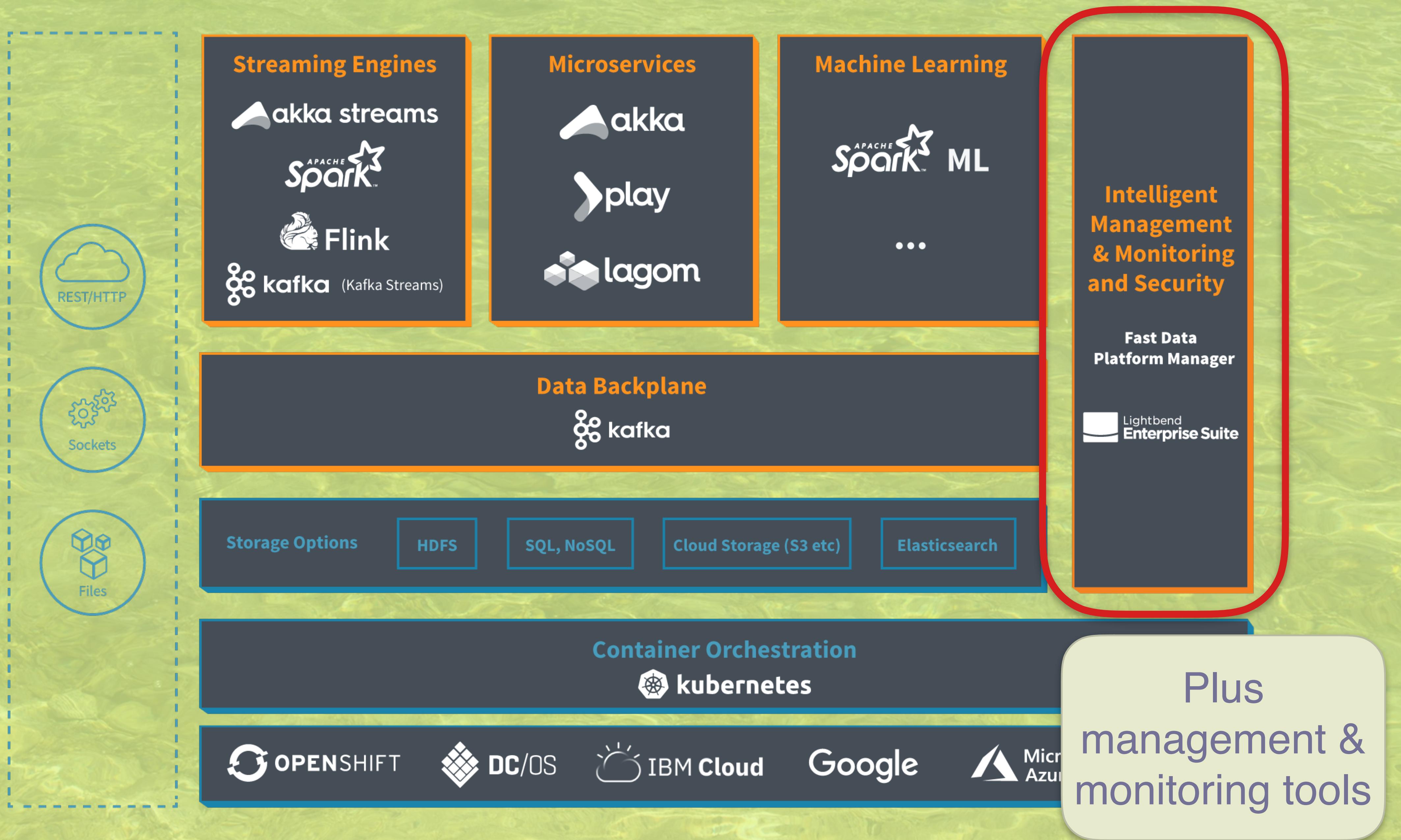


lightbend.com/fast-data-platform

What we
discusse



lightbend.com/fast-data-platform



lightbend.com/fast-data-platform

Questions?

Dean Wampler, Ph.D.

dean@deanwampler.com

@deanwampler

polyglotprogramming.com/talks

