

Stream All the Things!

Architectures for
Data Sets that
Never End



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



Photographs © Dean Wampler, 2007-2017. All rights reserved. All other content © Lightbend, 2014-2017. All rights reserved.

You can download this and my other talks from the polyglotprogramming.com/talks link.

Photograph: Tower Bridge, time-lapse photo at night.



Free as in

bit.ly/fastdata-ORbook

O'REILLY®

Fast Data Architectures for Streaming Applications

Getting Answers Now from
Data Sets that Never End



Dean Wampler

Compliments of
 Lightbend

Streaming in Context...



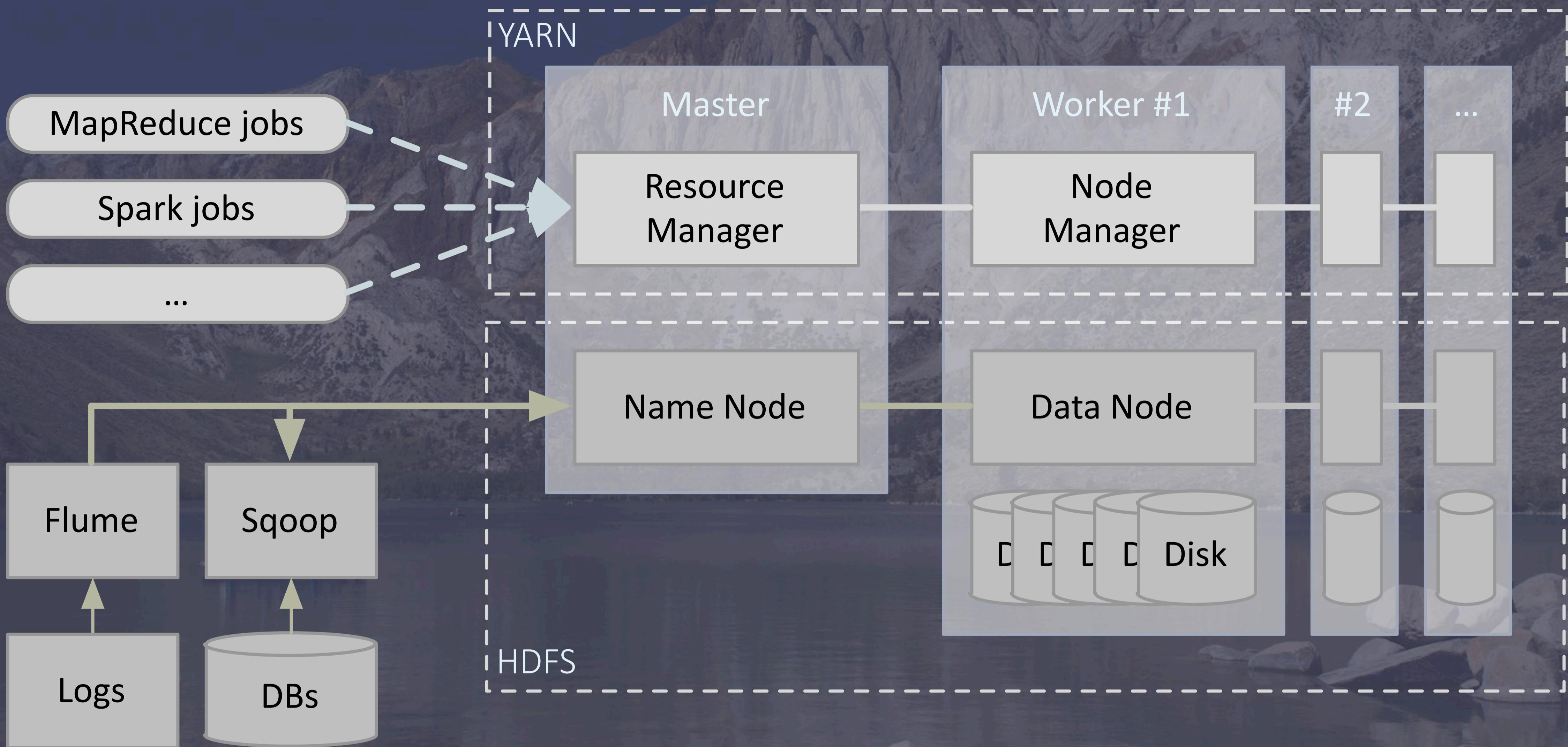
Photographs © Dean Wampler, 2007-2017. All rights reserved. All other content © Lightbend, 2014-2017. All rights reserved.

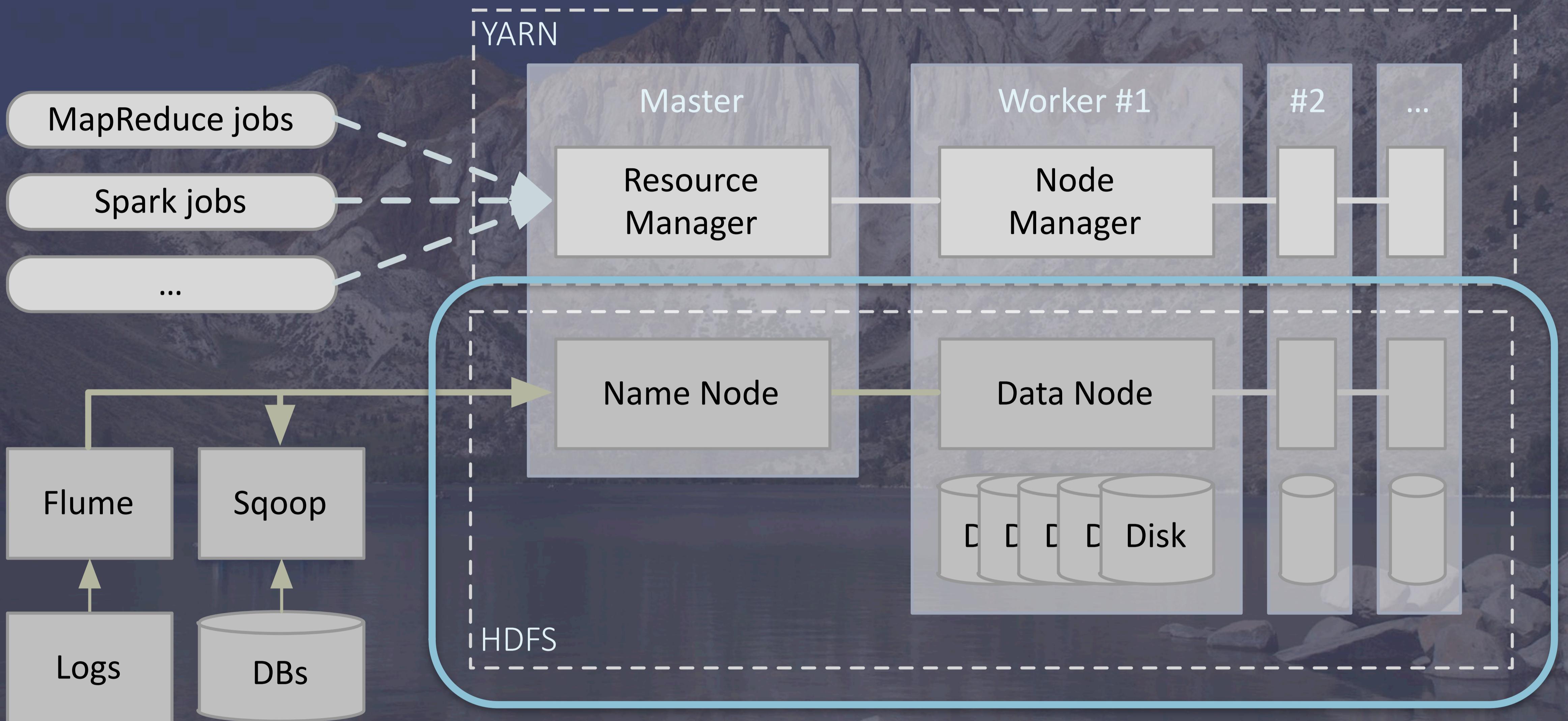
You can download this and my other talks from the polyglotprogramming.com/talks link.

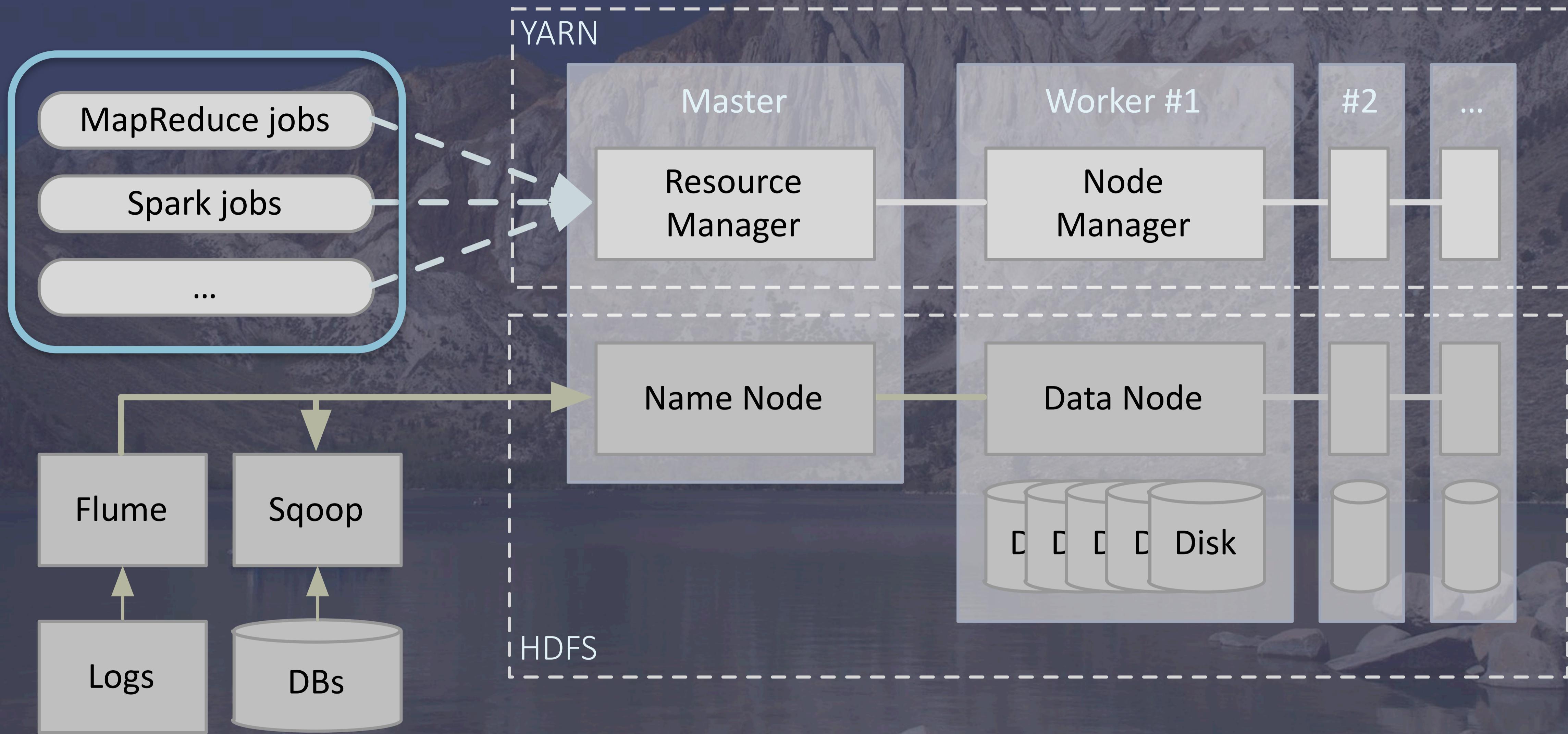
Photograph: Farms in the Sacramento River delta, ENE of San Francisco .

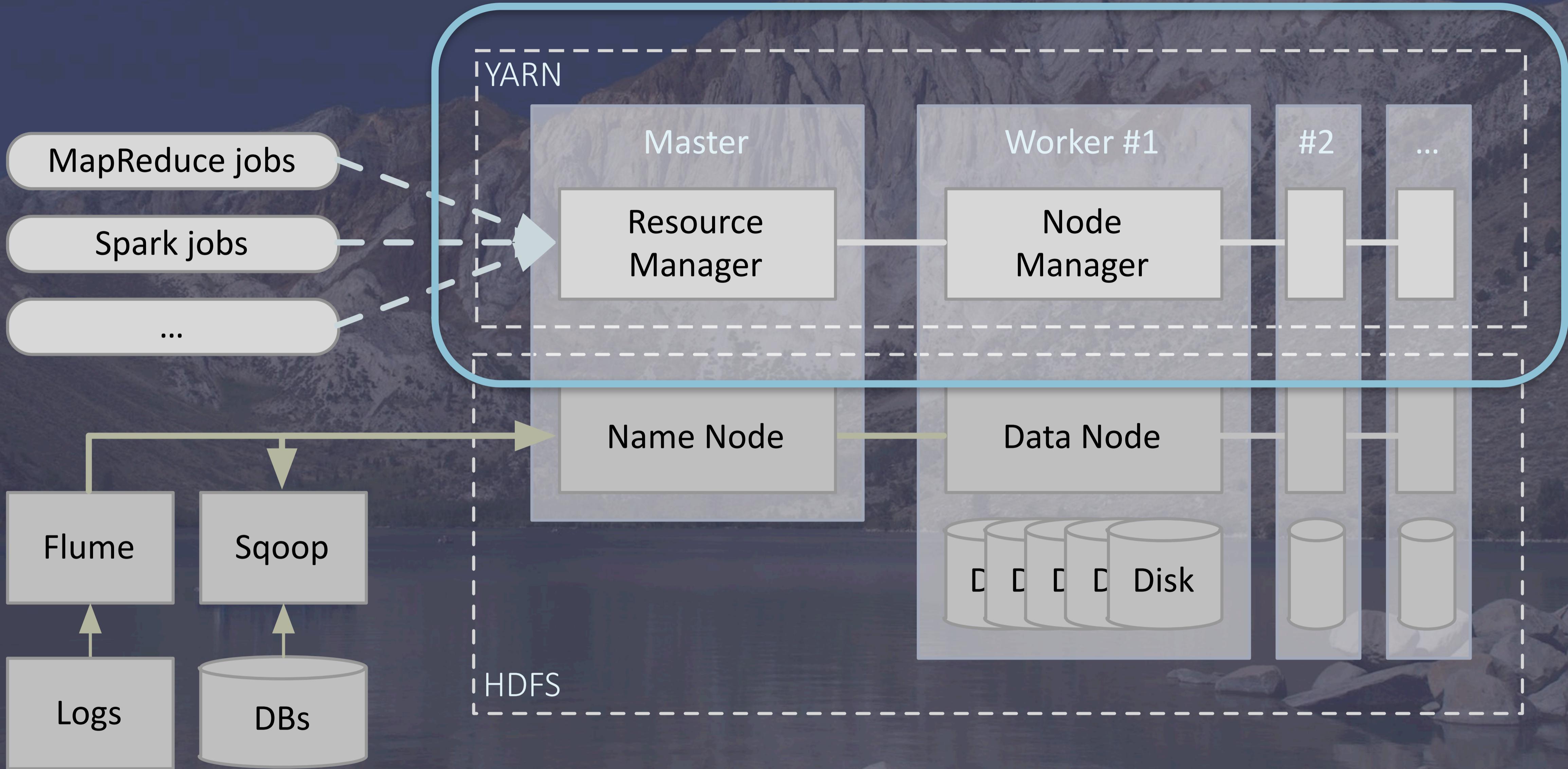


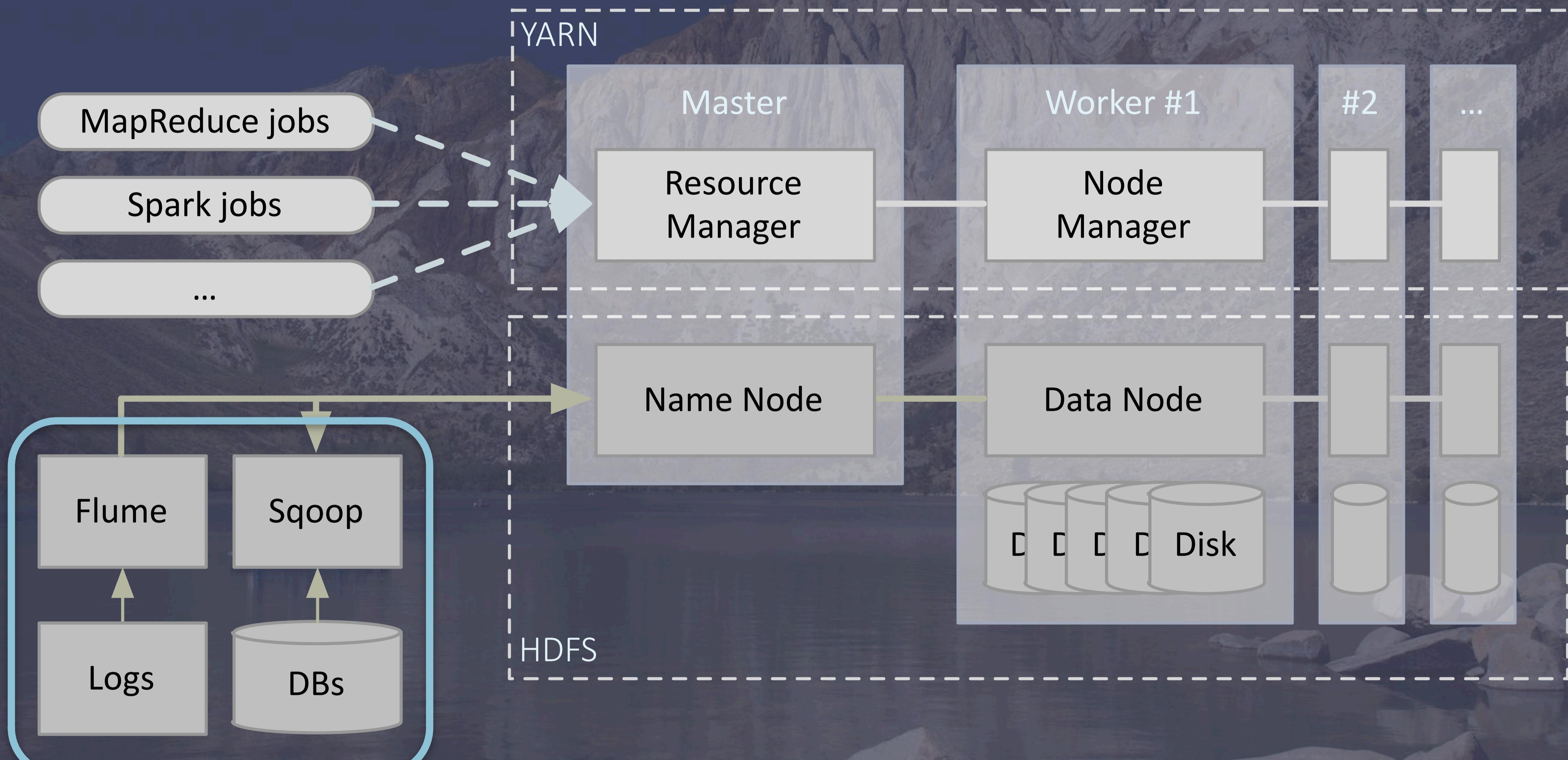
Hadoop: Classic Batch Architecture



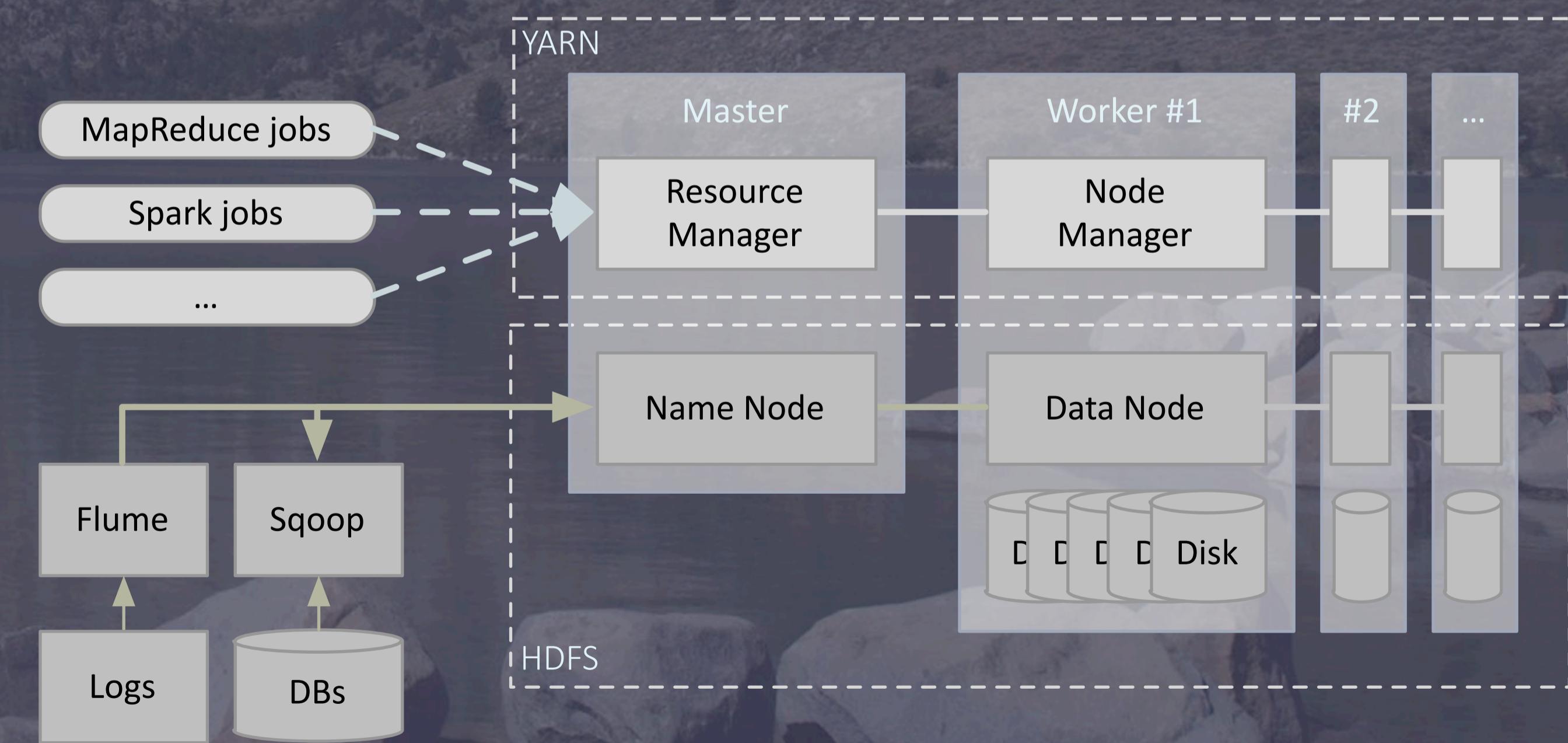








- Characteristics
 - Batch oriented
 - Massive storage
 - Multiuser jobs
- Data warehouse replacement

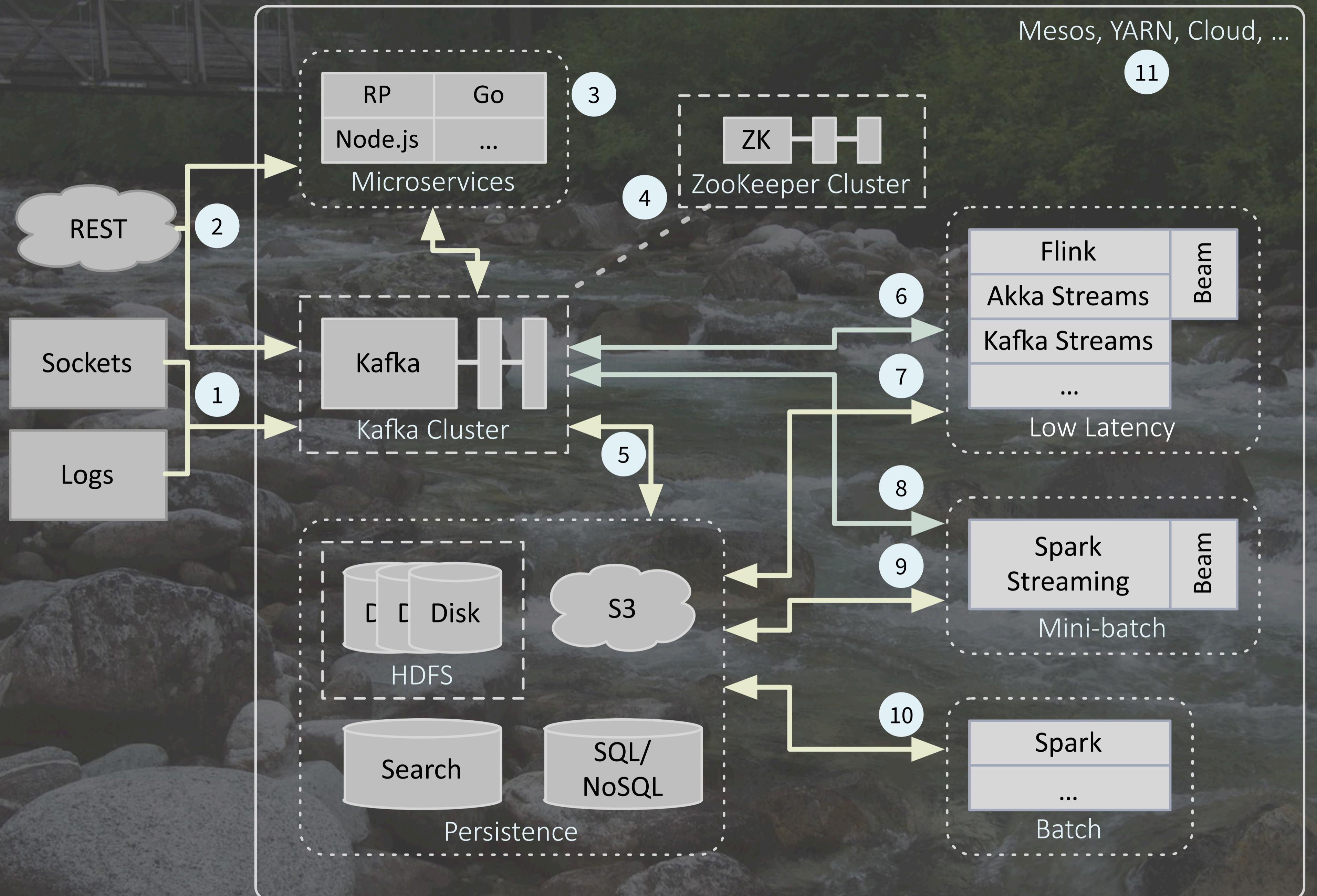


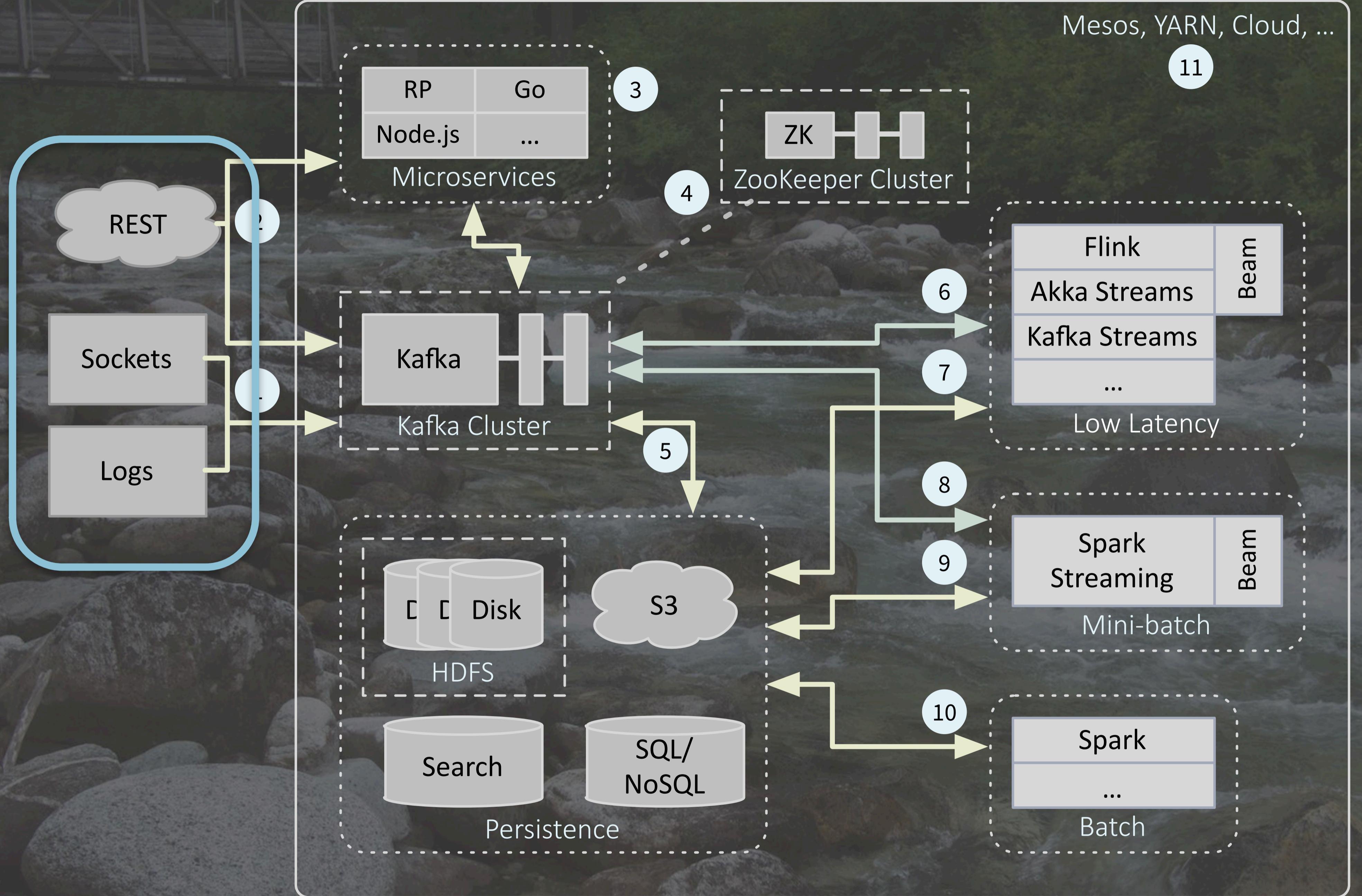
A photograph of a rocky stream flowing through a dense green forest. Large, smooth rocks are scattered along the bank and in the water. In the background, a rustic wooden bridge spans the stream. The water is clear and turbulent, reflecting the surrounding greenery.

New Streaming, “Fast Data” Architecture

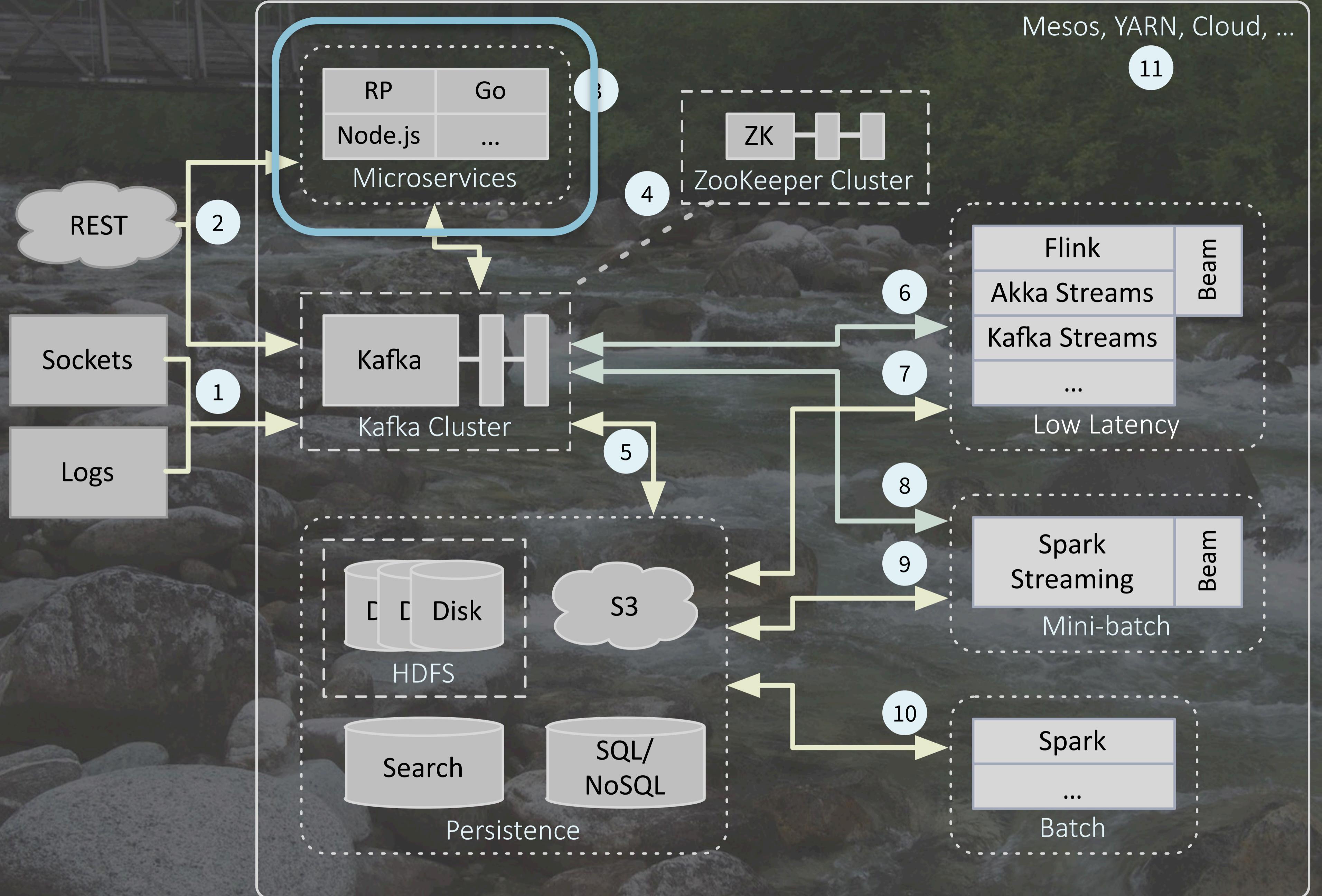
But it will also support batch!!

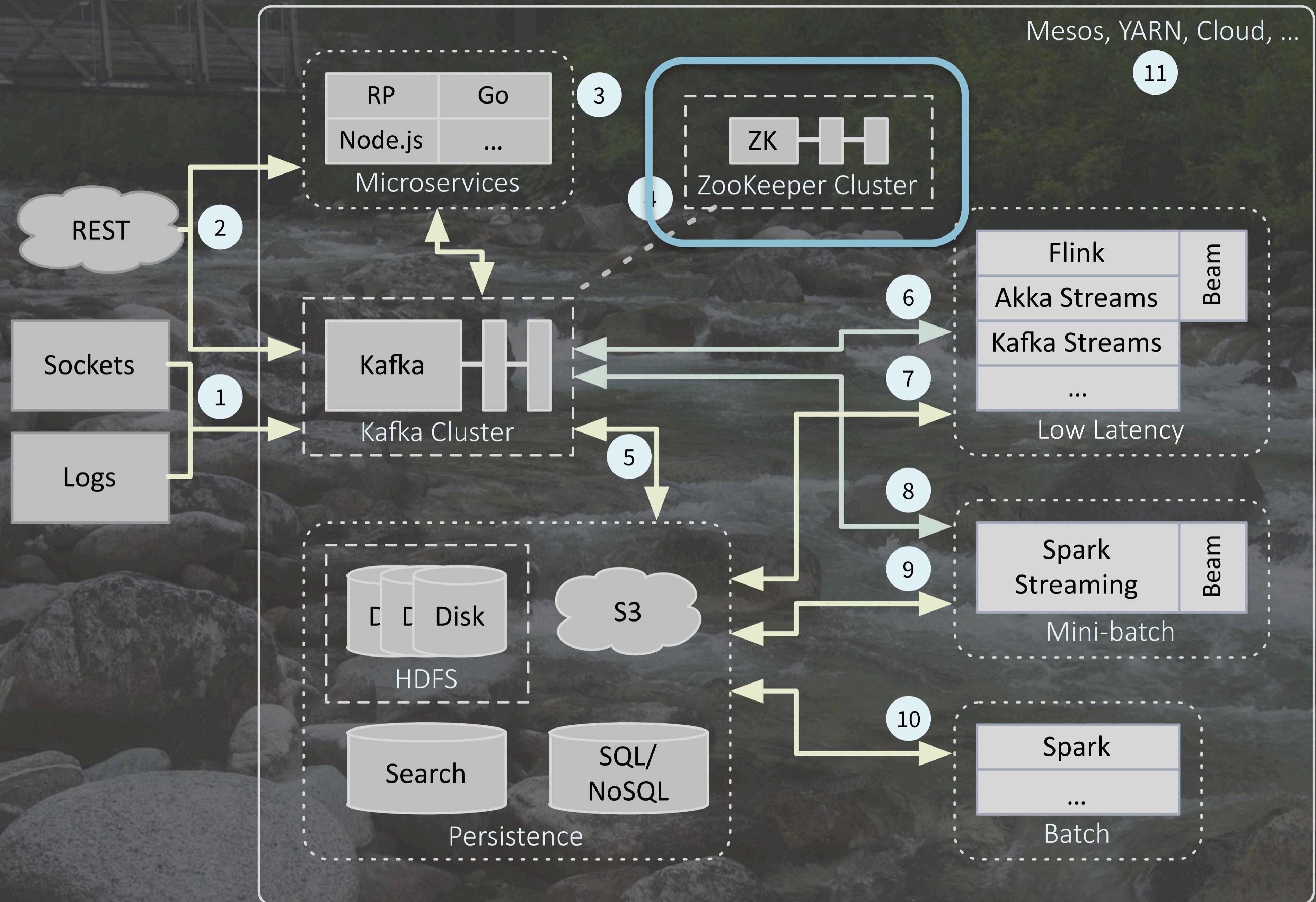
Photo: Stream in North Cascades National Park, Washington State.

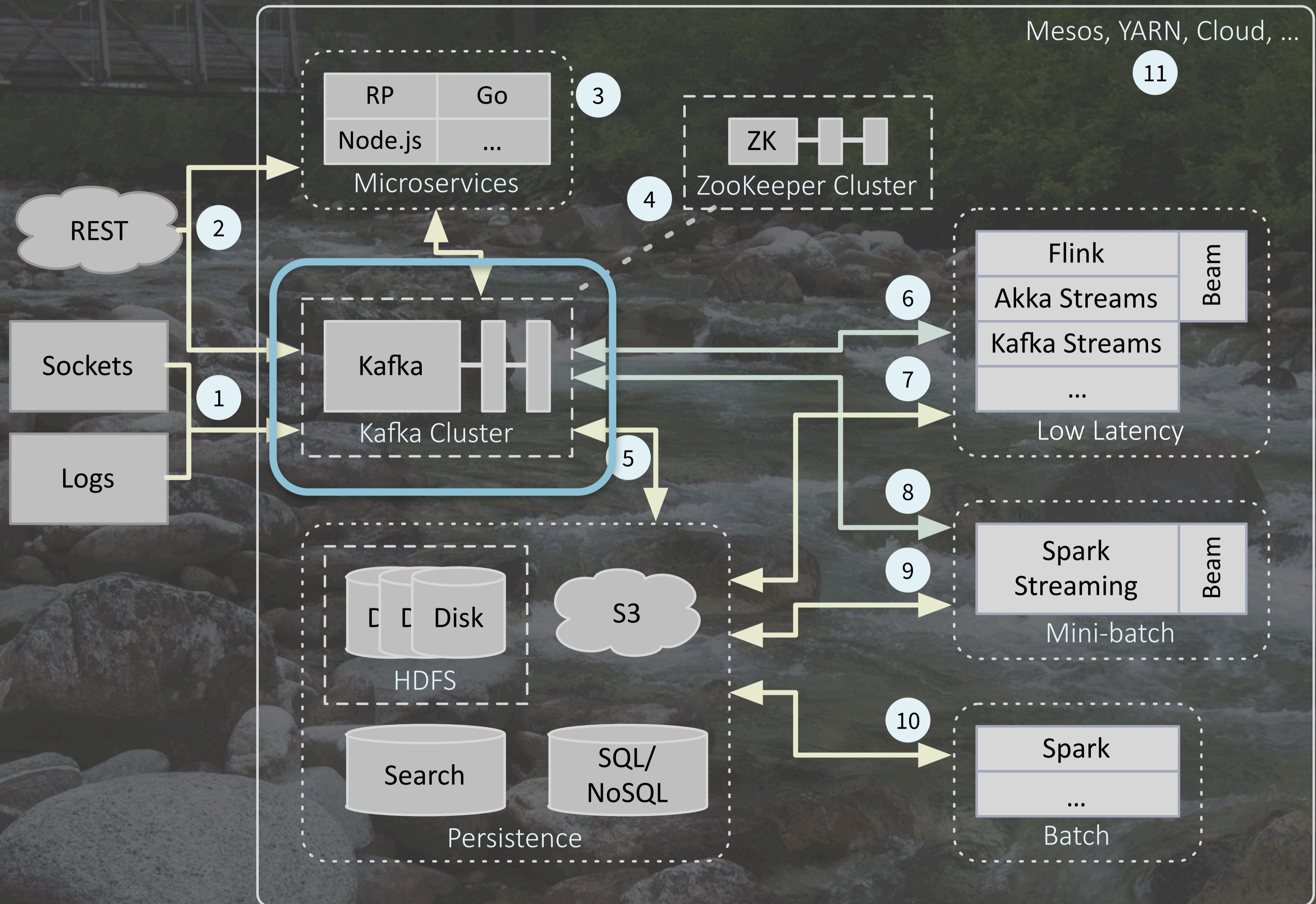




1 & 2. Data sources include streams of data over sockets and from logs. You might also ingest through REST channels, but unless they are async, the overhead will be too high, so REST might be used only for communicating with the microservices (3) you write to complete your environment.



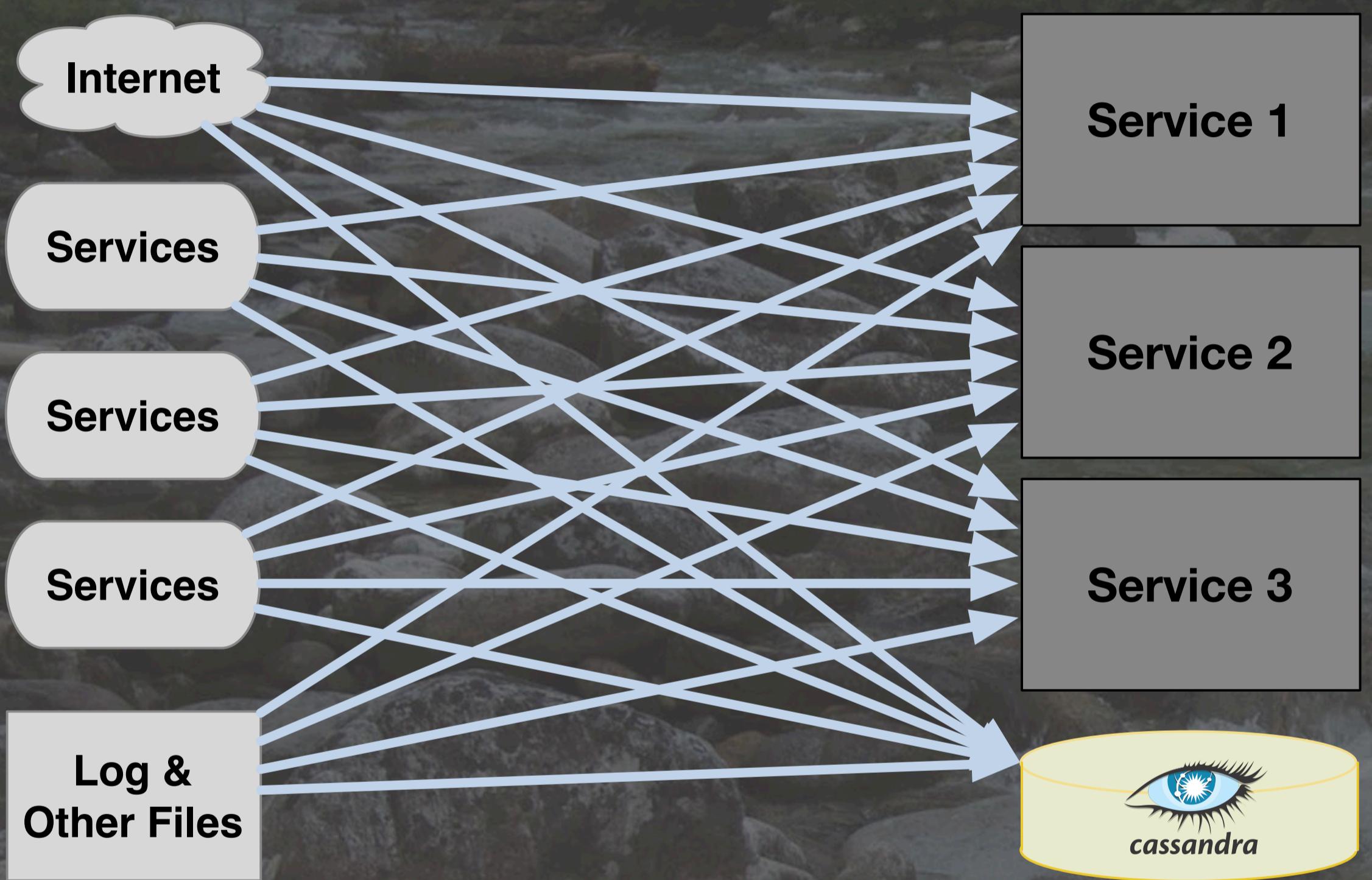




5. Kafka is the core of this architecture, the place where data is ingested in queues, organized into topics. Publishers are consumers are decoupled and N-to-M. Kafka has massive scalability and excellent resiliency and data durability. All services can communicate through each other using Kafka, too, rather than having to manage arbitrary point-to-point connections.

Using Kafka

Before:

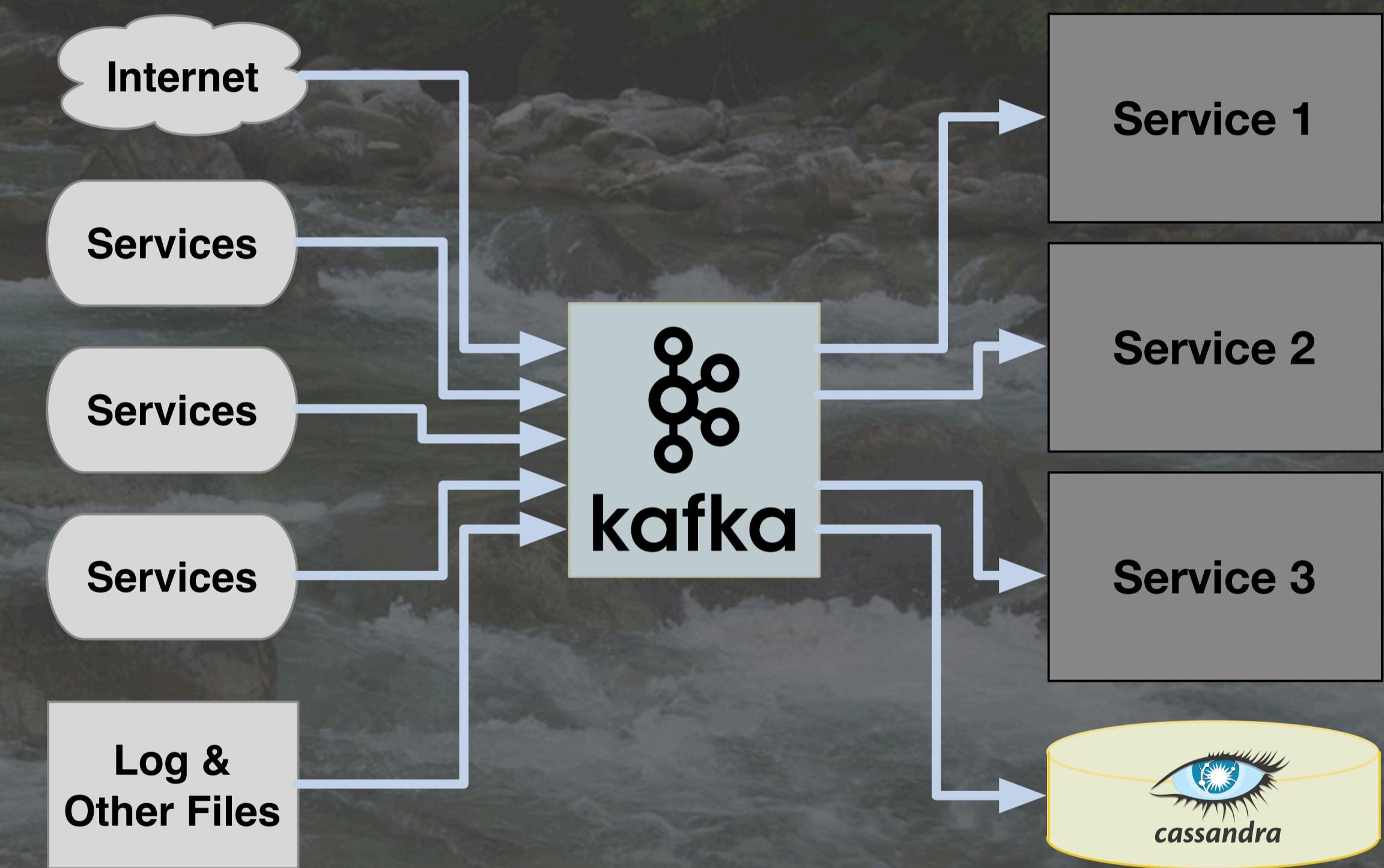


Producers

$N * M$ links

Consumers

After:

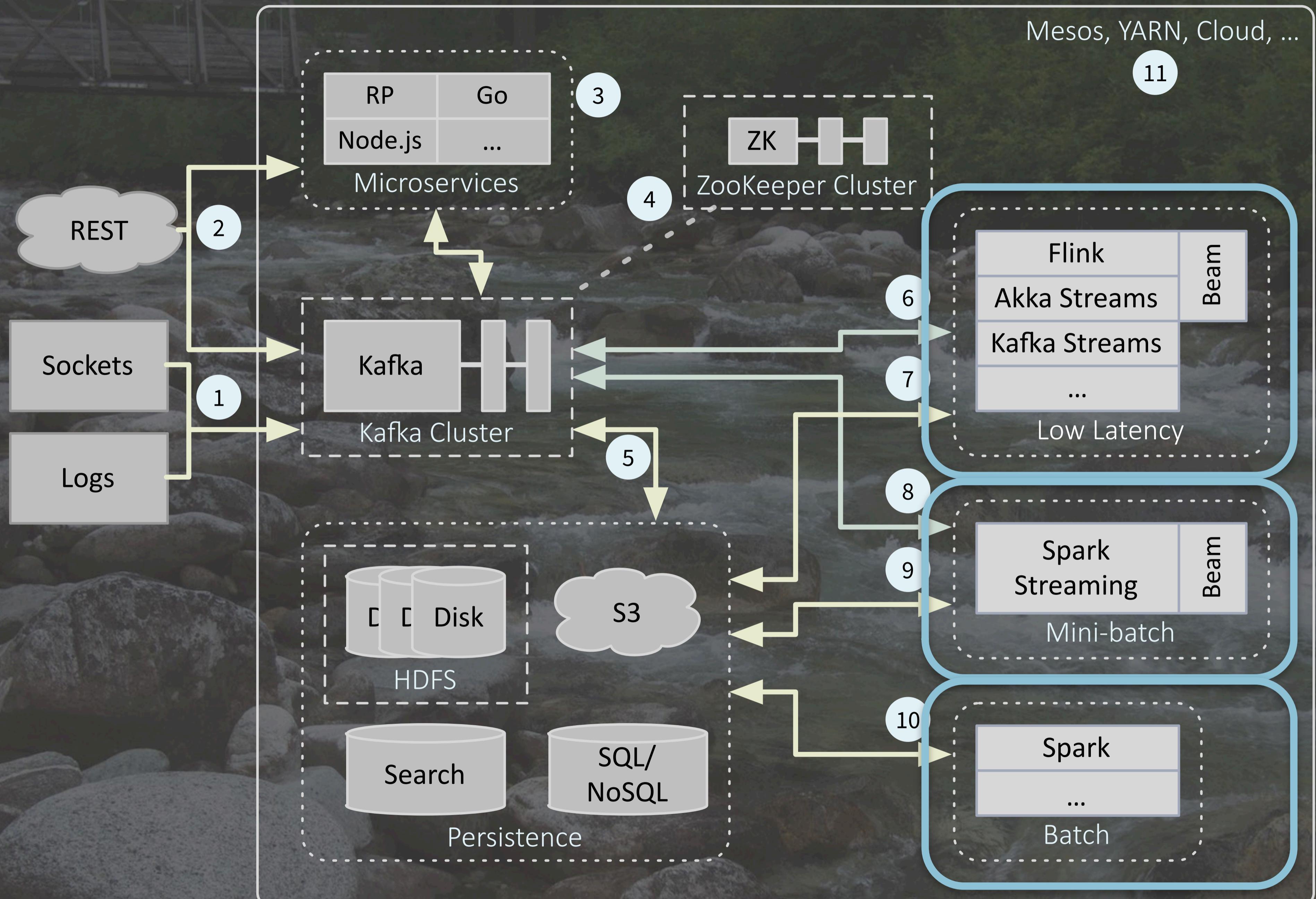


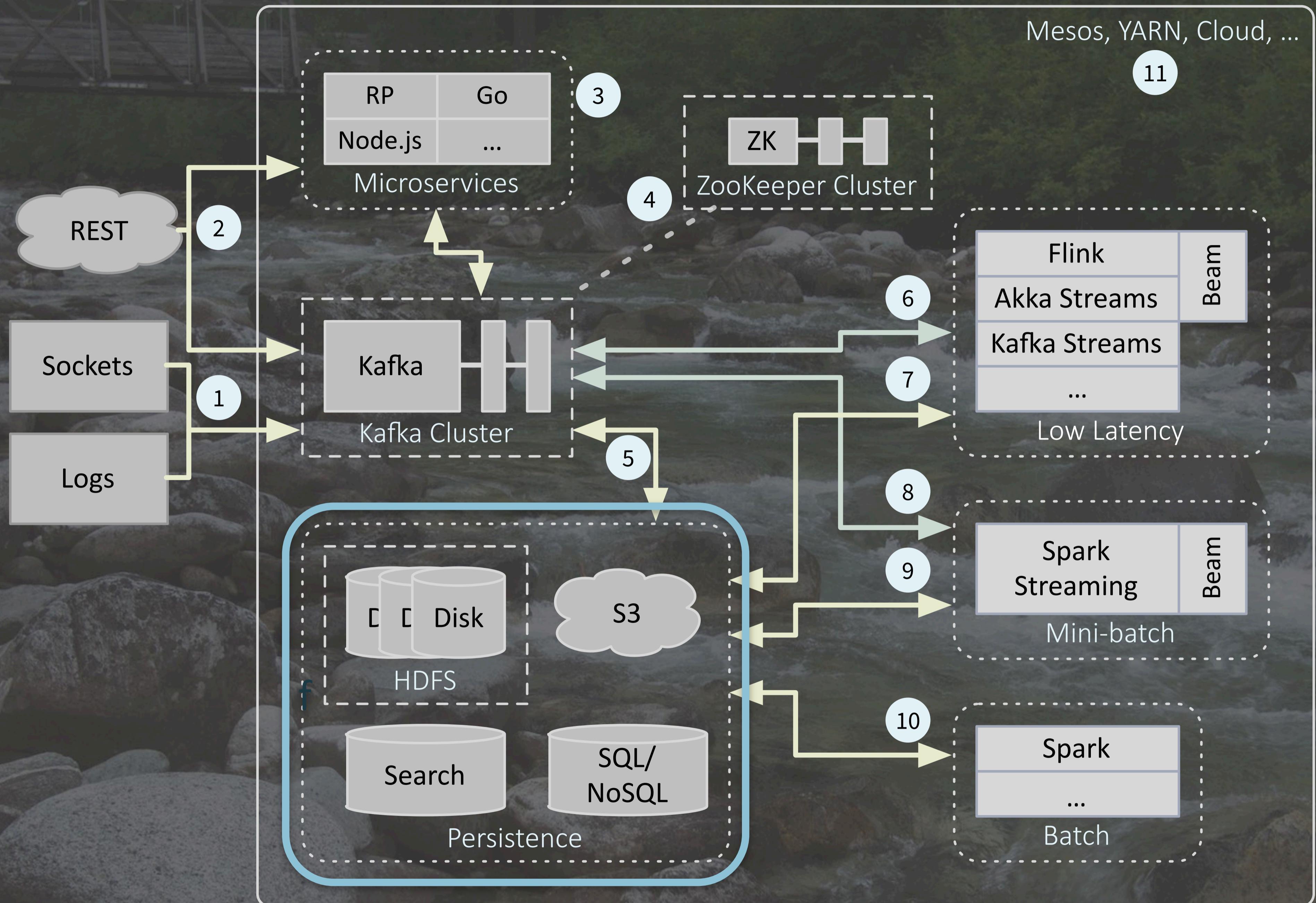
Producers

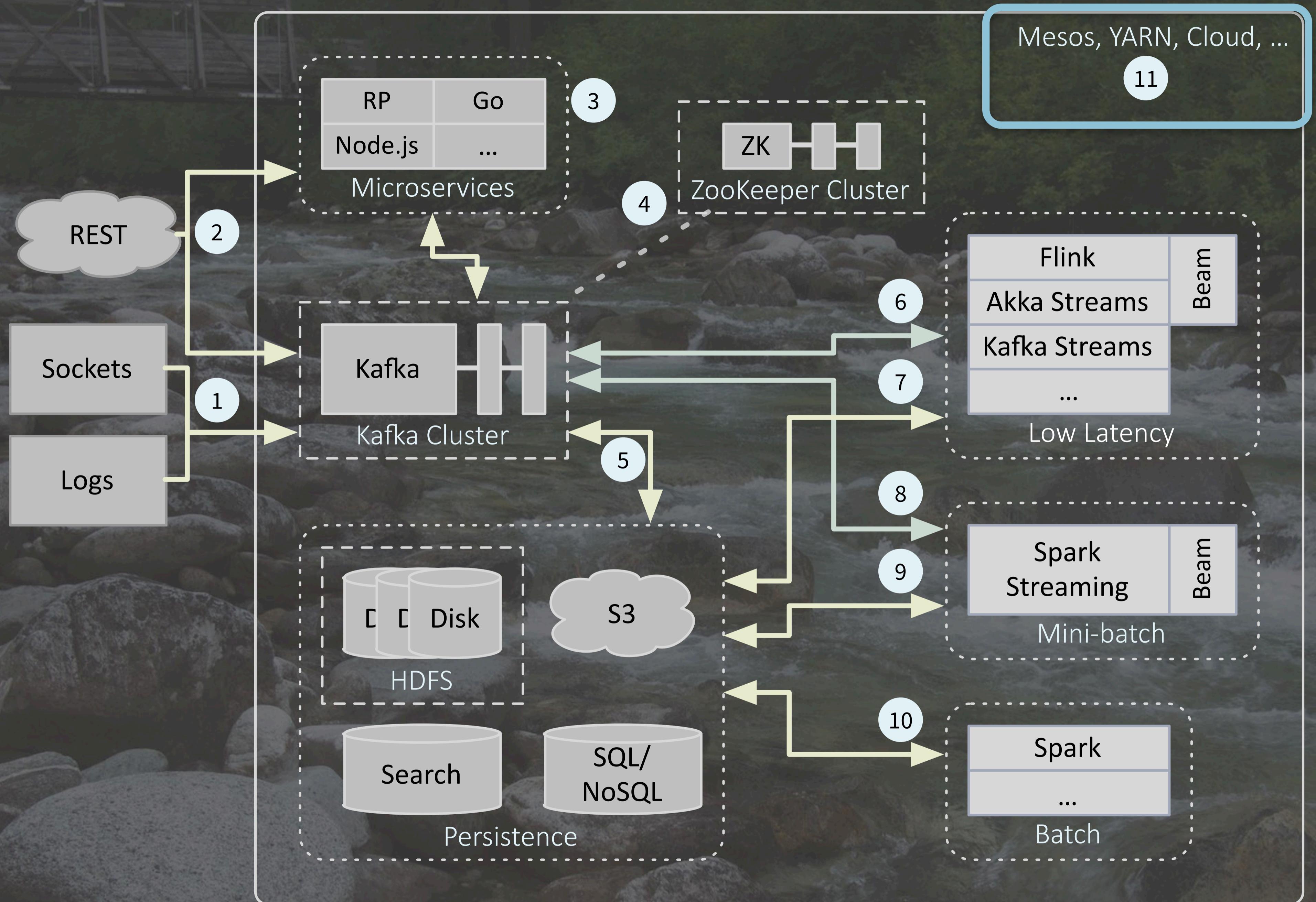
$N + M$ links

Consumers

Kafka can simplify your architecture, too. Not only is this cleaner, it's more robust, as the point-to-point connections are more fragile, more likely to result in data and service loss if one point goes down, where Kafka keeps the other point "healthy" while the failed point is restored. Kafka also lets you easily support multiple consumers or producers per topic (the way data is organized, as in classic message queues). Finally, the uniformity of always (or most of the time) communicating through Kafka simplifies the challenge of connecting services together with different APIs.





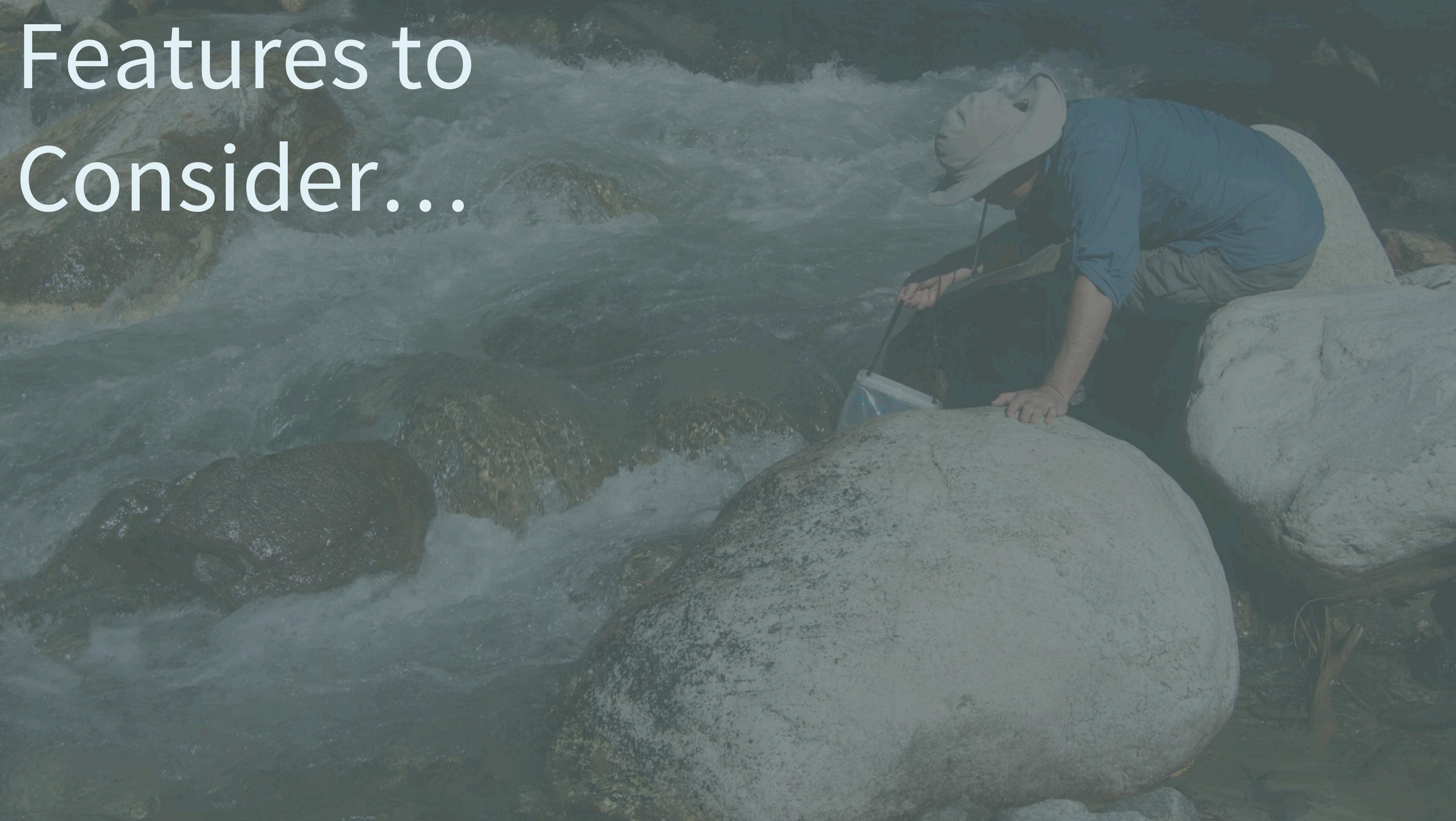




Streaming Engines

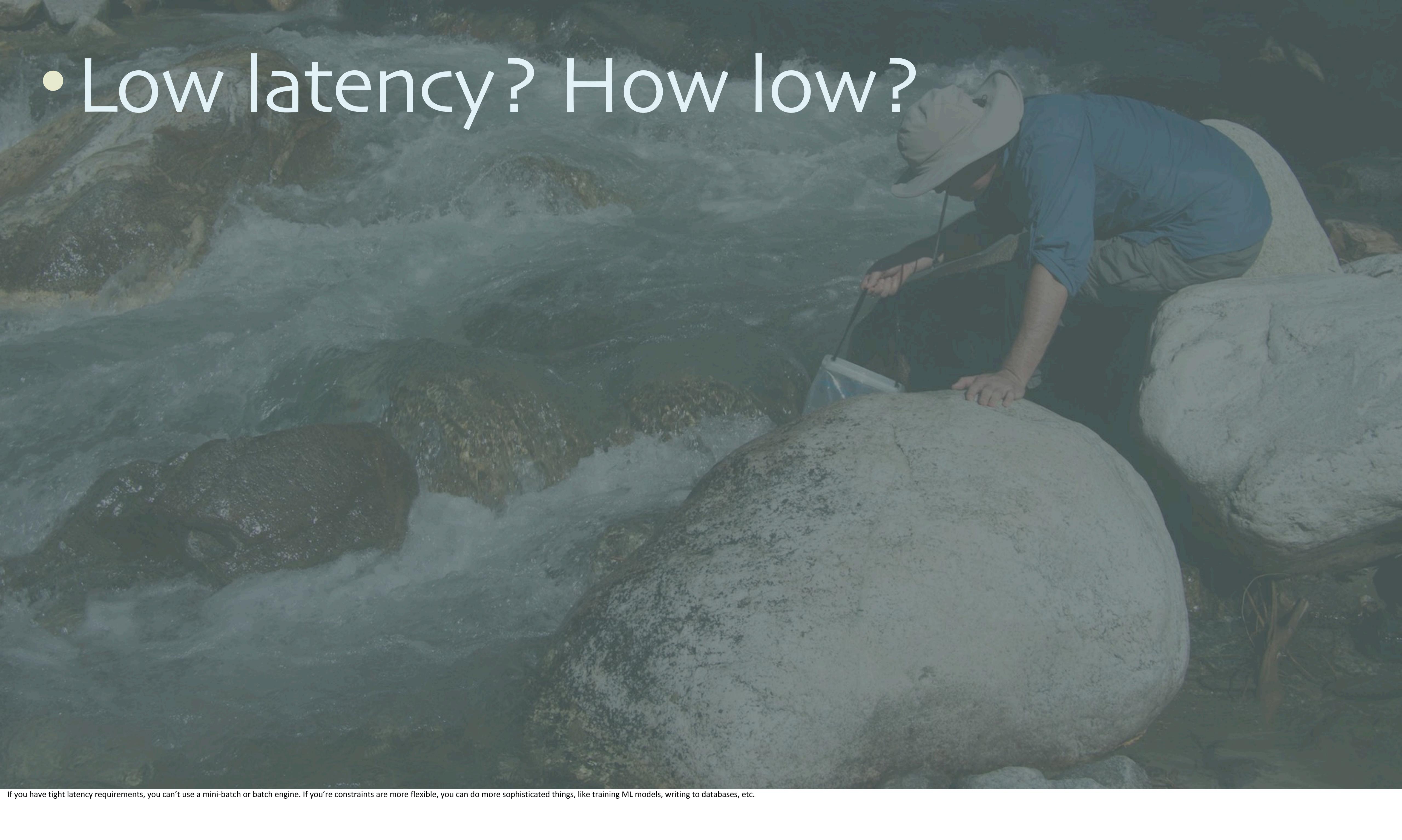
Let's dive into a representative set of available streaming engines.

Photo: Fetching water from a stream in North Cascades National Park, Washington State.

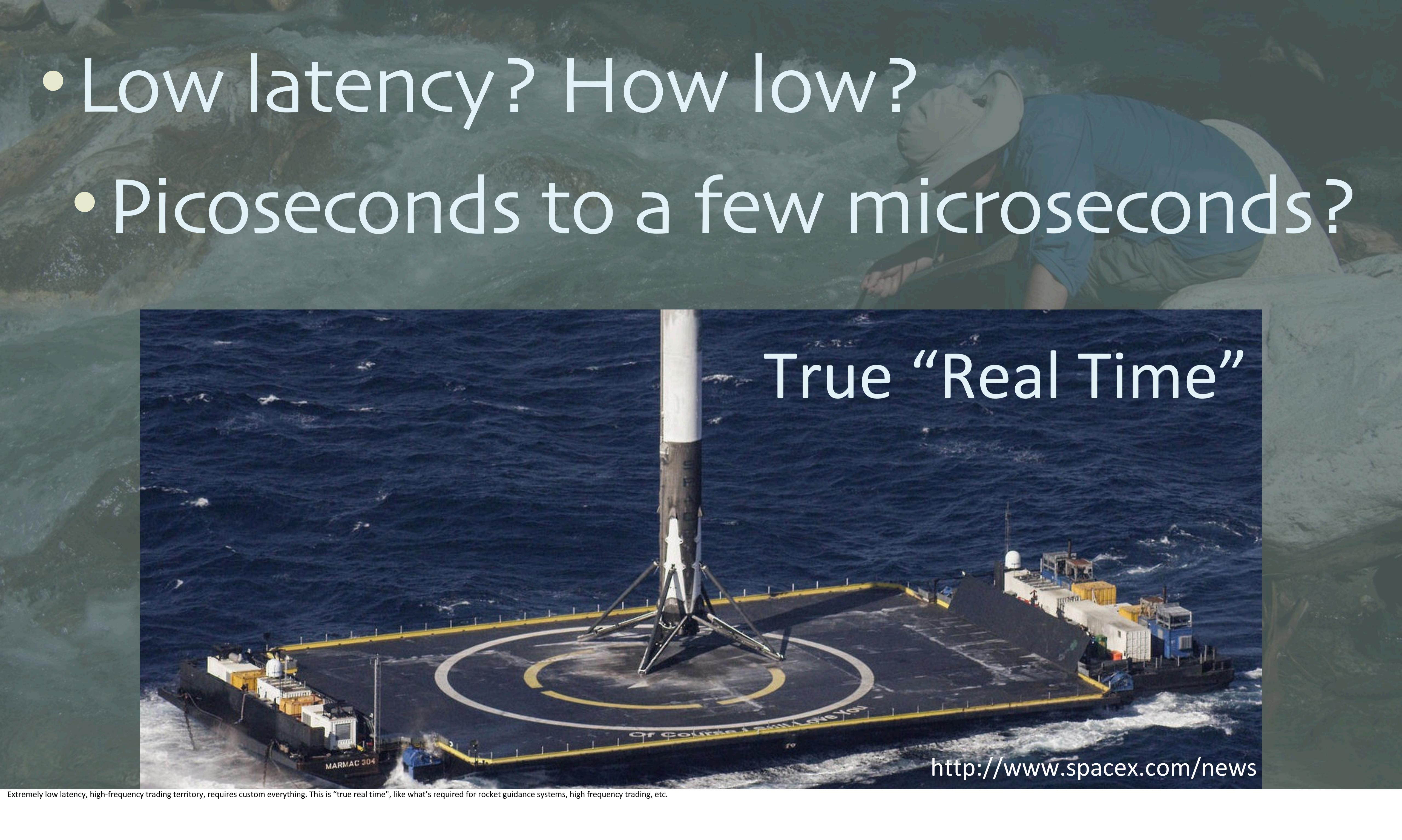


Features to Consider...

- Low latency? How low?



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



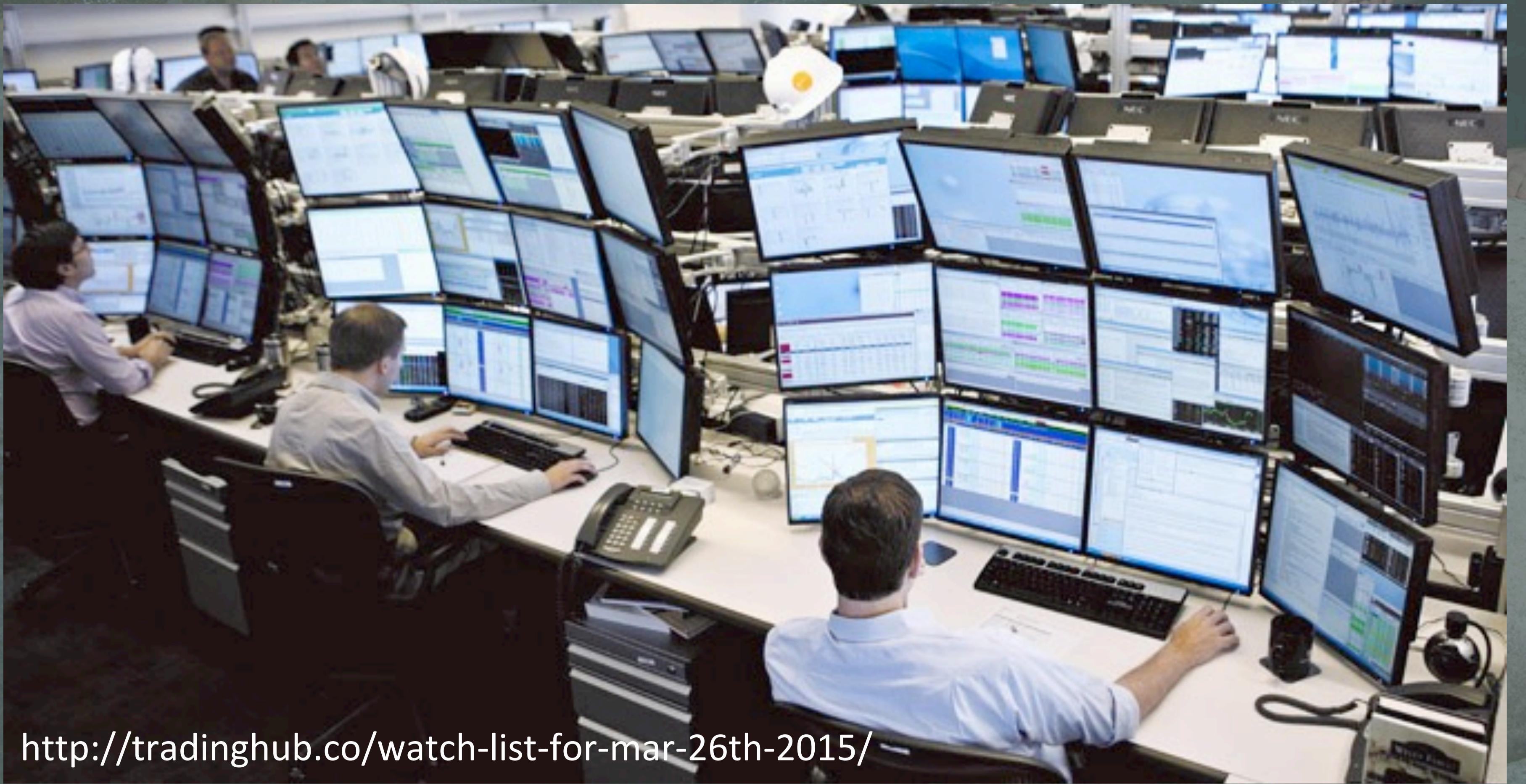
True “Real Time”



<http://www.spacex.com/news>

- 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?



<http://tradinghub.co/watch-list-for-mar-26th-2015/>

With more latency tolerance. This is too long for high frequency trading, but fast enough for a lot of trading systems. Also, this is the realm of many medical devices, like cardiac monitors.



<http://www.usa.philips.com/>

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

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

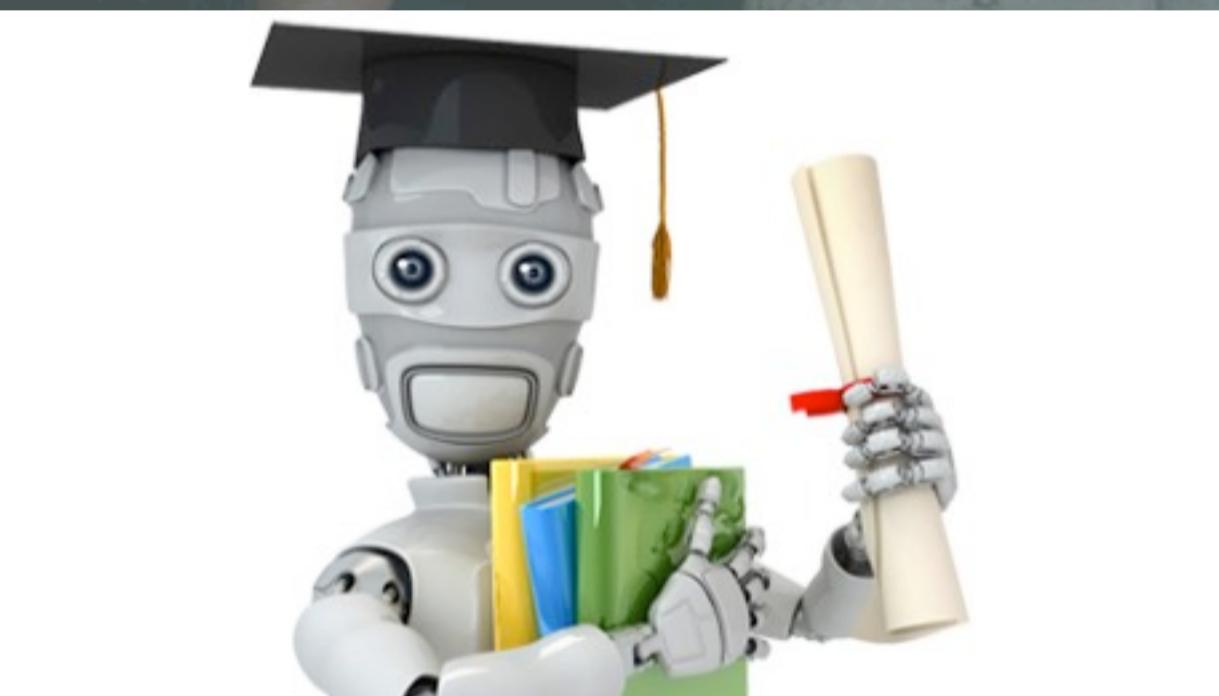


<http://money.cnn.com/2017/05/12/pf/credit-card-mistakes/index.html>

Now we're in the low end of what high volume, fast data architectures are designed for. I was told recently that when you buy something on a typical ecommerce web site, the credit card processor gets only about 10 milliseconds in that round-trip transaction.

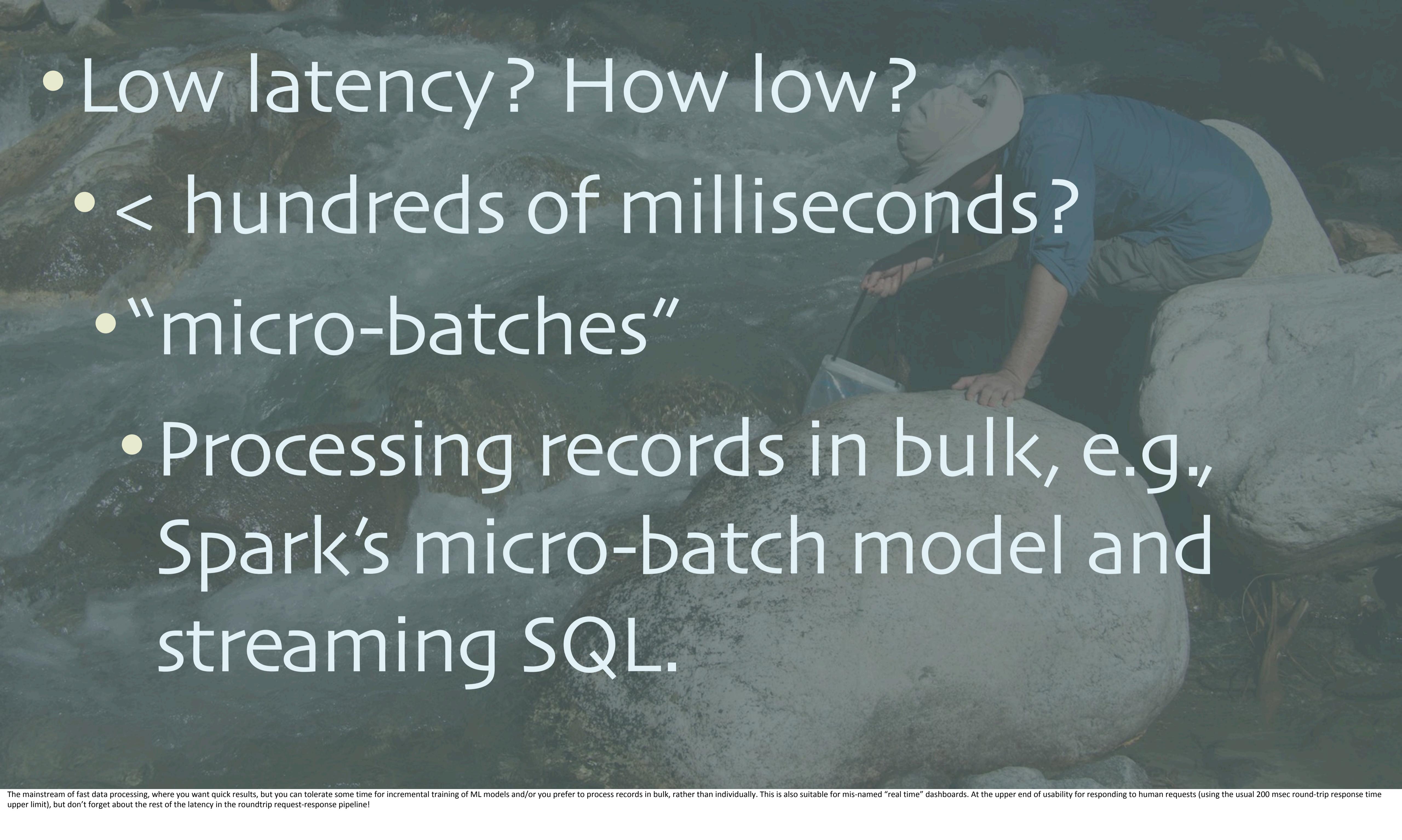
- Low latency? How low?
 - < 10 milliseconds?
 - Fast data streaming tools like Flink, Akka (and Akka Streams), Kafka Streams.

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

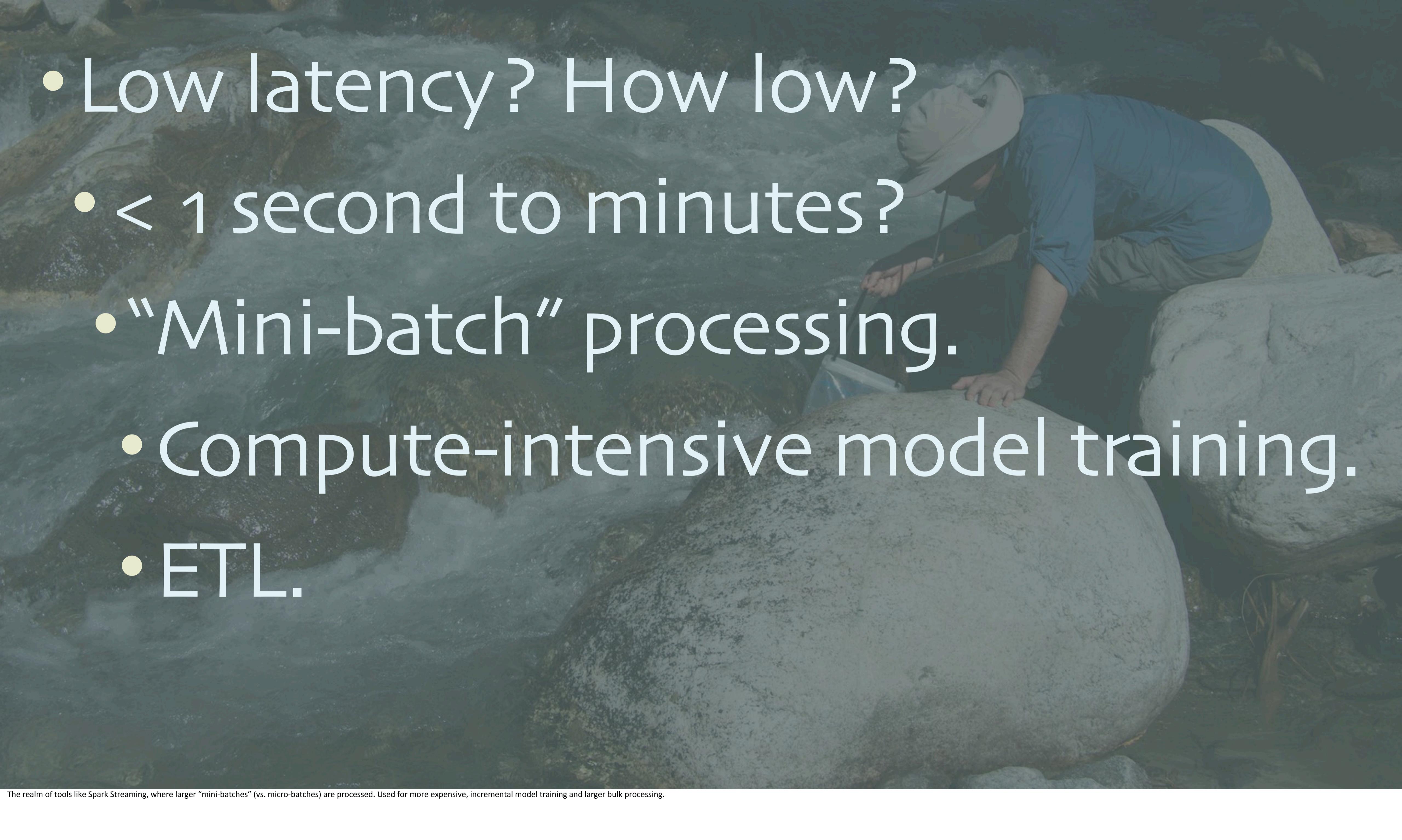


<https://www.coursera.org/learn/machine-learning>

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



- Low latency? How low?
 - < 1 second to minutes?
 - “Mini-batch” processing.
- Compute-intensive model training.
- ETL.



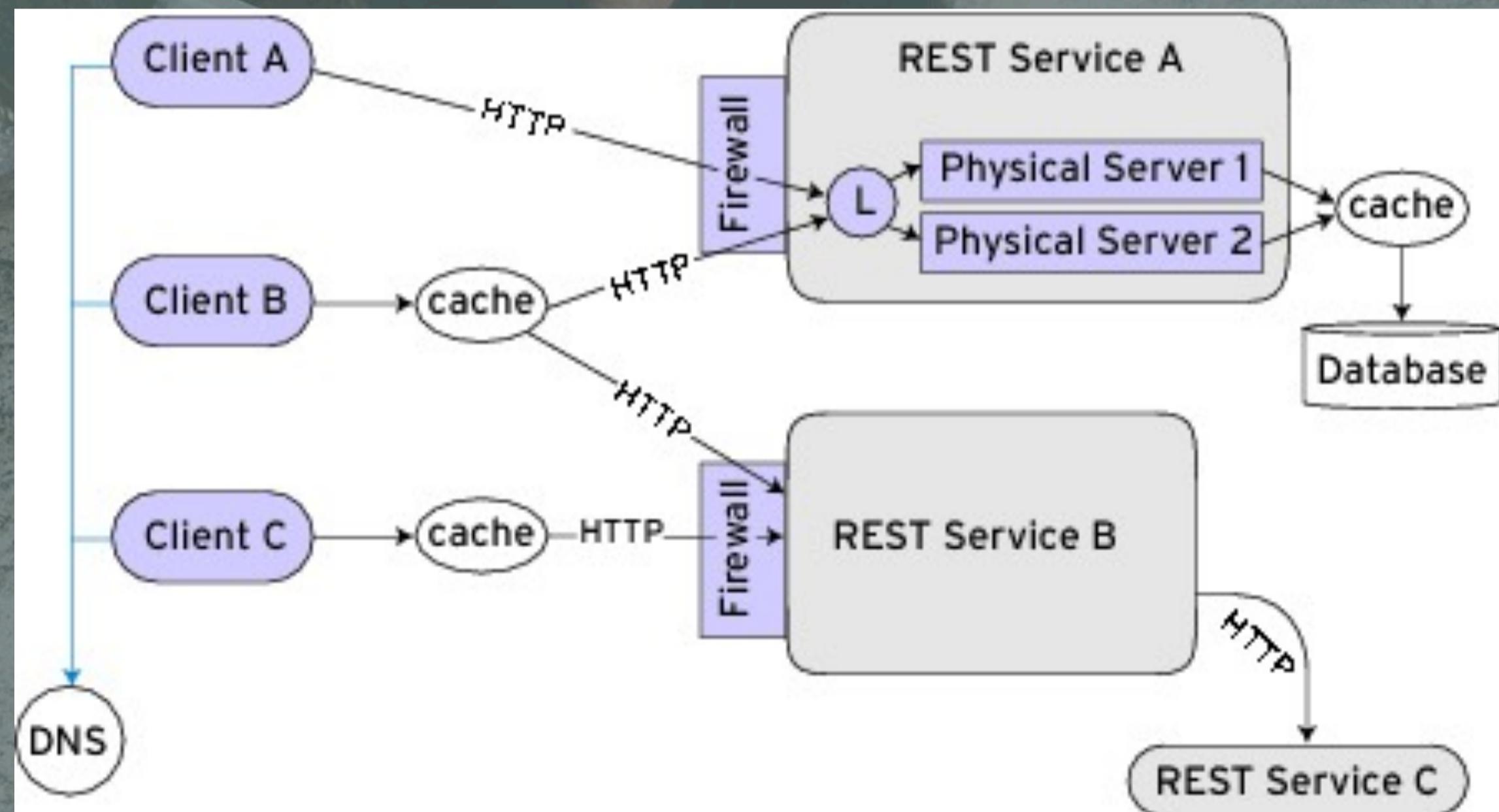
- Low latency? How low?
 - > a few minutes?
 - Consider periodic batch jobs!

- High Volume: How high?



Some tools are better at large volumes than others. If you have low volumes, you'll want tools that are efficient at low volumes, whereas some of the high-volume tools are efficient per record when amortized over the stream.

- 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 events/second?
- Nonblocking REST!
 - e.g., parallel Akka actors
- Still process individually (?)

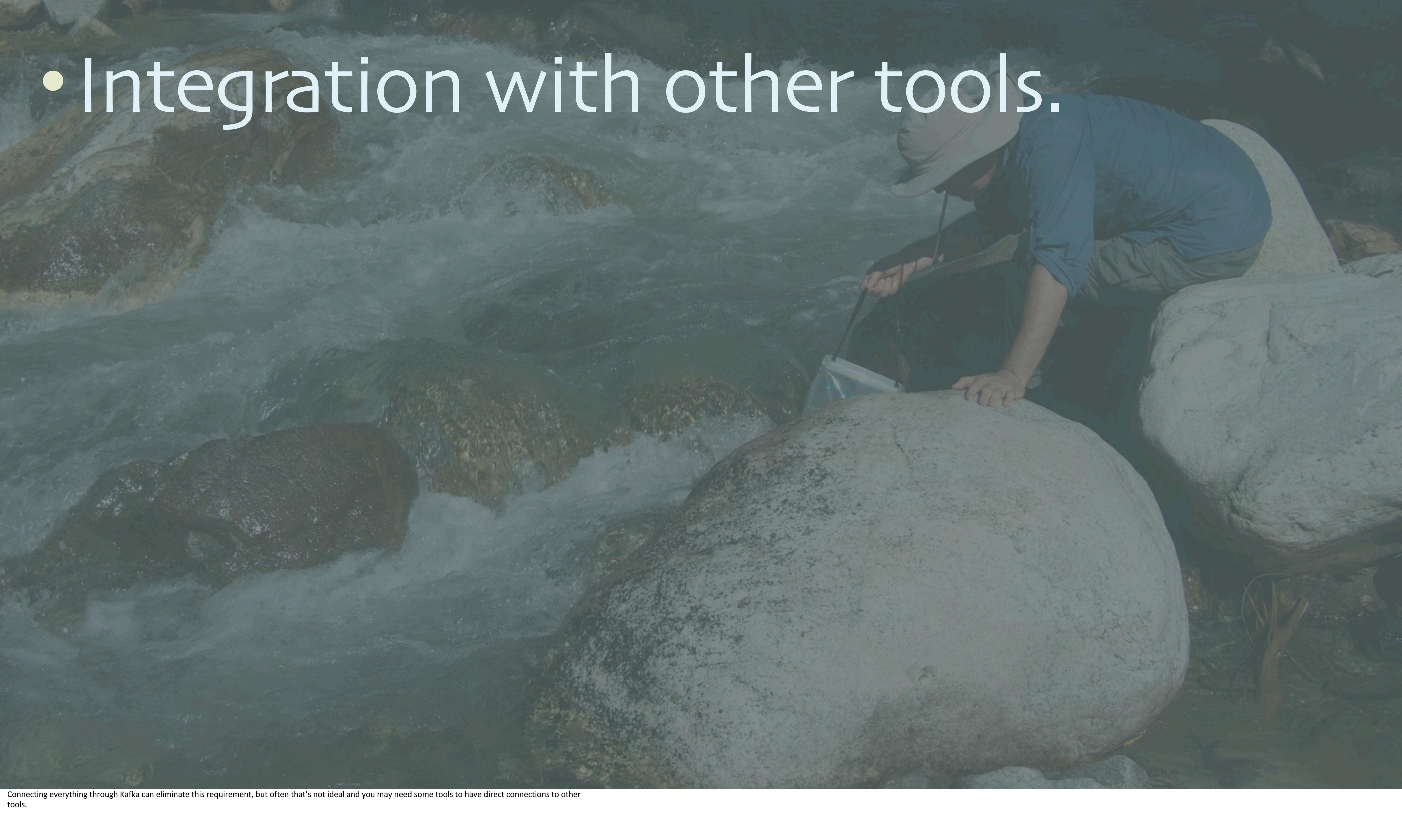
<http://www.reactivemanifesto.org/>

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

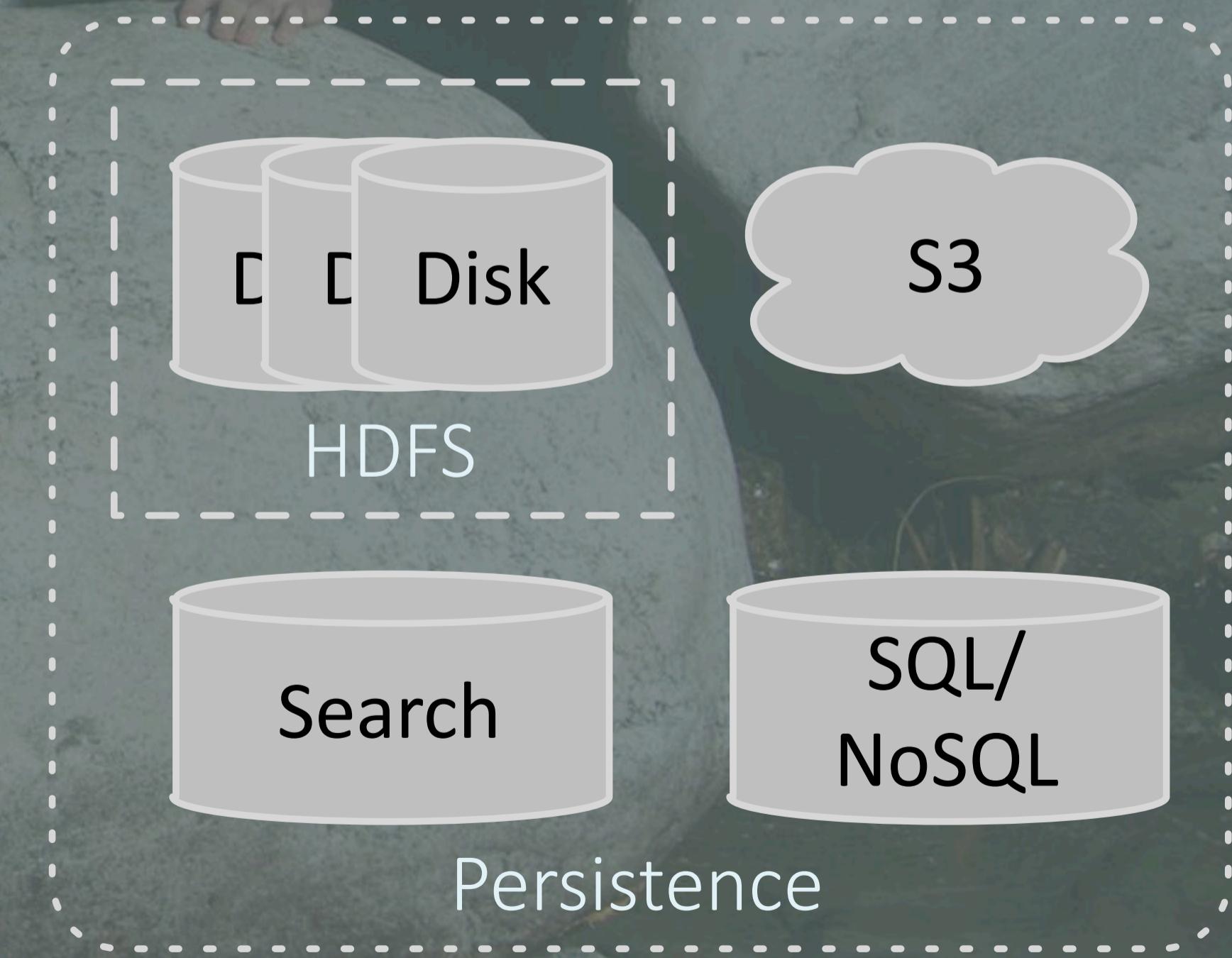


<https://store.nest.com/product/thermostat/>

- Integration with other tools.



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

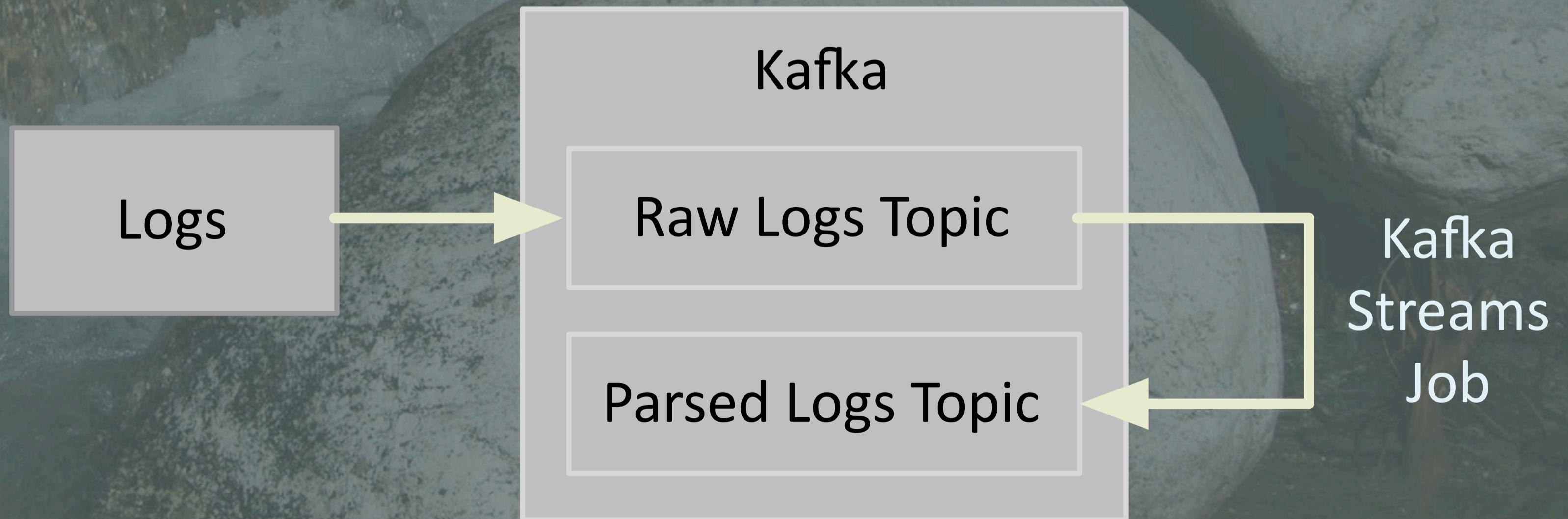


- Which kinds of data processing, analytics?
- SQL?

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

```
input.groupBy("zip-code").  
  count()
```

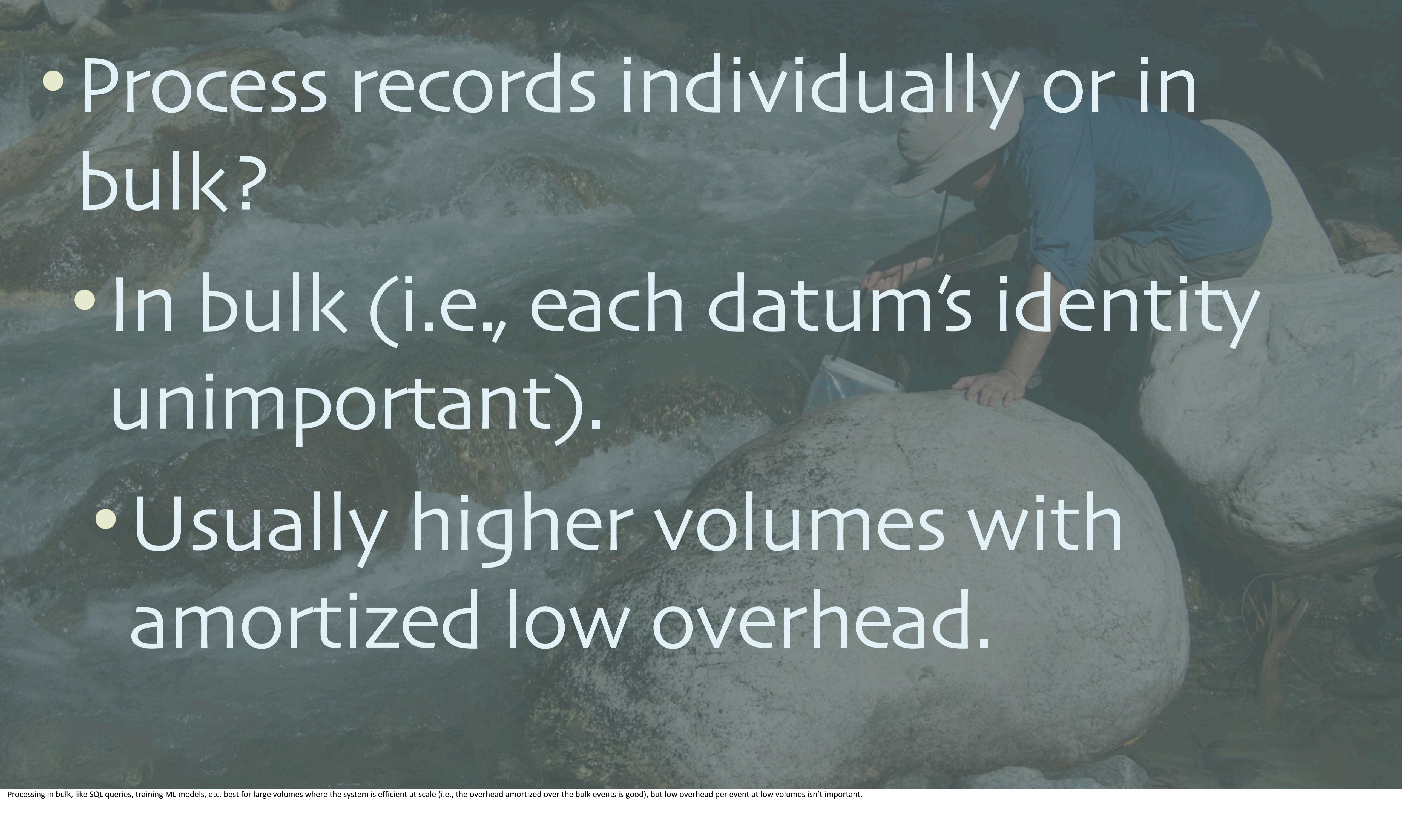
- Which kinds of data processing, analytics?
- Extract, transform, and load (ETL)?



- Which kinds of data processing, analytics?
- Machine learning?
- Train models: mini-batch to batch
- Serve models ("score" events): any latency

- Process records individually or in bulk?
- Individually (i.e., Complex event processing - CEP).
 - Usually lower volumes with per-event overhead low.

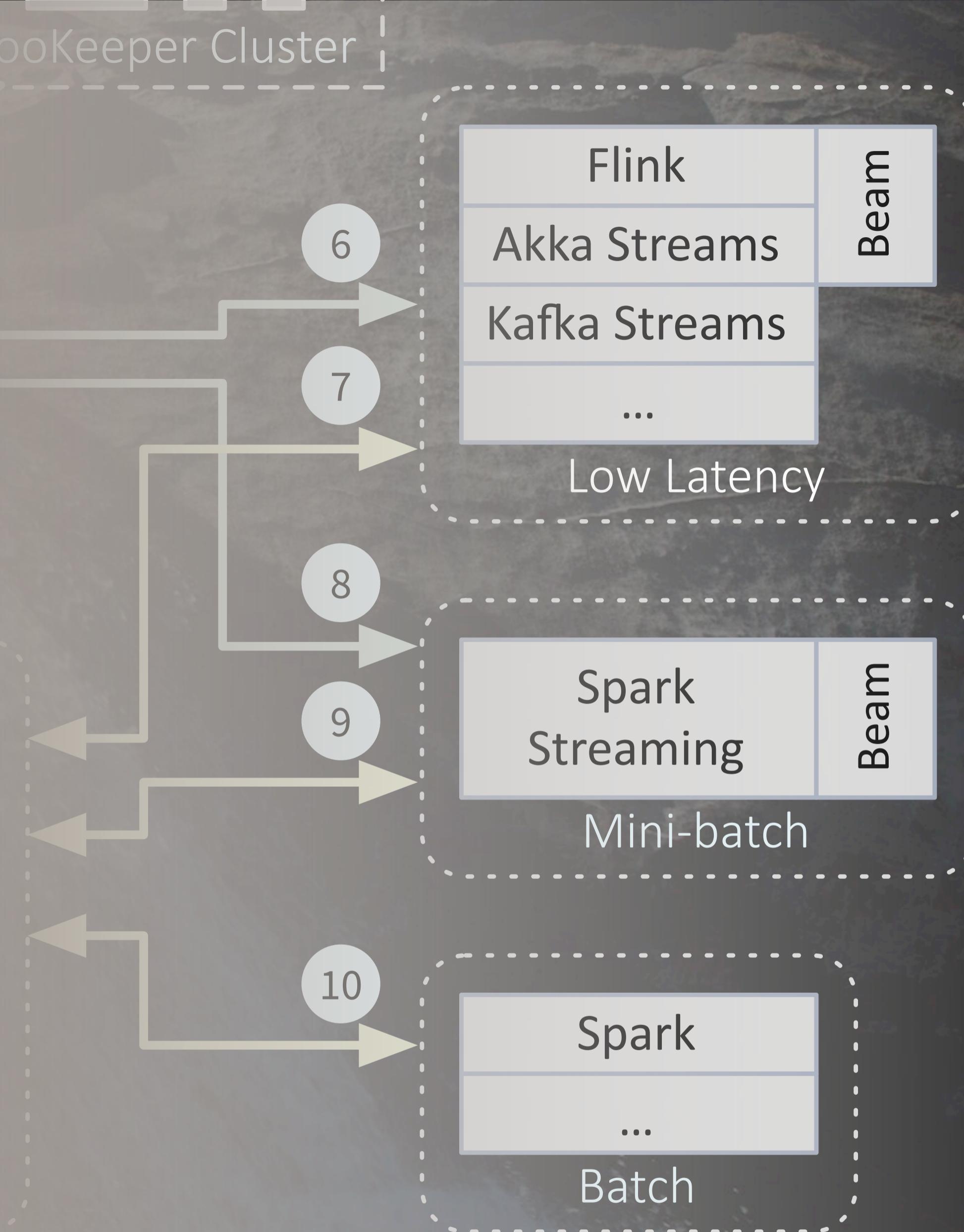
- Process records individually or in bulk?
- In bulk (i.e., each datum's identity unimportant).
 - Usually higher volumes with amortized low overhead.



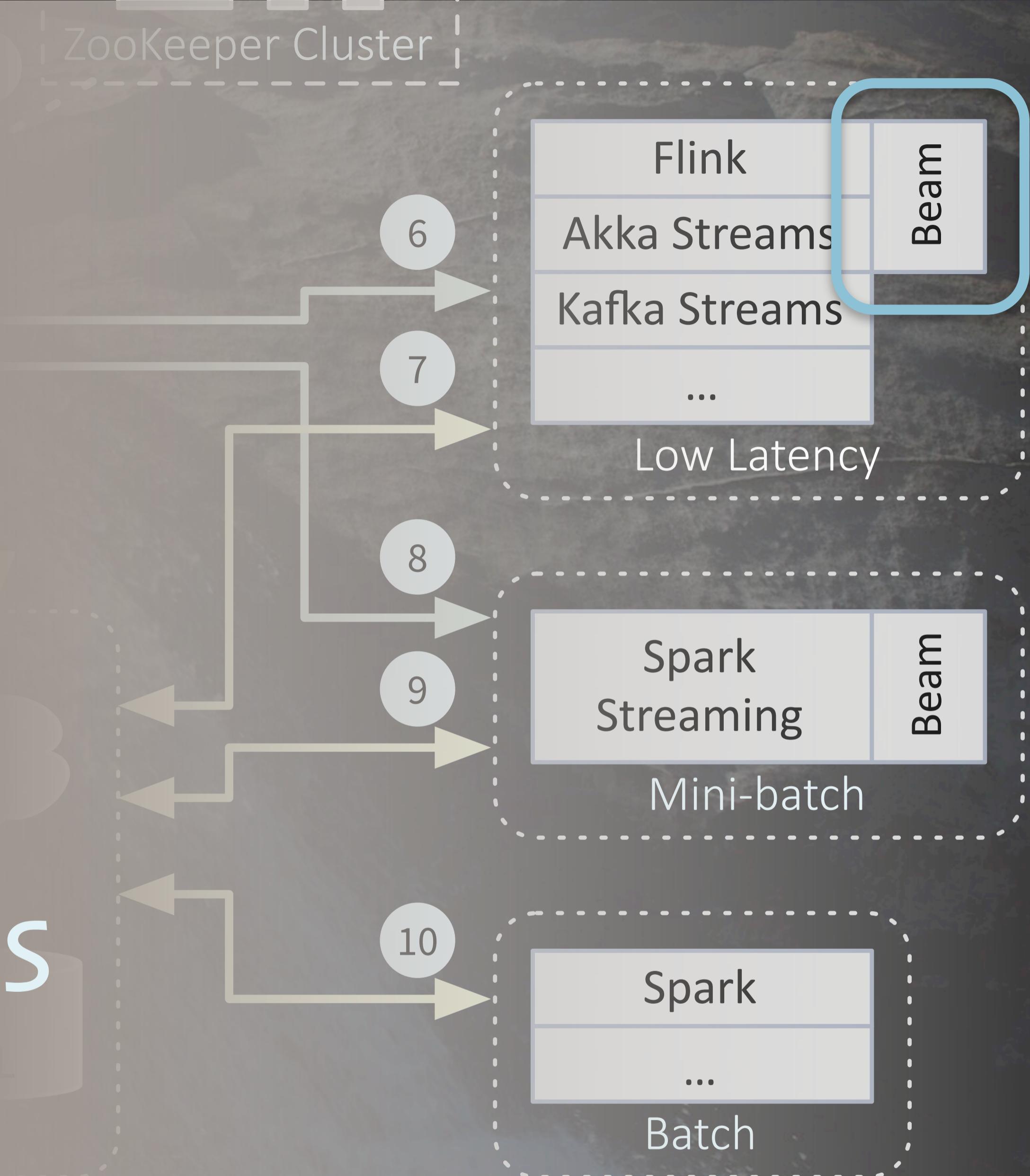


Best of Breed Streaming Engines

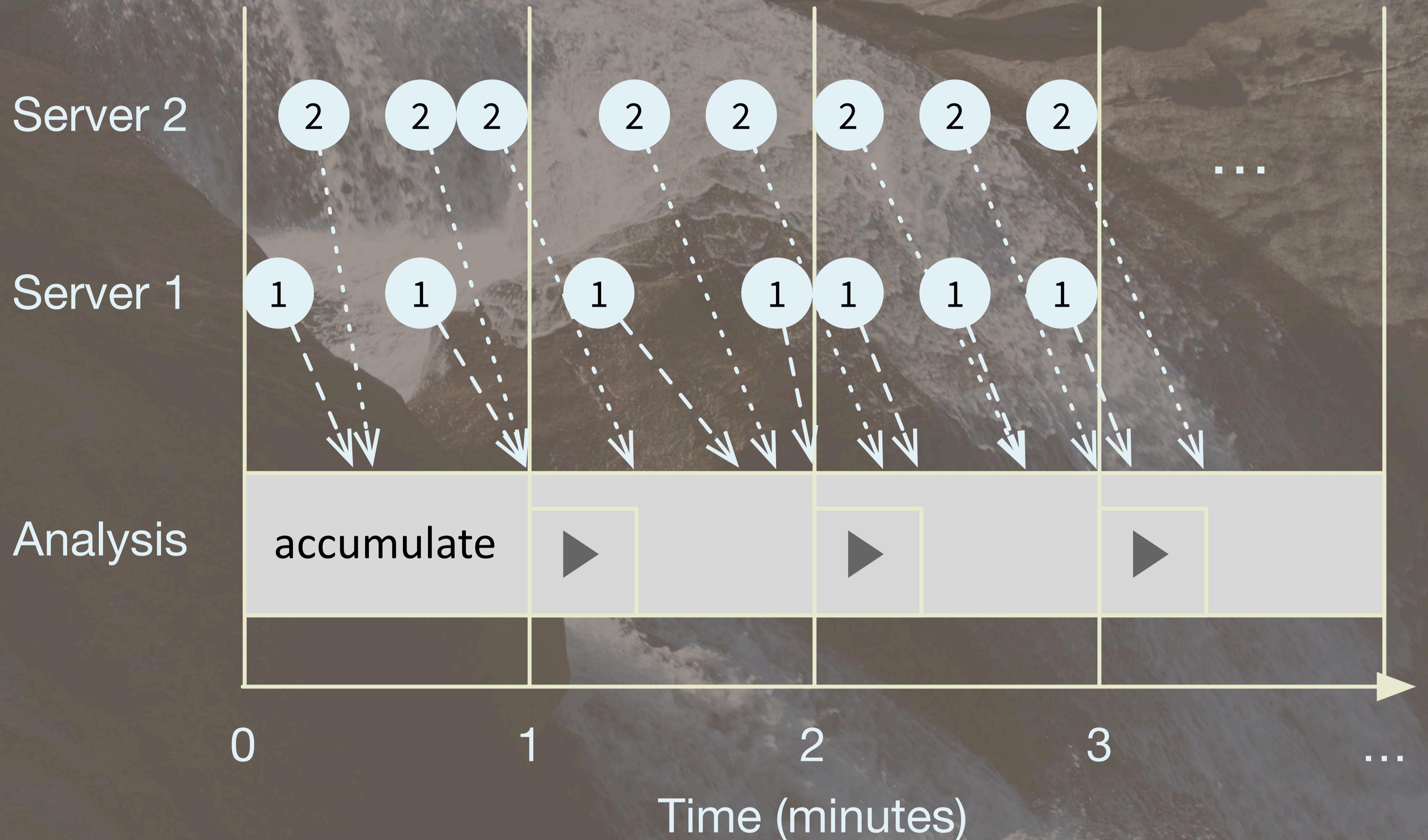
ZooKeeper Cluster



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



<https://conferences.oreilly.com/strata/strata-ca/public/schedule/detail/>



Key

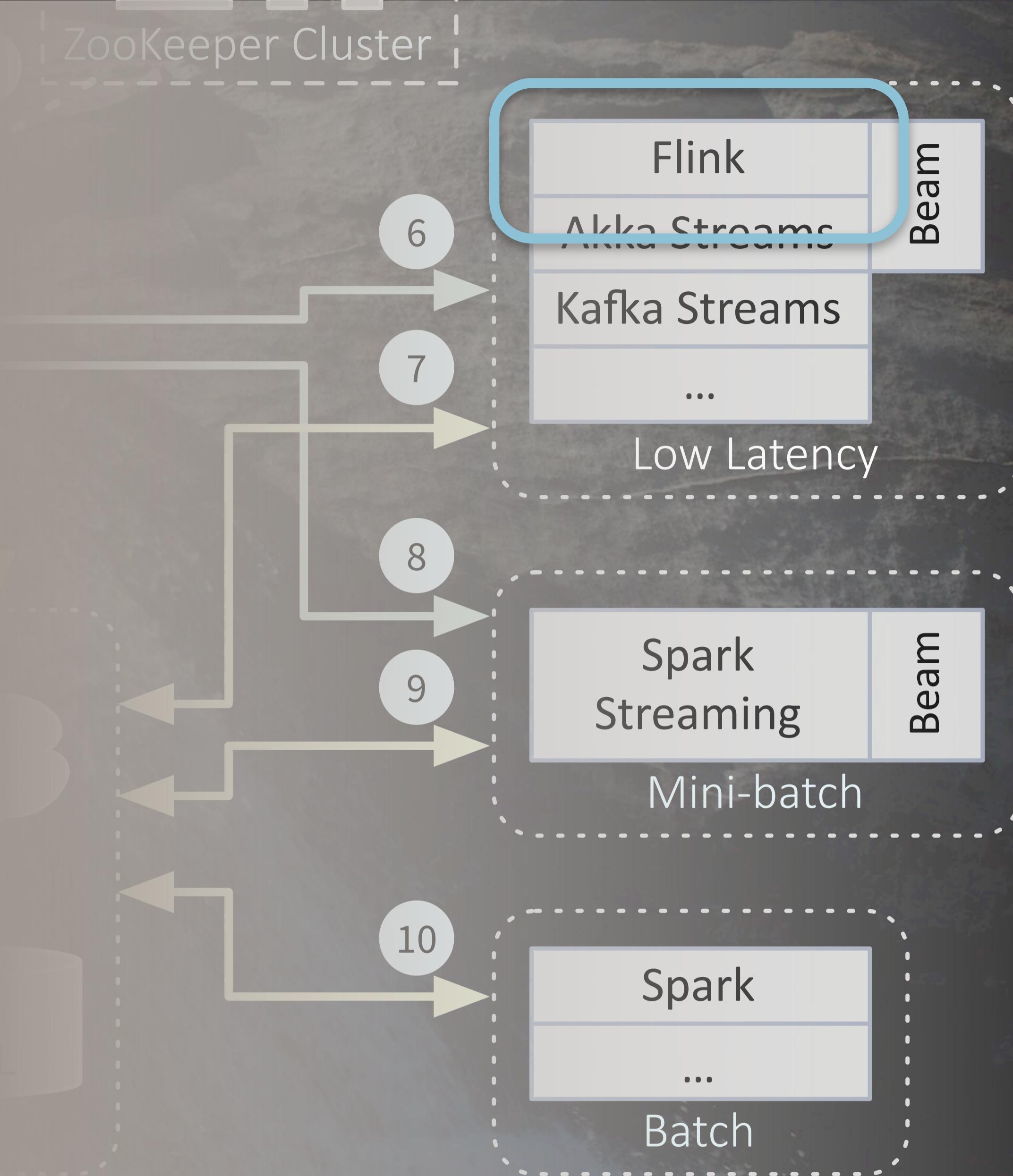
accumulate

Collect data,
Then process



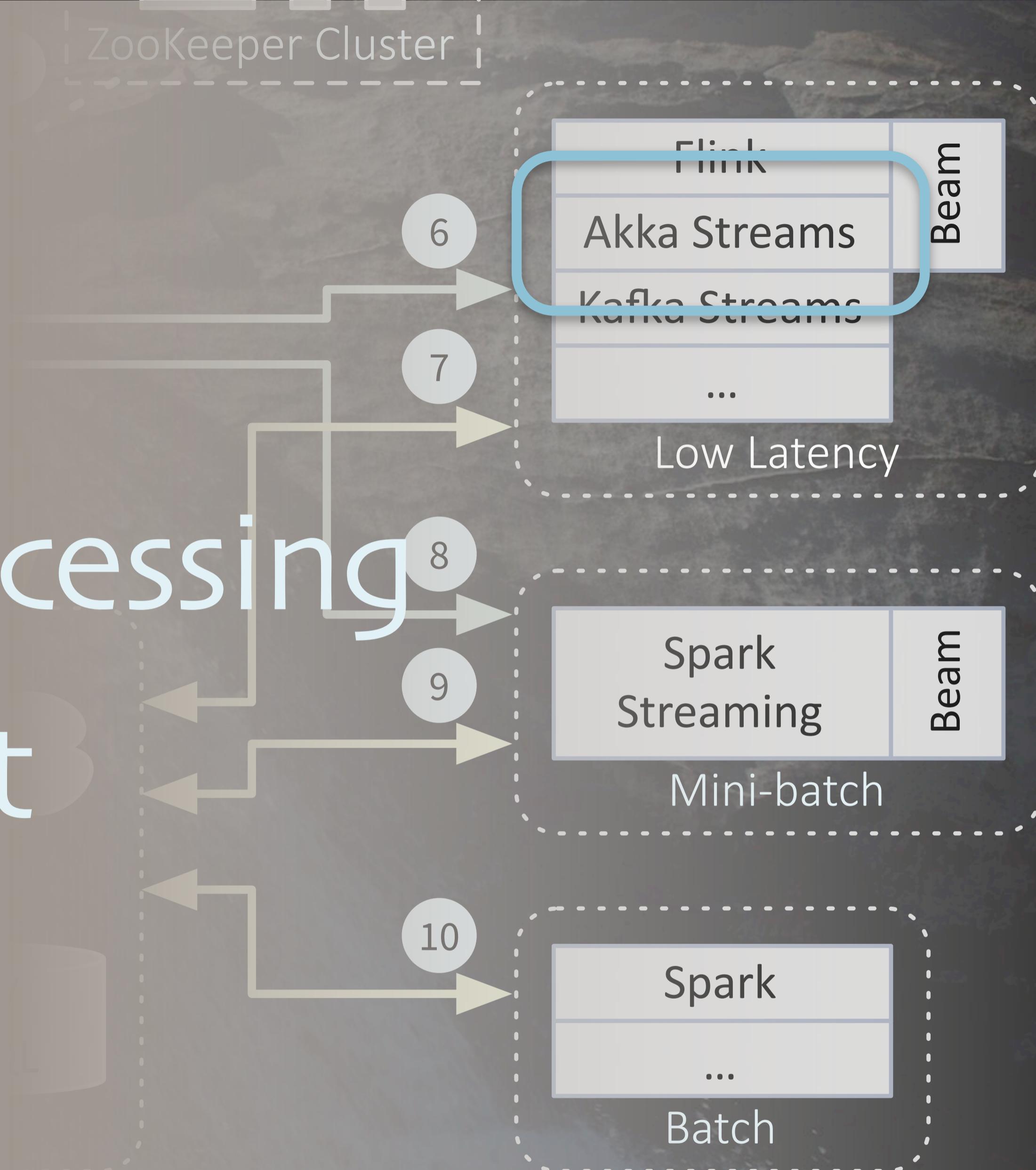
Event at Server n
propagated to
Analysis

- Apache Flink
 - High volume
 - Low latency
 - Beam Runner
 - Evolving SQL, ML



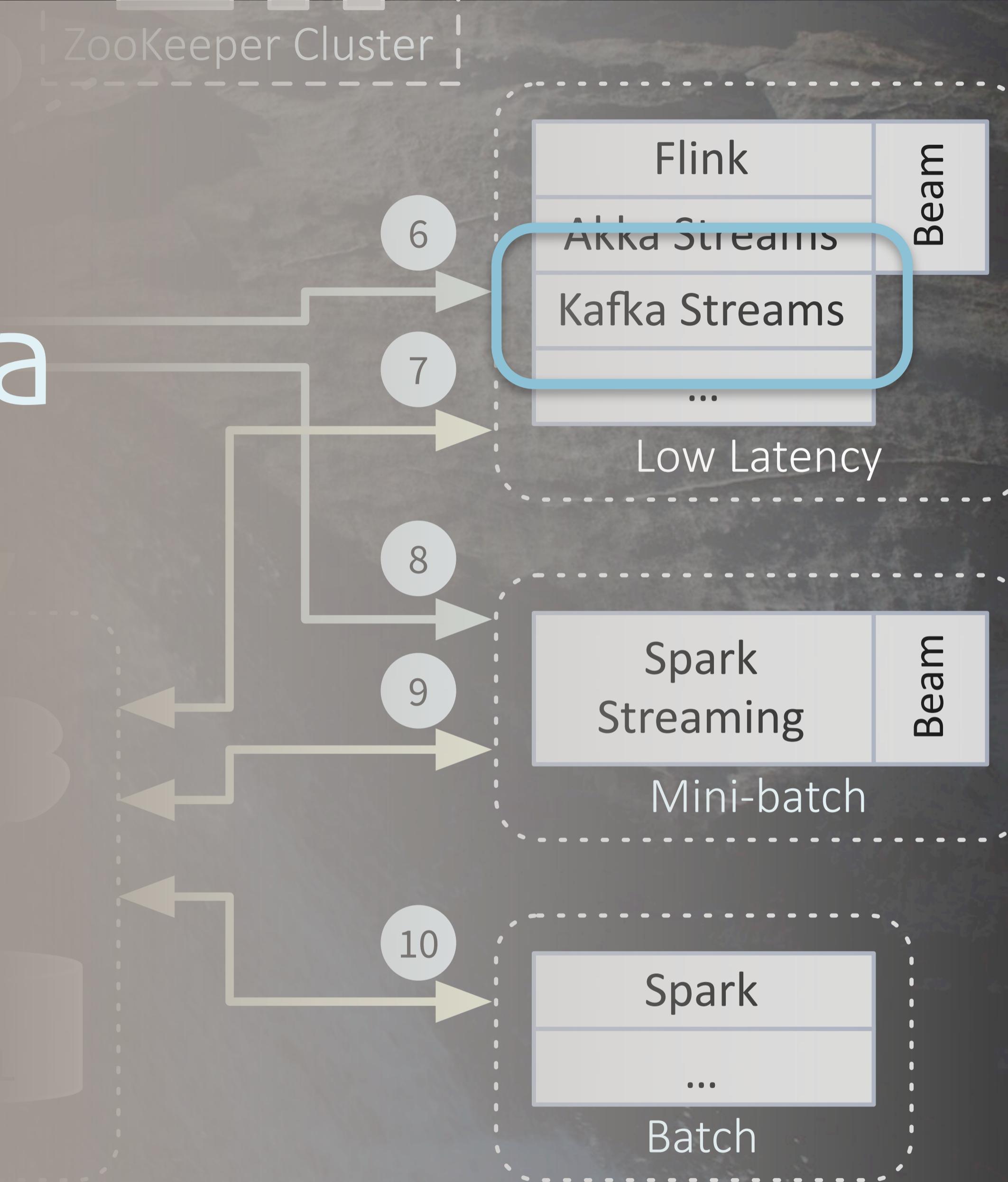
Akka Streams

- Low latency
- Complex Event Processing
- Efficient, per event
- Mid-volume pipes



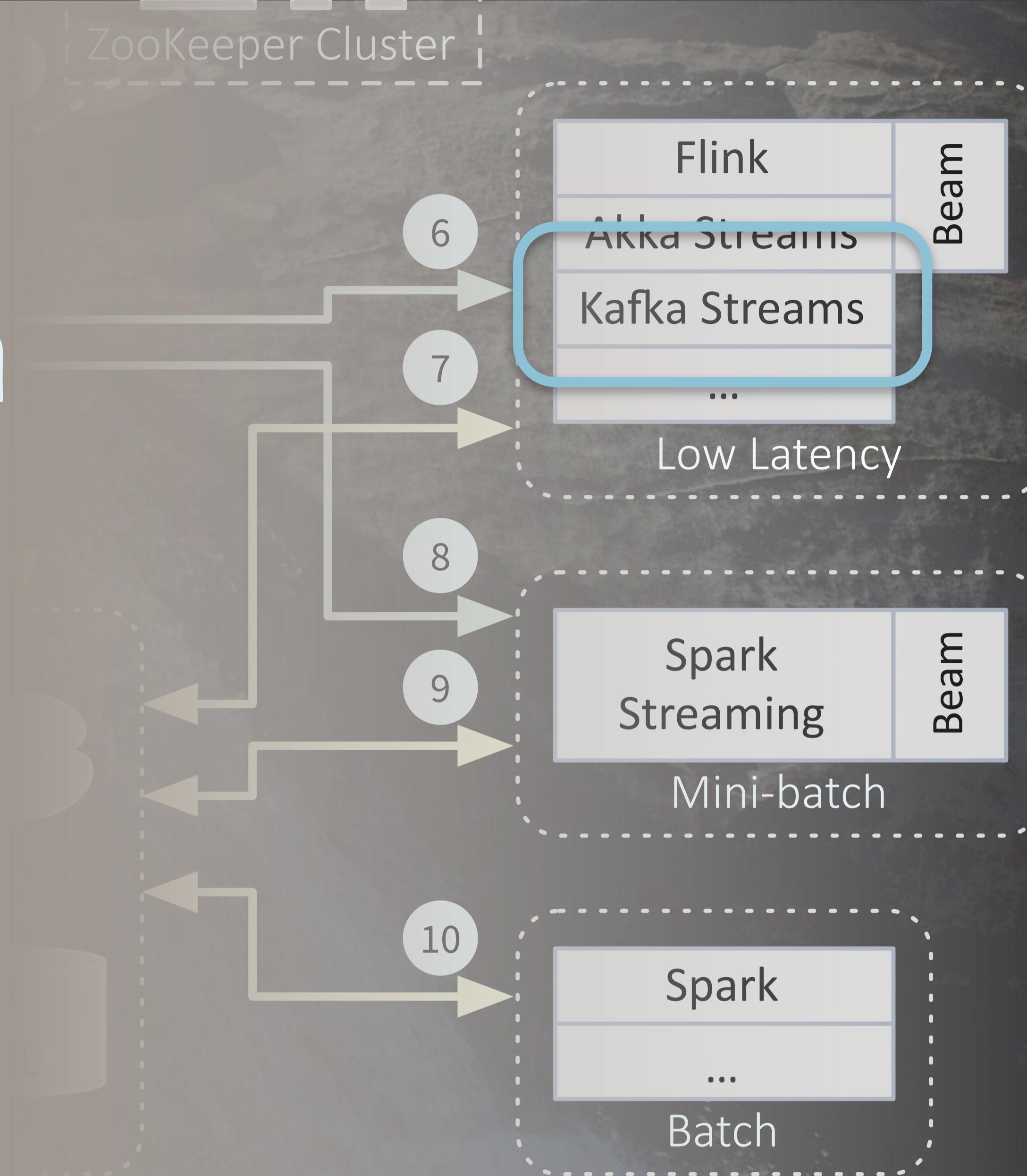
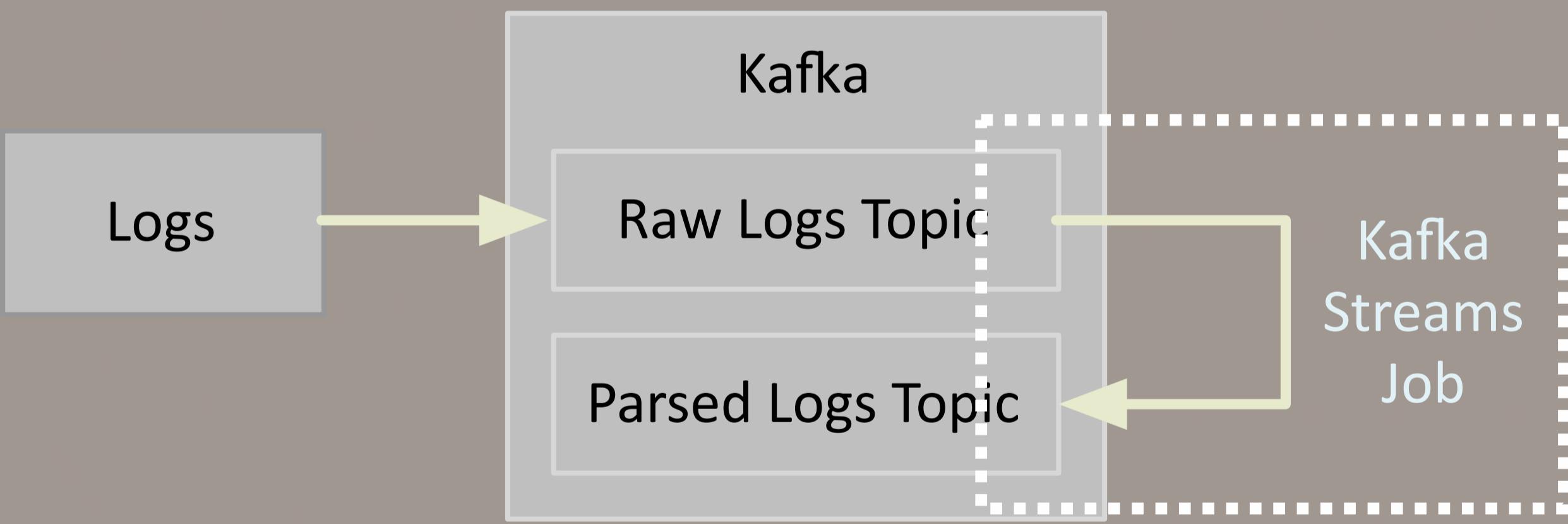
Kafka Streams

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



Kafka Streams

- “Exactly once” with Kafka 0.11.0

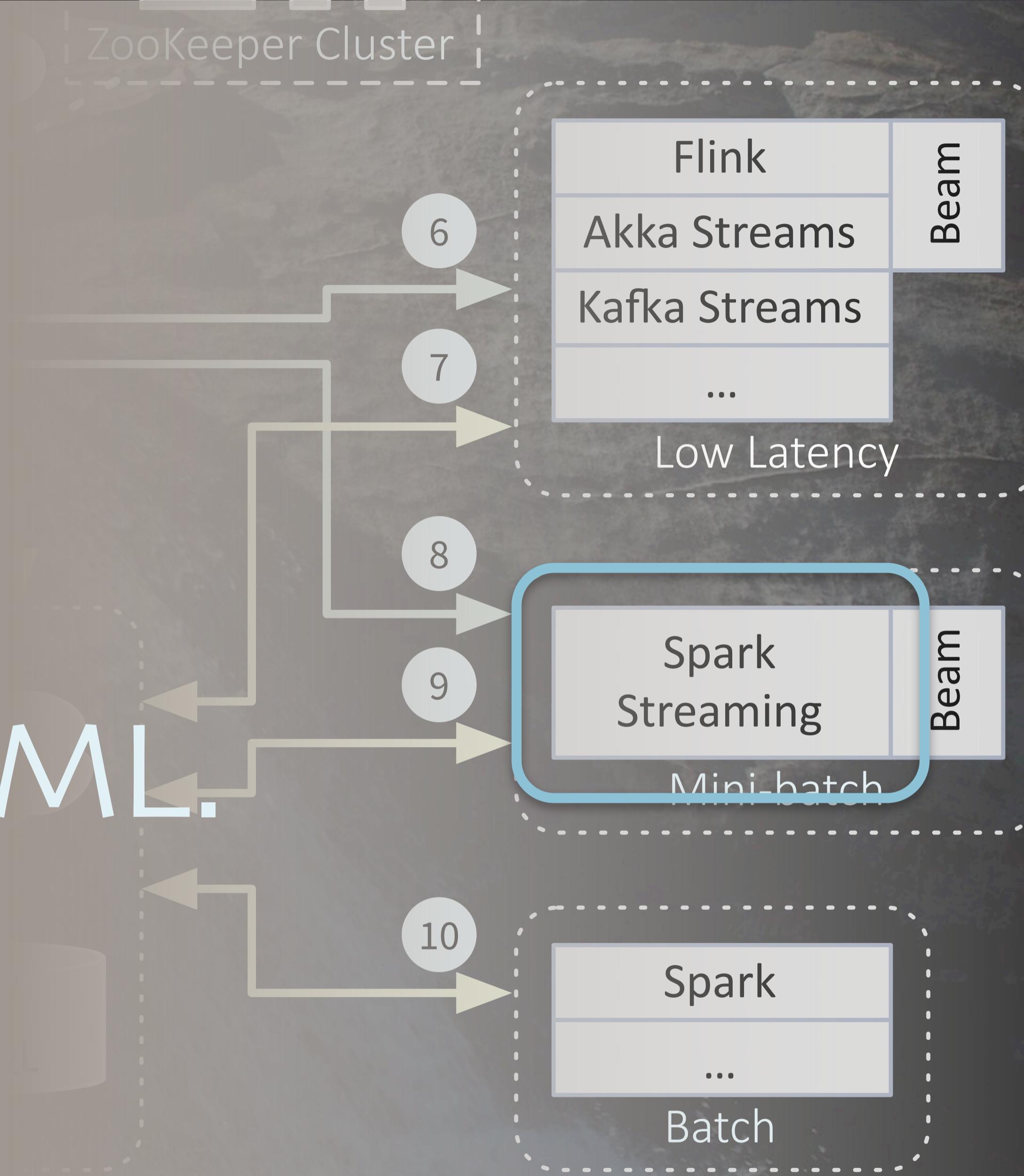


Spark Streaming

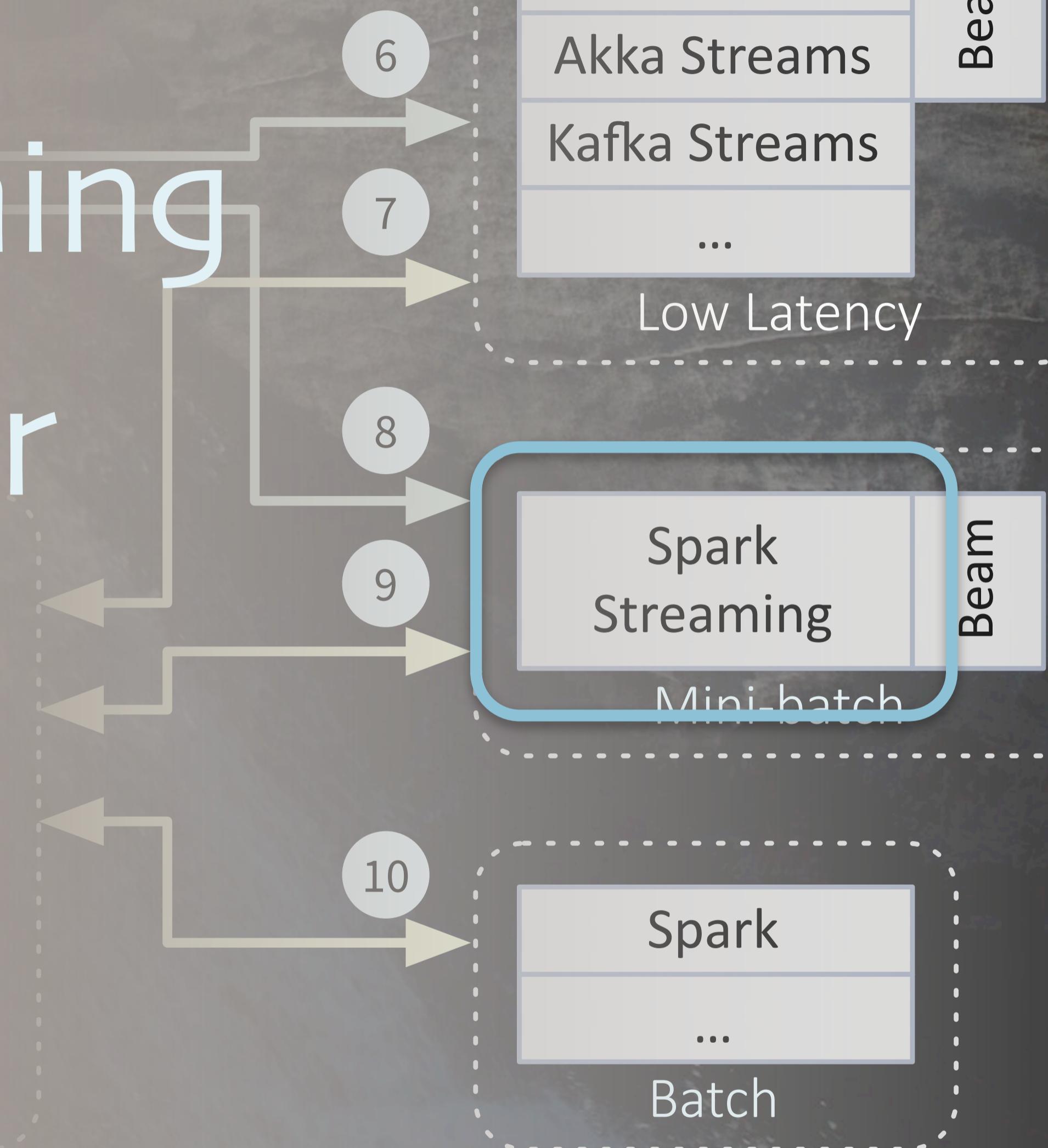
- Mini-batch model

- > 0.5 sec latency

- Ideal for Rich SQL, ML



- Spark Streaming
- True streaming - coming
- Beam support - under development

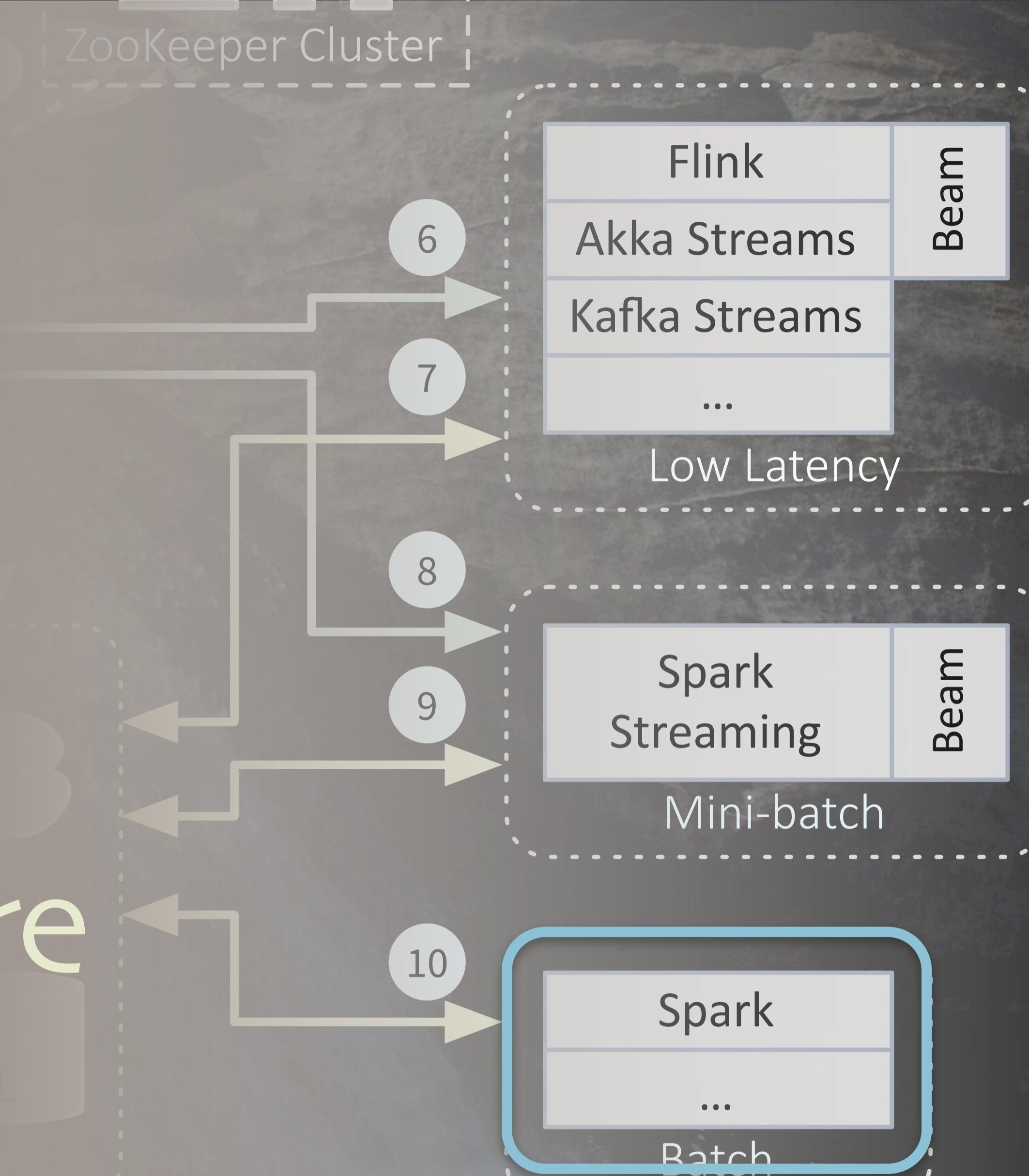


• Spark Batch

- Same features as streaming.

- Supports the Lambda Architecture

Lambda Architecture



- What about the Lambda Architecture?
- Implement analytics twice.
- Ad hoc, custom view
- So, less popular now

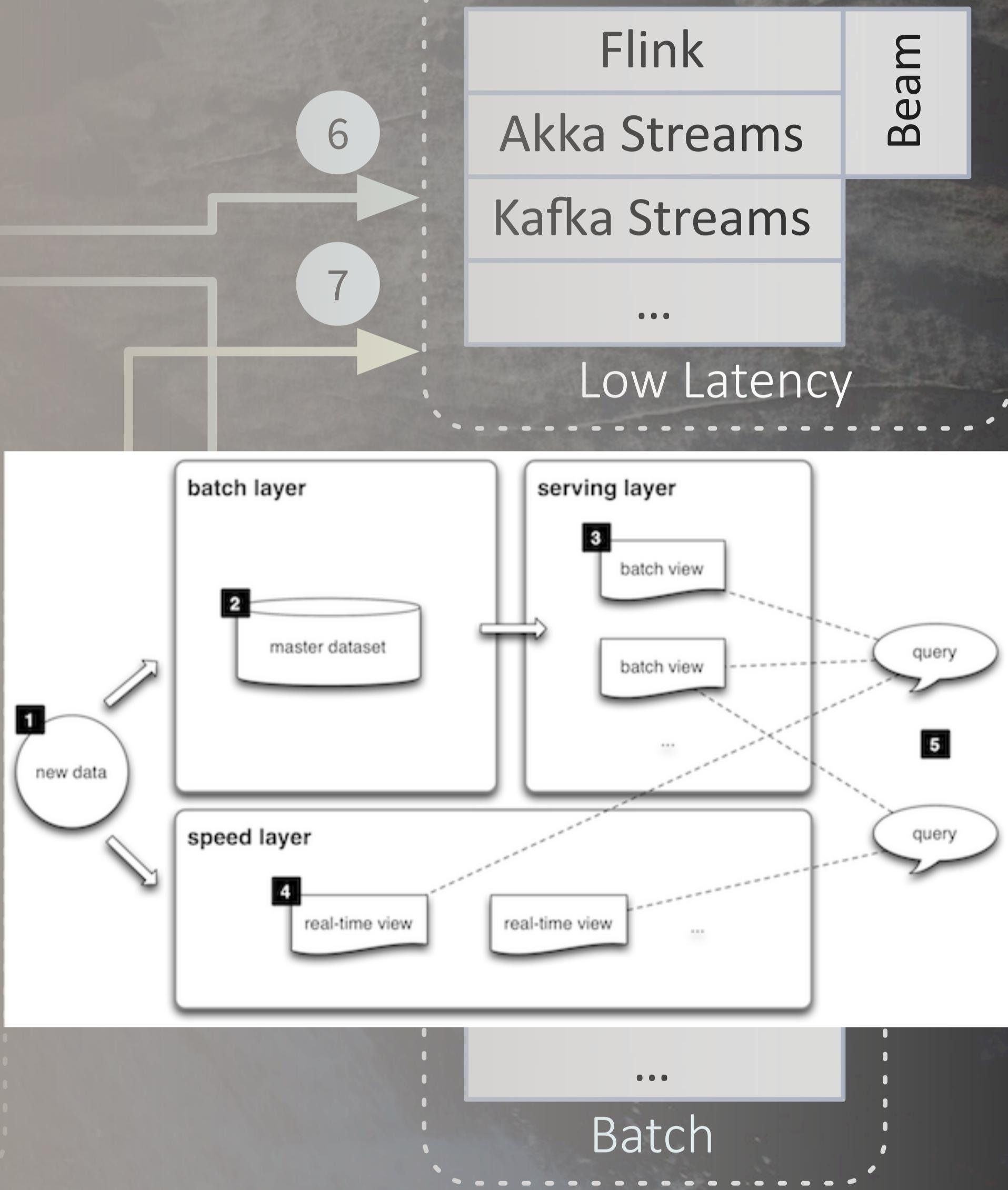


Image: <http://lambda-architecture.net/>

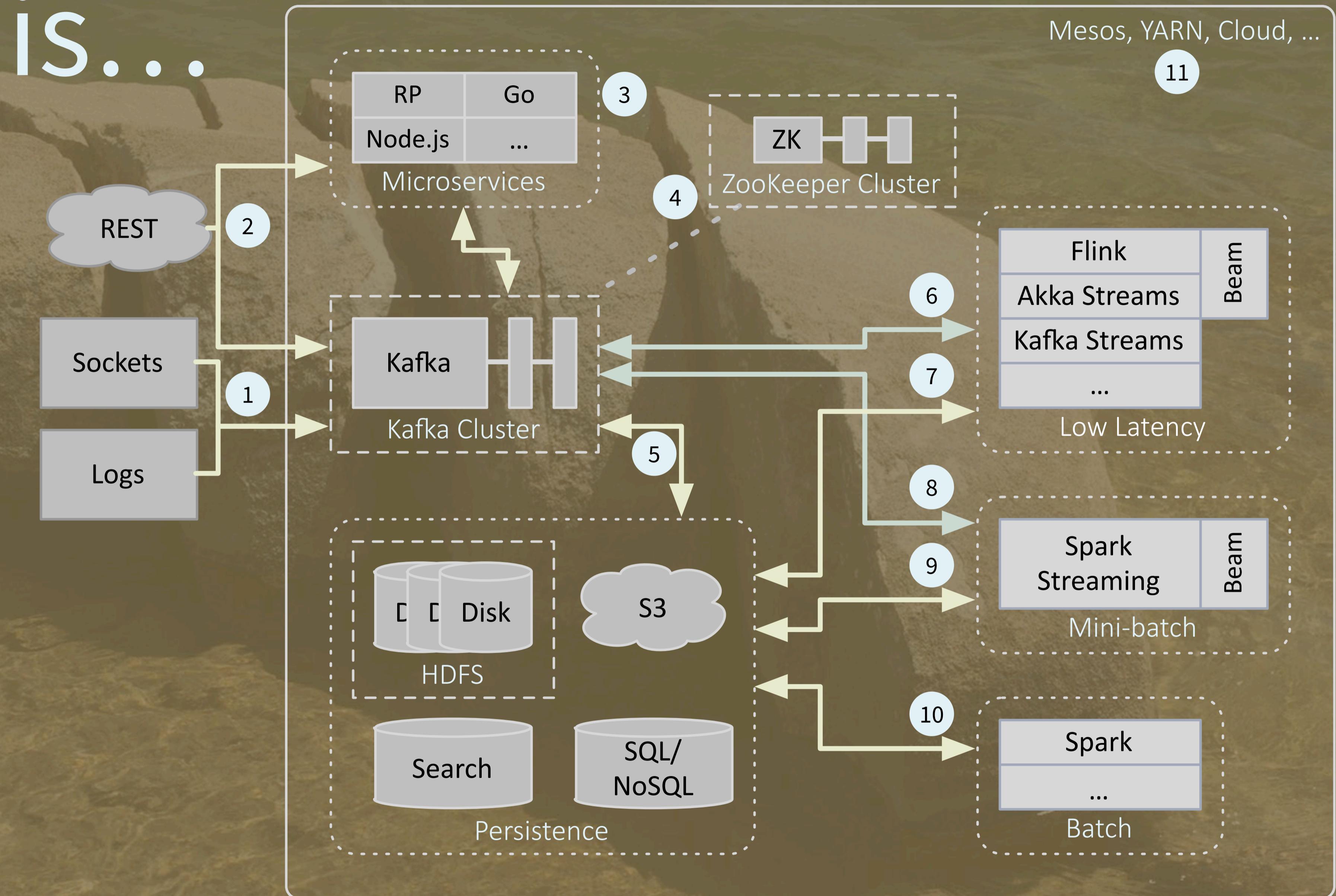
Microservices and Fast Data



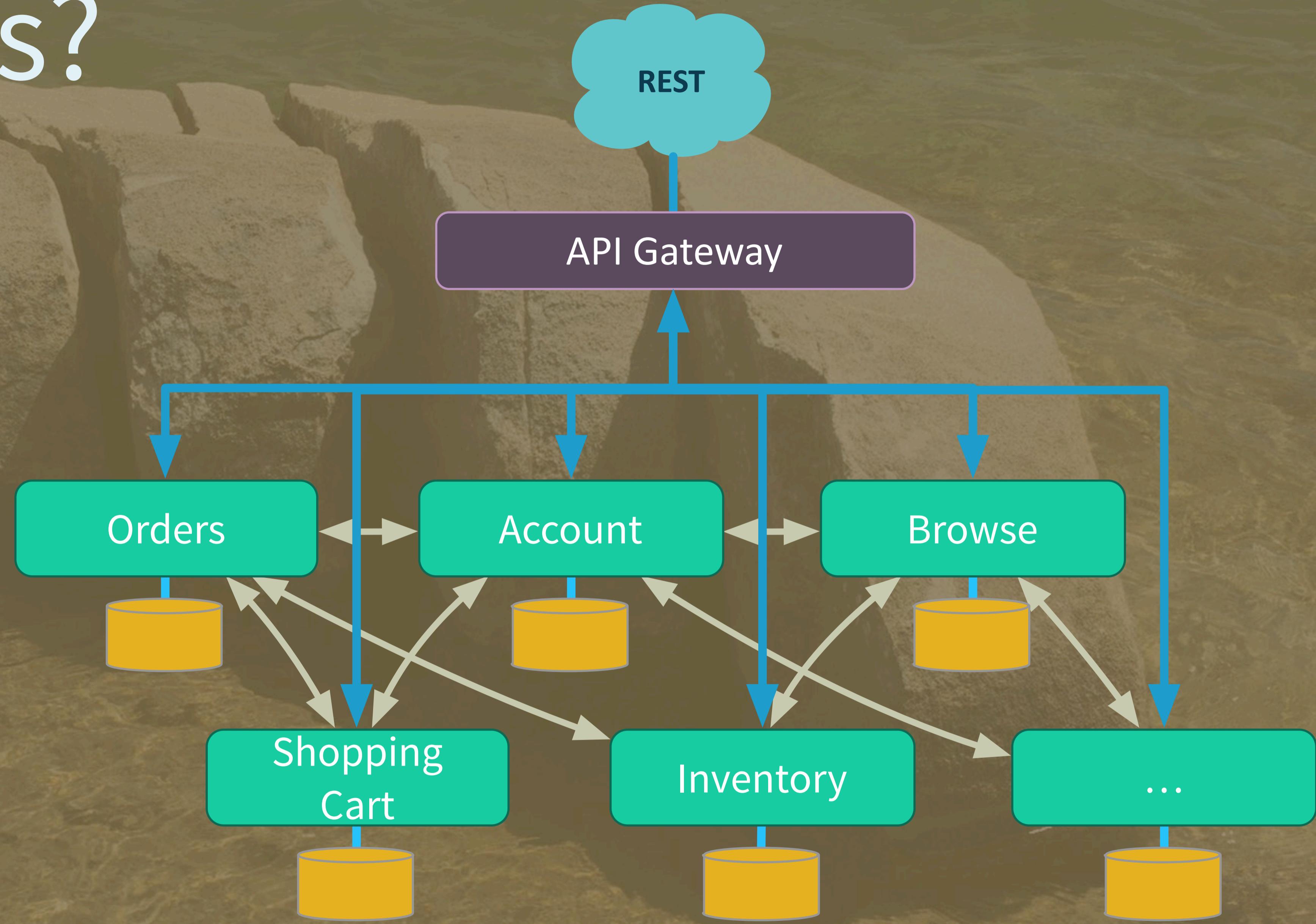
I'm going to argue that service architectures (classically three tier, but evolving...) and data architectures (classically Big Data like Hadoop) are converging.

Photo: "Sliced" rocks, 1000 Island Lake, Ansel Adams Wilderness, California.

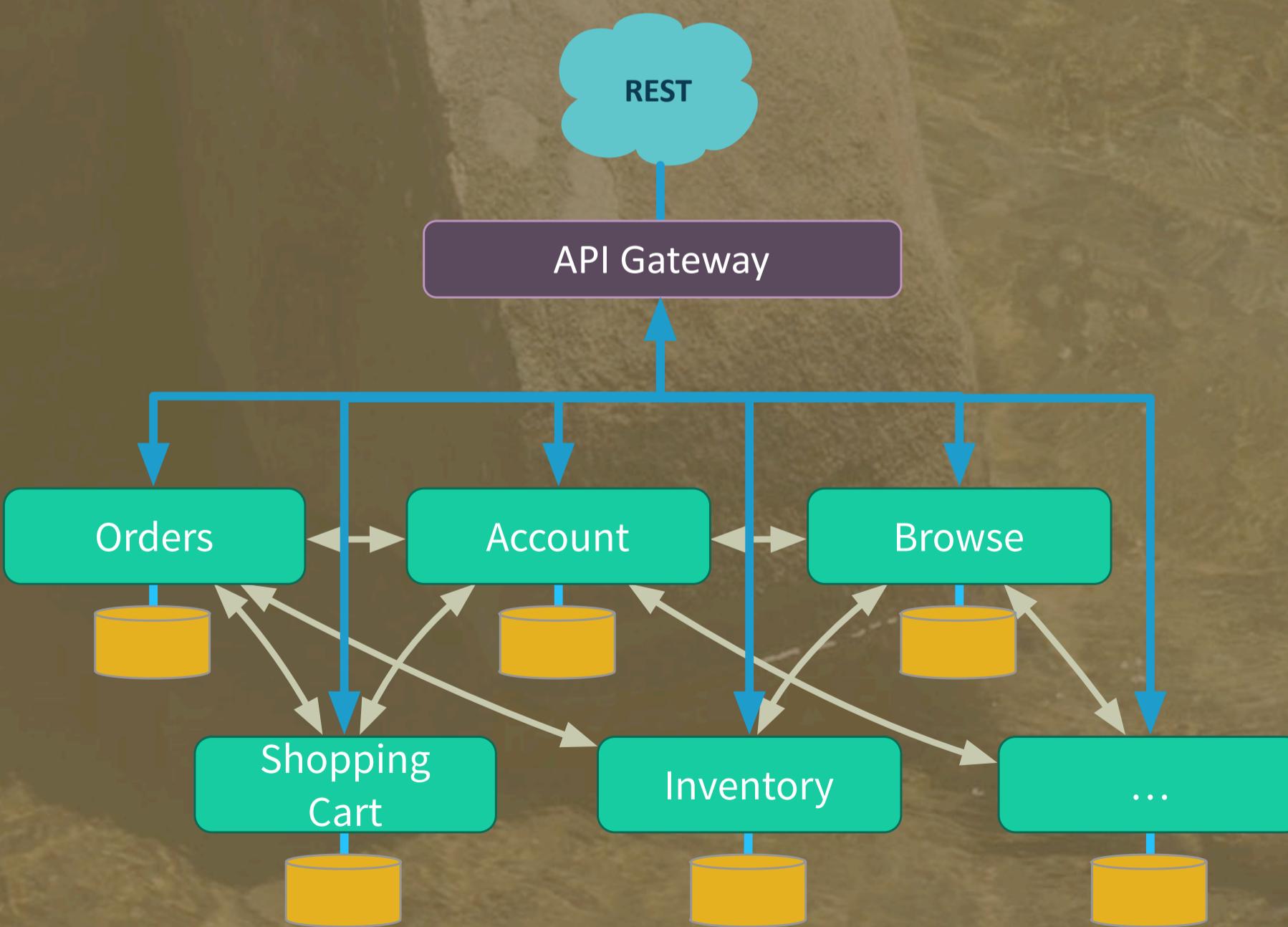
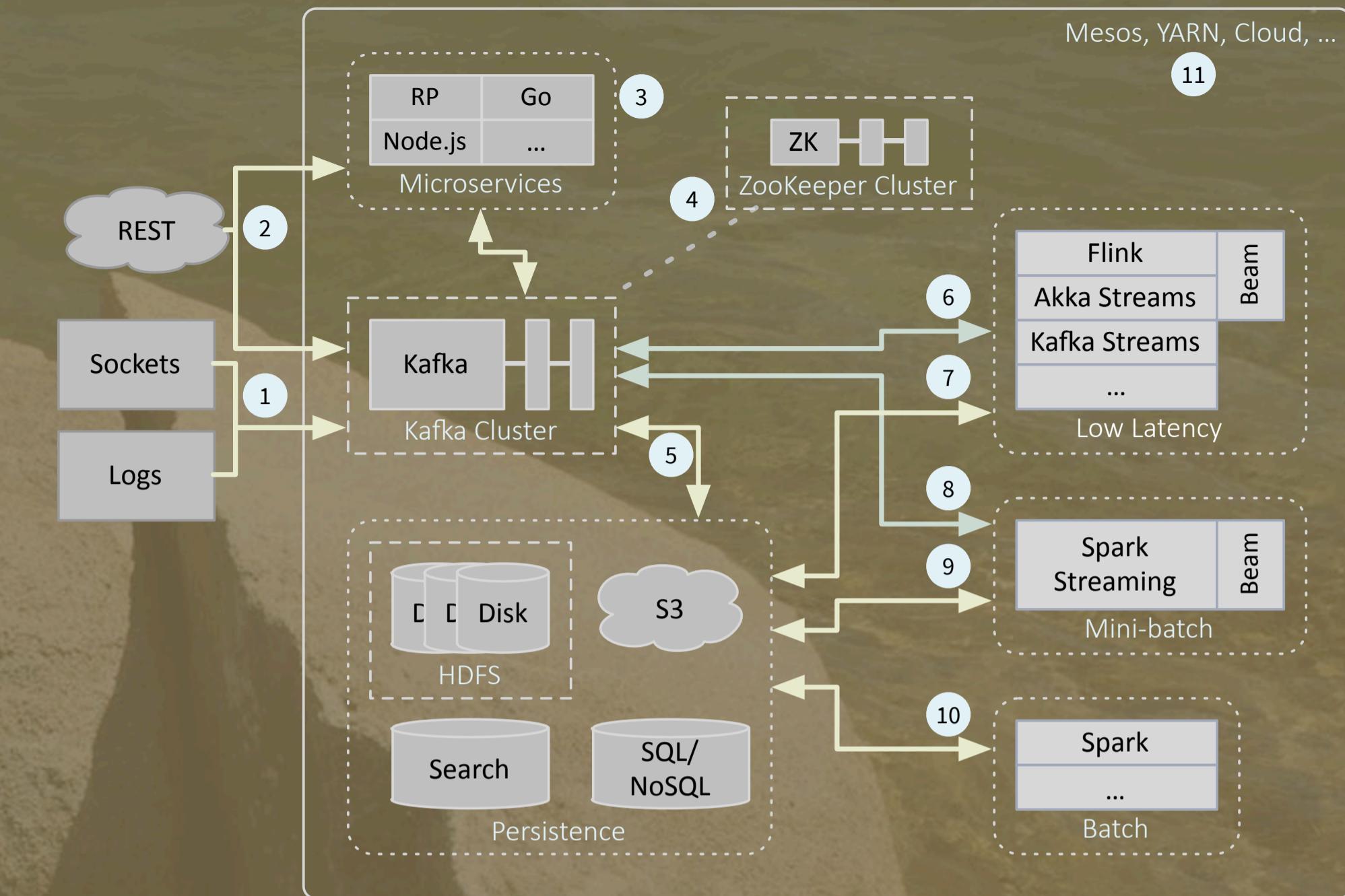
How is this...



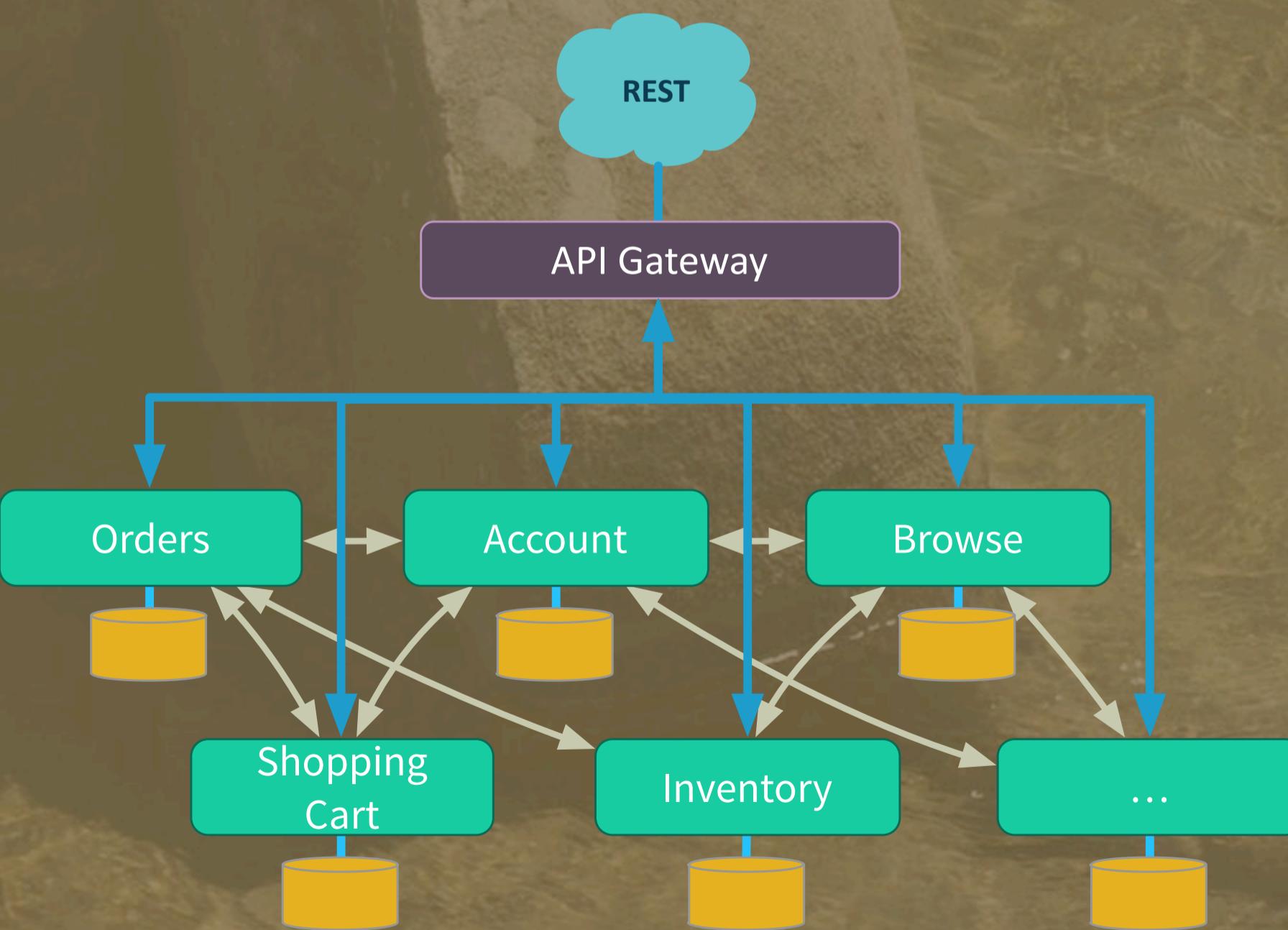
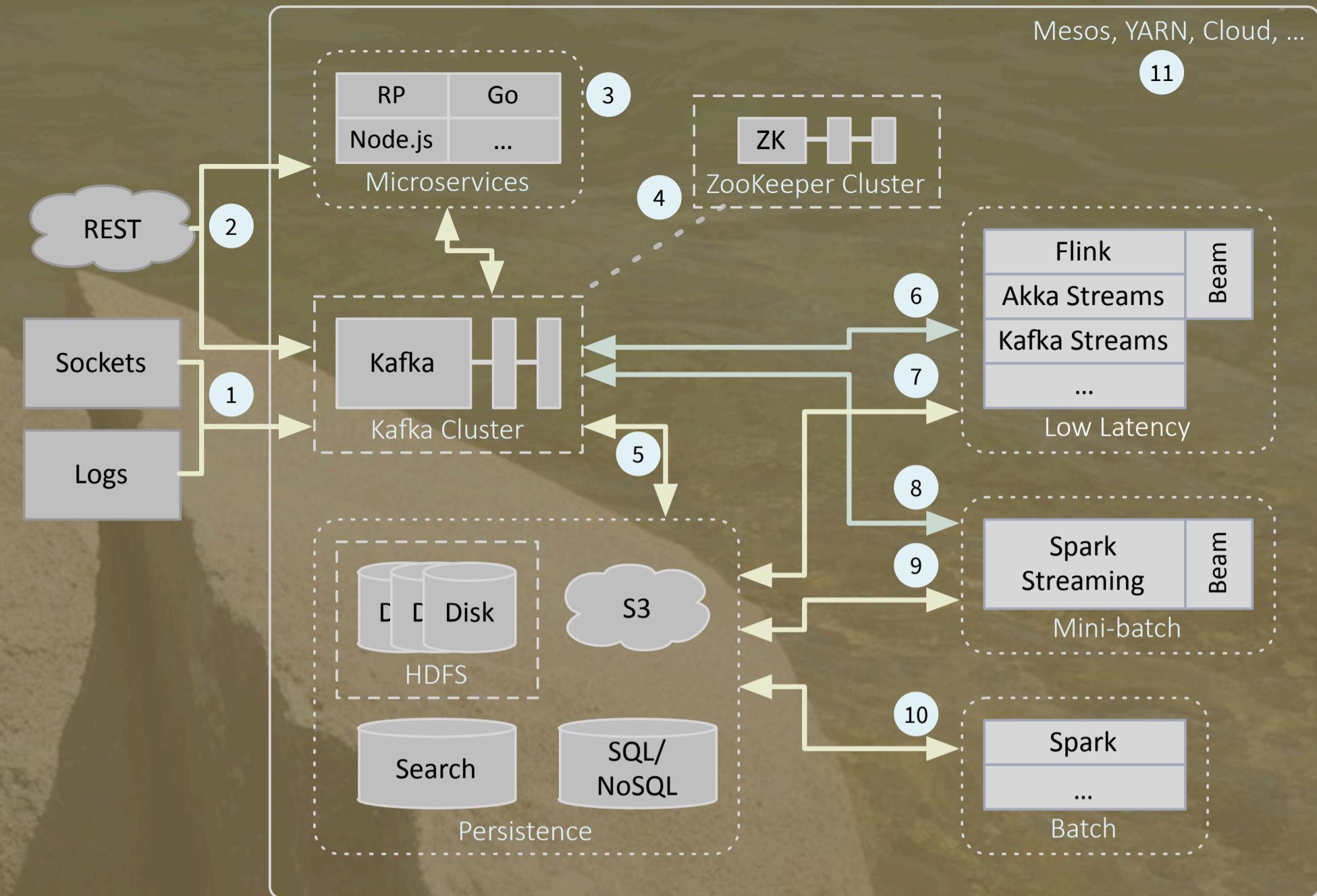
... like this?



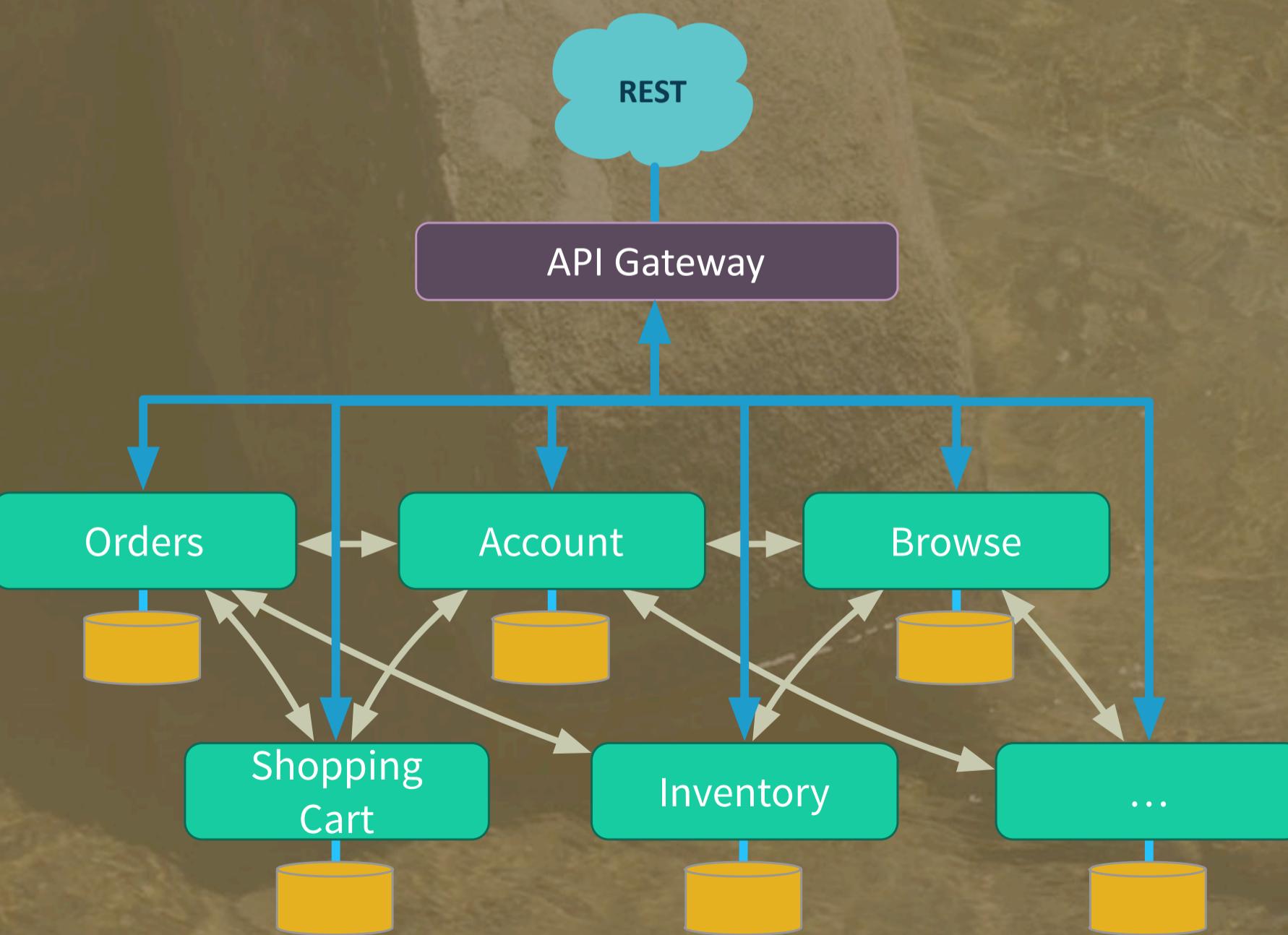
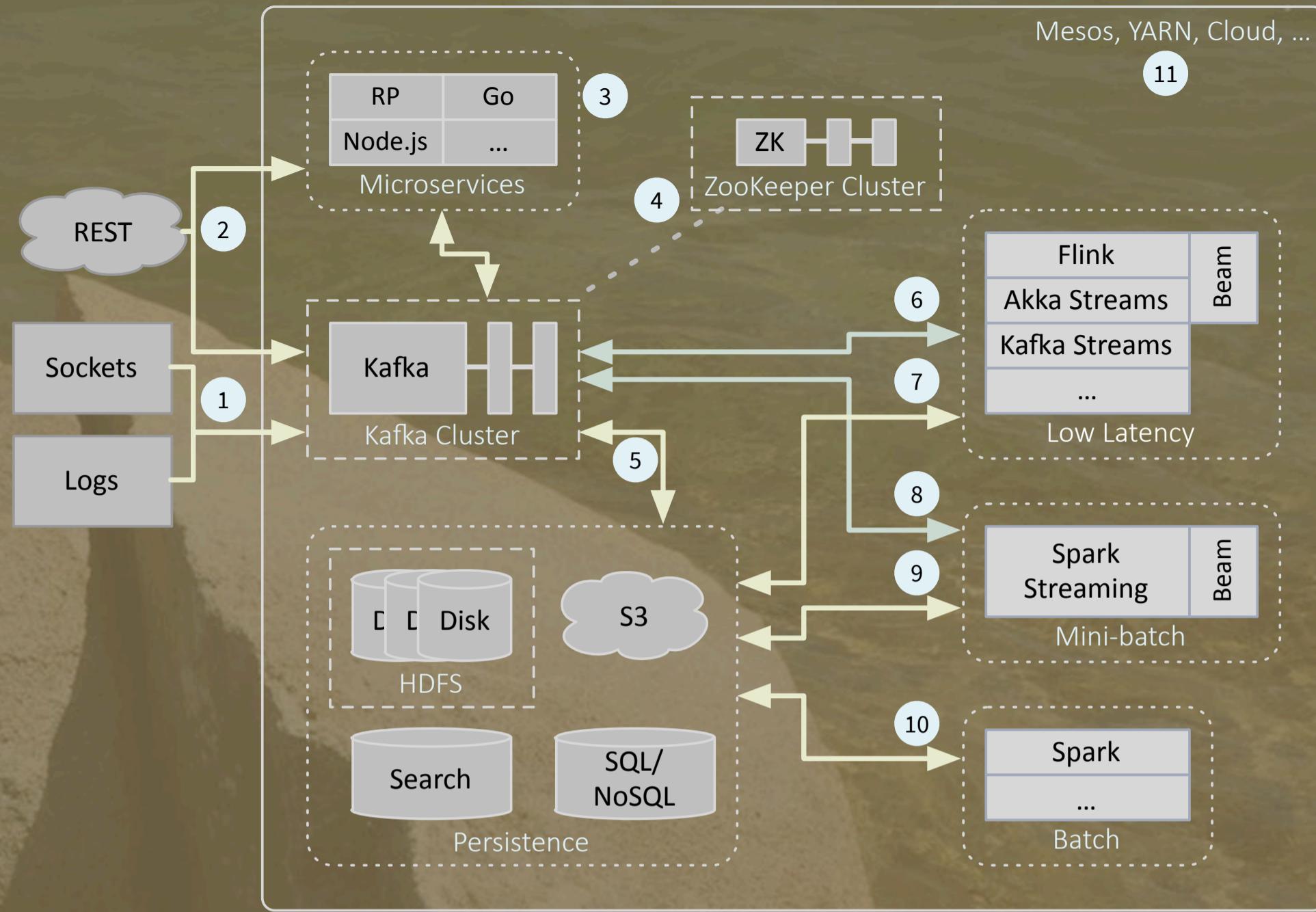
- A data app / microservice:
- A single responsibility.



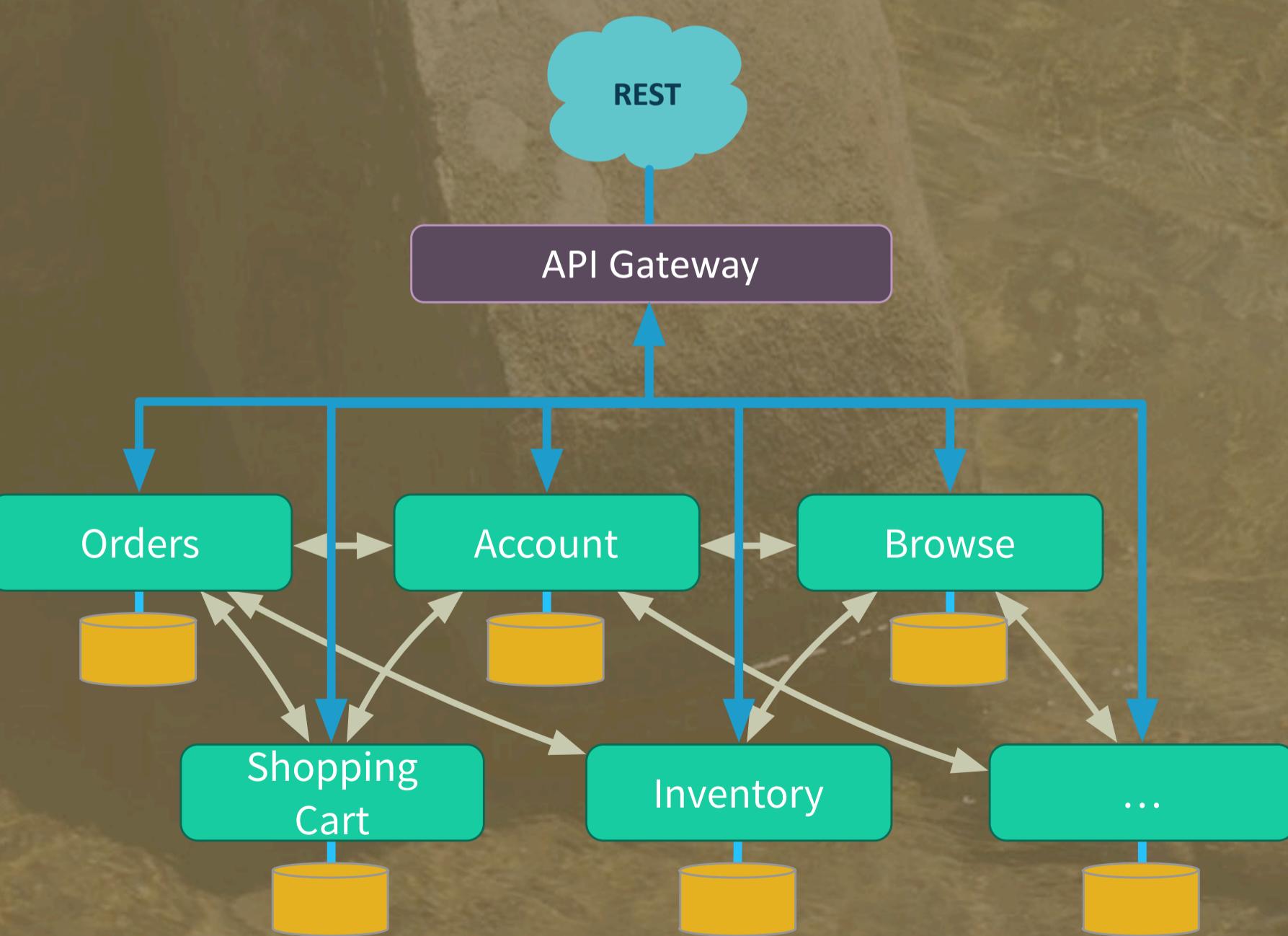
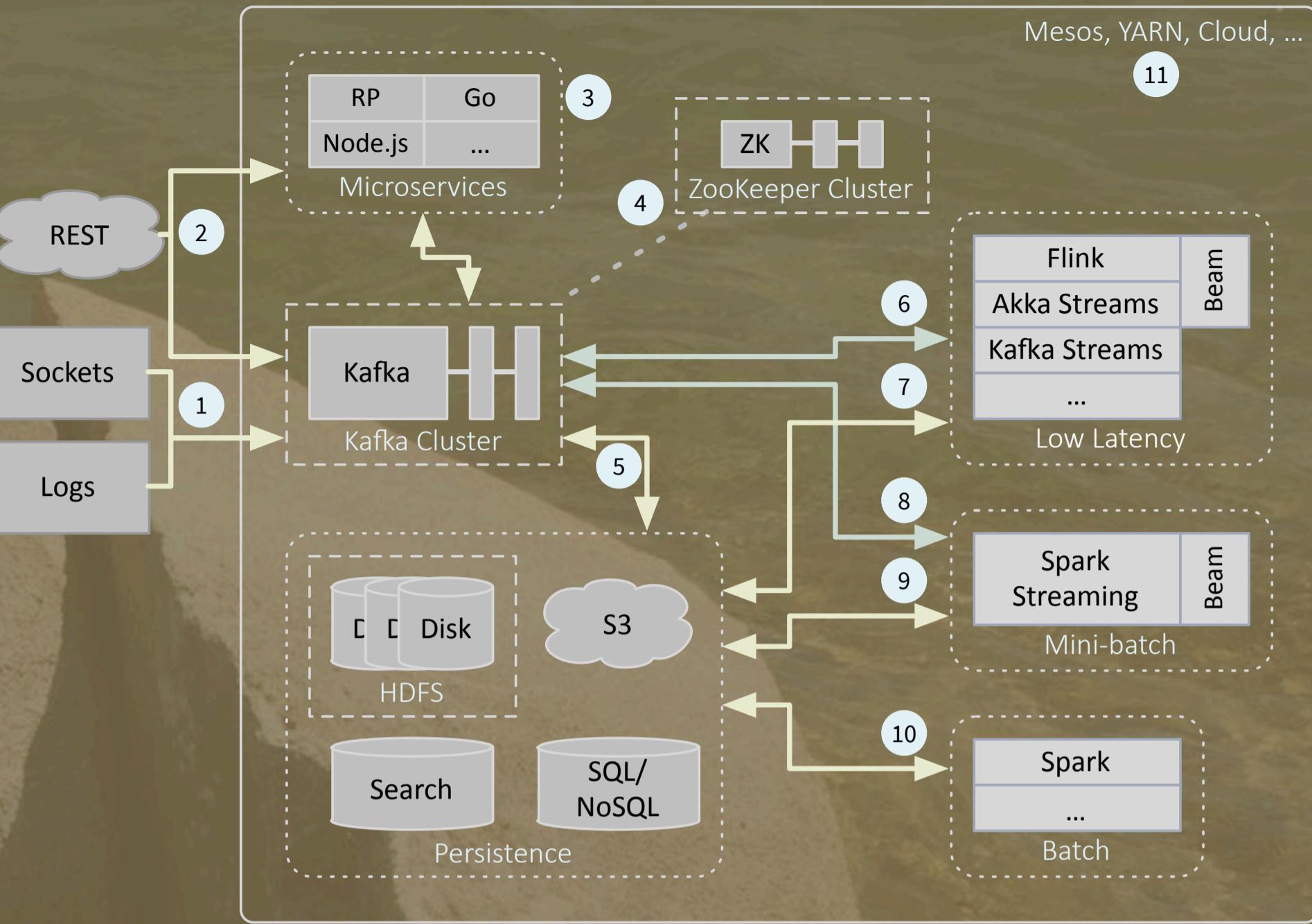
- A data app / microservice:
- A single responsibility.
- The input never ends.



- 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

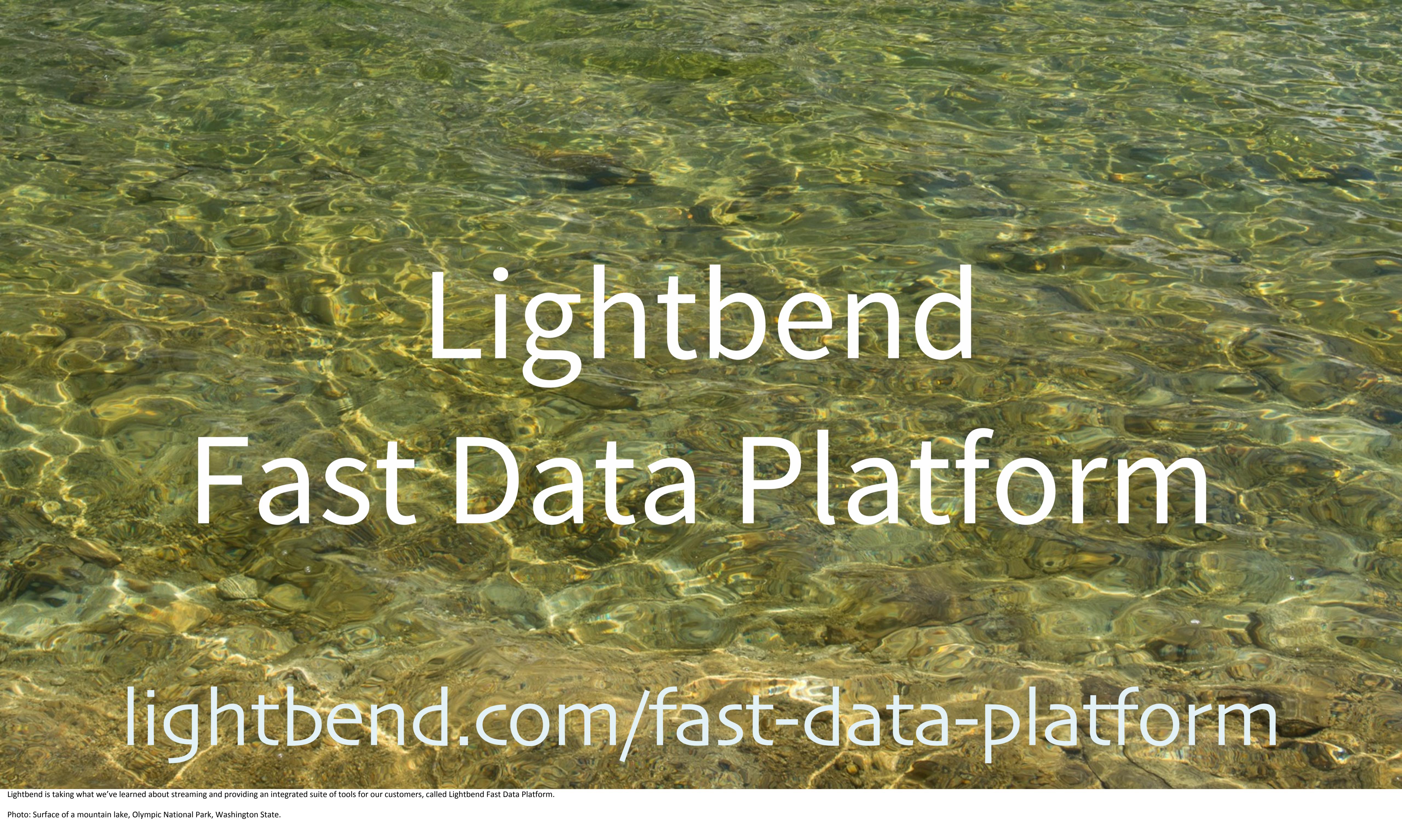
Some Overlap: Concerns, Architecture

The Present?



Microservices
& *Fast* Data

Much More Overlap



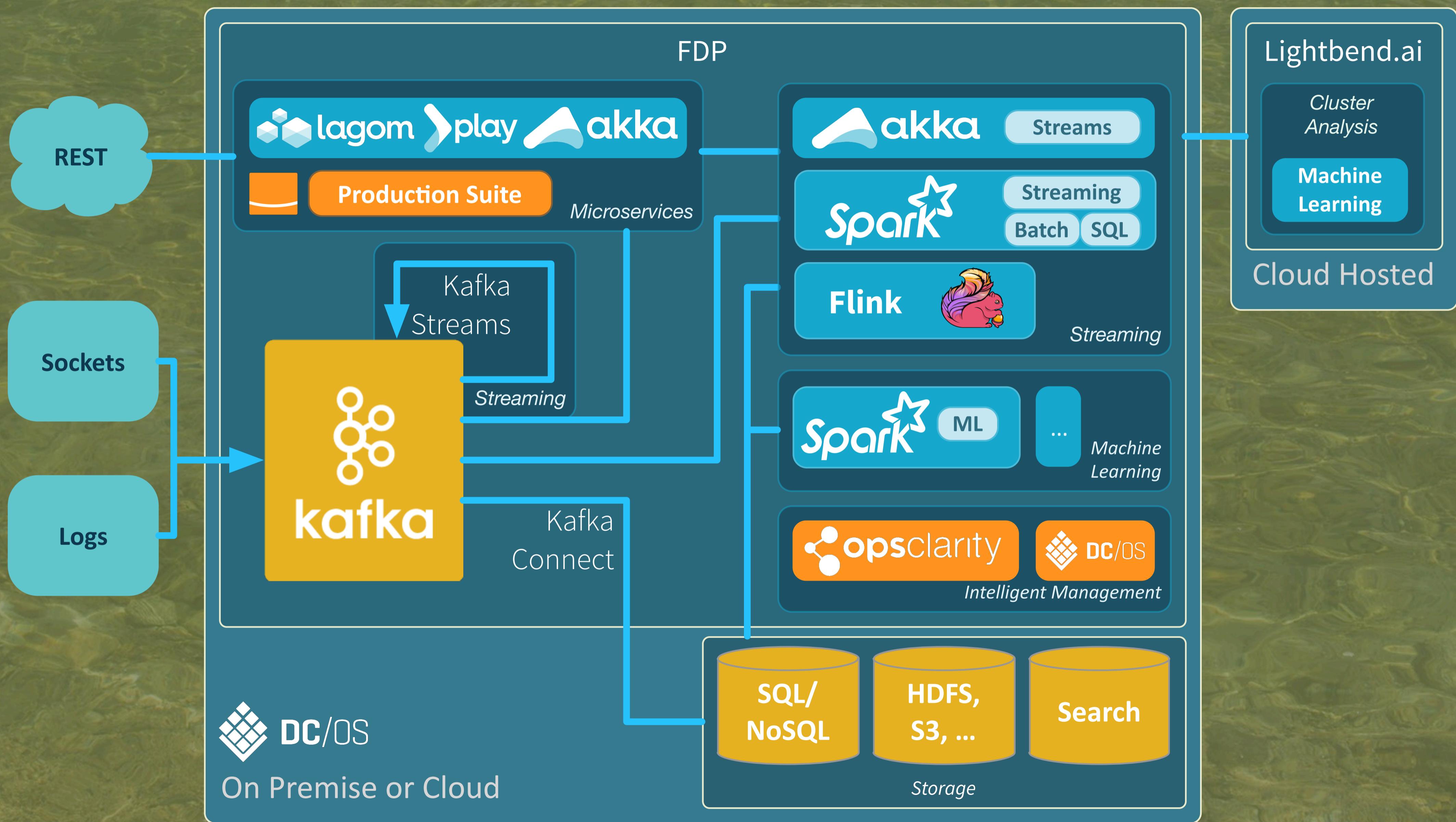
Lightbend Fast Data Platform

lightbend.com/fast-data-platform

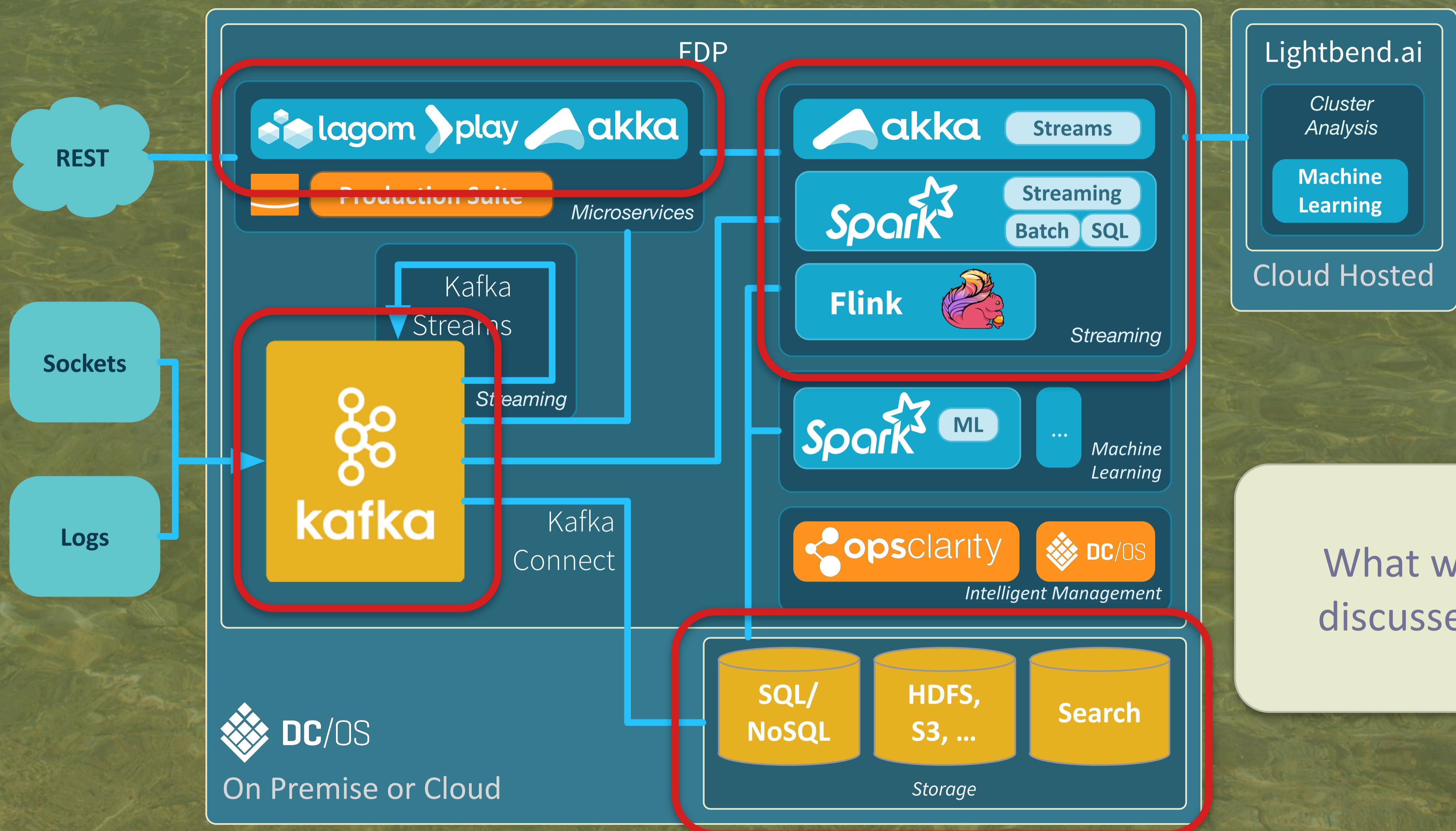
Lightbend is taking what we've learned about streaming and providing an integrated suite of tools for our customers, called Lightbend Fast Data Platform.

Photo: Surface of a mountain lake, Olympic National Park, Washington State.

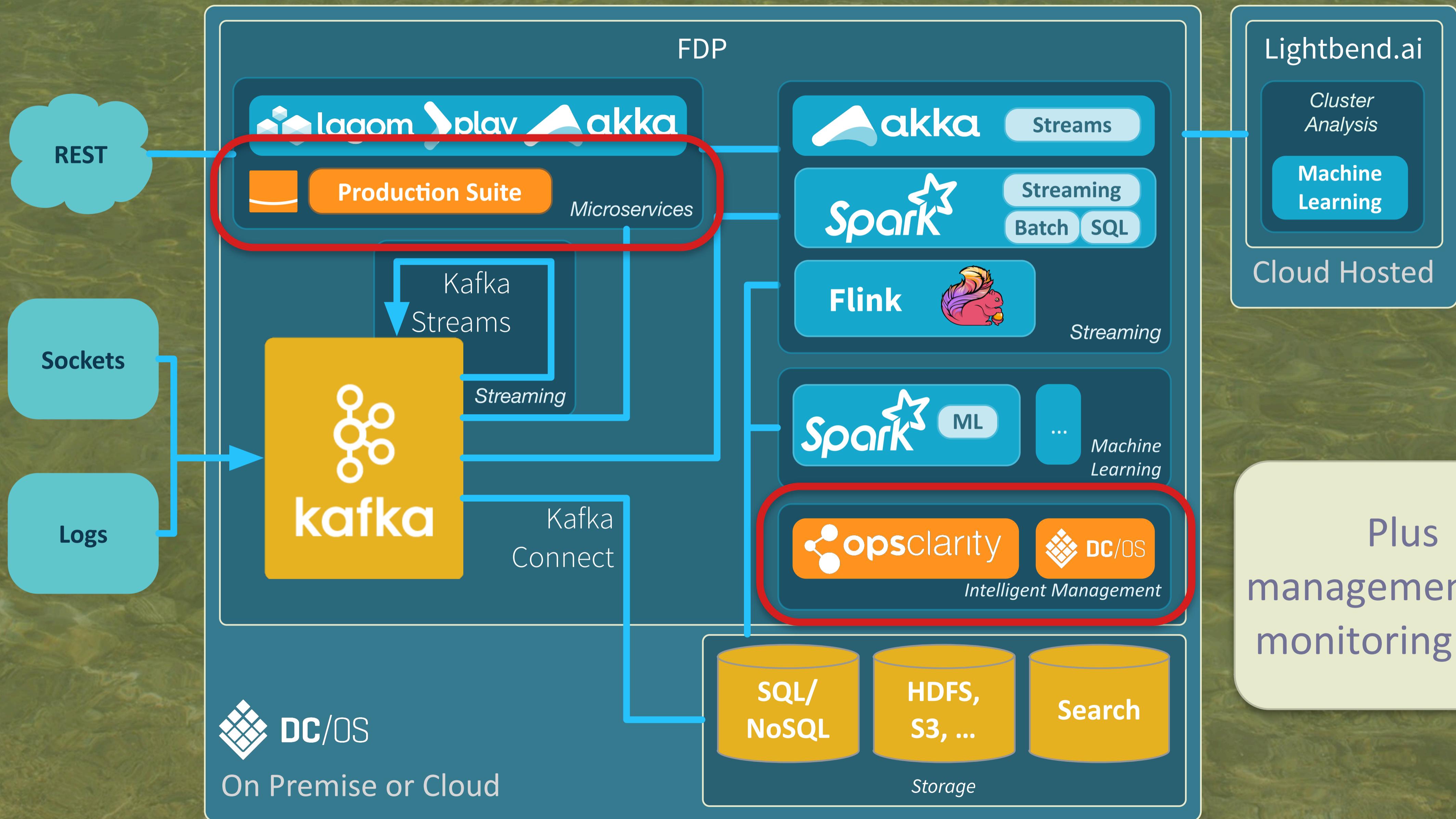
Lightbend Fast Data Platform V1.0



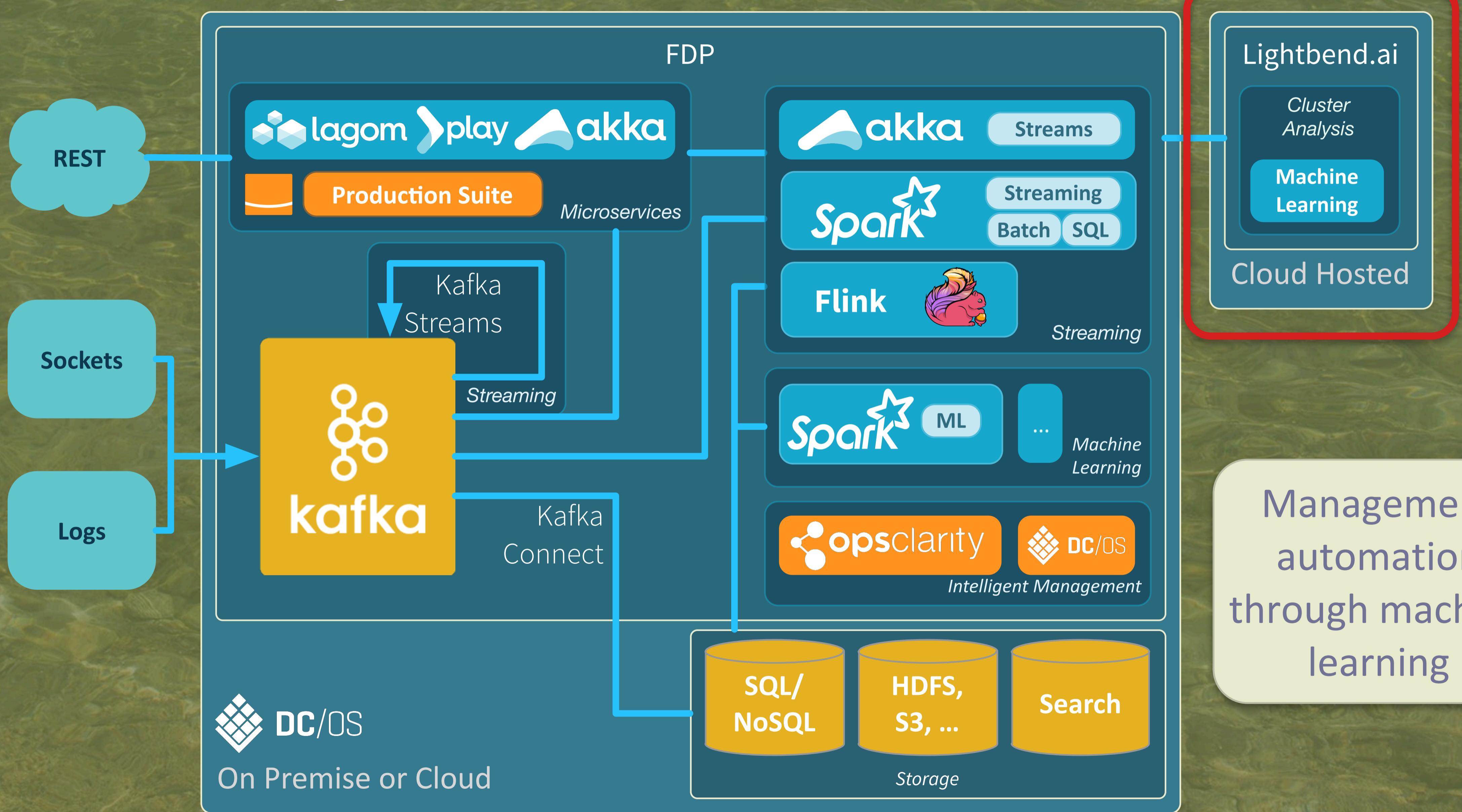
Lightbend Fast Data Platform V1.0



Lightbend Fast Data Platform V1.0



Lightbend Fast Data Platform V1.0



Free as in

bit.ly/fastdata-ORbook

O'REILLY®

Fast Data Architectures for Streaming Applications

Getting Answers Now from
Data Sets that Never End



Dean Wampler

Compliments of
Lightbend



Lightbend

Thank you!

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

lightbend.com/fast-data-platform