Lightbend

Stream All the Things!

Architectures for Data that Never Ends

Dean Wampler, Ph.D.
dean@lightbend.com
@deanwampler

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

Photograph: Iceberg Lake, Glacier National Park, Montana, USA.

Free as in

lightbend.com/fast-data-platform

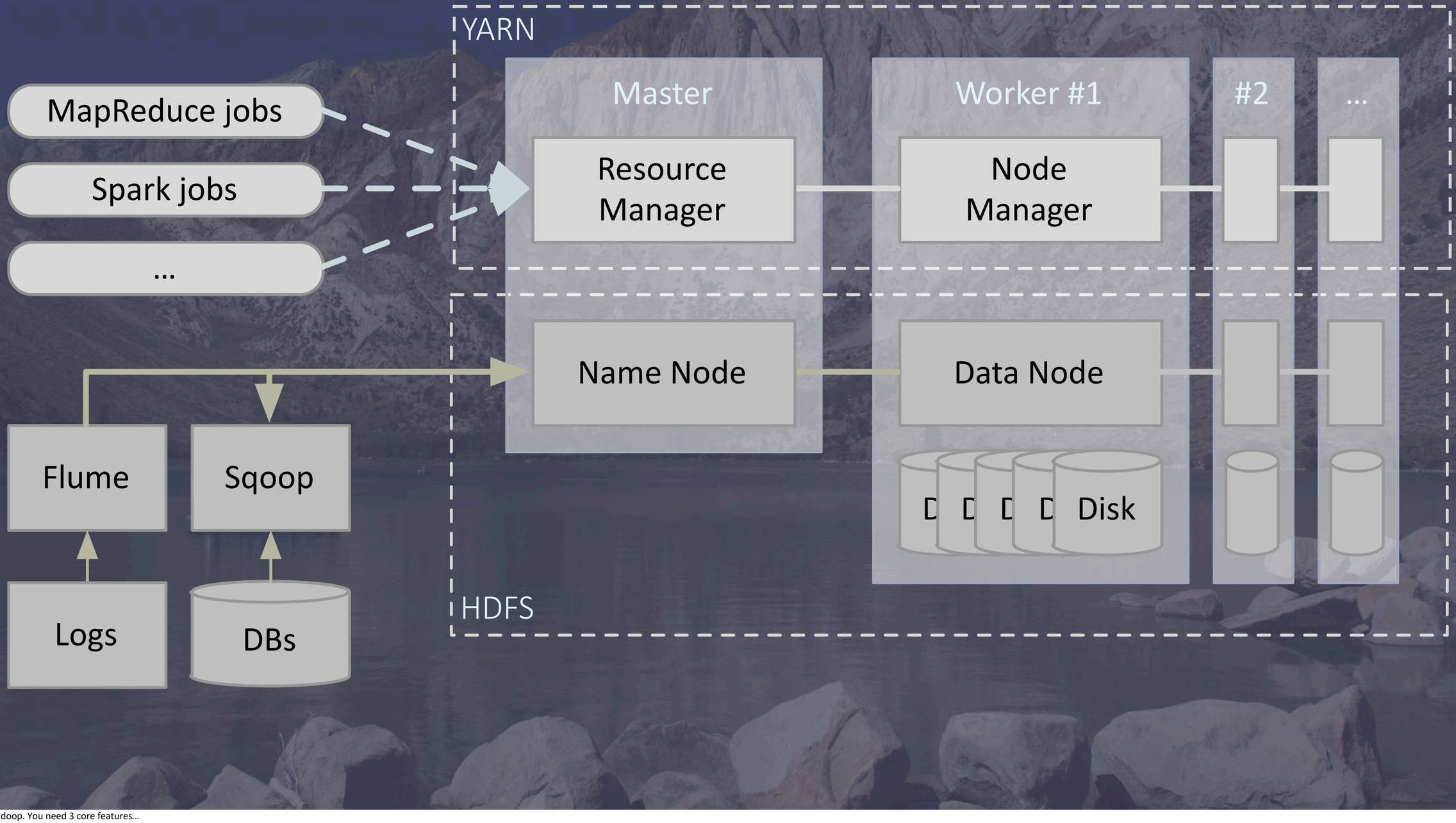
My book, published last fall that describes the points in this talk in greater depth. I've refined the talk a bit since this was published.

Streaming in Context…

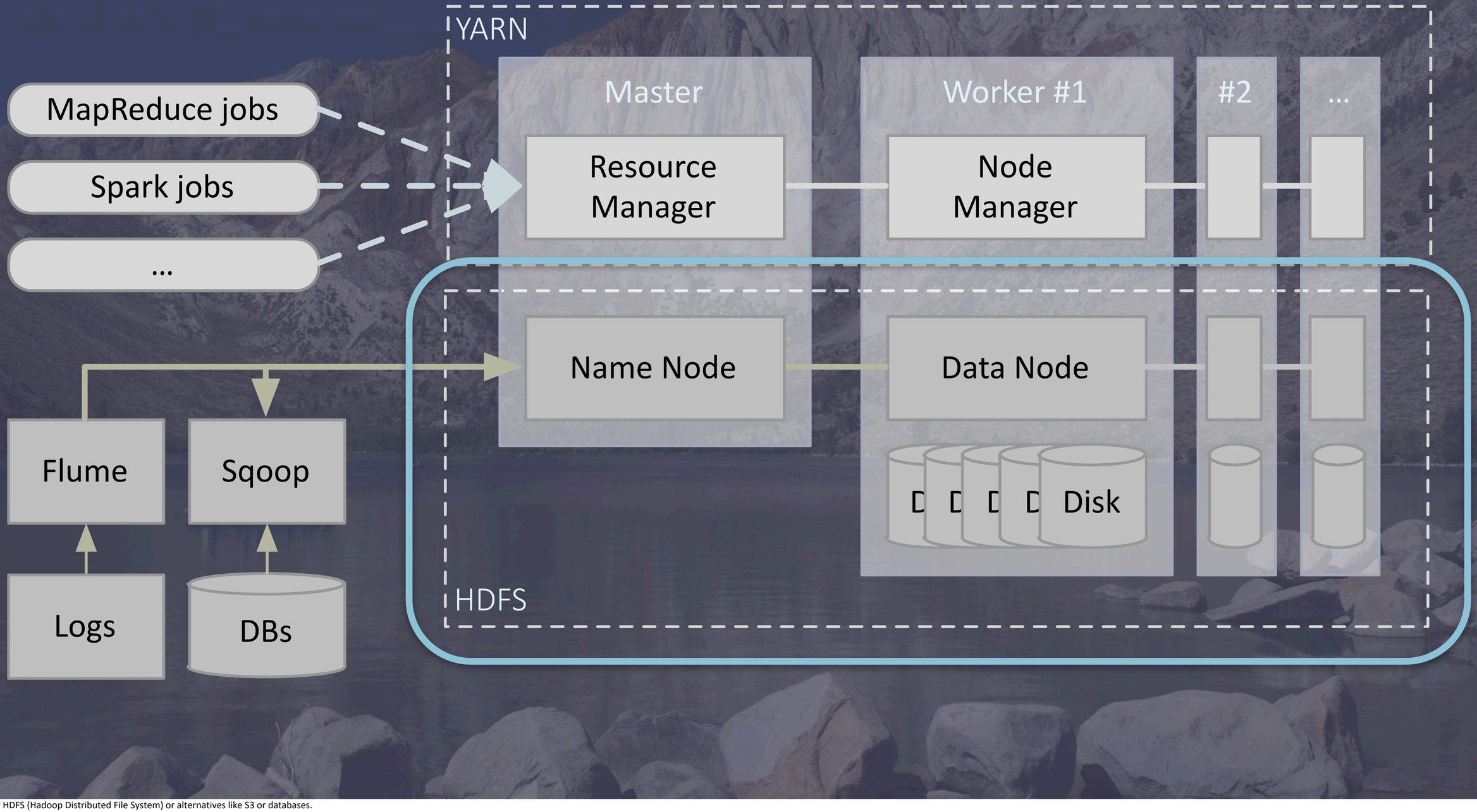
First, let’s put streaming in context; why are we interested in Streaming Data, a.k.a. Fast Data? What did we use before?

Hadoop: Classic Batch Architecture

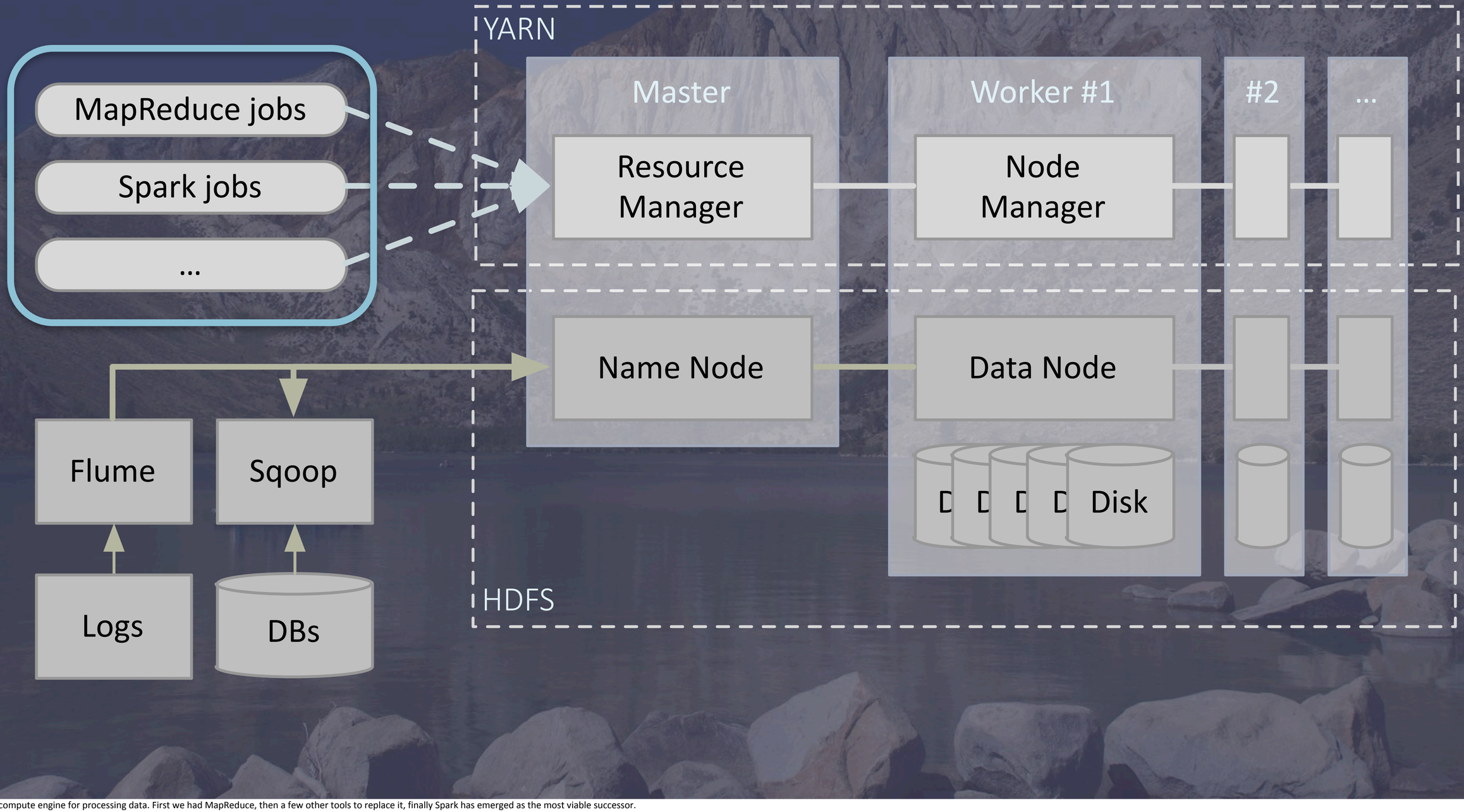
A data lake??
Photo: Convict Lake in the Sierras, California



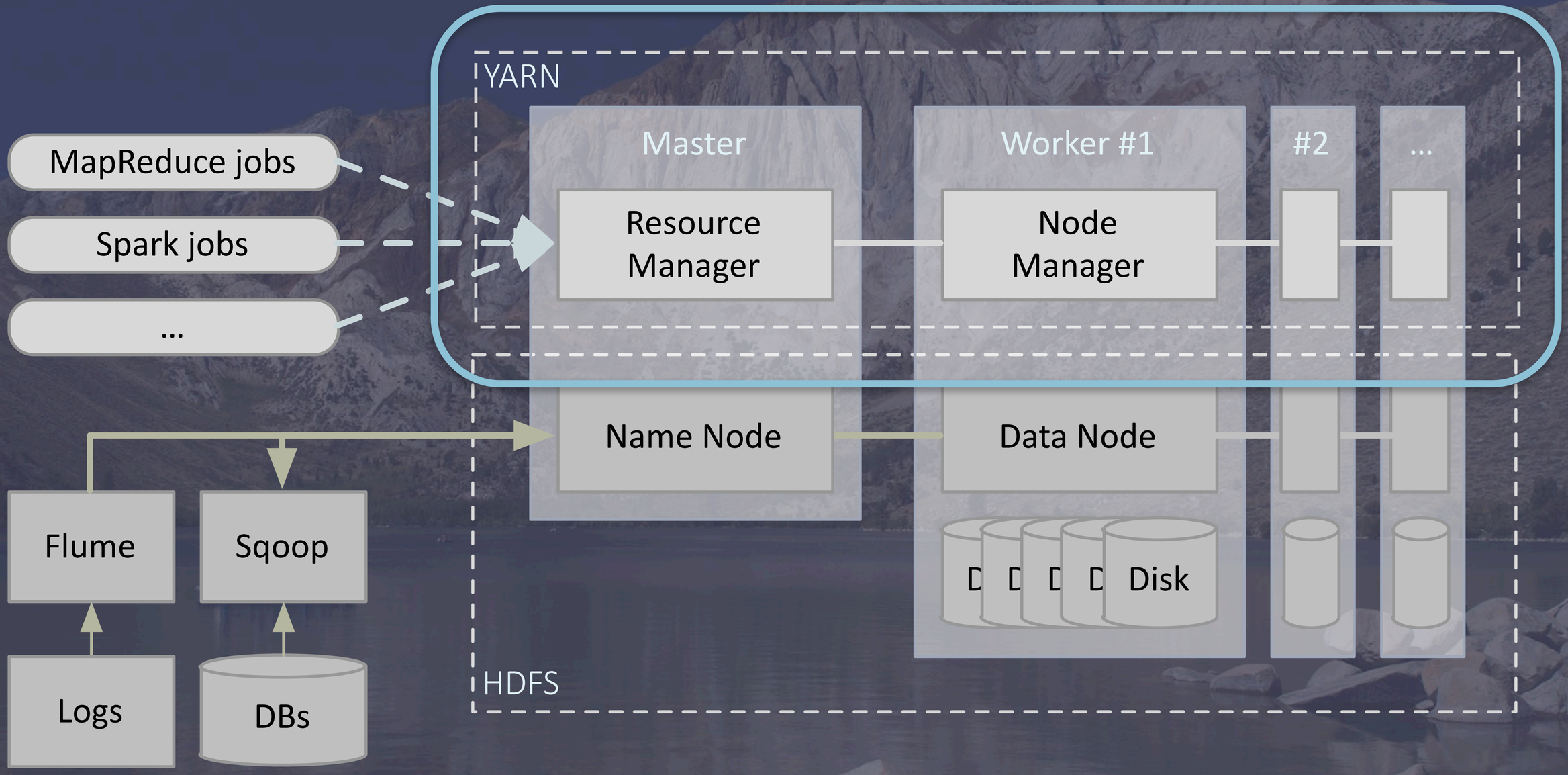
Schematic view of Hadoop. You need 3 core features…



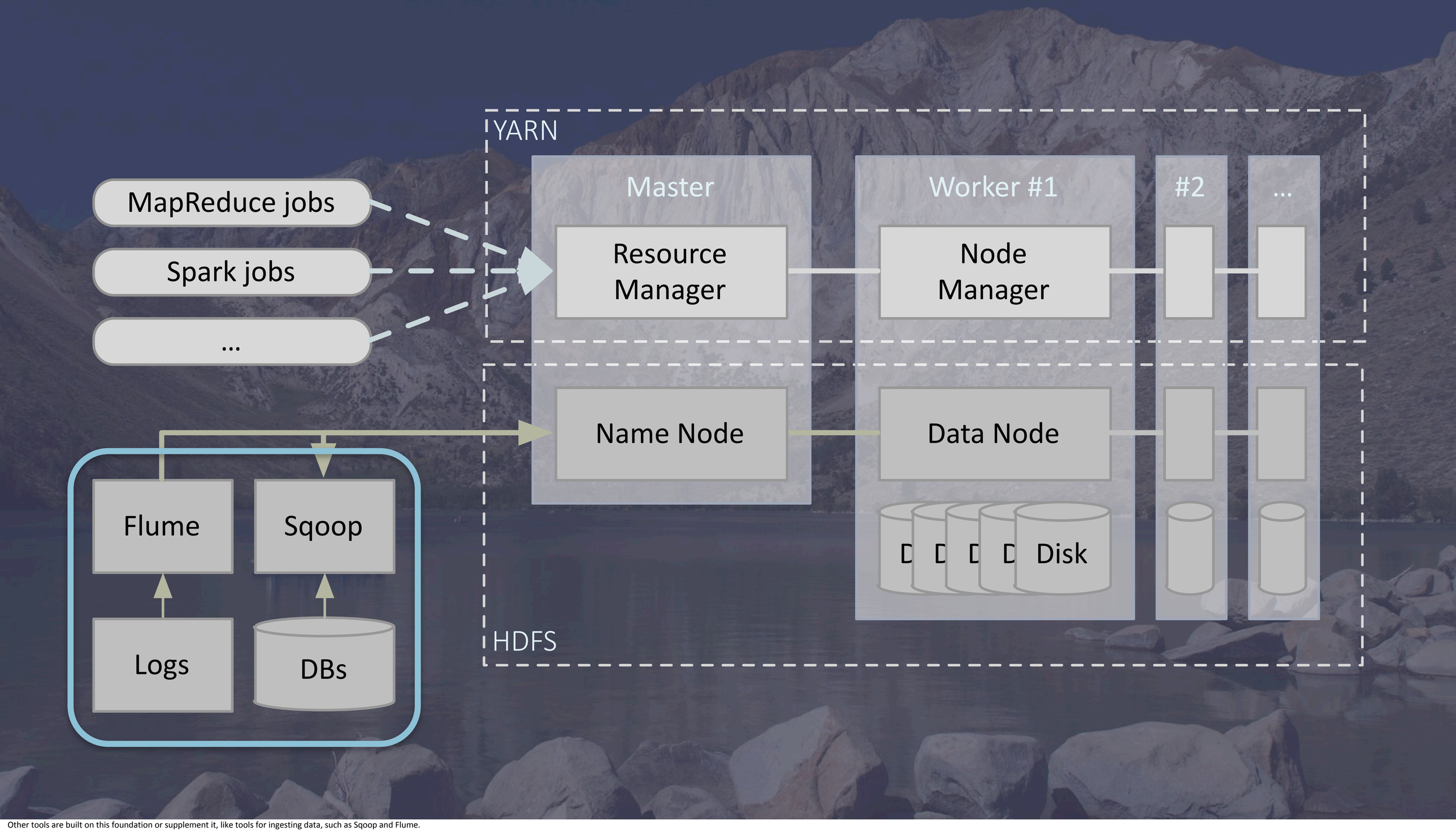
1. Storage tier, either HDFS (Hadoop Distributed File System) or alternatives like S3 or databases.



2. You need a compute engine for processing data. First we had MapReduce, then a few other tools to replace it, finally Spark has emerged as the most viable successor.

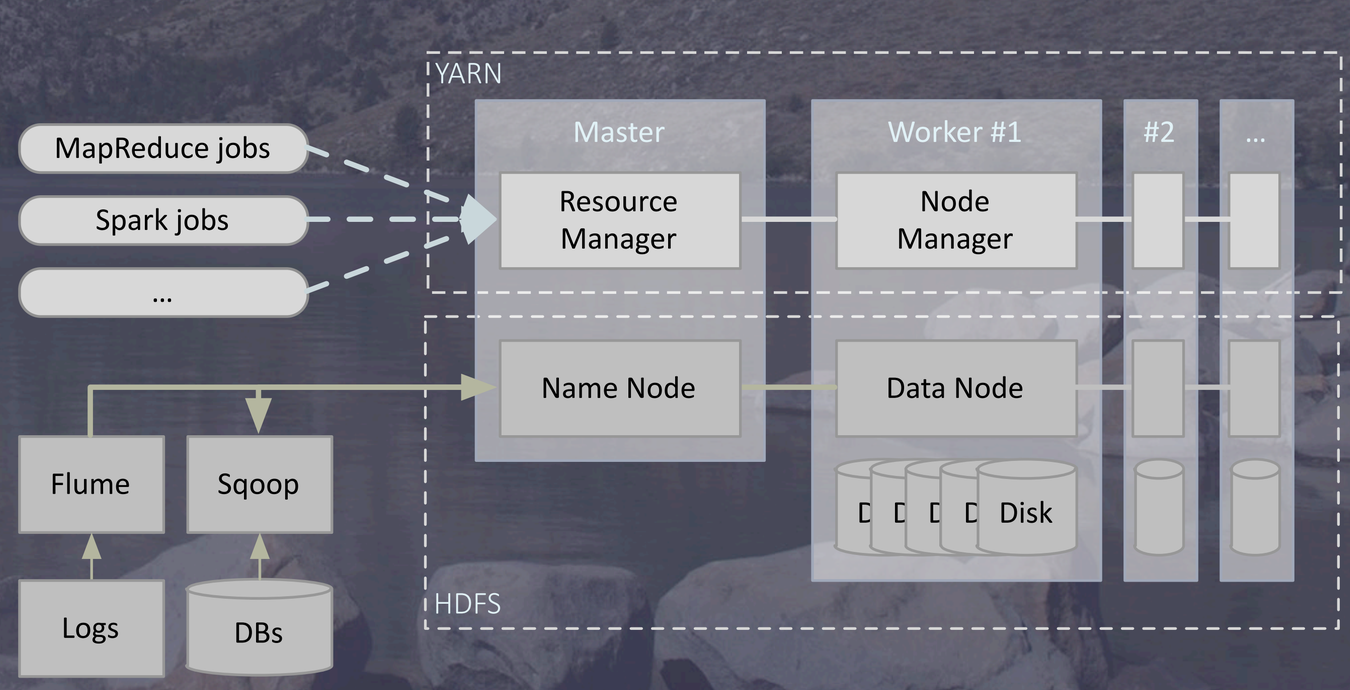


3. Finally, you need a manager of resources, scheduler of jobs and tasks, etc.



Other tools are built on this foundation or supplement it, like tools for ingesting data, such as Sqoop and Flume.

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

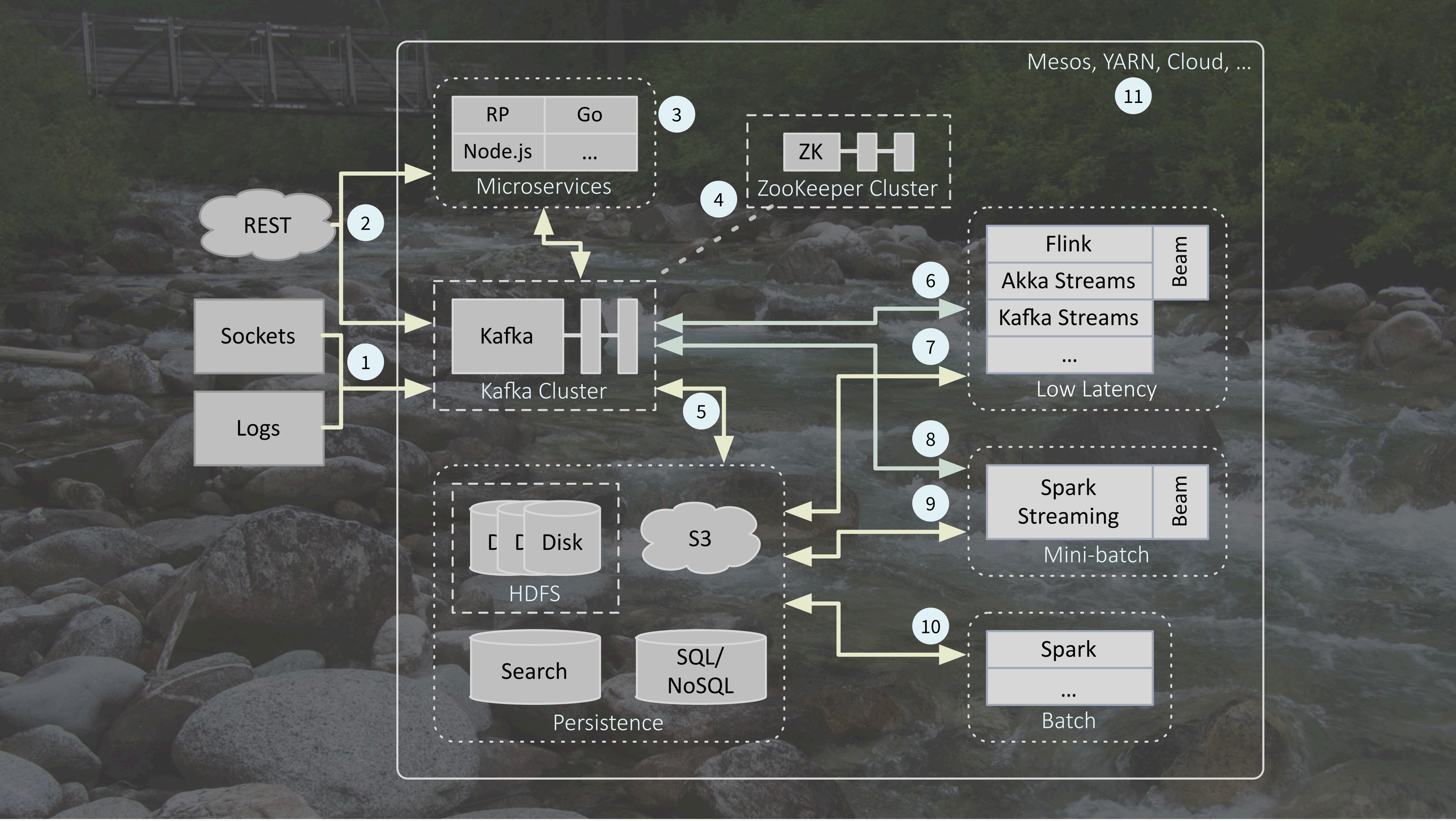


DW replacement is a common use of Hadoop, because it nicely supports SQL analysis over large data sets, using Hive, Impala, Kudu, etc.

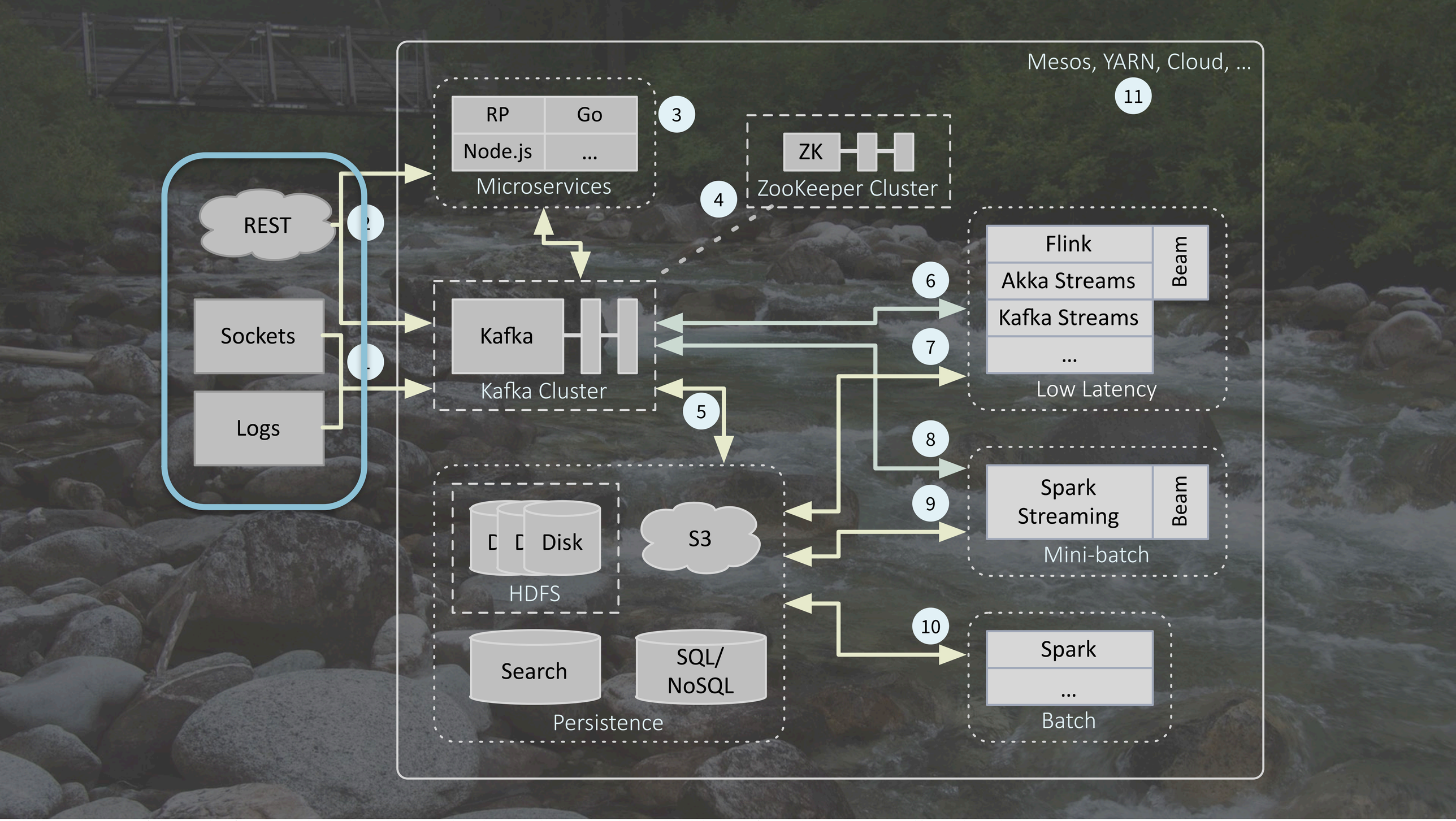
New Streaming, “Fast Data” Architecture

But it will also support batch!!

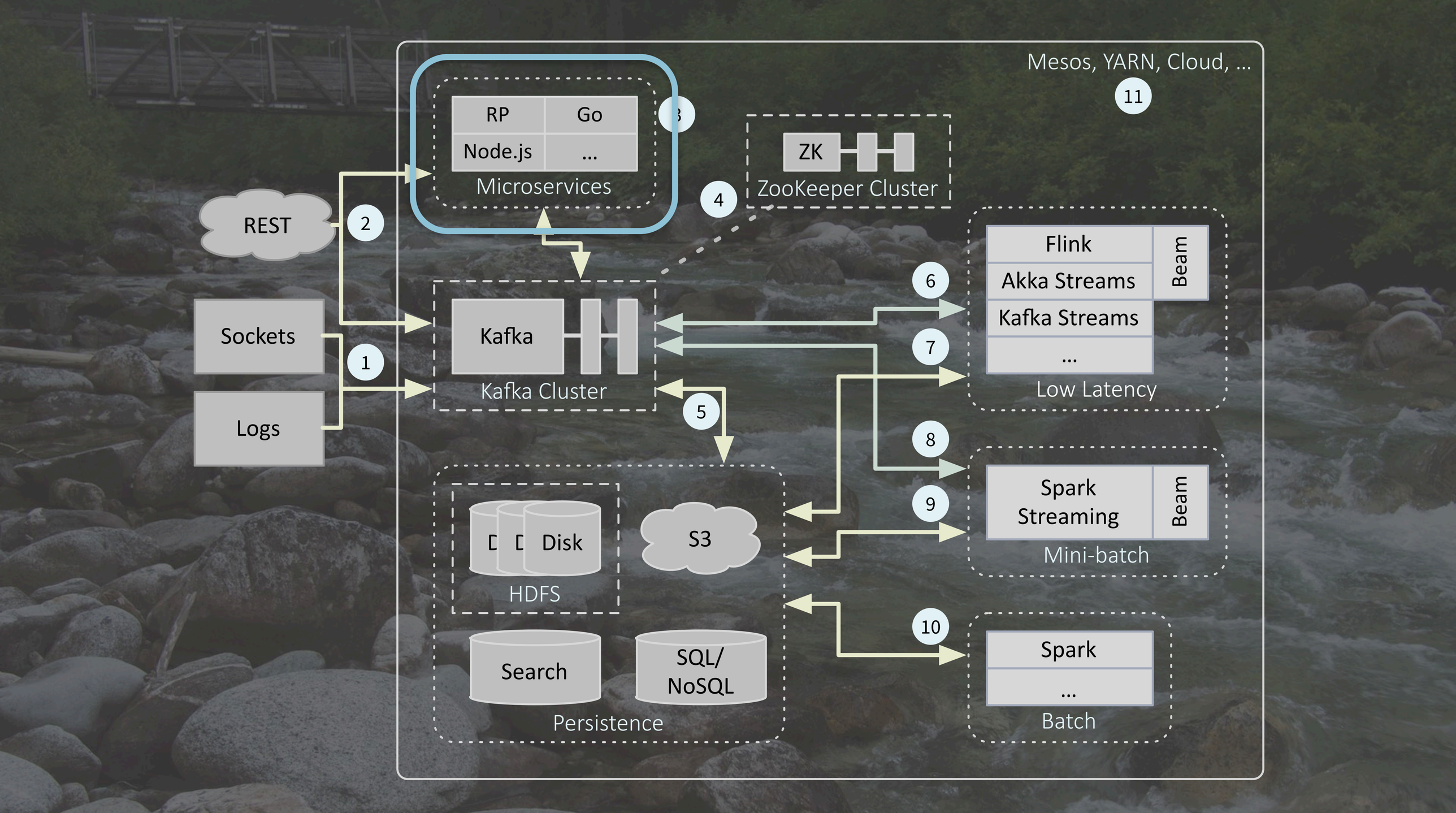
Photo: Stream in North Cascades National Park, Washington State.



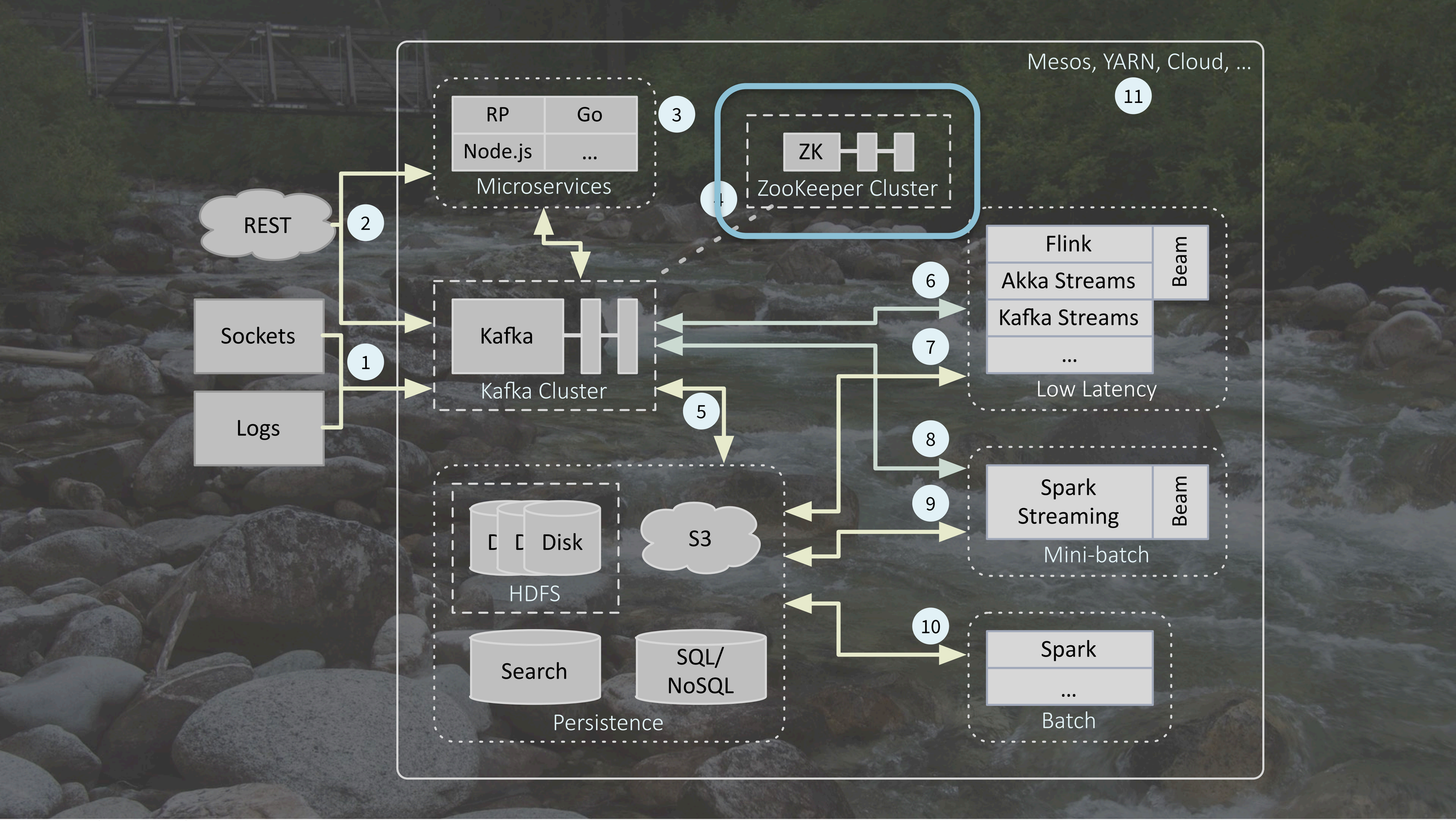
Numbering is in the report, so it's easier for the text to refer back to the figure.



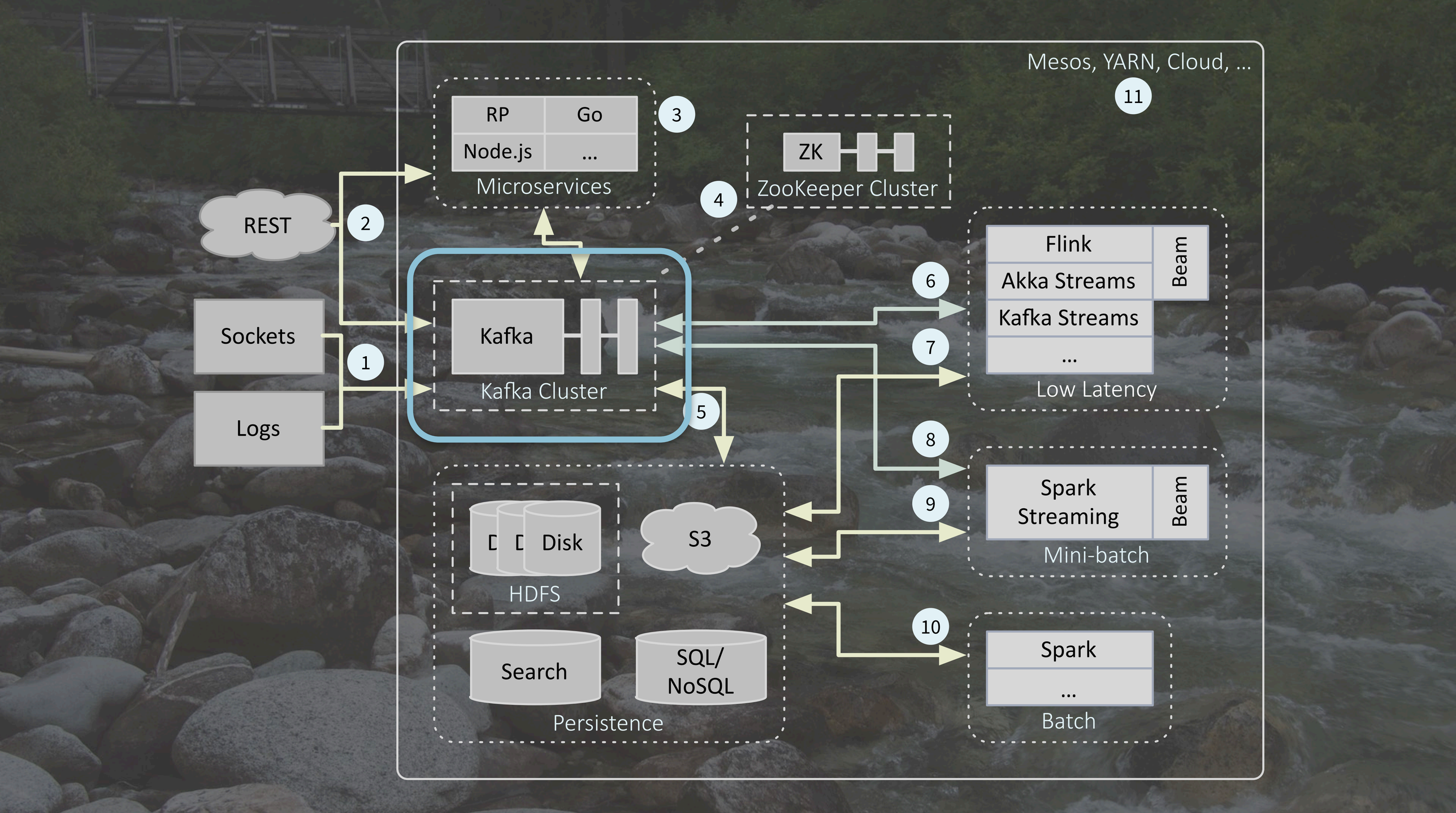
1 & 2. Data sources include streams of data over sockets and from logs. You might also ingest through REST channels, but unless they are async, the overhead will be too high, so REST might be used only for communicating with the microservices (3) you write to complete your environment.



3. A real fast data environment is similar to more general services, you'll have the "heavy hitters" like Spark, Kafka, etc., but you'll also need to write support microservices to complete the environment.

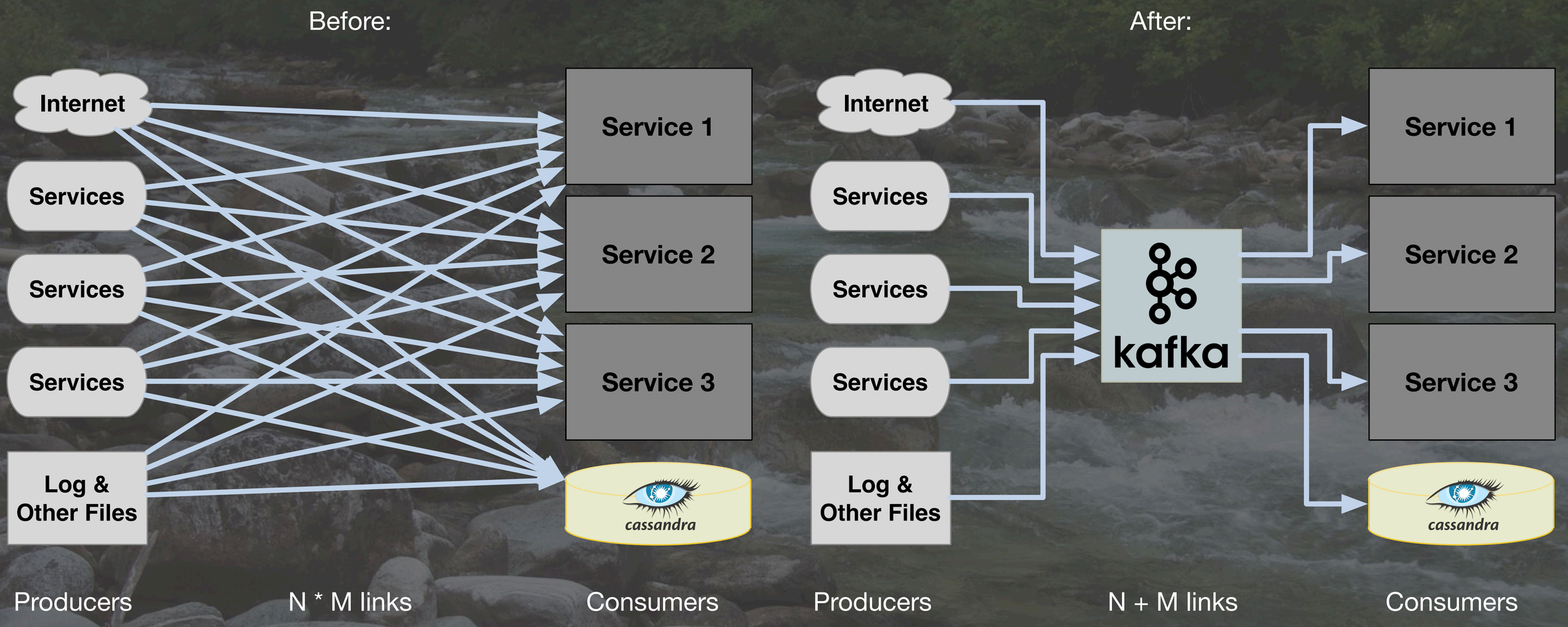


4. Some services, like Kafka, require ZooKeeper for managing shares state, such as leaders.

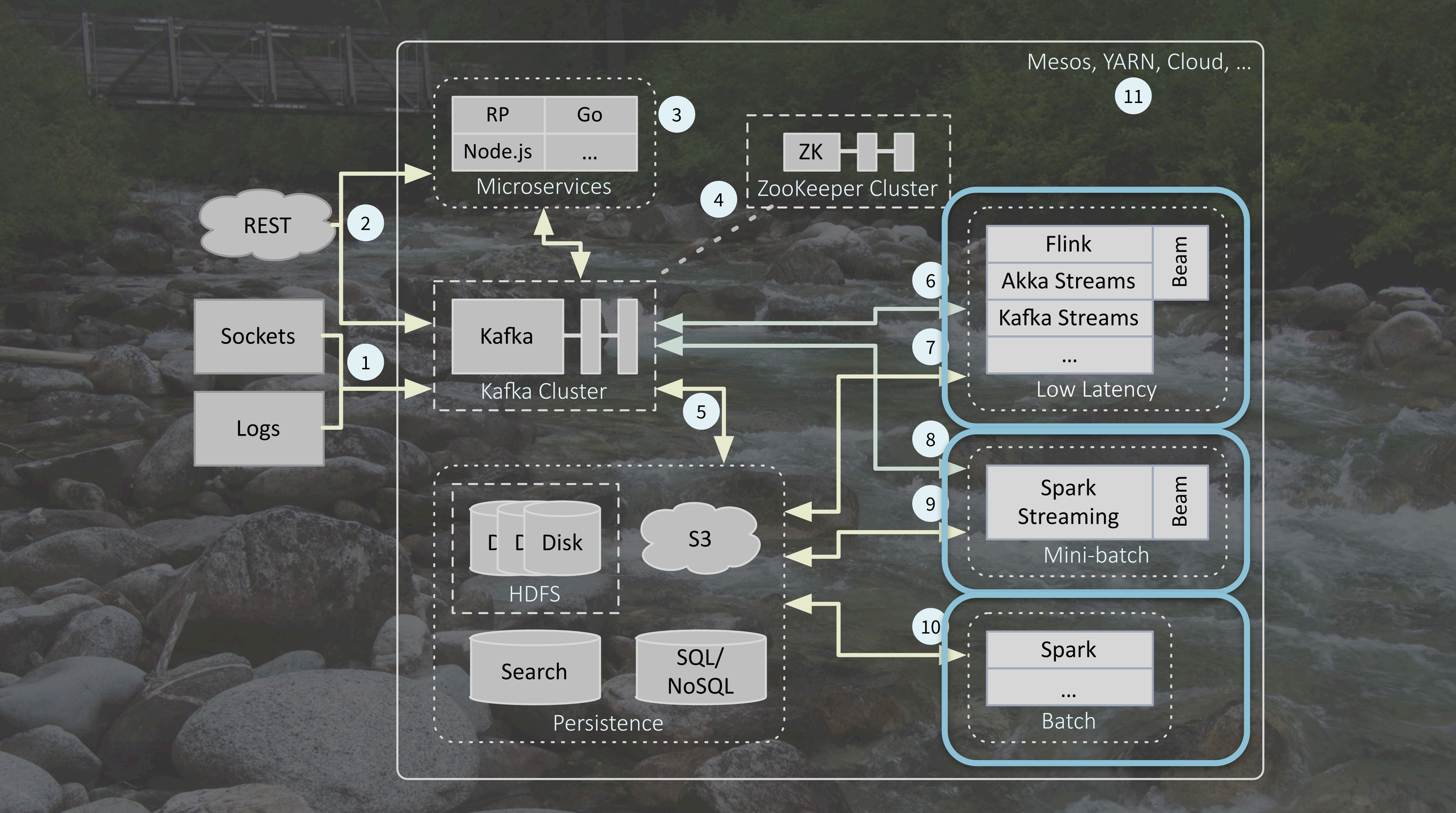


5. Kafka is the core of this architecture, the place where data is ingested in queues, organized into topics. Publishers are consumers are decoupled and N-to-M. Kafka has massive scalability and excellent resiliency and data durability. All services can communicate through each other using Kafka, too,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

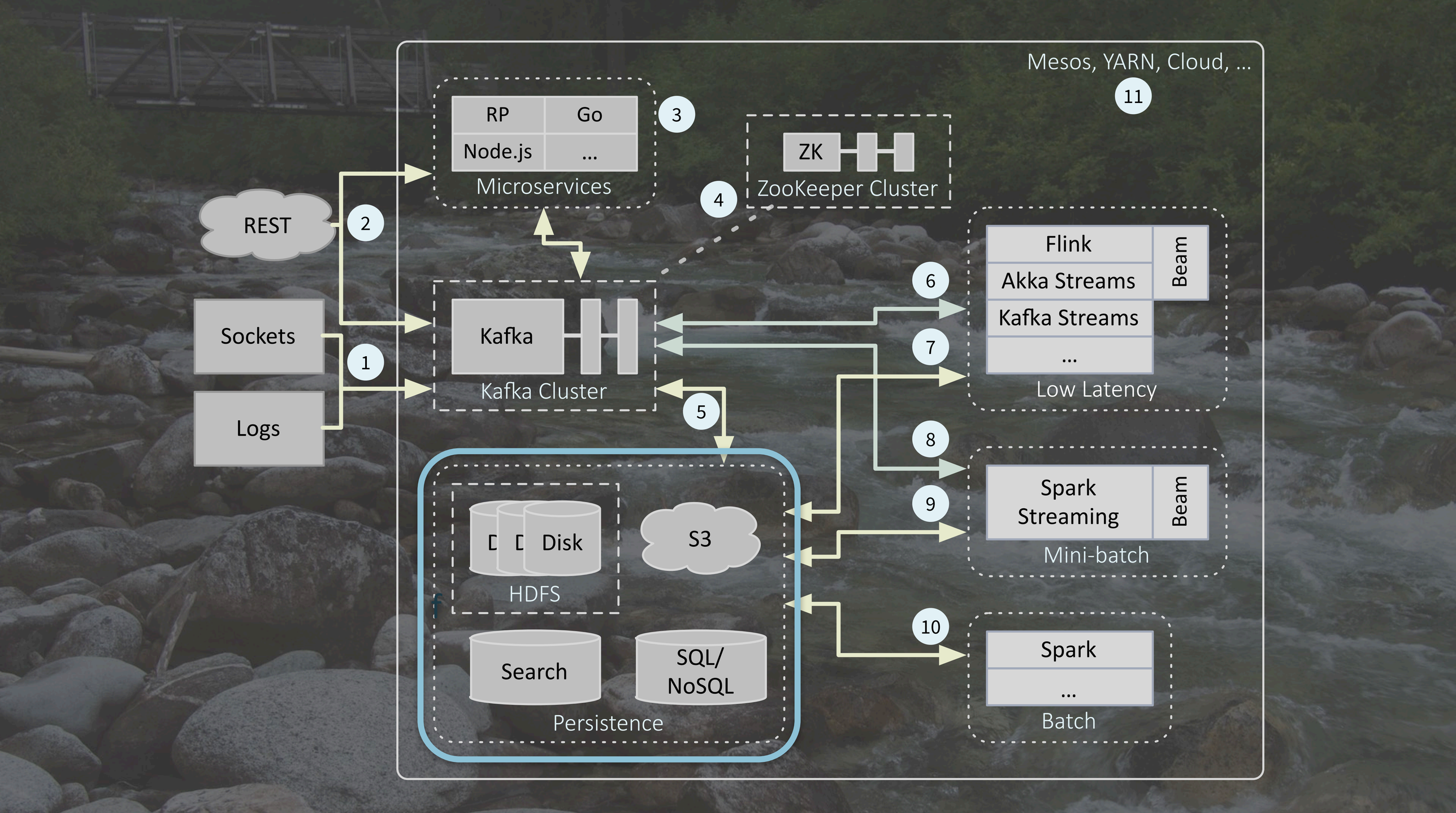
Using Kafka



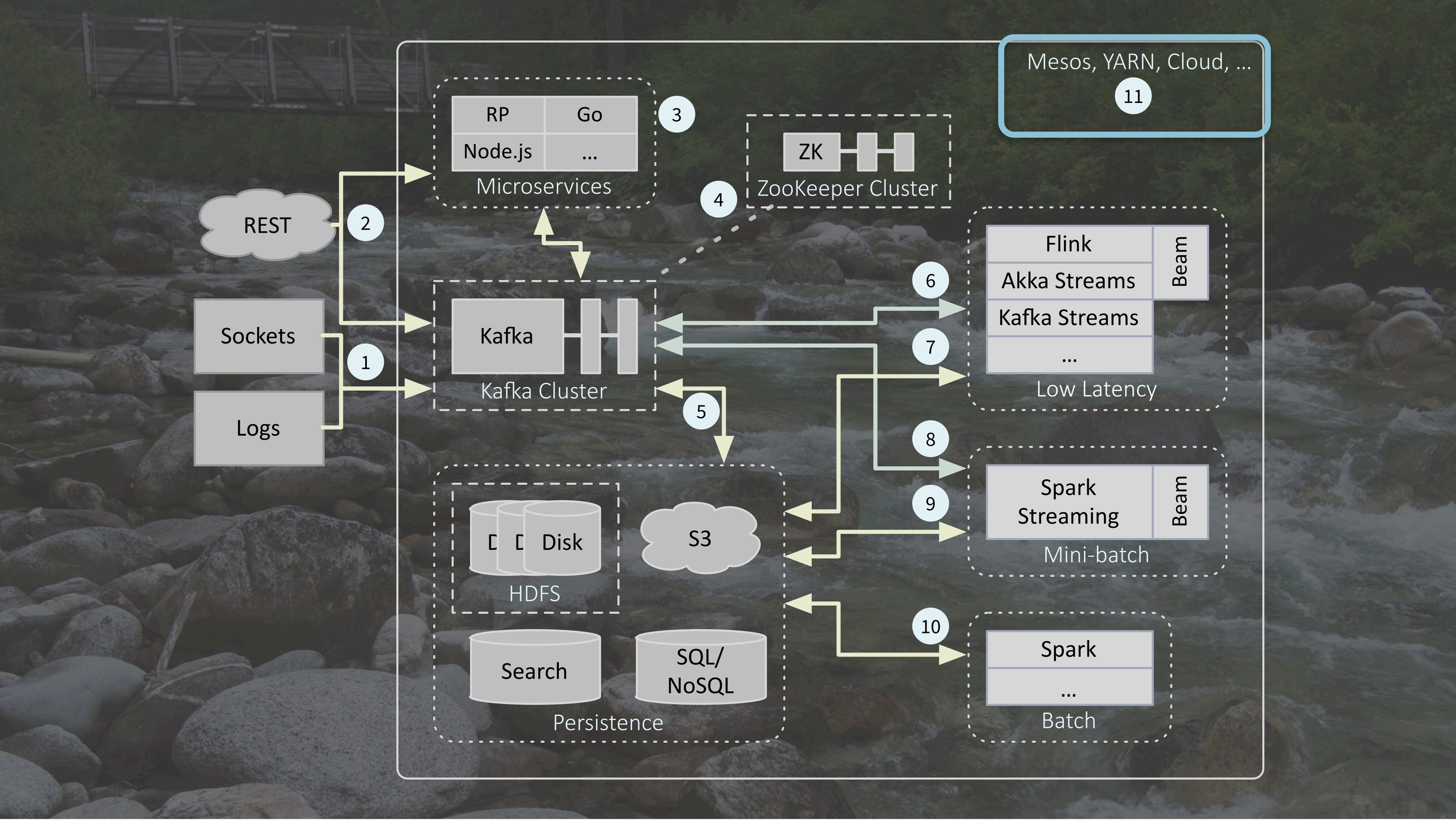
Kakfa can simplify your architecture, too. Not only is this cleaner, it's more robust, as the point-to-point connections are more fragile, more likely to result in data and service loss if one point goes down, where Kafka keeps the other point "healthy" while the failed point is restored. Kafka also 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.



6, 8, and 10. Many pure streaming, mini-batch streaming, and batch engines are vying for your attention. We'll focus into this area after this overview.



5, 7, 9, and 10. Data can also be read and written between these compute engines and storage, not just Kafka. For 5, Kafka Connect is used.



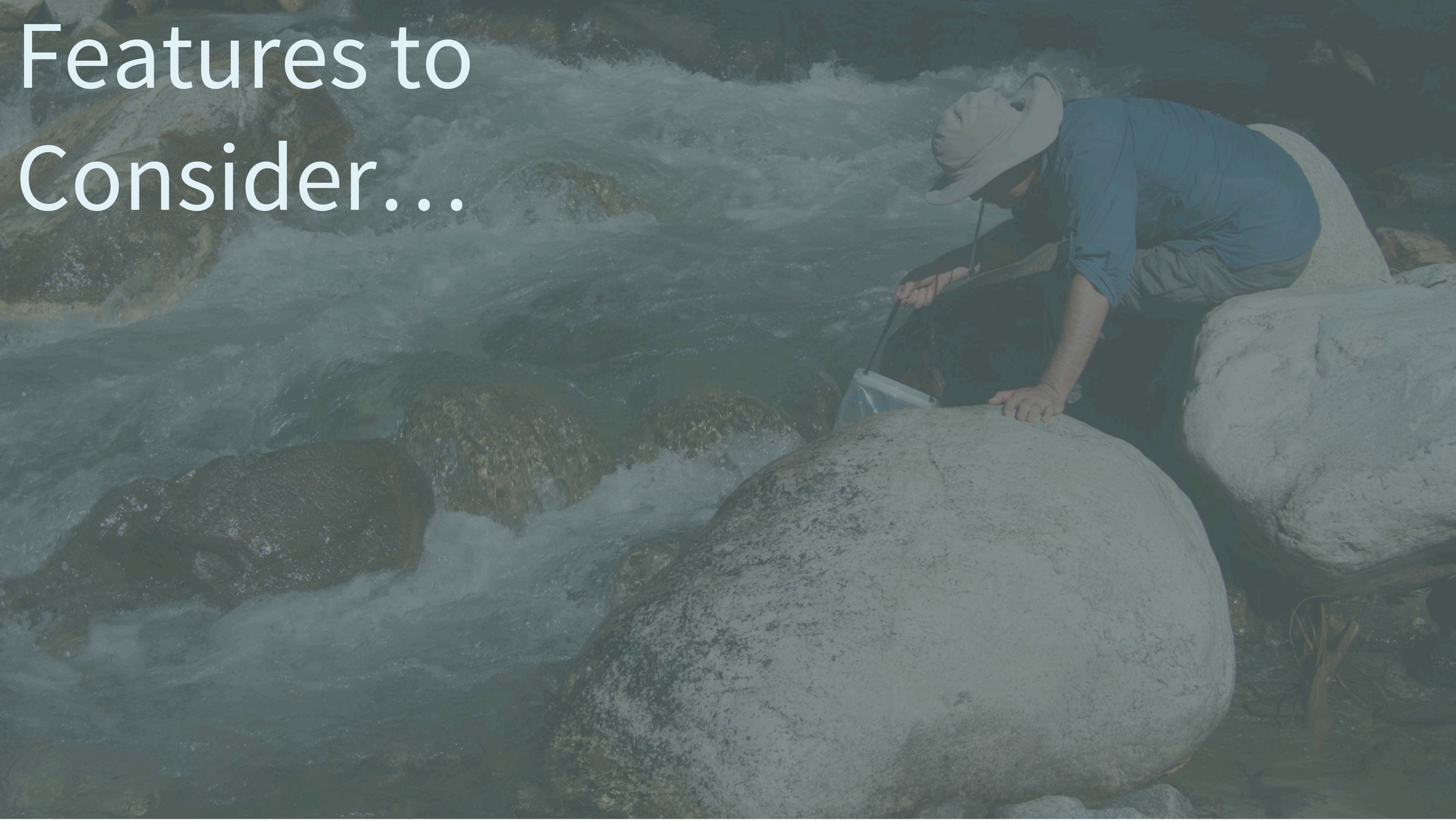
11. You can run this architecture on Mesos, YARN (Hadoop), on premise or in the cloud. (At Lightbend, we think Mesos is the best choice.)

Streaming Engines

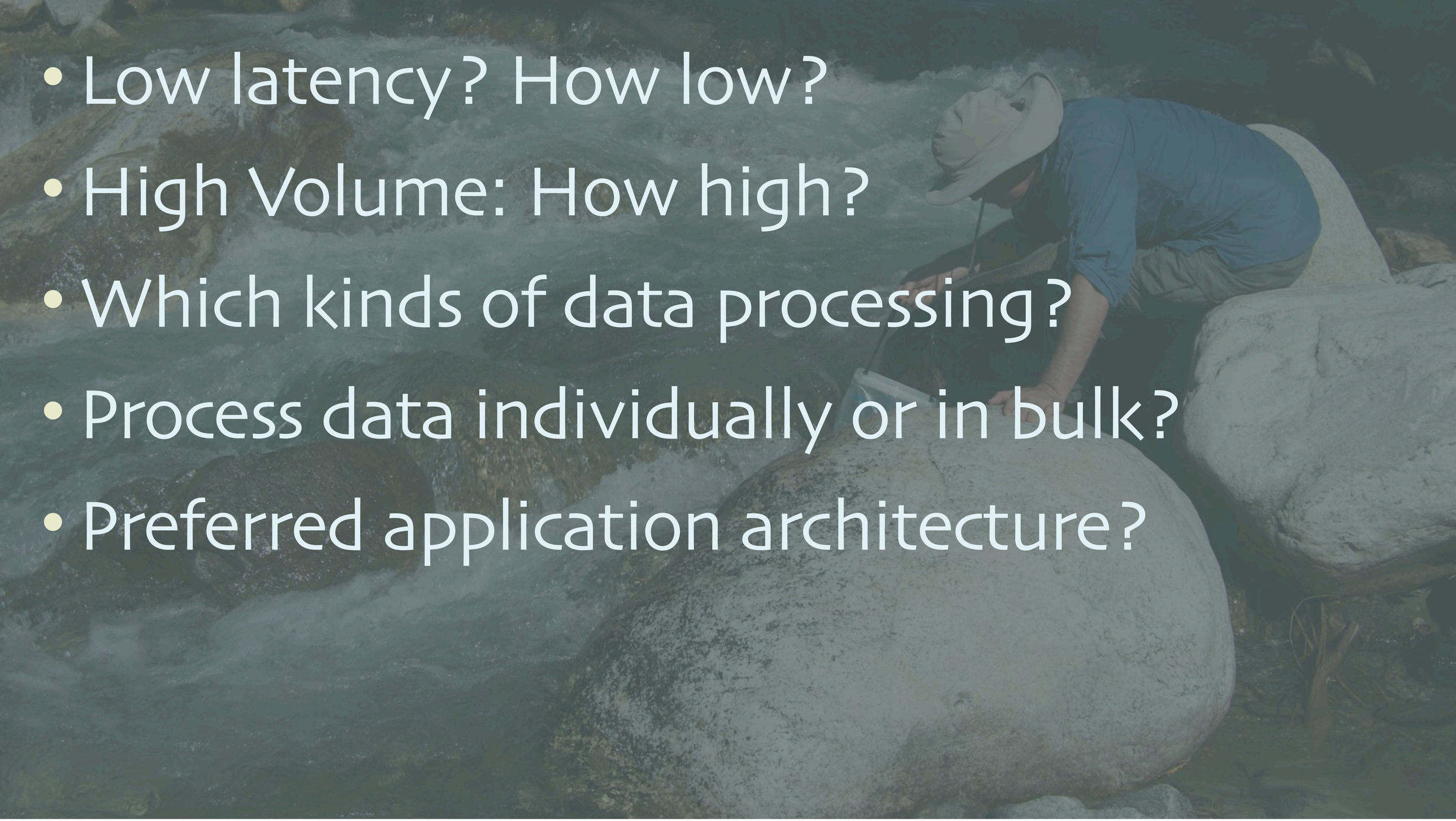
Let’s dive into a representative set of available streaming engines.

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

Features to Consider…



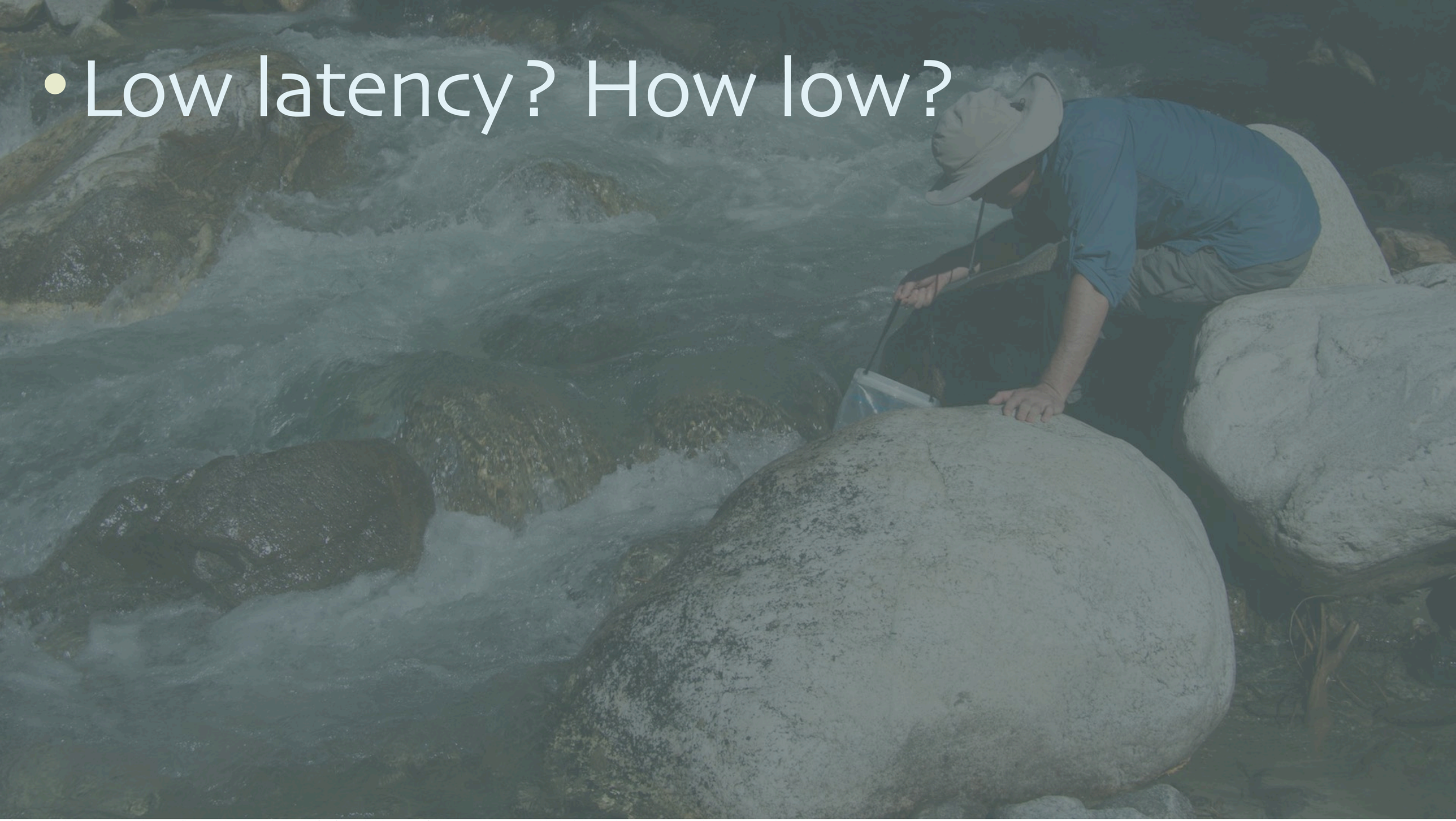
When deciding what to use, there are several dimensions or features to consider.



- Low latency? How low?
- High Volume: How high?
- Which kinds of data processing?
- Process data individually or in bulk?
- Preferred application architecture?

If you have tight latency requirements, you can’t use a mini-batch or batch engine. If you’re constraints are more flexible, you can do more sophisticated things, like training ML models, writing to databases, etc.

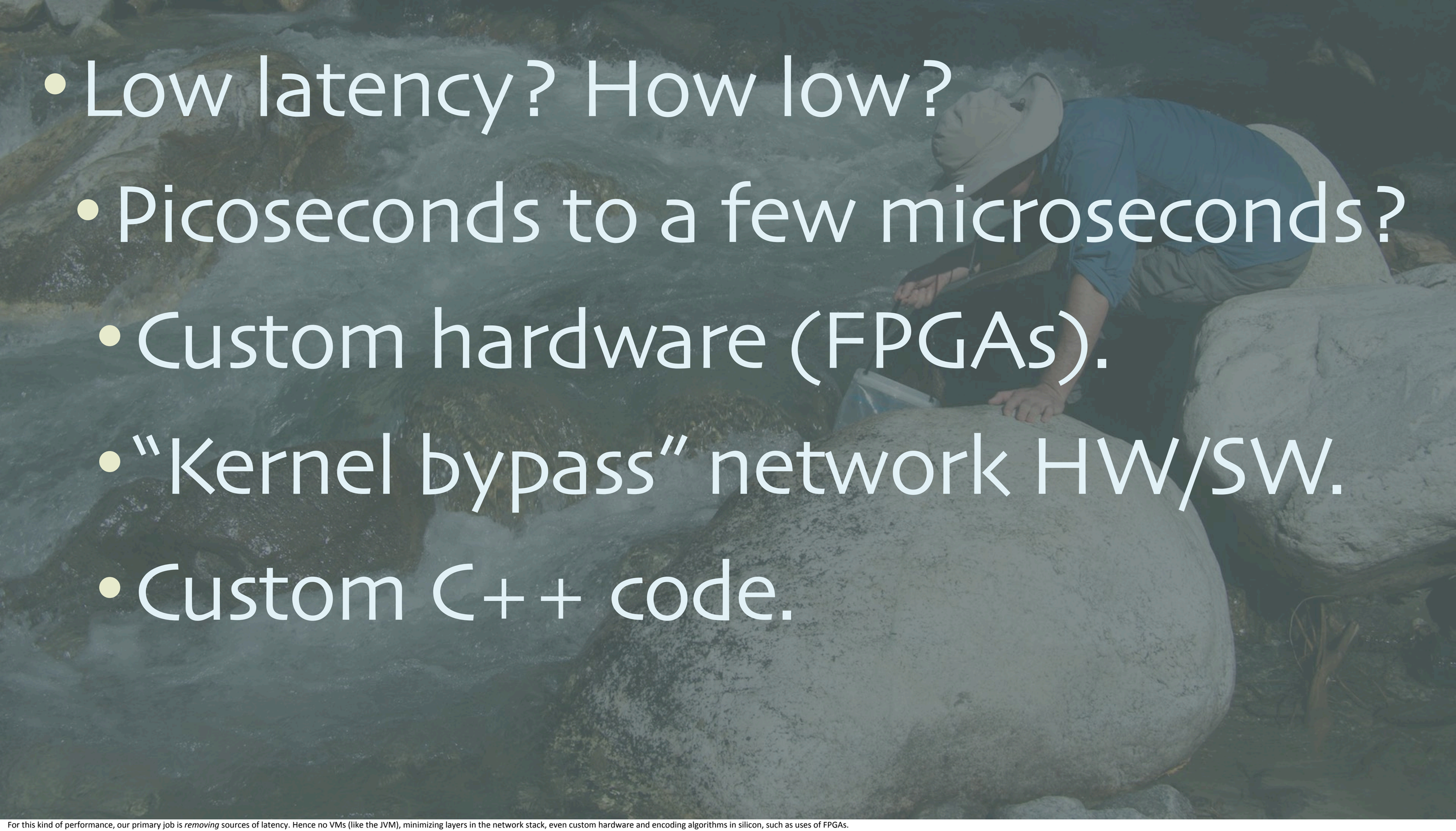
- Low latency? How low?



If you have tight latency requirements, you can’t use a mini-batch or batch engine. If you’re constraints are more flexible, you can do more sophisticated things, like training ML models, writing to databases, etc.

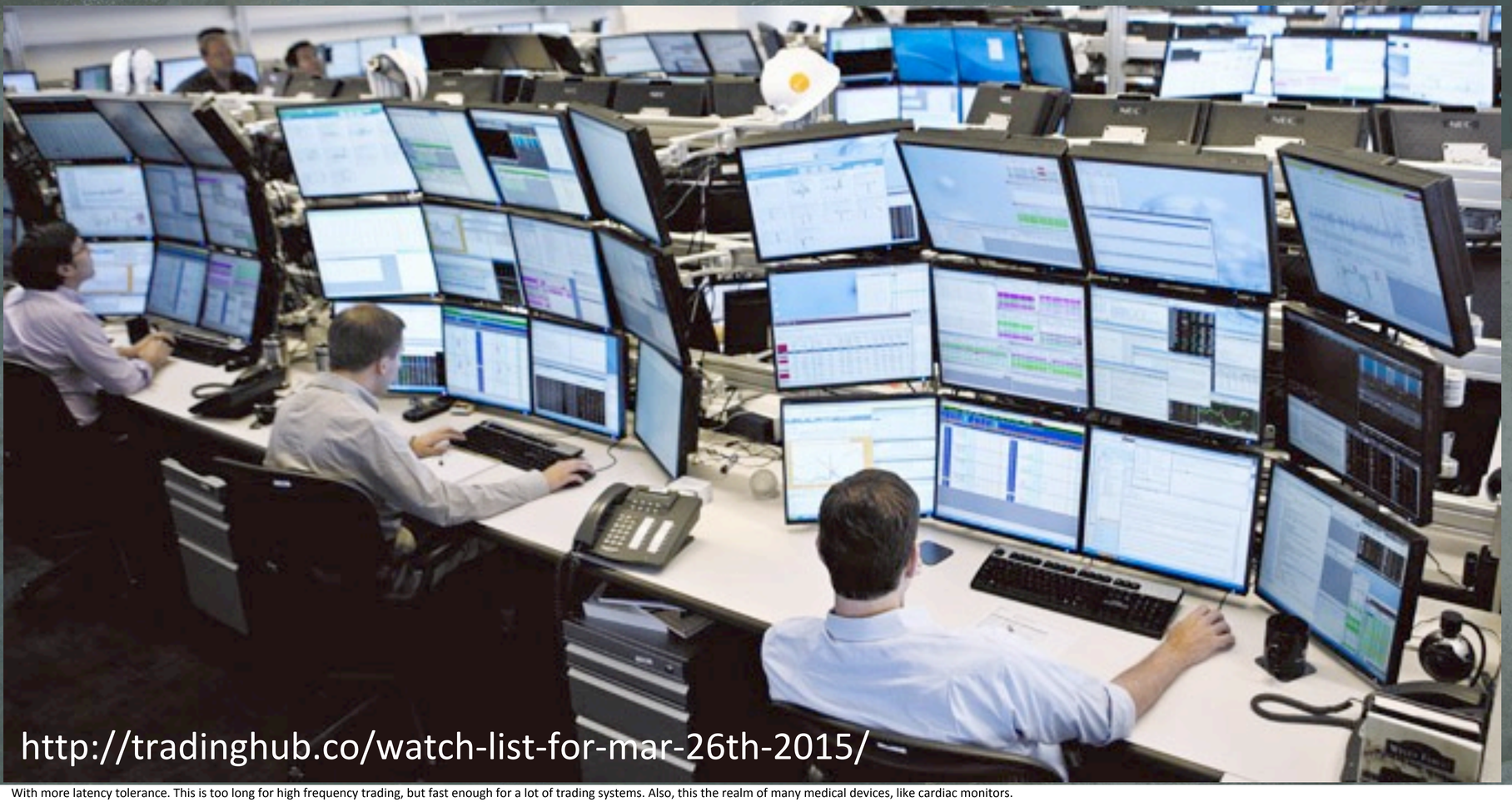
- Low latency? How low?
  - Picoseconds to a few microseconds?

Extremely low latency, high-frequency trading territory, requires custom everything. This is “true real time", like what’s required for rocket guidance systems, high frequency trading, etc.

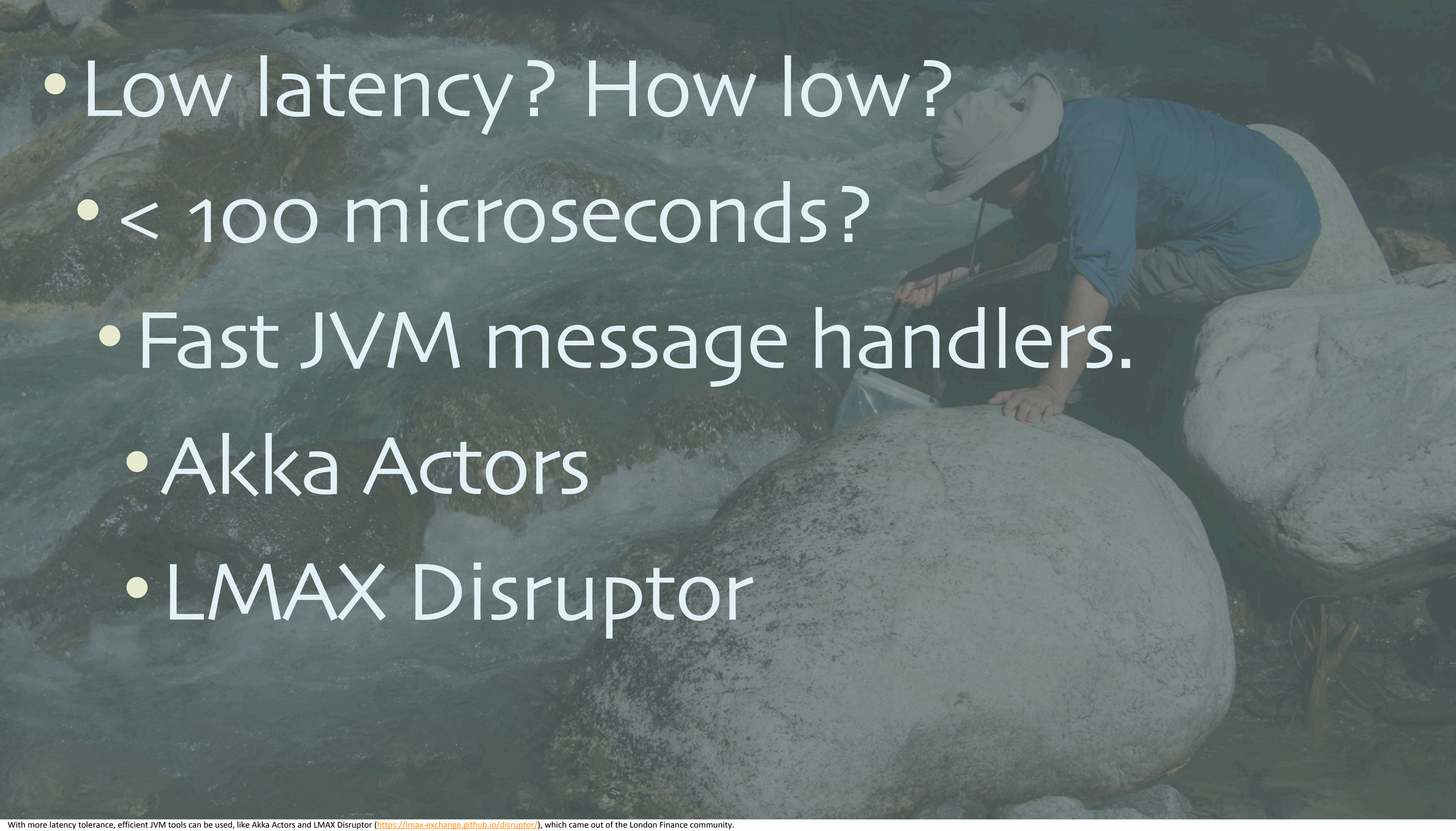


For this kind of performance, our primary job is *removing* sources of latency. Hence no VMs (like the JVM), minimizing layers in the network stack, even custom hardware and encoding algorithms in silicon, such as uses of FPGAs.

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



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

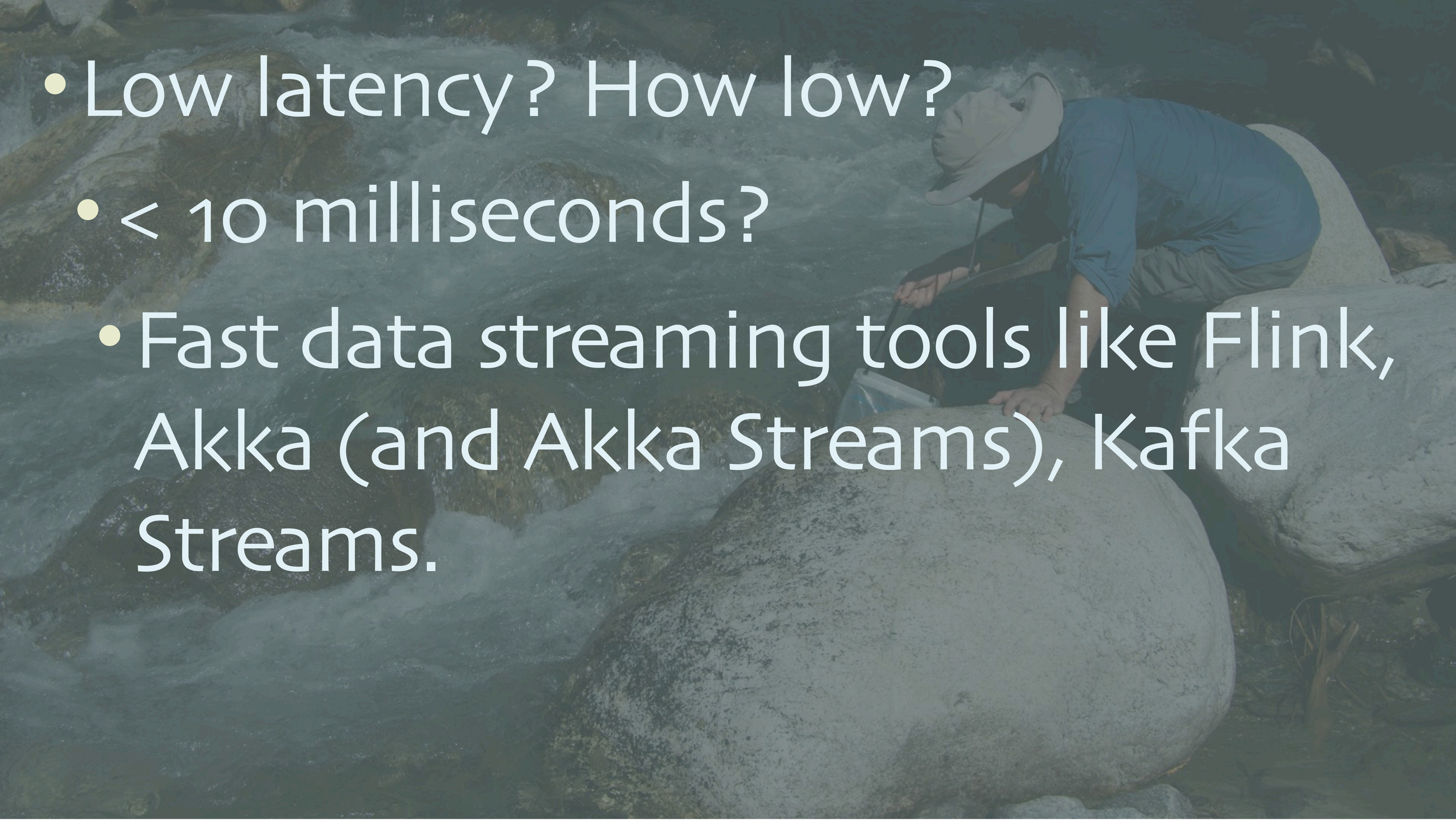


With more latency tolerance, efficient JVM tools can be used, like Akka Actors and LMAX Disruptor (https://lmax-exchange.github.io/disruptor/), which came out of the London Finance community.

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

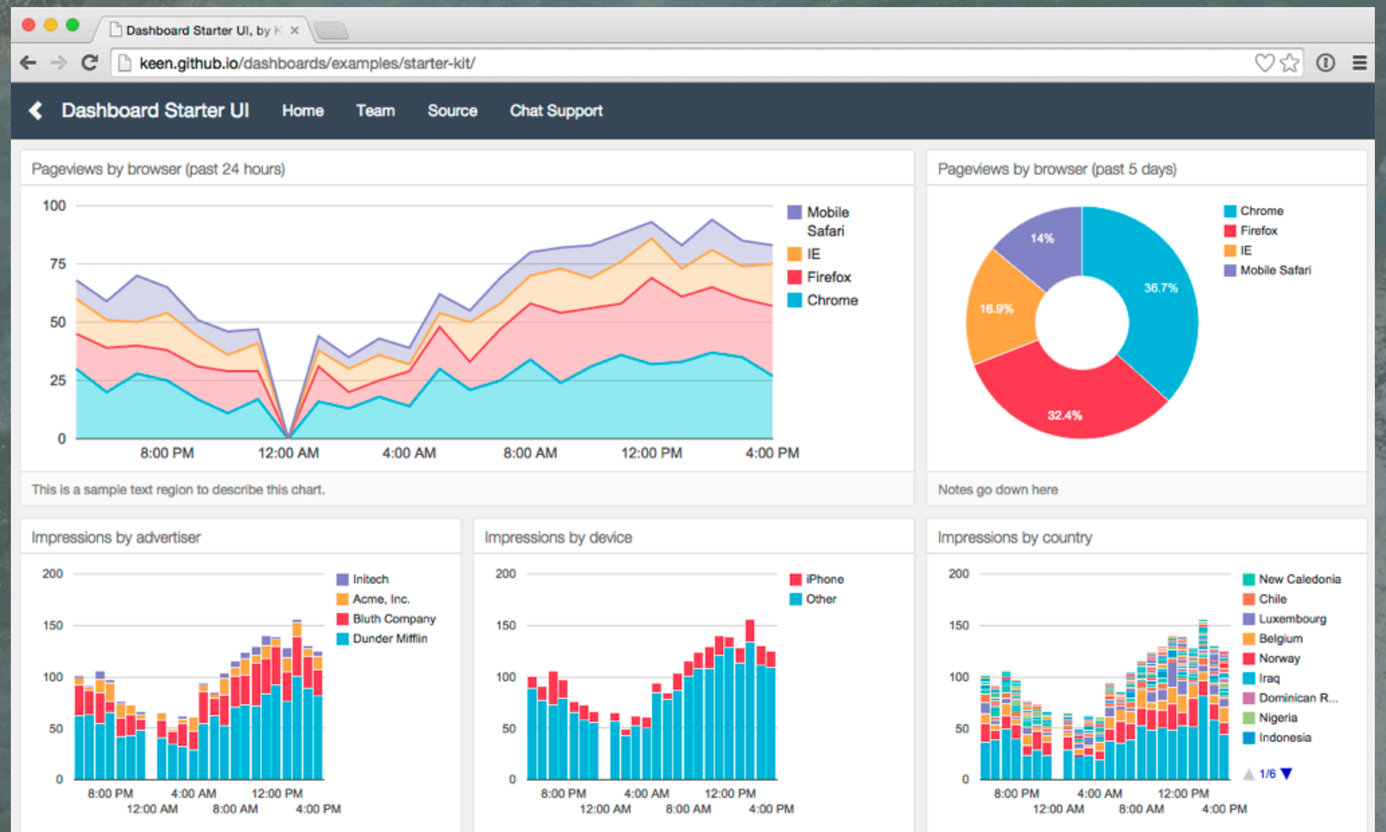
http://money.cnn.com/2017/05/12/pf/credit-card-mistakes/index.html

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.



Now we're in the low end of what high volume, fast data architectures are designed for. Note that part of the latency here is the end-to-end system latency for passing through Kafka (with short topic queues!!), etc.

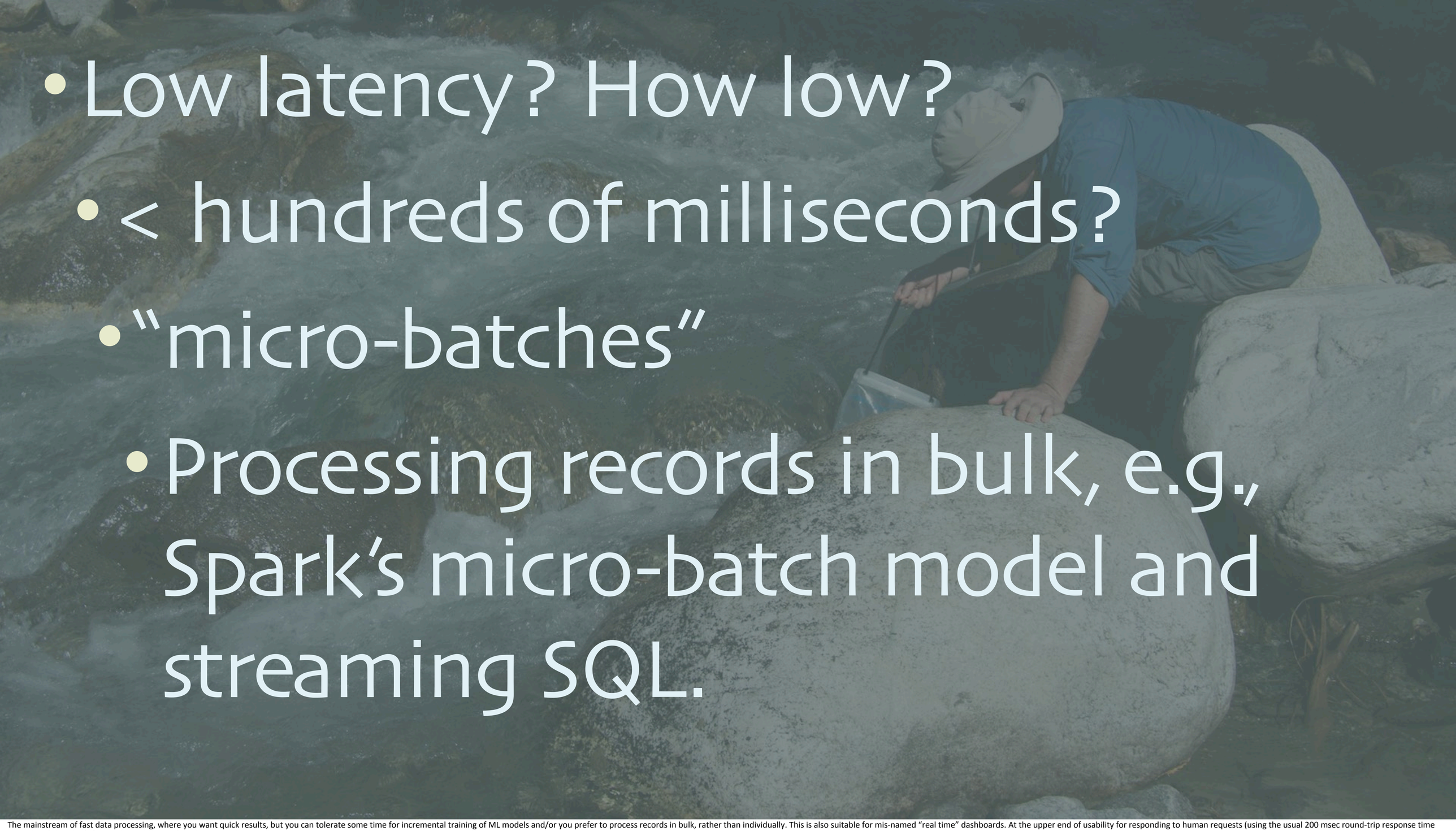
- Low latency? How low?
- < hundreds of milliseconds?



https://github.com/keen/dashboards

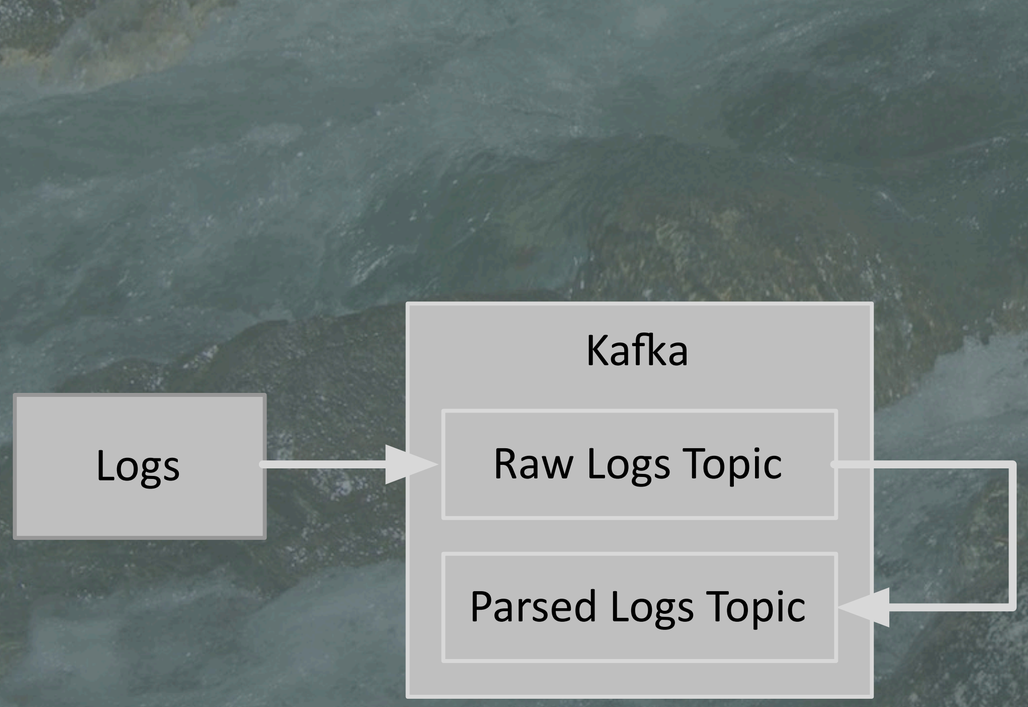
https://www.coursera.org/learn/machine-learning

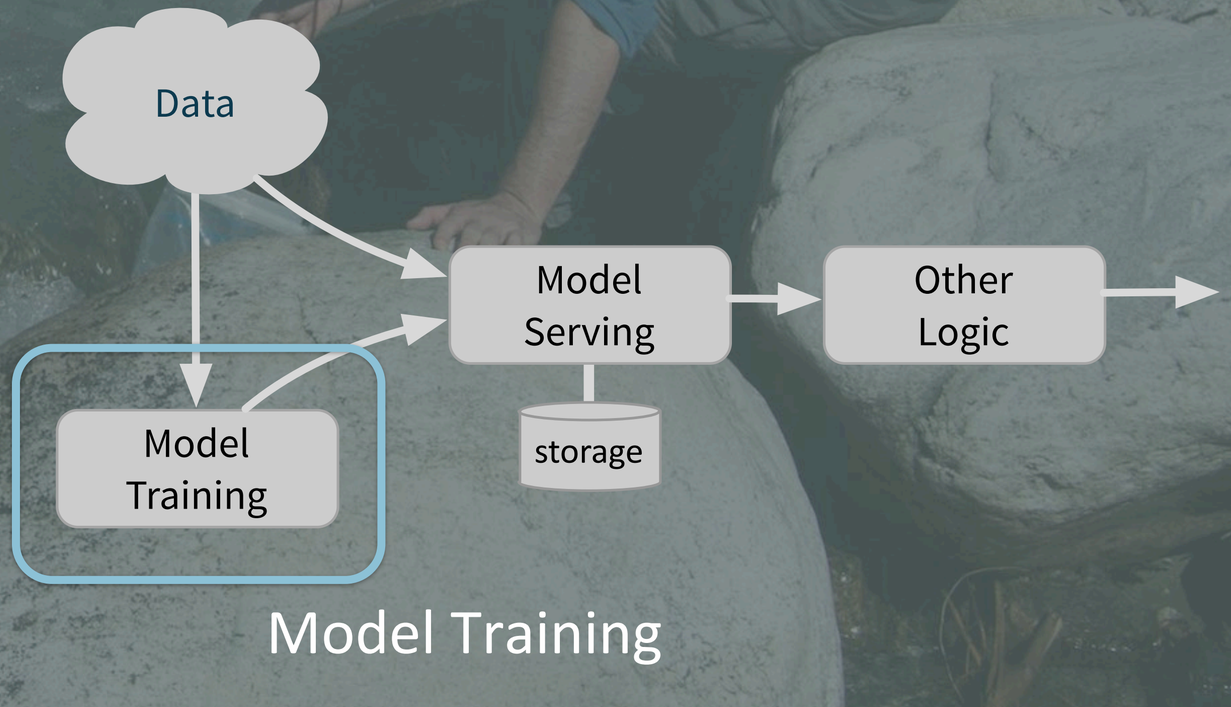
The mainstream of fast data processing, where you want quick results, but you can tolerate some time for incremental training of ML models and/or you prefer to process records in bulk, rather than individually. This is also suitable for mis-named “real time” dashboards. At the upper end of usability for responding to human requests (using the usual 200 msec round-trip response time upper limit), but don’t forget about the rest of the latency in the roundtrip request-response pipeline!



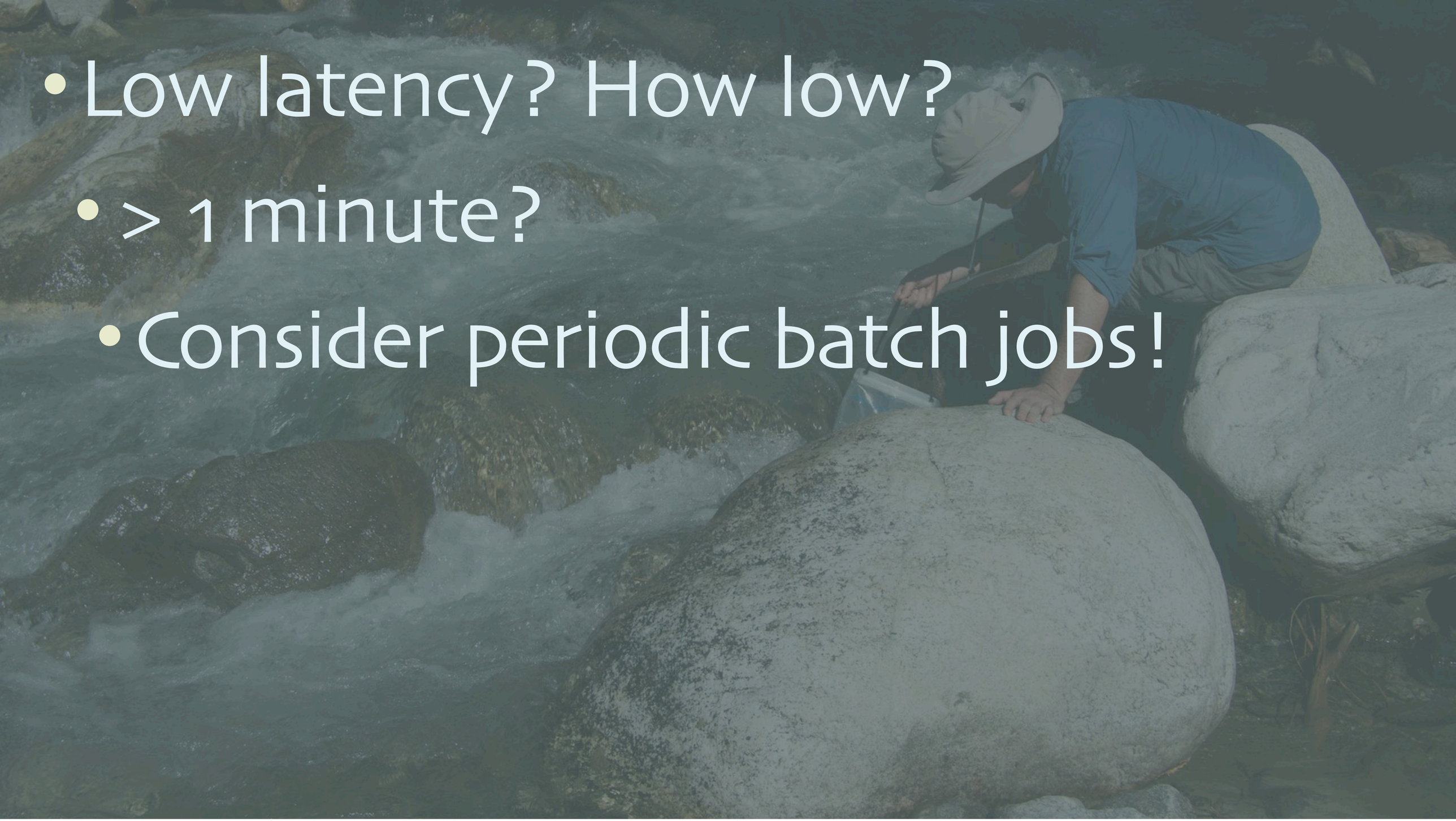
The mainstream of fast data processing, where you want quick results, but you can tolerate some time for incremental training of ML models and/or you prefer to process records in bulk, rather than individually. This is also suitable for mis-named "real time" dashboards. At the upper end of usability for responding to human requests (using the usual 200 msec round-trip response time upper limit), but don't forget about the rest of the latency in the roundtrip request-response pipeline!

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





The realm of tools like Spark Streaming, where larger “mini-batches” (vs. micro-batches) are processed. Used for more expensive, incremental model training and larger bulk processing.

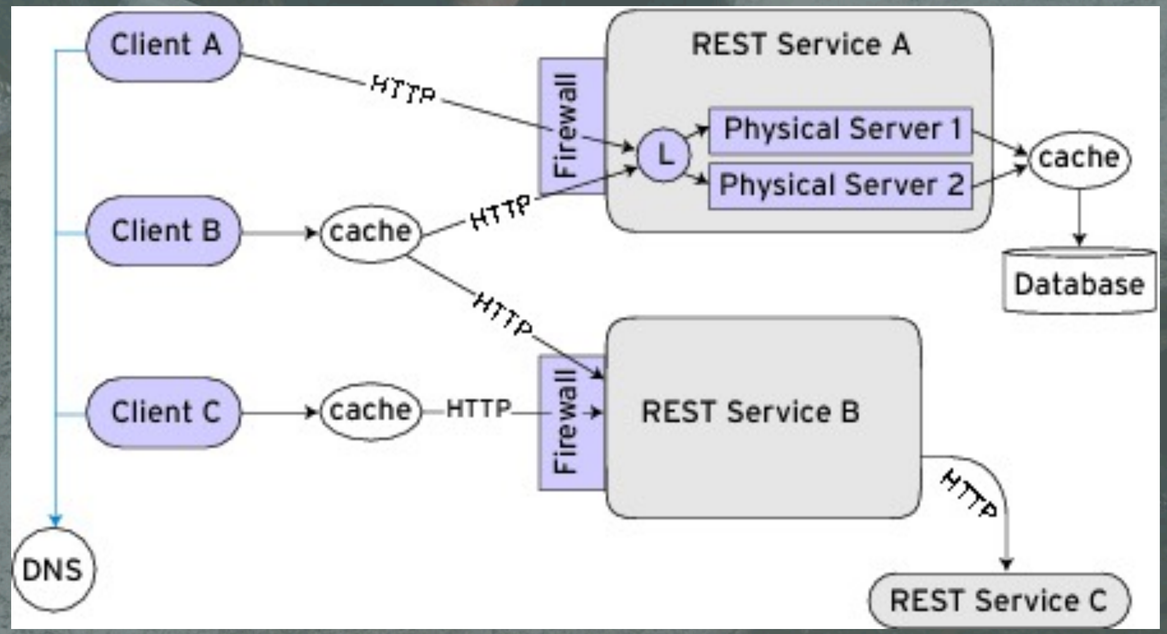


Once the latency is this high, it might actually make more sense to run frequent batch jobs, rather than keep a long-running, mini-batch system healthy.

- High Volume: How high?

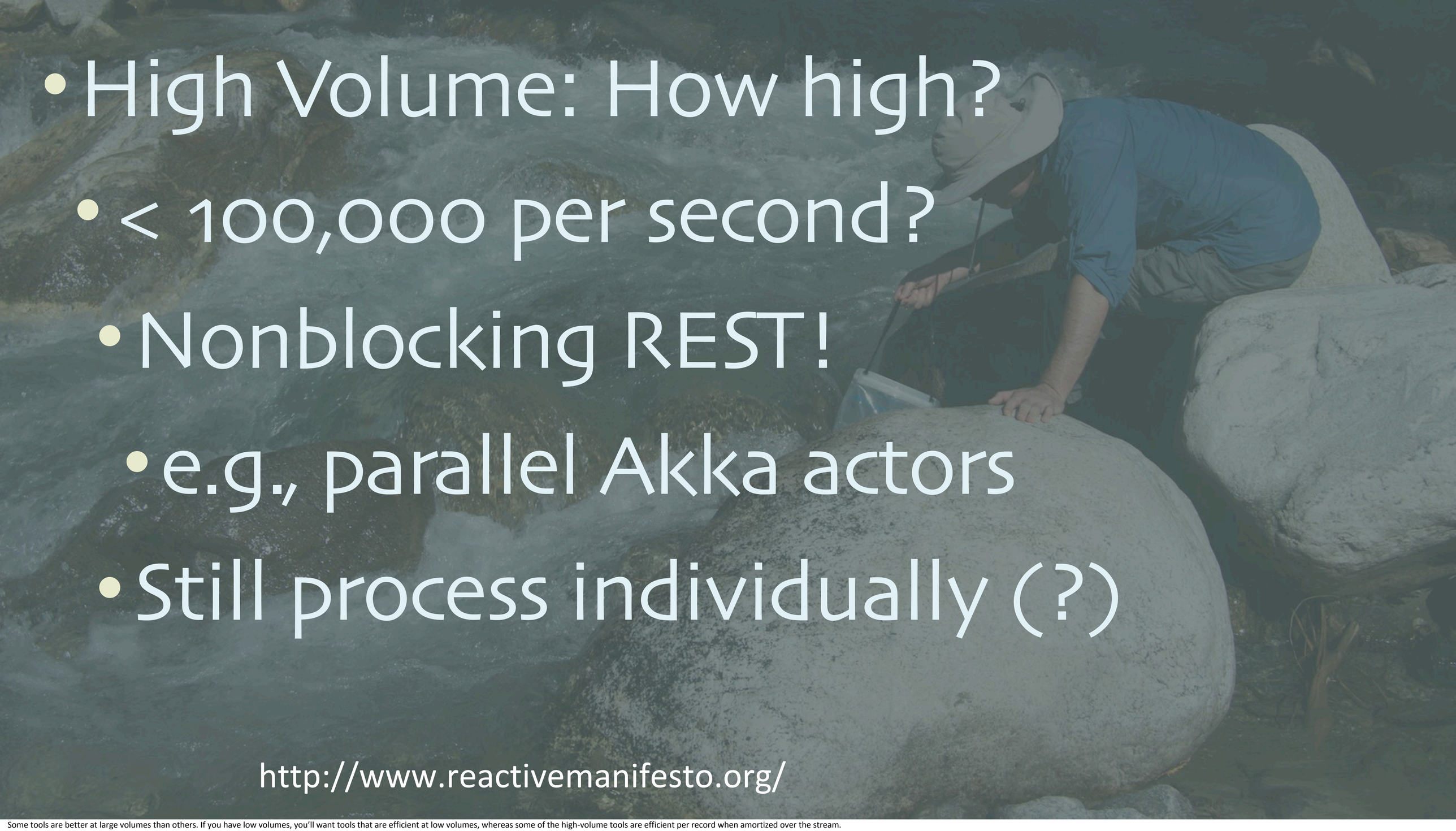
Some tools are better at large volumes than others. If you have low volumes, you'll want tools that are efficient at low volumes, whereas some of the high-volume tools are efficient per record when amortized over the stream.

- 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

These are relatively small volumes by modern standards and can be serviced individually by standard REST approaches.

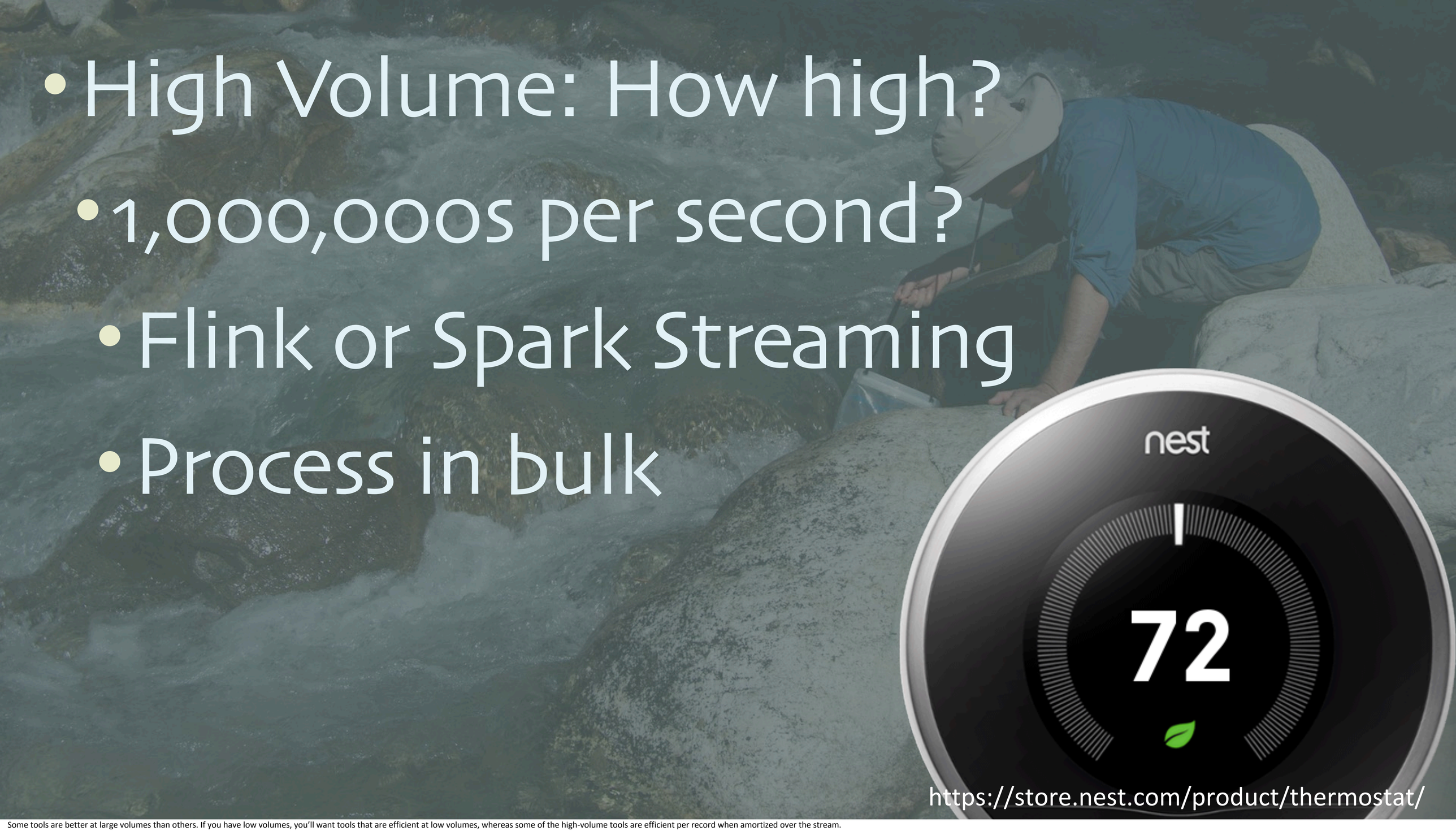


- High Volume: How high?
  - < 100,000 per second?
    - Nonblocking REST!
      - e.g., parallel Akka actors
    - Still process individually (?)

http://www.reactivemanifesto.org/

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?
  - 1,000,000s per second?
    - Flink or Spark Streaming
- Process in bulk



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

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.

- Which kinds of data processing?

What do you want to do with the data?

- 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()
```

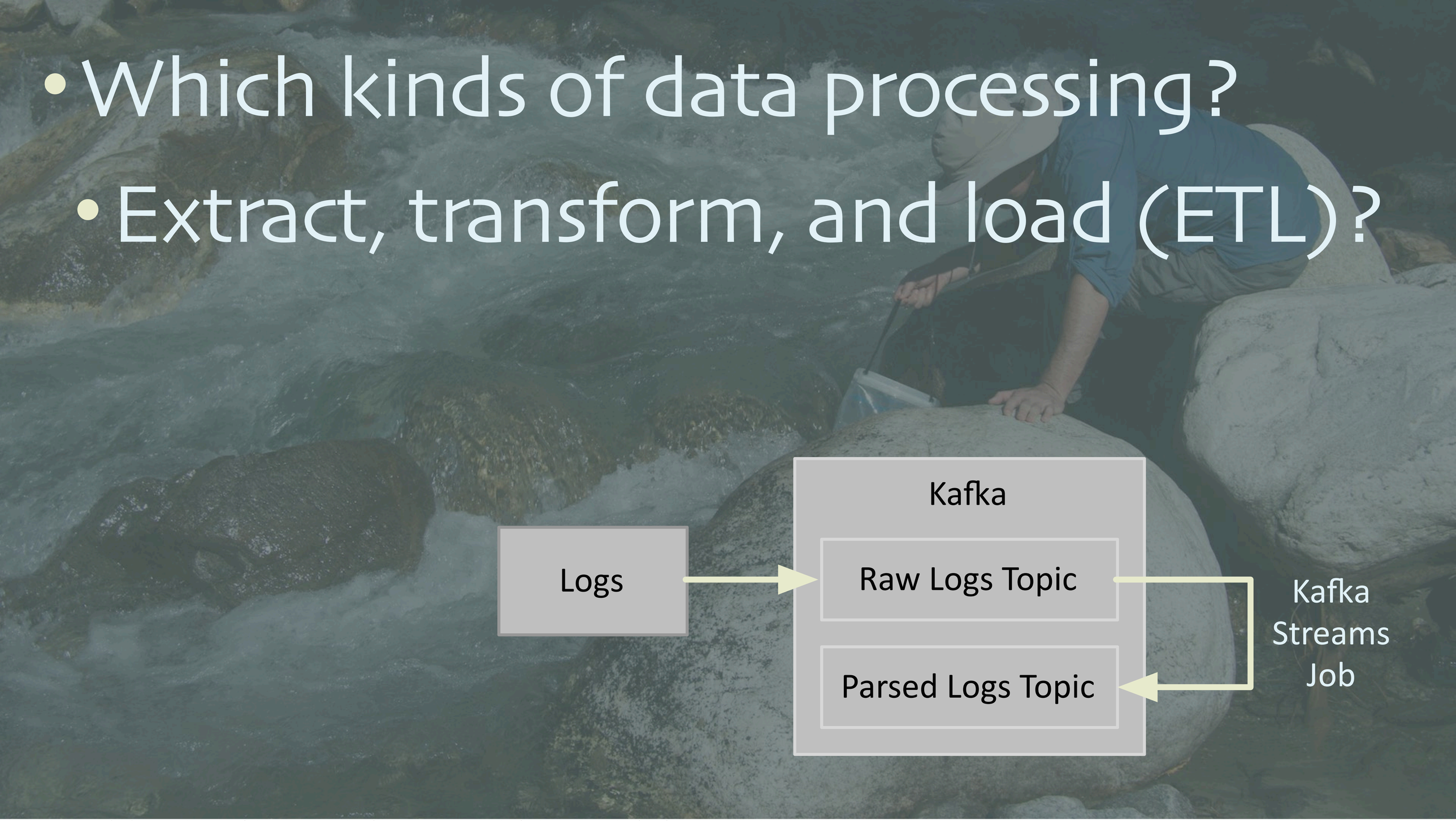
SQL example and a similar Spark Structured Streaming example using the SQL DSL in Scala, but I could have used actual SQL for it, too.

- Which kinds of data processing?
- "Dataflow"

```
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
```

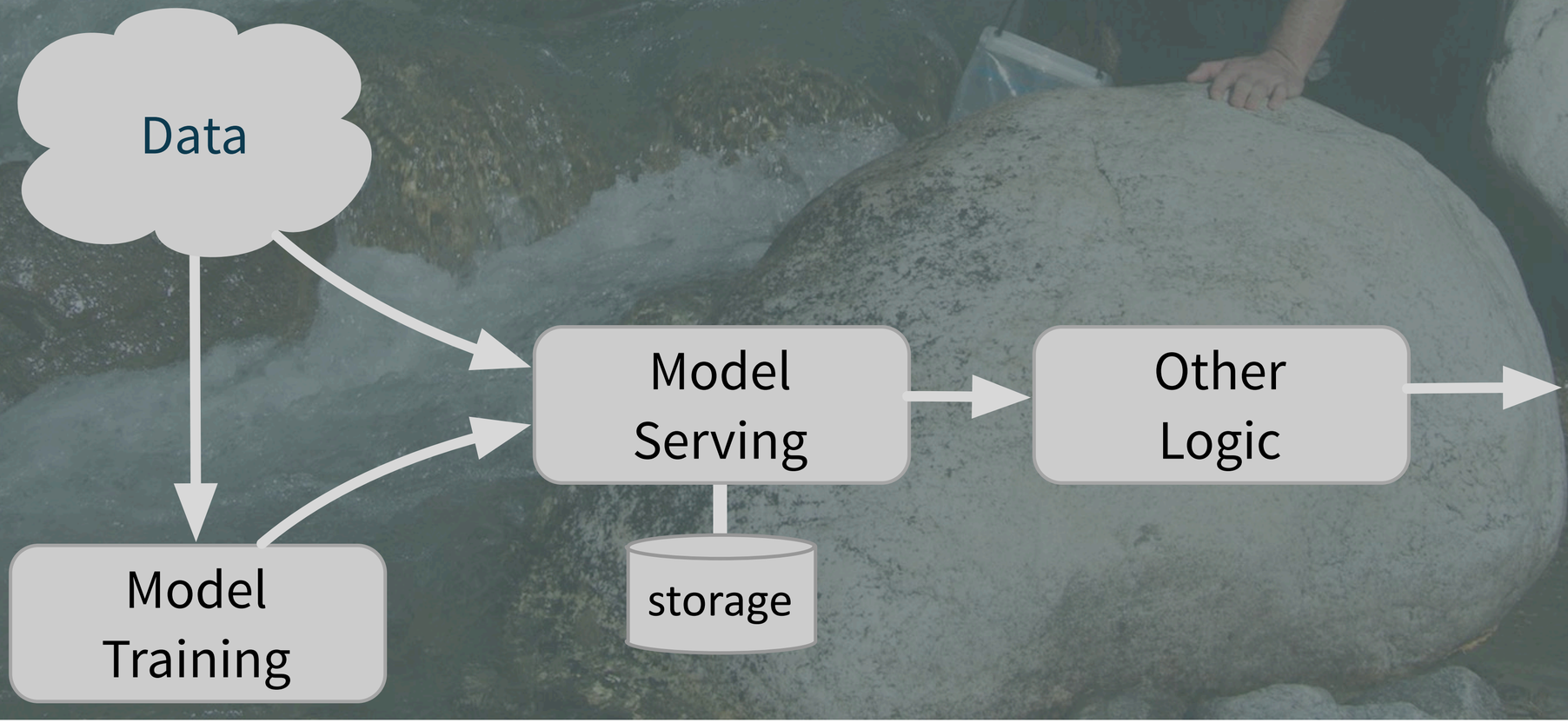
One reason Spark has become so popular is because it adopted Scala collection semantics for sequencing steps in a "pipeline". It's powerful and concise. This is part of the implementation for the "Inverted Index" algo. (The details aren't important for our purposes here, but if you're interested, see https://deanwampler.github.io/polyglotprogramming/papers/Spark-TheNextTopComputeModel.pdf)

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



Simple example where raw (text?) logs are ingested into one Kafka topic, then processed with a Kafka Streams job to clean them and transform them into a more suitable format for downstream consumption.

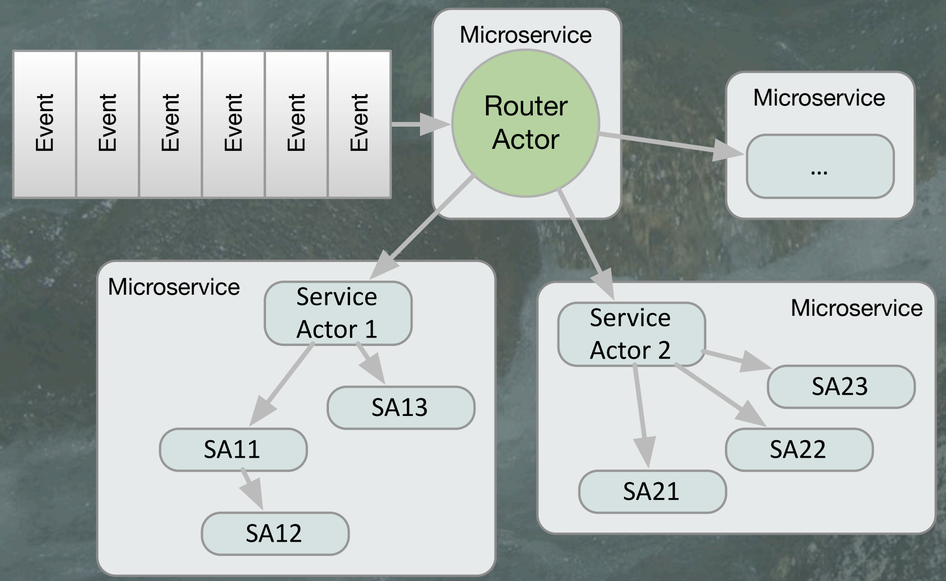
- Which kinds of data processing?
- Train and serve ML models?



Train models: use mini-batch to batch processing
Serve models (“score” data): any latency

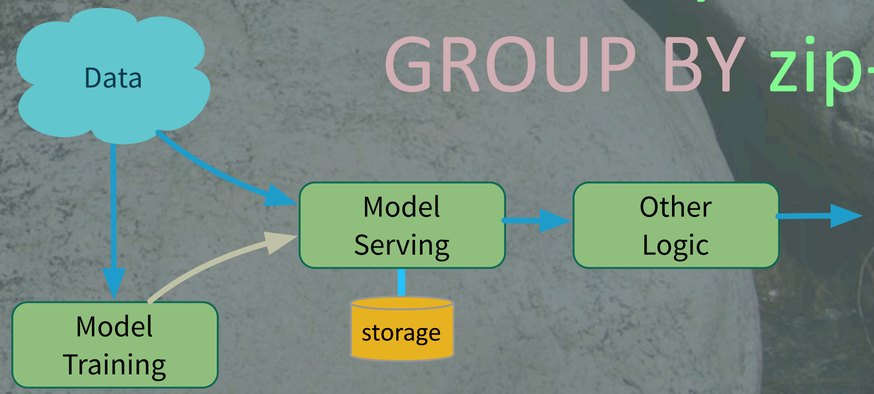
- Process data individually or in bulk?

Event-driven μ-services



"Record-centric" μ-services

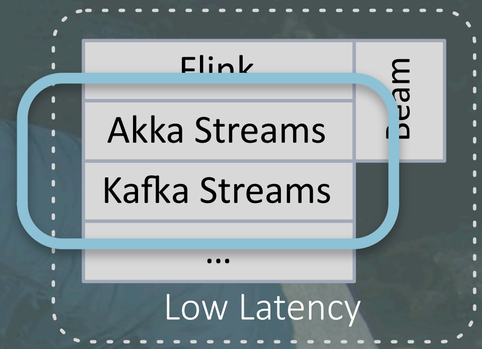
```
SELECT COUNT(*)
FROM my-iot-data
GROUP BY zip-code
```



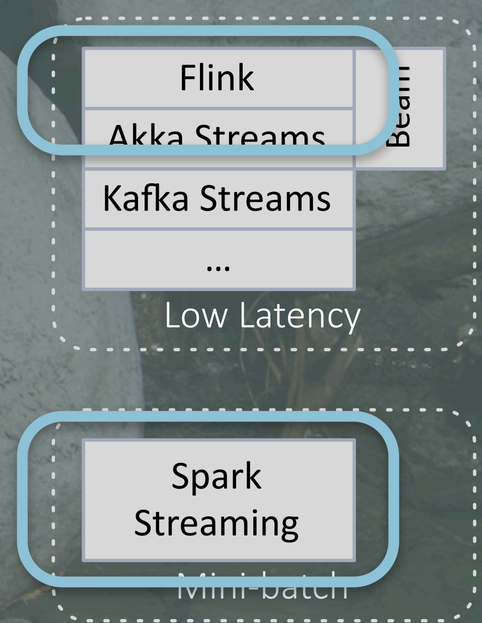
Events → Records

How we think about event processing falls on a spectrum. Per event is best for complex event processing, where each event needs to be analyzed, routed, etc. separately. Best when volumes are relatively low and the overhead per event is also low.
Processing in bulk, like SQL queries, training ML models, etc. best for large volumes where the system is efficient at scale (i.e., the overhead amortized over the bulk events is good), but low overhead per event at low volumes isn't important.

- Preferred application architecture?
- Streaming library in an app?

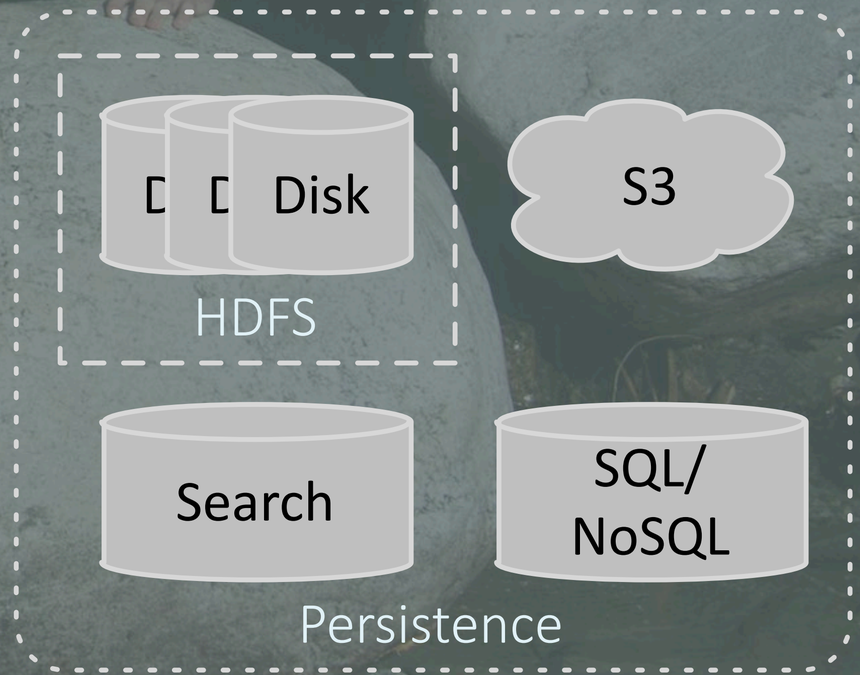


- Distributed services running your job?



The four streaming engines divide into those that are libraries which you embed into your apps (Akka Streams and Kafka Streams, although both systems also have available services that can be used for providing certain services) and systems that run daemons constantly that accept “job” submissions and manage decomposition into tasks, etc. (Spark and Flink).

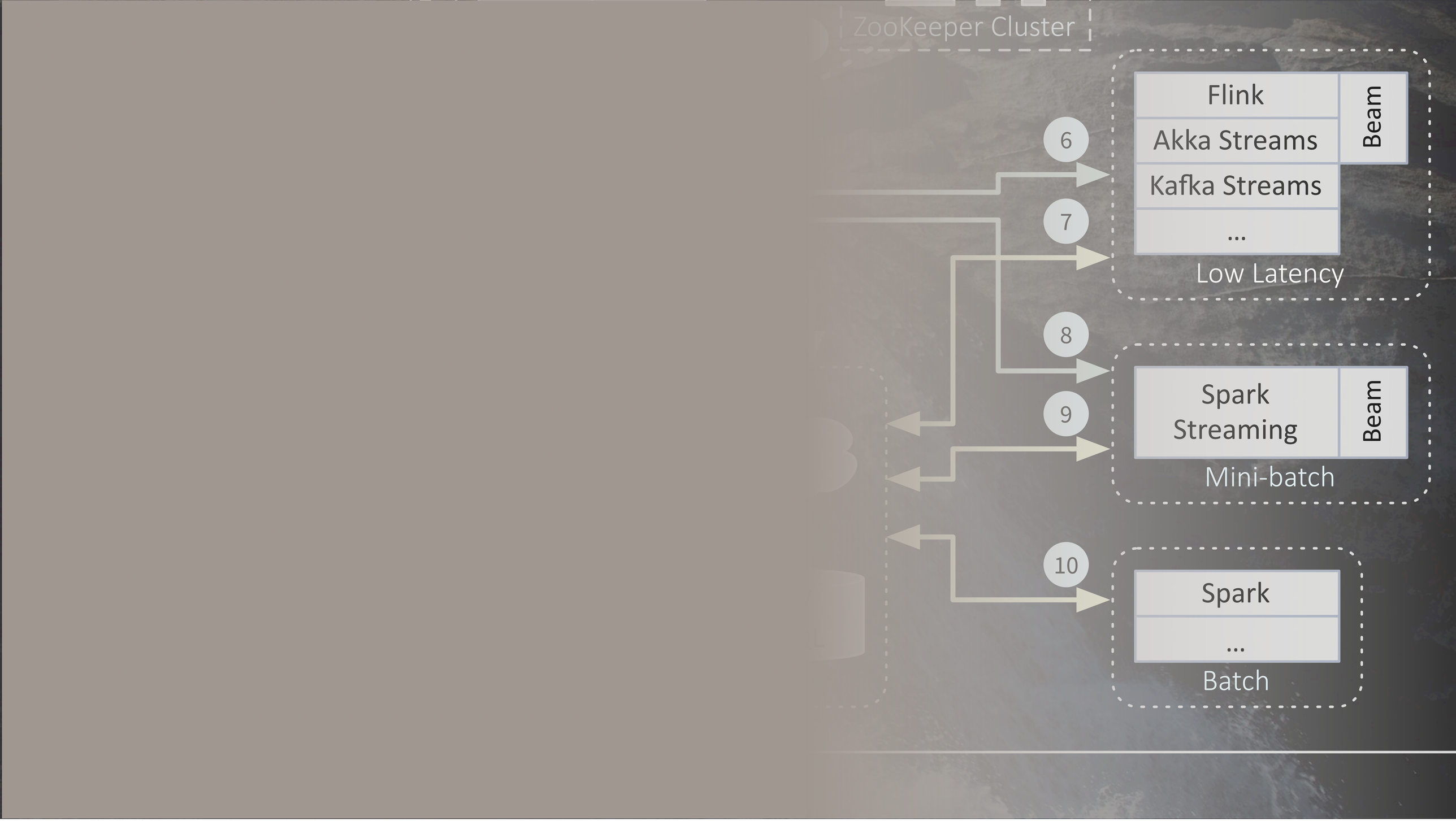
- Integration with other tools.
  - Akka, Flink, & Spark integrate with Databases, Kafka, file systems, REST, …
  - Kafka Streams only read & write Kafka topics.



Architecture is also concerned with connecting between components. Sending everything through Kafka provides a uniform, global mechanism, but often that's not ideal and you may need some tools to have direct connections to other tools. In most cases, there are plenty of connection options, but Kafka Streams deliberately only reads and writes Kafka topics. (However, Kafka Connect can be used to move data between Kafka and other stores.)

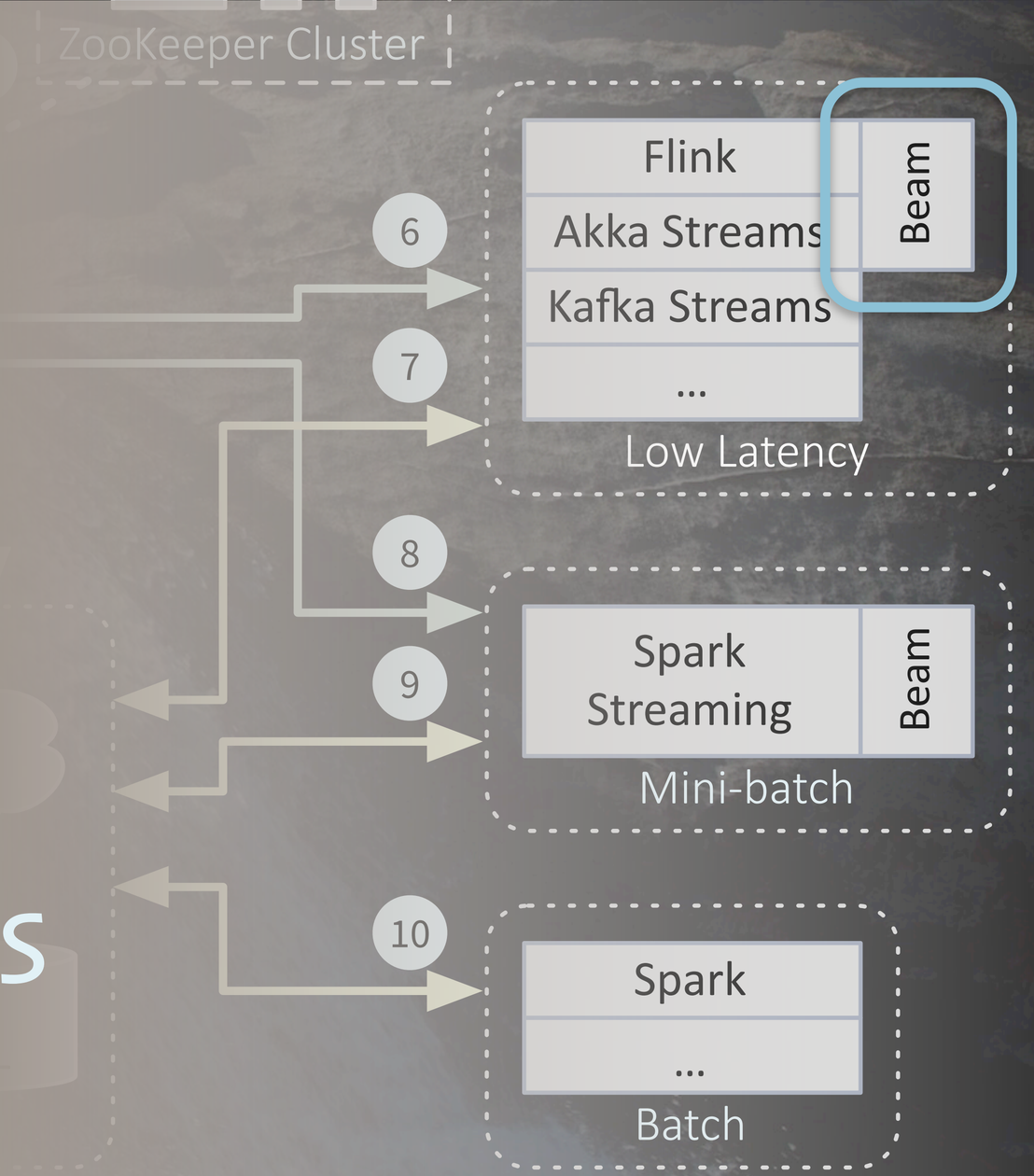
Best of Breed Streaming Engines

Photo: Top of Nevada Falls, Yosemite National Park, California.



Focusing in on the pure (low-latency) streaming and mini batch engines on the right...

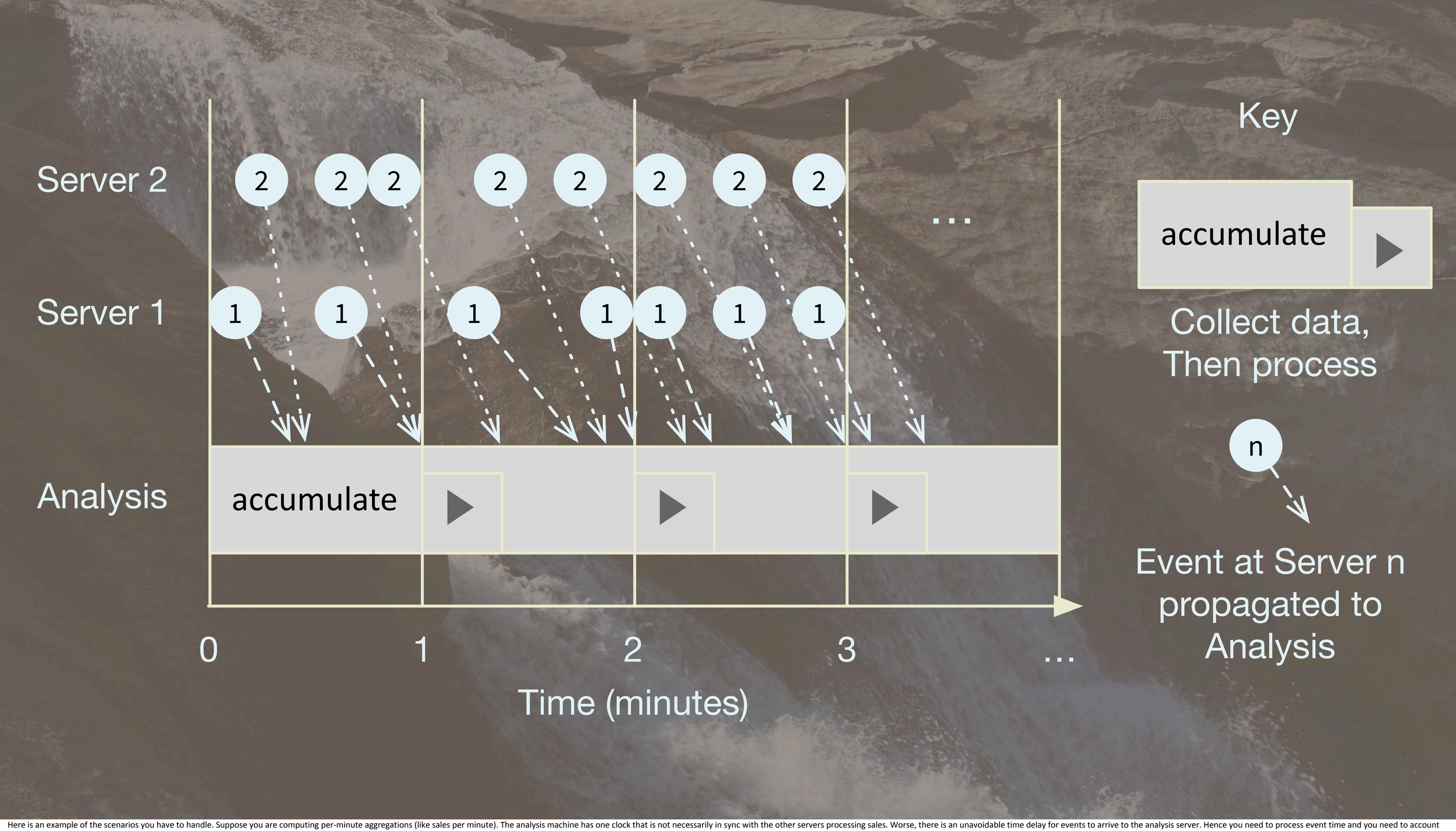
- Apache Beam
  - (Google Dataflow)
  - Requires a "runner"
  - Most sophisticated streaming semantics



beam

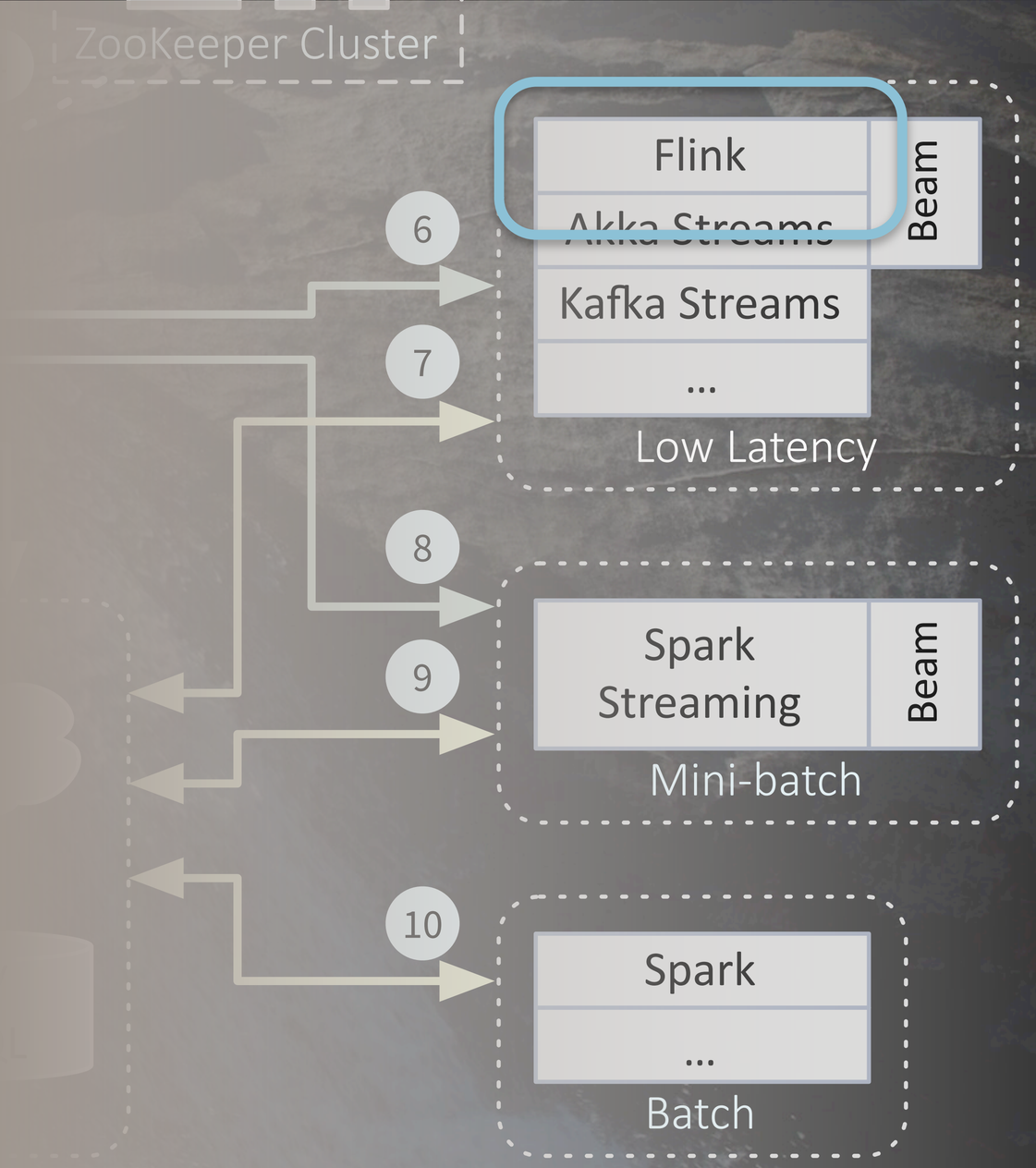
See these blog posts: https://www.oreilly.com/people/09f01-tyler-akidau

Google has spent years thinking about all the things a streaming engine has to handle if you want to do accurate calculations on streams, accounting for all the contingencies that can happen. The latest incarnation is Google Dataflow, available as a service in Google Cloud Platform. Google recently open-sourced the part of Dataflow for defining "data flows" with these sophisticated semantics, called Apache Beam. Beam doesn't provide a runner, so third-party tools provide this capability.
I recommend these blog posts by Tyler Akidau from Google, where he explains streaming semantics, as defined in Beam. See also his new O'Reilly book.



Here is an example of the scenarios you have to handle. Suppose you are computing per-minute aggregations (like sales per minute). The analysis machine has one clock that is not necessarily in sync with the other servers processing sales. Worse, there is an unavoidable time delay for events to arrive to the analysis server. Hence you need to process event time and you need to account for late arriving data, not only small delays where some events/minute will arrive within the following minute, but even large delays due to network partitions, servers busy, etc., etc. This is one example of the challenges posed by streaming when you want accurate vs. approximate results.

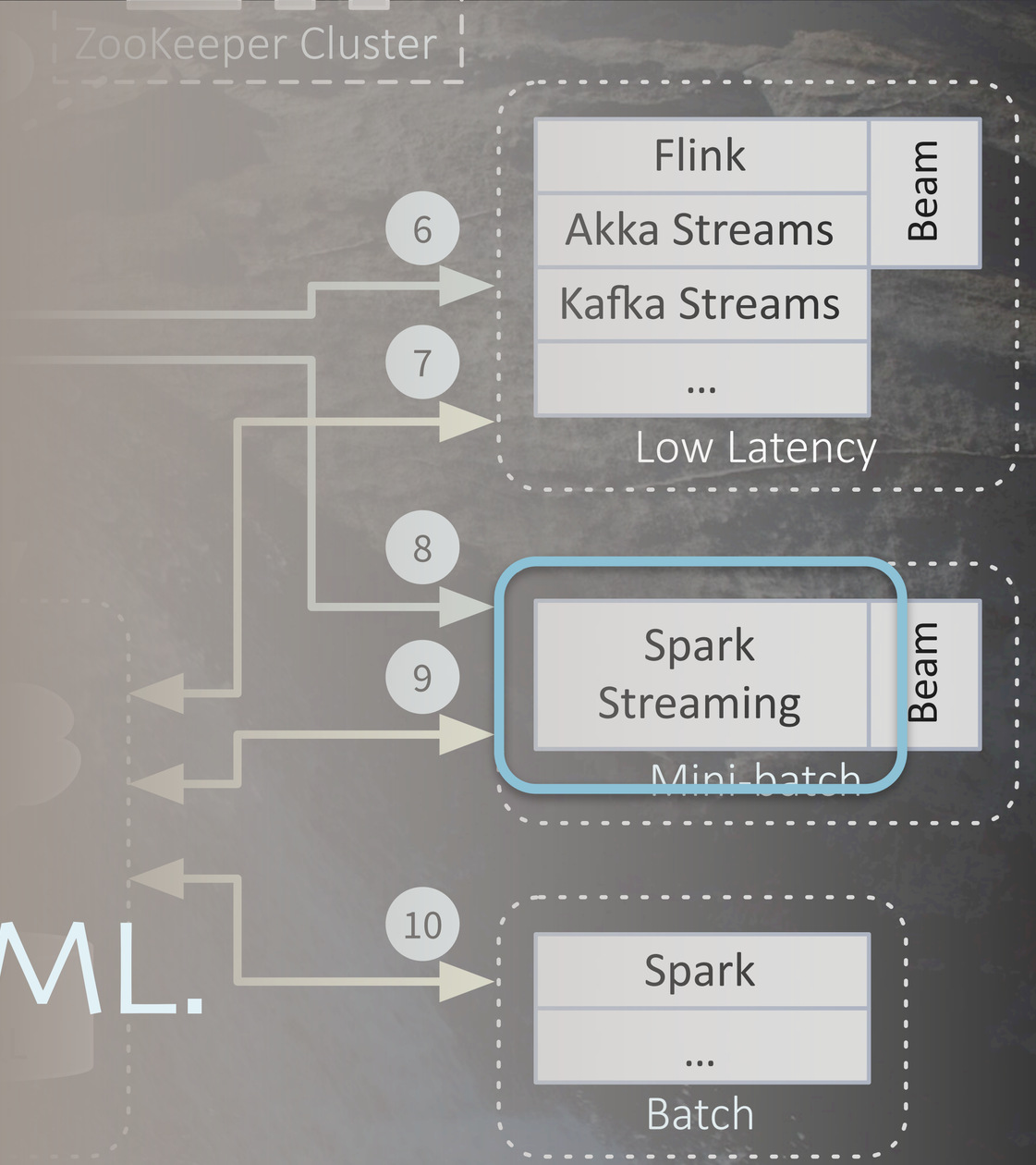
- Apache Flink
  - High volume
  - Low latency
  - Beam Runner
  - Evolving SQL, ML



Flink

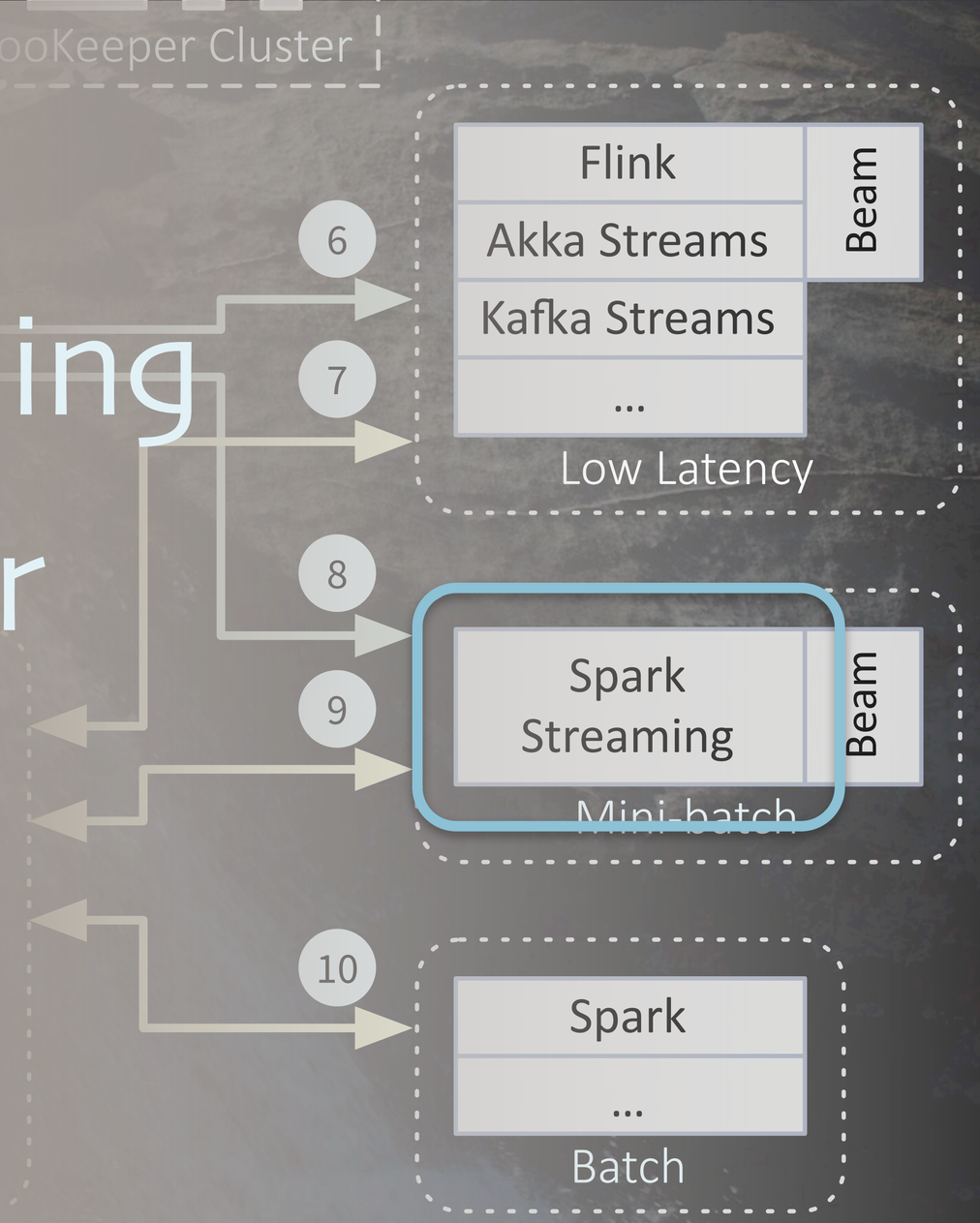
Flink provides two unique capabilities among these choices, 1) low-latency processing at scale and 2) it is the most mature runner for Apache Beam data flows (at least that I know of) outside Google's own Dataflow engine.

- Spark Streaming
  - Mini-batch model
    - ~0.5 sec latency
    - (but dropping…)
  - Ideal for Rich SQL, ML.



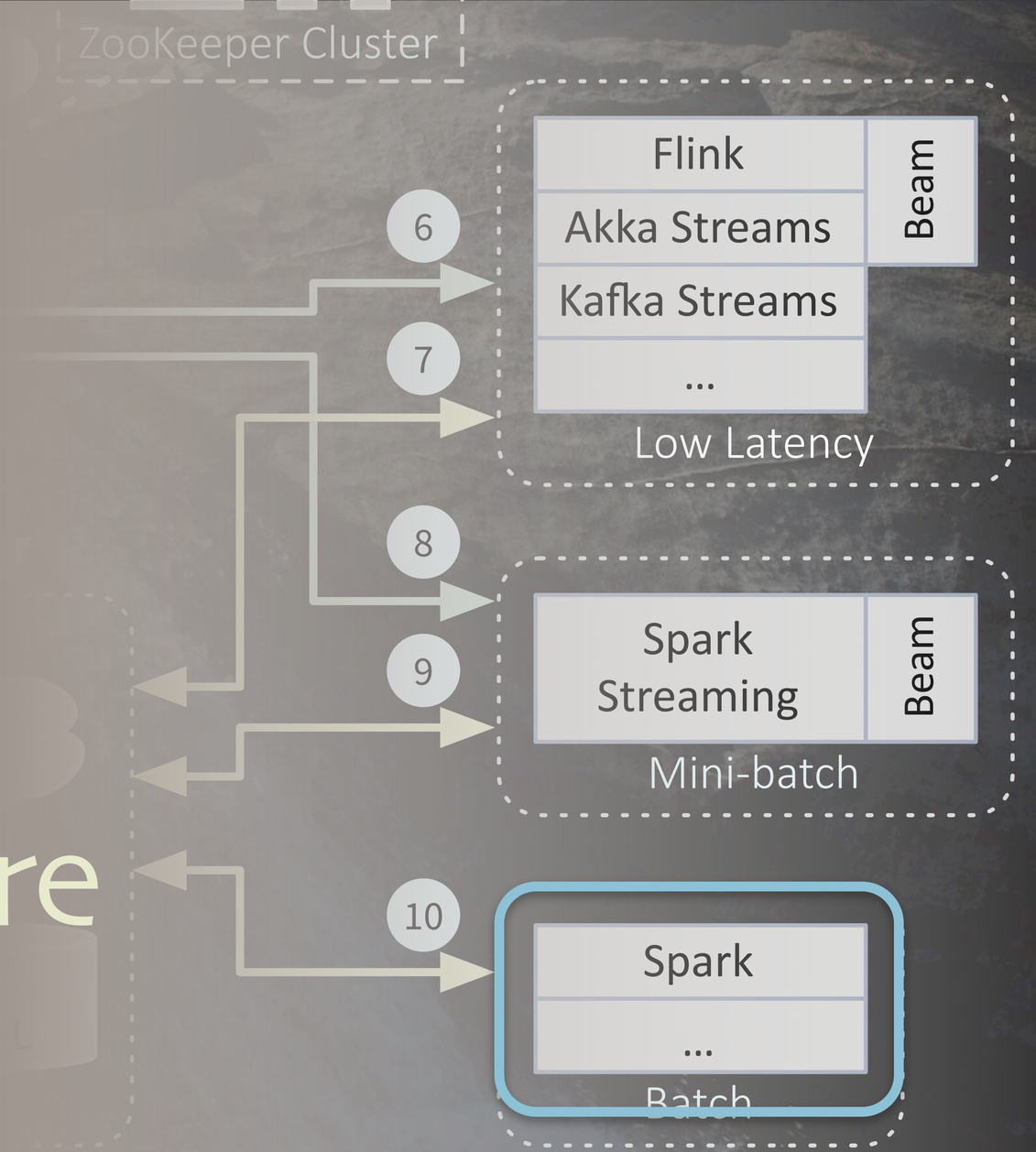
Spark Streaming is a mini-batch model, although there is a new low-latency streaming model that is available and is maturing. So, today, use it for more expensive calculations, like training ML models, but don't use it when the lowest-latency processing is required. Lightbend's Fast Data Platform is working on tools to make it easy to train ML models in Spark and serve them with the other tools. Another advantage of Spark is the rich ecosystem of tools for a wide variety of batch and streaming scenarios. You can maybe push the latency down to 200 or even 100 msecs, but that's difficult.

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



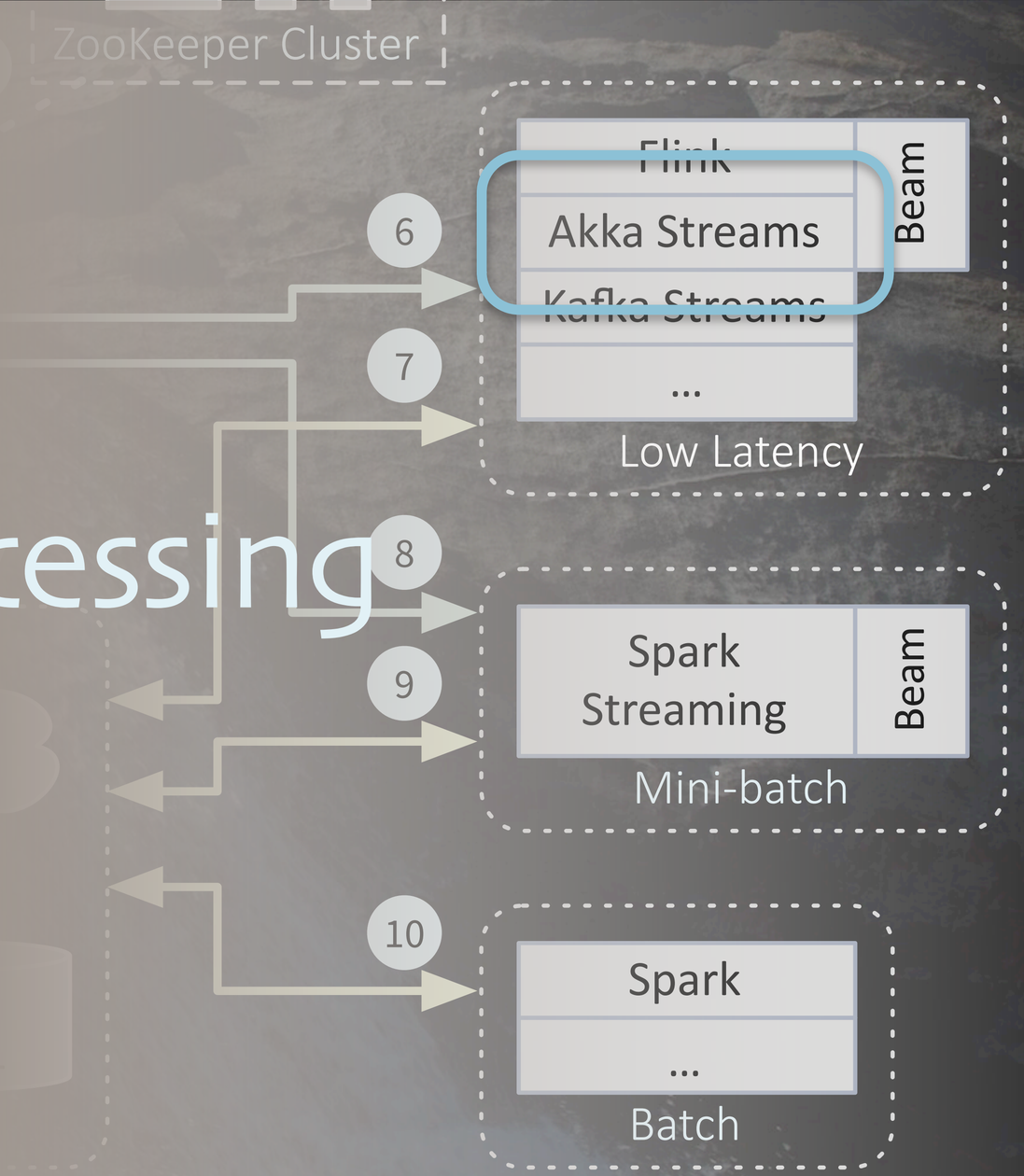
Spark Streaming is a mini-batch model, although this is slowly being replaced with a true, low-latency streaming model. So, today, use it for more expensive calculations, like training ML models, but don't use it when the lowest-latency processing is required. Lightbend's Fast Data Platform is working on tools to make it easy to train ML models in Spark and serve them with the other tools. Another advantage of Spark is the rich ecosystem of tools for a wide variety of batch and streaming scenarios. You can maybe push the latency down to 200 or even 100 msecs, but that's difficult.

- Spark Batch
  - Same features as streaming.
  - Supports the Lambda Architecture



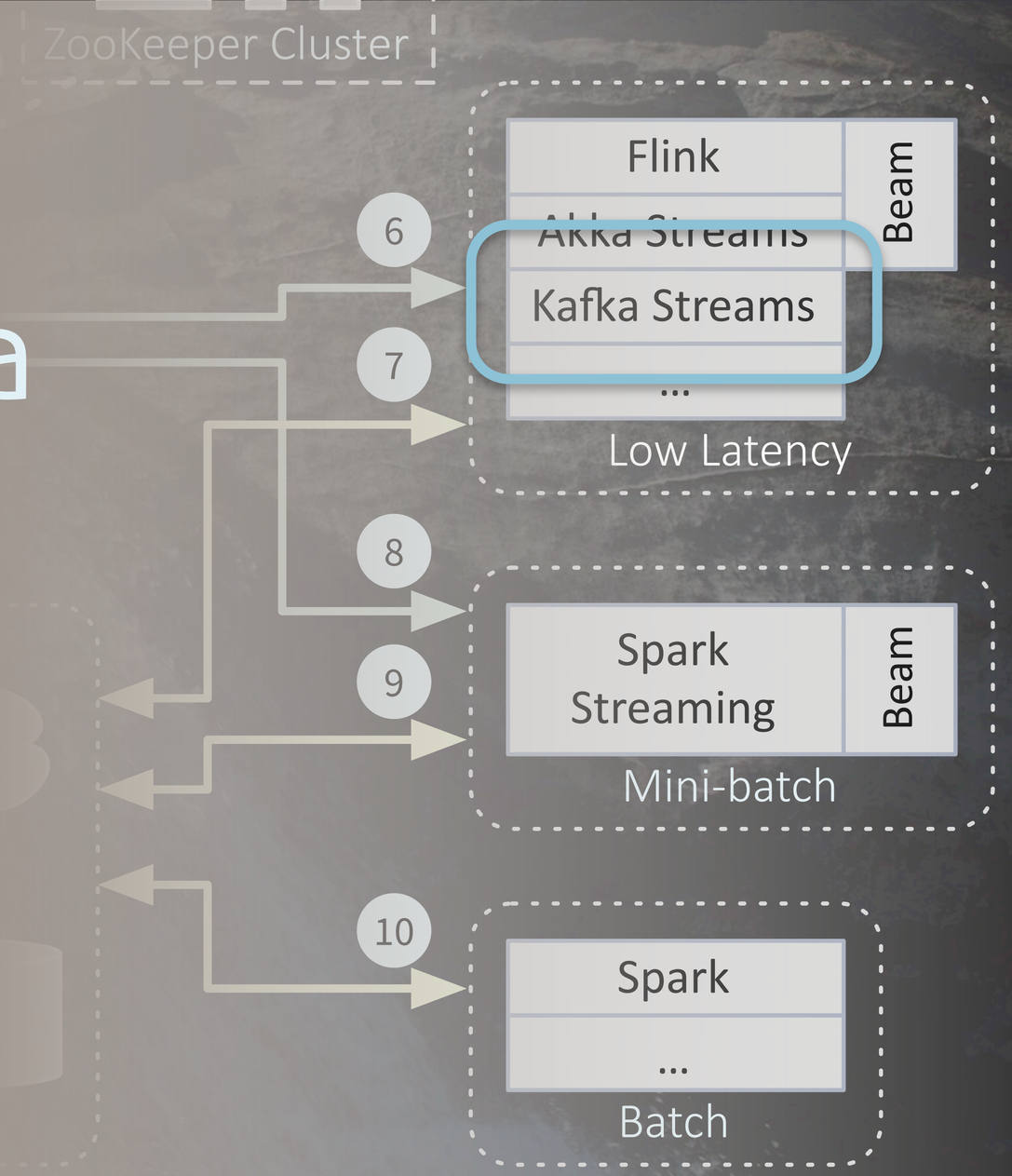
If your latency requirements are minutes, consider Spark batch jobs. Restarting every few minutes is generally easier than keeping a streaming process running for months! Spark's streaming and batch support is helpful for the Lambda Architecture (http://lambda-architecture.net/), which mixes streaming and batch data analytics into a unified view (although Lambda is falling out of favor).

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



Akka Streams adds a streaming abstraction on top of Akka actors. It's ideal when complex event processing (CEP) is the preferred model, as opposed to in bulk processing of data. Akka's powerful Actor model abstracts over the details of thread programming for highly concurrent apps, with libraries for clustering, persisting state, and data interchange with many sources and sinks (the "Alpakka" project).

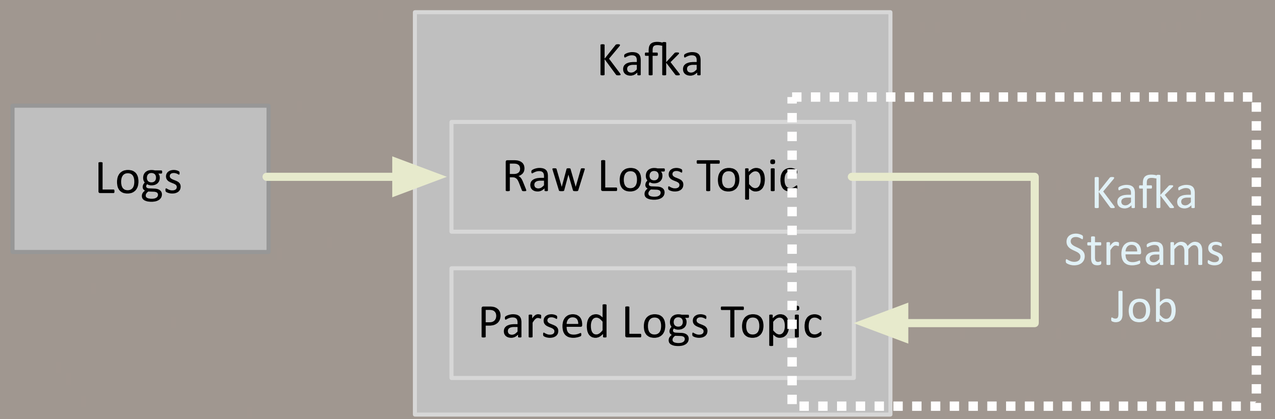
- Kafka Streams
  - Low overhead Kafka topic processing
  - Ideal for ETL and aggregations

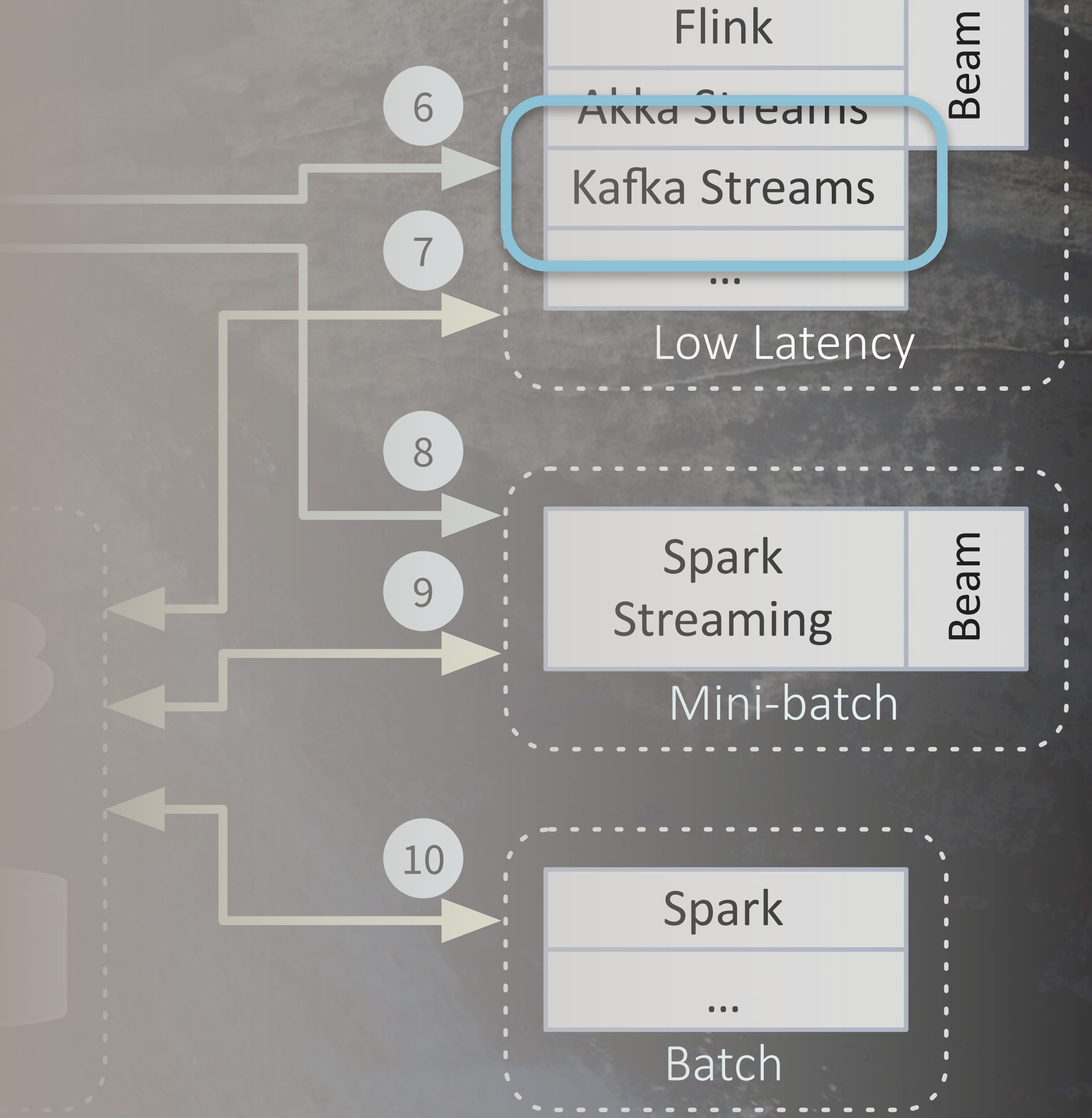


APACHE kafka™
A distributed streaming platform

Kafka Streams is a great 80% solution that works close with Kafka (in terms of production deployment and overhead). It is designed to read Kafka topics, perform processing like transformations, filtering, aggregations, “last-seen” value for keys, etc. then write results back to Kafka. It nicely addresses many common design challenges, but isn’t designed to be a complete solution for stream processing, e.g., running SQL queries and training ML models.

- Kafka Streams
  - "Exactly once" with Kafka 0.11.0

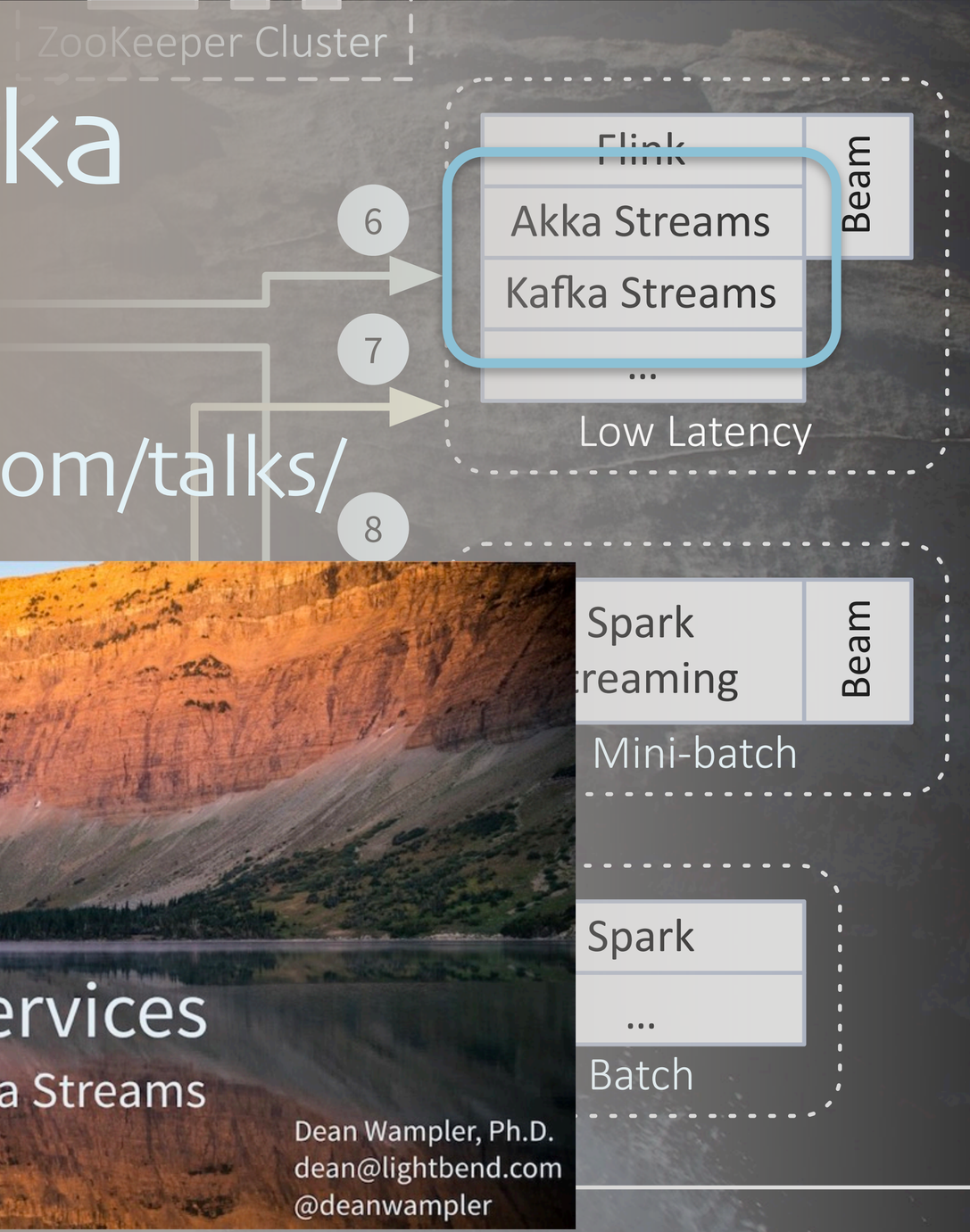




APACHE kafka™
A distributed streaming platform

Kafka Streams is a great 80% solution that works close with Kafka (in terms of production deployment and overhead). It is designed to read Kafka topics, perform processing like transformations, filtering, aggregations, "last-seen" value for keys, etc. then write results back to Kafka. It nicely addresses many common design challenges, but isn't designed to be a complete solution for stream processing, e.g., running SQL queries and training ML models.

- Akka Streams vs. Kafka Streams
  - Also at polyglotprogramming.com/talks/



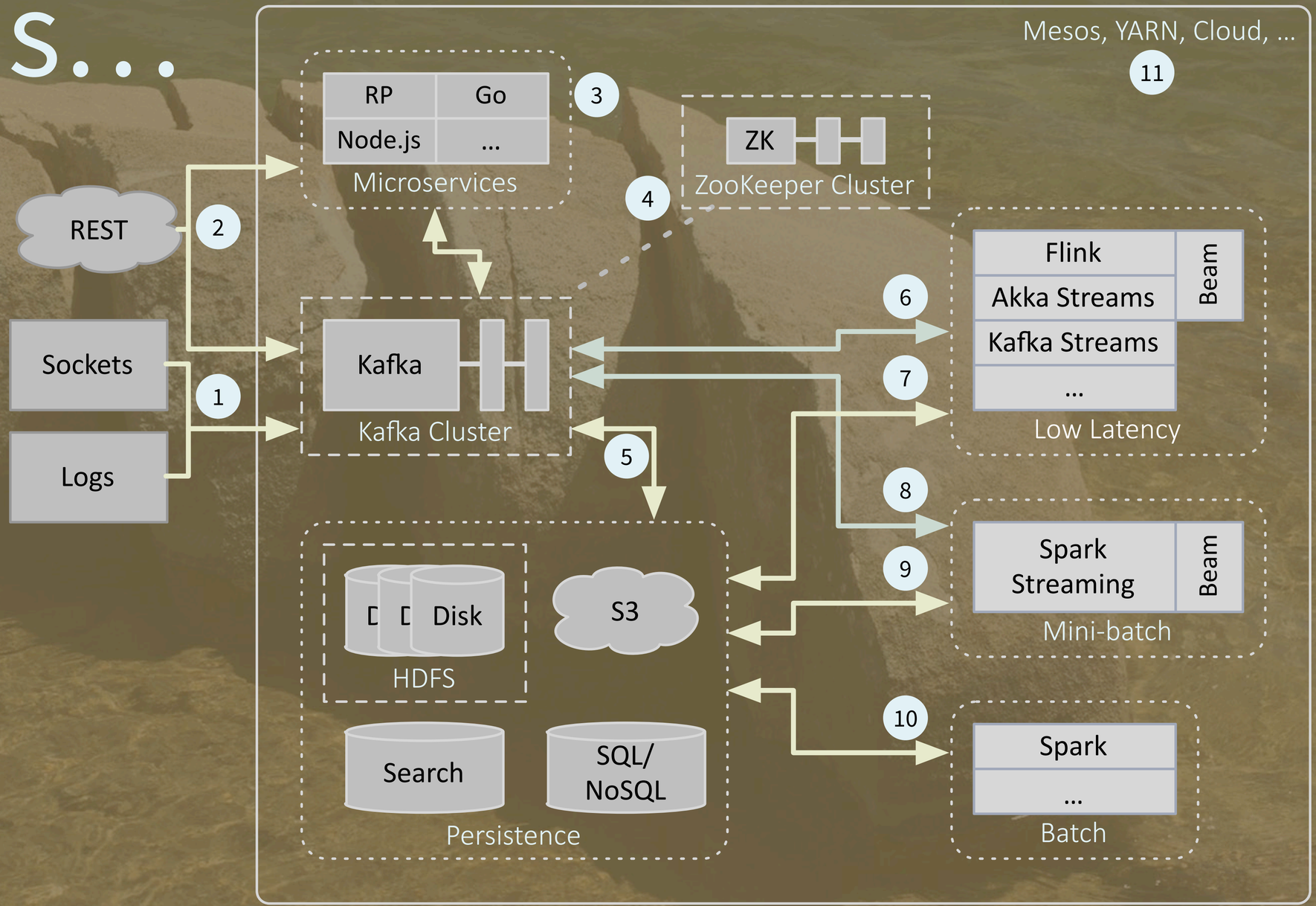
A new talk I've been doing this year explores Akka Streams vs. Kafka Streams in greater depth. It's available at the same polyglotprogramming.com/talks site.

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.

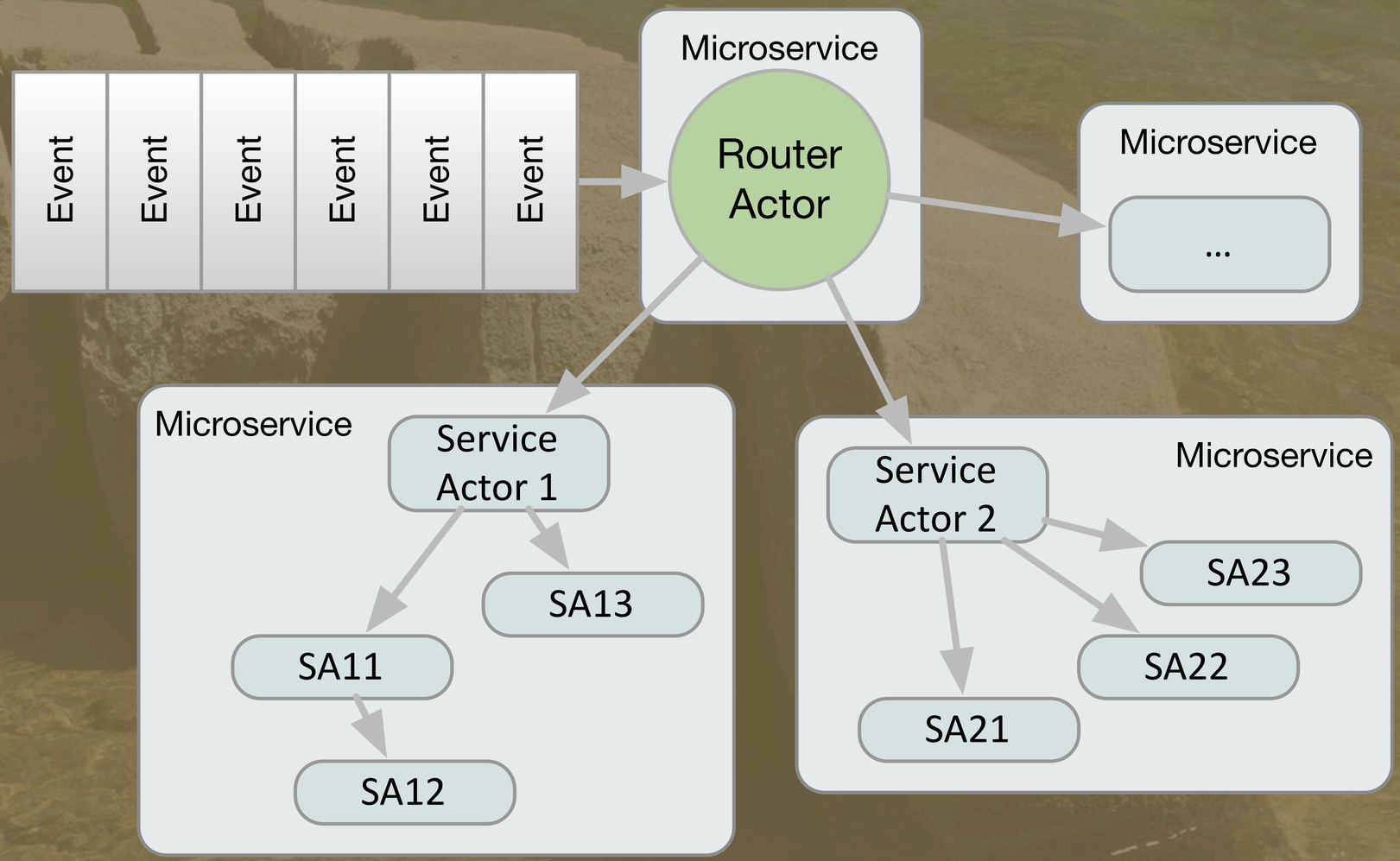
Photo: "Sliced" rocks, 1000 Island Lake, Ansel Adams Wilderness, California.

How is this…



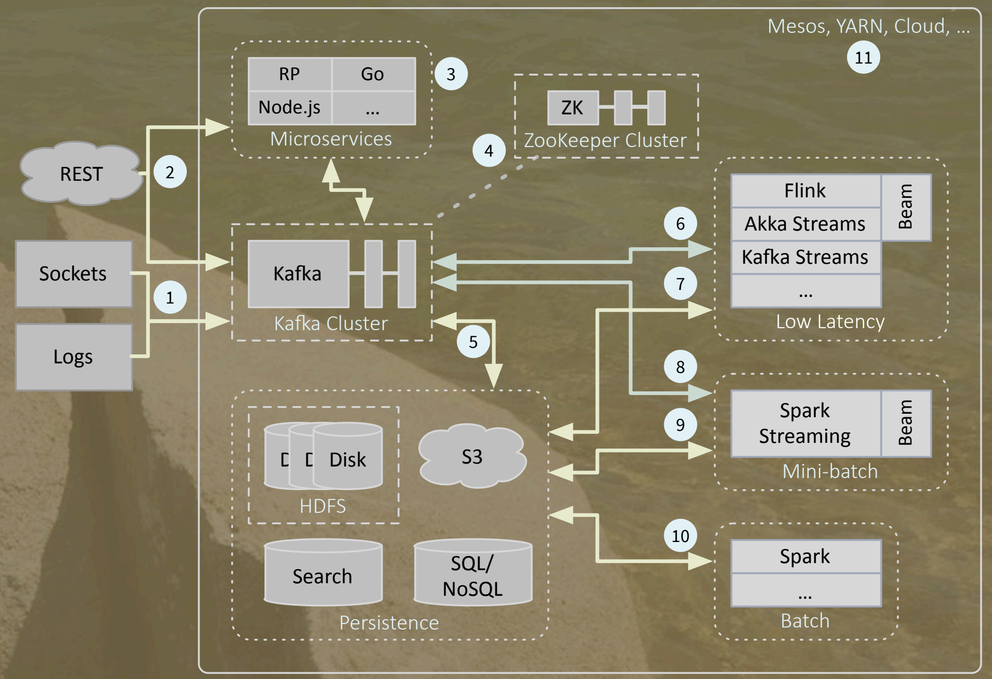
How are fast data architectures like …

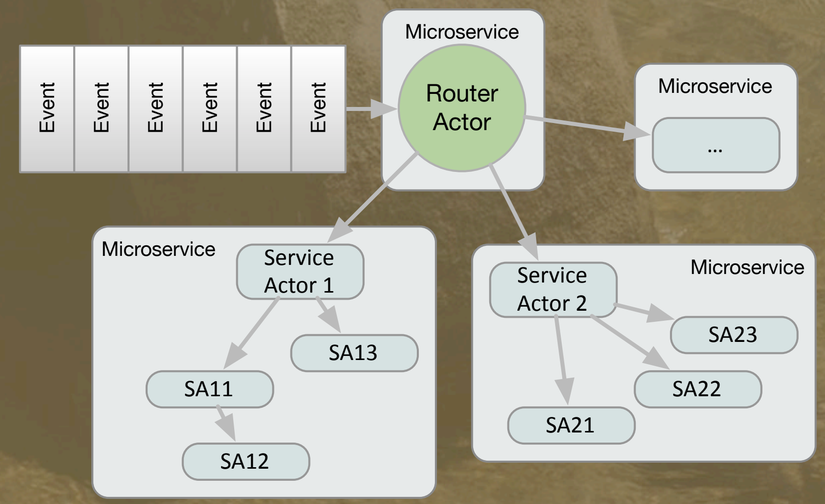
… like this?



… microservice architectures??

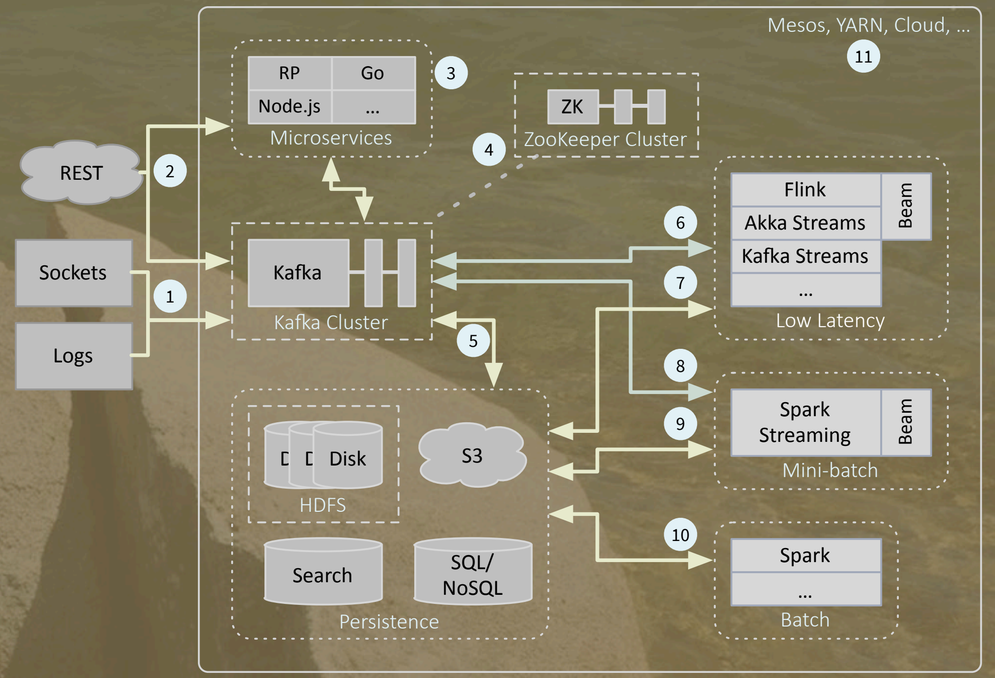
- A data app / microservice:
  - A single responsibility.
  - …

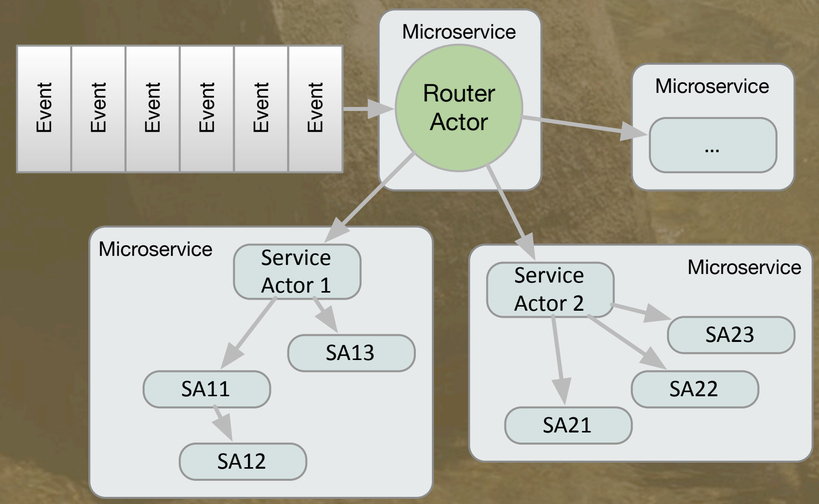




Each data app, streaming or batch, is typically doing one thing, like ETL this Kafka topic to Cassandra, or compute aggregates for a dashboard, etc. Similarly, services evolving now into microservices are also supposed to do one thing and communicate with other services for the “help” they need.

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

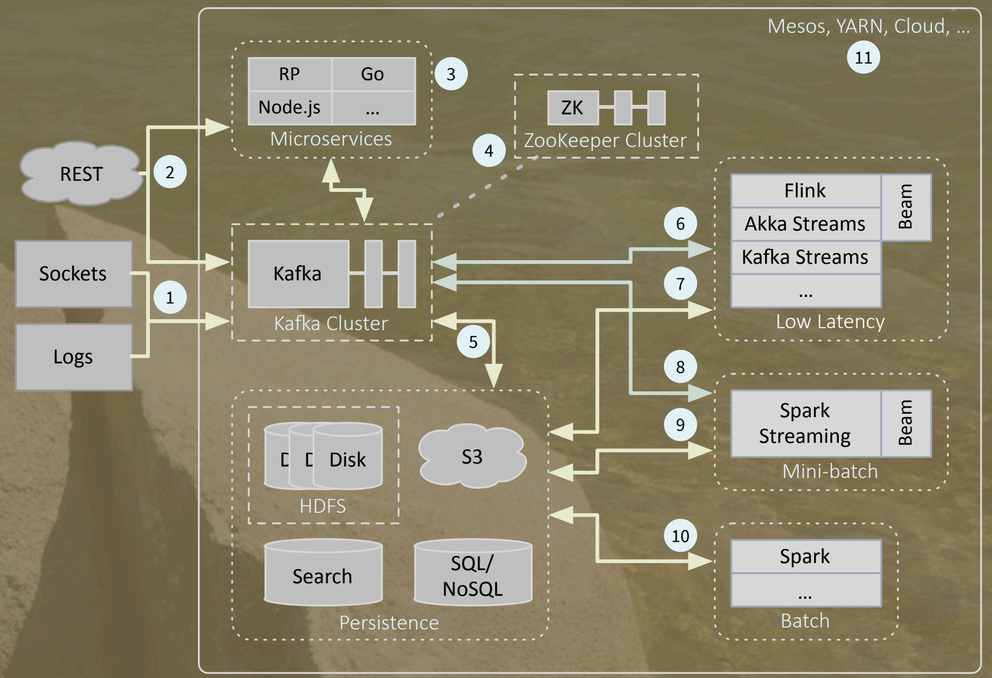


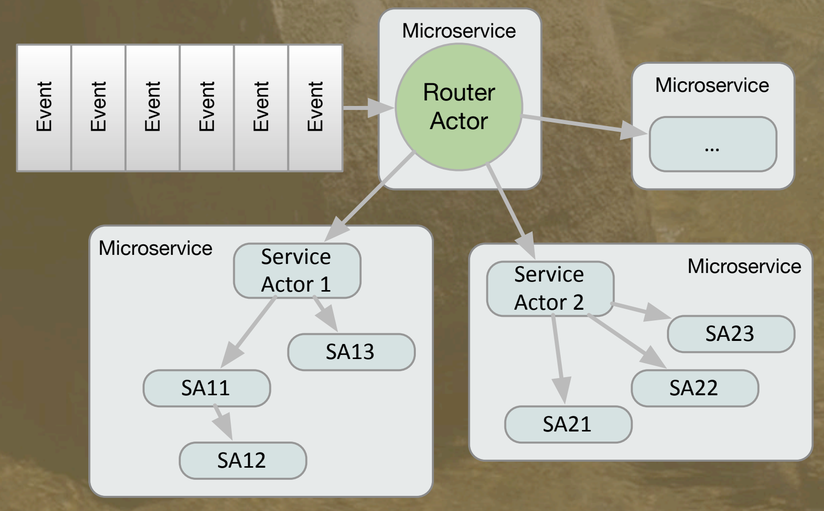


A streaming data app will have a never-ending stream of data to process. Similarly, requests for service will never stop coming to microservices.

- A data app/microservice:
  - A single responsibility.
  - The input never ends.
    - So, both must be

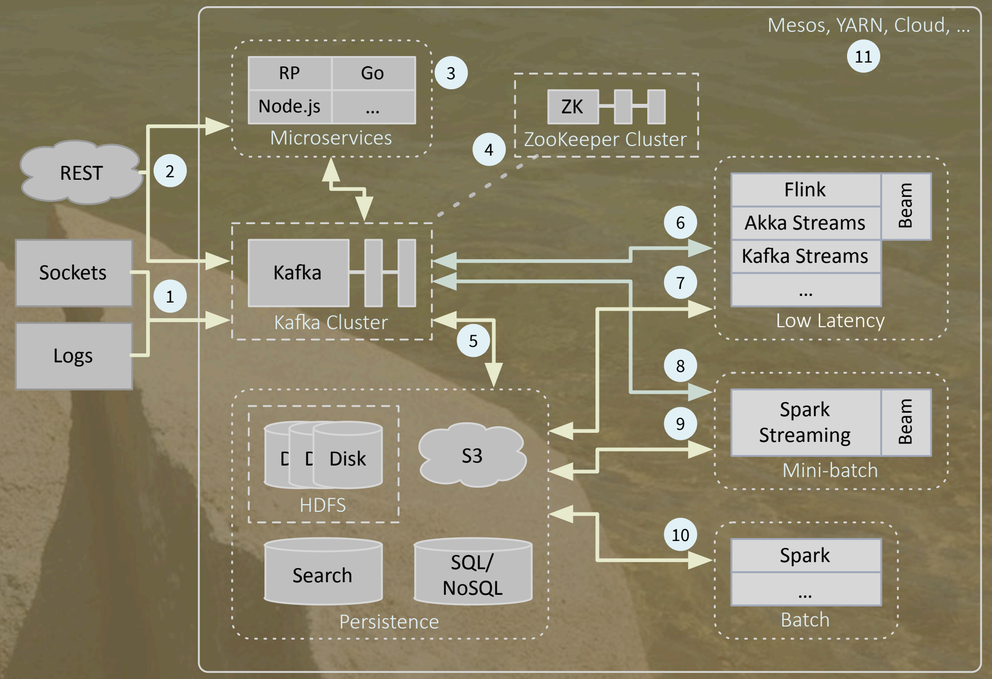
available, responsive, resilient, & scalable. I.e., reactive

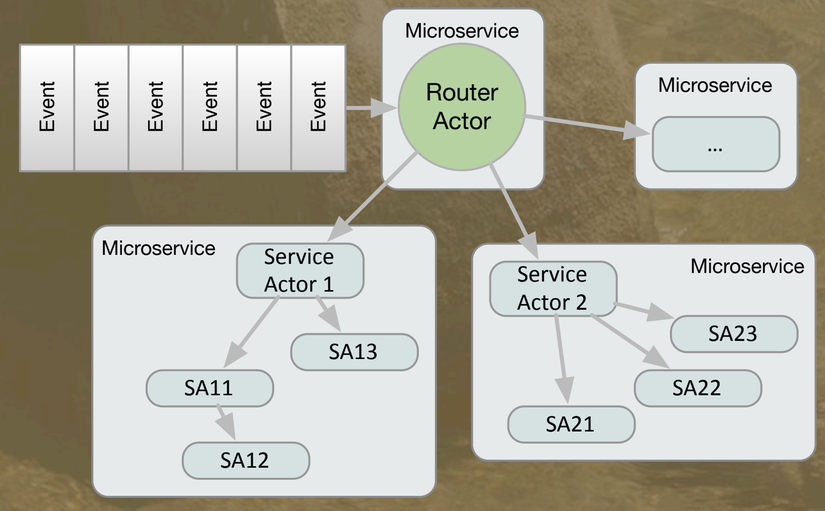




So, both kinds of systems have similar design problems, hence both should be implemented in similar ways.

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





Also, organizations that aren't particularly data focused, especially when they are new endeavors, often find that data grows to be a dominant concern, a sign of success! For example, Twitter started as a classic three-tier application, but now most of their infrastructure looks like a petroleum refinery.

The Recent Past

Services

Big Data

Some Overlap: Concerns, Architecture

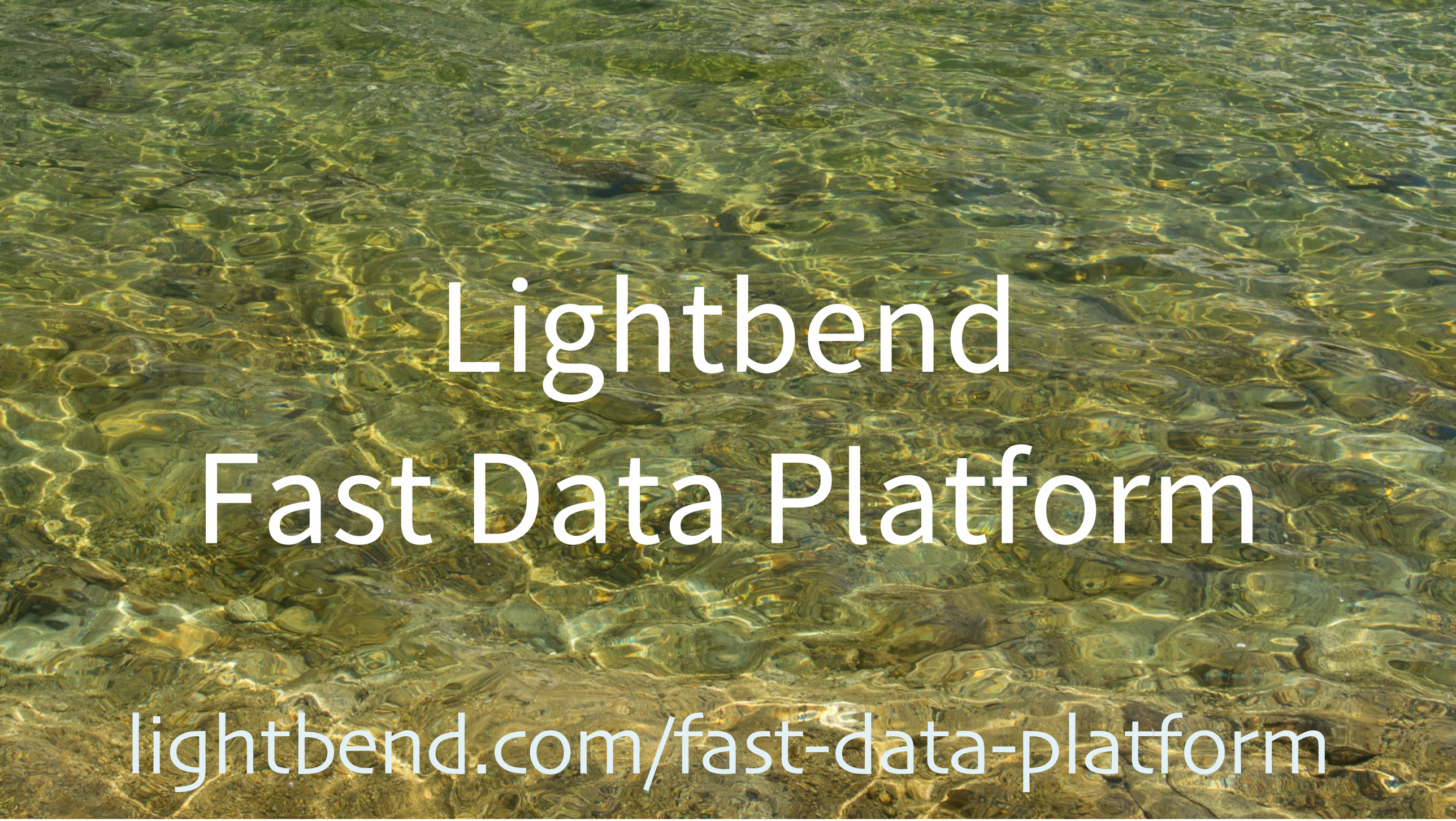
Until recently, people building “canonical” services for general-purpose IT apps have focused on high availability, scalability, resilience, etc. (i.e., the Reactive principles), recently moving to microservices to do this better.
The Big Data world has focused on data storage and cluster scalability, with less need to worry about the Reactive principles.
Of course these was some overlap, but they were different spheres…

The Present?

Microservices
& *Fast* Data

Much More Overlap

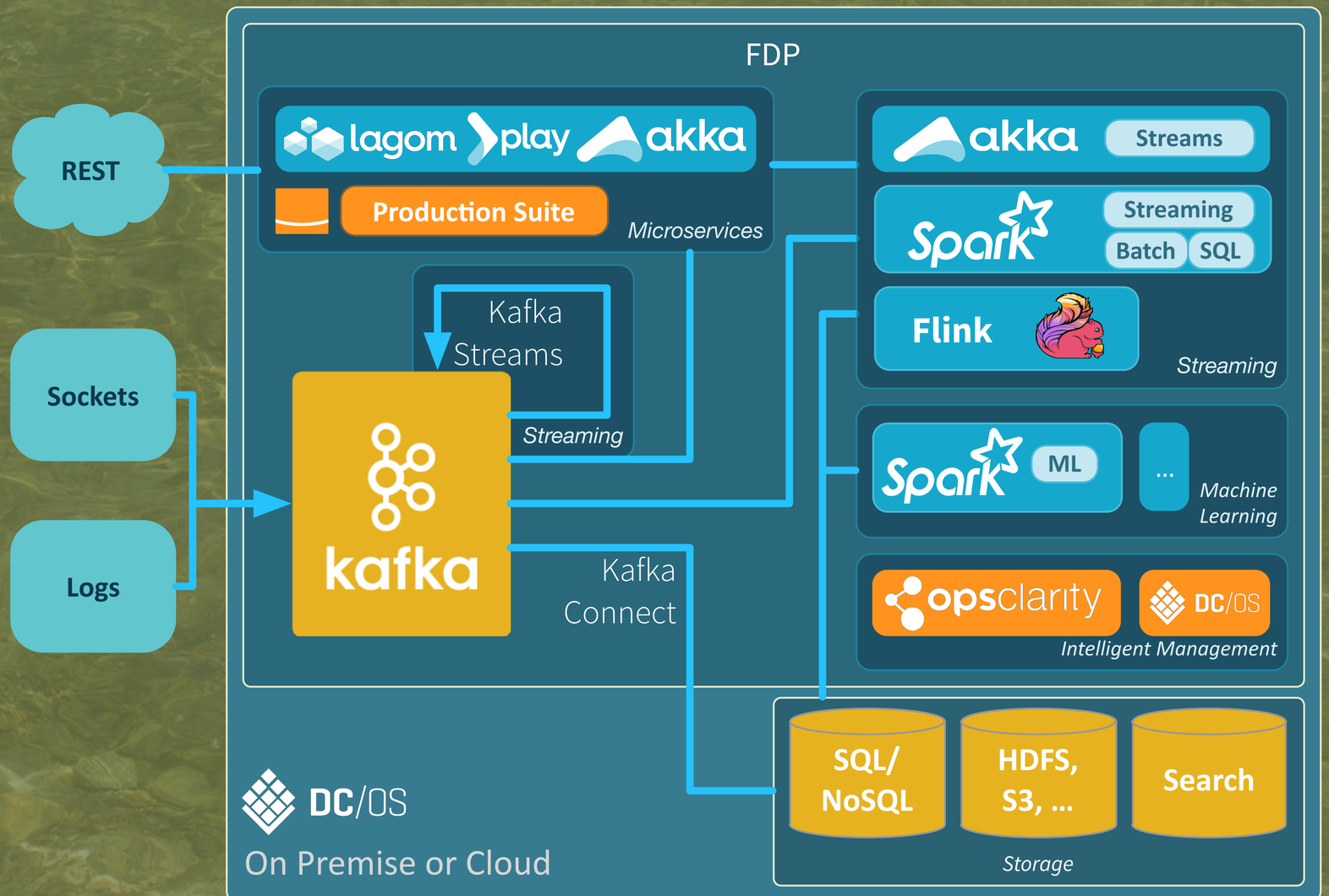
IT apps have been evolving towards finer-grained microservices and offline Big Data environments have been evolving towards online, stream processing or “Fast Data” environments. I’m going to argue that the architectures for these two spheres are converging towards the middle.



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.

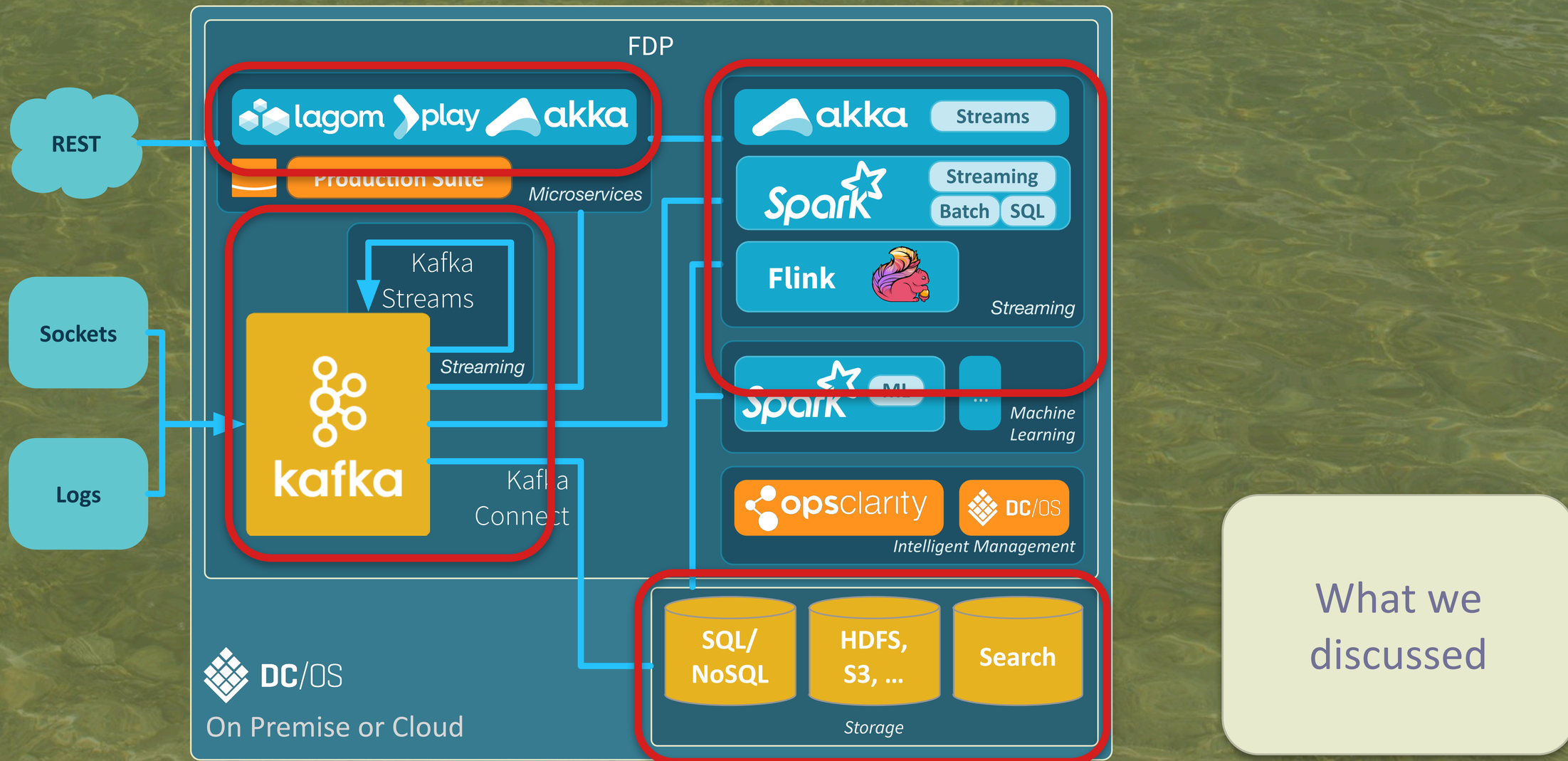
Lightbend Fast Data Platform V1.0



lightbend.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.

Lightbend Fast Data Platform V1.0

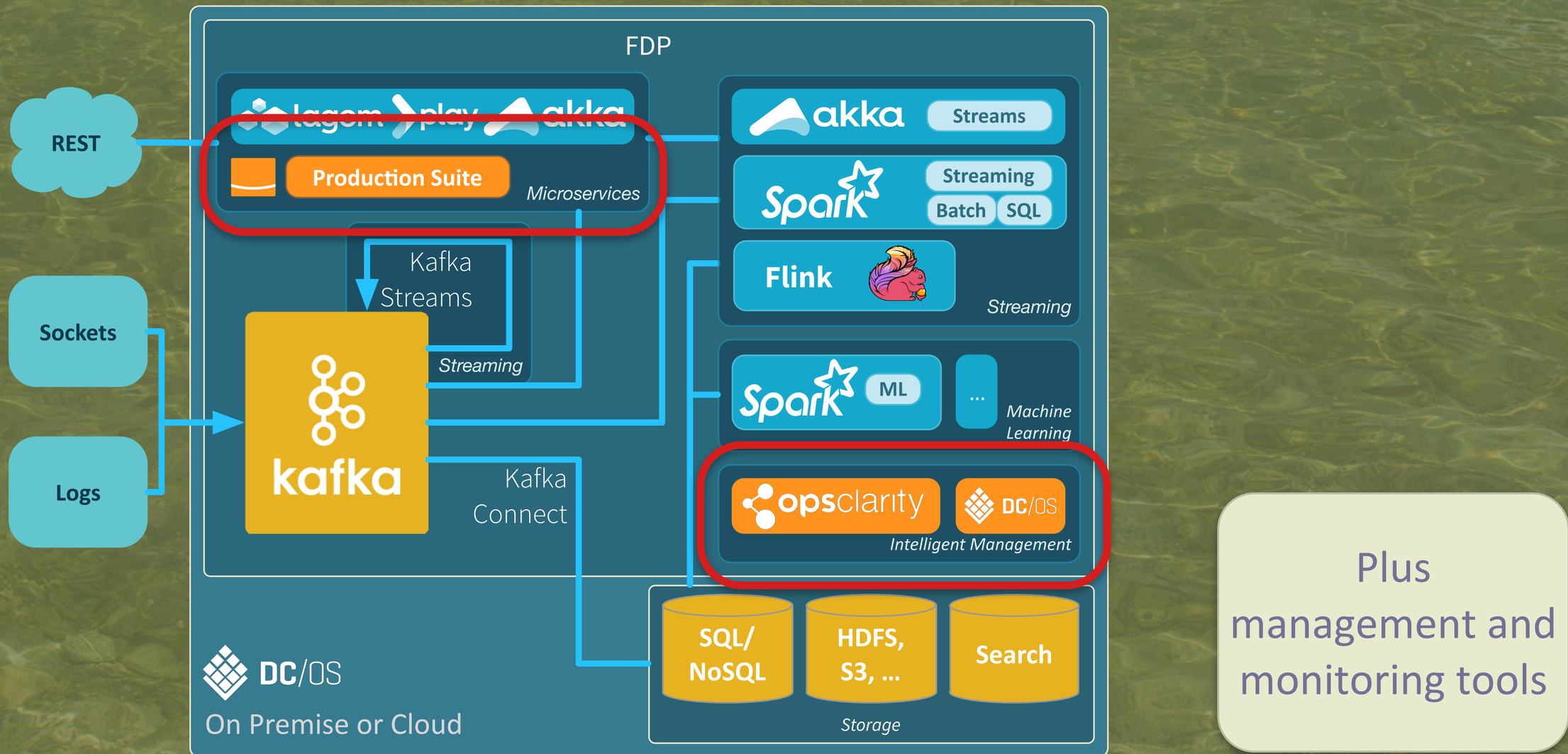


What we discussed

lightbend.com/fast-data-platform

Here's what we discussed so far.

Lightbend Fast Data Platform V1.0

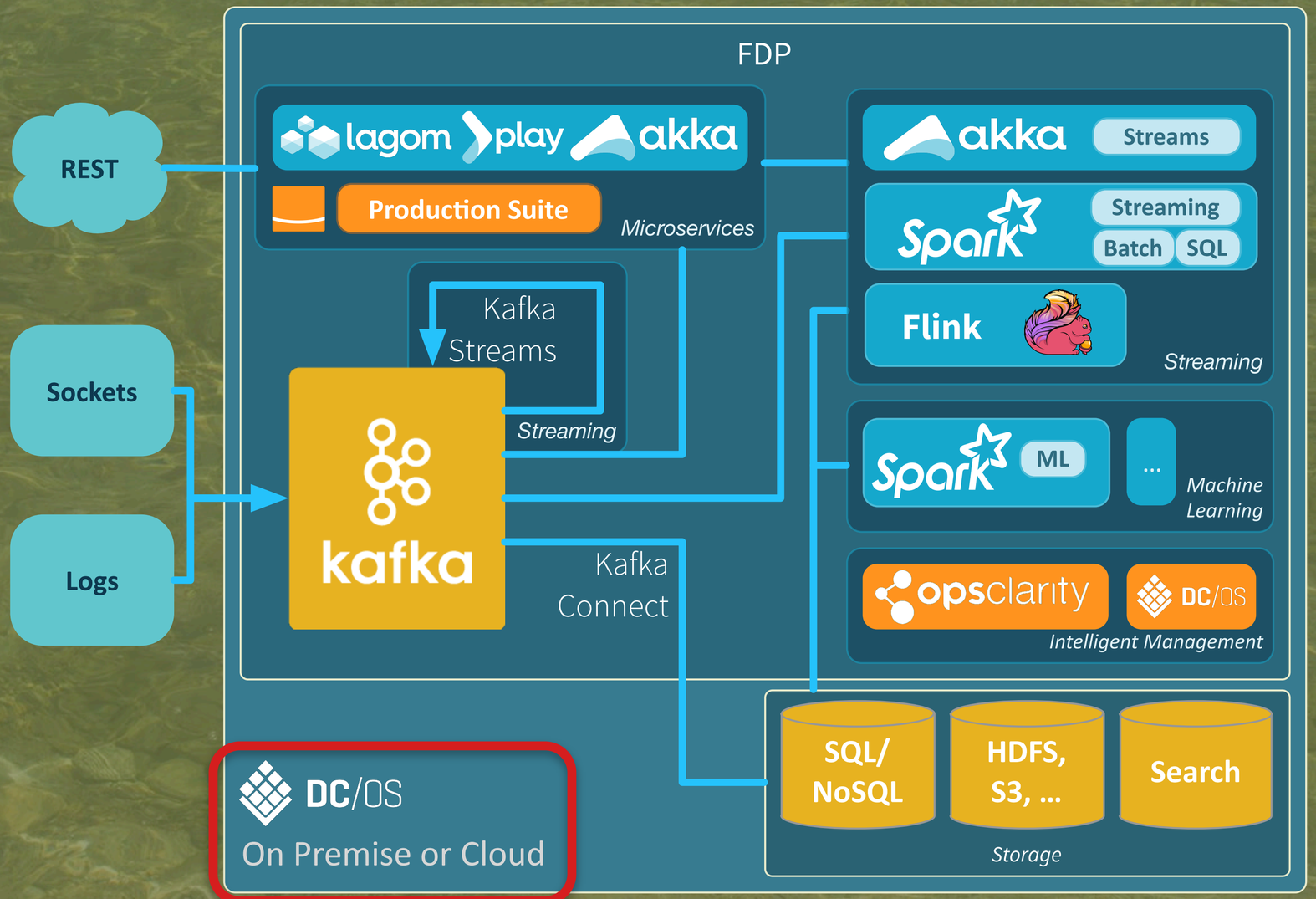


Plus management and monitoring tools

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.

Lightbend Fast Data Platform V1.0



DC/OS,
on premise or
in the cloud

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.

Lightbend

Thank you!

Dean Wampler, Ph.D.
dean@lightbend.com
@deanwampler
polyglotprogramming.com/talks

lightbend.com/fast-data-platform