Executive Briefing: What You Need to Know about Fast Data

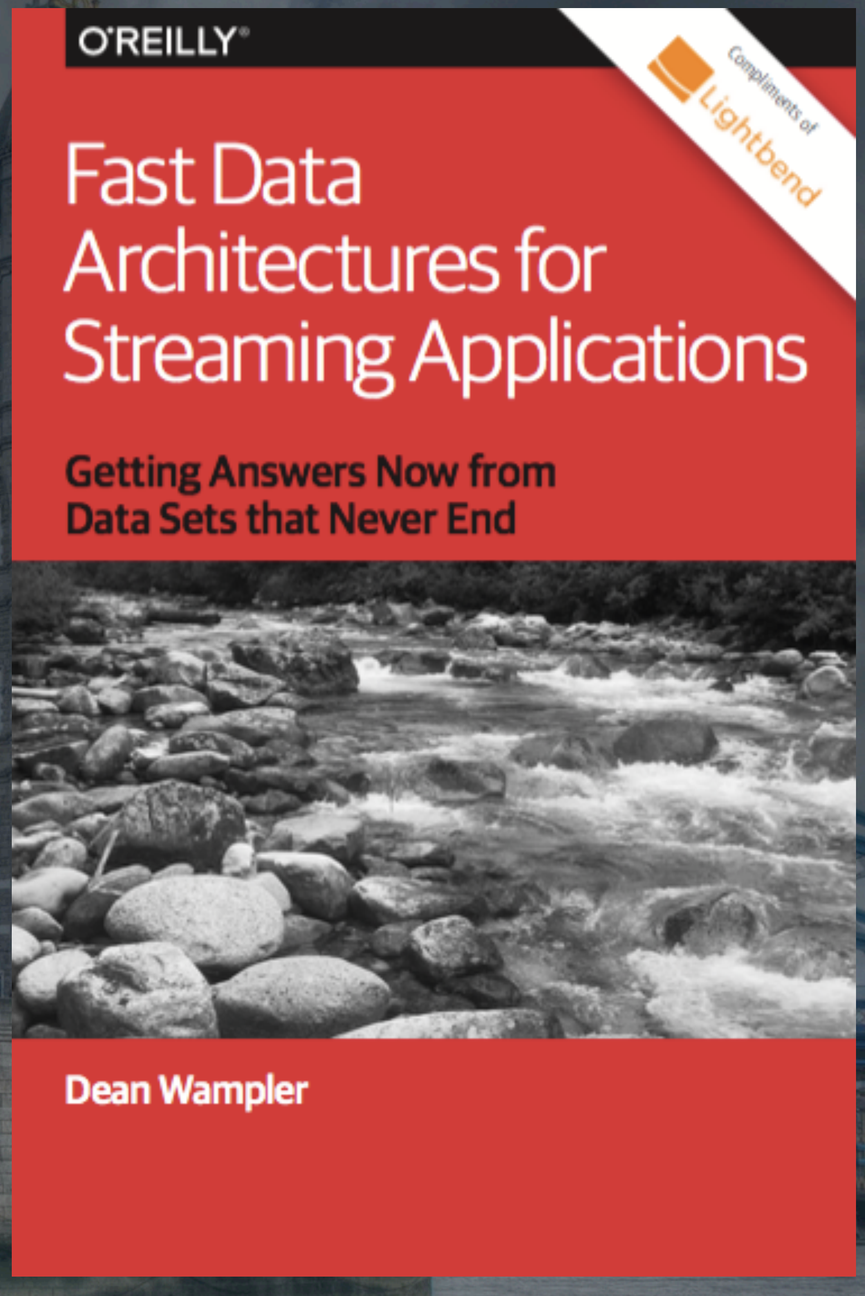
Dean Wampler, Ph.D.
dean@lightbend.com
@deanwampler

Photograph: Tower Bridge, London.
Last update: May 19, 2018

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

Based on this report

lightbend.com/fast-data-platform



My report, published late 2016, which describes the points in this talk in greater depth. It's more technically oriented than this talk, so give it to your engineers ;)

What We'll Discuss

Outline

- Why streaming? Why now?
- How should I choose technologies?
- How will this impact the rest of my organization?

What We'll Discuss

Outline

Why Streaming?

Why should we bother in the first place.

Why Streaming?

- New opportunities that require streaming
- Upgrading batch applications for competitive advantage

Let’s discuss two classes of applications, opportunities, and business drivers. One is fairly obvious; some new things have low latency requirements, so you have to act on data as it arrives. Others have been successful with traditional batch approaches, but it’s a competitive advantage to accelerate them through streaming.

Fast Data Use Cases

Predictive Analytics	Real-time Personalization	Real-time Financial Processes	IoT	Legacy Modernization
Apply ML models to large volumes of device data to pre-empt failures / outages	Real-time marketing based on behavior, location, inventory levels, product promotions, etc.	Drive better business outcomes through real-time risk, fraud detection, compliance, audit, governance, etc.	Real-time consumer and industrial Device and Supply Chain management at scale	Accelerate decision making processes and optimize infrastructure costs by moving from batch to streaming
Hewlett Packard Enterprise	Royal Caribbean International	Capital One	Starbucks Coffee	IQVIA

More at: https://www.lightbend.com/customers

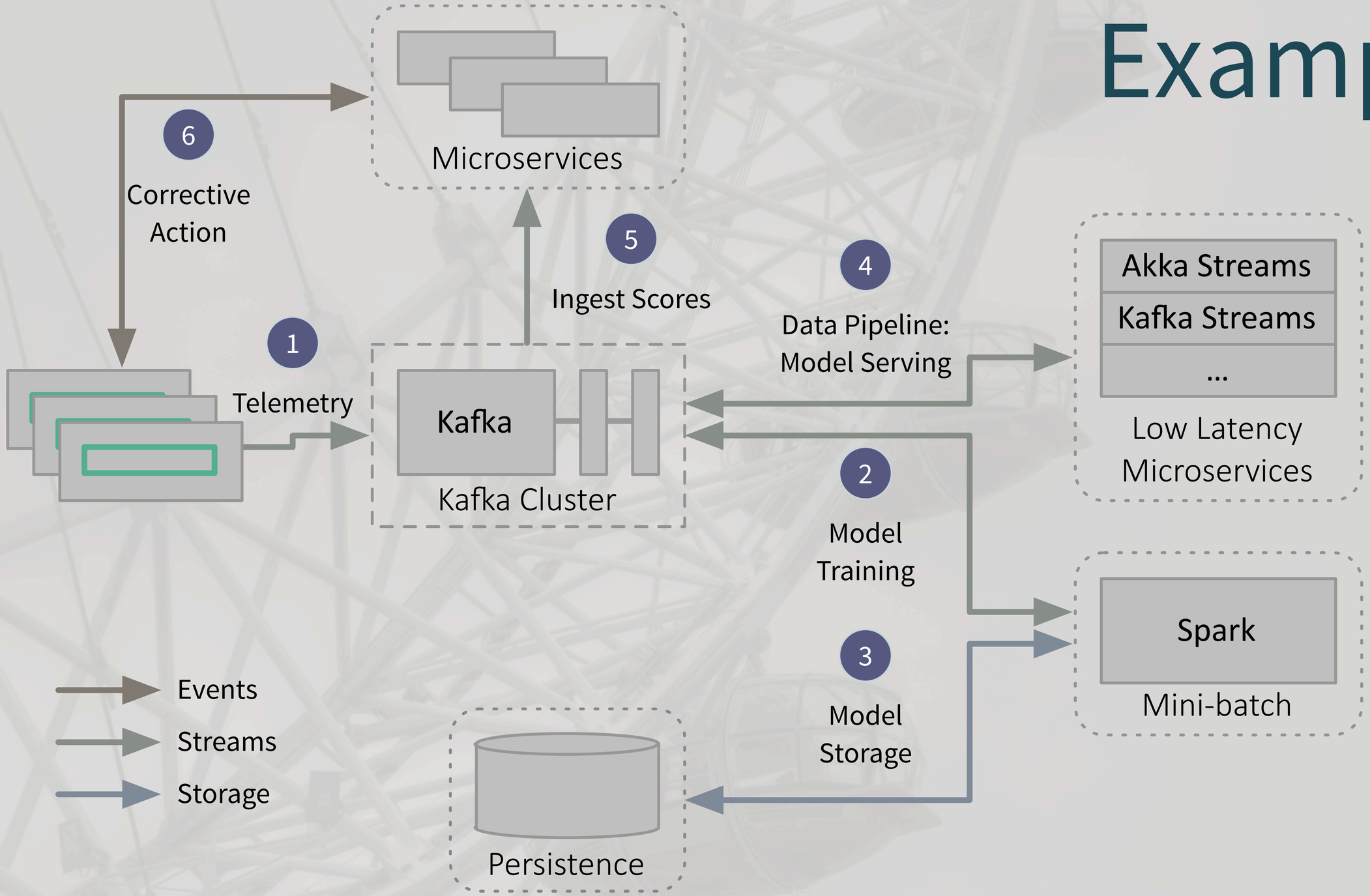
Real use cases from Lightbend Customers. For reasons that should become clear later, not all of these projects think of themselves as “fast data” application. Some are “just microservices”.

Predictive Analytics

Hewlett Packard Enterprise

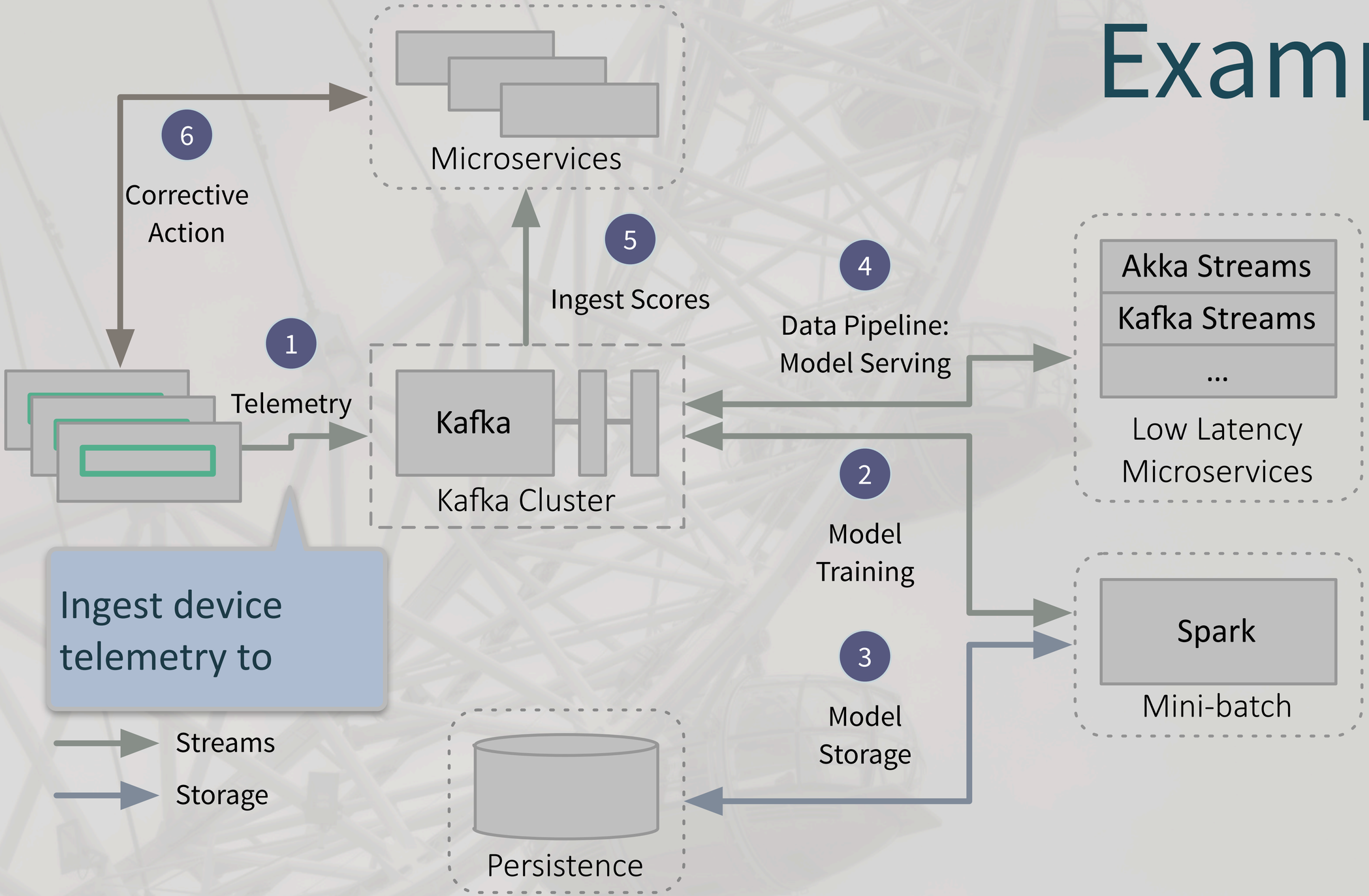
- ML models applied to device telemetry to detect anomalies
- Preemptive maintenance prevents user impacts from potential failures

Example



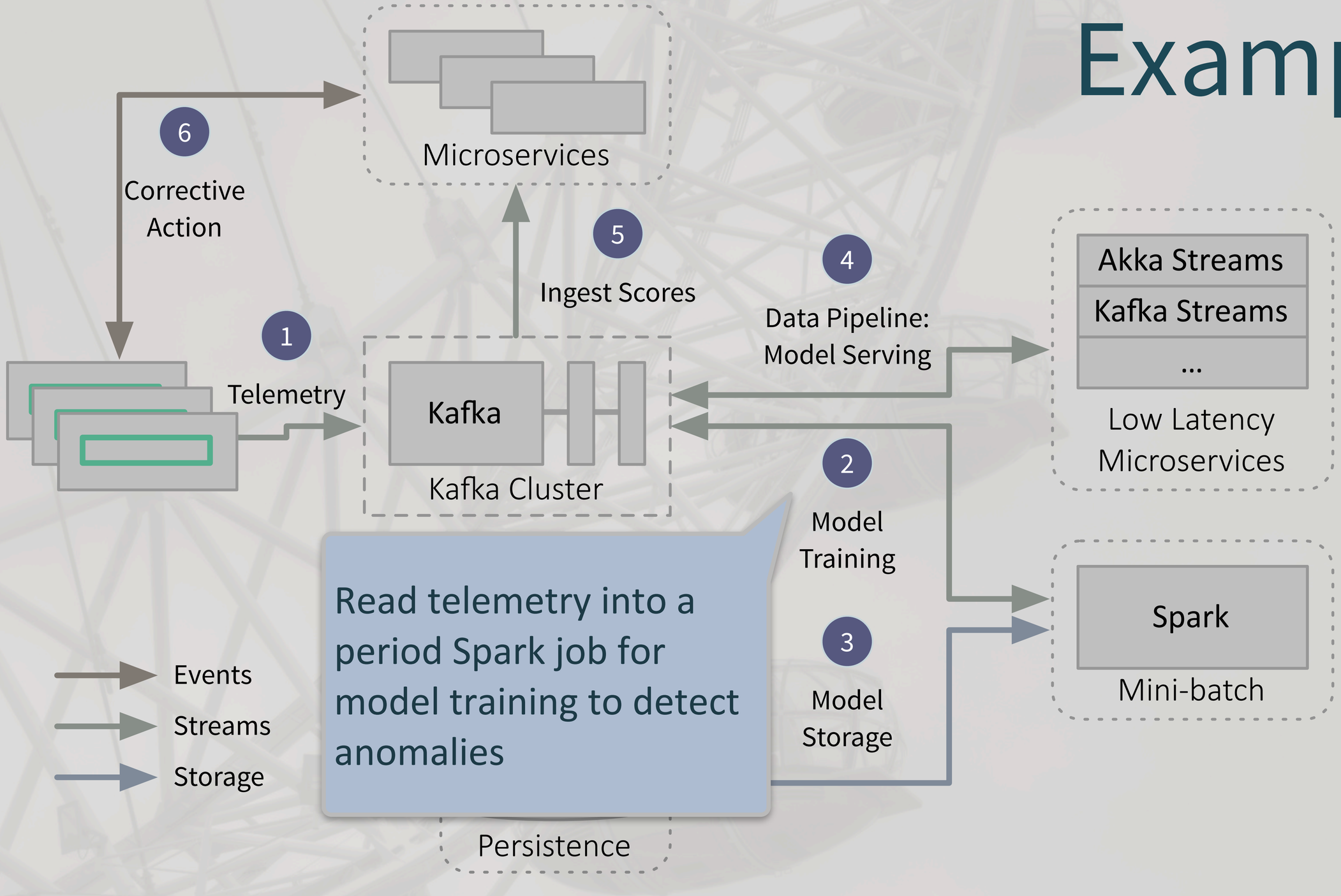
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



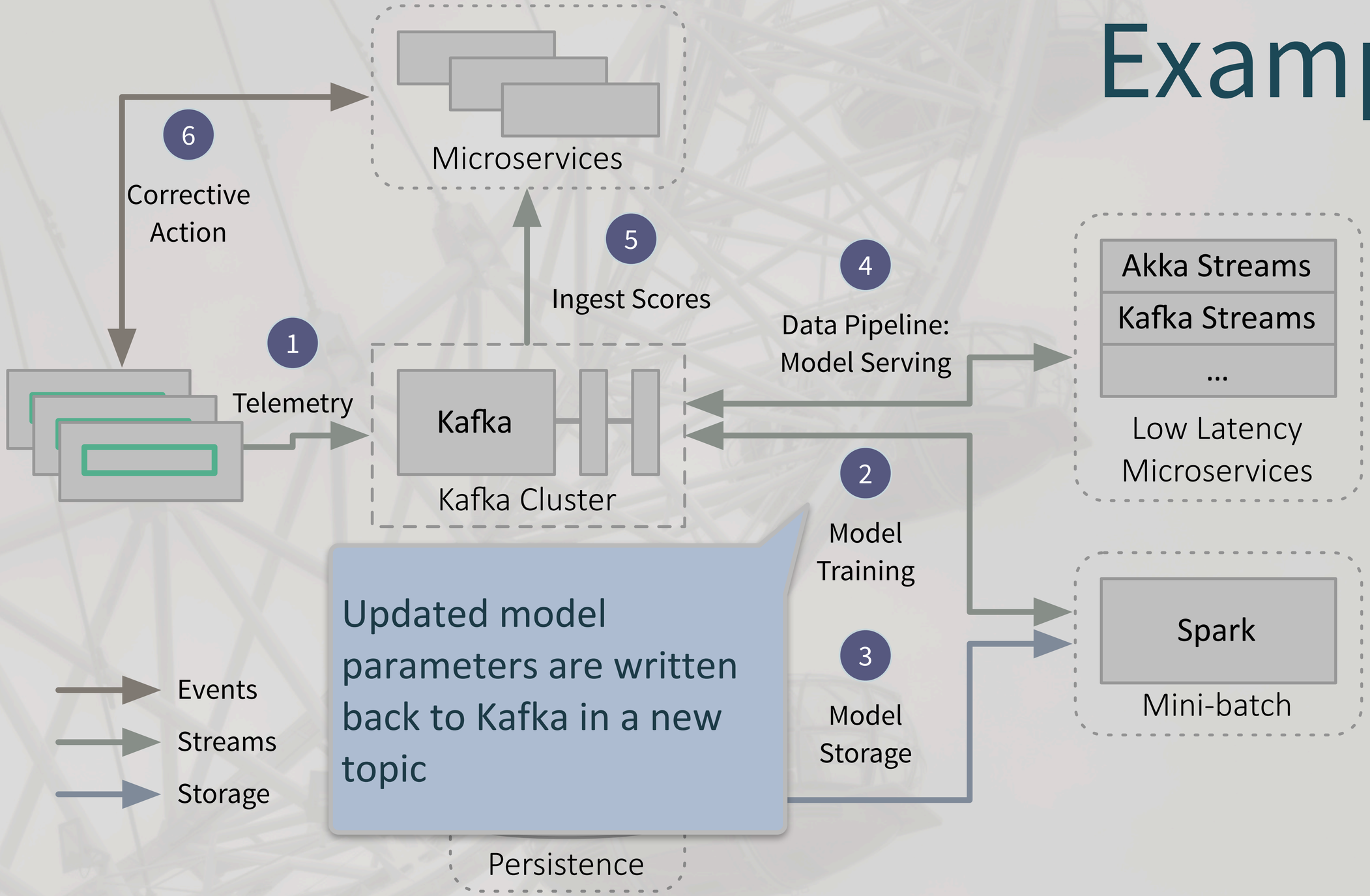
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



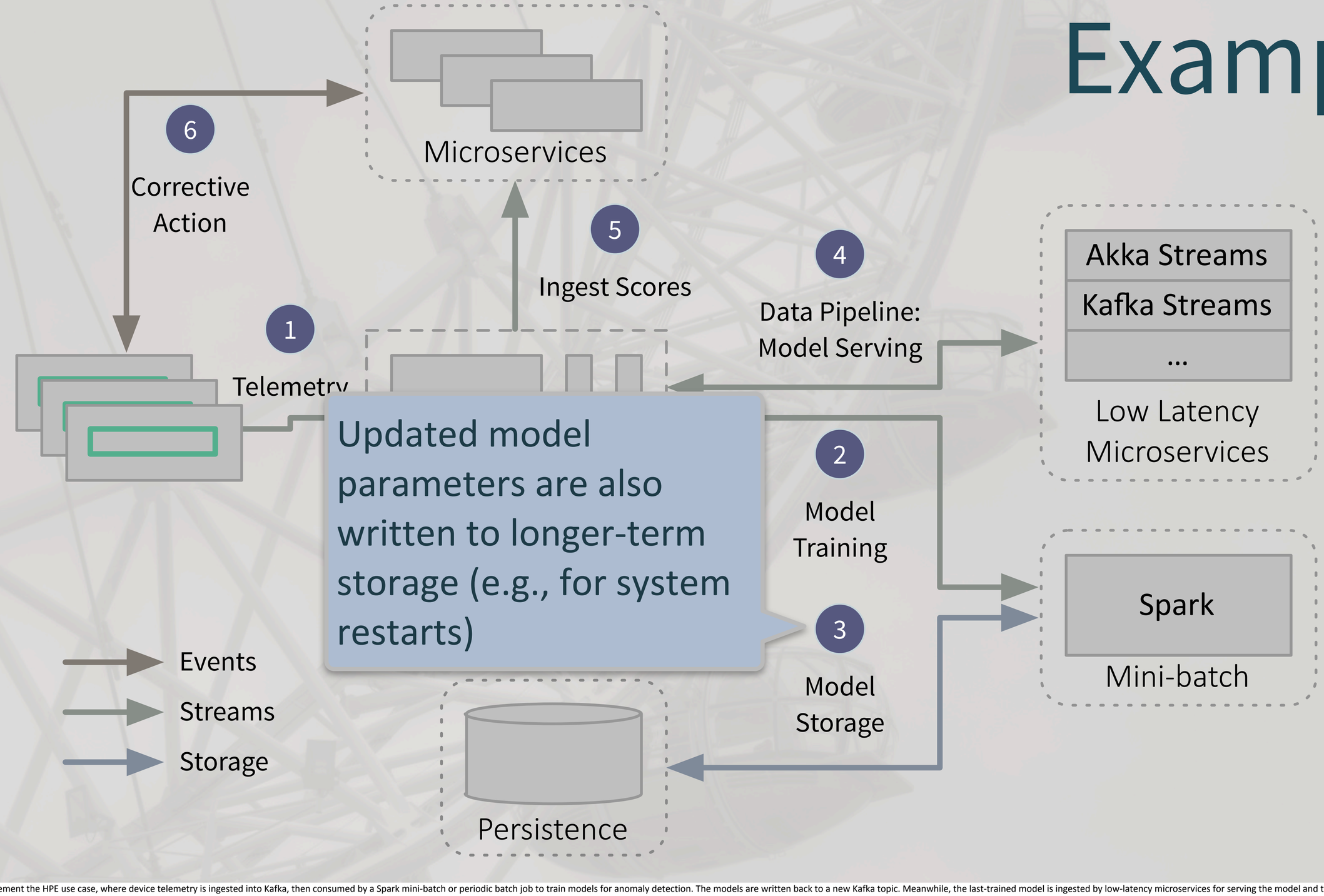
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



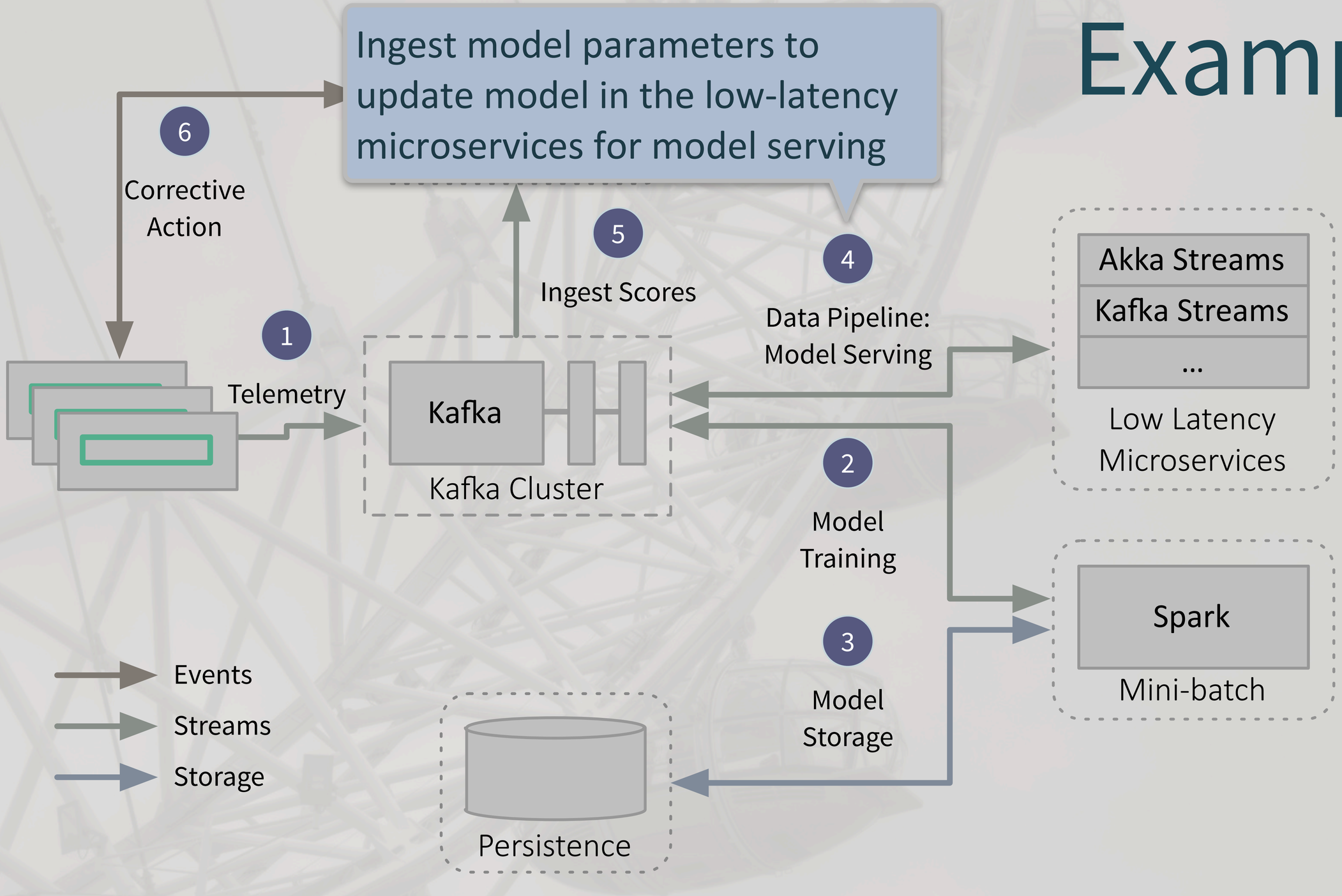
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



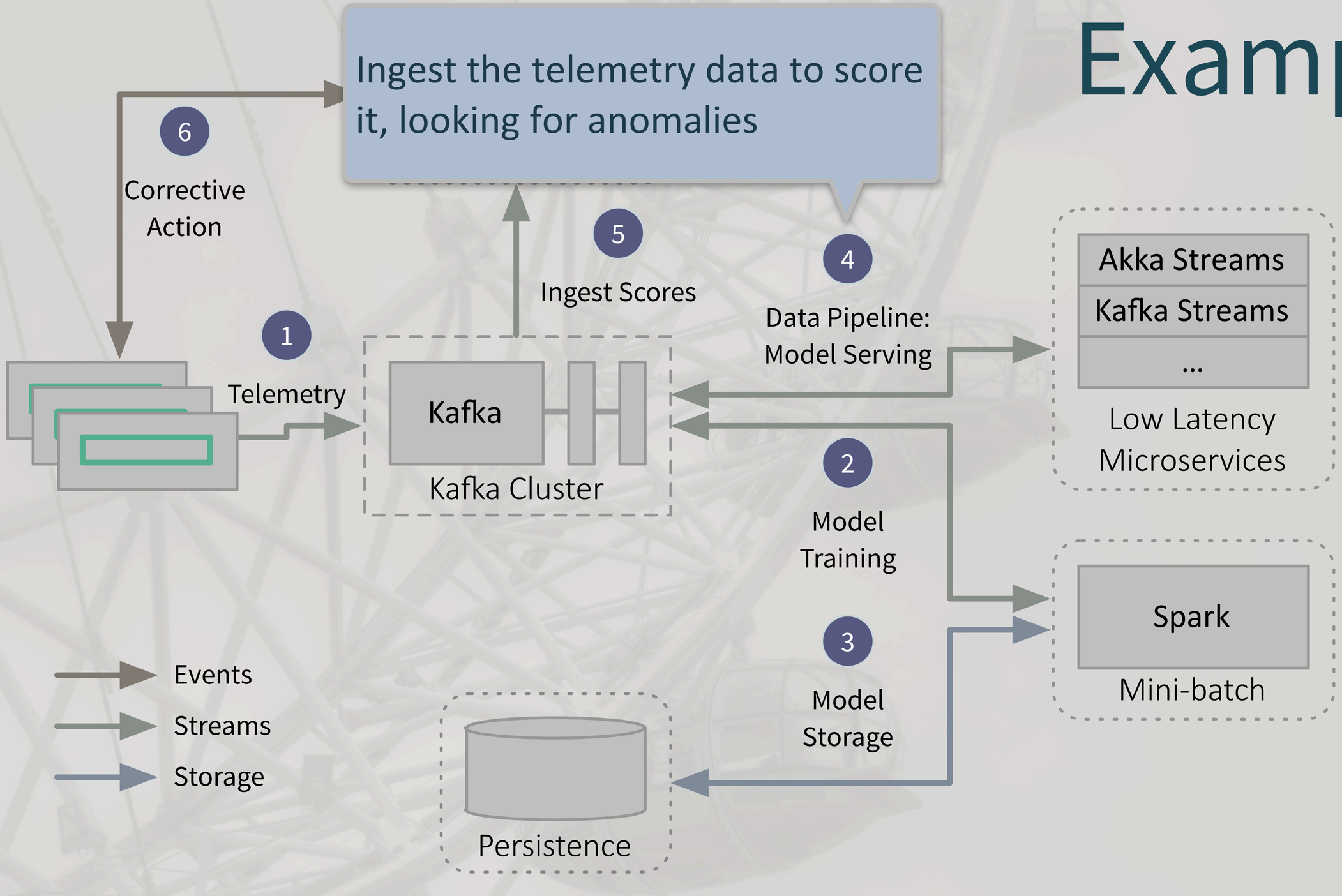
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



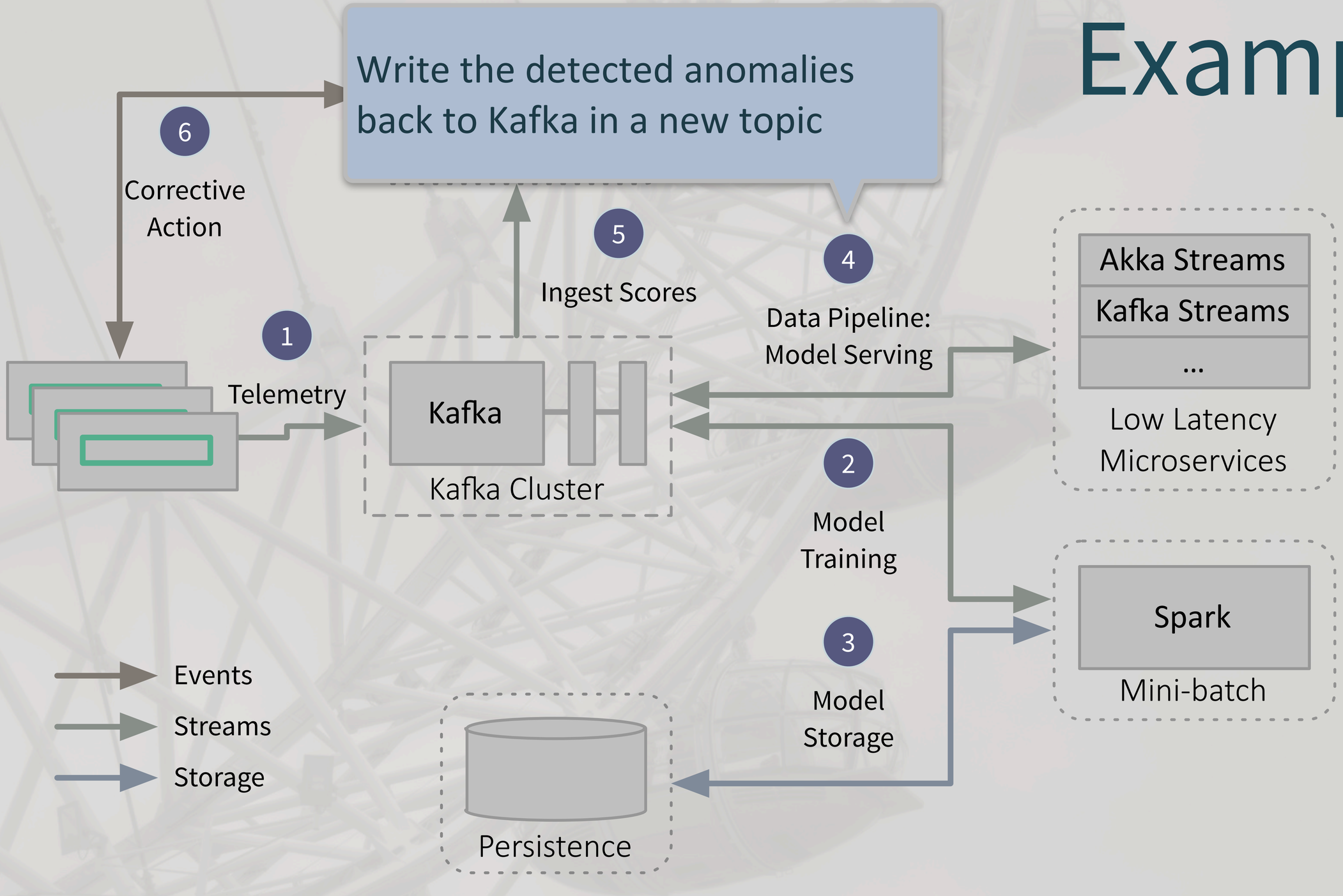
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



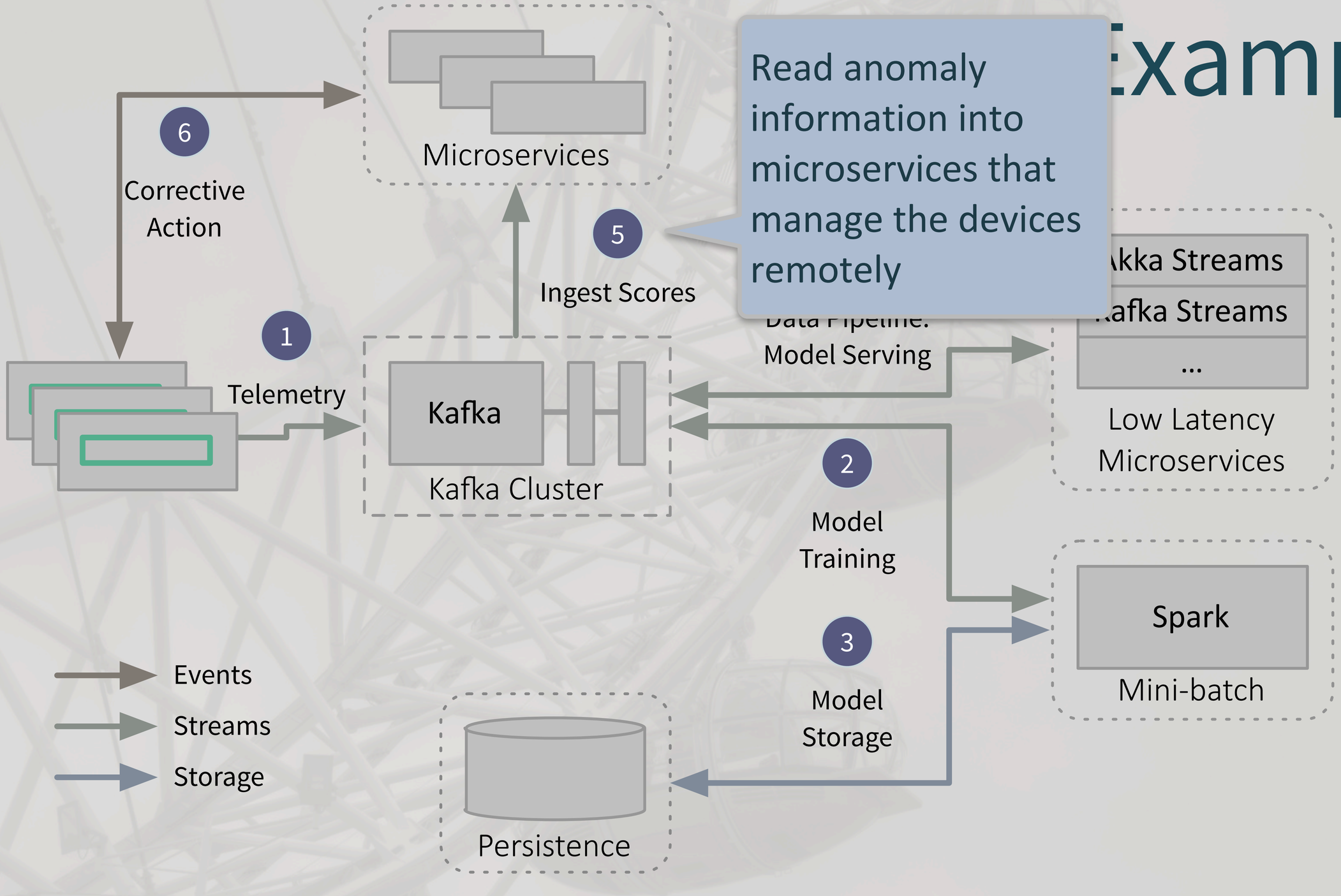
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



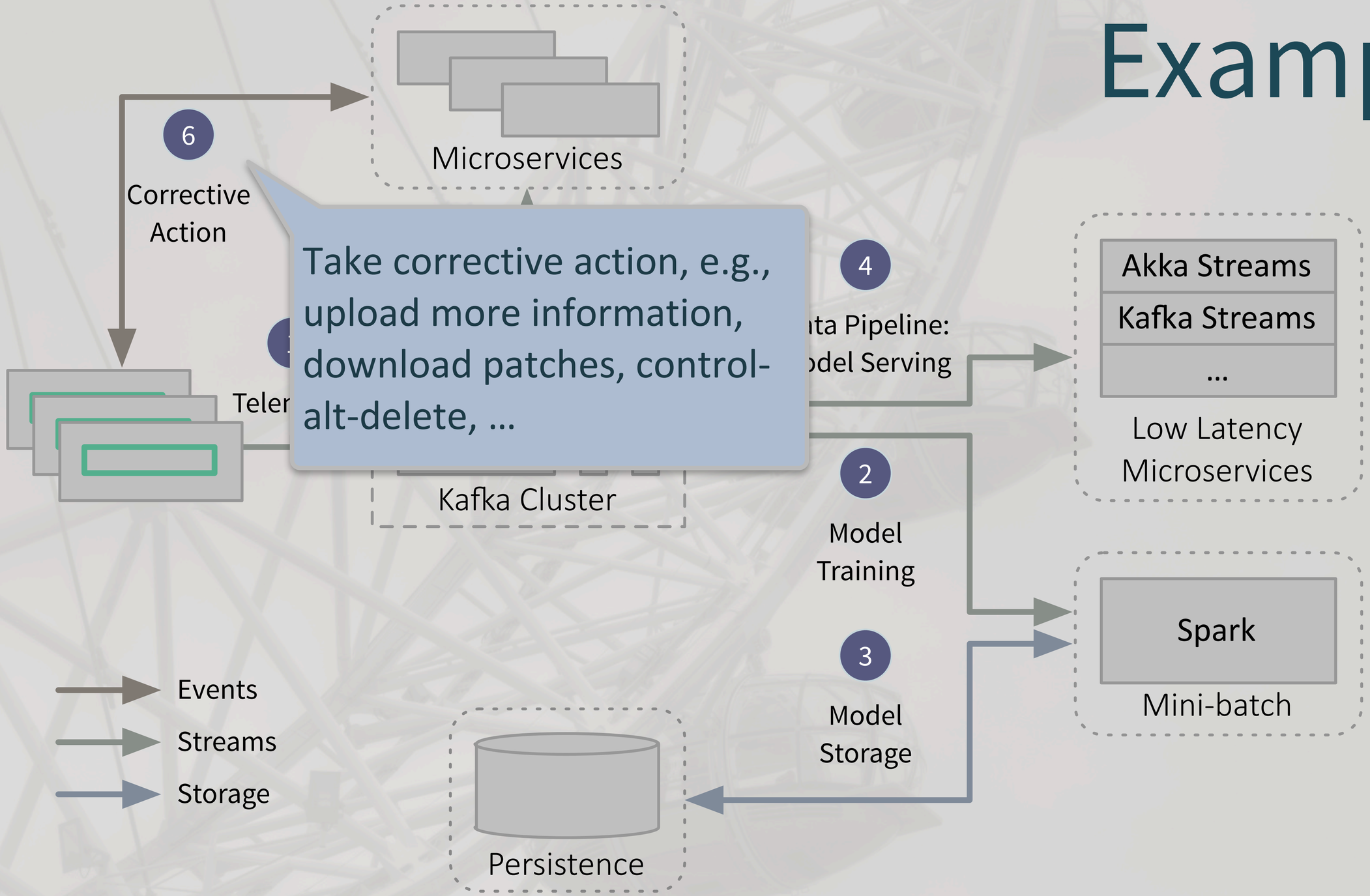
Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Example



Example of how you might implement the HPE use case, where device telemetry is ingested into Kafka, then consumed by a Spark mini-batch or periodic batch job to train models for anomaly detection. The models are written back to a new Kafka topic. Meanwhile, the last-trained model is ingested by low-latency microservices for serving the model and the data is also ingested here for scoring. Scores for anomalies are written to a new topic where other microservices that manage the devices take corrective action.

Real-time Personalization

- Model users to provide real-time marketing based on behavior, location, inventory levels, product promotions, etc.

In fact, all these use cases boil down to very similar technical challenges, so I won't show additional diagrams for each of them.

Real-time Financial

- Drive better business outcomes through real-time risk, fraud detection, compliance, audit, governance, etc.

Internet of Things

- Real-time consumer and industrial device and supply chain management at scale

Legacy Modernization

- Accelerate decision making processes and optimize infrastructure costs by moving from batch to streaming
- Hadoop replacement

And One More… Hard Real Time

SPACEX

One more we didn't mention, but for completeness… there is "hard" real-time, like flight control software. This is beyond our scope here, because it usually involves custom hardware, real-time operating systems, and software written in C/C++, all of which squeeze out as much latency and overhead as possible. However, if you do high-frequency trading, then this is relevant to you ;)

photo and logo: http://www.spacex.com/

And One More… Hard Real Time

SPACEX

- Flight control software (and high-frequency training) require custom hardware, real-time operating systems, C/C++ software, etc.

Technology Choices

Now let’s explore technology choices (ignoring the hard real-time case…)

Photo: Millenium Bridge with St. Pauls in the background

Technology Choices

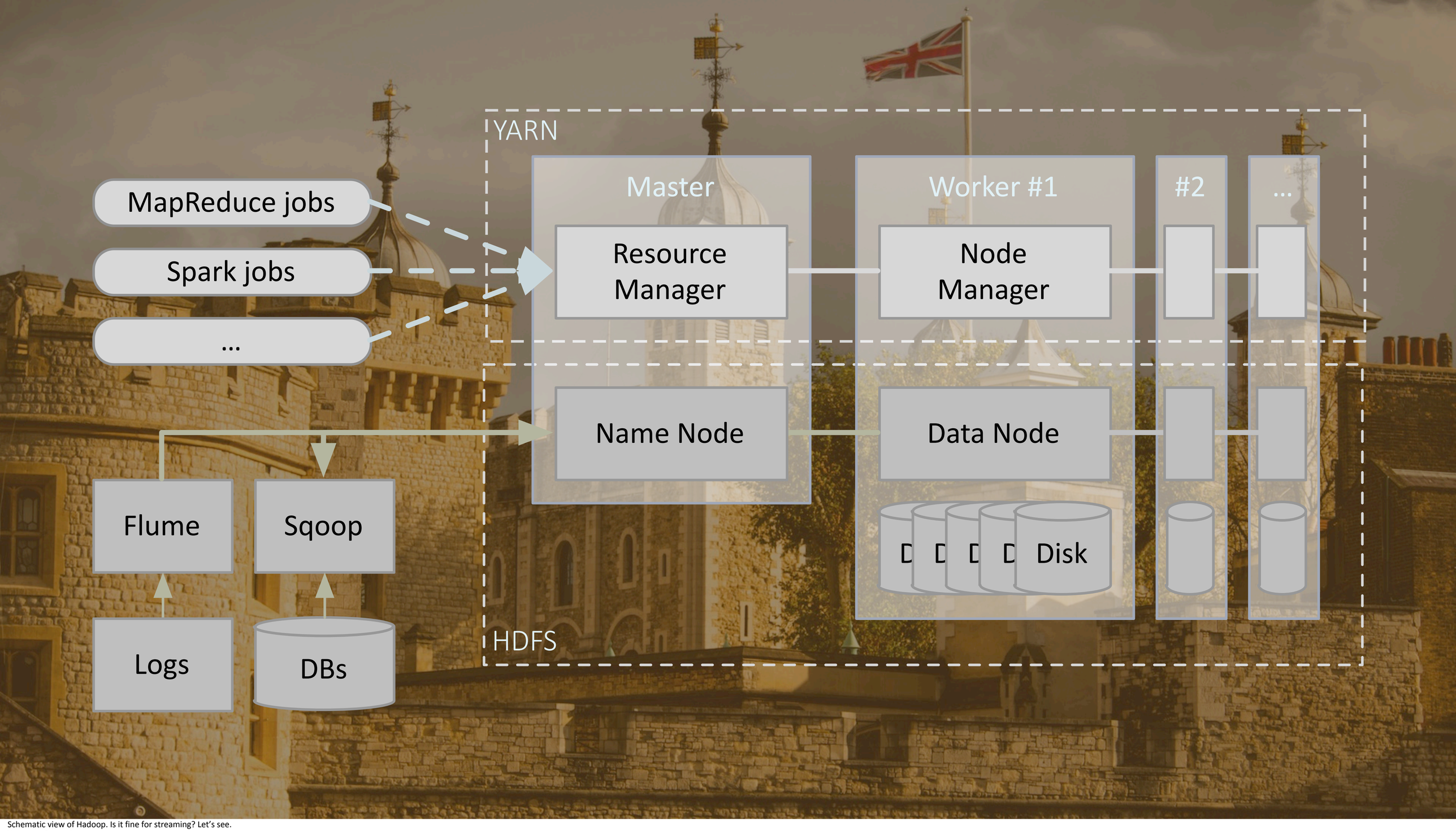
- More than "tweaking" Hadoop…
- New architectures that merge data processing with microservices

Key points of this section.

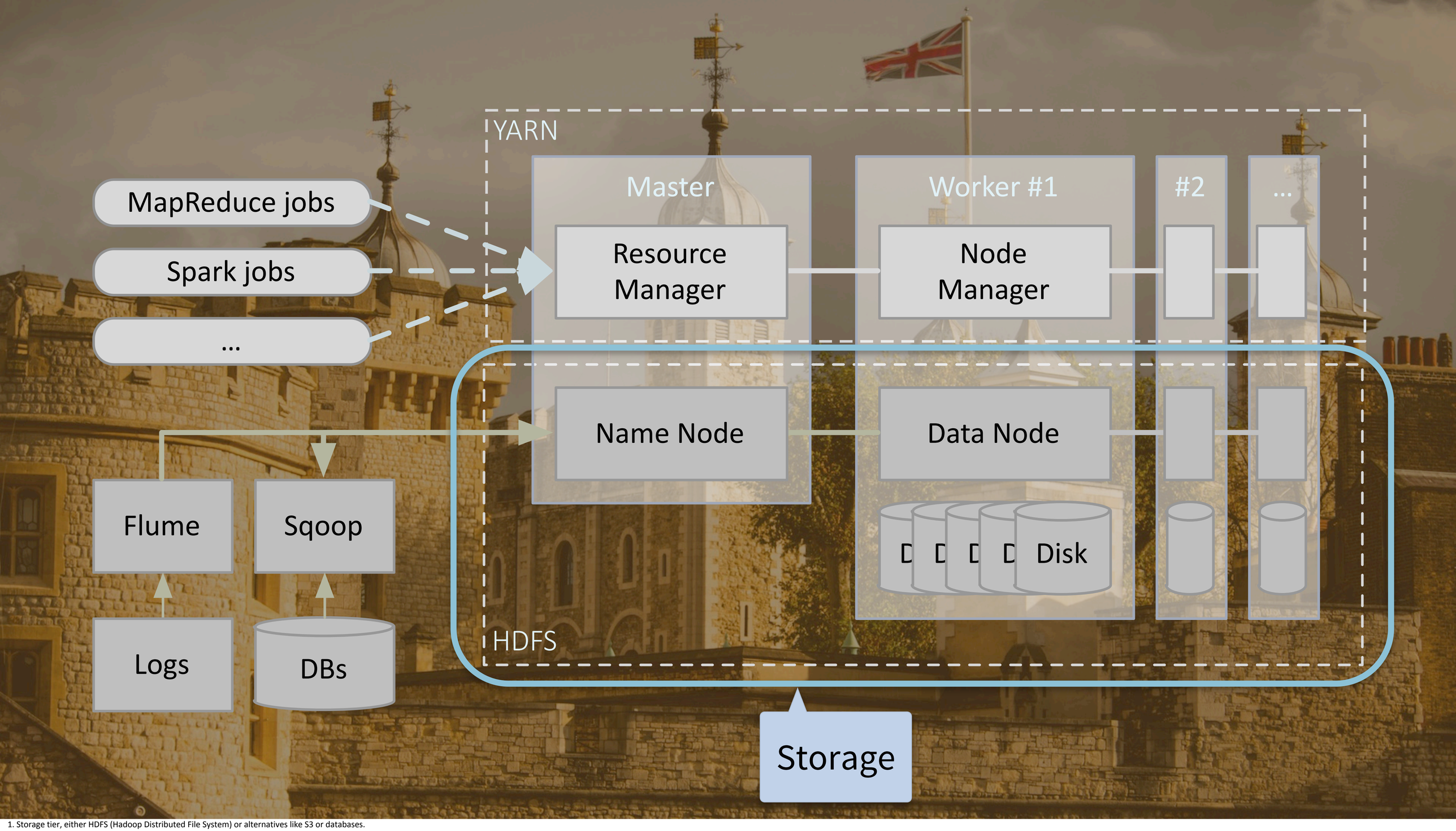
Hadoop

Photo: Detail of the Tower of London

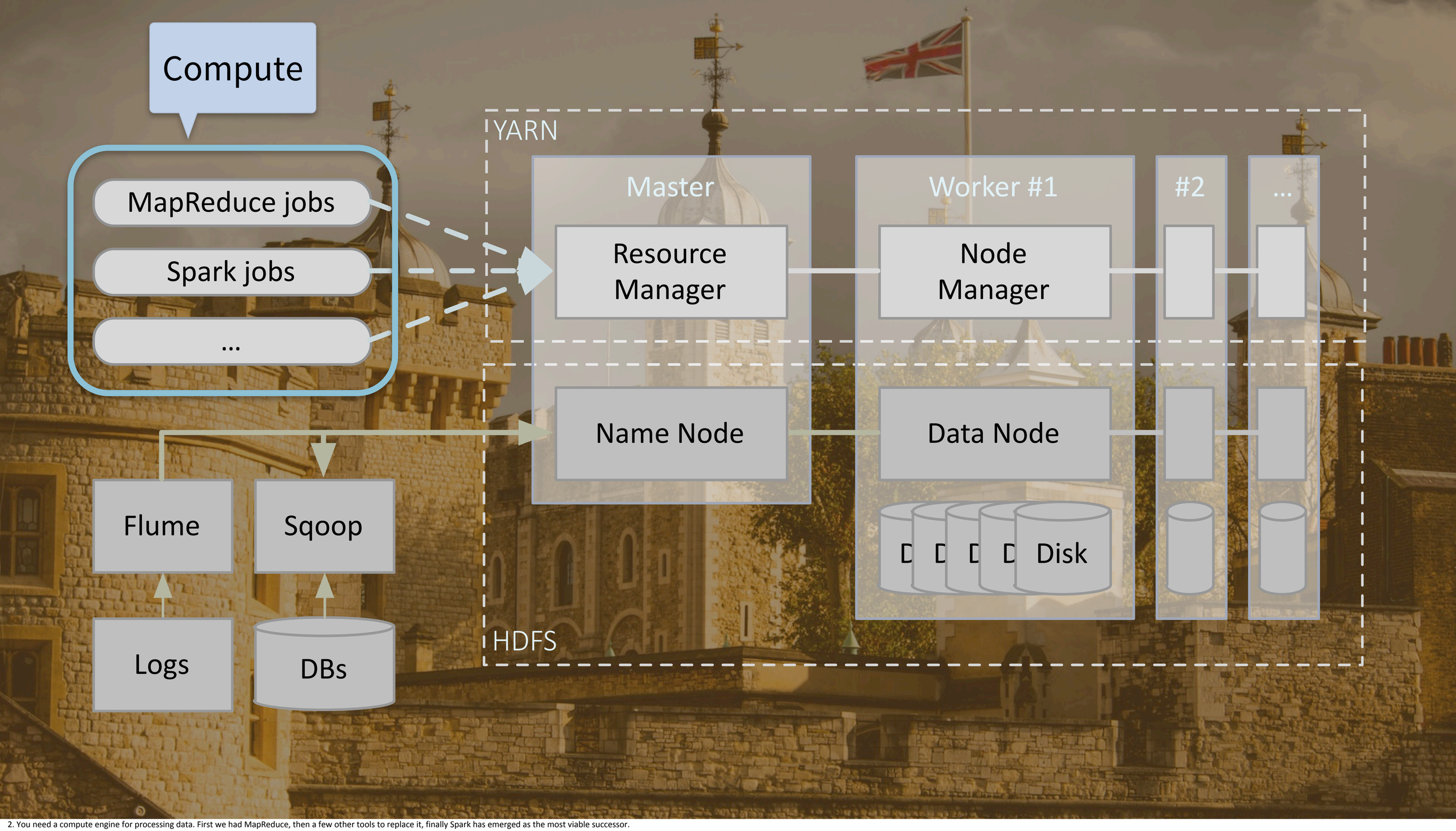
- Data warehouse replacement
- Historical analysis
- Interactive exploration
- Offline training of machine learning models
- …



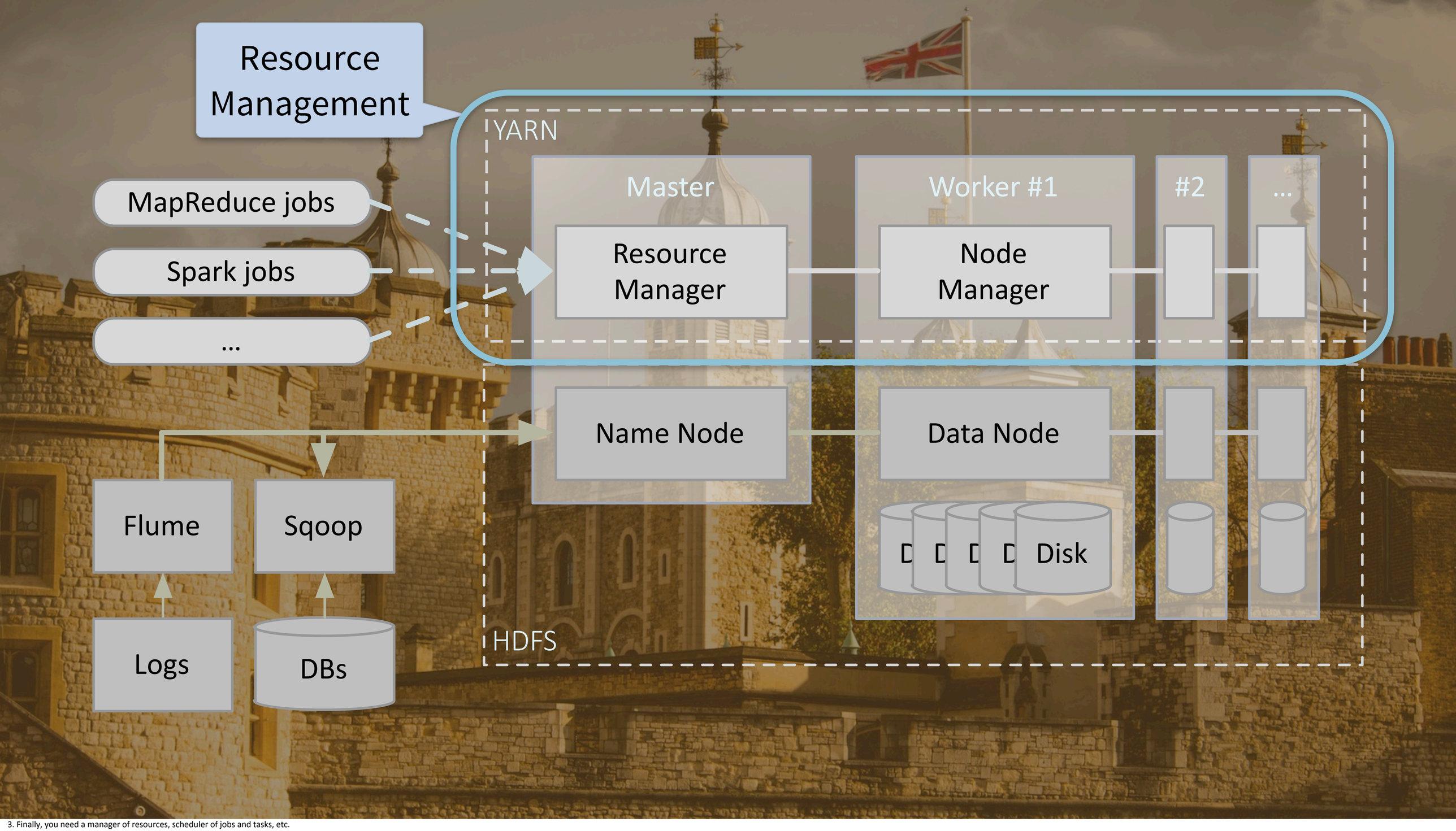
Schematic view of Hadoop. Is it fine for streaming? Let's see.
You need 3 core features…



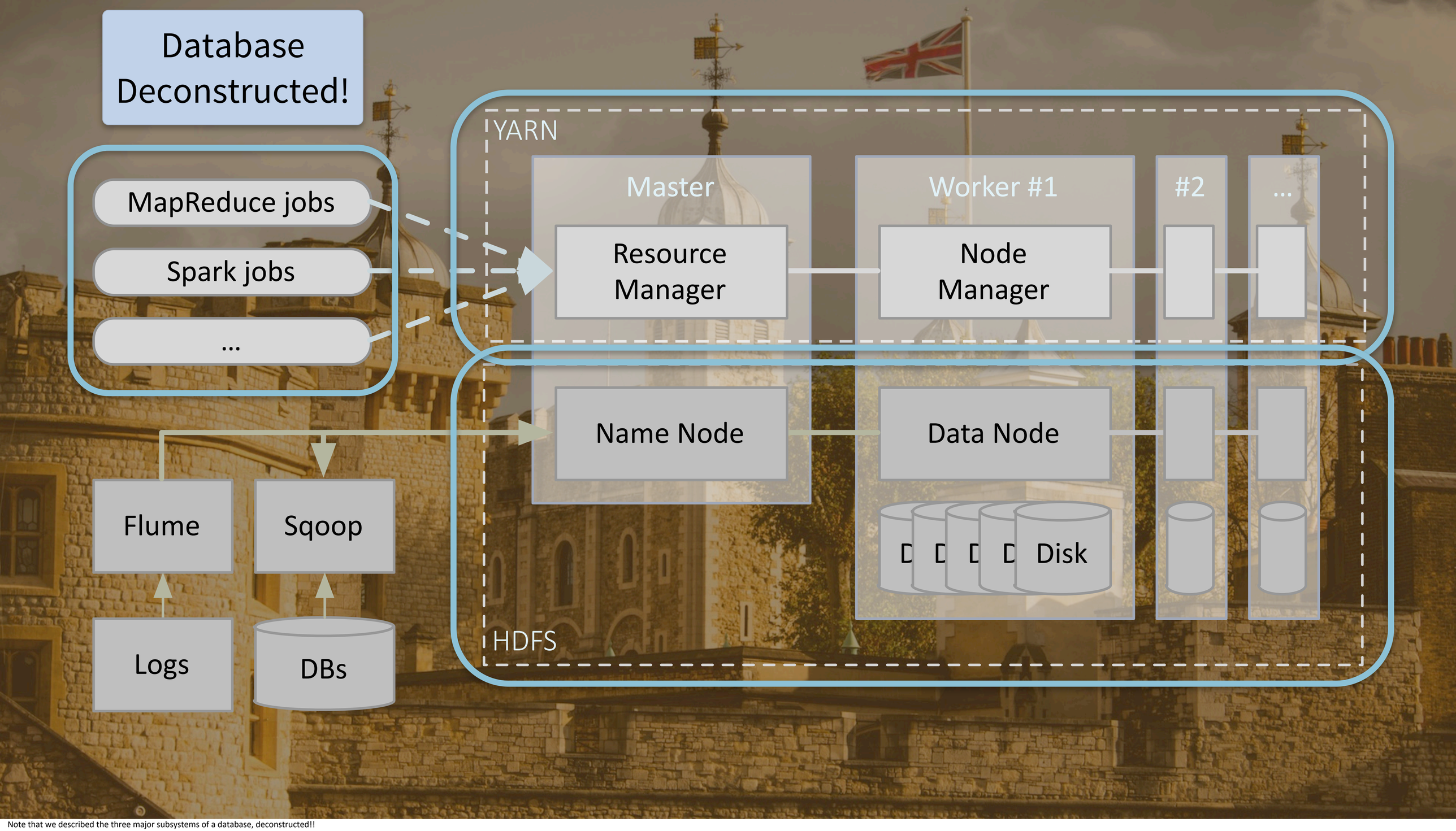
1. Storage tier, either HDFS (Hadoop Distributed File System) or alternatives like S3 or databases.



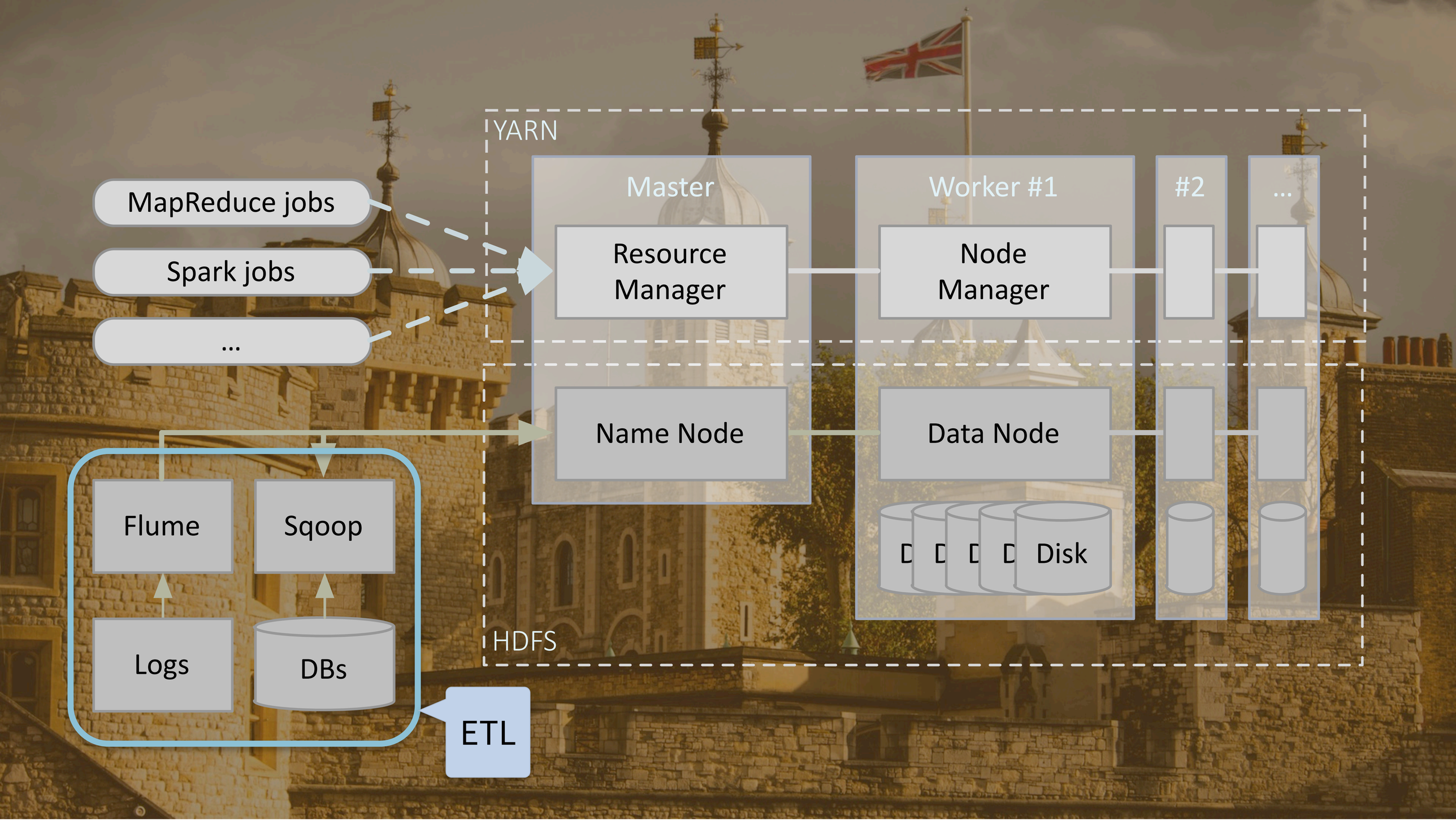
2. You need a compute engine for processing data. First we had MapReduce, then a few other tools to replace it, finally Spark has emerged as the most viable successor.



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

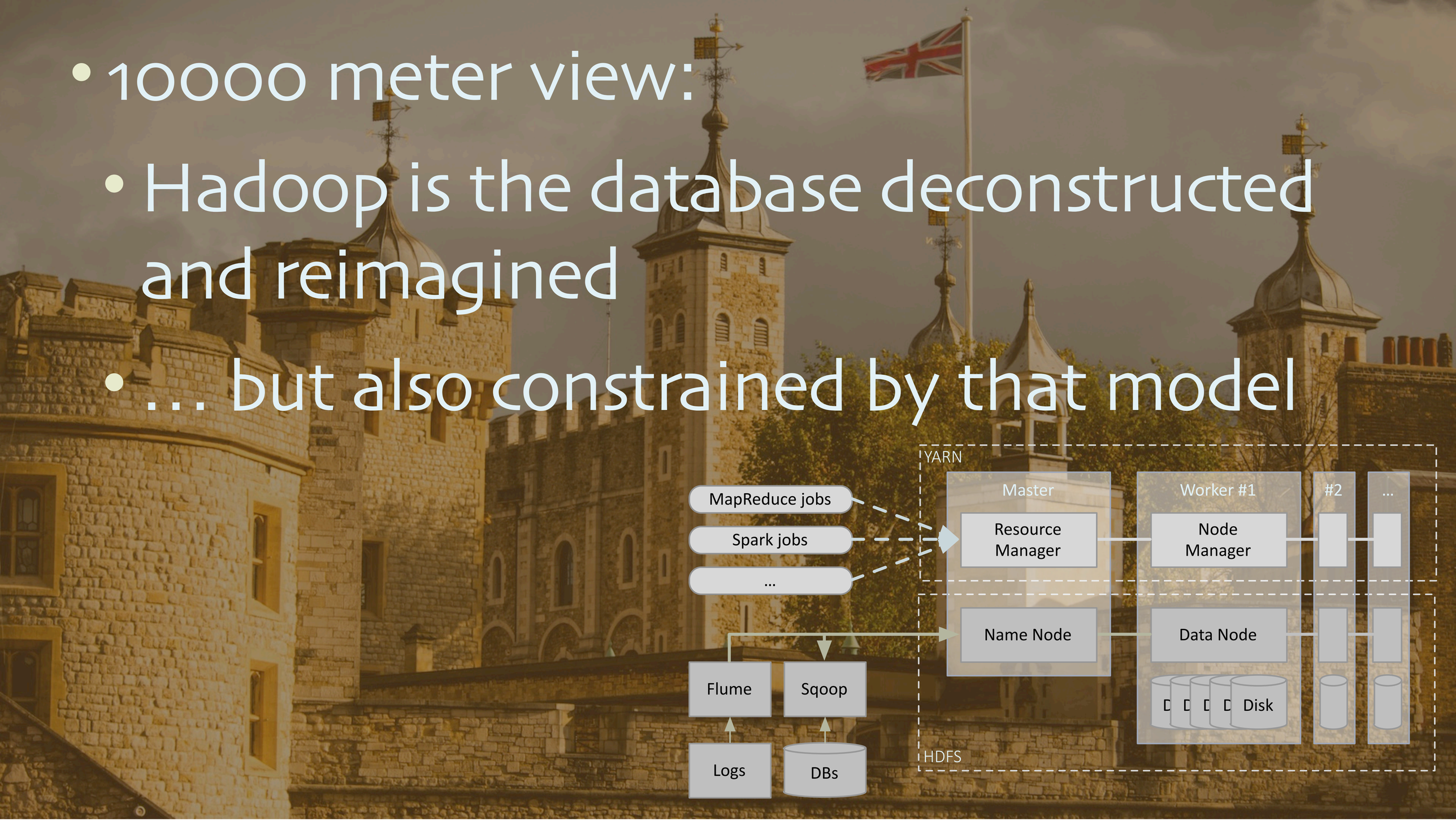


Note that we described the three major subsystems of a database, deconstructed!!



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

- 10000 meter view:
  - Hadoop is the database deconstructed and reimagined
  - … but also constrained by that model

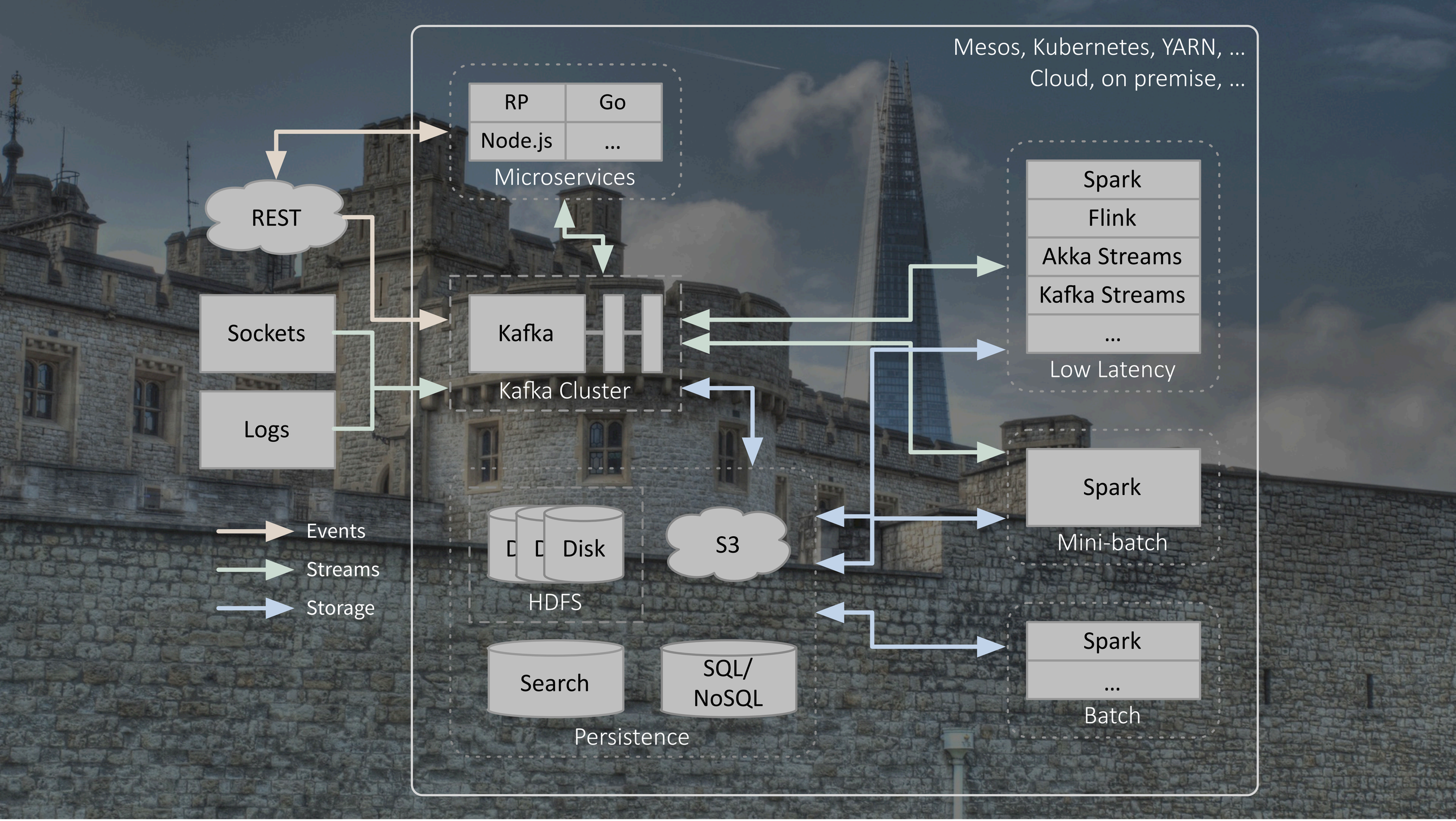


You do similar tasks in traditional RDBMS systems and Hadoop. Of course, I'm overly simplifying things and ignoring some important differences, like high-performance CRUD operations, for which Hadoop is a poor fit (partially alleviated if you use HBase or similar stores integrated with core Hadoop)

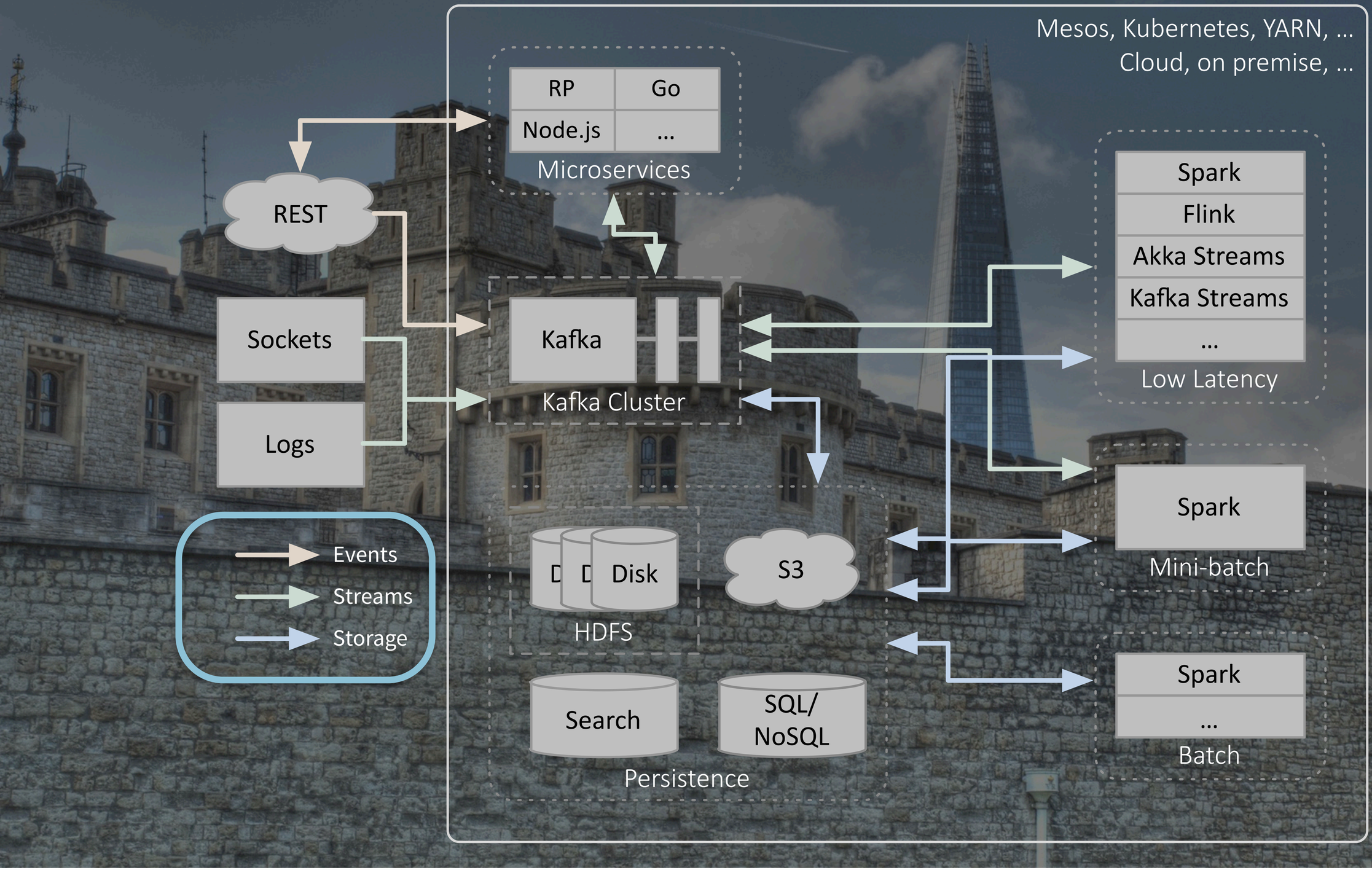
New Fast Data Architecture

But it will also support batch!!

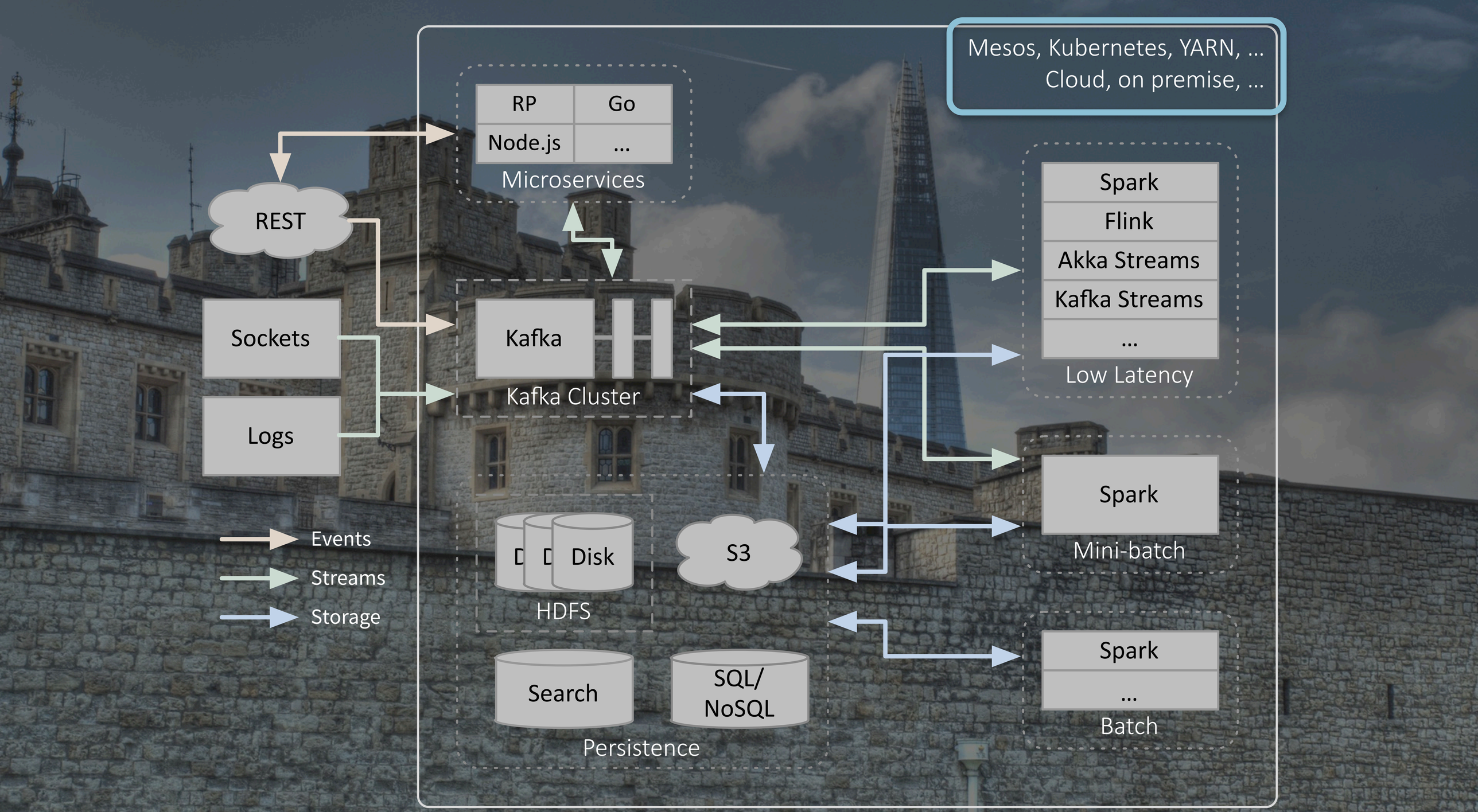
Photo: New vs. old architectures; The Shard looming over The Tower



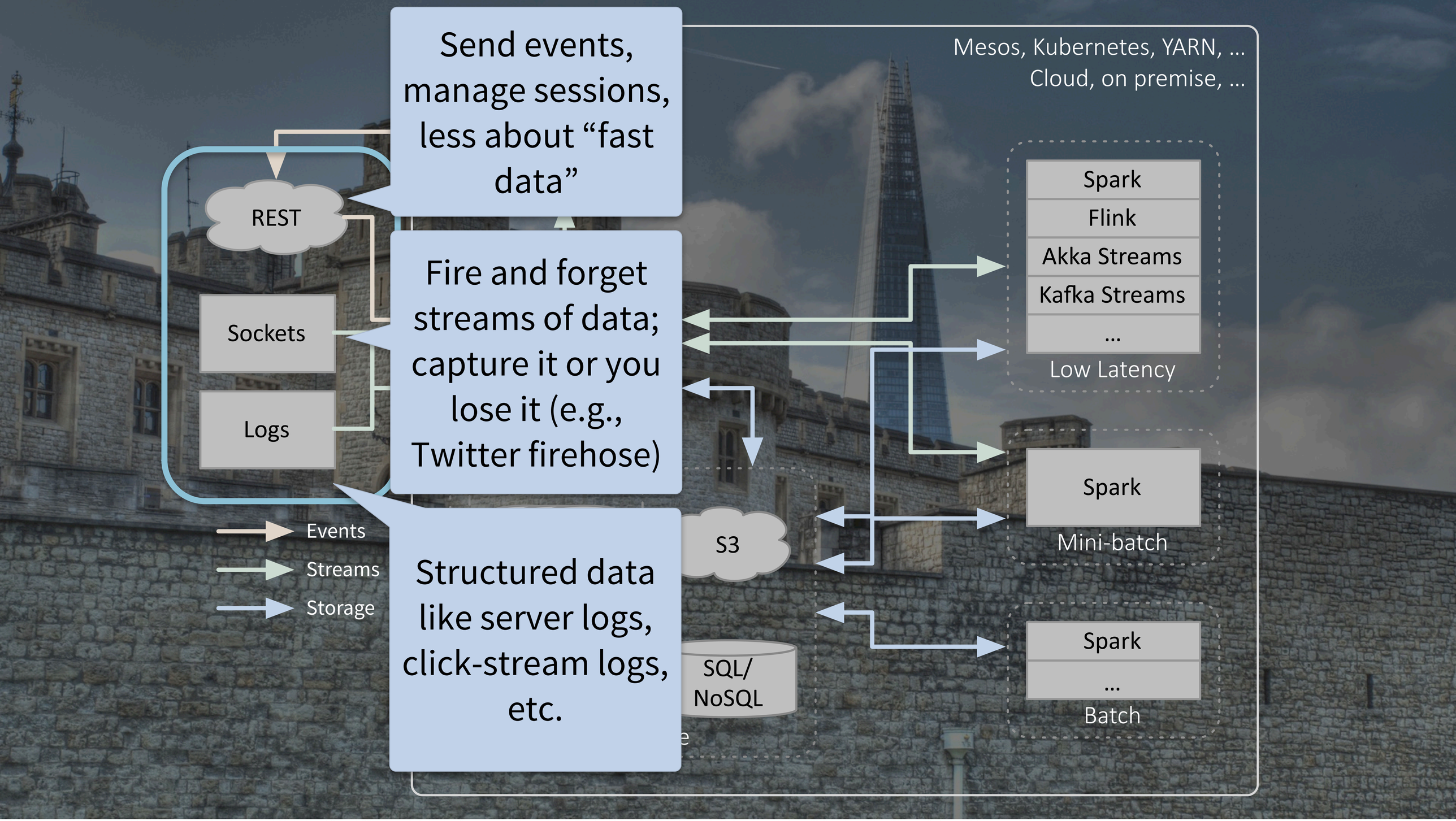
This diagram is a simplified and updated version of the one in the report. Some details updated to reflect a few changes in my thinking and industry developments.



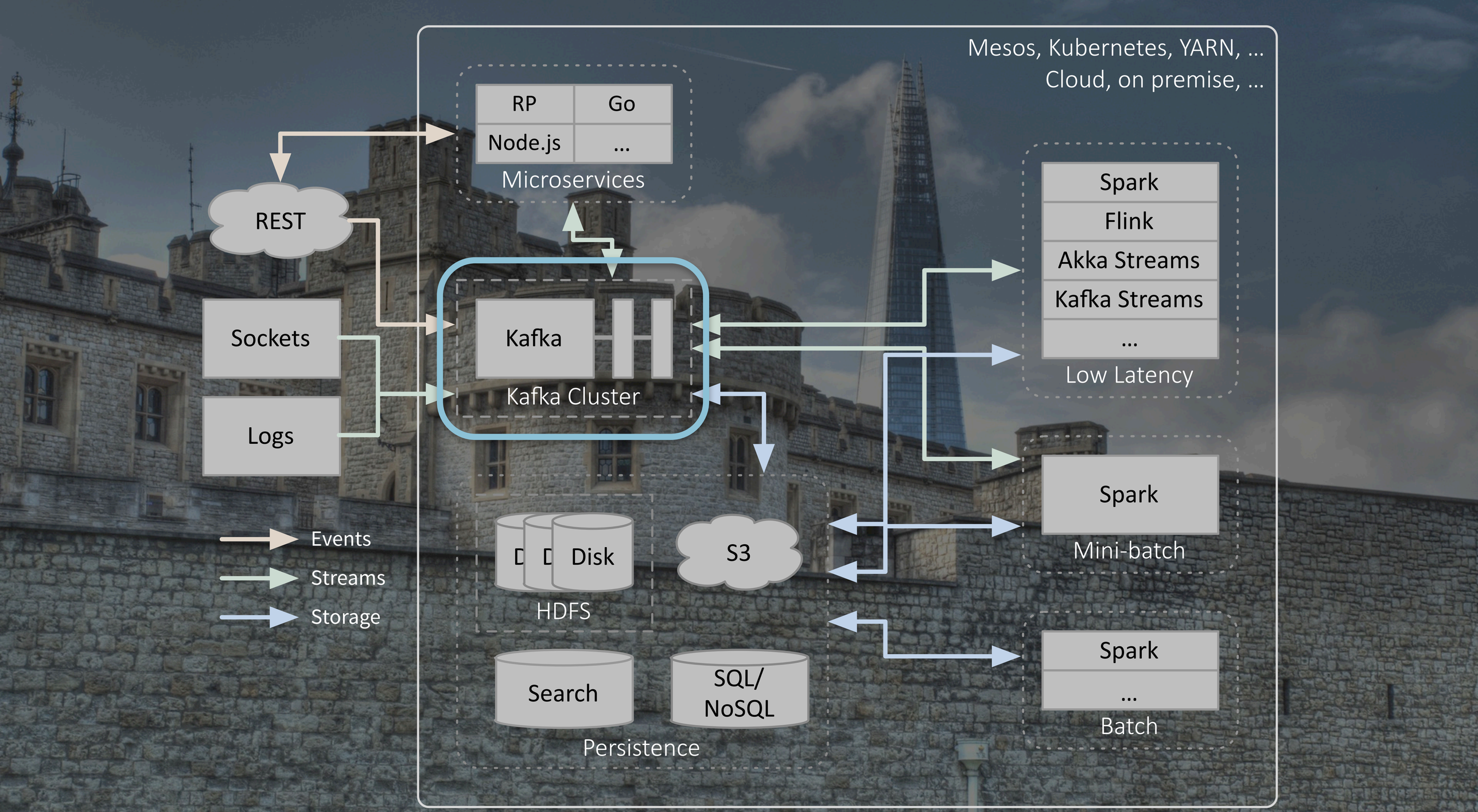
"Events" here refers to data associated with the kind of messages you use in REST, where each datum is handled uniquely (e.g., "add to cart", "purchase cart", etc.)
"Streams" could contain events or "anonymous" records, i.e., structured data like log entries, but where they are typically handled in bulk. Also, there is usually not two-way communication, except when some sort of transactional system is used to achieve "exactly once" handling, unlike REST.
"Storage" encapsulates the many ways that data is written to and read from durable stores.
The differences are mostly intended to capture different volumes, latency characteristics, communication protocols, etc.



You can run this architecture on Mesos, Kubernetes, even YARN (Hadoop), although YARN is not really flexible enough to run the wide variety of services in this picture. You can also deploy on premise or in cloud environments. At Lightbend, we have a lot of customers who are standing up these fast data clusters beside their Hadoop clusters, reserving the later for the batch and interactive workloads described previously.



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.

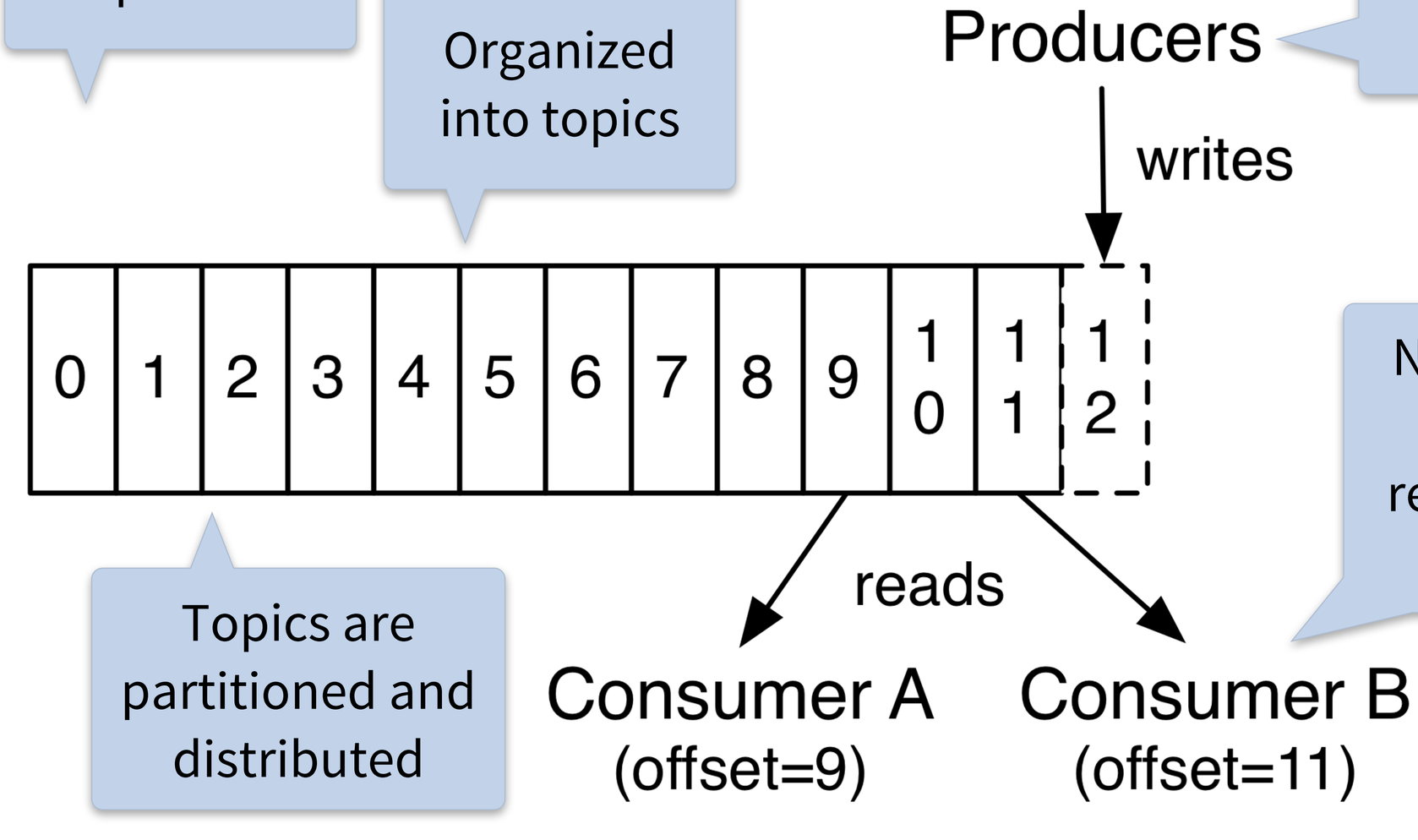


Kafka is the core “backplane” of this architecture, the place where data is ingested in log-oriented storage, organized into topics. They are not quite “queues”, like traditional message queue environments, because each consumer doesn’t delete the message it reads. Rather, Kafka manages message lifecycles, so that N consumers can see the whole stream. M Publishers and N consumers are decoupled for any numbers M and N. 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.

Why Kafka?

Logs, not queues

Organized into topics



M Producers

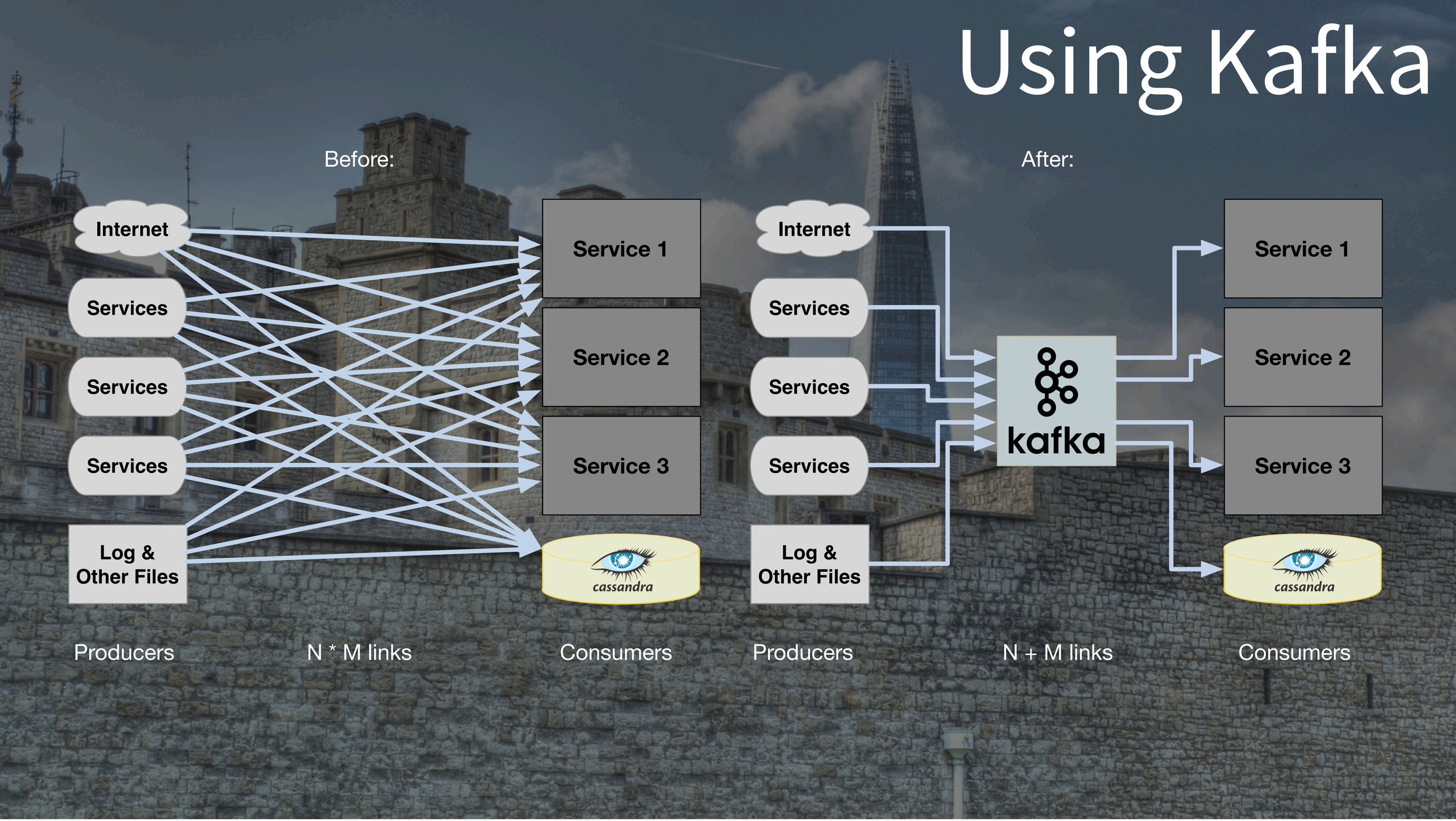
N Consumers, who start reading where they want

Topics are partitioned and distributed

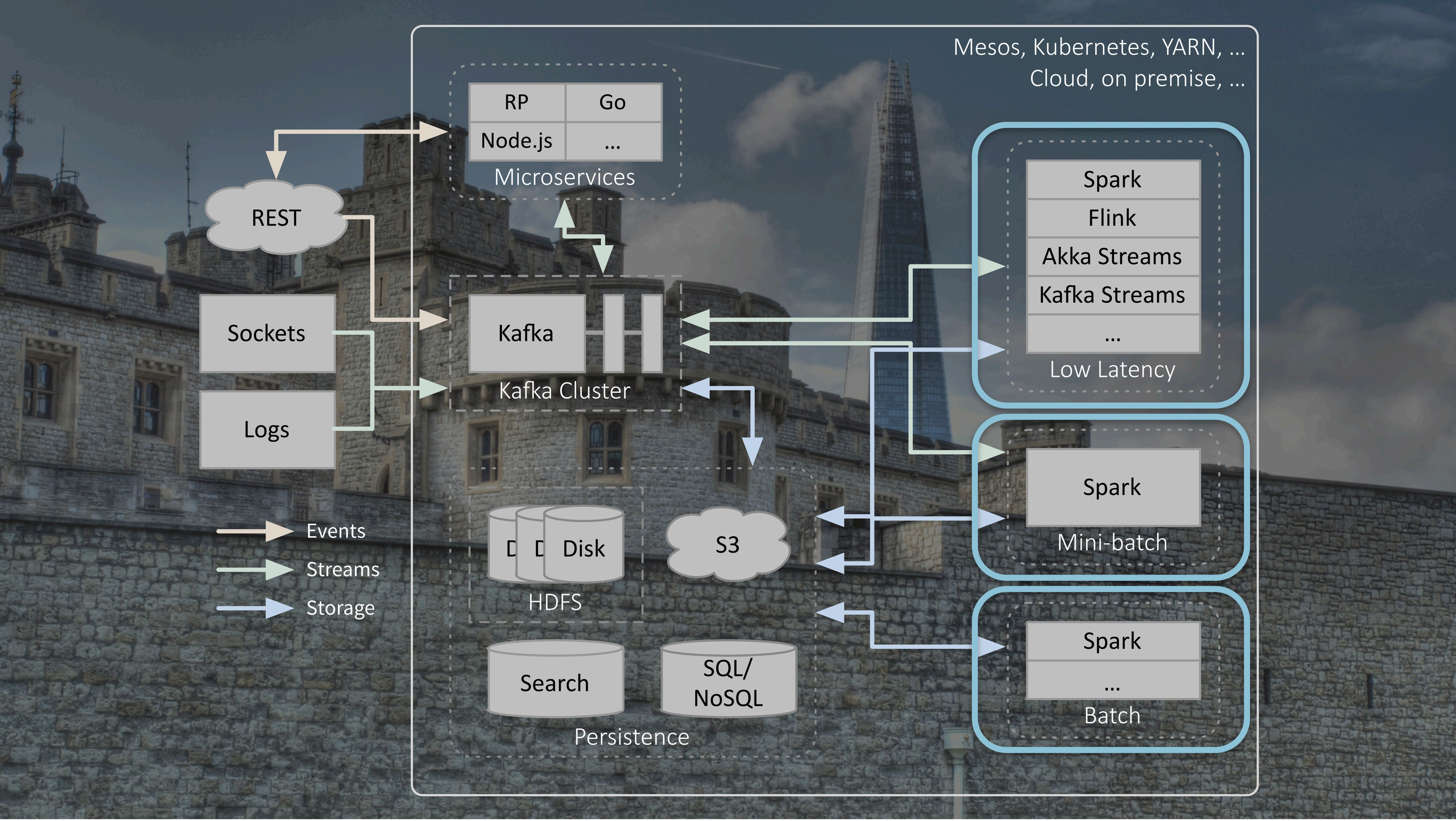
Kafka is a distributed log, storing messages sequentially. Producers always write to the end of the log, consumers can read on the log offset that they want to read from (earliest, latest, …)
Kafka can be used as either a queue or pub sub
The main differences are:
1. Log is persistent where queue is ephemeral (reads pop elements)
2. Traditional message brokers manage consumer offsets, while log systems allow users to manage offsets themselves

Alternatives to Kafka include Pravega (EMC) and Distributed Log/Pulsar (Apache)

Using Kafka



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.

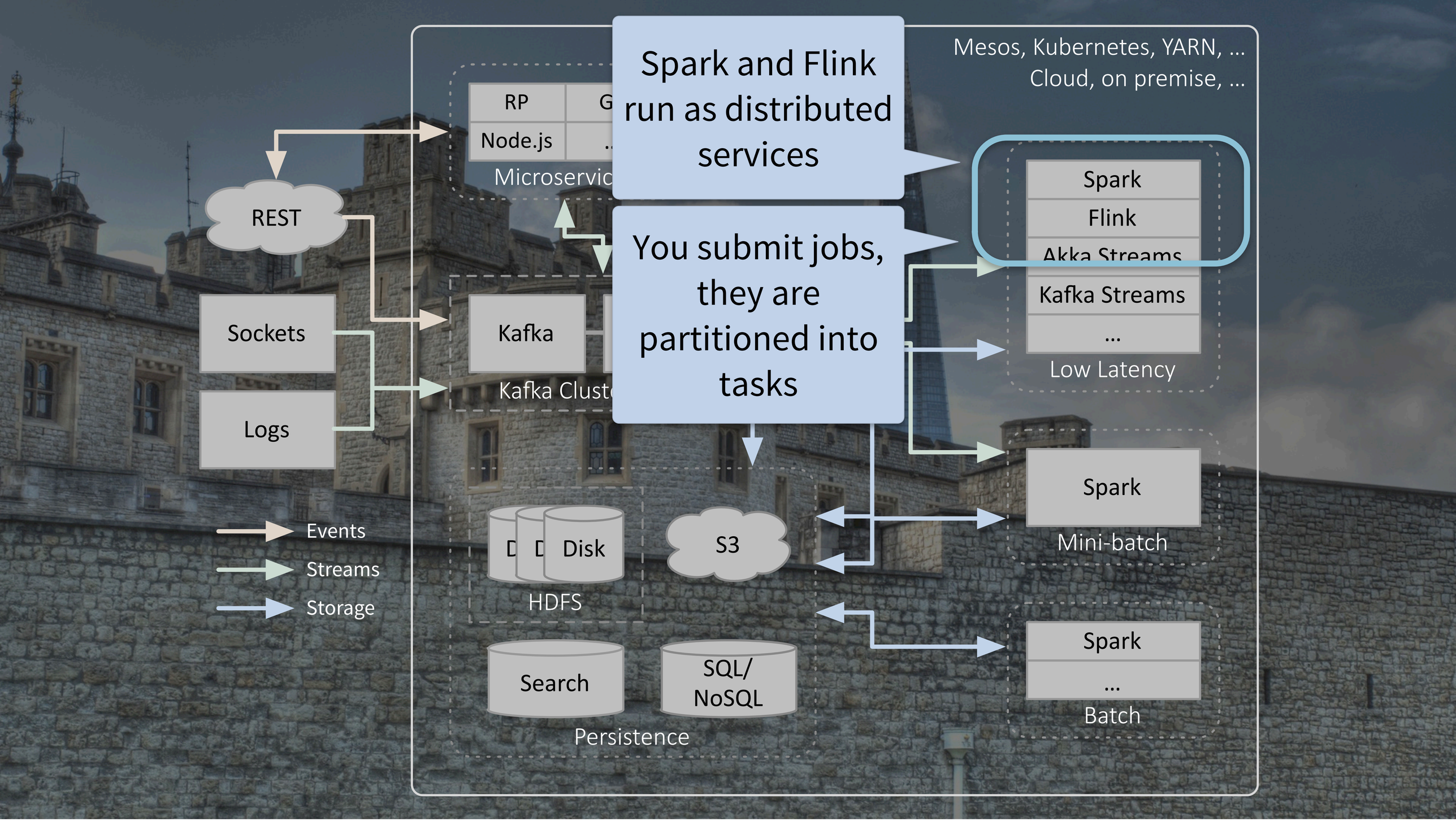


Many low-latency (“pure”) streaming, mini-batch streaming, and batch engines are vying for your attention.

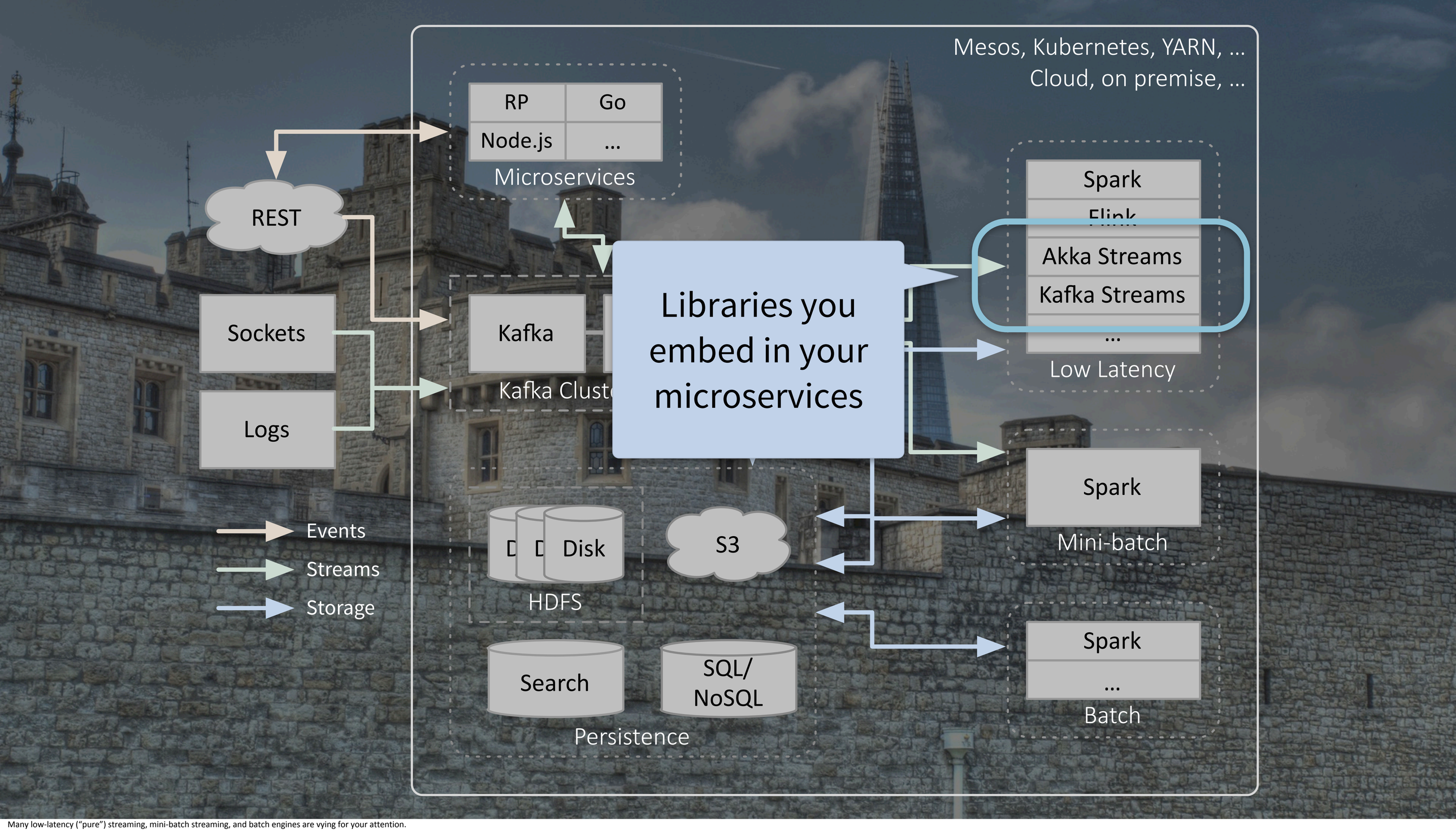
How do you choose?

- Latency? How low?
- Volume: How high?
- Which kinds of data processing?
- What's your preferred application architecture?

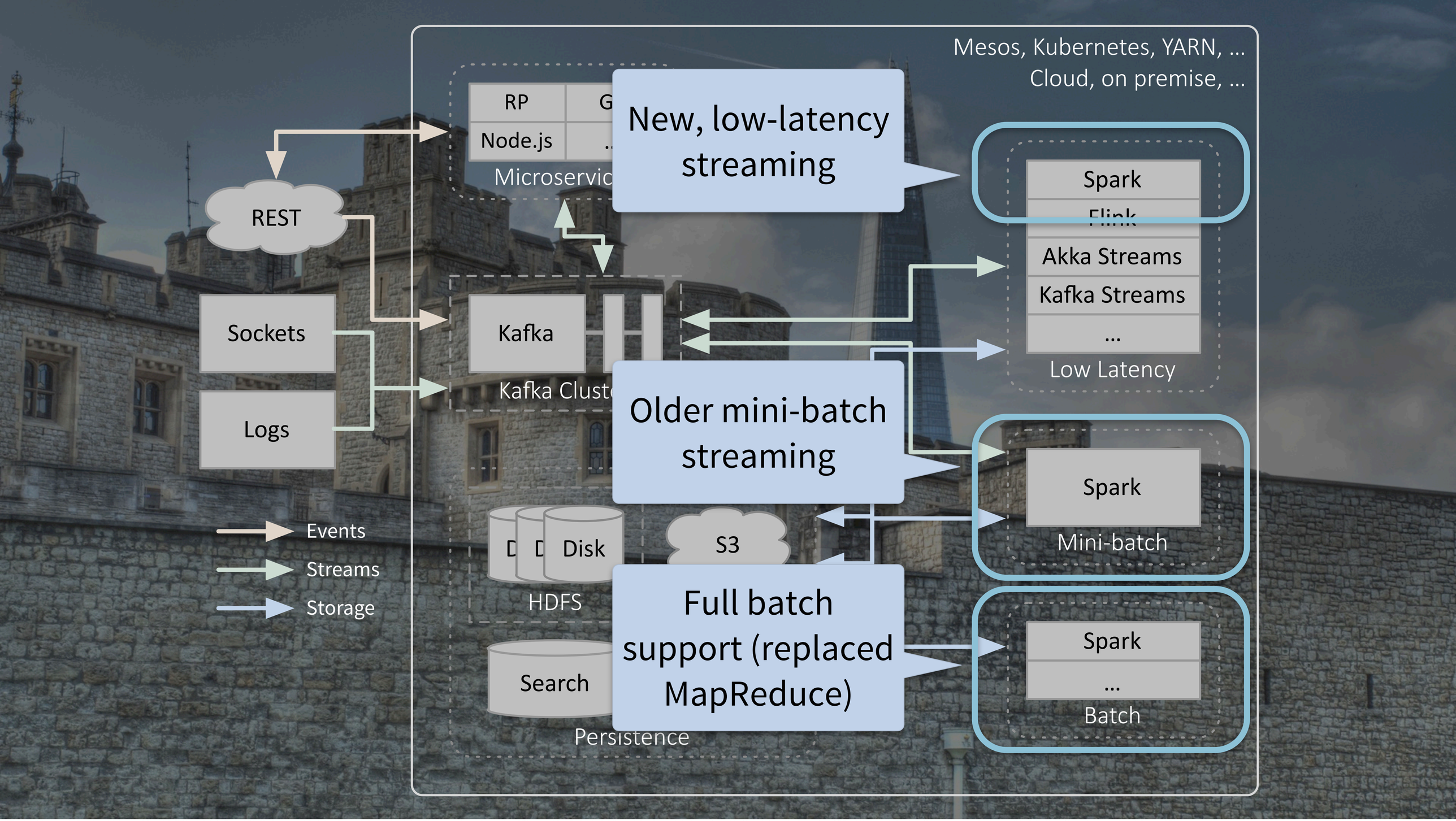
If you have tight latency requirements, you can't use a mini-batch or batch engine. If you're latency constraints are more flexible, you have more tool options and you can do more sophisticated and expensive things, like training ML models, write to databases, etc.
What's your volume? Do you need to partition your streams over a cluster to use parallelism to handle the load?
Are you running SQL-like queries over the data? Are you doing "simple" ETL? Are you training ML models or serving models?
Do you want to run microservices for data, too? Is the overhead and compute "model" of Spark or Flink right for you, or do you need more flexibility?



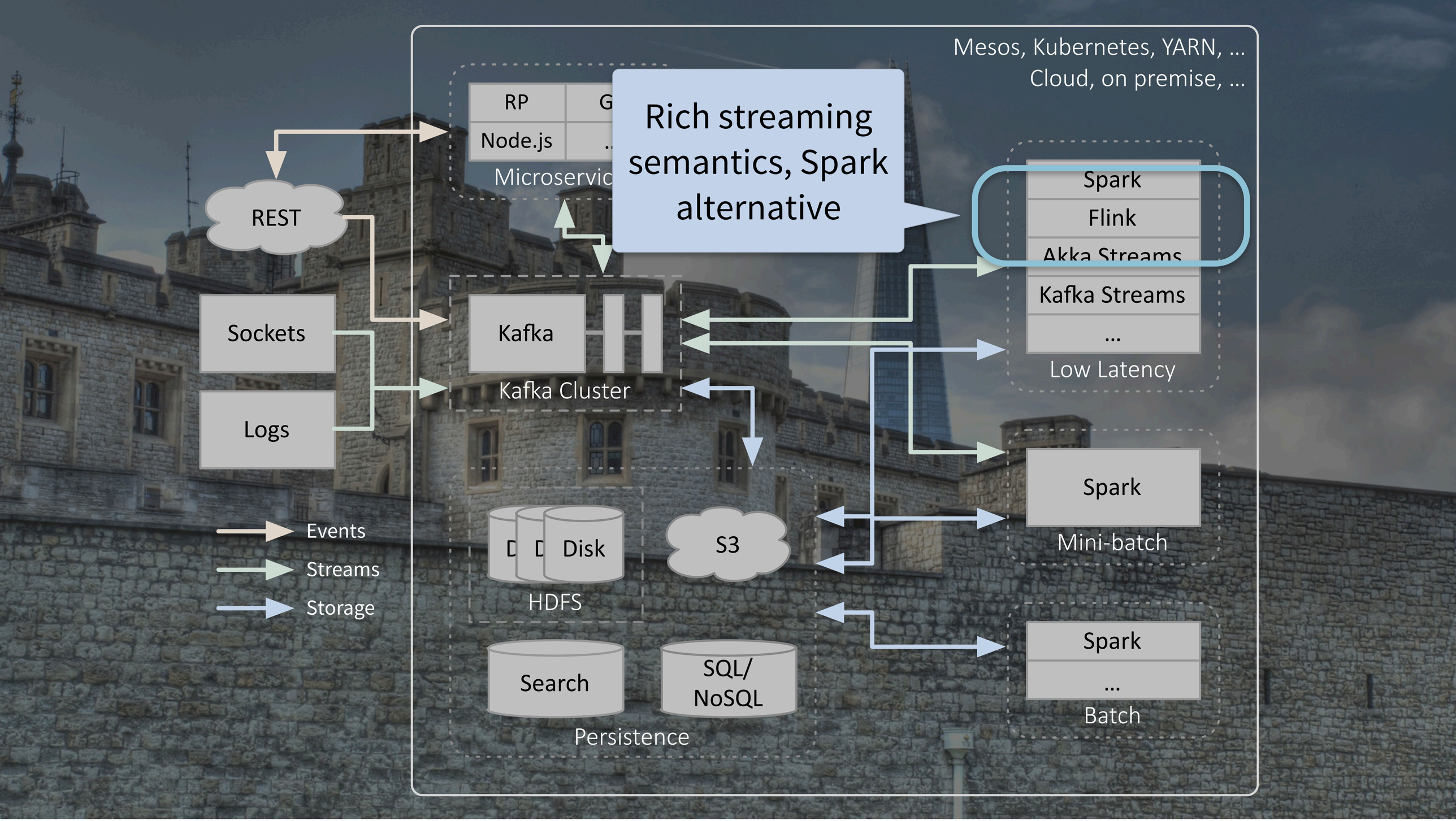
The best known is Spark, which provides rich SQL and dataflow APIs that support batch jobs and streaming jobs. There are two streaming APIs, an older “mini-batch” model, where data is captured in time intervals down to ~100 milliseconds in size, then processed as a small batch. Now there is a new low-latency API.



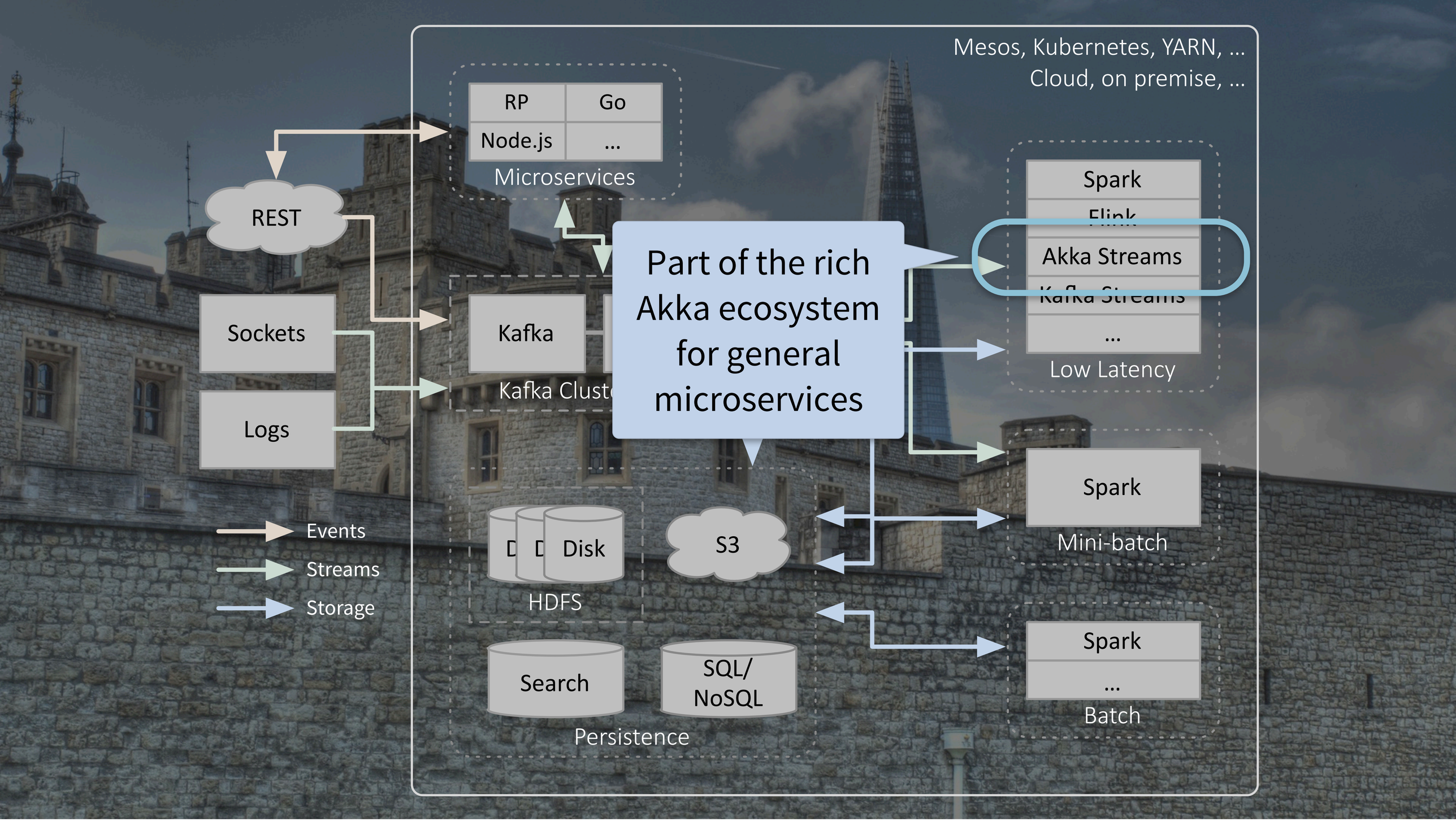
Many low-latency (“pure”) streaming, mini-batch streaming, and batch engines are vying for your attention.



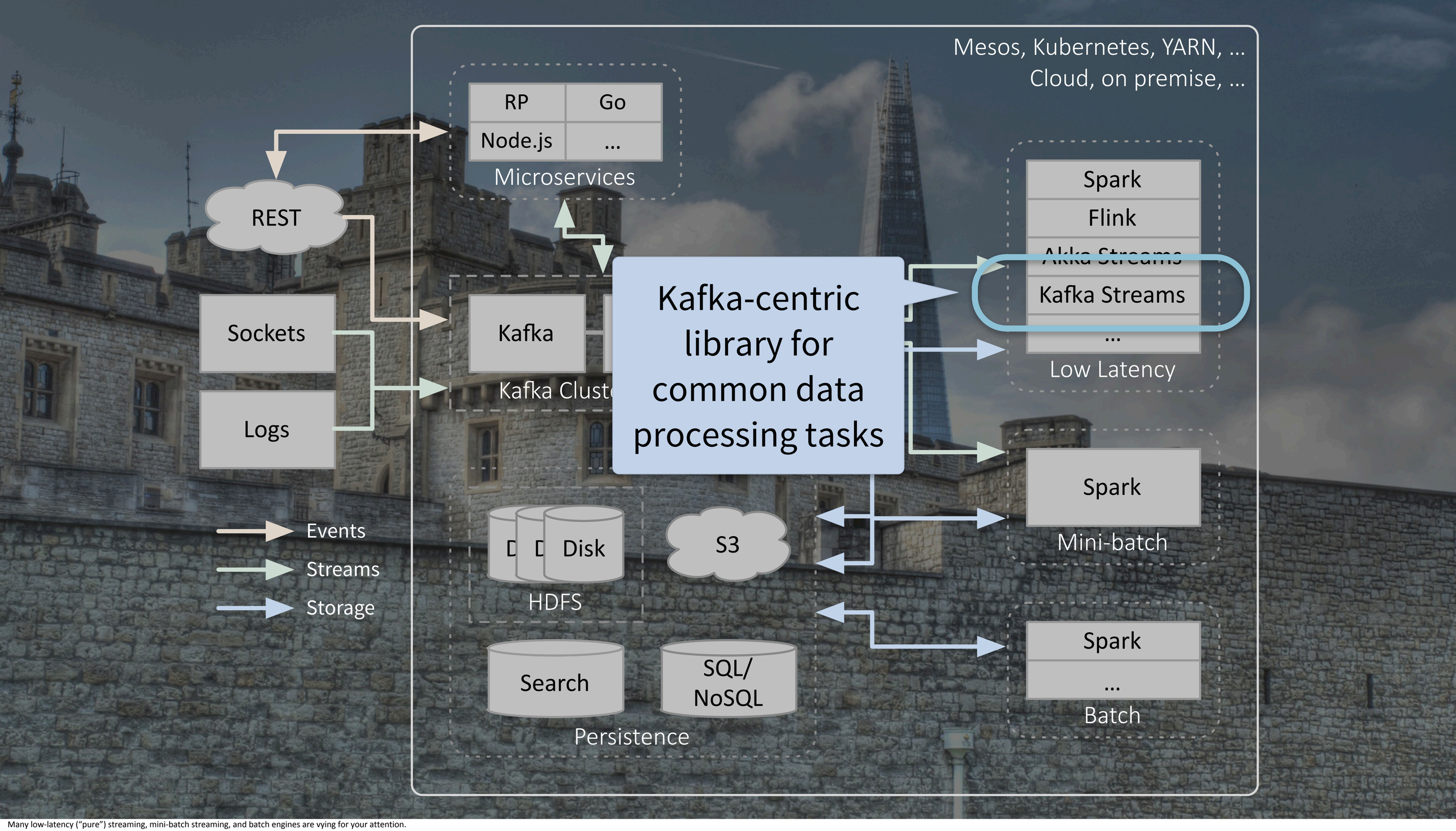
The best known is Spark, which provides rich SQL and dataflow APIs that support batch jobs and streaming jobs. There are two streaming APIs, an older “mini-batch” model, where data is captured in time intervals down to ~100 milliseconds in size, then processed as a small batch. Now there is a new low-latency API.



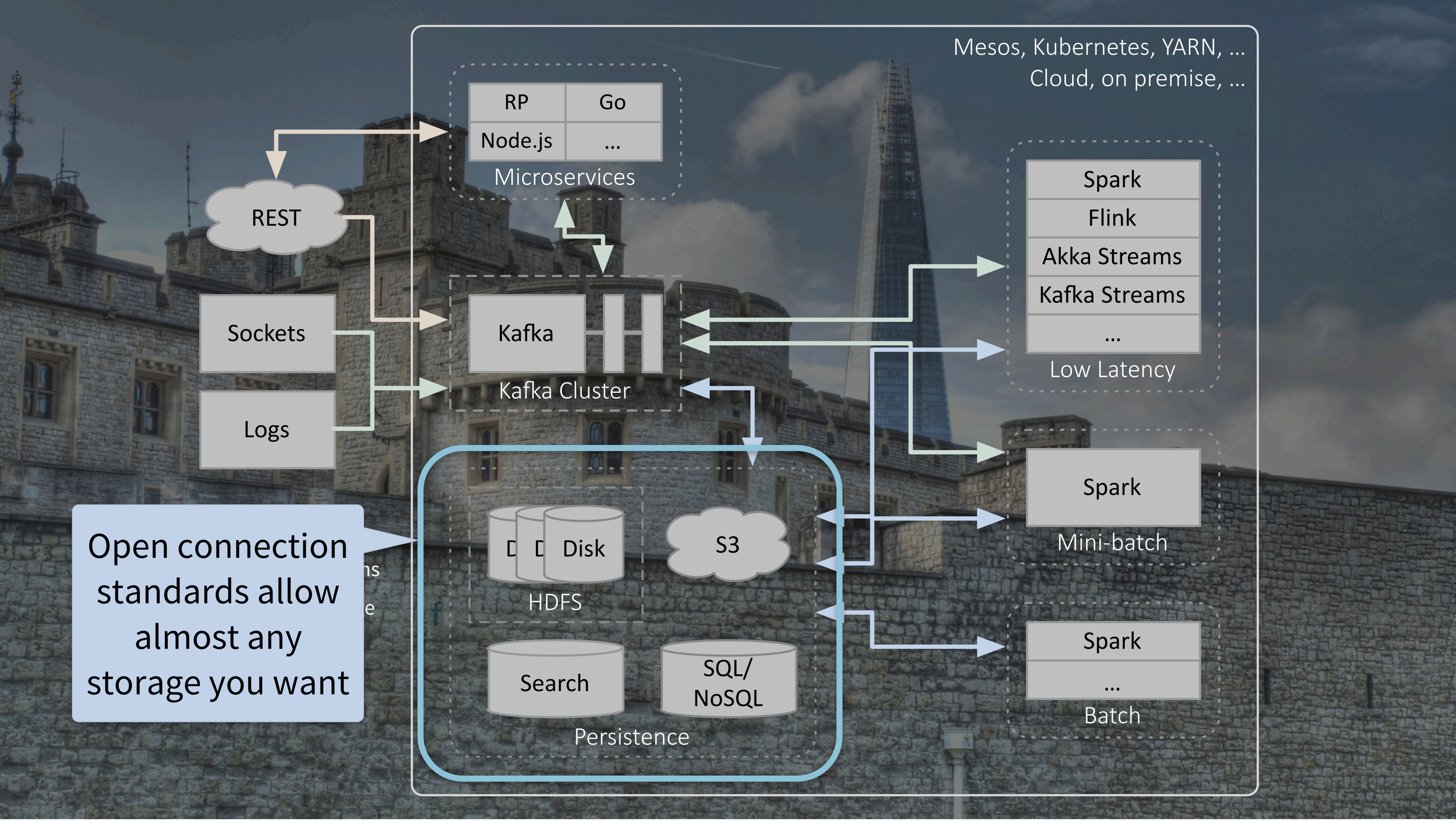
The best known is Spark, which provides rich SQL and dataflow APIs that support batch jobs and streaming jobs. There are two streaming APIs, an older “mini-batch” model, where data is captured in time intervals down to ~100 milliseconds in size, then processed as a small batch. Now there is a new low-latency API.



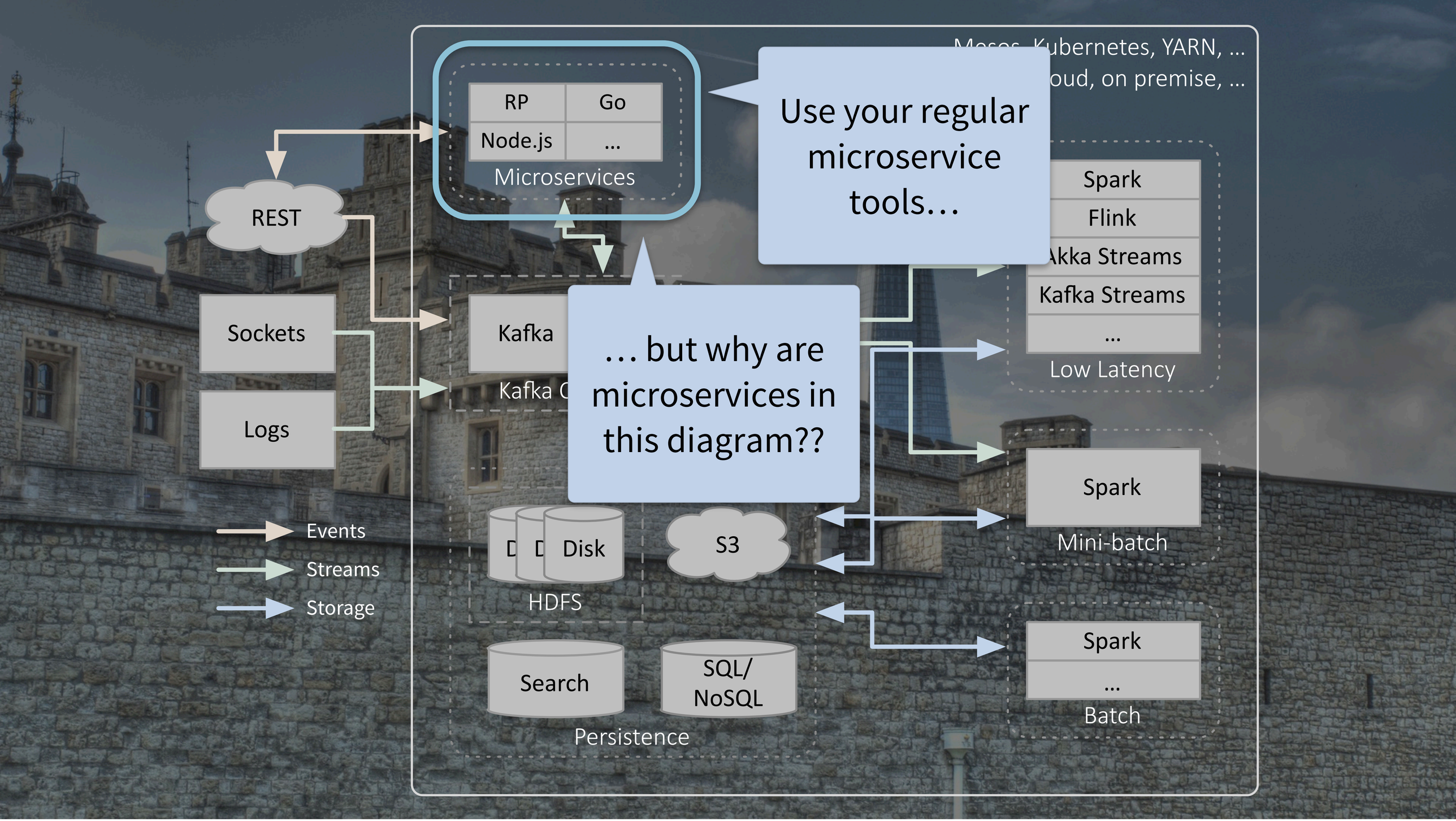
Many low-latency (“pure”) streaming, mini-batch streaming, and batch engines are vying for your attention.



Many low-latency (“pure”) streaming, mini-batch streaming, and batch engines are vying for your attention.



Data can also be read to and written from almost any storage environment, especially for longer term storage than Kafka is typically used for, and for “non-streaming” applications, like data warehousing, interactive queries, search and indexing, etc.



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. We'll come back to this later.

Microservices in Fast Data

- Run everything in big clusters using Kubernetes or Mesos
  - In the cloud or on-premise

Why an integrated architecture with microservices and data services combined? First, there is a growing trend where people just want to deploy and manage one or a few large, heterogeneous clusters that run most work, with occasionally using small, specialized clusters for particular workloads (like databases, Kafka in performance critical scenarios, etc.)

Microservices in Fast Data

- If streaming gives you information faster from new data, you want those answers quickly and easily available in your other services

If you are doing streaming to extract information more quickly from your data, you probably want to make that information available in all your other services as quickly as possible.

Microservices in Fast Data

- Streaming raises the bar on data services
  - Compare to batch services, long-running streaming services must be more:
- Scalable
- Resilient against failure
- Flexible

Streaming is forcing data engineers to learn skills they didn't need as much before; how to build services that scale up and down with load, remain available and resilient when rare failures happen eventually (if you wait long enough), and support a wider class of performance characteristics, data processing scenarios, etc.

Microservices in Fast Data

- This leads to our last major point…

Organizational Impact

Outline

Organizational Impact

- Data engineers have to become good at highly-available microservices
- Microservice engineers have to become good at data

The Recent Past

Some overlap in 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…

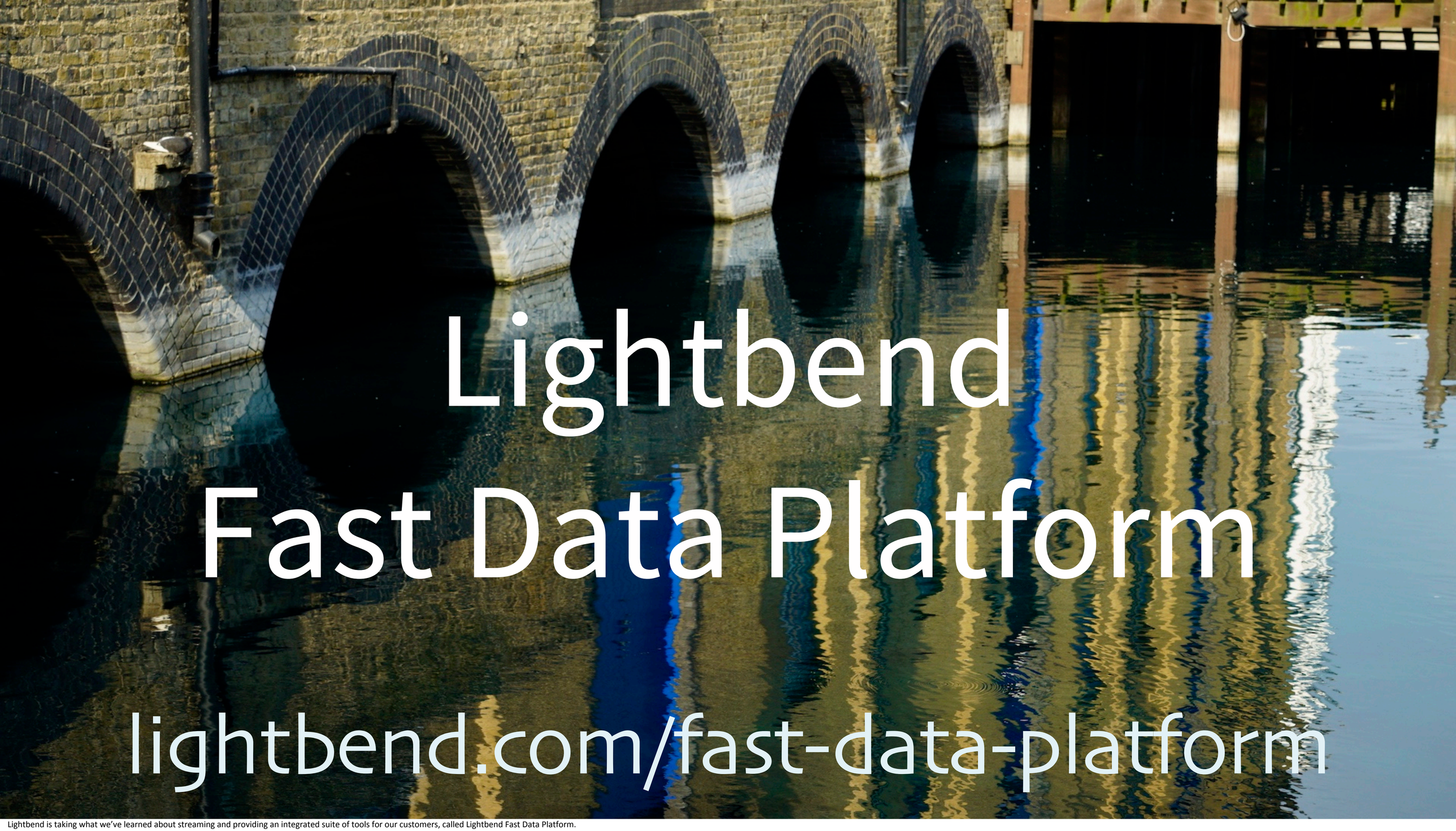
The Present

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.

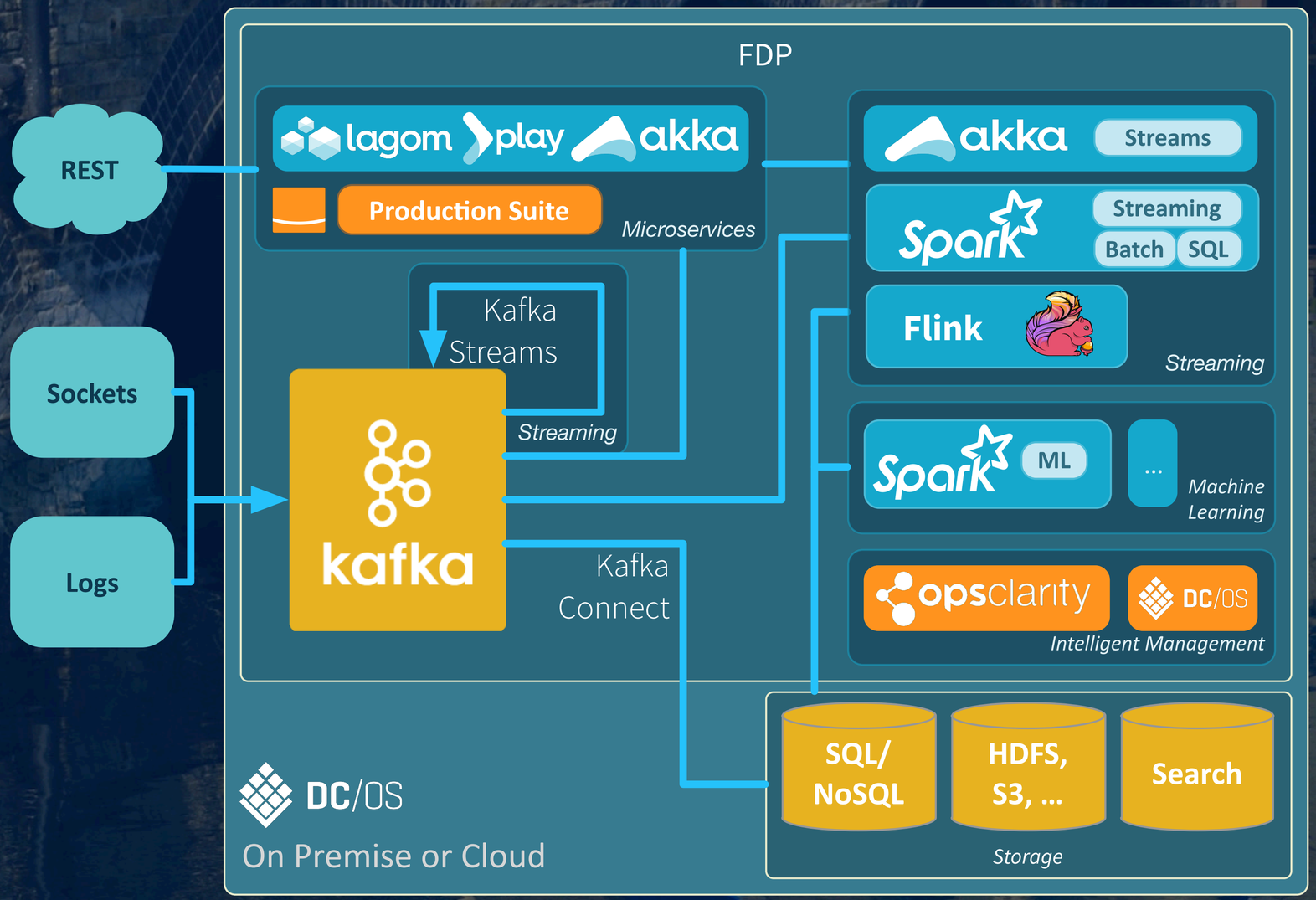
Finally I predict that as these architectures evolve, using microservices to process data, rather than systems like Spark and Flink, will become dominant. One simple reason is that streaming almost always means processing much smaller volumes of data at a time, so the distribution capabilities of Spark, etc. become less critical (compared to batch processing and interactive queries). If you like having one approach to learn, deploy, and manage, and you're already doing microservices for everything else, then why not use that same application design for streaming, too?



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: Building and water in the Docklands area of London

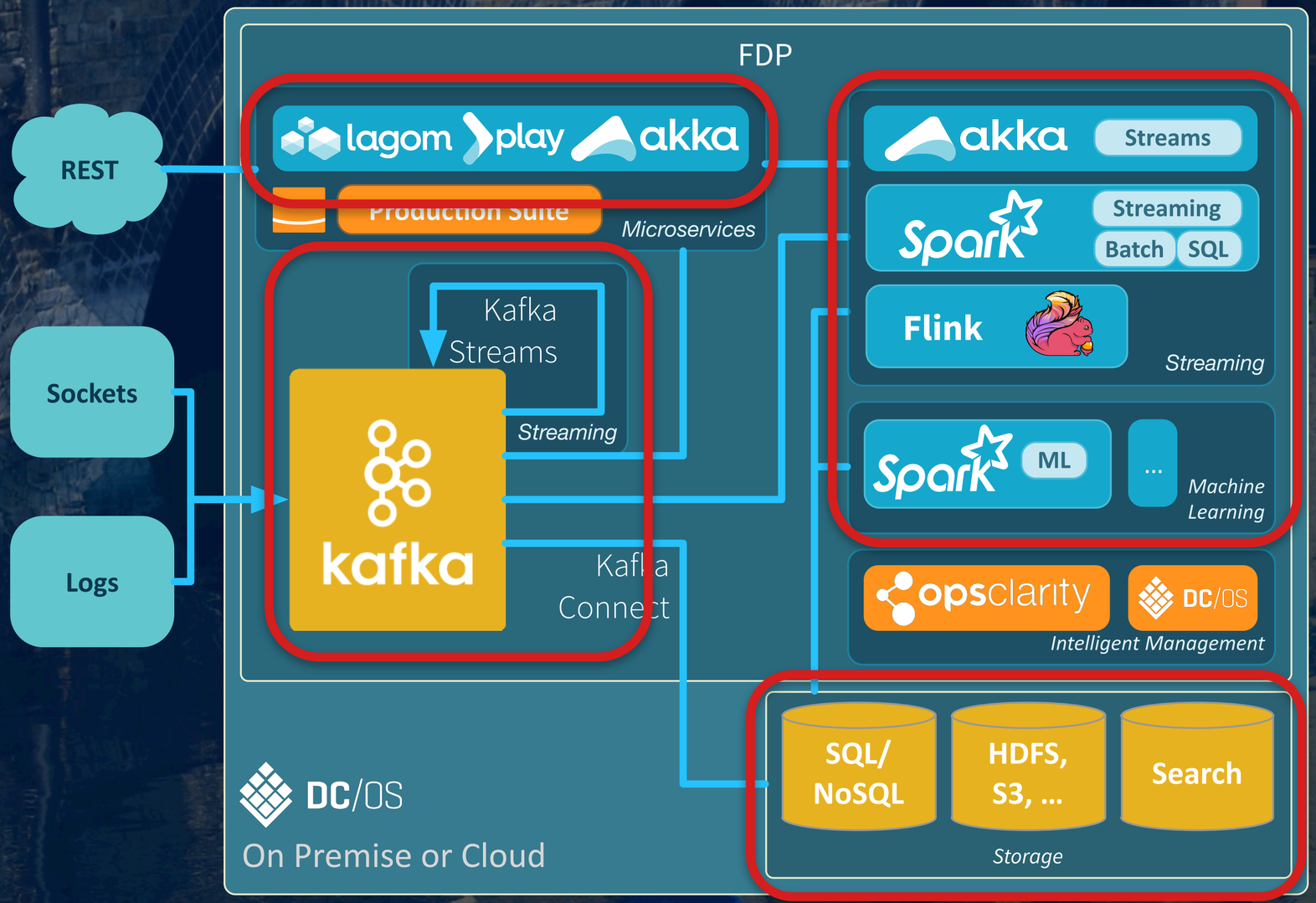
Lightbend Fast Data Platform V1.0



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.
(You can tell this is a commercial product, because the image is in color…)

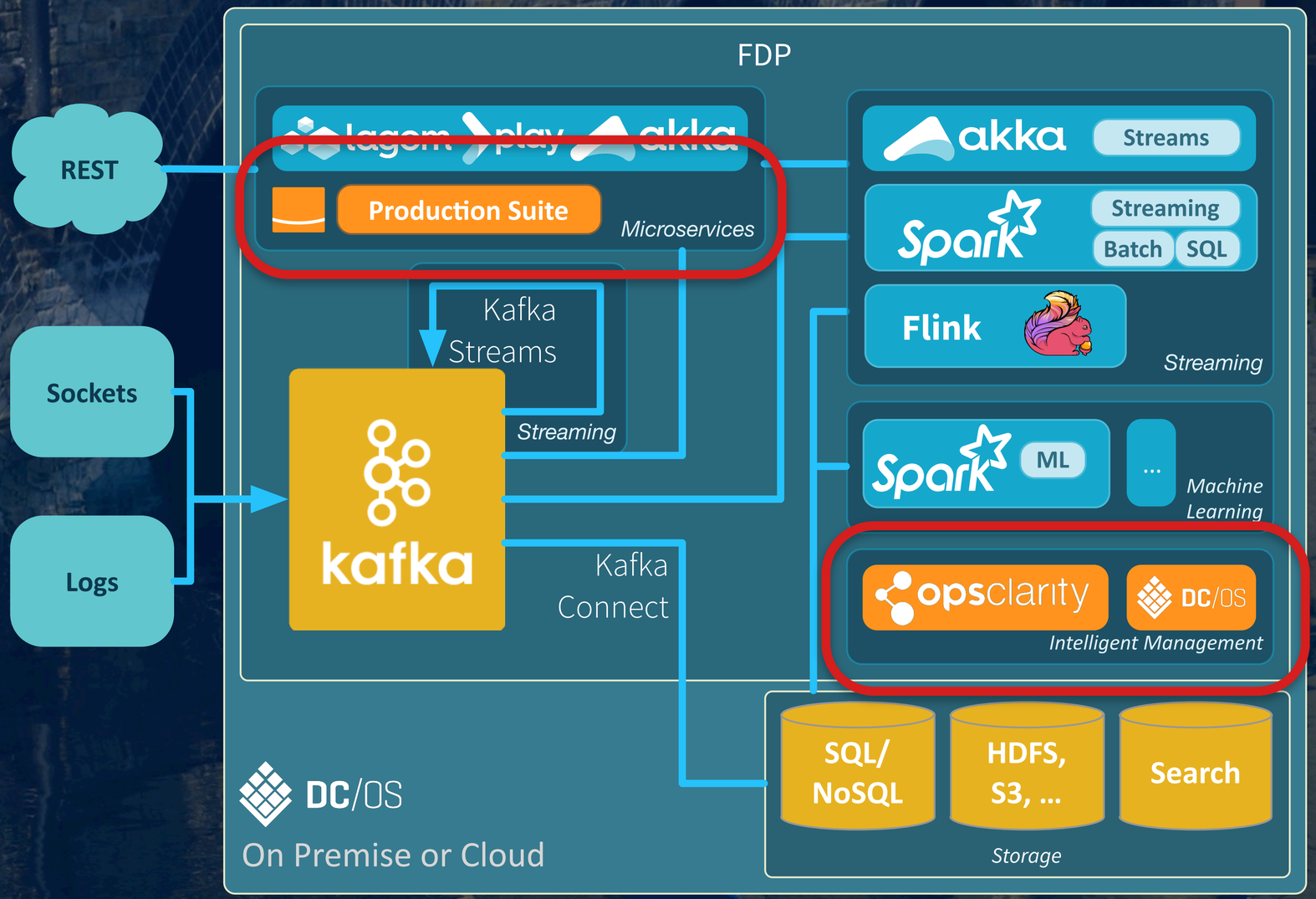
Lightbend Fast Data Platform V1.0



What we discussed

lightbend.com/fast-data-platform

Lightbend Fast Data Platform V1.0



Plus management and monitoring tools

lightbend.com/fast-data-platform

lightbend.com/fast-data-platform

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

Lightbend