Building Kafka-based Microservices with Akka Streams and Kafka Streams

Boris Lublinsky and Dean Wampler, Lightbend

boris.lublinsky@lightbend.com
dean.wampler@lightbend.com

Lightbend

Fast Data Architectures for Streaming Applications

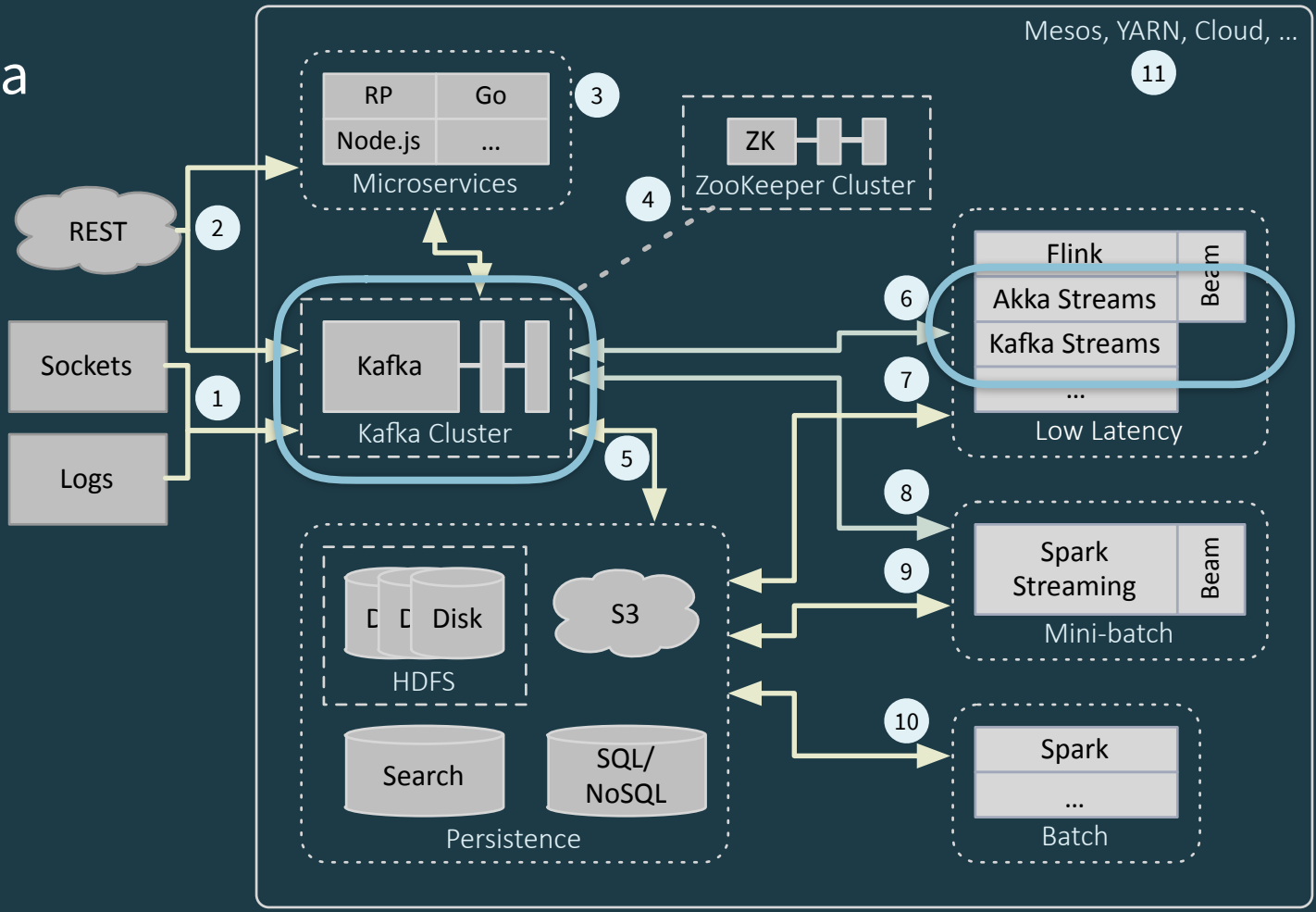
Getting Answers Now from Data Sets that Never End

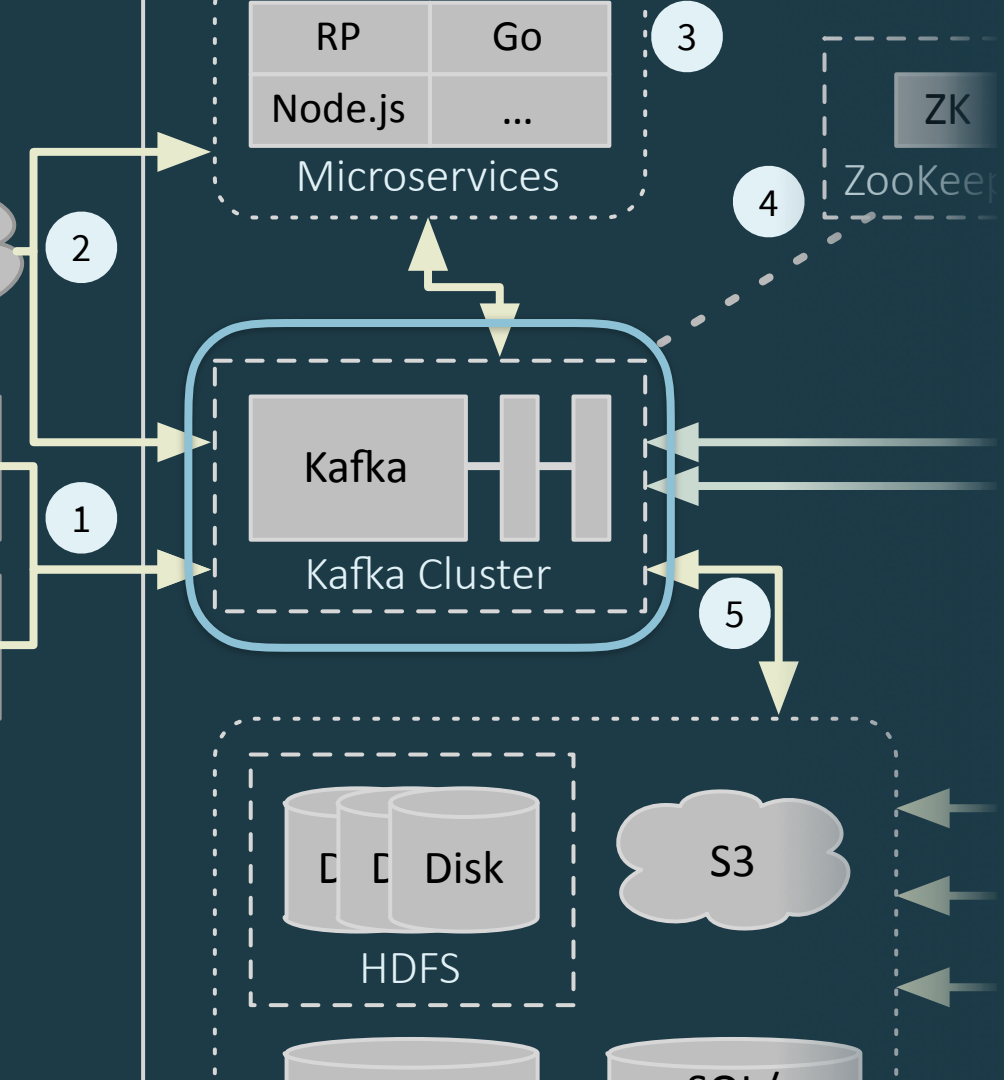
By Dean Wampler, Ph. D., VP of Fast Data Engineering

Get Your Free Copy

Today's focus:

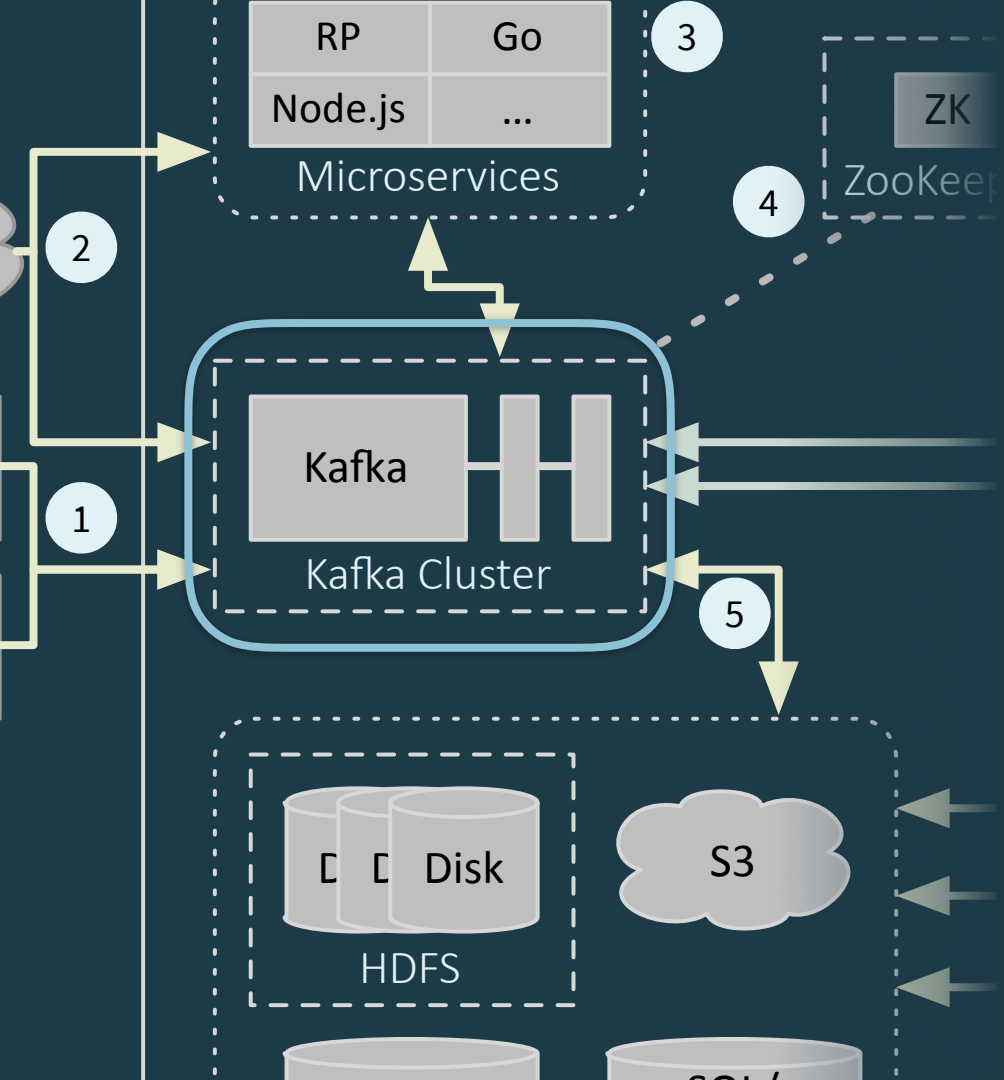
- Kafka - the data backplane
- Akka Streams and Kafka Streams - streaming microservices





Kafka:

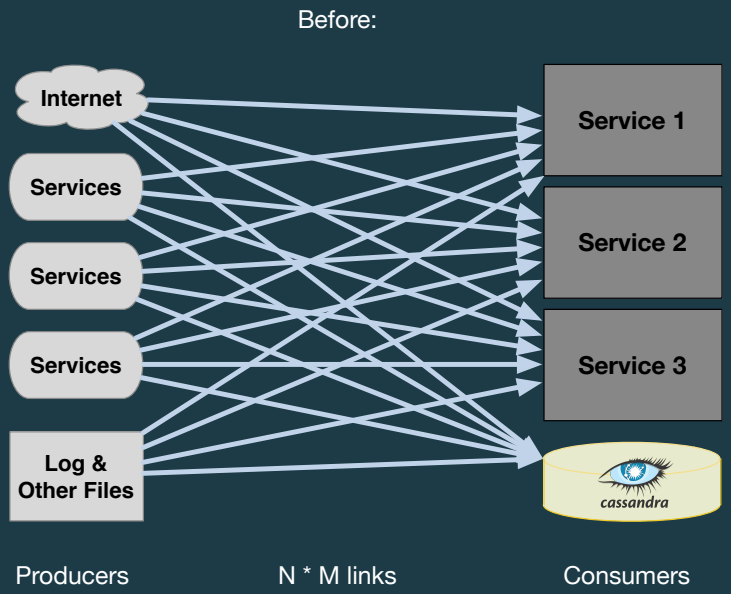
- Run as a cluster on 1+ servers
- Stores *logs* of records in topics.
- Each record: key, value, and timestamp.
- Topics can be partitioned



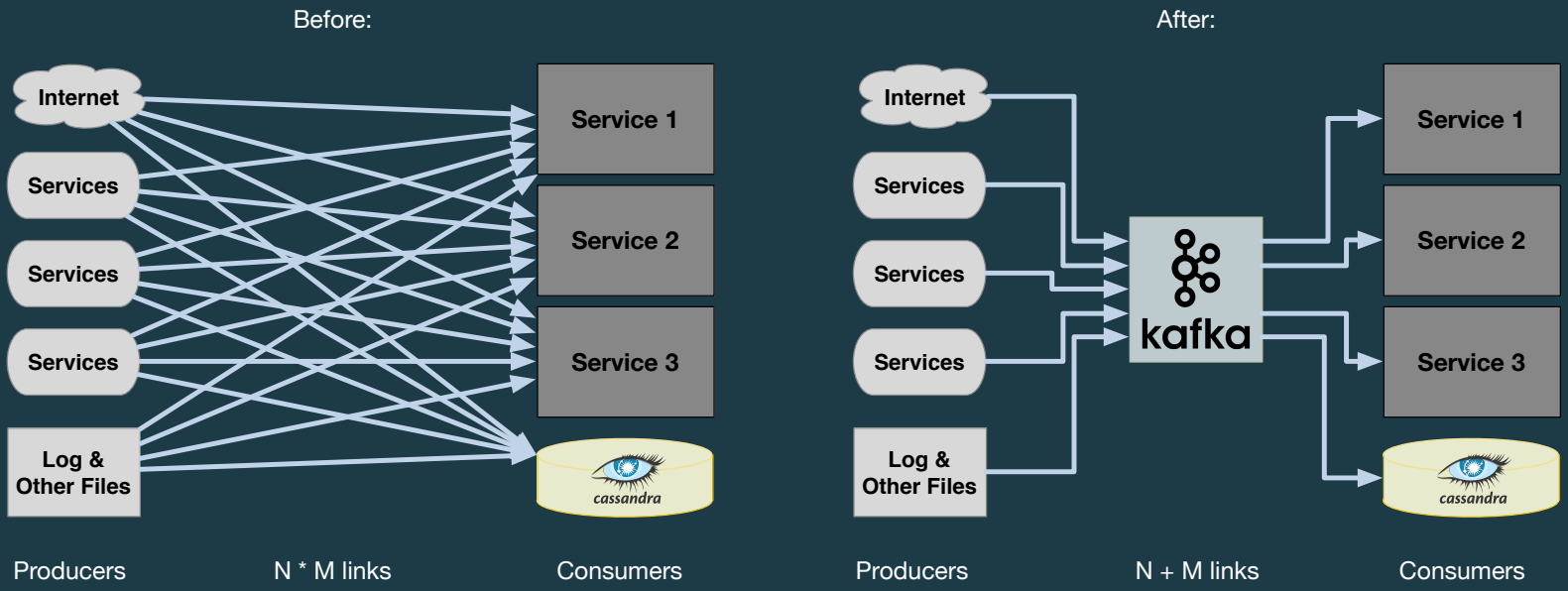
Kafka:

- Each topic can have 1 or more
  - Producers
  - Consumers

Why Kafka for Connectivity?



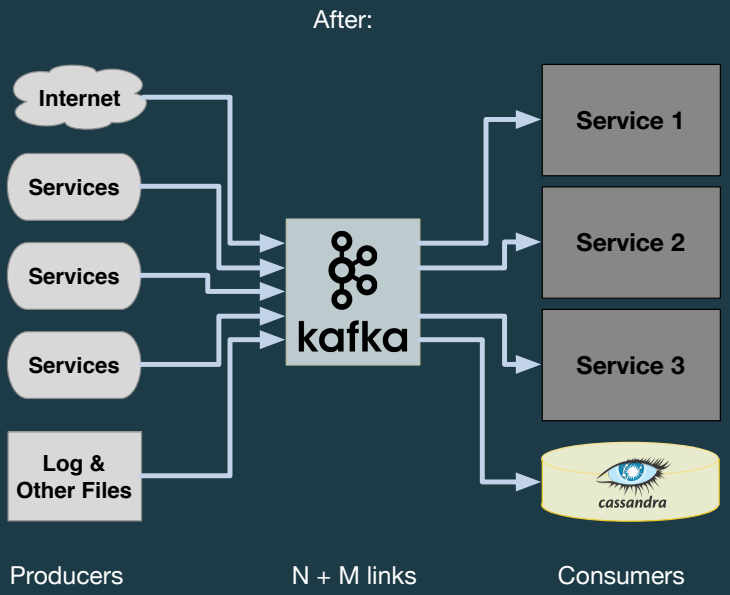
Why Kafka for Connectivity?



Why Kafka for Connectivity?

Kafka:

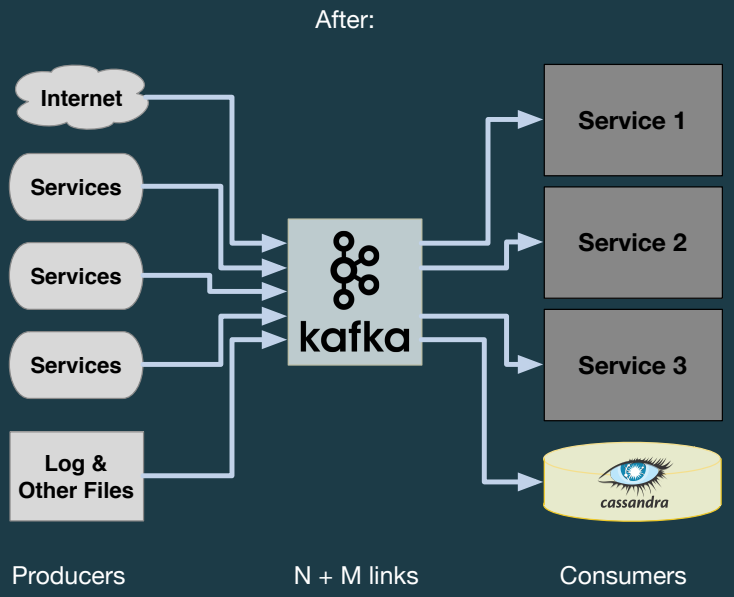
- Simplify dependencies between services
    - Improved data consistency
- Minimize data transmissions
- Reduce data loss when a service crashes



Why Kafka for Connectivity?

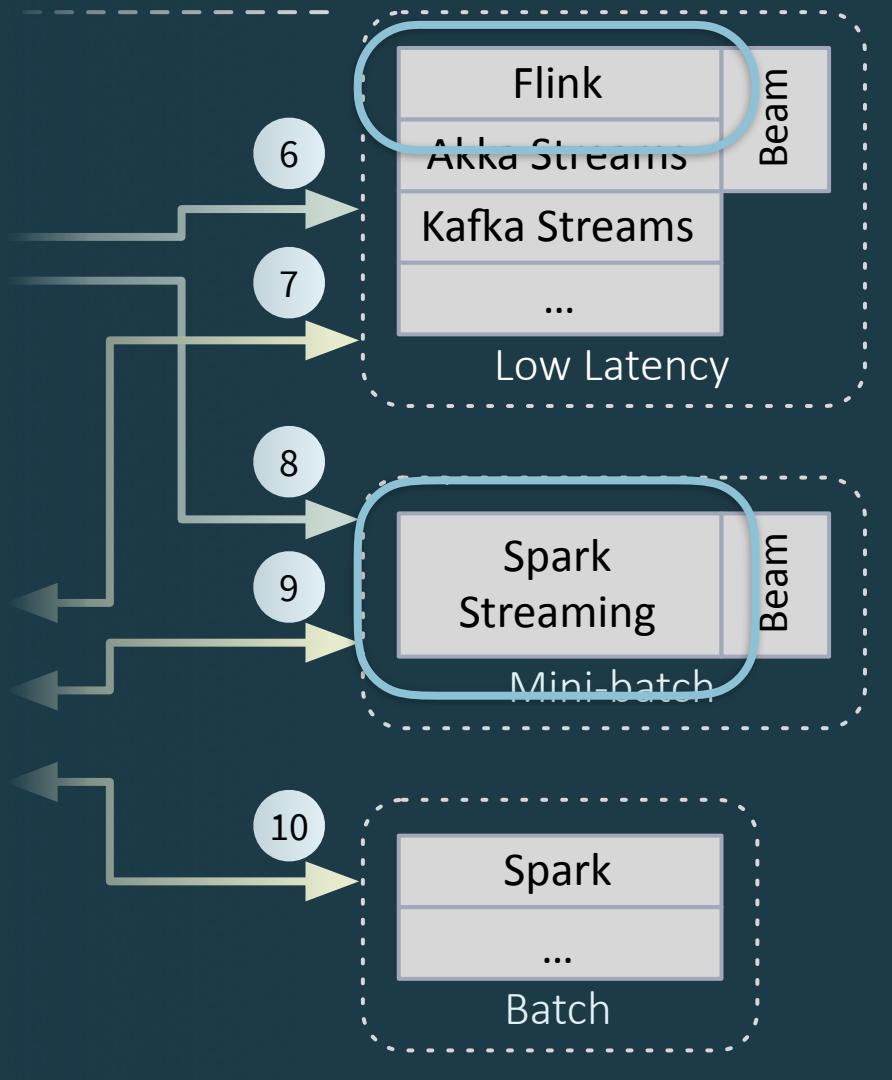
Kafka:

- M producers, N consumers
    - Improved extensibility
- Simplicity of one “API” for communication



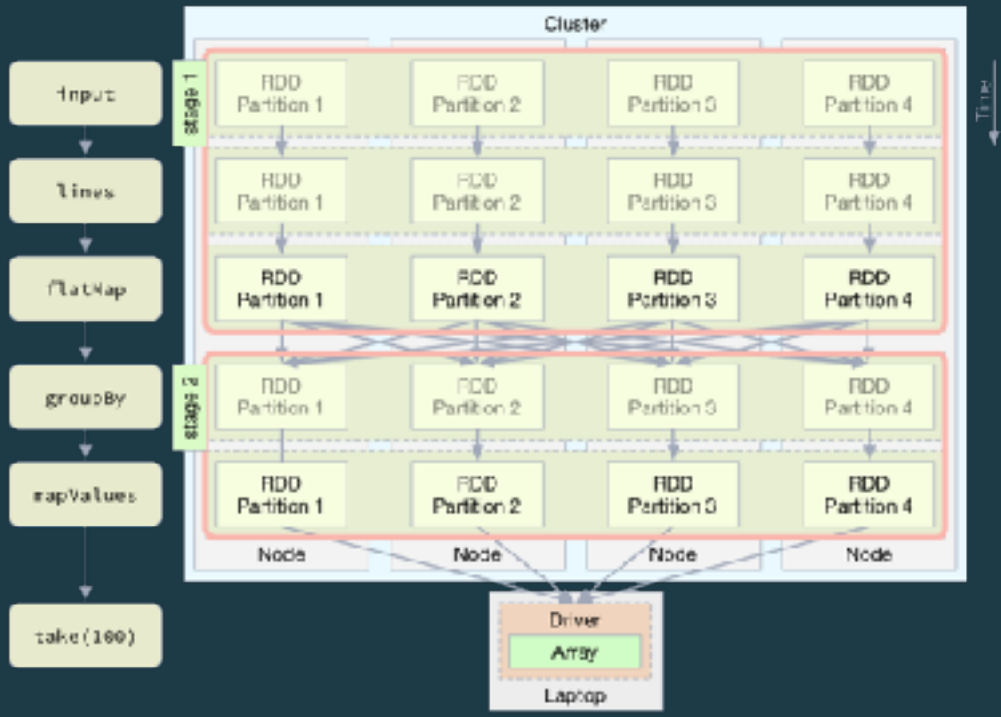
Streaming Engines:

Spark, Flink - services to which you submit work. Large scale, automatic data partitioning.



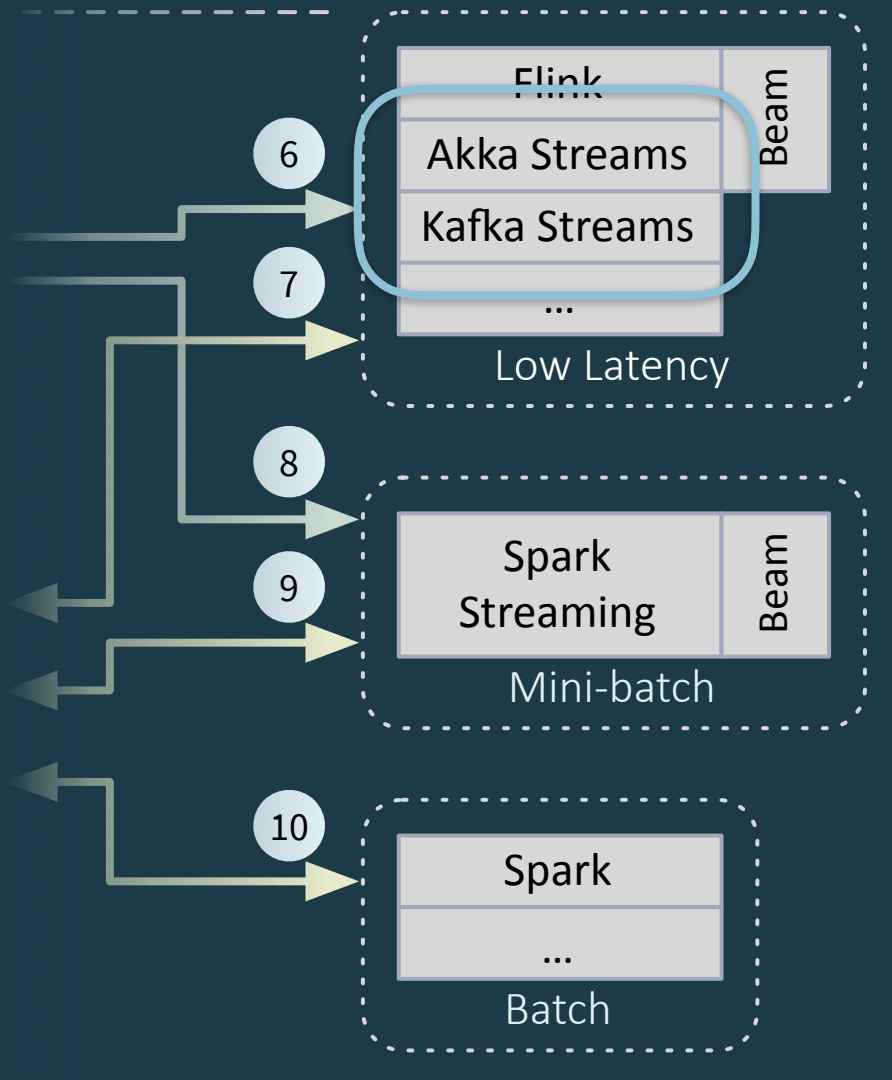
Streaming Engines:

Spark, Flink - services to which you submit work. Large scale, automatic data partitioning.



Streaming Frameworks:

Akka Streams, Kafka Streams - libraries/Frameworks for “data-centric micro services”. Smaller scale, but great flexibility



Microservice All the Things!

Scott Hanselman 

@shanselman

Follow

Microservices, for when your in-process methods have too little latency.

> **Dave Cheney** @davecheney
> Microservices, for when function calls are too reliable.

4:11 AM - 25 Feb 2018

207 Retweets **566** Likes

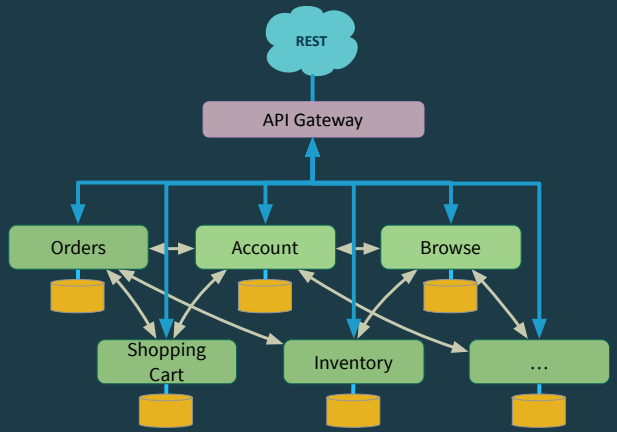
 25

 207

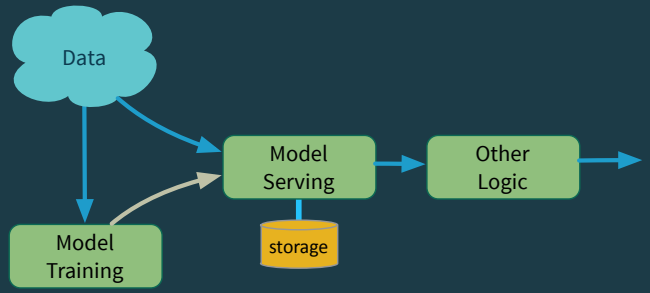
 566

A Spectrum of Microservices

Event-driven μ-services



“Record-centric” μ-services

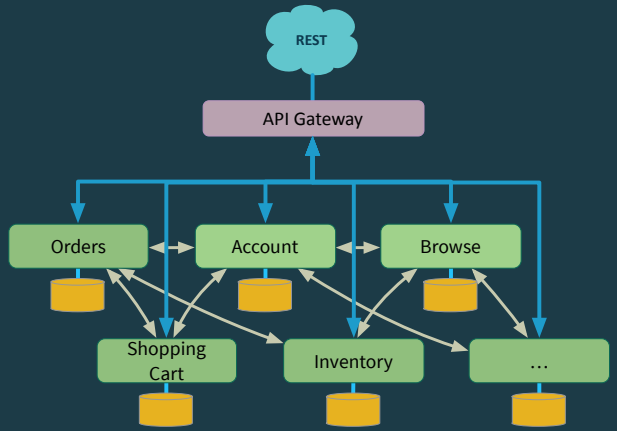


Events

Records

A Spectrum of Microservices

Event-driven μ-services



Akka emerged from the left-hand side of the spectrum, the world of highly *Reactive* microservices.

Akka Streams pushes to the right, more data-centric.

Events

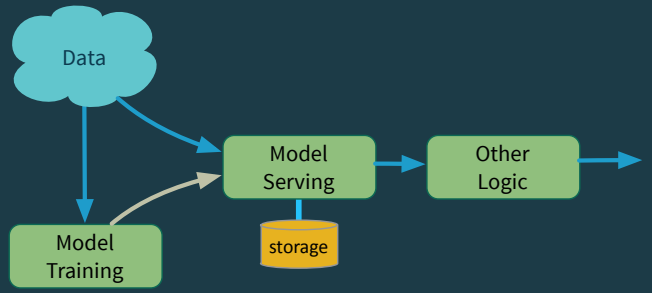
Records

A Spectrum of Microservices

Emerged from the right-hand side.

Kafka Streams pushes to the left, supporting many event-processing scenarios.

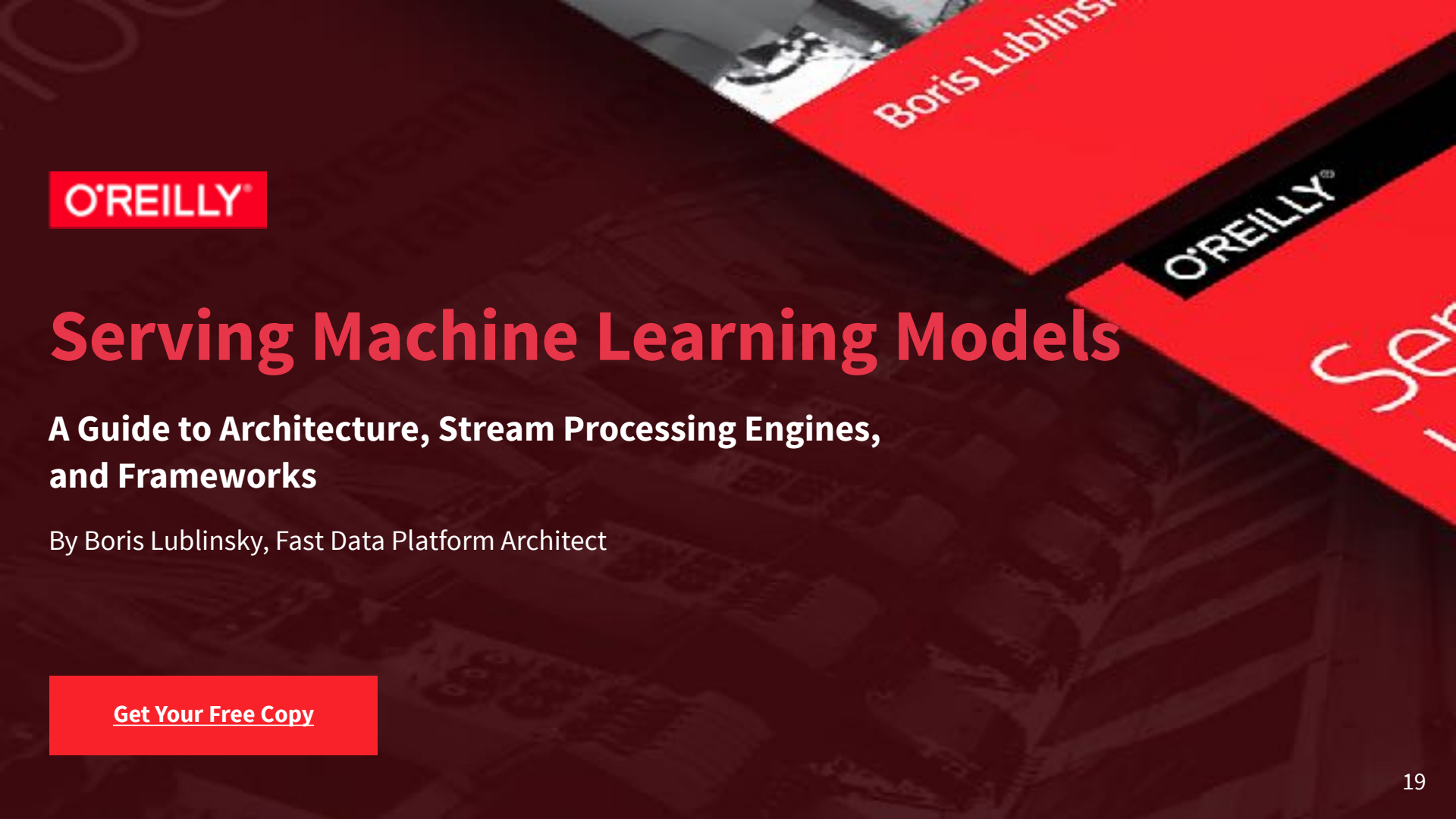
“Record-centric” μ-services



Events

Records

Machine Learning and Model Serving: A Quick introduction

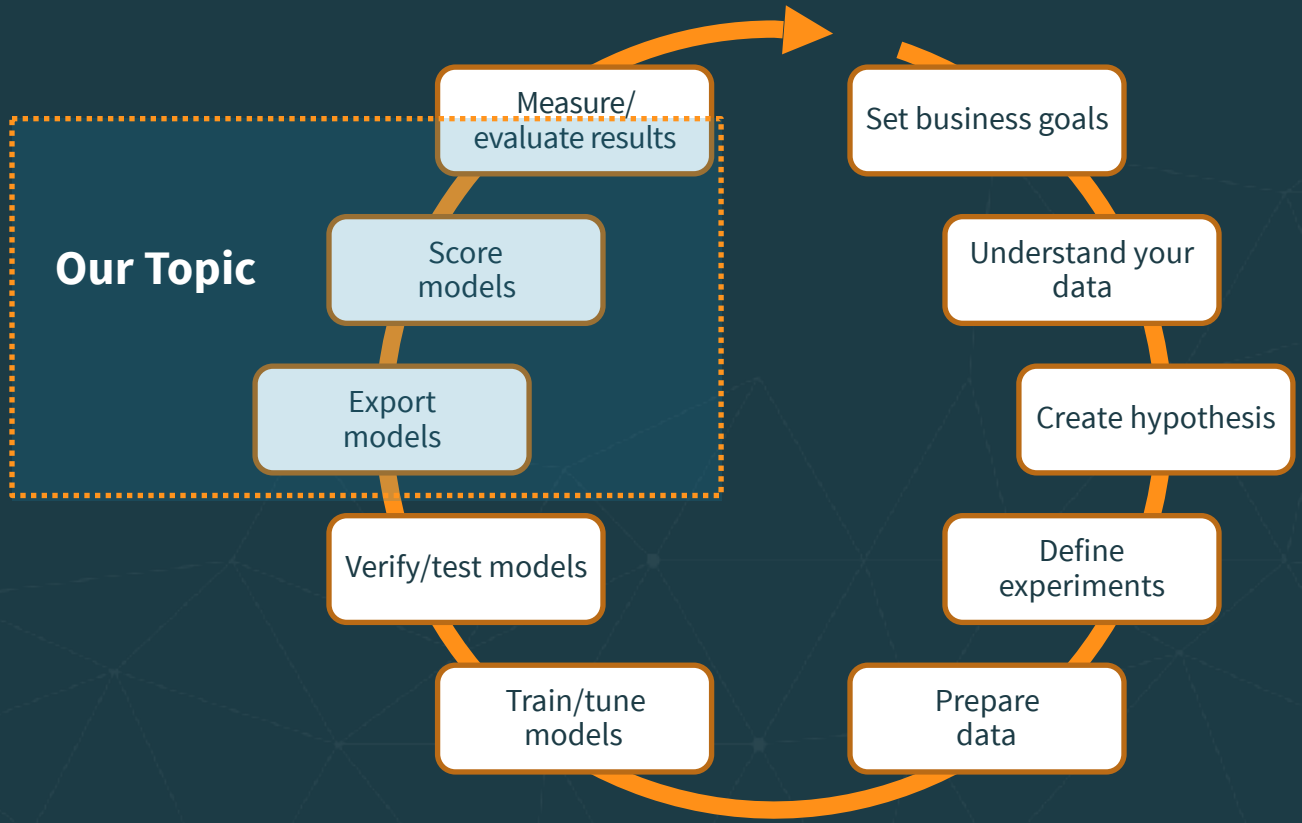

O'REILLY
Serving Machine Learning Models
A Guide to Architecture, Stream Processing Engines, and Frameworks
By Boris Lublinsky, Fast Data Platform Architect
Get Your Free Copy
Boris Lublinsky
O'REILLY

ML Is Simple

Maybe Not

Even If There Are Instructions

The Reality


Our Topic
Measure/ evaluate results
Score models
Export models
Verify/test models
Train/tune models
Set business goals
Understand your data
Create hypothesis
Define experiments
Prepare data

What Is The Model?

A model is a function transforming inputs to outputs - $y = f(x)$

for example:

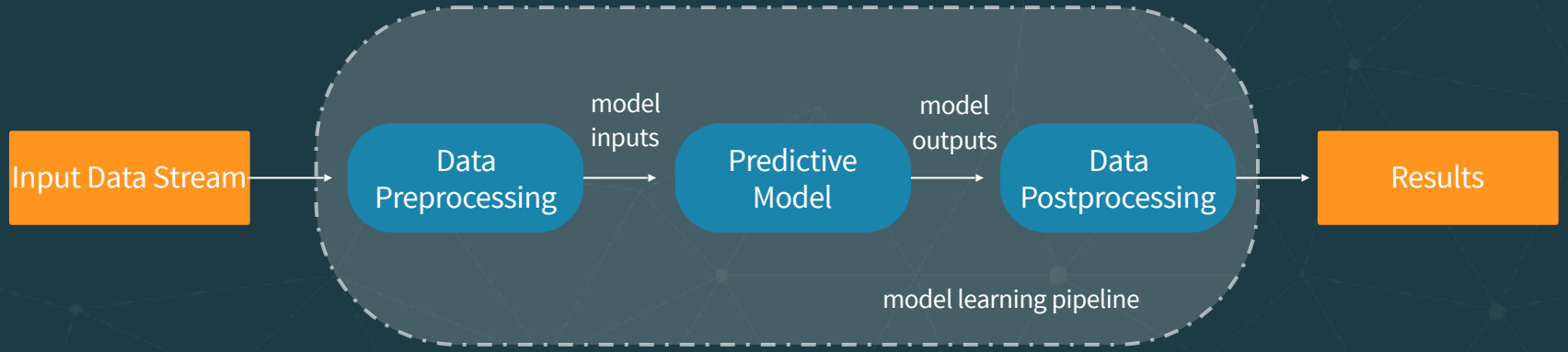
Linear regression: $y = a_c + a_1 * x1 + \ldots + a_n * x_n$

Neural network: $f(x) = K(\sum_i w_i g_i(x))$

Such a definition of the model allows for an easy implementation of model's composition. From the implementation point of view it is just function composition

Model Learning Pipeline

UC Berkeley AMPLab introduced machine learning pipelines as a graph defining the complete chain of data transformation.



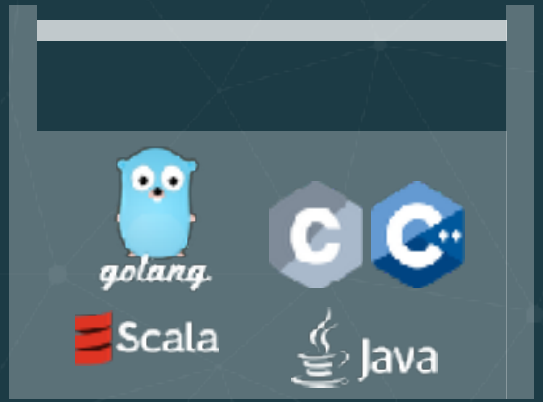
Traditional Approach To Model Serving

- Model is code
- This code has to be saved and then somehow imported into model serving

Why is this problematic?

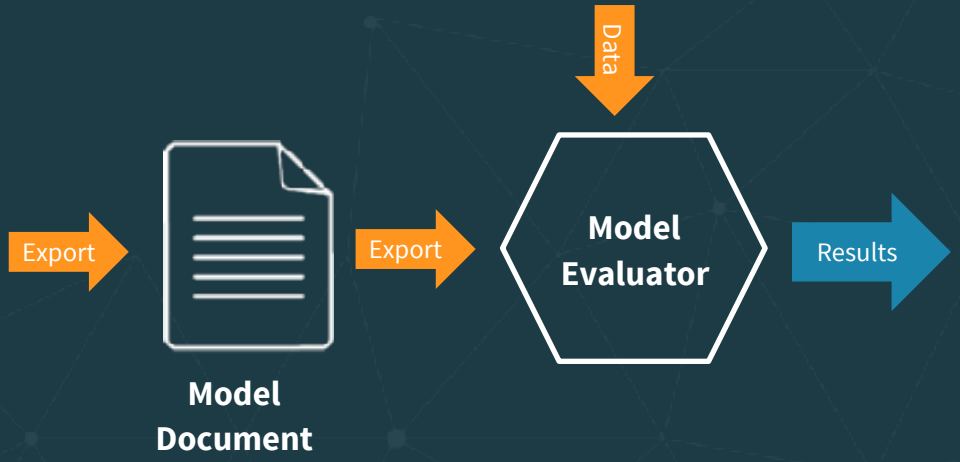
Impedance Mismatch

Continually expanding
Data Scientist toolbox



Defined Software
Engineer toolbox

Alternative - Model As Data



Standards

PMML Predictive Model Markup Language

Portable Format for Analytics **(PFA)**

TensorFlow

Exporting Model As Data With PMML

There are already a lot of export options

Spark MLlib https://github.com/jpmml/jpmml-sparkml

scikit learn https://github.com/jpmml/jpmml-sklearn

R https://github.com/jpmml/jpmml-r

TensorFlow https://github.com/jpmml/jpmml-tensorflow

Evaluating PMML Model

There are also a couple PMML evaluators

https://github.com/jpmml/jpmml-evaluator

https://github.com/opendatagroup/augustus

Exporting Model As Data With Tensorflow

- Tensorflow execution is based on Tensors and Graphs
- Tensors are defined as multilinear functions which consists of various vector variables
- A computational graph is a series of Tensorflow operations arranged into graph of nodes.
- Tensorflow support exporting of such graph in the form of binary protocol buffers.
- There are two different export format - optimized graph and a new format - saved model

Evaluating Tensorflow Model

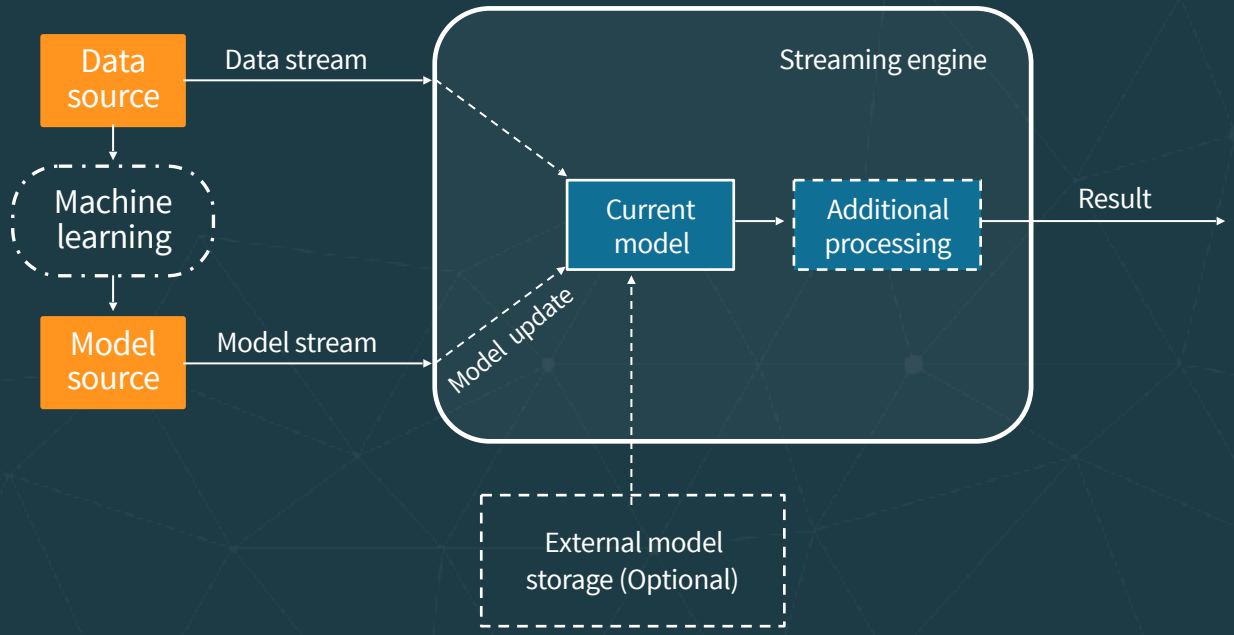
- Tensorflow is implemented in C++ with Python interface.
- In order to simplify Tensorflow usage from Java, in 2017 Google introduced Tensorflow Java APIs.
- Tensorflow Java APIs supports import of the exported model and allows to use them for scoring.

Additional Considerations – Model Lifecycle

- Models tend to change
- Update frequencies vary greatly – from hourly to quarterly/yearly
- Model version tracking
- Model release practices
- Model update process

The Solution

A streaming system allowing to update models without interruption of execution (dynamically controlled stream).



Model Representation (Protobufs)

```
// On the wire
syntax = “proto3”;
// Description of the trained model.
message ModelDescriptor {
  string name = 1;    // Model name
  string description = 2;    // Human readable
  string dataType = 3;  // Data type for which this model is applied.
  enum ModelType {    // Model type
    TENSORFLOW  = 0;
    TENSORFLOWSAVED = 2;
    PMML = 2;
  };
  ModelType modeltype = 4;
  oneof MessageContent {
    // Byte array containing the model
    bytes data = 5;
    string location = 6;
  }
}
```

Model Representation (Scala)

```
trait Model {
  def score(input : AnyVal) : AnyVal
  def cleanup() : Unit
  def toBytes() : Array[Byte]
  def getType : Long
}

def ModelFactoryl {
  def create(input : ModelDescriptor) : Model
  def restore(bytes : Array[Byte]) : Model
}
```

Additional Considerations: Monitoring

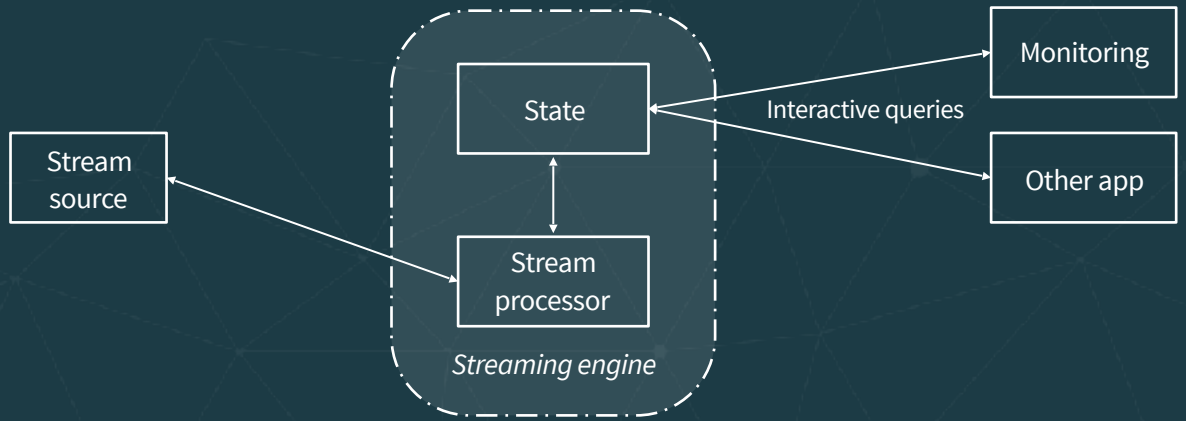
Model monitoring should provide information about usage, behavior, performance and lifecycle of the deployed models

```
case class ModelToServeStats(
name: String,                                  //  Model name
     description: String,                      // Model descriptor
     modelType: ModelDescriptor.ModelType,     // Model type
     since : Long,                             // Start time of model usage
     var usage : Long = 0,                     // Number of servings
     var duration : Double = 0.0,              // Time spent on serving
     var min : Long = Long.MaxValue,           // Min serving time
     var max : Long = Long.MinValue            // Max serving time
)
```

Queryable State

Queryable state: ad hoc query of the state in the stream. Different than the normal data flow.

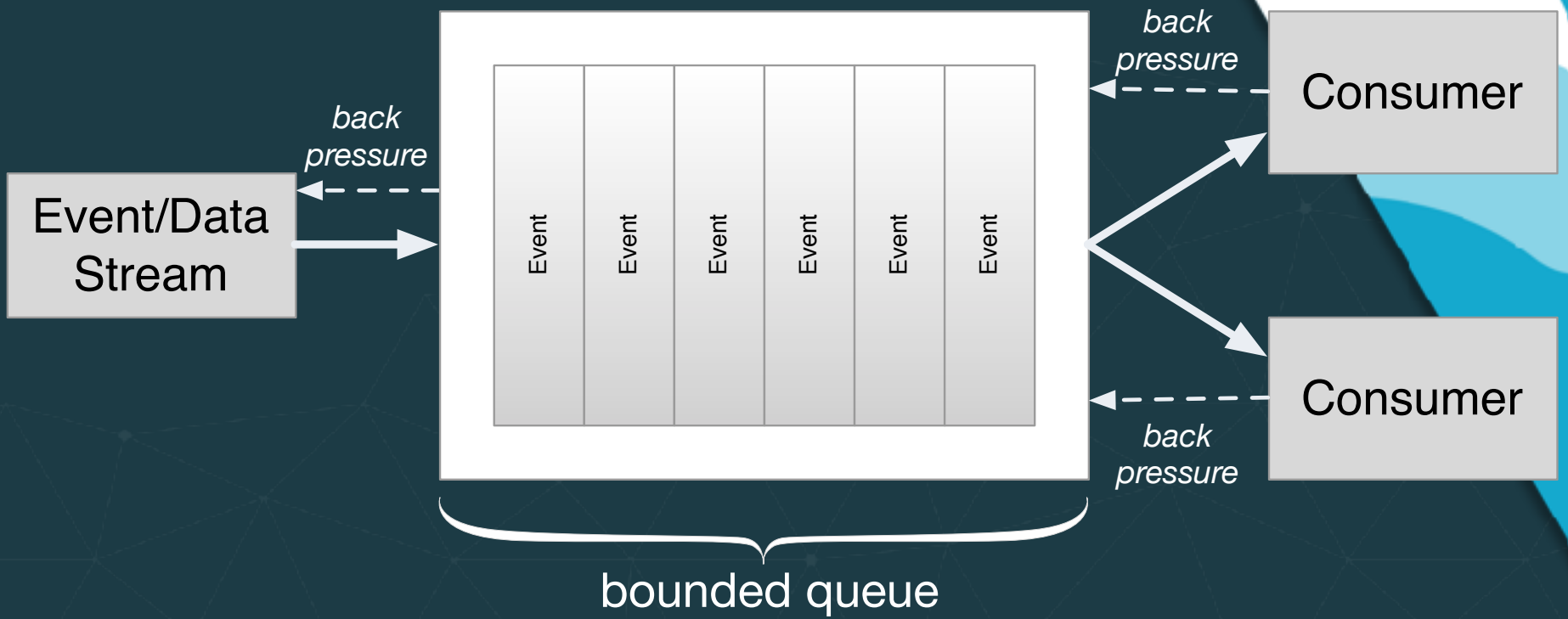
Treats the stream processing layer as a lightweight embedded *database. Directly query the current state* of a stream processing application. No need to materialize that state to a database, etc. first.



- We'll work with Akka Streams examples first

- A *library*
- Implements Reactive Streams.
  - http://www.reactive-streams.org/
  - *Back pressure* for flow control

akka streams



- Part of the Akka ecosystem
  - Akka Actors, Akka Cluster, Akka HTTP, Akka Persistence, …
  - Alpakka - rich connection library
    - like Camel, but implementing reactive streams
  - Commercial support from Lightbend

```
import akka.stream._
import akka.stream.scaladsl._
import akka.{ NotUsed, Done }
import akka.actor.ActorSystem
import scala.concurrent._
import scala.concurrent.duration._

implicit val system = ActorSystem("QuickStart")
implicit val materializer = ActorMaterializer()

val source: Source[Int, NotUsed] = Source(1 to 10)
val factorials = source.scan(BigInt(1)) ( (acc, next) => acc * next )
factorials.runWith(Sink.foreach(println))
```

```
import akka.stream._
import akka.stream.scaladsl._
import akka.{ NotUsed, Done }
import akka.actor.ActorSystem
import scala.concurrent._
import scala.concurrent.duration._
```

Imports!

```
implicit val system = ActorSystem("QuickStart")
implicit val materializer = ActorMaterializer()

val source: Source[Int, NotUsed] = Source(1 to 10)
val factorials = source.scan(BigInt(1)) ( (acc, next) => acc * next )
factorials.runWith(Sink.foreach(println))
```

```
import akka.stream._
import akka.stream.scaladsl._
import akka.{ NotUsed, Done }
import akka.actor.ActorSystem
import scala.concurrent._
import scala.concurrent.duration._

implicit val system = ActorSystem("QuickStart")
implicit val materializer = ActorMaterializer()

val source: Source[Int, NotUsed] = Source(1 to 10)
val factorials = source.scan(BigInt(1)) ( (acc, next) => acc * next )
factorials.runWith(Sink.foreach(println))
```

Initialize and specify now the stream is “materialized”

```
import akka.stream._
import akka.stream.scaladsl._
import akka.{ NotUsed, Done }
import akka.actor.ActorSystem
import scala.concurrent._
import scala.concurrent.duration._

implicit val system = ActorSystem("QuickStart")
implicit val materializer = ActorMaterializer()

val source: Source[Int, NotUsed] = Source(1 to 10)
val factorials = source.scan(BigInt(1)) ( (acc, next) => acc * next )
factorials.runWith(Sink.foreach(println))
```

Create a Source. Scan it and compute factorials, output to a Sink, and run it.

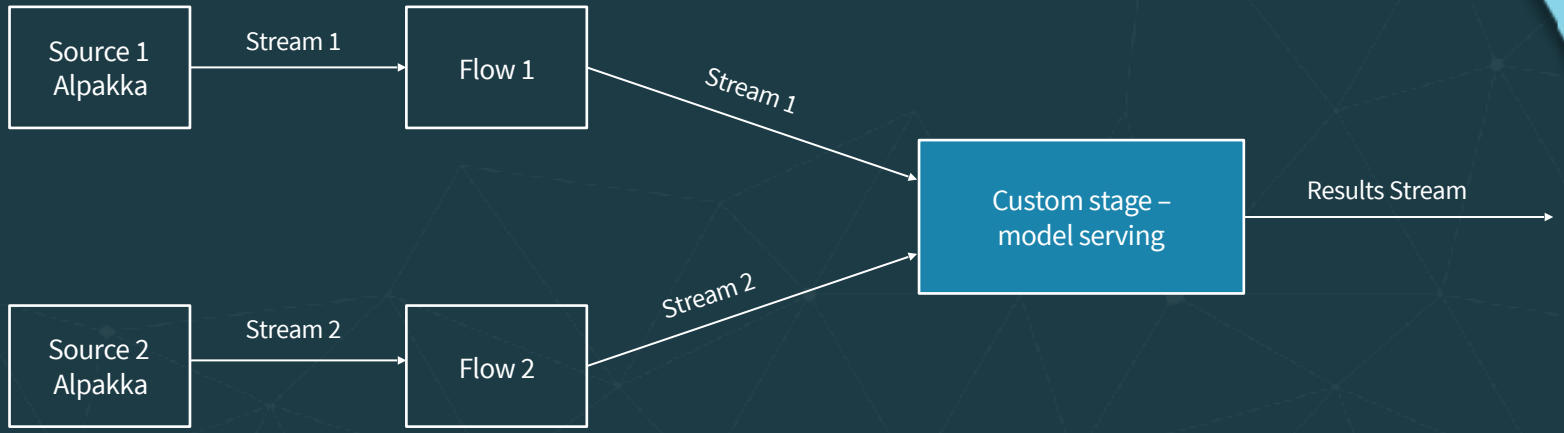
```
import akka.stream._
import akka.stream.scaladsl._
import akka.{ NotUsed, Done }
import akka.actor.ActorSystem
import scala.concurrent._
import scala.concurrent.duration._

implicit val system = ActorSystem("QuickStart")
implicit val materializer = ActorMaterializer()

val so      e[Int, No      urce(1 to
val fa      urce.      (acc      next )
facto      n(Sink.fo      ln))
```

Source → Flow → Sink

Create a Source. Scan it and compute factorials, output to a Sink, and run it.

Using Custom Stage

Create a custom stage, a fully type-safe way to encapsulate new functionality.



Using a Custom Stage

Code time

1. Walk through the whole tutorial Project
2. Run the *client* project
   - Creates in-memory Kafka instance and our topics
   - Pumps data into them
3. Explore and run *akkaStreamsCustomStage* project

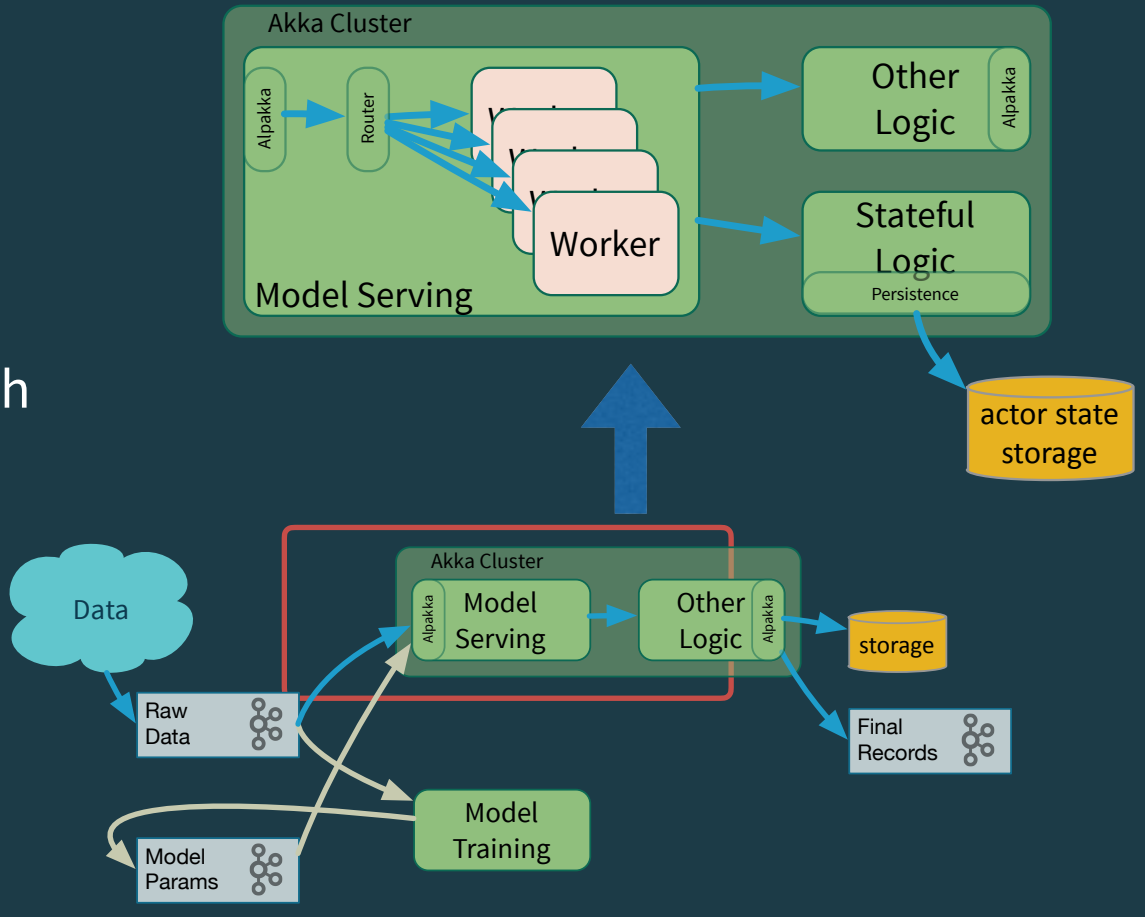
Exercises!

We've prepared some exercises. We may not have time during the course to work on them, but take a look at the *exercise* branch in the Git project (or the separate X.Y.Z_exercise download).

To find them, search for "// Exercise". The *master* branch implements the solutions.

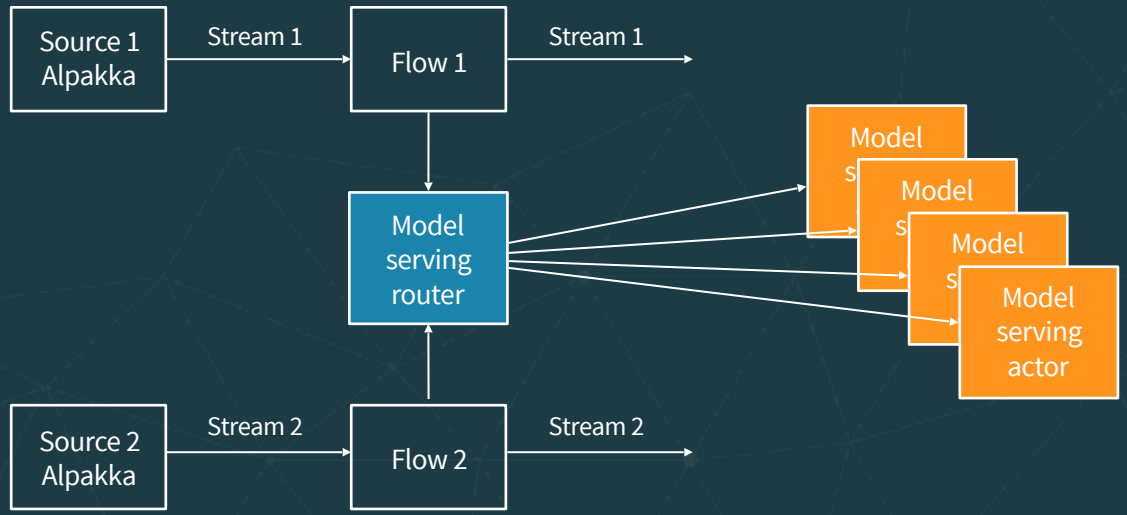
Other Production Concerns

- Scale scoring with workers and routers, across a cluster
- Persist actor state with Akka Persistence
- Connect to *almost* anything with Alpakka
- *Lightbend Enterprise Suite*
  - for production



Improve Scalability for Model Serving

Use a router actor to forward requests to the actor responsible for processing requests for a specific model type.



Akka Streams with Actors and Persistence

Code time

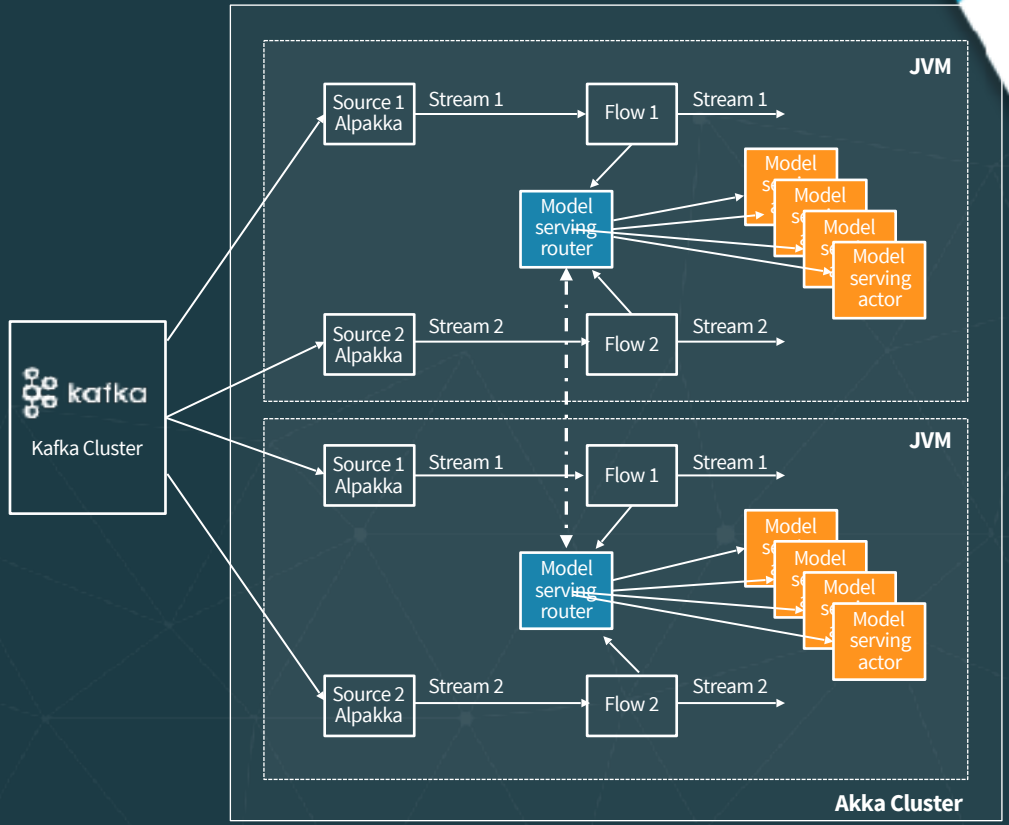
1. While still running the *client* project…
2. Explore and run *akkaActorsPersistent* project

More Production Concerns

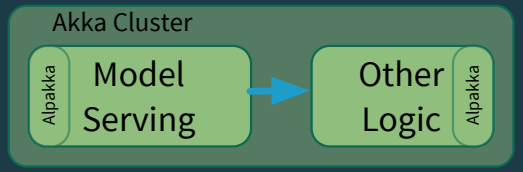
Using Akka Cluster

Two levels of scalability:

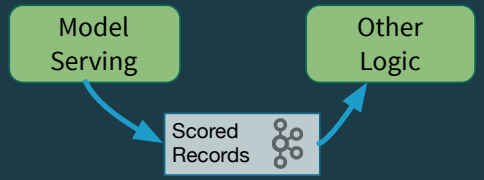
- Kafka partitioned topic allow to scale listeners according to the amount of partitions.
- Akka cluster sharing allows to split model serving actors across clusters.



Go Direct or Through Kafka?



vs.

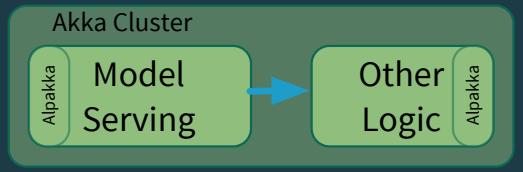


?

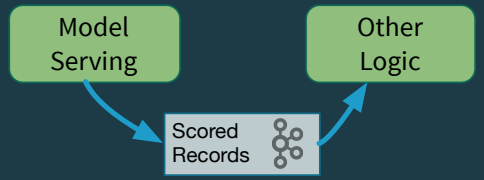
- Extremely low latency
- Minimal I/O and memory overhead
- No marshaling overhead

- Higher latency (including queue depth)
- Higher I/O and processing (marshaling) overhead
- Better potential reusability

Go Direct or Through Kafka?



vs.



?

- *Reactive Streams* back pressure
- Direct coupling between sender and receiver, but indirectly through a URL

- Very deep buffer (partition limited by disk size.
- Strong decoupling - M producers, N consumers, completely disconnected

Kafka Streams

- Sample use case, now with Kafka Streams

Kafka Streams

- Important stream-processing concepts, e.g.,
  - Distinguish between *event time* and *processing time*
  - Windowing support.
  - For more on these concepts, see
    - Dean’s book ;)
    - Talks, blog posts, writing by Tyler Akidau

Kafka Streams

- KStream - per-record transformations
- KTable - aggregations, last value per key
  - Efficient management of application state

Kafka Streams

- Two types of APIs:
  - Process Topology (compare to Apache Storm)
  - DSL based on collection transformations
    - Compare to Spark, Flink, Scala collections.

Kafka Streams

- Provides Java API
- Lightbend donating a Scala API
  - https://github.com/lightbend/kafka-streams-scala
  - See also our convenience tools for distributed, queryable state: https://github.com/lightbend/kafka-streams-query
- SQL!

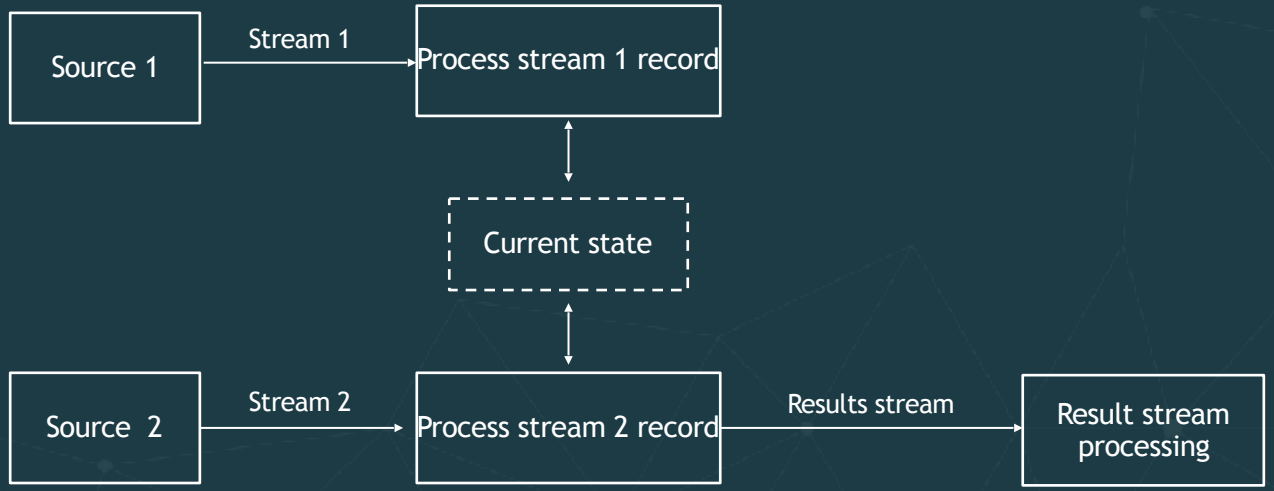
Kafka Streams

- Low overhead
- Read from and write to Kafka topics, memory
  - Could use Kafka Connect for other sources and sinks
- Load balance and scale based on partitioning of topics
- Built-in support for Queryable State

Kafka Streams

- Ideally suited for:
  - ETL -> KStreams
  - Aggregations -> KTable
  - Joins, including Stream and Table joins
  - “Effectively once” semantics
- Commercial support from Confluent, Lightbend, and others

Model Serving With Kafka Streams



State Store Options We'll Explore

- "Naive", in memory store
- Built-in key/value store provided by Kafka Streams
- Custom store

Model Serving With Kafka Streams

Code time

1. Still running the *client* project
2. Explore and run:
   kafkaStreamsModelServerInMemoryStore

Model Serving With Kafka Streams, KV Store

Code time (as time permits)

1. Still running the *client* project
2. Explore and run:
   kafkaStreamsModelServerKVStore

Model Serving With Kafka Streams, KV Store

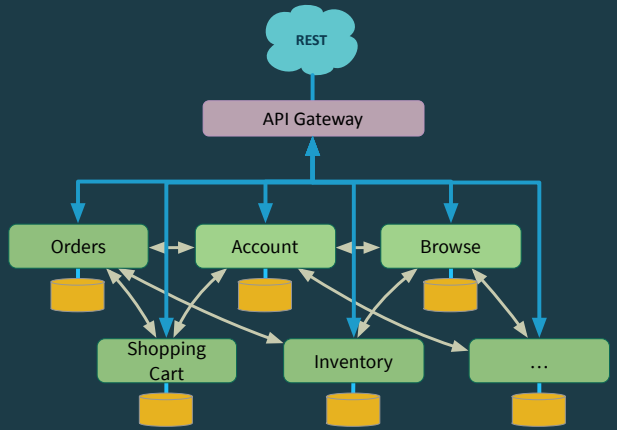
Code time (as time permits)

1. Still running the *client* project
2. Explore and run:
   kafkaStreamsModelServerCustomStore

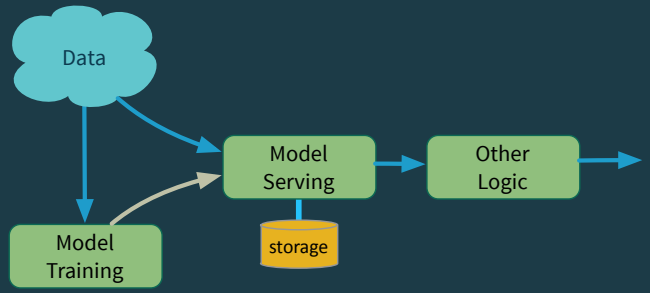
To Wrap Up

akka

Event-driven μ-services



“Record-centric” μ-services



Events

Records

Thank You

Questions?

- AMA, with Dean and Boris
  - Thursday 11:50 - 12:30, 212 A-B
- Meet the Expert, with Dean
  - Thursday 11:50 - 12:30, Expo Hall
- Kafka streaming applications with Akka Streams and Kafka Streams, with Dean
  - Thursday 11:00 - 11:40, Expo Hall 1
- Approximation data structures in streaming data processing, with Debasish Ghosh
  - Wednesday 1:50 - 2:30, 230A
- Machine-learned model quality monitoring in fast data and streaming applications, Emre Velipasaoglu
  - Thursday 1:50 - 2:30, LL21 C/D

https://www.lightbend.com/products/fast-data-platform
boris.lublinsky@lightbend.com
dean.wampler@lightbend.com

Lightbend