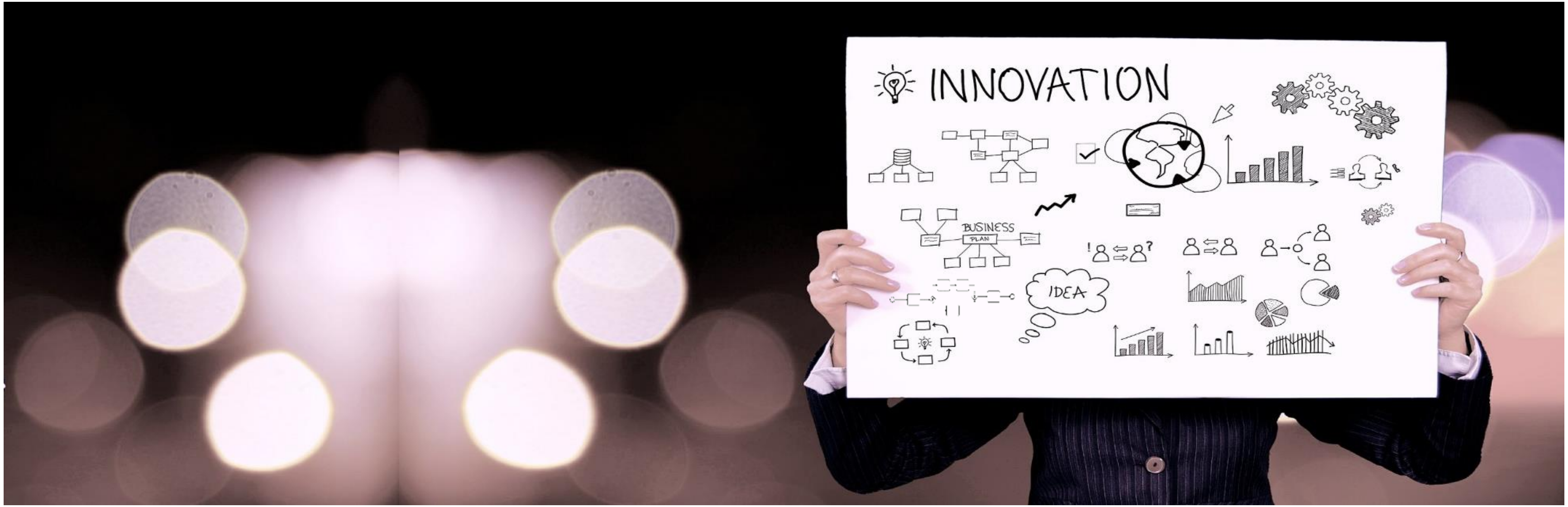
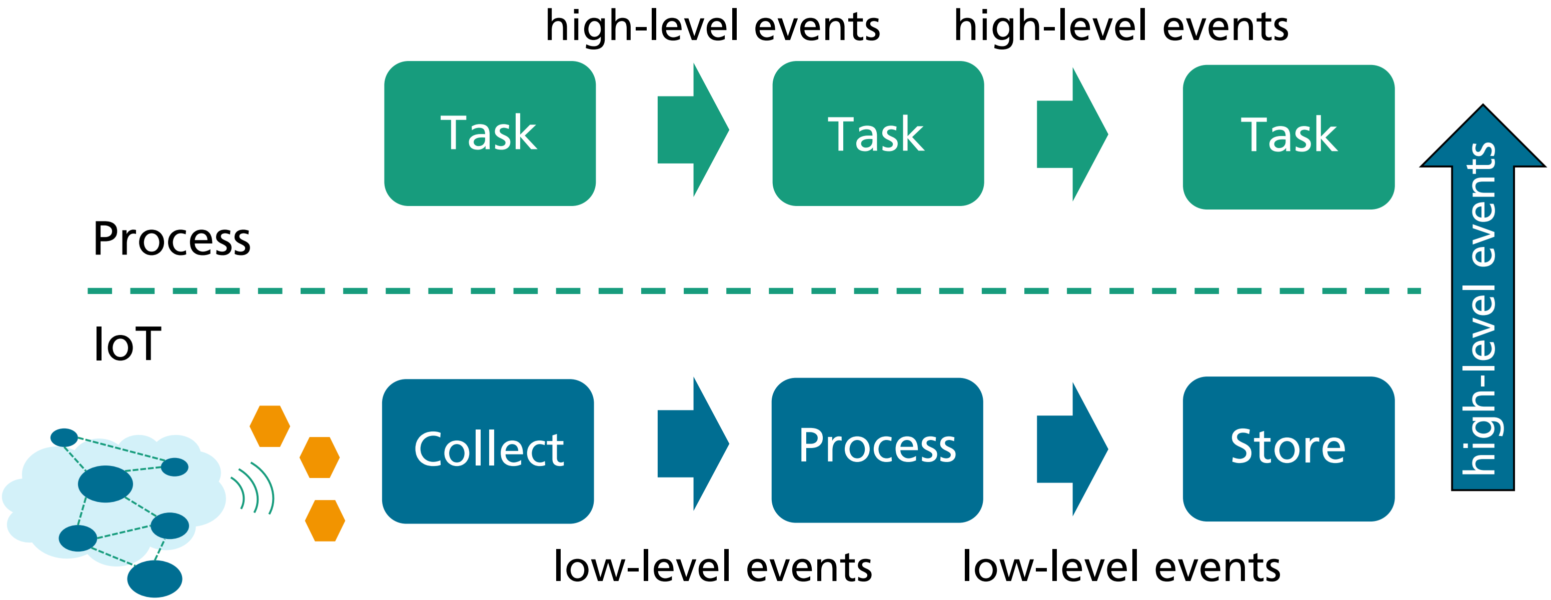
IOT ANALYTICS PLATFORM ON TOP OF SMACK

Yevgen Pikus | February 24th, 2017 | Berlin



Fraunhofer
ISST

Motivation – Connecting IoT & Proceses Layers



high-level events high-level events

Task → Task → Task

Process

IoT

Collect → Process → Store

low-level events low-level events

high-level events

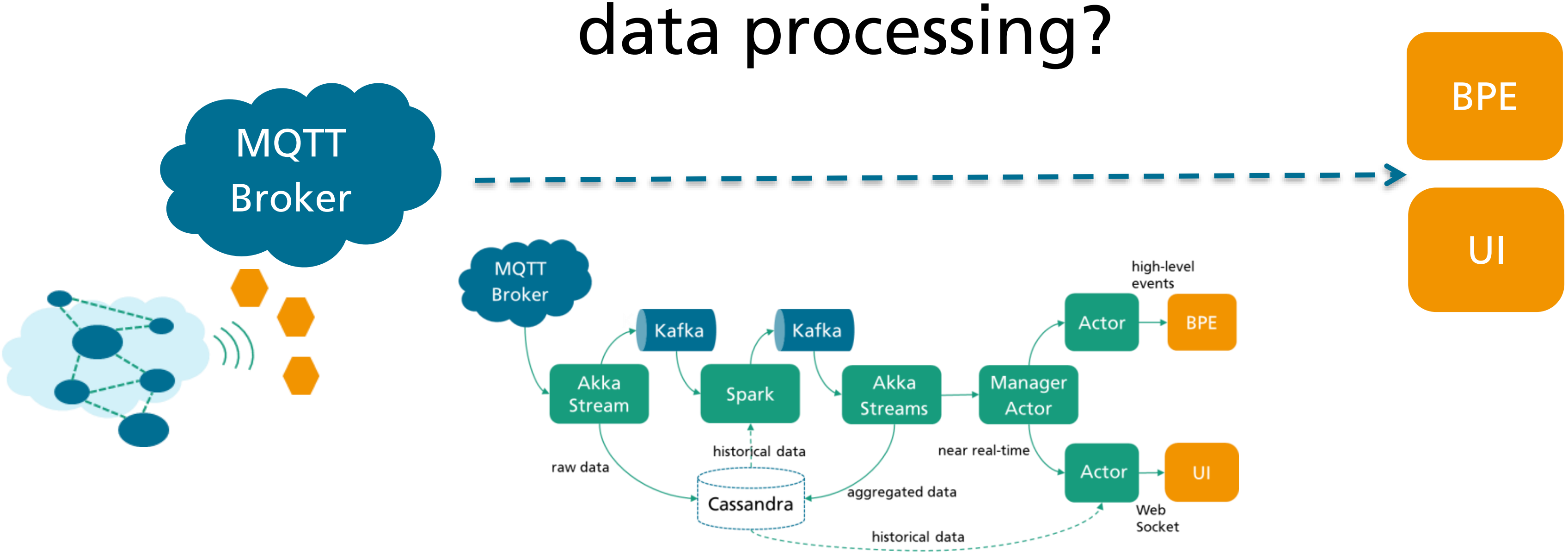
Scenario – Predictive Maintenance

- Vibration data is continuously measured on different parts of a machine
- Sensor data is collected and analyzed
- Prediction of a failure triggers the maintenance process
- Visualization of data

pixabay,com

Data Flow from IoT to Business

Near real-time data processing?



SMACK

APACHE Spark

- Is a fast large-scale **data processing engine**
- Provides an interface for programming entire clusters with **implicit data parallelism** and fault-tolerance.

MESOS

- Is built using the same principles as the **Linux kernel**, only at a different level of abstraction
- Runs on every machine and provides applications with API's for **resource management and scheduling** across entire datacenter and cloud environments

akka

- Is a toolkit and runtime simplifying the construction of **concurrent and distributed applications** on the JVM

cassandra

- Is a **distributed database** designed to handle large amounts of data, providing high availability with **no single point of failure**

- Is a **message broker** that provides a unified, **high-throughput**, low-latency platform for handling real-time data feeds

en.wikipedia.org

Actor Model

The actor model in computer science is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation. In response to a message that it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next message received. Actors may modify private state, but can only affect each other through messages.

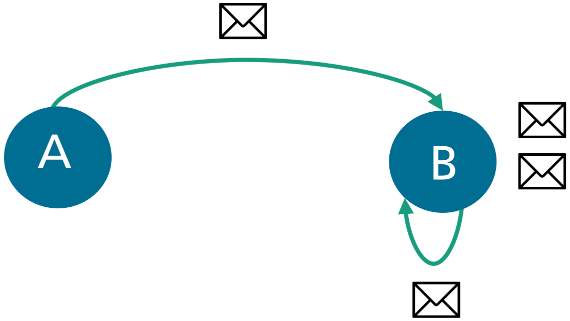
– Wikipedia

Wikipedia

Akka Actors

- Actor
  - **Encapsulates** state and behavior
  - Sends and receive **messages**
  - **Creates** new Actors
  - Is **location transparent**

- Akka
  - Toolkit for highly **concurrent, distributed**, and **resilient message-driven** applications on the JVM
  - **Millions** of **messages** per second
  - Akka Cluster, Akka HTTP, Akka Persistence, Akka Streams



Fraunhofer
ISST

Reactive Streams

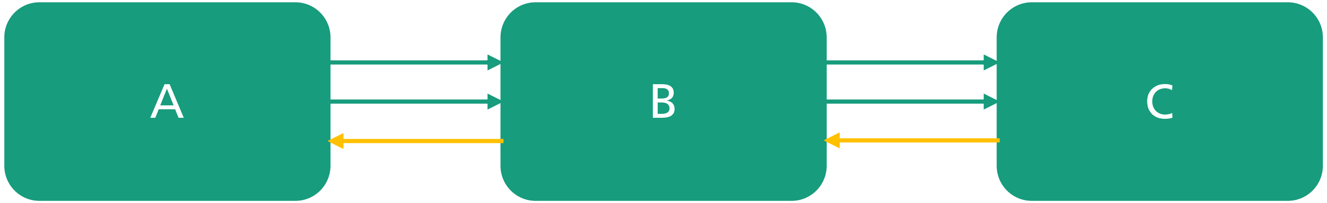
> Reactive Streams is an initiative to provide standard for **asynchronous** stream processing with non-blocking **back-pressure**.
>
> – Wikipedia

Wikipedia

What is back-pressure?

slow Publisher and fast Subscriber

fast Publisher and slow Subscriber

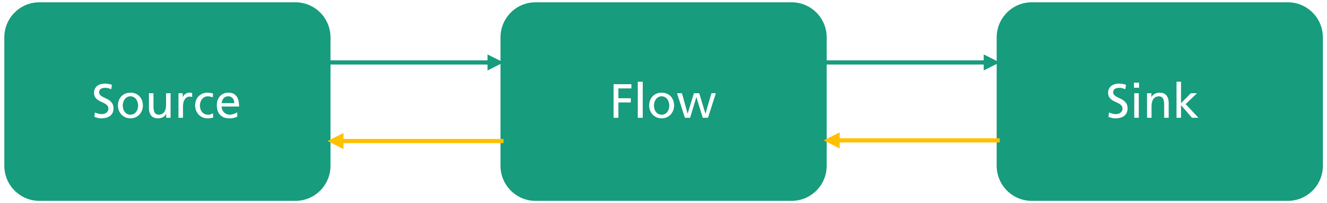


Akka Streams

Asynchronous back-pressured stream processing

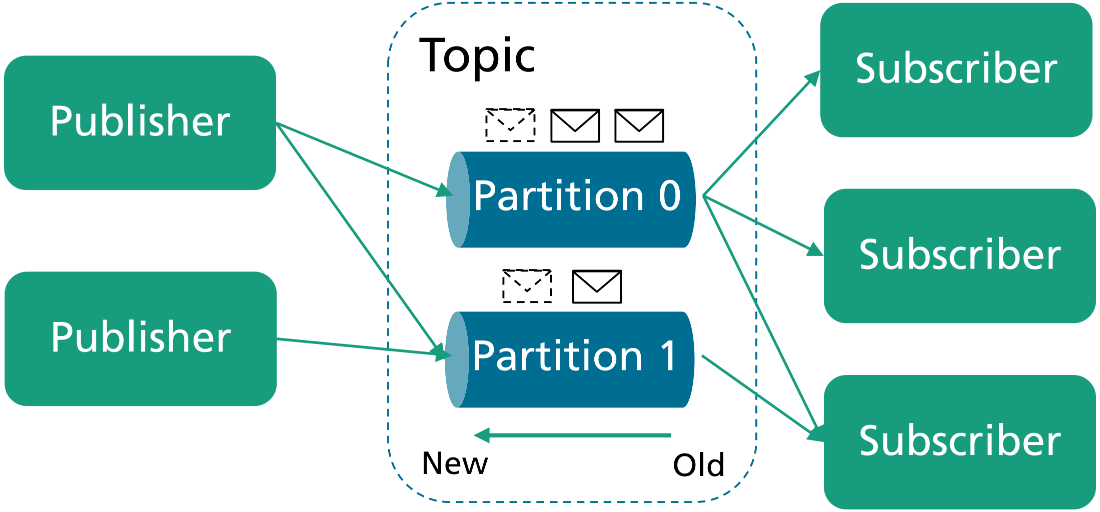
Complex structured stream flows

Integration with Akka Actors

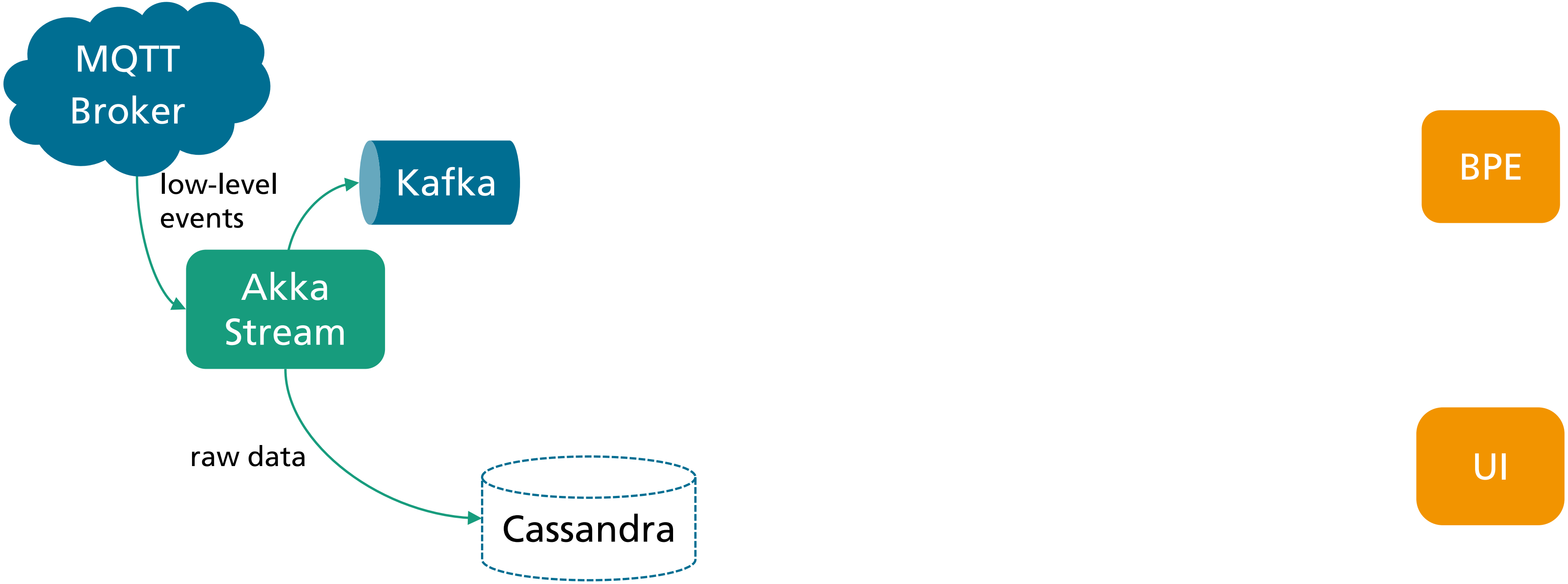


Kafka

- Publisher / subscriber messaging model
- Batching
- Durability
- Horizontally scalable
- Very high throutput
- Replication



Data Flow from IoT to Business



Collect and process Sensor Data

Scala DSL for Akka Streams

```
val g = RunnableGraph.fromGraph(GraphDSL.create(){implicit builder: GraphDSL.Builder[NotUsed] =>
  import GraphDSL.Implicits._

  val mqttSource: Source[MqttMessage, Future[Done]] = MqttSource(settings, bufferSize = 8)
  val transform = Flow[MqttMessage].map(m => m.payload.utf8String)
  val broadcast = builder.add(Broadcast[String](2))
  val validate = Flow[String].filter(m => isValid(m))
  val toProducerRecord = Flow[String].map(m => new ProducerRecord[String, String](topic, m))
  val producerSink = Producer.plainSink(producerSettings)
  val parse = Flow[String].map(m => parseMessage(m))
  val cassandraSink = CassandraSink[SensorRecord](parallelism = 1, preparedStatement,
           statementBinder)

  mqttSource ~> transform ~> validate ~>  broadcast ~> toProducerRecord ~> producerSink
  broadcast ~> parse ~> cassandraSink

  ClosedShape
})
```

Flow ops

Processing graph definition

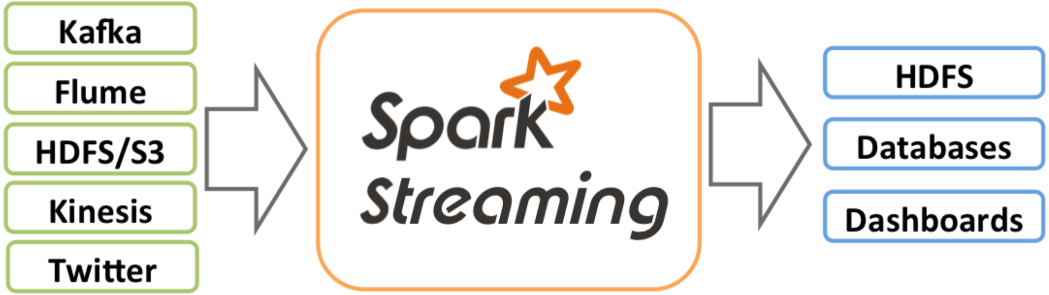
Spark Streaming

Scalable, fault-tolerant near real-time stream processing

Programming and infrastructure abstraction

Ecosystem: Spark SQL, Spark MLib, Spark GraphX

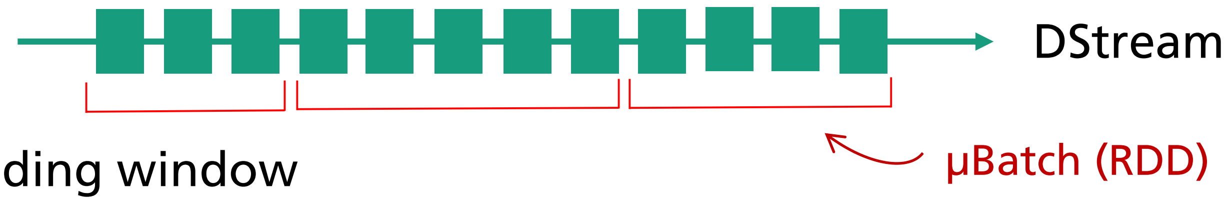
APIs: Scala, Java, Python, R



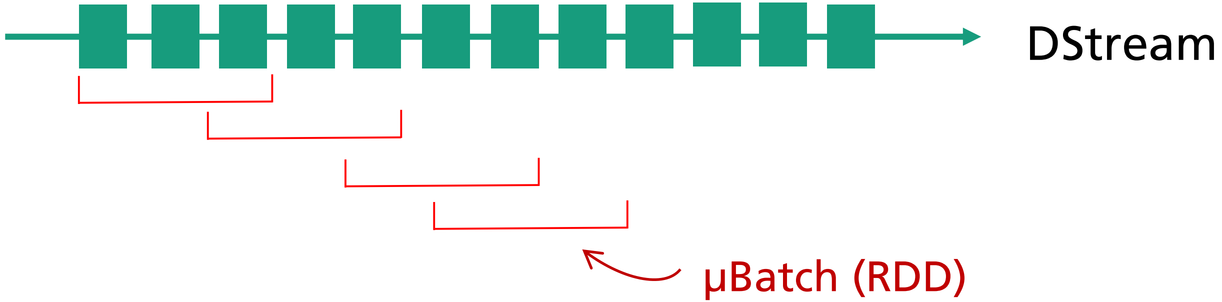
http://spark.apache.org/

Spark Streaming

- Tumbling window

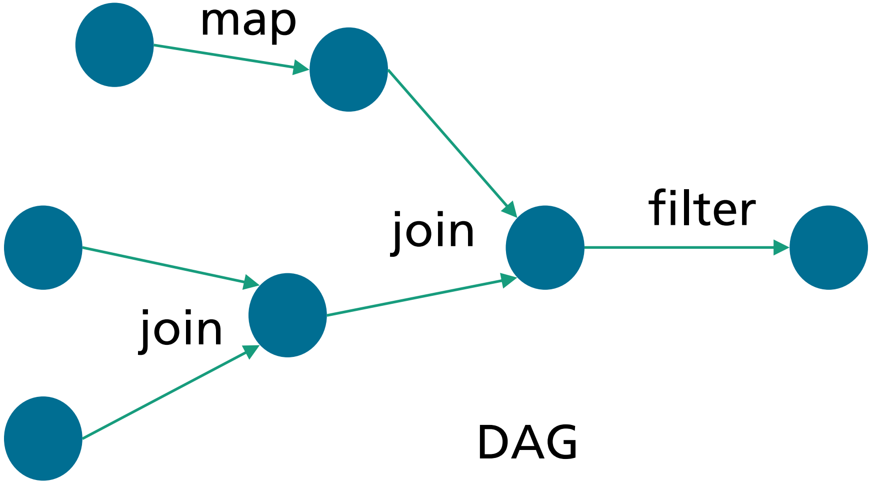


- Sliding window



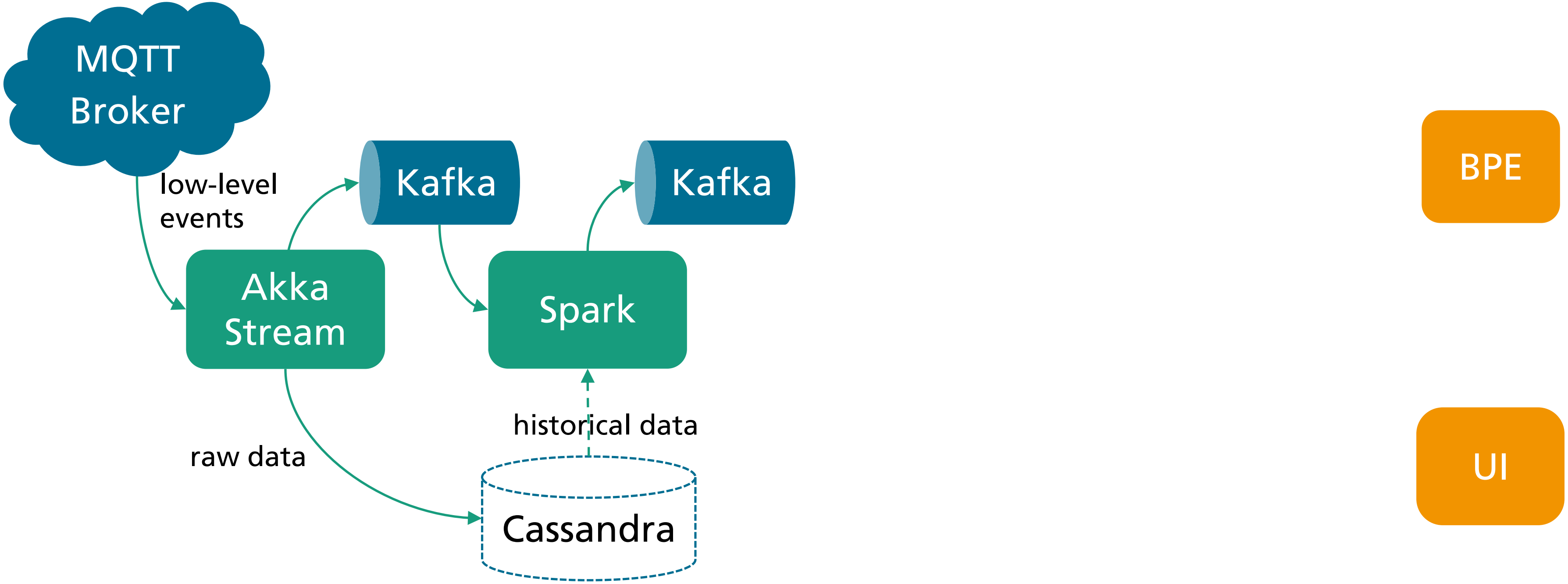
Higher-level API (DStream)

- map(*func*)
- flatMap(*func*)
- filter(*func*)
- repartition(*numPartitions*)
- union(*otherStream*)
- count()
- reduce(*func*)
- countByValue()
- reduceByKey(*func*, [*numTasks*])
- join(*otherStream*, [*numTasks*])
- etc.



DAG

Data Flow from IoT to Business



Predictive Maintenance in Spark

```
val dStream = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](
  ssc, kafkaParams, sensors)

val recordsStream: DStream[(Long, List[SensorRecord])] = dStream
  .flatMap(m =>  parse(m._2))
  .map(r => (r.sensorId, List(r)) )
  .reduceByKey((s1,s2) => s1 ::: s2)

recordsStream.foreachRDD{ rdd =>
  rdd.foreachPartition(recordsIterator => {
    val producer = new KafkaProducer[String, String](producerConf)
    recordsIterator.foreach{records =>
      val transformedRecords = fft(records._2)
      val state = similaritySearch(transformedRecords)
      val inform = SensorInformation(state, transformedRecords)
      val message = new ProducerRecord[String, SensorInformation]("SensorInformation", inform)
      producer.send(message)
    }
  })
}
```

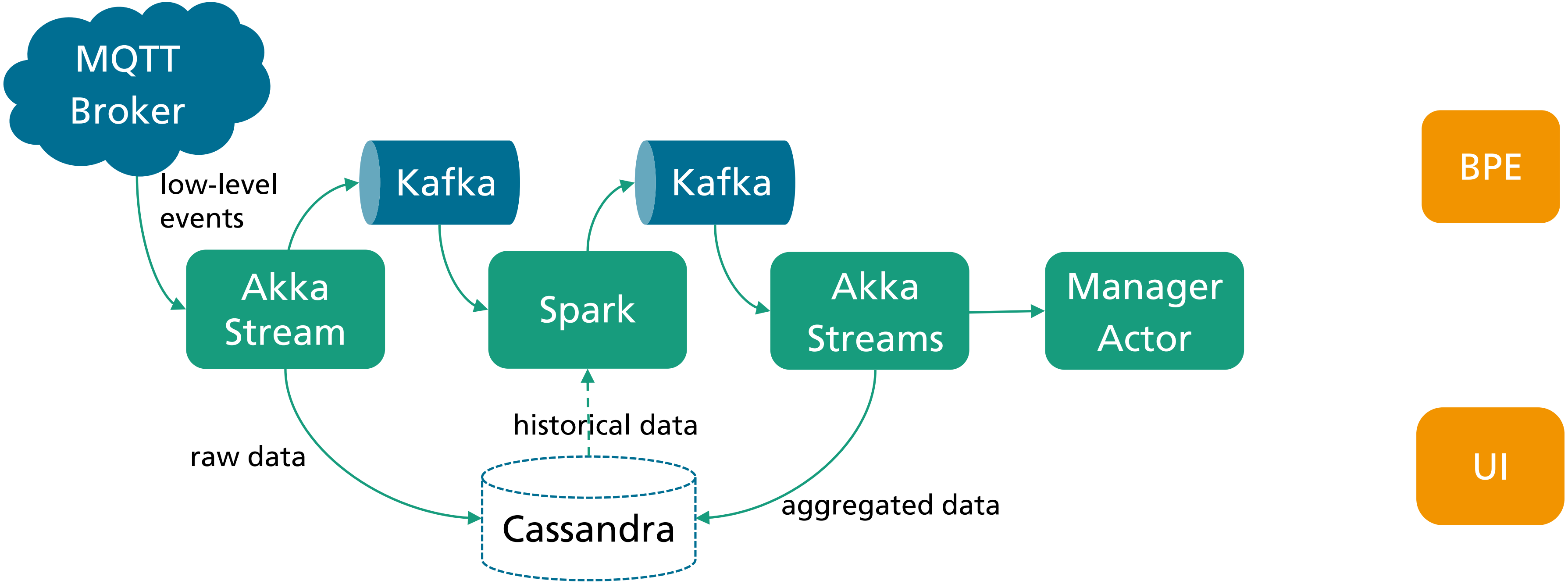
Create direct stream from Kafka

Parse strings and group by sensor id

Measured data is transformed by Fast Fourier Transformation and is compared with historical data

Send results to Kafka topic

Data Flow from IoT to Business



Send Processed Data to Manager Actor

Create reactive stream from Kafka topic

```
val kafkaStream = Consumer.atMostOnceSource(consumerSettings, Subscriptions.topics("SensorInformation"))
val source = kafkaStream.map(_.value)
```

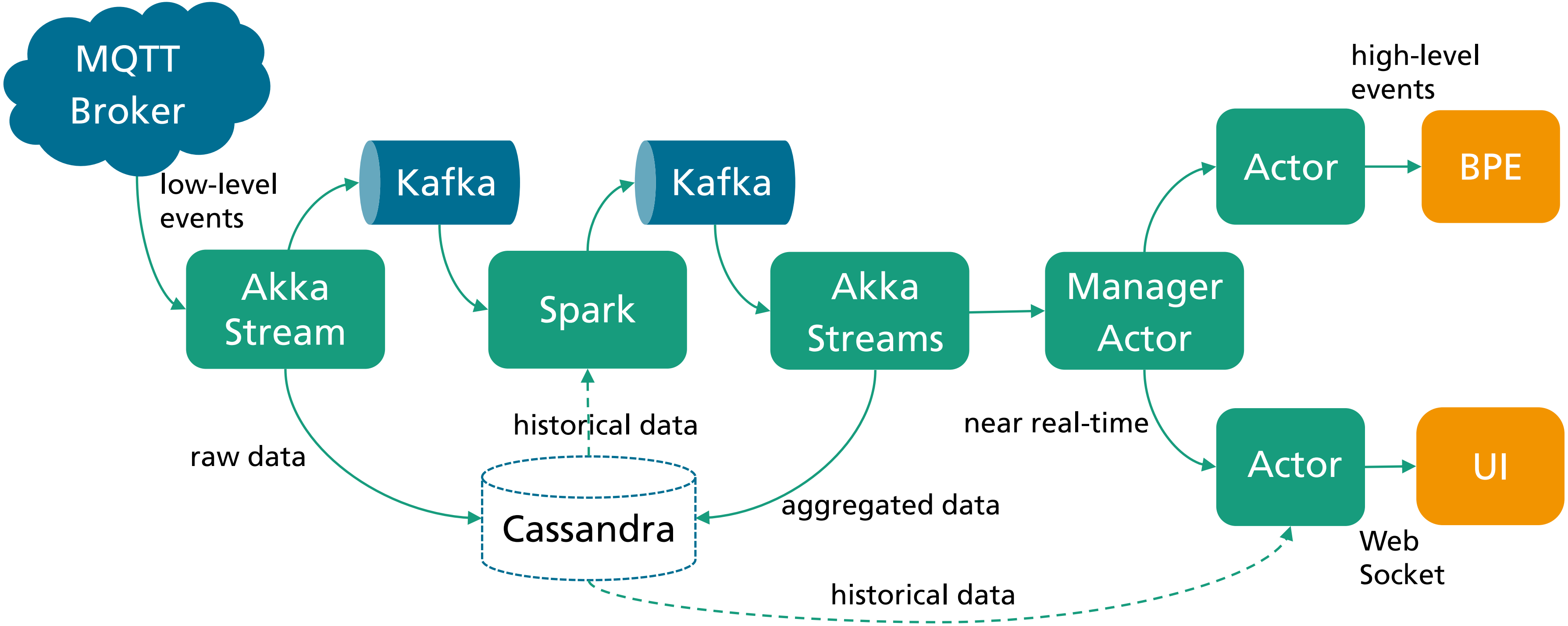
Send the machine state to the manager actor

```
source.runForeach(machineInformation => managerActor ! machineInformation)
```

```
val sink = CassandraSink[SensorInformation](parallelism = 2, preparedStatement, statementBinder)
val result = source.runWith(sink)
```

Store aggregated data to Cassandra

Data Flow from IoT to Business



Manager Actor

```
class ManagerActors extends Actor {

  private var routees = Set[Routee]()

  override def receive: Receive = {
    case add: AddRoutee => routees = routees + add.routee
    case remove: RemoveRoutee => routees = routees - remove.routee
    case msg: Any => routees.foreach(_.send(msg,sender))
  }
}
```

Add and remove routees

Forward messages to registered routees

Process Reference Actor

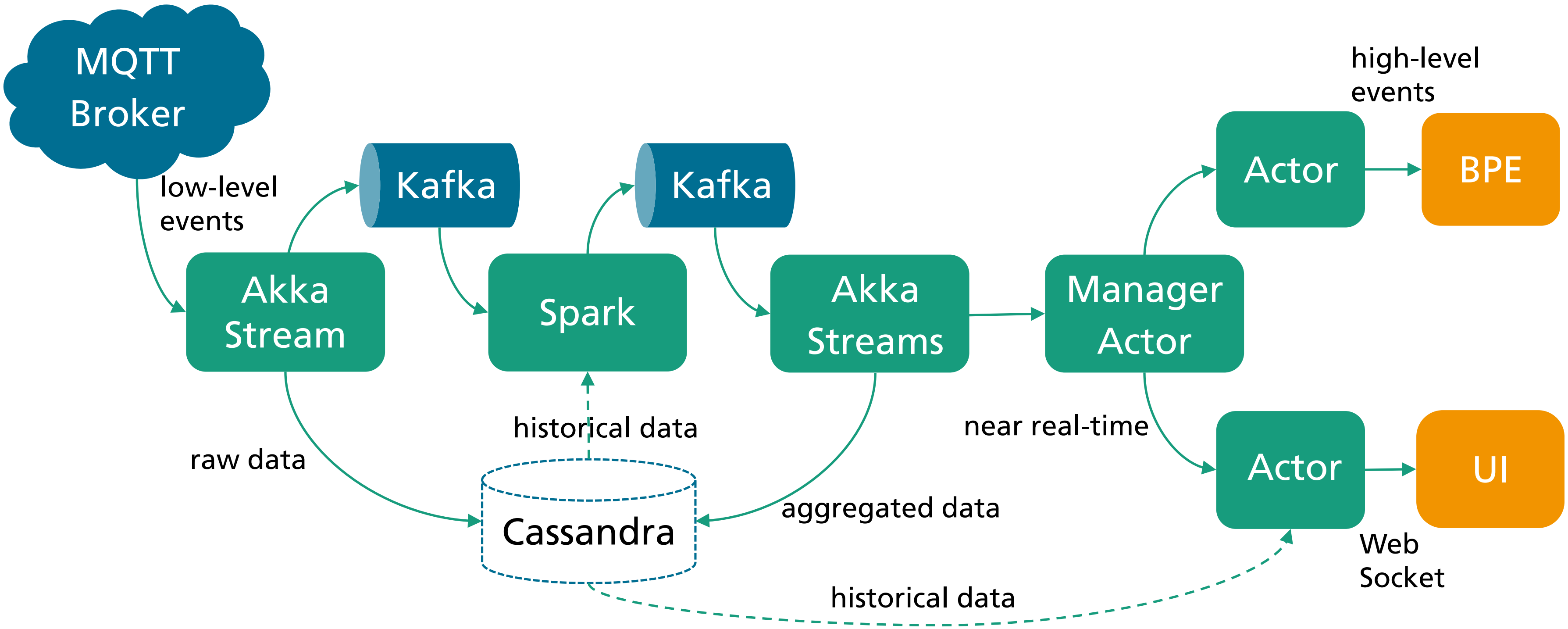
```
class ProcessActor(manager: ActorRef, process: ProcessReference) extends Actor {

  override def preStart() {
    manager ! AddRoutee(ActorRefRoutee(self))
  }

  override def postStop(): Unit = {
    manager ! RemoveRoutee(ActorRefRoutee(self))
  }

  override def receive: Receive = {
    case machineInfo: SensorInformation =>
      if(process.isRefernced(machineInfo)) {
        val msg = process.stateToMessage(machineInfo)
        process.notifyProcessInstance(msg)
      }
    case _ => None
  }
}
```

Register herself as a routee before actor is started

Remove this actor from the routees list

Notify process instance if conditions are satisfied

Data Flow from IoT to Business



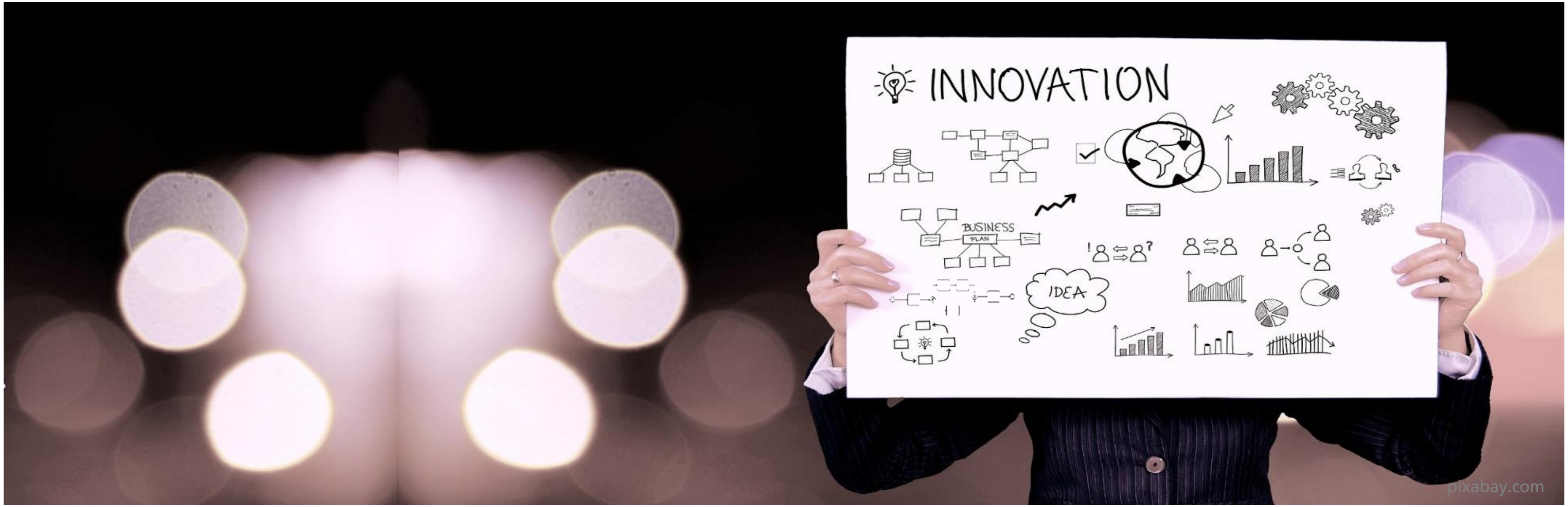
Fraunhofer ISST

Conclusion

- **Separation** of **IoT tier** and **business process tier**
  - Handle **vast amount of events** on the SMACK tier
  - Define business process for reaction on **high-level events**
- Kafka message broker between processing stages as **buffering layer**
- Performing **complex computation on scale** with Apache Spark
- Actors for **notifying** relevant **process instances**
- **Integration** of SMACK components
- **Benchmarking**

IOT ANALYTICS PLATFORM ON TOP OF SMACK

Yevgen Pikus | February 24th, 2017 | Berlin



pixabay.com

Fraunhofer
ISST

Tips and Tricks

- Do's
  - **Event sourcing** as Data Model
  - Tune **streaming batch size** and **processing time**
  - Balance between each worker process **one-to-many streams** and partitioning of single source
  - **Asynchronous boundaries** in Akka Streams

- Be careful
  - **Shared state** across cluster
  - **Shuffle data** across cluster
  - Processing time larger then **batch duration** in Spark
  - Kafka/spark **partitions** (parallel reads)
  - **Fault tolerance**