# Structured Streaming on Azure Databricks for Predictive Maintenance of Coordinate Measuring Machines

Jan-Philipp Simen
Data Scientist
Carl Zeiss AG

Spark + AI Summit Europe, London, 2018-10-03

#SAISEnt1

# Agenda

# ZEISS Camera Lenses

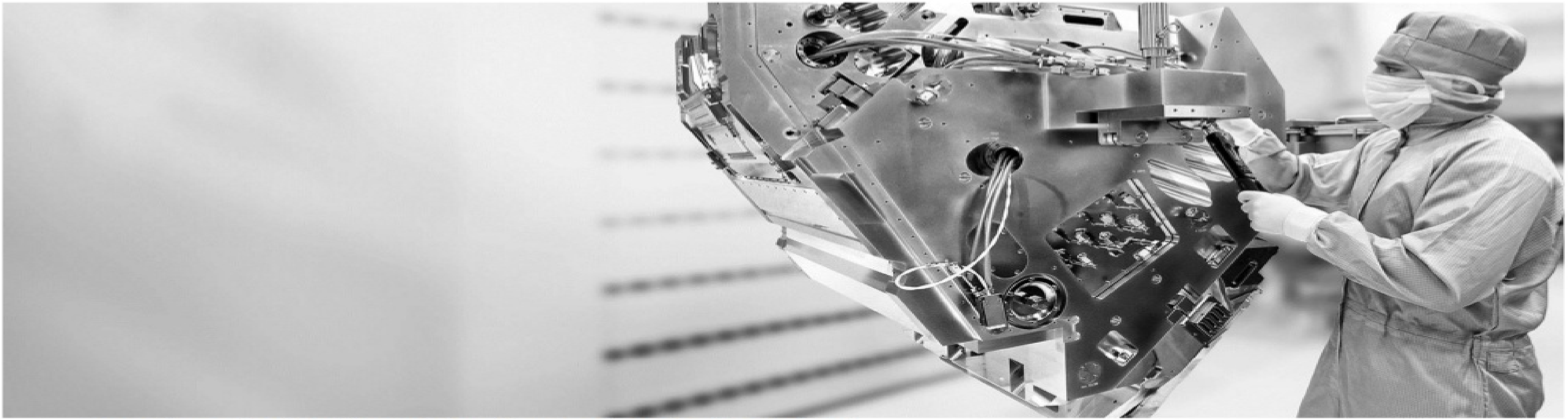
## Three Technical Oscars

# ZEISS Research & Quality Technology

## More Than 20 Nobel Prizes Enabled by ZEISS Microscopes

https://www.zeiss.com/microscopy/int/about-us/nobel-prize-winners.html

# ZEISS Semiconductor Manufacturing Technology With Lithography at 13.5 Nanometers Wavelength, ZEISS Is Enabling the Digital World

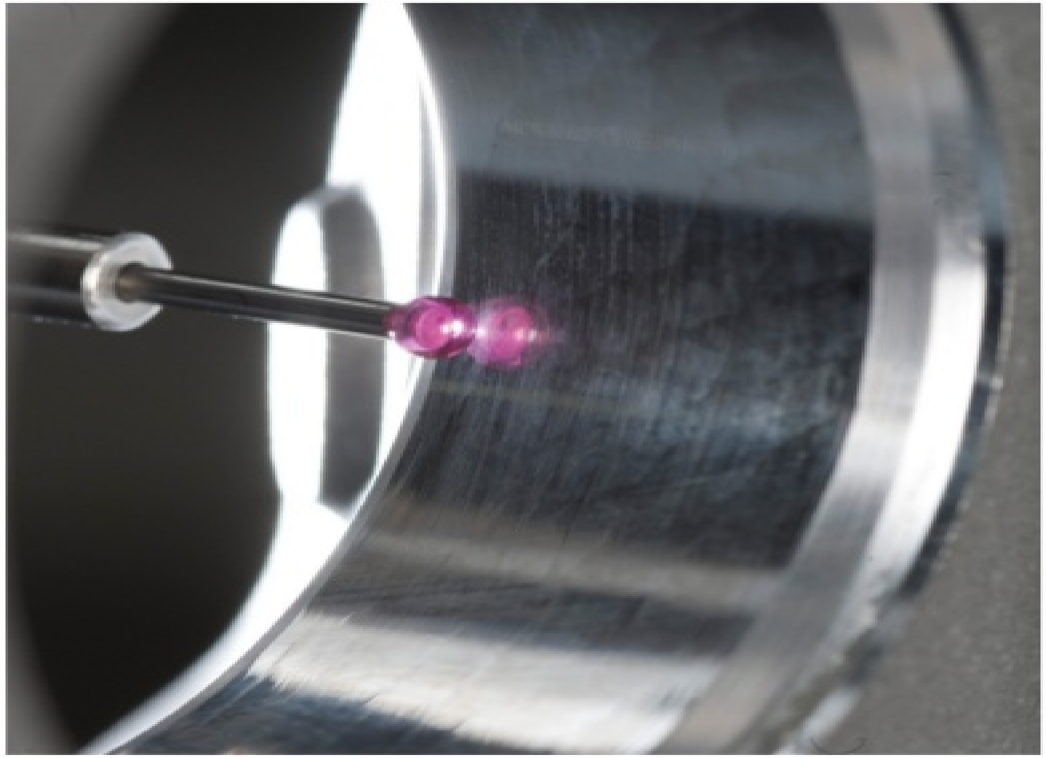


More on Extreme Ultra Violet Lithography: https://www.youtube.com/watch?v=Hfsp2jljDpI

## ZEISS Industrial Metrology

### Precision up to 0.3 µm

ZEISS



Quality Assurance

Metrology

Coordinate Measuring Machines

Usually comparing CAD model to produced workpiece

- tactile
- optical
- x-ray
- laser

# Agenda

1 About ZEISS

2 **Motivation for Predictive Maintenance**

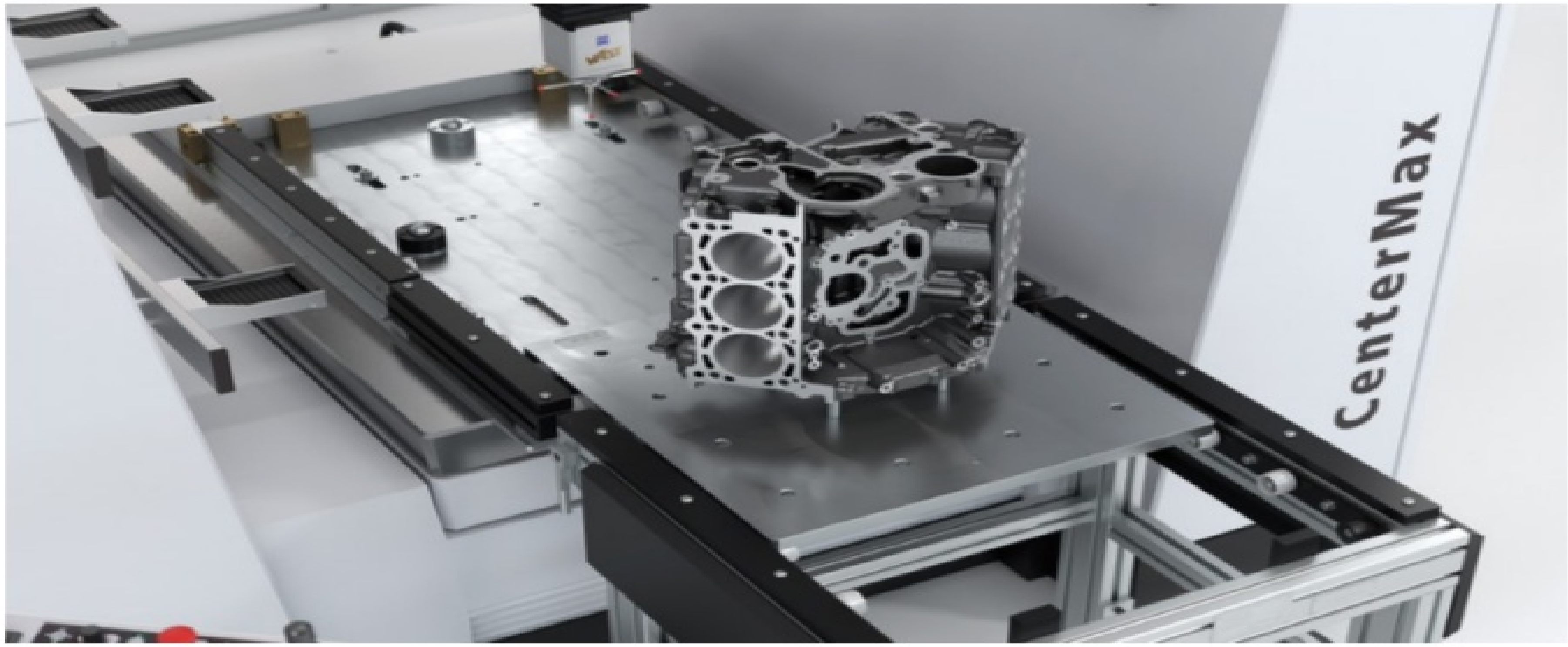
3 Processing Live Data Streams

4 Processing Historical Data Batches

5 Bringing It Together

6 Summary

# How Will Predictive Maintenance Benefit Our Customers?




**Main goals:**

1. avoid downtime
2. ensure reliable measurements

# Delivering Customer Value from the Start

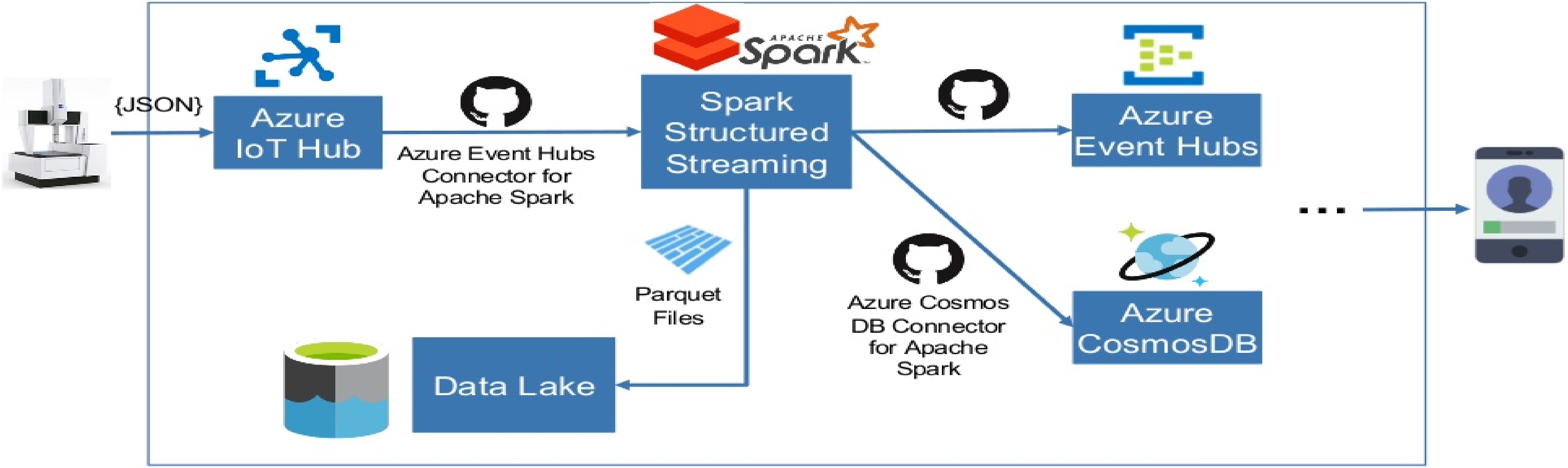

Start small, learn from data and develop first value-adding services

1. Condition monitoring app ✔
2. One-click support request ✔
3. Predictive models for internal use at ZEISS ✔
4. Predictive maintenance notifications in app

# Agenda

1. About ZEISS
2. Motivation for Predictive Maintenance
3. **Processing Live Data Streams**
4. Processing Historical Data Batches
5. Bringing It Together
6. Summary

# Streaming Architecture on Azure



# In a Streaming Pipeline the Drawbacks of Untyped SQL Operations Are Even More Apparent

What is in there, again?

Don't forget to document the schema!

Did I spell this right?

```
def readSensor9Untyped(streamingDF: DataFrame): DataFrame =
  streamingDF.select(
    $"machineID",
    $"dateTime",
    udfParseDouble("sensor9")($"jsonString"))

def udfParseDouble(key: String) = ???
```

Our solution: Spark Datasets and Scala Case Classes

# Spark Datasets + Scala Case Classes = Typesafe Data Processing

```scala
case class Event(machineID: String,
                 dateTime: Timestamp
                 jsonString: String)
```

```scala
case class Record[T](machineID: String,
                     timestamp: Timestamp,
                     value: T)
```

```scala
def jsonToRecord(sensorID: Int)(event: Event): Record[Option[Double]] =
  Record[Option[Double]](event.machineID,
                         event.dateTime,
                         parseSensorValue(sensorID, event.jsonString))

def parseSensorValue = ???
```

Leveraging the full potential of a good IDE:

- Auto completion
- Compile checks
- Quick fixes
- Refactoring

```scala
def readSensor9(streamingDS: Dataset[Event]): Dataset[Record[Option[Double]]] =
  streamingDS.map(jsonToRecord(9))
```

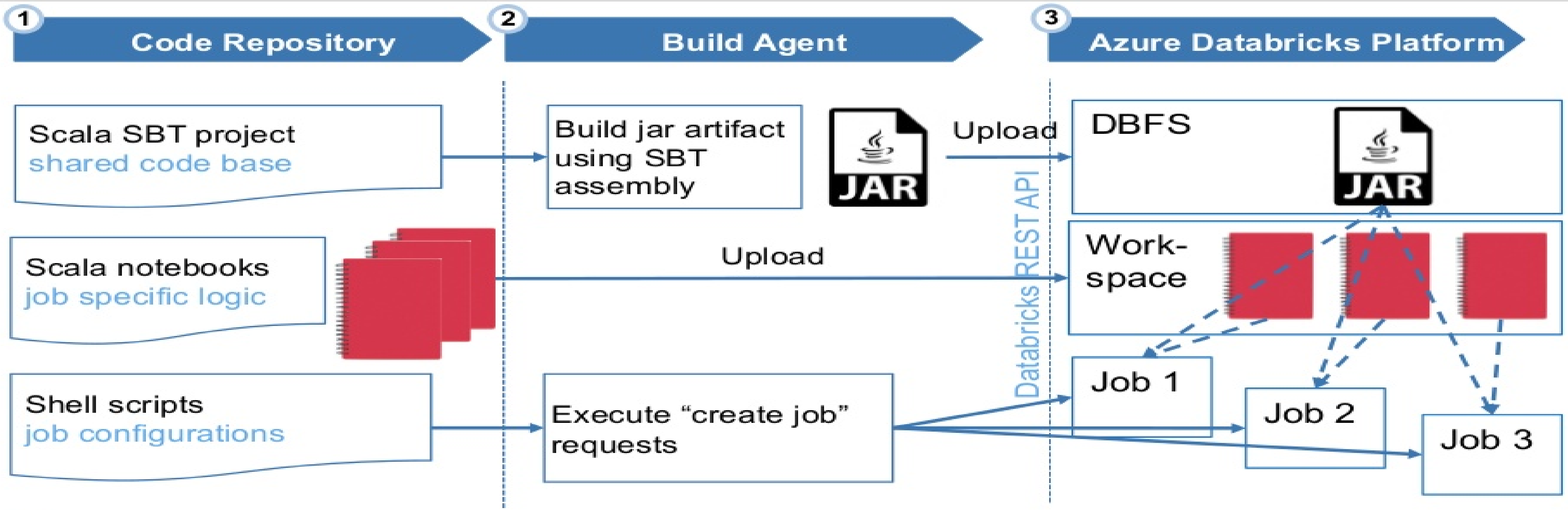
**Spark features wish list**

- Typesafe joins for Datasets
- Import / export case classes from / to JSON Schema would allow centralized schema repository

## Implementing Alert Logic with Stateful Streaming

```scala
/** Checks a stream of JSON messages for threshold violations.
  *
  * @param jsonStream Dataset containing JSON messages as received from IoT Hub
  * @param cmms Dataset with thresholds and other specifications for each machine
  * @return Dataset with alert events as expected by the Eventhub
  */
def createAlerts(jsonStream: Dataset[JsonMessage],
                 cmms: Dataset[CmmSpecs]): Dataset[AlertEvent] = {

  val joined: Dataset[JsonPlusCmmSpecs] = joinStaticOnStream(jsonStream, cmms)

  // Group by serial number
  val grouped = joined.groupByKey(_.machineID)

  // Evaluate incoming messages in each group (that is for each individual machine)
  grouped.flatMapGroupsWithState[AlertState, AlertEvent](...)(evaluateTemperatureState)
}

def evaluateTemperatureState = ???
```

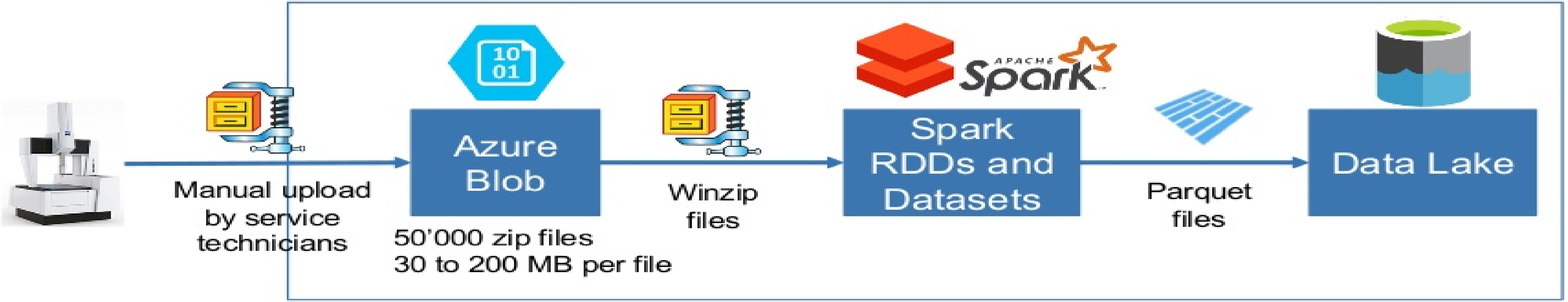
# Automated Build and Deployment



## Agenda

1. About ZEISS
2. Motivation for Predictive Maintenance
3. Processing Live Data Streams
4. **Processing Historical Data Batches**
5. Bringing It Together
6. Summary

# Batch ETL Architecture



# Unzipping Lessons Learned

**In Theory**

```
val allFilesOnBlob: RDD[(String, PortableDataStream)] = sc.binaryFiles(path)
def unzip(zipFile: PortableDataStream): Map[String, String] = easyUnzipFunction(zipFile)
val unzippedFiles: RDD[(String, Map[String, String])] = allFilesOnBlob.mapValues(unzip)
```

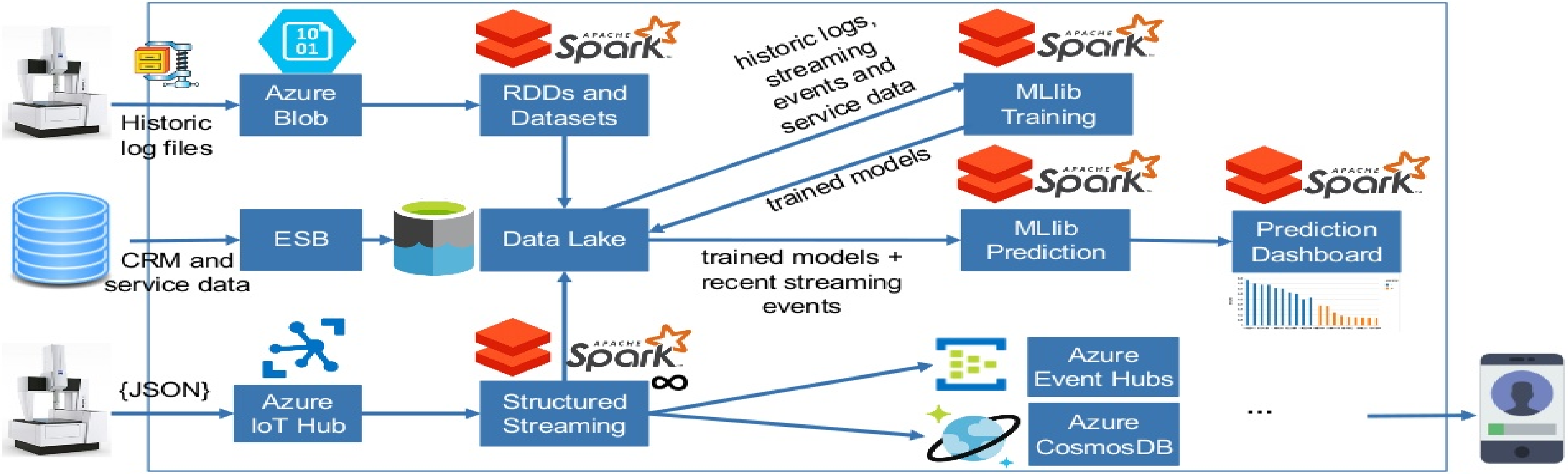
**In Practice**

- 250 lines of code only for the unzipping (Winzip format)!
- A lot of exception handling during unzipping.
- Many debugging iterations.
- Size of zip files ranging from 30 MB to 200 MB (compressed) ➡️ Had to reserve a lot of memory overhead for big zip files ➡️ Split RDD into three buckets, depending on file size

**Next time:**
**Use gzip compression on file level instead of putting varying amounts of files into a Winzip archive.**

# Agenda

## Combining Batch and Streaming Data Sources and Adding Scheduled Machine Learning Notebooks



## Agenda

1. About ZEISS
2. Motivation for Predictive Maintenance
3. Processing Live Data Streams
4. Processing Historical Data Batches
5. Bringing It Together
6. **Summary**

# Things We Like

***Things we like about Spark***

- Datasets: Typesafe data processing
- Stateful Streaming: Powerful transformations of streaming datasets
- MLlib: State-of-the-art distributed algorithms
- All the other great things: Scalability, performance, ...

***Things we like about Azure Databricks***

- Cluster management: Convenient setup, auto-scaling, auto-shutdown
- Job management: Convenient creation and scheduling of Spark jobs
- REST API: Enables automated deployment

# Wish List

***Features we would love to see in the future***

- ❑ Datasets: Typesafe joins
- ❑ Central schema repository with export to different formats (Scala case class, JSON schema)
- ❑ Better monitoring for multiple, long-running jobs: Integration with external log aggregators or customizable log metrics in Databricks

# Contact

Dr. Jan-Philipp Simen

Data Scientist
Digital Innovation Partners

Carl Zeiss AG
Kistlerhofstraße 70
81379 Munich

jan-philipp.simen@zeiss.com

https://www.linkedin.com/in/jpsimen

ZEISS