Quickly Adapting to Changes
Christian Adriano, Felix Boelter, Lasse Kohlmeyer, Holger Karl, Holger Giese,
System Analysis & Modeling / Network Softwarization

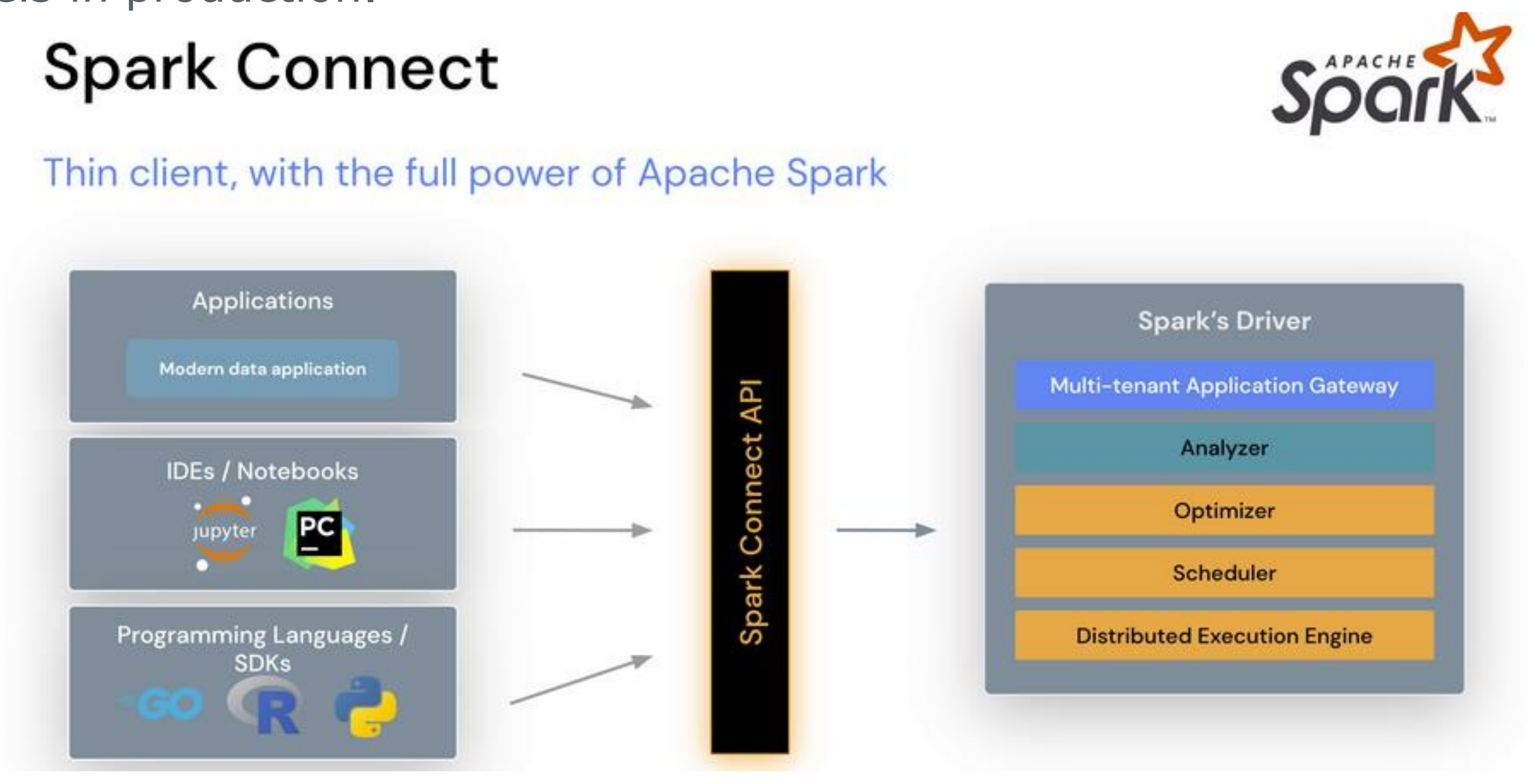
Project partners

Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Quickly Adapting to Changes

Basic Concepts

Apache Spark™ [1] is a multi-language unified engine for executing large scale data engineering, data science, and machine learning on single-node machines or clusters. *Apache Spark* became the backbone of **80% of Fortune 500 companies** running A/B Tests and deploying machine learning models in production.


Spark Connect
Thin client, with the full power of Apache Spark
Applications
Modern data application
IDEs / Notebooks
Programming Languages / SDKs
Spark Connect API
Spark's Driver
Multi-tenant Application Gateway
Analyzer
Optimizer
Scheduler
Distributed Execution Engine

To integrate with *Spark*, one relies on **Spark Connect** [2] = an open-source communication protocol that leverages the concept of unresolved logical plans to communicate with *Spark*. This interface correspond to a very large fraction of the user surface with *Spark*.

To facilitate the use of *Spark Connect*, thousands of customers leverage **Databricks Connect** [3] = to connect easily to a remote *Databricks* cluster and execute their Spark code written locally in an IDE of their choice. This flexibility makes it possible to integrate with the *Databricks* platform in any development environment or other applications.

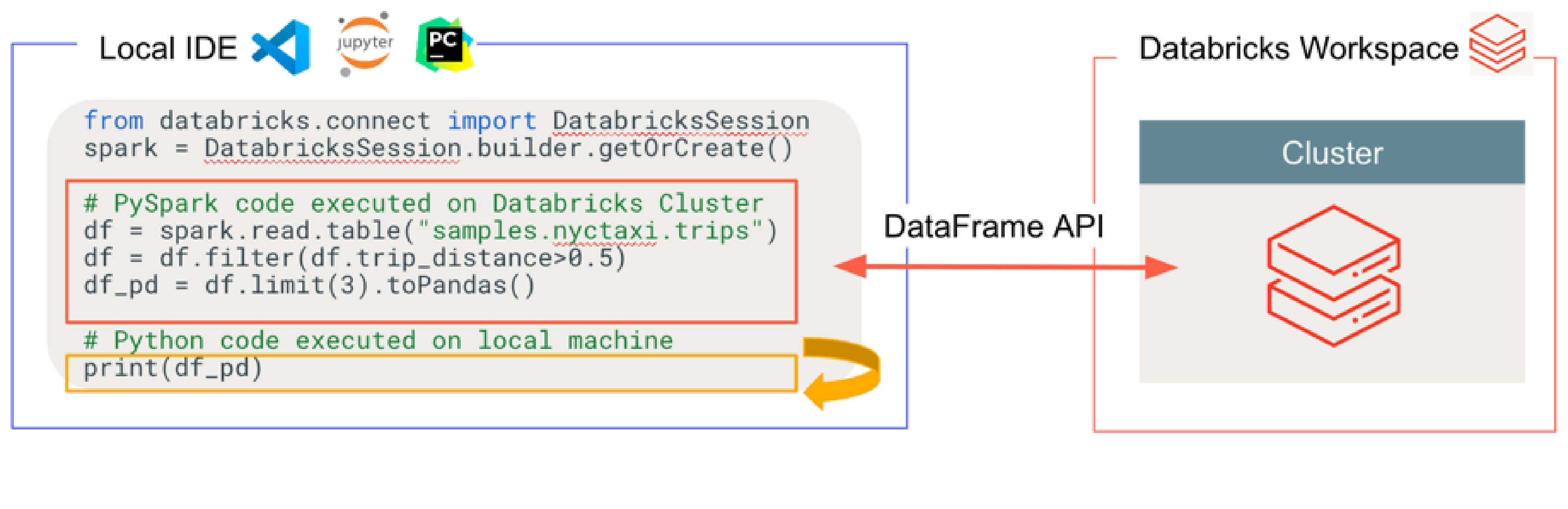
Detail: How does Databricks Connect look like?

Databricks Connect is a client library for the Databricks Runtime. It allows you to write code using *Spark* APIs and run them remotely on a *Databricks* cluster instead of in the local Spark session.

For example, when you run the DataFrame command `spark.read.format(...).load(...).groupBy(...).agg(...).show()` using *Databricks Connect*, the logical representation of the command is sent to the *Spark* server running in *Databricks* for execution on the remote cluster.

Databricks Connect determines where your code runs and debugs, as shown in the following figure.

For running code: All code runs locally, while all code involving DataFrame operations runs on the cluster in the remote Databricks workspace and run responses are sent back to the local caller.
For debugging code: All code is debugged locally, while all *Spark* code continues to run on the cluster in the remote Databricks workspace. The core Spark engine code cannot be debugged directly from the client.



```
from databricks.connect import DatabricksSession
spark = DatabricksSession.builder.getOrCreate()

# PySpark code executed on Databricks Cluster
df = spark.read.table("samples.nyctaxi.trips")
df = df.filter(df.trip_distance>0.5)
df_pd = df.limit(3).toPandas()

# Python code executed on local machine
print(df_pd)
```

[1] What is Spark? - https://spark.apache.org/docs/latest/quick-start.html
[2] Spark Connect - https://spark.apache.org/docs/latest/spark-connect-overview.html
[3] Databricks Connect? - https://docs.databricks.com/en/dev-tools/databricks-connect/index.html

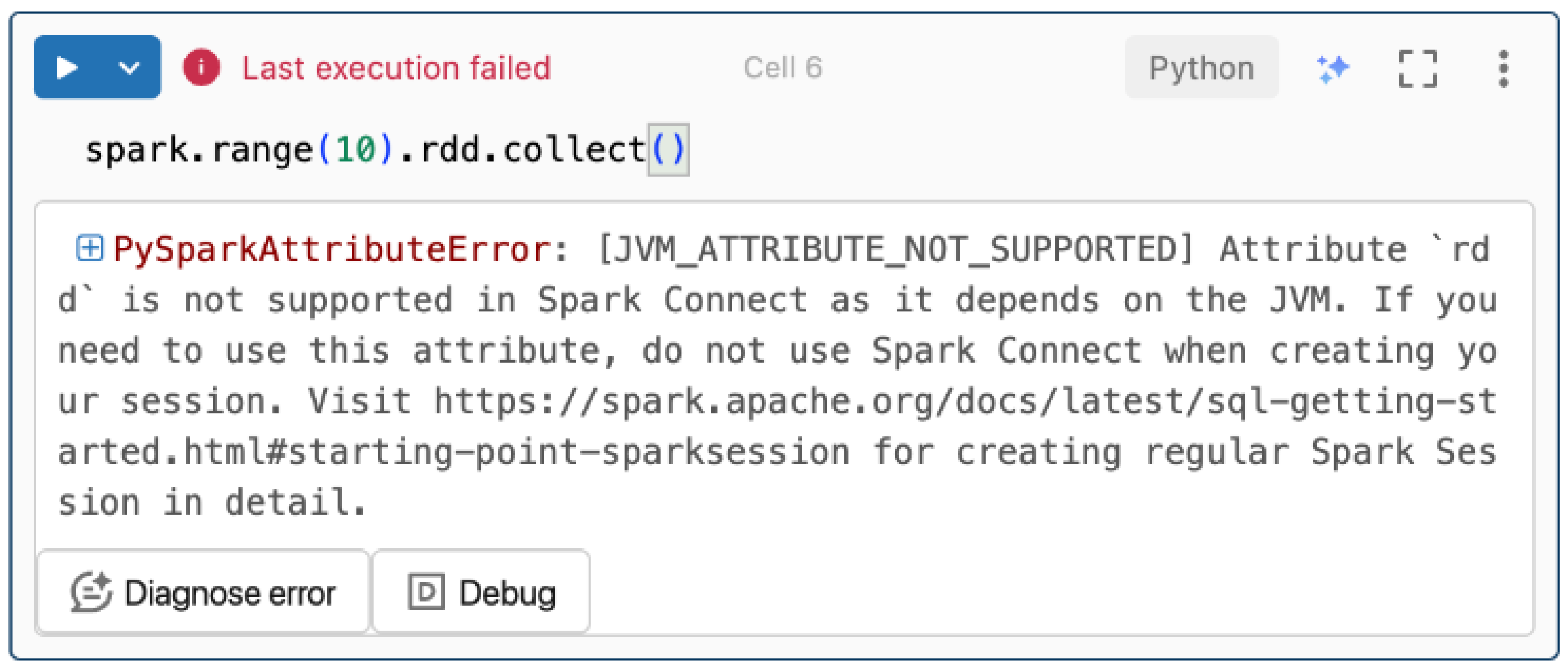
Context

In the adoption journey of *Spark* Connect within the context of *Databricks* and open-source, customers frequently try out code that they found from internet sources like *StackOverflow* or generated by LLMs.

However, for most of the existing code, there are better ways to rewrite it using appropriate primitives that are part of *Apache Spark* and the available APIs. Our general **goal** is to help engineers find these best ways.

Detail – Example of a Failure

With the adoption of Spark Connect [2], we knew that there were certain areas of the application surface that were no longer accessible because they either required direct access to the driver JVM using Py4J or required arbitrary code serialization (called RDD).



```
spark.range(10).rdd.collect()
```

```
PySparkAttributeError: [JVM_ATTRIBUTE_NOT_SUPPORTED] Attribute `rd
d` is not supported in Spark Connect as it depends on the JVM. If you
need to use this attribute, do not use Spark Connect when creating yo
ur session. Visit https://spark.apache.org/docs/latest/sql-getting-st
arted.html#starting-point-sparksession for creating regular Spark Ses
sion in detail.
```

Project Goals & Tasks

We propose to explore the following avenues to guide software engineers to adapt their system to make *Spark Connect* compatible and thus future-proof.

1. **Map of Pain-Points (requirements)** – List the types of API changes that are more critical to the functioning of the customer code. For each type of change, estimate the complexity of the adaptation of the customer code.
2. **Draft a Roadmap** – Prioritize pain points, their interdependencies, effort and risks.
3. **Prototype & Proof-of-Concept** – Select a pain-point to ideate, specify, and implement. Define the evaluation criteria. Deploy and test the prototype on a realistic setting.
4. **Make an Open-Source Contribution!** – Depending on the outcome, submit a pull request to the corresponding open-source project.
5. **Write the Paper!** - Submit the project report to a conference.

Toolbox - we plan to explore existing tools for code generation, summarization, and refactoring - both rule-based (static analysis) and machine learning-based (LLM).

Detail: Tasks

The team will decide together with *Databricks* the following:

- scope/timing of an introductory tutorial on *Spark* and *Databricks* technology stack, eventually involving a visit to Databricks headquarters.
- interview software engineers to understand the pain-points involved in adapting customer code to new API.
- specify the criteria to prioritize the pain-points.
- design the proof-of-concept (scenarios, tasks, tests).
- ideate solutions, prototype and evaluate them via a proof-of-concept.

For more information contact

Christian Adriano (christian.adriano@hpi.de or Room A-2.7)
Felix Boelter (felix.boelter@hpi.de)