The Best of Both Worlds

Unlocking the Power of Apache Beam with Apache Flink

Maximilian Michels
maximilianmichels.com
@stadtlegende

Agenda

1. What is Apache Beam?
2. Why use Beam + Flink?
3. Classic Flink Runner
4. Portable Flink Runner
5. Getting Started

About

- Software Engineer and Consultant
  - PMC / Committer at Beam / Flink
  - Ververica (dataArtisans) on Flink / Beam
  - Google and Lyft on Beam / Flink
- Focus: Beam Portability / Flink Runner

Maximilian Michels
maximilianmichels.com
@stadtlegende

What Is Apache Beam?

🤔

What is Beam?

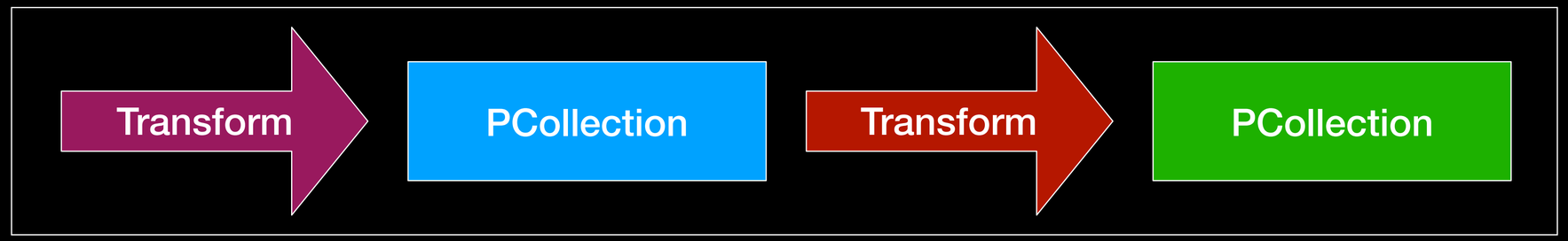
- Apache open-source project since 2016
- Parallel/distributed data analytics

1. Unified API for batch and streaming
2. Programming language of your choice
3. Execution engine of your choice

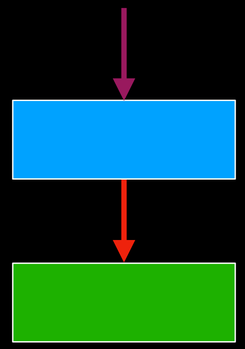
1. Unified API

- Apache Flink: DataStream, DataSet, Table
- Apache Spark: RDD, DStream, Dataset, Dataframe
- Apache Beam: Pipeline*

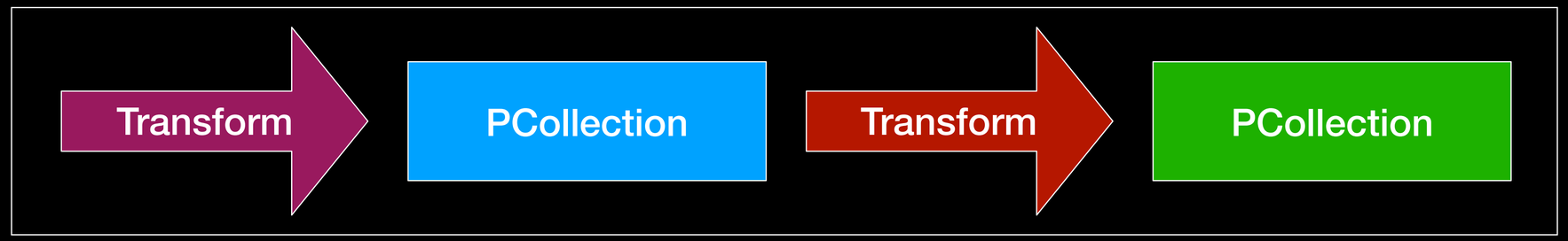
Pipeline



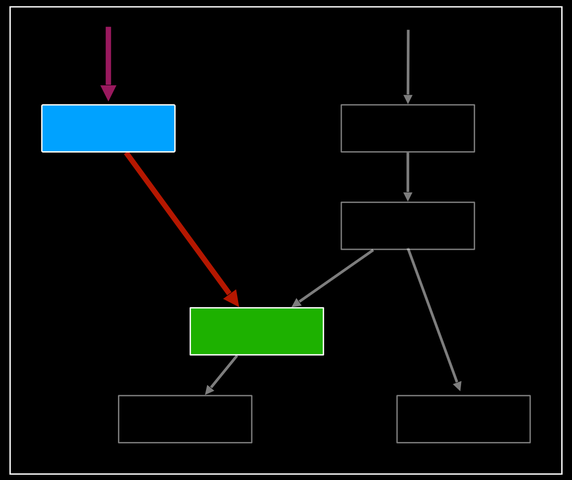
Pipeline



Pipeline

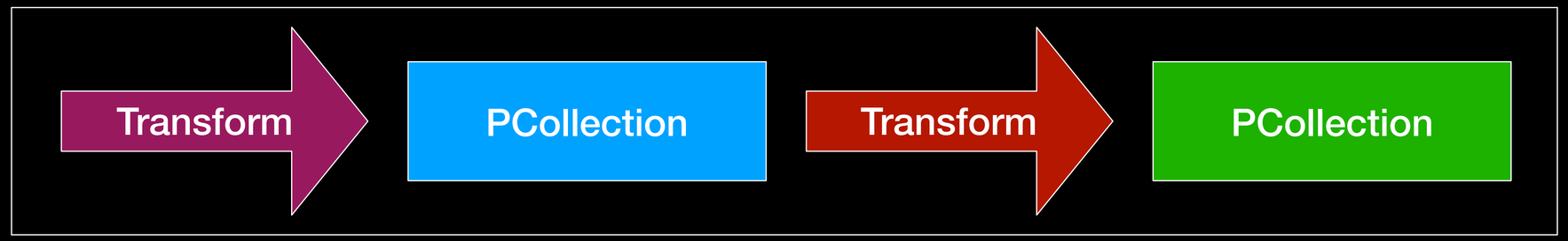


Pipeline



Also a Pipeline

The Beam API



Pipeline

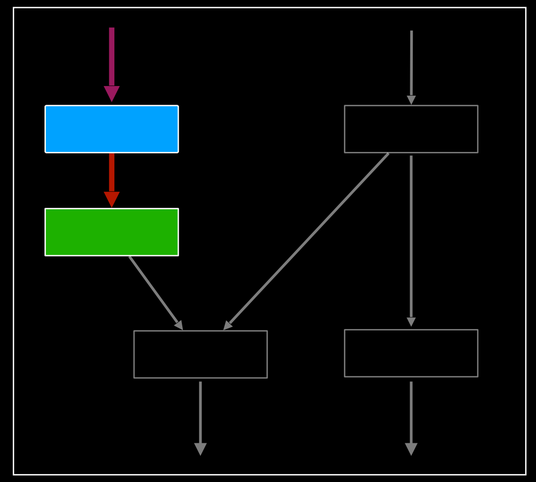
```
Pipeline p = Pipeline.create()

PCollection pCol1 = p.apply(transform)

PCollection pcol2 = pCol1.apply(transform)

PCollection pcol3 = p.apply(...).apply(...)

p.run()
```



Transforms

- **Primitive** or **Composite**
- Composite transforms expand to primitive
  - For example:
    Combine = ParDo + GroupByKey + ParDo

PRIMITIVE TRANSFORMS
Read
ParDo
GroupByKey
AssignWindows
Flatten

WordCount

ParDo

```
input -> output

str   -> KV<str,int>

"to" -> KV<"to", 1>
"be" -> KV<"be", 1>
"or" -> KV<"or", 1>
"not"-> KV<"not",1>
"to" -> KV<"to", 1>
"be" -> KV<"be", 1>
```

GroupByKey

```
KV<k,v>     -> KV<k, [v1,v2,...]>

KV<str,int> -> KV<str,[int…]>

KV<"to", [1,1]>
KV<"be", [1,1]>
KV<"or", [1  ]>
KV<"not",[1  ]>
```

ParDo

```
input -> output

KV<str, [int…]> -> KV<str,int>

KV<"to", 2>
KV<"be", 2>
KV<"or", 1>
KV<"not",1>
```

"Map Phase"

"Shuffle Phase"

"Reduce Phase"

Wordcount - Raw version

```
pipeline
    .apply(Create.of("to", "be", "or", "not", "to", "be"))
    .apply(ParDo.of(
        new DoFn<String, KV<String, Integer>>() {
          @ProcessElement
          public void processElement(ProcessContext ctx) {
            KV<String, Integer> outputElement = KV.of(ctx.element(), 1);
            ctx.output(outputElement);
          }
        }))
    .apply(GroupByKey.create())
    .apply(ParDo.of(
        new DoFn<KV<String, Iterable<Integer>>, KV<String, Long>>() {
          @ProcessElement
          public void processElement(ProcessContext ctx) {
            long count = 0;
            for (Integer wordCount : ctx.element().getValue()) {
              count += wordCount;
            }
            KV<String, Long> outputElement = KV.of(ctx.element().getKey(), count);
            ctx.output(outputElement);
          }
        }))
```

Wordcount – Composite Transforms

```
pipeline
  .apply(Create.of("to", "be", "or", "not", "to", "be"))
  .apply(MapElements.via(
      new SimpleFunction<String, KV<String, Integer>>() {
          @Override
          public KV<String, Integer> apply(String input) {
            return KV.of(input, 1);
          }
   }))
  .apply(Sum.integersPerKey());
```

Composite Transforms

Wordcount - More Composite Transforms

```
pipeline
    .apply(Create.of("to", "be", "or", "not", "to", "be"))
    .apply(Count.perElement());
```

Composite Transforms

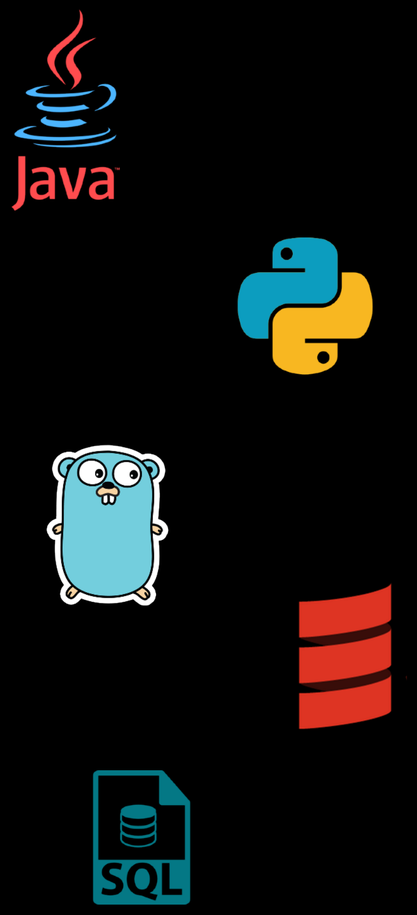
Python to the Rescue

```
pipeline
    | beam.Create(['to', 'be', 'or', 'not', 'to', 'be'])
    | beam.Map(lambda word: (word, 1))
    | beam.GroupByKey()
    | beam.Map(lambda kv: (kv[0], sum(kv[1])))
```

Python to the Rescue

```
pipeline
    | beam.Create(['to', 'be', 'or', 'not', 'to', 'be'])
    | beam.Map(lambda word: (word, 1))
    | beam.CombinePerKey(sum)
```

2. Language

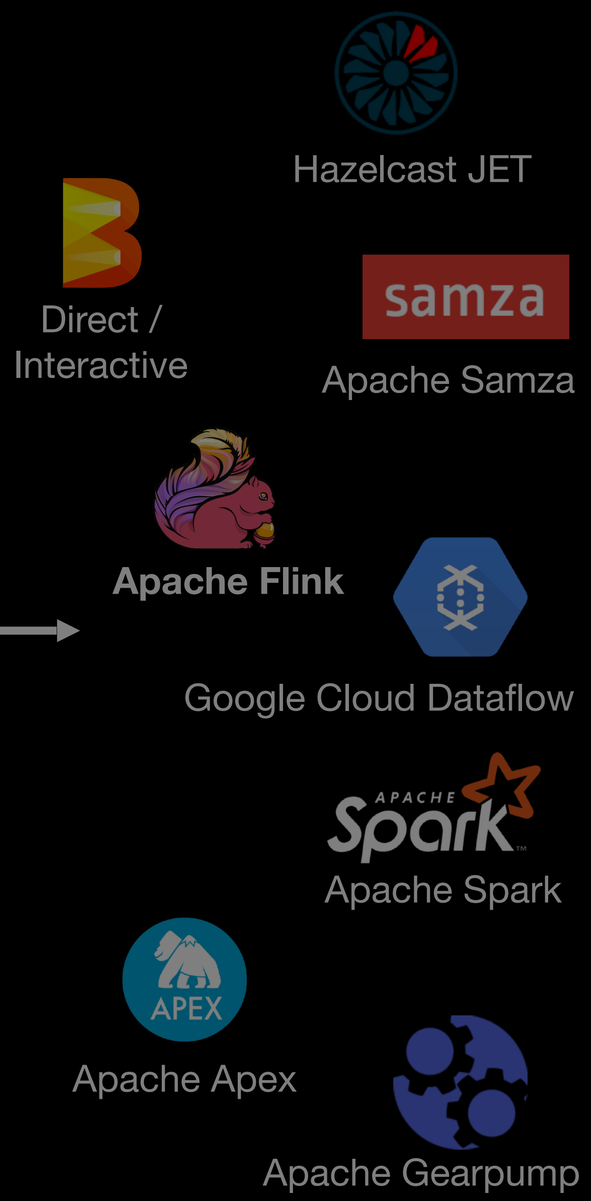


SDKs

Write

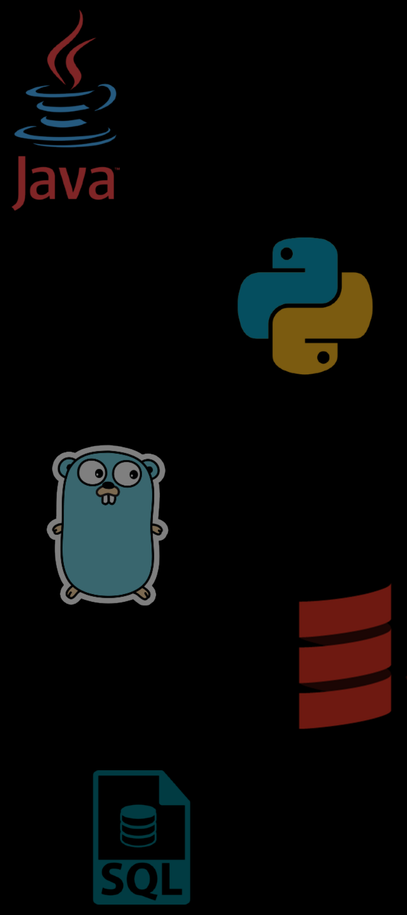
Pipeline
+
Runners

Translate



Execution Engines

3. Execution Engine



SDKs

Write

Pipeline
+
Runners

Translate

Direct / Interactive

Hazelcast JET

samza

Apache Samza

Apache Flink

Google Cloud Dataflow

APACHE Spark

Apache Spark

Apache Apex

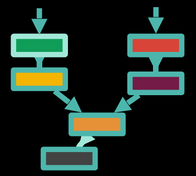
Apache Gearpump

Execution Engines

Apache Beam

SDKs

Write



Pipeline

+

Runners

Translate

Direct / Interactive

Hazelcast JET

samza

Apache Samza

Apache Flink

Google Cloud Dataflow

APACHE Spark

Apache Spark

Apache Apex

Apache Gearpump

Execution Engines

Why Use Beam + Flink?

What is Flink?

- "Stateful Computations over Data Streams"
- A batch and stream processing engine
- Managed memory, exactly-once, checkpointing/savepoints
- Many of the same semantics as Beam

Common: Beam vs Flink

- Distributed dataflow model
- Batch and stream processing
- Event time / processing Time
  - Watermarks
  - Timers / State
  - Windows / Triggers
    (we don't use the Flink primitives in Beam)

Differences: Beam vs Flink

	Beam	Flink
Type	Execution abstraction	Execution engine
API	Unified Batch/Stream API	Separate Batch/Stream API, Unified Table API
Languages	Java/Scala, Python, Go, SQL	Java/Scala, (Python), SQL
Control	Auto-tuning	Very flexible (parallelism, partitioning/keying, resource usage)
Beam	Event Time (default) / Processing Time	Event Time / Processing Time (default)
Window Assignment	Eagerly	Lazily
Key Grouping	Explicit (via GroupByKey)	Implicit (window materialization or aggregations)
Side Input	Yes (including windowing)	Yes (low-level)
Type System	Type system with inference	Type system with inference and versioned serializers

Why use Beam with Flink?

- Unified API
  (batch and streaming)

- Native Python support
  (+Java +Go +SQL +Scala)

- Power of Flink
  (exactly-once, robustness, memory management, etc.)

- Fully Flink-compatible
  (but runs also on Google Cloud Dataflow, Spark, etc.)

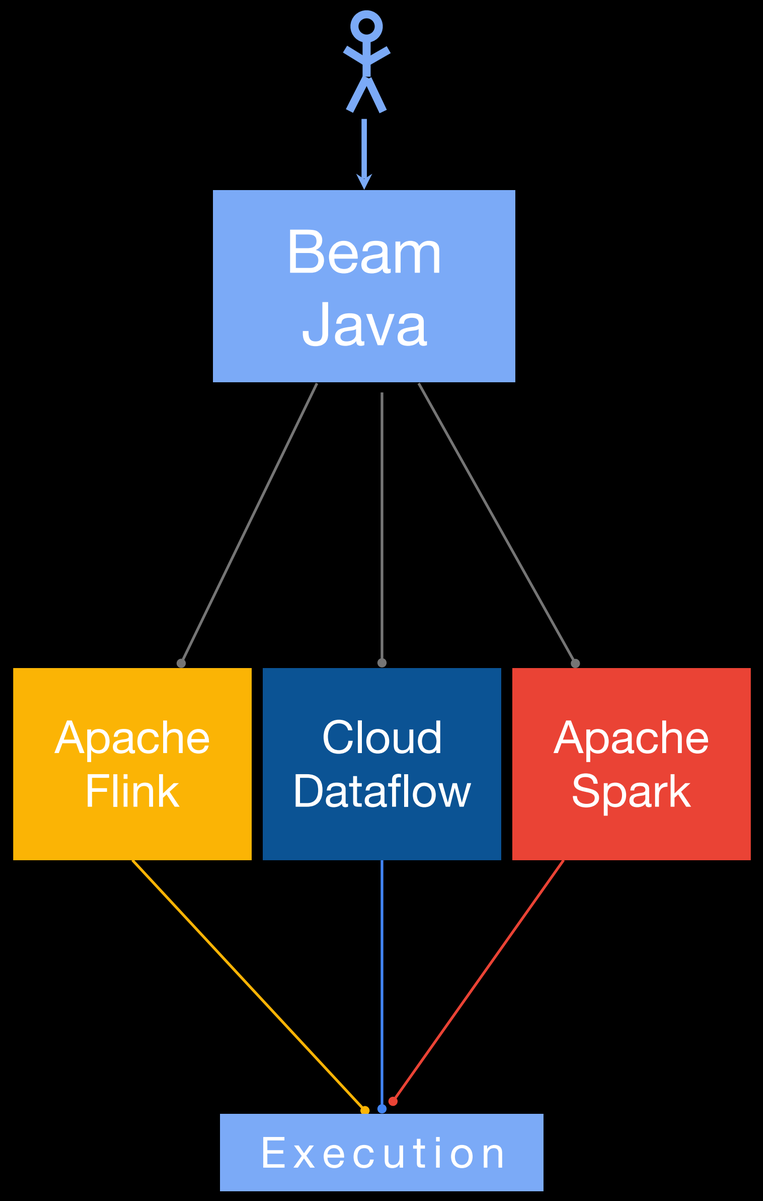
- More features
  (side inputs, cross-language pipelines)

The Flink Runner

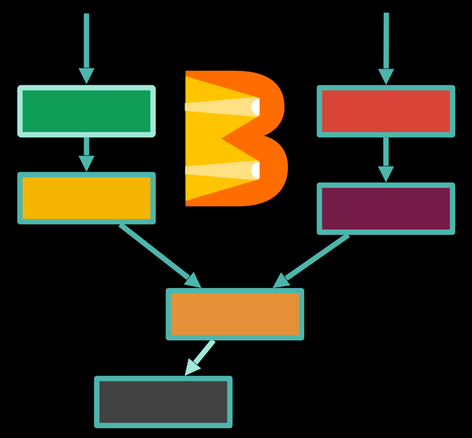
Classic Runner

(Java/Scala, SQL)

Classic Model



Translating Beam to Flink



Translators

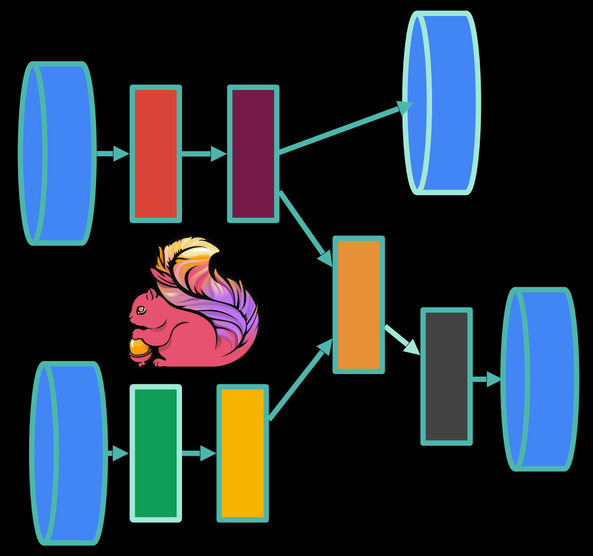
Read

ParDo

GroupByKey

AssignWindows

Flatten

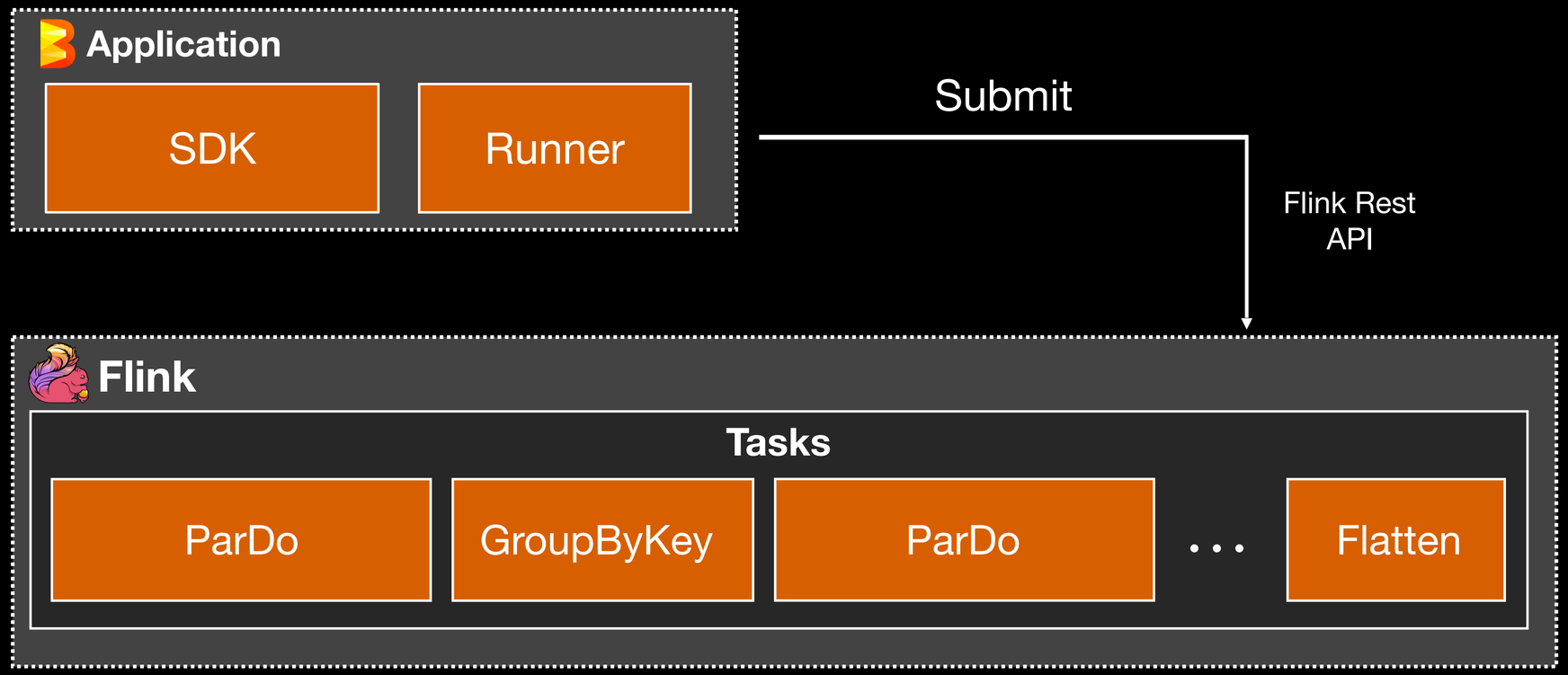


Pipeline

Pipeline Translators

Flink Job

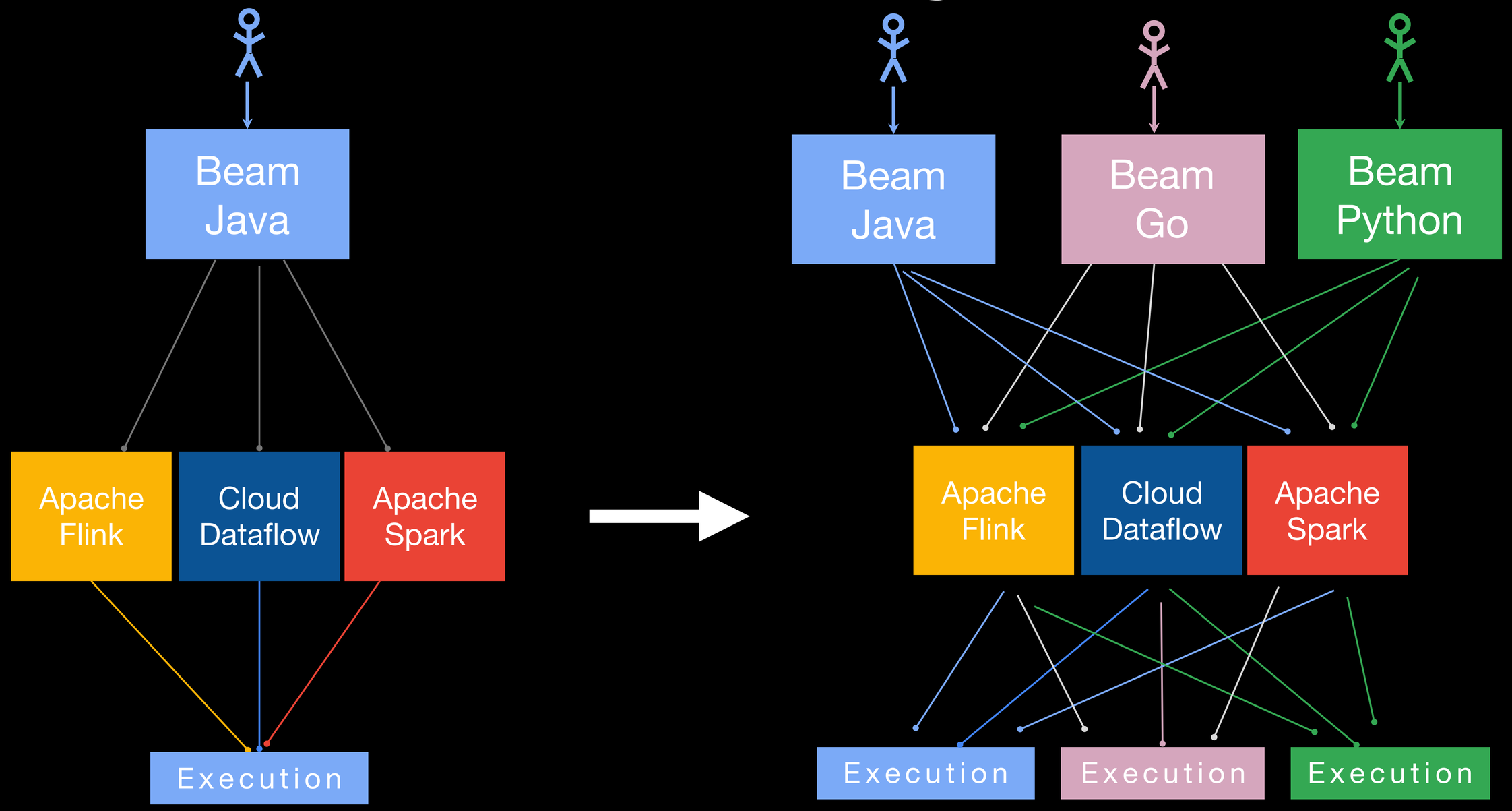
"Classic" Architecture



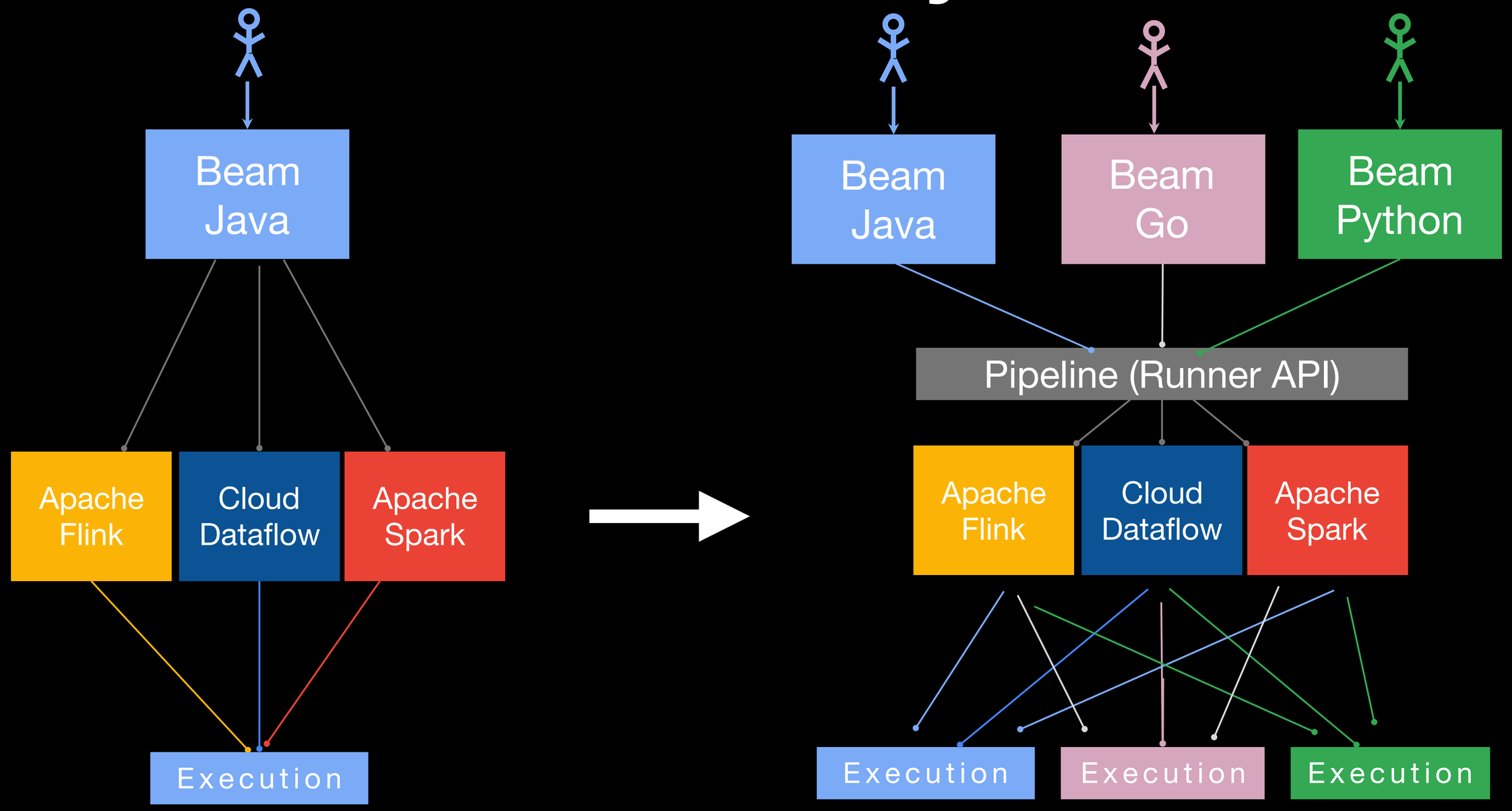
Portable Runner

(Python, Go, SQL, Java/Scala)

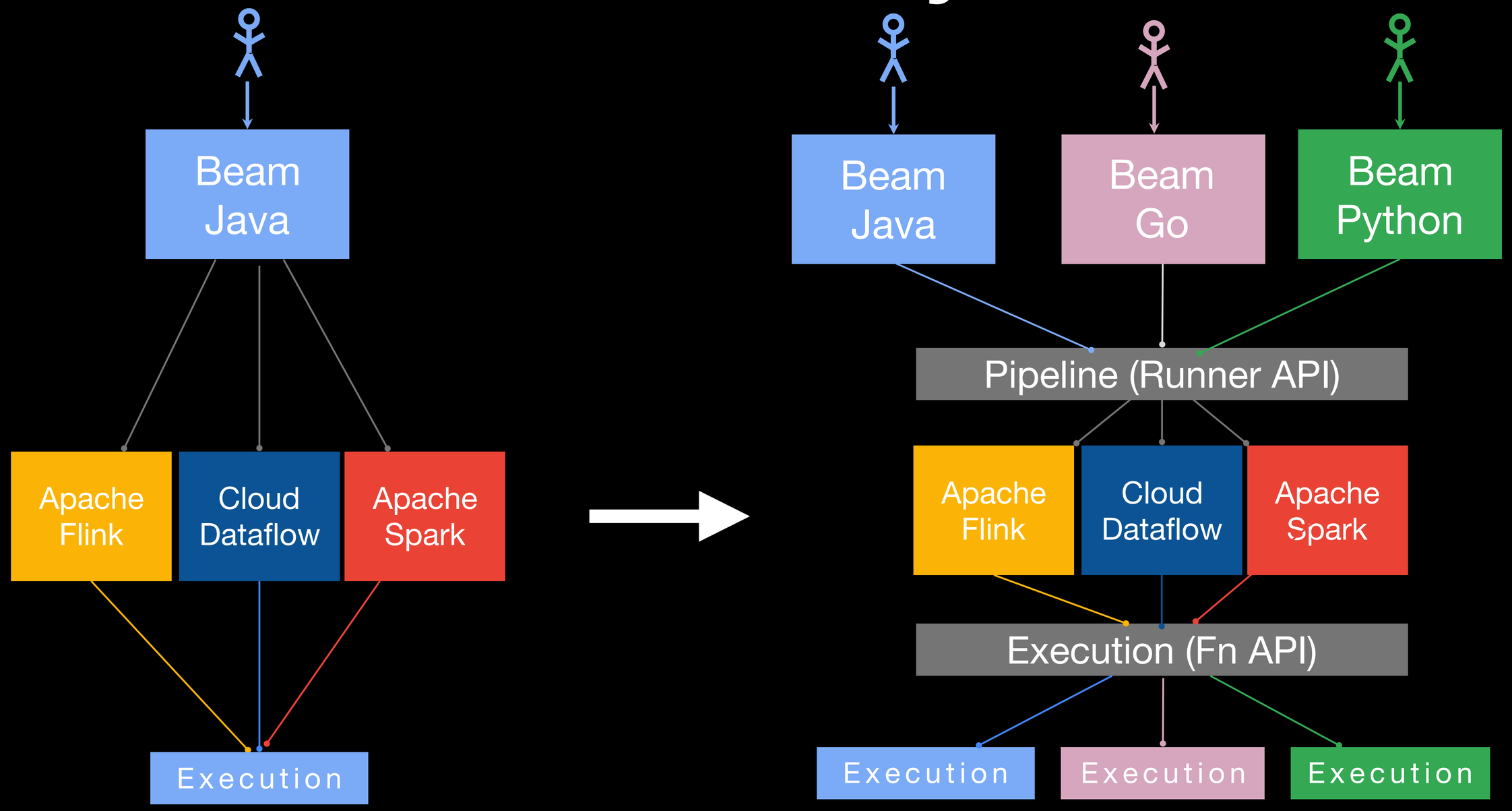
Portability



Portability



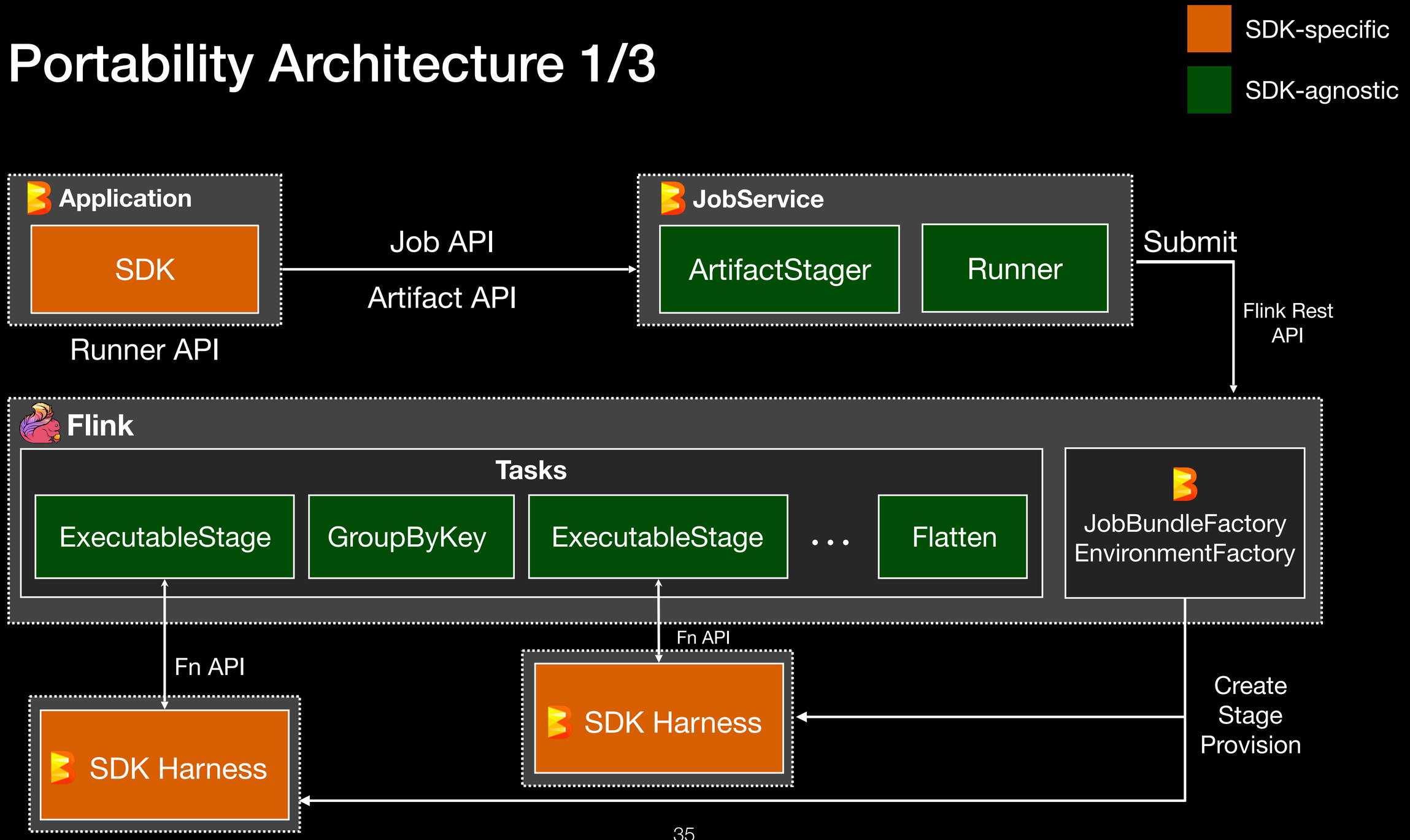
Portability



Primitive Transforms

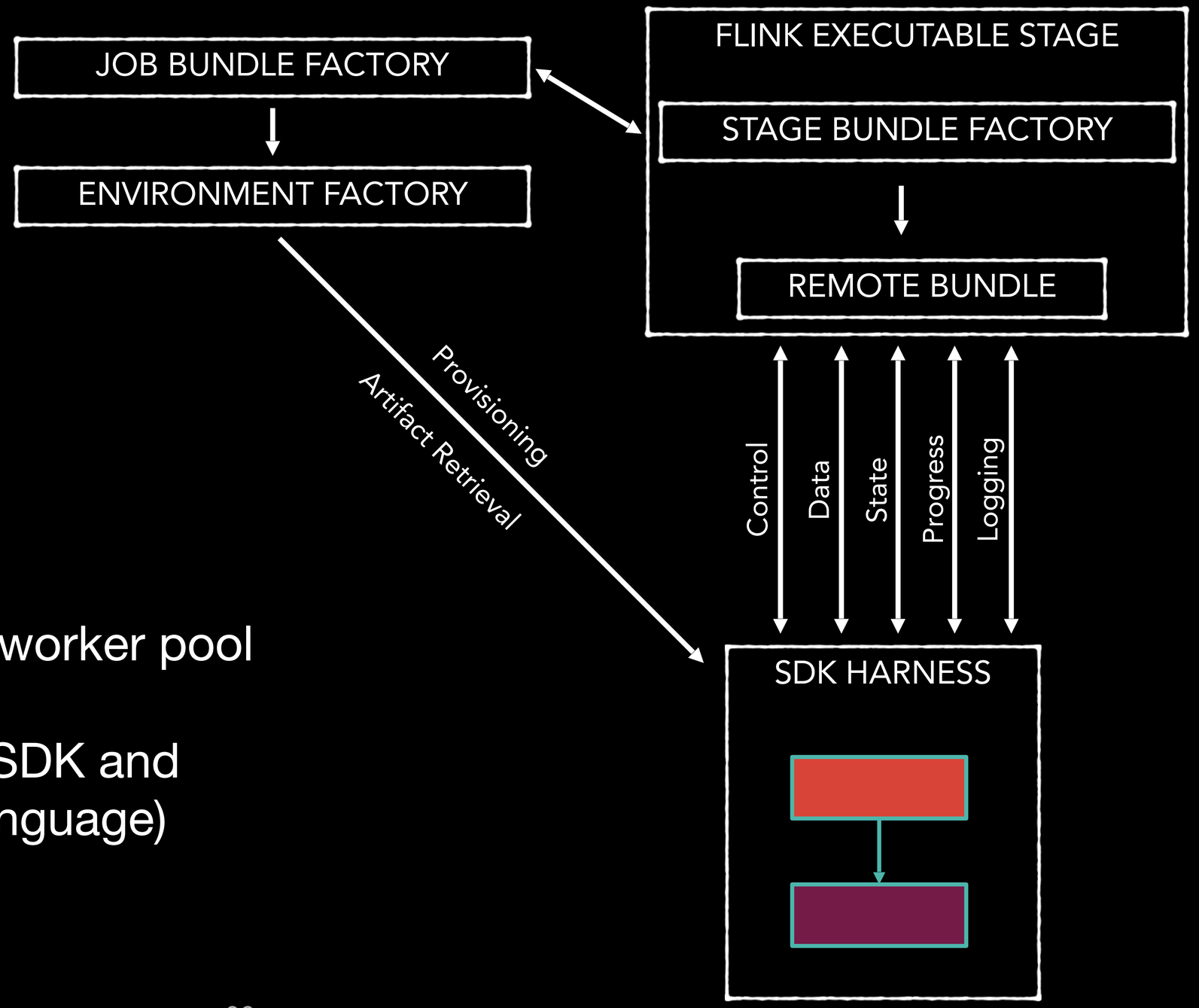
Classic	**Portable**
Read	Impulse + SDF
ParDo	ExecutableStage
GroupByKey	
Assign Windows	ExecutableStage
Flatten	

Portability Architecture 1/3



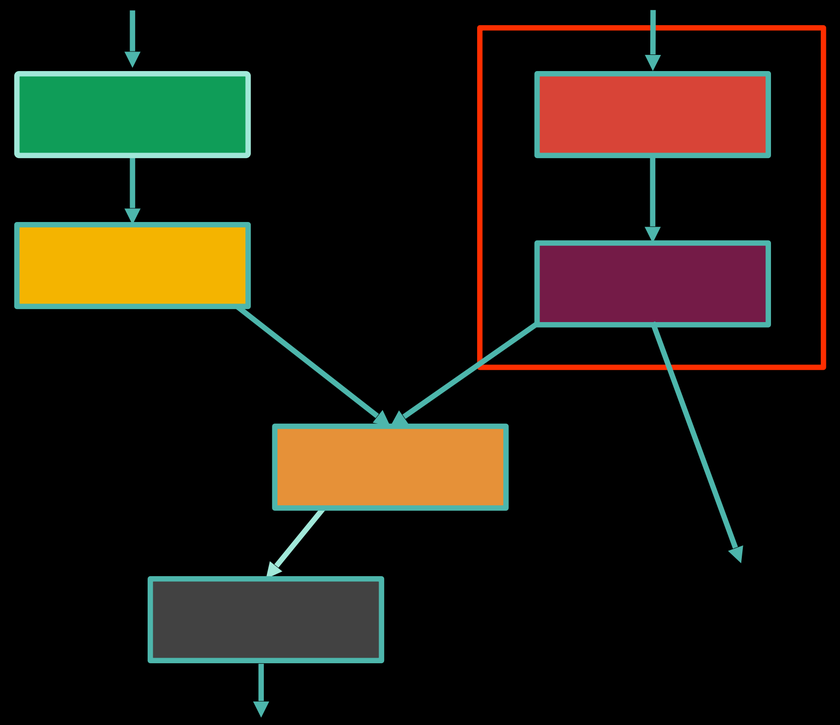
SDK Harness

- SDK Harness runs
  - in a Docker container
  - in a dedicated process
  - in an externally managed worker pool
  - embedded (only works if SDK and Runner share the same language)

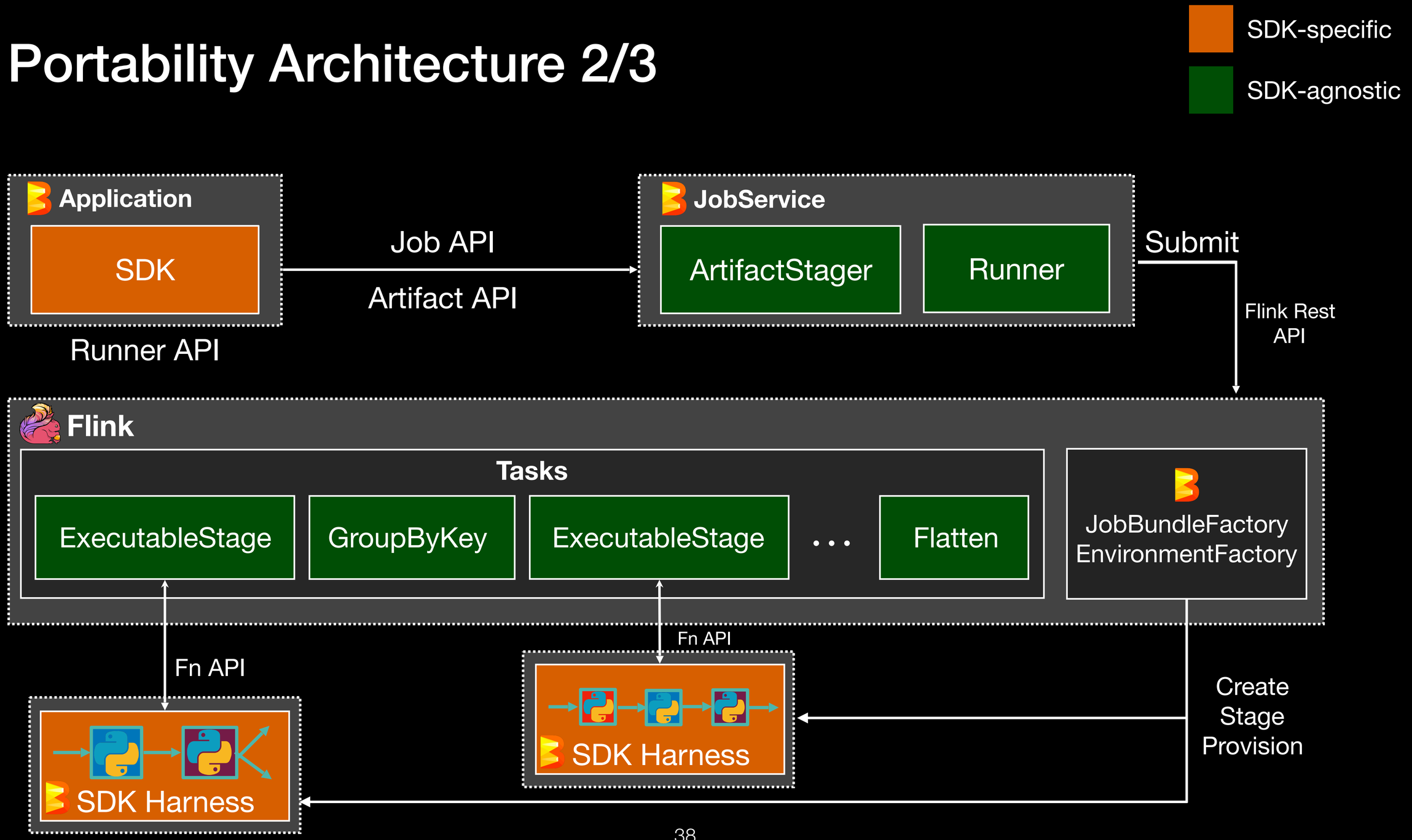


Pipeline Fusion

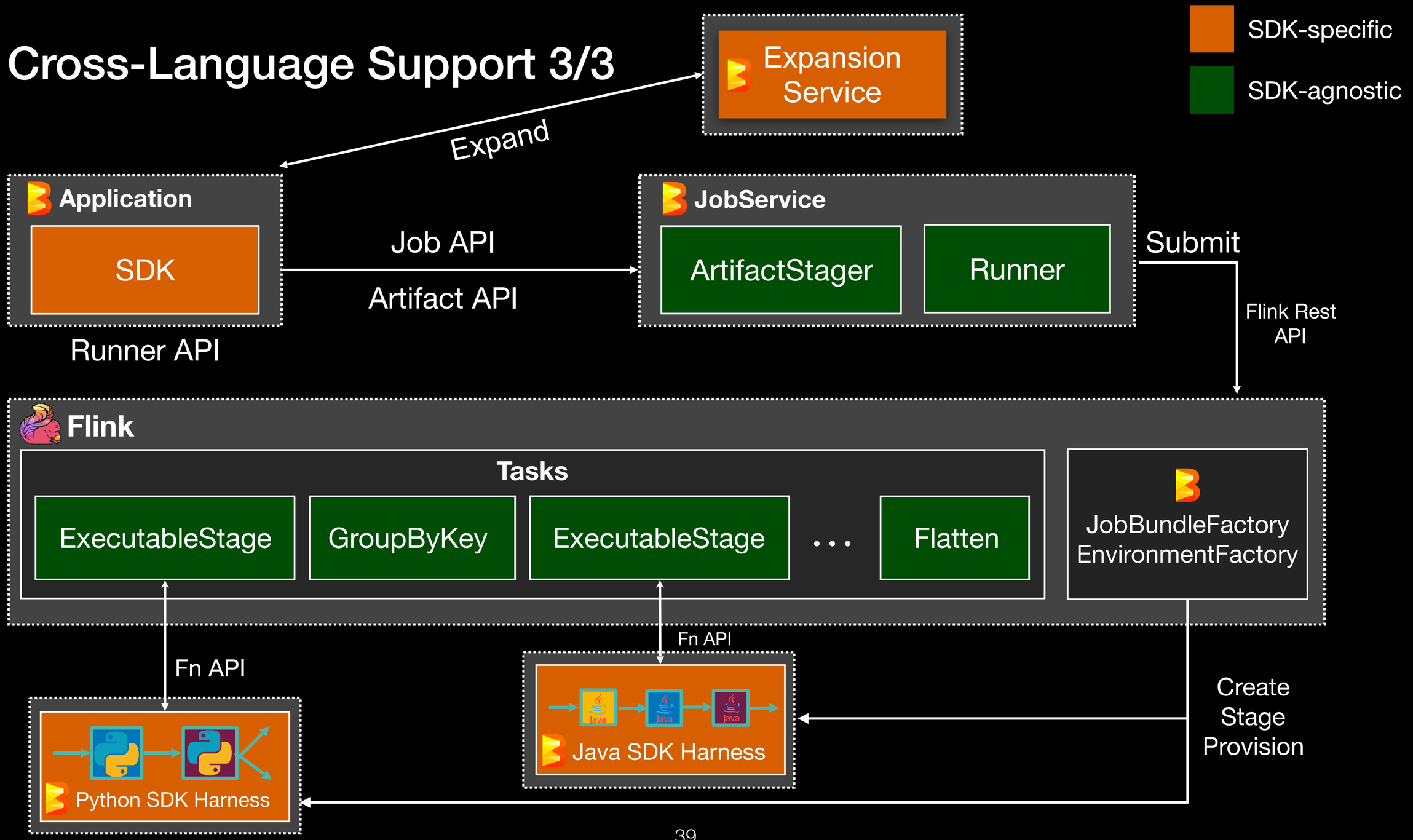
- SDK Harness environment comes at a cost
  - Serialization step before and after processing with SDK harness
- User defined functions should be chained and share the same environment



Portability Architecture 2/3



Cross-Language Support 3/3



Getting Started

General Information

- Beam docs
  - beam.apache.org/documentation/
- Getting Started
  - beam.apache.org/get-started/
- Flink Runner
  - beam.apache.org/documentation/runners/flink/

Java

- Guide: beam.apache.org/get-started/quickstart-java

1. Use the Maven Quickstart

2. Import project into IntelliJ

3. Set the Runner: `--runner=FlinkRunner`

    - Optionally: `flinkMaster="flinkCluster:8081"`

4. Run :)

Java Blueprint

```
FlinkPipelineOptions options = PipelineOptionsFactory.fromArgs(args).as(FlinkPipelineOptions.class);

// Options can be set here or supplied via arguments
options.setStreaming(true); // Also via --streaming
options.setRunner(FlinkRunner.class); // Also via --runner=FlinkRunner

Pipeline p = Pipeline.create(options);

PCollection<String> hamlet = p
    .apply(Create.of("to", "be", "or", "not", "to", "be")));

PCollection<KV<String, Long>> wordCounts = hamlet
    .apply(Count.perElement());

wordCounts
    .apply(ToString.Kvs())
    .apply(TextIO.write().to("/output/path"));

p.run();
```

Python

- https://beam.apache.org/get-started/quickstart-py

  1. `virtualenv --python=python3 env && source env/bin/activate`
  2. `pip install apache_beam`
  3. Write your pipeline
  4. Supply `--runner=FlinkRunner`
     - Optionally: `--flink_master=flinkCluster:8081`
  5. `Run :)`

Python Blueprint

```
import apache_beam as beam

# By default this will load the command-line arguments
options = beam.PipelineOptions()

# This will automatically execute the pipeline
with beam.Pipeline(options=options) as p:
  (p
   | beam.Create(['to be or not to be'])
   | beam.Map(lambda el: el.split(' '))
   | beam.CombinePerKey(sum)
   | beam.Map(lambda kv: str(kv))
   | beam.io.filesio.WriteToFiles(path='output/path')
   )
```

SQL

- https://beam.apache.org/documentation/dsls/sql/overview/
- Add artifact: org.apache.beam:beam-sdks-java-extensions-sql
  - Maven

```
<dependency>
    <groupId>org.apache.beam</groupId>
    <artifactId>beam-sdks-java-extensions-sql</artifactId>
    <version>2.20.0</version>
</dependency>
```

  - Gradle

```
compile group: 'org.apache.beam',
    name: 'beam-sdks-java-extensions-sql',
    version: '2.20.0'
```

SQL Blueprint

```
Schema schema =
    Schema
        .builder()
        .addInt32Field("id")
        .addStringField("name")
        .build();


PCollection<Row> input = pipeline
    .apply(Create.of(Row.withSchema(schema).addValues(1, "Max").build()));

PCollection<Row> selectMax = input.apply(
    SqlTransform.query("SELECT name FROM PCOLLECTION WHERE id=1"));
```

```java
public static class MyData {
  public int id;
  public String name;
}
```

SQL Blueprint

```java
Schema schema = POJOUtils.schemaFromPojoClass(
    OptionsTest.class,
    JavaFieldSchema.JavaFieldTypeSupplier.INSTANCE);
```

```java
PCollection<Row> input = pipeline
    .apply(Create.of(Row.withSchema(schema).addValues(1, "Max").build()));

PCollection<Row> selectMax = input.apply(
    SqlTransform.query("SELECT name FROM PCOLLECTION WHERE id=1"));
```

Conclusion

Apache Beam + Apache Flink

1. Unified API for batch and streaming
2. Programming language of your choice
3. ~~Execution engine of your choice~~
   The robustness and speed of Apache Flink

Thank you

Visit beam.apache.org

Read the docs

Subscribe to the mailing lists

Follow @ApacheBeam

Register for Beam Summit Digital!

August 24-28, 2020

beamsummit.org

Maximilian Michels

maximilianmichels.com

@stadtlegende