TOWARDS PORTABILITY AND BEYOND

DATA PROCESSING WITH

APACHE BEAM

Maximilian Michels
mxm@apache.org
@stadtlegende
maximilianmichels.com

BEAM VISION

Write Pipeline

Execute

samza

SDKs **Runners** **Backends**

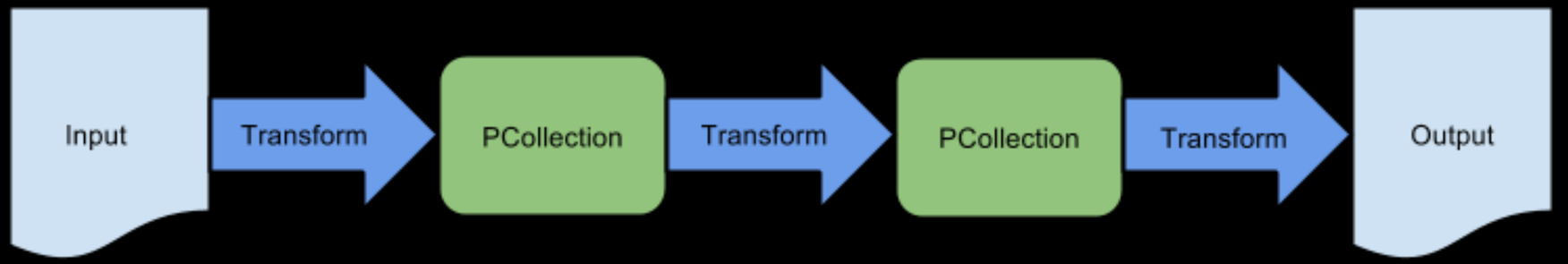
THE BEAM MODEL

- Unified **batch** and **stream** programming model
- **Stream:** Batch is just a bounded stream
- Massively parallelizable **Transformations**
- **Event Time** as an explicit concept

2015 VLDB: “The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing”

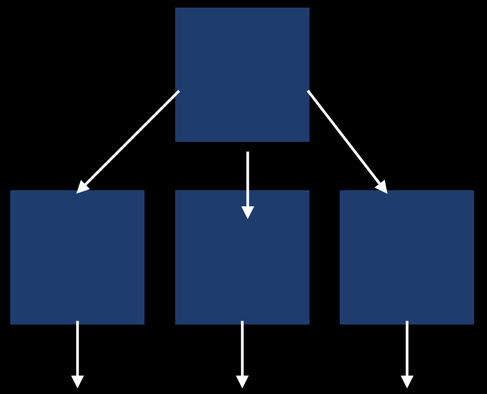
BEAM API

THE BEAM API



- Pipeline: An (acyclic) graph of PCollections
- [Output PCollection] = [Input PCollection].apply([Transform])
  - Pipeline p = `Pipeline.create(options)`
  - `PCollection pCollection = p.apply(…).apply(…).…`
  - `p.run()`

INPUT

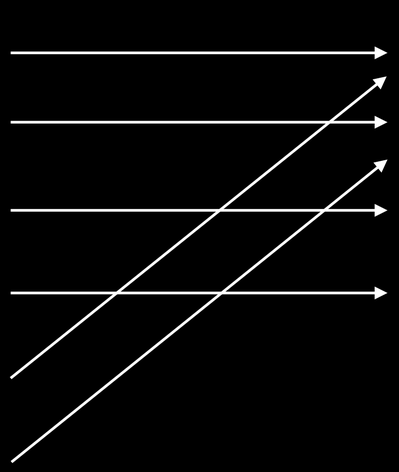


```
Pipeline p = Pipeline.create(options);

PCollection<String> input1 = p.apply(
    “ReadMyFile”,
    TextIO.read().from(“protocol://path/to/some/inputData.txt”));

PCollection<String> input2 = p.apply(
    Create.of(“DataEngConf”, “is”, “awesome”));

PCollection<KV<Long, String>> input3 = p.apply(
    KafkaIO.<Long, String>read()
        .withBootstrapServers("broker_1:9092,broker_2:9092")
        .withTopic("my_topic")
        .withKeyDeserializer(LongDeserializer.class)
        .withValueDeserializer(StringDeserializer.class))
```

CORE PRIMITIVES TRANSFORMS

ParDo

```
input -> output

“to” -> KV<“to”, 1>
“be” -> KV<“be”, 1>
“or” -> KV<“or”, 1>
“not”-> KV<“not”,1>
“to” -> KV<“to”, 1>
“be” -> KV<“be”, 1>
```



```
KV<k,v>… -> KV<k, [v…]>

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

Map/Reduce Phase

Shuffle Phase

CORE PRIMITIVES TRANSFORMS

ParDo

```
PCollection<String> words = ...;

PCollection<KV<String, Integer>> wordCounts = words.apply("AssignWordCounts",
    ParDo.of(new DoFn<String, KV<String, Integer>>() {
        @ProcessElement
        public void processElement(@Element String word, OutputReceiver<Integer> out) {
          out.output(KV.of(word, 1));
        }
      }
    ));
```

GroupByKey

```
PCollection<KV<String, Iterable<Integer>>> groupByWordCounts =
    wordCounts.apply("GroupByWords", GroupByKey.create());
```

COMPOSITE TRANSFORMS

Combine

```
PCollection<KV<String, Integer> wordCounts = …

PCollection<KV<String, Integer> combinedWordCounts = wordCounts.apply(
Combine.perKey(new SerializableFunction<Iterable<Integer>, Integer> {
  @Override
  public Integer apply(Iterable<Integer> input) {
    int sum = 0;
    for (int count : input) {
      sum += count;
    }
    return sum;
  }
});
```

For more sophisticated combing, you can define a CombineFn.

(Map/) Shuffle/Reduce Phase

MORE TRANSFORMS

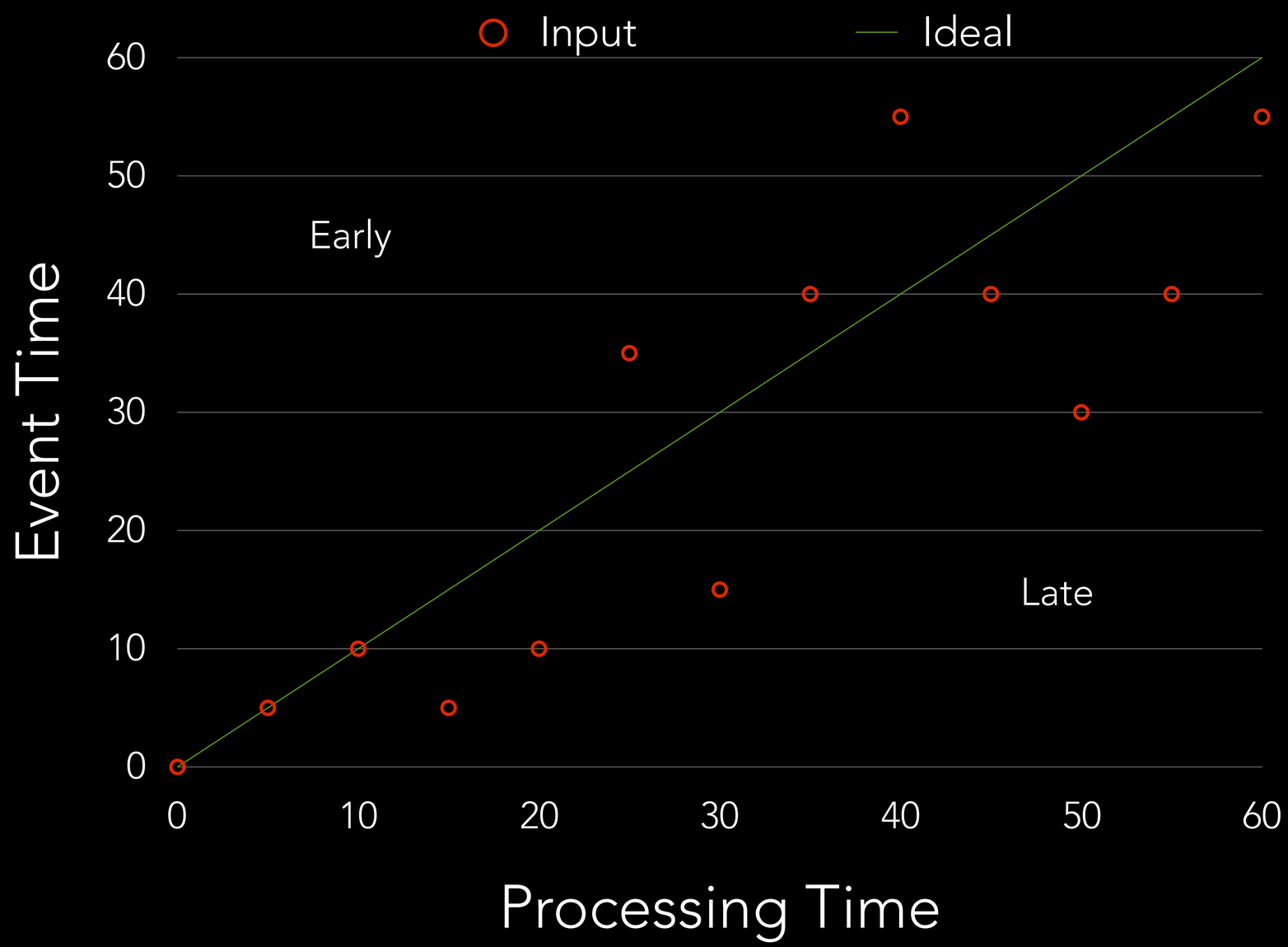
- CoGroupByKey (Join)
- Flatten
- Partition
- Define your own!
- Also: Side Inputs / Multiple Outputs / State / Timers

PROCESSING UNBOUNDED DATA

PROCESSING (UN)BOUNDED DATA

- Stream are unbounded by nature
- **Windows** group data according to a windowing function
- **Triggers** decide when to kick off execution of windows
- Event Time is the predominant domain for windowing
  - Every element has an event **timestamp**
  - **Watermark** indicates the current event time

EVENT VS PROCESSING TIME



EVENT VS PROCESSING TIME: STAR WARS

Year	1977	1980	1983	1999	2002	2005	2015	2017	2019
Episode	IV	V	VI	I	II	III	VII	VIII	IX

These episodes appeared out-of-order

Year	1999	2002	2005	1977	1980	1983	2015	2017	2019
Episode	I	II	III	IV	V	VI	VII	VIII	IX

That's much better!

PROCESSING TIME

EVENT TIME

THE BEAM MODEL: A SIMPLE PIPELINE

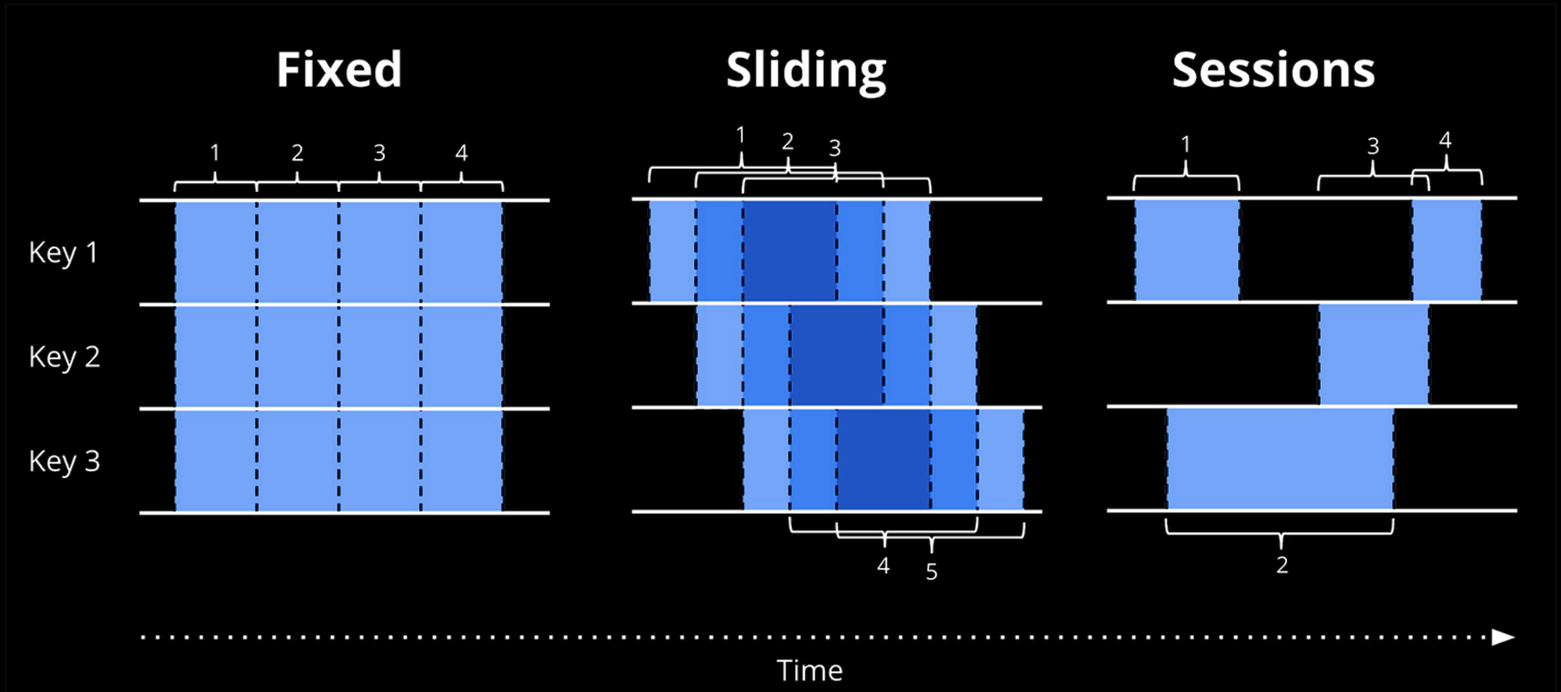
What, Where, When, How

```
PCollection<KV<String, Integer>> scores = input
        .apply(Sum.integersPerKey());
```

A SIMPLE PIPELINE: WINDOWING

What, Where, When, How

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))))
    .apply(Sum.integersPerKey());
```



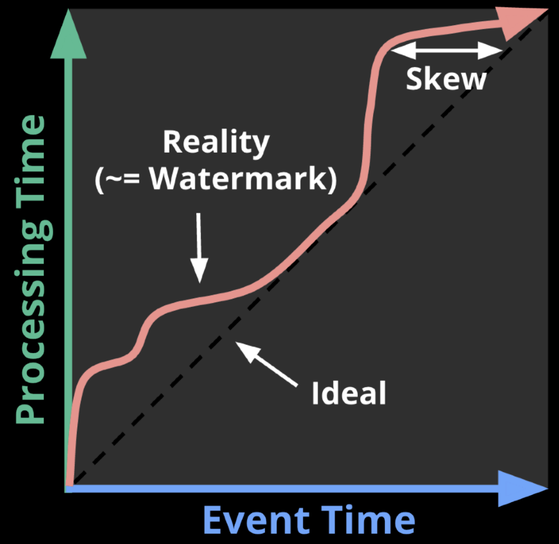
Window Types

- Global Window
- Fixed Time Windows
- Sliding Time Windows
- Per-Session Windows

A SIMPLE PIPELINE: TRIGGERING

What, Where, When, How

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark())
    .apply(Sum.integersPerKey());
```



Triggers

- Event time
- Processing
- Data-driven
- Composite triggers

A SIMPLE PIPELINE: TRIGGERING

What, Where, When, How

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark()
              .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
              .withLateFirings(AtCount(1)))
    .apply(Sum.integersPerKey());
```

Triggers

- Event time
- Processing
- Data-driven
- Composite triggers

A SIMPLE PIPELINE: REFINEMENT

What, Where, When, How

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark()
              .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
              .withLateFirings(AtCount(1))
           .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

JAVA VS PYTHON


```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark()
              .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
              .withLateFirings(AtCount(1))
           .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

```
scores = input
    | WindowInto(FixedWindows(120)
                 trigger=AfterWatermark(
                     early=AfterProcessingTime(60),
                     late=AfterCount(1))
                 accumulation_mode=ACCUMULATING)
    | CombinePerKey(sum)
```

EXECUTE WITH CHOICE OF RUNNER

```
input  = pipeline | ReadFromText("/path/to/text*") | Map(lambda line: ...)
scores = input
       | WindowInto(FixedWindows(120)
                    trigger=AfterWatermark(
                        early=AfterProcessingTime(60),
                        late=AfterCount(1))
                    accumulation_mode=ACCUMULATING)
       | CombinePerKey(sum))
       | WriteToText("/path/to/outputs")

MyRunner().run(pipeline)
```

RUNNERS

RUNNERS

Direct

Apache Flink

Apache Spark

Apache Apex

Apache Samza

Google Cloud Dataflow

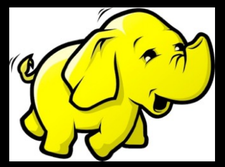
Apache Gearpump

WIP

Ali Baba JStorm

Apache Storm

IBM Streams



Hadoop MapReduce

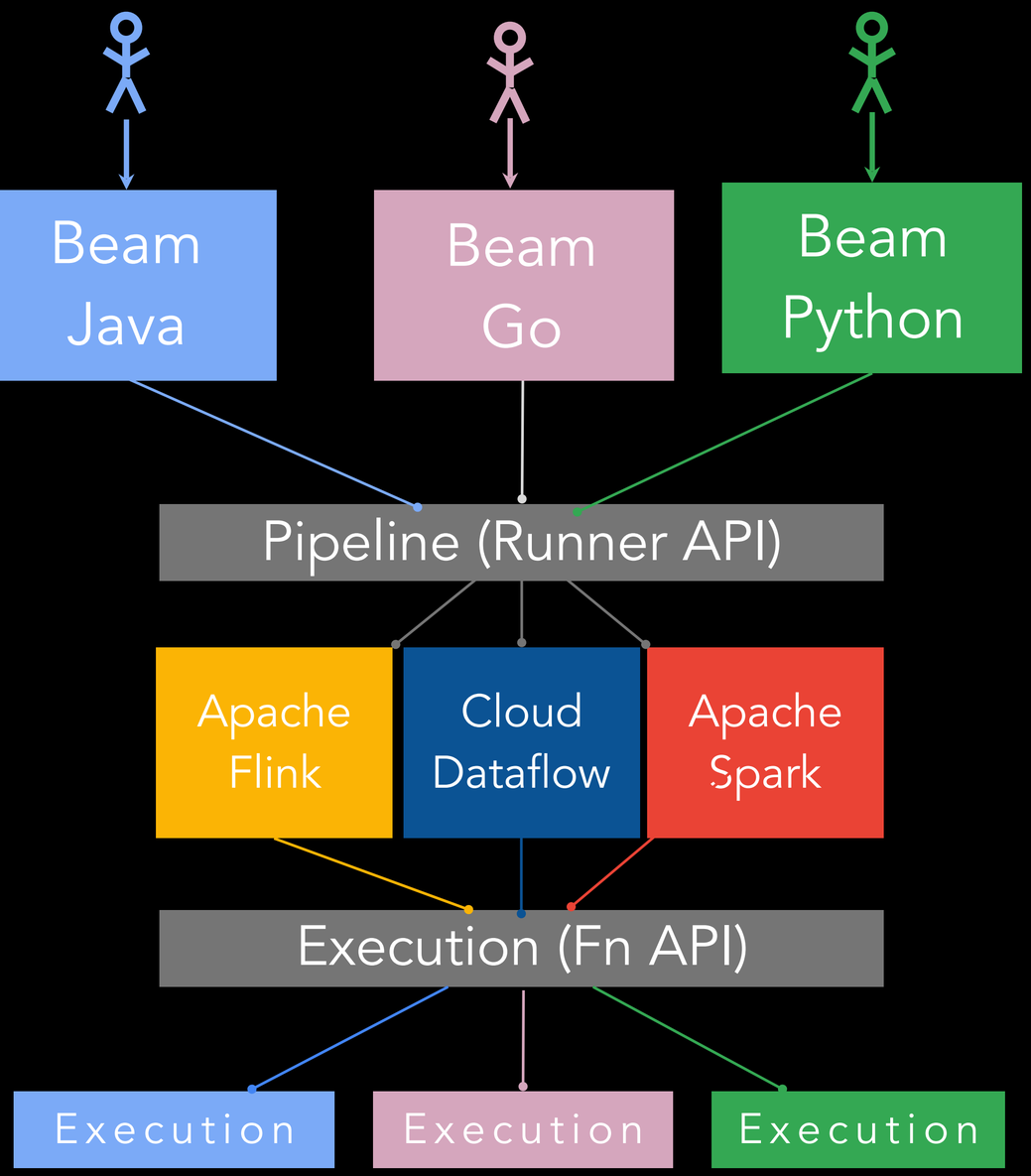
PROBLEMS WITH THIS APPROACH

- All execution backends written in Java
- Can Python run on top of Java?
- N Runners, M languages => N*M translation paths?
- Submission Flow

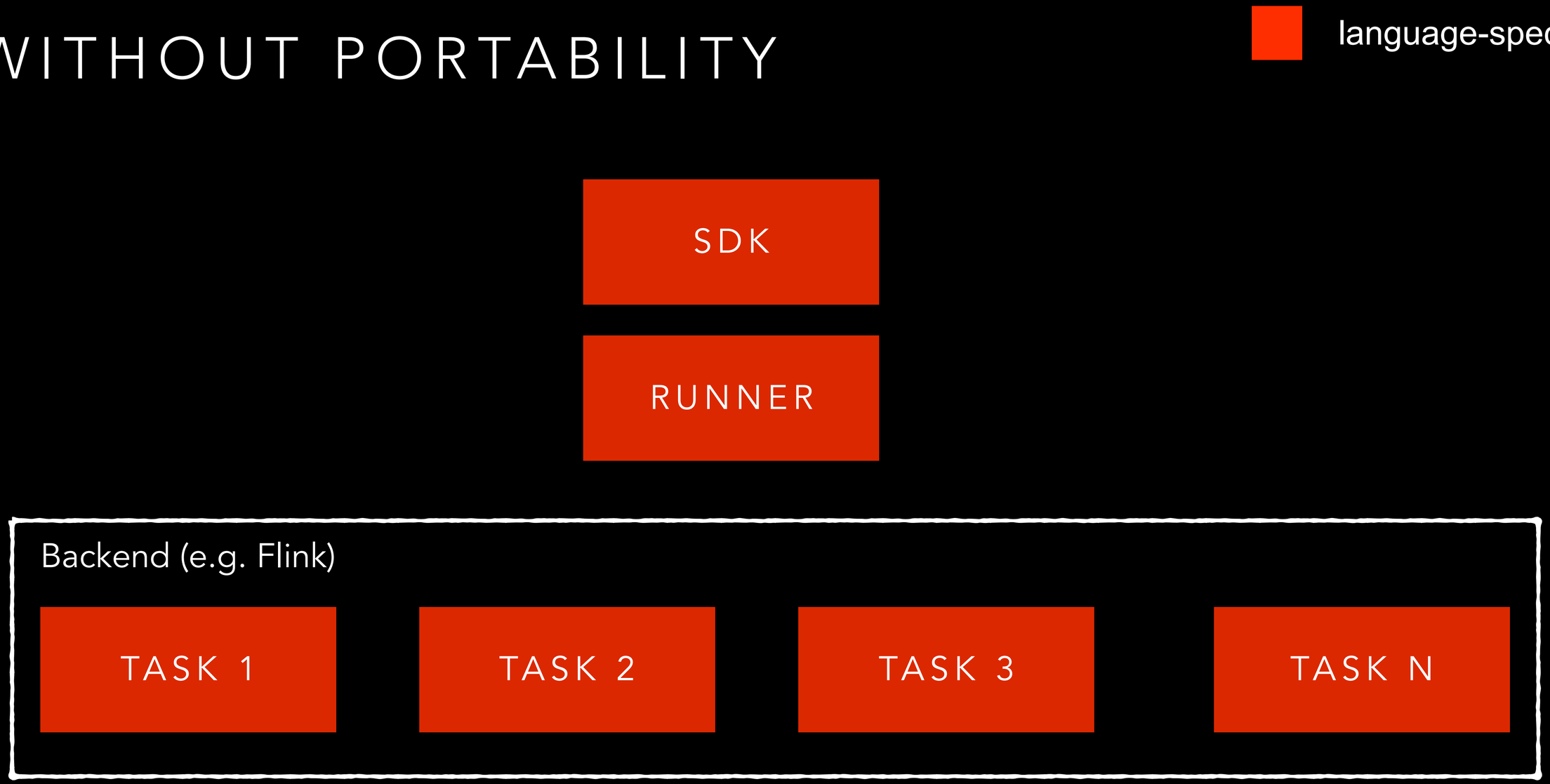
PORTABILITY

THE BEAM VISION

- Portability in Beam means Pipelines can be written and executed in any supported SDK (Java/Python/Go)
- Pipelines also contain language-specific code (e.g. map/reduce functions)
- Libraries of the language can be used (!)

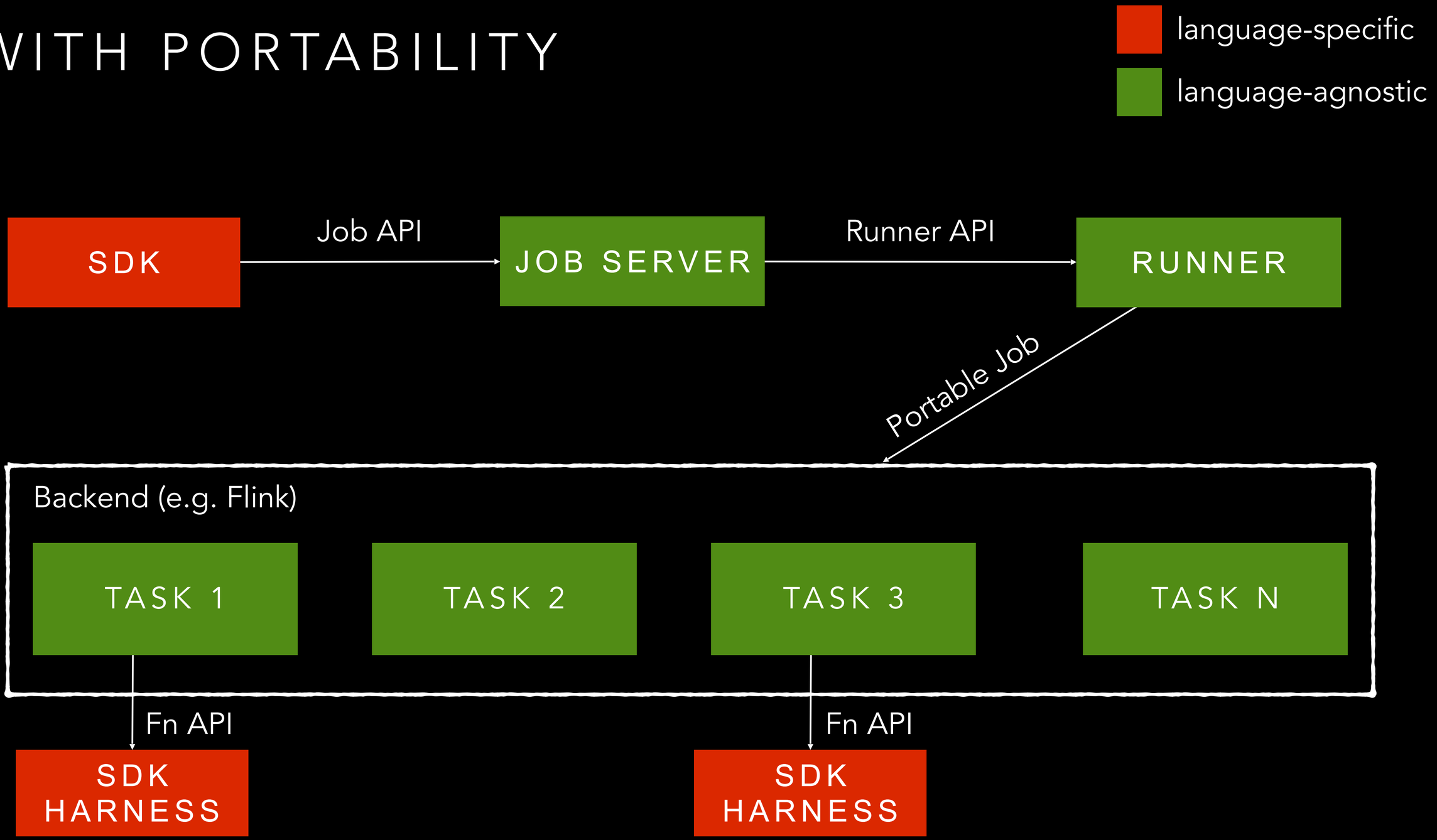


WITHOUT PORTABILITY



All components are tight to a single language

WITH PORTABILITY



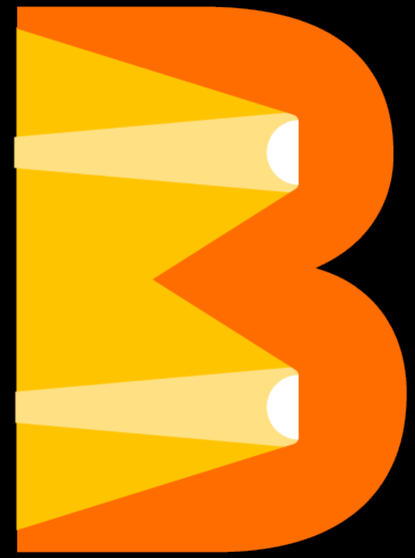
WHAT IS THE STATE OF THE BEAM VISION?

- |Runners| * |SDKs| is not feasible
- Instead each SDK only implements a Portable Runner
- Flink Runner is the first OSS Runner to work with the Portable Runner
- Cross-language pipelines in the future
- Overhead has been measured to be 5-10%, could be less in real-world scenarios

DEMO TIME

HOW TO GET INVOLVED

- **Visit** beam.apache.org
  - Documentation
  - Examples
- **Subscribe** to the mailing lists:
  - user-subscribe@beam.apache.org
  - dev-subscribe@beam.apache.org
- **Join** the ASF Slack channel #beam
- **Follow** @ApacheBeam



Maximilian Michels
mxm@apache.org
@stadtlegende
maximilianmichels.com