Introduction to Apache Flink

2015. 12. 22.
박치완 <chiwanpark@apache.org>

발표자 소개

박치완 <chiwanpark@apache.org>

대용량 데이터 처리에 관심이 많은 학생

Apache Flink Committer (since Jun. 2015)

오늘은 이런 이야기를 해보려고 합니다.

MapReduce와 Hadoop 소개

Post-MapReduce Framework로 나온 다양한 시도들

Apache Flink 소개

Apache Flink는 그래서 무엇이 좋은가

MapReduce

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of com-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy de-

2004년 구글이 발표한 대용량 데이터 처리 방식

이후 MapReduce와 분산 파일 시스템(GFS)을 구현한

으로 인해 폭발적인 인기

MapReduce

2015년 12월 16일
15656번 인용

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of com-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy de-

2004년 구글이 발표한 대용량 데이터 처리 방식

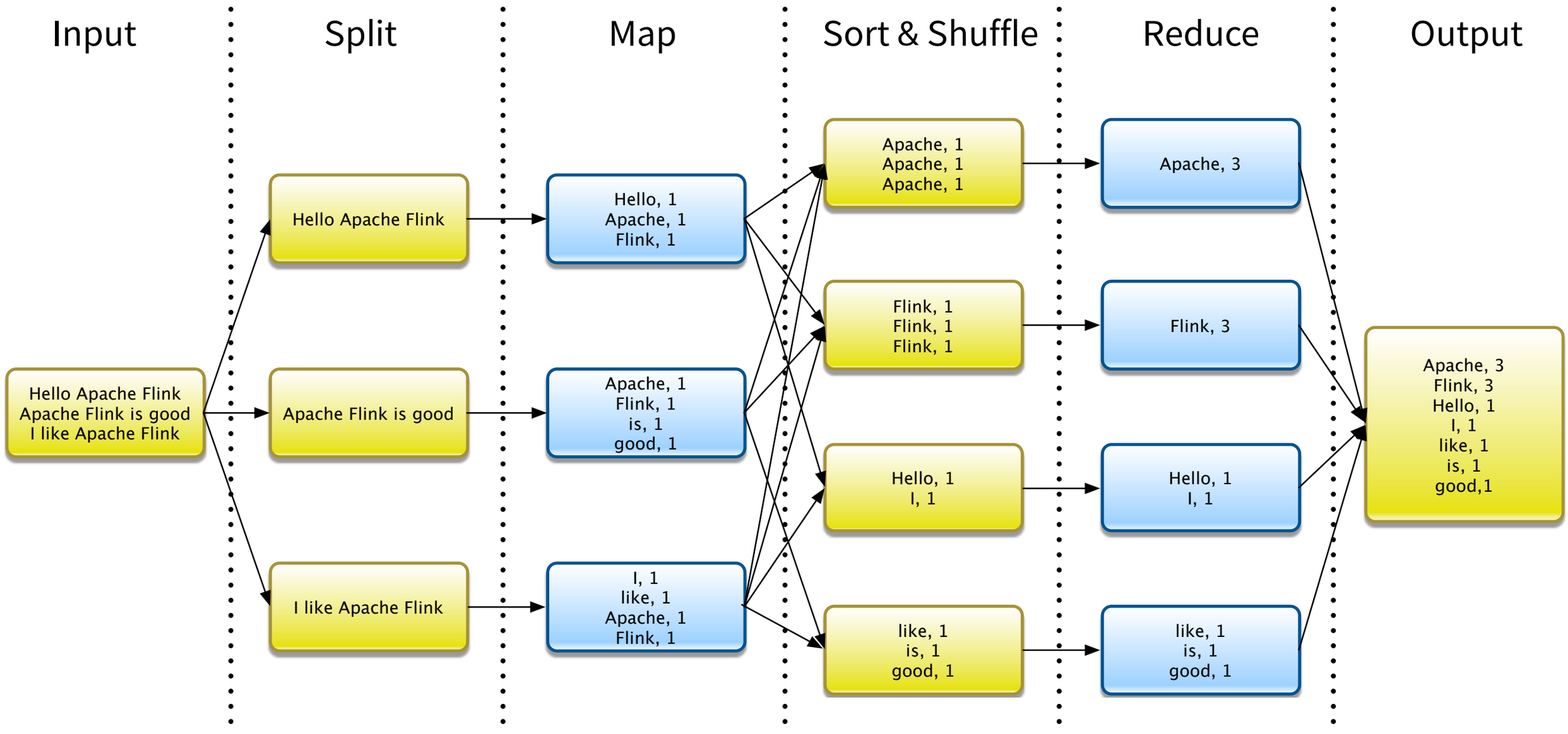
이후 MapReduce와 분산 파일 시스템(GFS)을 구현한

으로 인해 폭발적인 인기

MapReduce

전체 데이터의 단어별 개수를 세는 작업

Wordcount로 보는 MapReduce의 이해



Hadoop

Hadoop이 나오기 전 분산처리를 할 때 고민해야 하는 문제

클러스터 내 컴퓨터 간 데이터는 어떻게 주고 받지?

중간에 처리하던 컴퓨터가 오류로 인해 멈추면?

어떻게 해야 효율적인 분산처리가 될까?

데이터는 어떻게 분산시켜 놓지?

...

Hadoop

Hadoop이 나온 뒤 분산처리를 할 때 고민해야 하는 문제

Map에서 무슨 일을 할까?

Reduce에서 무슨 일을 할까?

Hadoop

그런데 Hadoop MapReduce가 장점만 있는 것은 아닙니다.

Hadoop

Hadoop MapReduce의 장점

Map과 Reduce만 생각하면 된다.
(정말?)

Hadoop

Hadoop의 단점

Map과 Reduce밖에 없다.

Hadoop

Hadoop의 단점 (보다 자세히)

split-map-sort-shuffle-reduce로 강제되는 파이프라인 구조

기본적으로 MapReduce의 입력은 1개만 가능

중간 결과를 디스크에 저장

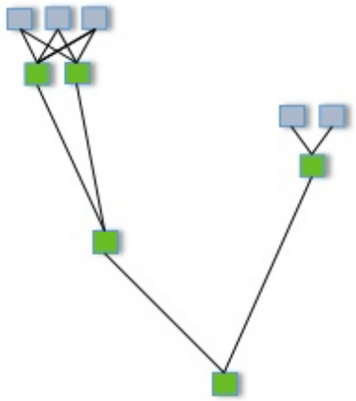
스트림 데이터에 대한 고려가 없음

그래서 사람들은 새로운 것을 만들기 시작했습니다.

새로운 것 (1) - Apache Tez

DAG (Directed Acyclic Graph) 형태로
대용량 데이터 처리를 수행하는 플랫폼

새로운 것 (1) - Apache Tez



입력 여러 개 받을 수
있다는 얘기입니다.

DAG (Directed Acyclic Graph) 형태로
대용량 데이터 처리를 수행하는 플랫폼

새로운 것 (2) - Apache Spark

디스크 대신 메모리를 적극적으로 사용함으로써
성능 향상을 꾀하는 대용량 데이터 처리 플랫폼

그리고, 새로운 것 (3) - Apache Flink

스트리밍 데이터 처리 엔진을 기반으로 만든
대용량 데이터 처리 플랫폼

Apache Flink

오픈소스 대용량 데이터 처리 플랫폼

유럽 대학들의 연합 연구 프로젝트인 Stratosphere의 결과물

2014년 4월, Apache Incubator에 들어와

2014년 12월, Top-level Project로 승격

현재[1] 커미터는 20명이고 컨트리뷰터는 146명

[1] 2014년 12월 19일 확인

Apache Flink

세련된 Java, Scala, Python API 제공

Web UI, Scala Shell 등 다양한 도구 지원

그래프 처리, 기계 학습 등을 위한 라이브러리 제공

똑똑한 Flink 런타임의 자동 최적화

오픈소스 진영에서 가장 진보된 스트리밍 데이터 처리 시스템

Hadoop WordCount Example

```
public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }
 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }
 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = new Job(conf, "wordcount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
 }
}
```

Flink WordCount Example

```
object WordCount extends App {
  val env = ExecutionEnvironment.getExecutionEnvironment
  val data = env.readTextFile(“hdfs://...“)
  val wc = data.flatMap(_.split("\\W+")).map((_, 1)).groupBy(0).sum(1)

  wc.writeAsText(“hdfs://...“)
  env.execute(“Flink WordCount Example”)
}
```

Flink WordCount Example

1. 데이터를 불러와서 (split)

5. reduce를 수행

```
object WordCount extends App {
  val env = ExecutionEnvironment.getExecutionEnvironment
  val data = env.readTextFile("hdfs://...")
  val wc = data.flatMap(_.split("\\W+")).map((_, 1)).groupBy(0).sum(1)

  wc.writeAsText("hdfs://...")
  env.execute("Flink WordCount Example")
}
```

2. map을 수행하고

3. map을 또 수행하고

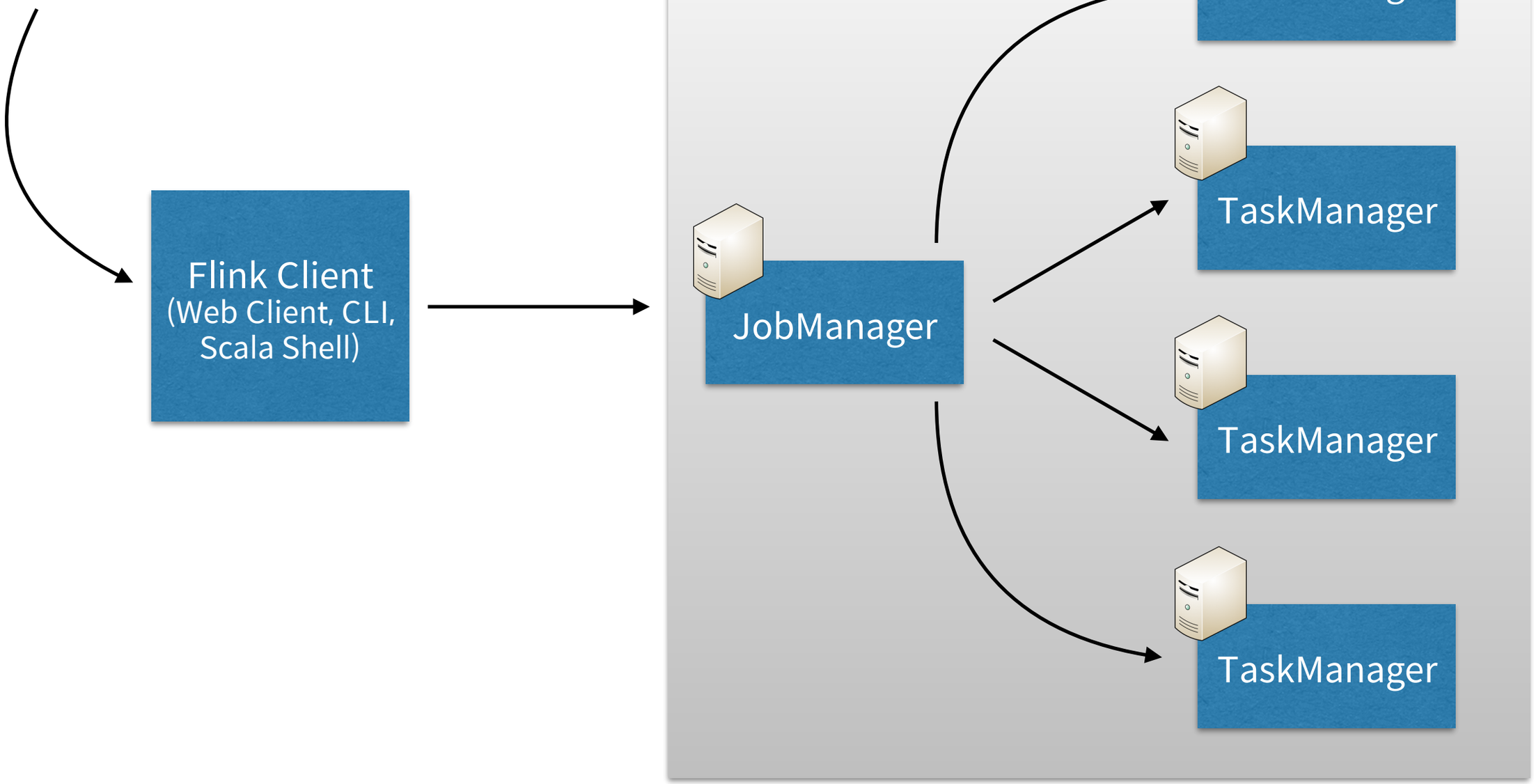
4. 단어 단위로 값을 모아서 (sort-shuffle)

6. 출력

Easy Deploy

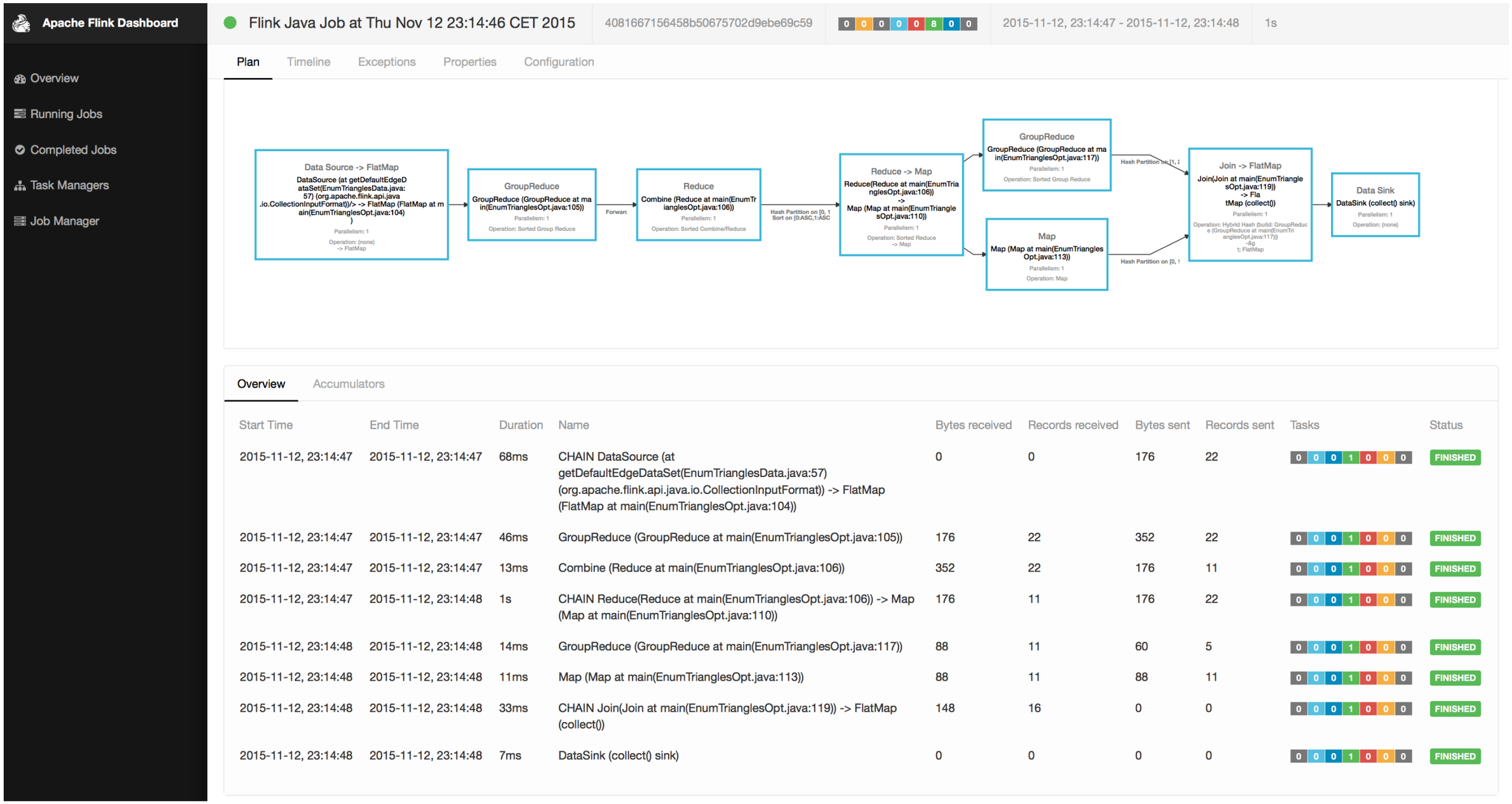
```scala
object WordCount extends App {
  val env = ExecutionEnvironment.getExecutionEnvironment
  val data = env.readTextFile(“hdfs://...“)
  val wc = data.flatMap(_.split("\\W+")).map((_, 1)).groupBy(0).sum(1)

  wc.writeAsText(“hdfs://...“)
  env.execute(“Flink WordCount Example”)
}
```



Flink Scala Shell

Flink Web UI



Start Time	End Time	Duration	Name	Bytes received	Records received	Bytes sent	Records sent	Tasks	Status
2015-11-12, 23:14:47	2015-11-12, 23:14:47	68ms	CHAIN DataSource (at getDefaultEdgeDataSet(EnumTrianglesData.java:57) (org.apache.flink.api.java.io.CollectionInputFormat)) -> FlatMap (FlatMap at main(EnumTrianglesOpt.java:104))	0	0	176	22	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:47	2015-11-12, 23:14:47	46ms	GroupReduce (GroupReduce at main(EnumTrianglesOpt.java:105))	176	22	352	22	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:47	2015-11-12, 23:14:47	13ms	Combine (Reduce at main(EnumTrianglesOpt.java:106))	352	22	176	11	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:47	2015-11-12, 23:14:48	1s	CHAIN Reduce(Reduce at main(EnumTrianglesOpt.java:106)) -> Map (Map at main(EnumTrianglesOpt.java:110))	176	11	176	22	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:48	2015-11-12, 23:14:48	14ms	GroupReduce (GroupReduce at main(EnumTrianglesOpt.java:117))	88	11	60	5	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:48	2015-11-12, 23:14:48	11ms	Map (Map at main(EnumTrianglesOpt.java:113))	88	11	88	11	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:48	2015-11-12, 23:14:48	33ms	CHAIN Join(Join at main(EnumTrianglesOpt.java:119)) -> FlatMap (collect())	148	16	0	0	0 0 0 1 0 0 0	FINISHED
2015-11-12, 23:14:48	2015-11-12, 23:14:48	7ms	DataSink (collect() sink)	0	0	0	0	0 0 0 1 0 0 0	FINISHED

Gelly

Graph processing library on Flink

Google Pregel 형태의 Vertex-centric Iteration 지원

Gather-Sum-Apply Iteration 지원

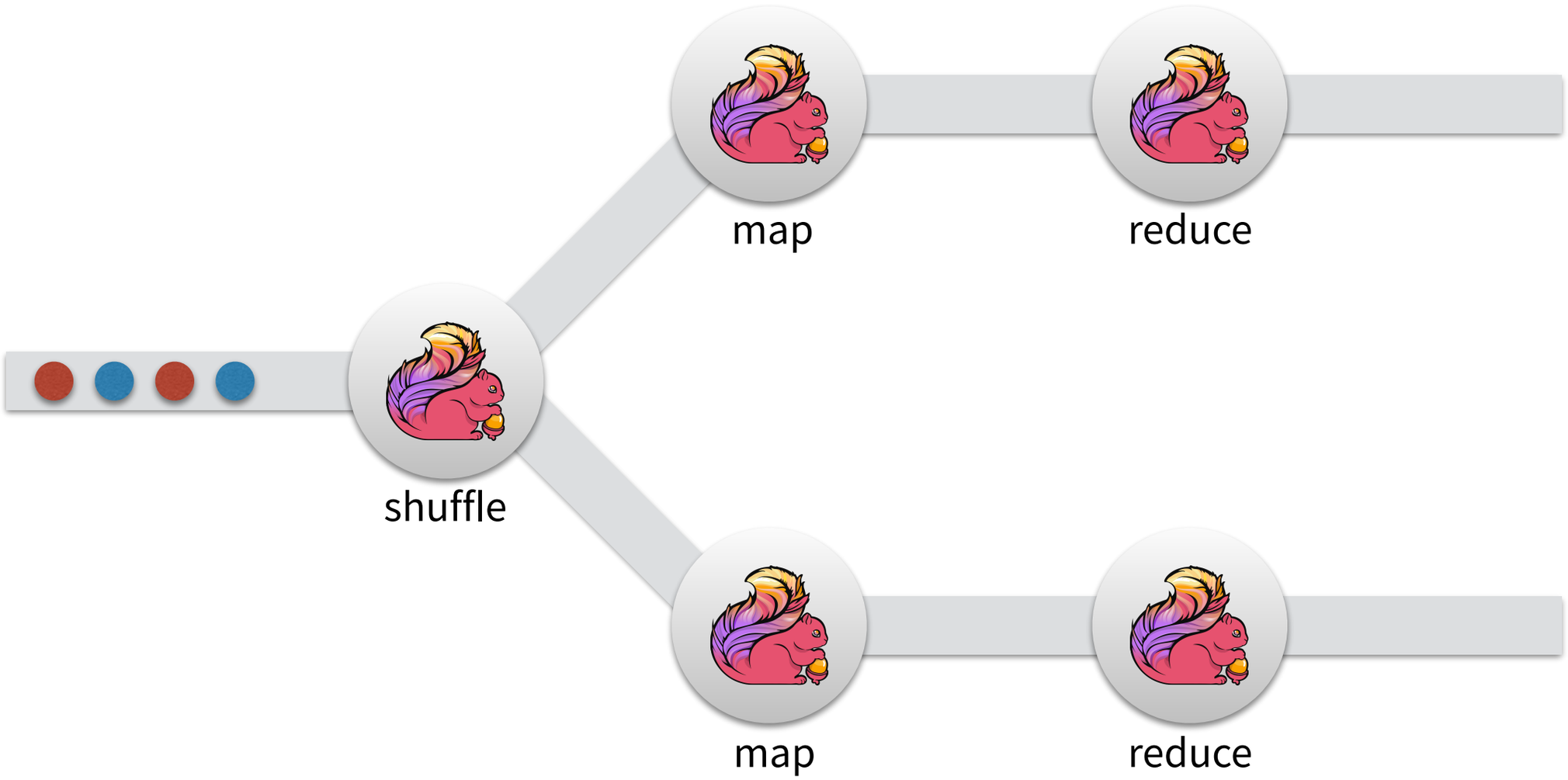
FlinkML

Machine learning library on Flink

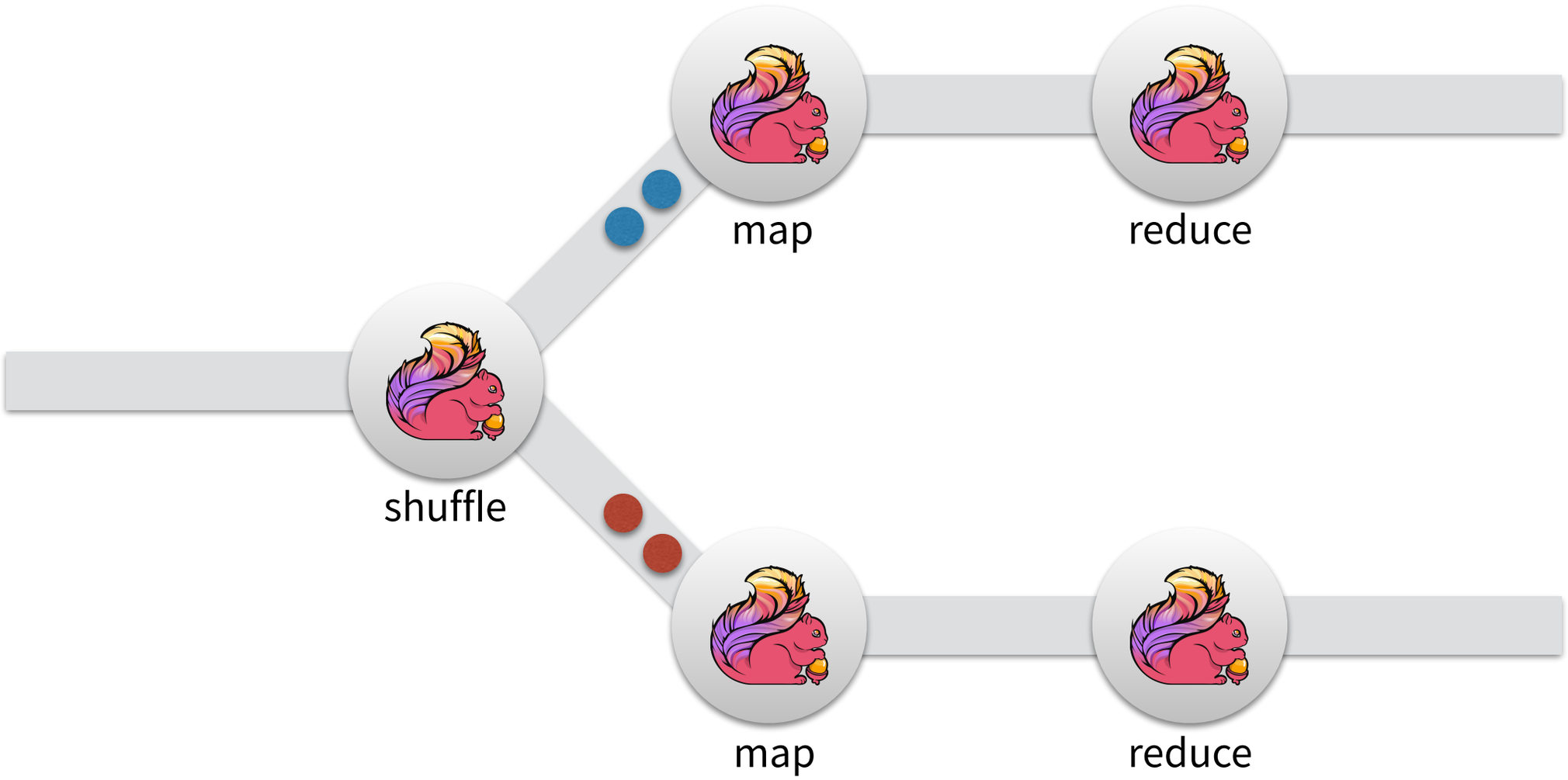
scikit-learn 스타일의 파이프라인 지원

CoCoA, Linear Regression, ALS 등이 구현되어 있음

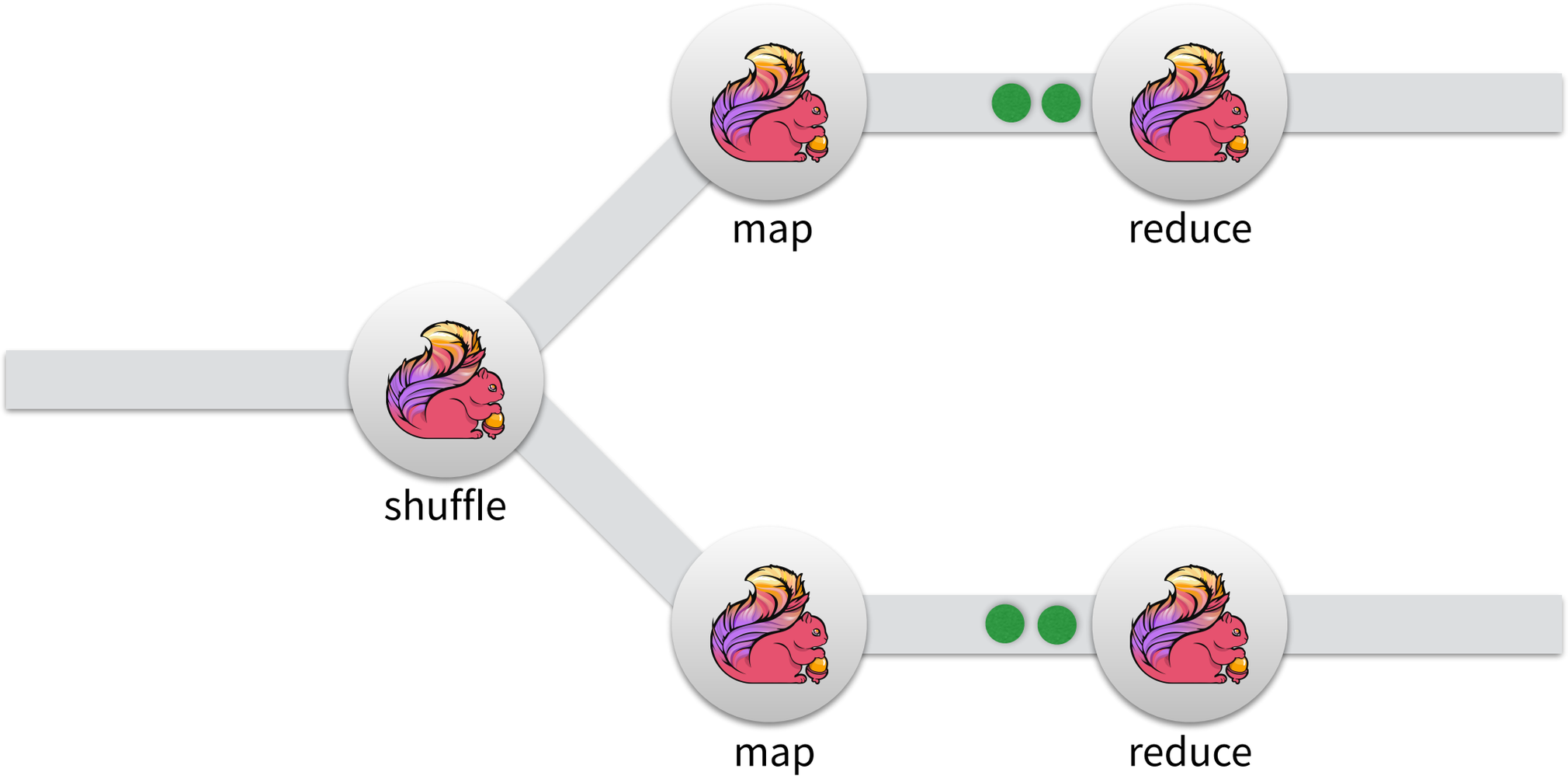
Streaming Dataflow Engine



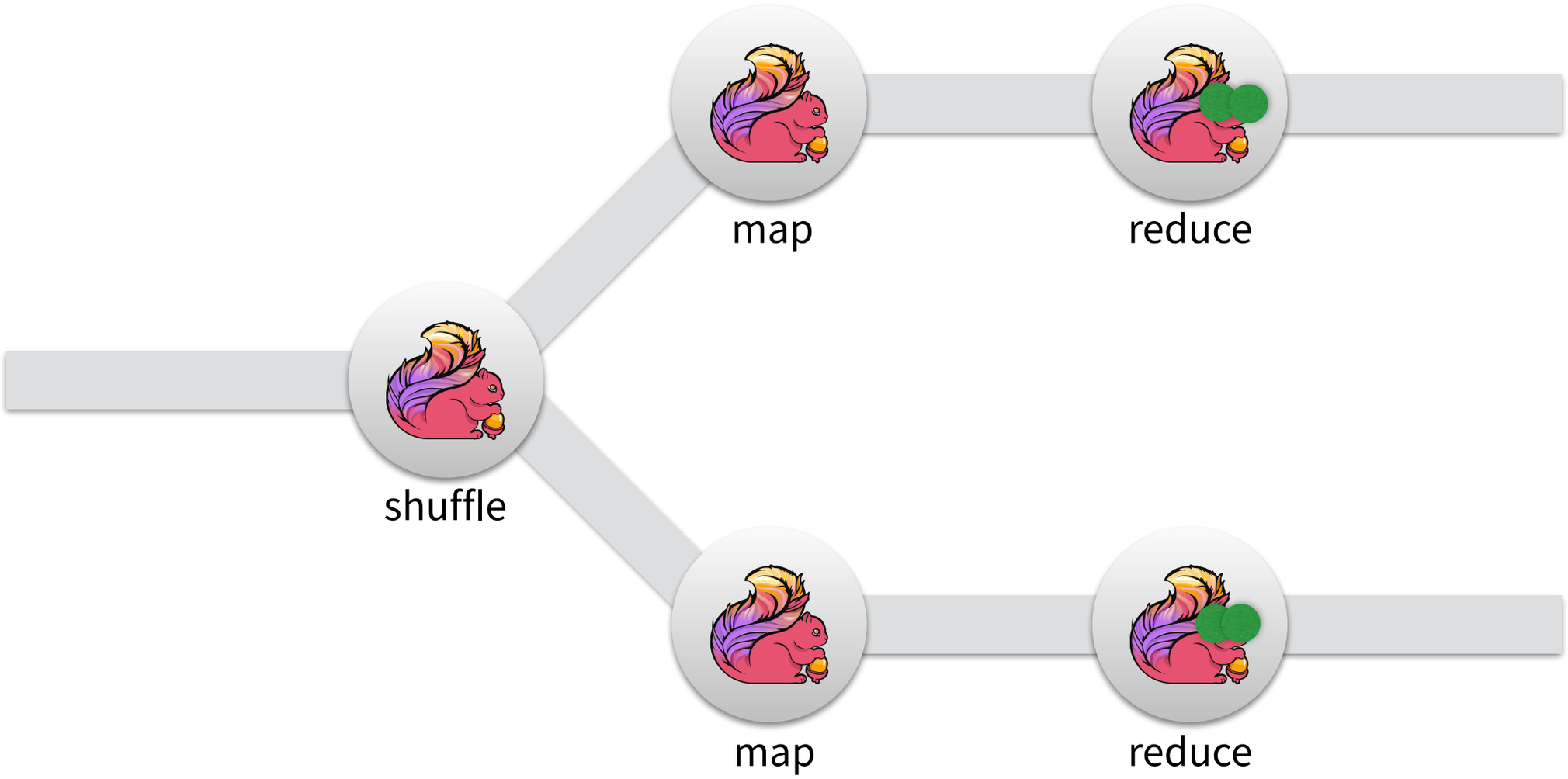
Streaming Dataflow Engine



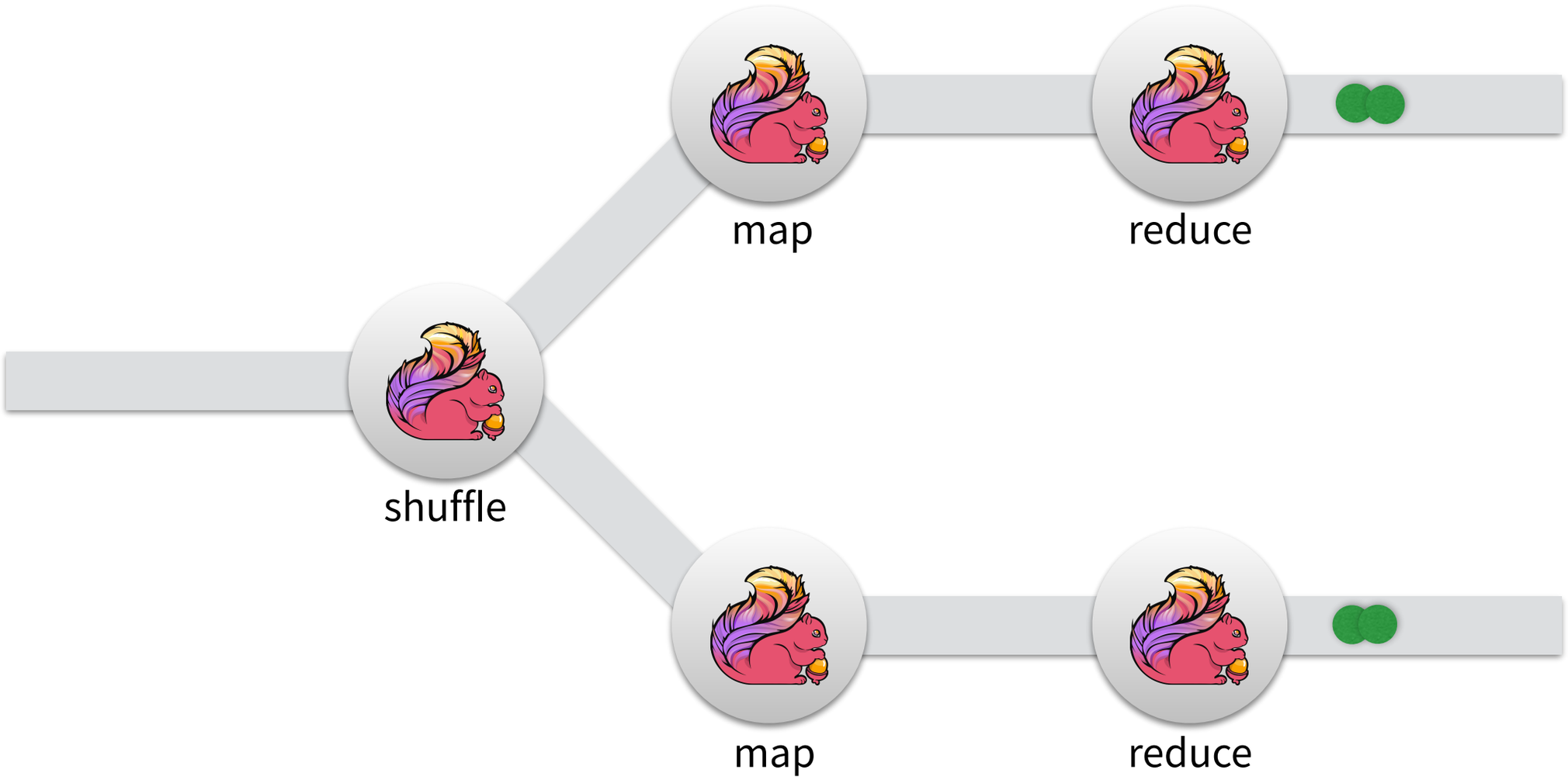
Streaming Dataflow Engine



Streaming Dataflow Engine



Streaming Dataflow Engine



Batch is a special case of streaming

스트림 데이터 = 끝 없이(unbounded) 주어지는 데이터

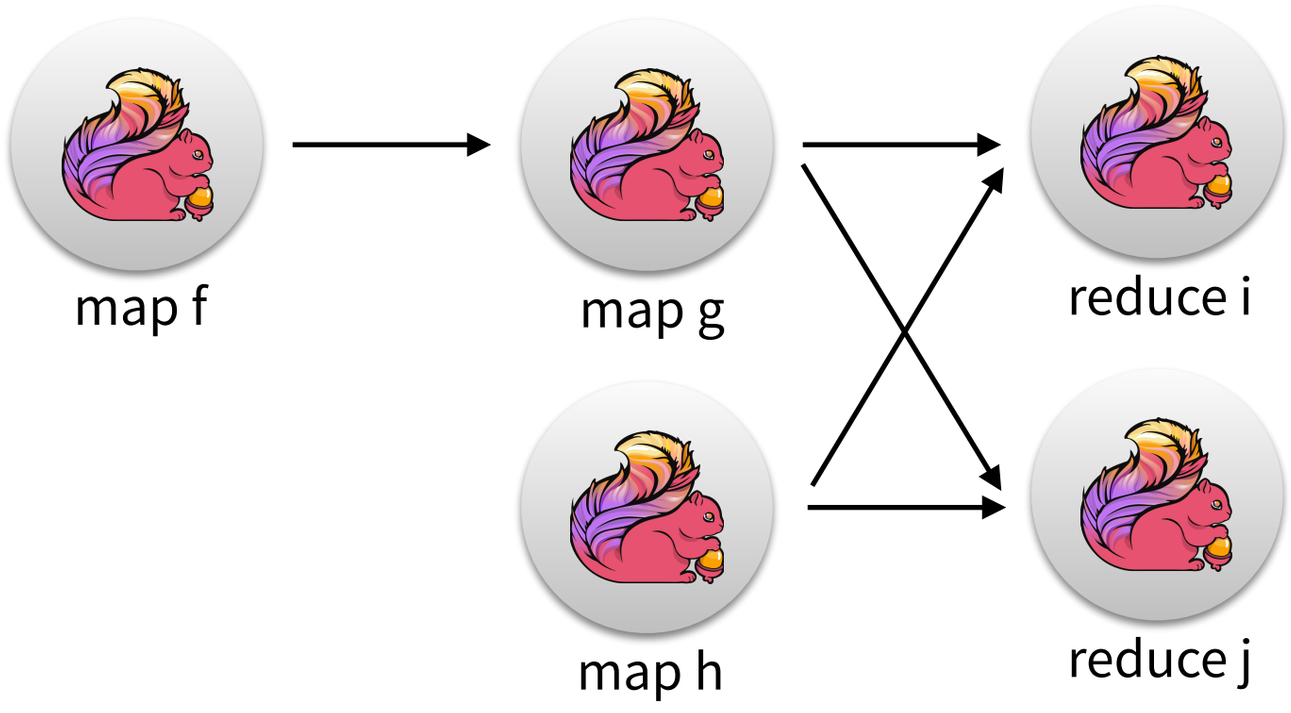
배치 데이터 = 끝이 있는(bounded) 데이터

Batch

Streaming

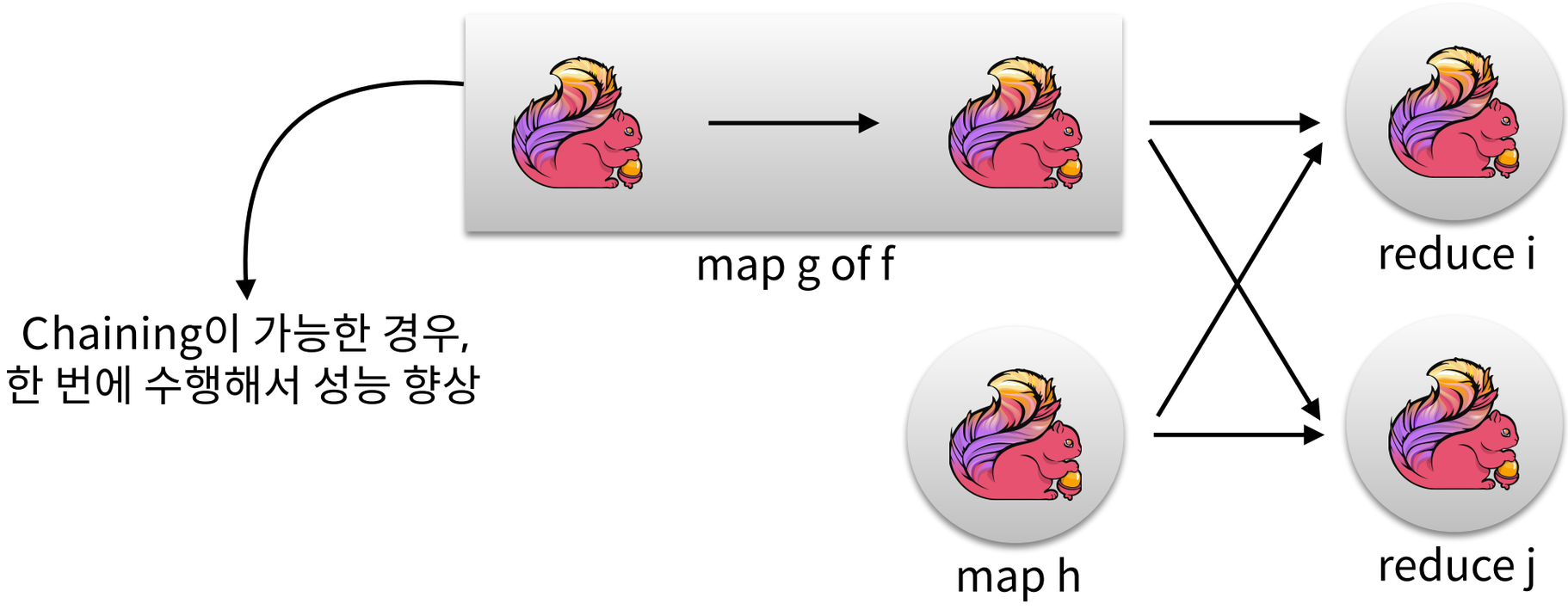
똑똑한 Flink 런타임 (1)

Pipelining



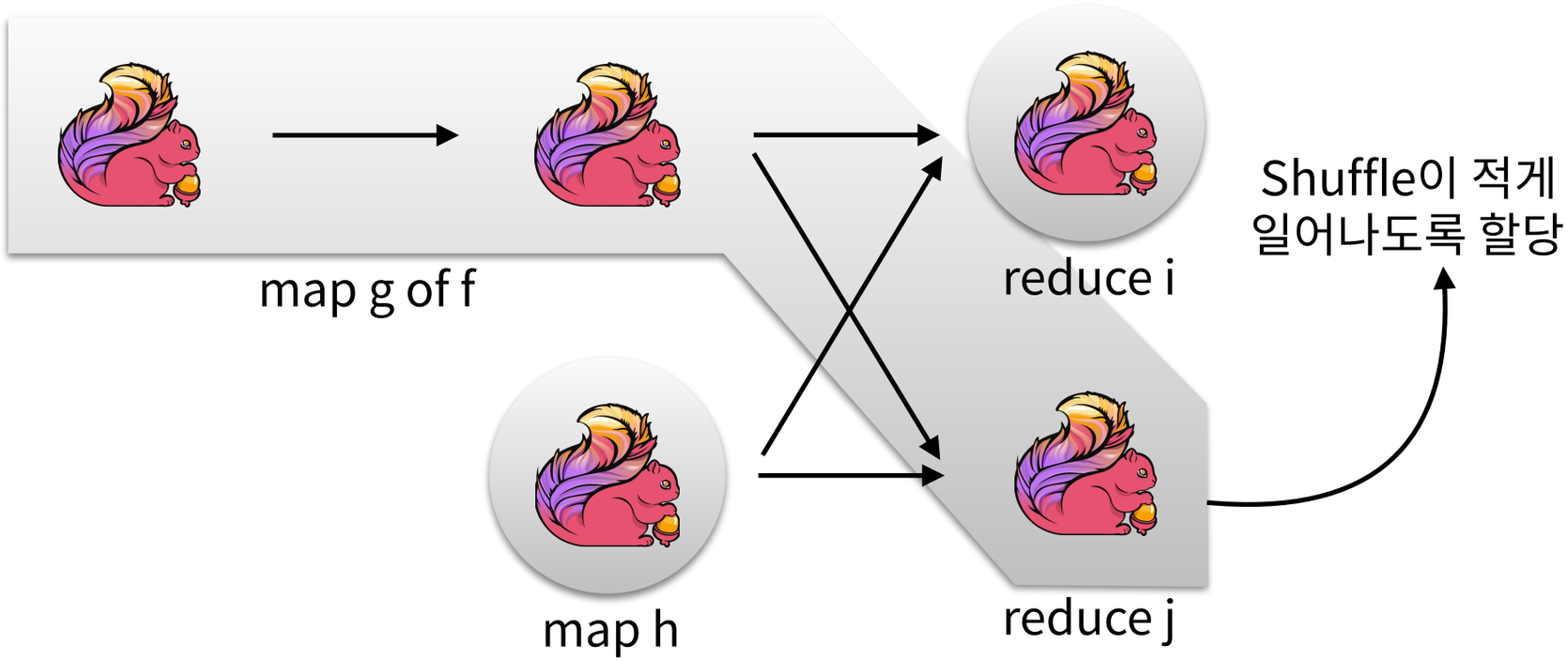
똑똑한 Flink 런타임 (1)

Pipelining



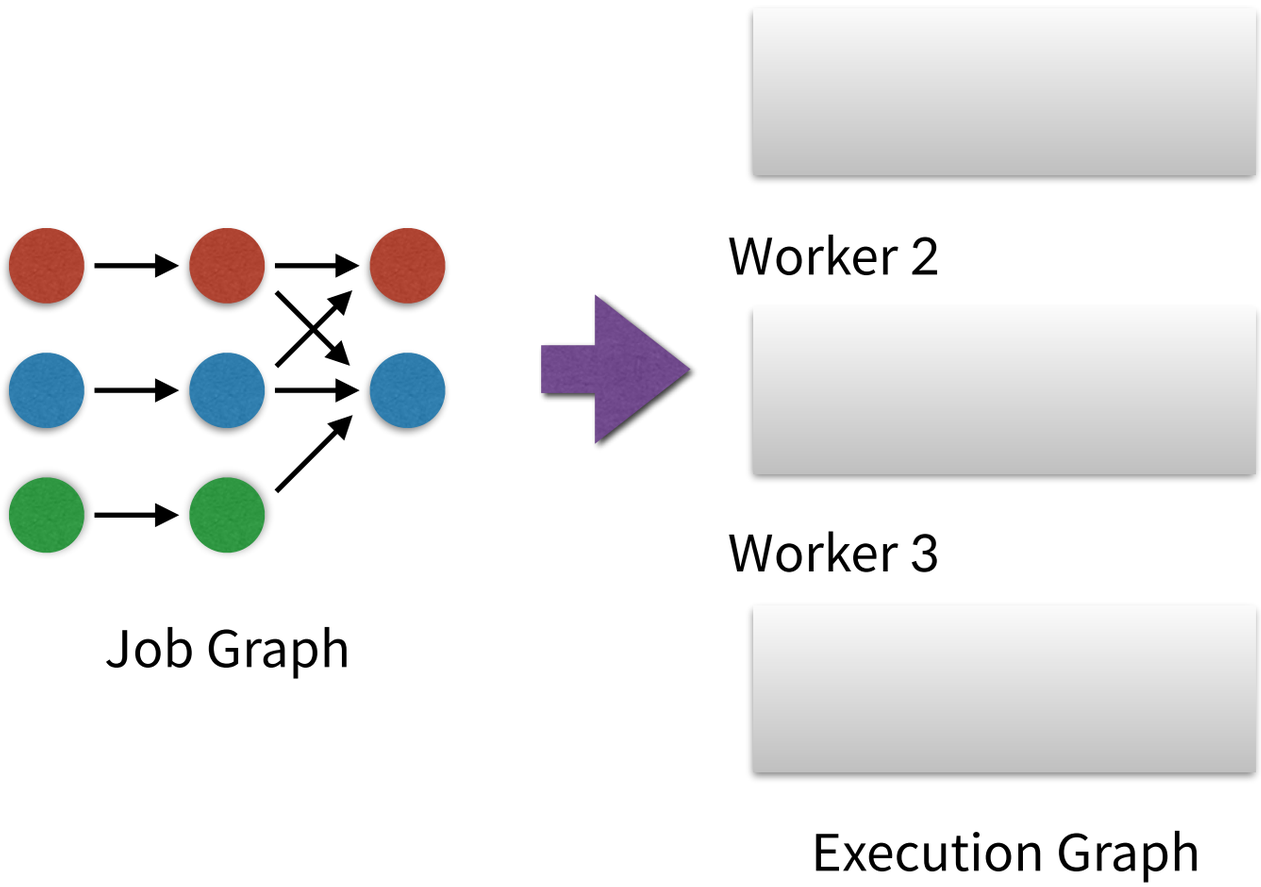
똑똑한 Flink 런타임 (1)

Pipelining



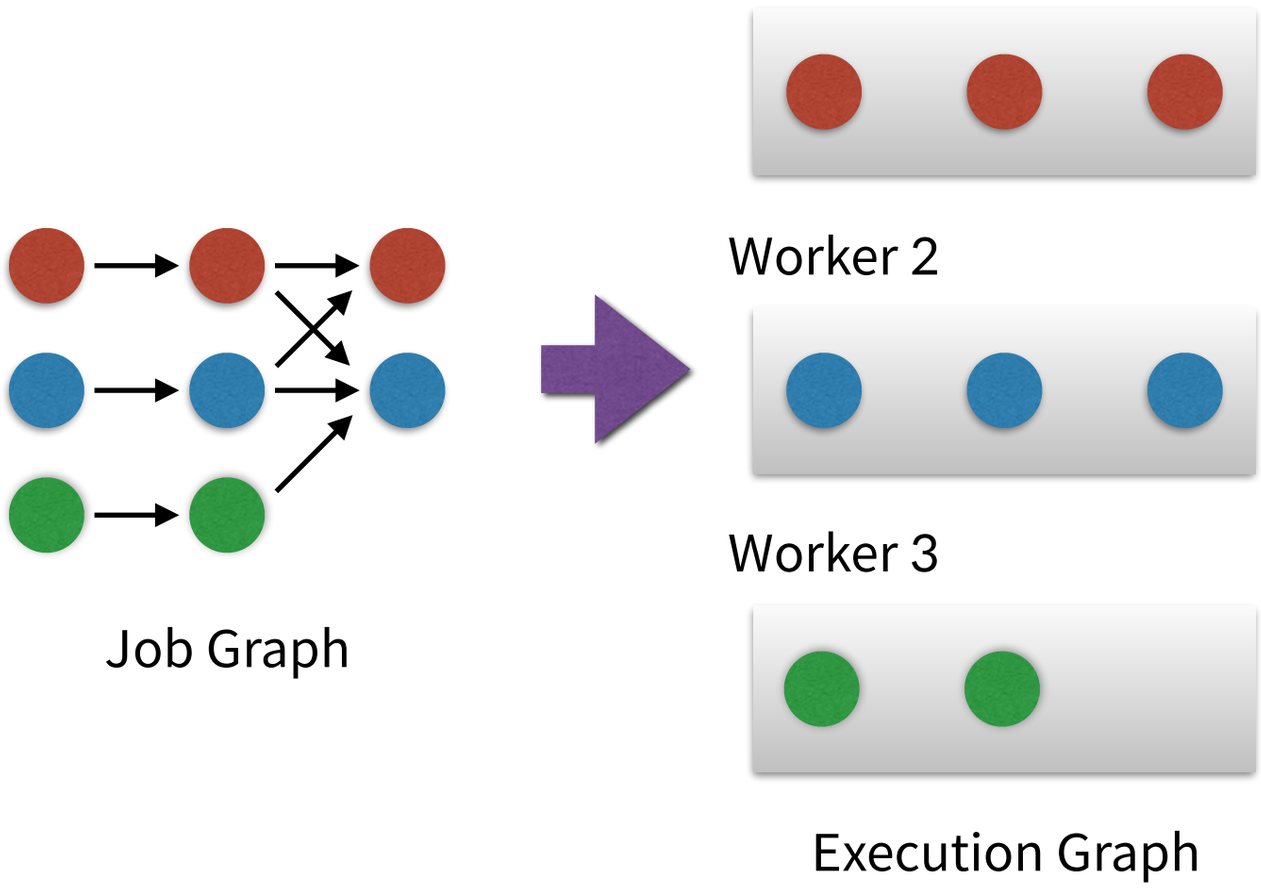
똑똑한 Flink 런타임 (1)

Pipelining



똑똑한 Flink 런타임 (1)

Pipelining

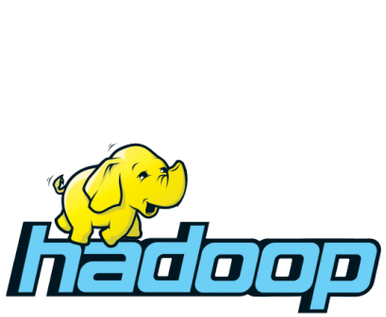


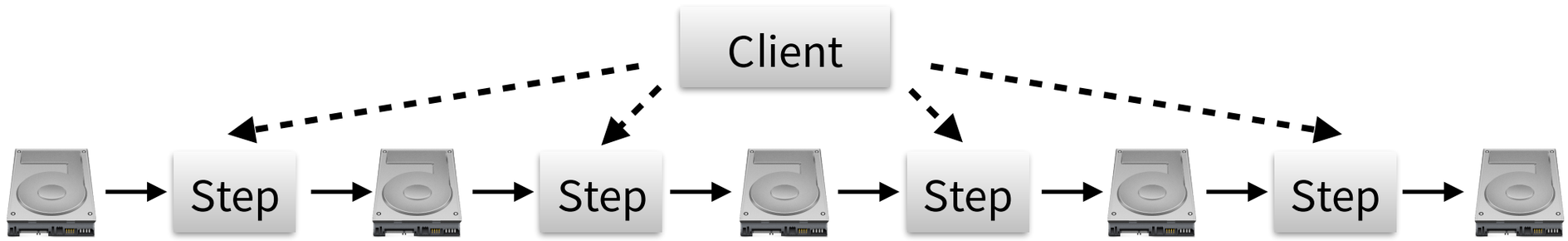
똑똑한 Flink 런타임 (2)

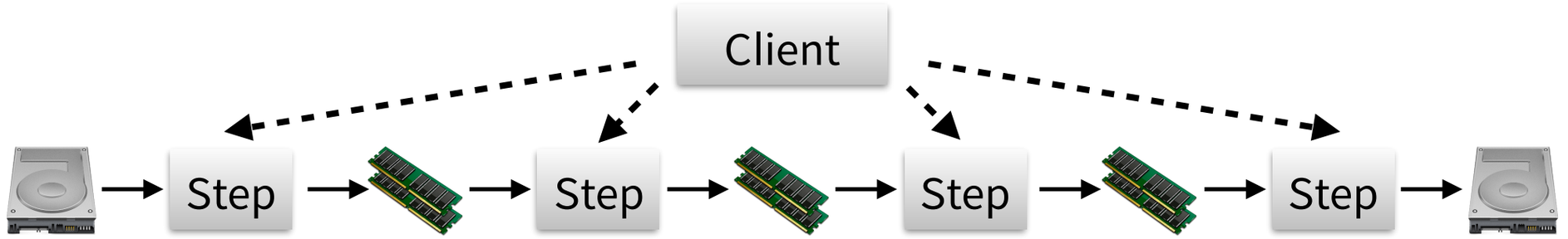
Native Iteration

똑똑한 Flink 런타임 (2)

Native Iteration

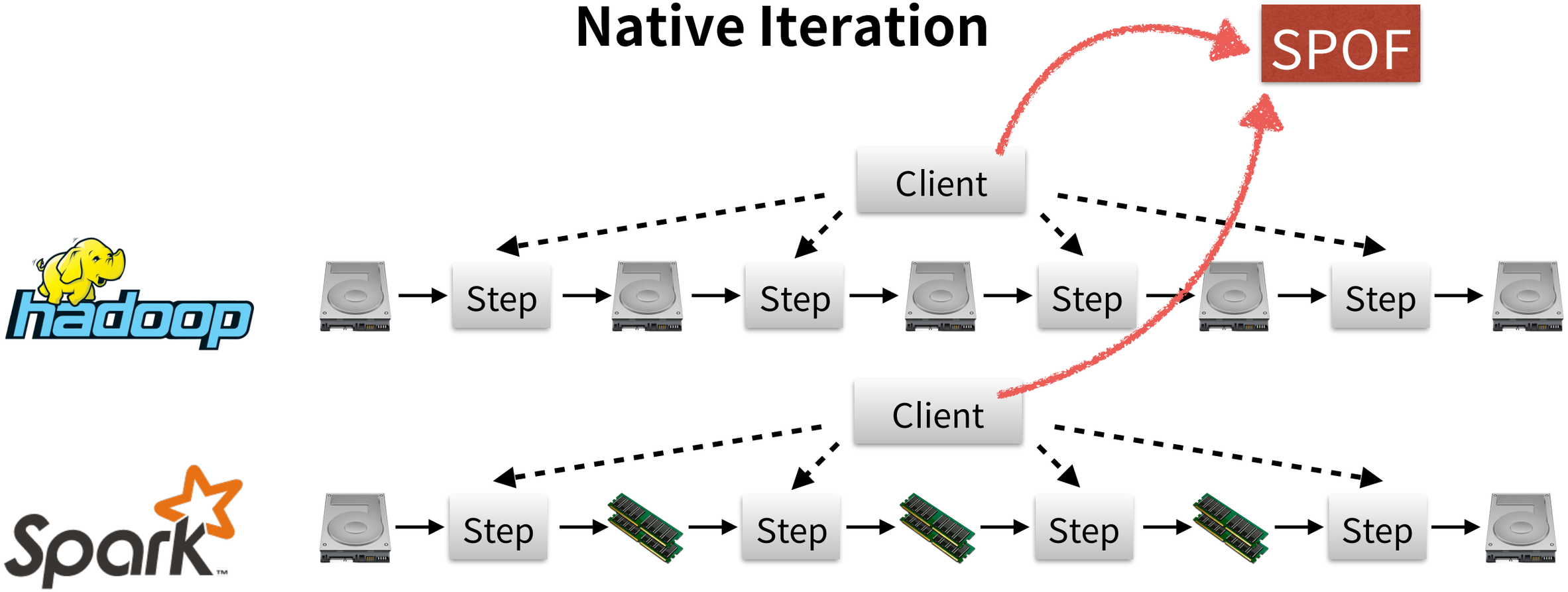






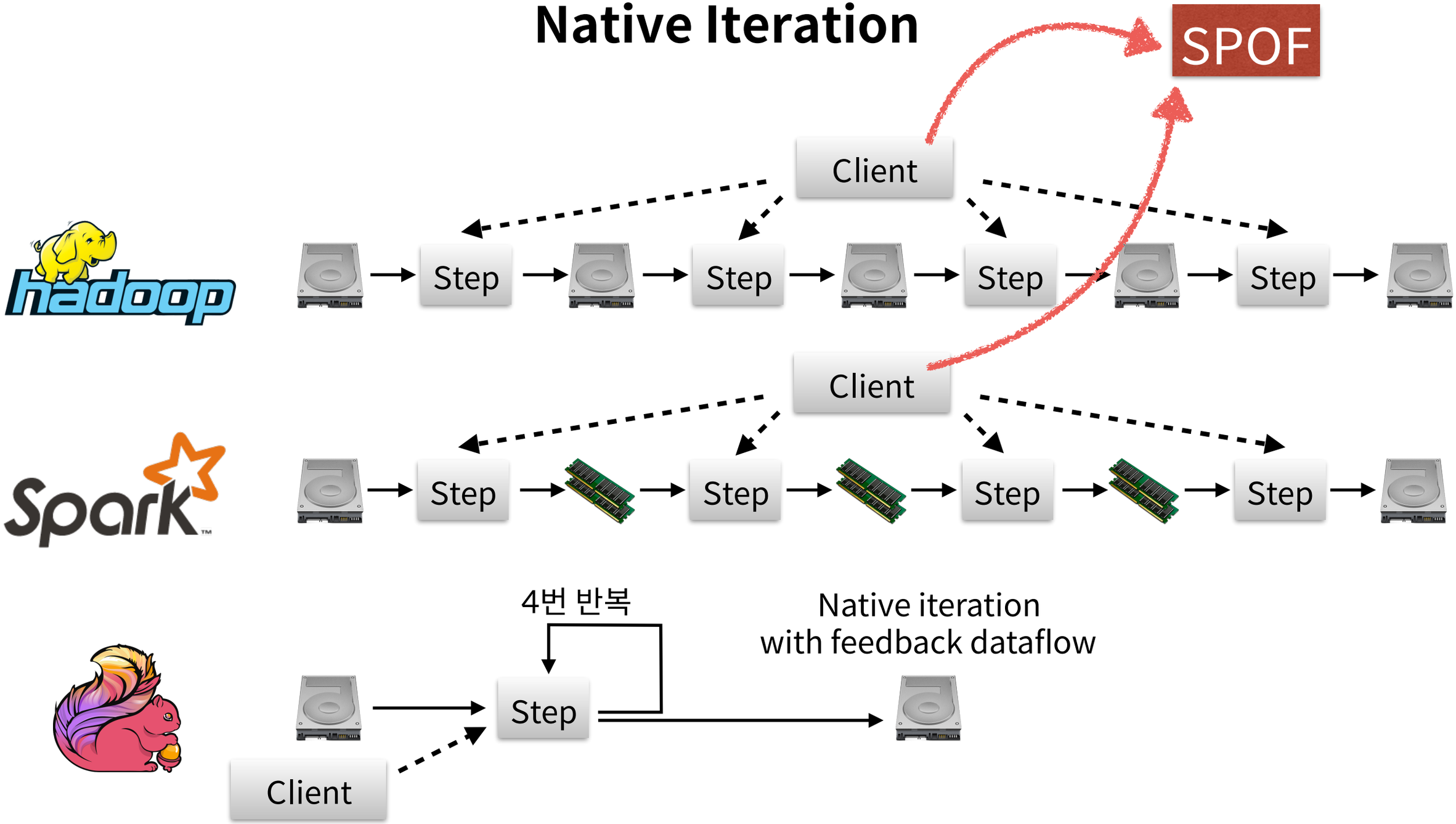
똑똑한 Flink 런타임 (2)

Native Iteration



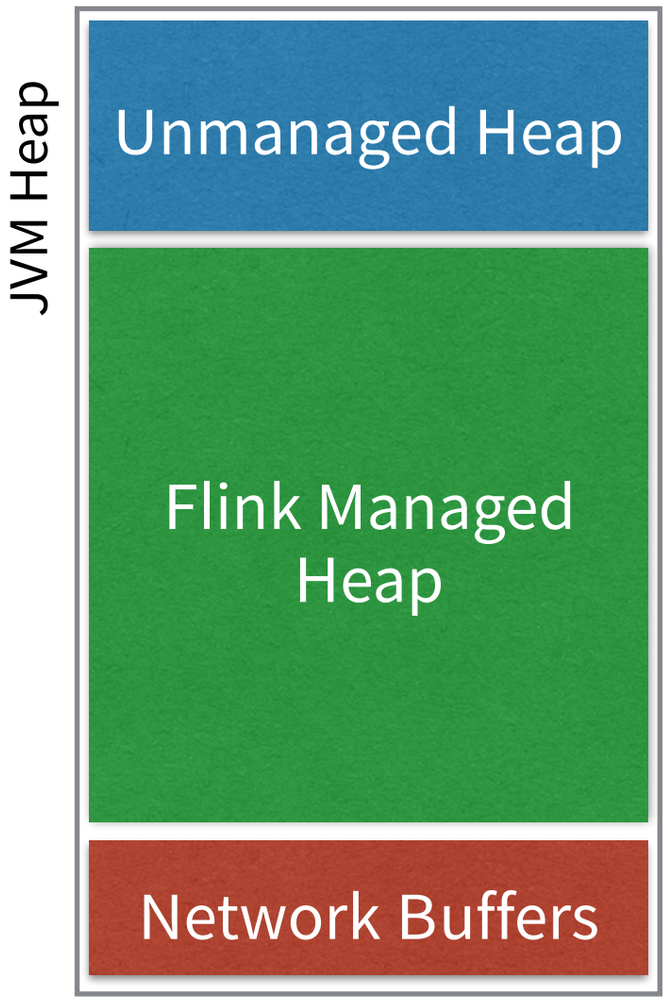
똑똑한 Flink 런타임 (2)

Native Iteration



똑똑한 Flink 런타임 (3)

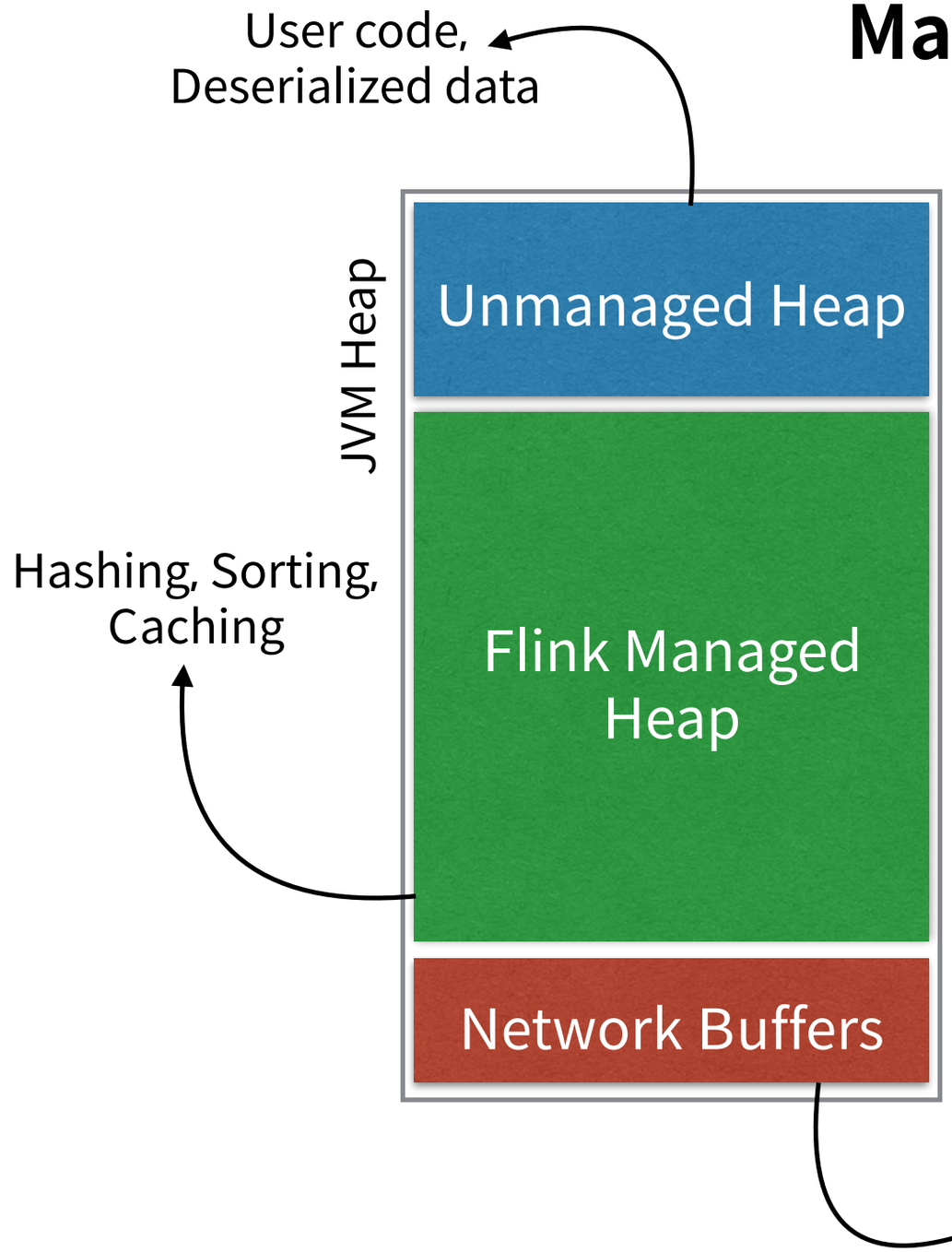
Managed Memory



- Flink는 메모리를 직접 관리함
  - GC로 인한 성능 감소가 적음
- Serde 오버헤드를 줄이기 위해 바이너리 데이터에서 바로 연산을 수행
  - 이를 위해, 타입 정보를 미리 조사
- 덕분에 메모리 튜닝이 거의 필요 없음
- Spark는 버전 1.4부터 지원

똑똑한 Flink 런타임 (3)

Managed Memory



- Flink는 메모리를 직접 관리함
  - GC로 인한 성능 감소가 적음
- Serde 오버헤드를 줄이기 위해 바이너리 데이터에서 바로 연산을 수행
  - 이를 위해, 타입 정보를 미리 조사
- 덕분에 메모리 튜닝이 거의 필요 없음
- Spark는 버전 1.4부터 지원

Features for Real-time Streaming

빠른 결과를 위한 **Low latency**

많은 이벤트를 처리할 수 있도록 **High throughput**

정확한 결과를 위한 **Exactly-once guarantees**

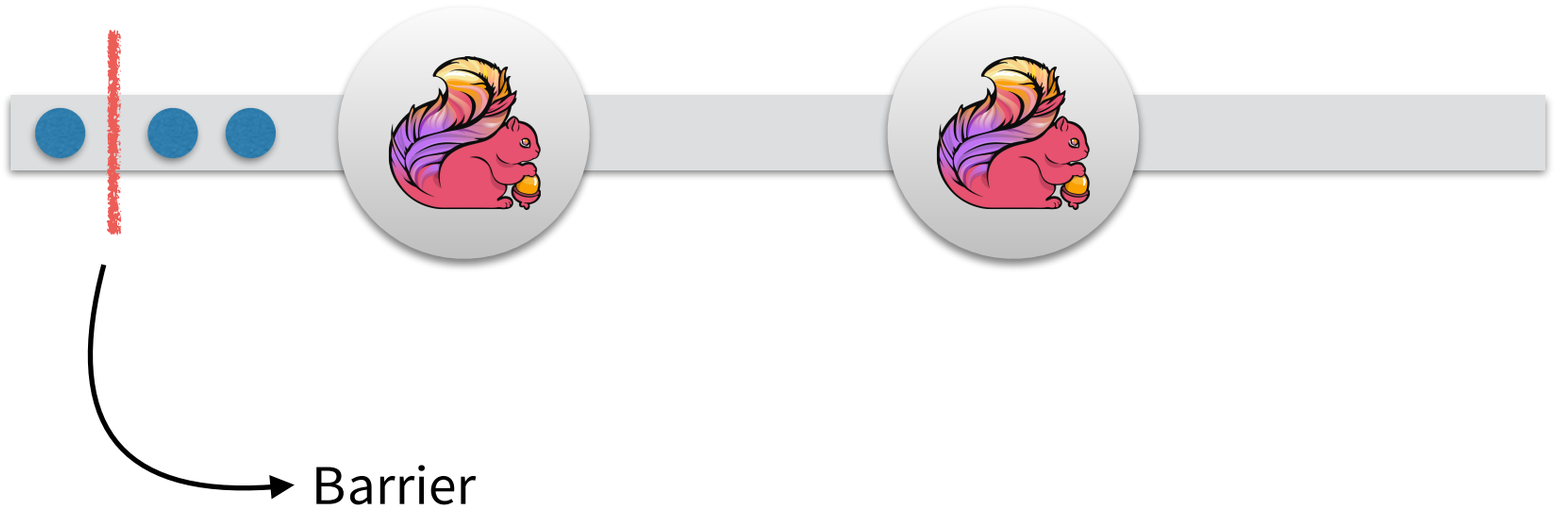
이벤트 순서에 자유로운 처리를 위한 **Event-time windows**

Exactly-once Guarantees by distributed snapshots

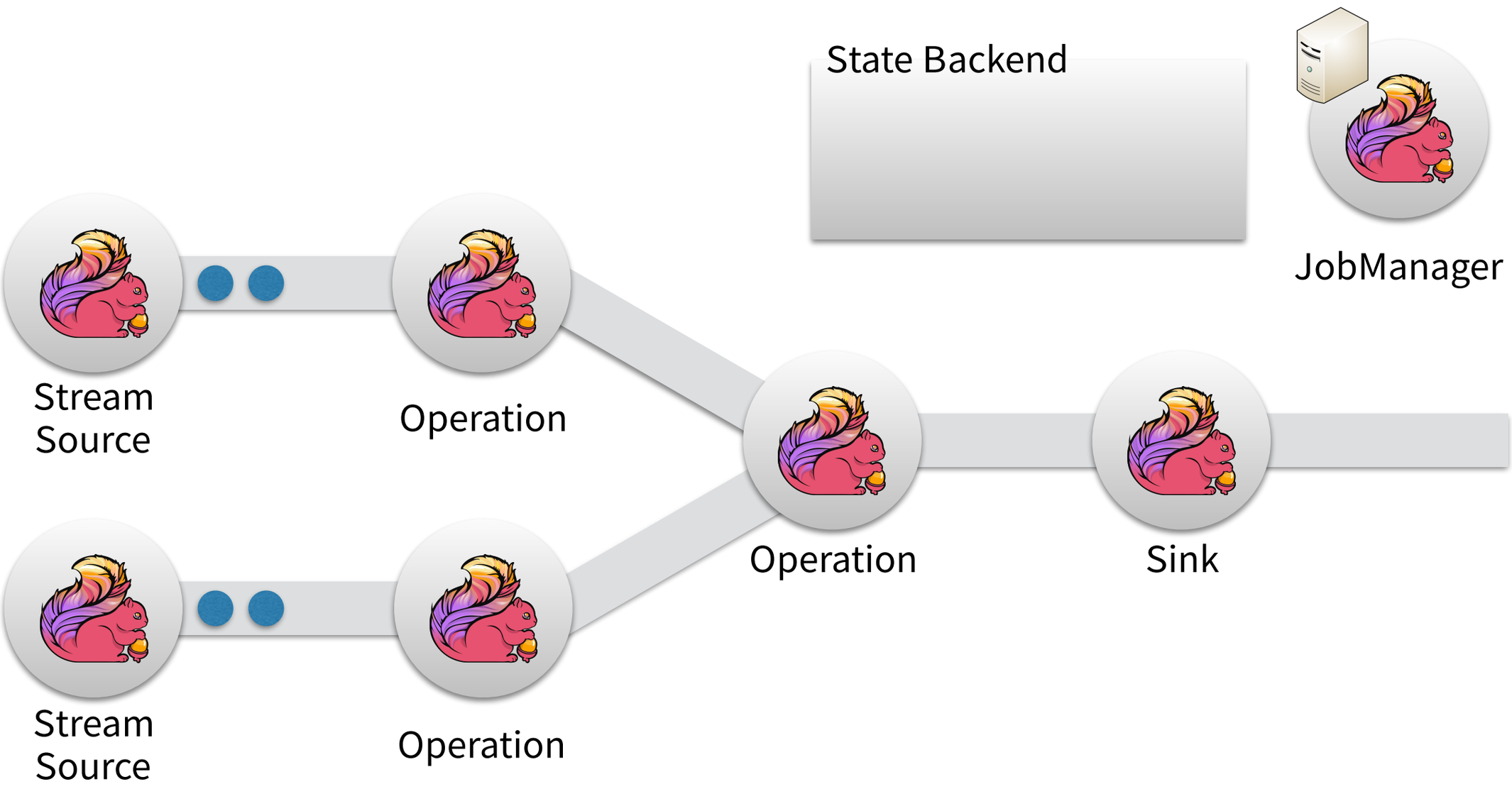
모든 스트림 데이터를 정확하게 1번만 처리하는 것을 보장

Exactly-once Guarantees by distributed snapshots

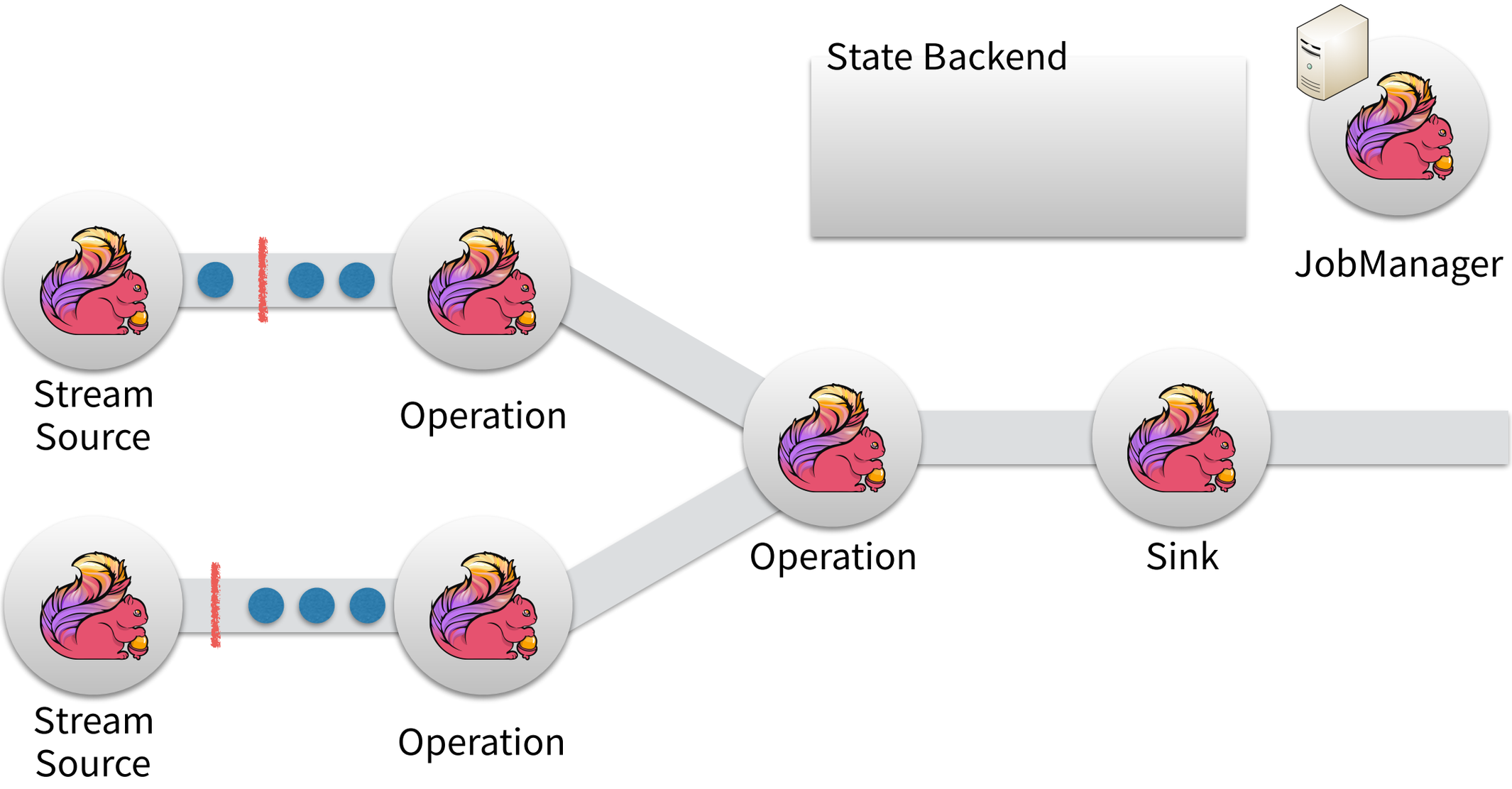
모든 스트림 데이터를 정확하게 1번만 처리하는 것을 보장



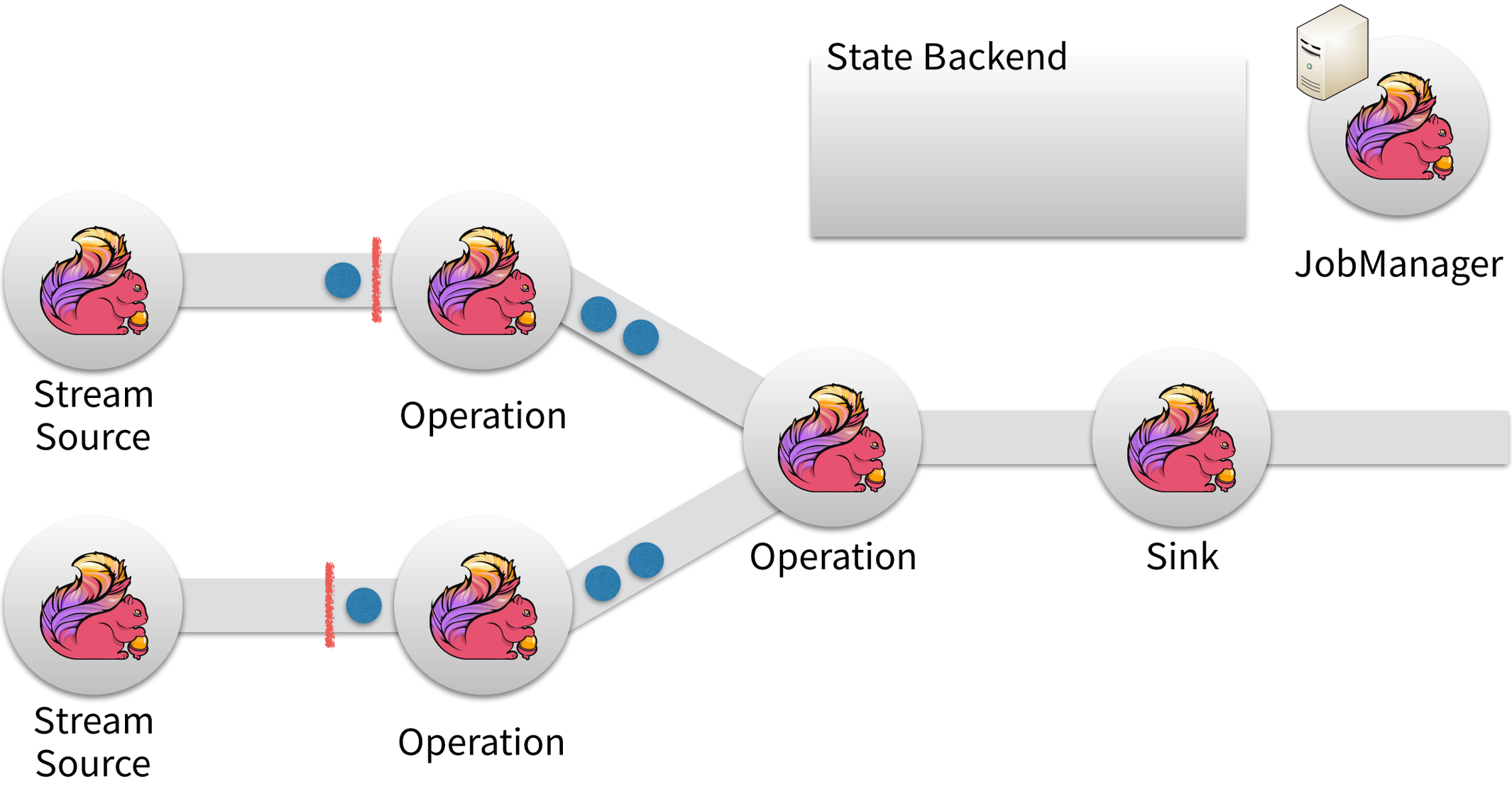
Distributed Snapshots



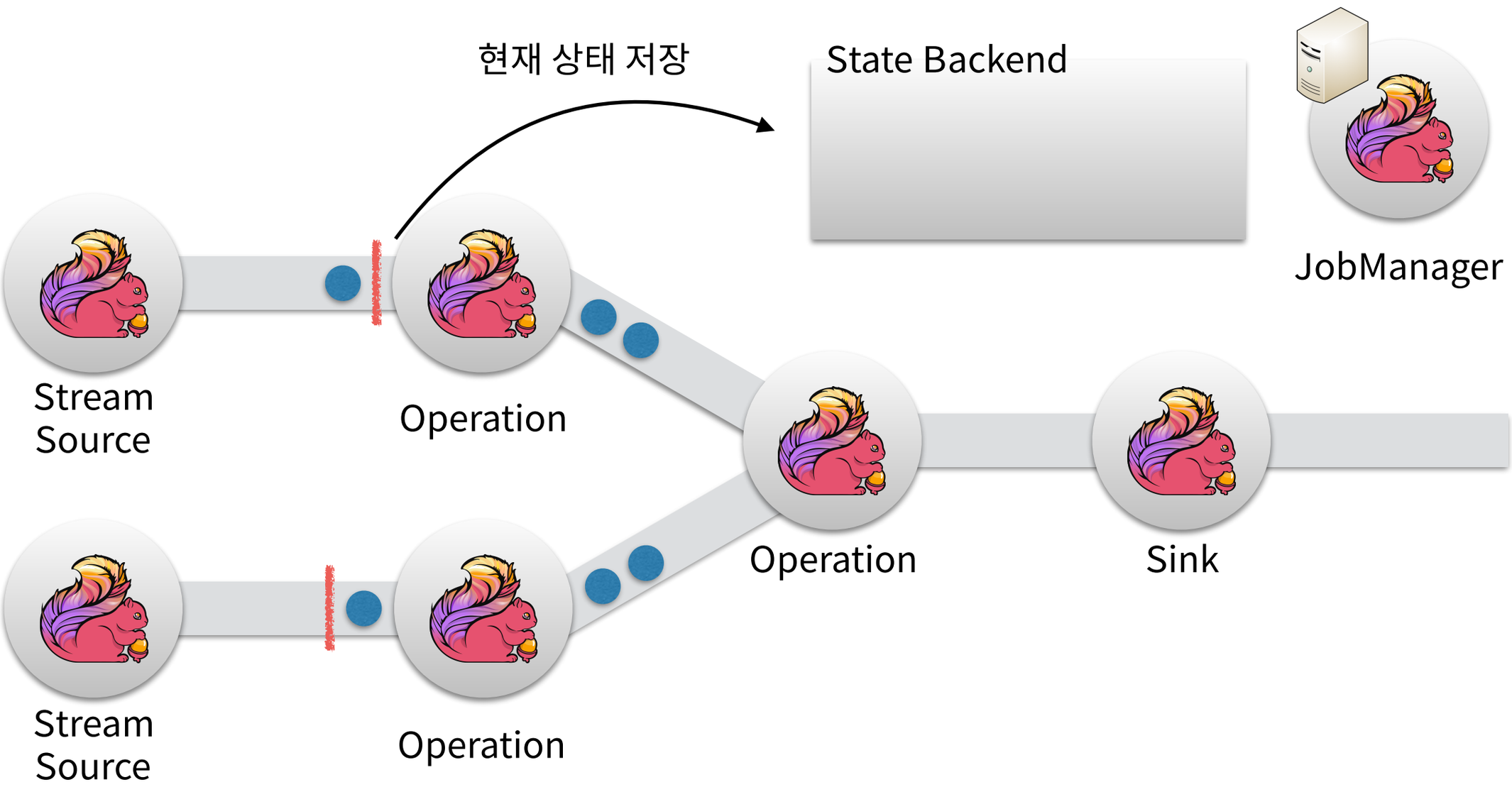
Distributed Snapshots



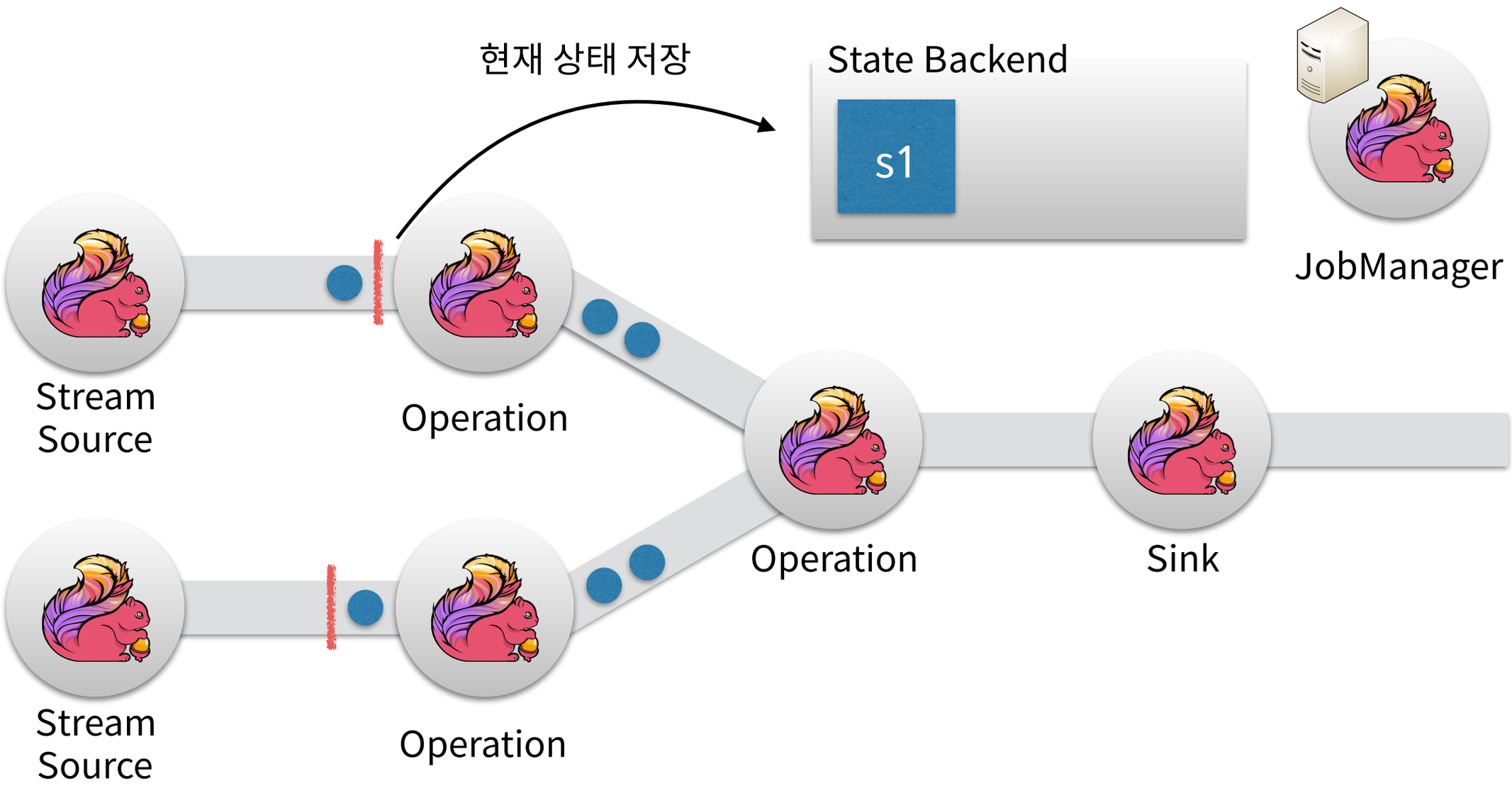
Distributed Snapshots



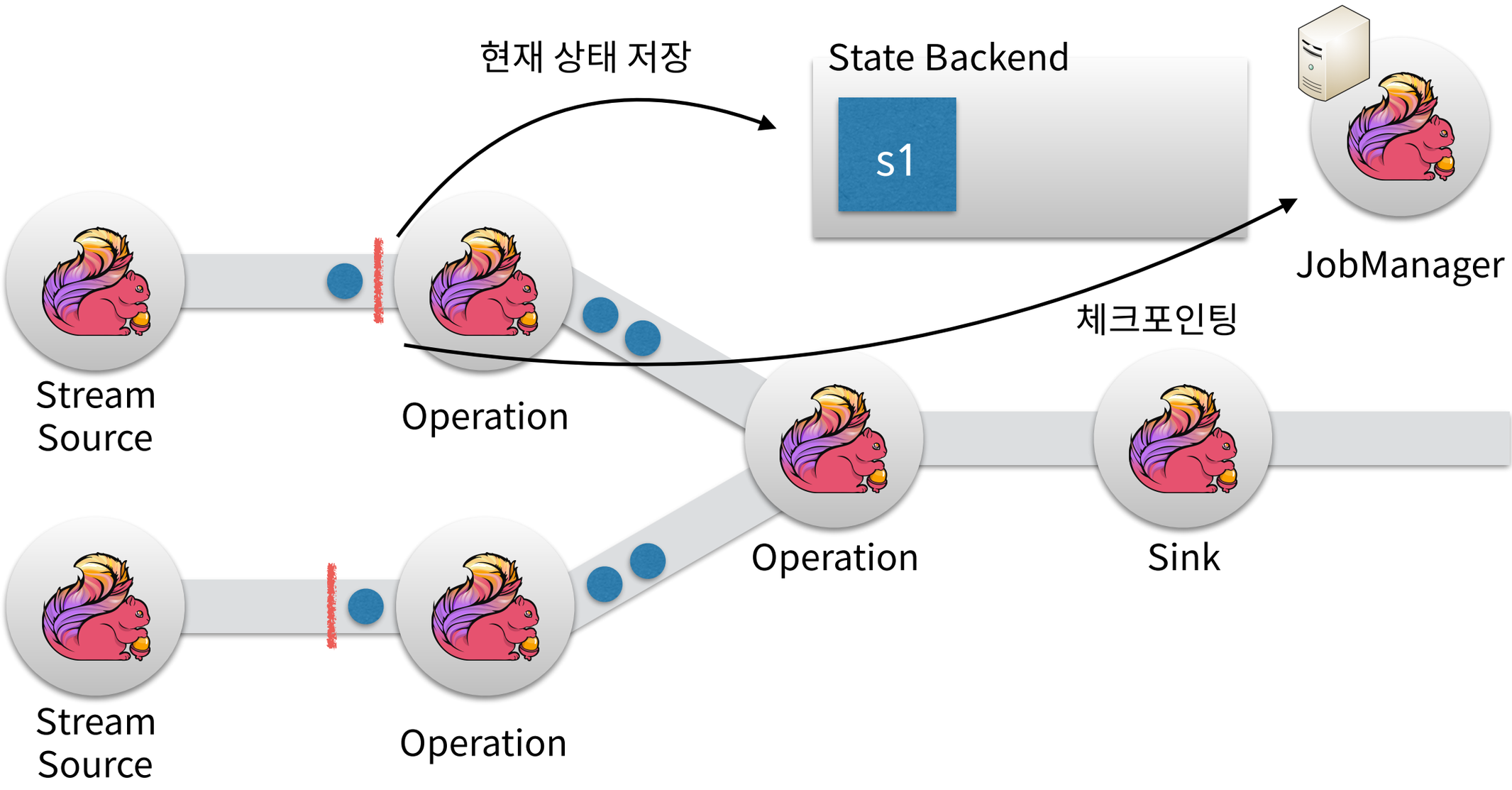
Distributed Snapshots



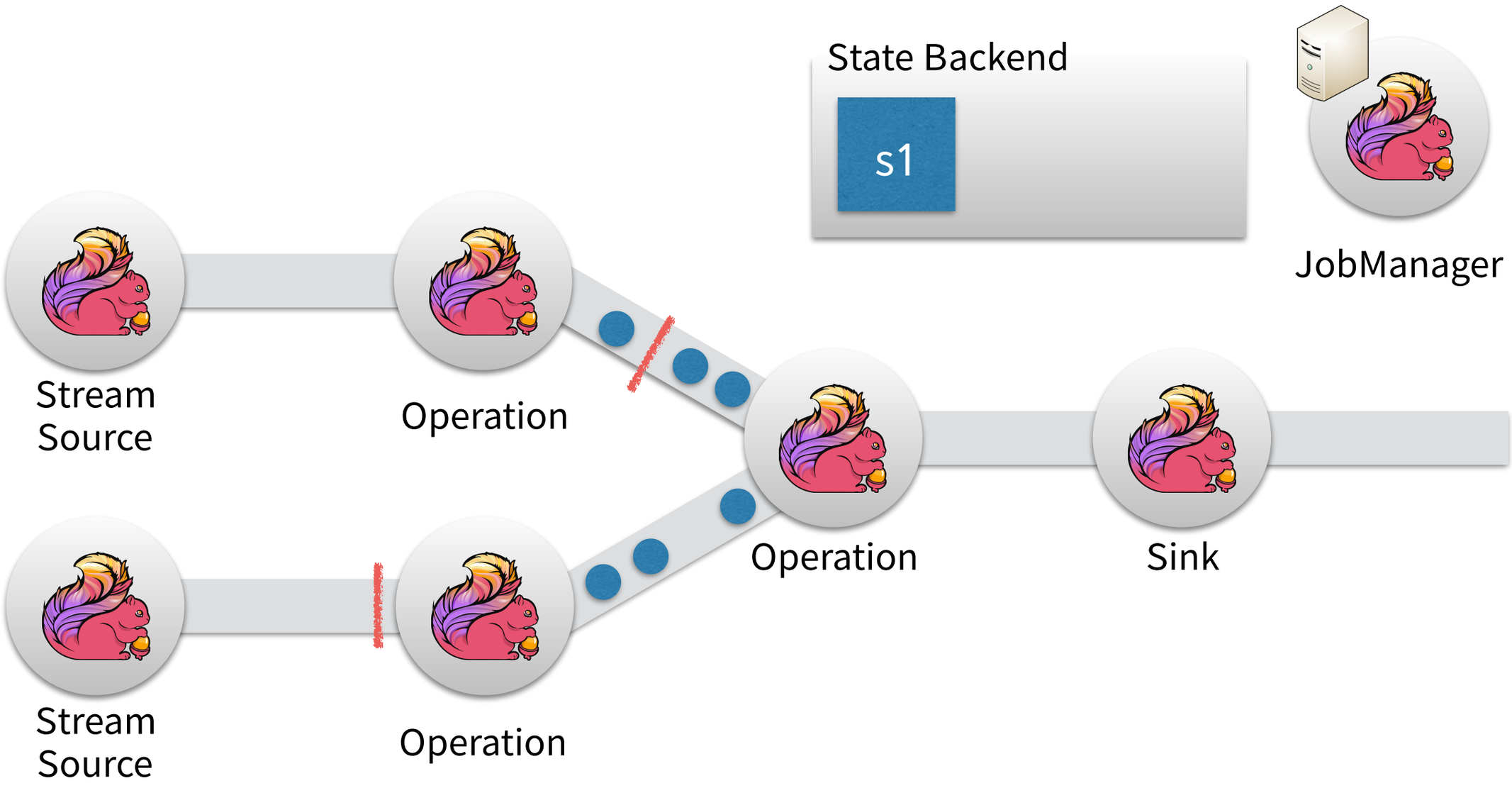
Distributed Snapshots



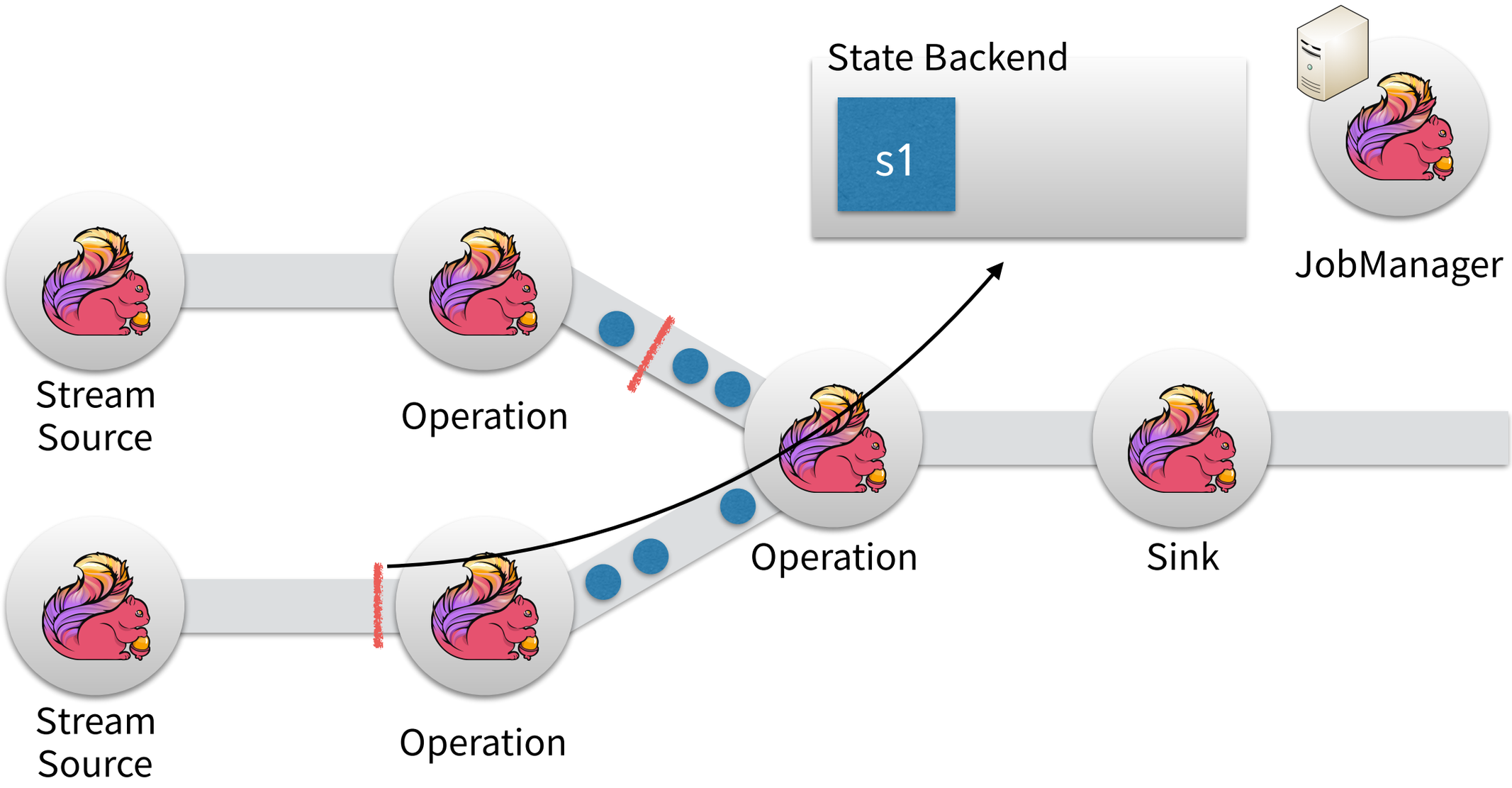
Distributed Snapshots



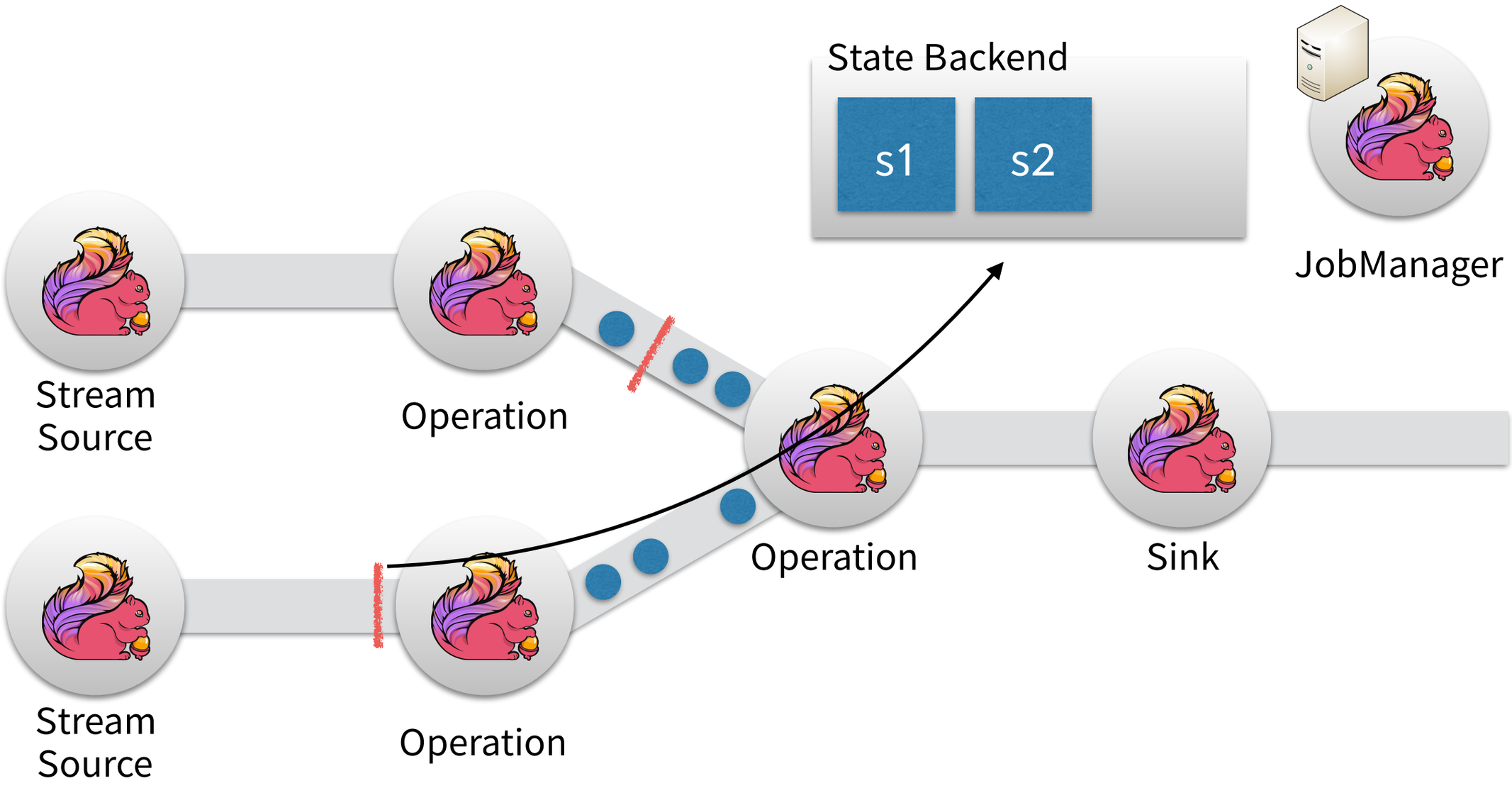
Distributed Snapshots



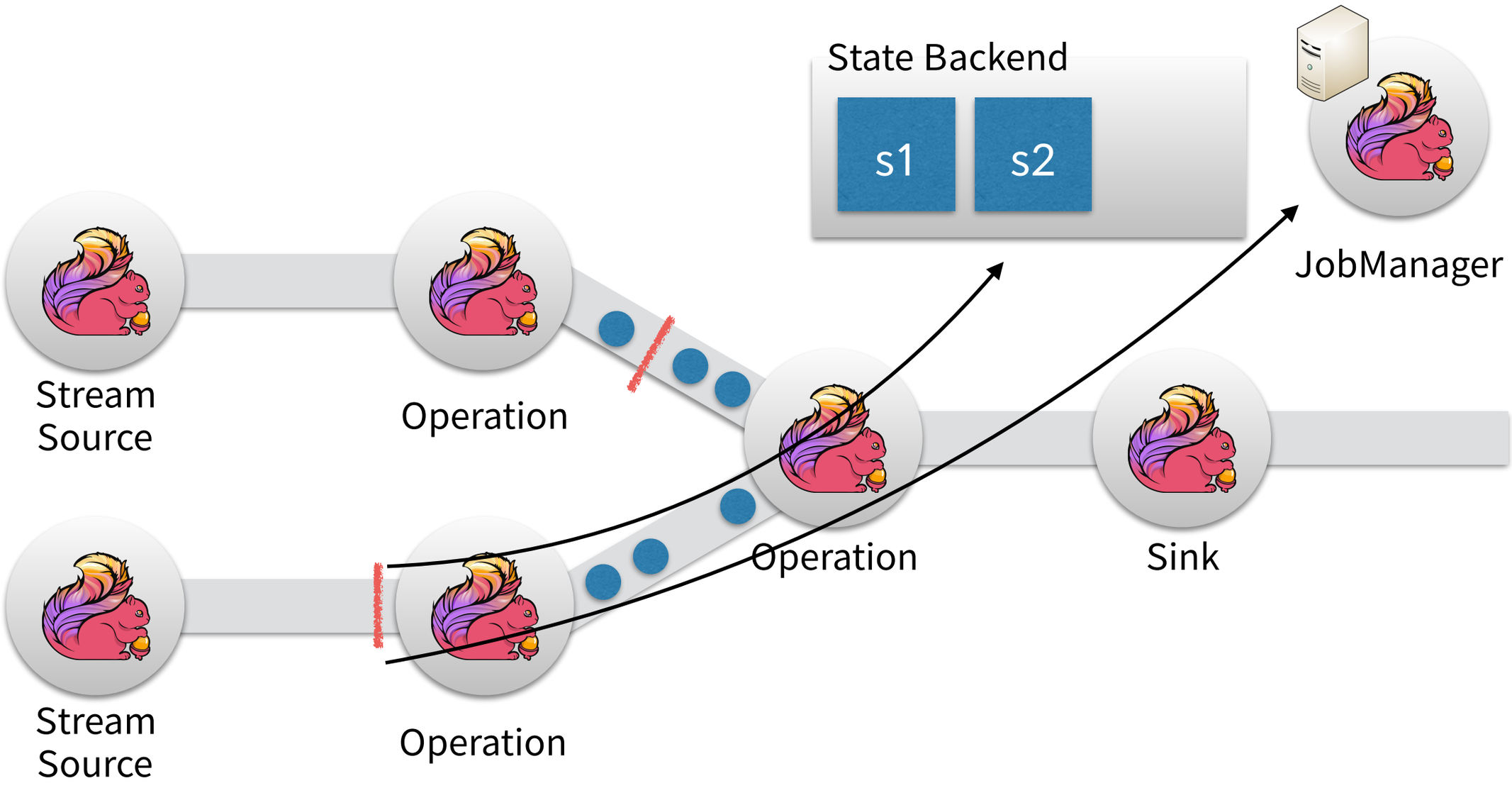
Distributed Snapshots



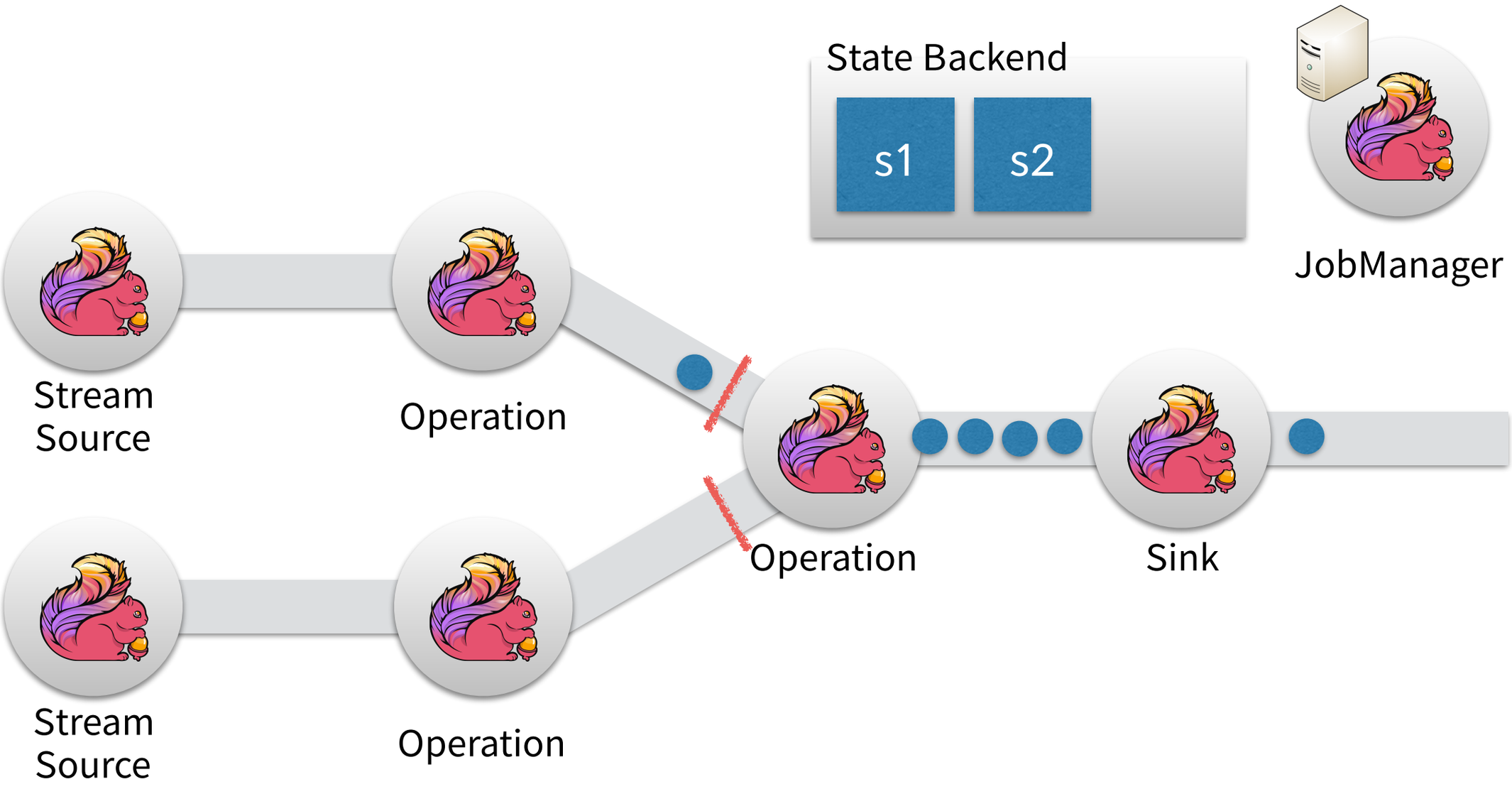
Distributed Snapshots



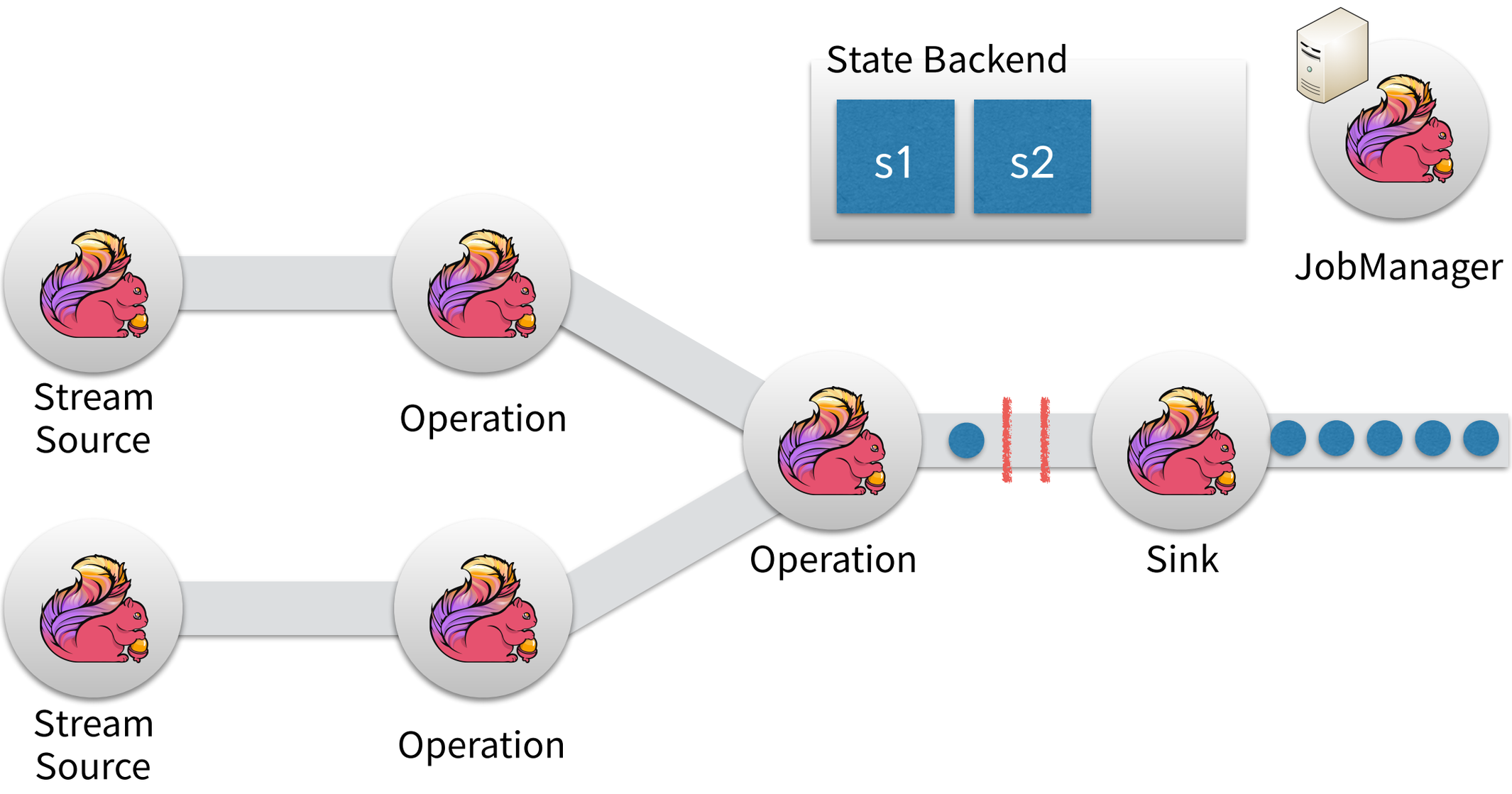
Distributed Snapshots



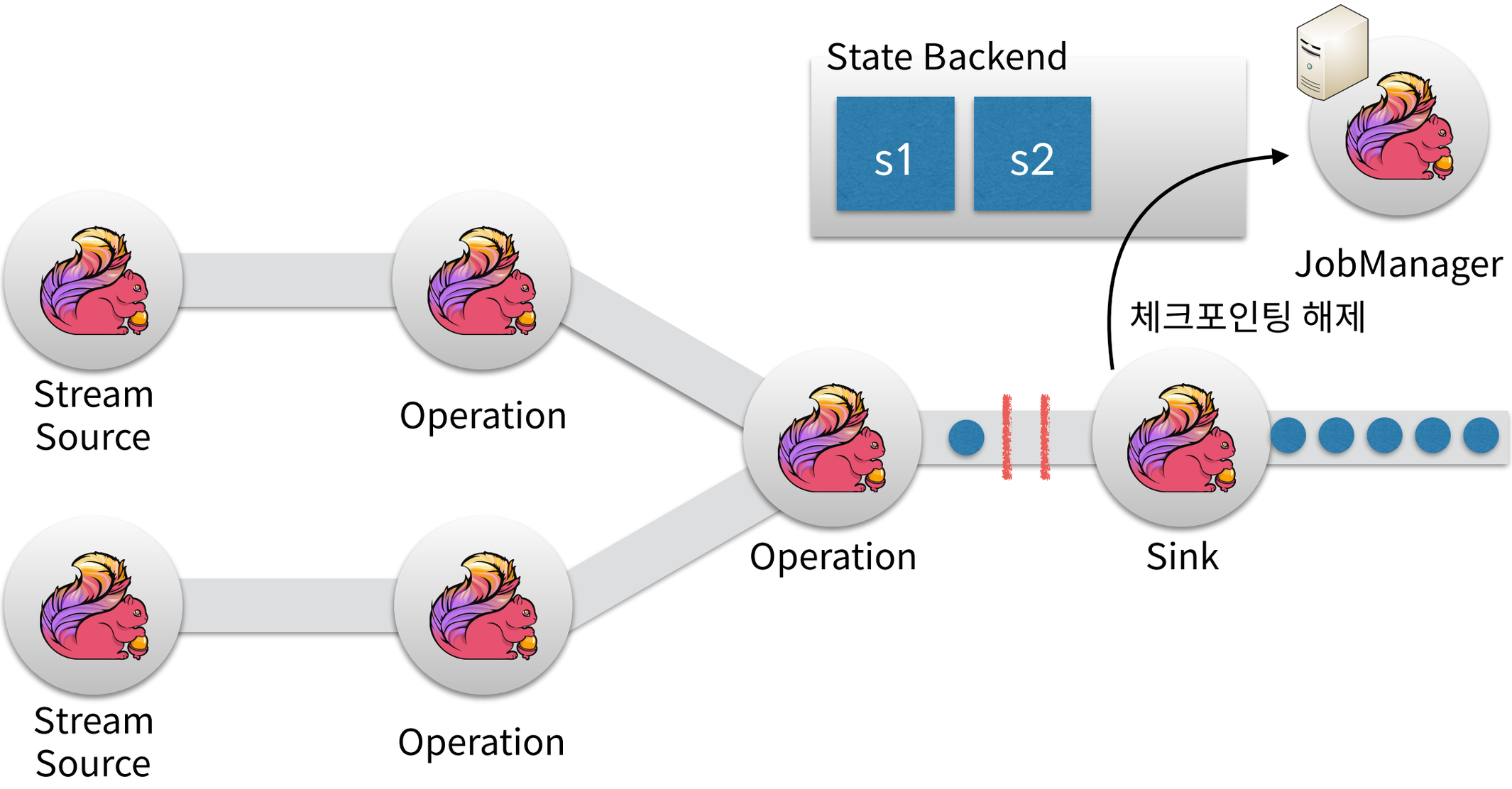
Distributed Snapshots



Distributed Snapshots



Distributed Snapshots



Event-time Windows

실제 스트림 처리 시스템에서는 다양한 이유로
이벤트의 생성 시간과 처리 시간이 다릅니다.

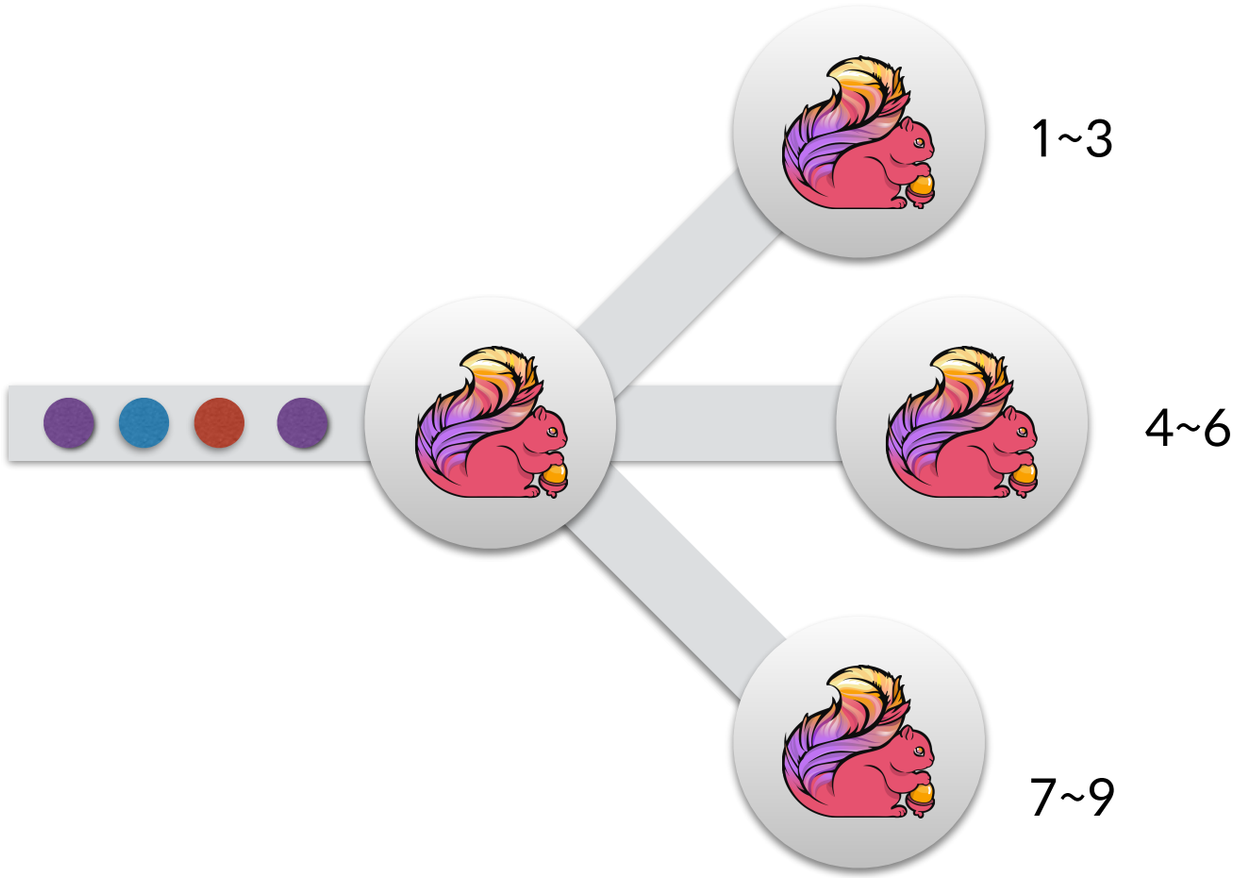
네트워크로 인한 지연 효과

스트림 처리 과정에서 Backpressure로 인한 지연 효과

이벤트 소스가 항상 연결되어 있지 않는 경우 (모바일 등)

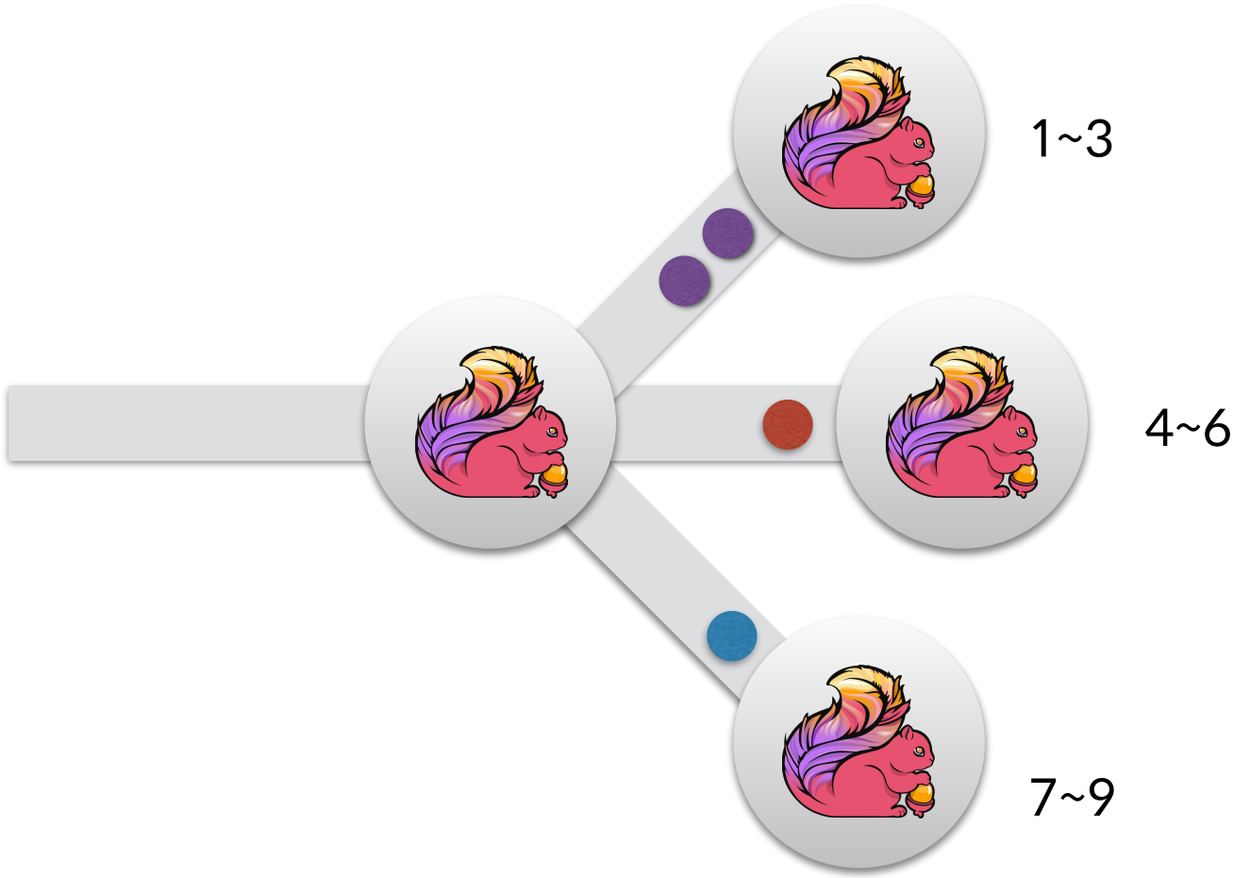
Event-time Windows

Flink는 클러스터의 수행 시간이 아닌 실제 이벤트의
발생 시간을 기준으로 스트림 처리를 수행할 수 있습니다.

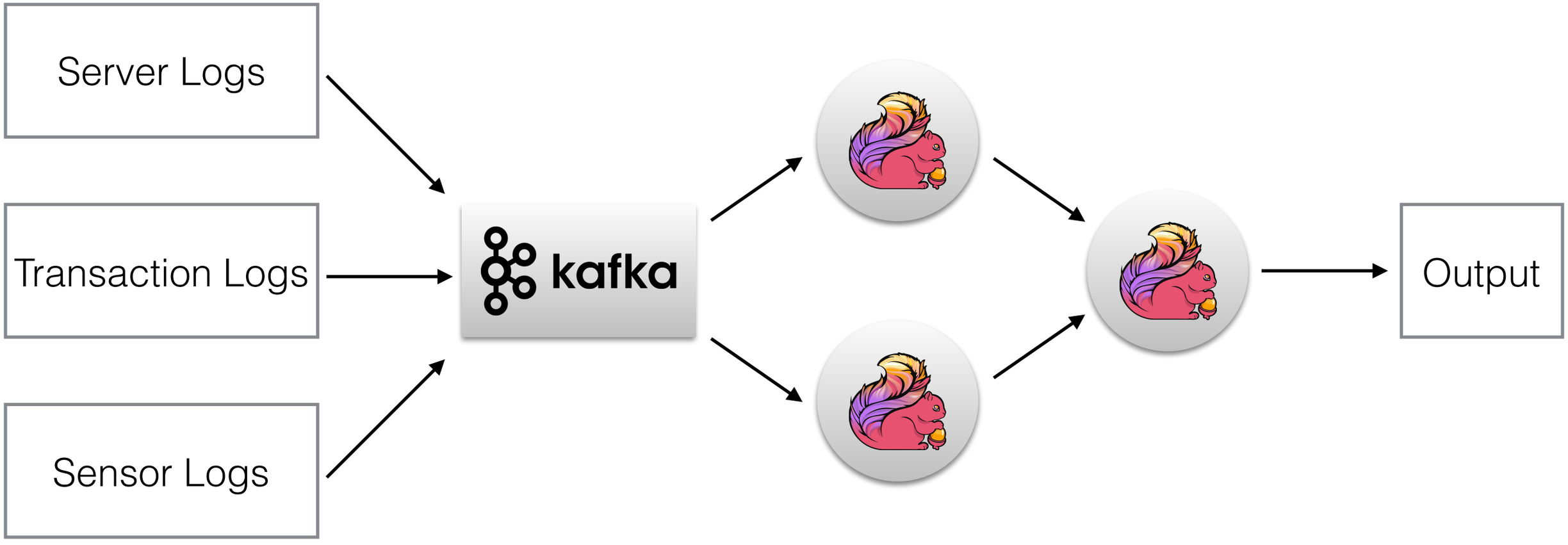


Event-time Windows

Flink는 클러스터의 수행 시간이 아닌 실제 이벤트의
발생 시간을 기준으로 스트림 처리를 수행할 수 있습니다.



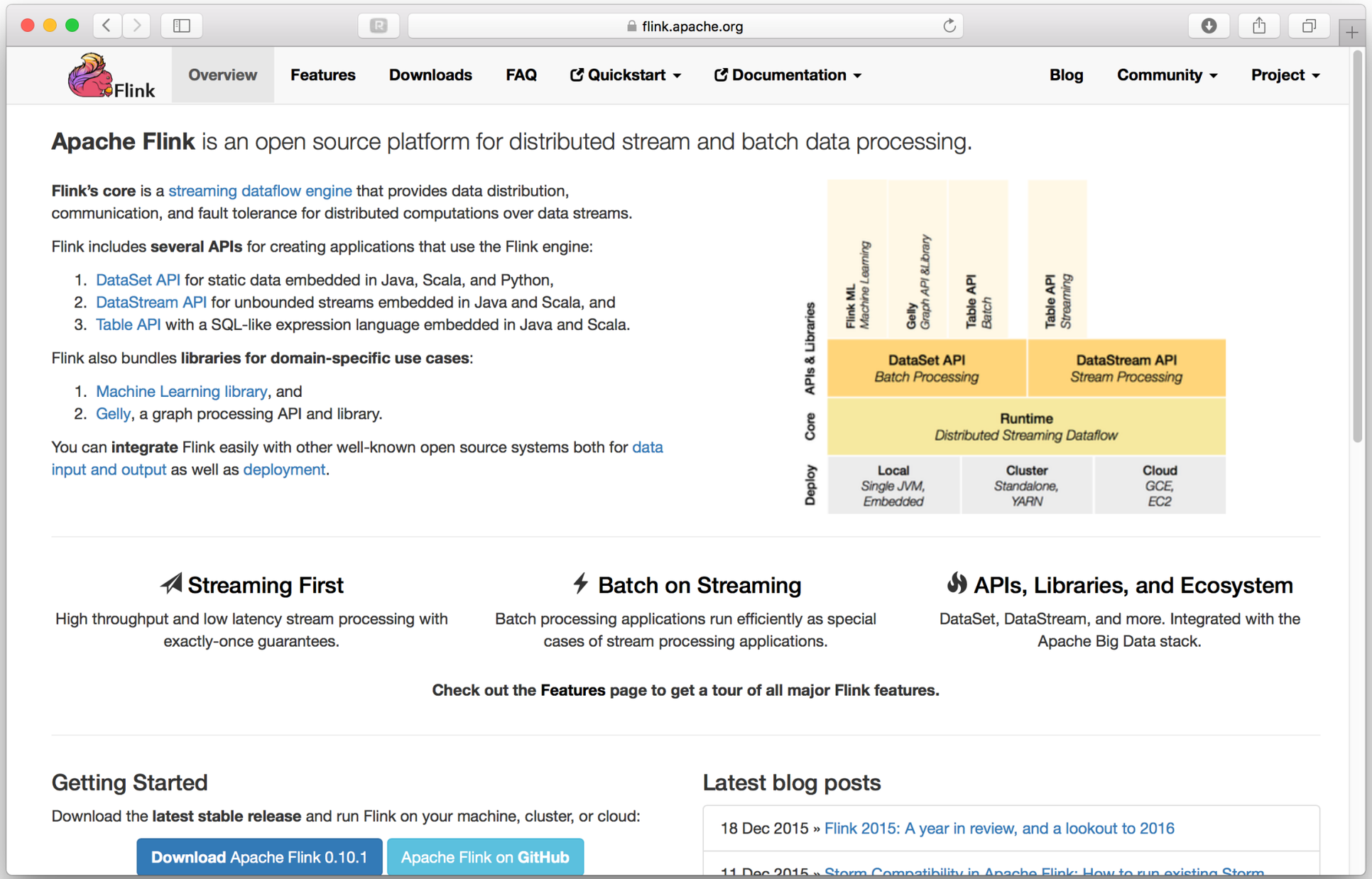
Stream Platform Architecture (a.k.a Kappa Architecture)



다른 스트림 처리 엔진과의 비교

				
Delivery Semantics	At least once	Exactly once	At least once	Exactly once
State Management	Stateless	Stateful	Stateful	Stateful
Latency	Sub-second	Seconds	Sub-seconds	Sub-seconds
Language Support	Ruby, Python, Javascript, Perl	Scala, Java, Python	Scala, Java	Scala, Java, Python

Getting Started



http://flink.apache.org

Thank you!

http://j.mp/ossdevconf-2015-park