Spark 사용자를 위한

Apache Flink 둘러보기

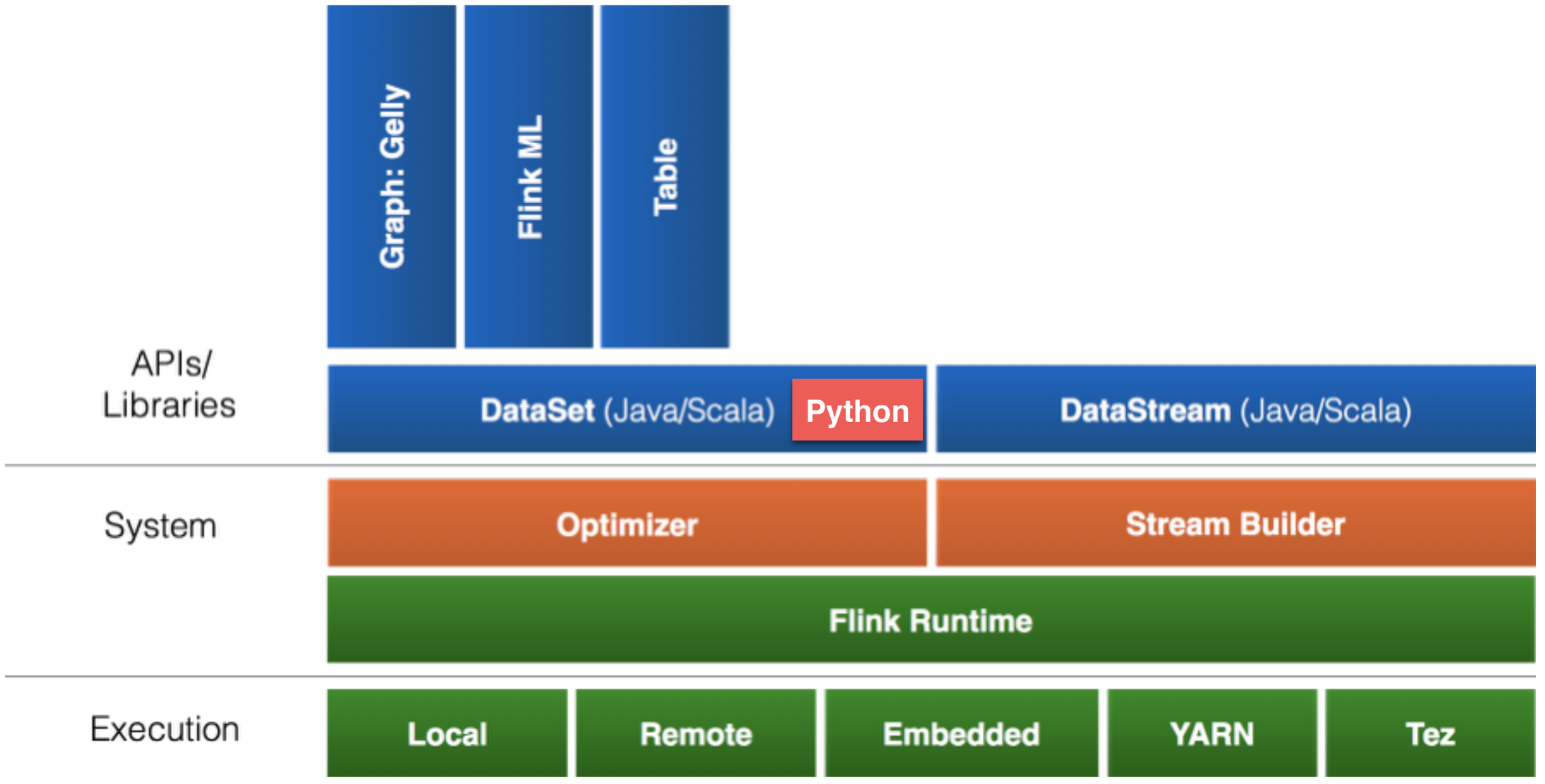
이문수
NFLabs
moon@nflabs.com

Apache Flink

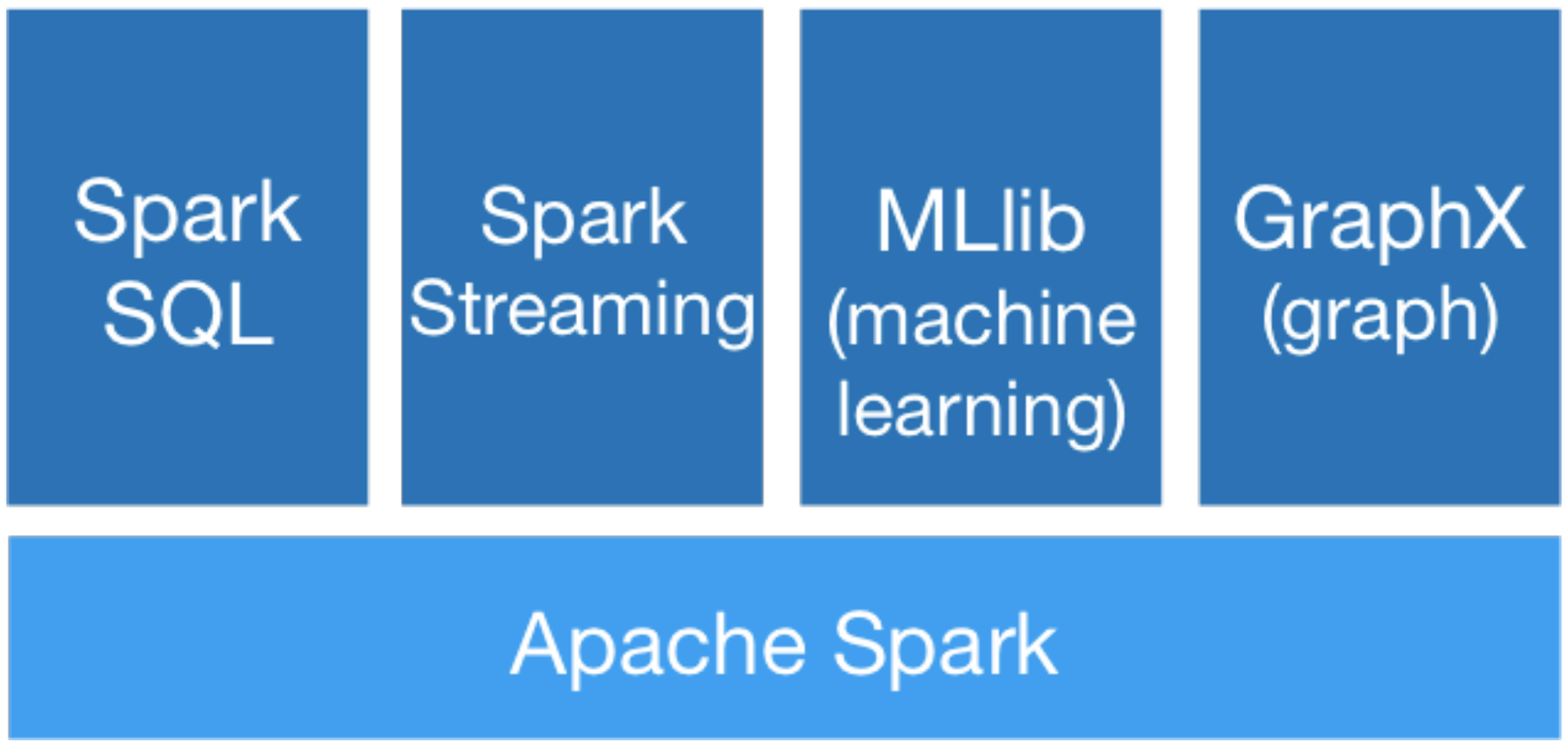
Streaming dataflow 엔진을 기반으로 한

스트리밍 & 배치 프로세싱 플랫폼

Flink 구성 요소를 살펴보면



Spark 과 비슷


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

비교해보면

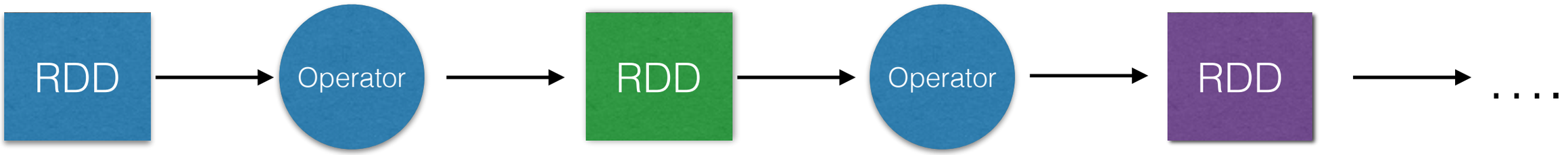
	Flink	Spark
Data	DataSet	RDD
DataFrame	Table	DataFrame
Machine Learning	Flink ML	MLlib
Graph	Gelly	Graphx
SQL	-	SparkSQL
Streaming	DataStream	SparkStreaming

DataSet

DataSet

Spark 의 RDD에 해당

Spark 은 RDD 변환하고, 변환이 끝나면 또 변환하고, 이 과정을 반복 -> Data centric

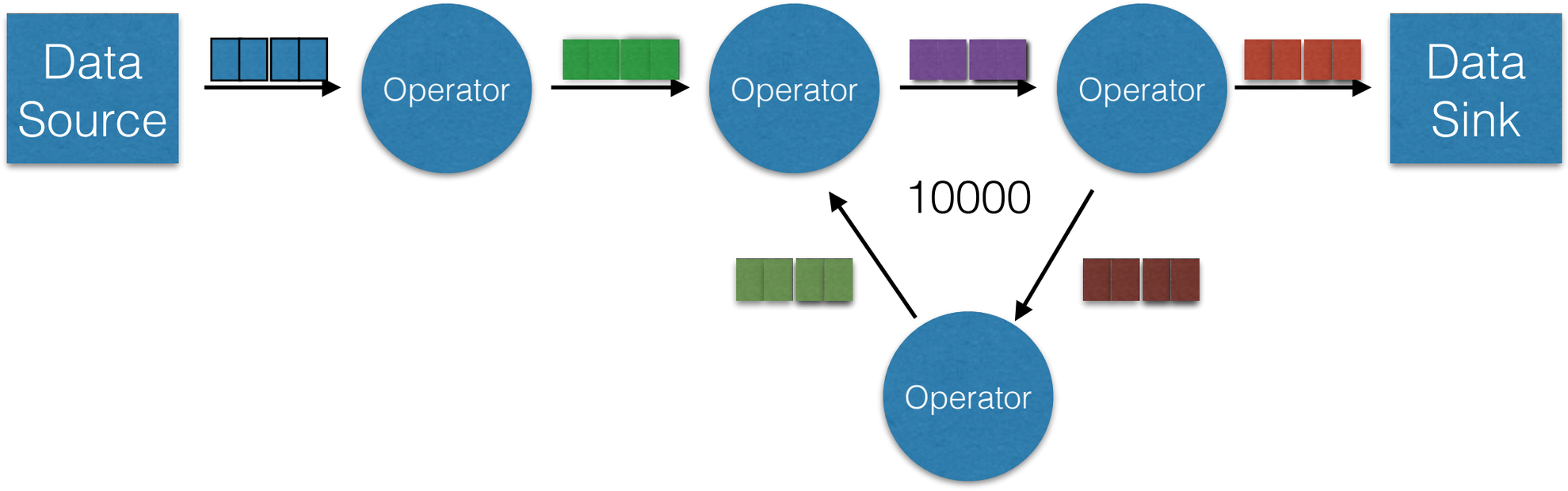


Flink 는 Operator 들 사이로 데이터를 흘려보냄

-> Operator centric

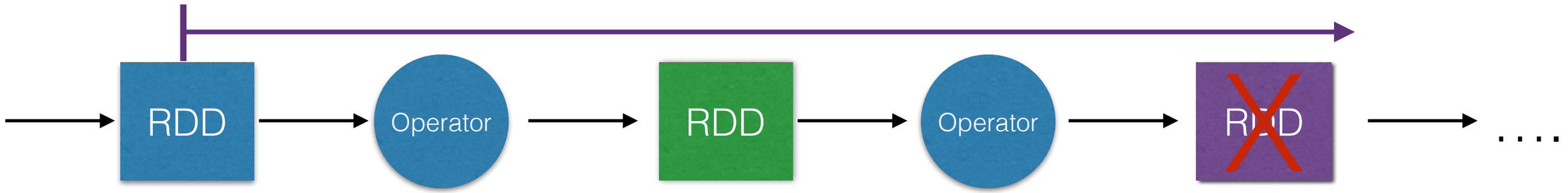
Iteration

Spark 처럼 Operator 와 RDD 를 끊임없이 생성하지 않고 수행가능

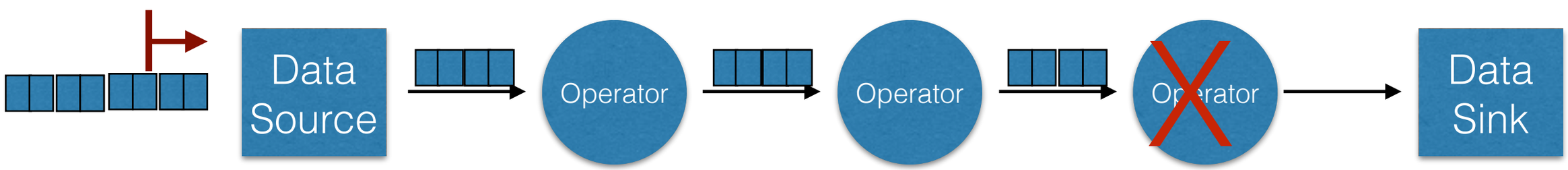


Fault Tolerance

Spark 은 Lineage 을 이용해 전체 과정을 다시 계산
(persistance 된 RDD 가 있으면 그 이후로 계산)

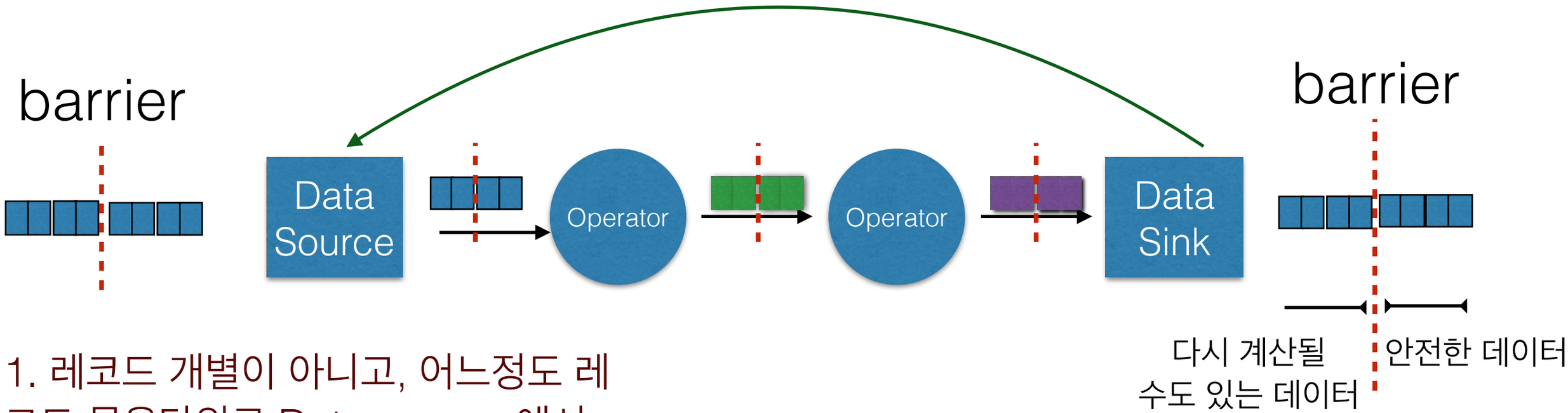


Flink 는 마지막 checkpoint 이후로 DataSource 에서 Replay



Checkpoint

3. Sink 에서는 Source 에 barrier 도착을 알려 Source 에서 필요없는 데이터를 지우도록 한다.



1. 레코드 개별이 아니고, 어느정도 레코드 묶음단위로 Data source 에서 data stream 중간에 barrier 을 끼워 넣는다

2. Data sink 에 barrier 가 도착하면 해당 부분까지 계산이 완료 된것으로 본다.

DataSet API

DataSet 을 만들 때

SparkContext 에 해당하는 ExecutionEnvironment 를 통해 만들 수 있다.

```
val env = ExecutionEnvironment.getExecutionEnvironment
```

메모리에 있는 collection으로 부터 생성하거나

```
val text = env.fromElements(“hello”, “world”)
```

파일로 부터 생성하거나

```
val text = env.readTextFile("file:///path/to/file")
```

사용자가 지정한 InputFormt 으로부터

```
val data = env.createInput(InputFormat<X, ?> inputFormat)
```

DataSet 을 저장 할때

파일로 저장하거나

```
val text = data.writeAsText(“file:///…”)
```

화면에 뿌리거나

```
data.print()
```

사용자가 지정한 OutputFormat 으로 내보내거나

```
data.output(OutputFormat<T> outputFormat)
```

Transformations

map
flatMap
mapPartition
filter
reduce
reduceGroup
aggregate
join
coGroup
cross
union
hashPartition
sortPartition

```
val text = env.fromElements(
      "Who's there?",
      "I think I hear them. Stand, ho! Who's
there?”)
```

```
val counts = text
      .flatMap {_.toLowerCase.split("\\W+") }
      .map { (_, 1) }
      .groupBy(0)
      .sum(1)
```

DataStream API

DataStream 을 만들 때

SparkContext 에 해당하는 ExecutionEnvironment 를 통해 만들 수 있다.

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
```

소켓으로 부터 생성하거나

```
val dataStream = env.socketTextStream(“localhost”, “9999”)
```

디렉토리에 추가/변경되는 파일을 모니터링

```
val dataStream = env.readFileStream(String path, long
      checkFrequencyMillis, WatchType watchType)
```

사용자가 지정한 Source 으로부터

```
val dataStream = env.addSource(sourceFunction)
```

Kafka, RabbitMQ, Flume

DataStream 을 저장 할 때

파일로 저장하거나

```
dataStream.writeAsText(parameters)
```

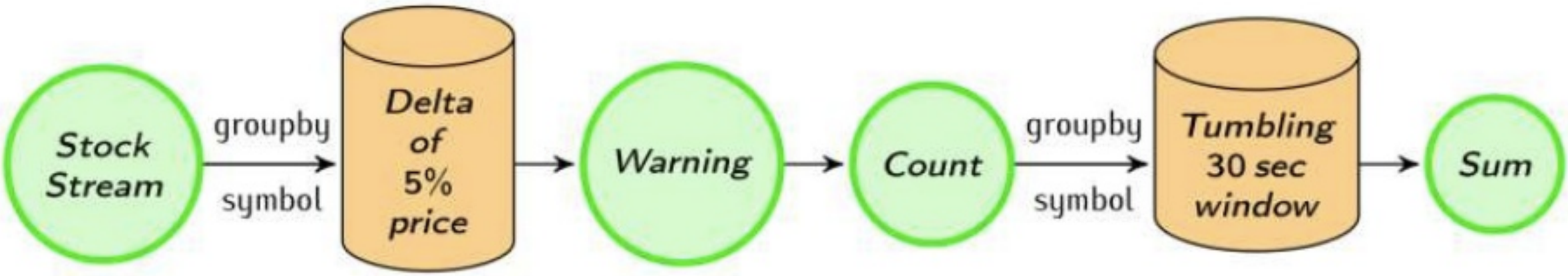
화면에 뿌리거나

```
dataStream.print()
```

사용자가 구현한 Sink를 사용

```
dataStream.addSink(sinkFunction)
```

DataStream API 예




```
case class Count(symbol: String, count: Int)
val defaultPrice = StockPrice("", 1000)

//Use delta policy to create price change warnings
val priceWarnings = stockStream.groupBy("symbol")
  .window(Delta.of(0.05, priceChange, defaultPrice))
  .mapWindow(sendWarning _)

//Count the number of warnings every half a minute
val warningsPerStock = priceWarnings.map(Count(_, 1))
  .groupBy("symbol")
  .window(Time.of(30, SECONDS))
  .sum("count")
```

Windowing policy

- Count
- Time
- Delta

Table API

Spark, R 의 DataFrame, Python 의 Pandas 와 비슷한 API 제공

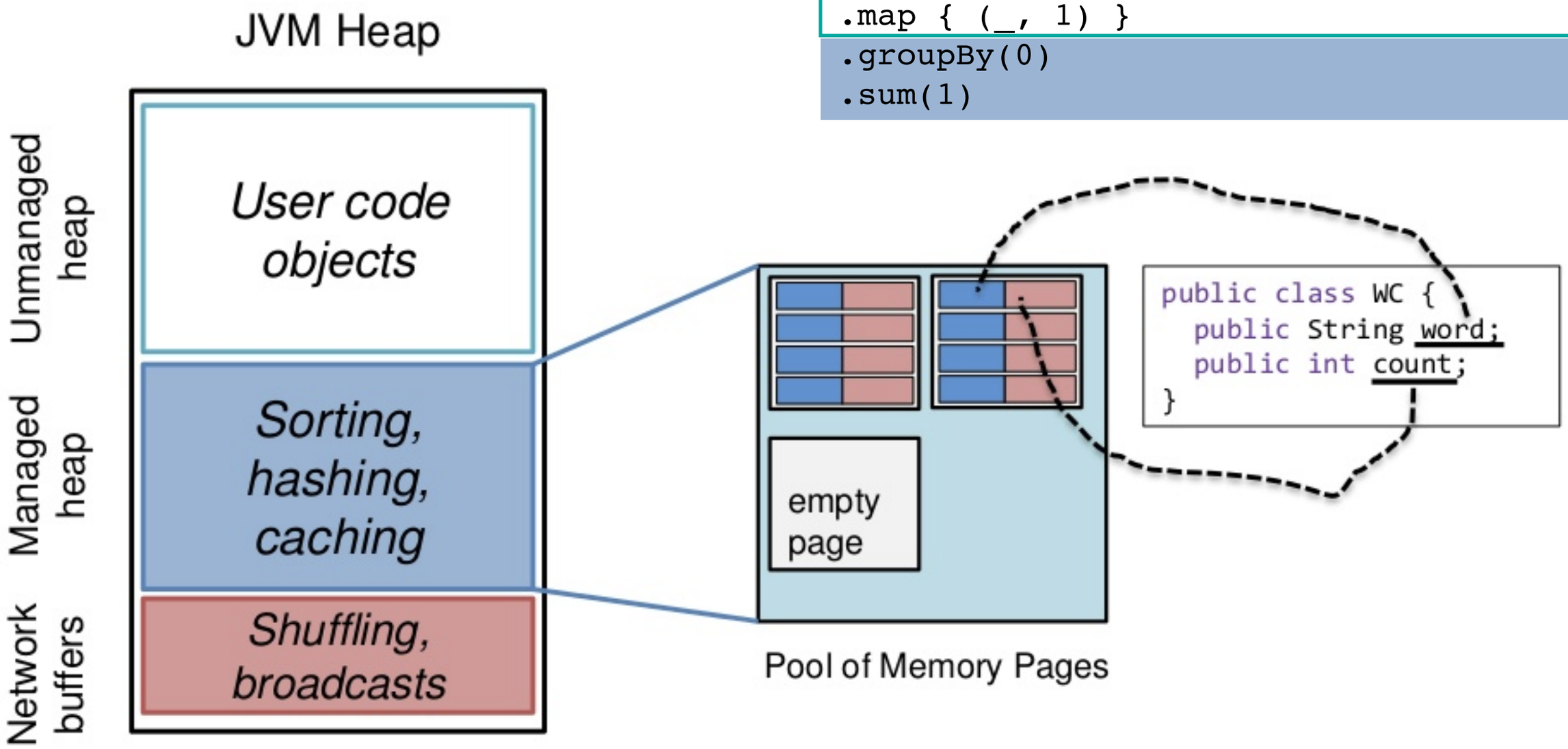
```
import org.apache.flink.api.scala._
import org.apache.flink.api.scala.table._

case class WC(word: String, count: Int)
val input = env.fromElements(WC("hello", 1), WC("hello", 1), WC("ciao", 1))
val expr = input.toTable
val result = expr.groupBy('word).select('word, 'count.sum as 'count).toSet[WC]
```

```
val customers = envreadCsvFile(…).as('id, 'mktSegment)
      .filter("mktSegment = AUTOMOBILE")

val orders = env.readCsvFile(…)
      .filter( o => dateFormat.parse(o.orderDate).before(date) )
      .as("orderId, custId, orderDate, shipPrio")
```

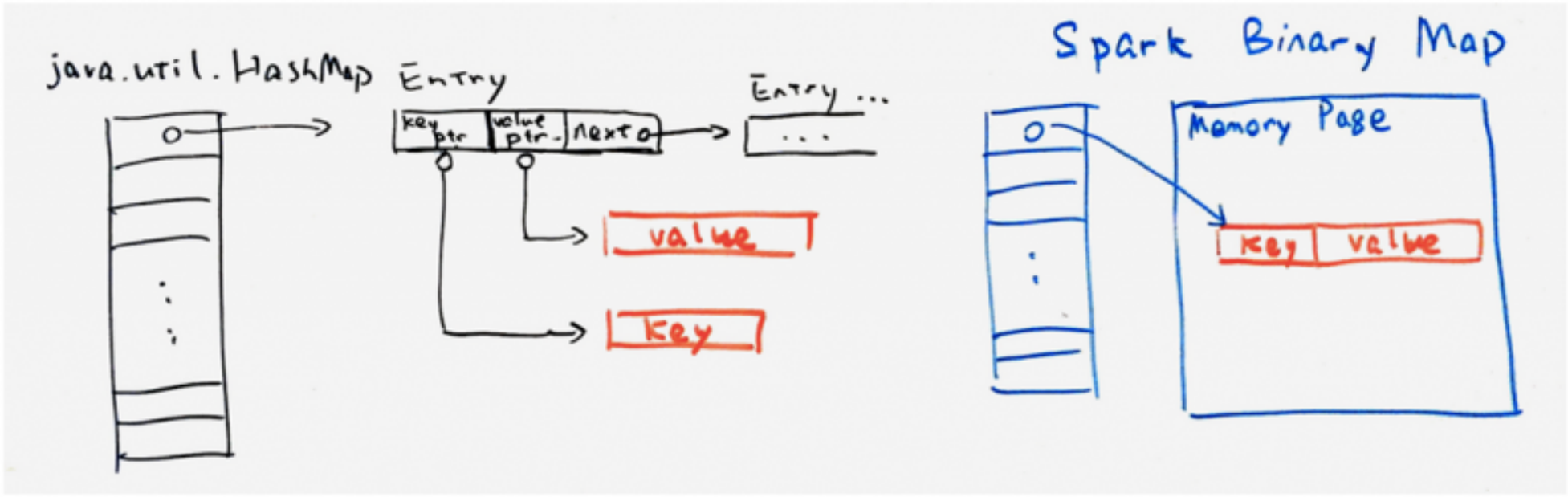
```
val items = orders
      .join(customers).where("custId = id")
      .join(lineitems).where("orderId = id")
      .select("orderId, orderDate, shipPrio,
          extdPrice * (Literal(1.0f) - discount) as revenue")

val result = items
      .groupBy("orderId, orderDate,  shipPrio")
      .select('orderId, revenue.sum, orderDate, shipPrio")
```

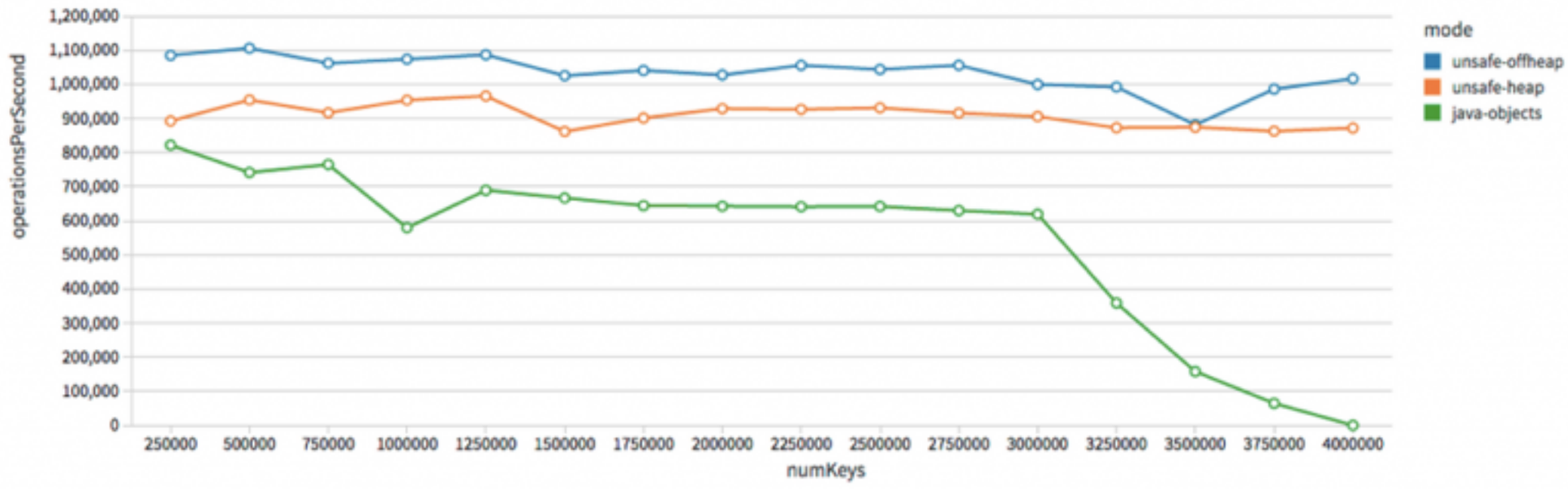
Memory Management

```
val counts = text
      .flatMap {_.toLowerCase.split("\\W+") }
      .map { (_, 1) }
      .groupBy(0)
      .sum(1)
```

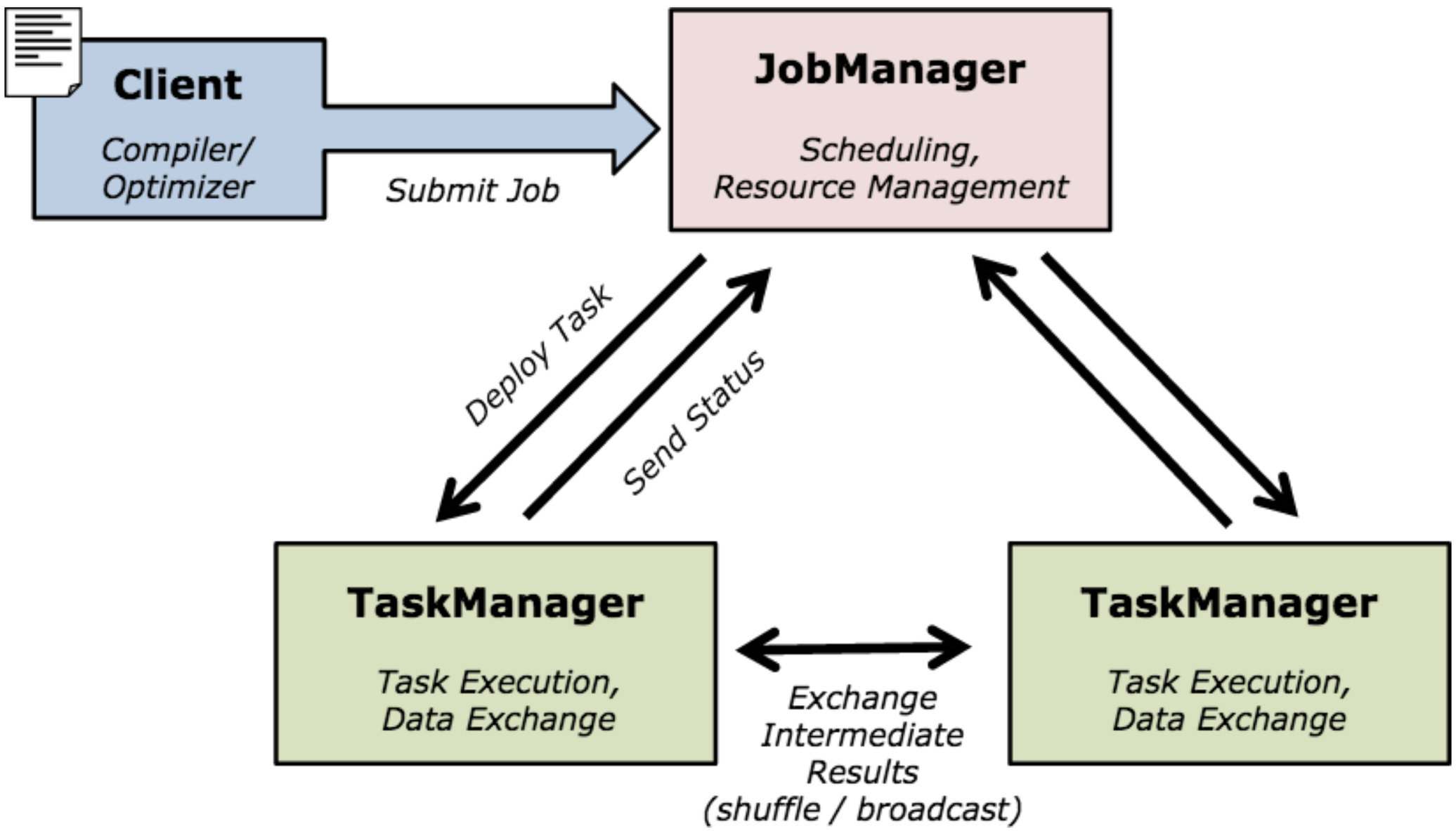


Spark에서는 Tungsten 으로 이제하려고 시도하는 것





Architecture



감사합니다

이문수
http://zeppelin.incubator.apache.org
moon@apache.org