Scala在大数据中应用

曹宝@挖财

“曹宝，原名曹静静。一个程序猿，一个屌丝。多年的互联网和金融从业经验并畅游在大数据的海洋。曾主导开发过淘宝分布式消息中间件，Ebay广告实时和离线计算平台，摩根斯坦利OTC清算系统。资深JAVA程序猿，Scala爱好者，架构师”

–某大牛

Agenda

- Scala & BigData
- Lambda Architecture
- Word Count
- Pain Point
- Q&A

Scala & BigData

Why Scala

Good balance between productivity and performance

Integration with big data ecosystem

Functional paradigm

Capture functions and ship them across the network

Static typing

Productivity and Performance

If you are Python/R/Matlab or Blank background, Scala is a lot less intimidating than Java or C++.

Build on JVM and keep most of performance from Java

Integration with Big Data Ecosystem

- Scalding (Cascading)
- Summingbird (Scalding and Storm)
- Kafka
- Spark
- MLlib(Spark ML lib)
- Algebird(Scalding's Matrix API)

Function Paradigm

- Functional paradigm which fits well within the Map/Reduce and big data model. Batch processing works on top of immutable data, transforms with map and reduce operations, and generates new copies.
- Real time log streams are essentially lazy streams.
- Most Scala data frameworks have the notion of some abstract data type that's extremely consistent with Scala's collection API. A glance at TypedPipe in Scalding and RDD in Spark, and you'll see that they all have the same set of methods, e.g. map, flatMap, filter, reduce, fold and groupBy. One could just learn the standard collection and easily pick up one of the libraries
- Many libraries also have frequent reference of category theory like monoid to guarantee the correctness of distributed operations. Equipped such knowledge it'll be a lot easier to understand techniques like map-side reduce.

Ship Functions across The Network

- Function is Object
- Object can be shipped across the network
- Distribute computation across network at runtime

Static Typing

- Typing reduce error prone code
- No-structure or semi-structure can be bind with business logic easily with Typing
- Typing is not stored with data and be maintained within separated storage like Hive meta store

```java
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class LineIndexer {
    public static void main(String[] args) {
        JobClient client = new JobClient();
        JobConf conf =
                new JobConf(LineIndexer.class);
        conf.setJobName("LineIndexer");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(conf,
                new Path("input"));
        FileOutputFormat.setOutputPath(conf,
                new Path("output"));
        conf.setMapperClass(
                LineIndexMapper.class);
        conf.setReducerClass(
                LineIndexReducer.class);
        client.setConf(conf);
        try {
            JobClient.runJob(conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static class LineIndexMapper extends MapReduceBase
            implements Mapper<LongWritable, Text,
            Text, Text> {
        private final static Text word =
                new Text();
        private final static Text location =
                new Text();

        public void map(
                LongWritable key, Text val, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
            FileSplit fileSplit = (FileSplit) reporter.getInputSplit();
            String fileName = fileSplit.getPath().getName();
            location.set(fileName);
            String line = val.toString();
            StringTokenizer itr = new
                    StringTokenizer(line.toLowerCase());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                output.collect(word, location);
            }
        }
    }

    public static class LineIndexReducer extends MapReduceBase
            implements Reducer<Text, Text,
            Text, Text> {
        public void reduce(Text key,
                           Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
            boolean first = true;
            StringBuilder toReturn =
                    new StringBuilder();
            while (values.hasNext()) {
                if (!first) toReturn.append(", ");
                first = false;
                toReturn.append(
                        values.next().toString());
            }
            output.collect(key,
                    new Text(toReturn.toString()));
        }
    }
}
```

```
import com.twitter.scalding._

class InvertedIndex(args:Args)extends Job(args){

  val tweets=Tsv("tweets.tsv",('id, 'text))

  val wordToTweets=
    tweets
      .flatMap(('id, 'text)->('word, 'tweetId)){
      fields:(Long,String)=>
        val(tweetId,text)=fields
        text.split("\\s+").map{word=>(word,tweetId)}
    }

  val invertedIndex=
    wordToTweets.groupBy('word) {  _.toList[Long]('tweetId->'tweetIds) }

  invertedIndex.write(Tsv("output.tsv"))
}
```

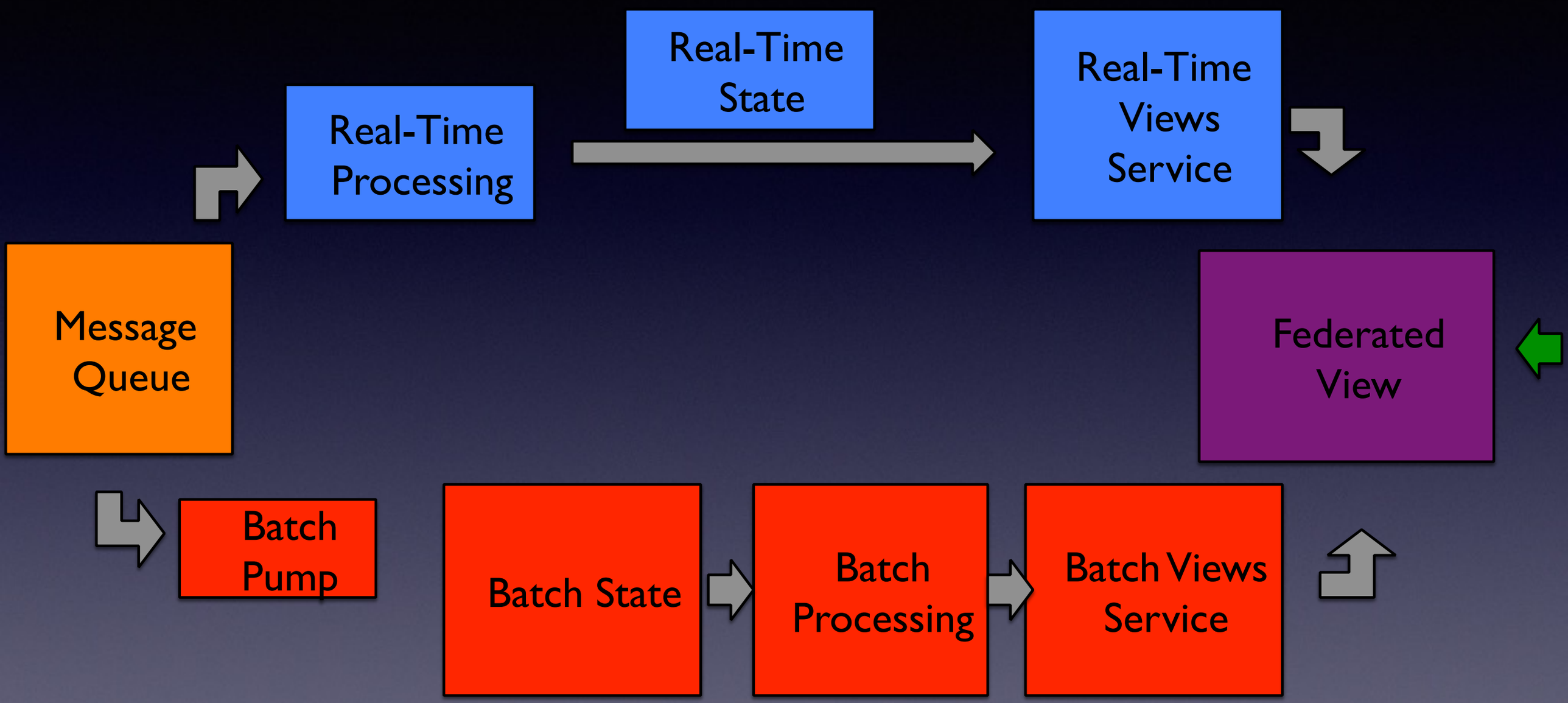
74 vs 18

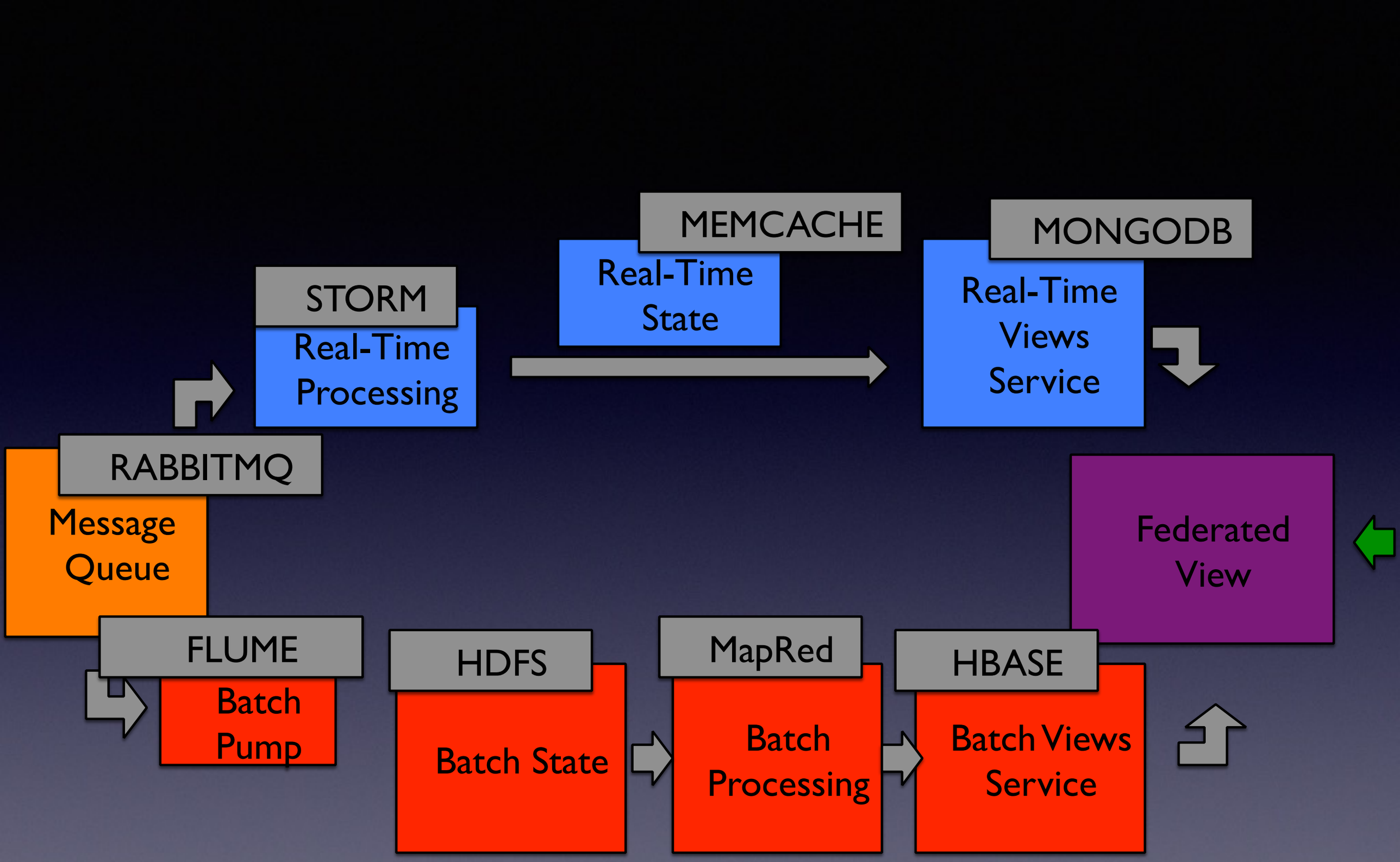
Lambda Architecture

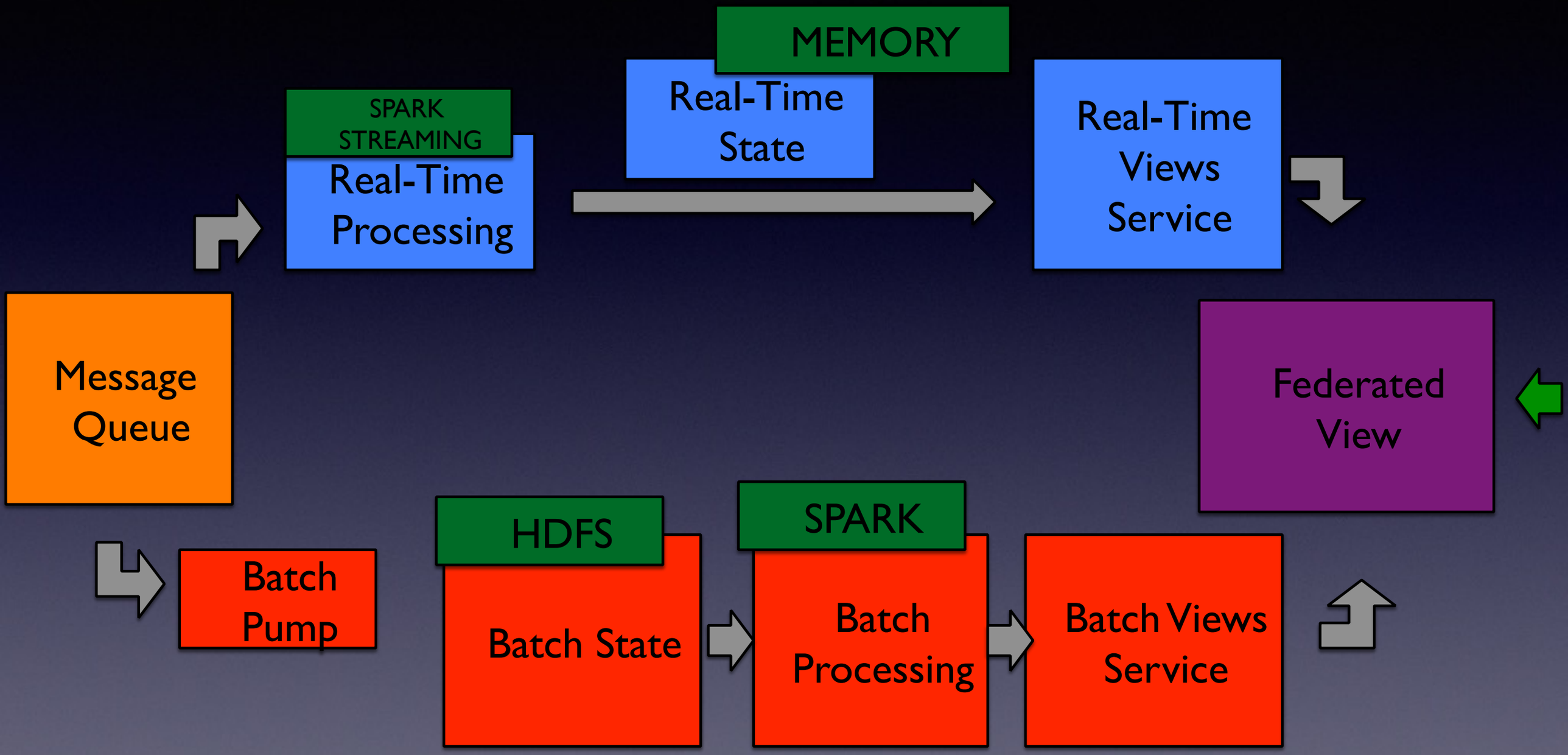
Lambda expression is a anonymous function and a closure

But what Lambda architecture is…

Lambda architecture is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. This approach to architecture attempts to balance latency, throughput, and fault-tolerance by using batch processing to provide comprehensive and accurate views of batch data, while simultaneously using real-time stream processing to provide views of online data.


Real-Time Processing
Real-Time State
Real-Time Views Service
Message Queue
Federated View
Batch Pump
Batch State
Batch Processing
Batch Views Service


MEMCACHE
Real-Time State
MONGODB
STORM
Real-Time Processing
Real-Time Views Service
RABBITMQ
Message Queue
Federated View
FLUME
Batch Pump
HDFS
Batch State
MapRed
Batch Processing
HBASE
Batch Views Service


MEMORY
SPARK STREAMING
Real-Time Processing
Real-Time State
Real-Time Views Service
Federated View
Message Queue
Batch Pump
HDFS
Batch State
SPARK
Batch Processing
Batch Views Service

Two Proven LA Framework

SummingBird

Spark Batch and Streaming

Word Count

Scalding

```scala
import com.twitter.scalding._

class WordCountJob(args: Args) extends Job(args) {
  TypedPipe.from(TextLine(args("input")))
    .flatMap { line => line.split(" ") }
    .groupBy { word => word } // use each word for a key
    .size // in each group, get the size
    .write(TypedTsv[(String, Long)](args("output")))
}
```

SummingBird

```
def wordCount[P <: Platform[P]](
    source: Producer[P, Status],
    store: P#Store[String, Long]) =
  source
    .filter(_.getText != null)
    .flatMap { tweet: Status => tweet.getText.split(" ").map(_ -> 1L) }
    .sumByKey(store)
```

Spark

```
val textFile = spark.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

Pain Point

- Before Scala 2.11 Tuple/Case class have 22 fields limitation
- Spark 2G max shuffle size
- In the case of null values, Java treats two null values as equivalent. SQL does not treat null values as equal.
- Abuse “reduce” liked collection function lead to low performance of batch process
- Lack of optimization Using column statistics
- Schema change auto detect or tolerate

Q&A