WHY DO WE NEED 9 DIFFERENT DATA PROCESSING ENGINES?

AND WHAT MAKES SPARK 2.32X BETTER THAN FLINK?

Wojciech Pituła @ Nordea Common Platforms

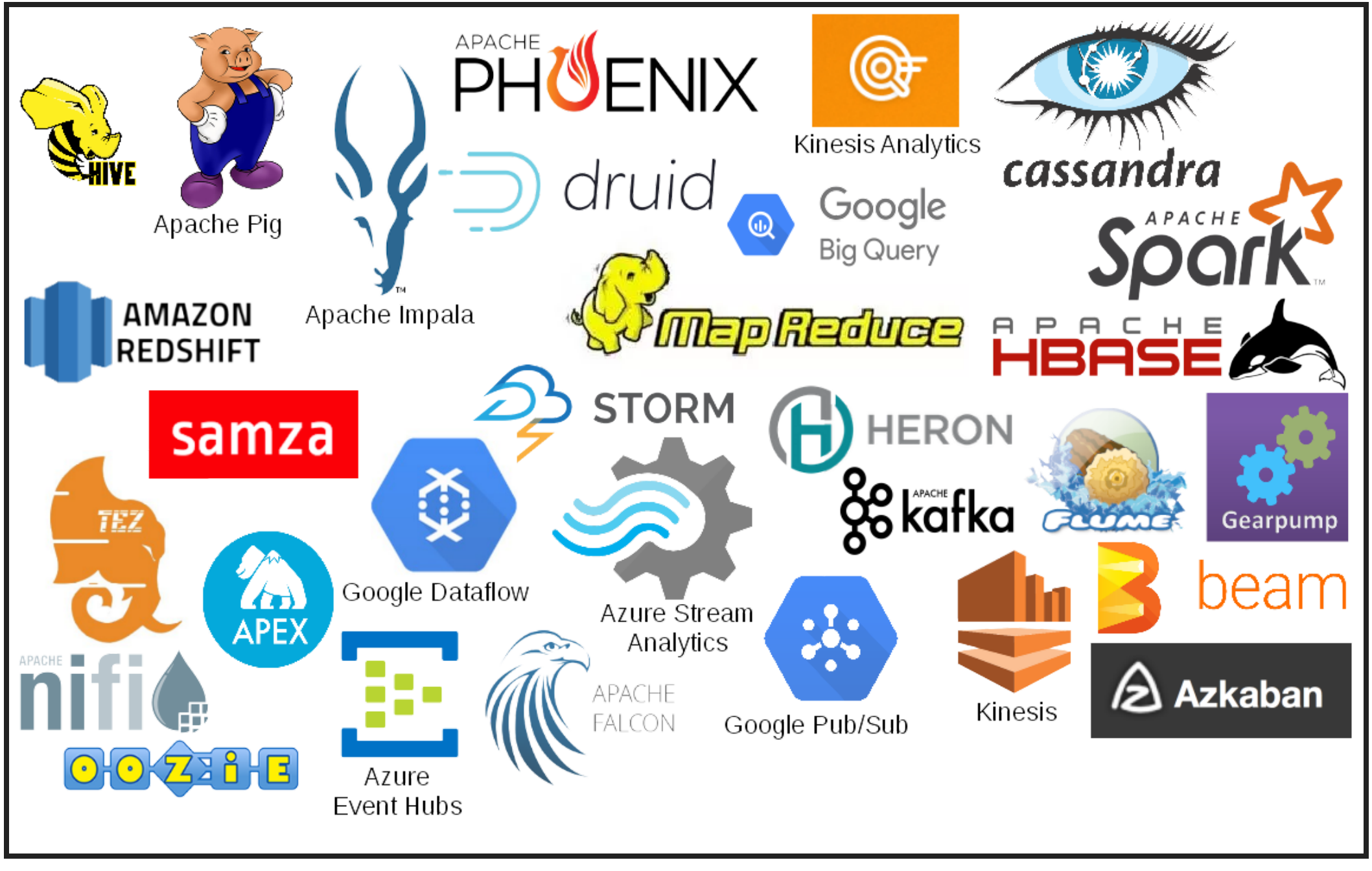
WHAT IS IT ALL ABOUT?

AFTER A YEAR WITH SPARK ...


APACHE
Spark
TM

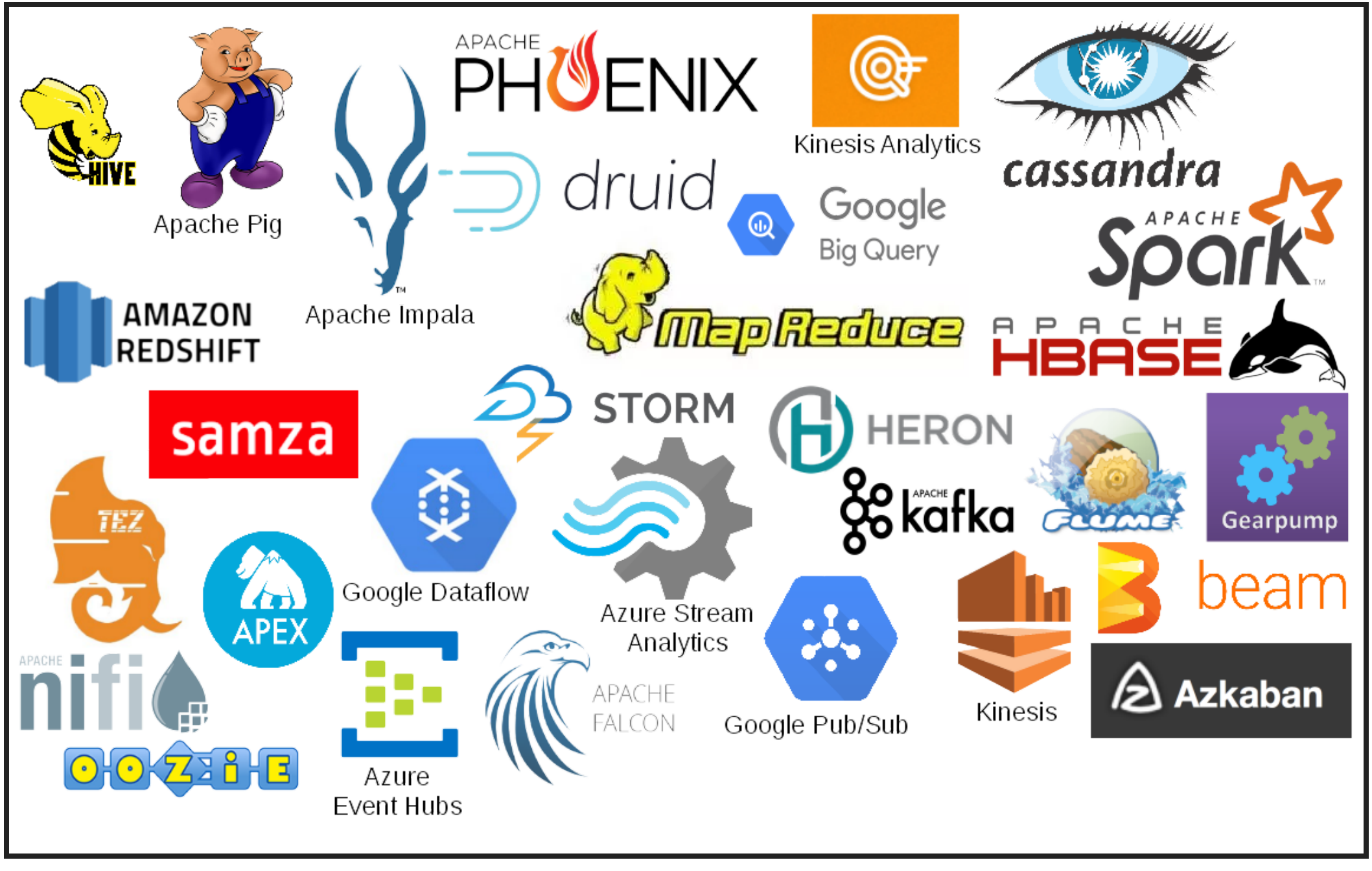

hadoop
Map Reduce

Flink

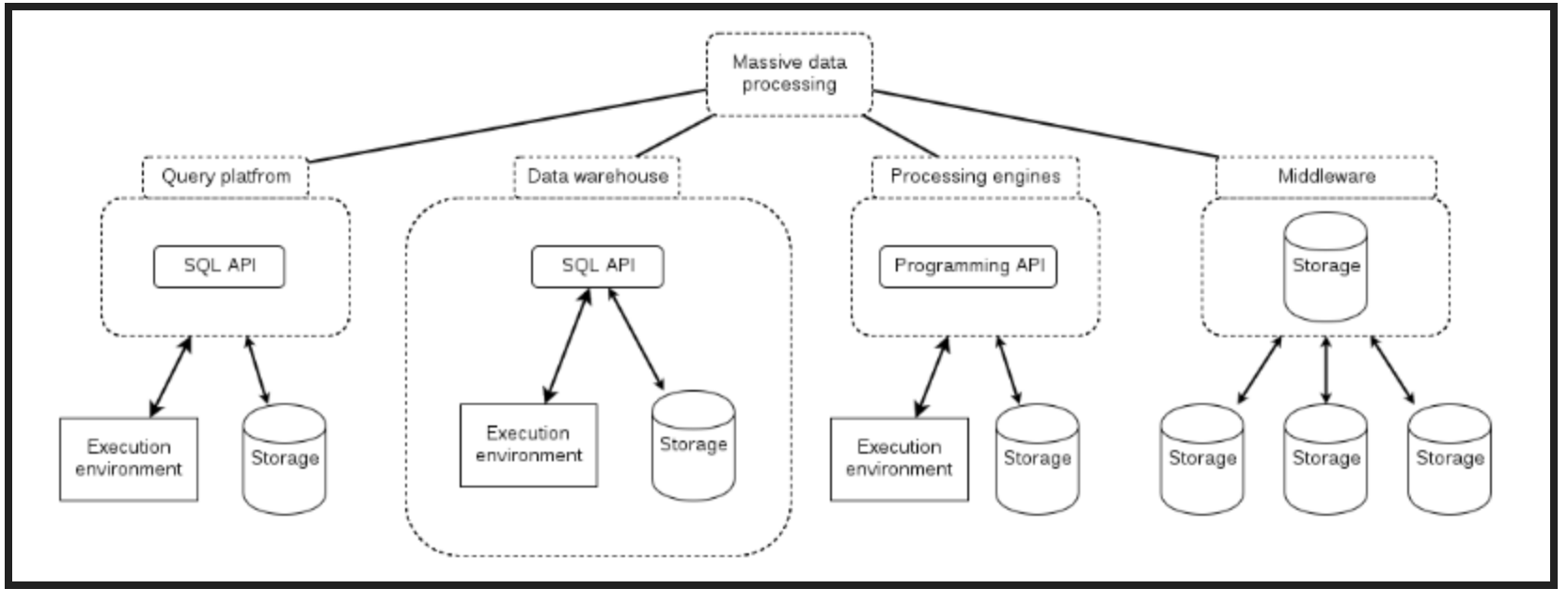

HIVE
Apache Pig
APACHE PHOENIX
Apache Impala
druid
Kinesis Analytics
cassandra
Google Big Query
APACHE Spark
AMAZON REDSHIFT
MapReduce
APACHE HBASE
samza
STORM
HERON
TEZ
Google Dataflow
Azure Stream Analytics
APACHE kafka
FLUME
Gearpump
APEX
beam
APACHE nifi
APACHE FALCON
Google Pub/Sub
Kinesis
Azkaban
OOZIE
Azure Event Hubs

Comparison of modern platforms for massive data processing

OVERVIEW


HIVE
Apache Pig
APACHE PHOENIX
Apache Impala
druid
Kinesis Analytics
cassandra
Google Big Query
APACHE Spark
AMAZON REDSHIFT
MapReduce
APACHE HBASE
samza
STORM
HERON
TEZ
Google Dataflow
Azure Stream Analytics
APACHE kafka
FLUME
Gearpump
APEX
beam
APACHE nifi
APACHE FALCON
Google Pub/Sub
Kinesis
Azkaban
OOZIE
Azure Event Hubs

CLASSIFICATION



CLASSIFICATION

Query platforms	Data Warehouses	Engines	Middleware
• Apache Hive • Apache Pig • Apache Impala • Apache Phoenix • Apache HAWQ • Amazon Kinesis Analytics	• Amazon Redshift • Druid.io • Cassandra • Google BigQuery • Apache Hbase	• Hadoop MapReduce • Hadoop Streaming • Apache Spark • Apache Storm • Twitter Heron • Apache Flink • Apache Samza • Apache Tez • Google Dataflow • Azure Stream Analytics • Apache Beam • Apache S4 • Apache Apex • Kafka Streams • Apache Gearpump	• Apache Kafka • Amazon Kinesis • Google Pub/Sub • Azure Event Hub • Apache NiFi • Apache Falcon • Apache Oozie • Azkaban • Apache Flume

WHAT TO COMPARE ?

- Actively maintained
- Open-source

BATCH VS STREAM

Batch	Stream
• Hadoop MapReduce • Apache Spark • Apache Flink • Apache Apex	• Apache Spark • Apache Flink • Apache Apex • Apache Storm • Twitter Heron • Apache Samza • Kafka Streams • Apache Gearpump

APIS

PROCESSORS VS DSL

PROCESSOR

```
public static class SplitSentence extends BaseBasicBolt {

    @Override
    public void execute(Tuple input, BasicOutputCollector collector) {
        String[] split = input.getString(0).split("[\\s]+");
        for (String word : split) {
            collector.emit(Arrays.asList(word));
        }
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }
}
```

PROCESSORS COMPOSITION

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", kafkaSpout, 5);
builder.setBolt("split", new SplitSentence(), 8)
        .shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12)
        .fieldsGrouping("split", new Fields("word"));
```

DSL

```
val sc = new SparkContext(conf)

sc.textFile(args(0))
  .flatMap(_.split(' '))
  .map(t => (t,1))
  .reduceByKey(_+_)
```

PROCESSORS VS DSL

	Processors	DSL
MapReduce	☑	☒
Apache Samza	☑	☒
Apache Spark	☒	☑
Apache Flink	☒	☑
Kafka Streams	☑	☑
Apache Gearpump	☑	☑
Apache Storm	☑	☑
Twitter Heron	☑	☒
Apache Apex	☑	☒

LANGUAGES

LANGUAGES

	Java	Scala	Other
MapReduce	☑	☒	Any, via Hadoop Streaming
Apache Samza	☑	☒	
Apache Spark	☑	☑	Python, R
Apache Flink	☑	☑	
Kafka Streams	☑	☒	
Apache Gearpump	☑	☑	
Apache Storm	☑	☒	
Twitter Heron	☑	☒	
Apache Apex	☑	☒	

TYPE-SAFETY

NOT TYPE-SAFE

```
public void execute(Tuple input, BasicOutputCollector collector) {
        String[] split = input.getString(0).split("[\\s]+");
        for (String word : split) {
            collector.emit(Arrays.asList(word));
        }
    }
```

TYPE-SAFE

```
val ints: RDD[Int] = ???
val intsPlus2: RDD[Int] = ints.map(_ + 2)
val strings: RDD[String] = intsPlus2.map(_.toString)
```

SAFE VS UNSAFE

	Type-safe
MapReduce	☑
Apache Samza	☒
Apache Spark	☑
Apache Flink	☑
Kafka Streams	☑
Apache Gearpump	☑ ☒
Apache Storm	☒
Twitter Heron	☒
Apache Apex	☑

OTHER APIS

Cascading

Weakly-typed processor-based Java API for MapReduce

Scalding

Strongly-typed DSL Scala API based on Cascading

Apache Beam

Strongly-typed DSL-based Java and Python API for Apache Spark, Apache Flink, Apache Apex, Google Dataflow

Apache Crunch

Strongly typed DSL-based Java ans Scala API running on top of MapReduce and Apache Spark

MESSAGE DELIVERY GUARANTEES


WARNING
MATH
PROBLEM
AHEAD

GUARANTEES

- at-least-once
- at-most-once
- exactly-once

PROBLEMS

- ill-defined
- hard to verify

DEPLOYMENT

DEPLOYMENT

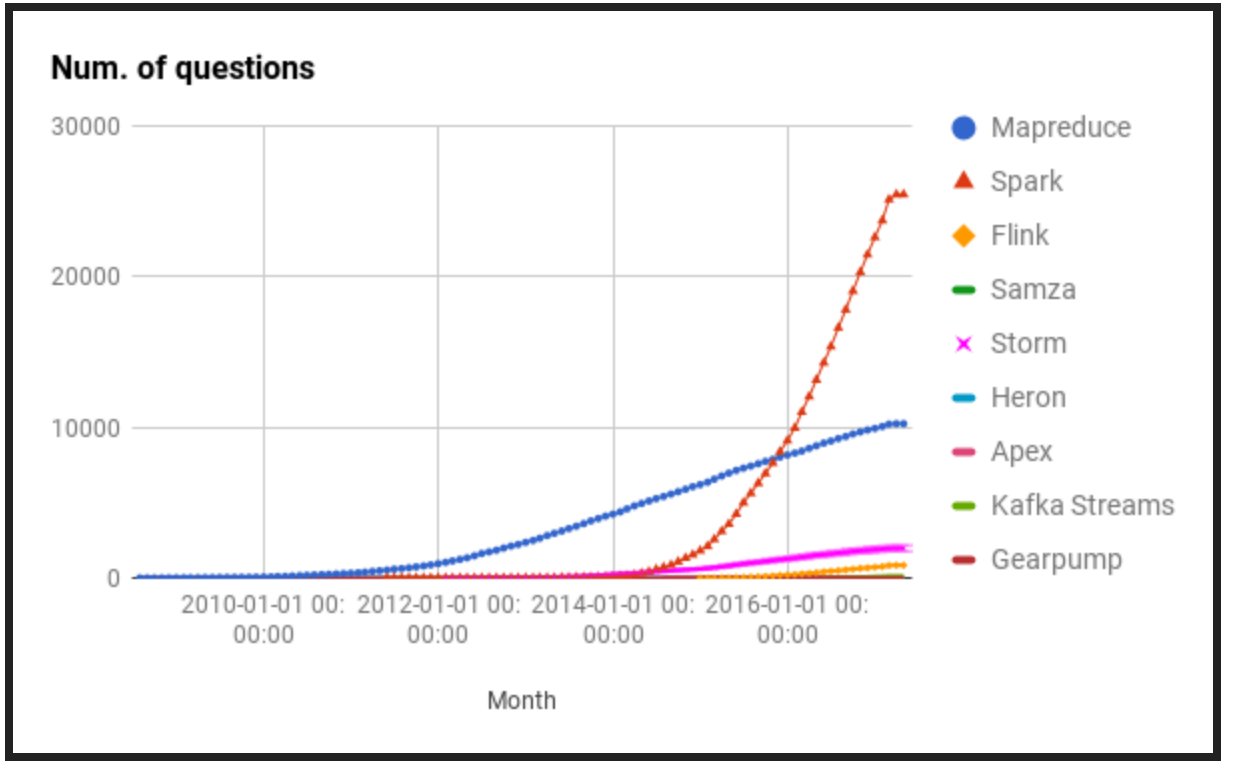
- Local
- Yarn
- Mesos
- Standalone

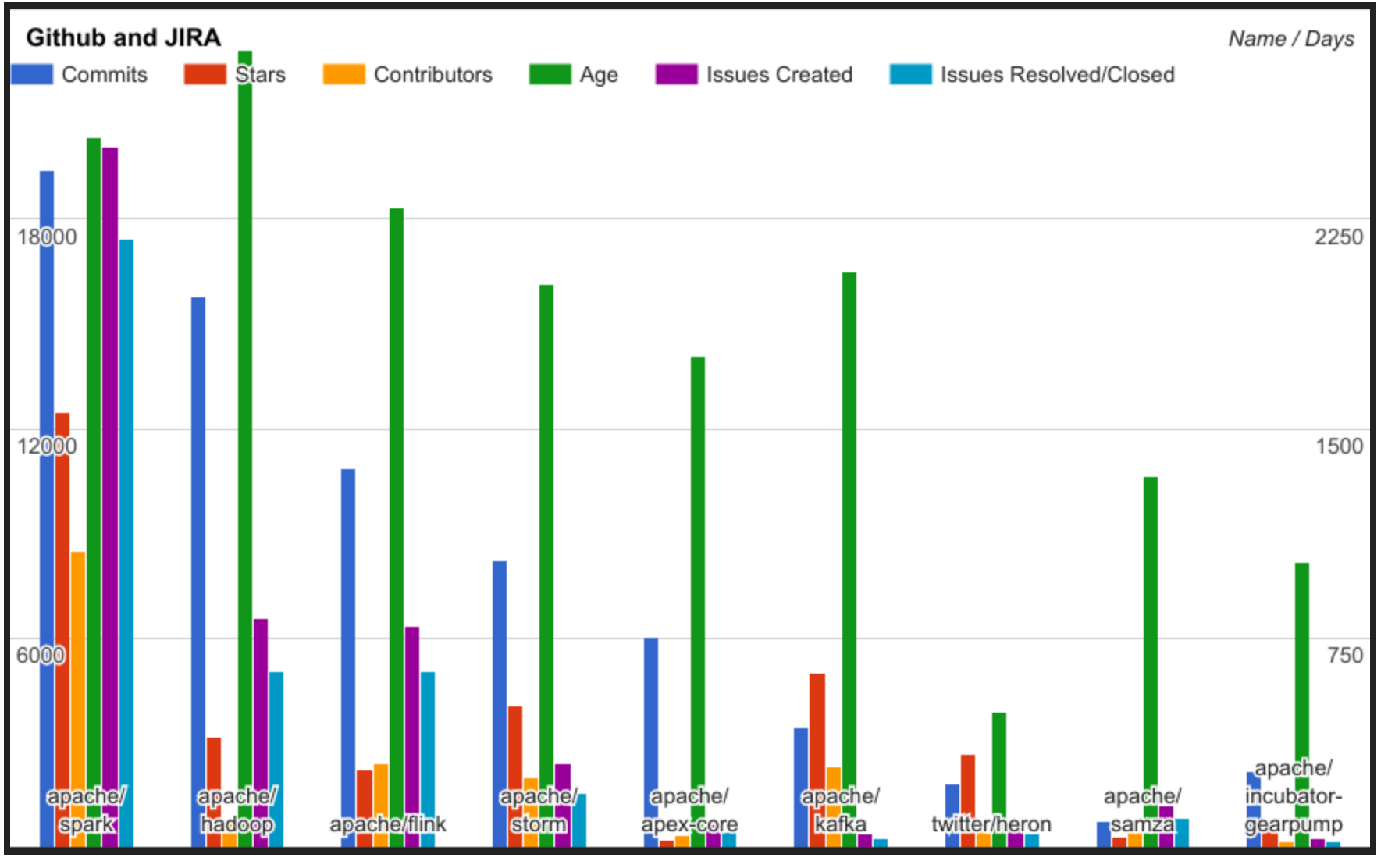
DEPLOYMENT

	Local	Yarn	Mesos	Standalone	Any/None
MapReduce	☑	☑	☒	☒	☒
Apache Samza	☑	☑	☒	☒	☒
Apache Apex	☑	☑	☒	☒	☒
Apache Storm	☑	☒	☒	☑	☒
Twitter Heron	☑	☑	☑	☒	☒
Apache Gearpump	☑	☑	☒	☑	☒
Apache Spark	☑	☑	☑	☑	☒
Apache Flink	☑	☑	☑	☑	☒
Kafka Streams	☑	☑	☑	☑	☑

POPULARITY

Maturity


Num. of questions
30000
20000
10000
0
2010-01-01 00:00:00
2012-01-01 00:00:00
2014-01-01 00:00:00
2016-01-01 00:00:00
Month
Mapreduce
Spark
Flink
Samza
Storm
Heron
Apex
Kafka Streams
Gearpump


Github and JIRA
Name / Days
Commits
Stars
Contributors
Age
Issues Created
Issues Resolved/Closed
18000
12000
6000
2250
1500
750
apache/ spark
apache/ hadoop
apache/flink
apache/ storm
apache/ apex-core
apache/ kafka
twitter/heron
apache/ samza
apache/ incubator- gearpump

ENTERPRISE SUPPORT

ENTERPRISE SUPPORT

	Support	Company
Apache Samza	☒	
Twitter Heron	☒	
Apache Gearpump	☒	
Apache Storm	☒	HortonWorks
MapReduce	☑	Cloudera/HortonWorks/MapR
Apache Apex	☑	DataTorrent
Apache Spark	☑	Databricks
Apache Flink	☑	data Artisans
Kafka Streams	☑	Confluent.io

PERFORMANCE ?

SPARK IS FAST

MAPREDUCE IS SLOW

Everything else is not measured

DOING BENCHMARKS IS HARD

- scale the cluster
- scale the dataset
- compare different applications
- implement them optimally
- automate the process

OTHER

- processing model
- data windowing
- state handling
- dynamic scalability
- SQL support
- ML support
- notebooks support

OTHER

- GPU support
- Logging
- Metrics
- History servers
- Licensing

IT'S ALL MEANINGLESS

DOCS VS EXPERIENCE

There are lies, damn lies and documentation

EXPERIENCE THEN?

- 9 technologies
- 6 months for each
- **4.5 years**

2.32X

- 36 trenary features
- Popularity
- Maturity

2.32X

	Features	Popularity	Maturity	Result
Apache Spark	1.0	1.0	0.99	**1.0**
Apache Flink	0.92	0.07	0.31	**0.43**
Hadoop MapReduce	0.6	0.16	0.45	**0.40**
Kafka Streams	0.92	0.04	0.20	**0.39**
Apache Storm	0.81	0.08	0.26	**0.38**
Apache Apex	0.81	0.02	0.13	**0.32**
Tiwtter Heron	0.68	0.01	0.07	**0.25**
Apache Samza	0.60	0.01	0.09	**0.23**
Apache Gearpump	0.42	0.00	0.07	**0.16**

SUMMARY

- Start with **NONE**
- If you really have to, go with Spark
- If you don't want Spark:
  - If you're doing streaming: Flink/Kafka Streams
  - If you're doing batch: Flink/MapReduce
- Samza, Storm, Heron, Gearmpump, Apex are alive
- Kafka Streams has unique deployment
- Doing comparisons is hard

SUBJECTIVE SUMMARY

- mostly false/inaccurate
- based on my own judgement
- short and simple

APACHE SPARK

Most popular, most advanced, first choice for data processing.
Not going anywhere.

HADOOP MAPREDUCE

Old, simple, quite reliable, slow and verbose.

APACHE FLINK

Similar to Spark but with direct message processing and better windowing.

APACHE STORM

First choice for stream processing before Flink/Spark, now losing popularity. Not-so-great API and doesn't work with YARN.

TWITTER HERON

Storm with better deployment(can run on YARN) and without Trident.

KAFKA STREAMS

Very nice deployment scheme(just a library), limited to Kafka, quite fancy.

APACHE APEX

Good for authors, not popular enough, quite sophisticated but looks over engineered.

APACHE SAMZA

Simple, tightly coupled with Kafka na Yarn, based on properties files.

APACHE GEARPUMP

Still incubating without any killer features.

THANKS

Wojciech Pituła

@Krever01

w.pitula.me/presentations