FOSSCOM 2016

Big Data Streaming processing using Apache Storm

CC BY SA

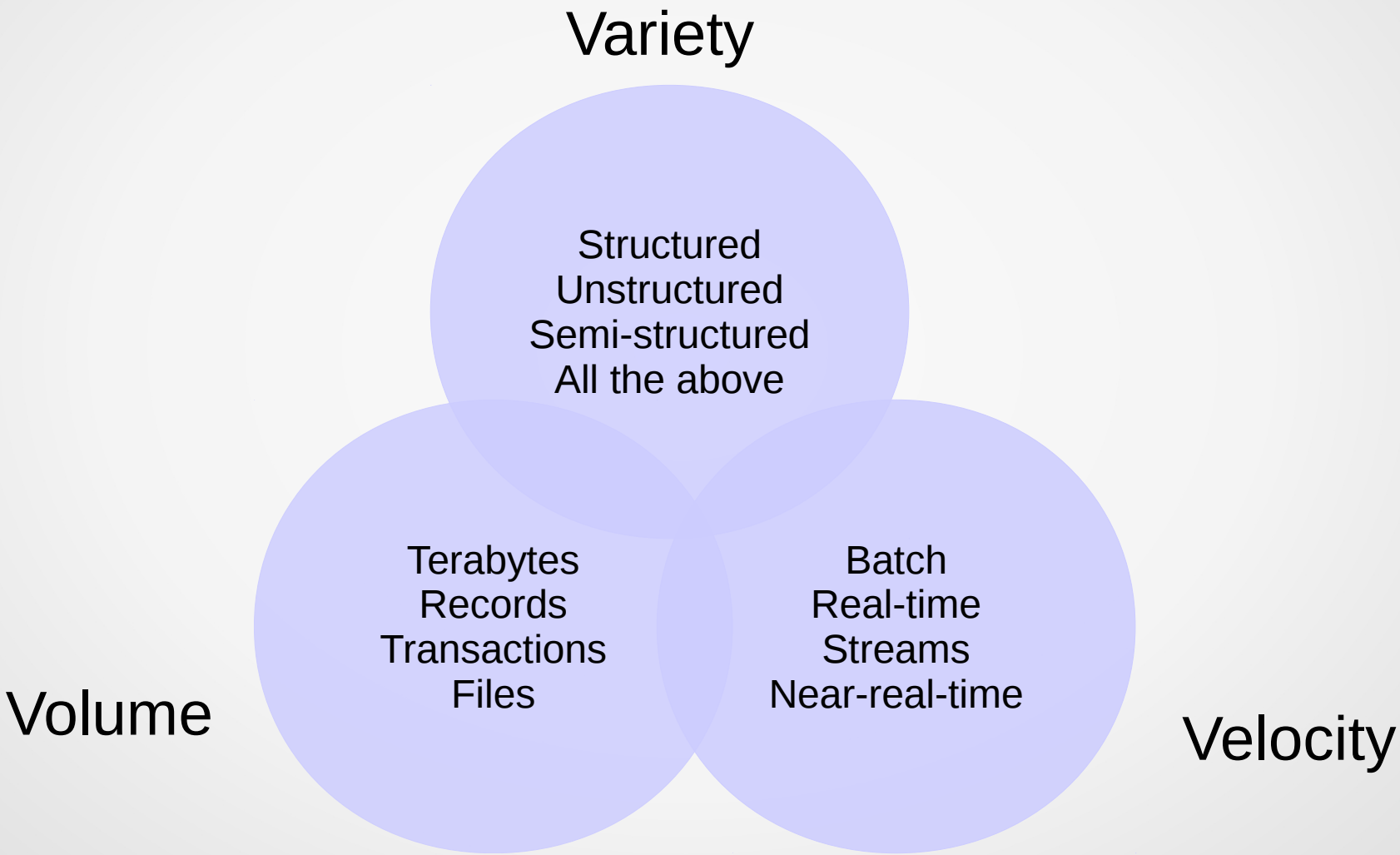
Who we are?

- Adrianos Dadis (@qiozas)
- Patroclos Christou
- Eleftheria Chavelia
- Sofia Nomikou

Agenda

- Main Big Data concepts
- Streaming processing and NoSQL persistence
- Introduction to Apache Storm and Apache Kafka
- Big Data streaming application demo
- Considerations for Big Data applications

Big Data characteristics



Use cases

- 360-degree customer view
- Internet of Things
- Data warehouse optimization
- Information security
- Sentiment analysis
- Urban Planning (MIT)

Big Data

- Pros
  - Better user experience
  - Predictive analytics
  - Answer complex business questions
- Cons
  - Privacy
  - Strict personalized content

Database issues

- Relational Databases provide benefits
- Relational Databases pitfalls:
  - Application development productivity
  - Large-scale data
    - Capture more data
    - Process data faster
    - Costs

NoSQL

- Schema agnostic
- Nonrelational
- Commodity hardware
- Highly distributable

- NOT a silver bullet

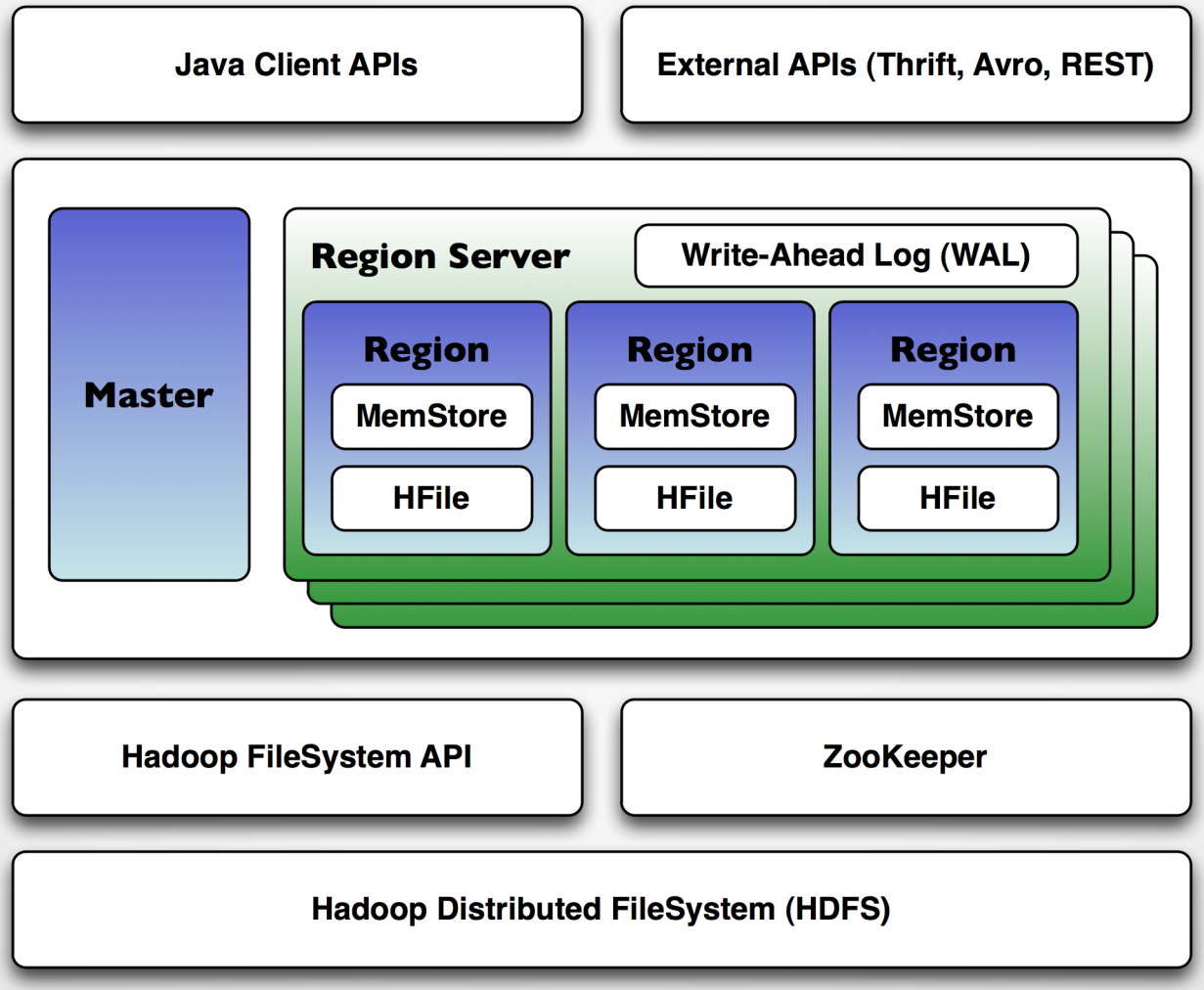
NoSQL & CAP theorem

- Consistency
  - all nodes see the same data at the same time
- Availability
  - a guarantee that every request receives a response about whether it succeeded or failed
- Partition tolerance
  - the system continues to operate despite arbitrary partitioning due to network failures

NoSQL Data Models

- Key Value
  - Riak, Redis, Aerospike
- Document
  - MongoDB, Elasticsearch
- Column Family
  - Cassandra, HBase
- Graph
  - Neo4J, ArangoDB, OrientDB

Apache HBase



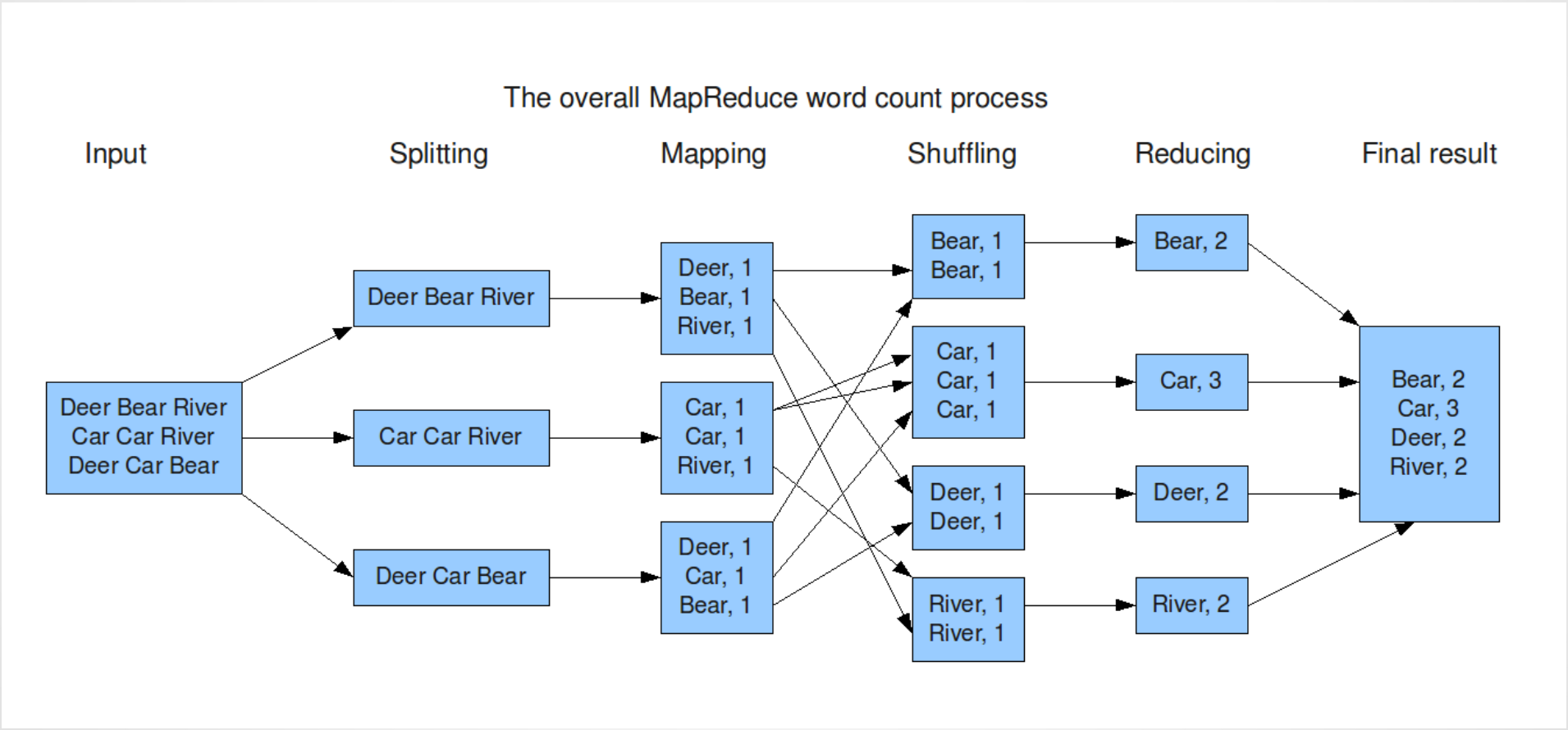
Big Data processing modes

- Batch
  - Long running jobs
- Streaming processing
  - One-at-a-time
  - Micro batch

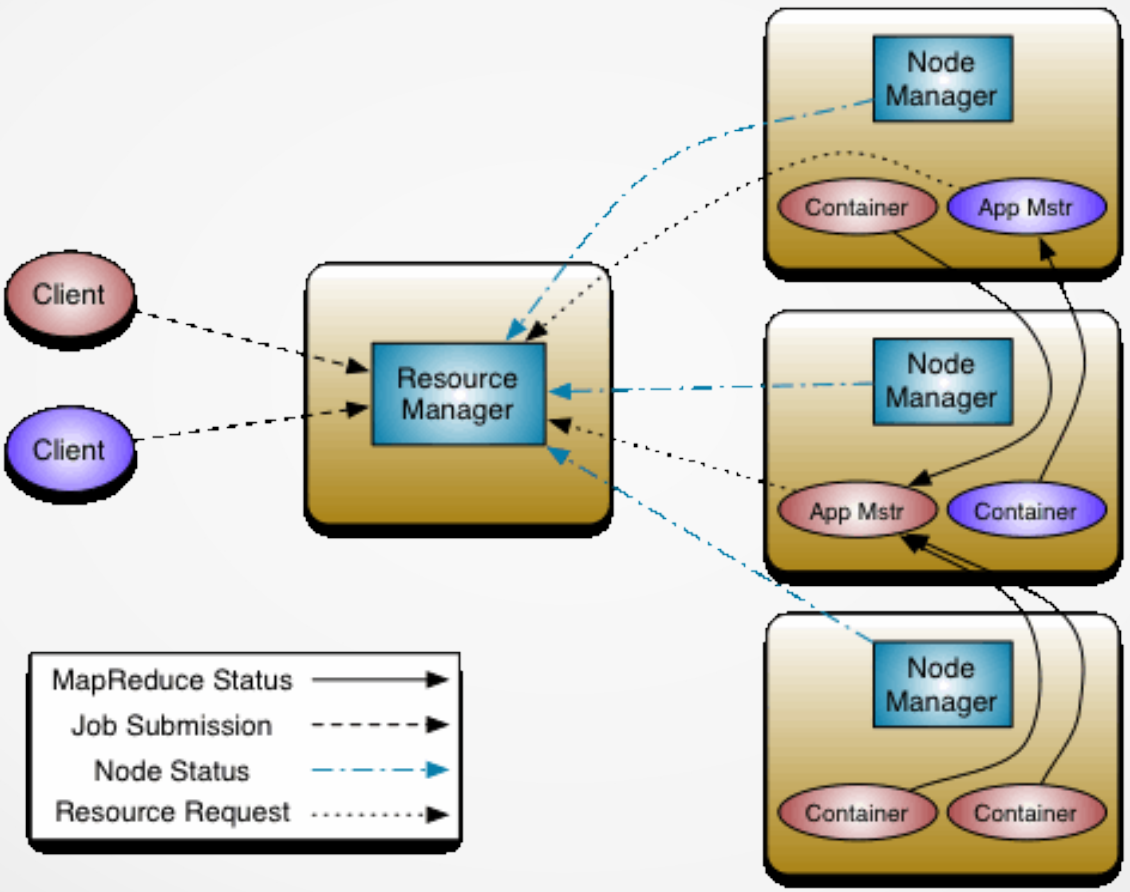
Map Reduce

- Apache Hadoop
- Programming/Processing model
- Steps:
  - Map
  - Shuffle
  - Reduce
- HDFS
- Apache Hadoop ecosystem
  - Pig
  - Hive
  - *more*

Map Reduce – word count



YARN


Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

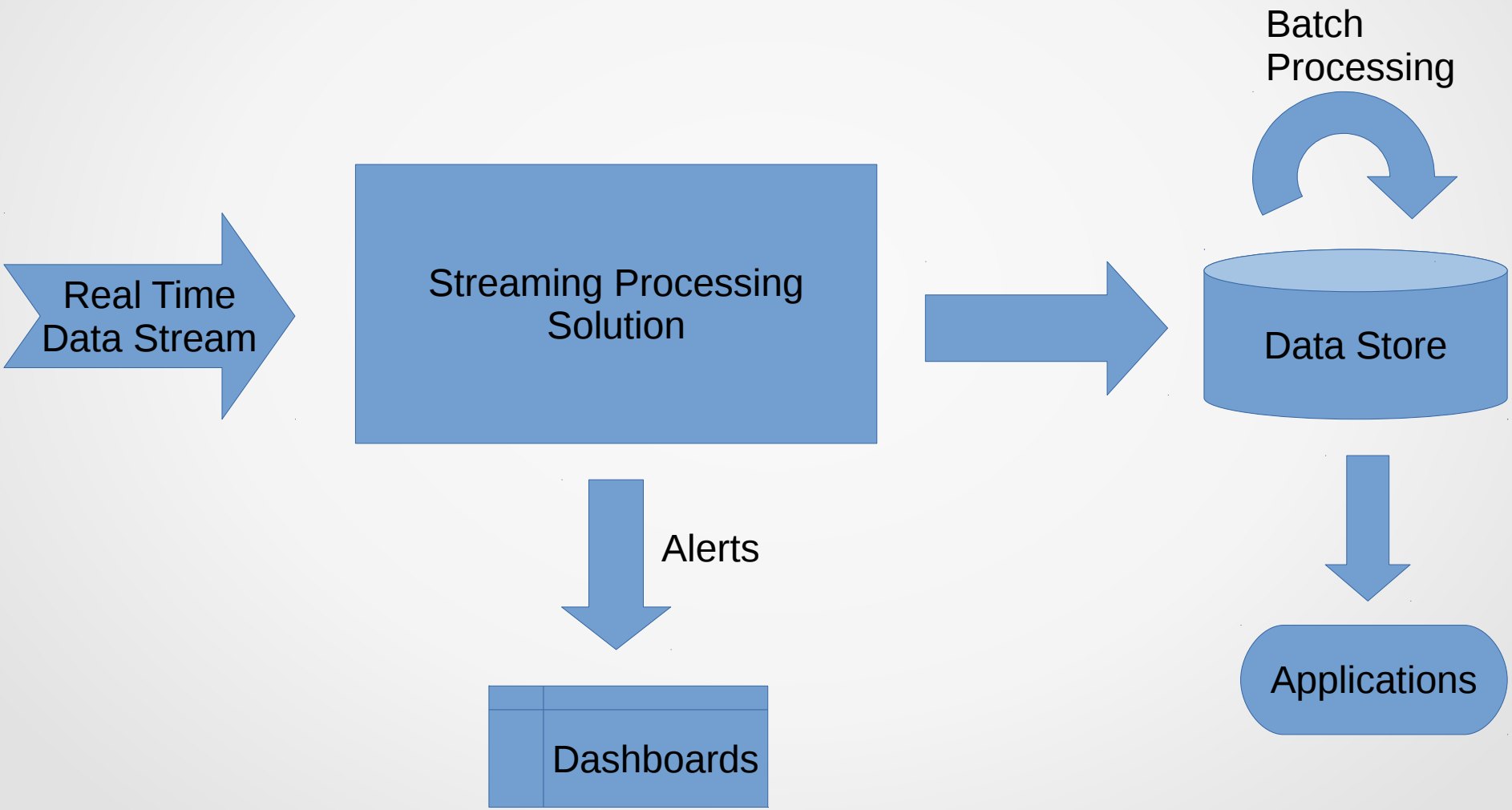
Apache Spark (batch processing)

- Similar to MapReduce but faster
- Resilient Distributed Dataset
- Ecosystem
  - Spark SQL
  - MLlib
  - GraphX
  - Streaming
  - Scala / R / Python / Java / SQL

Streaming processing Use Cases

- Prevent security fraud
- Optimize order routing
- Predictive maintenance
- Optimize personalized content
- Optimize prices or offers
- Prevent stock outs
- Optimize bandwith allocation
- Prevent application failures

Streaming processing



Streaming processing frameworks

- Apache Storm
  - one-at-a-time processing (pure Storm)
  - micro batch (Trident)
- Apache Spark Streaming
  - micro batch
- Apache Flink
  - one-at-a-time processing
  - micro batch

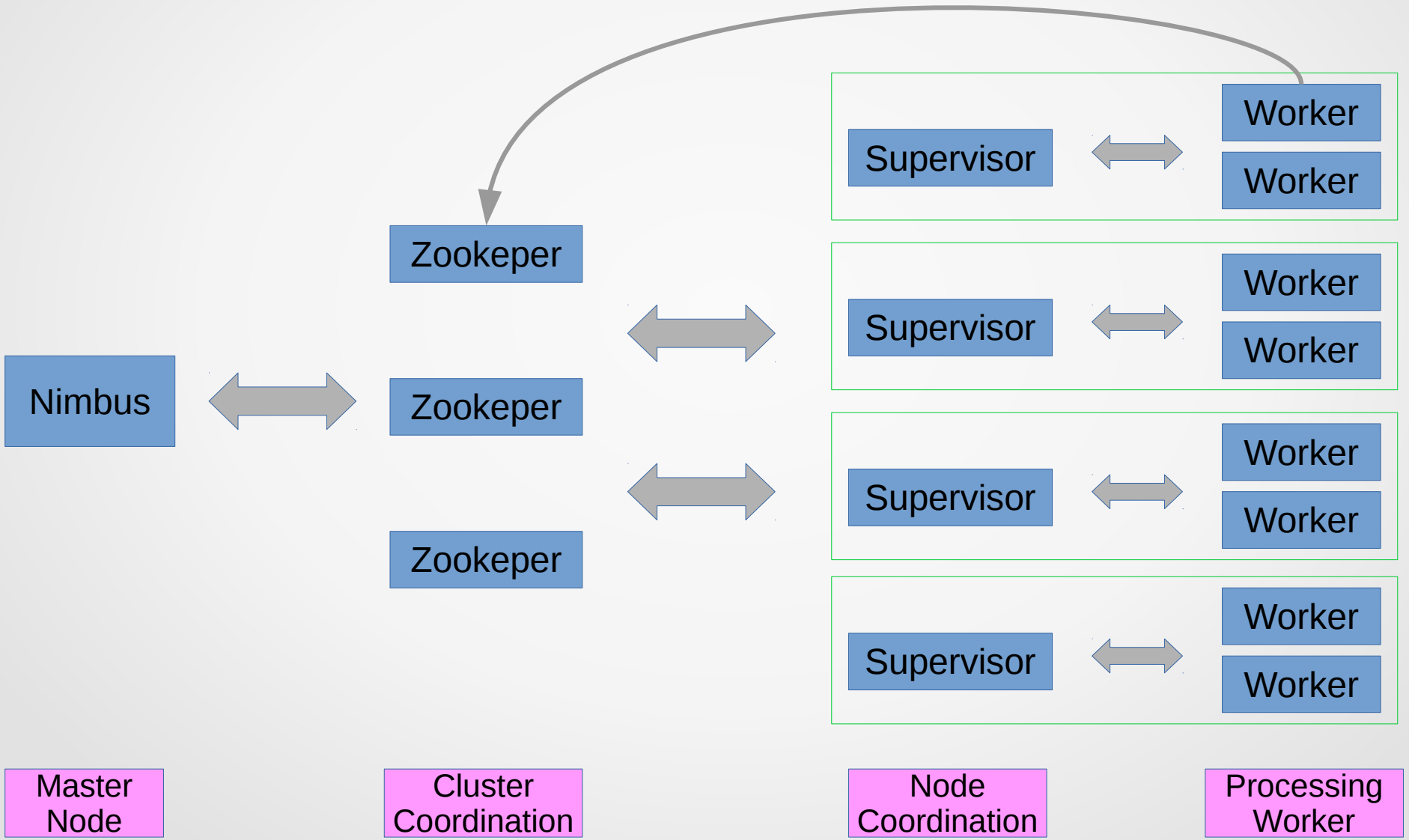
Apache Storm

- Creator: Nathan Marz
- Distributed real-time computation system for processing large volumes of high-velocity data
- Most popular engine in industry
- Characteristics:
  - Fast
  - Scalable
  - Fault-tolerant
  - Reliable
  - Easy to operate

Storm modes

- One-at-a-time processing
  - Very low latency
  - Very simple development model
  - At-Most-Once and At-Least-Once semantics
- Micro batch processing (Trident)
  - Better throughput for large rates
  - Increased latency for each event
  - More complex development
  - Most efficient than simple Bolts model for particular use cases.
  - Exactly-Once semantics (per batch)

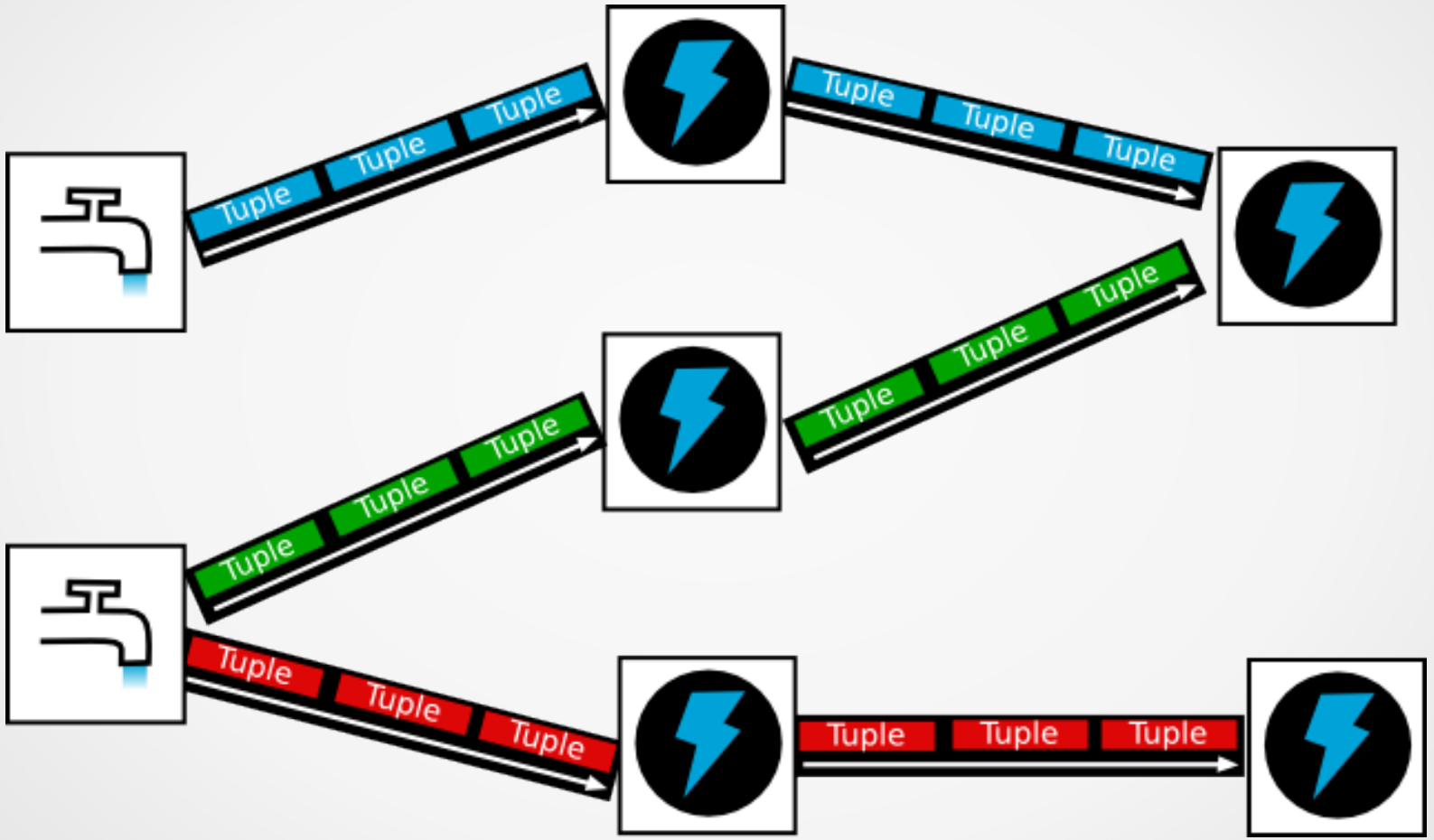
Storm Architecture



Storm concepts

- **Topology** : A graph of computation
- **Tuple** : Storm uses tuples as its data model
- **Stream** : An unbounded sequence of tuples
- **Spout** : A source of streams in a topology
- **Bolt** : All processing in topologies is done in bolts
- **Stream Grouping** : Defines how that stream should be partitioned among the bolt's tasks.

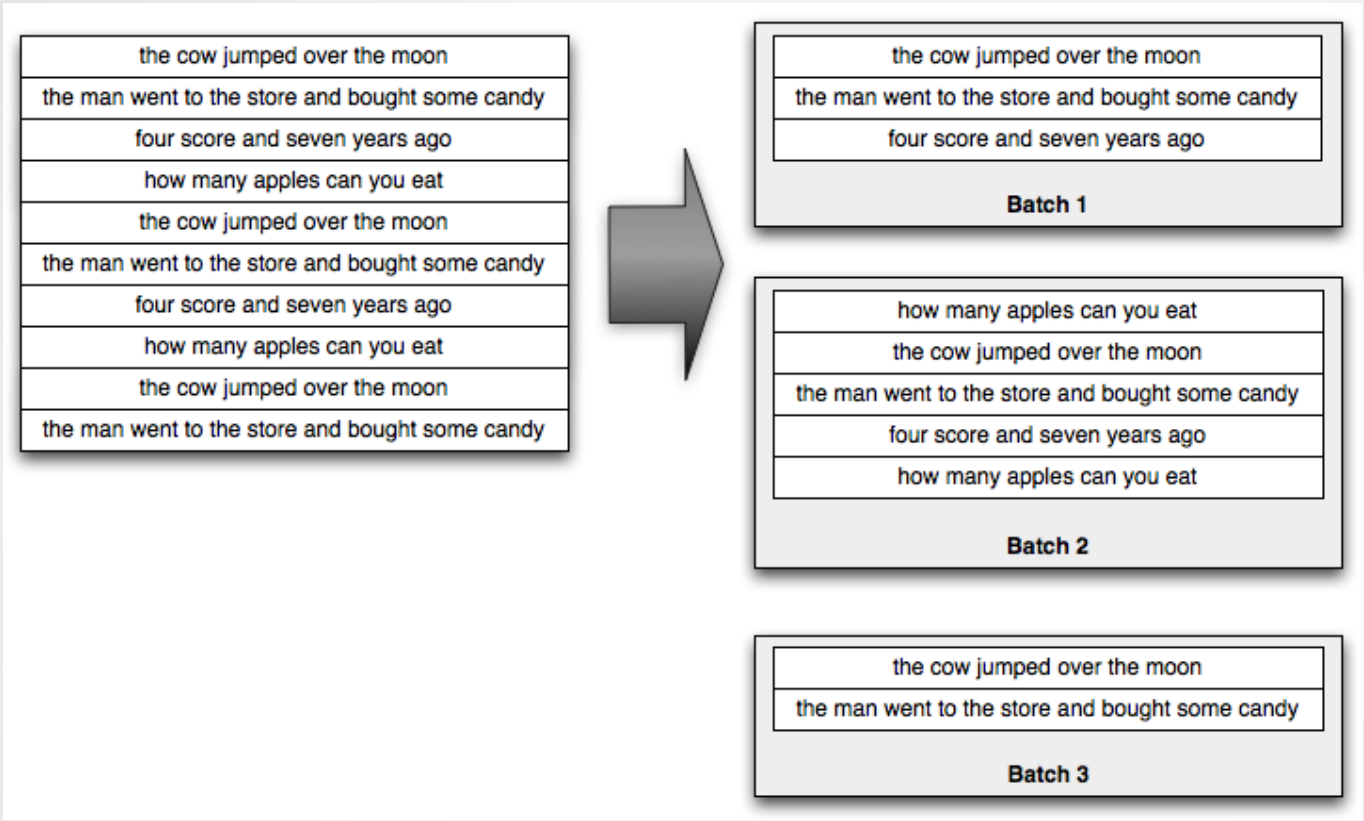
Storm topology



Storm topology example

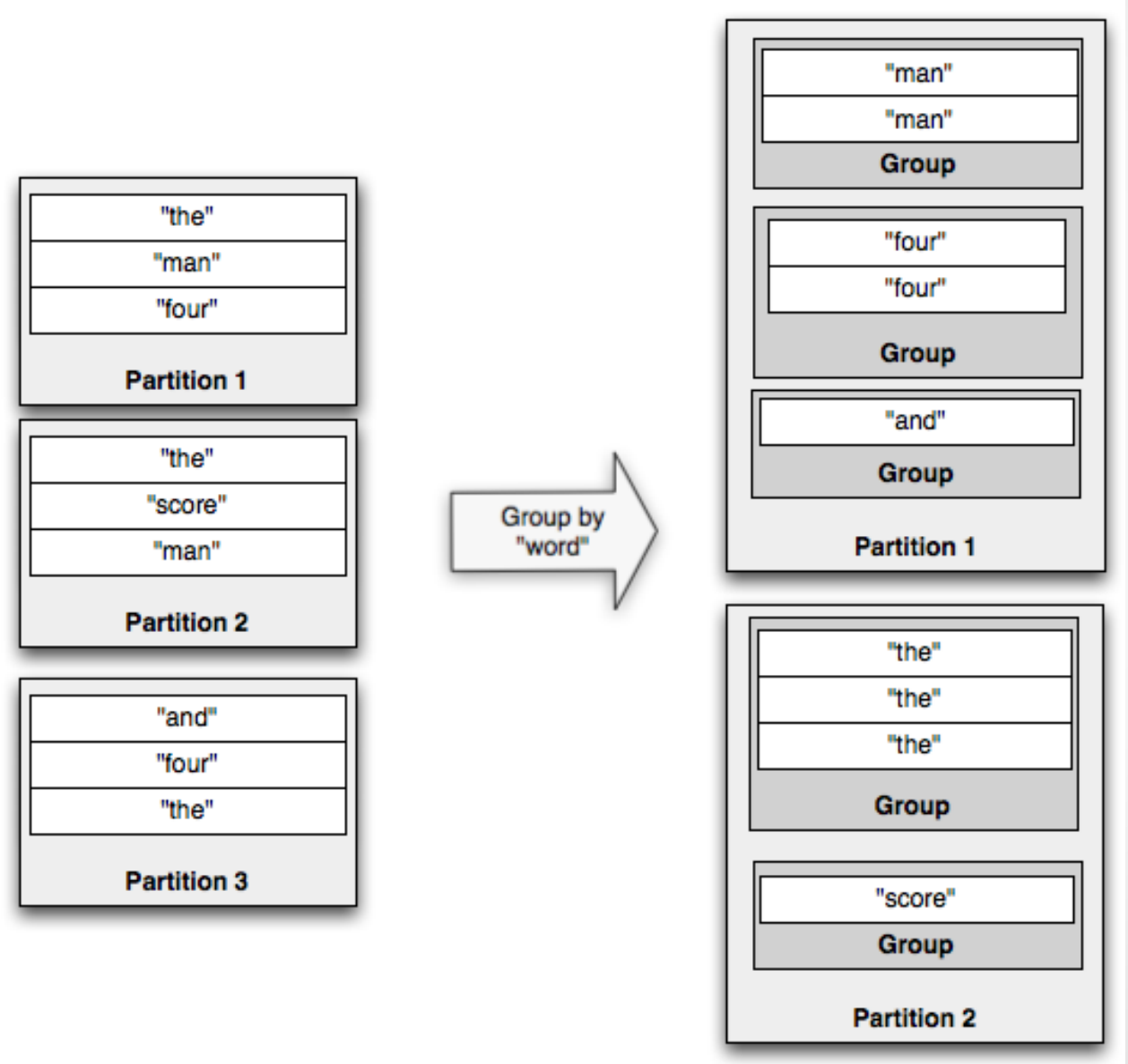
```
Config conf = new Config();
conf.setNumWorkers(2); // use two worker processes

topologyBuilder.setSpout("blue-spout", new BlueSpout(), 2); // set paral
lelism hint to 2

topologyBuilder.setBolt("green-bolt", new GreenBolt(), 2)
               .setNumTasks(4)
               .shuffleGrouping("blue-spout");

topologyBuilder.setBolt("yellow-bolt", new YellowBolt(), 6)
               .shuffleGrouping("green-bolt");

StormSubmitter.submitTopology(
        "mytopology",
        conf,
        topologyBuilder.createTopology()
    );
```

Storm Trident

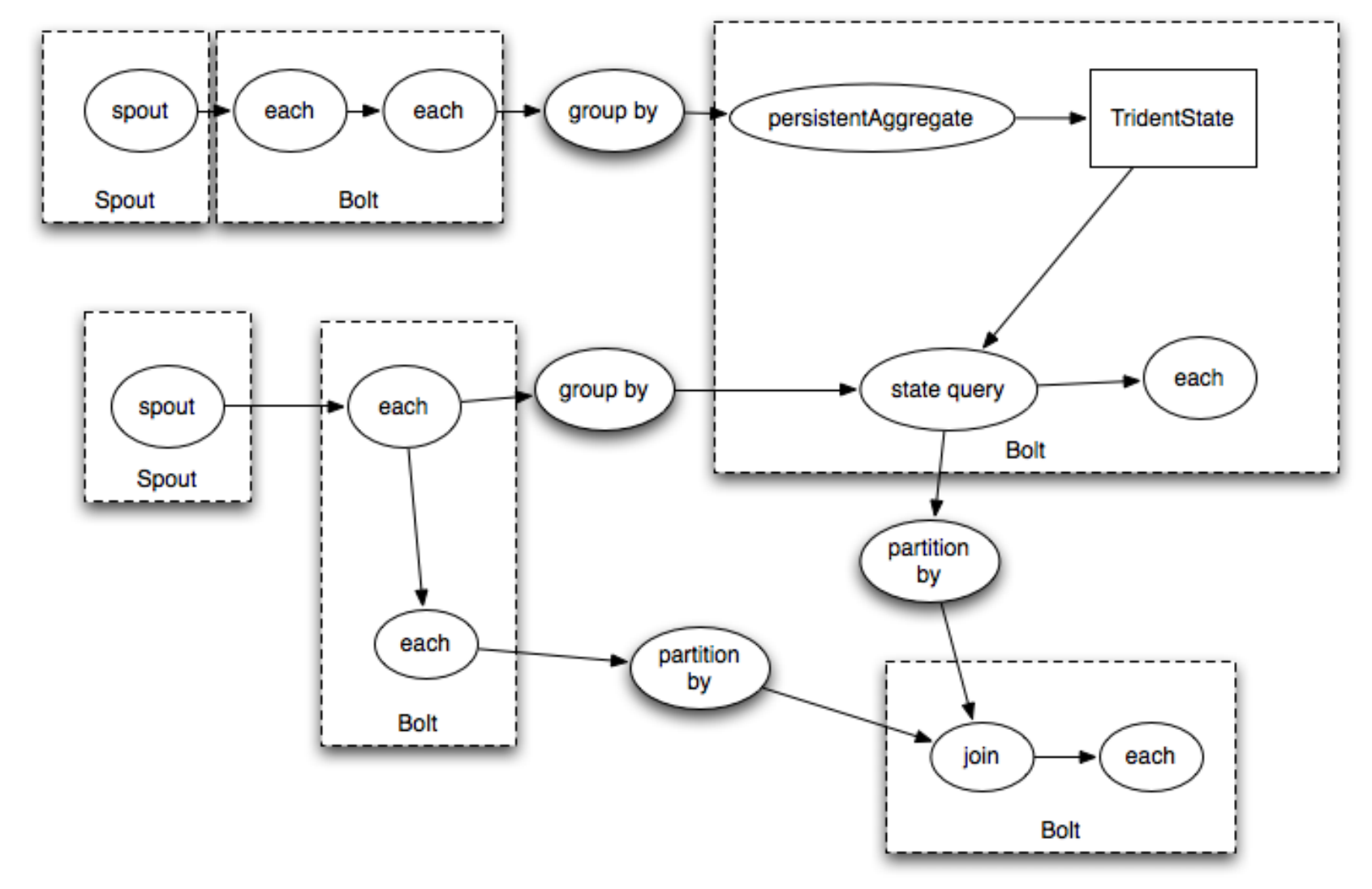



```
TridentTopology topology = new TridentTopology();
TridentState wordCounts =
     topology.newStream("spout1", spout)
       .each(new Fields("sentence"), new Split(), new Fields("word"))
       .groupBy(new Fields("word"))
       .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
       .parallelismHint(6);
```

Storm Trident partitioning



Storm Trident

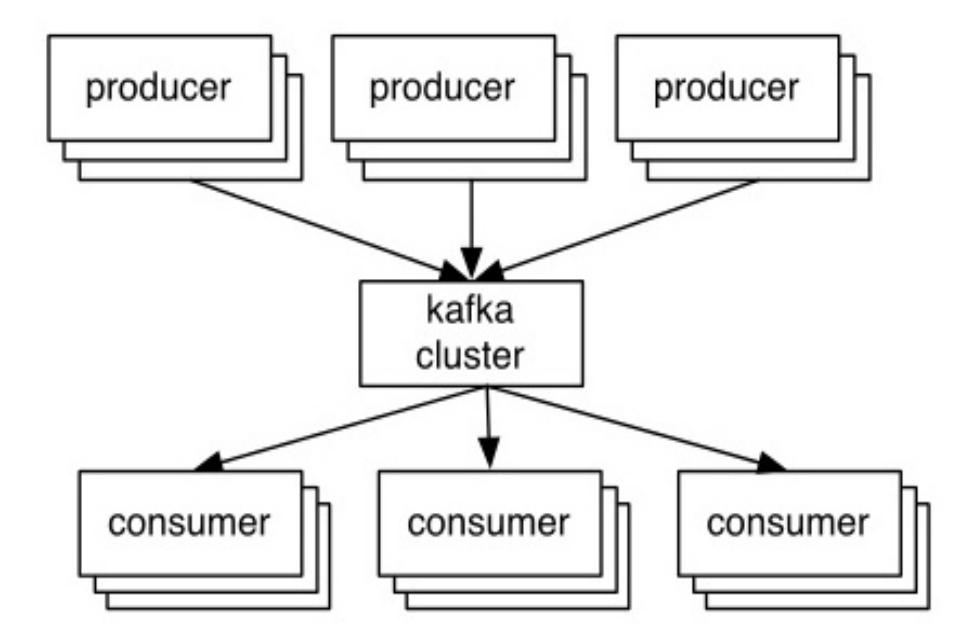


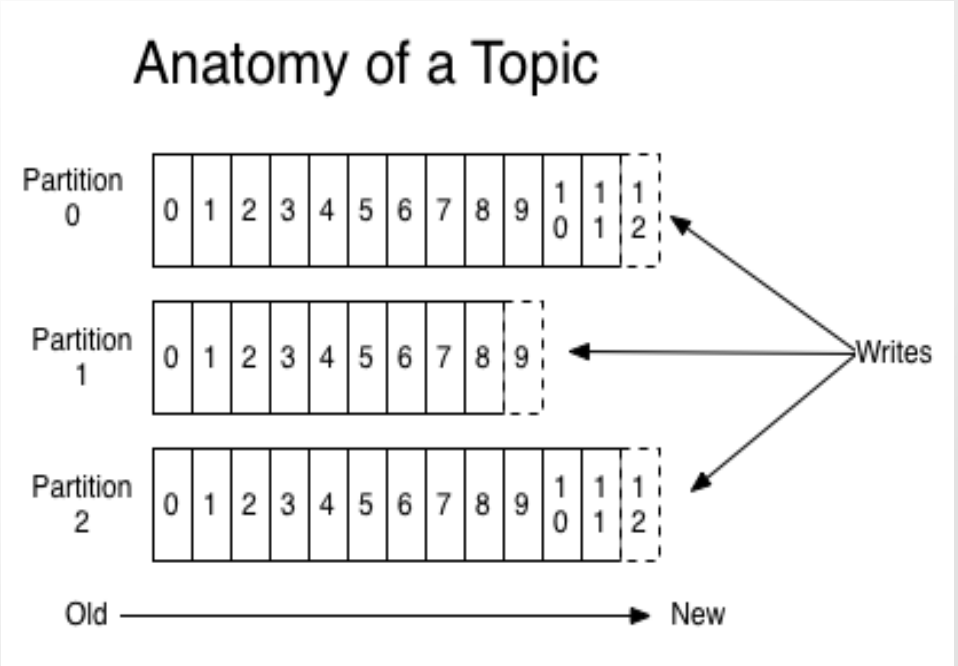
Messaging Systems

- Models: queuing & publish-subscribe
- Decouple processing from data producers
- Buffer unprocessed messages
- Frameworks
  - Kafka
  - RabbitMQ
  - ActiveMQ

Apache Kafka

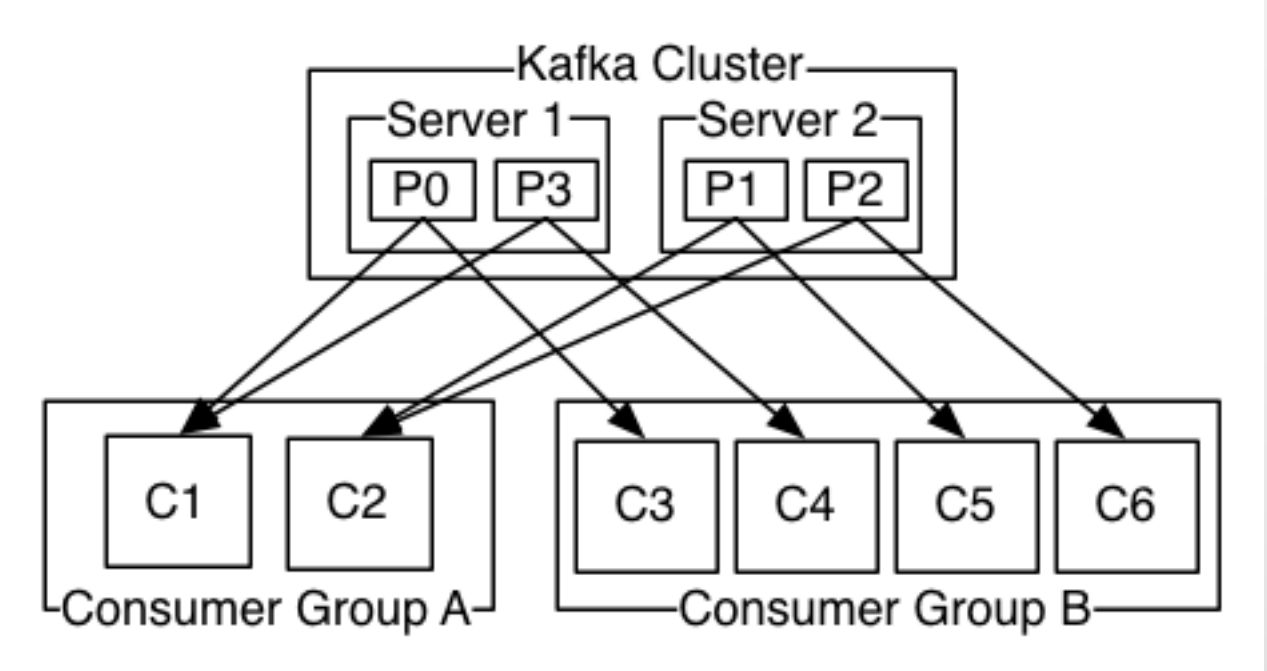
- Distributed, partitioned, replicated commit log service
- Maintains feeds of messages in Topics
- Publish-Subscribe model





Apache Kafka

- Consumers coordinate what to read
- Consumer & Consumer Group
- Producers
- Brokers



Demo Architecture

- TODOs

Big Data Apps common problems

- Integrate many input sources
- Performance & Scalability
- Complex infrastructure
- Hard to debug
- Difficult to select correct frameworks

Big Data Apps hints

- Failure (HW or data) is a normal case
- Design for scalability from day one
- Queries drive schema design
- Continuous Integration
- Metrics from day one
- Performance tests (real data)
- Monitor
- Appropriate people

Thank YOU :-)

Questions ???