DECK36

Storm w/ Node.js

enter.js 2014

About Stefan & Mike

Dr. Stefan Schadwinkel

Co-Founder / Analytics Engineer

stefan.schadwinkel@deck36.de

@steschas

Mike Lohmann

Co-Founder / Software Engineer

mike.lohmann@deck36.de

@mikelohmann

DECK36

- DECK36 is a young spin-off from ICANS
- Small team of 7 engineers
- Longstanding expertise in designing, implementing and operating complex web systems
- Developing own data intelligence-focused tools and web services
- Offering our expert knowledge in:
  - Automation & Operations
  - Architecture & Engineering
  - Analytics & Data Logistics

WE’RE HERE!

What will be going on today?

WE'RE HERE

Roadmap - Part 01 - 90 Minutes

Introduction

- WTF?

Storm

- An introduction
- Storm —> Node.js

Plug and Play

- Probably the best kitten game ever invented: Plan 9 From Outer Kitten
- Project review

Coffee, finally!

WE'RE HERE

Roadmap - Part 02 - 90 Minutes

Badge Development

- Your own Storm-based backend module!

Deploy

- Local
- Remote "Cloud"

WHY STORM?

What’s all the fuzz about?

WHY STORM?
It's like Hadoop, but for Real-Time!

Hadoop is cutting-edge and old-school at the same time

- Relatively low / early stage adoption of Hadoop in Germany
- Hadoop is becoming Enterprise Software and is evolving (YARN, MRv2)
- Batch-processing becomes increasingly painful for the business operations
- Storm enables new ways to approach business problems

Before Storm

- Real-time processing using a network of queues and workers
- Lot's of config, diversity, complicated fault-tolerance, unintuitive to scale
- Hard to reason about system state, hard to fix failures

WHY STORM?
It's like Hadoop, but for Real-Time!

Storm to the Rescue!

- Abstraction of a queue/worker network

- Like Hadoop MapReduce, but for message streams
  - few core primitives, programming-language agnostic

- Easy to scale, just add machines and increase parallelism settings
- Strong guarantees: each message is processed ("exactly once" possible w/ Trident)
- Explicit project goal: painless cluster management
- Fault-tolerant: tasks will be reassigned if parts break, computation is always continuous

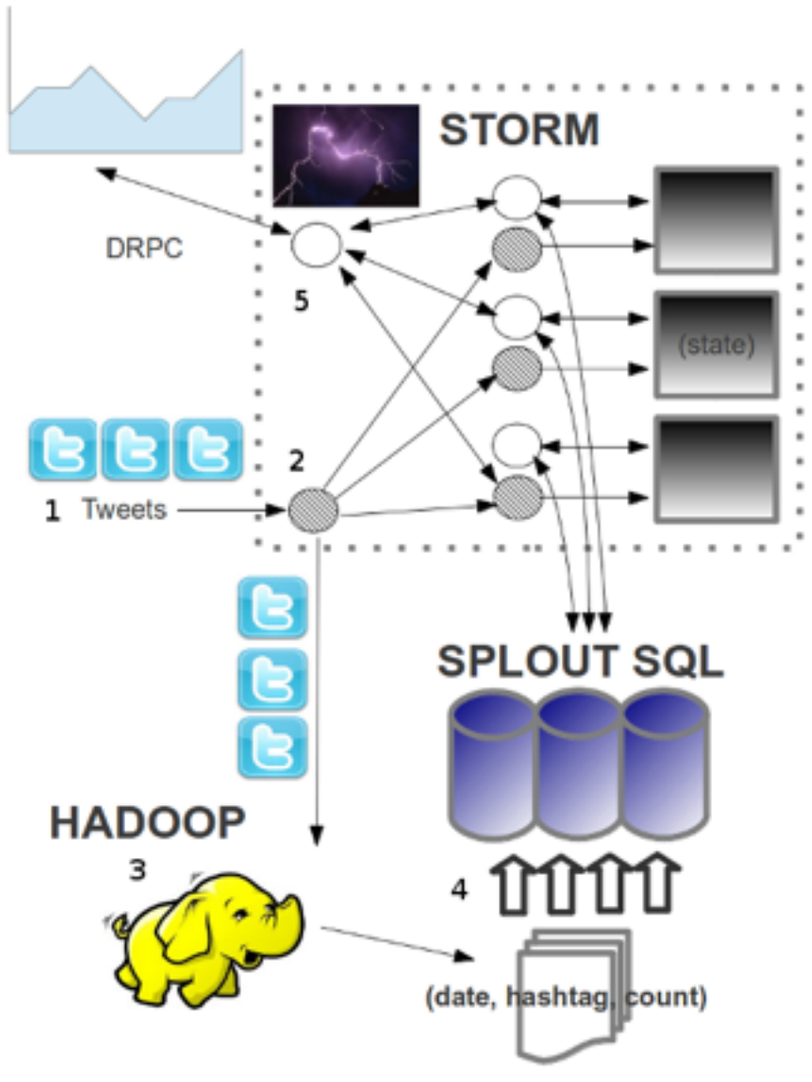
SOLUTION ARCHITECTURE
Lambda Architecture

The Real-Time Big Data Architecture

Named by Nathan Marz, the creator of Storm. http://lambda-architecture.net/

Example: Count tweets per #hashtag

1. Tweets come in from Persistent Queue
2. Trident Topology to (A) save all data to Hadoop & (B) update Trident State with counts for current day
3. Trident Topology triggers Hadoop to compute the counts on “all” raw data
4. Push the Hadoop result to database
5. Trident Timeline Query via DRPC



http://www.datasalt.com/2013/01/an-example-lambda-architecture-using-trident-hadoop-and-splout-sql/
https://github.com/pereferrera/trident-lambda-splout

COMPETITION?

Storm vs. S4 vs. Samza

Yahoo Samoa

Apache Spark Streaming

Amazon Kinesis

Google BigQuery for Streams

CONCEPTS

Let’s view the ontological garden.

CONCEPTS
Tuple

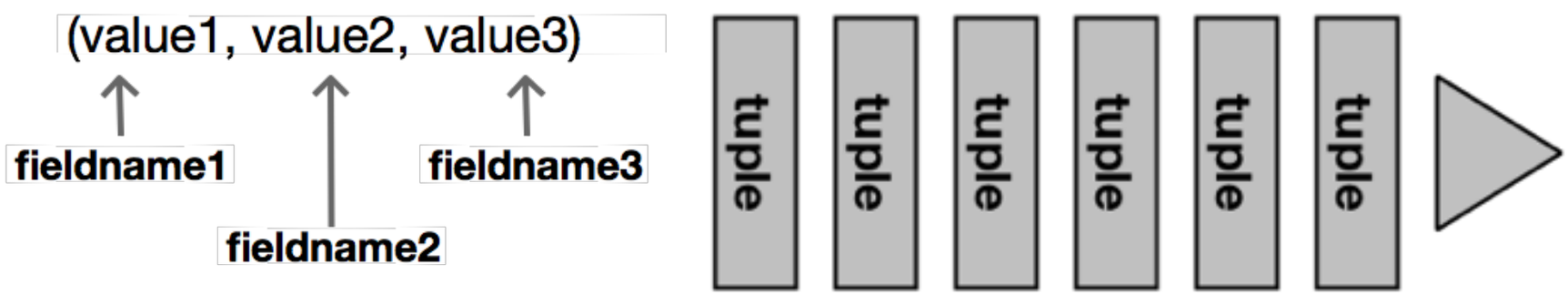
A tuple is one single “message”

- List (Java ArrayList to be precise) of Objects (= different data)
- A tuple has a “schema” known to storm, but not part of the message

CONCEPTS
Stream

A “stream” is an unbounded sequence of tuples

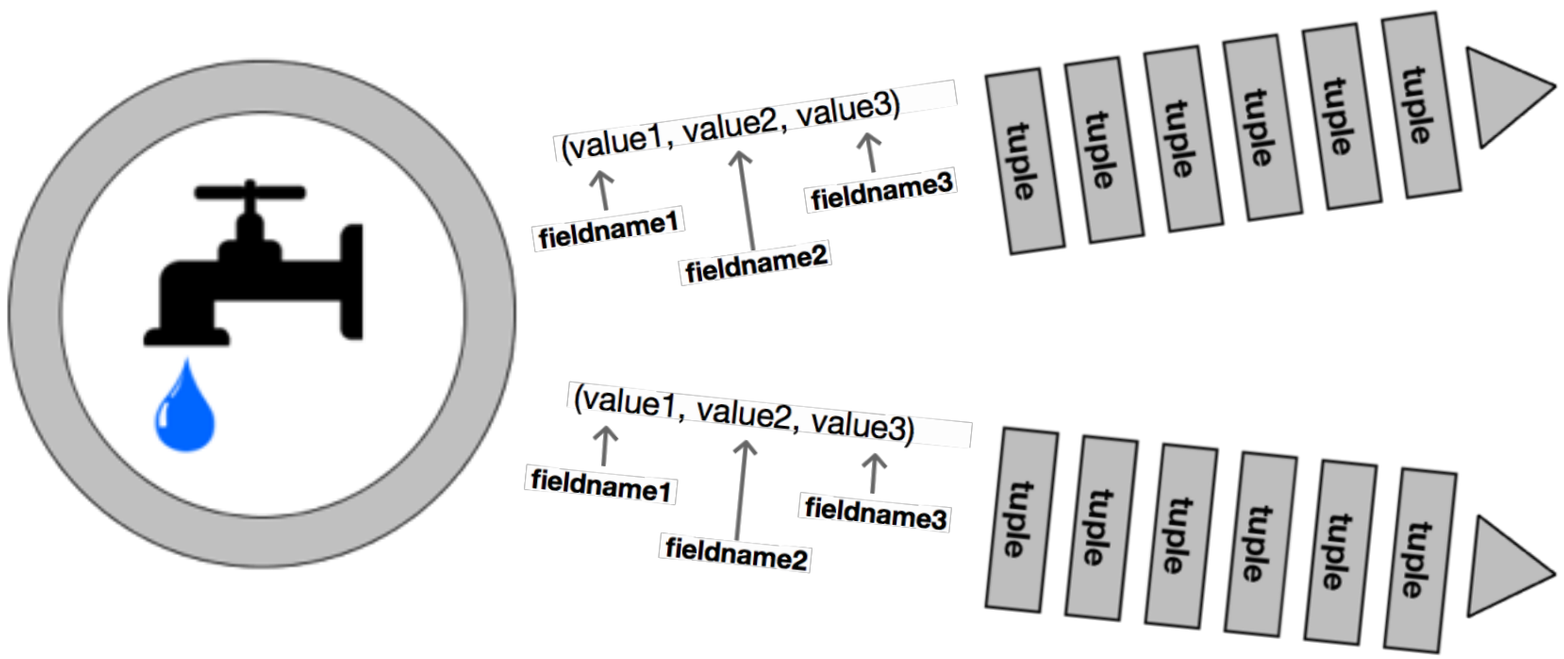
- The “stream” is the core abstraction construct within storm
- “Storm transforms streams into new streams”
- Storm components can consume and produce multiple different streams
- Tuples in a single stream can be handled differently based on a key
- All tuples in a single stream must follow the same “schema”



CONCEPTS
Spout

A spout is a source of streams

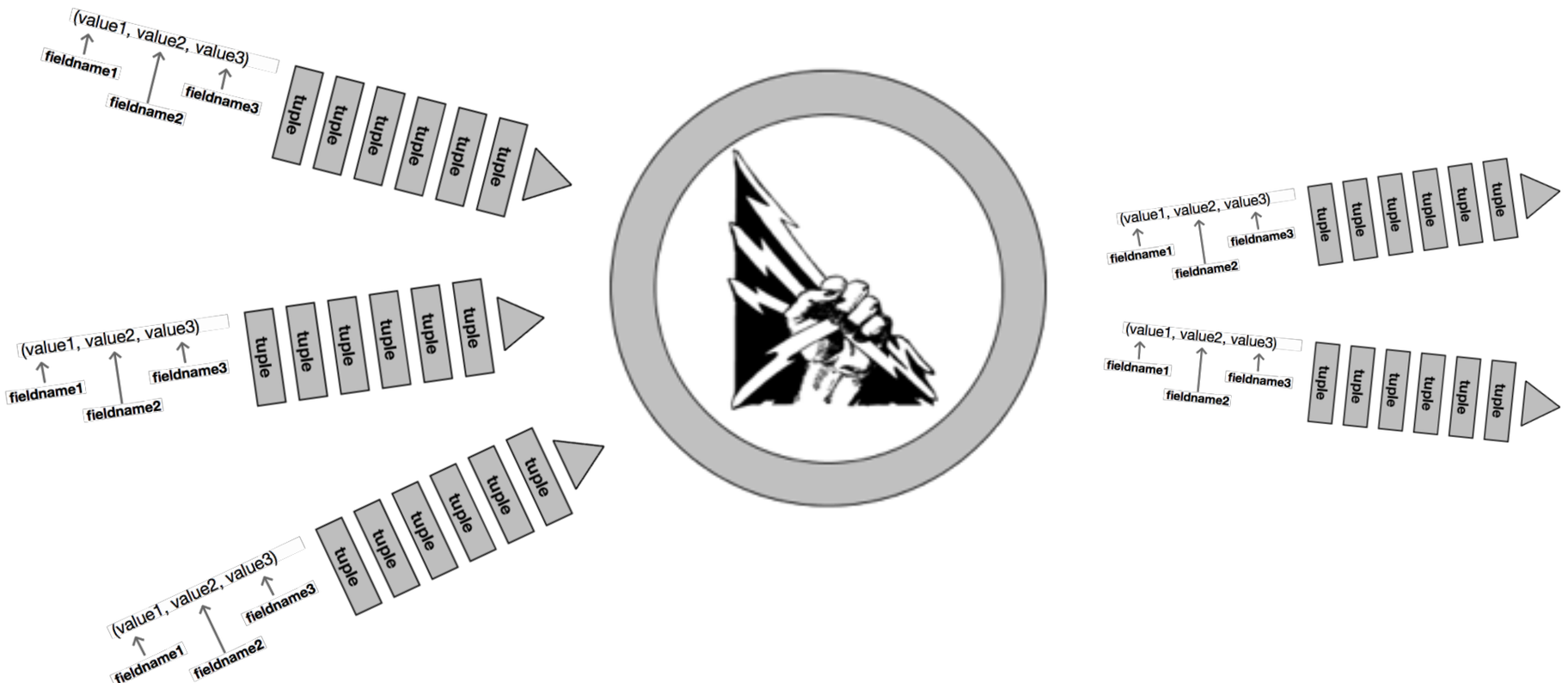
- A spout is one of the core components of storm
- Spouts can be implemented in PHP, node.js, etc. We'll do that later.
- Data comes from an outside infinite source, not triggered by storm events
- A single spout can emit multiple streams with different schemas



CONCEPTS
Bolt

A bolt is a worker process

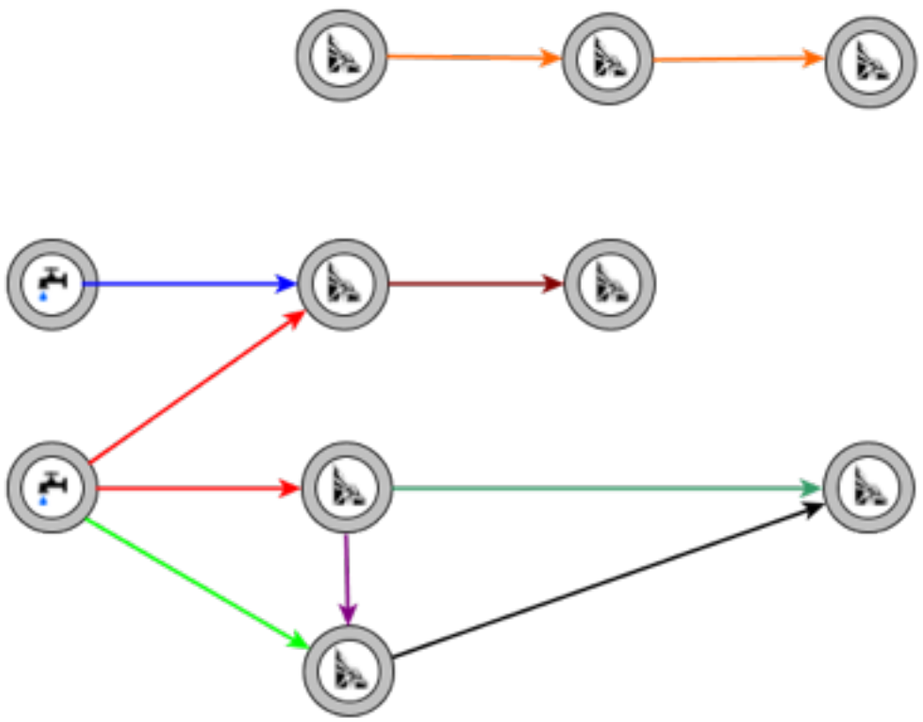
- A bolt is the second core component of storm
- We'll also implement bolts later.
- A bolt consumes one or more streams. These can use multiple schemas.
- A bolt emits zero or more streams.
- A bolt can produce side effects. For instance, a database update.



CONCEPTS
Topology

A topology is a network comprised of spouts and bolts.

- A topology represents a “storm program”.
- Storm cluster executes and manages multiple topologies.
- All code for one topology is packaged into a single file.
- The graph representing a single topology can be partitioned.



CONCEPTS
Stream Grouping

Stream Groupings partition *one* stream among the tasks of *one* target bolt.

- Shuffle grouping: Random, each task gets the amount of tuples.
- Fields grouping: Fields from a tuple can be used as key. Tuples with the same key, will always go to the same task.
- All grouping: Every tuple goes to every task. “Use this grouping with care.”

Bare metal:

- Global grouping: All tuples only go to one task, the task with the lowest id.
- None grouping: Undefined, but like shuffle. Reserved for future task management.
- Direct grouping: The producer of the tuples decided which task gets the tuple.
- Local or shuffle grouping: If the target bolt has tasks in the same worker process, only those tasks get the tuples. Otherwise, it behaves like normal shuffle grouping.

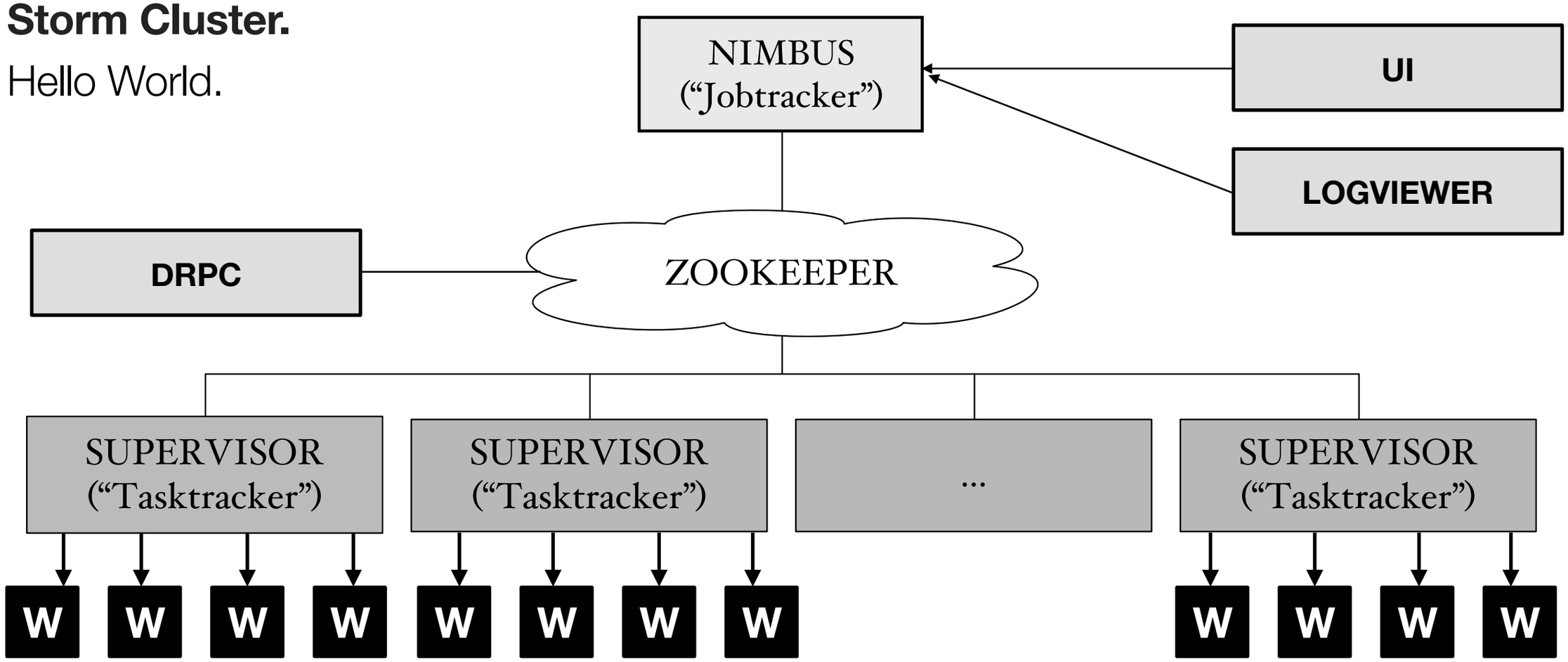
CONJURING DAEMONS

Storm Cluster Setup

CONJURING DAEMONS
Overview

Storm Cluster.

Hello World.



STORM

The Nuts and Bolts

SPOUT; BOLT

Java

```java
public class JavaSpout extends BaseRichSpout {

    // private data, constructor, etc.

    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {

        // Setup output collector
        _collector = collector;
    }

    @Override
    public void nextTuple() {

        _collector.emit(new Values(value1, value2, value3));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {

        // use default stream
        declarer.declare(new Fields("fieldname1", "fieldname2", "fieldname3"));

        // declare or more explicit streams
        declarer.declareStream("stream_name", new Fields("fieldname1", "fieldname2", "fieldname3"));
    }

    @Override
    public void cleanup() {
    }

}
```

```java
public class JavaBolt extends BaseRichBolt {

    // private data, constructor, etc.

    @Override
    public void prepare(Map stormConf, TopologyContext context, OutputCollector collector) {

        // Setup output collector
        _collector = collector;
    }


    @Override
    public void execute(Tuple tuple) {

        // anchor (or not) the emitted tuple to the input tuple
        _collector.emit(/* tuple, */ new Values(value1, value2, value3));
        _collector.ack(tuple);
    }


    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {

        // use default stream
        declarer.declare(new Fields("fieldname1", "fieldname2", "fieldname3"));

        // declare or more explicit streams
        declarer.declareStream("stream_name", new Fields("fieldname1", "fieldname2", "fieldname3"));
    }

    @Override
    public void cleanup() {
    }

}
```

SPOUT; BOLT

node.js

```
var Spout = require('./spout').Spout;

//The Spout constructor takes a definition function,
//an input stream and an output stream
var itemEmitter = new Spout(function(events) {
    var collector = null;
    var seqId = 1;

    //You can listen to the spout "open", "ack" and "fail" events to
    events.on('open', function(c) {
        collector = c;
    });

    events.on('ack', function(messageId) {});

    events.on('fail', function(messageId) {});

    //The definition function must return a function used as
    //the nextTuple function.
    return function(cb) {

        collector.emit([value1, value2, value3], seqId++);
        cb();

    }

}, process.stdin, process.stdout);


process.stdin.setEncoding('utf8');
process.stdin.resume();
```

```
var Bolt = require('./bolt').Bolt;

//The Bolt constructor takes a definition function,
//an input stream and an output stream
var joinBolt = new Bolt(function(events) {
    var collector = null;

    //You can listen to the bolt "prepare" event to
    //fetch a reference to the OutputCollector instance
    events.on('prepare', function(c) {
        collector = c;
    });

    //The definition function must return a function used as
    //the execute function.
    return function(tuple, cb) {

        collector.emit(tuple.values.join(', '));
        collector.ack(tuple);
        cb();
    }

}, process.stdin, process.stdout);

process.stdin.setEncoding('utf8');
process.stdin.resume();
```

TOPOLOGY

Java

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// To launch the topology, we need a Config object. This config is forwarded into every component of the topology.

Config conf = new Config();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// The topology is created through using the topology builder.

TopologyBuilder builder = new TopologyBuilder();


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Now, we can add the components (spouts and bolts) to the topology

int parallelism_hint = 1;

// Random Java Spout: Emits random names.
builder.setSpout("random_java_spout", new RandomJavaSpout(), parallelism_hint);

// Random PHP Spout: Emits random verbs.
builder.setSpout("random_php_spout", new MultilangAdapterSpout("/usr/bin/php", "RandomPHPSpout.php", "", "src", "random"), parallelism_hint);

// Random Node.js Spout: Emits random items.
builder.setSpout("random_nodejs_spout", new MultilangAdapterSpout("/usr/local/bin/node", "RandomNodeJSSpout.js", "", "src", "random"), parallelism_hint);
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Empty (printer) bolts (java + multilang)

builder.setBolt("empty_java_bolt", new EmptyJavaBolt(), parallelism_hint)
        .setNumTasks(4)
        .shuffleGrouping("random_java_spout")
        .shuffleGrouping("random_php_spout")
        .shuffleGrouping("random_nodejs_spout");


builder.setBolt("empty_php_bolt", new MultilangAdapterBolt("/usr/bin/php", "EmptyPHPBolt.php", "", "src", "random"), parallelism_hint)
        .setNumTasks(4)
        .shuffleGrouping("random_java_spout")
        .shuffleGrouping("random_php_spout")
        .shuffleGrouping("random_nodejs_spout");


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Build and submit the topology to the cluster

StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
```

RELIABILITY

Guaranteed Message Processing

Storm guarantees that each message coming off a spout will be fully processed.
It might, however, be processed multiple times (more on that in a minute).

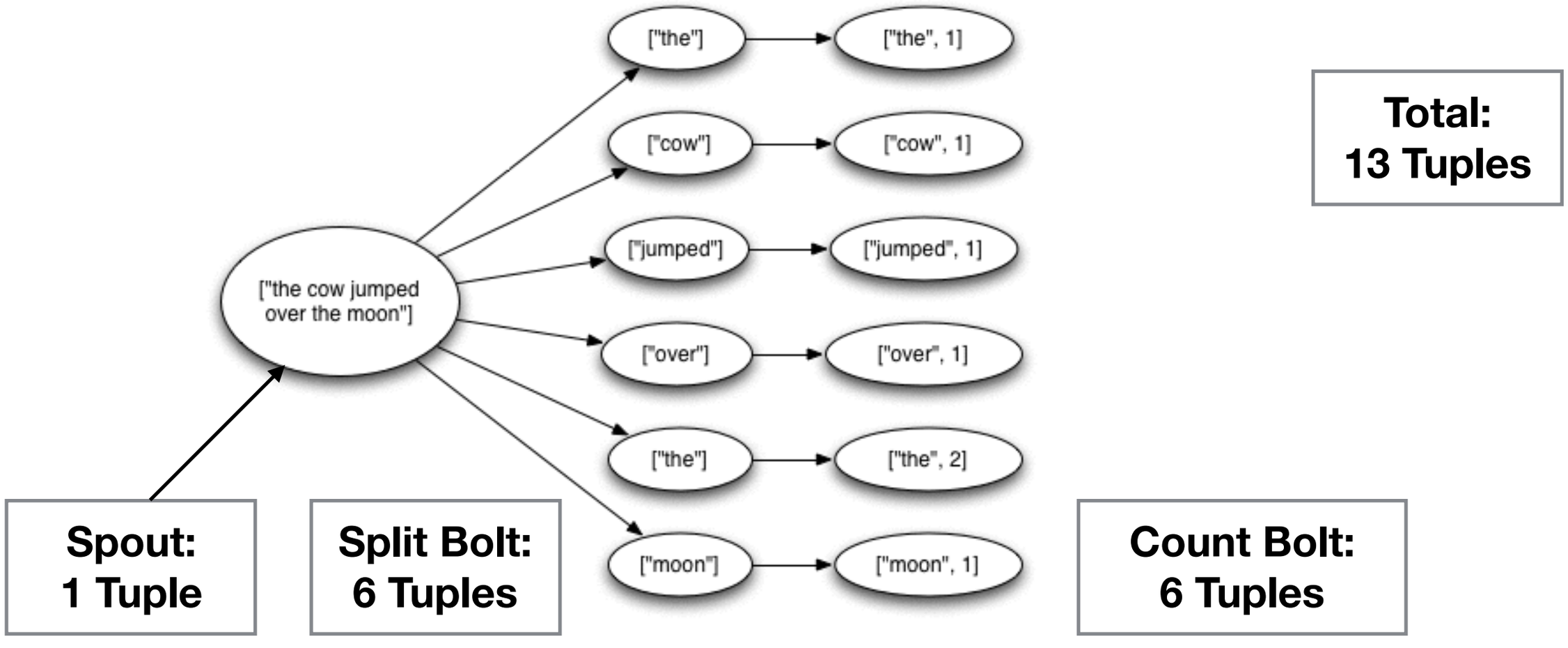
And it requires some work by the user.
== You can configure the level of guarantee.

RELIABILITY
Guaranteed Message Processing

Meet the mighty Tuple Tree.

Example from:

https://github.com/nathanmarz/storm/wiki/Guaranteeing-message-processing



RELIABILITY

Guaranteed Message Processing

“Fully processed” means the whole tuple tree has been processed.

This considers a tuple that was produced by a spout.

A tuple is considered failed when its tree of messages fails to be fully processed *within a specified timeout.*

Config.TOPOLOGY_MESSAGE_TIMEOUT_SECS

– This timeout can be configured on a topology-specific basis.

– Defaults to 30 seconds.

PLAN 9 FROM OUTER KITTEN

Probably the best game ever invented. Seriously.

PLAN 9 FROM OUTER KITTEN
Overview

Overview Game Arena

- One image with all pixels blocked.
- There is a collaborative effort to uncover the whole

Local Playground

- Your local area represents small part of the image
- You unlock a pixel by matching kittens in a cat-based trial (CBT)!
- One solved CBT will give you 1 point.
- Special events happen based on player behaviour or the surrounding wheeling and dealings of the ill-intended Kitten Robbers!

Special Events Create Player Badges

- This is what we will be implementing.

Arghh!

They’re coming!

Phew.

Hands-on!

PLAN9 FROM OUTER KITTEN - ONLINE!
Project Links

Tutorial

https://github.com/DECK36/plan9_workshop_tutorial

PHP Web App

https://github.com/DECK36/deck36-php-web-app

Node.js API Backend

https://github.com/DECK36/deck36-api-backend

Storm Backend for Node.js

https://github.com/DECK36/deck36-storm-backend-nodejs

PLAN9 FROM OUTER KITTEN - ONLINE!
Further Links

Further Links

http://storm.incubator.apache.org/

https://storm.incubator.apache.org/documentation/Multilang-protocol.html

http://lambda-architecture.net/

http://www.datasalt.com/2013/01/an-example-lambda-architecture-using-trident-hadoop-and-splout-sql/

https://github.com/pereferrera/trident-lambda-splout