DECK36

Storm w/ PHP

IPC Autumn 2014

About Martin & Mike

Martin Schütte

System Automation Engineer

martin.schuette@deck36.de

@m_schuett

Mike Lohmann

Co-Founder / Software Engineer

mike.lohmann@deck36.de

@mikelohmann

DECK36

- DECK36 is a young spin-off from ICANS
- Small team of 5 engineers
- Longstanding expertise in designing, implementing and operating complex web systems
- Offering our expert knowledge in:
  - Automation & Operations
  - Architecture & Engineering
  - Rapid Prototyping
- @deck_36

WE’RE HERE!

What will be going on today?

WE'RE HERE

Roadmap - Part 01

Introduction

Plug and Play

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

Preparation

- Checkout tutorial and sources

Technology overview and Hands on Part 1

- Vagrant

Coffee!

WE'RE HERE

Roadmap - Part 02

Technology overview and Hands on Part 2

- RabbitMQ, Redis, Storm
- Your own Dev environment for pf9ok!

Lunch time!

WE'RE HERE

Roadmap - Part 03

Viewing Code

- Let`s go inside the kittens and see how Sf2 plays with Storm.

Deploy

- Bring some changes into the game.

Coffee, again!

WE'RE HERE

Roadmap - Part 04

Badge Development

- Your own Storm-based backend module!

Cold hops

Schorle, finally!

WE'RE HERE

Roadmap - Part 01

Introduction

Plug and Play

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

Preparation

- Checkout tutorial and sources

Technology overview and Hands on Part 1

- Vagrant

PLAN 9 FROM OUTER KITTEN

Probably the best game ever invented. Seriously.

PLAN 9 FROM OUTER KITTEN

Probably the best game ever. Seriously.

Play Plan 9.

Register and unlock pixels by matching kittens while the Kitten Robbers try to stop you!

PLAN 9 FROM OUTER KITTEN
Overview

Overview Game Arena

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

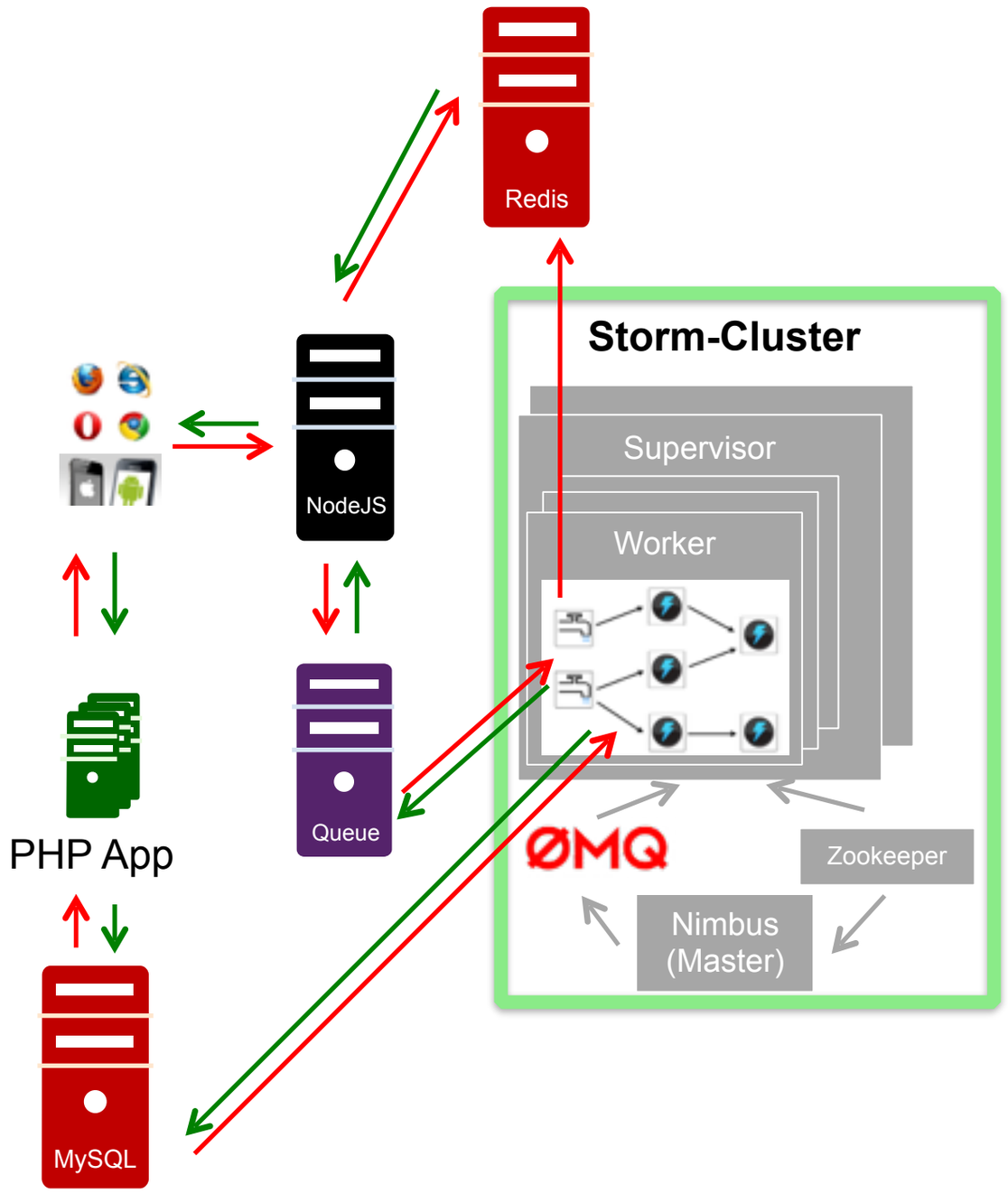
Local Playground

- 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.

PLAN 9 FROM OUTER KITTEN
System Architecture



Bring in new logic or update existing by deploying new revisions of existing Topologies

Reuse existing business logic

PLAN 9 FROM OUTER KITTEN
Badges

High Five

- Two players solve a pixel with the same cat at the same time.

Record Breaker and Record Master

- Points for successful trials accumulate until you fail.
- Each time you fail, a new record is made.
- If you break your own record, you receive **RecordBreaker**
- If you have a Top 3 record, you receive bonus points
- If the points between your new record and your old record is best for all users, you receive **RecordMaster**

PLAN 9 FROM OUTER KITTEN
Badges

Prime Cat

- If you made the most points in the last 60 seconds, you receive **PrimeCat**.
- Your points during the next 60s are then doubled.

Stumble Blunder a.k.a. LOLCAT

- You fail 5 times in a row, you receive **StumbleBlunder**.
- You don’t receive any points during the next 3 minutes.
- (You need to earn your ability to receive points.)

PLAN 9 FROM OUTER KITTEN
Badges

Kitten Robbers From Outer Space

- The Kitten Robbers will randomly attack a player.
- (The player will loose the last solved pixel.)
- The player will loose 100 points.

Raider Of The Kitten Robbers

- If the Kitten Robbers attack you, you fight them off with a **HighFive**.
- (The Raider will receive all points from all other players during the next 60s).
- The Raider will receive 1000 bonus points.

Hands-on: Installation!

Plan9 Tutorial
Insanity! Easy to use.

Install the VM, IDE, PHP-Web-App, Node-Backend, Storm-PHP-Project

- Follow: https://github.com/DECK36/plan9_workshop_tutorial
- Windows < 8? (Please try, but without guarantee).

Let`s start.

VAGRANT

VAGRANT
VMs Configuration and Provisoning

“Local cloud”

- Self service
- Instant provisioning
- Cost efficient
- Elastic
- Pay per use

VAGRANT
Providers

Vagrant VM Providers:

- VirtualBox: “default”, works offline, ressource hungry
- Docker: lightweight, requires Linux, good for testing
- AWS EC2: remote VMs, good for automation (Jenkins)
- 3rd party plugins for KVM, libvirt, ESXI, …

Provisioning:

- Shell script
- Puppet, apply manifest or run agent
- Chef, solo or client
- Ansible playbooks
- Docker containers

Coffee!

WE'RE HERE

Roadmap - Part 02

Technology overview and Hands on Part 2

- RabbitMQ, Redis, Storm
- Your own Dev environment for pf9ok!

RABBITMQ

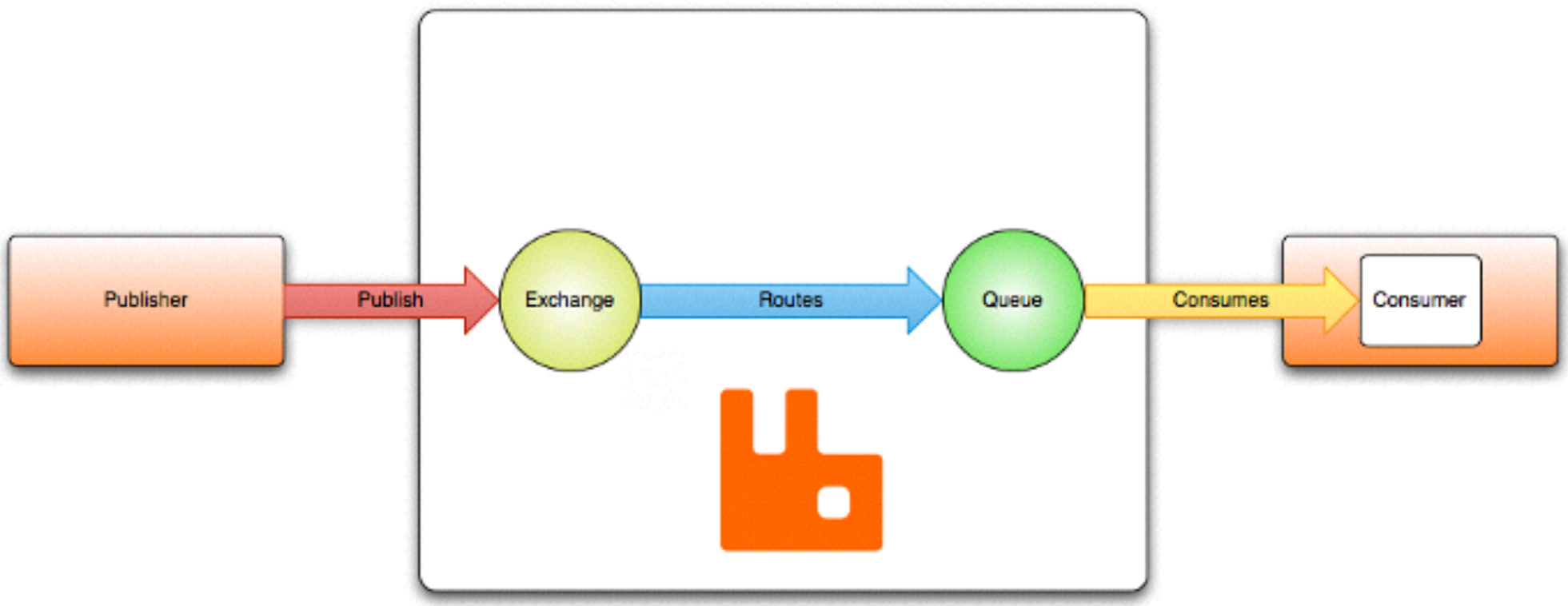
RabbitMQ

Message queueing and routing

Messaging:

- Implementation of Advanced Message Queuing Protocol (AMQP)
- Message Queue to connect services with reliable delivery

"Hello, world" example routing

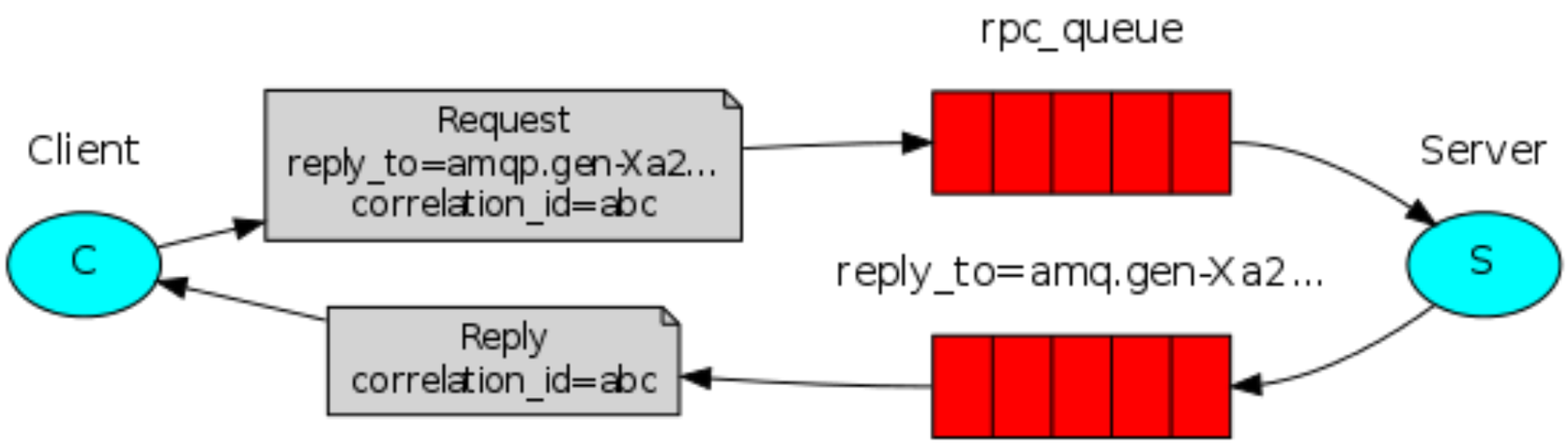


RabbitMQ

Message queueing and routing

Routing:

- Features: Exchanges, routing keys
  - allow for advanced routing and forwarding decisions.
- Our use-case: build a channel for RPC



REDIS

Redis
caching and storing key-values

Redis is an open source, BSD licensed, advanced key-value cache and store.

Possible clients:

- http://redisdesktop.com/
- Redis-Cli

Redis
caching and storing key-values

Redis Data Examples

Keys		**Values**	
page:index.html	—>	<html><head>[...]	<— String
count	—>	898	<— int
plan9_pixel_free	—>	{{1,2}, {3,4}, …}	<— SET
user_1	—>	points=>10, socket_id=>9392	<— HASH
ids	—->	[1,2,3,4,5]	<— LIST
plan9_highscores	—>	user_1=>5, user_2=>10	<— Sorted Set

STORM

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
- 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

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

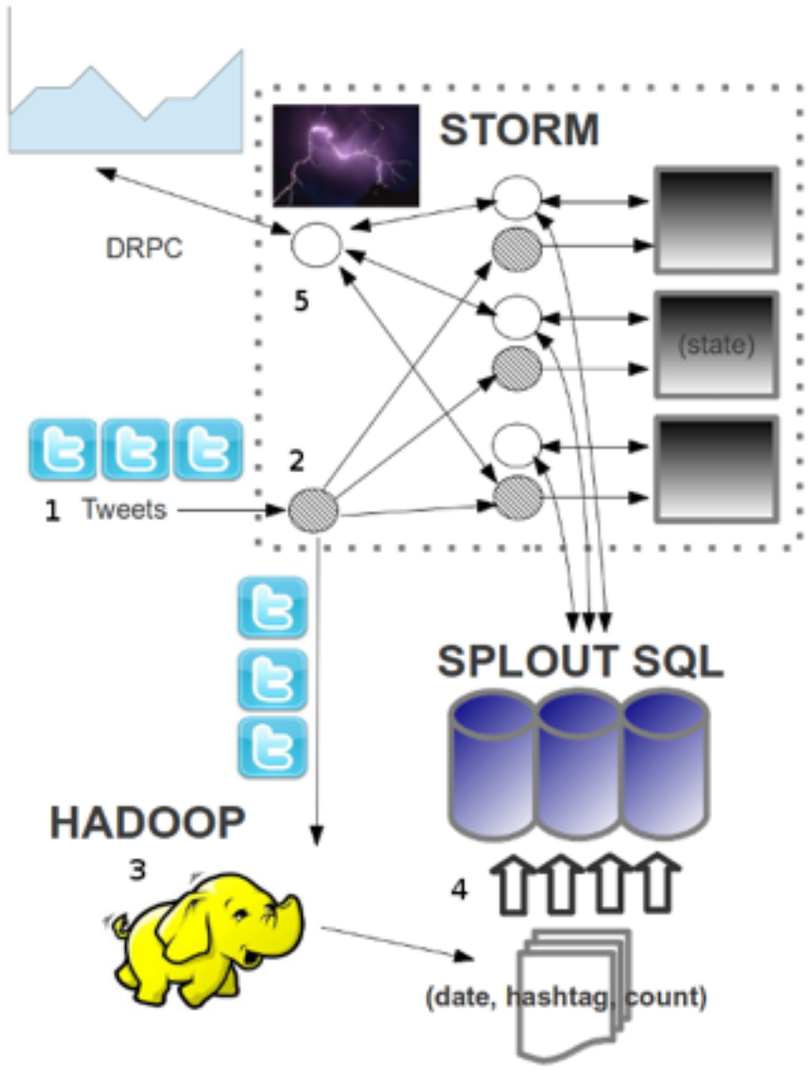
SOLUTION ARCHITECTURE
Lambda Architecture

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

CONCEPTS
Tuple

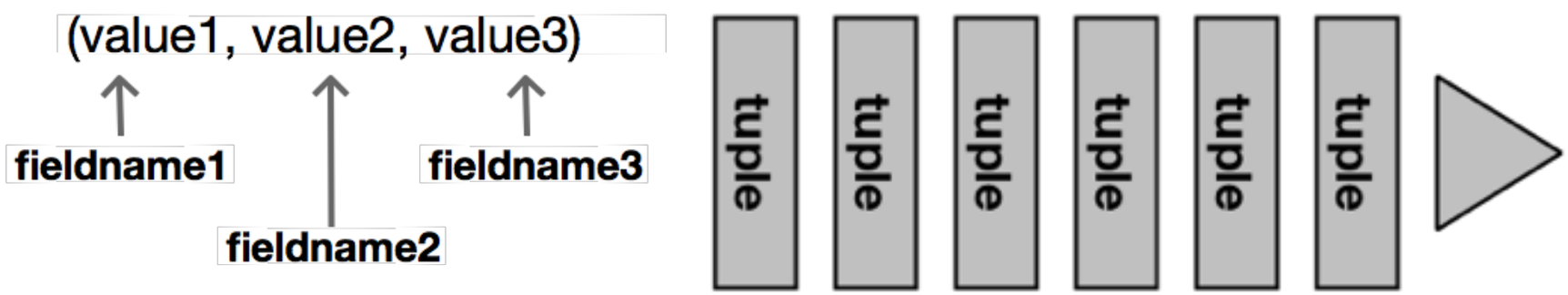
A tuple is one single “message”

- 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

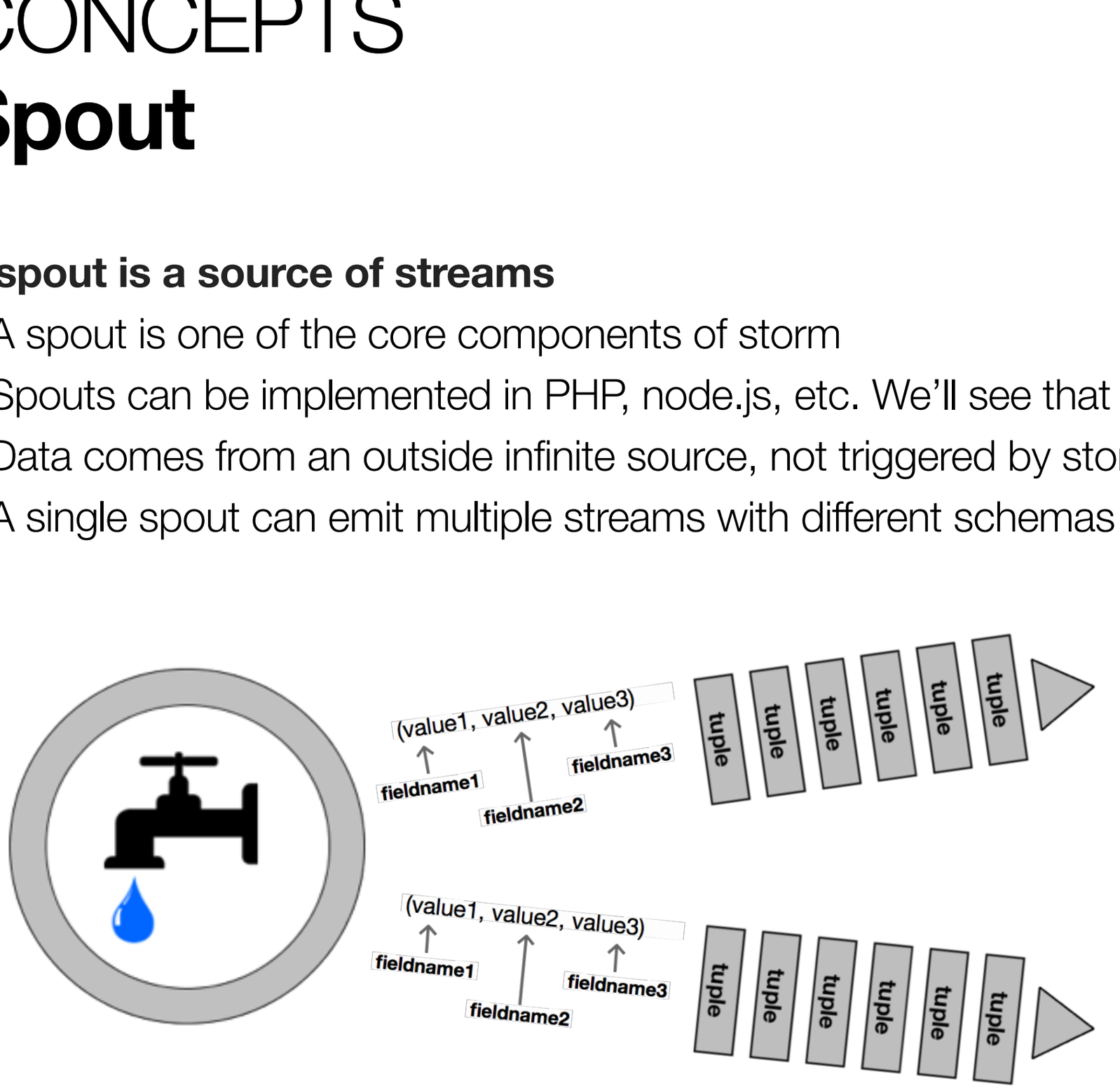
- The “stream” is the core abstraction construct within storm
- “Storm transforms streams into new streams”
- Storm components can consume and produce multiple different streams
- Tuples in a single stream can be handled differently based on a key
- All tuples in a single stream must follow the same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



CONCEPTS
Spout

A spout is a source of streams

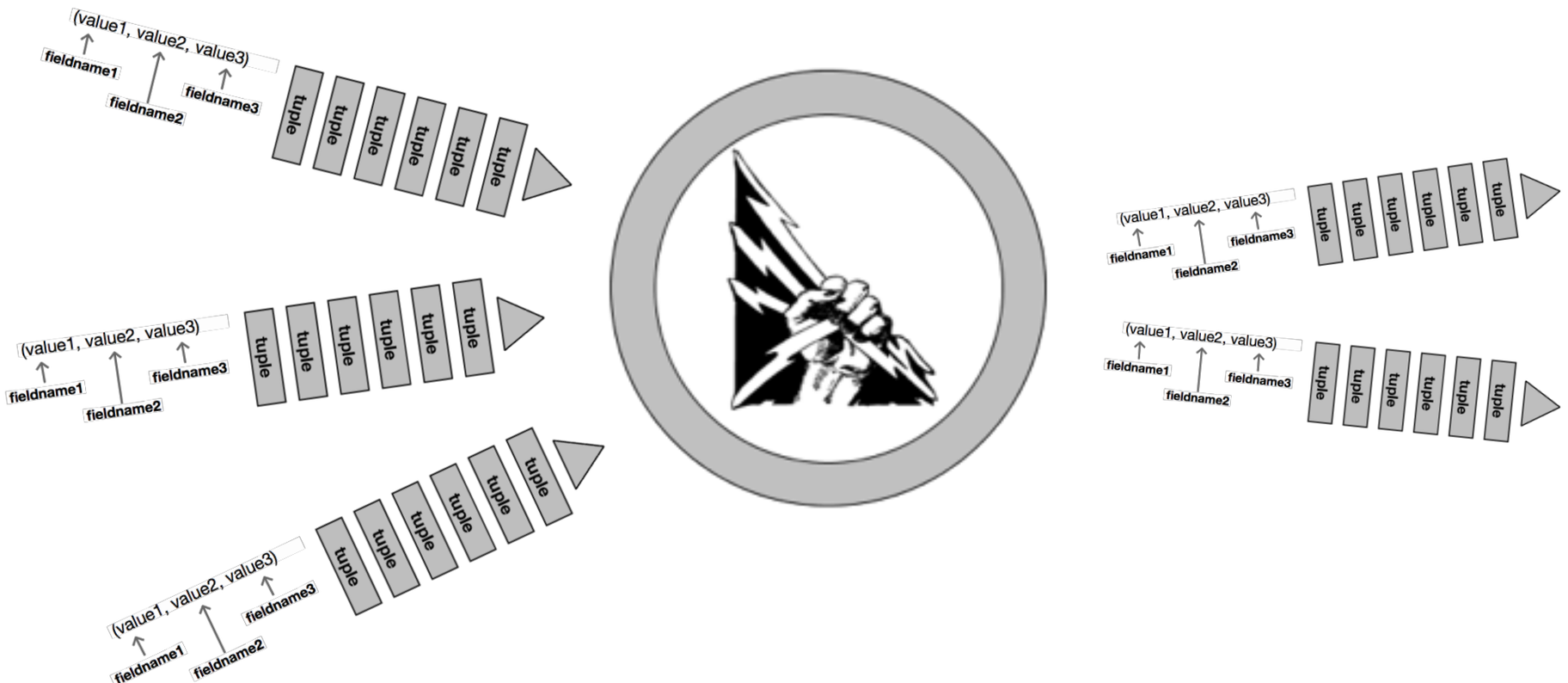
- A spout is one of the core components of storm
- Spouts can be implemented in PHP, node.js, etc. We'll see that later.
- Data comes from an outside infinite source, not triggered by storm events
- A single spout can emit multiple streams with different schemas



CONCEPTS
Bolt

A bolt is a worker process

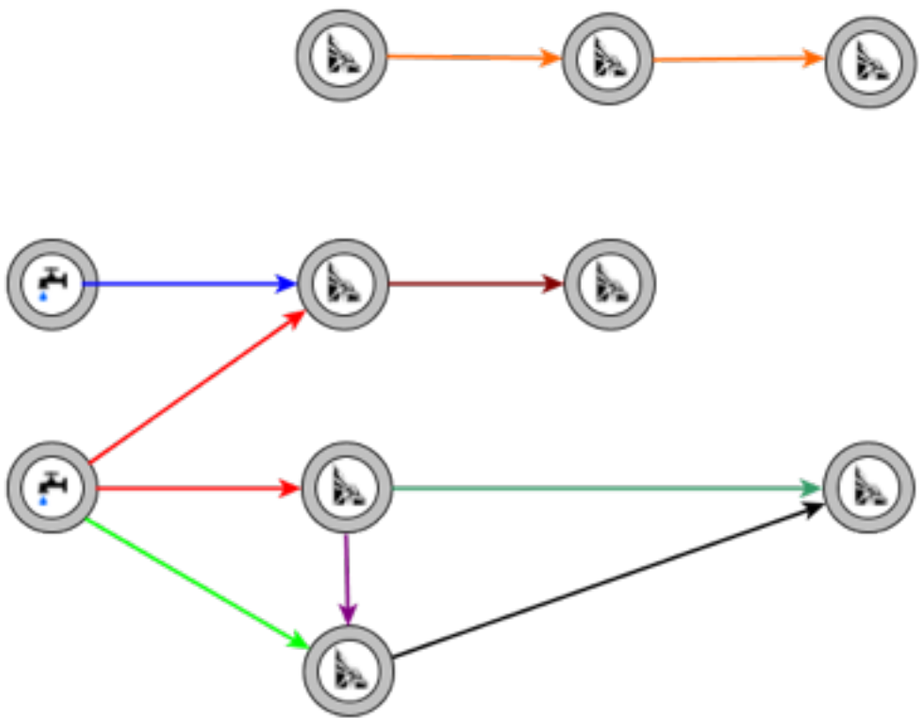
- A bolt is the second core component of storm
- We'll implement bolts later.
- A bolt consumes one or more streams. These can use multiple schemas.
- A bolt emits zero or more streams.
- A bolt can produce side effects. For instance, a database update.



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.”

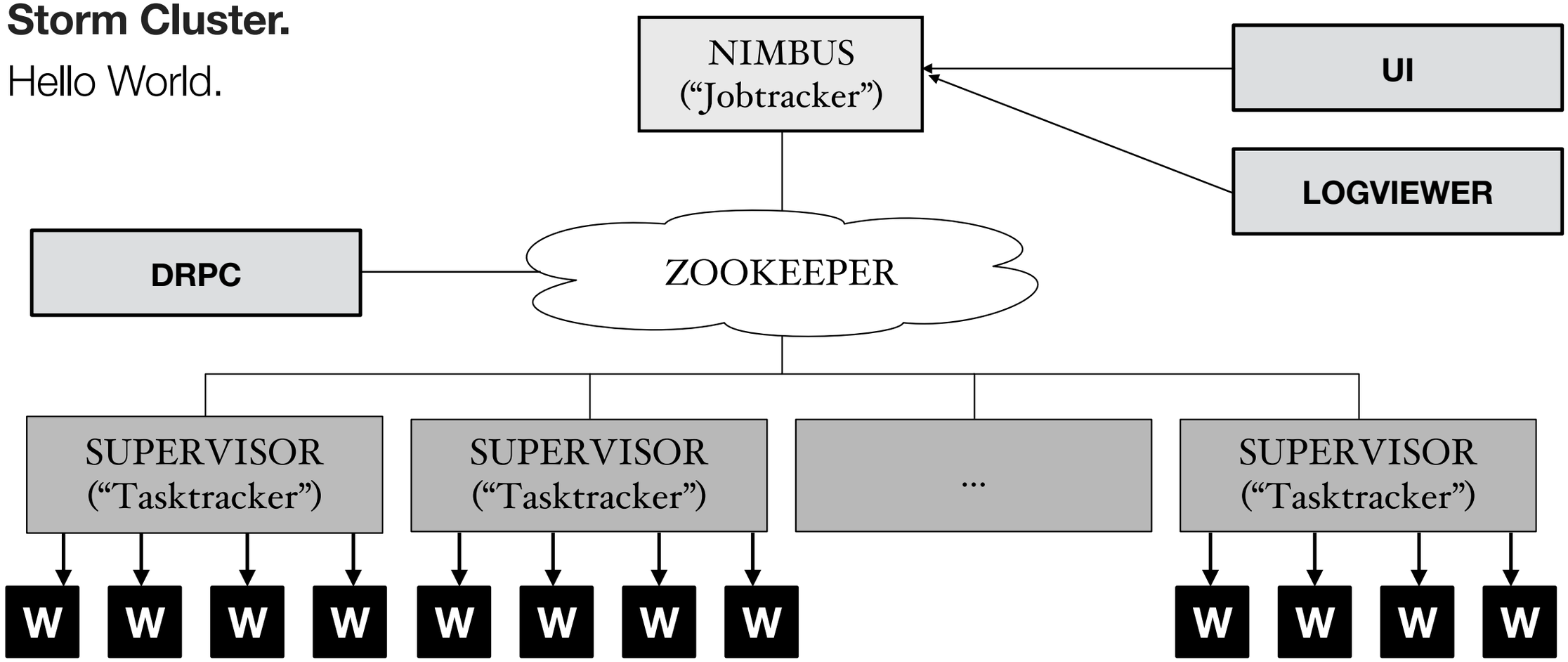
CONJURING DAEMONS

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

PHP

```php
<?php

require_once('storm.php');

class PHPSpout extends ShellSpout
{

    protected function nextTuple()
    {
        $this->emit(array($value1, $value2, $value3));
    }

    protected function ack($tuple_id)
    {
        return;
    }

    protected function fail($tuple_id)
    {
        return;
    }
}

$RandomSpout = new RandomPHPSpout();
$RandomSpout->run();
```

```php
<?php

require_once('storm.php');

class EmptyPHPBolt extends BasicBolt
{
    public function process(Tuple $tuple)
    {
        $this->emit($tuple->values);
        // $this->ack($tuple); is automatically called by the PHP Bolt implementation
    }

}

$bolt = new EmptyPHPBolt();
$bolt->run();
```

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).

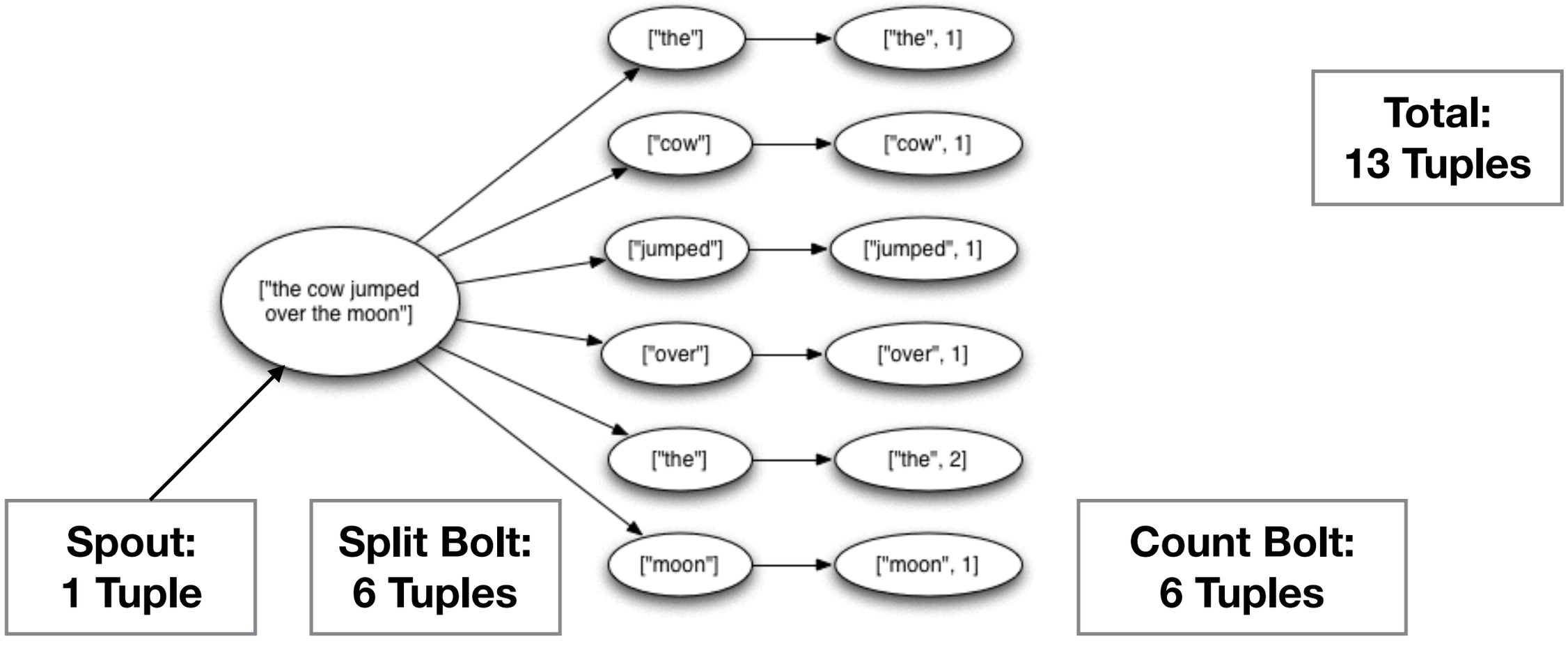
And it requires some work by the user.
== You can configure the level of guarantee.

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.

Lunch time!

WE'RE HERE

Roadmap - Part 03

Viewing Code

- Let's go inside the kittens and see how Sf2 plays with Storm.

Deploy

- Bring some changes into the game.

Eyes-on: Code!

Symfony2 + Storm PHP Bolts

Symfony2 + Storm PHP Bolts
Configuration and Execution by Sf2

Storm Bundle

- Each Bolt has a command -> To be executable by Storm
- Each Bolt has a configuration -> storm.yml
- Whole Sf2 universe is accessible in Storm
- Bolts implement storm.php abstract class BasicBolt
- Storm MultilangProtocol works on STDIN and STDOUT
- A bolt waits for Tuples on STDIN
  - -> while(true) to wait for input on STDIN
  - -> If input -> process unit stop word end
    - write something to Redis
- Badges run in Topologies

Symfony2 + Storm PHP Bolts
PHP in Storm

StatusLevelTopology

- Using MultilangBolt to execute Bolt implemented in PHP
- Decide by config which level to send
- all configuration from storm.yml
- Create queue subscribed to plan9 exchange for routing key
- Do the business logic
- Push message to plan9-backchannel

Deploying the PHP Bolts

Deploying the PHP Bolts
Phar archives for the win

StatusLevelBolt

- Change storm.yml config (points for status)
- Add new configuration to the phar
- Build the jar
- deploy to the „cluster“

Deploying the PHP Bolts
Phar archives for the win

Environments

- Environment could become hhvm as well (not provided in the vm)

Coffee, again!

WE'RE HERE

Roadmap - Part 04

Badge Development

- Create your own plan9 idea.
- Implement a bolt with game logic.
- Play around with the dev environment.
- Play the game.
- Create a pull request for the plan9 repo.

Tick-Tuple Bolt to start over

Cold hops

Schorle, finally!

TRIDENT

API, Functionality, Concepts

TRIDENT

Topology

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// For Trident, we have the same configuration object as with Storm.

Config conf = new Config();


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// We don't use the topology builder directly, but create TridentTopology instead.

TridentTopology topology = new TridentTopology();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Now, we don't add components to the topology, but we define "streams".

// We can use regular Storm spouts to produce streams:
Stream randomJavaStream     = topology.newStream("java_random",
                                new RandomJavaSpout());
Stream randomPHPStream      = topology.newStream("php_random",
                                new MultilangAdapterSpout("/usr/bin/php", "RandomPHPSpout.php", "", "src", "random"));
Stream randomNodejsStream   = topology.newStream("nodejs_random",
                                new MultilangAdapterSpout("/usr/local/bin/node", "RandomNodeJSSpout.js", "", "src", "random"));
```

```
// We can now define new "streams" or "states" that do computation.
// This is similar to one part (a connected subcomponent) of a complete Storm topology.
topology.merge(randomJavaStream, randomPHPStream, randomNodejsStream)
        .each(new Fields("src", "random"), new TridentPrintFilter("MERGED_STREAM"))
        .each(
            new Fields("src", "random"),                                                    // input fields for the function
            new MultilangBoltTridentFunction("/usr/bin/php", "SimpleStreamTupleJoinBolt.php"),      // the function
            new Fields("name", "verb", "object"))                                           // result fields of the function
                                                                                            // are added to the tuple
        .each(new Fields("name"), new EmptyStringFilter())
        .groupBy(new Fields("name"))
        .aggregate(new Fields("name"), new Count(), new Fields("nTuples"))
        .each(new Fields("name", "nTuples"), new TridentPrintFilter("AGGREGATION_RESULT"));

// The TridentTopology contains a topology builder:
StormTopology tridentStormTopology = topology.build();


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// We can now submit the Trident topology to the cluster, exactly like any other Storm topology.

StormSubmitter.submitTopology(args[0], conf, topology.build());
```

TRIDENT STATE

Persistent Topology State

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// We don't use the topology builder directly, but create TridentTopology instead.

TridentTopology topology = new TridentTopology();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Now, we don't add components to the topology, but we define "streams".

// We can use regular Storm spouts to produce streams:
Stream randomJavaStream     = topology.newStream("java_random",
                                new RandomJavaSpout());
Stream randomPHPStream      = topology.newStream("php_random",
                                new MultilangAdapterSpout("/usr/bin/php", "RandomPHPSpout.php", "", "src", "random"));
Stream randomNodejsStream   = topology.newStream("nodejs_random",
                                new MultilangAdapterSpout("/usr/local/bin/node", "RandomNodeJSSpout.js", "", "src", "random"));



// We now explicitly define a TridentState. To do so, the last step in the topology must create such.
TridentState countState =
topology.merge(randomJavaStream, randomPHPStream, randomNodejsStream)
        .each(new Fields("src", "random"), new TridentPrintFilter("MERGED_STREAM"))
    .each(new Fields("src", "random"),
        new MultilangBoltTridentFunction("/usr/bin/php", "SimpleStreamTupleJoinBolt.php"),
        new Fields("name", "verb", "object"))
    .each(new Fields("name"), new EmptyStringFilter())
    .groupBy(new Fields("name"))

    // We can simply swap "aggregate" for "persistentAggregate"
    .persistentAggregate(new MemoryMapState.Factory(), new Fields("name"), new Count(), new Fields("count"));
```

TRIDENT

DRPC

```
// We now explicitly define a TridentState. To do so, the last step in the topology must create such.
TridentState countState =
topology.merge(randomJavaStream, randomPHPStream, randomNodejsStream)
        .each(new Fields("src", "random"), new TridentPrintFilter("MERGED_STREAM"))
        .each(new Fields("src", "random"),
            new MultilangBoltTridentFunction("/usr/bin/php", "SimpleStreamTupleJoinBolt.php"),
            new Fields("name", "verb", "object"))
        .each(new Fields("name"), new EmptyStringFilter())
        .groupBy(new Fields("name"))
        // We can simply swap "aggregate" for "persistentAggregate"
        .persistentAggregate(new MemoryMapState.Factory(), new Fields("name"), new Count(), new Fields("count"));



// The state can now be accessed anywhere in the topology using "stateQuery".
// We add a distributed remote procedure call (DRPC) service to allow external
// clients to query data based on the stored data.
topology.newDRPCStream("getNameCount")
        .each(new Fields("args"), new Split(), new Fields("name"))
        .groupBy(new Fields("name"))
        .stateQuery(countState, new Fields("name"), new MapGet(), new Fields("count"))
        .each(new Fields("count"), new FilterNull());
```