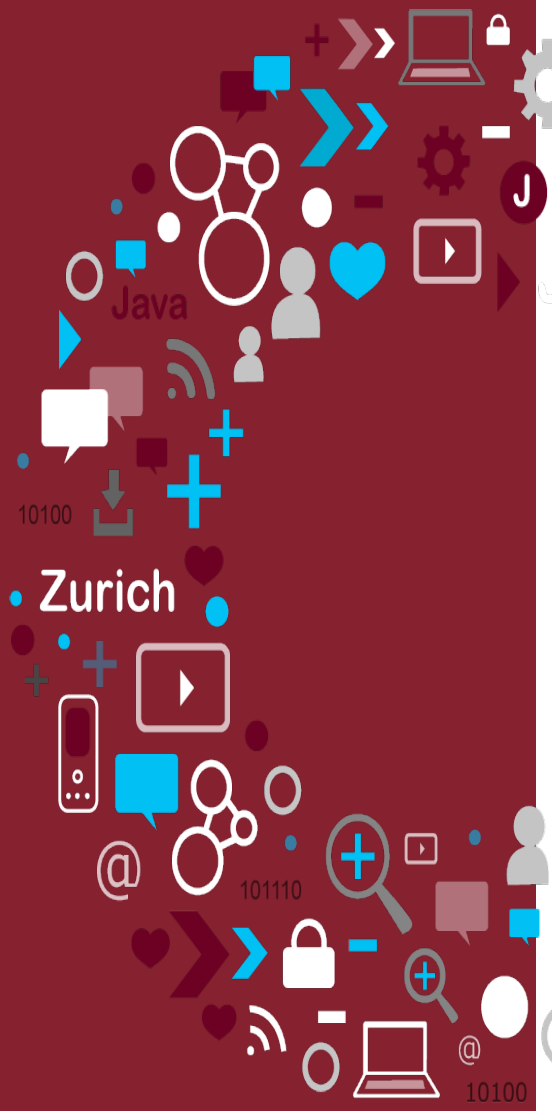
JAZOON'13

INTERNATIONAL CONFERENCE FOR THE SOFTWARE COMMUNITY

Kafka and Storm – event processing in realtime

Guido Schmutz



WELCOME

Kafka and Storm – event processing in realtime

Guido Schmutz

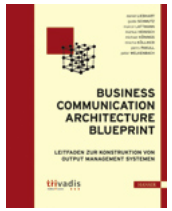
Jazoon 2013

23.10.2013

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

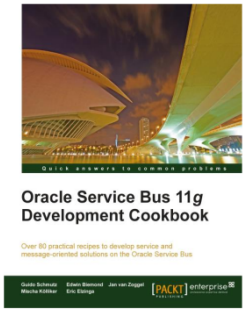
Guido Schmutz

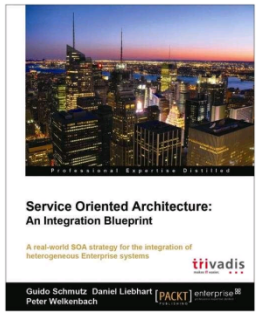
- Working for Trivadis for more than 16 years
- Oracle ACE Director for Fusion Middleware and SOA
- Co-Author of different books
- Consultant, Trainer Software Architect for Java, Oracle, SOA, EDA, BigData und FastData
- Member of Trivadis Architecture Board
- Technology Manager @ Trivadis



- More than 20 years of software development experience

- Contact: guido.schmutz@trivadis.com
- Blog: http://guidoschmutz.wordpress.com
- Twitter: gschmutz



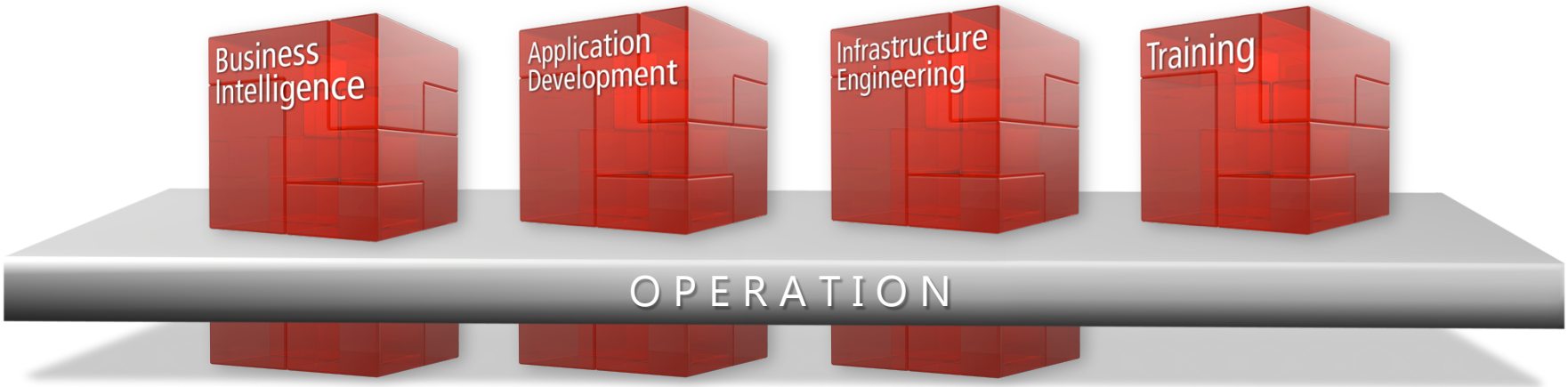


trivadis
makes IT easier.

Our company

Trivadis is a market leader in IT consulting, system integration, solution engineering and the provision of IT services focusing on ORACLE and Microsoft technologies in Switzerland, Germany and Austria.

We offer our services in the following strategic business fields:



Trivadis Services takes over the interacting operation of your IT systems.

With over 600 specialists and IT experts in your region

12 Trivadis branches and more than 600 employees

200 Service Level Agreements

Over 4,000 training participants

Research and development budget: CHF 5.0 / EUR 4 million

Financially self-supporting and sustainably profitable

Experience from more than 1,900 projects per year at over 800 customers

trivadis
makes IT easier.

Agenda

1. **Introduction**
2. What is Apache Kafka?
3. What is Twitter Storm?
4. Summary

Big Data Definition (Gartner et al)

Characteristics of Big Data: Its Volume, Velocity and Variety in combination

Tera-, Peta-, Exa-, Zetta-, Yota- bytes and constantly growing

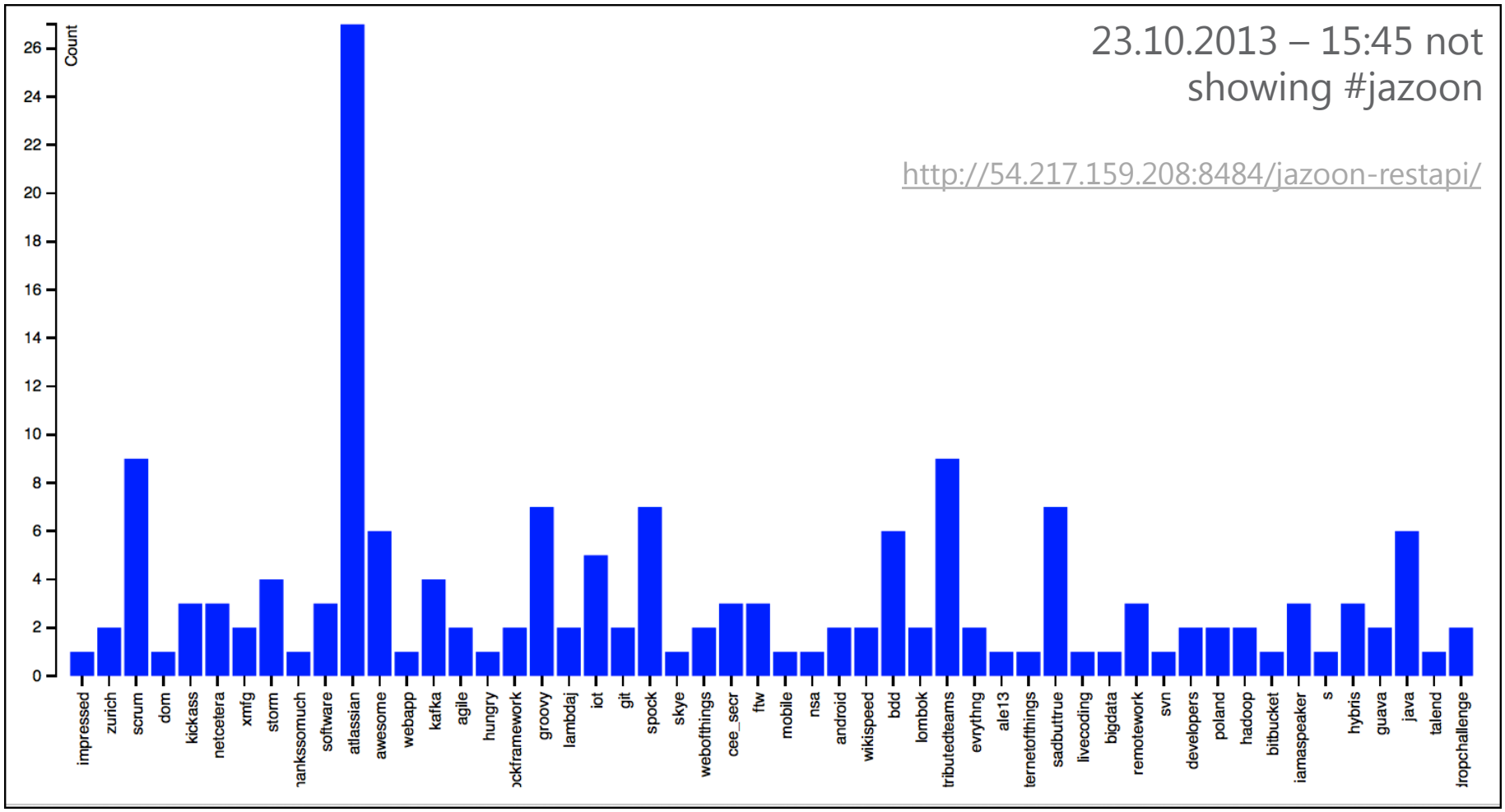
"Traditional" computing in RDBMS is not scalable enough.
We search for "linear scalability"

"Only ... structured information is not enough" – "95% of produced data in unstructured"

+ **Veracity (IBM)** - information uncertainty
+ **Time to action ?** – Big Data + Event Processing = Fast Data


IN 60 SECONDS...
1 NEW DEFINITION IS ADDED ON urban
1,600+ READS ON Scribd.
13,000+ HOURS MUSIC STREAMING ON PANDORA
12,000+ NEW ADS POSTED ON craigslist
370,000+ MINUTES VOICE CALLS ON skype
98,000+ TWEETS
20,000+ NEW POSTS ON tumblr.
THE LARGEST SOCIAL READING PUBLISHING COMPANY!!
New Craigslist Ads
320+ NEW twitter ACCOUNTS
13,000+ iPhone APPLICATIONS DOWNLOADED
100+ NEW Linked in ACCOUNTS
1 associatedcontent NEW ARTICLE IS PUBLISHED
THE WORLD'S LARGEST COMMUNITY CREATED CONTENT!!
QUESTIONS ASKED ON THE INTERNET...
100+ Answers.com
40+ YAHOO! ANSWERS
6,600+ NEW PICTURES ARE UPLOADED ON flickr
600+ NEW VIDEOS
50+ WordPress DOWNLOADS
25+ HOURS TOTAL DURATION
70+ DOMAINS REGISTERED
695,000+ facebook STATUS UPDATES
125+ PLUGIN DOWNLOADS
60+ NEW BLOGS
168 MILLION EMAILS ARE SENT
694,445 SEARCH QUERIES
1,700+ Firefox DOWNLOADS
79,364 WALL POSTS
1,500+ BLOG POSTS
510,040 COMMENTS
Google
Google Search
GO-Globe.com
web technologies

Sample – Show the counts of tweets related to Jazoon (using term jazoon)



Velocity

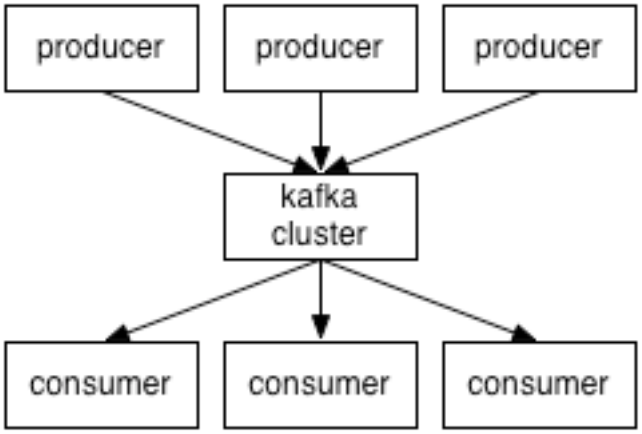
- Velocity requirement examples:
  - Recommendation Engine
  - Predictive Analytics
  - Marketing Campaign Analysis
  - Customer Retention and Churn Analysis
  - Social Graph Analysis
  - Capital Markets Analysis
  - Risk Management
  - Rogue Trading
  - Fraud Detection
  - Retail Banking
  - Network Monitoring
  - Research and Development

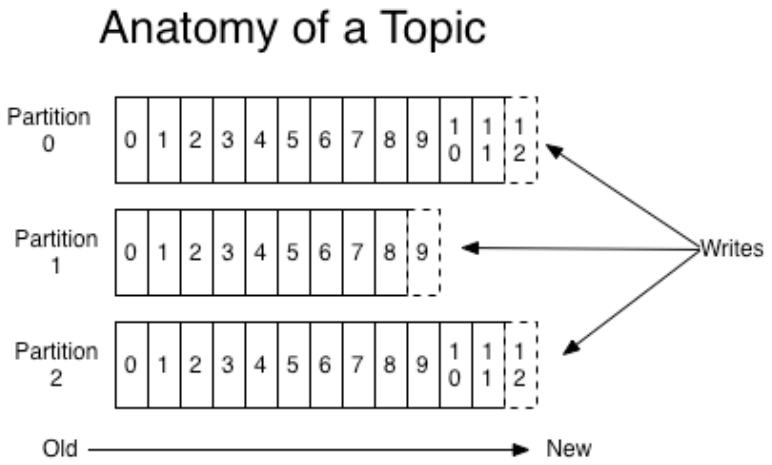
Agenda

1. Introduction
2. **What is Apache Kafka?**
3. What is Twitter Storm?
4. Summary

Apache Kafka

- A distributed publish-subscribe messaging system
- Designed for processing of real time activity stream data (logs, metrics collections, social media streams, …)
- Initially developed at LinkedIn, now part of Apache
- Does not follow JMS Standards and does not use JMS API
- Kafka maintains feeds of messages in topics





Sample (#1)

1) Use the Twitter Streaming API (Twitter Horsebird Client) to get all the tweets containing the term jazoon

Create a Filter Stream for tracking a list of terms

```
StatusesFilterEndpoint endpoint = new StatusesFilterEndpoint();
endpoint.trackTerms(trackTerms);

endpoint.stallWarnings(false);
```

```
<bean id="filterStreamCollector" destroy-method="stop" class="ch.trivadis.sample.twitter.StreamCollector">
    <property name="kafkaHostname" value="${kafka.hostname}" />
    <property name="kafkaPort" value="${kafka.port}" />
    <property name="kafkaTopicName" value="${kafka.topicName11}" />
    <property name="consumerKey" value="[redacted]"/>
    <property name="consumerSecret" value="[redacted]"/>
    <property name="accessToken" value="[redacted]"/>
    <property name="accessTokenSecret" value="[redacted]"/>
    <property name="trackTerms" value="jazoon"/>

    <property name="numberOfProcessingThreads" value="5" />
</bean>
```

Sample (#1)


```
Authentication auth = new OAuth1(consumerKey, consumerSecret,
                        accessToken, accessTokenSecret);

client = new ClientBuilder().name("sampleExampleClient")
        .hosts(Constants.STREAM_HOST).endpoint(endpoint)
        .authentication(auth)
        .processor(new StringDelimitedProcessor(queue)).build();
```

Create a fully configured Horsebird Client instance

```
ExecutorService service = Executors
        .newFixedThreadPool(this.numberOfProcessingThreads);

// Wrap our BasicClient with the twitter4j client
Twitter4jStatusClient t4jClient = new Twitter4jStatusClient(client,queue,
                                        Lists.newArrayList(listener), service);

t4jClient.connect();

for (int threads = 0; threads < this.numberOfProcessingThreads; threads++) {
    t4jClient.process();
}
```

Start the client in multiple threads

Sample (#1)


```
public void store(Status status) throws IOException, InterruptedException {

    final String zkConnection = kafkaHostname + ":" + kafkaPort;
    final String topic = kafkaTopicName;

    TwitterStatusUpdate update = TwitterStatusUpdateConverter.convert(status);

    ByteArrayOutputStream out = new ByteArrayOutputStream();
    DatumWriter<TwitterStatusUpdate> writer =
            new SpecificDatumWriter<TwitterStatusUpdate>(TwitterStatusUpdate.class);

    Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
    writer.write(update, encoder);
    encoder.flush();
    out.close();

    Message message = new Message(out.toByteArray());
    if (producer == null) {
        Properties props = new Properties();
        props.put("zk.connect", zkConnection);
        props.put("producer.type", "async");
        props.put("compression.codec", "1");
        producer = new kafka.javaapi.producer.Producer<Message, Message>(new ProducerConfig(props));
    }
    producer.send(new ProducerData<Message, Message>(topic, message));
```

Convert Twitter status to Avro format

Create the Kafka producer

Send the message to the Kafka topic

Agenda

1. Introduction
2. What is Apache Kafka?
3. **What is Twitter Storm?**
4. Summary

What is Twitter Storm ?

A **platform** for doing analysis on streams of data as they come in, so you can react to data **as it happens.**

- A highly distributed real-time computation system
- Provides general primitives to do real-time computation
- To simplify working with queues & workers
- scalable and fault-tolerant
- complementary to Hadoop

Originated at Backtype, acquired by Twitter in 2011

Open Sourced late 2011

Part of Apache Incubator since September 2013

Hadoop vs. Storm

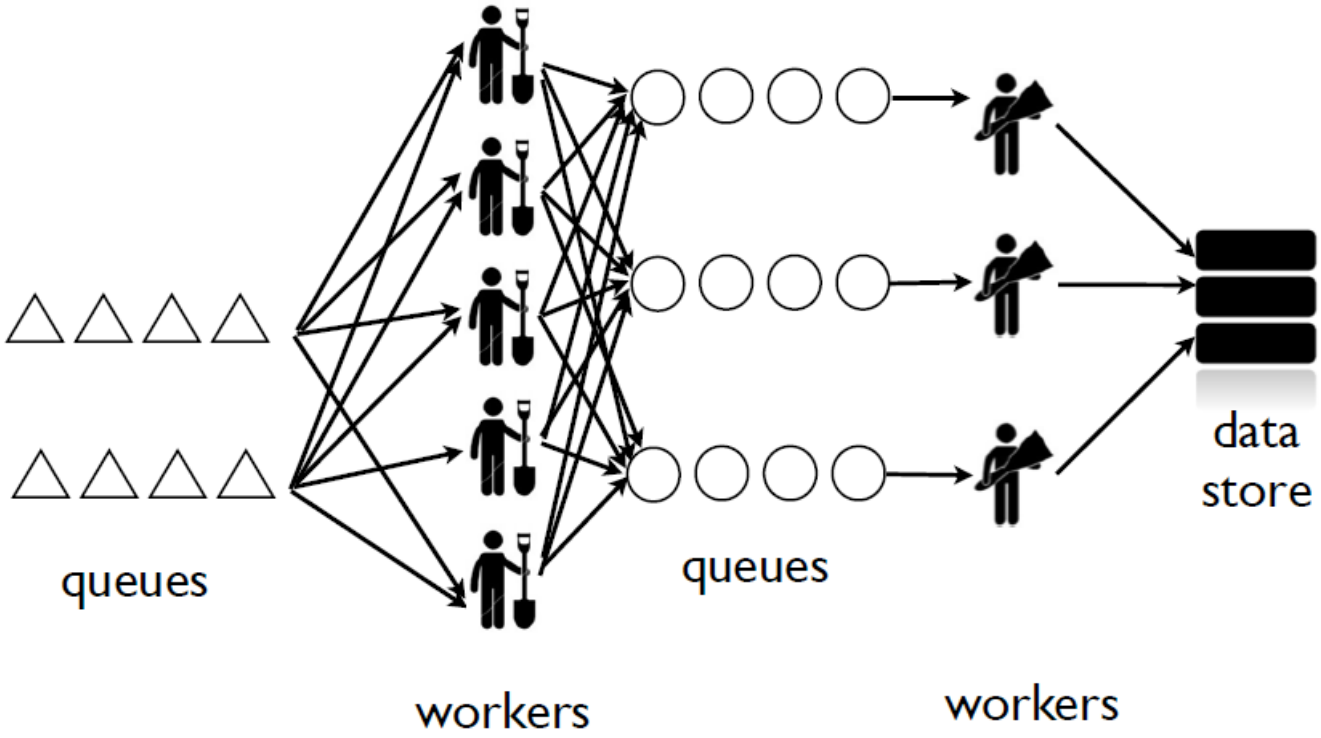
Hadoop		**Storm**
Batch processing		Real-time processing
Jobs runs to completion	!=	Topologies run forever
Stateful nodes		Stateless nodes
Scalable		Scalable
Guarantees no data loss	=	Guarantees no data loss
Open Source		Open Source
=> big batch processing		**=> Fast, reactive, real time procesing**

Idea: Simplify dealing with queue & workers

Scaling is painful – queue partitioning & worker deploy

Operational overhead – worker failures & queue backups

No guarantees on data processing



What does Storm provide?

- At least once message processing
- Fault-tolerant
- Horizontal scalability
- No intermediate queues
- Less operational overhead
- „Just works“
- Broad set of use cases
  - Stream processing
  - Continous computation
  - Distributed RPC

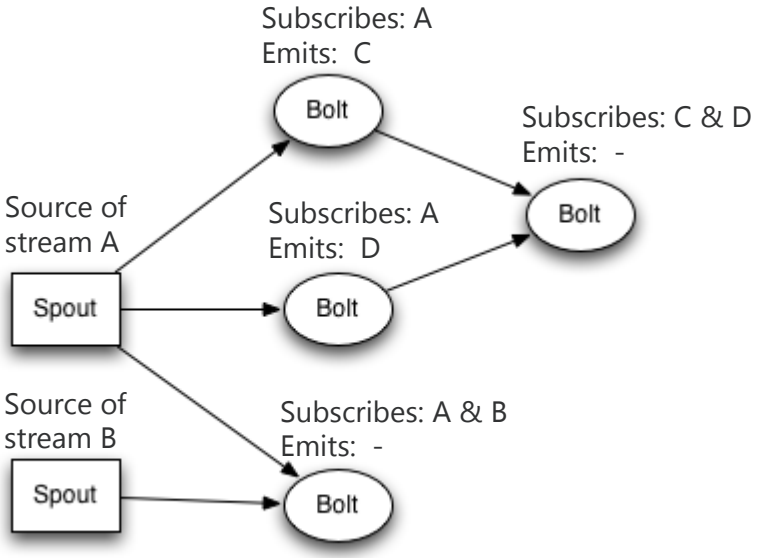
Main Concepts - Streams

Stream

- an unbounded sequence of tuples
- core abstraction in Storm
- Defined with a schema that names the fields in the tuple
- Value must be serializable
- Every stream has an id

Topology

- graph where each node is a spout or bolt
- edges indicating which bolt subscribes to which streams



Main Concepts – Worker, Executor, Task

Worker

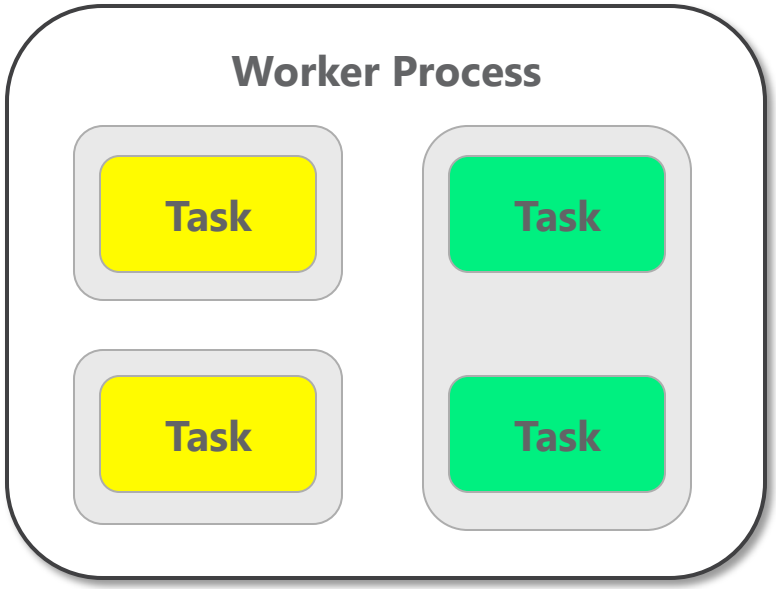
- executes a subset of a topology, may run one or more executors for one or more components

Executor

- thread spawned by worker

Task

- performs the actual data processing

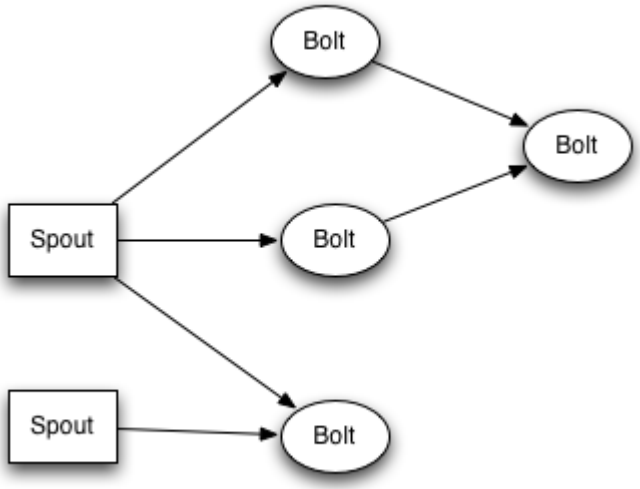


Main Concepts - Spout

A spout is the component which acts as the source of streams in a topology

Generally spouts read tuples from an external source and emit them into the topology

Spouts can emit more than one stream

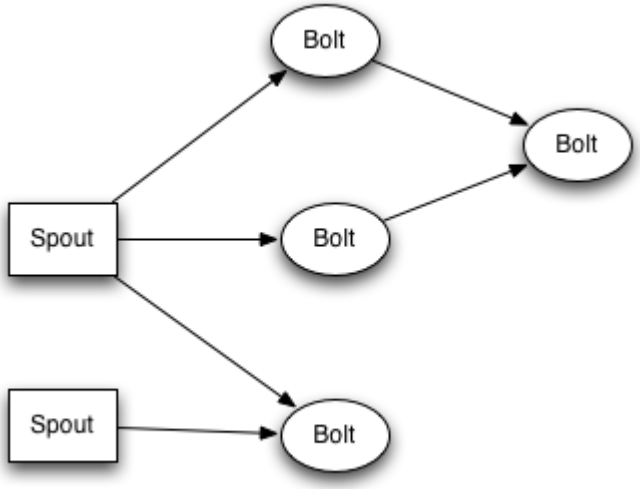


Main Concepts - Bolt

Bolt is doing the processing of the message

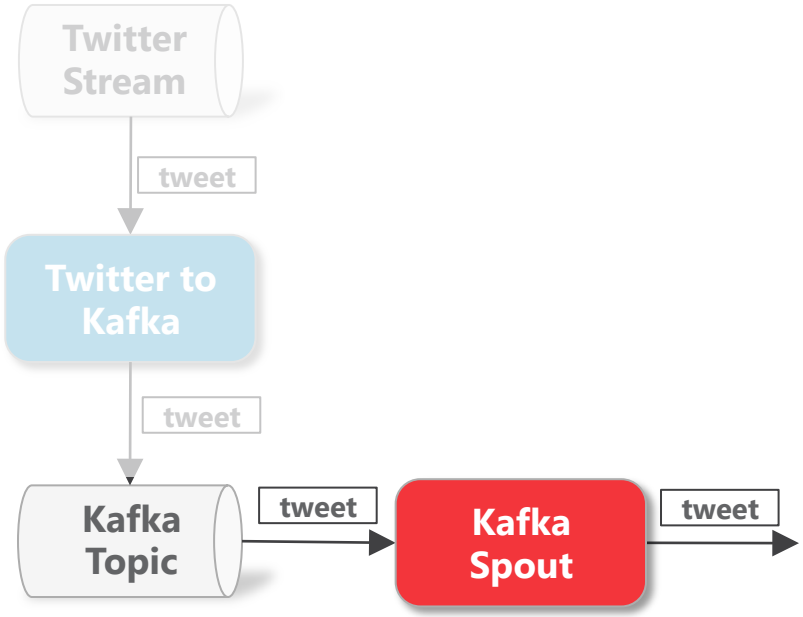
- filtering, functions, aggregations, joins, talking to databases….
- Can do simple stream transformations
- Complex transformations often require multiple steps and therefore multiple bolts

Bolts can emit one or more streams



Sample (#2)

2) Use a Spout to subscribe to the Kafka Topic to get the related tweets



Sample (#2)

Use existing implementation of the Kafka Spout from storm-kafka project

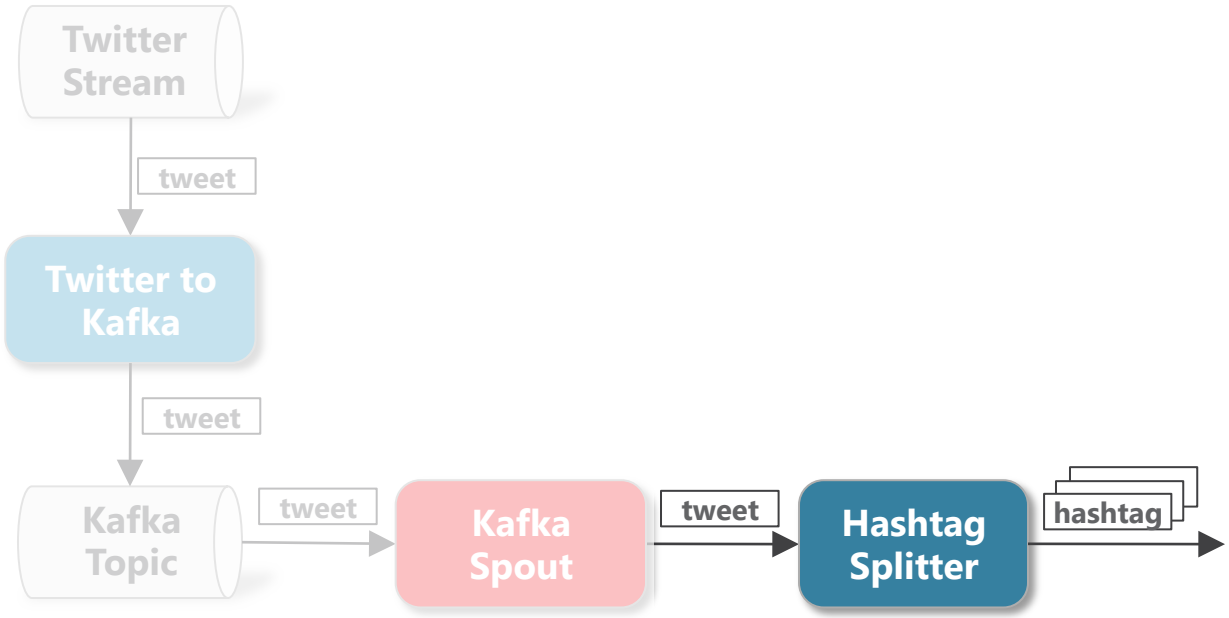
```
int partitions = 1;
final String offsetPath = "/tweet_v11";
final String consumerId = "v1";
final String topic = "TOPIC_TWEET_V10";

List<String> hosts = new ArrayList<String>();
hosts.add("localhost");
SpoutConfig kafkaConfig =
            new SpoutConfig(KafkaConfig.StaticHosts.fromHostString(hosts, partitions),
                            topic, offsetPath, consumerId);
kafkaConfig.bufferSizeBytes = 1024*1024*4;
kafkaConfig.fetchSizeBytes = 1024*1024*4;
kafkaConfig.forceFromStart = true;

KafkaSpout kafkaSpout = new KafkaSpout(kafkaConfig);
```

Create a Kafka configuration

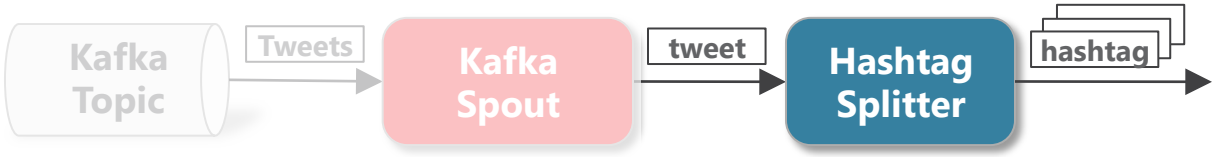
Create Kafka spout to connect to the topic

Sample (#3)

3) Use a bolt to retrieve the hashtags for each tweet



Sample (#3)



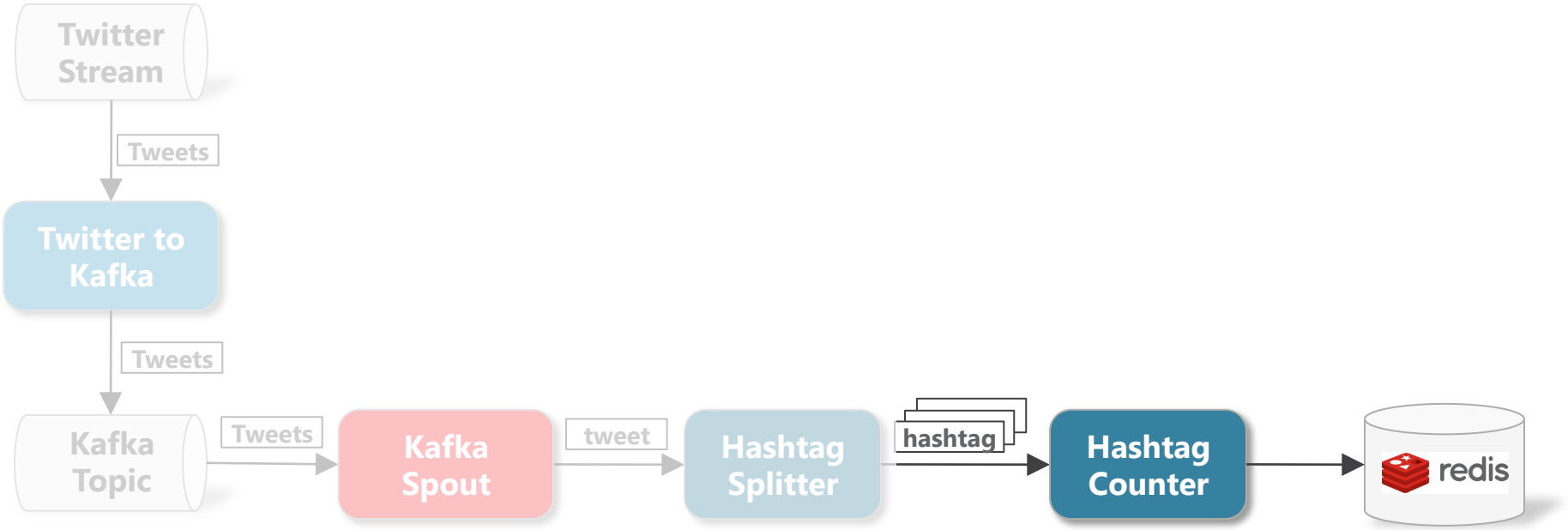

```
public class HashtagSplitter extends BaseBasicBolt {

    public HashtagSplitter() {
    }

    private String normalizeString(String input) {

    @Override
    public void execute(Tuple input, BasicOutputCollector collector) {
        Tweet tweet = (Tweet)input.getValueByField("tweet");
        for (String hashtag : tweet.getHashtags()) {
            String hashtagNormalized = normalizeString(hashtag);

            collector.emit(new Values(hashtagNormalized));
        }
    }

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

Execute is called for each tuple arriving on the input stream

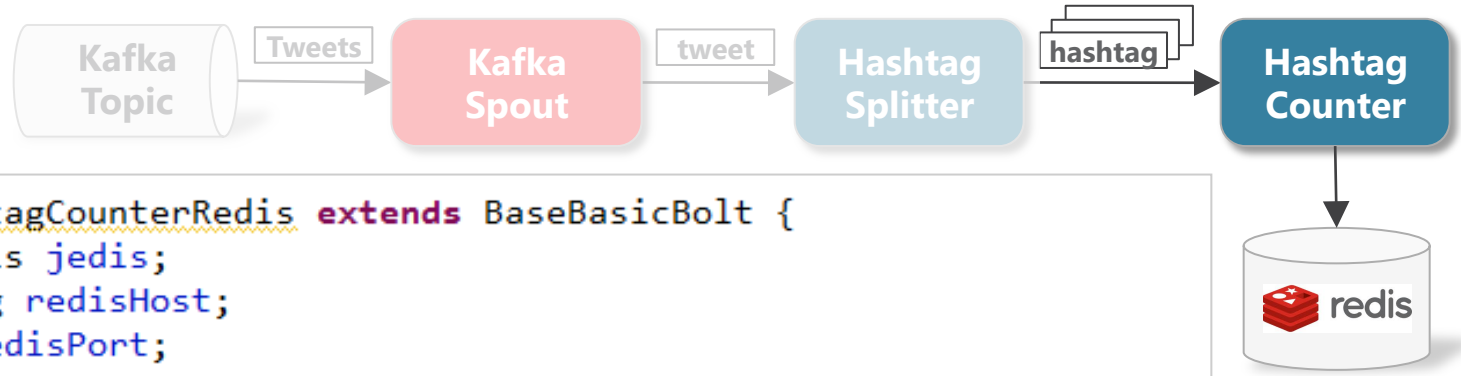
Declares the output fields for the component and is called when bolt is created

Sample (#4)

4) Use a bolt to count the occurrences of hashtags into a Redis NoSQL DB



Sample (#4)

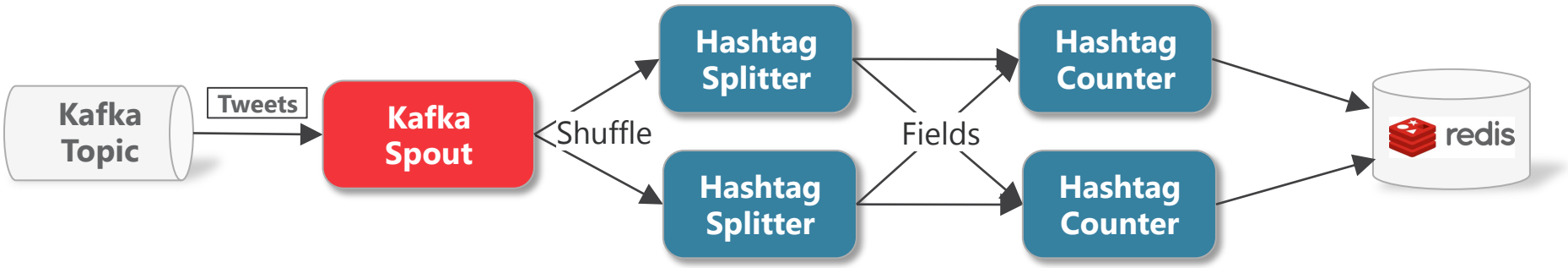



```
public class HashtagCounterRedis extends BaseBasicBolt {
    transient Jedis jedis;
    private String redisHost;
    private int redisPort;

    public HashtagCounterRedis(String redisHost, int redisPort) {

    @Override
    public void execute(Tuple input, BasicOutputCollector collector) {
        String hashtag = input.getStringByField("hashtag");
        System.out.println("hashtag : " + hashtag);
        jedis.hincrBy("jfs2013:hashtags", hashtag, 1);
    }

    @Override
    public void prepare(Map stormConf, TopologyContext context) {
        jedis = new Jedis(redisHost, redisPort, 1800);
        jedis.connect();
        super.prepare(stormConf, context);
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
    }
```

Execute is called for each tuple arriving on the input stream

Prepare is called when bolt is created

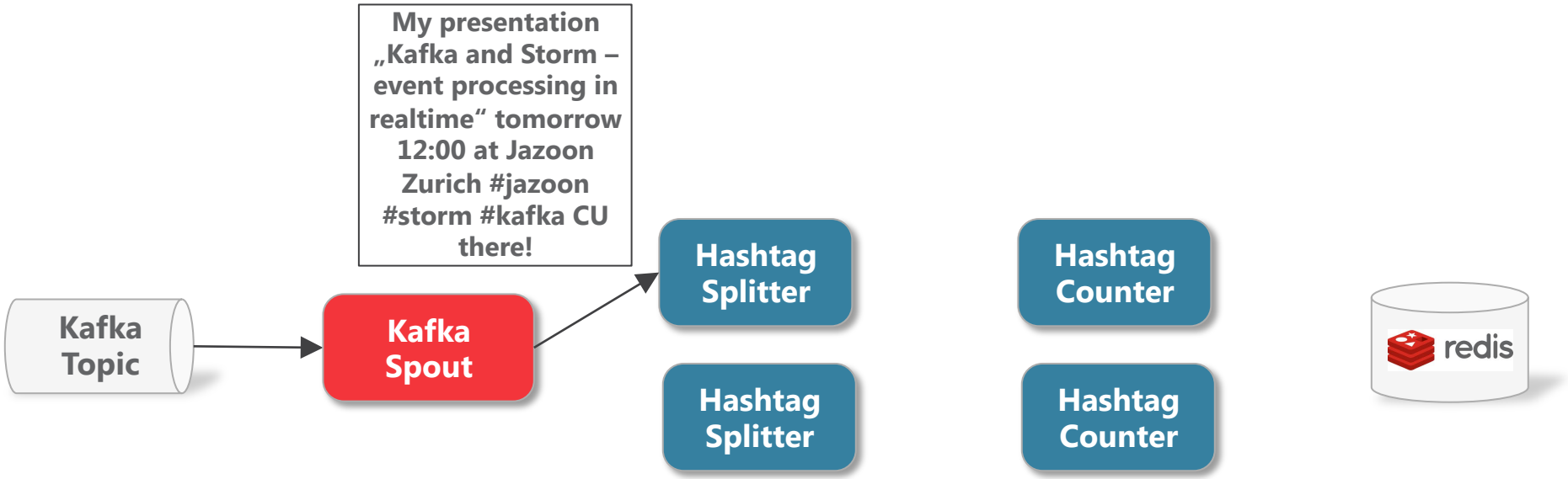
Main Concepts - Topology



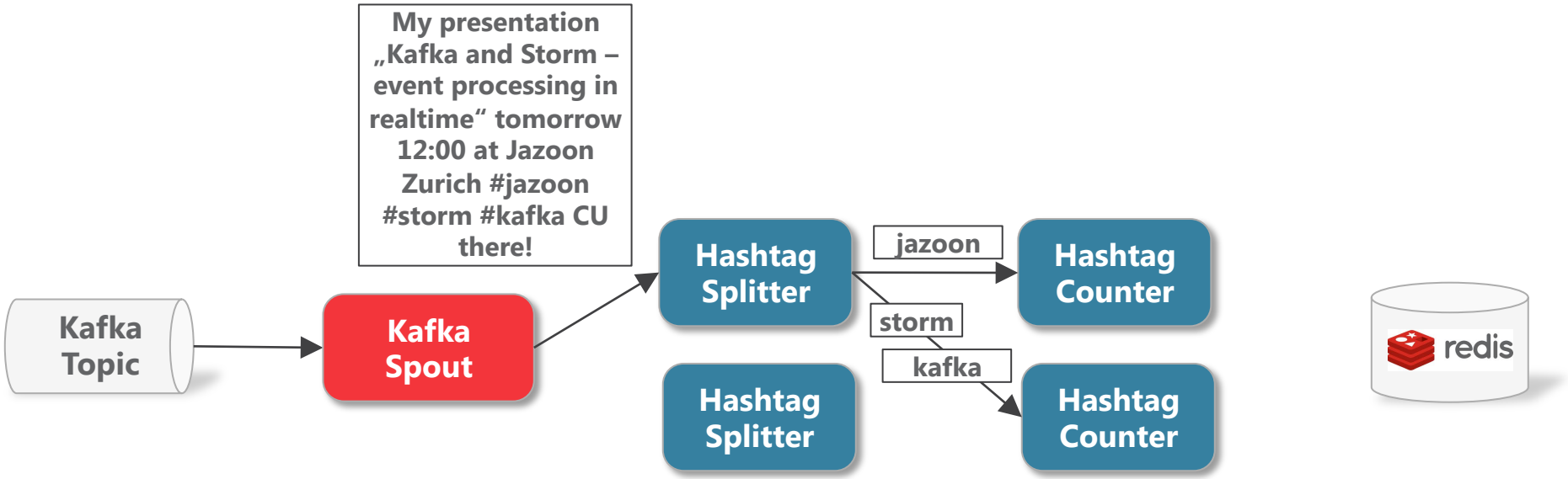
Each Spout or Bolt are running N instances in parallel

Shuffle grouping	is random grouping
Fields grouping	is grouped by value, such that equal value results in equal task
All grouping	replicates to all tasks
Global grouping	makes all tuples go to one task
None grouping	makes bolt run in the same thread as bold/spout it subscribes to
Direct grouping	producer (task that emits) controls which consumer will receive

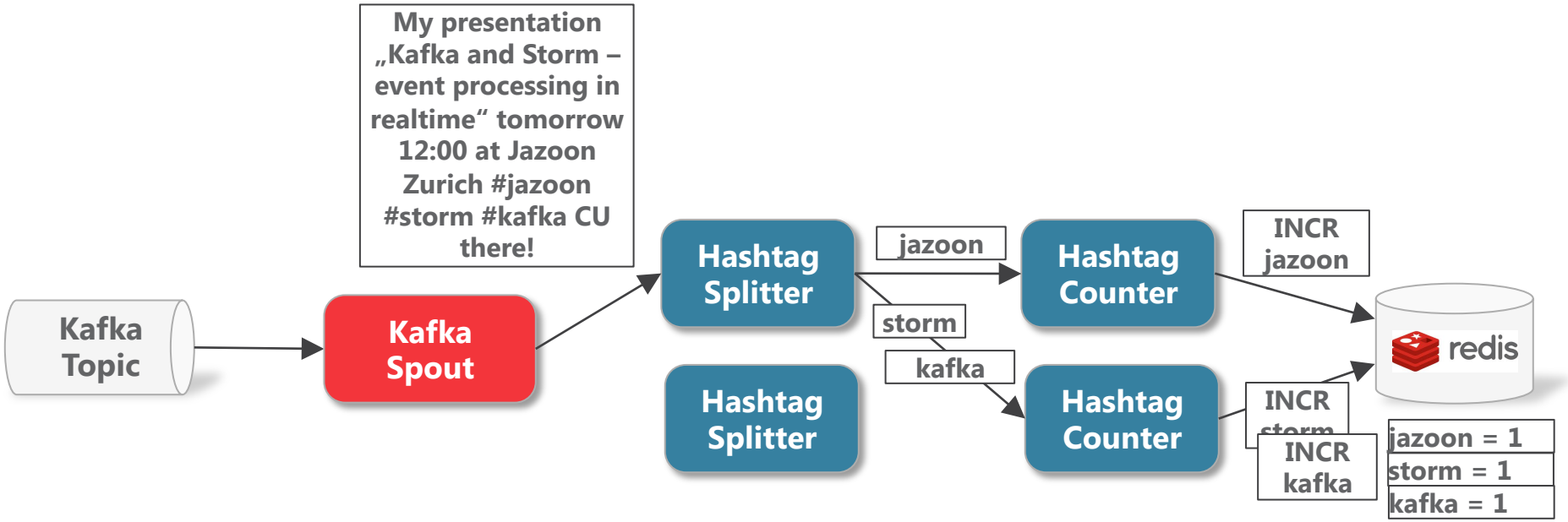
Sample – How does it work ?



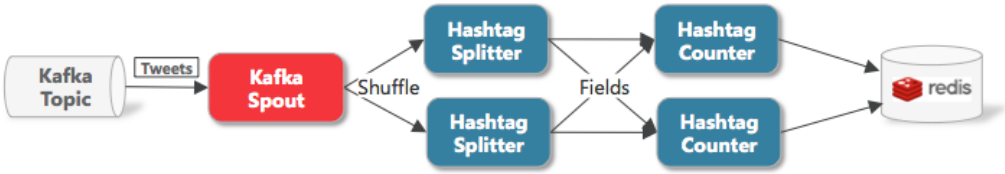
Sample – How does it work ?



Sample – How does it work ?



Sample (#5)



5) Setup the topology with the necessary groupings (distribution)

Create Stream called „tweet-stream“

```
builder.setSpout("tweet-stream", kafkaSpout, 1);

builder.setBolt("hashtag-splitter", new HashtagSplitter(), 2)
    .shuffleGrouping("tweet-stream");

builder.setBolt("hashtag-counter",
                new HashtagCounterRedis("localhost",6379), 2)
    .fieldsGrouping("hashtag-splitter", new Fields("hashtag"));

return builder.createTopology();
```

Register Kafka spout and run by 1 worker

Subscribe to stream „tweet-stream“ using shuffle grouping

Run this bolt by 2 workers

Subscribe to „hashtag-splitter“ using fields grouping on field „hashtag“

Trident

- High-Level abstraction on top of storm
- Makes it easier to build topologies
- Core data model is the stream
  - Processed as a series of batches
  - Stream is partitioned among nodes in cluster
- 5 kinds of operations in Trident
  - **Operations** that apply locally to each partition and cause no network transfer
  - **Repartitioning operations** that don't change the contents
  - **Aggregation operations** that do network transfer
  - Operations on **grouped streams**
  - **Merges** and **Joins**

Trident

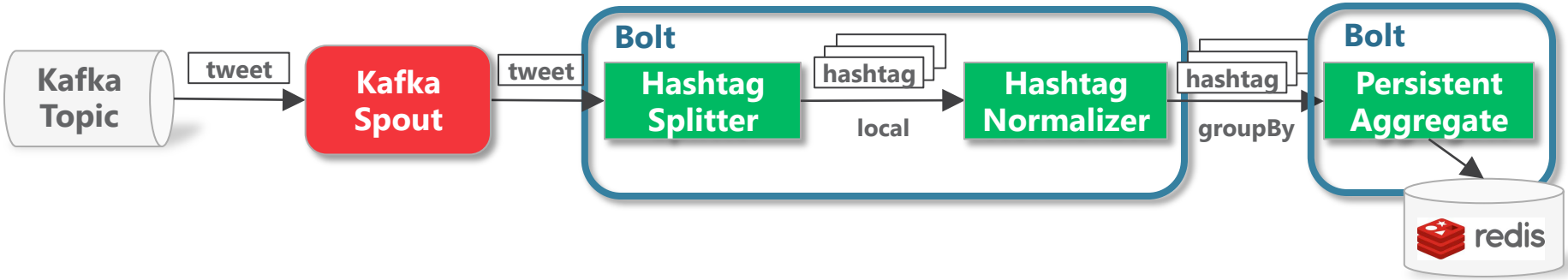
Supports

- Joins
- Aggregations
- Grouping
- Functions
- Filters

Similar to Hadoop and Pig/Cascading

Trident Concepts - Topology

```
public static StormTopology createTopology(String consumerKey,
        String consumerSecret, String token, String secret) {

    StateFactory redis = RedisState.nonTransactional(new InetSocketAddress(
            "localhost", 6379));

    TwitterStreamingSpout spout = new TwitterStreamingSpout(consumerKey,consumerSecret, token, secret,
                                                            new String[] { "#JFS2013" });

    TridentTopology topology = new TridentTopology();
    TridentState wordCounts = topology.newStream("twitter-stream", spout)
            .parallelismHint(1)
            .each(new Fields("tweet"), new HashtagSplitter(), new Fields("hashtag"))
            .each(new Fields("hashtag"), new HashtagNormalizer(), new Fields("hashtagNormalized"))
            .groupBy(new Fields("hashtagNormalized"))
            .persistentAggregate(redis, new Count(), new Fields("count"))
            .parallelismHint(2);

    return topology.build();
}
```



Trident Concepts - Function

- takes in a set of input fields and emits zero or more tuples as output
- fields of the output tuple are appended to the original input tuple in the stream
  - If a function emits no tuples, the original input tuple is filtered out
  - Otherwise the input tuple is duplicated for each output tuple

```
public class HashtagSplitter extends BaseFunction {

    @Override
    public void execute(TridentTuple tuple, TridentCollector collector) {
        Status tweet = (Status)tuple.getValueByField("tweet");
        for (HashtagEntity hashtagEntity : tweet.getHashtagEntities()) {
            collector.emit(new Values(hashtagEntity.getText()));
        }
    }
}
```

```
public class HashtagNormalizer extends BaseFunction {

    @Override
    public void execute(TridentTuple tuple, TridentCollector collector) {
        String hashtag = tuple.getStringByField("hashtag");
        hashtag = StringUtils.lowerCase(hashtag);
        collector.emit(new Values(hashtag));
    }
```

Agenda

1. Introduction
2. What is Apache Kafka?
3. What is Twitter Storm?
4. **Summary**

Summary

Kafka

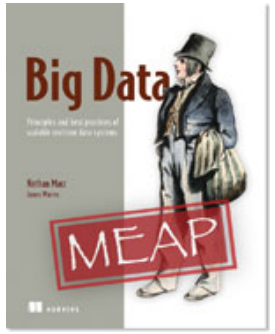
- Distributed Scalable Pub/Sub system for Big Data
- Producer => Broker => Consumer of message topics
- Persists messages with ability to rewind
- Consumer decides what he as consumed so far

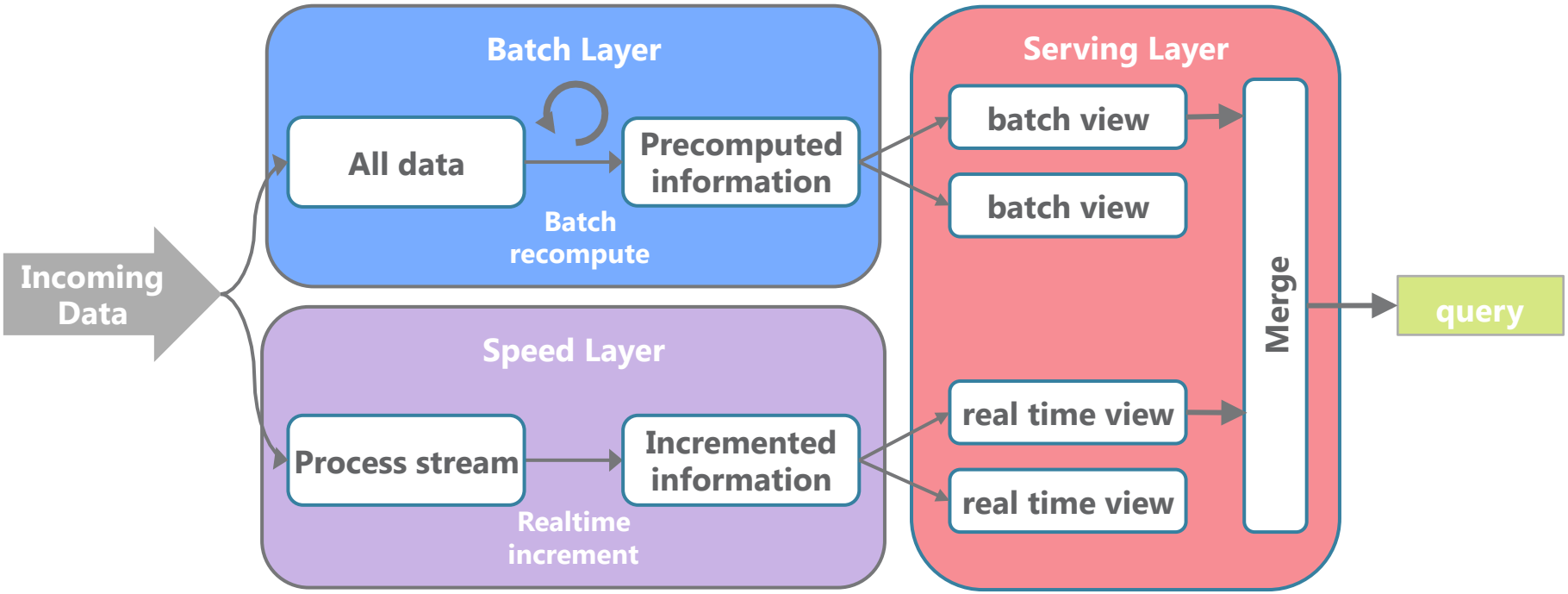
Storm

- Express real-time processing naturally
- Not a Hadoop/MapReduce competitor
- Supports other languages
- Hard to debug
- Object Serialization
- Didn't cover
  - Reliability through guaranteed message processing
  - Distributed RPC
  - Storm cluster setup and deploy

http://54.217.159.208:8484/jazoon-restapi/

Lambda Architecture
Big Data and Fast Data combined

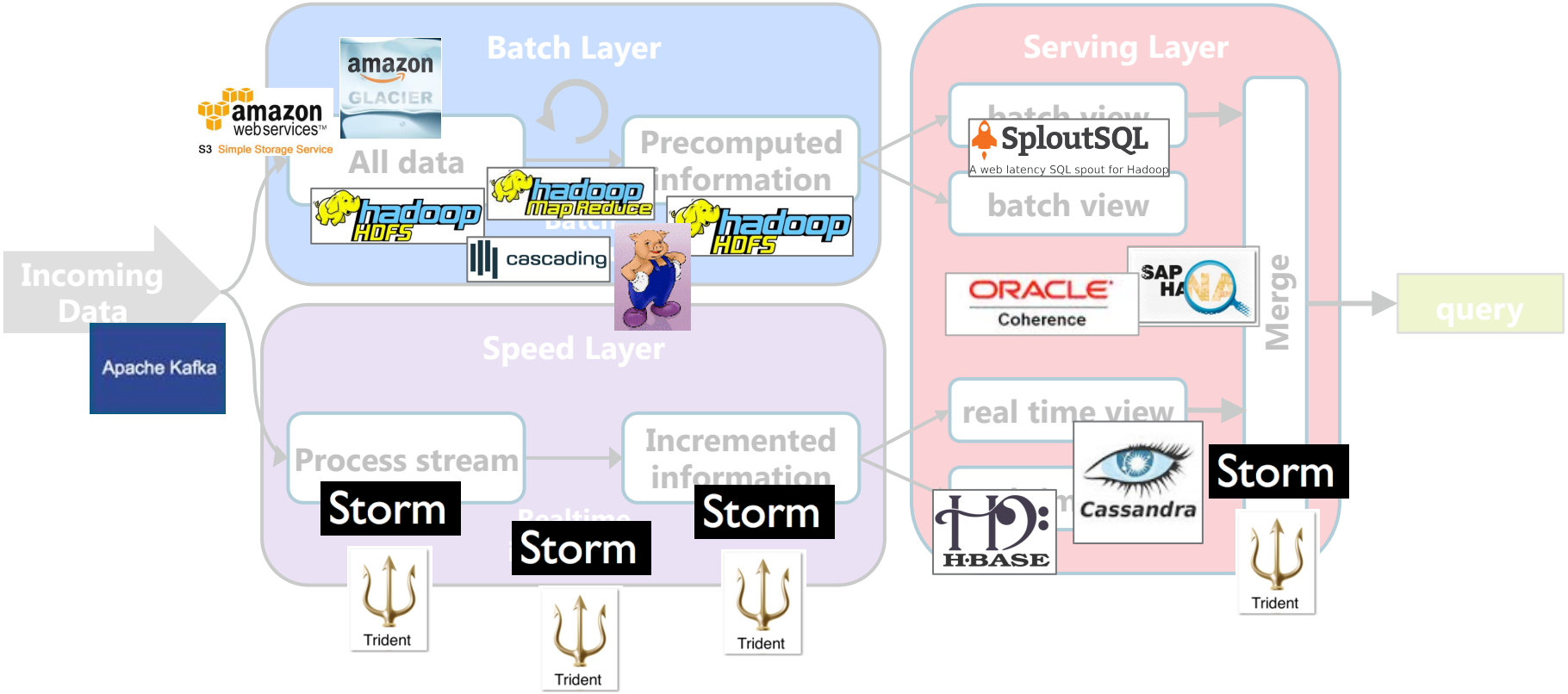




Source: Marz, N. & Warren, J. (2013) Big Data. Manning.

Lambda Architecture
Big Data and Fast Data combined

one possible product/framework mapping



Resources

Sample Code: https://github.com/gschmutz/jazoon-storm-twitter-sample

Twitter Streaming API: https://dev.twitter.com/docs/streaming-apis

Twitter Horsebird Client (HBC): https://github.com/twitter/hbc

Apache Kafka: http://kafka.apache.org/

Storm Website: http://storm-project.net/

Storm Wiki: https://github.com/nathanmarz/storm/wiki

Storm Doc: https://github.com/nathanmarz/storm/wiki/Documentation

Thank You!

Trivadis AG

Guido Schmutz
guido.schmutz@trivadis.com

BASEL BERN LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

Lambda Architecture
Big Data and Fast Data combined