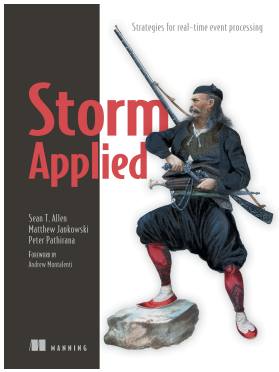
Streaming
&
Apache Storm

Recommended Text:



Storm Applied
Sean T. Allen , Matthew Jankowski, Peter Pathirana
Manning

Big Data

- **Volume**

- **Velocity**
  - Data flowing into the system very fast

Stream Processing

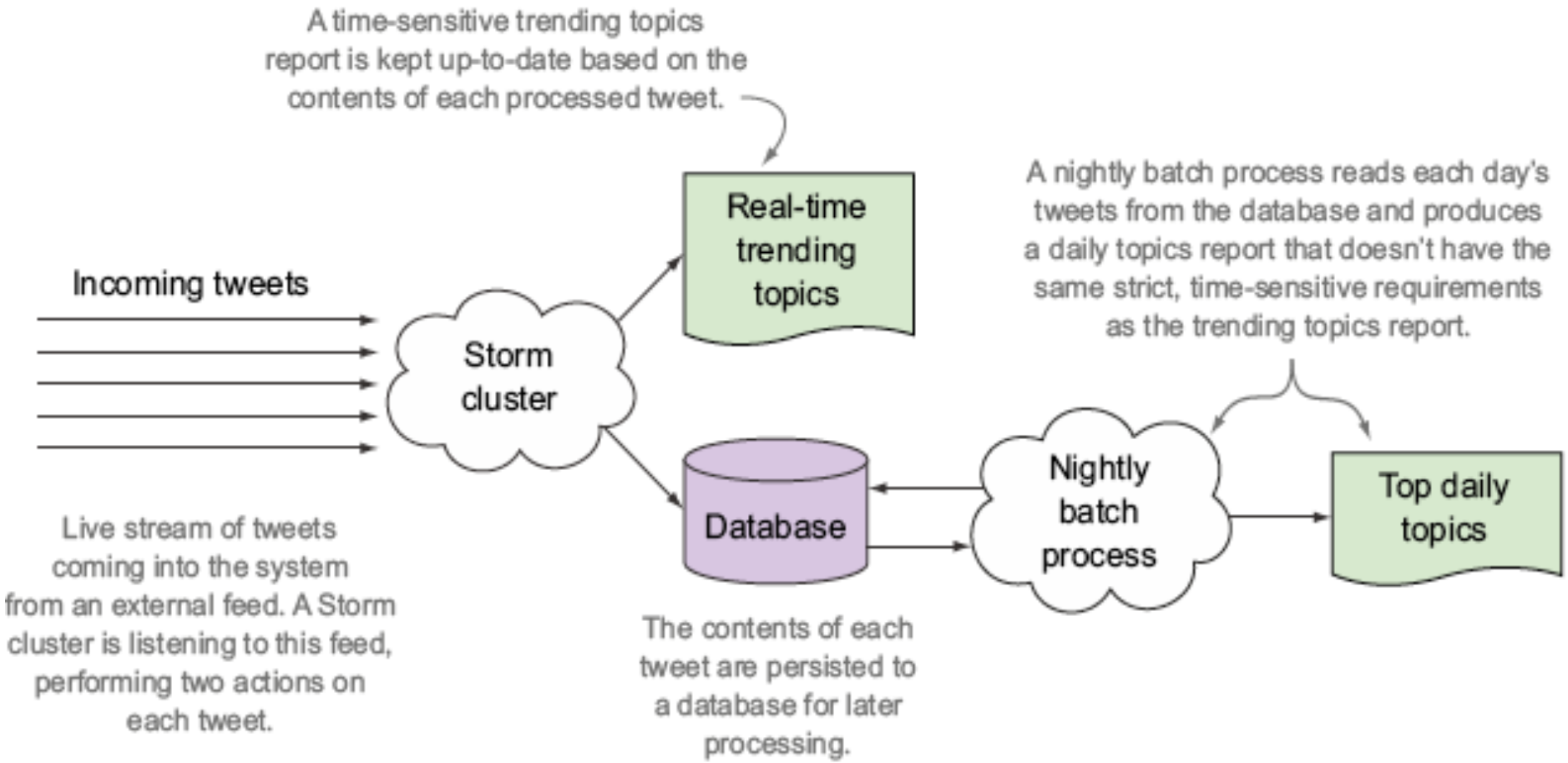
- **A stream processor acts on an unbounded stream of data instead of a batch of data points.**
- **A stream processor is continually ingesting new data (a “stream”).**
- **The need for stream processing usually follows a need for immediacy in the availability of results.**
- **Operate on a single (or small number of) data point(s) at a time**
  - Work on multiple data points in parallel
  - Sub-second-level latency in between the data being created and the results being available.
- **Scenarios:**
  - financial applications, network monitoring, social network analysis, sentiment analysis on tweets, etc.

Apache Storm

- **Distributed, real-time computational framework that makes processing unbounded streams of data easy.**

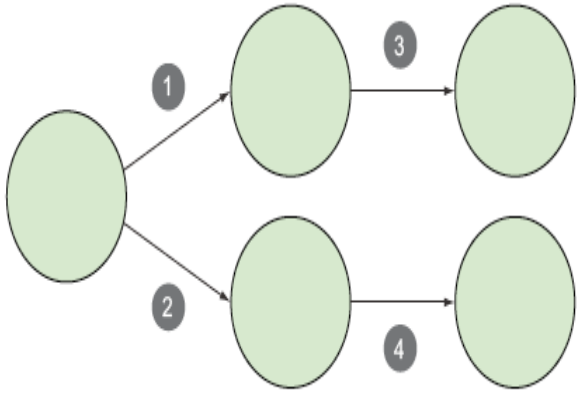
- **Stream-processing tool**
  - Runs indefinitely
  - Listens to a stream of data
  - Does “something” any time it receives data from the stream.

Apache Storm



Storm Concepts

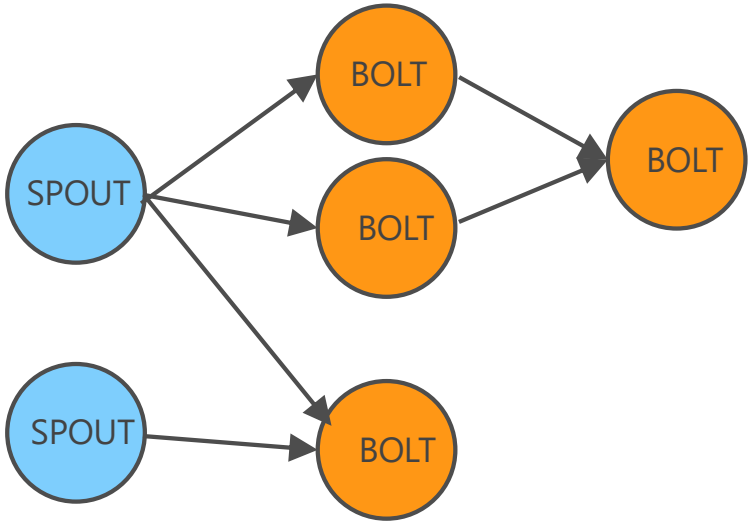
- **Topology:** a graph of computation where the nodes represent some individual computations and the edges represent the data being passed between nodes.



- **Tuple:** A *tuple is an ordered list of values, where each value is assigned a name.* Nodes in the topology send data between one another in the form of tuples.
- **Stream:** An unbounded sequence of tuples between two nodes in the topology.

Storm Concepts

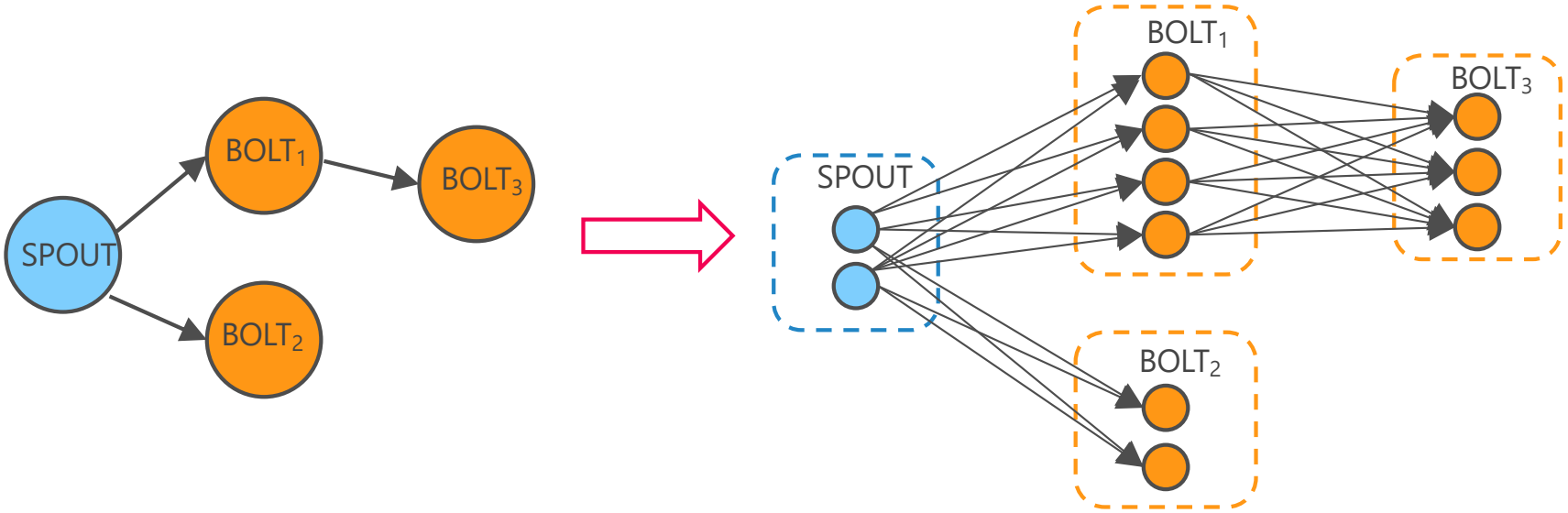
- **Spout:** *Source of a stream in the topology. Read data from an* external data source and emit tuples into the topology.
- **Bolt:** Accepts a tuple from its input stream, performs some computation or transformation—filtering, aggregation, or a join, perhaps—on that tuple, and then optionally emits a new tuple(s).



With permission from : Tiziano De Matteis, "Introduction to Apache Storm," https://www.slideshare.net/tizianodem/introduction-to-apache-storm-55467258

Application Deployment

- When executed, the topology is deployed as a set of processing entities over a set of computational resources (typically a cluster). Parallelism is achieved in Storm by running multiple replicas of the same spout/bolt:



Groupings specify how tuples are routed to the various replicas

Stream Grouping

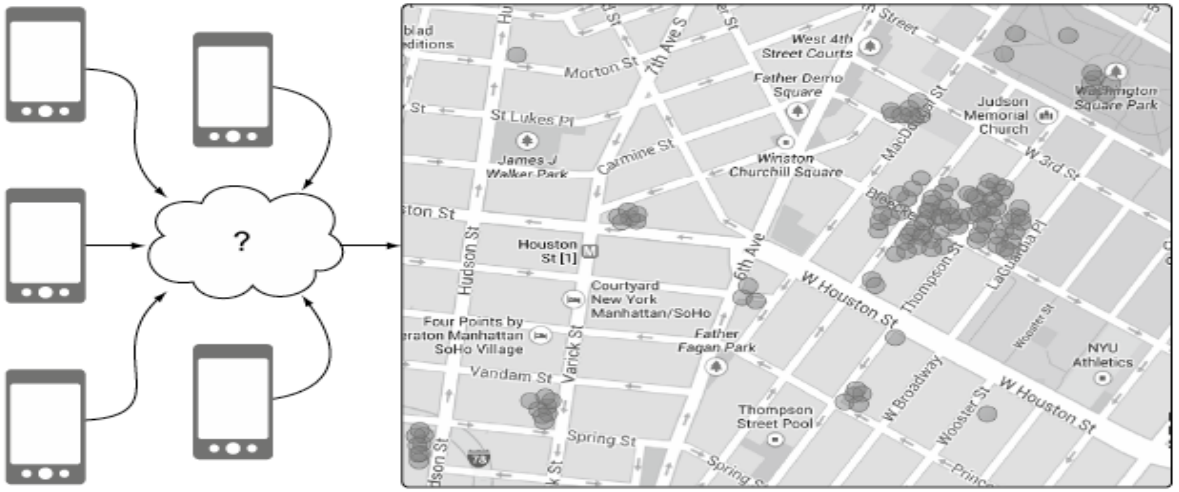
There are 7 built-in possibilities, the most interesting are:

- *shuffle grouping*: tuples are randomly distributed;
- *field grouping*: the stream is partitioned according to a tuple attribute. Tuples with the same attribute will be scheduled to the same replica;
- *all grouping*: tuples are replicated to all replicas;
- *direct grouping*: the producer decides the destination replica
- *global grouping*: all the tuples go to the same replica (low. ID).

Users have also the possibility of implementing their own grouping through the `CustomStreamGrouping` interface

Example: Heat map

- **Goal: Create a geographical map with a heat map overlay identifying neighborhoods with the most popular bars.**



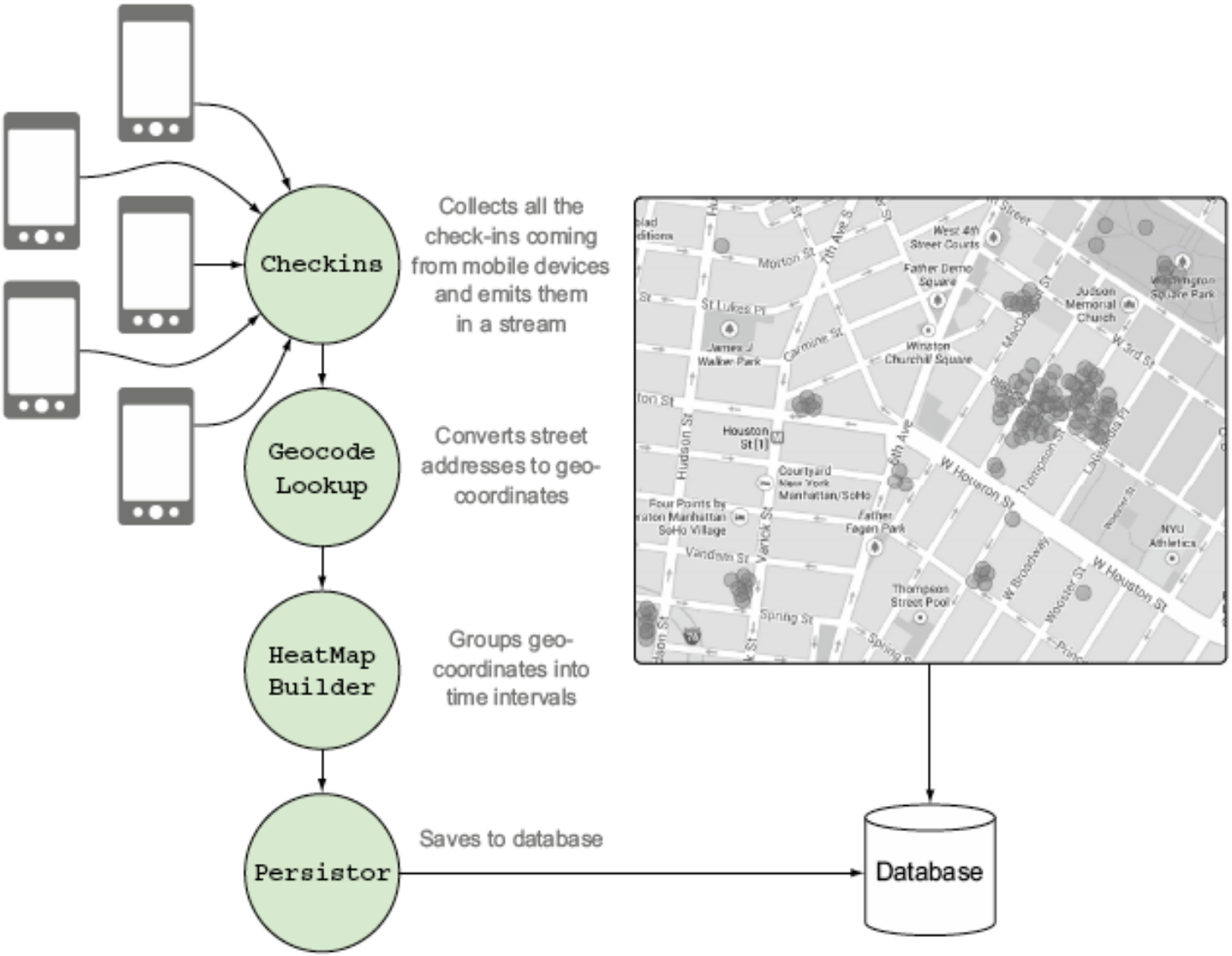
- **Input: Social network check-ins**

```
[time="9:00:07 PM", address="287 Hudson St New York NY 10013"]
```

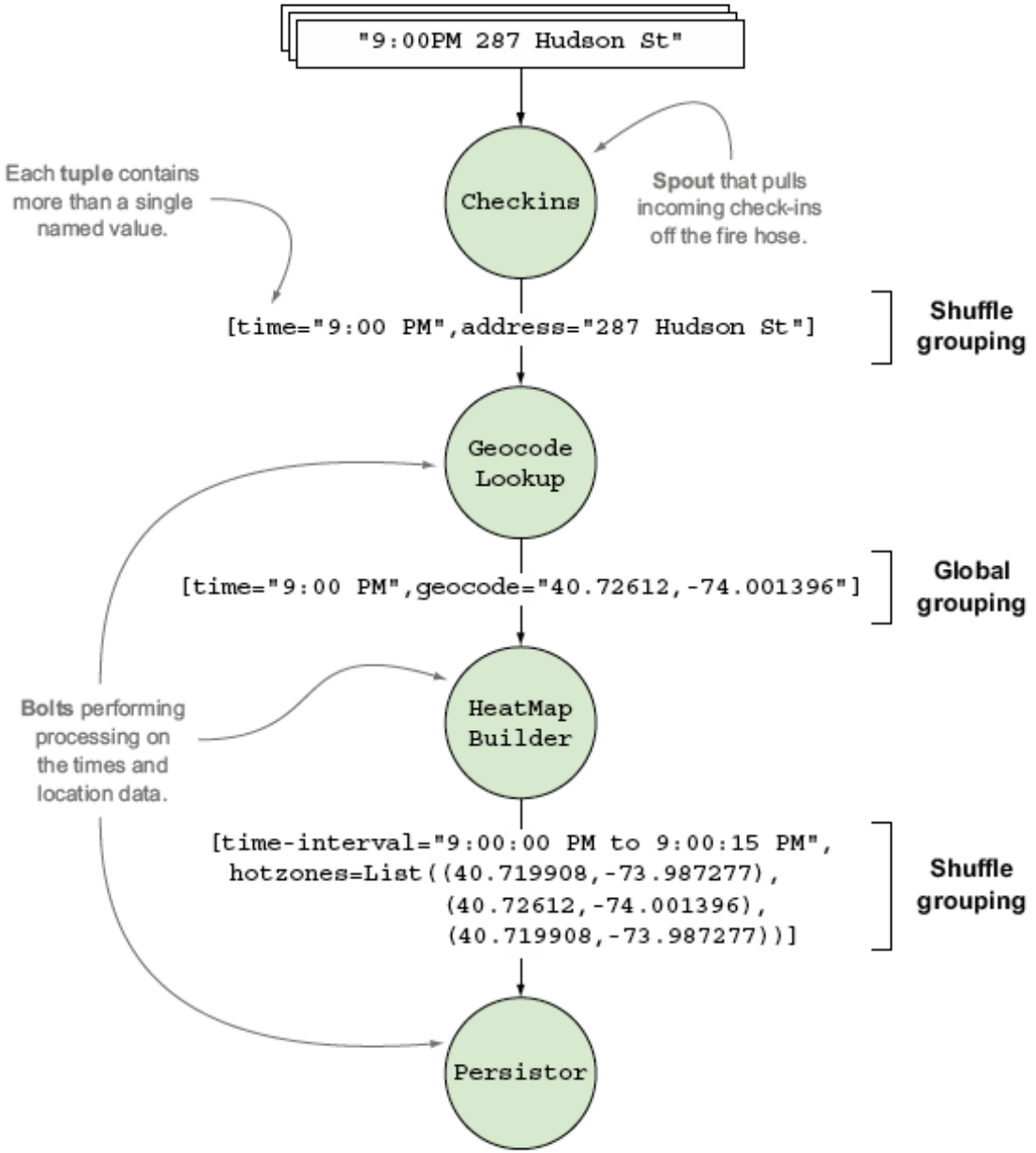
- **Output: Time interval with list of coordinates**

```
[time-interval="9:00:00 PM to 9:00:15 PM",
 hotzones=List((40.719908,-73.987277),
               (40.72612,-74.001396),
               (40.719908,-73.987277))]
```

Example: Heat map

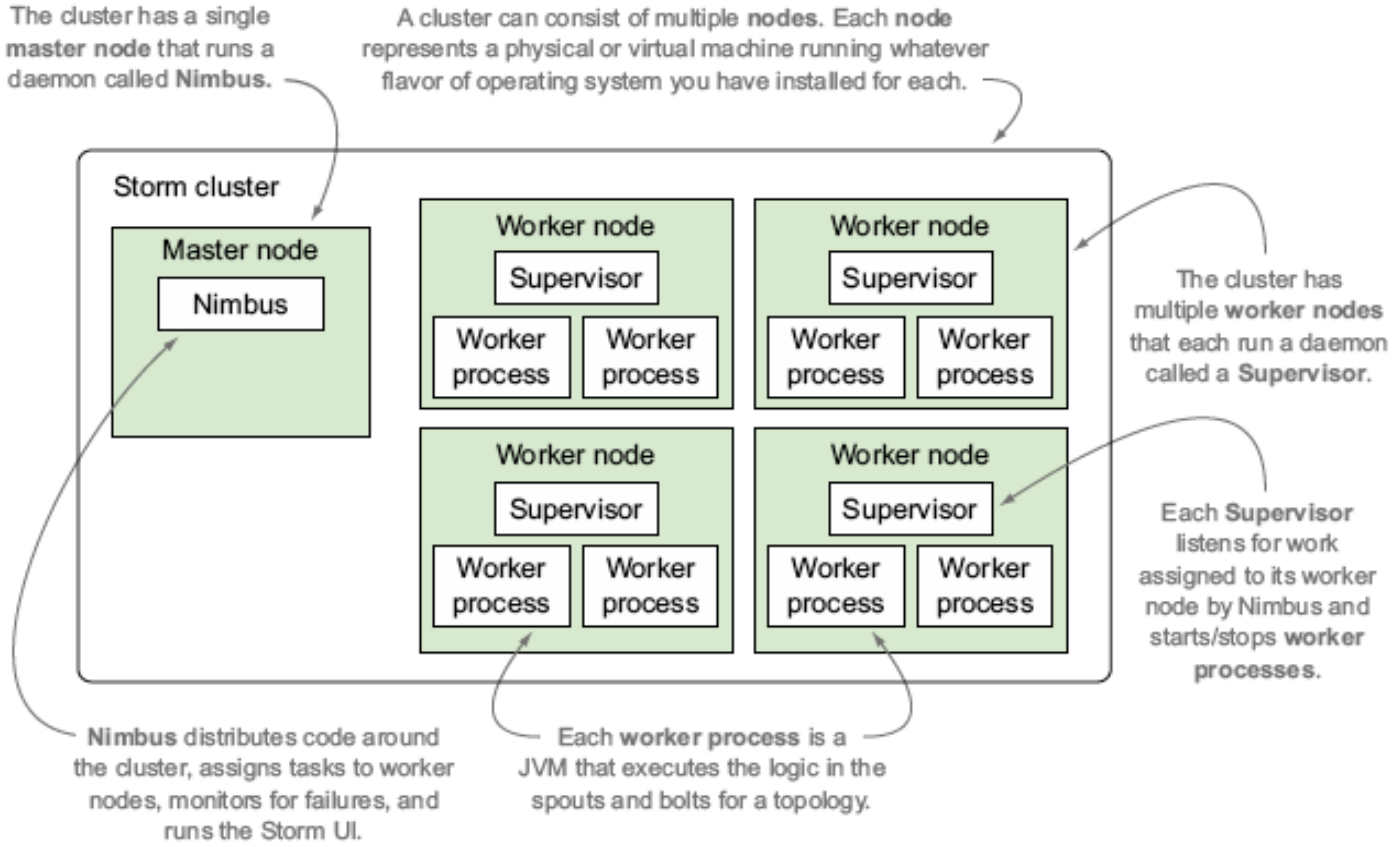


Example: Heat map



Strom Architecture

- **Master node: runs the *Nimbus*, a central job master to which topologies are submitted . It is in charge of scheduling, job orchestration, communication and fault tolerance;**
- **Worker nodes: nodes of the cluster in which applications are executed. Each of them run a Supervisor.**



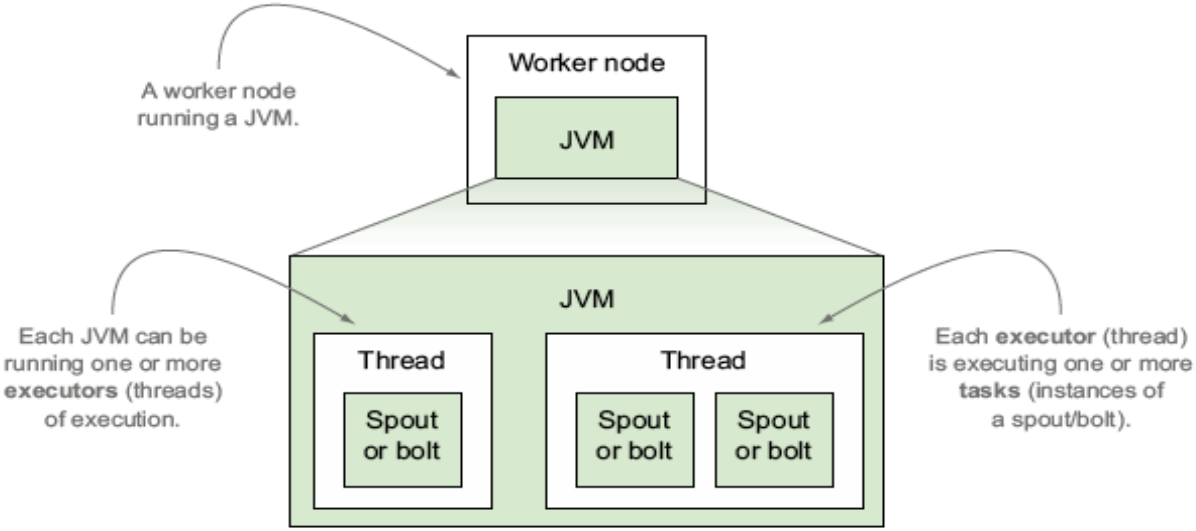
Master and workers coordinate through *Zookeper.*

Strom Architecture

Three entities are involved in running a topology:

- Worker: 1+ per cluster node, each one is related to one topology;
- Executor: thread spawned by the Worker. It runs one or more tasks for the same component (bolt or spout);
- Task: a component replica.

.



By default there is a 1:1 association between Executor and Tasks

```
builder.setBolt("split-bolt", new SplitSentenceBolt(),2).setNumTasks(4)
        .shuffleGrouping("sentences-spout");
```

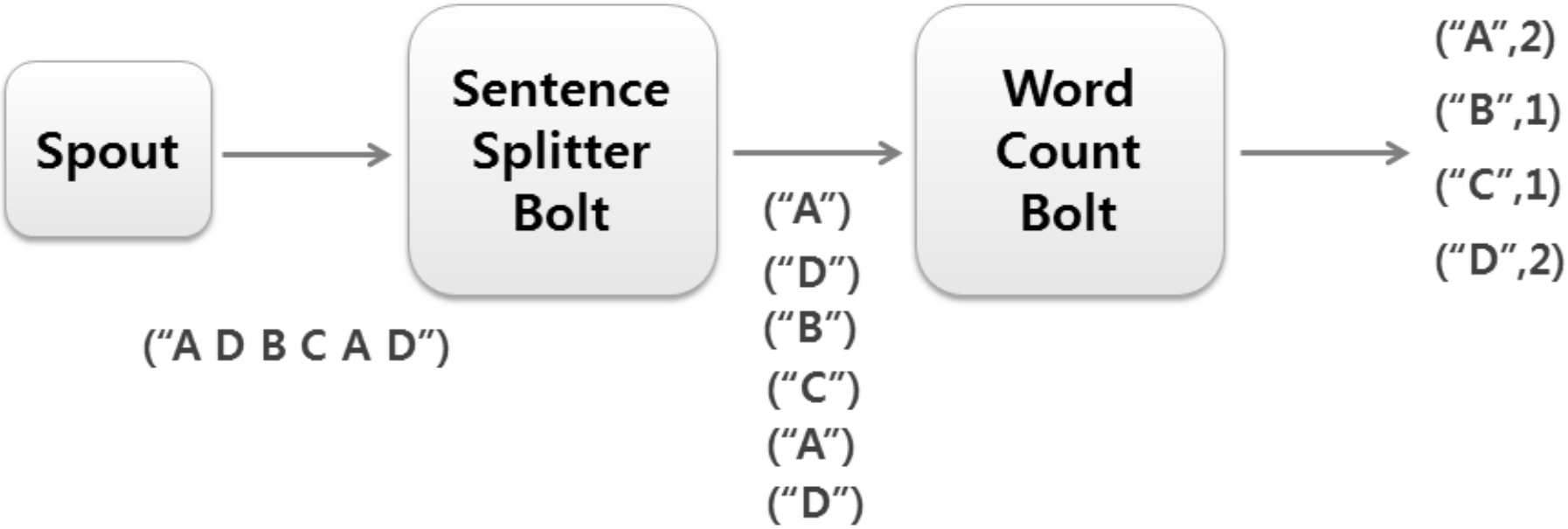
With permission from : Tiziano De Matteis, "Introduction to Apache Storm," https://www.slideshare.net/tizianodem/introduction-to-apache-storm-55467258

streamparse

- A framework for storm applications written in python
- https://streamparse.readthedocs.io/en/stable/index.html
- Classes
  - streamparse.Bolt
  - streamparse.Sprout
  - streamparse.Topology

- Commands
  - Deploy sparse run
  - Undeploy sparse kill

Example: Word Count



Example: Word Count

```
from streamparse import Grouping, Topology

from bolts.split import SplitBolt
from bolts.count import CountBolt
from spouts.sentences import SentencesSpout


class WordCount(Topology):
    sentences_spout = SentencesSpout.spec()
    split_bolt = SplitBolt.spec(inputs=[sentences_spout], par=2)
    count_bolt = CountBolt.spec(inputs={split_bolt: Grouping.fields("word")}, par=2)
```

Example: Word Count

```
from itertools import cycle

from streamparse import Spout


class SentencesSpout(Spout):
    outputs = ["sentence"]
    count = 0

    def initialize(self, stormconf, context):
        self.sentences = cycle(["To everything turn, turn, turn",
                                "There is a season turn, turn, turn",
                                "And a time to every purpose",
                                "Under heaven"])

    def next_tuple(self):
        if self.count < 10:
            self.count = self.count + 1
            sentence = next(self.sentences)
            self.emit([sentence])
```

Example: Word Count

```
import os
from streamparse import Bolt

class SplitBolt(Bolt):
    outputs = ["word"]

    def process(self, tup):
        words = tup.values[0].split()
        for word in words:
            self.emit([word])
```

Example: Word Count

```
import os
from collections import Counter

from streamparse import Bolt

class CountBolt(Bolt):
    outputs = ["word2", "count"]

    def initialize(self, conf, ctx):
        self.counter = Counter()
        self.pid = os.getpid()
        self.total = 0

    def _increment(self, word, inc_by):
        self.counter[word] += inc_by
        self.total += inc_by

    def process(self, tup):
        word = tup.values[0]
        self._increment(word, 1)
        self.logger.info("CountBolt {} {} pid={}".format(word, self.counter[word], self.pid))
```