Parallelizing Data Processing Using Storm Components

Swetha Kolalapudi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Understand how a Storm cluster works

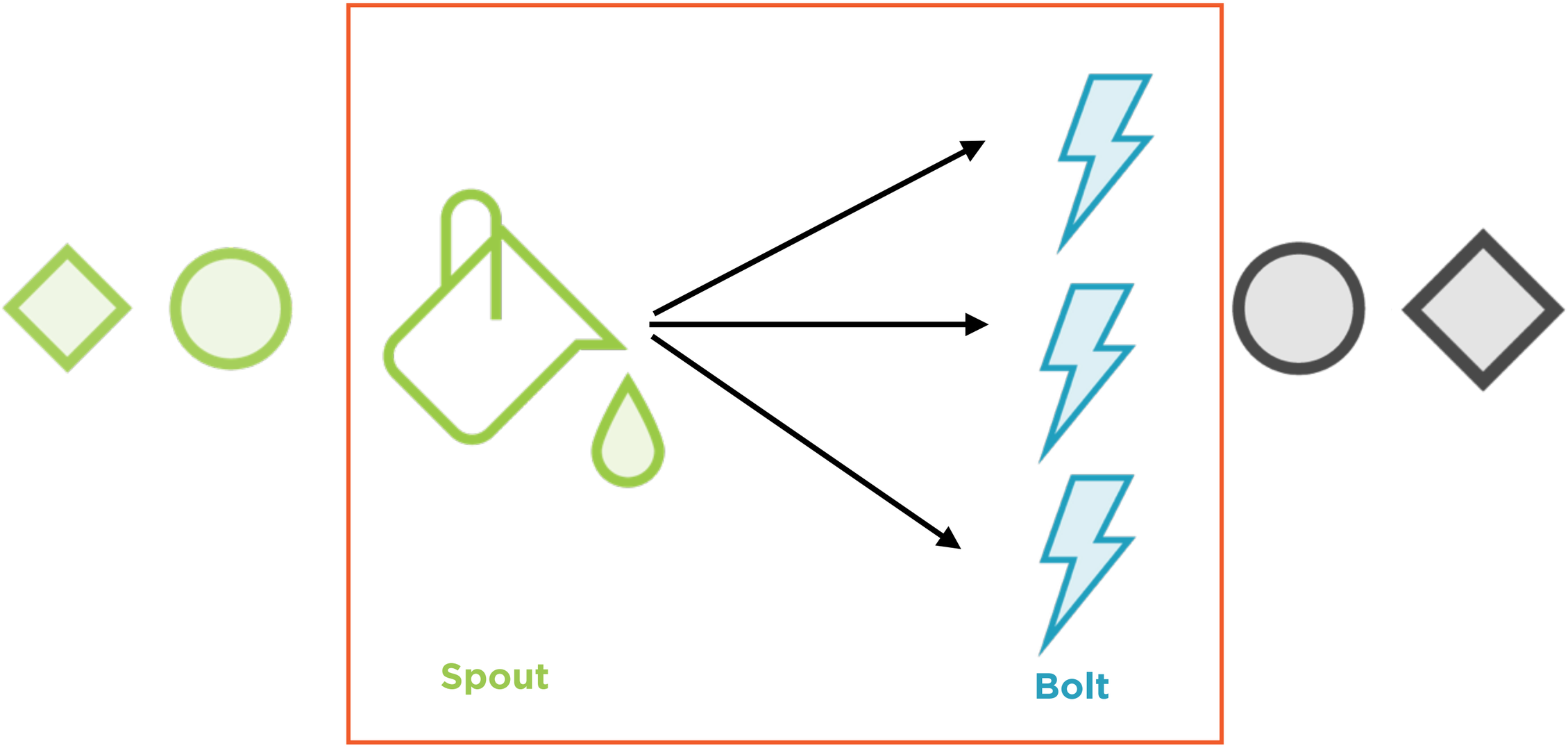
Set up a remote cluster and run a topology

Control data flow using stream grouping

Implement a custom grouping strategy

Stock Price Tracker

Stock Price Tracker



```
builder.setBolt("Bolt", new yfBolt())
       .shuffleGrouping("Spout")
```

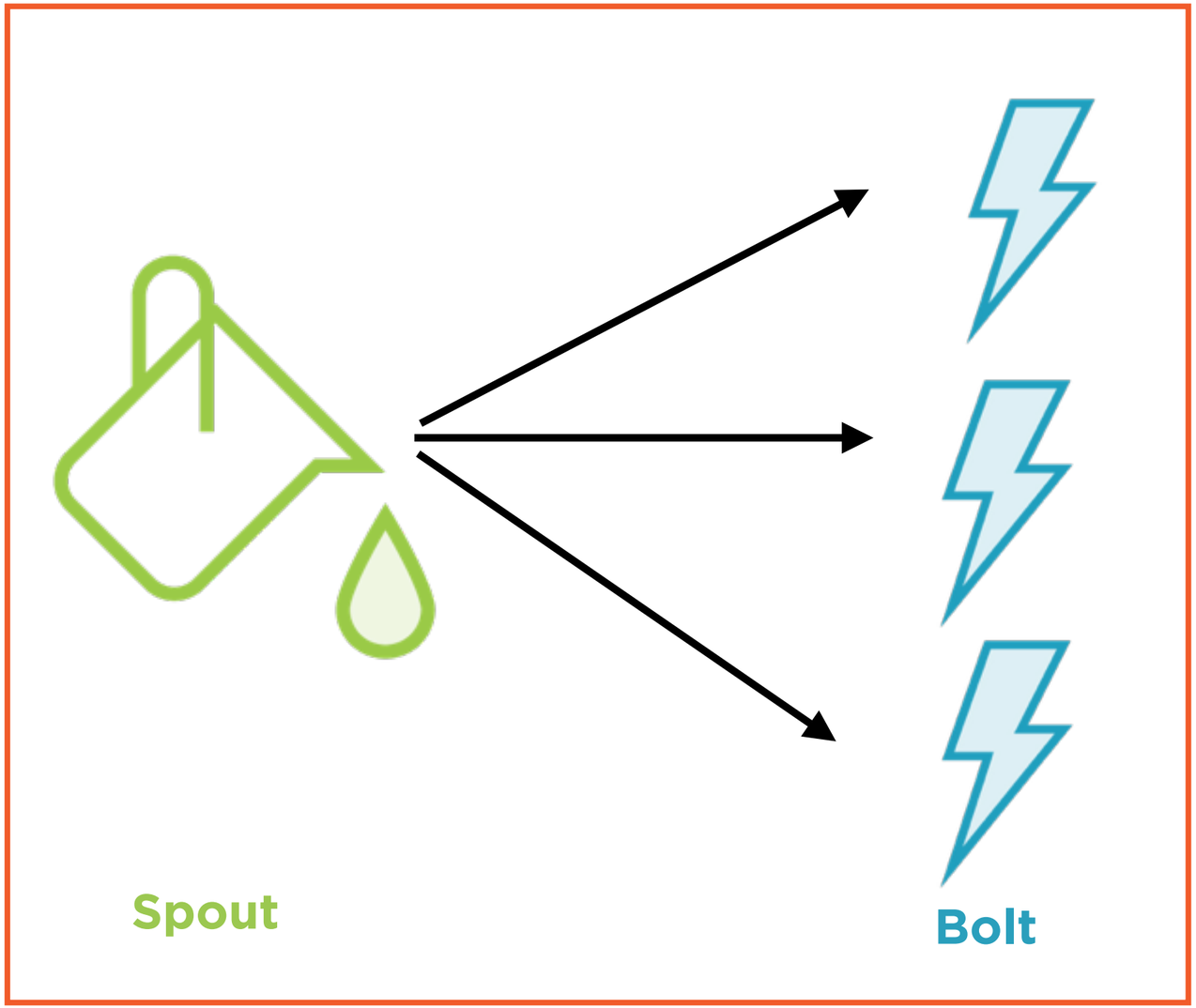
Adding a Bolt

```
builder.setBolt("Bolt", new yfBolt(),3)
       .shuffleGrouping("Spout")
```

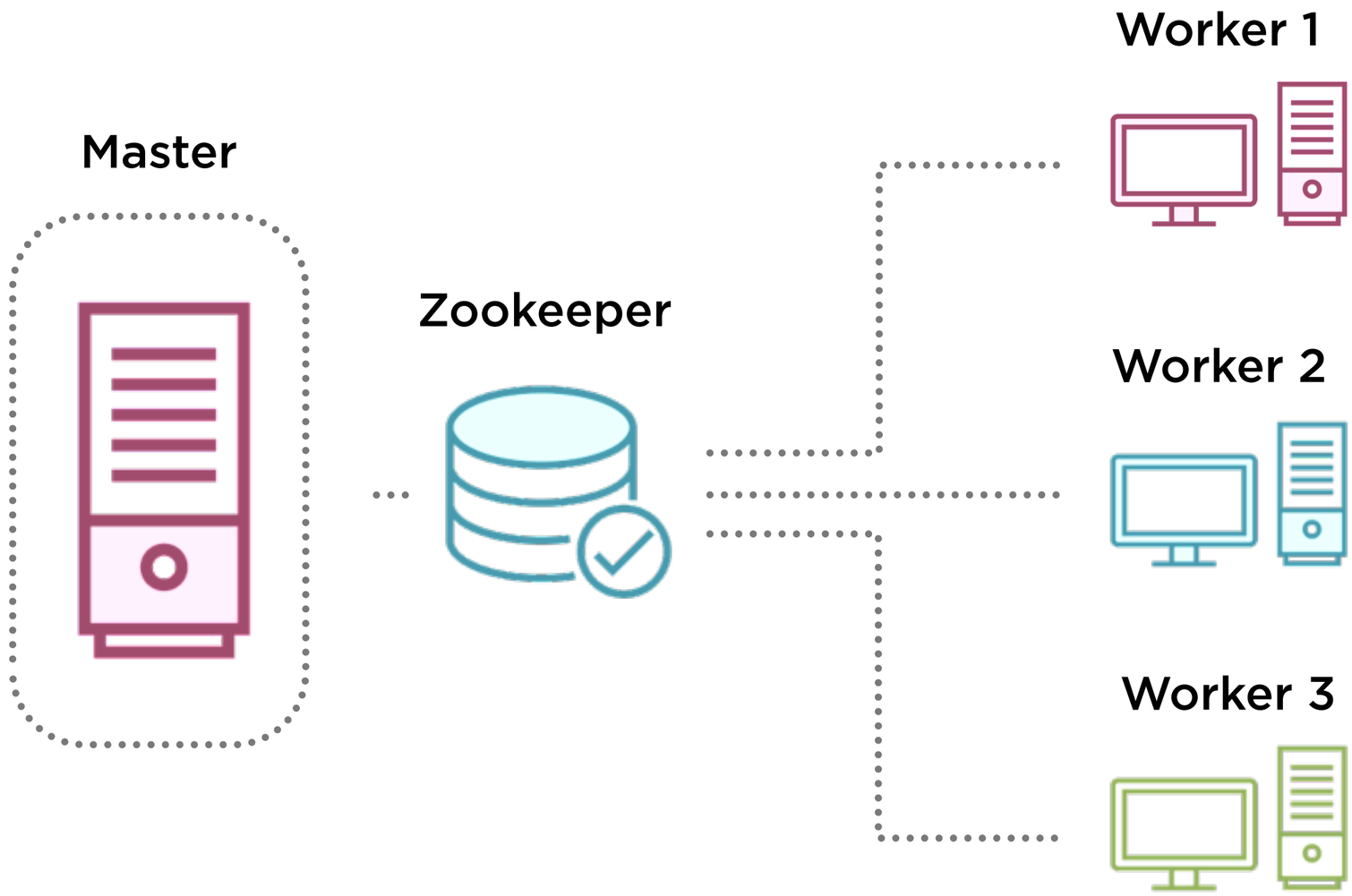
Parallelizing the Bolt

Each component instance is a task

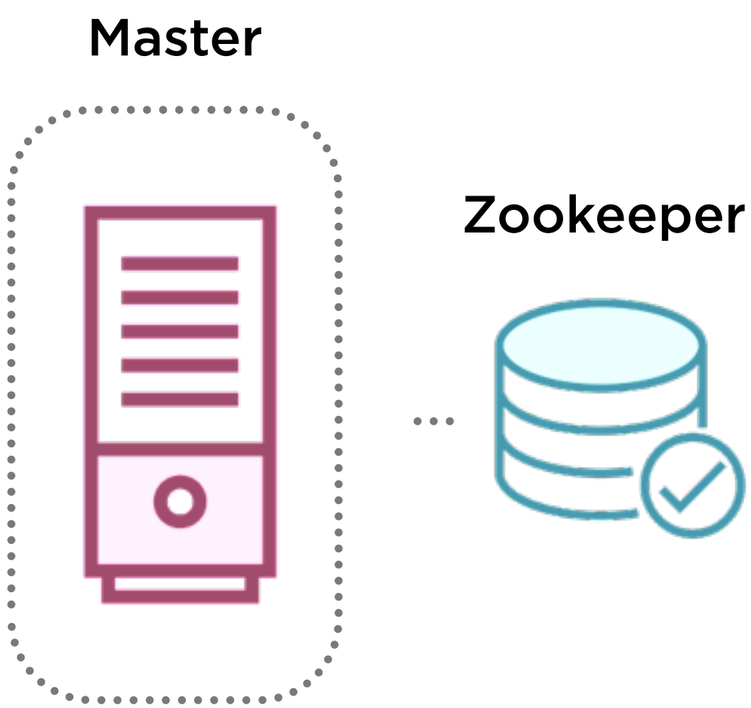
Tasks are launched and managed by a cluster



Storm Cluster



Nimbus



- Central co-ordinator
- Runs the topology
- Identifies the tasks
- Distributes tasks to workers

Supervisor

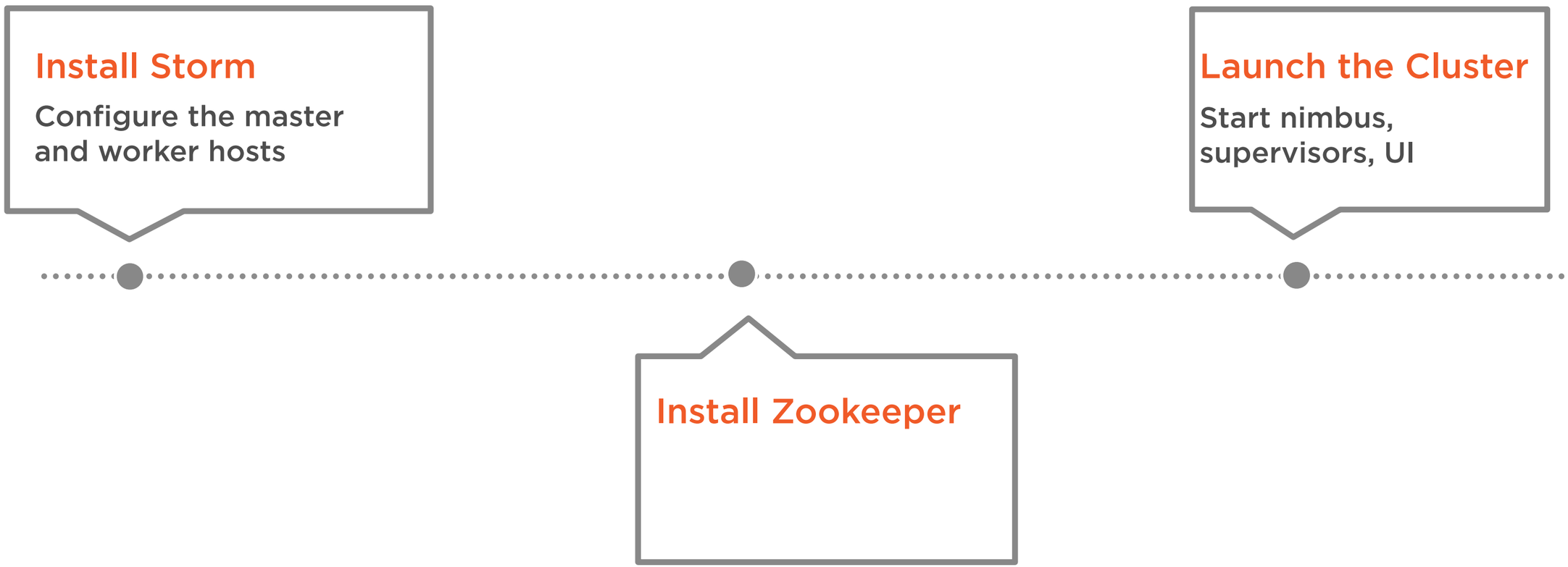
Worker

Spawns **executors** to run the tasks

Each executor is a single thread dedicated to a task

Setting Up a Remote Cluster

Remote Cluster Setup

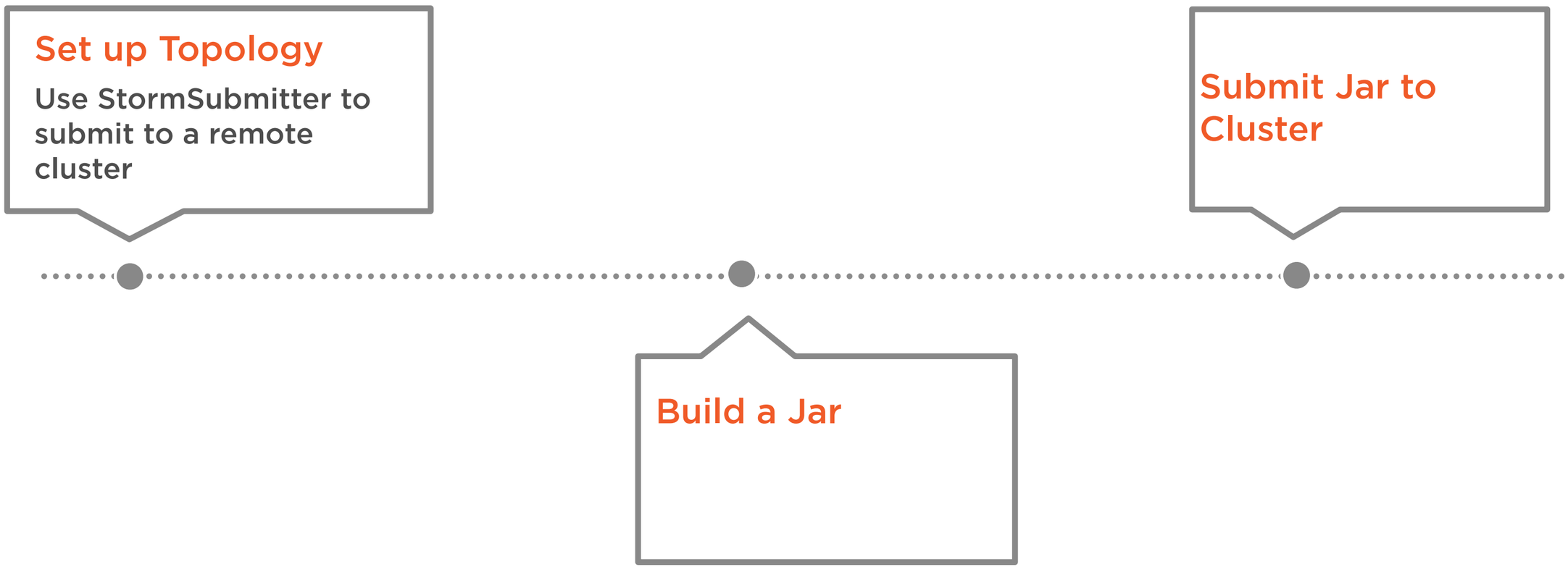


Demo

Set up and launch a remote cluster

Running a Remote Topology

Run Remote Topology



Run Remote Topology

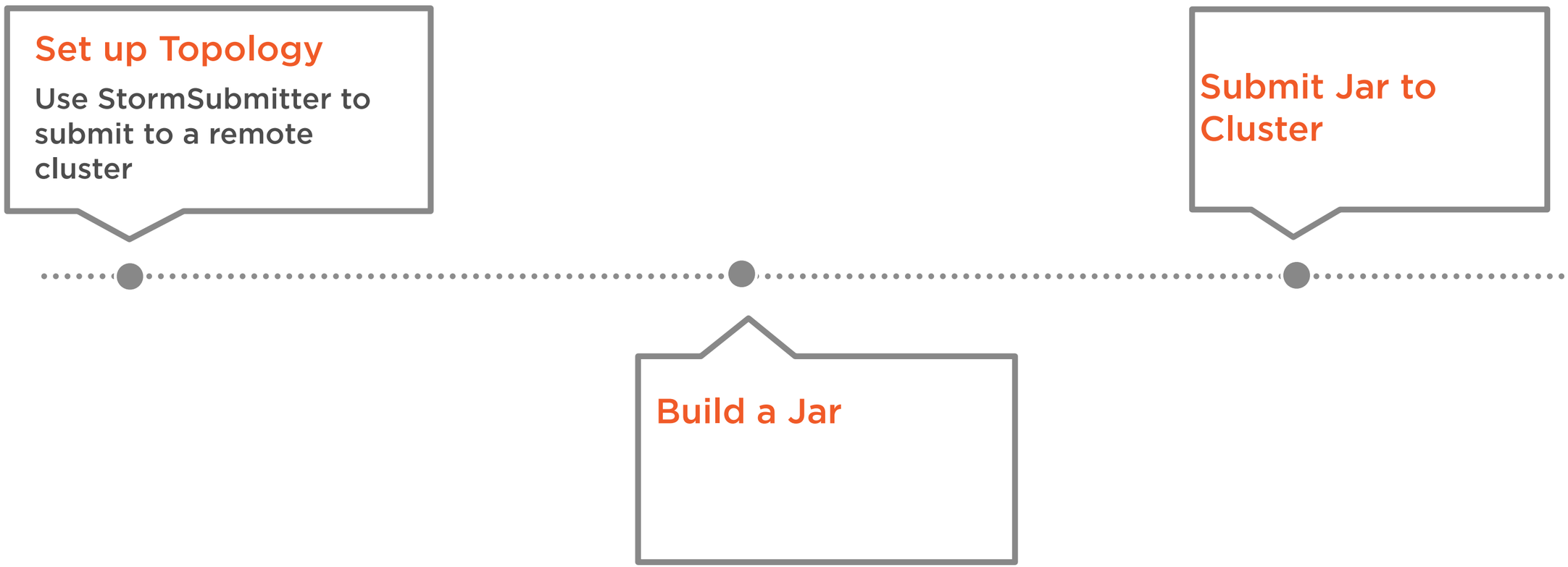
Set up Topology

Use StormSubmitter to submit to a remote cluster

```
StormSubmitter.submitTopology(“T-ID", conf, builder.createTopology());
```

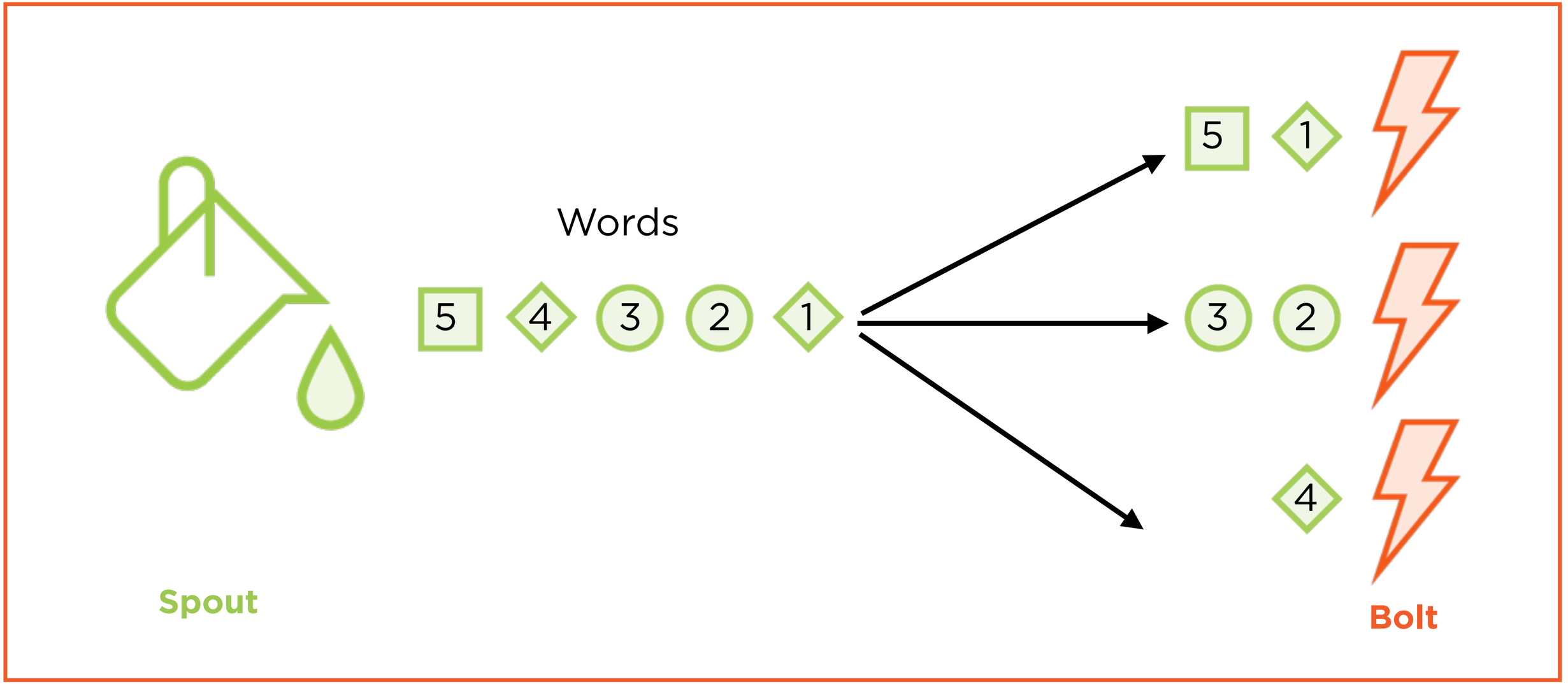
Using StormSubmitter

Run Remote Topology



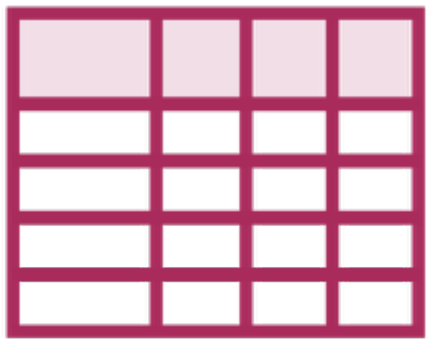
Controlling Data Flow Using Stream Grouping

Word Processor



Stream Grouping

Shuffle

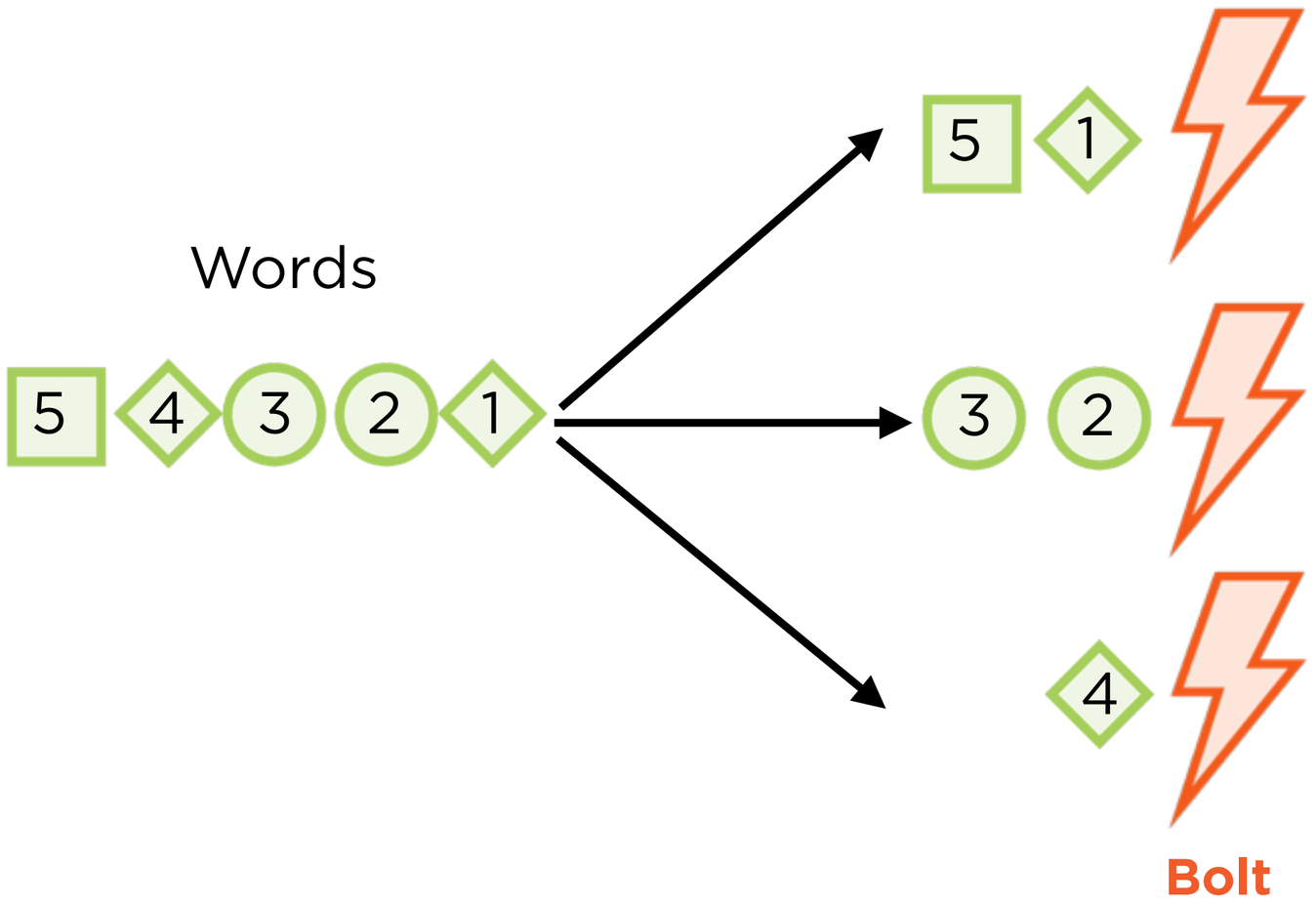


Fields

All

Custom

Shuffle

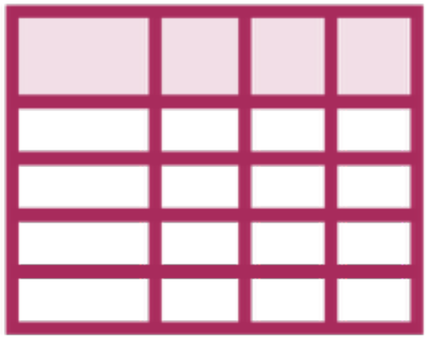


Shuffle

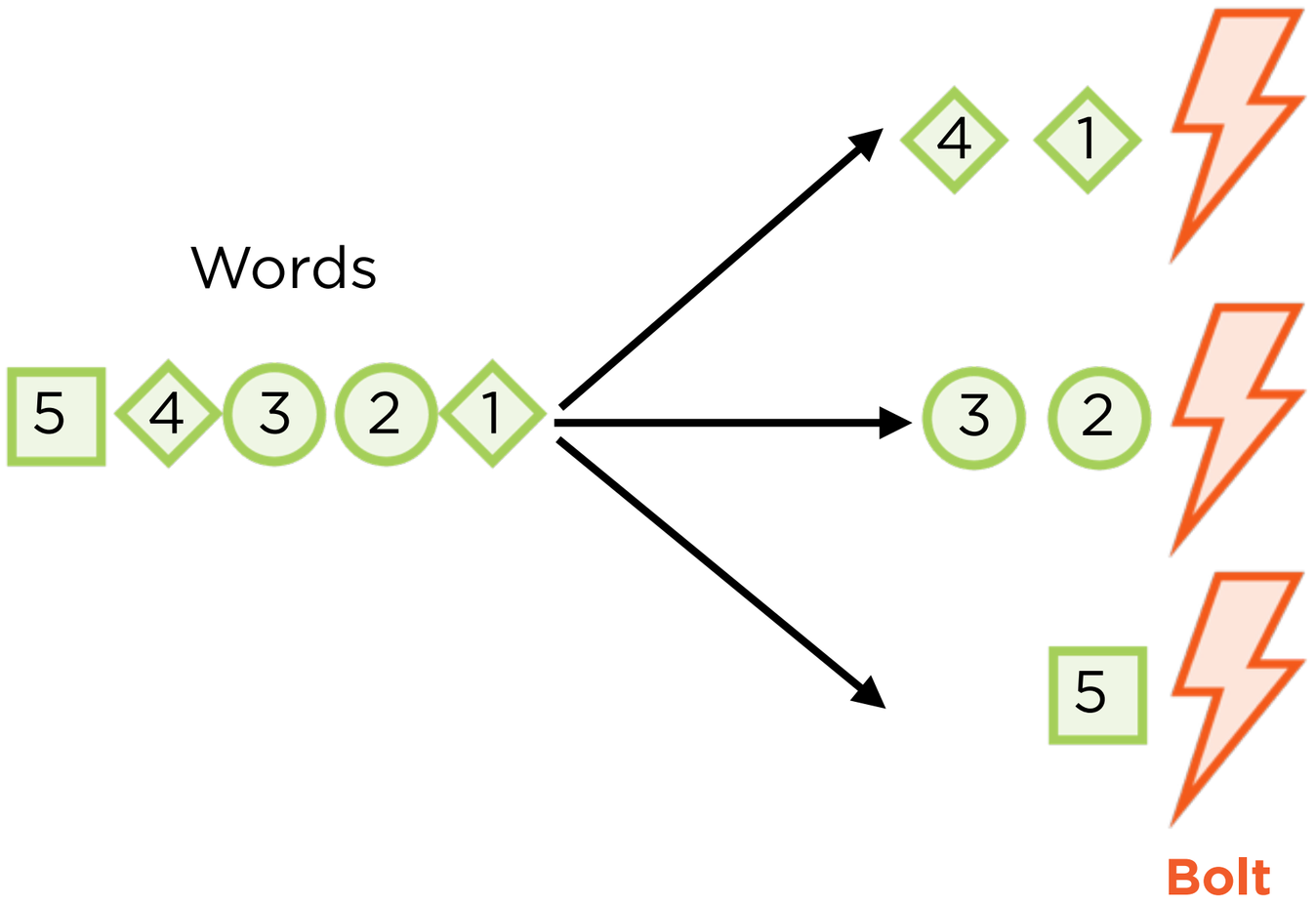
Default grouping strategy

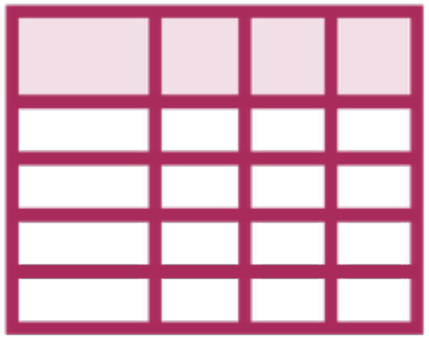
Task id chosen at random

Distributes workload evenly



Fields





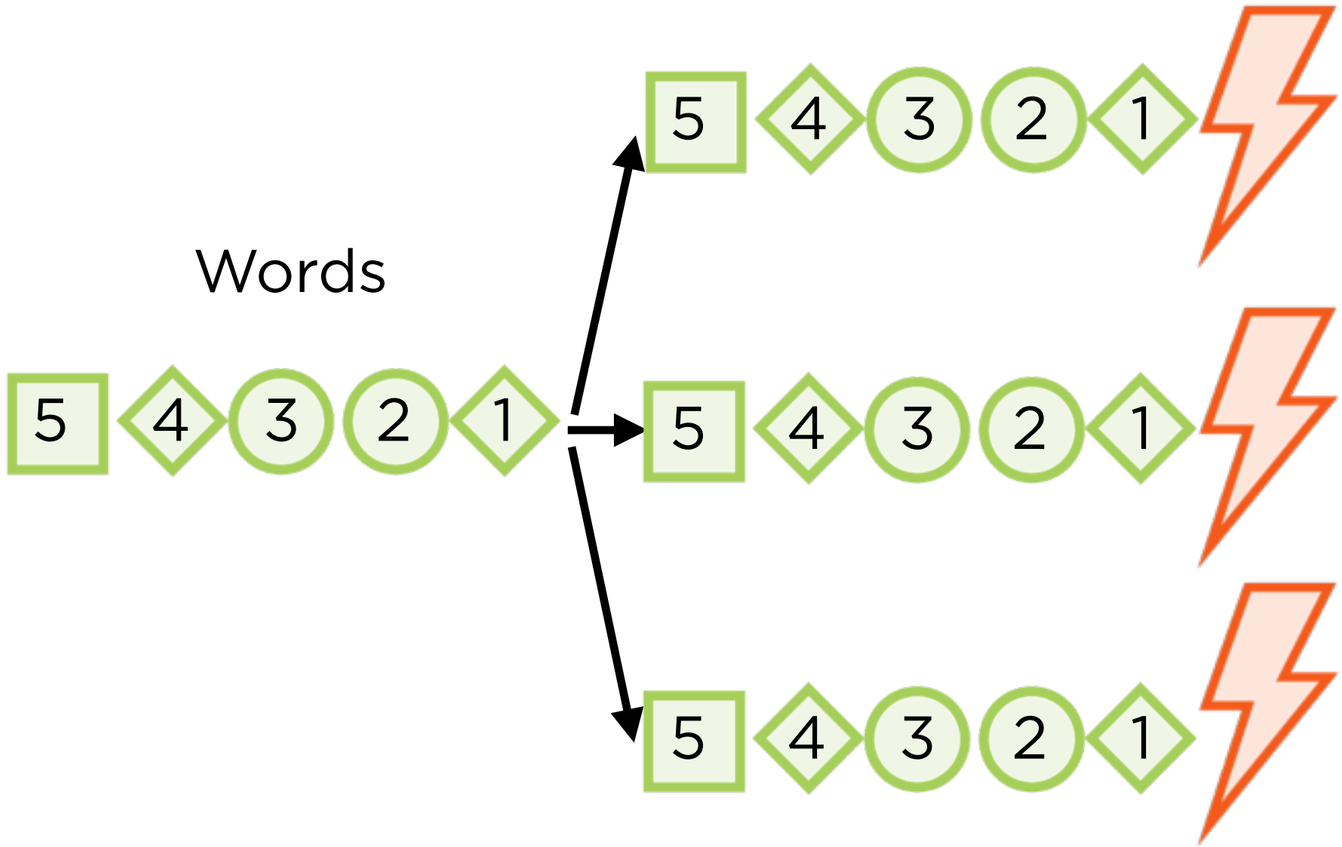
Fields

Task chosen based on the value of specified fields

All tuples with same value sent to the same task

Use for aggregating by field i.e. sum, count, min etc

All



All

All tuples sent to all tasks

Send a signal to all tasks

- Ex : Clear cache

Custom

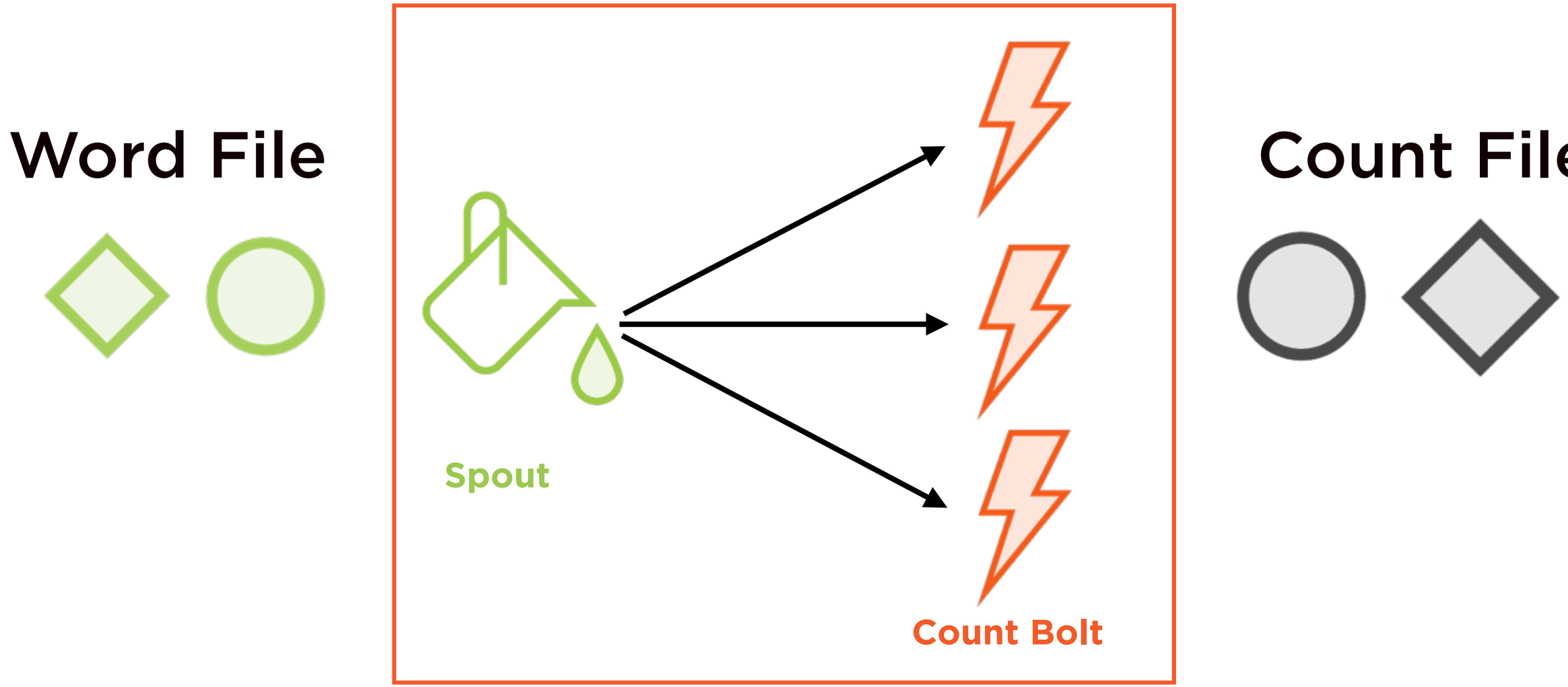
Implement your own custom grouping

- Ex : Words which begin with “a” sent to 1 task

Onus of load balancing shifts to user

Building a Word Count Topology

Word Count Topology



Spout

Read lines from a file

Each line is a single word

Emit each word as a Tuple

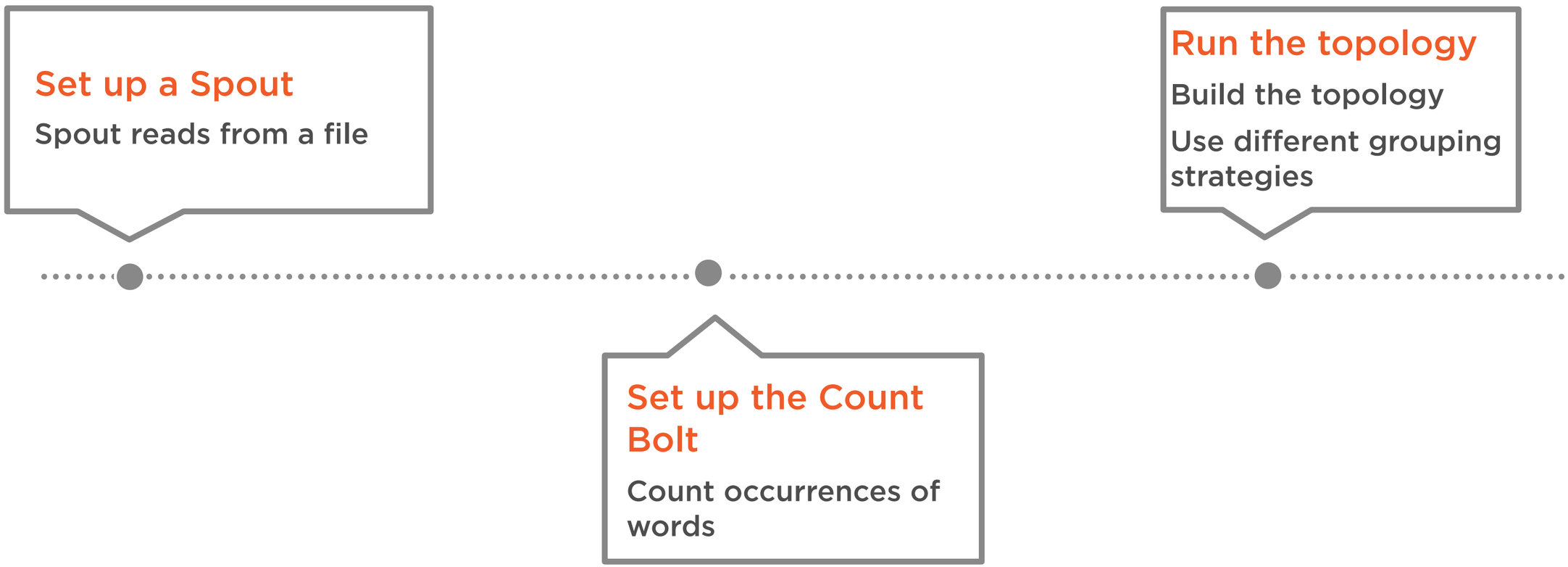
Count Bolt

Maintain a map in-memory with word counts

Increment counts in the map

Write map to file at shutdown

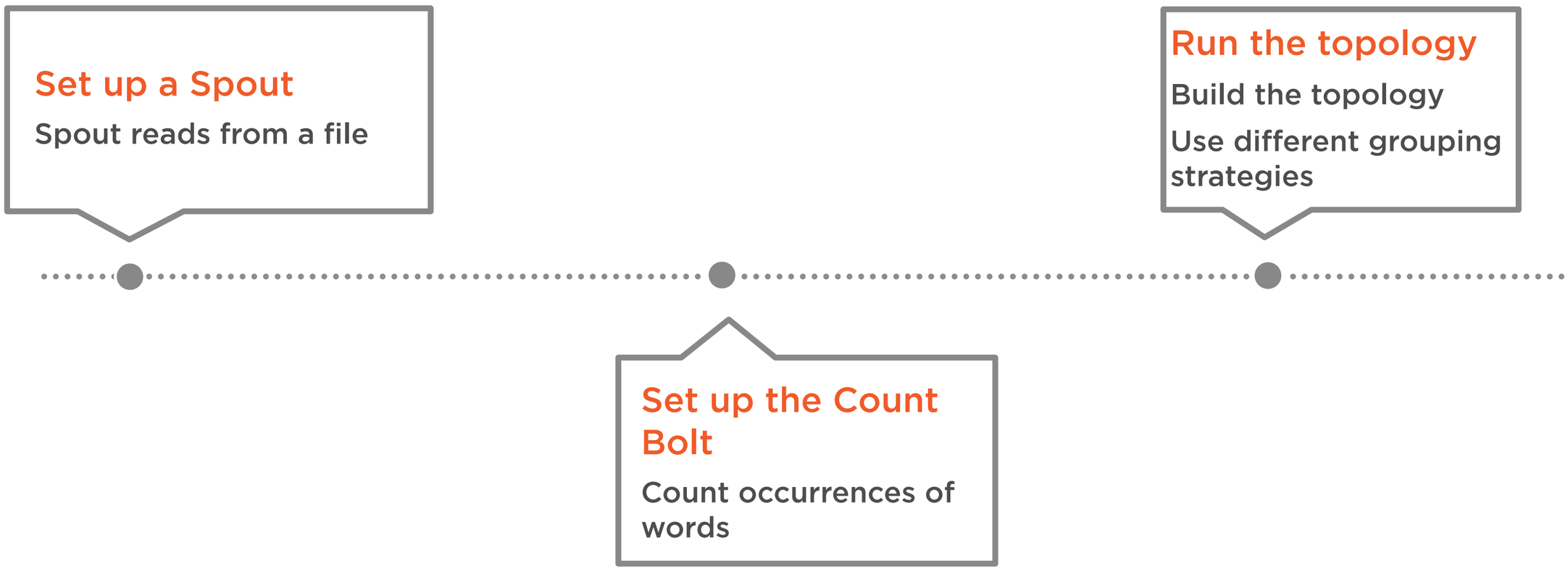
Word Count Topology



Demo

Setting up the Topology components

Word Count Topology



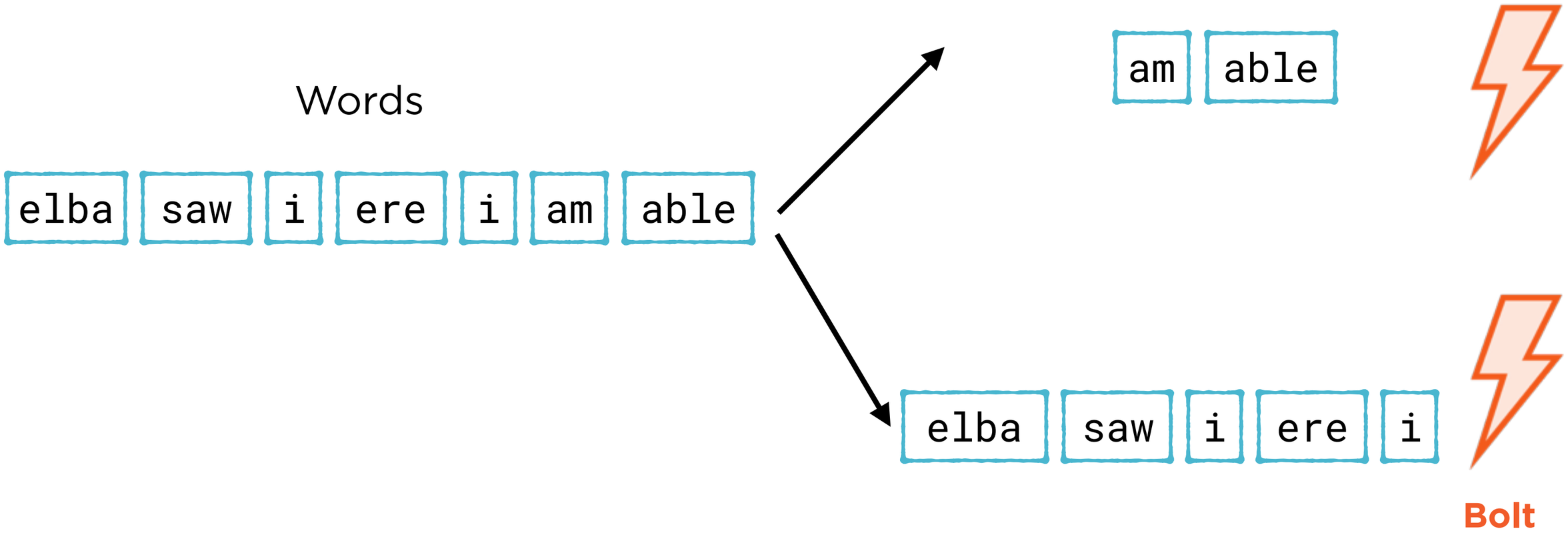
Demo

Build the topology

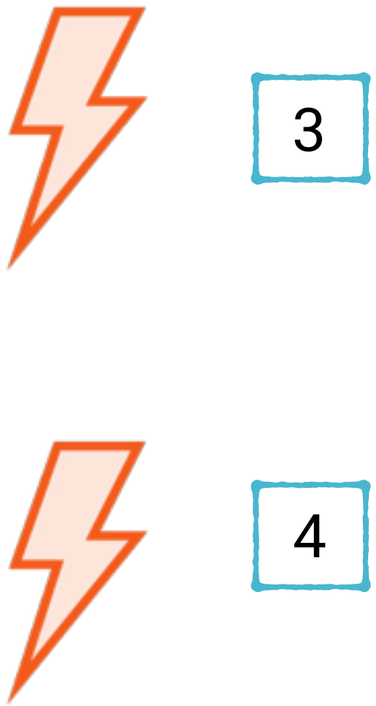
Experiment with different grouping strategies

Implementing a Custom Grouping Strategy

Custom Stream Grouping



Custom Stream Grouping



Identify all possible target task ids

Choose task id(s) for a tuple

```
public void prepare(WorkerTopologyContext context,
GlobalStreamId stream, List<Integer> targetTasks){}
```

Getting Target Tasks

Target task ids provided during initialization

```
public List<Integer> chooseTasks(int taskId, List<Object> values){}
```

Choosing Target Tasks

Use the tuple values to choose tasks

Demo

Implement custom grouping for the word count topology

Summary

- Understand how Storm cluster works
- Set up a remote cluster and run a topology
- Control data flow using stream grouping
- Implement a custom grouping strategy