Università degli Studi di Roma “Tor Vergata”
Dipartimento di Ingegneria Civile e Ingegneria Informatica

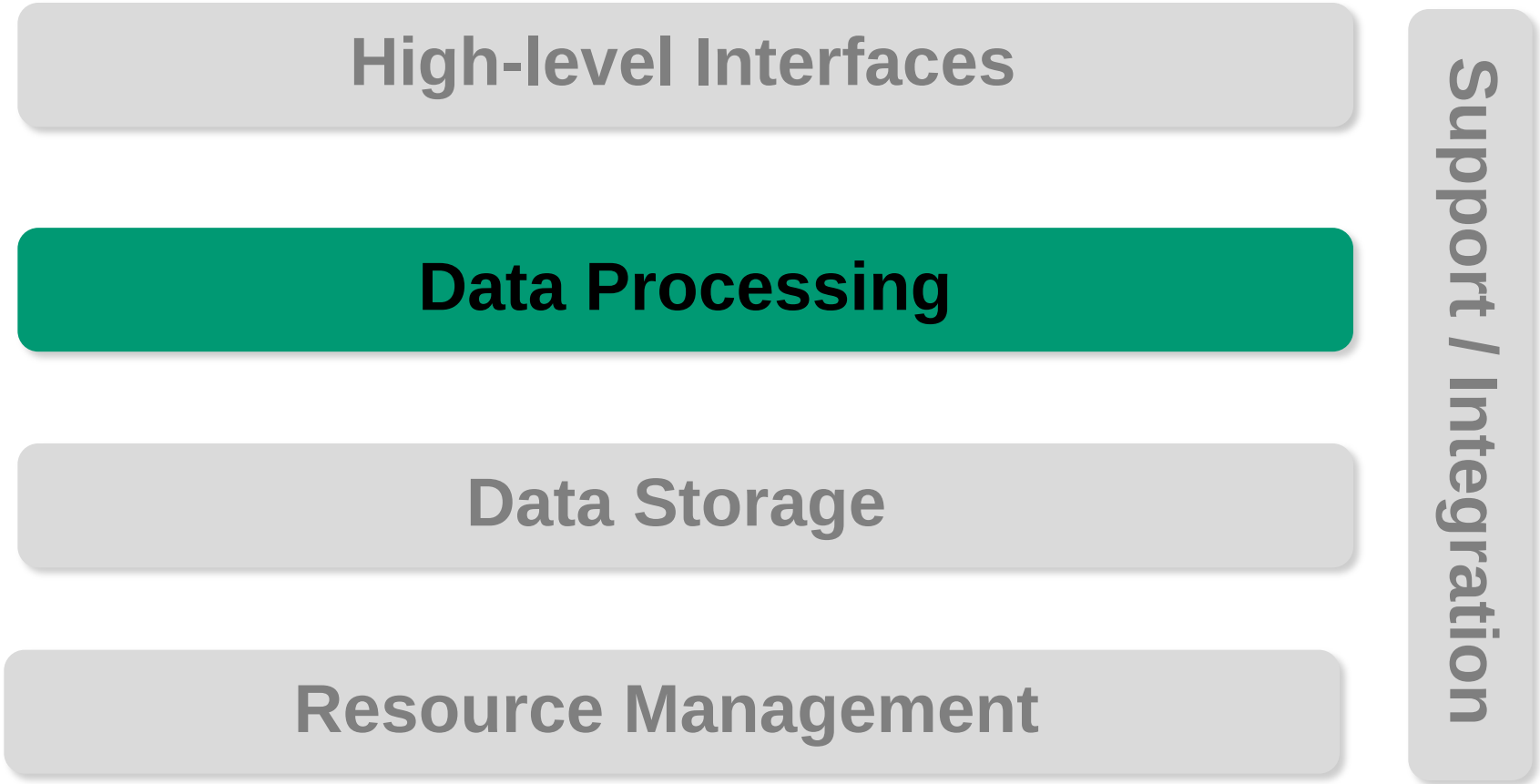
Apache Storm: Hands-on Session

A.A. 2016/17

Matteo Nardelli

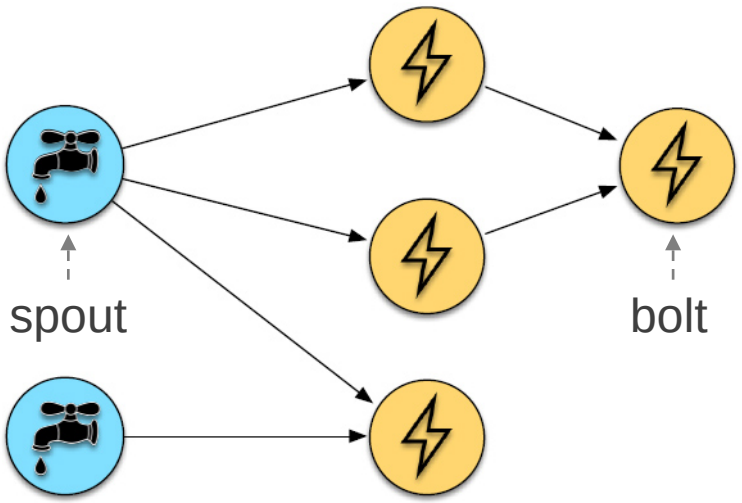
Laurea Magistrale in
Ingegneria Informatica - II anno

The reference Big Data stack



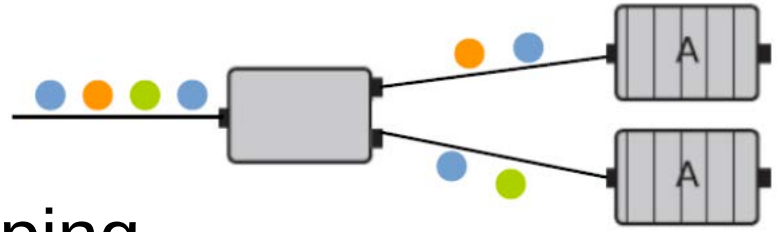
Apache Storm

- Apache Storm
  - Open-source, real-time, scalable streaming system
  - Provides an abstraction layer to execute DSP applications
  - Initially developed by Twitter
- Topology
  - DAG of spouts (sources of streams) and bolts (operators and data sinks
  - stream: sequence of key-value pairs

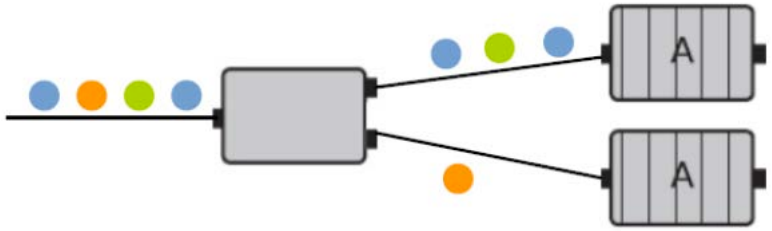


Stream grouping in Storm

- Data parallelism in Storm: how are streams partitioned among multiple tasks (threads of execution)?
- Shuffle grouping
  - Randomly partitions the tuples

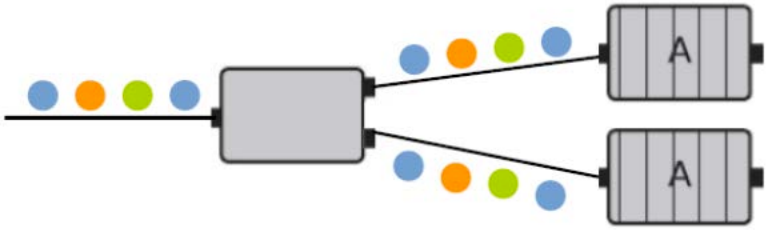


- Field grouping
  - Hashes on a subset of the tuple attributes



Stream grouping in Storm

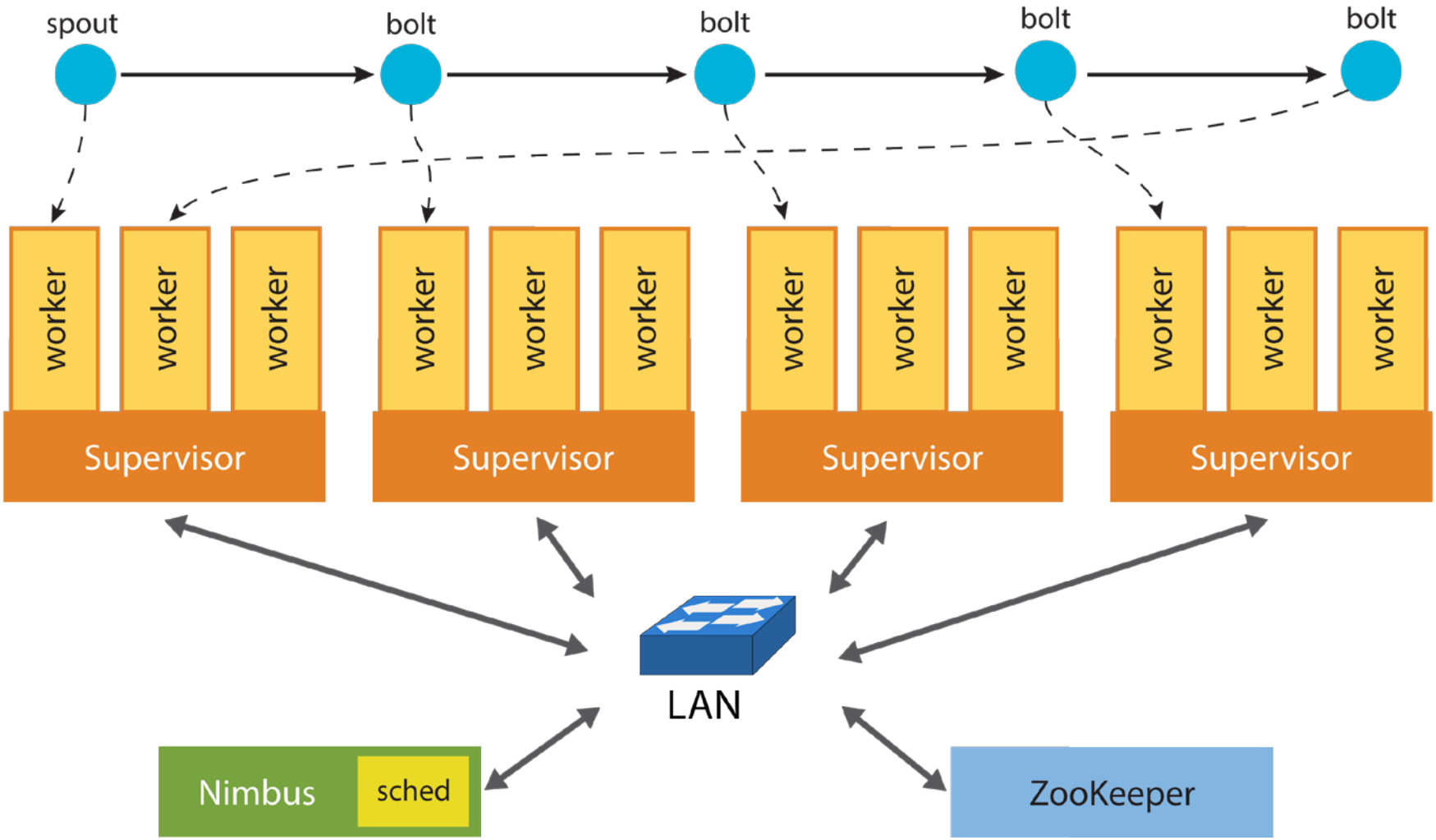
- All grouping (i.e., broadcast)
  - Replicates the entire stream to all the consumer tasks



- Global grouping
  - Sends the entire stream to a single bolt
- Direct grouping
  - Sends tuples to the consumer bolts in the same executor

Storm architecture

- Master-worker architecture



Running a Topology in Storm

Storm allows two running mode: local, cluster

- Local mode: the topology is execute on a single node
  - the local mode is usually used for testing purpose
  - we can check whether our application runs as expected

- Cluster mode: the topology is distributed by Storm on multiple workers
  - The cluster mode should be used to run our application on the real dataset
  - Better exploits parallelism
  - The application code is transparently distributed
  - The topology is managed and monitored at run-time

Running a Topology in Storm

To run a topology in local mode, we just need to create an in-process cluster

- it is a simplification of a cluster
- lightweight Storm functions wrap our code
- It can be instantiated using the LocalCluster class.
  For example:

```
...
LocalCluster cluster = new LocalCluster();
cluster.submitTopology("myTopology", conf, topology);
Utils.sleep(10000); // wait [param] ms
cluster.killTopology("myTopology");
cluster.shutdown();
...
```

Running a Topology in Storm

To run a topology in cluster mode, we need to perform the following steps:

1. Configure the application for the submission, using the StormSubmitter class. For example:

```
...
Config conf = new Config();
conf.setNumWorkers(20);
conf.setMaxSpoutPending(5000);
StormSubmitter.submitTopology("mytopology", conf, topology);
...
```

Running a Topology in Storm

2. Create a jar containing your code and all the dependencies of your code
   - do not include the Storm library
   - this can be easily done using Maven: use the Maven Assembly Plugin and configure your `pom.xml`:

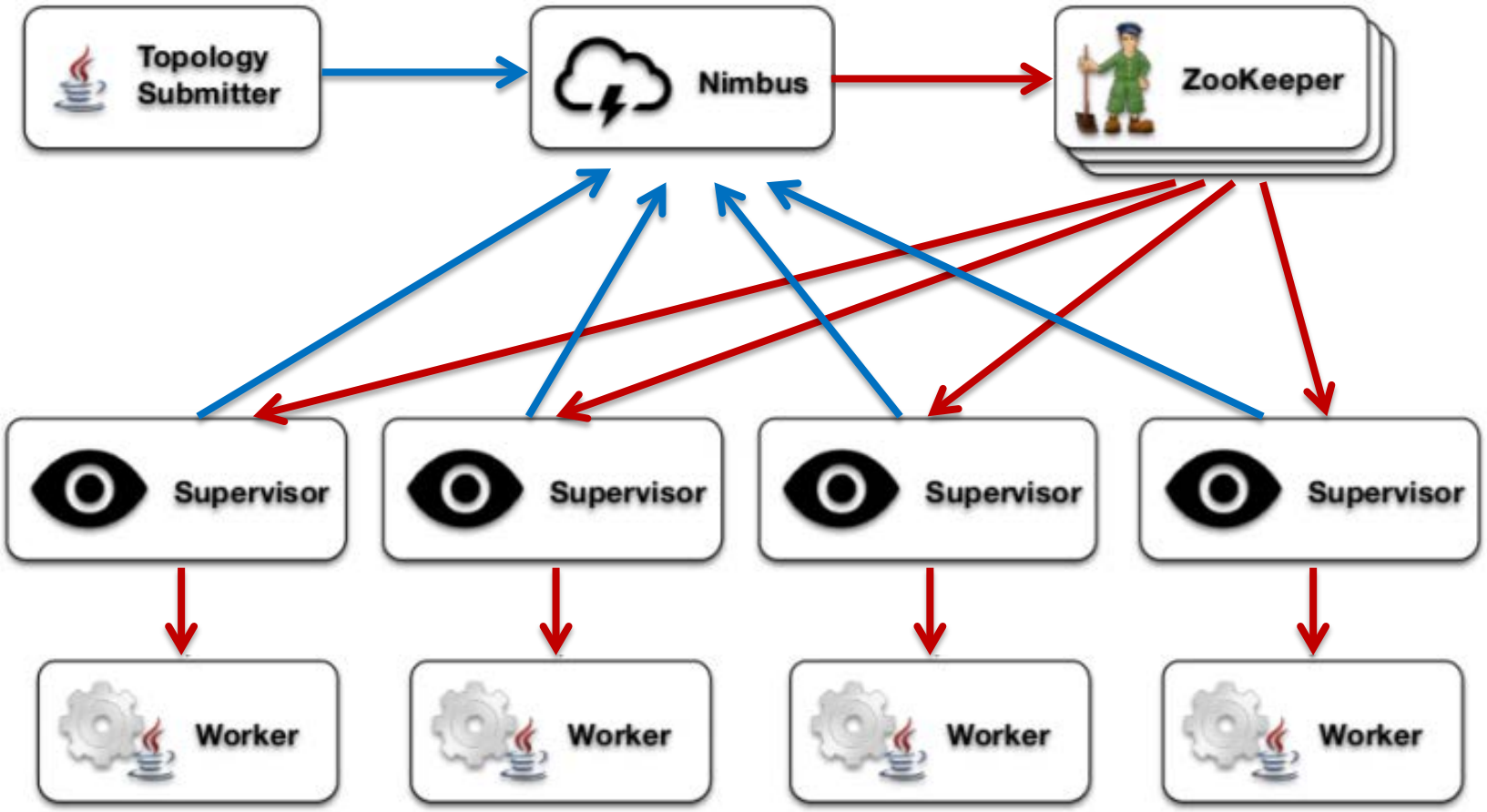
```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
     <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
     </descriptorRefs>
     <archive>
        <manifest>
          <mainClass>com.path.to.main.Class</mainClass>
        </manifest>
     </archive>
  </configuration>
</plugin>
```

Running a Topology in Storm

3. Submit the topology to the cluster using the storm client, as follows

```
$ $STORM_HOME/bin/storm jar
    path/to/allmycode.jar
    full.classname.Topology arg1 arg2 arg3
```

Running a Topology in Storm



A container-based Storm cluster

Running a Topology in Storm

We are going to create a (local) Storm cluster using Docker

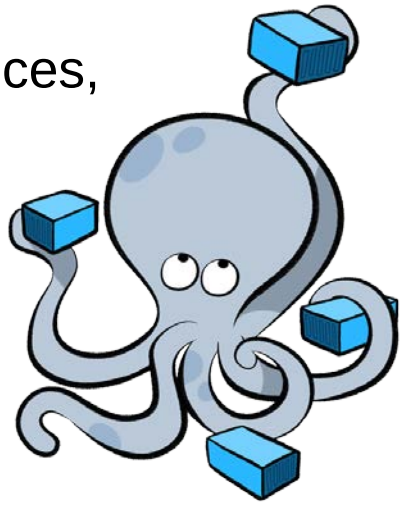
We need to run several containers, each of which will manage a service of our system:

- Zookeeper
- Nimbus
- Worker1, Worker2, Worker3
- Storm Client (storm-cli): we use storm-cli to run topologies or scripts that feed our DSP application

Auxiliary services: they that will be useful to interact with our Storm topologies

- Redis: we know it :-)
- RabbitMQ: a message queue service

Docker Compose



To easily coordinate the execution of these multiple services, we use **Docker Compose**

- Read more at https://docs.docker.com/compose/

Docker Compose:

- is not bundled within the installation of Docker
- it can be installed following the official Docker documentation
  - https://docs.docker.com/compose/install/
- Allows to easily express the container to be instantiated at once, and the relations among them
- By itself, docker compose runs the composition on a single machine; however, in combination with Docker Swarm, containers can be deployed on multiple nodes

Docker Compose

- We specify how to compose containers in a easy-to-read file, by default named `docker-compose.yml`
- To start the docker composition (in background with -d):

```
$ docker-compose up -d
```

- To stop the docker composition:

```
$ docker-compose down
```

- By default, docker-compose looks for the `docker-compose.yml` file in the current working directory; we can change the file with the configuration using the `-f` flag

Docker Compose

- There are different versions of the docker compose file format
- We will use the version 3, *supported from Docker Compose 1.13*

```
version: '3'

services:
    storm-nimbus:
        image: storm
        container_name: nimbus
        command: storm nimbus
        depends_on:
            - zookeeper
        links:
            - zookeeper
        ports:
            - "6627:6627"
```

```
    zookeeper:
        image: zookeeper
        container_name: zookeeper
        ports:
            - "2181:2181"

    worker1:
        image: storm
        command: storm supervisor
        depends_on:
            - storm-nimbus
            - zookeeper
        links:
            - storm-nimbus
            - zookeeper
```

On the docker compose file format: https://docs.docker.com/compose/compose-file/