

MapReduce and Hadoop: Advanced topics

Multiple inputs

Multiple inputs

- In some applications data are read from two or more datasets
 - Datasets could have different formats
- Hadoop allows reading data from **multiple inputs** (multiple datasets) with different **formats**
 - One different mapper for each input dataset must be specified
 - However, the key-value **pairs emitted** by the mappers must be **consistent in terms of data types**

Multiple inputs

- Example of a use case
 - Input data collected from different sensors
 - All sensors measure the same “measure”
 - But sensors developed by different vendors use a different data format to store the gathered data/measurements

Multiple inputs

- In the driver
 - Use the `addInputPath` method of the `MultipleInputs` class multiple times to
 - Add one input path at a time
 - Specify the input format class for each input path
 - Specify the Mapper class associated with each input path

Multiple inputs

- E.g.,

```
MultipleInputs.addInputPath(job, new Path(args[1]),  
    TextInputFormat.class, Mapper1.class);
```



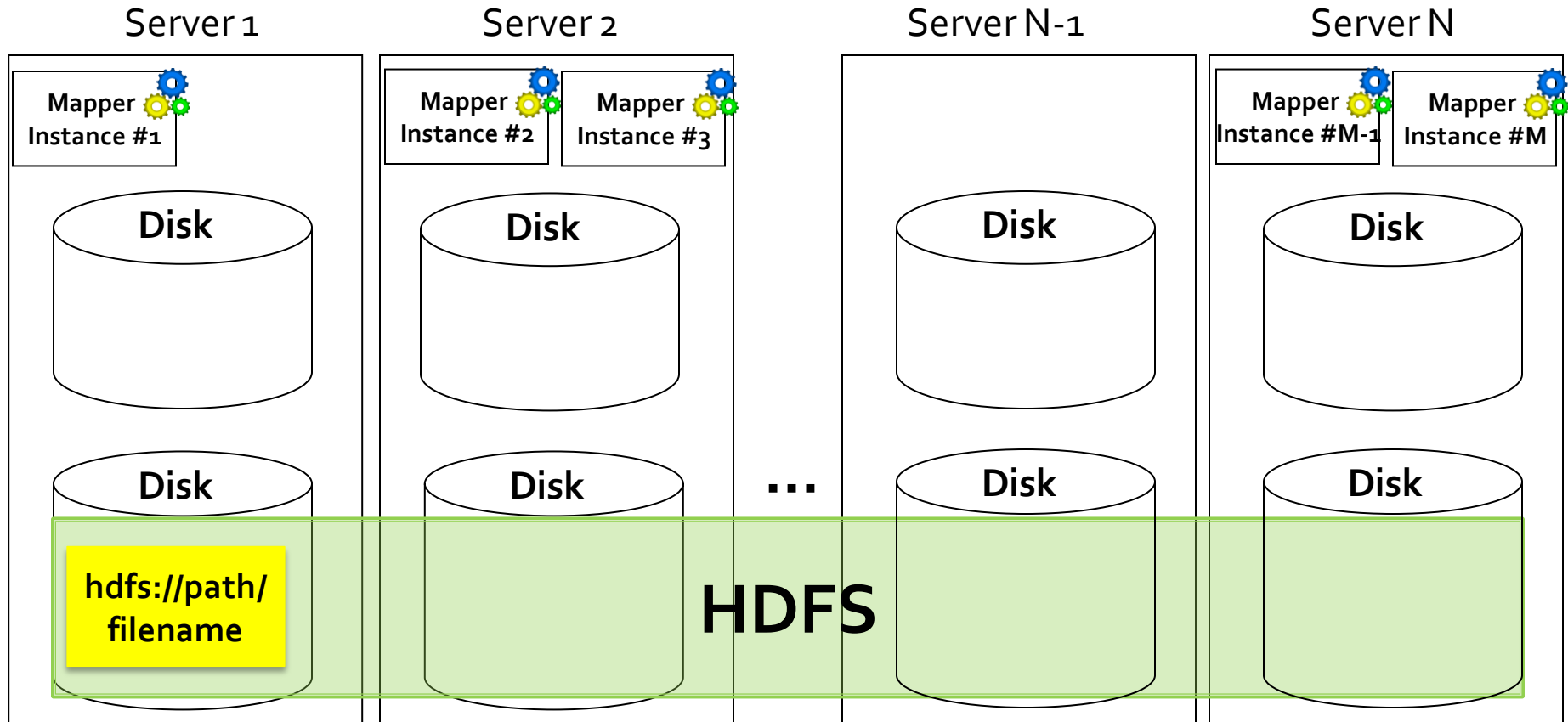
```
MultipleInputs.addInputPath(job, new Path(args[2]),  
    TextInputFormat.class, Mapper2.class);
```
- Specify two input paths (args[1] and args[2])
- The data of both paths are read by using the TextInputFormat class
- Mapper1 is the class used to manage the input key-value pairs associated with the first path
- Mapper2 is the class used to manage the input key-value pairs associated with the second path

Distributed cache

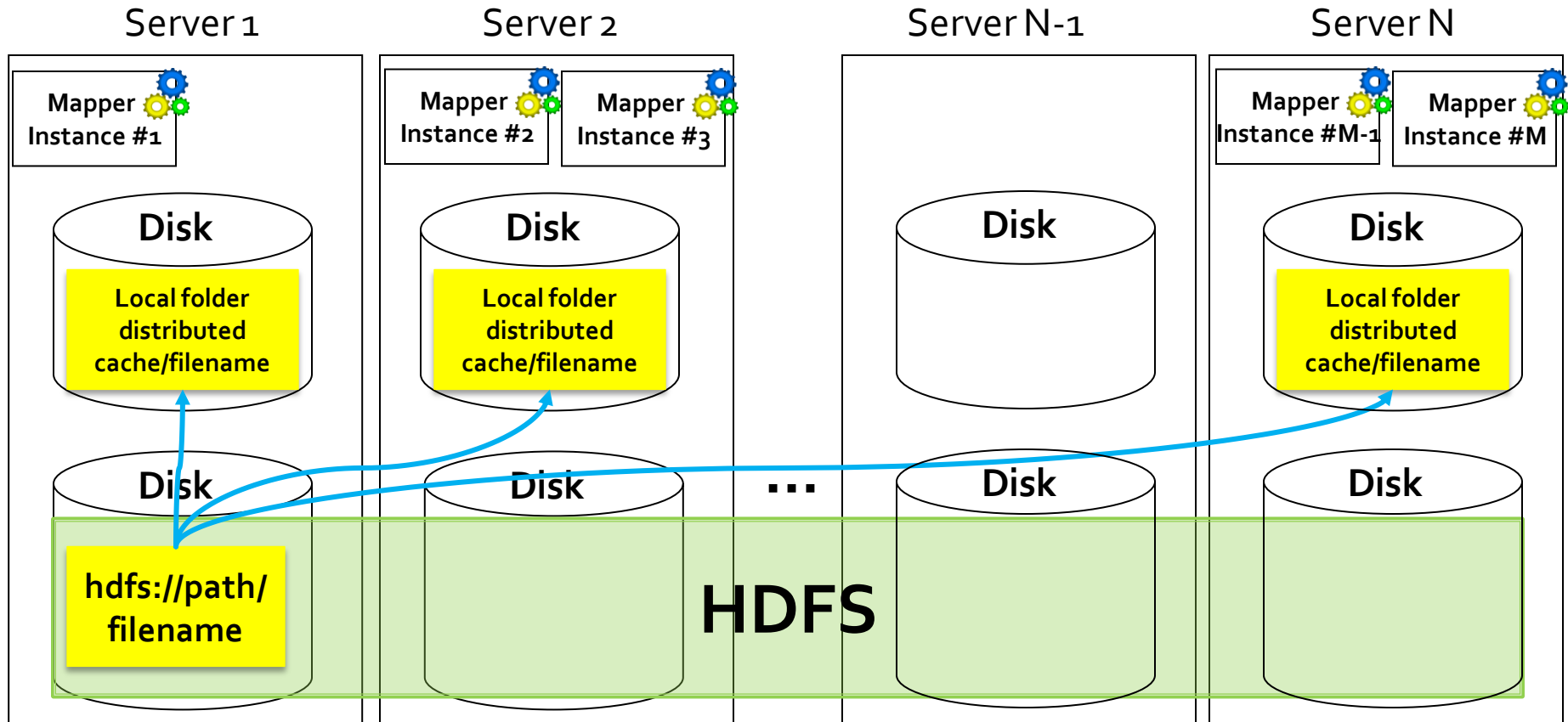
Distributed cache

- Some applications need to share and cache (small) read-only files to perform efficiently their task
- These files should be accessible by all nodes of the cluster in an efficient way
 - Hence a copy of the shared/cached (HDFS) files should be available locally in all nodes used to run the application
- **DistributedCache** is a facility provided by the Hadoop-based MapReduce framework to cache files
 - E.g., text, archives, jars needed by applications

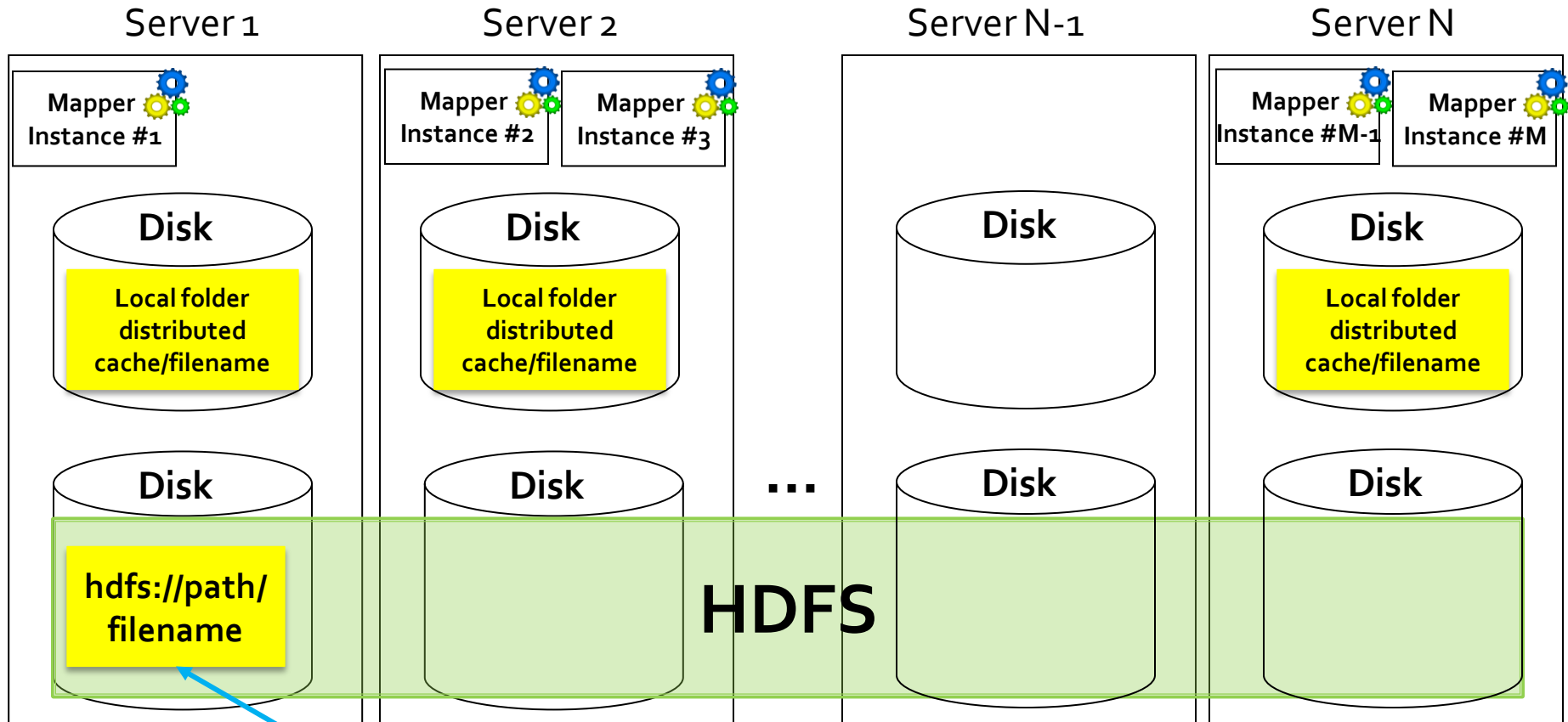
Distributed cache



Distributed cache

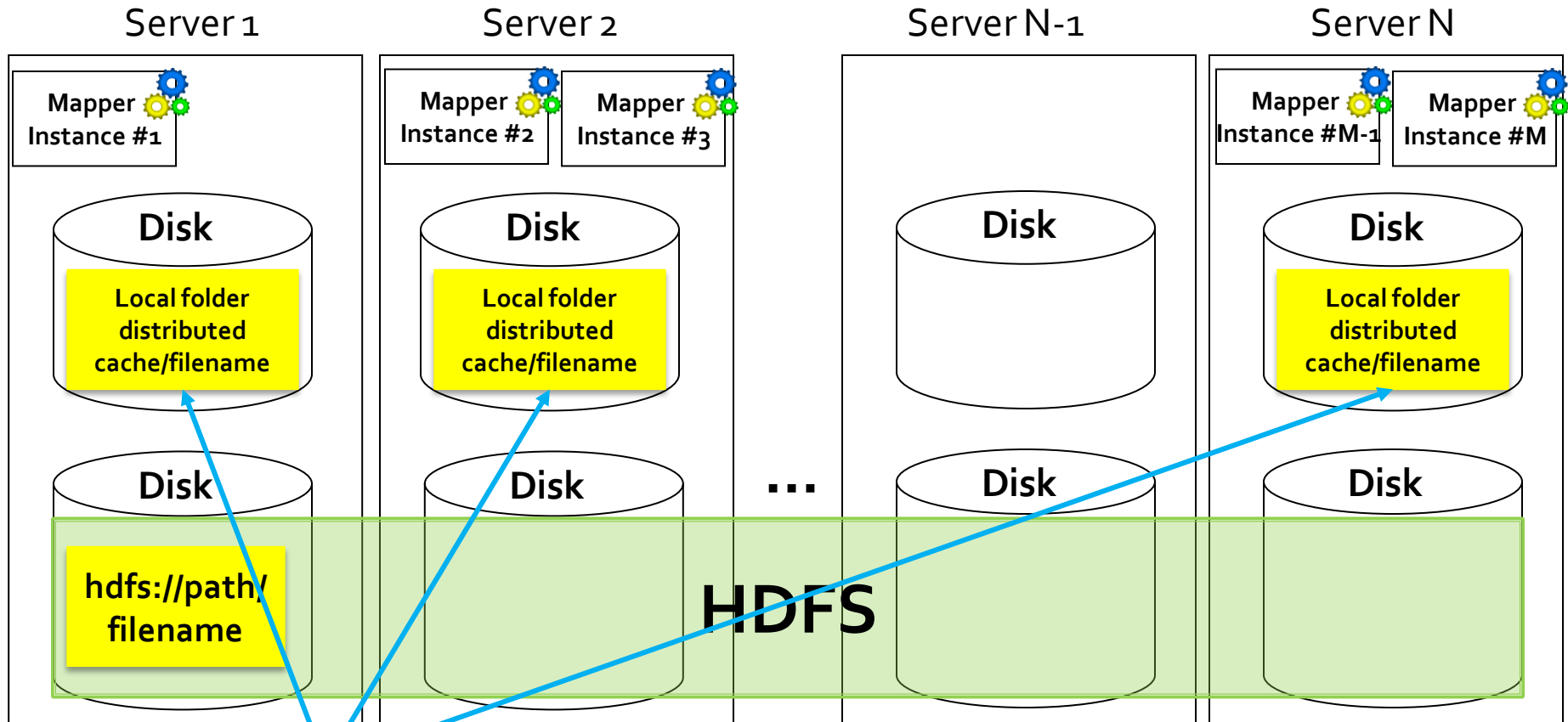


Distributed cache



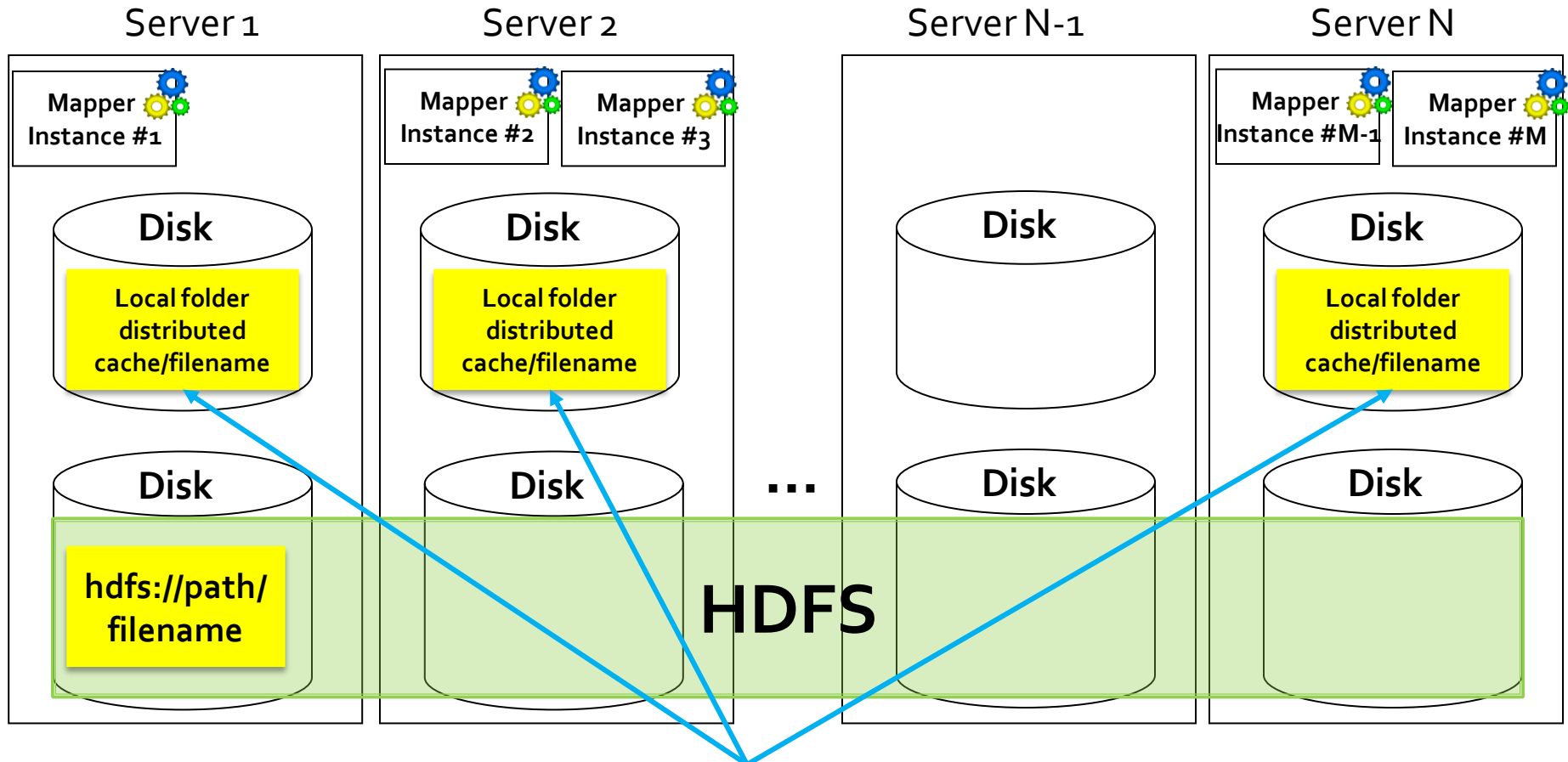
HDFS file(s) to be shared by means of the distributed cache

Distributed cache



Local copies of the file(s) shared by means of the distributed cache

Distributed cache



A local copy of the file(s) shared by means of the distributed cache is created only in the servers running the application that uses the shared file(s)

Distributed cache

- In the Driver of the application, the set of shared/cached files are specified
 - By using the job.`addCacheFile`(path) method
- During the initialization of the job, Hadoop creates a “local copy” of the shared/cached files in all nodes that are used to execute some tasks (mappers or reducers) of the job (i.e., of the running application)
- The shared/cache file is read by the mapper (or the reducer), usually in its setup method
 - Since the shared/cached file is available **locally** in the used nodes/servers, its content can be read efficiently

Distributed cache

- The **efficiency** of the distributed cache depends on the **number of** multiple **mappers** (or reducers) running on the **same node/server**
 - For each node a local copy of the file is copied during the initialization of the job
 - The **local** copy of the **file** is **used by all mappers** (reducers) running on the **same node/server**
- **Without the distributed cache**, each mapper (reducer) should read, in the setup method, the shared HDFS file
 - Hence, **more time** is needed because reading **data from HDFS** is more inefficient than reading data from the local file system of the node running the mappers (reducers)

Distributed cache

Distributed cache: driver

```
public int run(String[] args) throws Exception {
```

```
.....
```

```
// Add the shared/cached HDFS file in the  
// distributed cache
```

```
job.addCacheFile(new Path("hdfs  
path/filename").toUri());
```

```
.....
```

```
}
```

Distributed cache: mapper/reducer

```
protected void setup(Context context) throws IOException,  
    InterruptedException {
```

```
    .....
```

```
    String line;
```

```
    // Retrieve the (original) paths of the distributed files  
    URI[] urisCachedFiles = context.getCacheFiles\(\);
```

Distributed cache: mapper/reducer

```
// Read the content of the cached file and process it.
// In this example the content of the first shared file is opened.
BufferedReader file = new BufferedReader(new FileReader(
    new File(new Path(urisCachedFiles[0].getPath()).getName())));

// Iterate over the lines of the file
while ((line = file.readLine()) != null) {
    // process the current line
    .....
}

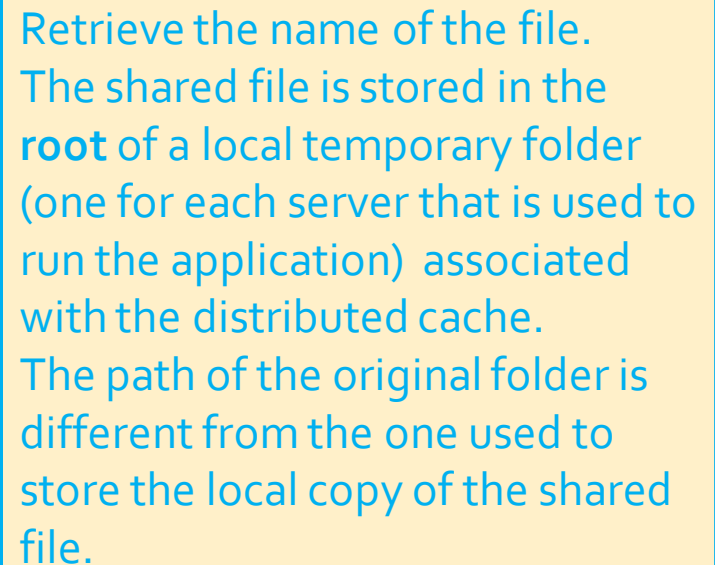
file.close();
}
```

Distributed cache: mapper/reducer

```
// Read the content of the cached file and process it.
// In this example the content of the first shared file is opened.
BufferedReader file = new BufferedReader(new FileReader(
    new File(new Path(urisCachedFiles[0].getPath()).getName())));

// Iterate over the lines of the file
while ((line = file.readLine()) != null) {
    // process the current line
    ....
}

file.close();
}
```



Retrieve the name of the file.
The shared file is stored in the **root** of a local temporary folder (one for each server that is used to run the application) associated with the distributed cache.
The path of the original folder is different from the one used to store the local copy of the shared file.