Building a Robust Spark Streaming Application

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Understand checkpointing in streaming applications

Understand how driver, executor and receiver fault tolerance works

Build a real world application to work with streaming Twitter data

Build robustness into this application using checkpointing

A Robust Application

Fault tolerance

The ability of an application to recover from failures

A Robust Application

Spark runs on a distributed system

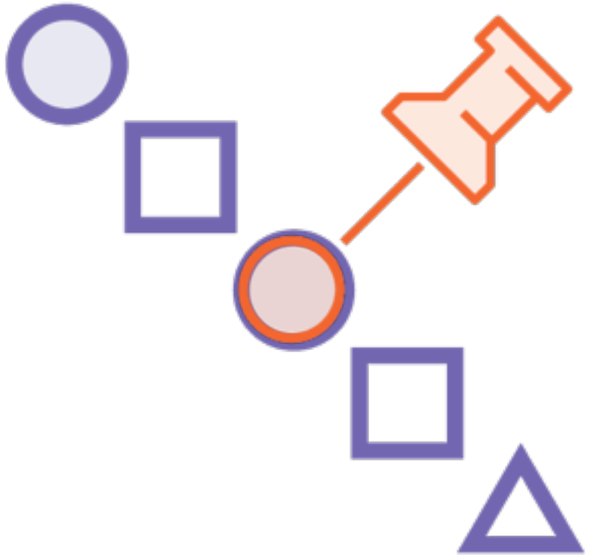
- Data replication
- Process restart on new nodes in case of crashes

A Robust Application

Streaming applications require additional features to protect against data loss

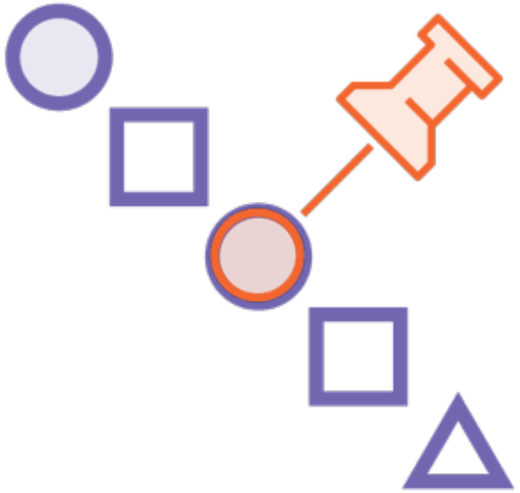
Fault Tolerance with Checkpointing

Checkpointing



Fault tolerance

Periodically save data to a reliable storage system

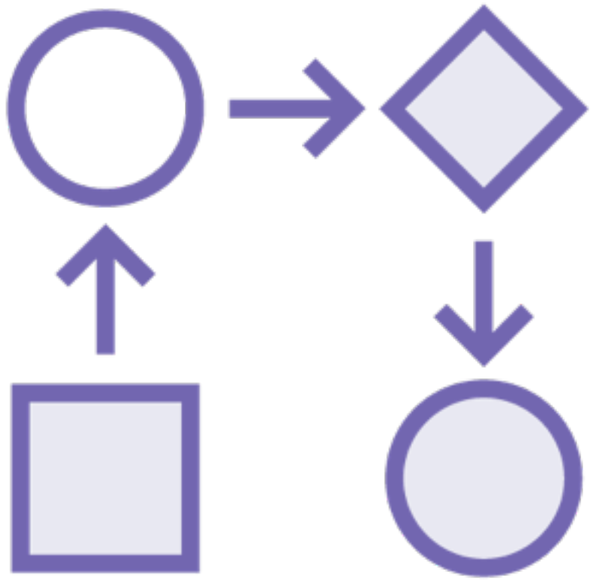


Checkpointing

Limiting state re-computation in case of failure

Fault tolerance for driver jobs

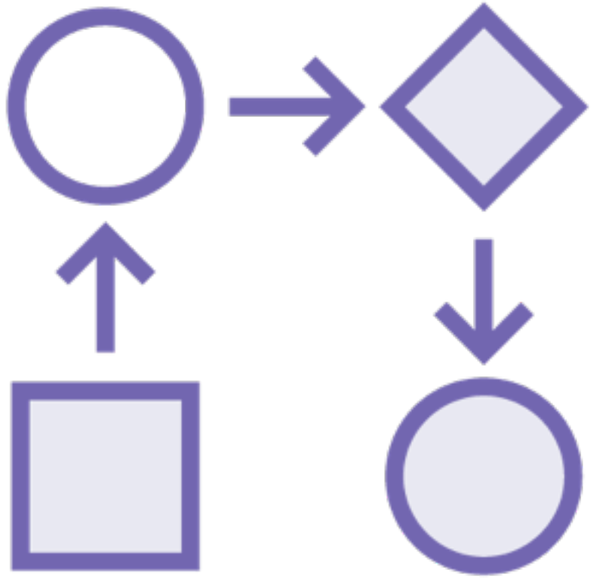
The Directed Acyclic Graph



A Spark program is split into discrete tasks

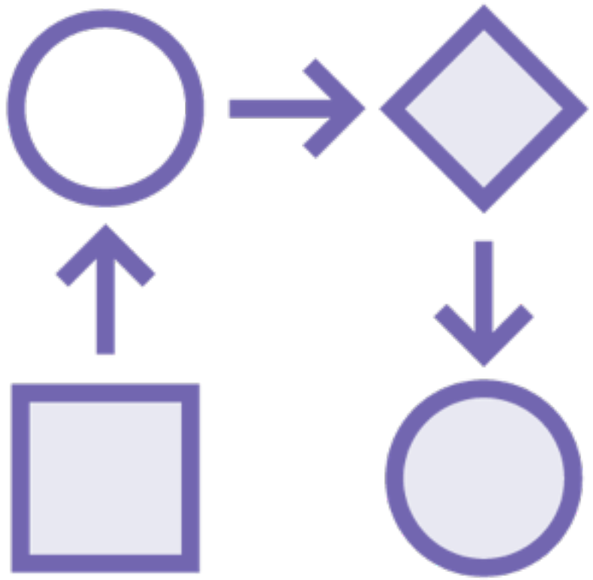
A task usually has an RDD associated with it

The task and its associated RDD depend on other tasks



The Directed Acyclic Graph

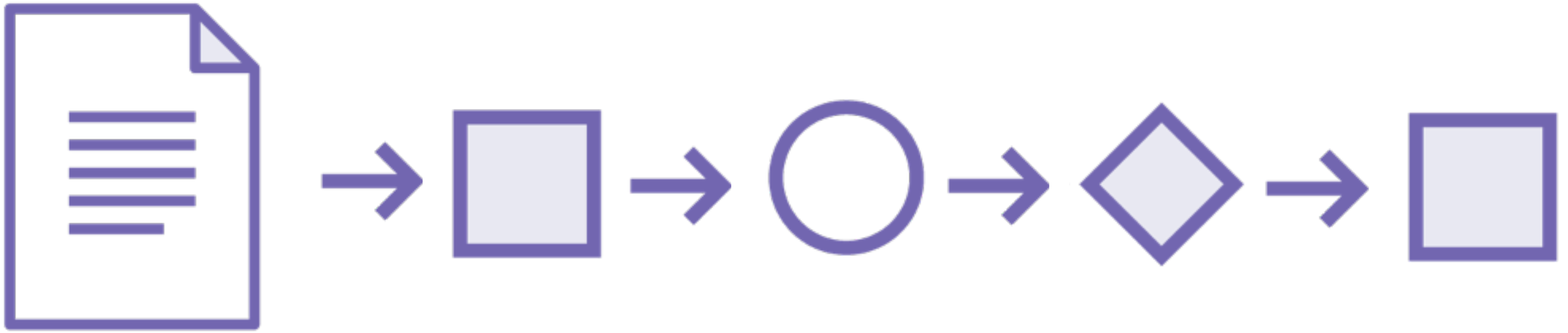
The tasks and RDDs form a **Directed Acyclic Graph**



Lineage

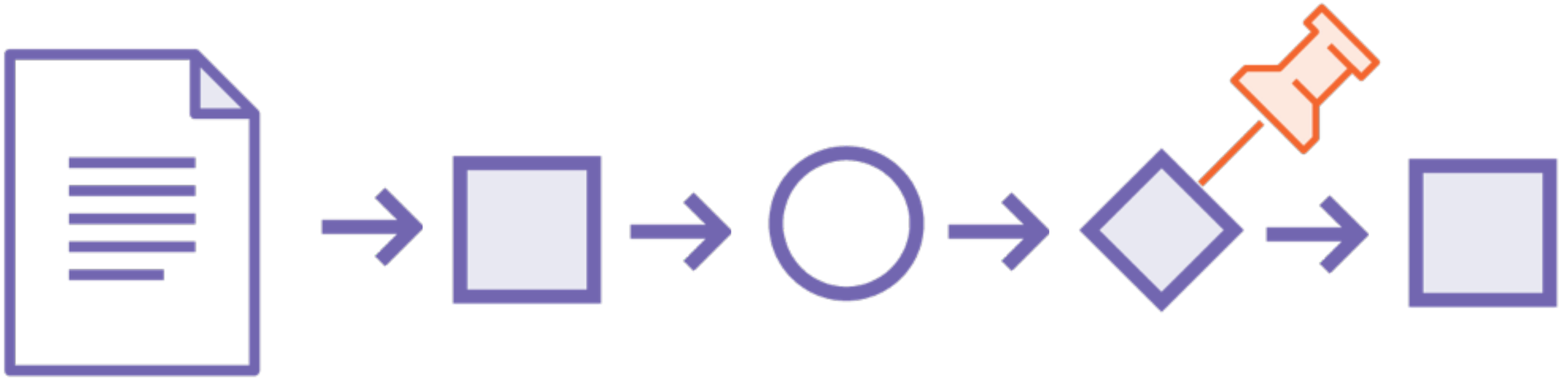
An RDD can always be **reconstructed** using its lineage

Limiting State Re-computations



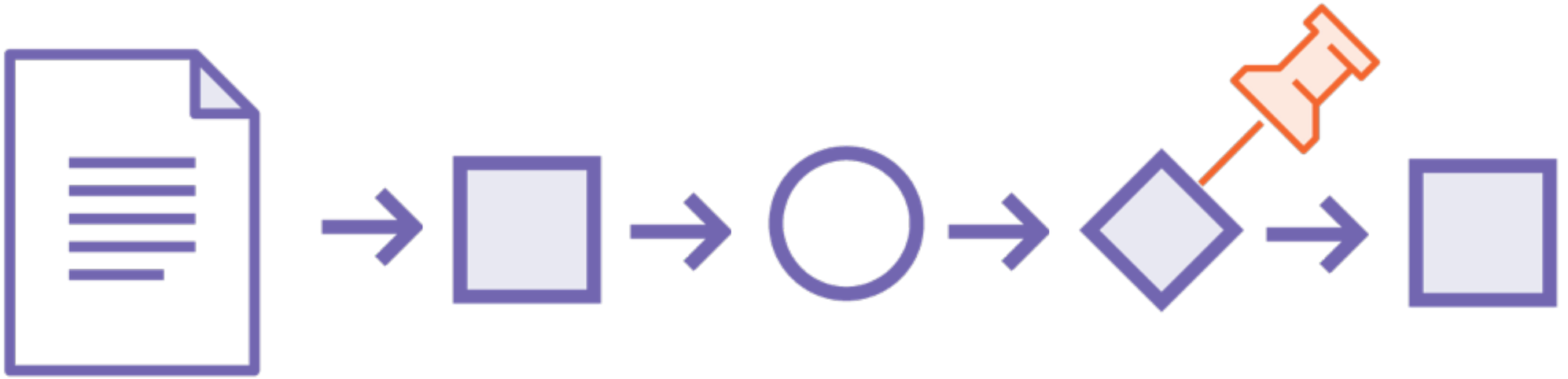
Reconstructing an RDD from scratch is a lot of re-computation

Limiting State Re-computations



Checkpointing reduces how much state has to be recomputed

Limiting State Re-computations



If the driver program crashes it can recover from the checkpoint rather than the beginning of time

```
sc = SparkContext(appName="StreamingWordCount")
ssc = StreamingContext(sc, 30)

ssc.checkpoint("file:///tmp/spark")
```

Checkpoint Streaming Applications

For production systems, store checkpoints in a reliable replicated source like HDFS or Amazon's S3

Demo

Connect to the Twitter Streaming API to access tweets using the Python tweepy package

Stream tweets to a port on the local machine

Build a Spark application to listen to streaming tweets and perform transformations

Fault Tolerance in Spark Streaming Components

Spark Architecture

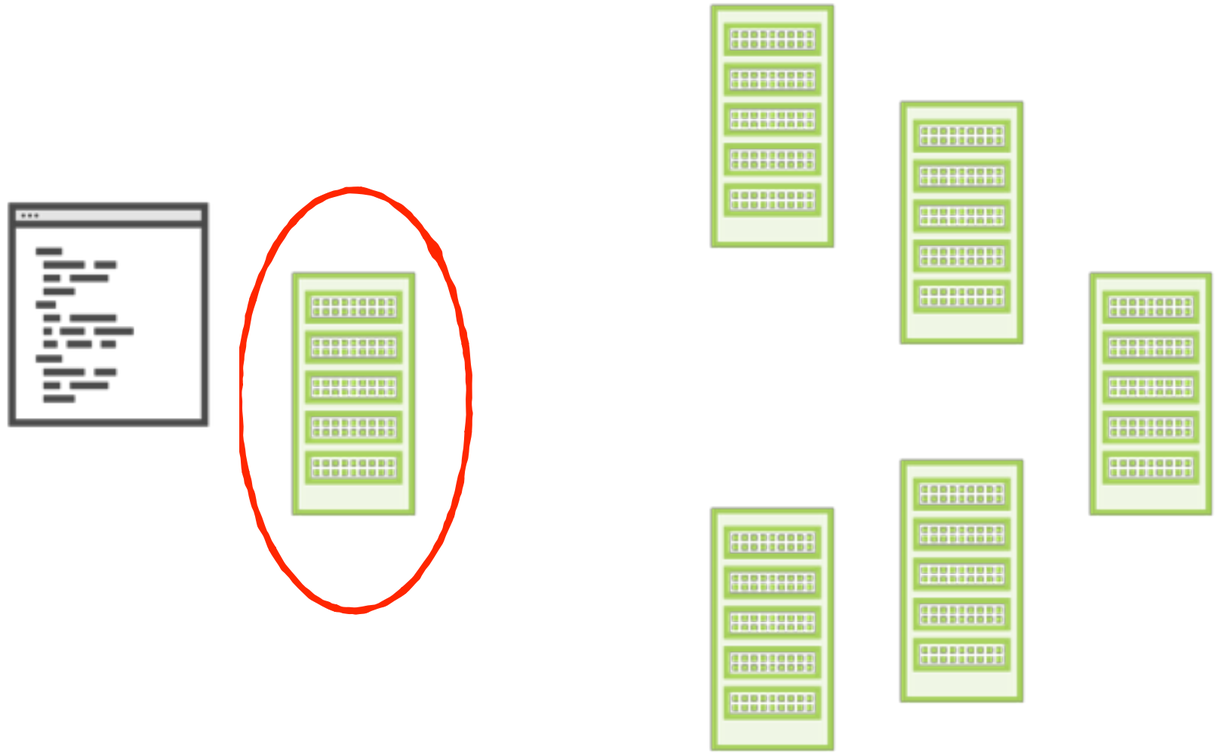
Master/Slave architecture

A central coordinator with many distributed workers

Spark Architecture

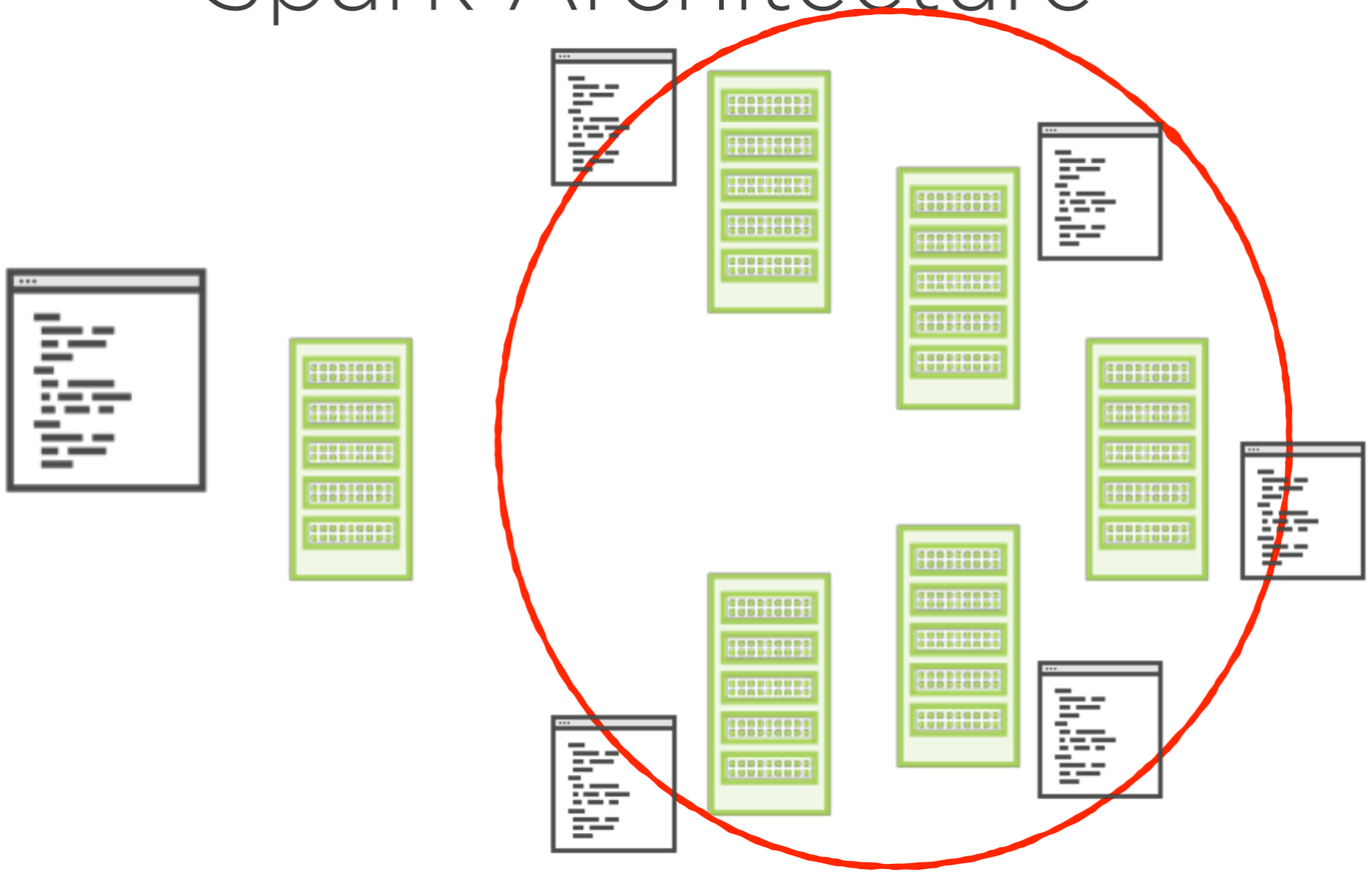
You write code, and submit it to Spark

Spark Architecture



A coordinator receives the program and breaks it into discrete tasks

Spark Architecture



Each task is assigned to worker machines to execute

Spark Architecture

Driver

The coordinator process which executes the user program

Executor

The worker process responsible for running individual jobs

Driver

Converts a user program into physical executables called tasks

Schedules tasks on executors

Runs in its own Java process

Spark Architecture

Driver

Executor

Executors

Worker processes which run the tasks in a Spark job

Provide in-memory storage for RDDs so that tasks run close to the data

Each executor is a separate Java process

Spark Architecture

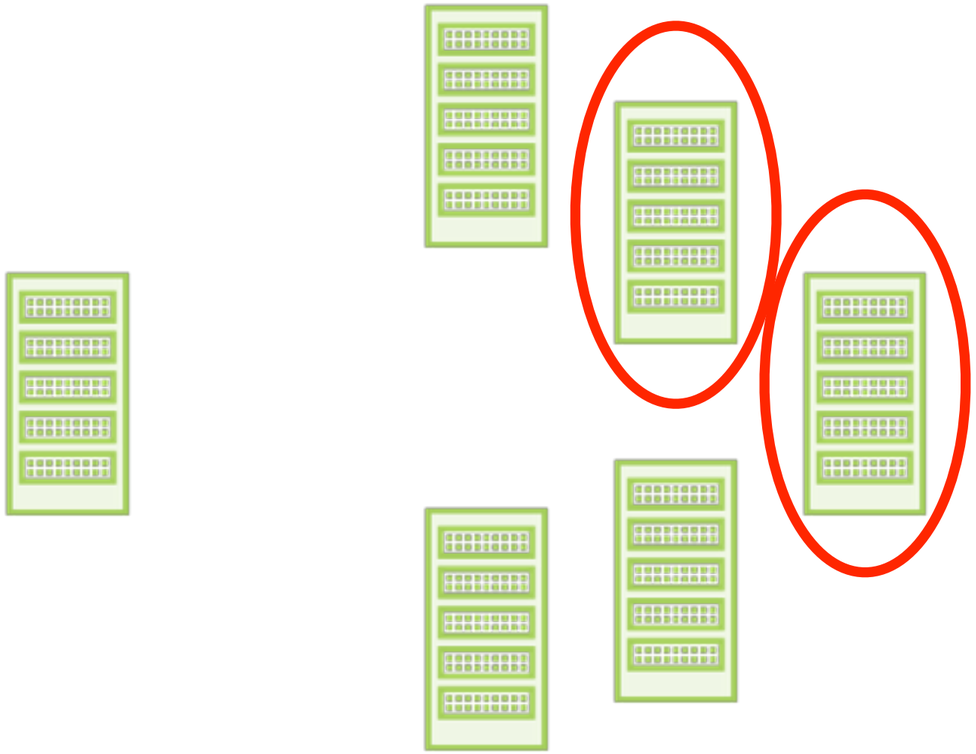
A cluster manager for distributed systems launches the driver and executor programs

Spark Architecture

Cluster managers are pluggable, the built-in one is called the Standalone cluster manager

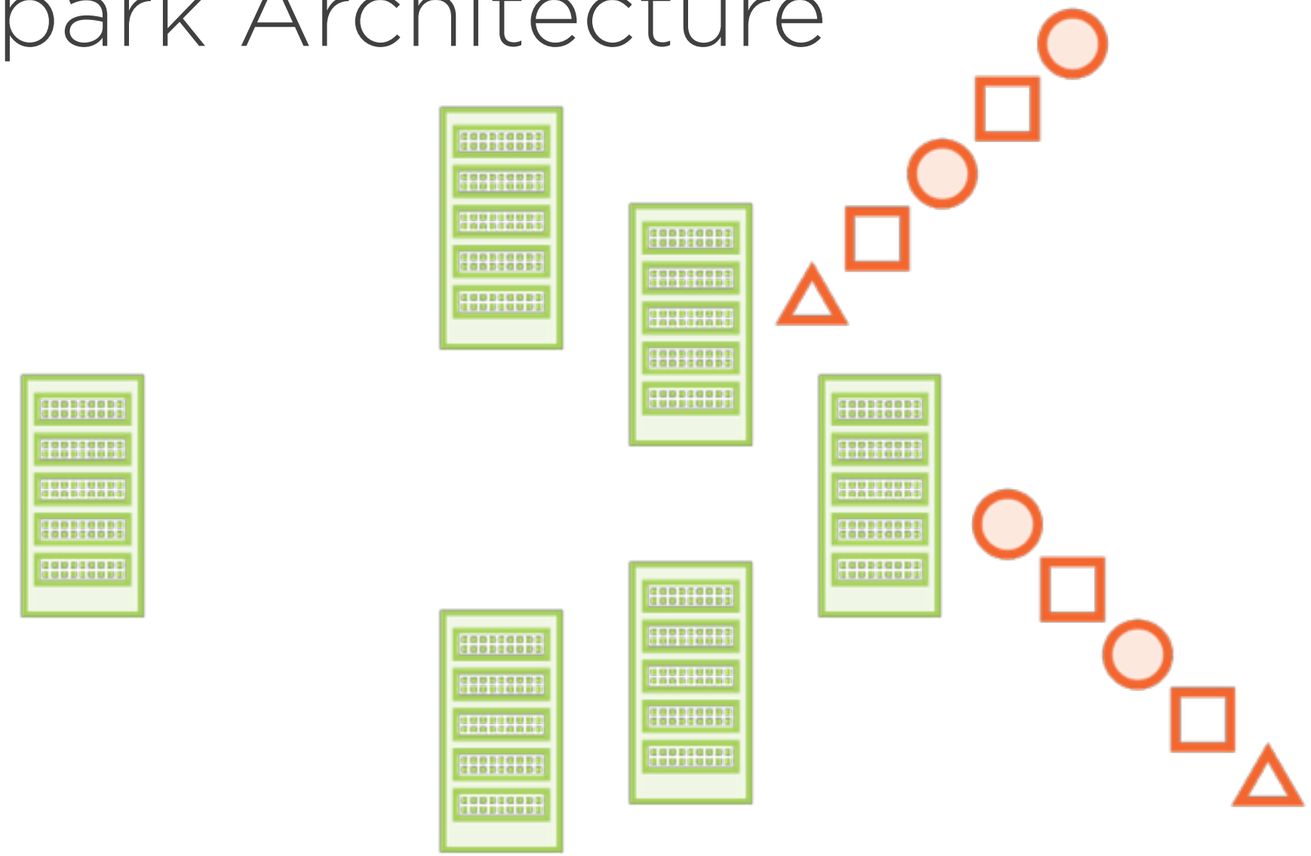
Spark streaming uses an additional component called **receivers**, one for each input source

Spark Architecture



Allocate executors to accept streaming data

Spark Architecture



Tasks within those executors receive data from external sources

Input Data in Streaming Spark

Receivers

Collect input data from different sources

Receivers

Tasks run within executors

Collect data from input sources and save them as RDDs

Replicate the collected data to another executor for fault tolerance

Spark Architecture

The streaming application should be able to recover from failures of all these components

Driver Fault Tolerance

Driver Fault Tolerance

Recover from driver crashes from the last checkpoint state

Driver Fault Tolerance

No intermediate checkpoint state

- Set up a brand new state for the job

Intermediate state present

- Recreate state from the checkpoint

Demo

Use the StreamingContext.getOrCreate() function to re-create context from stored state

Executor and Receiver Fault Tolerance

Executor Fault Tolerance

Worker crashes and recovery are similar to Spark

Executor Fault Tolerance

All data from external sources are replicated on multiple nodes

RDDs from this source data use lineage to reconstruct themselves

Receiver Fault Tolerance

Workers running receivers have different fault tolerance mechanics

Receiver Fault Tolerance

Received data is replicated to other data nodes

Receiver Fault Tolerance

Sources which are reliable can resend data

- HDFS, S3, storage systems

Sources such as Twitter, Kafka may be prone to some data loss if the data is not checkpointed

Processing Guarantees

Spark streaming provides exactly once semantics for processes

Processing Guarantees

External systems might consume Spark output

Transformed data might get pushed to them multiple times

- **this has to be accounted for either in Spark or the external system**

Overview

Understood the importance of saving intermediate state with checkpointing

Know how driver, worker and receiver fault tolerance works

Built a robust real world application to work with streaming Twitter data