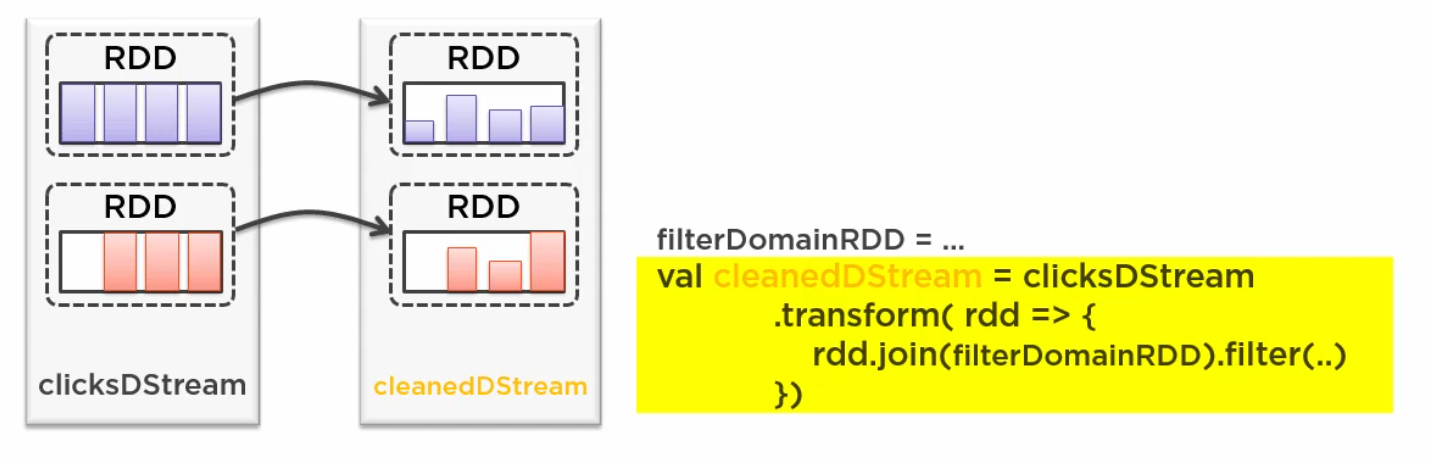
**WINDOW**

* + Create new Dstream computed by applying window parameters to the old stream.
  + For example, you want to POST all the active users from the last five seconds to a web service, but you want to update the results every second.
  + These operations describe two parameters – windowLength and slideInterval.
  + REDUCE LAST 30 SECONDS OF DATA FOR EVERY 10 SECONDS
    - val windowedWordCounts = pairs.reduceByKeyAndWindow((a:Int,b:Int) => (a + b), Seconds(30), Seconds(10))

**TRANSFORM (func)**

* + Allows to drop down to RDD and operate against the rdd api
  + Higher order fuction like map
  + We can transform to an RDD of any other type



**FOR EACH RDD**

* + Similar to transform
  + Function returns a unit(void)

USE CASE

saving data to Cassandra



**SPARK RECIEVER MODEL**

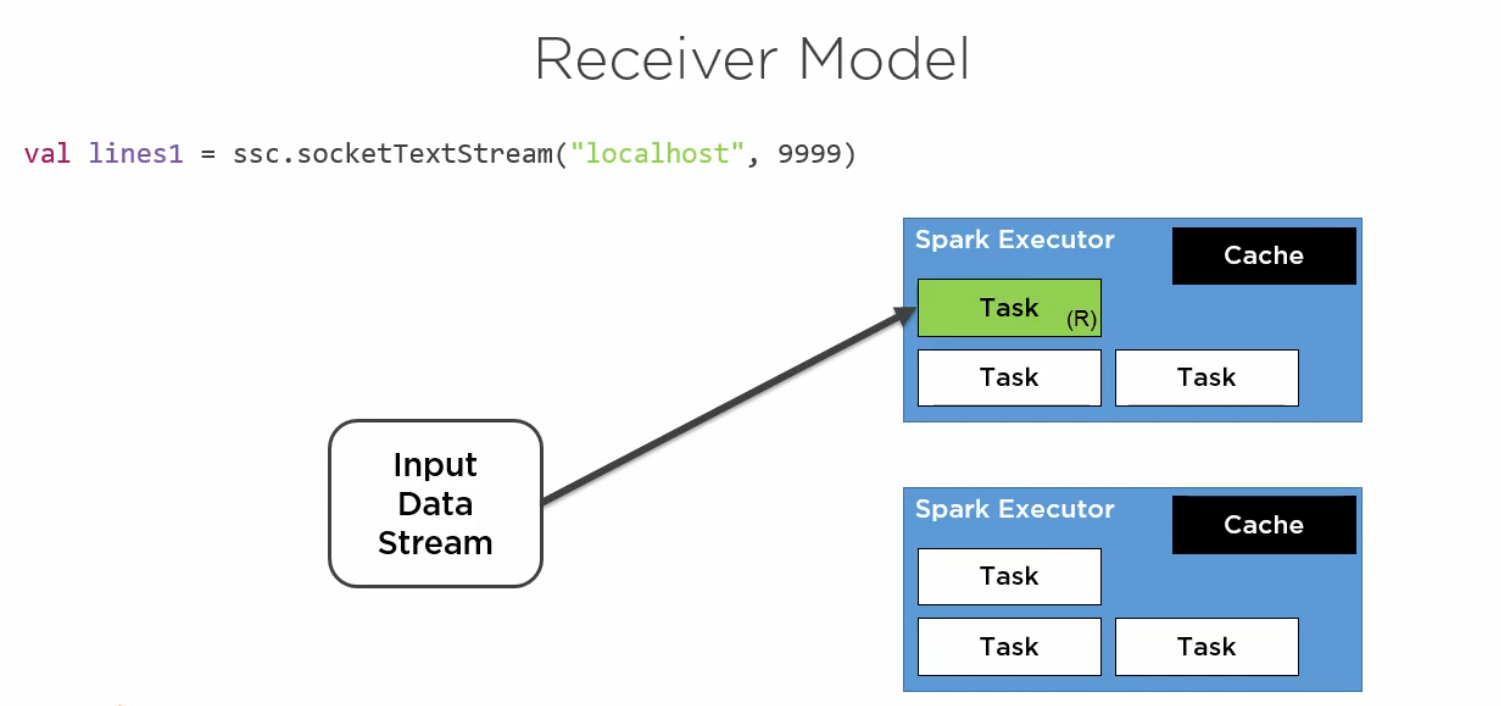
* + Object that consumes a task and is responsible for receiving data from and storing it in spark memory for processing
  + Occupies a task for itself essentially eating up cpu core
  + If the data is more, we would want to free that memory for another task

TYPICAL ALLOCATION OF CORES

1 for receiver

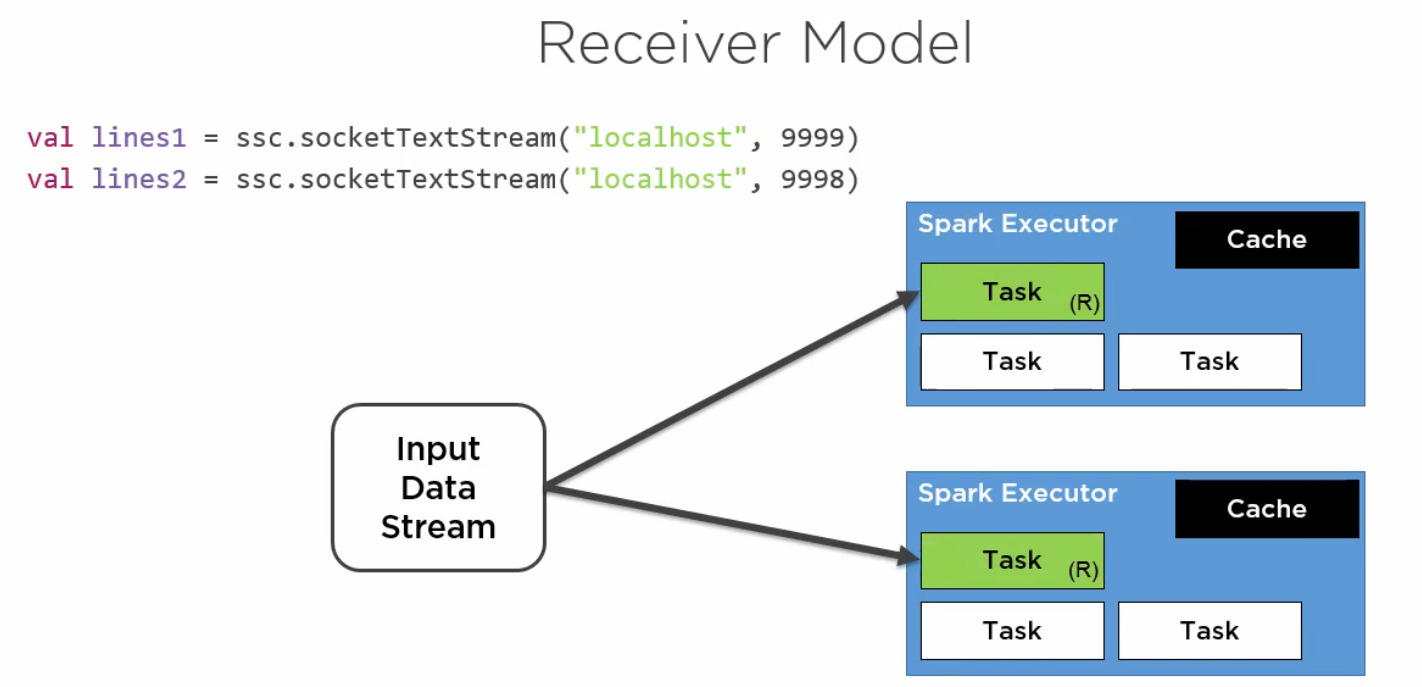
1 for driver

1 for task processing

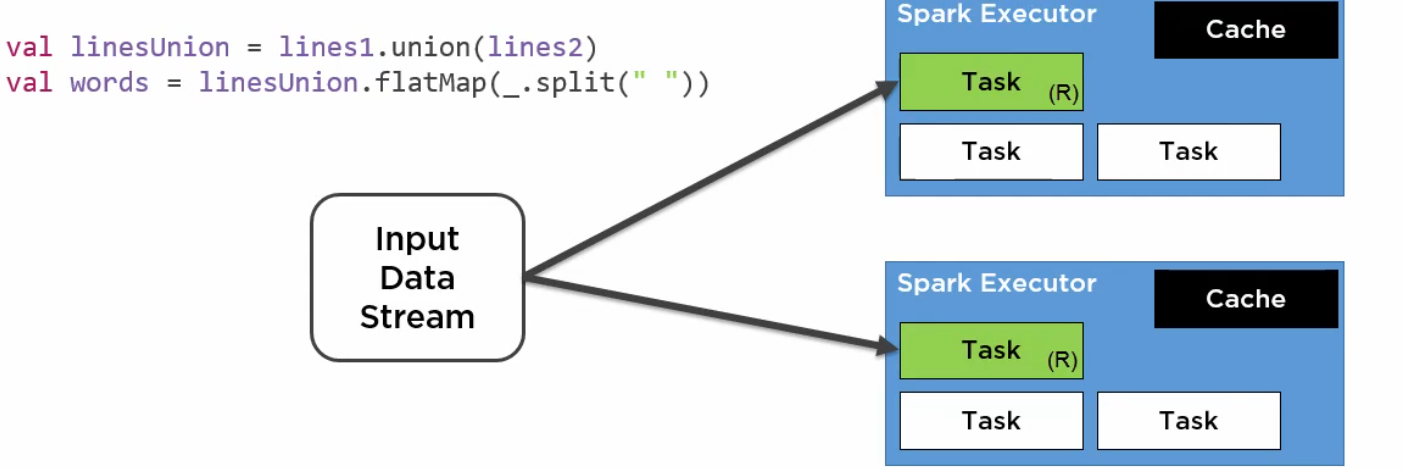


Note:

1. We can increase the parallelism by increasing the receiver inputs



1. We can combine the results of the 2 receivers using union



**CHECKPOINTING**

Can be used in normal spark applications

Required in streaming applications

**TWO TYPES OF CHECKPOINTING**

1)METADATA CHECKPOINTING (Driver recovery)

Configuration

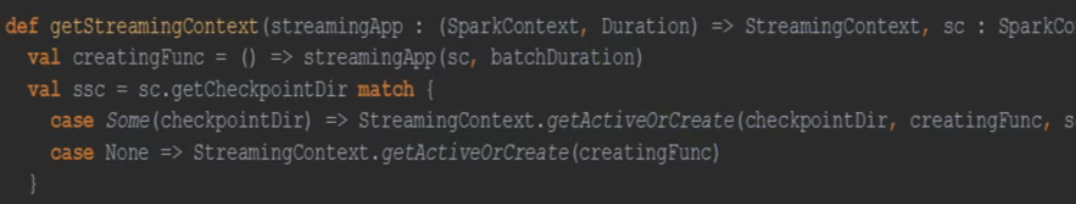
Dstream operations

Incomplete batches

Note: Driver failures causes losing executors

2) DATA CHECKPOINTING (Stateful Transformation)

Stateful transformations using data across batches



**ENCODER**

* Fundamental concept in the serialization and deserialization
* Spark SQL 2.0 uses the Serde framework for IO to make it effecient time- and space-wise

Encoders are integral (and internal) part of any Dataset[T] (of records of type T) with a Encoder[T] that is used to serialize and deserialize the records of this dataset.

* Encoders are modelled in Spark SQL 2.0 as Encoder[T] trait.
* Encoders know the schema of the records. This is how they offer significantly faster serialization and deserialization (comparing to the default Java or Kryo serializers).

trait Encoder[T] extends Serializable {

def schema: StructType

def clsTag: ClassTag[T]

}

case class Person(id: Long, name: String)

import org.apache.spark.sql.Encoders

scala> val personEncoder = Encoders.product[Person]

personEncoder: org.apache.spark.sql.Encoder[Person] = class[id[0]: bigint, name[0]: string]