Slide 6:
You heard me mention direct acyclic graph several times now. On this slide, you will see how to view the DAG of any particular RDD. A DAG is essentially a graph of the business logic and does not get executed until an action is called – often called lazy evaluation.

To view the DAG of a RDD after a series of transformation, use the toDebugString method as you see here on the slide. It will display the series of transformation that Spark will go through once an action is called. You read it from the bottom up. In the sample DAG shown on the slide, you can see that it starts as a textFile and goes through a series of transformation such as map and filter, followed by more map operations. Remember, that it is this behavior that allows for fault tolerance. If a node goes offline and comes back on, all it has to do is just grab a copy of this from a neighboring node and rebuild the graph back to where it was before it went offline.

Slide 7
In the next several slides, you will see at a high level what happens when an action is executed.

Let's look at the code first. The goal here is to analyze some log files. The first line you load the log from the hadoop file system. The next two lines you filter out the messages within the log errors. Before you invoke some action on it, you tell it to cache the filtered dataset – it doesn't actually cache it yet as nothing has been done up until this point.

Then you do more filters to get specific error messages relating to mysql and php followed by the count action to find out how many errors were related to each of those filters.

Slide 8
Now let's walk through each of the steps. The first thing that happens when you load in the text file is the data is partitioned into different blocks across the cluster.

Slide 9
Then the driver sends the code to be executed on each block. In the example, it would be the various transformations and actions that will be sent out to the workers. Actually, it is the executor on each workers that is going to be performing the work on each block. You will see a bit more on executors in a later lesson.

Slide 10
Then the executors read the HDFS blocks to prepare the data for the operations in parallel.

Slide 11
After a series of transformations, you want to cache the results up until that point into memory. A cache is created.

Slide 12
After the first action completes, the results are sent back to the driver. In this case, we're looking for messages that relate to mysql. This is then returned back to the driver.

Slide 13
To process the second action, Spark will use the data on the cache -- it doesn't need to go to the HDFS data again. It just reads it from the cache and processes the data from there.

Slide 14
Finally the results go back to the driver and we have completed a full cycle.

Slide 15:
So a quick recap. This is a subset of some of the transformations available. The full list of them can be found on Spark's website.

Remember that Transformations are essentially lazy evaluations. Nothing is executed until an action is called. Each transformation function basically updates the graph and when an action is called, the graph is executed. Transformation returns a pointer to the new RDD.

I'm not going to read through this as you can do so yourself. I'll just point out some things I think are important.

The flatMap function is similar to map, but each input can be mapped to 0 or more output items. What this means is that the returned pointer of the func method, should return a sequence of objects, rather than a single item. It would mean that the flatMap would flatten a list of lists for the operations that follows. Basically this would be used for MapReduce operations where you might have a text file and each time a line is read in, you split that line up by spaces to get individual keywords. Each of those lines ultimately is flatten so that you can perform the map operation on it to map each keyword to the value of one.

The join function combines two sets of key value pairs and return a set of keys to a pair of values from the two initial set. For example, you have a K,V pair and a K,W pair. When you join them together, you will get a K, (V,W) set.

The reduceByKey function aggregates on each key by using the given reduce function. This is something you would use in a WordCount to sum up the values for each word to count its occurrences.

Slide 16:
Action returns values. Again, you can find more information on Spark's website. This is just a subset.

The collect function returns all the elements of the dataset as an array of the driver program. This is usually useful after a filter or another operation that returns a significantly small subset of data to make sure your filter function works correctly.

The count function returns the number of elements in a dataset and can also be used to check and test transformations.

The take(n) function returns an array with the first n elements. Note that this is currently not executed in parallel. The driver computes all the elements.

The foreach(func) function run a function func on each element of the dataset.