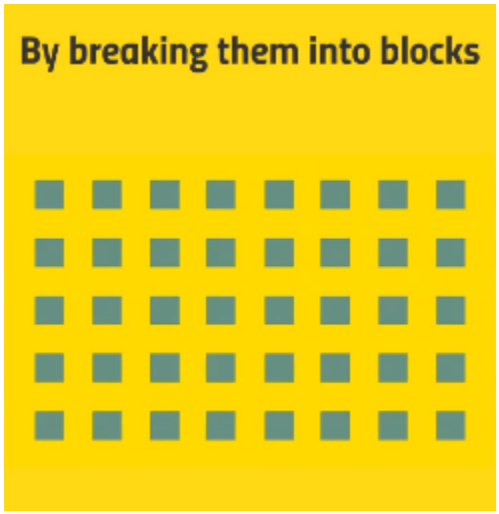
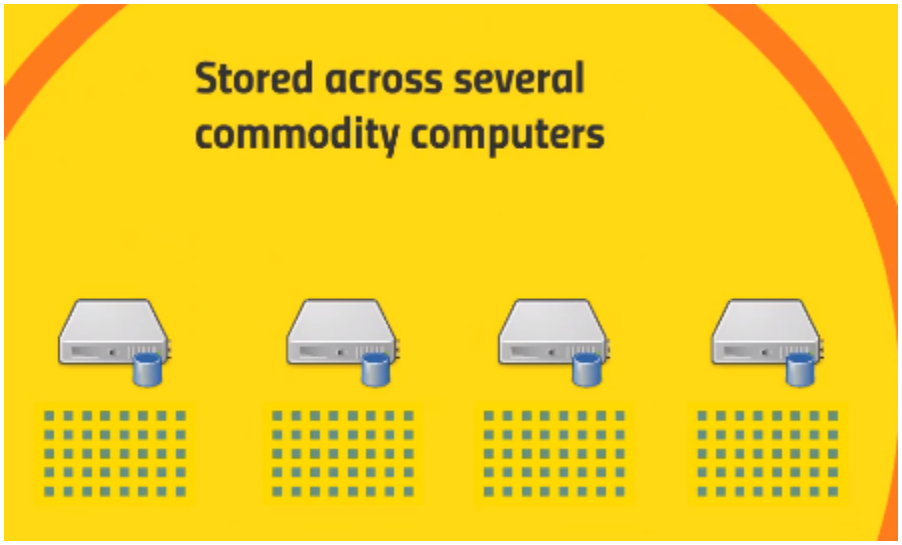
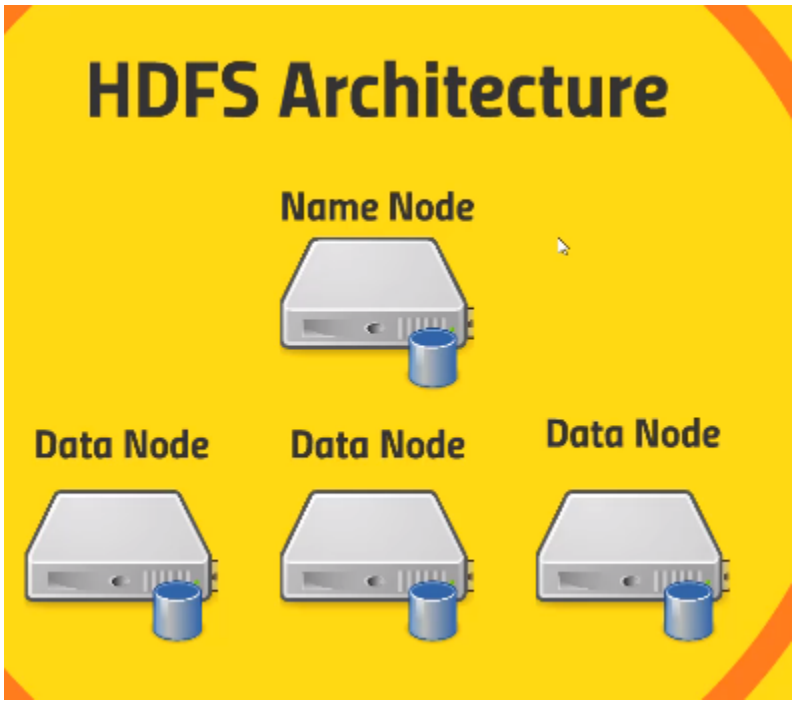
HDFS: What it is, and how it works

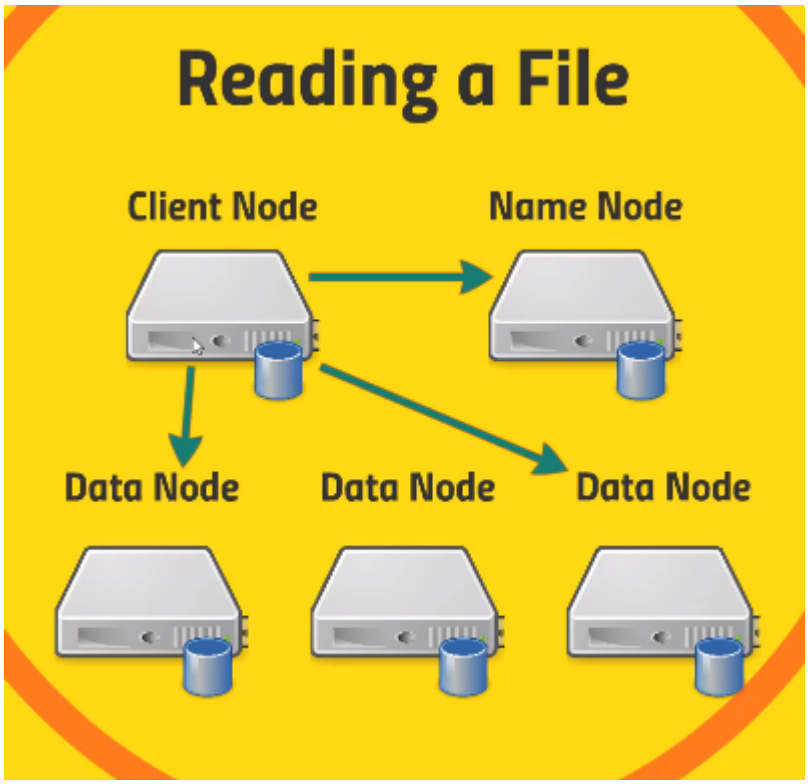
1. It allows your big data to be stored across an entire cluster in a distributed manner in a reliable manner and allows your applications to analyze that data to access that data quickly and reliably.
2. It is made to handle big data

3. 

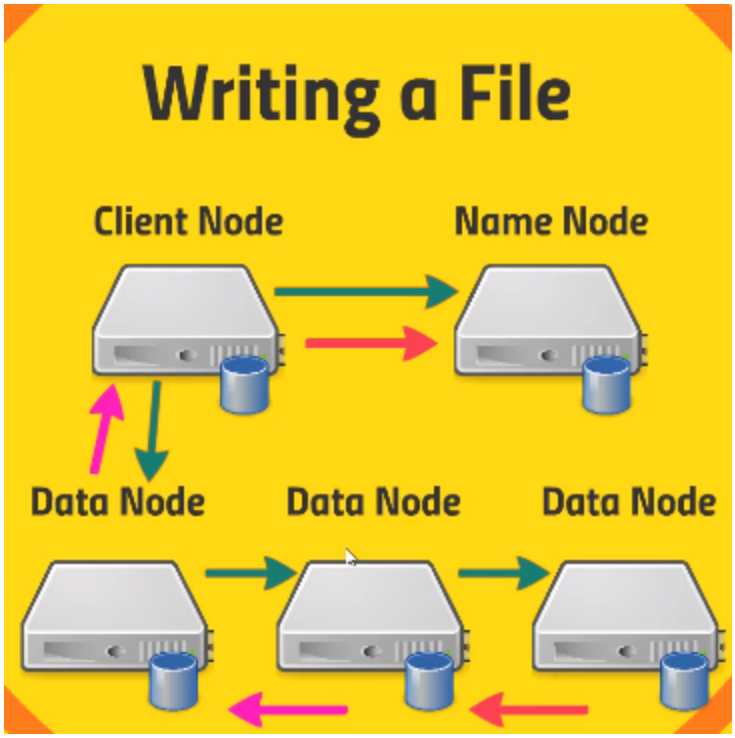
4. There blocks of data will be stored across several commodity computers



5. So like I said those blocks can be stored across several commodity computers and it doesn't just store one copy of each block. In order to handle failure.
6. it will actually store more than one copy of each block and so that way if one of these individual computers goes down HDFS can deal with that and actually start retrieving information from a different computer that had a backup copy of that block and it can do this in a clever way such that if a single node goes down you don't actually lose any blocks from any given file because there'll be another block stored somewhere else at all times.
7. HDFS Architecture
8. 

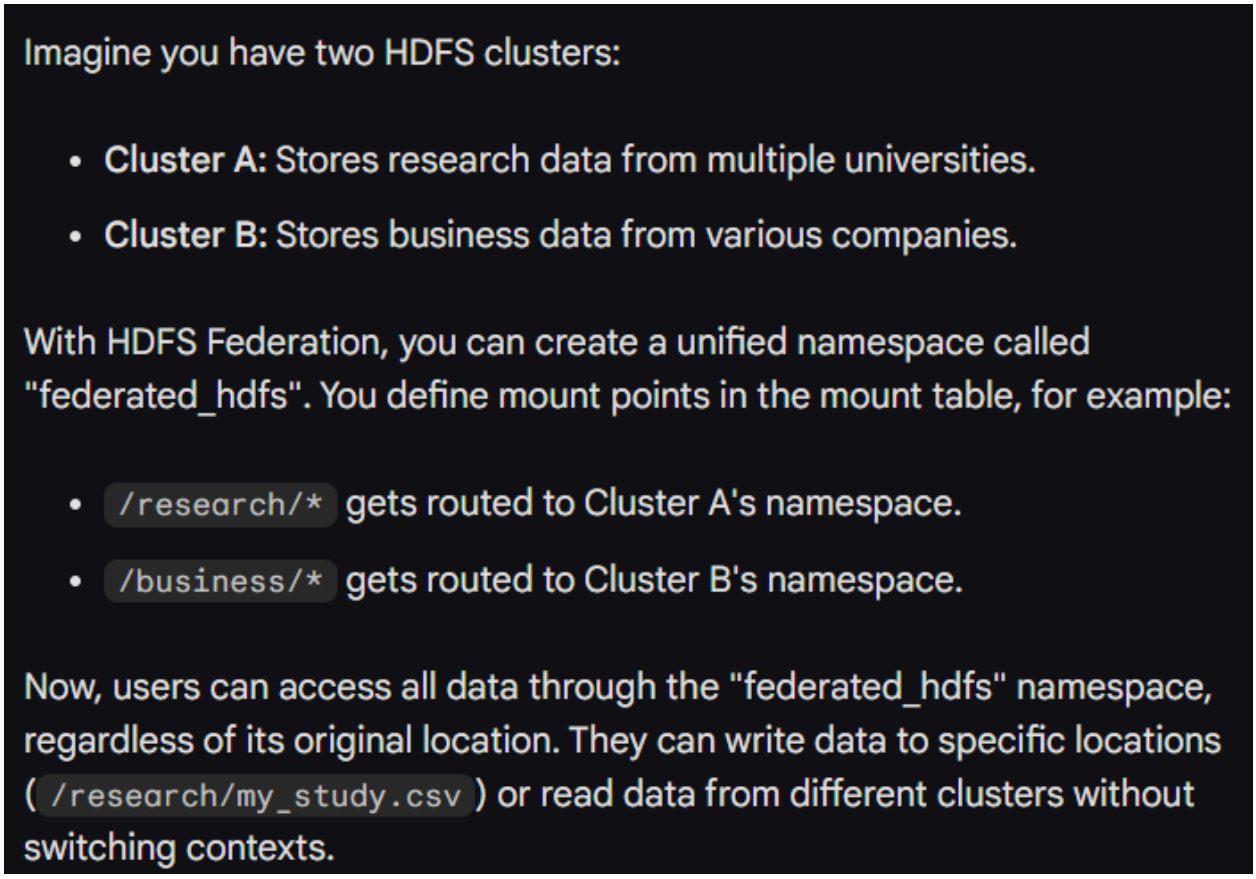
9. Here name node tracks all the data nodes..so basically it maintains a table under some virtual directory structure and it knows wer to go to actually find copy of every block
10. It also maintains data info about file changes,modifications etc
11. Data Node are the ones which stores these blocks of data ..and these data nodes talk with each other to maintain the data copies etc
12. Reading a file in hdfs
13. 

14. Here the client node sends request to name node.."I need this files" then the name node "give the address of the data nodes where this files/data are living
15. It's all happening under the hood but architecturally The point is your client would first talk to the name node to figure out where that file is stored which blocks are on which data nodes and which ones are the most efficient data nodes for you to reach from your client and it will then direct your client to go retrieve data directly from those data nodes that it wants.
16. What if we're writing a file? Its little complicated

17. 

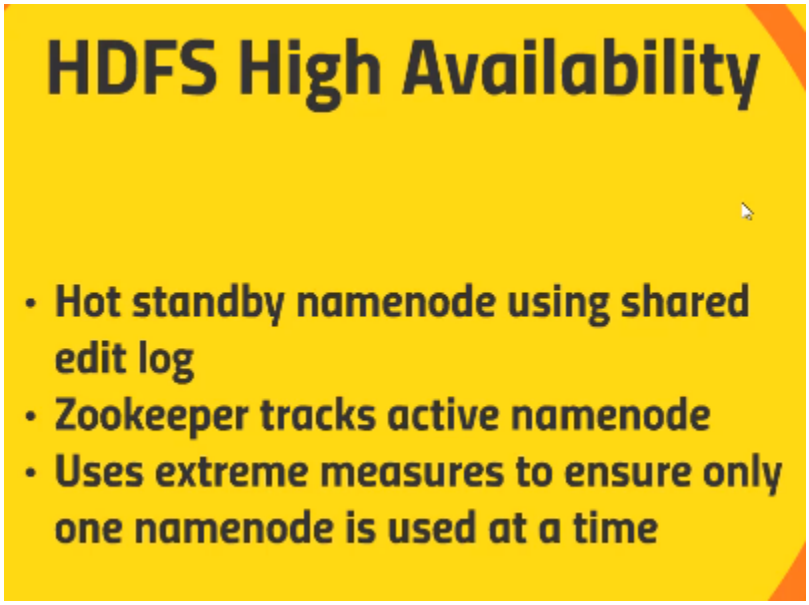
18. First the client sends request to Name node “I need to write these files” ..Now name node stores the meta info and replied back to client node to which data nodes these files must be assigned
19. Then client node sends file to a single data node..after receiving the file..the data nodes communicate each other and shares the back up file.
20. At last the name node will record the fact that this new file exists and it is stored in replicated manner across the data nodes
21. What if the name node crashes? 

22. If NN(name node) crashes …one method we can do is to store the metadata of the name node frequently, If the name node crashes then we can copy this metadata to a new name node
23. 2nd method is to maintain secondary namenode
24. All it's doing is maintaining a merged copy of the edit log from your primary name node. So it's not really a name node per se it's just maintaining a copy at all times of that edit log So this is a little bit better than the previous situation where if you do need to restore from a failed
25. 3rd method 

26. 

27. So the relevance here to resilience is that if one of those name notes went down well at least you only lose a portion of your data. Not all of it. Right. So you don't have to worry about restoring that given name node which might not be quite as catastrophic
28. 4th

29. If we cannot stand losing a name node and we want immediate namenode then



30. **Imagine a Library with a Backup Librarian:**

    - **The Head Librarian (Active Namenode):** Manages the entire library's catalog and operations, tracking book locations and handling requests.
    - **The Assistant Librarian (Standby Namenode):** Works closely with the head librarian, constantly mirroring their knowledge of the catalog and operations.
    - **The Journal Shelf (JournalNodes):** Acts as a detailed log of all recent book movements and transactions, ensuring consistency.

    When the Head Librarian is Absent:

    1. **Unforeseen Circumstance:** The head librarian suddenly falls ill and can't work.
    2. **Assistant Steps Up:** The assistant librarian, fully prepared and equipped with the journal shelf's records, seamlessly takes over as the new head librarian.
    3. **Service Continues:** Library patrons barely notice the change and continue accessing books and services without major interruption.

31. **Key Components in HDFS HA:**

    - **Active Namenode:** The primary Namenode, responsible for managing the HDFS namespace, metadata, and client requests.
    - **Standby Namenode:** A secondary Namenode that mirrors the Active Namenode's state in real-time, ready to take over if needed.
    - **JournalNodes:** A set of daemons that maintain a durable log of all edits made to the filesystem, ensuring consistency during failover.

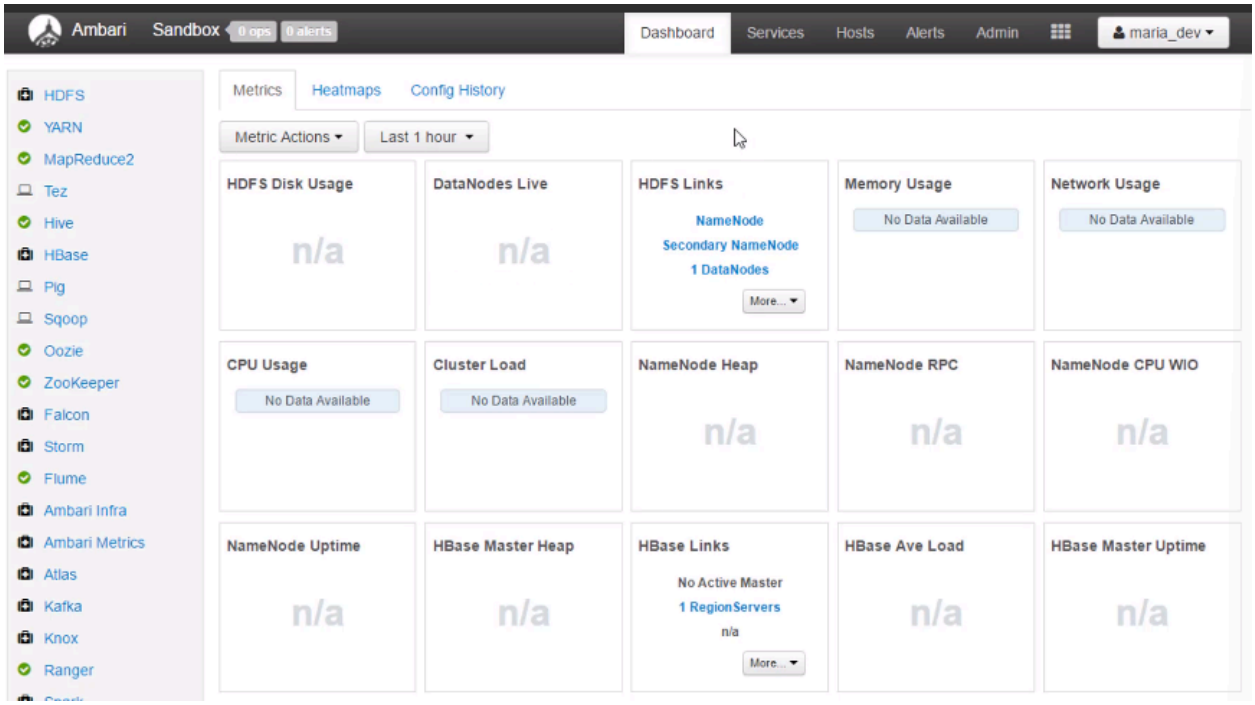
32. Uses extreme measures to ensure only one namenode is used at a time to avoid read-write conflicts
33. We can HDFS thru this means

Using HDFS

UI (Ambari)
Command-Line Interface
HTTP / HDFS Proxies
Java interface
NFS Gateway

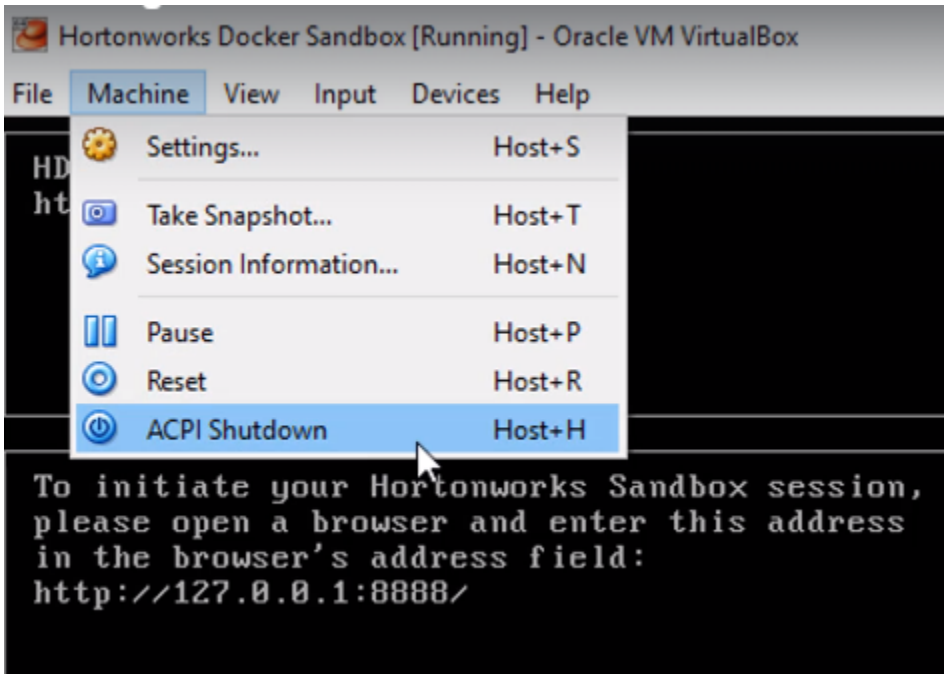
Installing MovieLens Dataset

1. First we go to Ambari



2. We created ml-100k folder in maria dev

3. Next We will upload u.data file and u.item file…
4. In this lecture we learned how to upload the files into ambari
5. If we want to close down the session we can just shut down our virtual machine



Installing the data using CLI

1. To list files in the hadoop file system we

```
[maria_dev@sandbox ~]$ hadoop fs -ls
Found 3 items
drwx------   - maria_dev hdfs          0 2016-11-11 20:53 .Trash
drwxr-xr-x   - maria_dev hdfs          0 2016-11-11 20:16 files-view
drwxr-xr-x   - maria_dev hdfs          0 2016-11-09 17:21 hive
[maria_dev@sandbox ~]$
```

2. To make a directory we use

```
[maria_dev@sandbox ~]$ hadoop fs -mkdir ml-100k
[maria_dev@sandbox ~]$ hadoop fs -ls
Found 4 items
drwx------   - maria_dev hdfs          0 2016-11-11 20:53 .Trash
drwxr-xr-x   - maria_dev hdfs          0 2016-11-11 20:16 files-view
drwxr-xr-x   - maria_dev hdfs          0 2016-11-09 17:21 hive
drwxr-xr-x   - maria_dev hdfs          0 2016-11-11 20:58 ml-100k
[maria_dev@sandbox ~]$
```

3. Now we will get u.data through a web server

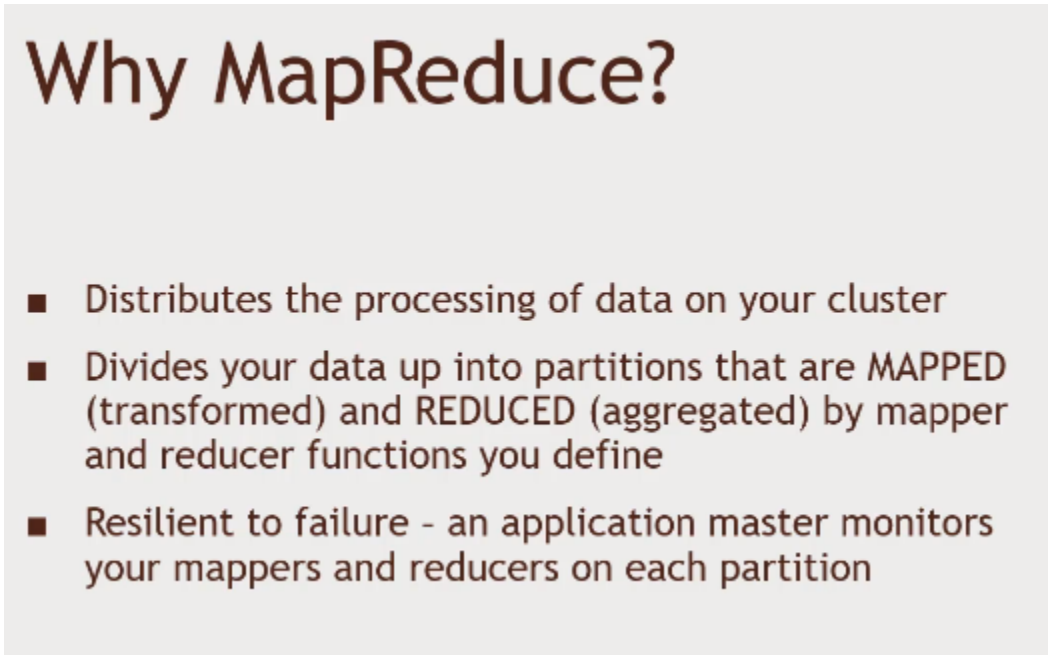
```
[maria_dev@sandbox ~]$ wget http://media.sundog-soft.com/hadoop/ml-100k/u.data
```

4. Next we will add this file into hadoop fs from local

```
[maria_dev@sandbox ~]$ hadoop fs -copyFromLocal u.data ml-100k/u.data
```

5. Now we have our file in HDFS
6. At last we clear our selves by removing the data

MapReduce Fundamental Concepts

1. 

2. So its job is to divide all your data up into partitions that can be processed in parallel across your cluster and it manages how that's done and how failure is handled and whatnot.
3. But basically mappers are what transform your data and reducers are what aggregates your data.

4.

- **Task:** Each researcher takes a section of books and extracts relevant information (title, author, keywords).
- **Process:**
  - Read books line-by-line.
  - Break down data into smaller, meaningful chunks (key-value pairs).
  - Sort pairs based on keywords.
- **Output:** Hand off sorted pairs to reducers.

Reducers (Senior Researchers):

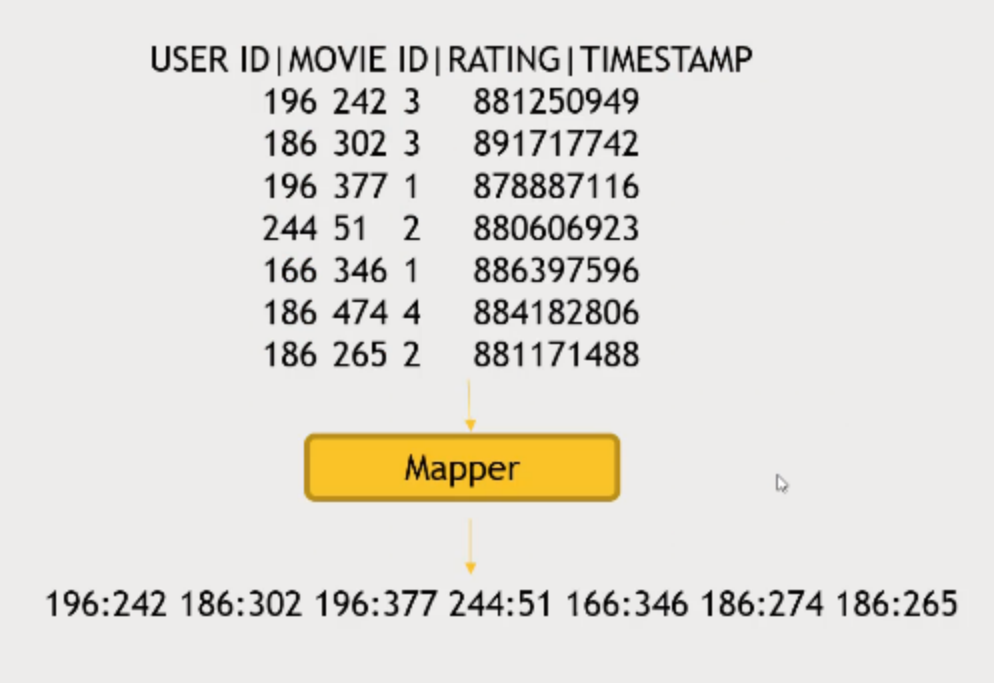
- **Task:** Synthesize information from multiple researchers.
- **Process:**
  - Receive sorted pairs from mappers.
  - Group pairs with the same keyword.
  - Perform calculations or aggregations (e.g., count books per author).
- **Output:** Produce final results (e.g., author popularity lists).

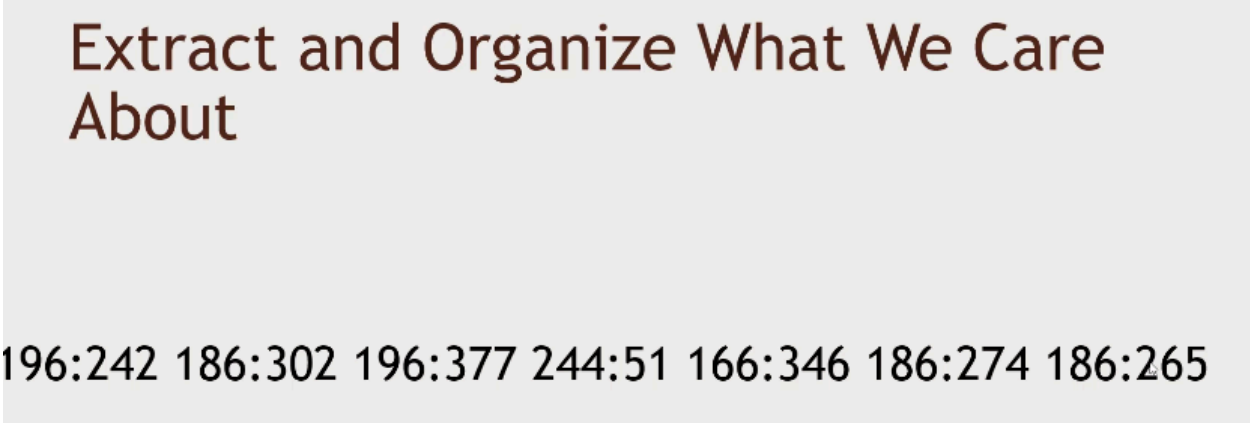
5. Lets understand with an example

6. # Example: MovieLens Data (u.data file)

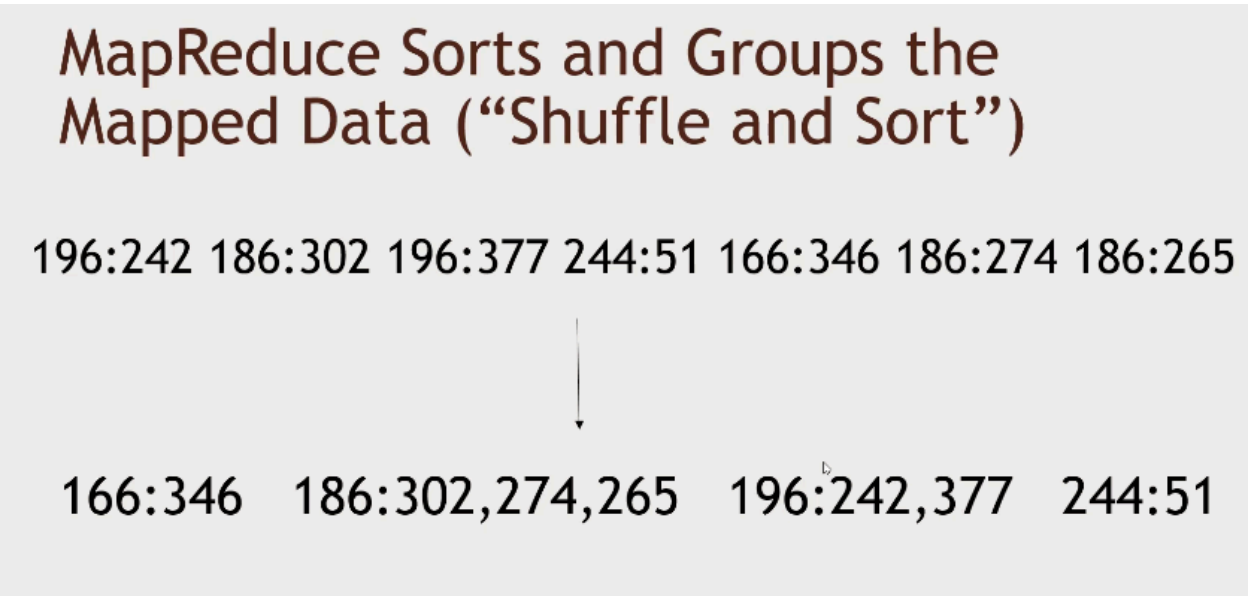
USER ID	MOVIE ID	RATING	TIMESTAMP
196	242	3	881250949
186	302	3	891717742
196	377	1	878887116
244	51	2	880606923
166	346	1	886397596
186	474	4	884182806
186	265	2	881171488

7. Mappers maps users to movies they watched

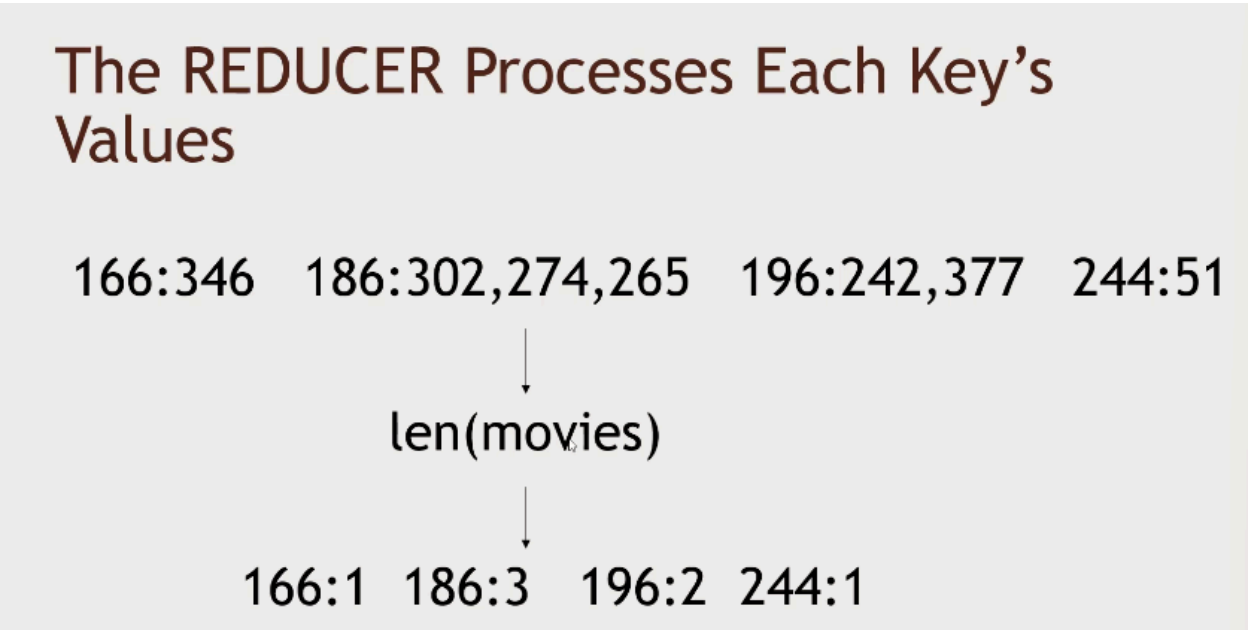
8. 

9. 

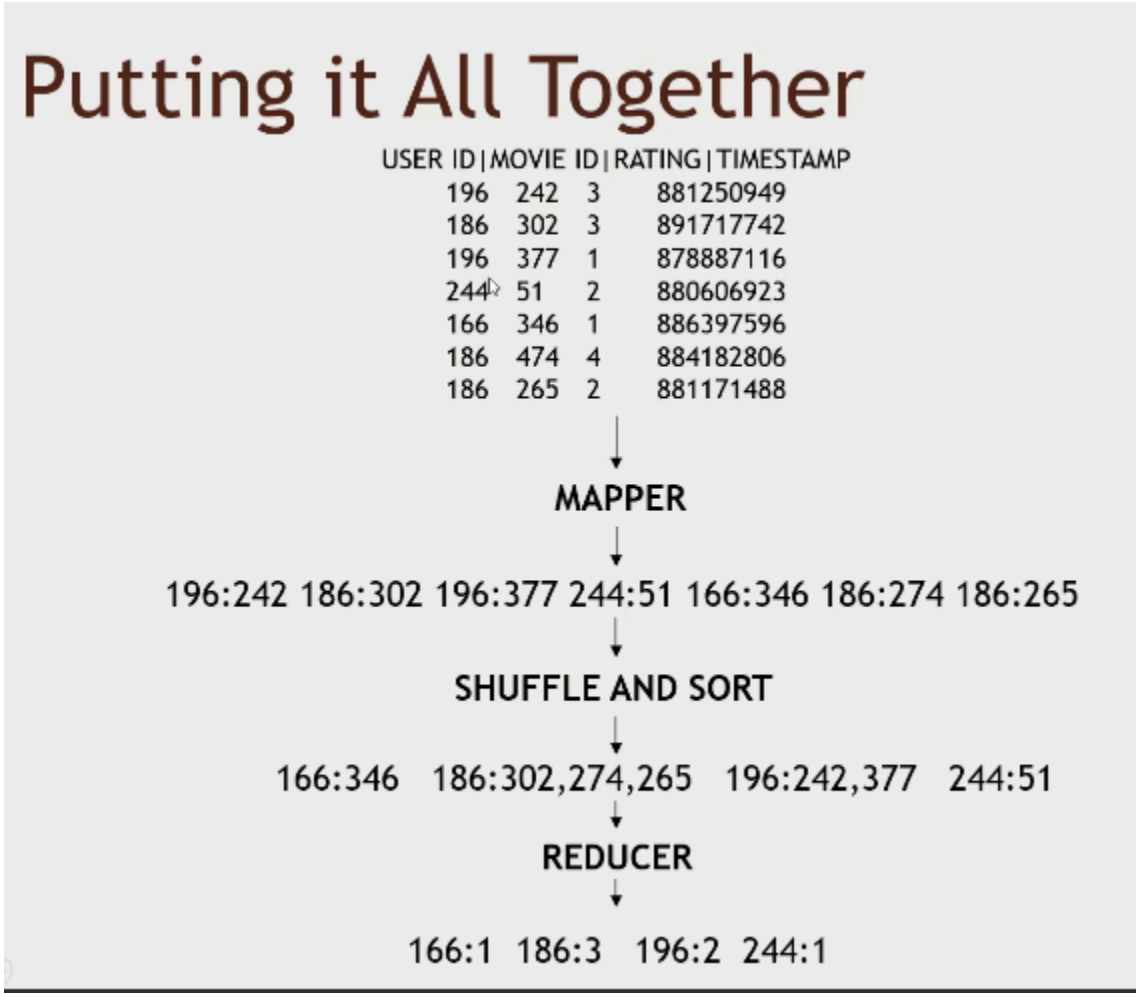
9. Shuffle & Sort



10.
11. Here user id 166 watch movie_id(346), similarly used id 186 watched 3 movies(302,274,265)
12. Reducers aggregates these values

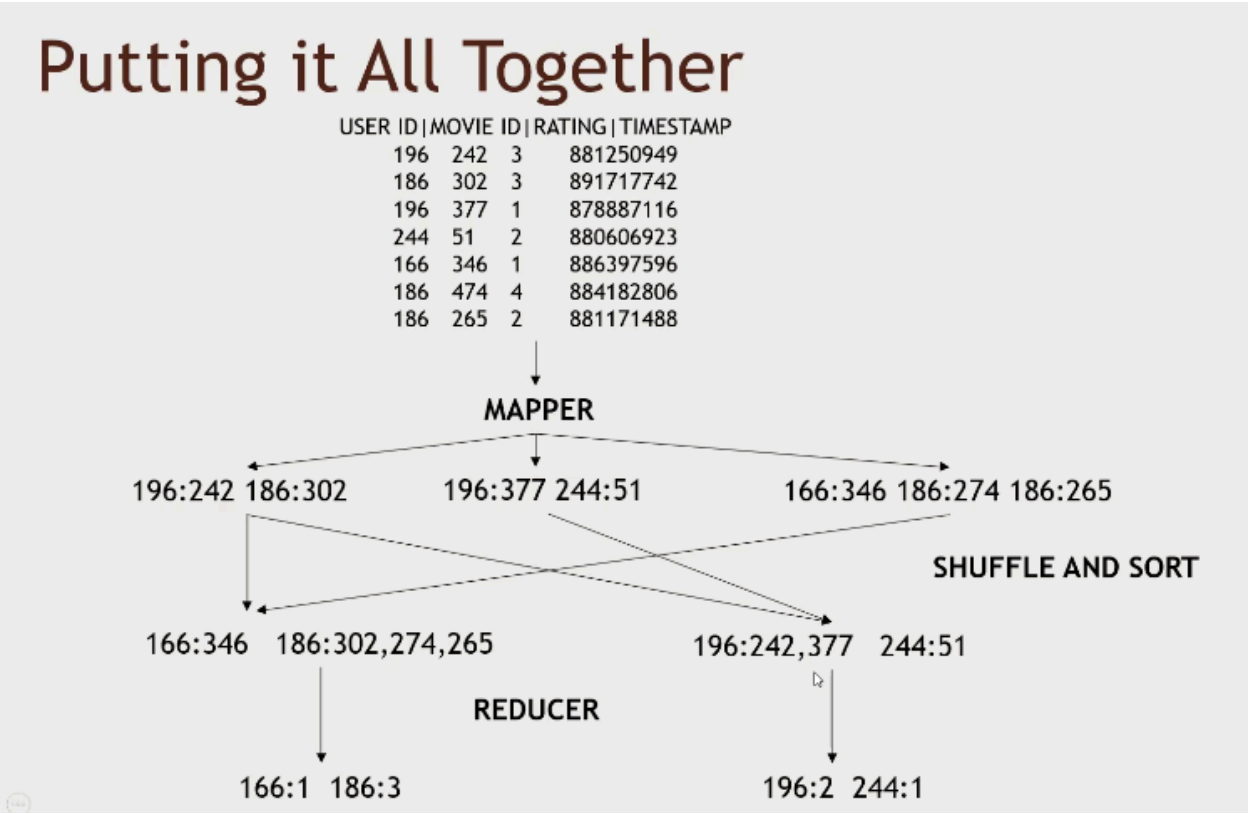


13. Putting it all together



How mapreduce distributes processing?

1. Lets see our prev example



2. Our mapper gives us the all key-pair
3. How are mappers and reducers written?
4.