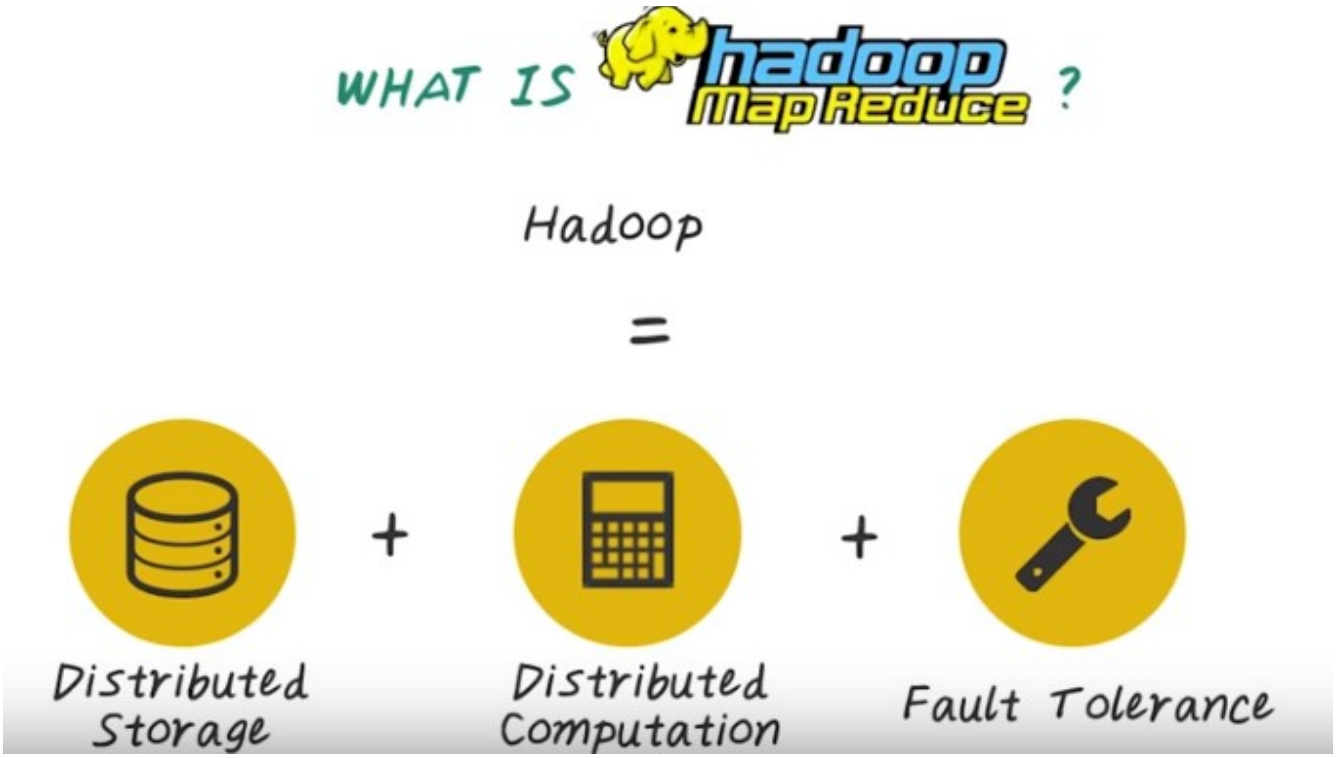
1. In the past couple of lessons, we have been talking about predictive modeling on big data. Today we're going to talk about a tool for processing big data called MapReduce. MapReduce is a powerful system that can perform some of the methods we have talked about so far, on big data set using distributed computation and distributed storage. We'll start by discussing what MapReduce is. Then we'll discuss how MapReduce takes care of fault tolerance ina distributed environment. Finally we'll talk about some of the analytics that can be performed when map produce and the limitations it carries.

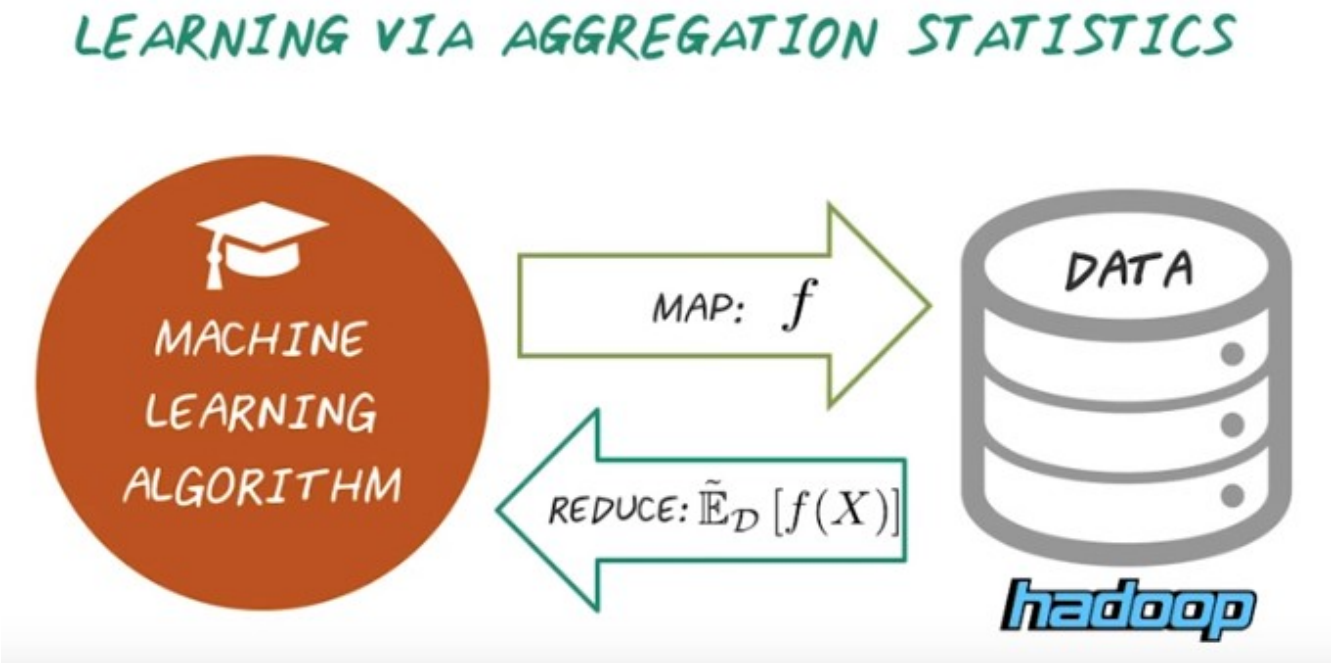
2. Now let's talk about Hadoop and MapReduce. So what is Hadoop or MapReduce? Is it a programming model for developer to specify parallel competition algorithms? Yup. We talk about MapReduce paradigm(范例). Is it an execution environment? (Yes.) Hadoop is the Java implementation of MapReduce and Hadoop Distributed File System. So it is an execution environment. Is it a software package? Sure is! In fact, there are many software tools have been developed to facilitate development effort for data science tasks, such as data processing, extraction, transform and loading process, statistic computation, and analytic modeling using Hadoop. In summary, Hadoop and MapReduce enables a powerful big data ecosystem by providing the combination of all these things (圖中的).



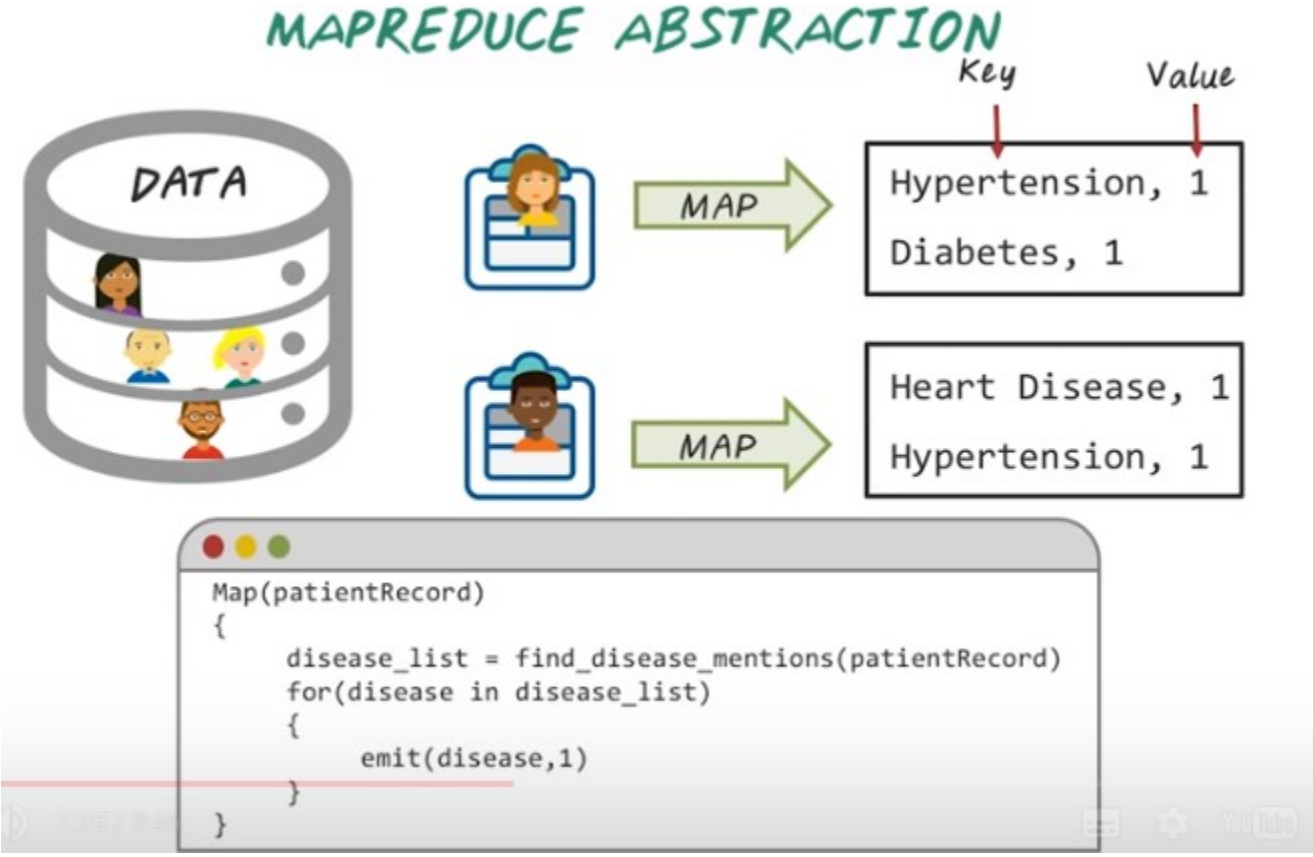
So in fact, MapReduce or Hadoop is a big data system that provides the following capability: distributed storage for large data set through Hadoop distributed file system, distributed computation through programming interface MapReduce, and fault tolerance systems in order to cope with constant system failures on large distributed systems that are built on top of commodity hardware.

3. Originally, MapReduce was proposed by Jeff Dean and Sanjay Ghemawat from Google in 2004 and were implemented inside Google as a proprietor software for supporting many of their search engine activities. Then later on, Apache Hadoop is developed as a open source software that mimic original Google's MapReduce systems. It was written in Java for distributing storage and distribute processing of very large dataset. To program on Hadoop systems, we have to use the programming abstraction MapReduce, which provides a very limited, but powerful programming paradigm for parallel processing on a large dataset. All the algorithm running on MapReduce or Hadoop have to be specified

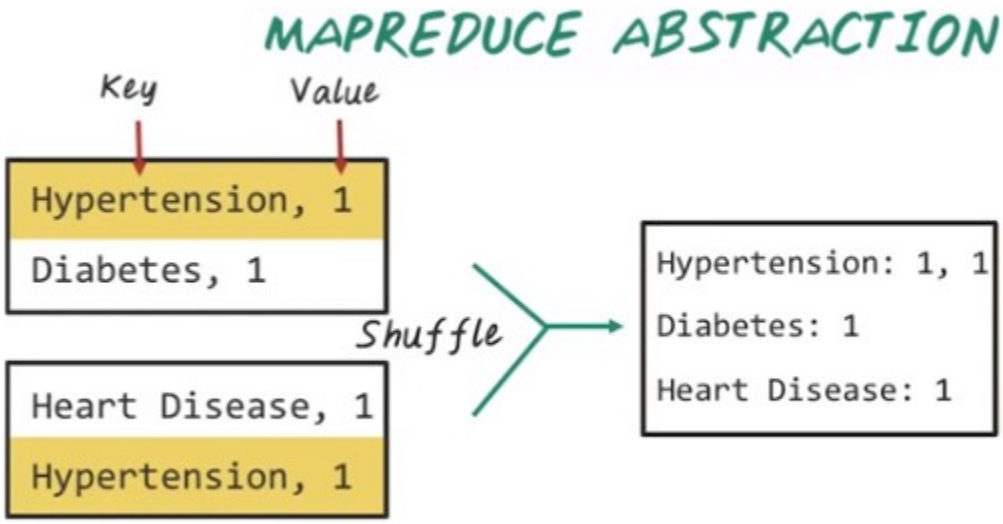
as MapReduce programs. The reason for this very constrained programming model is to support super scalable, parallel and fault tolerance implementation of data processing algorithm that can run on large dataset. To utilize Hadoop and MapReduce for data analytics, we have to understand and master common patterns for computation using MapReduce. We explain this next.



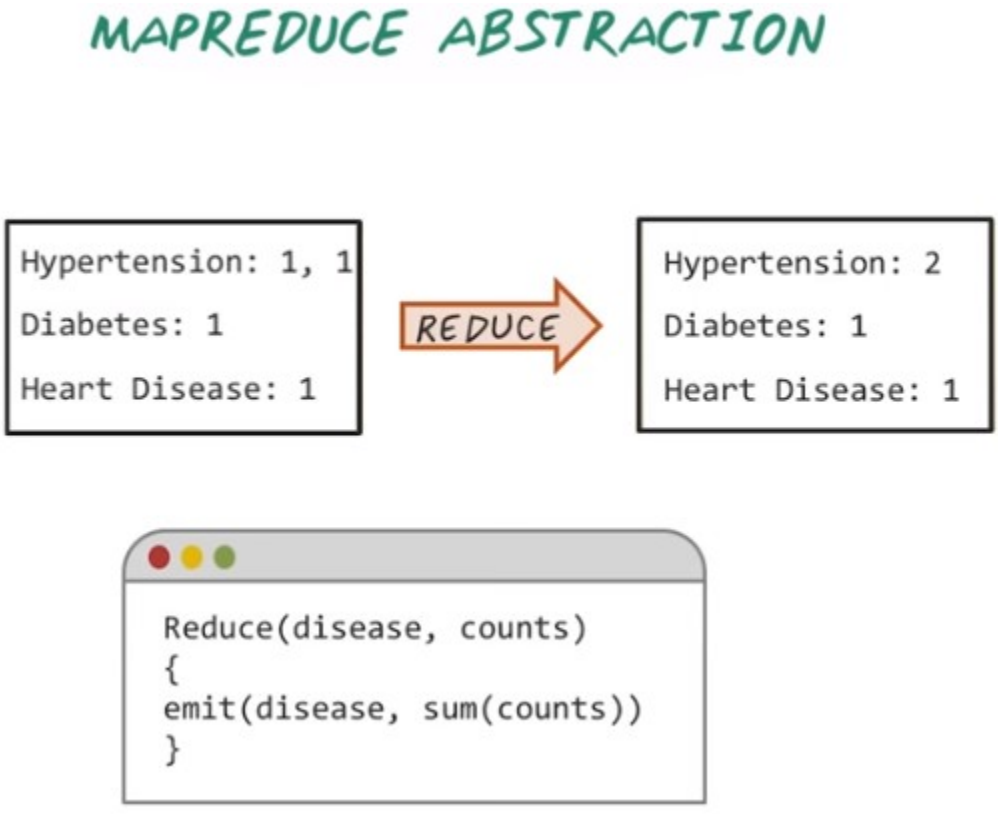
4. There's a fundamental pattern for writing a data mining or machine learning algorithm using Hadoop is to specify machine learning algorithms as computing aggregation statistics. Say we want to implement in a machine learning algorithm for identifying the most common risk factors of heart failure. We need to decompose the algorithm into a set of smaller computation units. In particular, we need to specify a map function f, where f will be applied to all the heart failure patients in the large database. For example, we want to extract the list of risk factors related to heart failure appear in each patient's record. Then result from this map function will be aggregated by a reduce function. For example, instead of listing the risk factors for each patient, we want to compute the frequency of each risk factor over the entire population. Then in this case, the reduce function would do that by performing the aggregation statistic on the result from the map function. So this process is quite abstract. Next we'll go into more details to explain why such abstraction is required and what are the benefit and limitation of this abstraction.



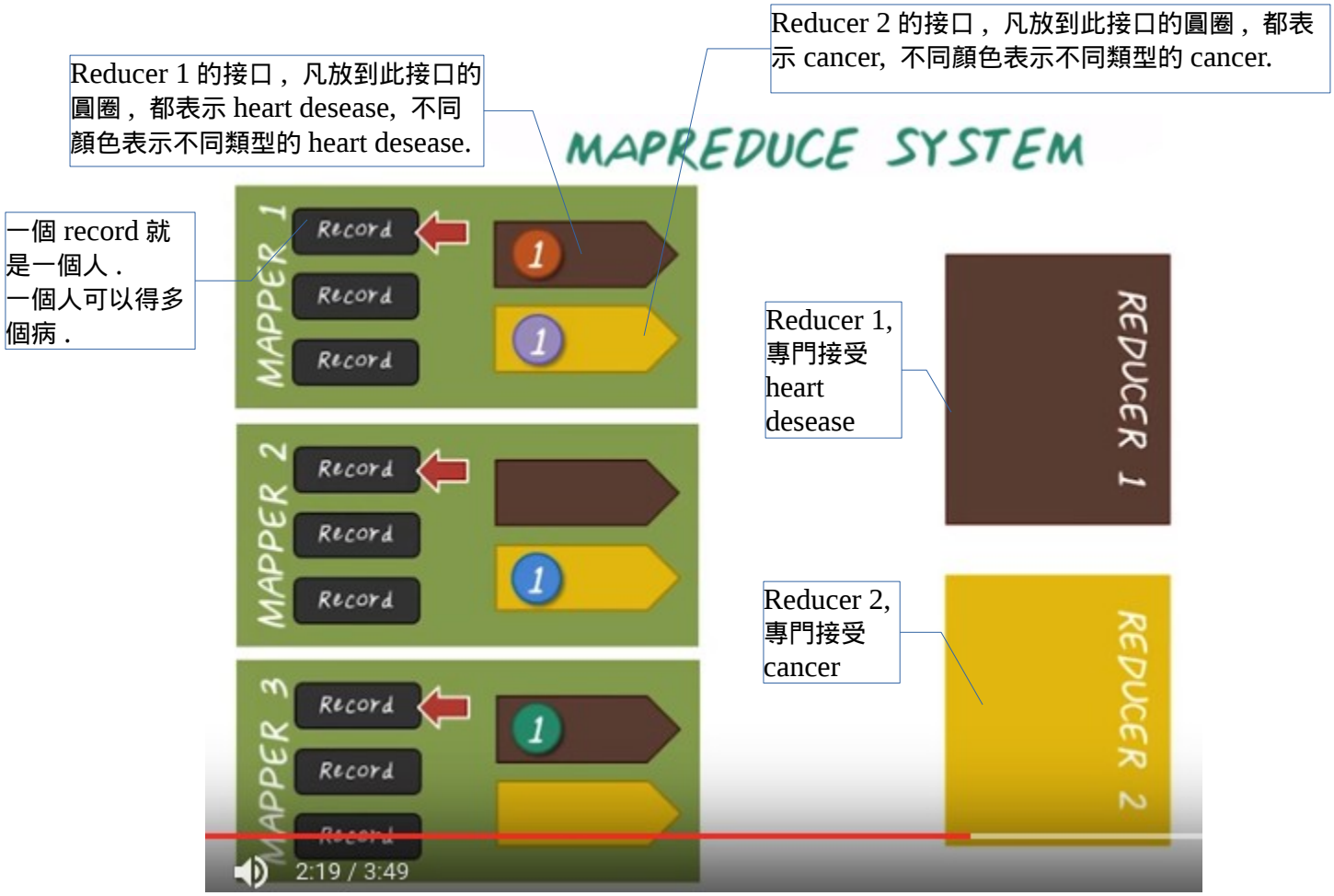
5. Here's an example to illustrate MapReduce in more details. Say we have a large database of patients stored in the Hadoop distributed file system. Each patient is stored as a separate record, and each record consists of the history of this patient encounter. For example, the diagnosis, medication, procedure, and clinical notes are all stored in those patient records. Our goal is to write up MapReduce programs to compute the number of cases in each disease. To do this in MapReduce, we first specify the map function that goes through each patient records, and extracting the disease mentions and output that. For example, we have two patient record over here, and when we apply the map function, we'll first find the disease mentioned in this record. For example, for the first record, we identify hypertension and diabetes are in the record. Then, for the list of disease mentioned, we identified inside this record, we emit the disease name and the value 1. And the output from each map function looks like this. For example, for this patient we have a hypertension and value 1, and diabetes with a value 1. For the second patient, he has heart disease with value 1 and a hypertension with value 1. And the disease name in this case is the key, and the value 1 is the value. So map function would process input record, and output a set of key value pairs. You notice that the same key value pairs may appear as output from different map function. For example (Hypertension, Value 1) happened in this output and also in this output.

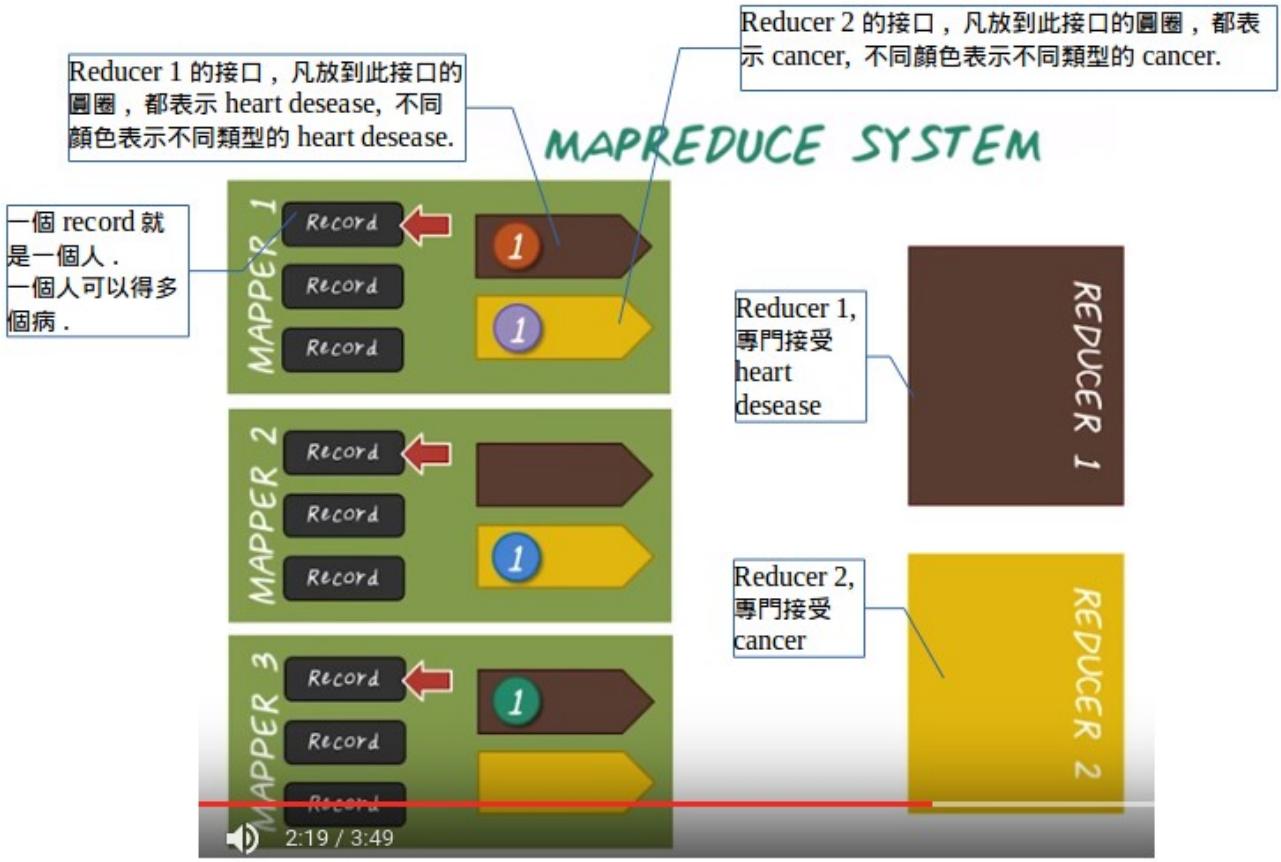


So, the next phase will combine them. All the output from map function will be processed internally by Hadoop. In particular, all those output will be shuffled and aggregated. For example, hypertension happened twice over here, and after the shuffling and combination phase, we have hypertension and all those associate value. Diabetes, on the other hand, only happened once in this record, so the corresponding lists of value only have one value, and similarly for heart failure, we only have one value here.



And this intermediate result will be the input for the reduce function. To write a map reduce program we need to specify the map function, and the reduce function. So here's one example for a reduce function. The reduce function will take the disease, key, and a list of disease values. For example, we have hypertension as the disease key, and we have a values 1 and 1. So in this reduce function, we'll take the disease and the list of value, and we sum up those value. So the result of the reduce function would give us hypertension value 2, diabetes value 1, and heart disease value 1. So in this very simple example, you may feel like this two-stage process is very strange. But if we're dealing with large data set, with billions of records, this two-faced process is very important. And next, we'll explain how internally MapReduce system works. And you understand why we have to specify the computation in these two phases.

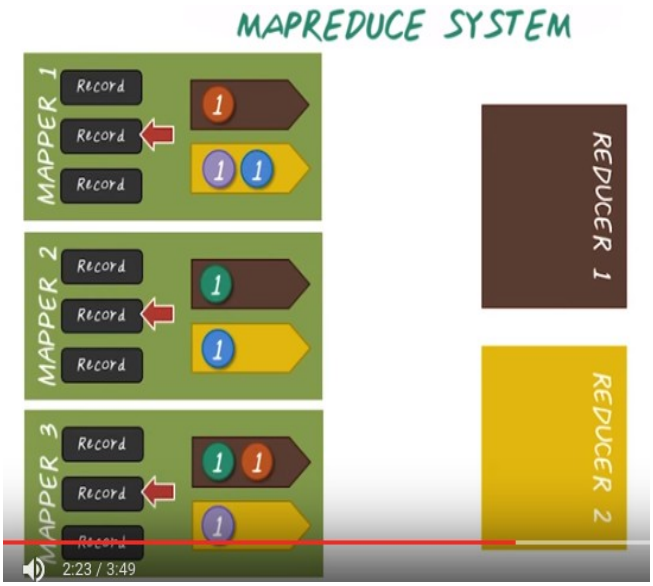


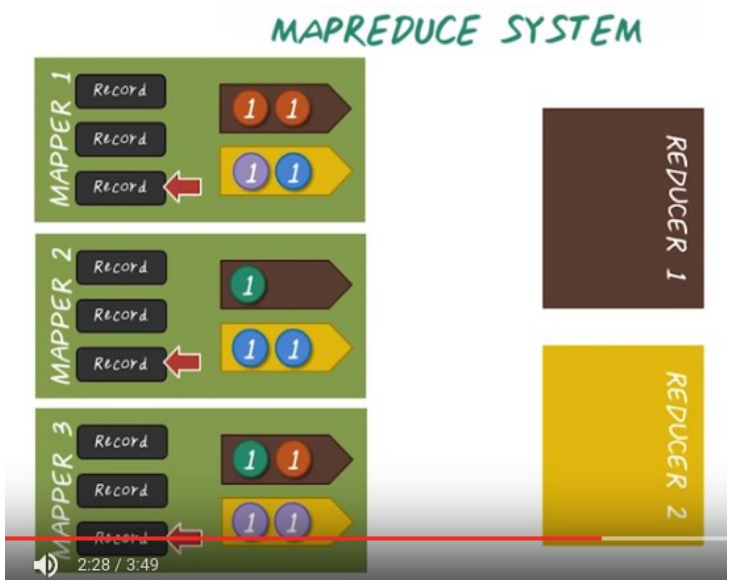


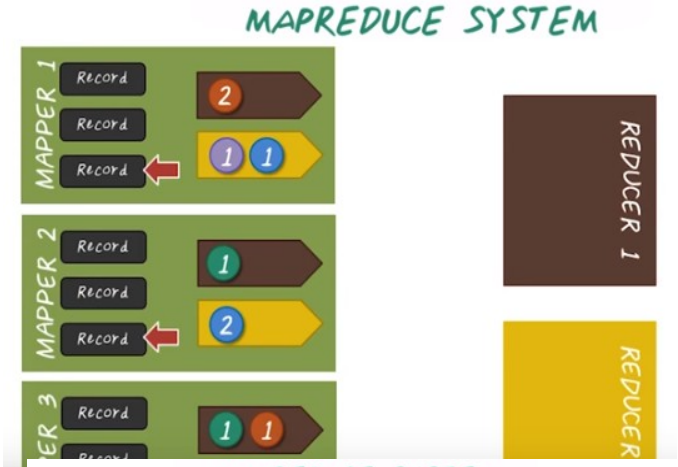
上圖是上上圖的截圖版, 為了防止格式改變.

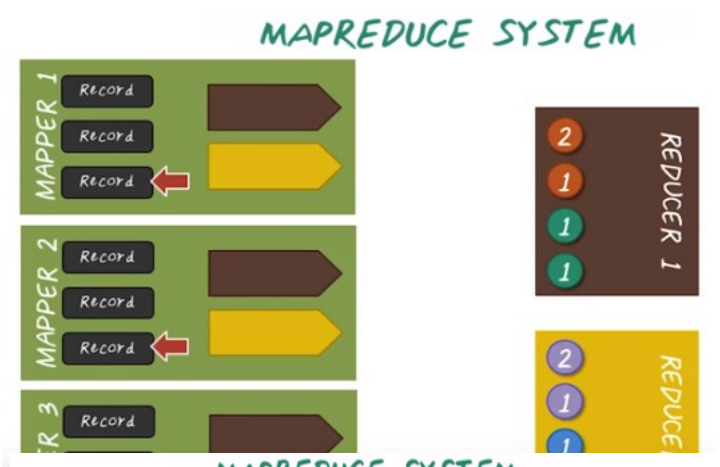
6. So far, we illustrate the high level ideas on how to write a mapreduce program. But how does mapreduce work? And why it requires such a strange two-phased process? We have to realize, the real word data set is often too big to be stored and processed on a single machine. So, we have to split the data into partitions so that each partition can be stored and processed in parallel on multiple machines. So, mapreduce systems has two components, mappers and reducers. So, all those data will be partitioned and processed by multiple mappers. And each mapper deal with a partition, which is a subset of records in the entire dataset. For example, mapper 1 processes three records, mapper 2 processes another three records and mapper 3 process the remaining records. Then, we have reducers, in this case, we have reducer 1 and reducer 2. So, they are also divided the work by processing intermediate result, in certain ranges. For example, reducer 1 will be in charge of heart disease, and reducer 2 will be in charge of cancers. So, in the map phase, each mapper will iteratively apply the map function on the data that they're in charge of. And also, they will prepare the output that will be sent to the corresponding reducer. For example, we have a set of intermediate result prepared for reducer one. And we have a set of intermediate result prepared to be sent to reducer 2. And across different mappers, this process are happening in parallel. For example, mapper 1 is processing the first record by applying the map function. So, identify two diseases mentioned. One is to be sent to reducer 1, is about heart disease. And second disease mentioned is about cancer, it will be sent to the reducer 2. And the same process is happening concurrently on different mappers. For mapper 2, the first record would emit one disease mentioned for reducer 2. And for mapper 3, we emit one record for reducer 1.

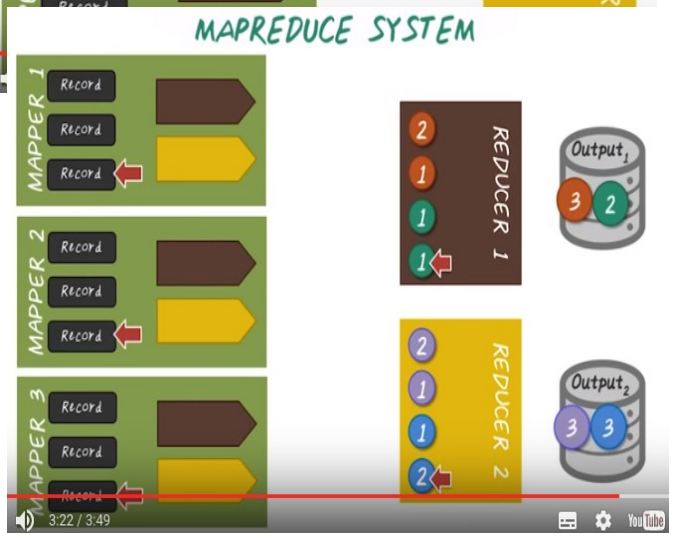
以下先從左往右看完一行, 再看下一行:

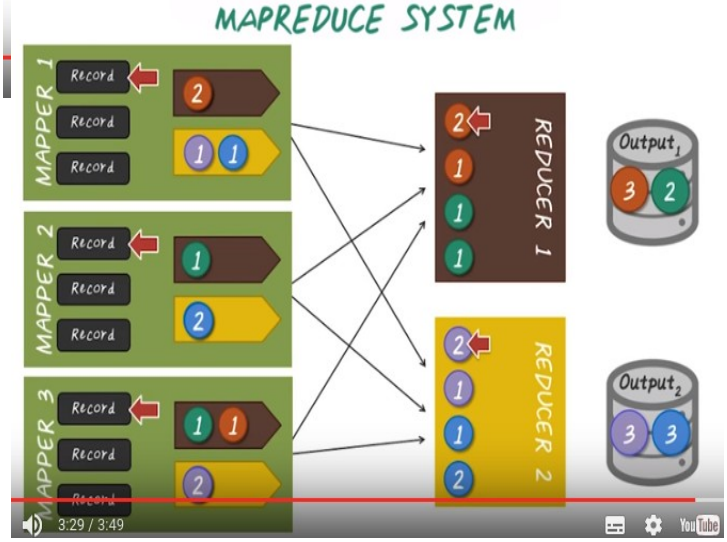






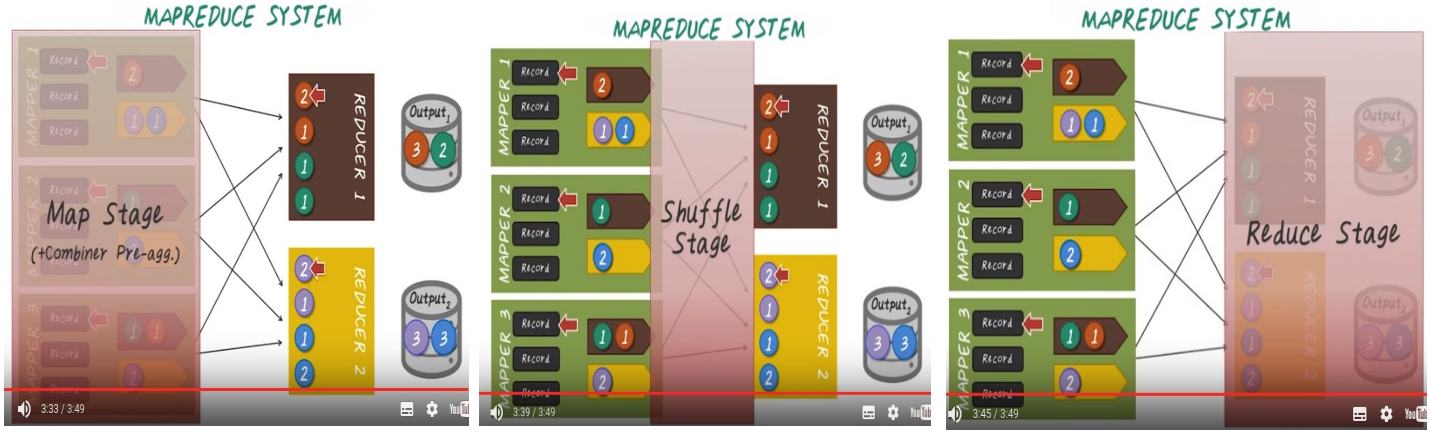




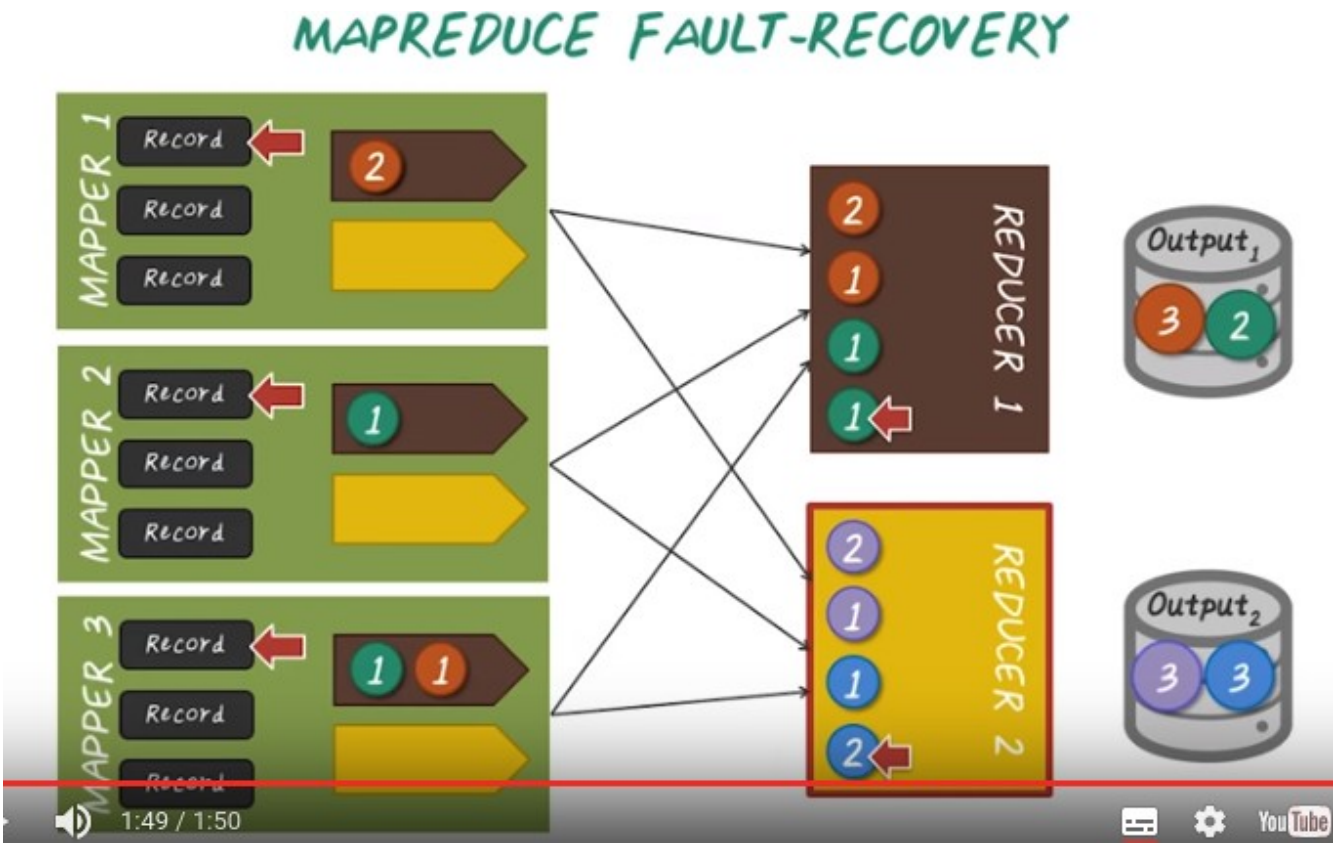


And this process is iteratively going through all the records. And all those intermediate results will be processed internally through a combination and shuffling process. So, we'll combine the values for the same disease at each mapper, then stand out all those intermediate results to the corresponding

reducers. So, at the reducer side, once they receive those records, they can start generating the final output by applying the reduce function. For example, this R2 hypertension mentions and we obtain the final count, that's 3. And we're going through the other disease and compute the final count. Then output the result, so these are the final output from two different reducers. For example, for hypertension, it happened 3 times. For another heart disease, it happened 2 times, for these two types of cancers, they both happen 3 times.

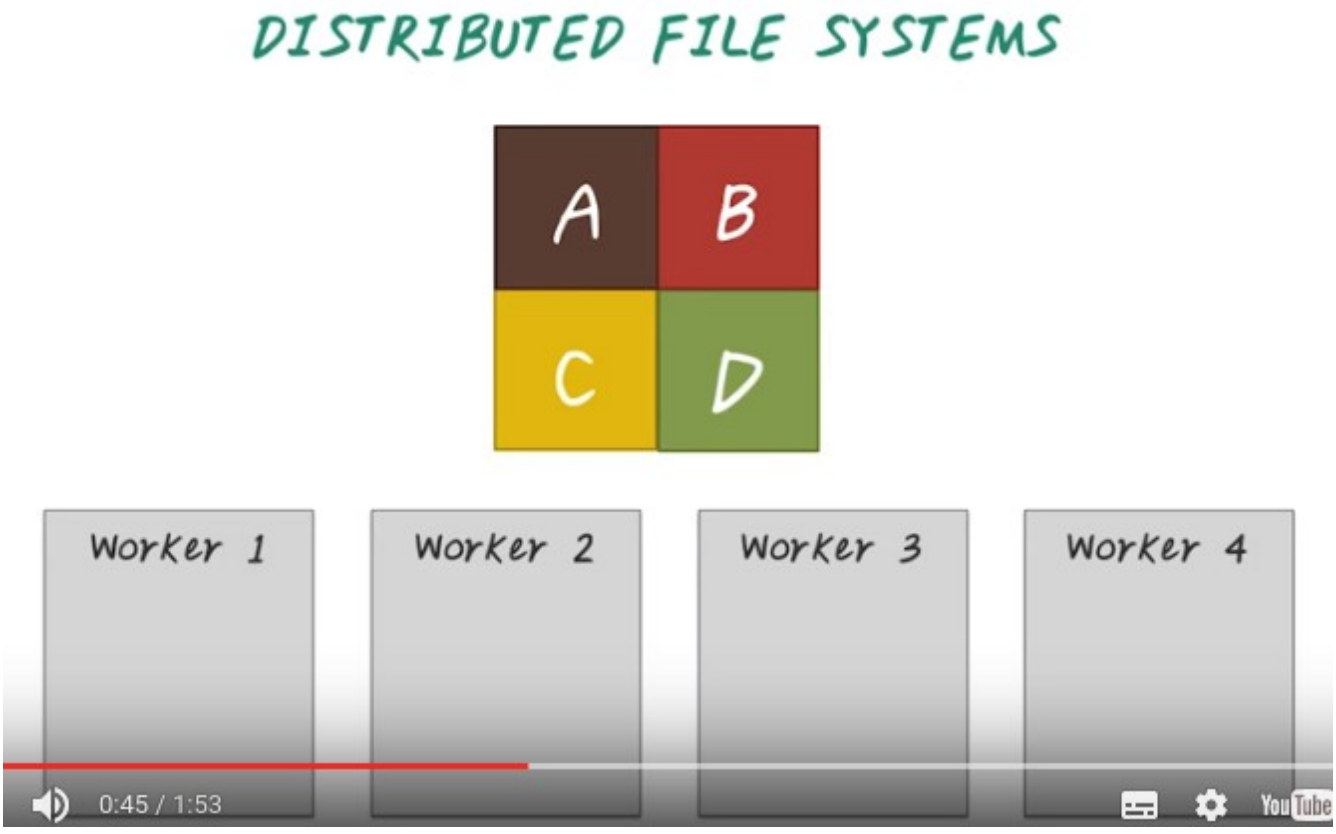


So to summarize, this mapreduce system has three different phases. The map stage, where we performed map function and the pre-aggregation and combination functions. Then we'll have a shuffle stage, all of the intermediate results will be sent to the corresponding reducers. And we have the final reduce stage, where the final output are generated.



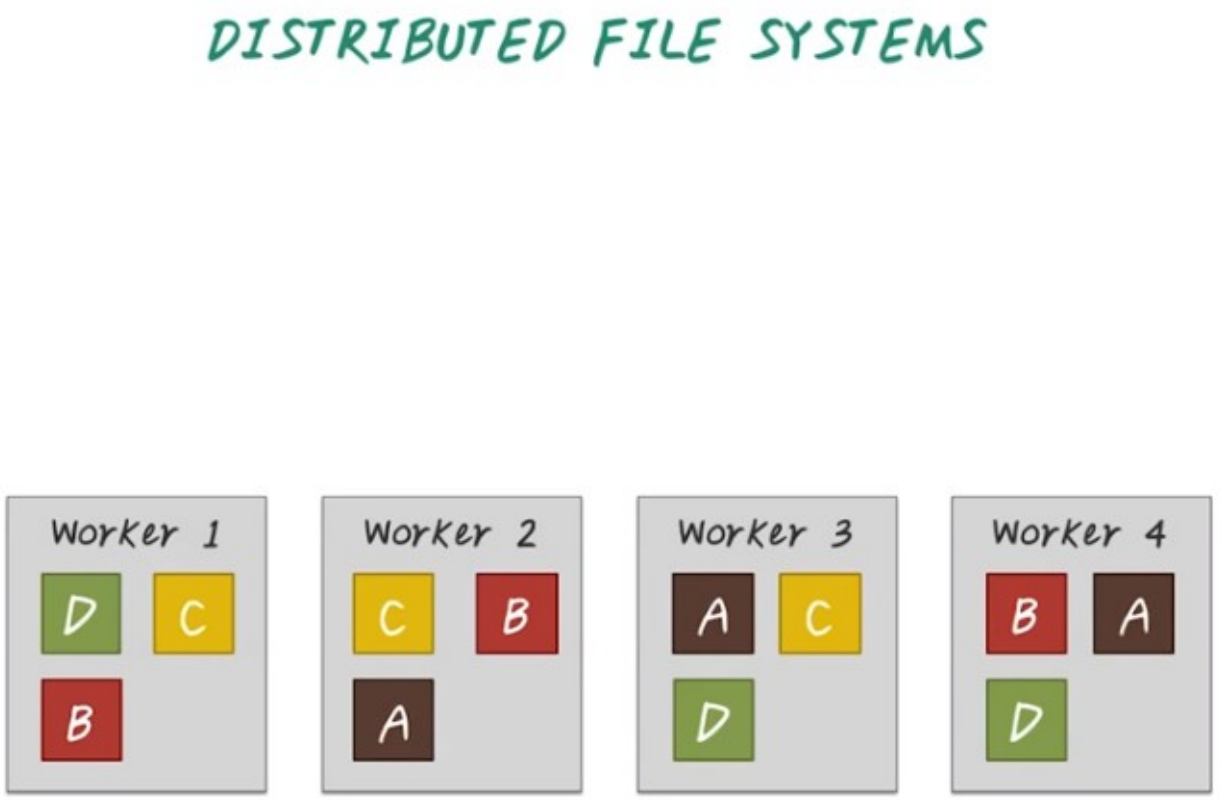
上圖跟前面的圖一樣.

7. So one of the key functionality that MapReduce or Hadoop system provide is fault tolerance. On a large cluster environment, many things can fail at any given times. So, failure recovery is very costly, and often times, data scientists do not want to deal with failure recovery when they implement their algorithms. They want to assume the system will always work. It's really on the shoulder of the system to take care of all the potential failures that can happen in this distributed environment. For example, this mapper 2 can fail during execution of map reduce program. And this time, the map reduce system will restart mapper 2 then goes through the same workload again, by generating the intermediate result. Then all the intermediate results will be sent to the corresponding reducers and the final output will be generated. Similarly, reducer can also fail. For example, if reducer 2 failed, then Map Reduce system will automatically restart reducer 2 and extract the corresponding intermediate result again back from the mappers and re compute all those reduce functions. But keep in mind, the system of Map Reduce is designed in a way that they want to minimize the re-computation. Ideally, when a component failed, only that component should be re-computed. For example, when reducer two failed, reducer one should not be impacted at all. So Map Reduce systems is designed in a way that such optimization is taken into place, and all those re-computation, is minimized.



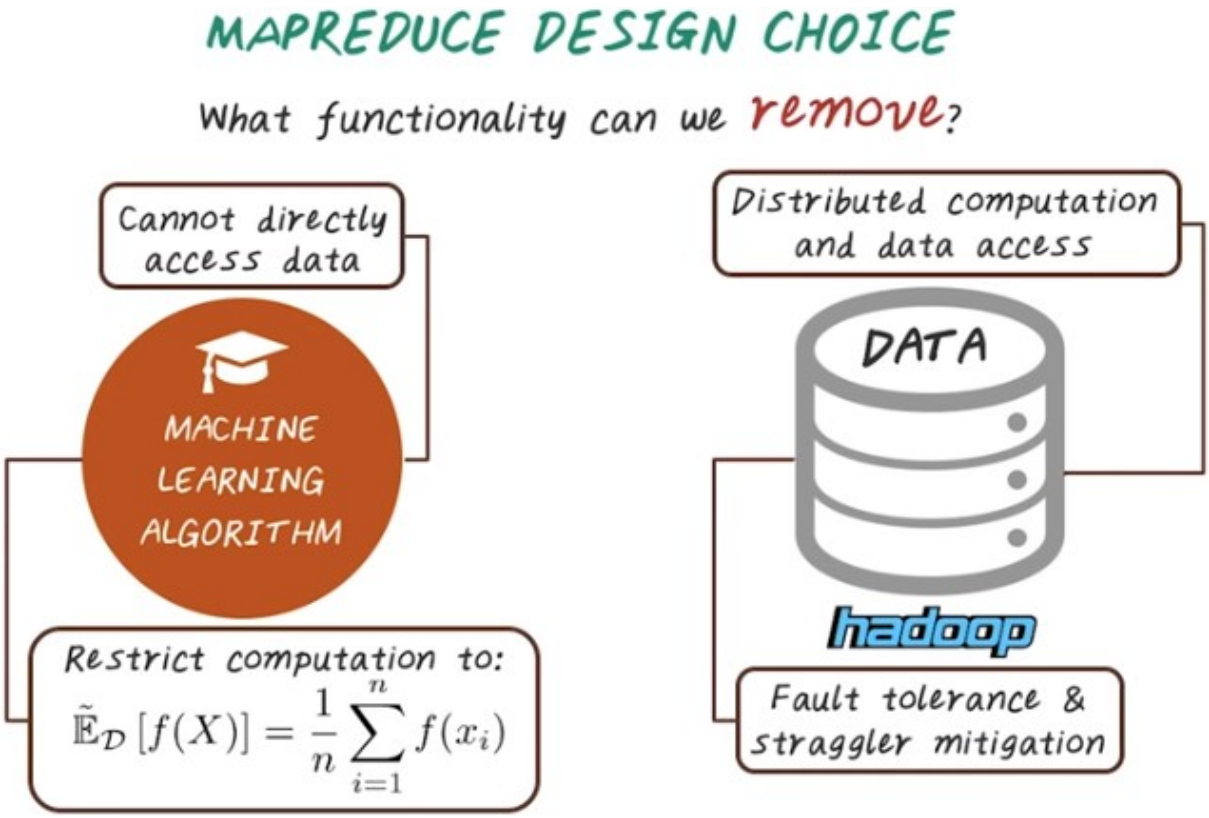
8. The MapReduce system is just part of the Hadoop systems. There is another part that's very crucial as well, that is the distributed file systems. Here, we're going to illustrate the ideas behind the Hadoop Distributed File System or HDFS. Given a large file, it's impossible and impractical to store the whole file on a single machine. Oftentimes, we split this large file into different partition, for example, here we have four partition, A, B, C, D. Then we store those different partition on different machines where

we call workers.

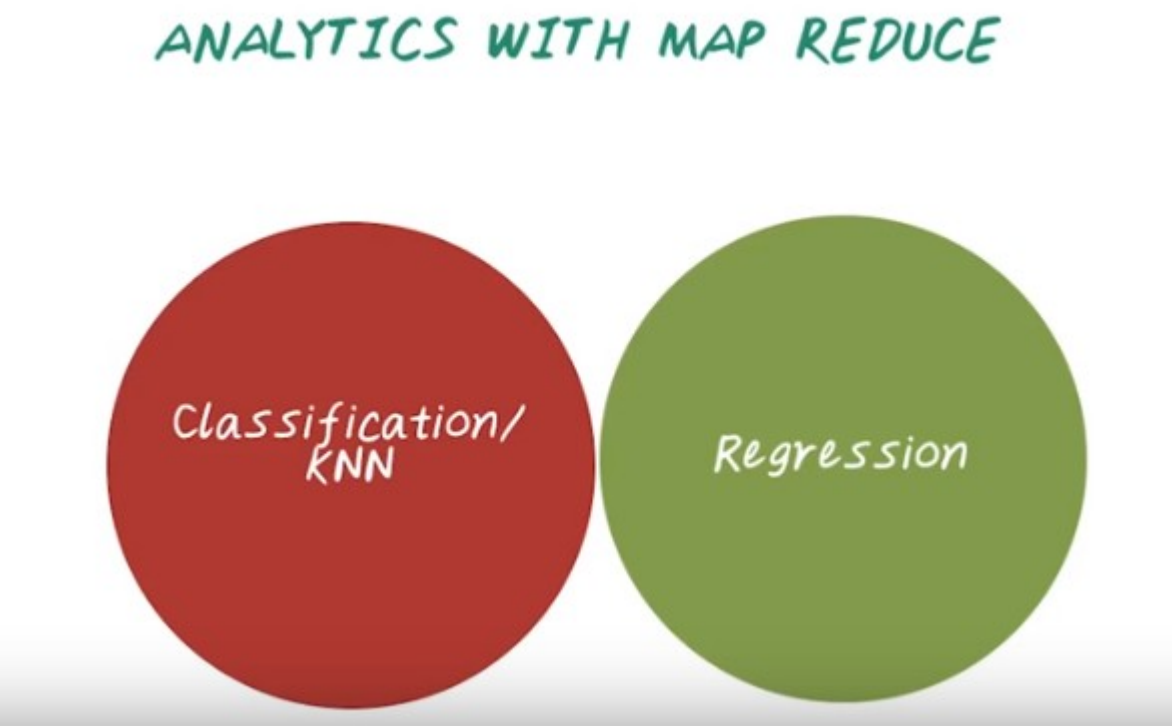


Here we have four workers, and the other thing we do is, we store multiple copy of the same partition on different workers. For example, for partition A, we store it on three workers, on 2, 3, and 4. Same for B, C, and D. So this way, we distribute a large file across multiple machines. In order to access these large files, we can retrieve those different partitions many different ways. There are two benefits for doing this. One is now we can access the multiple partitions in the single file concurrently. So the root speed is actually faster than accessing the single file on a single machine.

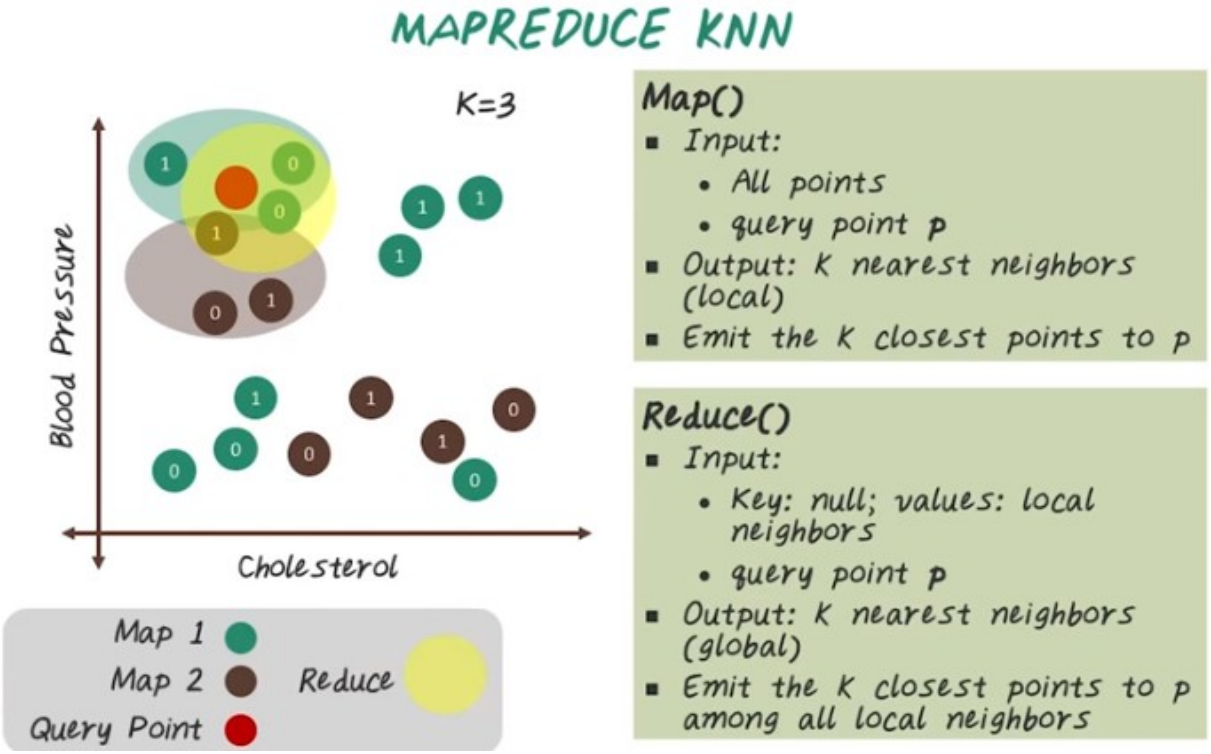
But more importantly, if any of those workers failed, we still can't retrieve a large file. Say we have two worker failed, worker 1 and worker 3. We can still access the entire file by retrieving all of those partitions from worker 2 and 4. So HDFS is the backhand file system to store all the data you want to process using MapReduce program.



9. So the key design philosophy behind MapReduce is not to include many functionalities that people may want. The key design decision is, what functionality can we remove so the system is more reliable and more scalable? At the same time, we want to make sure even with the minimum functionality that provided by MapReduce, so we can still enable powerful computational algorithms such as machine learning. So given the machine learning algorithms, there's a lot of limitation to utilize MapReduce. For example, we can't really direct access data. Instead, we have to specify map and reduce function and compute in a very restricted forms this aggregation query. So all those map function are those function applied to the individual data records. And then this aggregation query in this example, which is taking the sum and averaging that, is the reduce function. So this seems to be a very restricted computation paradigm, but surprisingly, many of the machine learning algorithms can still be supported with MapReduce. On the other hand, hadoop systems will provide distributed file systems and distributed computation. And more importantly they will ensure fault tolerance and straggler(失群之鸟) mitigation(缓和). So here, straggler mitigation is really a extension of fault tolerance. For example, if we have one mapper that runs very slowly, we'll correctively start the same mapper on a different machine. Then leave both mapper runs, and whichever finish first will take the result and stop the other one.

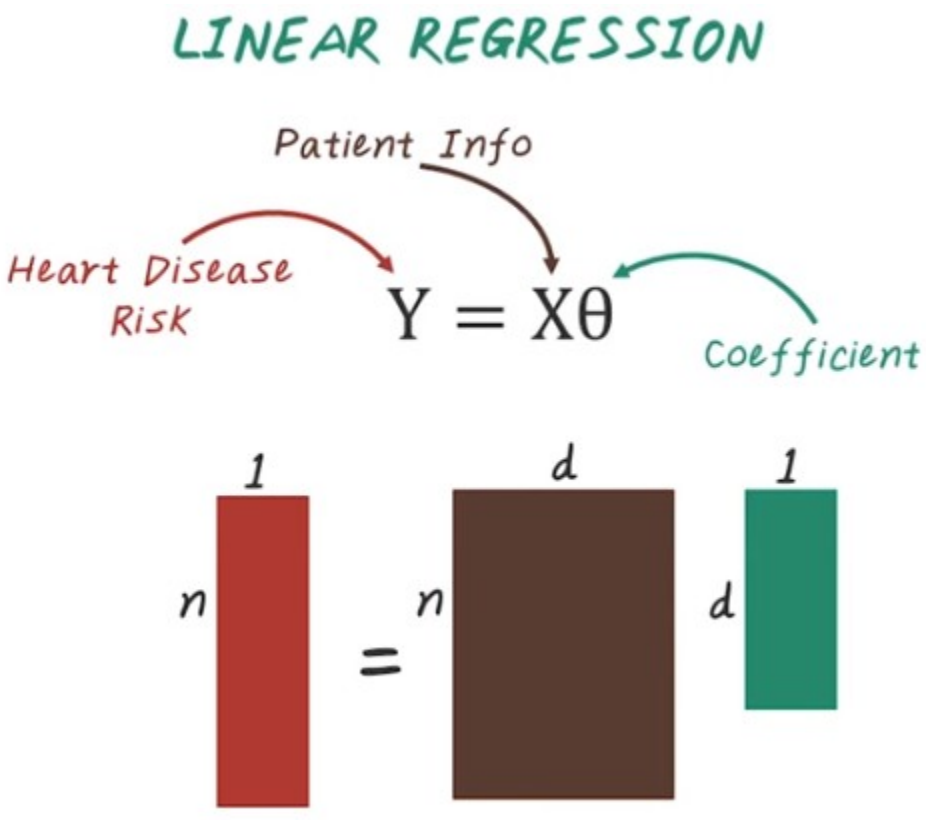


10. Next we'll talk about some analytic example using MapReduce. So in particular, we'll cover a k-nearest neighbor classifier and linear regression.



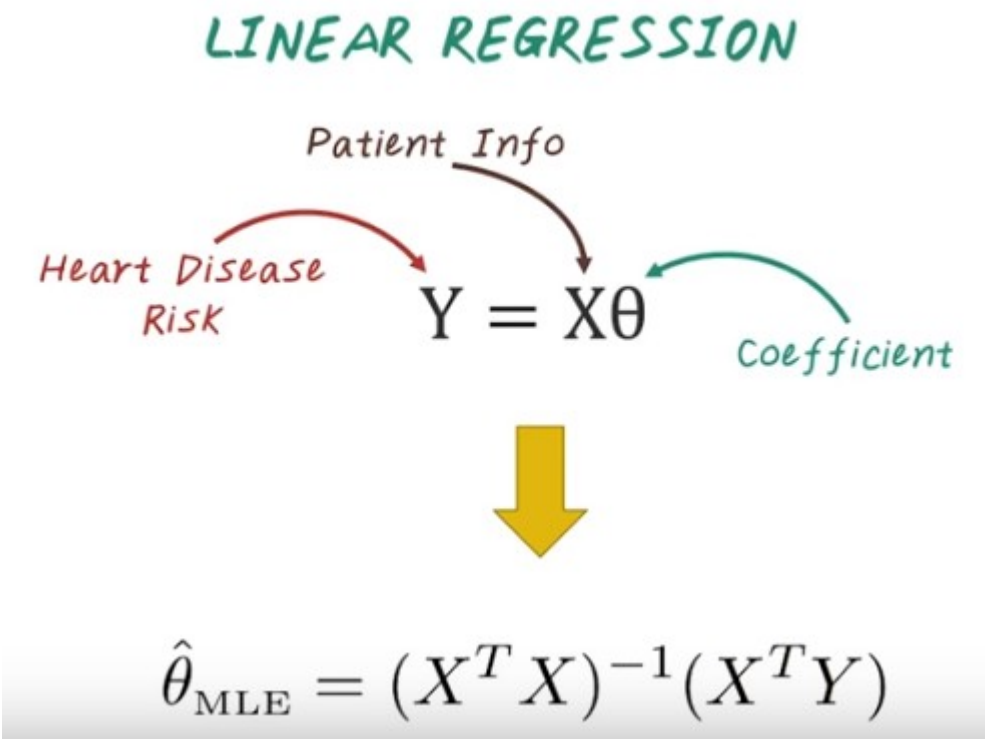
11. Now let's talk about how to use MapReduce to write a K nearest neighbor classifier. Given a set of

patients, say we want to predict whether a given patient will have a heart failure or not, by finding a similar patient to this query patient. So here we have a set of patients, and we plot them in this two-dimensional space. X axis is the cholesterol level, and Y axis the blood pressure. And every black point over here indicate a patient, and the value on those points are whether they had heart failure or not (1 means have, 0 means does not have). For example, a patient over here as heart failure, and a patient over here does not. And the goal is, given a query patient, for example, this patient over here. We want to find the nearest neighbors, then do the majority vote and to predict whether this patient will have heart failure or not based on those similar patients. In the map reduce setting, all those patients will be split into multiple partition. In this case we have a two different partition and this green partition for mapper one is brown partition for mapper two. And this red point over here is our query data point. And we want to find three nearest neighbor. Now we need to specify the map and reduce functions. So the input to the map function are all those data points and this particular query point. To output are K nearest neighbors for each partition, and the algorithm is quite simple. For each partition we'll go through all those data points, and you need the K closest point. For example, in these three points over here are the local nearest neighbor in partition one, and this three points over here are the local nearest neighbor in partition two. And those will be the intermediate results sending to the reduced phase. So, in the reduced phase we need the local nearest neighbors and the query point, and the output is a final global nearest neighbor, and the algorithm is almost the same as the map function, so reduced function will go through all those local nearest neighbor to identify the global nearest neighbor, which are the three points over here.

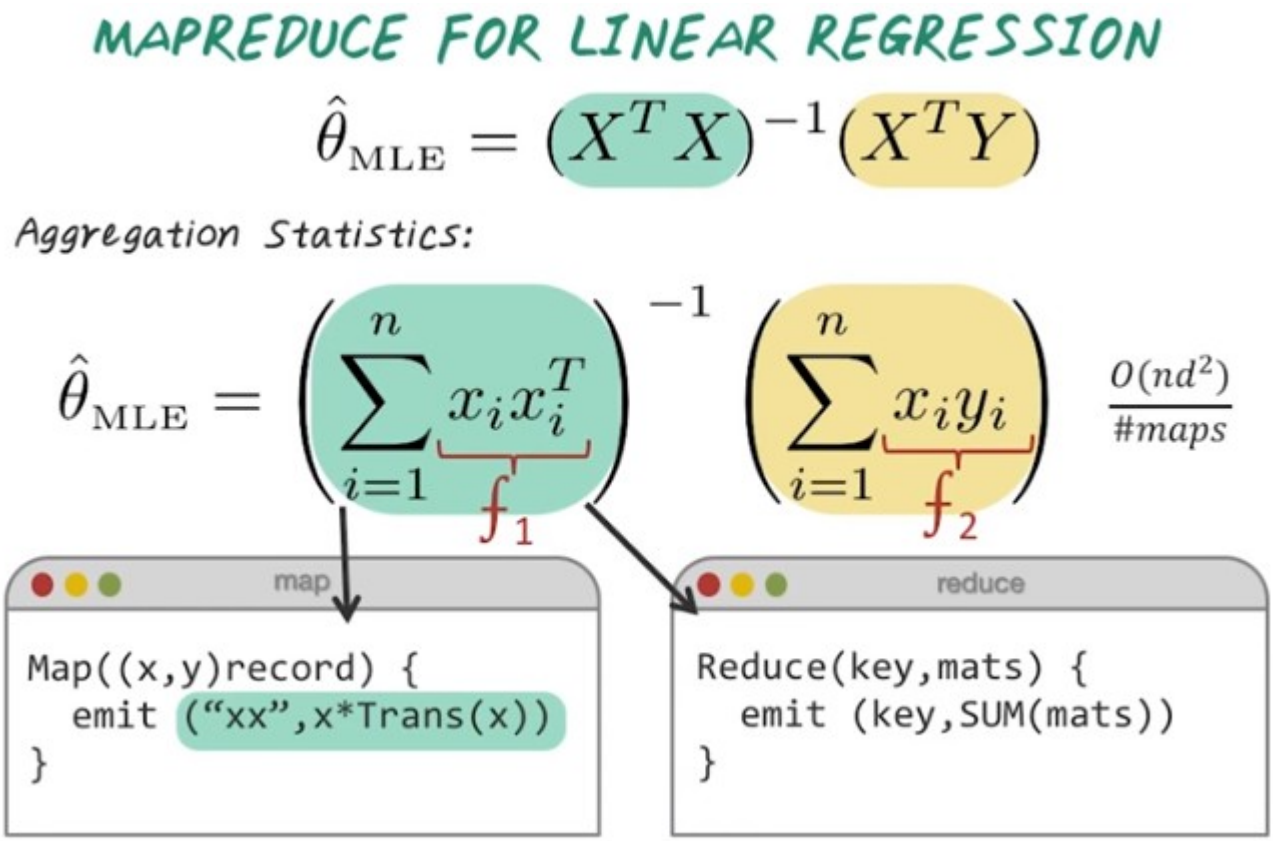


12. Now let's talk about how to use to implement linear regression. For example, we want to view the linear regression that map patients informations to heart disease risk. And we want to find out the coefficients associated with all those patient features. So in this particular case, we have input feature matrix which is n*d and n is number of patients, d is number of features and the output target variable

which is n by 1. Every row here is the heart disease risk associated with that individual. Finally, we have a d by 1 vector and every element here tells us the coefficients in their linear regression model for that corresponding features. For example, the first features may be the coefficient for age, the second feature may be the coefficient for height, and so on.



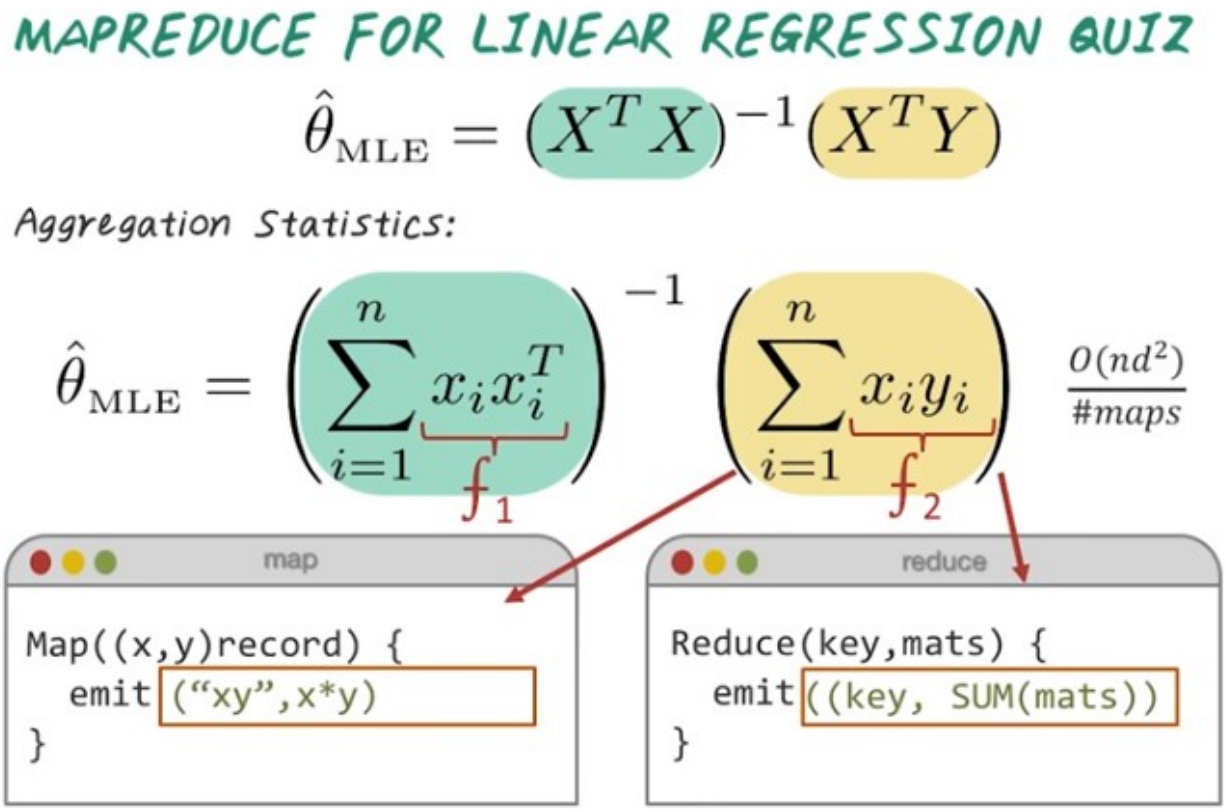
So now let's see how we compute such a model using MapReduce. So from statistic class, we know that there's many way to solve linear regression, and one of the popular way to do that is using this normal equation. That is, the optimal coefficient in maximum likely cosines can be computed by taking X transpose X inverse times X transpose Y. If we have a small data set, this can be easily computed on a single machine using your favorite statistical tools, such as R or MadLab. However, if we have a very large data set with billions of patients, this computation cannot be done on a single machine.



MO:
Map 中的"xx"就變為 Reduce 中的 key,
Map 中的"x*Trans(x)"就變為 Reduce 中的 mats, 然後再 SUM(mats).

And next we'll see how we can use MapReduce to help us to do this. To write this as a MapReduce program, we have to understand this equation a little bit better. There are two steps involved here. One is to compute X transpose X. The other part is to compute X transpose Y. And both can be further decomposed into aggregation statistics. For example, here X transpose X becomes summation from a one to n over xi times xi transpose. And, similarly, x transpose y becomes summation over i from 1 to n, xi times yi. So here, immediately, we can see the patterns we're looking for in term of aggregation statistics. This can easily be mapped to MapReduce computation. For example, this xi times xi transpose becomes the map function, and this xi times yi becomes another map function, f2. For example, to implement the first one as a MapReduce function, here are the pseudo code. For the map function, the input is those x and y pairs and x is the patient feature and y is the heart disease risk. In this particular case, we only need the x feature vector, and we want to compute x times x transpose. In the reduce phase, we take all the output of those xi times xi transpose and compute the sum

13. Now we understand how to decompose this normal equation into two map reduce formulation. We already shown you how to specify this part, x transpose x, as map reduce computation. And now let's do a quiz. Can you specify the pseudo code for map and reduce function for computing x transpose y?

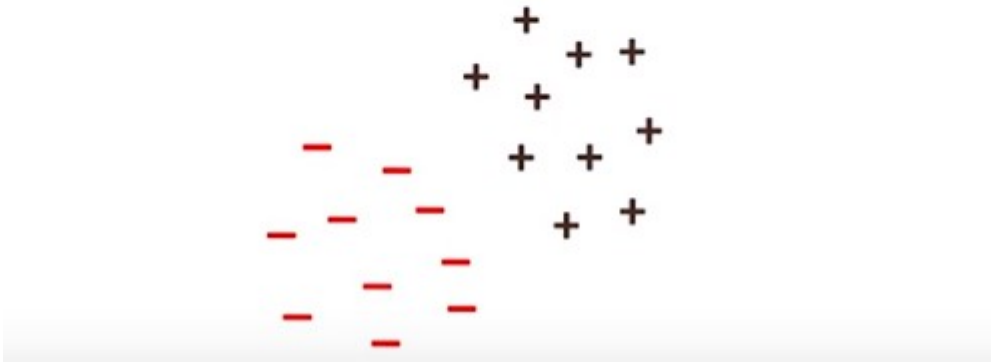


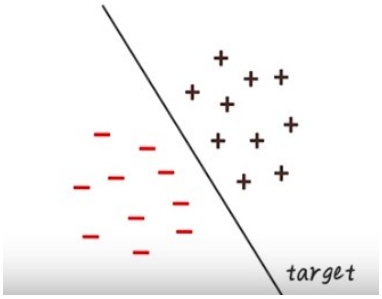
14. So here are the answers. So in the map phase, we're taking our patient record and their heart disease risk and we just simply compute x times y. So in this case, this corresponding to Xi times the heart disease risk, Yi. And the reduce function is the same as before. So we take all those Xi times yi from each patients and just compute the sum.

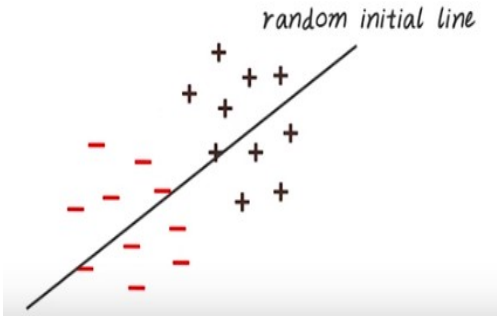
15. So far we've talked about MapReduce and have demonstrate some example using MapReduce for machine learning. And what are the limitation of MapReduce? Let's illustrate that through an example. One of the popular machine learning algorithm is logistic regression. So it's the classification model, and the formulation is very similar to linear regression, but It's not very easy to implement using map reduce. Here's why. For example, we have a set of data points and x are the set of patients who are likely to have heart failure. Yellows are the set of patients who do not have heart failure. And we want to learn a classifier to separate them. And this line is optimal separation and we want to find that.

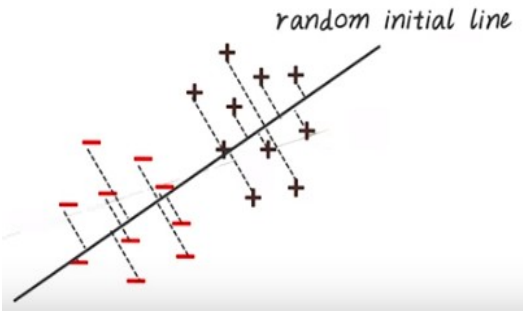
LOGISTIC REGRESSION

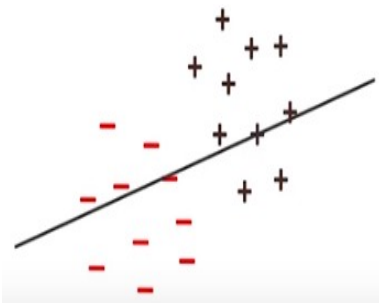
Iterative batch gradient descent

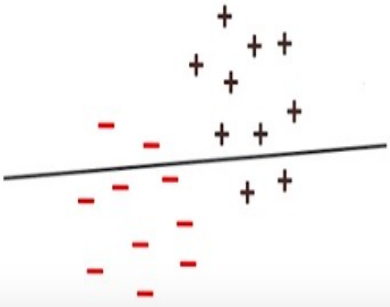


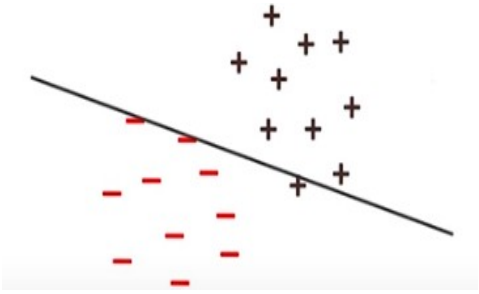


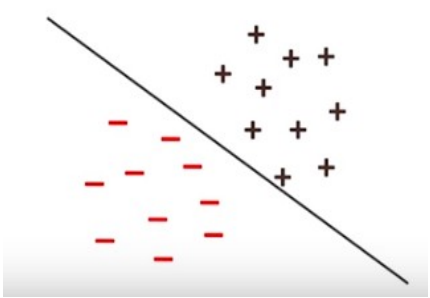


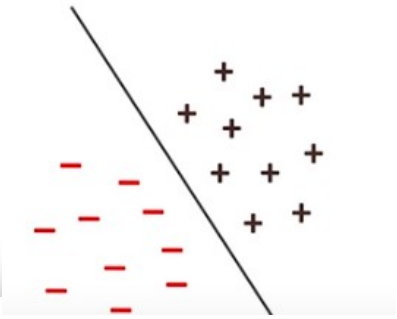




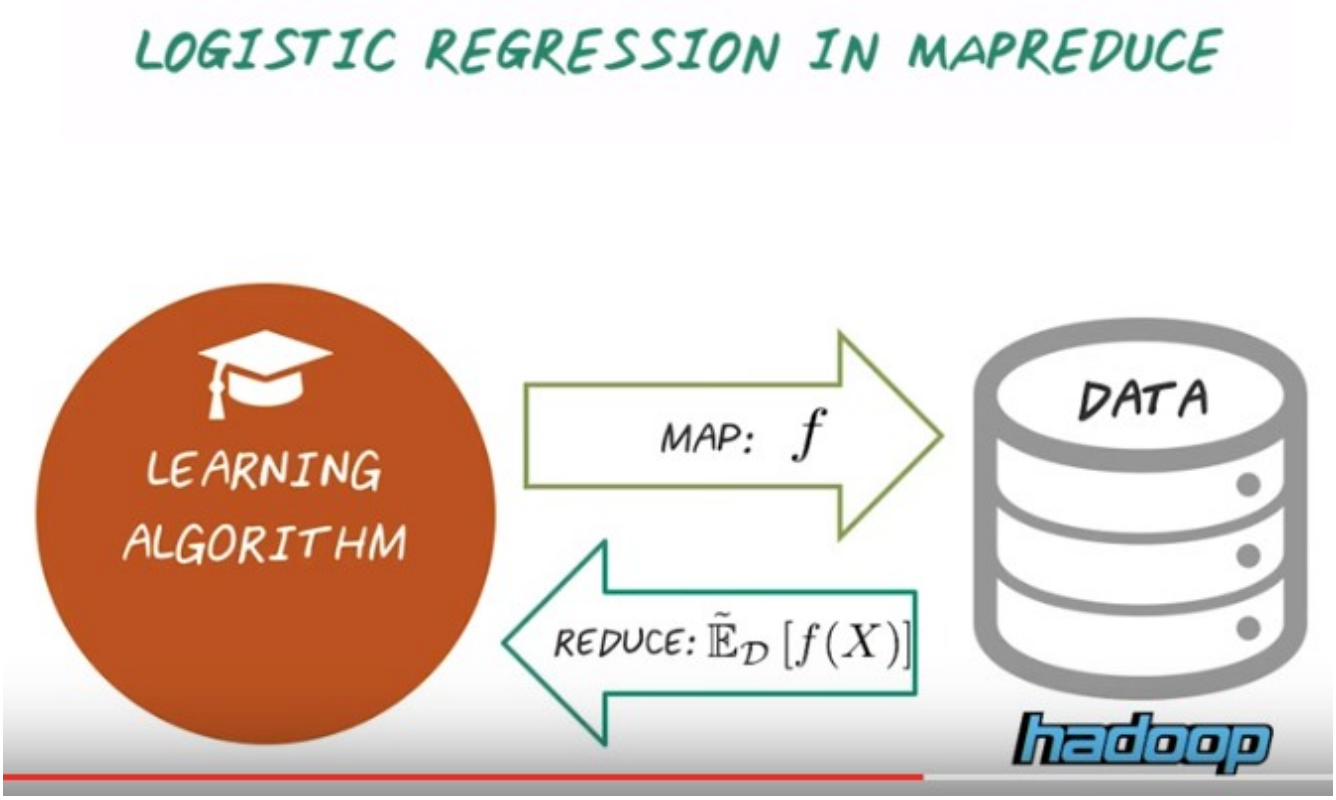




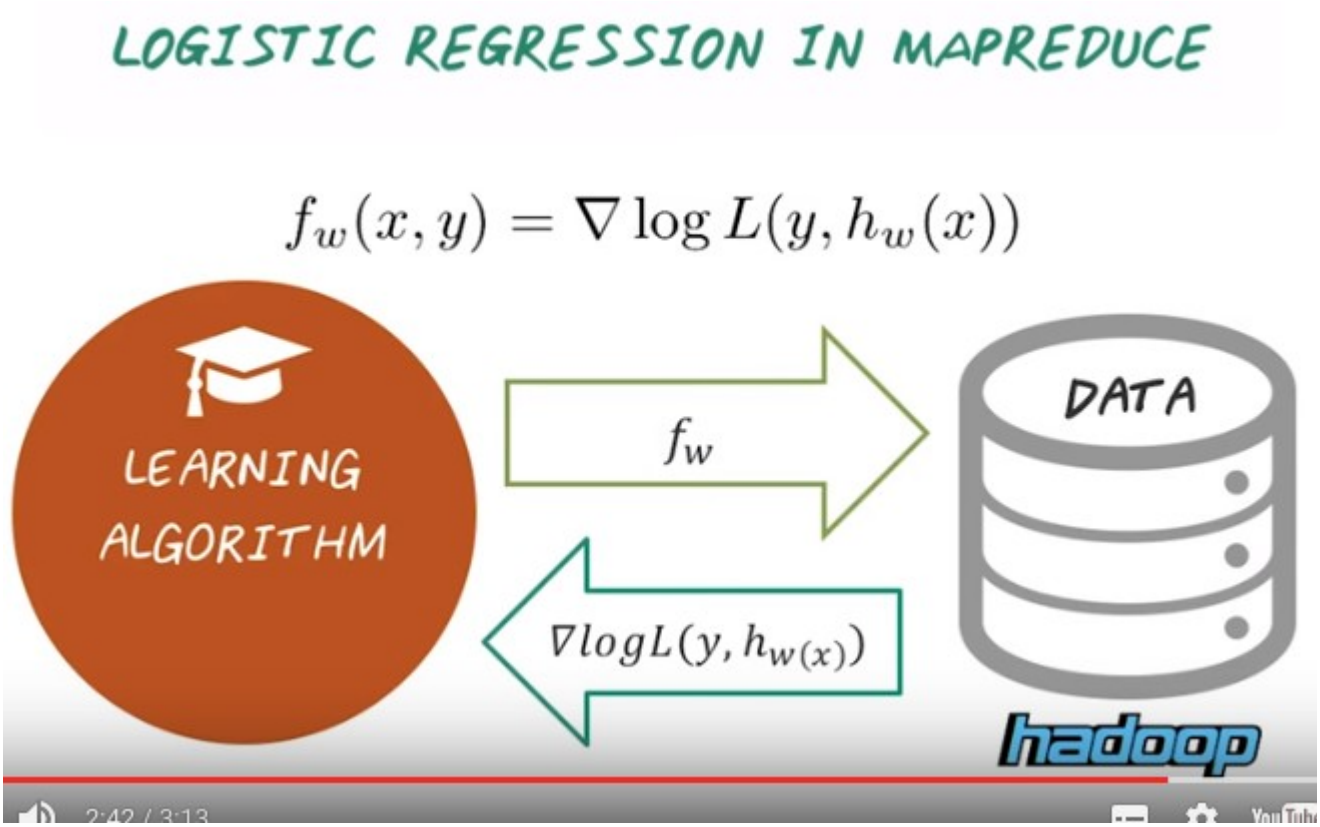




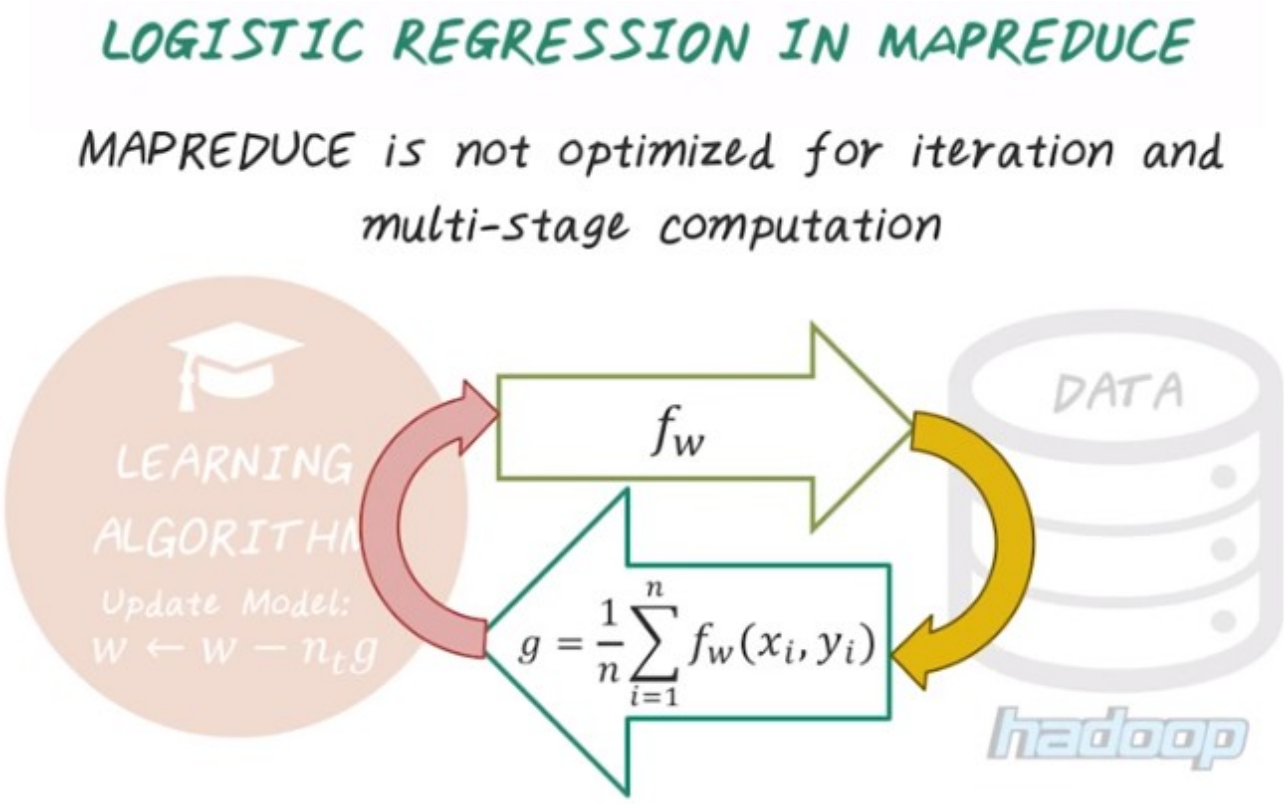
So recall we have talked about gradient descent and stochastic gradient descent in an earlier lecture. And here, if we want to use gradient descent to find optimal classification line, how would that work? So for simple demonstration we have a very clear separation between this two groups of patients. And this line is what we want to learn. So we may start with a random initial points, then start computing the gradient, then update the gradients, then iteratively it will converge to the optimal line. So that's how gradient design works, and that's also how many machine learning algorithms work. They all require this iterative computation. So if we want to map this using Mapreduce paradigms, how would this look like? Next, we'll see how we can map this into a Mapreduce computation. Then you will realize why it's not efficient to use Mapreduce for this type of workload.



In order to implement the machine learning algorithms, we have to formulate this as a set of aggregation statistic computation. So after specified the map functions, then it will be applied to the entire data set and then we aggregate the result and then that will be the reduced function.

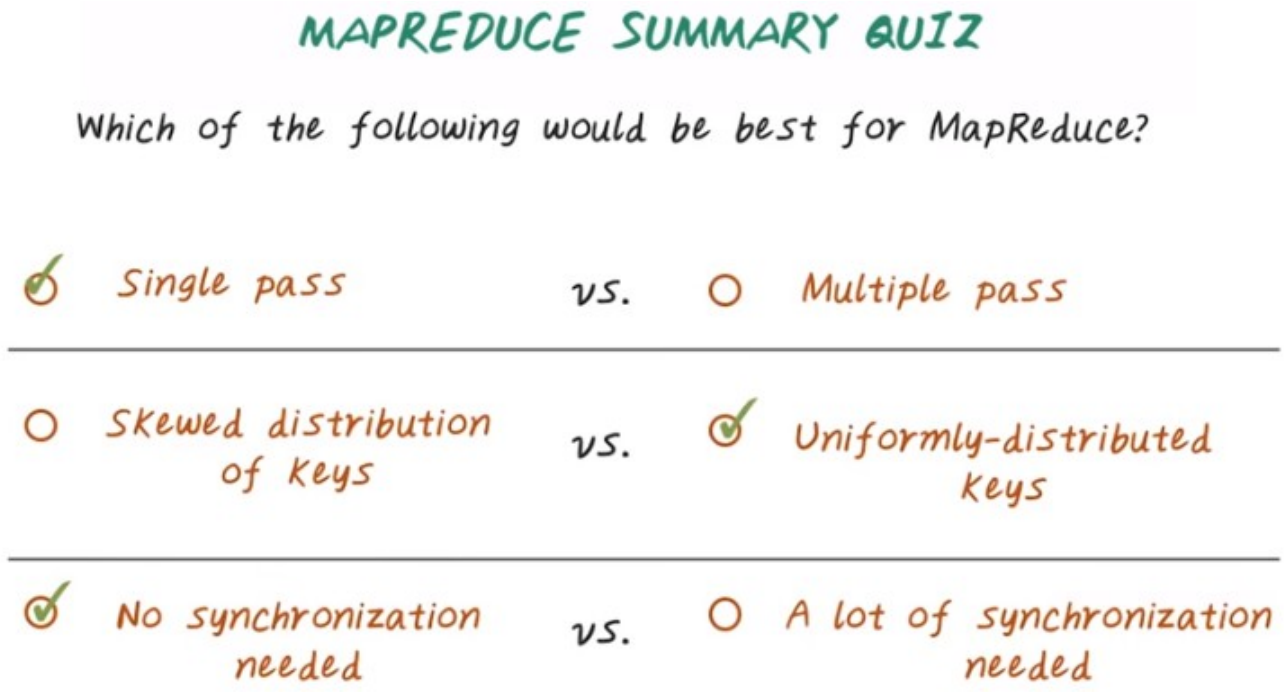


To compute logistic regression using map reduce, we have to specify a map function which corresponding to computing the greeting for a single patient. Then this function will be applied to a large data set with millions of patient. Then, after we apply this math function over all the patients, then we aggregate them together. That give us the gradient. Then we update the parameters of the logistic regression using gradient descent and iterate.



To use an MapReduce to compute logistic regression requires this iterative process. Every iteration requires loading the data two times, one for compute the map functions, one for compute the re use function. And we have to do this iteration many, many times, so it's not efficient. So in summary, MapReduce is not optimized for iteration or this multistage computation.

16. Now let's do a quiz about MapReduce. So which of the following would be the best for a MapReduce job? Single pass over the data versus multiple pass over the data. The key distributions are skewed versus uniformly-distributed keys. So here what I mean by that is, for example, if the age is the key, and age is concentrated only on the age range of twenty to thirty, then it's skewed distribution. While if the age is uniformly distributed from one to eighty, then it's uniform distributed keys. Third, no synchronization needed between different part of computation, versus a lot of synchronization is required.



17. The answer is MapReduce is best for single pass, such as computing histograms. But it's not good for multi pass, for example, computing iterative optimization algorithms, such as logistic regression. So, MapReduce is good for uniformly distributed keys. And also good for Skewed distribution of Keys. If we have a Skewed distribution of Keys, then only one reducer has to do all the jobs. As opposed to Uniformly-distributed across all reducers. Finally, map reduce is good for no synchronization is required (意思是 MapReduce 沒多少 synchronization). It's not so good when a lot of synchronization is required. In fact MapReduce has very little synchronization. The only synchronization in the MapReduce job, is between map and reduce phase. So during map phase, everything is done in parallel independently. And everything inside the reduce phase, is done independently in parallel. And the only synchronization is between map and reduce phase.