

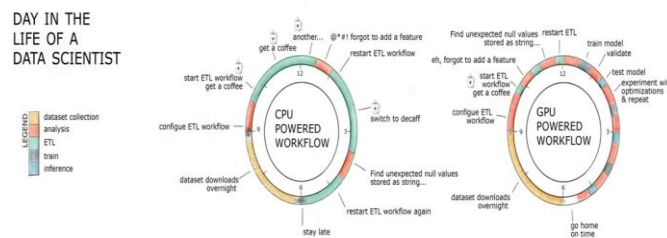
**Applied Accelerated Artificial Intelligence**  
**Prof. Bharatkumar Sharma**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Palakkad**

**Lecture - 45**  
**Accelerated Data Analytics Part 2**

(Refer Slide Time: 00:19)



## DAY IN THE LIFE OF A DATA SCIENTIST



So, let us look at the day in the life of a data scientist, traditionally versus the GPU. So, if you have a CPU powered workflow how will it look like right? So, you start with the data set download that is your first thing; obviously, it takes a long time for you to download the data set you configure the ETL. ETL stands for extract transform and load workflow. So, generally your data would be in a format, which is not very which needs to be transformed. So, it needs to be extracted, transformed and then again loaded.

So, you start the ETL workflow, but it takes a lot of time. And you can see a most of the time why the data scientists has spent just waiting for the things to be done. And then suddenly you realize that in your ETL workflow, you forgot to add a feature, then you again start the ETL workflow, again you wait for it. Then after your ETL workflow ends you suddenly realize that there is was unexpected null values, which are coming; then you again restart the workflow here.

And by the end of it you see how much time is actually spent in doing execution. So, the red part is the one where the actual action is happening and rest all part is just waiting.

You can see different legends, some for training, some for inferencing, some for analysis, and some for data set collection. Now, what we want to do is, we want to move from this left hand side of the diagram to the right hand side.

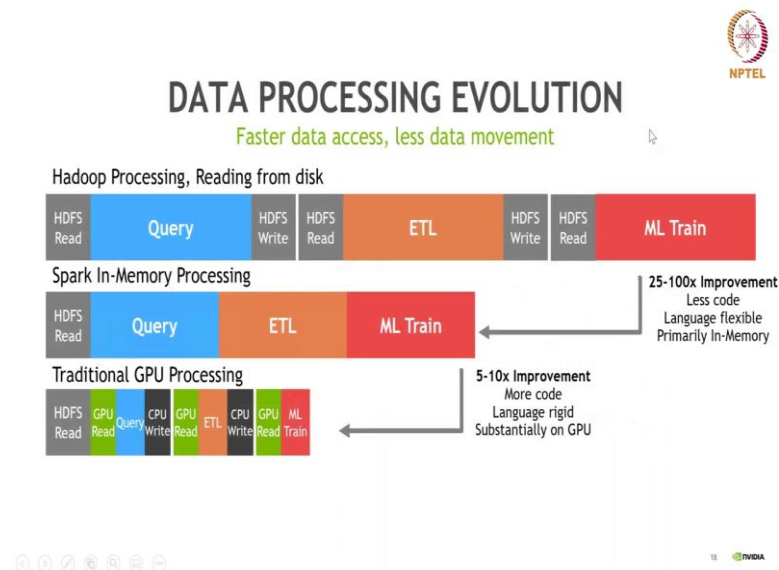
So, your data set download obviously, cannot be accelerated. You move towards starting your ETL workflow and you can see that it does not take a lot of time; to do the extract transform load. Then you realize that you have missed the feature, you again started, you find the unexpected value you restart the ETL. You train the model, then you validate the model, test the model experiment with it and optimize it and repeat it.

So, and you basically even can go home on time. So, the idea what we are trying to say here is that our time should not be spent on the data scientist are one of the most critical resources in an organization. And if the time of a data scientist you can think of something like you know hospital you would realize that one of the most critical resource in a hospital whose time needs to be save the most and that is why most of the healthcare workflow are around this person is the radiologist.

Radiologist is the one who looks at all of your images and tags them with the observation that they have, after collection of the data. And if you look at the whole healthcare flow, it revolves around making sure that the time of this radiologist is saved. The same logic kind of applies even for the enterprise segment also. One of the most critical resource that you can have for the enterprise segment of is the data scientist; because data scientists are most critical resources and they are hard to find.

And that is why the whole software stack should revolve around making sure that the data scientist time is more optimally used. And we want to move from this left hand side to the right hand side diagram with help of rapids.

(Refer Slide Time: 03:51)



But, you need to also understand to certain extent the history of how this evolution has happened with respect to data processing. All of us would have read about the big bang of the big data regime. Well one of the most popular framework Hadoop was built over the map reduce phenomenon right. So, it was primarily this cycle which existed. So, you have your data, it is being spread out across different nodes different storage and you have to do Hadoop data file system read; then you query the system.

The query basically meant that you are going to transfer they compute to where the data resides. So, the data is spread out and you basically send your computation towards where data is and then finally, whatever is the result gets accumulated. And then again you have to do the HDFS write and then you keep on doing HDFS read, you first query it and then you do your extract transform load; then you again do a write, you do a read and then you actually start your ML training, which is the machine learning training.

Now, and for a user it kind of change the industry for a large extent, because it solve one of the problems of distributed computing; because the data set was increasing and you used to get tons of data every day and there was no other way to do it. And that is why this map reduced framework kind of solved a lot of challenges, but there was one particular problem with this which needs to be sorted out.

And that challenge was this reads and writes, which were happening to the file system. Anyone of us would be able to tell that reading and writing from disk is the most costly

operation which can be done; because that is the one which takes majority of the time compute is kind of free, but your memory is kind of or the file reads are the most costly thing which can happen.

To solve this problem, there was innovation which was spark. Spark is basically solve the problem by keeping all of it in memory. So, most of the computation whatever we want to do comes from the data which is residing in memory. So, you can see here for the first time there was a HDFS read, but after that whether you are doing the query extract transform load or ML train everything is happening in memory.

So, your overall pipeline is happening in memory. Because of this innovation there was almost 100 x speed up as compared to the to doing the file reads and writes. There were was also necessity to there was also an improvement in terms of writing lesser code and it was one of the main innovations which resulted into a lot of improvement.

The next set of improvement came was that now we have solved the problem of reading it from file, but my query extract transform load and ML train takes a long amount of time, which is the compute part of it also. So, in order to solve that problem as GPUs were becoming more and more popular deep learning also was becoming more popular there were innovations terms in terms of making sure the query ETL and ML train pipeline is also put on the GPU.

So, you can see here rather than doing a normal query ETL or ML train. So, you do HDFS read, which means you read from the file system, but then after they transfer that data to the GPU using the GPU read functionality. Then the query takes a very short amount of time, because now you can isolate it using different cores on the GPU.

Later on you have to again do the transfer to the CPU not to the file system, but to the CPU; because of doing because there were certain path pieces which are missing. So, again you need to go back to the CPU, CPU will do certain amount of work. Again it was given to the GPU for the extract transform load, again write it to the CPU ram, read it from GPU and finish the ML train.

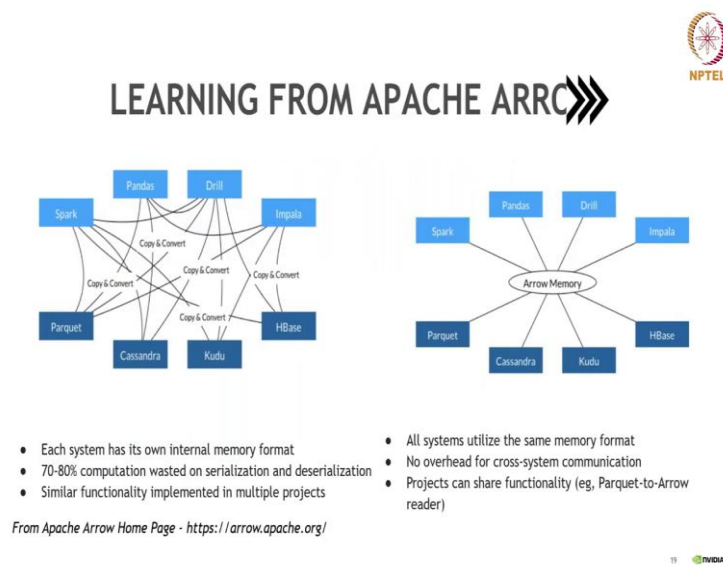
Now, with the innovation of the GPU processing you could result into reducing the time for calculation of your query, extract transform load and ML training, but there was still a problem which was reading and writing to the CPU, reading to GPU and writing to the

CPU. Hence, the improvement which could have been much much manifolds more basically was only 5 to 10 x.

But then because I had to read and write into memory again and again. I had to increase my source code and it became to language rigid, because I had to use certain specific language to read or write code on the GPU and a large portion of it was then doing this transfers rather than actually doing the computation.

So, yes there was a 5 to 10 x speedup in this overall pipeline, but there were certain additional overheads also which came. To solve that problem at that time there were not all the right set of pieces and that is why we had to do all of this then came a project, which is called as apache arrow. Apache arrow was a innovation, which kind of solved one big problem which was existing in this data science space.

(Refer Slide Time: 09:27)



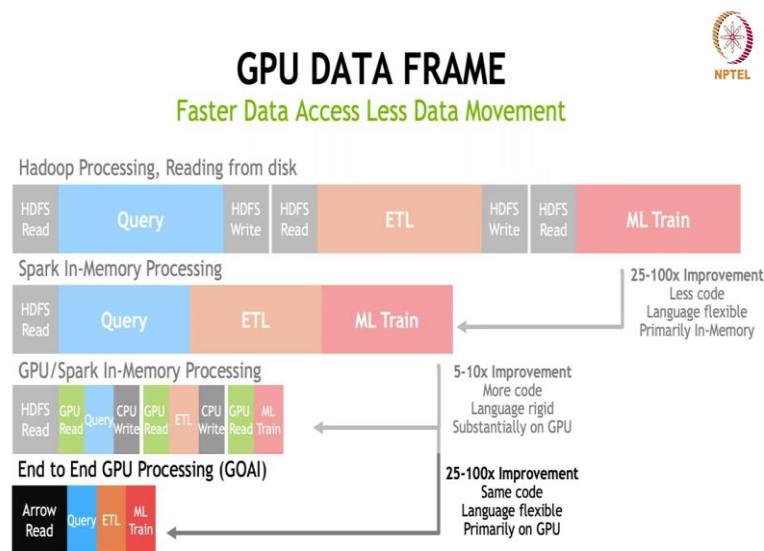
There are so many frameworks out there and each and every framework has its own way of talking or reading or writing or interfaces. Apache spark, pandas, impala, Hbase cassandra all of them basically each system has its own internal memory format. And generally one framework is not enough in a enterprise segment you need to provide data from one to the other.

And hence, whenever you had to do that interface between different frameworks you have to do serialization and deserialization. So, you had to convert like from spark if I

were to send to pandas or if I were to send to drill, I will first do this serialization and again when it comes to drill I will do deserialization. So, 70 to 80 percent of the time was wasted in this conversion from one framework to the other.

Apache arrow kind of solve this problem, because it exposed all of them with the same memory format. There was no need for any kind of cross system communication and the project, basically can share the functionality as well. And with this innovation all of the missing pieces to make the overall thing run on the GPU were kind of present and that is when the goal that behind in mind came.

(Refer Slide Time: 11:07)

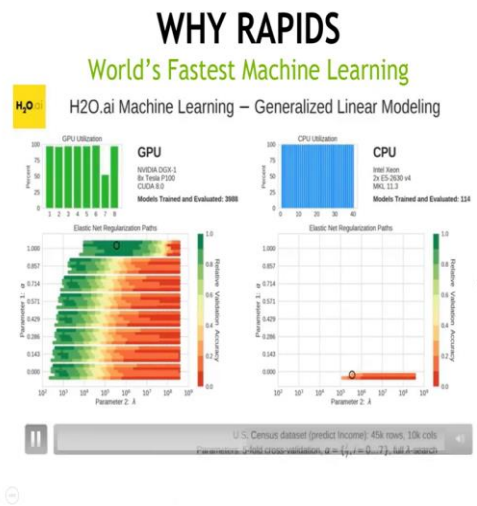


So, instead of doing the HDFS read; now, the whole thing sits over apache arrow. So, what we do is we read it from apache arrow and then after that there is no read or write happening through the through the CPU everything remains on the GPU and once you have done or everything is in apache arrow, then you do your query extract transform load and ML train all of them on the GPU itself.

Now, that is where a lot of innovations came into the picture and you can see here how many folds increase. This is the cycle of the data science. First moving from file system to in memory system, giving you 100 x speed up; further moving towards using GPU giving your 10 x speedup, which is like 1000 x speedup from the top.

Then again more 100x speed up by making use of apache arrow, which is almost like 10000 x speedup by the innovations with respect to changing the pipeline and using the right set of tools and technology and that is what is the necessity of the arc.

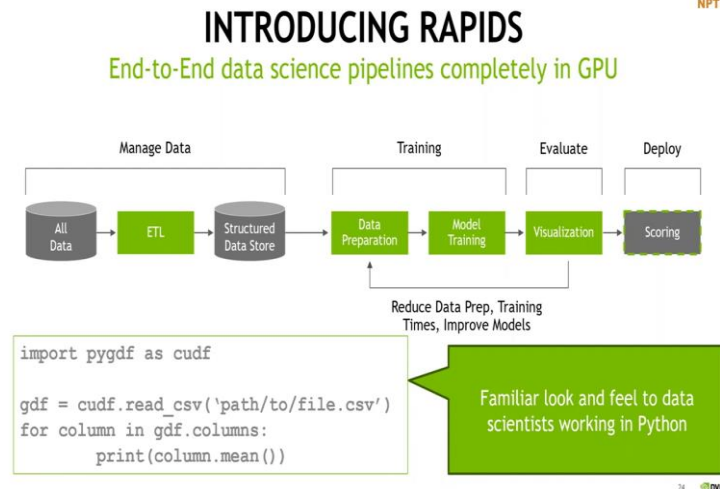
(Refer Slide Time: 12:23)



So, as I said why RAPIDS is kind of visible you can see here one of the example from H2O.AI machine learning framework and you can see the CPU and the GPU which is kind of listed. These are kind of old GPUs and CPUs, both of them are old. They are 3 generation old. But you can see here on the GPU side, the models trains and evaluated, because you need to try out different combinations, different type of parameterization.

The GPU is already doing the validation for all the different types of parameters, but the CPU is yet to even start giving out results. And this is for the U. S. census data which has 45 k rows, 10 k columns and for validation we are using fivefold cross validation. Say in fact, we have got one of the results here. Yes, here now is when it has actually started doing something, while the GPU is already finished all the parameter.

(Refer Slide Time: 13:55)

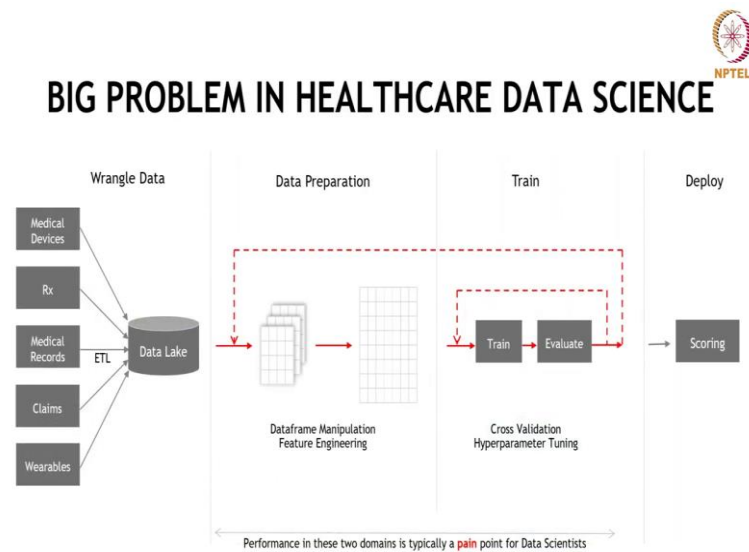


So, the big problem in data science again it comes to the part of what is the cycle in this data science community. The first part is managing the data itself. So, you have your data, you need to do the extract transform and load part of it and you store it in form a structured data store, which is then provided to the training phase, where you again prepare the data you do the model training. You have to evaluate it using some visualization tool and this is kind of a cycle. The cycle is to keep on doing this and then finally, you deploy it also.

So, RAPIDS is basically an end to end data science pipeline, which is completely in GPU. So, it helps to meet your ETL pipeline, the data preparation, model training, and visualization and to certain extent even your deployment ready and accelerated on the GPUs. And one of the best thing which we talked about RAPIDS also is it looks very similar and the feel is very similar to what a data scientist would have traditionally used, when GPUs were not existent and it will look very similar to how the APIs were in the CPU world.



(Refer Slide Time: 15:27)

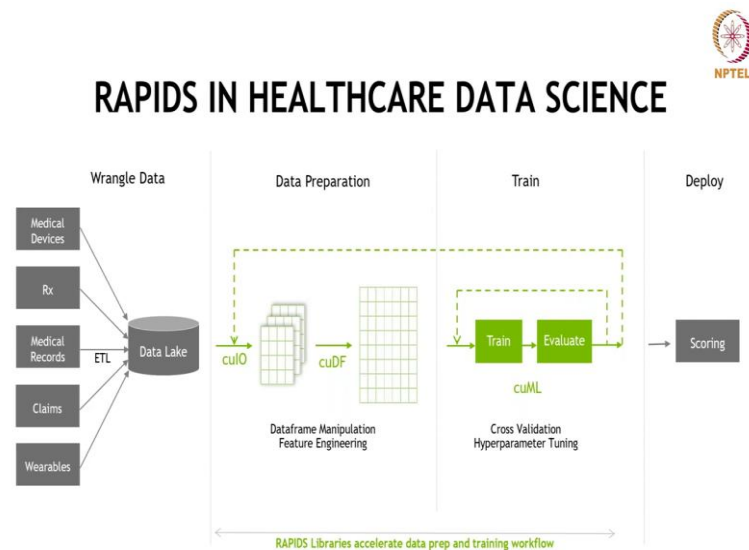


So, this is an example of the healthcare data science where you have data. The data basically comes from different devices. You have different sets of medical records, claims, variables, medical devices, like MRI machine, CT machine and all. You have to convert all of this, you have to extract it transform and then finally, load it into something called as a data lake. Once the data is in the data lake, you need to basically do this you need to prepare the data for training.

So, you would do feature engineering and also you would try to do some in form of data frames you would manipulate the data like, you will maybe consolidate some of the columns, you may average some of the columns. So, there are different sets of feature engineering and manipulation that will be done. Finally, it is provided to your training. So, you will apply different types of models here and evaluate finally, which is the one which gives you right thing, you will also do cross validation and hyper parameter tuning here to see that you are able to get the right thing.

In the process, you might realize that the features that you are using might not be correct or they need to be change and it again goes back to your data preparation stage and it is like a cycle which keeps on happening. And these 2 domains is typically the pain point for the data scientist and finally, yes you have to also deploy it once that cycle is over.

(Refer Slide Time: 17:03)



So, in the RAPIDS world you are going to start seeing now, certain terminologies or certain names which we are going to introduce you. In fact, each one of them that you are seeing here is going to be one lecture that we are going to cover in this series for the data science or data analytics. We have a library or a module inside rapids, which is called as CuIO, which is responsible for taking it from the data lake and then for the data manipulation of feature engineering you are going to use something called as CuDF.

And then for the training and evaluation you will use a module inside RAPIDS called as CuML. The name Cu here comes from the fact that it is isolated on the GPU and all of the libraries on the GPUs or the architecture of the GPU is called as CUDA, which stands for compute unified device architecture. That is why the libraries are tagged or are in the beginning with the name cu which is primarily coming from the CUDA side of it.

So, the RAPIDS library there are different modules which are present inside it and each and every module is responsible for their own task.

(Refer Slide Time: 18:25)



## DATA PREPARATION



So, as I said CuDF is basically responsible for the data preparation. You can see here as an example, there is a patient and the patient is having certain features attached to them date of birth and everything blood type and all. And then there is another column, which is kind of also having the medication there.

So, you will have each and every patient having different sets of medicine that they would have taken over time. Or there are certain observations which have been happening based on the test results which are there. So, what you are trying to do here is to combine all of this into one big data frame like join, we are trying to do basically in the data based community world the name here is join.

So, what you are trying to do is, you are trying to join on the patient ID and also try to do group by and you may also end up doing certain operation during that group by like doing an average over the last 5 readings. And that is what finally you will get a combined data frame right. So, you have the patient here. So, you have combined the part of this particular database and at the same time taken certain observations from here, but also you have done certain averaging based on say the 5 reading.

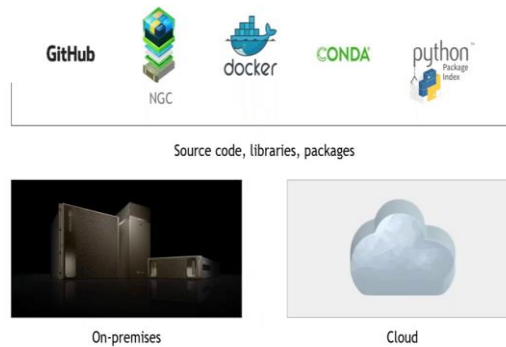
So, this is like a data preparation that needs to be done, before even providing or provided to the machine learning for the training part of it.

(Refer Slide Time: 20:01)



## HOW? DOWNLOAD AND DEPLOY

Source available on Github | Container available on NGC and Dockerhub | PIP available at a later date

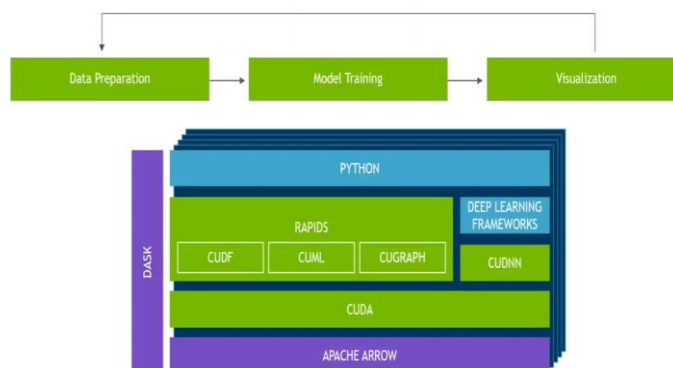


Before even you go there. So, one thing as I told you about RAPIDS is that it is basically available on GitHub, the source code is available on GitHub. The it is present in form of availability via different kinds of platforms. You can install it via Python package like PIP, you can do installation using CONDA, you can also use Docker or you can download it in form of a container from NVIDIA GPU cloud or you can directly take it from GitHub.

(Refer Slide Time: 20:39)

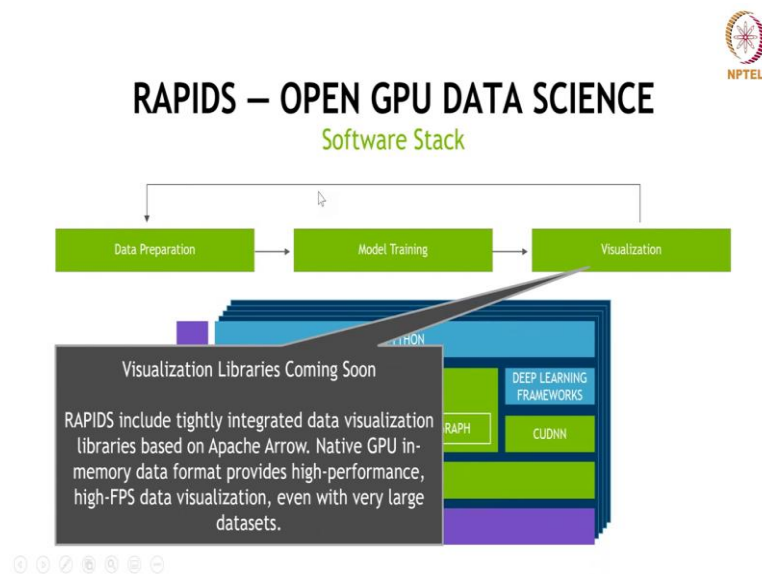


## RAPIDS – OPEN GPU DATA SCIENCE Software Stack



And you can run it on any system including cloud as well as on premise. From a RAPIDS point of view again it is a open GPU data science platform and this is where you see there are different parts of the overall rapid stack. So, the bottom is apache arrow as we talked about and there are different components that will go through briefly and we are going to take certain of these components as one lecture each in the next couple of days of this training.

(Refer Slide Time: 21:11)



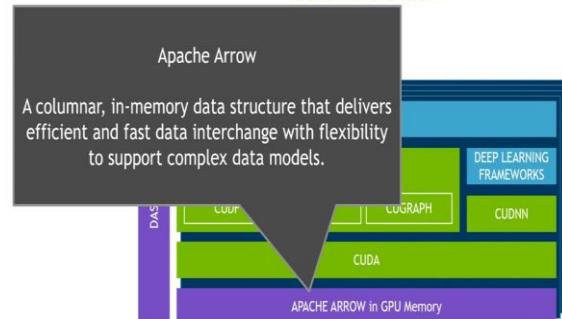
So, the first one here that you are seeing is the visualization library itself. There is a slight correction here that the visualization library is not coming soon, but it already exist. The idea behind the visualization library is to do the analysis in the end right. So, it is going to be based on it is based on apache arrow and it is primarily used for visualizing the data, which is coming out of model training. So, that you are sure that whatever the model has learned is as per the expectation it helps you in the validation part of it.

(Refer Slide Time: 21:47)



## RAPIDS – OPEN GPU DATA SCIENCE

Software Stack



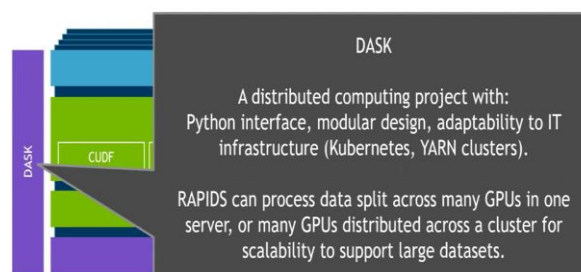
As I said the one of the most critical piece in the overall rapids, which was required was a apache arrow. Apache arrow is basically a columnar in memory data structure. It is one of the most the 2 critical word there is its columnar, which means it is structured data. As I as we had said earlier deep learning is for unstructured data and this is for structured data, it provides in memory support. And it helps in doing data interchange between different frameworks and that is why it is also supports more complex data models.

(Refer Slide Time: 22:25)



## RAPIDS – OPEN GPU DATA SCIENCE

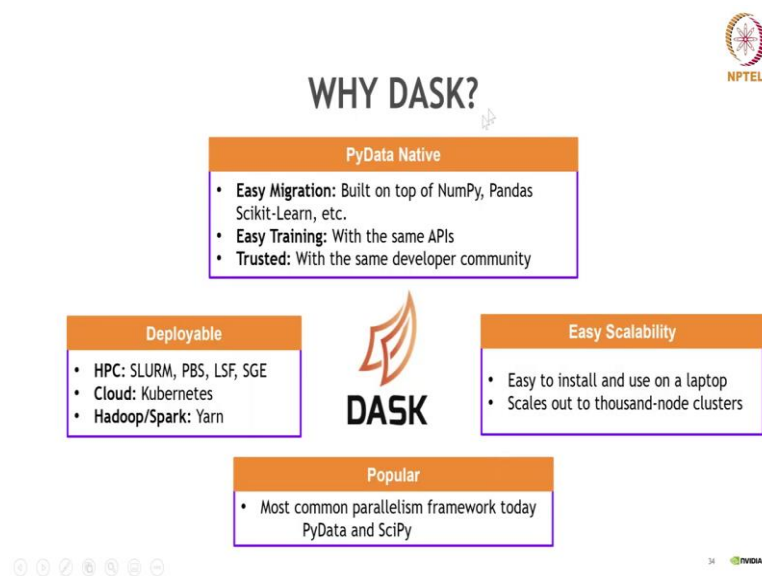
Software Stack



Then we have something called as DASK. DASK is a distributed computing project which was launched some time back. It has Python interface one of the most critical thing is that in this world Python is the most popular language. So, it has a Python interface. It is modular in nature and also the infrastructure it take scales pretty well with different kinds of infrastructure like Kubernetes and also YARN clusters.

So, RAPIDS also makes use of DASK. Dask is not a project by rapids, but it is a open source project and RAPIDS extends it to do the same job on the GPU.

(Refer Slide Time: 23:09)



So, again why DASK? One of the critical thing about DASK is that it is easy to migrate. It is it looks very similar to NumPy. So, it is built on top of NumPy and pandas and sklearn. So, it is very easy; there is its very because it uses almost the same API, you can use it pretty well. And again it is supported by the developer community it is a open source.

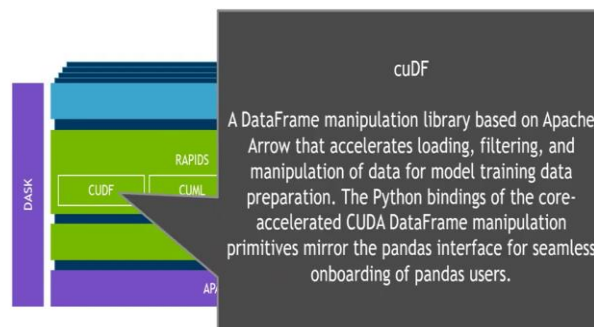
And as I said one of the critical thing about DASK is to scale it across different nodes with using Python APIs. So, it supports all different types of HPC cloud and other kinds of mechanisms to go across nodes. Like if you are a high performance computing user, if you have a high performance computing cluster, you would use schedulers like SLURM, PBS and all. So, it kind of supports the same. If you are in a cloud environment you would use Kubernetes is the most popular framework for data orchestration that we have seen previously.

So, it kind of supports that. It is very easy to install you can install it on your laptop or it can scales to thousands of nodes as well. And as I said since, it is Python based its it is kind of also very easily used, otherwise you had to use other kinds of things like R programming and all which was not so easy for many of the data scientist.

(Refer Slide Time: 24:37)

## RAPIDS – OPEN GPU DATA SCIENCE

Software Stack



So, RAPIDS again let us move ahead with the from the task. Now, we move towards cuDF. cuDF is a data manipulation library, which we just saw sometime back with an example and it has it is actually a C++ library, but having Python bindings and it is used for manipulation of your structured data.



(Refer Slide Time: 24:57)



## ETL - THE BACKBONE OF DATA SCIENCE

cuDF is...

```
In [2]: #Read in the data. Notice how it decompresses as it reads the data into memory.
gdf = cudf.read_csv('https://data.s3.amazonaws.com')

In [3]: #Taking a look at the data. We use "to_pandas()" to get the pretty printing.
gdf.head(5).to_pandas()

Out[3]:
   User ID  Product ID  Gender  Age  Occupation  City  Category  Stay_In_Current_City_Years  Marital_Status  Product_Cat
0  1000001  P0008042     F      17      A        2        2          0          0          3
1  1000001  P0024842     F      17      A        2        2          0          0          1
2  1000001  P0007842     F      17      A        2        2          0          0          12
3  1000001  P0008542     F      17      A        2        2          0          0          12
4  1000002  P0028542     M     35+     C        4+        0          0          9

In [4]: #Grabbing the first character of the years in city string to get rid of plus sign, and converting
to int
gdf['city_years'] = gdf['Stay_In_Current_City_Years'].str.get(0).astype(int)

In [7]: #Now we can see how we can control what the value of our dummies with the replace method and turn
strings to ints
gdf['City_Category'] = gdf['City_Category'].str.replace('A', '1')
gdf['City_Category'] = gdf['City_Category'].str.replace('B', '2')
gdf['City_Category'] = gdf['City_Category'].str.replace('C', '3')
gdf['City_Category'] = gdf['City_Category'].str.astype(int)
```

- Python**
- A Python library for manipulating GPU DataFrames following the Pandas API
  - Python interface to CUDA C++ library with additional functionality
  - Creating GPU DataFrames from Numpy arrays, Pandas DataFrames, and PyArrow Tables
  - JIT compilation of User-Defined Functions (UDFs) using Numba

36 NVIDIA

So, the backbone of data science as we said sometime back is cuDF. We are going to go into more details of cuDF in the next section, but it is a Python library. It is actually having a Python interface for the CUDA C++ library behind the scenes. It has the interface very similar to your CPU side of the world and it supports high performance using JIT based compilations also if required, using module called as Numba.

(Refer Slide Time: 25:33)



## GPU DATA FRAME (cuDF)

Technical Overview

- ▶ Basic approach
  - ▶ Define in-memory format for tabular data + metadata
  - ▶ Applications + libraries exchange device pointers to data structures
  - ▶ CUDA Inter-Process Communication (IPC) allows device pointers to be moved between processes
- ▶ cuDF Data Format
  - ▶ Currently is a subset of [Apache Arrow](#) specification
  - ▶ Precise subset not fully defined
  - ▶ Includes: numerical columns, and dictionary-encoded columns ("categorical" columns)

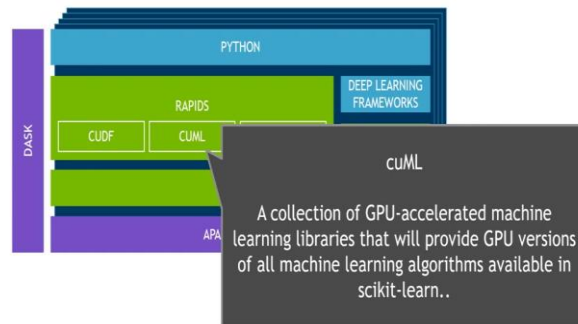
37 NVIDIA

So, again this is a very again a repetition of what we said, but it is kind of clear with respect to what cuDF is used for.

(Refer Slide Time: 25:43)




## RAPIDS – OPEN GPU DATA SCIENCE Software Stack



Once you are done with the data manipulation that is why you move towards the next one which is cuML. cuML as the name says its CUDA accelerated machine learning; it is a collection of GPU accelerated machine learning libraries. And it provides the equivalence to something called as sklearn or scikit-learn, which is very popular and almost all the data scientist or everyone who is working on machine learning uses sklearn.

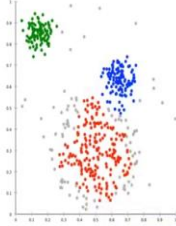
So, sklearn is a CPU equivalent while the GPU, if you want to isolate and run on the GPU it provides it is provided by RAPIDS in form of a package called as cuML.

(Refer Slide Time: 26:25)



## ALGORITHMS

GPU-accelerated Scikit-Learn



- Cross Validation
- Hyper-parameter Tuning


More to come!

- Classification / Regression
  - Decision Trees / Random Forests
  - Linear Regression
  - Logistic Regression
  - K-Nearest Neighbors
- Inference
- Clustering
  - K-Means
  - DBSCAN
  - Spectral Clustering
- Decomposition & Dimensionality Reduction
  - Principal Components
  - Singular Value Decomposition
  - UMAP
  - Spectral Embedding
- Time Series
  - Holt-Winters
  - Kalman Filtering

39 NVIDIA

So, as I said it has almost all the functionalities of sklearn. It provides your time series based options, it has clustering algorithms, it has classification and regression algorithms and day and day there are many many more, which are basically coming into this cuML and since, its open source initiatives there are a lot of algorithms which are getting added. It is still does not have all the features of sklearn. But it is becoming more and more comprehensive each and every day.

(Refer Slide Time: 26:59)



## RAPIDS MATCHES COMMON PYTHON APIS

CPU-Based Clustering

```
from sklearn.datasets import make_moons
import pandas

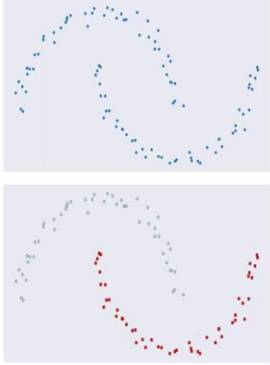
X, y = make_moons(n_samples=int(1e2),
                 noise=0.05, random_state=0)

X = pandas.DataFrame({'feat%d'%i: X[:, i]
                    for i in range(X.shape[1])})
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)


y_hat = dbscan.predict(X)
```



40 NVIDIA

So, this is just a very short example, if you were to doing CPU based clustering, you will obviously, use sklearn and you would do it in form of a pandas data frame. So, you will first create a pandas data frame, then you will import a sklearn clustering algorithm like DBSCAN you will do a fit and then a predict.

(Refer Slide Time: 27:23)




## RAPIDS MATCHES COMMON PYTHON APIS

GPU-Accelerated Clustering

```
from sklearn.datasets import make_moons
import cudf

X, y = make_moons(n_samples=int(1e2),
                  noise=0.05, random_state=0)


X = cudf.DataFrame({'feat%d'%i: X[:, i]
                   for i in range(X.shape[1])})
```





```
from cuml import DBSCAN
dbscan = DBSCAN(eps = 0.3, min_samples = 5)

dbscan.fit(X)

y_hat = dbscan.predict(X)
```



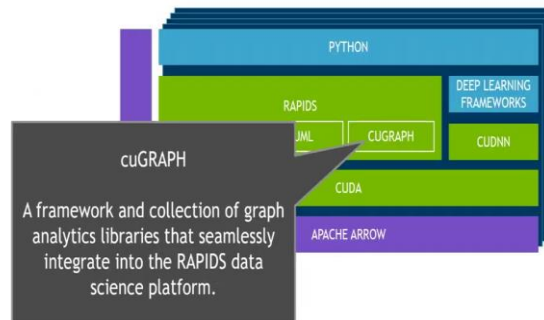
 41 

Same thing on a GPU you can see practically there is only 2 change. One is instead of importing sklearn you are basically, sorry, instead of importing pandas you are importing cuDF and instead of importing sklearn you are importing cuML and rest of your code remains the same. So, if you have a GPU just by changing or importing you can actually get the kind of performance that we were talking about sometime back.

(Refer Slide Time: 27:49)



## RAPIDS – OPEN GPU DATA SCIENCE Software Stack

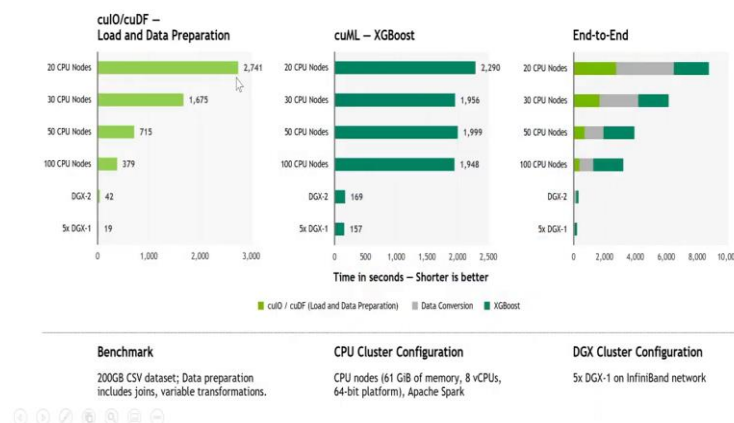


cuGRAPH is a framework and it is basically a connection of graph analytics library. So, many of us want to work on graph algorithms. So, there is a package which is primary responsible for doing only graph related activities.

(Refer Slide Time: 28:09)



## BENCHMARKS



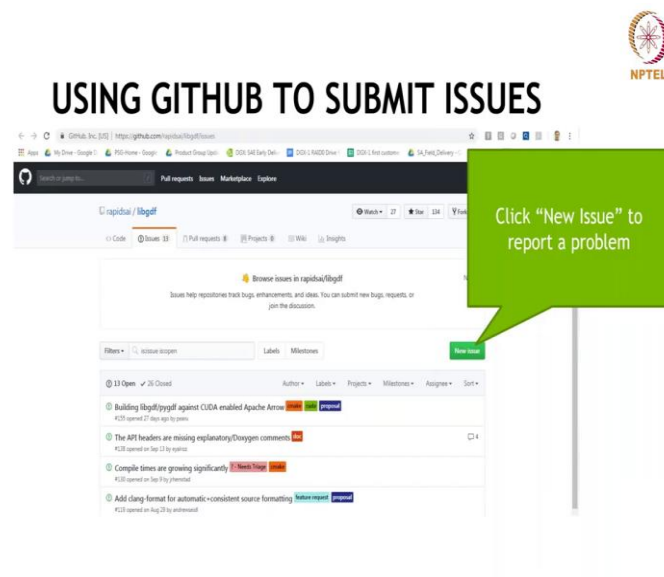
So, here are again certain additional benchmarks like if using cuIO and cuDF. So, if you were to do the data frame manipulations, how many CPU nodes. So, everything is kind of listed here what database data set it is, what configuration is being used and all. So, it takes so many amount of time, but on a one DGX system which is DGX with 16 GPUs,

you can see still 100 CPU nodes is not able to meet the same speed that we are expecting.

Same goes with cuML given with 100 CPU nodes for this algorithm called as XGBoost. You 1 GPU node is still surpassing by different folds a combination of 100 different CPU nodes connected to each other. And if I talk about the whole end to end pipeline, again you can see 1 GPU node is surpassing 100 CPU node by many folds and that is the part that I had brought out in the beginning which is return on investment.

The return on investment is very critical to making data science go in the production environment and GPU kind of or accelerated data science with the help of GPUs can actually solve that particular problem as well. It is not just about acceleration or reducing time, but also getting your return on investment with respect to the never ending increasing workload.

(Refer Slide Time: 29:45)

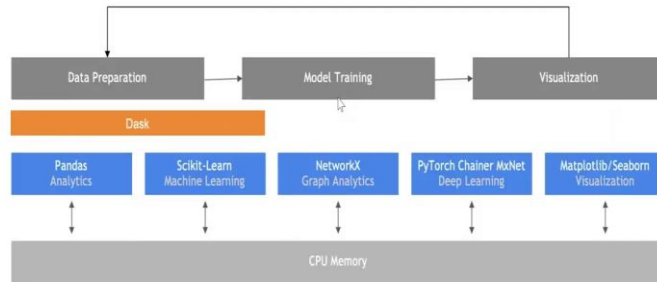


So, as I said RAPIDS is basically an open source project. You can anytime go to [github.com/rapidsai](https://github.com/rapidsai) and you can basically download it. Or you can in fact, in case you are facing any problem, you can report a "New Issue", report a problem, or you can in fact, raise a feature request which you want which is not there in your current data science pipeline.

(Refer Slide Time: 30:09)

## OPEN SOURCE DATA SCIENCE ECOSYSTEM

Familiar Python APIs



If I were to summarize, whatever I talked about, this is the open source data science platform, which is familiar to all the CPU or the sequential computing environment. So, for data preparation you would generally use the most popular library out there is pandas, which is used for analytics and then for machine learning you have sklearn or scikit-learn, which is the most popular machine learning framework and if you want to take it across different nodes or distribute it you have DASK.

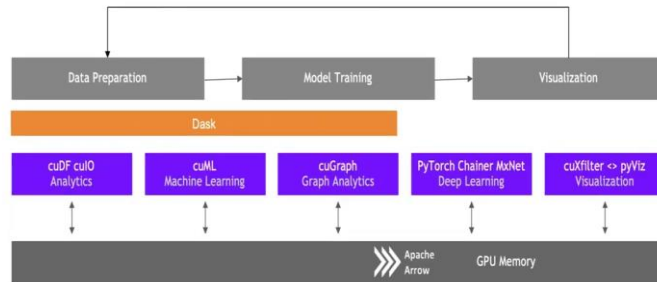
For graph analytics you have the most popular library called as networkX and then for deep learning we have seen and for visualization you will use Matplotlib or Seaborn.

(Refer Slide Time: 30:55)



## RAPIDS ECOSYSTEM

End-to-End GPU-Accelerated Data Science

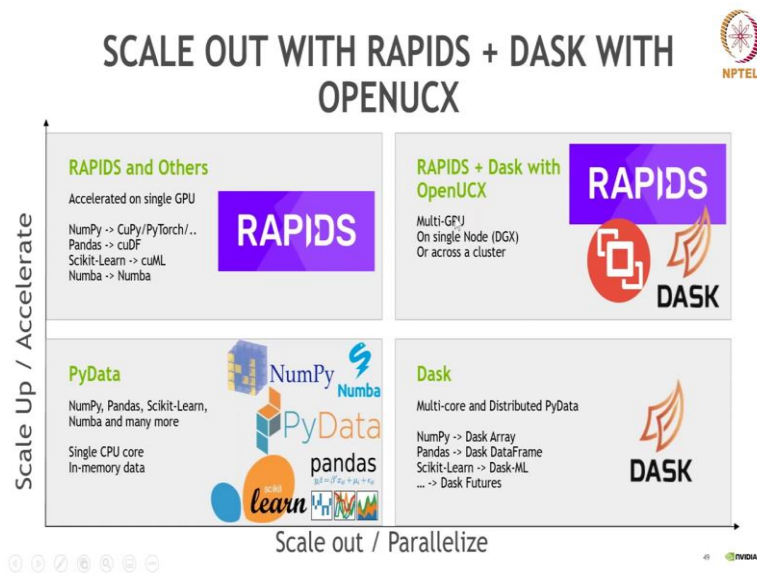


Now, the same equivalent in the RAPIDS ecosystem would look something like this, where instead of pandas for analytics you end up using cuDF or cuIO. You know instead of using the sklearn you equivalent you just change the imports to using cuML for machine learning. Instead of using networkX you can use a library called as cuGraph for graph analytics.

We have already covered the deep learning part in extreme, but for visualization also we have a library cuXfilter and pyViz which kind of accelerates not just visualization of small data, but the large amount of data as well. But also to support the overall pipeline in memory and to have the compatibility across different frameworks everything is done over apache arrow. So, that it remains also in the GPU memory.



(Refer Slide Time: 31:49)



So, if I were to say with respect to scaling, the traditional environment of PyData which is NumPy pandas, scikit-learn Numba and all you can scale up or accelerate by using RAPIDS framework like CuPy, cuDF and all, but you can scale out across different nodes with help of DASK as well.

So, you can scale by using traditional DASK or you can scale using RAPIDS+DASK, which kind of supports multi GPU, multi node kind of environment as well. So, today what we have seen is a very quick introduction or the motivation towards accelerated data analytics or data science. We talked about different kinds of roles data scientists, data engineers the cycle the pipeline through which the overall cycle goes which is the data preparation followed by the training and finally, the validation and also deployment.

We talked about a framework for the package which is called as RAPIDS, which helps in accelerating all of this. And different components of the RAPIDS package and we are going to look in the next couple of lectures into more detail of each one of them in form of a demo session.