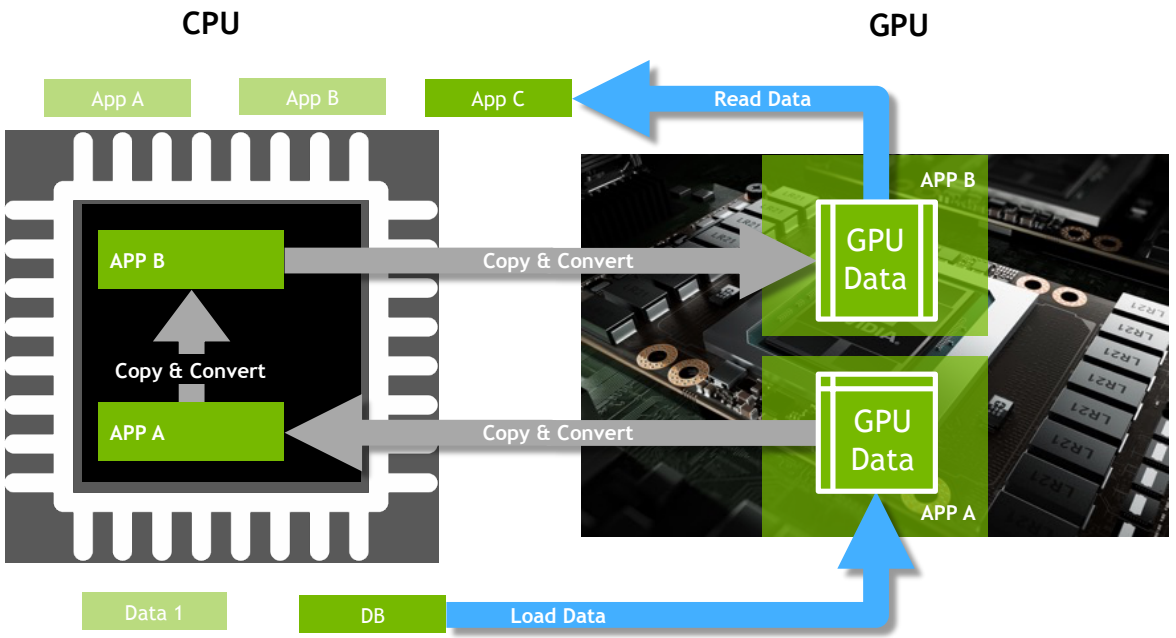

WHY RAPIDS

IMPETUS

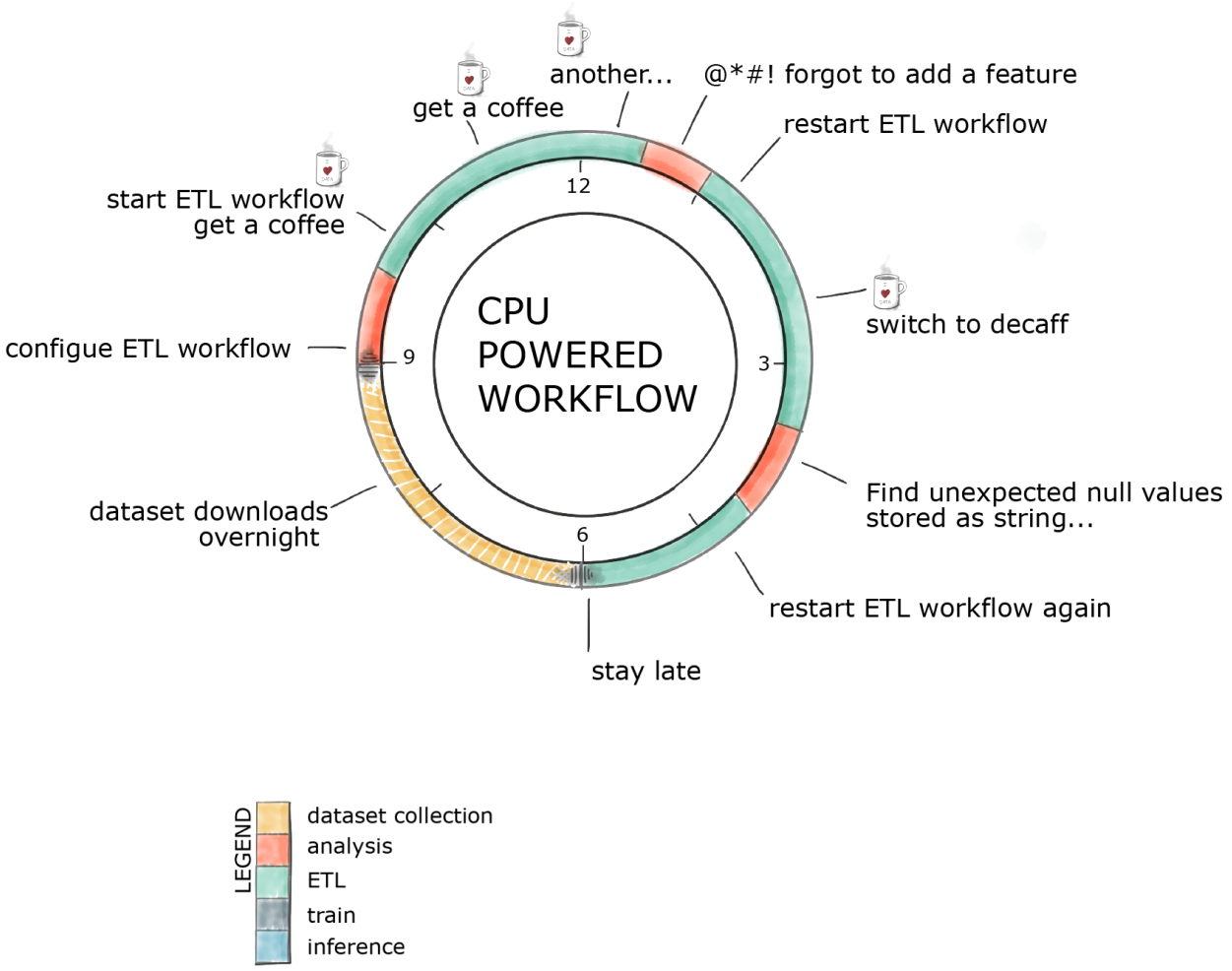
No one was *addressing* the big issue

- Alternative tools and frameworks
  - Too many data formats (data marshalling)
  - Too much data movement (in and out of GPU to perform marshalling)
  - ❖ Too much focus on individual algorithms and not end-to-end performance



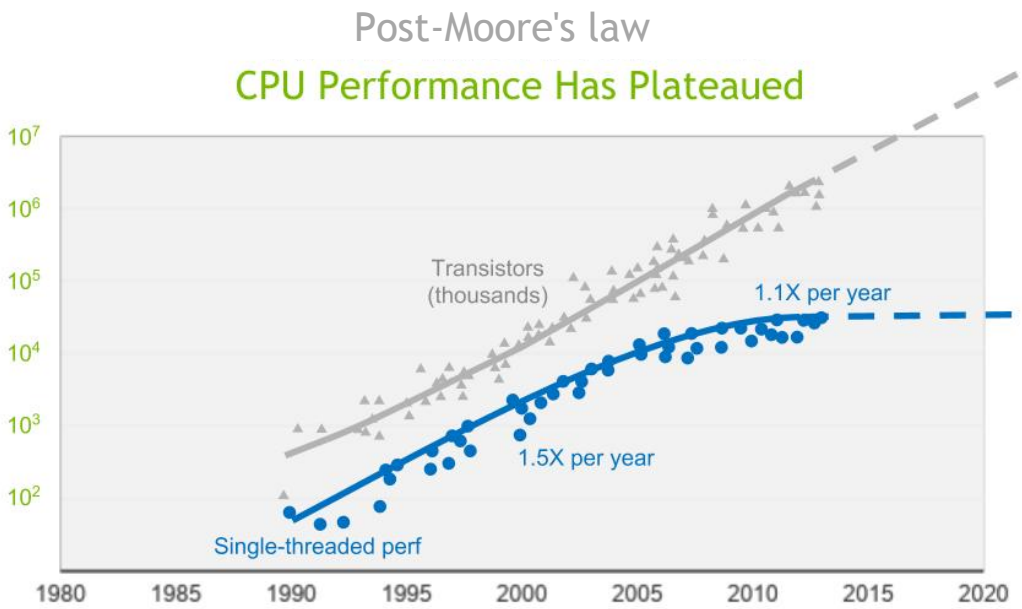
DAY IN THE LIFE

Or Why I want to be a Data Scientist



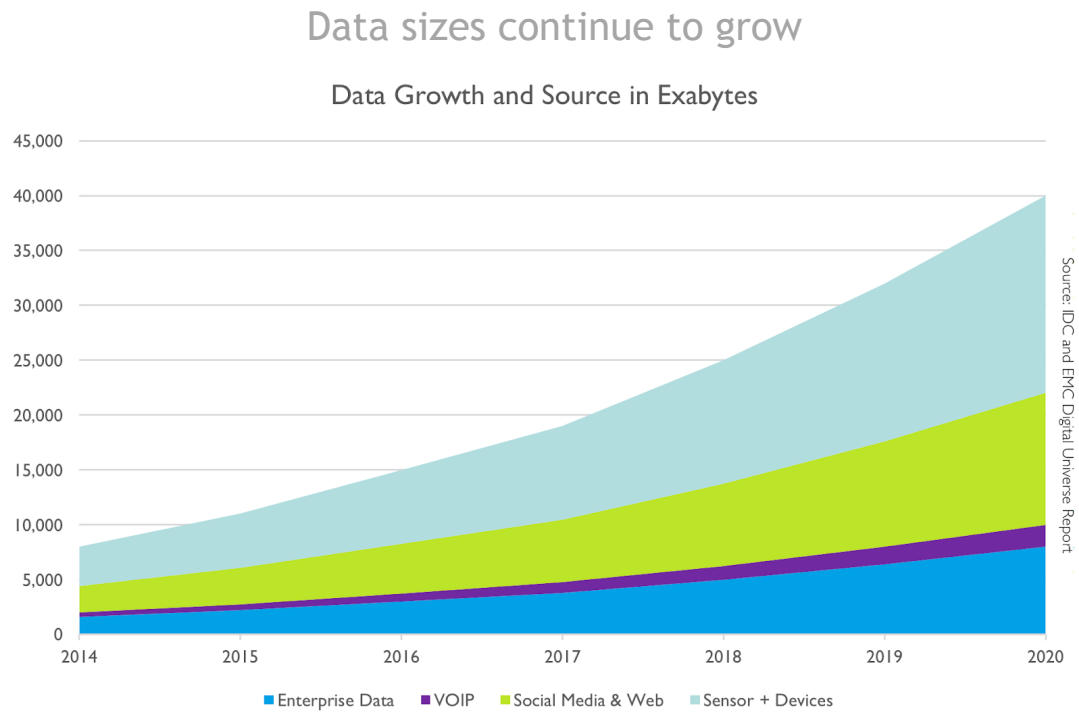
The average data scientist spends 90+% of their time in ETL as opposed to training models

SEEMS LIKE A LOT OF UNPRODUCTIVE TIME?



Moore's law is no longer a predictor of capacity in CPU market growth

- Distributing CPUs exacerbates the problem



WE NEED MORE COMPUTE!

Basic workloads are bottlenecked by the CPU

- In a simple benchmark consisting of aggregating data, the **CPU is the bottleneck**
- This is after the data is parsed and cached into memory which is another common bottleneck
- The CPU bottleneck is even worse in more complex workloads!

SELECT cab_type, count(*) FROM trips_orc GROUP BY cab_type;

```
top - 08:54:14 up  1:50,  4 users,  load average: 0.20, 1.64, 6.43
Tasks: 360 total,   2 running, 358 sleeping,   0 stopped,   0 zombie
%Cpu0  : 94.7 us,  1.7 sy,  0.0 ni,  3.3 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0
%Cpu1  : 95.0 us,  1.7 sy,  0.0 ni,  3.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu2  : 98.3 us,  0.3 sy,  0.0 ni,  1.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu3  : 87.3 us,  4.3 sy,  0.0 ni,  8.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu4  : 95.0 us,  1.3 sy,  0.0 ni,  3.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu5  : 98.3 us,  0.0 sy,  0.0 ni,  1.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu6  : 96.7 us,  1.3 sy,  0.0 ni,  2.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu7  : 92.7 us,  1.0 sy,  0.0 ni,  5.6 id,  0.3 wa,  0.0 hi,  0.3 si,  0.0
%Cpu8  : 93.7 us,  1.3 sy,  0.0 ni,  5.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu9  : 92.3 us,  0.7 sy,  0.0 ni,  7.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu10 : 97.3 us,  0.7 sy,  0.0 ni,  2.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu11 : 97.3 us,  0.7 sy,  0.0 ni,  2.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu12 : 92.0 us,  3.0 sy,  0.0 ni,  5.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu13 : 94.9 us,  1.0 sy,  0.0 ni,  4.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu14 : 88.3 us,  3.0 sy,  0.0 ni,  8.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu15 : 92.6 us,  2.3 sy,  0.0 ni,  4.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0
%Cpu16 : 94.7 us,  2.3 sy,  0.0 ni,  2.6 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0
%Cpu17 : 93.0 us,  0.7 sy,  0.0 ni,  6.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0
%Cpu18 : 93.0 us,  3.7 sy,  0.0 ni,  3.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
%Cpu19 : 91.2 us,  0.7 sy,  0.0 ni,  8.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
```

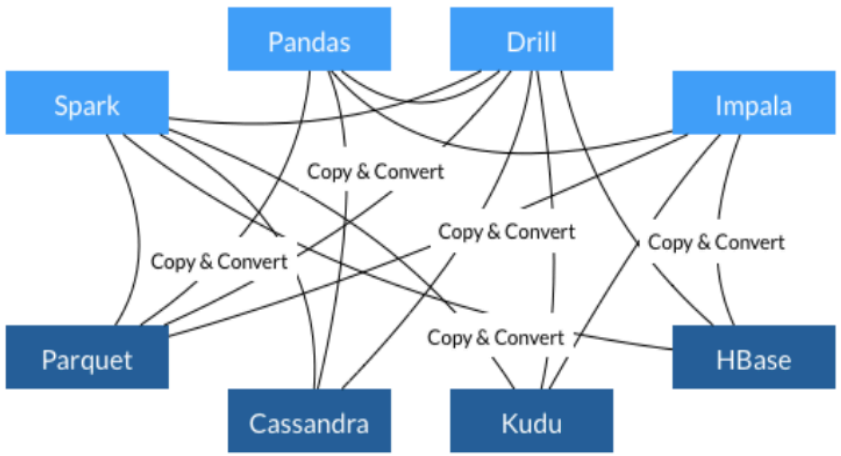
Source: Mark Litwintschik's blog: 1.1 Billion Taxi Rides: EC2 versus EMR

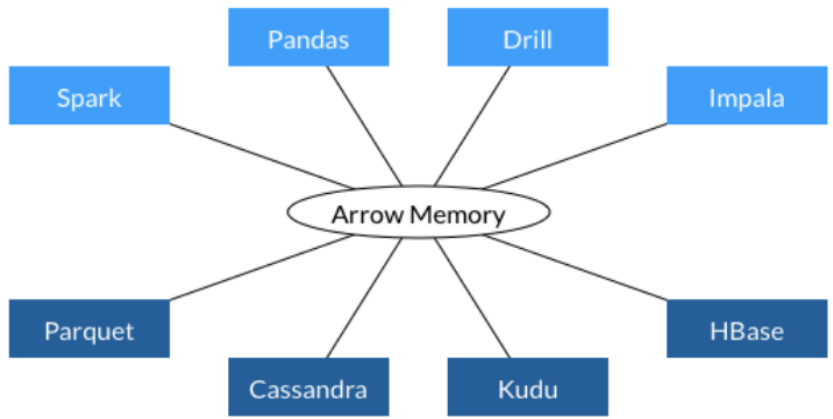
HOW CAN WE DO BETTER?

- Focus on the full Data Science workflow
  - Data Loading
  - Data Transformation
  - Data Analytics

- Python
  - Provide close to a drop-in replacement for existing tools
- Performance - Leverage GPUs

LEARNING FROM APACHE ARROW





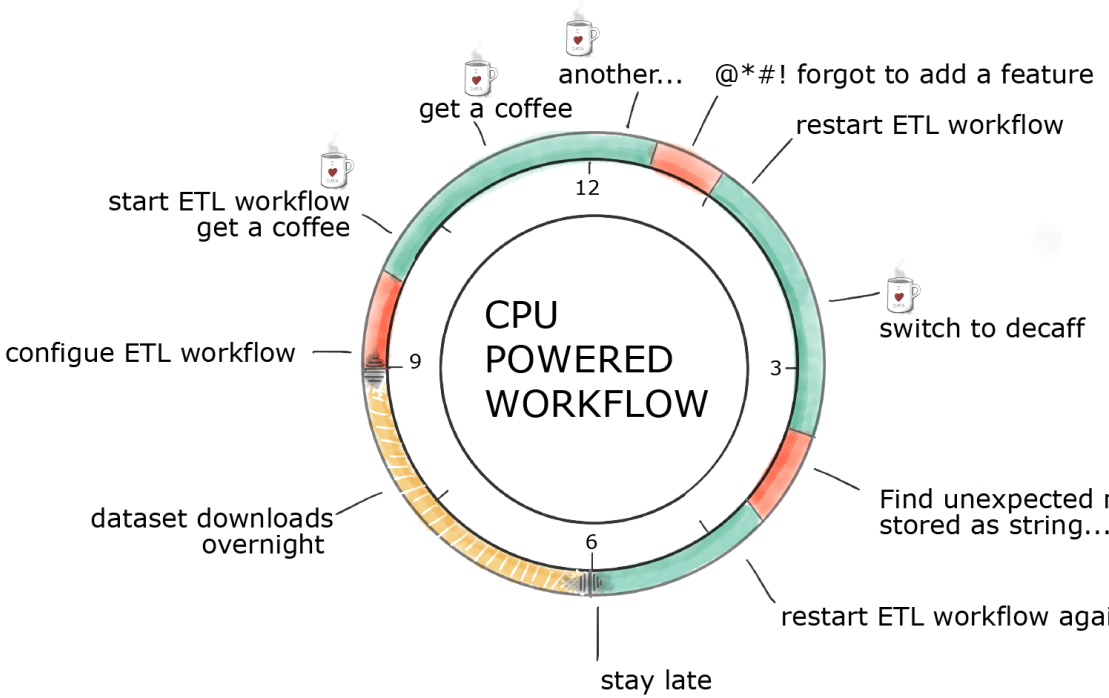
- Each system has its own internal memory format
- 70-80% computation wasted on serialization and deserialization
- Similar functionality implemented in multiple projects

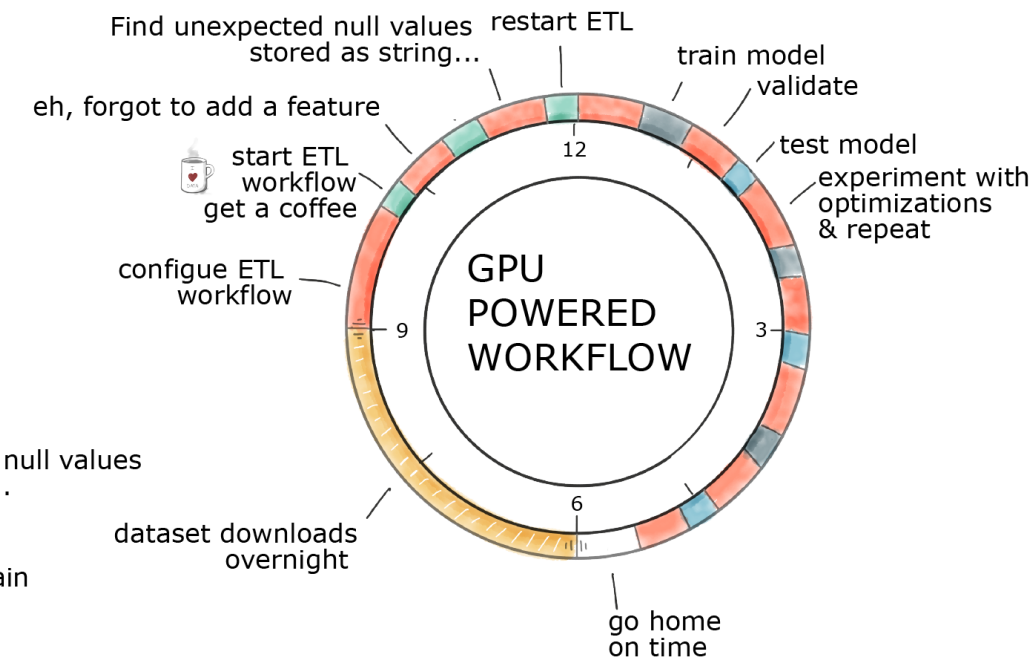
- All systems utilize the same memory format
- No overhead for cross-system communication
- Projects can share functionality (eg, Parquet-to-Arrow reader)

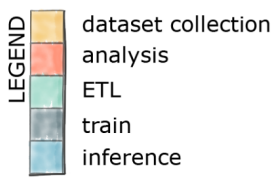
From Apache Arrow Home Page - https://arrow.apache.org/

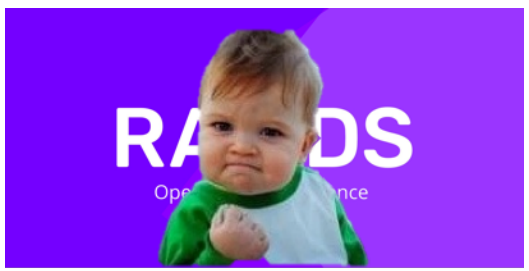
DAY IN THE LIFE

Data Scientist Using RAPIDS

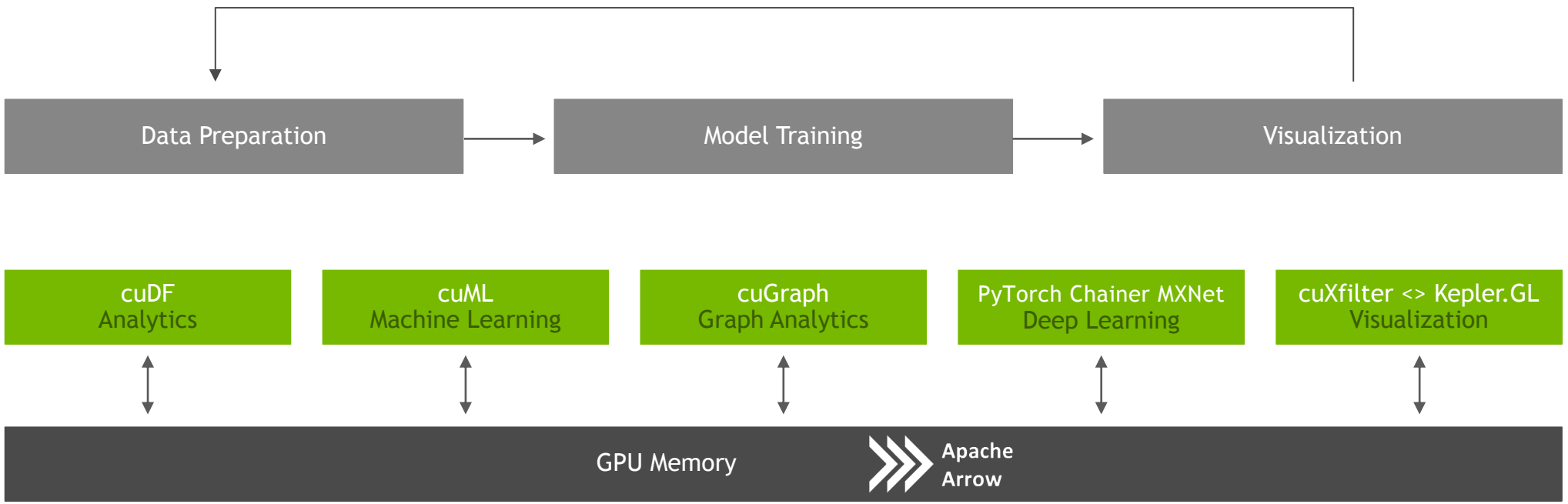








RAPIDS OPEN SOURCE SOFTWARE



We will touch on many of these components

“More data requires better approaches!”

~ Xavier Amatriain (CTO of Curai)

CUDF

Sorry about the upcoming wall of words

DATA PROCESSING ENGINE

cuDF

The World's First Accelerated Data Frame Library

Over a decade of NVIDIA Engineering has made this possible
Ease of use of pandas, scalability of Spark, in Python the most used data science language
Performs up to 50x faster than Spark on a single V100*

Key Features

Complete support for all standard data frame routines found in Pandas, Spark, R, or SAS
Single, Double, INT, Float, String, Dictionary, Date Time support and more
Support for multiple GPUs and concurrent kernels
Supports CUDA streams for concurrent operations
Python Bindings, Numba
Device API that can be called with your own CUDA kernels
I/O & Accelerated parsing of standard formats: CSV, Parquet, ORC, JSON, XML
JDBC, ODBC, and HDFS Connectors
Direct integrations with cuArray, cuGraph, & cuML (no copies needed)

*Dual 20-core vs a single V100.

CUDF

GPU DataFrame library

	Area Abbreviation	Area Code	Area	Item Code	Item	Element Code	Element	Unit	latitude	longitude	...	Y2004	Y2005	Y2006	Y2007	Y2008
0	AF	2	Afghanistan	2511	Wheat and products	5142	Food	1000 tonnes	33.94	67.71	...	3249.0	3486.0	3704.0	4164.0	4252.0
1	AF	2	Afghanistan	2805	Rice (Milled Equivalent)	5142	Food	1000 tonnes	33.94	67.71	...	419.0	445.0	546.0	455.0	490.0
2	AF	2	Afghanistan	2513	Barley and products	5521	Feed	1000 tonnes	33.94	67.71	...	58.0	236.0	262.0	263.0	230.0
3	AF	2	Afghanistan	2513	Barley and products	5142	Food	1000 tonnes	33.94	67.71	...	185.0	43.0	44.0	48.0	62.0
4	AF	2	Afghanistan	2514	Maize and products	5521	Feed	1000 tonnes	33.94	67.71	...	120.0	208.0	233.0	249.0	247.0
5	AF	2	Afghanistan	2514	Maize and products	5142	Food	1000 tonnes	33.94	67.71	...	231.0	67.0	82.0	67.0	69.0
6	AF	2	Afghanistan	2517	Millet and products	5142	Food	1000 tonnes	33.94	67.71	...	15.0	21.0	11.0	19.0	21.0
7	AF	2	Afghanistan	2520	Cereals, Other	5142	Food	1000 tonnes	33.94	67.71	...	2.0	1.0	1.0	0.0	0.0
8	AF	2	Afghanistan	2531	Potatoes and products	5142	Food	1000 tonnes	33.94	67.71	...	276.0	294.0	294.0	260.0	242.0
9	AF	2	Afghanistan	2536	Sugar cane	5521	Feed	1000 tonnes	33.94	67.71	...	50.0	29.0	61.0	65.0	54.0
10	AF	2	Afghanistan	2537	Sugar beet	5521	Feed	1000 tonnes	33.94	67.71	...	0.0	0.0	0.0	0.0	0.0

- Apache Arrow data format
- Pandas-like API

- Unary and Binary Operations
- Joins / Merges
- Group Bys
- Filters

- User-Defined Functions (UDFs)
- Accelerated file readers
- Etc.

Anything you can do in Pandas you will be able to do in cuDF*

Methods

`add_column` (name, data[, forceindex])	Add a column
`apply_chunks` (func, incols, outcols[, ...])	Transform user-specified chunks using the user-provided function.
`apply_rows` (func, incols, outcols, kwargs[, ...])	Apply a row-wise user defined function.
`as_gpu_matrix` ([columns, order])	Convert to a matrix in device memory.
`as_matrix` ([columns])	Convert to a matrix in host memory.
`assign` (**kwargs)	Assign columns to DataFrame from keyword arguments.
`copy` ([deep])	Returns a copy of this dataframe
`drop` (labels)	Drop column(s)
`drop_column` (name)	Drop a column by *name*
`fillna` (value[, method, axis, inplace, limit])	Fill null values with `value`.
`from_arrow` (table)	Convert from a PyArrow Table.
`from_gpu_matrix` (data[, index, columns, ...])	Convert from a numba gpu ndarray.
`from_pandas` (dataframe[, nan_as_null])	Convert from a Pandas DataFrame.
`from_records` (data[, index, columns, nan_as_null])	Convert from a numpy recarray or structured array.
`groupby` ([by, sort, as_index, method, level])	Groupby
`hash_columns` ([columns])	Hash the given *columns* and return a new Series
`head` ([n])	Returns the first n rows as a new DataFrame
`iteritems` ()	Iterate over column names and series pairs
`join` (other[, on, how, lsuffix, rsuffix, ...])	Join columns with other DataFrame on index or on a key column.
`label_encoding` (column, prefix, cats[, ...])	Encode labels in a column with label encoding.
`merge` (right[, on, how, left_on, right_on, ...])	Merge GPU DataFrame objects by performing a database-style join operat
`nlargest` (n, columns[, keep])	Get the rows of the DataFrame sorted by the n largest value of *columns*
`nsmallest` (n, columns[, keep])	Get the rows of the DataFrame sorted by the n smallest value of *columns*
`one_hot_encoding` (column, prefix, cats[, ...])	Expand a column with one-hot-encoding.
`partition_by_hash` (columns, nparts)	Partition the dataframe by the hashed value of data in *columns*.
`quantile` ([q, interpolation, columns, exact])	Return values at the given quantile.
`query` (expr)	Query with a boolean expression using Numba to compile a GPU kernel.
`rename` ([mapper, columns, copy, inplace])	Alter column labels.
`replace` (to_replace, value)	Replace values given in *to_replace* with *value*.
`select_dtypes` ([include])	Return a subset of the DataFrame's columns based on the column dtypes.
`set_index` (index)	Return a new DataFrame with a new index
`sort_index` ([ascending])	Sort by the index
`sort_values` (by[, ascending, na_position])	Sort by the values row-wise.
`tail` ([n])	Returns the last n rows as a new DataFrame
`to_arrow` ([preserve_index])	Convert to a PyArrow Table.
`to_dlpack` ()	Converts a cuDF object into a DLPack tensor.
`to_feather` (path, *args, **kwargs)	Write a DataFrame to the feather format.
`to_gpu_matrix` ()	Convert to a numba gpu ndarray
`to_hdf` (path_or_buf, key, *args, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
`to_json` ([path_or_buf])	Convert the cuDF object to a JSON string.
`to_pandas` ()	Convert to a Pandas DataFrame.
`to_parquet` (path, *args, **kwargs)	Write a DataFrame to the parquet format.
`to_records` ([index])	Convert to a numpy recarray
`to_string` ([nrows, ncols])	Convert to string
`transpose` ()	Transpose index and columns.

https://docs.rapids.ai/api/cudf/nightly/

Methods

`append` (arbitrary)	Append values from another `Series` or array-like object.
`applymap` (udf[, out_dtype])	Apply a elemenwise function to transform the values in the Column.
`argsort` ([ascending, na_position])	Returns a Series of int64 index that will sort the series.
`as_mask` ()	Convert booleans to bitmask
`astype` (dtype)	Convert to the given `dtype`.
`ceil` ()	Rounds each value upward to the smallest integral value not less than the or
`count` ([axis, skipna])	The number of non-null values
`digitize` (bins[, right])	Return the indices of the bins to which each value in series belongs.
`factorize` ([na_sentinel])	Encode the input values as integer labels
`fillna` (value[, method, axis, inplace, limit])	Fill null values with `value`.
`find_first_value` (value)	Returns offset of first value that matches
`find_last_value` (value)	Returns offset of last value that matches
`floor` ()	Rounds each value downward to the largest integral value not greater than th
`from_categorical` (categorical[, codes])	Creates from a pandas.Categorical
`from_masked_array` (data, mask[, null_count])	Create a Series with null-mask.
`hash_encode` (stop[, use_name])	Encode column values as ints in [0, stop) using hash function.
`hash_values` ()	Compute the hash of values in this column.
`label_encoding` (cats[, dtype, na_sentinel])	Perform label encoding
`masked_assign` (value, mask)	Assign a scalar value to a series using a boolean mask df[df < 0] = 0
`max` ([axis, skipna])	Compute the max of the series
`mean` ([axis, skipna])	Compute the mean of the series
`mean_var` ([ddof])	Compute mean and variance at the same time.
`min` ([axis, skipna])	Compute the min of the series
`nlargest` ([n, keep])	Returns a new Series of the *n* largest element.
`nsmallest` ([n, keep])	Returns a new Series of the *n* smallest element.
`nunique` ([method, dropna])	Returns the number of unique values of the Series: approximate version, and
`one_hot_encoding` (cats[, dtype])	Perform one-hot-encoding
`product` ([axis, skipna])	Compute the product of the series
`quantile` (q[, interpolation, exact, quant_index])	Return values at the given quantile.
`rename` ([index, copy])	Alter Series name.
`replace` (to_replace, value)	Replace values given in *to_replace* with *value*.
`reset_index` ([drop])	Reset index to RangeIndex
`reverse` ()	Reverse the Series
`scale` ()	Scale values to [0, 1] in float64
`set_index` (index)	Returns a new Series with a different index.
`set_mask` (mask[, null_count])	Create new Series by setting a mask array.
`sort_index` ([ascending])	Sort by the index.
`sort_values` ([ascending, na_position])	Sort by the values.
`std` ([ddof, axis, skipna])	Compute the standard deviation of the series
`sum` ([axis, skipna])	Compute the sum of the series
`tail` ([n])	Returns the last n rows as a new Series
`take` (indices[, ignore_index])	Return Series by taking values from the corresponding *indices*.
`to_array` ([fillna])	Get a dense numpy array for the data.
`to_dlpack` ()	Converts a cuDF object into a DLPack tensor.
`to_frame` ([name])	Convert Series into a DataFrame
`to_gpu_array` ([fillna])	Get a dense numba device array for the data.
`to_hdf` (path_or_buf, key, *args, **kwargs)	Write the contained data to an HDF5 file using HDFStore.
`to_json` ([path_or_buf])	Convert the cuDF object to a JSON string.
`to_string` ([nrows])	Convert to string
`unique` ([method, sort])	Returns unique values of this Series.
`value_counts` ([method, sort])	Returns unique values of this Series.
`values_to_string` ([nrows])	Returns a list of string for each element.
`var` ([ddof, axis, skipna])	Compute the variance of the series

CUDF

Today

CUDA

- Low level library containing function implementations and C/C++ API
- Importing/exporting Apache Arrow using the CUDA IPC mechanism
- CUDA kernels to perform element-wise math operations on GPU DataFrame columns
- CUDA sort, join, groupby, and reduction operations on GPU DataFrames

With Python Bindings

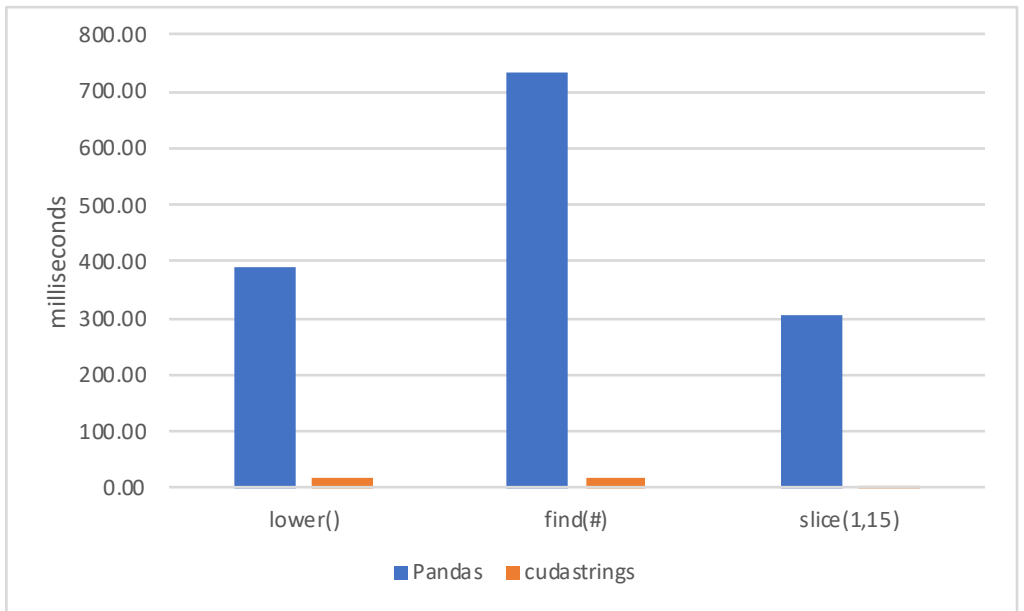
- A Python library for manipulating GPU DataFrames
- Python interface to CUDA C++ with additional functionality
- Creating Apache Arrow from Numpy arrays, Pandas DataFrames, and PyArrow Tables
- JIT compilation of User-Defined Functions (UDFs) using Numba

WOW!!! String on GPUs

STRING SUPPORT

GPU-Accelerated string functions with a Pandas-like API

- API and functionality is following Pandas: https://pandas.pydata.org/pandas-docs/stable/api.html#string-handling
- Handles ingesting and exporting typical Python objects (Pandas Series, Numpy arrays, PyArrow arrays, Python lists, etc.)
- Initial performance results:
  - lower(): ~22x speedup
  - find(): ~40x speedup
  - slice(): ~100x speedup



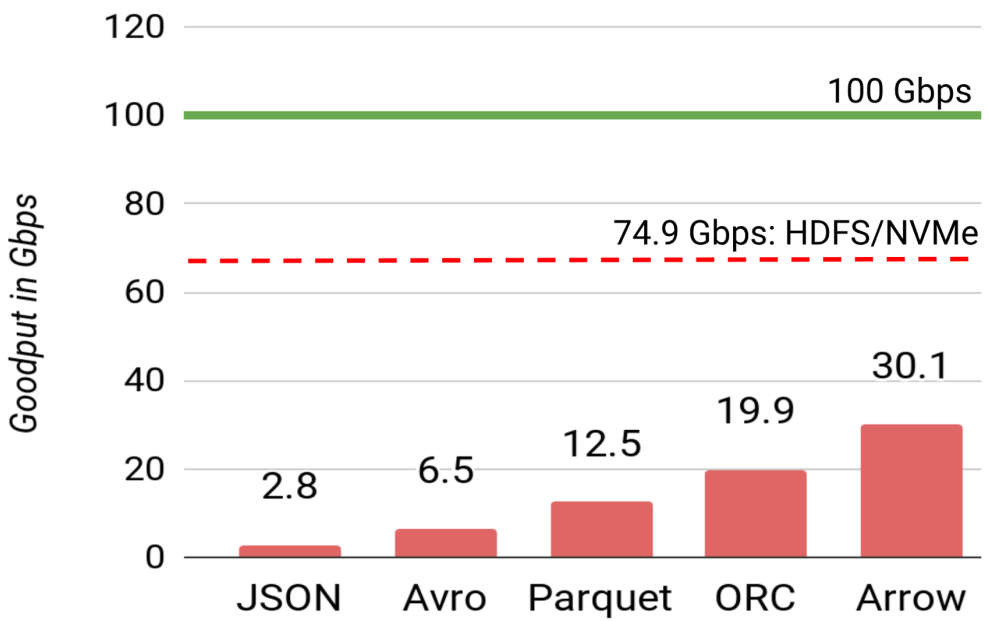
WOW!!! Data loading on GPUs

ACCELERATED DATA LOADING

- CSV Reader
  - CSV >10x speed improvement over pandas
  - Parquet
  - ORC Reader
  - JSON Reader
  - Feather Reader
  - HDF5 Reader
  - Arrow Reader
- Matches Pandas-API
- Decompression of the data ***will be*** GPU-accelerated as well!

Also performs writing

CPUs bottleneck data loading in high throughput systems



Source: Apache Crail blog: SQL Performance: Part 1 - Input File Formats

WOW!!! Data loading on GPUs

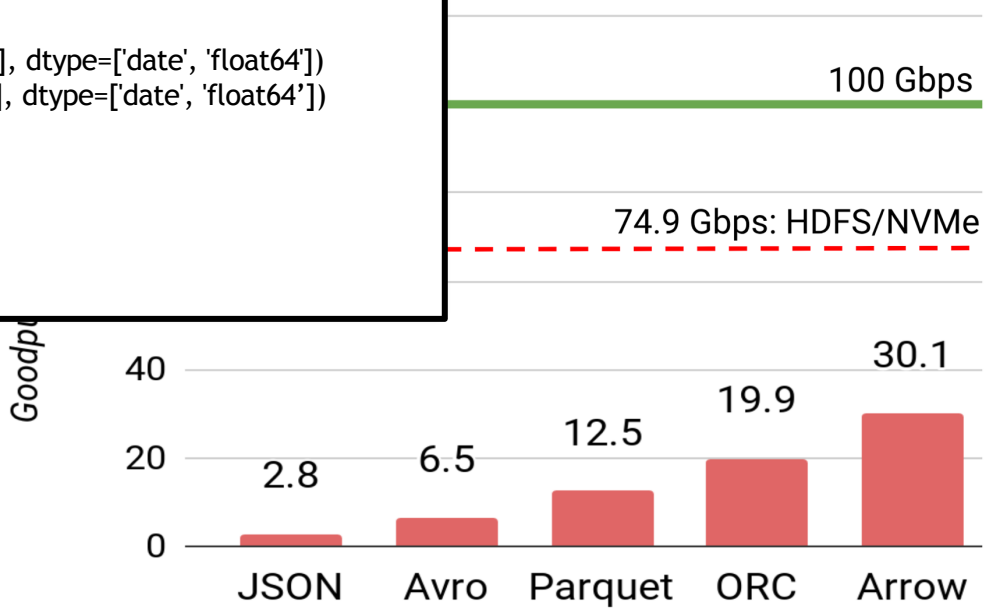
ACCELERATED DATA LOADING

- CSV Reader
  - CSV >10x speed improvement over pandas
  - Parquet
  - ORC Reader
  - JSON Reader
  - Feather Reader
  - HDF5 Reader
  - Arrow Reader

```
import pandas as pd
import cudf
…
df   =    pd.read_csv('foo.csv', names=['index', 'A'], dtype=['date', 'float64'])
gdf = cudf.read_csv('foo.csv', names=['index', 'A'], dtype=['date', 'float64'])
…

----- Or just ---
gdf = cudf.read_csv('foo.csv')
```

- Matches Pandas-API
- Decompression of the data ***will be*** GPU-accelerated as well!

CPUs bottleneck data loading in high throughput systems



Also performs writing

Source: Apache Crail blog: SQL Performance: Part 1 - Input File Formats

PYTHON CUDA ARRAY INTERFACE

Interoperability for Python GPU Array Libraries

- The CUDA array interface is a standard format that describes a GPU array to allow sharing GPU arrays between different libraries without needing to copy or convert data
- Native ingest and export of __cuda_array_interface__ compatible objects via Numba device arrays in cuDF
- Numba, CuPy, and PyTorch are the first libraries to adopt the interface:
  - https://numba.pydata.org/numba-doc/dev/cuda/cuda_array_interface.html
  - https://github.com/cupy/cupy/releases/tag/v5.0.0b4
  - https://github.com/pytorch/pytorch/pull/11984

Numba

CuPy

DLPACK

Interoperability with Deep Learning Libraries

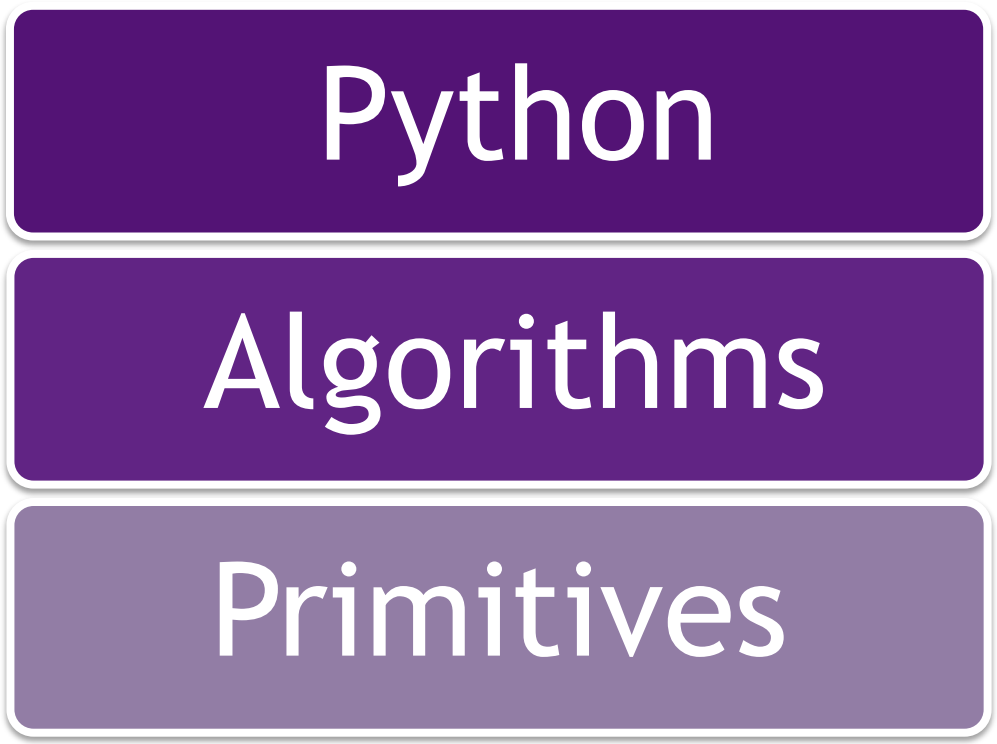
- DLPack is an open-source memory tensor structure designed to allow sharing tensors between deep learning frameworks
- Currently supported by PyTorch, MXNet, and Chainer / CuPy
- cuDF supports ingesting and exporting column-major DLPack tensors
  - If you're interested in row-major tensor support please let us know!

“Details are confusing. It is only by selection, by elimination, by emphasis, that we get to the real meaning of things.”

~ Georgia O'Keefe

cuML API

GPU-accelerated machine learning at every layer



Scikit-learn-like interface for data scientists utilizing cuDF & Numpy

CUDA C++ API to leverage accelerated machine learning algorithms.

Reusable building blocks for composing machine learning algorithms.

PRIMITIVES

GPU-accelerated math optimized for feature matrices

Linear Algebra

- Element-wise operations
- Matrix multiply
- Norms
- Eigen Decomposition
- SVD/RSVD
- Transpose
- QR Decomposition

Probability & Statistics

Random Sampling

Distances

Scores & Metrics

Objective Functions

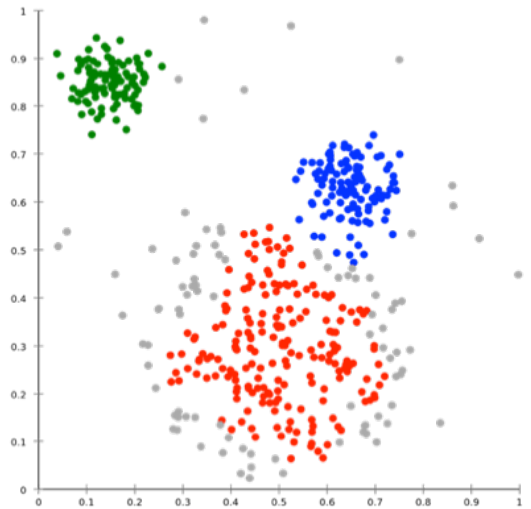
Sparse Matrices

More to come!

C++ & Python C++ Coming Soon

ALGORITHMS

GPU-accelerated Scikit-Learn



Classification / Regression
- Decision Trees / Random Forests
- Linear Regression
- Logistic Regression
- K-Nearest Neighbors
- Kalman Filtering
- Bayesian Inference
- Gaussian Mixture Models
- Hidden Markov Models

Clustering
- K-Means
- DBSCAN
- Spectral Clustering

Decomposition & Dimensionality Reduction
- Principal Components
- Singular Value Decomposition
- UMAP
- Spectral Embedding

Timeseries Forecasting
- ARIMA
- Holt-Winters

Recommendations
- Implicit Matrix Factorization

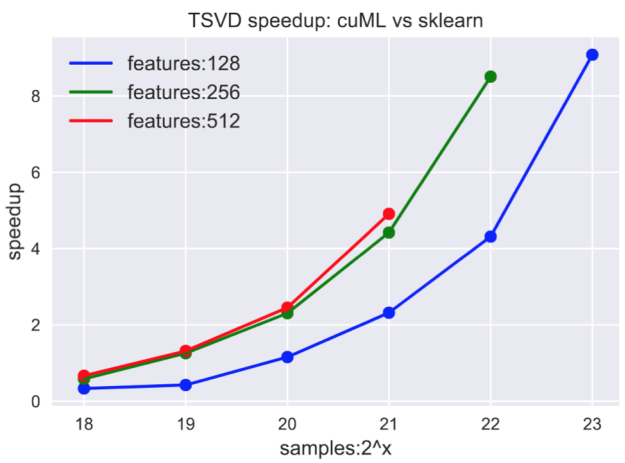
Cross Validation

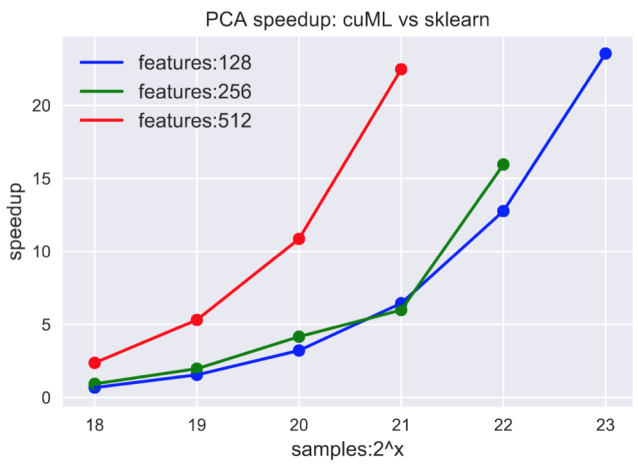
Hyperparameter Tuning

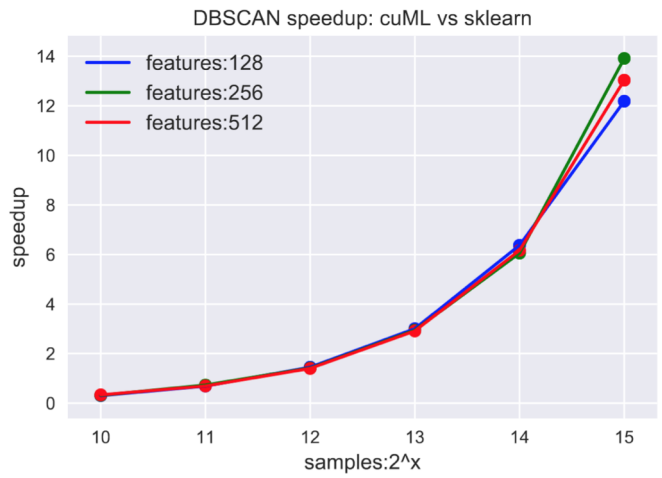
More to come!

CUML

Speedup comparison to scikit-learn

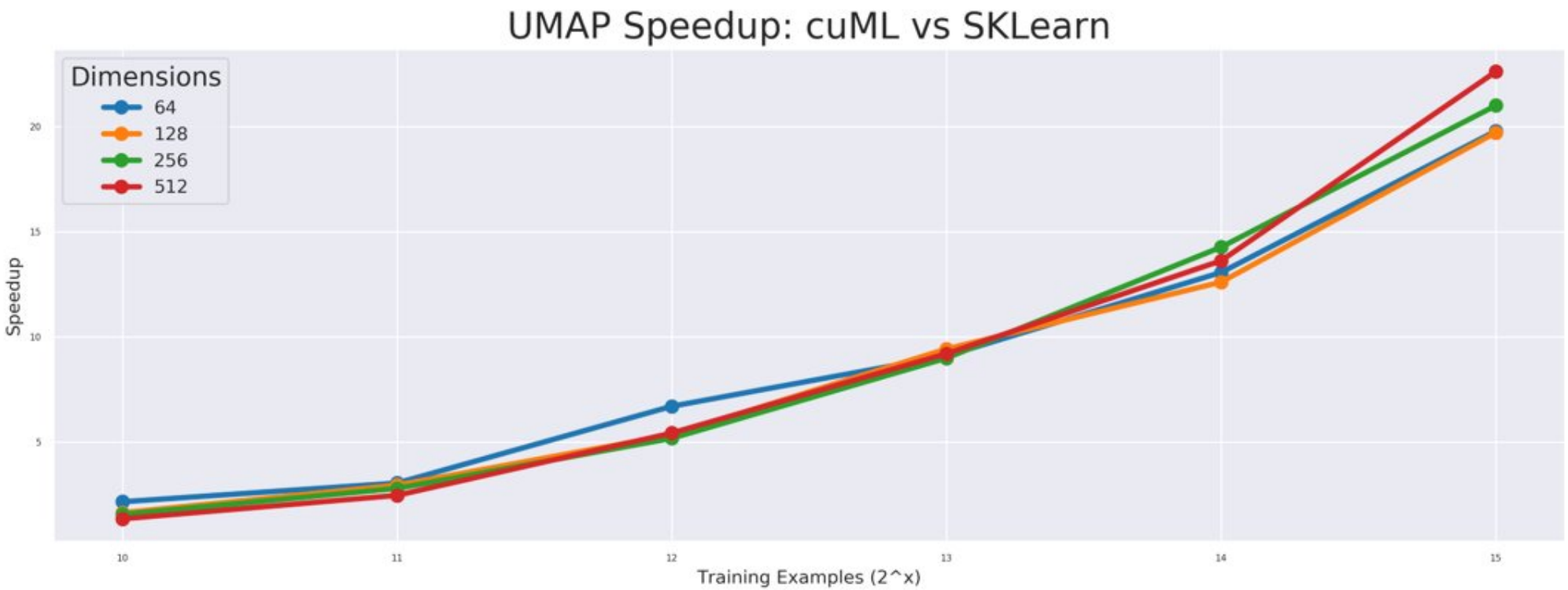






UMAP

Speedup comparison to scikit-learn



SINGLE NODE MULTI-GPU

Linear Regression

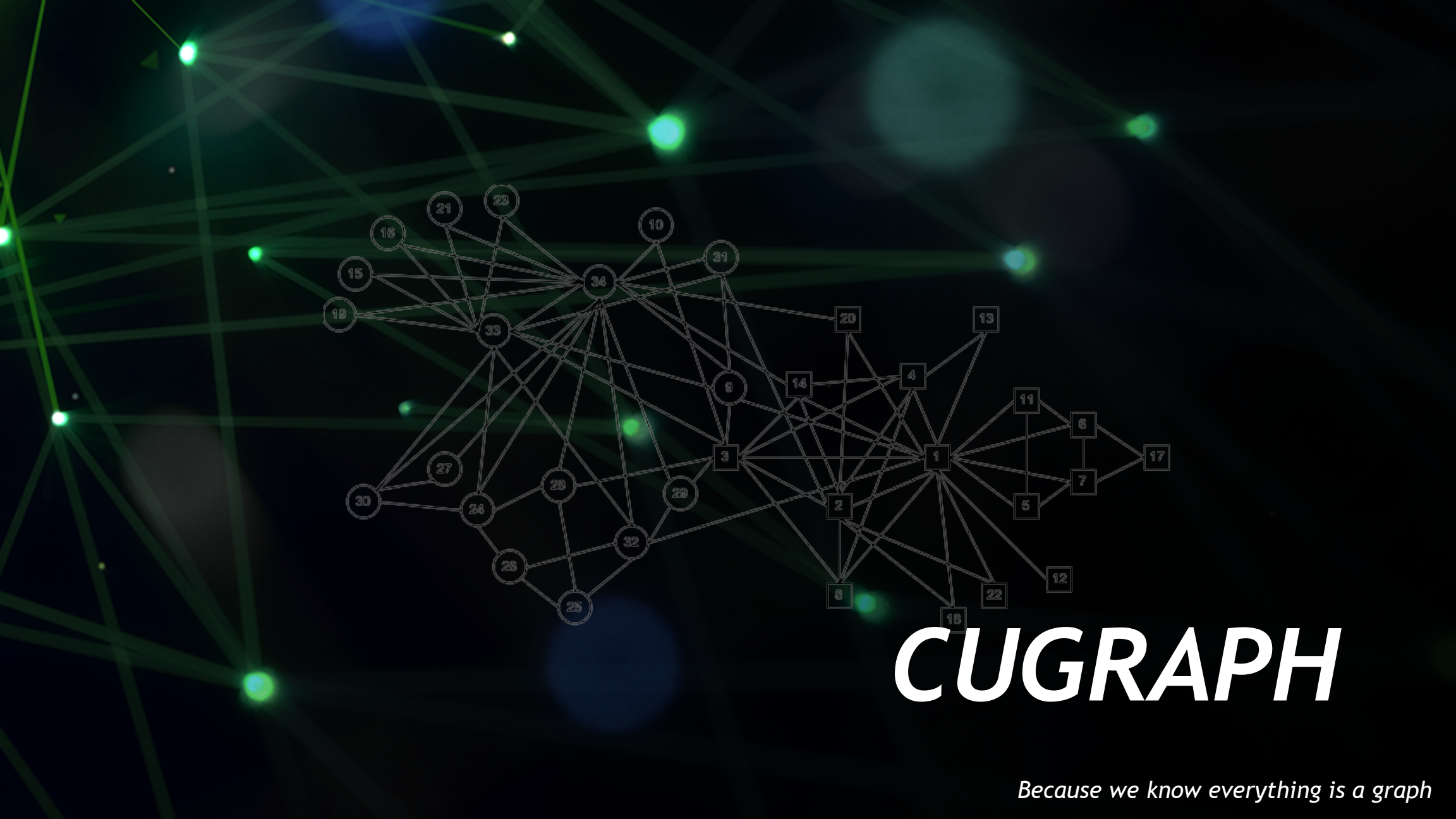
- Reduction: 40mins -> 1min
- Size: 225gb
- System: DGX2

tSVD

- Reduction: 1.6hrs-> 1.5min
- Size: 220gb
- System: DGX2

Nearest Neighbors

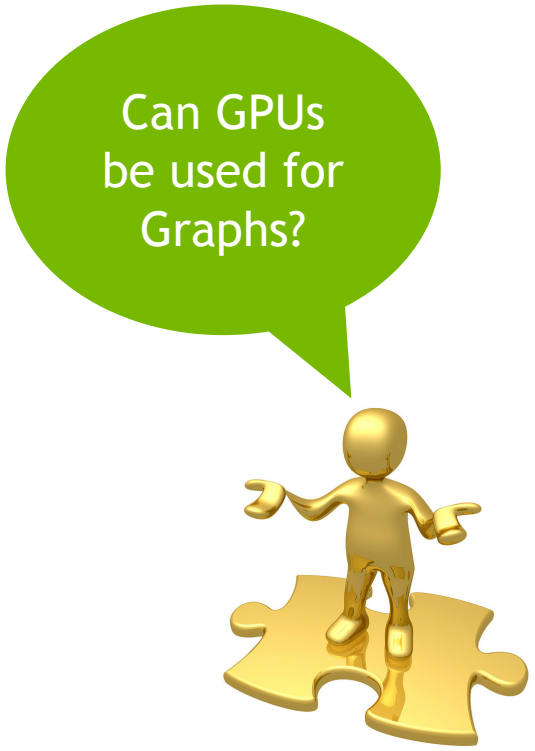
- Reduction: 4+hrs-> 30sec
- Size: 128gb
- System: DGX1



CUGRAPH

Because we know everything is a graph

GRAPHS ON GPU



- Yes!
  - Larger amount of GPU memory supports bigger graphs
  - CUDA supports faster computation
  - BLAS and BSP/GAS model for algorithms
  - Higher memory bandwidth supports faster pointer chasing

GOALS AND BENEFITS OF CUGRAPH

Focus on Features an Easy-of-Use

- Seamless integration with cuDF and cuML
  - Inputs are output are cuDF Dataframes

This is awesome

- Features
  - targeting the most commonly used algorithms from NetworkX, and Spark, as the initial set
  - Overlap Coefficient, for example, is not in NetworkX

- NetworkX-like API*

- Breakthrough Performance
  - 10 to 10000x faster *(your mileage may vary)*

* when possible

KDD will be a week before 0.9 is released

WHAT'S IN CUGRAPH

Current Single GPU Algorithms – as of release 0.8

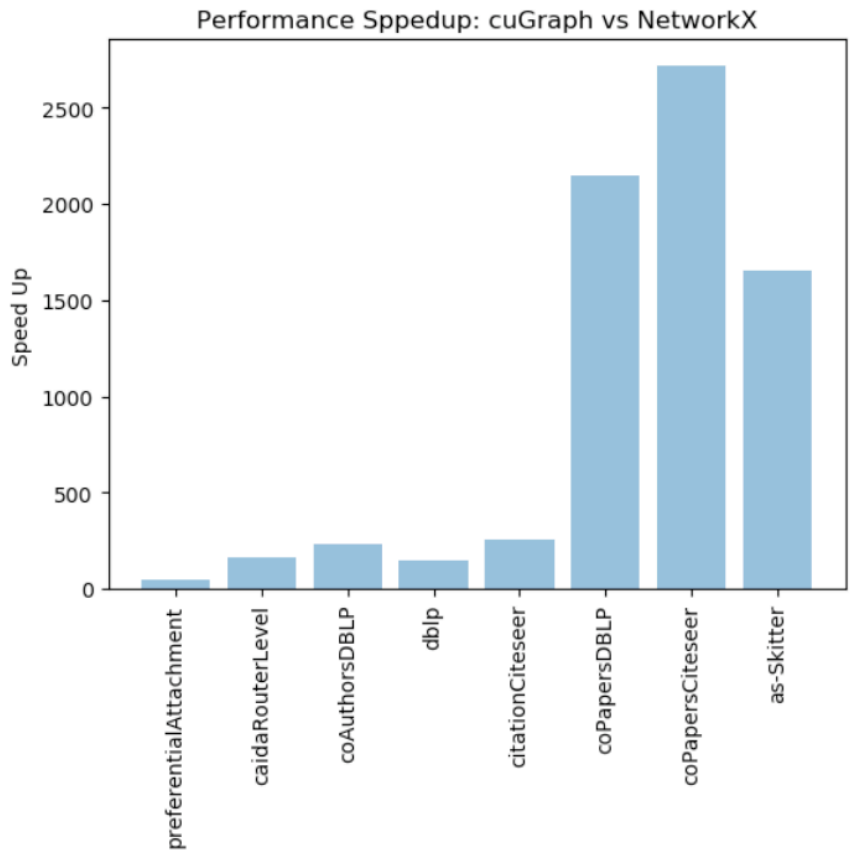
- Centrality
  - PageRank
    - MG being released in 0.9 (mid-Aug)
- Similarity
  - Jaccard
  - Weighted Jaccard
  - Overlap Coefficient
- Transveral
  - Single Source Shortest Path (SSSP)
  - Breadth First Search (BFS)
- Triangle Counting (TC)
- Subgraph Extraction
- Clustering
  - Spectral Clustering
    - Balanced-Cut
    - Modularity Maximization
  - Louvain
- Renumbering
- Graph functions
  - Size, Order, Degree
  - Get two hop neighbors

PAGERANK SPEEDUP

cuGraph PageRank vs NetworkX PageRank

```
G = cugraph.Graph()
G.add_edge_list(gdf['src'], gdf['dst'], None)

df = cugraph.pagerank(G, alpha, max_iter, tol)
```

File Name	Num of Vertices	Num of Edges
preferentialAttachment	100,000	999,970
caidaRouterLevel	192,244	1,218,132
coAuthorsDBLP	299,067	1,955,352
dblp-2010	326,186	1,615,400
citationCiteseer	268,495	2,313,294
coPapersDBLP	540,486	30,491,458
coPapersCiteseer	434,102	32,073,440
as-Skitter	1,696,415	22,190,596



https://github.com/rapidsai/notebooks-extended/tree/master/advanced/benchmarks/cugraph_benchmark

PageRank PERFORMANCE

Looking to the future – Multi-GPU processing

Single and Dual GPU on Commodity Workstation (PCIe)

Nodes	Edges	Single	Dual
1,048,576	16,777,216	0.019	0.020
2,097,152	33,554,432	0.047	0.035
4,194,304	67,108,864	0.114	0.066
8,388,608	134,217,728	0.302	0.162
16,777,216	268,435,456	0.771	0.353
33,554,432	536,870,912	1.747	0.821
67,108,864	1,073,741,824		1.880

DGX-2 (SMX)

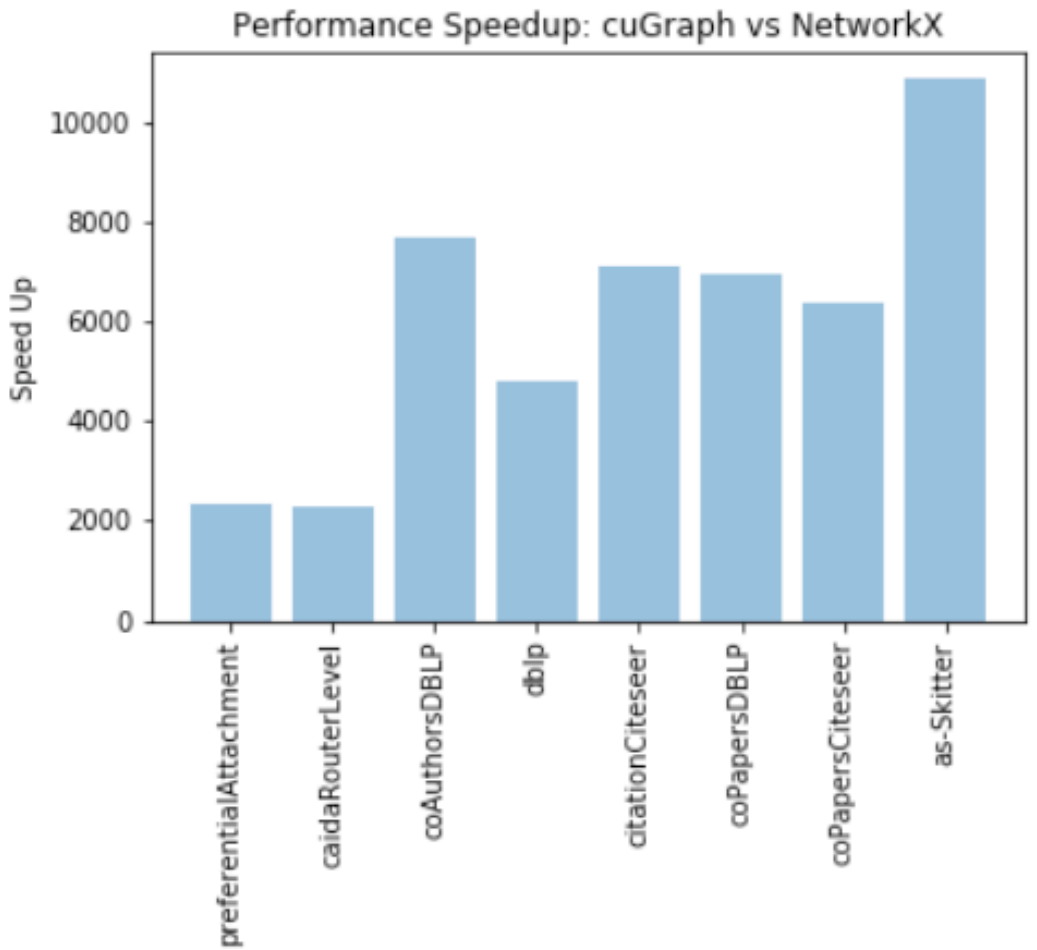
GPUS	RMAT	Nodes	Edges	Runtime
1	25	33,554,432	536,870,912	1.4052
2	26	67,108,864	1,073,741,824	1.3891
4	27	134,217,728	2,122,307,214	1.3891
8	28	268,435,456	4,294,967,296	1.4103
16	29	536,870,912	8,589,934,592	1.4689

Running on V100 32G

LOUVAIN SINGLE RUN

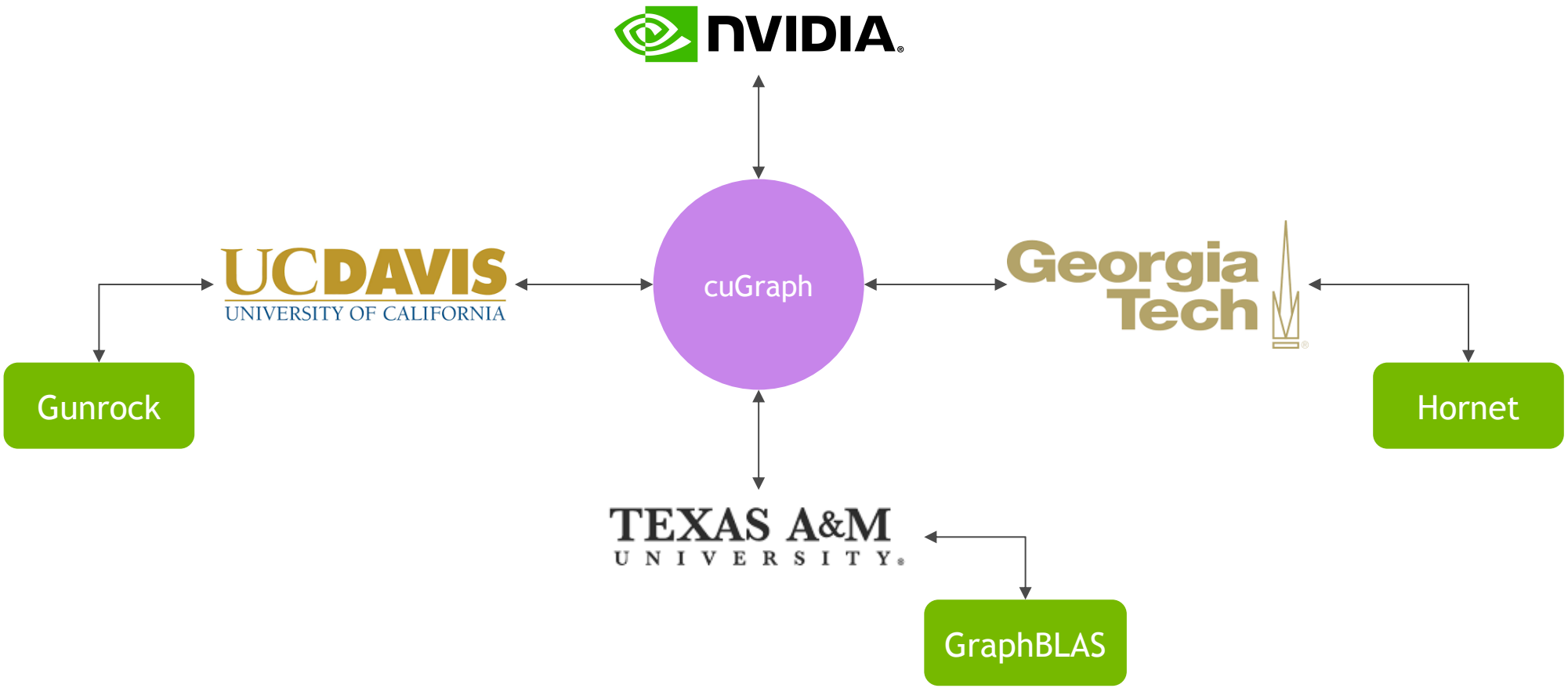
```
G = cugraph.Graph()

G.add_edge_list(gdf["src_0"], gdf["dst_0"], gdf["data"])

df, mod = cugraph.nvLouvain(G)
```

louvain: cudf.DataFrame with two names columns:
louvain["vertex"]: The vertex id.
louvain["partition"]: The assigned partition.



BRINGING IN LEADING RESEARCHERS

Leveraging the great work of others



https://news.developer.nvidia.com/graph-technology-leaders-combine-forces-to-advance-graph-analytics/


Scalable Python and now RAPIDS
DASK

DASK

What is Dask and why does RAPIDS use it for scaling out?

- Dask is a distributed computation scheduler built to scale Python workloads from laptops to supercomputer clusters.
- Extremely modular with scheduling, compute, data transfer, and out-of-core handling all being disjointed allowing us to plug in our own implementations.
- Can easily run multiple Dask workers per node to allow for an easier development model of one worker per GPU regardless of single node or multi node environment.

DASK

Scale up and out with cuDF

- Use cuDF primitives underneath in map-reduce style operations with the same high level API
- Instead of using typical Dask data movement of pickling objects and sending via TCP sockets, take advantage of hardware advancements using a communications framework called OpenUCX:
  - For intranode data movement, utilize NVLink and PCIe peer-to-peer communications
  - For internode data movement, utilize GPU RDMA over Infiniband and RoCE

http://www.openucx.org/

https://github.com/rapidsai/dask-cudf

LET'S START BUILD ANALYTICS USING RAPIDS