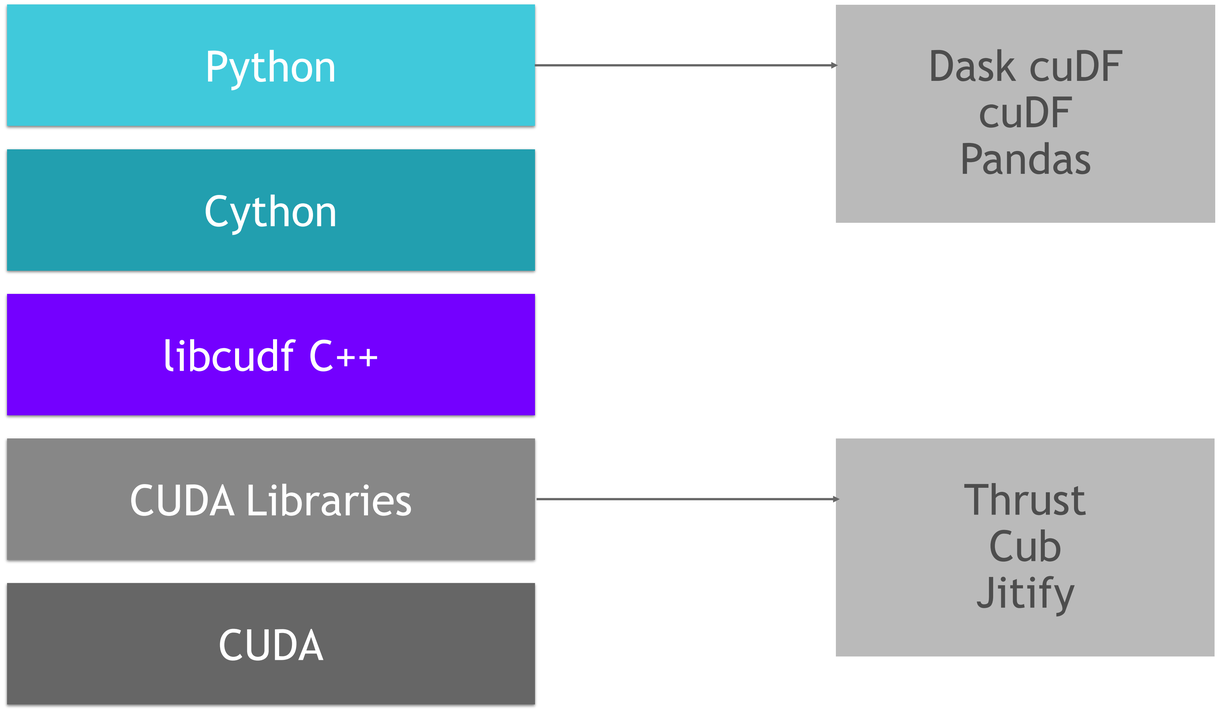

NVIDIA
CUDF

What is cuDF?

Expandable platform for GPU data science



- Familiar pandas-like Python API
- Table (dataframe) and column types and algorithms
- High-performance C++ layer provides GPU-optimized CUDA kernels, data types, operations, and primitives
- CUDA/C++ is top level supported and used by many for integrating RAPIDS

ETL - THE BACKBONE OF DATA SCIENCE

cuDF is…

Python library

- 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

In [2]:
```
#Read in the data. Notice how it decompresses as it reads the data into memory.
gdf = cudf.read_csv('/rapids/Data/black-friday.zip')
```

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

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Ca
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

In [6]:
```
#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).stoi()
```

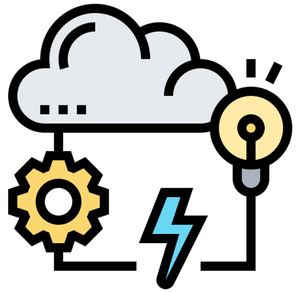
In [7]:
```
#Here we can see how we can control what the value of our dummies with the replace method and turn
strings to ints
gdf['City_Category'] = gdf.City_Category.str.replace('A', '1')
gdf['City_Category'] = gdf.City_Category.str.replace('B', '2')
gdf['City_Category'] = gdf.City_Category.str.replace('C',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

cuDF

Features Today

Datatypes

- Int, float, time, duration, String, Categorical, List, Struct, Dictionary, and Decimal types
- Time Series
- Numpy, CuPy, Numba
- Fixed point Decimal128



Features

- Dataframe indexing, manipulations
- Null operations
- Aggregations, Sort, Merge, Groupby
- Join, Conditional Joins
- Abstract Syntax Tree Evaluation
- User Defined Functions
- Rolling window operations

IO

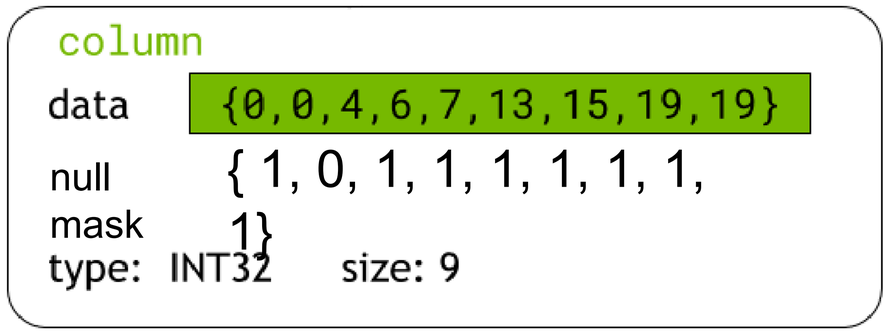
- cuIO- CSV, JSON, Parquet, ORC, AVRO, HDF, Feather
- GPU Direct Storage Support – Avro, Parquet, ORC, CSV
- Interop - Pytorch, TF, CuPy, mxnet
- CUDA 11.5 and CUDA Enhanced Compatibility

LIBCUDF

CUDA C++ data analytics library

- Algorithms using CUDA, **thrust** and **cub** libraries.
- column, table, scalar.
- Datatypes
- int 8,16,32,64 – signed, unsigned
- Float, double
- Decimal 32, 64, 128
- Strings
- Lists, Structs, Dictionary

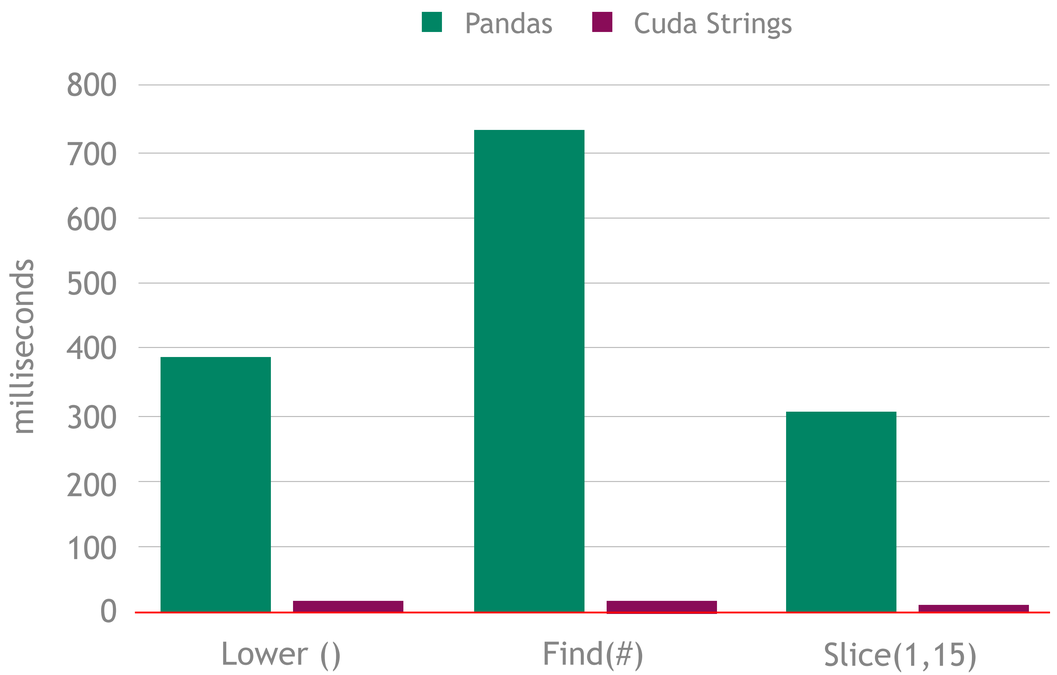
{ 0, NULL, 4, 6, 7, 13, 15, 19, 19 }



STRING SUPPORT

CURRENT STRING SUPPORT

- Regular Expressions
- Element-wise operations
  - Split, Find, Extract, Cat, Typecasting, etc…
- String GroupBys, Joins, Sorting, etc.
- Categorical columns fully on GPU
- NLP Preprocessors
  - Tokenizers, Normalizers, Edit Distance, Porter Stemmer, etc.
- JIT-compiled String UDFs
- Character parallel algorithms for long strings
- Further performance optimization



TIME SERIES FUNCTIONS

cuDF masters the fourth dimension

- API additions for convenient time-series analysis: date_range() for timestamp generation, interpolate() for fast linear interpolation
- Grouping by a time frequency is now supported
- Upsampling and downsampling of time-series data via resample()
- Now calendar-aware! Functions like isocalendar(), quarter(), dayofweek() now available. DateOffsets with non-fixed frequencies like month and year supported.

```
>>> df
                   ts  value
0 2000-01-01 00:00:02      1
1 2000-01-01 00:00:07      2
2 2000-01-01 00:00:02      3
3 2000-01-01 00:00:15      4
4 2000-01-01 00:00:05      5
5 2000-01-01 00:00:09      6

>>> grouper = cudf.Grouper(key="ts", freq="4s")
>>> df.groupby(grouper).mean()

                     value
ts
2000-01-01 00:00:00    2.0
2000-01-01 00:00:04    3.5
2000-01-01 00:00:08    6.0
2000-01-01 00:00:12    4.0
```