Scaling RAPIDS with Dask

Matthew Rocklin, Systems Software Manager
GTC San Jose 2019

PyData is Pragmatic, but Limited

How do we accelerate an existing software stack?

The PyData Ecosystem

- NumPy: Arrays
- Pandas: Dataframes
- Scikit-Learn: Machine Learning
- Jupyter: Interaction
- … (many other projects)

Is well loved

- Easy to use
- Broadly taught
- Community Governed

But sometimes slow

- Single CPU core
- In-memory data

95% of the time, PyData is great

(and you can ignore the rest of this talk)

5% of the time, you want more performance

Scale up and out with RAPIDS and Dask

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale Up / Accelerate

Scale out / Parallelize

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Scale out / Parallelize

RAPIDS: GPU variants of PyData libraries

- **NumPy -> CuPy, PyTorch, TensorFlow**
  - Array computing
  - Mature due to deep learning boom
  - Also useful for other domains
  - Obvious fit for GPUs
- **Pandas -> cuDF**
  - Tabular computing
  - New development
  - Parsing, joins, groupbys
  - Not an obvious fit for GPUs
- **Scikit-Learn -> cuML**
  - Traditional machine learning
  - Somewhere in between

RAPIDS: GPU variants of PyData libraries

- **NumPy -> CuPy, PyTorch, TensorFlow**
  - Array computing
  - Mature due to deep learning boom
  - Also useful for other domains
  - Obvious fit for GPUs
- **Pandas -> cuDF**
  - Tabular computing
  - New development
  - Parsing, joins, groupbys
  - Not an obvious fit for GPUs
- **Scikit-Learn -> cuML**
  - Traditional machine learning
  - Somewhere in between

```
1]: import pandas, cudf

2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))

CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s

2]: 12748986

3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))

CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s
Wall time: 2.12 s

3]: 12748986

4]: !du -hs data/nyc/yellow_tripdata_2015-01.csv

1.9G    data/nyc/yellow_tripdata_2015-01.csv
```

RAPIDS: GPU variants of PyData libraries

- **NumPy -> CuPy, PyTorch, TensorFlow**
  - Array computing
  - Mature due to deep learning boom
  - Also useful for other domains
  - Obvious fit for GPUs
- **Pandas -> cuDF**
  - Tabular computing
  - New development
  - Parsing, joins, groupbys
  - Not an obvious fit for GPUs
- **Scikit-Learn -> cuML**
  - Traditional machine learning
  - Somewhere in between

[9]:
```
%%time
record_data = (('fea%d'%i, data[:,i]) for i in range(data.shape[1]))
gdf = cudf.DataFrame(record_data)
```
```
CPU times: user 4.14 s, sys: 4.2 s, total: 8.34 s
Wall time: 9.7 s
```

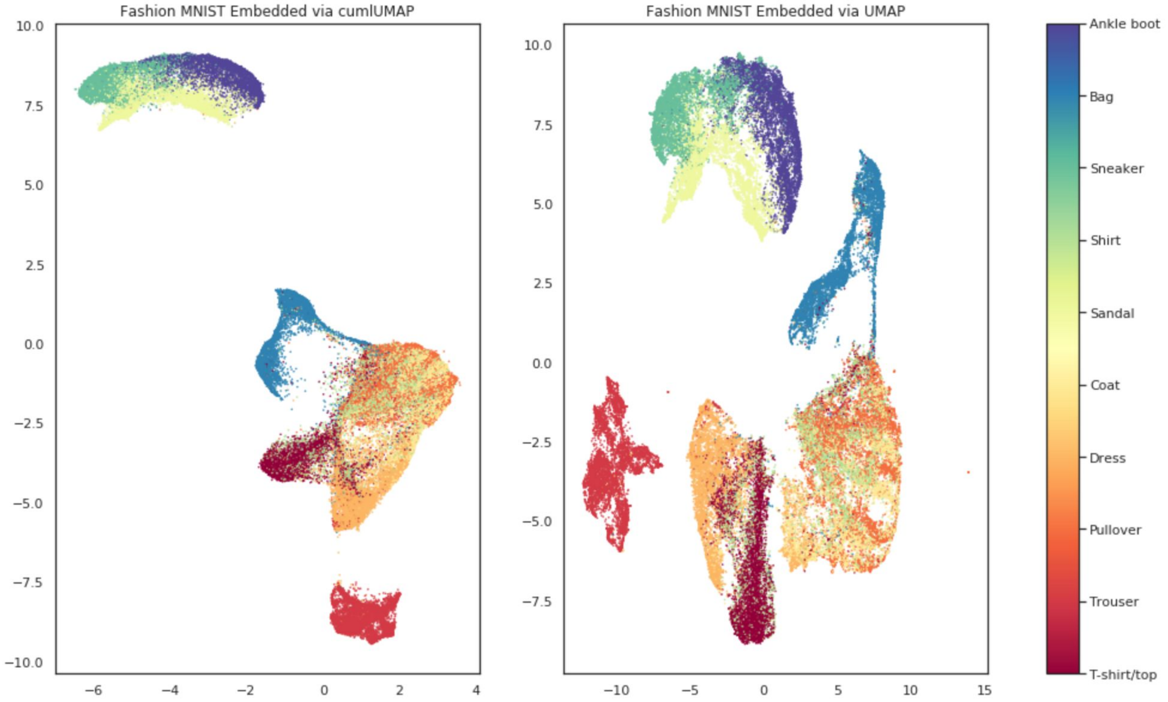
[10]:
```
%%time
embedding = umap.UMAP(n_neighbors=5, init="spectral").fit_transform(data)
```
```
CPU times: user 4min 34s, sys: 1min 27s, total: 6min 2s
Wall time: 1min 49s
```

[11]:
```
%%time
g_embedding = cumlUMAP(n_neighbors=5, init="spectral").fit_transform(gdf)
```
```
CPU times: user 50.9 s, sys: 0 ns, total: 50.9 s
Wall time: 19.5 s
```

[12]:
```
print(f'Size of data in memory: {data.nbytes / 1e6} MB')
```
```
Size of data in memory: 439.04 MB
```

RAPIDS: GPU variants of PyData libraries

- **NumPy -> CuPy, PyTorch, TensorFlow**
  - Array computing
  - Mature due to deep learning boom
  - Also useful for other domains
  - Obvious fit for GPUs
- **Pandas -> cuDF**
  - Tabular computing
  - New development
  - Parsing, joins, groupbys
  - Not an obvious fit for GPUs
- **Scikit-Learn -> cuML**
  - Traditional machine learning
  - Somewhere in between



Scale up and out with RAPIDS and Dask

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale Up / Accelerate

Scale out / Parallelize

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale out / Parallelize

Dask Parallelizes PyData

Natively

- **PyData Native**
  - Built on top of NumPy, Pandas Scikit-Learn, … (easy to migrate)
  - With the same APIs (easy to train)
  - With the same developer community (well trusted)
- **Scales**
  - Scales out to thousand-node clusters
  - Easy to install and use on a laptop
- **Popular**
  - Most common parallelism framework today at PyData and SciPy conferences
- **Deployable**
  - HPC: SLURM, PBS, LSF, SGE
  - Cloud: Kubernetes
  - Hadoop/Spark: Yarn

Parallel NumPy

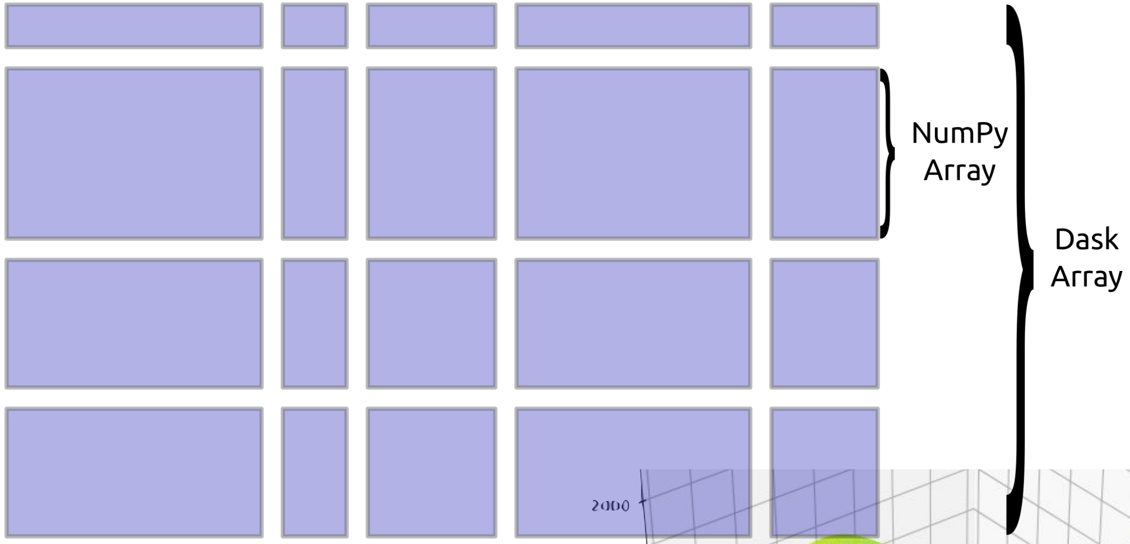
For imaging, simulation analysis, machine learning

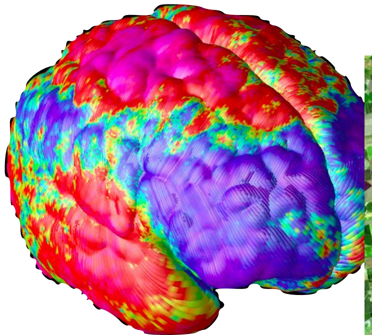
- Same API as NumPy

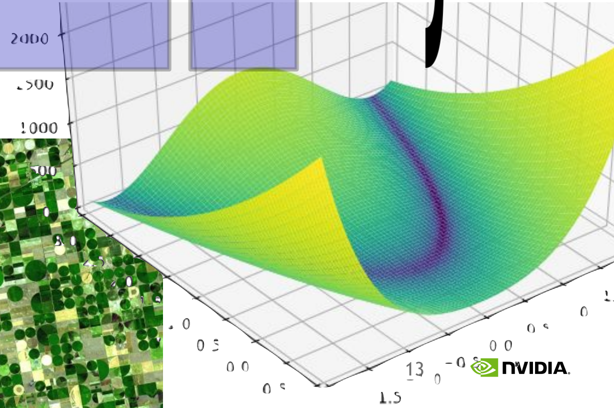
```
import dask.array as da
x = da.from_hdf5(...)
x + x.T - x.mean(axis=0)
```

- One Dask Array is built from many NumPy arrays

  Either lazily fetched from disk
  Or distributed throughout a cluster







Parallel Pandas

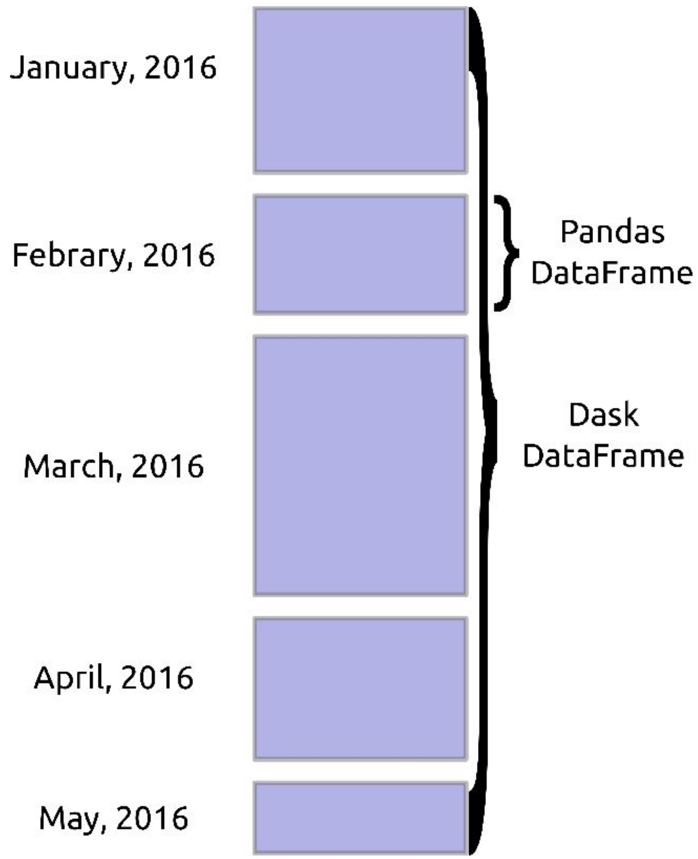
For ETL, time series, data munging

- Same API as Pandas

```
import dask.dataframe as dd
df = dd.read_csv(...)
df.groupby('name').balance.max()
```

- One Dask DataFrame is built from many Pandas DataFrames

  Either lazily fetched from disk
  Or distributed throughout a cluster

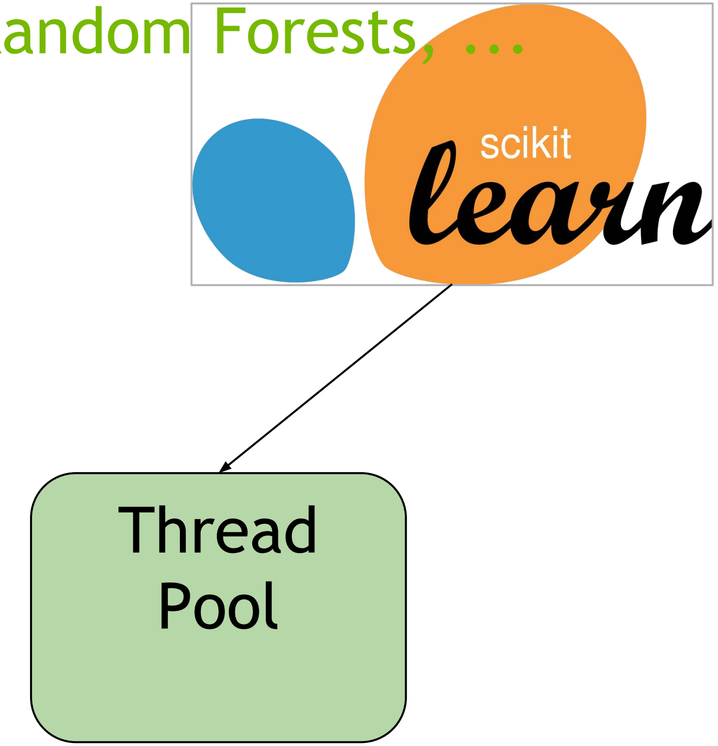


Parallel Scikit-Learn

For Hyper-Parameter Optimization, Random Forests, ...

- Same API

```
estimator = RandomForest()
estimator.fit(data, labels)
```



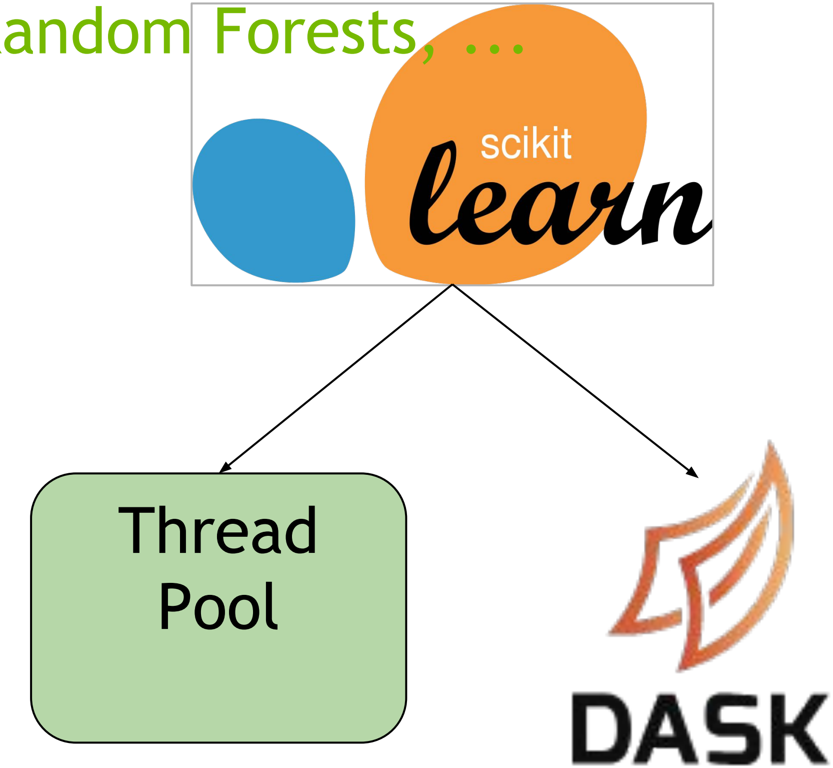
Parallel Scikit-Learn

For Hyper-Parameter Optimization, Random Forests, ...

- Same API

```
from scikit_learn.externals import joblib
with joblib.parallel_backend('dask'):
    estimator = RandomForest()
    estimator.fit(data, labels)
```

- Same exact code, just wrap with a decorator
- Replaces default threaded execution with Dask
  Allowing scaling onto clusters
- Available in most Scikit-Learn algorithms where joblib is used



NVIDIA.

Parallel Python

For custom systems, ML algorithms, workflow engines

- Parallelize existing codebases

```
results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)
```

Parallel Python

For custom systems, ML algorithms, workflow engines

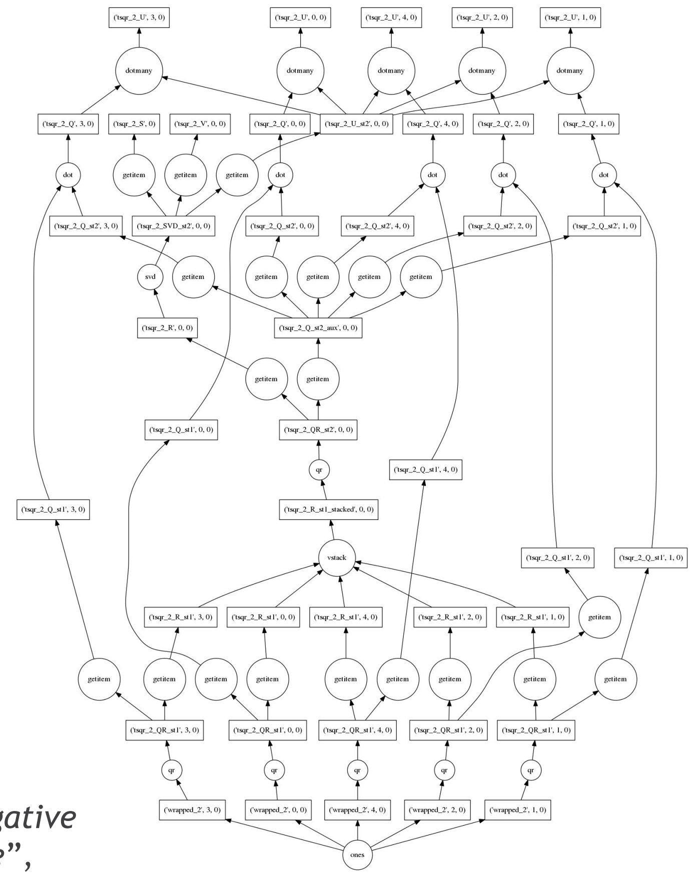
- Parallelize existing codebases

```
f = dask.delayed(f)
g = dask.delayed(g)

results = {}

for x in X:
  for y in Y:
    if x < y:
      result = f(x, y)
    else:
      result = g(x, y)
    results.append(result)

result = dask.compute(results)
```



M Tepper, G Sapiro "*Compressed nonnegative matrix factorization is fast and accurate*", IEEE Transactions on Signal Processing, 2016

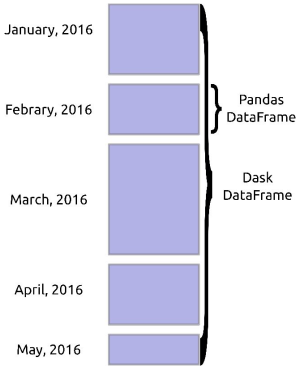
Dask Connects Python users to Hardware

User

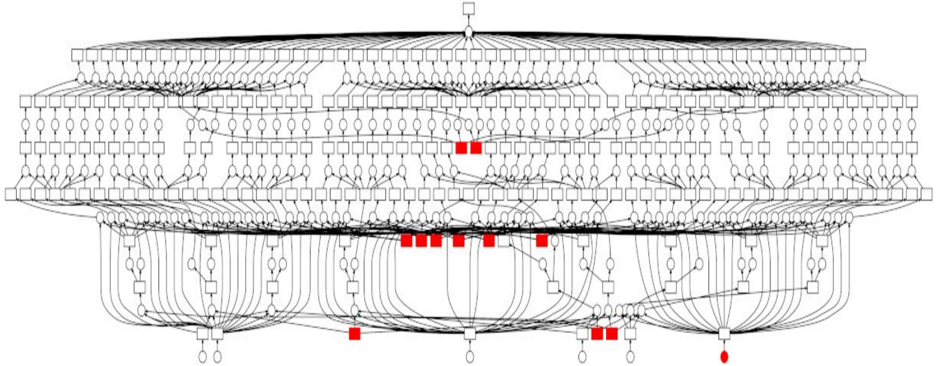
Execute on distributed hardware

Dask Connects Python users to Hardware

User



Writes high level code (NumPy/Pandas/Scikit-Learn)



Turns into a task graph

Executes on distributed hardware

Example: Dask + Pandas on NYC Taxi

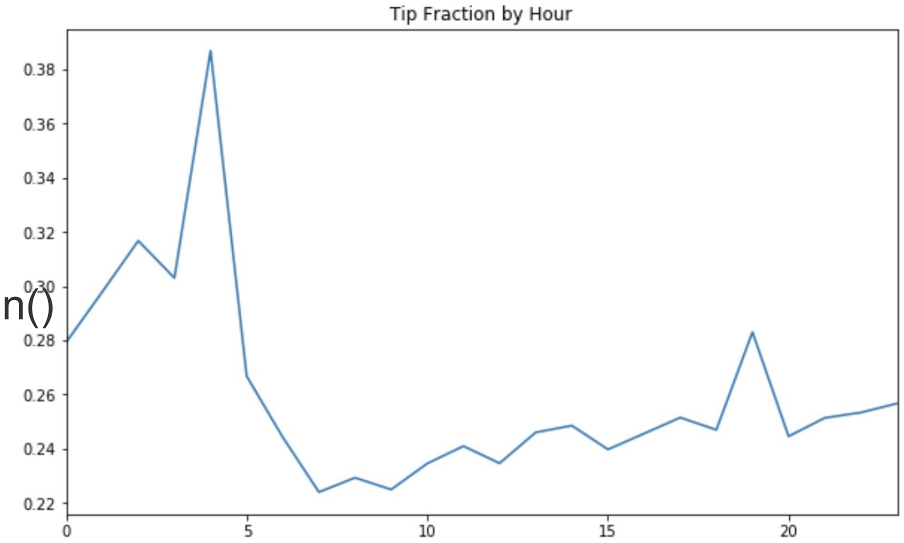
We see how well New Yorkers Tip

```
import dask.dataframe as dd

df = dd.read_csv('gcs://bucket-name/nyc-taxi-*.csv',
                 parse_dates=['pickup_datetime', 'dropoff_datetime'])

df2 = df[(df.tip_amount > 0) & (df.fare_amount > 0)]
df2['tip_fraction'] = df2.tip_amount / df2.fare_amount

hour = df2.groupby(df2.pickup_datetime.dt.hour).tip_fraction.mean()
hour.compute().plot(figsize=(10, 6), title='Tip Fraction by Hour')
```



examples.dask.org

Try live

Dask scales PyData libraries

But is compute-agnostic to those libraries

(A good fit if you’re building a new data science platform)

Scale up and out with RAPIDS and Dask

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale out / Parallelize

Scale up and out with RAPIDS and Dask

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

Dask + RAPIDS

Multi-GPU
On single Node (DGX)
Or across a cluster

PyData

NumPy, Pandas, Scikit-Learn
and many more

Single CPU core
In-memory data

Dask

Multi-core and Distributed PyData

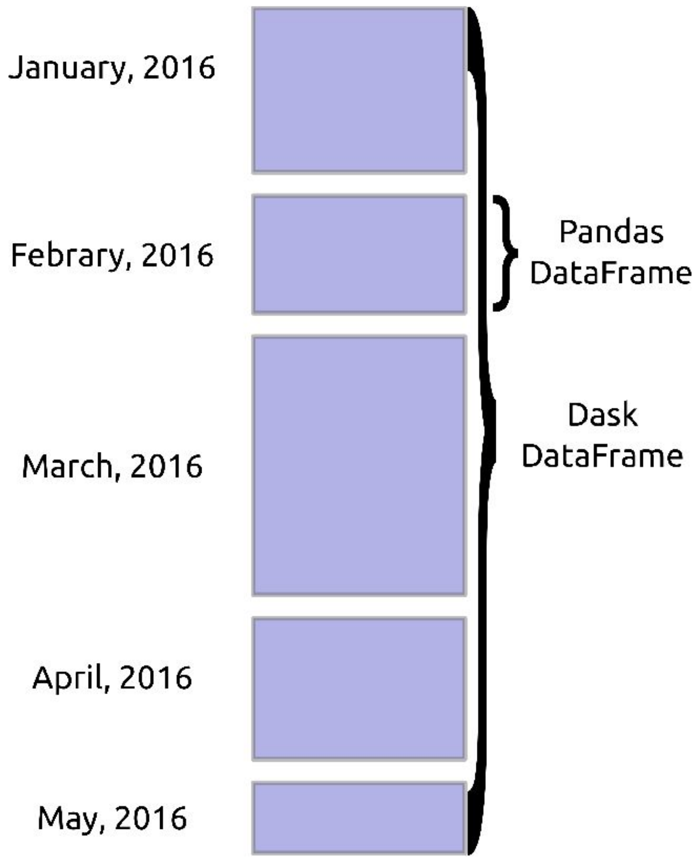
NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale Up / Accelerate

Scale out / Parallelize

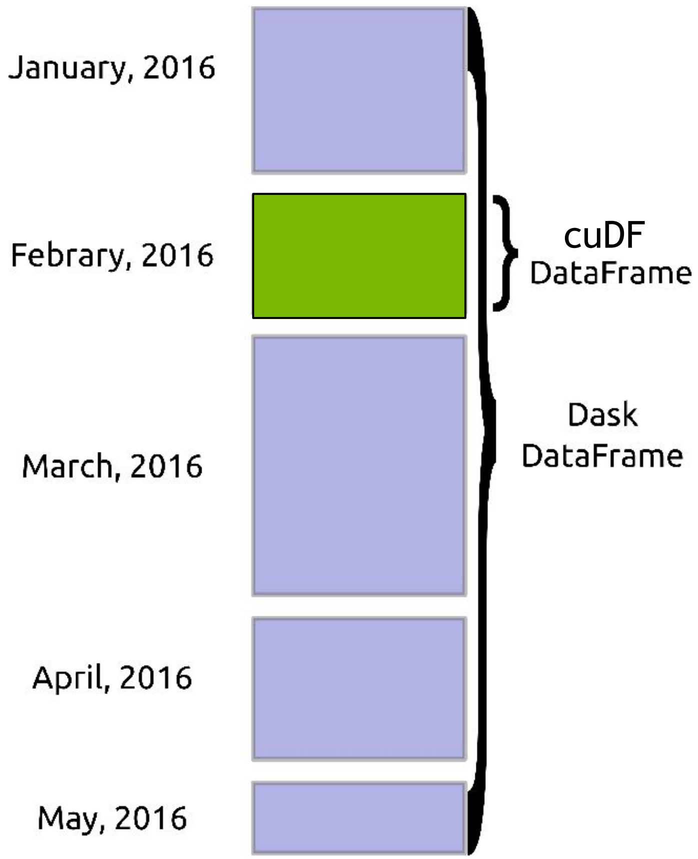
Combine Dask with cuDF

Many GPU DataFrames form a distributed DataFrame



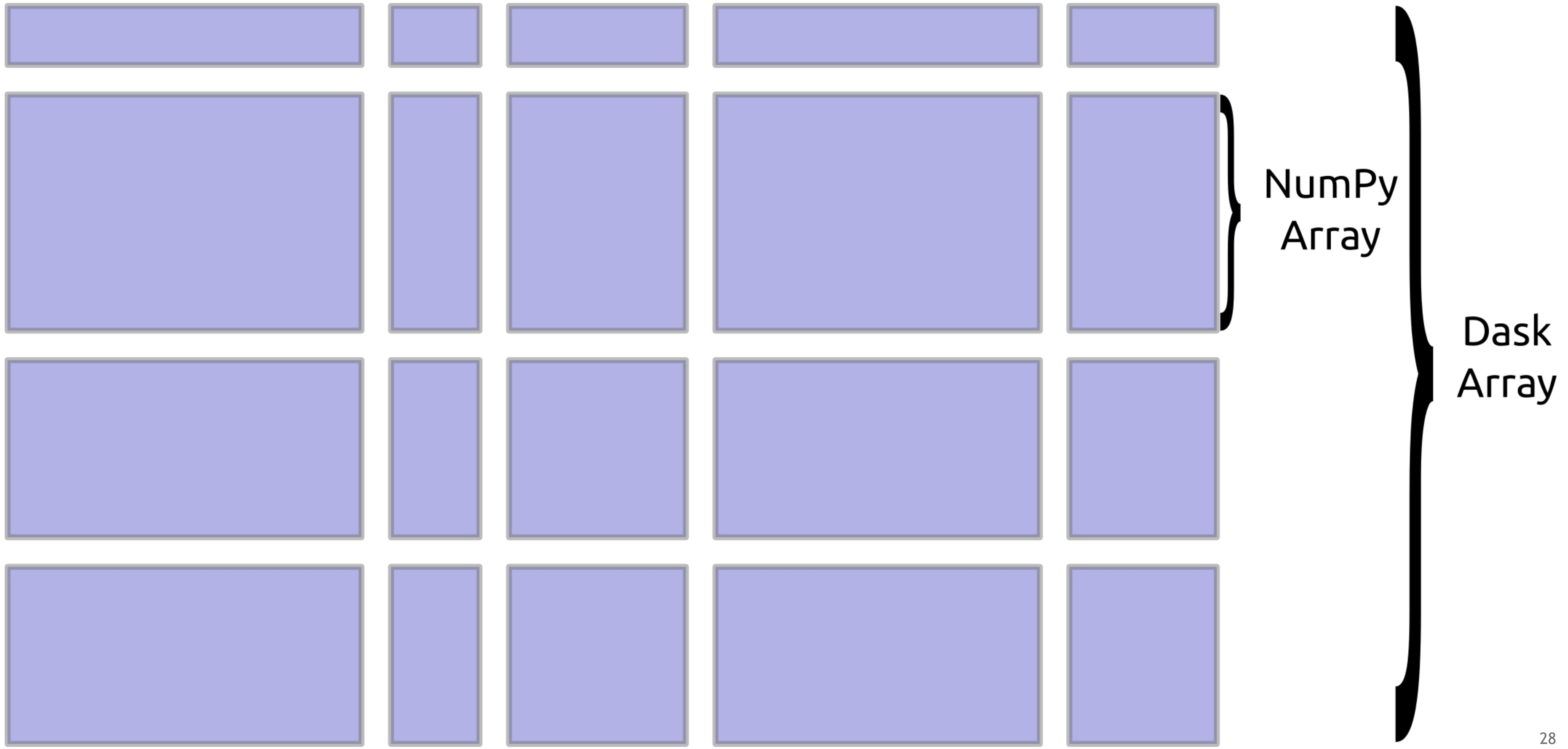
Combine Dask with cuDF

Many GPU DataFrames form a distributed DataFrame



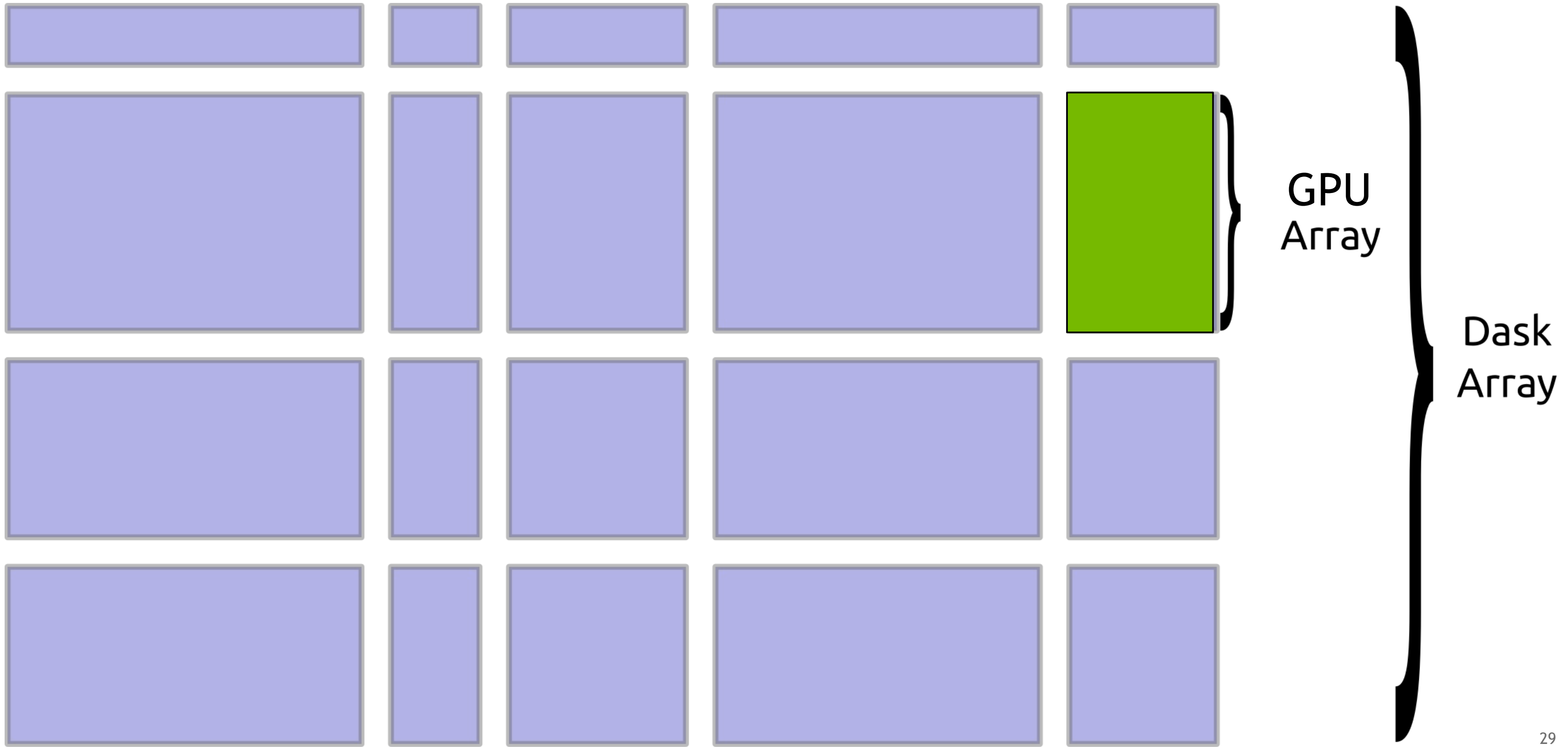
Combine Dask with CuPy

Many GPU arrays form a Distributed GPU array

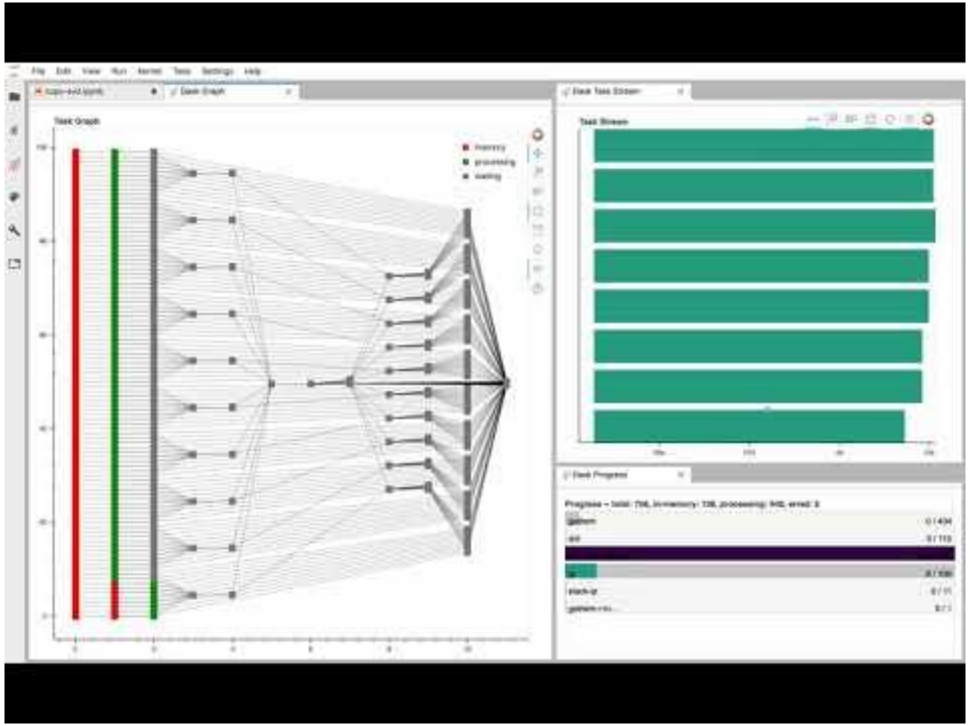


Combine Dask with CuPy

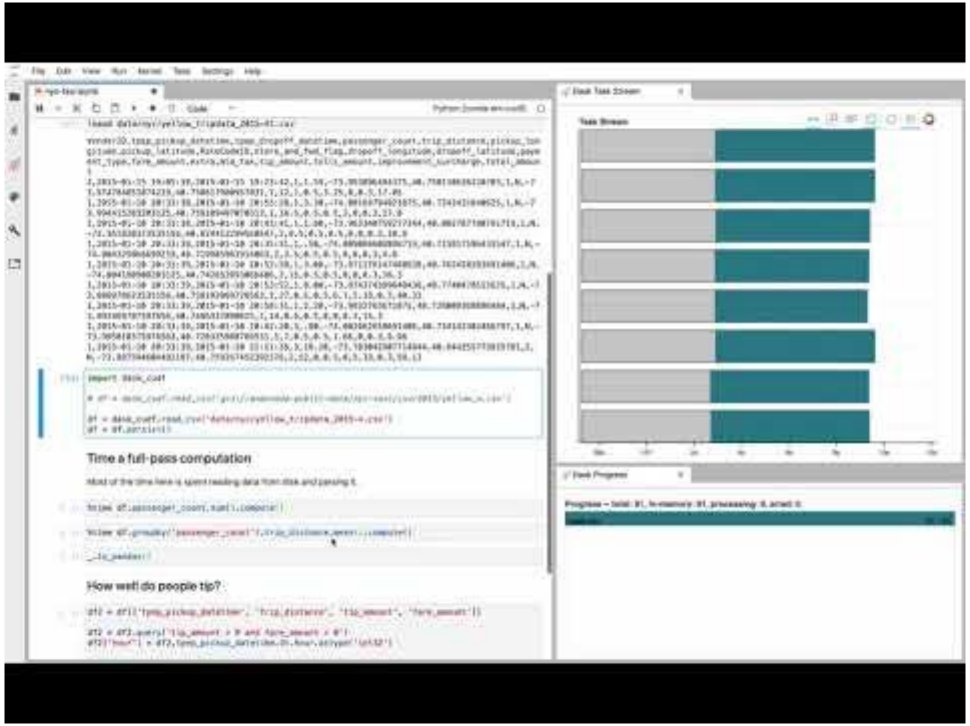
Many GPU arrays form a Distributed GPU array



Experiments



SVD with Dask Array



NYC Taxi with Dask DataFrame

So what works in DataFrames?

Lots!

- Read CSV: `read_csv('s3://bucket/*.csv')`
- Elementwise operations: `df + 1, df['z'] = df.x + df.y`
- Reductions: `df.x.sum()`
- Groupby Aggregations: `df.groupby('x').mean()`
- Joins (hash, sorted, large-to-small): `left.merge(right, on='key'), …`
- …

Leverages Dask DataFrame algorithms (been around for years)
API matches Pandas

So what doesn't work?

Lots!

- Read Parquet/ORC
-
- Reductions: `df.sum()`
- Groupby Aggregations: `df.groupby(['x', 'y']).agg({'z': ['max', 'min']})`
- Rolling window operations
- …

Leverages Dask DataFrame algorithms (been around for years)
API matches Pandas

So what doesn't work?

API Alignment

- When cuDF and Pandas match, existing Dask algorithms work seamlessly
- But the APIs don't always match

```
In [1]: import pandas, cudf

In [2]: cudf.DataFrame.set_index
Out[2]: <function cudf.DataFrame.set_index(self, index)>

In [3]: pandas.DataFrame.set_index
Out[3]: <function pandas.DataFrame.set_index(self, keys, drop=True, append=False,
inplace=False, verify_integrity=False)>
```

So what doesn’t work?

API Alignment

- When cuDF and Pandas match, existing Dask algorithms work seamlessly
- But the APIs don’t always match

```
In [1]: import pandas, cudf

In [2]: pandas.get_dummies                # These are the same

In [3]: cudf.DataFrame.one_hot_encoding  # These are the same
```

So what works in Arrays?

We genuinely don't know yet

- This work is much younger, but moving quickly

- CuPy has been around for a while, and is fairly mature
- Most work today happening upstream in NumPy and Dask

  Thanks Peter Entschev, Hameer Abbasi, Stephan Hoyer, Marten van Kerkwijk, Eric Wieser

- Ecosystem approach benefits other NumPy-like arrays as well, sparse arrays, Xarray, ...

So what's next?

Lots of issues with Dask, too!

- **High Performance Communication**
  - Today Dask uses in-memory or TCP
  - For Infiniband and NVLink, now integrating OpenUCX with ucx-py
- **Spilling to main memory**
  - Today Dask spills from memory to disk
  - For GPUs, we'd like to spill from device, to host, to disk
- **Mixing CPU and GPU workloads**
  - Today Dask has one thread per core, or one thread per GPU
  - For mixed systems we need to auto-annotate GPU vs CPU tasks
- **Better recipes for deployment**
  - Today Dask deploys on Kubernetes, HPC job schedulers, YARN
  - Today these technologies also support GPU workloads
  - Need better examples using both together

Learn More

Thank you for your time

PyData: pydata.org

RAPIDS: rapids.ai

Dask: dask.org

examples.dask.org