Scaling Python with Dask

Matthew Rocklin

© 2018 Anaconda, Inc. - Confidential & Proprietary

DASK

A library for parallel computing in Python

- **Parallelizes** libraries like NumPy, Pandas, and Scikit-Learn
- **Scales** from a personal laptop to large clusters
- **Integrates** easily into existing Python codes
- **Adapts** to custom and sophisticated algorithms

© 2018 Anaconda, Inc. - Confidential & Proprietary

The Numeric Python Ecosystem

Capabilities

- **Arrays:** Numpy
- **Dataframes:** Pandas
- **Machine Learning:** Scikit-Learn, …
- **Interaction:** Jupyter
- **Natural Language:** NLTK, Spacy, Gensim
- **Visualization:** Matplotlib, Bokeh, Altair
- … thousands of other packages
- … web, networking, concurrency
- … mathematics, image processing

Jake Vanderplas PyCon 2017 Keynote

ANACONDA.
© 2018 Anaconda, Inc. - Confidential & Proprietary

The Numeric Python Ecosystem

Strengths

- **Easy** to use, popular in education
- **Fast** with both efficient code (C/Fortran) and sophisticated algorithms
- **Applied** across a wide set of industries

Weaknesses

- Designed for **in-memory data**
- Designed for **single-core execution**

Jake Vanderplas PyCon 2017 Keynote

ANACONDA.
© 2018 Anaconda, Inc. - Confidential & Proprietary

The Numeric Python Ecosystem

Strengths

- **Easy** to use, popular in education
- **Fast** with both efficient code (C/Fortran) and sophisticated algorithms
- **Applied** across a wide set of industries

- Works on **larger-than-memory** data
- Parallelizes across **multi-core** workstations or **distributed** clusters

Jake Vanderplas PyCon 2017 Keynote

© 2018 Anaconda, Inc. - Confidential & Proprietary

DASK

A library for parallel computing in Python

- **Parallelizes** libraries like NumPy, Pandas, and Scikit-Learn
- **Scales** from a personal laptop to large clusters
- **Integrates** easily into existing Python codes
- **Adapts** to custom and sophisticated algorithms

© 2018 Anaconda, Inc. - Confidential & Proprietary

High Level:
Parallel Numpy and Pandas

Low Level:
Parallelize custom code with task scheduling

© 2018 Anaconda, Inc. - Confidential & Proprietary

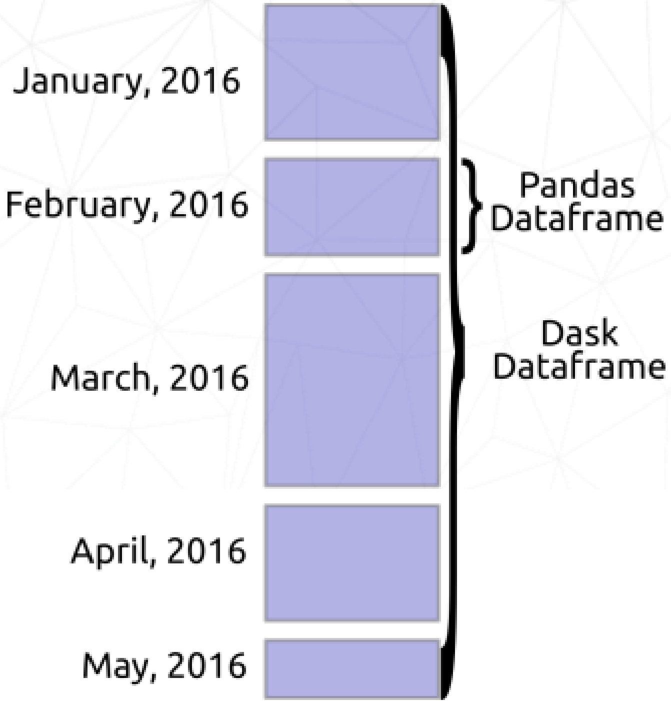
Dask Dataframe

- Scalable implementation of Pandas

  Same API (subset)
  Larger datasets

- ```
  import pandas as pd
  df = pd.read_csv('my-file.csv')
  df.groupby('name').value.mean()
  ```

- Internally coordinates many Pandas dataframes
- Supports reductions, groupbys, joins, timeseries, …
- Co-evolves with Pandas



© 2018 Anaconda, Inc. - Confidential & Proprietary
  ```

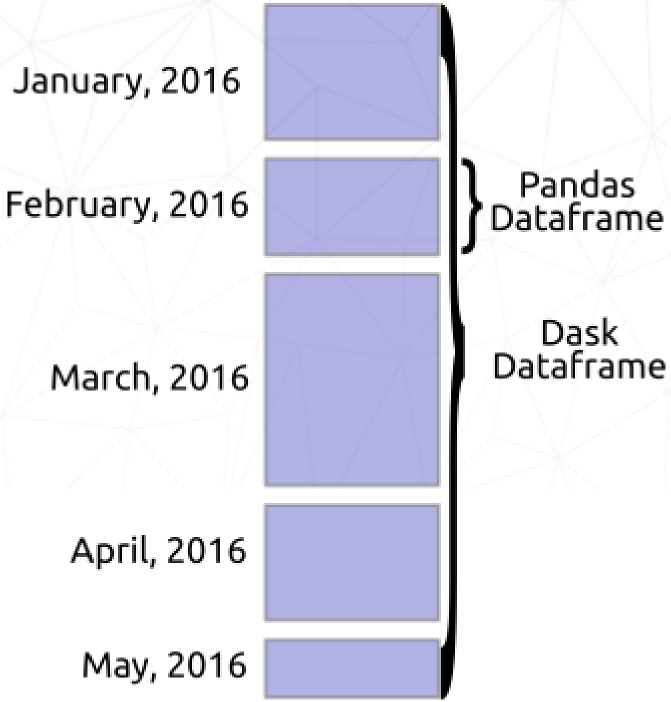
Dask Dataframe

- Scalable implementation of Pandas

  Same API (subset)
  Larger datasets

- ```
  import dask.dataframe as dd
  df = dd.read_csv(‘s3://bucket/*.csv')
  df.groupby(‘name’).value.mean().compute()
  ```

- Internally coordinates many Pandas dataframes
- Supports reductions, groupbys, joins, timeseries, …
- Co-evolves with Pandas



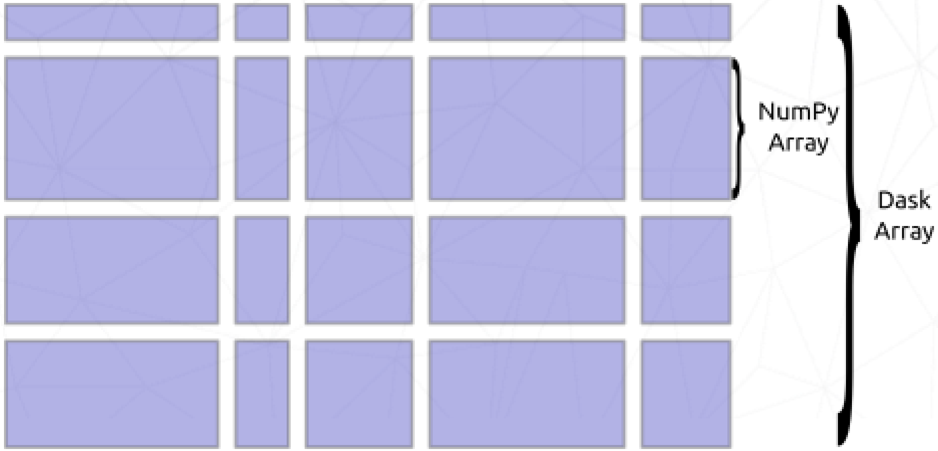
© 2018 Anaconda, Inc. - Confidential & Proprietary
  ```

Dask Array

- Scalable implementation of Numpy

  Same API (subset)
  Larger datasets

- ```
  import dask.array as da
  x = da.random.random(…)
  y = x.dot(x.T) - x.mean(axis=0)
  ```



- Internally coordinates many Numpy arrays
- Supports reductions, blockwise, overlapping, linear algebra
- Co-evolves with Numpy

© 2018 Anaconda, Inc. - Confidential & Proprietary
  ```

Demonstration with Dask dataframe

© 2018 Anaconda, Inc. - Confidential & Proprietary

Parallelizing custom Python code with Dask core

© 2018 Anaconda, Inc. - Confidential & Proprietary

Parallelize Custom Python Code

- Not all problems are big arrays or dataframes
- Often need fine-grained control
- How would we parallelize this code?

```
for x in A:
    for y in B:
        if x < y:
            z = f(x, y)
        else:
            z = g(x, y)
        results.append(z)
```

- Shouldn't need to rewrite to get parallelism

© 2018 Anaconda, Inc. - Confidential & Proprietary

Parallelize Custom Python Code

Dask records individual function calls in a task graph

```
def inc(x):
    return x + 1


def add(x, y):
    return x + y

>>> a = inc(1)
>>> b = inc(2)
>>> c = add(a, b)
>>> c
5
```


© 2018 Anaconda, Inc. - Confidential & Proprietary

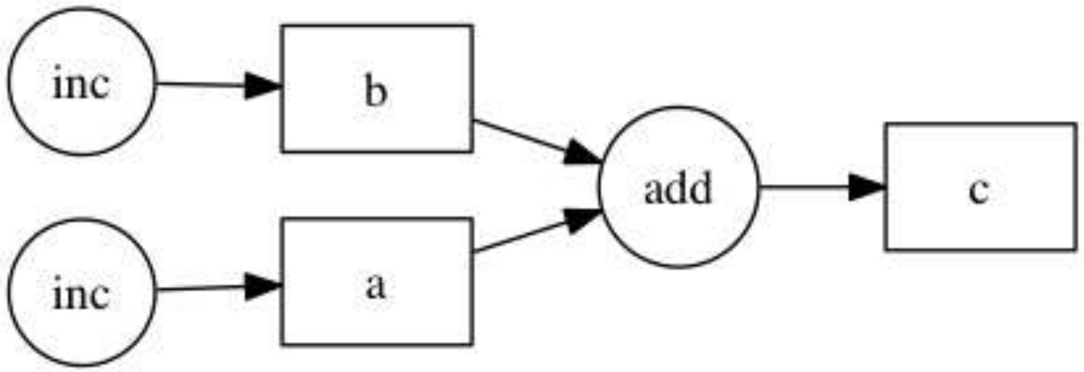
Parallelize Custom Python Code

Dask records individual function calls in a task graph

```
def inc(x):
    return x + 1


def add(x, y):
    return x + y

>>> a = inc(1)
>>> b = inc(2)
>>> c = add(a, b)
>>> c
5
```



© 2018 Anaconda, Inc. - Confidential & Proprietary

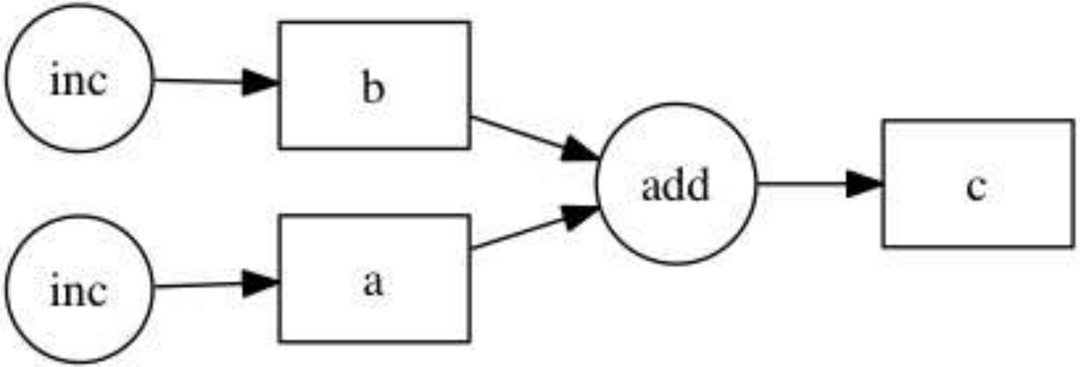
Parallelize Custom Python Code

Dask records individual function calls in a task graph

```
@dask.delayed
def inc(x):
    return x + 1

@dask.delayed
def add(x, y):
    return x + y

>>> a = inc(1)
>>> b = inc(2)
>>> c = add(a, b)
>>> c
<Delayed value>
```



© 2018 Anaconda, Inc. - Confidential & Proprietary

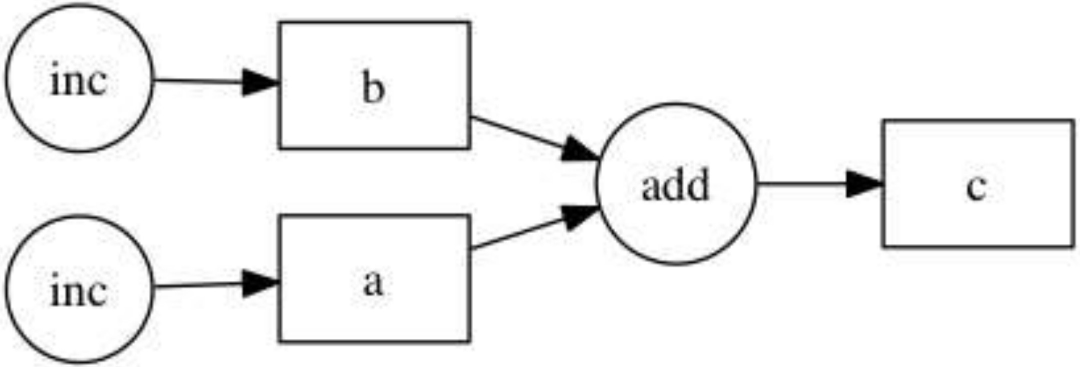
Parallelize Custom Python Code

Dask records individual function calls in a task graph

```
@dask.delayed
def inc(x):
    return x + 1

@dask.delayed
def add(x, y):
    return x + y

>>> a = inc(1)
>>> b = inc(2)
>>> c = add(a, b)
>>> c.compute()
5
```

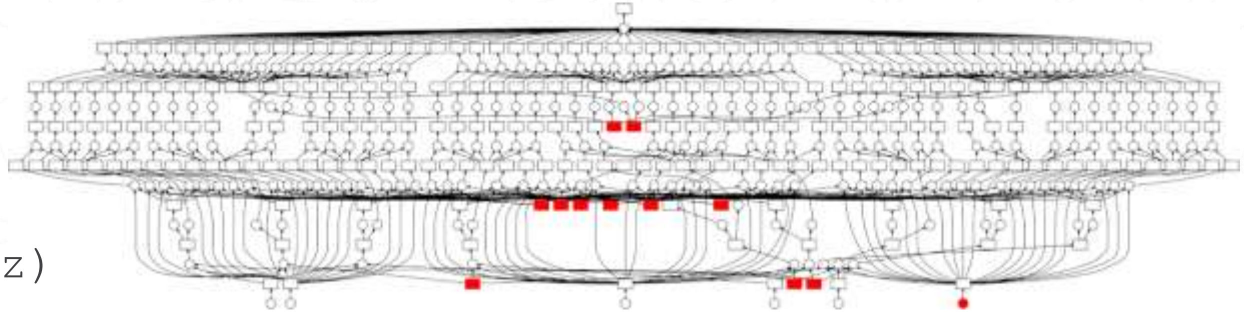


© 2018 Anaconda, Inc. - Confidential & Proprietary

Parallelize Custom Python Code

Dask records individual function calls in a task graph

```
for x in A:
  for y in B:
    if x < y:
      z = f(x, y)
    else:
      z = g(x, y)
    results.append(z)
```



- Removes barriers to parallelism, used more broadly
- More everyday use of the cluster
- Less rewriting of existing systems

© 2018 Anaconda, Inc. - Confidential & Proprietary

Demonstration with Task scheduling

© 2018 Anaconda, Inc. - Confidential & Proprietary

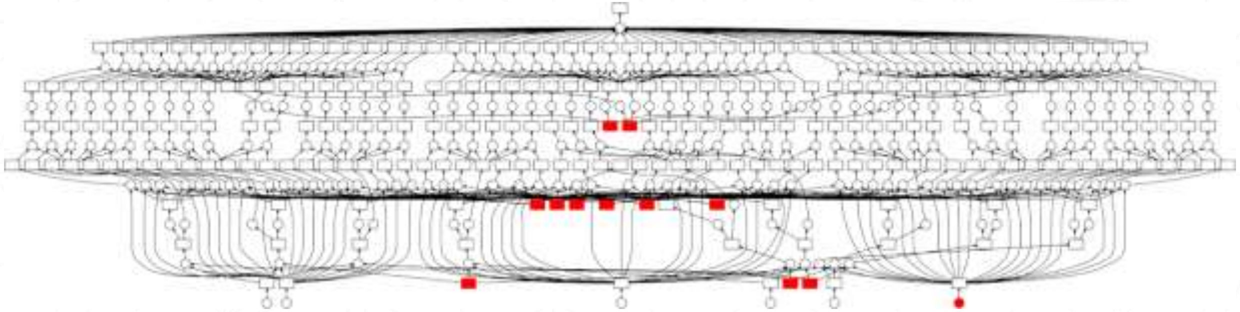
High Level:
Parallel Numpy and Pandas

Low Level:
Parallelize custom code with task scheduling

© 2018 Anaconda, Inc. - Confidential & Proprietary

How does Dask work?

- Dask APIs produce task graphs
- Dask Schedulers execute task graphs



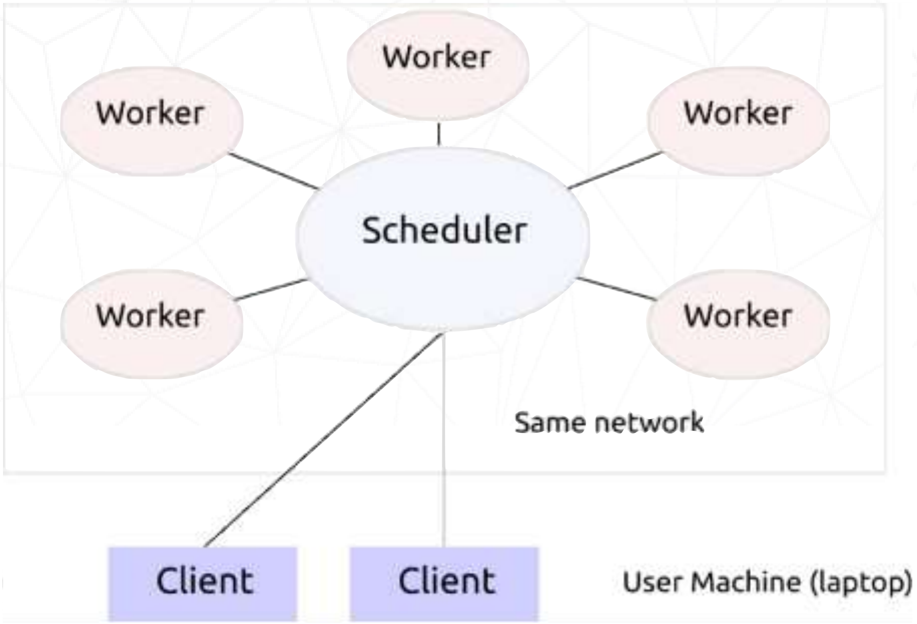
- Dask schedulers can run either …
  - In a local thread or process pool (lightweight)
  - On a distributed cluster (scalable)

© 2018 Anaconda, Inc. - Confidential & Proprietary

Dask on a cluster

- Three components
  - Scheduler: centralized metadata
  - Workers: distributed work and storage
  - Clients: like a Jupyter notebook

- Clients interface with users
  Handle APIs like dask.dataframe, …
- Workers do computation
  hold data in distributed memory
  communicate peer-to-peer
- Scheduler coordinates everyone together



© 2018 Anaconda, Inc. - Confidential & Proprietary

Dask on a cluster

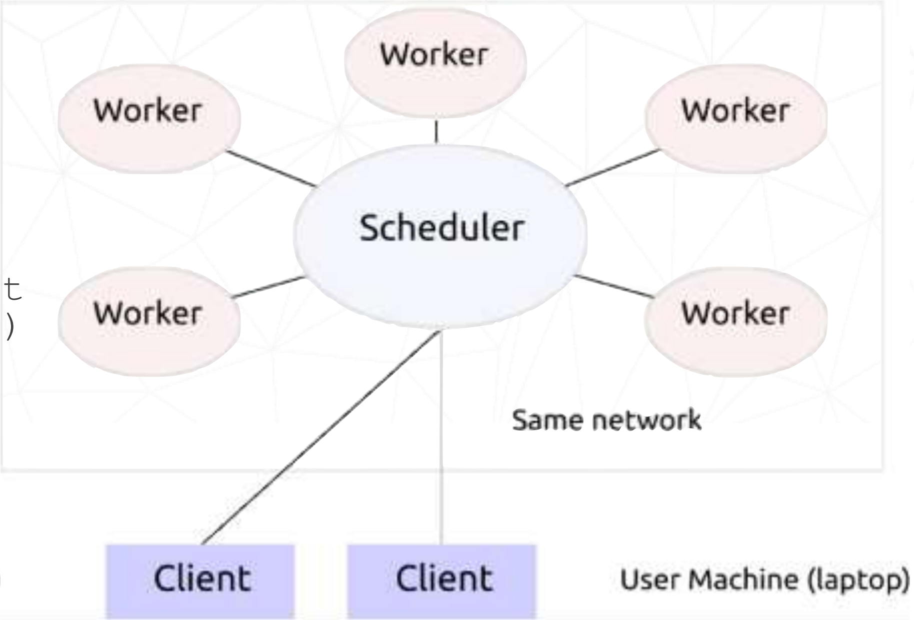
- Manual Setup

```
$ dask-scheduler
Scheduler at 192.168.0.1:8786

$ dask-worker 192.168.0.1:8786
$ dask-worker 192.168.0.1:8786
$ dask-worker 192.168.0.1:8786

>>> from dask.distributed import Client
>>> client = Client('192.168.0.1:8786')
```

- Automatic Setup
  - dask-kubernetes
  - dask-yarn
  - dask-jobqueue (PBS, SLURM, LSF, …)
  - dask-ssh
  - …
- https://dask.pydata.org/en/latest/setup.html



© 2018 Anaconda, Inc. - Confidential & Proprietary

Dask on a laptop

- Dask also works fine without any setup

```
import dask.dataframe as dd
df = dd.read_parquet('my-data.parquet')
result = df.groupby(df.name).balance.sum()

result.compute()   # uses local threads
```

- Uses a local thread pool by default

  Easy to get started

  Shared memory, very low overhead

ANACONDA. © 2018 Anaconda, Inc. - Confidential & Proprietary

Dask on a Cluster

- Dask also works fine without any setup

```
import dask.dataframe as dd
df = dd.read_parquet('my-data.parquet')
result = df.groupby(df.name).balance.sum()
client = dask.distributed.Client('…')
result.compute()   # uses local threads
```

- Now uses a distributed cluster

  Easy to scale out when necessary

  Cost of switching is low

© 2018 Anaconda, Inc. - Confidential & Proprietary

Reasons people choose Dask

© 2018 Anaconda, Inc. - Confidential & Proprietary

1. Familiar APIs

```
import pandas as pd
df = pd.read_csv(‘/path/to/my-file.csv’)

import dask.dataframe as dd
df = dd.read_csv(‘hdfs://path/to/my-files-
*.csv')
```


© 2018 Anaconda, Inc. - Confidential & Proprietary

2. Scales Up

- Cloud friendly
- HPC friendly
- Scales to thousands of machines

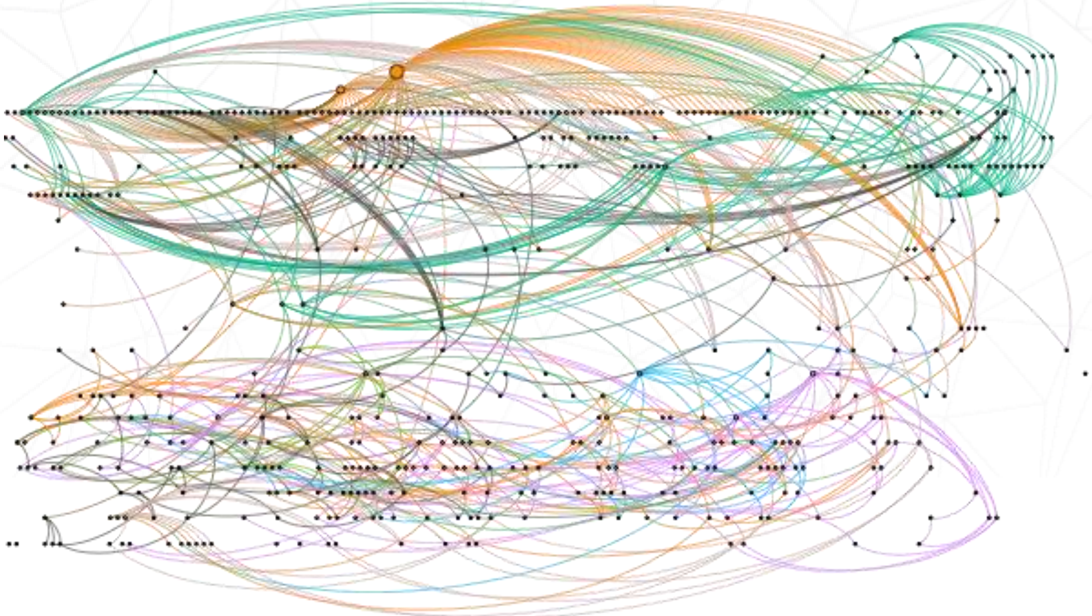
ANACONDA.
© 2018 Anaconda, Inc. - Confidential & Proprietary

3. Scales Down

- Trivial to install and use on a laptop in a single process
- Smooth transition to multi-core and then distributed operation

© 2018 Anaconda, Inc. - Confidential & Proprietary

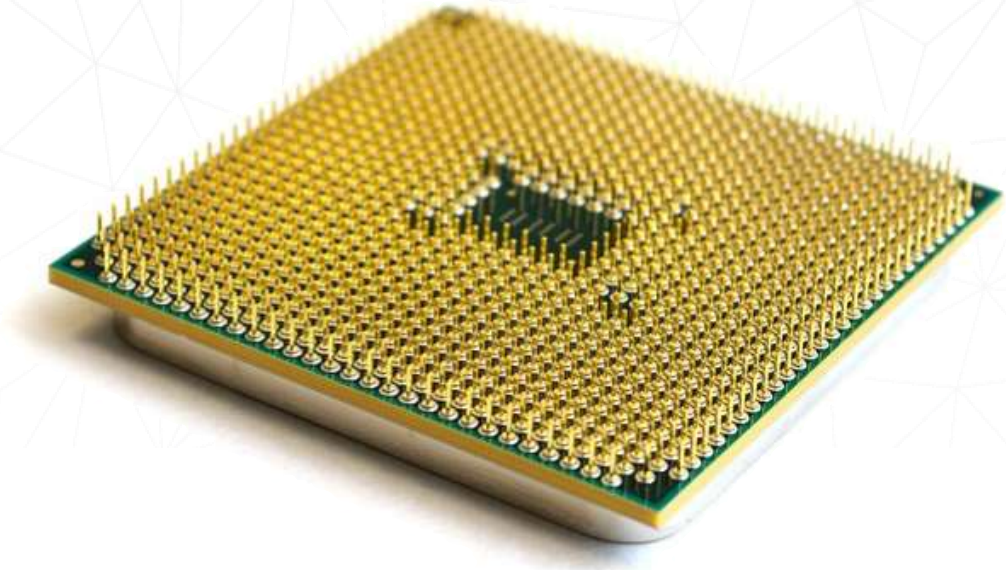
4. Solves Complex Problems



- Many problems are too complex for traditional big data tools
- Dask's scheduler handles these well
- Dask's APIs make it easy to develop bespoke distributed computing systems

© 2018 Anaconda, Inc. - Confidential & Proprietary

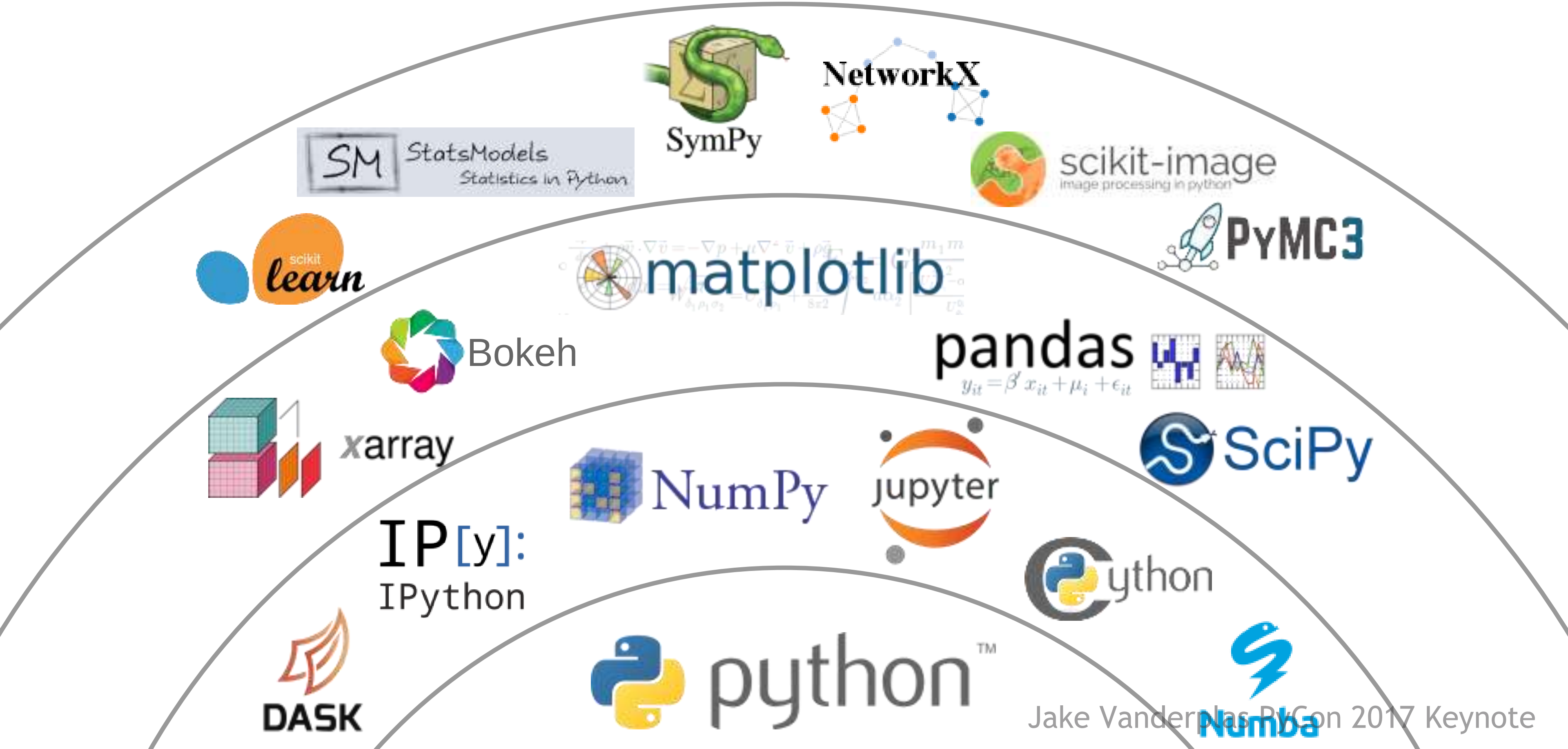
5. Native Execution



- One of the few non-JVM distributed frameworks
- Plays nicely with native compiled code, GPUs, etc..

© 2018 Anaconda, Inc. - Confidential & Proprietary

6. Part of a Broader Ecosystem



Some Alternatives to Choosing Dask

© 2018 Anaconda, Inc. - Confidential & Proprietary

Multiprocessing

Strengths

- Easy to use
- Well understood by many people
- Handles common case problems well

Weaknesses

- Doesn't handle complex workloads
- Recommend threads when using Numpy/Pandas/Scikit-Learn code

Please also consider concurrent.futures

© 2018 Anaconda, Inc. - Confidential & Proprietary

Apache Spark

Strengths

- **Implements broad subset of SQL**
  Hooks into BI tooling
- **Integrates well to traditional JVM infrastructure**
- **All-in-one framework**
  You only need to install one thing
- **Well trusted and broadly deployed**

Weaknesses

- **Doesn't extend well** beyond classic tabular computing
- **JVM-Native code barrier**
  Performance, debugging, usability
- **Reinvents its own ecosystem**
  Doesn't play well with existing systems
- **Often requires wholesale rewrite**

© 2018 Anaconda, Inc. - Confidential & Proprietary

MPI

Strengths

- **Very fast**
  still the fastest game in town
- **Very flexible**
  you can implement just about anything
- **Well deployed and supported**
  on high performance computers
- **The only real option for massive parallelism today**

Weaknesses

- **Hard to use by non-experts**
- **Often overkill for data analysis problems**
- **Not ideal** for problems with dynamic load

© 2018 Anaconda, Inc. - Confidential & Proprietary

Some Reasons not to Choose Dask

© 2018 Anaconda, Inc. - Confidential & Proprietary

Dask's limitations

- **Dask is not a SQL database.**
  Does Pandas well, but won't optimize complex queries.
  Consider PostgreSQL, Impala, SparkSQL
- **Dask is not MPI**
  200us task overhead, milliseconds of latency
- **Dask is not a JVM technology**
  Dask targets Python and associated languages (C/C++/…)
- **Dask is not always necessary**
  You may not need parallelism
  Find better algorithms, storage formats, compilers

© 2018 Anaconda, Inc. - Confidential & Proprietary

Who uses Dask?

© 2018 Anaconda, Inc. - Confidential & Proprietary

Three main user groups

1. **People who want big Pandas data frames**

   Common among early users.
   This is the most common class of questions on Stack Overflow

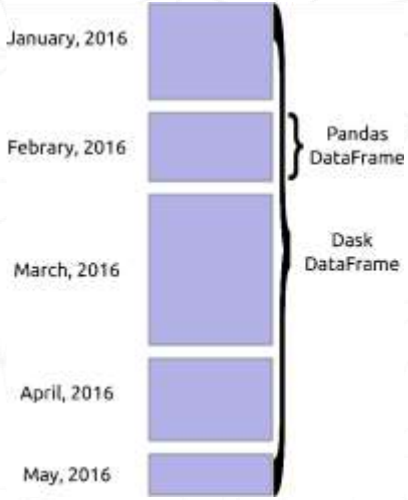
2. **Numeric groups with multi-dimensional arrays**

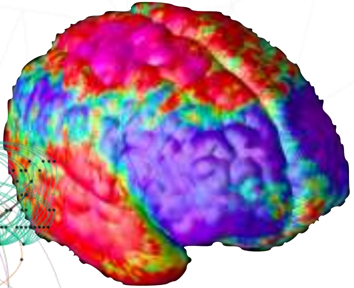
   Satellite imagery, advanced medical imaging, simulation analysis, signals processing

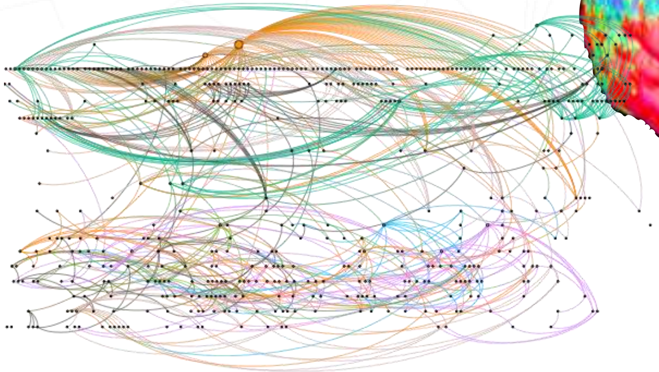
   No other big data system handles this kind of data well

3. **Advanced groups** accelerating their own **internal pipelines**

   Common in finance, operational data acquisition, hardware







ANACONDA.
© 2018 Anaconda, Inc. - Confidential & Proprietary

What we didn’t talk about

- Scikit-Learn and machine learning
- Array computing
- Real-time applications
- Understanding performance and scalability
- Deploying Dask

© 2018 Anaconda, Inc. - Confidential & Proprietary

Learn More

© 2018 Anaconda, Inc. - Confidential & Proprietary

Where to go next

Download Anaconda
https://www.anaconda.com/distribution/

Test Drive Anaconda Enterprise
ambassador@anaconda.com

Learn about consulting, training, and support
ambassador@anaconda.com

ANACONDA.
© 2018 Anaconda, Inc. - Confidential & Proprietary