Scale your data, not your process.
Welcome to the Blaze ecosystem!

Europython 2015

by Christine Doig

Introduction

About me

Data Scientist at Continuum Analytics

Barcelona & Austin

http://chdoig.github.com

About Continuum Analytics

Free Python distribution: Anaconda

Open source: conda, blaze, dask, bokeh, numba...

Proud sponsor of PyData, SciPy, PyCon and Europython

We are hiring!

http://continuum.io

About the talk

- Five Areas of Data Science
- The Data Science Triforce
- The Blaze ecosystem
  - blaze
  - datashape
  - odo
  - dask

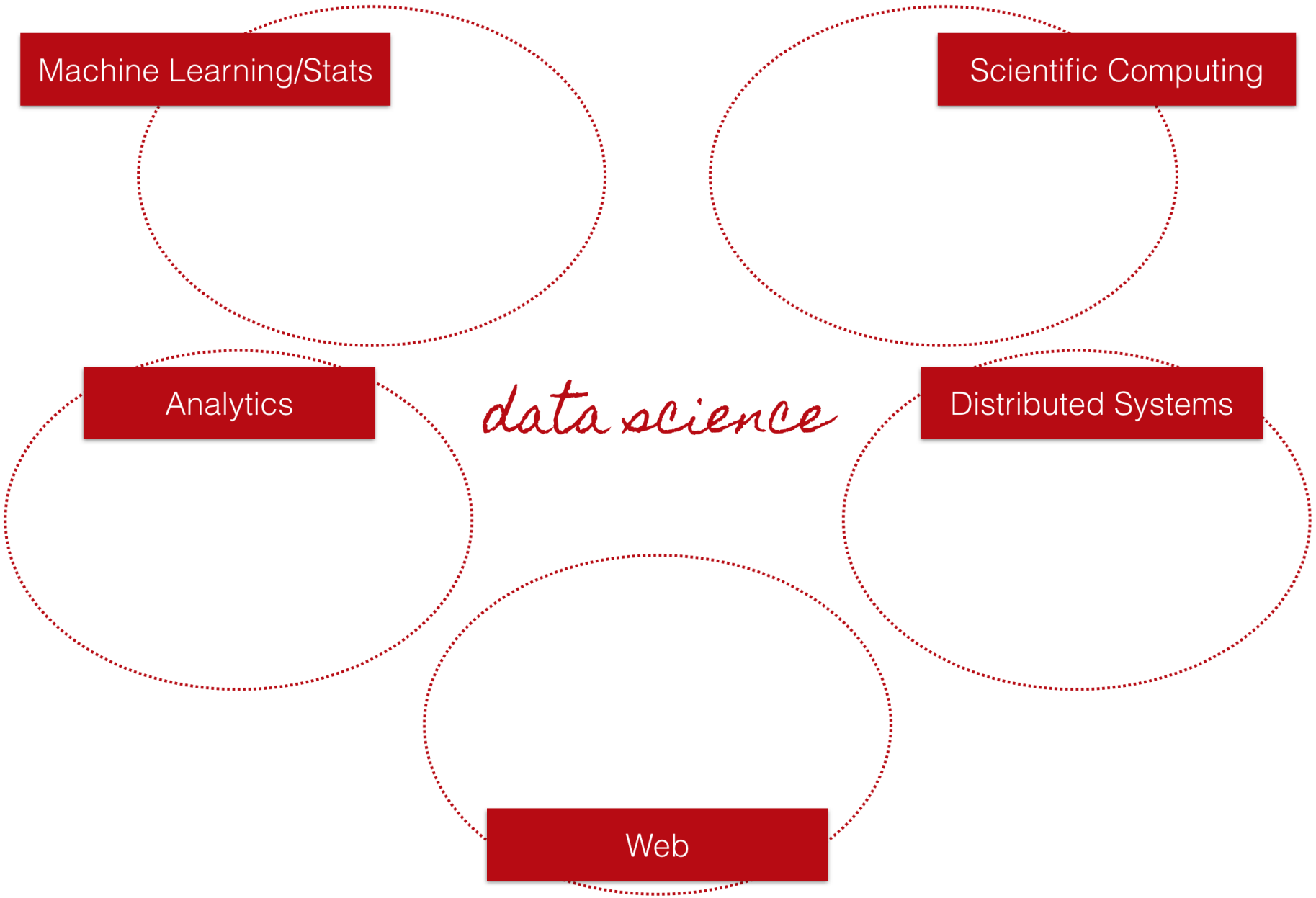
Slides

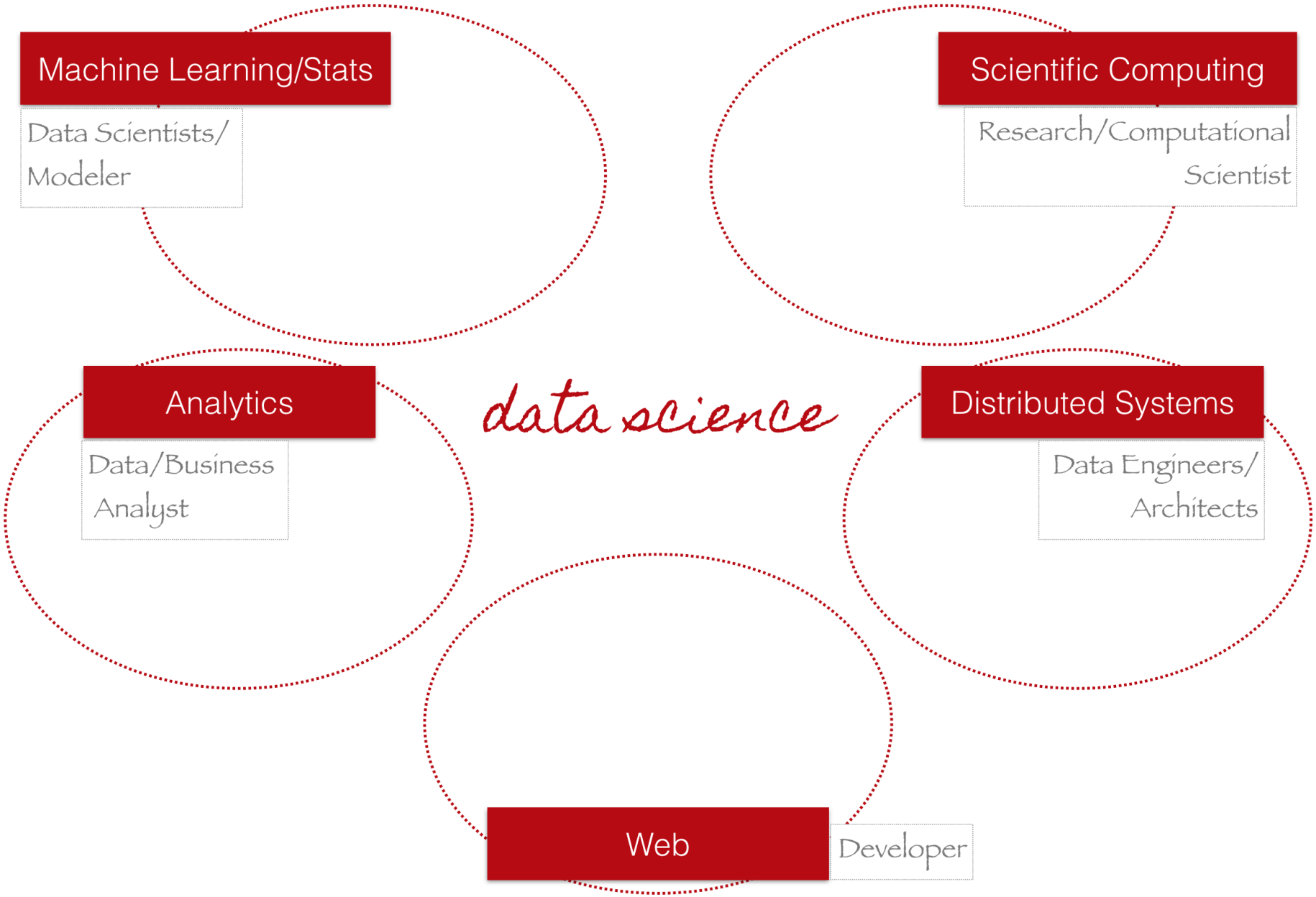
http://chdoig.github.io/ep2015-blaze

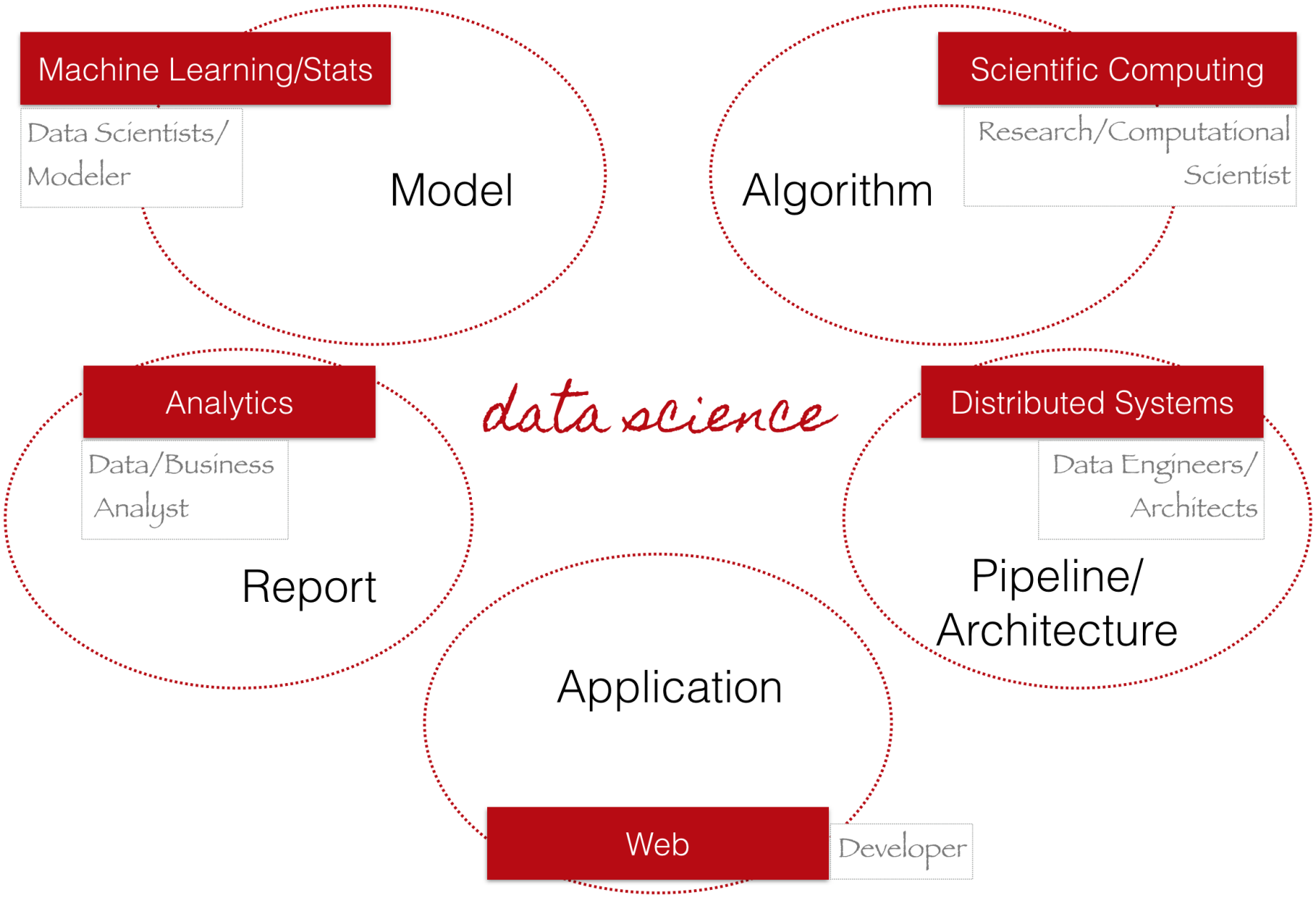
Repository

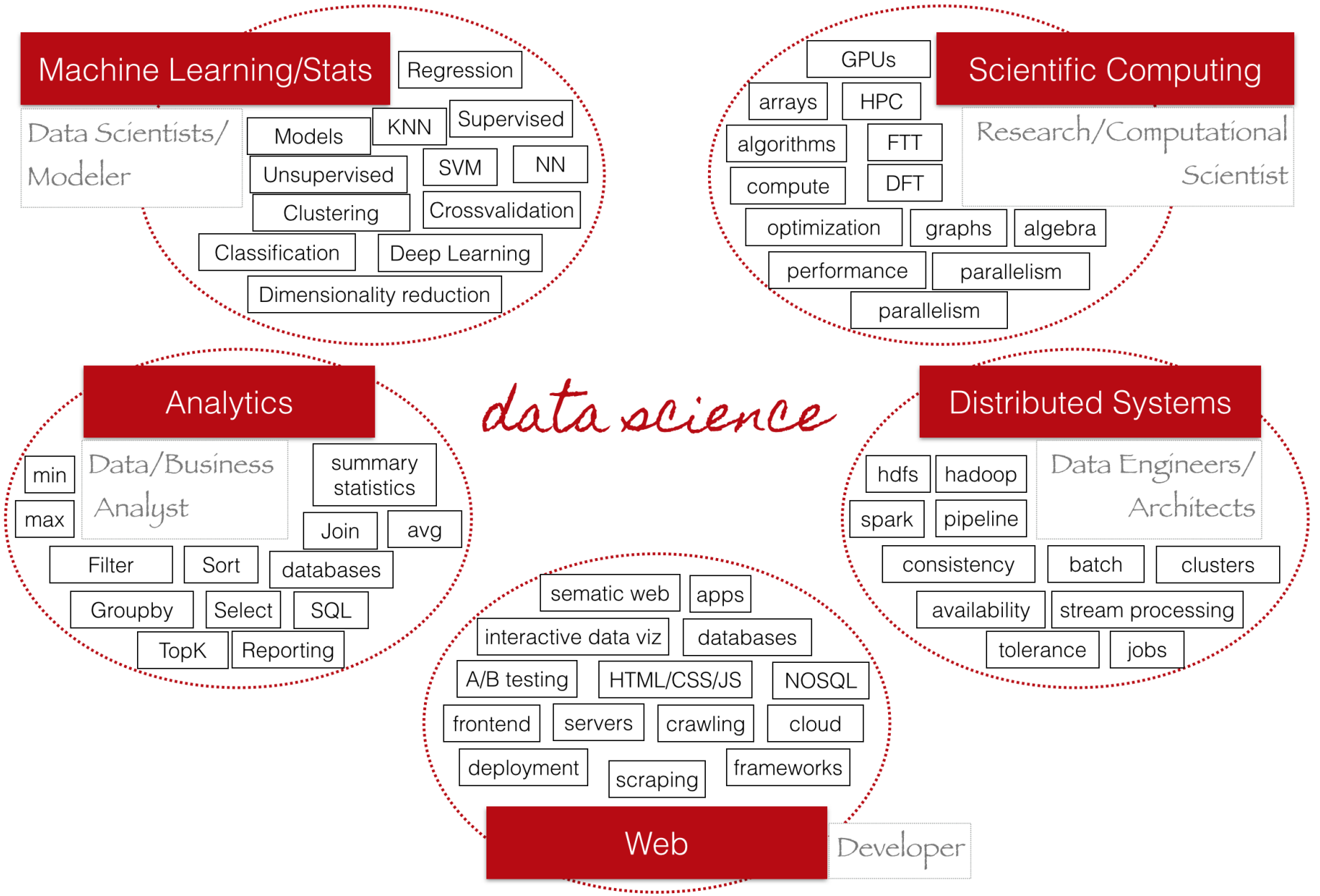
http://github.com/chdoig/ep2015-blaze

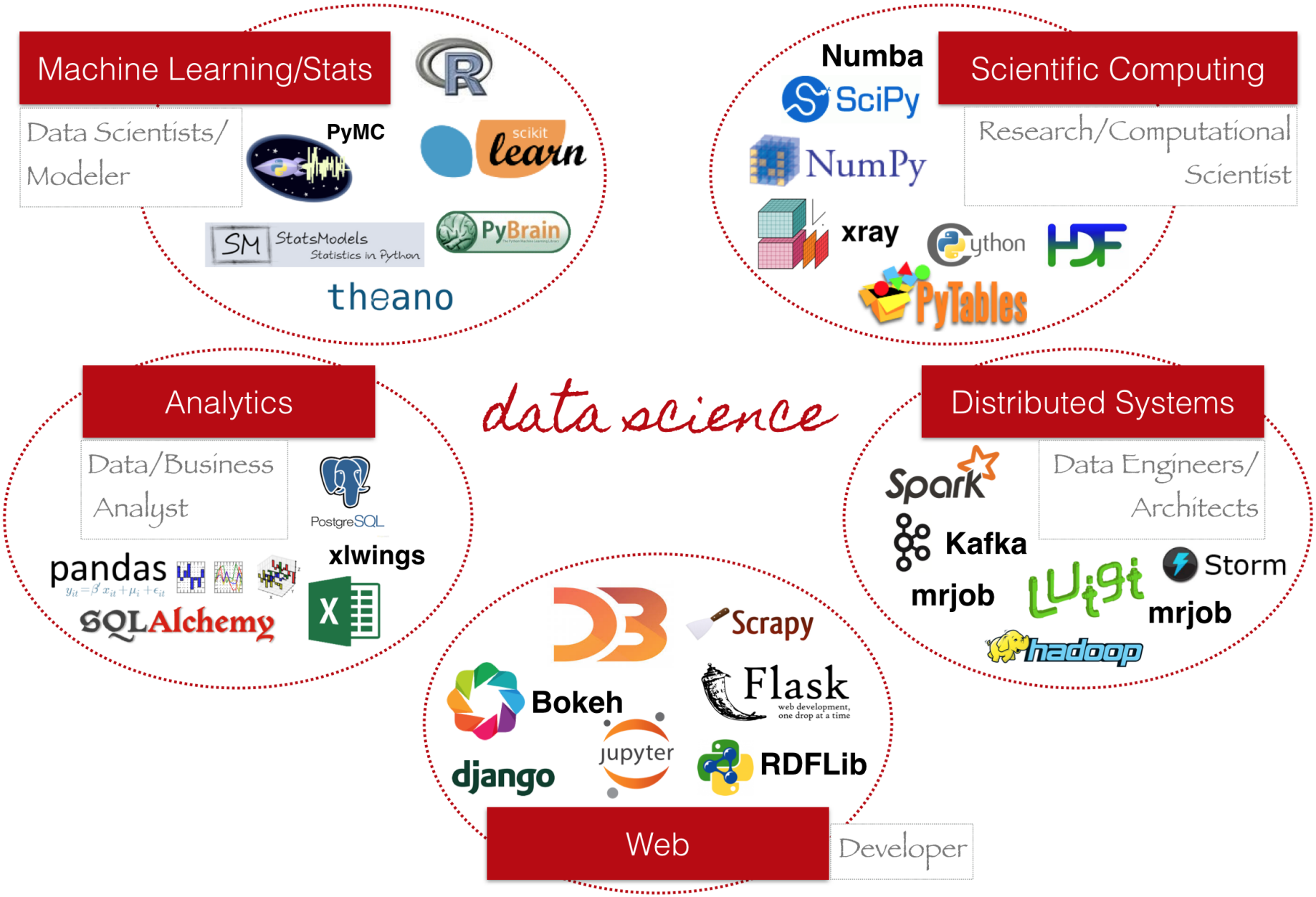
Five Areas of Data Science


Machine Learning/Stats
Scientific Computing
Analytics
data science
Distributed Systems
Web

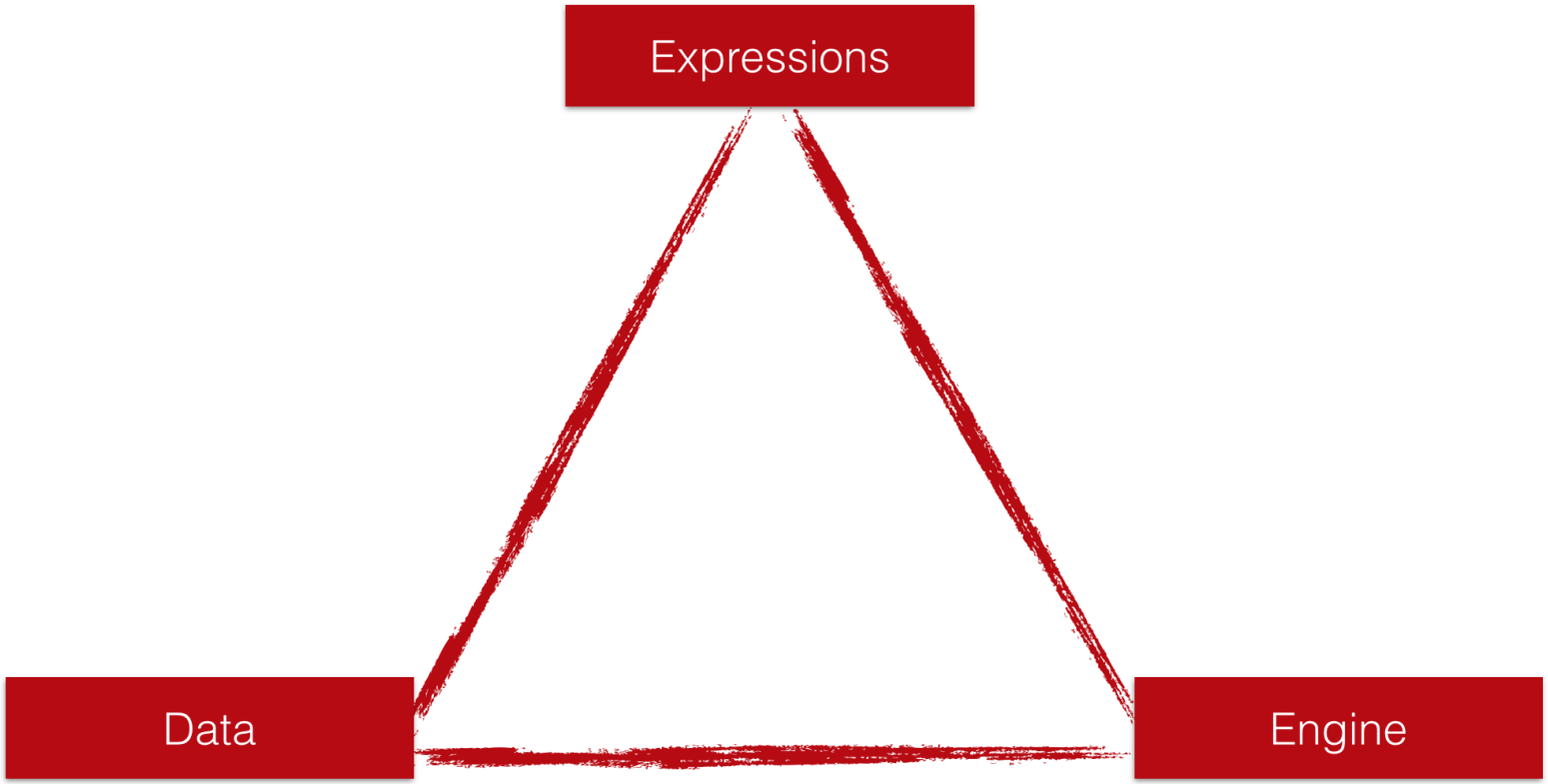

Machine Learning/Stats
Data Scientists/ Modeler
Scientific Computing
Research/Computational Scientist
Analytics
Data/Business Analyst
data science
Distributed Systems
Data Engineers/ Architects
Web
Developer


Machine Learning/Stats
Data Scientists/ Modeler
Model
Scientific Computing
Research/Computational Scientist
Algorithm
Analytics
Data/Business Analyst
Report
data science
Distributed Systems
Data Engineers/ Architects
Pipeline/ Architecture
Application
Web
Developer

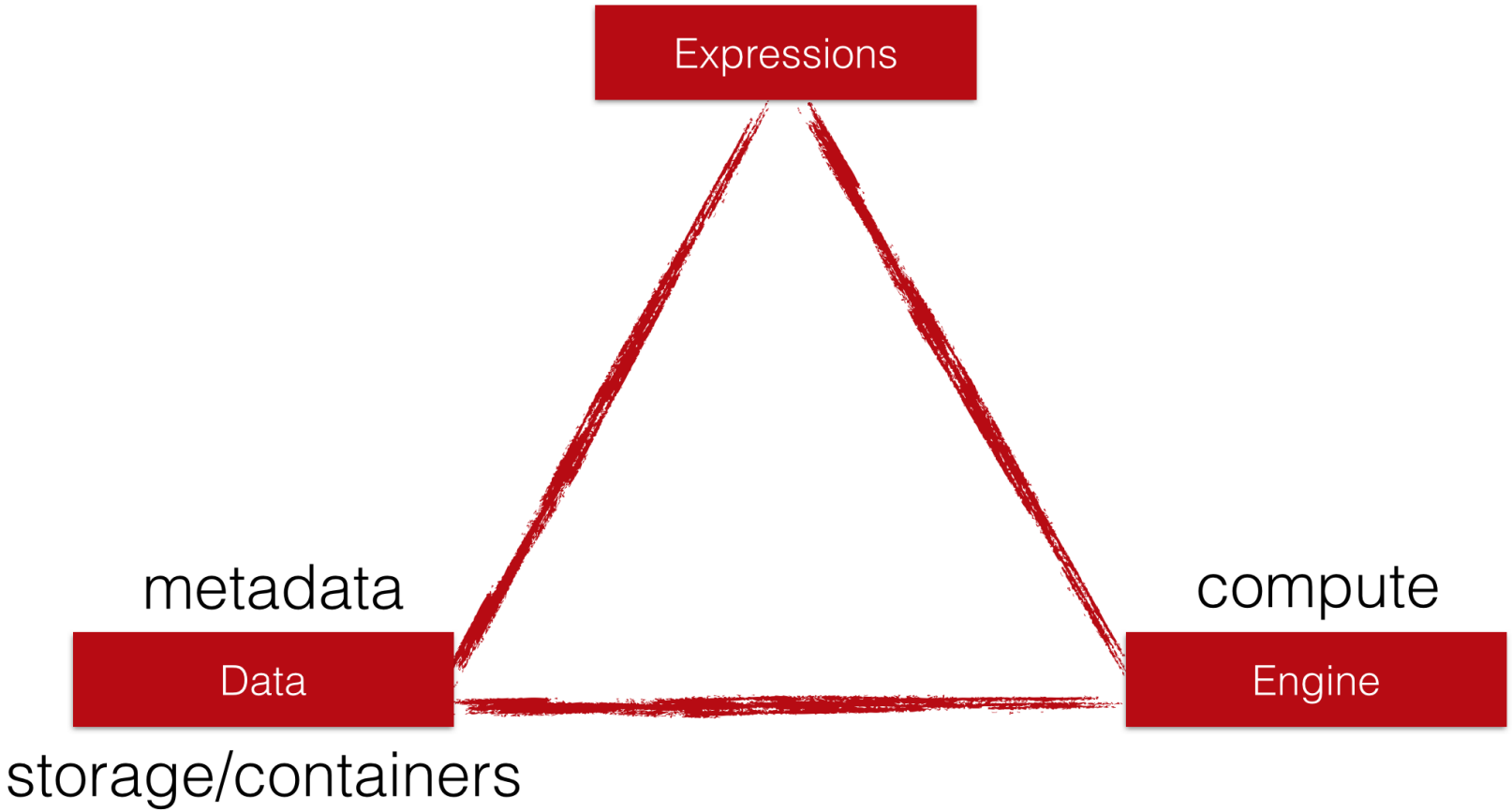

Machine Learning/Stats
Regression
Data Scientists/ Modeler
Models
KNN
Supervised
Unsupervised
SVM
NN
Clustering
Crossvalidation
Classification
Deep Learning
Dimensionality reduction
Scientific Computing
GPUs
arrays
HPC
algorithms
FTT
compute
DFT
Research/Computational Scientist
optimization
graphs
algebra
performance
parallelism
parallelism
data science
Analytics
min
max
Data/Business Analyst
summary statistics
Join
avg
Filter
Sort
databases
Groupby
Select
SQL
TopK
Reporting
Distributed Systems
hdfs
hadoop
Data Engineers/ Architects
spark
pipeline
consistency
batch
clusters
availability
stream processing
tolerance
jobs
sematic web
apps
interactive data viz
databases
A/B testing
HTML/CSS/JS
NOSQL
frontend
servers
crawling
cloud
deployment
scraping
frameworks
Web
Developer


Machine Learning/Stats
Data Scientists/ Modeler
PyMC
scikit learn
StatsModels Statistics in Python
PyBrain
theano
Numba
SciPy
NumPy
xray
ython
HDF
PyTables
Scientific Computing
Research/Computational Scientist
data science
Analytics
Data/Business Analyst
PostgreSQL
pandas
xlwings
SQLAlchemy
Distributed Systems
Data Engineers/ Architects
Spark
Kafka
mrjob
luigi
Storm
mrjob
hadoop
Scrapy
Flask
web development, one drop at a time
Bokeh
jupyter
RDFLib
django
Web
Developer

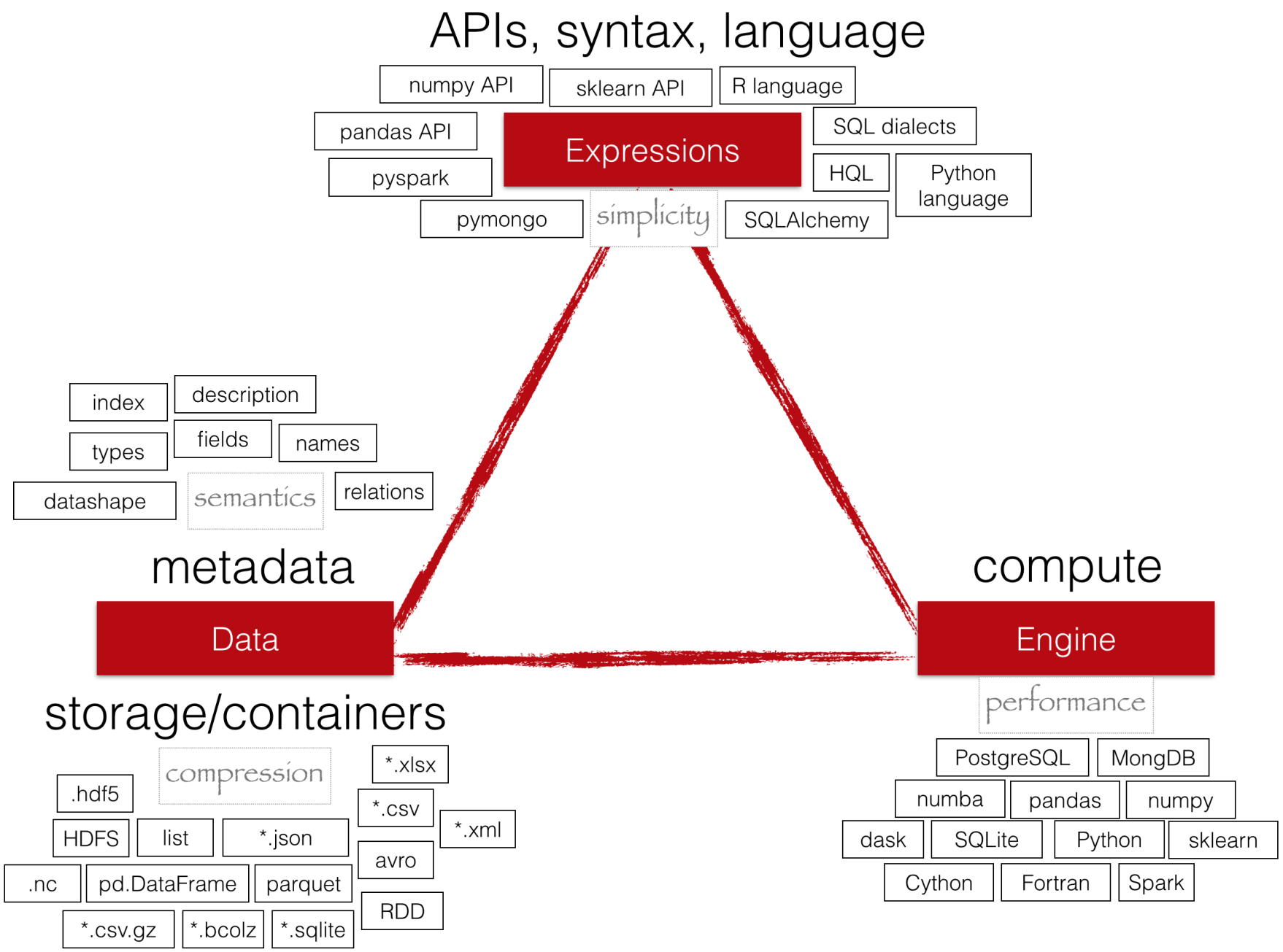
The Data Science Triforce


Expressions
Data
Engine

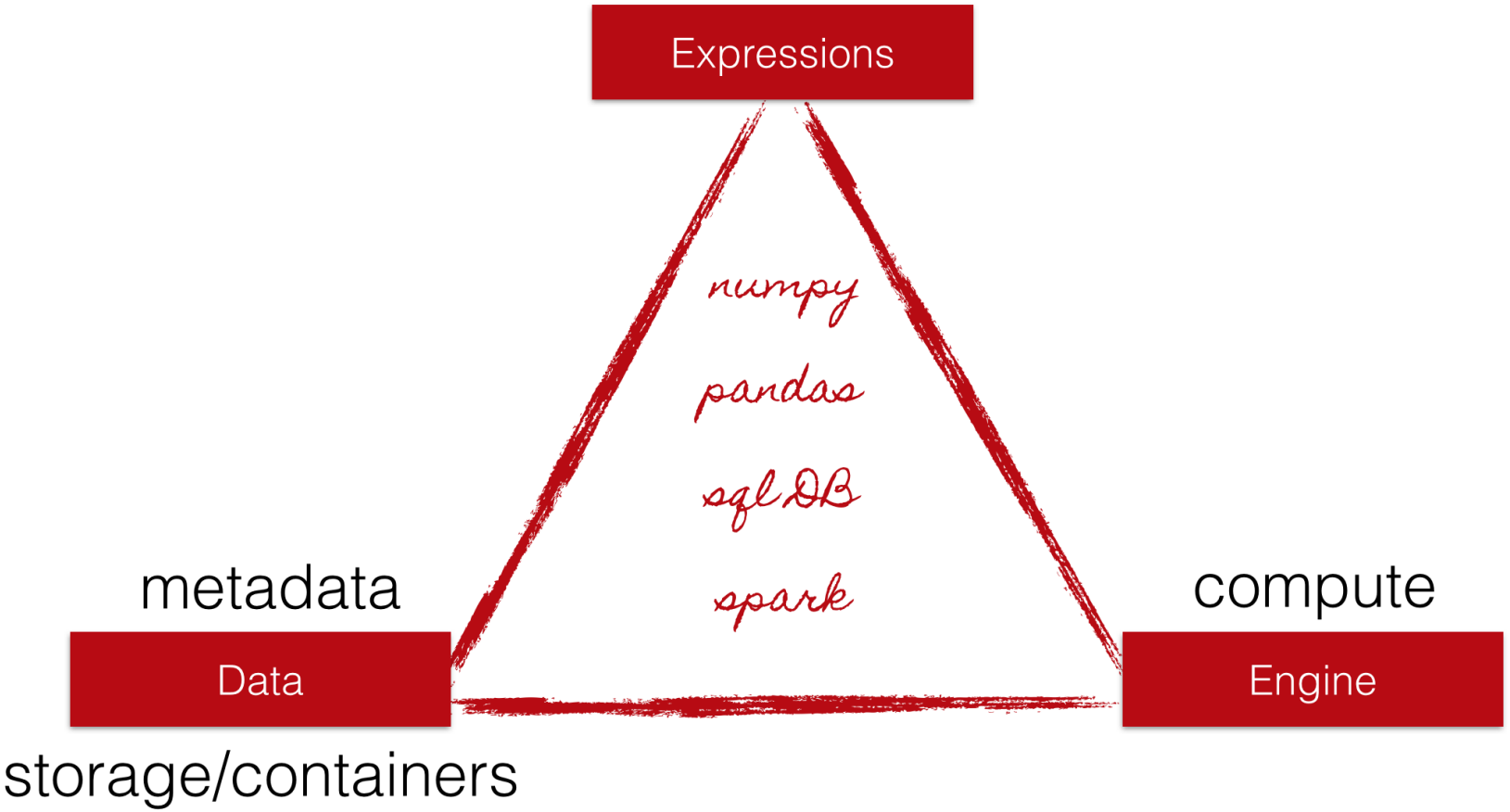
APIs, syntax, language



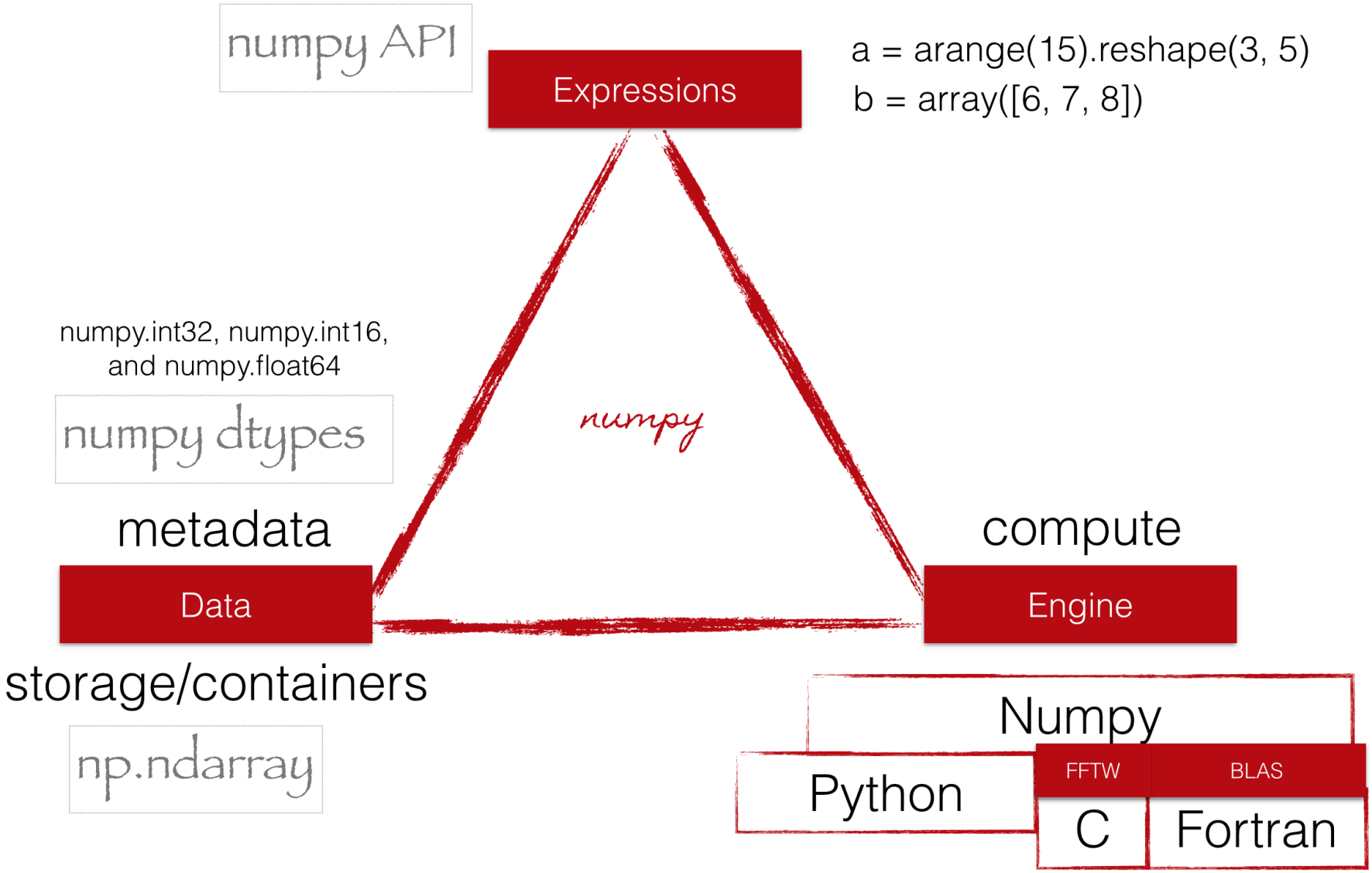
APIs, syntax, language


APIs, syntax, language
numpy API
sklearn API
R language
pandas API
Expressions
SQL dialects
pyspark
HQL
Python language
pymongo
simplicity
SQLAlchemy
index
description
types
fields
names
datashape
semantics
relations
metadata
compute
Data
Engine
performance
storage/containers
compression
*.xlsx
.hdf5
*.csv
*.xml
HDFS
list
*.json
avro
.nc
pd.DataFrame
parquet
RDD
*.csv.gz
*.bcolz
*.sqlite
PostgreSQL
MongDB
numba
pandas
numpy
dask
SQLite
Python
sklearn
Cython
Fortran
Spark

APIs, syntax, language

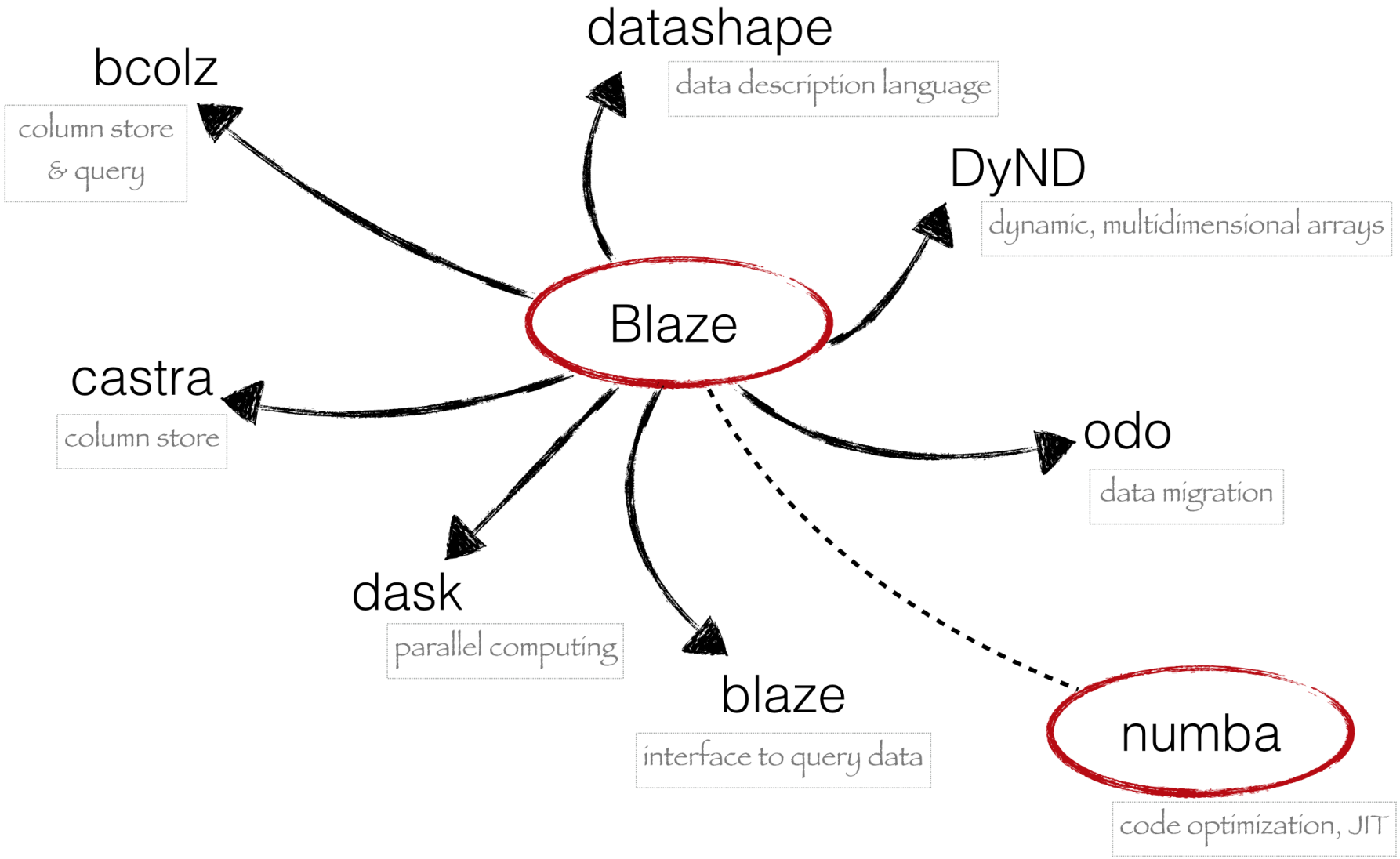


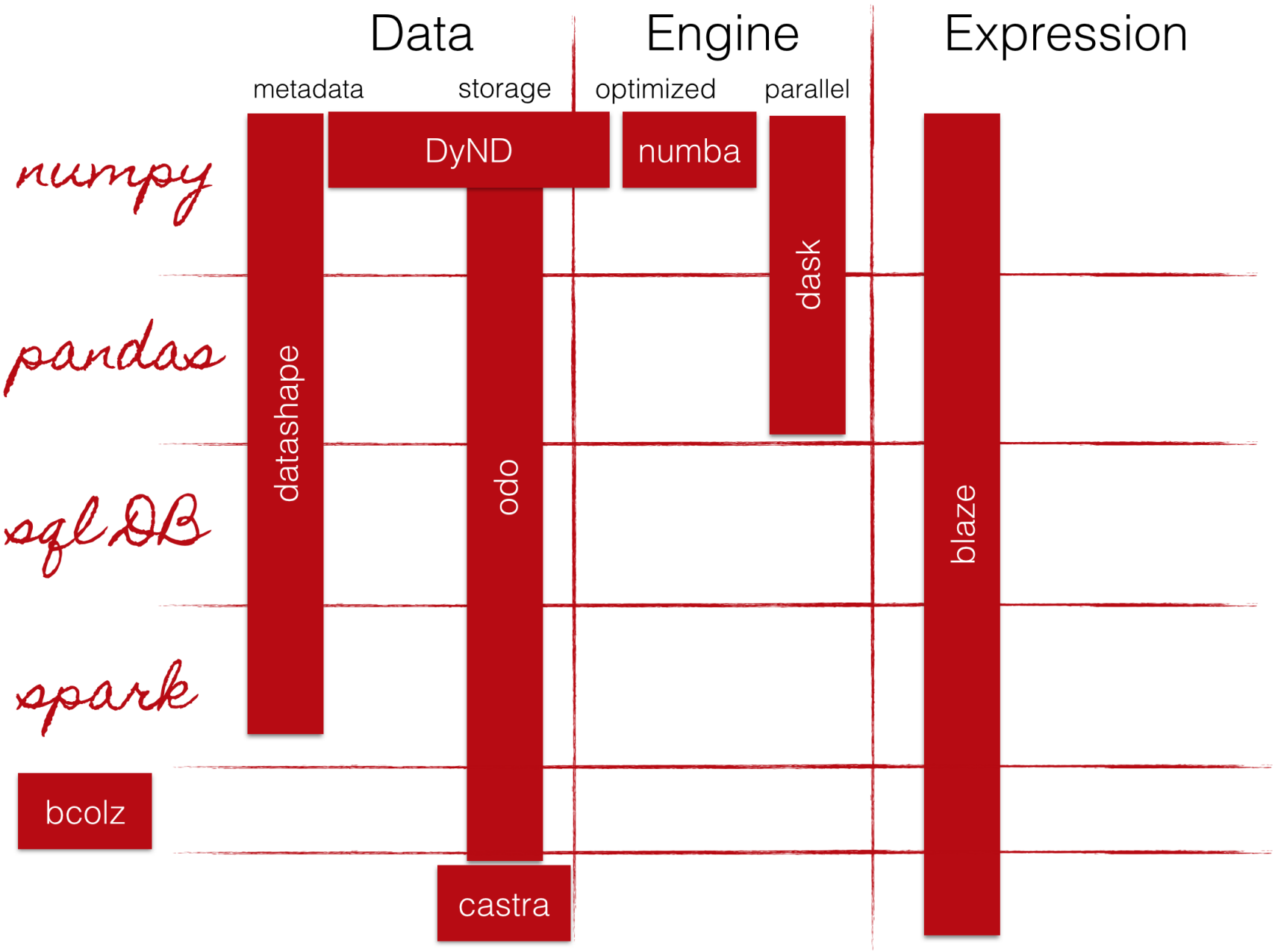
APIs, syntax, language



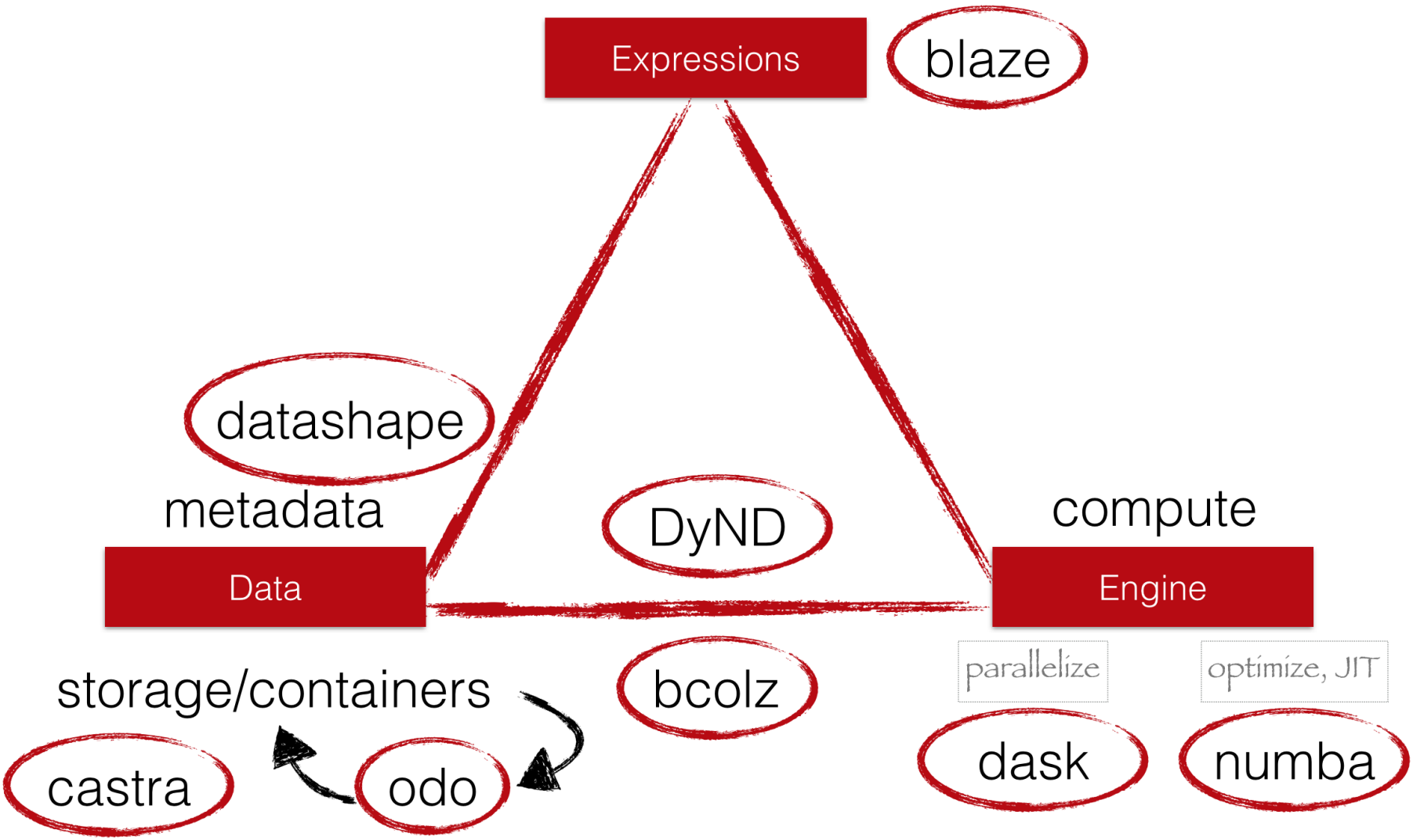
	Data	Engine	Expression
numpy	 < memory, types	 memory bound	 scientists, arrays
pandas	 < memory	 memory bound	 data scientist, analysts, DataFrames
sqlDB	 > memory, < TBs	 index, optimizations	 dialects, SQL, overhead setup
spark	 connectors	 size	 RDDs DataFrames

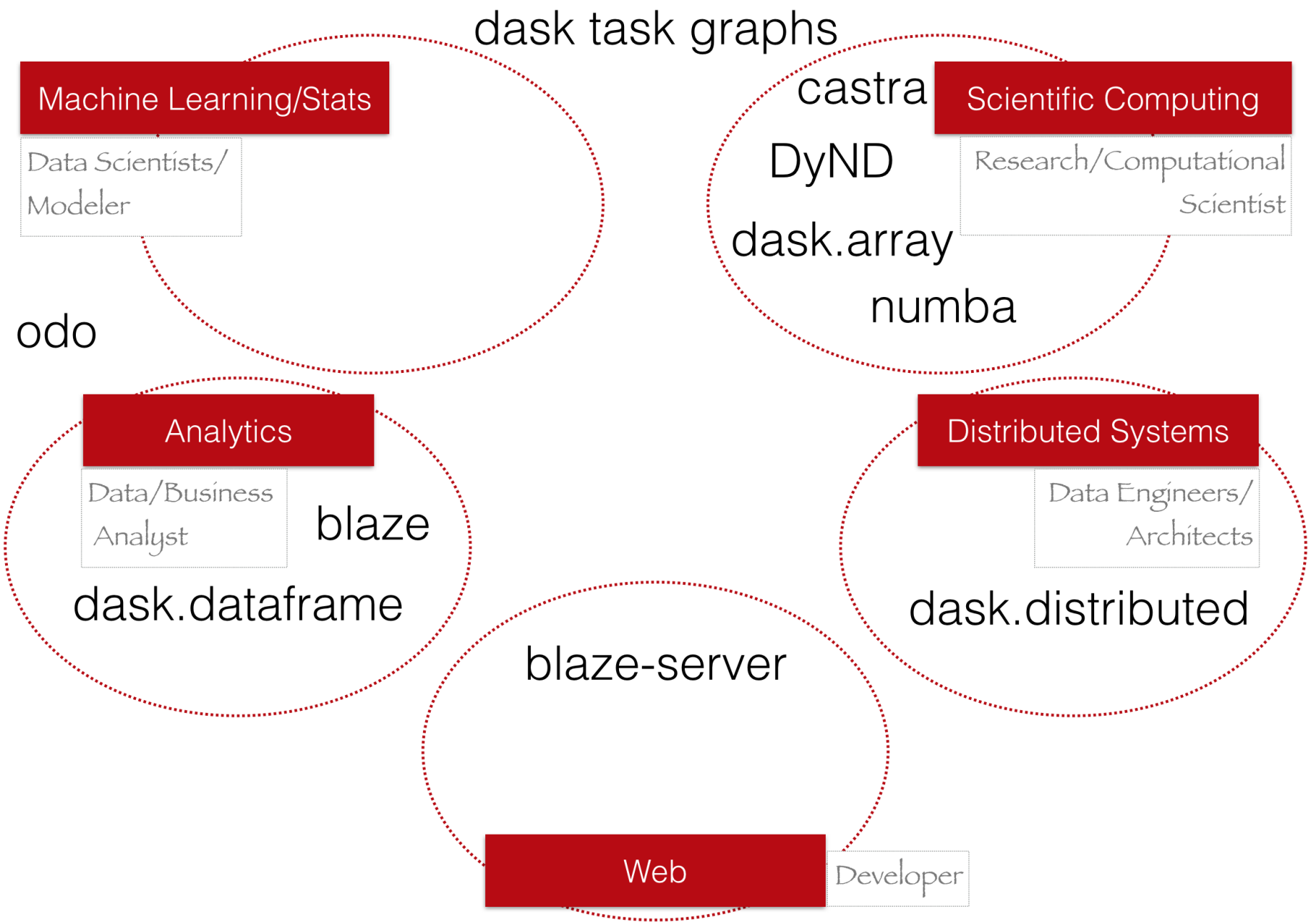
Blaze ecosystem

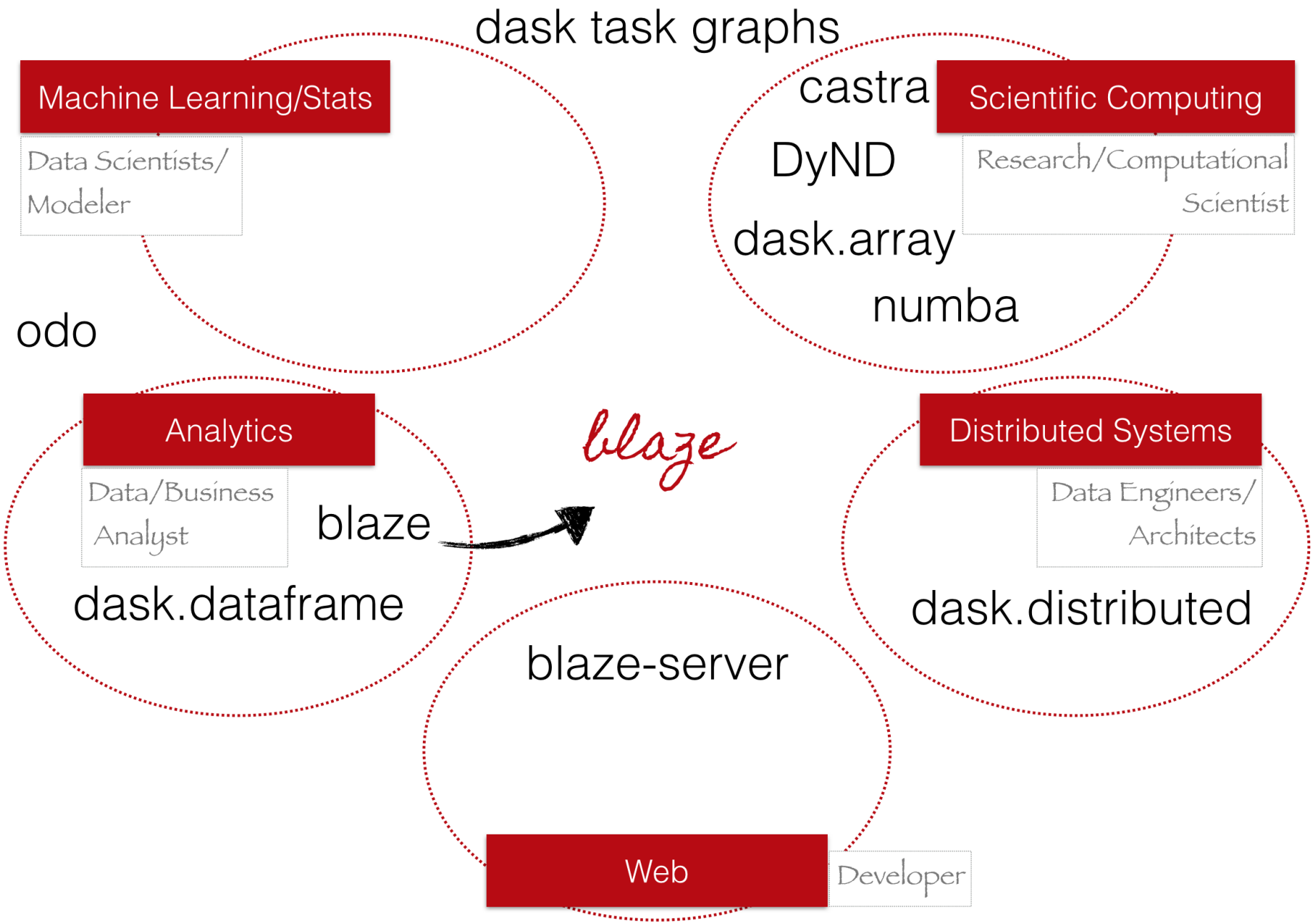

bcolz
column store & query
datashape
data description language
DyND
dynamic, multidimensional arrays
Blaze
castra
column store
odo
data migration
dask
parallel computing
blaze
interface to query data
numba
code optimization, JIT

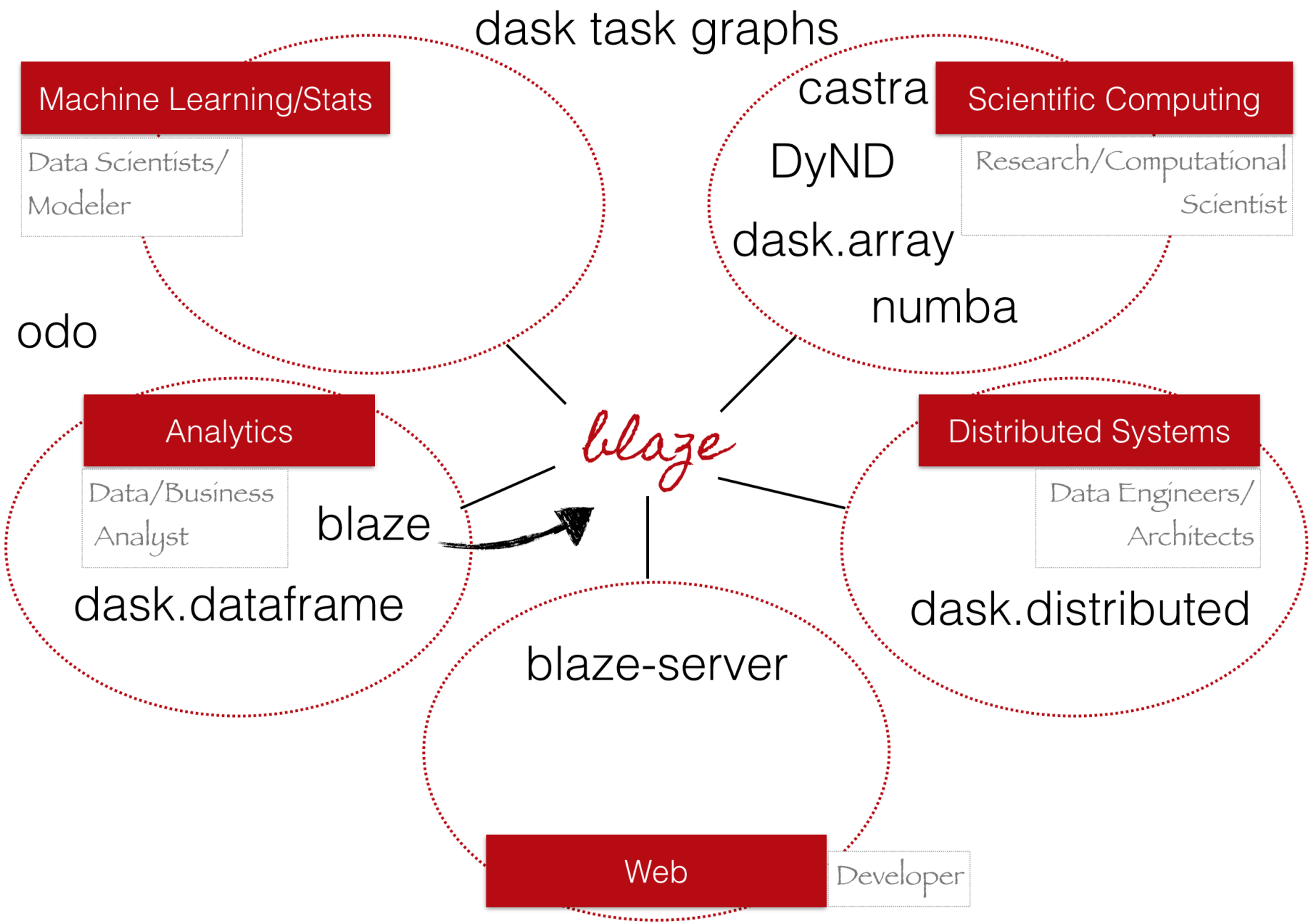

Data
Engine
Expression
metadata
storage
optimized
parallel
numpy
pandas
sqlDB
spark
bcolz
DyND
numba
dask
datashape
odo
blaze
castra

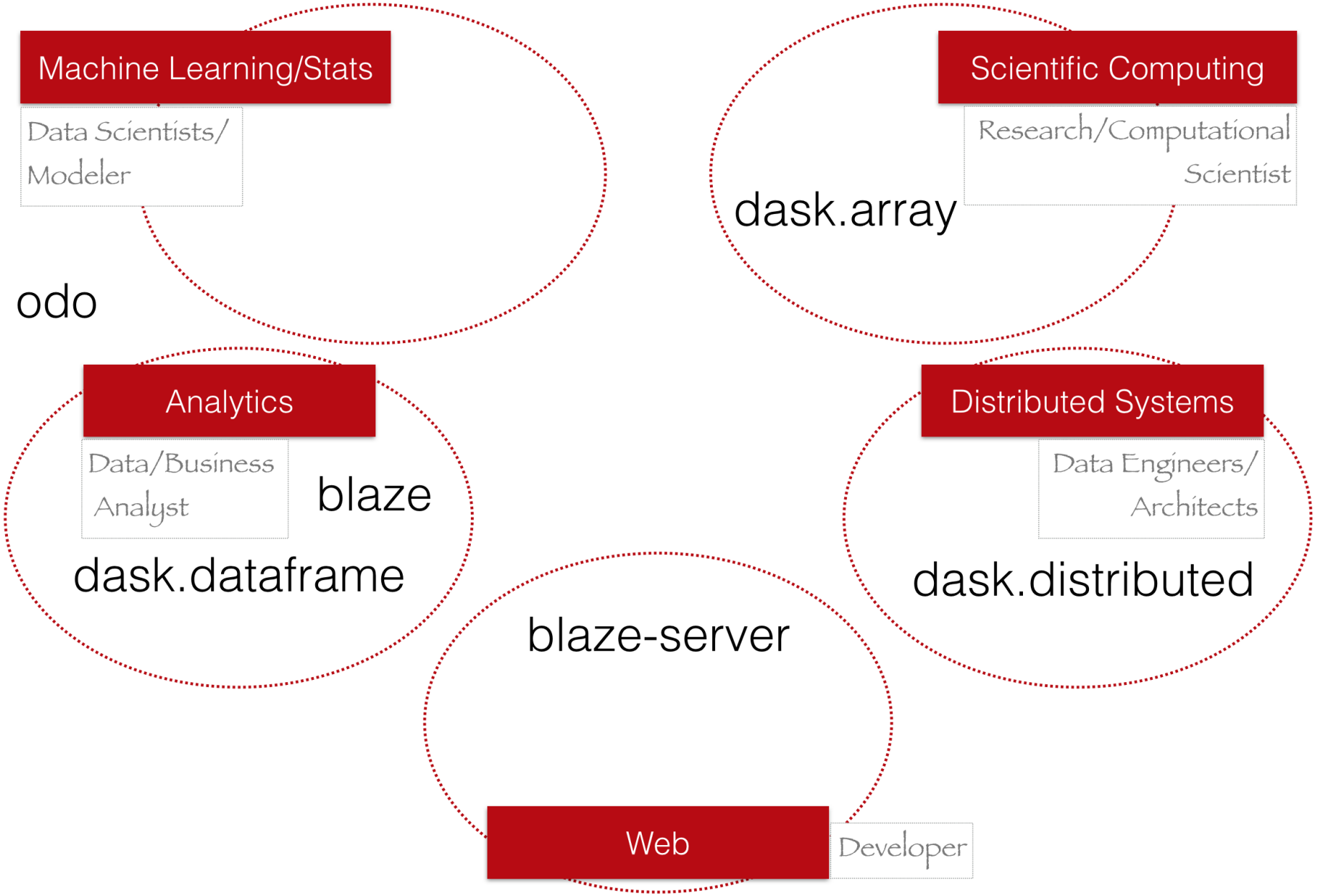
APIs, syntax, language



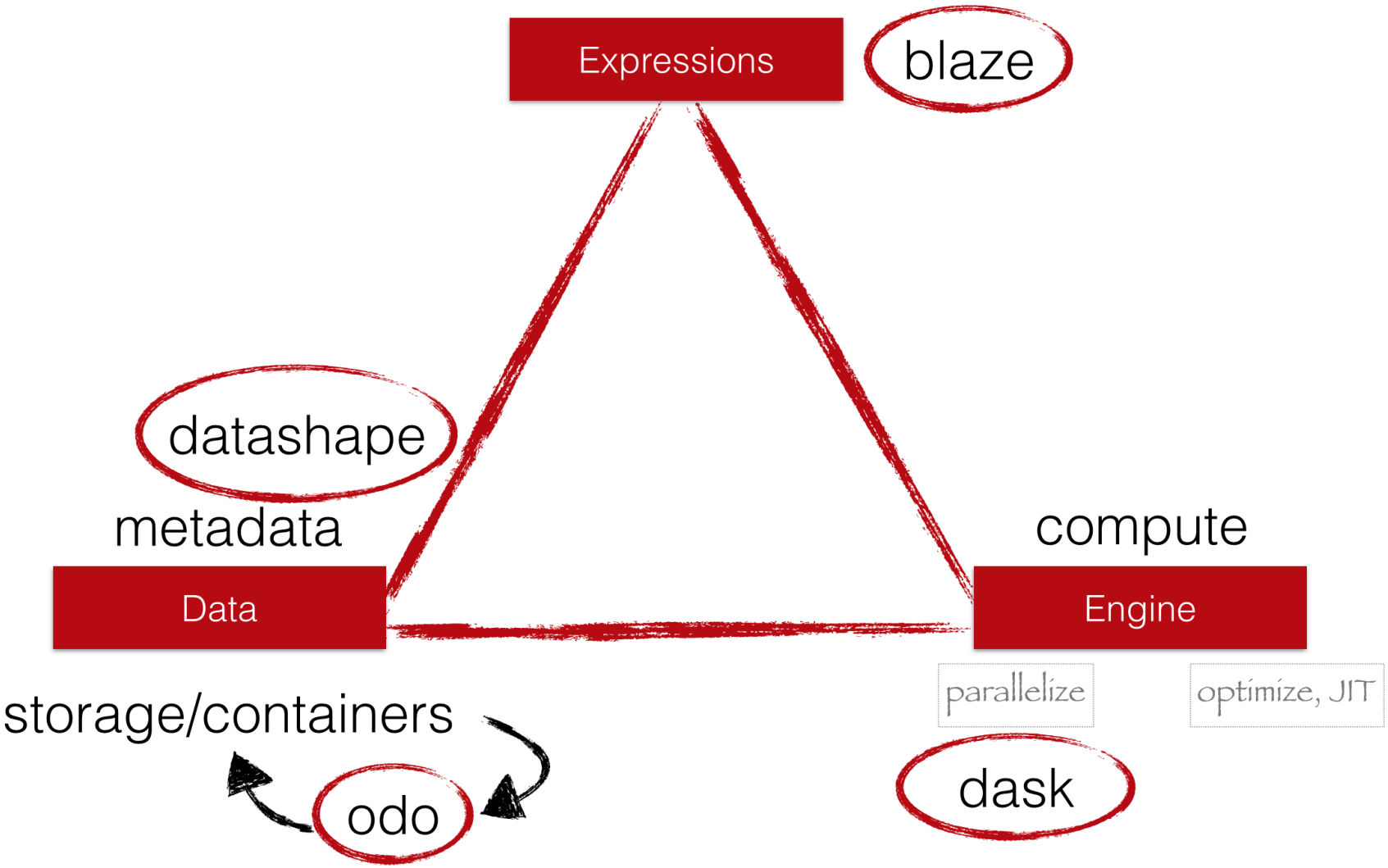

dask task graphs
Machine Learning/Stats
Data Scientists/
Modeler
castra
DyND
dask.array
numba
Scientific Computing
Research/Computational
Scientist
odo
Analytics
Data/Business
Analyst
blaze
dask.dataframe
Distributed Systems
Data Engineers/
Architects
dask.distributed
blaze-server
Web
Developer


dask task graphs
Machine Learning/Stats
Data Scientists/
Modeler
castra
DyND
dask.array
numba
Scientific Computing
Research/Computational
Scientist
odo
Analytics
Data/Business
Analyst
blaze
dask.dataframe
blaze
Distributed Systems
Data Engineers/
Architects
dask.distributed
blaze-server
Web
Developer


dask task graphs
Machine Learning/Stats
Data Scientists/
Modeler
castra
DyND
dask.array
numba
Scientific Computing
Research/Computational
Scientist
odo
Analytics
Data/Business
Analyst
blaze
blaze
dask.dataframe
Distributed Systems
Data Engineers/
Architects
dask.distributed
blaze-server
Web
Developer


Machine Learning/Stats
Data Scientists/
Modeler
odo
Scientific Computing
Research/Computational
Scientist
dask.array
Analytics
Data/Business
Analyst
blaze
dask.dataframe
Distributed Systems
Data Engineers/
Architects
dask.distributed
blaze-server
Web
Developer

APIs, syntax, language



blaze

interface to query data on different storage systems

http://blaze.pydata.org/en/latest/

```
from blaze import Data
```

CSV
```
iris = Data('iris.csv')
```

SQL
```
iris = Data('sqlite:///flowers.db::iris')
```

MongoDB
```
iris = Data('mongodb://localhost/mydb::iris')
```

JSON
```
iris = Data('iris.json')
```

S3
```
iris = Data('s3://blaze-data/iris.csv')
```

...

blaze

Select columns

```
iris[['sepal_length', 'species']]
```

Filter

```
iris[(iris.species == 'Iris-setosa') &
     (iris.sepal_length > 5.0)]
```

Operate

```
log(iris.sepal_length * 10)
```

Reduce

```
iris.sepal_length.mean()
```

Split-apply-combine

```
by(iris.species, shortest=iris.petal_length.min(),
                 longest=iris.petal_length.max(),
                 average=iris.petal_length.mean())
```

Add new columns

```
transform(iris,
          sepal_ratio = iris.sepal_length / iris.sepal_width,
          petal_ratio = iris.petal_length / iris.petal_width)
```

Relabel columns

```
iris.relabel(petal_length='PETAL-LENGTH',
             petal_width='PETAL-WIDTH')
```

Text matching

```
iris.like(species='*versicolor')
```

Builds off of Blaze uniform interface to host data remotely through a JSON web API.

```yaml
iriscsv:
  source: iris.csv

irisdb:
  source: sqlite:///flowers.db::iris

irisjson:
  source: iris.json
  dshape: "var * {name: string, amount: float64}"

irismongo:
  source: mongodb://localhost/mydb::iris
```

server.yaml YAML

$ blaze-server server.yaml --follow-links -e

localhost:6363/compute.json

Blaze Client

```
>>> from blaze import Data

>>> s = Data('blaze://localhost:6363')
>>> t.fields

[u'iriscsv', u'irisdb', u'irisjson', u'irismongo']

>>> t.iriscsv
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

```
>>> t.irisdb
```

	petal_length	petal_width	sepal_length	sepal_width	species
0	1.4	0.2	5.1	3.5	Iris-setosa
1	1.4	0.2	4.9	3.0	Iris-setosa
2	1.3	0.2	4.7	3.2	Iris-setosa

blaze server

curl

```
curl  \
   -H "Content-Type: application/json"  \
   -d '{"expr": {"op": "Field", "args": [":leaf", "iriscsv"]}}'  \
  localhost:6363/compute.json
```

blaze.server.to_tree + requests

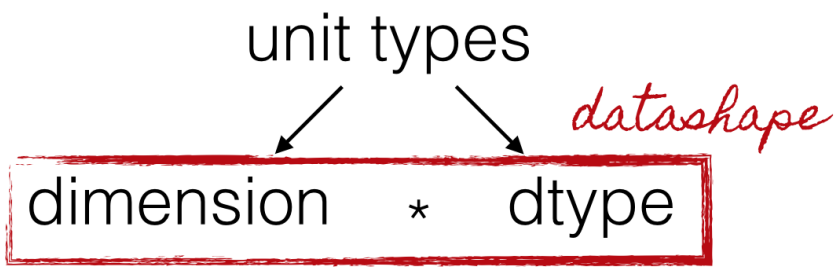
... need to explain datashape first...

datashape

datashape

a structured data description language

http://datashape.pydata.org/



var * int32

3 * string

4 * float64

ordered struct dtype

record

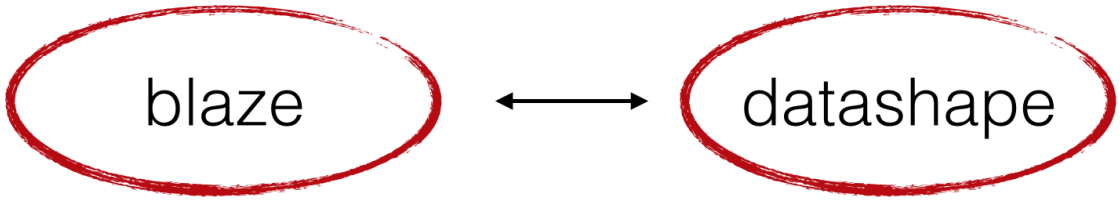
collection of types keyed by labels

{ x : int32, y : string, z : float64 }

tabular datashape

var * { x : int32, y : string, z : float64 }


```
{
  flowersdb: {
    iris: var * {
      petal_length: float64,
      petal_width: float64,
      sepal_length: float64,
      sepal_width: float64,
      species: string
      }
    },
  iriscsv: var * {
    sepal_length: ?float64,
    sepal_width: ?float64,
    petal_length: ?float64,
    petal_width: ?float64,
    species: ?string
    },
  irisjson: var * {
    petal_length: float64,
    petal_width: float64,
    sepal_length: float64,
    sepal_width: float64,
    species: string
    },
  irismongo: 150 * {
    petal_length: float64,
    petal_width: float64,
    sepal_length: float64,
    sepal_width: float64,
    species: string
    }
}
```



Blaze uses datashape as its type system.

```
>>> iris = Data('iris.json')
>>> s.iris.dshape

dshape("""var * {
  petal_length: float64,
  petal_width: float64,
  sepal_length: float64,
  sepal_width: float64,
  species: string
  }""")
```

... back to blaze server...

blaze.server.to_tree + requests

```
>>> dshape = """{
  flowersdb: {
    iris: var * {
      petal_length: float64,
      petal_width: float64,
      sepal_length: float64,
      sepal_width: float64,
      species: string
      }
    },
  iriscsv: var * {
    sepal_length: ?float64,
    sepal_width: ?float64,
    petal_length: ?float64,
    petal_width: ?float64,
    species: ?string
    },
  irisjson: var * {
    petal_length: float64,
    petal_width: float64,
    sepal_length: float64,
    sepal_width: float64,
    species: string
    },
  irismongo: 150 * {
    petal_length: float64,
    petal_width: float64,
    sepal_length: float64,
    sepal_width: float64,
    species: string
    }
}"""
```

```
>>> from blaze import symbol
>>> from blaze.server import to_tree
>>> import requests

>>> s = symbol('s', dshape)

>>> expr = s.iriscsv.petal_length.max()

>>> d = to_tree(expr, names={s: ':leaf'})

>>> query = {'expr': d}

>>> r = requests.get('http://localhost:6363/compute.json',
...                  data=json.dumps(query),
...                  headers={'Content-Type': 'application/json'})

>>> json.loads(r.content)

{u'data': 6.9, u'datashape': u'?float64', u'names': [u'petal_length_max']}
```

odo

data migration, ~ cp with types, for data

http://odo.pydata.org/en/latest/

```
from odo import odo
odo(source, target)
```

```
odo('iris.csv', 'iris.json')
```

```
odo('iris.json', 'sqlite:///flowers.db::iris')
```

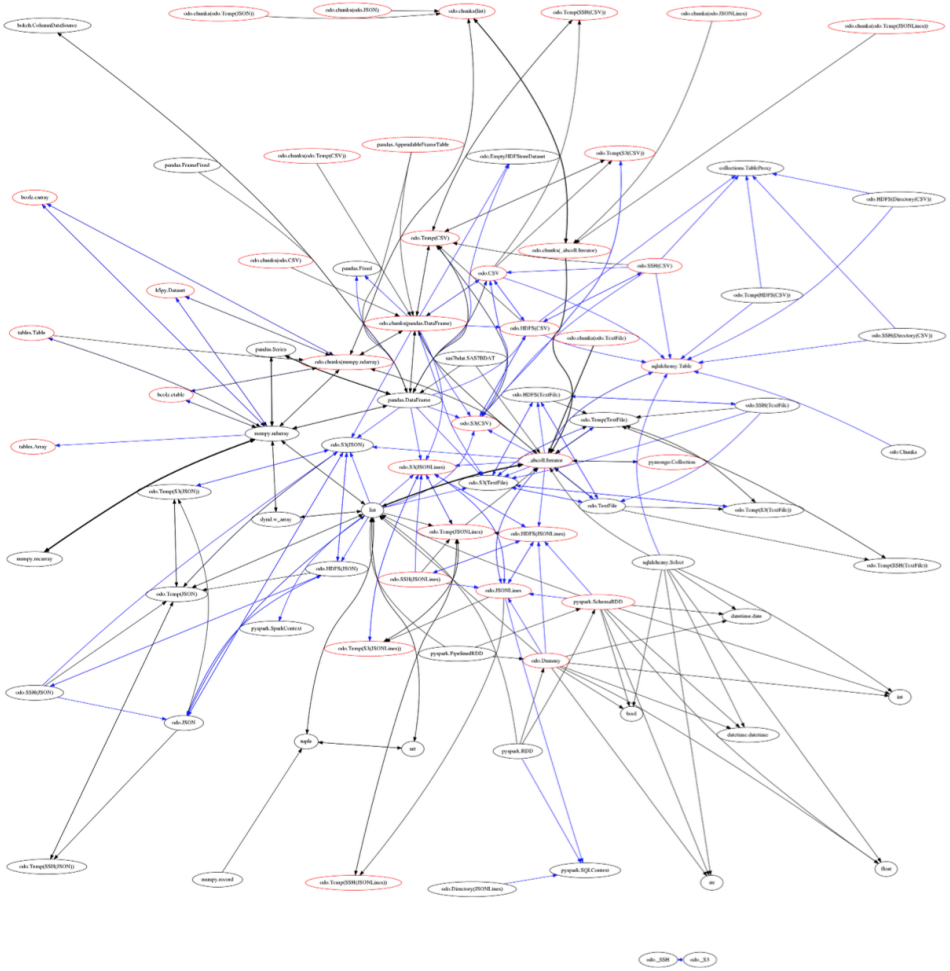
```
odo('iris.json', 'mongodb://localhost/mydb::iris')
```

```
odo('iris.csv', 'hdfs://hostname:iris.csv')
```

```
odo('hive://hostname/default::iris_csv',
    'hive://hostname/default::iris_parquet',
     stored_as='PARQUET', external=False)
```

odo

How do I go from X to Y in the most efficient way ...



JSON in S3 » postgres

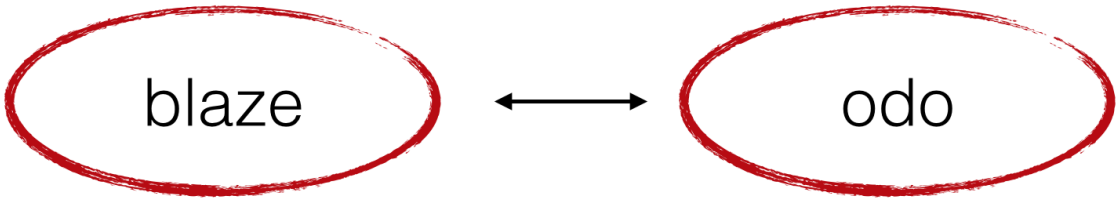
```
JSON S3 » Local temp file
boto.get_bucket().get_contents_to_filename()

Local temp file » DataFrame
pandas.read_json()

DataFrame » CSV
DataFrame.to_csv()

CSV » postgres
copy t from '/path/to/file.csv'
with
    delimiter ','
    header TRUE
```

```
odo('s3://mybucket/path/to/data.json', 'postgresql://user:passwd@localhost:port/db::data')
```



Blaze depends on odo to handle URIs.

```
iris = Data('iris.csv')
```

```
iris = Data('sqlite:///flowers.db::iris')
```

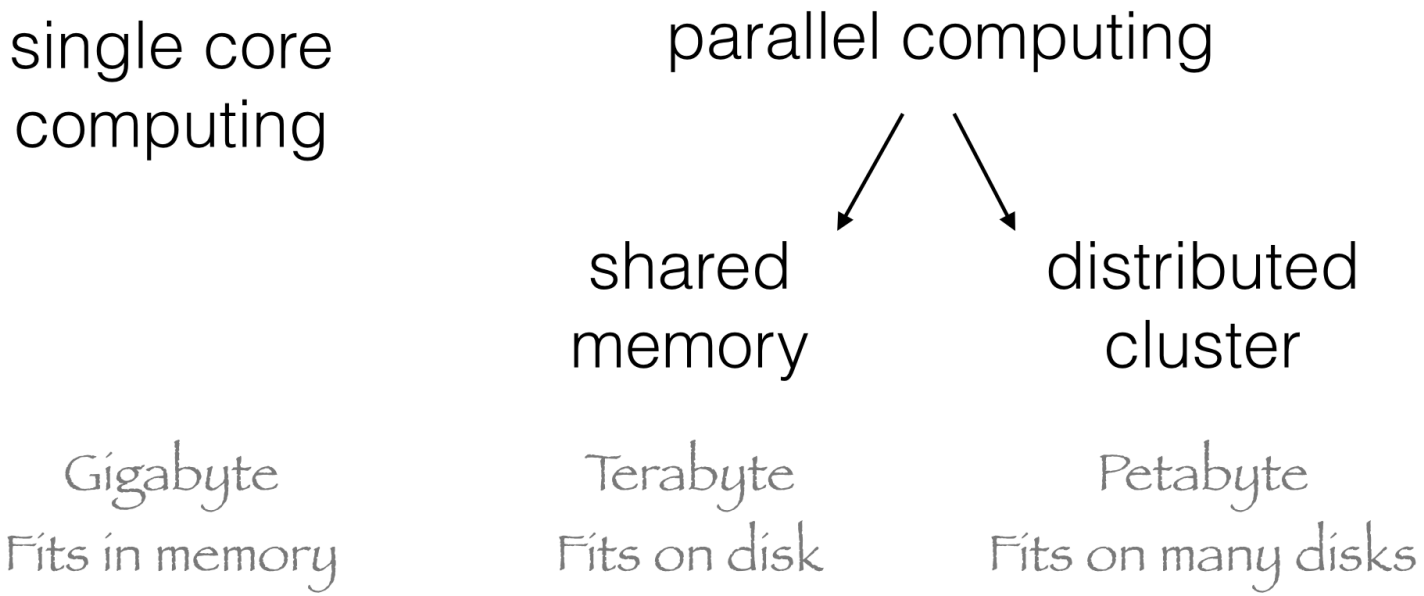
```
odo('iris.json', 'sqlite:///flowers.db::iris')
```

dask

dask

enables parallel computing

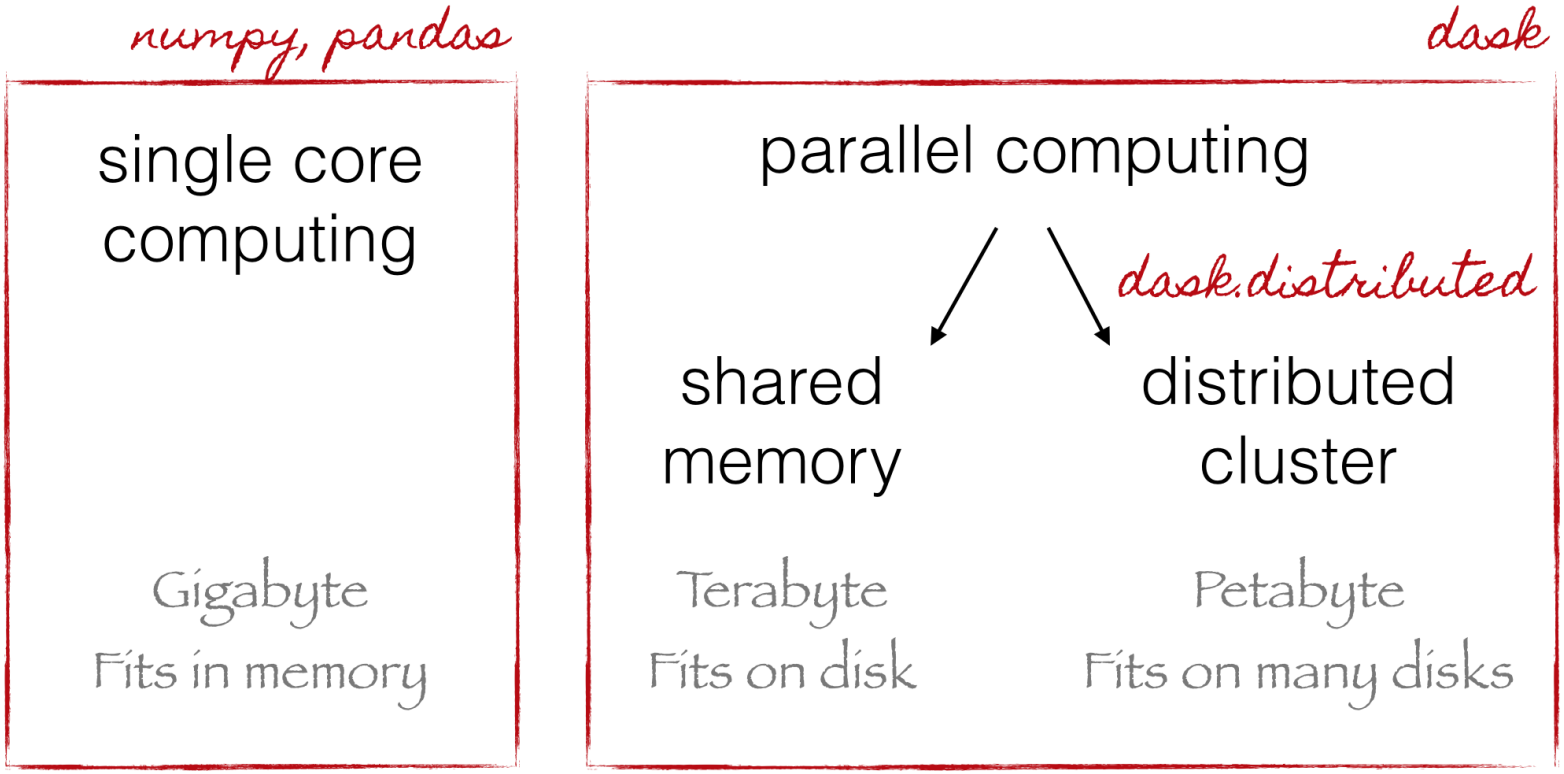
http://dask.pydata.org/en/latest/



dask

enables parallel computing

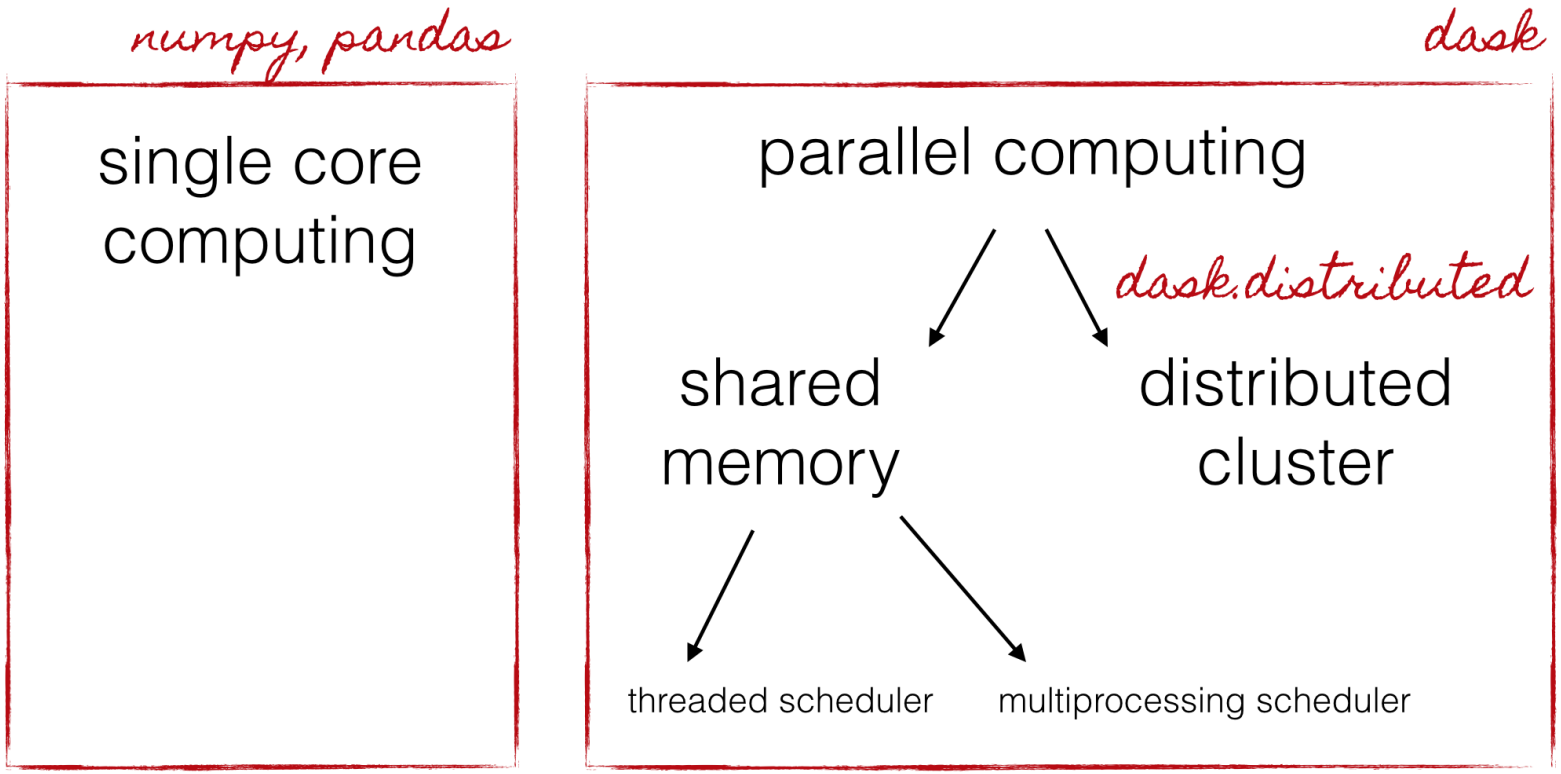
http://dask.pydata.org/en/latest/



dask

enables parallel computing

http://dask.pydata.org/en/latest/



dask array

numpy

```
>>> import numpy as np

>>> np_ones = np.ones((5000, 1000))


>>> np_ones

array([[ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       ...,
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.]])

>>> np_y = np.log(np_ones + 1)[:5].sum(axis=1)

>>> np_y

array([ 693.14718056,  693.14718056,
693.14718056,  693.14718056, 693.14718056])
```

dask

```
>>> import dask.array as da

>>> da_ones = da.ones((5000000, 1000000),
                      chunks=(1000, 1000))

>>> da_ones.compute()

array([[ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       ...,
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.],
       [ 1.,  1.,  1., ...,  1.,  1.,  1.]])

>>> da_y = da.log(da_ones + 1)[:5].sum(axis=1)

>>> np_da_y = np.array(da_y) #fits in memory

array([ 693.14718056,  693.14718056,
693.14718056,  693.14718056, …, 693.14718056])

# Result doesn't fit in memory
>>> da_y.to_hdf5('myfile.hdf5', 'result')
```

dask dataframe

pandas

```
>>> import pandas as pd

>>> df = pd.read_csv('iris.csv')

>>> df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
>>> max_sepal_length_setosa = df[df.species ==
'setosa'].sepal_length.max()

5.7999999999999998
```

dask

```
>>> import dask.dataframe as dd

>>> ddf = dd.read_csv('*.csv')

>>> ddf.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
…					

```
>>> d_max_sepal_length_setosa = ddf[ddf.species ==
'setosa'].sepal_length.max()

>>> d_max_sepal_length_setosa.compute()

5.7999999999999998
```

dask bag

semi-structure data, like JSON blobs or log files

```
>>> import dask.bag as db
>>> import json

# Get tweets as a dask.bag from compressed json files
>>> b = db.from_filenames('*.json.gz').map(json.loads)

# Take two items in dask.bag
>>> b.take(2)

({u'contributors': None,
  u'coordinates': None,
  u'created_at': u'Fri Oct 10 17:19:35 +0000 2014',
  u'entities': {u'hashtags': [],
   u'symbols': [],
   u'trends': [],
   u'urls': [],
   u'user_mentions': []},
  u'favorite_count': 0,
  u'favorited': False,
  u'filter_level': u'medium',
  u'geo': None …

# Count the frequencies of user locations
>>> freq = b.pluck('user').pluck('location').frequencies()

# Get the result as a dataframe
>>> df = freq.to_dataframe()
>>> df.compute()
       0      1
0             20916
1      Natal  2
2      Planet earth. Sheffield.    1
3      Mad, USERA    1
4      Brasilia DF - Brazil 2
5      Rondonia Cacoal      1
6      msftsrep || 4/5.     1
```

dask distributed

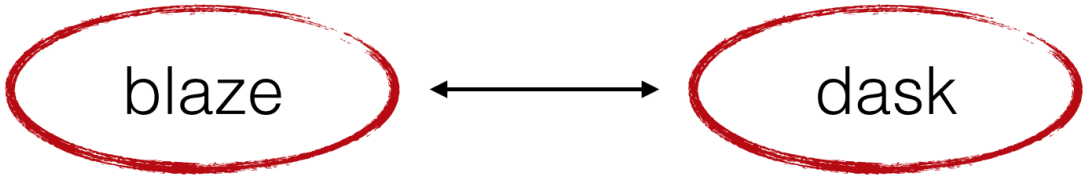
```
>>> import dask
>>> from dask.distributed import Client

# client connected to 50 nodes, 2 workers per node.
>>> dc = Client('tcp://localhost:9000')
# or
>>> dc = Client('tcp://ec2-XX-XXX-XX-XXX.compute-1.amazonaws.com:9000')
>>> b = db.from_s3('githubarchive-data', '2015-*.json.gz').map(json.loads)

# use default single node scheduler
>>> top_commits.compute()

# use client with distributed cluster
>>> top_commits.compute(get=dc.get)

[(u'mirror-updates', 1463019),
 (u'KenanSulayman', 235300),
 (u'greatfirebot', 167558),
 (u'rydnr', 133323),
 (u'markkcc', 127625)]
```



dask can be a backend/engine for blaze

e.g. we can drive dask arrays with blaze.

```
>>> x = da.from_array(...)                    # Make a dask array

>>> from blaze import Data, log, compute

>>> d = Data(x)                               # Wrap with Blaze
>>> y = log(d + 1)[:5].sum(axis=1)            # Do work as usual

>>> result = compute(y)                       # Fall back to dask
```

Developer Resources

blaze

- Blaze in the Real World, PyData Dallas 2015, Phillip Cloud

odo

- Blaze + Odo, SciPy 2015, Phillip Cloud
- Odo - Shape Shifting Data, PyData Dallas 2015, Ben Zaitlen

dask

- Dask, Out of core NumPy/Pandas through Task Scheduling, SciPy 2015, James Crist
- Dask Arrays or PyData's relationship with Parallelism, PyData Berlin 2015 Keynote, Matthew Rocklin

DyND

- Typing Arrays with DyND, SciPy 2015, Mark Wiebe

numba

- Accelerating Python with the Numba JIT Compiler, SciPy 2015, Stanley Seibert
- Odo - Shape Shifting Data, PyData Dallas 2015, Ben Zaitlen
- EuroPython 2015: Antoine Pitrou, Oscar Villellas, Graham Markall

bcolz

- Efficient memory/disk data containers with Python, EuroPython 2015, Francesc Alted
- New trends in storing large data silos with Python, EuroPython 2015, Francesc Alted

Summary

Goals

Rethink the term Data Science and explore connections between areas

Think in terms of data, expressions and engines

Encourage you start using any of the Blaze projects

Thanks to the Blaze Dream Team!

@FrancescAlted

bcolz

column store & query

@esc

datashape

data description language

@mwiebe

@izaid

DyND

dynamic, multidimensional arrays

Blaze

castra

column store

@mrocklin

dask

parallel computing

@eriknw

odo

data migration

@quasiben

blaze

interface to query data

@jcrist

@cowlicks

@cpcloud

Questions?

Slides:
http://chdoig.github.com/ep2015-blaze

Email: christine.doig@continuum.io

Twitter: ch_doig