Python and R, when in Rome

Jim Harner & Soren Harner, SDSS 2020

Origins

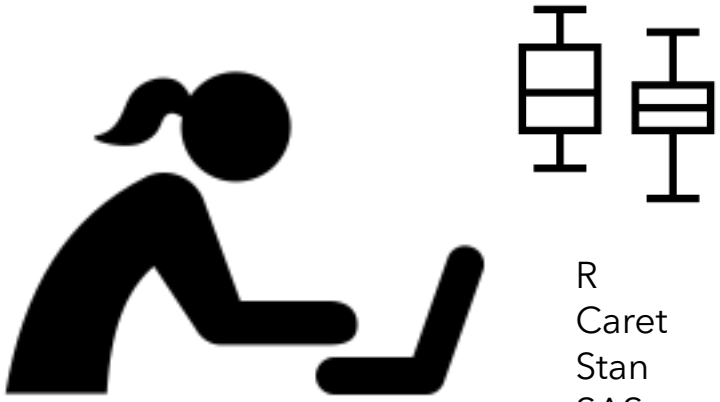
R

- First stable release in 2000
- An implementation of S in 1976
- Inspired by XLisp-Stat and Scheme
- Environment for statistical computing and graphics
- Statisticians, researchers, and analysts
- Written and extended in C/Fortran

Python

- Published to alt.sources in 1991
- Object-oriented and functional, inspired by SETL and Haskel
- Programming for everyone
- Extended with NumPy and Pandas for data science
- Software Developers
- Written and extended in C

Statistics and Natural Sciences



R
Caret
Stan
SAS

Business Intelligence

Excel
SQL
Tableau

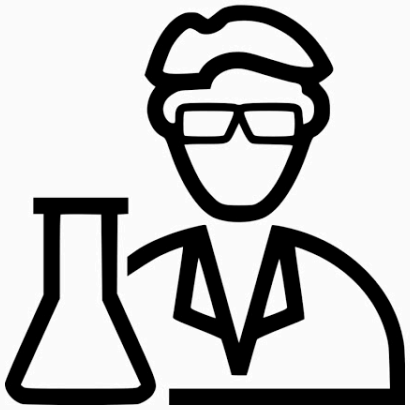
Data Science

Tidy Dataframes
Notebook workflows
Data pipelines
Vectorized Computation
Functional programming
SQL-like queries

Computer Science / AI

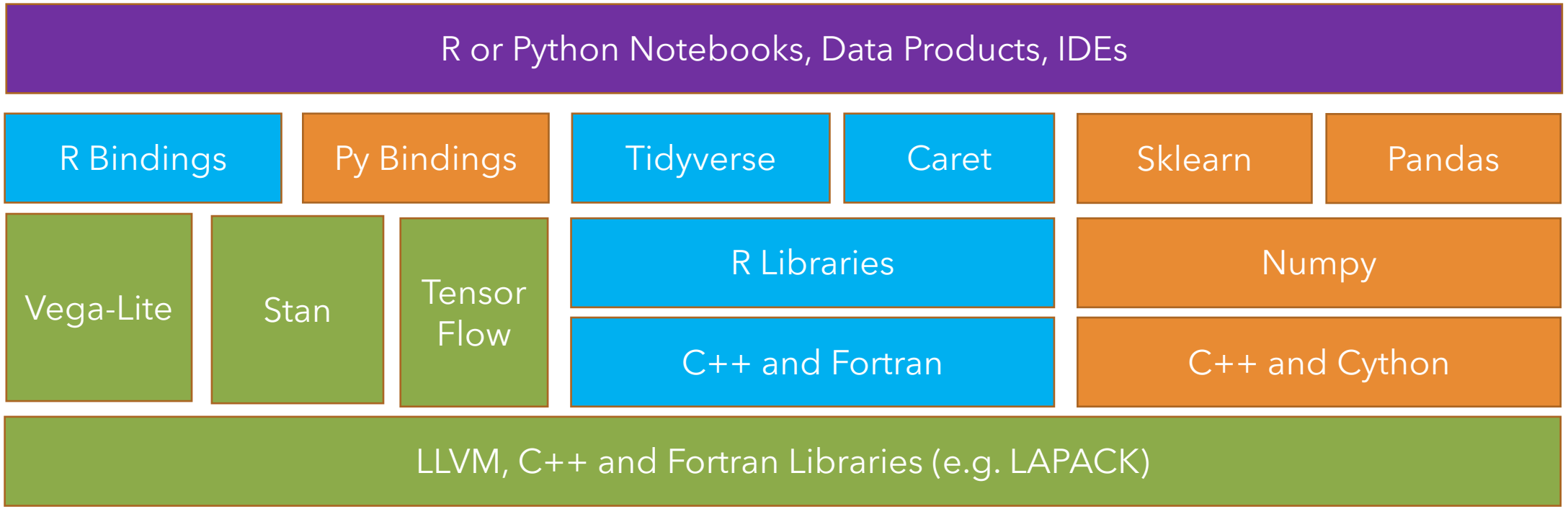
Python
Java
Hadoop/Spark
SciKit-Learn
Tensorflow
GPU/Cuda
Docker/ K8s

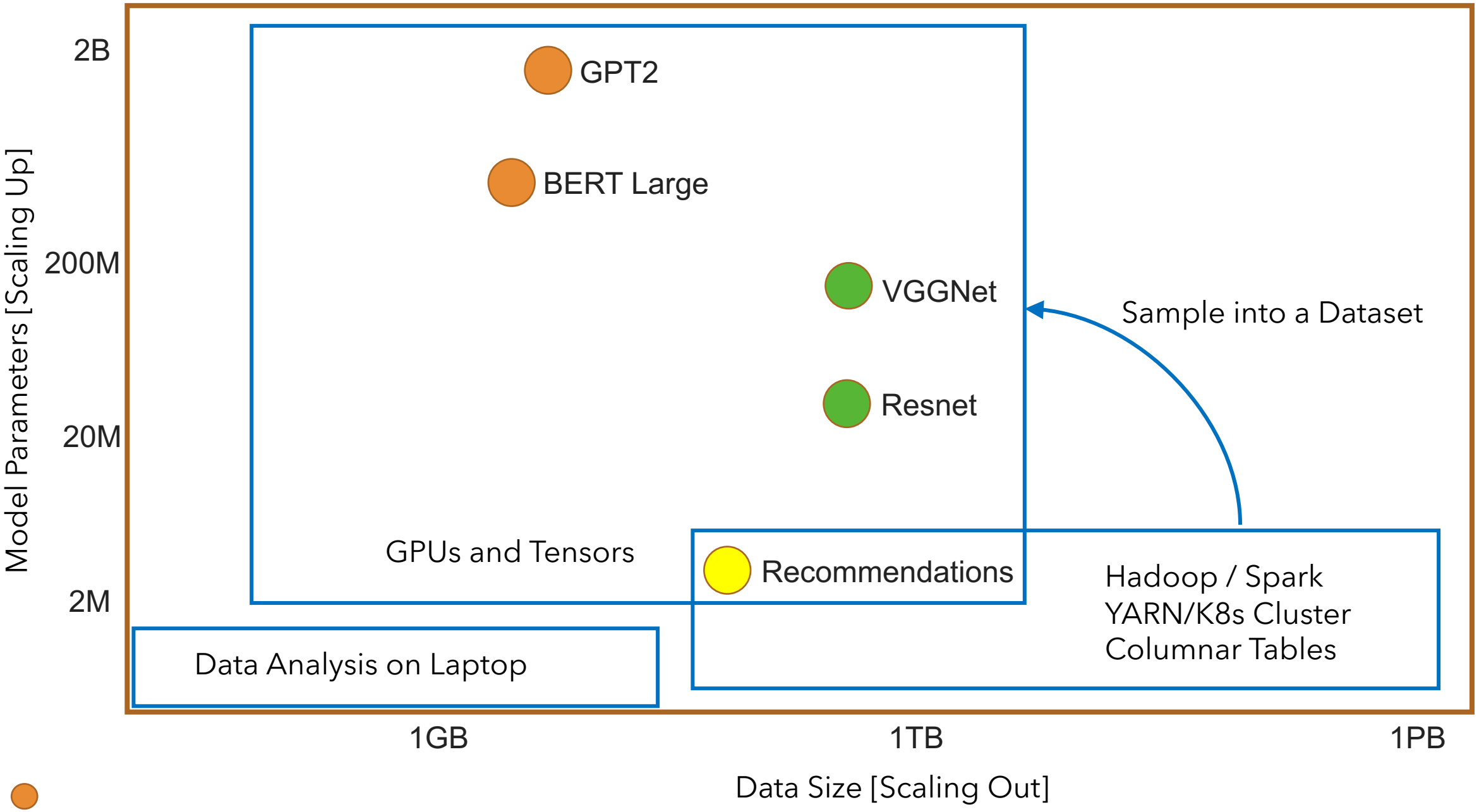
Physical Sci and Engineering



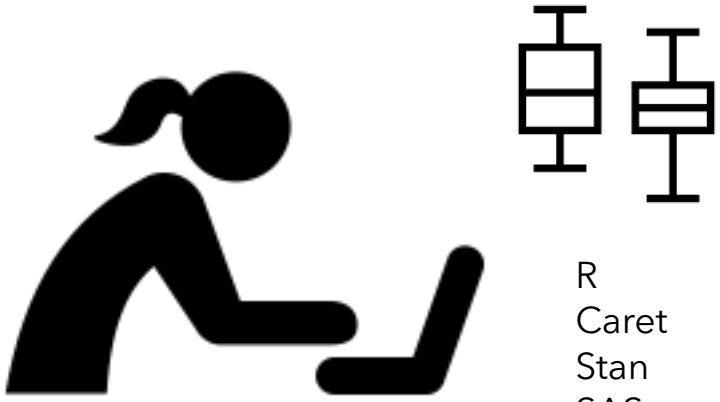
Matlab
Fortran
Python
Numpy
Scipy
Matplotlib

Sharing and Borrowing Code and Workflows




2B
200M
20M
2M
Model Parameters [Scaling Up]
GPT2
BERT Large
VGGNet
Resnet
Recommendations
Sample into a Dataset
GPUs and Tensors
Hadoop / Spark
YARN/K8s Cluster
Columnar Tables
Data Analysis on Laptop
1GB
1TB
1PB
Data Size [Scaling Out]

Statistics and Natural Sciences



R
Caret
Stan
SAS

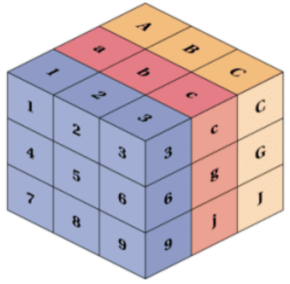
Computer Science / AI

Python
Java
Hadoop/Spark
SciKit-Learn
Tensorflow
GPU/Cuda
Docker/ K8s

Data Science and Analytics Engines

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

DataFrames
Dpylr, Pandas, Spark, Rapids
Based on Arrow



GPU Tensors and Graphs
TensorFlow, PyTorch, Rapids

Business Intelligence

Excel
SQL
Tableau

Physical Sci and Engineering

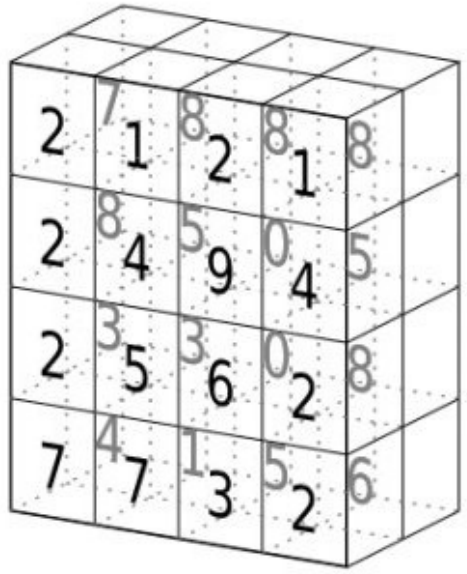
Matlab
Fortran
Python
Numpy
Scipy
Matplotlib

Tensors are the structure for DL and ML

- N-dimensional typed arrays
- Input and Output of PyTorch and TensorFlow
- Images, signals, series, encoded natural language
- The shape is a tuple with the tensor dimension
- Contiguous memory, except when sparse
- Mappable to GPU memory

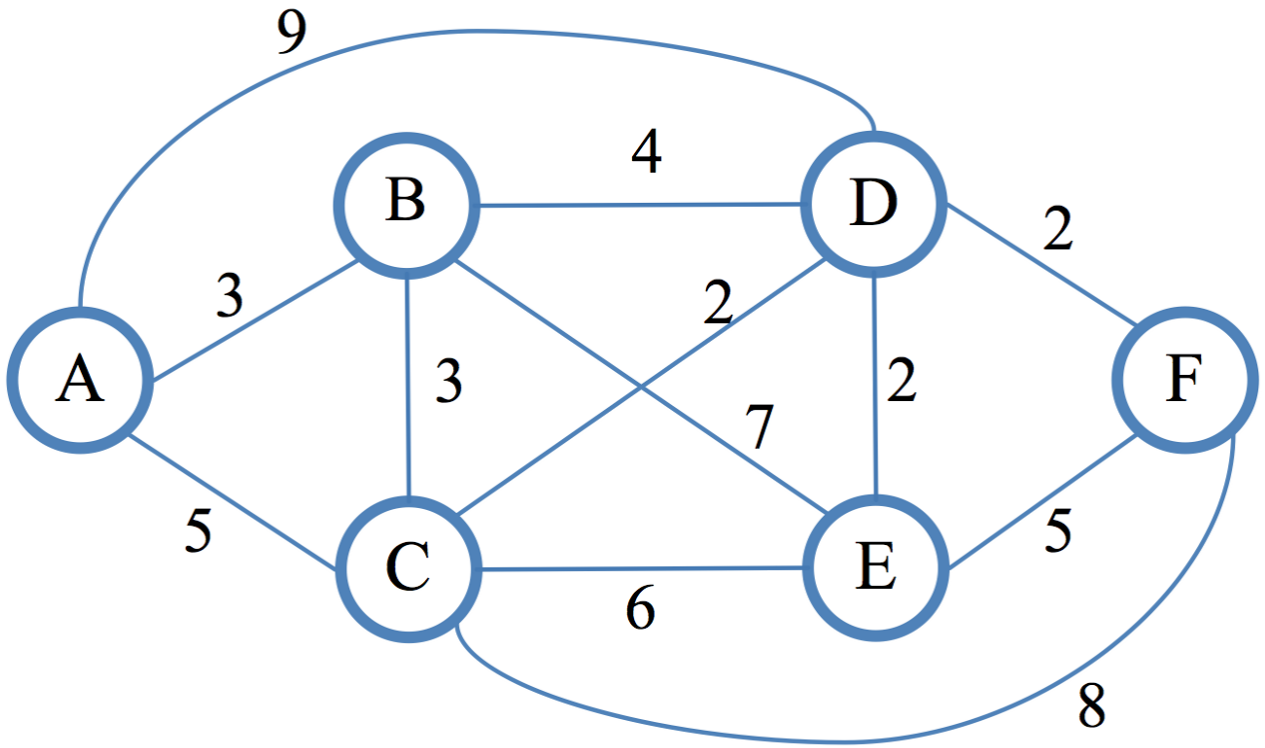
't'
'e'
'n'
's'
'o'
'r'

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4



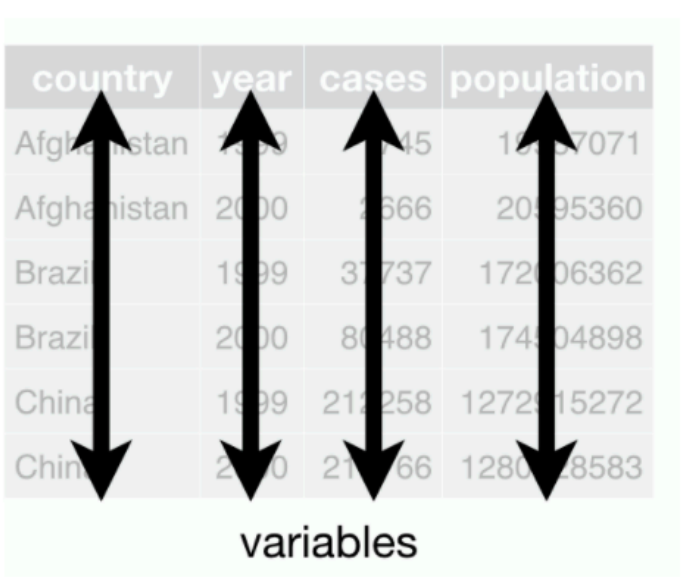
Graphs are the structure for relationships and connections

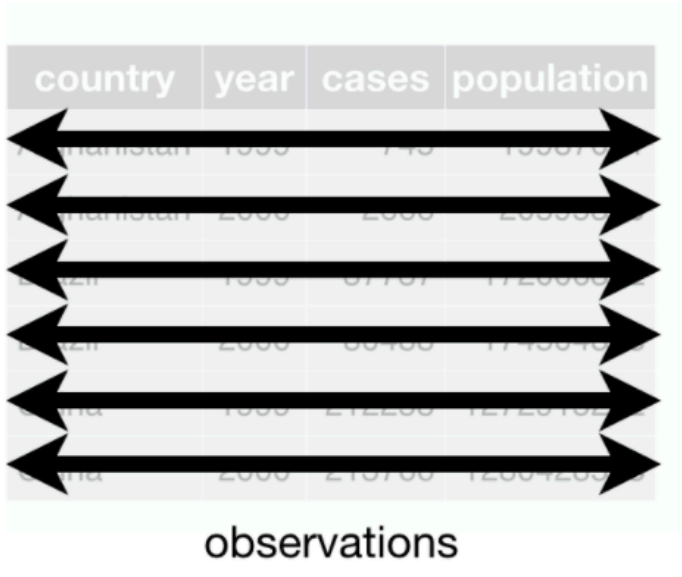
- Made of nodes and edges
- Edges have weights and types
- Nodes represent things
- Algorithms to quantity and visualize structure
- Planning, Fraud, Recommendations
- Probabilistic reasoning
- A tensor, possibly very sparse

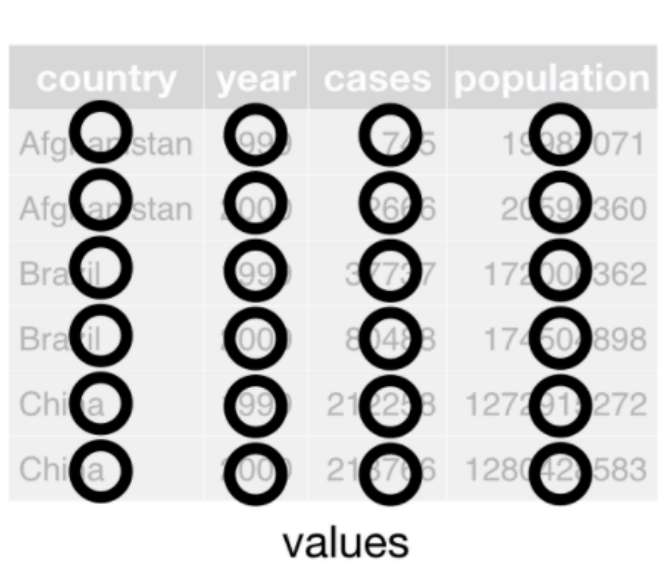


Data frames are the structure for rectangular data

- Each row is an observation
- Each column represent a variable and has a type (possibly hierarchical)
- Cells are values
- Uniform APIs for transformations and visualization
- Leverages columnar memory layout







Hadley Wickham, Tidy Data, http://www.jstatsoft.org/v59/i10/paper.

APACHE ARROW

Apache Arrow is a cross-platform specification for tensors and data frames

- Columnar In-Memory representation that can be broken into batches
- Data is organization for SIMD vectorization and GPU computation with “device” support
- Serialization and RPC without parsing based on Arrow IPC and Flight
- Pass pointers between languages, frameworks, and processes without copying with Plasma
- Integration with filesystems and on-disk formats, e.g. Parquet, with Arrow FS

Large multi-file datasets in Arrow

Load the Arrow and dplyr R packages

```
library(arrow, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)
```

Data in Parquet, a columnar format, partitioned

```
2009/01/data.parquet
2009/02/data.parquet
...
```

Load a directory of Parquet files

```
ds <- open_dataset("nyc-taxi", partitioning = c("year", "month"))
```

An Arrow data frame with portable types

```
ds
```

```
##
## ## FileSystemDataset with 125 Parquet files
## ## vendor_id: string
## ## pickup_at: timestamp[us]
## ## dropoff_at: timestamp[us]
## ## passenger_count: int8
## ## trip_distance: float
## ## pickup_longitude: float
## ## pickup_latitude: float
## ## rate_code_id: string
## ## store_and_fwd_flag: string
## ## dropoff_longitude: float
## ## dropoff_latitude: float
## ## payment_type: string
## ## fare_amount: float
## ## extra: float
## ## mta_tax: float
## ## tip_amount: float
## ## tolls_amount: float
## ## total_amount: float
## ## improvement_surcharge: float
## ## pickup_location_id: int32
## ## dropoff_location_id: int32
## ## congestion_surcharge: float
## ## year: int32
## ## month: int32
##
## See $metadata for additional Schema metadata
```

https://arrow.apache.org/docs/r/articles/dataset.html

Arrow integration with dplyr

```
system.time(ds %>%
  filter(total_amount > 100, year == 2015) %>%
  select(tip_amount, total_amount, passenger_count) %>%
  group_by(passenger_count) %>%
  collect() %>%
  summarize(
    tip_pct = median(100 * tip_amount / total_amount),
    n = n()
  ) %>%
  print())
```

```
##
## ## # A tibble: 10 x 3
## ##    passenger_count tip_pct      n
## ##              <int>   <dbl>  <int>
## ##  1               0    9.84    380
## ##  2               1   16.7  143087
## ##  3               2   16.6   34418
## ##  4               3   14.4    8922
## ##  5               4   11.4    4771
## ##  6               5   16.7    5806
## ##  7               6   16.7    3338
## ##  8               7   16.7      11
## ##  9               8   16.7      32
## ## 10               9   16.7      42
## ##
## ##    user  system elapsed
## ##   4.436   1.012   1.402
```

This is an operation on 2 billion rows under 2 seconds on a laptop.

Data is pulled into memory on `collect`. Until then, evaluation is lazy, allowing filtering and selecting to small slices, even ignoring some files.

Programming Languages with Arrow

Full Support

C++, Python, Java

Specific Use-Cases

R, C#, Gandiva, Go, JavaScript, MATLAB, Ruby, Rust

Efficient memory layout

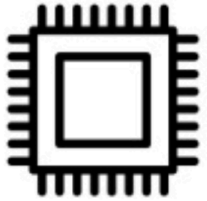
	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Traditional Memory Buffer

Row 1	1331246660
	3/8/2012 2:44PM
	99.155.155.225
Row 2	1331246351
	3/8/2012 2:38PM
	65.87.165.114
Row 3	1331244570
	3/8/2012 2:09PM
	71.10.106.181
Row 4	1331261196
	3/8/2012 6:46PM
	76.102.156.138

Arrow Memory Buffer

session_id	1331246660
	1331246351
	1331244570
	1331261196
timestamp	3/8/2012 2:44PM
	3/8/2012 2:38PM
	3/8/2012 2:09PM
	3/8/2012 6:46PM
source_ip	99.155.155.225
	65.87.165.114
	71.10.106.181
	76.102.156.138



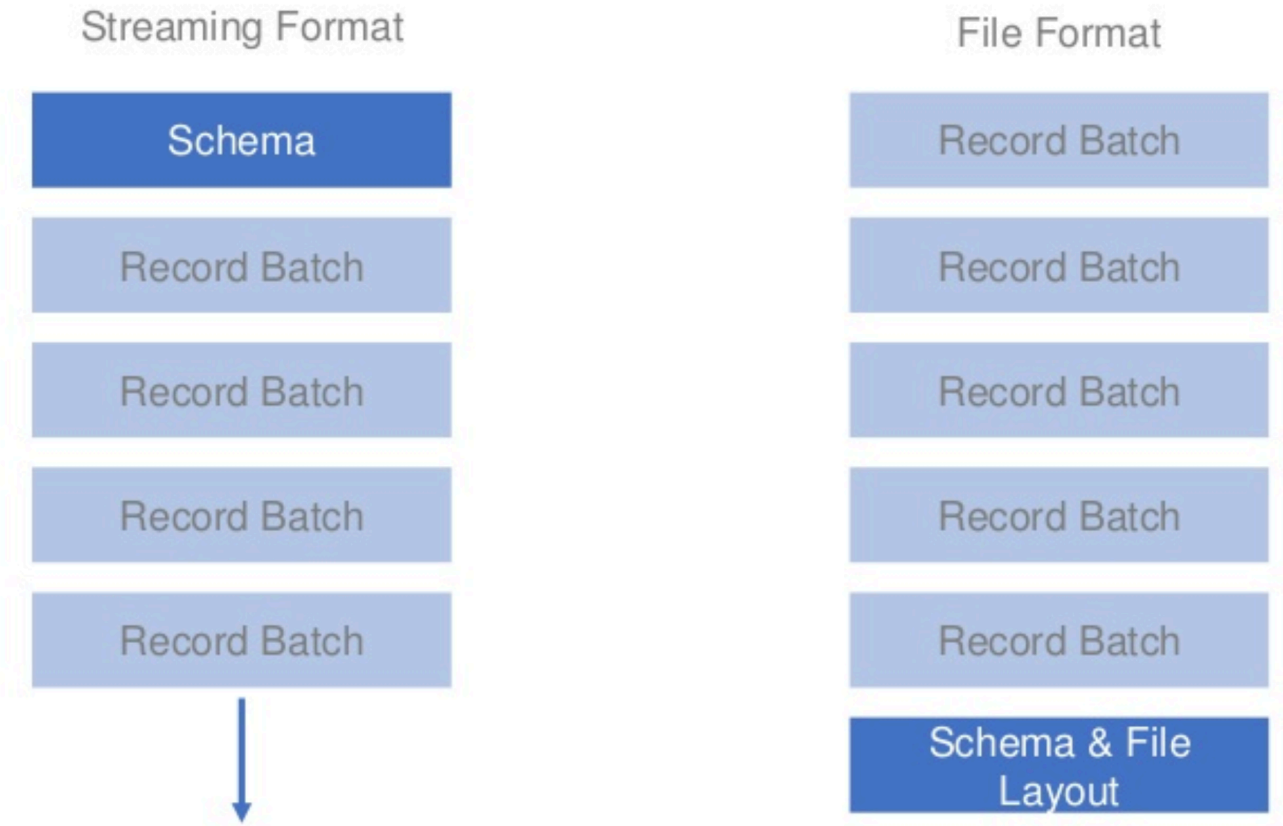
Arrow leverages the data parallelism (SIMD) in modern Intel CPUs:

```
SELECT * FROM clickstream WHERE
session_id = 1331246351
```

dremio

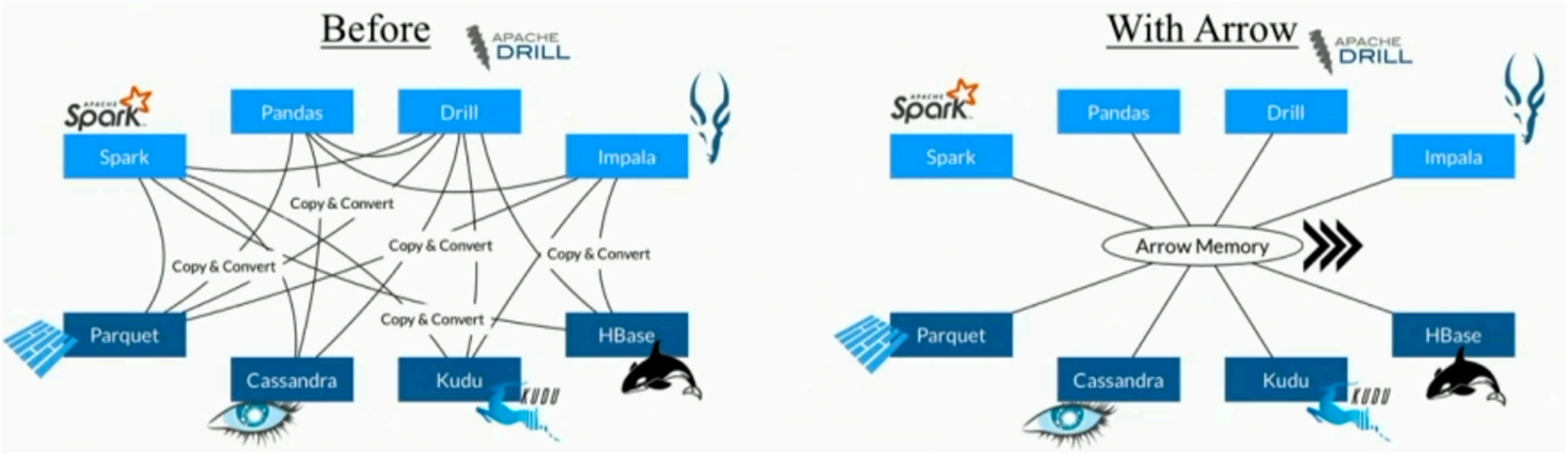
Scaling Out

Streaming and Persisting



https://www.slideshare.net/dremio/building-a-virtual-data-lake-with-apache-arrow

Integration minimizing copies and parsing



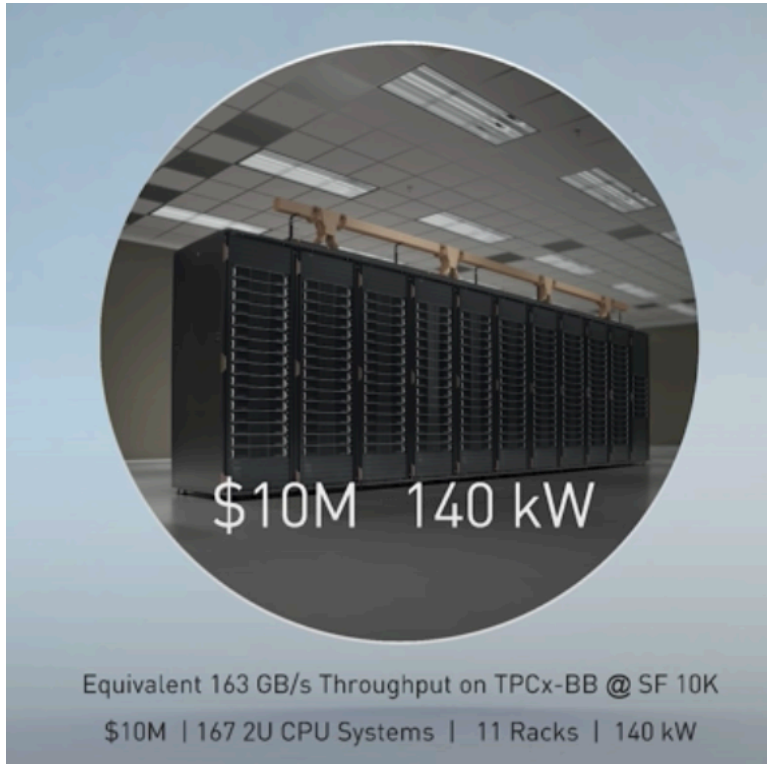
RAPIDS

Scale up data frames based on Arrow, such as group by, sort, summarize, etc.

Accelerate graph algorithms, such connected components, cores, shortest path, etc.

GPU ML implementations for clustering, regression, classification

Cluster with Spark on GPUs, 5x the price performance as the prior SOTA on TPCx-BB



Pandas-like Data Frames on GPU

```
import cudf, io, requests
from io import StringIO

url = "https://github.com/plotly/datasets/raw/master/tips.csv"
content = requests.get(url).content.decode('utf-8')

tips_df = cudf.read_csv(StringIO(content))
tips_df['tip_percentage'] = tips_df['tip'] / tips_df['total_bill'] * 100

# display average tip by dining party size
print(tips_df.groupby('size').tip_percentage.mean())
```

E.g. Sorting 1B records in 4 seconds
on 2 x V100 GPUs on Dask

XGBoost on the GPU

```
# instantiate params
params = {}

# general params
general_params = {'silent': 1}
params.update(general_params)

# booster params
n_gpus = 1  # change this to -1 to use all GPUs available or 0
booster_params = {}

if n_gpus != 0:
    booster_params['tree_method'] = 'gpu_hist'
    booster_params['n_gpus'] = n_gpus
params.update(booster_params)

# learning task params
learning_task_params = {}
if classification:
    learning_task_params['eval_metric'] = 'auc'
    learning_task_params['objective'] = 'binary:logistic'
else:
    learning_task_params['eval_metric'] = 'rmse'
    learning_task_params['objective'] = 'reg:squarederror'
params.update(learning_task_params)
print(params)
```

```
# model training settings
evallist = [(dvalidation, 'validation'), (dtrain, 'train')]
num_round = 100
```

```
bst = xgb.train(params, dtrain, num_round, evallist)
```

Feeding Models

Arrow as common format for Deep Learning frameworks

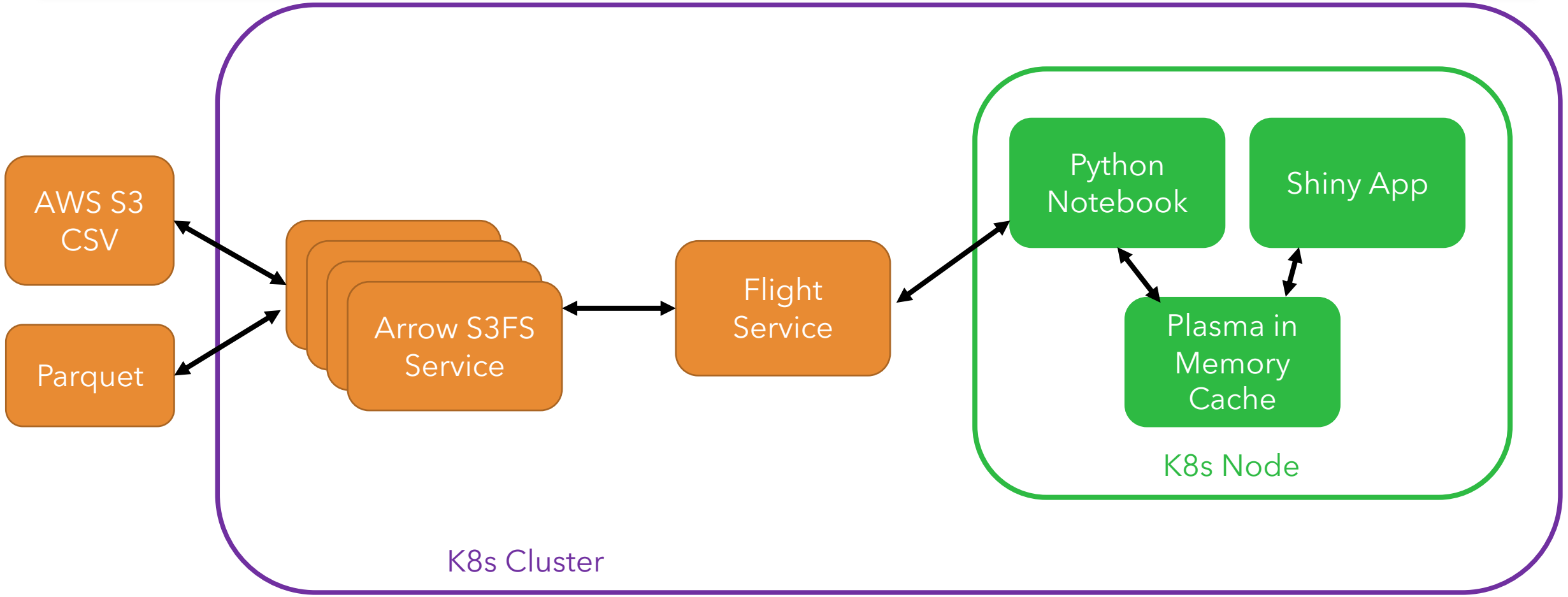
Tensors, not data frames, representing images, encoded text, etc., not typed columns

DL frameworks are expanding into probabilistic programming and HPC

TensorFlow

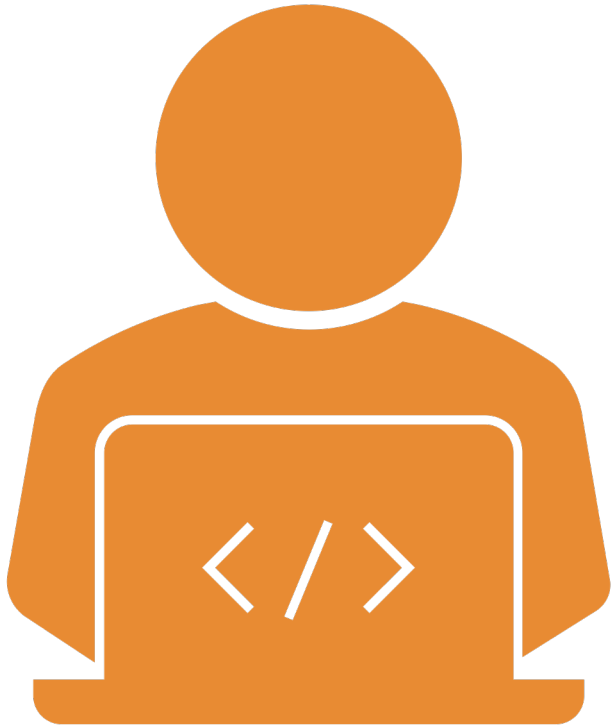
PyTorch

Running Python and R workloads



R & Python, when in Rome…

- Data analysis and statistics, mostly R
- Deep Learning, software development, and data engineering, mostly Python
- Build on the same foundation with compatible data structures, thanks to Arrow's data frames, to achieve scale



Or something else?

- Swift for TensorFlow is fast, safe, provides differential programming with LLVM support
- Julia has appeal of Matlab and speed of C with built in parallelism
- Rust implements a distributed database based on Arrow