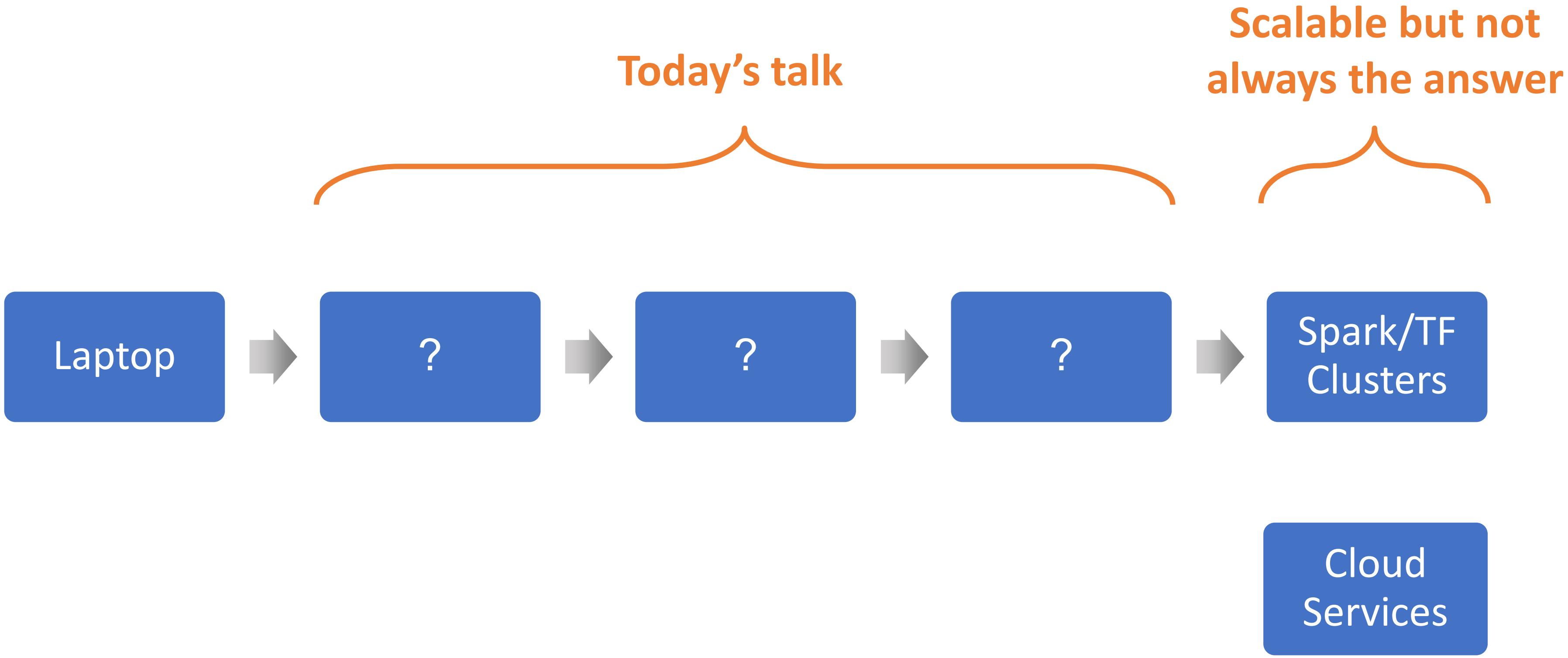
Scaling Machine Learning

Razvan Peteanu

Lead Architect – Machine Learning, TD Securities

linkedin.com/in/rpeteanu

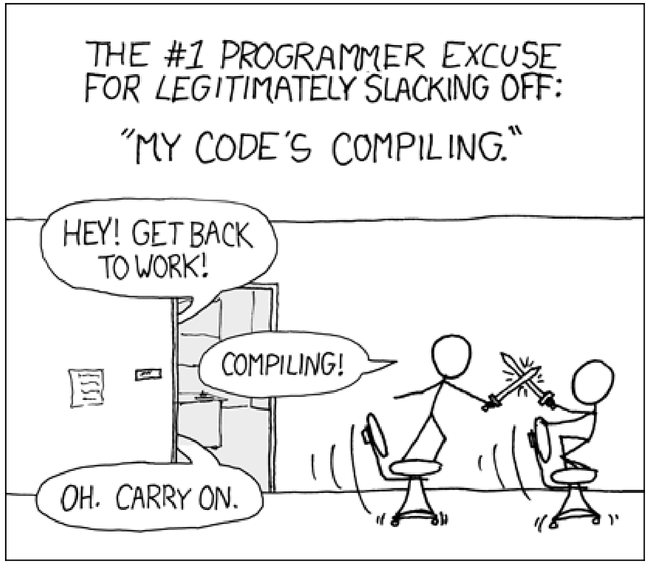
Choices



Today's talk
Scalable but not always the answer
Laptop
?
?
?
Spark/TF Clusters
Cloud Services

What do we need to scale?

Data Transformation → Model Search → Training → Prediction



Credit: xkcd.com

Python is developer-friendly but …

why Python is so

why python is so **popular**

why python is so **slow** → **Why Python is Slow: Looking Under the Hood**
Jake VanderPlas [1]

why python is so **powerful**

why python is so **popular with developers**

why python is so **popular for machine learning**

“My rule of thumb for pandas is that you should have 5 to 10 times as much RAM as the size of your dataset”

Apache Arrow and the "10 Things I Hate About pandas“
Wes McKinney, pandas’ author [2]

Scalability doesn’t come for free

Scale up machine → **Scale ‘out’ within machine** → **Scale out across machines**

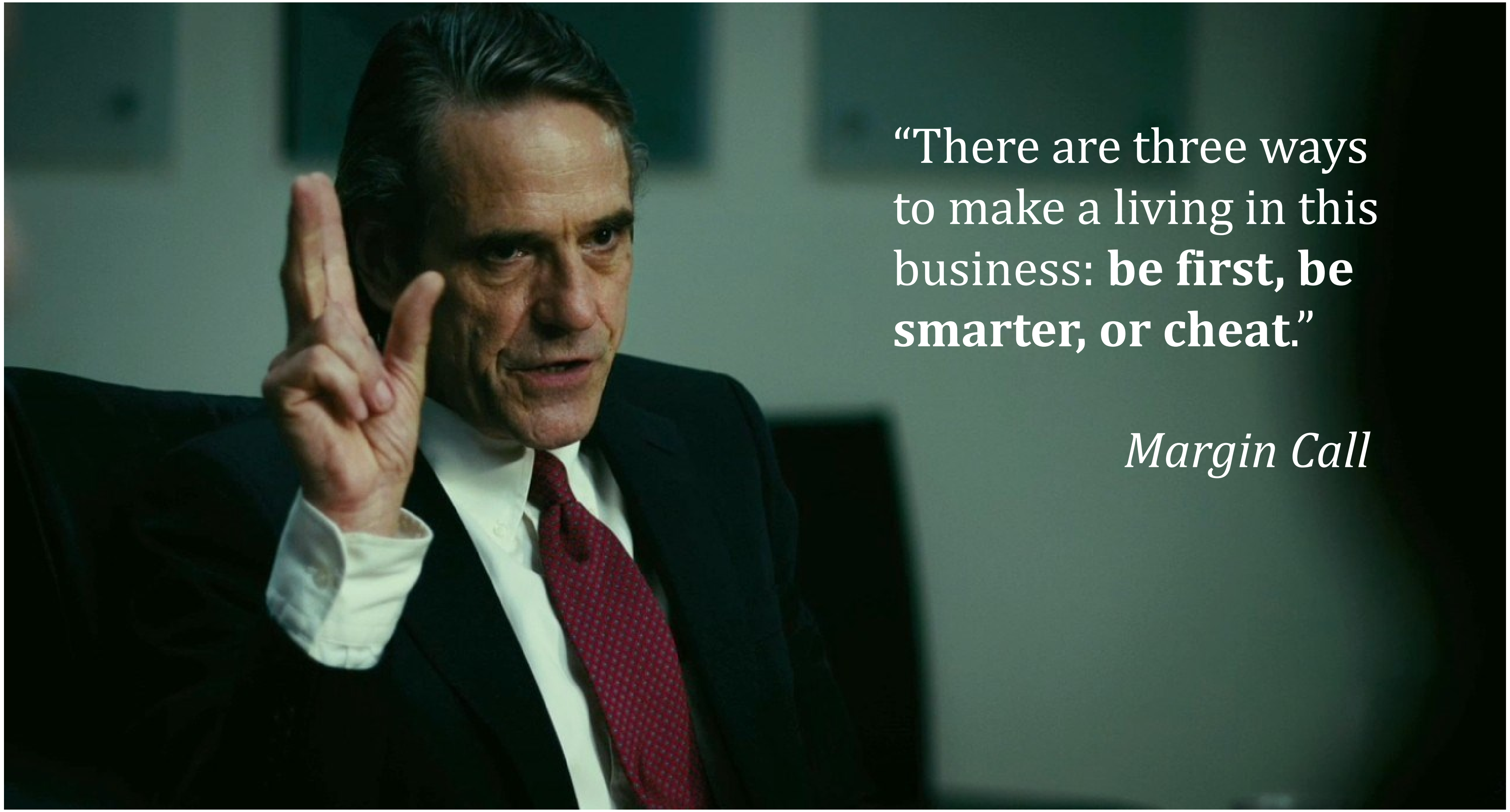
Scale up machine
Faster {CPU, I/O}
More RAM

Scale ‘out’ within machine
More {CPUs, GPUs}

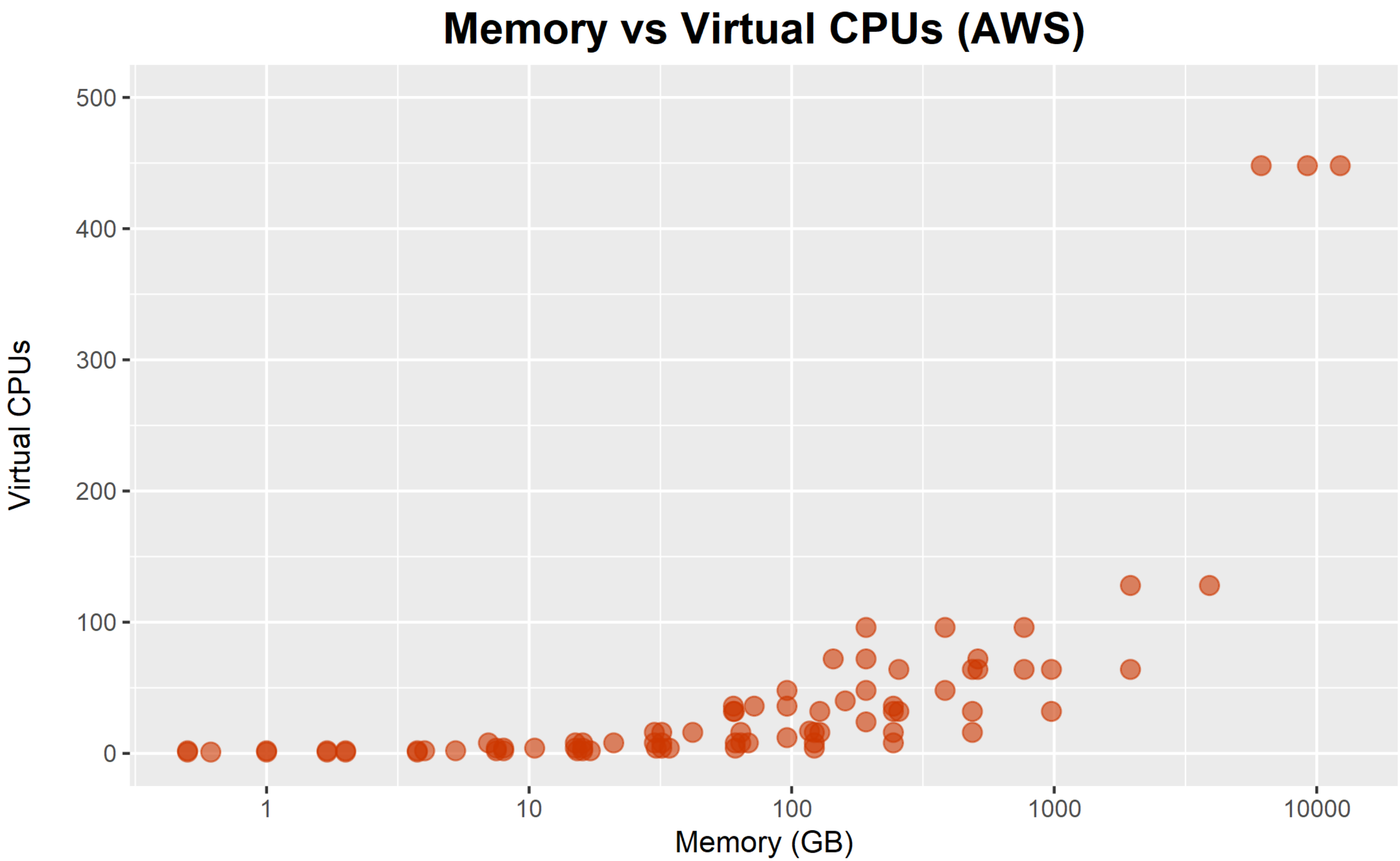
Scale out across machines
Distribute computation
Distribute {RAM, I/O, F/S}
Observability

Complexity & Latency

But do we have to scale?

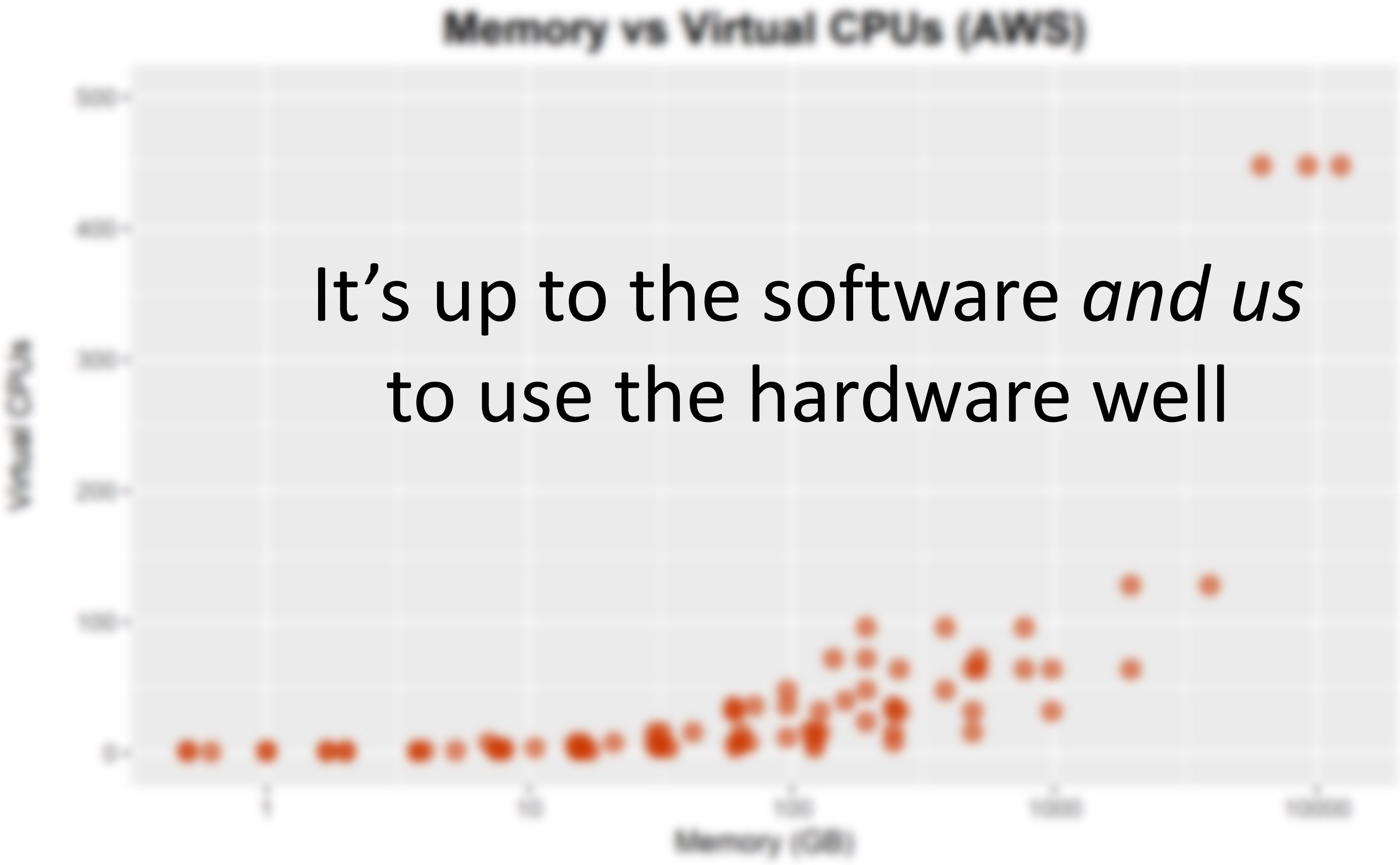


Can we scale CPU and RAM independently?



Data from https://ec2instances.info

Cores are a terrible thing to waste

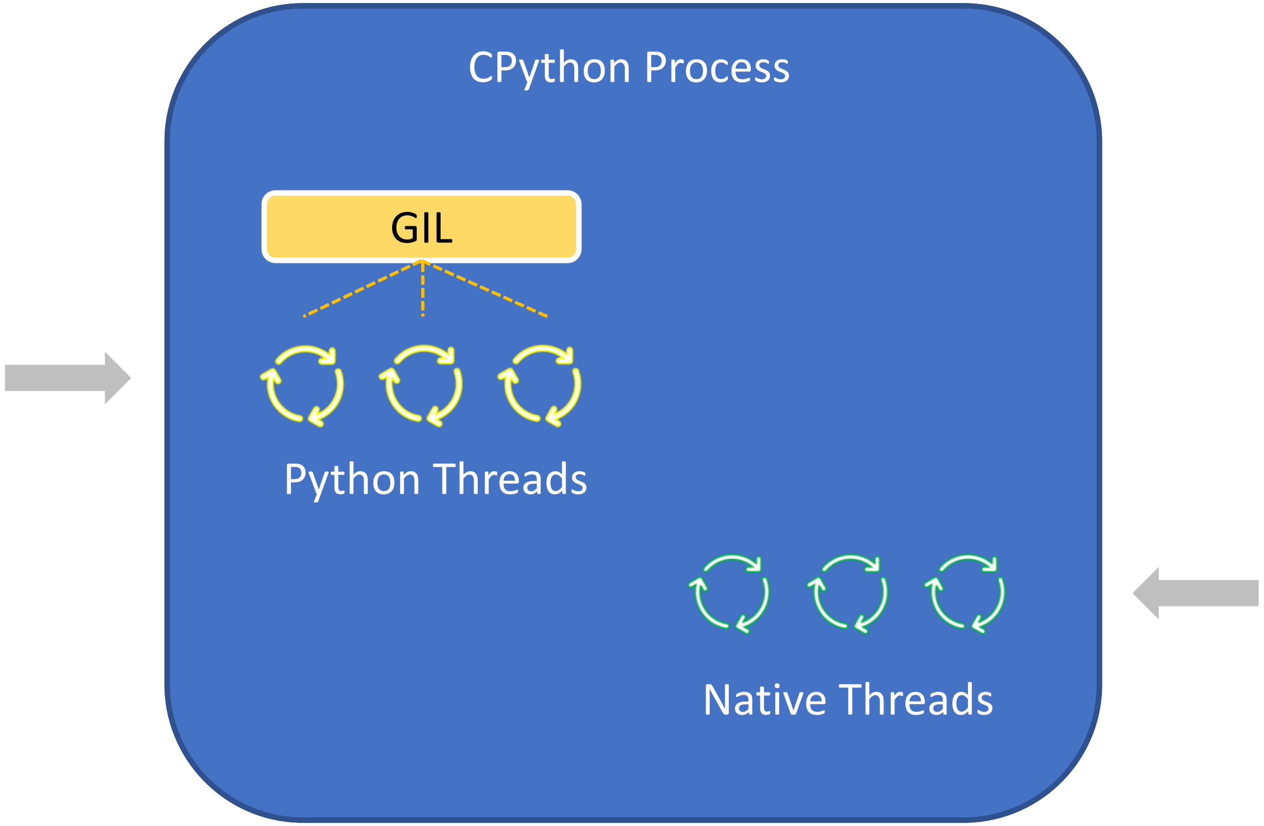


It's up to the software *and us* to use the hardware well

Threads, Processes, Machines – and `joblib` to bind them all

Python code

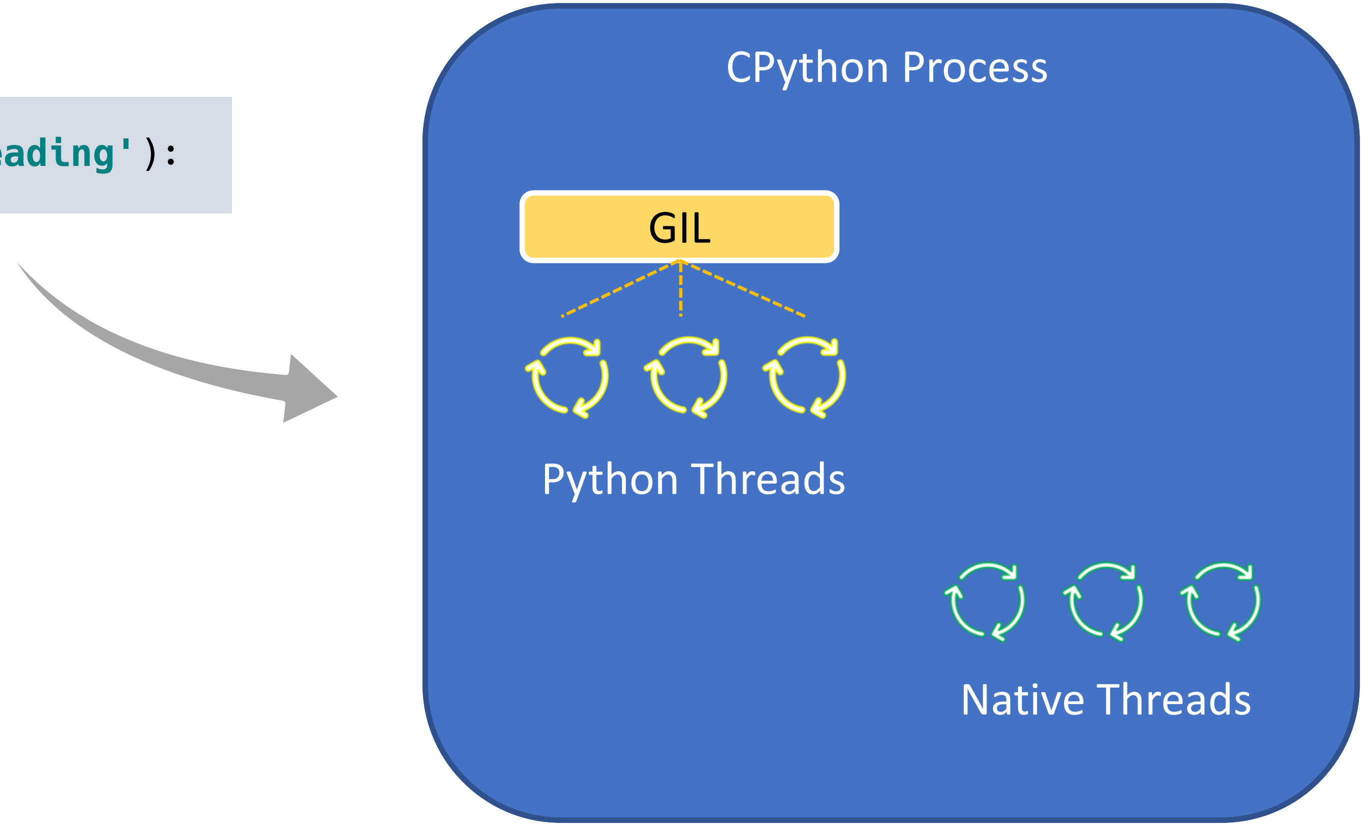
Strings
Lists,
Dictionaries,
…



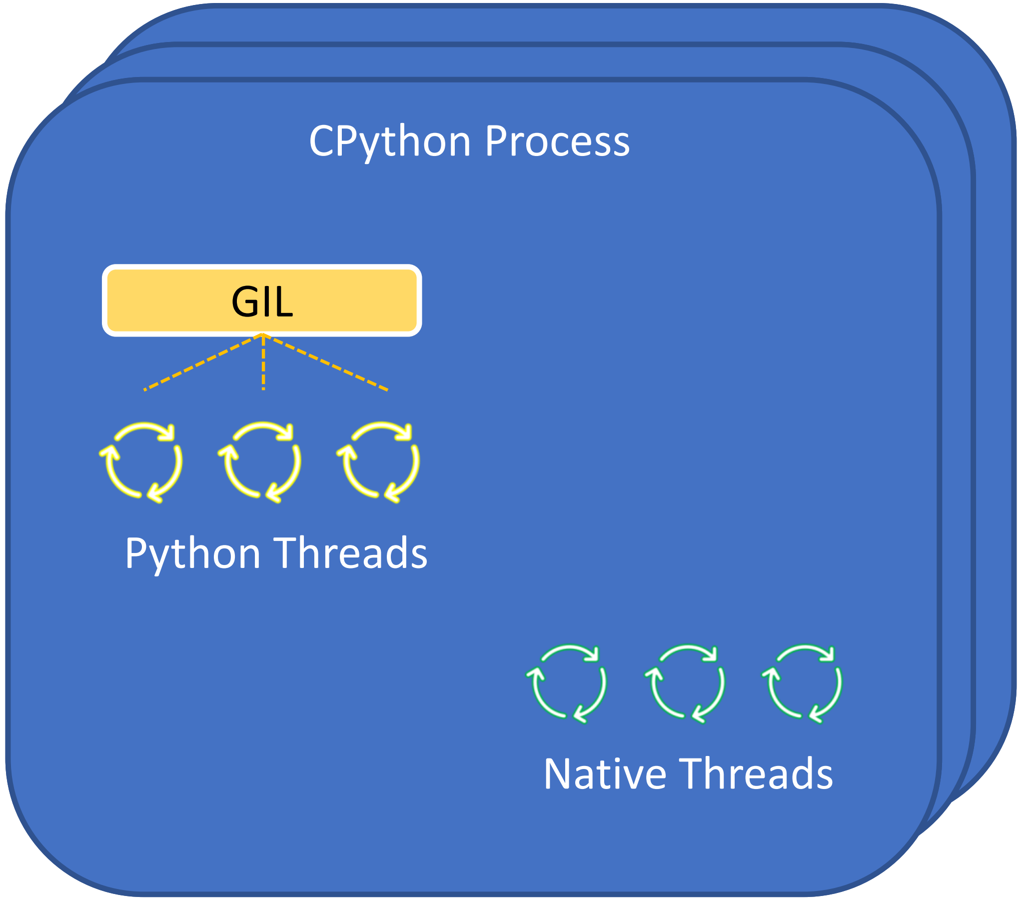
Native code:
NumPy,
Scikit-Learn (numeric data only),
Numba, …

Threads, Processes, Machines – and `joblib` to bind them all

```
with joblib.parallel_backend('threading'):
```



Threads, Processes, Machines – and `joblib` to bind them all




```python
with joblib.parallel_backend('loky'):
```

or

```python
with joblib.parallel_backend('multiprocessing'):
```

or

A better option in a few slides

Parallelism in practice

pandas:

- No built-in parallelization
- Numerical data? `@numba.jit(nopython=True, parallel=True)`
- Non-numerical data? We'll need processes because of the GIL

Parallelism in practice

pandas:

- No built-in parallelization
- Numerical data?
- Non-numerical data?

`@numba.jit(nopython=True, parallel=True)`

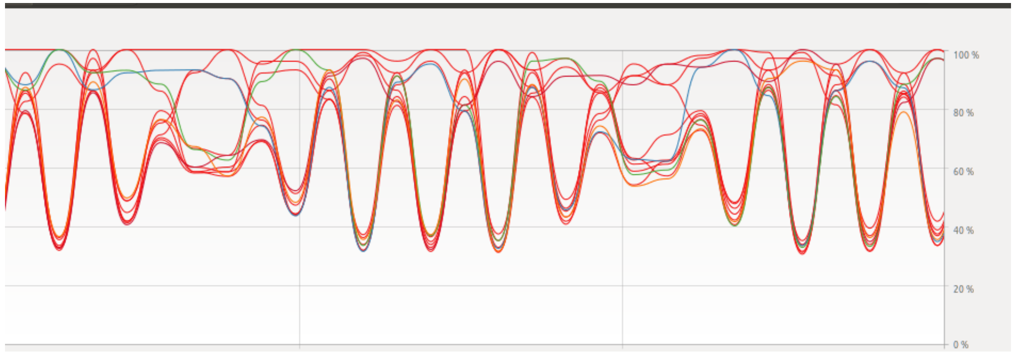
We'll need processes because of the GIL

scikit-learn:

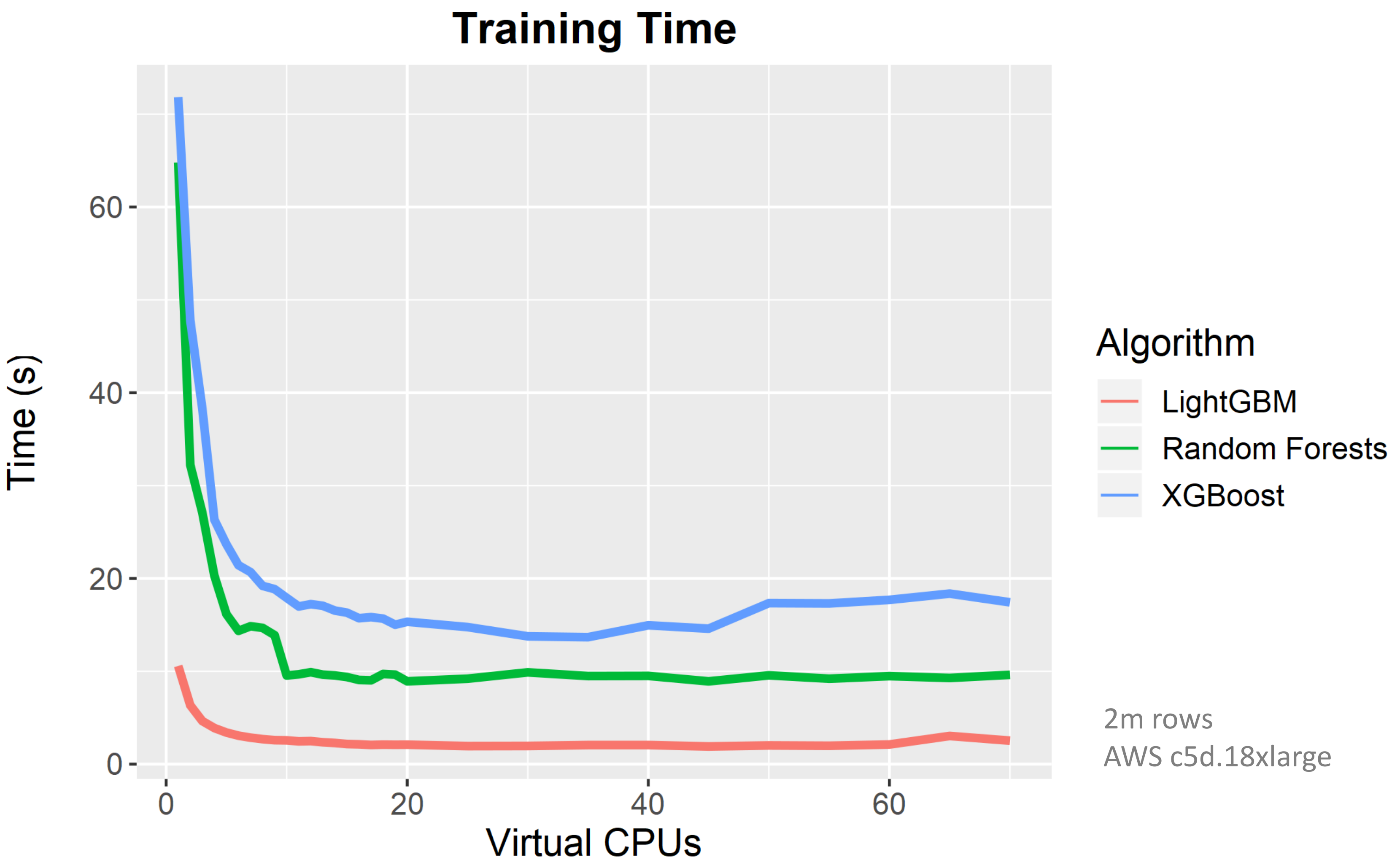
- Parallelizable estimators

```
RandomForestClassifier(n_jobs=-1)
XGBClassifier(n_jobs=os.cpu_count())
```

- Is it enough to saturate the CPU?



Amdahl's Law



2m rows
AWS c5d.18xlarge

Why observability matters
(and why benchmarks are hard)

"Any sufficiently complex system acts as a black box when it becomes easier to experiment with than to understand" – Google Vizier paper [3]

- Network bandwidth, latency & saturation
- File system bandwidth & latency
- Disk speed
- CPU/GPU utilization & waits
- Bus bandwidth
- Real core vs vCPU
- Algorithm implementation

Beyond a single machine (without Spark)

DASK

Written in for

H_2O.ai

Written in Java for Java

Large clusters ✔

Adaptive scaling ✔

Open source ✔

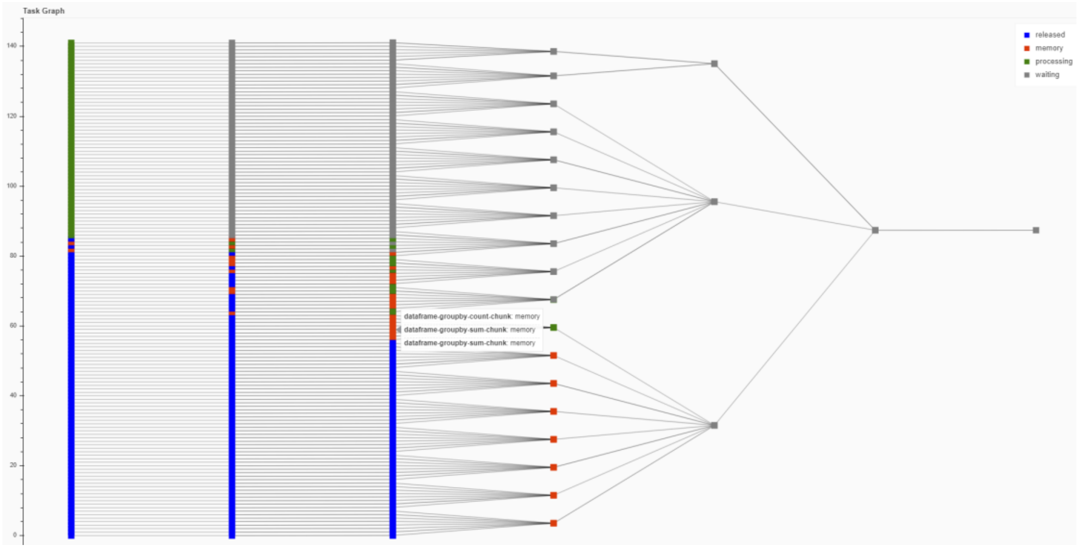
Parallelizing pandas

DASK

```python
from dask.distributed import Client
import dask.dataframe as dd

client = Client()  # starts local cluster

df = dd.read_csv('data*.csv')
df = df.groupby('some column').mean().compute()
```



Parallelizing scikit-learn

Training (data fits in RAM):

```
from dask.distributed import Client
import joblib

client = Client()  #starts local cluster

with joblib.parallel_backend('dask.distributed'):
    # scikit-learn code goes here
```

Predictions:

```
from dask_ml.wrappers import ParallelPostFit
clf = ParallelPostFit(RandomForestClassifier(…), scoring='accuracy')
```

Data larger than RAM?

Distribute `partial_fit()`-compatible scikit-learn estimators:

```
from sklearn.ensemble import RandomForestClassifier
from dask_ml.wrappers import Incremental

c = RandomForestClassifier()
inc = Incremental (c, scoring='accuracy')
inc.fit (X, y) # calls partial_fit()
```

Built-in distributed estimators in Dask-ML (GLM, clustering)

Support for distributed XGBoost

Hyper-Parameter Optimization

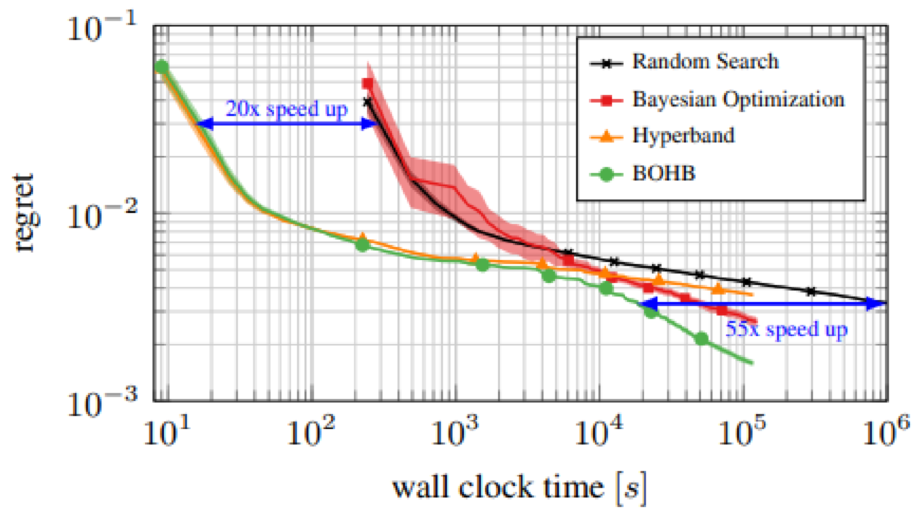
```
sklearn.model_selection.GridSearchCV       -> dask_ml.model_selection.GridSearchCV
sklearn.model_selection.RandomizedSearchCV -> dask_ml.model_selection.RandomizedSearchCV

                                              dask_ml.model_selection.HyperbandSearchCV
```
[4]

BOHB ("Bayesian Optimization and HyperBand") [5]

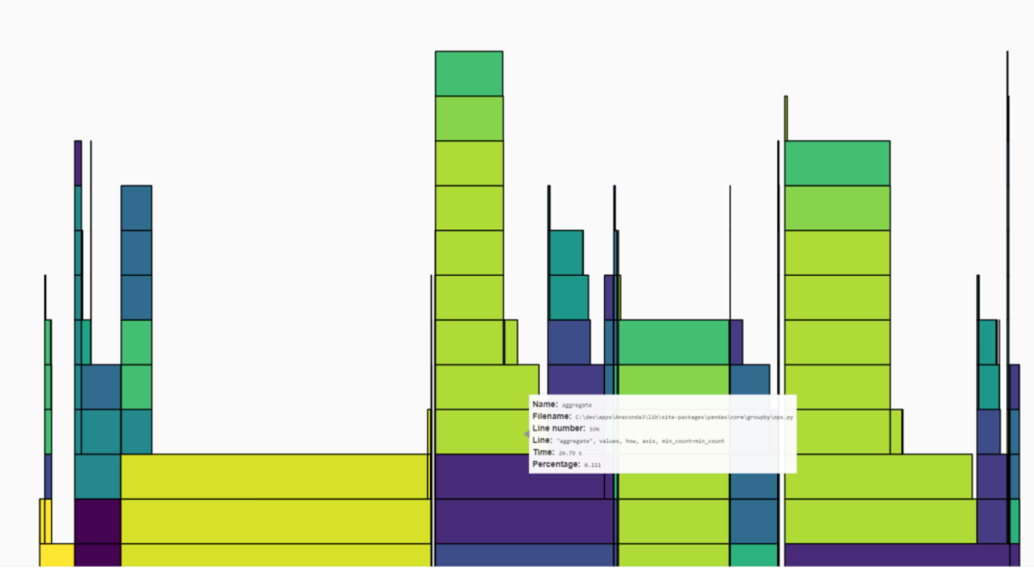
HpBandSter ("Hyperband on steroids") in **automl** [6]

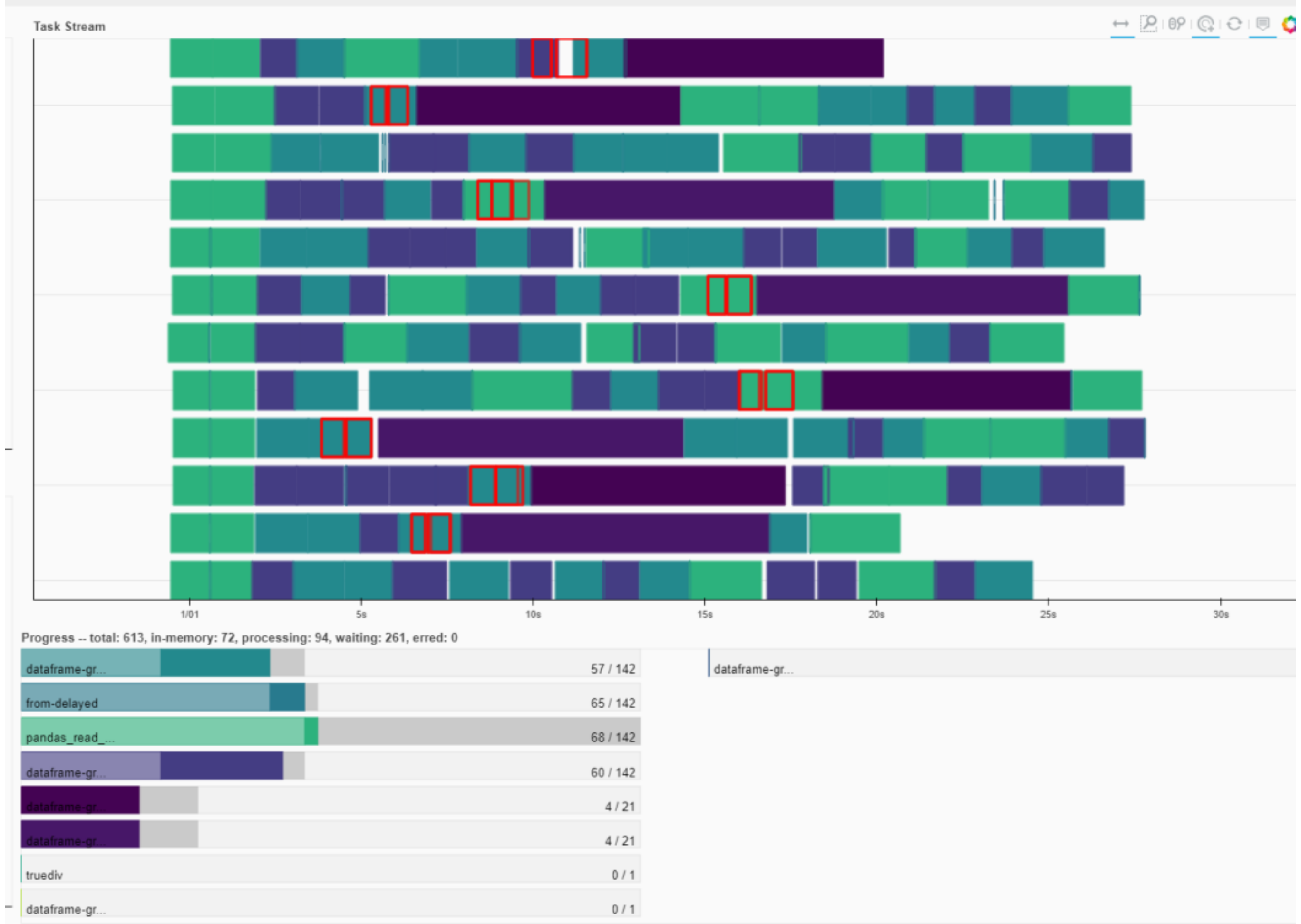
Data > RAM? See `IncrementalSearchCV` in dask_ml



Source: [5]

Monitoring





Task Stream
1/01
5s
10s
15s
20s
25s
30s
Progress -- total: 613, in-memory: 72, processing: 94, waiting: 261, erred: 0
dataframe-gr...
57 / 142
from-delayed
65 / 142
pandas_read_...
68 / 142
dataframe-gr...
60 / 142
4 / 21
4 / 21
truediv
0 / 1
dataframe-gr...
0 / 1
dataframe-gr...

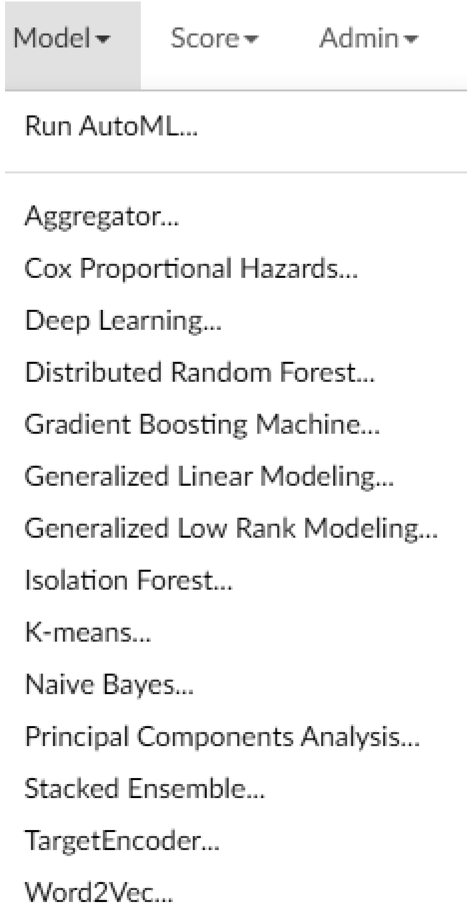
Supports R python Scala Java {JSON} Excel tableau

Multi-threaded (with Java's fork/join)

Memory efficient (compressed columnar storage)

Fast predictions with MOJOs/POJOs

H2O vs H2O4GPU



Multi-Machine Parallelism

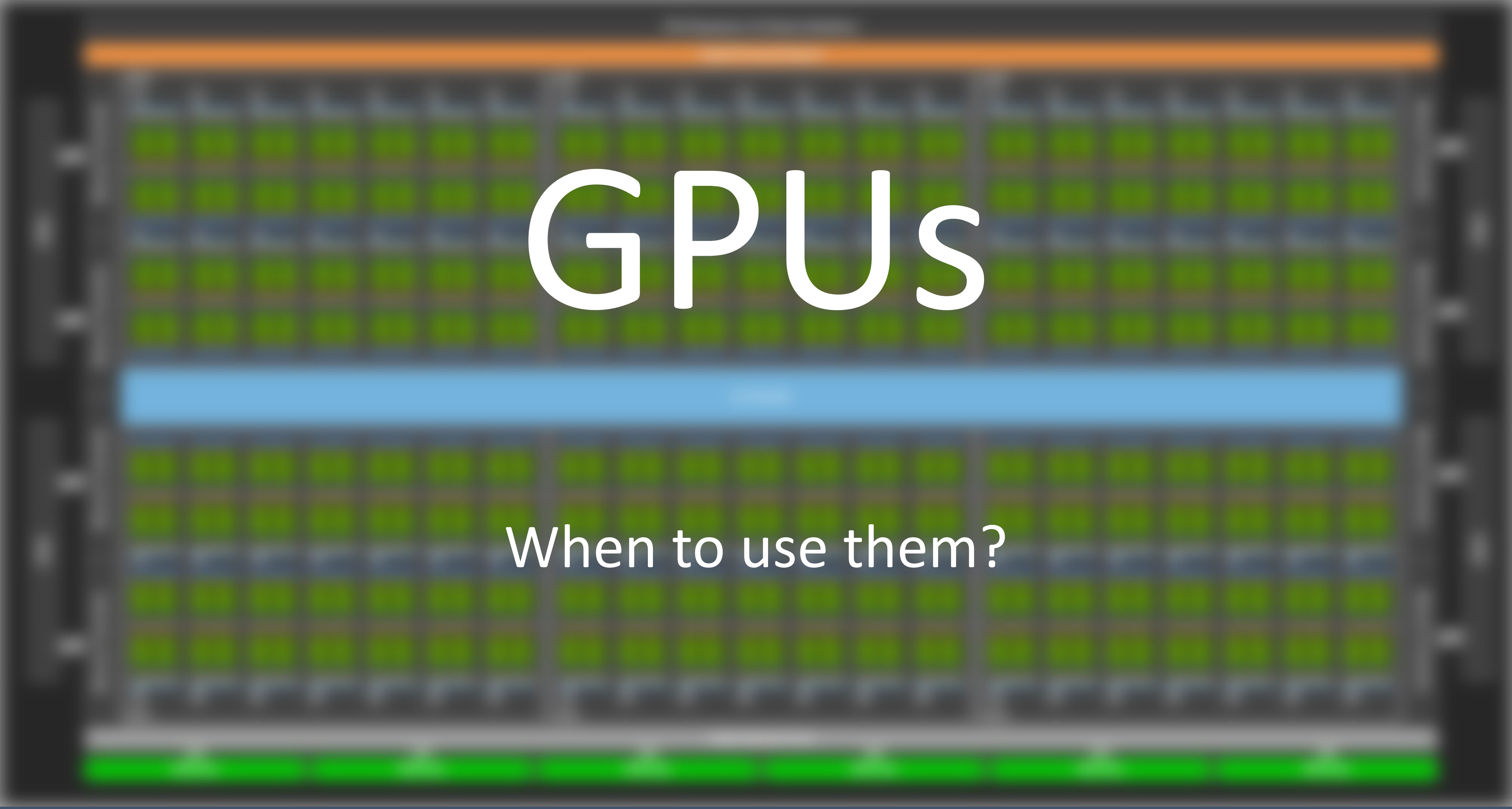
```python
import h2o
from h2o.estimators.gbm import H2OGradientBoostingEstimator

h2o.init() # start or connect to an H2O cluster
data = h2o.import_file(path='/data', pattern='.*\.csv')
training_frame, validation_frame = data.split_frame()

model=H2OGradientBoostingEstimator()
model.train( ['x1', 'x2', 'x3', 'x4'], # feature column names
             'y',                      # target column name
             training_frame,
             validation_frame)
```

Very similar in R

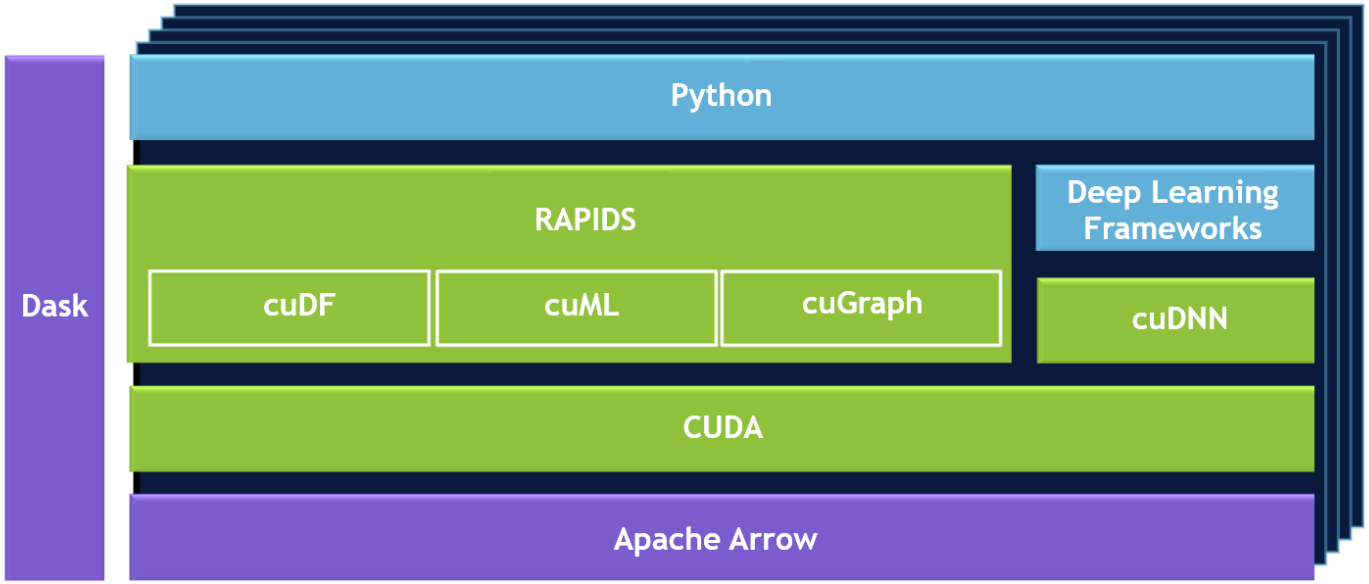
DASK or H_2O.ai ?



GPUs

When to use them?

Everything in the GPUs



"Classic" Stack	RAPIDS Stack
pandas	**cuDF** (1 GPU) Dask + **dask-cuDF** (>1 GPU)
scikit-learn	**cuML**
numpy	**cuPy**
NetworkX	**cuGraph**
	cuXfilter

I/O to GPUs – *wait for GPUDirectStorage*

Data transformation in GPUs ✔

Training & inference in GPUs ✔

https://rapids.ai

Multi-GPU Cluster

RAPIDS

```
from dask_cuda import LocalCUDACluster
cluster = LocalCUDACluster() # Dask CUDA cluster w/ one worker per device

import dask_cudf
df = dask_cudf.read_csv("/path/to/csv") # parallel read

from cuml.dask.neighbors import NearestNeighbors
nn = NearestNeighbors(n_neighbors = 10)
nn.fit(df)
neighbors = nn.kneighbors(df)
```

https://github.com/rapidsai/cuml

Take-aways

Saturate a machine's capability before scaling up

Scale up before scaling out

“Everything in the GPUs” can be faster than a cluster

Reliable distributed systems are not trivial – work with engineering

Monitoring is essential

Balance performance gains with effort & complexity

References

[1] https://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/

[2] https://wesmckinney.com/blog/apache-arrow-pandas-internals/

[3] https://ai.google/research/pubs/pub46180

[4] https://blog.dask.org/2019/09/30/dask-hyperparam-opt

[5] http://proceedings.mlr.press/v80/falkner18a/falkner18a.pdf

[6] https://github.com/automl/HpBandSter

Thank You

Slides @ https://github.com/rpeteanu/talks/TMLS-2019

Me @ https://www.linkedin.com/in/rpeteanu or
razvan.peteanu@gmail.com