Brad Miro - Google
@bradmiro
PyGotham 2019

Distributed Machine Learning with Python

Google Cloud

Agenda

Intro to Distributed Machine Learning

Hardware Considerations

Paradigms

Software Considerations

Closing Thoughts

Intro to Distributed Machine Learning

Distributed Machine Learning

2004 MapReduce

2006 hadoop

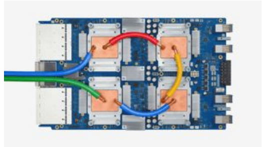
2009 MAHOUT

2012

2014 APACHE Spark

2015

2016



Machine Learning on One Machine

Machine Learning is **hard**

Optimizing linear algebra is computationally expensive

Large models may not fit into memory

Processing training data is time-consuming

What if machine crashes?

Machine Learning on Multiple Machines

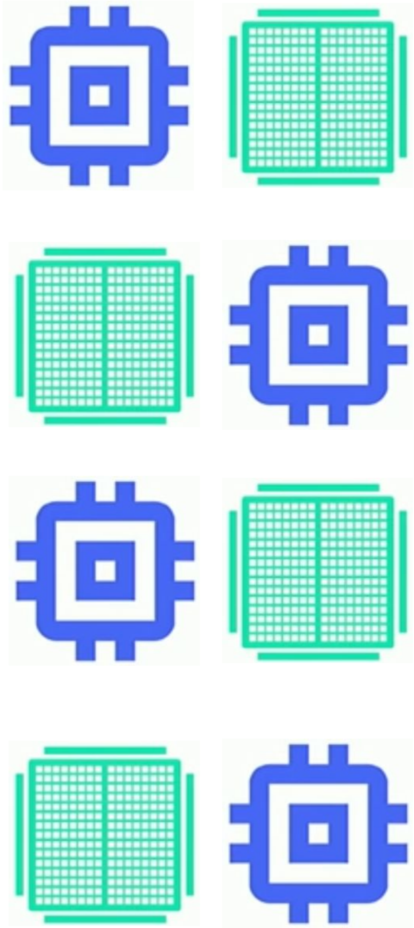
Linear algebra is parallelizable

Large models are split across multiple machines

Processing training data is parallelizable

Introduce fault tolerance

Provide near-linear additional performance per machine



Why?

Larger models are often better for complex problems - audio, images, text

Models can often benefit from large amounts of data

You Shouldn’t ALWAYS Distribute

Parallelization overhead

I/O limitation of older hardware

Smaller models can train faster on one machine

Questions to Ask

Do I REALLY need to distribute my training?

Will I expect to need to distribute my training?

Do I have a lot of data?

Will my data continue to grow?

Will my model continue to grow?

Hardware Considerations

Google Cloud

Central Processing Unit (CPU)

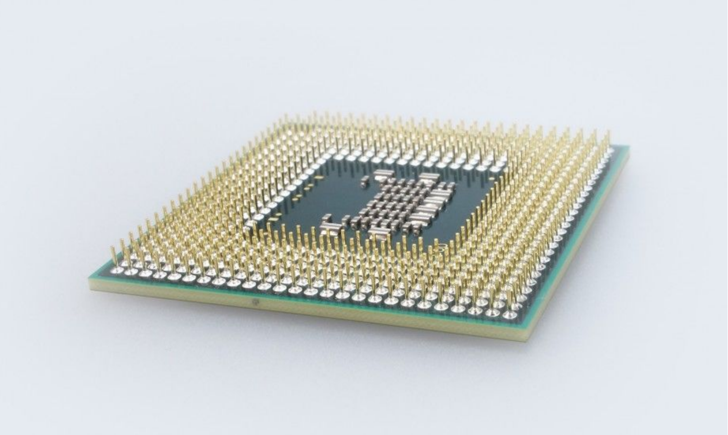
General-purpose programming

Designed for a wide-variety of use-cases

Designed for single-threaded operations

Operations happen synchronously

Can utilize multiple CPUs



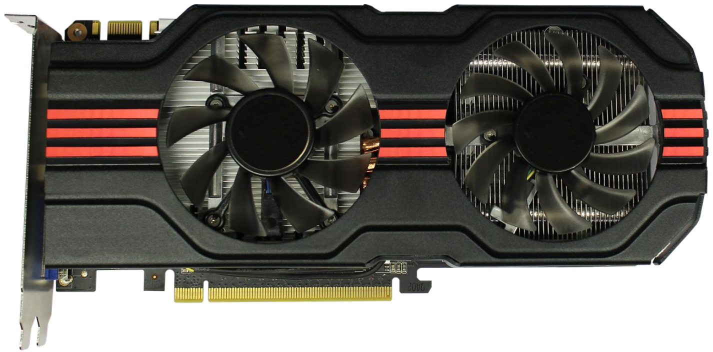
Graphics Processing Unit (GPU)

Designed for parallel processing of information

Render graphics and mathematical computations

Most commonly used for distributed ML

Can utilize multiple GPUs



Tensor Processing Unit (TPU)

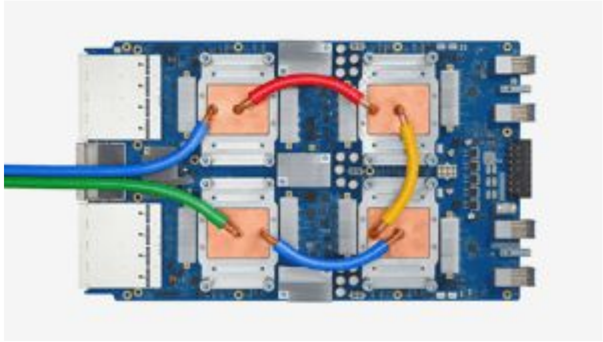
Designed specifically for ML use-cases

Built primarily for use with TensorFlow

Built by Google (who built TensorFlow)

Can access multiple via TPU pods

v1 2016



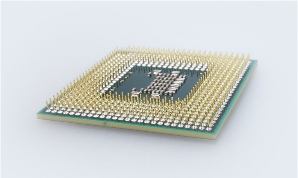
Google Cloud

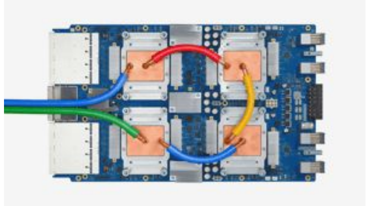
Which Should I Use?

CPU: Locally, parameter server, master

GPU: Mathematical computation

TPU: Optimized specifically for TensorFlow models





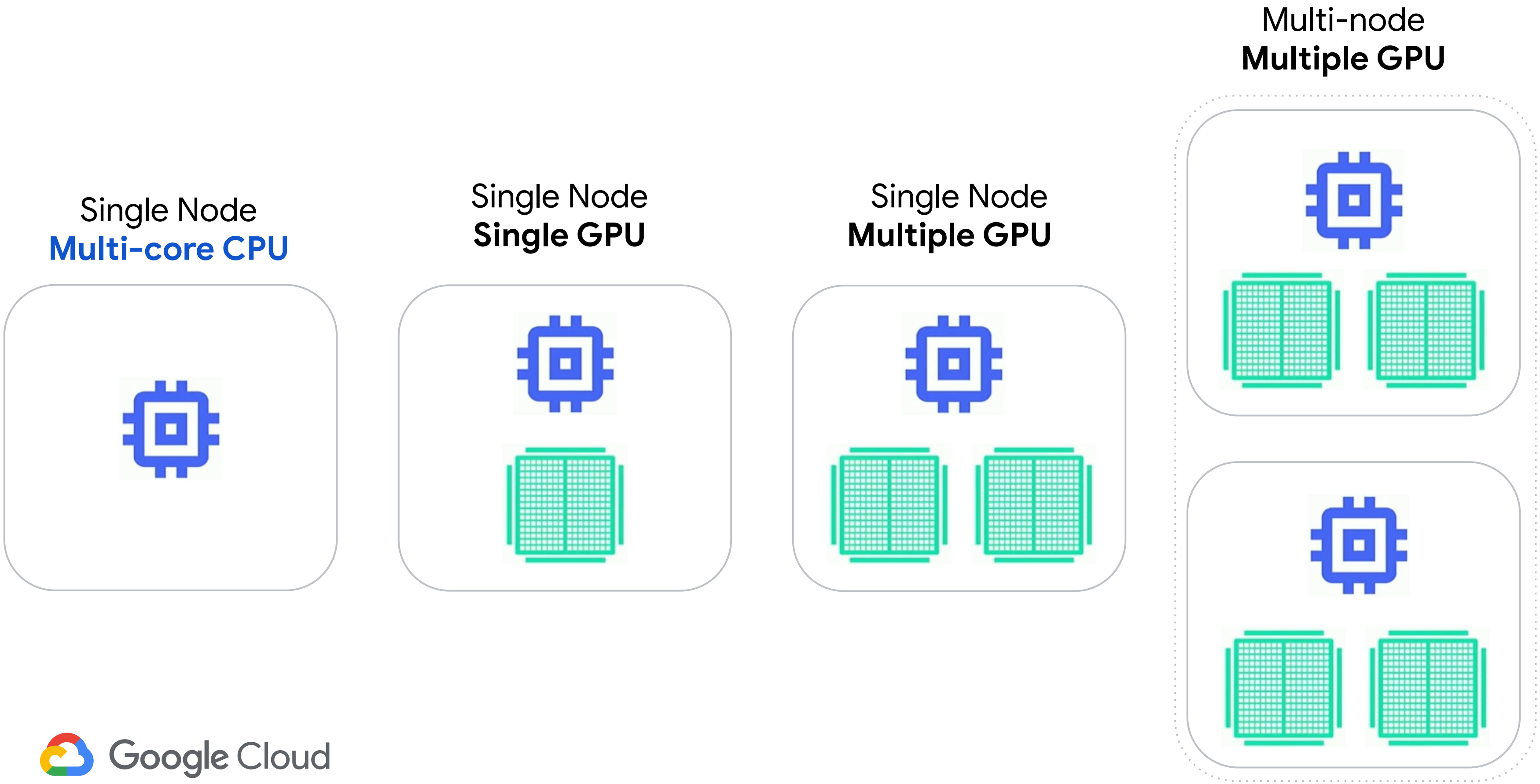
On-Prem vs The Cloud

Certain industries have stricter regulations around cloud-computing

Much easier to add more machines in the cloud vs on-prem

Maintaining physical hardware can be cumbersome

Paradigms



Google Cloud

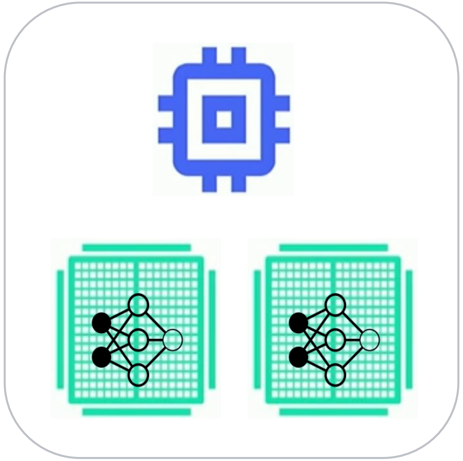
Data Parallelism

Data is partitioned across multiple machines

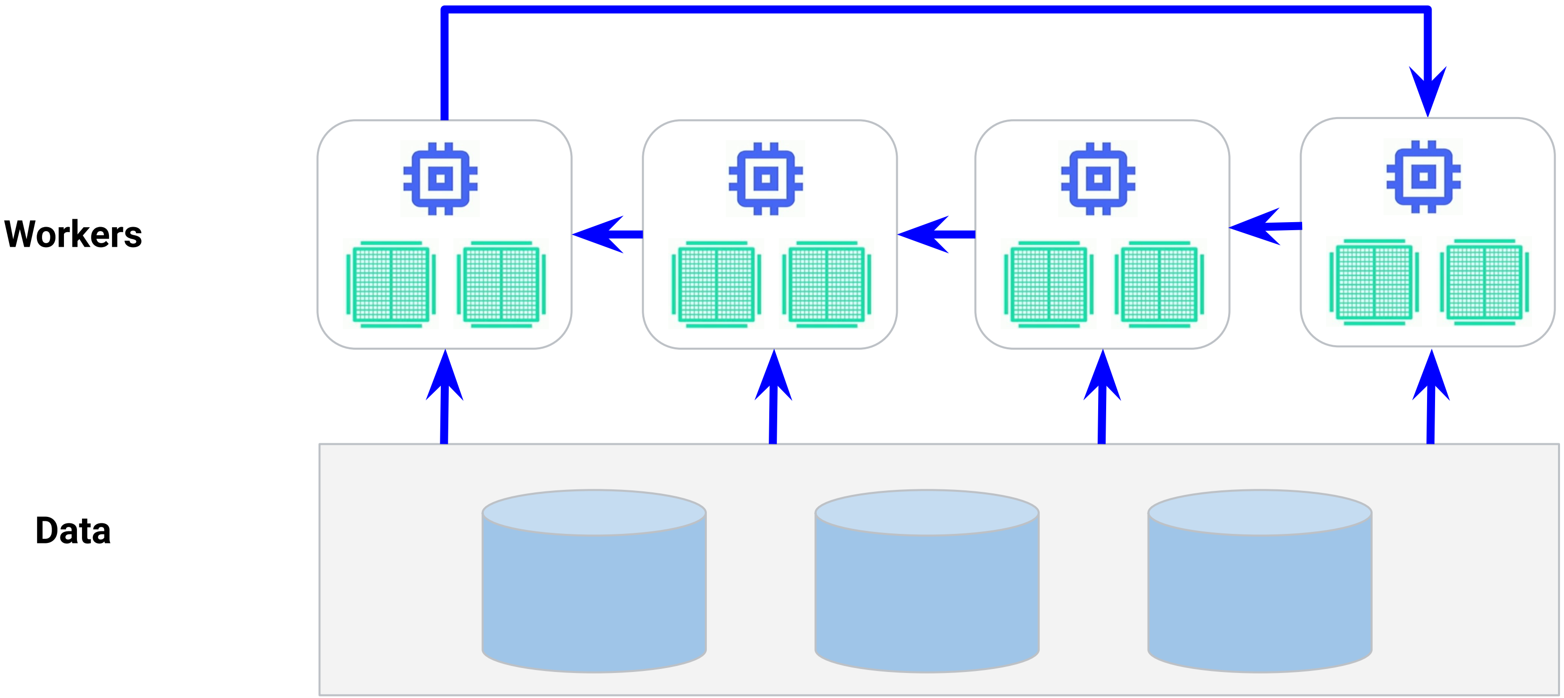
Processed in parallel

Each machine computes gradients / weights

Synchronous vs Asynchronous



SYNCHRONOUS



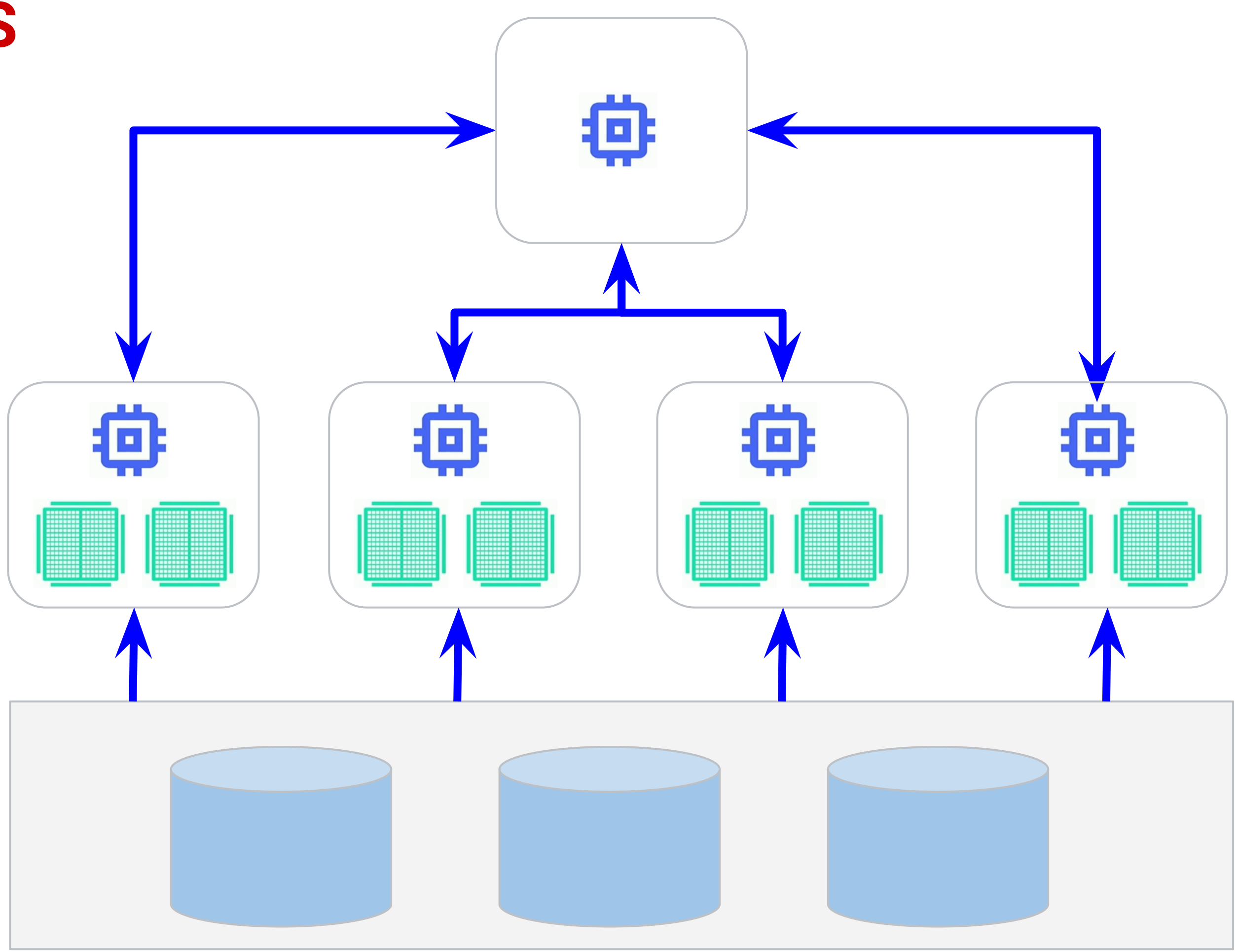
Google Cloud

ASYNCHRONOUS

Parameter Server

Workers

Data



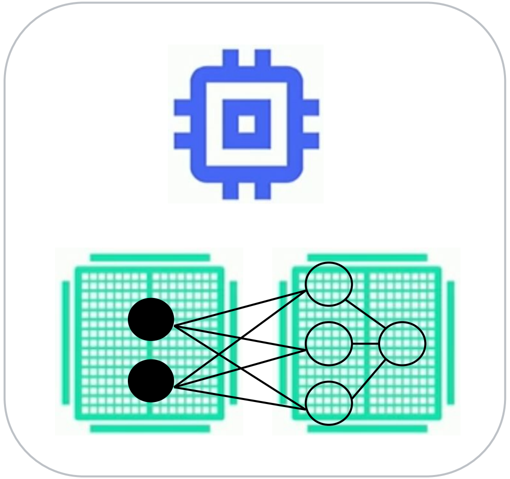
Google Cloud

Model Parallelism

Model is split across multiple machines

Process multiple layers in parallel

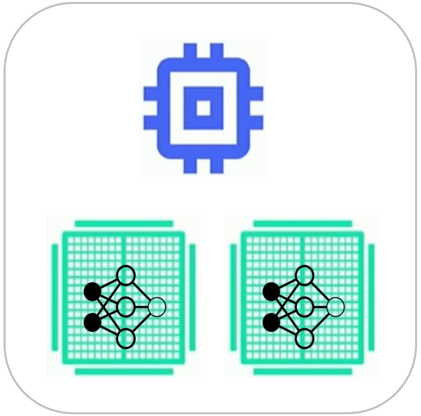
Commonly used for Deep Learning



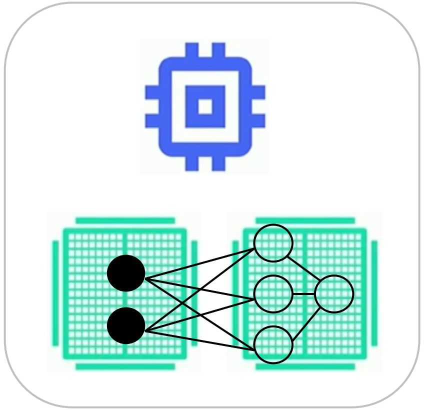
Google Cloud

When to Use Which

Data Parallelism:
model **can** fit onto
a single machine



Model Parallelism:
model **cannot** fit onto
a single machine



Software Considerations

Why Python?

Rich ecosystem for scientific computing

Ease of use

Interacts with engines written in other languages (Java / C++)

APACHE
Spark™

Google Cloud

Apache Spark

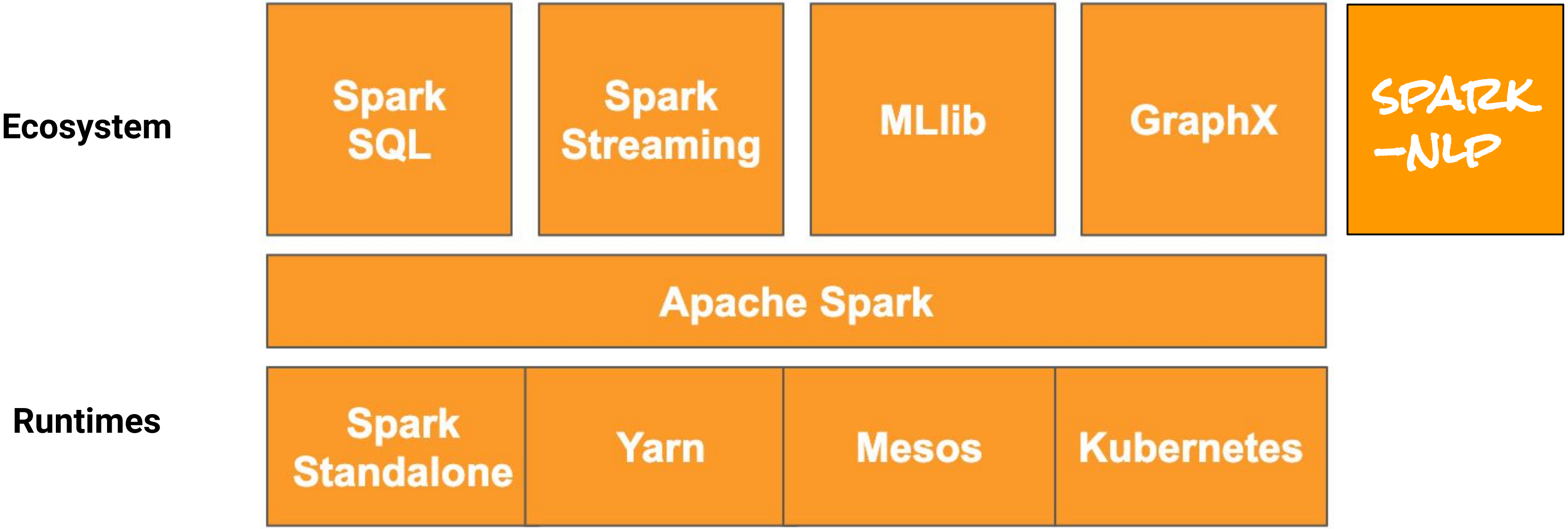
OSS “Unified analytics engine for large-scale data processing”

In-memory distributed data processing

Rich ecosystem - MLlib

Python, Java, Scala, and R

Abstracted parallelization



Google Cloud

```
spark = SparkSession.builder.appName("my_app").getOrCreate()

training = spark.read.format("csv").load("gs://my_bucket/my_data.csv")

lr = LinearRegression(maxIter=10, regParam=0.2)

model = lr.fit(training)

rmse = model.summary.rootMeanSquaredError

print(f"RMSE: {rmse}")
```

Google Cloud

TensorFlow

OSS Machine Learning Framework

Rich ecosystem

Keras as high-level API

Easy to distribute

Python, Javascript, Swift, Java...

TensorFlow

TF Probability

TF Agents

TF Ranking

TF Text

TF Federated

Google Cloud

TensorFlow Distribution Strategies

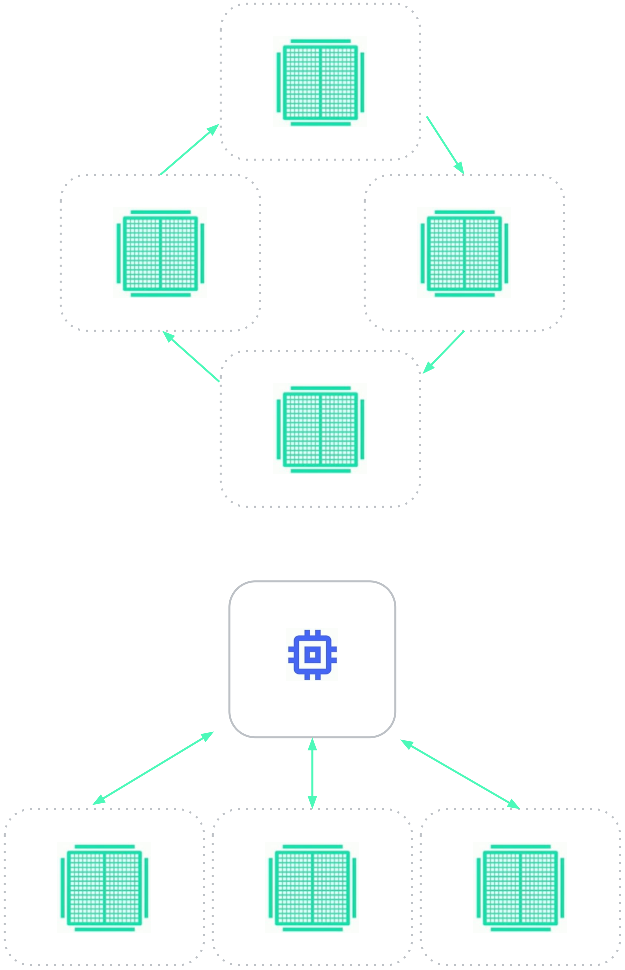
Designed to make ML distribution easy

Useful for researchers, practitioners, etc.

Provide good performance out of the box

Easy to switch between strategies

Little code changes



```python
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(64, input_shape=[10]),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    model.fit(training_data)
```

Closing Thoughts

Takeaways

Data Parallelism distributes training data

Model Parallelism distributes model parameters

CPUs are the brains

GPUs and TPUs do the math

Apache Spark MLlib for Machine Learning

TensorFlow for Deep Learning

In the Cloud

Apache Spark - Cloud Dataproc

TensorFlow - AI Platform

Continued Resources

Large Scale Distributed Deep Networks - Dean et al. (2012)

Deep Learning with COTS HPC - Coates et al. (2013)

Spark: Cluster Computing with Working Sets - Zaharia et al. (2014)

Thank you!

Brad Miro - Google
@bradmiro
PyGotham 2019

Google Cloud