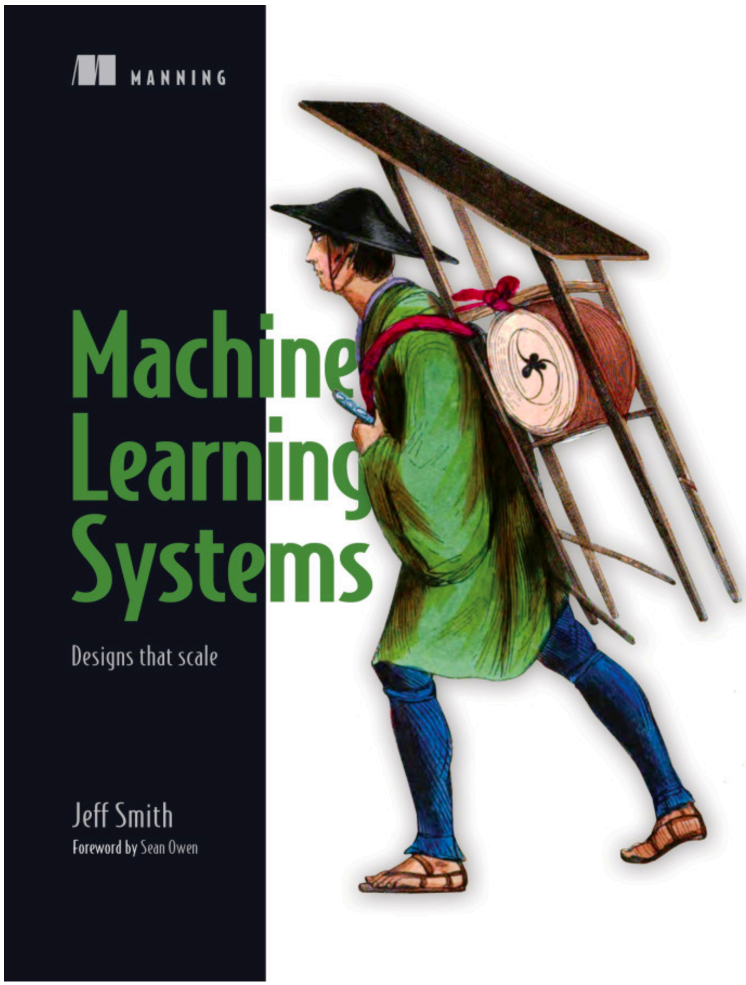
Distributed Machine Learning

Pooyan Jamshidi
USC

Learning goals

- Understand how to build a system that can put the power of machine learning to use.
- Understand how to incorporate ML-based components into a larger system.
- Understand the principles that govern these systems, both as software and as predictive systems.

Main Sources



UBER Engineering

THE NETFLIX TECH BLOG

Spotify Labs

Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow

By Alex Sergeev and Mike Del Balso

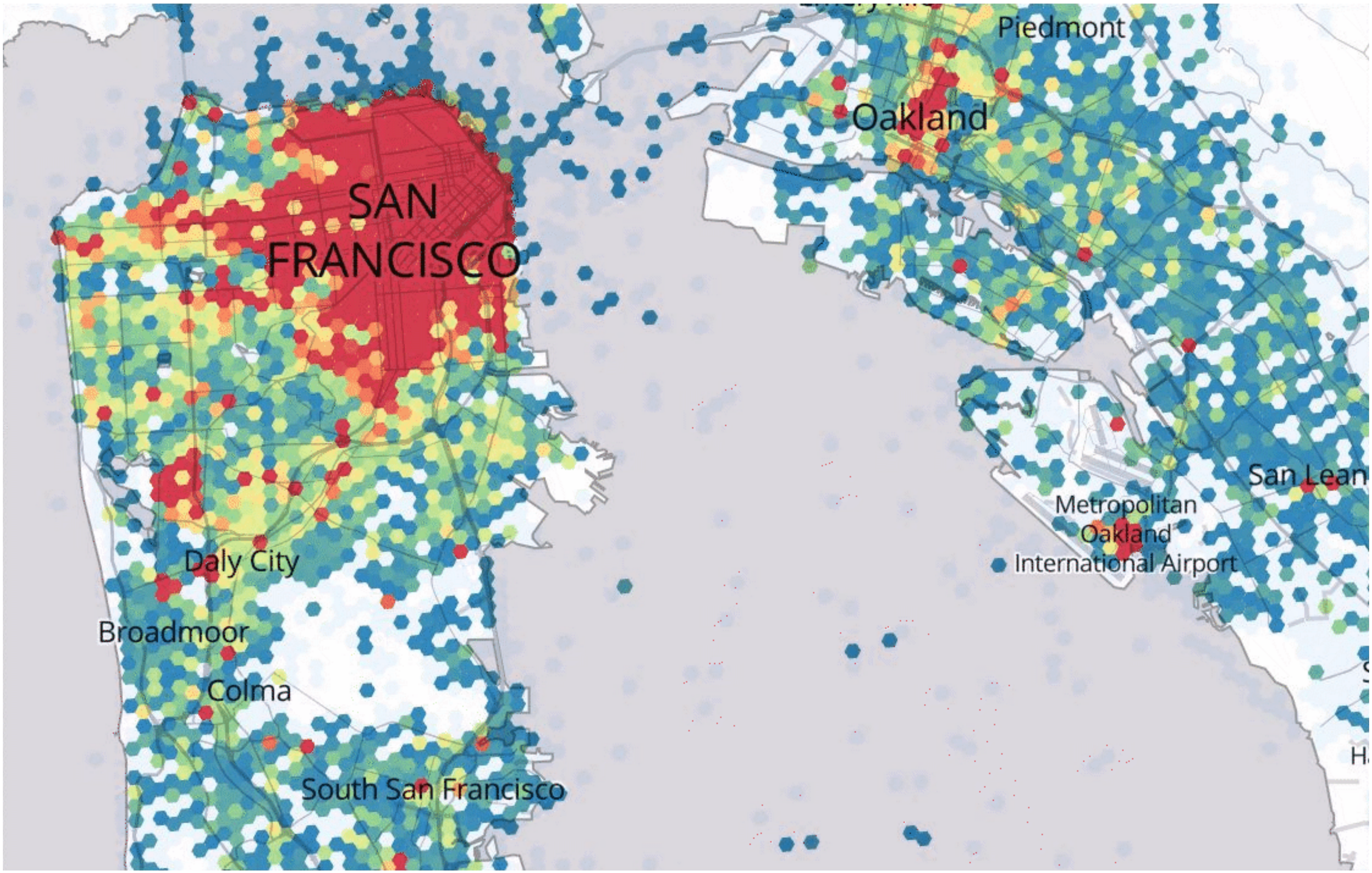
October 17, 2017

Any guess how Uber uses deep learning?

Deep learning across Uber

- Self-driving research
- Trip forecasting
- Fraud prevention

Marketplace forecasting



Let’s begin the story

Uber also uses TensorFlow

Do you know why?

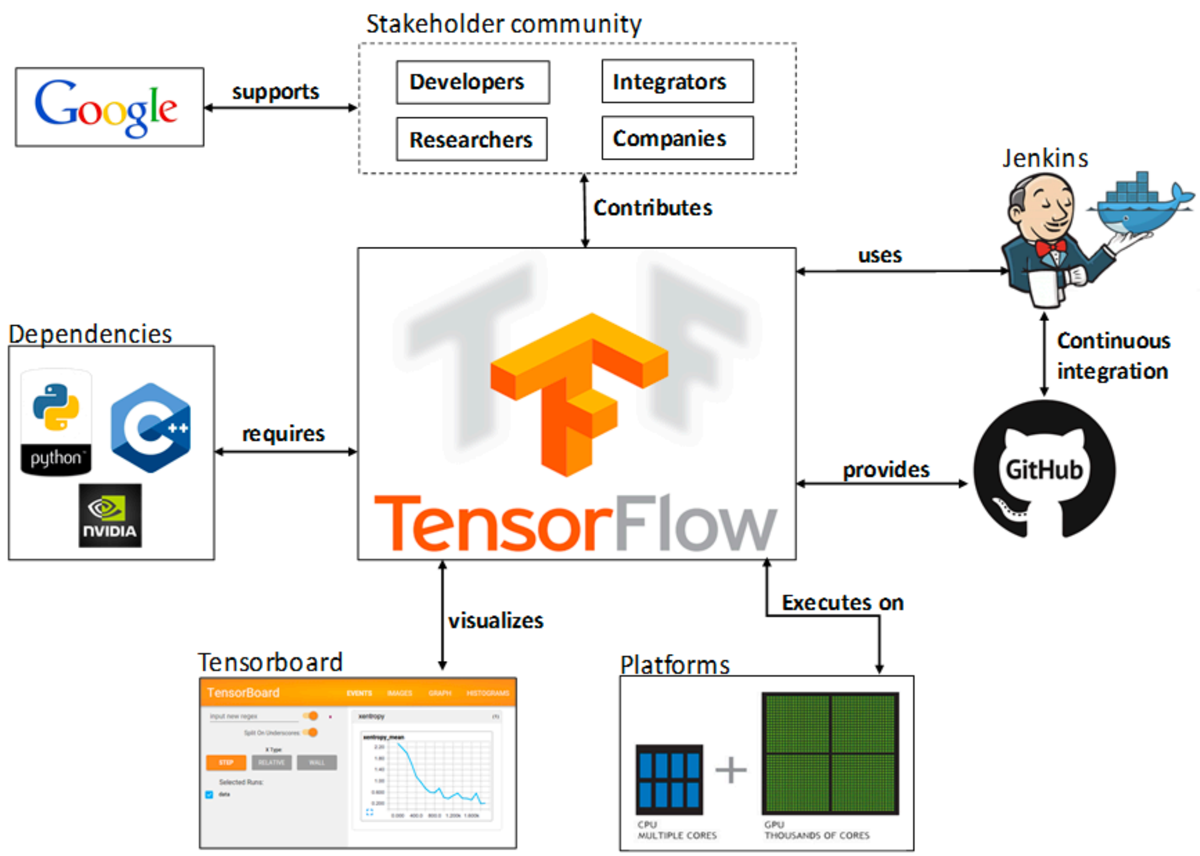
Why Uber adopts TensorFlow?

- TF is one of the most widely used open source frameworks for deep learning, which makes it easy to onboard new users.

Companies using TensorFlow

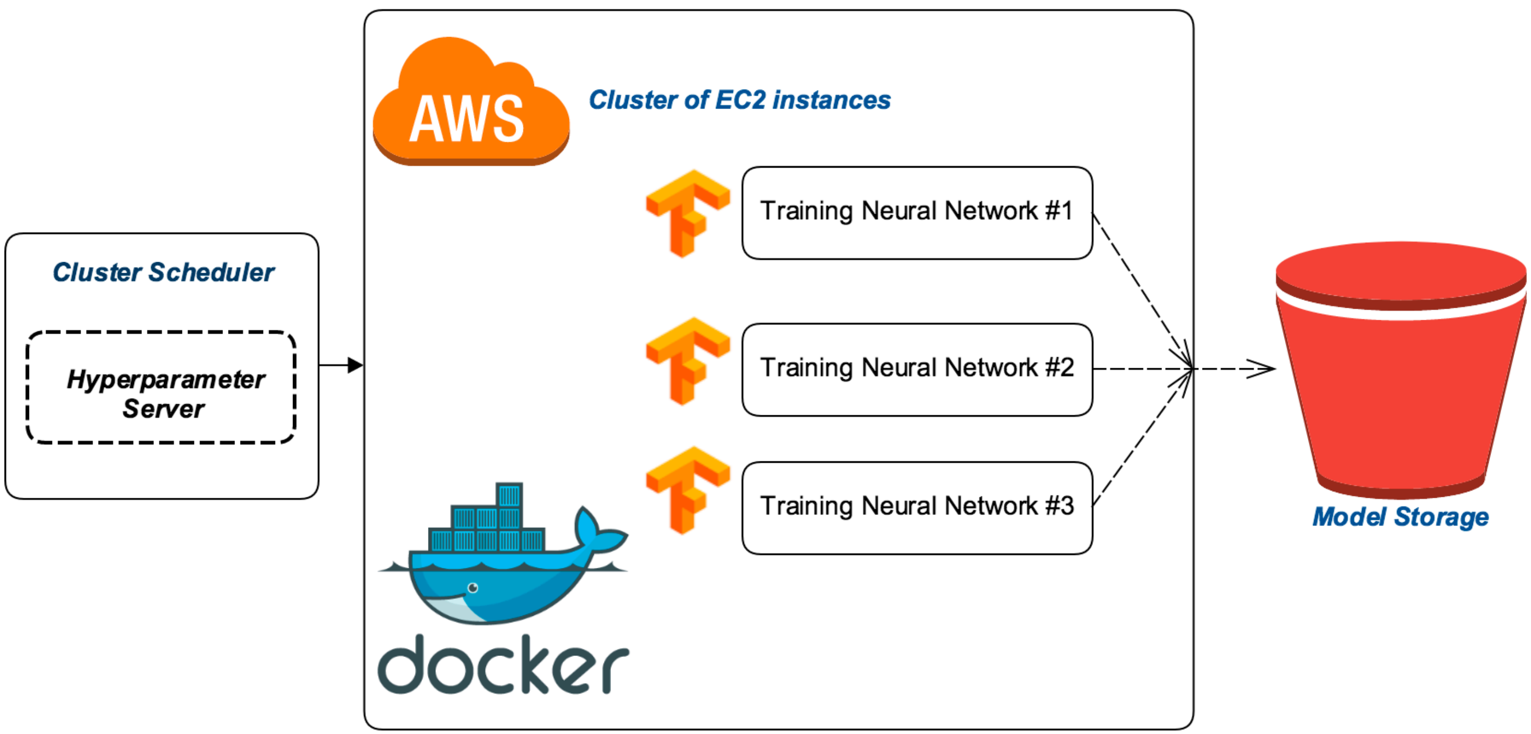
Why Uber adopts TensorFlow?

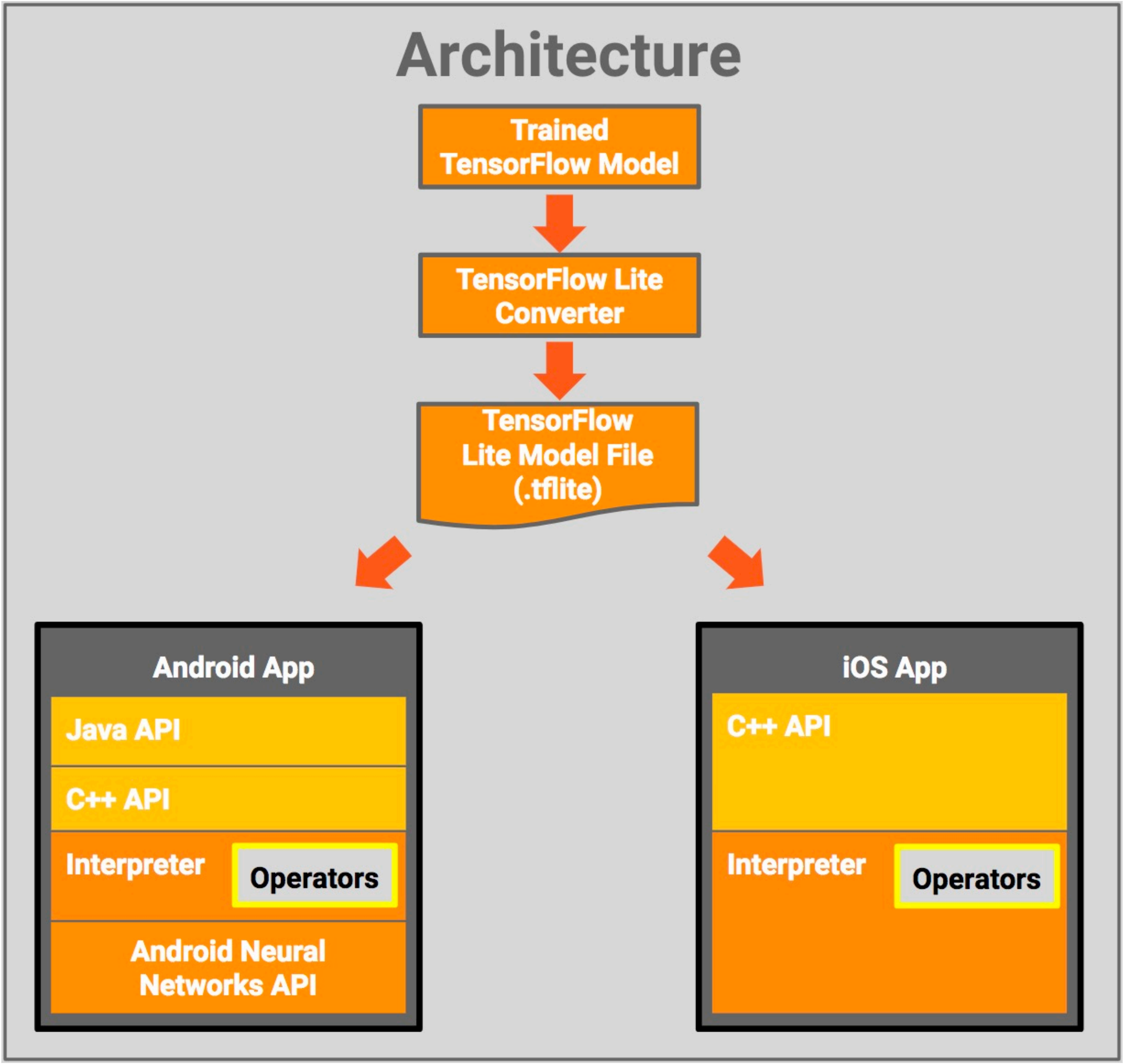
- TF combines high performance with an ability to tinker with low-level model details—for instance, we can use both high-level APIs, such as Keras, and implement our own custom operators using NVIDIA's CUDA toolkit.



Why Uber adopts TensorFlow?

- Additionally, TF has end-to-end support for a wide variety of deep learning use cases, from conducting exploratory research to deploying models in production on cloud servers, mobile apps, and even self-driving vehicles.



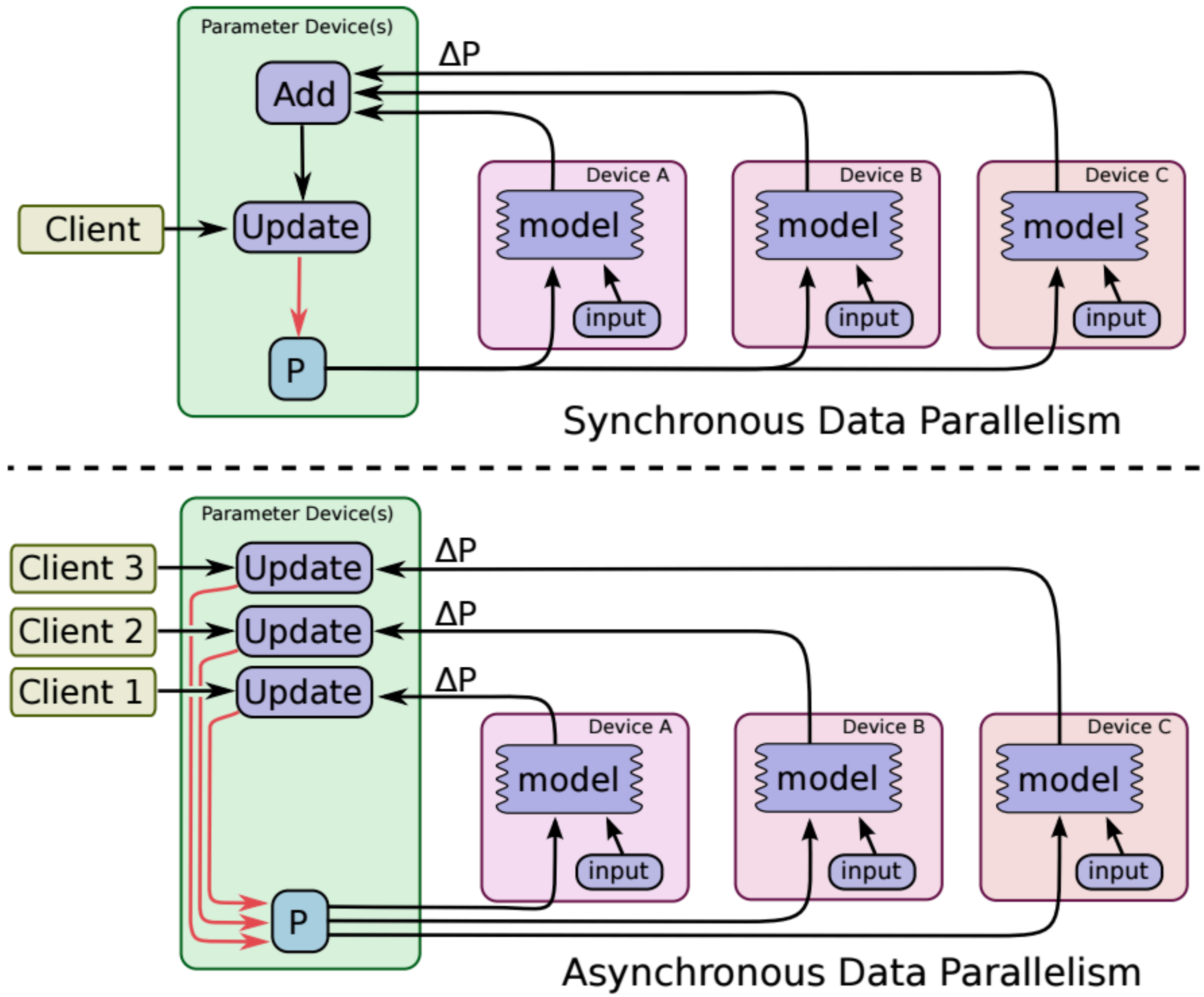

Architecture
Trained TensorFlow Model
TensorFlow Lite Converter
TensorFlow Lite Model File (.tflite)
Android App
Java API
C++ API
Interpreter
Operators
Android Neural Networks API
iOS App
C++ API
Interpreter
Operators

Training time increased a lot!

- Training more and more machine learning models at Uber,
- Their size and data consumption grew significantly.
- In a large portion of cases, the models were still small enough to fit on one or multiple GPUs within a server, but as datasets grew, so did the training times, which sometimes took a week—or longer!—to complete.

Going distributed

Distributed TF



Mapping job names to lists of network addresses

`tf.train.ClusterSpec` construction	Available tasks
`tf.train.ClusterSpec({"local": ["localhost:2222", "localhost:2223"]})`	`/job:local/task:0` `/job:local/task:1`

`tf.train.ClusterSpec({`
`    "worker": [`
`        "worker0.example.com:2222",`
`        "worker1.example.com:2222",`
`        "worker2.example.com:2222"`
`    ],`
`    "ps": [`
`        "ps0.example.com:2222",`
`        "ps1.example.com:2222"`
`    ]})`
	`/job:worker/task:0`
	`/job:worker/task:1`
	`/job:worker/task:2`
	`/job:ps/task:0`
	`/job:ps/task:1`

Specifying distributed devices in your model

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.Variable(...)
  biases_1 = tf.Variable(...)

with tf.device("/job:ps/task:1"):
  weights_2 = tf.Variable(...)
  biases_2 = tf.Variable(...)

with tf.device("/job:worker/task:7"):
  input, labels = ...
  layer_1 = tf.nn.relu(tf.matmul(input, weights_1) + biases_1)
  logits = tf.nn.relu(tf.matmul(layer_1, weights_2) + biases_2)
  # ...
  train_op = ...

with tf.Session("grpc://worker7.example.com:2222") as sess:
  for _ in range(10000):
    sess.run(train_op)
```

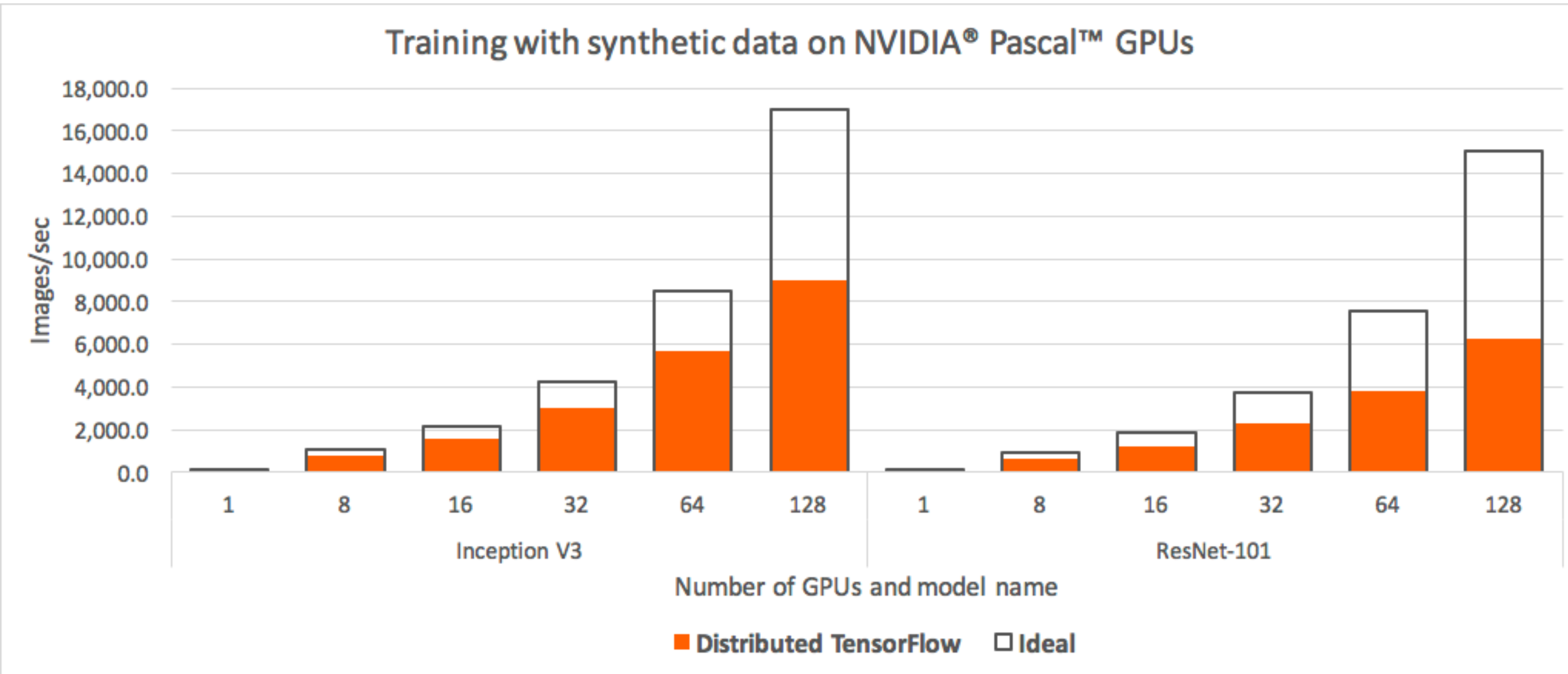
Uber first experience with Distributed TF

- It was not always clear which code modifications needed to be made to distribute their model training code.

- The standard distributed TensorFlow package introduces **many new concepts**: workers, parameter servers, `tf.Server()`

- The challenge of computing at **Uber's scale**. After running a few benchmarks, we found that we could not get the standard distributed TensorFlow to scale as well as our services required.

Distributed TF became inefficient at Uber scale



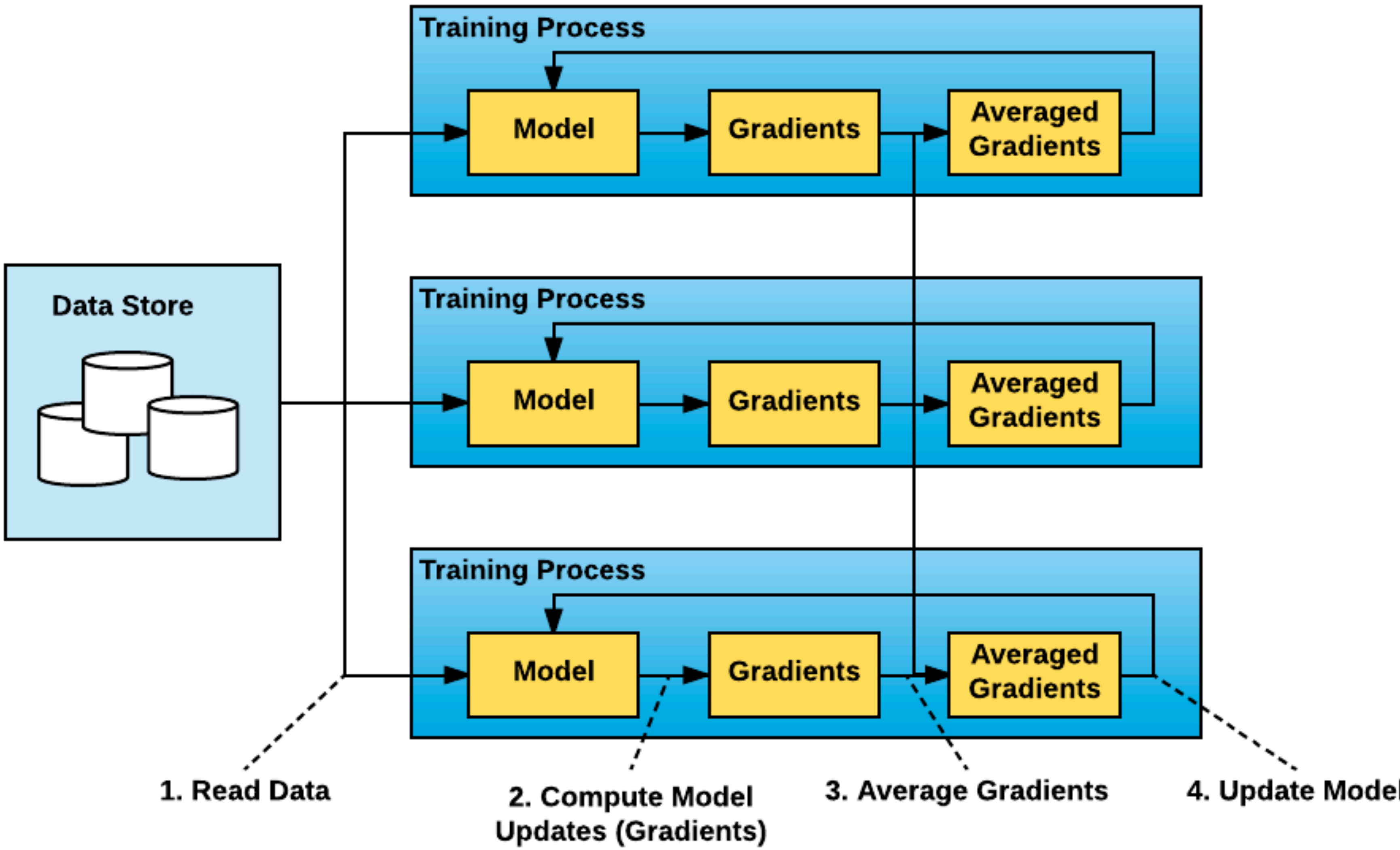
Models were unable to leverage half of the resource

They become even more motivated after observing Google training ResNet-50 in an hour!

- “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour,” demonstrating their training of a ResNet-50 network in one hour on 256 GPUs by combining principles of data parallelism with an **innovative learning rate adjustment technique**.

Leveraging a different type of algorithm

Data parallelism (Facebook)



But wait!

- What other approaches exist for distributing a (deep Learning) algorithm?
- And why Uber could possibly do Data Parallel approach? Any insight?

And here is why Uber started with Data Parallel

- Uber's models were small enough to fit on a single GPU, or multiple GPUs in a single server

How data parallel works?

1. Run multiple copies of the training script and each copy:
   A. reads a chunk of the data
   B. runs it through the model
   C. computes model updates (gradients)
2. Average gradients among those multiple copies
3. Update the model Repeat (from Step 1a)

Data parallel vs Model parallel

- **Data Parallel** ("Between-Graph Replication")
  - Send exact same model to each device
  - Each device operates on its partition of data § ie. Spark sends same function to many workers
  - Each worker operates on their partition of data
- **Model Parallel** ("In-Graph Replication")
  - Send different partition of model to each device
  - Each device operates on all data

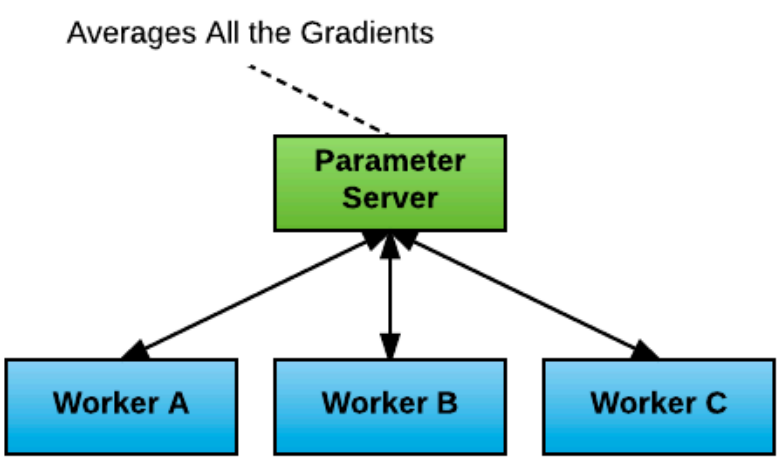
While this approach improved performance, they encountered two challenges

It was good, but they hit some challenges

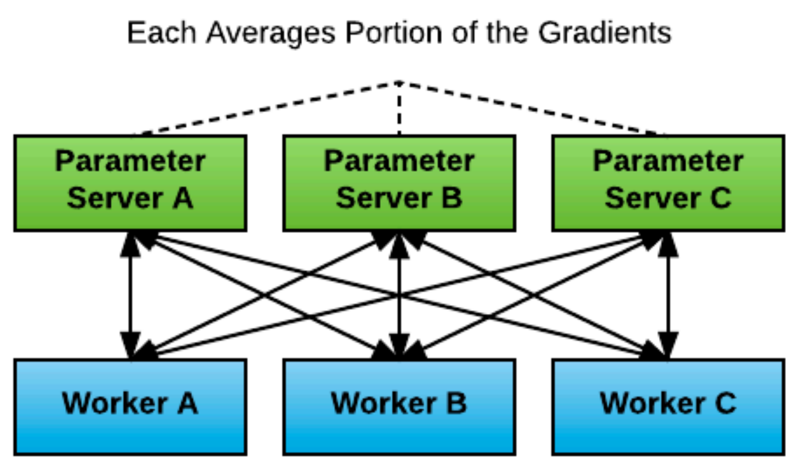
- Identifying the right ratio of worker to parameter servers.
  - 1 parameter server
  - Multiple parameter server
- Handling increased TensorFlow program complexity
  - Every user of distributed TensorFlow had to explicitly start each worker and PS, pass around service discovery information.
  - Users had to ensure that all the operations were placed appropriately and code is modified to leverage multiple GPUs.

It was good, but they hit some challenges

- Identifying the right ratio of worker to parameter servers.
    - 1 parameter server
    - Multiple parameter server



or

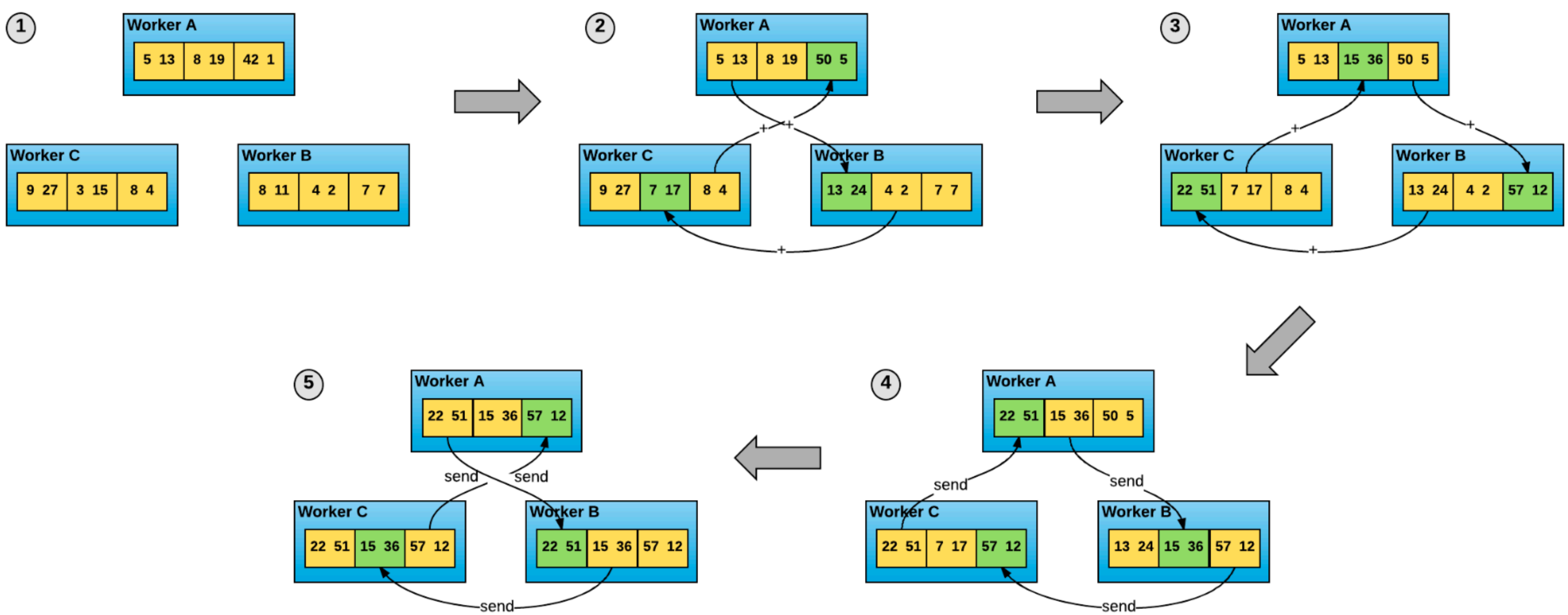


TF Complexity

- Handling increased TensorFlow program complexity
  - Every user of distributed TensorFlow had to explicitly start each worker and PS, pass around service discovery information.
  - Users had to ensure that all the operations were placed appropriately and code is modified to leverage multiple GPUs.

```
# On ps0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=0
# On ps1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=1
```

Baidu approach to avoid parameter server

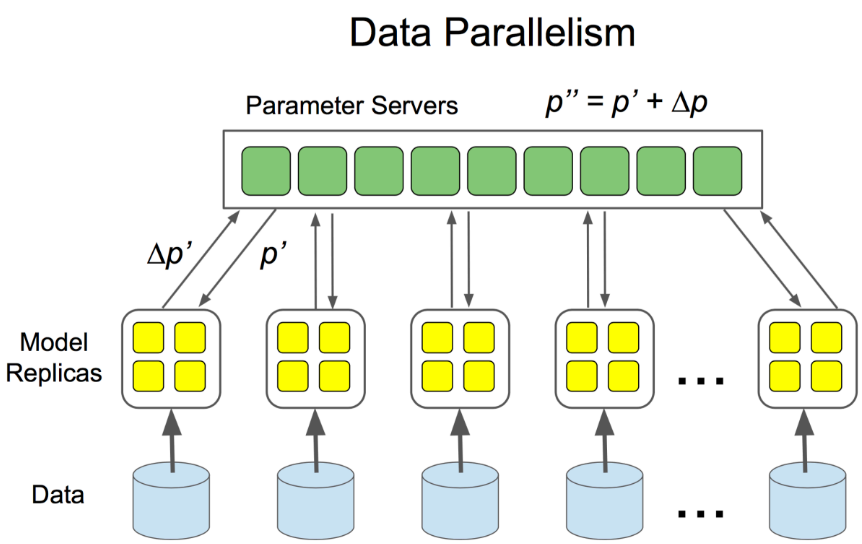


Baidu all reduce is not only network optimal, but easier to adopt

- Users utilize a Message Passing Interface (MPI) implementation such as OpenMPI to launch all copies of the TensorFlow program.
- MPI then transparently sets up the distributed infrastructure necessary for workers to communicate with each other.
- All the user needs to do is modify their program to average gradients using an allreduce() operation.

Horovod = Distributed deep learning with TensorFlow

Any similarity?



HOROVOD

Hovord was then built upon Baidu allreduce approach

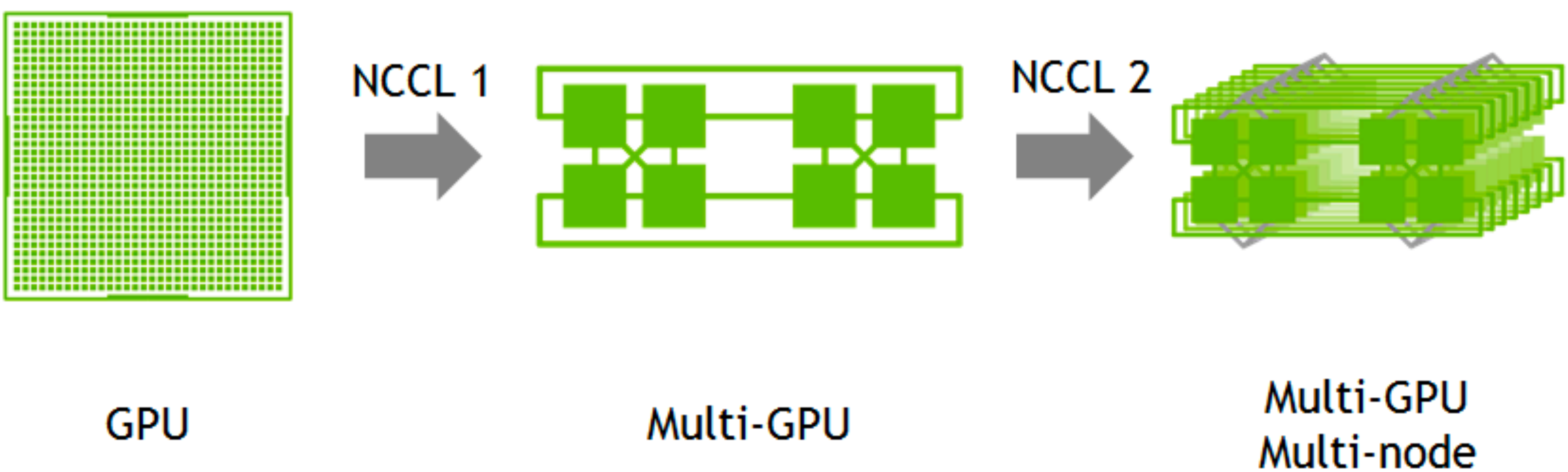
- A stand-alone Python package called Horovod.
- Distributed TensorFlow processes use Horovod to communicate with each other.
- At any point in time**, various teams at Uber may be using different releases of TensorFlow**. We wanted all teams to be able to leverage the ring-allreduce algorithm without needing to upgrade to the latest version of TensorFlow, apply patches to their versions, or even spend time building out the framework.
- Having a stand-alone package allowed Uber to cut the time required to **install Horovod from about an hour to a few minutes**, depending on the hardware.

From single GPU to Multi-GPU Multi-Node

- Replaced Baidu ring-allreduce with NCCL.

- NCCL is NVIDIA's library for **collective communication** that provides a highly optimized version of ring-allreduce.

- NCCL 2 introduced the ability to run ring-allreduce across **multiple machines**.



The update were included API improvements

- Several API improvements inspired by **feedback Uber** received from a number of initial users.
- A **broadcast operation** that enforces consistent initialization of the model on all workers.
- The new API allowed Uber to cut down the number of operations a user had to introduce to their single GPU program to four.

Distributing training job with Horovod

```
import tensorflow as tf
import horovod.tensorflow as hvd

# Initialize Horovod
hvd.init()

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list = str(hvd.local_rank())

# Build model…
loss = …
opt = tf.train.AdagradOptimizer(0.01)

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)
```

Distributing training job with Horovod

```
# Add hook to broadcast variables from rank 0 to all other processes during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Make training operation
train_op = opt.minimize(loss)

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when done
# or an error occurs.
with
tf.train.MonitoredTrainingSession(checkpoint_dir="/tmp/train_logs",
                                  config=config,
                                  hooks=hooks) as mon_sess:
 while not mon_sess.should_stop():
  # Perform synchronous training.
   mon_sess.run(train_op)
```

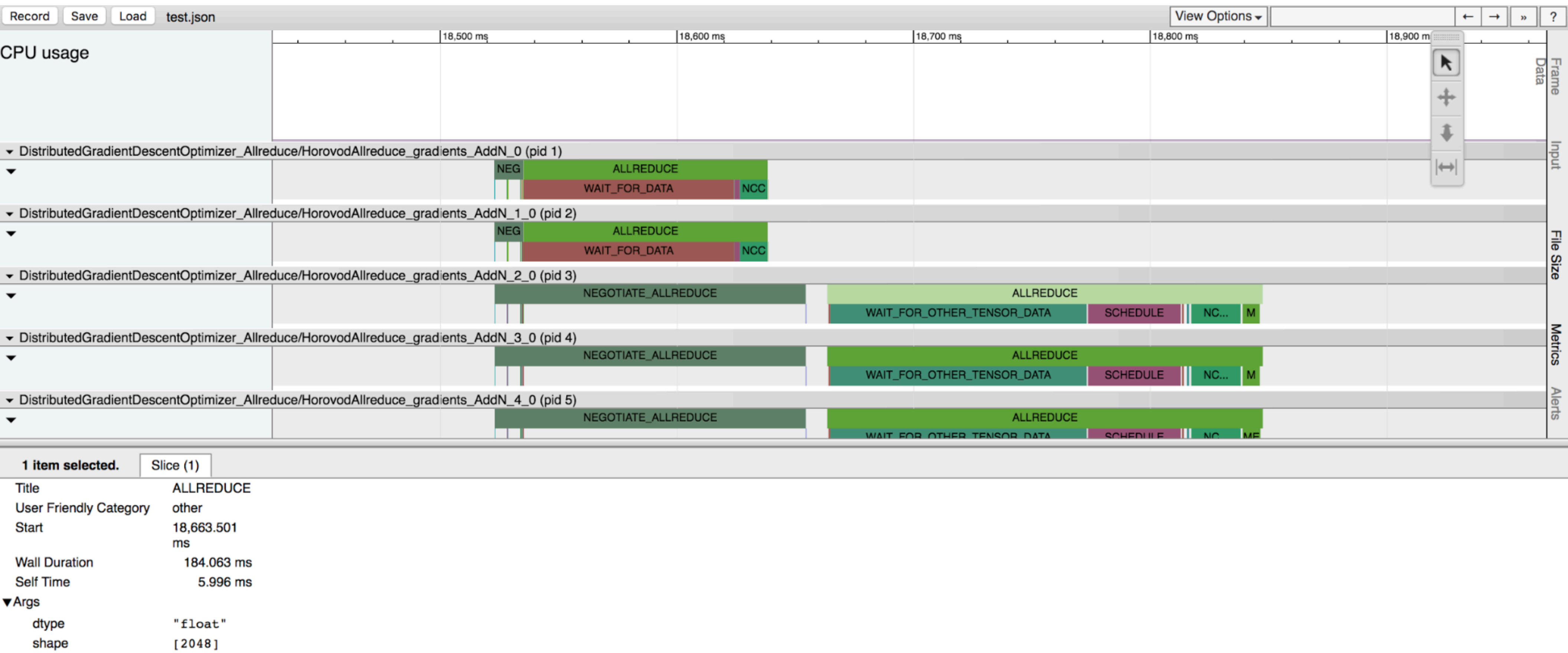
User can then run several copies of the program across multiple servers

```
$ mpirun -np 16 -x LD_LIBRARY_PATH -H
server1:4,server2:4,server3:4,server4:4 python train.py
```

Horovord also distribute Keras programs

- Horovod can also distribute Keras programs by following the same steps.

Now time come to debugging a distributed systems



Yet another challenge: Tiny allreduce

- After we analyzed the timelines of a few models, we noticed that those with a large amount of tensors, such as ResNet-101, tended to have many tiny allreduce operations.

- ring-allreduce utilizes the network in an optimal way if the tensors are large enough, but does not work as efficiently or quickly if they are very small.

Tensor Fusion

- What if multiple tiny tensors could be fused together before performing ring-allreduce on them?

Tensor Fusion

1. Determine which tensors are ready to be reduced. Select the first few tensors that fit in the buffer and have the same data type.

2. Allocate a fusion buffer if it was not previously allocated. Default fusion buffer size is 64 MB.

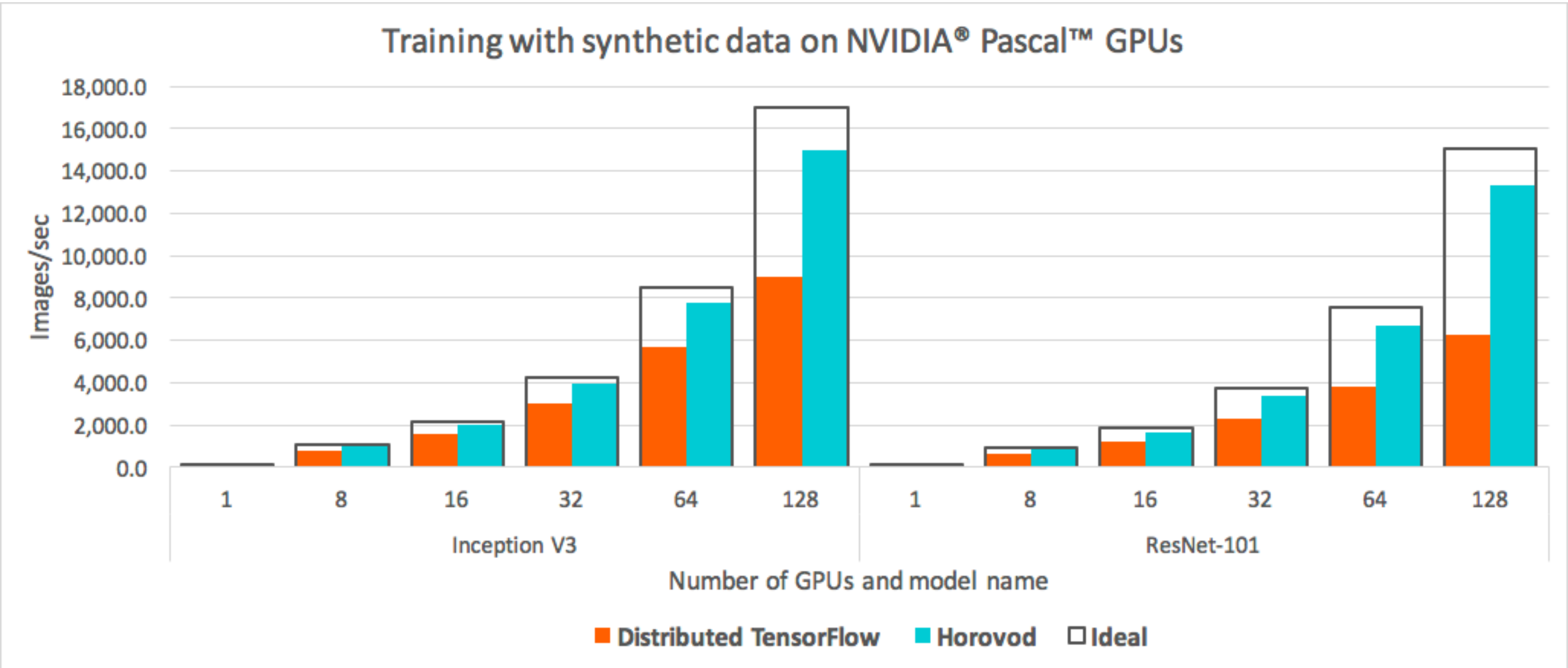
3. Copy data of selected tensors into the fusion buffer.

4. Execute the allreduce operation on the fusion buffer.

5. Copy data from the fusion buffer into the output tensors.

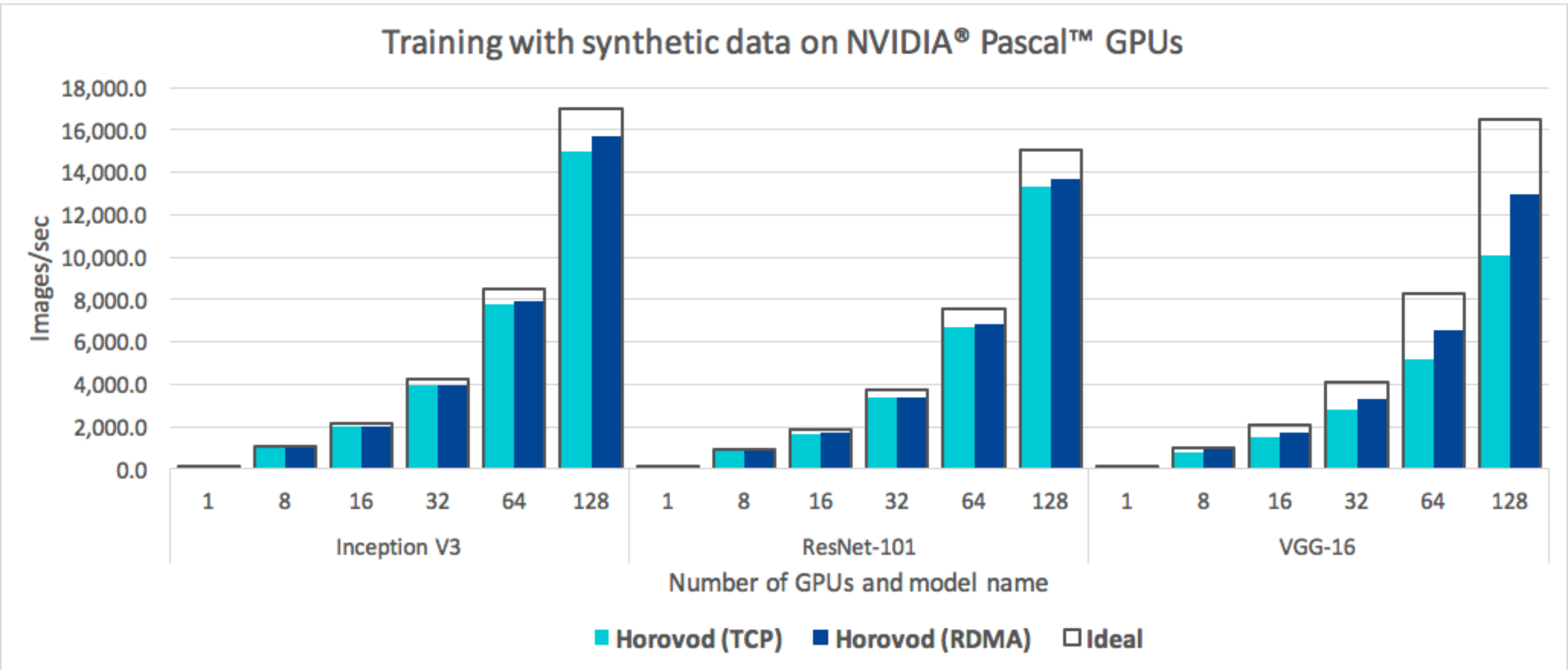
6. Repeat until there are no more tensors to reduce in the cycle.

Horovod vs TF



the training was about twice as fast as standard distributed TensorFlow.

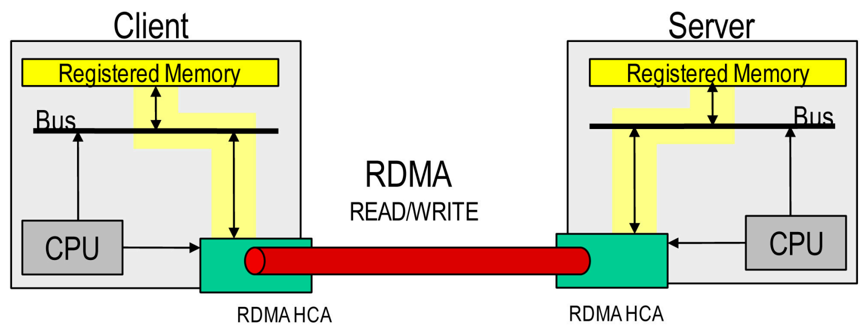
Benchmarking with RDMA network cards



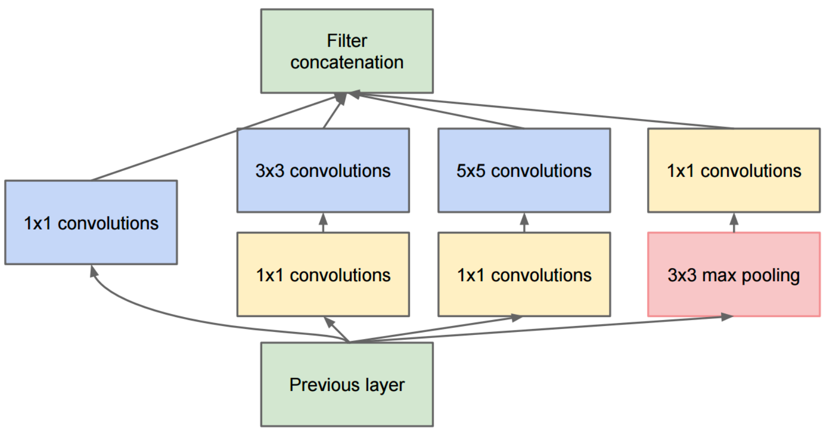
VGG-16 model experienced a significant 30 percent speedup when we leveraged RDMA networking.

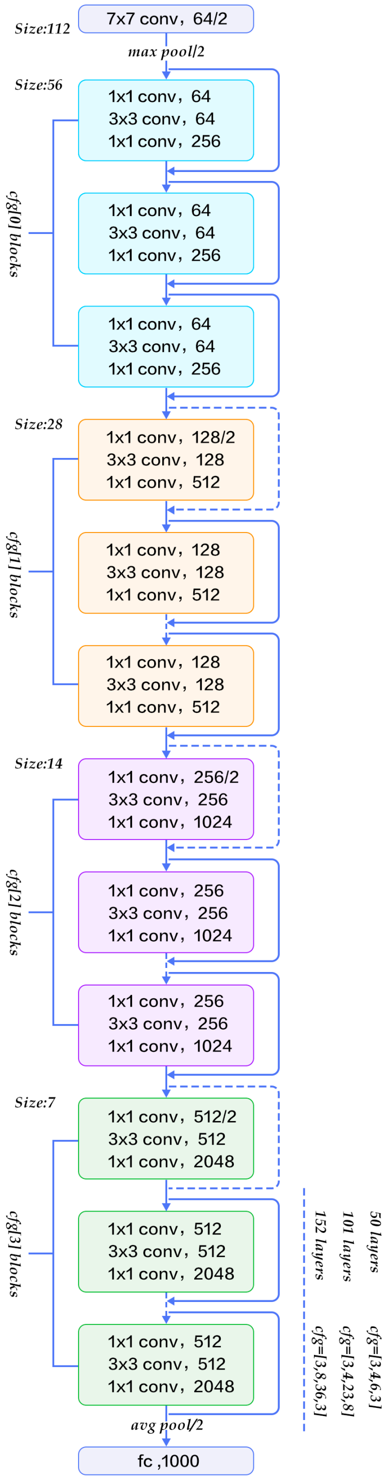
Do you know why that happened?

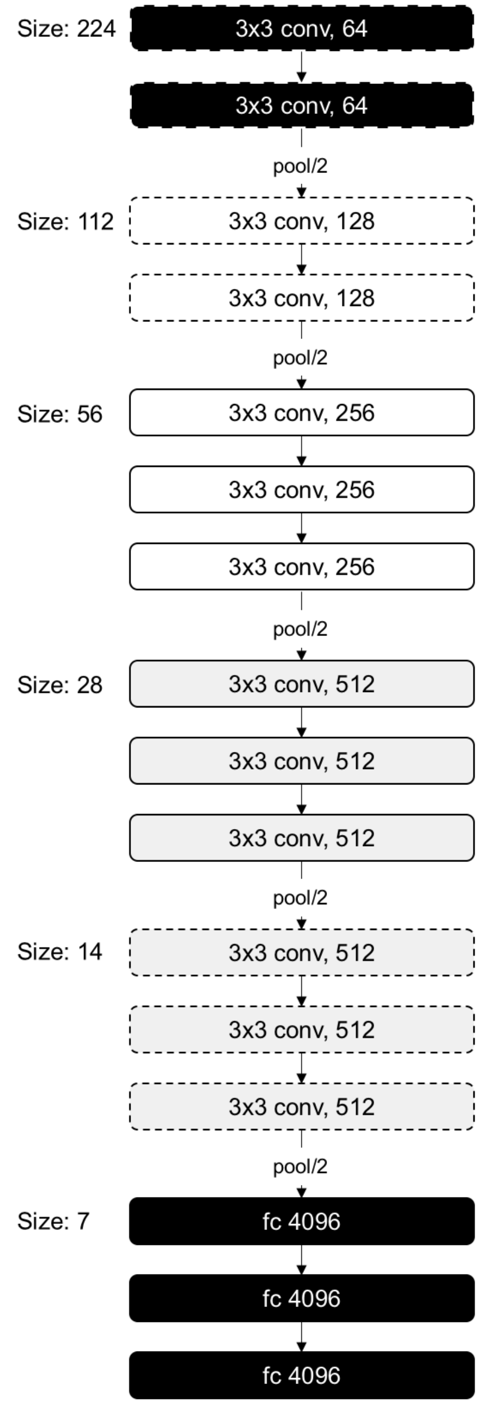
Any insight?



Any thought?







Parameters: 25 million

Parameters: 25 million

Parameters: 138 million

Summary

- We reviewed when ML needs to go distributed.
- We studies some alternative solutions and why Uber decided to built up their own solution
- We studied extensions that was made by Uber to accommodate their own requirements.
- We reviewed how Horovod helped Uber to scale up their training process.