



# Pretrain foundation models on AWS

*Generative AI Foundations on AWS*

Emily Webber, Principal ML Specialist SA at AWS

Lesson 4 – **Level 400**

# Today's activities



- When to pretrain a new foundation model from scratch
- What you need to do this effectively
- How to do this on AWS
- Distributed training fundamentals
- Hands-on walk through: pretraining a 30B parameter LLM on SageMaker

Reminder – everything we discuss today  
is possible on AWS and SageMaker!

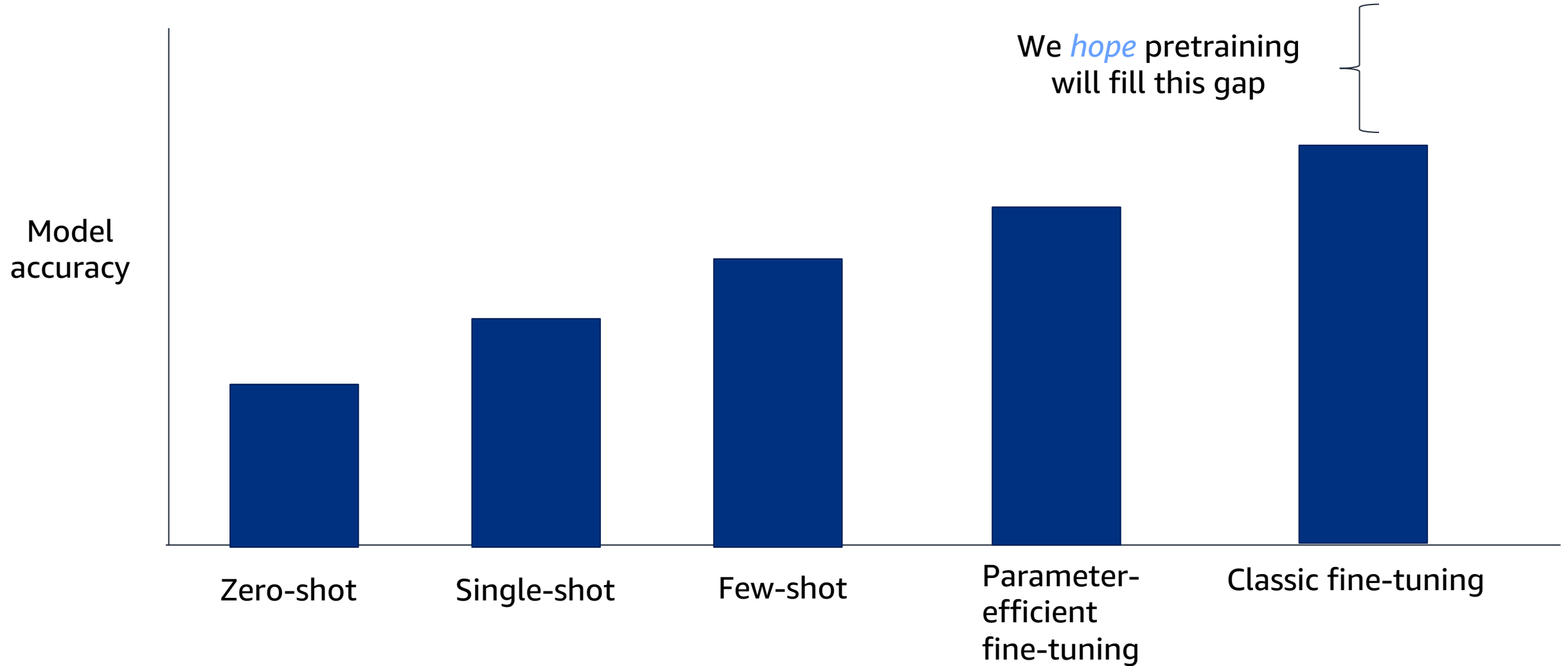
# At this point in your foundation model journey you should ...

---

1. Have tested many different foundation models with **prompt engineering**
2. Have tried a variety of **fine-tuning techniques** at different scales of models and dataset sizes
3. Have exposed these models to your end consumers and gotten **feedback** on their performance
4. Be able to empirically demonstrate where your current foundation models **both succeed and fail**



# To consider a pretraining project, you want a chart like this



# What does it take to pretrain a new foundation model?

---

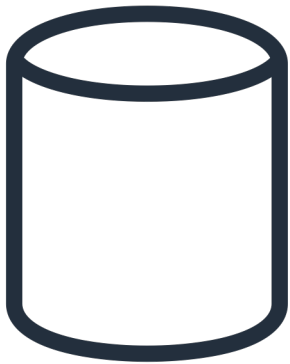
Model name	Dataset size	Model size in parameters	Cluster size	Time to train
Stable Diffusion 2.1	5B images, 240 TB	< 1 billion	37 p4d instances	28 days
Falcon	1T tokens, 2.8 TB	40B	48 p4d instances	Two months
BloombergGPT	700B tokens, 1.9 TB	50B	64 p4d instances	53 days

All trained on  
SageMaker



# What does it take to pretrain a new foundation model?

---



Data in TB's

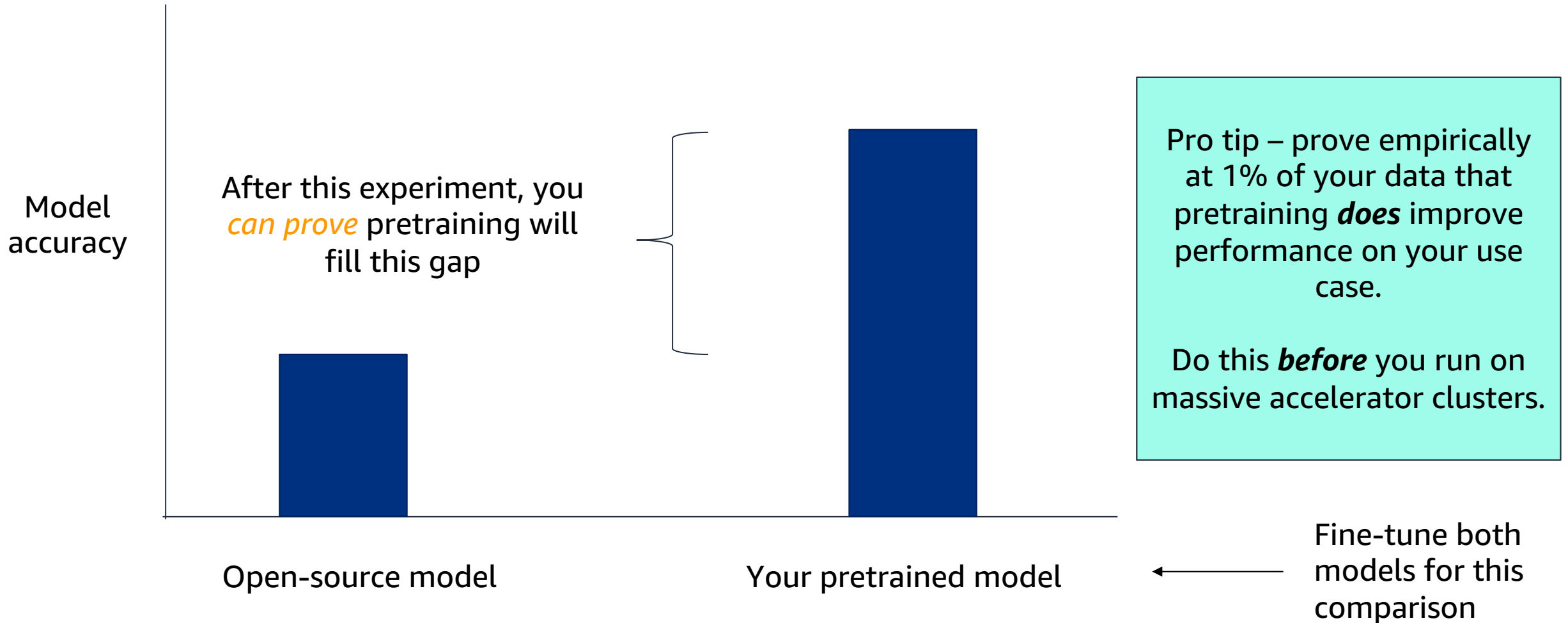


Tens of  
compute nodes



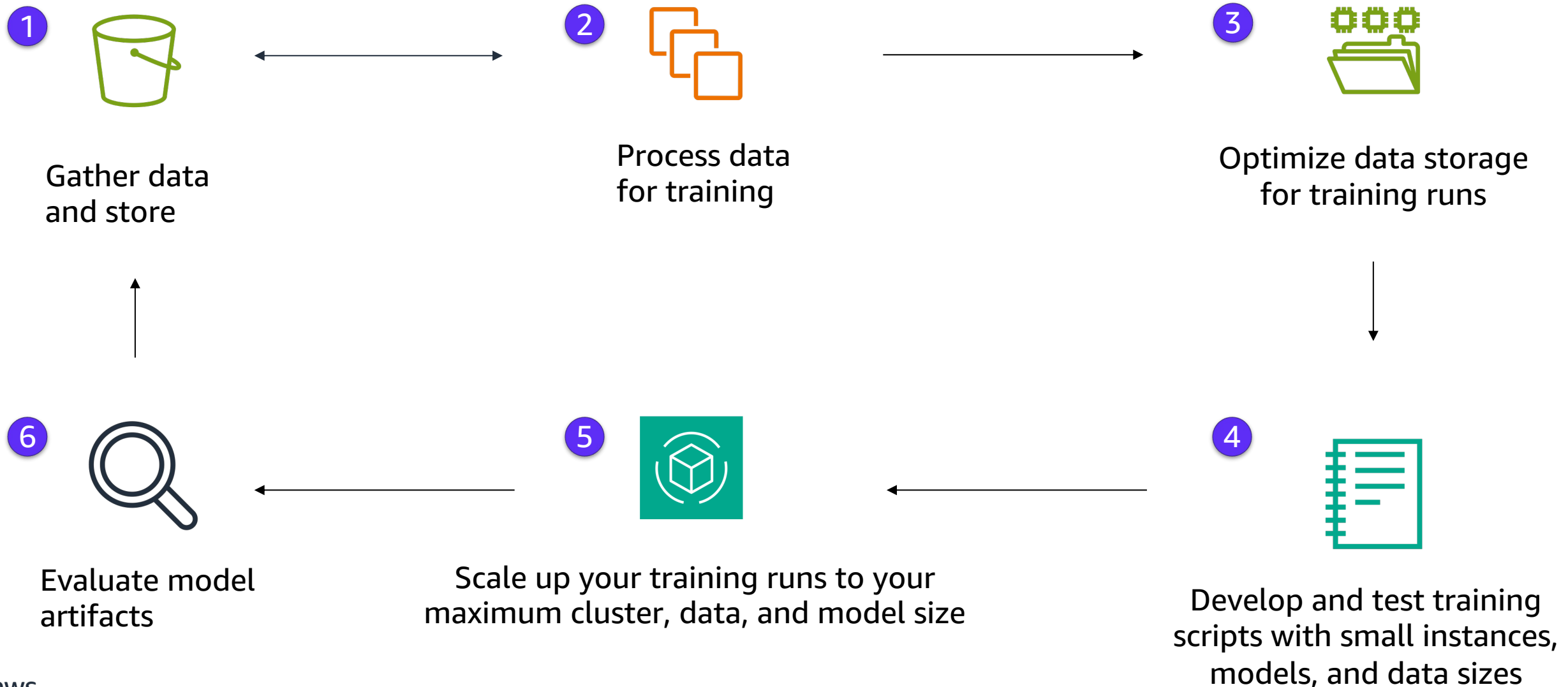
Really strong  
business case

# What to do before you launch all the accelerators





# How to pretrain foundation models on AWS





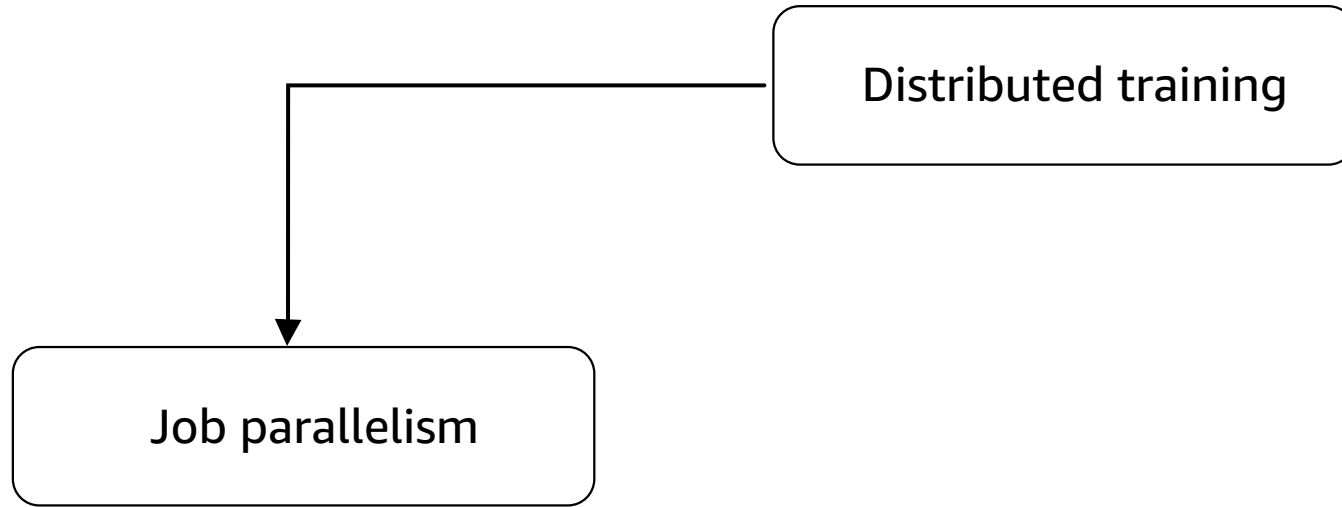
# What does this look like in action?

Phase Number	Dataset sample	Model size	Cluster size in accelerators	Development and compute time
1	1%	Base	1	Hours
2	5%	Medium	8	Days
3	50%	Large	16	Weeks
4	100%	Jumbo	Max	Months

- Set a plan for your project to **scale in steps**
- This gives you solvable goals that start at the smallest possible sizes and work your way up to hitting the largest compute size
- Make sure you test your model checkpoint at each step to ensure it's valid!

# There are many kinds of distributed training

---



# Run multiple jobs in parallel to process and train faster

1. Each job can train as many models as you need, or process as much data as you need.
2. You can use **warm pools** to reuse the instances

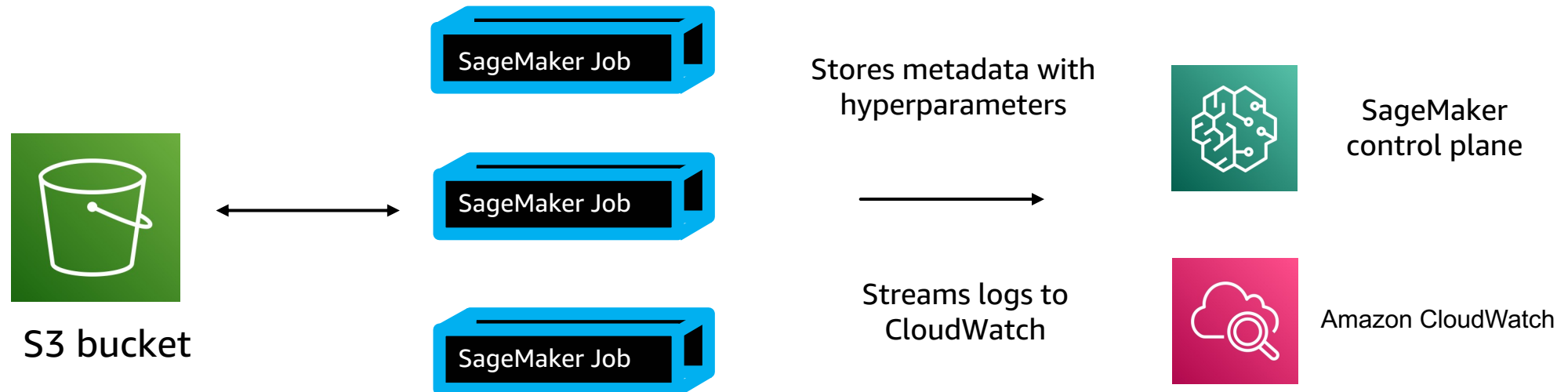
```
for model in list_of_models:
```

```
    s3_input = get_data(model)
```

```
    s3_output = get_location(model)
```

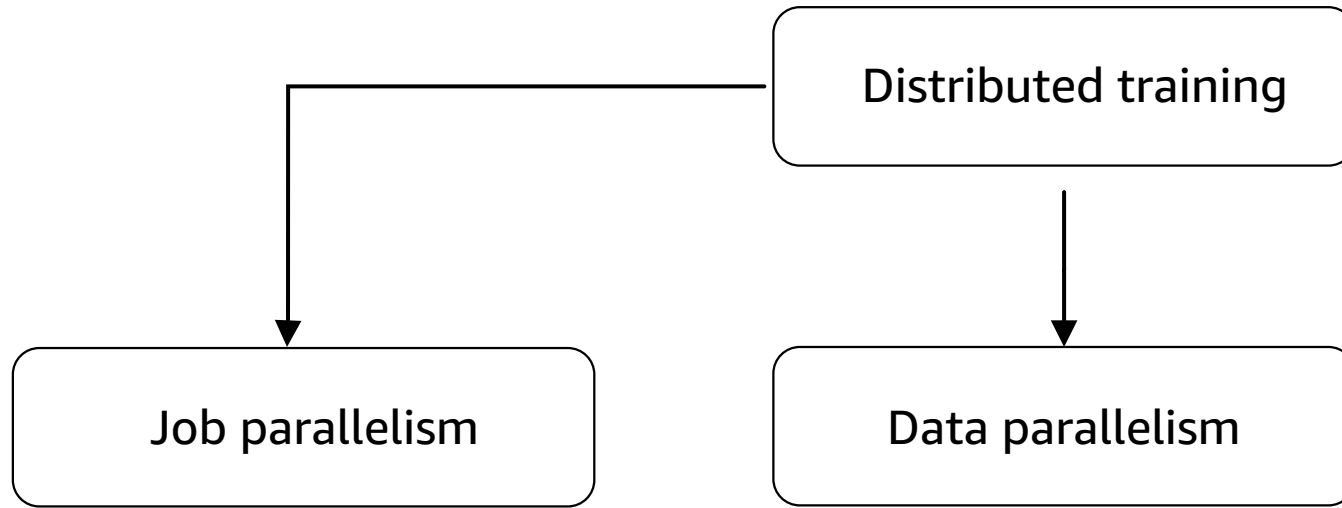
```
    estimator = get_estimator(model, s3_output)
```

```
    estimator.fit(s3_input, wait=False)
```



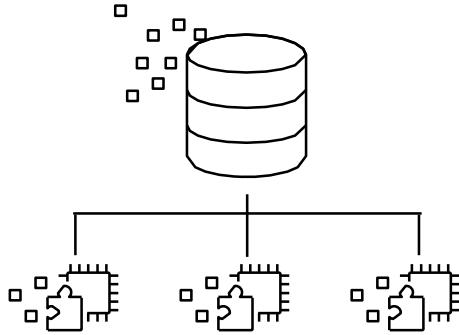
# There are many kinds of distributed training

---



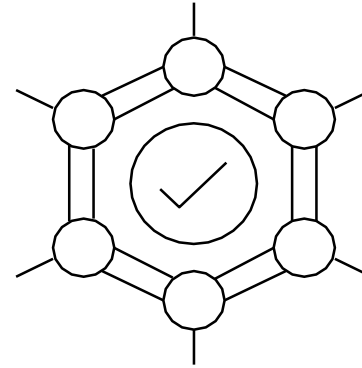
# Distributed gradient descent has evolved over time

---



Parameter server

E.g., TensorFlow  
ParameterServerStrategy



MPI AllReduce

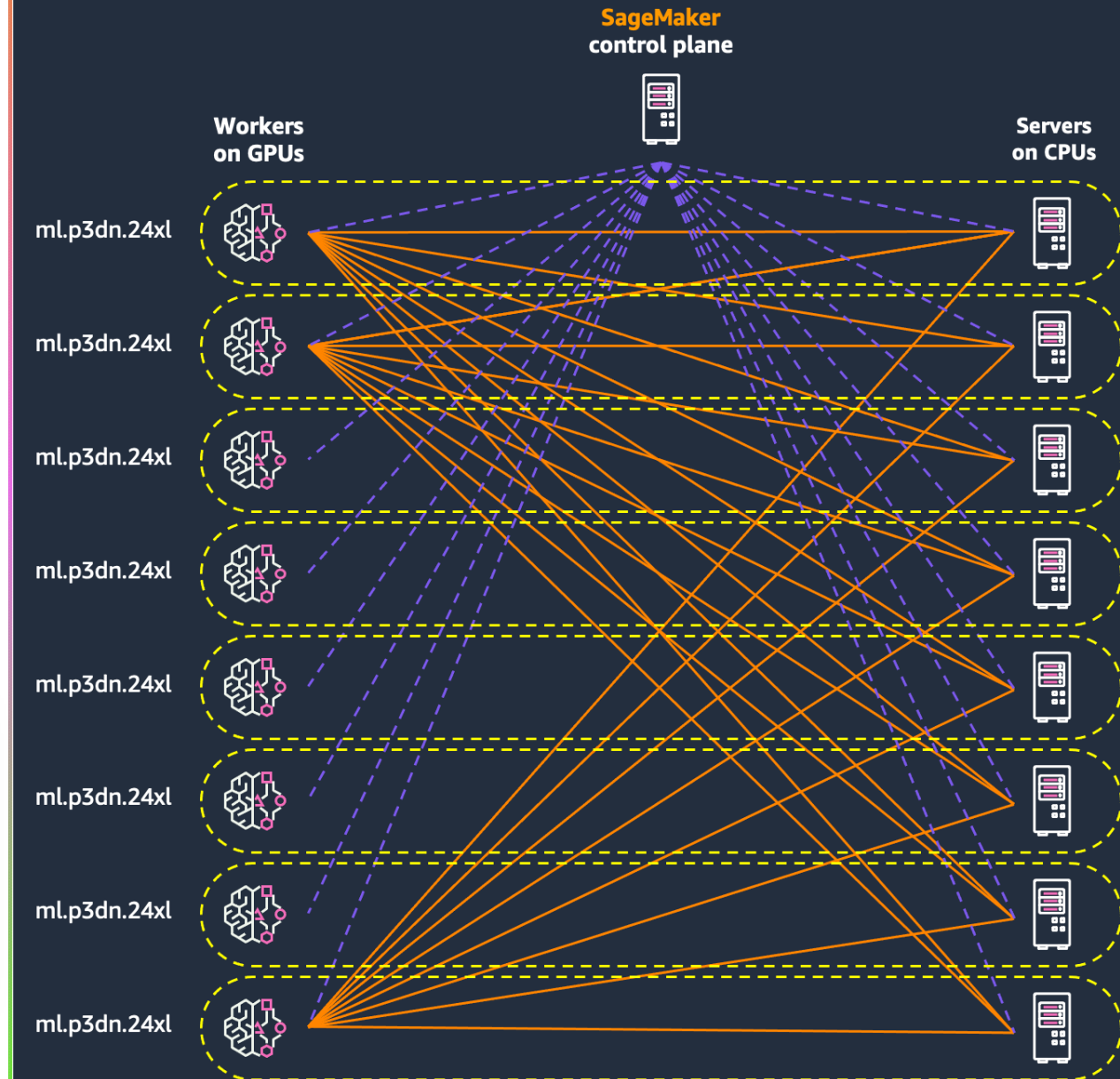
E.g., Horovod,  
PyTorch DistributedDataParallel

# SageMaker Distributed Data Parallel

- Optimized backend for distributed training of deep learning models in TensorFlow, PyTorch
- Accelerates training for network-bound workloads
- Built and optimized for AWS network topology and hardware
- 20%–40% faster and cheaper than NCCL and MPI-based solutions. Best performance on AWS for large clusters.

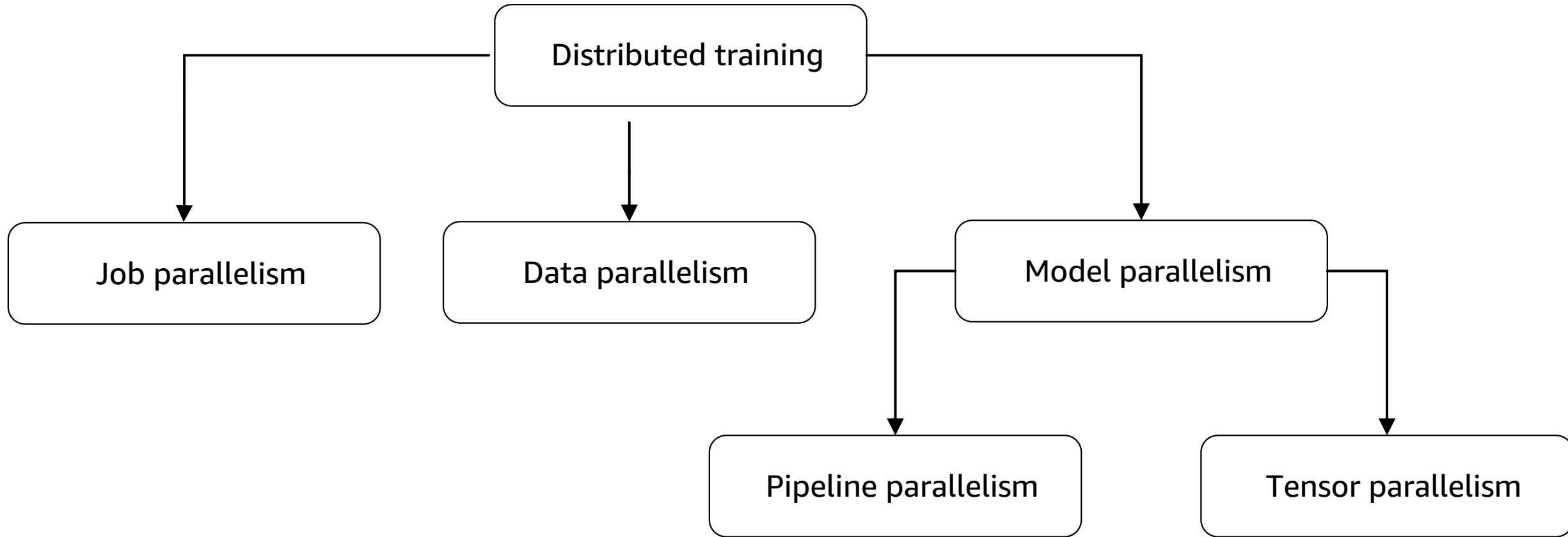
## Herring: Rethinking the Parameter Server at Scale for the Cloud

Indu Thangakrishnan, Derya Cavdar, Can Karakus,  
Piyush Ghai, Yauheni Selivonchyk, Cory Puce  
*Amazon Web Services*  
{thangakr, dcavdar, cakarak, ghaipiyu, yauheni, cpruce}@amazon.com



# There are many kinds of distributed training

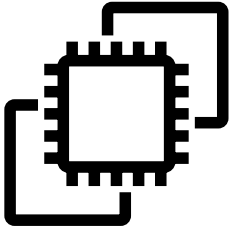
---



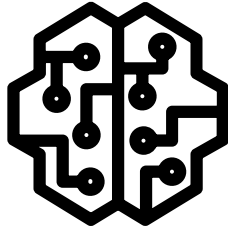


# SageMaker model parallel

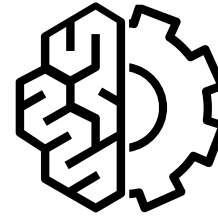
---



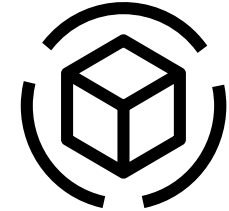
**Automated  
model partitioning**



**Interleaved  
pipelined training**



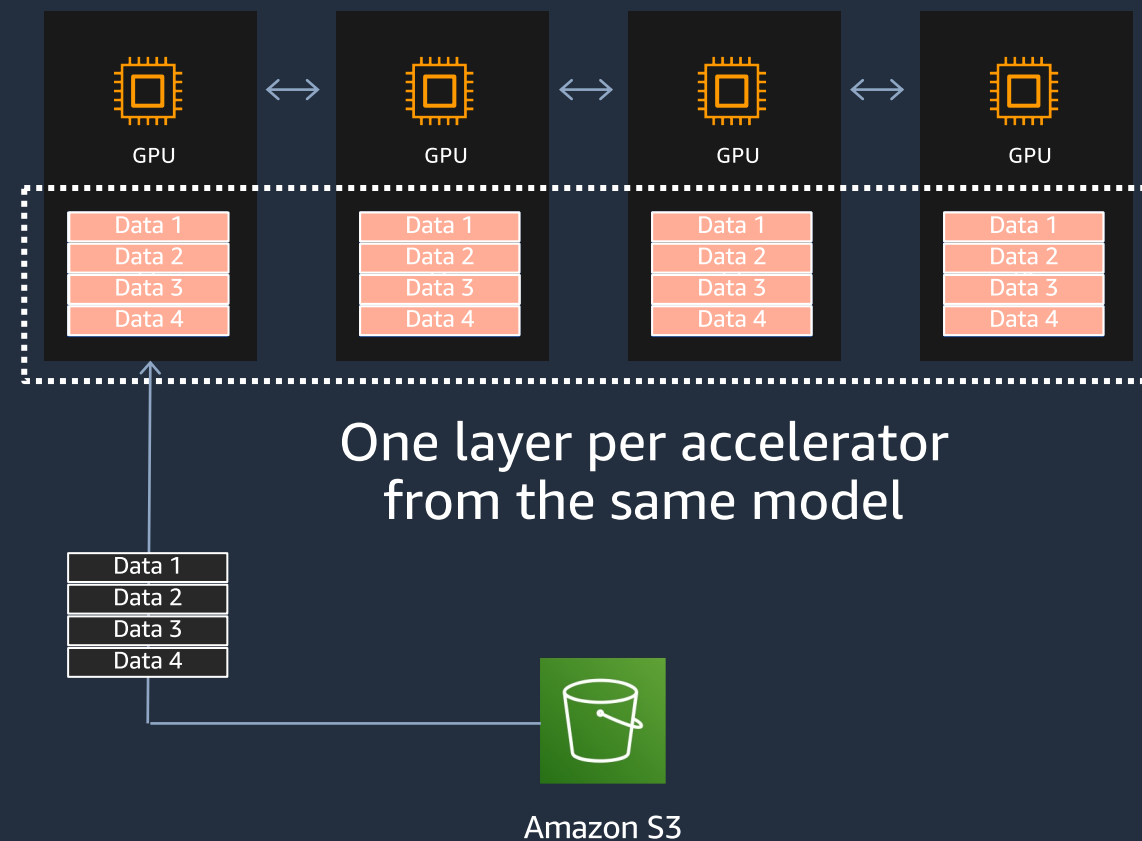
**Managed  
SageMaker training**



**Clean  
framework integration**

# SageMaker Model Parallel splits your model over multiple accelerators

- Split minibatches into N “microbatches”
- Feed microbatches sequentially, but process them to keep GPU utilization more even
- Minimize “idle” time on GPUs

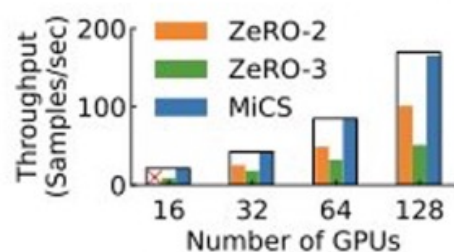


**AMAZON SAGEMAKER MODEL PARALLELISM: A GENERAL AND FLEXIBLE  
FRAMEWORK FOR LARGE MODEL TRAINING**

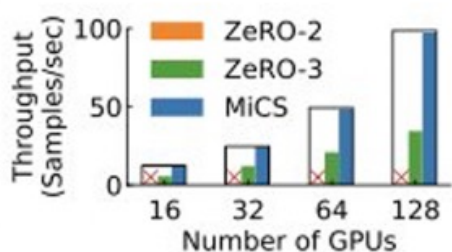
Can Karakus<sup>1</sup> Rahul Huilgol<sup>1</sup> Fei Wu<sup>1</sup> Anirudh Subramanian<sup>1</sup> Cade Daniel<sup>1</sup> Derya Cavdar<sup>1</sup> Teng Xu<sup>1</sup>  
Haohan Chen<sup>1</sup> Arash Rahn timer<sup>1</sup> Luis Quintela<sup>1</sup>

# Approach linear-scaling with *Sharded Data Parallelism*

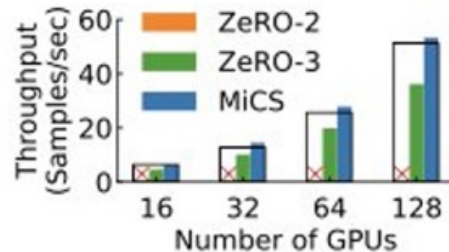
MiCS achieves 169 TFLOPS per GPU with 175B parameter model on AWS p4de.24xlarge instances



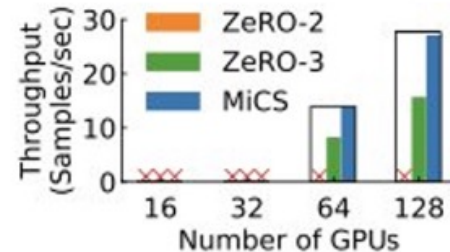
(a) BERT 10B.



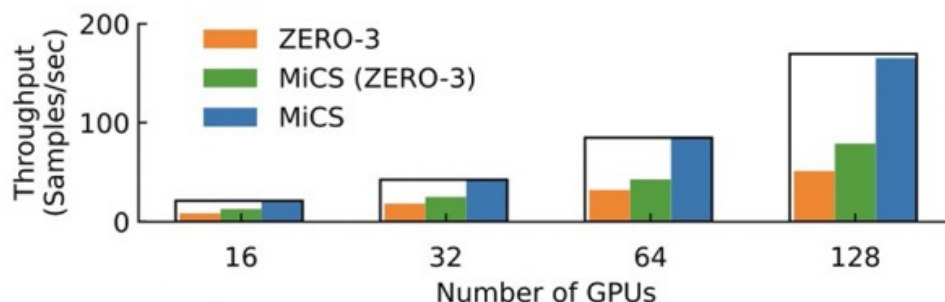
(b) BERT 15B.



(c) BERT 20B.



(d) BERT 50B.



- MiCS hits 99.4% of linear-scaling efficiency from 128 to 512 GPUs
- DeepSpeed hits only 72% , saturates at 62 TFLOPS per GPU

Available within SageMaker Model Parallel  
2.8x faster than DeepSpeed

**MiCS: Near-linear Scaling for Training Gigantic Model on Public Cloud**

Zhen Zhang\*  
Johns Hopkins University  
zzhen1@jhu.edu

Justin Chiu  
Amazon  
justchiu@amazon.com

Shuai Zheng  
Amazon Web Services  
shzheng@amazon.com

George Karypis  
Amazon Web Services  
gkarypis@amazon.com

Yida Wang  
Amazon Web Services  
wangyida@amazon.com

Trishul Chilimbi  
Amazon  
trishulc@amazon.com

Mu Li  
Amazon Web Services  
mli@amazon.com

Xin Jin  
Peking University  
xinjinpku@pku.edu.cn



# Get started with SageMaker distributed training

---



Example notebooks



Set as your backend



Add to Docker files



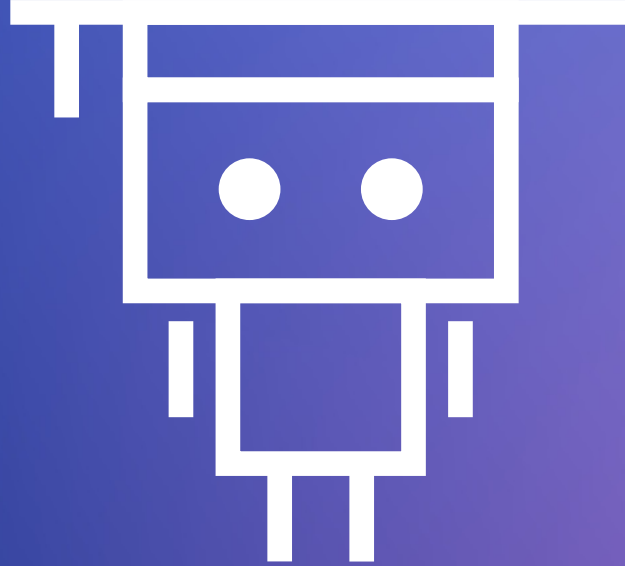
<https://bit.ly/sm-nb-4>

# Hands-on demo



```
amazon-sagemaker-examples / training / distributed_training / pytorch / model_parallel  
/ gpt2  
/ smp-train-gpt-sharded-data-parallel.ipynb
```





# Thank you!

Type: Corrections, feedback, or other questions?  
Contact us at <https://support.awsamazon.com/#/contacts/aws-academy>.  
All trademarks are the property of their owners.