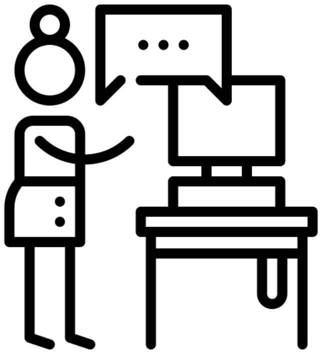
Cerebro: A Layered Data Platform for Scalable Deep Learning

Yuhao Zhang and Supun Nakandala
CSE Department, University of California, San Diego

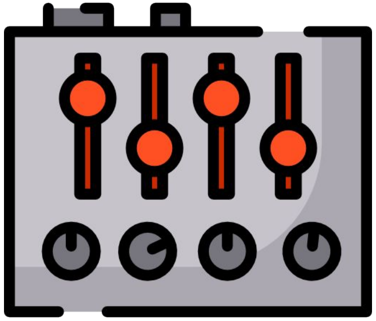
Deep Learning

Artificial Neural Networks (ANNs) are revolutionizing many domains - "Deep Learning"

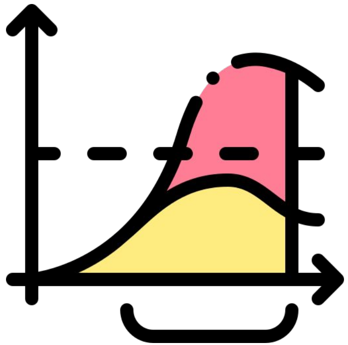


Model Selection

Out-of-box and simple



Complex and requires model selection (hyper-parameter tuning + architecture selection)



Non-linear -> trial and error

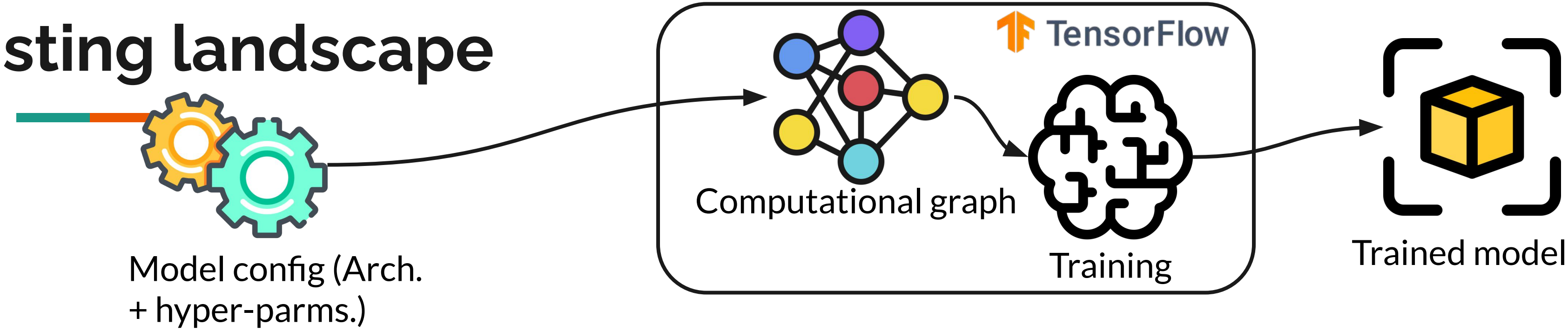
Model arch.: {VGG, ResNet, InceptionNet, Inception-ResNet ...}
Learning rate: {1e-3, 1e-4, 1e-5, 1e-6 ..}
Regularization: {1e-3, 1e-4, 1e-5, 1e-6 ..}
Batch size: {8, 32, 64, 128 ...}

4x4x4x4 = 256 options !

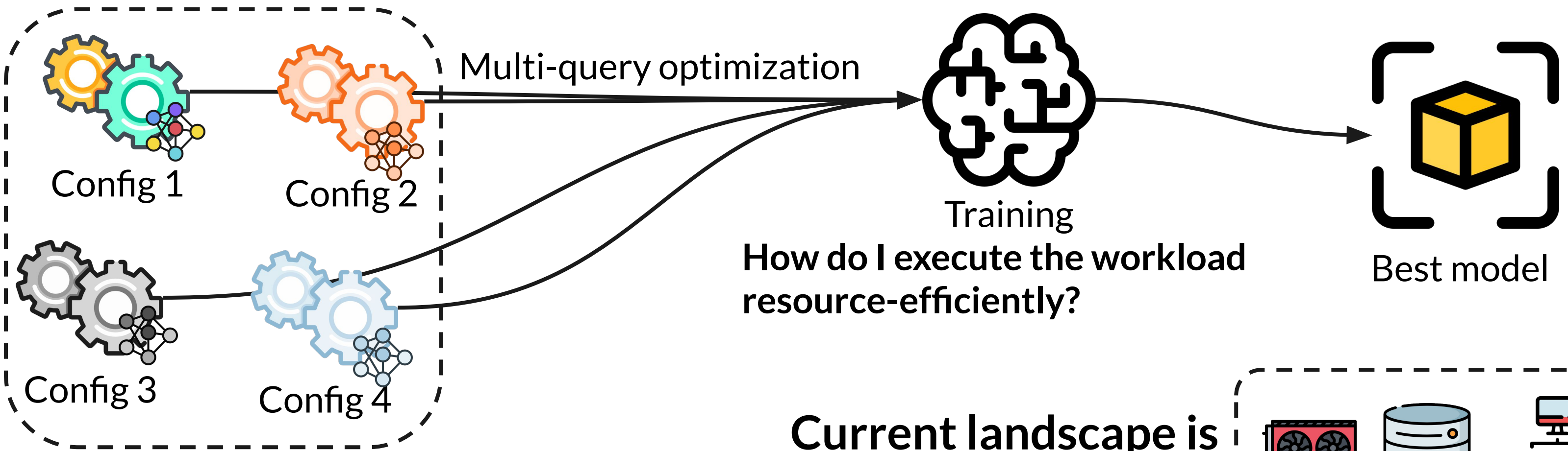
An engineer may need to attempt hundreds of models before picking the best one*

*Facebook Blog: Introducing FBLearner Flow: Facebooks AI backbone.
https://code.fb.com/ml-applications/introducing-fblearner-flow-facebook-s-ai-backbone

Existing landscape



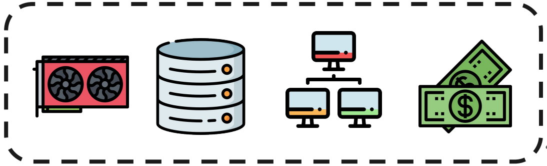
Reality



How do I execute the workload resource-efficiently?

How do I specify a model selection workload?

Current landscape is wasteful of:



Resources cost: an example

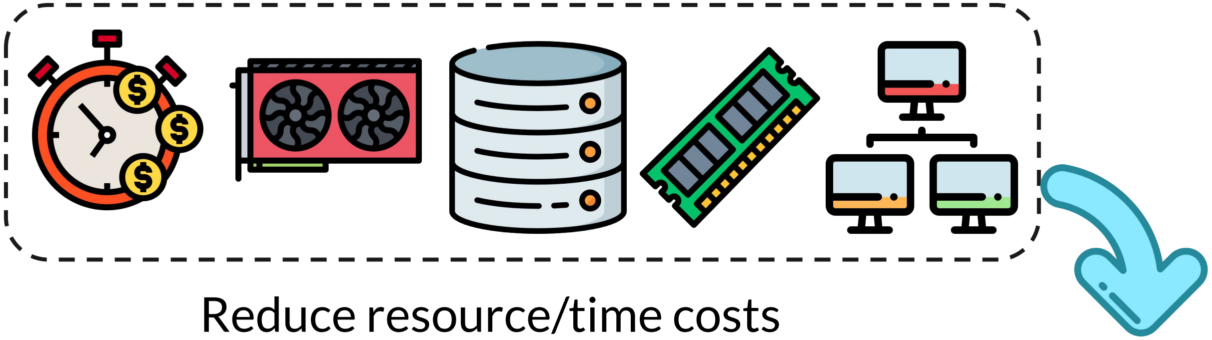
4789 models were trained during the R&D of LISA, a state-of-art NLP model

# of Models	GPU Time	Estimated cost (USD)	
		Cloud compute	Electricity
1	120 hrs	\$52–\$175	\$5
4789	**27 yrs**	**\$103k–\$350k**	**\$9870**

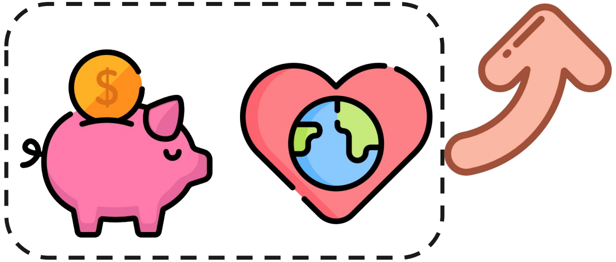
E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. In ACL, 2019.

Takeaways

1. Model selection deserves to be first-class citizen
2. Usability: need high-level model building APIs
3. Efficiency: need optimizations



Reduce resource/time costs

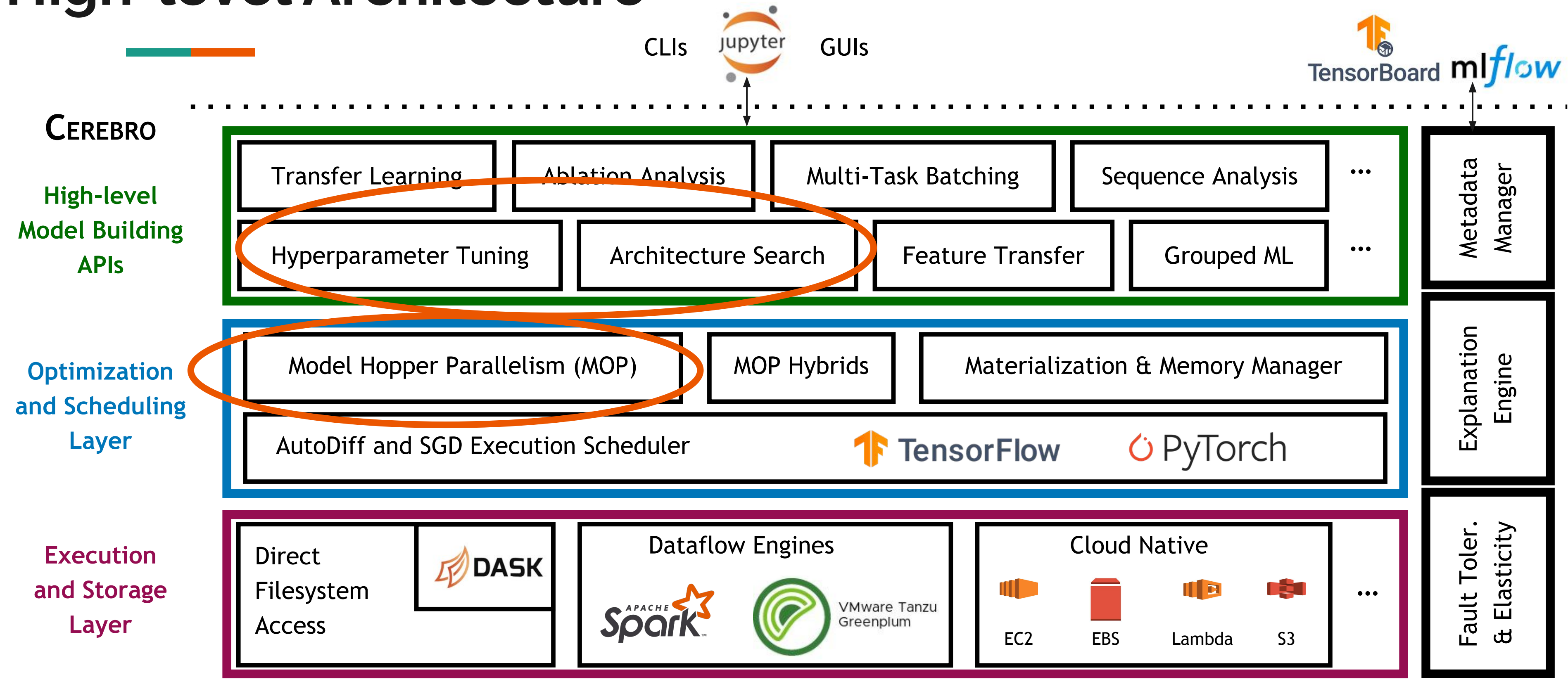


Save money/energy

Outline

1. Motivation
2. **High-level (layered) Architecture**
3. Execution Optimizations
4. Recent and Ongoing Research
5. Summary

High-level Architecture

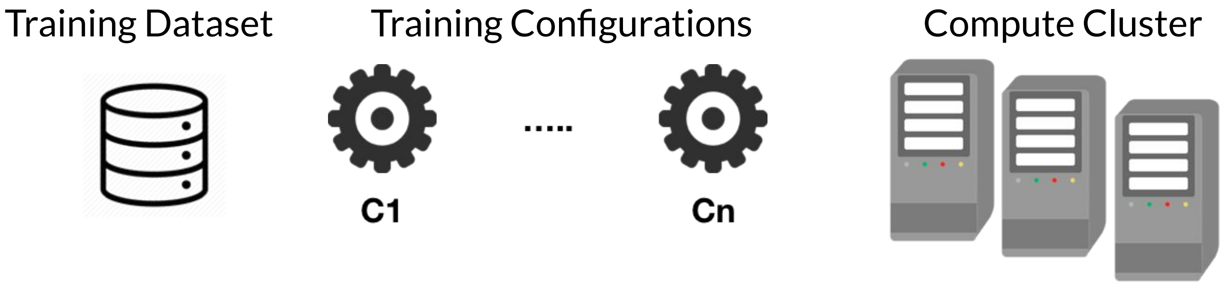


Outline

1. Motivation
2. High-level (layered) Architecture
3. Execution Optimizations
    a. Limitations of Existing Approaches
    b. Our Solution: Model Hopper Parallelism (MOP)
4. Recent and Ongoing Research
5. Summary

Existing Approaches

We are given three things:



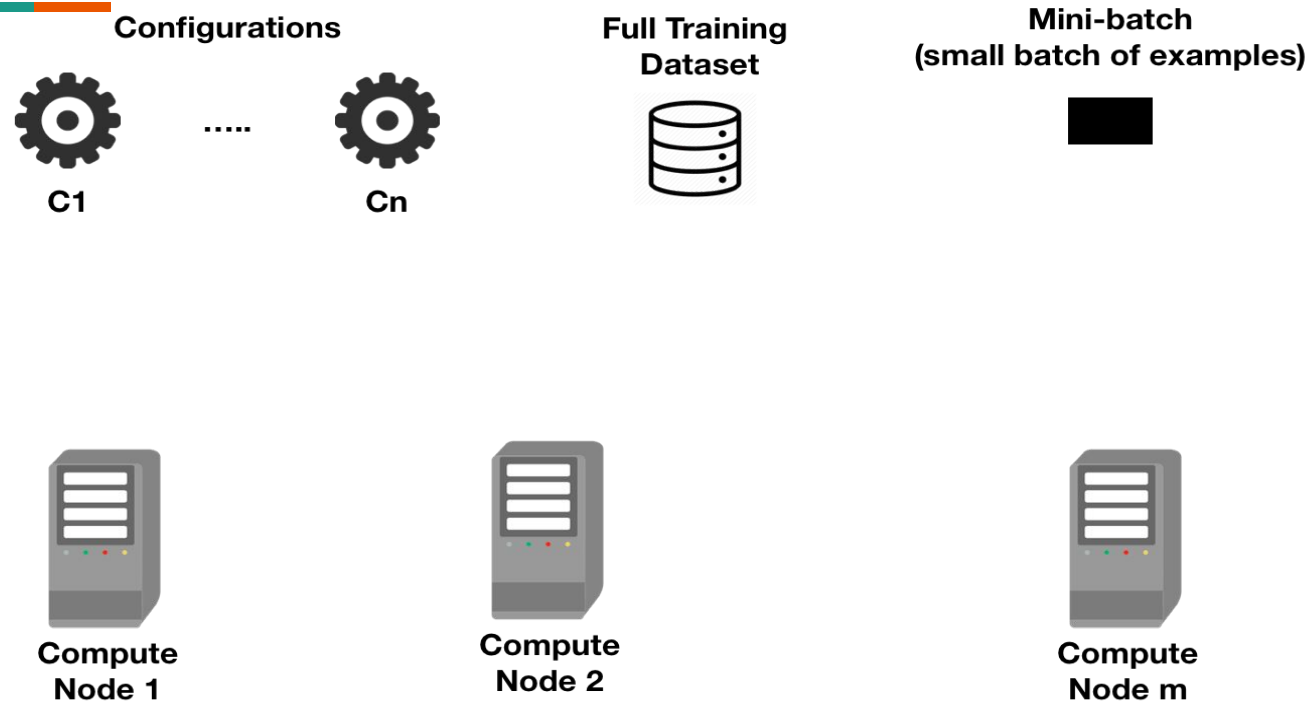
Existing systems to speed up model selection aim to exploit the parallelism of a cluster to raise throughput.

But all such systems suffer from major inefficiency or other.

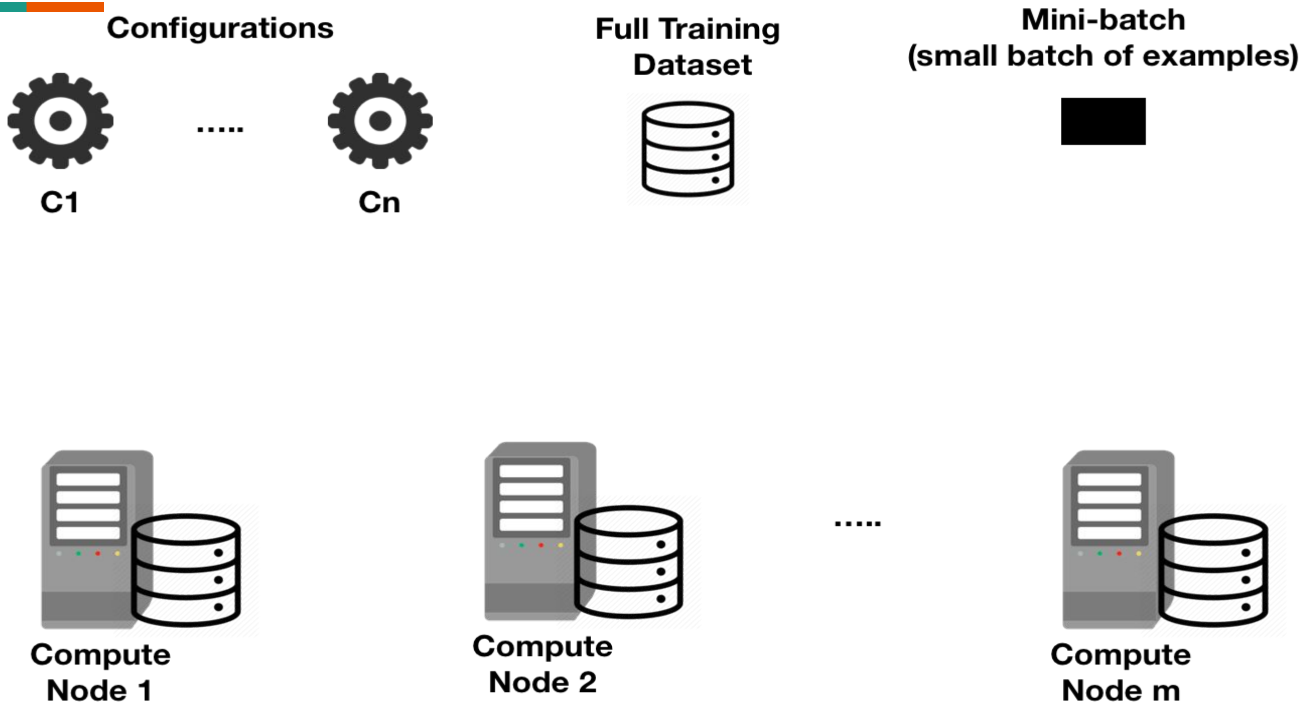
Task Parallelism
Multiple workers each training a single model

Data Parallelism
Single model training on multiple workers

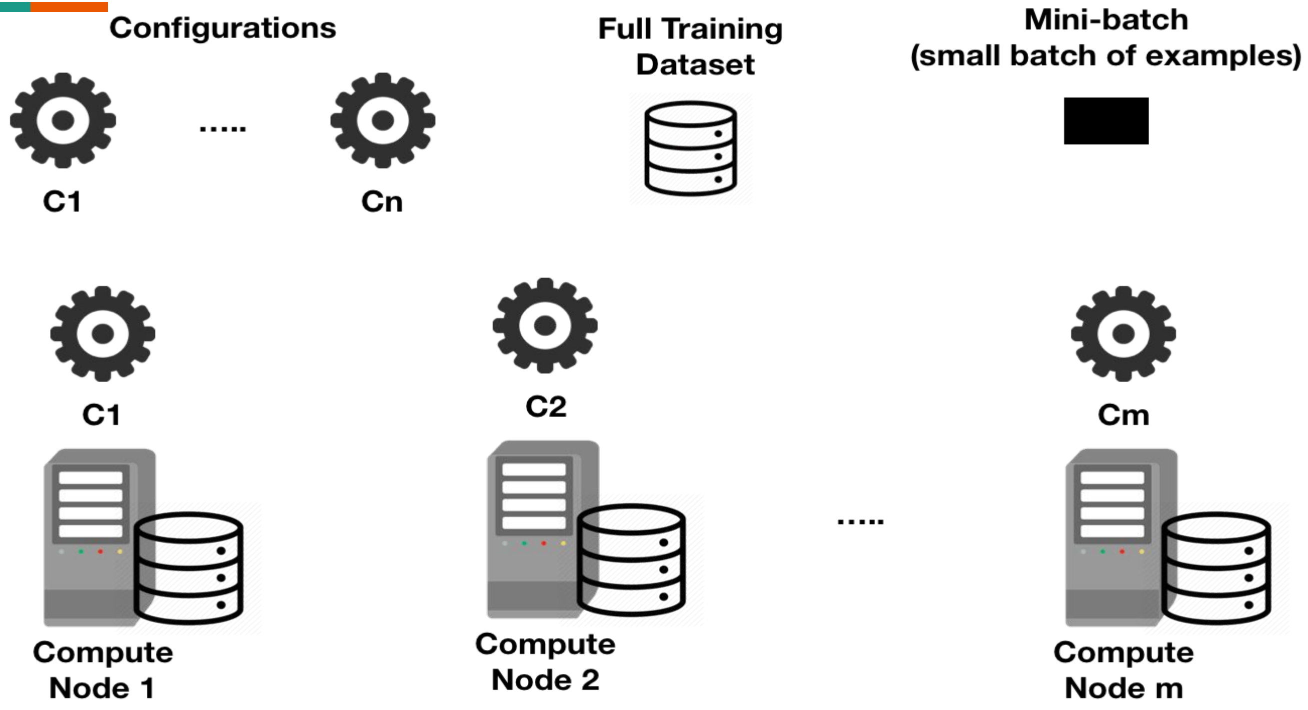
Task Parallelism (e.g., Ray, Dask, Celery)



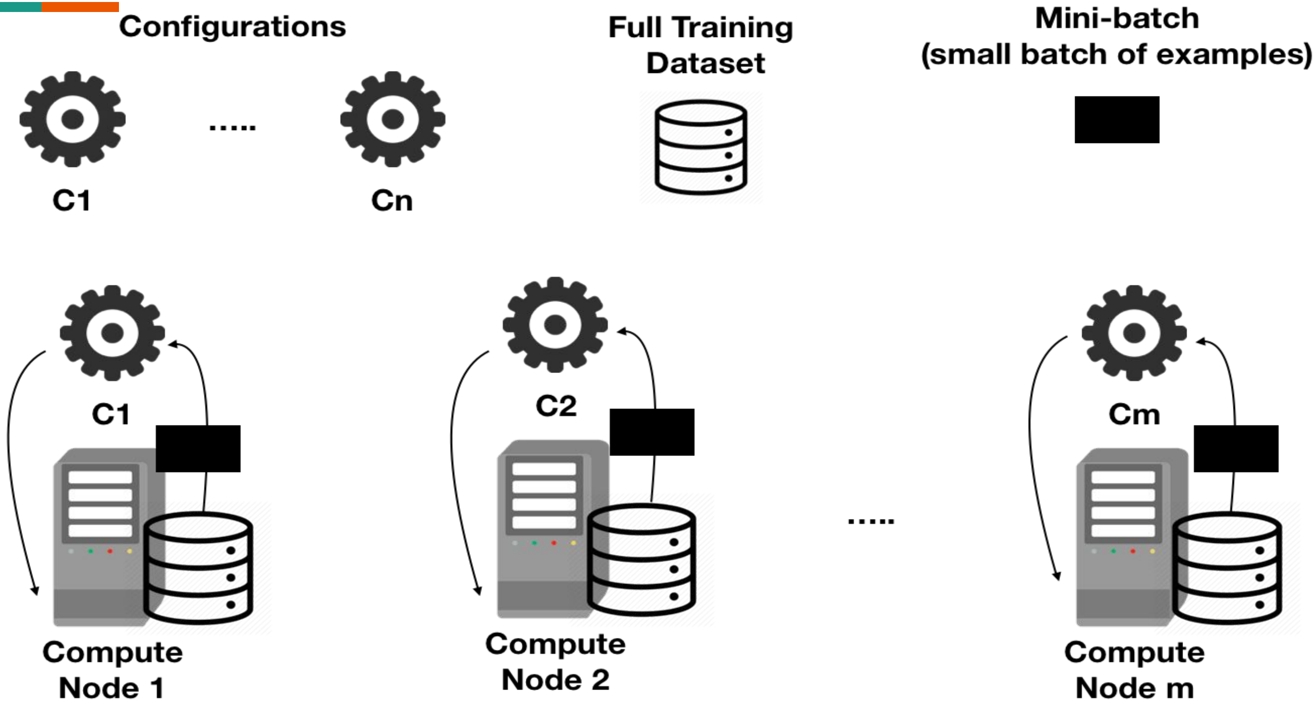
Task Parallelism (e.g., Ray, Dask, Celery)



Task Parallelism (e.g., Ray, Dask, Celery)



Task Parallelism (e.g., Ray, Dask, Celery)



Con: Wastes storage/memory (or network)

Existing Approaches

We are given three things:

Training Dataset

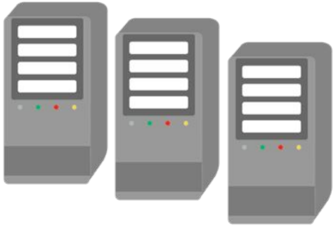
Training Configurations

C1

.....

Cn

Compute Cluster

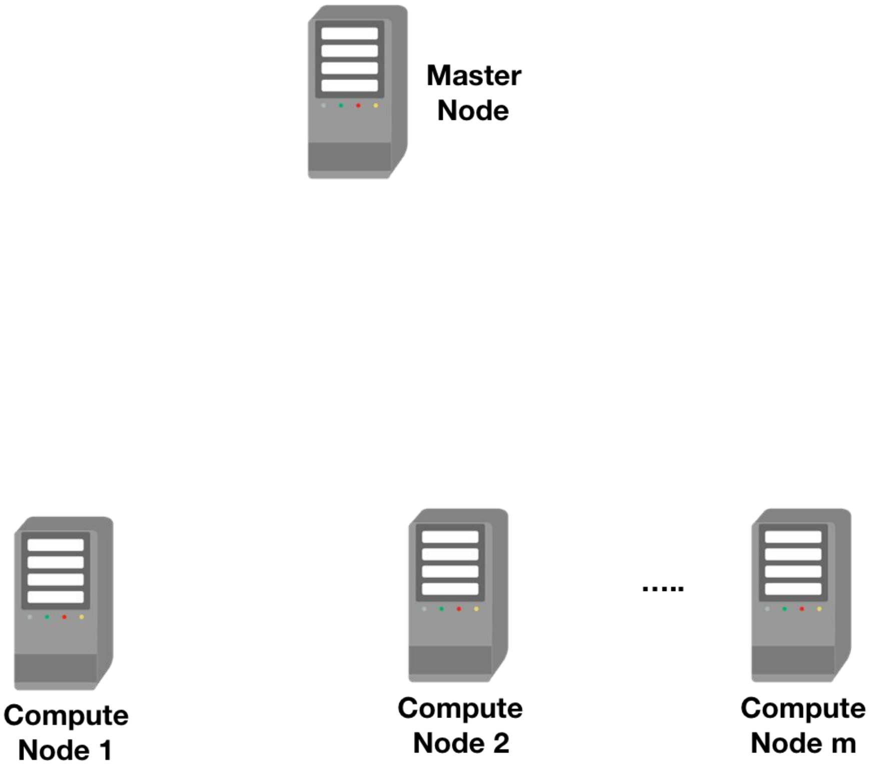


Existing systems to speed up model selection aim to exploit the parallelism of a cluster to raise throughput.

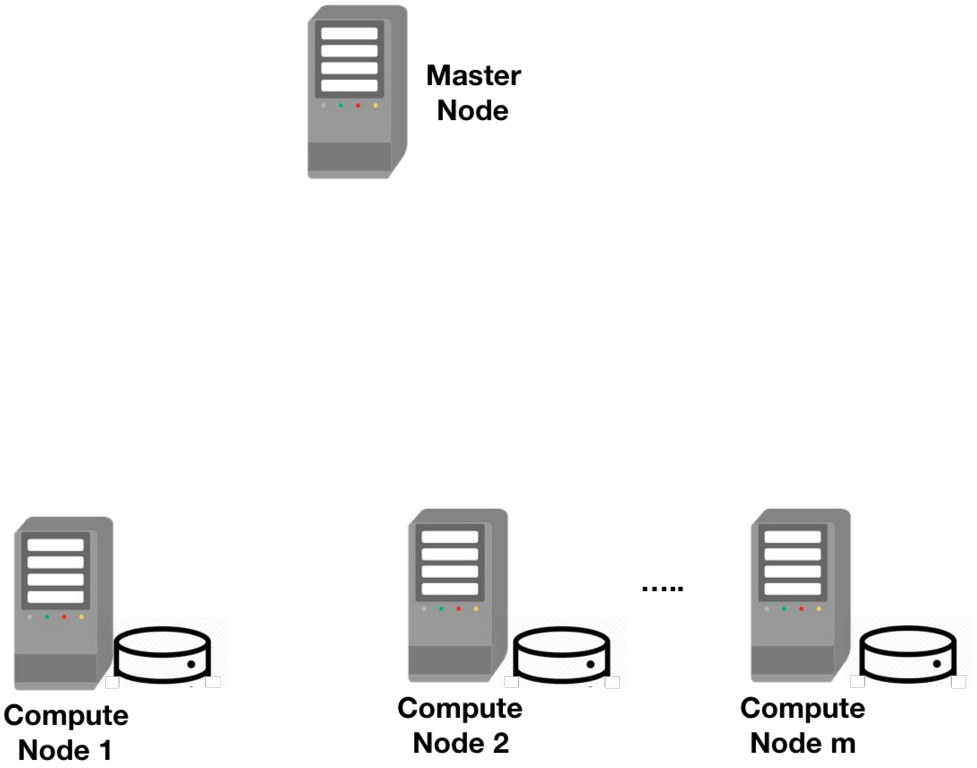
But all such systems suffer from major inefficiency or other.

Task Parallelism Multiple workers each training a single model	**Data Parallelism** Single model training on multiple workers

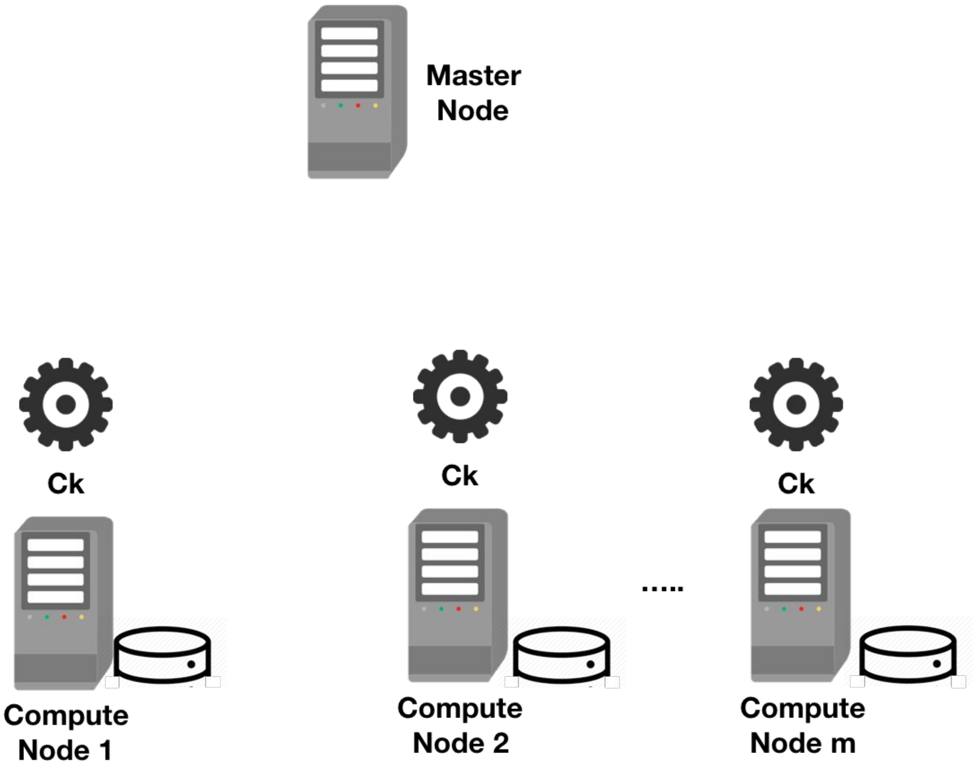
Data Parallelism (e.g., Parameter Server, Horovod)



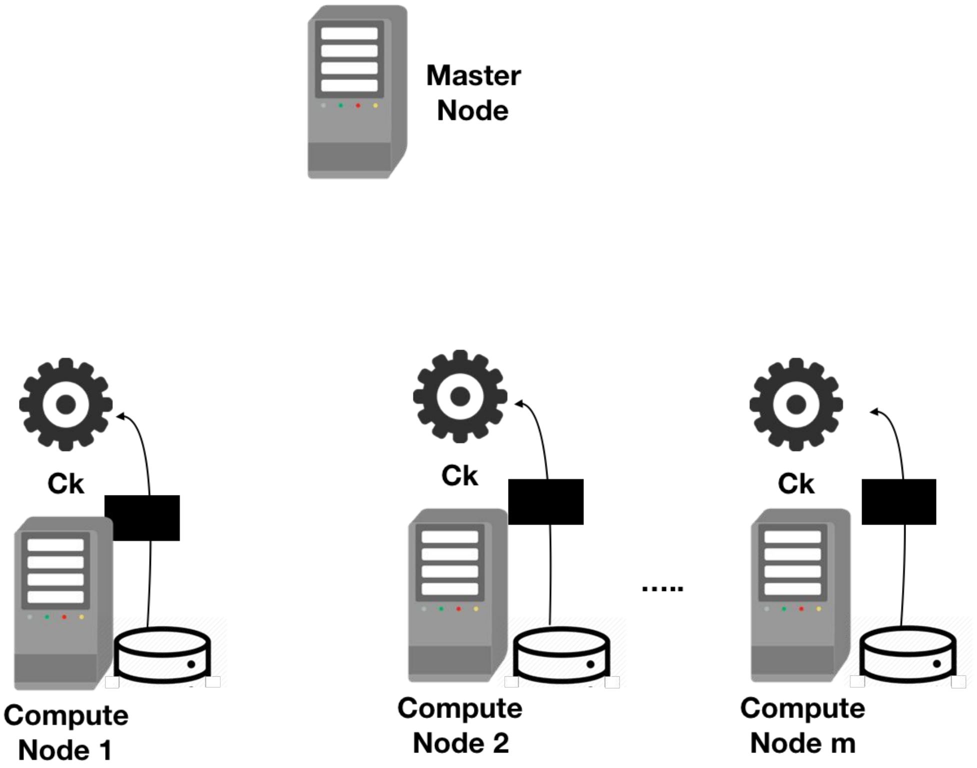
Data Parallelism (e.g., Parameter Server, Horovod)



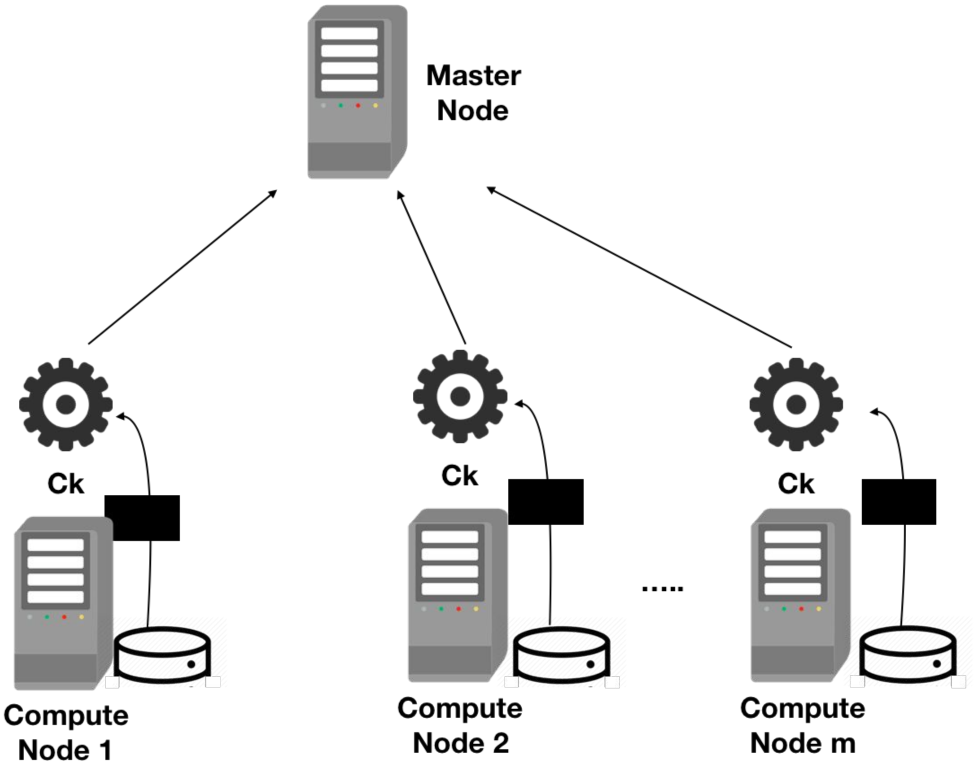
Data Parallelism (e.g., Parameter Server, Horovod)



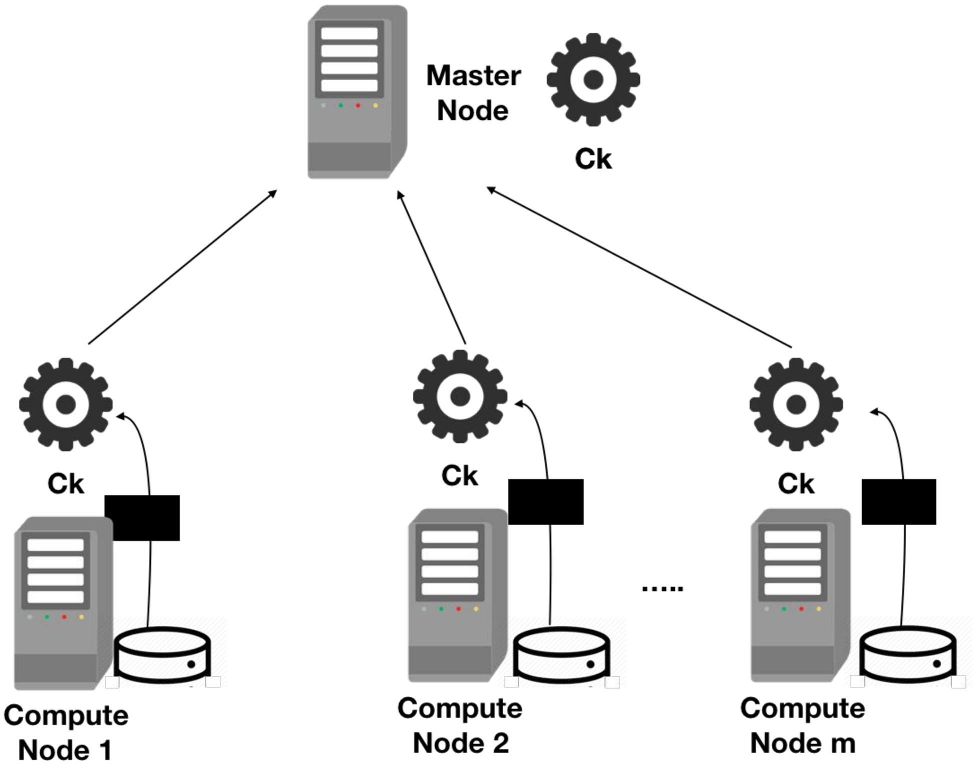
Data Parallelism (e.g., Parameter Server, Horovod)



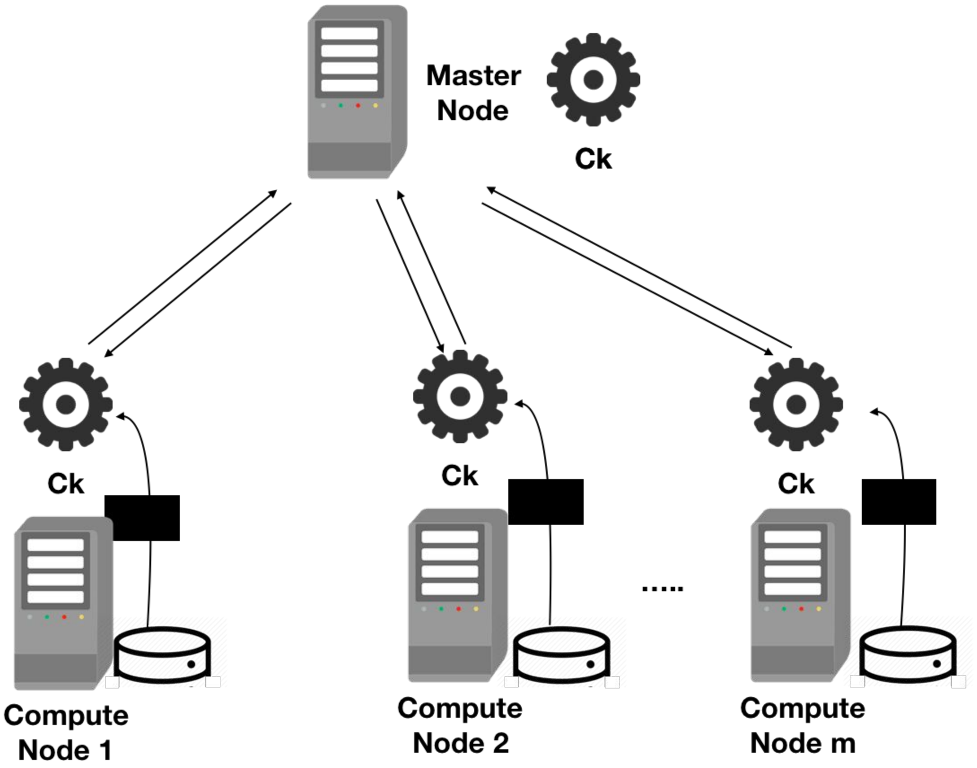
Data Parallelism (e.g., Parameter Server, Horovod)



Data Parallelism (e.g., Parameter Server, Horovod)



Data Parallelism (e.g., Parameter Server, Horovod)



Update after every mini-batch:

E.g., TensorFlow Parameter Server, Horovod

Con: High communication cost

?

Task Parallelism

Pro: High throughput

Con: Low data scalability

Con: Storage/memory wastage

+

Data Parallelism

Pro: High data scalability

Con: Low throughput

Con: High communication cost

Outline

Task Parallelism

Pro: High throughput

Con: Low data scalability

Con: Storage/memory wastage

+

Data Parallelism

Pro: High data scalability

Con: Low throughput

Con: High communication cost

↓

Model Hopper Parallelism (Cerebro)

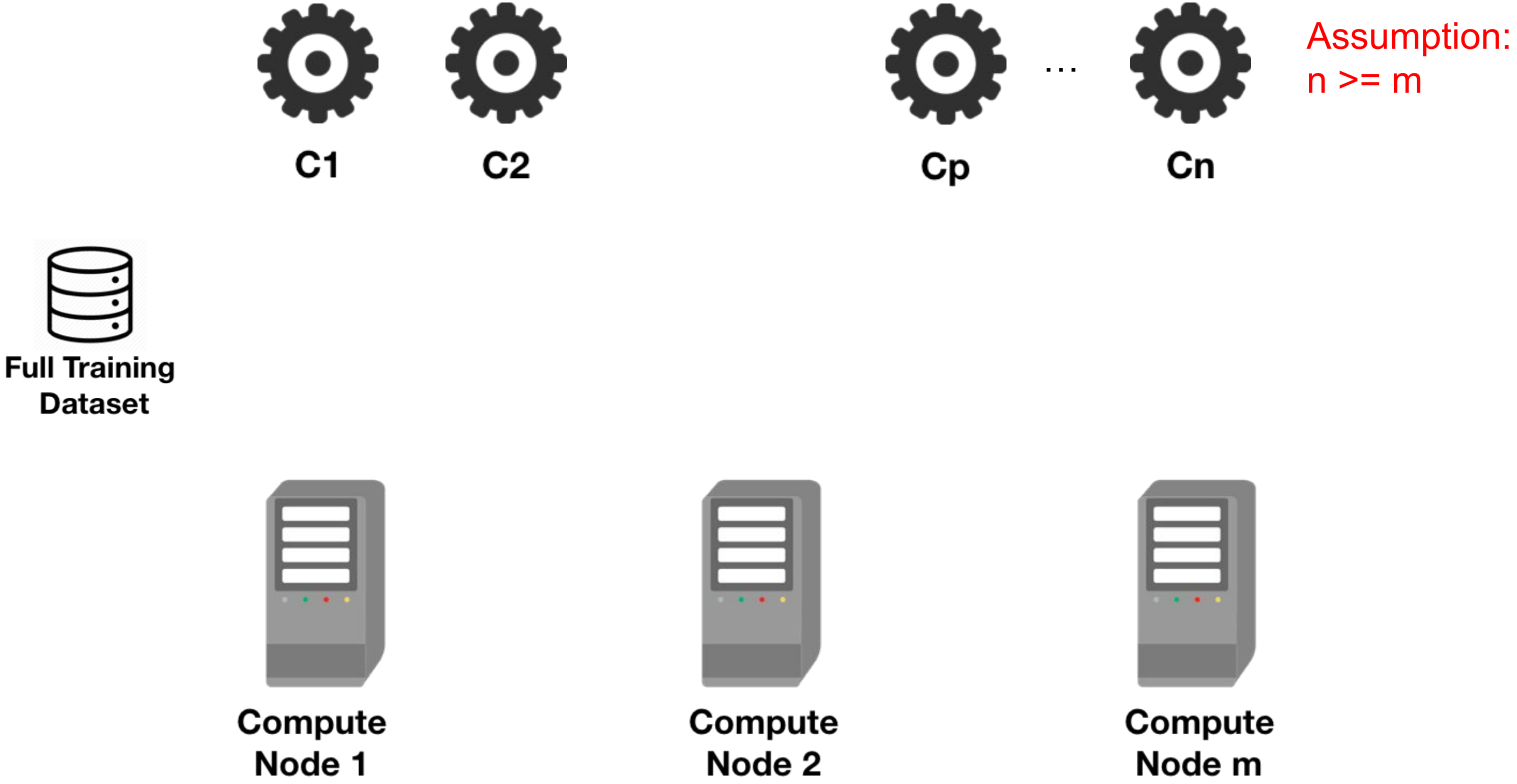
Pro: High throughput

Pro: High data scalability

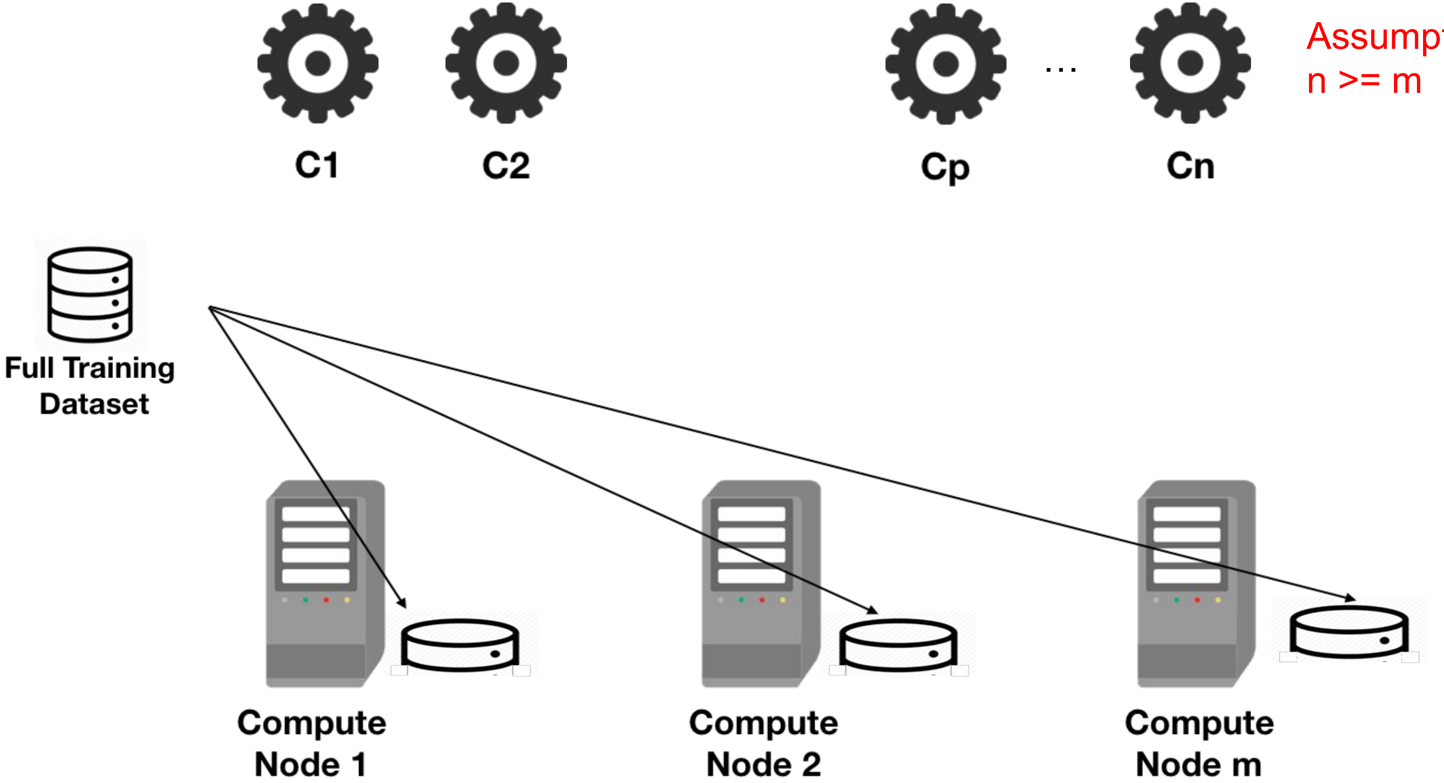
Pro: Low communication cost

Pro: No storage/memory wastage

Model Hopper Parallelism (MOP)

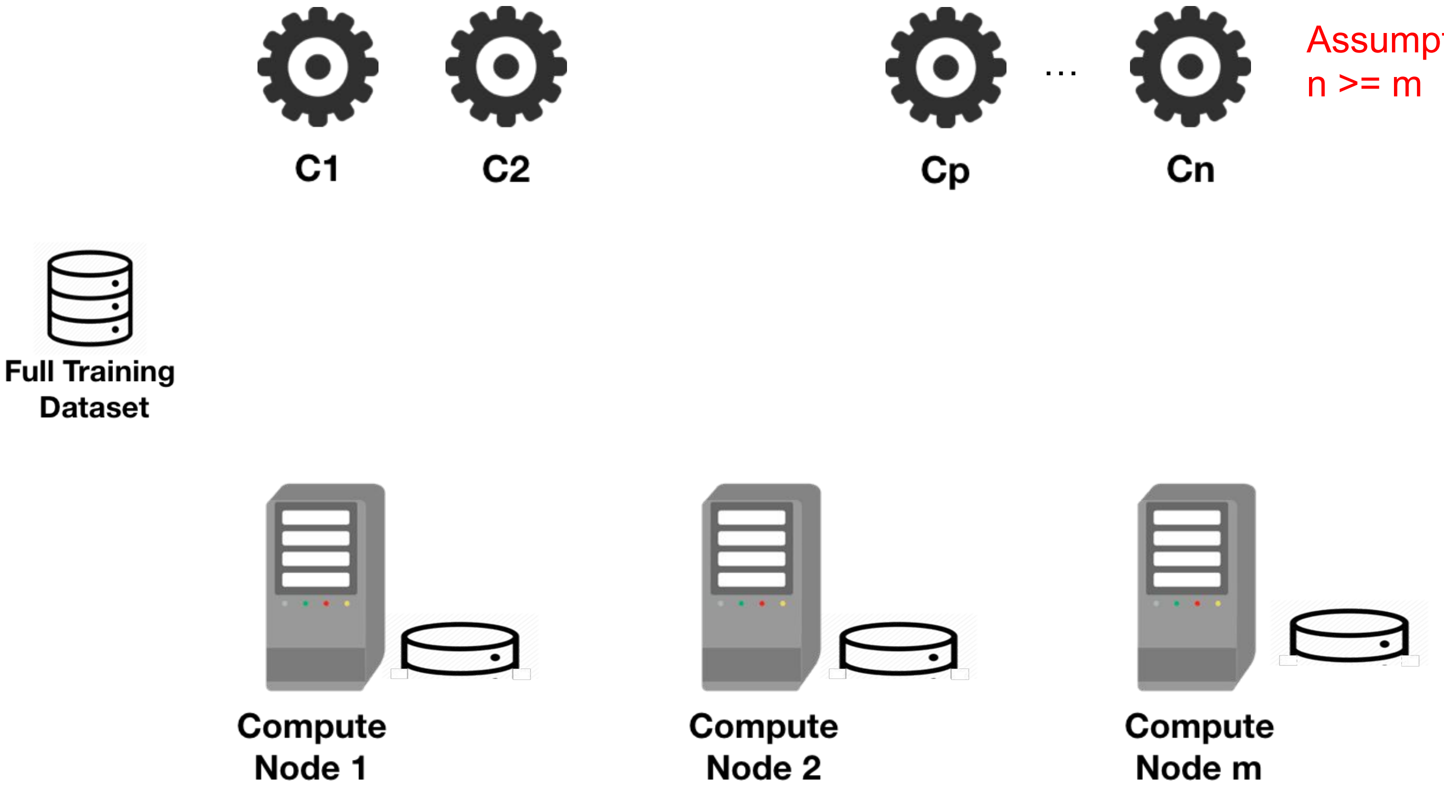


Model Hopper Parallelism (MOP)



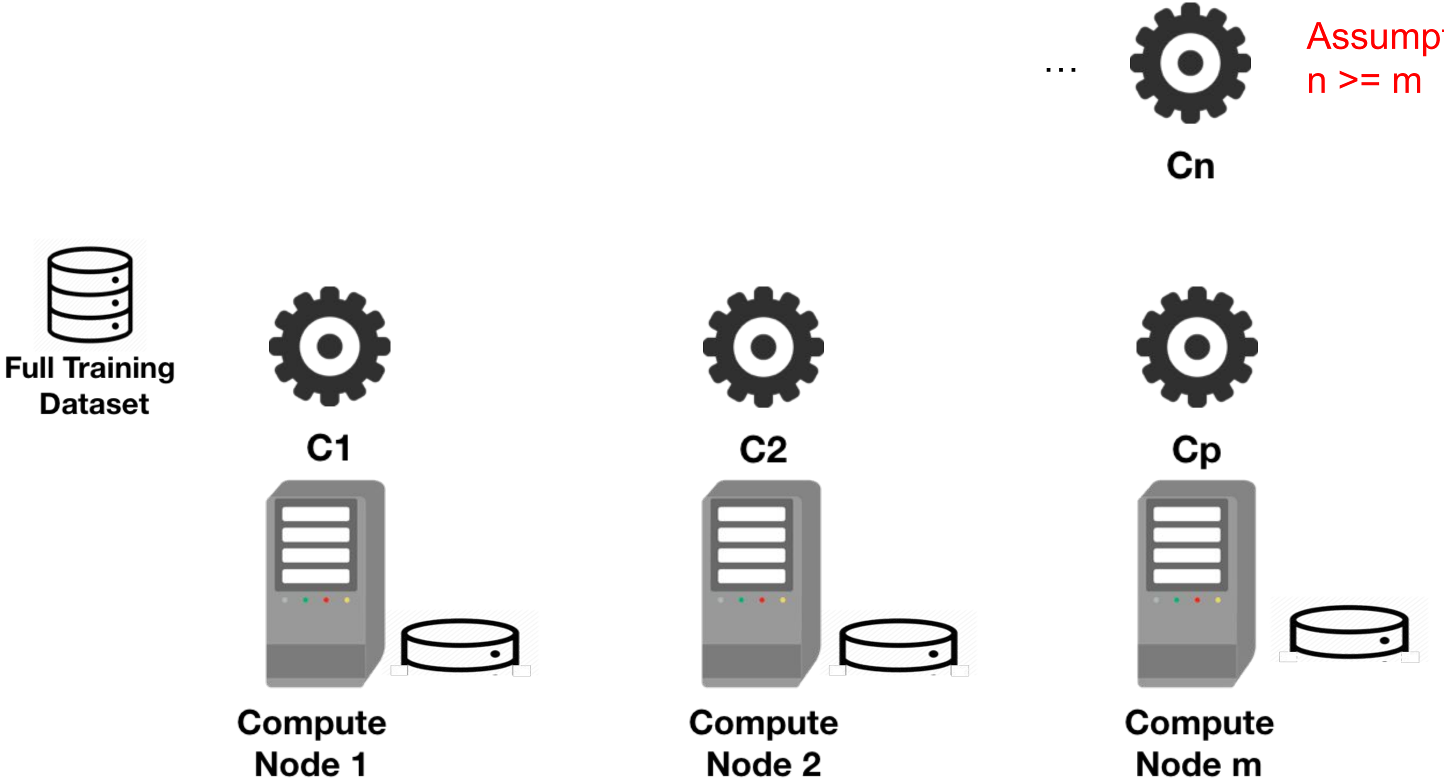
Assumption:
n >= m

Model Hopper Parallelism (MOP)



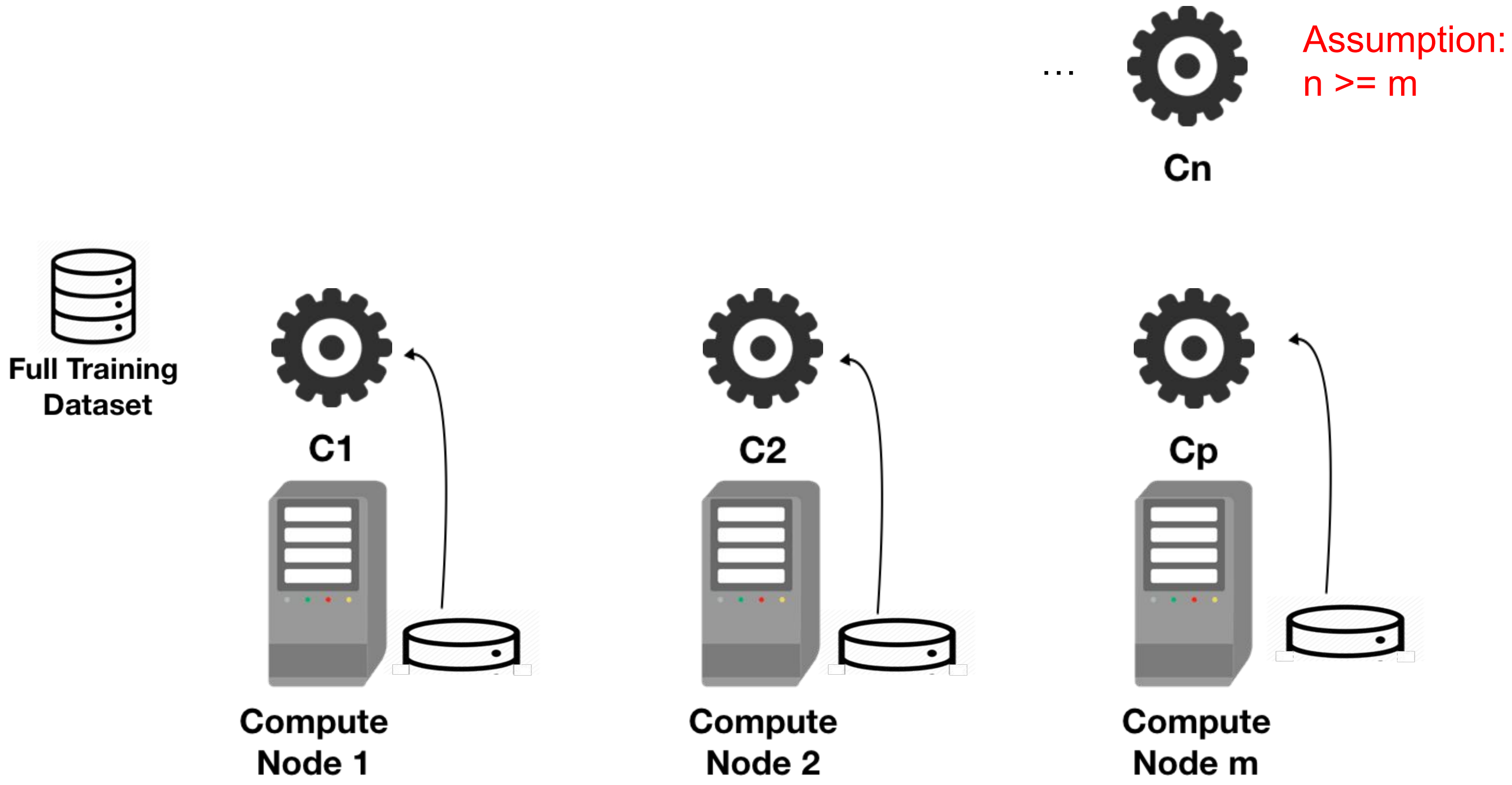
Assumption:
n >= m

Model Hopper Parallelism (MOP)

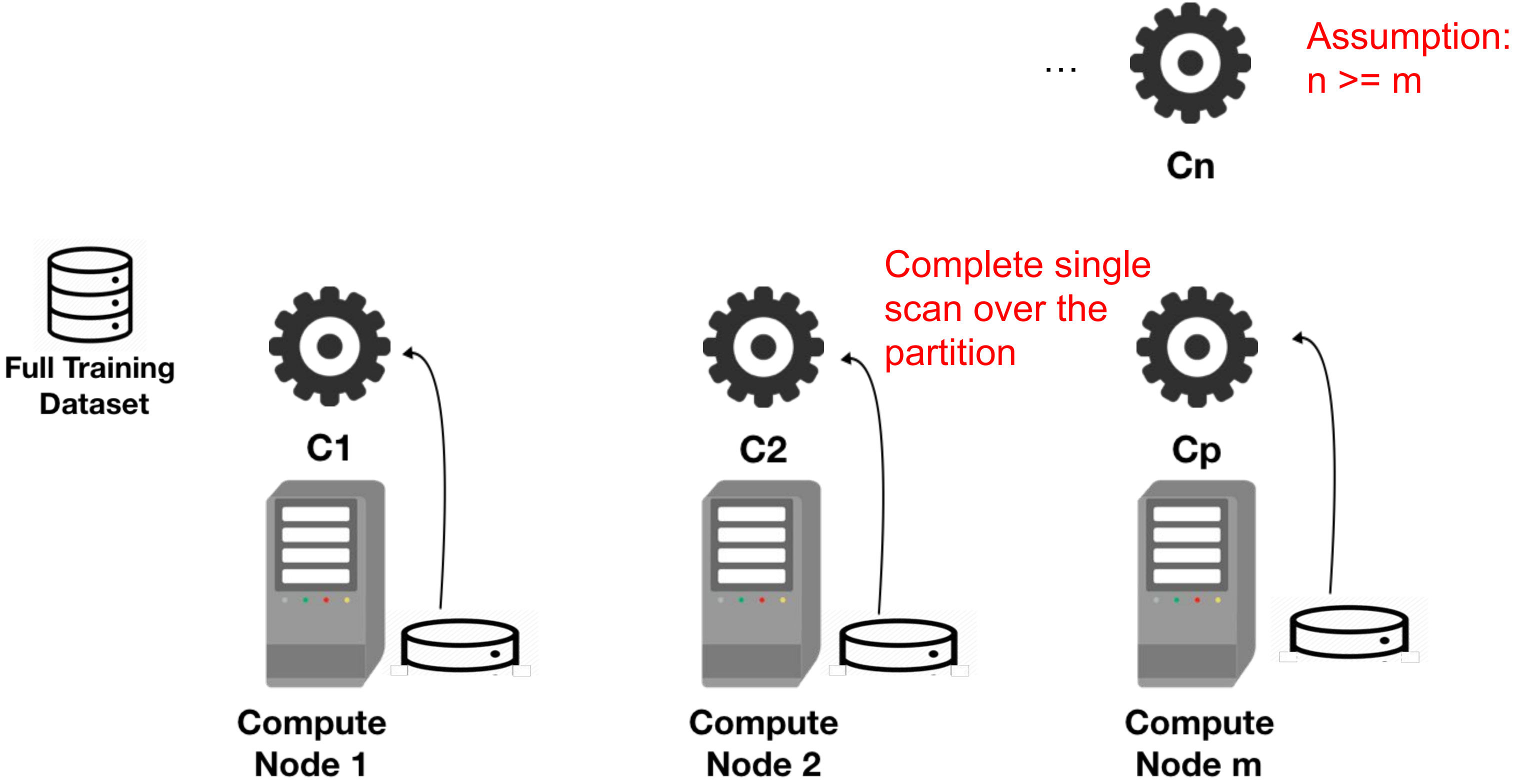


Assumption:
n >= m

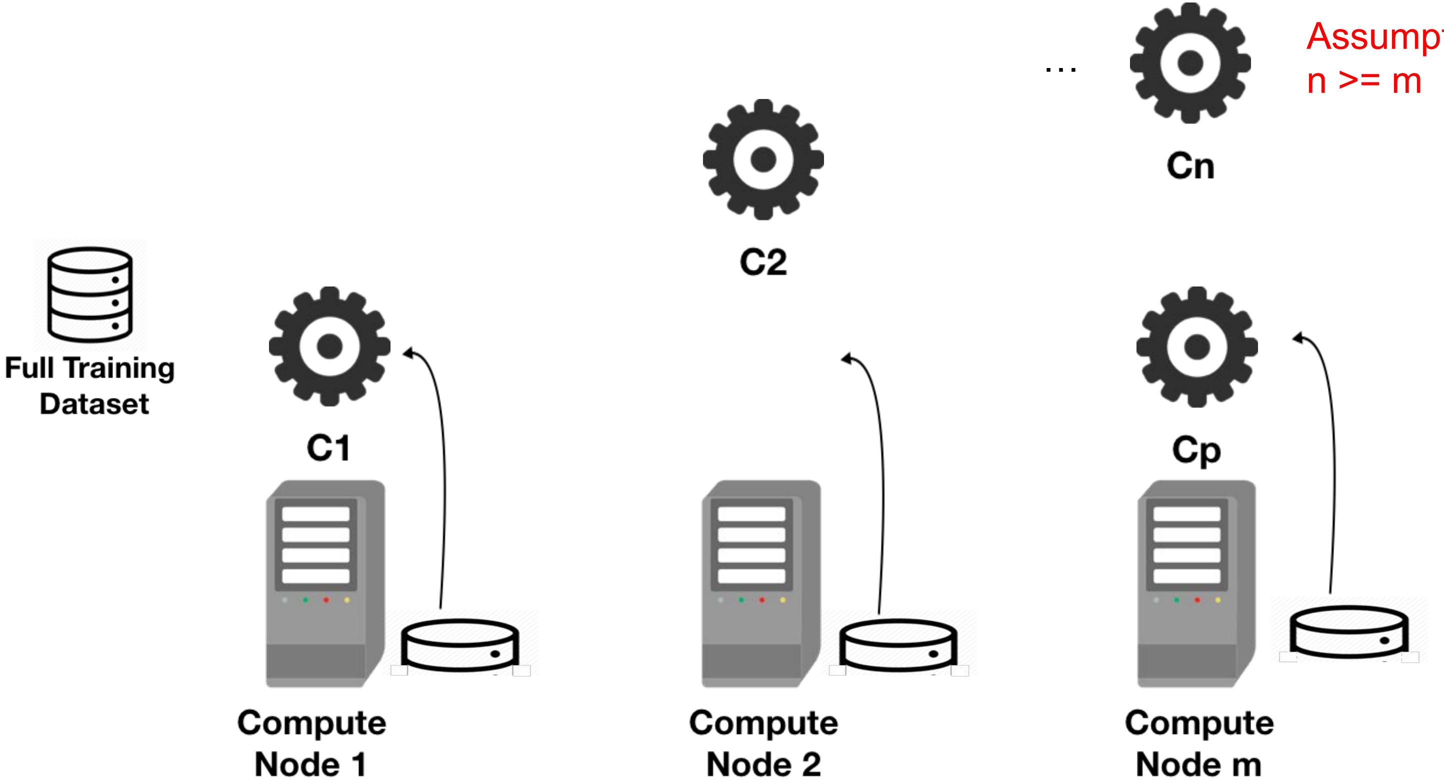
Model Hopper Parallelism (MOP)



Model Hopper Parallelism (MOP)

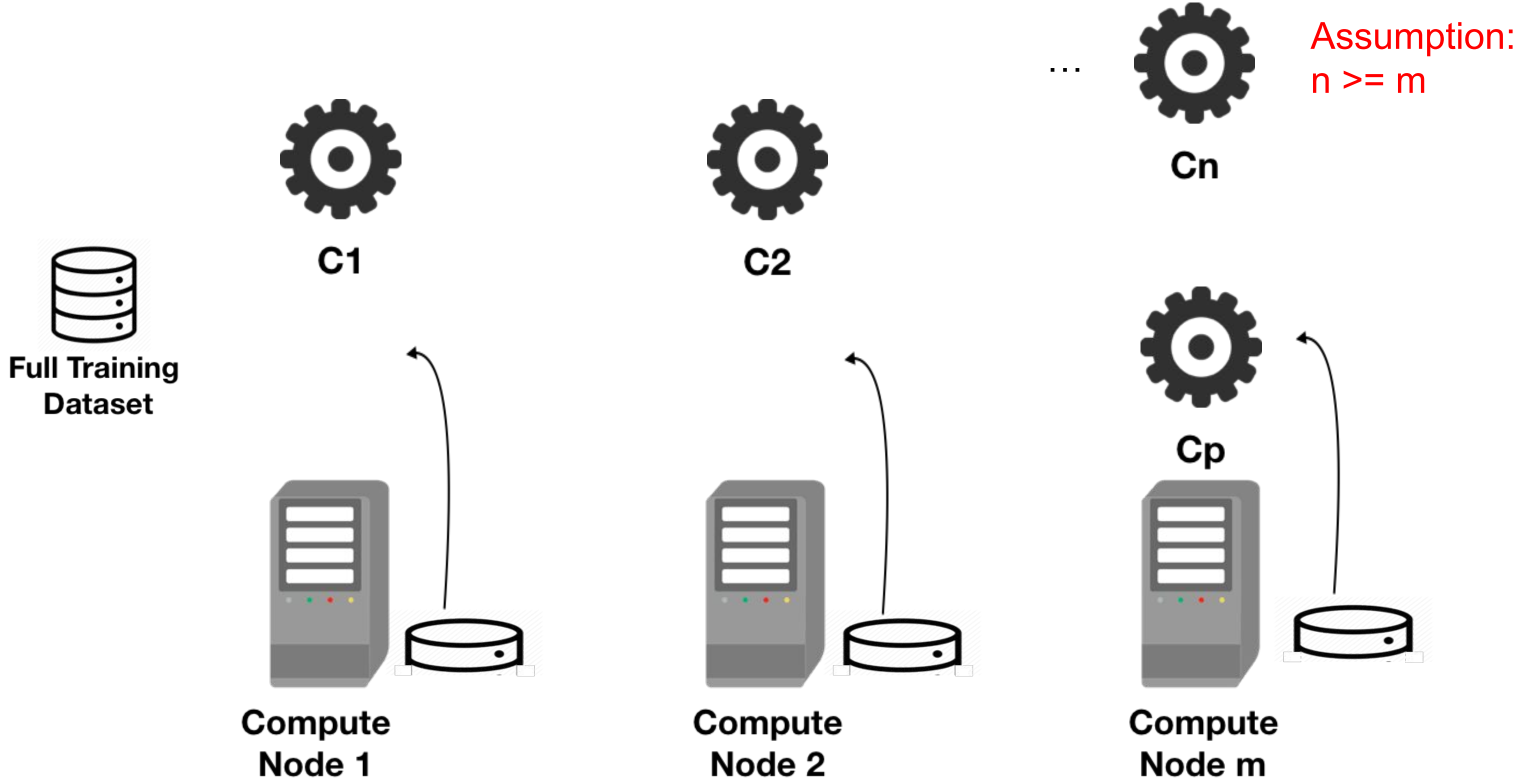


Model Hopper Parallelism (MOP)

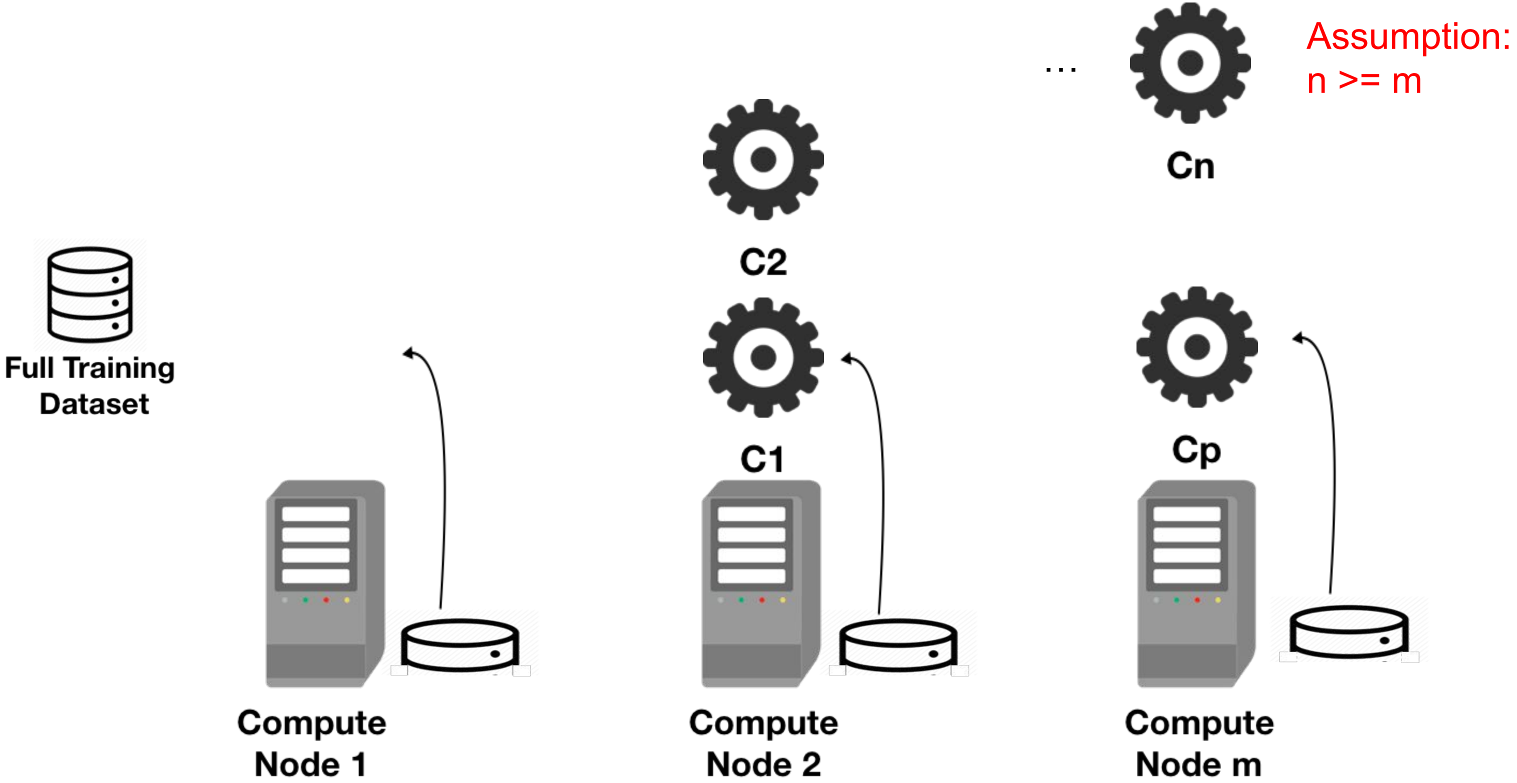


Assumption:
n >= m

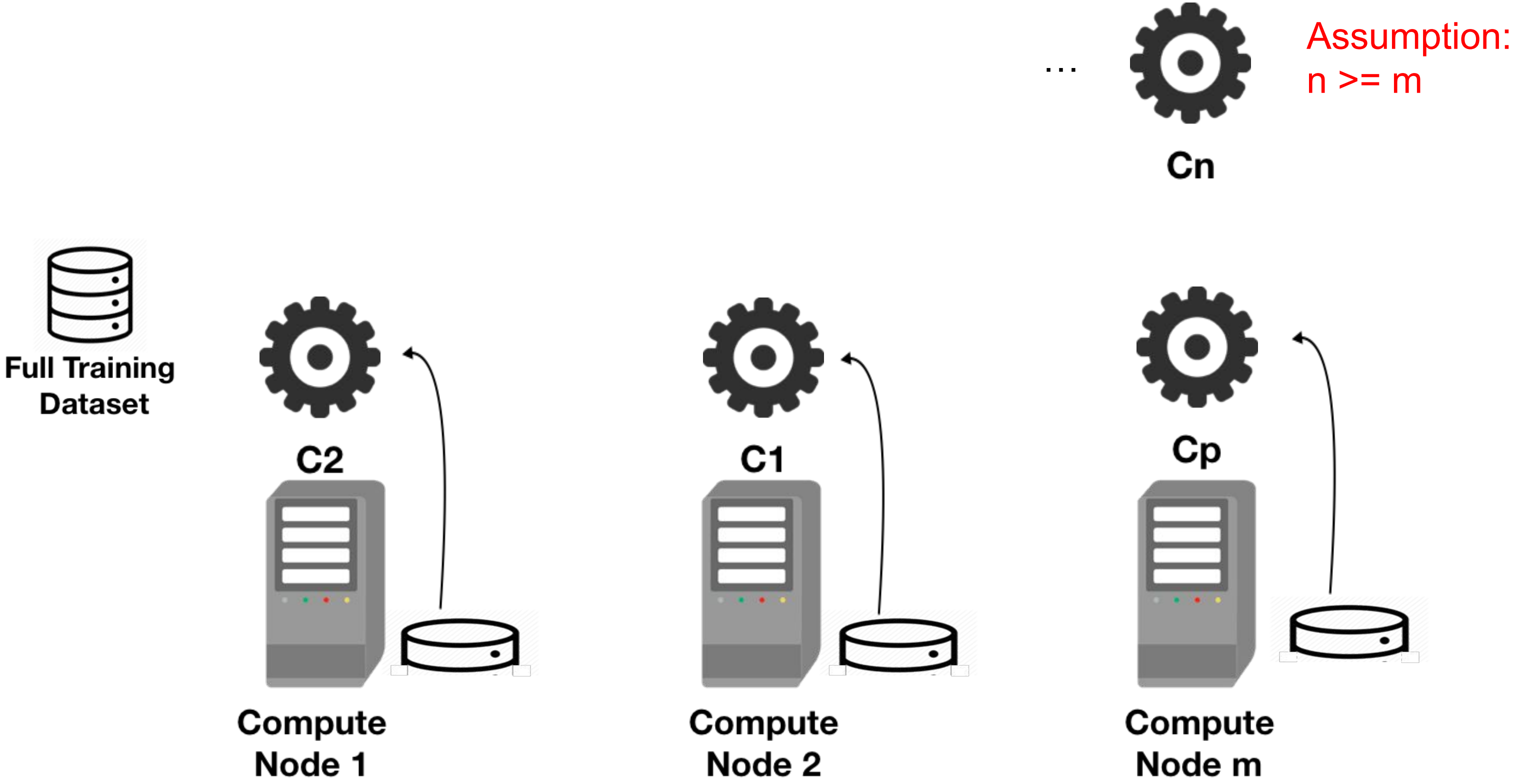
Model Hopper Parallelism (MOP)



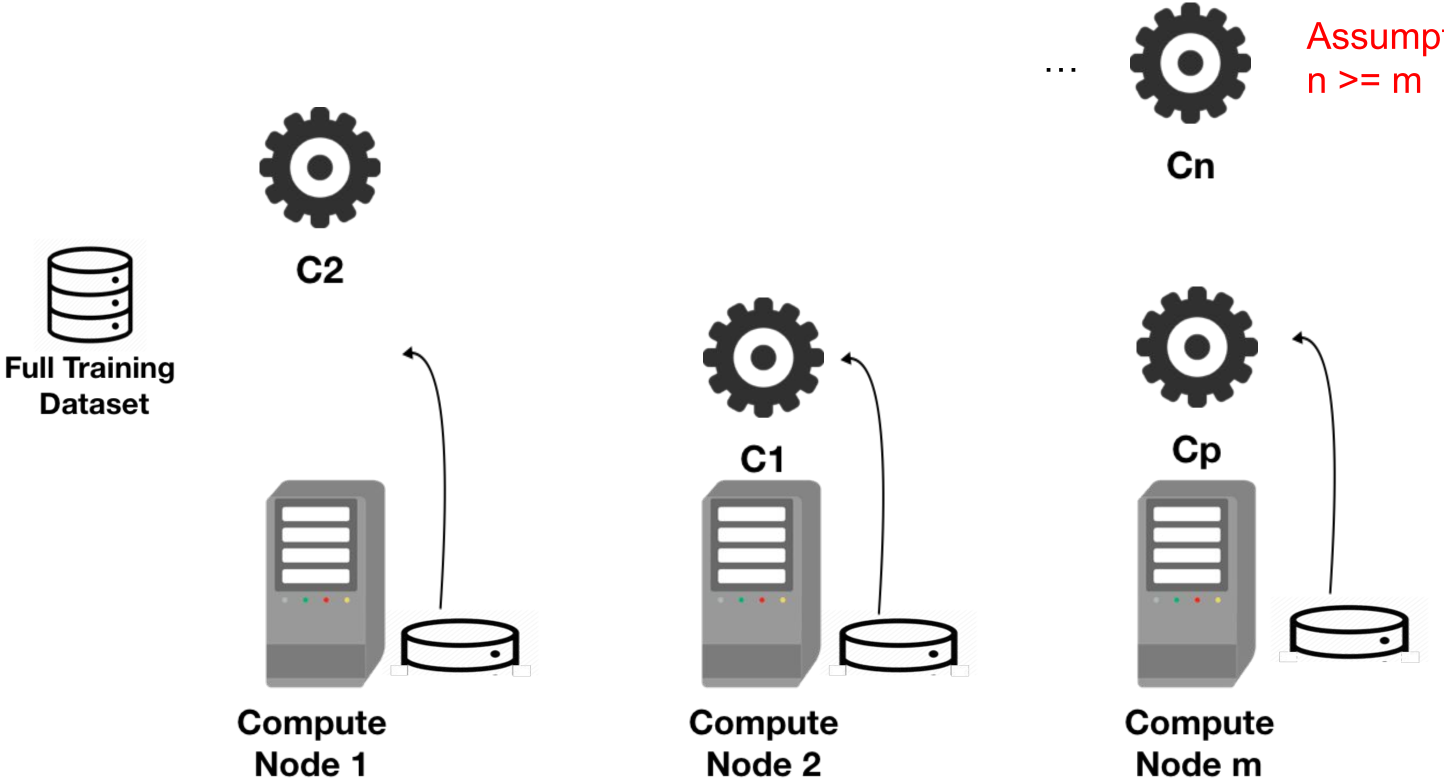
Model Hopper Parallelism (MOP)



Model Hopper Parallelism (MOP)

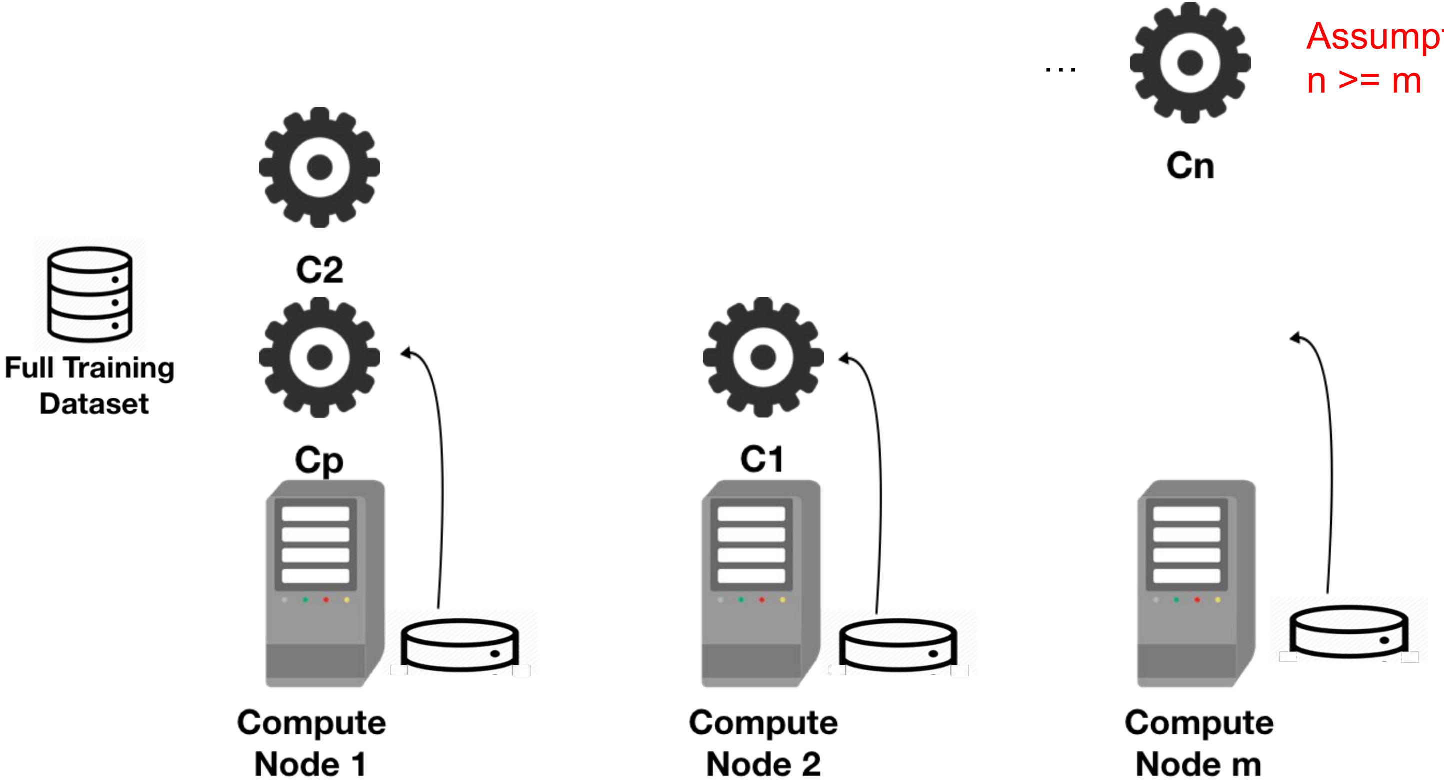


Model Hopper Parallelism (MOP)



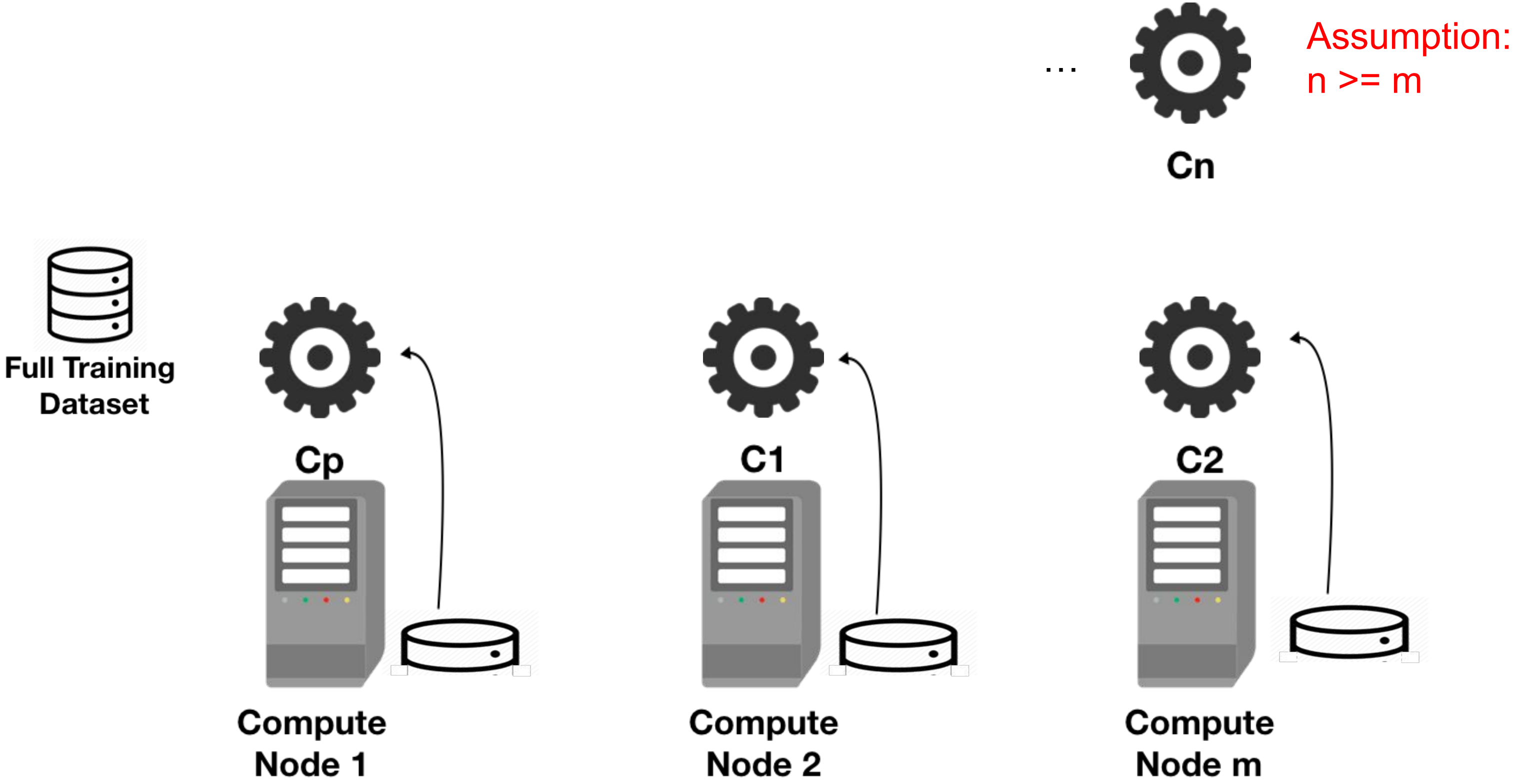
Assumption:
n >= m

Model Hopper Parallelism (MOP)

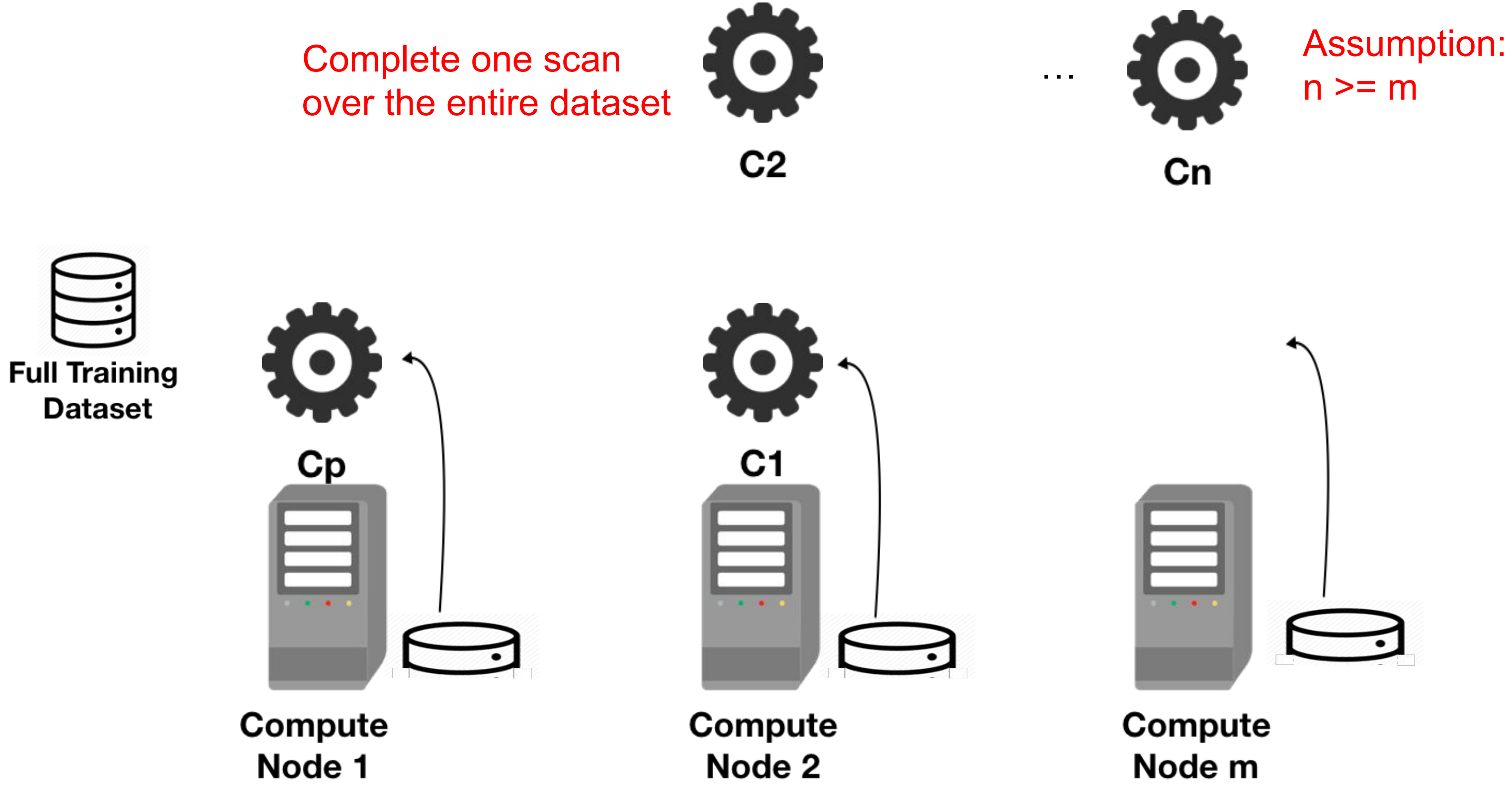


Assumption:
n >= m

Model Hopper Parallelism (MOP)



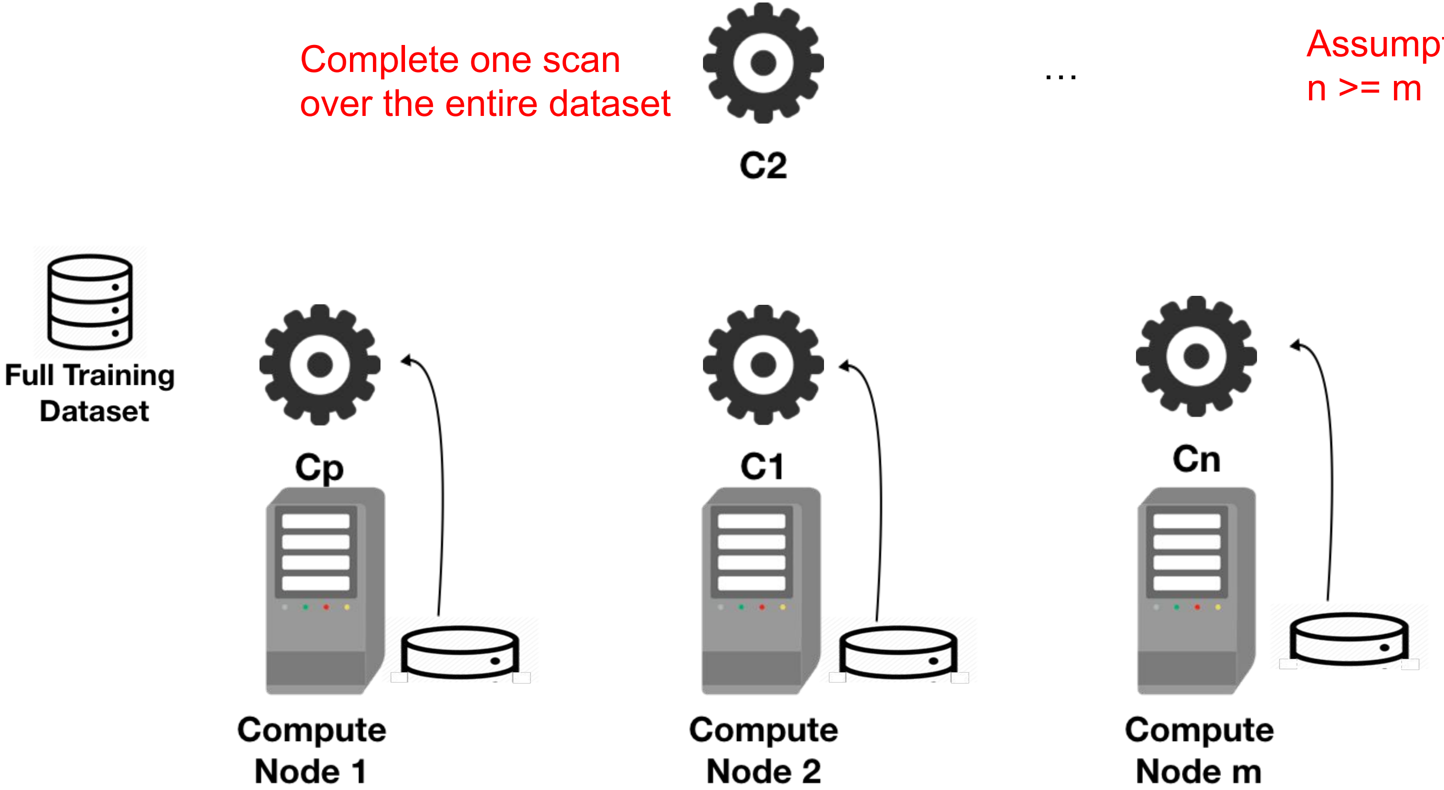
Model Hopper Parallelism (MOP)



Model Hopper Parallelism (MOP)

Complete one scan
over the entire dataset

Assumption:
n >= m



Model Hopper Parallelism (MOP)

MOP exploits the robustness of deep net training to the data visit order at partition level.

MOP is the most resource-efficient approach: over 10X storage/memory savings, minimum communication overheads.

Different configurations see the data in different yet sequential orders: best convergence efficiency, reproducible.

Detailed experimental evaluation results can be found in our VLDB 2020 paper.

Outline

1. Motivation
2. High-level (layered) Architecture
3. Execution Optimizations
4. Recent and Ongoing Research
    a. Feature Transfer and Transfer Learning
    b. Integration with Other Execution Backends
5. Summary

Feature Transfer and Transfer Learning

Goal: Enable feature transfer from pre-trained deep net models (e.g., BERT, GPT) for downstream analytics tasks.

Problem: Explore features from multiple layers before picking the best one. Wasted computations and storage/memory blowups!

Our Approach: Combine MOP with feature transfer-aware execution strategies that intelligently stages the computations.

Outline

1. Motivation
2. High-level (layered) Architecture
3. Execution Optimizations
4. Recent and Ongoing Research
    a. Feature Transfer and Transfer Learning
    b. Integration with Other Execution Backends

Greenplum MADlib APACHE Spark™

Ongoing work focuses on cloud native systems

Summary

Cerebro: A Layered Data Platform for Scalable Deep Learning.

At the core, Cerebro uses Model Hopper Parallelism, a novel hybrid of task- and data-parallelism, that exploits the properties of deep net training.

Ongoing research focuses on integration with other execution backends and supporting more deep learning workloads such as transfer learning.

Thank You!

Project Web Page: https://adalabucsd.github.io/cerebro.html

Outline

1. Motivation
2. High-level (layered) Architecture
3. Execution Optimizations
4. Recent and Ongoing Research
5. Summary

Outline

1. **Motivation**
2. High-level (layered) Architecture
3. Execution Optimizations
4. Recent and Ongoing Research
5. Summary

Outline

1. Motivation
2. High-level (layered) Architecture
3. Execution Optimizations
    a. Existing Approaches
    b. Our Solution: Model Hopper Parallelism (MOP)
    c. Experimental Results
4. Recent and Ongoing Research
5. Summary

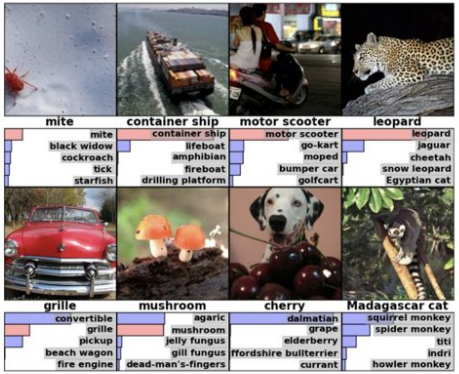
Experimental Workload

Dataset	ImageNet (250 GB)
Cluster	8 Node Cluster. P100 GPU, 192 GB RAM, 32 Cores, 10 Gbps Network
Model Architectures	VGG16, ResNet50
Learning Rates	0.0001, 0.00001
L2 Reg. Coefficient	0.0001, 0.00001
Batch Sizes	32, 256

ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



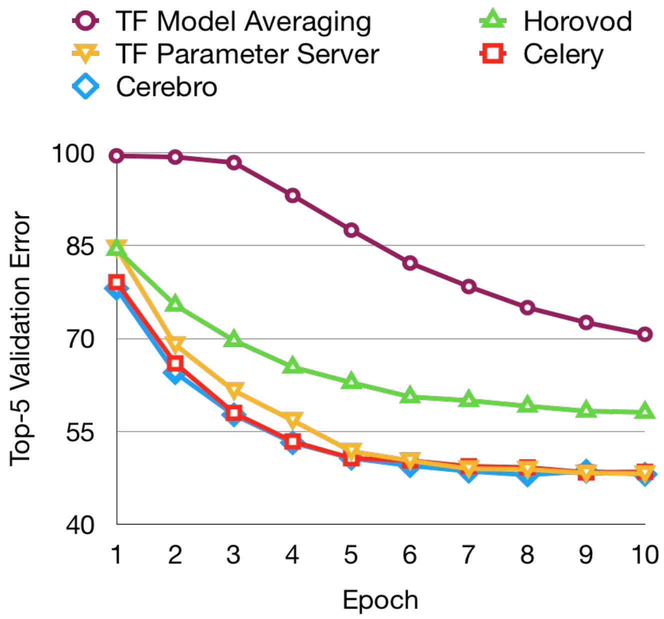
16 configurations trained for 10 epochs

Experimental Results

Runtime Efficiency

System	Runtime (hrs)	Storage Footprint (GB)
TF Parameter Server (Data Parallel)	190.0	250
Horovod (Data Parallel)	54.2	250
TF Model Averaging (Data Parallel)	19.70	250
Celery (Task Parallel)	17.2	2000
Cerebro (MOP)	17.7	250

Convergence Efficiency



More results including different datasets and drill-down experiments can be found in our VLDB 2020 paper.

Integration with Other Execution Backends

Goal: Integrate with DB/Dataflow/Cloud Native systems for easy adoption and for exploiting the auxiliary capabilities of those systems.

Problem: How can we emulate MOP on these systems with no or very little changes to those systems?

Our Approach: Explore the efficiency tradeoffs of alternatives for emulating MOP.

UDF-based Approach

Data Export-based Approach

?