Resource-Efficient Execution of Deep Learning Computations

Deepak Narayanan
Microsoft Research

Joint work with many awesome collaborators at
Stanford, Microsoft Research, and NVIDIA

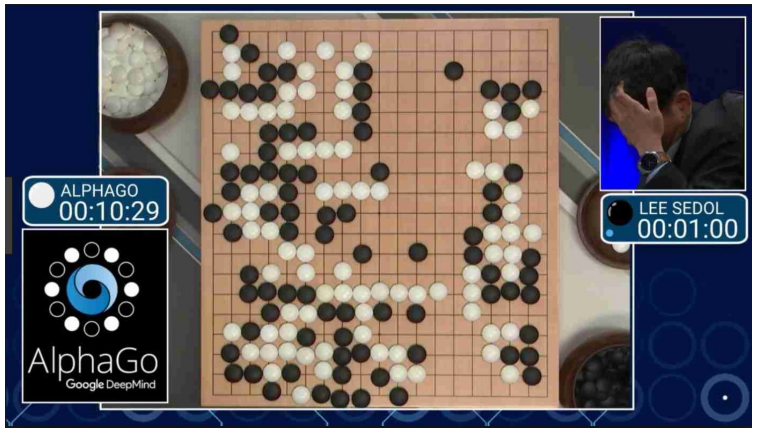
Deep Learning is powering new applications

வணக்கம் என் பெயர் தீபக்

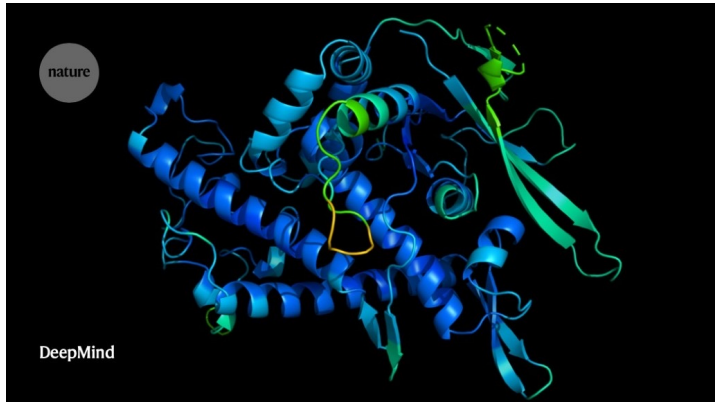
Hello, my name is Deepak

Machine Translation

Speech-to-Text



Game Playing



Drug Discovery

…but extremely compute intensive!

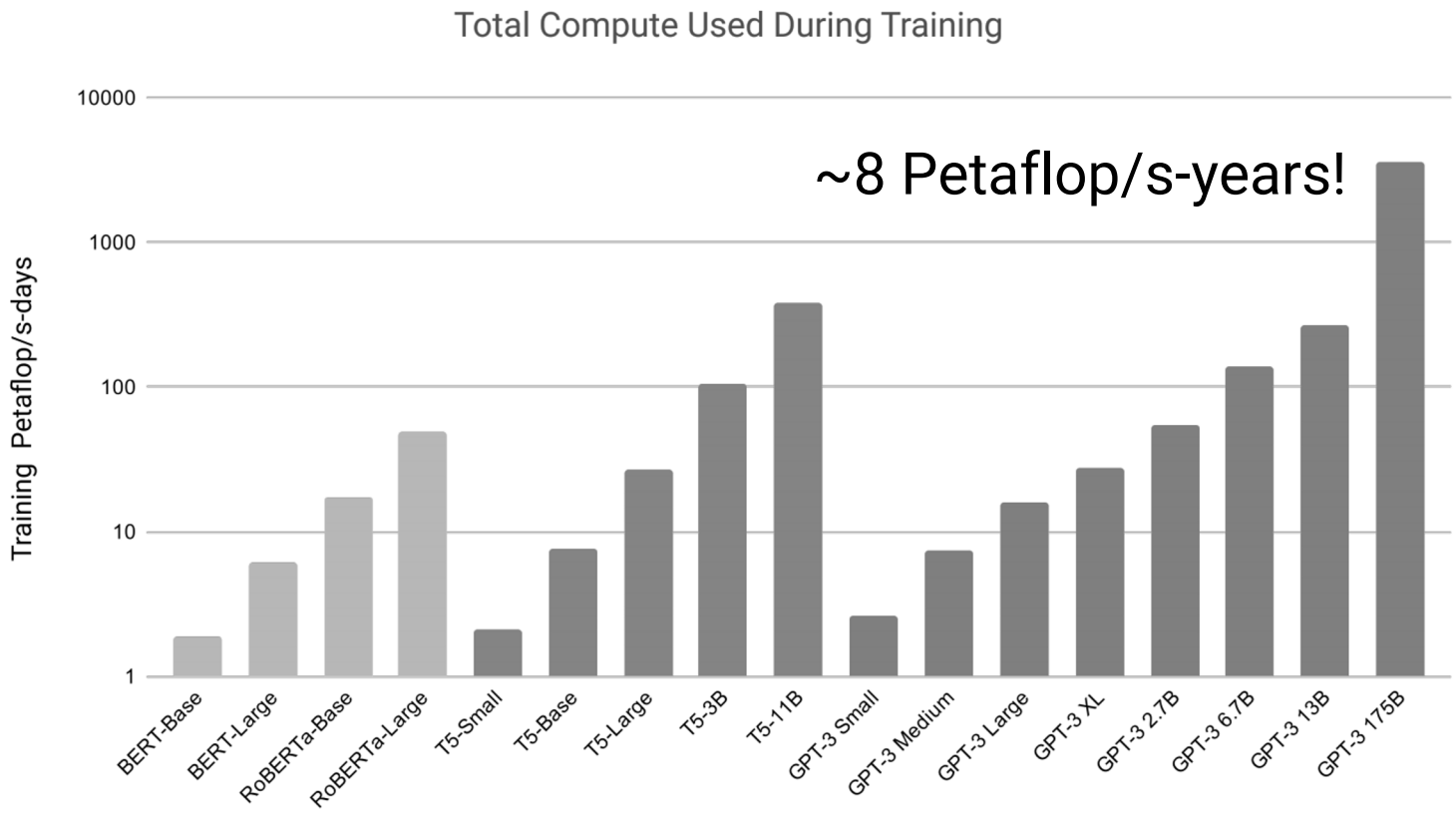
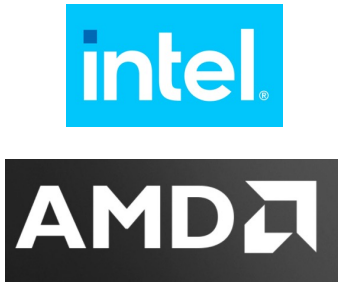


Figure from “Language Models are Few-Shot Learners”, Brown et al.

New parallel hardware is helping fulfill this demand

GPUs

TPUs

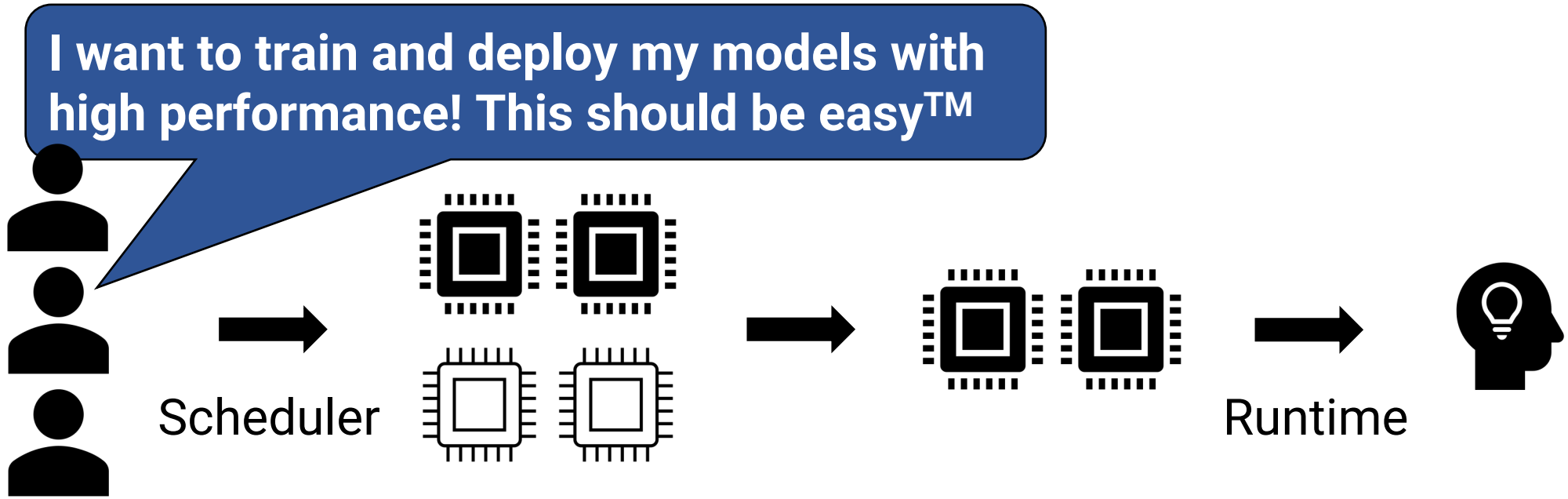


Multicore CPUs

Domain-specific accelerators

Heterogeneity becoming pervasive!

Heterogeneity complicates the DL workflow



Iterative, long-running, and compute-intensive

Heterogeneity complicates the DL workflow

Not easy for users :(

What resource type to use?
➤ Maybe just the fastest one!

But now stuck in queues waiting for resources all the time :(

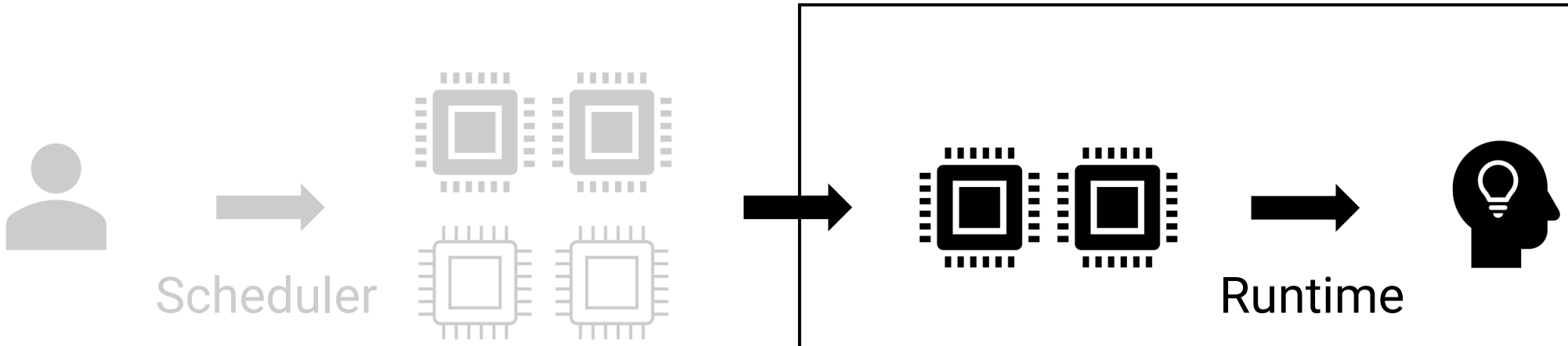
n resources?
➤ Maybe parallelize with a copy of the model on each worker!

But training bottlenecked by communication for certain models and resource types :(

This talk:

Careful automated scheduling of computation on (heterogeneous) resources across the software stack can significantly increase model training throughput

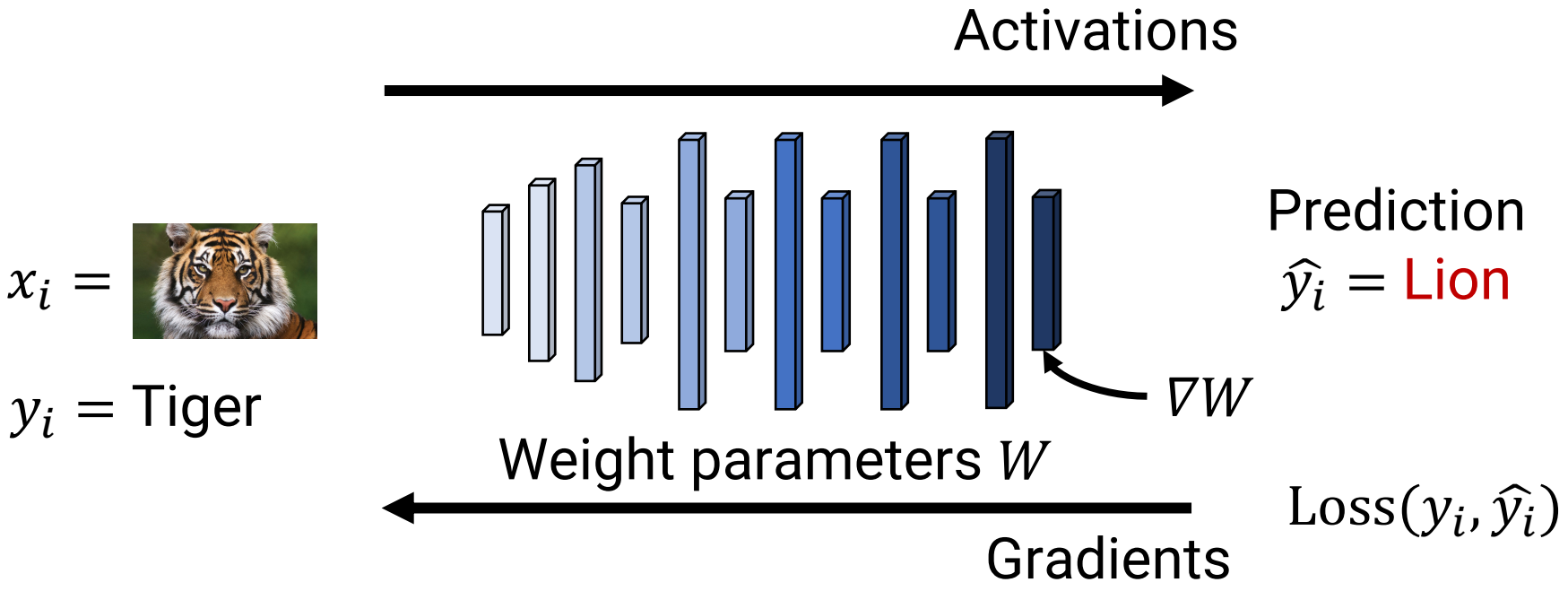
DL model training workflow



Scheduling
(OSDI 2020)

Distributed Training
(SOSP 2019, ICML 2021, SC 2021)

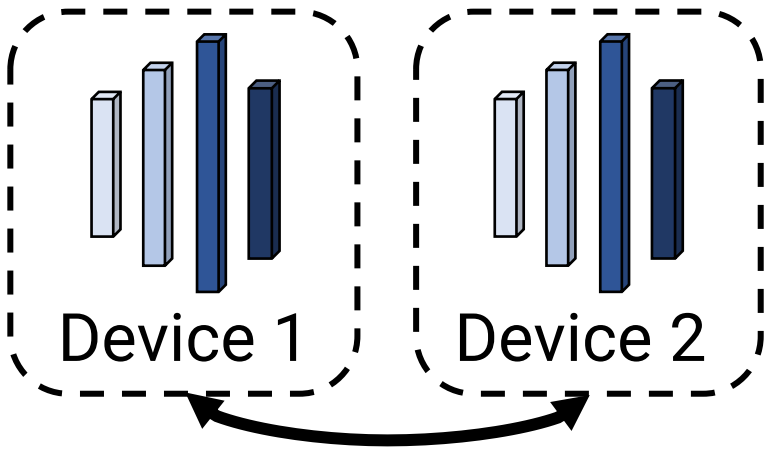
Background: DNN training



Optimization performed in iterations; each iteration can be parallelized within an accelerator (GPU) and also across accelerators

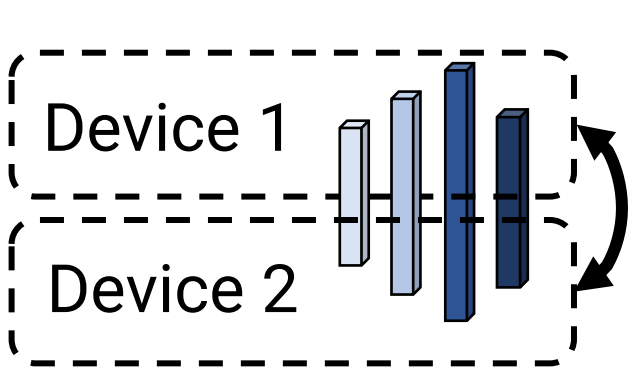
Parallel model training: existing approaches

Data Parallelism (DP)

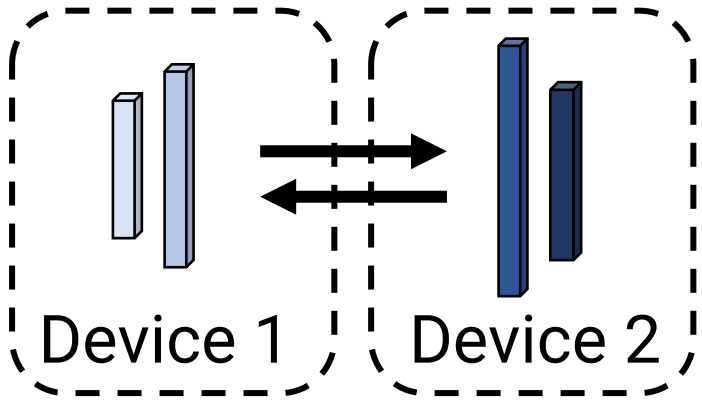


n copies of model parameters

Model Parallelism (MP)



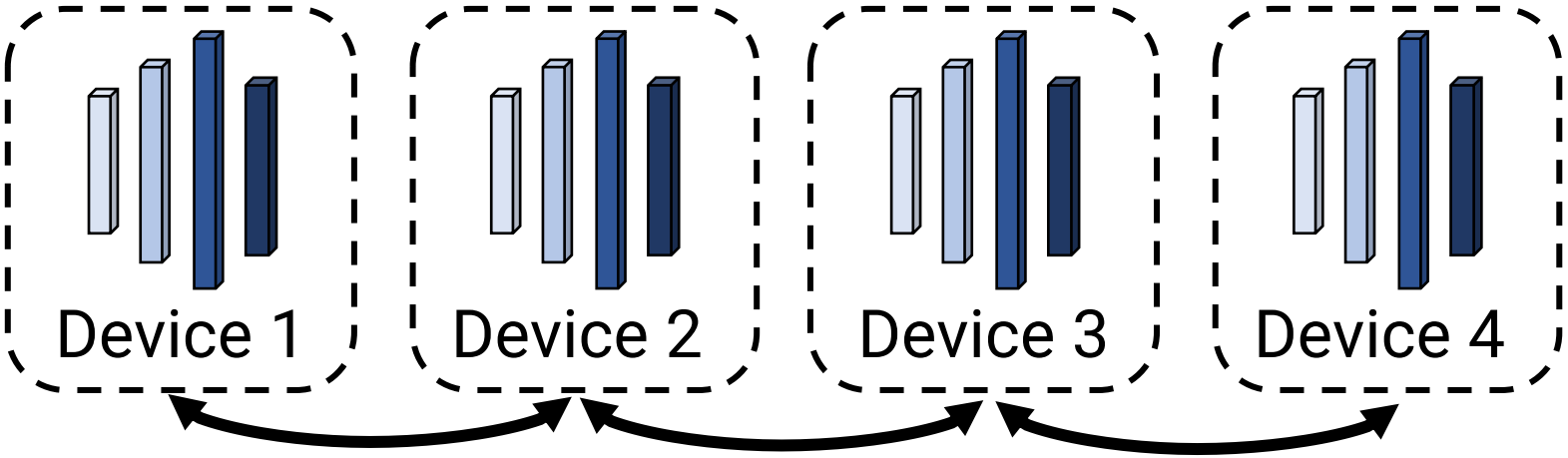
Tensor MP



Pipeline MP

Single copy of model parameters

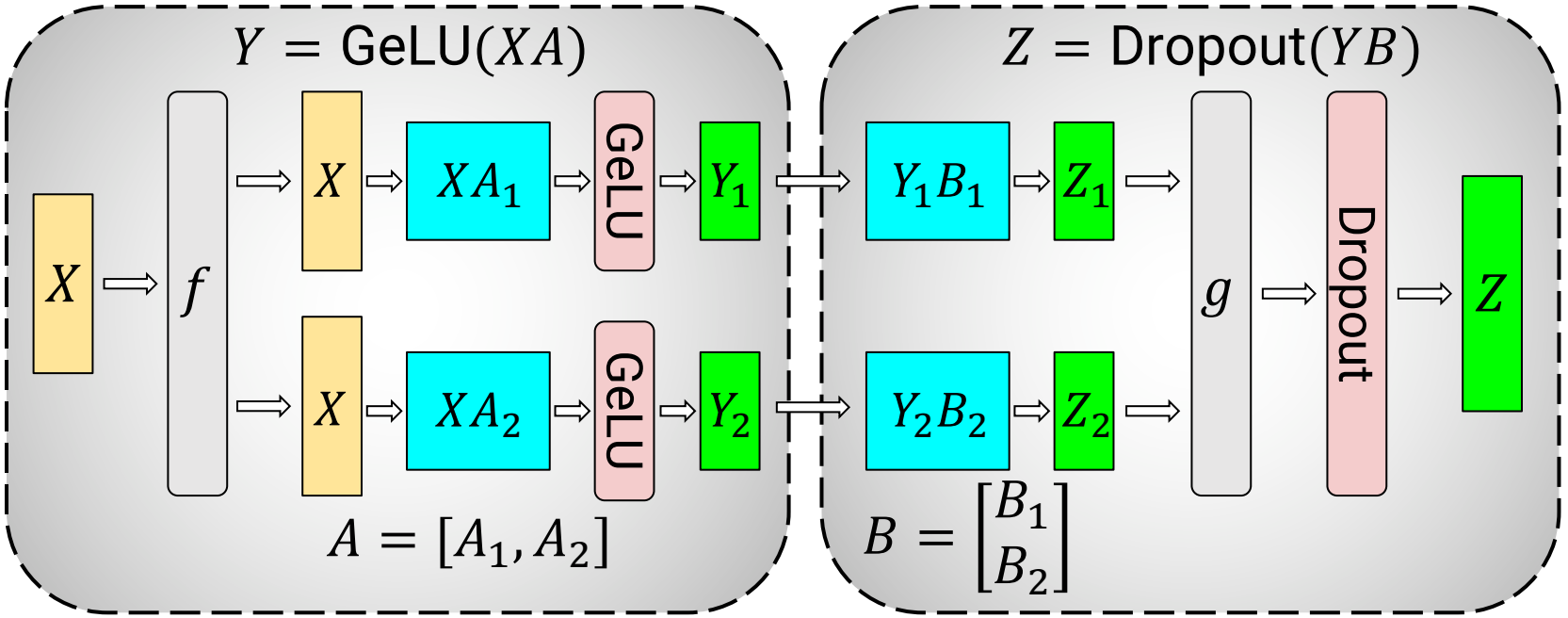
Data parallelism



- All-reductions of weight gradients after every iteration
- Cannot be used in isolation for large models, but can be used on smaller model shards

Tensor model parallelism

Each layer of model is partitioned over multiple devices



$g \rightarrow$ All-reduction $(Y_1B_1 + Y_2B_2)$ in forward pass

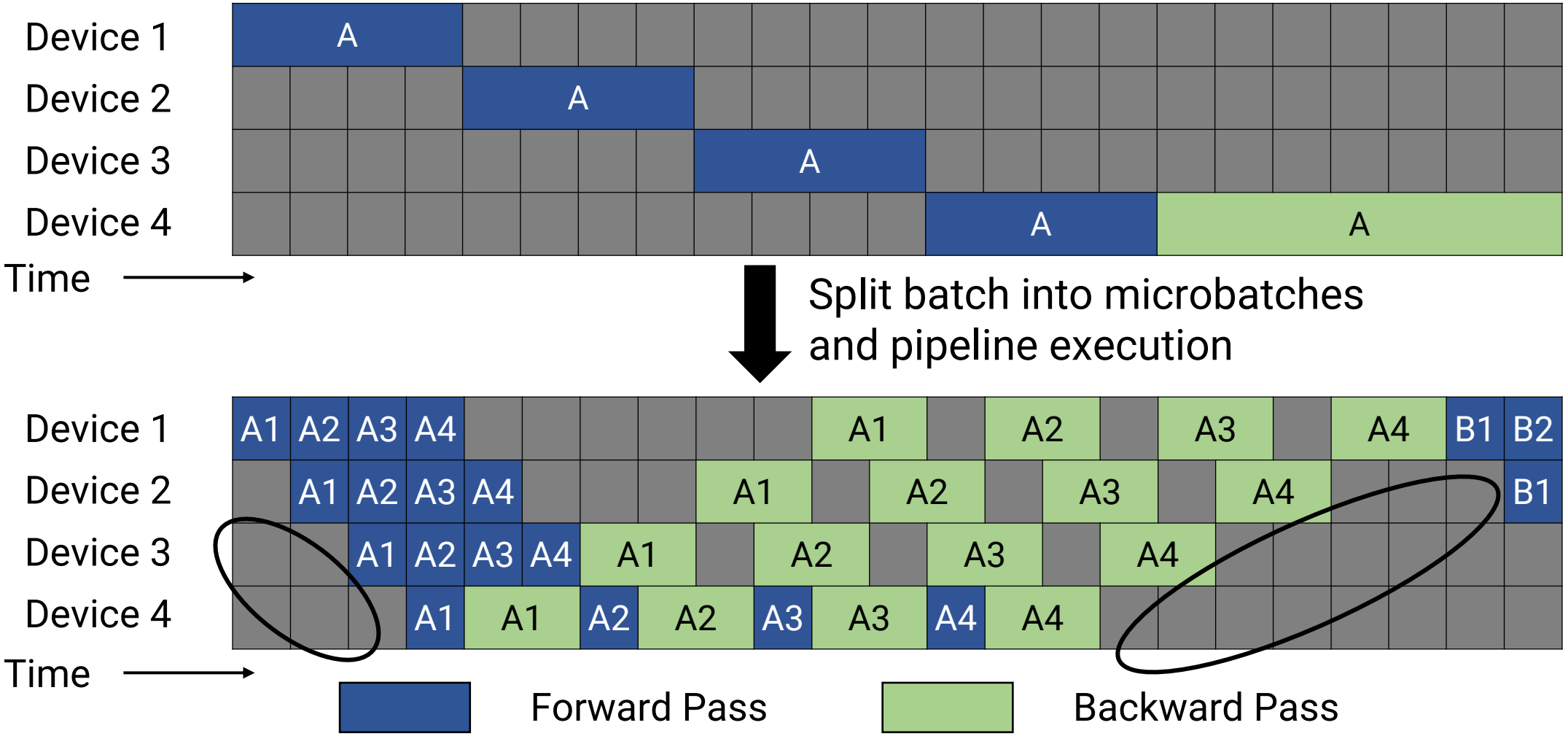
Slow across inter-server communication links

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, arXiv:1909.08053, Shoeybi et al.

Pipeline model parallelism

- Layers in model sharded over devices (i.e., each device is responsible for a subset of layers in the model)
- Each batch split into smaller microbatches and execution pipelined across these microbatches

Pipeline model parallelism



Goal: Heterogeneity-aware distributed training

Models + Hardware Deployments

Example 1: Model with large number of weight parameters

Example 2: Hardware deployment with slow interconnects

Tradeoffs when combining tensor and pipeline MP

Assume that total number of GPUs is n, tensor-model-parallel size is t, pipeline-model-parallel size is p $(t \cdot p = n)$

- Pipeline bubble size (fraction of ideal time spent idle) is then:

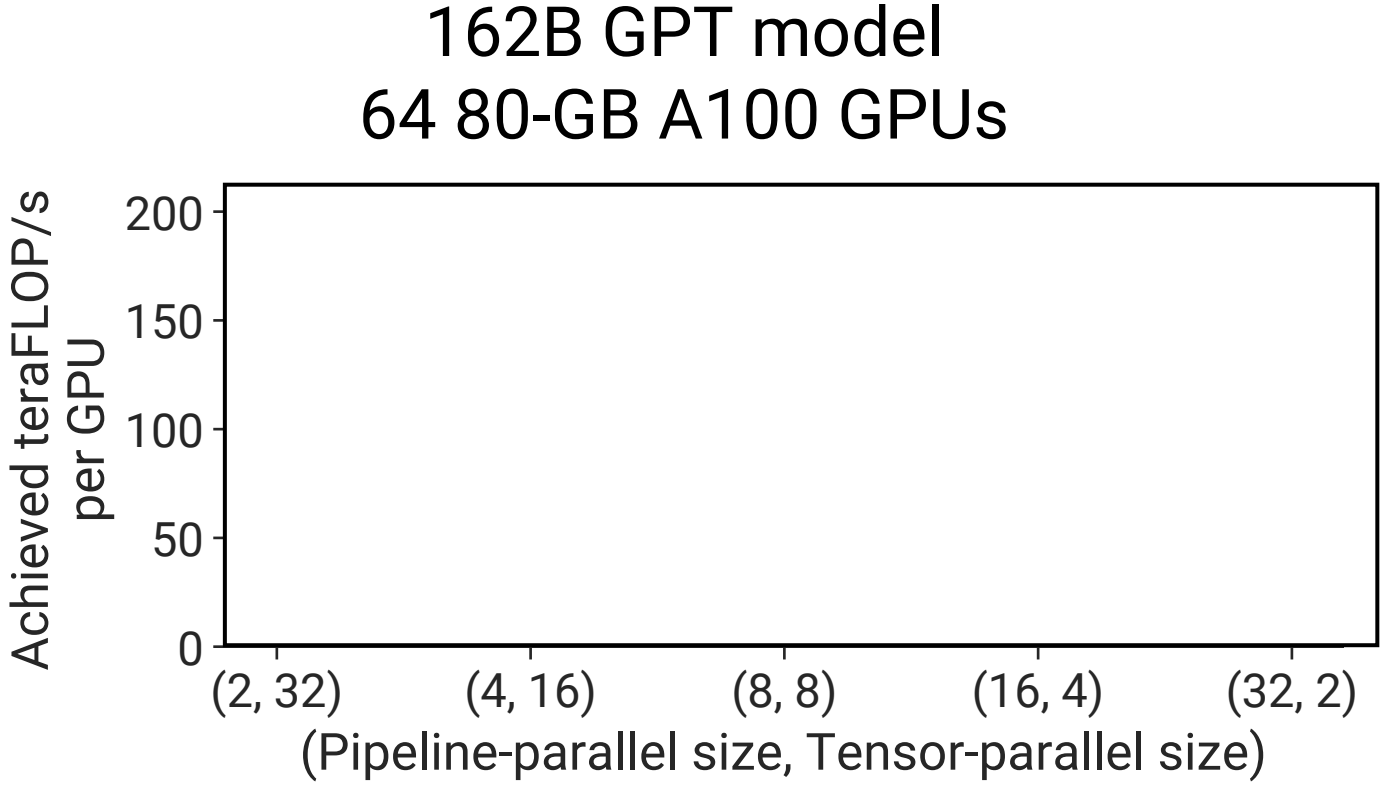
$$\frac{(p-1)\cdot\left(t_f + t_b\right)}{m\cdot\left(t_f + t_b\right)}$$

Number of microbatches in batch ⟵ m

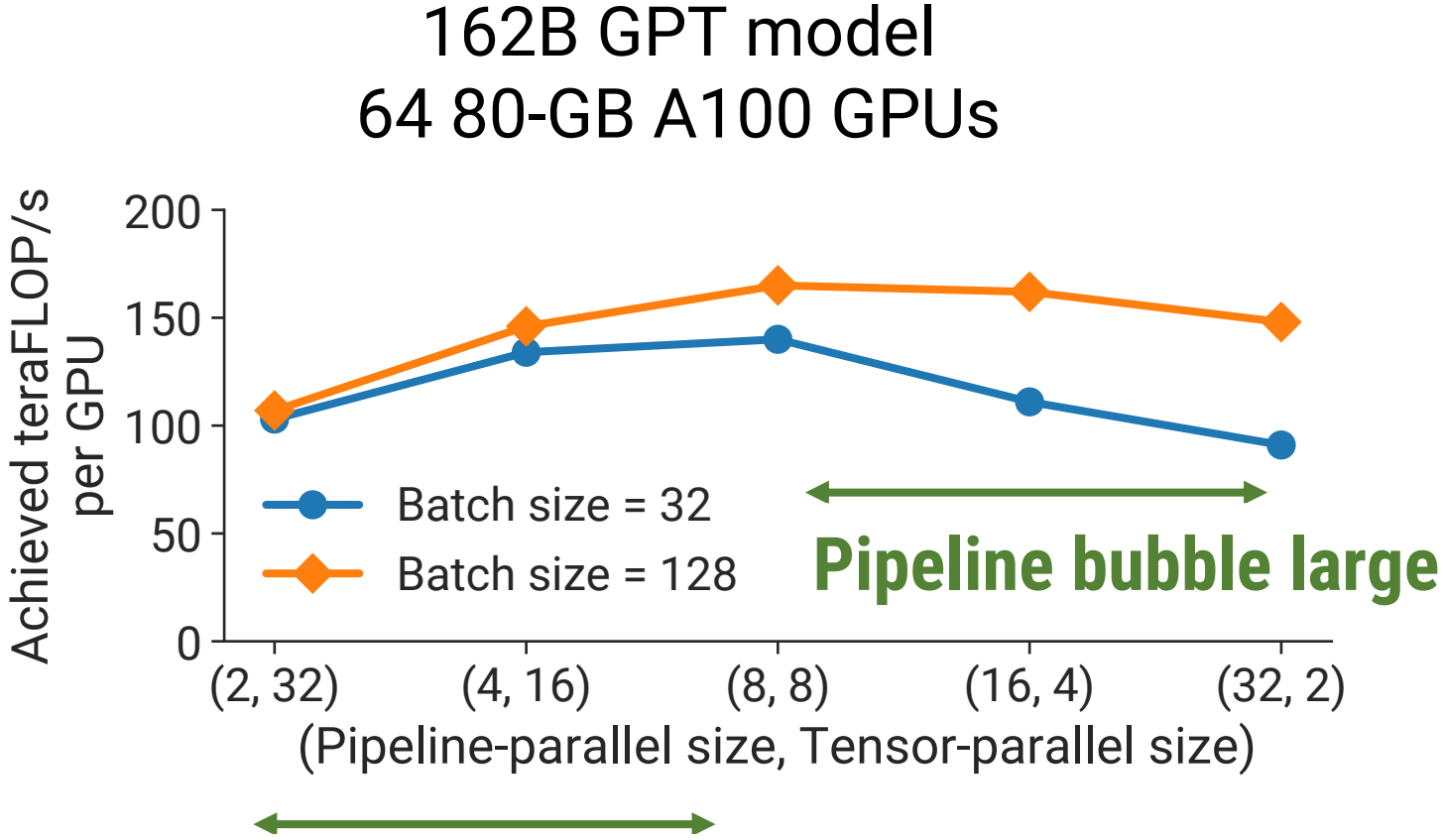
As t increases, pipeline bubble size decreases

- However, as t increases beyond the number of GPUs in a server, all-reduce communication is now cross-server and more expensive

Tradeoffs between tensor and pipeline MP

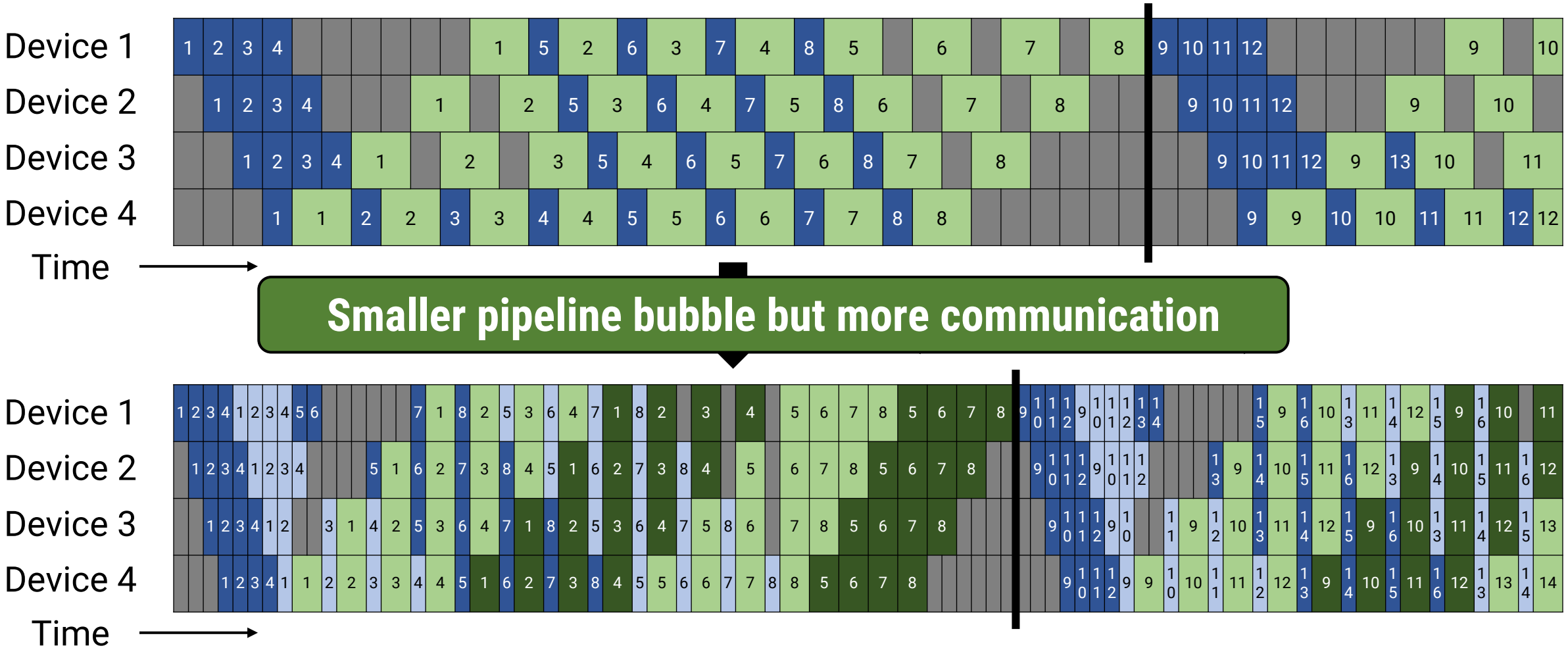


Tradeoffs between tensor and pipeline MP

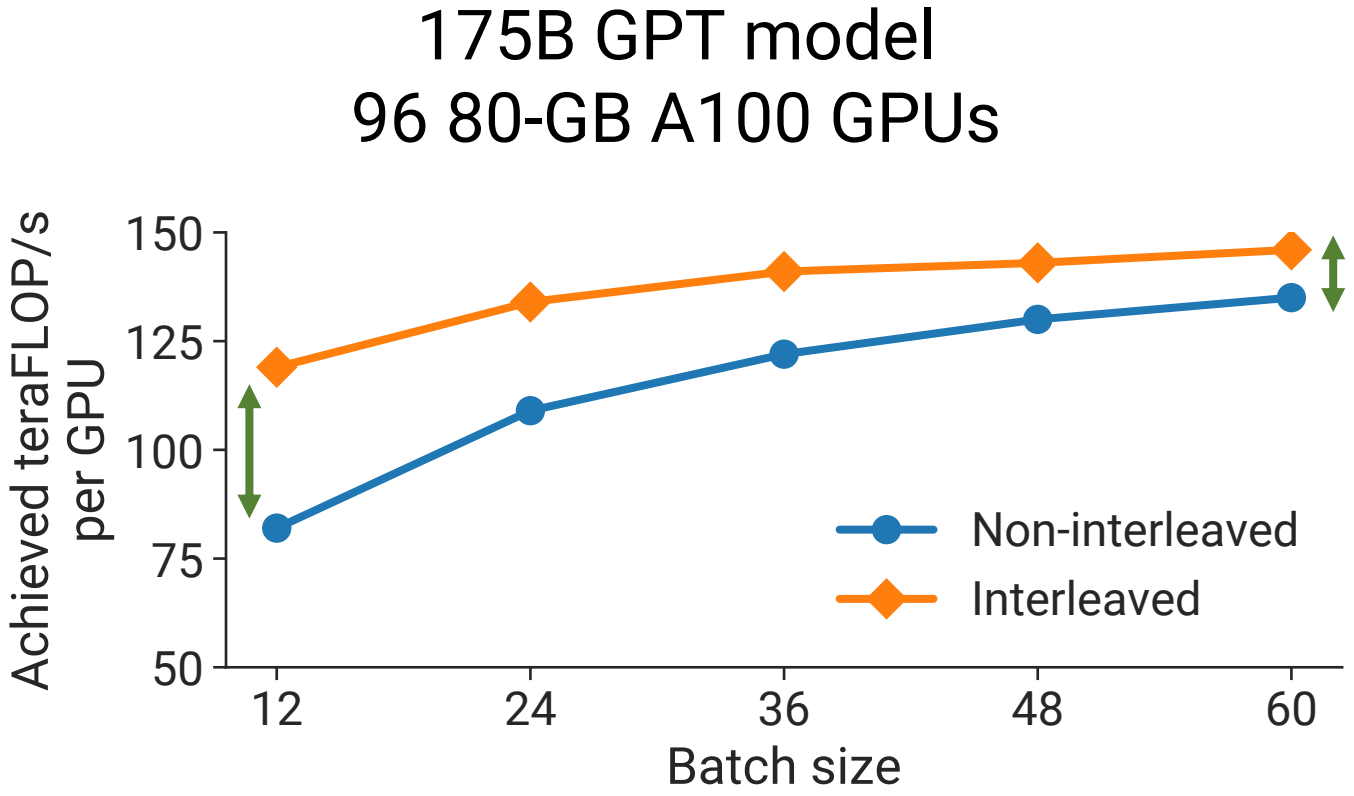


Yet more tradeoffs when also considering data parallelism!

New pipeline schedules affect throughput too!



New pipeline schedules affect throughput too!



Large throughput increases at small batch sizes, smaller at large batch sizes

How do we navigate this configuration space?

Degree of pipeline, tensor, and data parallelism

Pipelining schedule

Global batch size

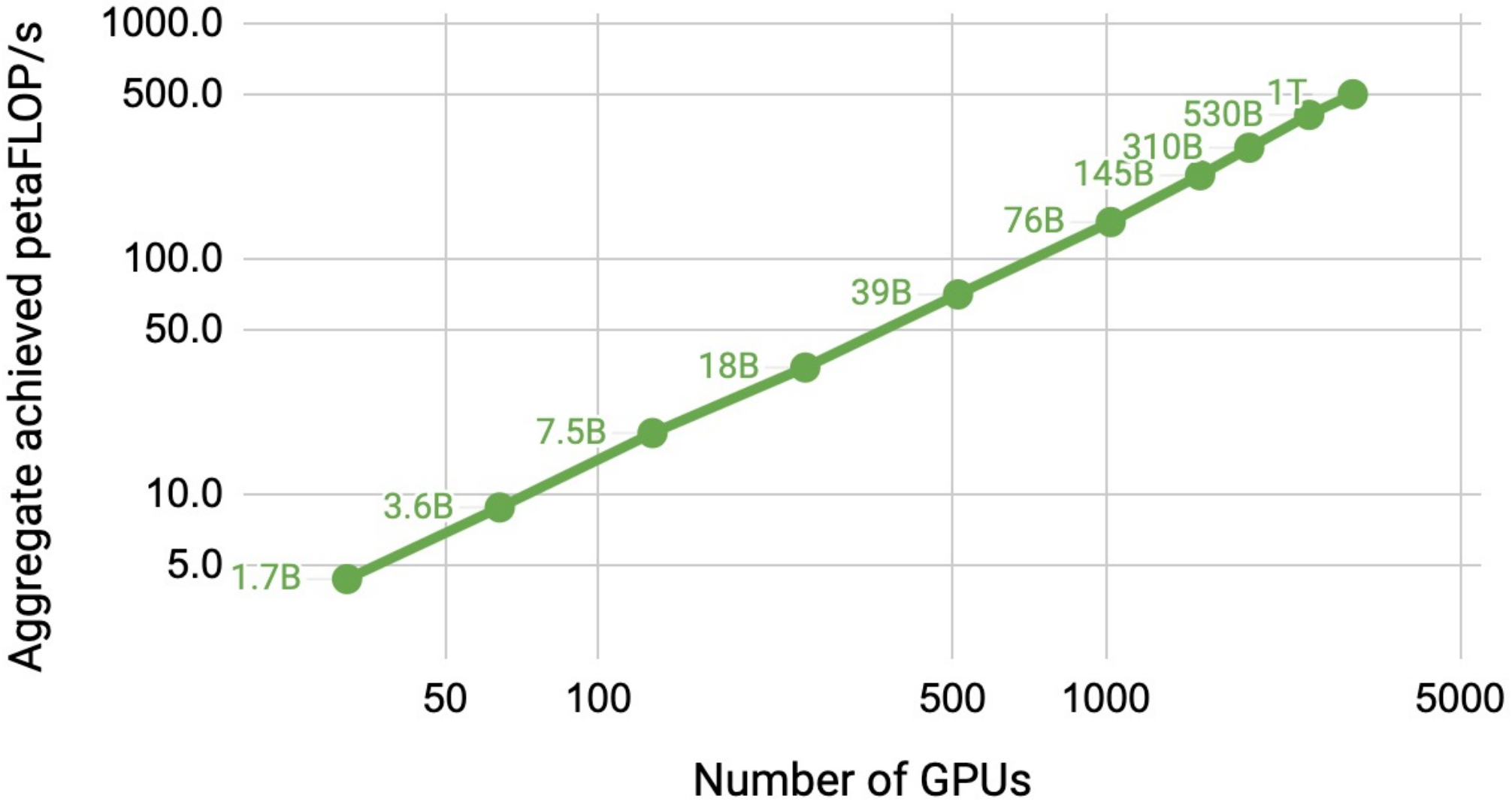
Microbatch size

Each of these influence amount of communication, size of pipeline bubble, memory footprint

More details in paper!

Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, SC 2021, Narayanan et al.

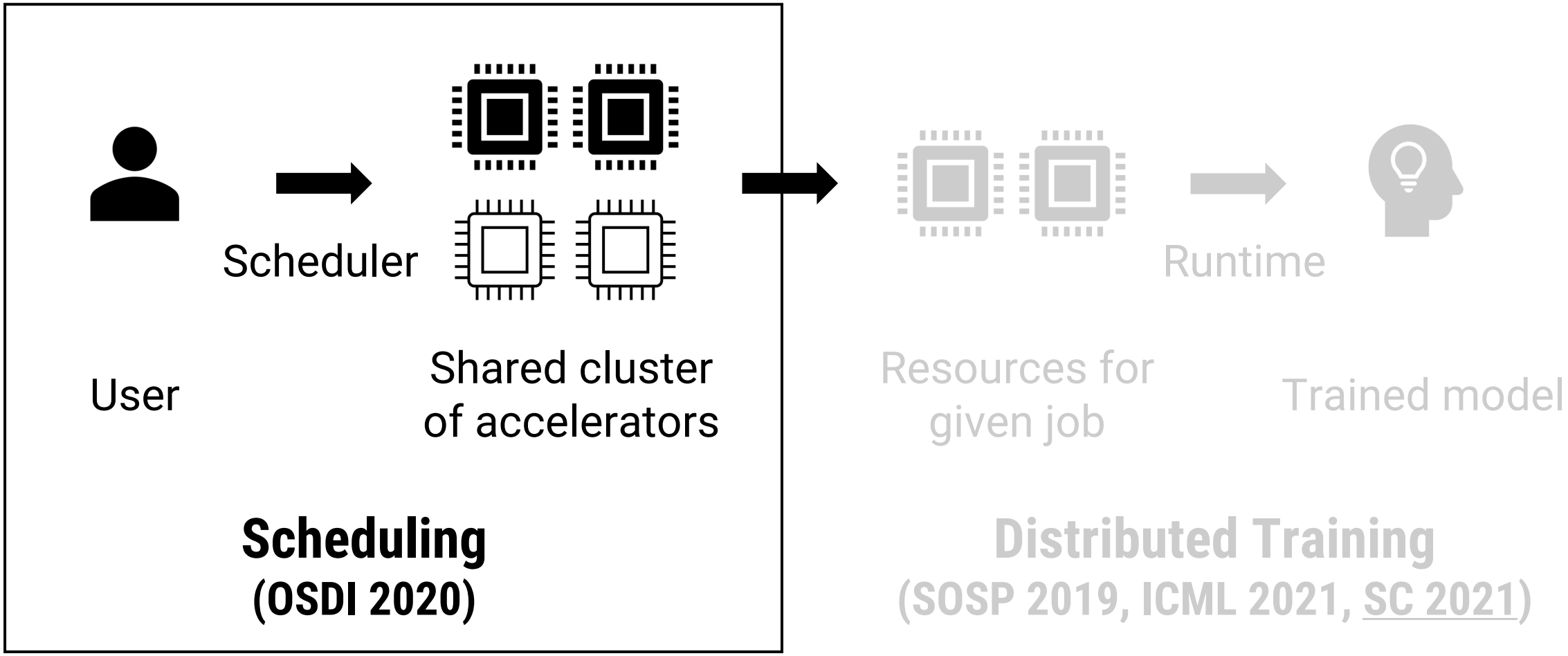
Strong scaling performance to 1000s of GPUs



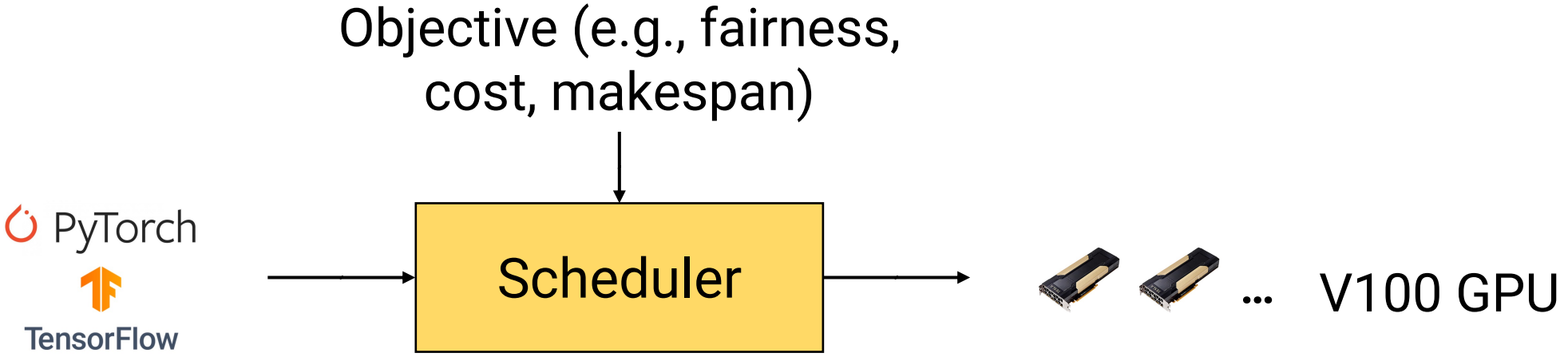
Takeaways

- One size cannot fit all models and all hardware deployments
- Existing parallelization strategies have different tradeoffs; these need to be considered carefully to obtain good performance
- End result: More cost-effective distributed training!

DL model training workflow



How should we allocate resources?



Training jobs in existing frameworks

Homogeneous cluster

Scheduling is a well-studied problem in computer systems in other contexts as well (e.g., big data clusters)

Heterogeneity is common in both public and private clusters!

Public clouds

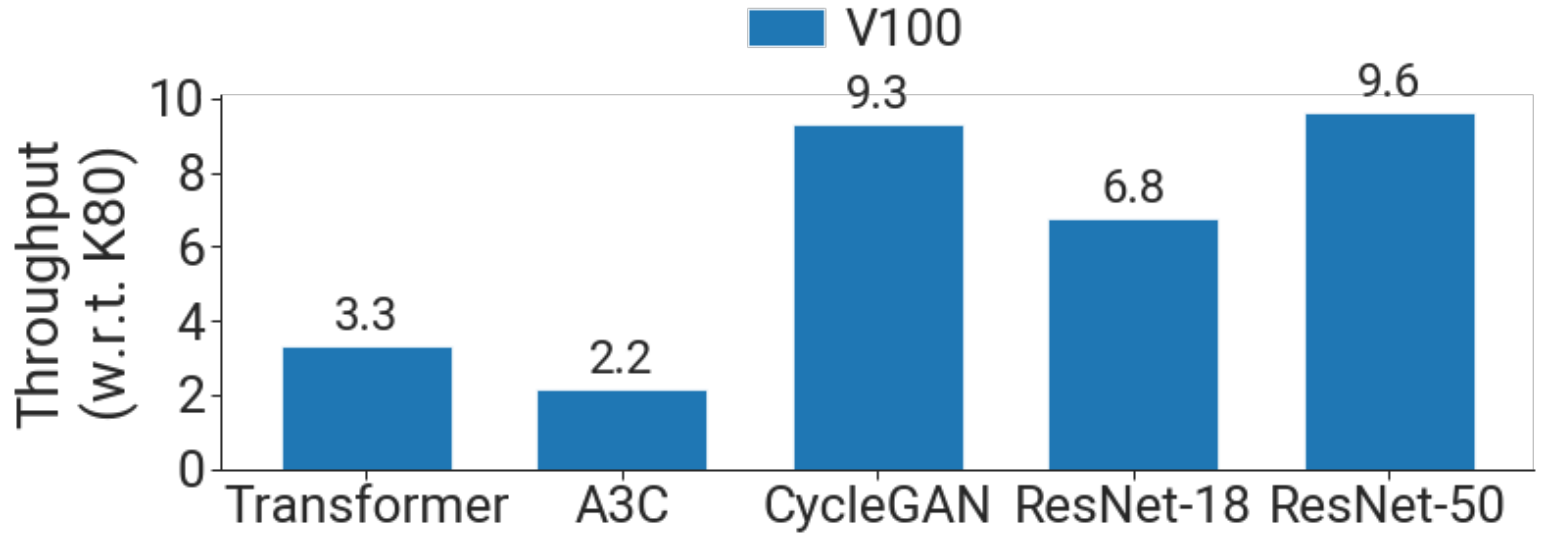
Private clusters

Various GPU generations
A100, V100, P100, K80

TPUs, FPGAs, and others

Models show heterogeneous performance behavior

Models and operators (e.g., convolution, attention) perform differently across hardware architectures



Disregarding heterogeneity can lead to sub-optimal allocations

Goal: Heterogeneity-aware scheduling across objectives

Performance Heterogeneity + Objectives

Single-job objectives: “minimize cost”

Multi-job objectives: fairness or hierarchical policies

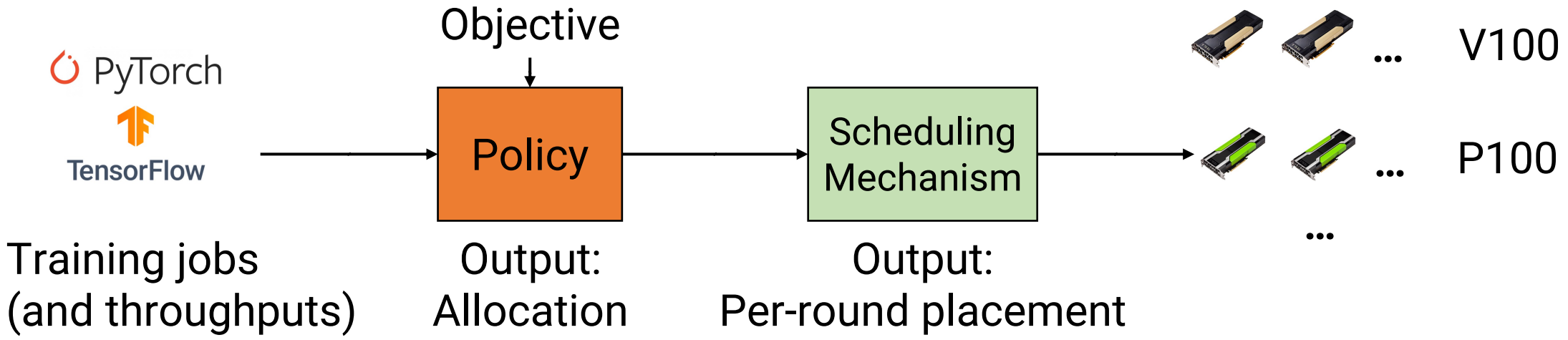
Related work

- Most existing cluster schedulers for deep learning (e.g., Gandiva [1], Themis [2], Tiresias [3]) do not consider performance heterogeneity
- AlloX [4] and Gandiva_fair [5] consider performance heterogeneity, but only support specific objectives

[1] Gandiva: Introspective Cluster Scheduling for Deep Learning, OSDI 2019, Xiao et al.
[2] Themis: Fair and Efficient GPU Cluster Scheduling, NSDI 2020, Mahajan et al.
[3] Tiresias: A GPU Cluster Manager for Distributed Deep Learning, NSDI 2019, Gu et al.
[4] AlloX: Compute Allocation in Hybrid Clusters, EuroSys 2020, Le et al.
[5] Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning, EuroSys 2020, Chaudhary et al.

Gavel: A new heterogeneity-aware scheduler

- Expresses policies as optimization problems over the allocation
- Policy-agnostic round-based scheduling mechanism to realize allocations optimized for various objectives
- Improves objectives such as average job completion time by **3.5**×



Scheduling policies to make heterogeneity-aware

- **FIFO:** First in, first out
- **Shortest Job First:** Minimize time taken by shortest job
- **Minimize cost (w/ SLOs):** Minimize total cost in public cloud (subject to SLOs)
- **LAS [1]:** Max-min fairness by total compute time
- **Finish Time Fairness [2]:** Maximize minimum job speedup
- **Hierarchical:** Multi-level policy with fairness as top-level policy, and FIFO or fairness as lower-level policies. Per-job weights can be specified

[1] Tiresias: A GPU Cluster Manager for Distributed Deep Learning, NSDI 2019, Gu et al.
[2] Themis: Fair and Efficient GPU Cluster Scheduling, NSDI 2020, Mahajan et al.

Insight: Policy objectives are functions of throughput

- Duration of a job (in **Shortest Job First**) = training iterations / throughput
- Cost of a job = cost per hour × (training iterations / throughput)
- Job speedup = throughput / reference throughput

How should we think about throughput in a heterogeneous setting?

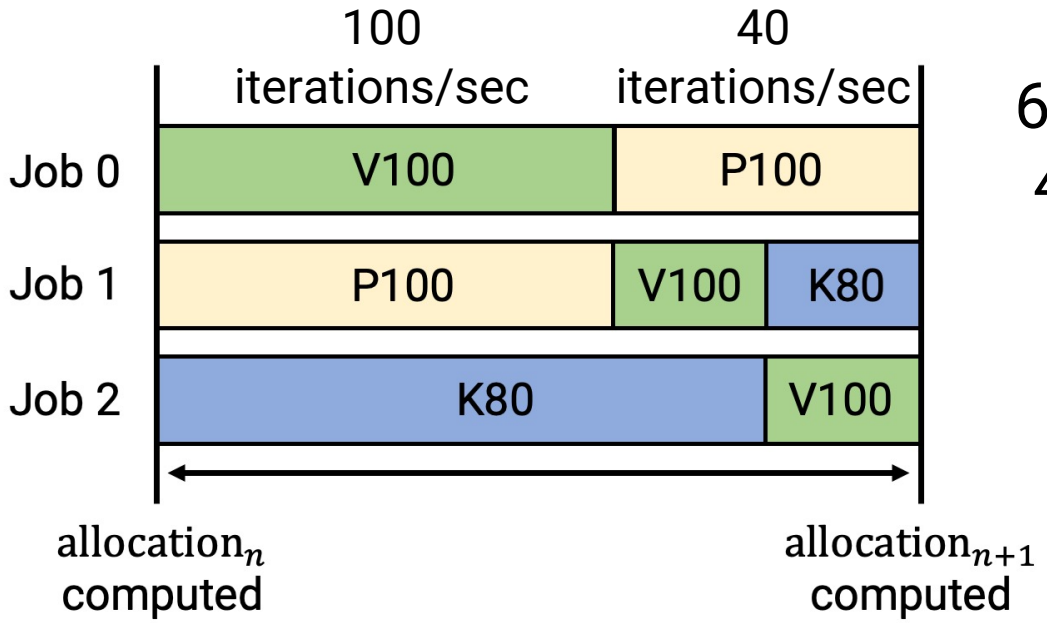
Allocations (X) as time fractions

X = fraction of time jobs spend on each accelerator type

$$X^{\text{example}} = \begin{array}{c} \begin{array}{ccc} V100 & P100 & K80 \end{array} \\ \begin{pmatrix} 0.6 & 0.4 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.0 & 0.8 \end{pmatrix} \end{array} \begin{array}{l} \text{job 0} \\ \text{job 1} \\ \text{job 2} \end{array}$$

Allocations recomputed either at periodic intervals of time, or on a reset event (new job arrives, or old job completes)

How do allocations affect job throughputs?



60% of time at 100 iterations/second +
40% of time at 40 iterations/second =
60 + 16 iterations/second

Given allocation X and raw throughputs T, **effective throughput** is:

$$\text{throughput}(\text{job } m, X) = T_m \cdot X_m$$

Fair sharing policy as optimization problem

Equalizes the resource share each job receives on compute resources

Homogeneous cluster:

$$\text{Maximize}_X \min_m X_m$$

Heterogeneous cluster:

$$\text{Maximize}_X \min_m \text{throughput}(m, X)$$

Policies for other objectives can be formulated similarly

Subject to

$$0 \leq X_m \leq 1$$

$$\sum_m X_m \leq \text{num\_workers}$$

Constraints to make sure resources are not over-allocated

Subject to

$$0 \leq X_{mj} \leq 1$$

$$\sum_m X_{mj} \leq \text{num\_workers}_j \quad \forall j$$

$$\sum_j X_{mj} \leq 1 \quad \forall m$$

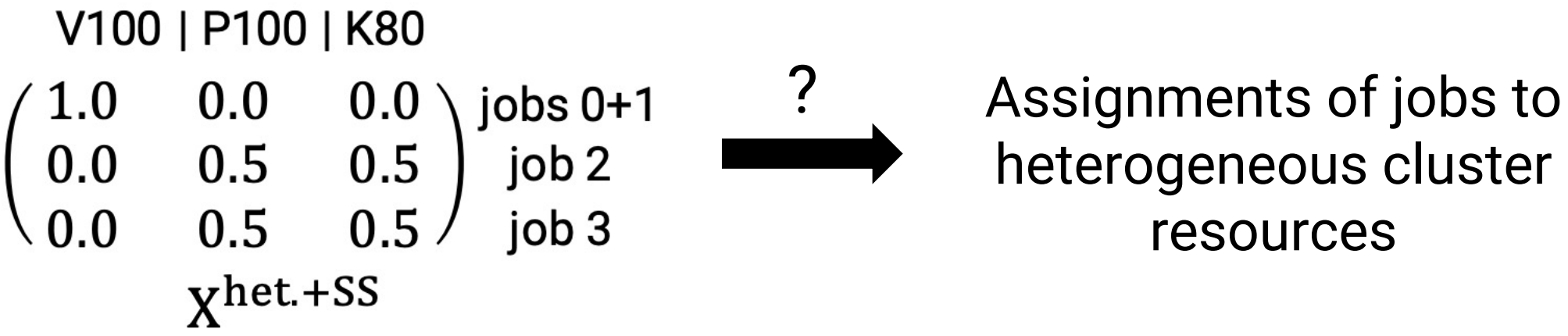
Performance optimizations: space sharing

- Gavel can also deploy existing performance optimizations like space sharing [1, 2] in a heterogeneity-aware way
- Objectives in terms of $\text{throughput}(m, X)$ unchanged!
- X needs to be modified (e.g., allocation for each job combination)

[1] Gandiva: Introspective Cluster Scheduling for Deep Learning, OSDI 2018, Xiao et al.
[2] Themis: Fair and Efficient GPU Cluster Scheduling, NSDI 2020, Mahajan et al.

How do we realize an optimal allocation?

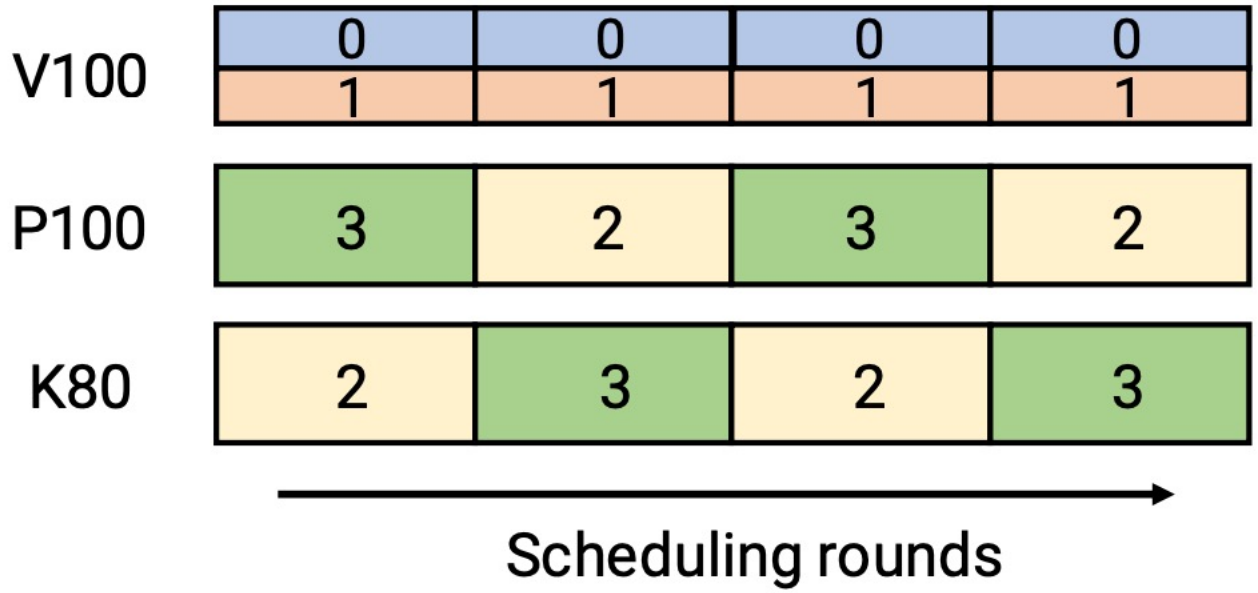
Given an optimal heterogeneity-aware allocation by a policy, how do we assign resources to jobs?



Gavel's round-based scheduling

- Round-based scheduler ensures jobs receive time on accelerator types according to the computed optimal allocation X

$$X^{\text{het.+SS}} = \begin{array}{c} \begin{matrix} \text{V100} & \text{P100} & \text{K80} \end{matrix} \\ \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.5 \\ 0.0 & 0.5 & 0.5 \end{pmatrix} \end{array} \begin{matrix} \text{jobs 0+1} \\ \text{job 2} \\ \text{job 3} \end{matrix}$$



Gavel's round-based scheduling

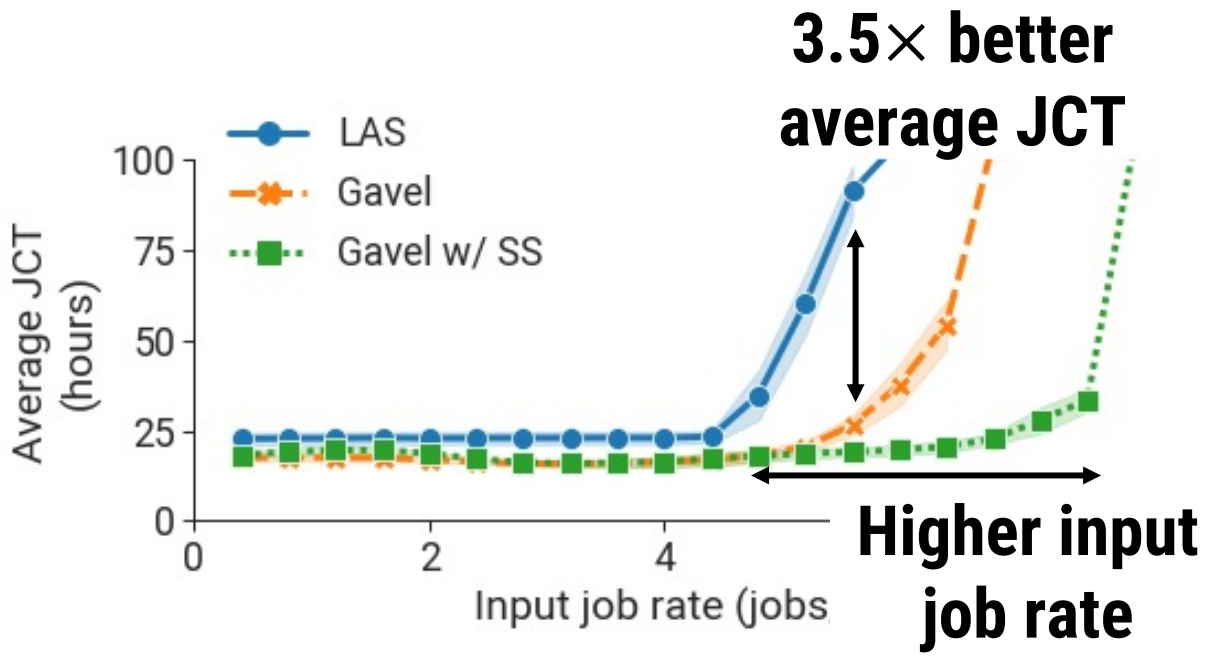
- Round-based scheduler ensures jobs receive time on accelerator types according to the computed optimal allocation X
- Priority score for every (job, accelerator) combination
  - $\text{priorities} = X^{\text{target}} / \text{rounds\_received}$ (element-wise division of matrices)

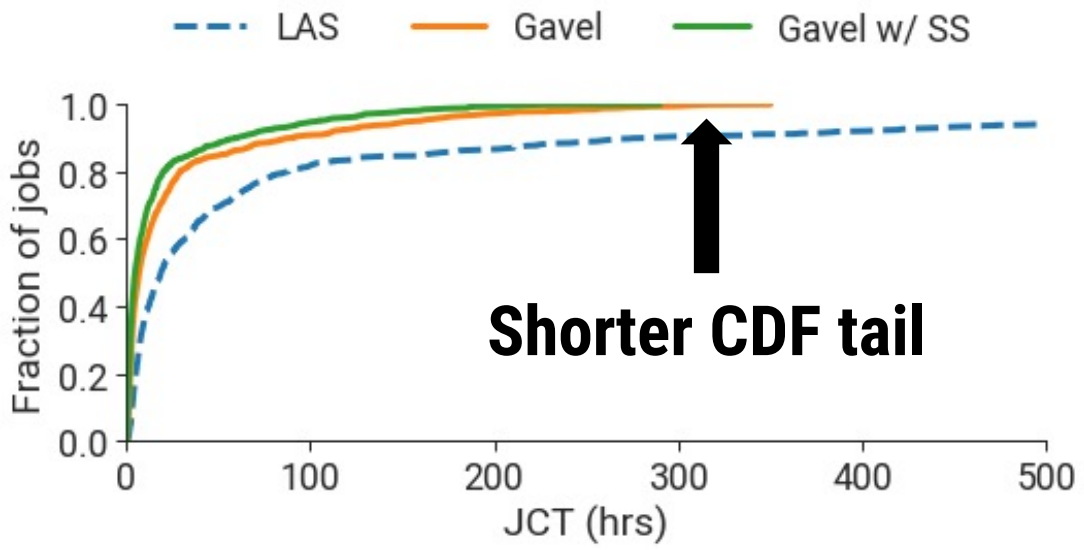
$$X^{\text{example}} = \begin{array}{c} \begin{matrix} V100 & P100 & K80 \end{matrix} \\ \begin{pmatrix} 0.6 & 0.4 & 0.0 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.0 & 0.8 \end{pmatrix} \end{array} \begin{matrix} \\ \text{job 0} \\ \text{job 1} \\ \text{job 2} \end{matrix}$$

$$\begin{array}{c} \text{V100} \mid \text{P100} \mid \text{K80} \\ \begin{pmatrix} 3 & 1 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \end{array} \begin{matrix} \\ \text{job 0} \\ \text{job 1} \\ \text{job 2} \end{matrix}$$

$$\text{rounds\_received}_n$$

Gavel improves objectives on heterogeneous clusters

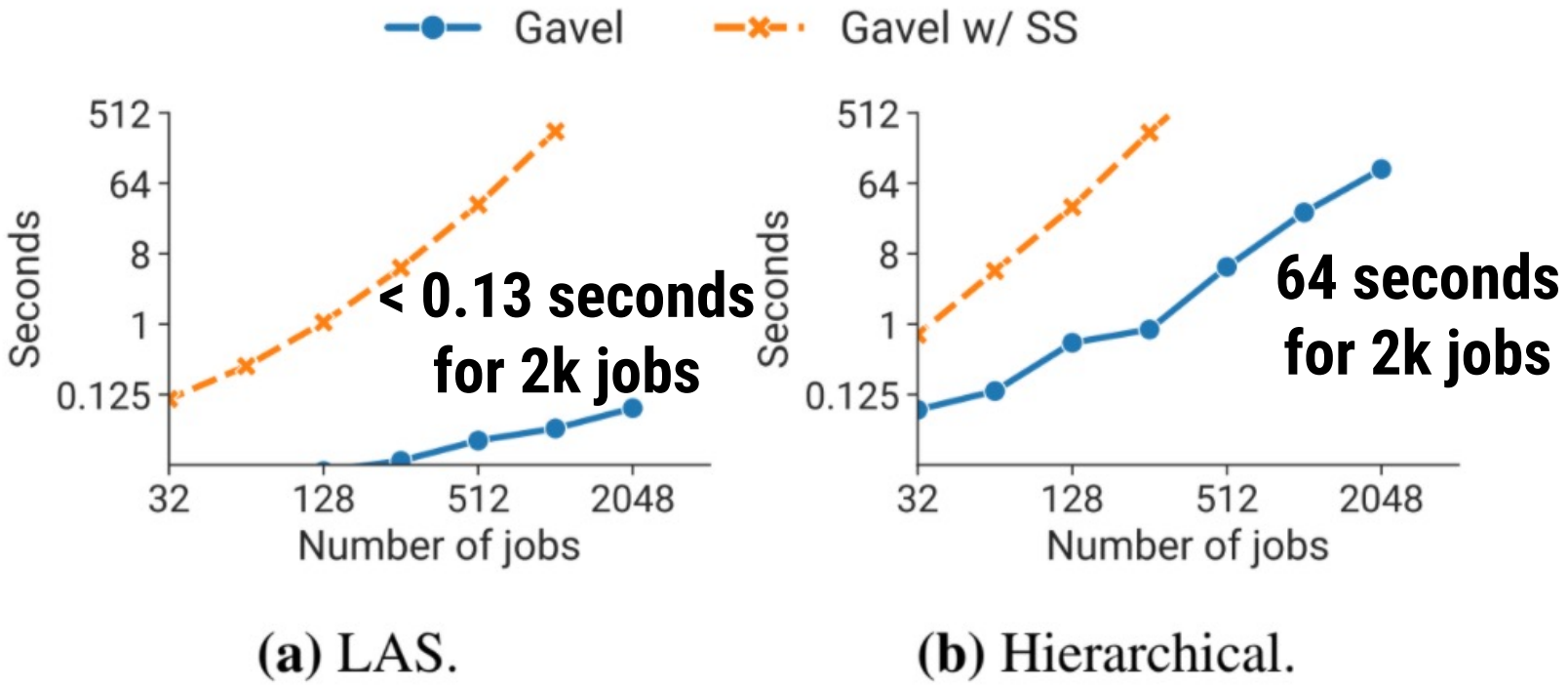




JCT CDF (input job rate = 5.6 jobs/hr)

- **Simulated cluster** with 36 V100 GPUs, 36 P100 GPUs, 36 K80 GPUs
- Each policy evaluated on multiple traces (diff. Poisson arrival rates)

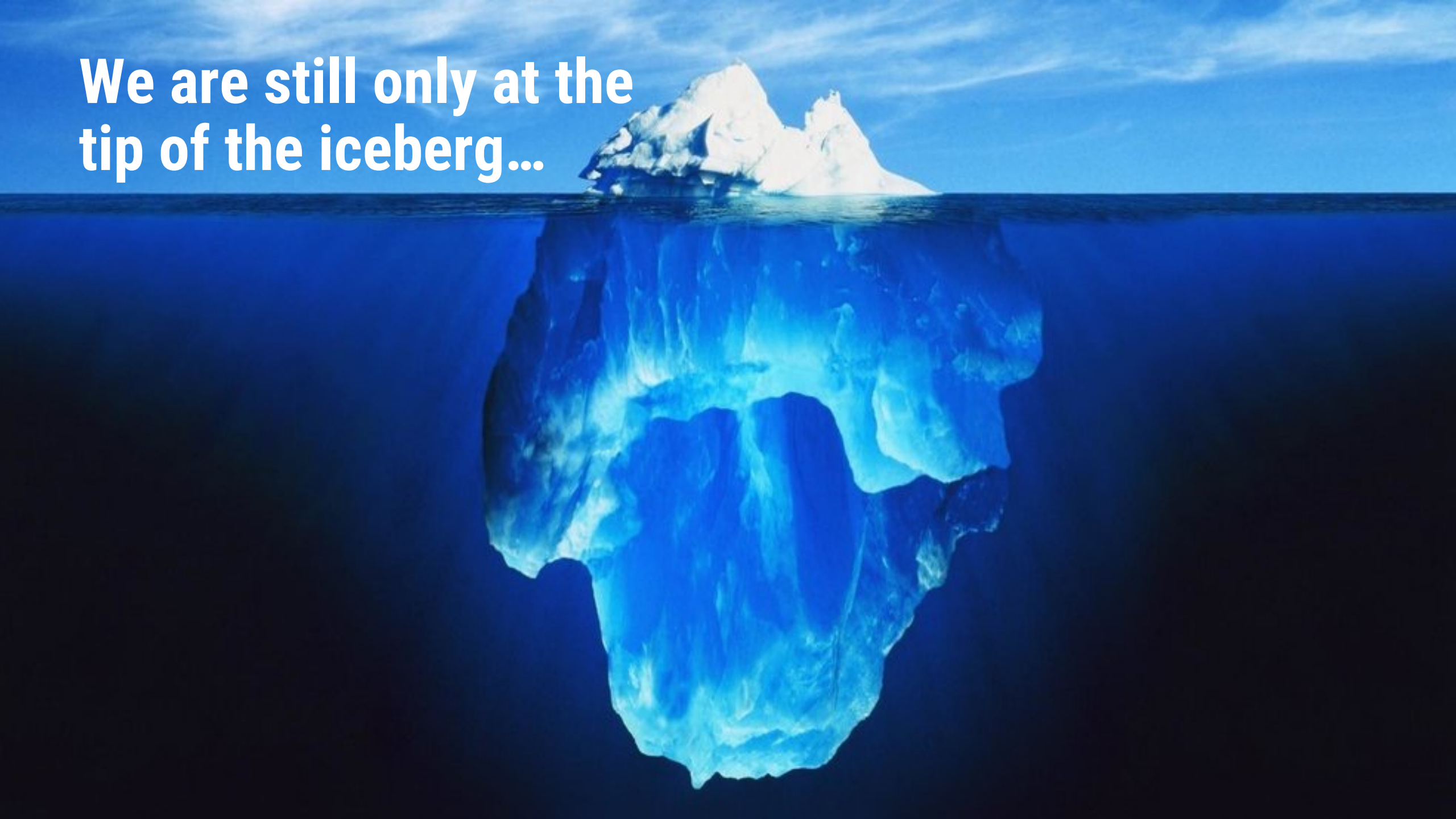
Gavel scales to clusters with 100s of active jobs



Gavel can compute heterogeneity-aware allocations over 2048 jobs in a minute

Takeaways

- Performance heterogeneity makes resource allocation harder
  - Deferring choice to user leads to bad outcomes (e.g., all users choose the fastest resource type, increasing queuing delay)
- Policies can be formulated as optimization problems
  - Effective throughput to incorporate performance heterogeneity
- Objectives such as average JCT can be improved by up to 3.5×


We are still only at the
tip of the iceberg…

Lots of exciting directions for future exploration!

- What is the story for inference?
- Can we think about scheduling and parallelization in a more unified framework? What are the performance implications of this?
- And more!

Resource-Efficient Execution of Deep Learning

Careful automated scheduling of computation on heterogeneous resources across the software stack can increase training throughput

Can improve performance across a number of different contexts, including **distributed training** and **scheduling**

https://www.microsoft.com/en-us/research/people/dnarayanan/

dnarayanan@microsoft.com

@deepakn94