High Performance Computing Systems (CMSC714)

Lecture 24: Machine Learning and HPC

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

Presentation and Final report format

- Upload pdf slides on ELMS after your presentation
  - Introduce your project so that it is understandable by a CS audience
  - Present what you are implementing or evaluating (serial / parallel algorithms)
  - Progress so far
  - Results (performance / performance analysis)
- Final report
  - Upload code and pdf report to ELMS
  - E-mail Abhinav and Joy how you are distributing your virtual dollars (100) among your teammates with justification

Summary of last lecture

- Discrete-event simulations (DES)
- Parallel DES: conservative vs. optimistic
- Trace-driven network simulations: model event sequences
- Simulation of epidemic diffusion: agent-based, time-stepped modeling

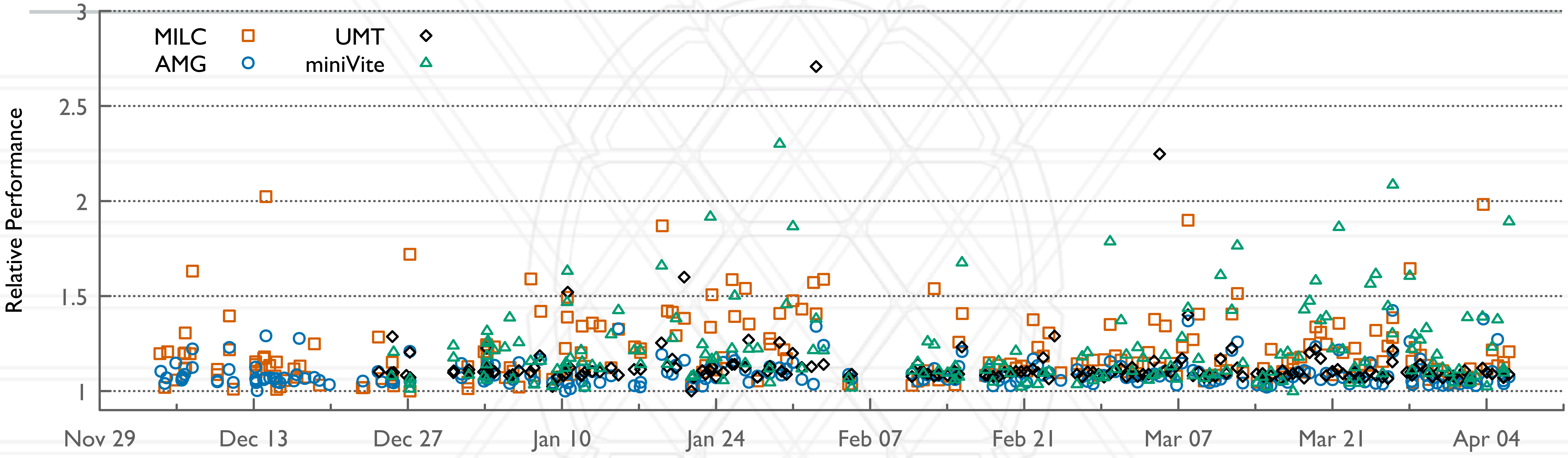
Why machine learning for HPC?

- Proliferation of performance data
  - On-node hardware counters
  - Switch/network port counters
  - Power measurements
  - Traces and profiles
- Supercomputing facilites' data
  - Job queue logs, performance
  - Sensors: temperature, humidity, power

Types of ML-related tasks in HPC

- Auto-tuning: parameter search
  - Find a well performing configuration
- Predictive models: time, energy, …
  - Predict system state in the future
  - Time-series analysis
- Identifying root causes/factors

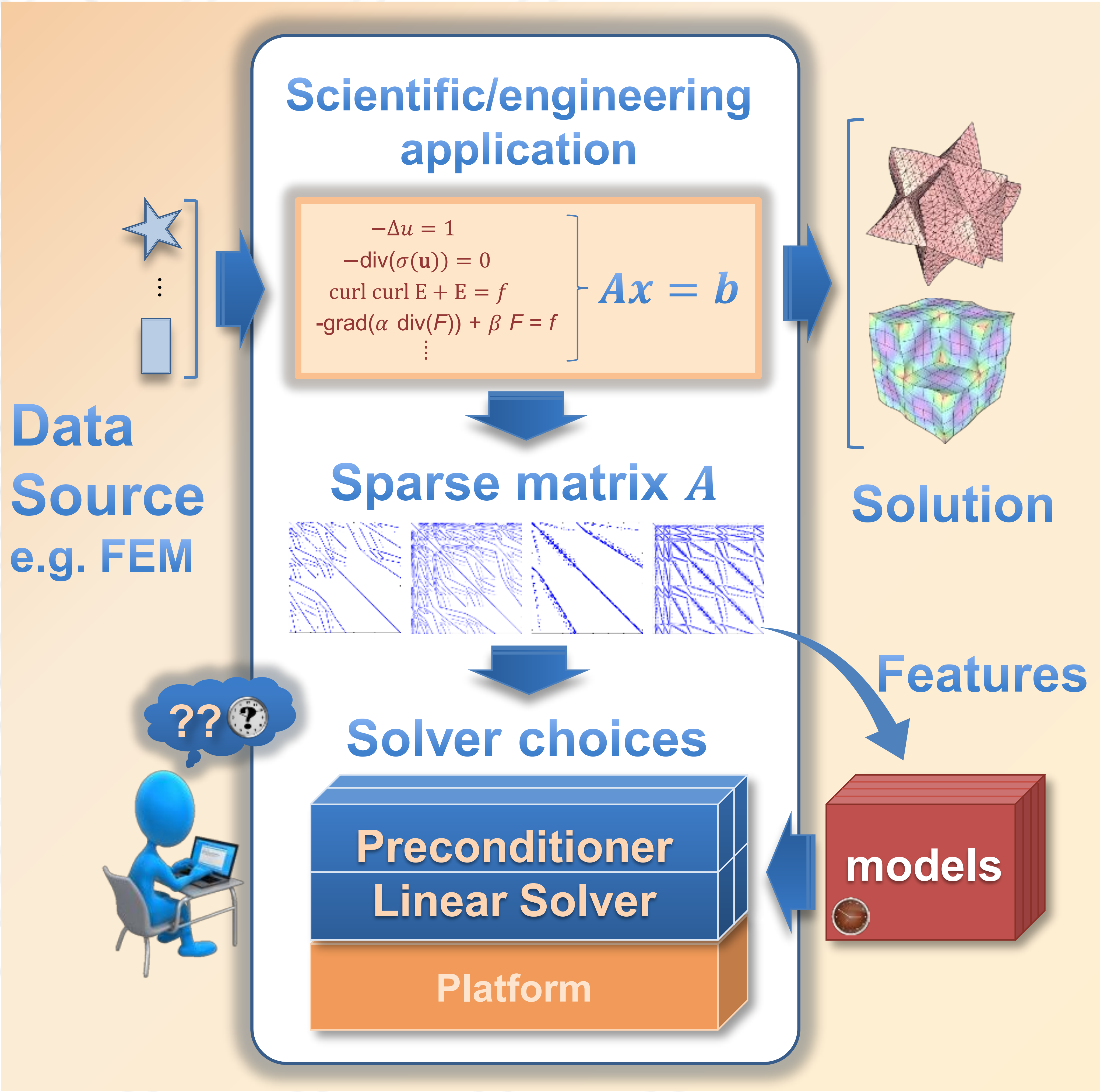
Investigating performance variability



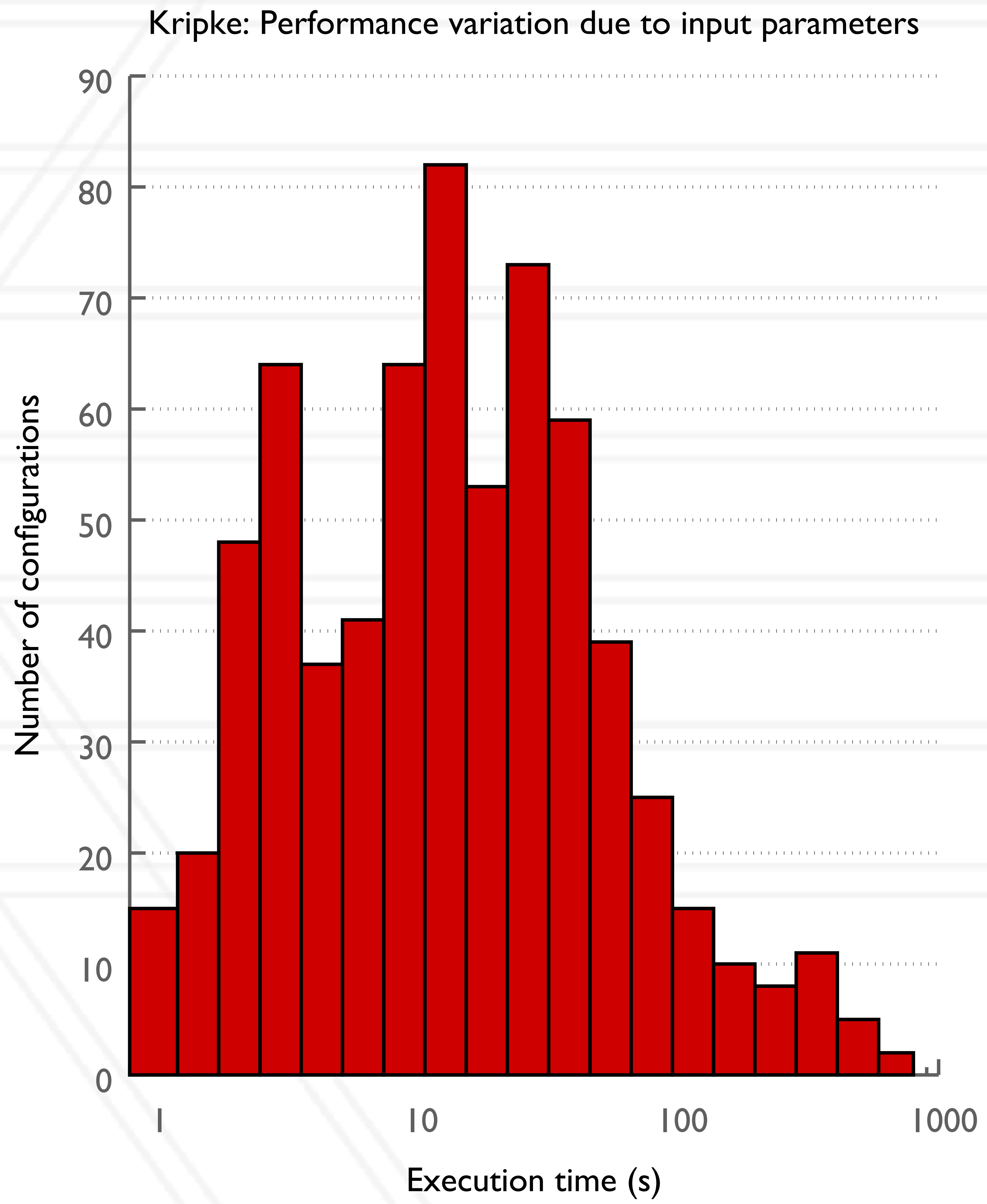
- Identify users to blame, important network counters
- Predict future performance based on historical time-series data

Identifying best performing code variants

- Many computational science and engineering (CSE) codes rely on solving sparse linear systems
- Many choices of numerical methods
- Optimal choice w.r.t. performance depends on several things:
  - Input data and its representation, algorithm and its implementation, hardware architecture

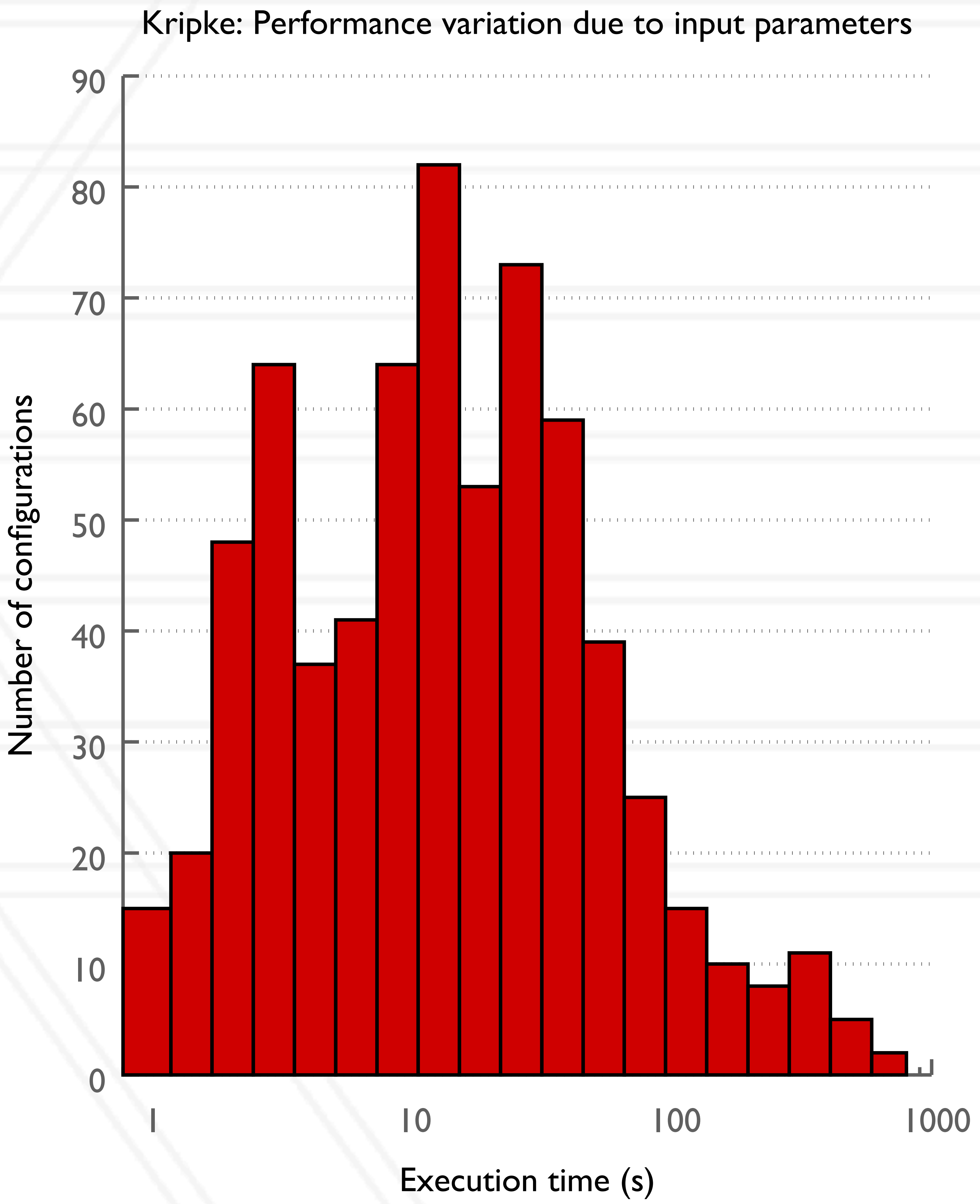


Auto-tuning with limited training data



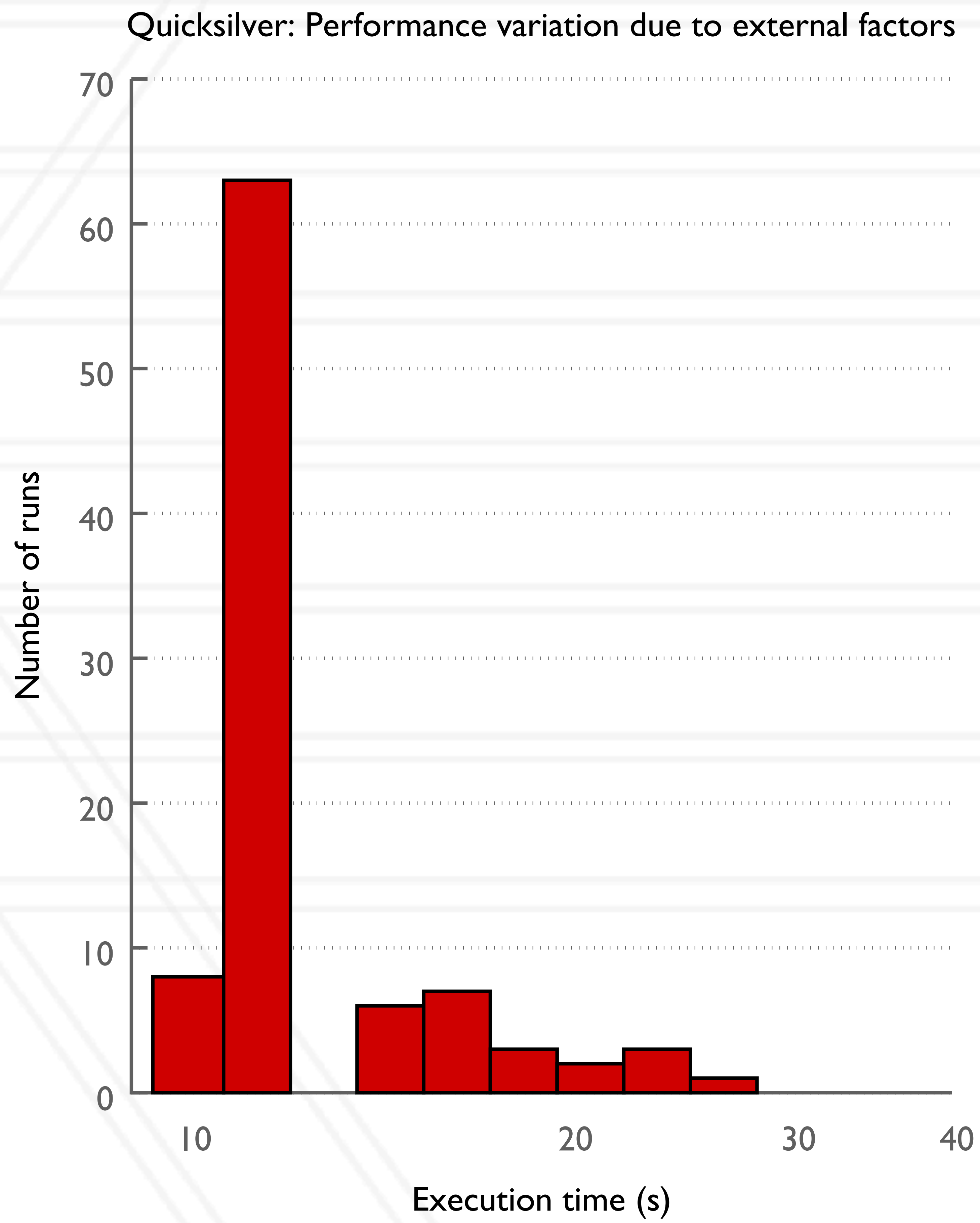
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters



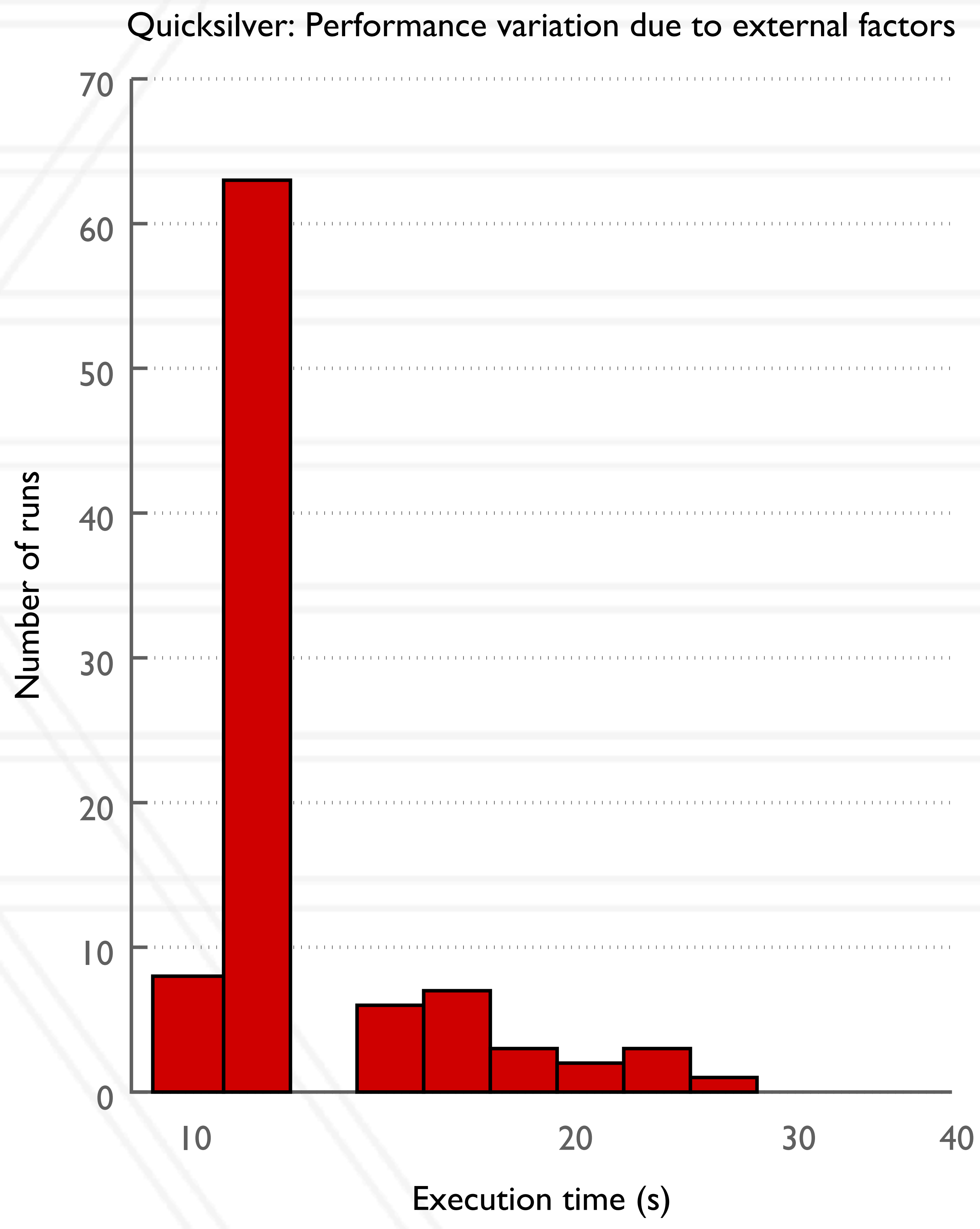
Auto-tuning with limited training data

- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters
- Performance also depends on:
  - Code changes, linked libraries
  - Compilers, architecture



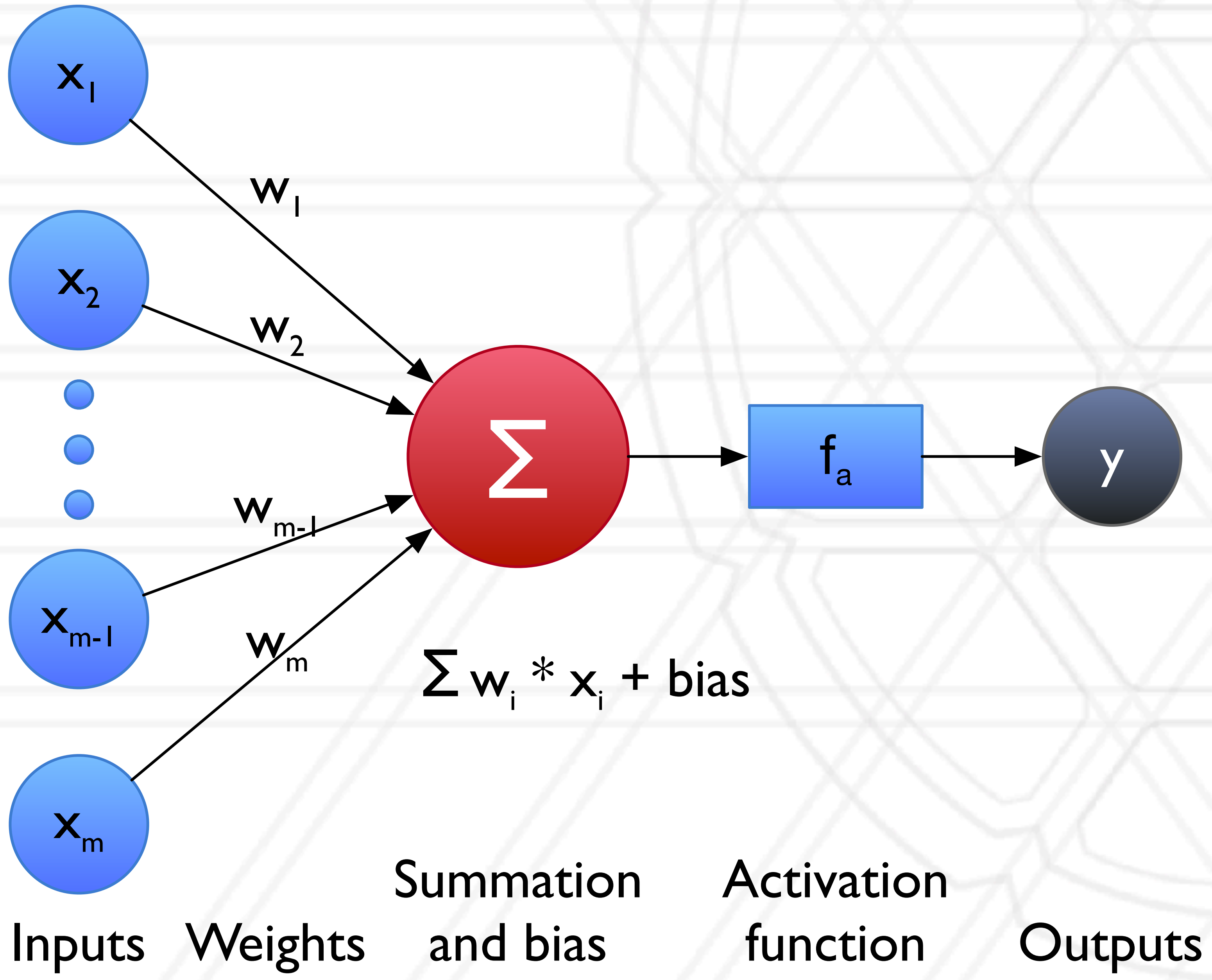
Auto-tuning with limited training data

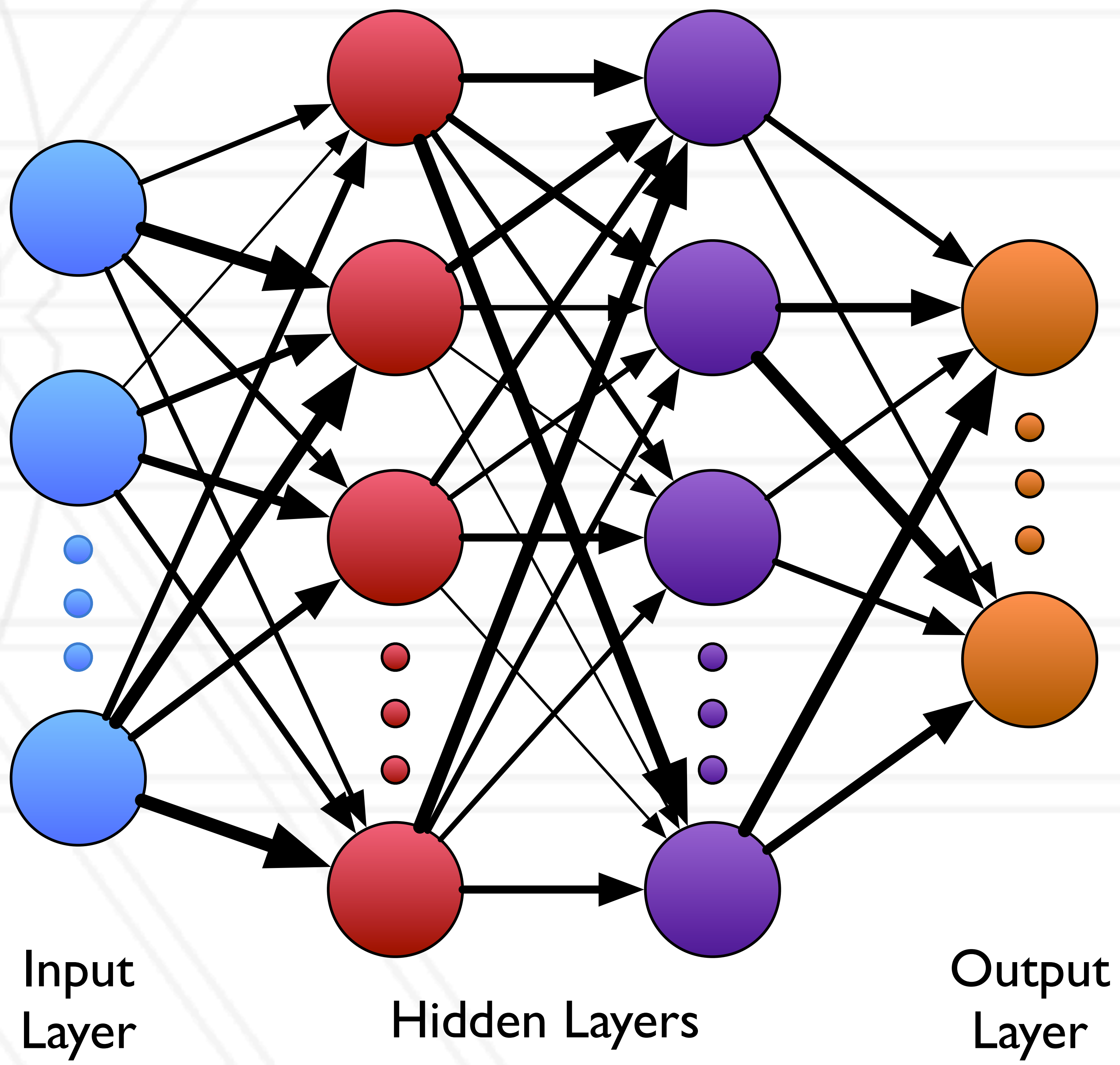
- Application performance depends on many factors:
  - Input parameters, algorithmic choices, runtime parameters
- Performance also depends on:
  - Code changes, linked libraries
  - Compilers, architecture
- Surrogate models + transfer learning



Deep neural networks

- Neural networks can be used to model complex functions
- Several layers that process "batches" of the input data



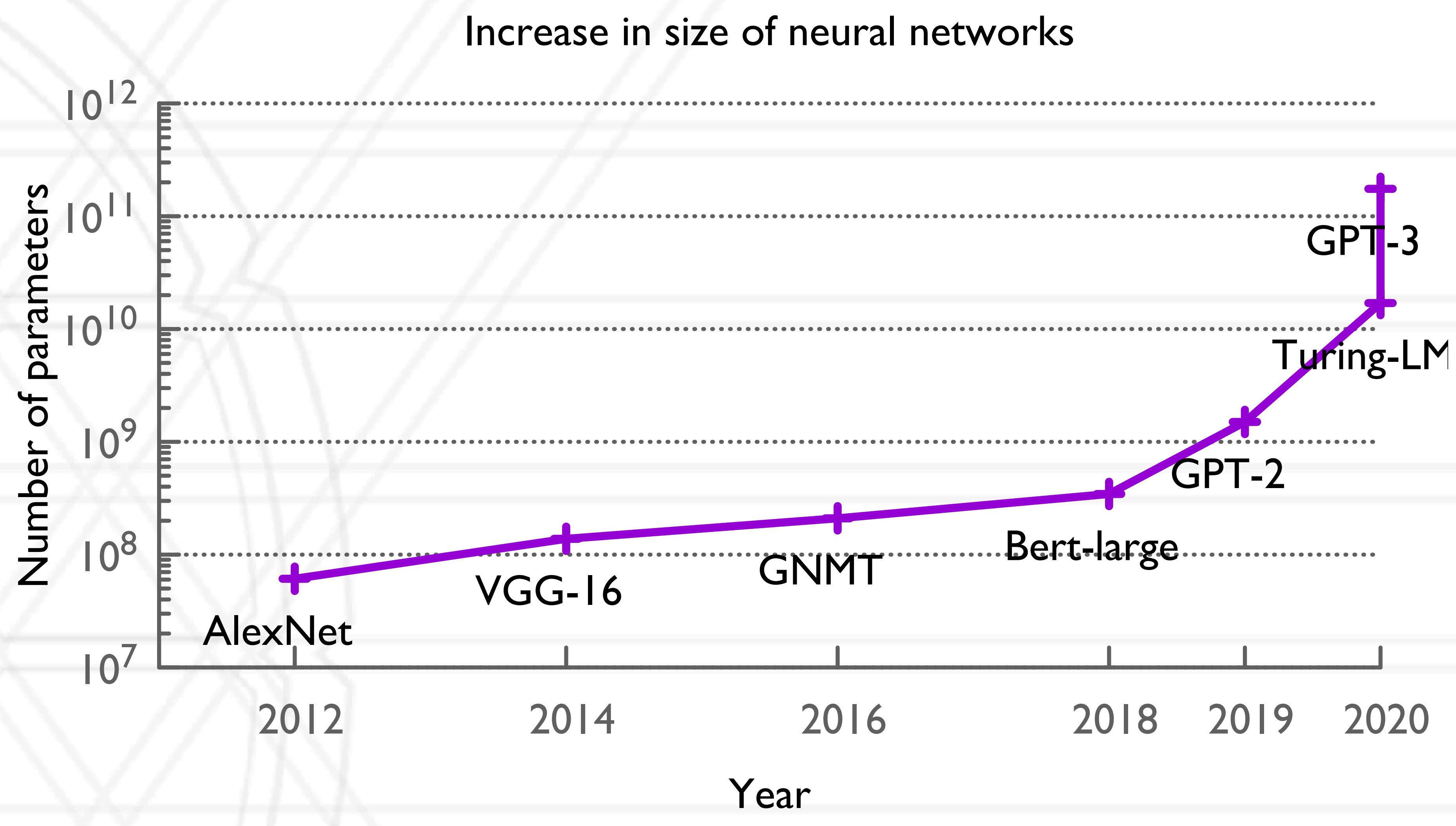


Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (batches)
- Many layers in DNNs

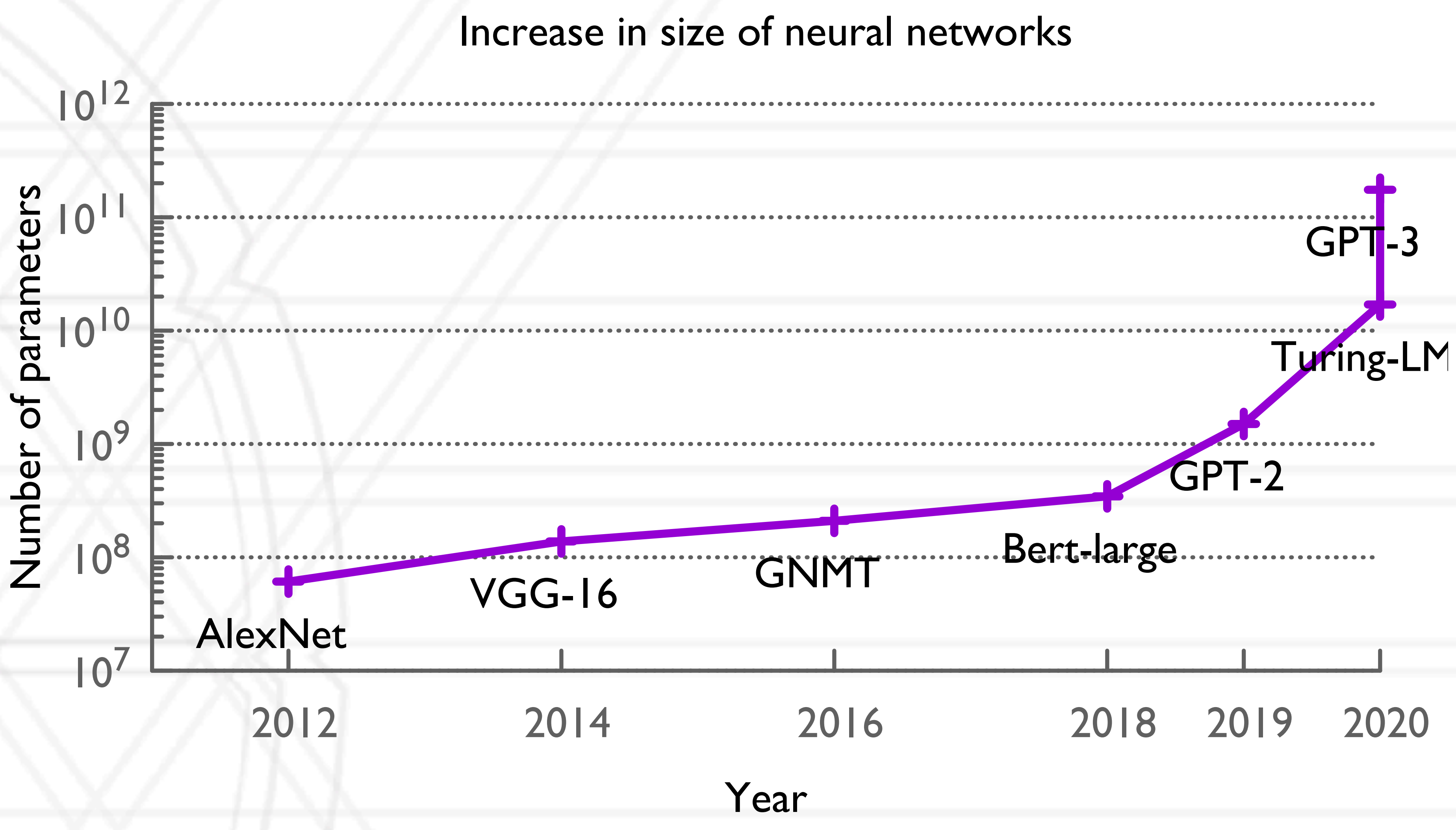
Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (batches)
- Many layers in DNNs



Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (batches)
- Many layers in DNNs



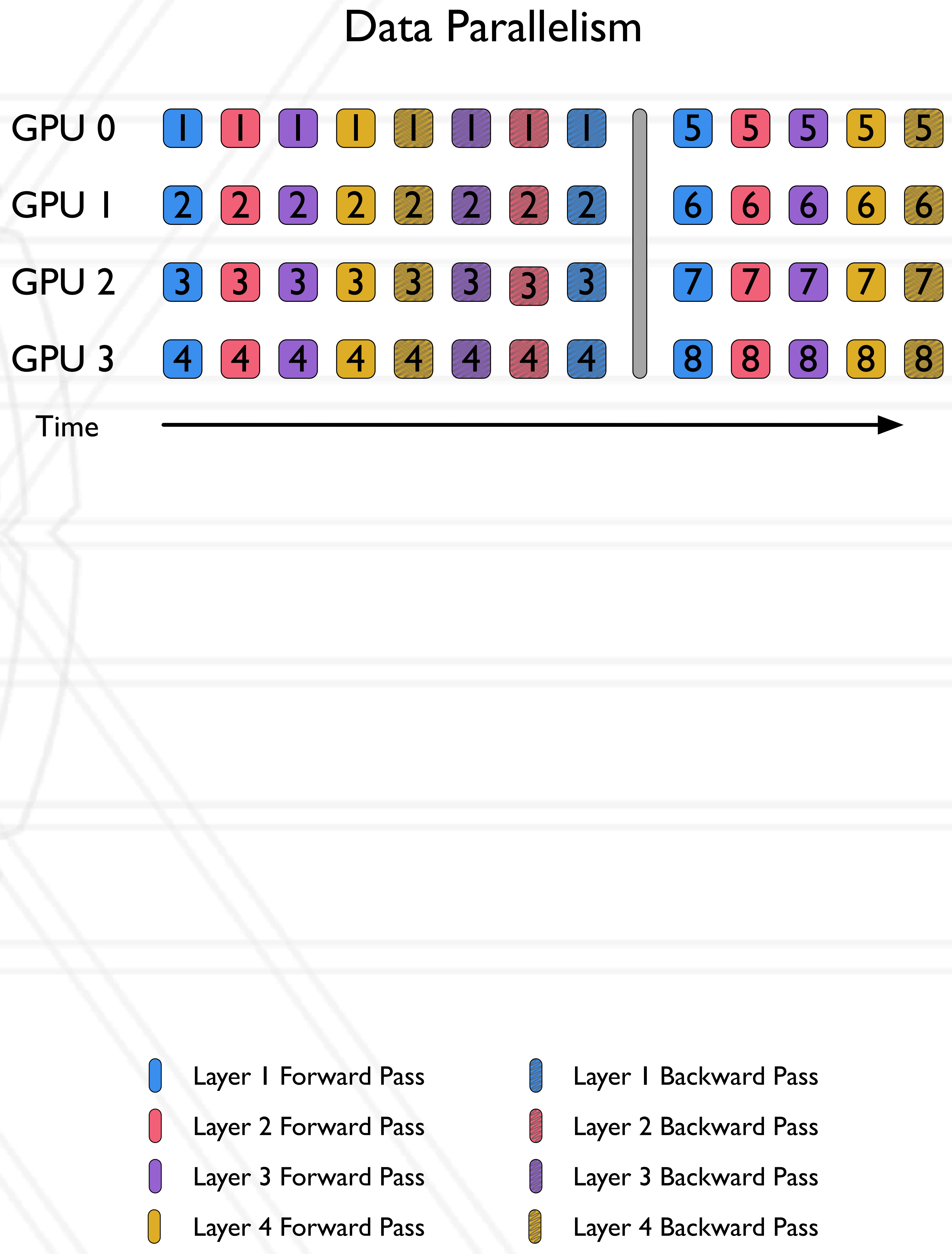
Framework	Type of Parallelism	Largest Accelerator Count	Largest Trained Network (No. of Parameters)
FlexFlow	Hybrid	64 GPUs	24M*
PipeDream	Inter-Layer	16 GPUs	138M
DDP	Data	256 GPUs	345M
GPipe	Inter-Layer	8 GPUs	557M
MeshTensorFlow	Intra-Layer	512-core TPUv2	4.9B
Megatron	Intra-Layer	512 GPUs	8.3B
TorchGPipe	Inter-Layer	8 GPUs	15.8B
KARMA	Data	2048 GPUs	17B
LBANN	Data	3072 CPUs	78.6B
ZeRO	Data	400 GPUs	100B

Different approaches

- Data Parallelism: Each process has a copy of the entire NN and processes different data
  - All reduce operation to synchronize gradients
- Intra-layer Parallelism: Distribute the work within a layer between multiple processes/GPUs
- Inter-layer Parallelism: Distribute entire layers to different processes/GPUs
  - Point-to-point communication (activations and gradients) between processes/GPUs managing different layers

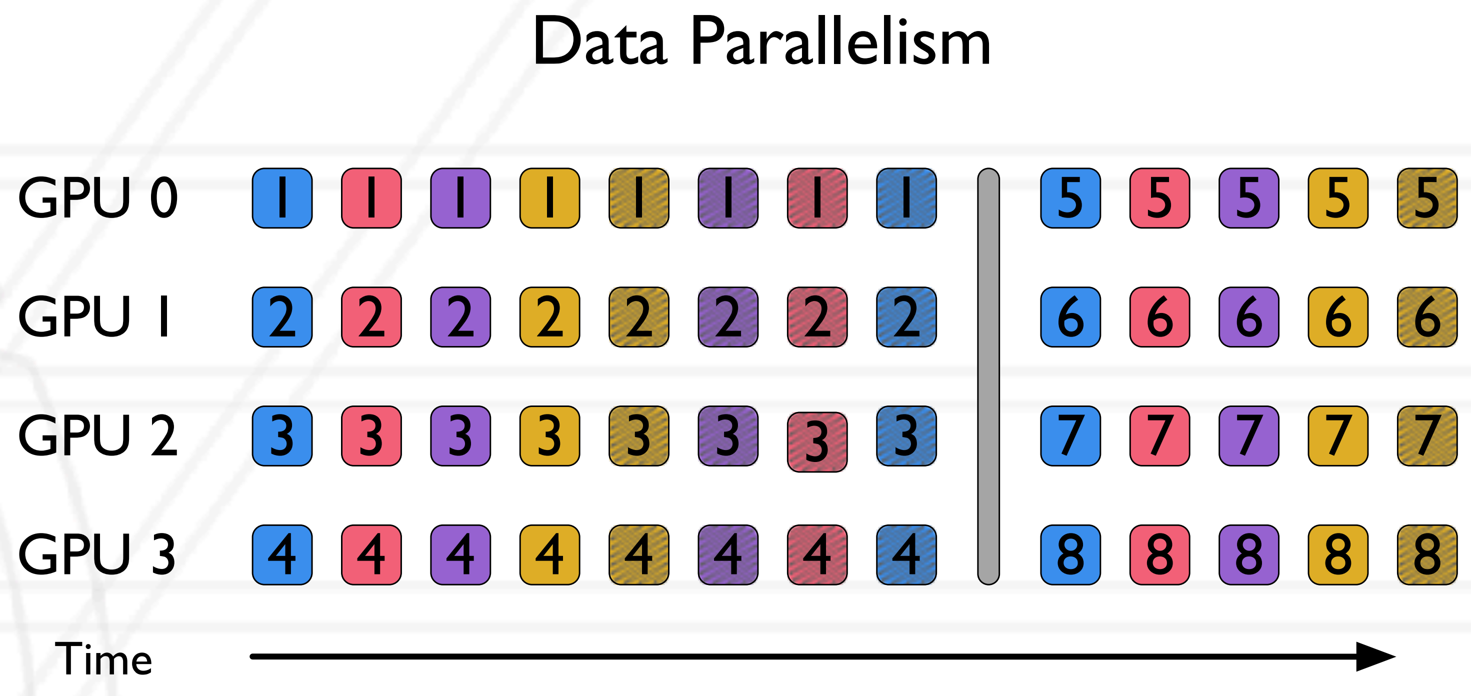
Different approaches

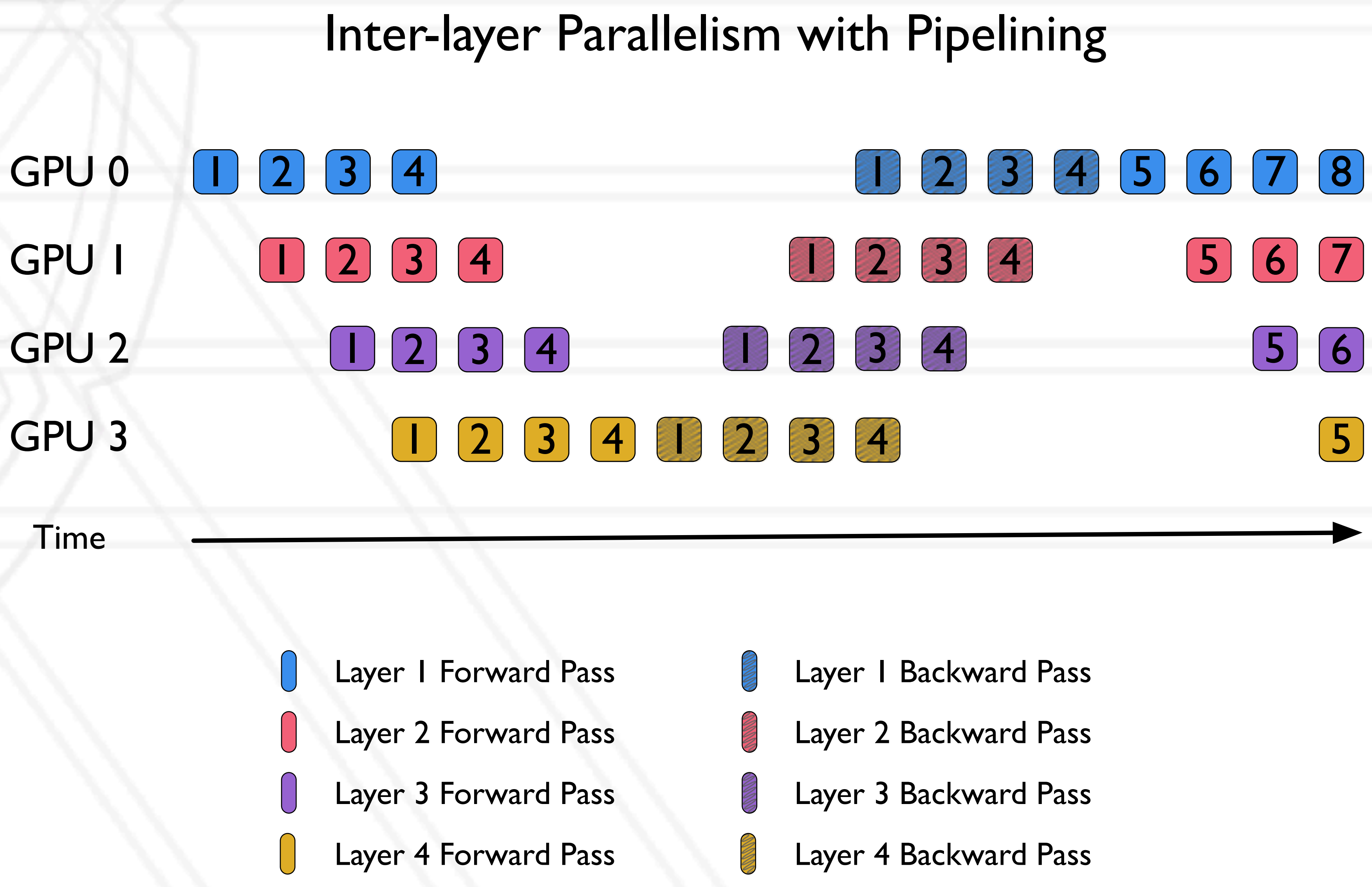
- Data Parallelism: Each process has a copy of the entire NN and processes different data
  - All reduce operation to synchronize gradients
- Intra-layer Parallelism: Distribute the work within a layer between multiple processes/GPUs
- Inter-layer Parallelism: Distribute entire layers to different processes/GPUs
  - Point-to-point communication (activations and gradients) between processes/GPUs managing different layers



Different approaches

- Data Parallelism: Each process has a copy of the entire NN and processes different data
  - All reduce operation to synchronize gradients
- Intra-layer Parallelism: Distribute the work within a layer between multiple processes/GPUs
- Inter-layer Parallelism: Distribute entire layers to different processes/GPUs
  - Point-to-point communication (activations and gradients) between processes/GPUs managing different layers





Questions?

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu