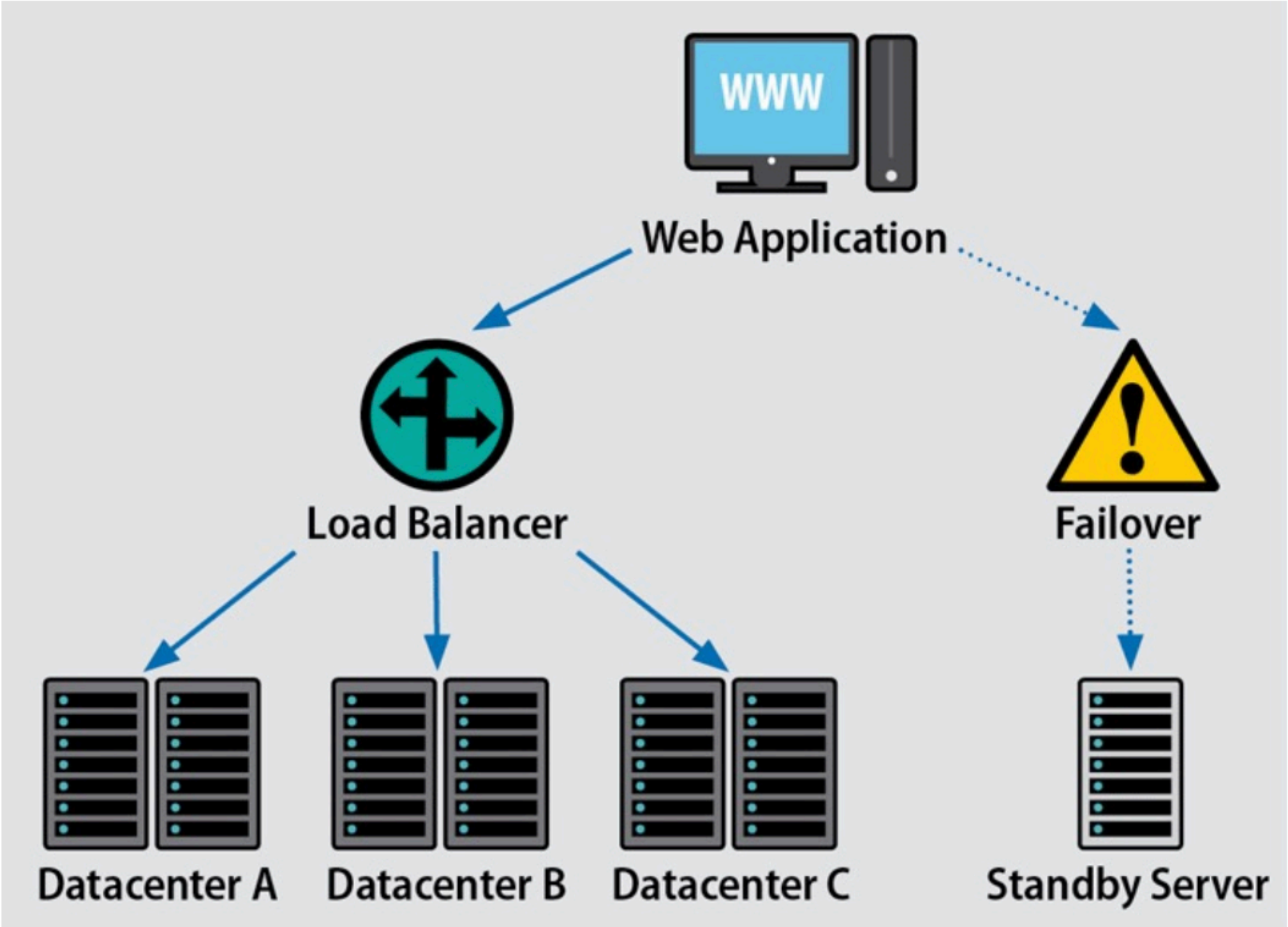
CS4740
CLOUD COMPUTING

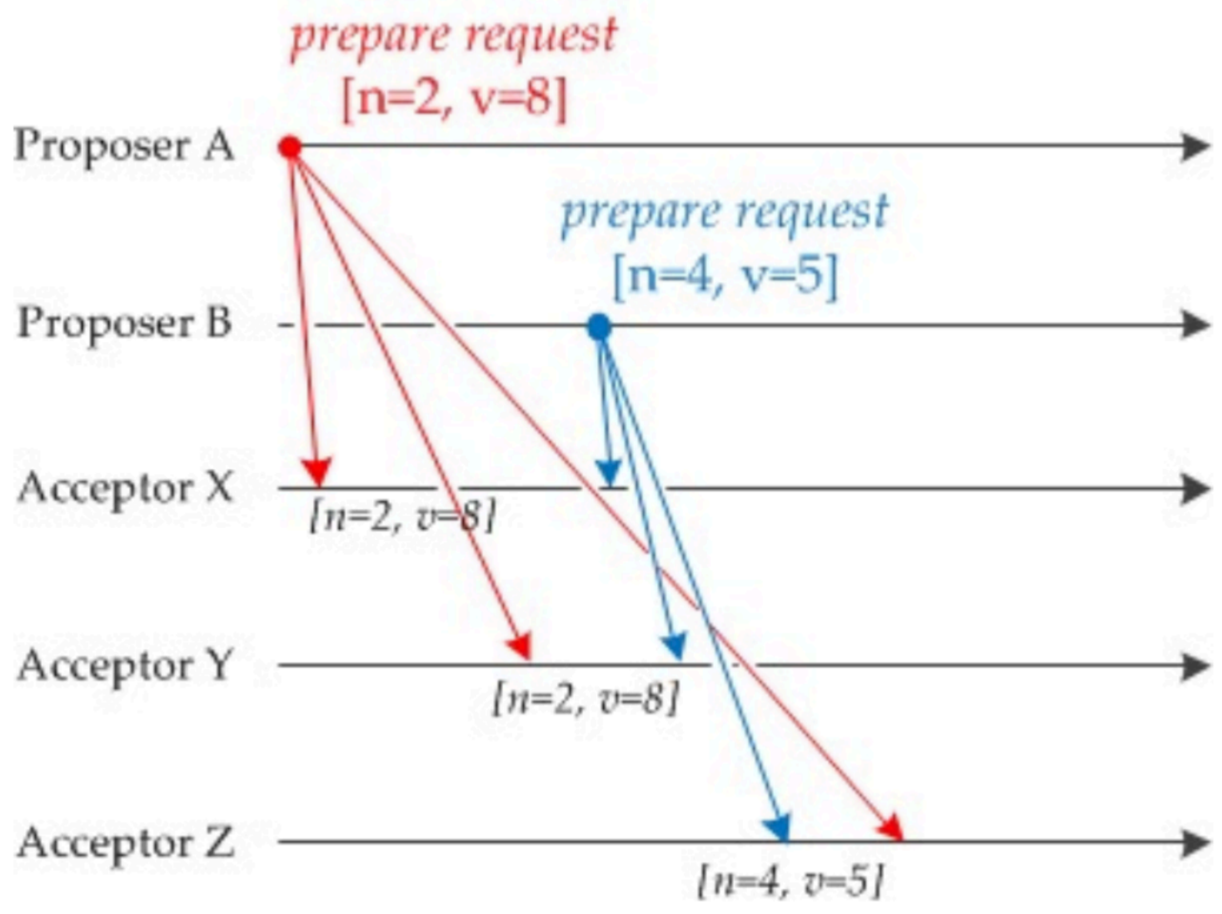
MLSys Primer

Prof. Chang Lou, UVA CS, Fall 2025

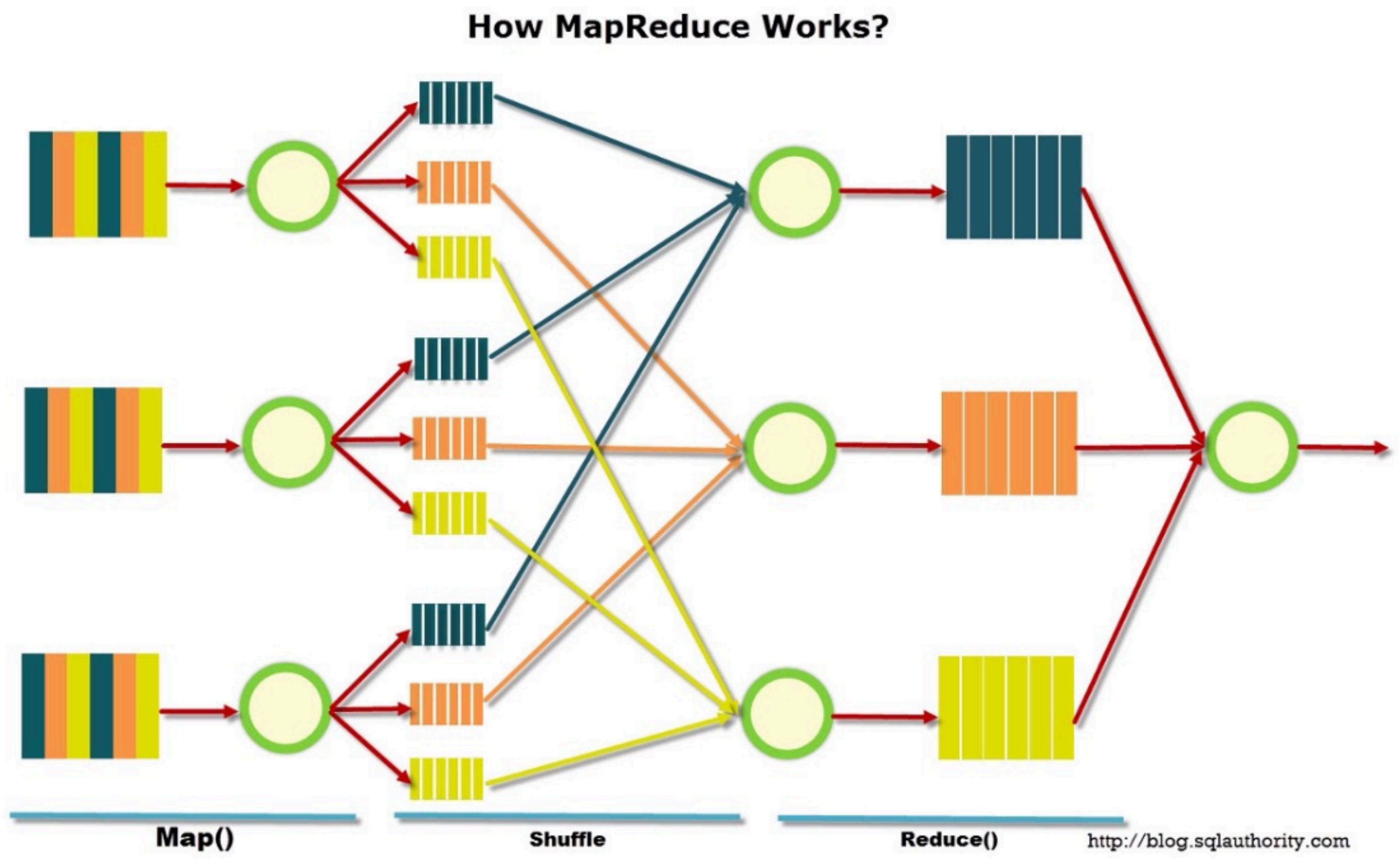
FUNDAMENTALS OF DISTRIBUTED SYSTEMS



Fault Tolerance



Synchronization/Consensus



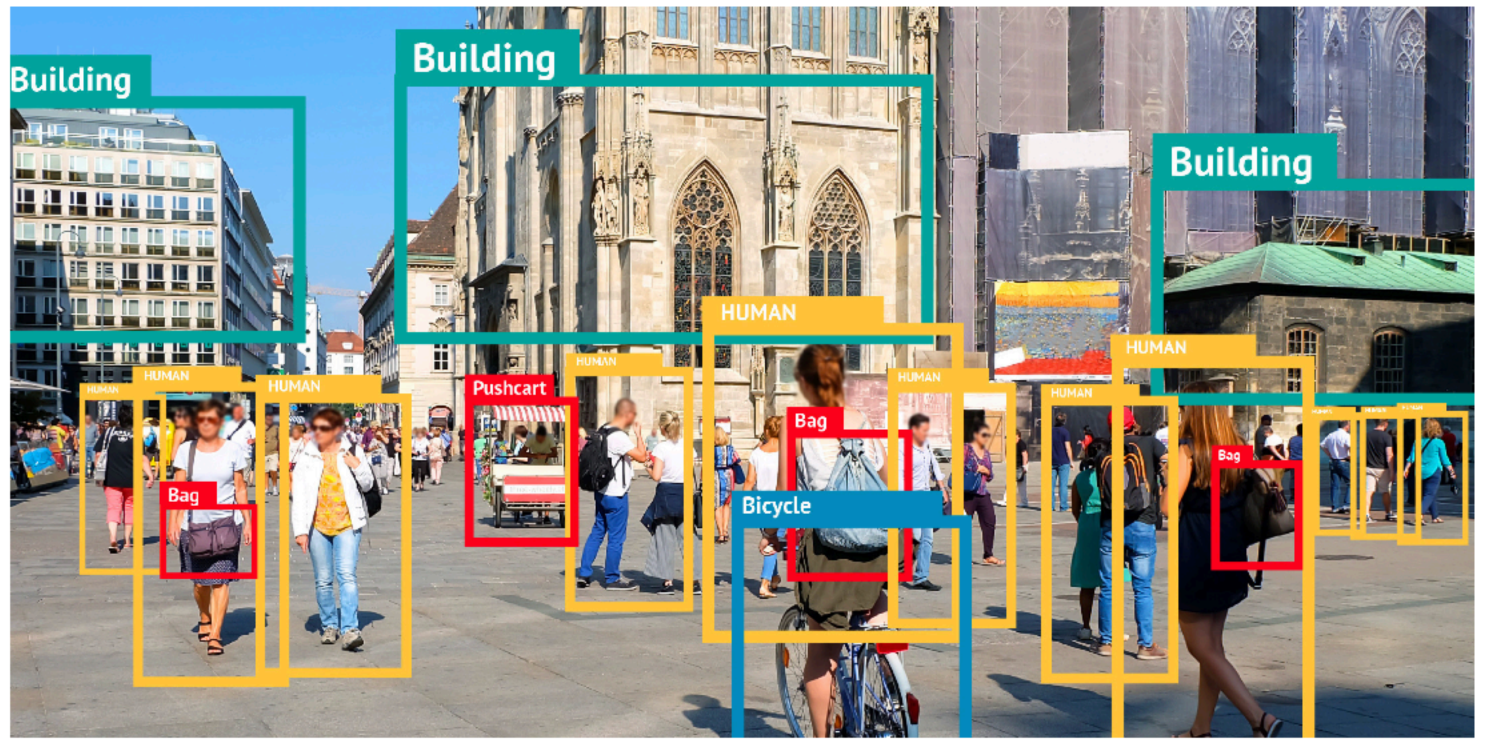
Massive Parallelization

Also apply to systems for machine learning!

LECTURE GOALS

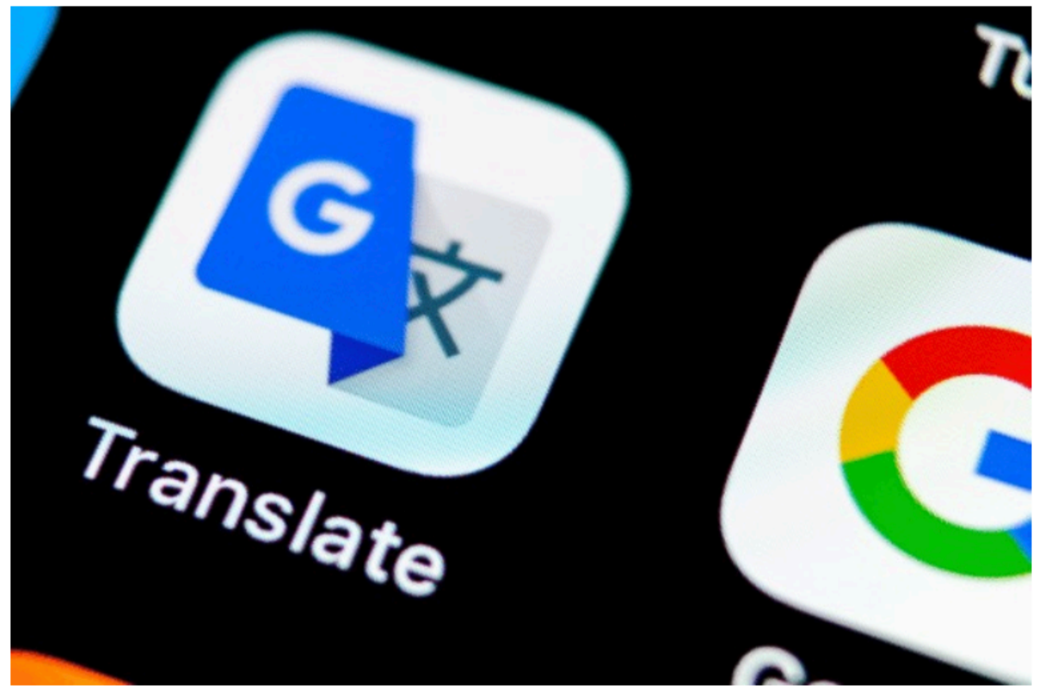
- Define systems for machine learning
- Understand challenges and considerations in designing such systems
- Explore a widely deployed system for ML (TensorFlow)

THE SUCCESS OF MACHINE LEARNING TODAY

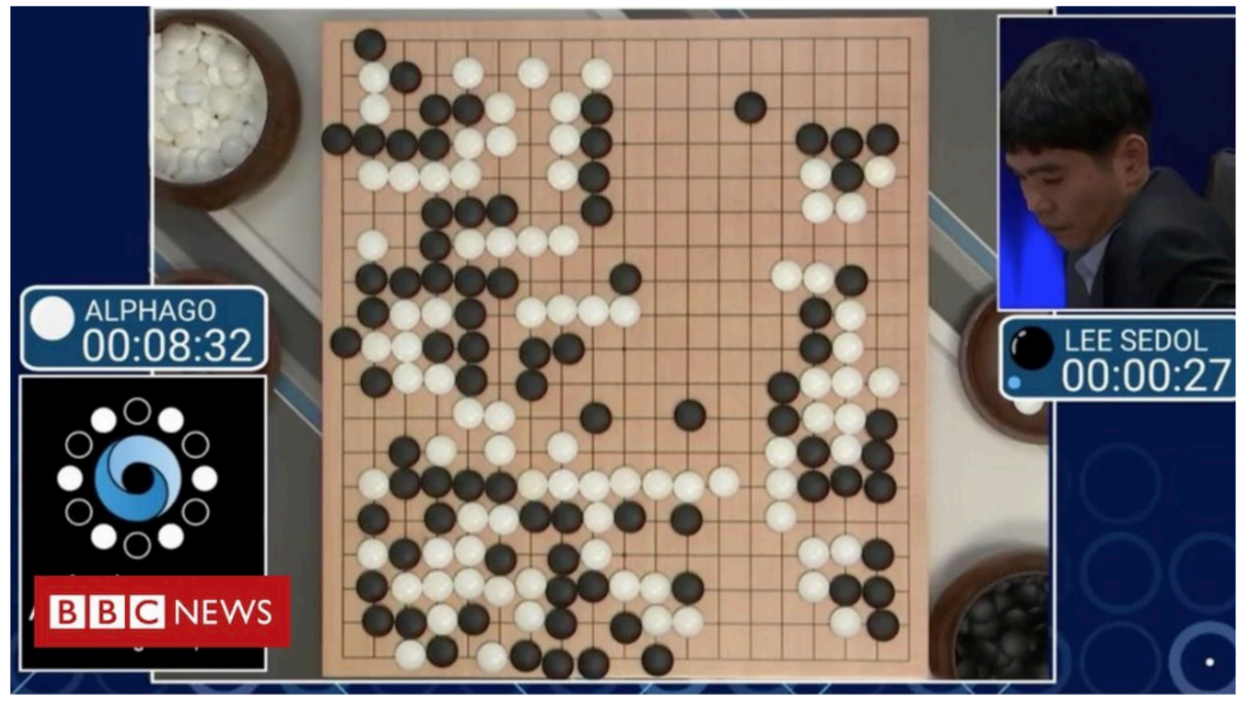


Object detection


Autonomous vehicles



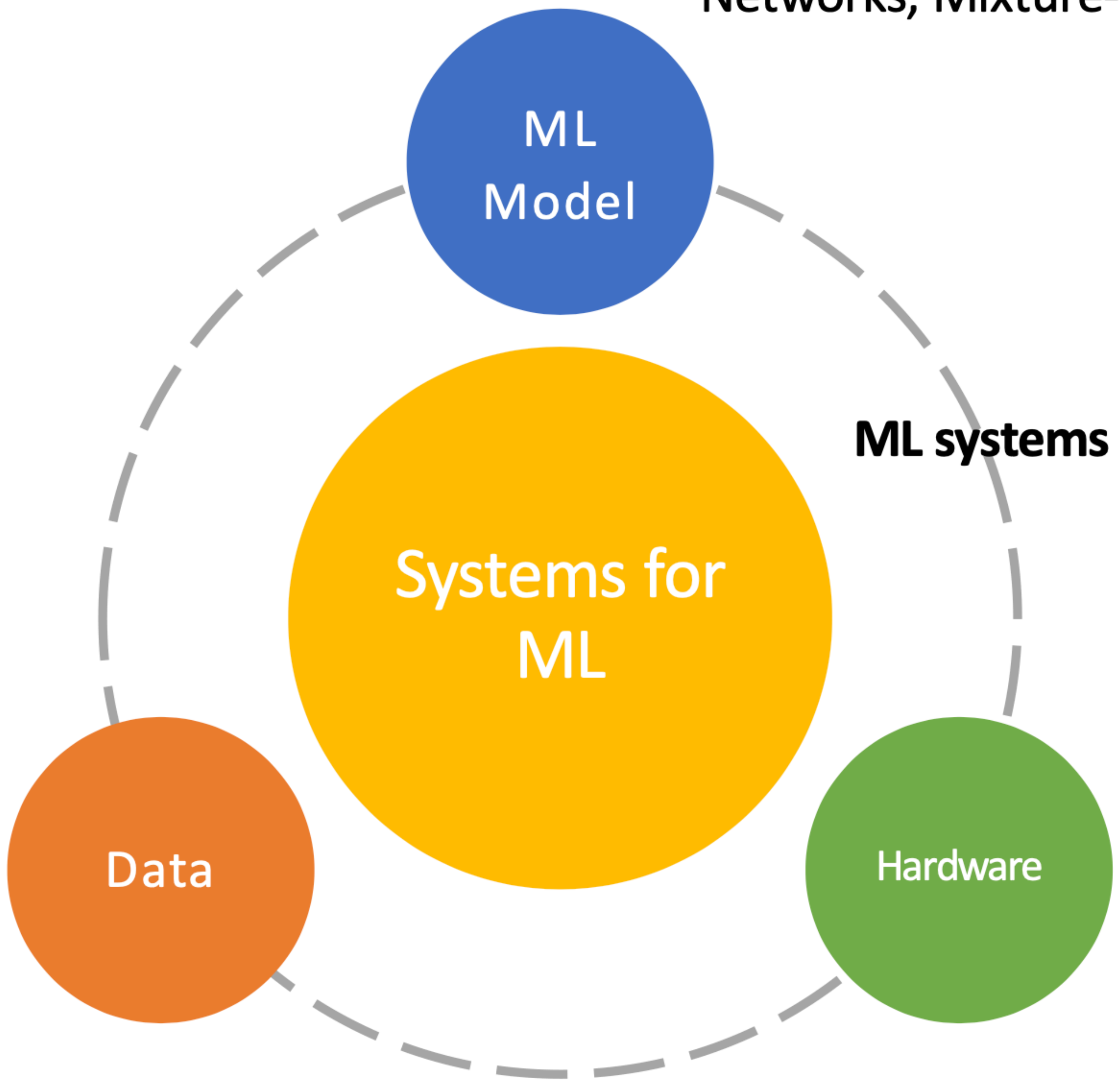
Language Modeling



Game playing

THREE KEY INGREDIENTS IN ML SUCCESS

ResNet, Transformers, Graph Neural Networks, Mixture-of-Experts, …

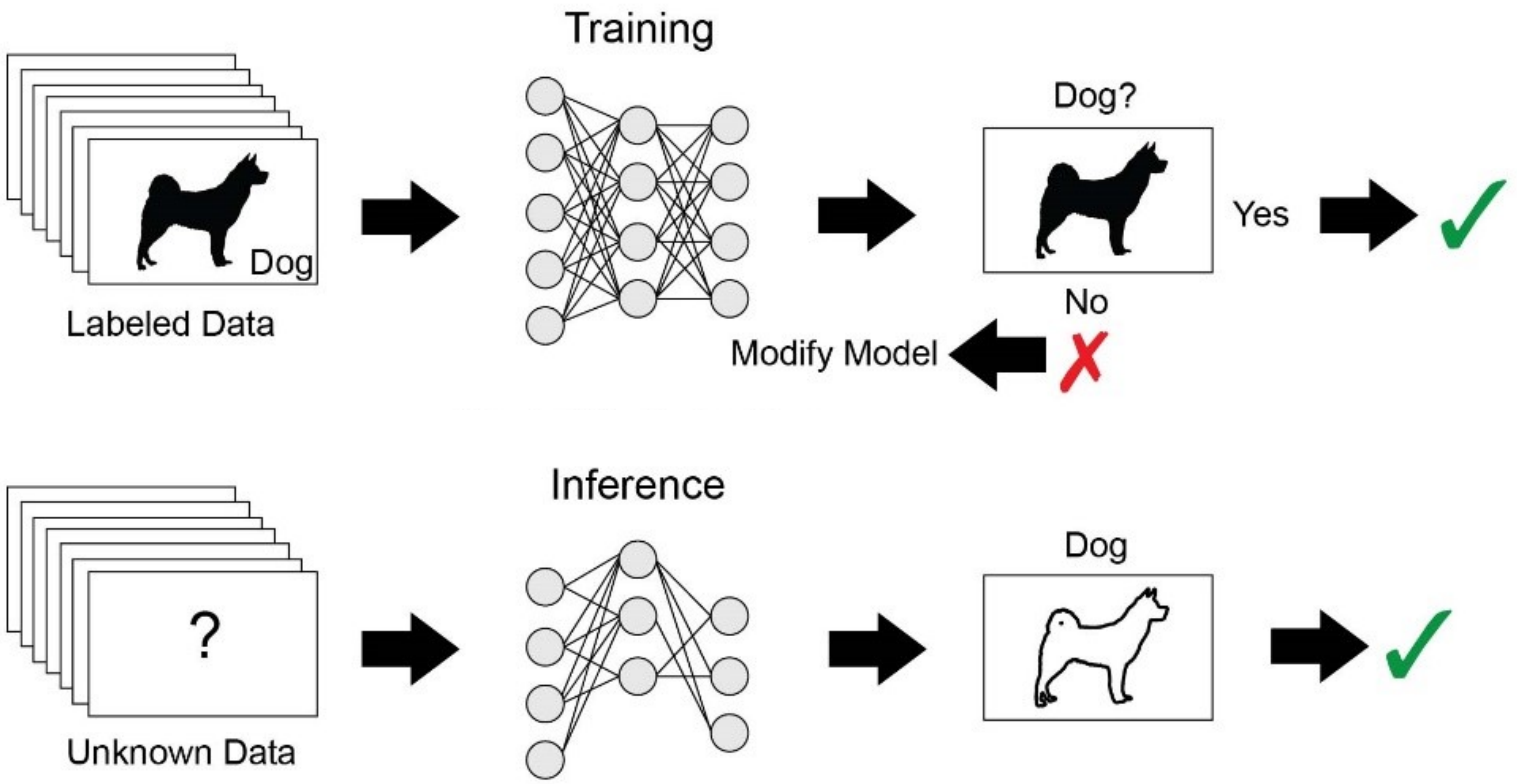


ML systems bridge model, data, and hardware

ImageNet, Kaggle, Flickr, NetFlix, …

GPUs, TPUs, Supercomputers, FPGAS

NOTE ON TRAINING VS INFERENCE



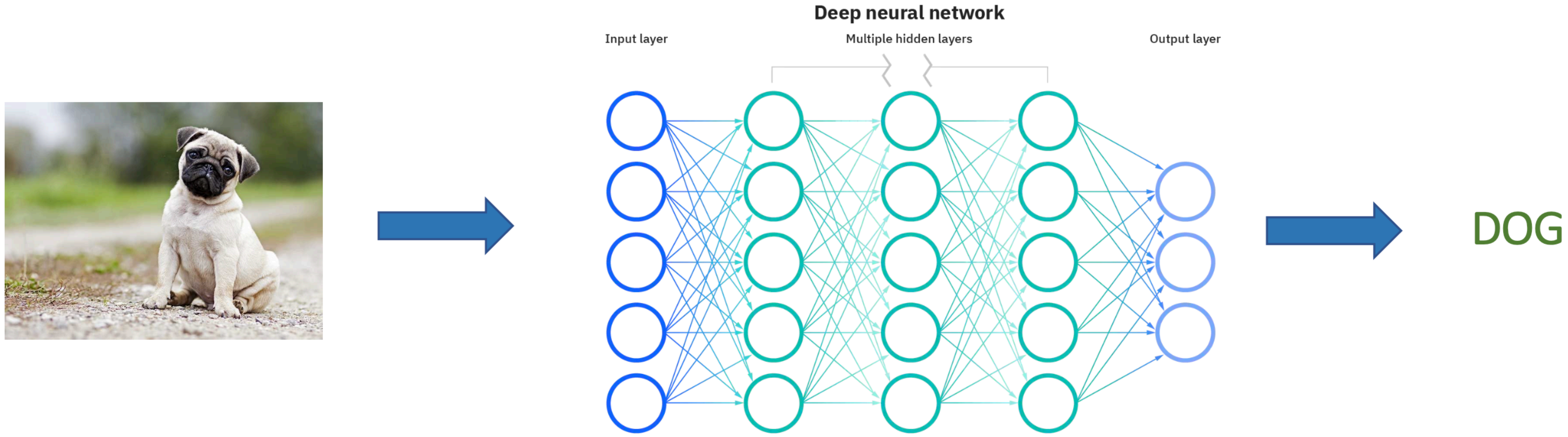
In today’s discussion, will focus on training!

Will briefly discuss inference later!

MACHINE LEARNING TRAINING PIPELINE

– 1. Users select a model architecture!

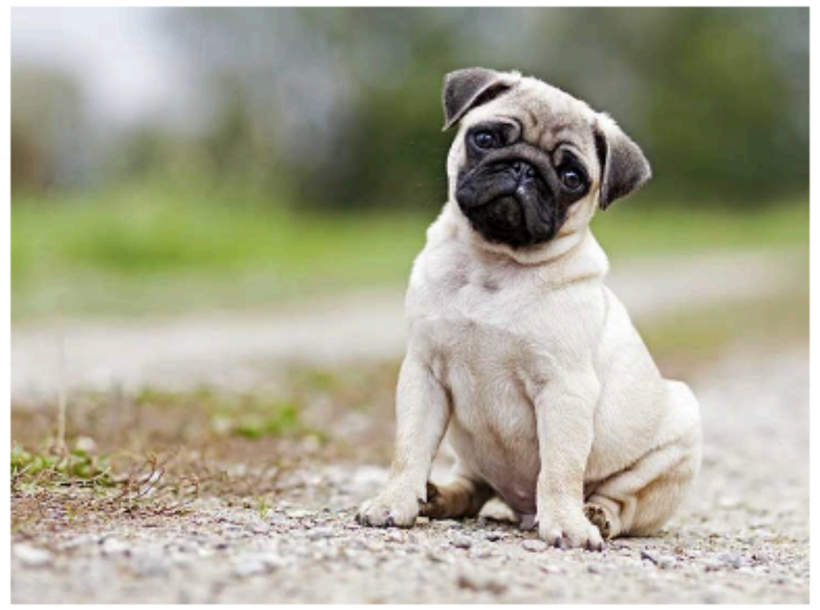
  – Typically Deep Neural Networks (DNNs)
  – Others types/variants: Recurrent Neural Networks, Graph Neural Networks, etc.



MACHINE LEARNING TRAINING PIPELINE

- 2. Users provide a large labeled dataset
  - images + classification labels
  - images + captions
  - sentence + sentiment analysis

DOG



CAT

DOG

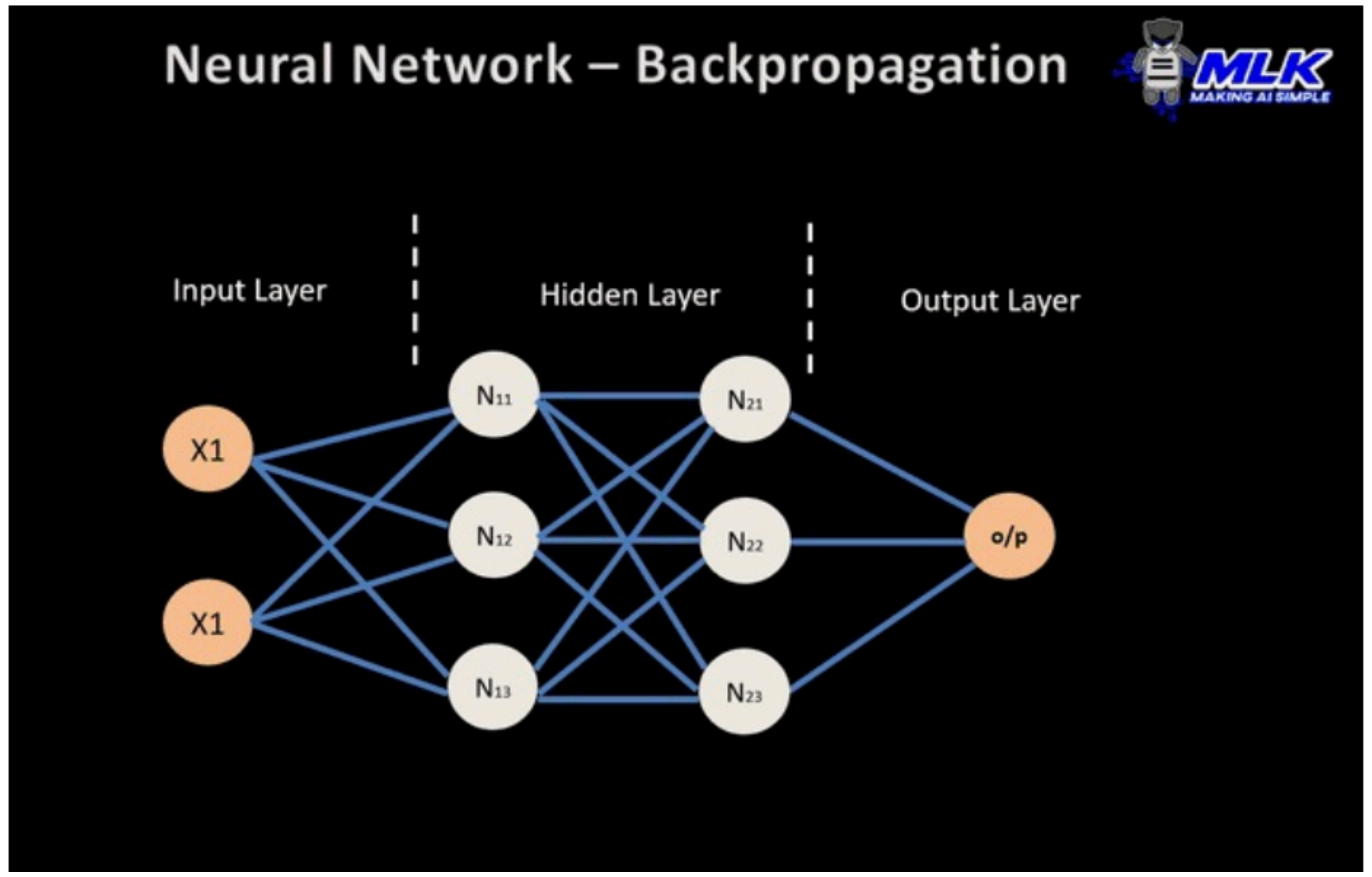
CAT

MACHINE LEARNING TRAINING PIPELINE

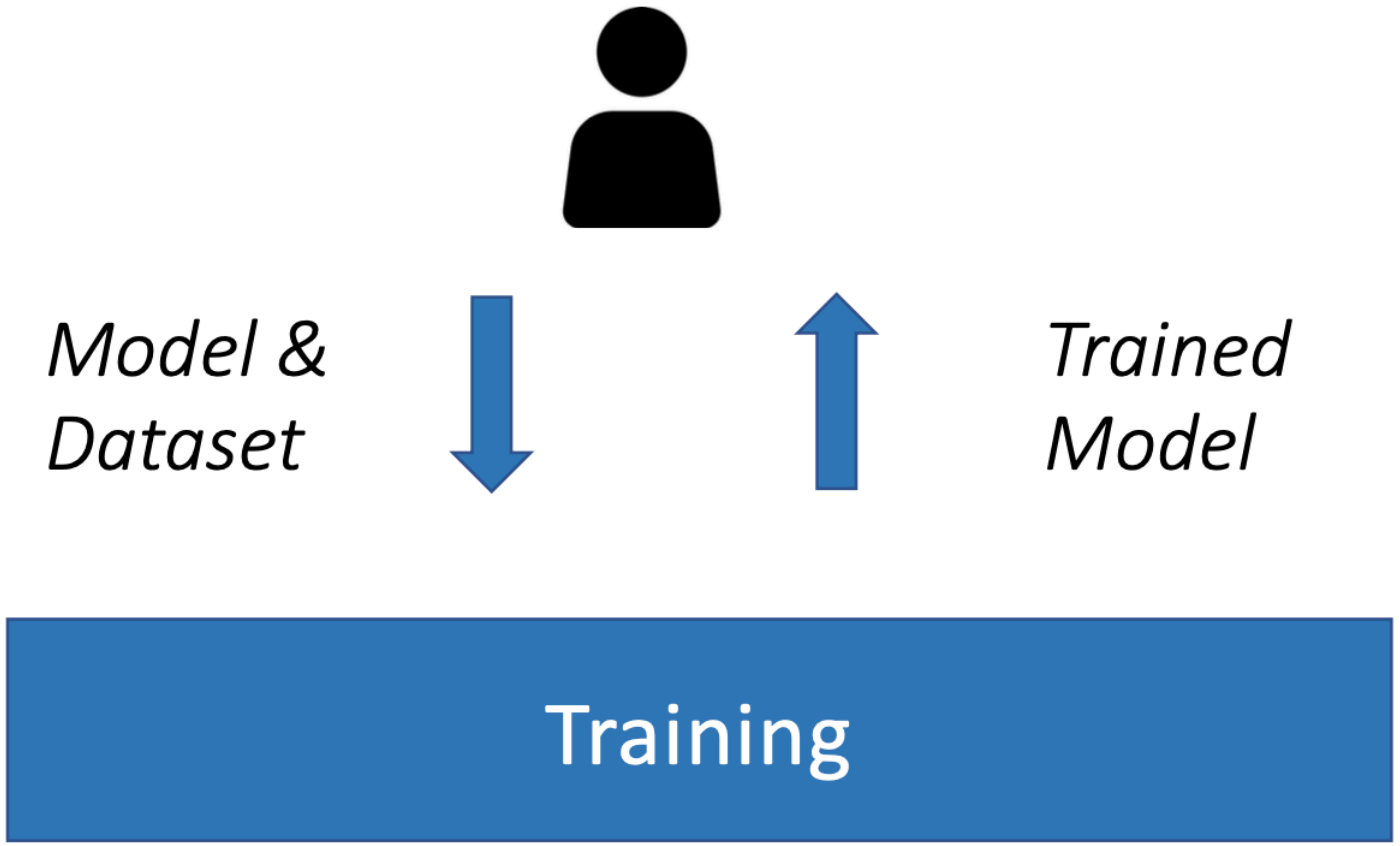
– 3. Train the model!

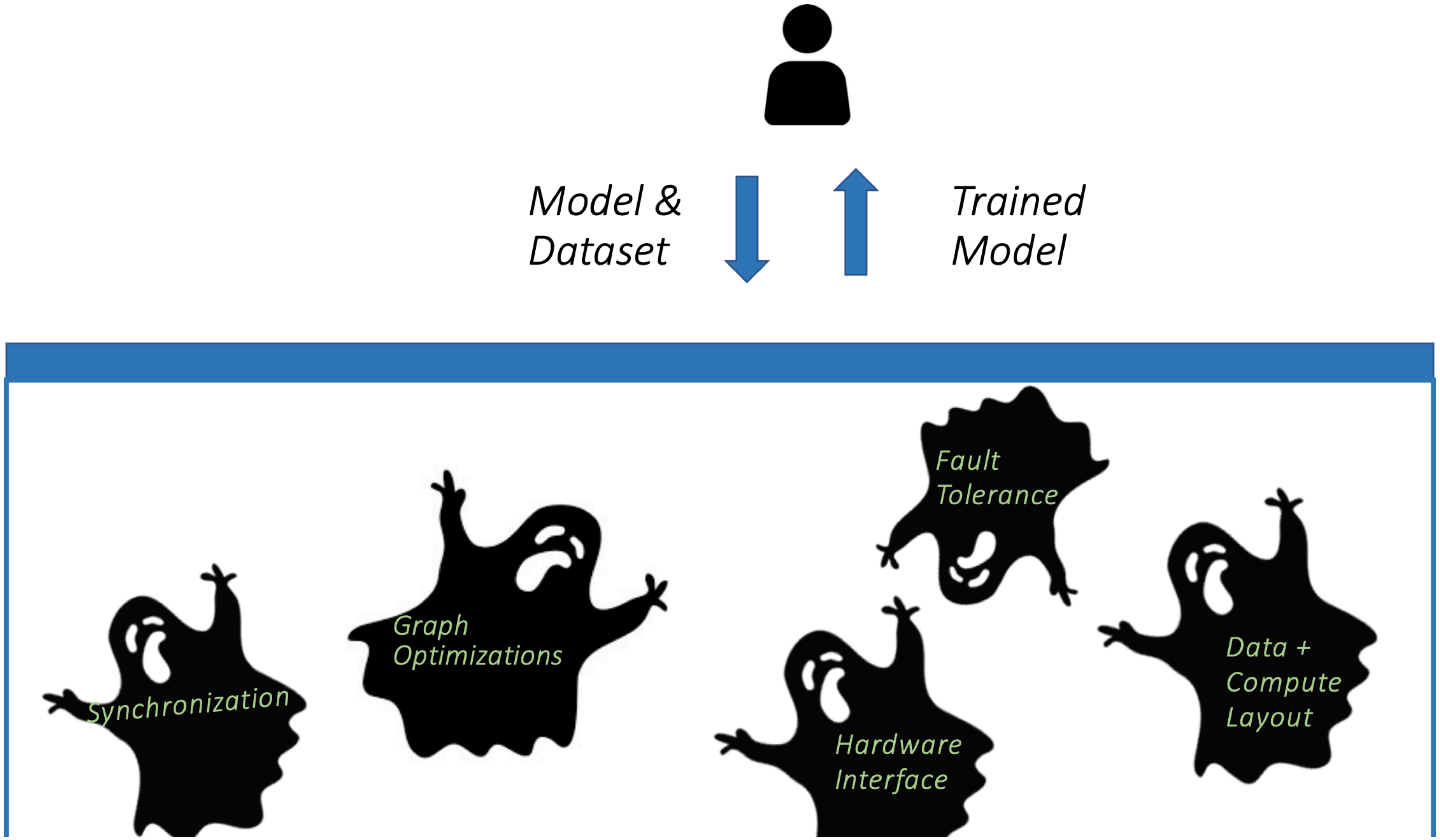
  – sequentially process the dataset

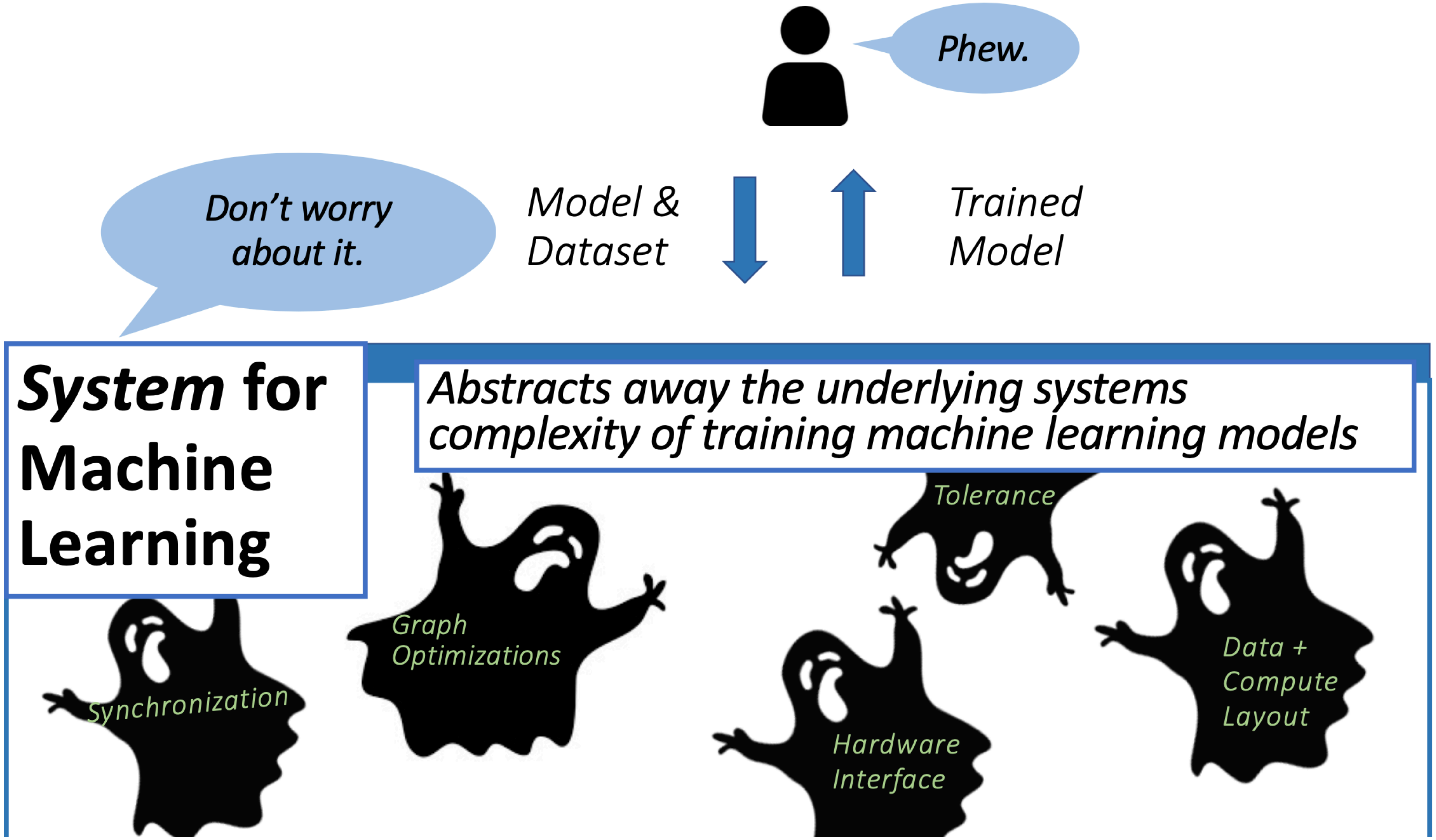
  – learn using a form of gradient descent (via backpropagation)



MACHINE LEARNING TRAINING PIPELINE




Model & Dataset
Trained Model
Synchronization
Graph Optimizations
Fault Tolerance
Hardware Interface
Data + Compute Layout


Phew.
Don't worry about it.
Model & Dataset
Trained Model
System for Machine Learning
Abstracts away the underlying systems complexity of training machine learning models
Tolerance
Graph Optimizations
Synchronization
Hardware Interface
Data + Compute Layout

SYSTEM FOR MACHINE LEARNING

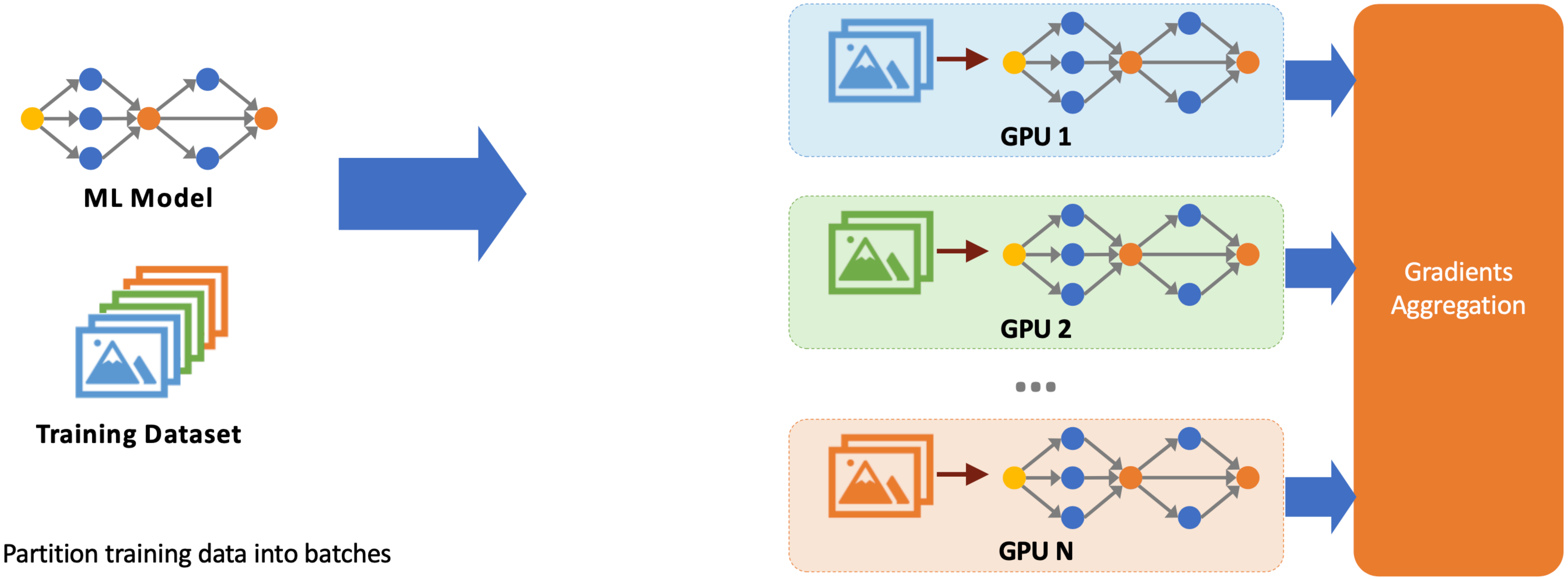
- Abstracts away the underlying systems complexities of executing the training of machine learning models
- Design Considerations
  - How to handle distributed computation?
  - How to support execution in different environments and on heterogeneous hardware?
  - What's the right interface for users that still supports customizations?

DESIGN CONSIDERATION #1: HANDLE DISTRIBUTED COMPUTATION

- Why perform distributed machine learning in the first place?
- Trends
  - Increasingly large datasets → Too slow to process on a single machine
    - millions/billions of images/samples
  - Increasingly large DNNs → The entirety of a DNN (and its weights/gradients) cannot fit on a single machine!
    - more layers, more parameters
  - For example,
    - GPT-3 is a language model with about 175 billion parameters
    - Is trained on 45 Terabytes of text data

DISTRIBUTED ML: DATA PARALLELISM

Challenge: All the workers must communicate with the centralized server for weight updates.



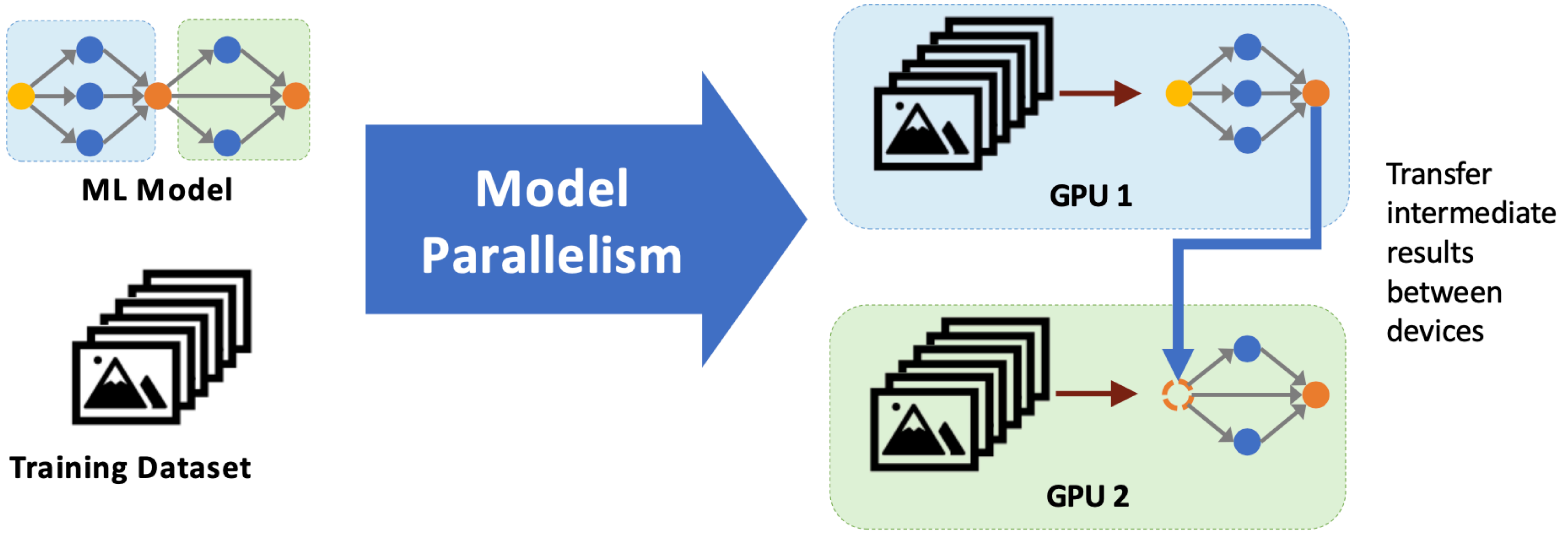
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

DISTRIBUTED ML: MODEL PARALLELISM

– Split a model into multiple subgraphs and assign them to different devices



Challenge: How split model across machines?

DISTRIBUTED ML CONSIDERATIONS

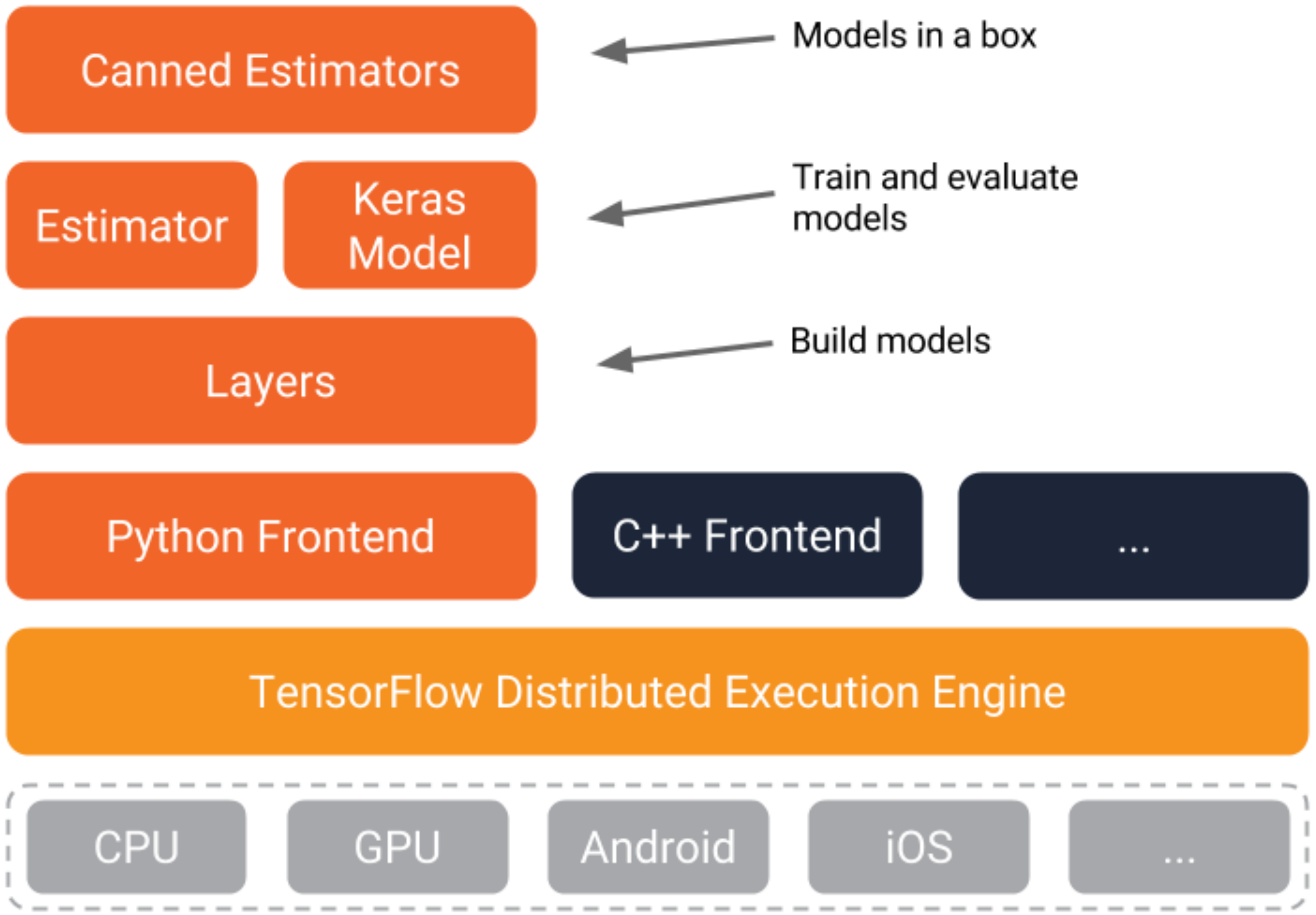
- Placement of computation across machines
  - Locality
- Communication of intermediate data between machines
  - efficient collective communication (AllReduce, etc.), pipeline
- Fault tolerance! What happens if a machine crashes?
  - checkpoints
- Synchronization
  - relaxed synchronization model (async, etc.)

DESIGN CONSIDERATION #2: SUPPORT HETEROGENEOUS ENV?

- Various types of compute settings:
  - datacenter (thousands of CPUs, GPUs)
  - workstation set up (single CPU, few GPUs)
  - laptop
- Heterogeneous Hardware: GPUs, TPUs, FPGAs
  - Each is optimized for different tasks
  - Optimal memory placement/computation configuration depends on type

Define Once + Run Everywhere

DESIGN CONSIDERATION #2: SUPPORT HETEROGENEOUS ENV?



DESIGN CONSIDERATION #3: INTERFACE FOR CUSTOMIZATIONS

- Support different user requirements
  - novice user: uses several default settings
  - expert user:
    - define new layers
    - try new training algorithms
    - introduce new optimizations
- Want easy-to-use interface while still being customizable

Case Study: TensorFlow

TENSORFLOW

- Developed by Google Brain
  - successor to DistBelief
- A system widely used in industry/academia for distributed machine learning
  - (updated in 2025): a trend of shifting to alternatives such as PyTorch or JAX
- Main Contributions
  - Support for large-scale distributed training
  - Modular architecture that decouples optimizations of the machine learning model from the infrastructure itself
    - supports diverse compute environments, heterogeneous hardware
  - User-friendly: Python interface that enables customizability across the stack

TENSORFLOW: EXAMPLE

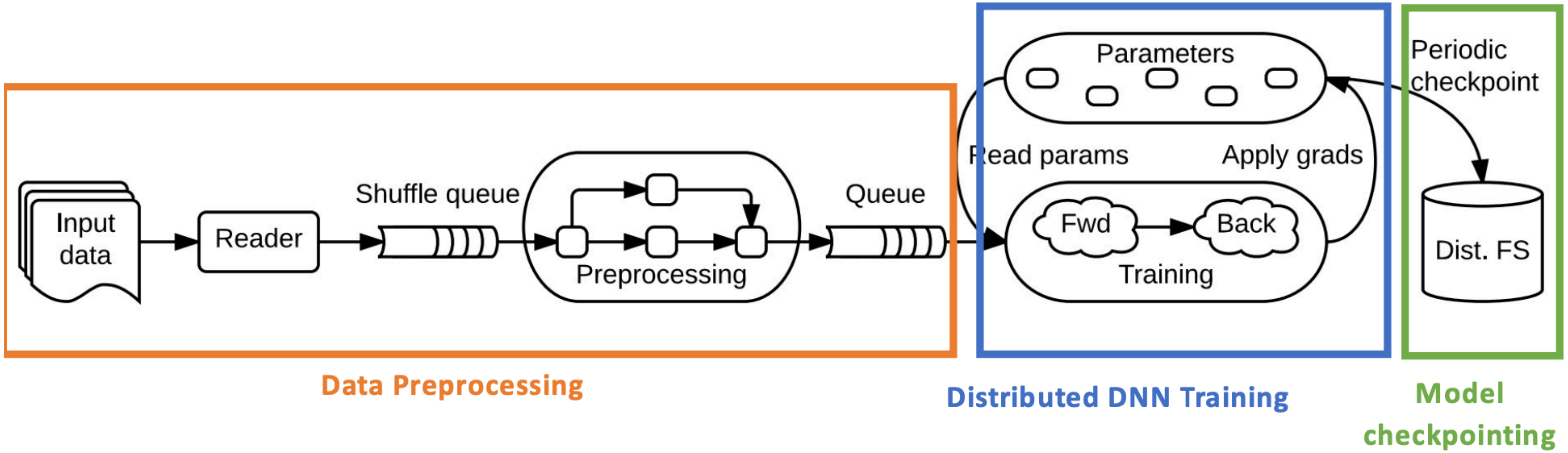
Phase 1: Define an ML model as a dataflow graph

```
# 1. Construct a graph representing the model.
x = tf.placeholder(tf.float32, [BATCH_SIZE, 784])  # Placeholder for input.
y = tf.placeholder(tf.float32, [BATCH_SIZE, 10])   # Placeholder for labels.

W_1 = tf.Variable(tf.random_uniform([784, 100]))   # 784x100 weight matrix.
b_1 = tf.Variable(tf.zeros([100]))                 # 100-element bias vector.
layer_1 = tf.nn.relu(tf.matmul(x, W_1) + b_2)      # Output of hidden layer.

W_2 = tf.Variable(tf.random_uniform([100, 10]))    # 100x10 weight matrix.
b_2 = tf.Variable(tf.zeros([10]))                  # 10-element bias vector.
layer_2 = tf.matmul(layer_1, W_2) + b_2            # Output of linear layer.

# 2. Add nodes that represent the optimization algorithm.
loss = tf.nn.softmax_cross_entropy_with_logits(layer_2, y)
train_op = tf.train.AdagradOptimizer(0.01).minimize(loss)
```

Phase 2: Execute an optimized version of the graph

```
# 3. Execute the graph on batches of input data.
with tf.Session() as sess:                         # Connect to the TF runtime.
  sess.run(tf.initialize_all_variables())          # Randomly initialize weights.
  for step in range(NUM_STEPS):                    # Train iteratively for NUM_STEPS.
    x_data, y_data = ...                           # Load one batch of input data.
    sess.run(train_op, {x: x_data, y: y_data})     # Perform one training step.
```

TENSORFLOW SYSTEM DESIGN



DISTRIBUTED EXECUTION

- Each operation resides on a device in a particular task
- A device is responsible for executing a kernel for each operation assigned to it
- The placement algorithm places operation on device subject to constraints in the graph
- Once operation is places on a device, TensorFlow partitions the operations into per-device subgraphs
- TensorFlow is optimized for executing large subgraphs repeatedly with low latency

FAULT TOLERANCE

- Fault tolerance: long-running jobs are likely to experience failure or pre-emption without adding too much overhead since failure might be rare
- Client library allows to construct appropriate graph structure and use save and restore for user-level checkpointing for fault tolerance
- This is customizable so the user can implement it as necessary and apply different checkpoints for different subsets of the graph

(A)SYNCHRONY

– Synchronous replica coordination: originally designed for asynchronous training, but have been experimenting with synchronous methods.

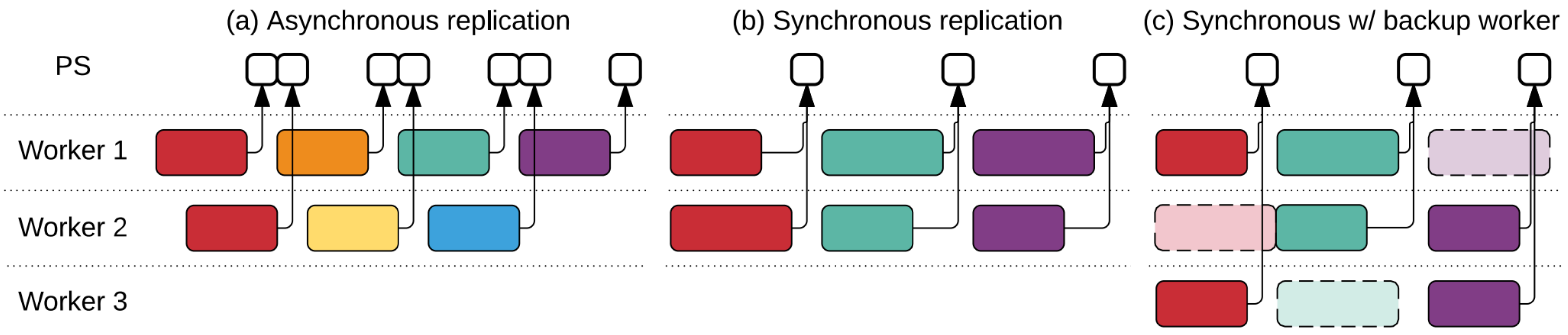


Figure 5: Three synchronization schemes for parallel SGD. Each color represents a different starting parameter value; a white square is a parameter update. In (c), a dashed rectangle represents a backup worker whose result is discarded.

SYSTEMS FOR MACHINE LEARNING INFERENCE

- Application/customer facing: stringent latency targets
- Deal with interactions with network
- Caching opportunities
- Model compression/pruning
  - tradeoff between speed and accuracy
- Edge deployments

ACTIVE RESEARCH AREAS IN ML+SYSTEMS

— Application-specific optimizations for machine learning (e.g., video analytics)

— ML for systems (e.g., learned databases, compilation optimizations)

— New computation models (spot instances, serverless computing, programmable networks)

University of Virginia

TAKEAWAYS

- Systems for machine learning are critical to the success of machine learning
- Handle the systems challenges involved in running large-scale distributed machine learning
  - e.g., fault tolerance, consistency, heterogeneous hardware, communication
- Provide an easy-to-use interface for developers while still enabling significant levels of customizability
- Next class: **Reliability**

ACKNOWLEDGEMENT

THIS COURSE IS DEVELOPED HEAVILY BASED ON COURSE MATERIALS SHARED BY PROF. INDRANIL GUPTA, PROF. ROBERT MORRIS, PROF. MICHAEL FREEDMAN, PROF. KYLE JAMIESON, PROF. WYATT LLOYD AND PROF. ROXANA GEAMBASU. MANY APPRECIATIONS FOR GENEROUSLY SHARING THEIR MATERIALS AND TEACHING INSIGHTS.