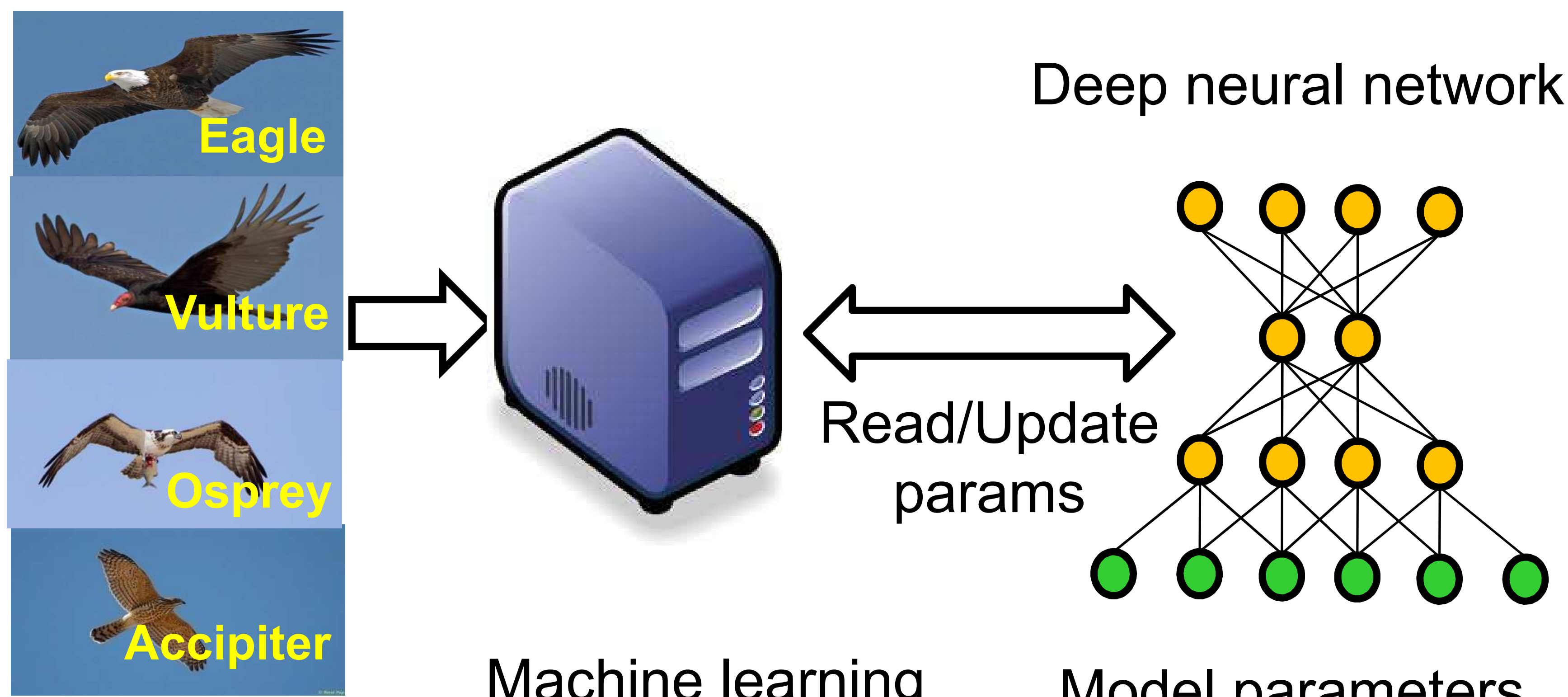
Exploiting Application Characteristics for Efficient System Support of Data-parallel Machine Learning

Henggang Cui

PARALLEL DATA LABORATORY
Carnegie Mellon University

Machine learning

Labelled images

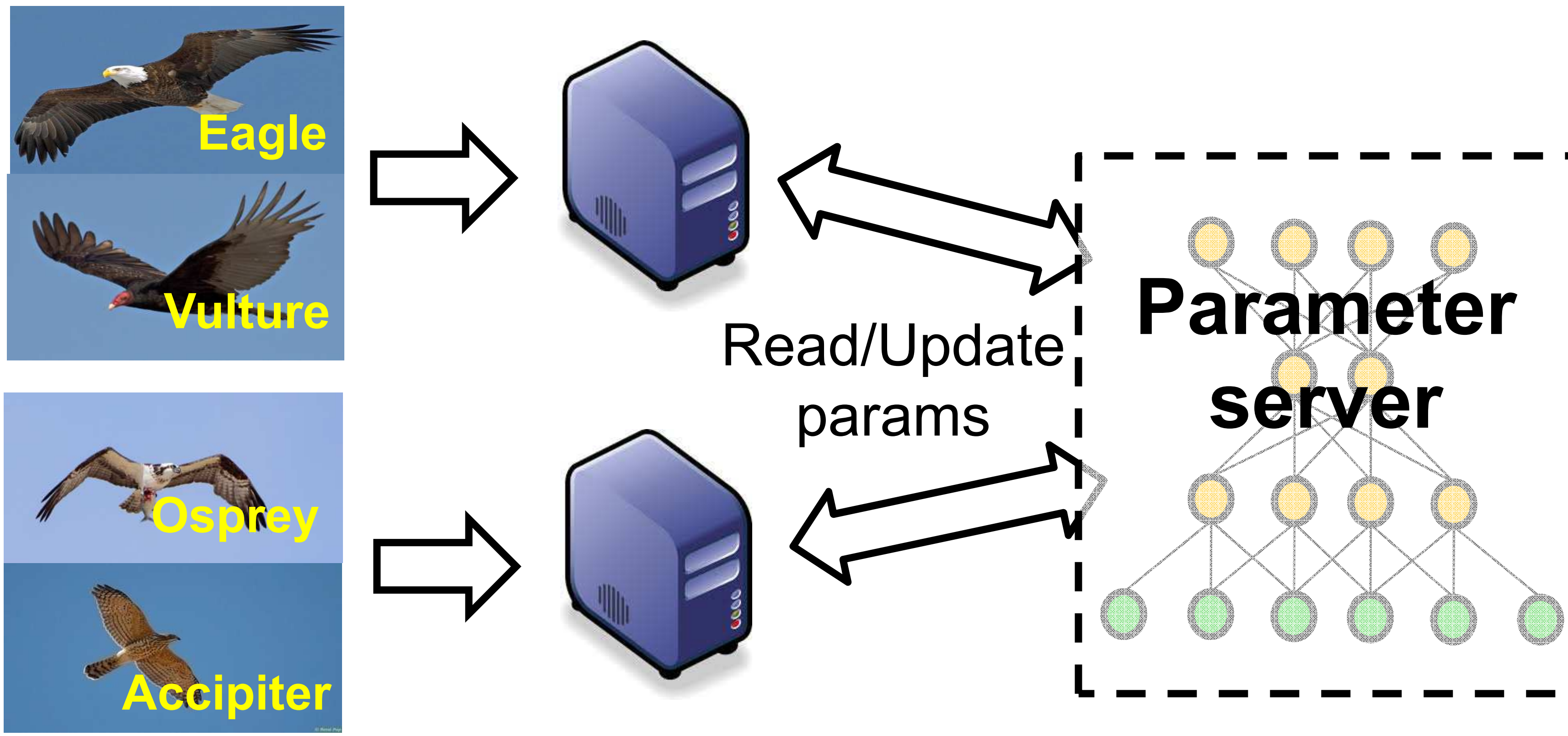


Training data

Machine learning program

Model parameters (solution)

Data-parallel machine learning



Partitioned
Training data

Distributed
ML workers

Shared
model parameters

Thesis statement

- *The characteristics of large-scale data-parallel machine learning computations can be exploited in the implementation of a parameter server to increase their efficiency by an order of magnitude or more.*

Three case studies

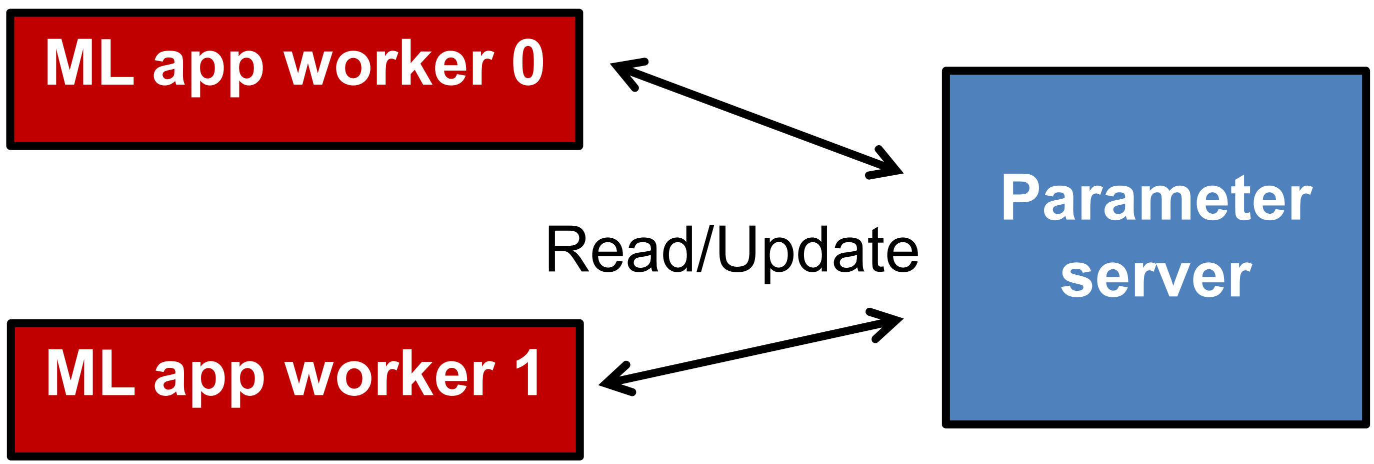
- IterStore [Cui et al. SoCC '14]
  - an efficient parameter server design
  - exploits repeated parameter data access
- GeePS [Cui et al. EuroSys '16]
  - specialized parameter server for GPU deep learning
  - exploits layer-by-layer pattern of deep learning
- MLtuner [In preparation]
  - system for automatic machine learning tuning
  - exploits quick decision of training hyperparam tuning

Three case studies

- IterStore [Cui et al. SoCC '14]
  - an efficient parameter server design
  - exploits repeated parameter data access
- GeePS [Cui et al. EuroSys '16]
  - specialized parameter server for GPU deep learning
  - exploits layer-by-layer pattern of deep learning
- MLtuner [In preparation]
  - system for automatic machine learning tuning
  - exploits quick decision of training hyperparam tuning

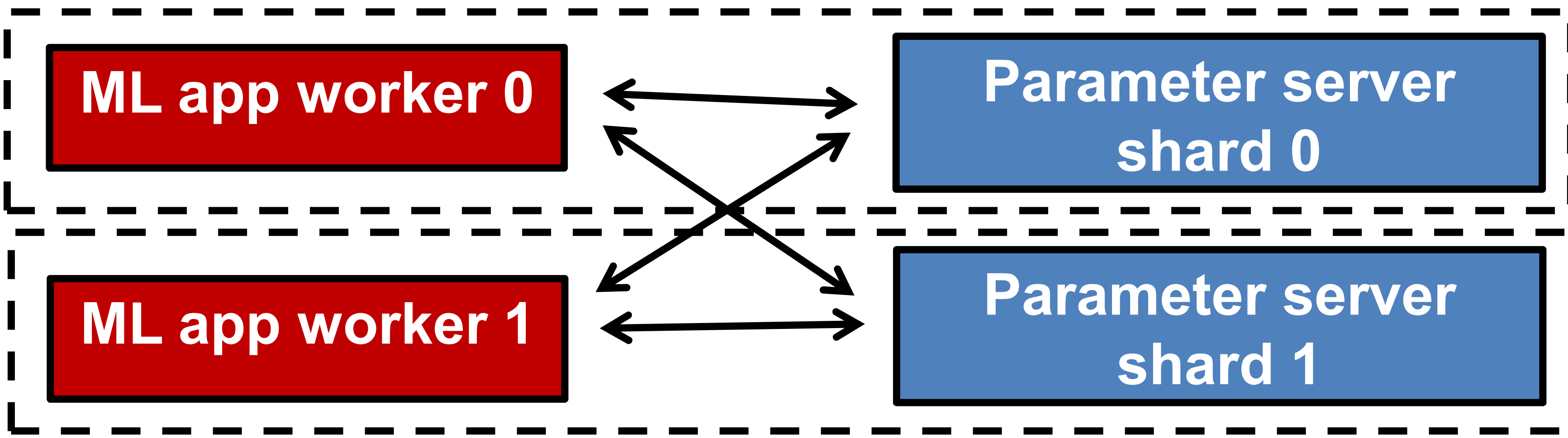
Traditional parameter server design

- Traditional PS is like a generic key-value store
  - data organized as a collection of key-value pairs
    - accessed with Read and Update interface



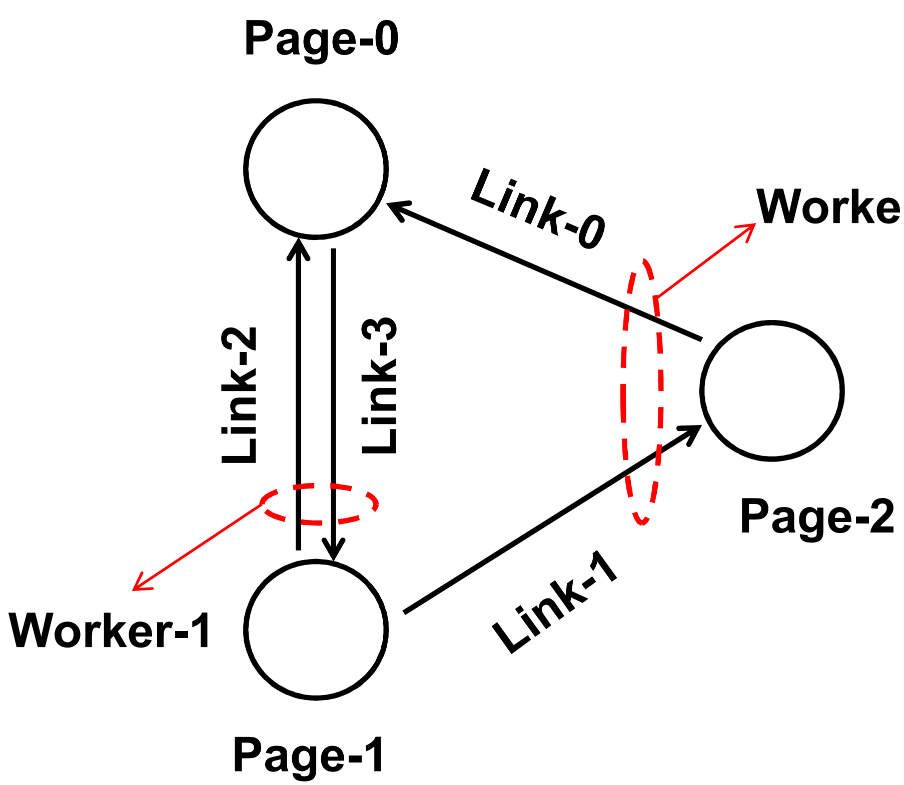
Traditional parameter server design

- Traditional PS is like a generic key-value store
  - data organized as a collection of key-value pairs
    - accessed with Read and Update interface
  - sharded distributedly and co-located with ML workers
  - **assumes no knowledge of the access pattern**



Repeated data access in ML applications

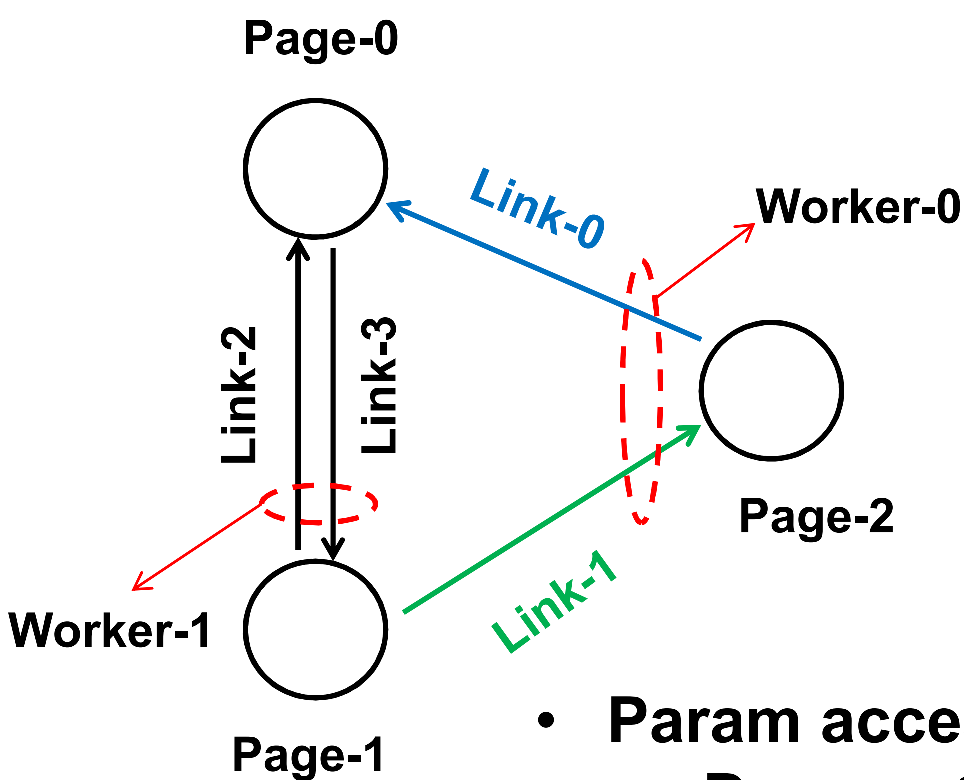
Example application: PageRank
Parameter data: ranks of pages, stored in parameter server




```
Init ranks to random value
LOOP
  FOREACH link from i to j
    Read rank[i]
    rank[j] += change of rank[i]
  ENDFOREACH
WHILE NOT CONVERGE
```

Repeated data access in ML applications

Example application: PageRank
Parameter data: ranks of pages, stored in parameter server



Worker-0:

```
LOOP
  # Link-0
  Read rank[2]
  Update rank[0]
  # Link-1
  Read rank[1]
  Update rank[2]
  Clock
WHILE NOT CONVERGE
```

- **Param access depends only on the model**
  - **Does not depend on param values**

Repeated data access sequence

- Many examples of ML applications
  - including deep learning, matrix factorization and LDA
- Knowledge of repeated access sequence can be exploited to improve efficiency

Obtain per-iter sequence via a *virtual* iteration

```
// Original
LoadTrainingData()
do {
  DoIteration()
} while (not stop)
```

Obtain per-iter sequence via a *virtual* iteration

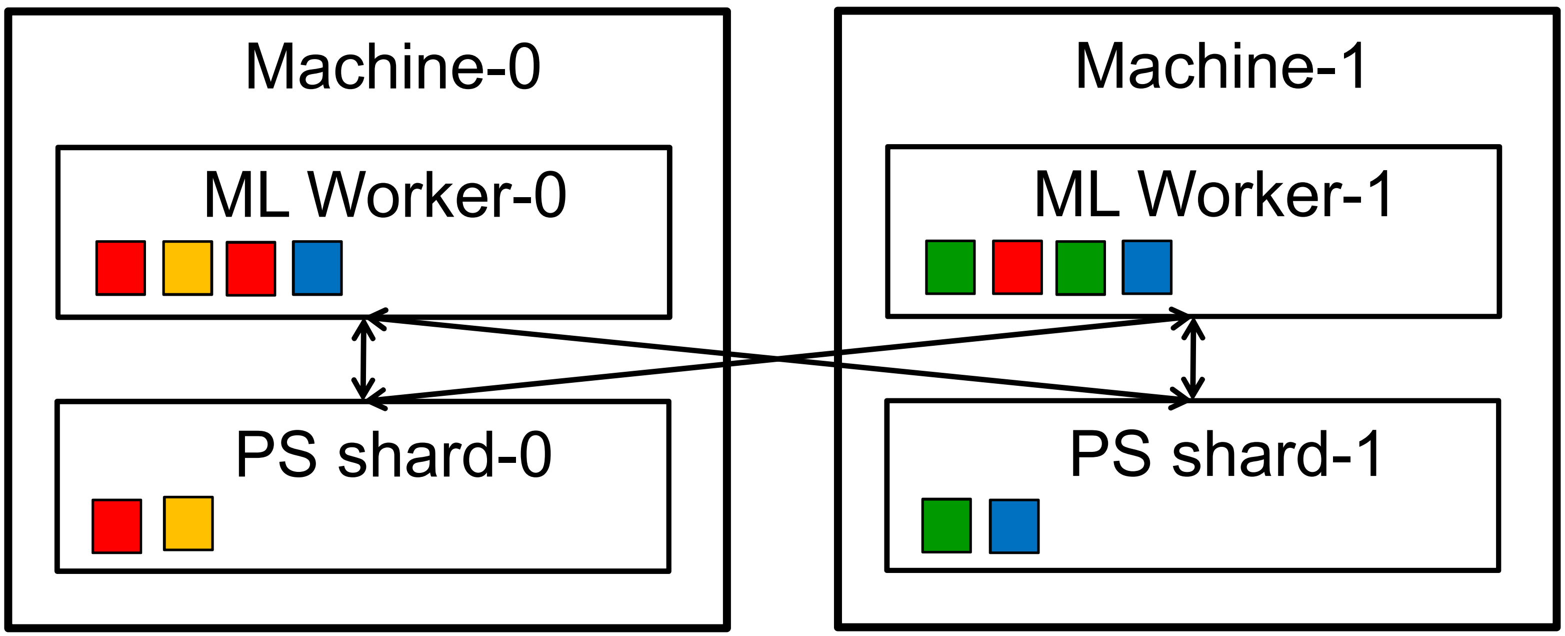
```
// Original
LoadTrainingData()
do {
  DoIteration()
} while (not stop)
```

```
// With virtual iteration
LoadTrainingData()
ps.StartVirtualIter()
DoIteration()
ps.FinishVirtualIter()
do {
  DoIteration()
} while (not stop)
```

- Calls in virtual iter. are recorded with no action taken
  - almost no overhead

Optimizations on informed access

- Many optimizations applied after virtual iteration
  1. parameter data sharding with better locality

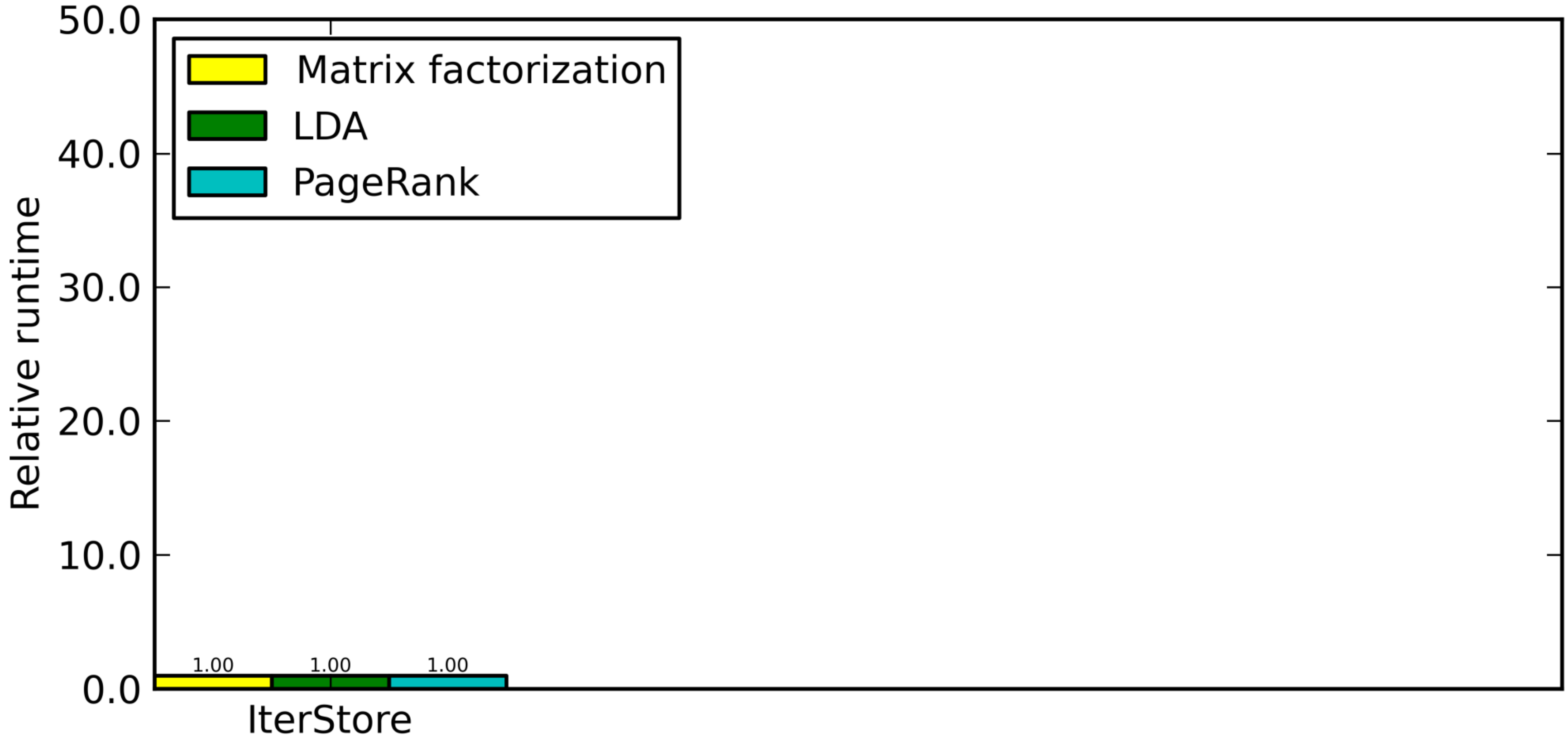


Optimizations on informed access

- Many optimizations applied after virtual iteration
  1. parameter sharding with better locality
  2. prefetching
  3. specialized caching policies
  4. efficient marshalling-free data structures
  5. NUMA-aware memory arrangement

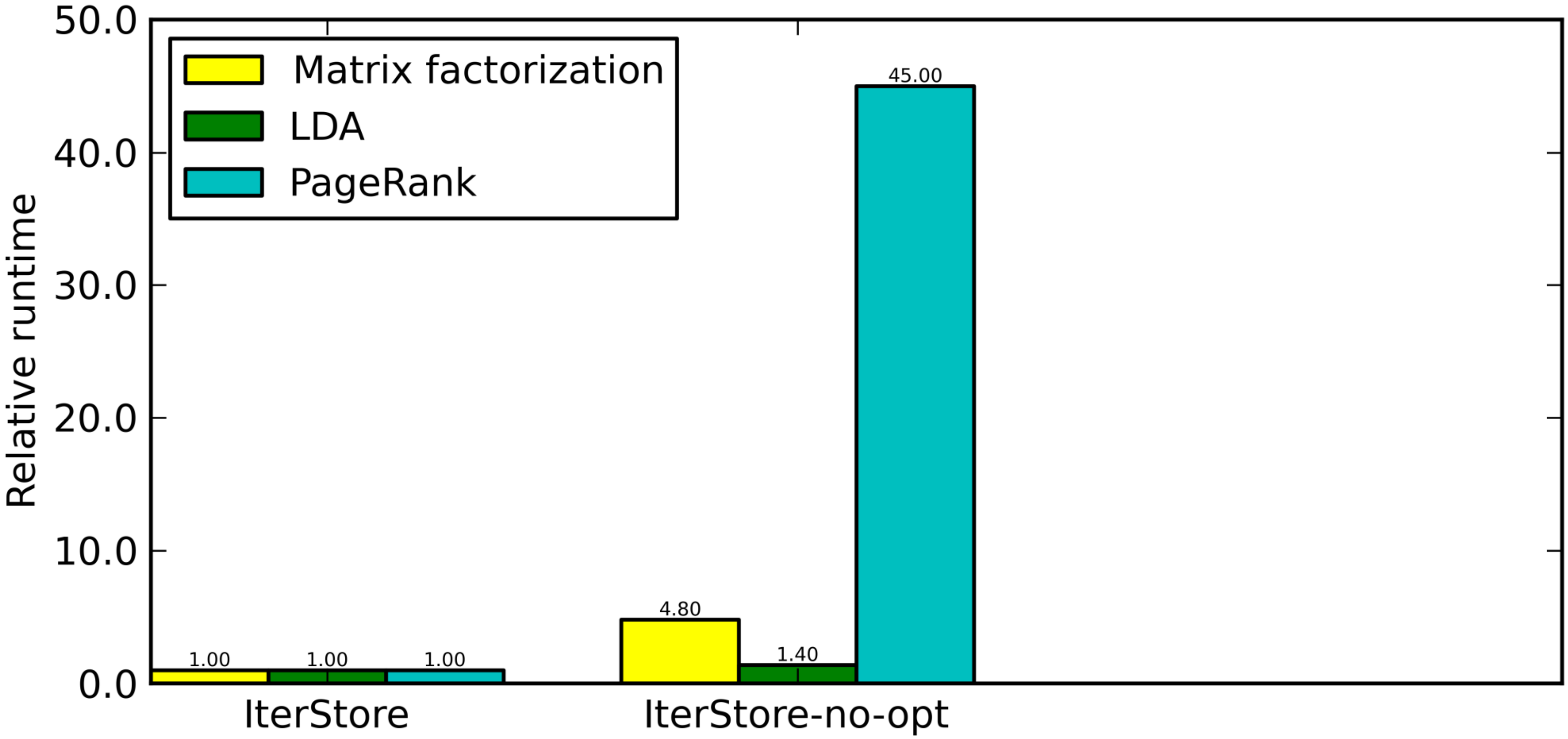
Henggang Cui © April 17

IterStore optimization speedups



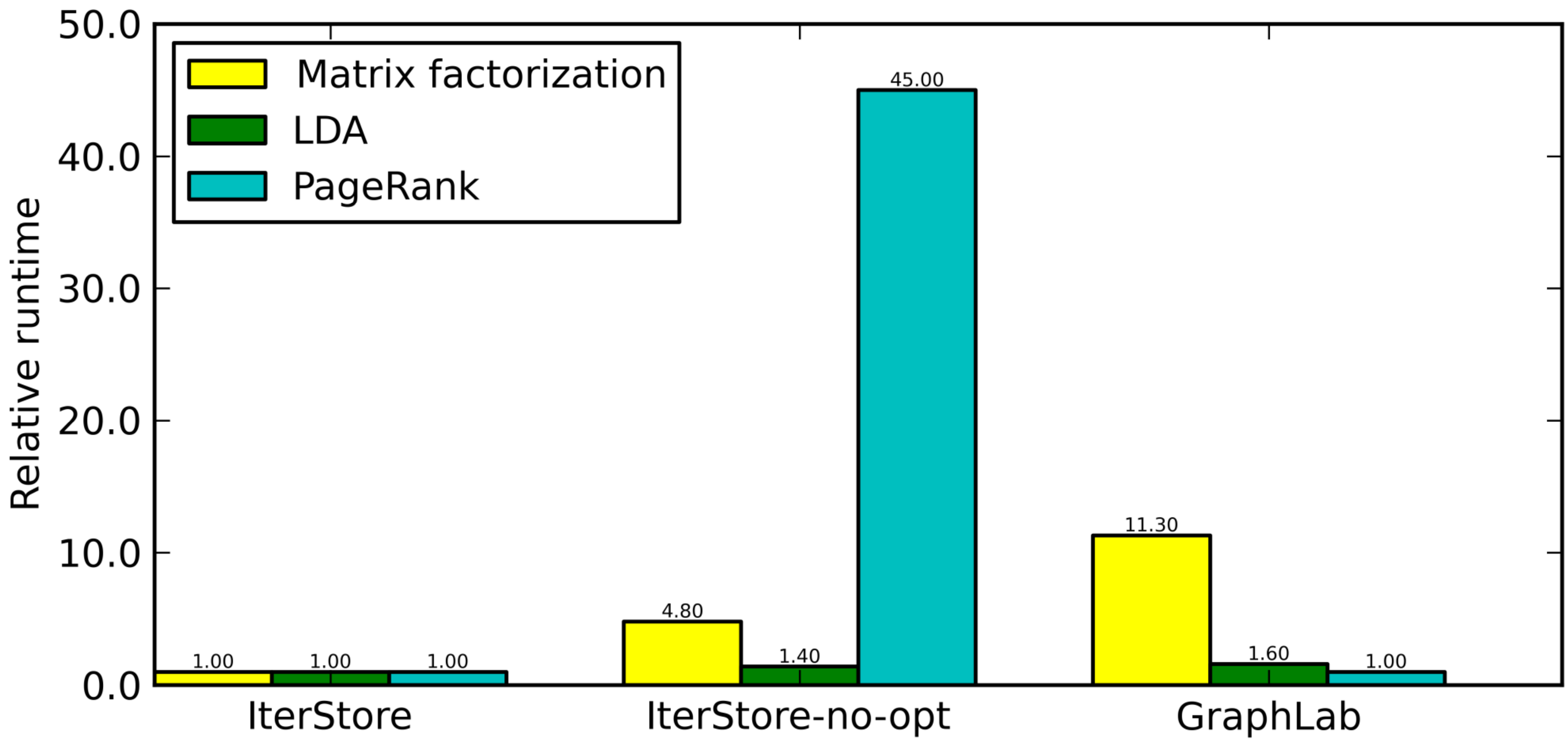
- **matrix factorization: Netflix dataset, rank 1000**
- **LDA: NYTimes dataset, 1000 topics**
- **PageRank: Twitter dataset**

IterStore optimization speedups



- **45x speedup on PageRank**
- **5x speedup on matrix factorization**

IterStore optimization speedups



- **faster than GraphLab (state-of-art at that time)**
  - **11x faster on matrix factorization**

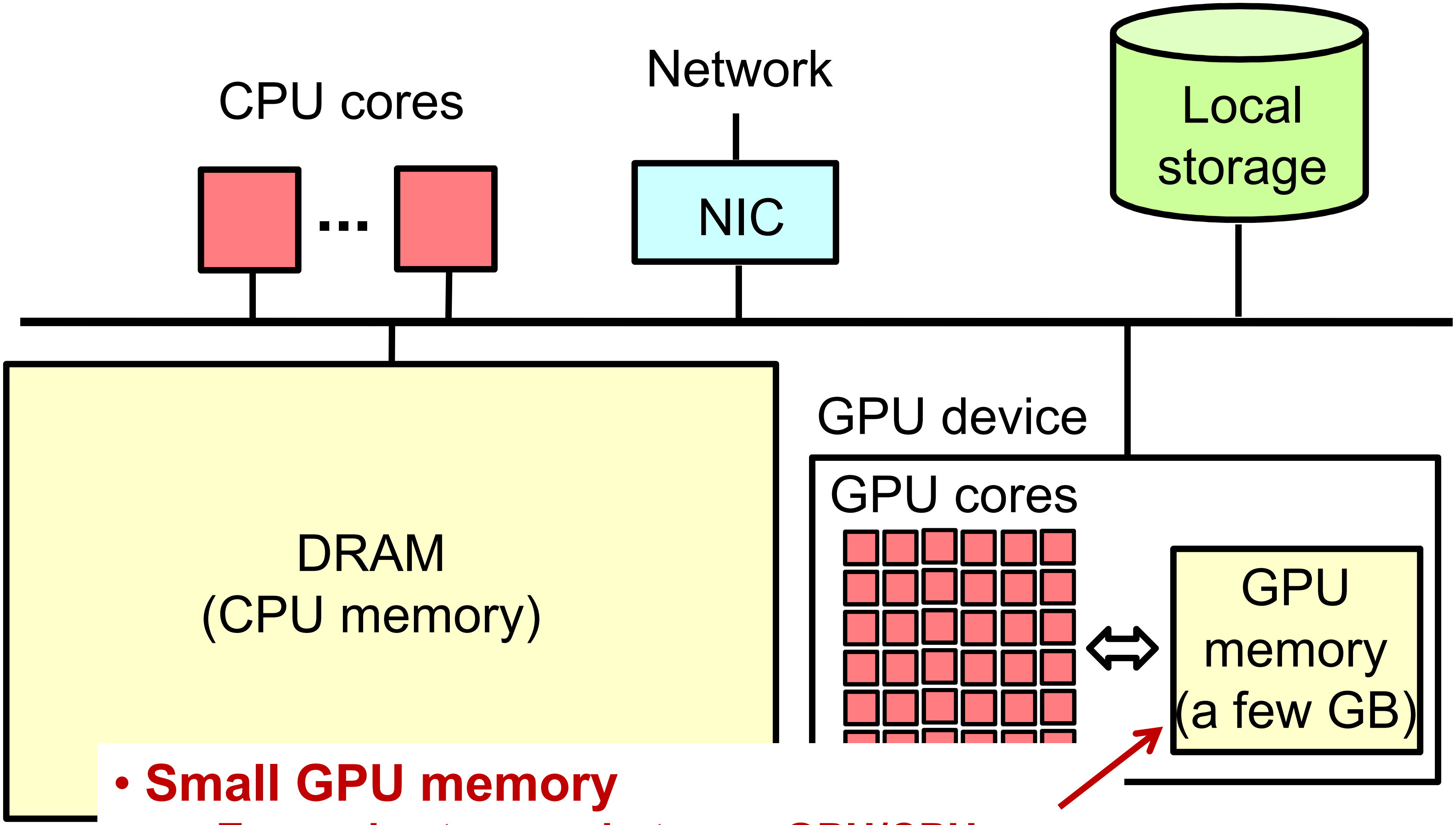
Take-away messages from IterStore

- Many ML applications exhibit iterativeness
  - same sequence of access every iteration
  - can be gathered via a virtual iteration
- Systems can exploit repeated access
  - speed up real ML benchmarks by up to 45x

Three case studies

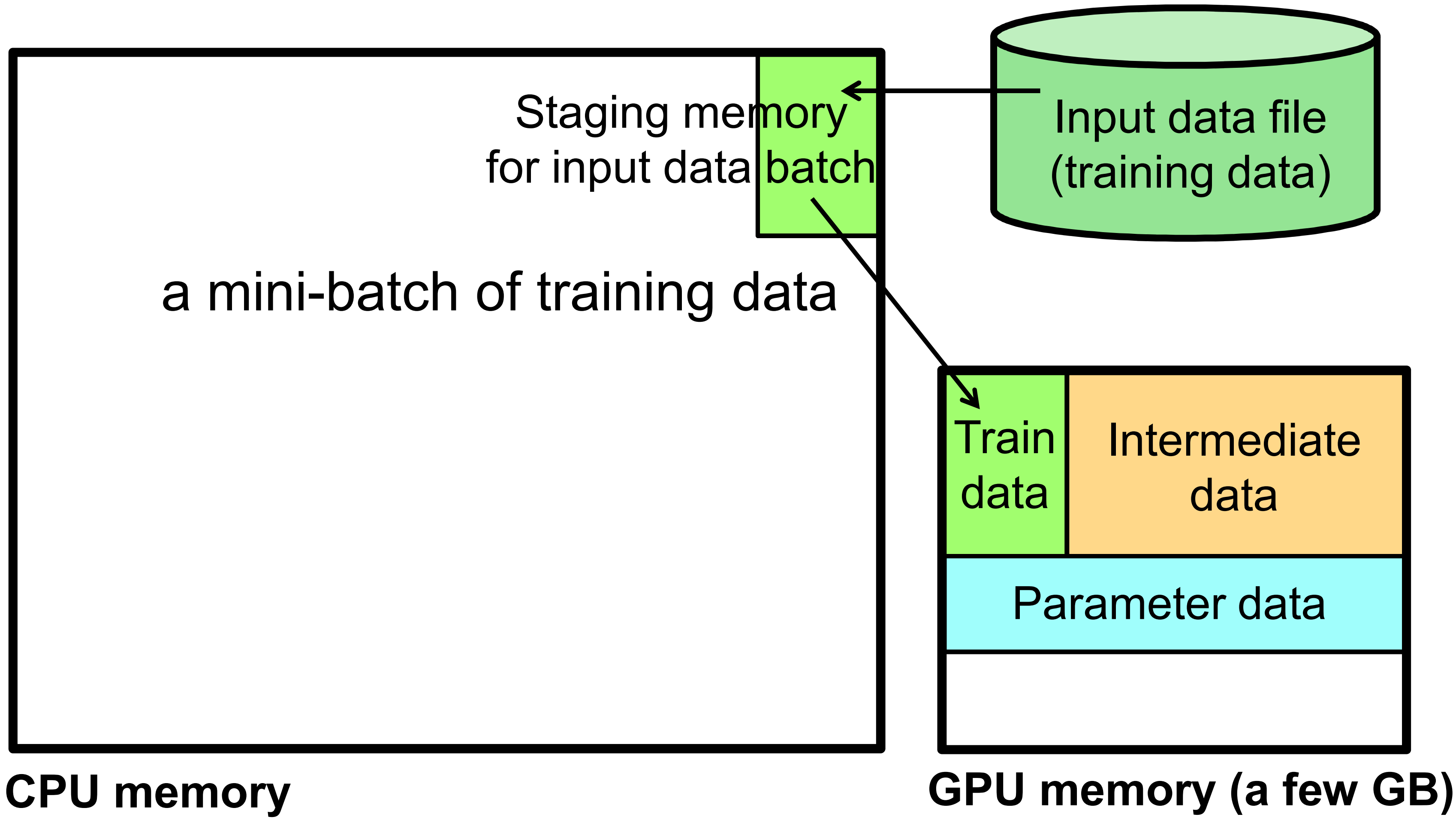
- IterStore [Cui et al. SoCC ‘14]
  - an efficient parameter server design
  - exploits repeated parameter data access
- GeePS [Cui et al. EuroSys ‘16]
  - specialized parameter server for GPU deep learning
  - exploits layer-by-layer pattern of deep learning
- MLtuner [In preparation]
  - system for automatic machine learning tuning
  - exploits quick decision of training hyperparam tuning

A machine with a GPU device

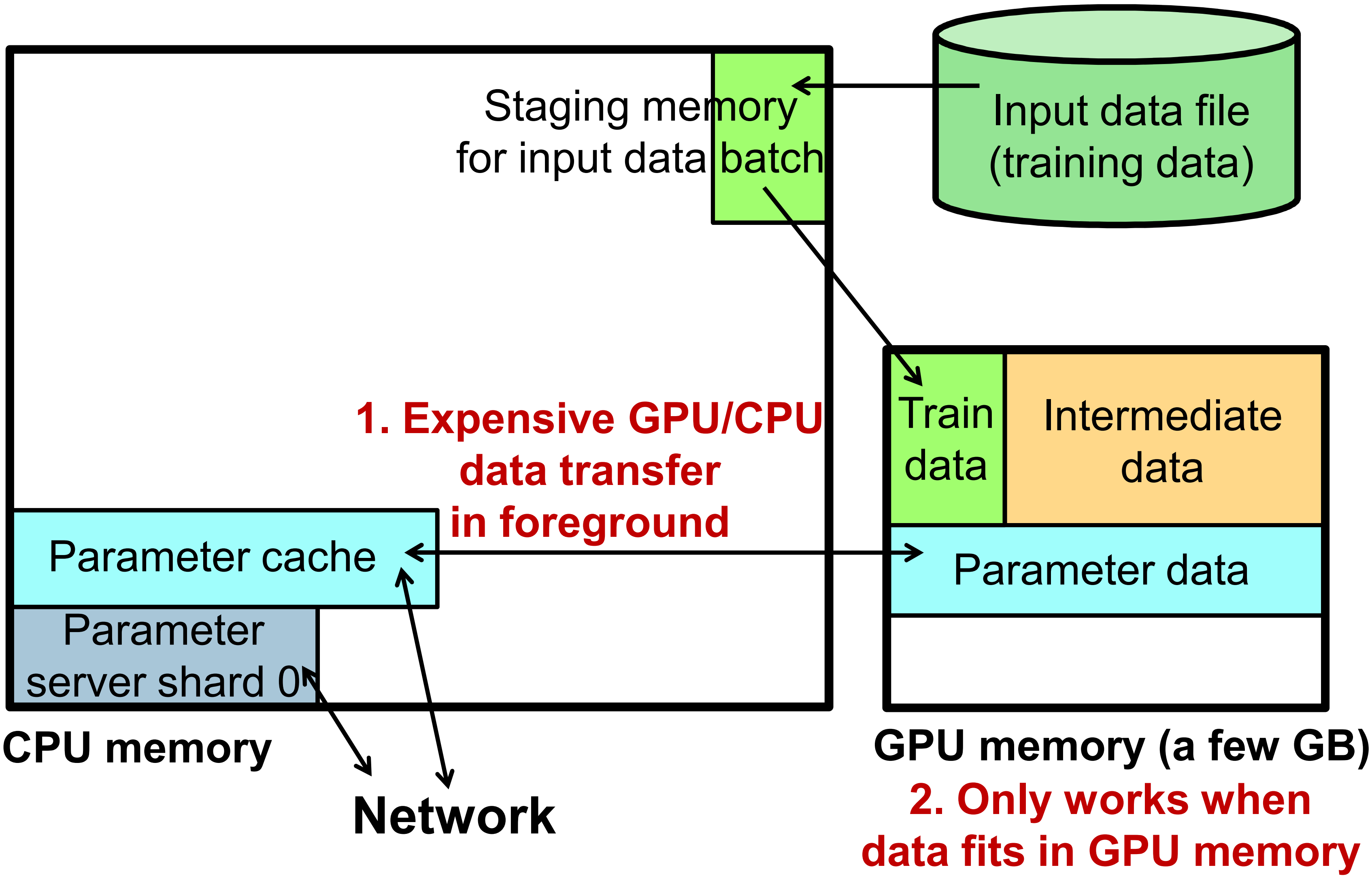


- **Small GPU memory**
  - **Expensive to copy between GPU/CPU mem**

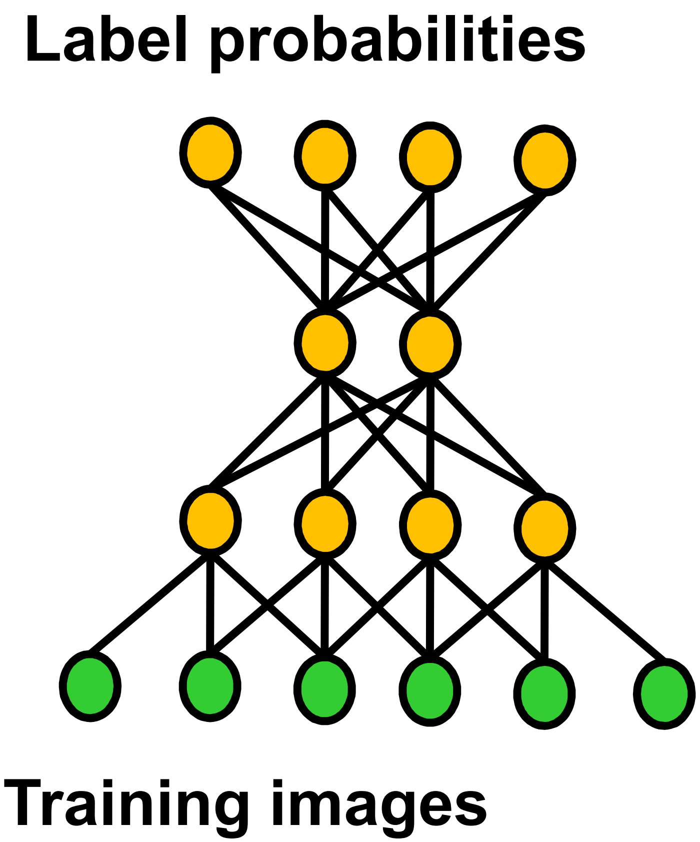
Single-GPU machine learning



Multi-GPU ML via CPU parameter server

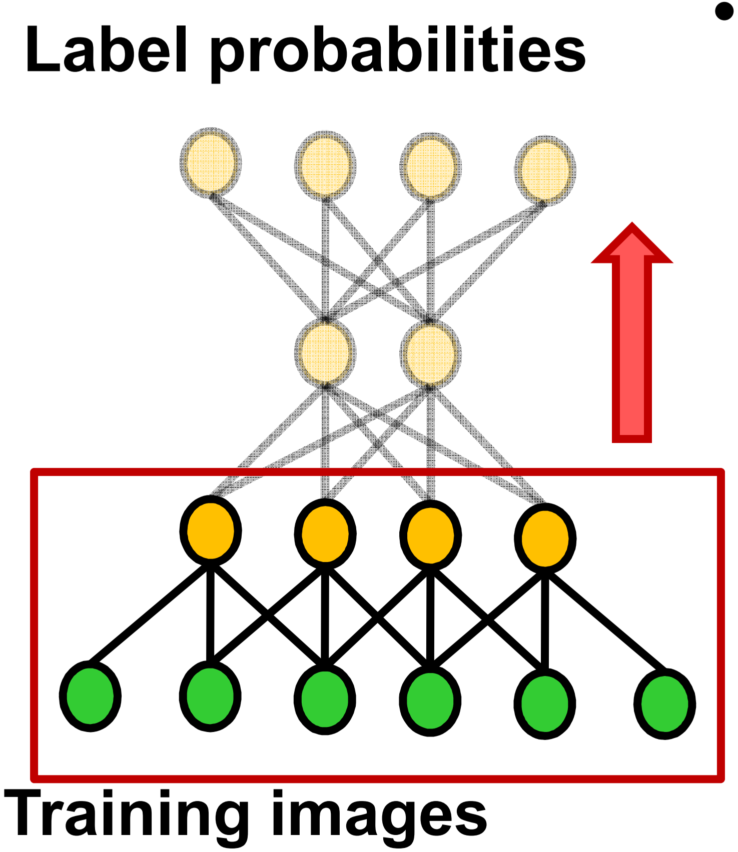


How we train a deep neural network



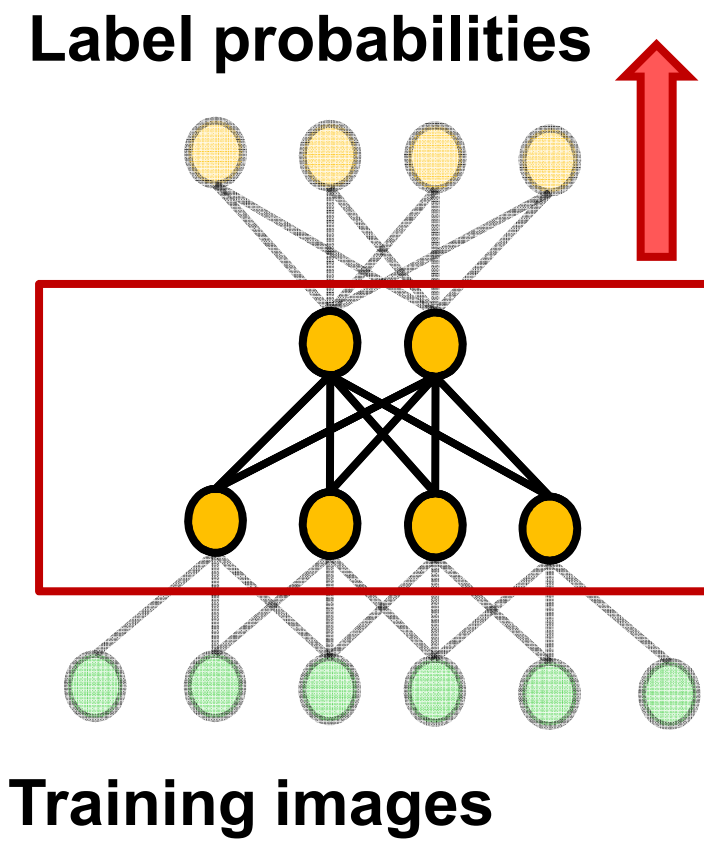
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass

How we train a deep neural network



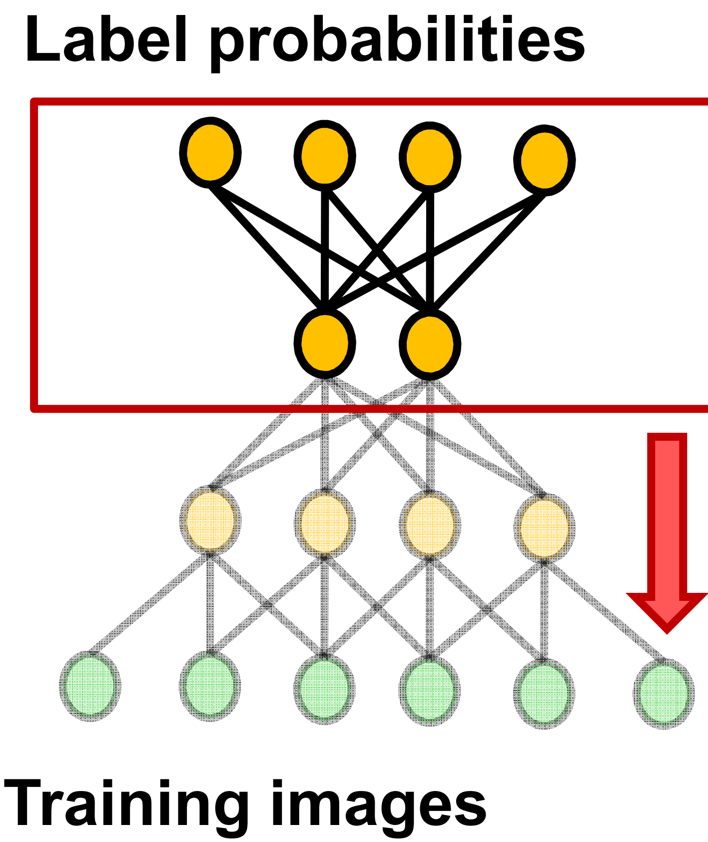
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass

How we train a deep neural network



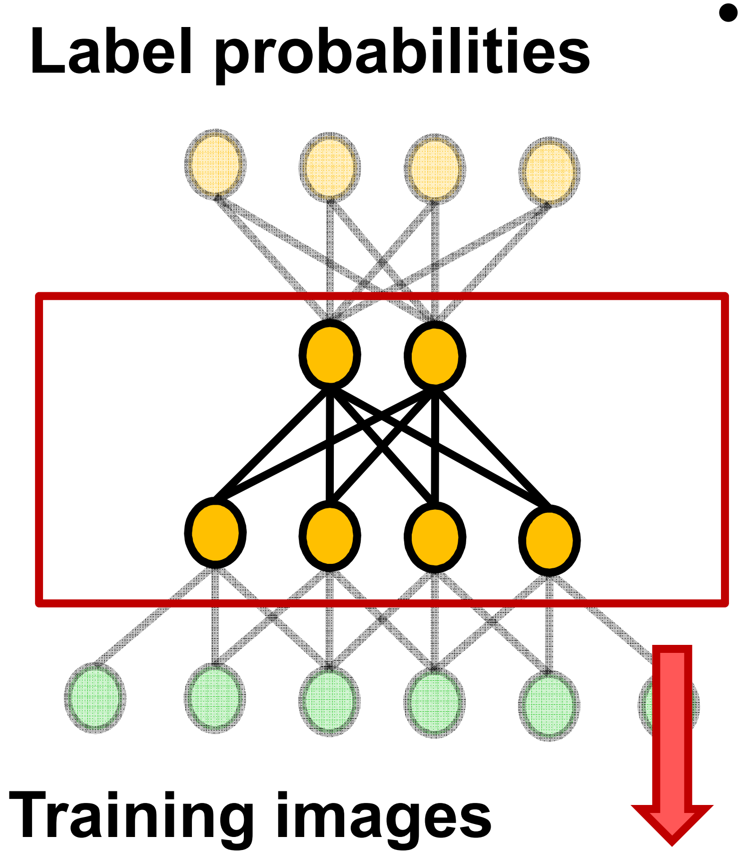
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass

How we train a deep neural network



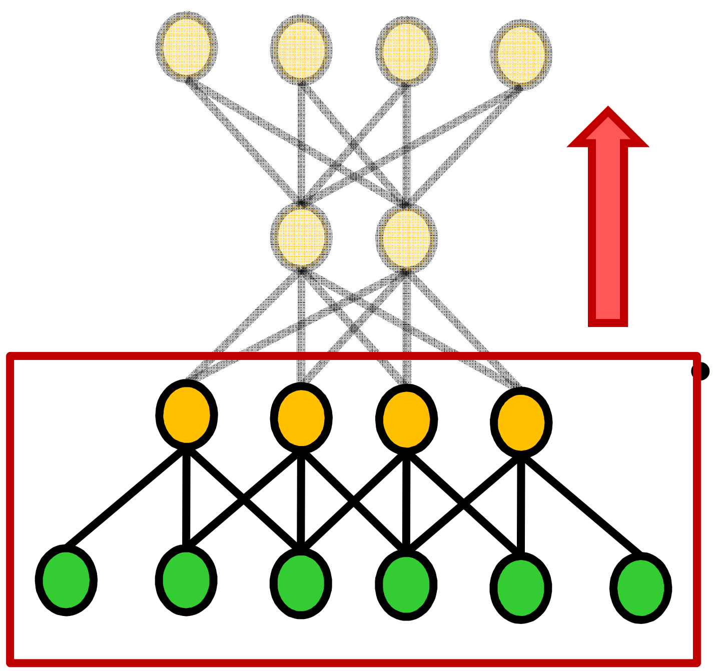
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass

How we train a deep neural network



- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass

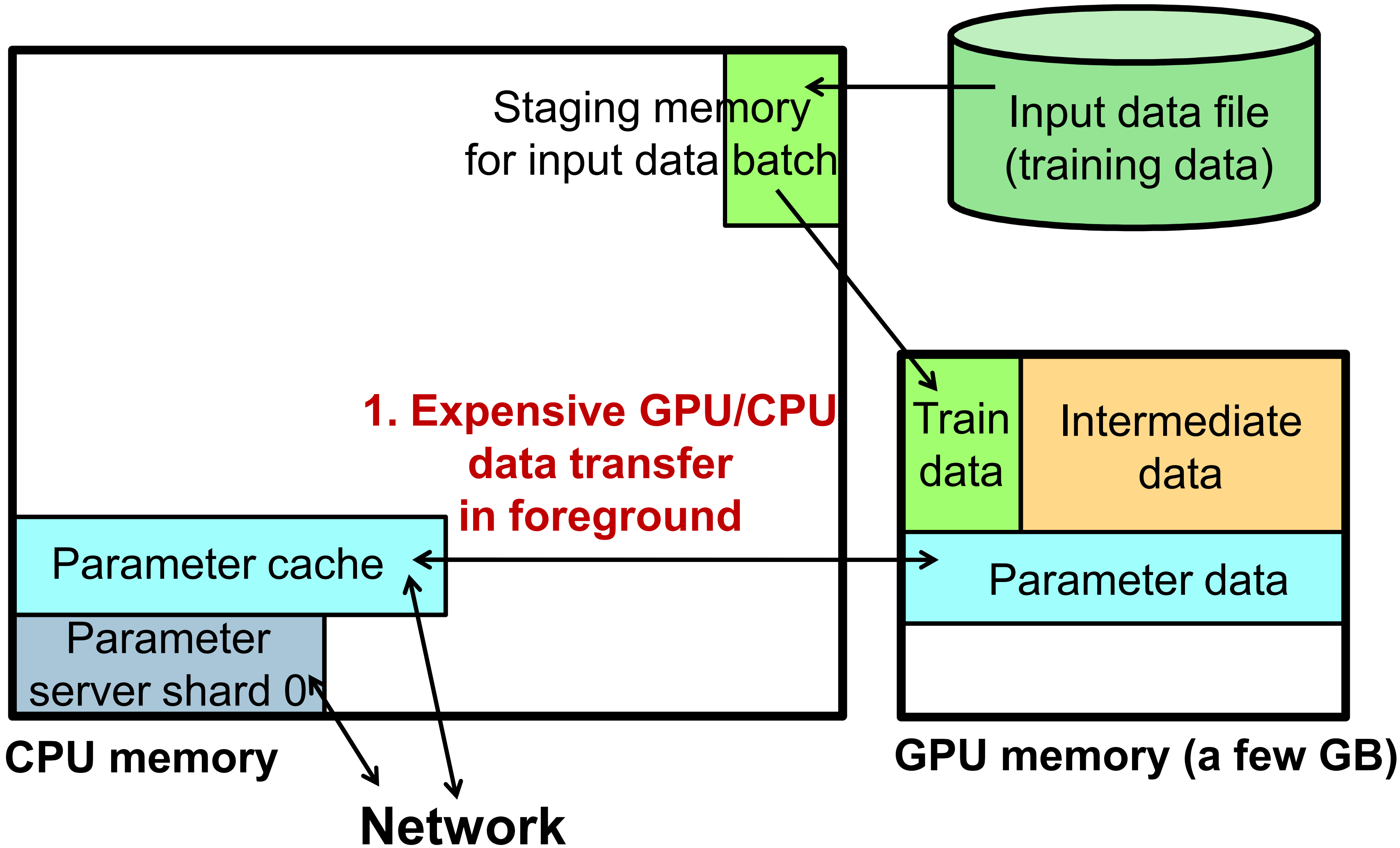
How we train a deep neural network



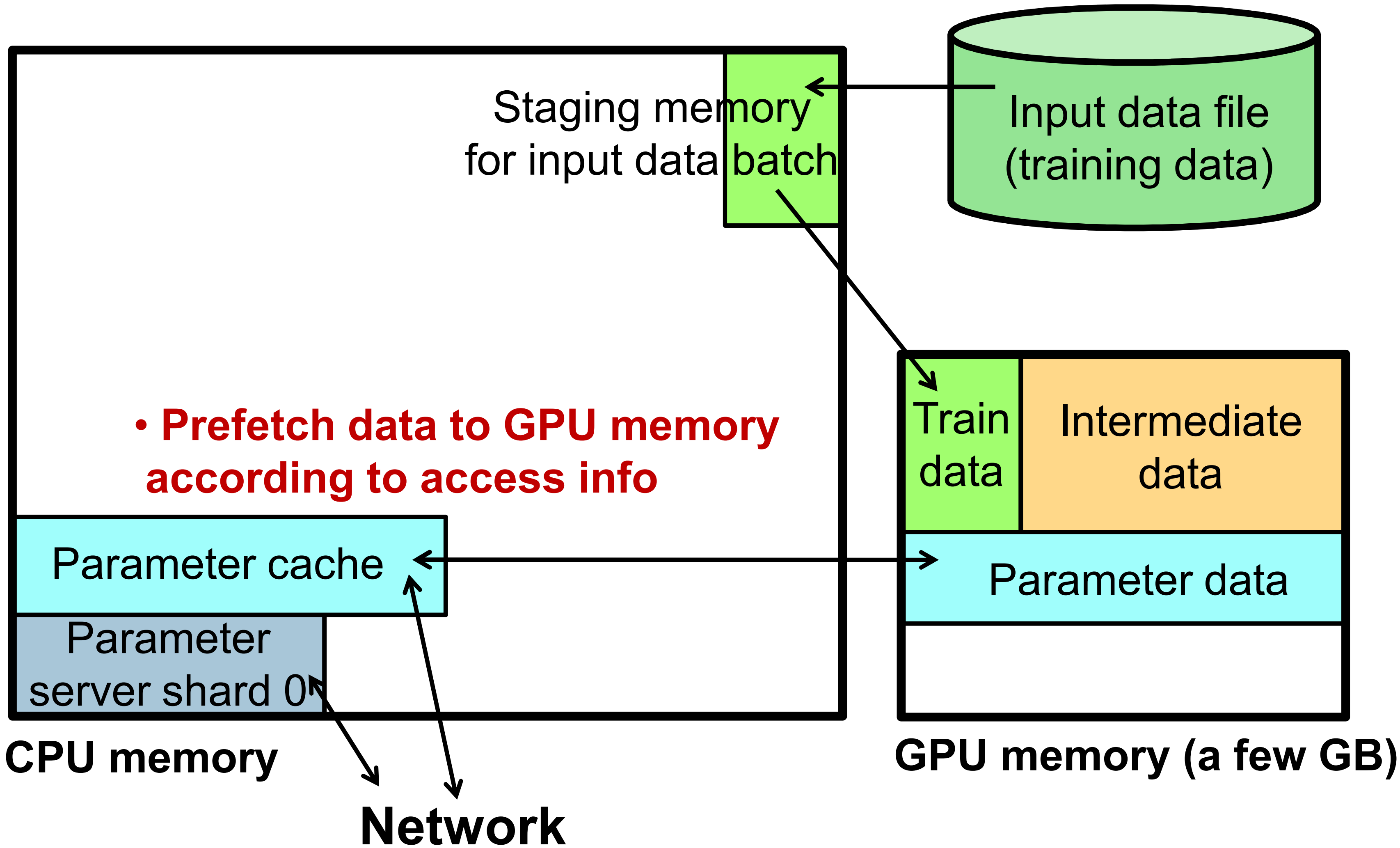
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass
- Important characteristics
  - repeating access ever iteration
  - only small fraction of data used at each step

GeePS will exploit those characteristics

Multi-GPU ML via GeePS

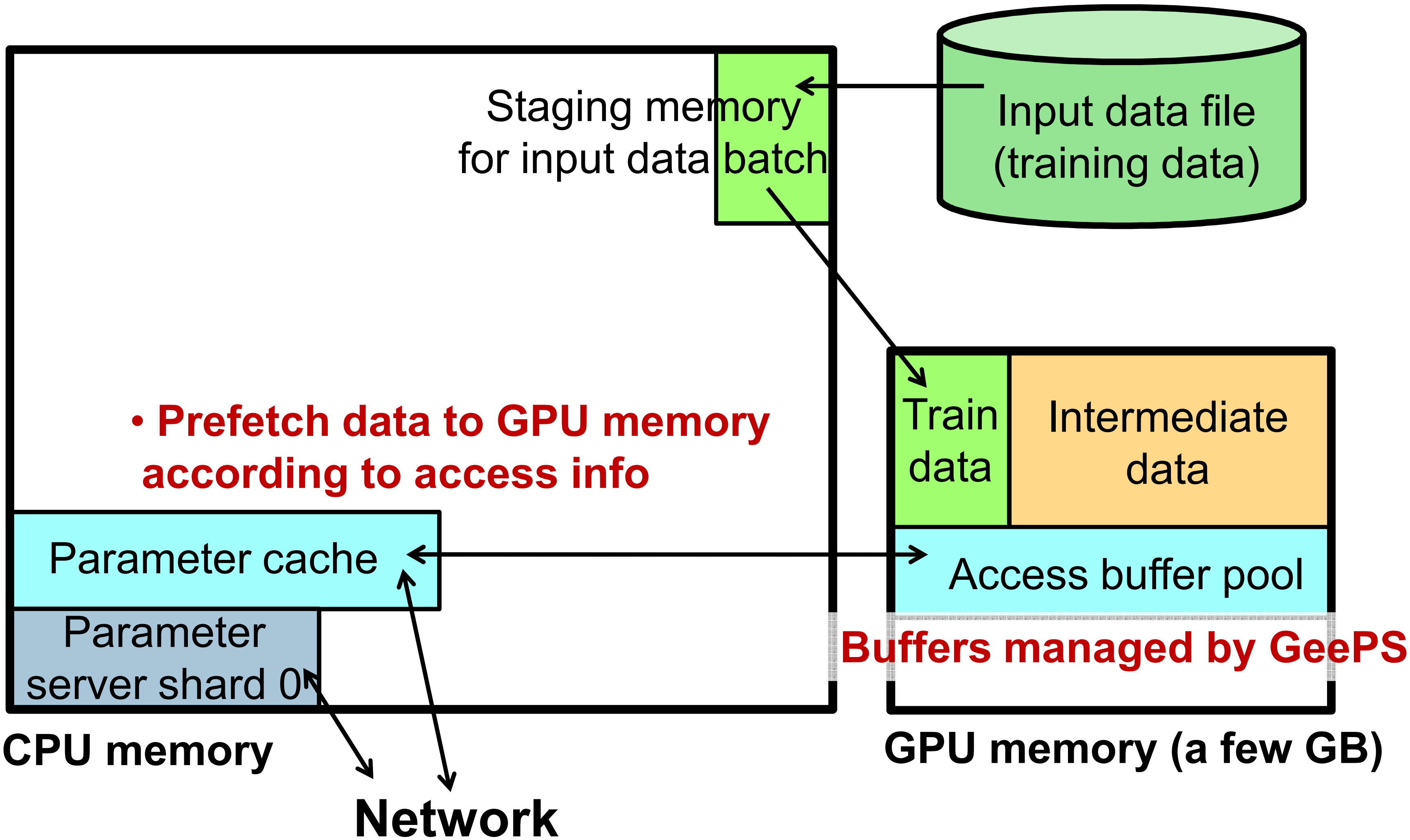


GeePS prefetches data in the background

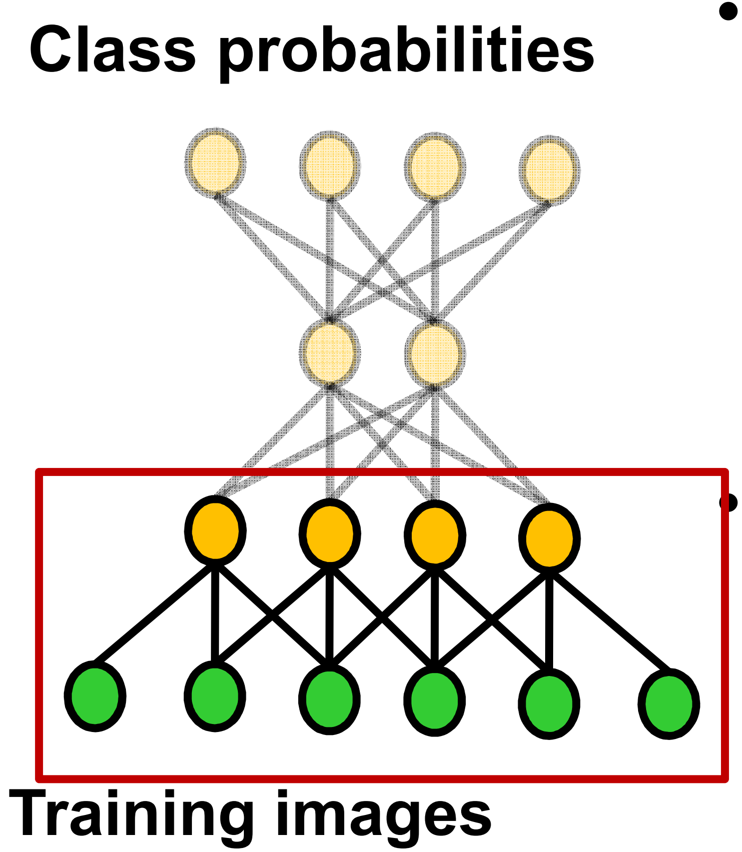


- **Prefetch data to GPU memory according to access info**

GeePS prefetches data in the background



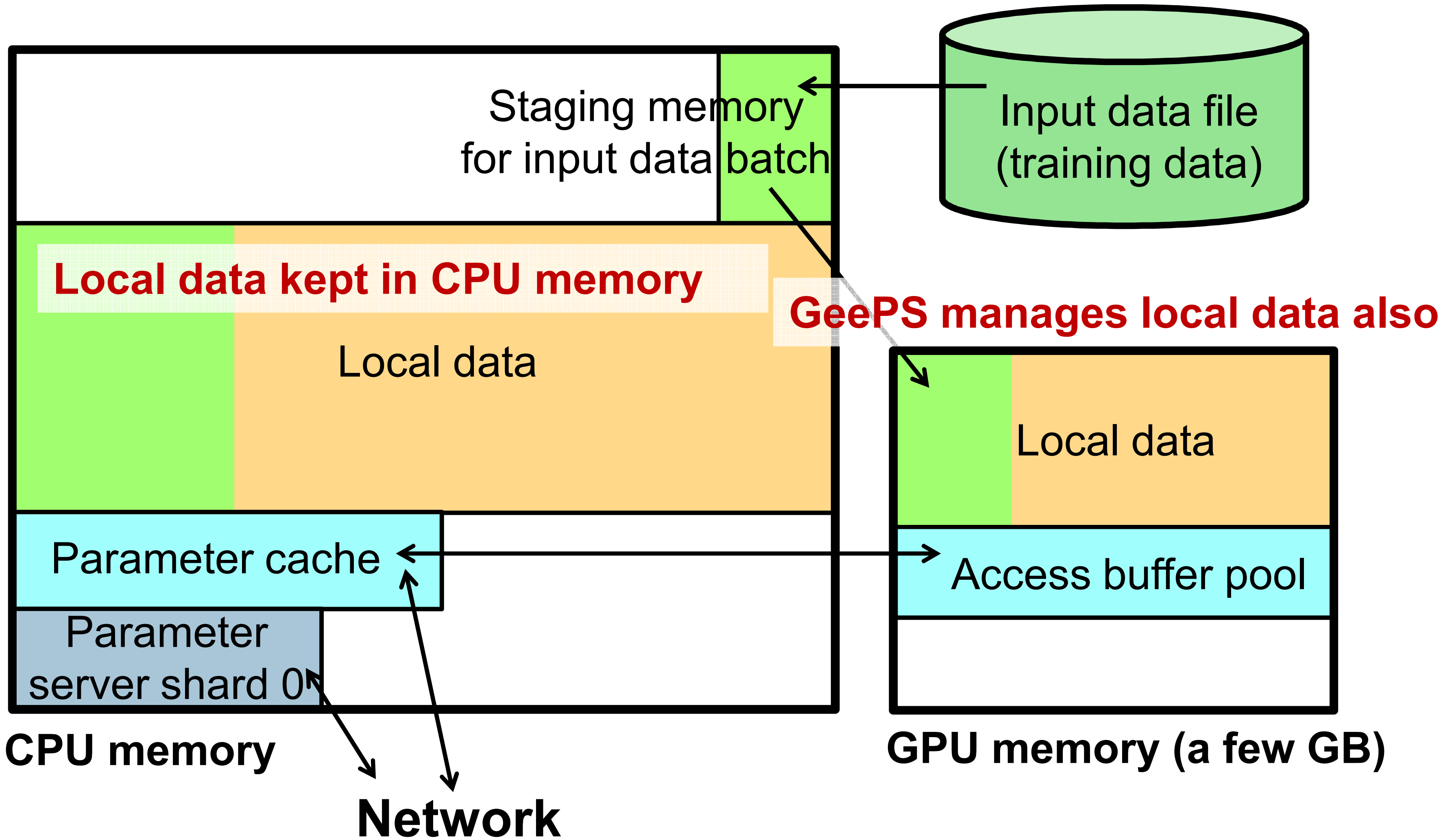
GPU memory management



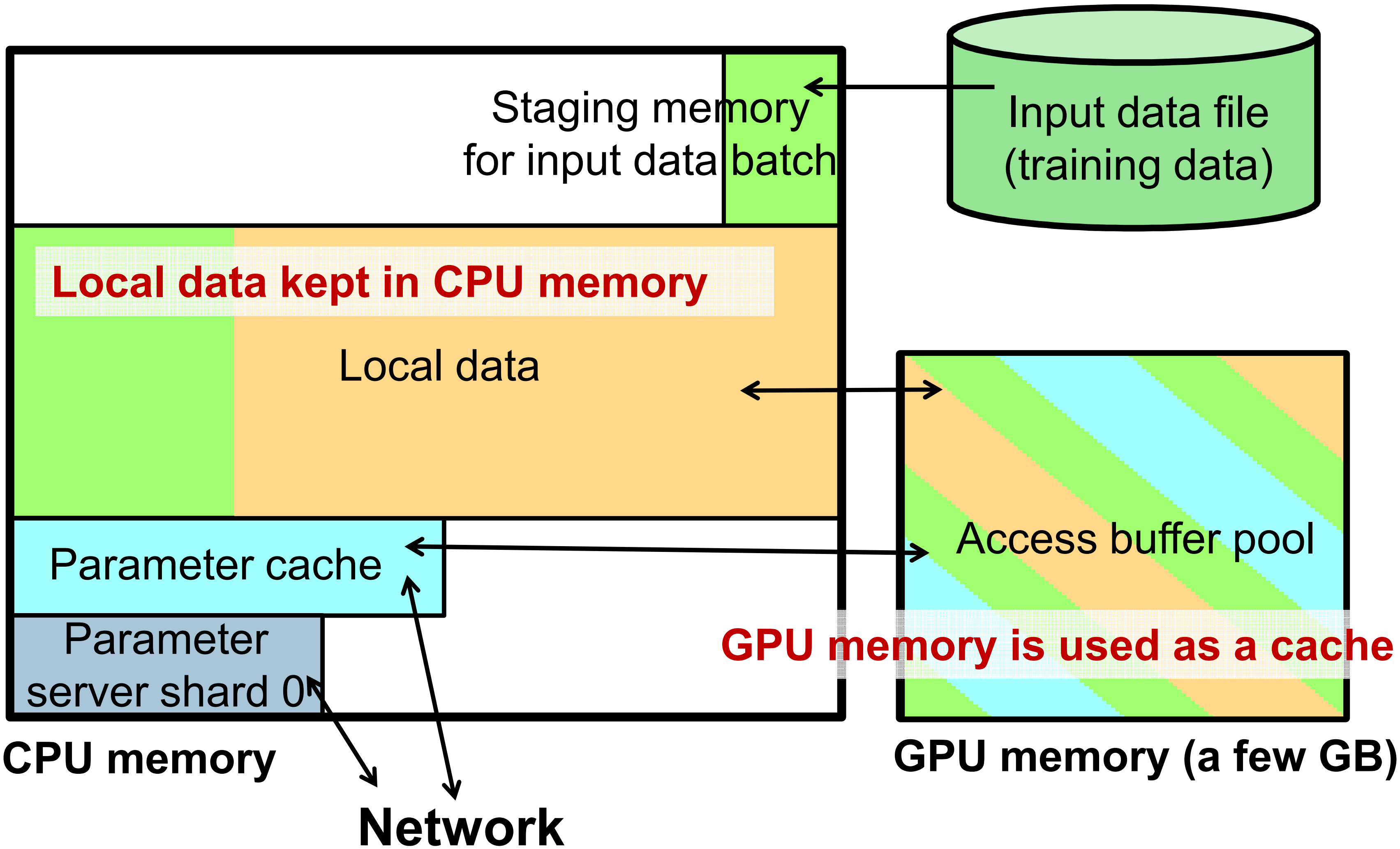
- For each iteration (mini-batch)
  - load one batch of training data
  - do a forward pass
  - do a backward pass
- Important characteristics
  - repeating access ever iteration
  - **only small fraction of data used at each step**

- **Use GPU memory as a cache to keep actively used data**
- **Keep the remaining data in CPU memory**

GPU memory management



GPU memory management

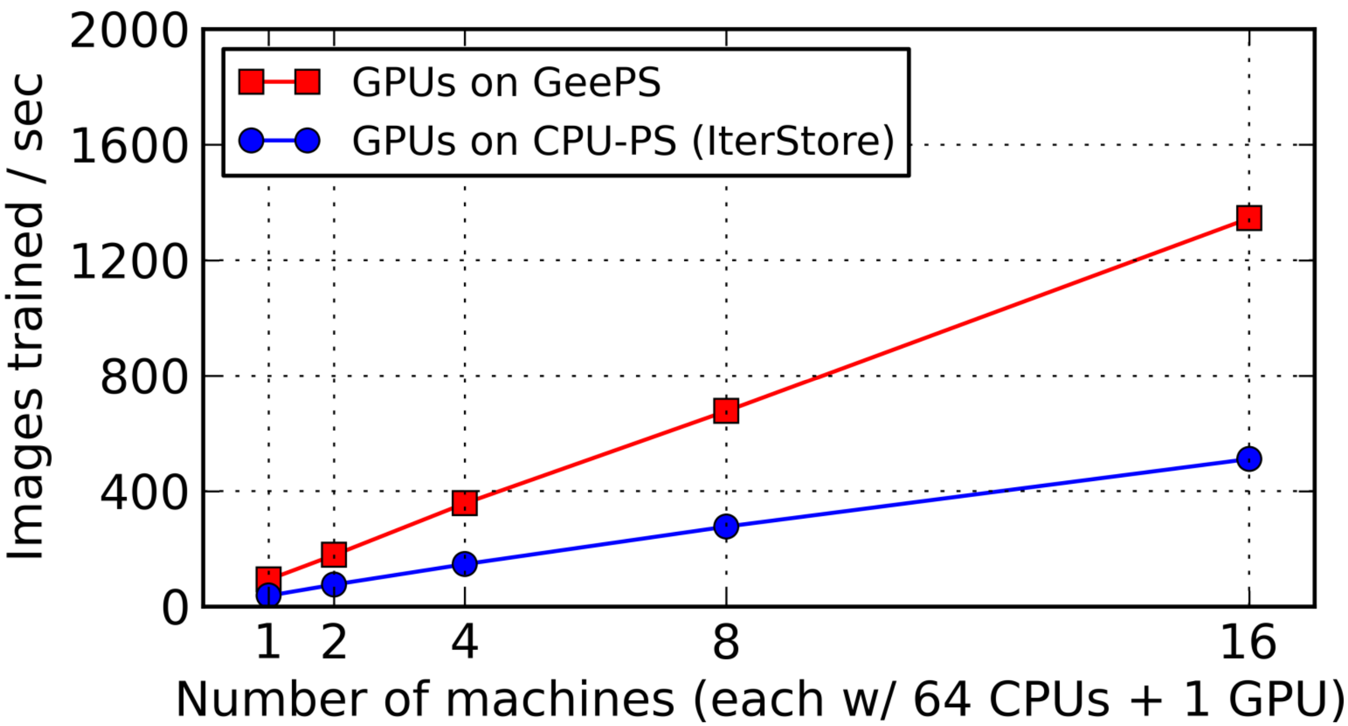


Experimental setups

- GeePS-Caffe setups
  - Caffe: single-machine GPU deep learning system
  - GeePS-Caffe: Caffe linked with GeePS
- Baseline
  - Caffe linked with CPU-based PS (IterStore)

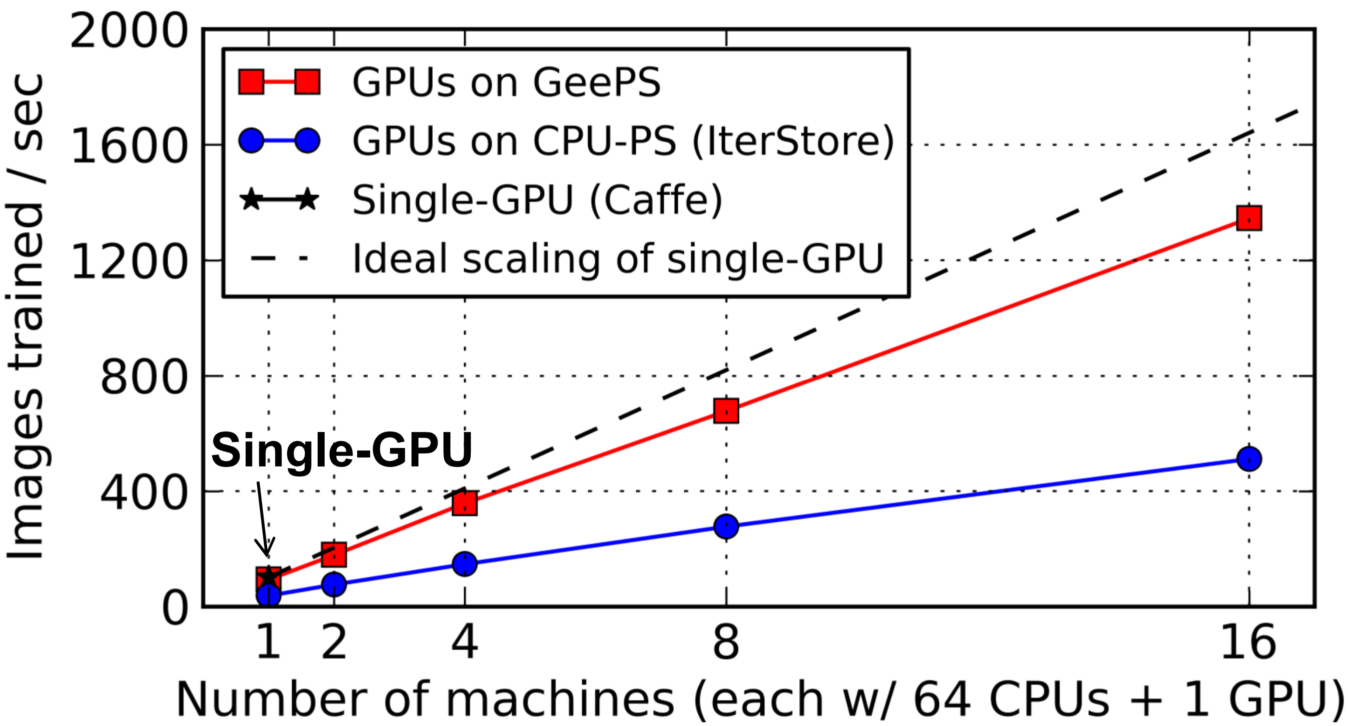
- Dataset and model
  - ImageNet: 7 million training images in 22,000 classes
  - Model: AlexNet

Training throughput



- **GeePS is much faster than CPU-based PS**
  - **2.6x higher throughput**

Training throughput



- **GeePS scales close to linear with more machines**
  - **with 16 machines, 13x faster than single-GPU**

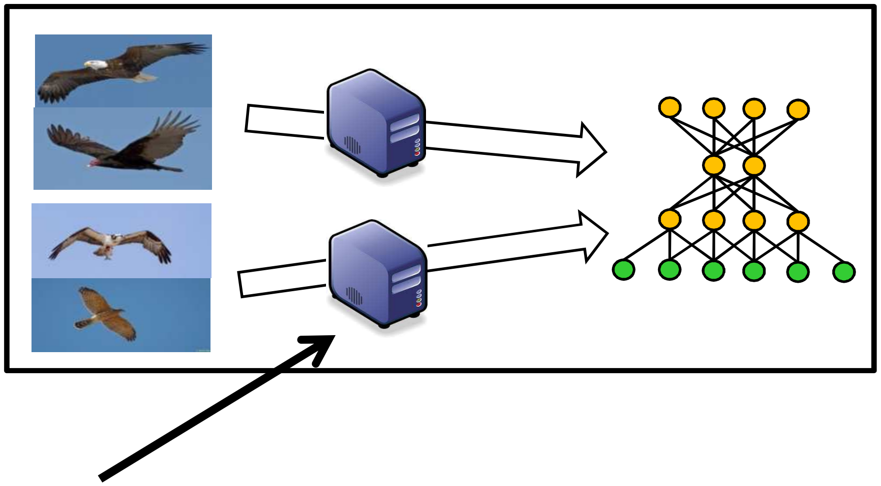
Take-away messages from GeePS

- GPU-specialized parameter server for GPU DL
  - exploits the layer-by-layer pattern
  - efficiently overlap data transfer with computation
  - 13x throughput speedup using 16 machines
  - efficiently handle problems larger than GPU memory

Three case studies

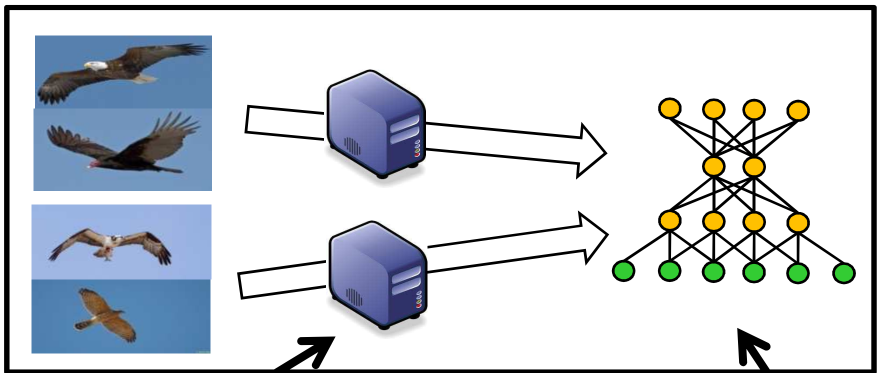
- IterStore [Cui et al. SoCC '14]
  - an efficient parameter server design
  - exploits repeated parameter data access
- GeePS [Cui et al. EuroSys '16]
  - specialized parameter server for GPU deep learning
  - exploits layer-by-layer pattern of deep learning
- MLtuner [In preparation]
  - system for automatic machine learning tuning
  - exploits quick decision of training hyperparam tuning

Training tunables in machine learning



- **Training tunables:**
  - learning rate (step size)
  - momentum
  - training batch size
  - data staleness bound
  - ...
- **Tuning them is important**
  - affect task completion time
  - affect solution quality

Training tunables in machine learning



- **Training tunables:**
  - learning rate (step size)
  - momentum
  - training batch size
  - data staleness bound
  - ...
- **NOT in the obj. function**

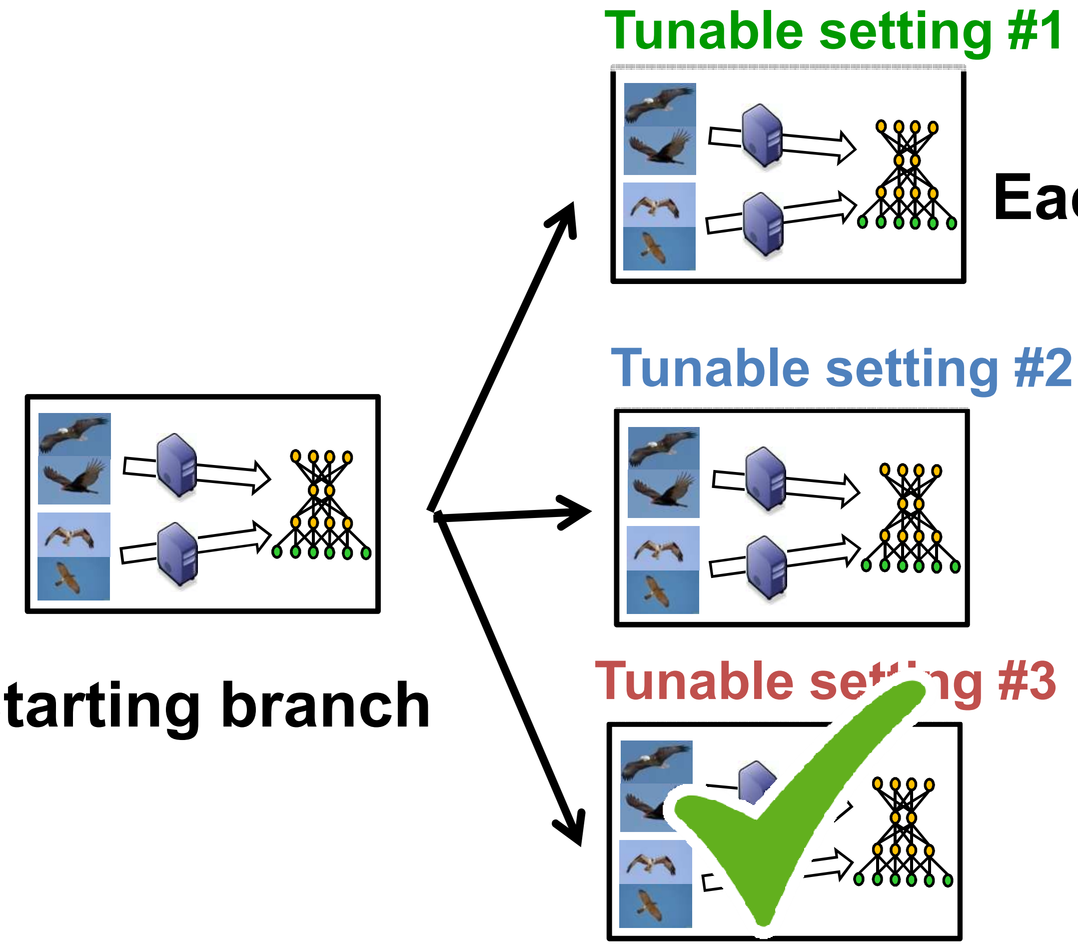
- **Model hyperparams:**
  - network depth
  - neuron layer sizes
  - neuron activation function
  - ...
- **In the obj. function**

Traditional tuning approaches

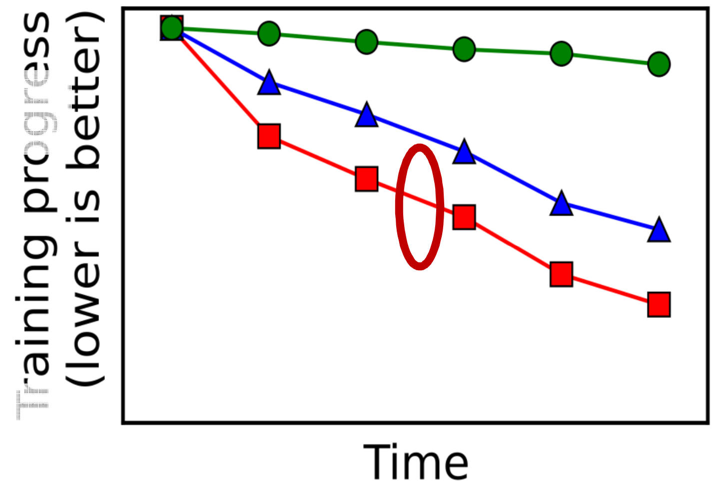
- Manual tuning
  - by domain expert, via trial and error
  - slow, expensive and prone to sub-optimal settings
- Automatic hyperparam tuning approaches
  - e.g., Spearmint [Snoek et al. '12] and HyperBand [Li et al. '16]
  - designed for model hyperparameter tuning
  - need to train models to completion multiple times
    - high overhead
  - cannot change tunables during training
    - fail to achieve good model quality for many apps

MLtuner will address those problems

Try & evaluate tunables in trial branches



Each running for some time

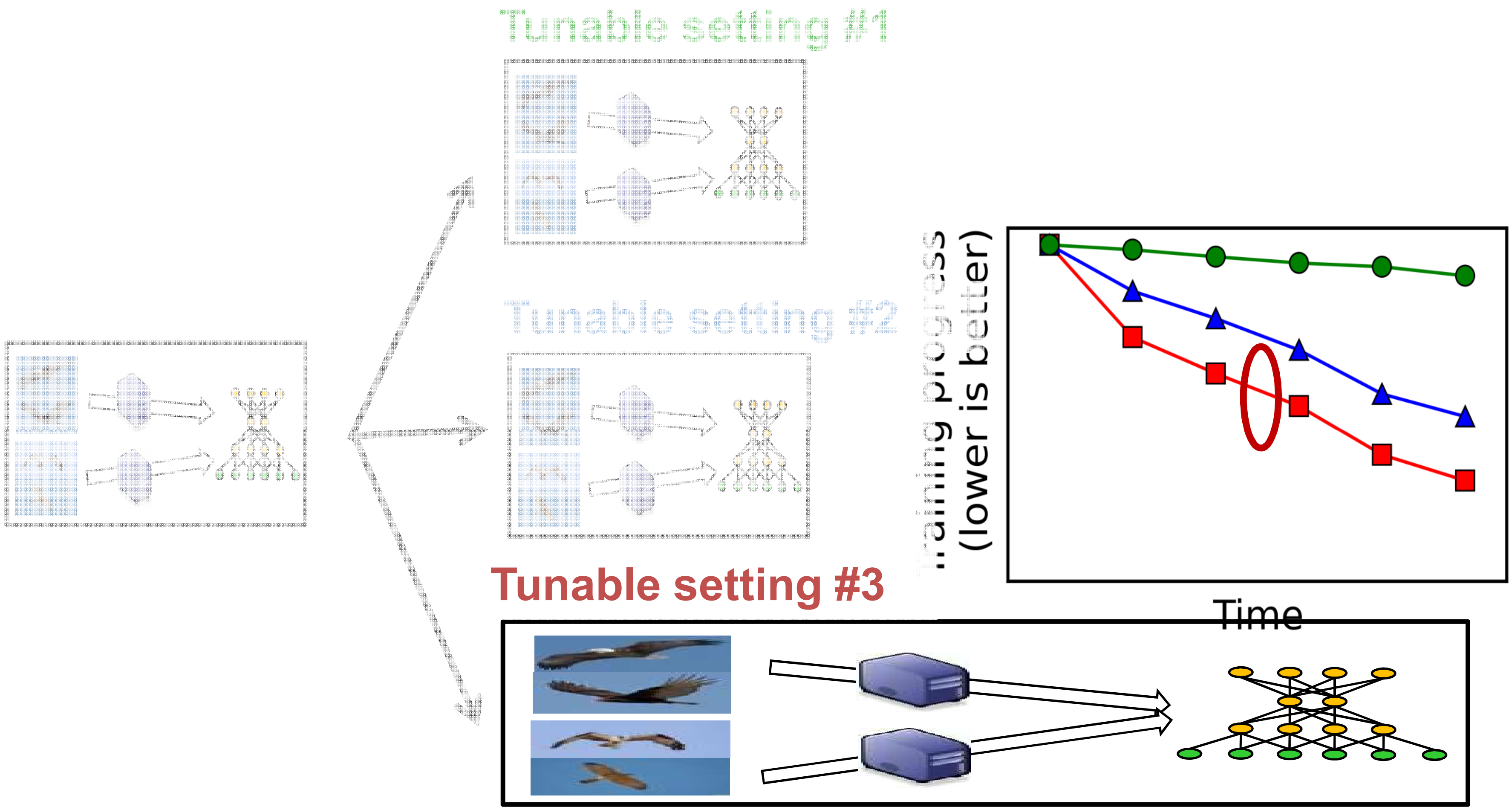


Starting branch

Monitor convergence speed

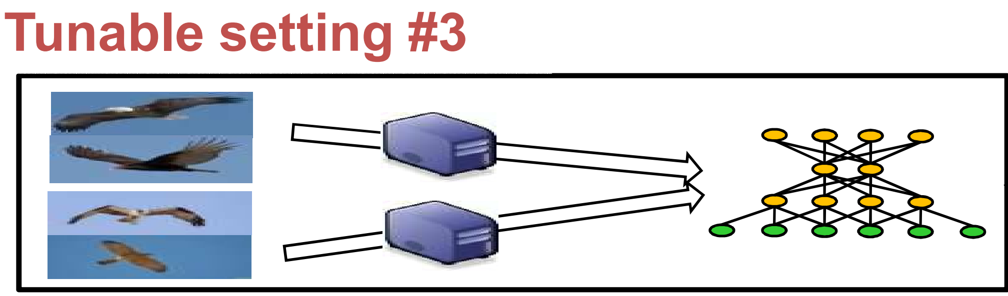
Forked trial training branches

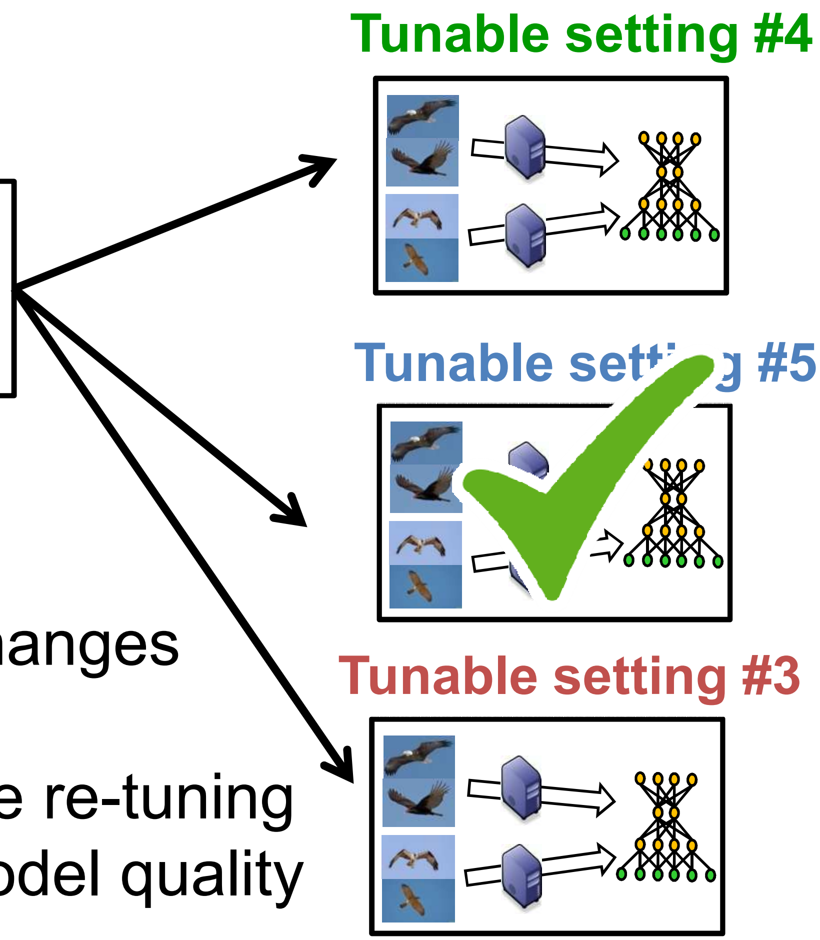
Try & evaluate tunables in trial branches



Keep training only the best branch

Re-tuning tunables

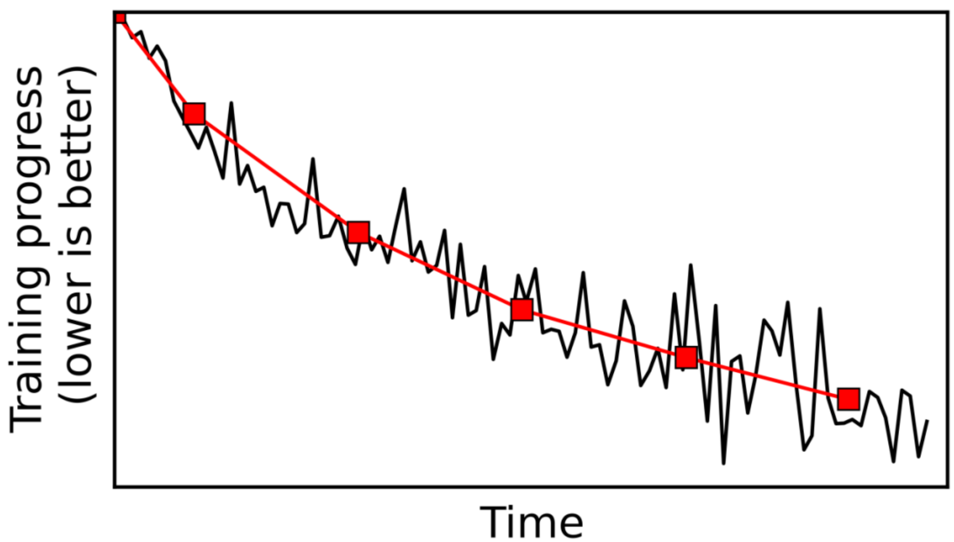




- Re-tune because best setting changes
- Many deep learning apps require re-tuning learning rate to achieve good model quality

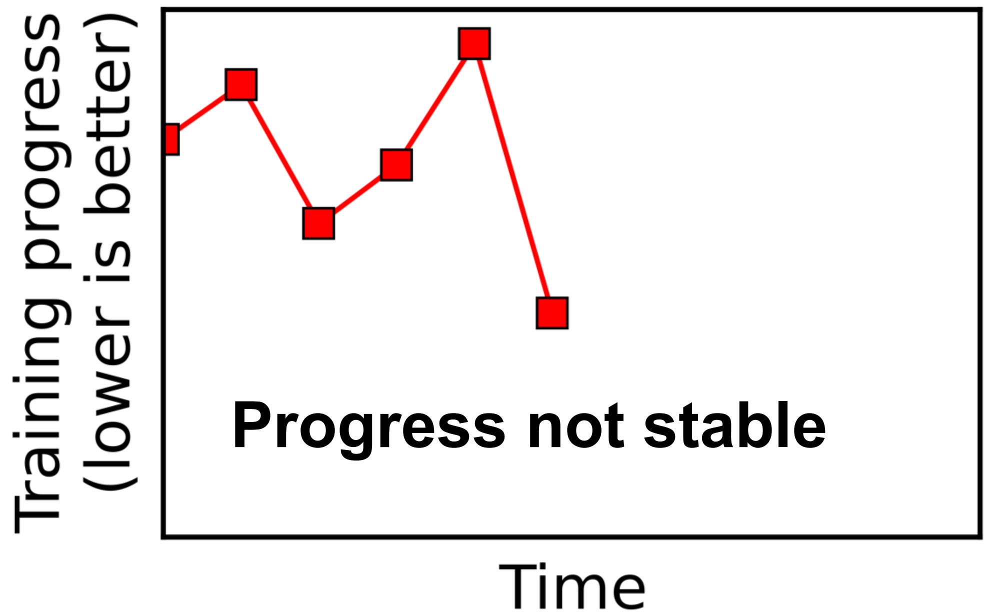
MLtuner design details

- Pick settings to try with HyperOpt algorithm
- Monitor training loss to estimate speed
- Downsample the noisy loss traces



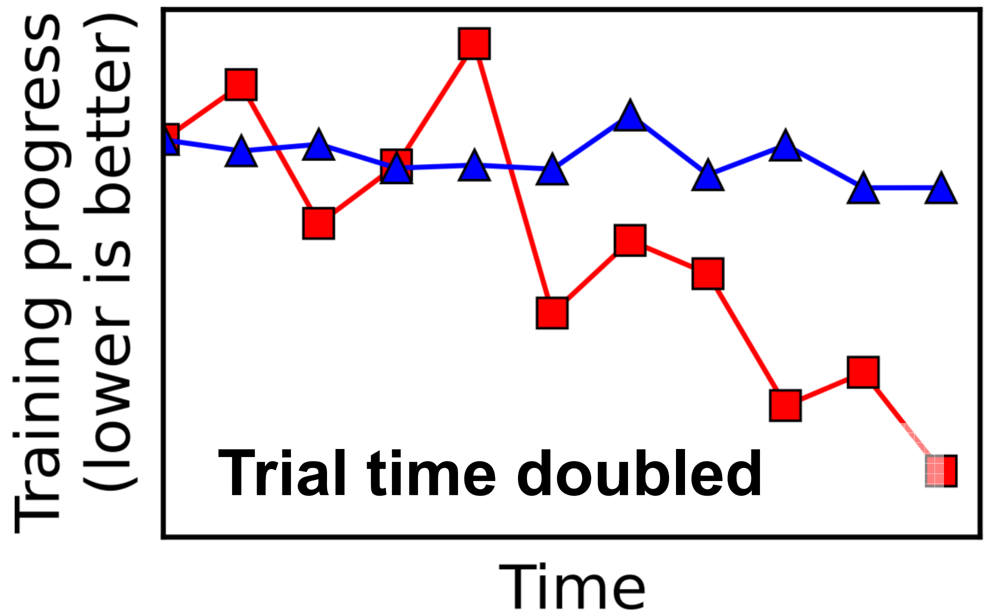
- Decide the trial time based on noisiness

Deciding trial time based on stability



- Start with minimal trial time
- Check the stability of converging progress

Deciding trial time based on stability



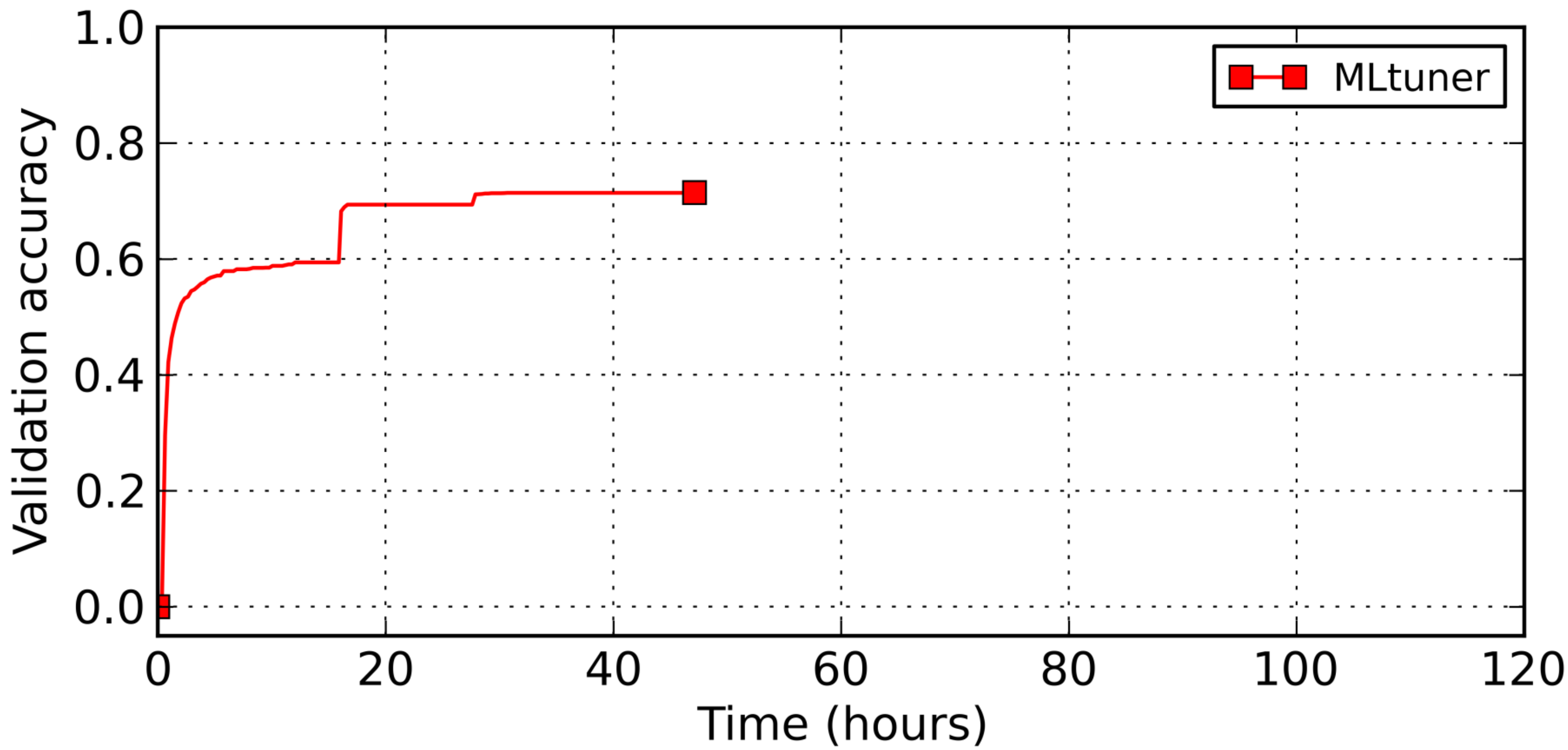
- Start with minimal trial time
- Check the stability of converging progress
- Double trial time, until any stable branch found
- Use the decided trial time for all future branches

Experimental setups

- Application: image classification
  - model: Inception-BN
  - dataset: ILSVRC12
  - other apps include RNNs and matrix factorization
- Tunables:
  - learning rate, momentum, batch size, data staleness
- Baselines
  - Spearmint [Snoek et al. '12]
  - HyperBand [Li et al. '16]

MLtuner vs. traditional tuning approaches

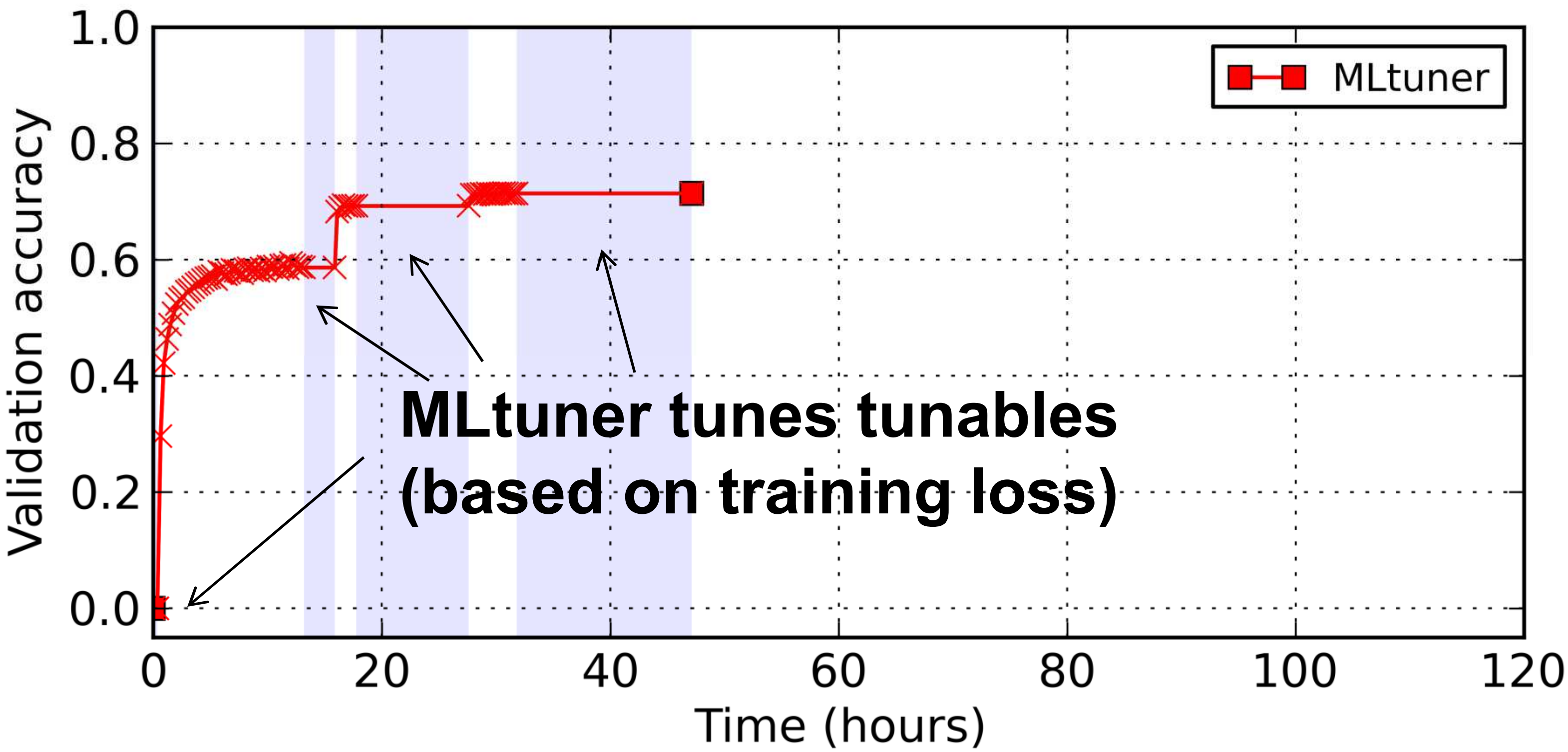
Benchmark: image classification with Inception-BN on ILSVRC12
Tunables: learning rate, momentum, batch size, data staleness



Re-tuning improves model accuracy

MLtuner vs. traditional tuning approaches

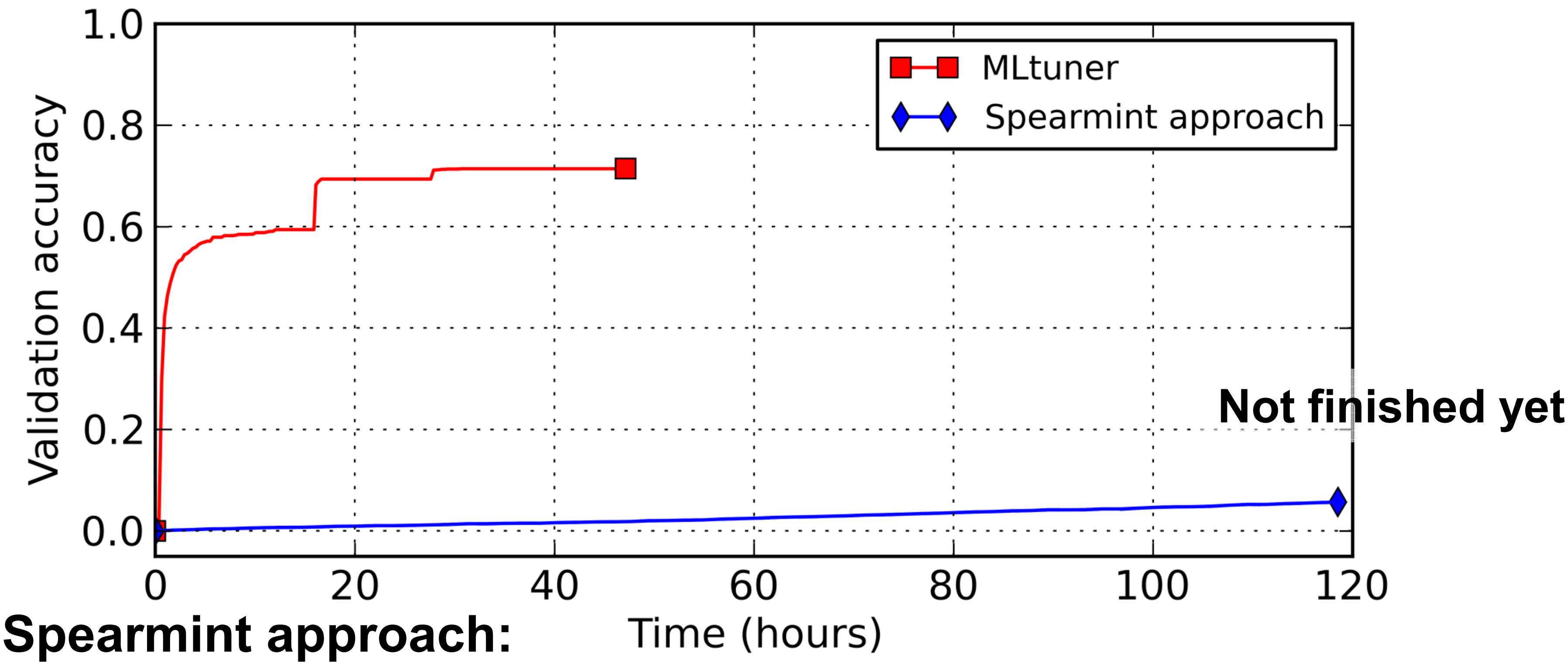
Benchmark: image classification with Inception-BN on ILSVRC12
Tunables: learning rate, momentum, batch size, data staleness



Re-tuning improves model accuracy

MLtuner vs. traditional tuning approaches

Benchmark: image classification with Inception-BN on ILSVRC12
Tunables: learning rate, momentum, batch size, data staleness

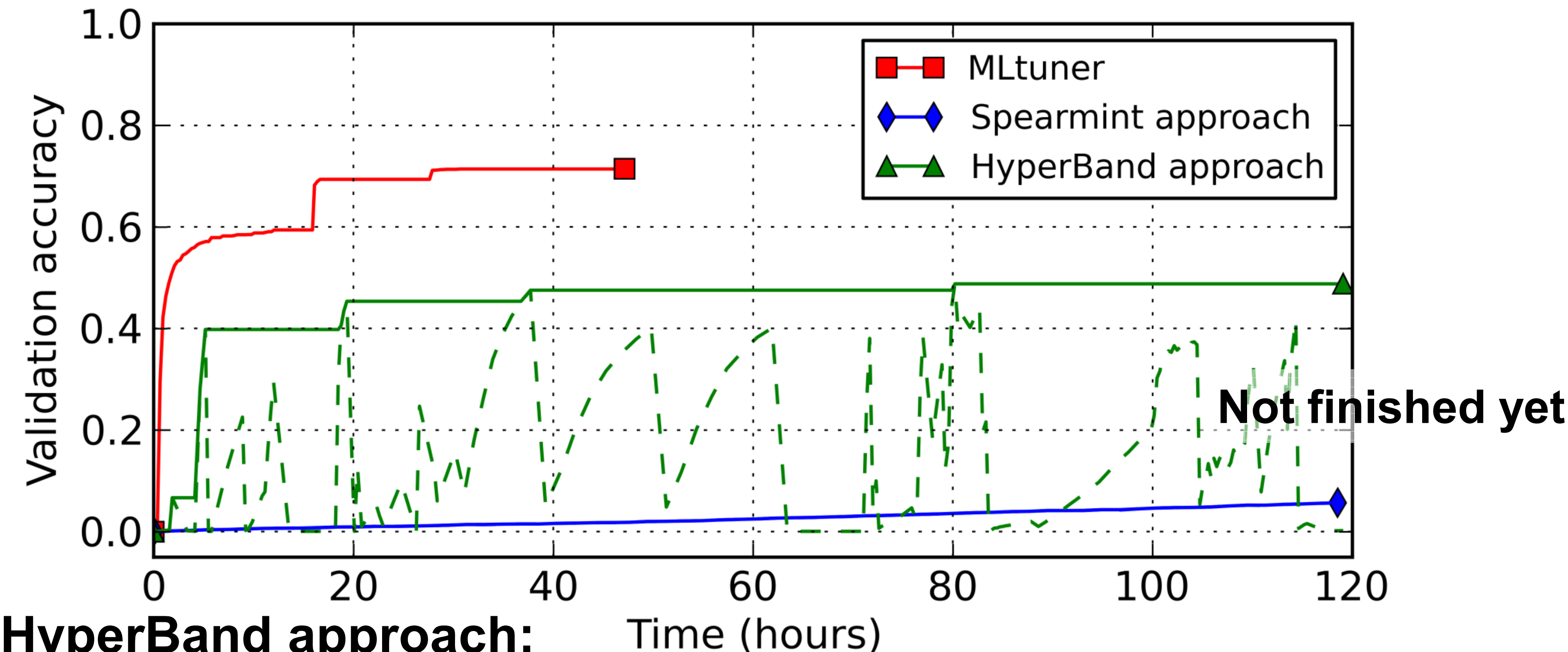


Spearmint approach:

- **sample settings with Bayesian optimization**
- **run each sampled setting to completion**
- **spent all time trying setting (lr=1e-5, m=0, bs=2, ds=0)**

MLtuner vs. traditional tuning approaches

Benchmark: image classification with Inception-BN on ILSVRC12
Tunables: learning rate, momentum, batch size, data staleness

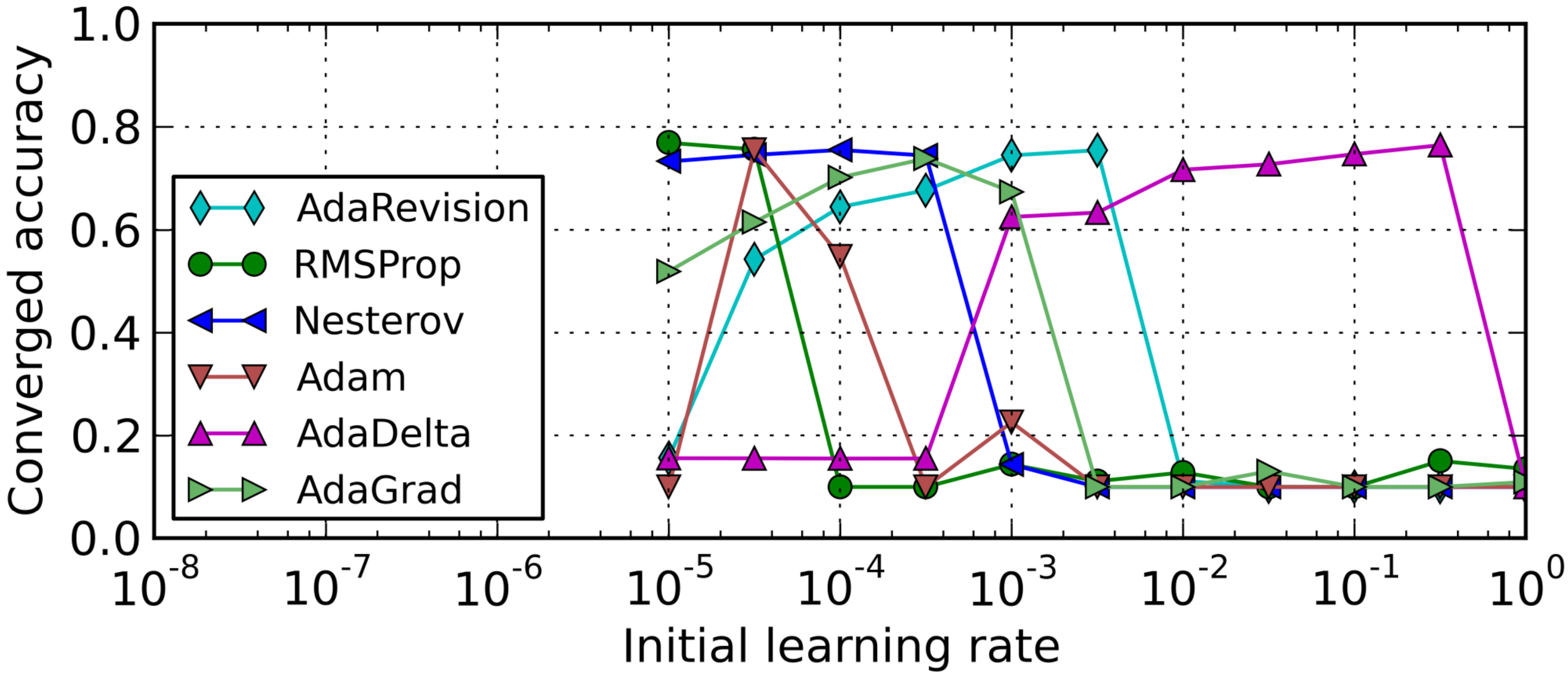


HyperBand approach:
- **try multiple settings in parallel**
- **stop half of the trying settings every a few iterations**

MLtuner converges faster and reaches higher accuracy

Tuning initial LR for adaptive LR algorithms

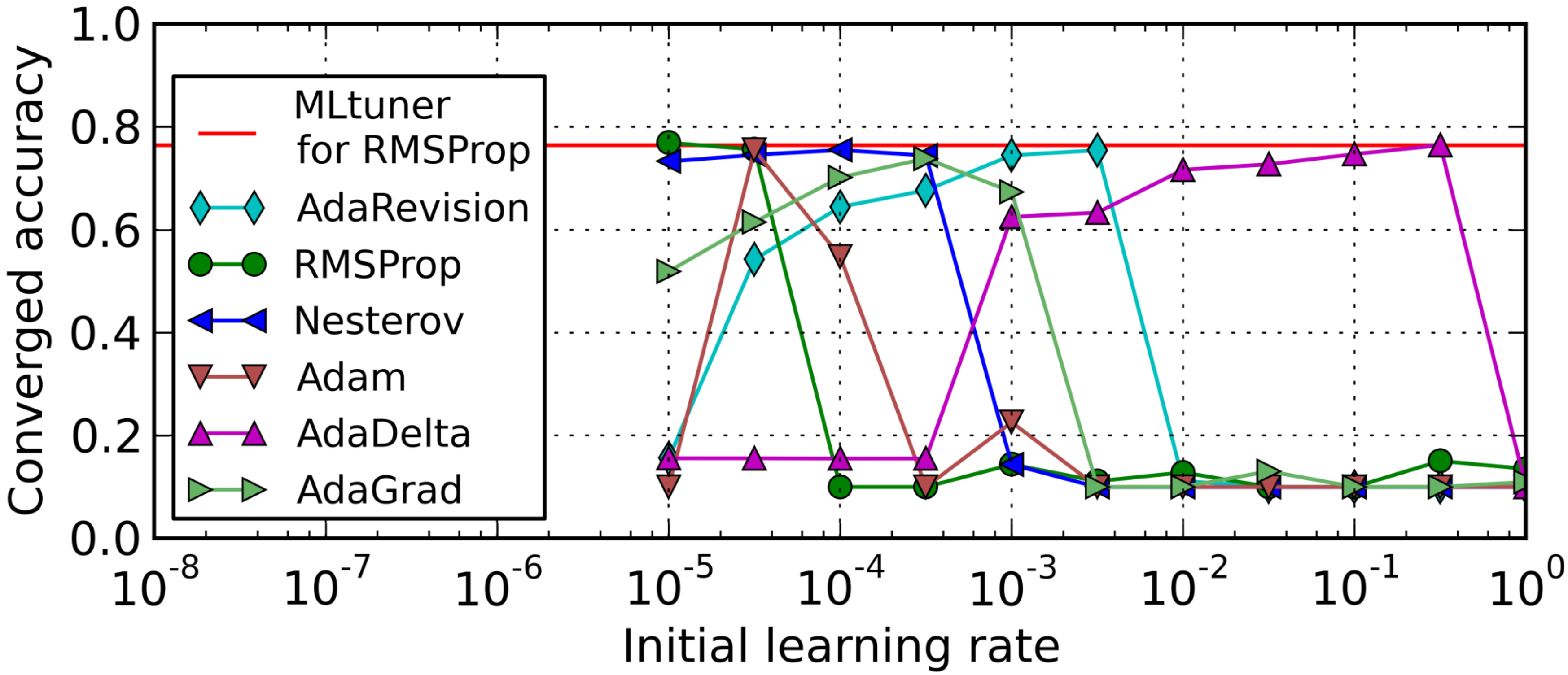
- **Benchmark: image classification with AlexNet on Cifar10**



Initial learning rate affects converged model accuracies

Tuning initial LR for adaptive LR algorithms

- **Benchmark: image classification with AlexNet on Cifar10**
- **Tunable: initial learning rate**



Initial learning rates picked by MLtuner are close to optimal
MLtuner complements the adaptive LR algorithms

Thesis contributions

- IterStore [Cui et al. SoCC ‘14]
  - exploited repeated data access
  - designed methods of determining it and specializations to exploit it
  - up to 50x speed up
- GeePS [Cui et al. EuroSys ‘16]
  - exploited layer-by-layer pattern of deep learning
  - designed a PS that overlaps GPU/CPU data transfer with computation
  - 2.5x speed up compared to traditional CPU parameter server
- MLtuner [In preparation]
  - identified training tunables as a special class of hyperparams
  - over an order of magnitude faster than traditional tuning approaches

Thank you!

- My wife
- My advisor: Greg Ganger
- My thesis committee
  - Phil Gibbons, Garth Gibson, Eric Xing, Derek Murray
- My collaborators and colleagues at PDL
  - Jason Boles, Jim Cipar, Chuck Cranor, David Dai, Joan Digney, Chad Dougherty, Zis Economou, Mitch Franzos, Aaron Harlap, Qirong Ho, Kevin Hsieh, Angela Jiang, Rajat Kateja, Kim Keeton, Jin Kyu Kim, Karen Lindenfelser, Hyeontaek Lim, Jun Woo Park, Aurick Qiao, Indrajit Roy, Alexey Tumanov, Jinliang Wei, Pengtao Xie, Lianghong Xu, Xiaolin Zang, Hao Zhang

References

- **[IterStore]** H. Cui, A. Tumanov, J. Wei, L. Xu, W. Dai, J. Haber-Kucharsky, Q. Ho, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Exploiting iterative-ness for parallel ML computations. In SoCC, 2014.
- **[GeePS]** H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In EuroSys, 2016.
- **[LazyTable]** H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Exploiting bounded staleness to speed up big data analytics. In ATC, 2014.
- **[Aperture]** H. Cui, K. Keeton, I. Roy, K. Viswanathan, and G. R. Ganger. Using data transformations for low-latency time series analysis. In SoCC, 2015.

References

- **[SSP]** Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons,G. A. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ML via a Stale Synchronous Parallel parameter server. In NIPS, 2013.
- **[GraphLab]** J. Gonzalez, Y. Low, et al. PowerGraph: Distributed graph-parallel computation on natural graphs. In OSDI, 2012.
- **[Caffe]** Y. Jia, et al. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014.
- **[Spearmint]** J. Snoek, et al. Practical bayesian optimization of machine learning algorithms. In NIPS, 2012.
- **[HyperBand]** L. Li, et al. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. arXiv preprint arXiv:1603.06560v3, 2016.

Additional related work

- M. Li, et al. Scaling distributed machine learning with the parameter server. In OSDI, 2014.
- T. Chilimbi, et al. Project Adam: Building an efficient and scalable deep learning training system. In OSDI, 2014.
- T. Chen, et al. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274, 2015.
- J. Dean, et al. Large scale distributed deep networks. In NIPS, 2012.
- E. R. Sparks, et al. Automating Model Search for Large Scale Machine Learning. In SoCC, 2015