# Week 6. Scaling Neural Nets

# Scaling inference

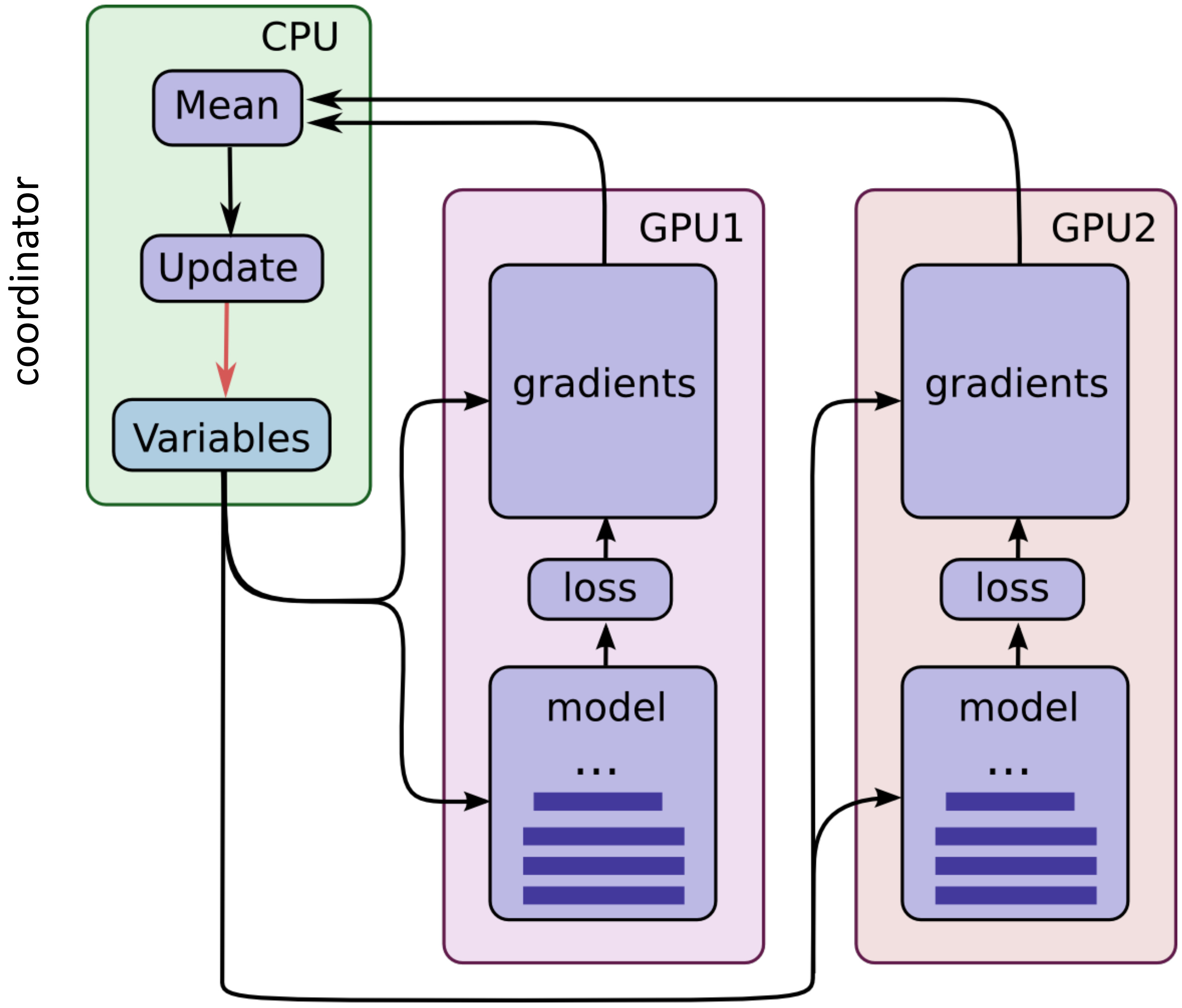
- In case of 1 object you can speed up inference with:
  - Fast GPU (TensorRT, etc) https://venturebeat.com/2018/03/27/nvidia-speeds-up-deep-learning-inference-processing/
  - Distillation (teacher networks) https://arxiv.org/pdf/1412.6550.pdf
  - Quantization (fast INT8 operations) https://www.tensorflow.org/performance/quantization https://www.theregister.co.uk/2016/09/13/nvidia_p4_p40_gpu_ai/
  - Mobile device optimized architectures (MobileNet) https://arxiv.org/pdf/1704.04861.pdf
  - …

- For many objects:
  - Embarrassingly parallel, not really an issue

# Scaling training is the real issue

- Two approaches:
  - Data-parallel
  - Model-parallel

# Data-parallel




- Copy of the model is stored on each GPU
- Each GPU processes half of the batch
- Gradients are aggregated on coordinator (maybe on CPU) and are sent to all GPUs to update parameters

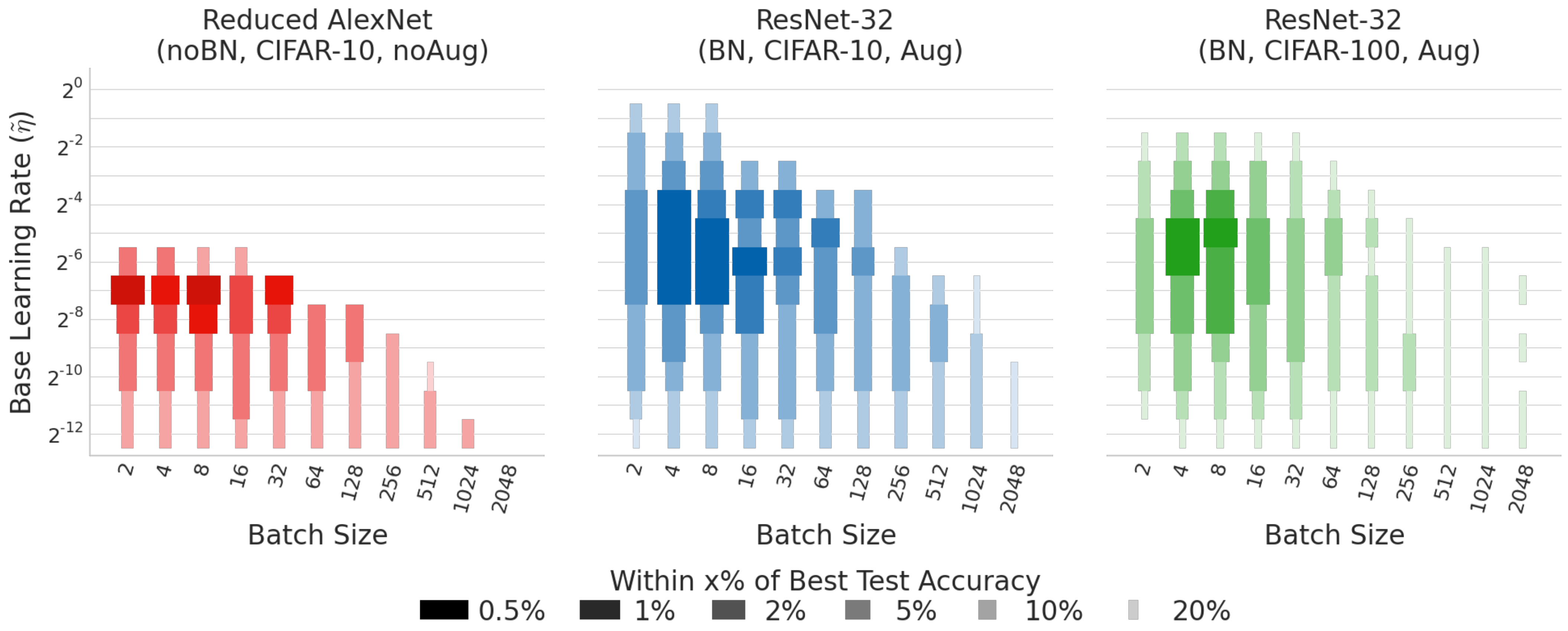
https://github.com/awslabs/deeplearning-benchmark/tree/master/tensorflow/inception

# Batch size dilemma

- Small batch size is bad for scaling:
  - Weak GPU utilization
  - Limited number of workers

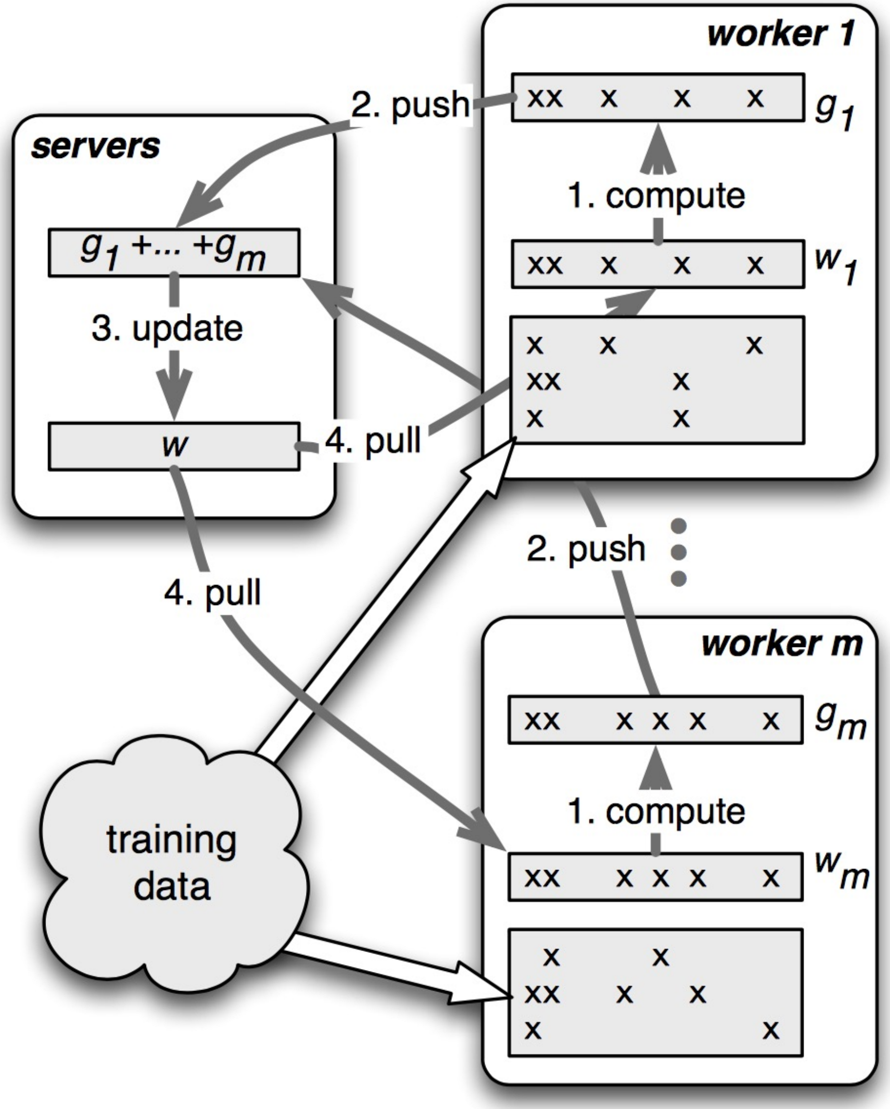
# Batch size dilemma

- Big batch size can hurt accuracy:




https://arxiv.org/pdf/1609.04836.pdf https://arxiv.org/pdf/1804.07612.pdf https://www.graphcore.ai/posts/revisiting-small-batch-training-for-deep-neural-networks

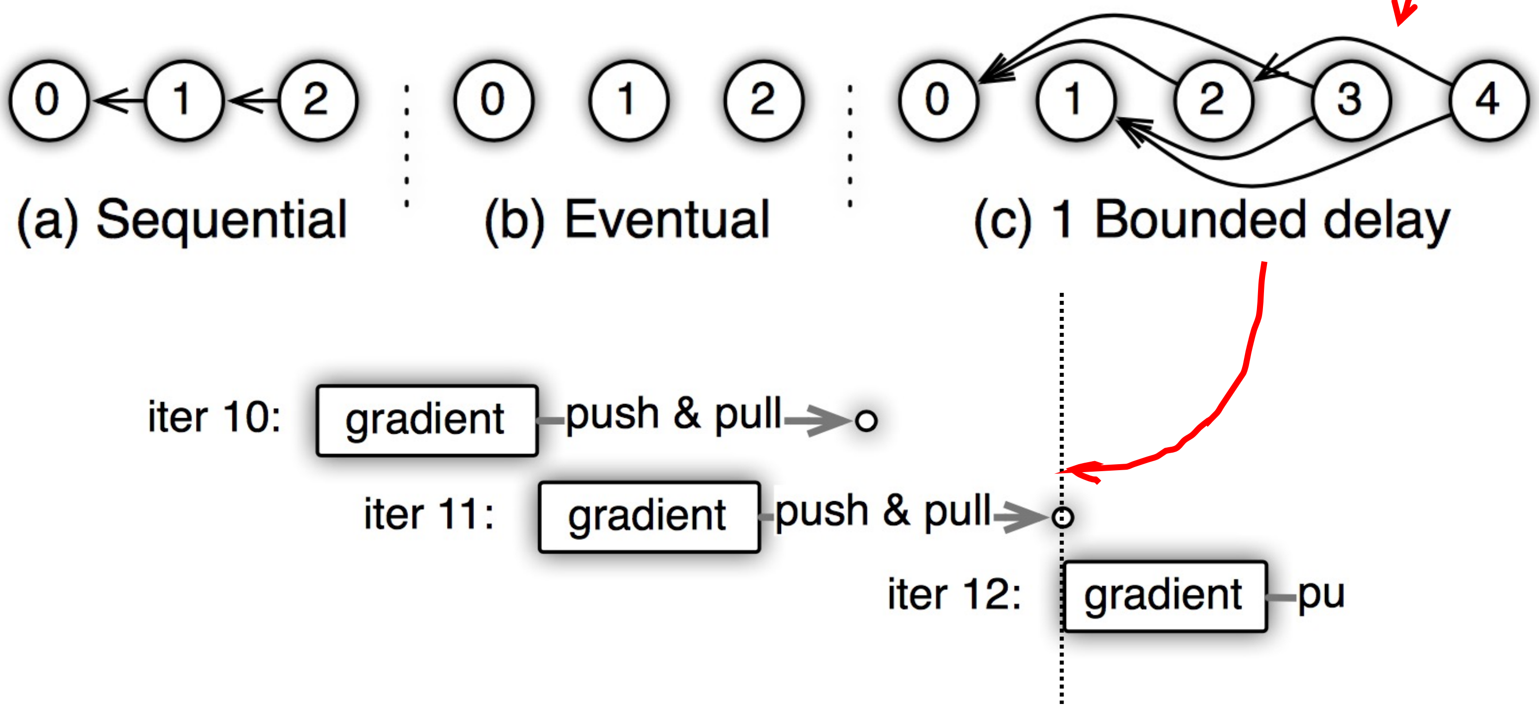
# Parameter Server (PS) as coordinator




1. Workers process local data and compute gradients **g**
2. Workers **push** gradients **g** to PS
3. PS updates parameters **w**
4. Workers **pull** updated parameters **w**

# Parameter Server (PS)

- PS consists of many nodes (parameters sharding)
- You can tune **consistency** of parameters, which allows to do certain computations **asynchronously**

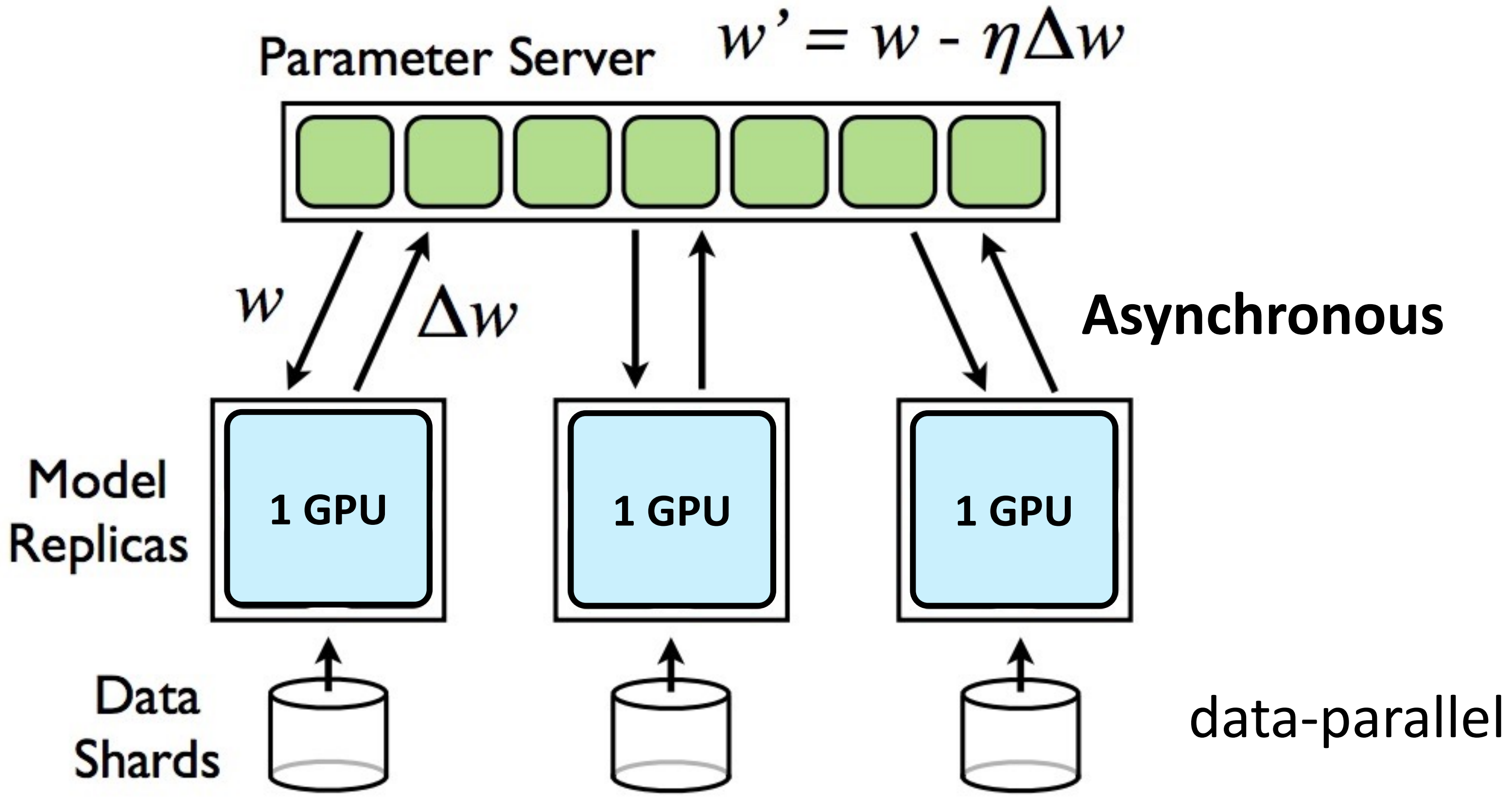



Example:
no more than
2 iterations
delay

https://www.cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf

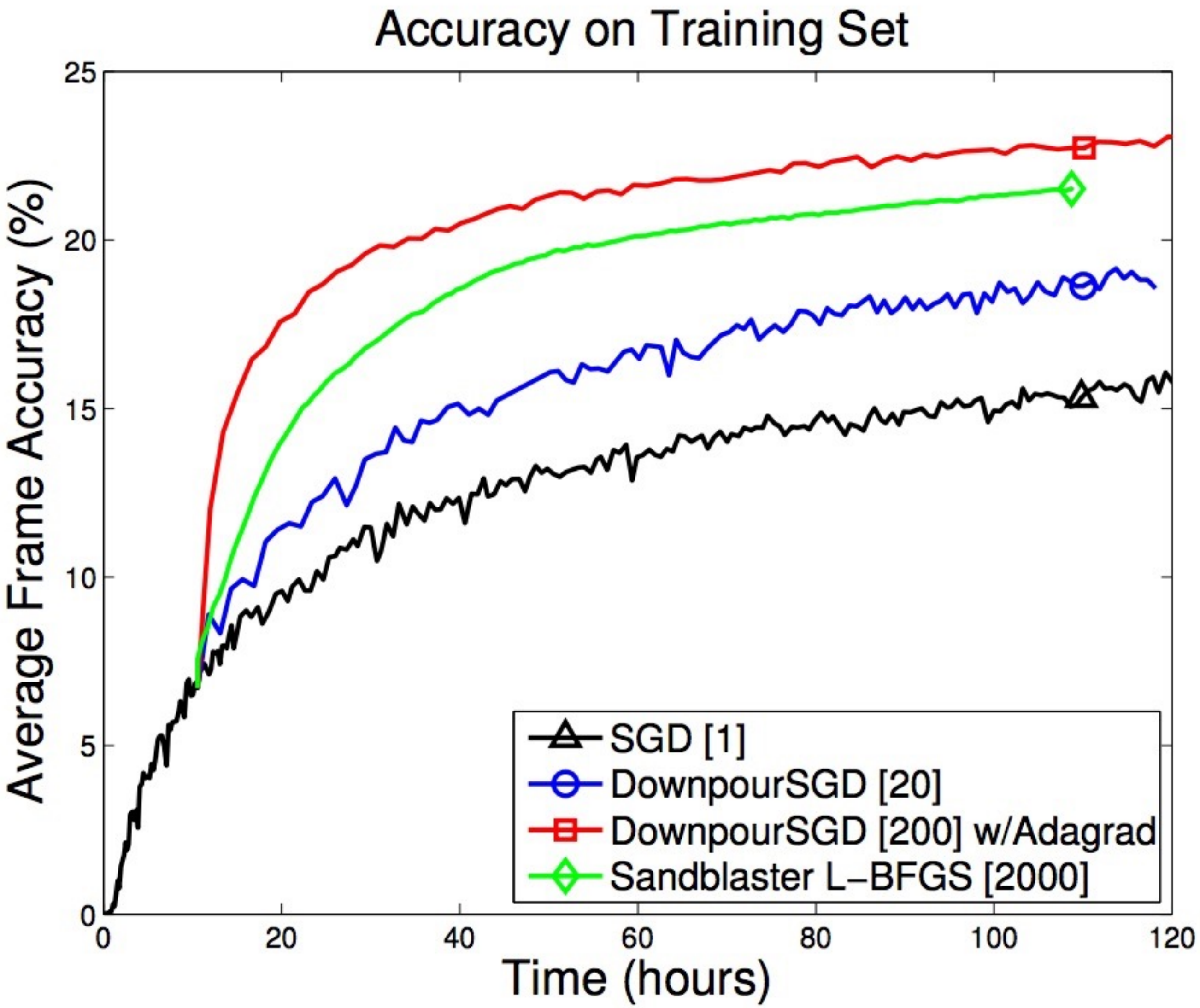
# Async SGD

- Each node in PS processes 1/7 of weights




https://storage.googleapis.com/pub-tools-public-publication-data/pdf/40565.pdf

# Async SGD from Google (DistBelief)




Async SGD only after 1 epoch of <u>synchronous</u> training

https://storage.googleapis.com/pub-tools-public-publication-data/pdf/40565.pdf

# Hogwild! for sparse tasks

- «This work aims to show using novel theoretical analysis, algorithms, and implementation that SGD can be implemented **without any locking**.»

- «We show that when the associated optimization problem is sparse, meaning most gradient updates only modify small parts of the decision variable, then Hogwild! achieves a nearly optimal rate of convergence.»

- Examples: w2v, matrix factorizations, etc.

https://arxiv.org/abs/1106.5730

# Delayed Block Proximal Gradient: bounded delay

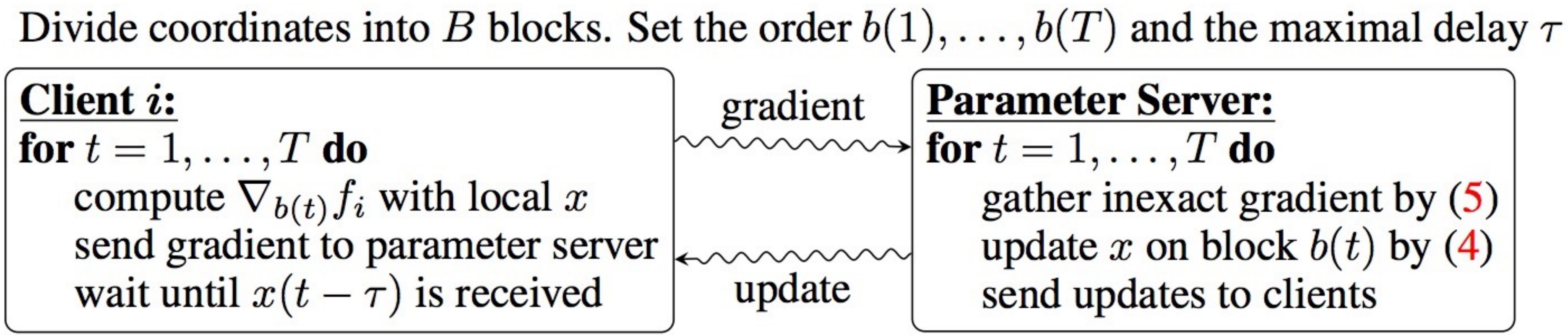



Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.

- Made for linear models (can be used for anything though)
- On iteration $t$ we update parameters in block $b(t)$
- We allow delay for no more than $\tau$ blocks

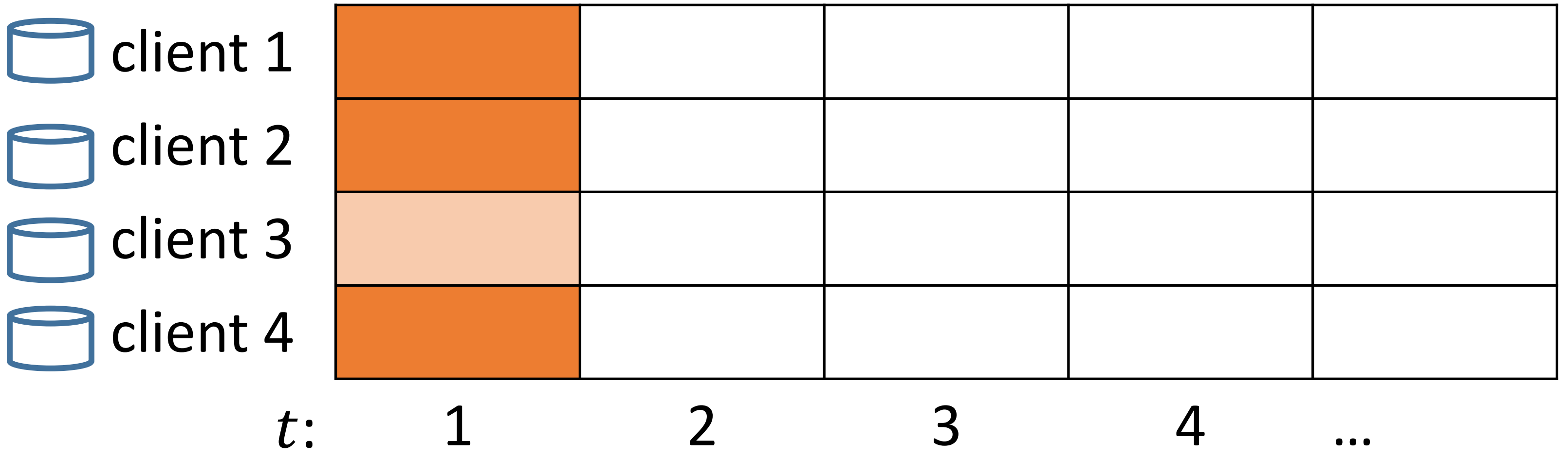
http://www.cs.cmu.edu/afs/cs/user/muli/www/file/ddp.pdf

# Delayed Block Proximal Gradient: bounded delay

Divide coordinates into $B$ blocks. Set the order $b(1), \ldots, b(T)$ and the maximal delay $\tau$

| **Client $i$:** | | **Parameter Server:** |
|---|---|---|
| **for** $t = 1, \ldots, T$ **do** | gradient $\rightsquigarrow$ | **for** $t = 1, \ldots, T$ **do** |
| compute $\nabla_{b(t)} f_i$ with local $x$ | | gather inexact gradient by (5) |
| send gradient to parameter server | $\leftsquigarrow$ update | update $x$ on block $b(t)$ by (4) |
| wait until $x(t - \tau)$ is received | | send updates to clients |

Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.



# Delayed Block Proximal Gradient: bounded delay

Divide coordinates into $B$ blocks. Set the order $b(1), \ldots, b(T)$ and the maximal delay $\tau$

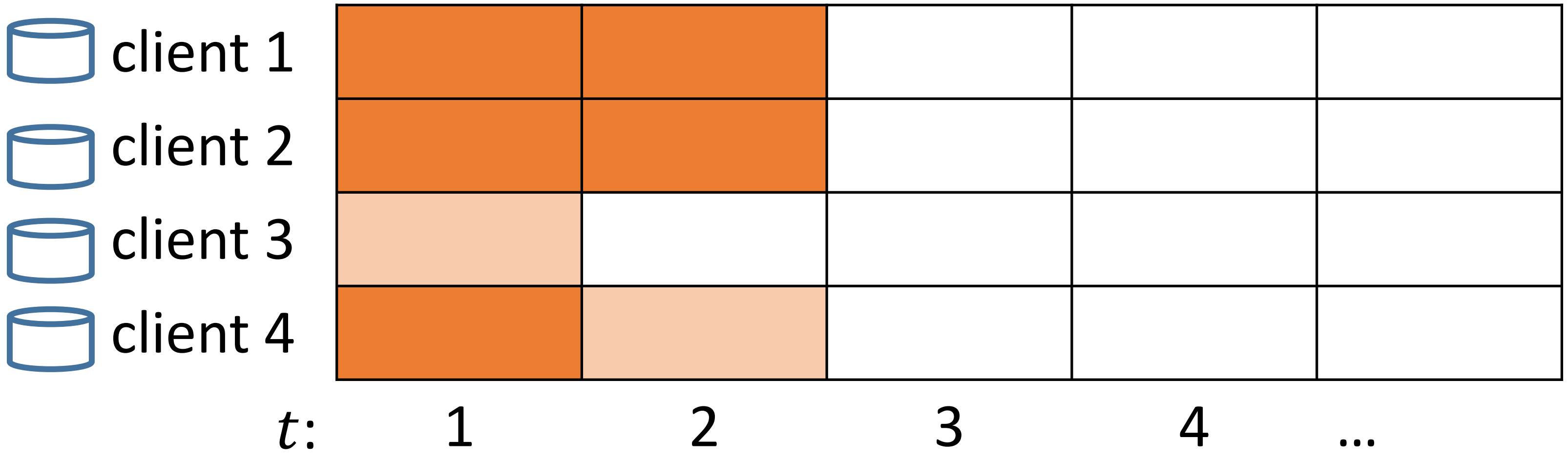
**Client $i$:**
**for** $t = 1, \ldots, T$ **do**
  compute $\nabla_{b(t)} f_i$ with local $x$
  send gradient to parameter server
  wait until $x(t-\tau)$ is received

gradient

update

**Parameter Server:**
**for** $t = 1, \ldots, T$ **do**
  gather inexact gradient by (5)
  update $x$ on block $b(t)$ by (4)
  send updates to clients

Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.



# Delayed Block Proximal Gradient: bounded delay

Divide coordinates into $B$ blocks. Set the order $b(1), \ldots, b(T)$ and the maximal delay $\tau$

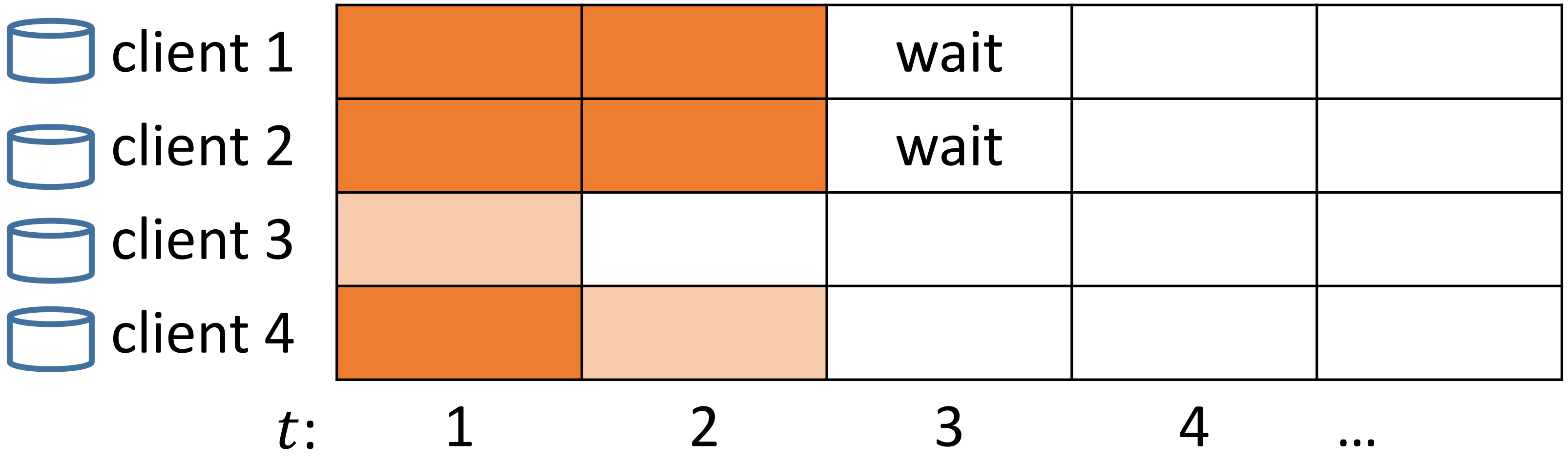
**Client $i$:**
**for** $t = 1, \ldots, T$ **do**
  compute $\nabla_{b(t)} f_i$ with local $x$
  send gradient to parameter server
  wait until $x(t - \tau)$ is received

gradient

update

**Parameter Server:**
**for** $t = 1, \ldots, T$ **do**
  gather inexact gradient by (5)
  update $x$ on block $b(t)$ by (4)
  send updates to clients

Figure 1: D2P, Distributed Delayed Proximal Gradient Methods. Both clients and the parameter server span several machines. All data sending and receiving are non-blocking.

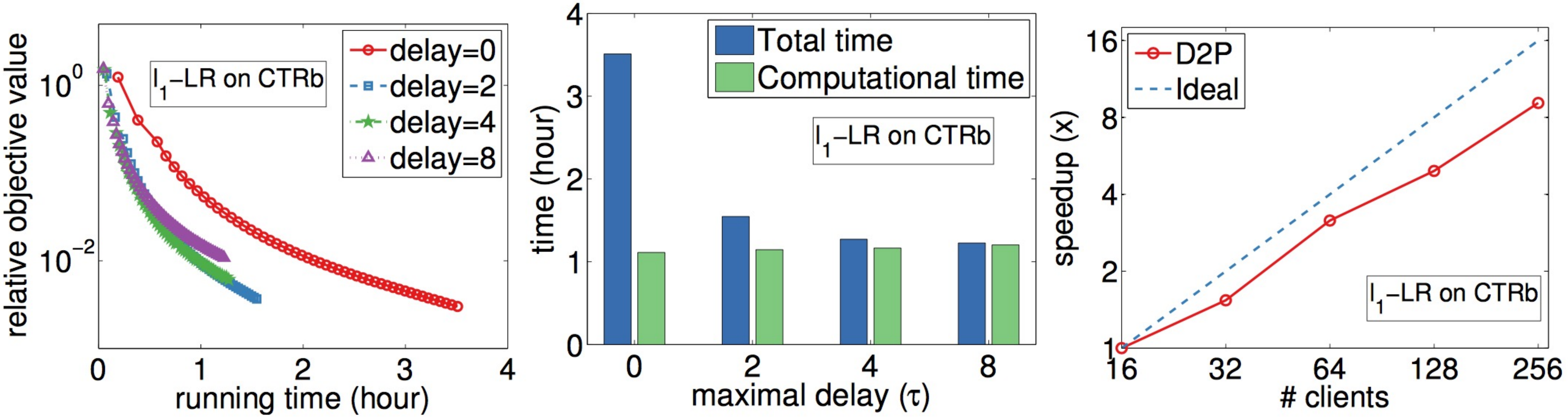



$\tau = 2$ means we have to wait

http://www.cs.cmu.edu/afs/cs/user/muli/www/file/ddp.pdf

# Delayed Block Proximal Gradient: bounded delay

- Speedup **compensates** slower convergence




http://opt.kyb.tuebingen.mpg.de/papers/opt2013_submission_1.pdf

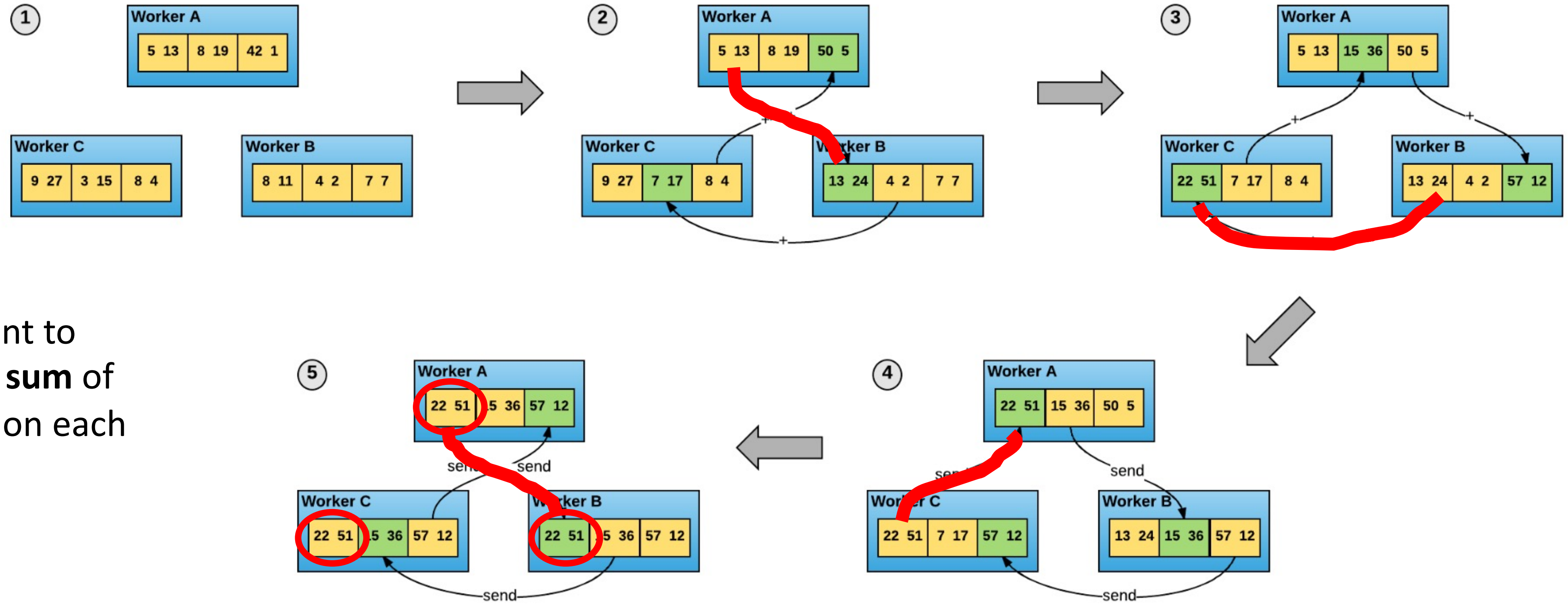
# Sync SGD: Horovod (ring AllReduce for gradients)

- Data-parallel (split batch)
- All interactions are peer-to-peer
- Virtually no barriers (efficiently utilizes network and GPU)

https://eng.uber.com/horovod/

# Sync SGD: Horovod (ring AllReduce for gradients)

- Basically we exchange every block in a loop
- Follow the ~~white~~ red rabbit:



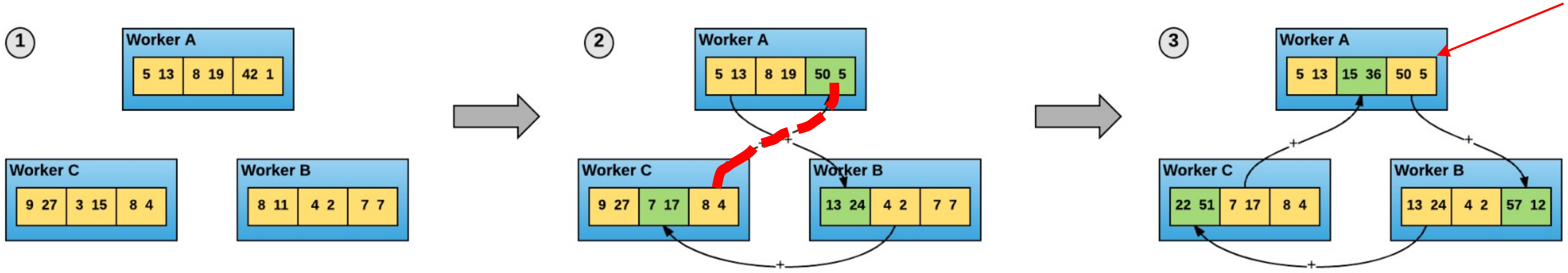

We want to have a **sum** of values on each node

Similar to tree AllReduce in VW

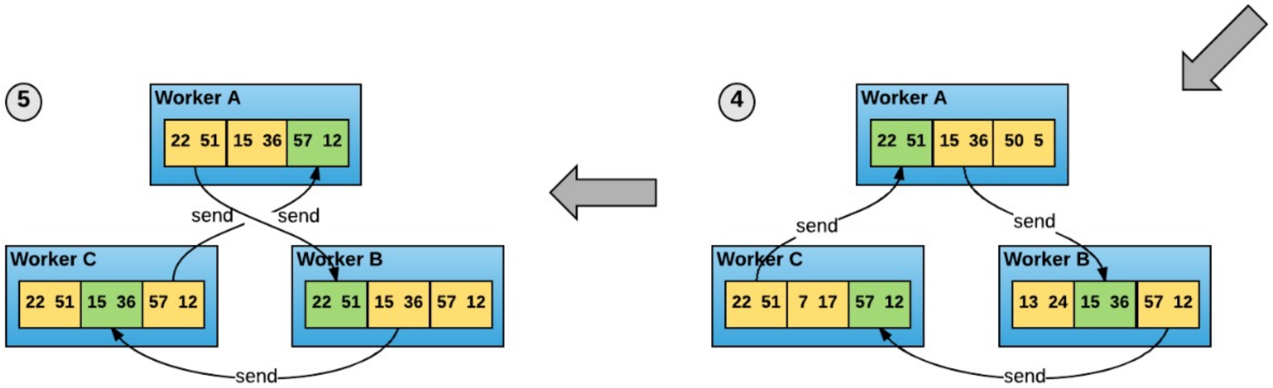
# Sync SGD: Horovod (ring AllReduce for gradients)

- Basically we exchange every block in a loop
- Follow the ~~white~~ red rabbit:

This block will wait,
but others continue



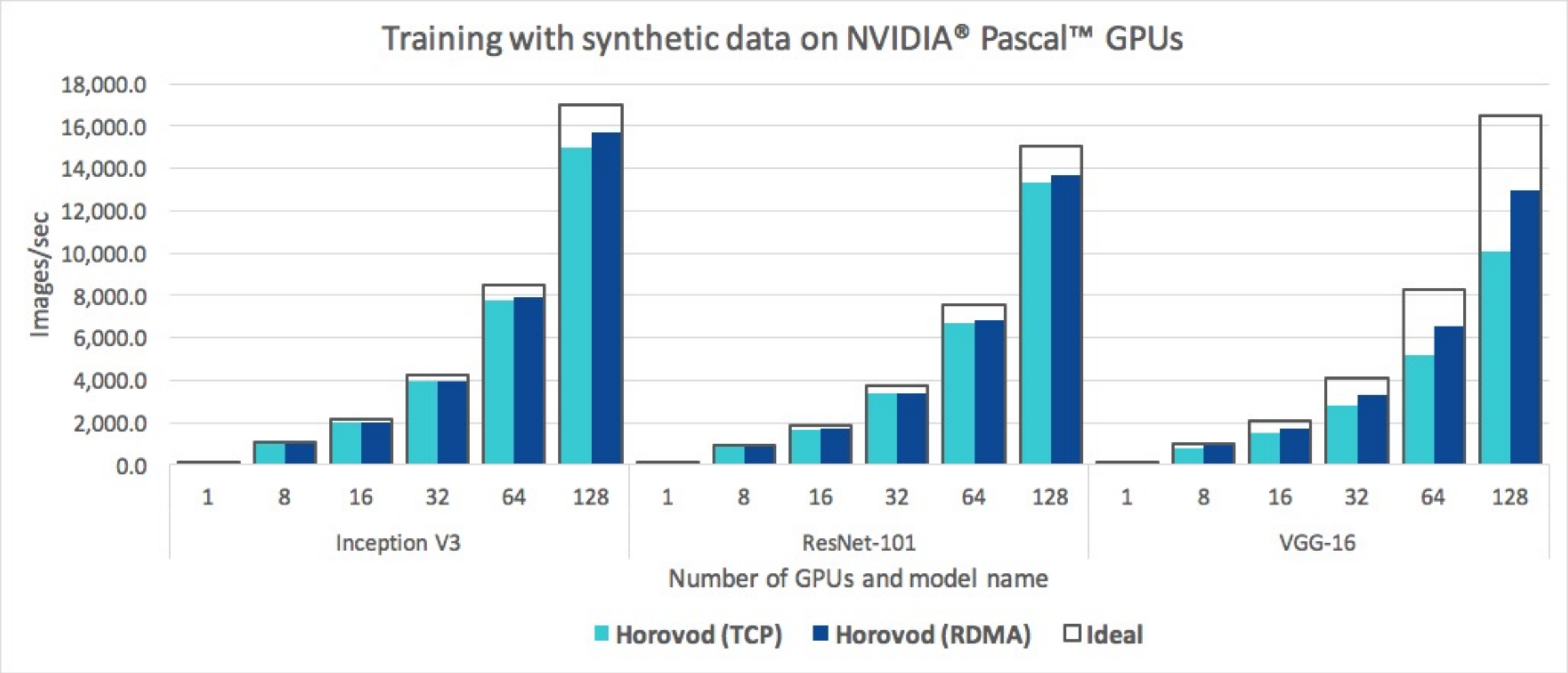

We want to
have a **sum** of
values on each
node




Similar to tree AllReduce in VW

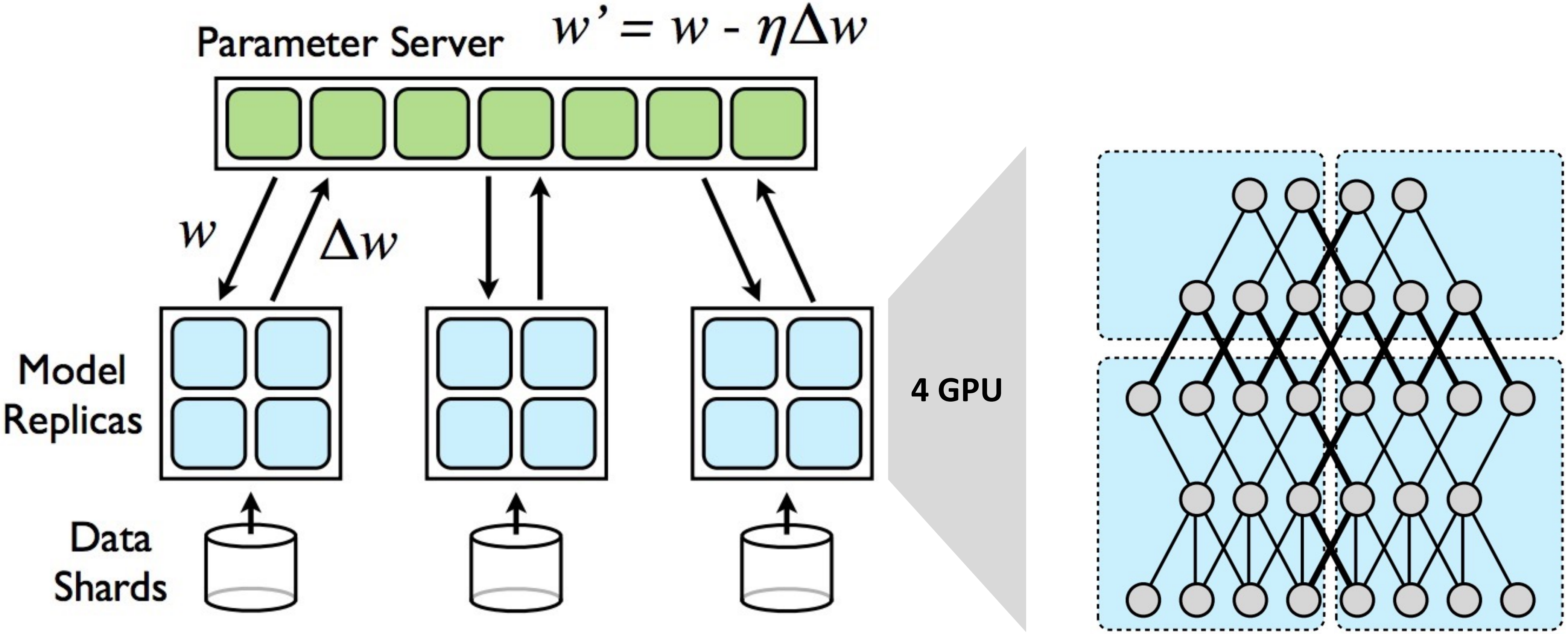
# Sync SGD: Horovod

- Scales really well:



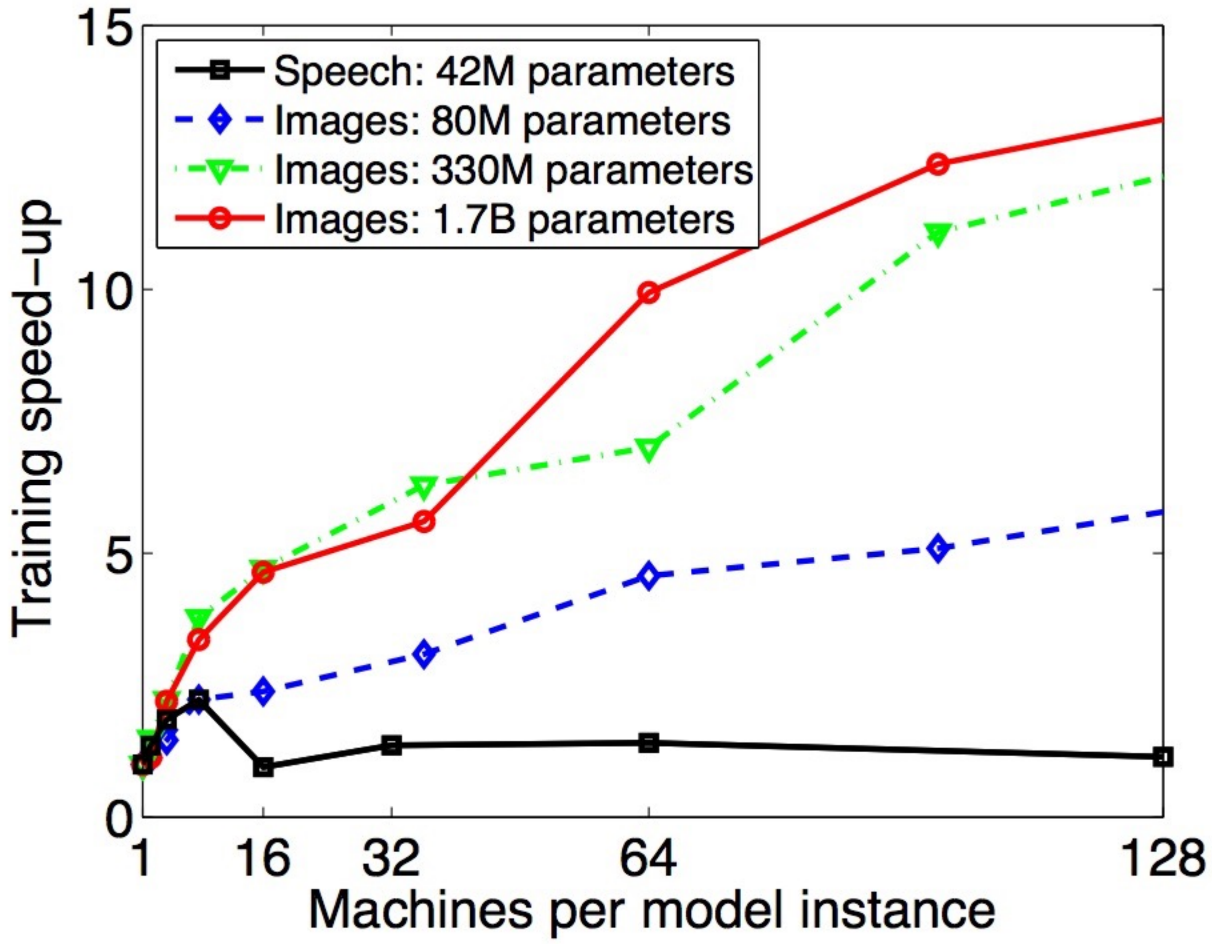
# Model-parallel (exotic use cases)

- Model weights are sharded across GPUs




https://storage.googleapis.com/pub-tools-public-publication-data/pdf/40565.pdf

# Model-parallel in DistBelief




Works OK for CNNs: beats state-of-the-art on huge ImageNet

# Model-parallel

- You probably don’t need it
- Modern GPUs have 32 GB of RAM

# Links

- Parameter Server
  https://www.cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf
- More PS
  http://opt.kyb.tuebingen.mpg.de/papers/opt2013_submission_1.pdf
- Hogwild!
  https://arxiv.org/pdf/1106.5730.pdf