# An Overview of Distributed Deep Learning

11th Vienna Deep Learning Meetup

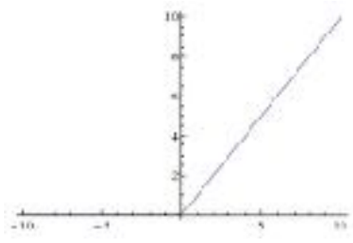
Peter Ruch

# Deep Learning, what got it started?

- large amounts of labeled data:
  *ImageNet, MS-Coco, …*


Source: http://www.image-net.org/

- new building blocks and techniques:
  *ReLU, Dropout, …*



Source: https://cs231n.github.io

- increase of computational power + availability of specialised hardware that fits computational characteristics of Deep Learning well
  *GPUs*


Source: nvidia.de

# And now?

- new application areas will generate vast amounts of new data
  *large datasets in medicine; self-driving cars generate ~ 0.75 GB/s* [1]

- larger and deeper architectures
  *ILSVRC`12 - AlexNet 8 Layers vs. ILSVRC`15 - ResNet 152 Layers*

one model —> tens of exaFLOPs!

1 http://www.kurzweilai.net/googles-self-driving-car-gathers-nearly-1-gbsec

# a short refresher

cat - 1
dog - 0
lion - 0
.
.
.

cat - 0.78
dog - 0.35
lion - 0.89
.
.
.

CrossEntropy

forward pass

backward pass

Source: wikimedia.org

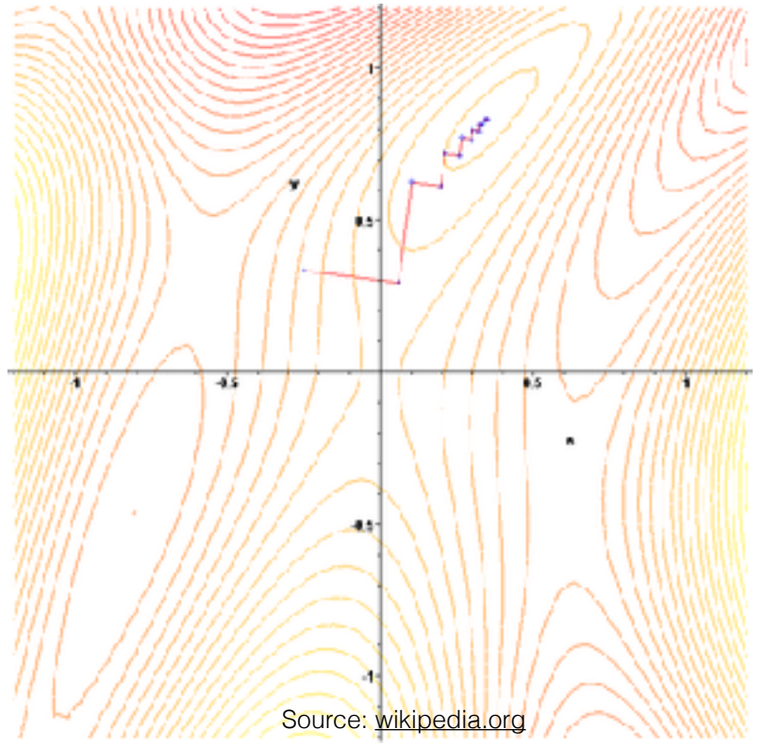
$\Delta P$

gradients

# a short refresher

## gradient descent

$$P_{new} = P - \lambda \, \Delta P$$



Source: wikipedia.org

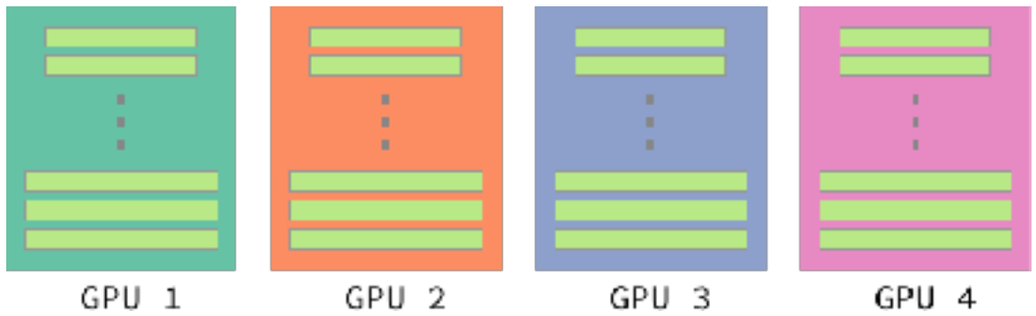
batch gradient descent: $P_{new} = P - \lambda \, \nabla_P J(P)$

stochastic gradient descent: $P_{new} = P - \lambda \, \nabla_P J(P; x^{(i)}; y^{(i)})$

minibatch gradient descent: $P_{new} = P - \lambda \, \nabla_P J(P; x^{(i:i+n)}; y^{(i:i+n)})$

# parallelisation strategies

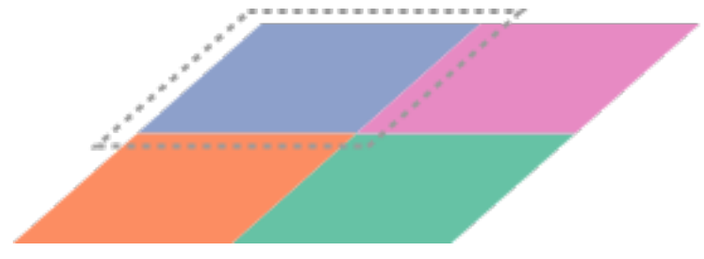
## data parallelism




activations

- transfer of gradients
- multiple models

## model parallelism



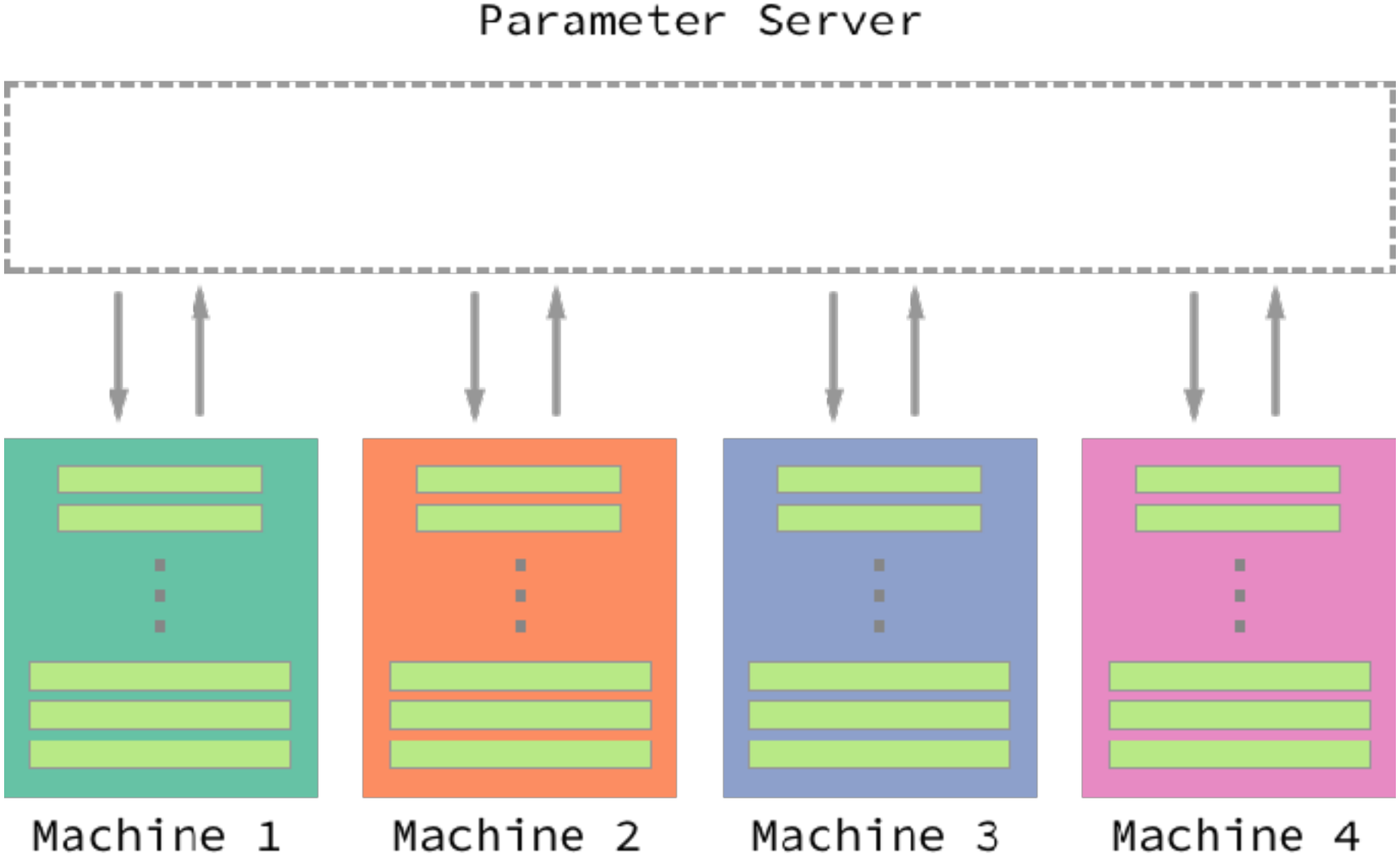
neurons

- transfer of partial activations
- model is distributed

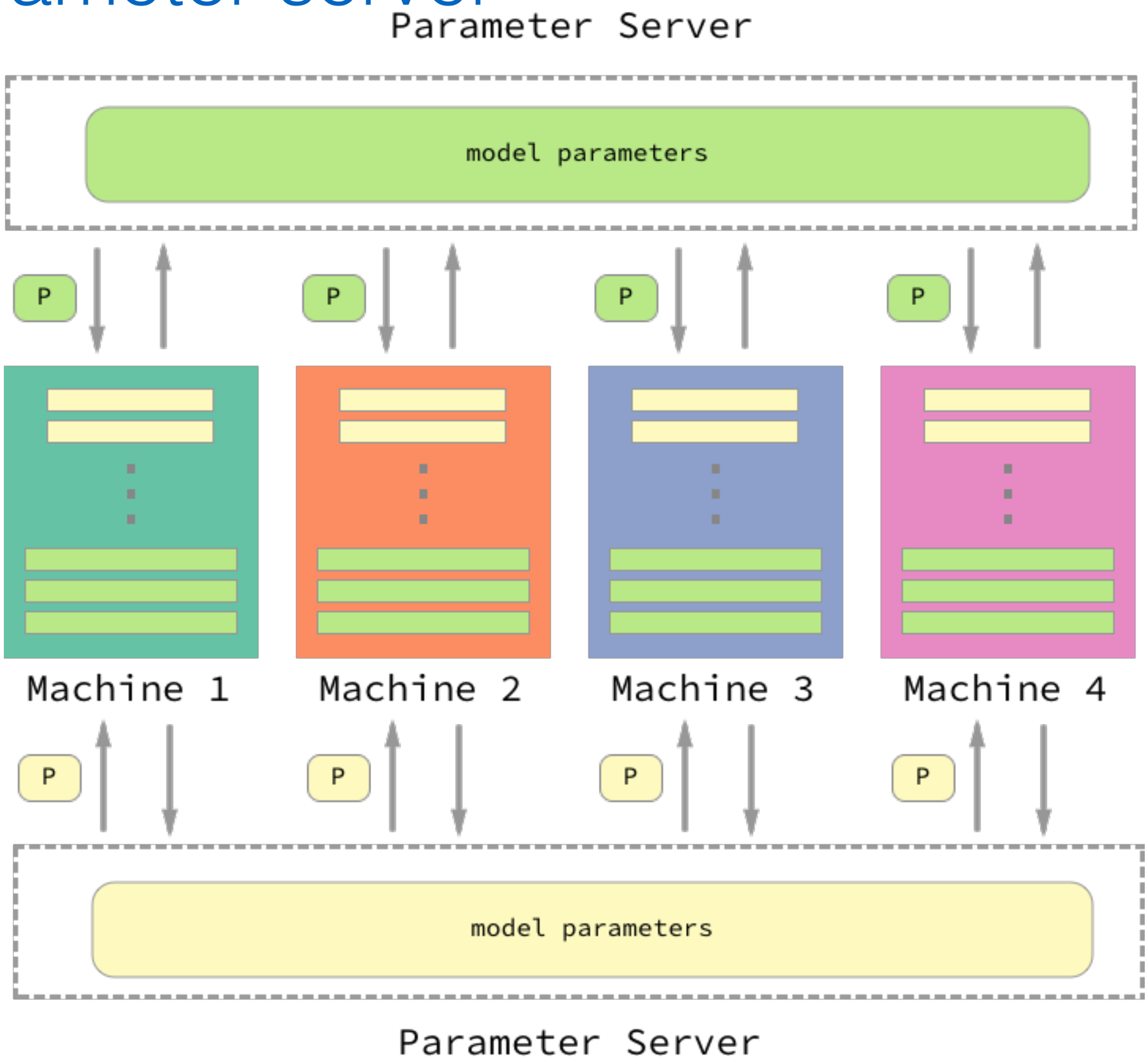
**We'll only look at the data parallel case**

# parameter server

- separate node or process that communicates with workers and is responsible for updating the model



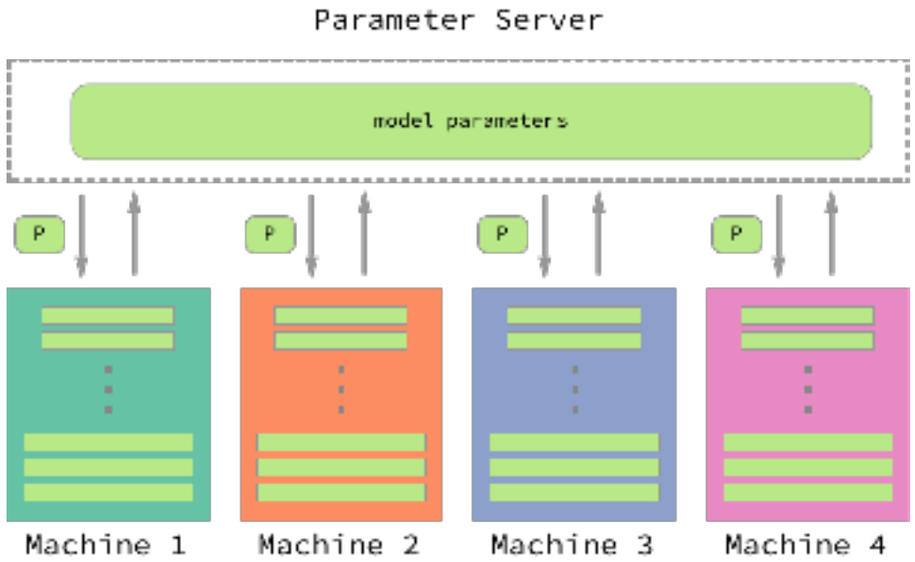
# parameter server



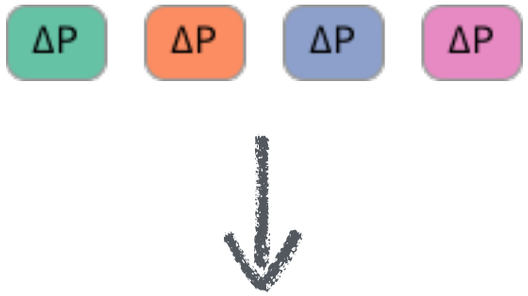
# parameter server

*update based*

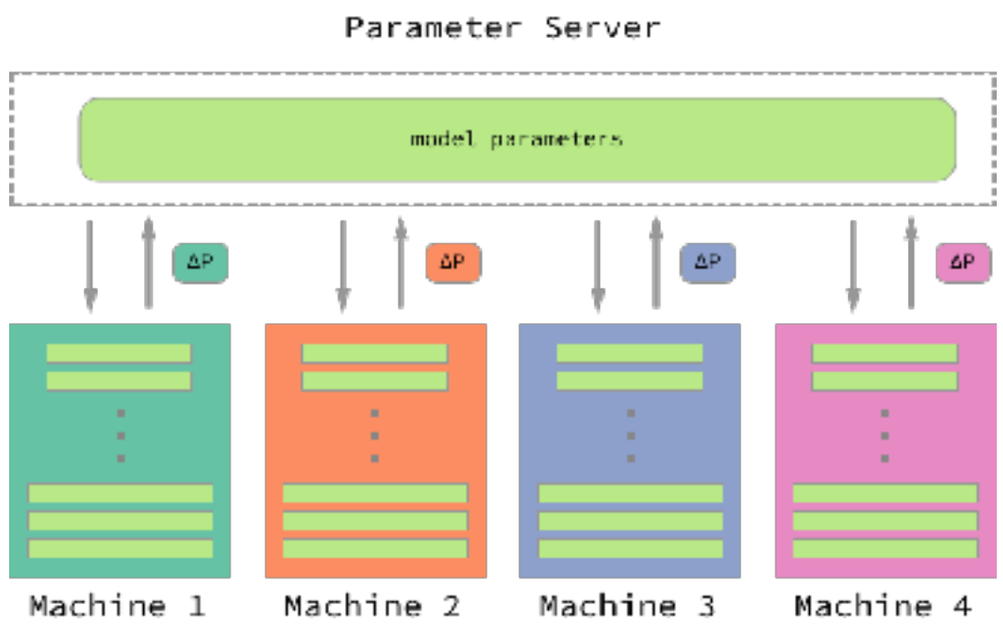
pull parameters from server




forward + backward pass through each model, and calculate gradients



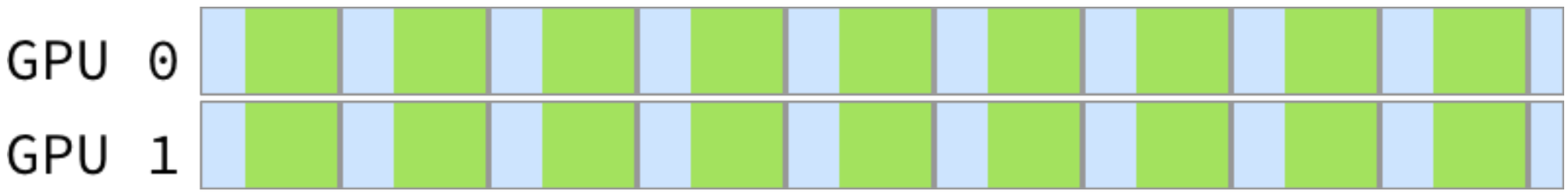

push gradients to parameter server




update parameters on server based on update strategy

(next slides)

# synchronous SGD



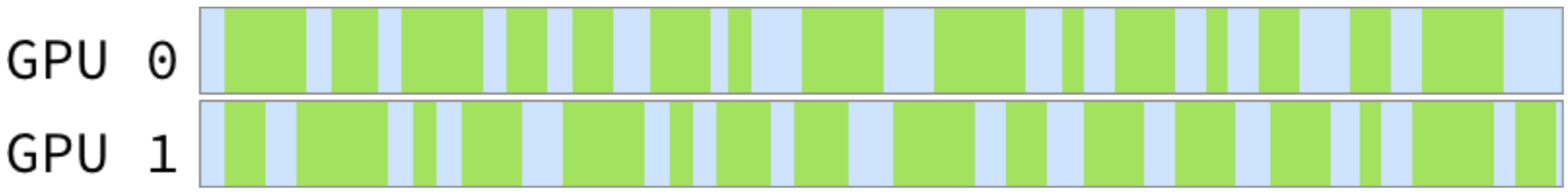

- parameters are only updated after we have gradient information from all workers

—> we have to wait for all workers to complete their minibatch (slow workers will stall everything)

- parameters are then updated using the averaged gradients

$$P_{n+1} = P_n - \lambda \frac{1}{4}(\Delta P_{n,0} + \Delta P_{n,1} + \Delta P_{n,2} + \Delta P_{n,3})$$

# asynchronous SGD



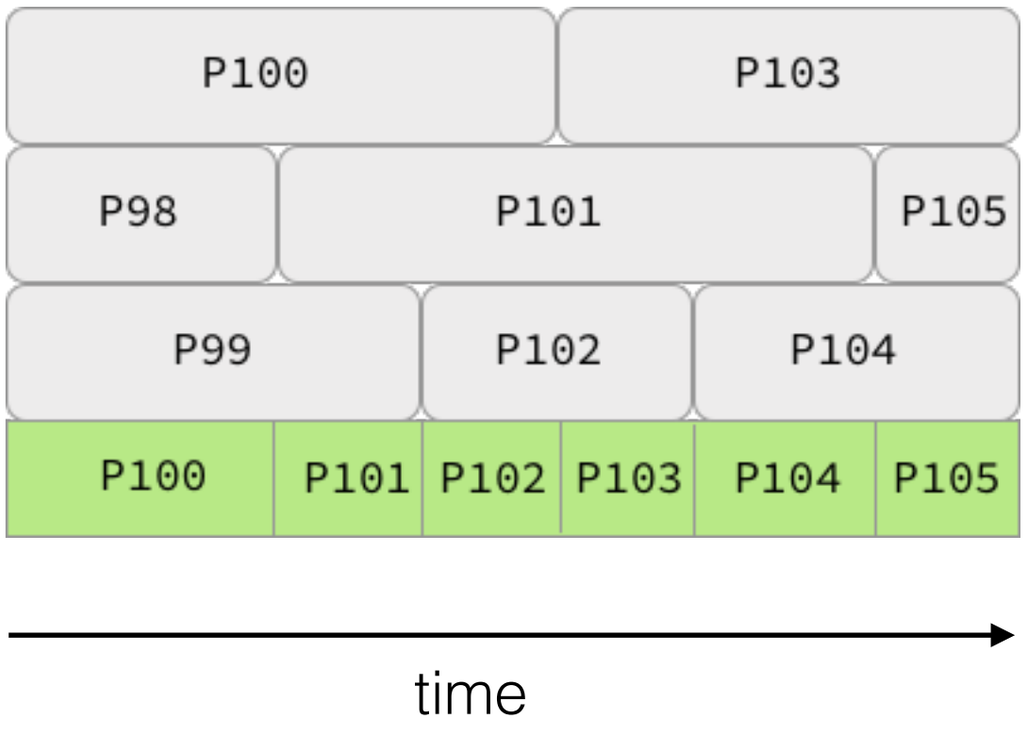

- parameters are updated as soon new gradient is available

—> every minibatch will be processed using a slightly different model

- randomness might make it difficult to reproduce experiments

—> synchronous SGD preferable!

# asynchronous SGD



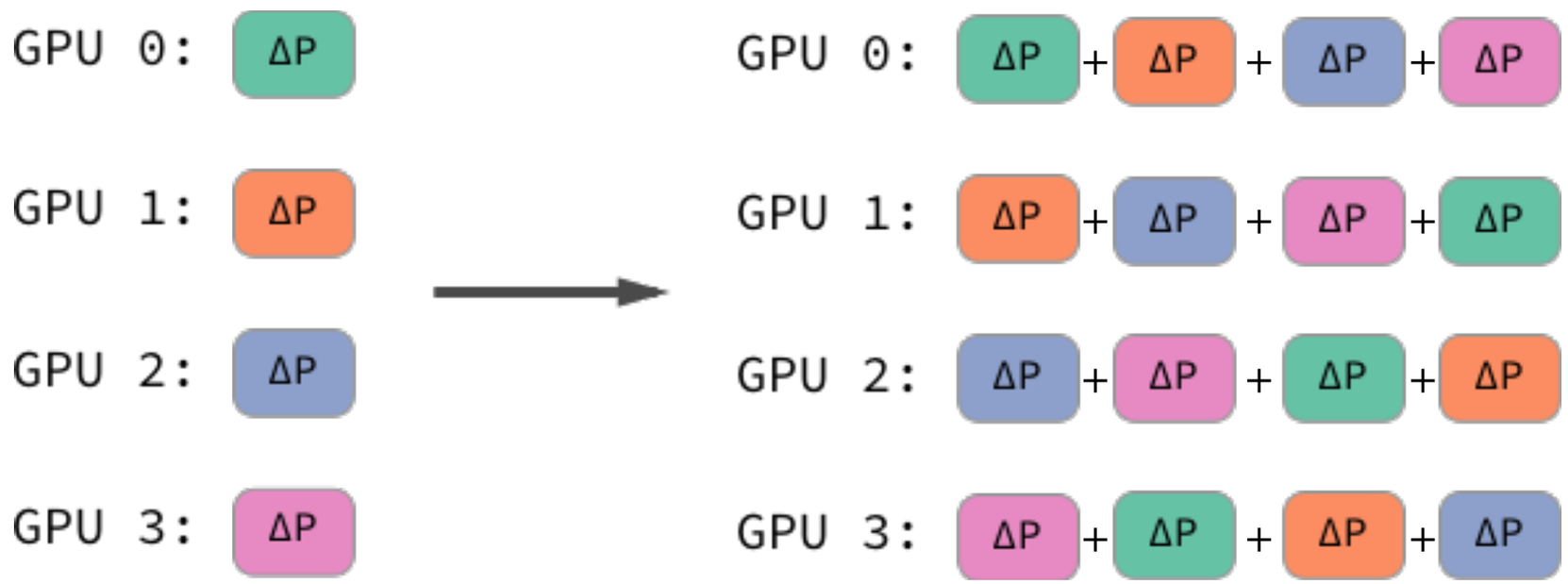

$$P_{101} = P_{100} - \lambda \Delta P_{98}$$
$$P_{102} = P_{101} - \lambda \Delta P_{99}$$
$$P_{103} = P_{102} - \lambda \Delta P_{100}$$
$$P_{104} = P_{103} - \lambda \Delta P_{102}$$
$$P_{105} = P_{104} - \lambda \Delta P_{101}$$

—> if a worker takes very long to push a gradient update the parameter server might reject it as the overall model already progressed further

# synchronous all-reduce SGD

- gradients are averaged without parameter server

—> after reduction every node has all the gradient information




—> there are multiple possible ways to implement the gradient exchange. a possible variant (ring-allreduce) is hinted here

# What’s available where?