Efficient training & inference

Deep Learning@HSE

Week {++i}, guest lecture

Yandex
Research

British Hedgehog
Preservation Society

Large problems need large models

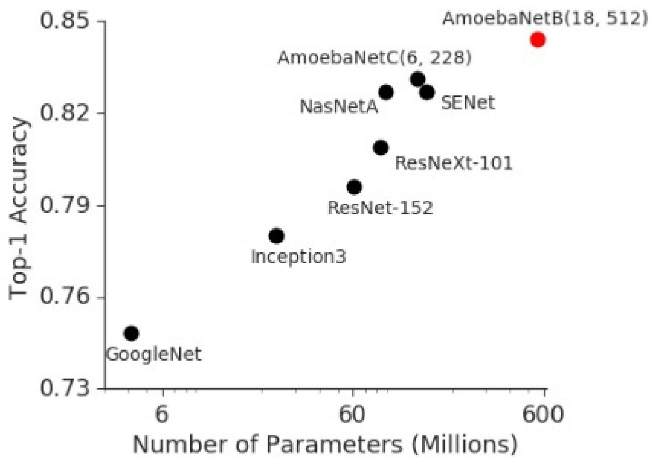
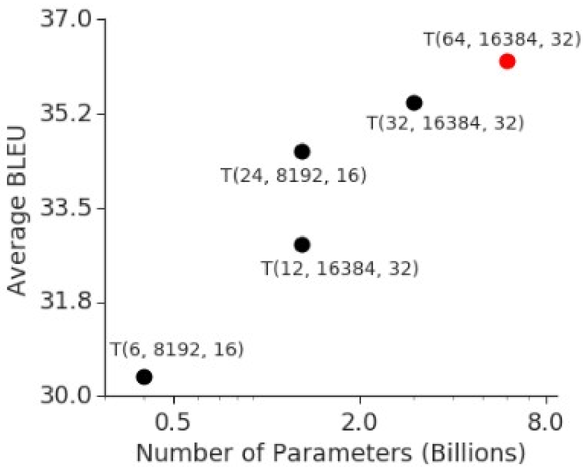


Image Classification
ImageNet

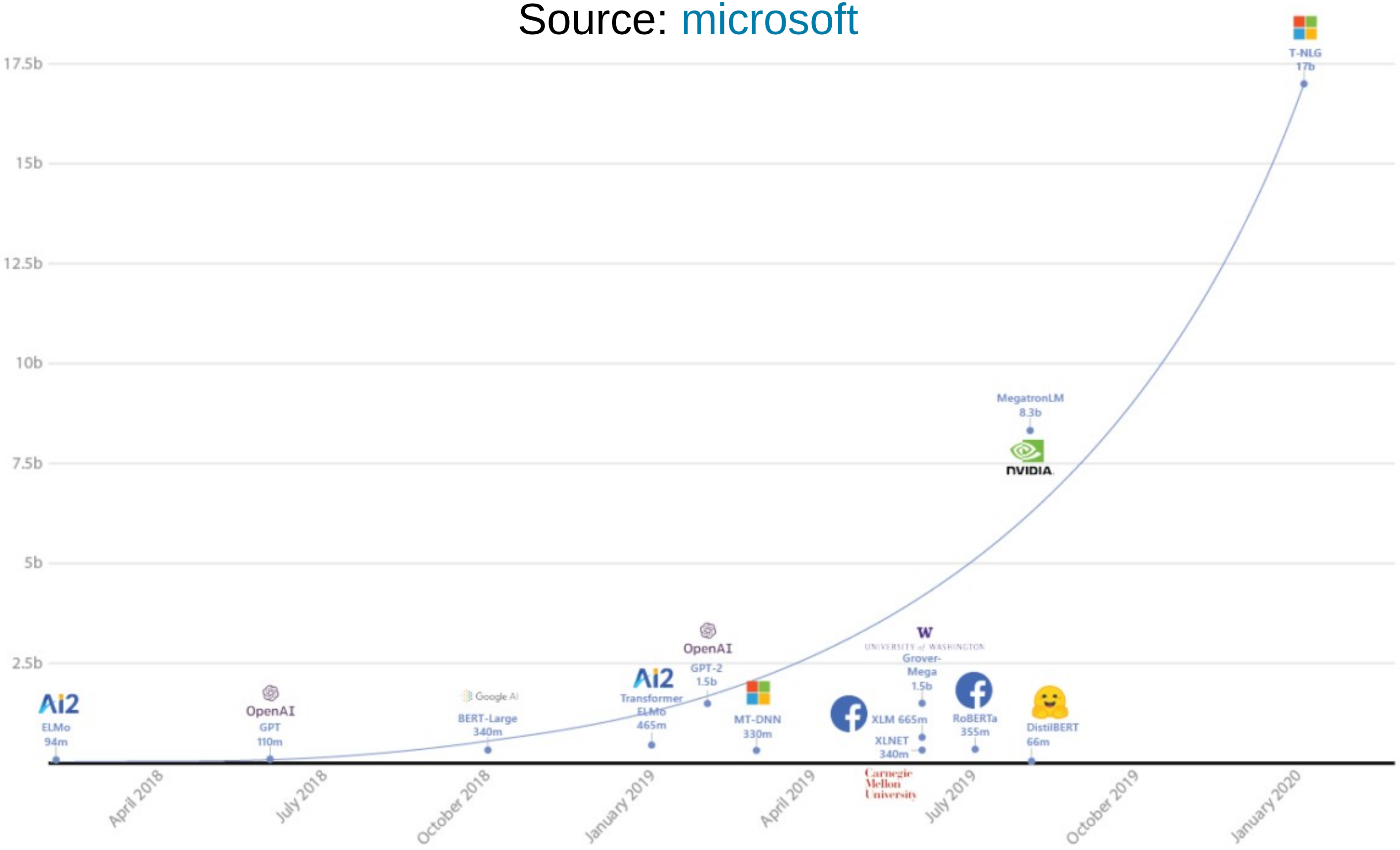


Machine Translation
average over WMT

Source: https://arxiv.org/abs/1811.06965

The transformer curve

Source: microsoft



Machine Learning Supertasks

Image classification – ImageNet, JFT300M

Generative models – ImageNet(biggan), the internet

Language Models – common crawl, BERT / MLM

Machine Translation – multilingual translation

Reinforcement Learning – playstation* & steam :)

* playstation for RL: https://arxiv.org/abs/1912.06101

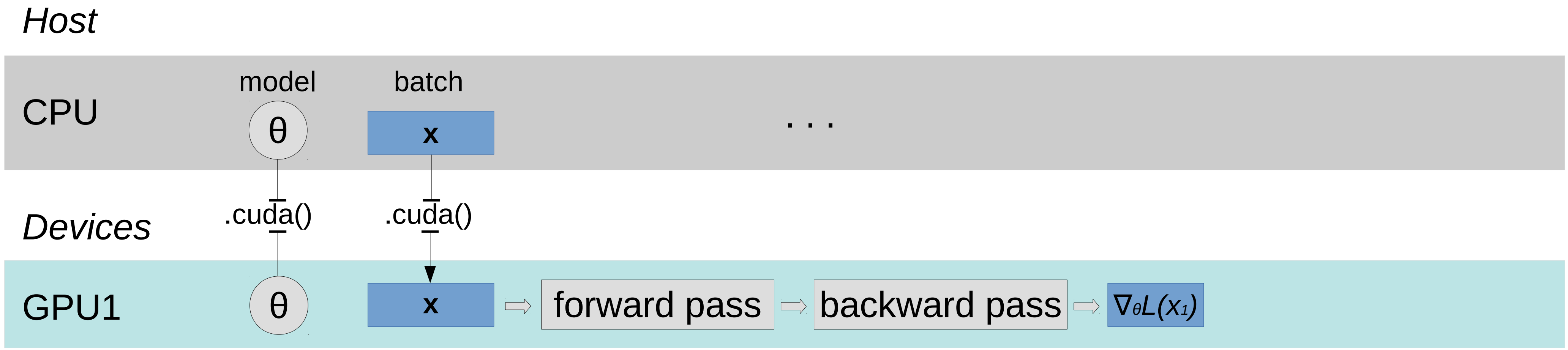
Meanwhile, exabytes of YouTube videos lay dormant across the web, waiting for someone who can make use of them

Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf

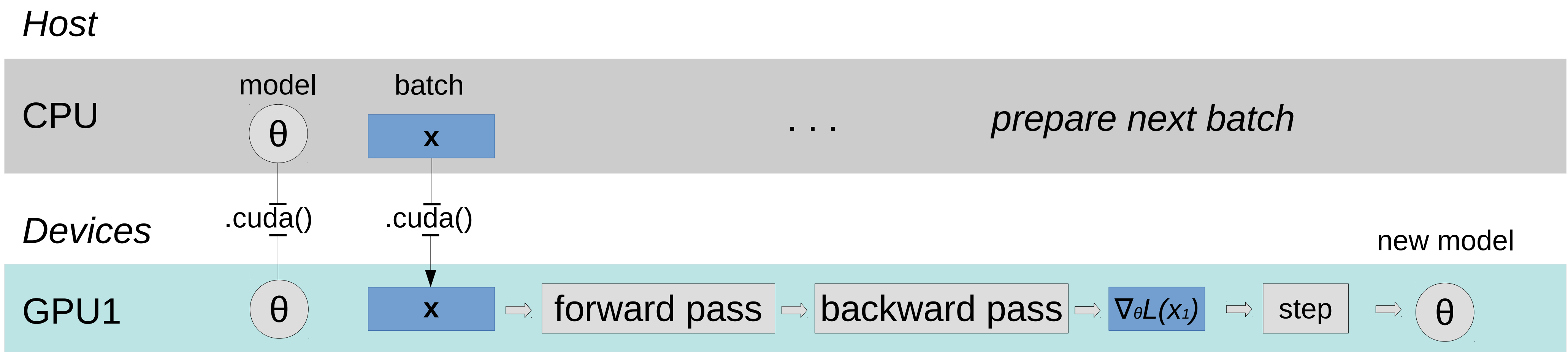
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



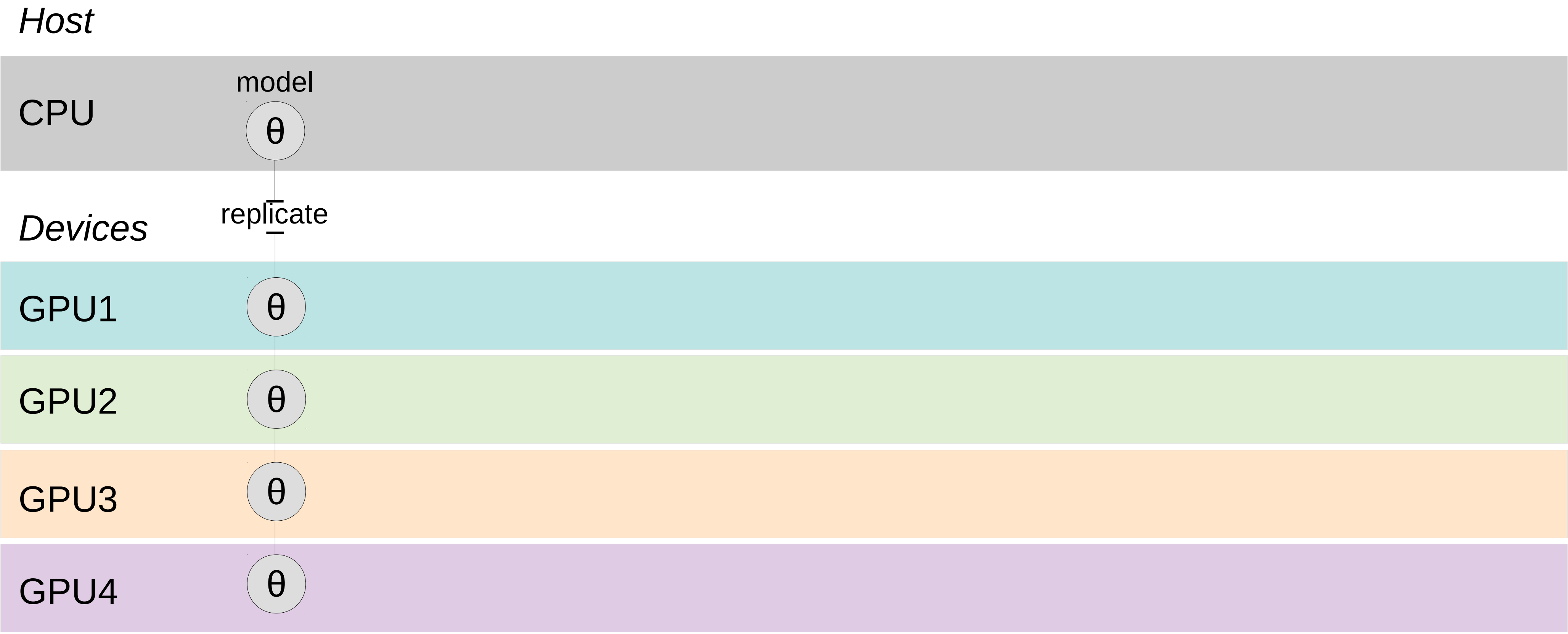
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



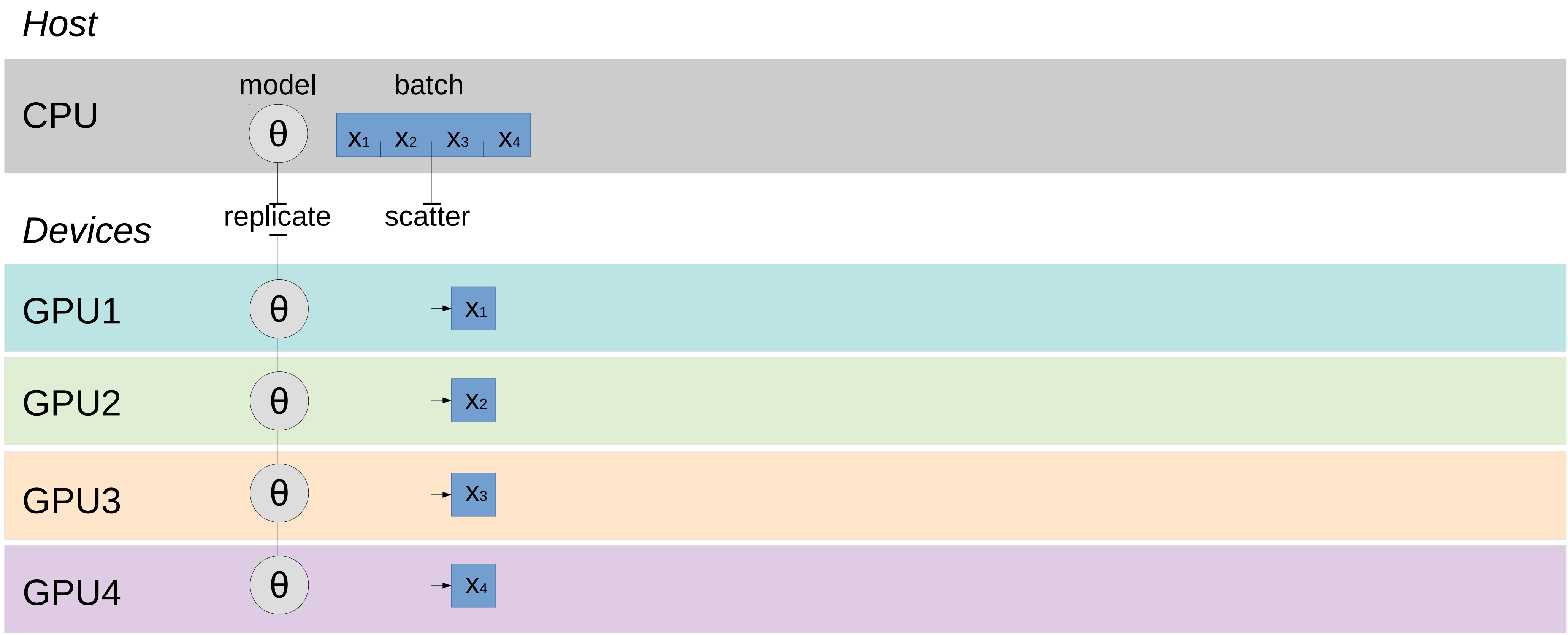
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



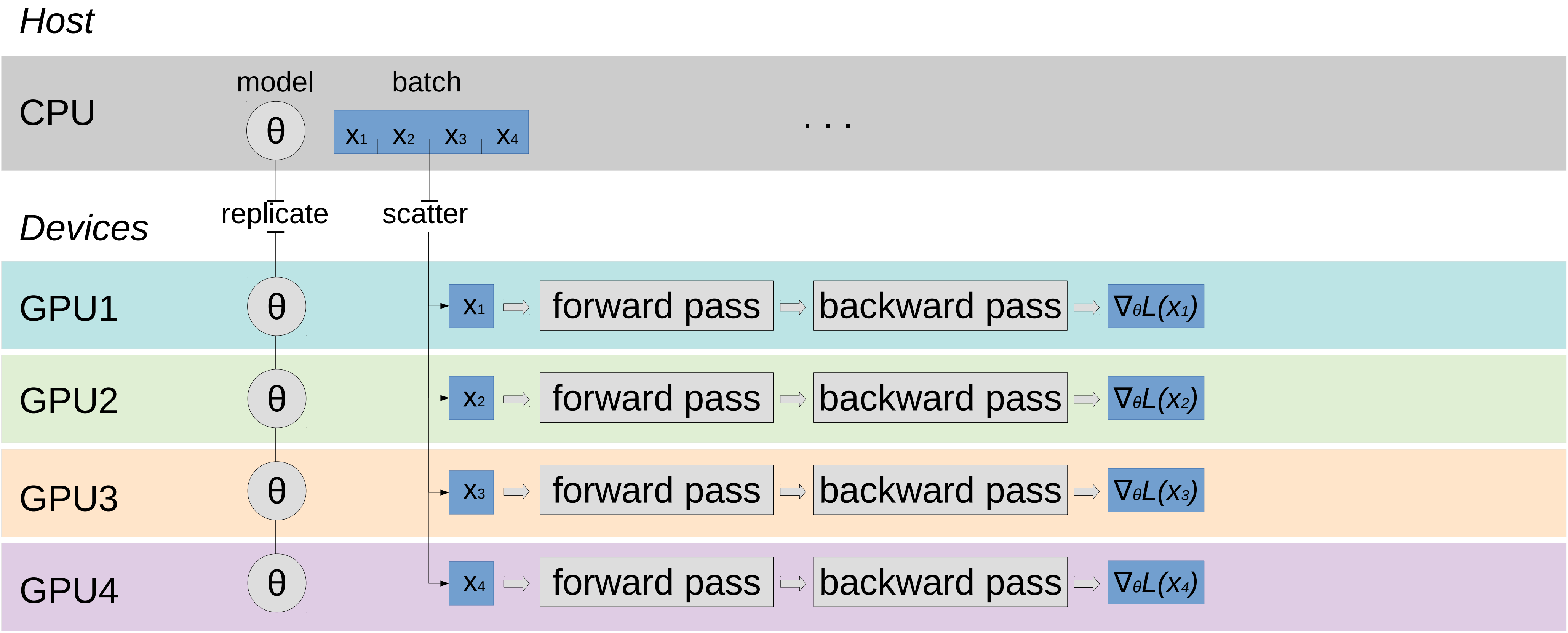
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



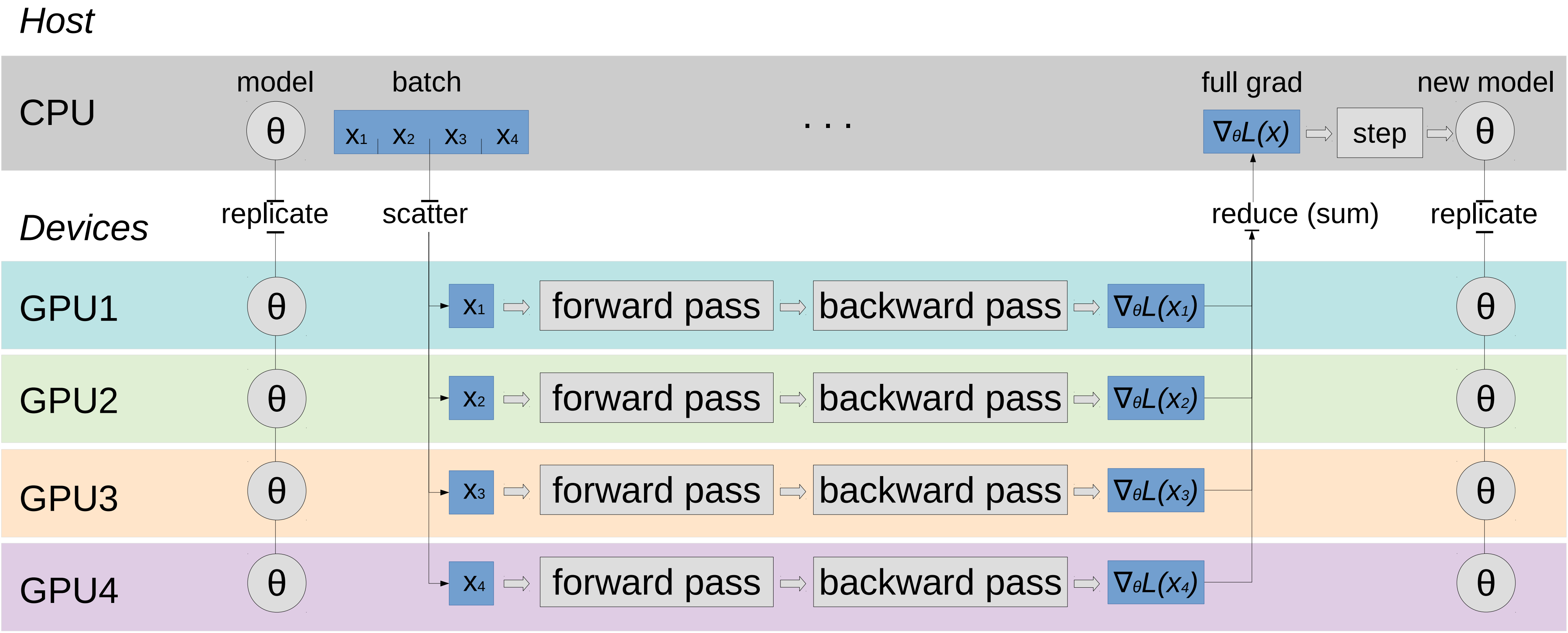
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



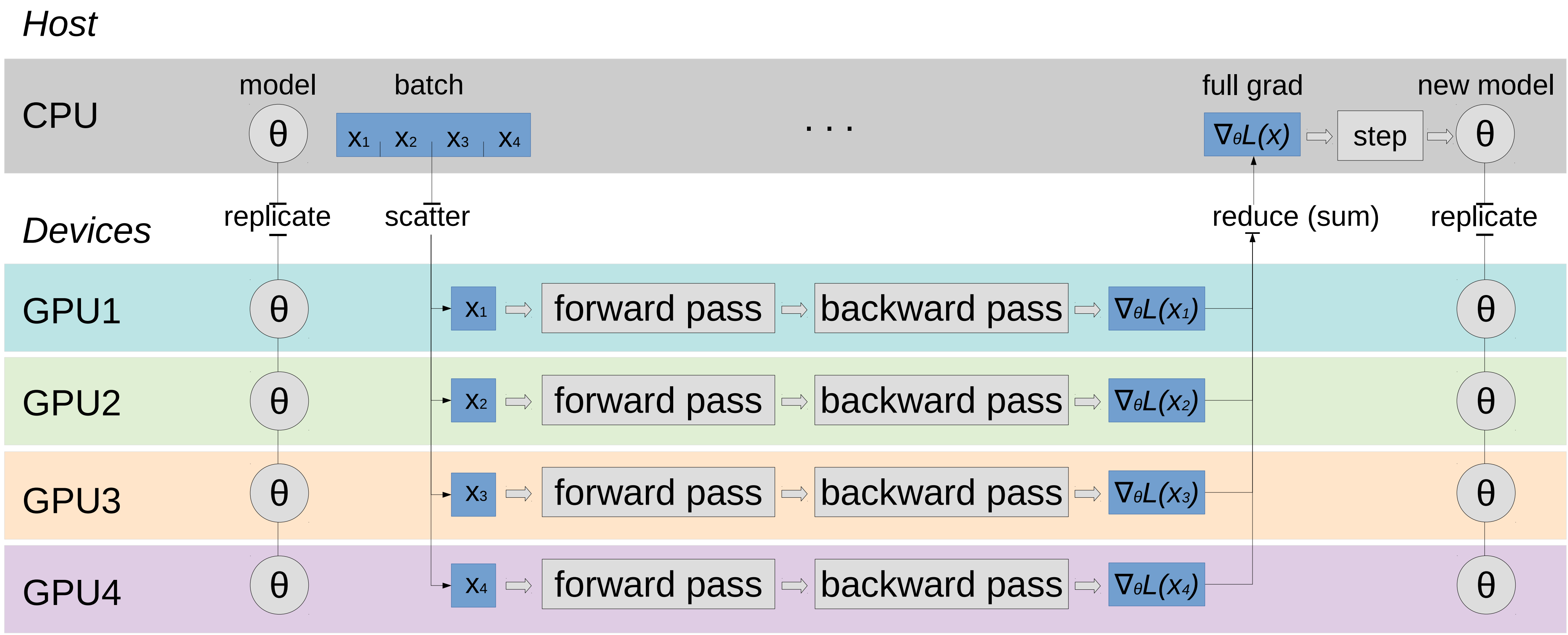
Data-parallel training (naive)

cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



Data-parallel training (naive)

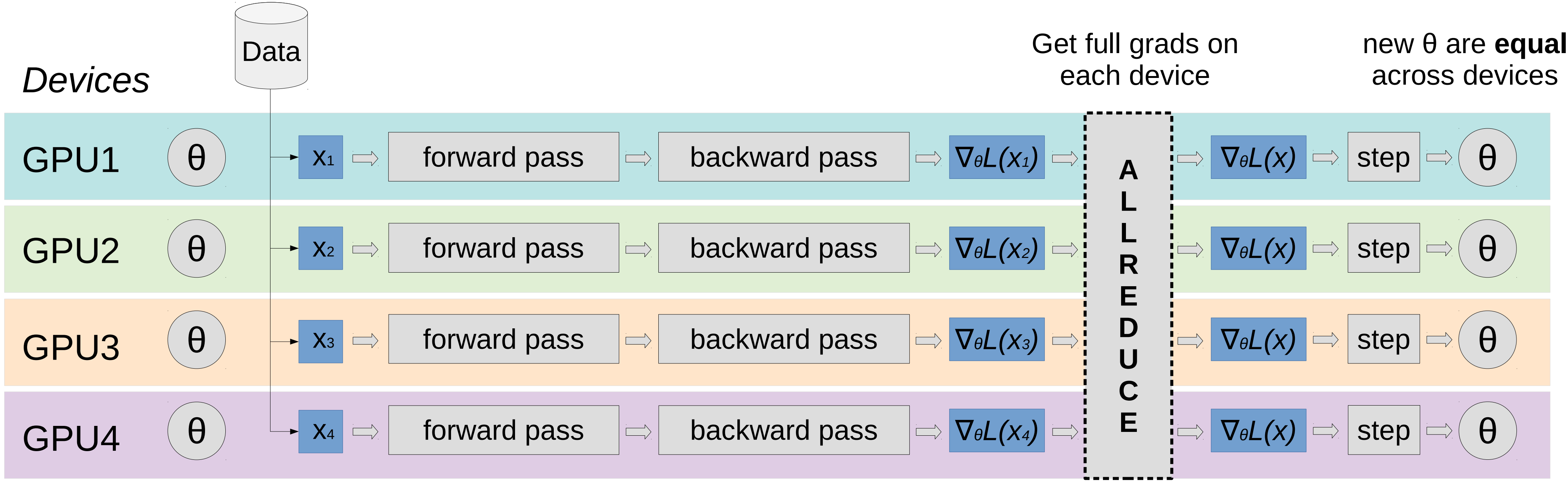
cs.cmu.edu/~muli/file/parameter_server_osdi14.pdf



Advanced data parallel

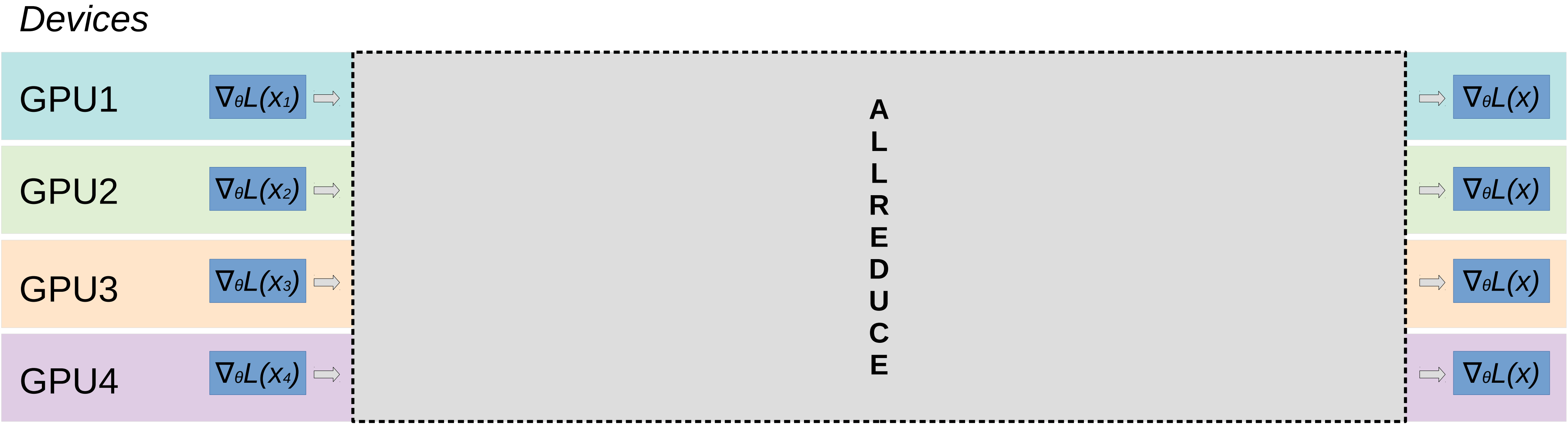
arxiv.org/abs/1706.02677

Idea: get rid of the host, each gpu runs its own computation
Q: why will weights be equal after such step?



Faster allreduce

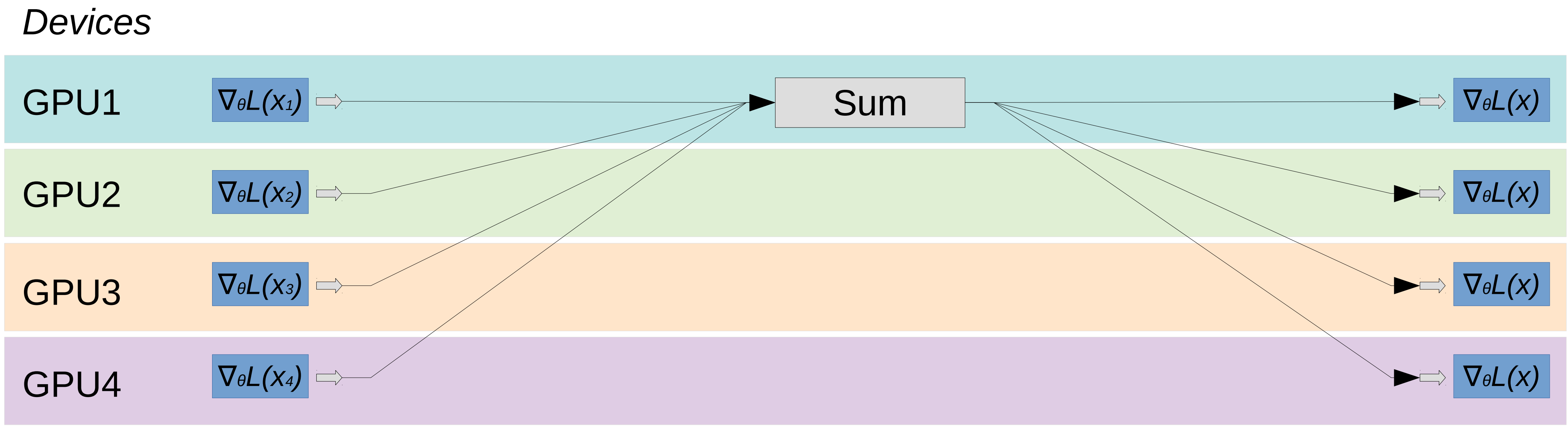
Input: each device has its its own vector
Output: each device gets a sum of all vectors



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

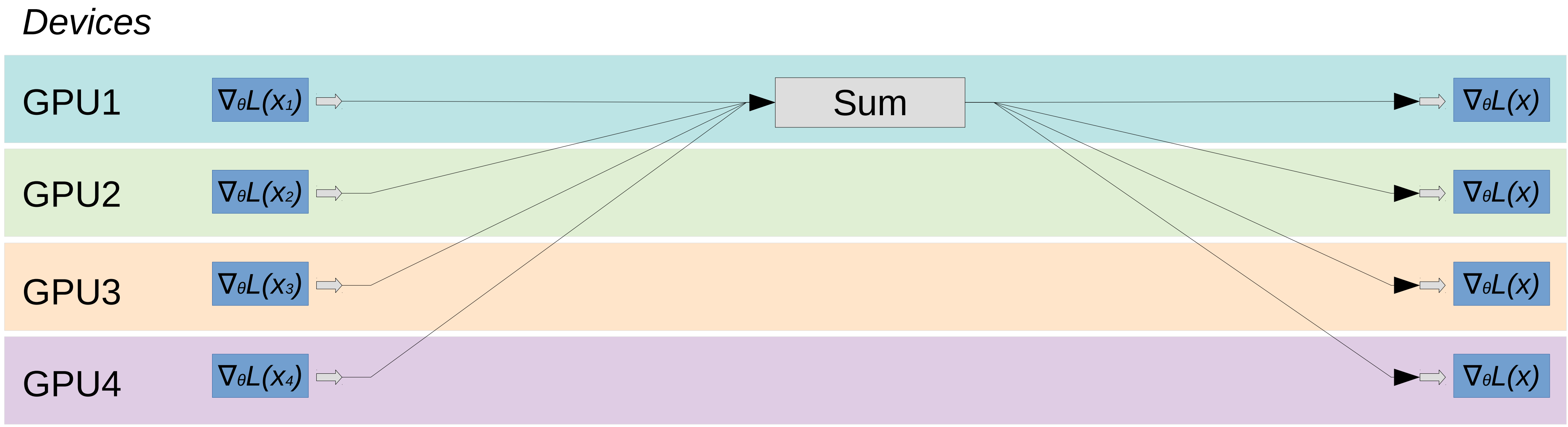
Naive implementation



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

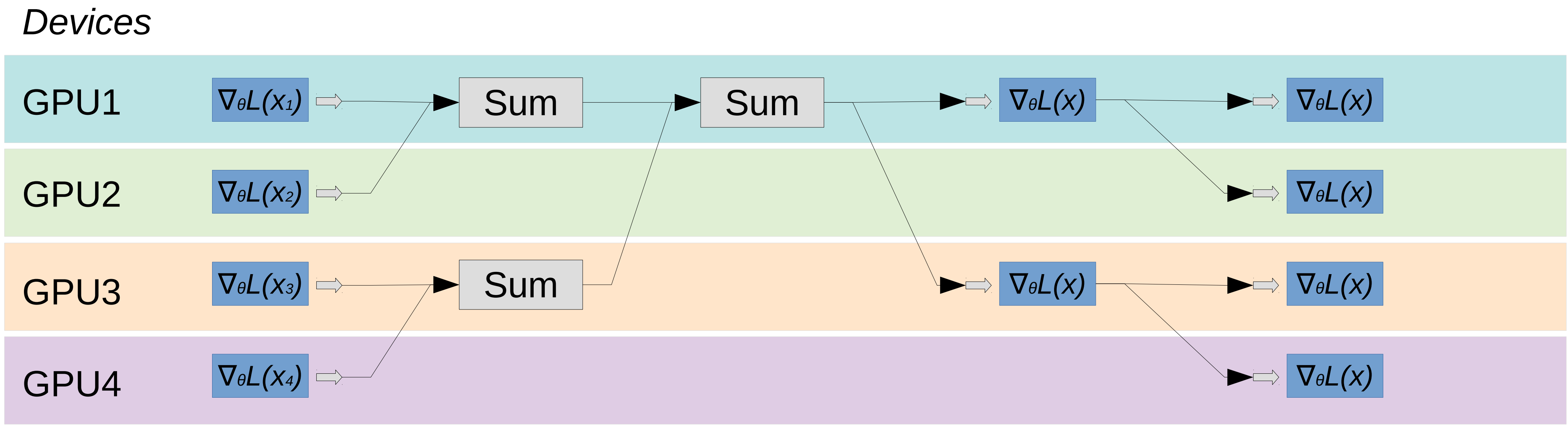
Q: Can we do better?



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

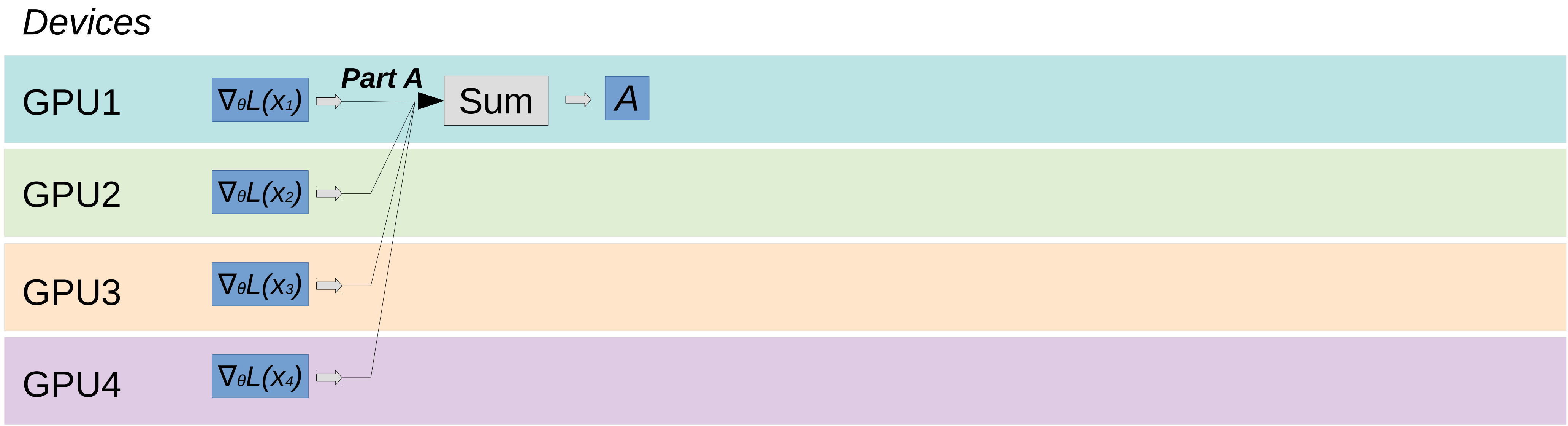
Tree-allreduce



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

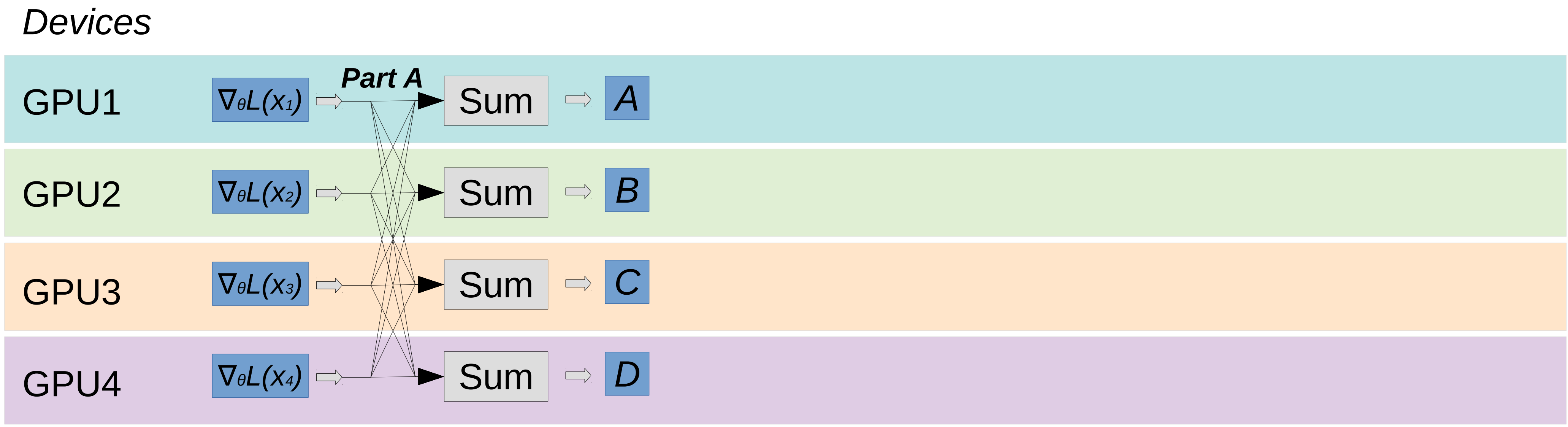
Ring-allreduce – split data into chunks (ABCD)



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

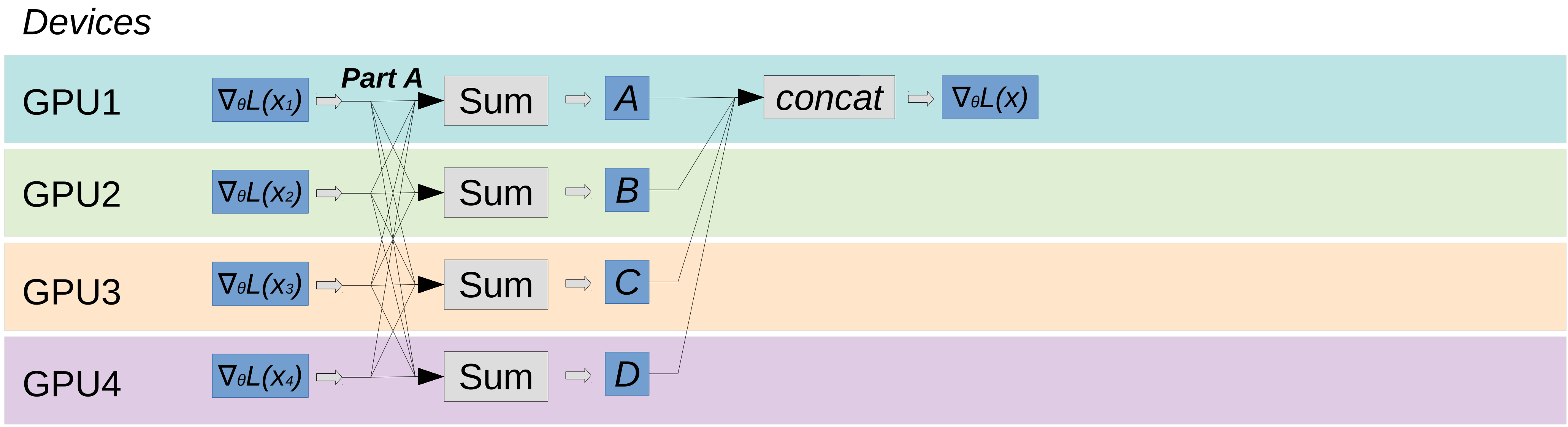
Ring-allreduce – split data into chunks (ABCD)



Faster allreduce

Input: each device has its its own vector
Output: each device gets a sum of all vectors

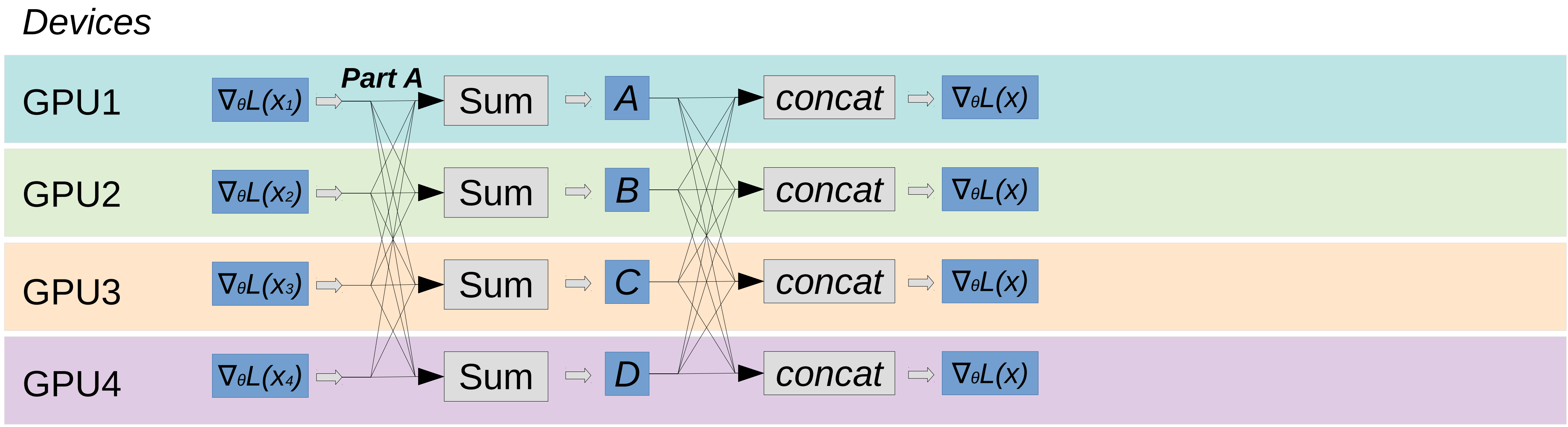
Ring-allreduce – split data into chunks (ABCD)



Faster allreduce

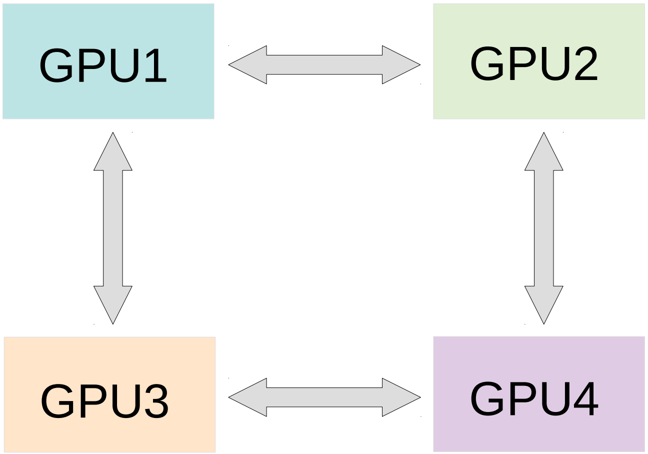
Input: each device has its its own vector
Output: each device gets a sum of all vectors

Ring-allreduce – split data into chunks (ABCD)



Ring allreduce

Bonus quest: you can only send data between **adjacent** gpus



Ring topology

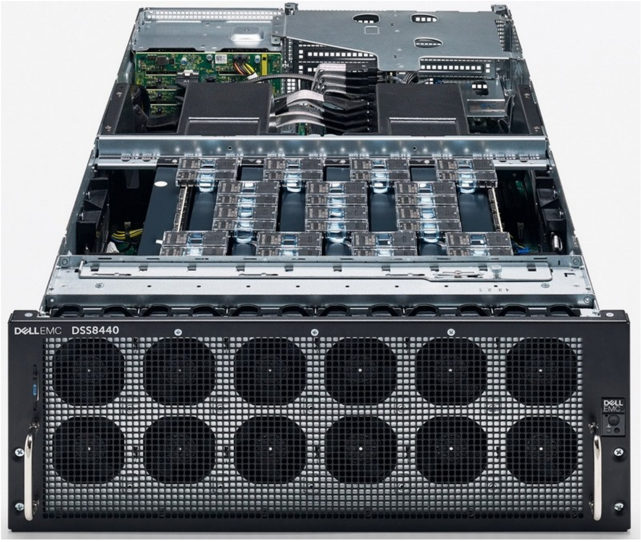


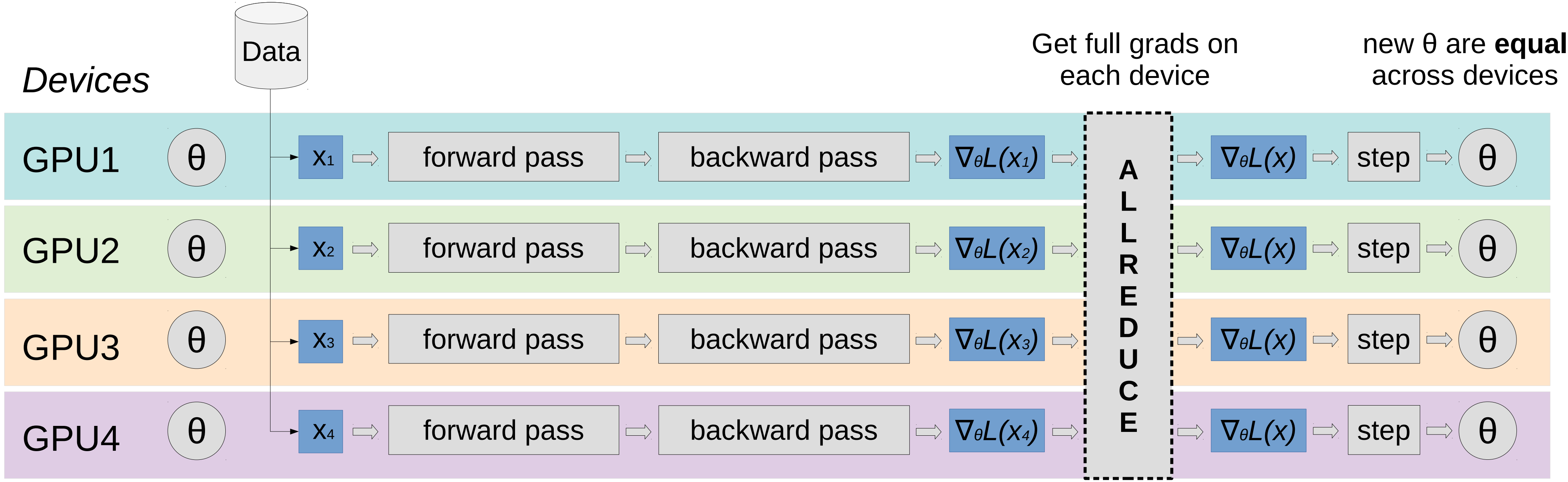
Image: graphcore ipu server

Answer & more: tinyurl.com/ring-allreduce-blog

Advanced data parallel

arxiv.org/abs/1706.02677

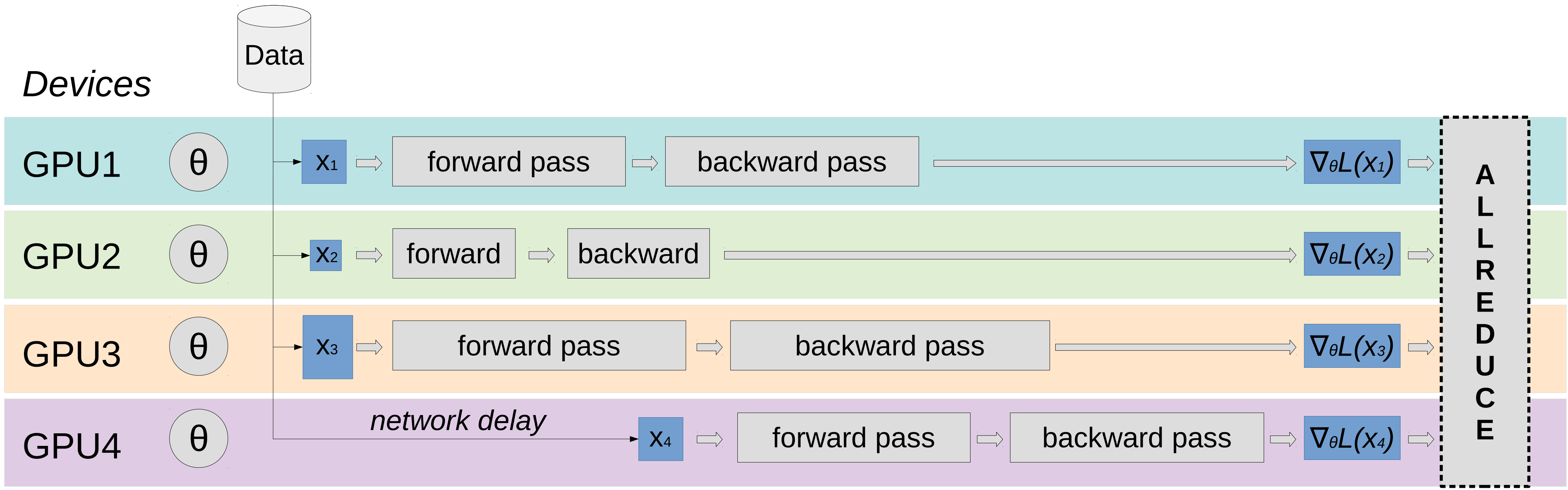
Idea: get rid of the host, each gpu runs its own computation
Q: why will weights be equal after such step?



Advanced data parallel vs reality

arxiv.org/abs/1706.02677

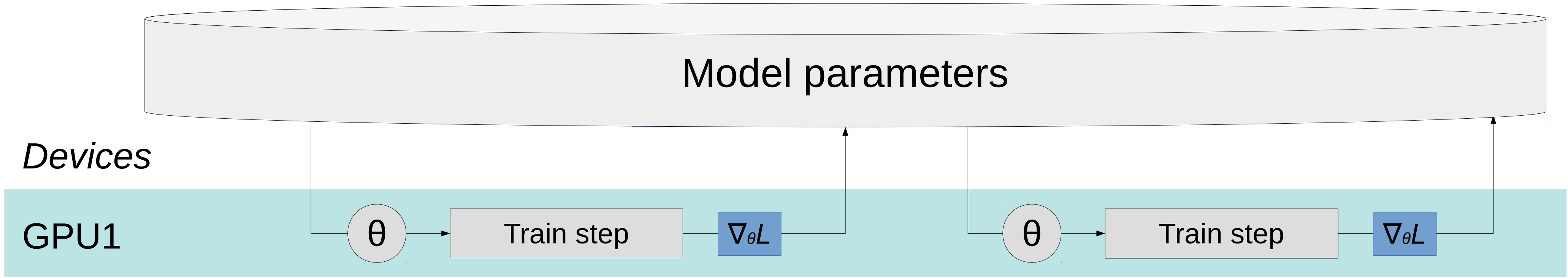
Each gpu has different processing time & delays
Q: can we improve device utilization?



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

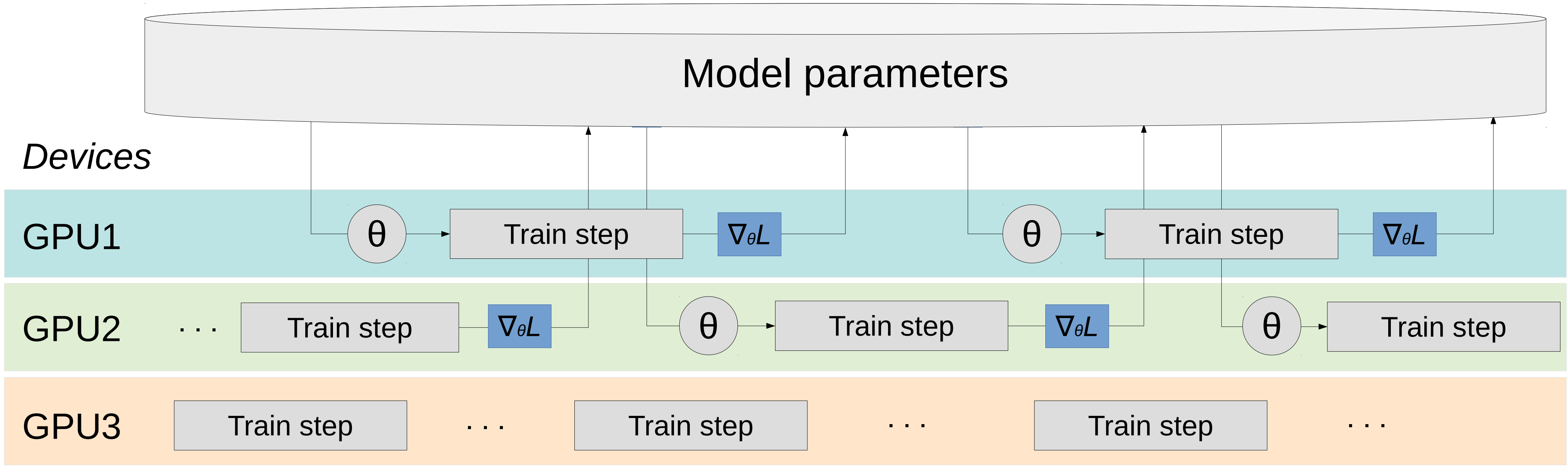
Idea: remove synchronization step alltogether, use parameter server



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

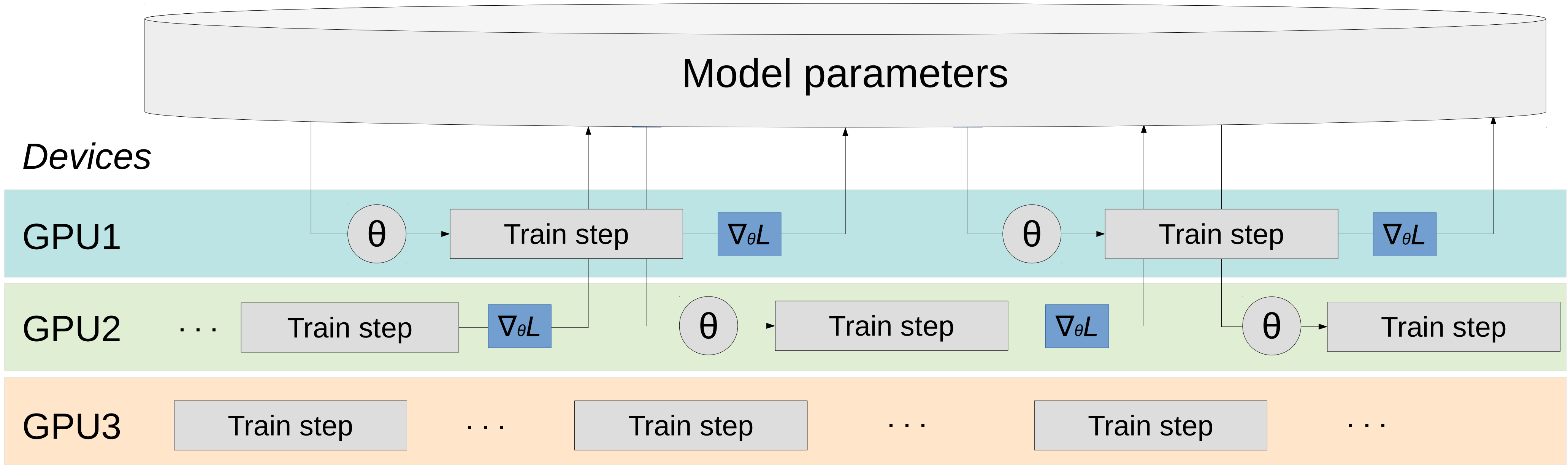
Idea: remove synchronization step alltogether, use parameter server



Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server

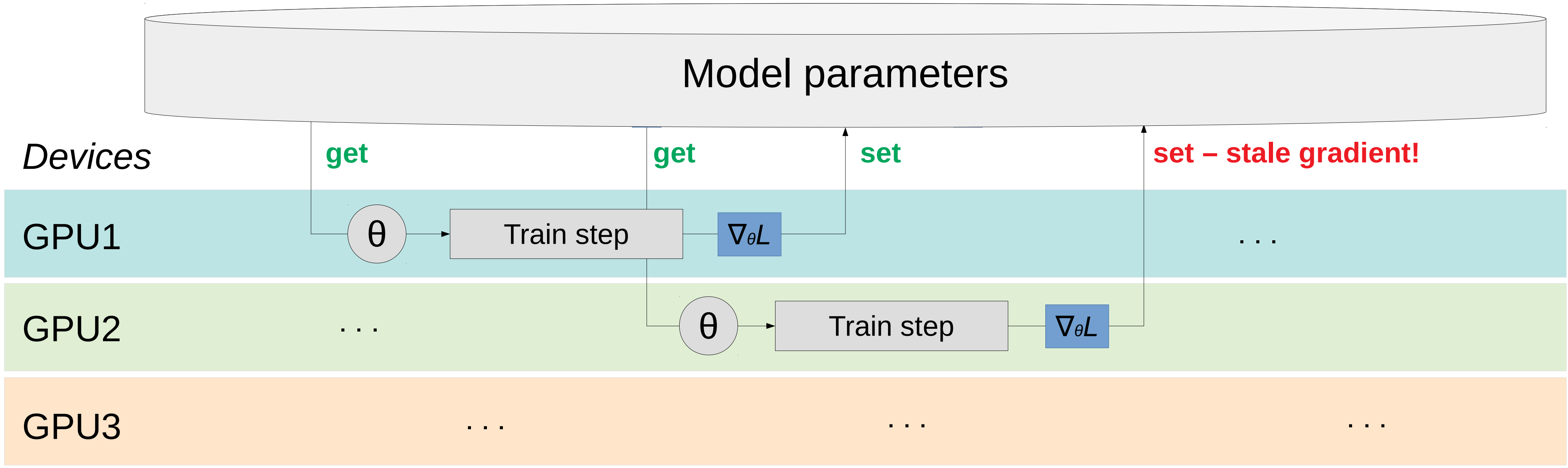


Q: have we lost anything by going asynchronous?

Asynchronous data-parallel training

HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server



Correction for staleness: arxiv.org/abs/1511.05950 & many others

Data-parallel Reinforcement Learning

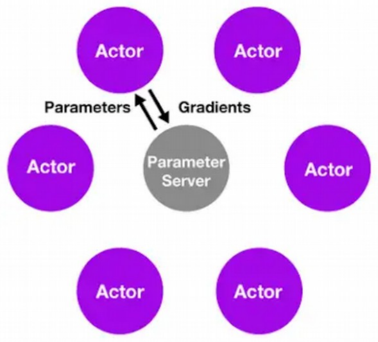
Synchronous data-parallel: A. Stooke & P. Abbeel, 2018 tinyurl.com/gtc-parallel-rl

Asynchronous data-parallel:

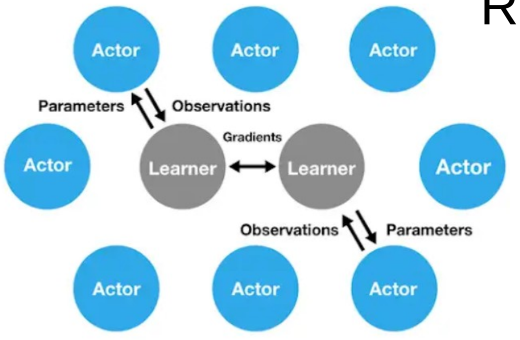
Asynchronous methods for deep RL: arxiv.org/abs/1602.01783

Distributed asynchronous data-parallel:

IMPALA: arxiv.org/abs/1802.01561
R2D2: openreview.net/forum?id=r1lyTjAqYX
SEED RL: arxiv.org/abs/1910.06591



A3C



IMPALA - Multiple Learners

More on this on the distributed RL lecture!

</Data-parallel>

+ easy to implement
+ can scale to 100s of gpus
+ can be fault-tolerant
- model must fit in 1 gpu
- large batches aren't always good for generalization

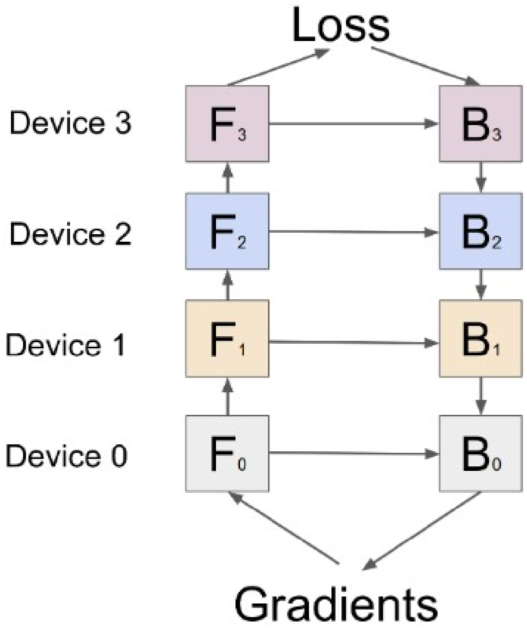
- 2-4 GPUs & no time – naive data parallel tinyurl.com/torch-data-parallel
- 4+ GPUs or multiple hosts – horovod (allreduce) github.com/horovod/horovod
  - High-level distributed pytorch (allreduce): tinyurl.com/distributed-dp
- Somewhat faulty GPU/network: synchronous data parallel + drop stragglers
- Very faulty or uneven resources: asynchronous data parallel (more later)
- Efficient training with large batches: LAMB https://arxiv.org/abs/1904.00962
- Dynamically adding or removing resources: https://tinyurl.com/torch-elastic

Chapter 2: Model-parallel training

Q: What if a model is larger than GPU?

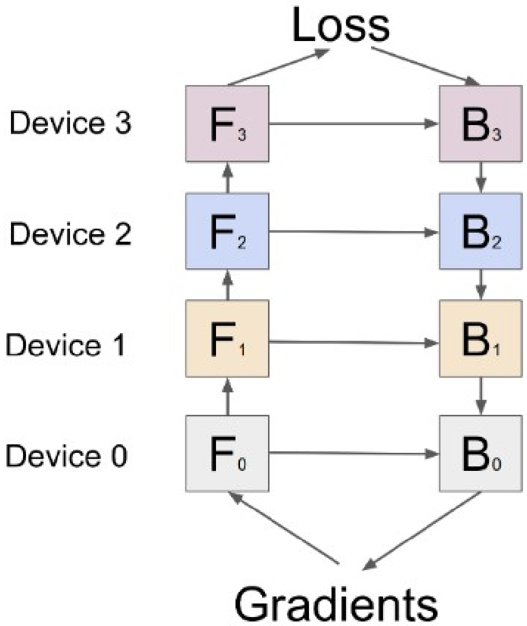
Model-parallel training

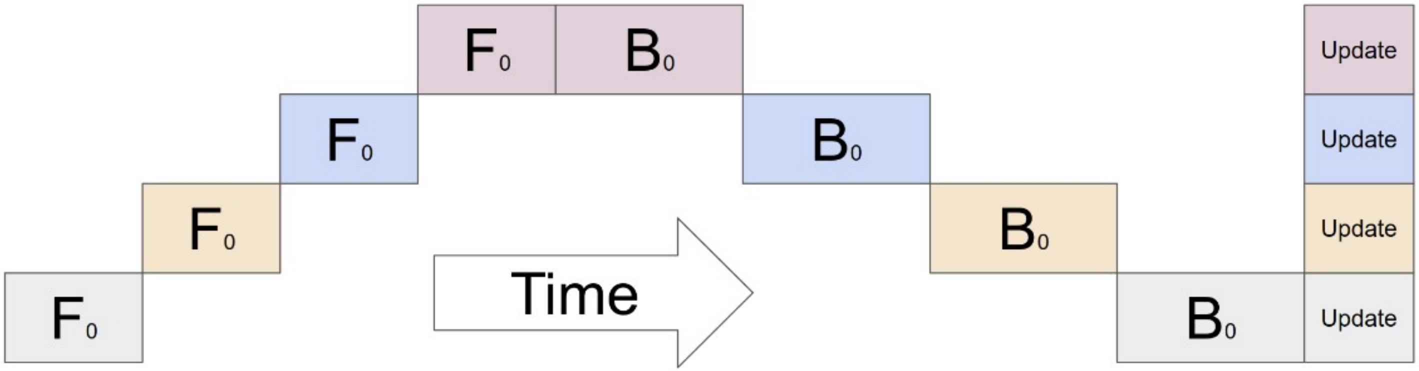
Q: What if a model is larger than GPU?



Model-parallel training

Q: What if a model is larger than GPU?





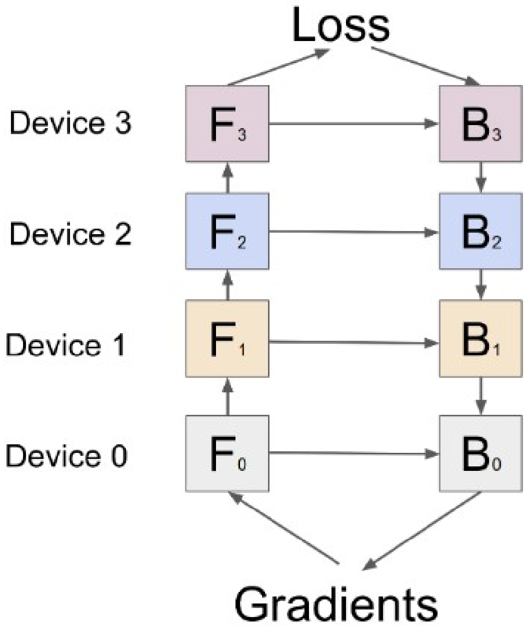
model size: O(N)
throughput: O(1)

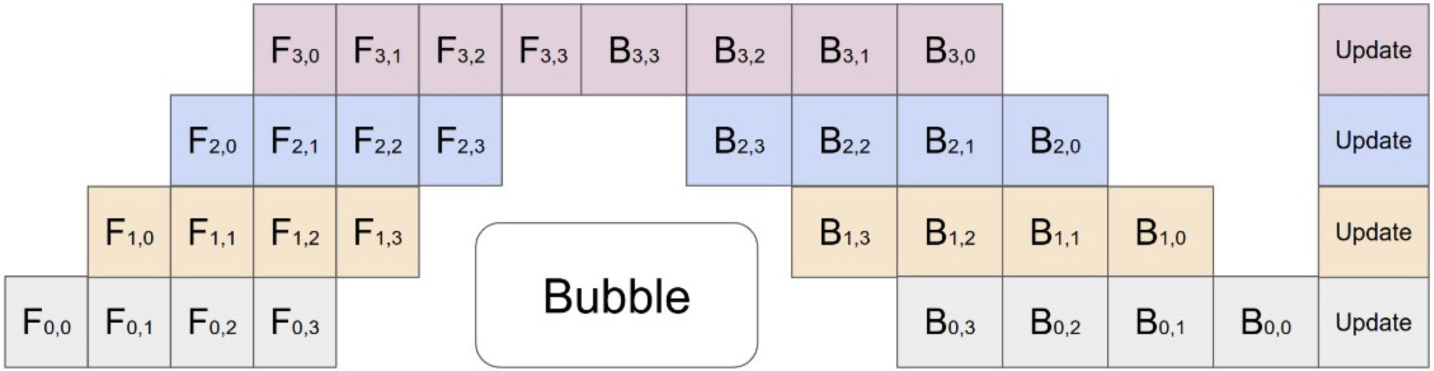
Q: Can we go faster?

Pipelining

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the 1st paper

Idea: split data into micro-batches and form a pipeline (right)



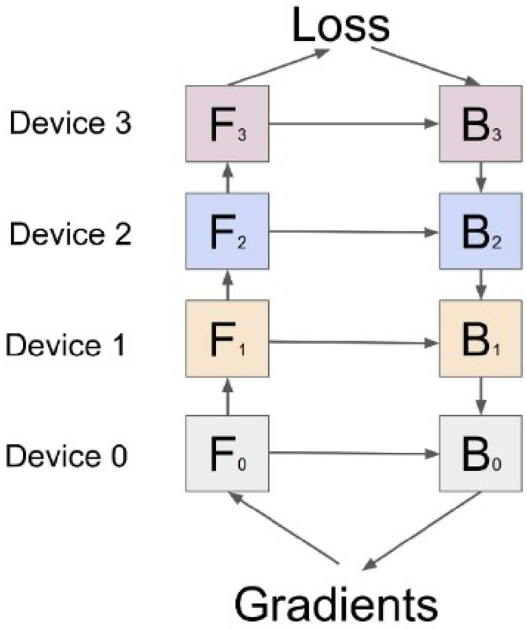


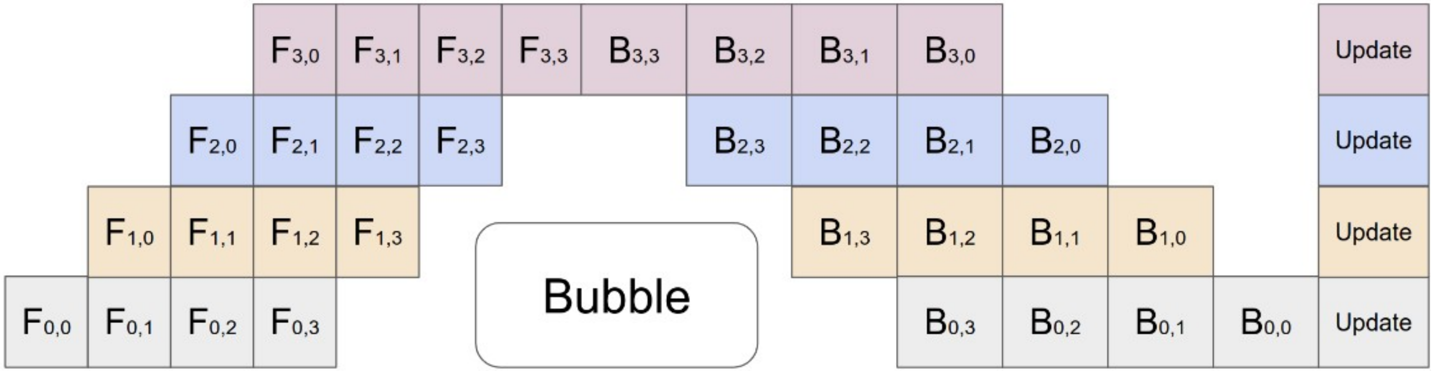
model size: O(n)
throughput: O(n) – with caveats

Pipelining

GPipe: arxiv.org/abs/1811.06965 – good starting point, *not* the 1st paper

Idea: split data into micro-batches and form a pipeline (right)





model size: O(n)
throughput: O(n) – with caveats

Q: Even faster?

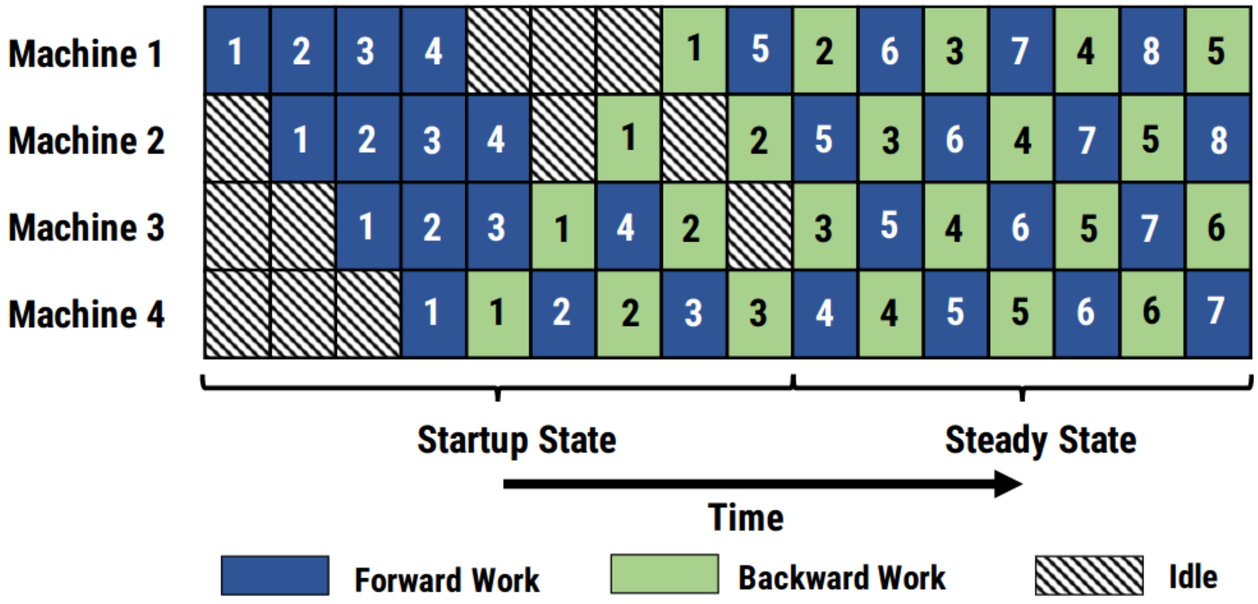
Pipeline-parallel training

PipeDream: arxiv.org/abs/1806.03377

Idea: apply gradients with every microbatch for maximum throughput

Also neat:

- Automatically partition layers to GPUs via dynamic programming
- *Store k* past weight versions to reduce gradient staleness
- Aims at high latency



</Model-parallel>

+ model larger than GPU
+ faster for small
***** typical size: 2-8 gpus
- model partitioning is tricky
- latency is critical, go buy nvlink
except for PipeDream

Tutorials:

- Simple pipelining in PyTorch – tinyurl.com/pytorch-pipelining
- Distributed model-parallel with torch RPC - https://tinyurl.com/torch-rpc
- Advanced but still in active development - github.com/microsoft/DeepSpeed

Virtual batch / virtual pipeline

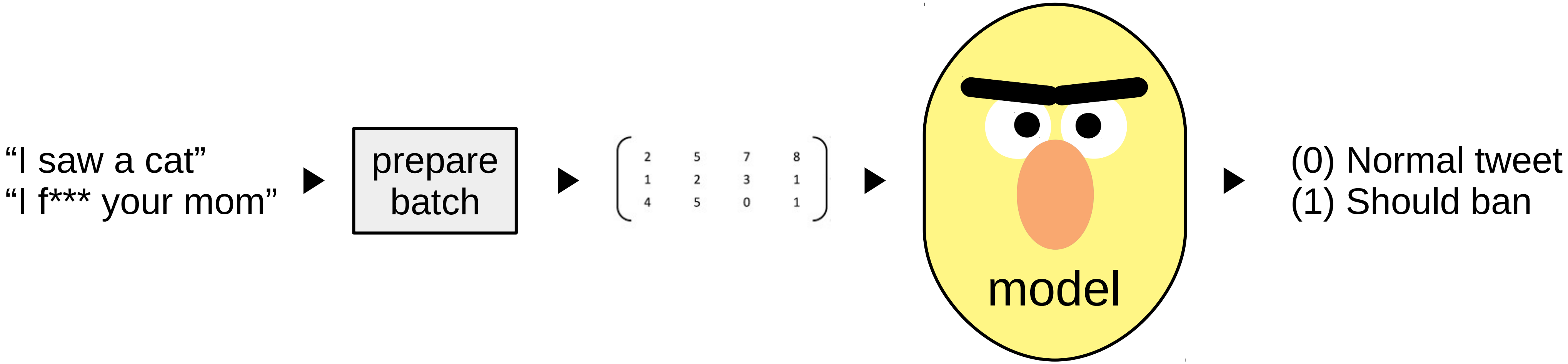
ёж, открой доску

Case study: DeepSpeed

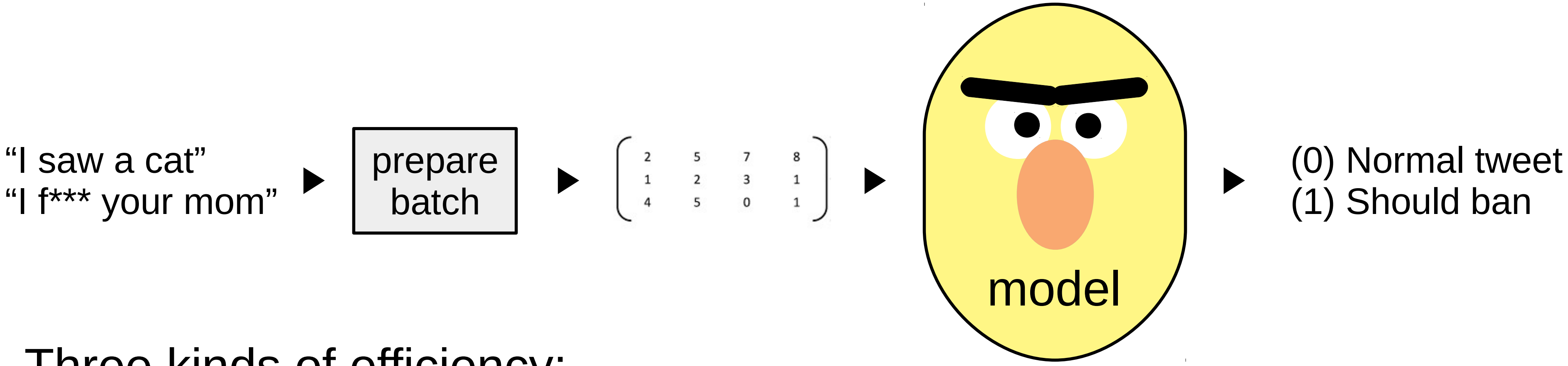
Source: microsoft

Chapter 3: What about inference?

Case study: text classification



Case study: text classification



Three kinds of efficiency:

Model Size
(mega)bytes

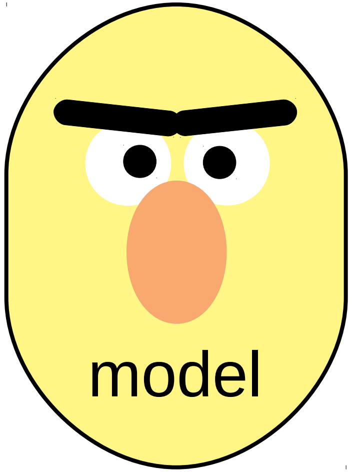
Case study: text classification

"I saw a cat"
"I f*** your mom"

▶ prepare batch ▶

$$\begin{bmatrix} 2 & 5 & 7 & 8 \\ 1 & 2 & 3 & 1 \\ 4 & 5 & 0 & 1 \end{bmatrix}$$

▶



▶

(0) Normal tweet
(1) Should ban

Three kinds of efficiency:

Model Size
(mega)bytes

Throughput
samples/second

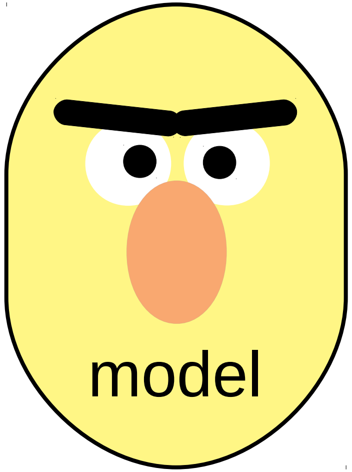
Case study: text classification

"I saw a cat"
"I f*** your mom"

▶ prepare batch ▶

$$\begin{pmatrix} 2 & 5 & 7 & 8 \\ 1 & 2 & 3 & 1 \\ 4 & 5 & 0 & 1 \end{pmatrix}$$

▶



▶

(0) Normal tweet
(1) Should ban

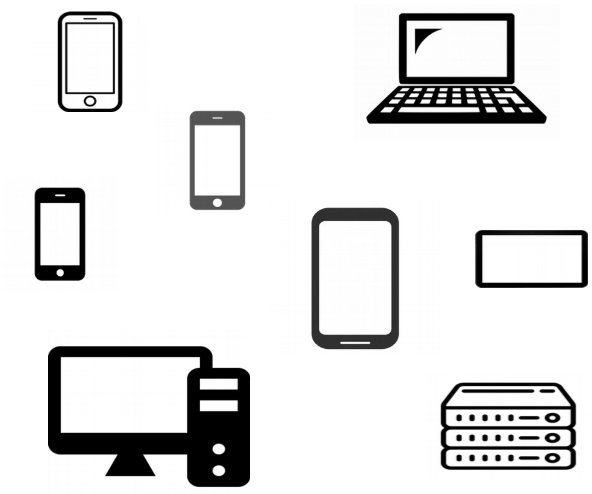
Three kinds of efficiency:

Model Size
(mega)bytes

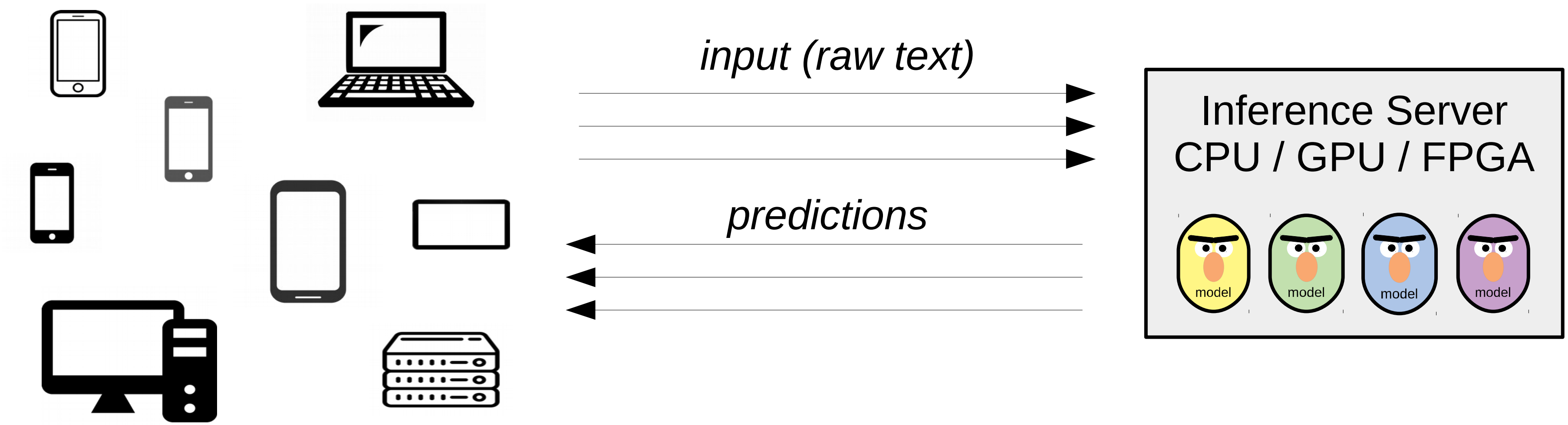
Throughput
samples/second

Latency
ms@percentile

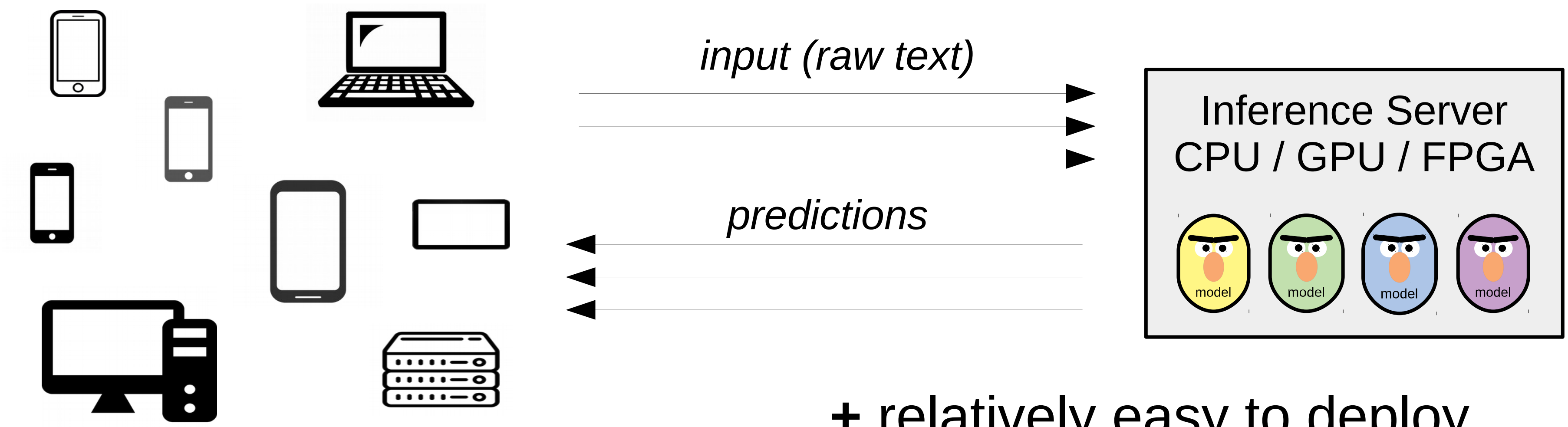
Scenario 1: inference server



Scenario 1: inference server

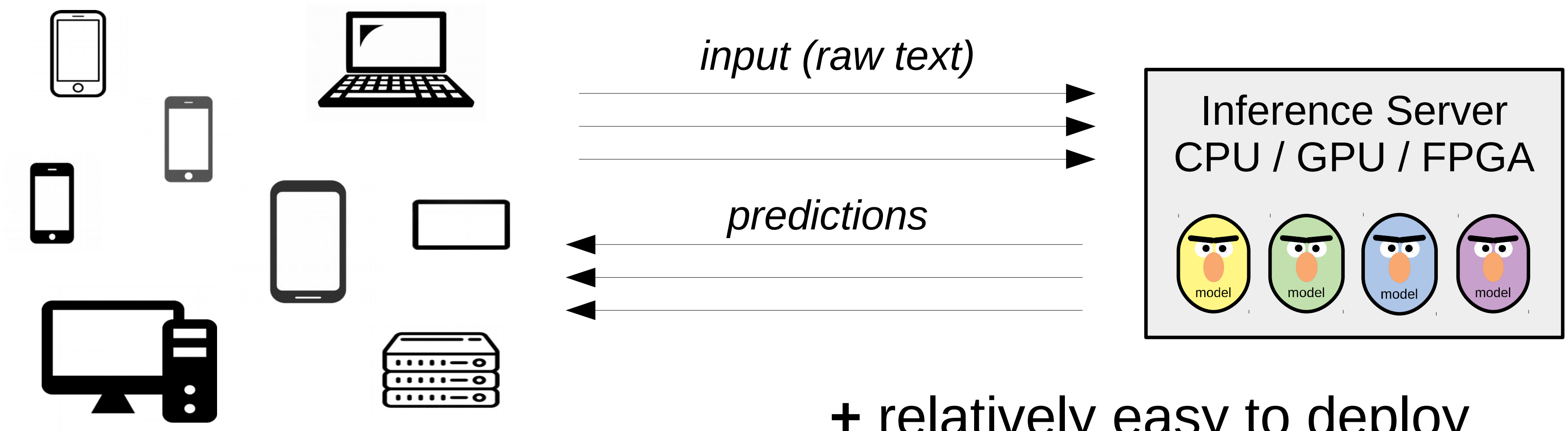


Scenario 1: inference server



+ relatively easy to deploy
+ you control model & inference
+ clients don’t run compute

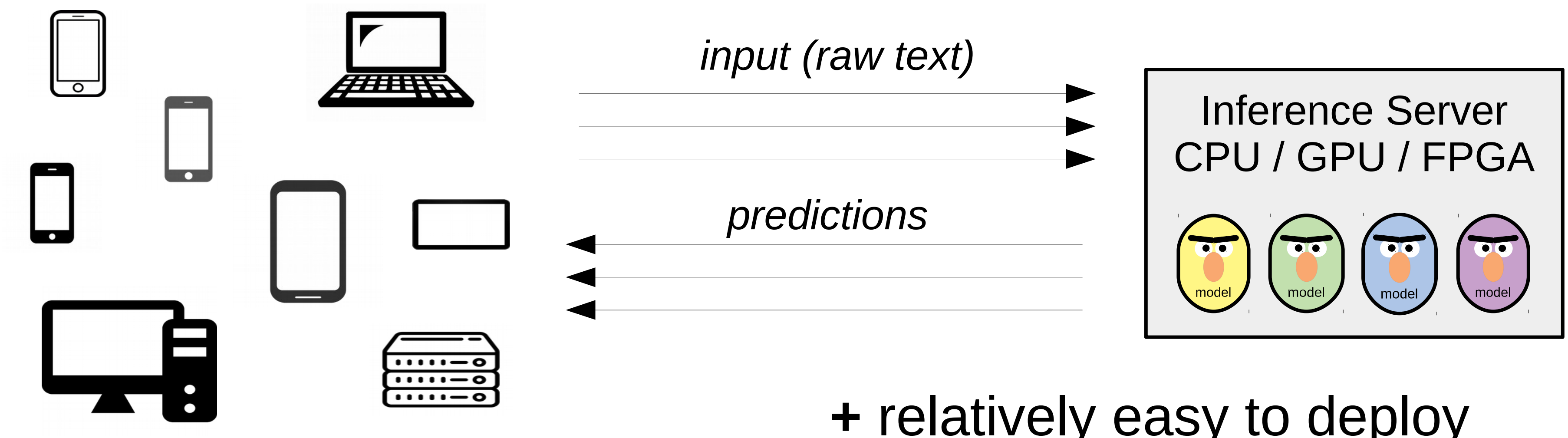
Scenario 1: inference server



+ relatively easy to deploy
+ you control model & inference
+ clients don't run compute

– you pay for each inference
– clients can't work offline
– network latency

Scenario 1: inference server

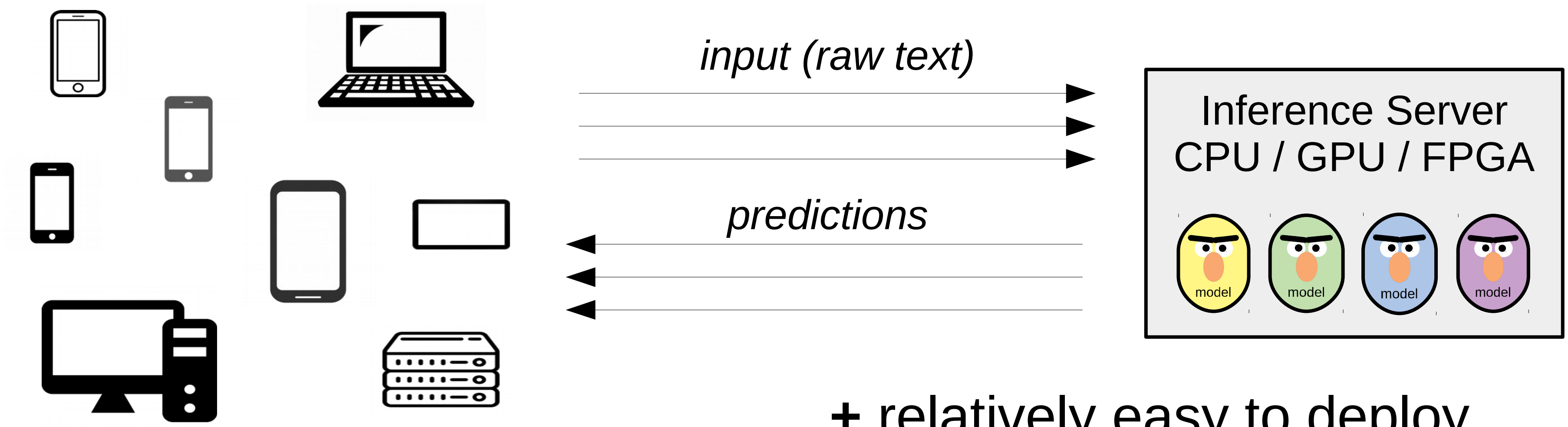


Which is the most important?

+ relatively easy to deploy
+ you control model & inference
+ clients don’t run compute

– you pay for each inference
– clients can’t work offline
– network latency

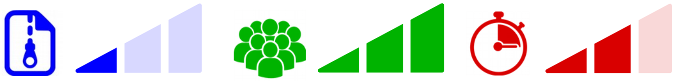
Scenario 1: inference server



+ relatively easy to deploy
+ you control model & inference
+ clients don’t run compute

– you pay for each inference
– clients can’t work offline
– network latency

Priorities:



Note: smaller model = you can fit more models in the same memory

Scenario 1: inference server

- Group inputs into batches (e.g. by length)
  improves throughput at the cost of latency
- Multiple servers with load balancing
  improves throughput at the cost of your budget :)

Scenario 1: inference server

- Group inputs into batches (e.g. by length)
  improves throughput at the cost of latency
- Multiple servers with load balancing
  improves throughput at the cost of your budget :)

Popular frameworks:	priorities
TensorFlow Serving	efficiency $\ll$ developer time
TensorRT Inference Server (Triton)	efficiency $\approx$ developer time
 Custom model-dependent code	efficiency $\gg$ developer time

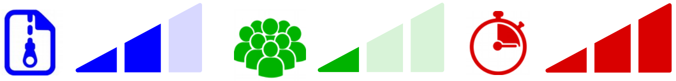
Scenario 2: local inference

Preload model onto a dedicated device, infer locally using that device

Typical use cases:

- Parallel speech recognition
- “Smart” cameras
- Autonomous drones
- Self-driving cars

Priorities:



Scenario 3: web/smartphone app

- Load model weights on the fly and infer locally
  - Model size is critical for both you and the user

Scenario 3: web/smartphone app

- Load model weights on the fly and infer locally
  Model size is critical for both you and the user

- Autonomous machine translation (tinyurl.com/yandex-translate-app)
- Pix2pix demo in a browser (https://affinelayer.com/pixsrv)

- Priorities: 

Scenario 3: web/smartphone app

- Load model weights on the fly and infer locally
  Model size is critical for both you and the user

- Autonomous machine translation (tinyurl.com/yandex-translate-app)
- Pix2pix demo in a browser (https://affinelayer.com/pixsrv)

- Priorities:

- Popular frameworks:

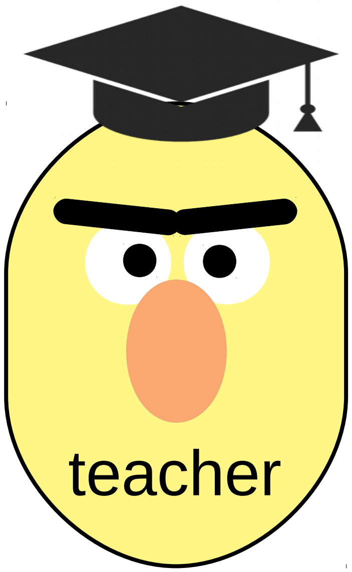
	Platform
TensorFlow.js	All modern browsers
CoreML	iOS devices
NNAPI	Android devices

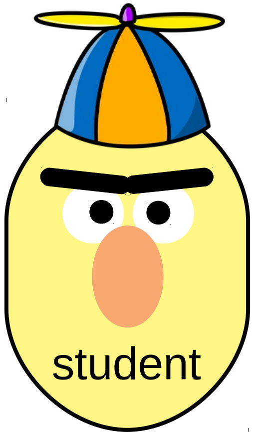
Chapter 4: how do I compress my model?

Compression by Distillation

Distillation…
Heard that word before?

Compression by Distillation

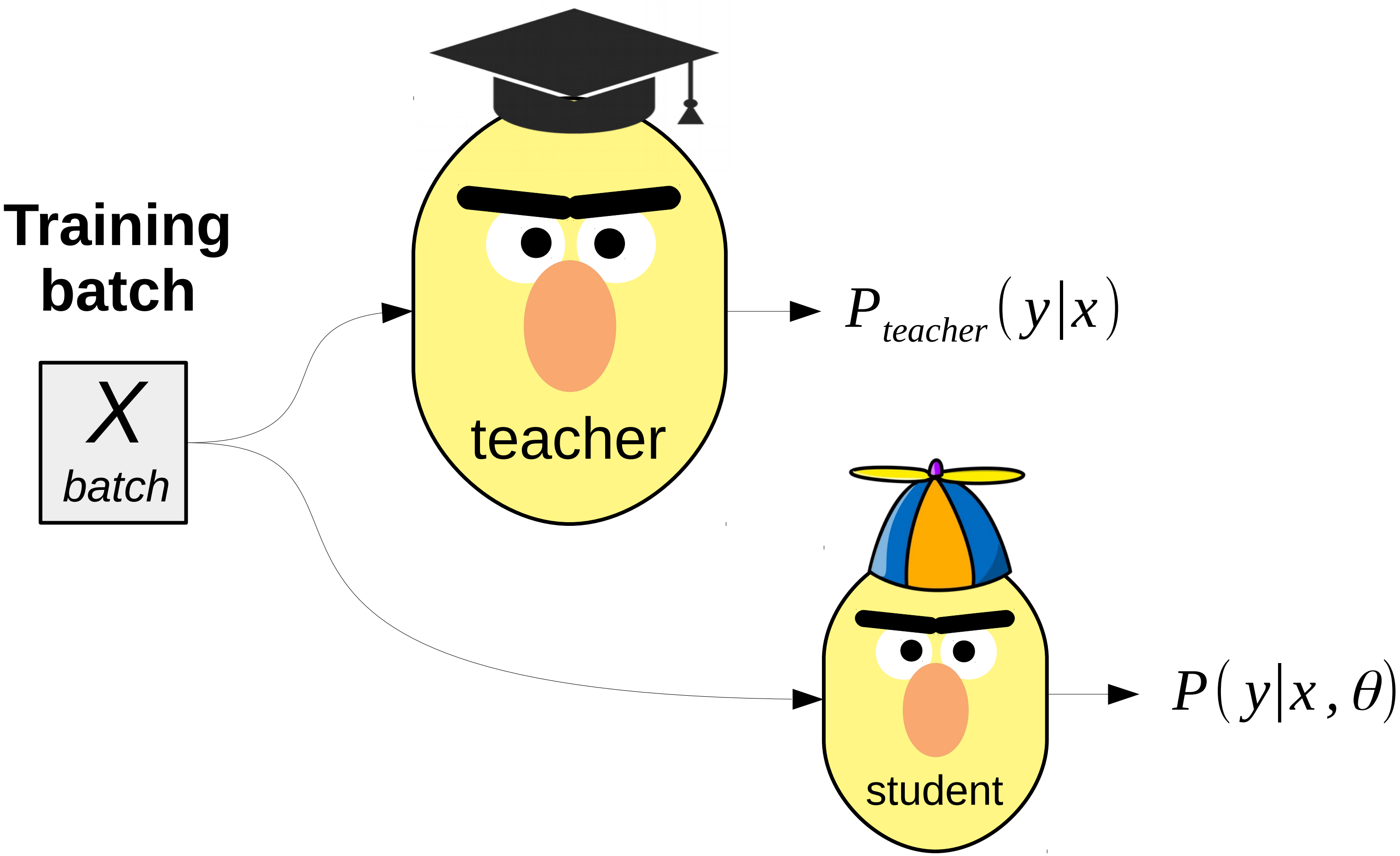




First, get the best performing model regardless of size

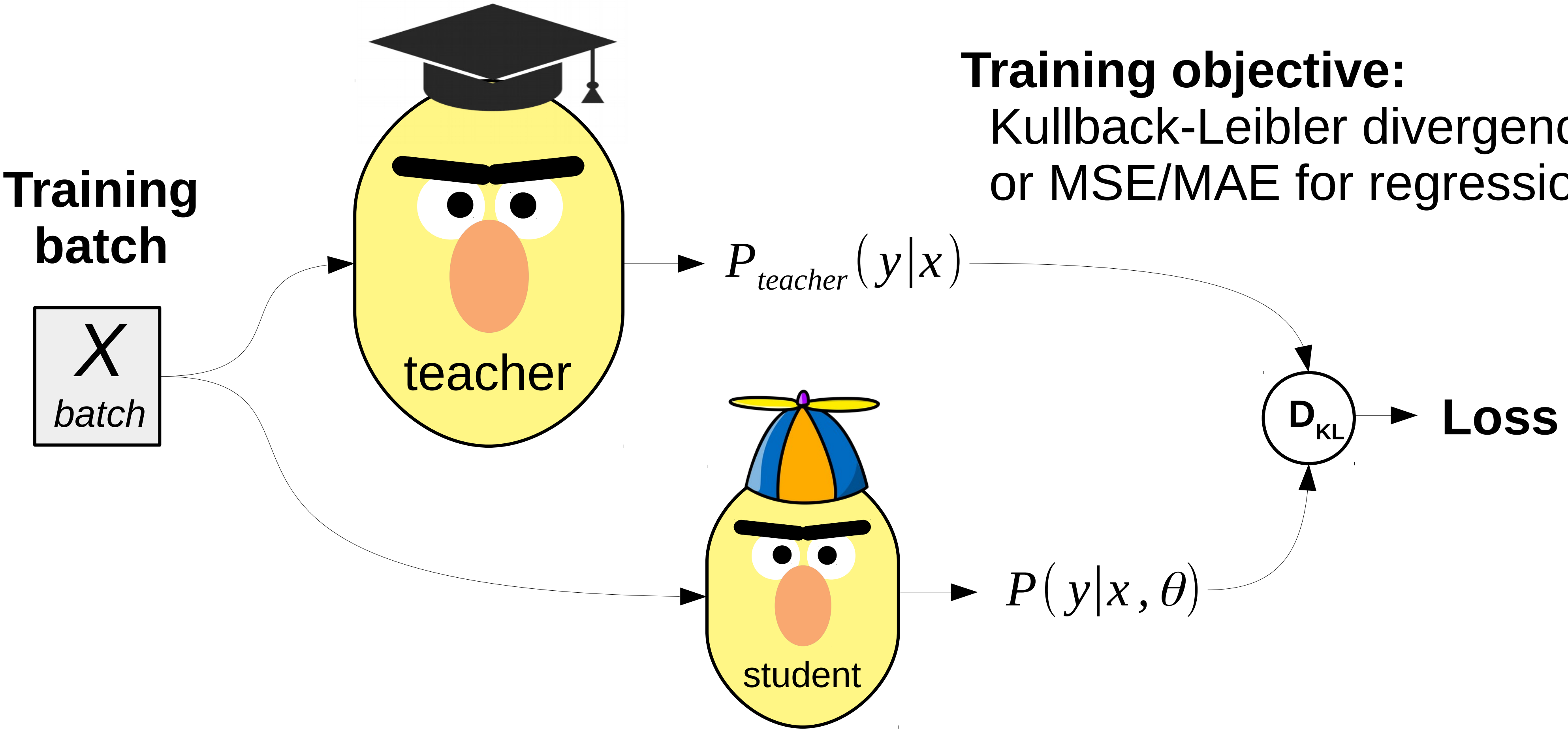
Then, train a more compact model to approximate it!

Compression by Distillation



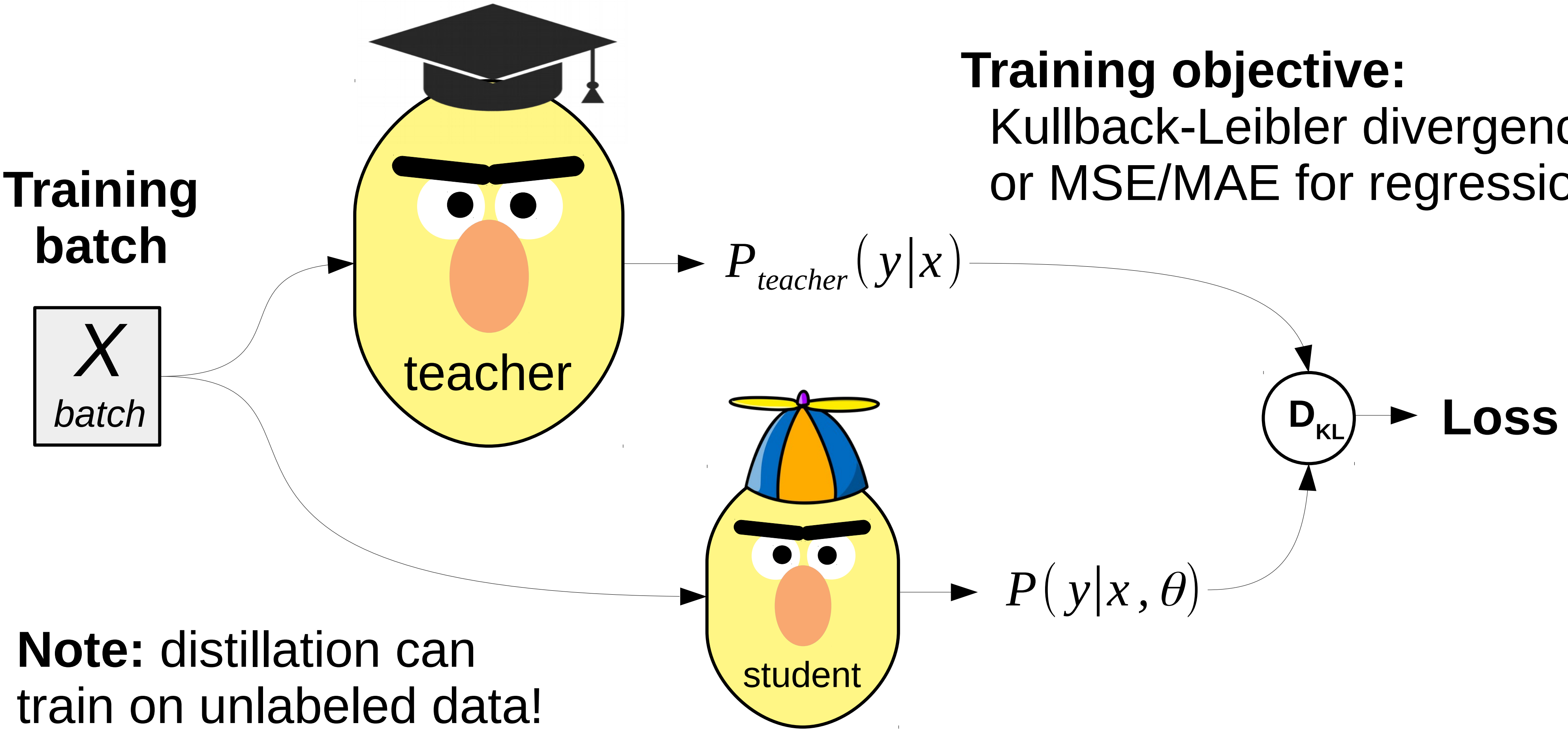
Compression by Distillation

Training objective:
Kullback-Leibler divergence
or MSE/MAE for regression



Compression by Distillation

Training objective:
Kullback-Leibler divergence
or MSE/MAE for regression



Note: distillation can
train on unlabeled data!

Compression by Distillation

- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  e.g. DistillBERT: https://arxiv.org/pdf/1910.01108.pdf

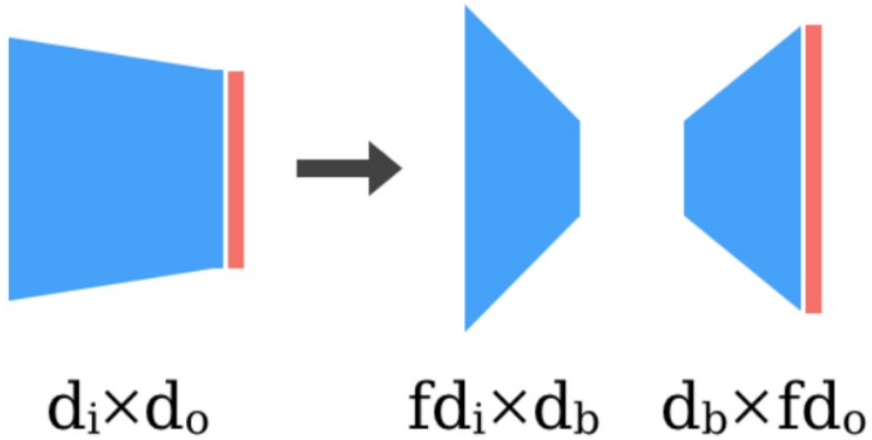
Same as BERT-base, but with *half as many layers* (and ≈1.5 times faster)

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Compression by Distillation

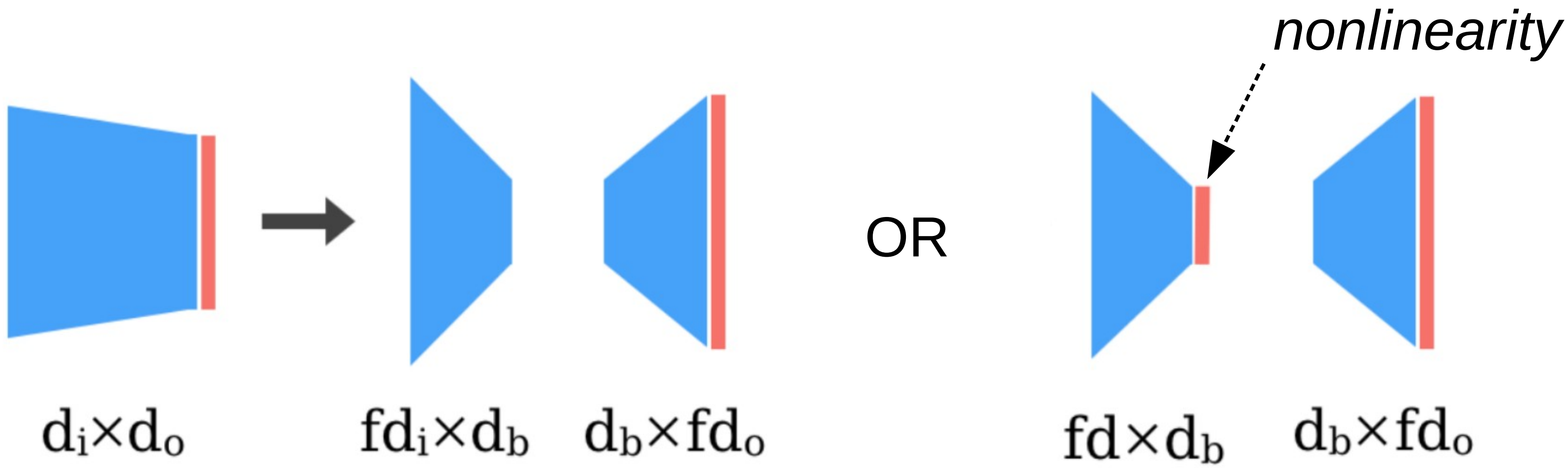
- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors



Images: https://openreview.net/pdf?id=_zx8Oka09eF

Compression by Distillation

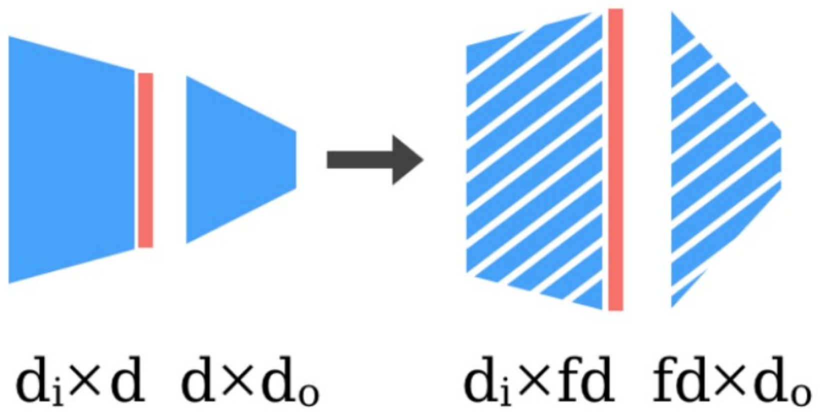
- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors



Images: https://openreview.net/pdf?id=_zx8Oka09eF

Compression by Distillation

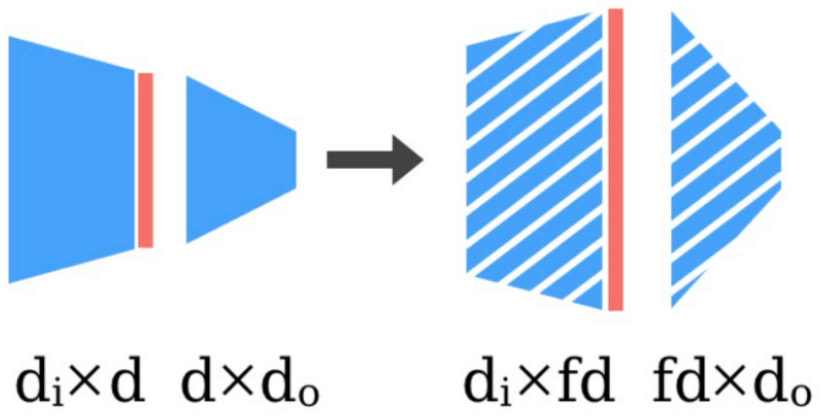
- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors
  Sparse: only a small (random) subset of weights are nonzero



Images: https://openreview.net/pdf?id=_zx8Oka09eF

Compression by Distillation

- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors
  Sparse: only a small (random) subset of weights are nonzero

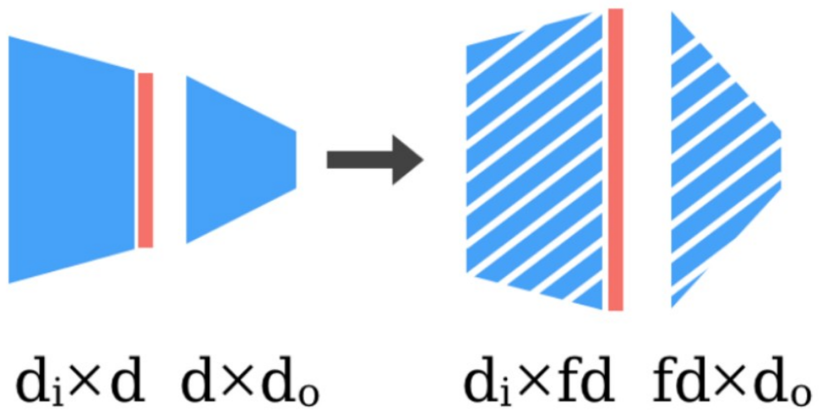


Q: how to store sparse weights?

Images: https://openreview.net/pdf?id=_zx8Oka09eF

Compression by Distillation

- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors
  Sparse: only a small (random) subset of weights are nonzero



Storage: only store random seed and nonzero weights.

Compute: sparse matrix multiply

Images: https://openreview.net/pdf?id=_zx8Oka09eF

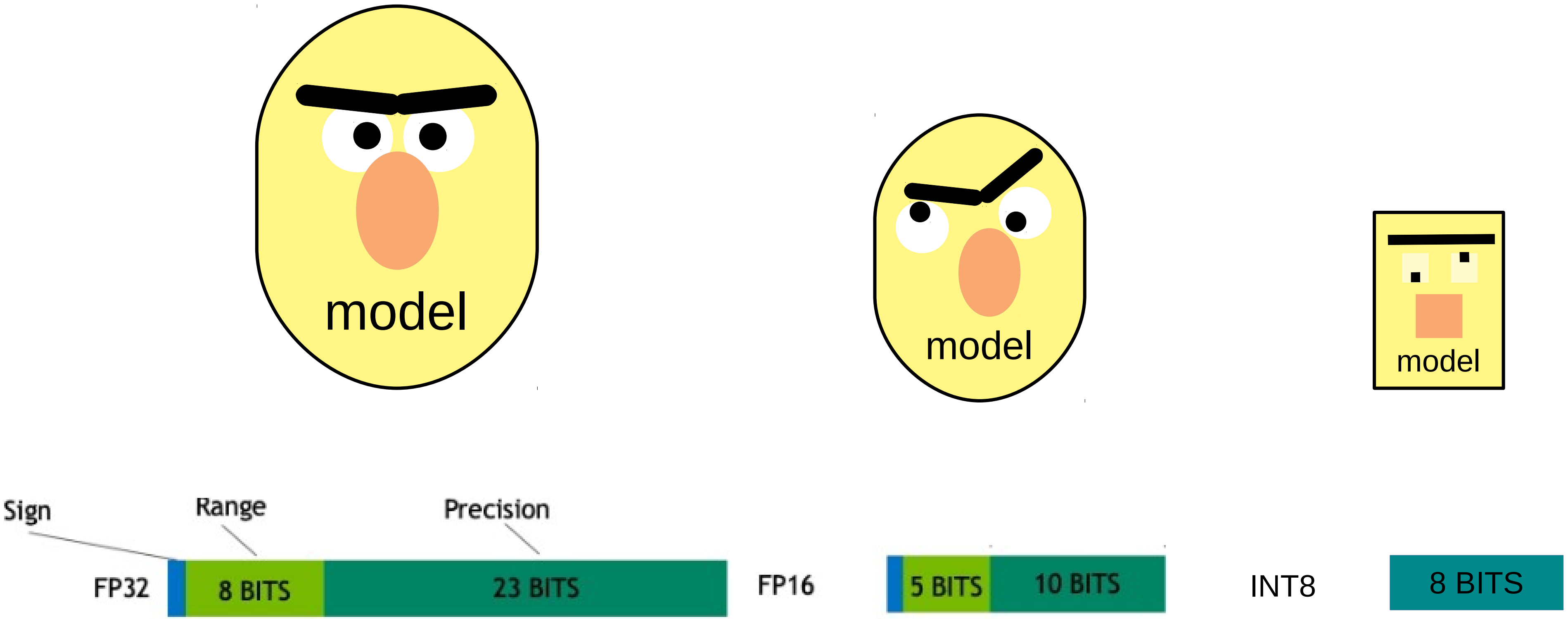
Compression by Distillation

- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors
  Sparse: only a small fraction of weights are nonzero
  Read more: https://openreview.net/pdf?id=_zx8Oka09eF
  Also: factorized embeddings https://arxiv.org/abs/1901.10787
  Also also: small-world sparse weights graphs for RNNs
  https://tinyurl.com/openai-blocksparse

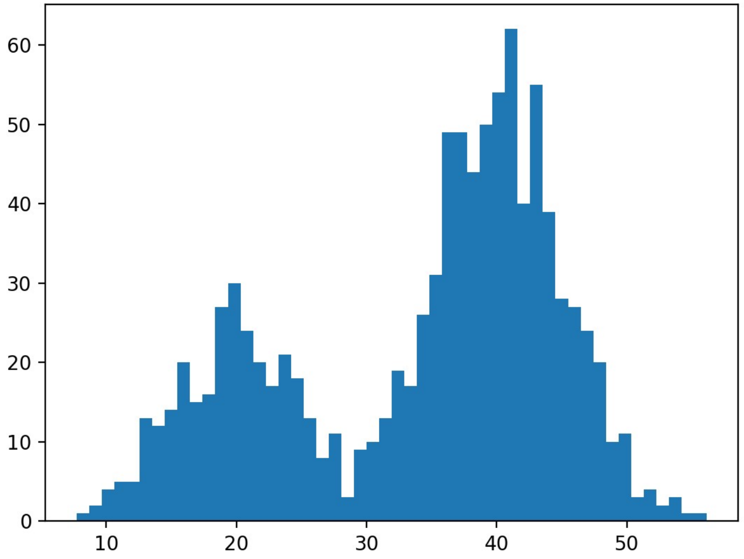
Compression by Distillation

- Student architecture choices:
  Naïve: same but smaller, less layers / hidden units
  Factorized: product of smaller matrices or tensors
  Sparse: only a small fraction of weights are nonzero
  Read more: https://openreview.net/pdf?id=_zx8Oka09eF
  Also: factorized embeddings https://arxiv.org/abs/1901.10787
  Also also: https://tinyurl.com/openai-blocksparse
- More distillation tricks:
  - Ensemble distillation https://arxiv.org/abs/1702.01802
  - Dropout distillation http://proceedings.mlr.press/v48/bulo16.pdf
  - Co-distillation https://arxiv.org/abs/1804.03235

Compression by quantization

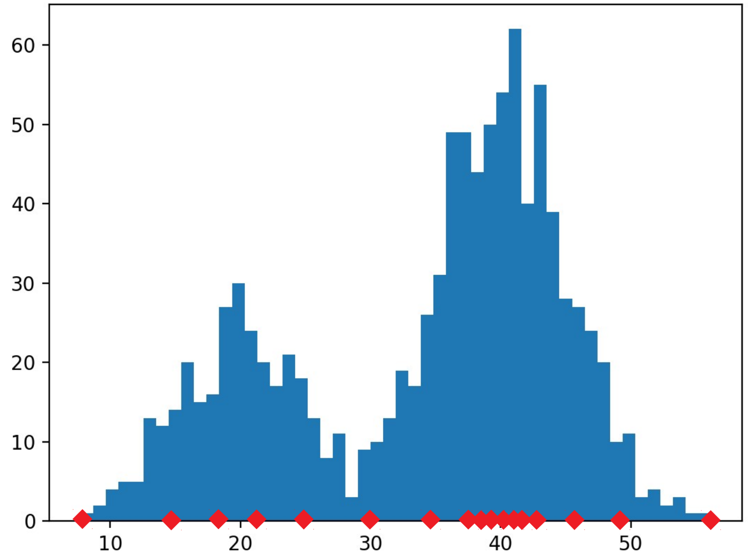


Basic quantization



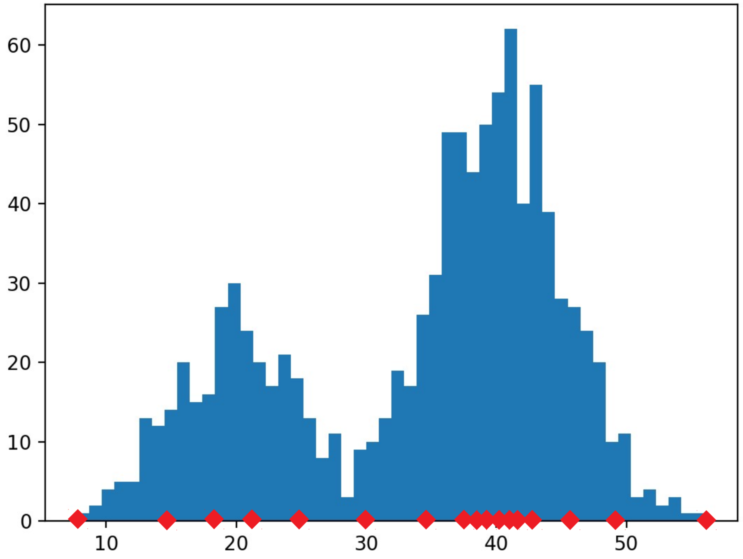
Consider weights as a distribution

Basic quantization



Compute a grid of percentiles

Basic quantization



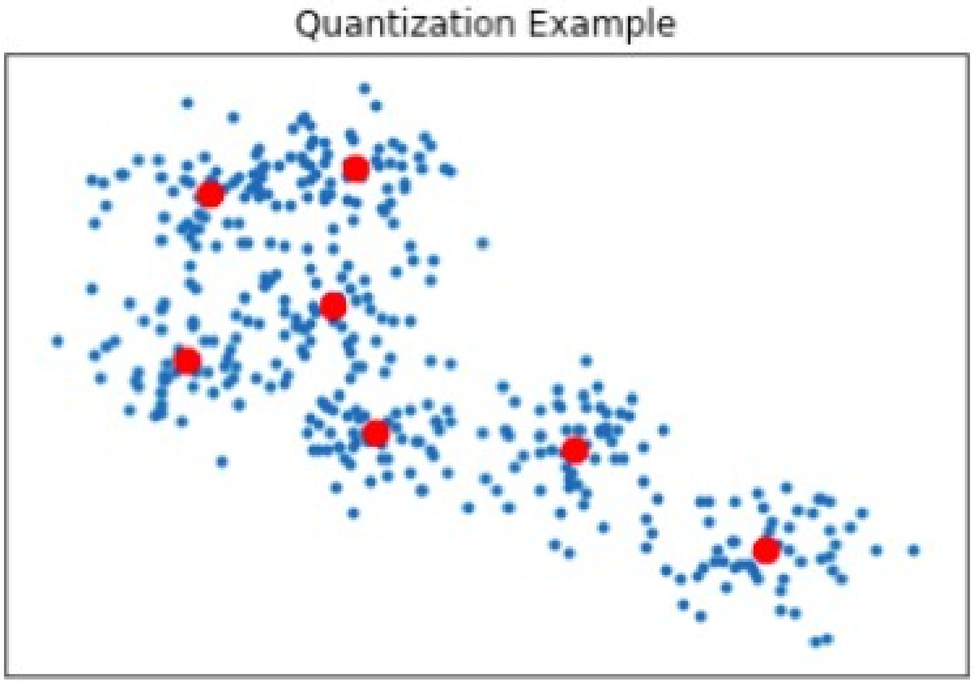
percentiles (32-bit)

Index (4- or 8-bit) of nearest percentile for each weight

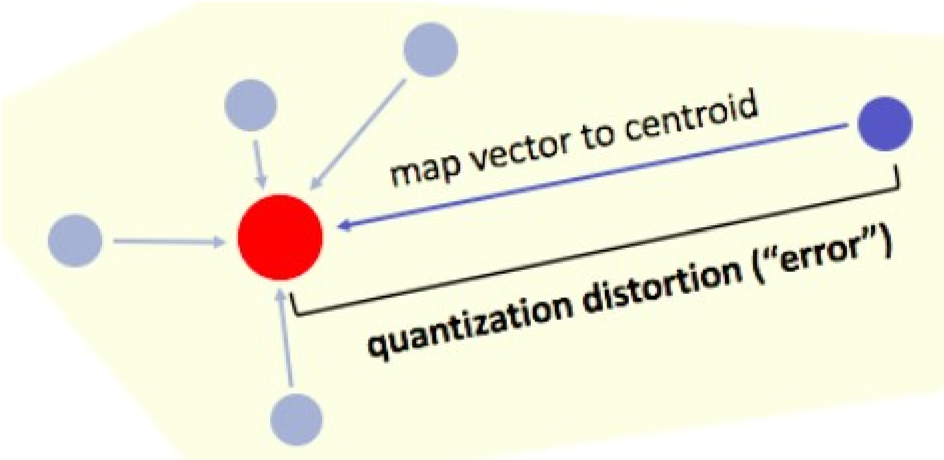
Store each weight as its nearest percentile

High-dimensional case

Quantize entire vectors as K-means



quantizer = KMeans(n_clusters=7).fit(X)



Images: Jeremy Jordan

OPQ, AQ, LSQ

Product **Q**uantization

Split vectors into chunks, quantize each chunk separately

Orthogonal **P**roduct **Q**uantization

First run orthogonal transform, then product quantization

http://kaiminghe.com/publications/cvpr13opq.pdf

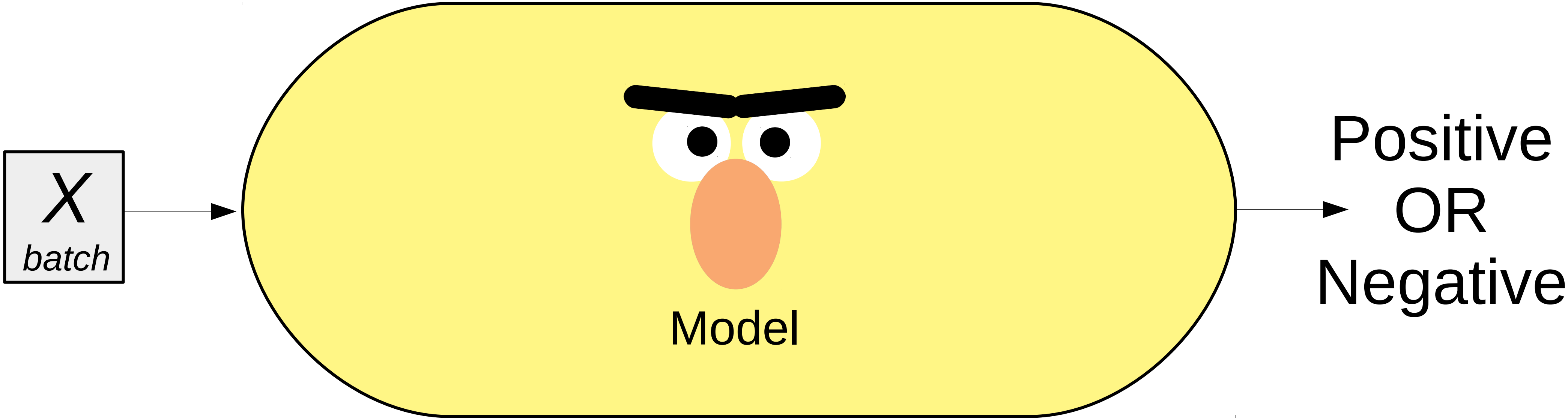
More:

Additive **Q**uantization https://tinyurl.com/babenko-aq-pdf

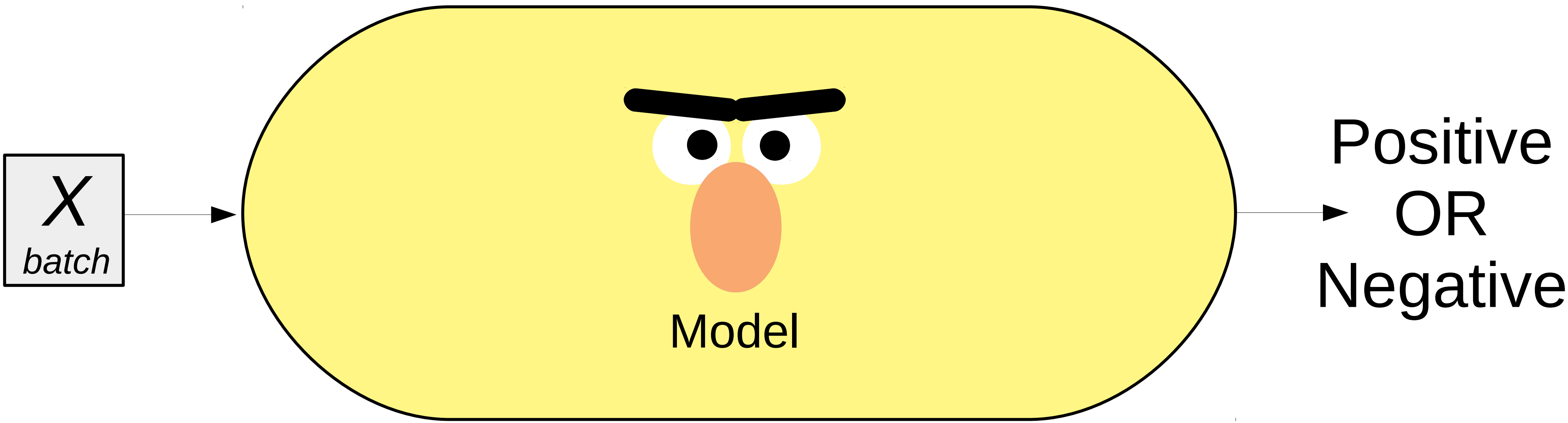
Local **S**earch **Q**uantization https://tinyurl.com/martinez-lsq-pdf

Images: Jeremy Jordan

Acceleration by cascading

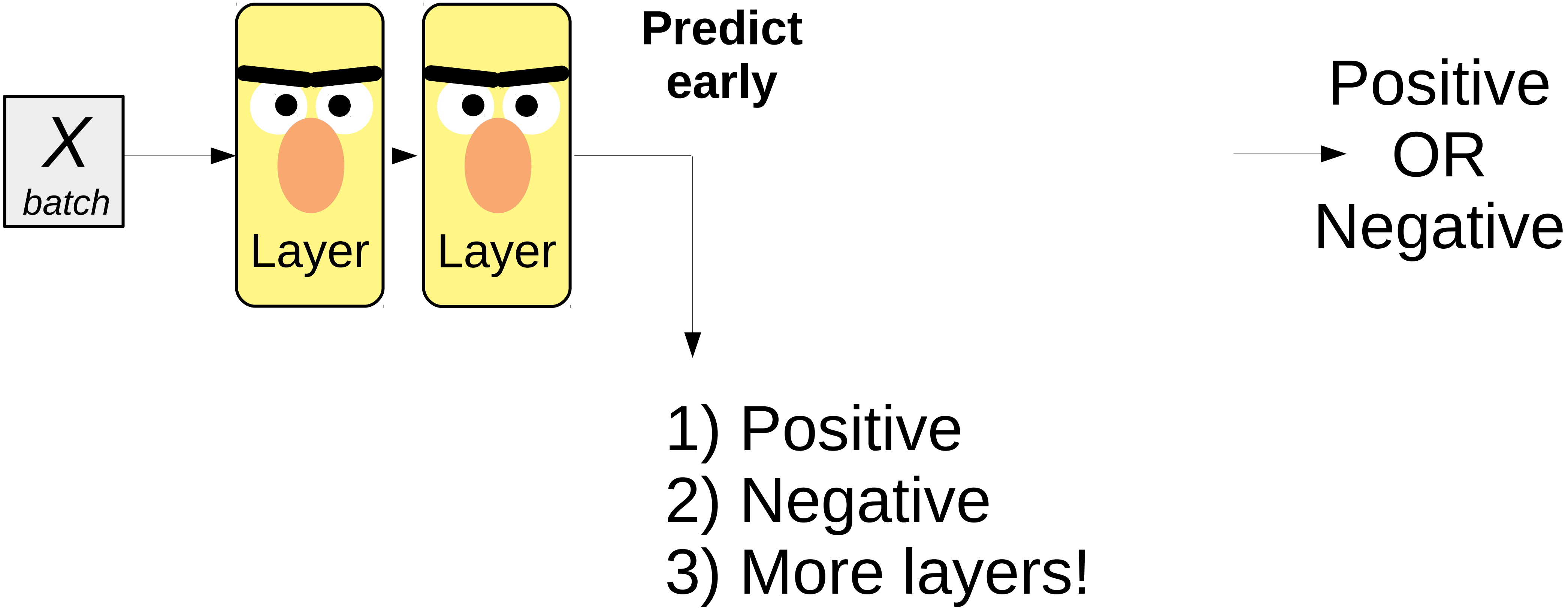


Acceleration by cascading

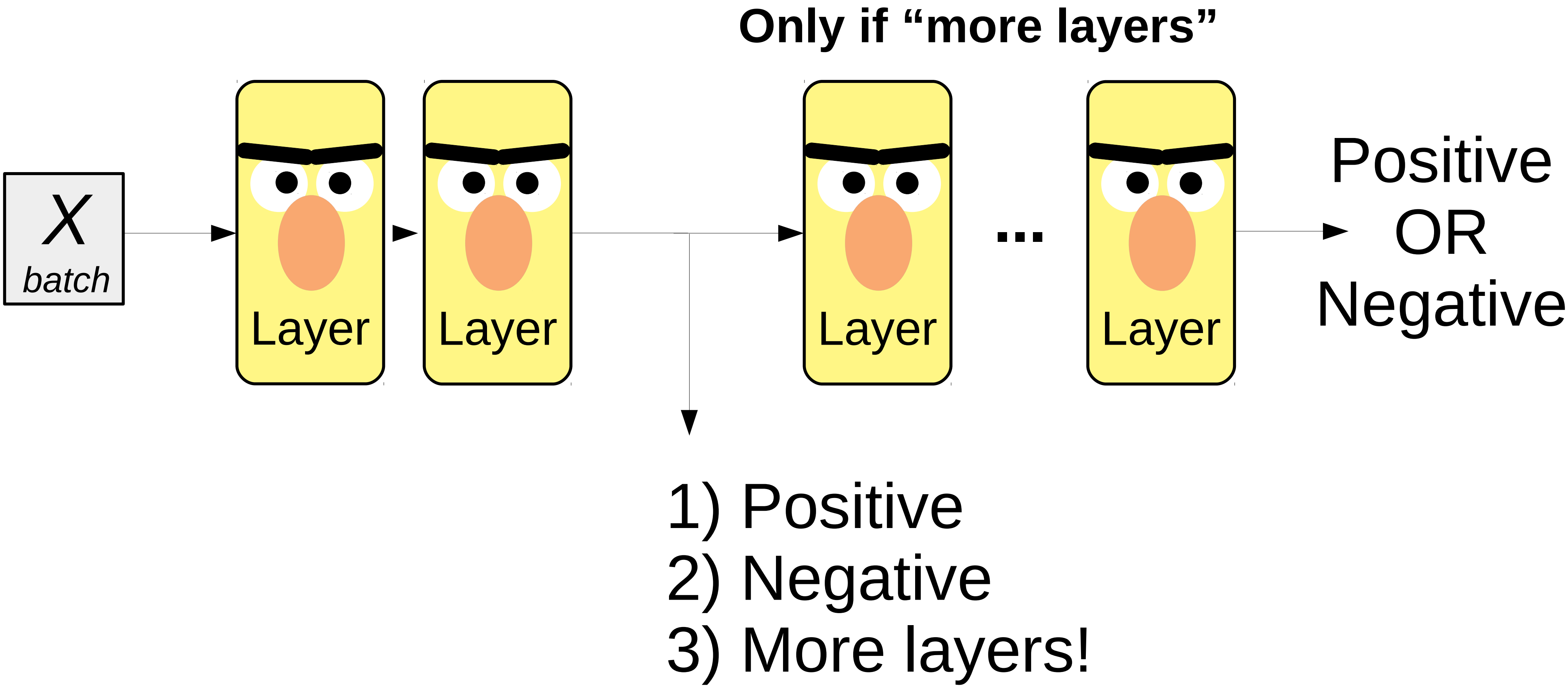


Do we really need every layer
all the time?

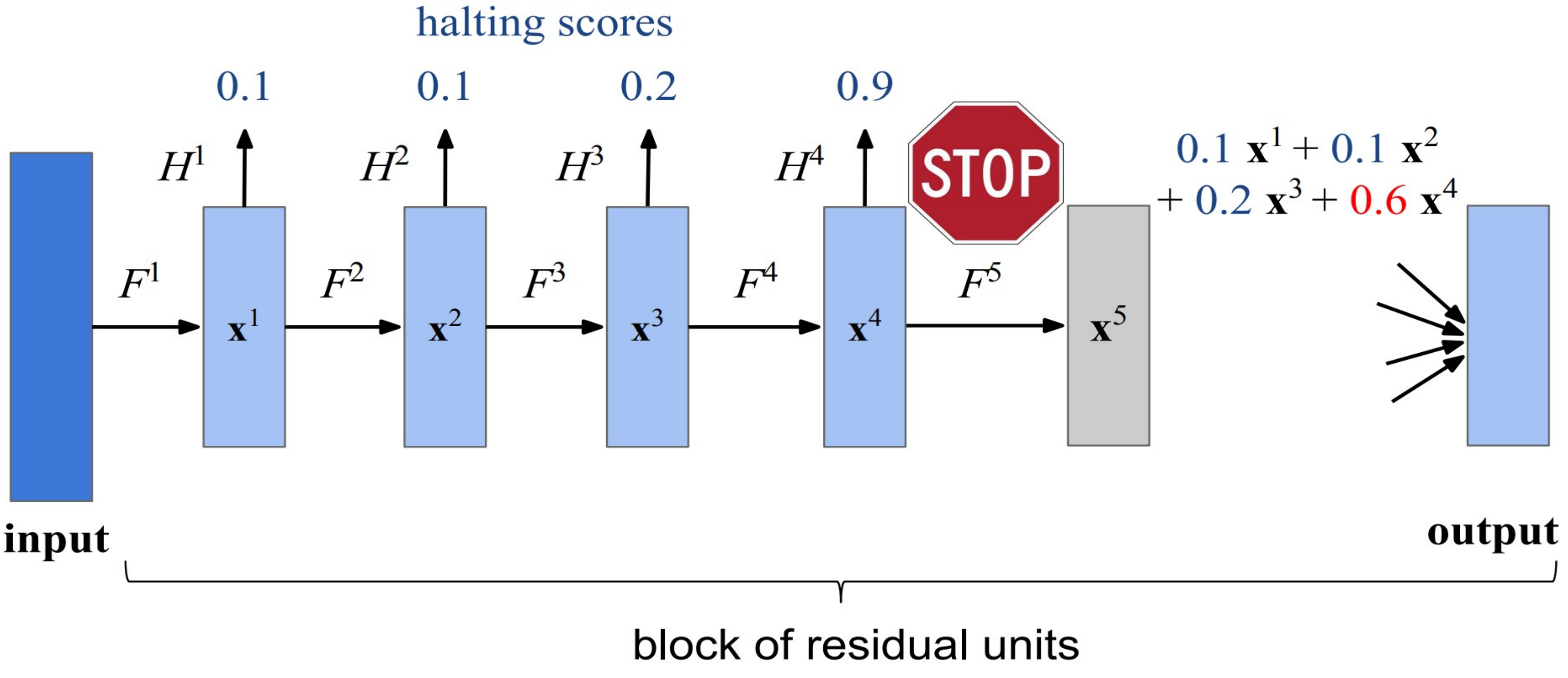
Acceleration by cascading



Acceleration by cascading



Adaptive Computation Time



Origina ACTl (for RNN)
https://arxiv.org/abs/1603.08983

Spatial ACT (conv)
https://tinyurl.com/sact-pdf

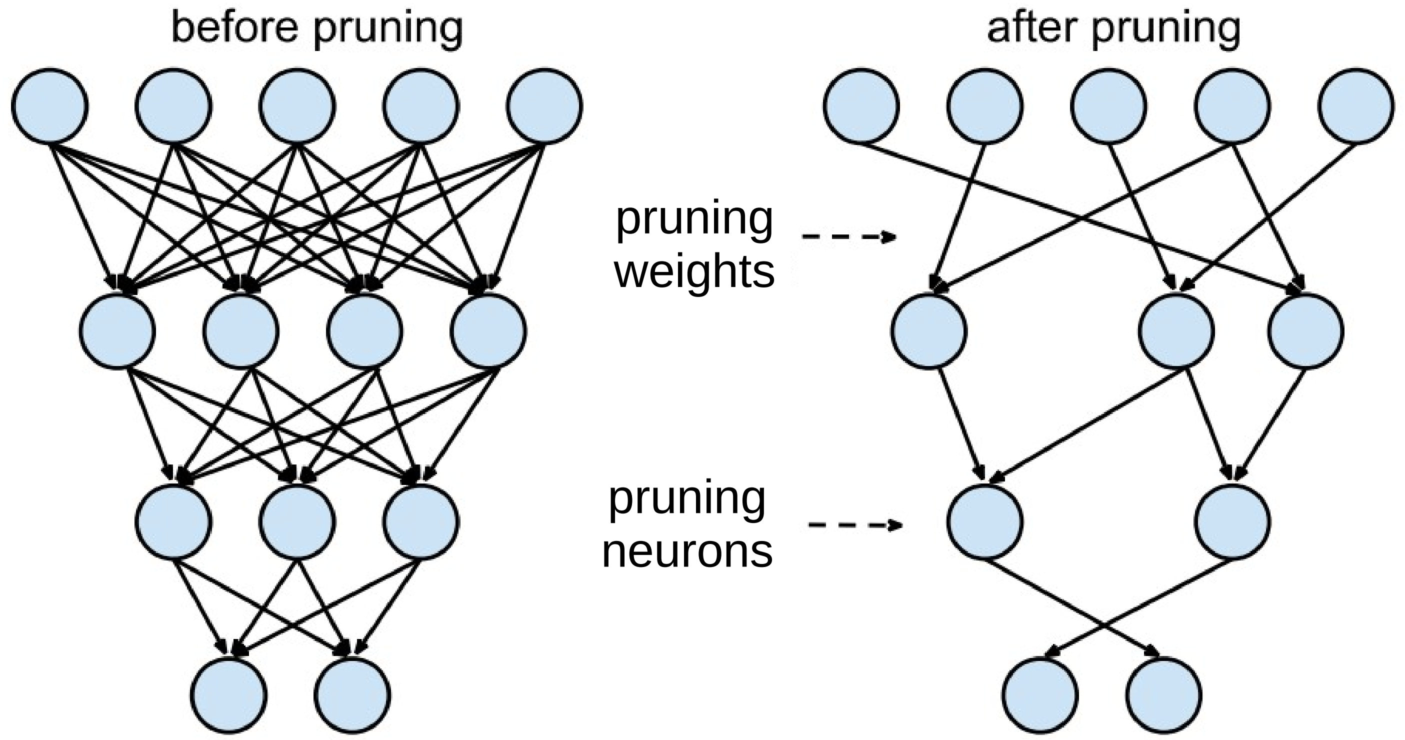
ACT Transformers
https://arxiv.org/abs/1807.03819

Compression by sparsification

Do we really need all D by D weights?

Compression by pruning

Do we really need all D by D weights?



Magnitude pruning

Drop ~5% smallest weights
from each layer every 1000 steps
(and keep training)

Reminds you of something?

Magnitude pruning

Drop ~5% smallest weights
from each layer every 1000 steps
(and keep training)

Reminds you of something?

See ML course, Optimal Brain Damage

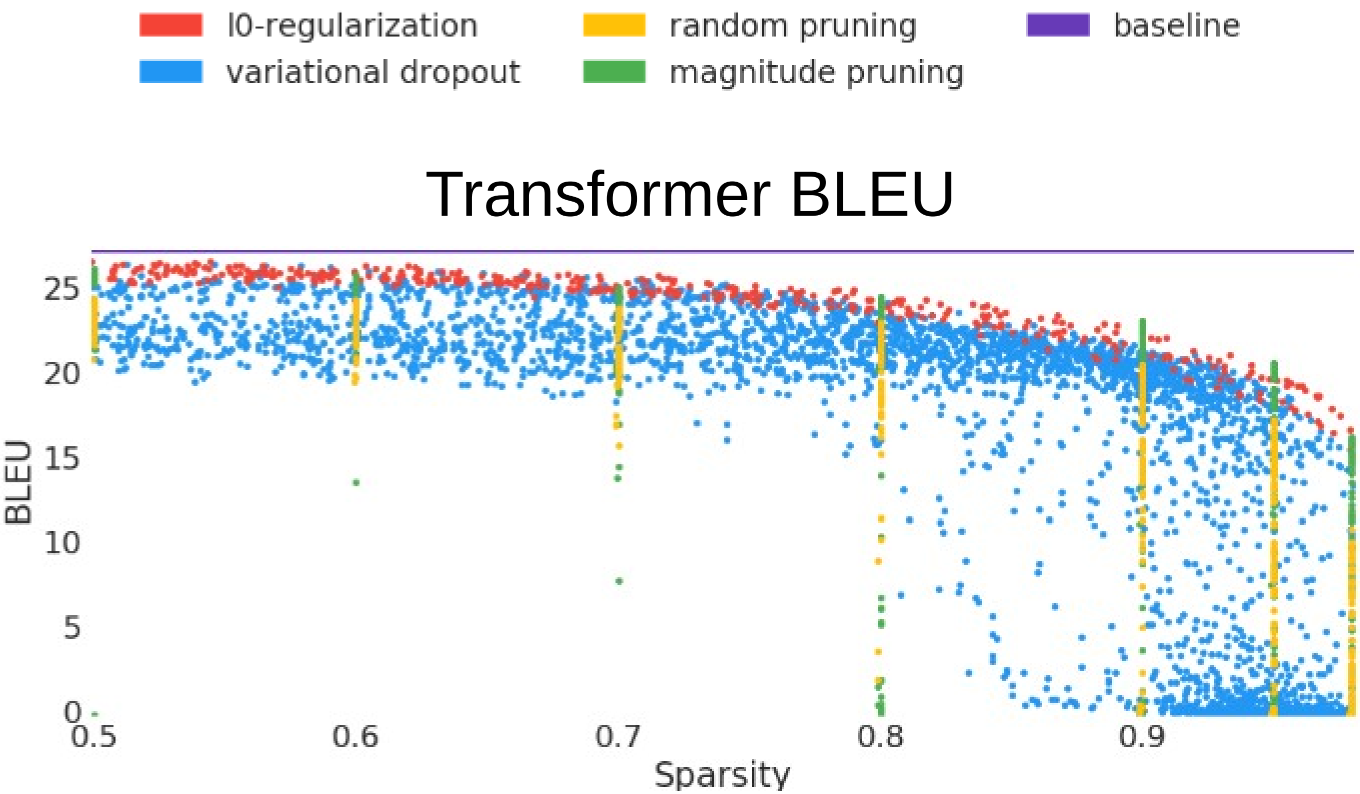
Pruning with L_0 regularization

Add a special regularizer that encourages dropping unnecessary weights

Whiteboard time!

Read more: https://arxiv.org/abs/1712.01312
Alternative: https://arxiv.org/abs/1701.05369

Which one works best?



Source https://arxiv.org/abs/1902.09574

Pruning with L_0 regularization

Add a special regularizer that encourages dropping unnecessary weights

Whiteboard time!

Pruning with L_0 regularization

Add a special regularizer that encourages dropping unnecessary weights

Can prune
- individual weights
- Individual neurons
- attention heads
- entire layers!

$$\lambda = 0.01$$

Pruning heads: https://lena-voita.github.io/posts/acl19_heads.html

Compression by sparsification

Как ужимать: prune/sparsify
можно сразу учить sparse (openai + та статья)

что умеет: только model size

Фенкс,
Квестионы?