Distributed Machine Learning
Efficient DL'23, Episode IV
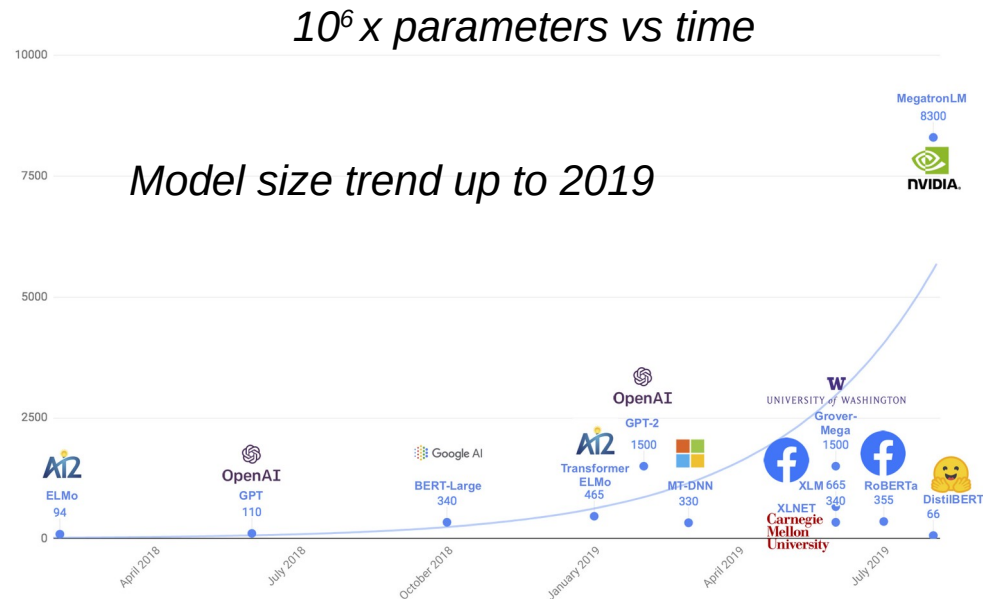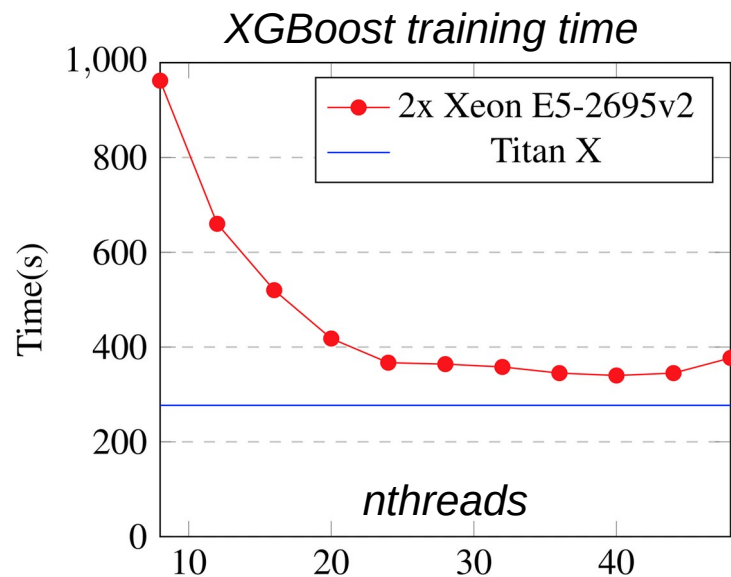
Yandex
Research
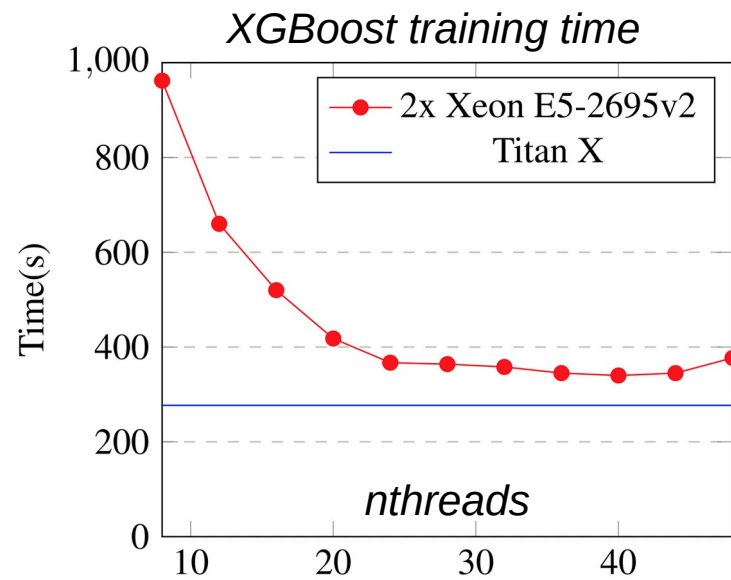
British Hedgehog Preservation Society

Зачем это всё?

XGBoost training time — 2x Xeon E5-2695v2, Titan X. Ось Y: Time(s) от 0 до 1,000. Ось X: nthreads от 10 до 40.

Зачем это всё?

XGBoost training time

10⁶ x parameters vs time

Model size trend up to 2019

BERT-Large Training Times on GPUs

Time	System	Number of Nodes	Number of V100 GPUs
47 min	DGX SuperPOD	92 x DGX-2H	1,472
67 min	DGX SuperPOD	64 x DGX-2H	1,024
236 min	DGX SuperPOD	16 x DGX-2H	256

*(single V100 – **over 2 weeks**)*

Зачем это всё?
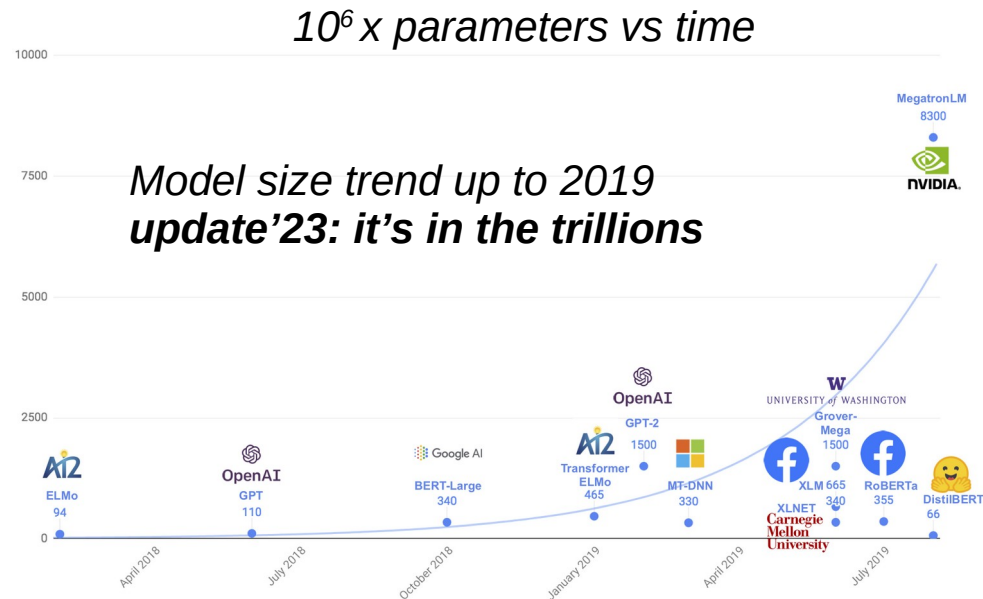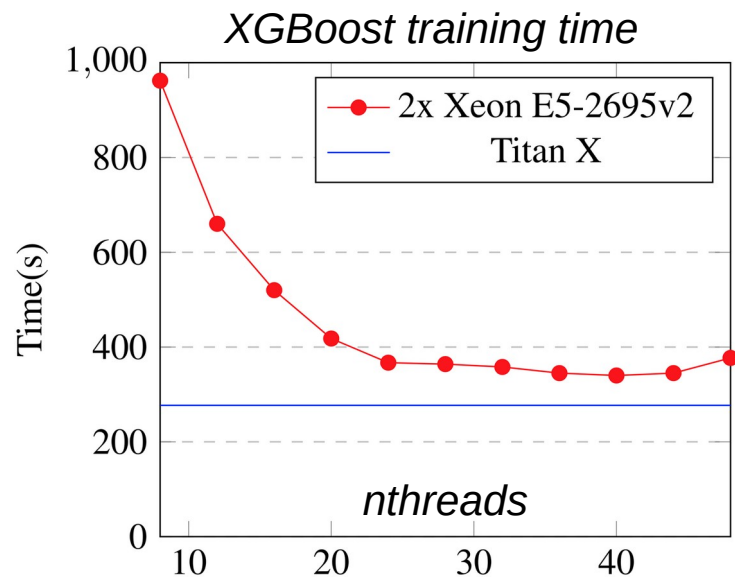
XGBoost training time

- 2x Xeon E5-2695v2
- Titan X

Time(s)

nthreads

10^6 x parameters vs time

Model size trend up to 2019
update'23: it's in the trillions

BERT-Large Training Times on GPUs

Time	System	Number of Nodes	Number of V100 GPUs
47 min	DGX SuperPOD	92 x DGX-2H	1,472
67 min	DGX SuperPOD	64 x DGX-2H	1,024
236 min	DGX SuperPOD	16 x DGX-2H	256

*(single V100 – **over 2 weeks**)*

Зачем мы тут?

Заставить много железяк вместе учить одну модель

Зачем мы тут?

Заставить много железяк вместе учить одну модель

понять общие подходы

закодить своими руками

на python / pytorch

TL;DR our plan
lectures 4,5,6

4) Distributed machine learning
Embeddings or log.regression with tons of training data

TL;DR our plan
lectures 4,5,6

4) Distributed machine learning
Embeddings or log.regression with tons of training data

5) Data-parallel deep learning
Train BERT-base on wikipedia in 20 minutes or less

TL;DR our plan
lectures 4,5,6

4) Distributed machine learning
Embeddings or log.regression with tons of training data

5) Data-parallel deep learning
Train BERT-base on wikipedia in 20 minutes or less

6) Model-parallel deep learning
Fine-tune and deploy models with 100B parameters

TL;DR our plan
lectures 4,5,6

4) Distributed machine learning
Embeddings or log.regression with tons of training data

5) Data-parallel deep learning
Train BERT-base on wikipedia in 20 minutes or less

6) Model-parallel deep learning
Fine-tune and deploy models with 100B parameters

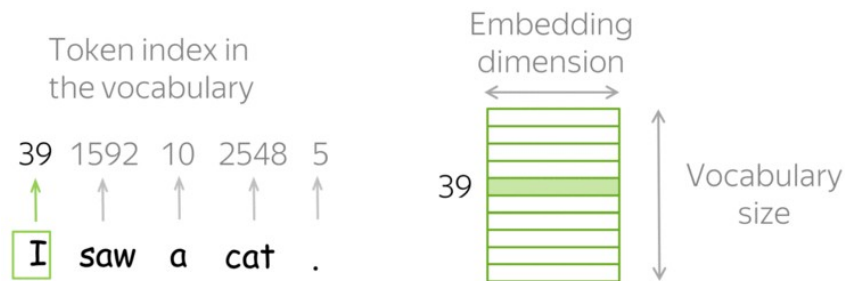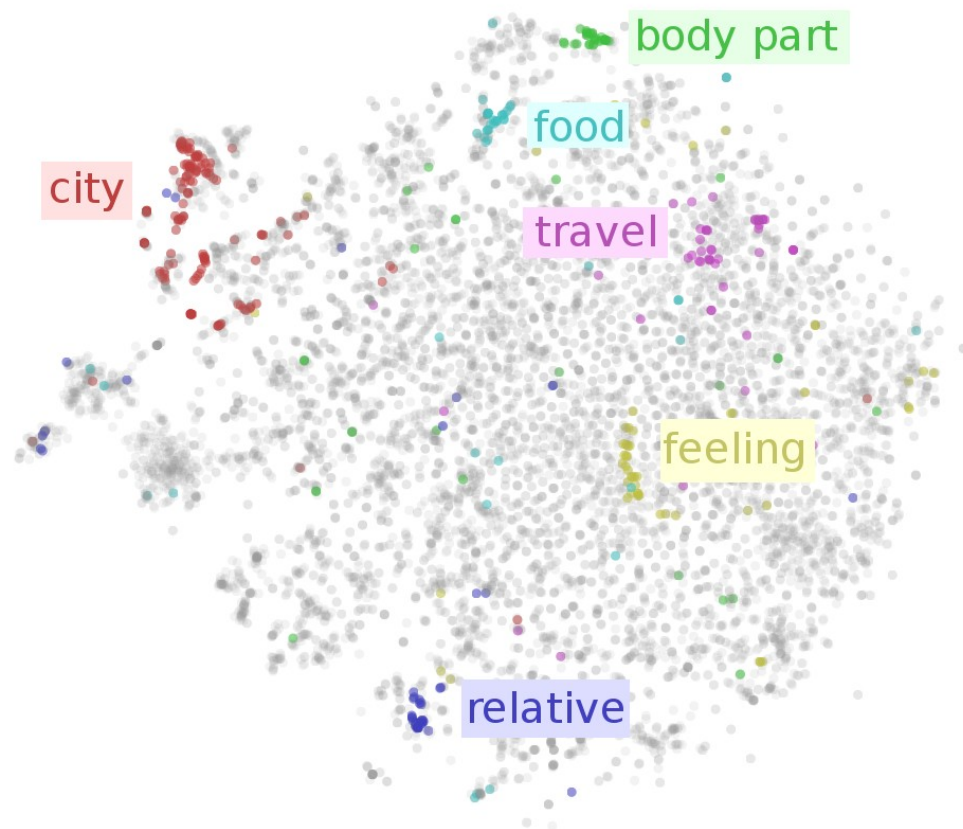*like OPT-175B, BLOOM-176B, YALM, GLM, Galactica*

TL;DR our plan
lectures 4,5,6

4) Distributed machine learning
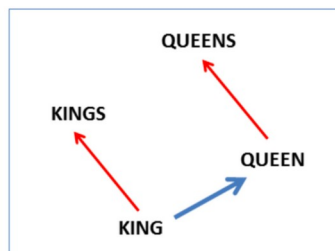*Embeddings or log.regression with tons of training data*

Today: learn the basics behind it all

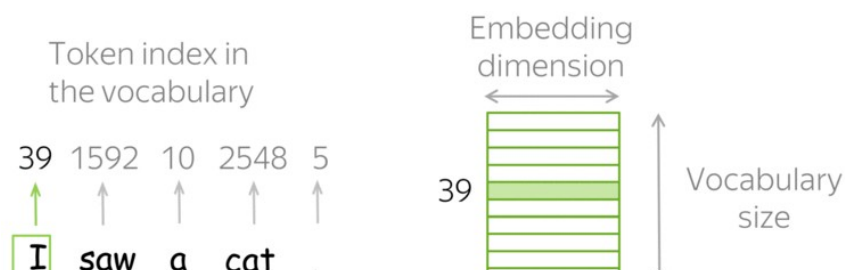# Example problem: word embeddings

Token index in the vocabulary

39 1592 10 2548 5

I saw a cat .

Embedding dimension

39

Vocabulary size

semantic: $v(king) - v(man) + v(woman) \approx v(queen)$

syntactic: $v(kings) - v(king) + v(queen) \approx v(queens)$

WOMAN
AUNT
MAN
UNCLE
QUEEN
KING

QUEENS
KINGS
QUEEN
KING

city
body part
food
travel
feeling
relative

Image source: Lena's blog, Ruder's blog

Example problem: word embeddings

Token index in the vocabulary

39 1592 10 2548 5

I saw a cat

Embedding dimension

39

Vocabulary size

city

body part

food

travel
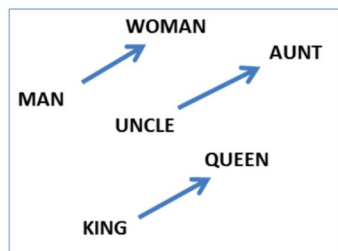
v(kings) - v(king) + v(queen) ~ v(queens)
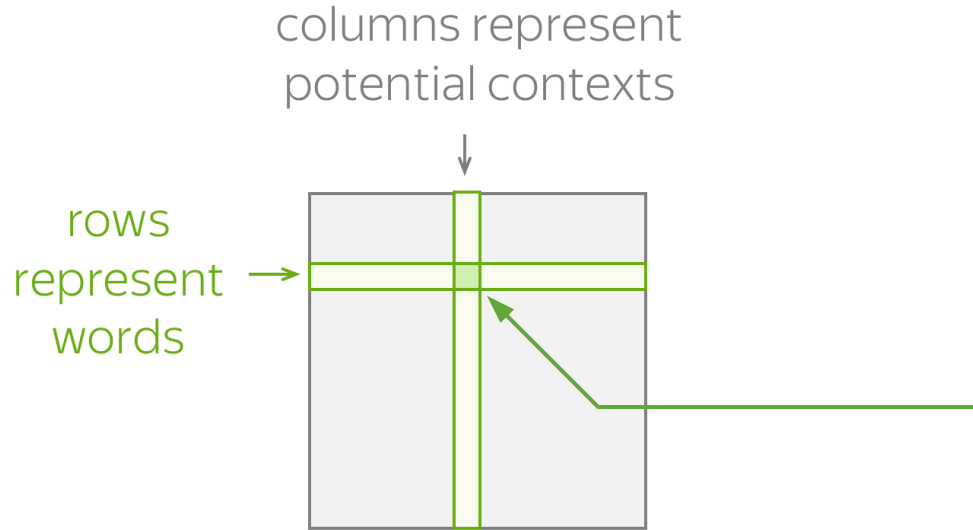
This is an example problem, don't focus on NLP too much
computationally similar: large-scale LogReg, SVD, GBDT

WOMAN
MAN AUNT
UNCLE
QUEEN
KING

QUEENS
KINGS
QUEEN
KING

relative

Image source: Lena's blog, Ruder's blog

Co-occurence matrix

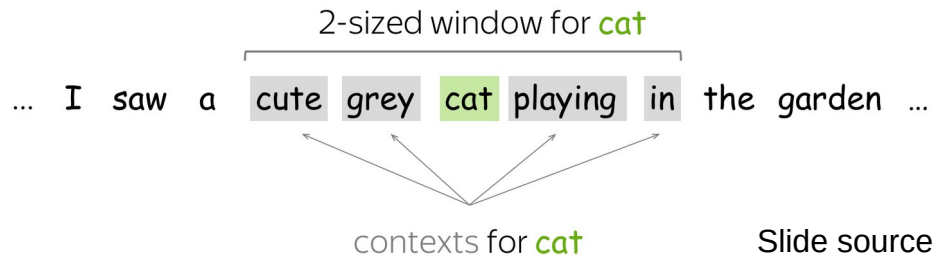columns represent
potential contexts

rows
represent
words
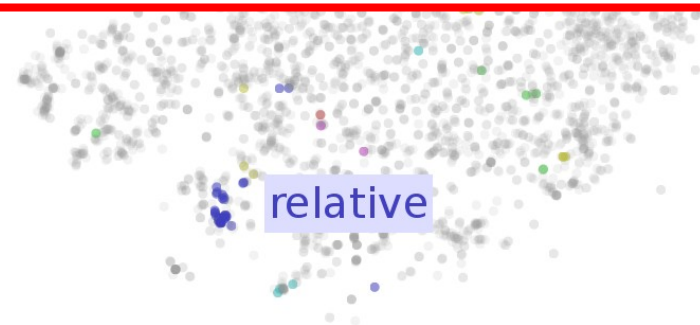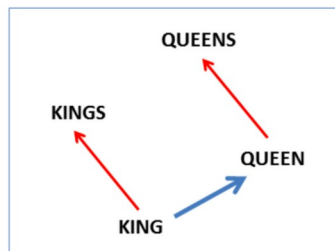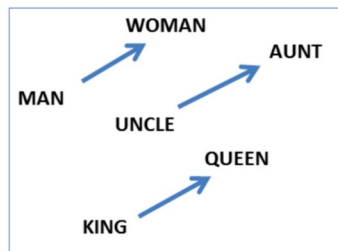
Context:

- surrounding words in a L-sized window

Matrix element:

- N(w, c) – number of times word w appears in context c

2-sized window for cat

… I saw a cute grey cat playing in the garden …

contexts for cat
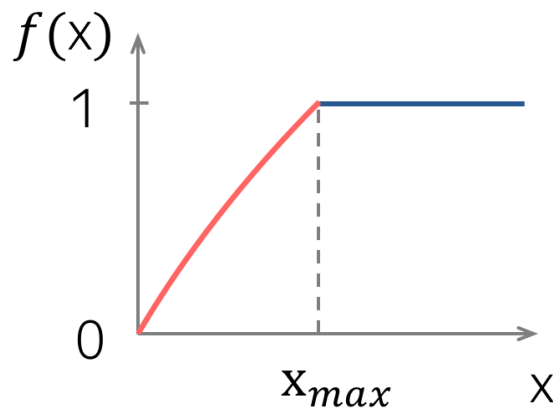
Note: in our case, N is symmetric!

GloVe

context vector   word vector   bias terms (also learned)

$$L = \sum_{i \neq j} w(N(i,j)) \cdot (\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j))^2$$

Slide source: Lena's blog

GloVe

context vector   word vector   bias terms (also learned)

$$L = \sum_{i \neq j} w(N(i,j)) \cdot (\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j))^2$$

Weighting function to:

- penalize rare events
- not to over-weight frequent events

$f(x)$

1

0

x_{max} X

$$\begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max}, \\ 1 & \text{otherwise.} \end{cases}$$
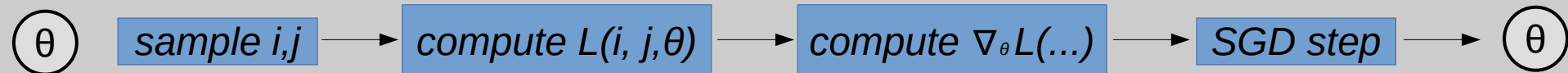
$$\alpha = 0.75, \, x_{max} = 100$$

Slide source: Lena's blog

GloVe

context vector   word vector   bias terms (also learned)

$$L = \sum_{i \neq j} w(N(i,j)) \cdot (\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j))^2$$

Learn more: *lena-voita.github.io/nlp_course/word_embeddings.html*
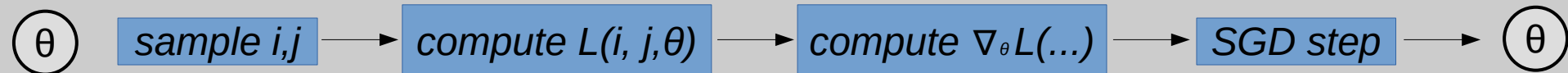
So how do we train 'em?

Training Step

θ

sample i,j → compute L(i, j,θ) → compute $\nabla_\theta L(...)$ → SGD step → θ

current
params
θ: {v, b}

$$L=\sum_{i \neq j} w\left(N\left(i,j\right)\right)\cdot\left(\langle\vec{v}_i,\vec{v}_j\rangle+b_i+b_j-\log N\left(i,j\right)\right)^2$$

updated
params

Training Step

θ → sample i,j → compute L(i, j,θ) → compute $\nabla_\theta L(...)$ → SGD step → θ

current
params
θ: {v, b}

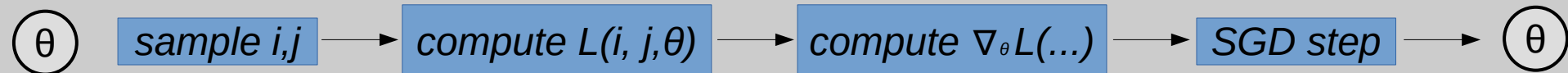$$L = \sum_{i \neq j} w\left(N(i,j)\right) \cdot \left(\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j)\right)^2$$

updated
params

Trainable parameters:

V matrix   b

a

cat

saw

Training Step

current
params
θ: {v, b}

$$L = \sum_{i \neq j} w\left(N(i,j)\right) \cdot \left(\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j)\right)^2$$

updated
params

Trainable parameters:

V matrix   *b*

a

cat

saw
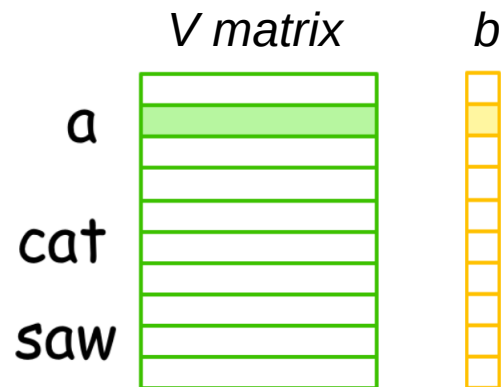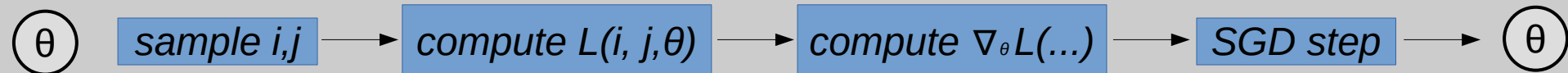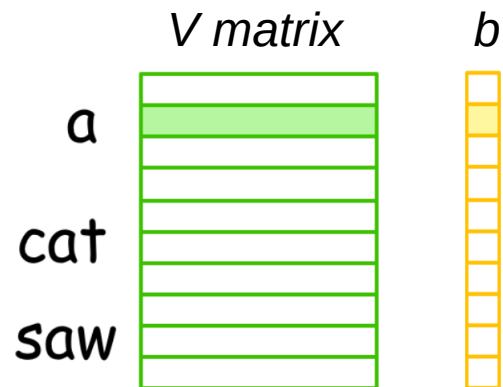
***How do we go faster
with 8 CPU cores?***

Training Step

current
params
θ: {v, b}

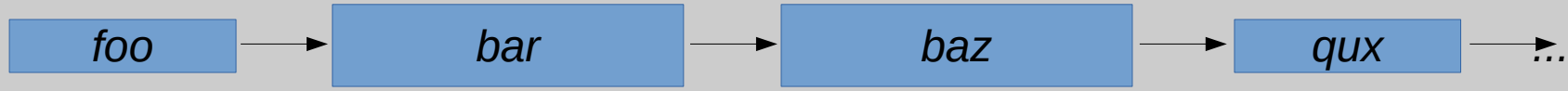$$L = \sum_{i \neq j} w(N(i,j)) \cdot (\langle \vec{v}_i, \vec{v}_j \rangle + b_i + b_j - \log N(i,j))^2$$

updated
params

Trainable parameters:

V matrix   *b*

a

cat

saw

[let's formalize your ideas]

Rules: Process

Process:
- Runs some code
- Has some memory
- No one else can access your memory

Rules: Process

foo → bar → baz → qux → ...

Process:
- Runs some code
- Has some memory
- No one else can access your memory

OH, REALLY.

YOU SO SURE ABOUT THAT.
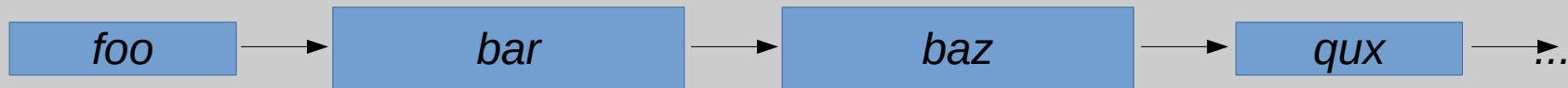
makeameme.org

Rules: Process

```
foo → bar → baz → qux → ...
```
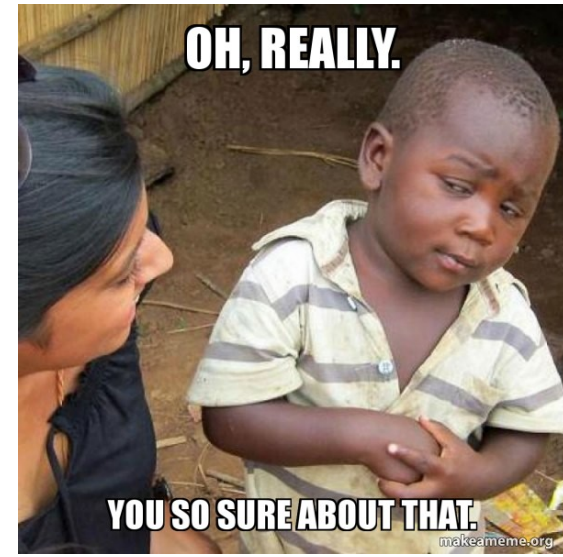
Process:
- Runs some code
- Has some memory
- No one else can access your memory*

* – not if you use shared memory

Rules: Process

Process:
- Runs some code
- Has some memory
- No one else can access your memory*[†]

* – not if you use shared memory
[†] – superuser can still do that (os-dependent)

Rules: Process

Process:
- Runs some code
- Has some memory
- No one else can access your memory*[†][‡]

* – not if you use shared memory
[†] – superuser can still do that (os-dependent)
[‡] – attacker can do that through spectre/meltdown/etc

Rules: Process

Process:
- Runs some code
- Has some memory
- No one else **should** access your memory*†‡

*†‡ – not relevant for this course
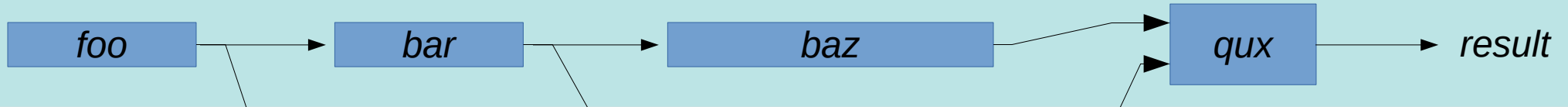
Rules: Process
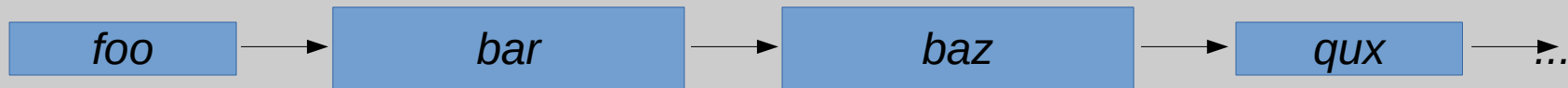
Process:
- Runs some code
- Has some memory
- No one else **should** access your memory*[†][‡]

*[†][‡] – not relevant for this course

Q: How do we make processes work together?
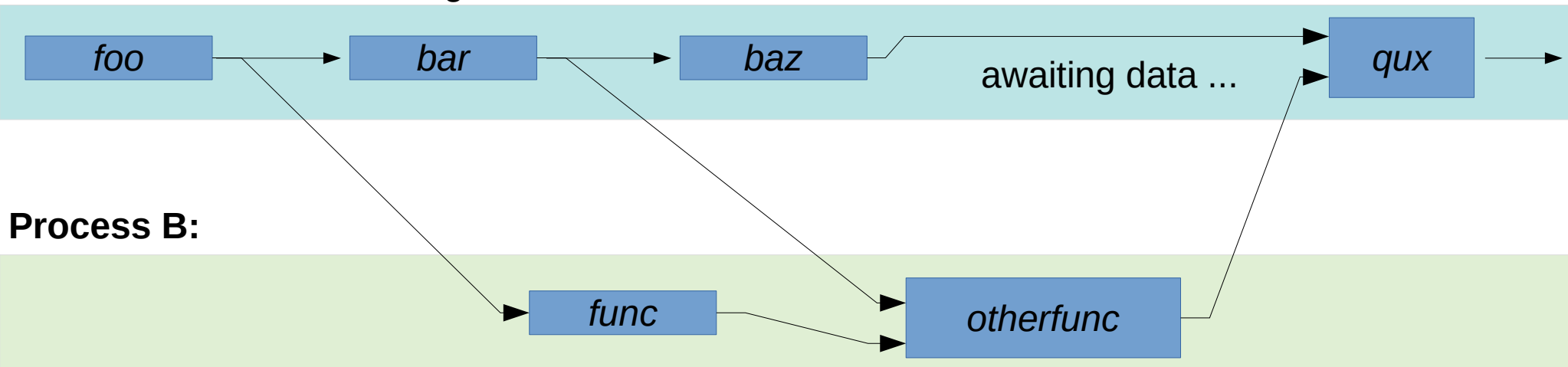
Rules: Channel / Pipe

Process A:

Process B:

Channel (pipe):
- Communication in O(message size)
- Asynchronous read/write
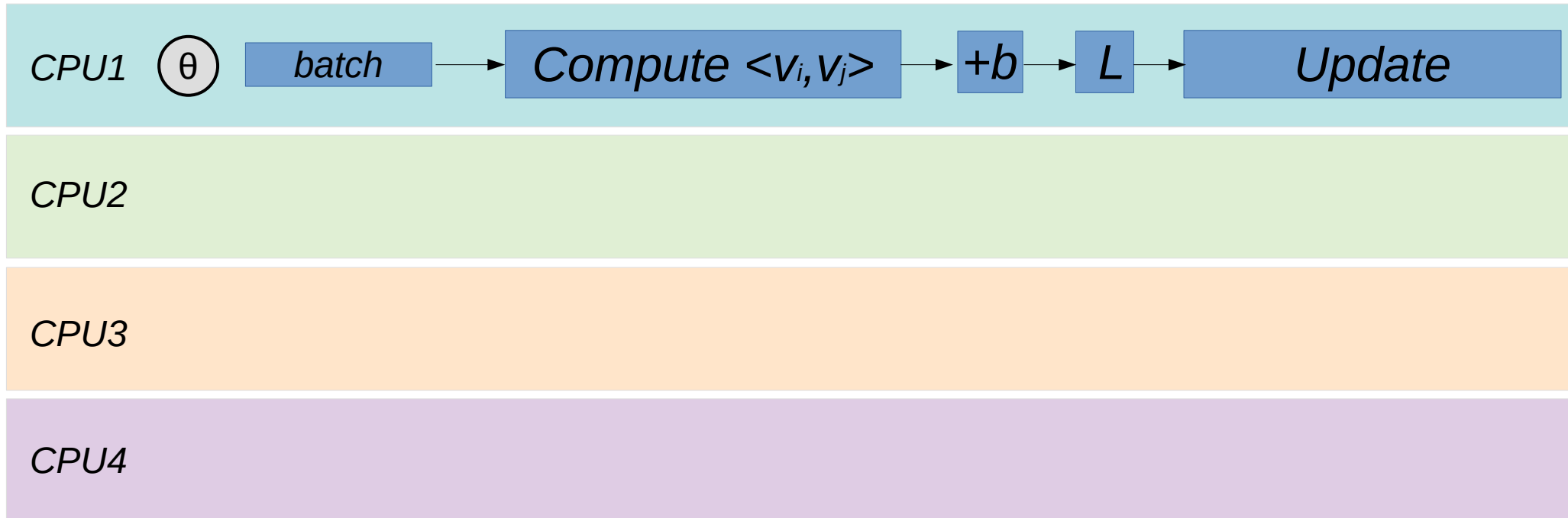
MP Rules

Process A: not waiting on write

foo → bar → baz   awaiting data ...   qux →

Process B:

func → otherfunc

Channel (pipe):
- Communication in O(message size)
- **Asynchronous** read/write
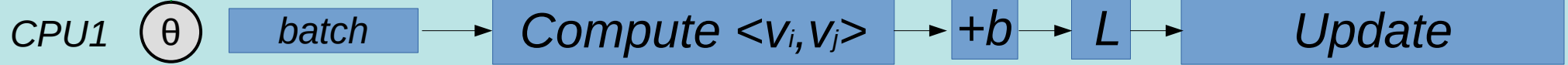
Details are (not) important
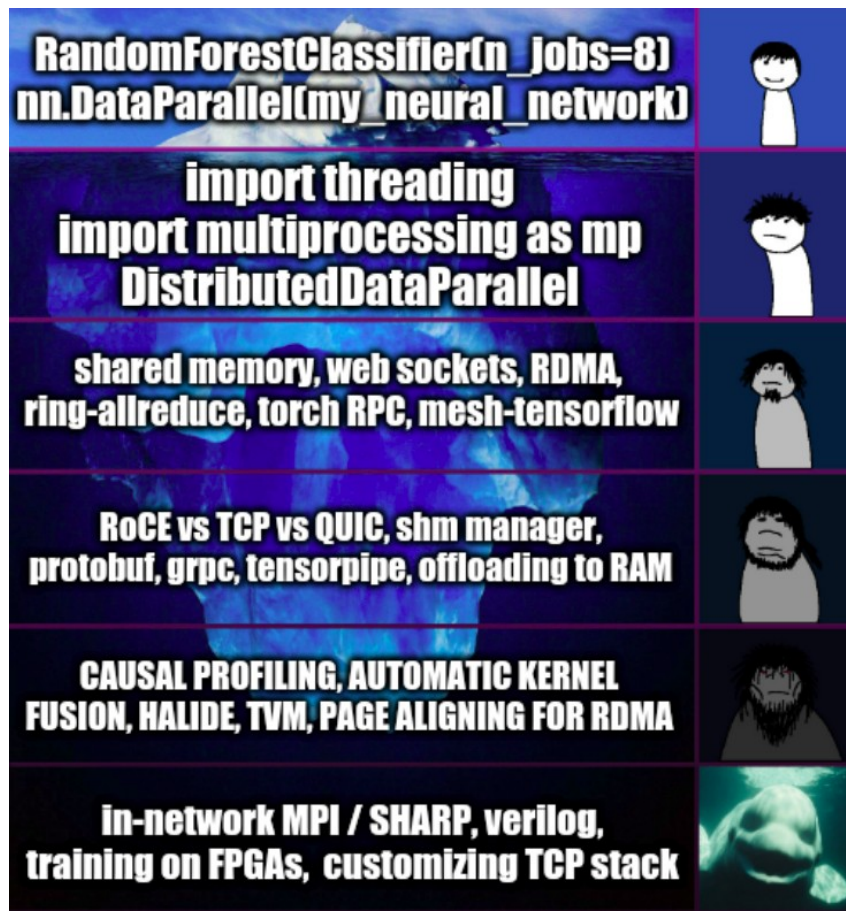
Operation parallelism

run algorithm in parallel without changing the math

Operation parallelism

model weights

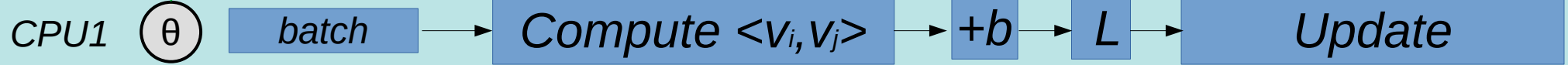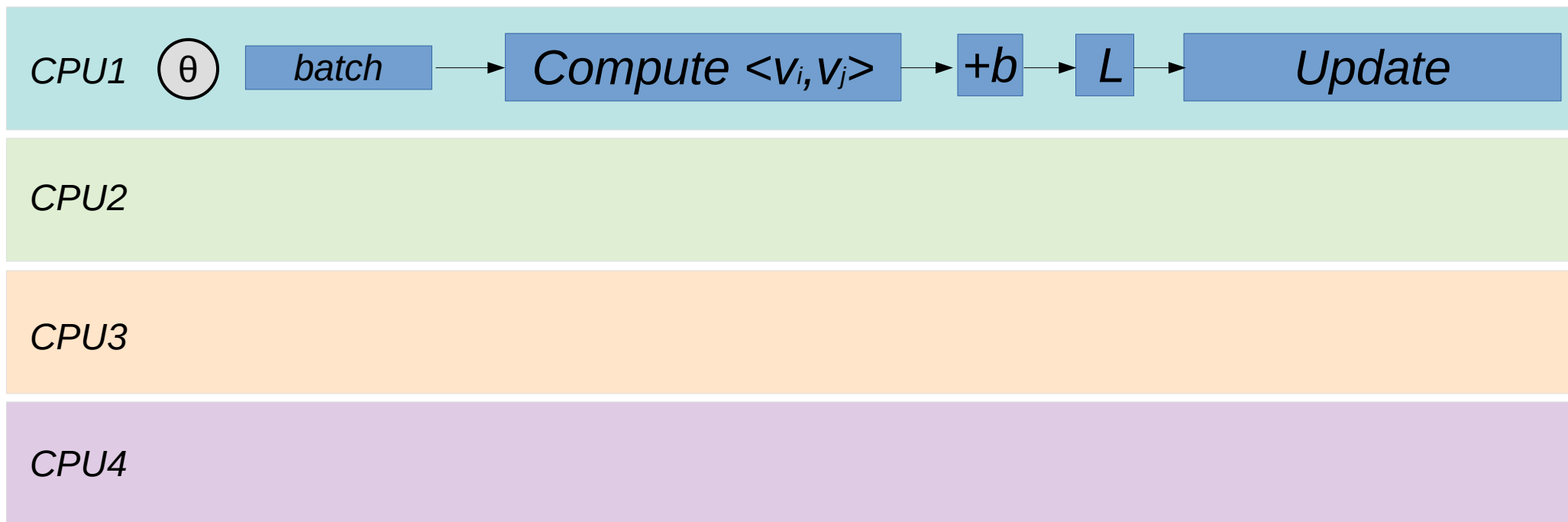*run algorithm in parallel without changing the math*

CPU1 θ batch → Compute $<v_i, v_j>$ → $+b$ → L → Update

CPU2

CPU3

CPU4

Operation parallelism

model weights *run algorithm in parallel without changing the math*

CPU1 θ [batch] → [Compute $<v_i, v_j>$] → [$+b$] → [L] → [Update]
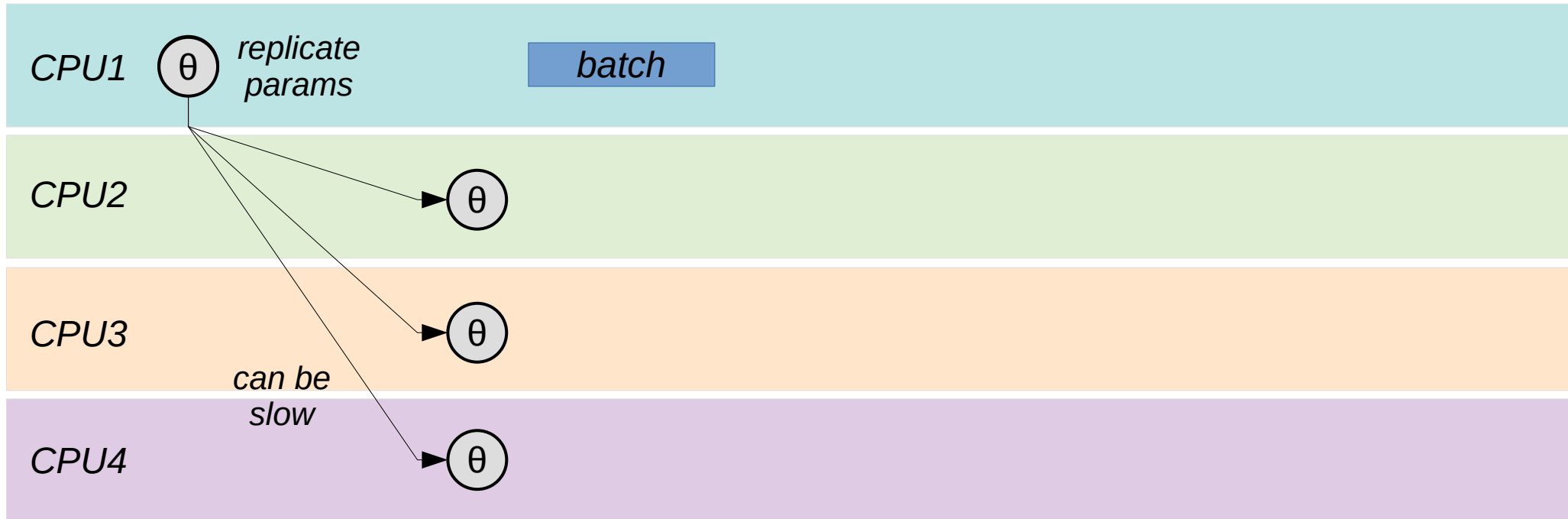
CPU2

CPU3

CPU4

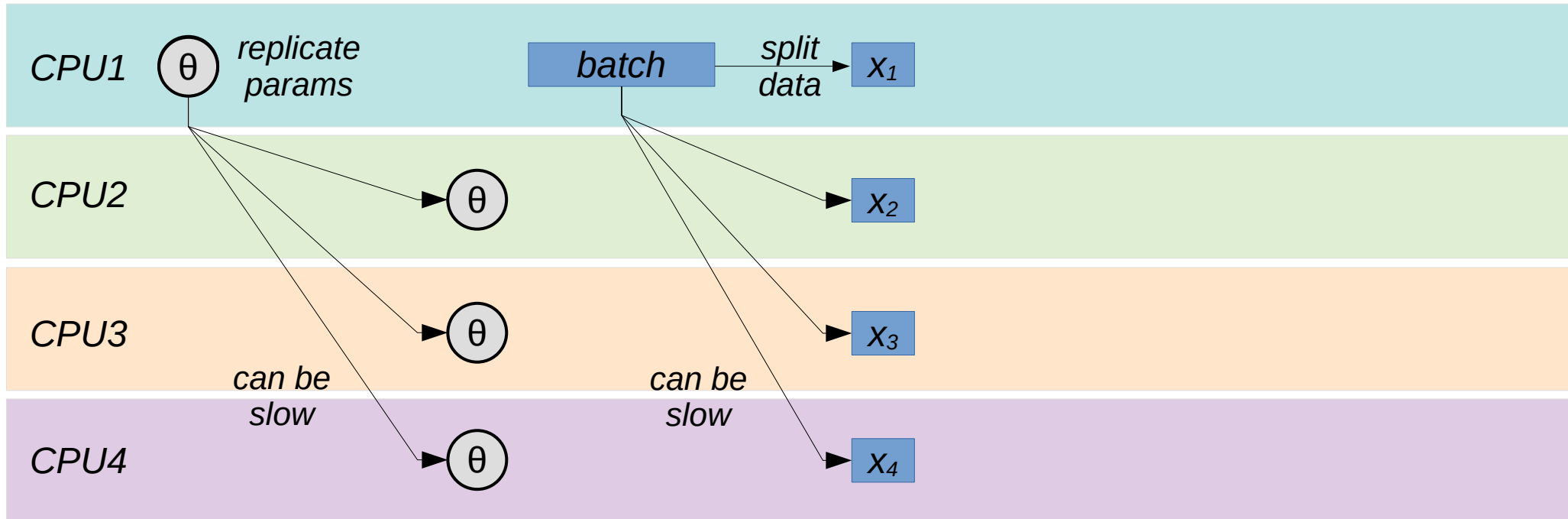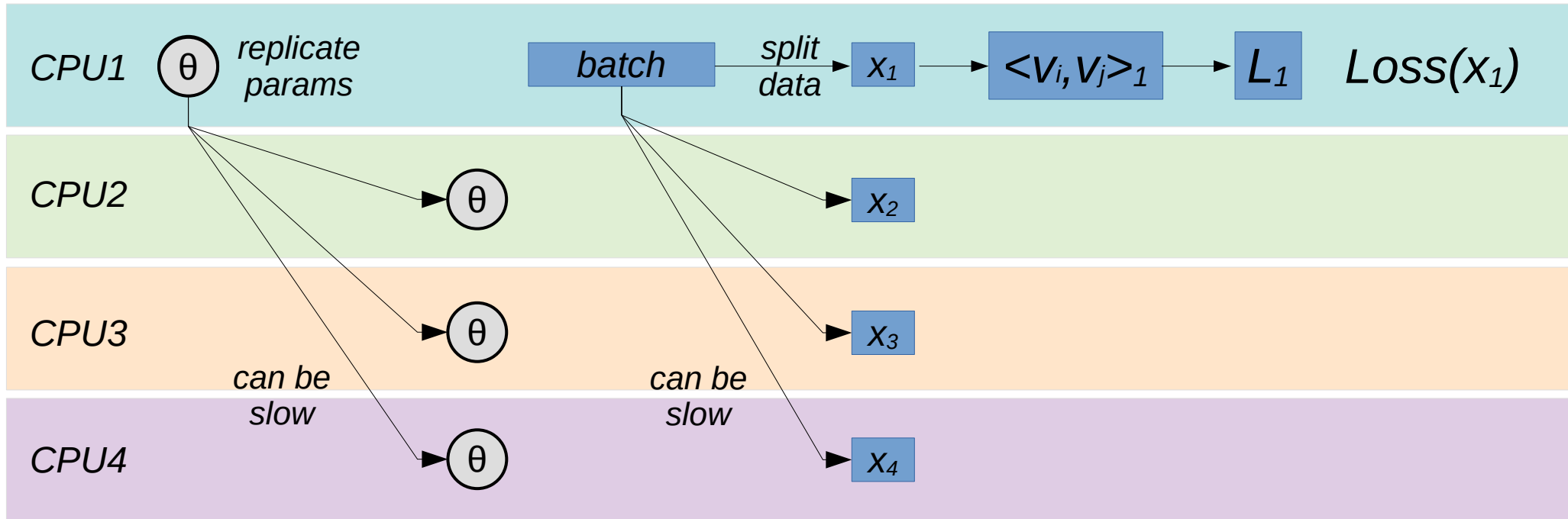# "Data parallelism"

*Each process runs **full** model on **some** samples*

"Data parallelism"

*Each process runs **full** model on **some** samples*

"Data parallelism"

*Each process runs **full** model on **some** samples*

"Data parallelism"

*Each process runs **full** model on **some** samples*

"Data parallelism"

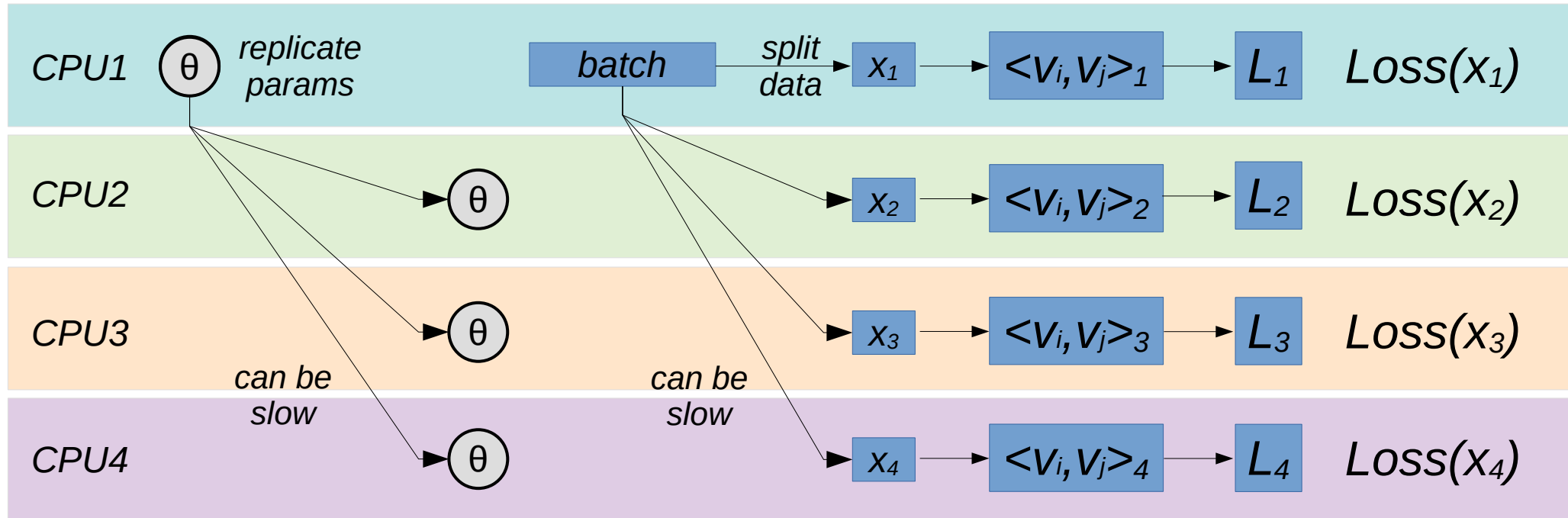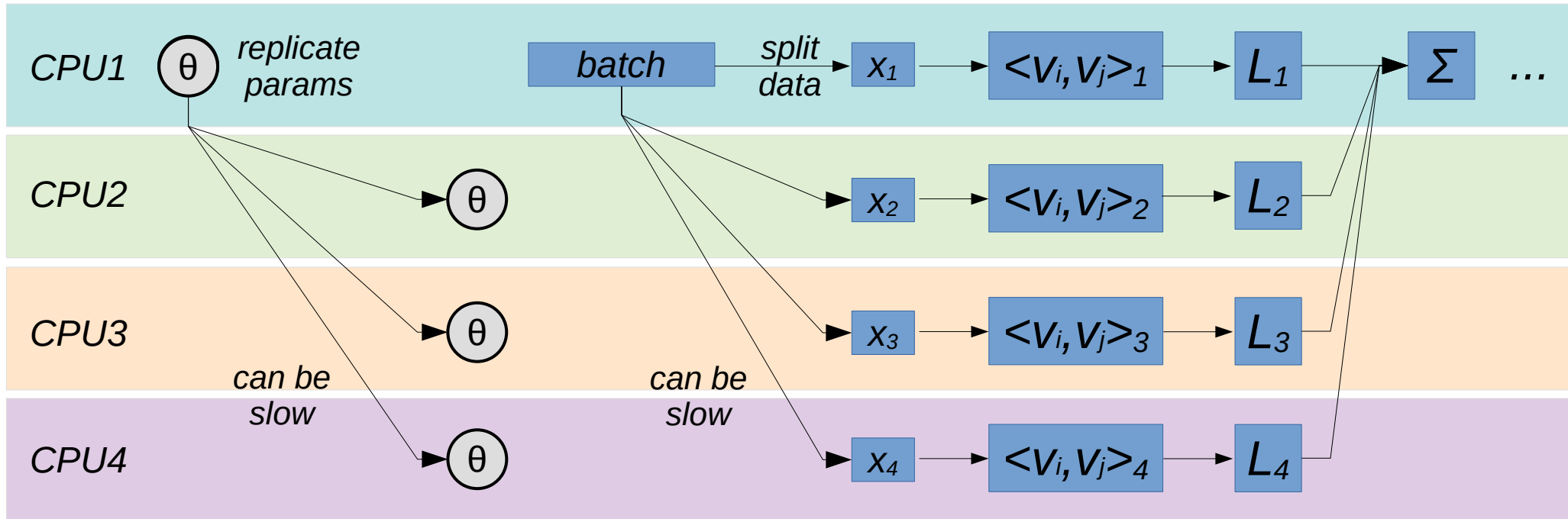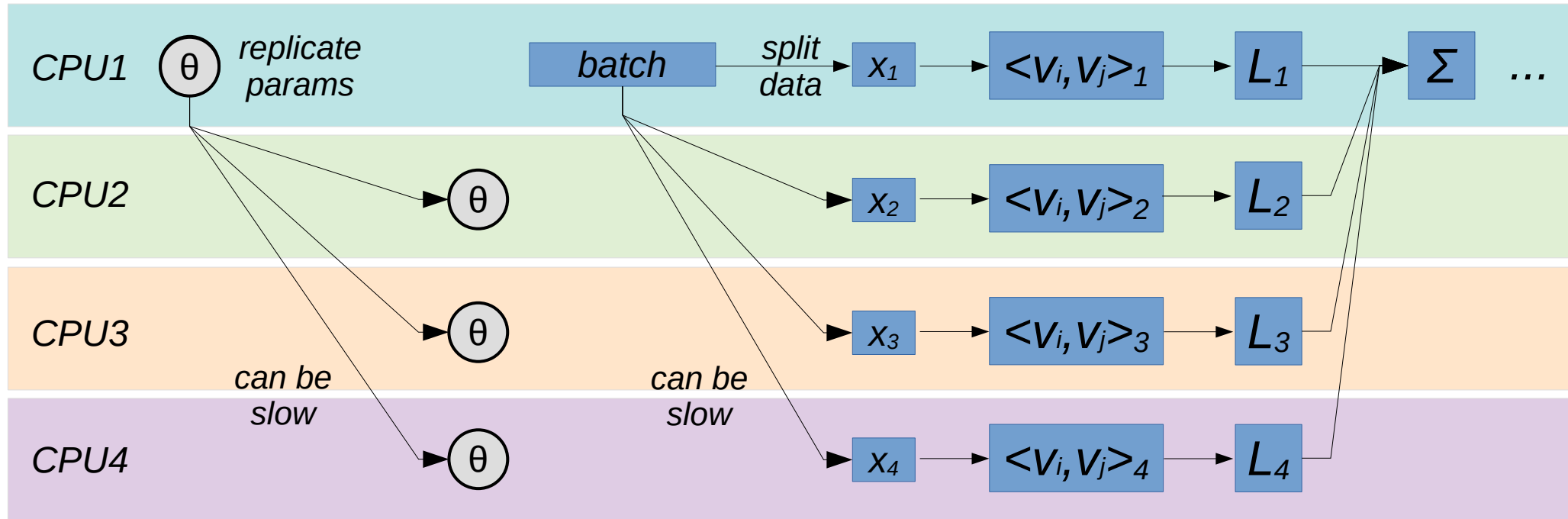*Each process runs **full** model on **some** samples*

"Data parallelism"

*Each process runs **full** model on **some** samples*

"Data parallelism"

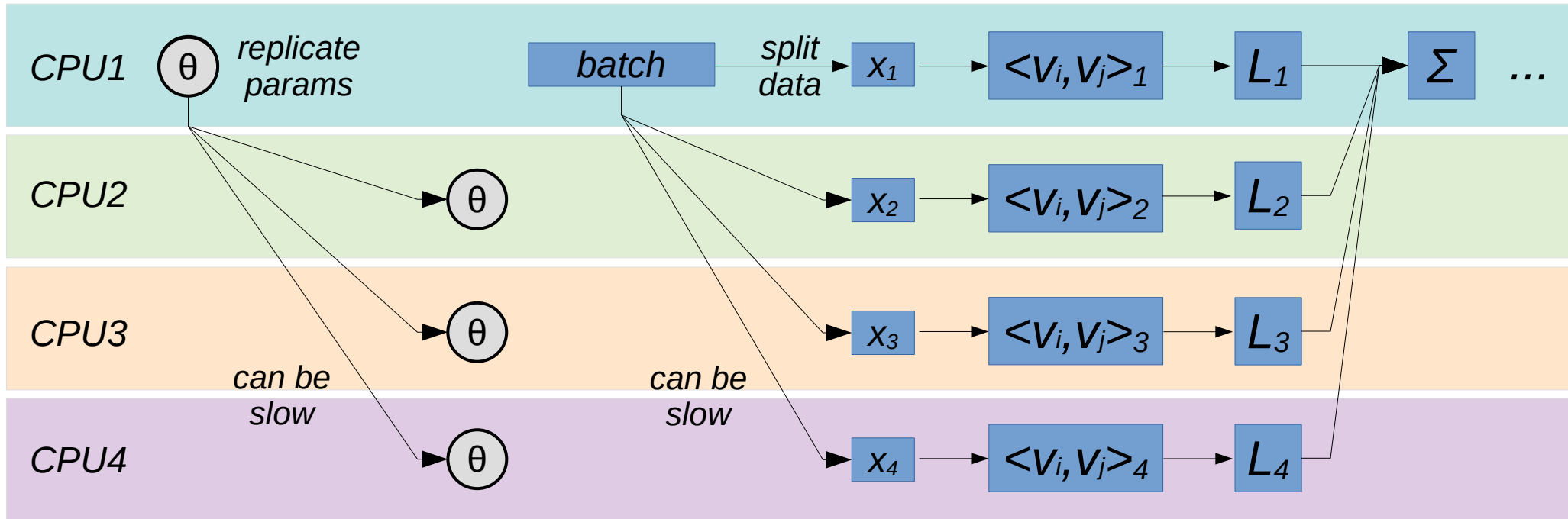*Each process runs **full** model on **some** samples*
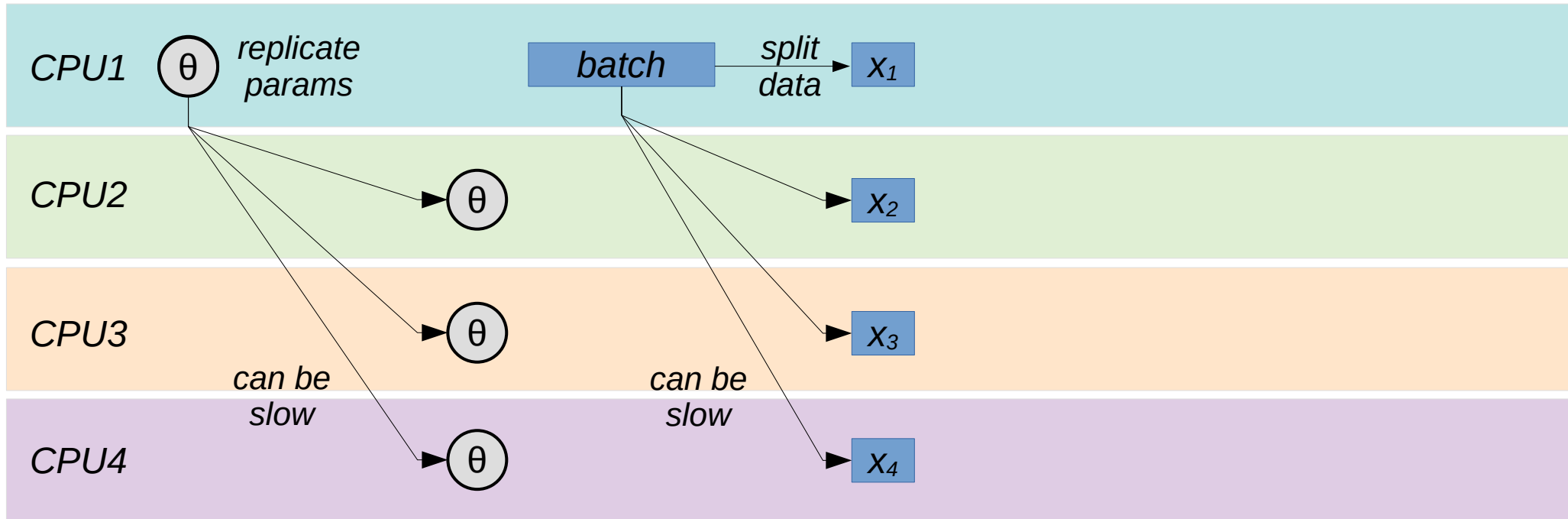
"Data parallelism"

Q: Is it guaranteed to be faster?

"Data parallelism"

Q: Is it guaranteed to be faster?

No, sending data may take longer than computing

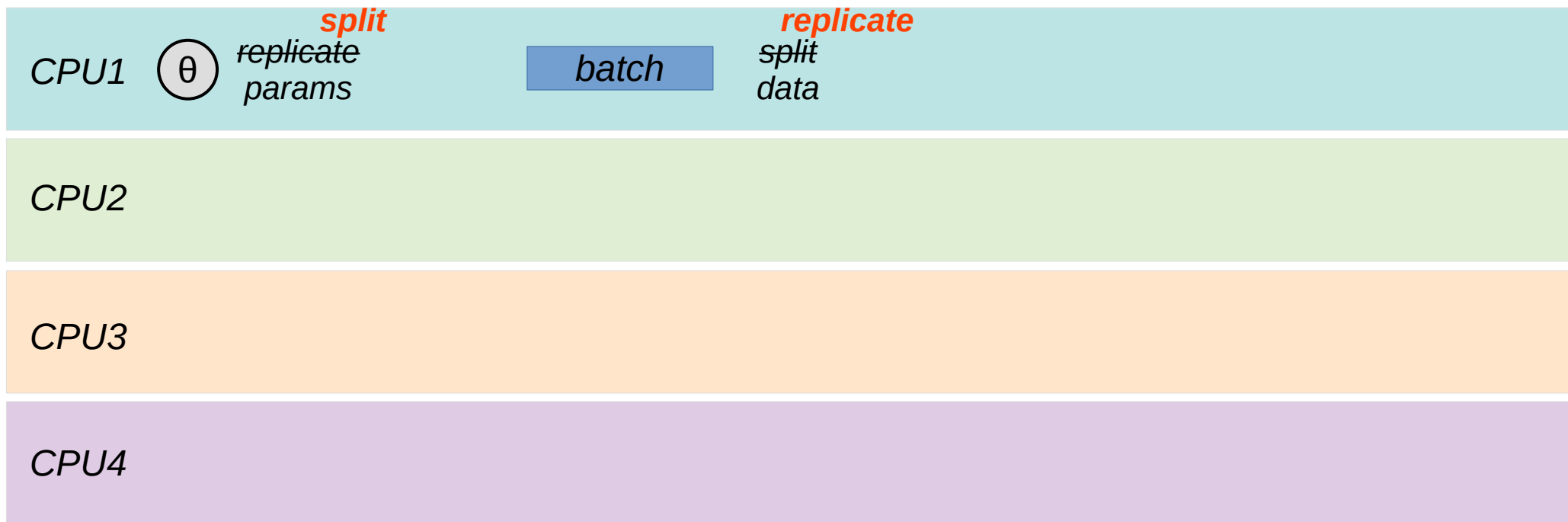|						
CPU1	(θ) replicate params		batch → split data →	x_1 →	$<v_i,v_j>_1$ →	L_1 →	Σ ...
CPU2		(θ)		x_2 →	$<v_i,v_j>_2$ →	L_2
CPU3	can be slow	(θ)	can be slow	x_3 →	$<v_i,v_j>_3$ →	L_3
CPU4		(θ)		x_4 →	$<v_i,v_j>_4$ →	L_4

"Data parallelism"

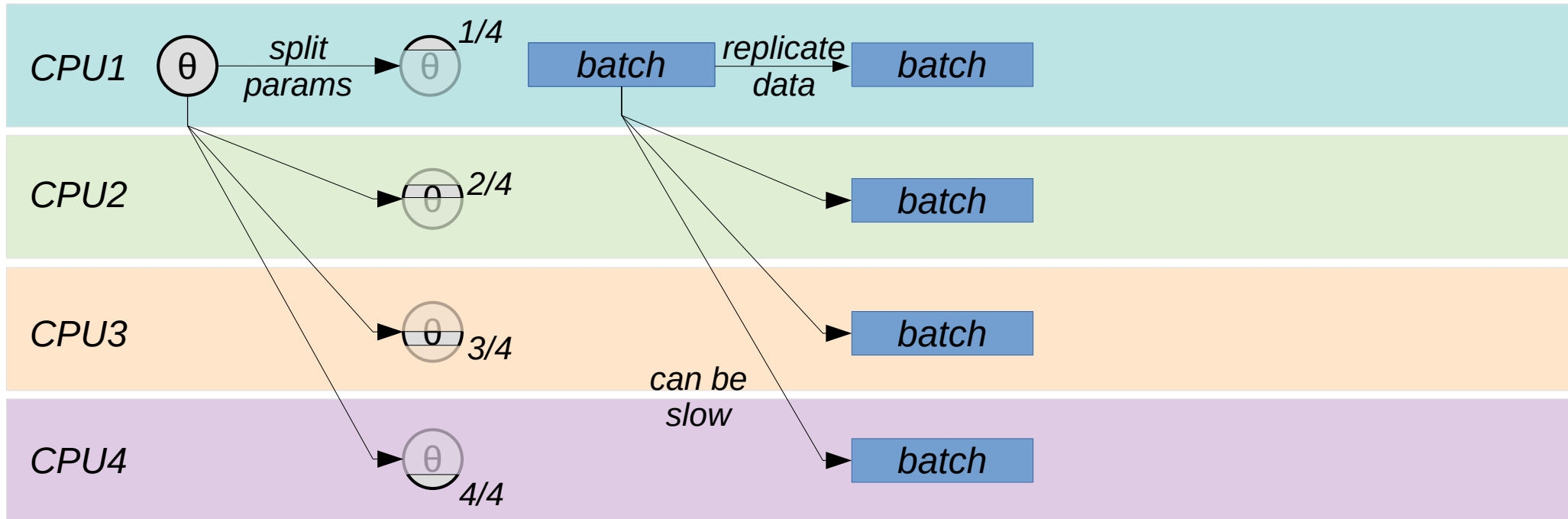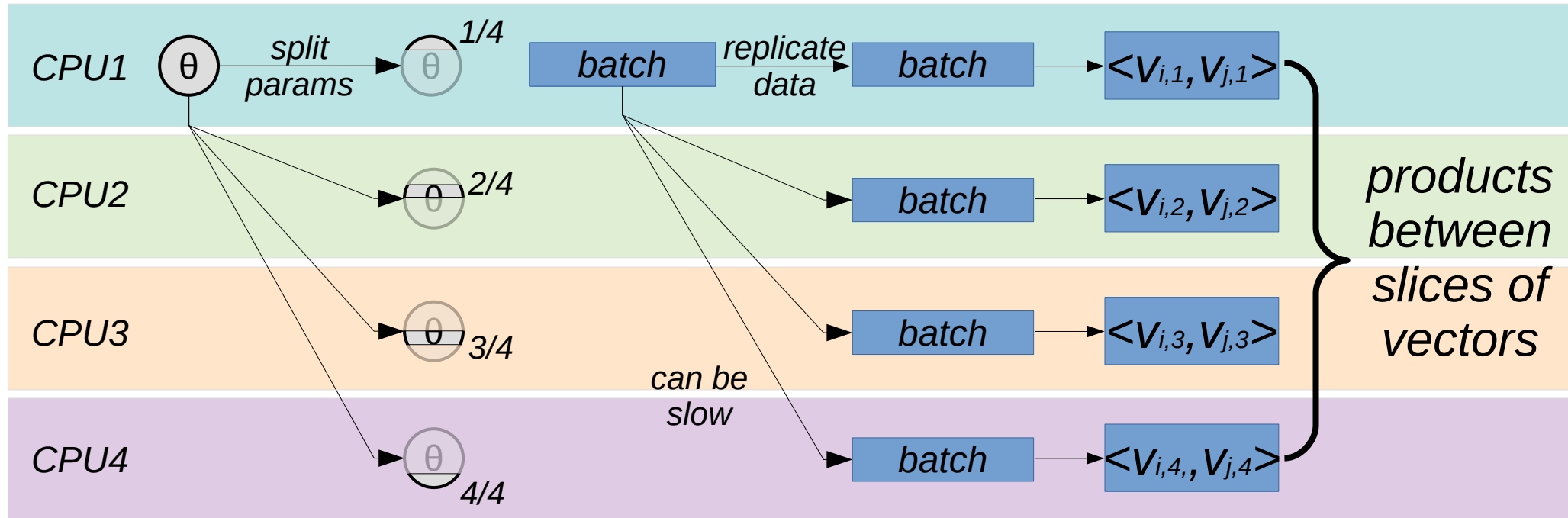*Q: any other way to compute <vi, vj> in parallel?*

"Model parallelism"

*Each process runs **partial** model on **all** samples*

"Model parallelism"

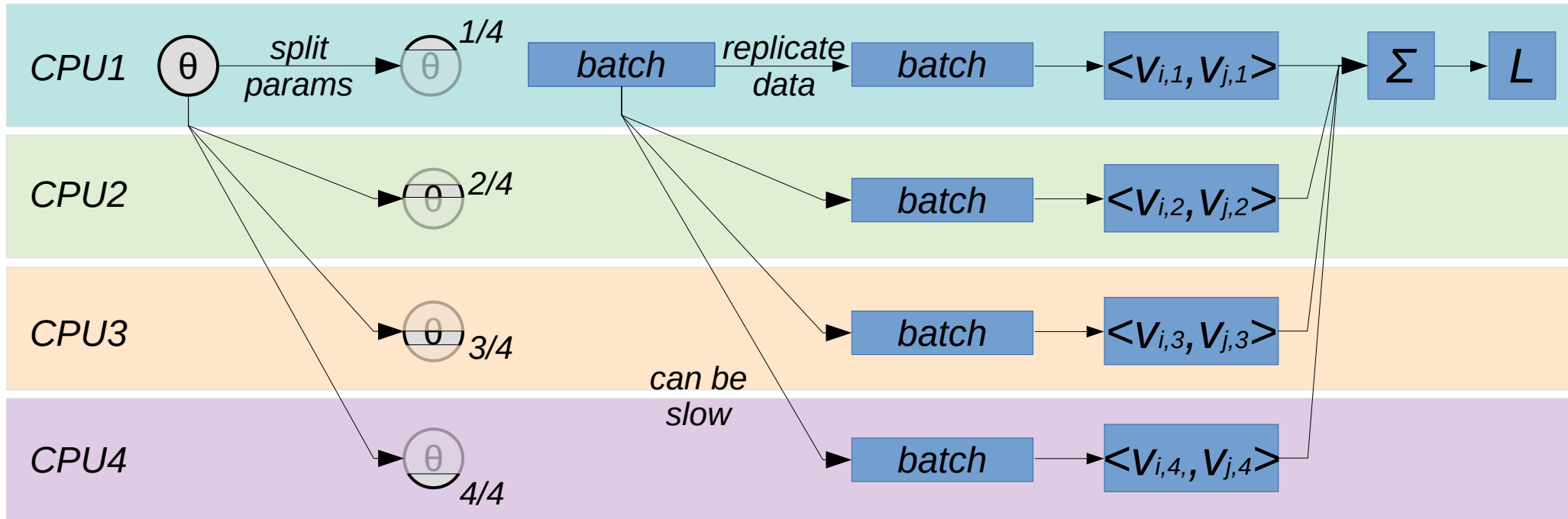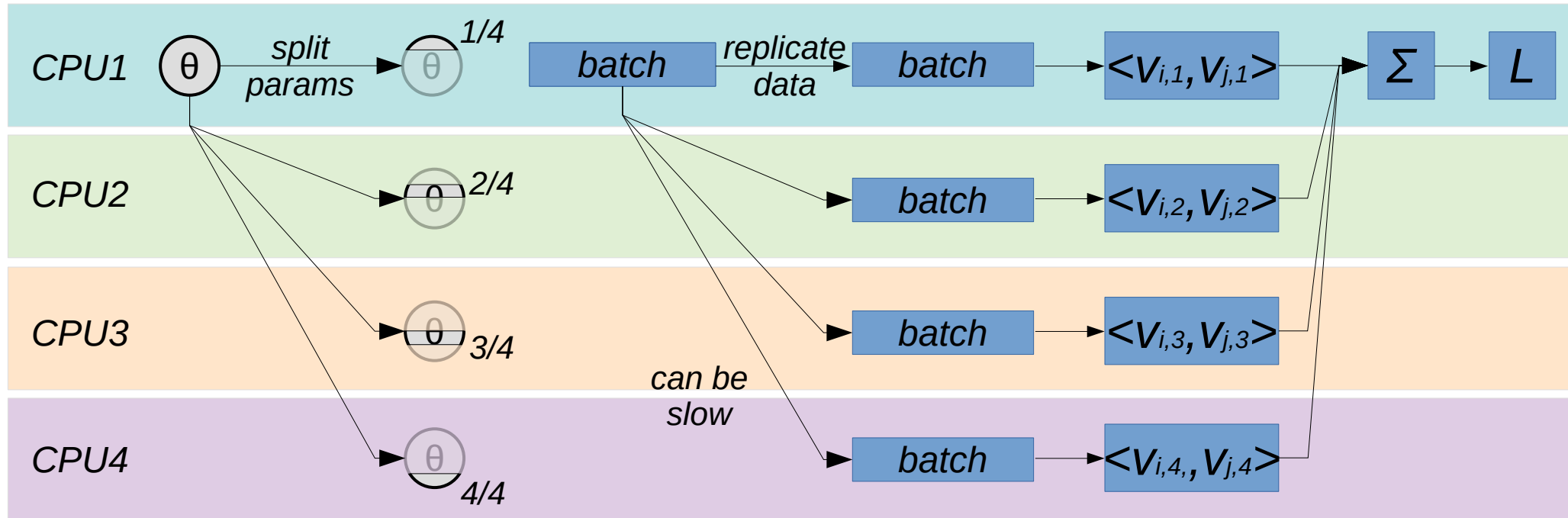*Each process runs **partial** model on **all** samples*

"Model parallelism"

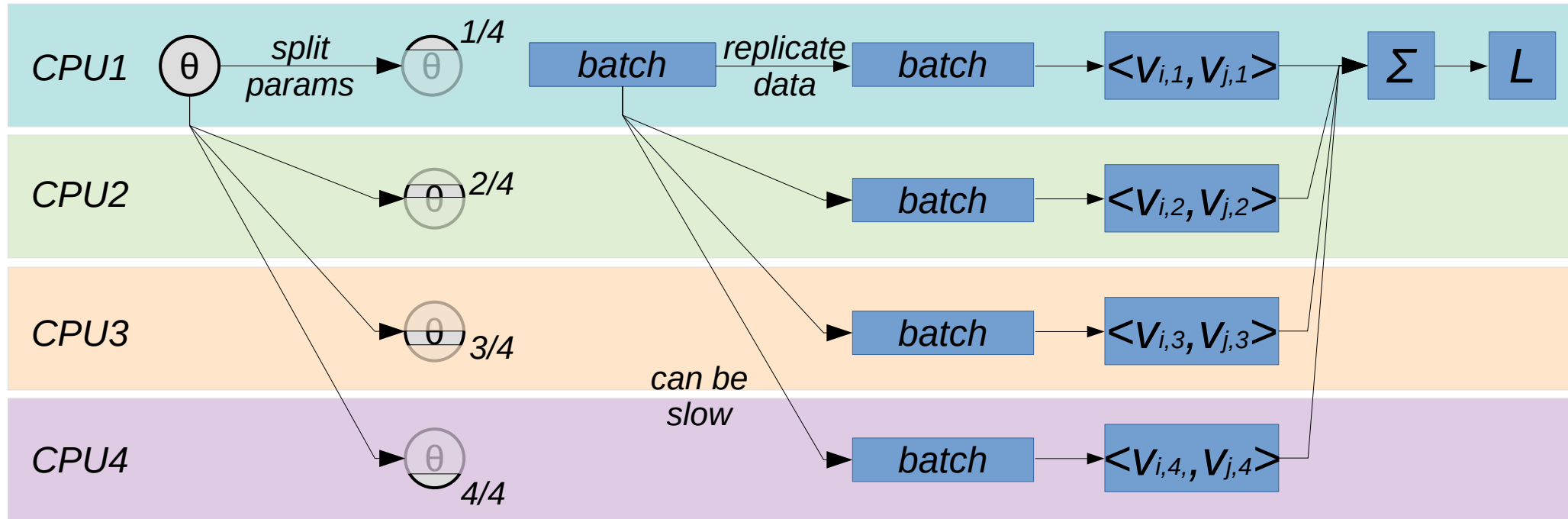*Each process runs **partial** model on **all** samples*

"Model parallelism"

Q: *do we have to send params from P1 each time?*

"Model parallelism"

Optimization: each process can update it's own params locally instead of receiving them from P1

Summary: operation parallelism

Data-parallel: *one process applies all model on **partial data***

Model-parallel: *one process applies **partial model** on all data*

Which one is better..
                    for word2vec?
                    In general?

Summary: operation parallelism

Data-parallel: *one process applies all model on **partial data***

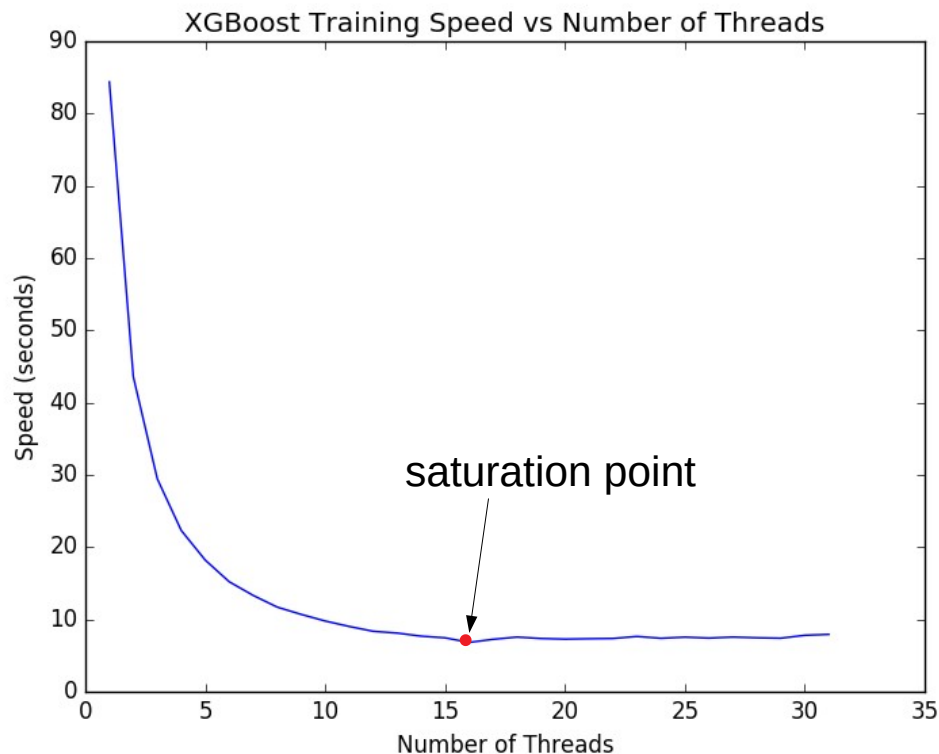*best for smaller model, more computations*

Model-parallel: *one process applies **partial model** on all data*

best for larger model, fewer computations

Which one is better..

for word2vec?

In general?

Summary: operation parallelism

Если time() < 19:00, тут можно поговорить про бустинг.

и-или сделать перерыв

Operation parallelism
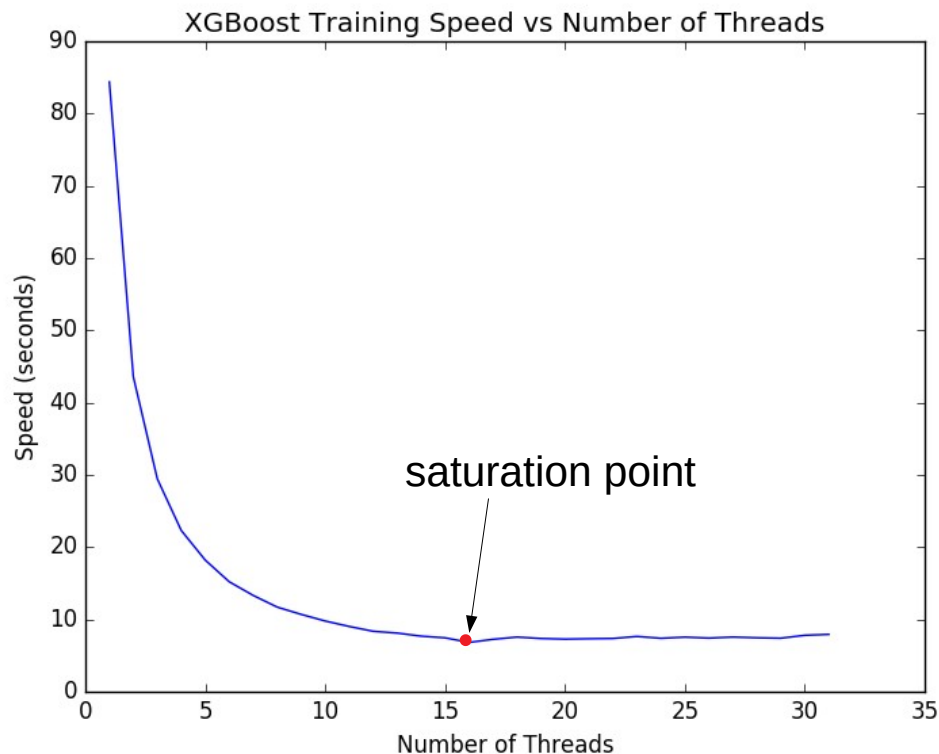

XGBoost Training Speed vs Number of Threads

More processes = more overhead

- *waiting for each other*
- *sending data over the network*
- *performance fluctuations*

Eventually adding more threads will no longer boost performance

Operation parallelism

XGBoost Training Speed vs Number of Threads

More processes = more overhead

- *waiting for each other*
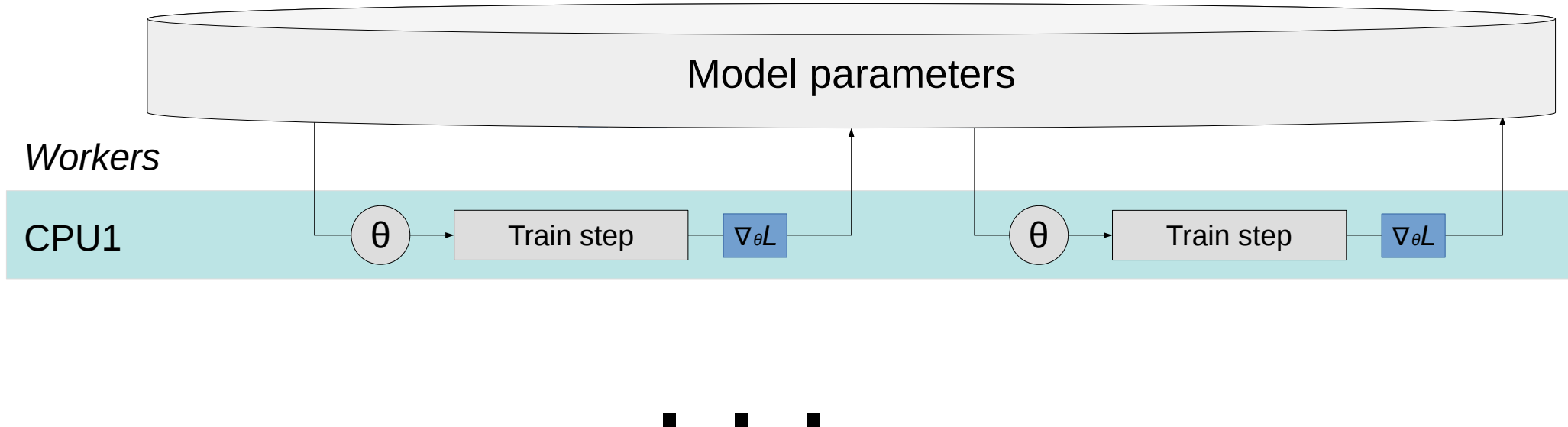- *sending data over the network*
- *performance fluctuations*

Eventually adding more threads will no longer boost performance

How do we push this point further?
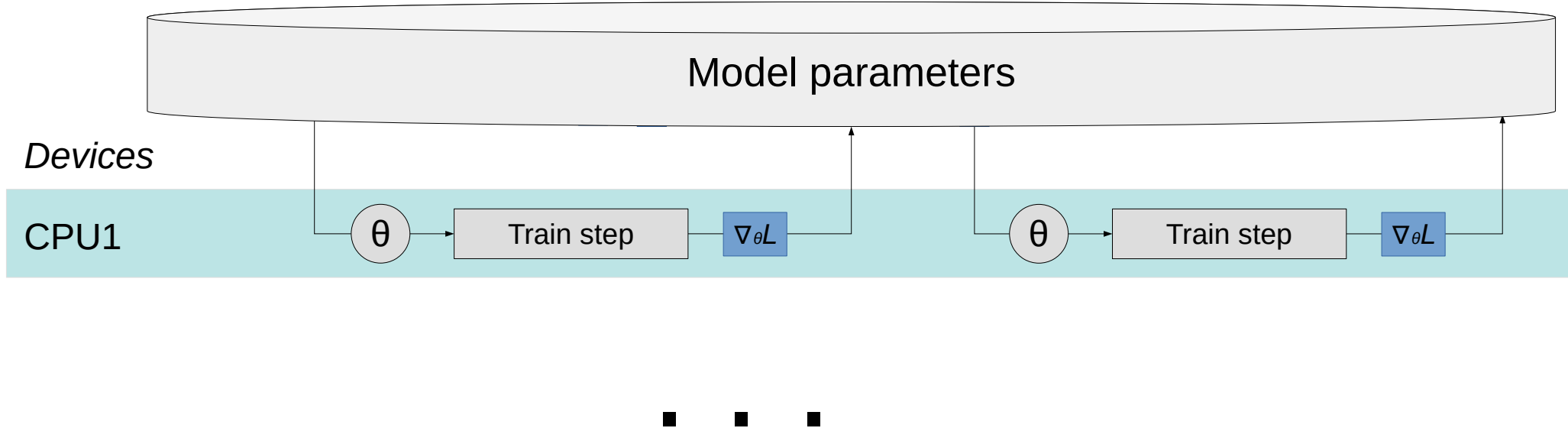
Parameter Server

Paper: Smola et al. (2010)

Make a dedicated process for parameters & optimizer
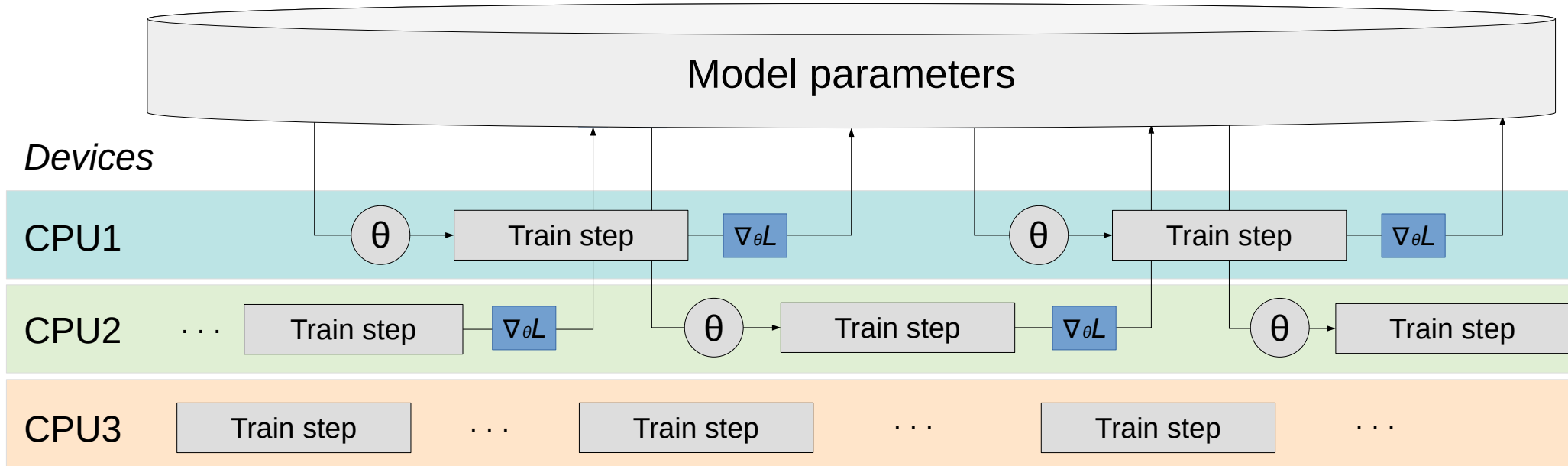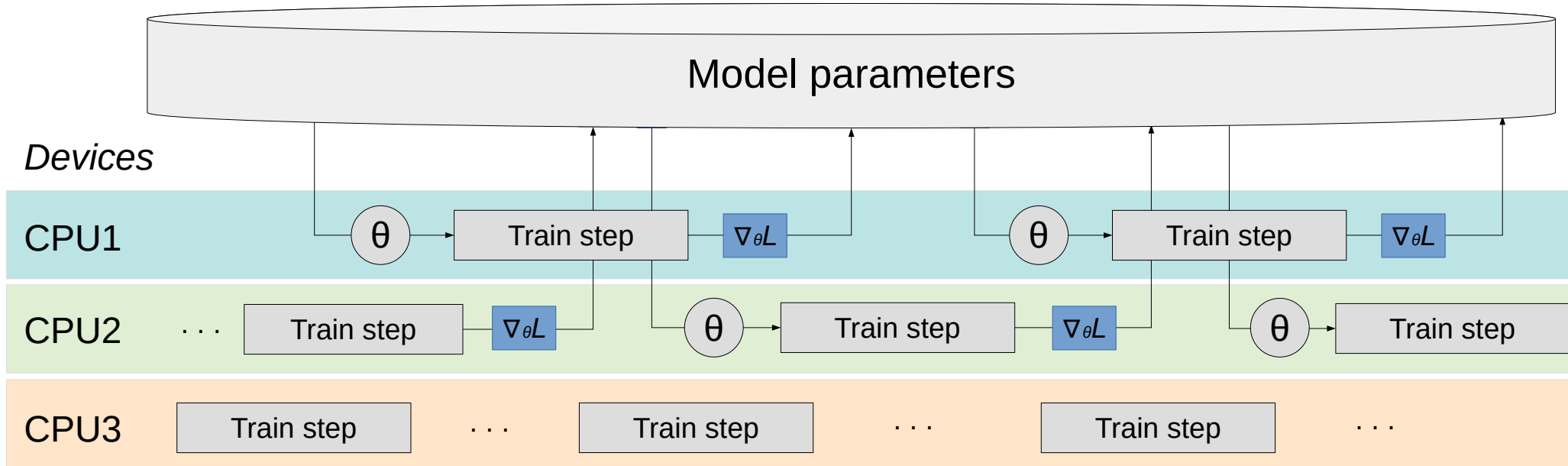
Asynchronous training

Idea: remove synchronization step alltogether, use parameter server

Asynchronous training

HOGWILD! arxiv.org/abs/1106.5730

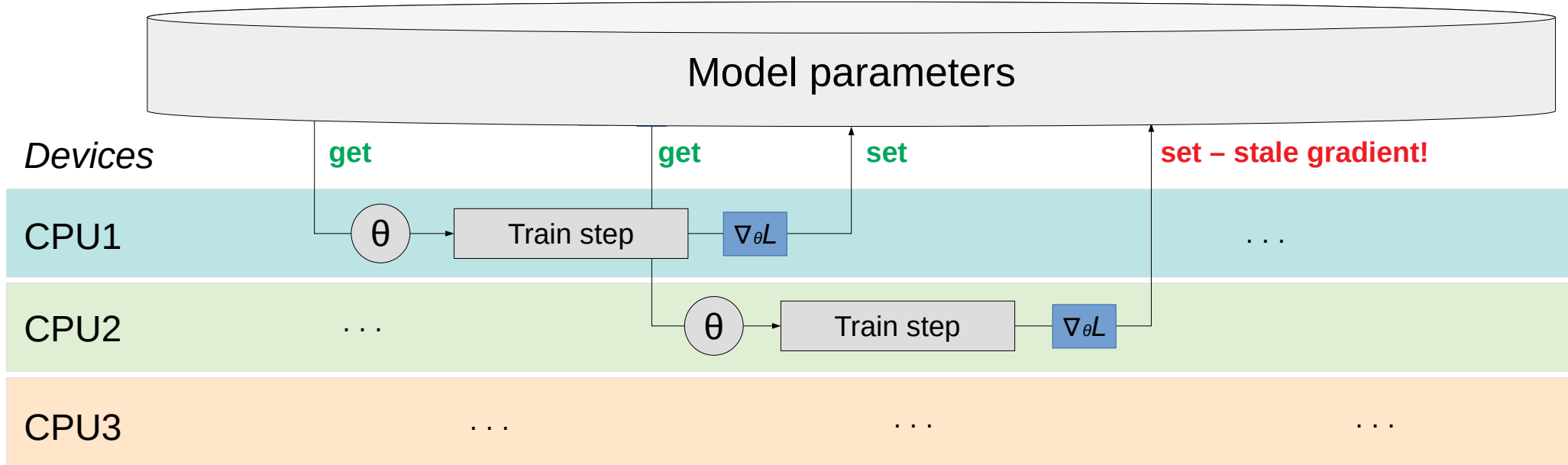**Idea:** remove synchronization step alltogether, use parameter server

Asynchronous training

HOGWILD! arxiv.org/abs/1106.5730

Idea: remove synchronization step alltogether, use parameter server

Q: have we lost anything by going asynchronous?

Asynchronous training

Idea: remove synchronization step alltogether, use parameter server

Staleness-aware SGD

Updates accumulated: $\quad c = \lfloor (\lambda/n) \rfloor$

Average gradient: $\quad g_i = \dfrac{1}{c} \sum_{l=1}^{c} \alpha(\tau_{i,l}) \Delta\theta_l, \; l \in \{1, 2, \ldots, \lambda\}$

New parameters: $\quad \theta_{i+1} = \theta_i - g_i,$

Staleness-aware SGD

Updates accumulated:

$$c = \lfloor (\lambda/n) \rfloor$$

n = total workers
λ = "accumulation factor"

Average gradient:

$$g_i = \frac{1}{c} \sum_{l=1}^{c} \alpha(\tau_{i,l}) \Delta \theta_l, \; l \in \{1, 2, \dots, \lambda\}$$

New parameters:

$$\theta_{i+1} = \theta_i - g_i,$$

Staleness-aware SGD

Updates accumulated:
$$c = \lfloor (\lambda/n) \rfloor$$

Average gradient:
$$g_i = \frac{1}{c} \sum_{l=1}^{c} \alpha(\tau_{i,l}) \Delta\theta_l, \; l \in \{1, 2, \ldots, \lambda\}$$

staleness-dependent "learning rate"

New parameters:
$$\theta_{i+1} = \theta_i - g_i,$$

Staleness-aware SGD

Updates accumulated:
$$c = \lfloor (\lambda/n) \rfloor$$

Average gradient:
$$g_i = \frac{1}{c} \sum_{l=1}^{c} \alpha(\tau_{i,l}) \Delta\theta_l, \; l \in \{1, 2, \ldots, \lambda\}$$

New parameters:
$$\theta_{i+1} = \theta_i - g_i,$$

$$\alpha_{i,l} = \frac{\alpha_0}{\tau_{i,l}}$$

**base learning rate**

staleness (≥1)
aka number of skipped updates

Parameter Server Applications

Conventional ML: *e.g. (Logistic Regression, CNN classifiers)*

Paper (sharded PS): https://www.cs.cmu.edu/~muli/file/ps.pdf
Another paper (optimizaton tricks): parameter_server_nips14.pdf
PyTorch tutorial (hogwild), TF tutorial (parameter server)

Reinforcement learning:

A3C

IMPALA - Multiple Learners
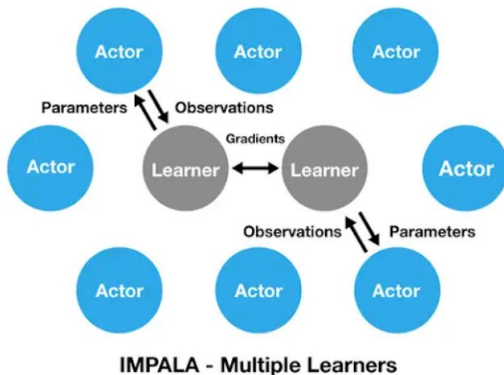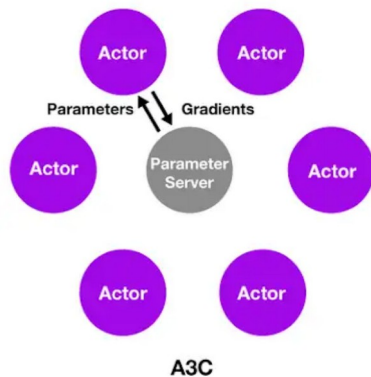
Async. RL: arxiv.org/abs/1602.01783
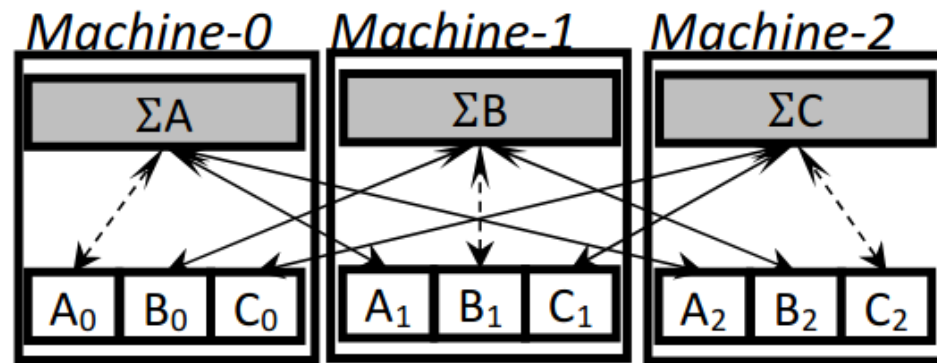IMPALA: arxiv.org/abs/1802.01561
SEED RL: arxiv.org/abs/1910.06591

More:
(english) https://youtu.be/kOy49NqZeqI
(russian) https://youtu.be/wswbMkT55mI

Modern Parameter Server Systems

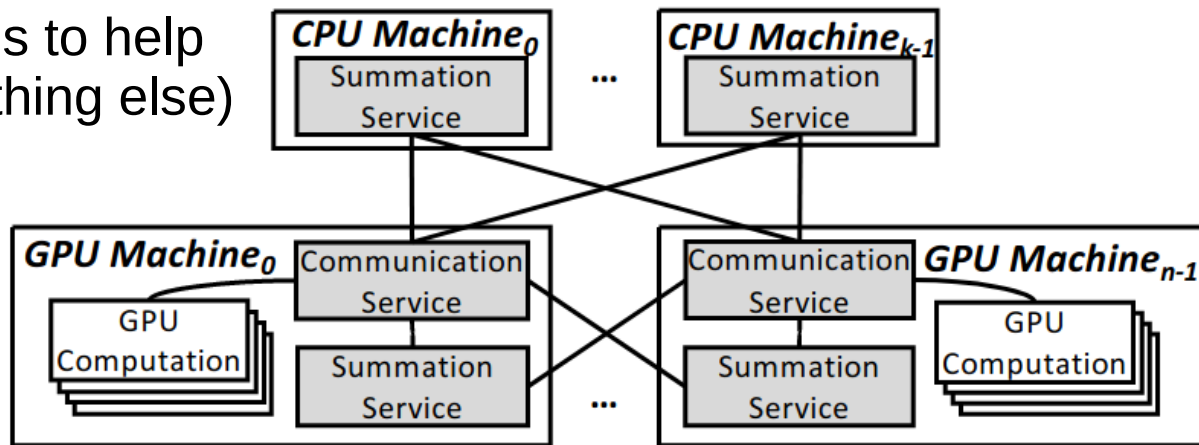Read more: https://www.usenix.org/system/files/osdi20-jiang.pdf

Baseline: sharded PS on every node
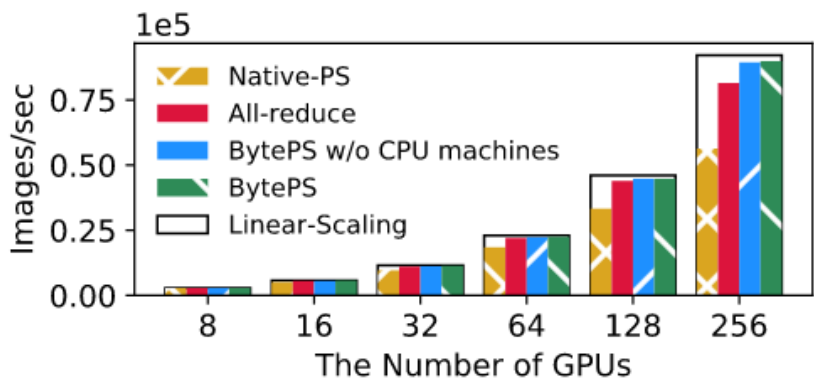Each server runs a subset of weights

BytePS: add non-GPU nodes to help
averaging gradients (and nothing else)

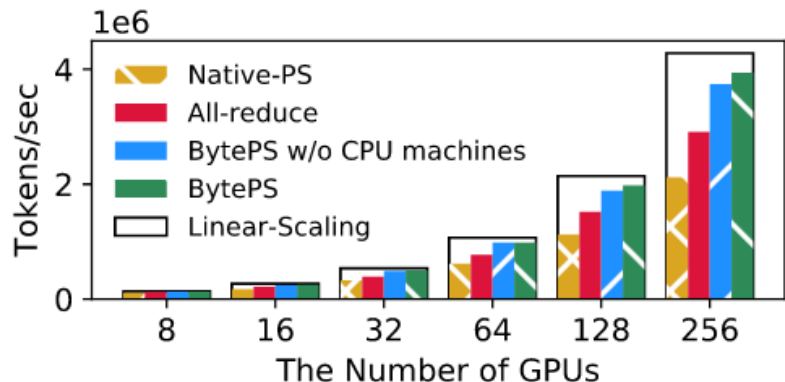CPU nodes are ~10x
cheaper to deploy / rent
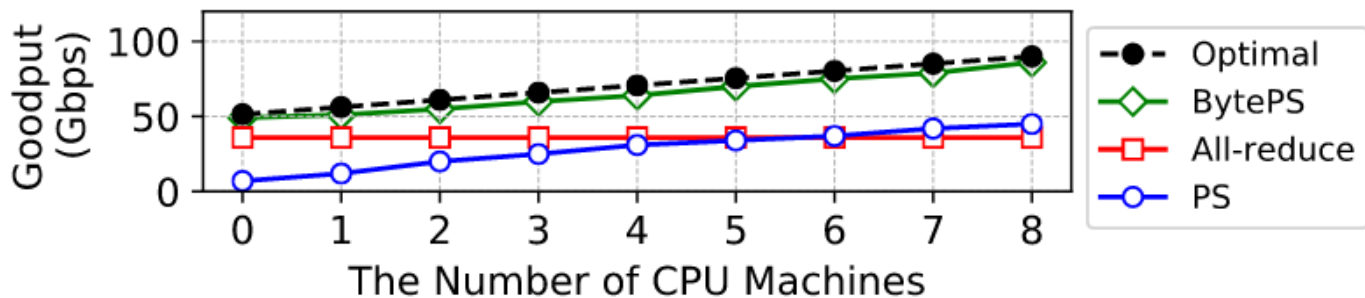
Modern Parameter Server Systems

Read more: https://www.usenix.org/system/files/osdi20-jiang.pdf

(a) TensorFlow, ResNet-50, batch=256 images

(b) MXNet, BERT-Large, batch=8192 tokens

The effect of auxiliary CPU nodes