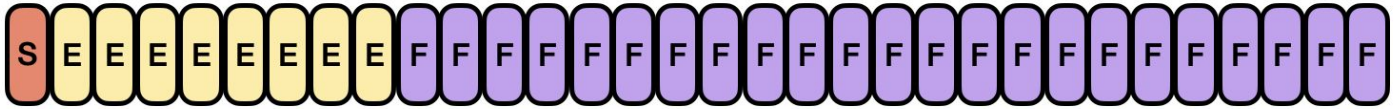
LSDL Seminar 03
Efficient training and inference

Ildus Sadrtdinov, 30.09.2024

Floating point numbers

float 32



Sign (1 bit)

Exponent (8 bits)

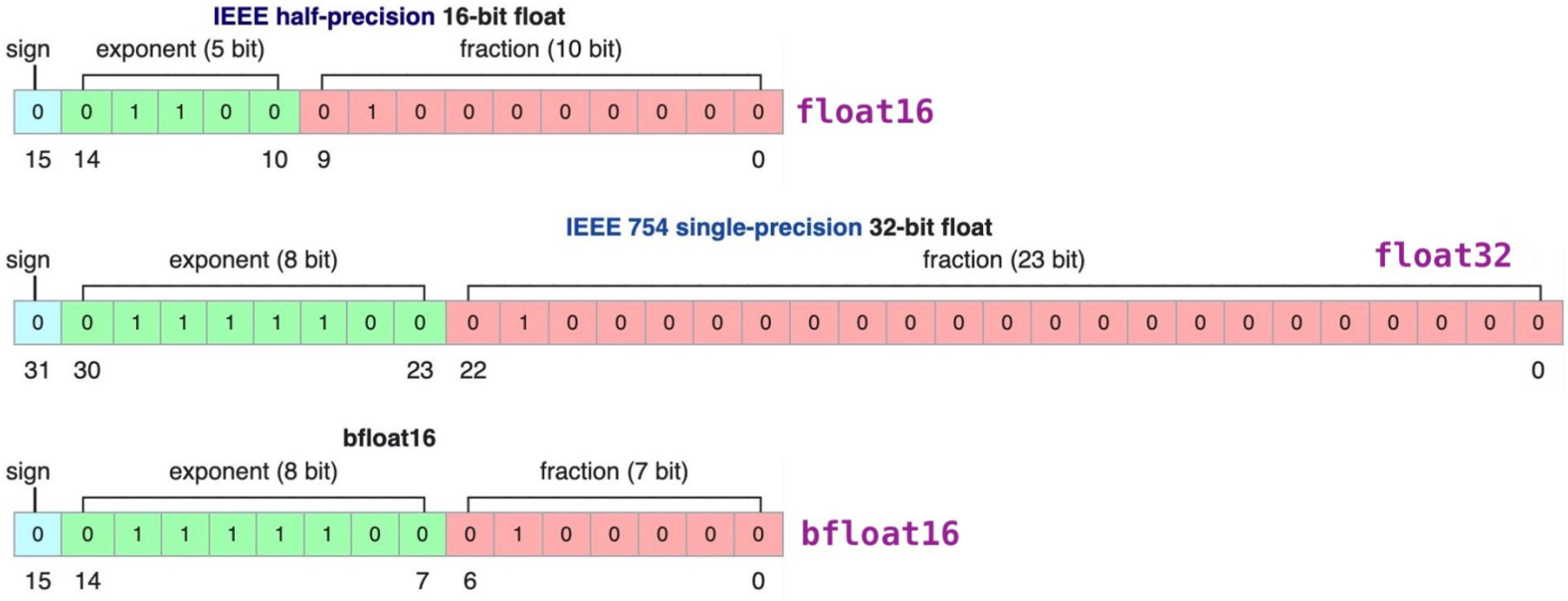
Fraction (23 bits)

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2 \,,$$

which yields

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) .$$

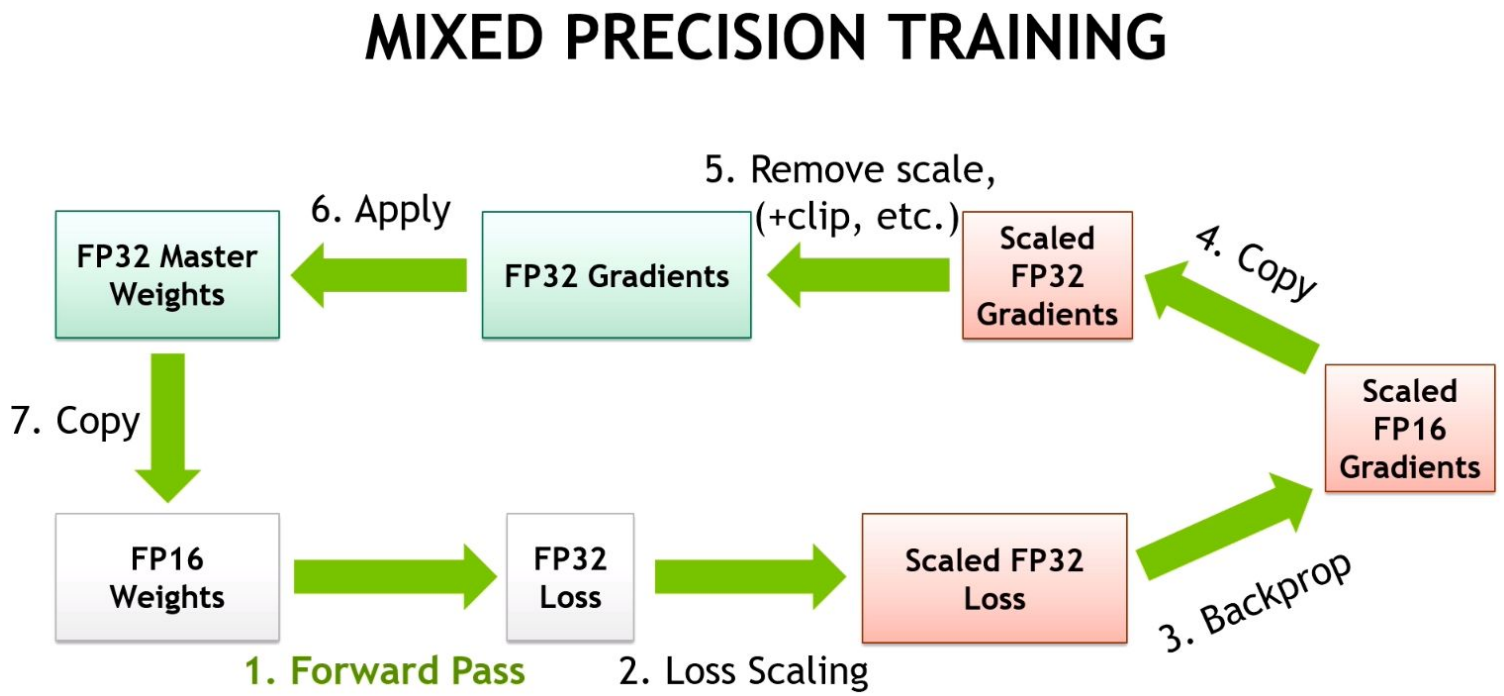
Floating point formats



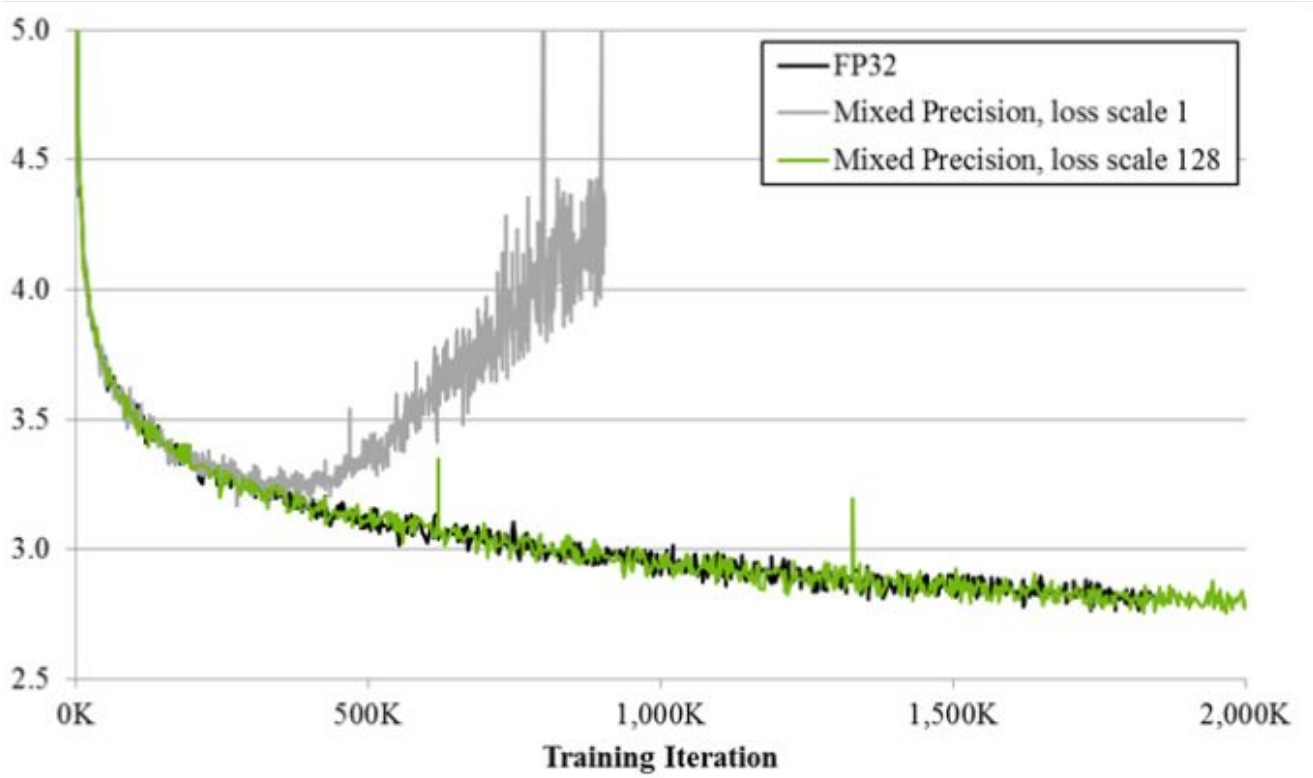
Automatic Mixed Precision (AMP)

- Training in **FP16** diverges
- Some operations are feasible with **FP16** (matrix multiplication), while other require **FP32** (softmax, normalization)
- Data cast gives almost no overhead
- We can use **FP16** where appropriate to speed up forward and backward passes
- Loss scaling is required

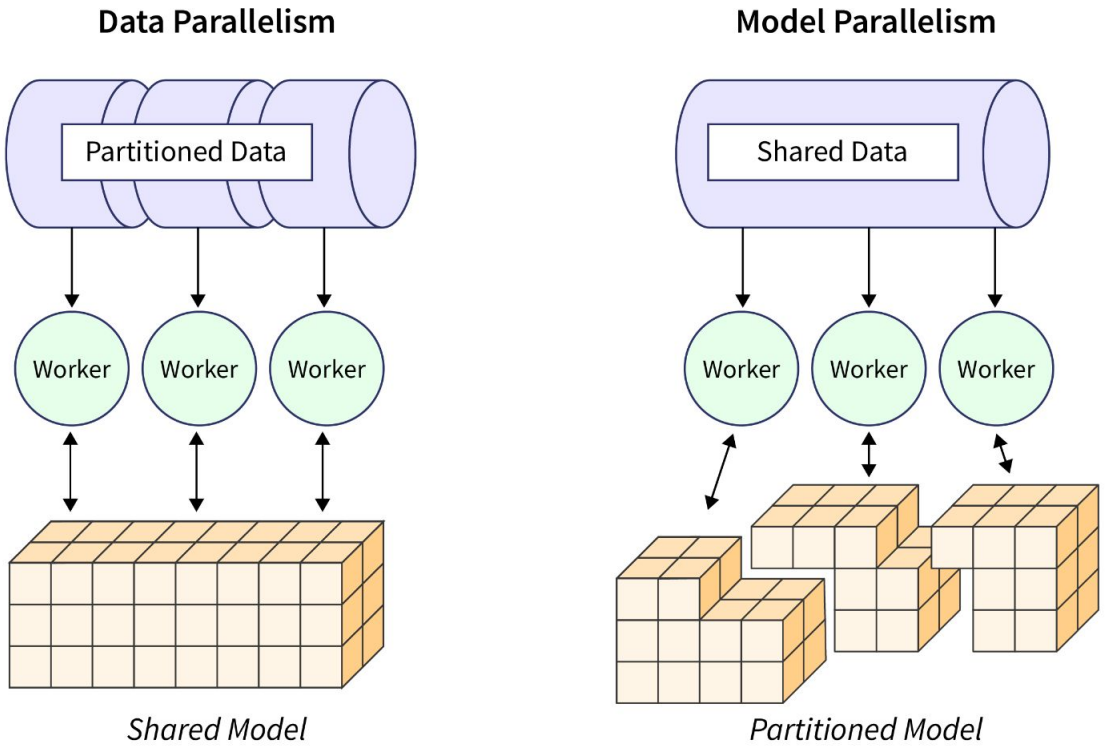
Automatic Mixed Precision (AMP)



Automatic Mixed Precision

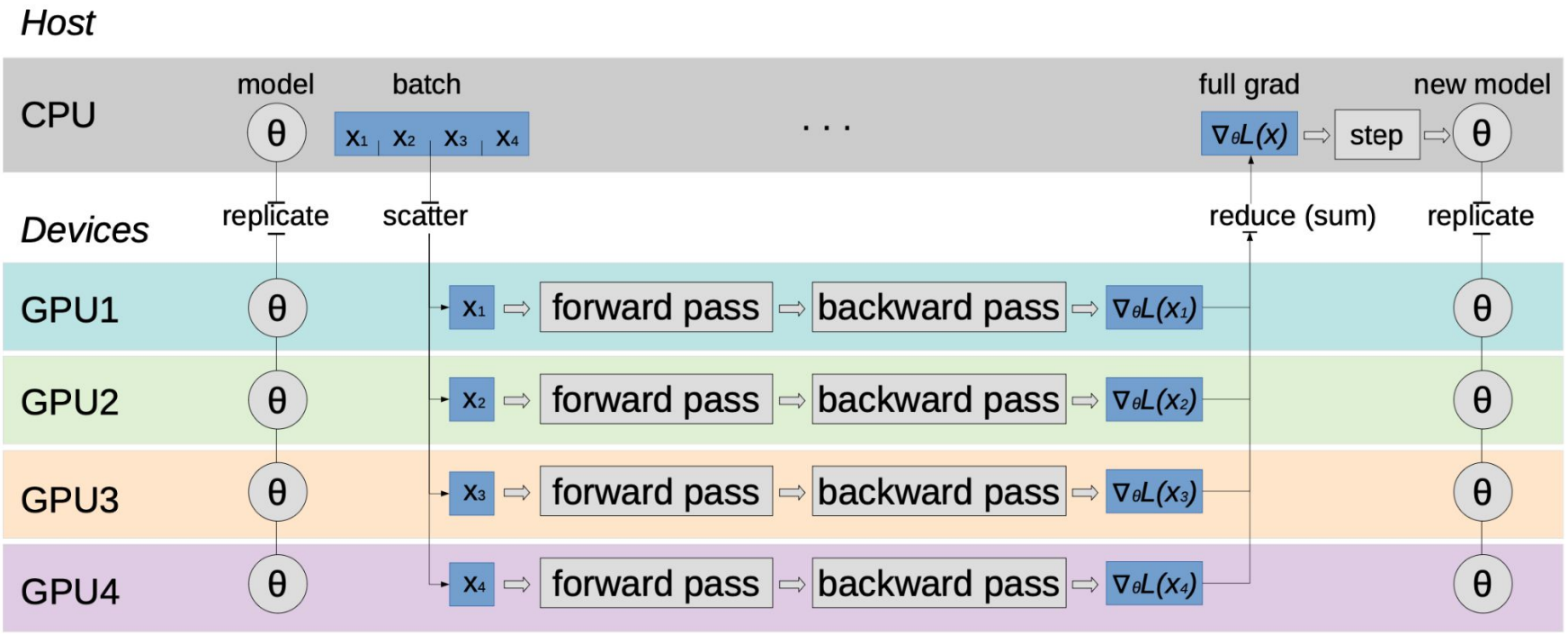


Two paradigms of parallelism

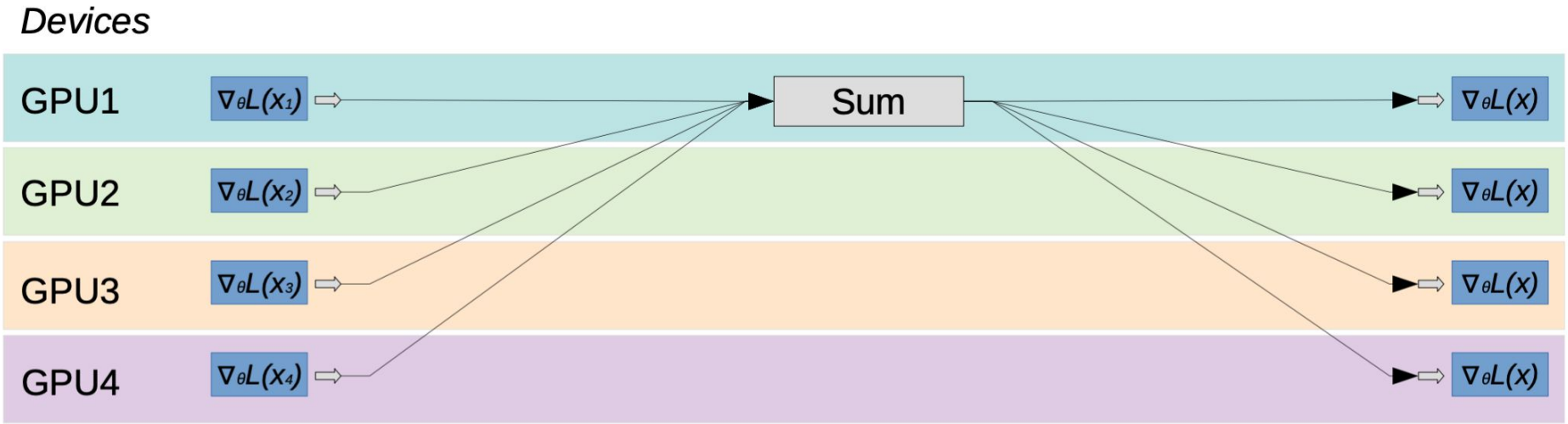


Data Parallelism

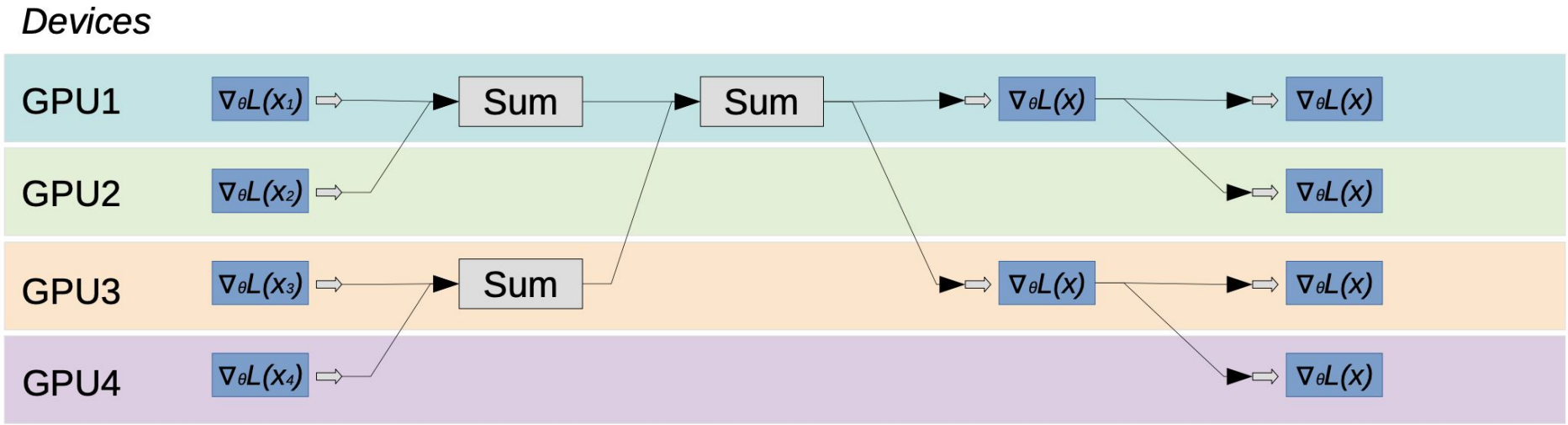
https://github.com/mryab/efficient-dl-systems/tree/2023/week05_data_parallel



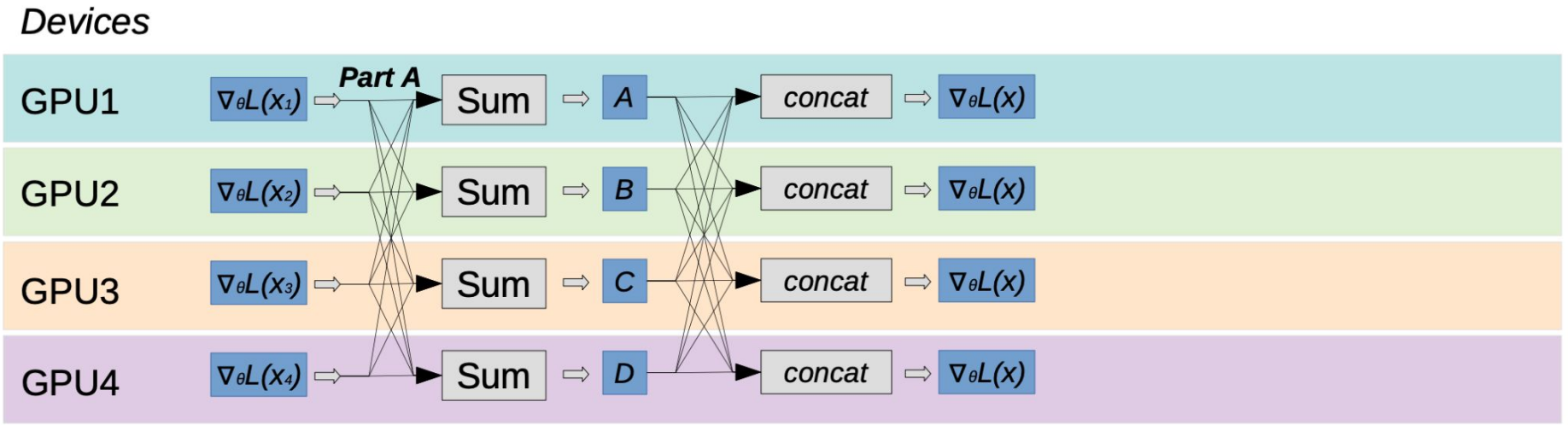
Naive All-Reduce



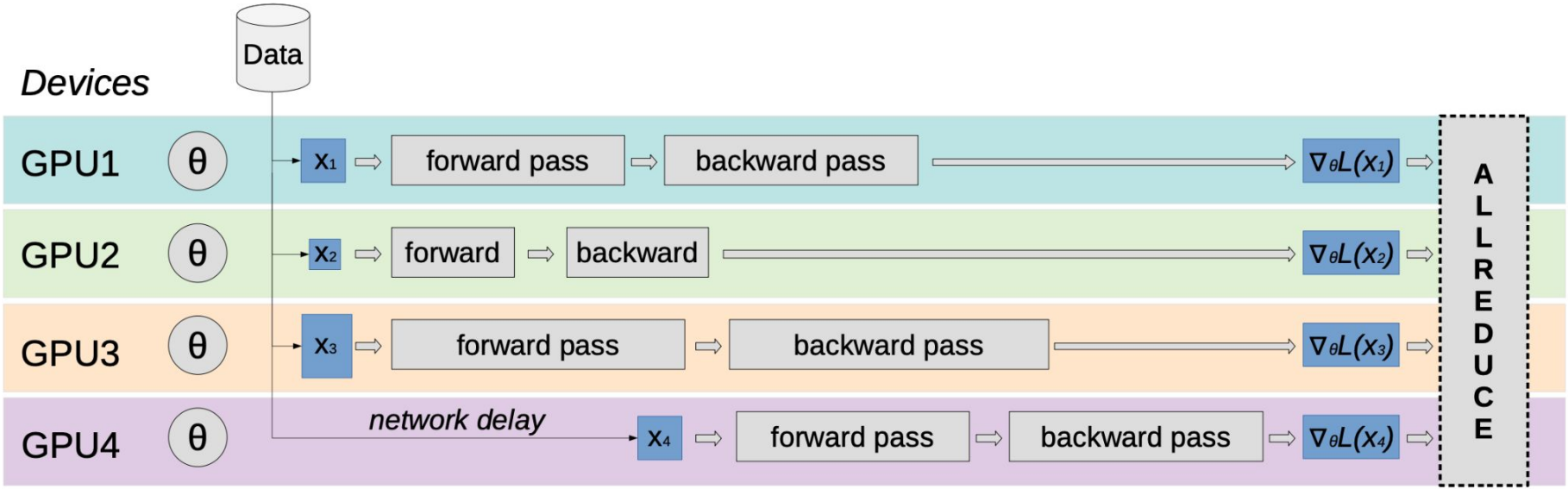
Tree All-Reduce



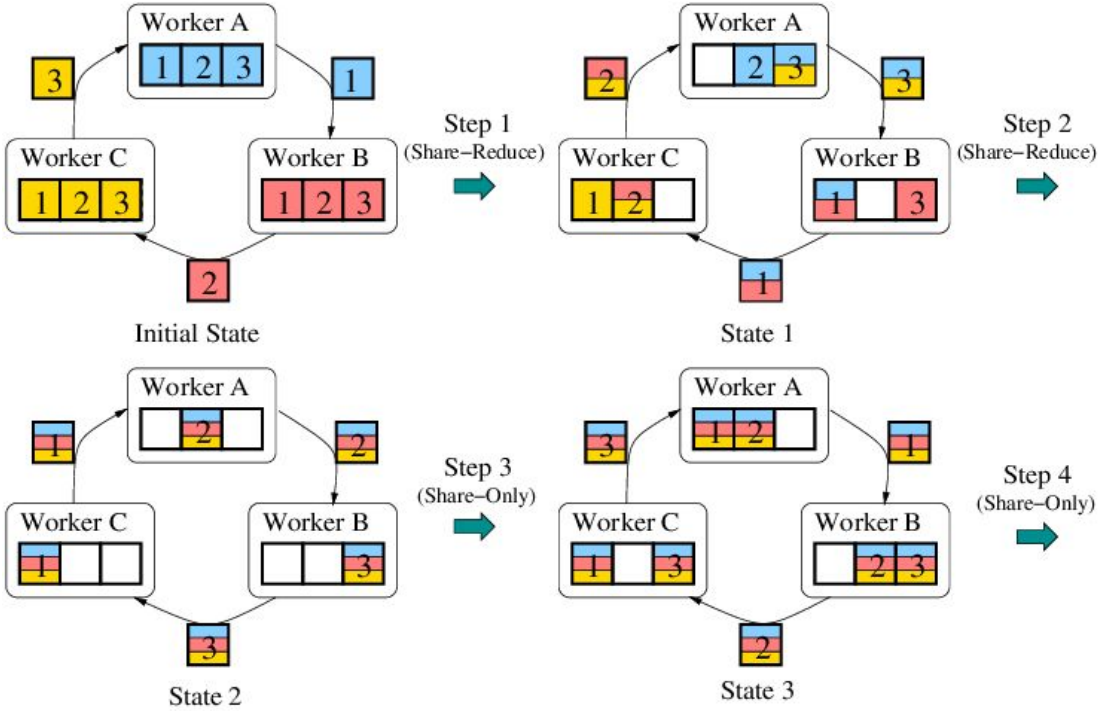
Butterfly All-Reduce



What happens in practice?



Ring All-Reduce



n Gradient subvector n of Worker A (n = 1, 2, 3)

n Gradient subvector n of Worker B (n = 1, 2, 3)

n Gradient Subvector n of Worker C (n = 1, 2, 3)

1 Gradient subvector 1 summation for Workers A and B

2 Gradient subvector 2 summation for Workers B and C

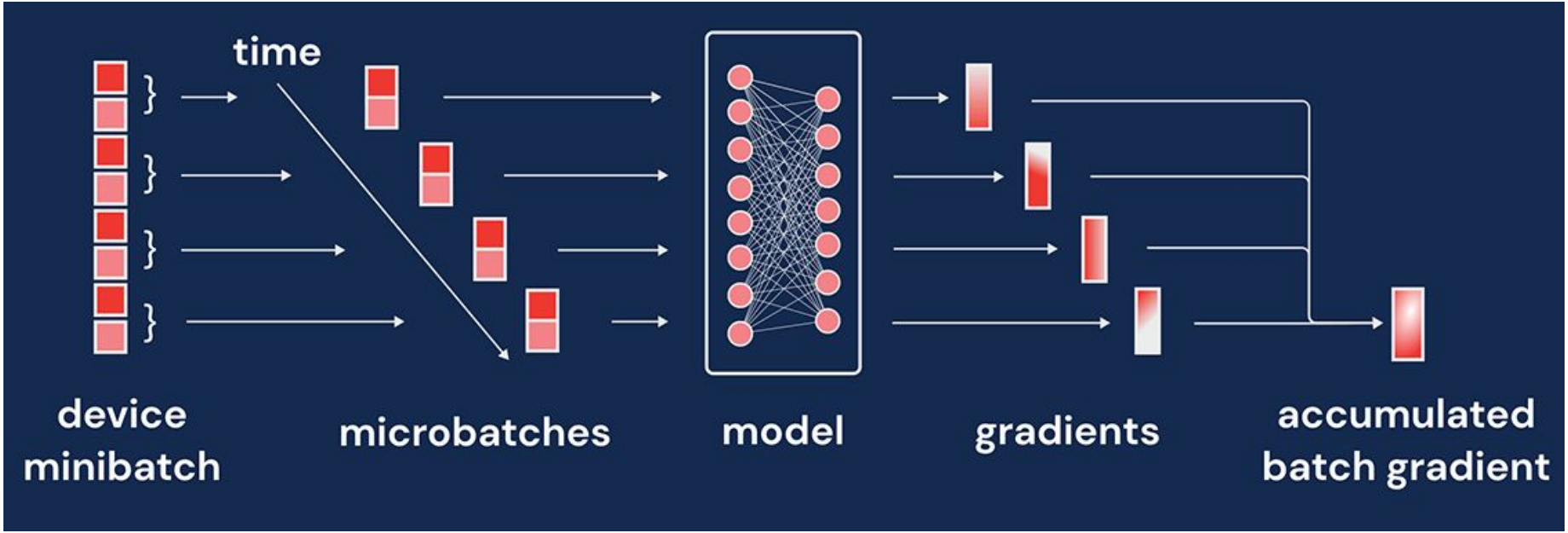
3 Gradient subvector 3 summation for Workers A and C

n Gradient subvector n summation for Workers A, B, and C (n = 1, 2, 3)

Sum of full gradient vectors of Workers A, B, and C

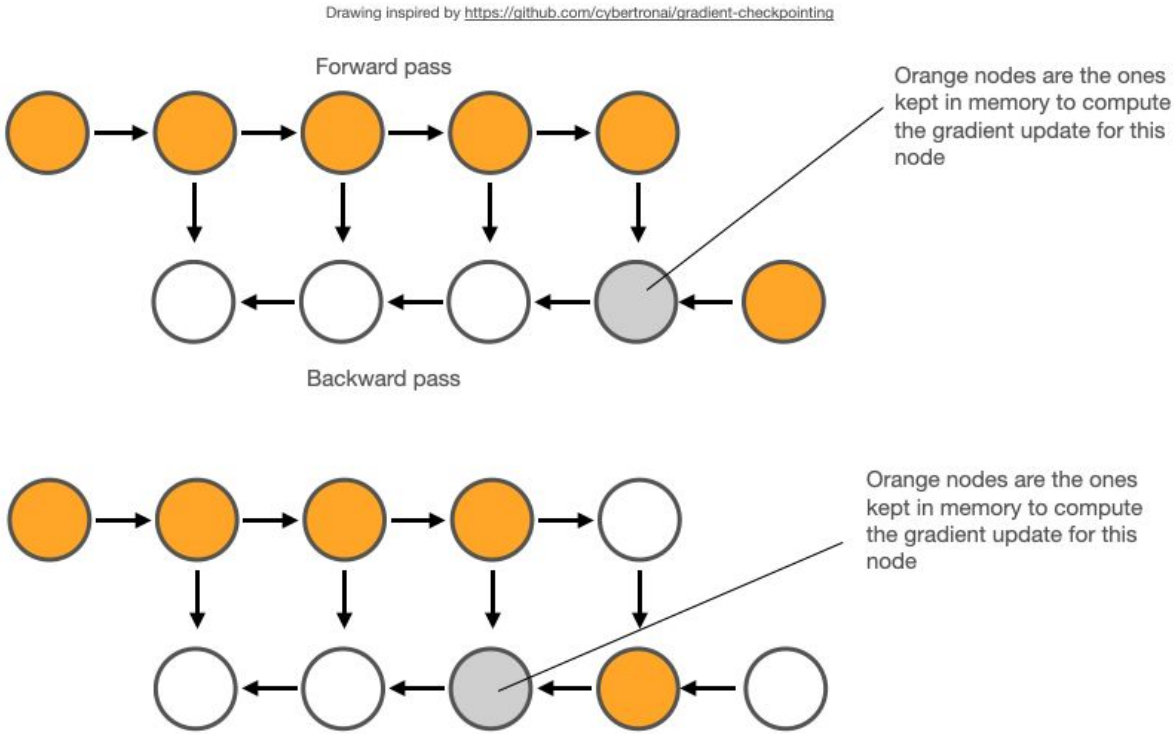
Gradient accumulation

1. **Desired batch size**
   does not fit in memory



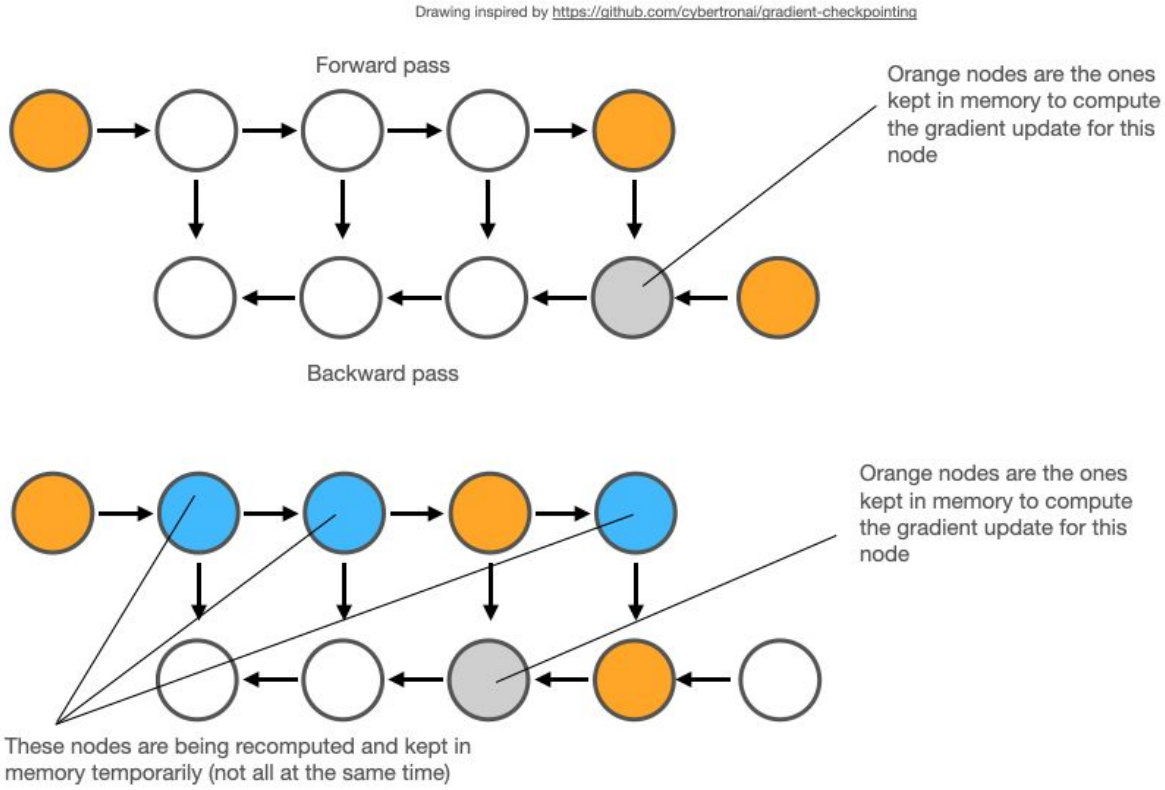
Regular backpropagation

2. **Single object** does not fit in memory



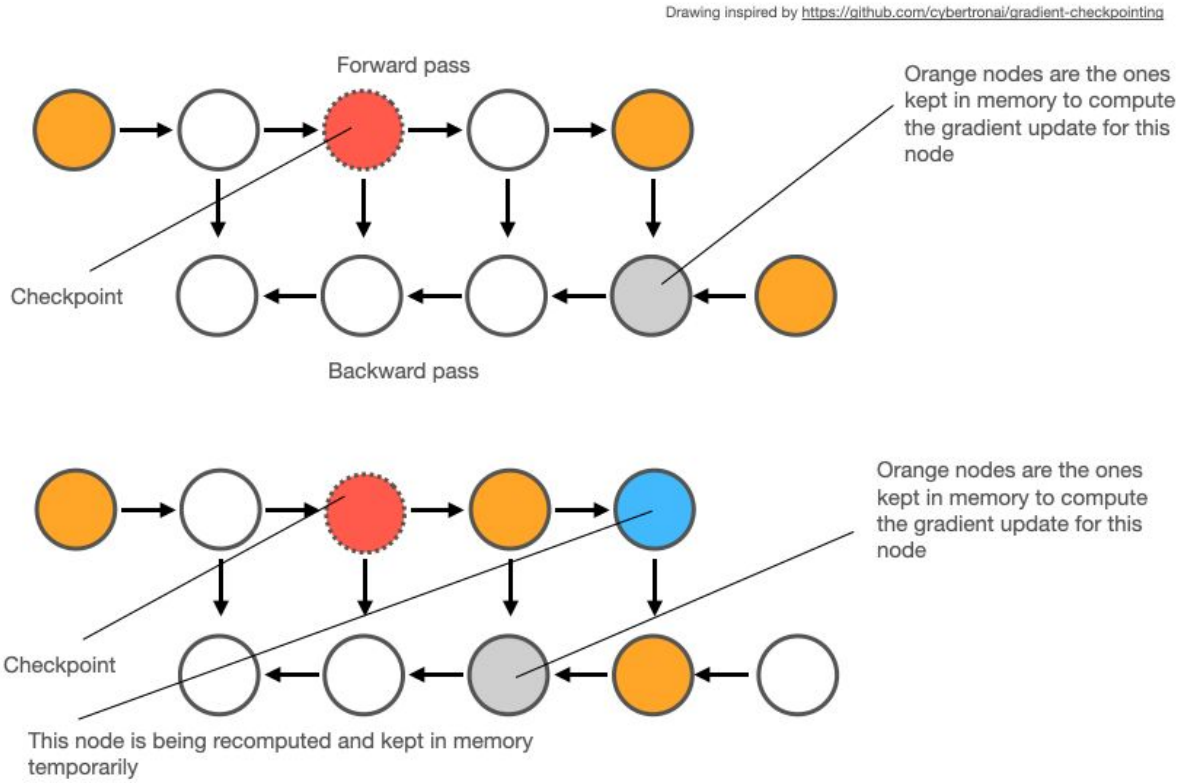
Low-memory backpropagation

2. **Single object** does not fit in memory



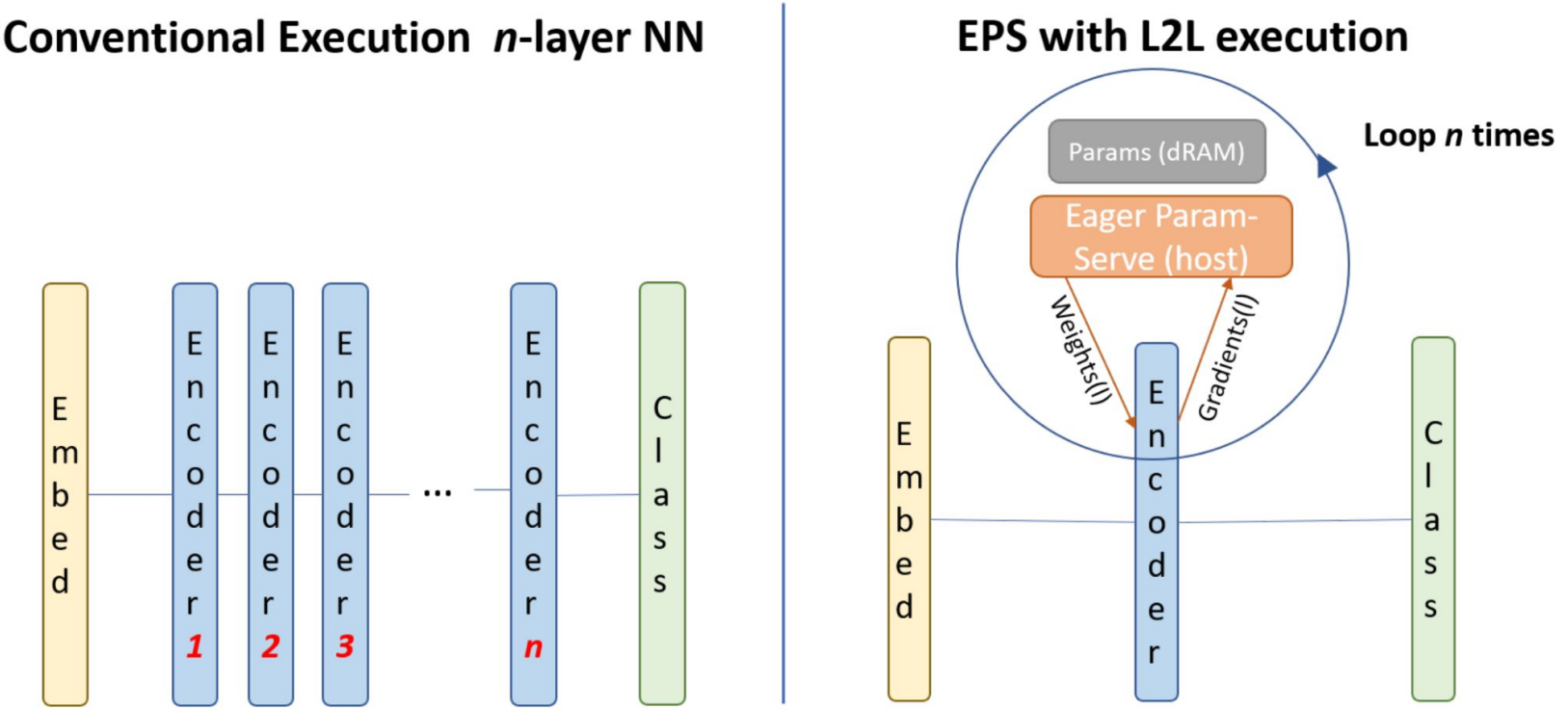
Gradient checkpointing

2. **Single object** does not fit in memory



Memory offloading

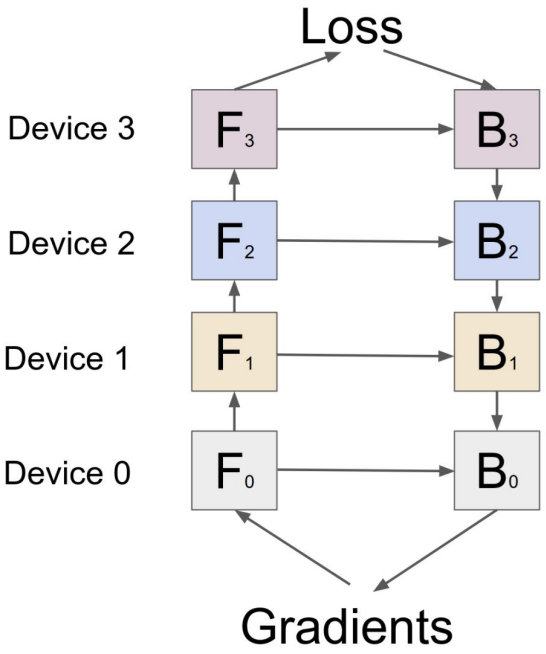
3. **Model parameters**
do not fit in memory

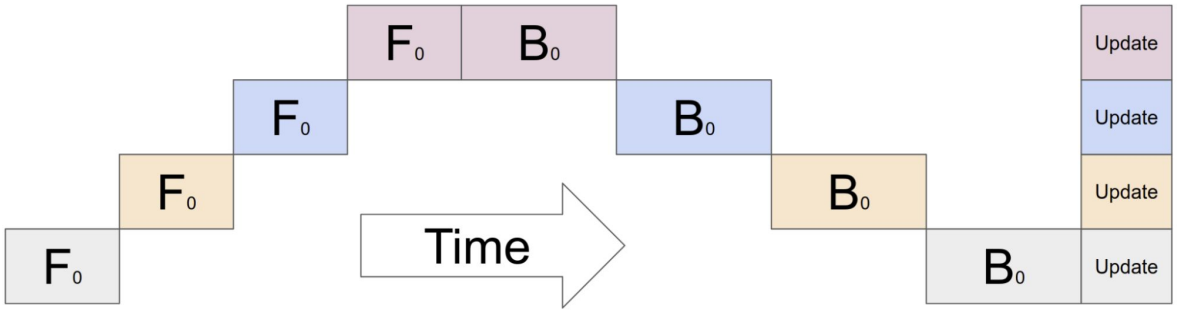


https://arxiv.org/abs/2002.05645

Model-parallel training

3. **Model parameters**
do not fit in memory

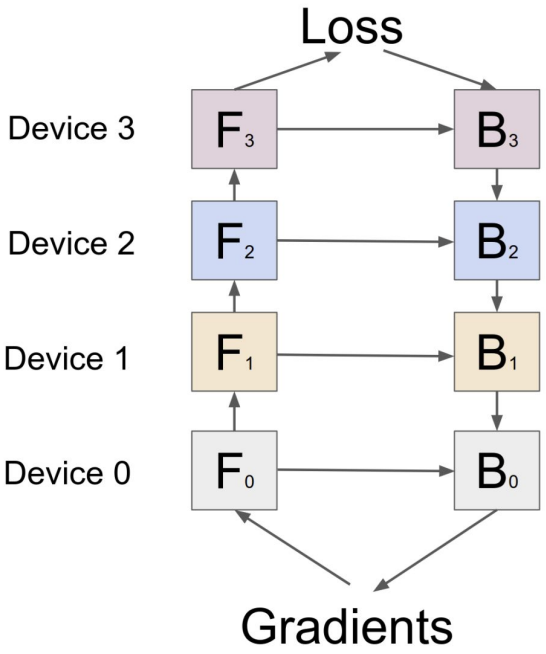


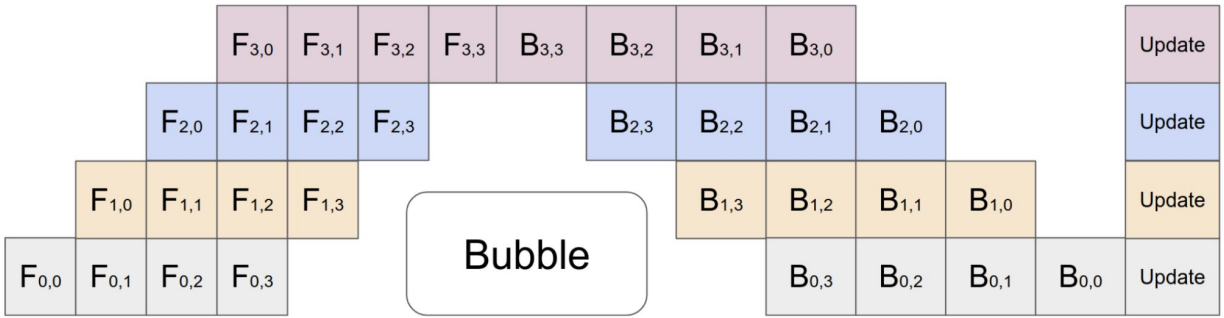


https://arxiv.org/abs/1811.06965

Pipeline Parallelism

3. **Model parameters**
do not fit in memory





https://arxiv.org/abs/1811.06965

Conclusion

- **AMP** accelerates training with a memory overhead
- **Data-parallel:** default choice if your model fits into 1 GPU
- **Model-parallel:** depends on model architecture but sometimes can be more effective