https://hao-ai-lab.github.io/dsc291-s24/

DSC 291: ML Systems Spring 2024

LLMs

Parallelization

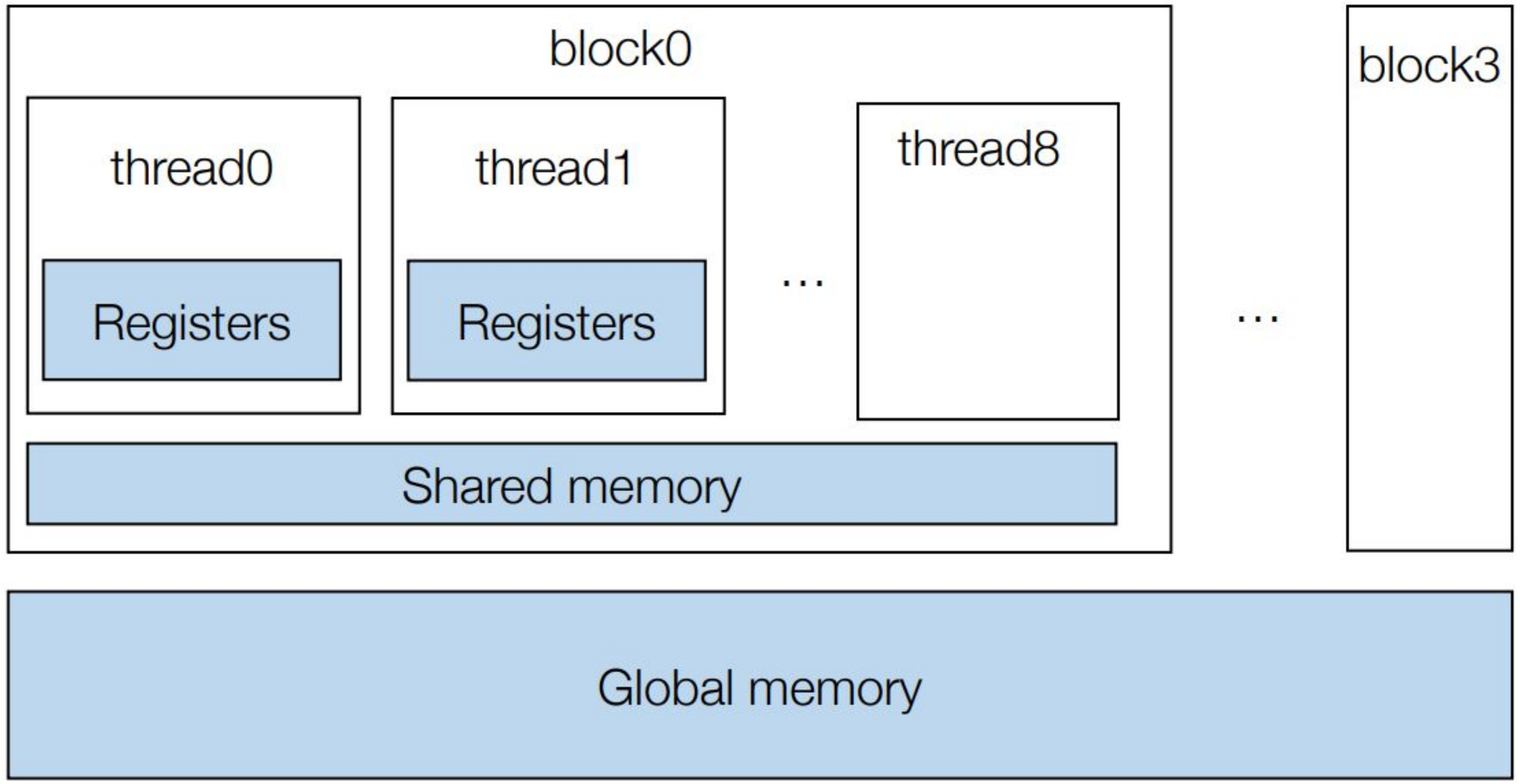
Single-device Optimization

Basics

Memory Optimization

- Checkpointing and rematerialization
- CPU Swapping
- Quantization and Mixed precision

Recap: Memory Hierarchy



Shared memory: 64 KB per core

GPU memory(Global memory):

RTX3080 10GB
RTX3090 24GB
A100 40/80 GB

Our Goal

- Fit the workload on limited memory and ensure

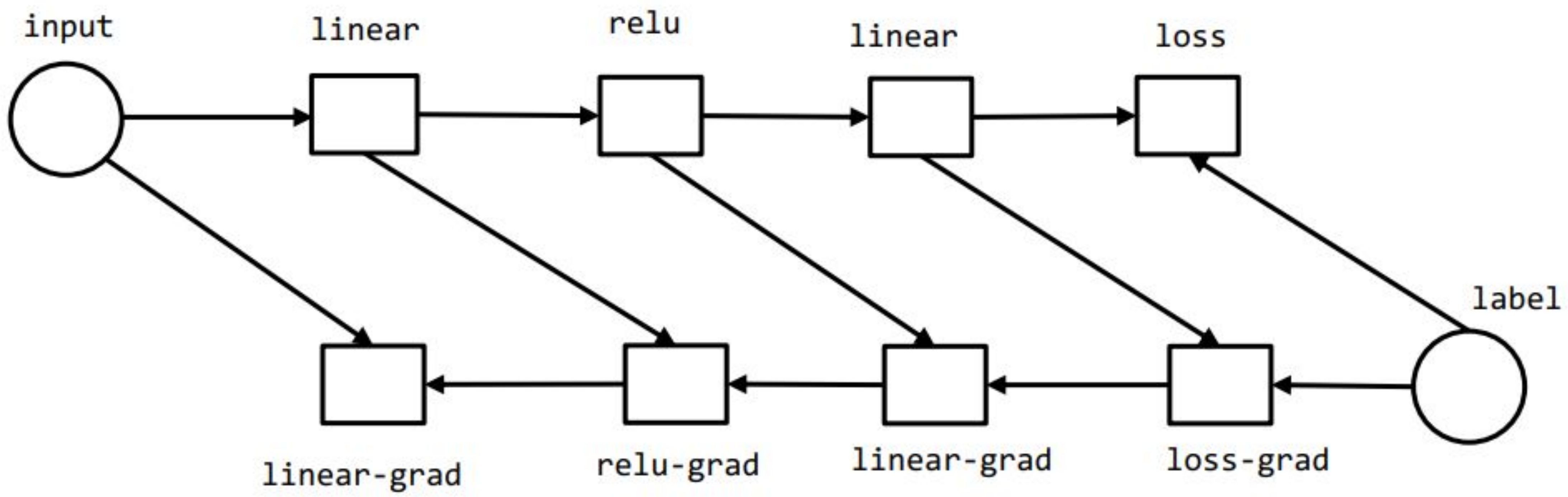
  peak memory < available memory

  Note:

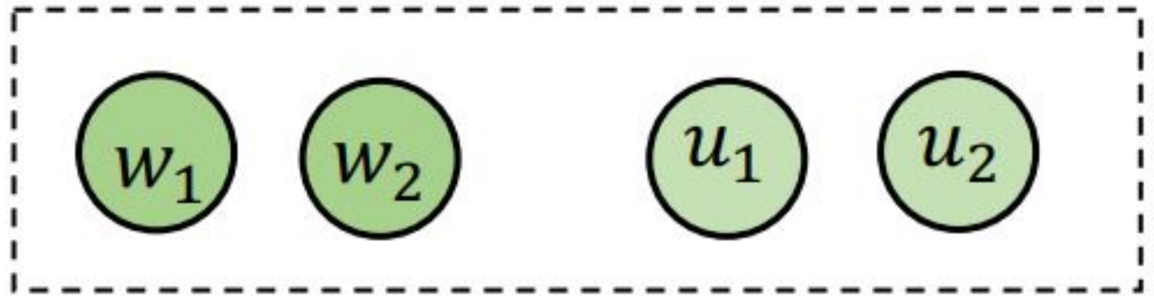
  - We are not min (memory)
  - We are not min(max(memory))
  - We just need max(memory) < available memory
    - Unless otherwise specificized

Source of Memory Consumption

A simplified view of a typical computational graph for training, weights are omitted and implied in the grad steps.



Optimizer states



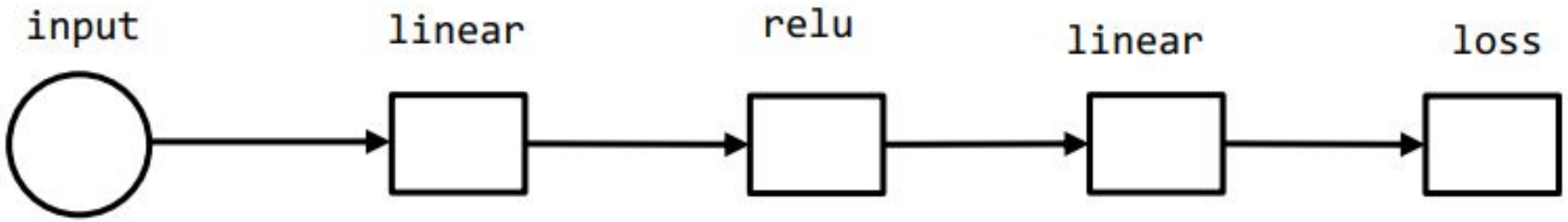
Sources of memory consumption

- Model weights
- Optimizer states
- Intermediate activation values

How to estimate memory of model weights

- Lifetime:
  - When will this memory be needed
- Size
  - How large the memory is

At Inference: Lifetime?

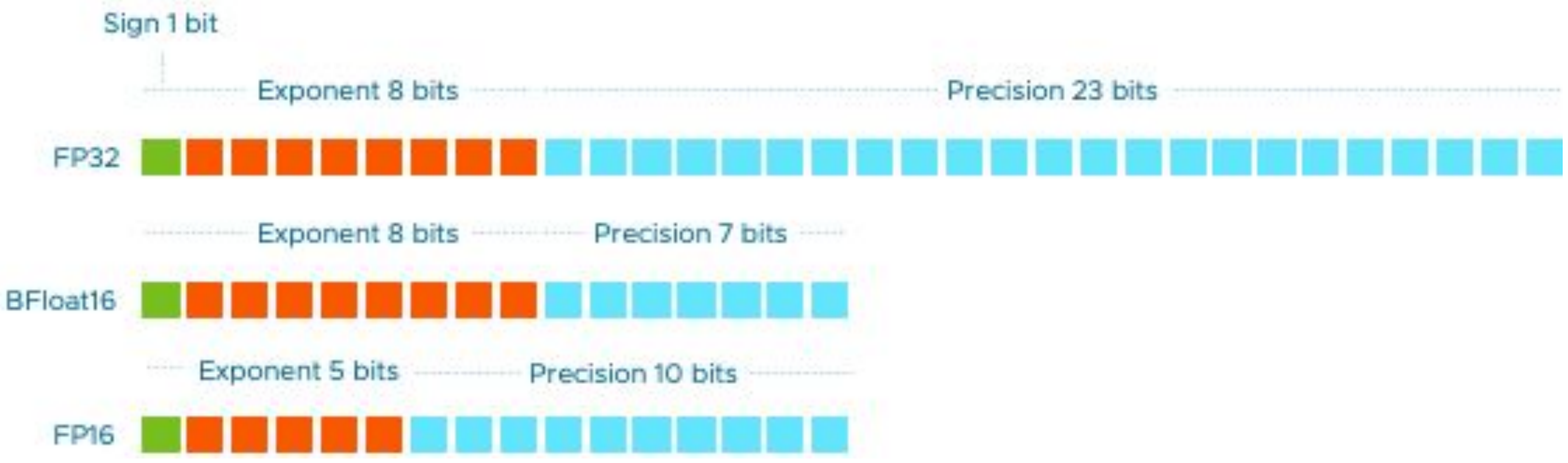


We only need O(1) memory for computing the final output of a N layer deep network by cycling through two buffers

Lifetime of

- weights
- activations?
- Optimizer states?

Estimate size: Popular float standards



- What does exponent and fraction control in float point representation?
- What's the difference between bf16 and fp16?

Estimate the weight size: GPT-3 as an example

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

- Model weights: 175B, each param = 16 / 32 bits = 2 / 4 bytes
  - 175B * 2 / 4 = 350G / 700G
  - Rule of thumb: check precision, and N * 2 or N * 4

Estimate the activation size

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

- Conv2d activation:
  - bs * nc * wi * hi -> bs * nc * wo * ho
- Transformers activation:
  - bs * seqlen * d_model: 3.2M * 12288 = 39.321B = 78 / 156 G

Estimate the Optimizer State Size?

- Adam Optimizer: What is the memory added?

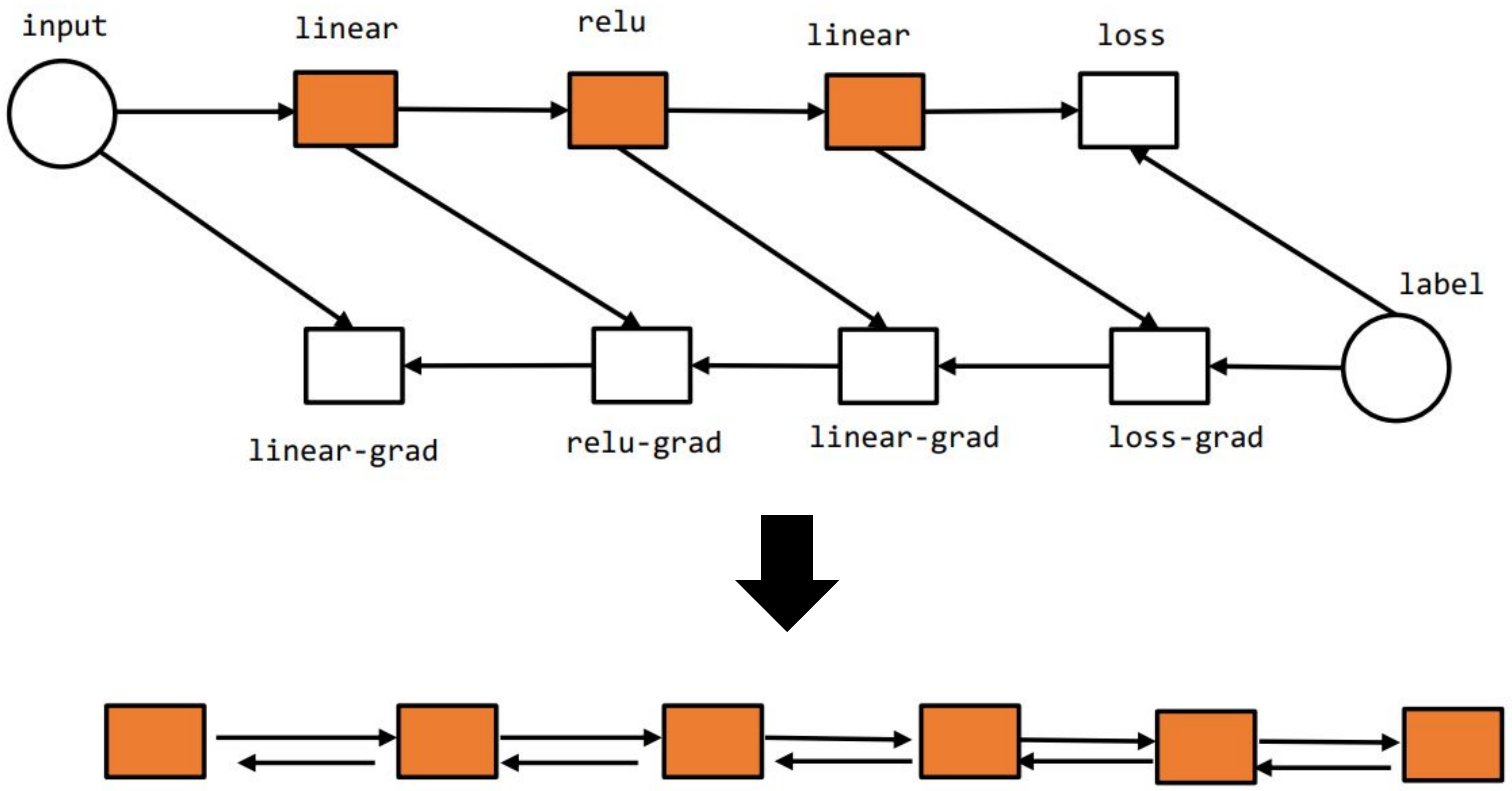
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Optimizer state: first moment estimate (mean)

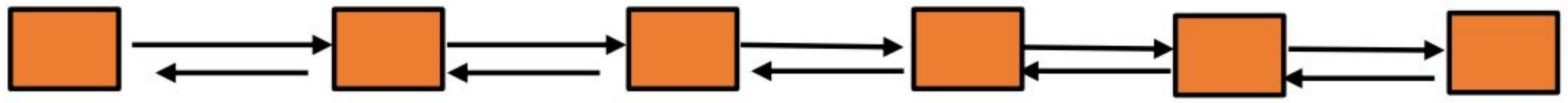
Optimizer state: second moment estimate (variance)

Put into practice



- Because the need to keep intermediate value around for the gradient steps. Training a N-layer neural network would require O(N).

Memory Overview

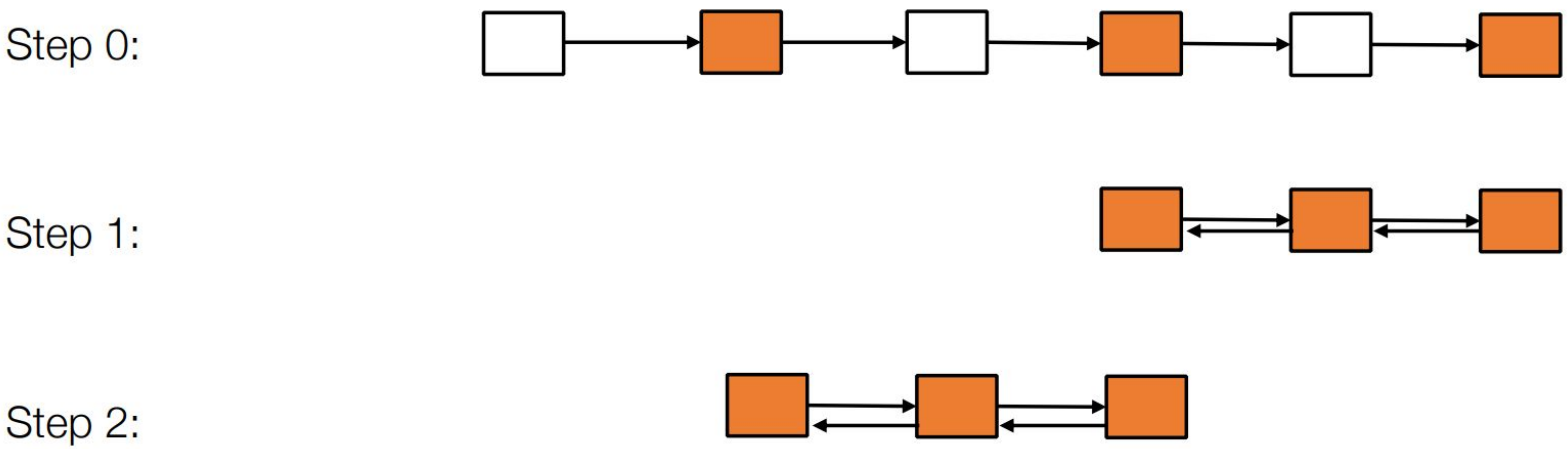


- Parameters: 175B * (fp32) = 350 / 700 G
- Activations:
  - At the transformer boundary: (N = 96) * 78 / 156 G = 7488 / 14976 G
  - This is not accurate because transformers is a composite layers.
  - A lot more than this: roughly 5 x (7488 / 14976).
  - Optimizer states: (precision: fp32) * 2 * 175B = (8) * 175 G

Reduce memory

- Single Device trick (today)
- Parallelization (next week)

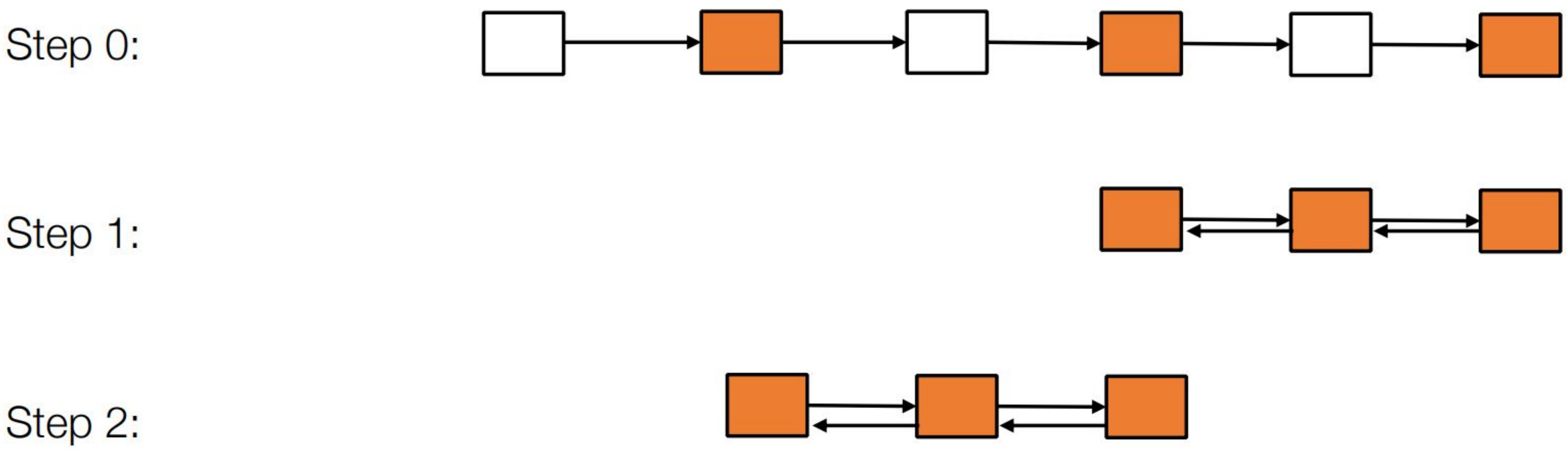
Reduce activation memory



Observation

- The activation is not needed again until the backward pass comes
- Discard some of them and recompute the missing intermediate nodes in small segments

Reduce activation memory

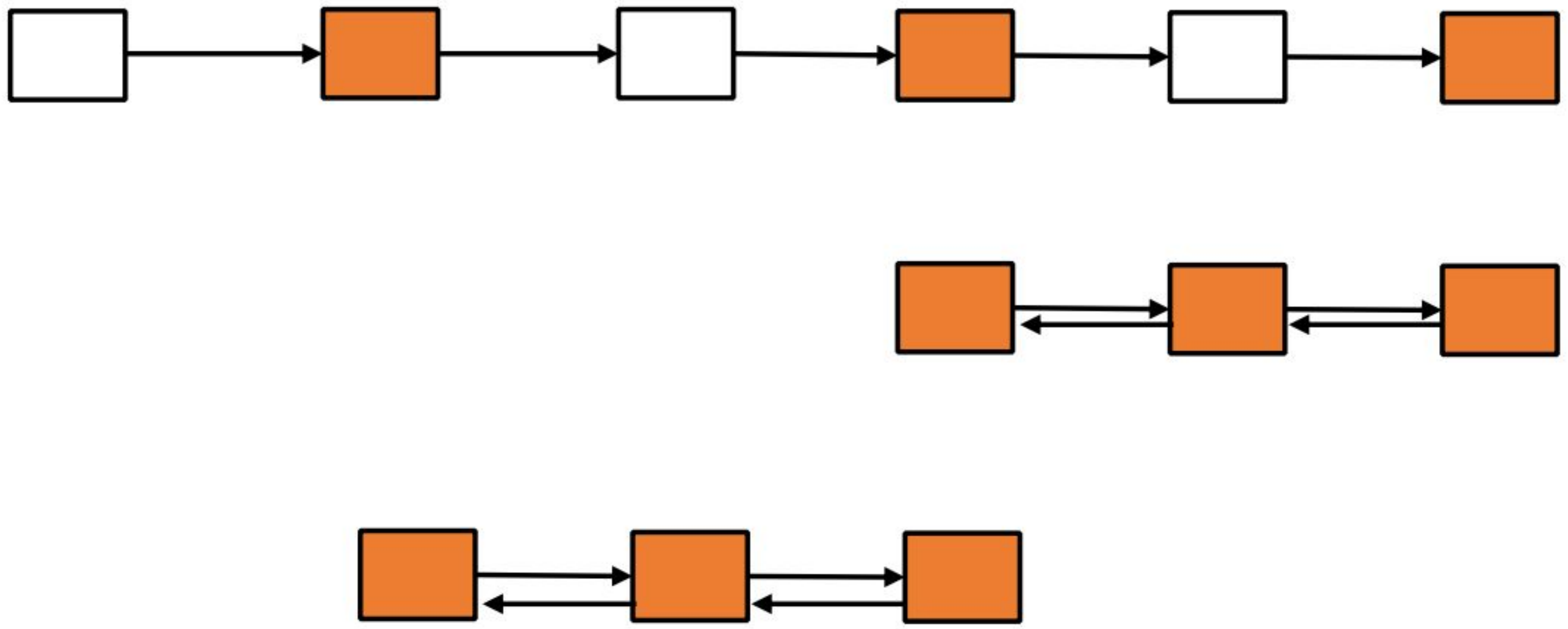


- Extreme case: discord nothing
  - Memory ++, compute --
- Extreme case: discard all and recompute for each layer
  - Memory --, compute++
- We want to strike a balance?

Reduce activation memory

Forward computation

Gradient per segment with re-computation



For a N layer neural network, if we checkpoint every K layers

$$Memory\ cost = O\left(\frac{N}{K}\right) + O(K)$$

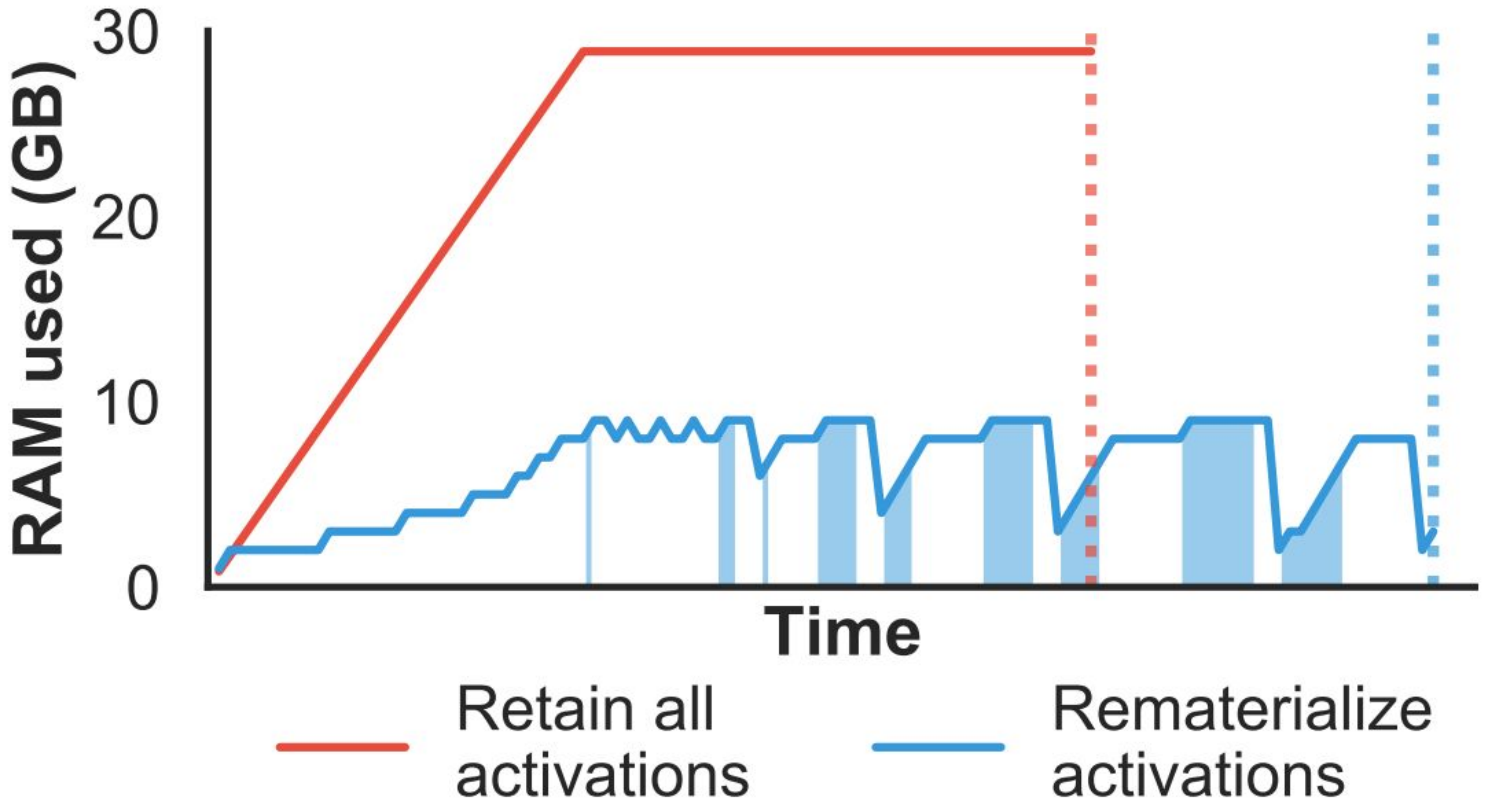
Checkpoint cost

Re-computation cost

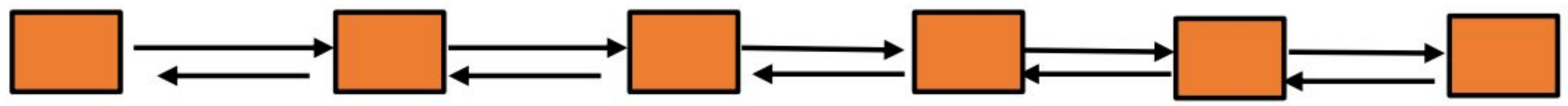
Pick $K = \sqrt{N}$

Q: what is the total recomputation cost?

Memory dynamics

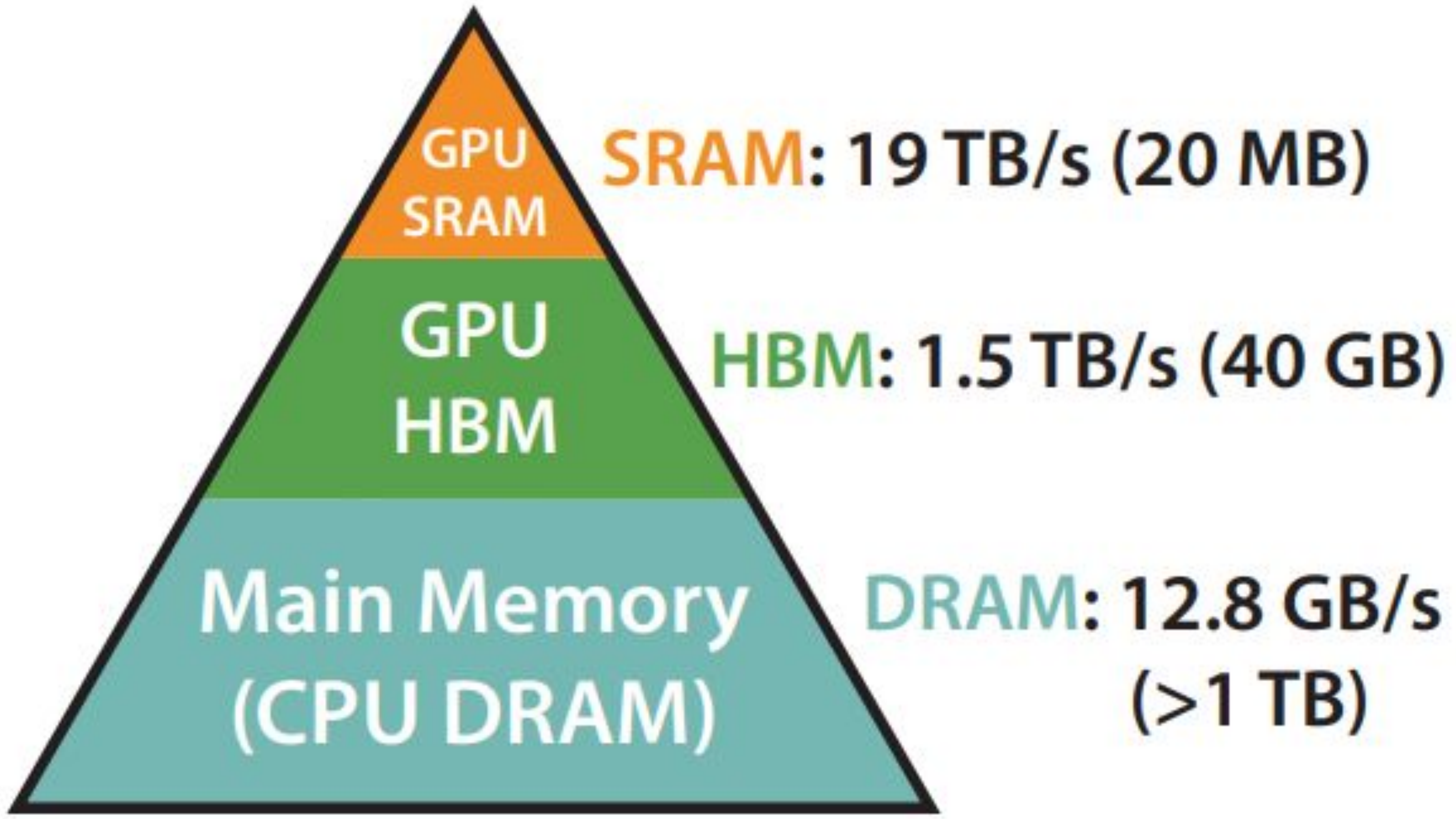


Other factors



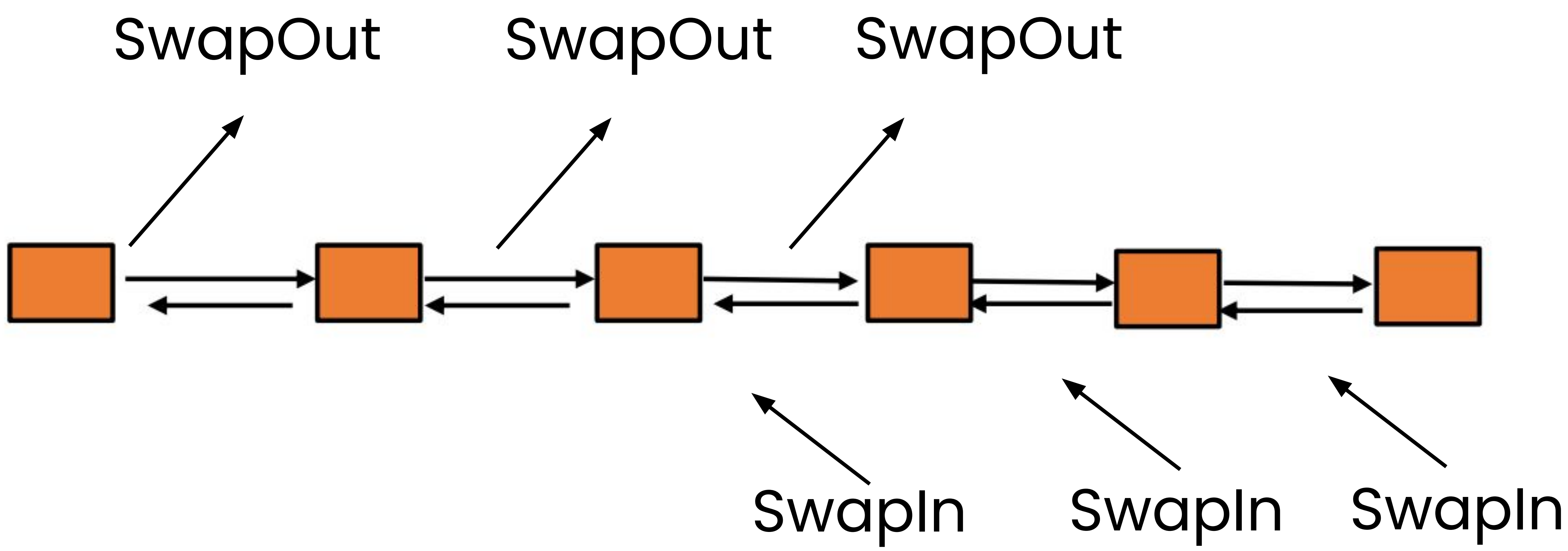
- Model are heterogenous
- Which layer to checkpoint at
  - Could influence memory cost because layer out has different sizes
  - Could influence the recompute cost because the computation between two checkpoints could be different
- Only applies to activations!

Alternative Method: Move to DRAM



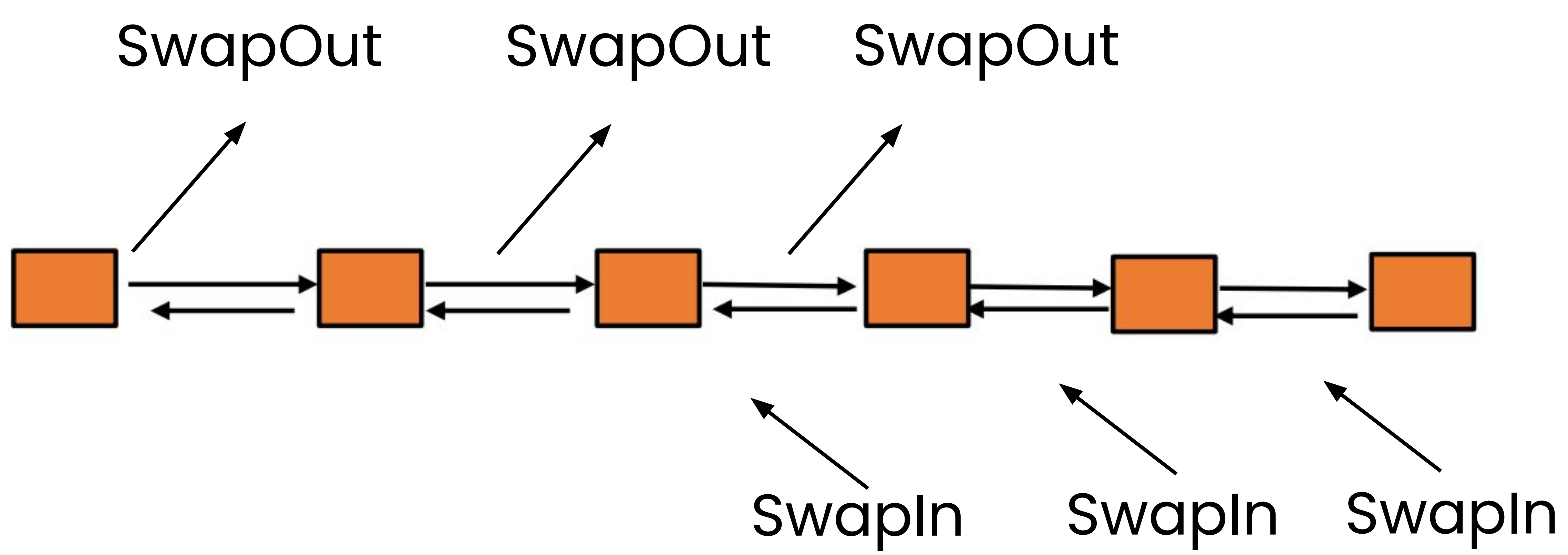
CPU Swap

- SwapIn: swap from CPU DRAM to HBM
- SwapOut: swap from HBM to CPU DRAM
- This applies to both weights and activations!



Discussion

- When will this work and when will this not work?

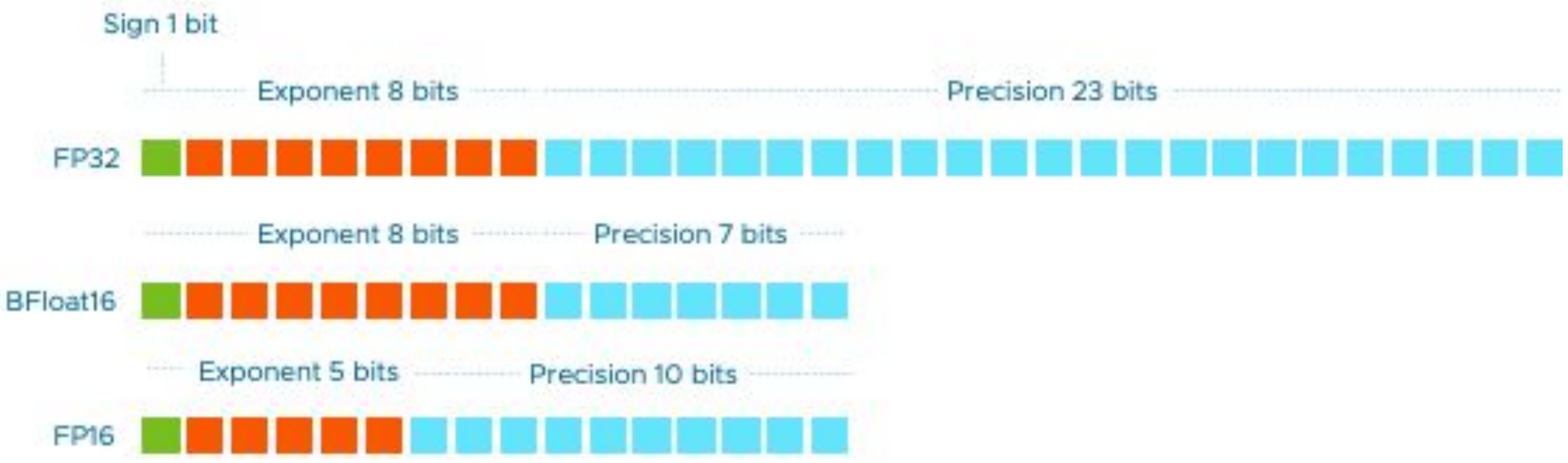


Reduce Memory of Parameters: Quantization

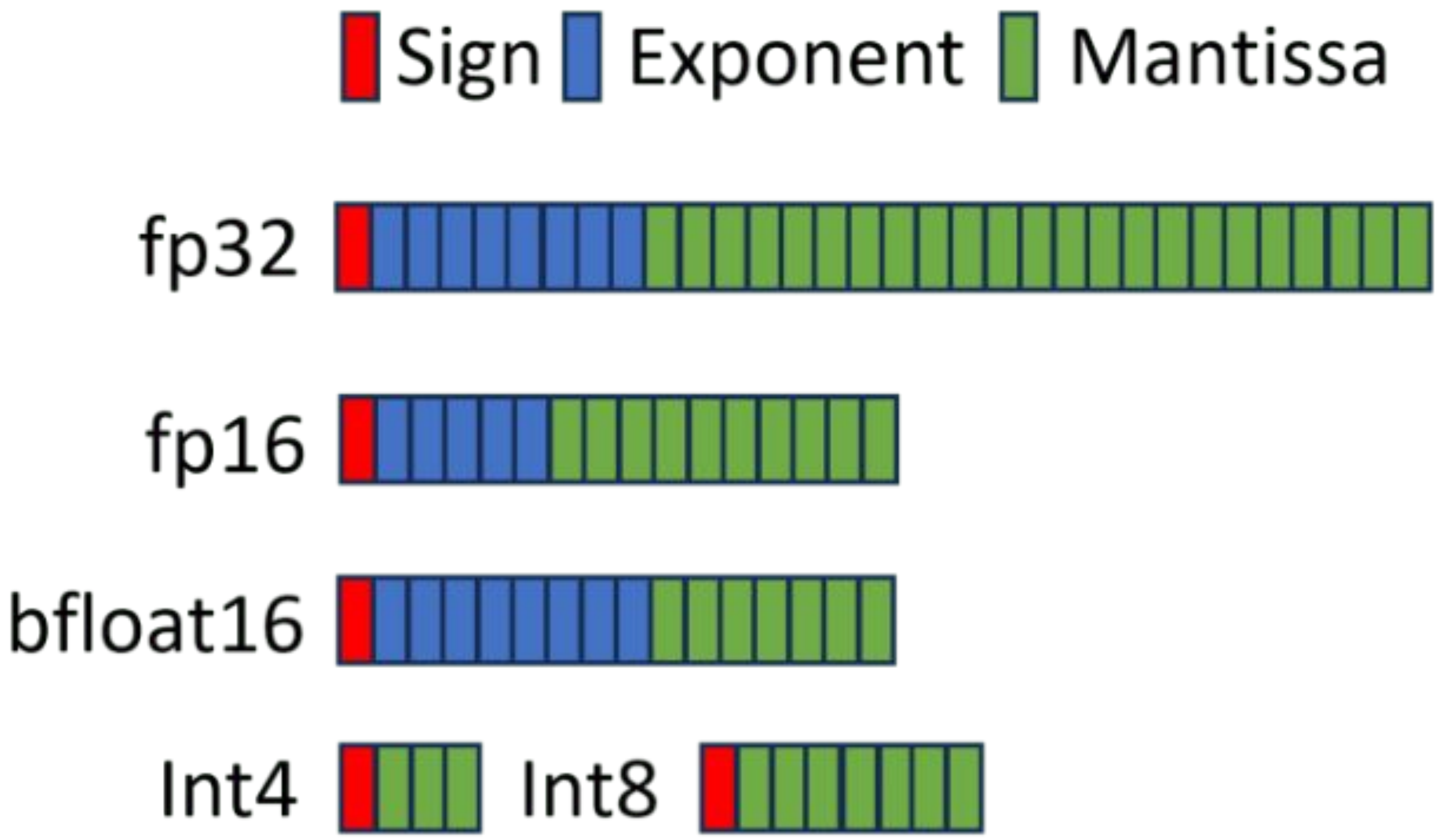
- Use a lower precision than fp32
- FP32 -> fp16: 2x reduction on memory
- Issue: might lose accuracy

How to do downcast?

- FP32 -> BF16: keep the exponent part and downcast the precision part
- FP32 -> fp16: convert exponent and precision part
- A lot of papers to discuss how to downcast without losing accuracy



Discussion: how to downcast to FP8, int4, 1-bit?



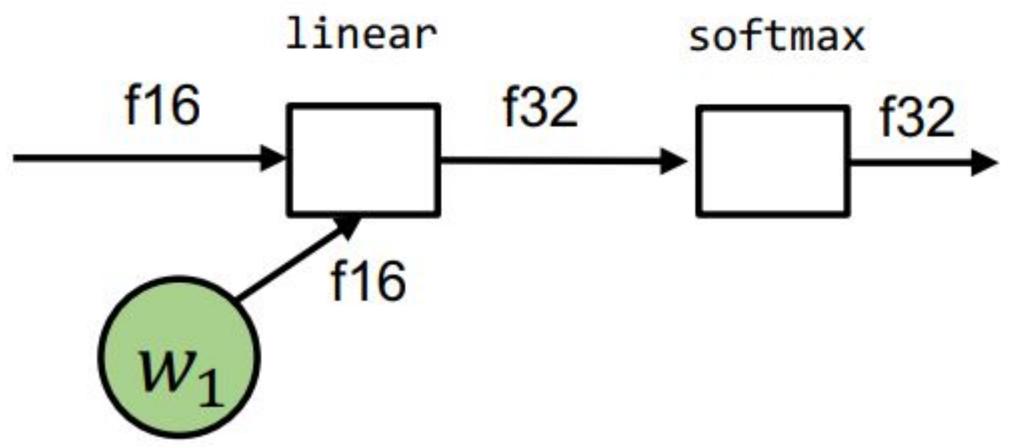
It is more about memory

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*	
INT8 Tensor Core	624 TOPS | 1248 TOPS*	

Form Factor	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core	989 teraFLOPS[2]
BFLOAT16 Tensor Core	1,979 teraFLOPS[2]
FP16 Tensor Core	1,979 teraFLOPS[2]
FP8 Tensor Core	3,958 teraFLOPS[2]

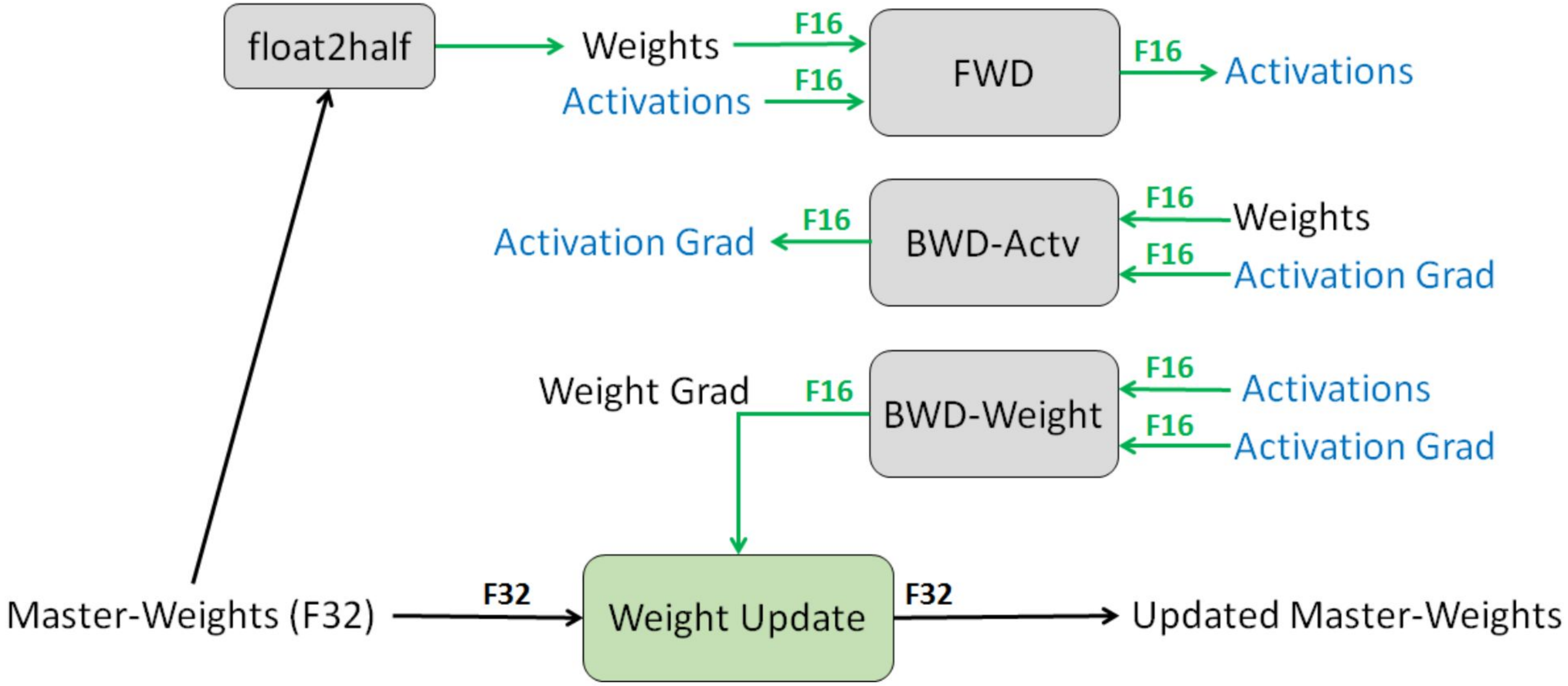
One way: Mix-precision training

- Some layers are more sensitive to dynamic range
  - Normalization: f / sum(f)
  - Softmax (same with normalization)



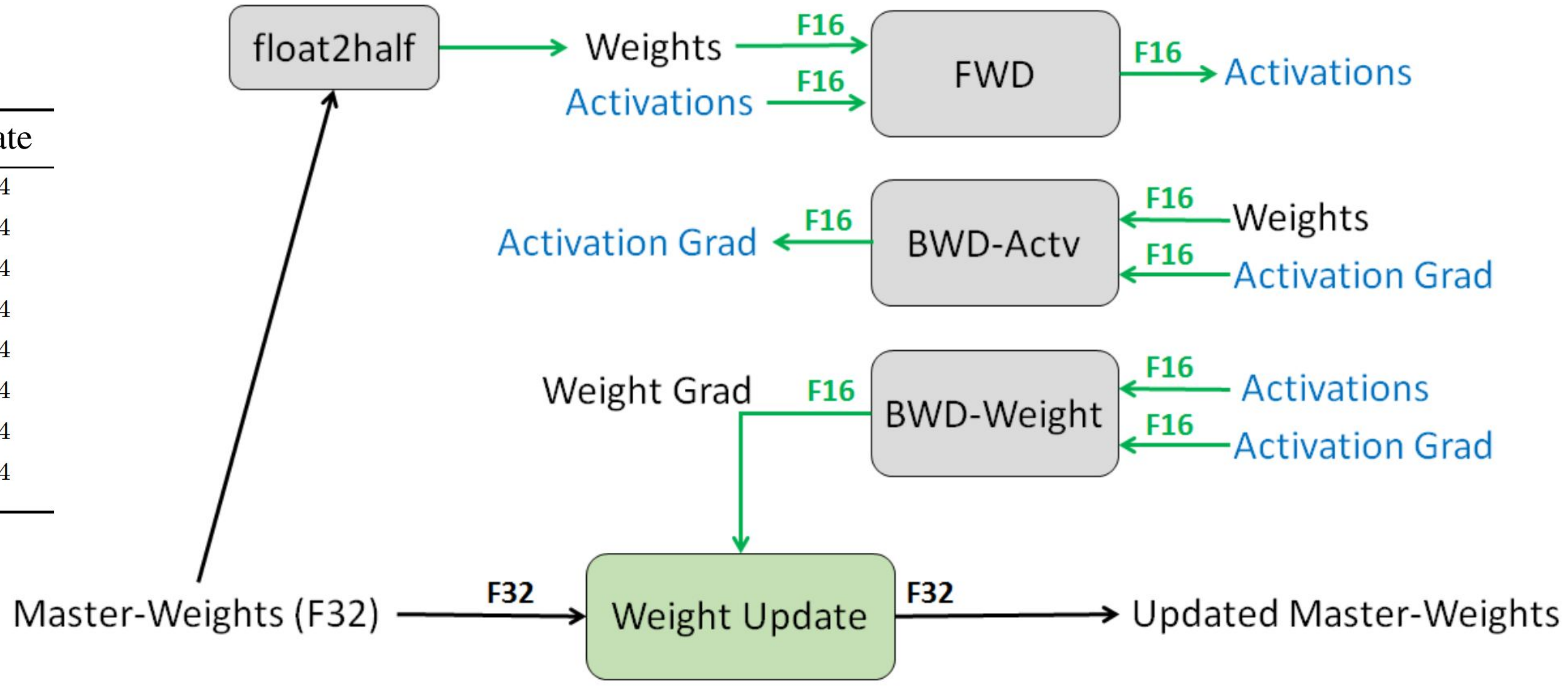
- Common issues: aggregation of a lot entries
  - Param += \sum(grad_t) -> can loss precision during accum
- Idea: identify which ops are sensitive to precisions:
  - Use full precision (fp32) for them via upcasting
  - Use half precision to those robust ops

A standardized 16-32 mix-precision pipeline



Analysis of the memory usage of Mix-precision training

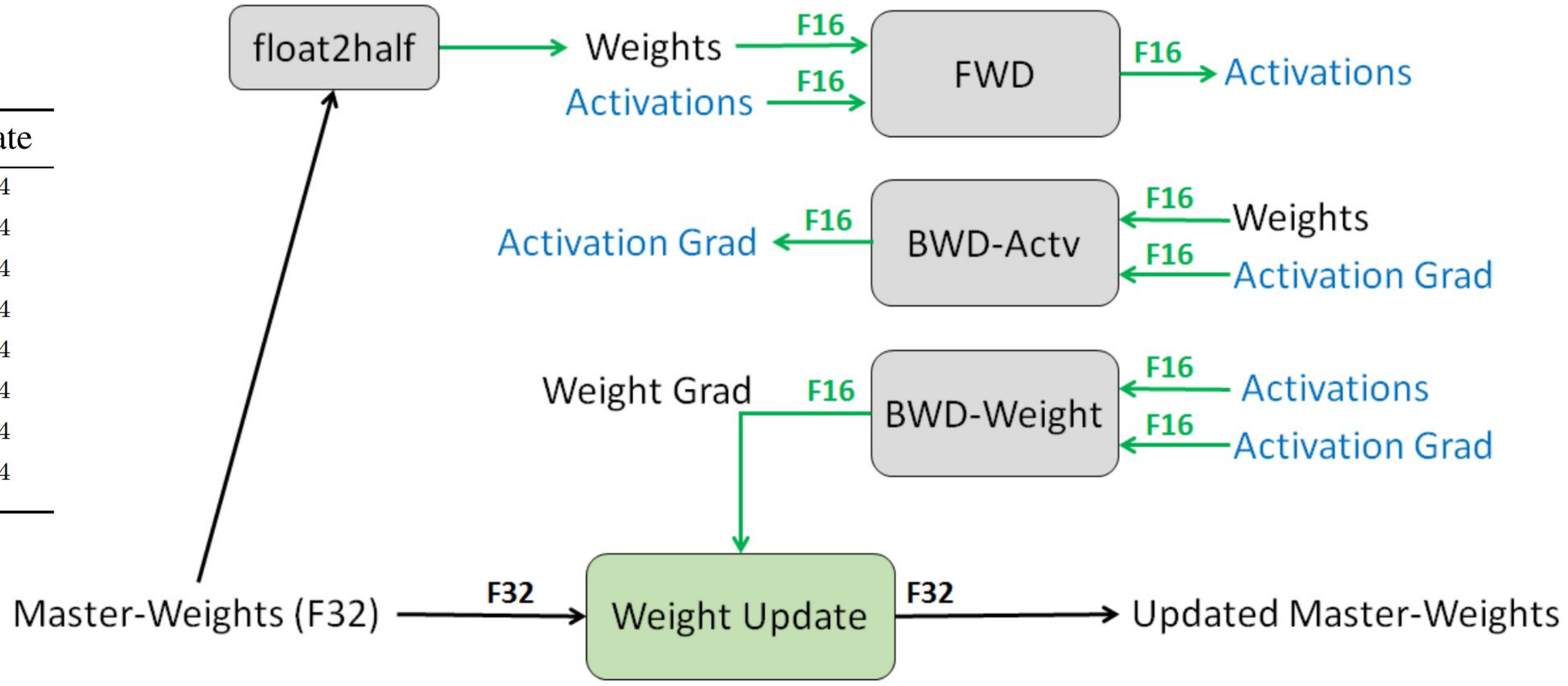
Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}



- Parameters: 175B * (fp16, 2 bytes) = 350G
- Assume we checkpoint at transformer layer boundary:
  - Activations: (N = 96) * 3.2M * 12288 * 2 = 7488 G
- How about optimizer states?

Analysis of the memory usage of Mix-precision training

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}



- How about optimizer states?
  - Master copy (fp32) = 4 * 175 = 700
  - Grad (fp16) = 2 * 175 = 350
  - Running copy (fp16) = 2 * 175 = 350
  - Adam mean and variance (fp32) = 2 * 4 * 175 = 1400G
- Rule the thumb: (4 + 2 + 2 + 4 + 4) N = 16N memory for an LLM

Summary

- Understanding deep learning memory
  - Size
  - Lifetime
- Single Device memory saving techniques
  - Checkpointing and rematerialization
  - CPU Swap
  - Quantization and mixed-precision training
- After applying single device memory saving, we still do not have enough memory, what to do?

Next Week

- Guest Lecture by Jason from PyTorch Team
- Attendance is mandatory
- Reading of next week: his lecture and related paper
- We start to talk about Paralellization