



<https://hao-ai-lab.github.io/cse234-w25/>

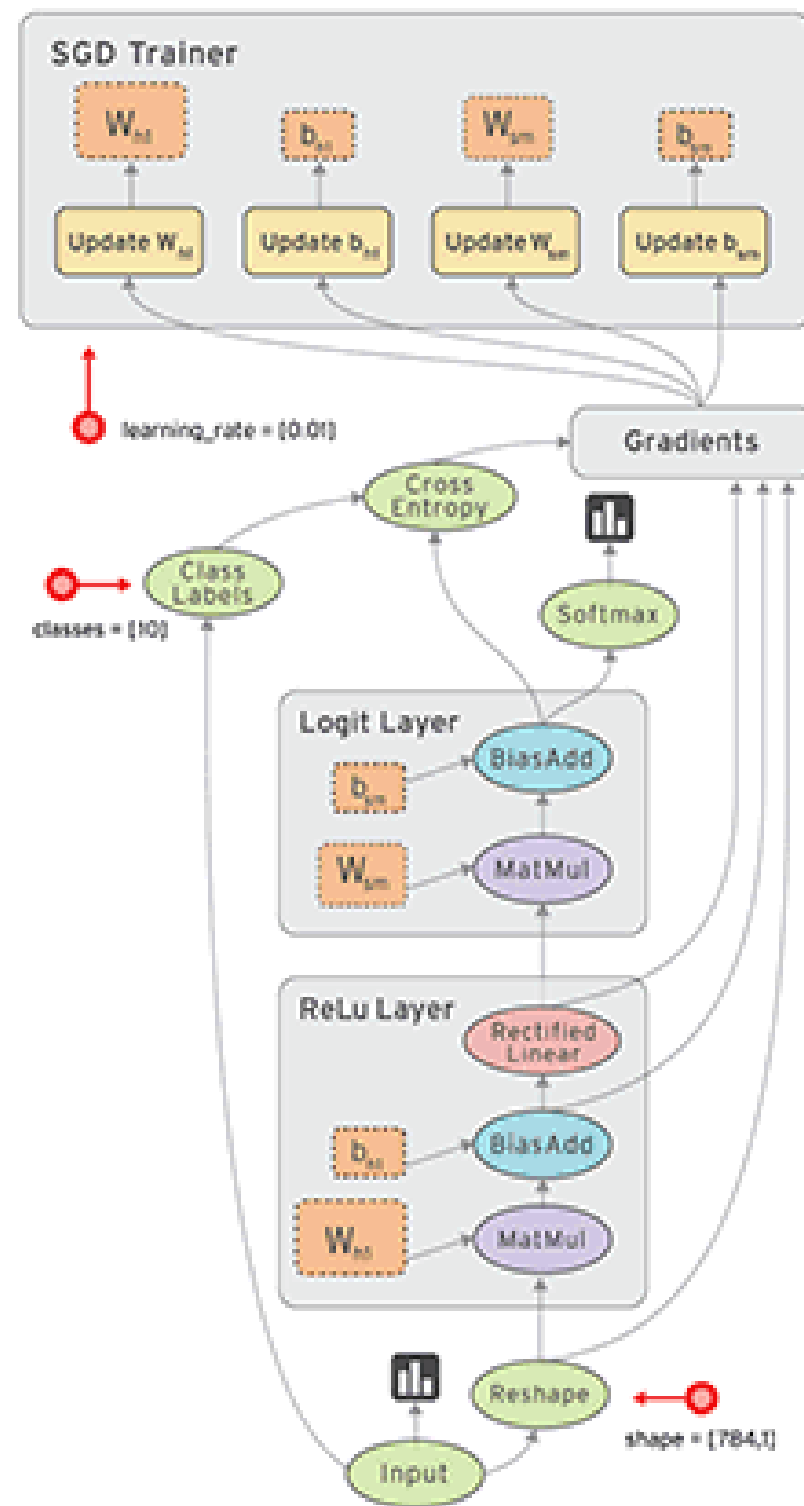
CSE 234: Data Systems for Machine Learning Winter 2025

LLMSys

Optimizations and Parallelization

MLSys Basics

Big Picture: Where We Are



Dataflow Graph

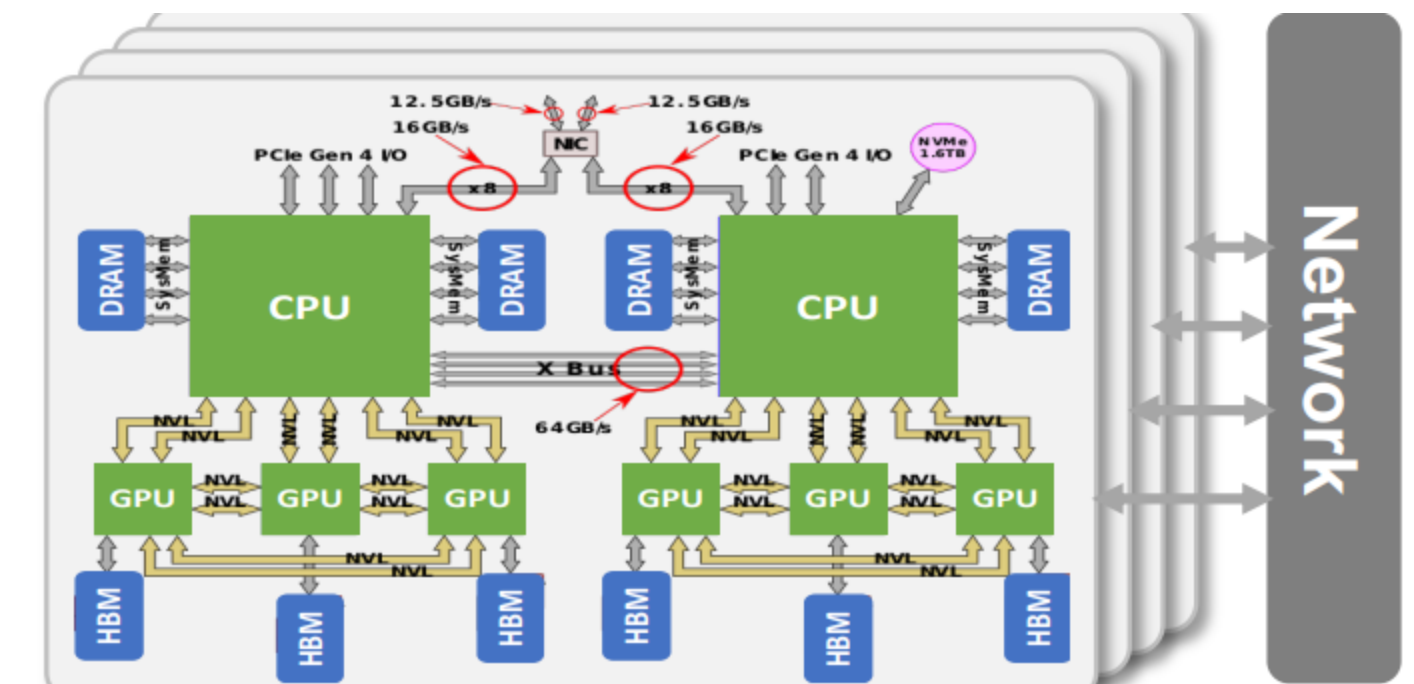
Autodiff

Graph Optimization

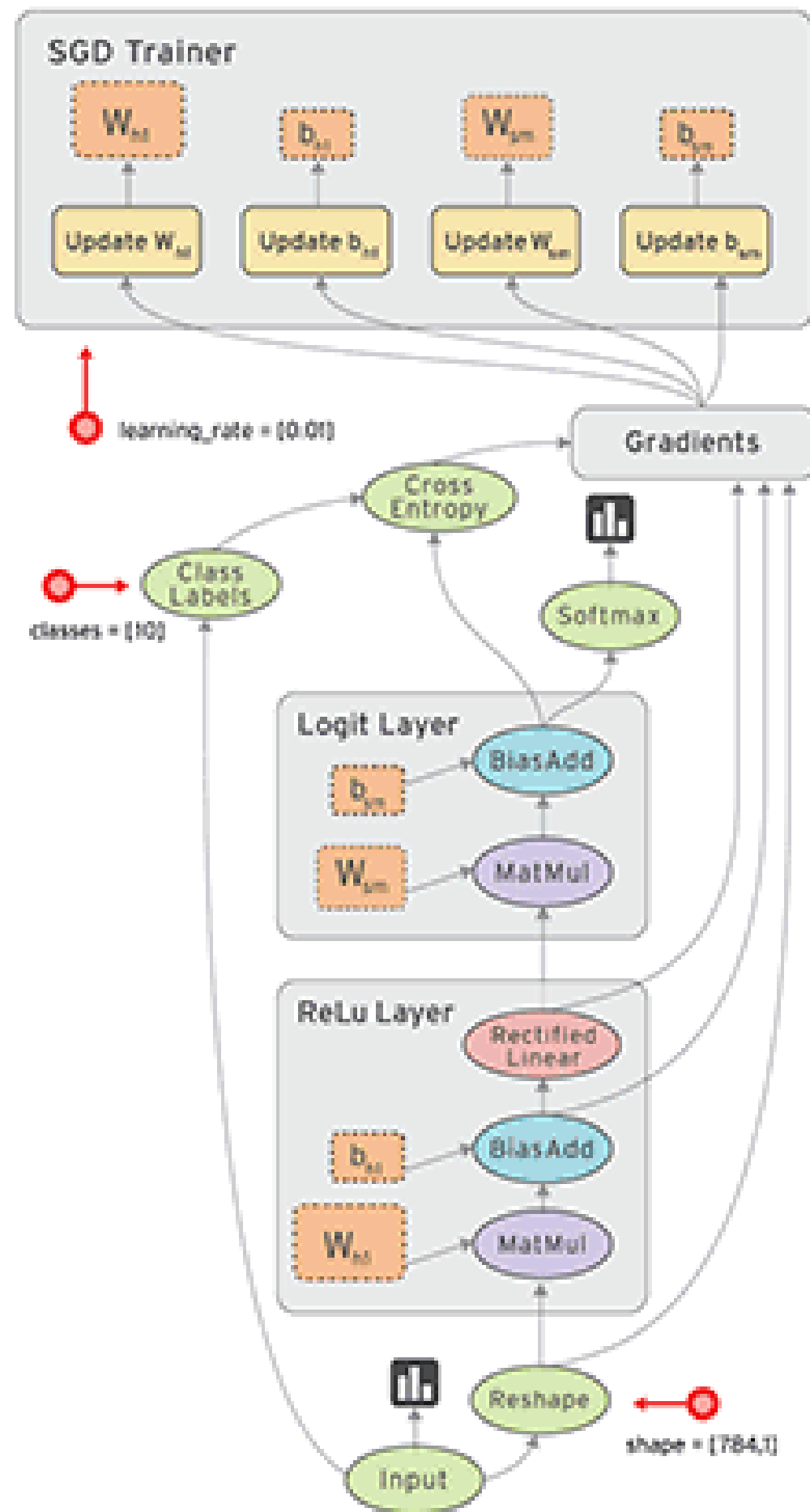
Parallelization

Runtime

Operator optimization/compilation



Run the graph: gradient descent with minibatches



for $t = 1 \rightarrow N$

$$\theta^{(t+1)} = f(\theta^{(t)}, \nabla_L(\theta^{(t)}, D^{(t)}))$$

Disambiguate the Term “Batch”

- In deep learning training
 - Batch
 - Mini-batch
 - Micro-batch

In optimization and gradient descent:

- Full-batch gradient descent
- stochastic gradient descent
- In big data processing
 - Batch/offline processing
 - Streaming/online processing

Disambiguate the Term “Batch”

Term	Context	Meaning
Batch	Deep Learning	A group of training samples processed together
Mini-Batch	Deep Learning	A smaller subset of a dataset used per training iteration
Micro-Batch	Deep Learning	A split of a mini-batch, often used for pipeline parallelism
Batch Processing	Big Data	Processing large datasets in bulk
Micro-Batching	Big Data	Processing small groups of events in short time windows

Today's Learning Goals

- Memory and Scheduling
 - Checkpointing and rematerialization
 - Swapping
- Memory and Compute
 - Quantization
 - Mixed precision

Dataflow Graph

Autodiff

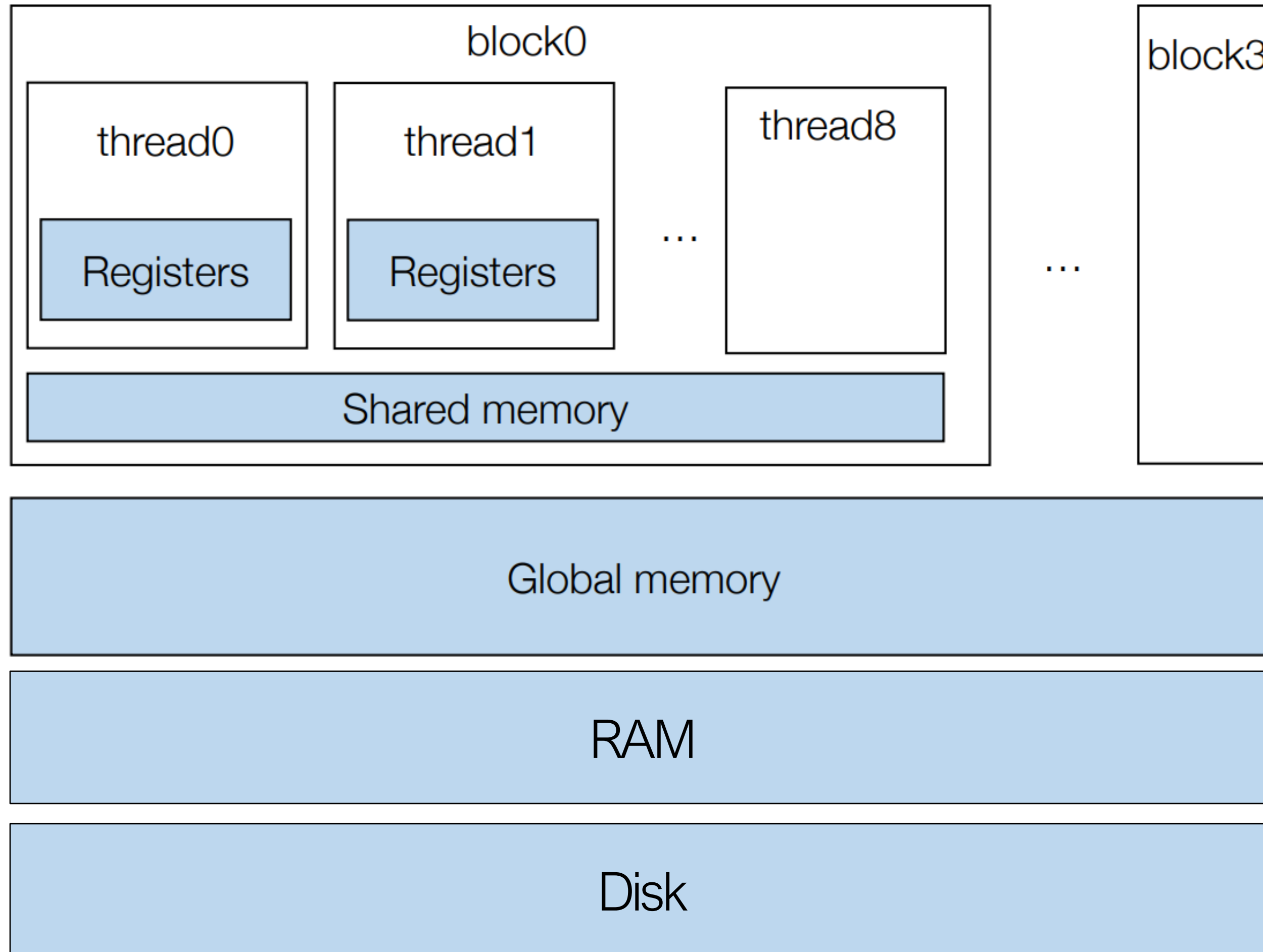
Graph Optimization

Parallelization

Runtime

Operator

Recall: data movement path



Shared memory: 64 KB per core

GPU memory(Global memory):

RTX3080 10GB

RTX3090 24GB

A100 40/80 GB

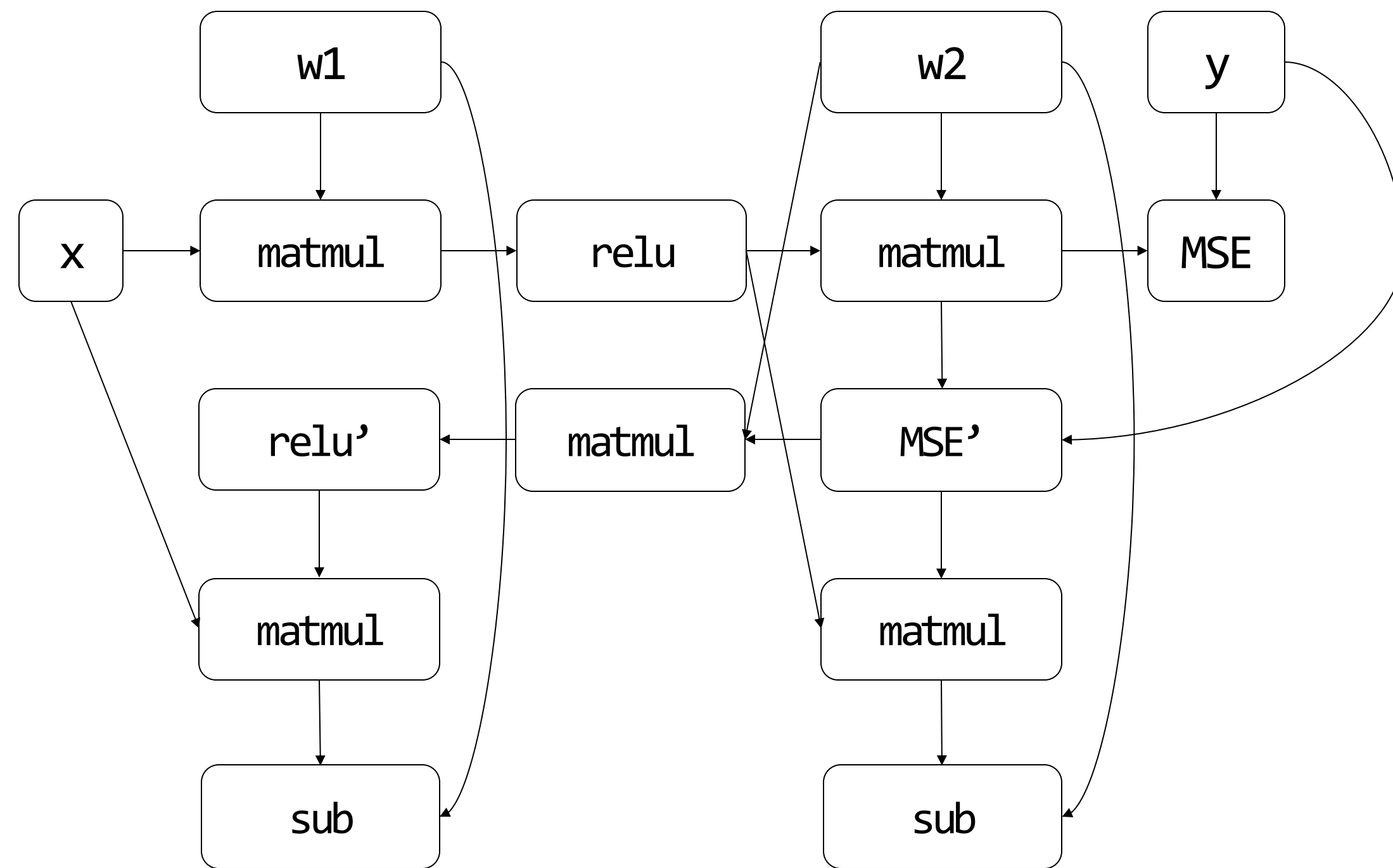
Our Goal on Memory

- Fit the workload on limited memory and ensure
peak memory < available memory

Note:

- Normally, we do not need to min (memory)
- We do not need to min(max(memory))
- We just need $\max(\text{memory}) < \text{available memory}$
 - Unless otherwise specificized

Source of Memory Consumption



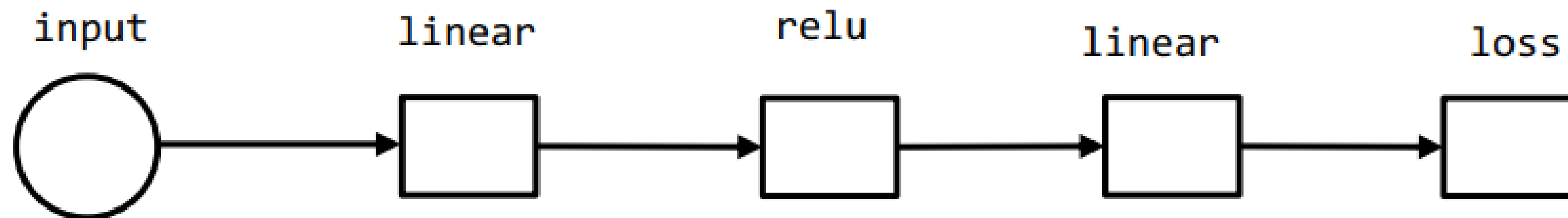
Sources of memory consumption

- Model weights
- Intermediate activation values
- Optimizer states

Methods to Analyze Memory

- Size
 - How large the memory is
- Lifetime:
 - When will this memory be needed and when not

Start with Inference: Lifetime?

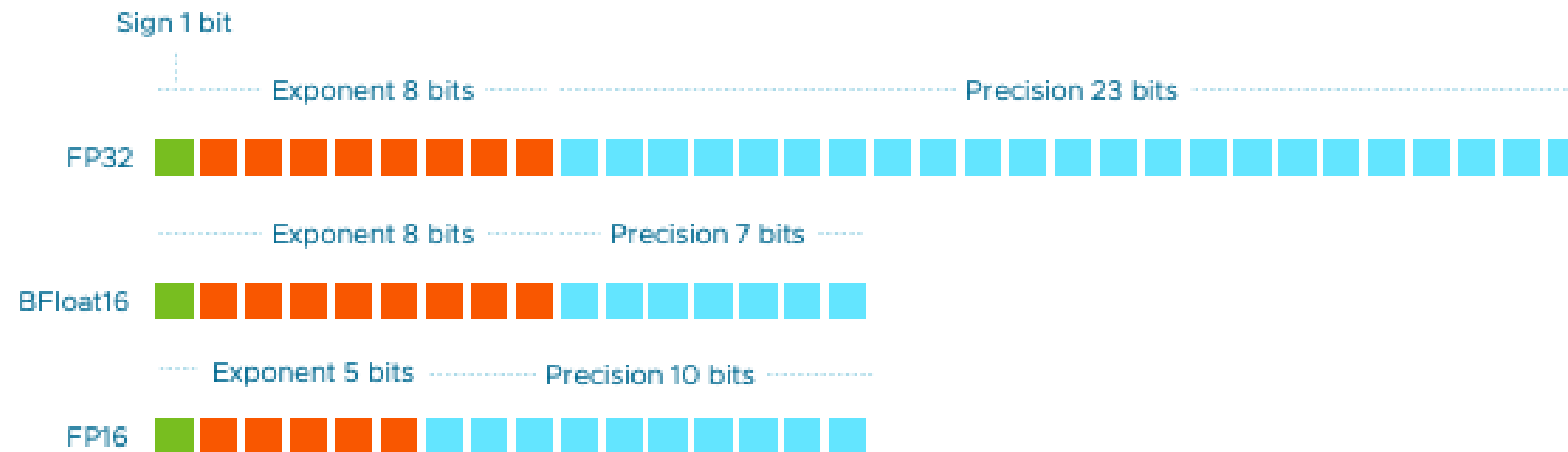


We only need $O(1)$ memory for computing the final output of a N layer deep network by cycling through two buffers

Lifetime of

- weights
- activations?
- Optimizer states?

Estimate size: Popular floating point standards



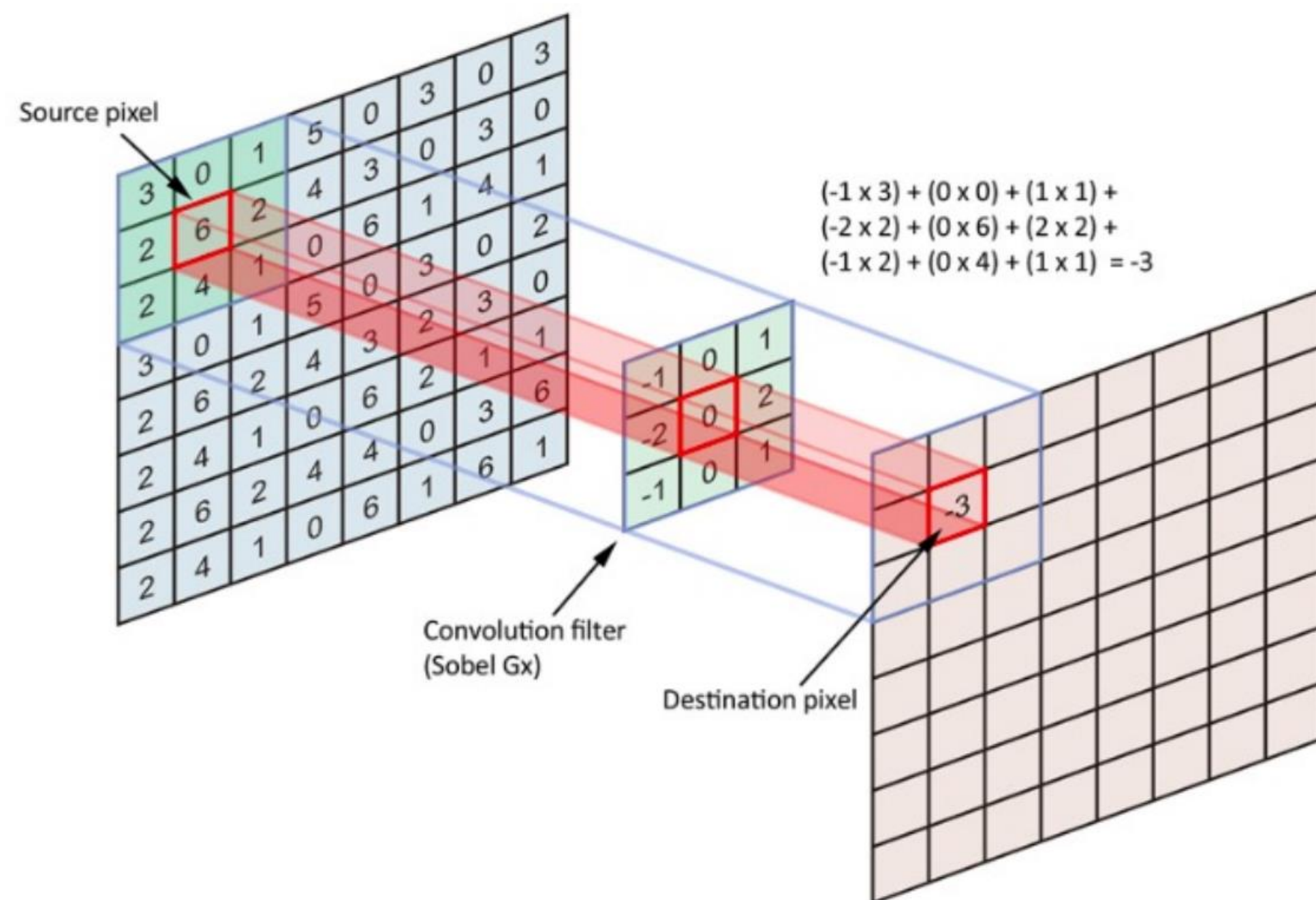
- What does **exponent** and **fraction** control in float point representation?
- What's the difference between bf16 and fp16?
- Will come back to this in detail later

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
 - $175\text{B} * 2 / 4 = 350\text{G} / 700\text{G}$
 - Rule of thumb: check precision, and $N*2$ (16bits) or $N*4$ (32bits)

Estimate the activation size



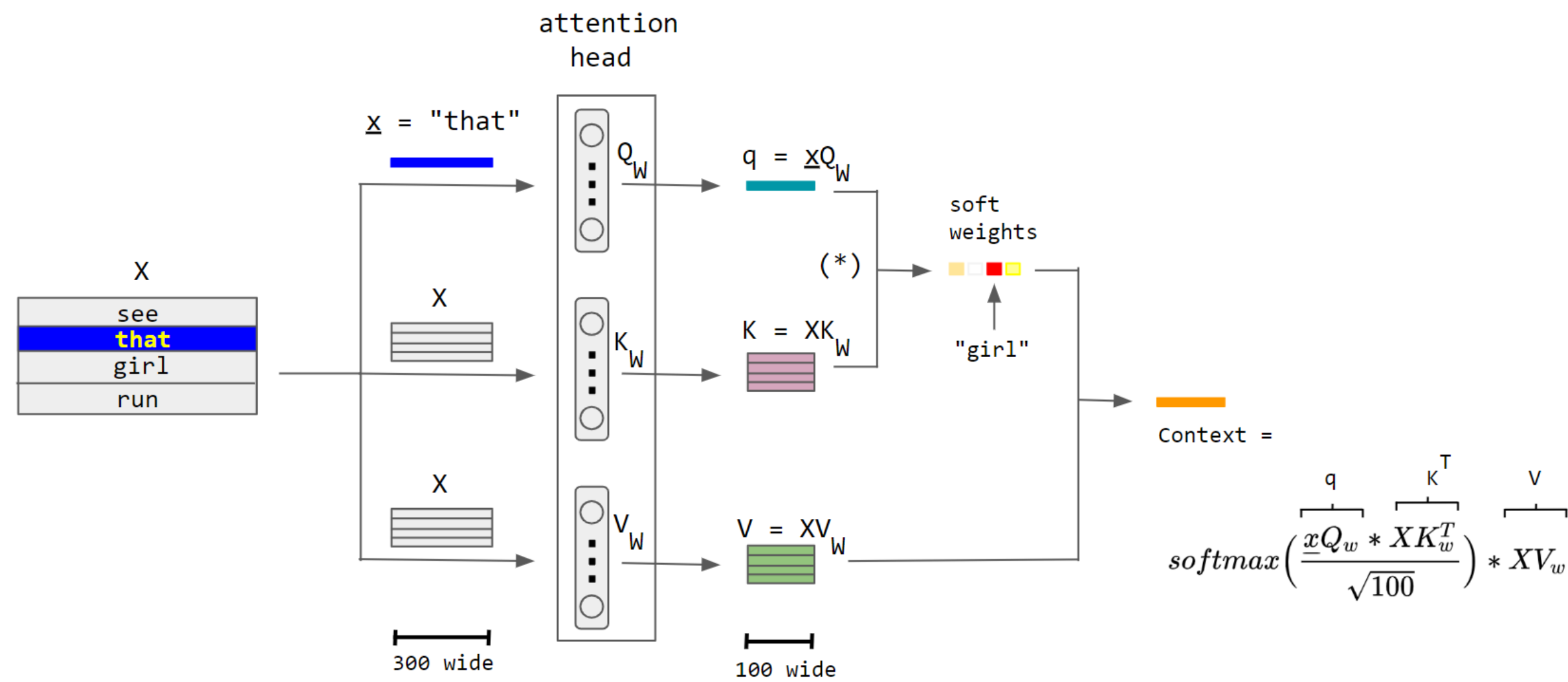
- Conv2d activation:
 - Input: bs, nc, wi, hi
 - Output: bs, nc, wo, ho
 - Activation size:
 - $bs * nc * wo * ho * \text{sizeof}(\text{element})$

Estimate the activation size: MLP

$$C = X * W$$

- Matmul activation
 - Input: $bs * m * n$
 - Output: $bs * m * p$
 - Activation size:
 - $bs * m * p * \text{sizeof}(\text{element})$

Estimate the activation size: Transformers



- transformer activation
 - Input: $bs * h * seq_len$
 - Output: $bs * h * seq_len$
 - Activation size:
 - $bs * h * seq_len * \text{sizeof}(\text{element})$

Estimate the activation size: Transformers

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}

GPT-3 Transformers per-layer activation (assuming seq_len = 1):

- $\text{bs} * \text{seq_len} * d_{\text{model}}$: $3.2\text{M} * 12288 = 39.321\text{B} = 78 / 156 \text{ G}$
- Note: For each operator inside a transformer layer, we also have activations. We'll come back to this later.

Estimate the Optimizer State Size: Adam

- 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)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t (Resulting parameters)

Optimizer state: first
moment estimate

(mean)

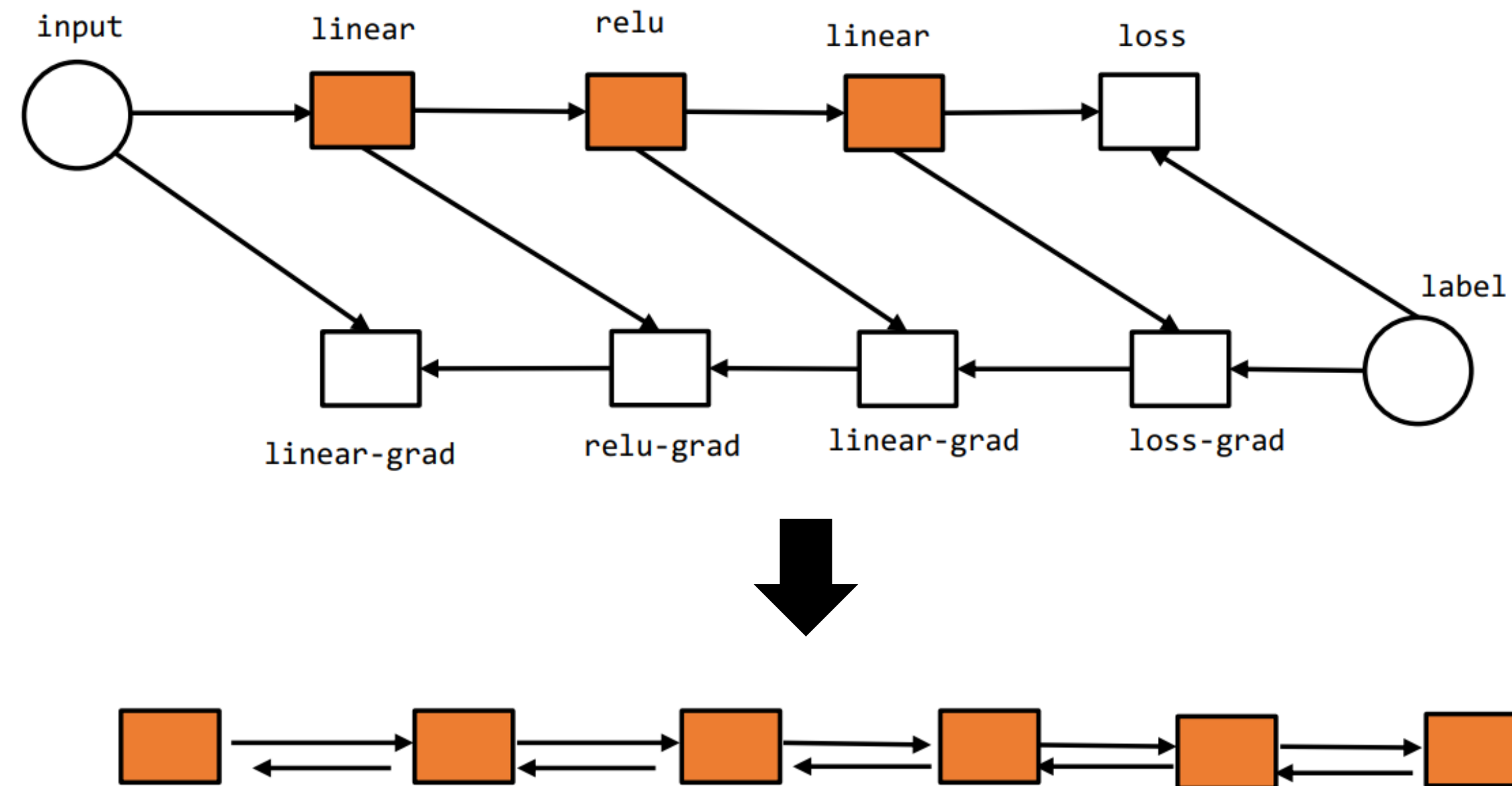
Optimizer state: second
moment estimate

(variance)

Estimate the Optimizer State Size?

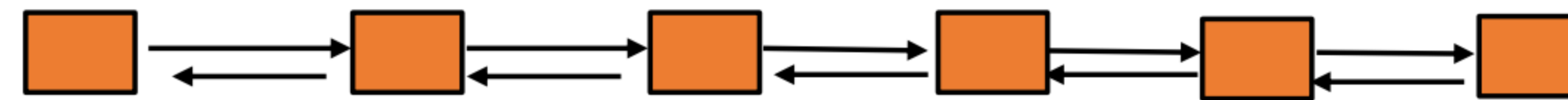
- Gradient w.r.t. parameters: $N * \text{sizeof}(\text{element})$
- First moment: $N * \text{sizeof}(\text{element})$
- Second moment: $N * \text{sizeof}(\text{element})$
- In total:
 - $3N * \text{sizeof}(\text{element})$

Lifetime of activations at training



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

Memory Overview



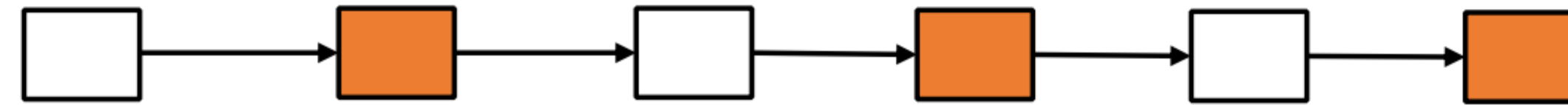
- Parameters: $175\text{B} * (\text{fp32}) = 350 / 700 \text{ G}$
- Activations:
 - At the transformer boundary: $(N = 96) * 78 / 156 \text{ G} = 7488 / 14976 \text{ G}$
 - (This is not accurate because transformers is a composite layers.)
 - Optimizer states: $(\text{precision: fp32}) * 3 * 175\text{B} = (12) * 175 \text{ G}$

Reduce memory

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

Reduce activation memory

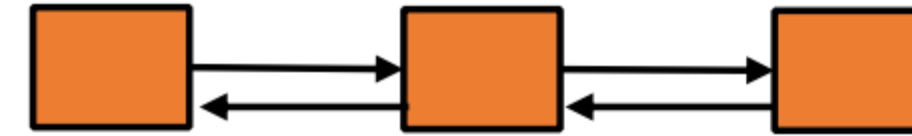
Step 0:



Step 1:



Step 2:

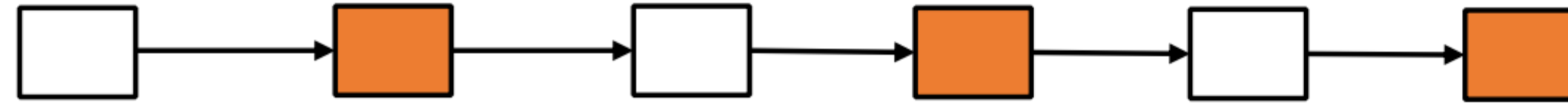


Idea:

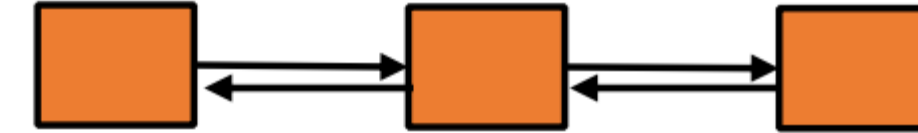
- The activation is not needed again until the backward pass comes
- Discard some of them and recompute the missing intermediate nodes in small segments

Reduce activation memory

Step 0:



Step 1:



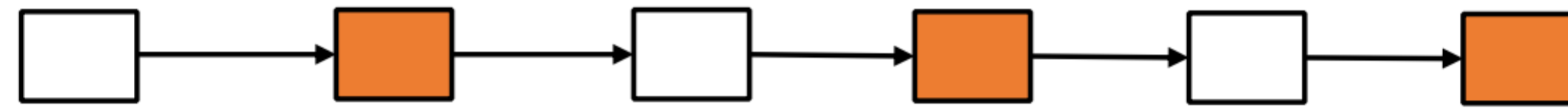
Step 2:



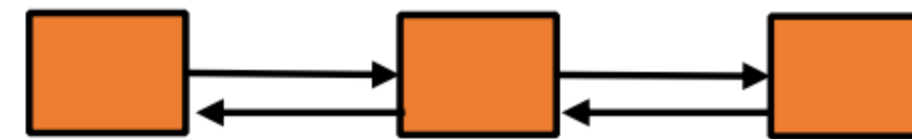
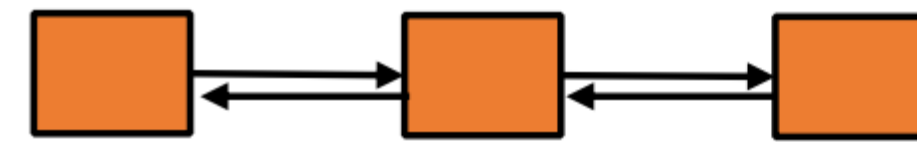
- Option 1: discard nothing
 - Memory +, compute --
- Option 2: discard all and recompute for each layer
 - Memory --, compute++
- We want to strike a balance?

Reduce activation memory

Forward computation



Gradient per segment
with re-computation



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

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

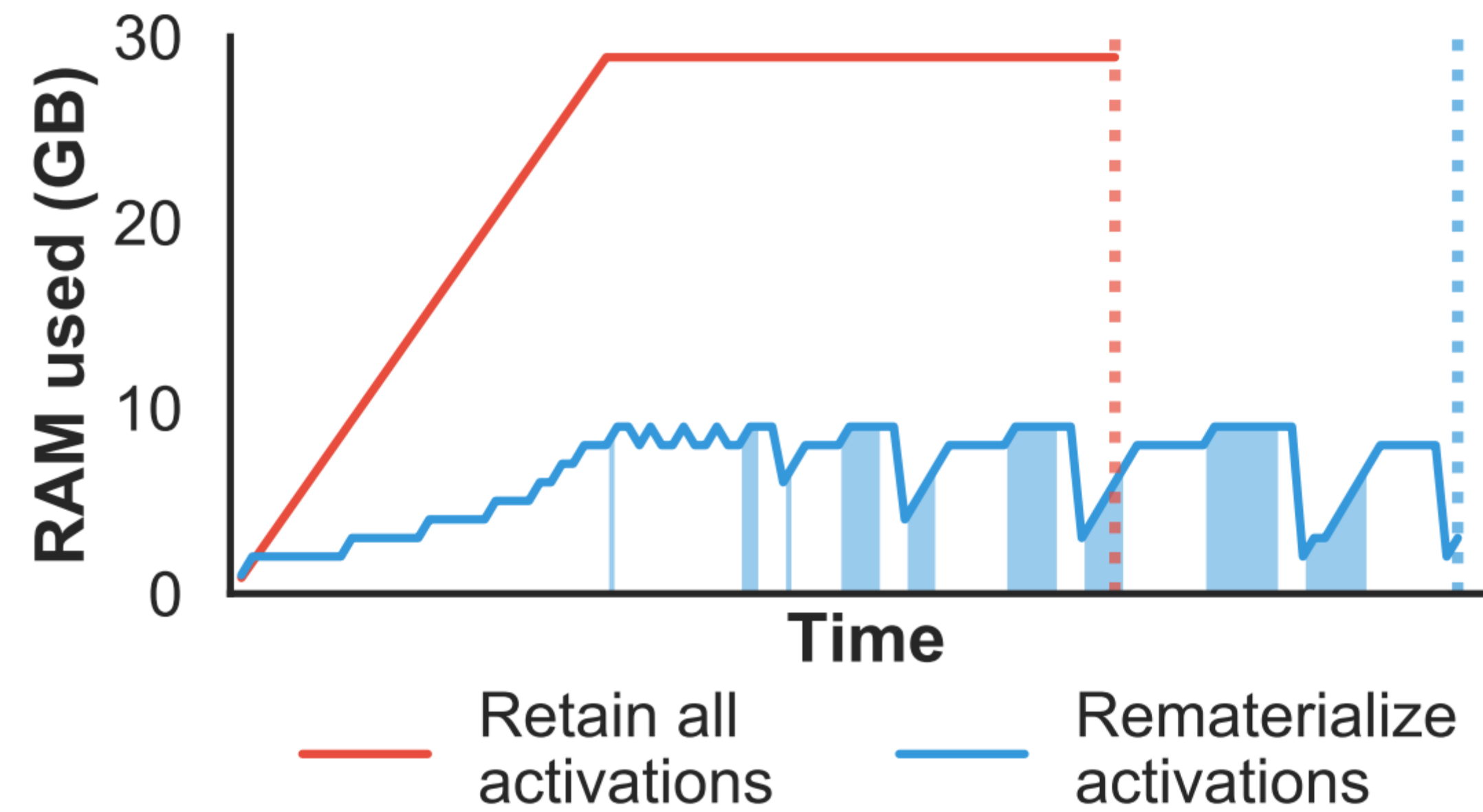
Pick $K = \sqrt{N}$

Checkpoint cost

Re-computation cost

Q: In this case: what is the total recomputation

Memory dynamics



In practice

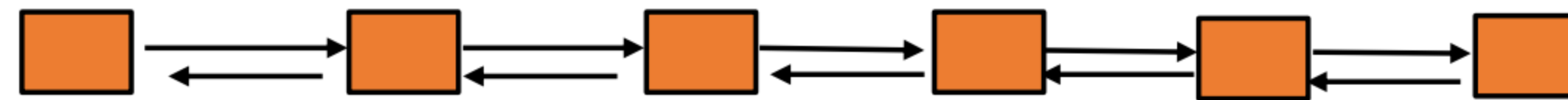
torch.utils.checkpoint

- NOTE

Checkpointing is implemented by rerunning a forward-pass segment for each checkpointed segment during backward propagation. This can cause persistent states like the RNG state to be more advanced than they would be without checkpointing. By default, checkpointing includes logic to juggle the RNG state such that checkpointed passes making use of RNG (through dropout for example) have deterministic output as compared to non-checkpointed passes. The logic to stash and restore RNG states can incur a moderate performance hit depending on the runtime of checkpointed operations. If deterministic output compared to non-checkpointed passes is not required, supply `preserve_rng_state=False` to `checkpoint` or `checkpoint_sequential` to omit stashing and restoring the RNG state during each checkpoint.

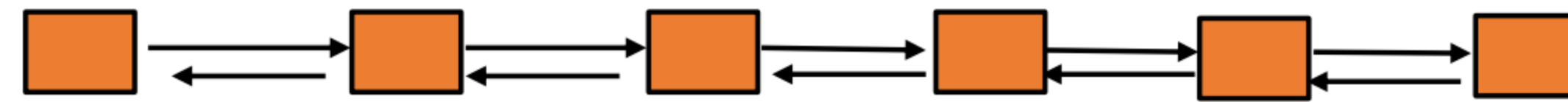
```
"activation_checkpointing": {  
  "partition_activations": false,  
  "cpu_checkpointing": false,  
  "contiguous_memory_optimization": false,  
  "number_checkpoints": null,  
  "synchronize_checkpoint_boundary": false,  
  "profile": false  
}
```

Discussion of Activation Checkpointing



- It is also called: rematerialization, recomputation
- Q: When to and not to enable it?
- Q: Optimal checkpointing policy
 - 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
- Cons: Only applies to activations!

Gradient Accumulation



- 💡 Activation memory is linear to batch size
- Can we still compute using the given batch size but with limited memory

$$\nabla\theta = \nabla_L(\{x_i\}_i^{bs}, \dots)$$
$$\theta = \theta + \nabla\theta$$

vs.

for $t = 1 \rightarrow \#mb$

$$\nabla\theta += \nabla_L(\{x_i\}_i^{mbs}, \dots)$$
$$\theta = \theta + \nabla\theta$$

Gradient Accumulation in Practice

```
optimizer = ...

for epoch in range(...):
    for i, sample in enumerate(dataloader):
        inputs, labels = sample

        # Forward Pass
        outputs = model(inputs)
        # Compute Loss and Perform Back-propagation
        loss = loss_fn(outputs, labels)
        loss.backward()
        # Update Optimizer
        optimizer.step()

    optimizer.zero_grad()
```

```
optimizer = ...
NUM_ACCUMULATION_STEPS = ...

for epoch in range(...):
    for idx, sample in enumerate(dataloader):
        inputs, labels = sample

        # Forward Pass
        outputs = model(inputs)
        # Compute Loss and Perform Back-propagation
        loss = loss_fn(outputs, labels)

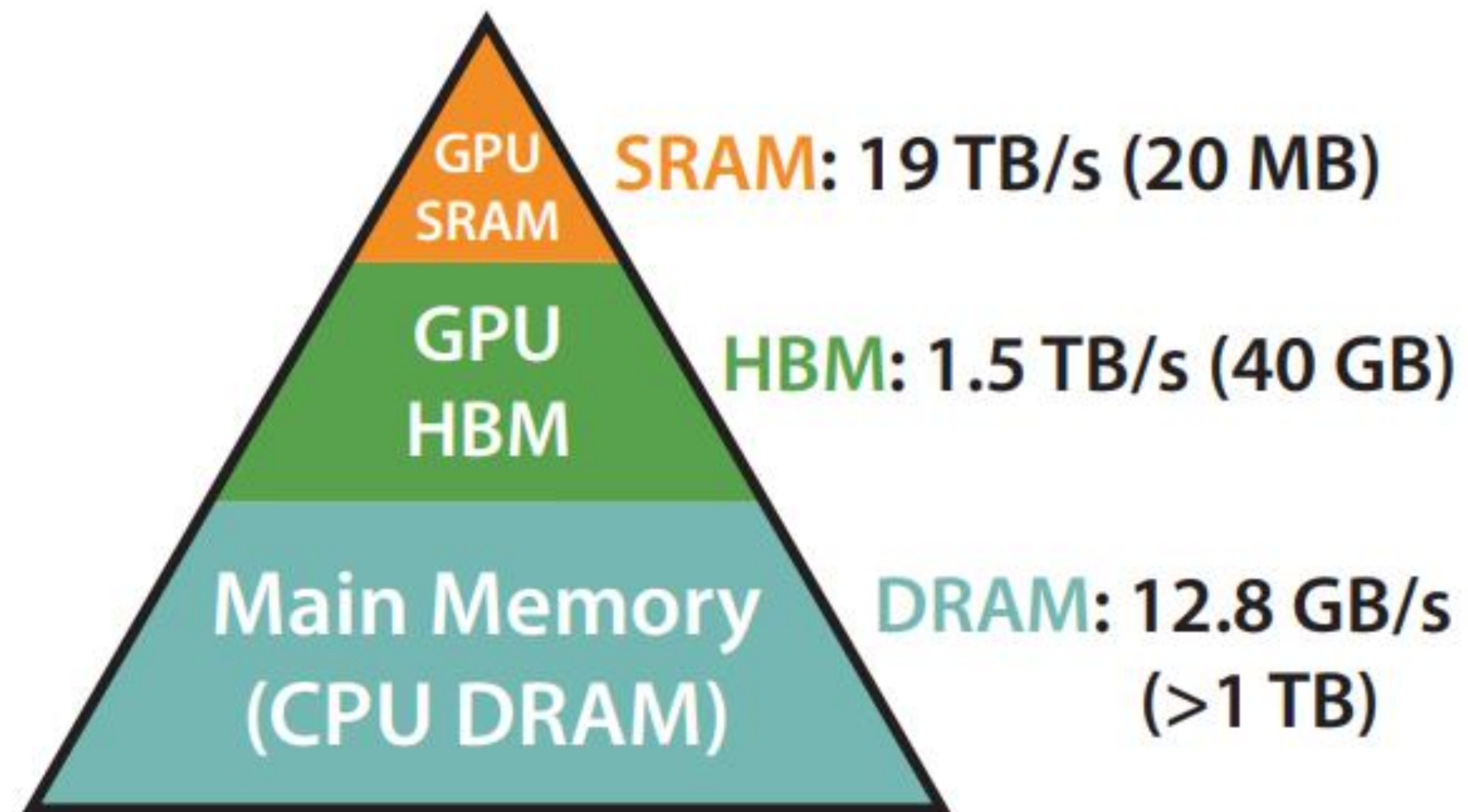
        # Normalize the Gradients
        loss = loss / NUM_ACCUMULATION_STEPS
        loss.backward()

    if ((idx + 1) % NUM_ACCUMULATION_STEPS == 0) or (idx + 1 == len(dataloader)):
        # Update Optimizer
        optimizer.step()

    optimizer.zero_grad()
```

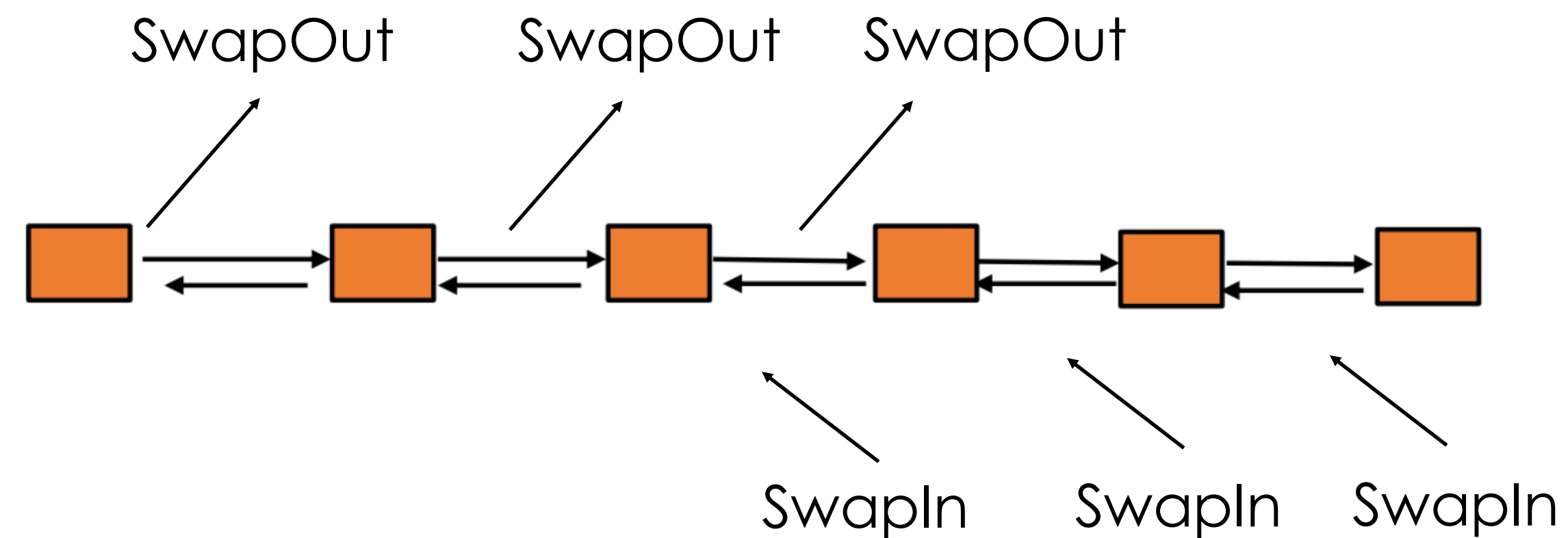
Q: potential cons of gradient accumulation?

Alternative Method: Move to DRAM



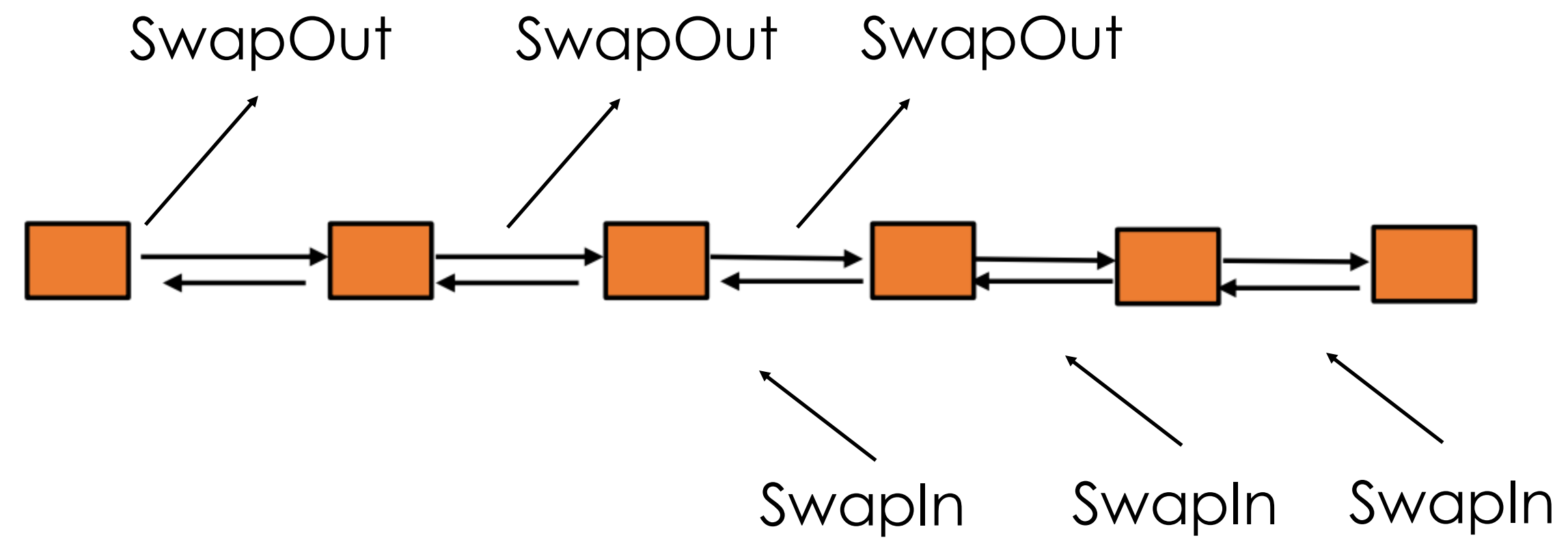
CPU Swap

- 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?



Summary of Memory Optimizations

- Gradient checkpointing
 - Trading compute for memory
- Gradient accumulation
 - Reduce “effective” batch size
- CPU Swapping
 - Use lower-level memory hierarchy

Today's Learning Goals

- Memory and Scheduling
 - Checkpointing and rematerialization
 - Swapping
- Memory and Compute
 - **Quantization**
 - Mixed precision

Dataflow Graph

Autodiff

Graph Optimization

Parallelization

Runtime

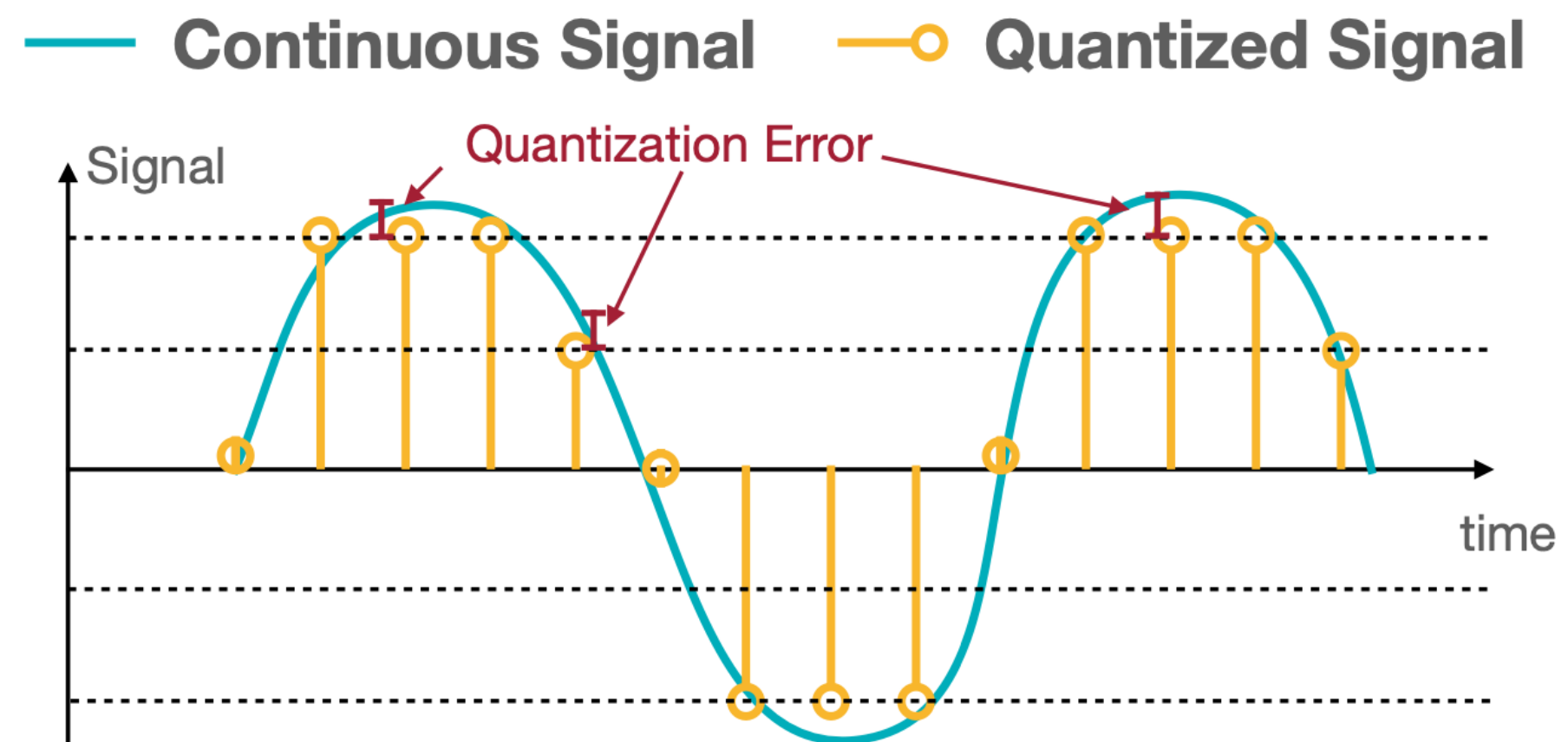
Operator

Reduce Memory of Parameters: Quantization

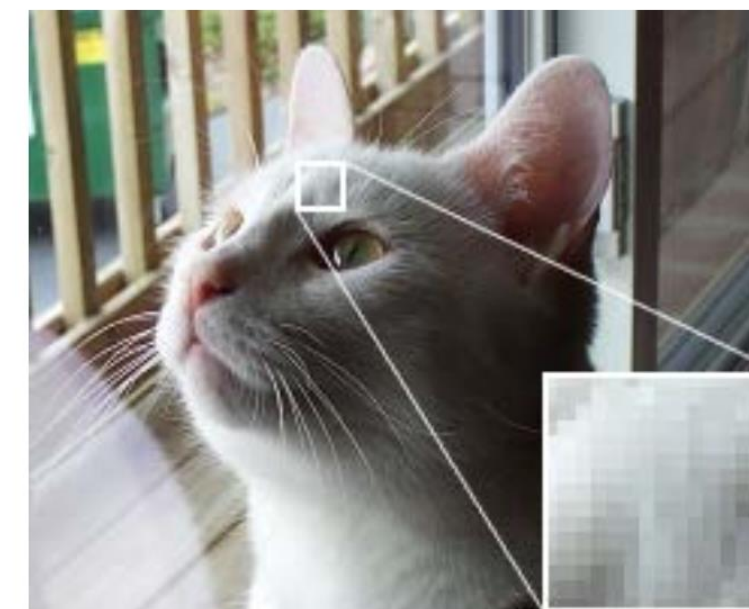
- Observation: all our memory usage (weights, activation, optimizer states) is a multiple of `sizeof(element)`
- 💡 Can we reduce `sizeof(element)`?

What is Quantization

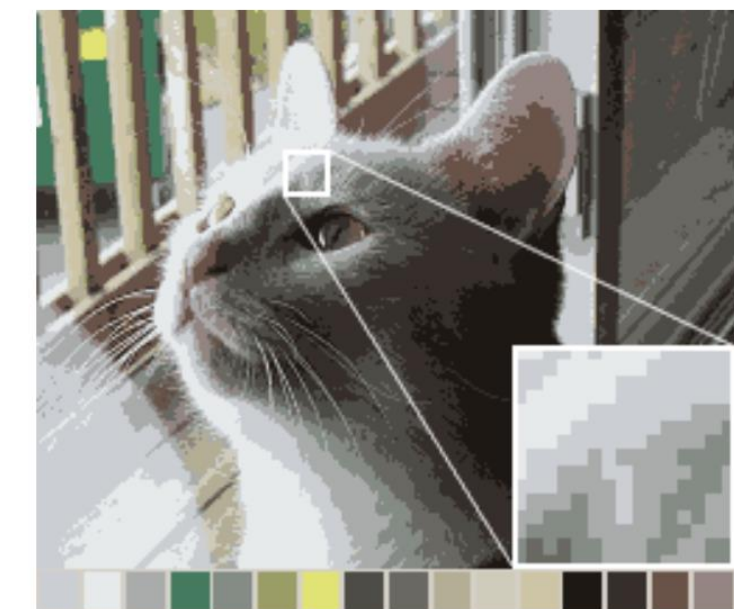
Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.



Original Image



16-Color Image



Images are in the public domain.

“Palettization”

Quantization in ML

High-level Idea: Use a lower-precision representation for data, while:

- (Almost) preserve ML performance, e.g., accuracy
- Accelerate compute
- Reduce memory
- Save energy
- etc.

Quantization

- Digital representation of data
- Basics of quantization
- Quantization in ML
 - Post-training quantization
 - Quantization aware training
- Mixed precision training

Representation of Data: Integer

Unsigned Integer

- n-bit range: $[0, 2^n - 1]$

Signed Integer

- Sign-Magnitude Representation
- n-bit range: $[-2^{n-1} - 1, 2^{n-1} - 1]$
- Problem: Both 000...00 and 100...00 represent 0

Two's complement representation

- n-bit range: $[-2^{n-1} - 1, 2^{n-1} - 1]$

0	0	1	1	0	0	0	1
x	x	x	x	x	x	x	x

$$2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 49$$

Sign Bit

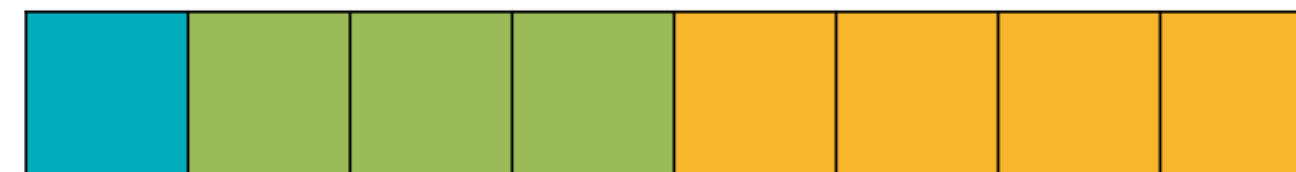
1	0	1	1	0	0	0	1
	x	x	x	x	x	x	x

$$- 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

1	1	0	0	1	1	1	1
x	x	x	x	x	x	x	x

$$-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = -49$$

Fixed-point Number



Integer . Fraction

“Decimal” Point



$\times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times$

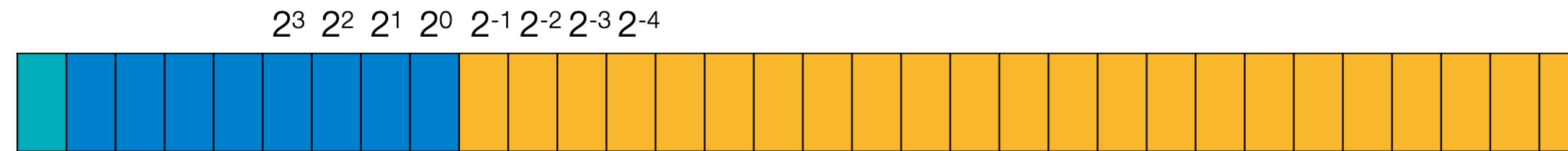
$$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 3.0625$$



$\times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times \quad \times$

$$(-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0) \times 2^{-4} = 49 \times 0.0625 = 3.0625$$

Floating-point Representation



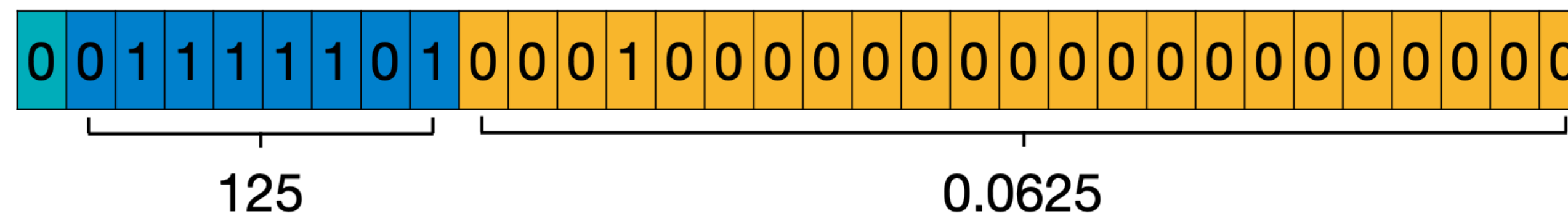
Sign 8 bit Exponent

23 bit Fraction

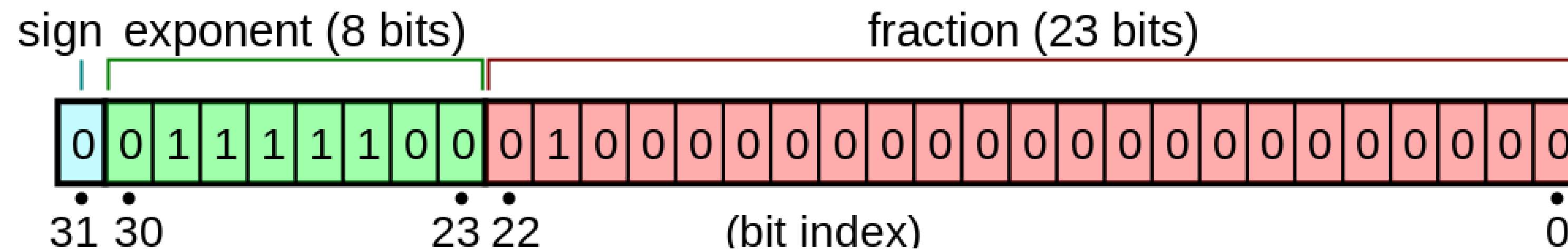
$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \quad \leftarrow \quad \text{Exponent Bias} = 127 = 2^{8-1}-1$$

(significant / mantissa)

$$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$$



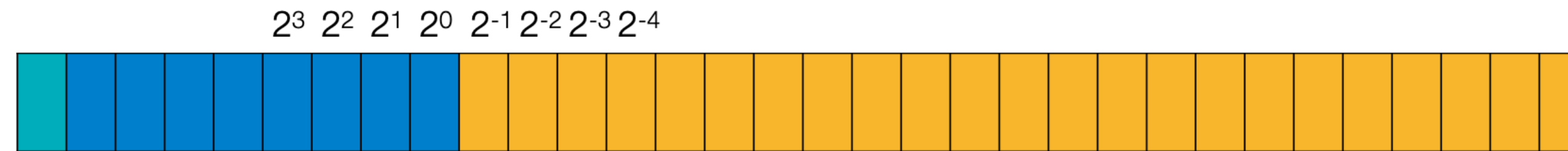
Practice: What is the number



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

$$(-1)^0 \times 2^{124-127} \times (1 + 1 \cdot 2^{-2}) = (1/8) \times (1 + (1/4)) = 0.15625$$

Floating-point Representation



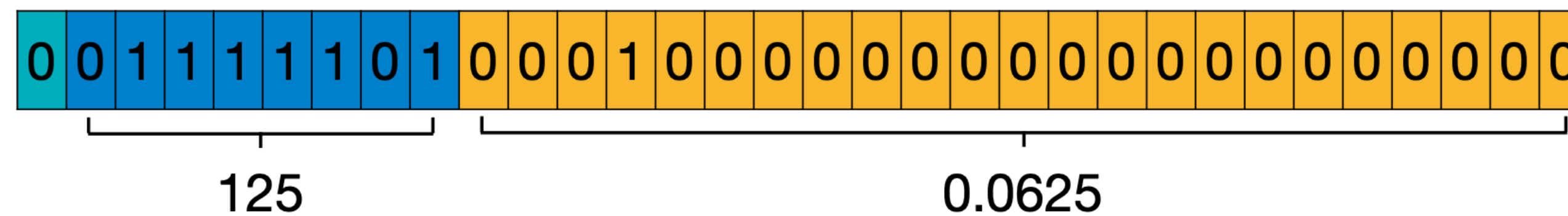
Sign 8 bit Exponent

23 bit Fraction

$$(-1)^{\text{sign}} \times (1 + \text{Fraction}) \times 2^{\text{Exponent}-127} \quad \leftarrow \quad \text{Exponent Bias} = 127 = 2^{8-1}-1$$

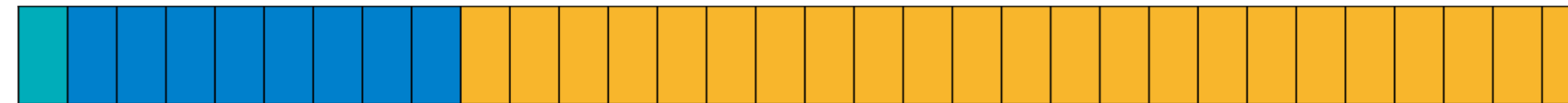
(significant / mantissa)

$$0.265625 = 1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127}$$



Q: How to represent 0?

Floating-point Number: normal vs. subnormal




Sign 8 bit Exponent

23 bit Fraction

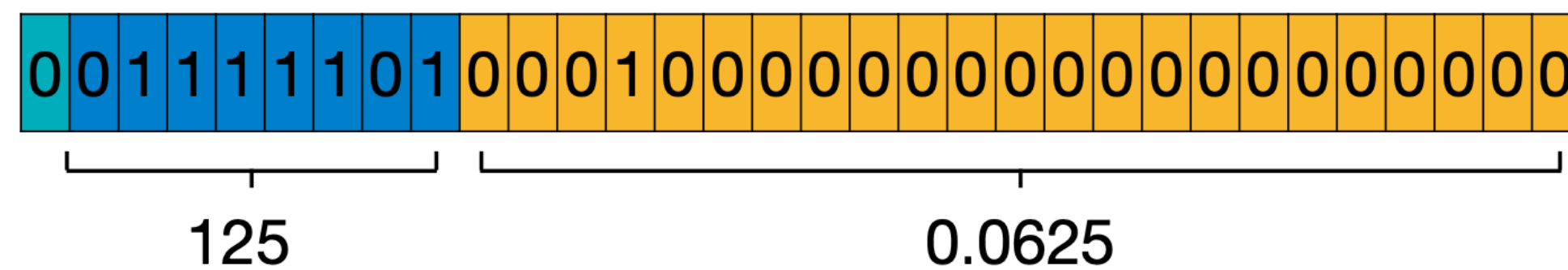
$$(-1)^{\text{sign}} \times (1 + \mathbf{\text{Fraction}}) \times 2^{\mathbf{\text{Exponent}}-127}$$

(Normal Numbers, Exponent≠0)

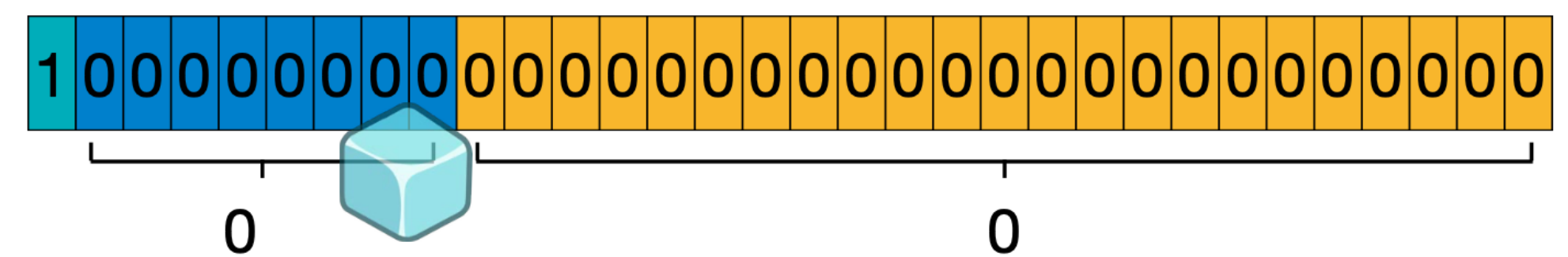
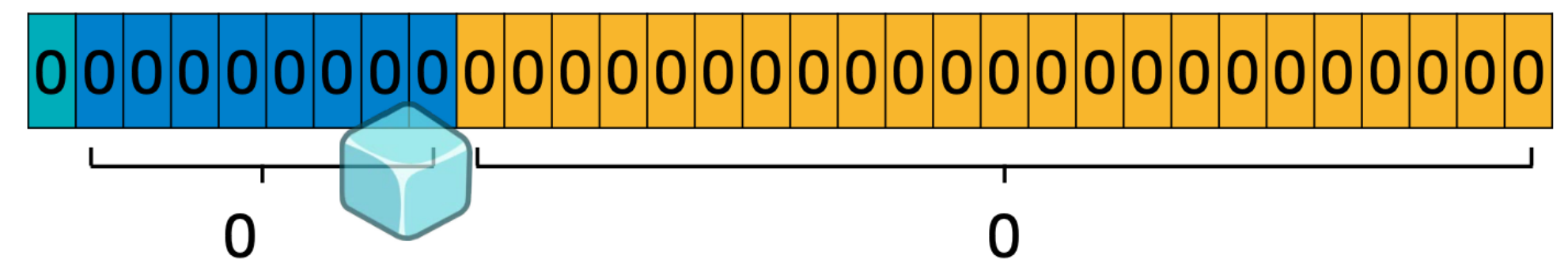
Should have been $(-1)^{\text{sign}} \times (1 + \mathbf{\text{Fraction}}) \times 2^{0-127}$

But we force to be $(-1)^{\text{sign}} \times \mathbf{\text{Fraction}} \times 2^{1-127}$ 

(Subnormal Numbers, Exponent=0)



$$0.265625 = 1.0625 \times 2^{-2} = (1 + 0.0625) \times 2^{125-127}$$



$$0 = 0 \times 2^{-126}$$

Q: What is the representation power of fp32?

What is the minimum positive value?



Sign 8 bit Exponent

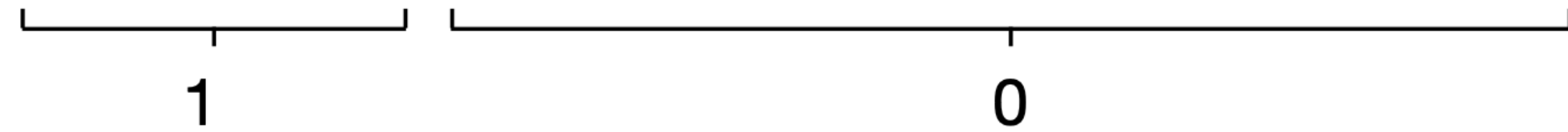
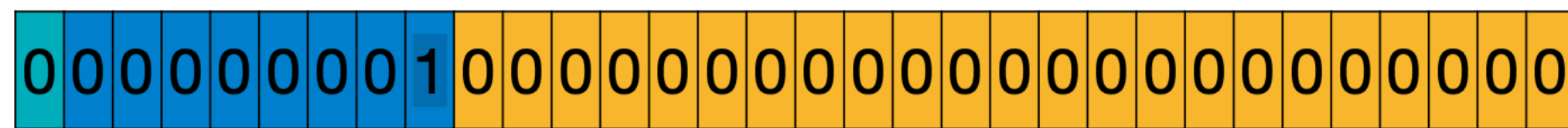
23 bit Fraction

$$(-1)^{\text{sign}} \times (1 + \mathbf{\text{Fraction}}) \times 2^{\mathbf{\text{Exponent}}-127}$$

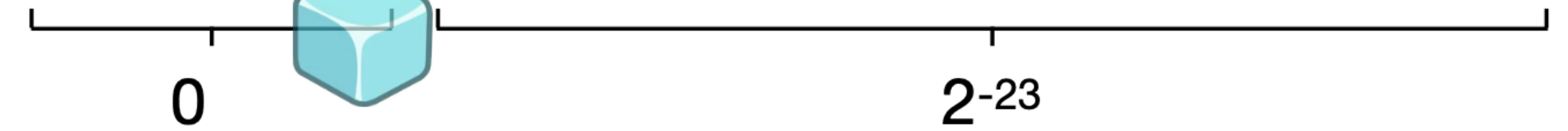
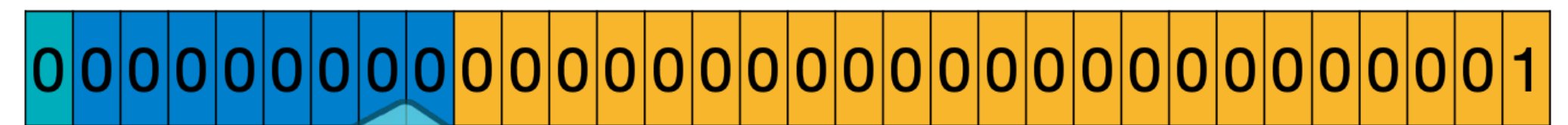
(Normal Numbers, Exponent≠0)

$$(-1)^{\text{sign}} \times \mathbf{\text{Fraction}} \times 2^{1-127}$$

(Subnormal Numbers, Exponent=0)

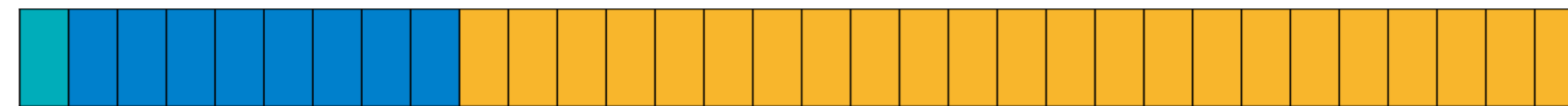


$$2^{-126} = (1 + \underline{0}) \times 2^{1-127}$$



$$2^{-149} = 2^{-23} \times 2^{-126}$$

Some Special Values



Sign 8 bit Exponent

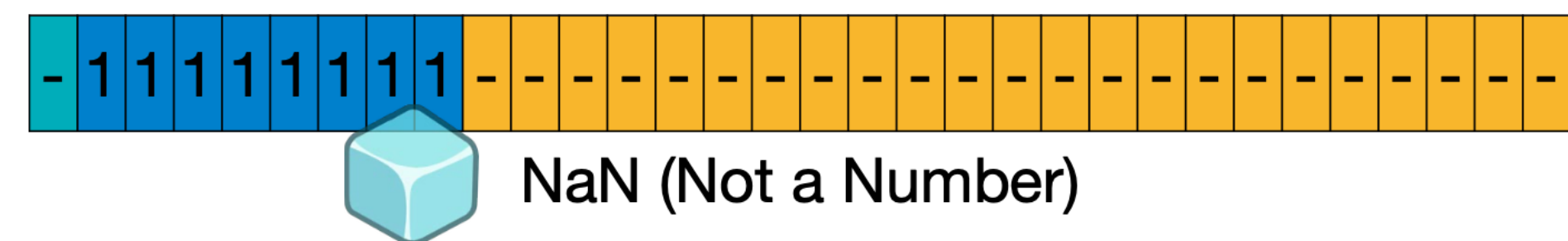
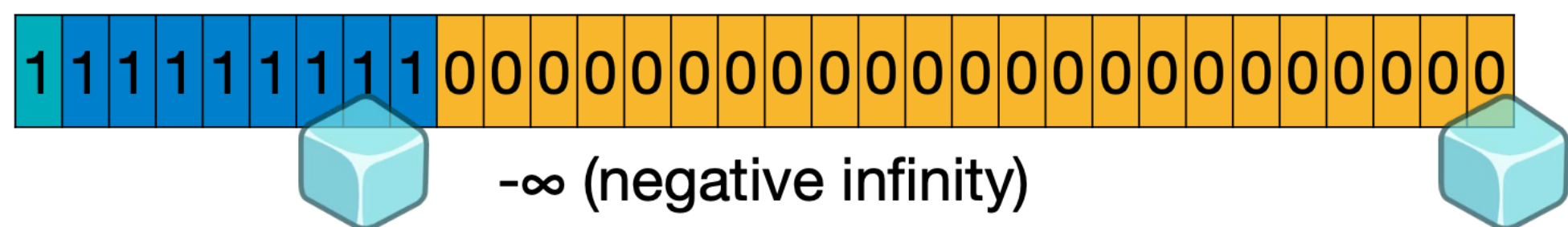
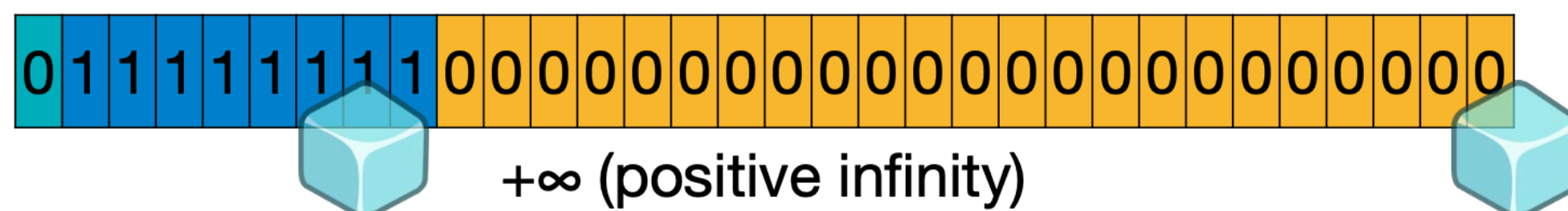
23 bit Fraction

$$(-1)^{\text{sign}} \times (1 + \mathbf{Fraction}) \times 2^{\text{Exponent}-127}$$

(Normal Numbers, Exponent≠0)

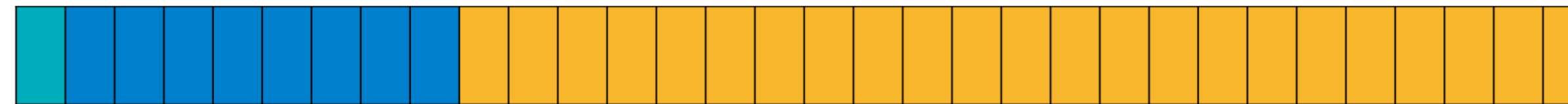
$$(-1)^{\text{sign}} \times \mathbf{Fraction} \times 2^{1-127}$$

(Subnormal Numbers, Exponent=0)






much waste. Revisit in fp8.

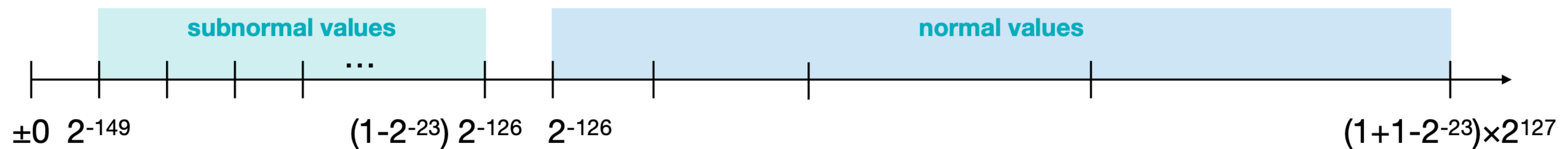
Summary of fp32



Sign 8 bit Exponent

23 bit Fraction

Exponent	Fraction=0	Fraction≠0	Equation
00 _H = 0 	±0	subnormal	$(-1)^{\text{sign}} \times \mathbf{Fraction} \times 2^{1-127}$
01 _H ... FE _H = 1 ... 254	normal		$(-1)^{\text{sign}} \times (1 + \mathbf{Fraction}) \times 2^{\text{Exponent}-127}$
FF _H = 255 	±INF 	NaN	

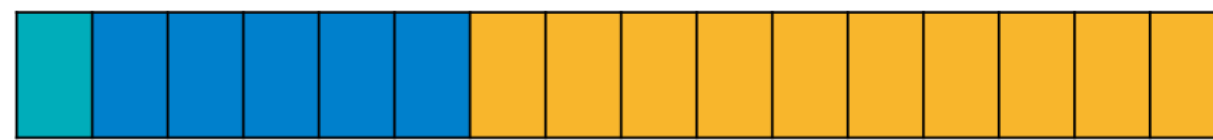


Next Lecture

[IEEE 754](#) Single Precision 32-bit Float (IEEE FP32)



[IEEE 754](#) Half Precision 16-bit Float (IEEE FP16)



[Google Brain Float \(BF16\)](#)



Exponent (bits)	Fraction (bits)	Total (bits)
8	23	32
5	10	16
8	7	16