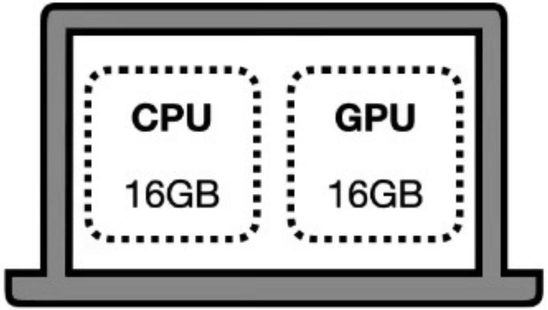
QLoRA

LLM fine-tuning made accessible

The Problem

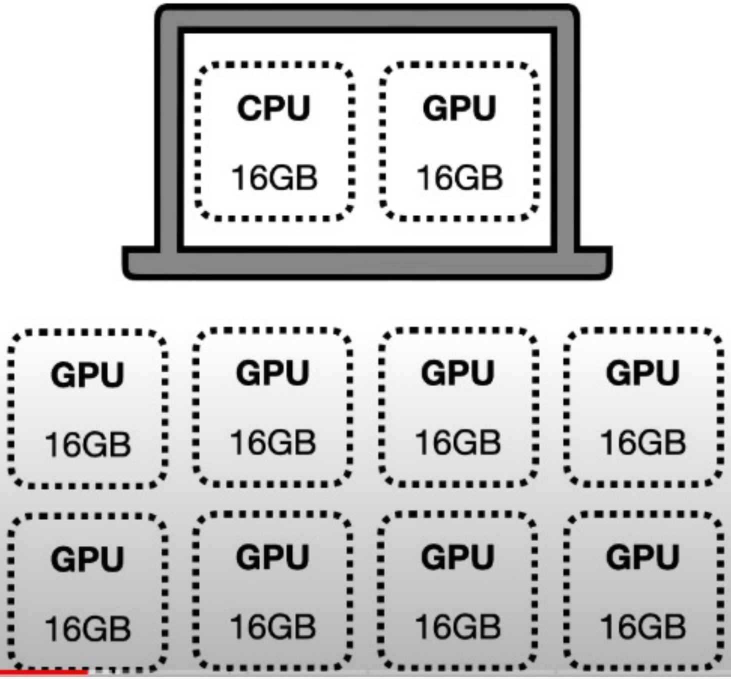
LLMs are (computationally) expensive

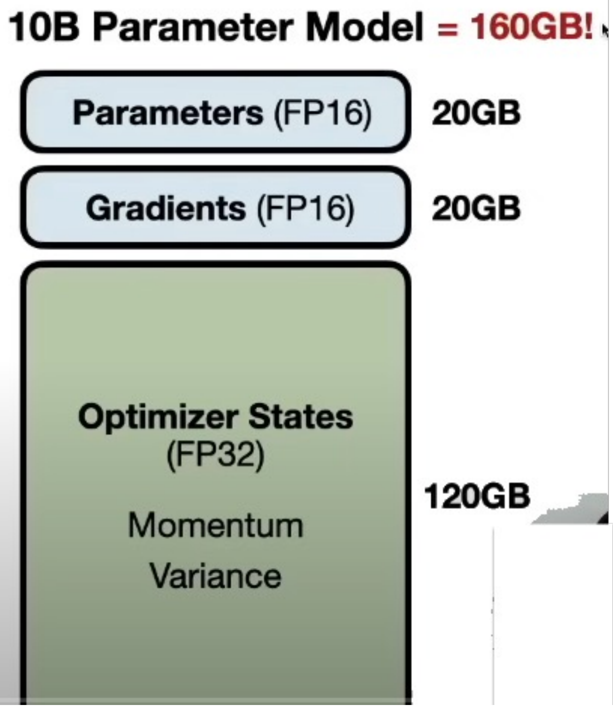


10B Parameter Model

The Problem

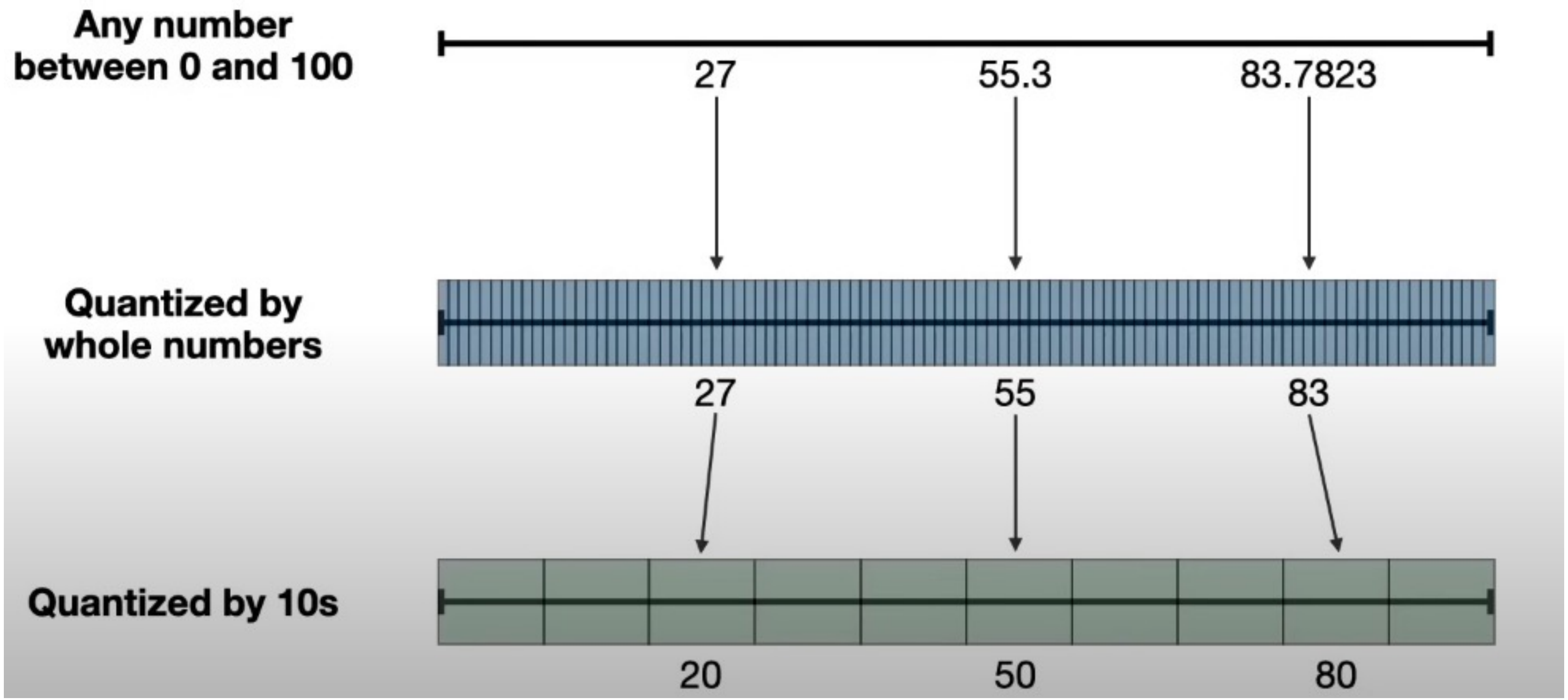
LLMs are (computationally) expensive





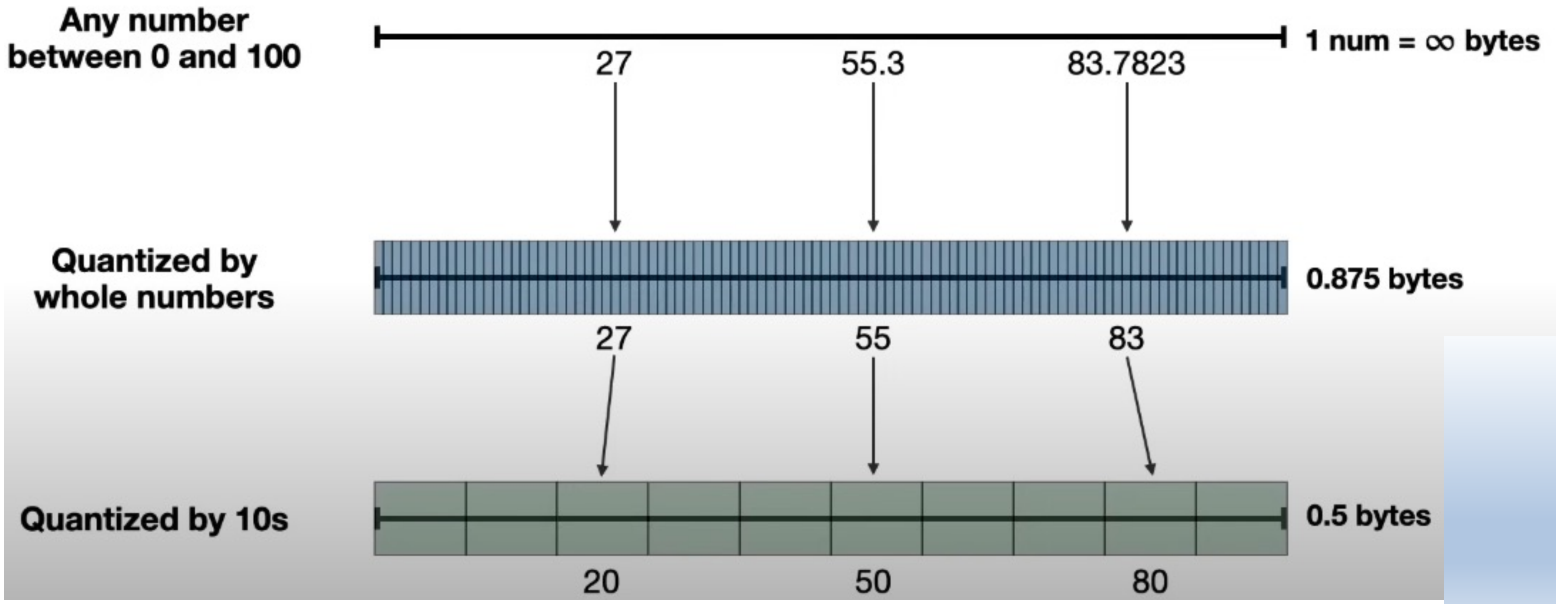
What is Quantization?

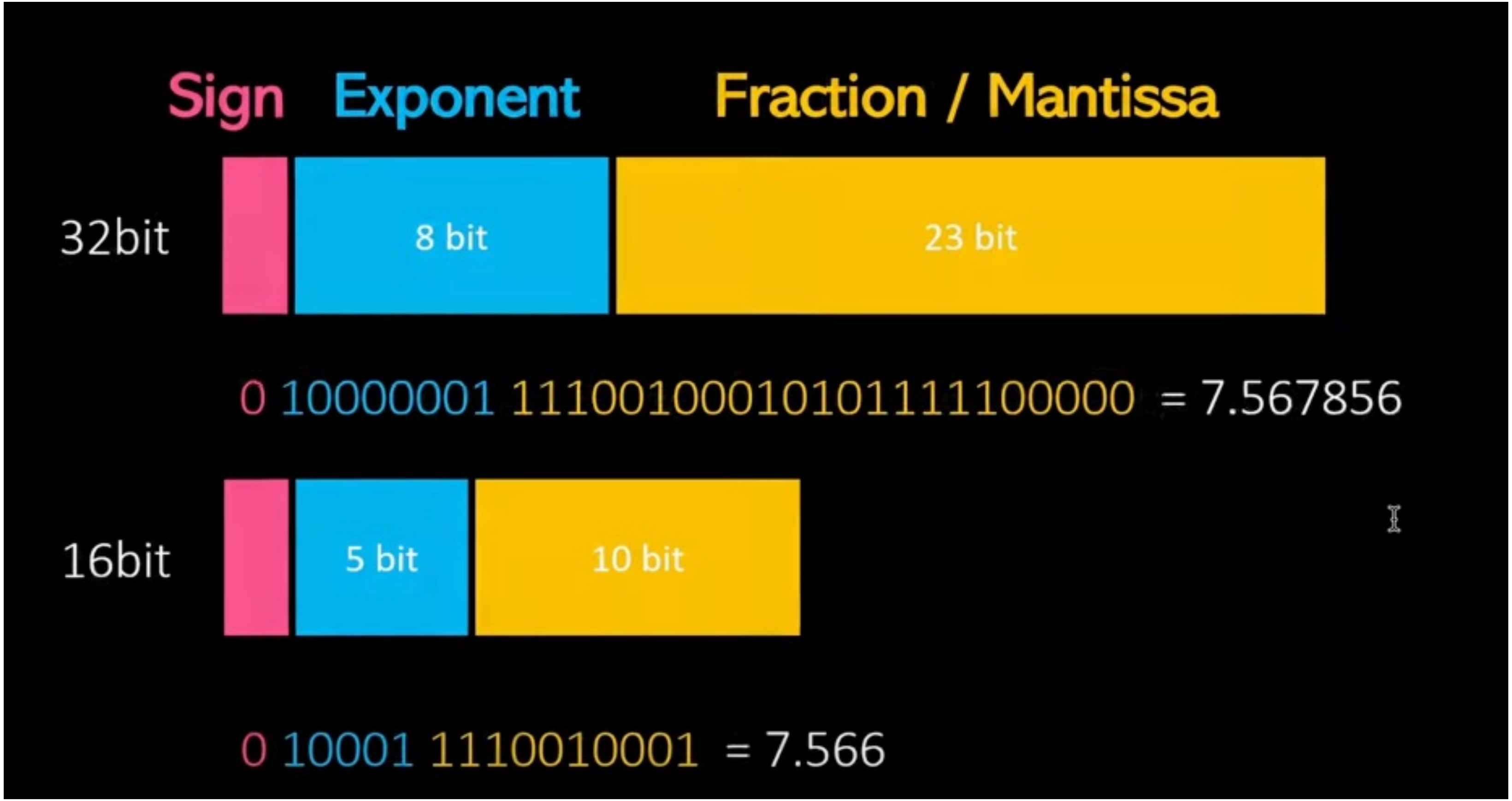
Quantization = splitting range into buckets



What is Quantization?

Quantization = splitting range into buckets




Sign
Exponent
Fraction / Mantissa
32bit
8 bit
23 bit
0 10000001 11100100010101111100000 = 7.567856
16bit
5 bit
10 bit
0 10001 1110010001 = 7.566

```
tensor([[ 0.0031, -0.0438,  0.0494,  ..., -0.0046, -0.0410,  0.0436],
        [-0.1013,  0.0394,  0.0787,  ...,  0.0986,  0.0595,  0.0162],
        [-0.0859, -0.1227, -0.1209,  ...,  0.1158,  0.0186, -0.0530],
        ...,
        [ 0.0804,  0.0725,  0.0638,  ..., -0.0487, -0.0524, -0.1076],
        [-0.0200, -0.0406,  0.0663,  ...,  0.0123,  0.0551, -0.0121],
        [-0.0041,  0.0865, -0.0013,  ..., -0.0427, -0.0764,  0.1189]],
       dtype=torch.float16)
```

```
tensor([[   3,  -47,   54,  ...,   -5,  -44,   47],
        [-104,   40,   81,  ...,  101,   61,   17],
        [ -89, -127, -125,  ...,  120,   19,  -55],
        ...,
        [  82,   74,   65,  ...,  -49,  -53, -109],
        [ -21,  -42,   68,  ...,   13,   57,  -12],
        [  -4,   88,   -1,  ...,  -43,  -78,  121]],
       device='cuda:0', dtype=torch.int8, requires_grad=True)
```

8bit-Quantization

Ingredient 1: 4-bit NormalFloat

A better way to bucket numbers

4-bit e.g. 0101

$\implies 2^4 = 16$ unique combinations

$\implies 16$ buckets for quantizations

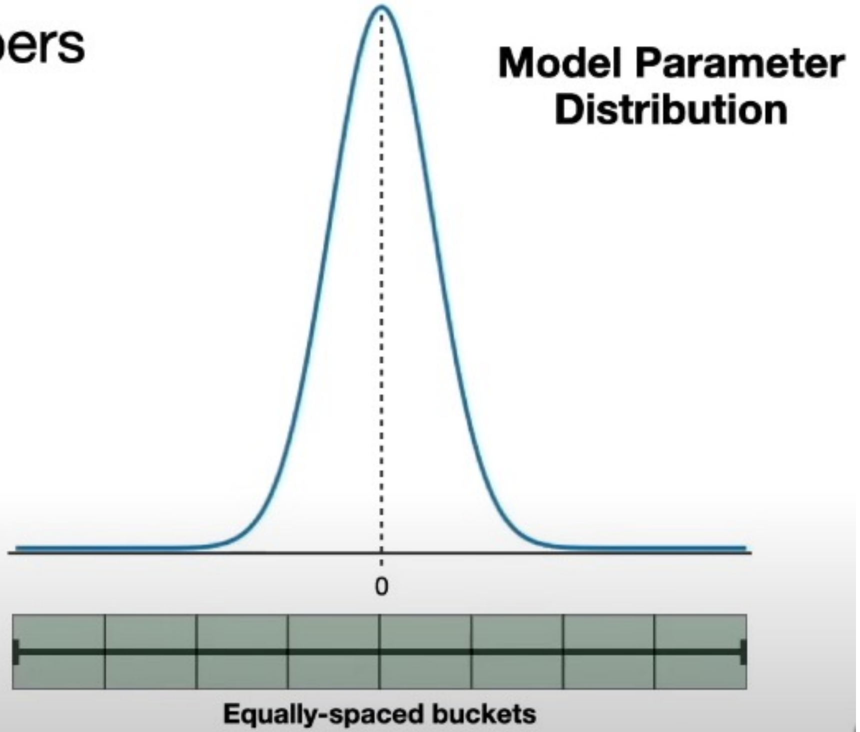
Ingredient 1: 4-bit NormalFloat

A better way to bucket numbers

4-bit e.g. 0101

$\Longrightarrow$ $2^4 = 16$ unique combinations

$\Longrightarrow$ 16 buckets for quantizations



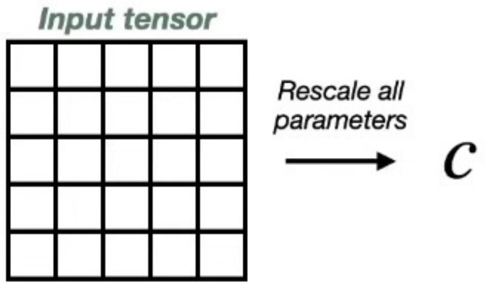
Ingredient 2: Double Quantization

Quantizing the Quantization Constants

$$X^{\mathrm{Int8}} = \mathrm{round}\left(\frac{127}{\mathrm{absmax}(X^{\mathrm{FP32}})} X^{\mathrm{FP32}}\right)$$

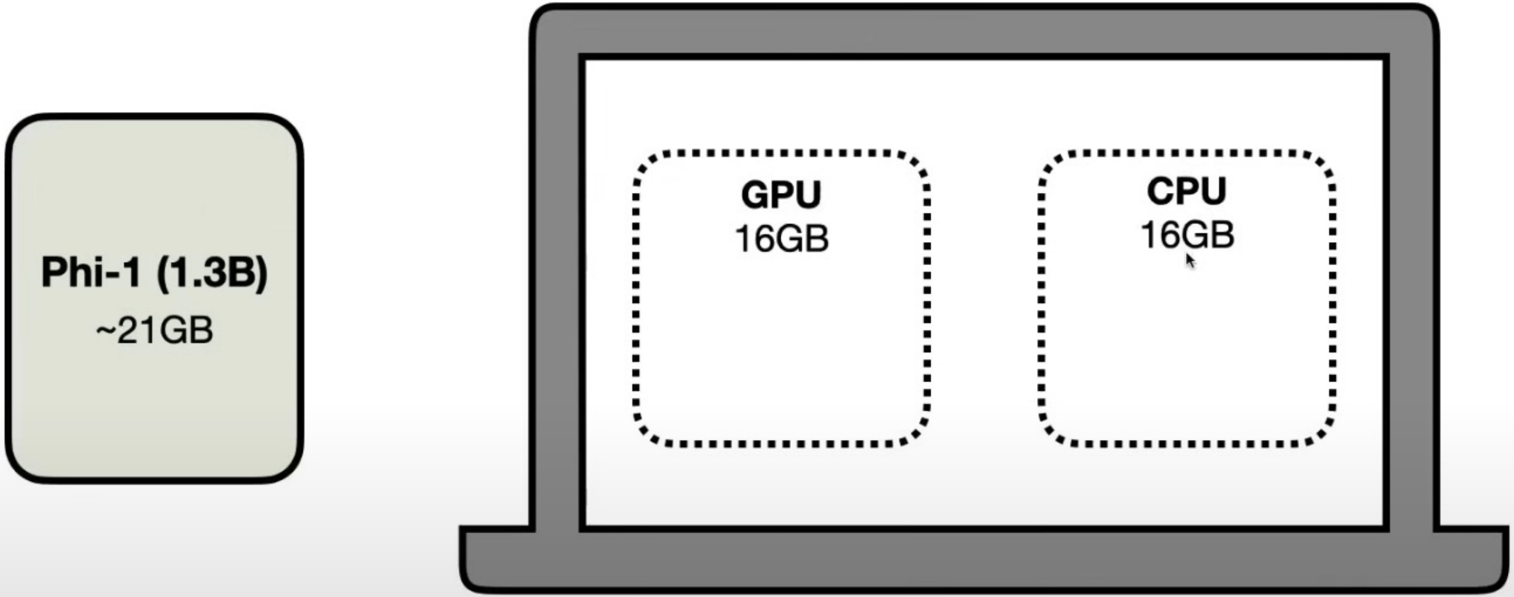
$$= \mathrm{round}\left(c^{\mathrm{FP32}} \cdot X^{\mathrm{FP32}}\right)$$

Takes up precious memory



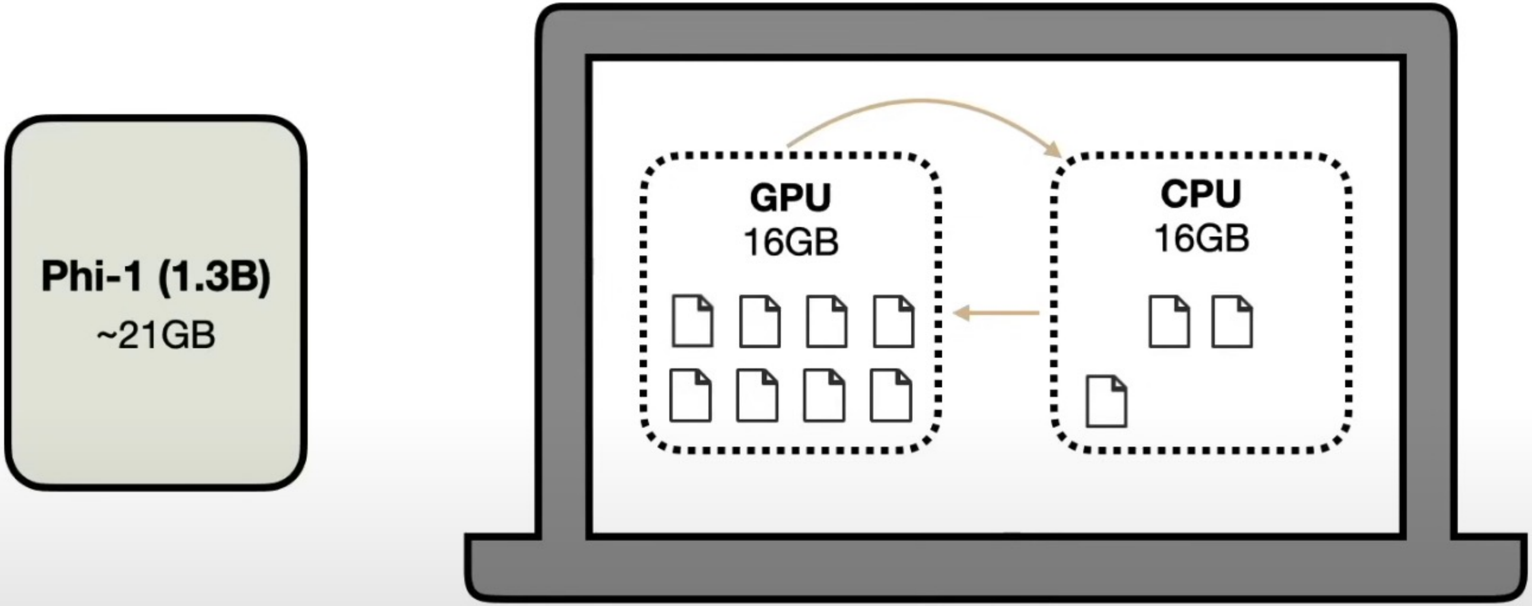
Ingredient 3: Paged Optimizer

Looping in your CPU



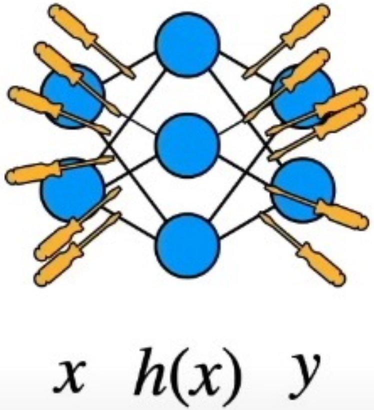
Ingredient 3: Paged Optimizer

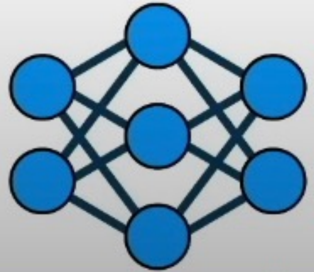
Looping in your CPU



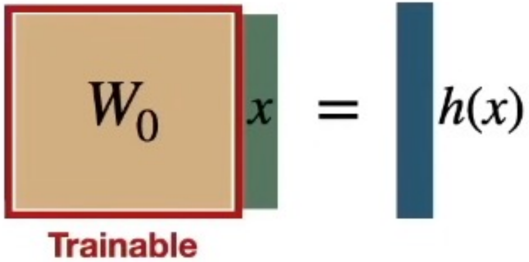
Ingredient 4: LoRA

Fine-tunes model by adding **small set** of trainable parameters



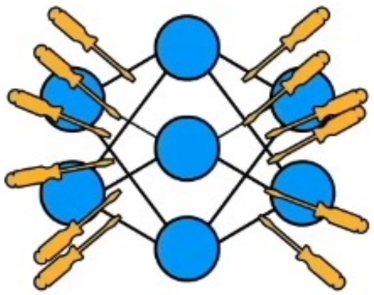


Full Fine-tuning: $h(x) = W_0 x$

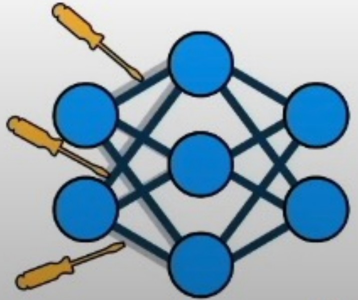


Ingredient 4: LoRA

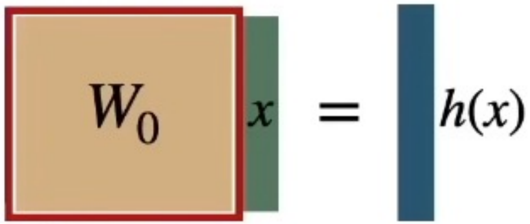
Fine-tunes model by adding **small set** of trainable parameters



$x \quad h(x) \quad y$



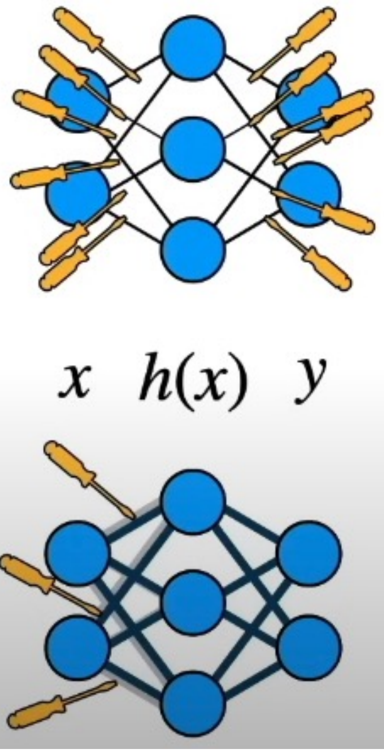
Full Fine-tuning: $h(x) = W_0 x$



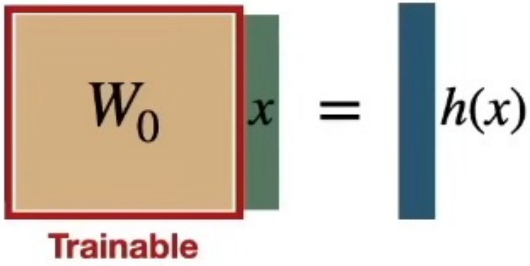
LoRA: $h(x) = W_0 x + \Delta W x = W_0 x + BAx$

Ingredient 4: LoRA

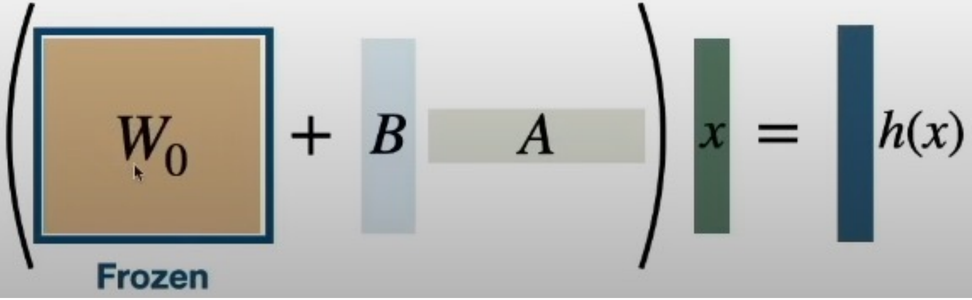
Fine-tunes model by adding **small set** of trainable parameters



Full Fine-tuning: $h(x) = W_0 x$



LoRA: $h(x) = W_0 x + \Delta W x = W_0 x + BAx$



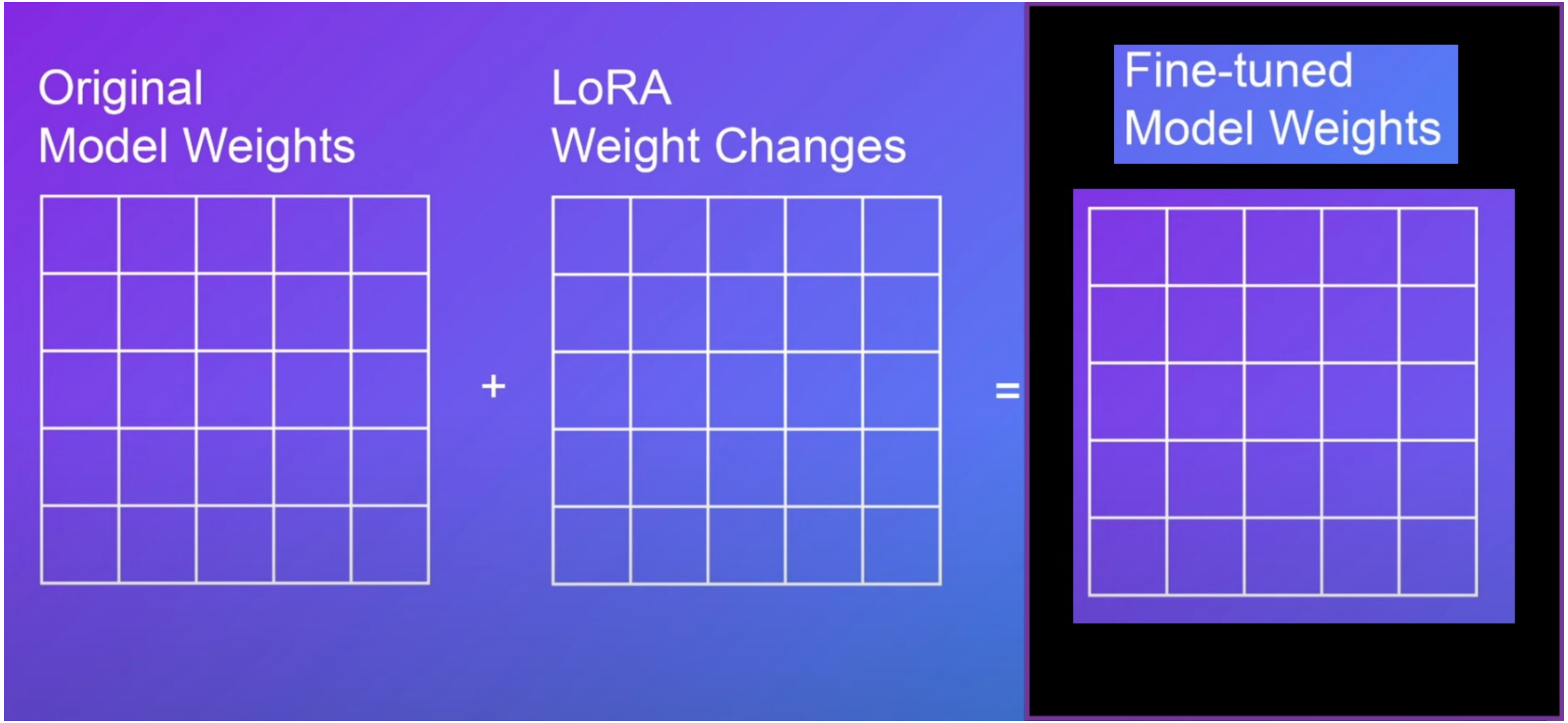
$B \in \mathbb{R}^{dxr}$

$A \in \mathbb{R}^{rxk}$

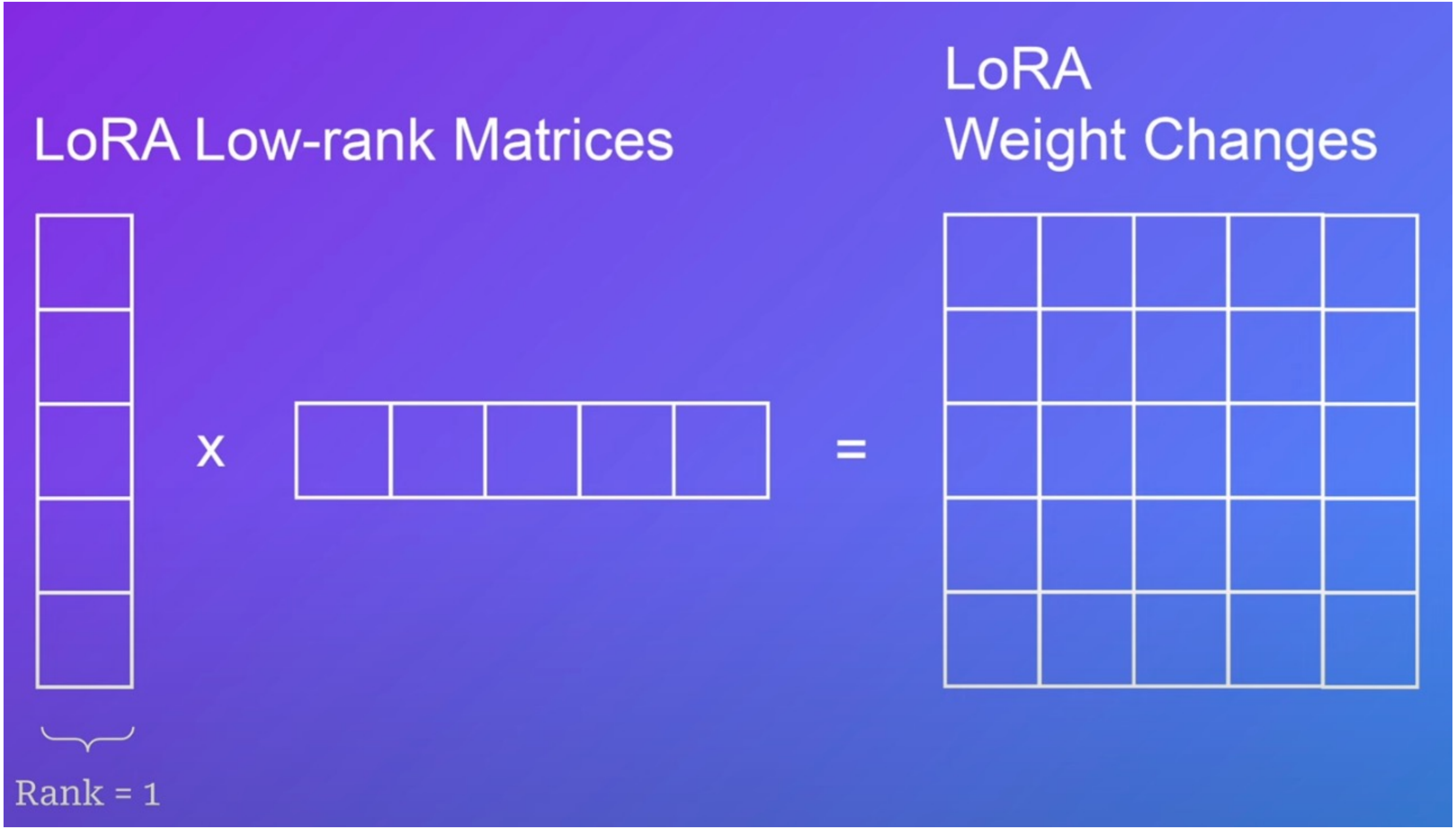
$rank\ r \ll \min(d, k)$

$$W_0 + \Delta W = W_0 + BA$$

Large Model



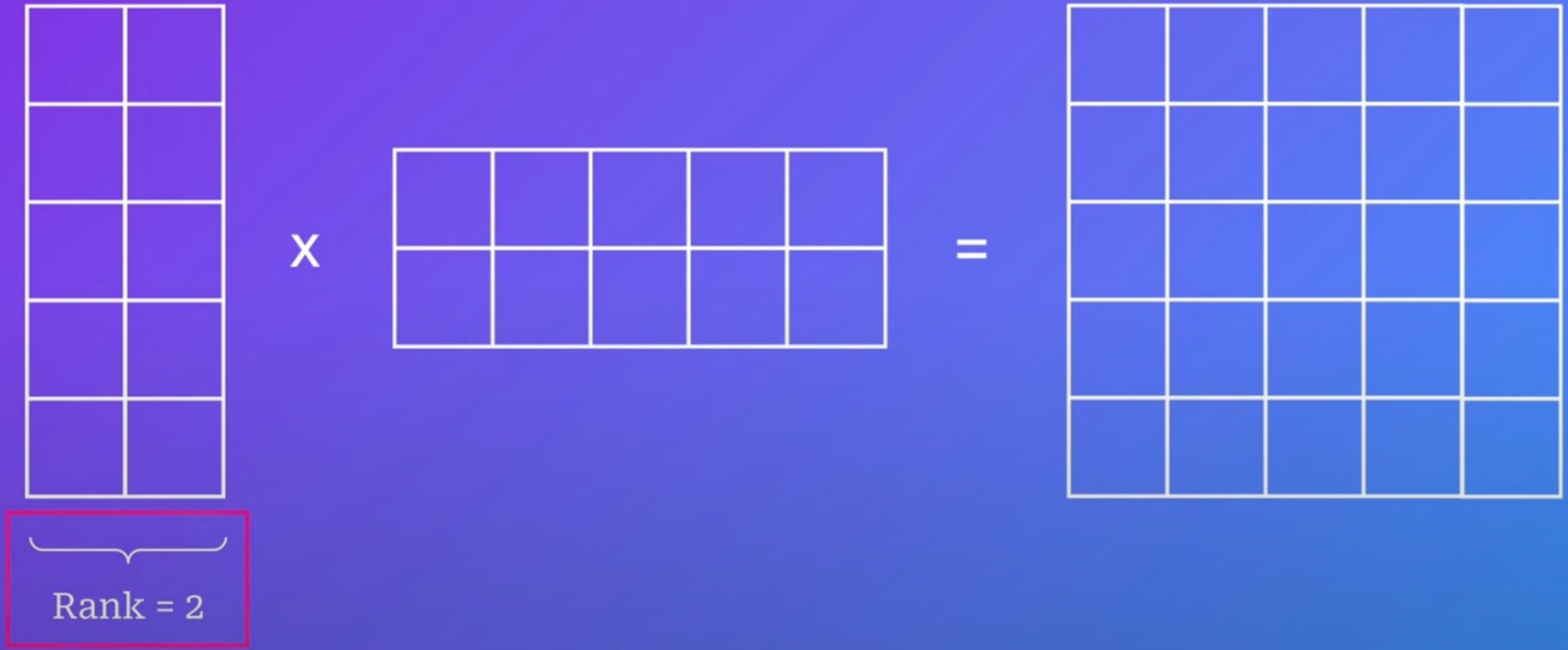
Original
Model Weights
+
LoRA
Weight Changes
=
Fine-tuned
Model Weights


LoRA Low-rank Matrices
x
=
LoRA
Weight Changes
Rank = 1

Increasing Precision by Increasing Rank

LoRA Matrices, Rank 2

Higher Precision Weight Changes



0.15	-0.14	-0.21	0.612
-0.22	0.204	0.308	-0.86
-0.30	-0.16	0.634	0.147
-0.07	-0.2	0.246	0.523

ΔW

Shape: (200, 200)

0.3	-0.14
-0.42	0.201
0.46	0.38
0.5	0.14

B

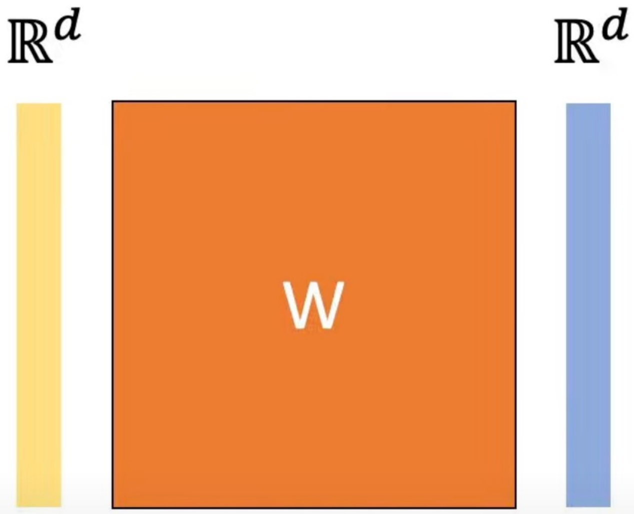
Shape: (200, 2)

0.1	-0.44	0.04	1.42
-0.92	0.1	1.62	-1.33

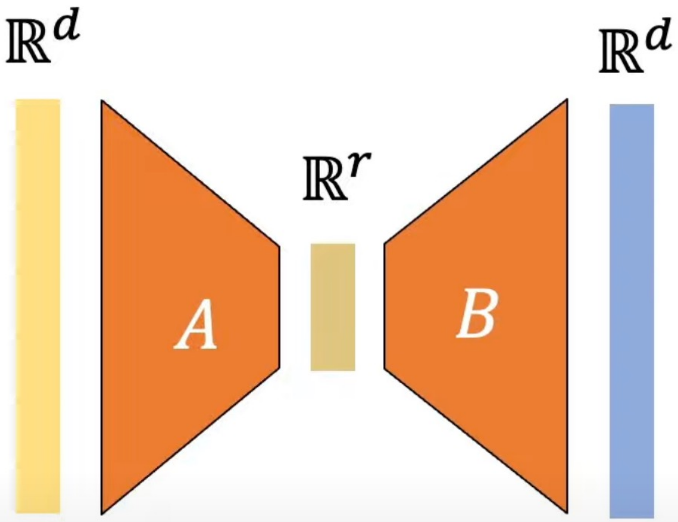
A

Shape: (2, 200)

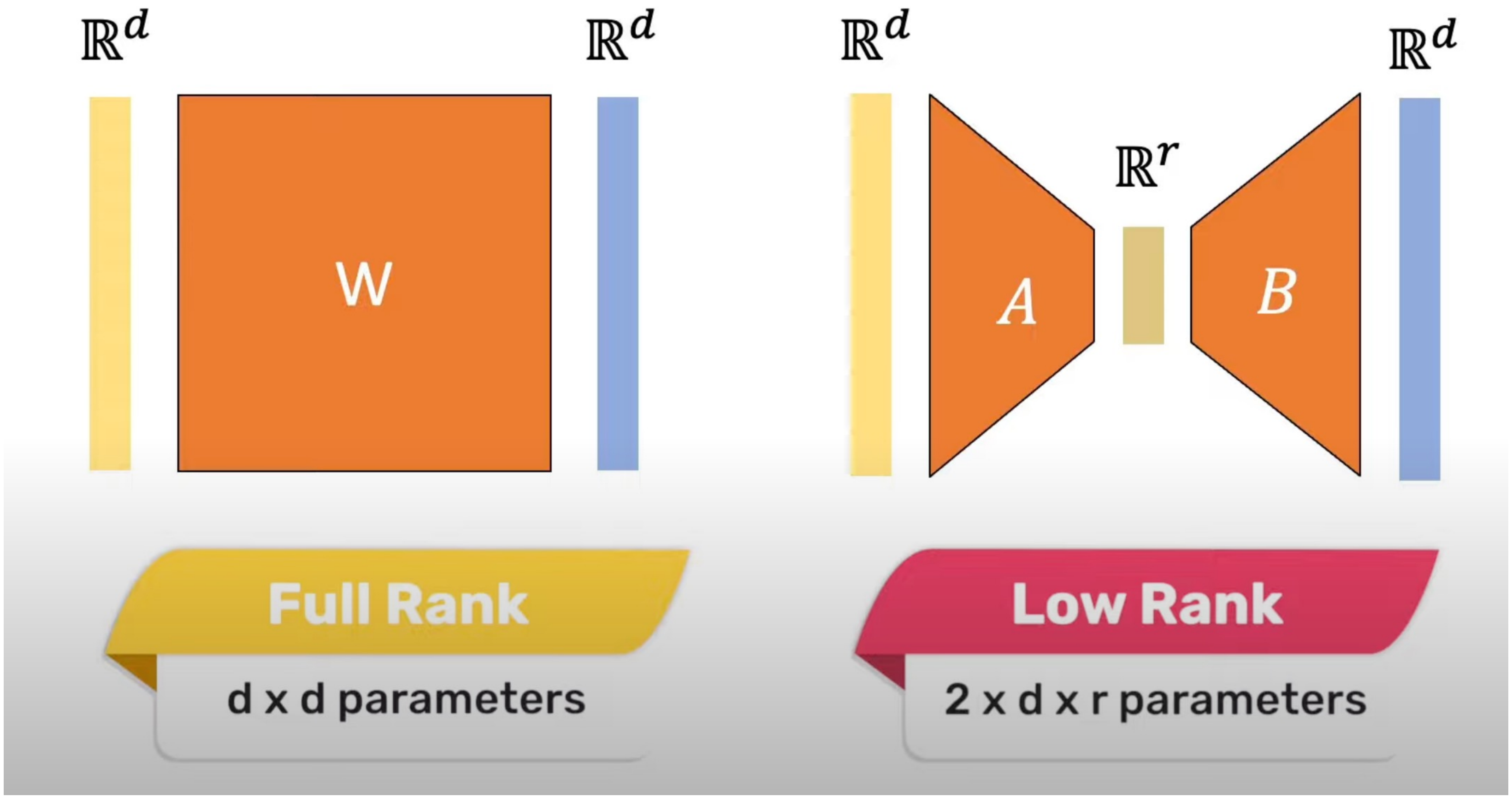
```python
def regular_forward_matmul(x, W):
    h = x @ W
return h

def lora_forward_matmul(x, W, W_A, W_B):
    h = x @ W  # regular matrix multiplication
    h += x @ (W_A @ W_B)*alpha # use scaled LoRA weights
return h
```



Full rank (rank = d)



Low rank (rank = r << d)

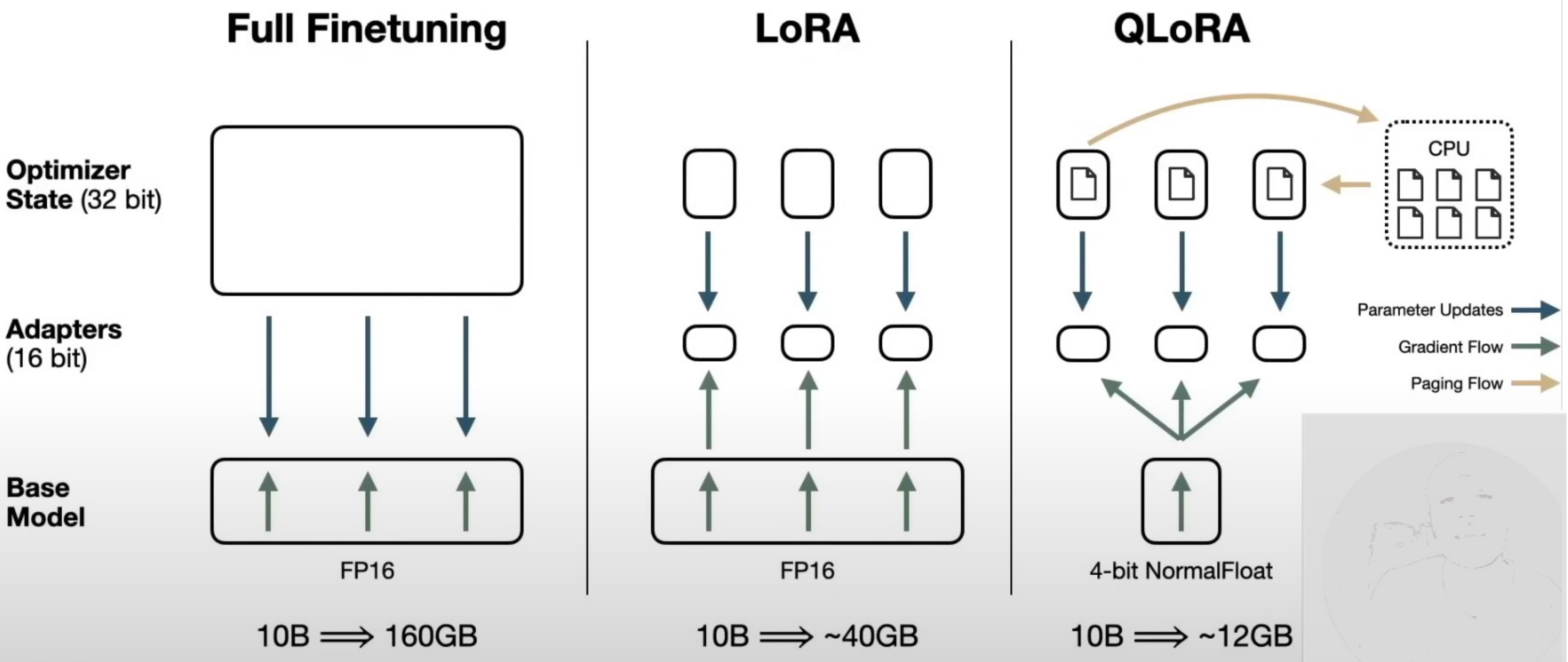

$\mathbb{R}^d$
W
$\mathbb{R}^d$
$\mathbb{R}^d$
A
$\mathbb{R}^r$
B
$\mathbb{R}^d$
Full Rank
d x d parameters
Low Rank
2 x d x r parameters

Number of Trainable Parameters

Rank	7B	13B	70B	180B
1	167k	228k	529k	849k
2	334k	456k	1M	2M
8	1M	2M	4M	7M
16	3M	4M	8M	14M
512	86M	117M	270M	434M
1,024	171M	233M	542M	869M
8,192	1.4B	1.8B	4.3B	7.0B

In reality, LLMs are made up of multiple layers of differing sizes. This is a generalization as if the model were a single layer.

Bringing it all together



 [2] [3]

```python
from peft import LoraConfig, get_peft_model

config = LoraConfig(
          r=16, # attention heads
          lora_alpha=32, # alpha scaling
          target_modules=["query_key_value"], # gathered from print(model)
          lora_dropout=0.05,
          bias="none",
          task_type="CAUSAL_LM" # set this for CLM or Seq2Seq
          )

model = get_peft_model(model, config)
```

```python
print(model)
```

```
BloomForCausalLM(
  (transformer): BloomModel(
    (word_embeddings): Embedding(250880, 1024)
    (word_embeddings_layernorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (h): ModuleList(
      (0-23): 24 x BloomBlock(
        (input_layernorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (self_attention): BloomAttention(
          (query_key_value): Linear(in_features=1024, out_features=3072, bias=True)
          (dense): Linear(in_features=1024, out_features=1024, bias=True)
          (attention_dropout): Dropout(p=0.0, inplace=False)
        )
        (post_attention_layernorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (mlp): BloomMLP(
          (dense_h_to_4h): Linear(in_features=1024, out_features=4096, bias=True)
          (gelu_impl): BloomGelu()
          (dense_4h_to_h): Linear(in_features=4096, out_features=1024, bias=True)
        )
      )
    )
    (ln_f): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=1024, out_features=250880, bias=False)
)
```

PEFT

```python
from transformers import AutoModelForSeq2SeqLM
from peft import get_peft_config, get_peft_model, LoraConfig, TaskType
model_name_or_path = "bigscience/mt0-large"
tokenizer_name_or_path = "bigscience/mt0-large"

peft_config = LoraConfig(
    task_type=TaskType.SEQ_2_SEQ_LM, inference_mode=False, r=8, lora_alpha=32, lora_dropout=0.1
)

model = AutoModelForSeq2SeqLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()
# output: trainable params: 2359296 || all params: 1231940608 || trainable%: 0.19151053100118282
```