End to End: Part 5

Earl Wong

Subjects

- Optics
- Sensor
- **ISP**
- GPU
- **NPU**

Different Naming Conventions in Mobile Space

- ANE (Apple) - Apple Neural Engine
- AI Engine (Qualcomm) - Hexagonal Tensor Processor
- APU (Mediatek) - AI Processing Unit
- NPU (Samsung) - Neural Processing Unit
- TPU (Google) - Tensor Processing Unit

Objective In Mobile Space

- Deliver fast, low latency ML inference under strict power and thermal constraints.

CPU vs GPU vs NPU

- CPU's are built for doing different types of work / general tasks = swiss army knife.
- The CPU runs the operating system.
- The CPU excels at control flow, because the CPU is designed for branching, prediction, and task management.
- The CPU excels at anything with unpredictable logic.

CPU vs GPU vs NPU

- GPU's originated, because of the need for faster graphics.
- In the process, GPU's sacrificed some flexibility, relative to their CPU counterparts.
- By construction, GPU's are graphic engines.
- By construction, GPU's exploit massive parallelism, and rely heavily on latency hiding.
- Since image processing tasks are often a subset of graphics tasks, they can fit nicely into the GPU architecture.

CPU vs GPU vs NPU

- NPU's originated because of the need for fast and power efficient neural network inference.
- By construction, NPU's are optimized for basic, fundamental neural network computations (multiply and accumulate (MAC), without using latency hiding.
- To achieve high performance, nearly all flexibility is sacrificed.
- i.e. No branching, etc.
- Note: A CPU or a GPU could perform the same tasks as an NPU, but only slower and / or with more power usage.

Strengths; Weakness

- CPU - swiss army knife, control flow, highly flexible; not optimized for GPU or NPU tasks
- GPU - massive parallelism; limited flexibility, high power
- NPU - dedicated, low power processor for fast, ML inference; minimal flexibility = one trick pony, but a highly optimized one

NPU

1) Dedicated and highly optimized ML operations

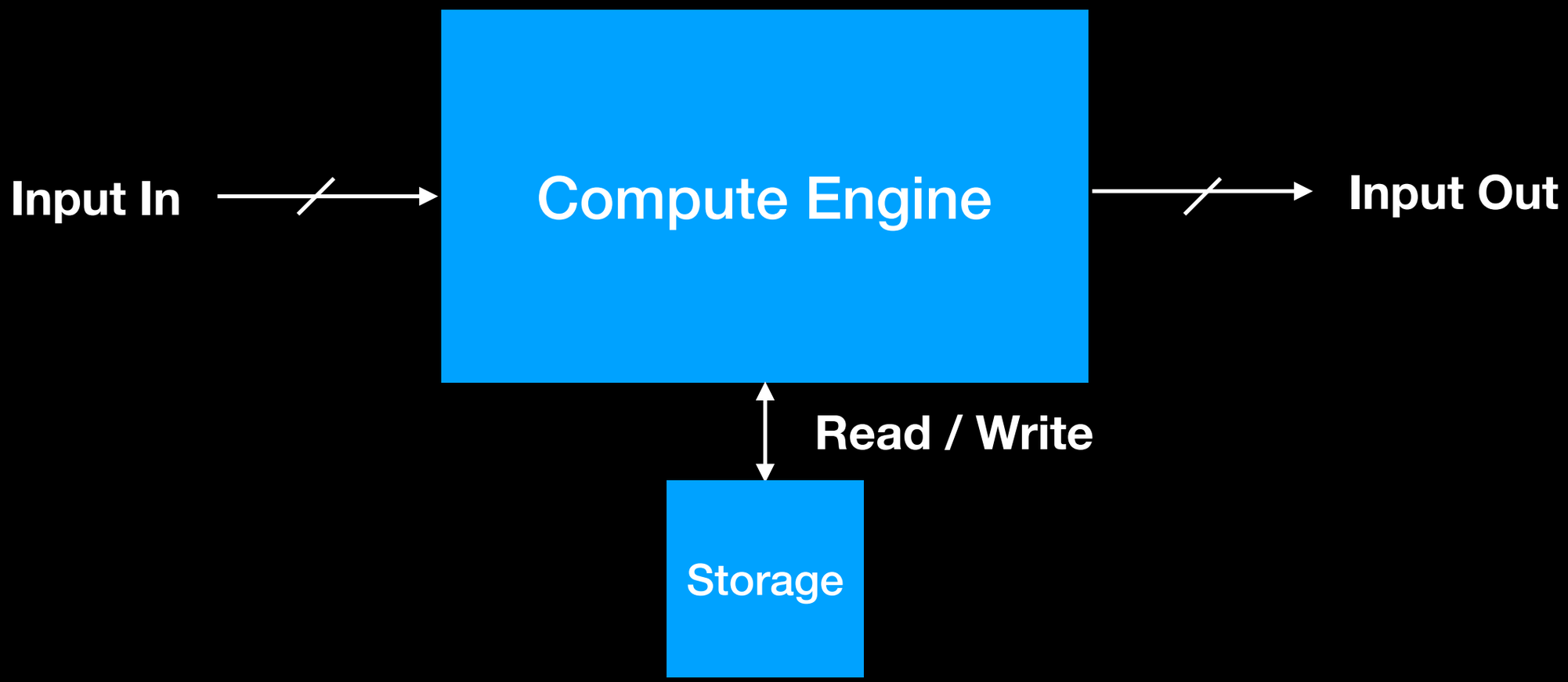
- Convolution (CNN’s): MAC
- Matrix Multiplication (MLP’s): MAC
- Activation (Non Linear Functionality): Comparator (for ReLU)

2) Minimize DRAM fetch; no latency hiding

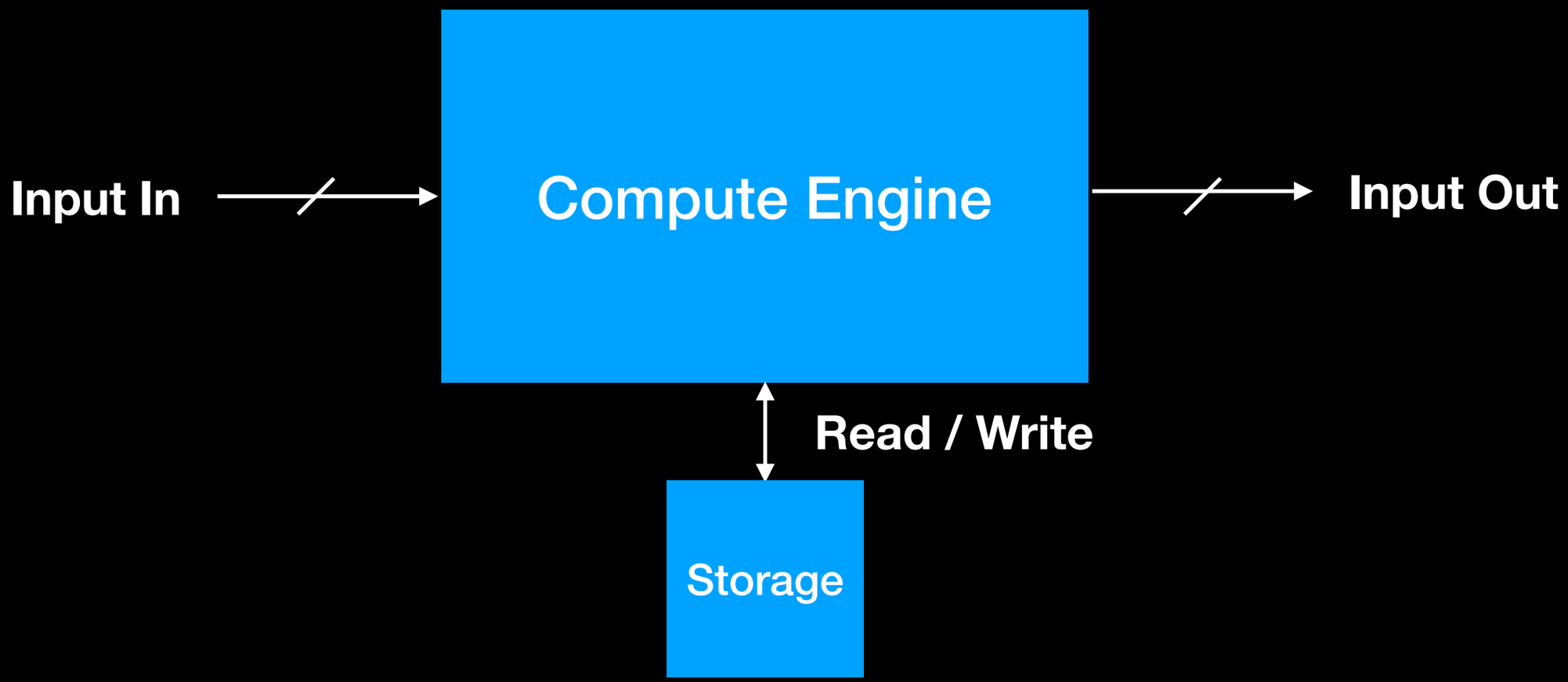
Data Fetch

- CPU’s and GPU’s utilize frequent data fetches from caches and DRAM.
- Caches misses and DRAM fetches are expensive, increasing power and bandwidth usage.
- NPU’s try to maximize data re-use, by streaming data through a fixed pipeline (=dataflow is fixed), minimizing external data fetches.
- Neural network weights are typically stored in the NPU SRAM.

High Level Evolution

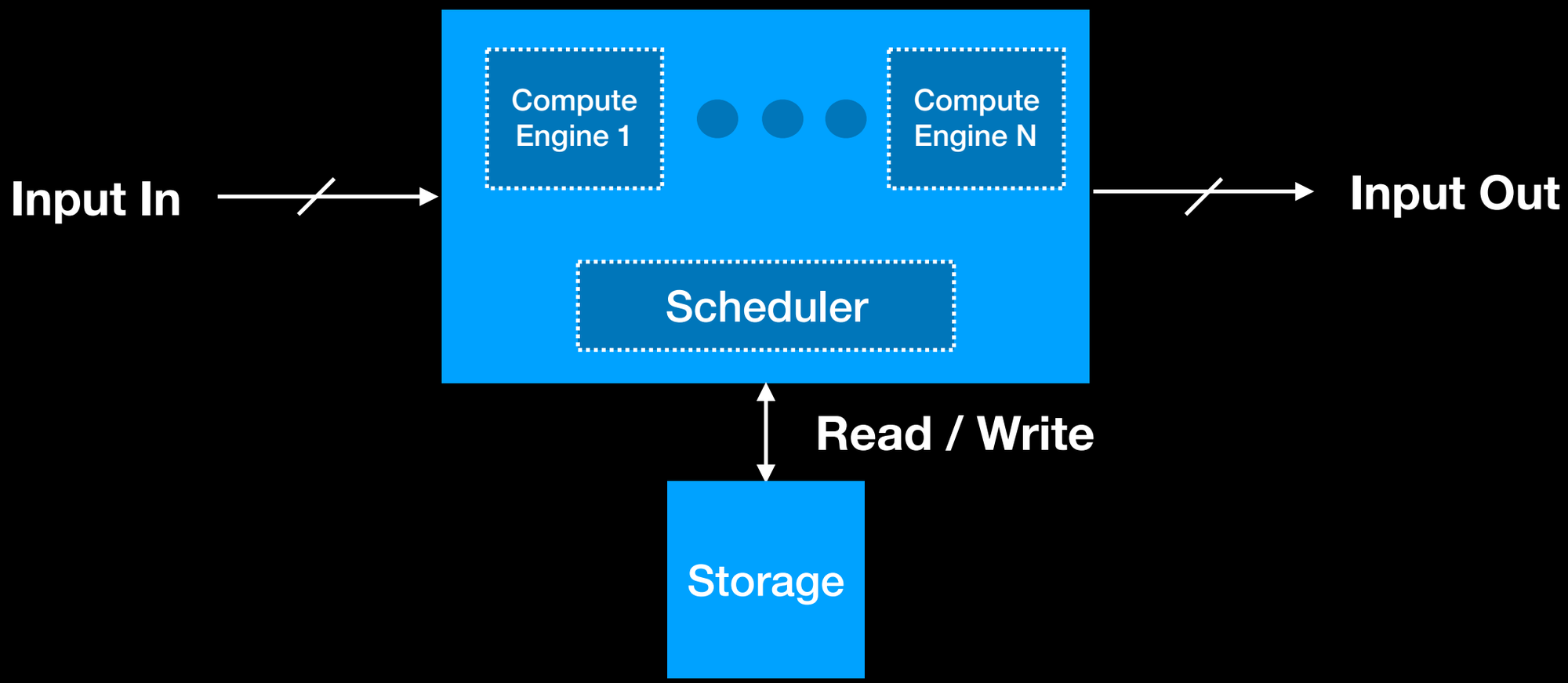


High Level Evolution

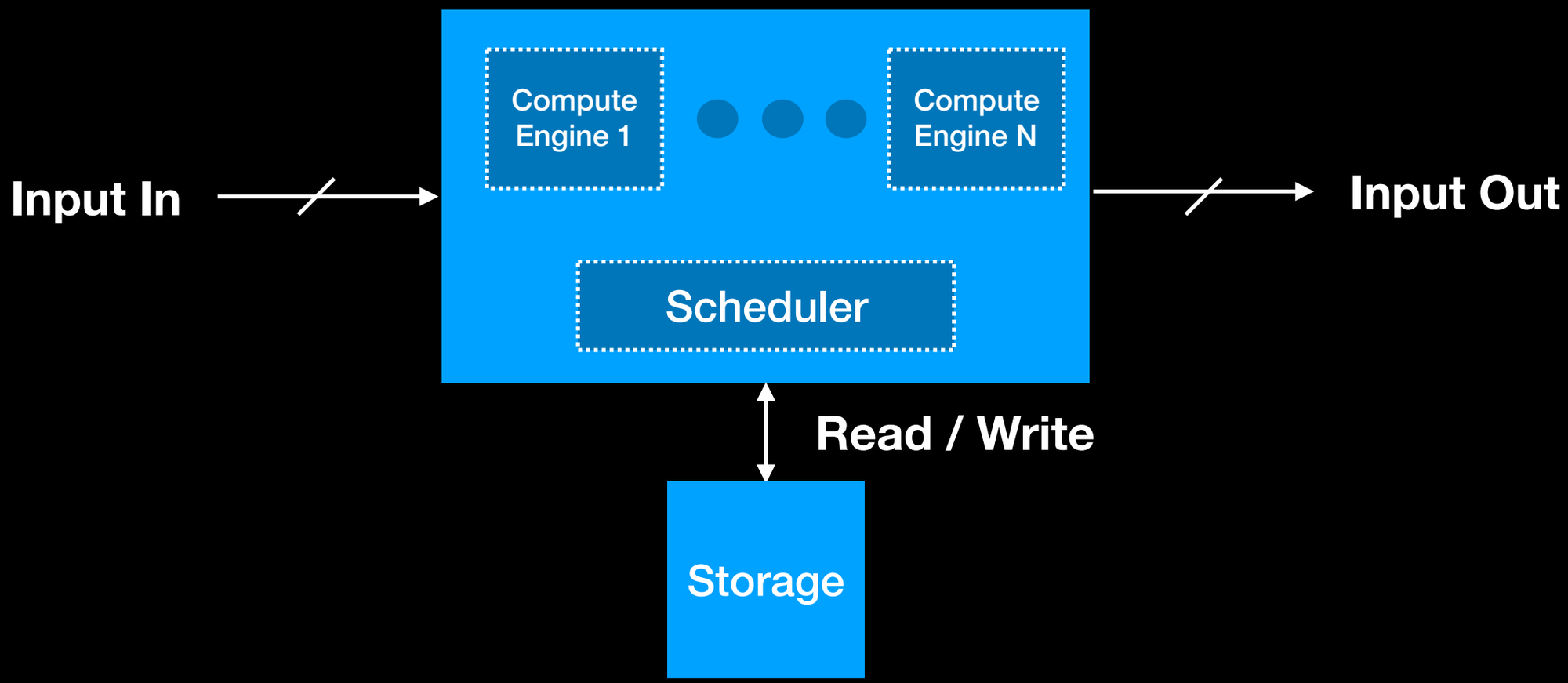


“I want more throughput!”

High Level Evolution

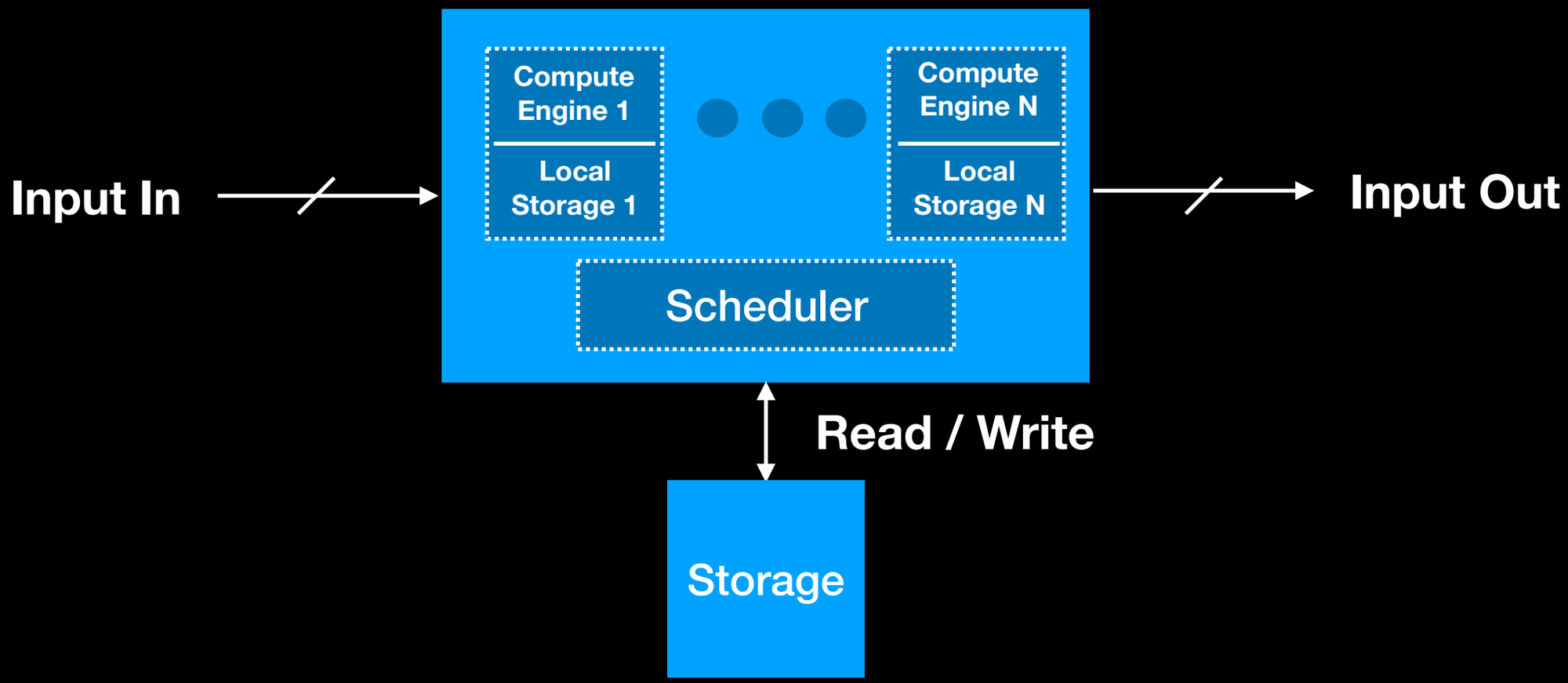


High Level Evolution

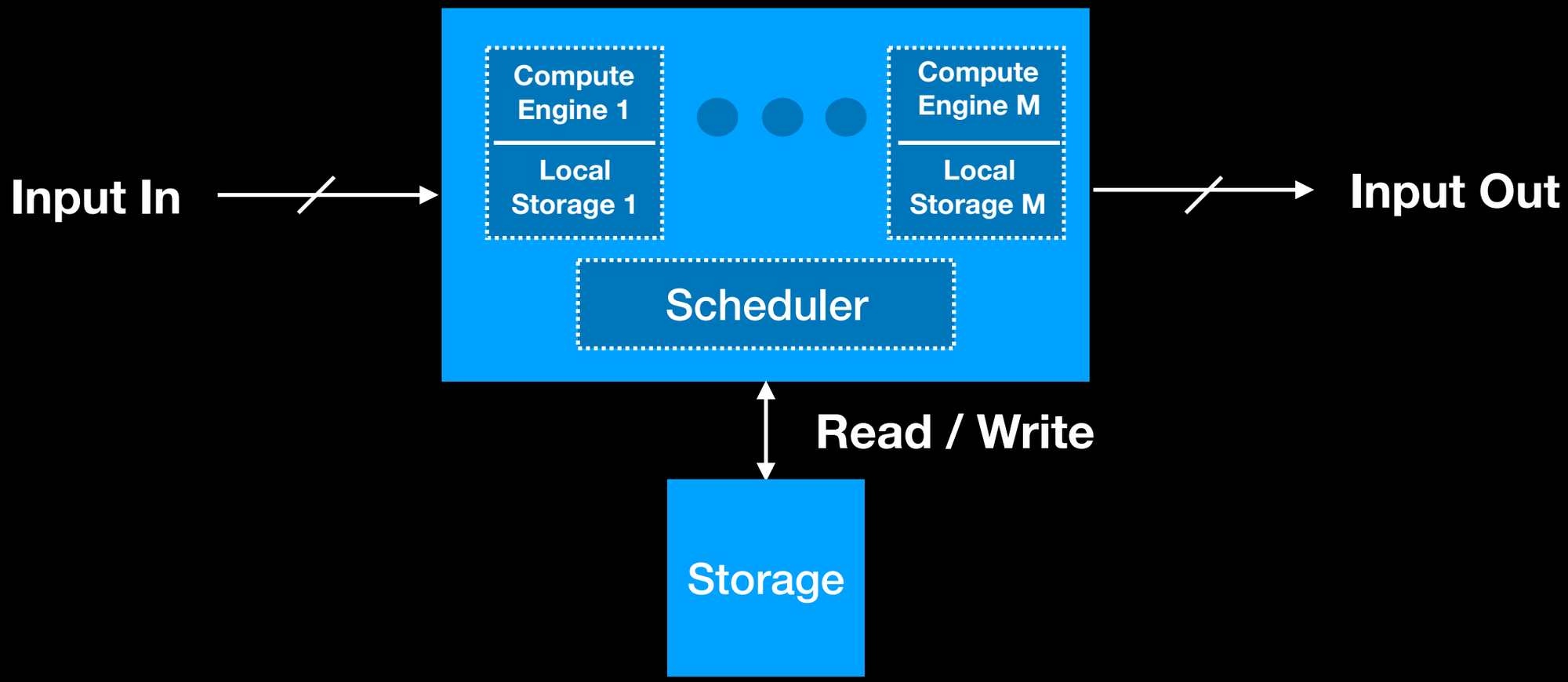


“The compute is under utilized, because we have to read / write from slow, external storage.”

High Level Evolution

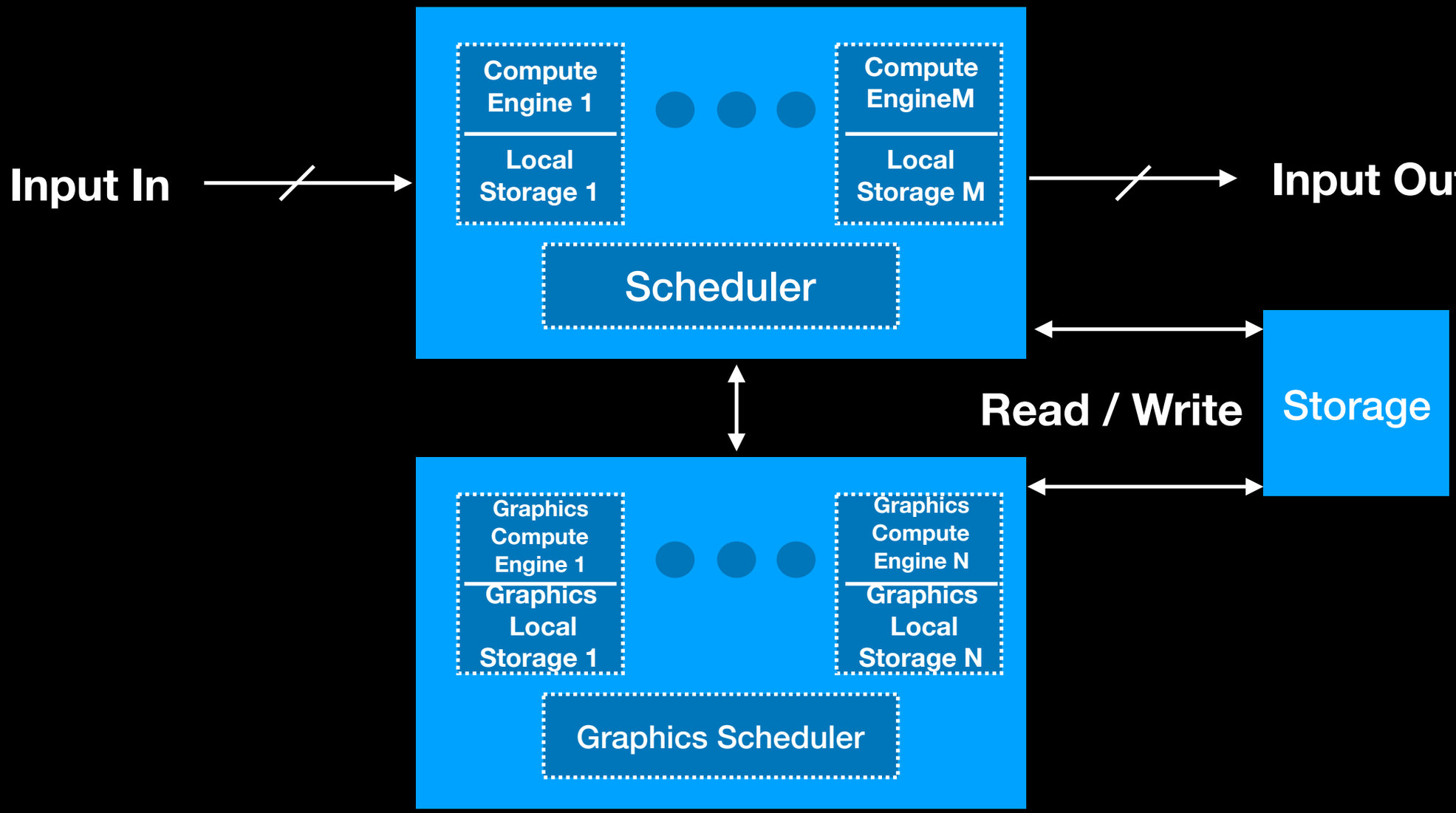


High Level Evolution

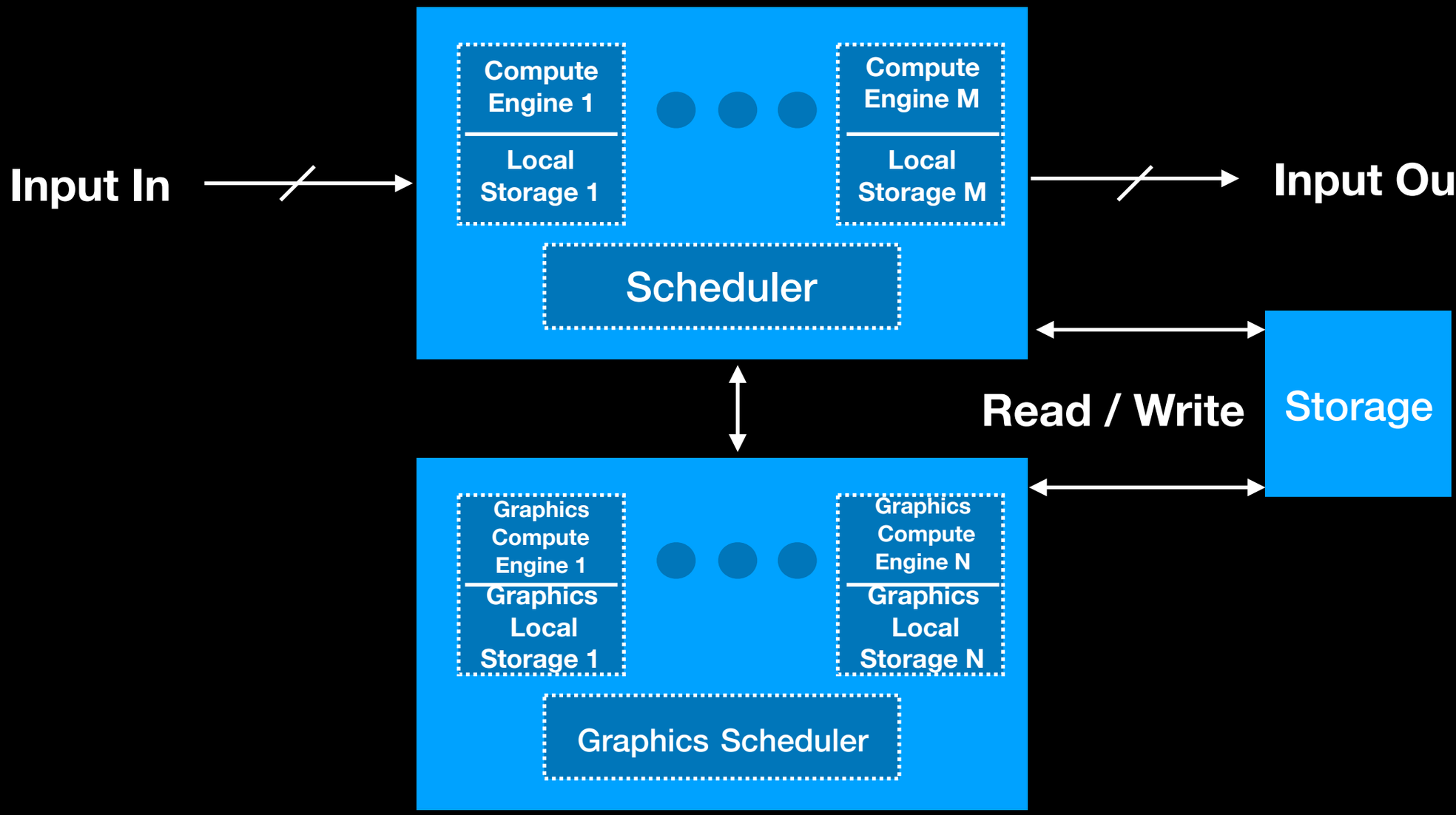


“The compute engine is inefficient / ill suited, for my graphics application.”

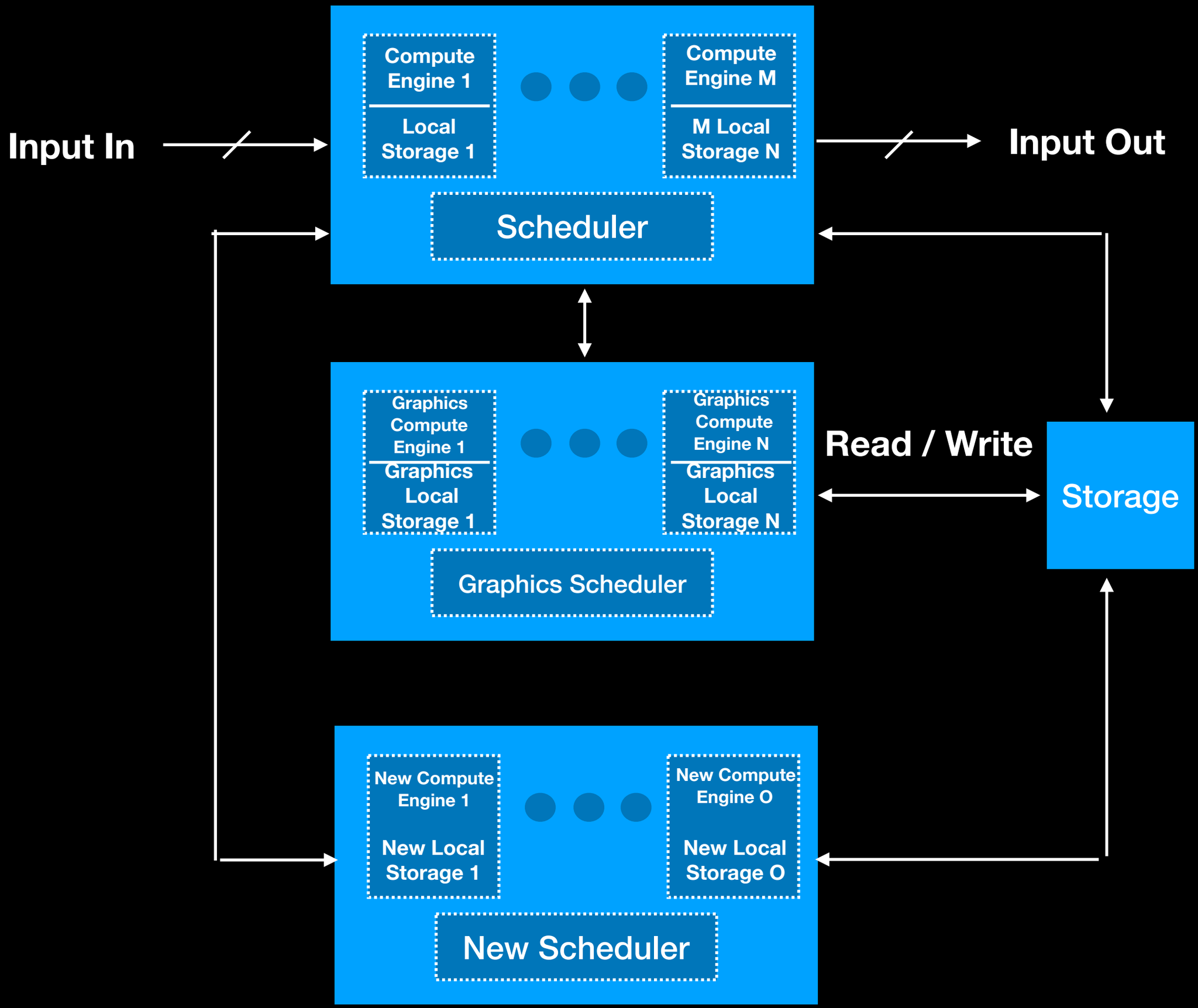
High Level Evolution

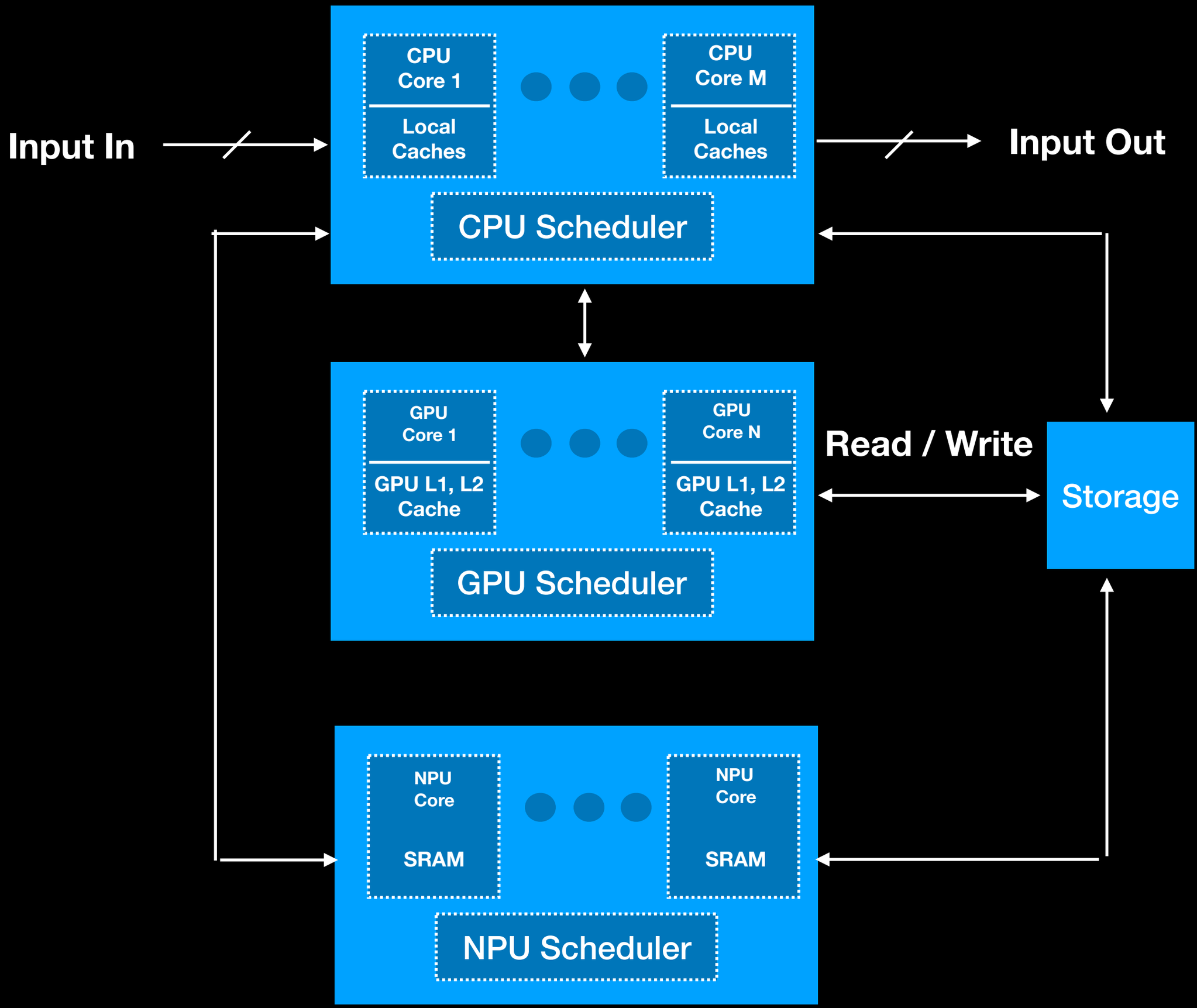


High Level Evolution



“The compute engine and the graphics engines are BOTH non-optimal for my new application. We really need to do it like this = NEW ….”


Input In
Compute Engine 1
Local Storage 1
Compute Engine M
M Local Storage N
Input Out
Scheduler
Graphics Compute Engine 1
Graphics Local Storage 1
Graphics Compute Engine N
Graphics Local Storage N
Read / Write
Storage
Graphics Scheduler
New Compute Engine 1
New Local Storage 1
New Compute Engine O
New Local Storage O
New Scheduler



In 2025 / 2026, it currently looks like this. On mobile devices, we are memory bound = “We have adequate compute. We need to make sure that operators (compute) have operands (data) to process.”

Apple CoreML

- CoreML acts a director, converting the learned model, and determining what portions of the computational graph will be computed “where”.

- Computation can be performed exclusively in the Apple Neural Engine (ANE), or, across the CPU, GPU and NPU.

- Ideally, the developer wants his / her model to “fit” perfectly, in the ANE.

- Necessary conditions for a perfect “fit” include:

Apple CoreML

- i.e. Does the model use only supported ANE operators?
- i.e. Are the tensor sizes static and fixed at compile time?
- i.e. Is the scratch ram (SRAM) adequate for the deployed model?
- i.e. Does the model perform well using only INT8 and INT16 (and minimal FP16) precision?

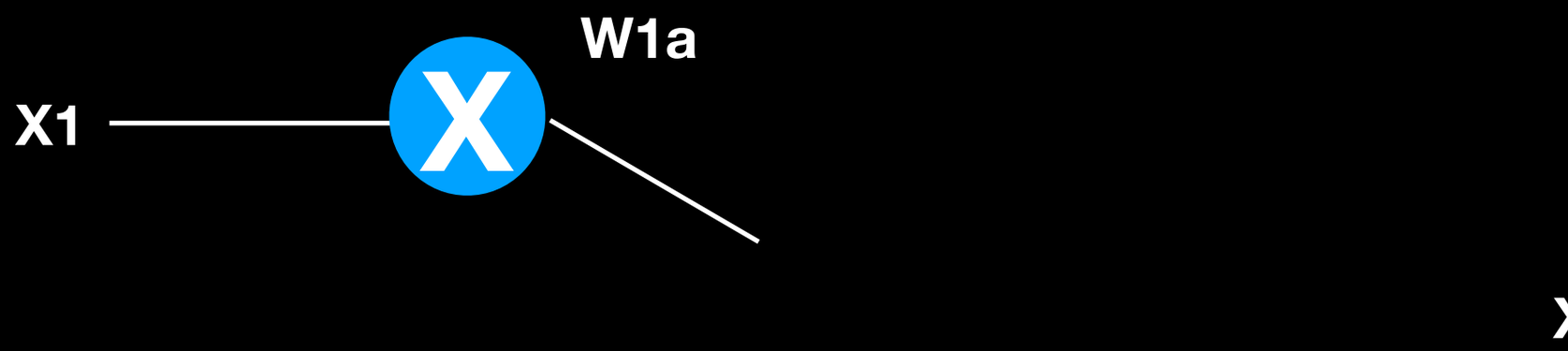
Ideal Accelerator Data Flow

- A pipeline of supported ML ops
- With static tensor shapes
- With only SRAM memory access and repeated data reuse
- And excellent model performance using low precision math

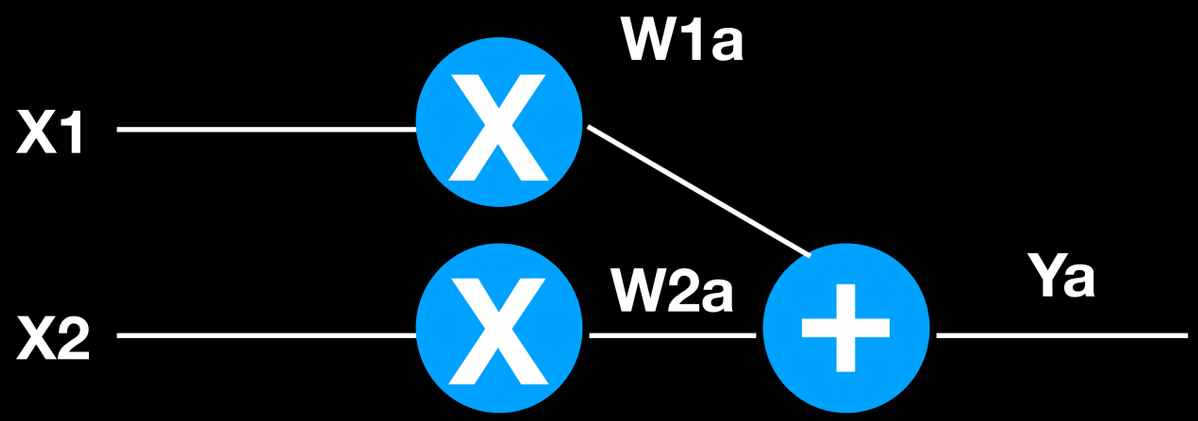
The Fundamental Building Blocks

- Multiply and Accumulate
- Non Linear Activation

Multiply

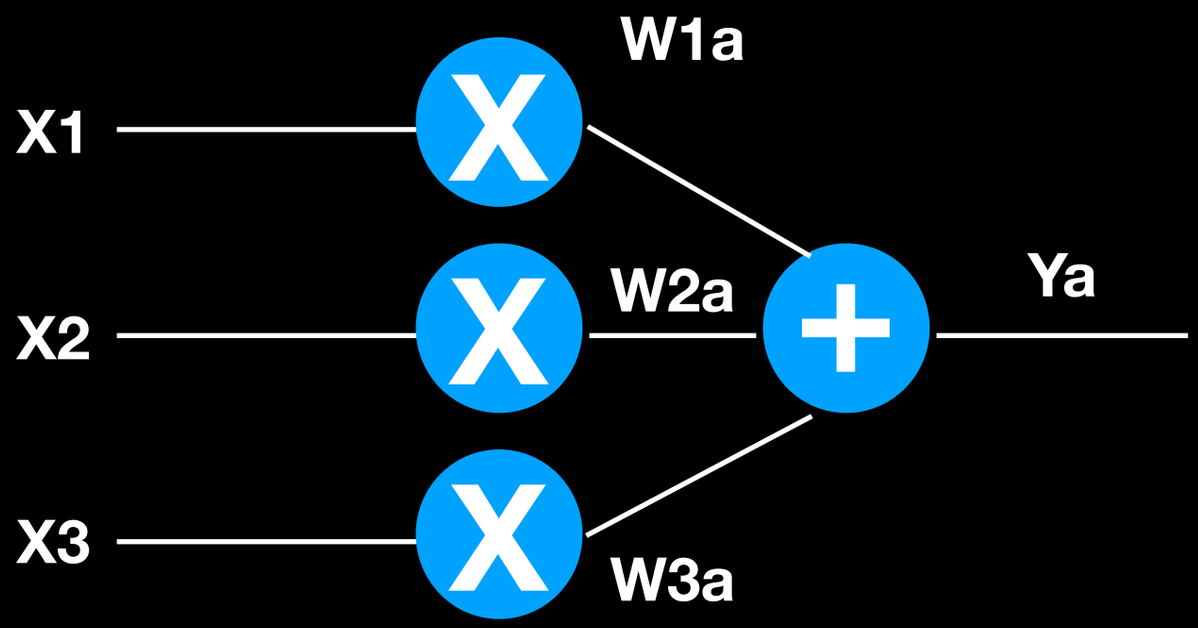


Multiply & Accumulate (MAC)



X1*W1a + X2*W2a

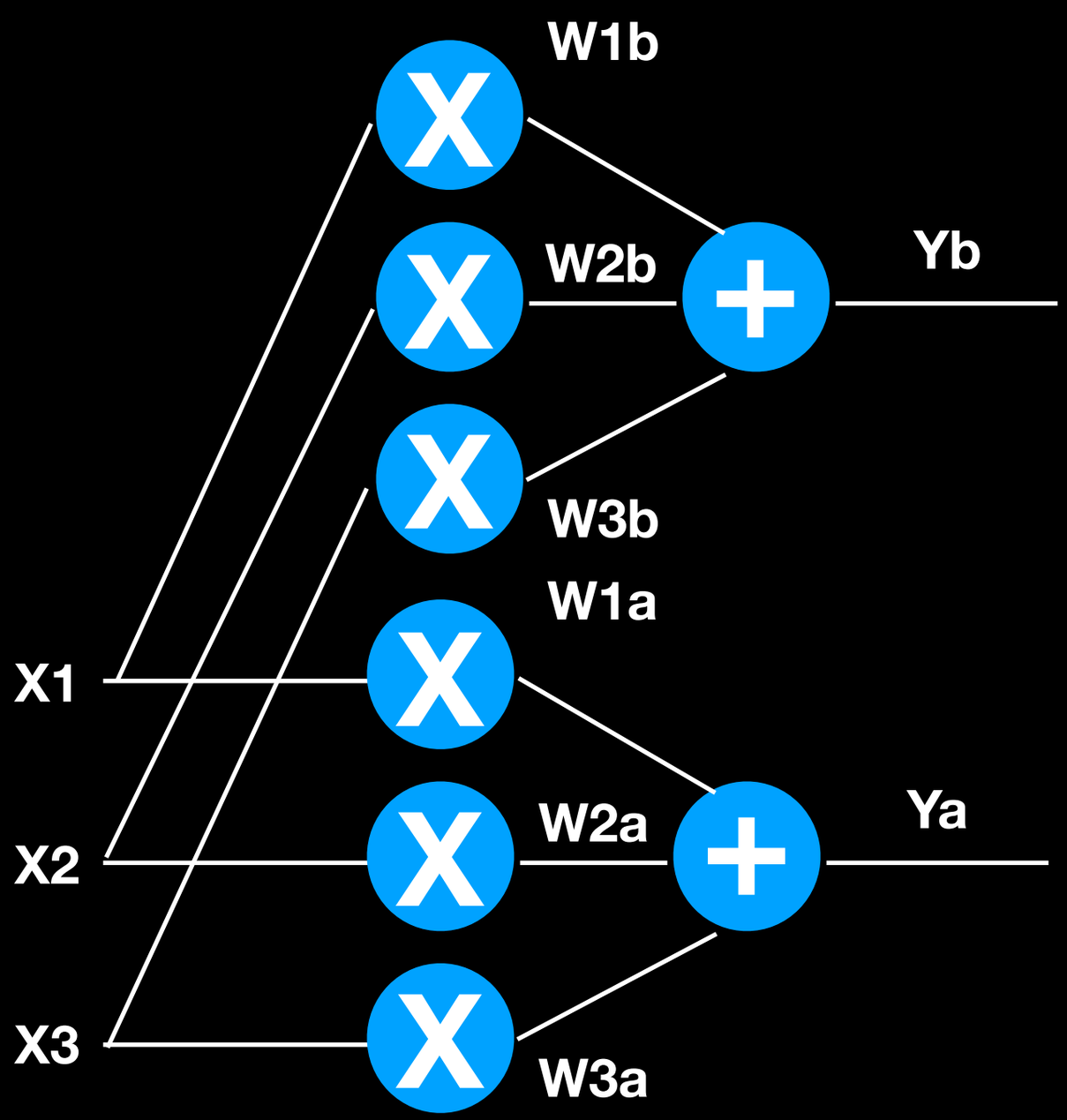
Three Multiply and Two Accumulate



X1*W1a + X2*W2a + X3*W3a = Ya

[X1 X2 X3] • [W1a W2a W3a] = Ya

Dot product is composed of MAC’s.

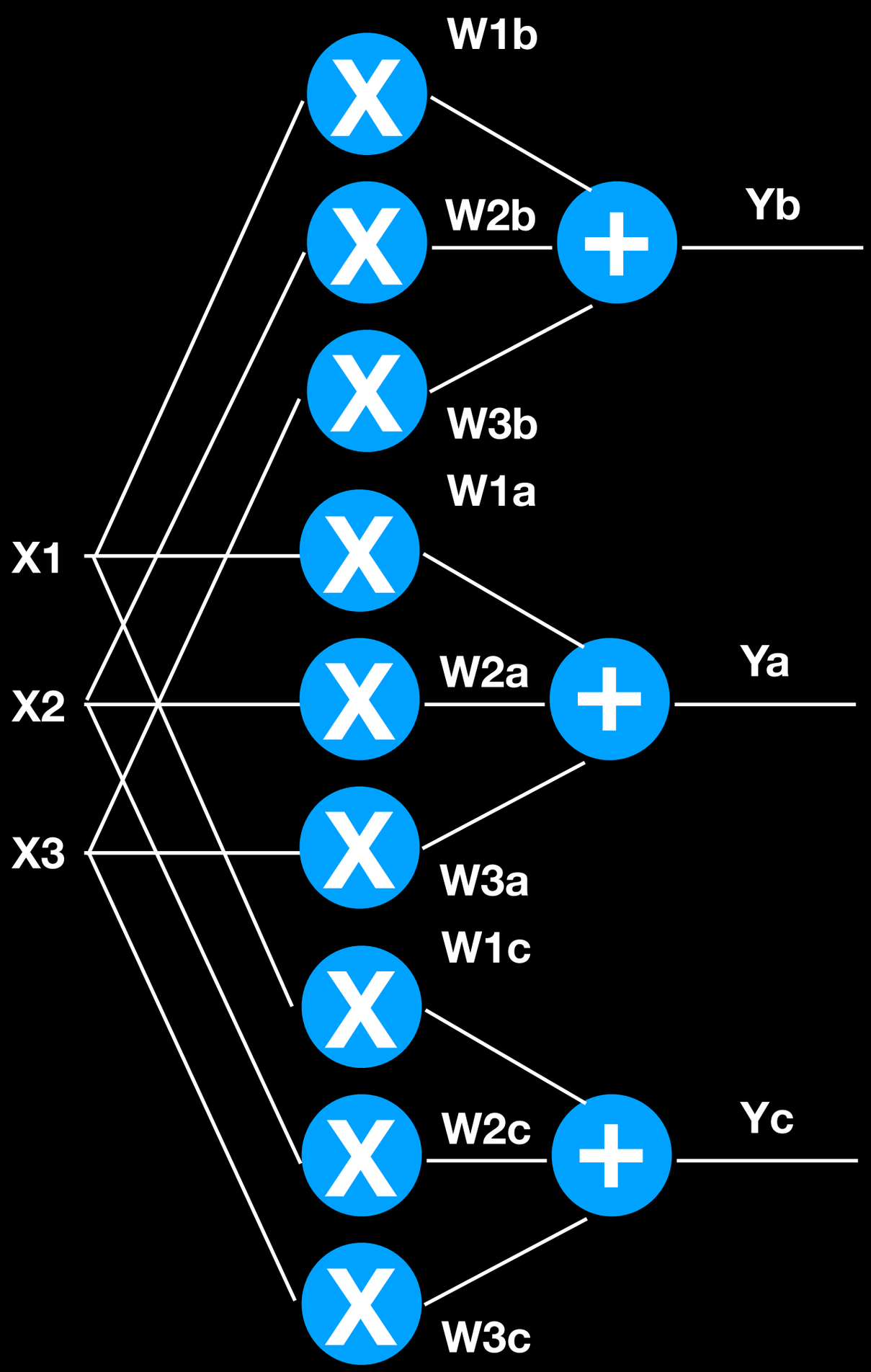


X1*W1b + X2*W2b + X3*W3b = Yb

[X1 X2 X3] • [W1b W2b W3b] = Yb

X1*W1a + X2*W2a + X3*W3a = Ya

[X1 X2 X3] • [W1a W2a W3a] = Ya



X1*W1b + X2*W2b + X3*W3b = Yb

[X1 X2 X3] ● [W1b W2b W3b] = Yb

X1*W1a + X2*W2a + X3*W3a = Ya

[X1 X2 X3] ● [W1a W2a W3a] = Ya

X1*W1c + X2*W2c + X3*W3c = Yc

[X1 X2 X3] ● [W1c W2c W3c] = Yc

[X1 X2 X3] ● [W1a W2a W3a] = Ya

[X1 X2 X3] ● [W1b W2b W3b] = Yb

[X1 X2 X3] ● [W1c W2c W3c] = Yc

$$\begin{bmatrix} X_1 & X_2 & X_3 \end{bmatrix} \begin{bmatrix} W_{1a} & W_{1b} & W_{1c} \\ W_{2a} & W_{2b} & W_{2c} \\ W_{3a} & W_{3b} & W_{3c} \end{bmatrix} = \begin{bmatrix} Y_a & Y_b & Y_c \end{bmatrix}$$

Matrix multiply is composed of MAC’s.

W1	W2	W3
W4	W5	W6
W7	W8	W9

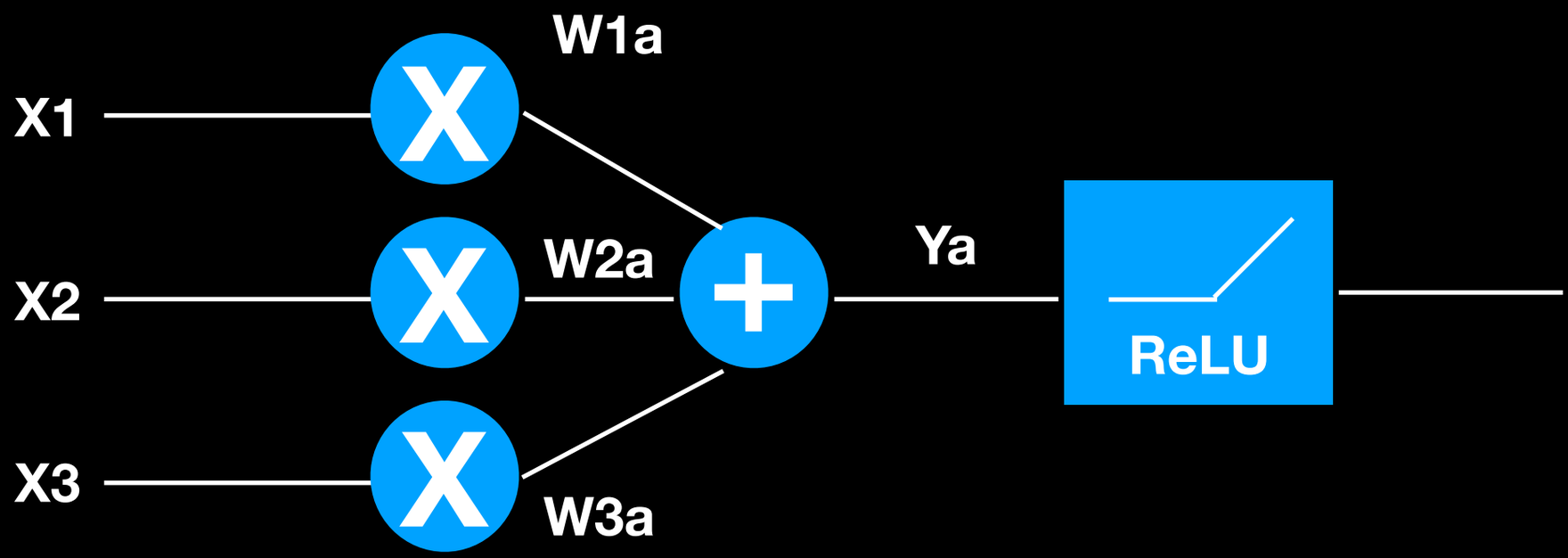
	P1	P2	P3					
	P4	P5	P6					
	P7	P8	P9					

Out = P1*W1 + P2*W2 + P3*W3 + P4*W4 + P5*W5 + P6*W6 + P7*W7 + P8*W8 + P9*W9

Out = [W1 W2 W3 W4 W5 W6 W7 W8 W9] ● [P1 P2 P3 P4 P5 P6 P7 P8 P9]

Convolution is composed of a dot product - which is composed of MAC’s.

Three Multiply and Two Accumulate + Activation



ReLU

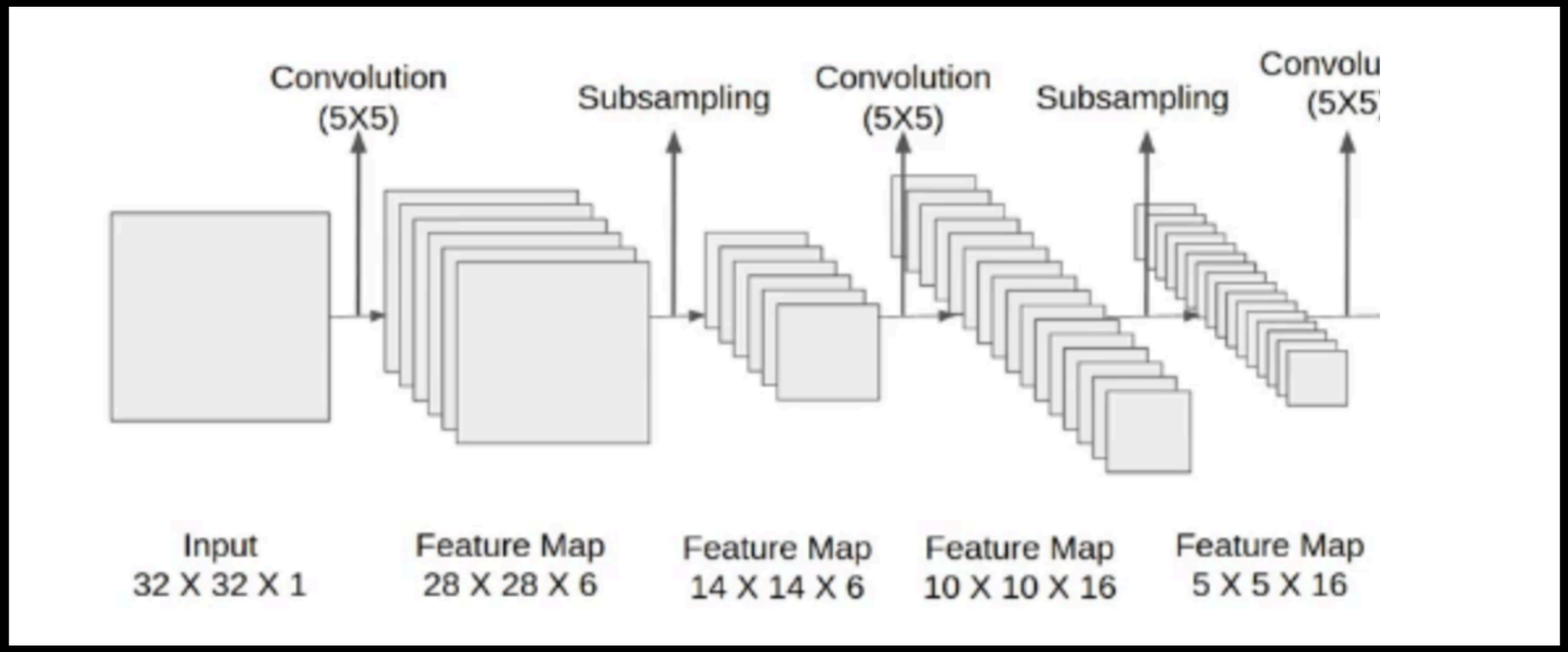
- cmp = (x > 0) // 1-bit binary result
- y = cmp * x
- 1 comparator and 1 multiplexer.

LeNet 5

Convolutional layer is composed of MAC's.

Non linear activations (like ReLU) are a simple comparator & multiplexer.

LeNet 5

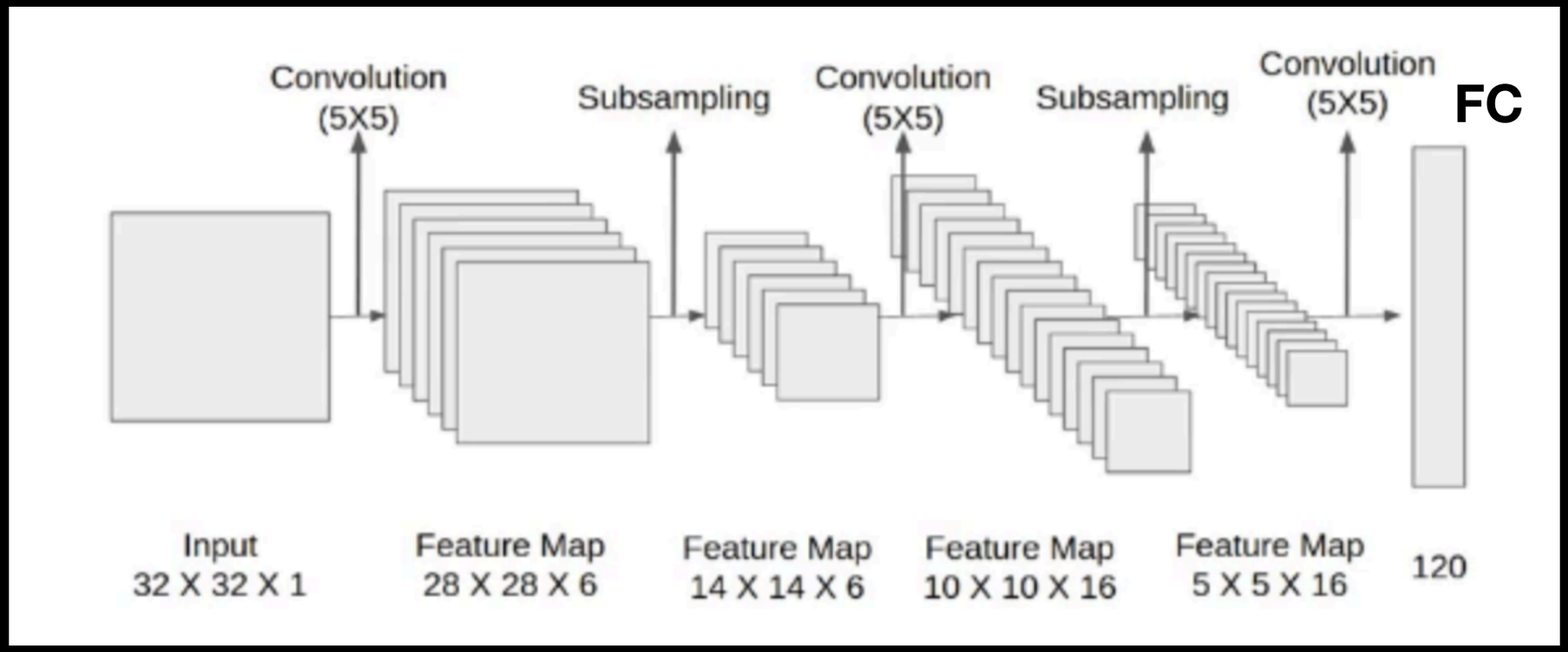


Convolutional layer is composed of MAC's.

Non linear activations (like ReLU) are a simple comparator & multiplexer.

Cascaded convolutional layers are just more MAC's operating on tensors with different dimensions.

LeNet 5



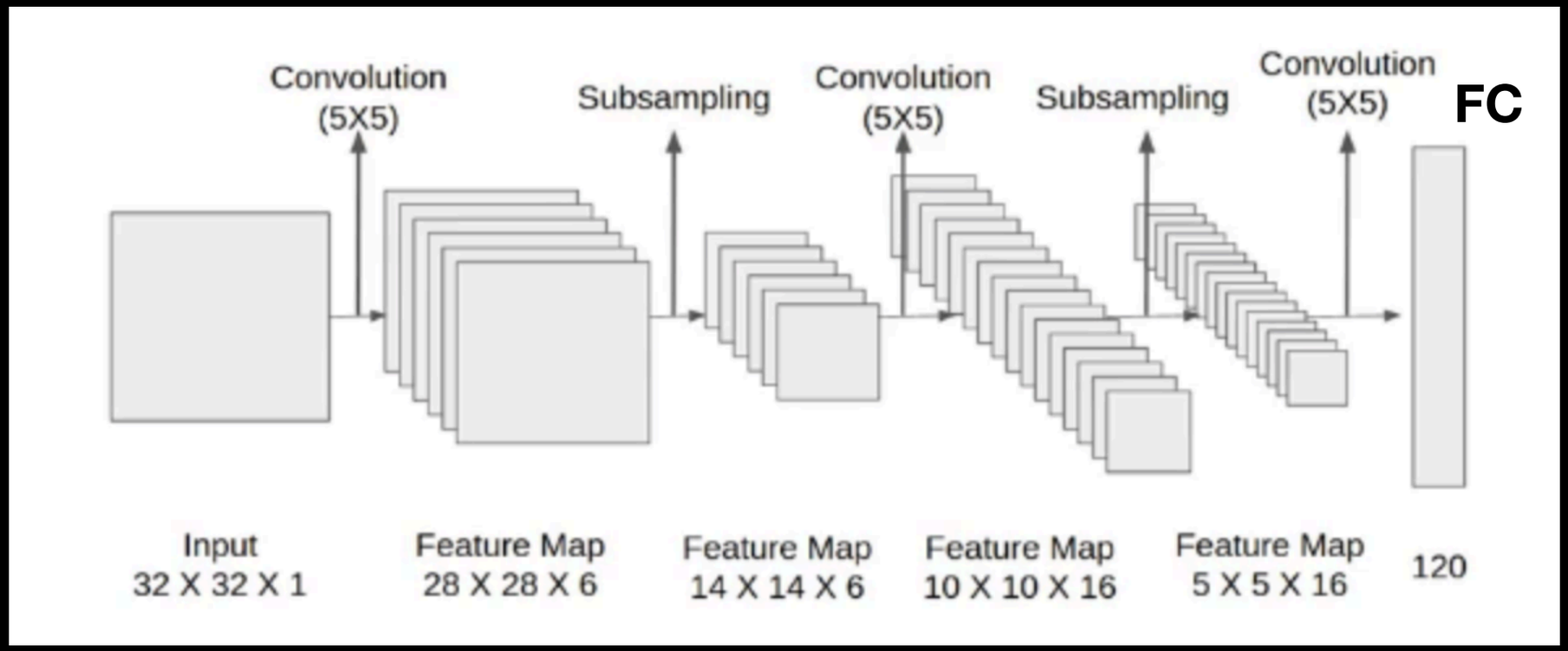
Convolutional layer is composed of MAC's.

Non linear activations (like ReLU) are simple comparator & multiplexer.

Cascaded convolutional layers are just more MAC's operating on tensors with different dimensions.

Fully connected (FC) layer is composed of MAC's / matrix multiplys.

LeNet 5

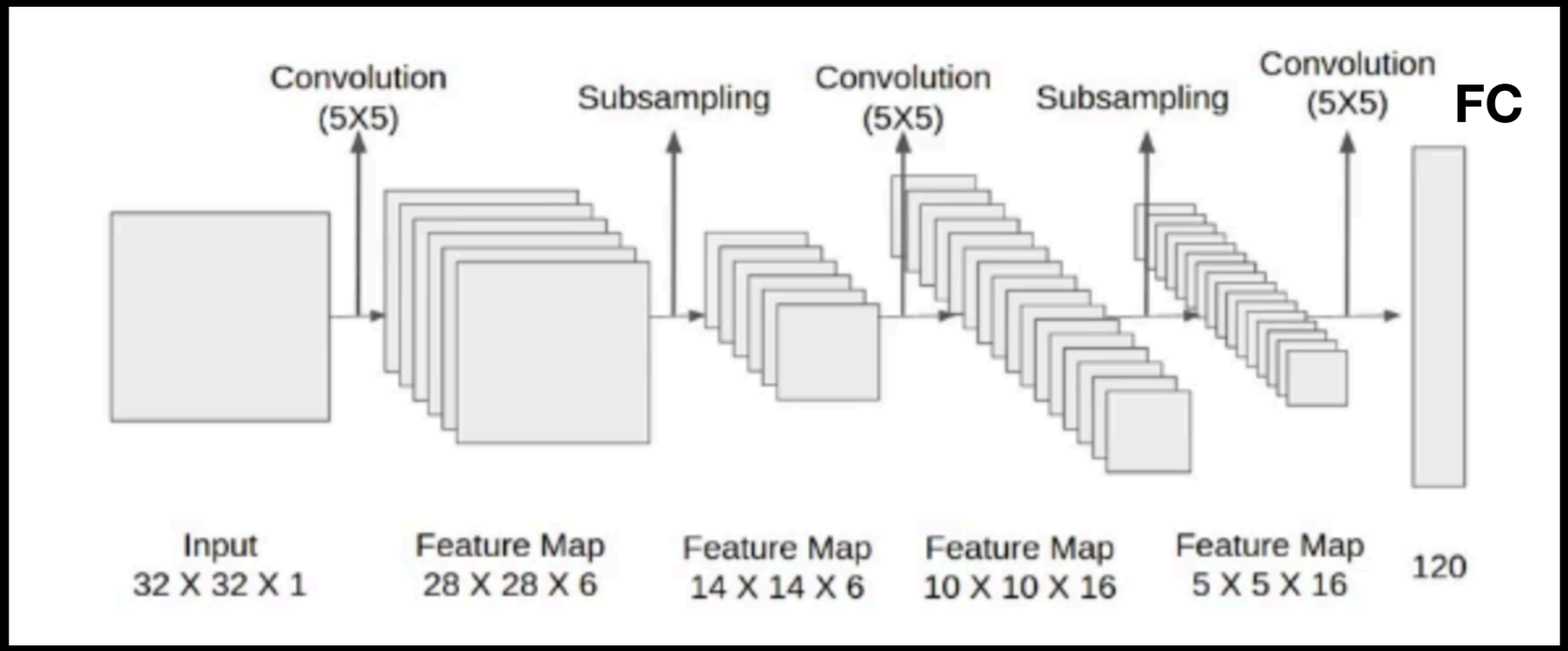


If the convolutional weights are stored in SRAM, they are close by and easy to access.

Notice how data is “re-used” / passed from one output to the next input, using the cascade structure and the feature maps.

Given fixed and known tensor dimensions, the compiler is well positioned to optimize the resource utilization.

LeNet 5



If the model “fits”, everything is self contained between the input and the output.

i.e. The MAC units are fully utilized, as the operands (weights and feature maps) reside In SRAM.

If the model is too big to fit in SRAM, performance degrades, since data needs to be transferred to and from SRAM via external DRAM.

Notes

- X1*W1a + X2*W1b can be implemented on the CPU, the GPU, or the NPU.
- For a single or very small number of operations, the CPU often produces the fastest result due to minimal setup and dispatch overhead.
- However, if billions of these calculations are needed, the shortcomings of the CPU are then exposed.
- Note: The GPU also has massive MAC units (ALU’s).
- However, by design, the GPU relies on latency hiding - via many concurrent threads - to keep the MAC units fully utilized.

Notes

- When stalls occur and there is insufficient parallel work to hide latency - often due to memory access or dependencies - GPU performance degrades.

- Unlike GPU's, NPU's do not primarily result on latency hiding; instead, they use deterministic data flow pipelines with tightly managed memory.

- The NPU pipeline assumes data re-use - which is the norm - for ML models.

- The NPU pipeline is designed with the intent of always having the needed operands in nearby SRAM to maximize MAC throughput.

- When the latter assumption is violated, hybrid techniques are employed as a workaround.

CNN vs Transformer On NPU

- If the CNN model is too large, it will not “fit” on the NPU.
- If the Transformer model is too large, it will also not “fit” on the NPU.
- If the model does not “fit” in on chip SRAM, frequent external DRAM accesses will be required, significantly increasing power and bandwidth usage.
- For the current NPU accelerators in 2025, CNN’s are significantly more amenable than Transformers, especially for high resolution vision workloads.

CNN

- Small weight matrices for convolution.
- 1x1 convolution for channel mixing; depth wise convolution for spatial filtering.
- Overall compute structure is highly amenable to tile based computation and data re-use.

Transformer

- Q, K and V matrices scale linearly with token count and embedding size = O(N*d)

- Attention matrix IS huge, and has computational complexity O(N*N) - where N is the number of tokens in the input embedding.

- Because of attention’s global data dependency, efficient tiling is difficult and often inefficient on NPU’s.

- Repeated use of the softmax at every transformer layer, adds to the complexity and is also not tile friendly.

CNN vs Transformer On NPU

- There are two significant advantages of Transformers versus CNN's.

Advantage1:

- Transformers have global attention built in at every layer.

- In contrast, CNN's have local attention that increases, with increasing CNN depth / layers.

Advantage2:

- Transformers exhibit strong scaling behavior with increased data and model capacity, often outperforming CNN's at large scale.

CNN vs Transformer On NPU

- But, are either of these advantages on NPU accelerators in 2025?

- Let’s first look at global attention, and then the scaling, using a vision transformer (ViT) as reference.

CNN vs Transformer On NPU

- Suppose we start with an input image of 224x224x3.
- We divide the image into patches of dimension 16x16.
- This results in 196 (=14x14) unique patches (or tokens) which are consumed by the transformer.
- At EVERY layer in the transformer, a 196x196 attention matrix is computed, since every token attends to every other token.
- For multi-head attention, this also occurs PER ATTENTION HEAD.

CNN vs Transformer
On NPU

- Because the attention mechanism is so big, a workaround is employed.
- Workaround: “local transformer attention via windowed attention”
- However, this trades off full global context for efficiency.

CNN vs Transformer On NPU

- What is windowed attention?
- Divide the 14x14 token array into four 7x7 token arrays.
- Now, instead of 196 tokens attending to each other, only 49 tokens attend to each other (i.e. only the tokens in the window).
- Windowed attention Complexity = 4 * O(N/4 * N/4) = O(N*N / number of windows)
- Non windowed attention complexity = O(N*N)

CNN vs Transformer On NPU

- But now, every token no longer attends to every other token - which is what made transformers appealing, to begin with.

- To mitigate this problem (via engineering), a shift is employed in the following layer, creating an overlap region.

CNN vs Transformer On NPU

- Also, for every row in the attention matrix, a softmax operation needs to be performed.

- Softmax is expensive due to the use of exponentials and normalization across tokens.

- In classic CNN classifiers, softmax is typically only at the output; in Transformers, softmax appears inside every attention layer.

CNN vs Transformer On NPU

- Another advantage of transformers over CNN's is the improved model performance on vision tasks, for very large transformer models.

- However, large models require a significant amount of memory.

- Unfortunately, SRAM memory is very limited on existing NPU's.

- Recall: If unnecessary DRAM fetches are required, performance degrades.

CNN vs Transformer
On NPU

- Workaround: Reduced precision
- Workaround: Smaller transformer models

CNN vs Transformer On NPU

- Reduced precision is a win for both CNN's and Transformers.
- However, Transformers often require more careful quantization to avoid accuracy loss.

CNN vs Transformer
On NPU

- Reduce the model size: Make each layer smaller by decreasing the length of the embedding vector associated with the patches and weight matrices, Wq, Wk, and Wv smaller.

- i.e. Project the input token length from 16*16*3 = 768 to say, 256.

- i.e. Make the resulting Q, K, and V matrices smaller, by decreasing the column dimension d associated with the Wq, Wk, and Wv matrices.

- i.e. Q = X*Wq, K = X*Wk, V = X*Wv.

- However, this directly impacts model performance.

CNN vs Transformer On NPU

A comment on tiling:

- Just like decreasing precision can be win when the model degradation is minimal, tiling is an easy win if the model or algorithm is amenable.

- CNN’s are naturally amenable to tiling, since by construction, convolution is a tile centric operation.

- Q, K, and V projections also tile well, since they are general matrix multiplication operations.

- However, the bottleneck for Transformers is the attention, softmax normalization and the resulting memory traffic.

CNN vs Transformers On NPU

Modern NPU’s support Transformer acceleration (e.g. fused attention, block sparse attention, etc.), but CNN’s remain more efficient for high resolution vision due to superior data reuse and tiling.

Summary / Concluding Thoughts

- This slide presentation was intended as a high level walk through of NPU's.

- First, we established the role of NPU's in modern day computing.

- Then, we compared two major deep learning architectures - CNN and Transformers, illustrating how attention, makes Transformers more challenging to deploy on an NPU.