



UNIVERSITY OF
TORONTO



Deep Learning Hardware Acceleration

Jorge Albericio⁺ Alberto Delmas Lascorz

Patrick Judd Sayeh Sharify

Taylor Hetherington*

Natalie Enright Jerger

Tor Aamodt*

Andreas Moshovos

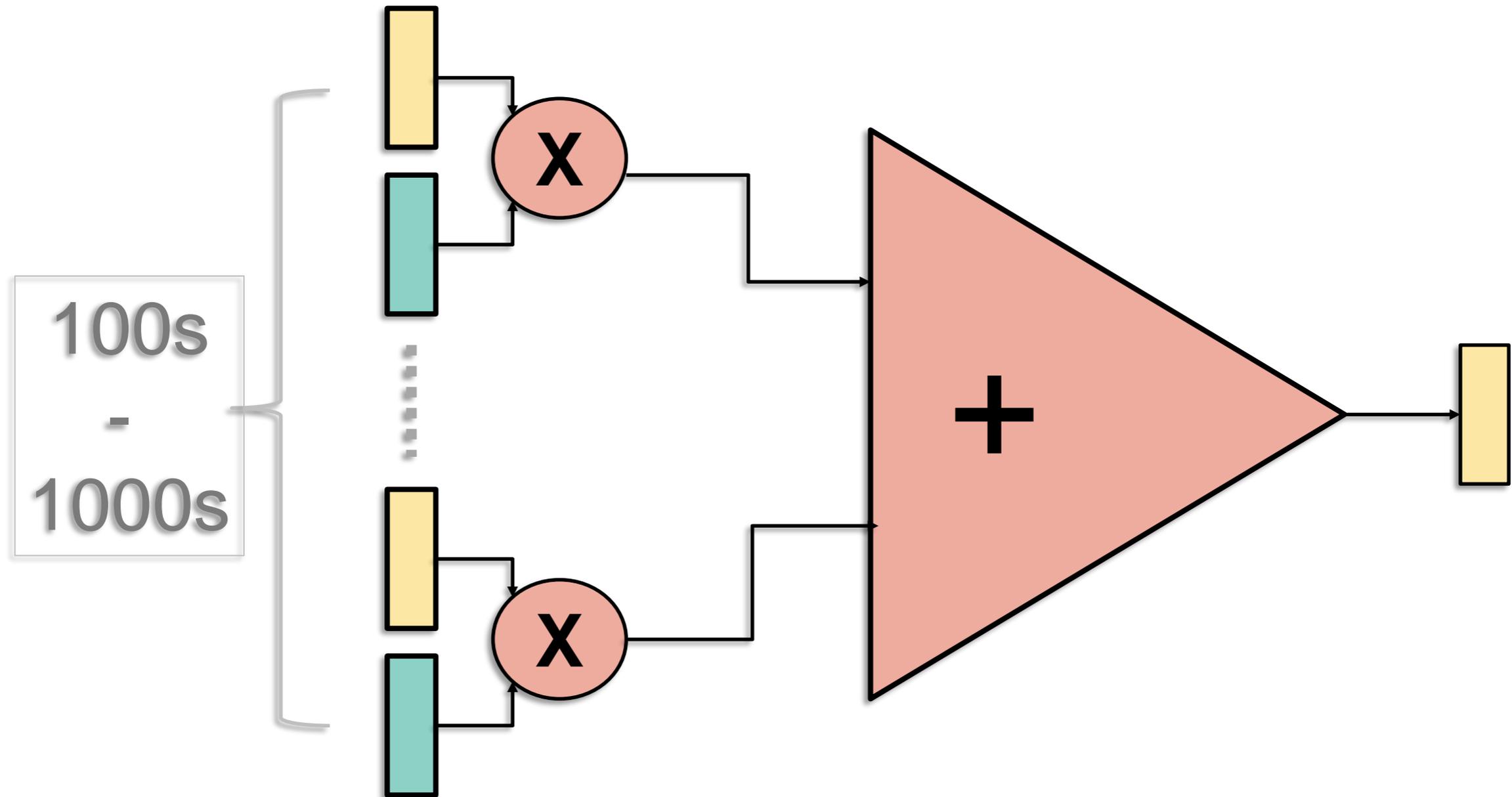
+ now at NVIDIA

Disclaimer

The University of Toronto have filed **patent applications** for the mentioned technologies.

Deep Learning: Where Time Goes?

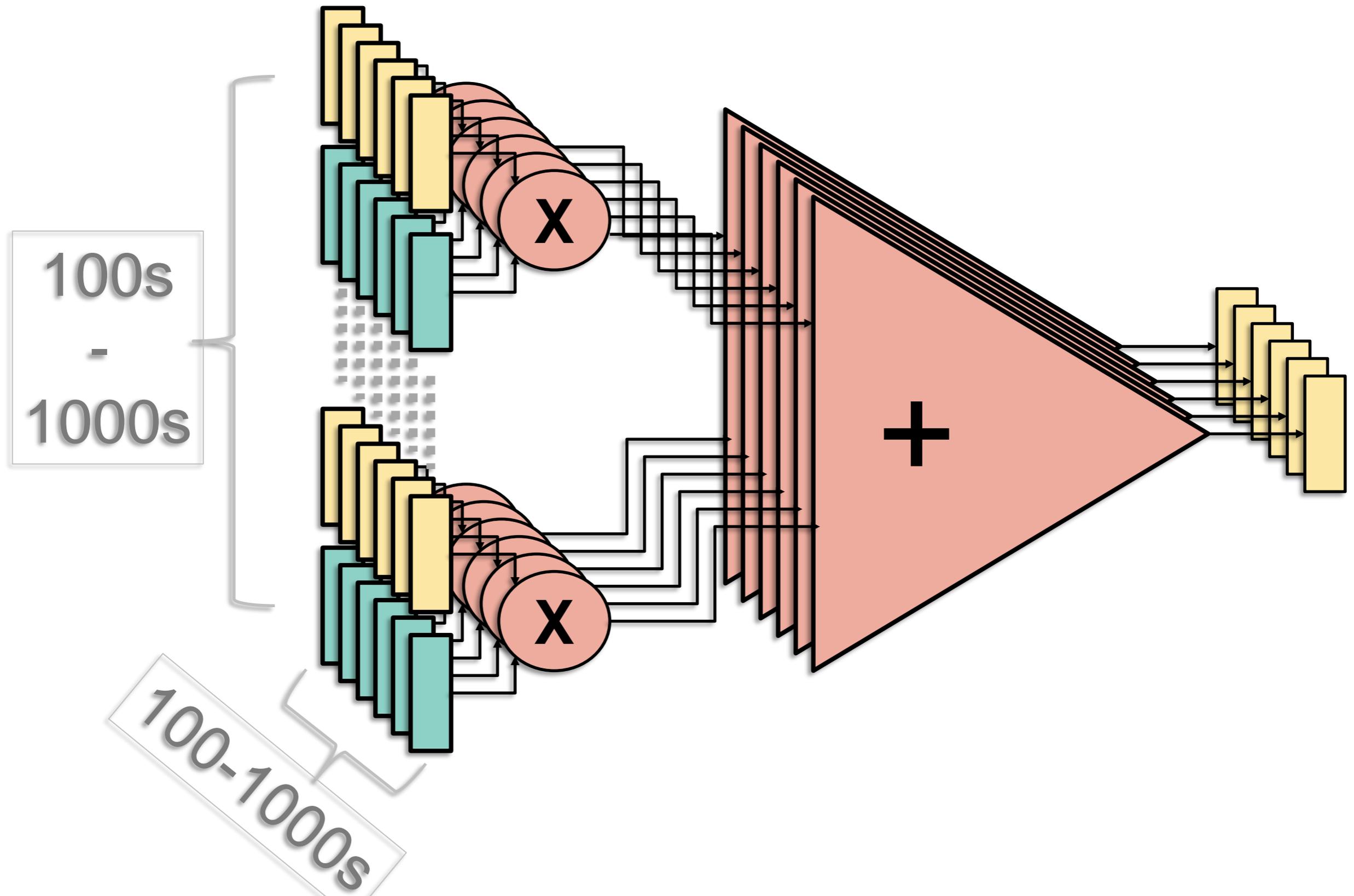
Time: ~ 60% - 90% → inner products



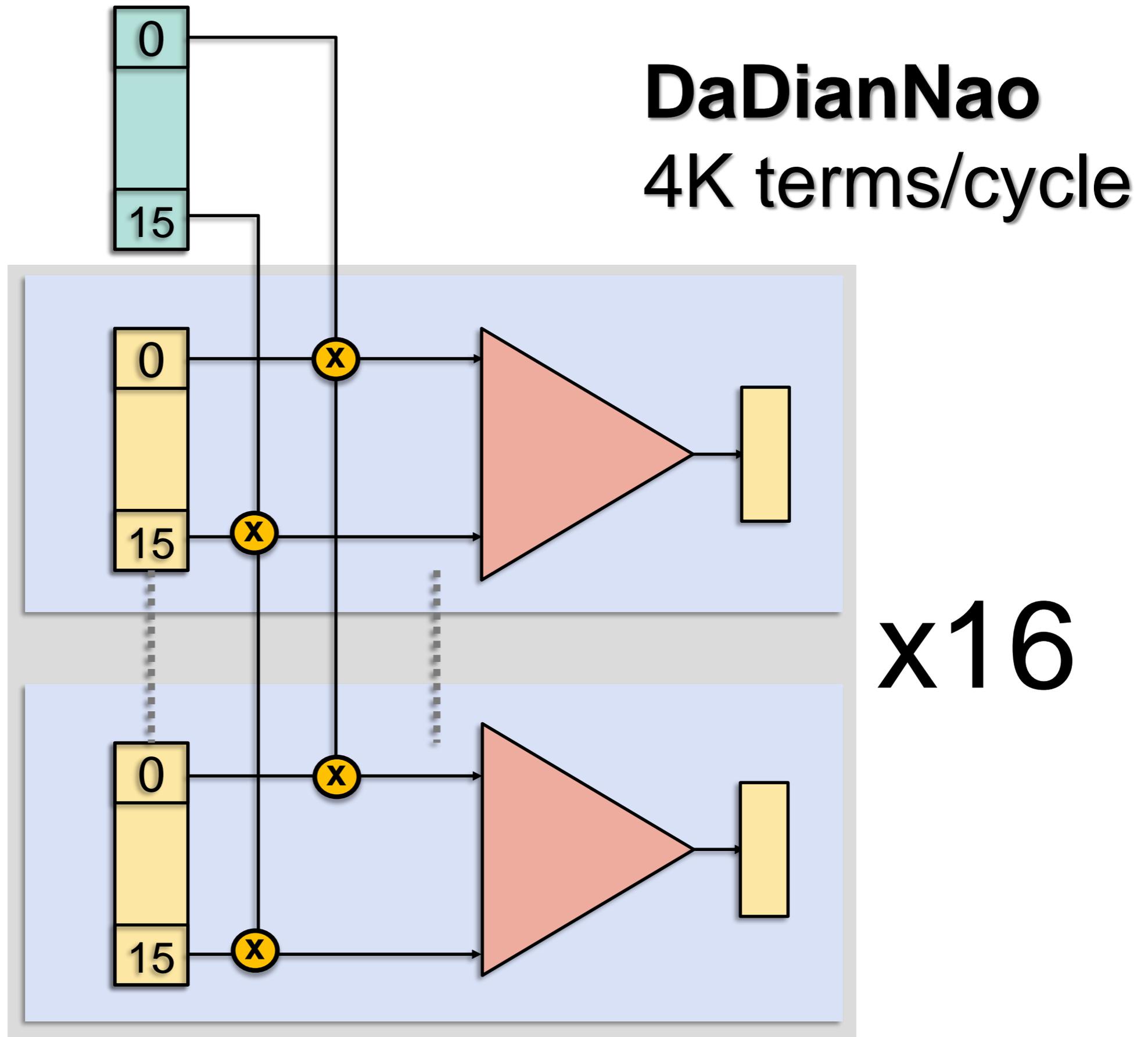
Convolutional Neural Networks: e.g., Image Classification

Deep Learning: Where Time Goes?

Time: ~ 60% - 90% → inner products



SIMD: Exploit Computation Structure

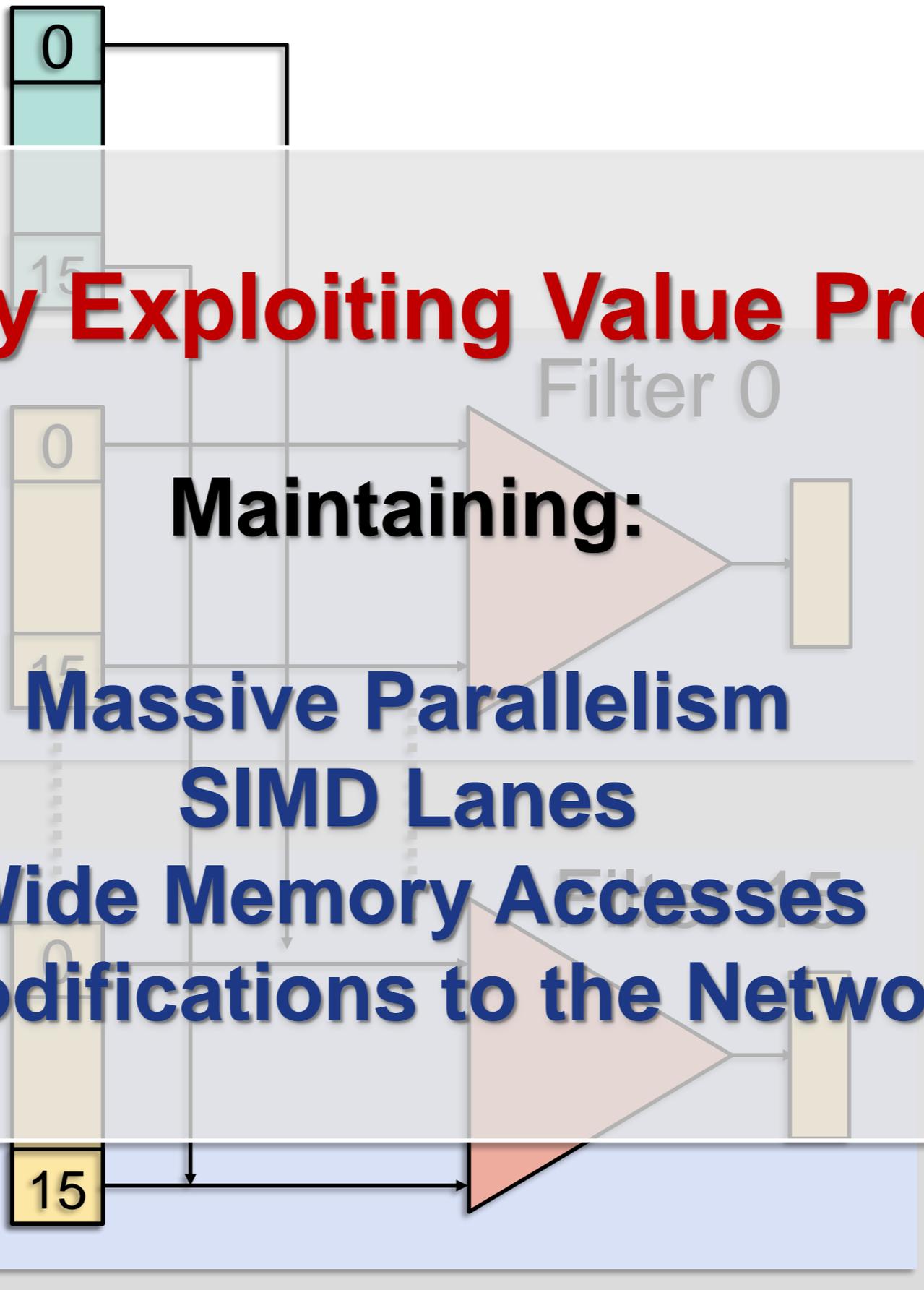


Our Approach

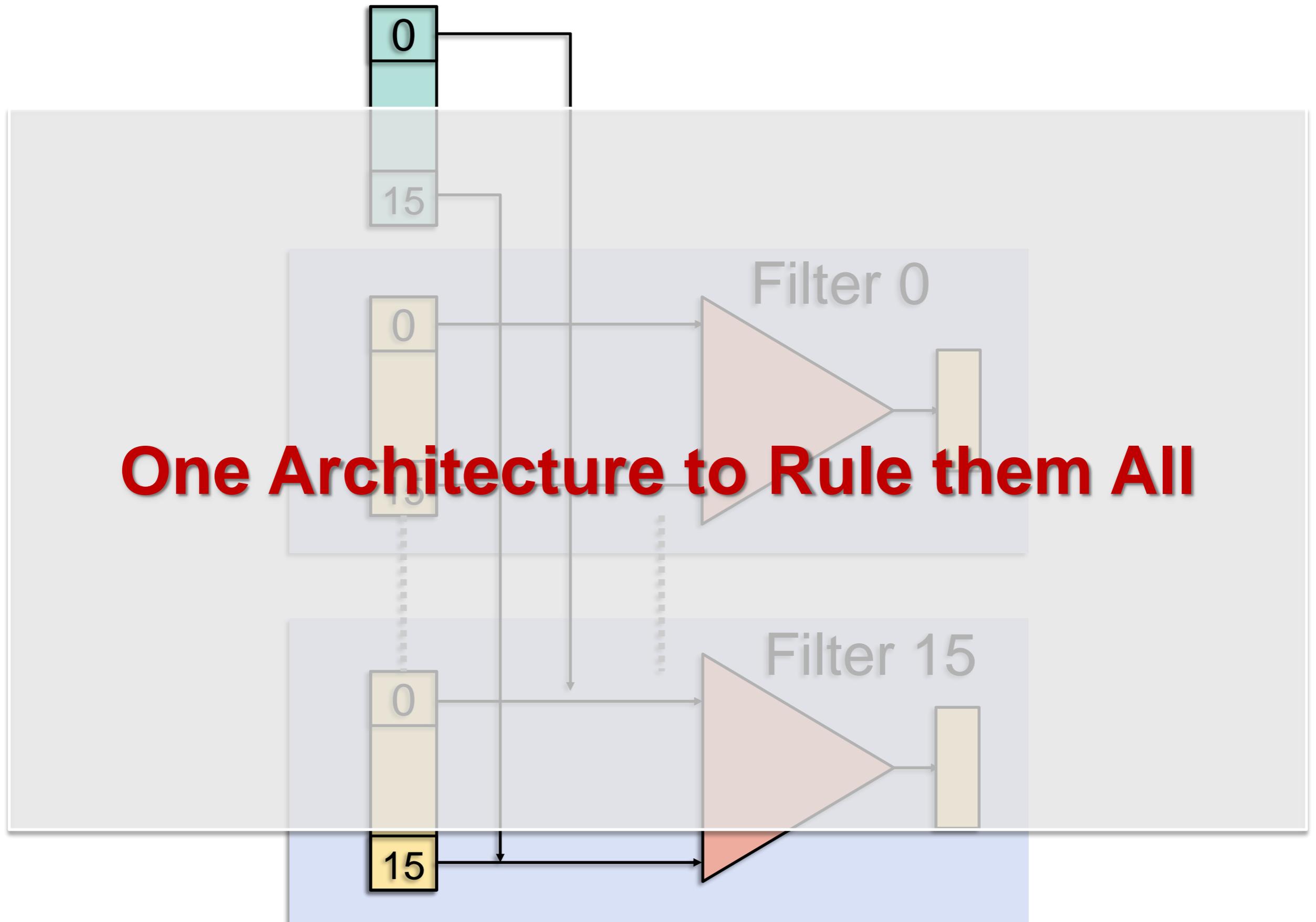
Improve by Exploiting Value Properties

Maintaining:

- Massive Parallelism**
- SIMD Lanes**
- Wide Memory Accesses**
- No Modifications to the Networks**



Longer Term Goal

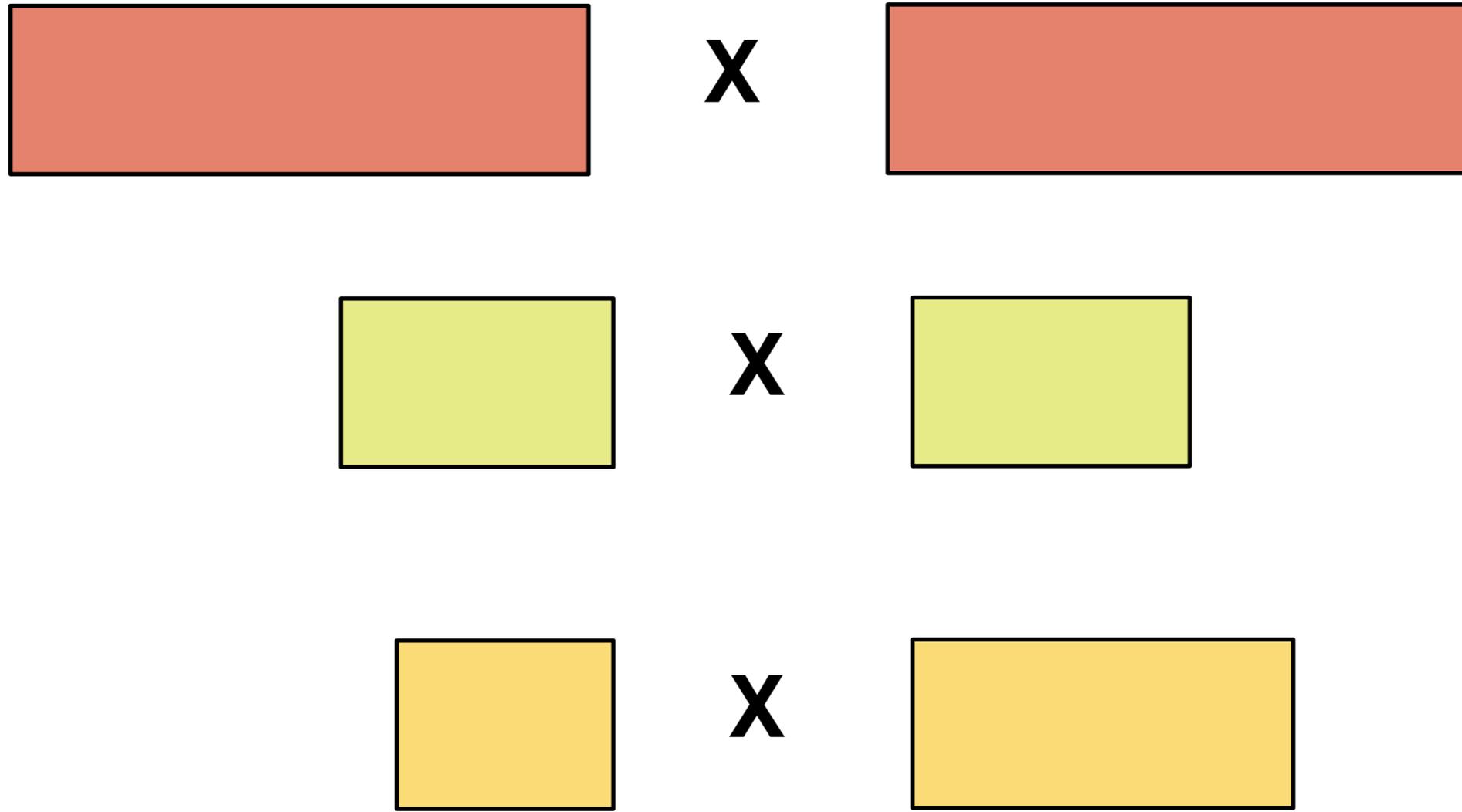


Value Properties to Exploit?

0 **x** **A**

0.0...0a **x** **A**

Value Properties to Exploit



Value Properties to Exploit



X

A



X

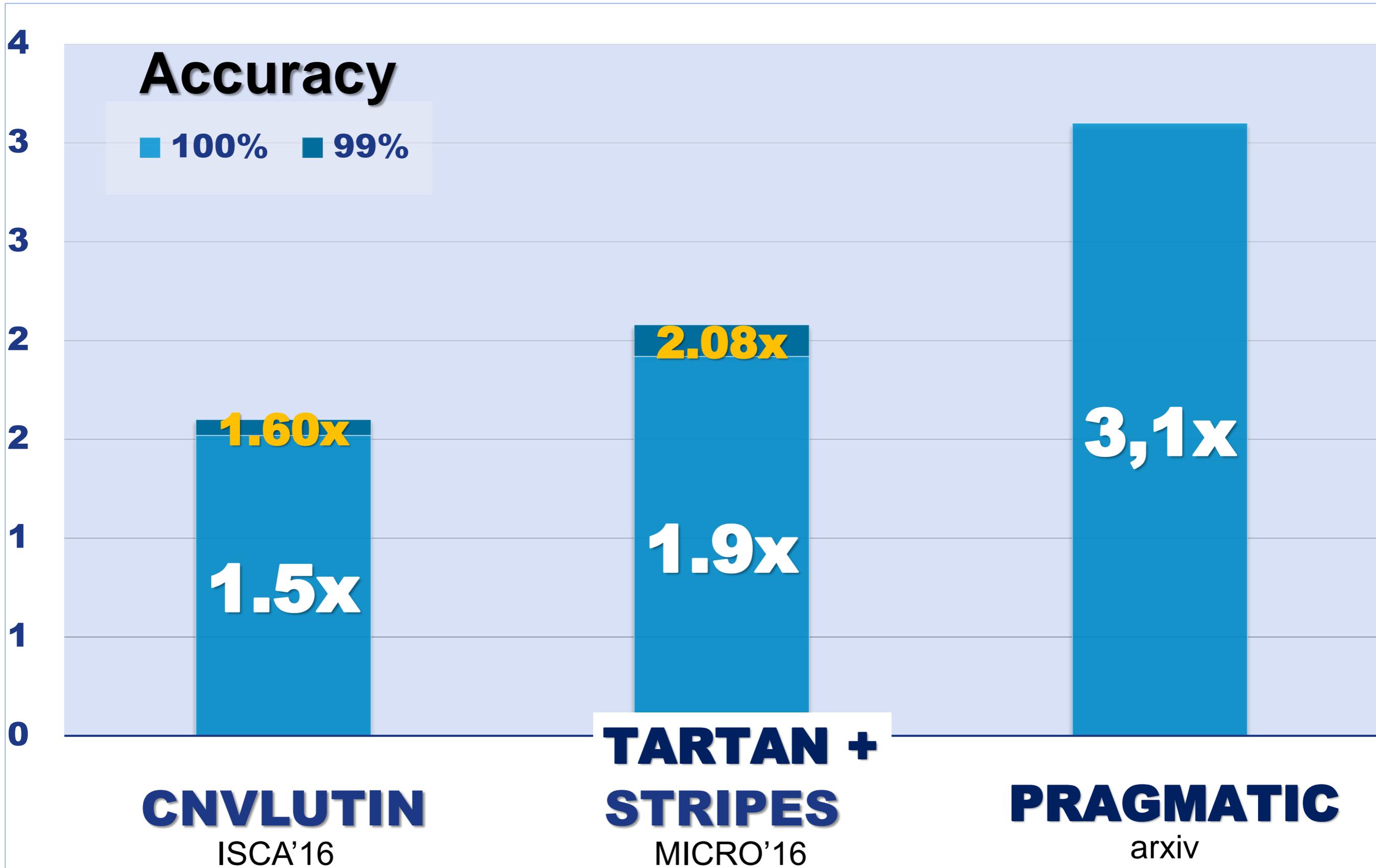
A



X

A

Our Results: Performance



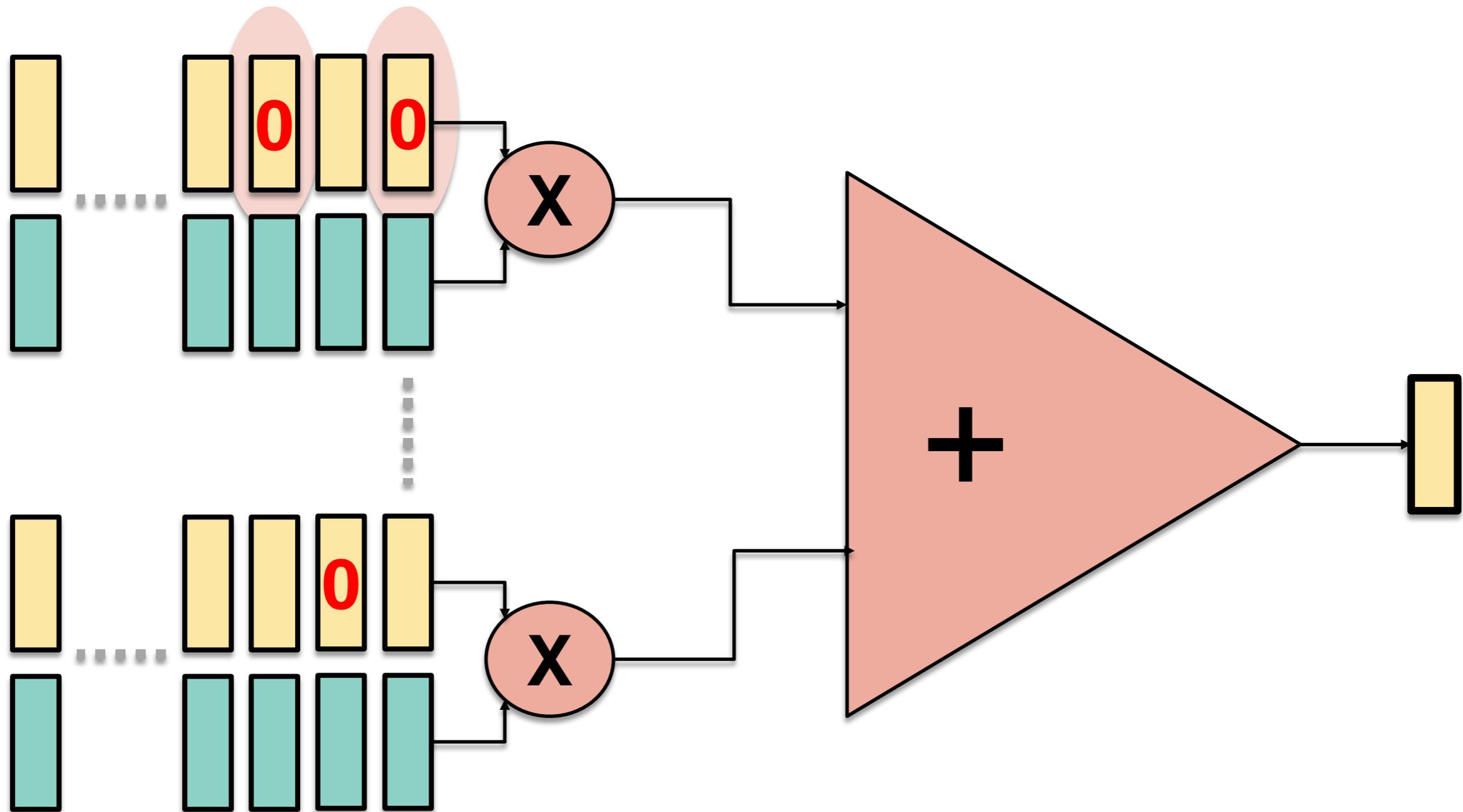
vs. DaDianNao which was ~300x over GPUs

Our Results: Memory Footprint and Bandwidth

- **Proteus:**

44% less memory bandwidth + footprint

Cnvlutin: ISCA'16



Many **ineffectual** multiplications

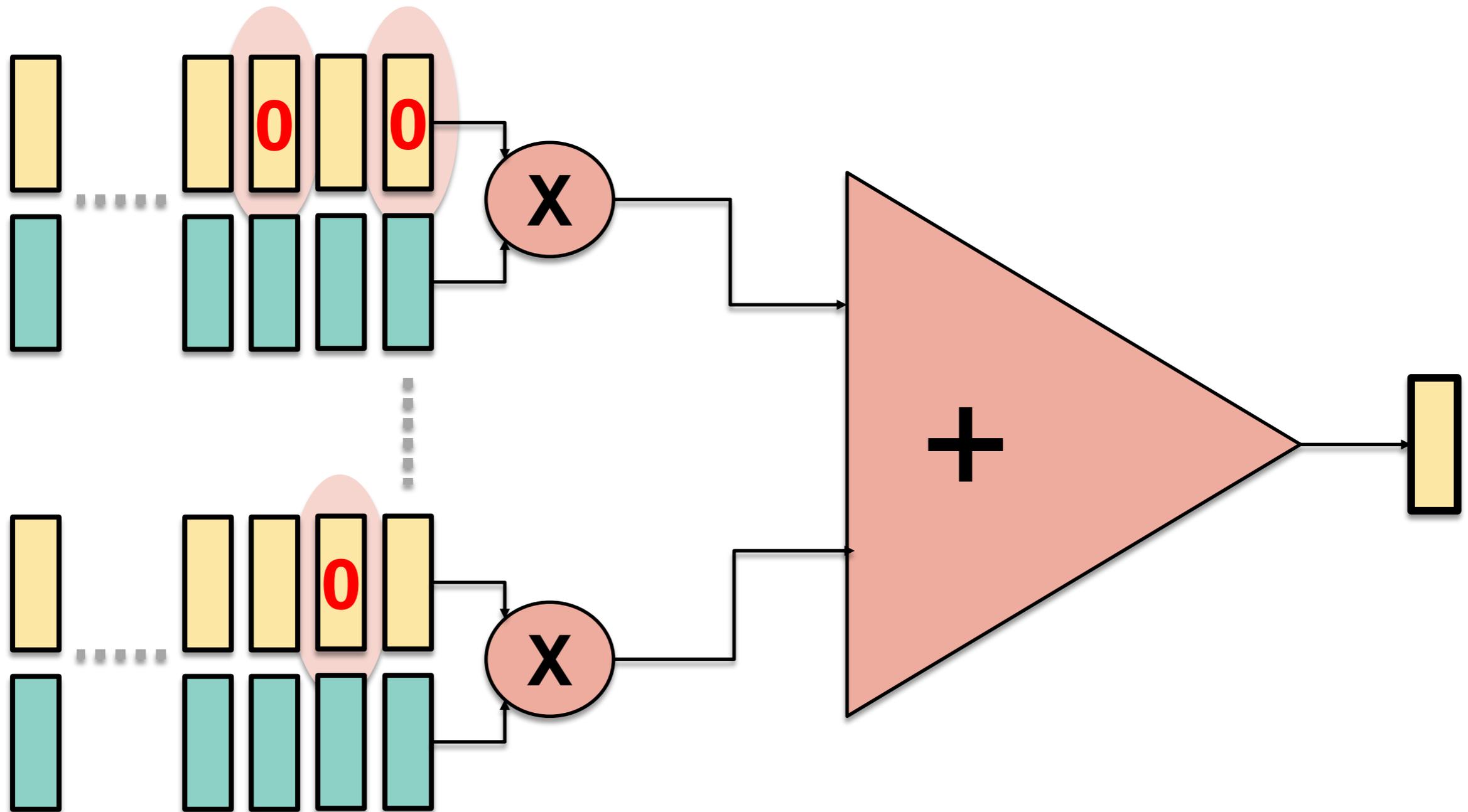
Many Activations and Weights are **Intrinsically** Ineffectual (zero)



- Zero Activations:
 - Runtime calculated
 - None that are **always** zero

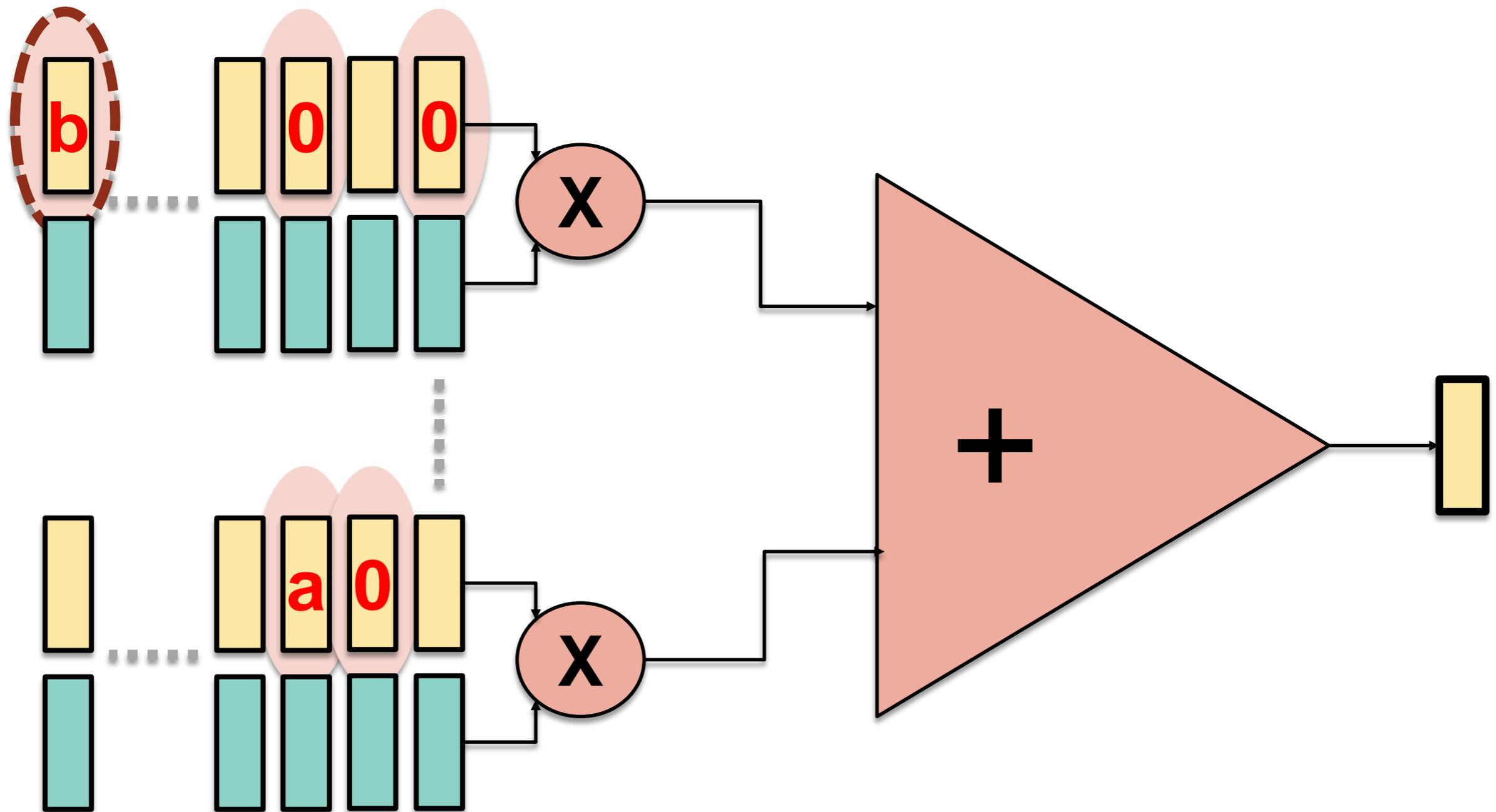
- Zero Weights:
 - Known in Advance
 - Not pursued in this work

Cnvlutin



Many **ineffectual** multiplications

Cnvlutin



Many **more ineffectual** multiplications

Beating Fast and “Dumb” SIMD is Hard



On-the-fly ineffectual product elimination

Performance + energy

Optional: accuracy loss + performance

Cnvlutin

No Accuracy Loss

+52% performance

-7% power

+5% area

Can relax the ineffectual criterion

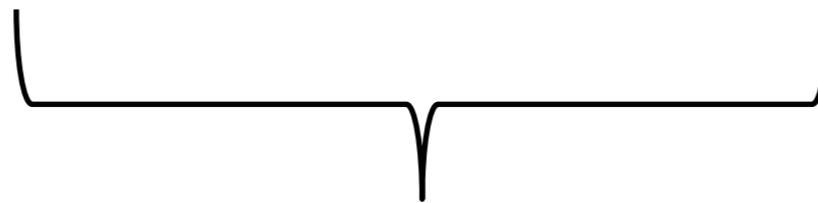
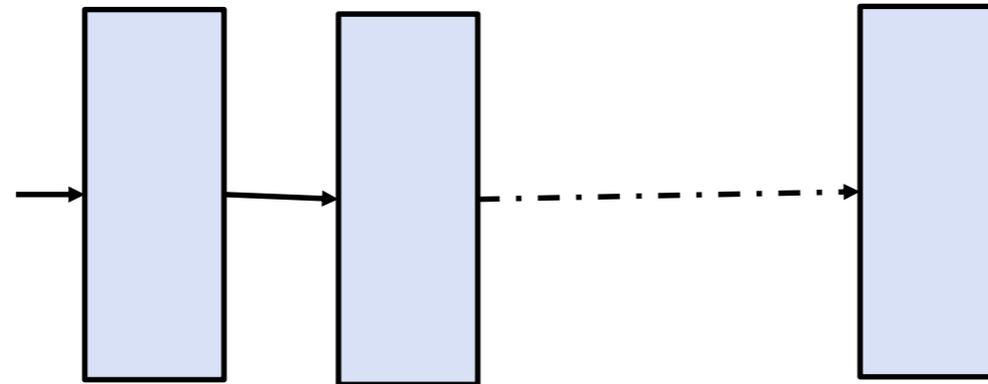
better performance: 60%

even more w/ some accuracy loss

Deep Learning: Convolutional Neural Networks



image



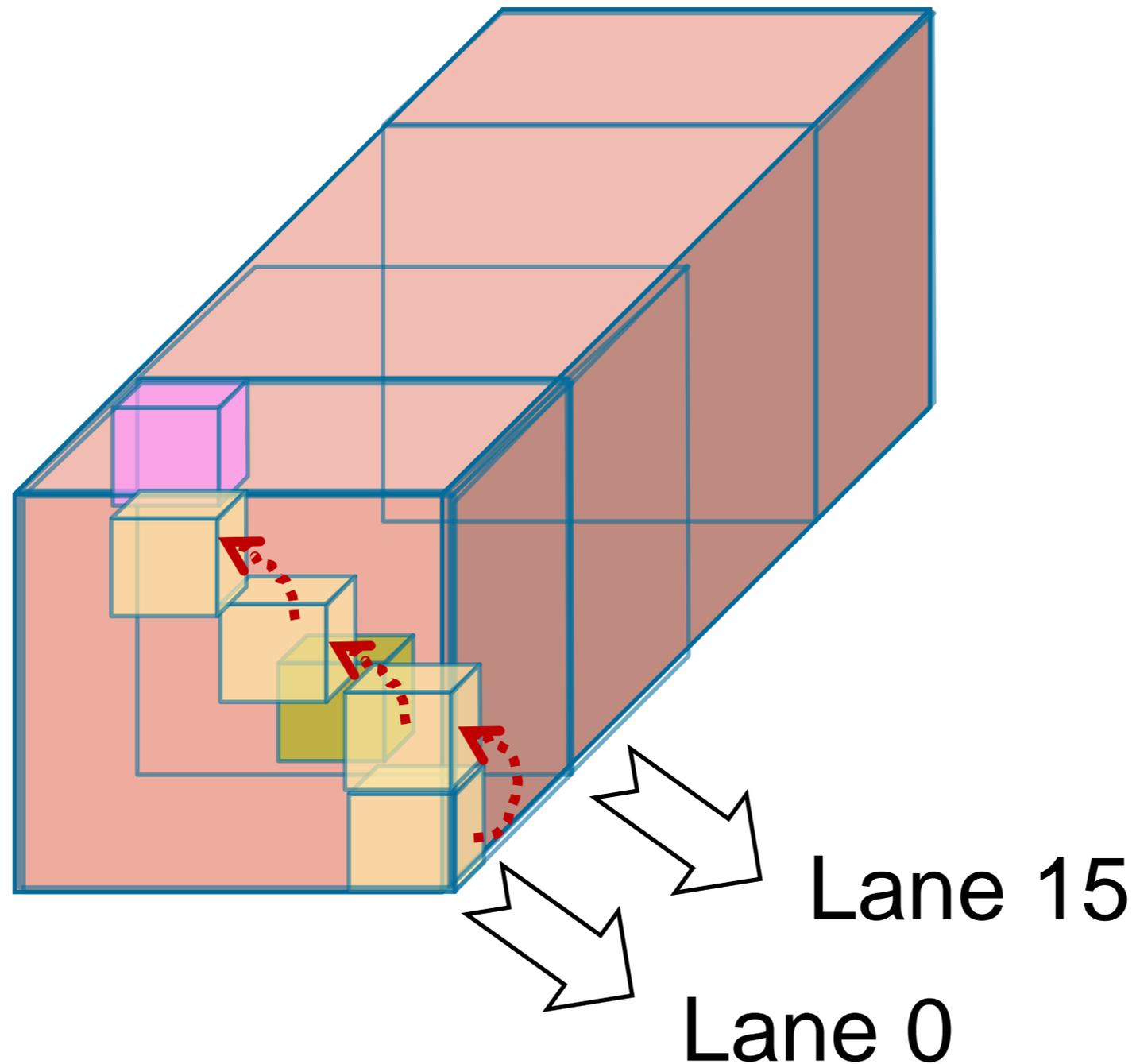
layers

10s-100

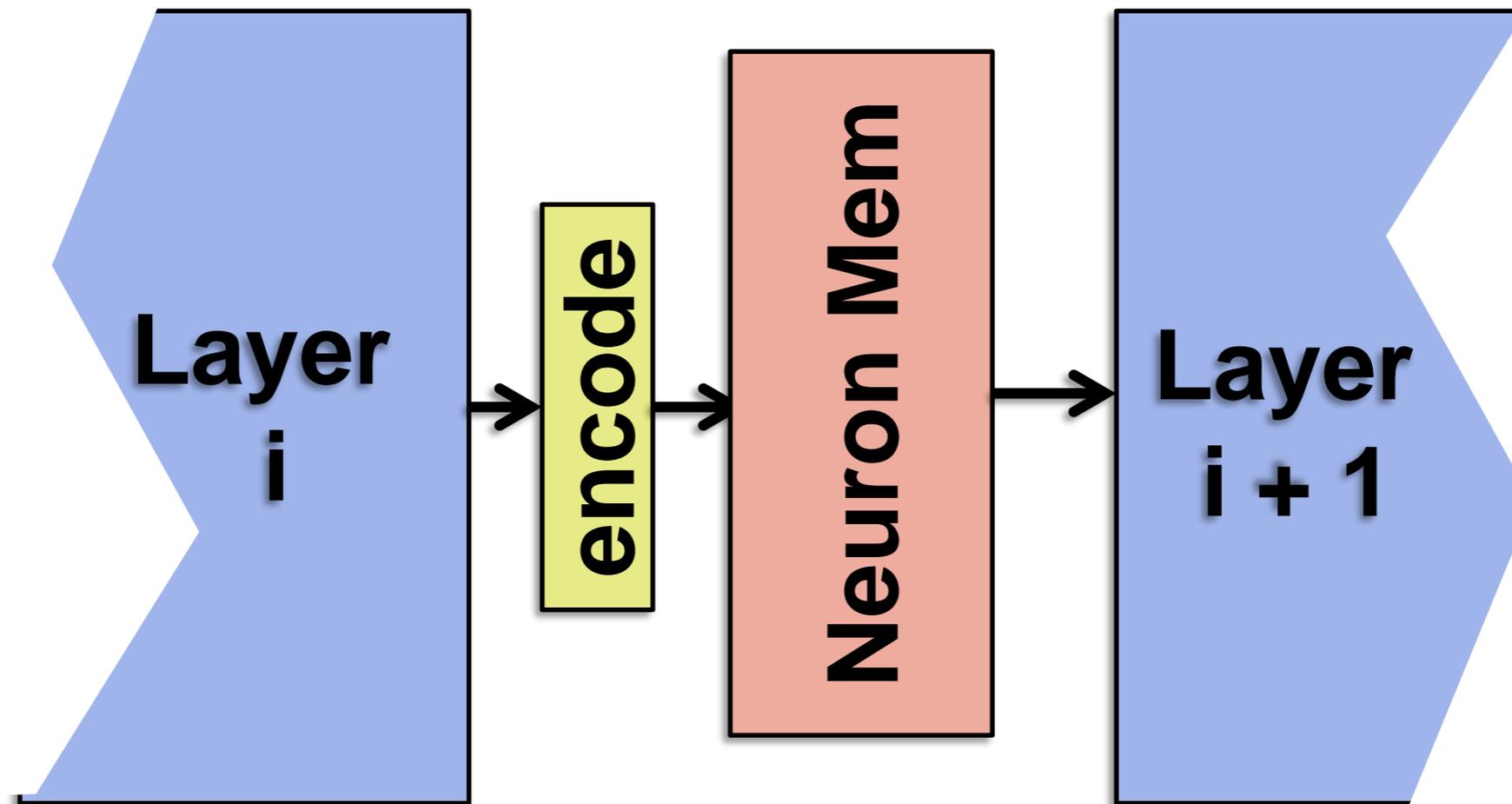
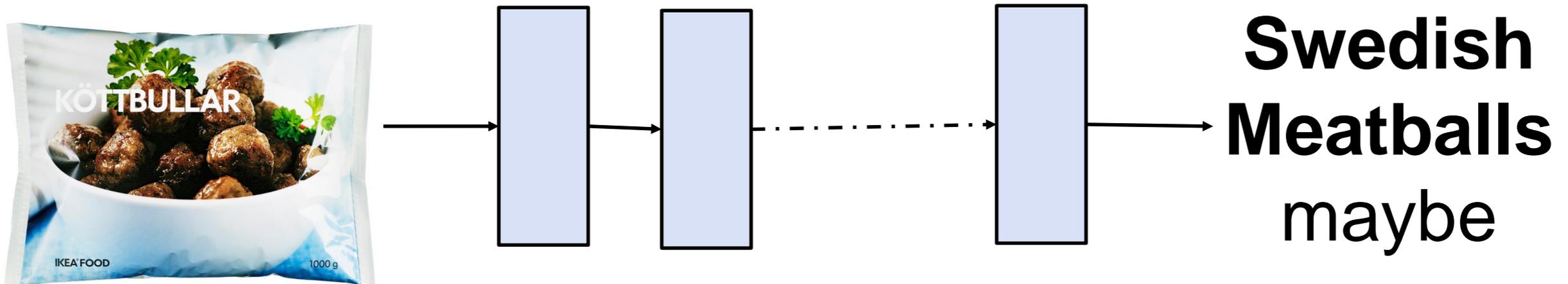
**Swedish
Meatballs
maybe**

Naïve Solution: No Wide Memory Accesses

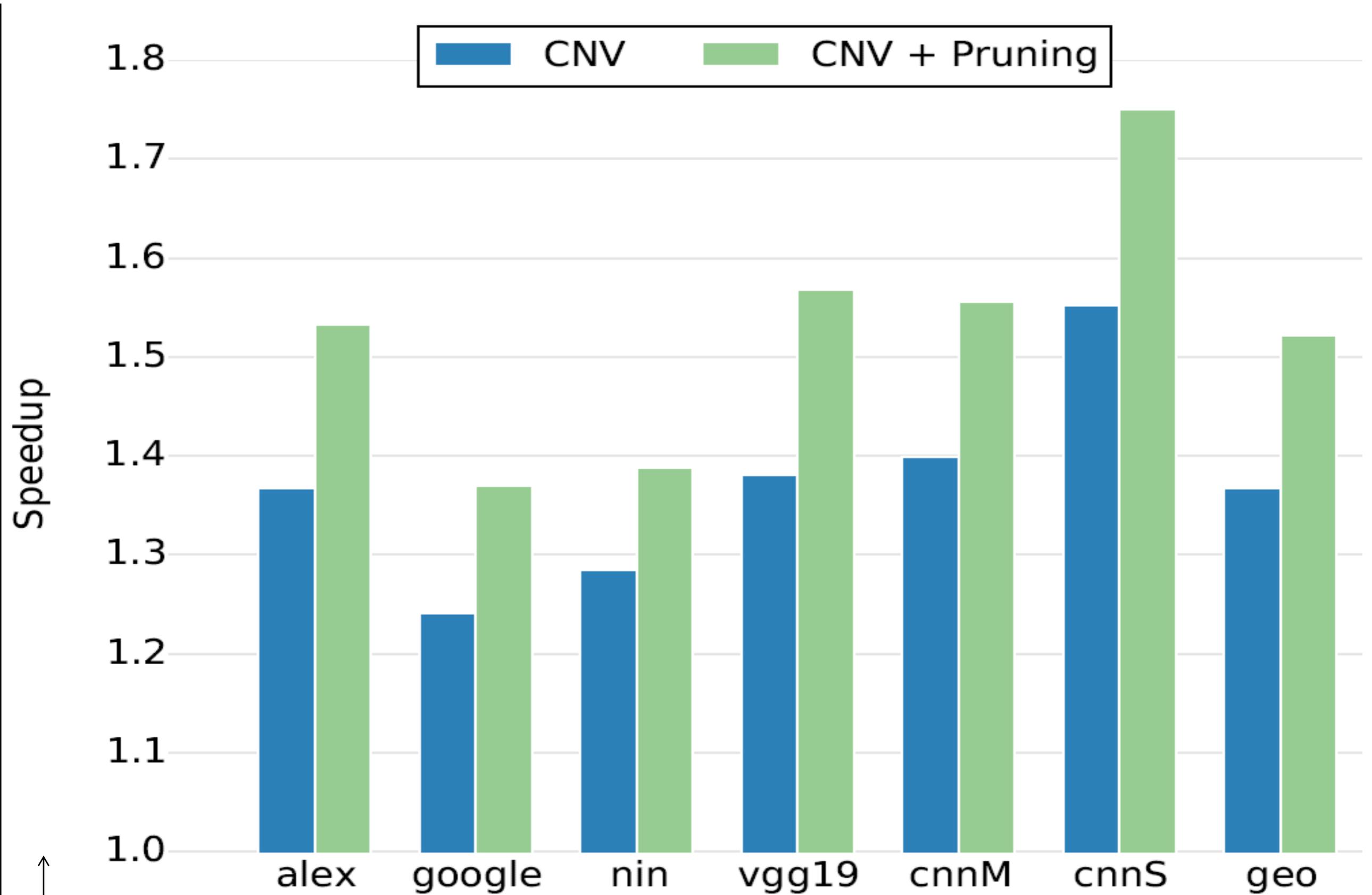
- **16 independent narrow activation streams**



Removing Zeroes: At the output of each layer

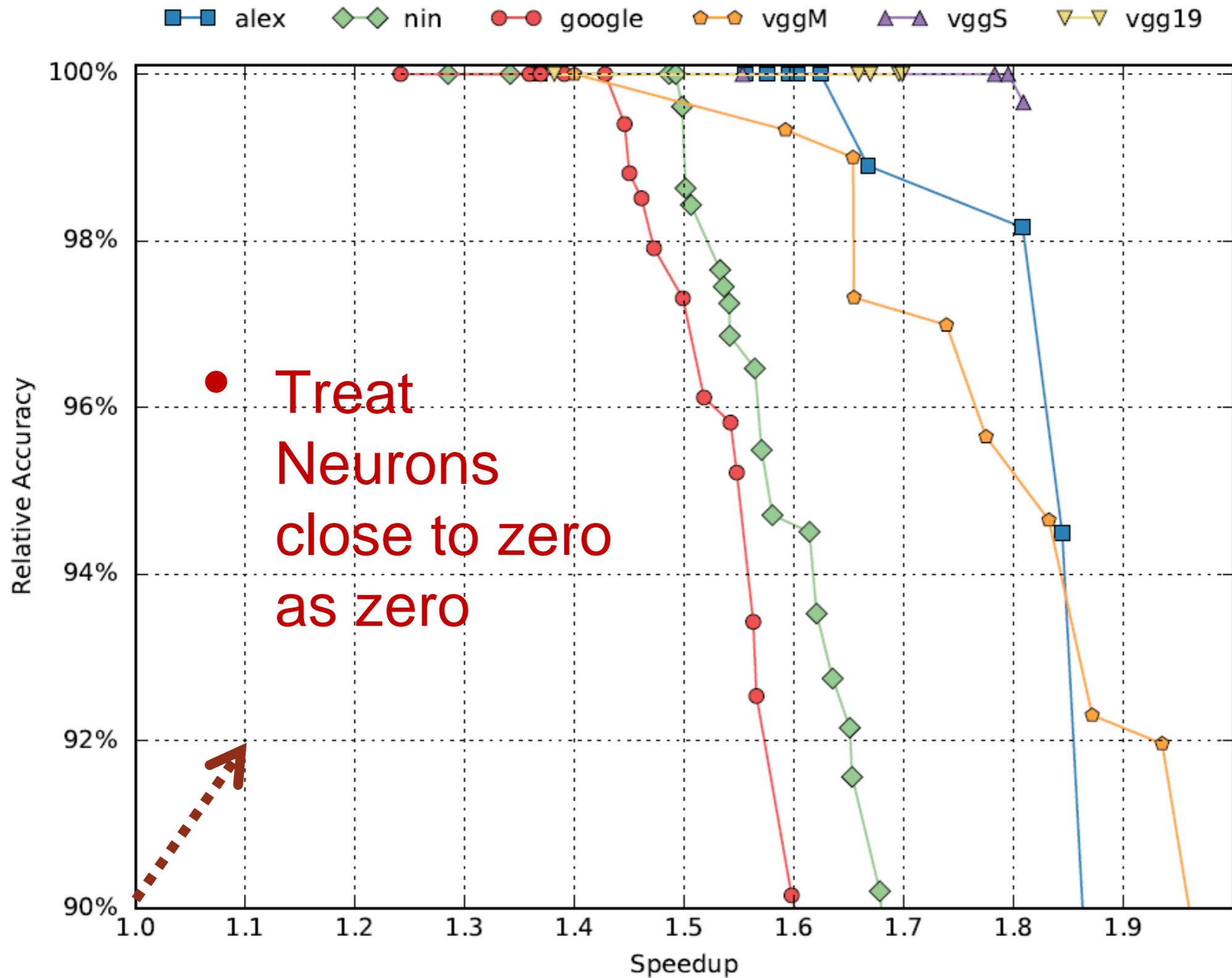


Cnvlutin: No Accuracy Loss



↑
better

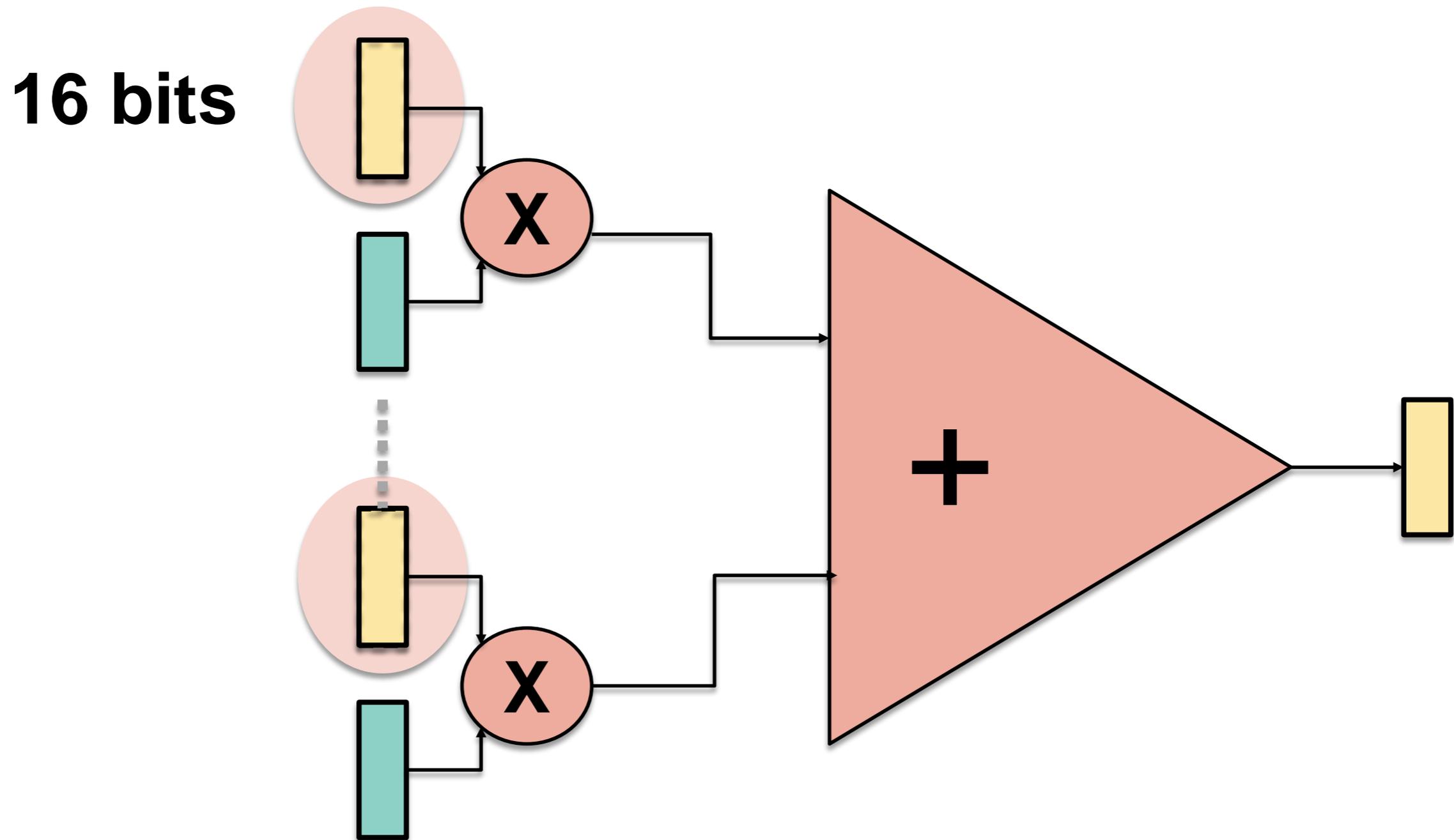
Loosening the Ineffectual Neuron Criterion



Open Questions:

Are these robust? How to find the best?

Another Property of CNNs

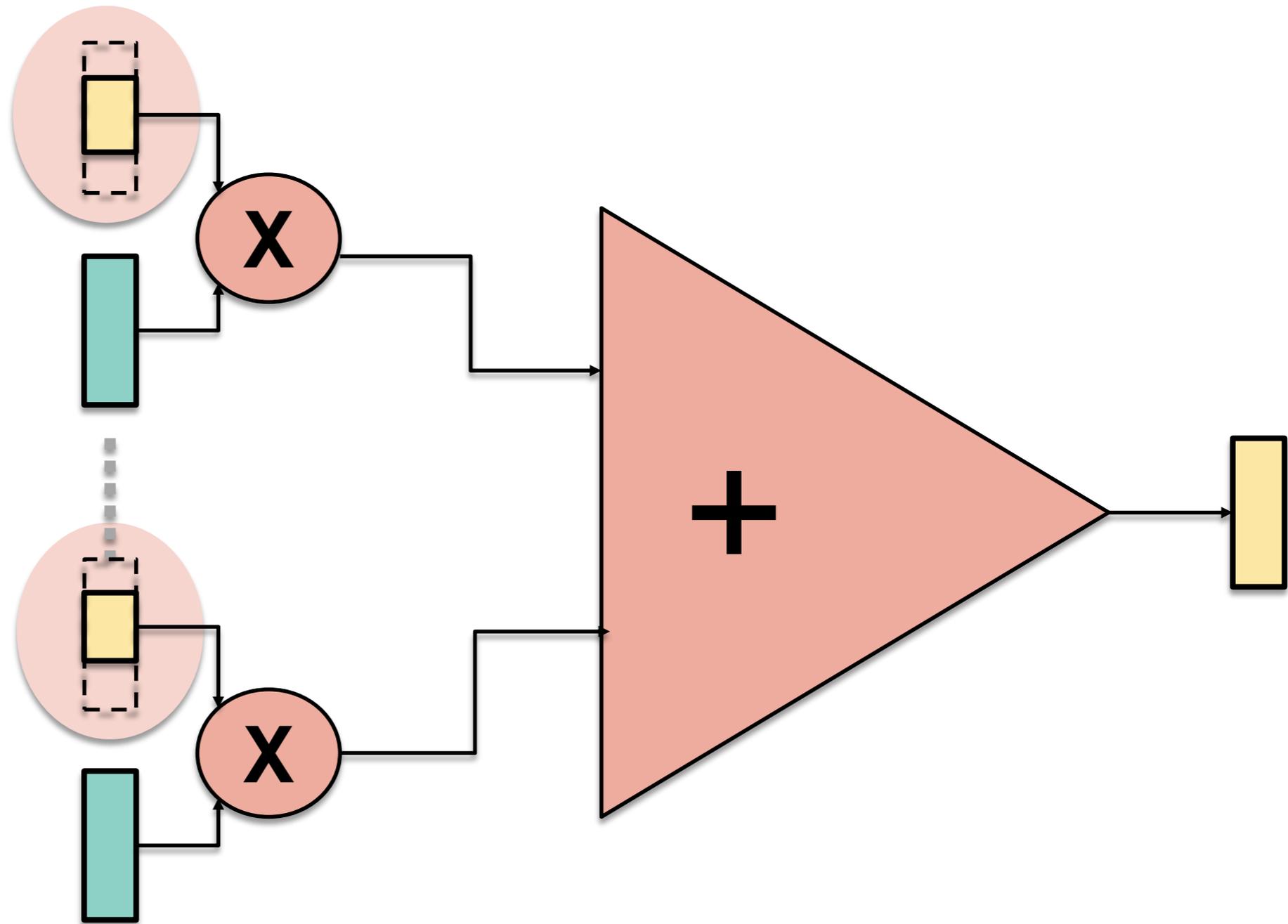


Operand Precision Required Fixed?

16 bits?

CNNs: Precision Requirements Vary

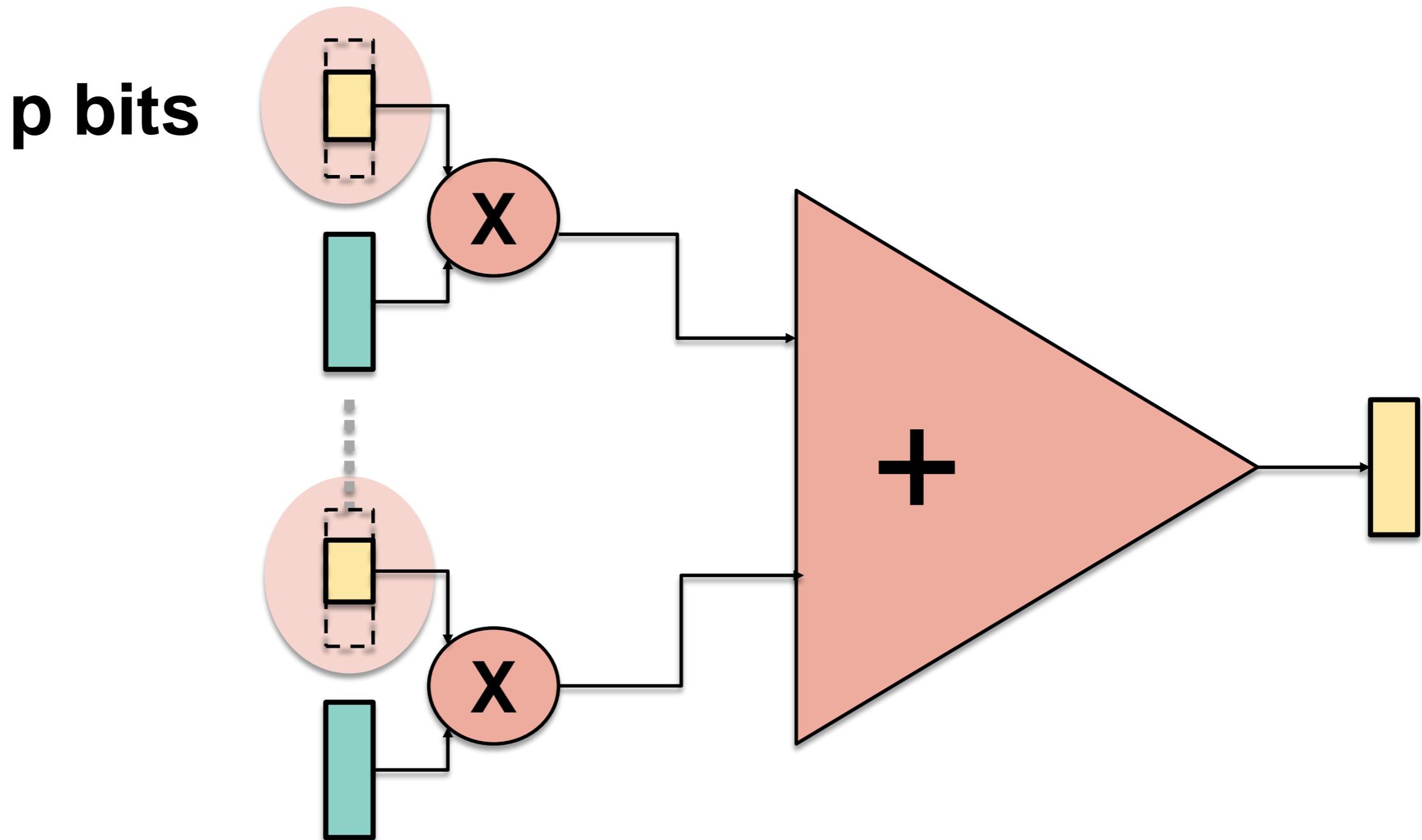
~~16 bits~~
p bits



Operand Precision Required Fixed-Varies

5 bits to 13 bits

Stripes



$$\text{Execution Time} = 16 / P$$

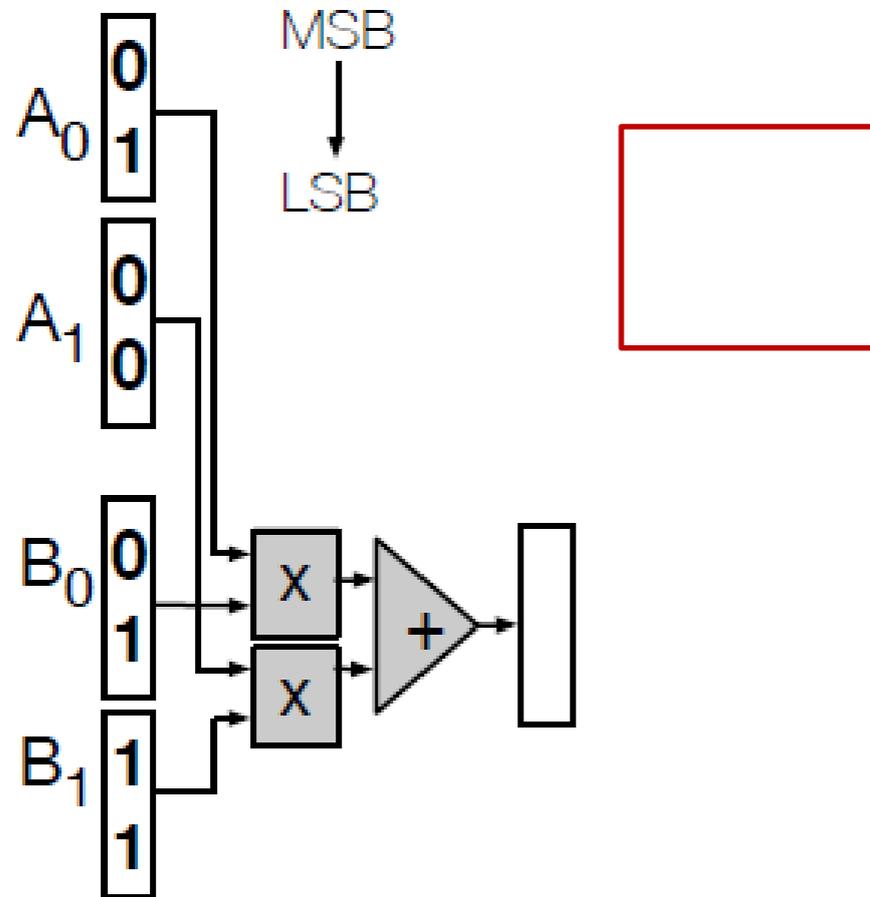
Performance + Energy Efficiency + Accuracy Knob

Stripes: Key Concept

2 2x2b
Terms/Step

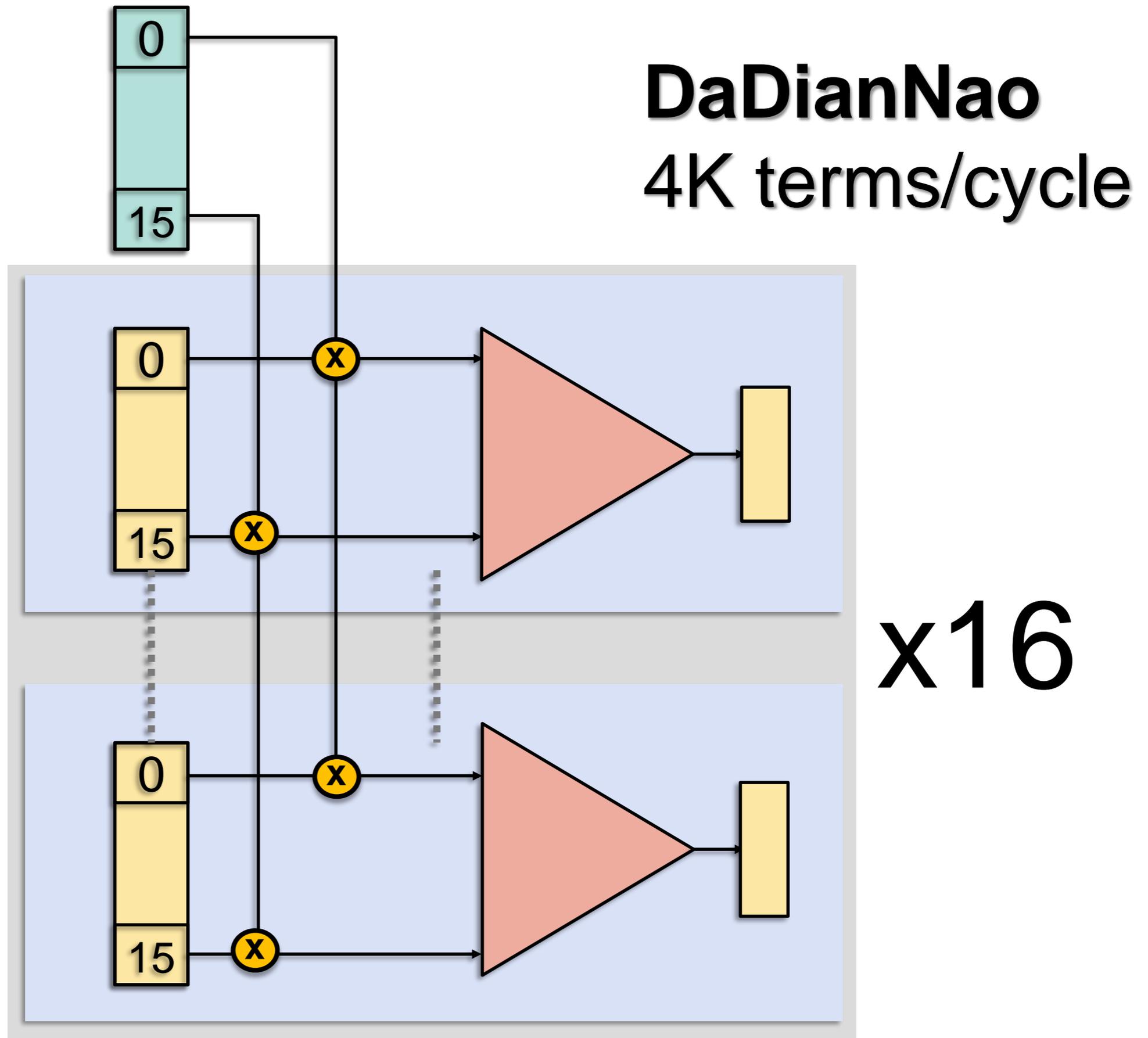
2 1x2b

4 1x2b

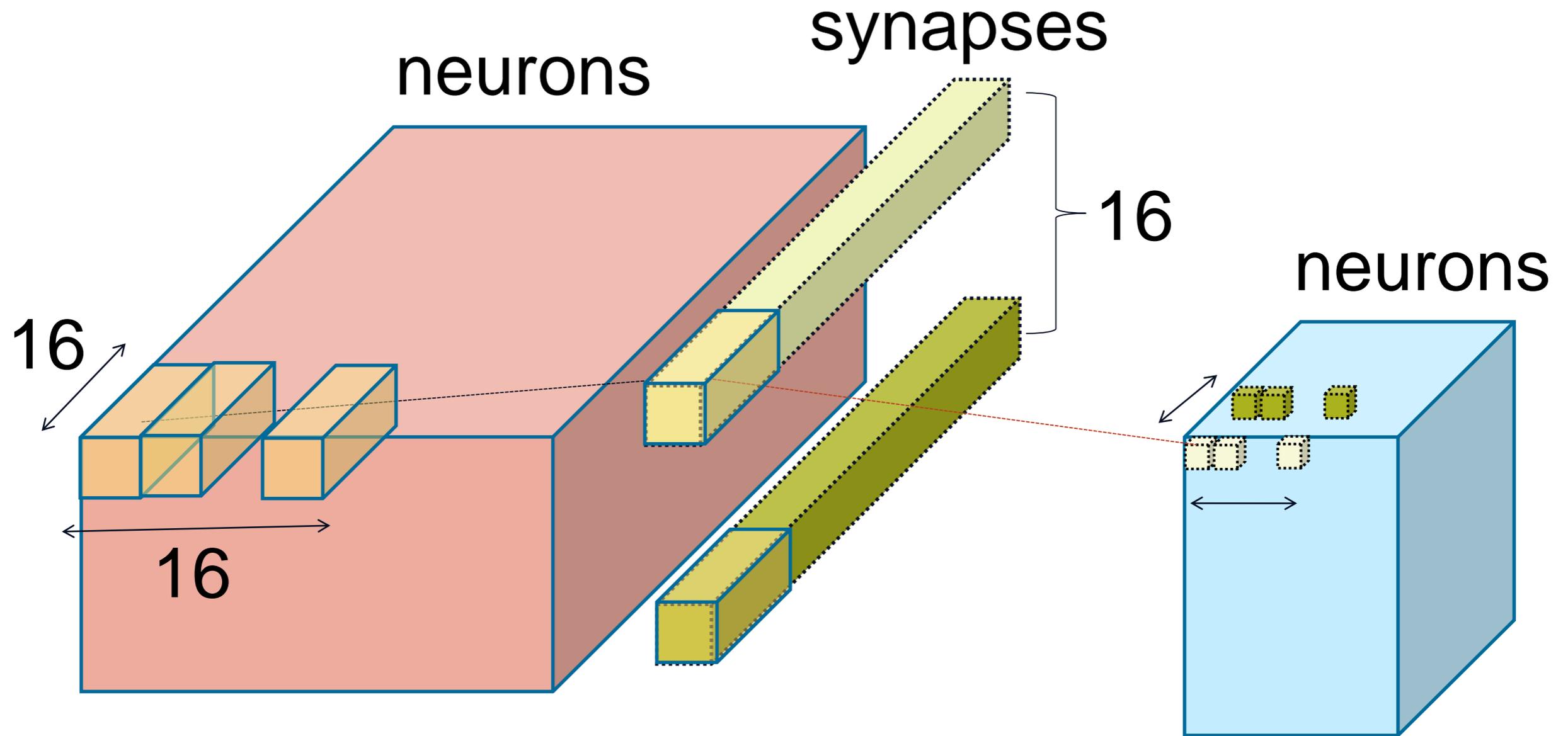


- **Devil in the Details:** Carefully chose what to serialize and what to reuse → same input wires as baseline

SIMD: Exploit Computation Structure



Compensating for Bit-Serial's Compute Bandwidth Loss



- **Each Tile:**
 - 16 Windows Concurrently – 16 neurons each
 - 16 Filters
 - 16 partial output neurons

Stripes

No Accuracy Loss

+192% performance*

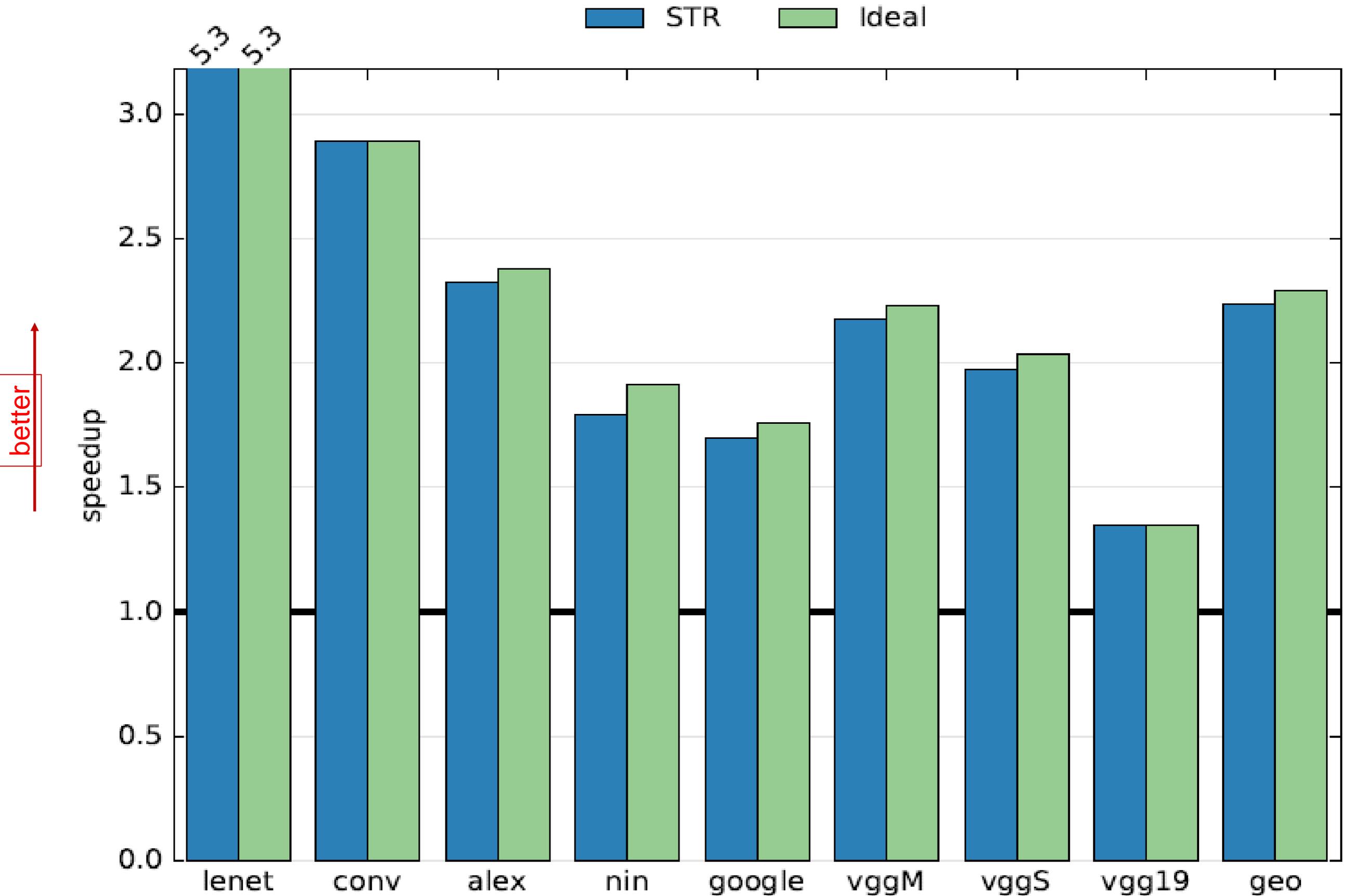
-57% energy

+32% area

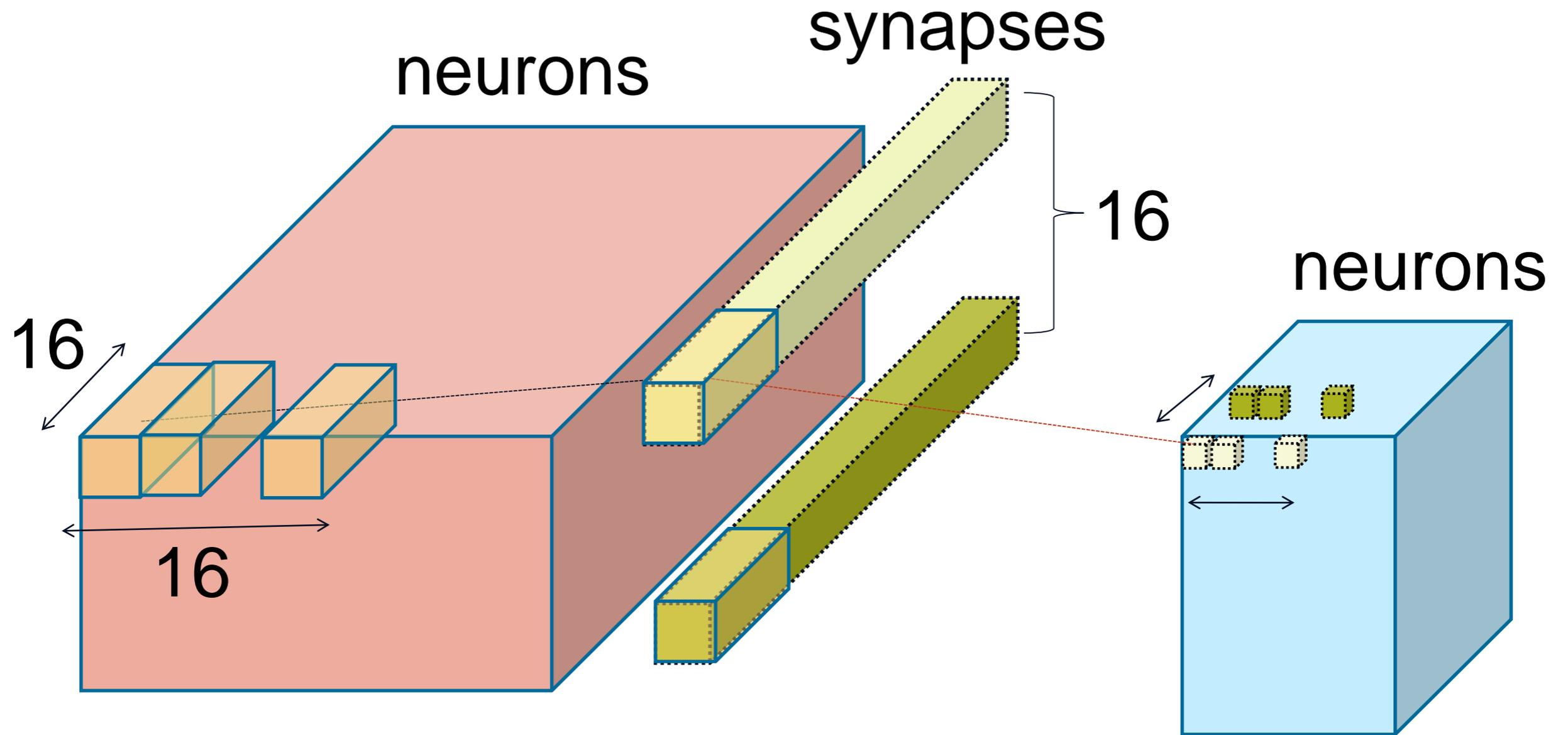
More performance w/ accuracy loss

* W/O Older: LeNet + Covnet

Stripes: Performance Boost

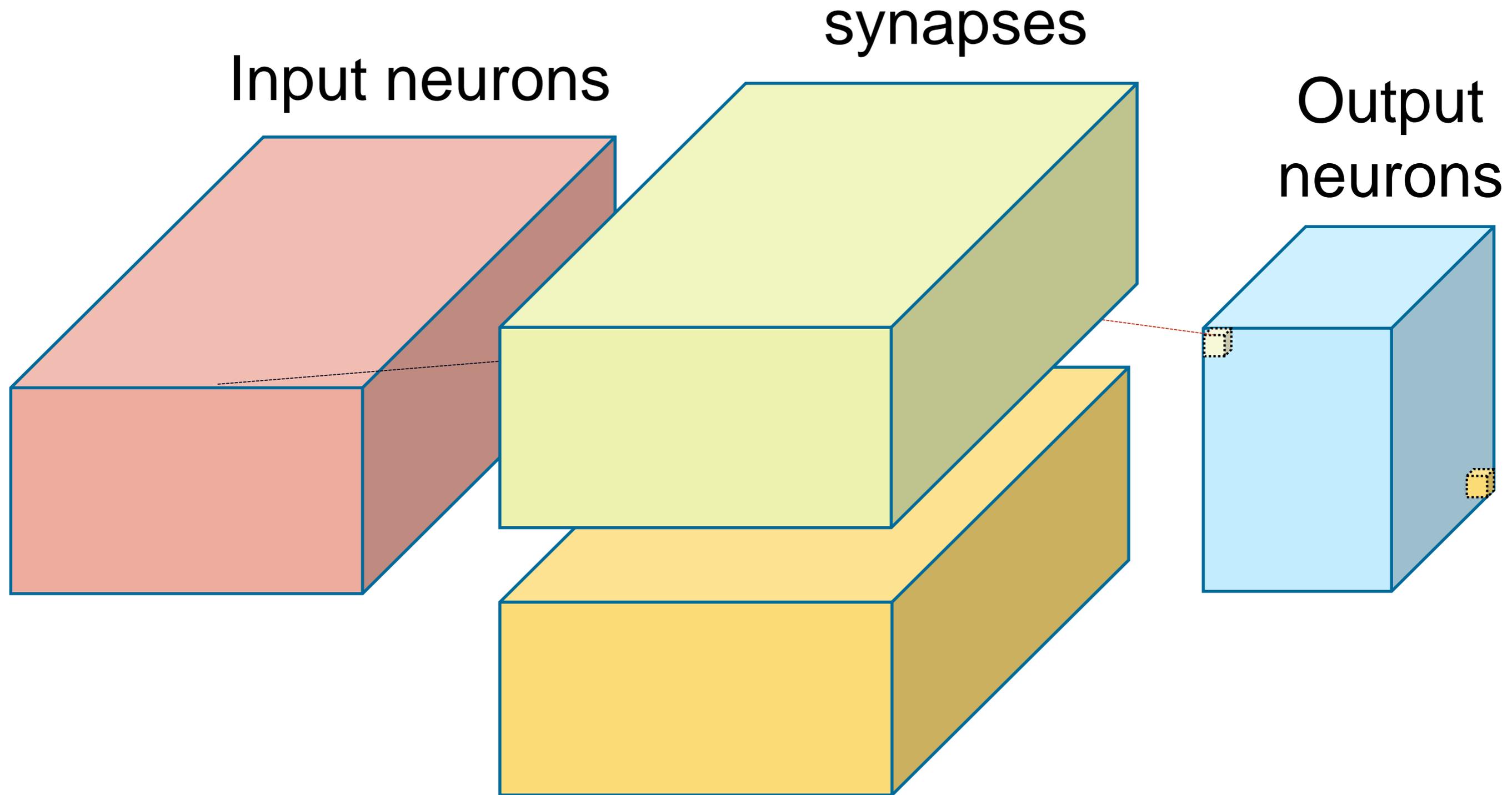


Fully-Connected Layers?



- **Each Tile:**
 - No Weight Reuse
- Cannot Have 16 Windows

Fully-Connected Layers

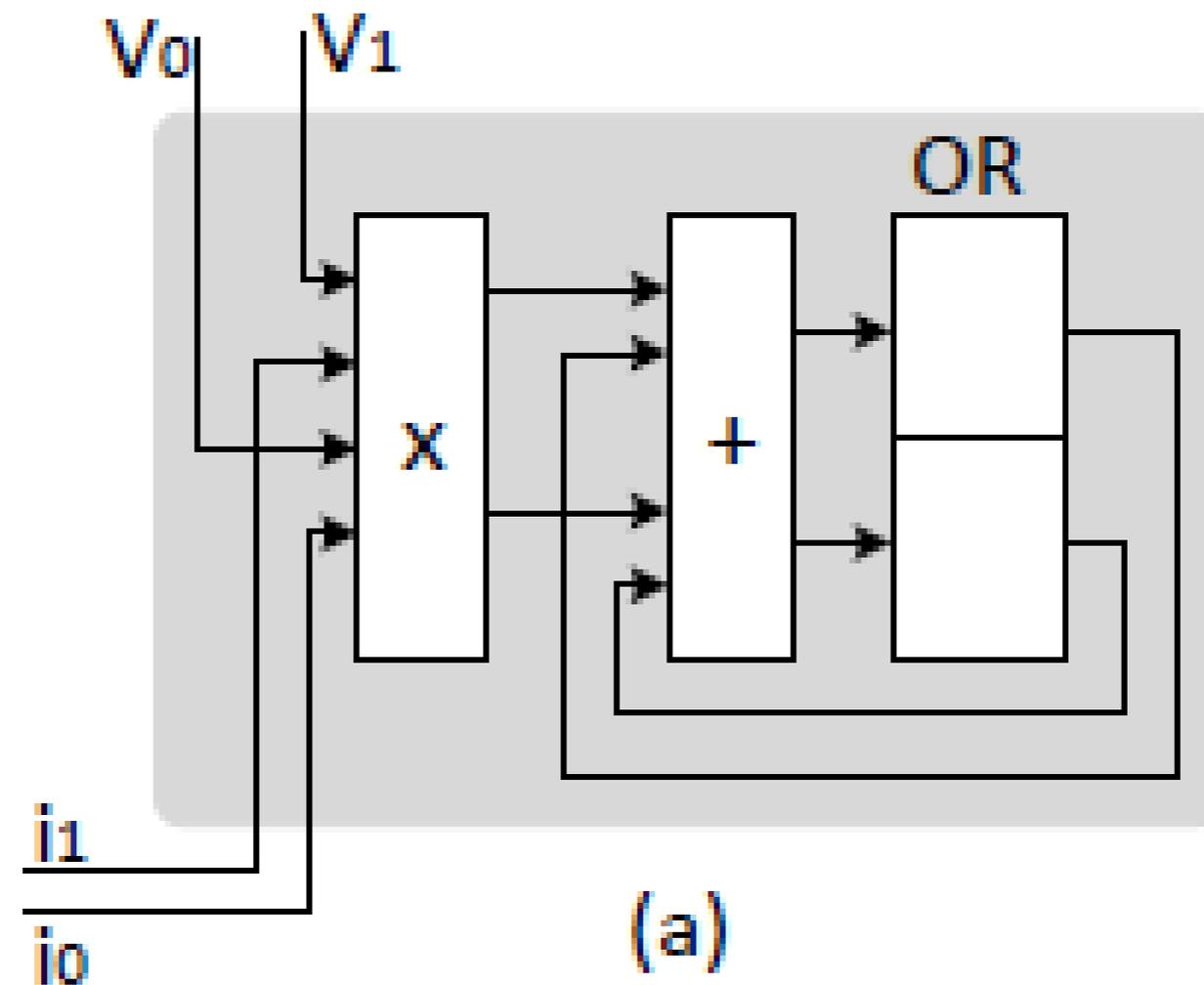


- No Weight Reuse
- Cannot Have 16 Windows

TARTAN: Accelerating Fully-Connected Layers

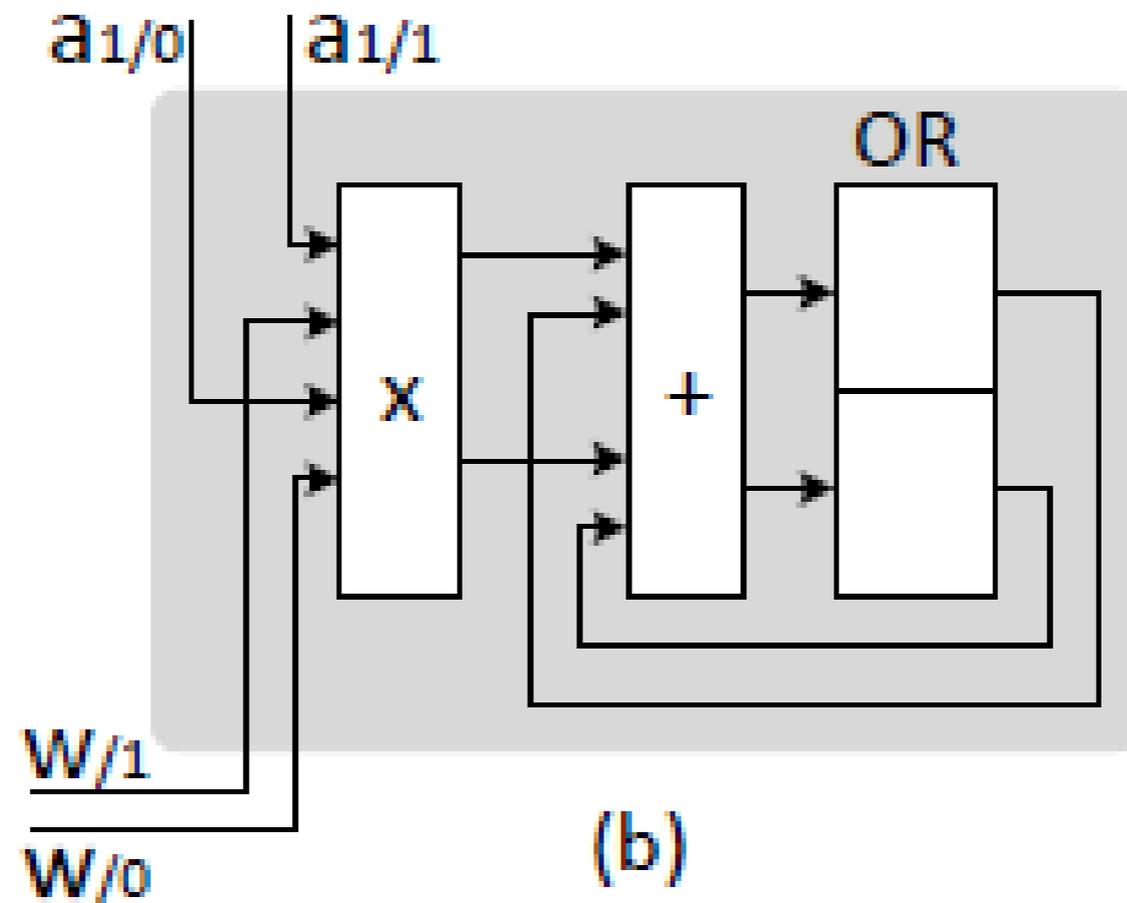
- **Bit-Parallel Engine**

- V: activation
- I: weight
- Both 2 bits



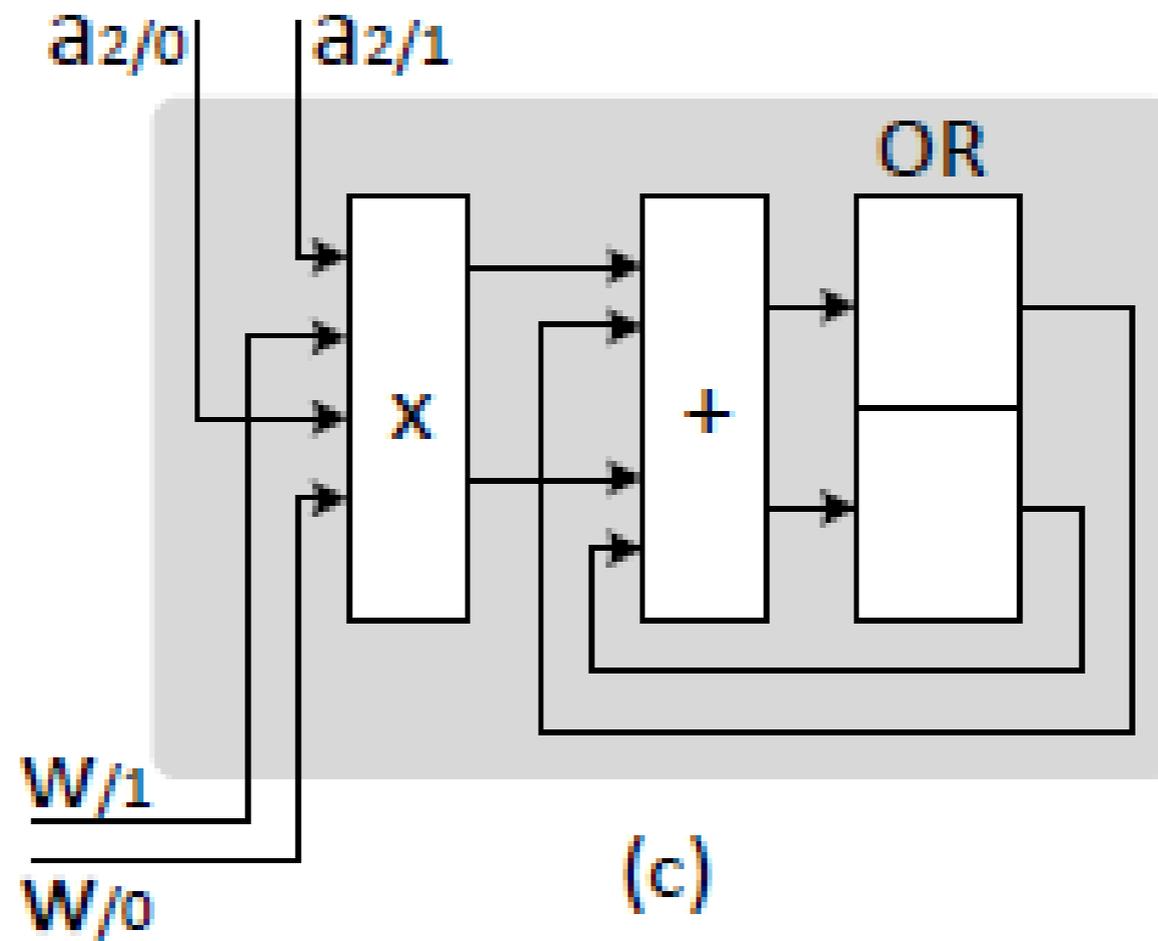
Bit-Parallel Engine: Processing one Activation x Weight

- **Cycle 1:**
 - *Activation: a_1 and Weight: W*



Bit-Parallel Engine: Processing Another Pair

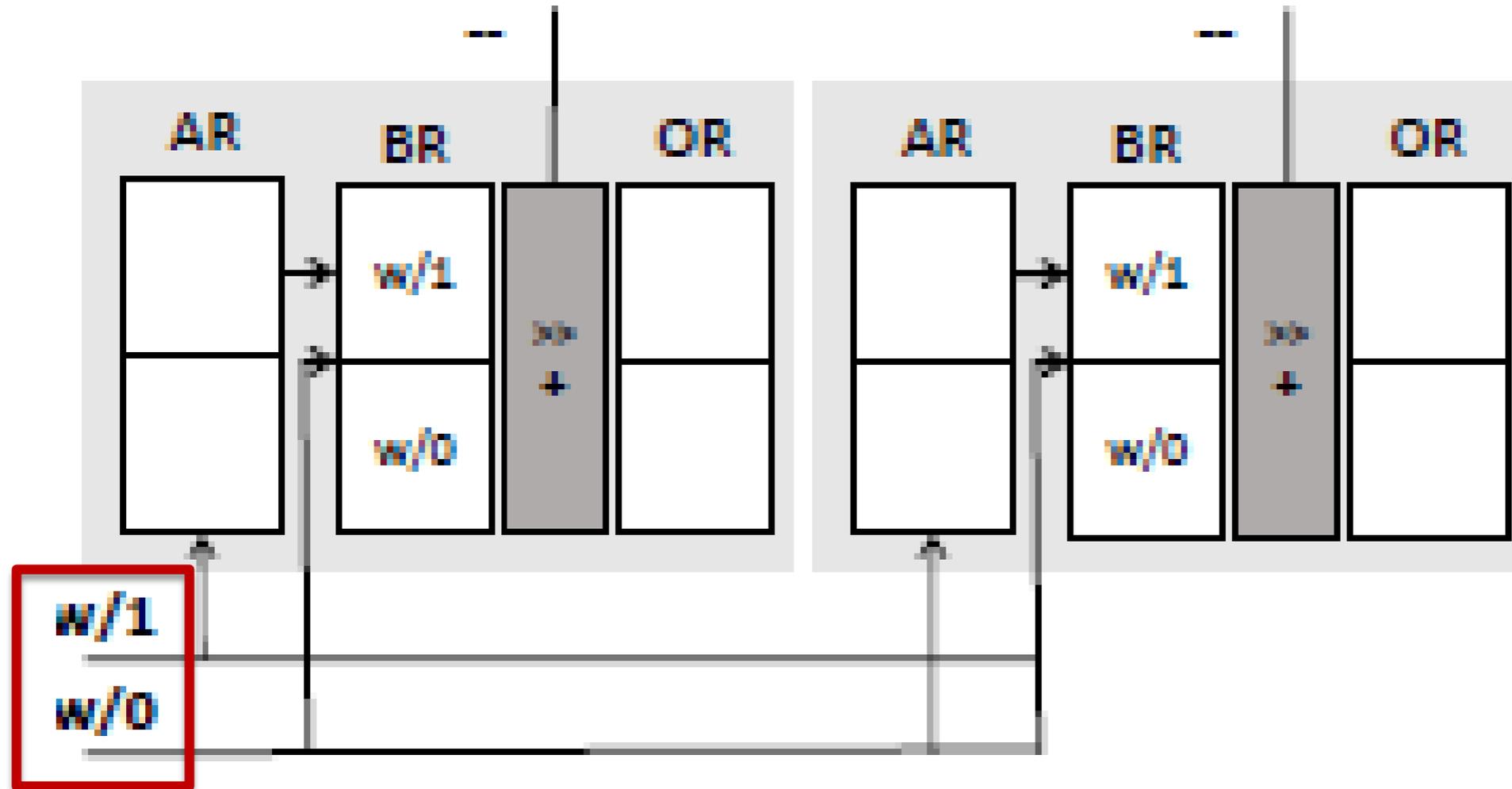
- **Cycle 2:**
 - *Activation: a_2 and Weight: W*



- **$a_1 \times W + a_2 \times W$ over two cycles**

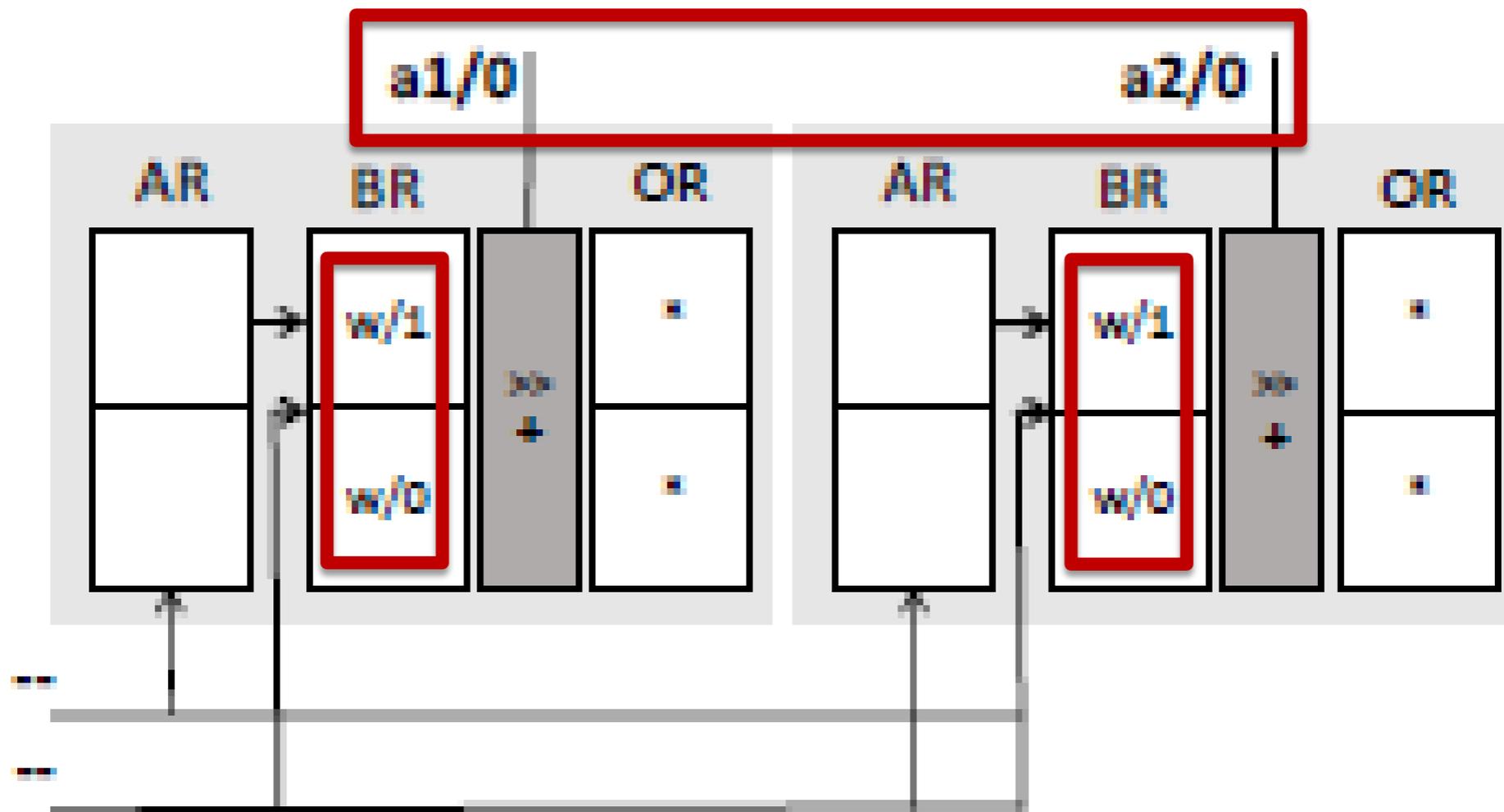
TARTAN: Convolutional Layer Processing

- Cycle 1: load 2b weight into BRs



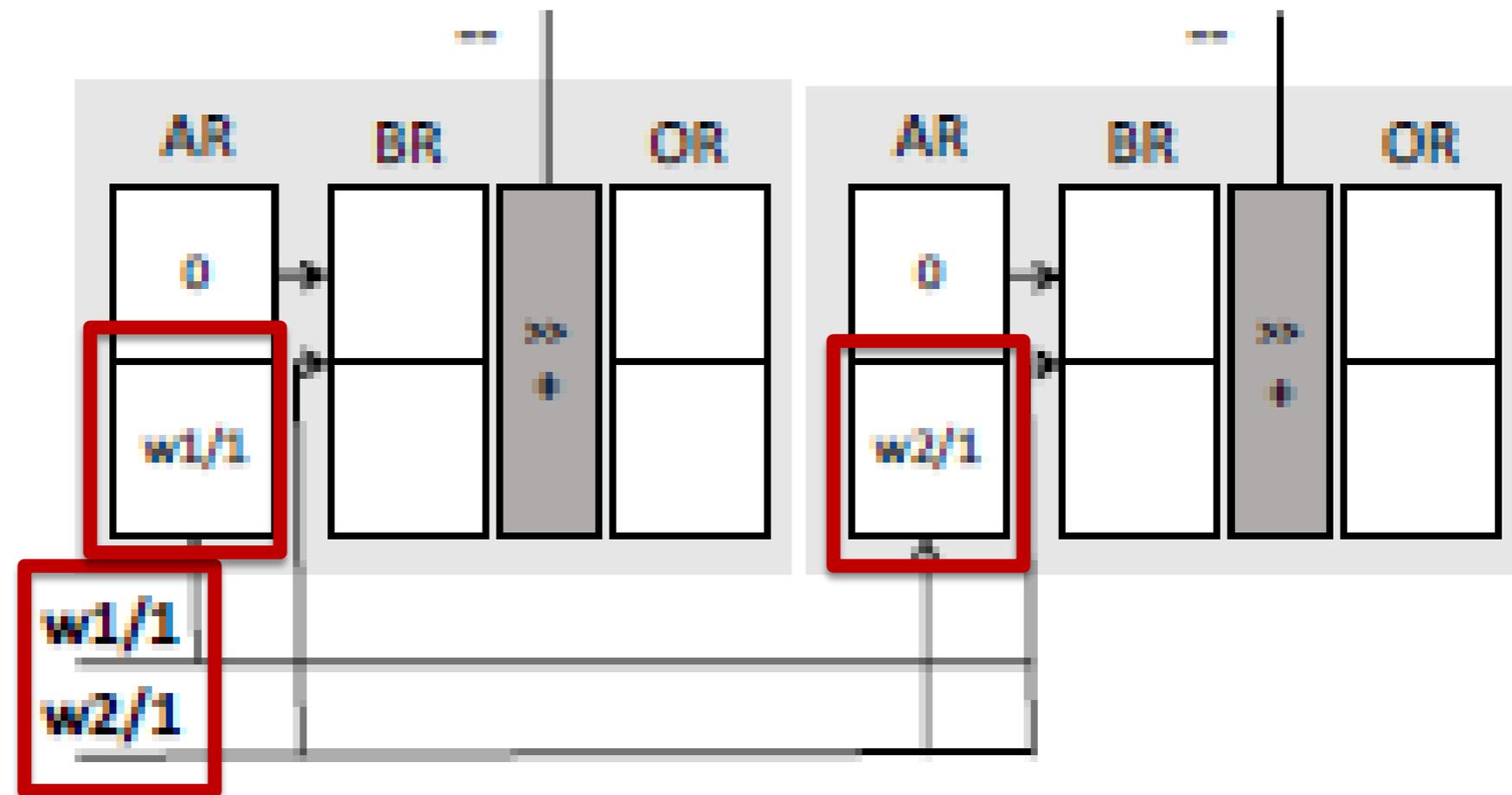
TARTAN: Weight x 1st bit of Two Activations

- Cycle 2: Multiply W with bit 1 of activations a_1 and a_2



TARTAN: Fully-Connected Layers: Loading Weights

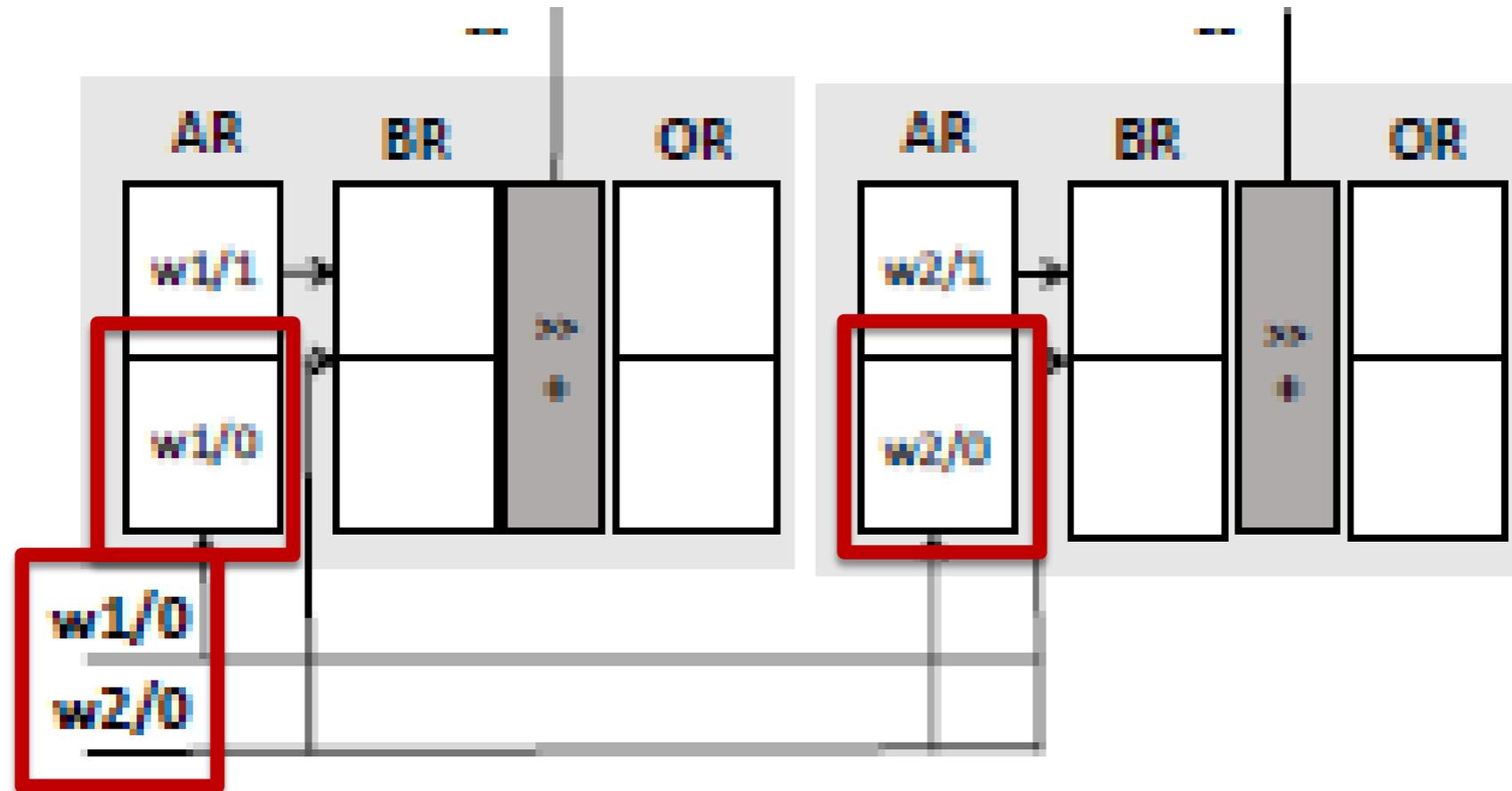
- What is different? Weights cannot be reused
- Cycle 1: Load first bit of two weights into Ars



Bit 1 of Two Different Weights

TARTAN: Fully-Connected Layers: Loading Weights

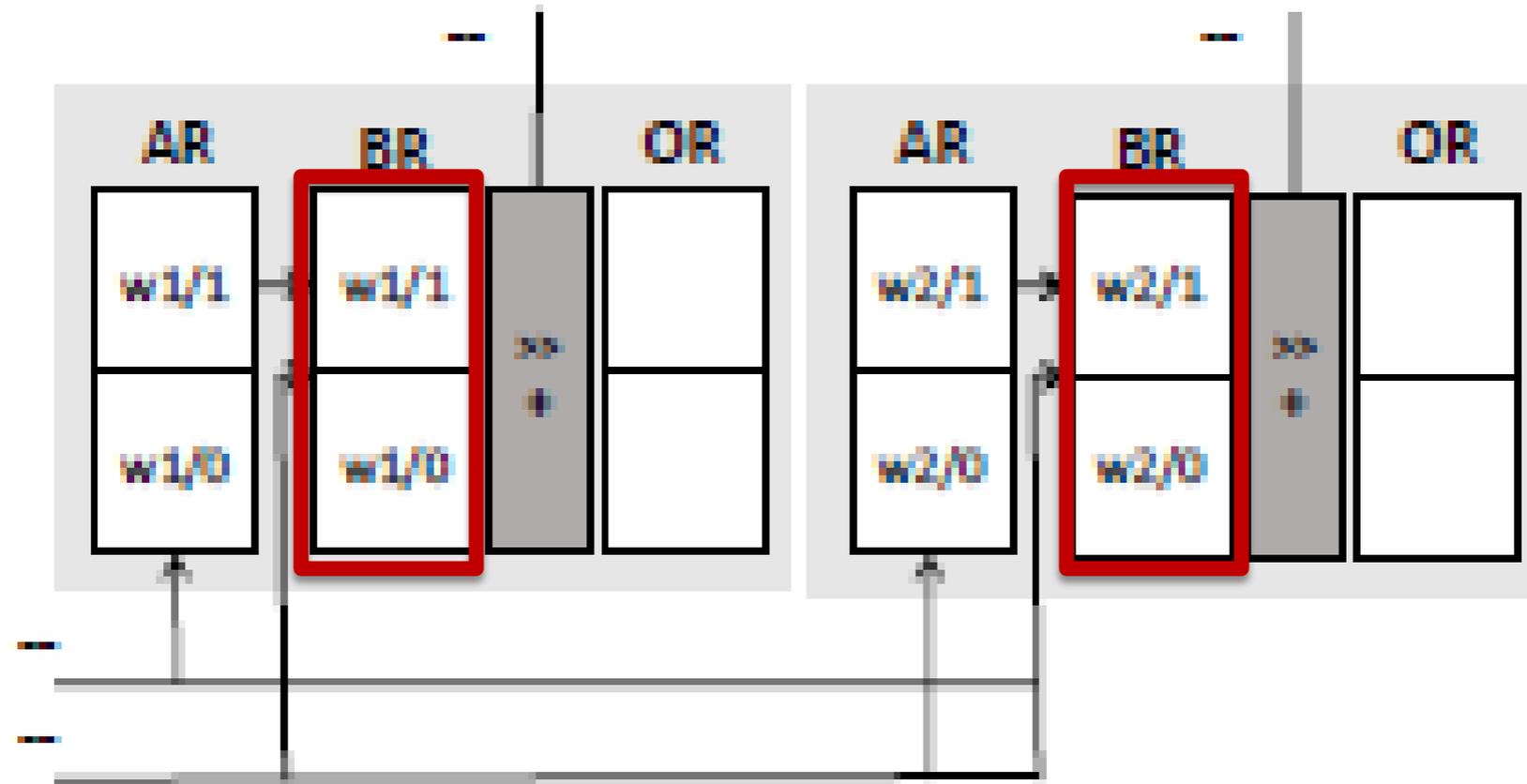
- **Cycle 2: Load 2nd bit of w1 and w2 into ARs**



- **Bit 2 of Two Different Weights**
- **Loaded Different Weights to Each Unit**

TARTAN: Fully-Connected Layers: Processing Activations

- **Cycle 3: Move AR into BR and proceed as before over two cycles**



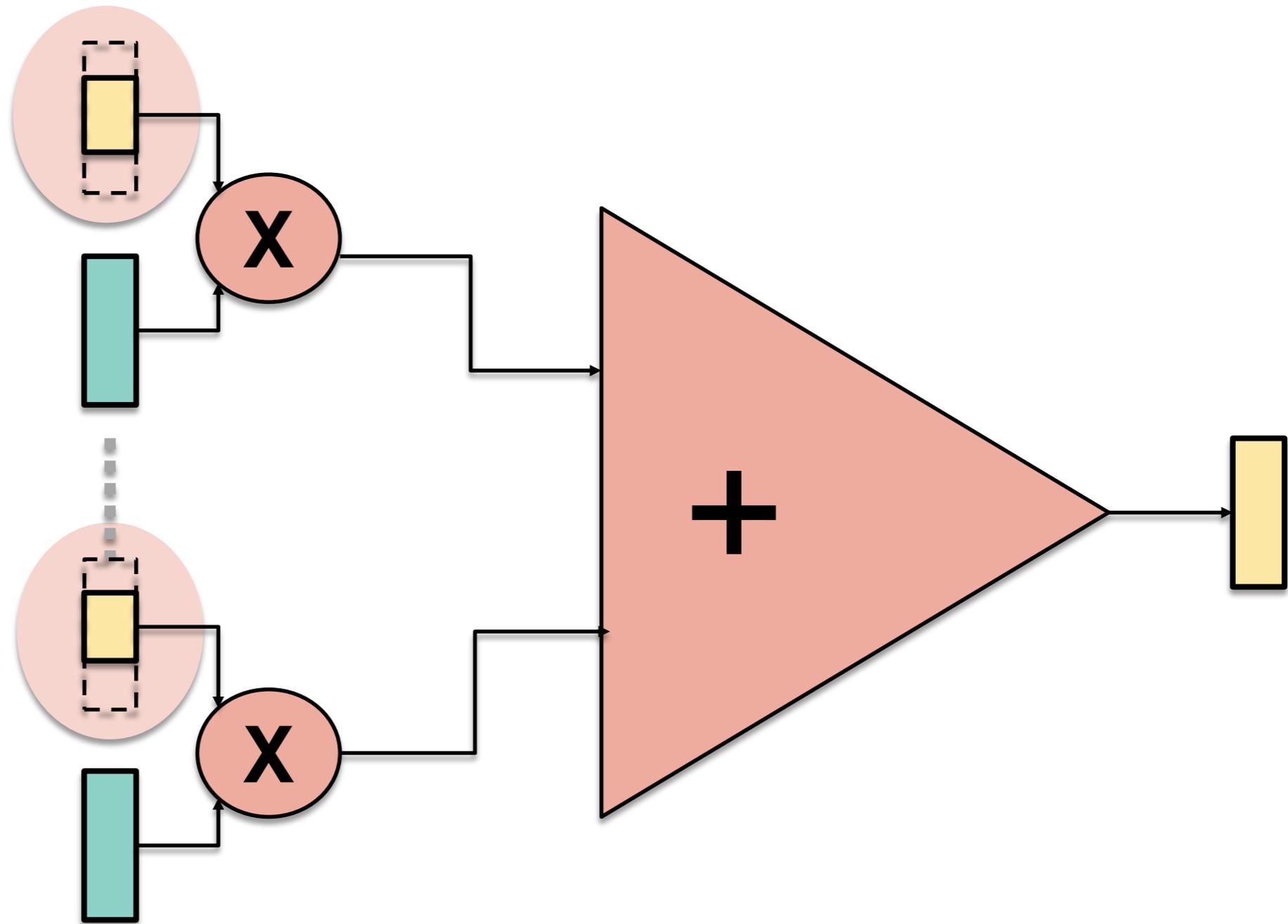
- **5-stage pipeline to do:**
 - *TWO of (2b activation x 2b weight)*

TARTAN: Result Summary

- **Bit-Serial TARTAN**
 - **2.04x faster** than DaDiannao
 - **1.25x more energy efficient** at the same frequency
 - **1.5x area overhead**

- **2-bit at-a-time TARTAN**
 - **1.6x faster** over DaDiannao
 - Roughly same energy efficiency
 - **1.25x area overhead**

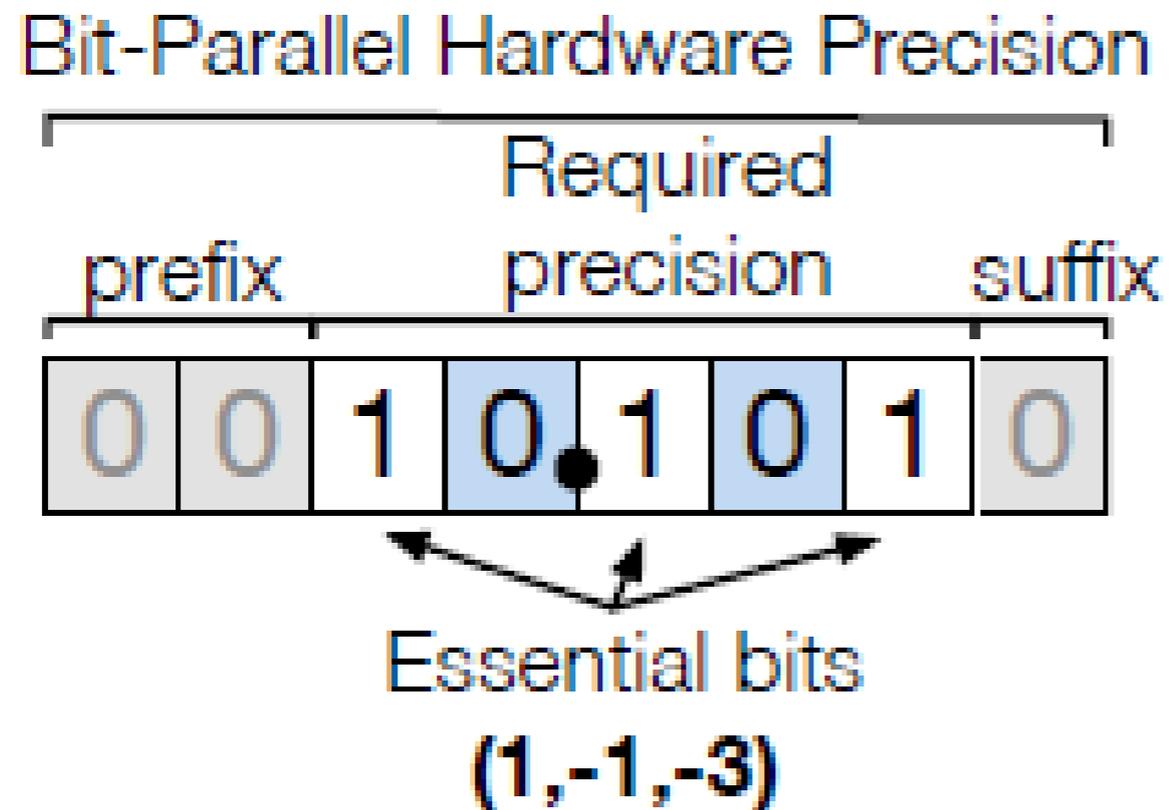
Bit-Pragmatic Engine



Operand Information Content Varies

Inner-Products

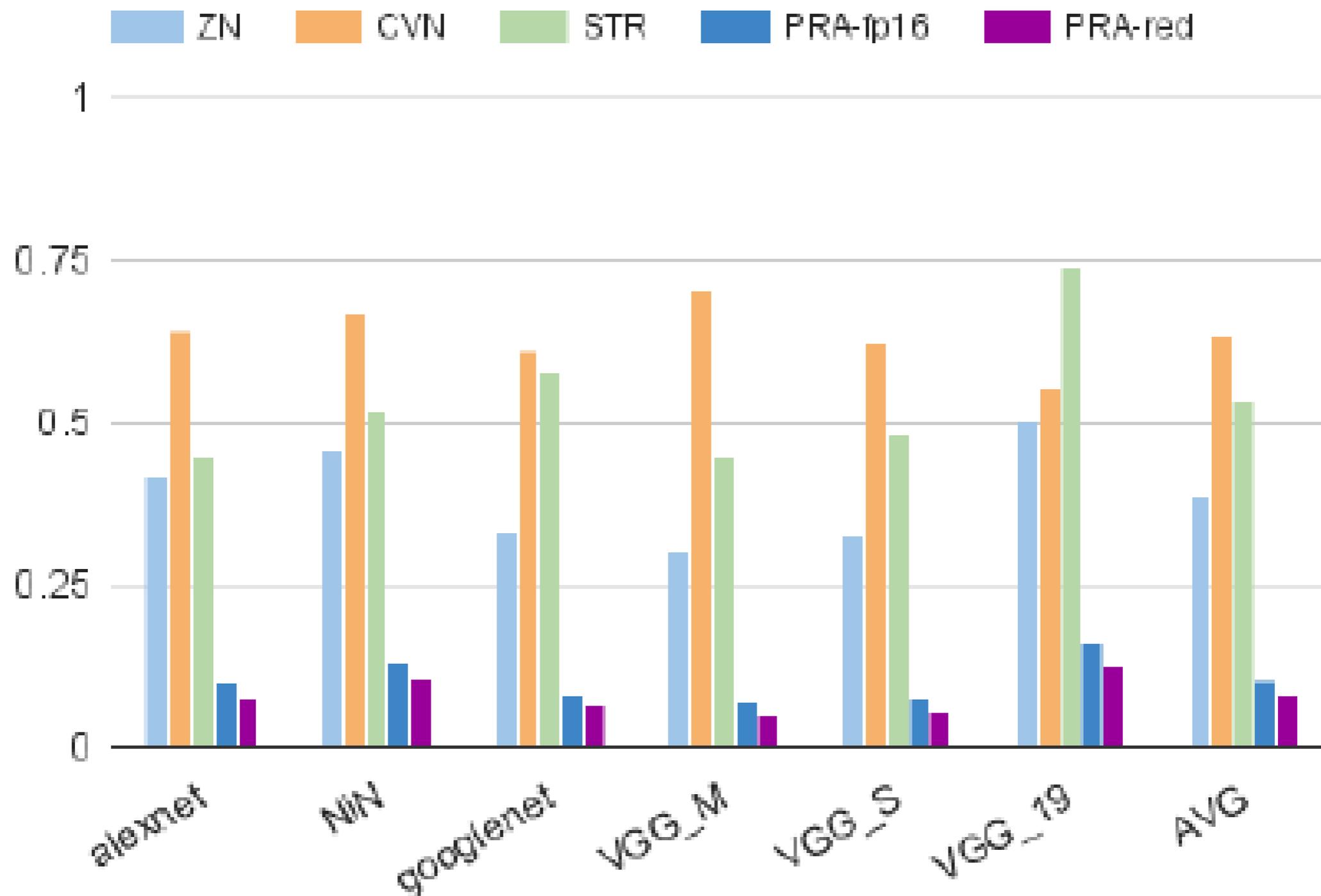
- Want to $A \times B$
- Let's look at A



$\times B$

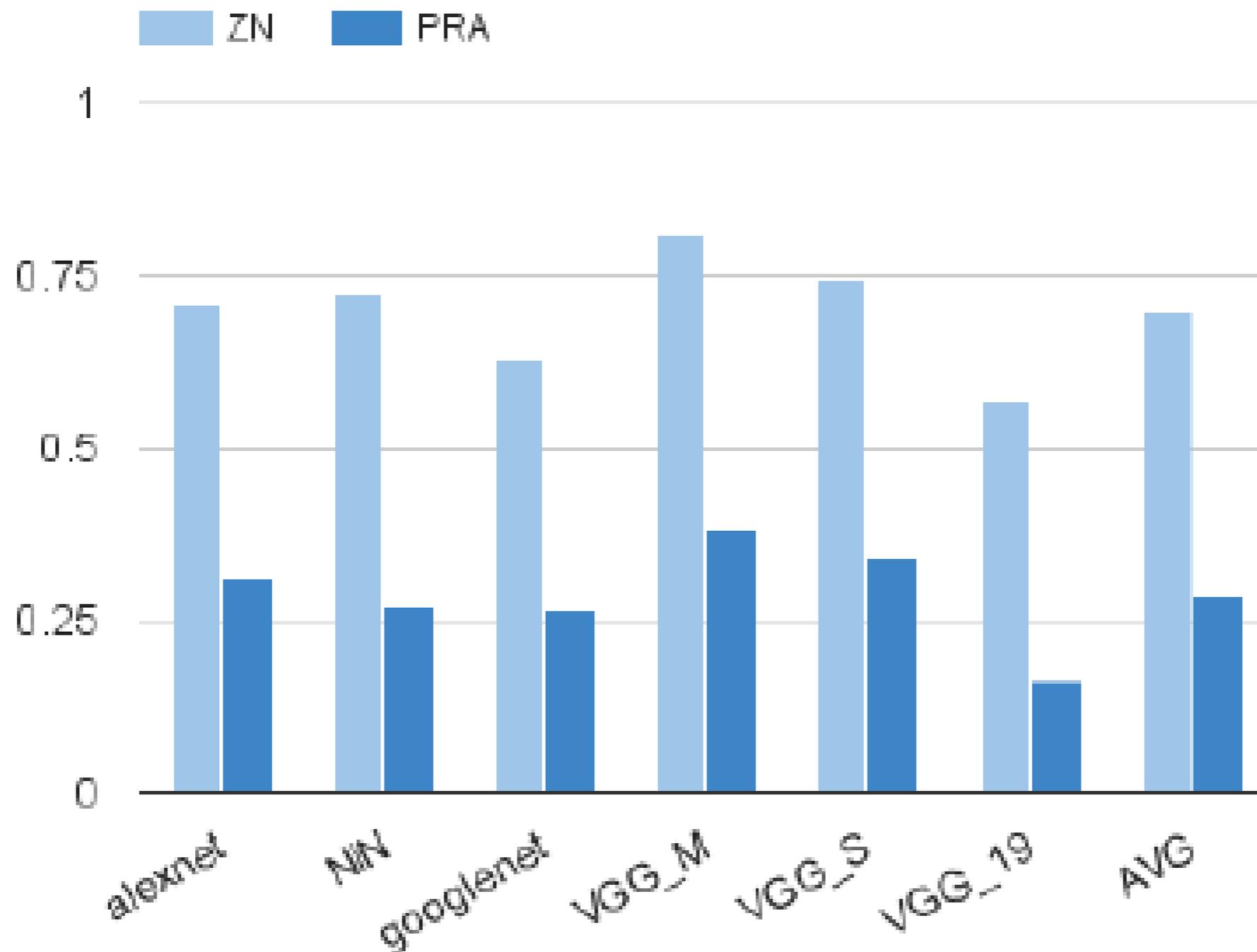
- Which bits really matter?

Zero Bit Content: 16-bit fixed-point



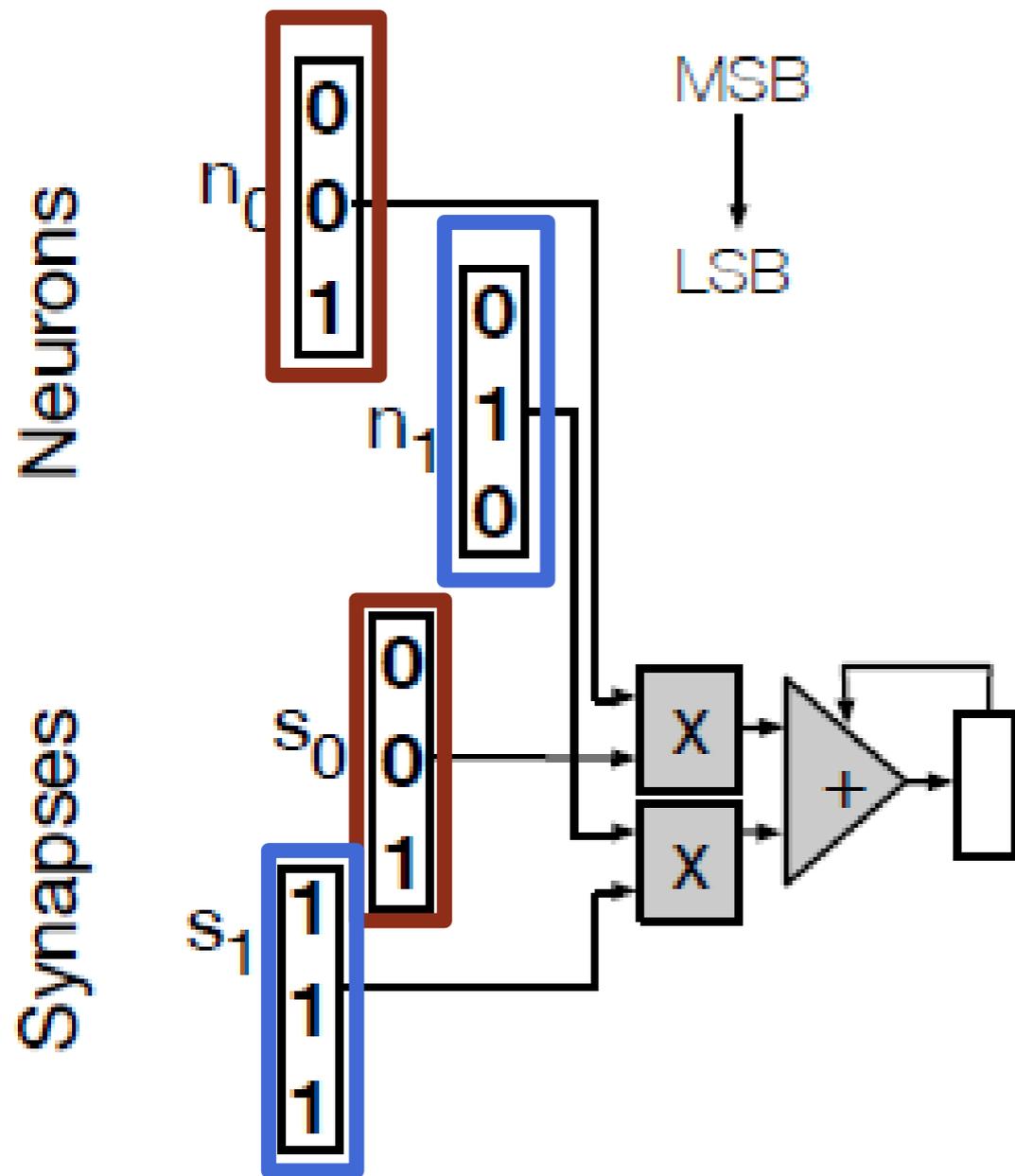
- **Only 8% of bits are non-zero once precision is reduced**
- 15%-10% otherwise

Zero Bit Content: 8-bit Quantized (Tensorflow-like)

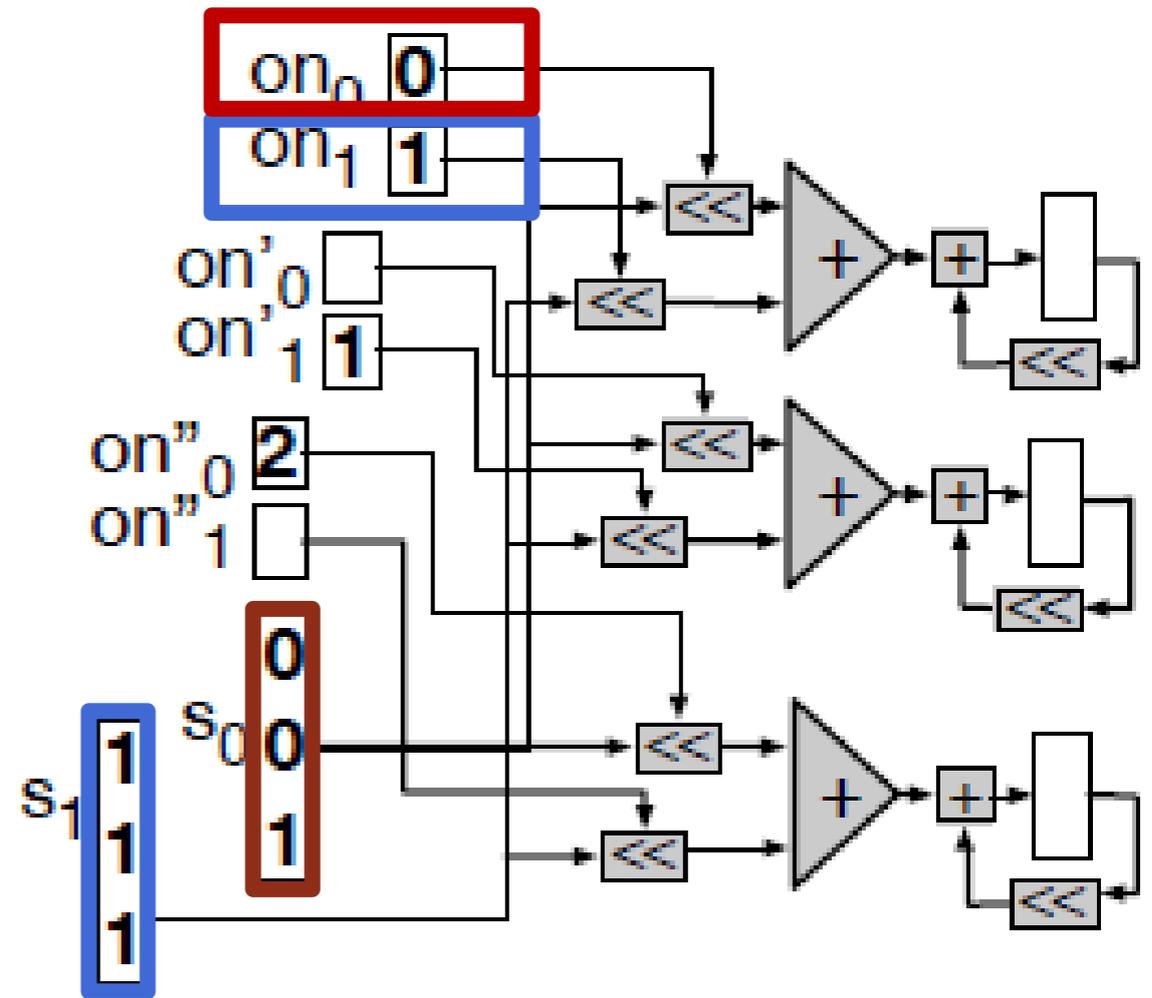


- Only 27% of bits are non-zero

Pragmatic Concept: Use Shift-and-Add



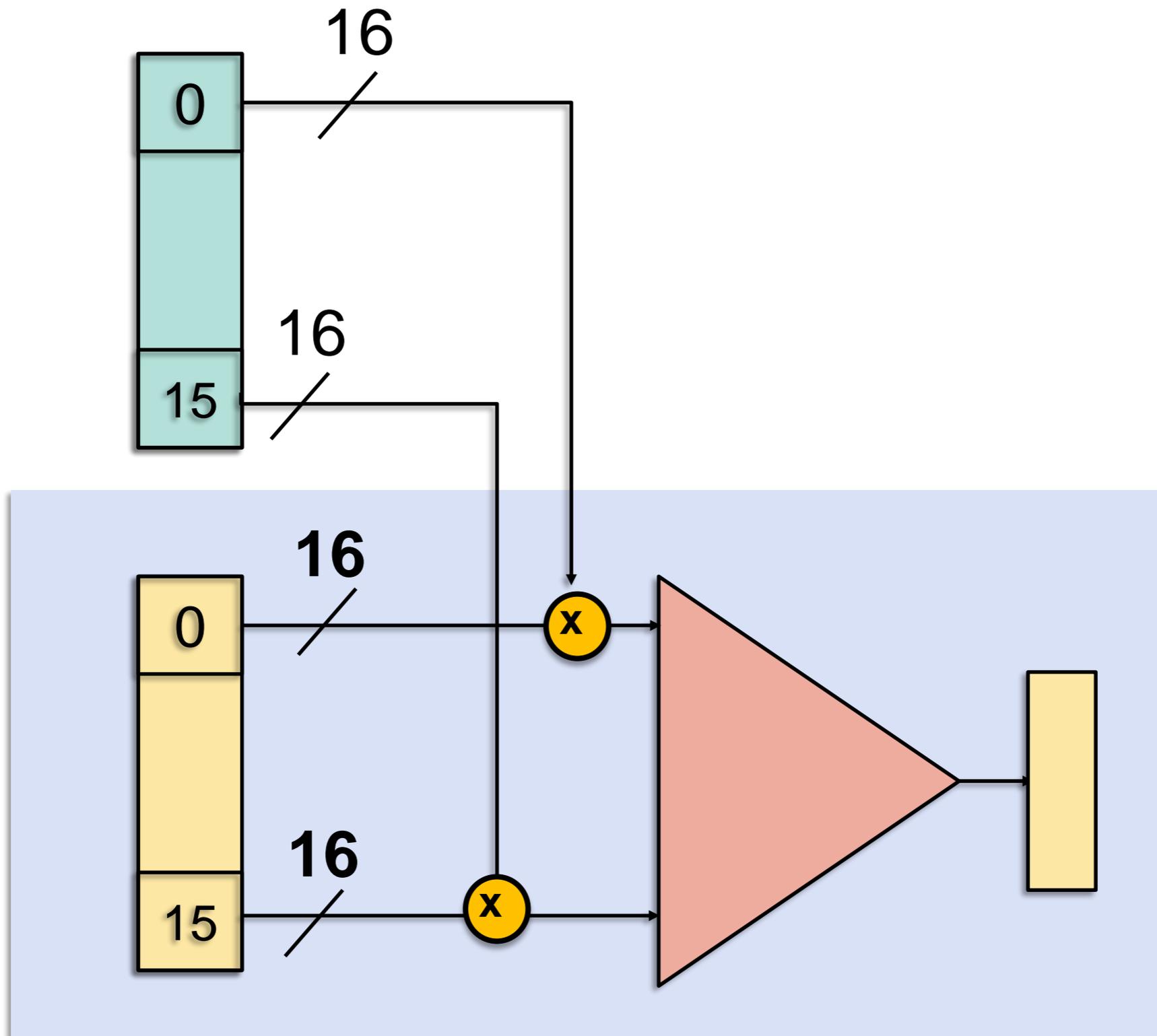
(a) Bit-Parallel Unit



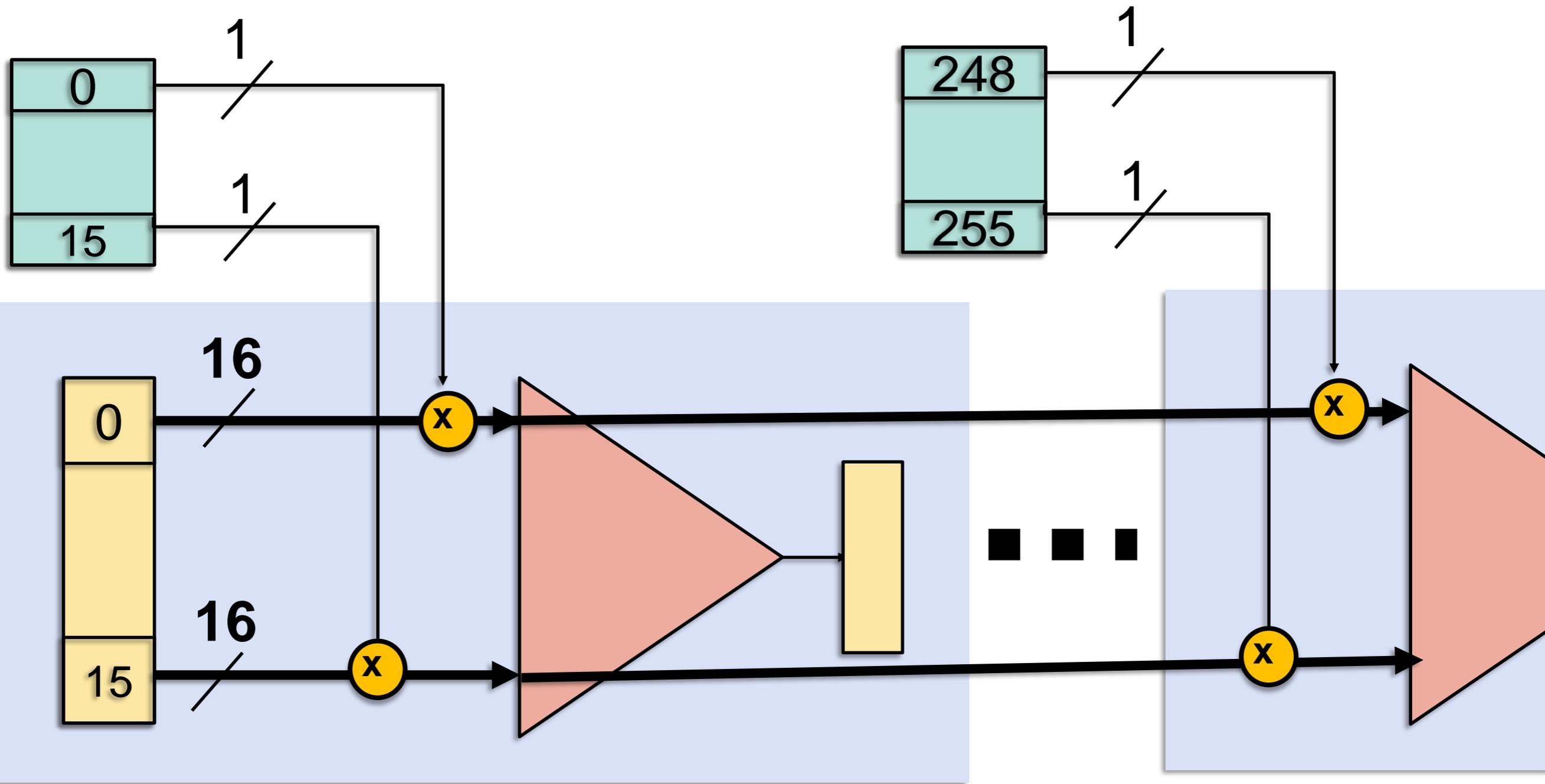
(b) Pragmatic Unit

- **Simply Modify Stripes?**
- Too Large + Cross Lane Synchronization

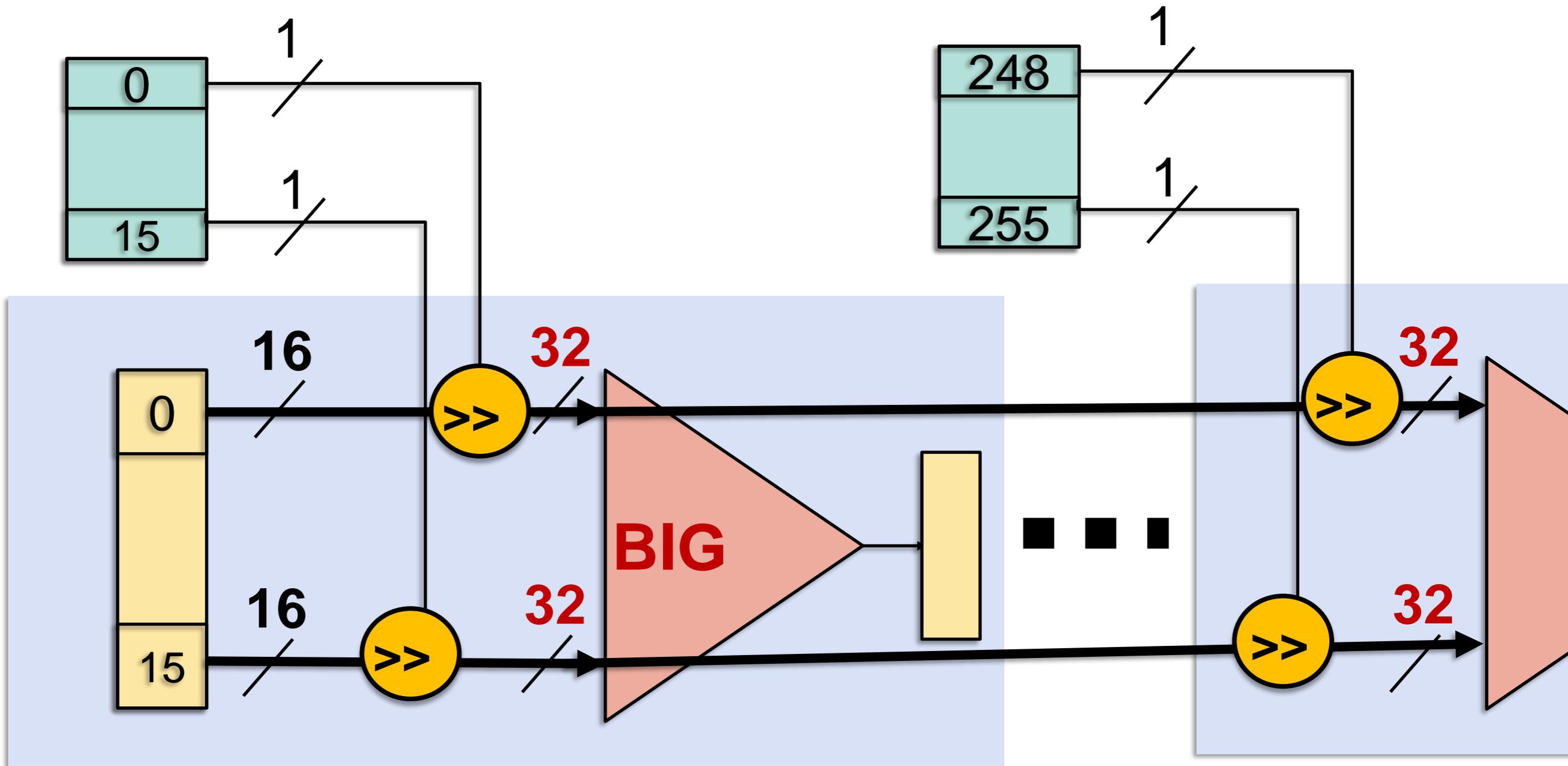
Bit-Parallel Engine



STRIPES

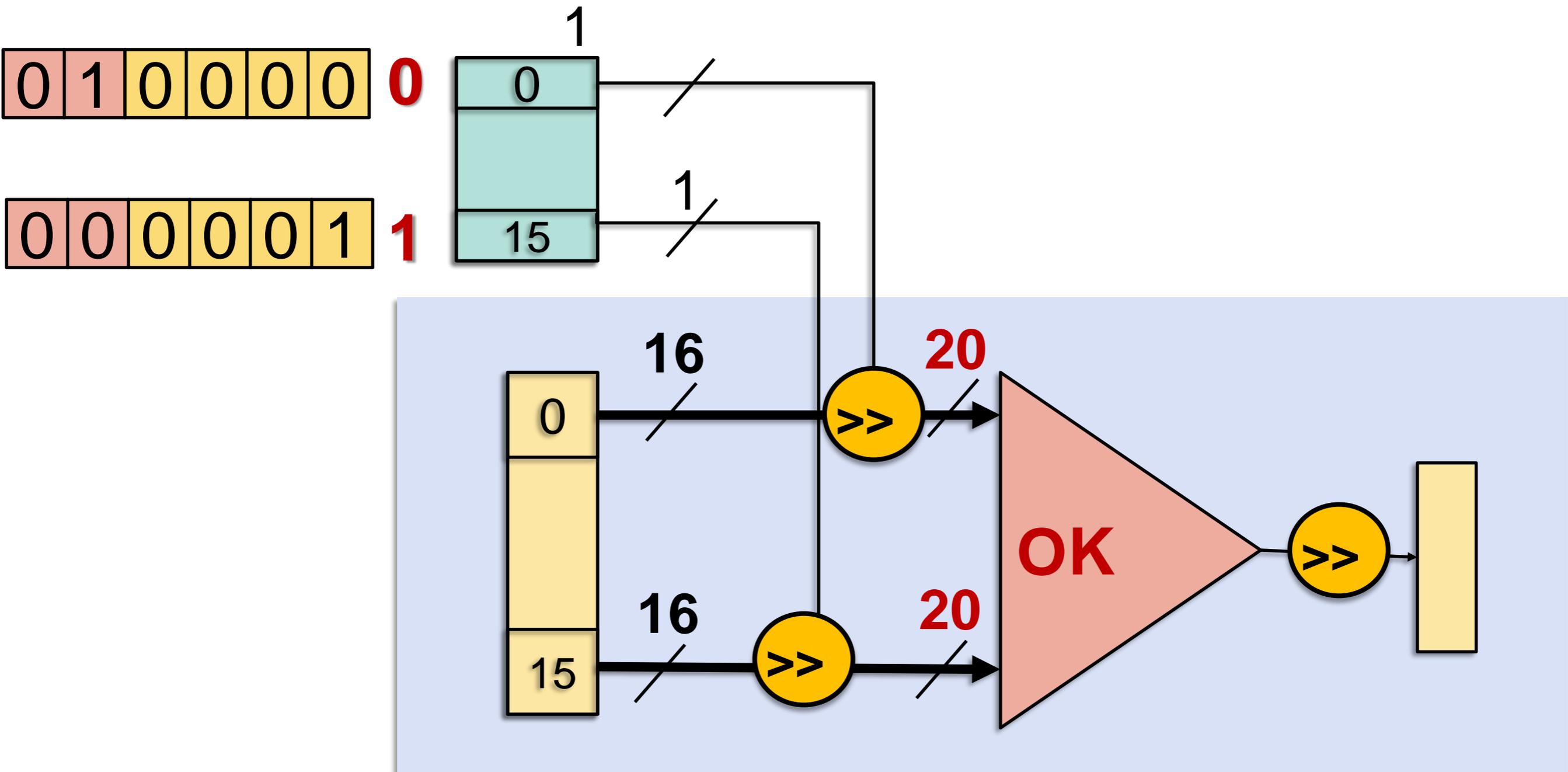


Pragmatic: Naive STRIPES extension? Problem #1: Too Large



Solution to #1? 2-Stage Shifting

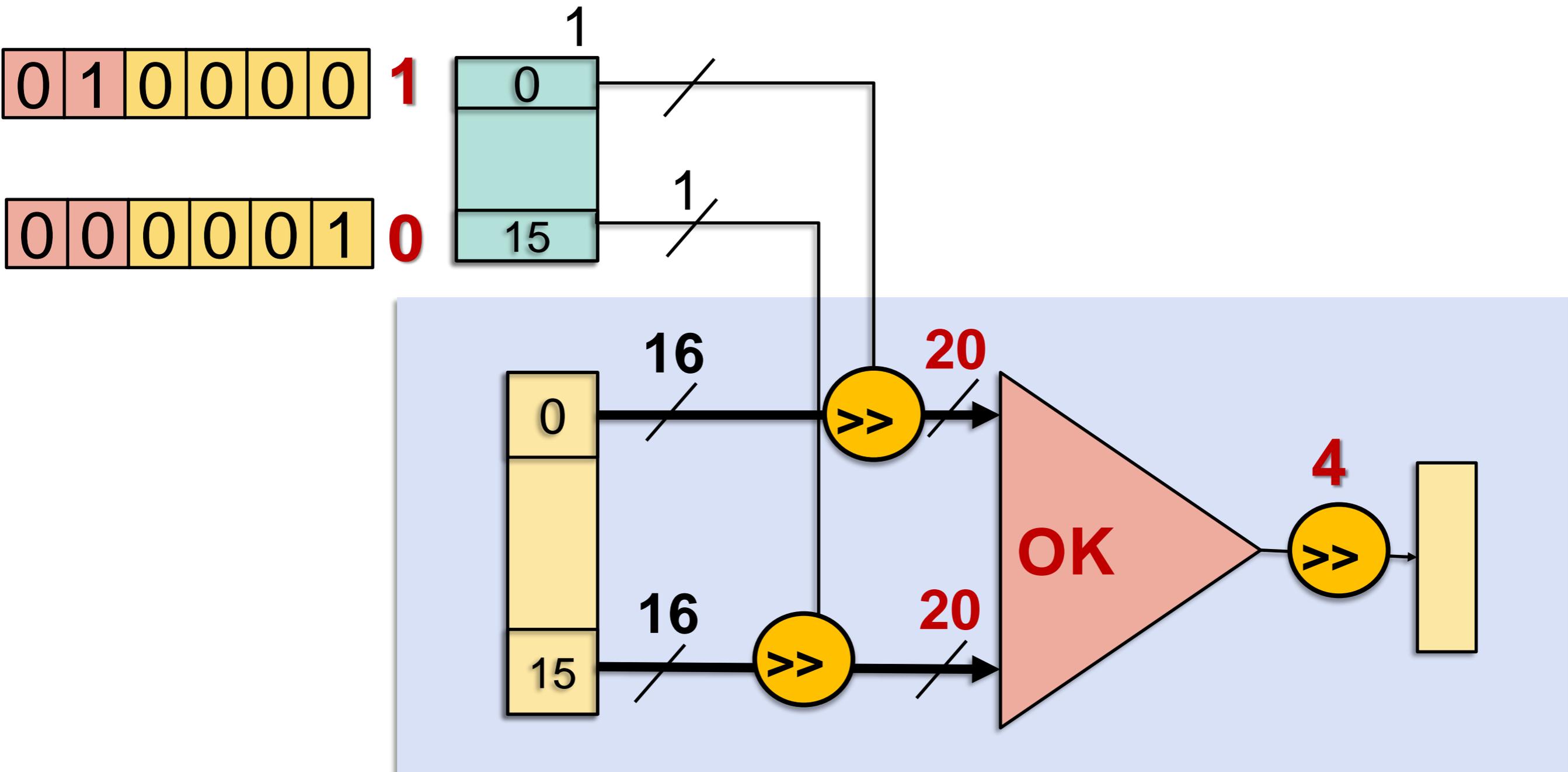
- Process in groups of Max N Difference
- Example with N = 4



- Some opportunity loss, much lower area overhead
- Can skip groups of all zeroes

Solution to #1? 2-Stage Shifting

- Process in groups of Max N Difference
- Example with N = 4



- Some opportunity loss, much lower area overhead

Lane Synchronization

- **Different # of 1 bits**
- **Lanes go out of sync**
- **May have to fetch up to 256 different activations from NM**
- **Keep Lanes Synchronized:**
 - No cost: All lanes
 - Extra register per column: some cost better performance

Bit-Pragmatic

No Accuracy Loss

+310% performance

- 48% Energy

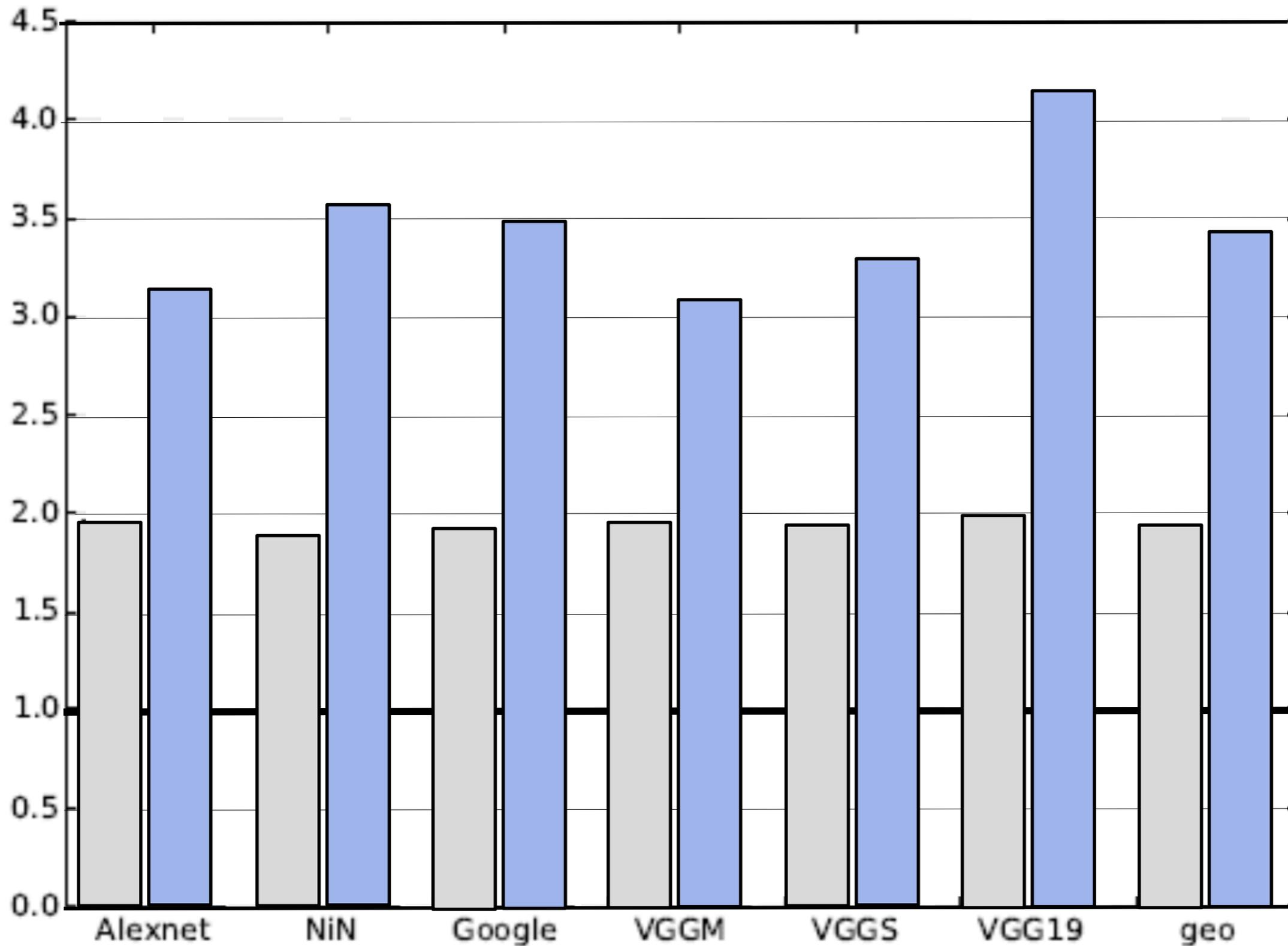
+ 45% Area

**Better w/ 8-bit Quantized
Nets**

Processing Only The Essential Information

Stripes 8b

Pragmatic

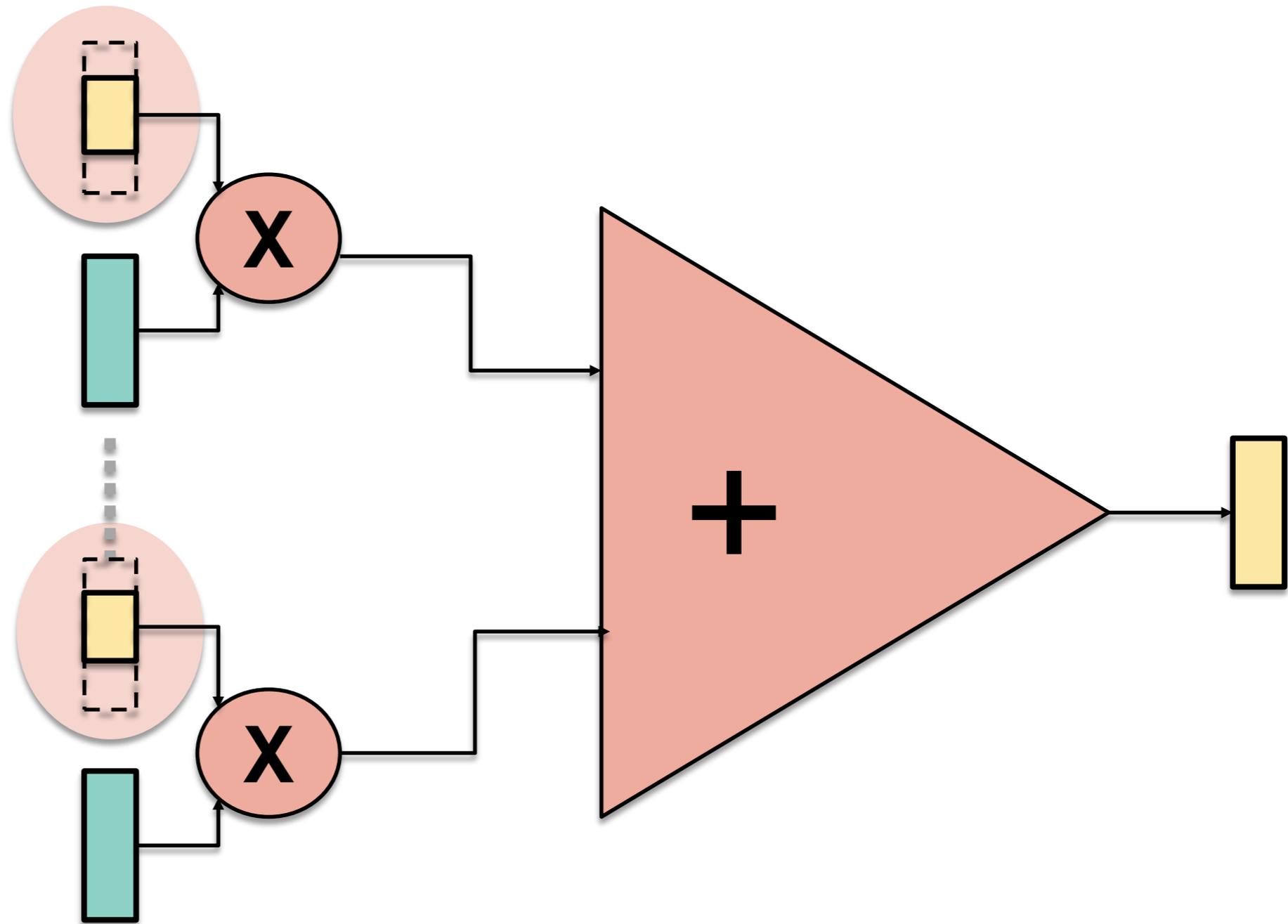


- **8-bit Quantized Representation**

- **Better encoding is possible and improves performance**

Proteus

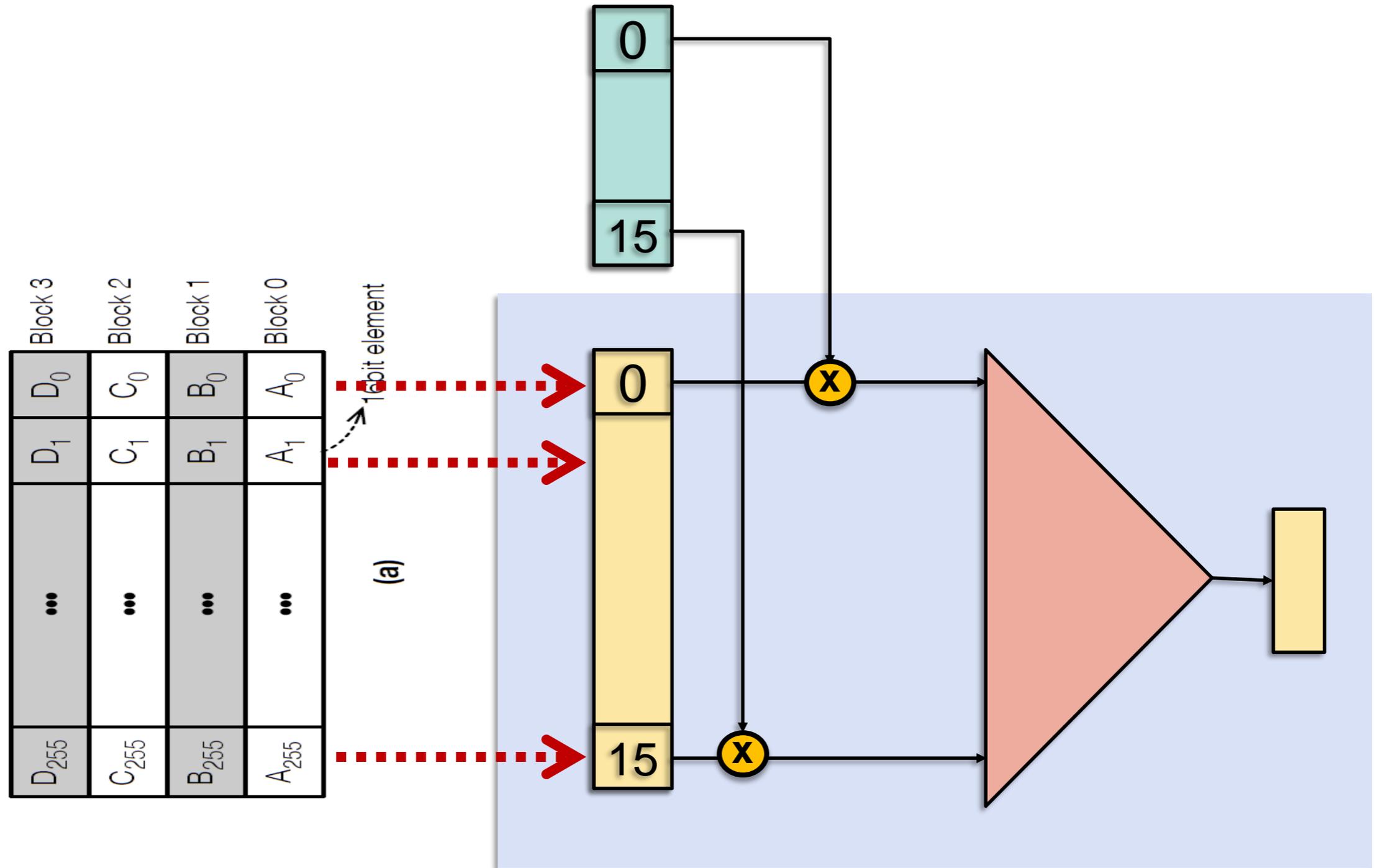
- **Operand Precision Required Varies**



Proteus: Store in reduced precision in memory

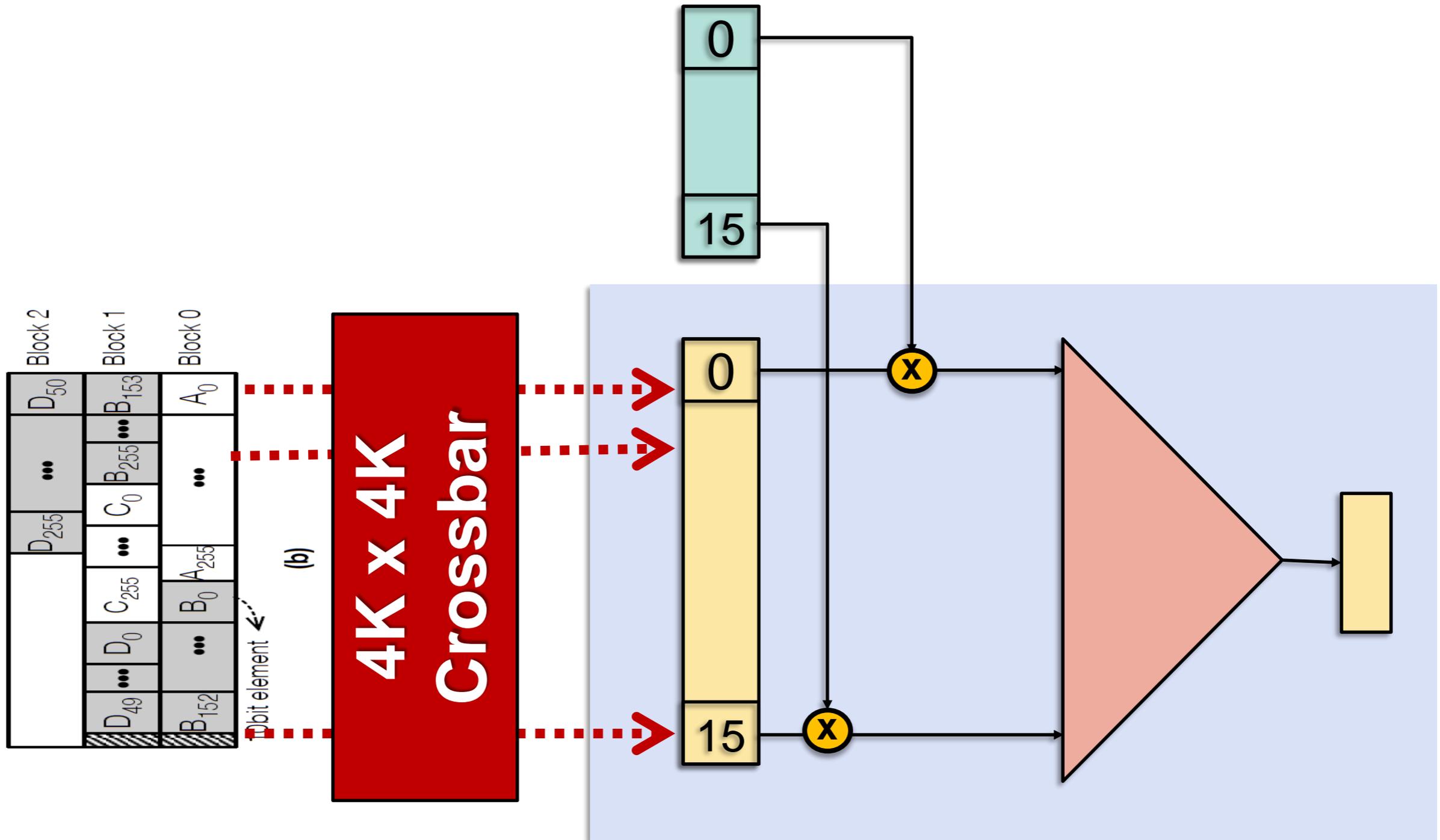
Less Bandwidth, Less Energy

Conventional Format: Base Precision



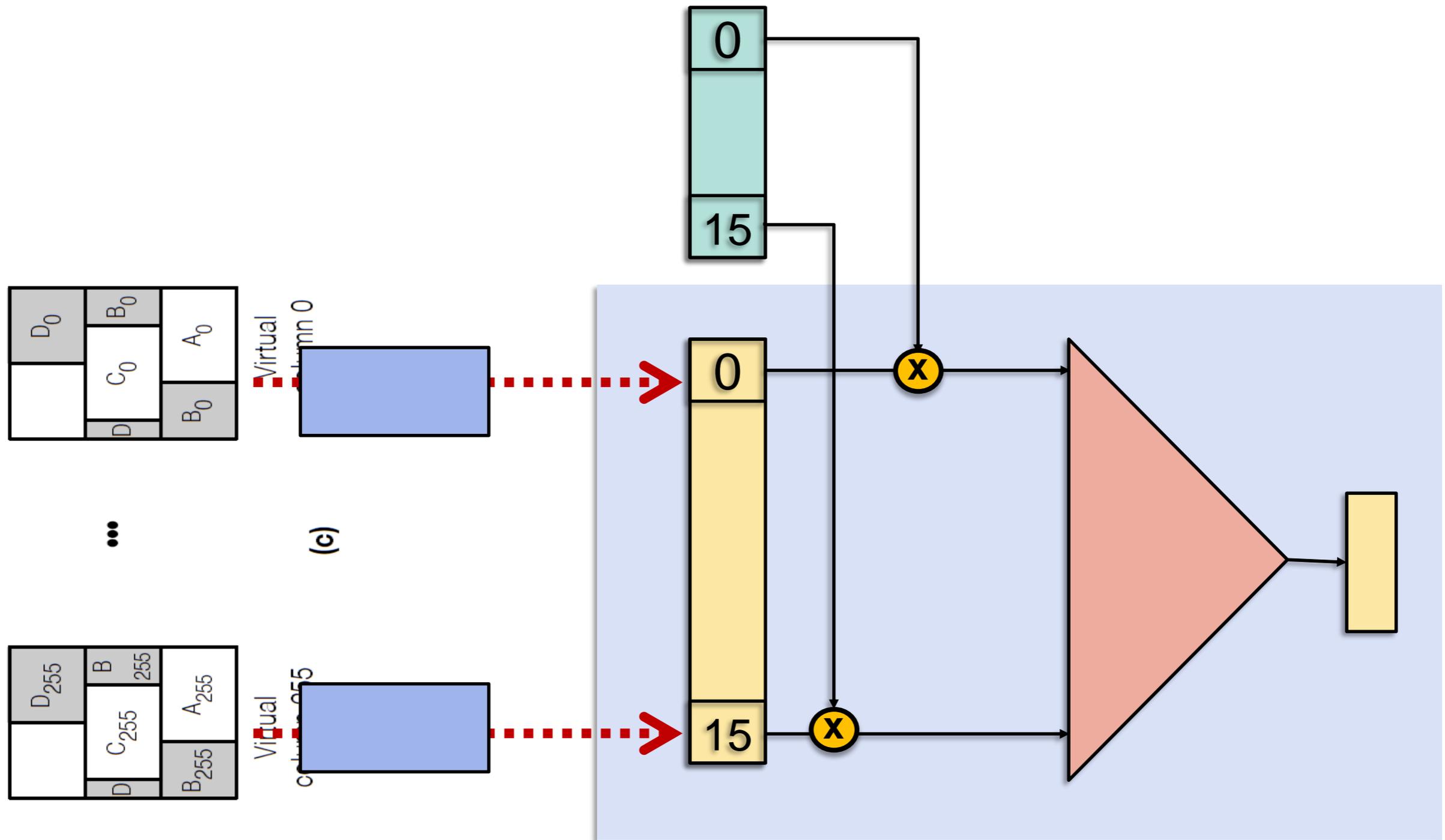
Data Physically aligns with Unit Inputs

Conventional Format: Base Precision



Need Shuffling Network to Route Synapses
4K input bits \rightarrow Any 4K output bit position

Proteus' Key Idea: Pack Along Data Lane Columns

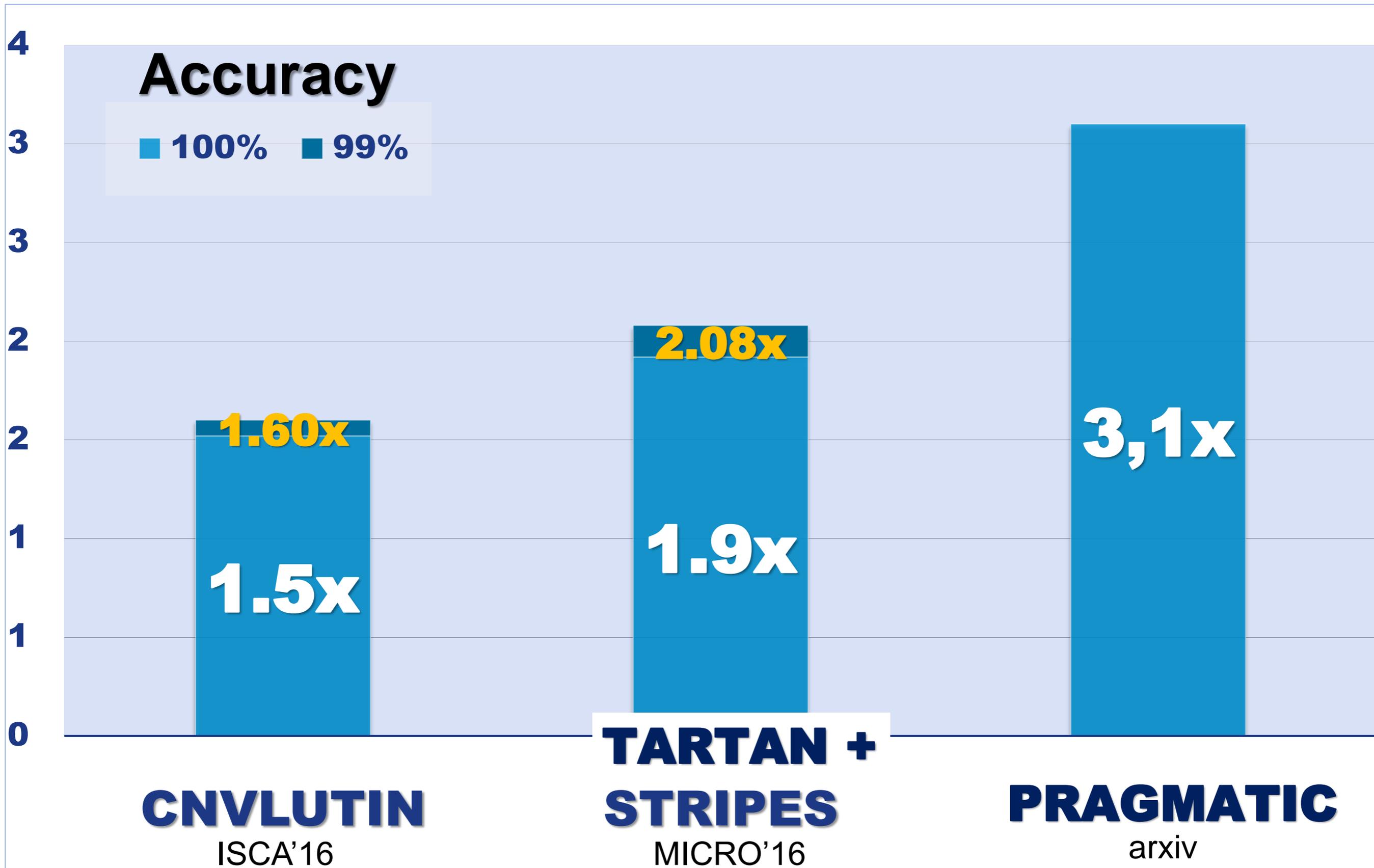


Local Shufflers: 16b input 16b output
Much simpler

**44% less memory
bandwidth**

- **Training**
- **Prototype**
 - *Design Space: lower-end confs*
- **Unified Architecture**
 - *Dispatcher + Compute*
 - *Other Workloads: Comp. Photo*
- **General Purpose Compute Class**

Our Results: Performance



vs. DaDianNao which is ~300x over GPUs

A Value-Based Approach to Acceleration

- **More properties to discover and exploit**
 - E.g., Filters do overlap significantly

- **CNNs one class**
 - Other networks
 - Use the same layers
 - Relative importance different

- **Training**