Bit Fusion

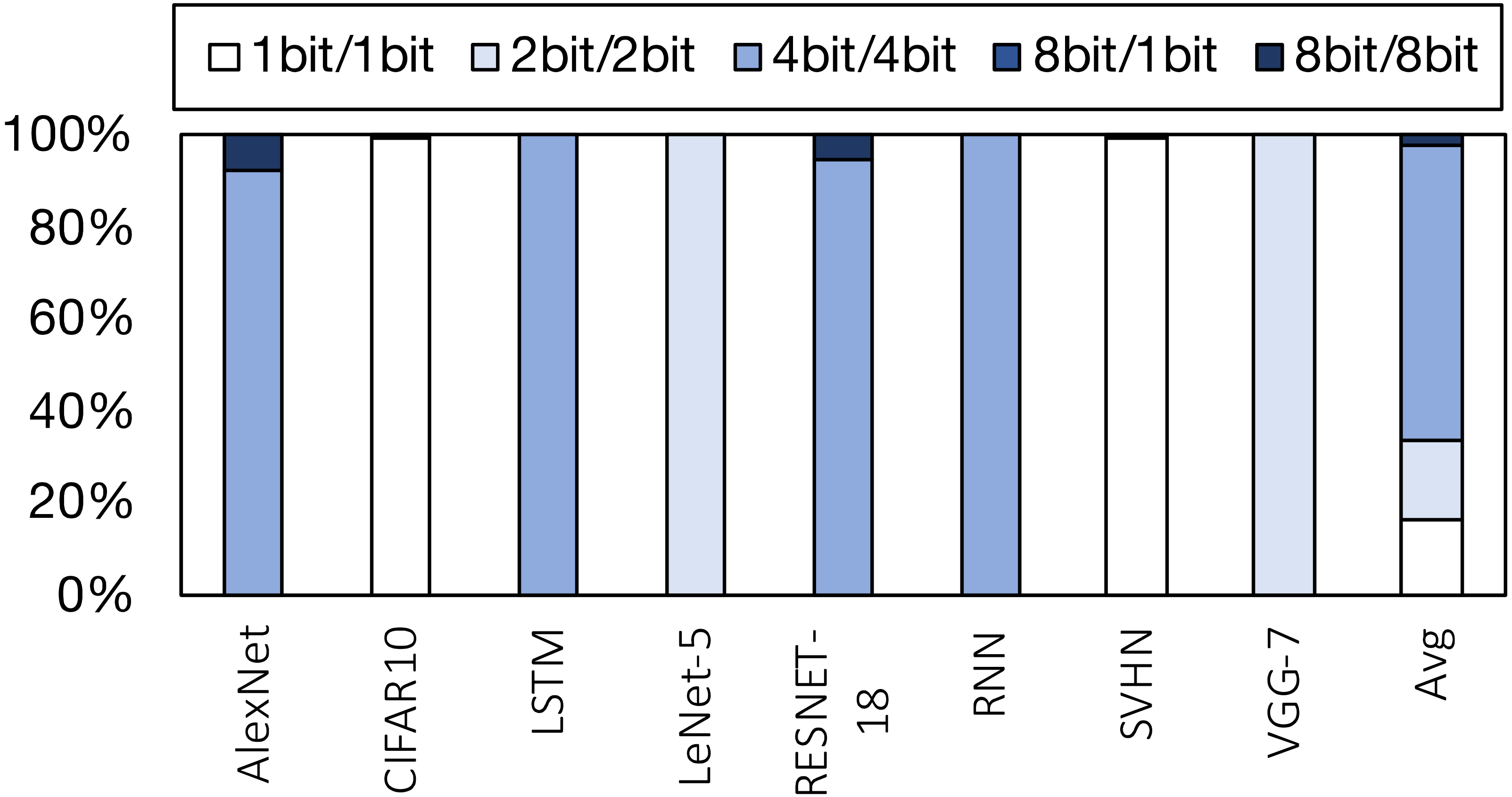
Bit-Level Dynamically Composable Architecture for Deep Neural Networks

Hardik Sharma
Jongse Park
Naveen Suda[†]
Liangzhen Lai[†]
Benson Chau
Vikas Chandra[†]
Hadi Esmaeilzadeh[‡]

Georgia Institute of Technology
[†] Arm, Inc.
[‡] University of California, San Diego

Alternative Computing Technologies (ACT) Lab

DNNs Tolerate Low-Bitwidth Operations

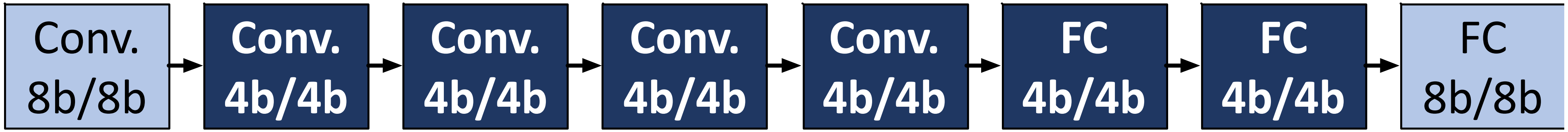


>99.4% Multiply-Adds require **less than 8-bits**

Bitwidth Flexibility is Necessary for Accuracy

AlexNet: IMAGENET dataset

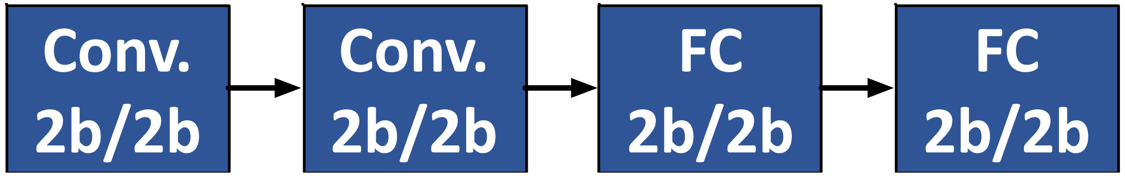
(Mishra et al., WRPN, arXiv 2017)



Conv. 8b/8b → Conv. 4b/4b → Conv. 4b/4b → Conv. 4b/4b → Conv. 4b/4b → FC 4b/4b → FC 4b/4b → FC 8b/8b

LeNet: MNIST dataset

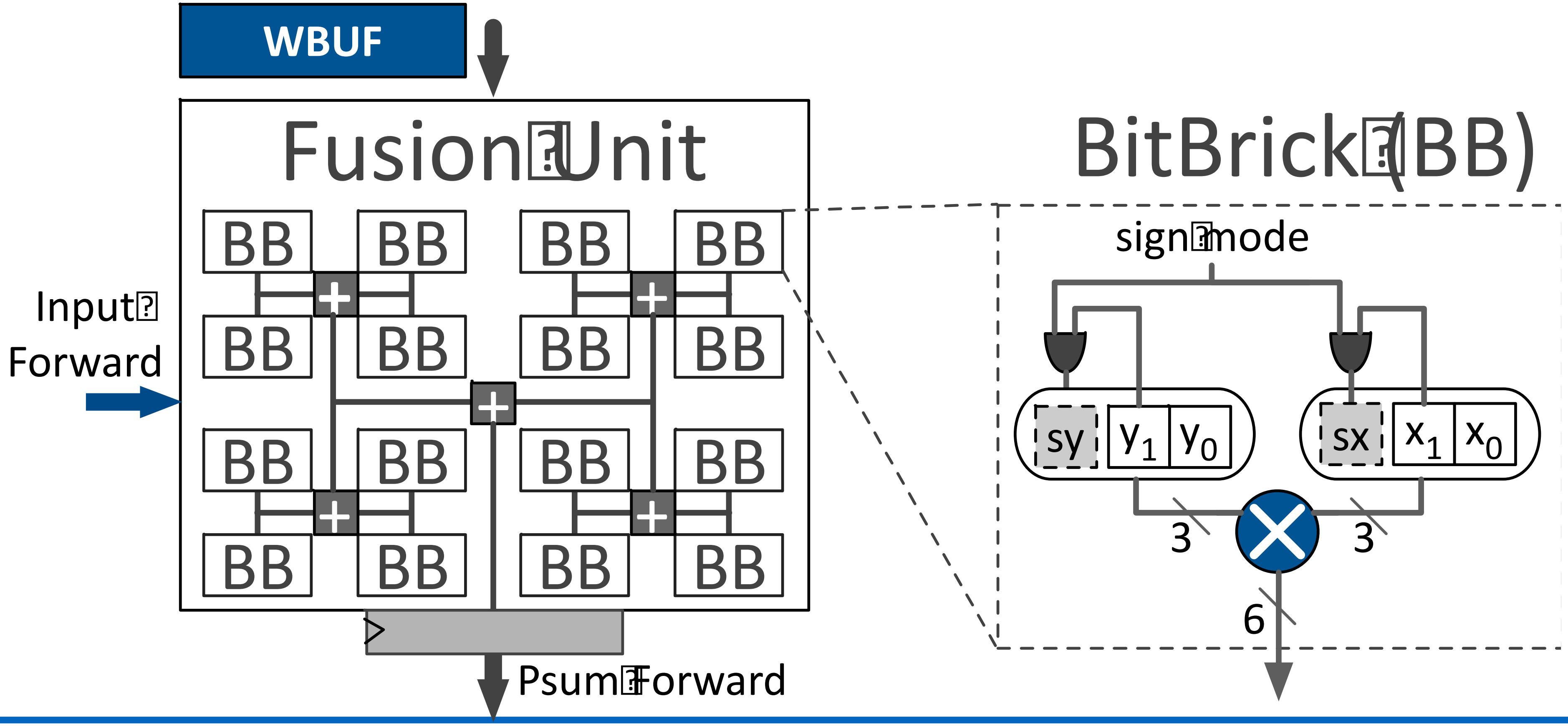
(Li et al., TWN, arXiv 2016)



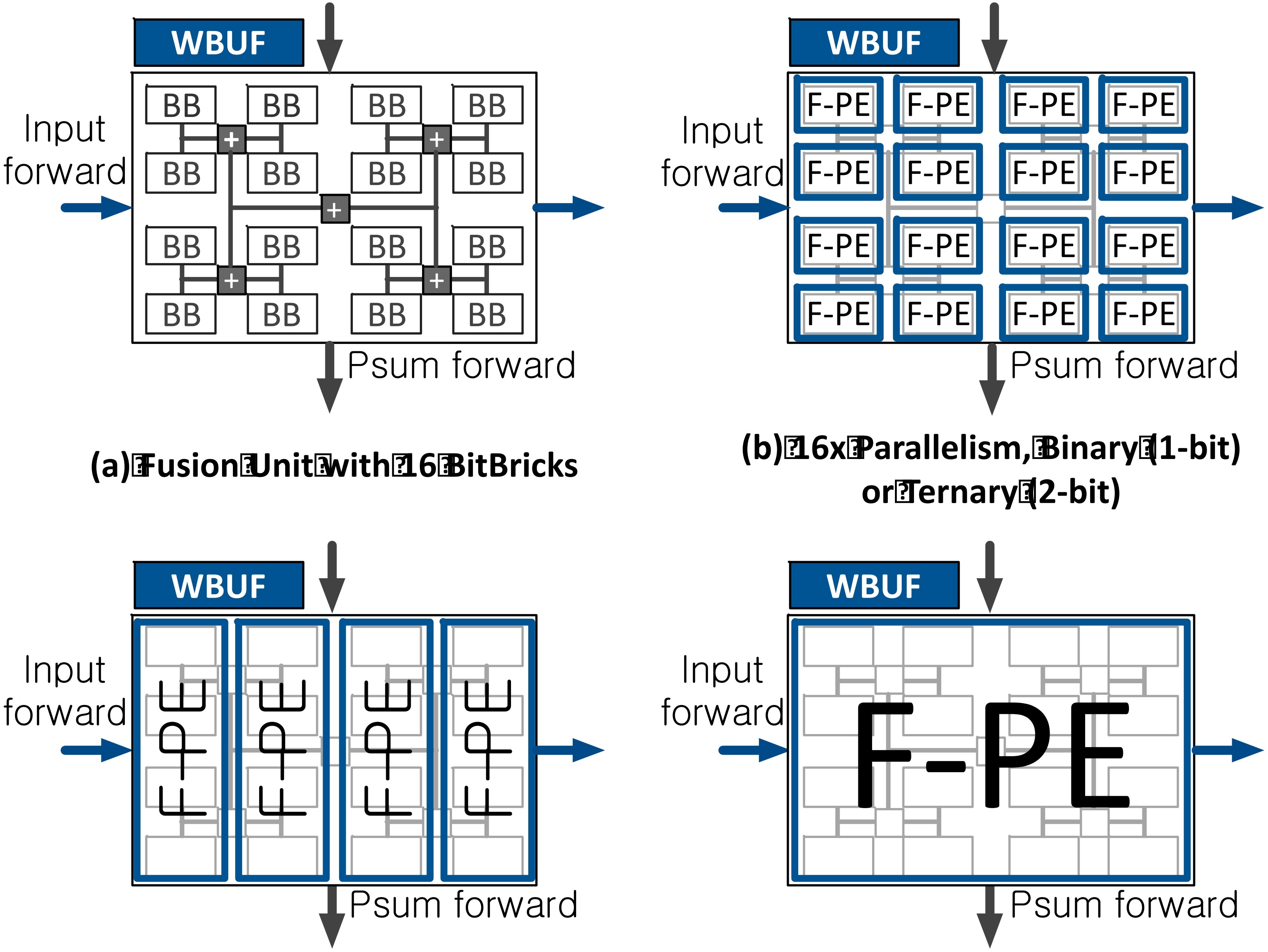
Conv. 2b/2b → Conv. 2b/2b → FC 2b/2b → FC 2b/2b

A **fixed-bitwidth accelerator** would either achieve **limited benefits (8-bit), or compromise on accuracy (<8-bit)**

Our Approach: Bit-level Composability



BitBricks (BBs) are **bit-level composable compute units**



(a) Fusion Unit with 16 BitBricks

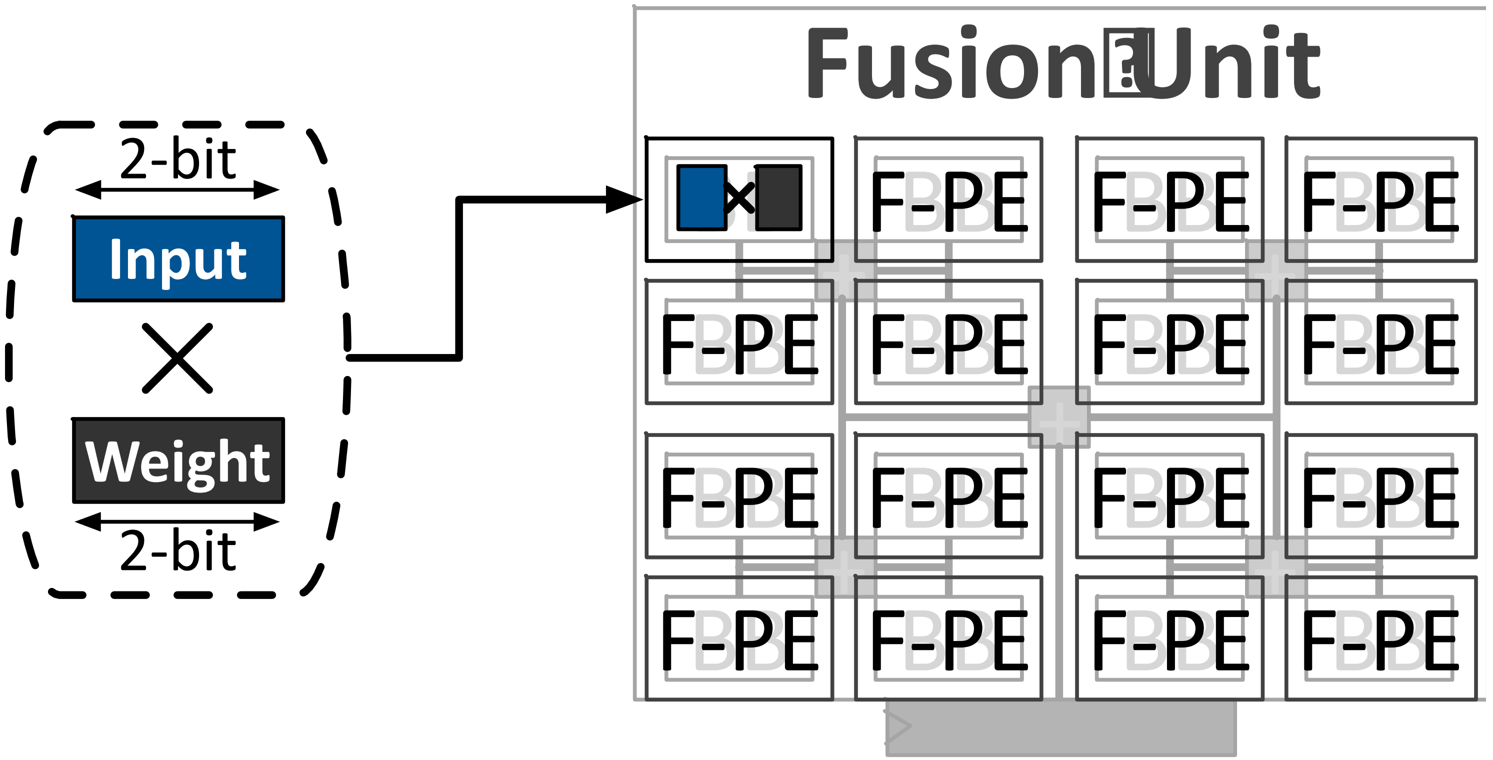
(b) 16x Parallelism, Binary (1-bit) or Ternary (2-bit)

(c) 4x Parallelism, Mixed-Bitwidth (2-bit weights, 8-bit inputs)

(d) No Parallelism, 8-bits

Compute units (BitBricks) **logically fuse at runtime** to form Fused-PEs (F-PEs) that **dynamically match bit-width** of the DNN layers

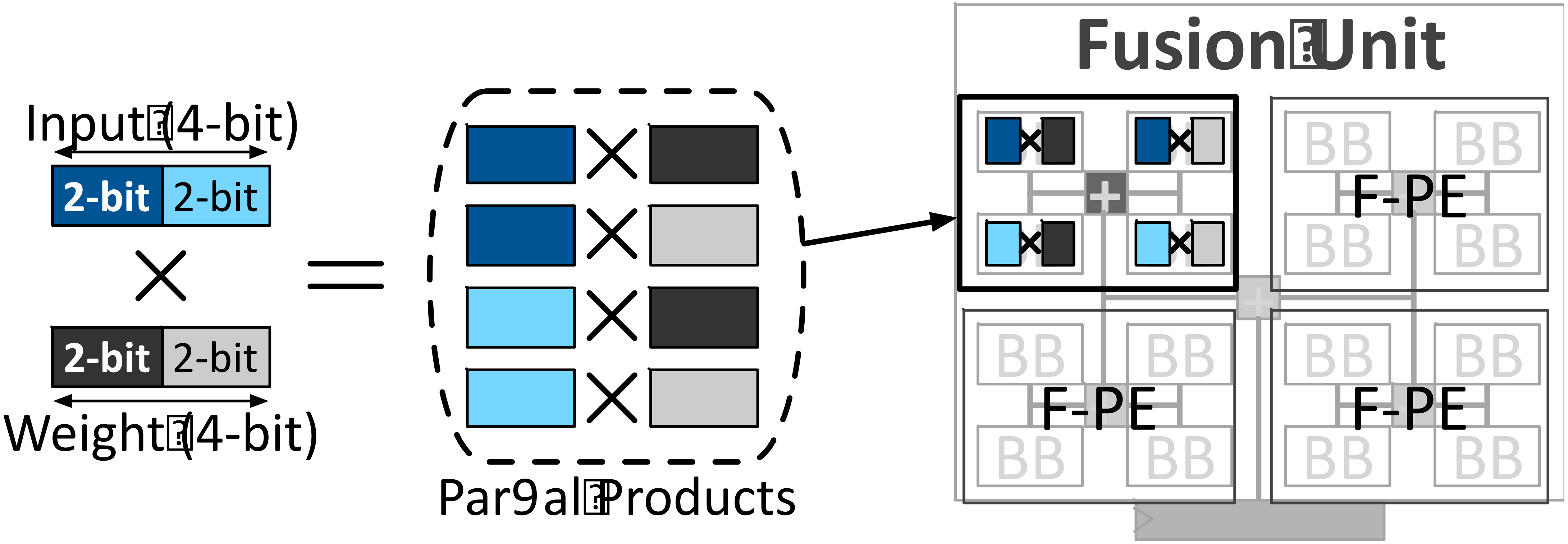
Config #1 : Binary/Ternary Mode



Each BitBrick performs a binary/ternary multiplication

16x parallelism

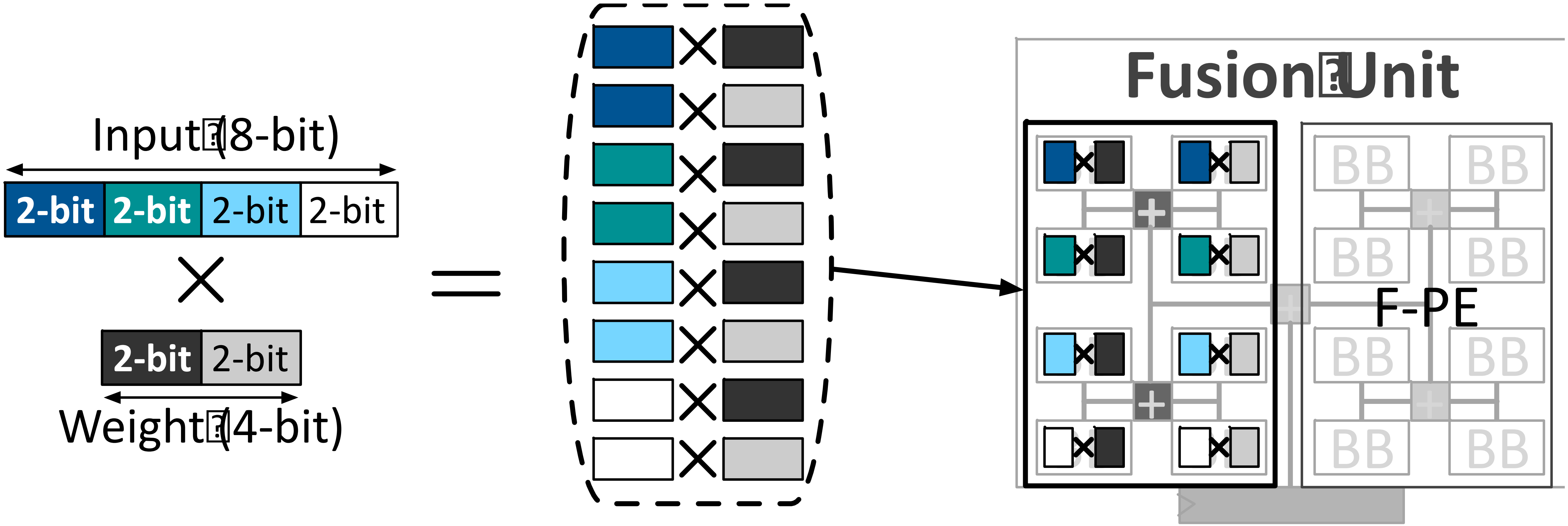
Config #2: 4-bit Mode



Four BitBricks fuse to form a Fused-PE (F-PE)

4x Parallelism

Config #3 : 8-bit, 4-bit (Mixed-Mode)

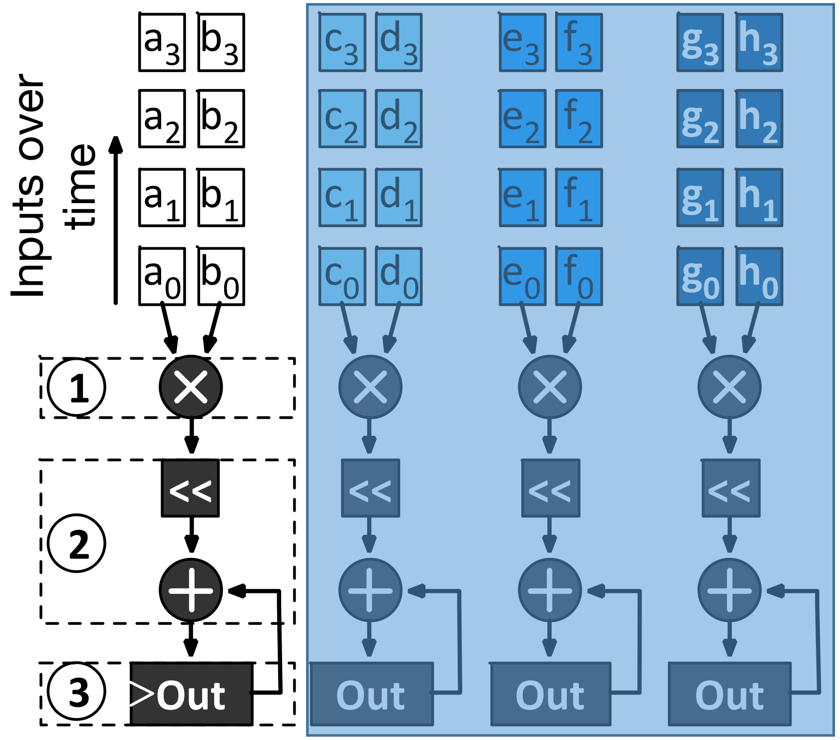


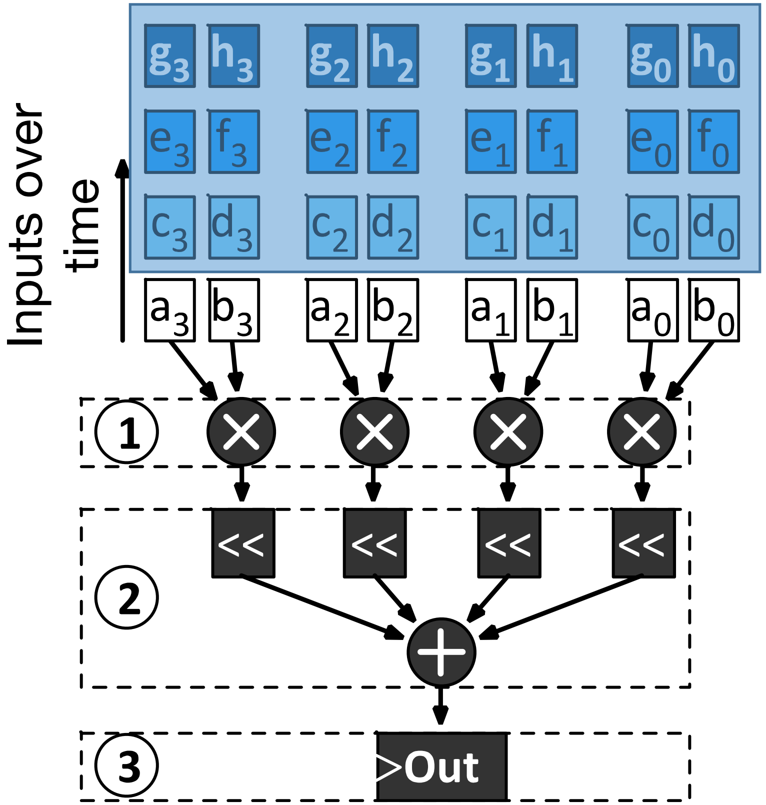
Par: all Products

Eight BitBricks fuse to form a Fused-PE (F-PE)

2x Parallelism

Spatial Fusion vs. Temporal Design


Inputs over time
a_3 b_3
a_2 b_2
a_1 b_1
a_0 b_0
c_3 d_3 e_3 f_3 g_3 h_3
c_2 d_2 e_2 f_2 g_2 h_2
c_1 d_1 e_1 f_1 g_1 h_1
c_0 d_0 e_0 f_0 g_0 h_0
1
2
3
<<
Out
Out
Out
Out


Inputs over time
g_3 h_3 g_2 h_2 g_1 h_1 g_0 h_0
e_3 f_3 e_2 f_2 e_1 f_1 e_0 f_0
c_3 d_3 c_2 d_2 c_1 d_1 c_0 d_0
a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0
1
2
3
<<
Out

Temporal Design (Bit Serial):
Combine results over time

Spatial Fusion (Bit Parallel):
Combine results over space

Spatial Fusion Surpasses Temporal Design

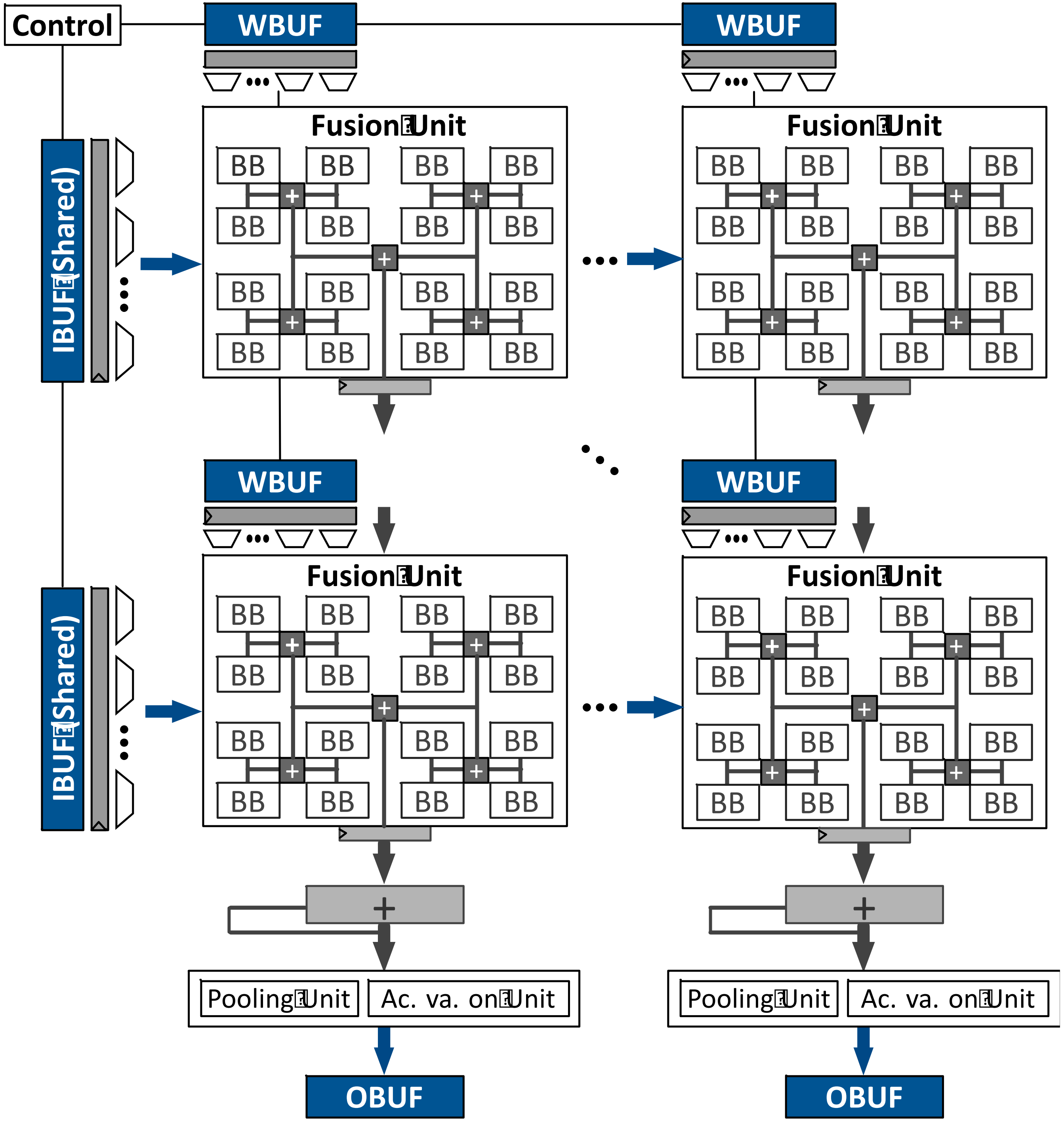
Area (um^2)	BitBricks	Shift-Add	Register	Total Area
Temporal	463	2989	1454	4905
Fusion Unit	369	934	91	1394

3.5x lower area

Power (nW)	BitBricks	Shift-Add	Register	Total Power
Temporal	60	550	1103	1712
Fusion Unit	46	424	69	538

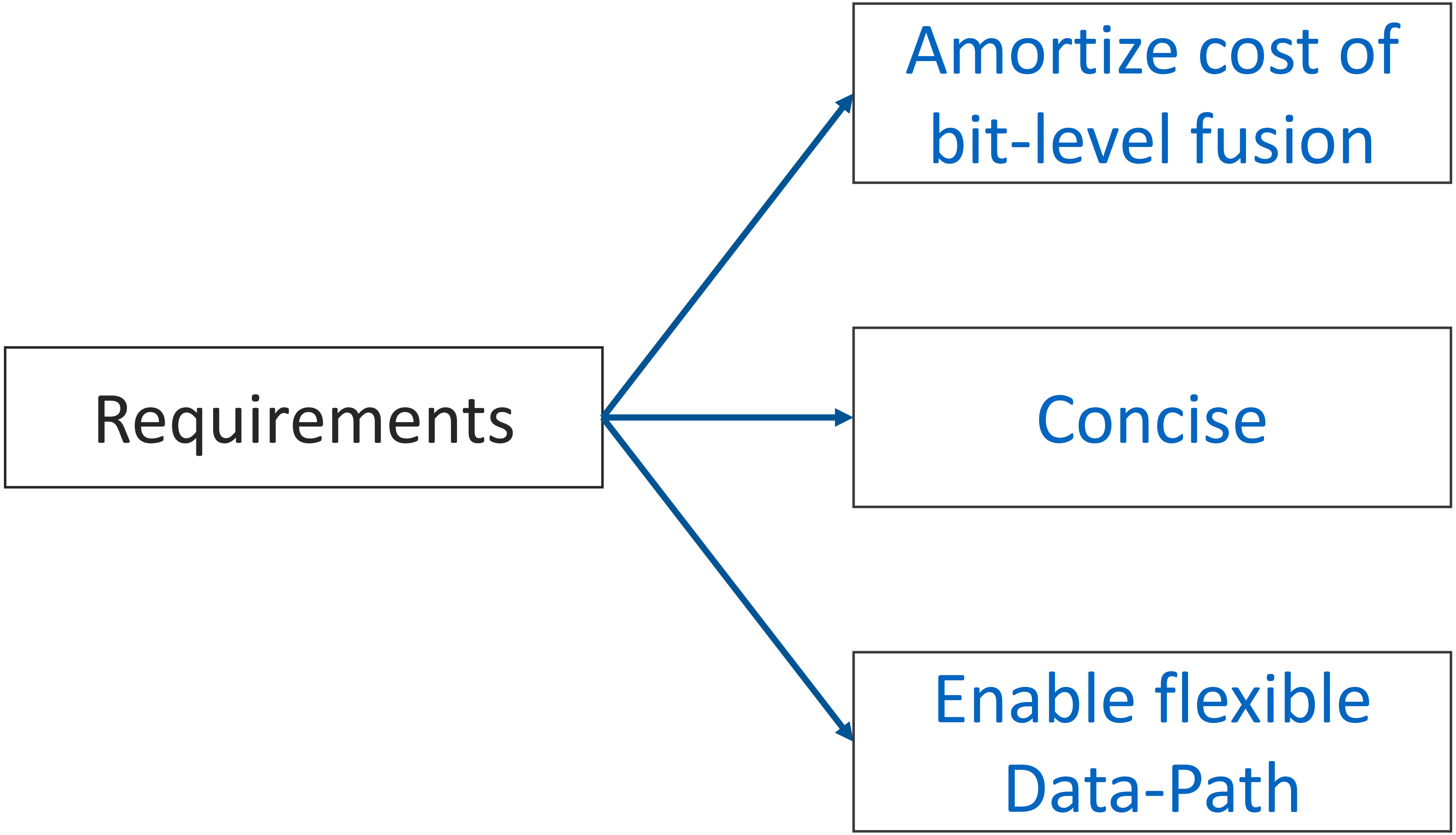
3.2x lower power

Synthesized using a commercial 45 nm technology



Bit Fusion Systolic Array Architecture

Programmability: BitFusion ISA



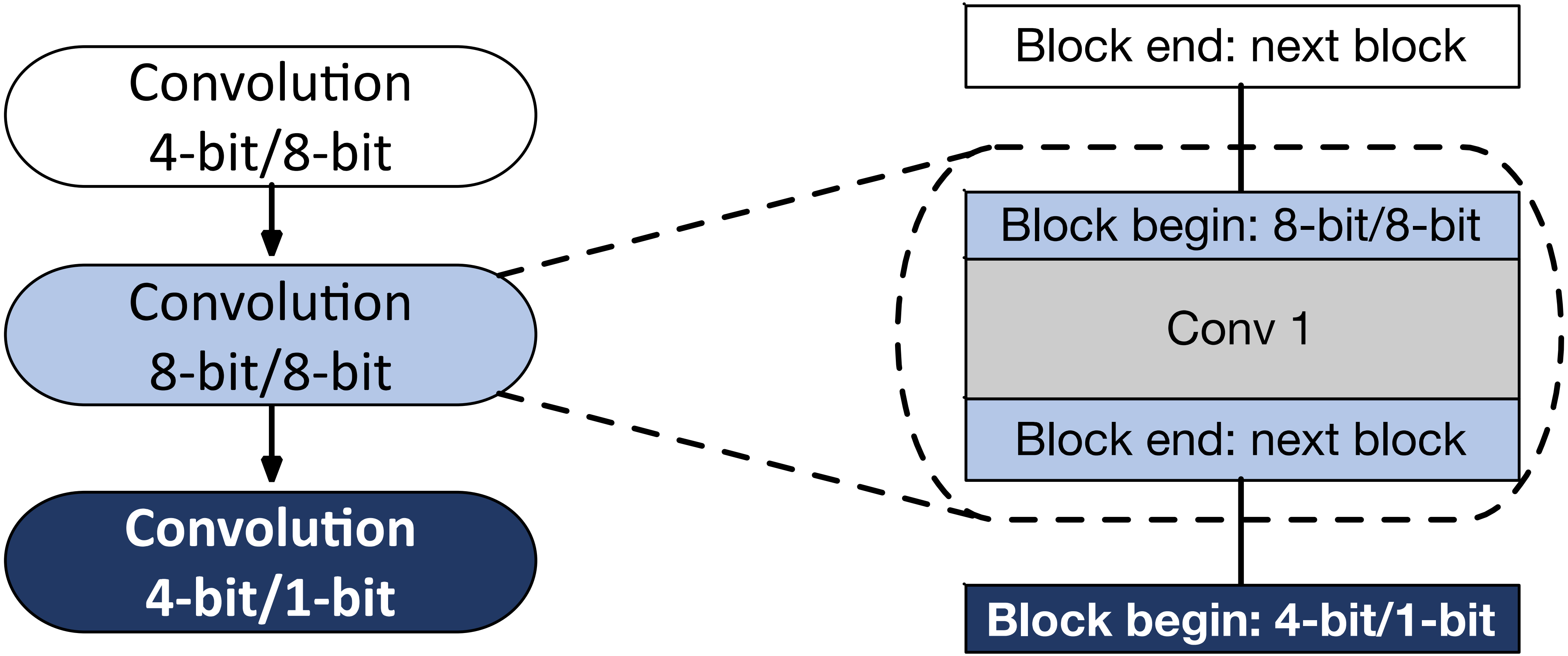
Requirements

Amortize cost of bit-level fusion

Concise

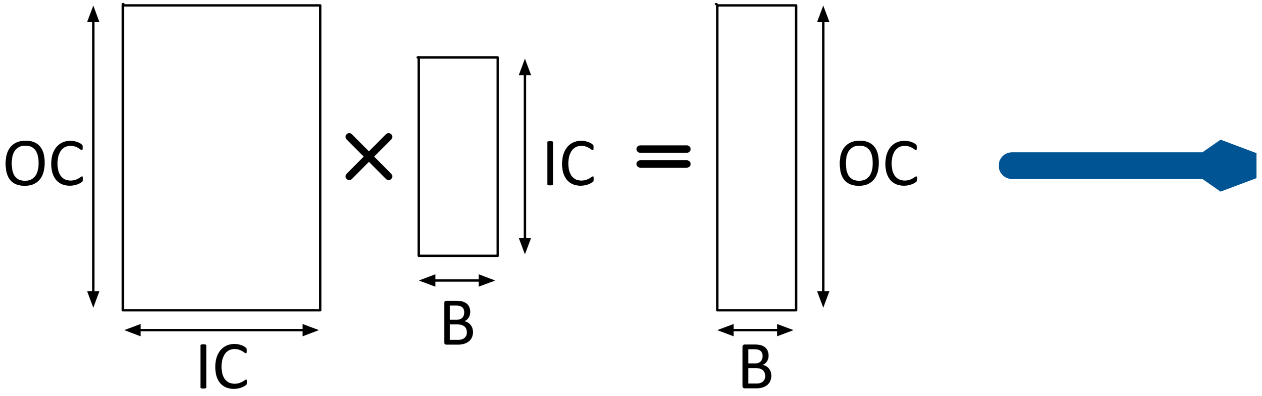
Enable flexible Data-Path

ISA: Amortize the Cost of Bit-Level Fusion



Use a block-structured ISA for groups of operations (layers)

ISA: Concise Expression for DNNs

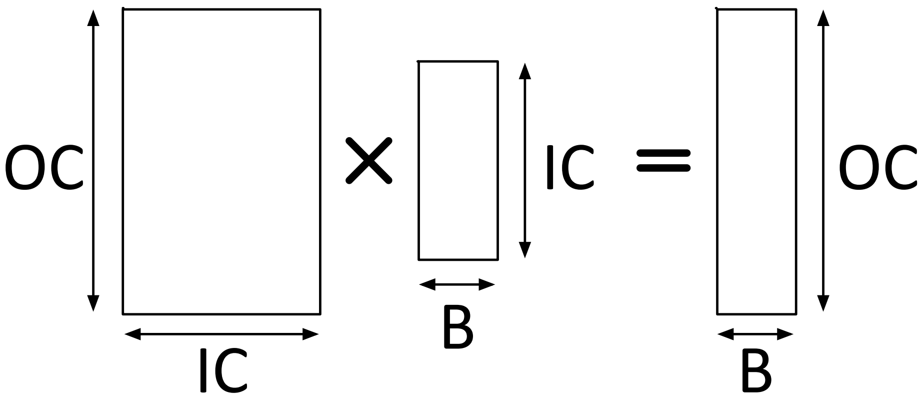


Fully-Connected Layer

```
loop: for i in (1 → B)
  loop: for j in (1 → OC)
    loop: for k in (1 → IC)
```

Use loop instructions as DNNs consist of large number of repeated operations

ISA: Concise Expression for DNNs



Fully-Connected Layer

```
loop: for i in (1 → B)
  loop: for j in (1 → OC)
    loop: for k in (1 → IC)
      input ← k × 1 + j × 0  + i × IC
      weight← k × 1 + j × IC + i × 0
      output← k × 0 + j × 1  + i × OC
```

DNNs have regular memory access pattern

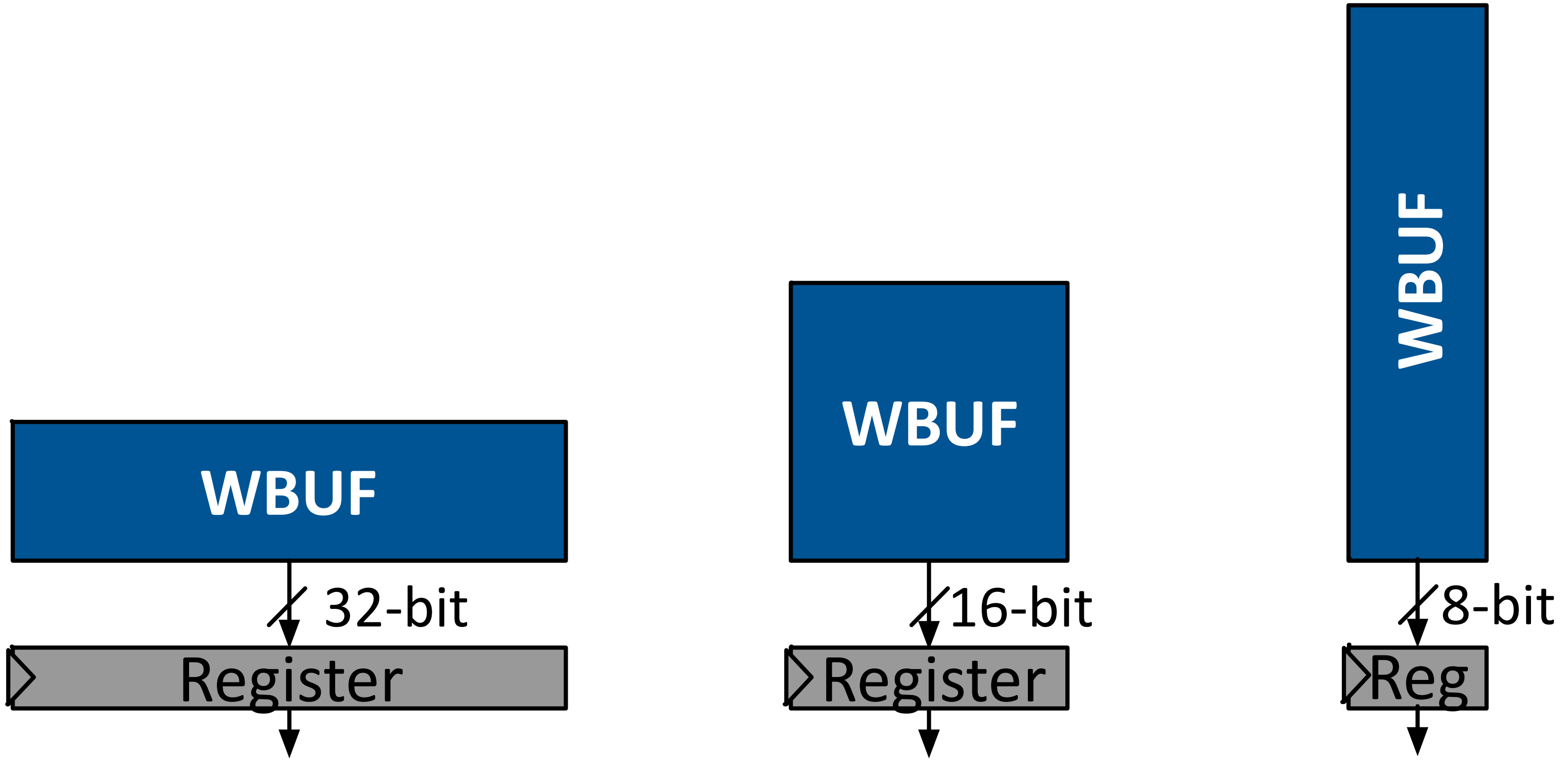
Use loop indices to generate memory accesses

ISA: Flexible Storage

2-bit mode	8-bit mode
16x parallelism	1x parallelism
Need: 32-bit inputs, 32-bit weights	**Need: 8-bit input, 8-bit weight**

ISA changes the semantics of off-chip and on-chip memory accesses according to bitwidth of operands

ISA: Flexible Storage *(Software View)*



Software views the buffers as having a flexible aspect ratio

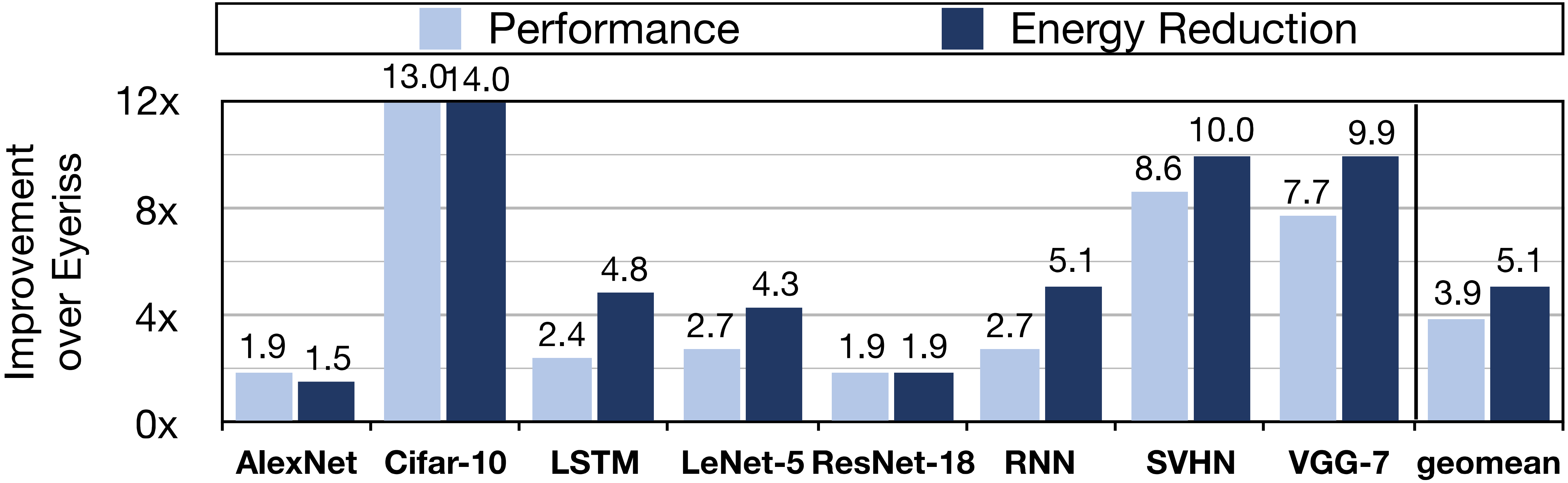
Benchmarked Platforms

GPU	Low Power	Nvidia Tegra TX2
	High Performance	Nvidia Titan-X
ASIC	Bit-Serial	Stripes (Micro’16)
	Optimized Dataflow	Eyeriss (ISCA’16)

Benchmarked DNN Models

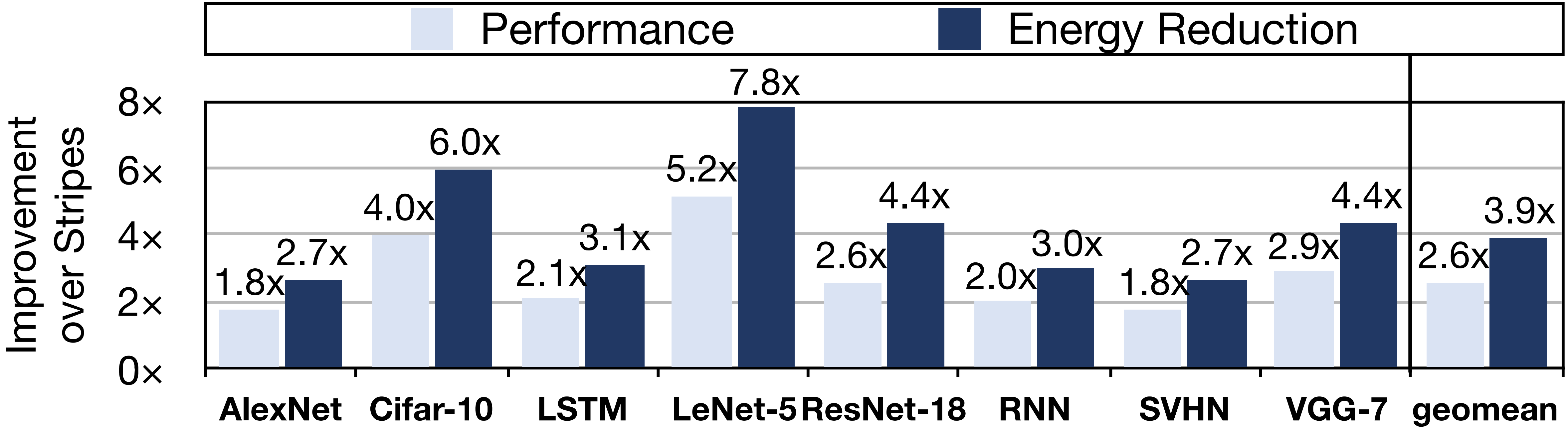
DNN	Type	Multiply-Adds	Bit-Flexible Model Weights	Original Model Weights
AlexNet	CNN	2,678 MOps	116.3 MBytes	898.6 MBytes
CIFAR10	CNN	617 MOps	3.3 MBytes	53.5 MBytes
LSTM	***RNN***	13 MOps	6.2 MBytes	49.4 MBytes
LeNet-5	CNN	16 MOps	0.5 MBytes	8.2 MBytes
RESNET-18	CNN	4,269 MOps	13 MBytes	103.7 MBytes
RNN	***RNN***	17 MOps	8.0 MBytes	64.0 MBytes
SVHN	CNN	158 MOps	0.8 MBytes	24.4 MBytes
VGG-7	CNN	317 MOps	2.7 MBytes	43.3 MBytes

Comparison with Eyeriss



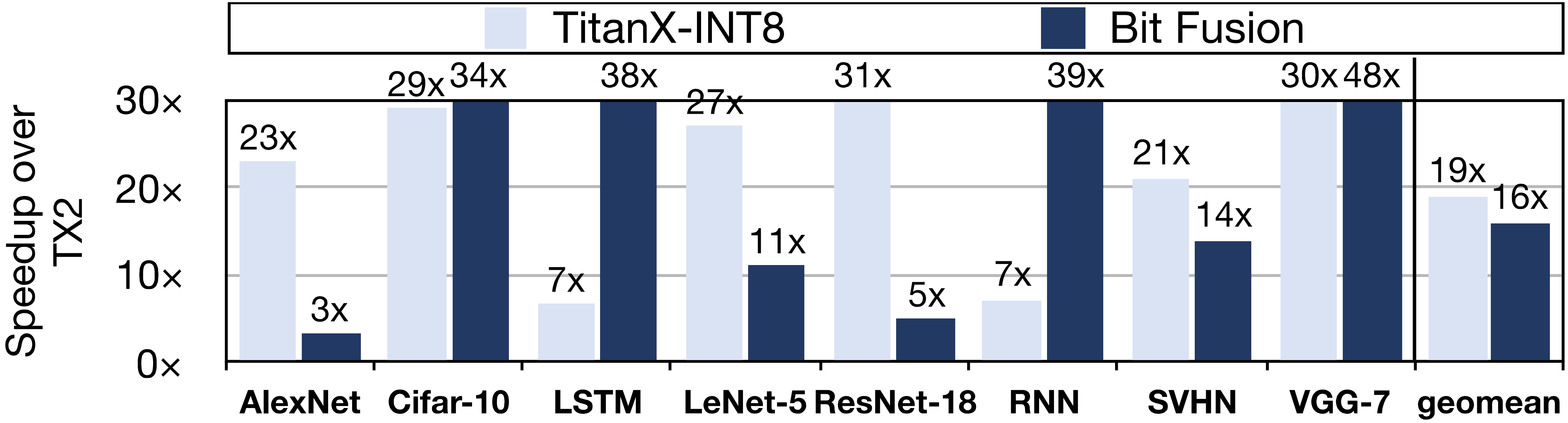
3.9× speedup and **5.1× energy reduction** over Eyeriss

Comparison with Stripes



2.6× speedup and **3.9× energy reduction** over Stripes

Comparison with GPUs



Bit Fusion provides almost the same performance as **Titan Xp (250 W)** with only **895 mW**

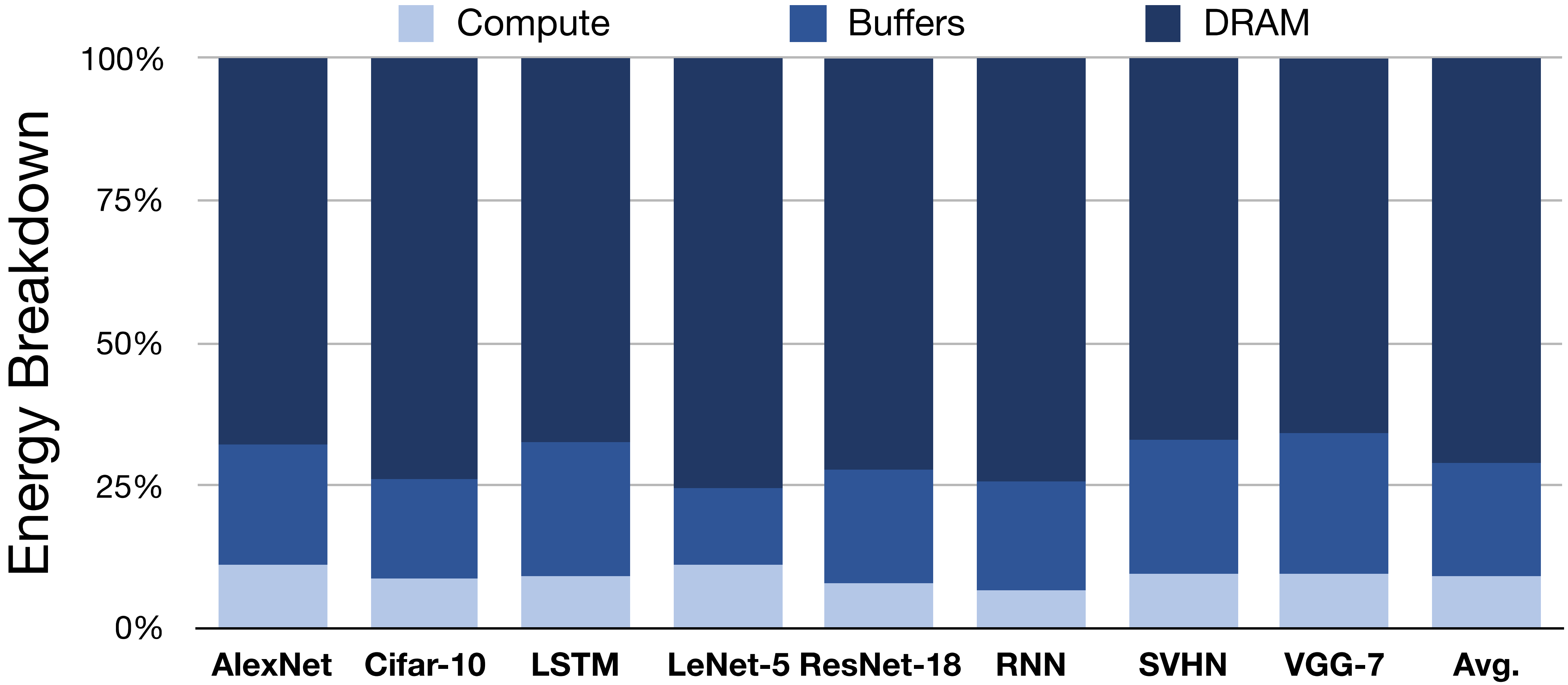
Conclusion

Emerging research shows we can **reduce bitwidths** for DNNs **without losing accuracy**

Bit Fusion defines a new dimension of **bit-level dynamic composability** to leverage this opportunity

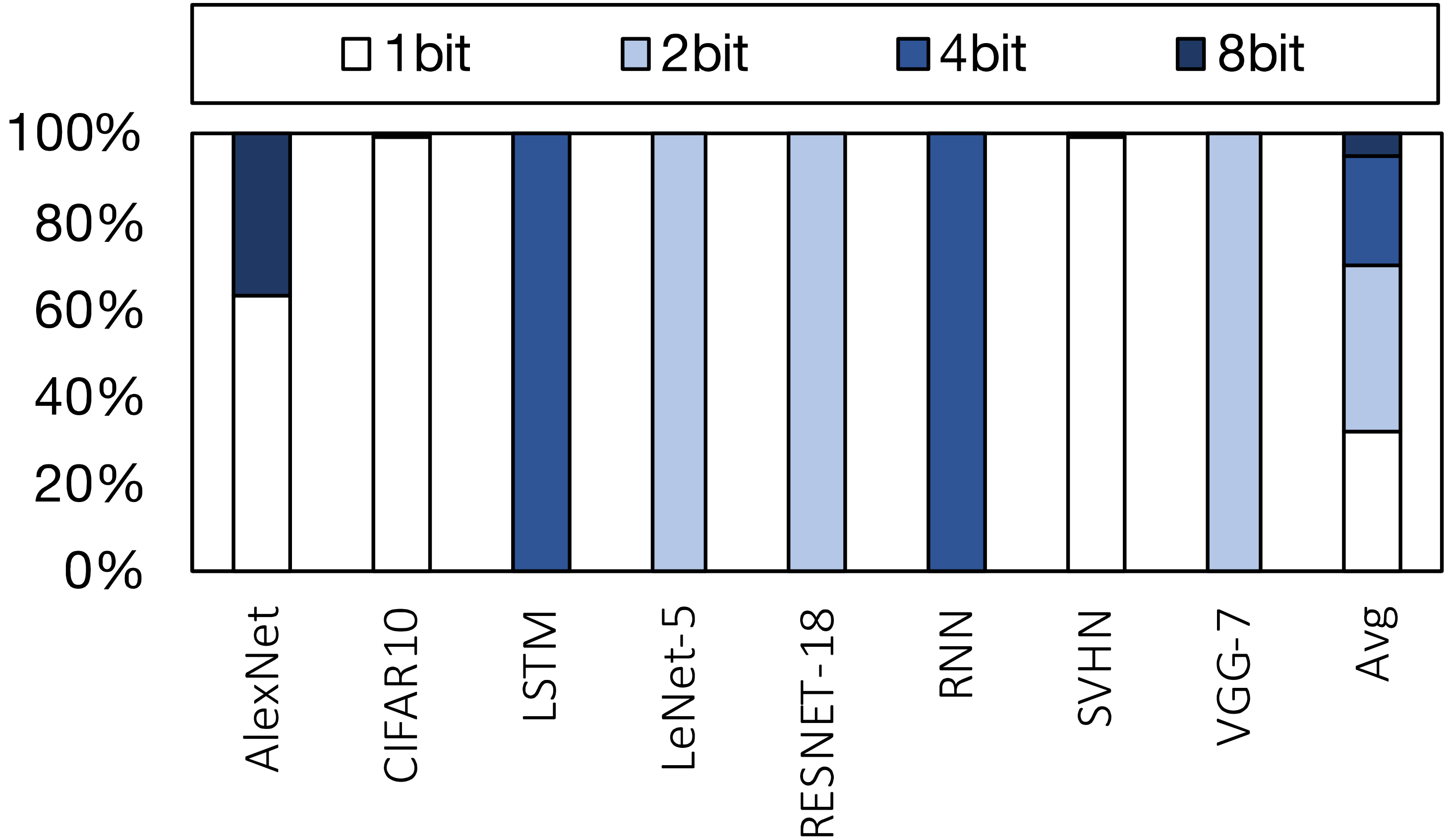
BitFusion ISA exposes this capability to **software stack**

DRAM Accesses Bottleneck Energy Benefits



DRAM accesses consume **~70%** of total Energy

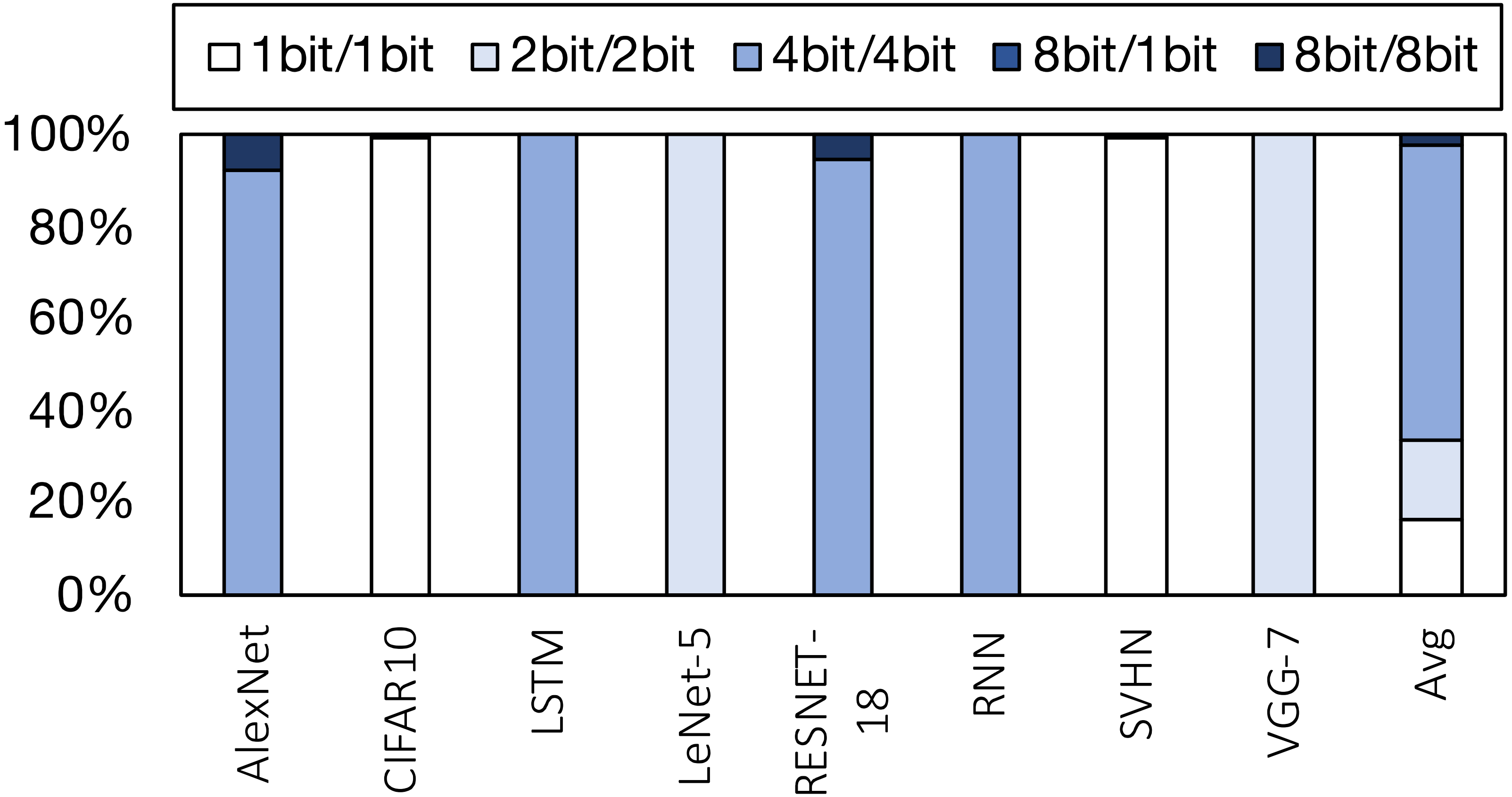
Tolerance to low bitwidth in DNN weights



> 95% of DNN weights require **less than 8-bits**

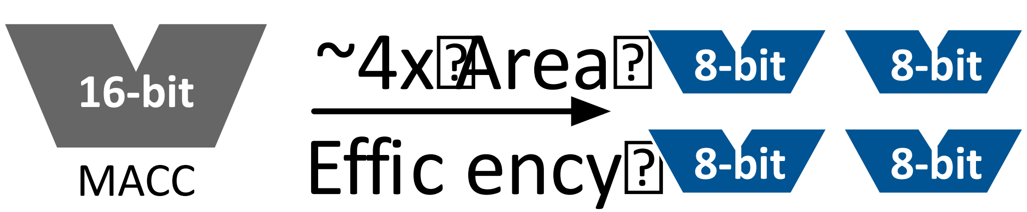
DNNs Tolerate Low-Bitwidth Operations

DNN	% Multiply-Add
AlexNet	99.8 %
CIFAR10	99.8 %
LSTM	99.9 %
LeNet-5	99.4 %
RESNET-18	99.9 %
RNN	99.9 %
SVHN	99.8 %
VGG-7	99.5 %

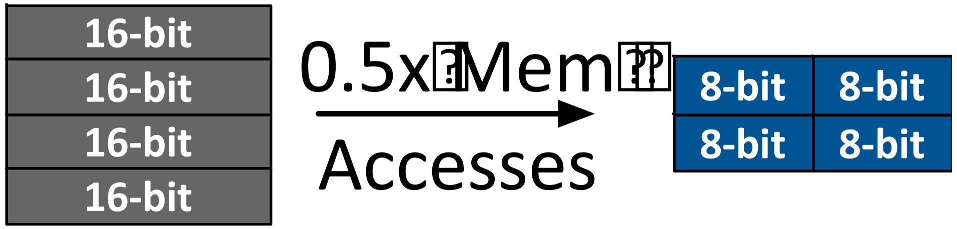


>99.4% Multiply-Adds require **less than 8-bits**

Opportunity for performance/energy benefits



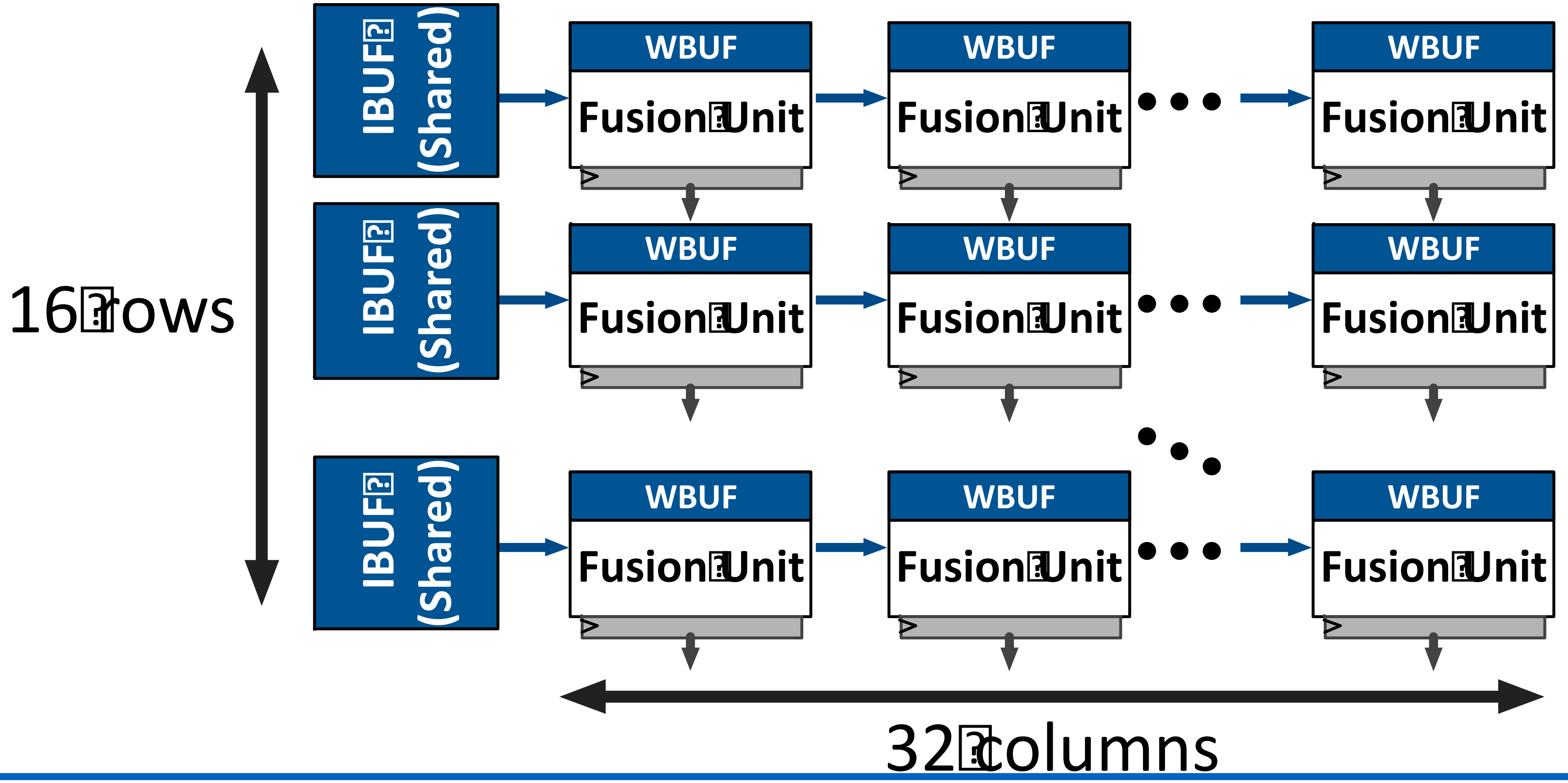
$$\text{MACC Area} \sim O(bitwidth^2)$$



$$Energy \sim O(bitwidth)$$

Quadratic improvement in Speedup, Linear in Energy

BitFusion Systolic Array



BitFusion offers **512 Fusion Units** and **8192 BitBricks**

same area budget (5.87 mm^2) as Eyeriss (168 PEs)

Loop Blocking

```
loop: b -> (B)
  loop: oc -> (OC)
    loop: ic -> (IC)
```

(a) Initial code

```
loop: b -> (B)
  loop: $t_{oc}$ -> (1, $\#tile_{oc}$)
    loop: $t_{ic}$ -> (1, $\#tile_{ic}$)
      loop: oc -> (1, $tile_{oc}$)
        loop: ic -> (1, $tile_{ic}$)
```

(b) Optimized code

Loop blocking to maximize on-chip data reuse

Loop Reordering

```
loop: b -> (B)
  loop: $t_{oc}$ -> (1, #$tile_{oc}$)
    loop: $t_{ic}$ -> (1, #$tile_{ic}$)
      loop: oc -> (1, $tile_{oc}$)
        loop: ic -> (1, $tile_{ic}$)
```

(a) Output stationary

↔

```
loop: b -> (B)
  loop: $t_{oc}$ -> (1, #$tile_{oc}$)
    loop: $t_{ic}$ -> (1, #$tile_{ic}$)
      loop: ic -> (1, $tile_{ic}$)
        loop: oc -> (1, $tile_{oc}$)
```

(b) Input stationary

Loop reordering allows switching between Input stationary and output stationary, depending on DNN layer