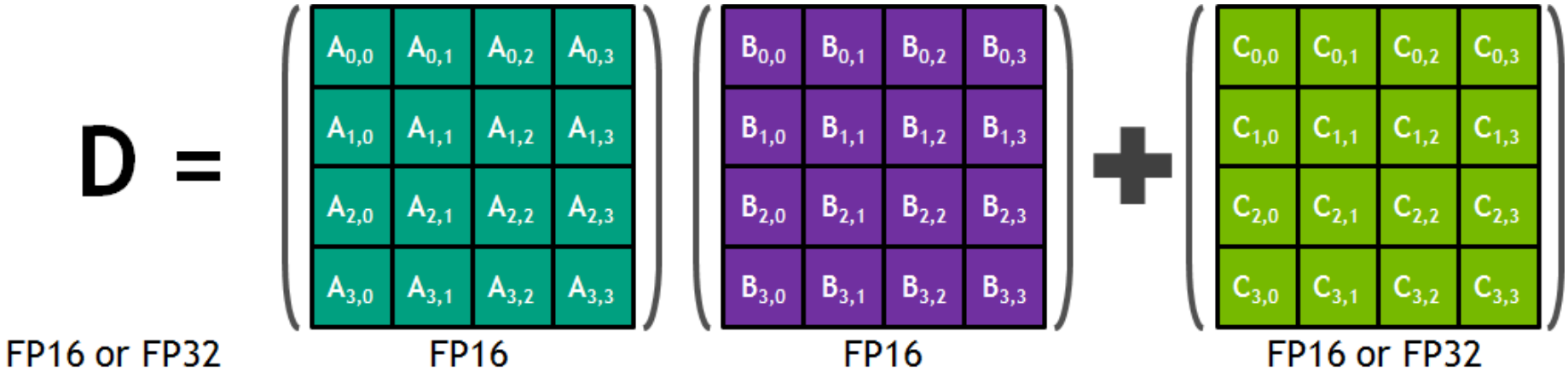
Lecture: Training Innovations

- Topics: NVIDIA Volta, Intel NNP-T/I, ScaleDeep, vDNN
- No class on Tuesday

NVIDIA Volta GPU

- 640 tensor cores
- Each tensor core performs a MAC on 4x4 tensors
- Throughput: 128 FLOPs x 640 x 1.5 GHz = 125 Tflops
- FP16 multiply operations
- 12x better than Pascal on training and 6x better on inference
- Basic matrix multiply unit – 32 inputs being fed to 64 parallel multipliers; 64 parallel add operations



Reference: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

Intel NNP-T

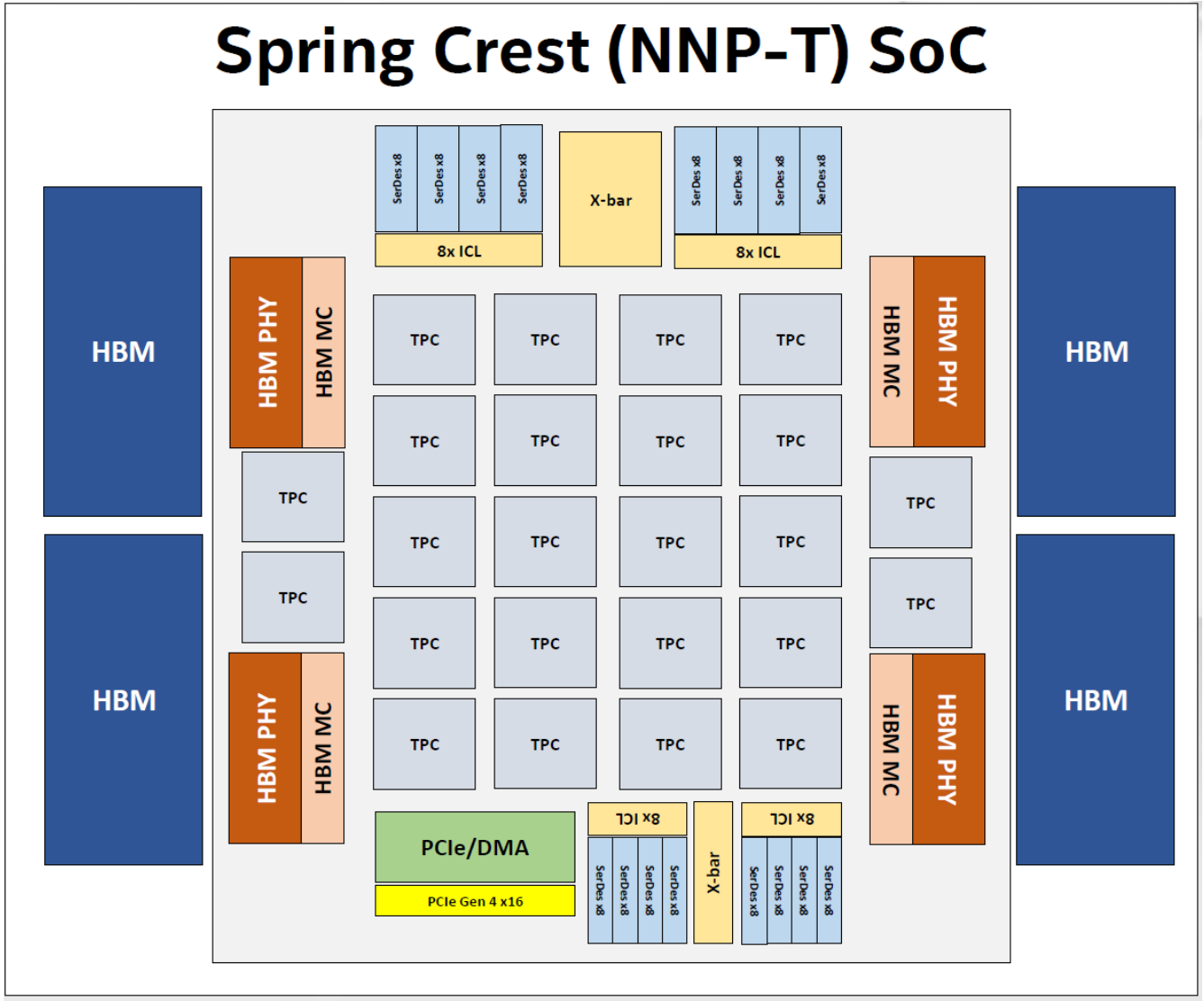


Image Source: Intel presentation HotChips'19

Intel NNP-T

- Each Tensor Processing Cluster (TPC) has two 32x32 MAC grids supporting bfloat16 (appears similar to TPU)
- Conv engine to marshall data before each step
- 60MB on-chip SRAM (2.5MB scratchpad per TPC)
- Communication is key: grid network among TPCs, 64 SerDes lanes (3.6 Tb/s) for inter-chip communication, 4 HBMs
- Relatively low utilization for GeMM (< 60%) and Conv (59-87%)

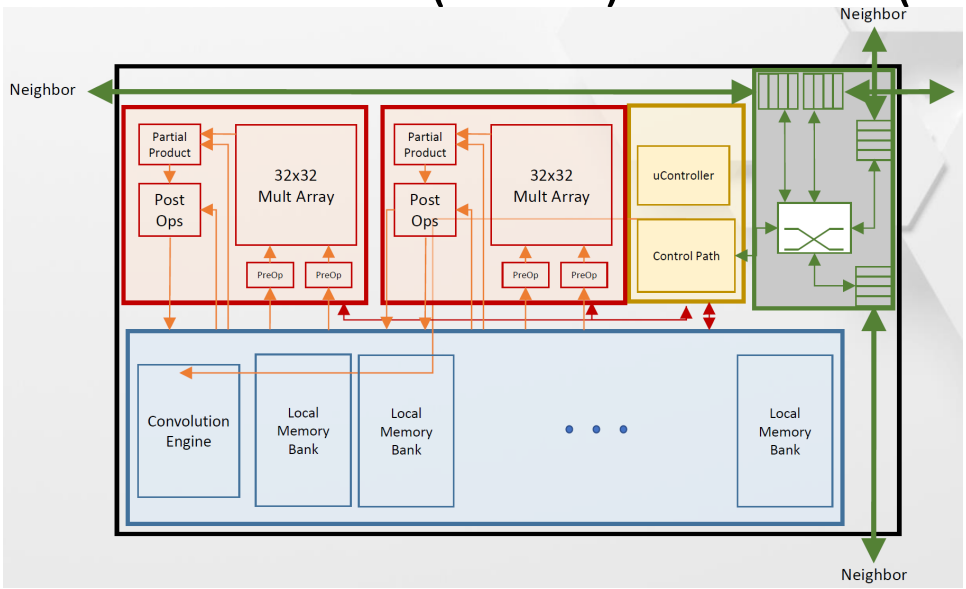
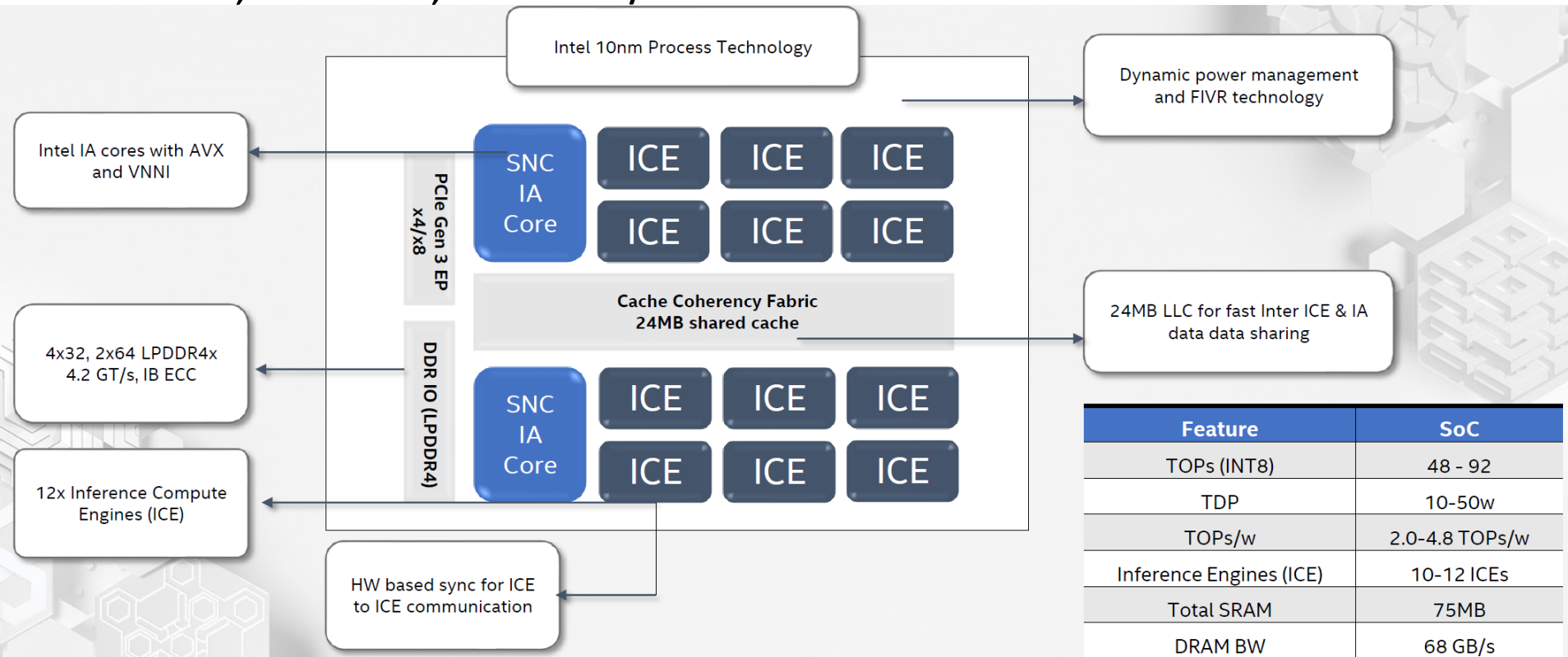


Image Source: Intel presentation HotChips'19

Intel NNP-I

- 12 Inference Compute Engines (ICE) that can work together or independently; 24MB central cache and 4MB per ICE
- Each ICE has 4K 8b MAC unit
- 10W, 48 TOPs, 3600 inf/s



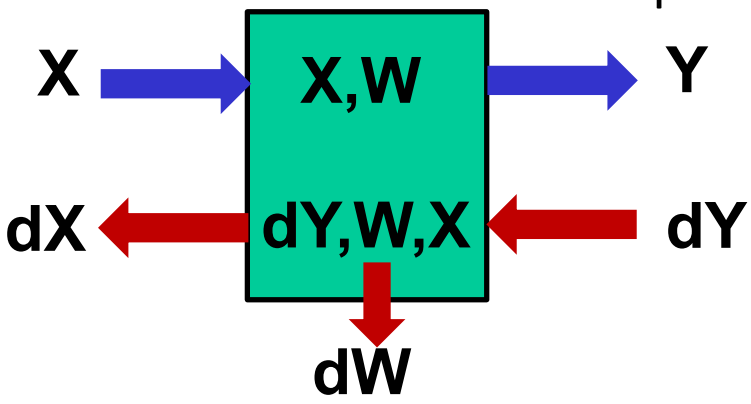
Feature	SoC
TOPs (INT8)	48 - 92
TDP	10-50w
TOPs/w	2.0-4.8 TOPs/w
Inference Engines (ICE)	10-12 ICEs
Total SRAM	75MB
DRAM BW	68 GB/s

Image Source: Intel presentation HotChips'19

Training Take-Homes

- Create a mini-batch which is a random set of training inputs; apply the fwd pass; compute error; backprop the errors (while using a transpose of weights); compute the deltas for all the weights (using the Xs of previous layer); aggregate the deltas and update the weights at the end of the mini-batch; keep creating mini-batches until you run out of training samples; this is one epoch.

- On the fwd pass, we use inputs X and weights W to compute output Y
- On the bwd pass, we use dY and weights W to compute dX;
  we also use dX and X to compute dW



ScaleDeep Intro

- Pays attention to training (network, storage)
- Introduces heterogeneity
- Uses a spatial pipeline
  (instead of time-multiplexed execution)
- Heavy use of batching for efficiency
- Novel interconnect structure (for a 1.4 KW server) that helps with batching and training

Layer Breakdown

High weight reuse;
Large feature maps

High neuron reuse;
Many feature maps;
80% of all computations

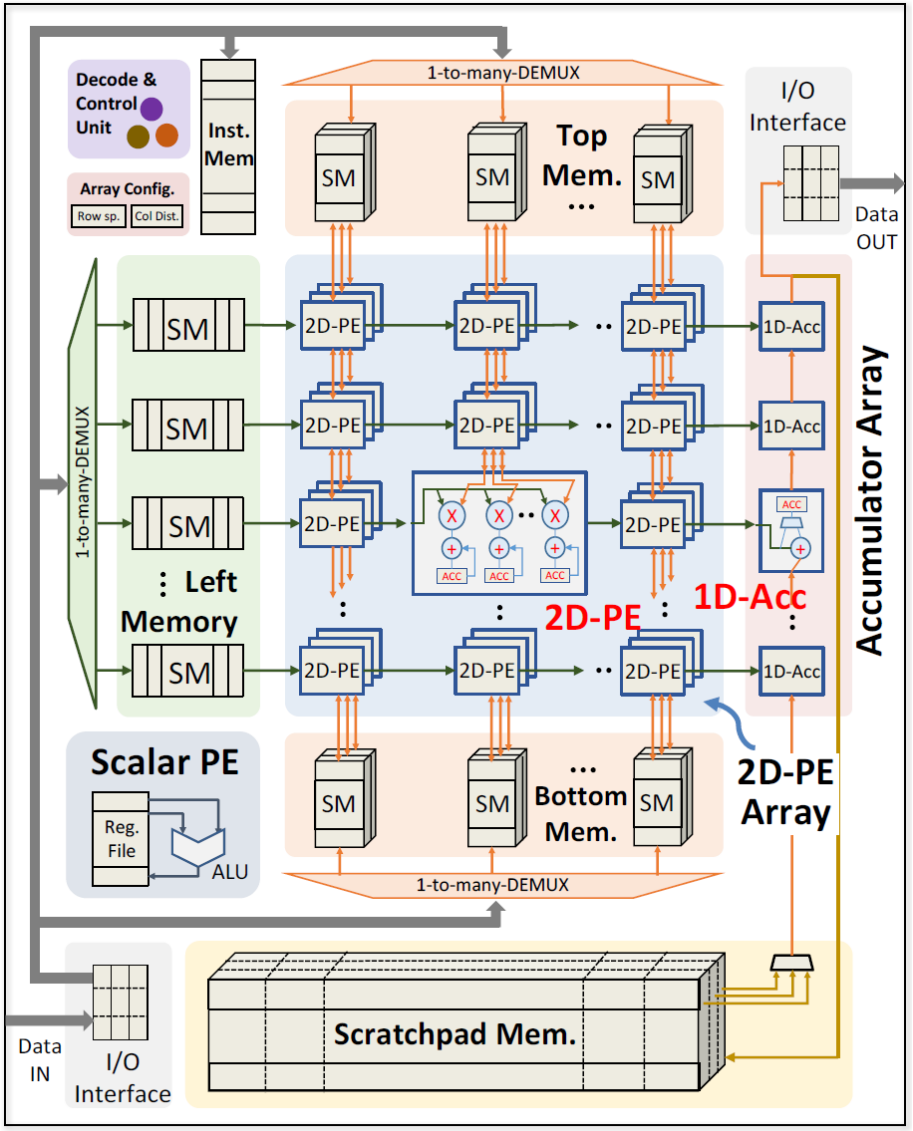
Large memory;
High bytes/flop

Type	Initial Conv.	Mid Conv.	Fully Conn.	Sub Samp.
Layer	Input, C1, C2	C3, C4, C5	F1, F2, F3	S1, S2, S3
Feature Count	3-256	256-1024	1K-4K	96-1024
Feature Size	24x24 - 231x231	12x12	1	12x12 - 56x56
Weights	30K-600K	1M-10M	4M-120M	---
Data (SP Float)	Feature 560K - 1.14M; Weight 132K - 2.3M	280K – 560K; 4.5M – 36M	12K – 4K; 15M – 432M	140K – 290K; ---
FLOPs → FP+BP; Bytes/FLOP →	11%; 0.006; Conv. 98.3, Acc. 1.6, Act. 0.1	54%; 0.015; Conv. 94.6, Acc. 5.3, Act. 0.1	3%; 2; Mat Mul. 99.9, Act. 0.1	0.1%; 5; Up/Down sampling 100
WG	5%; 0.005; Conv. 100	26%; 0.014; Conv. 100	0.9%; 4; Vec. Ele. Mul. 100	Bytes /FLOP: Low, Medium, High

Note: FP, BP, WG

Figure 4: Breakdown of compute and data requirements for OverFeat DNN

CompHeavy Tile



Kernels enter from top and down

Can do two small kernels or 1 large kernel

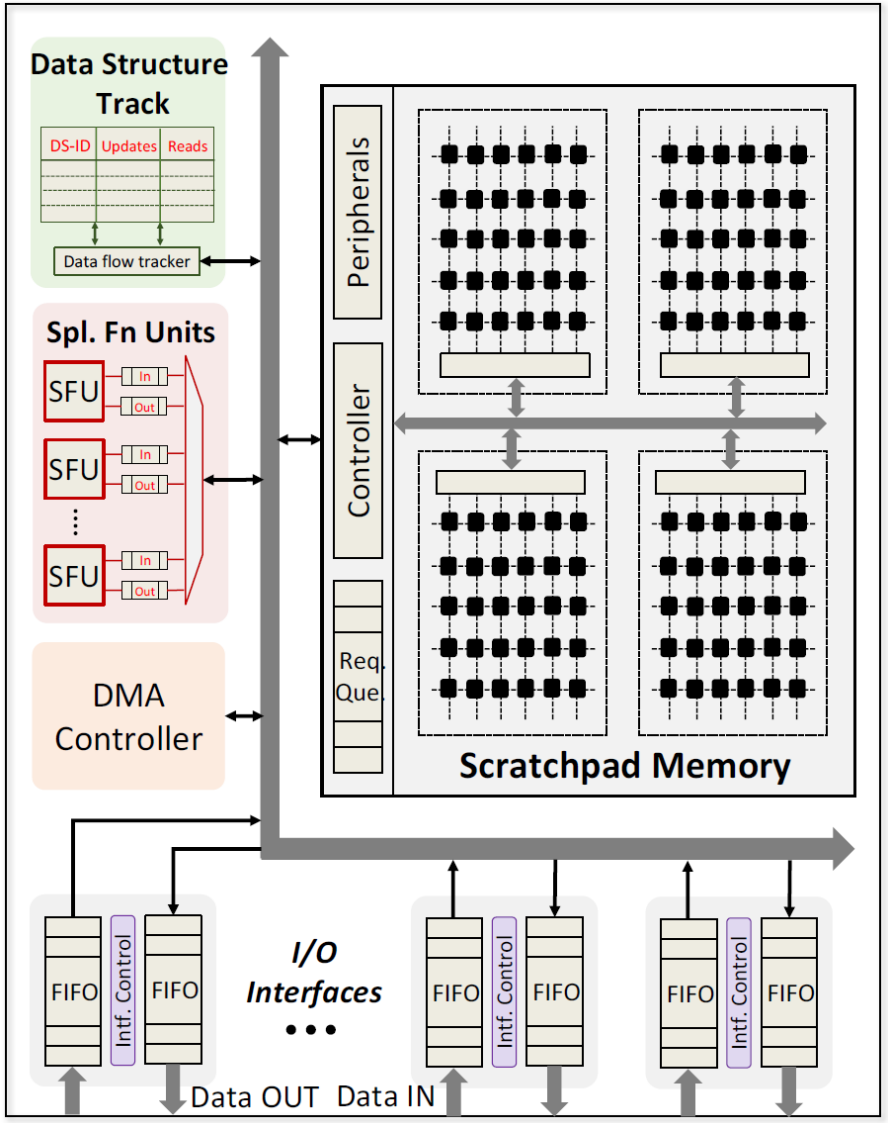
Inputs enter from left memory

Accumulation happens on right

Scratchpad used for partial sums

Multiple lanes in each PE (can handle different kernels)

MemHeavy Tile

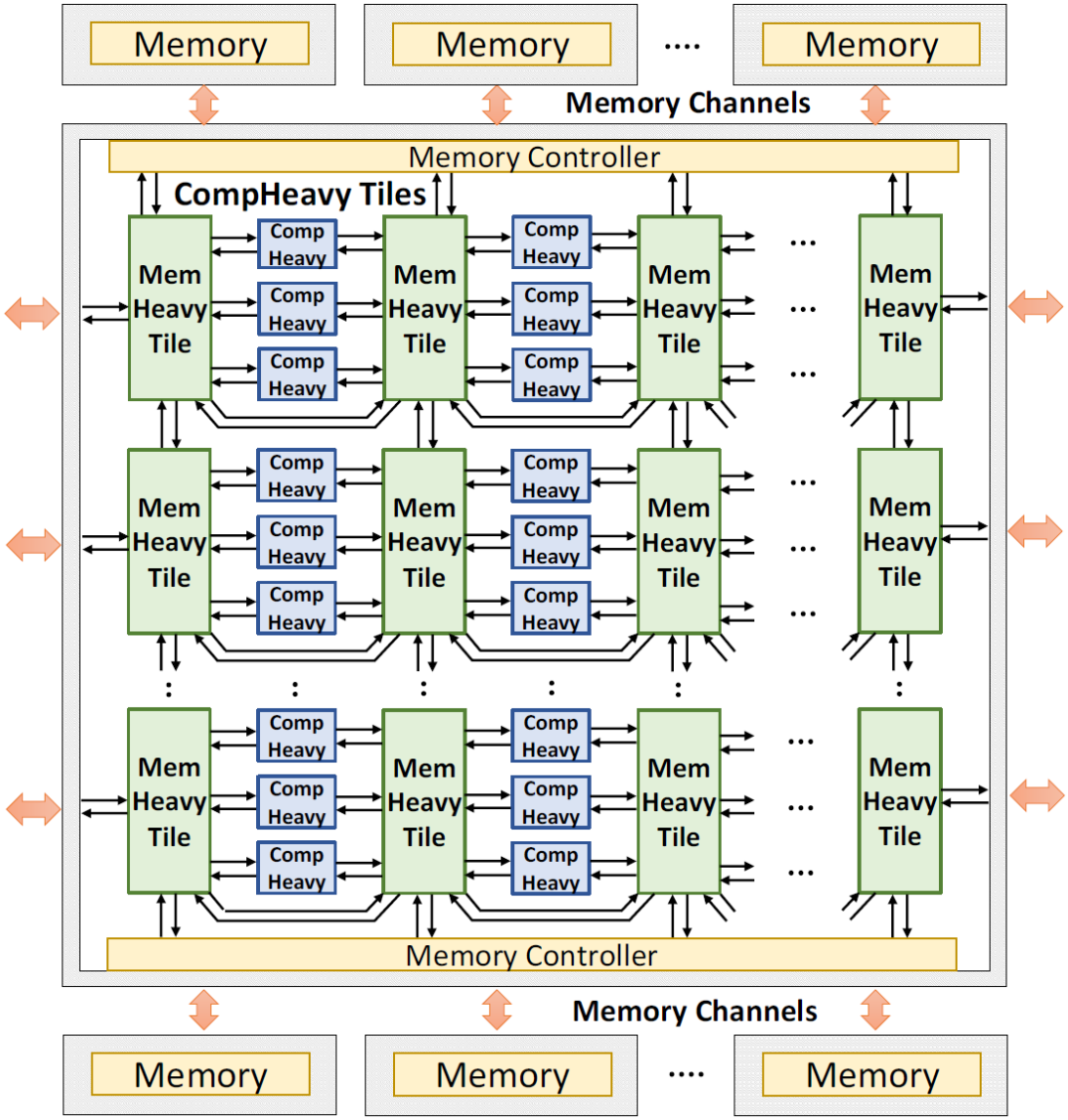


Intra-chip heterogeneity

Mainly used for data storage.

Can compute activations and pooling.

ScaleDeep Chip

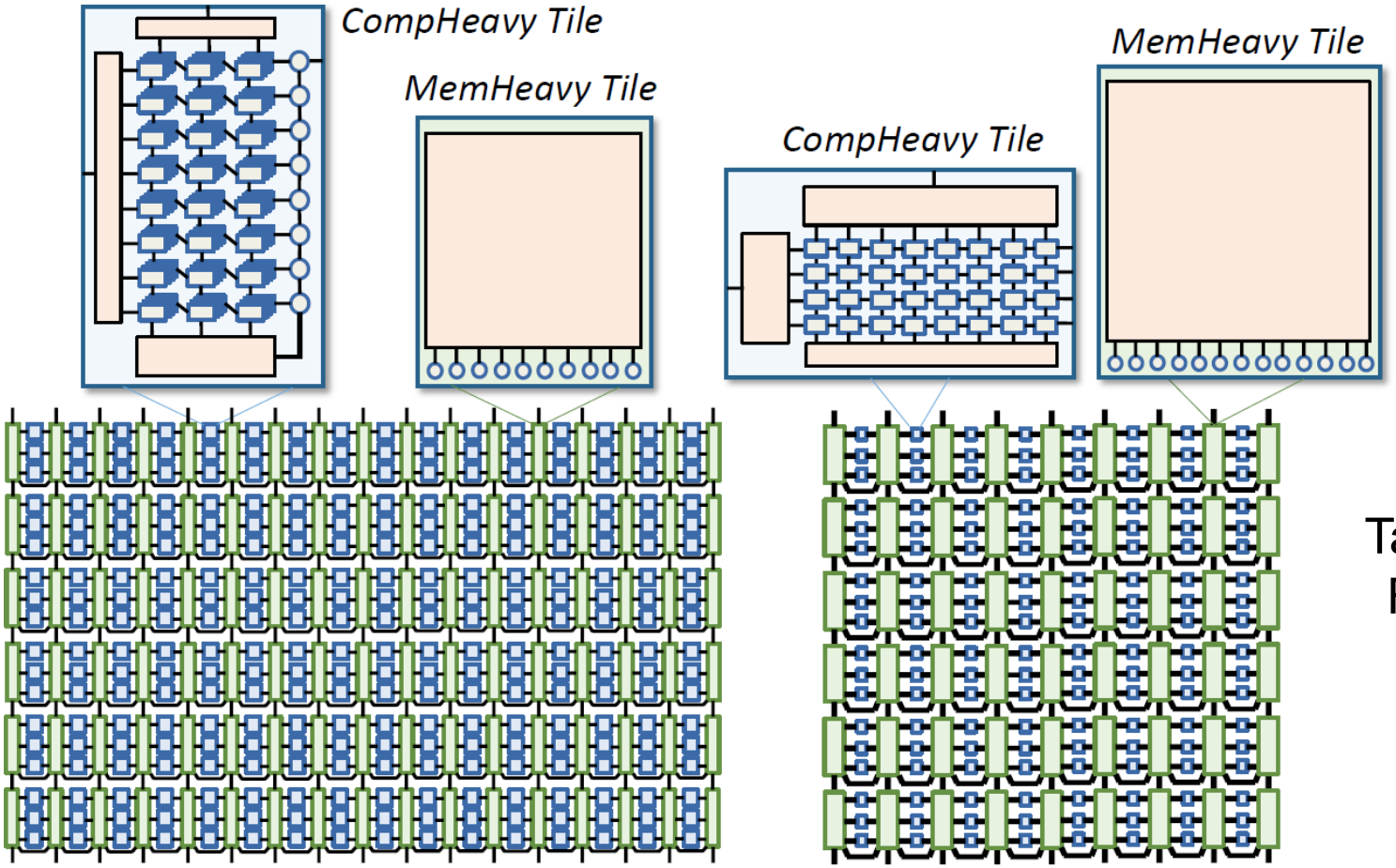


3 Comp tiles per Mem tile (for FP, BP, and WG)

Tiles are dedicated for specific layers (like ISAAC)

Latency/thruput trade-off

Chip Heterogeneity

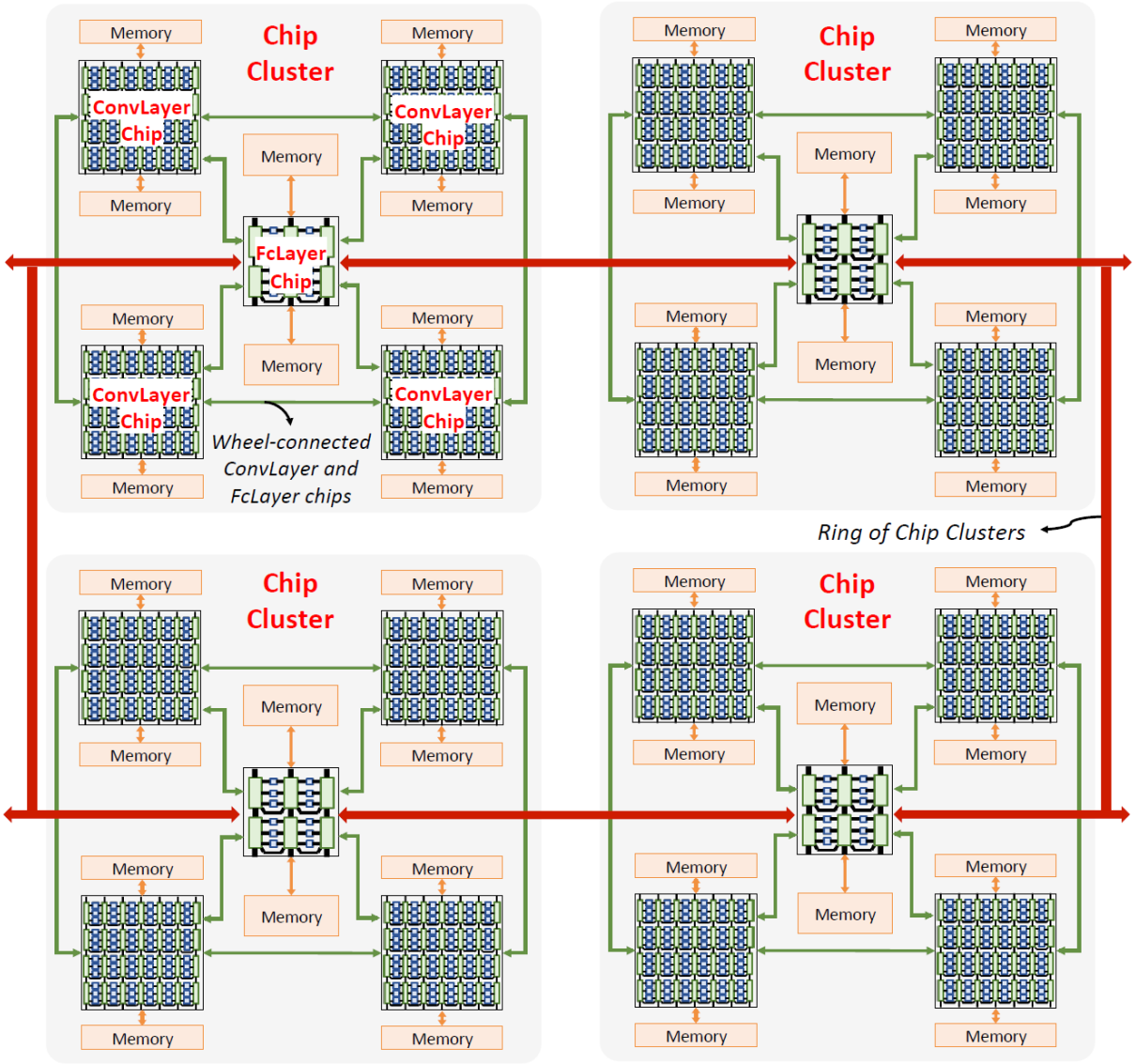


Tailored for FC layer

ConvLayer Chip: More compute → CompHeavy tiles larger in size and number

FcLayer Chip: MemHeavy tiles with large data array, high bandwidth links

ScaleDeep Node



Each ConvChip works on a different image.

The outputs converge on the FC chip (spokes) so it can use batching.

The wheel arcs are used for weight updates and for very large convs.

The FC layer is split across multiple FCchips; increases batching, reduces memory, low network bw

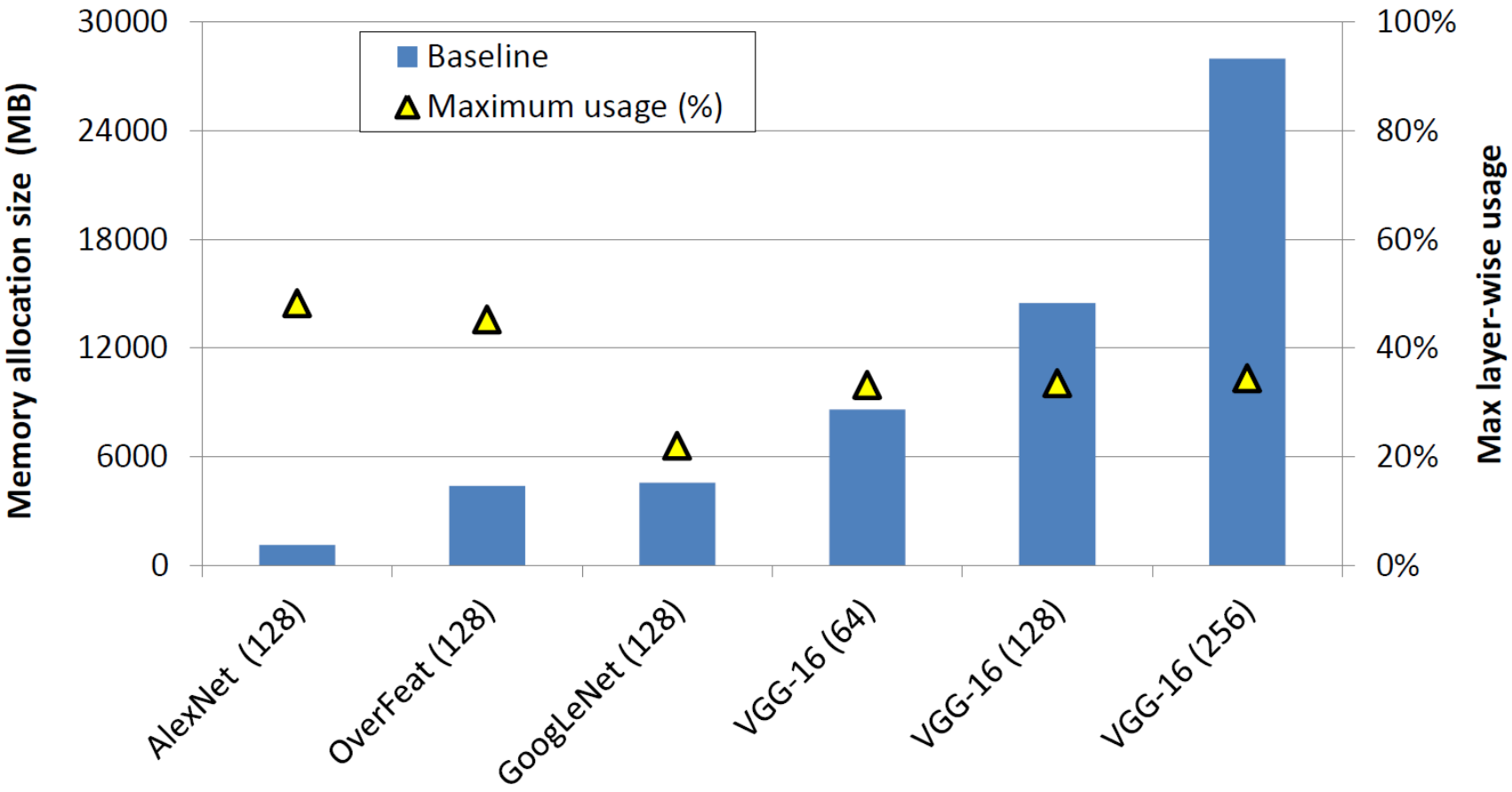
Results Summary

- Both chips have about 50 MB capacity
- Convchip consumes 58 W, FCchip consumes 15 W
- Inference is 3x faster than training
- One chip-cluster (320 W, comparable to a GPU) has 5-11x speedup over the GPU
- 5x speedup over DaDianNao at iso-power

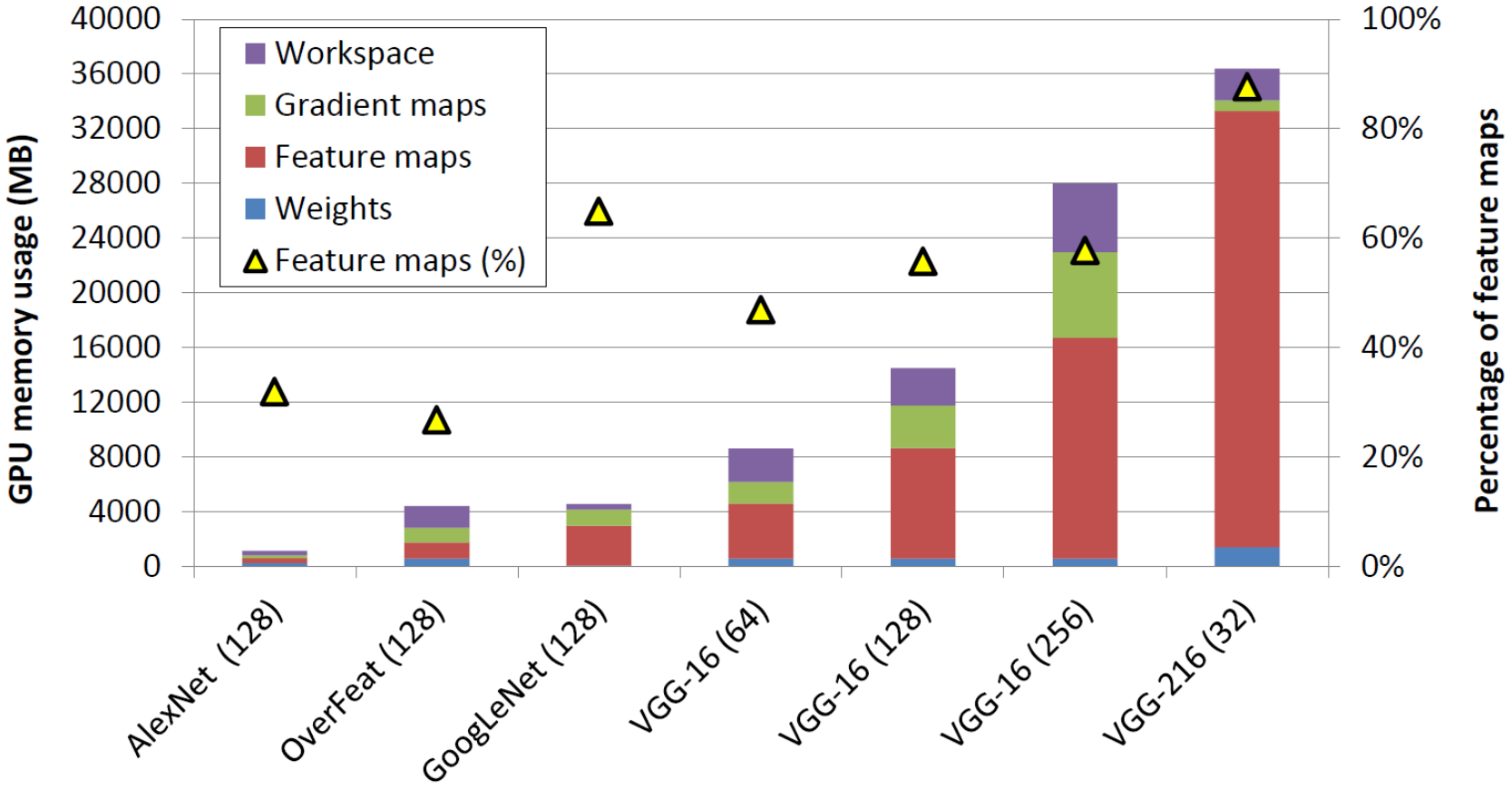
Training on GPUs

- GPUs have limited memory capacity (max 12 GB)
- When training, in addition to storing weights, we must store all the activations as they are required during backprop; plus the weight deltas
- In convolution layers, the activations consume way more memory than the weights; convolution activations also have longer reuse distance than classifier weights
- vDNN is a runtime that automatically moves activations to CPU memory and back, so the network fits in GPU memory (avoids constrained network, multi GPUs, small batches, slower algos)

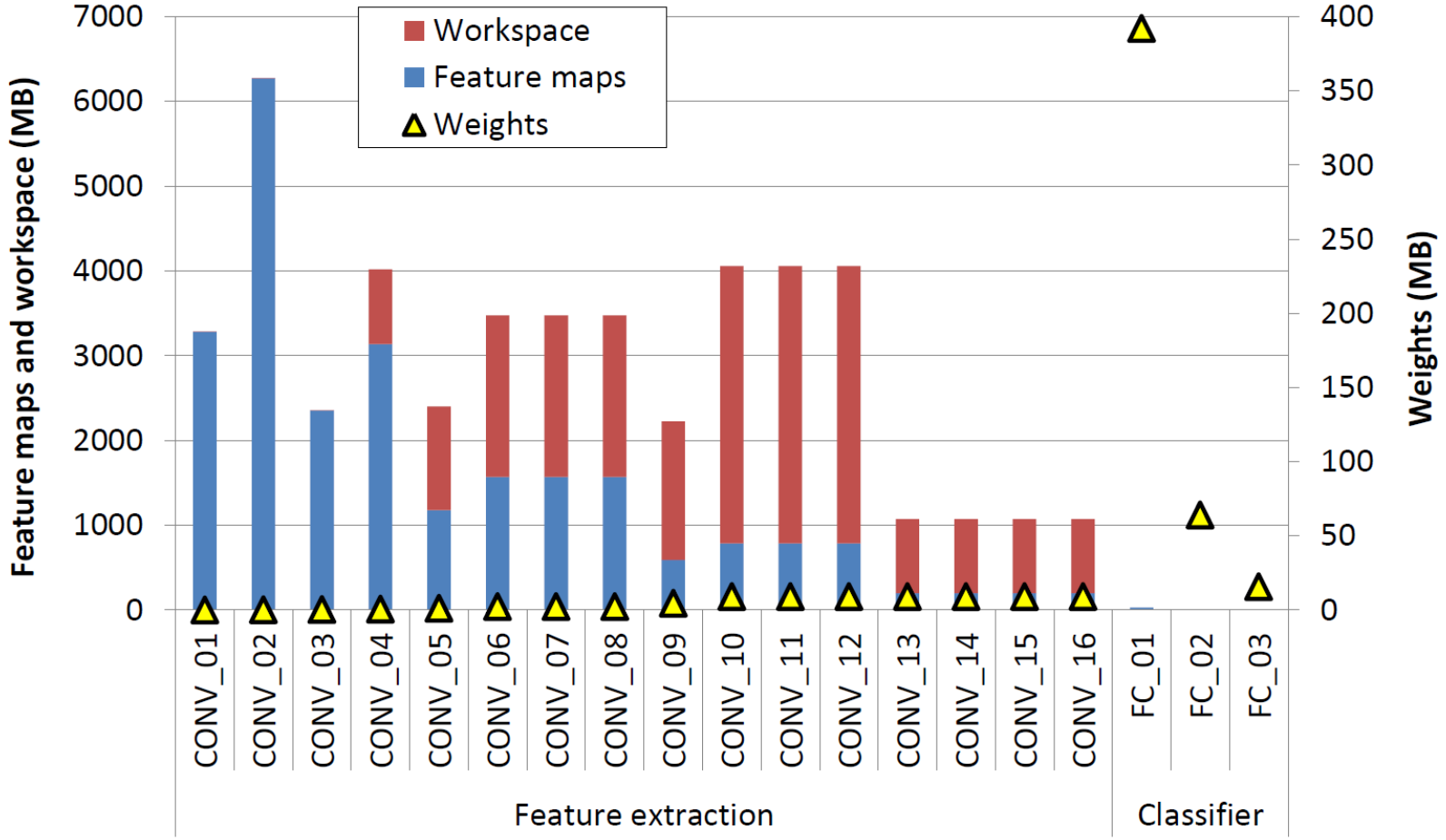
Motivation I



Motivation II



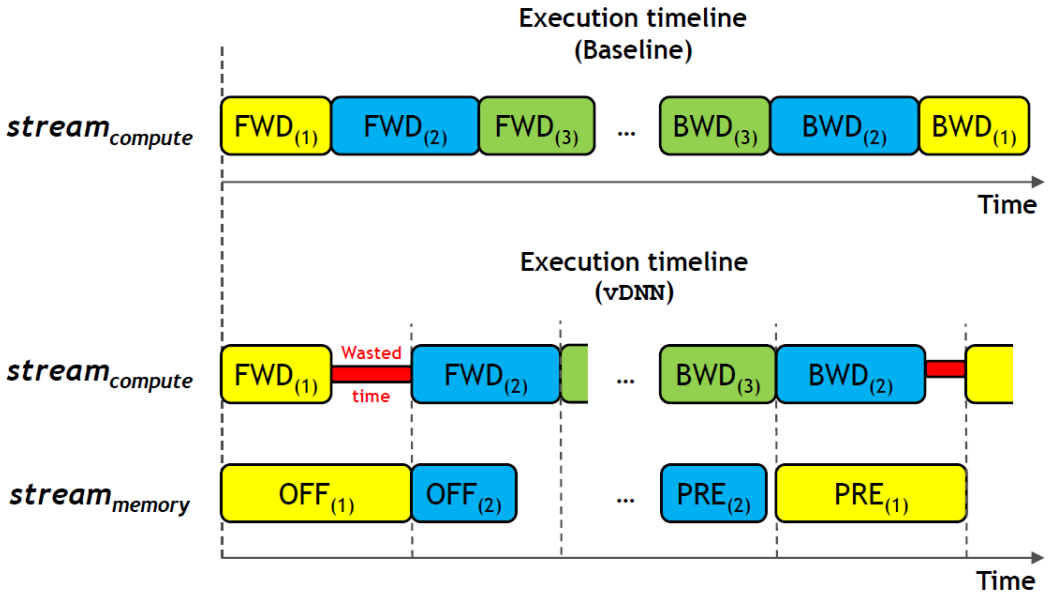
Motivation III



Can use frequency-domain FFT algorithms that are faster, but need additional workspace

vDNN Proposal

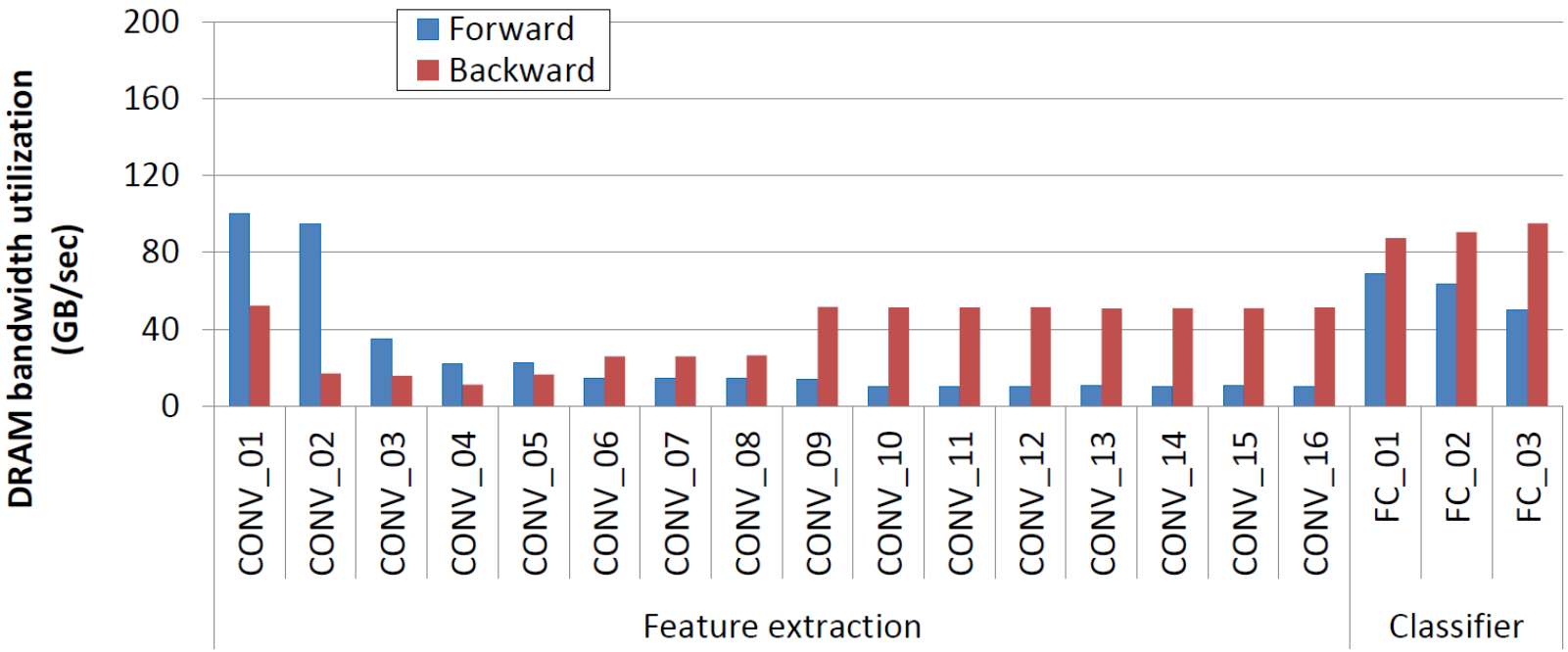
- Runtime system that, during fwd pass, initiates an off-load of a layer's X while it is computing on X (no races because X is read-only)
- The layer waits if the offload hasn't finished; trying to optimize memory capacity
- During back-prop, initiate a prefetch of X for previous layer; must wait if prefetch hasn't finished



System Parameters

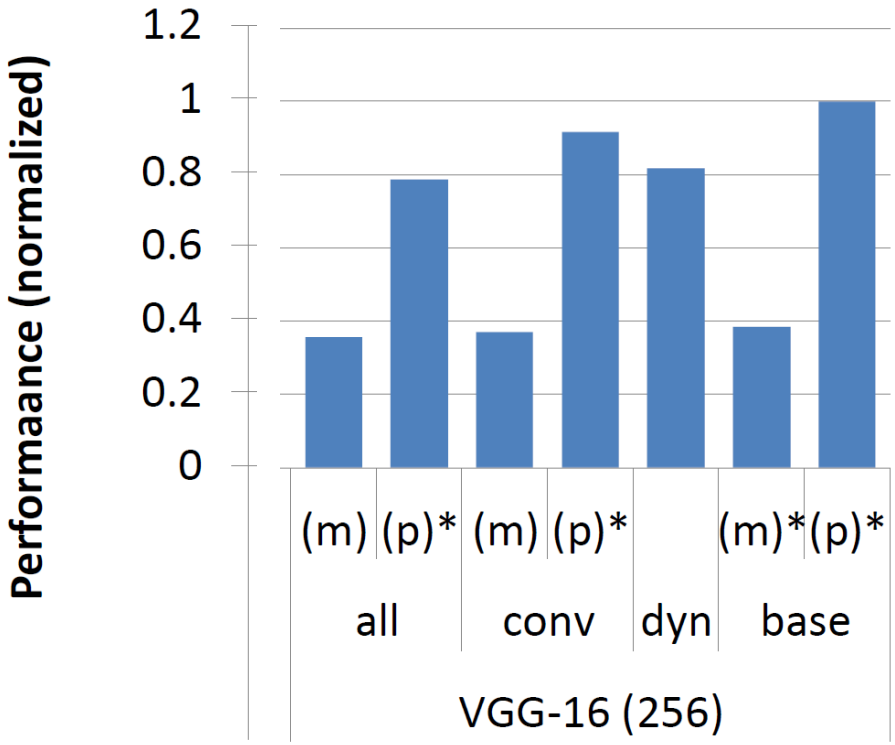
- Baseline: NVIDIA Titan X; 7 TFLOP/s; 336 GB/s GDDR5 memory bandwidth, 12 GB, 16 GB/s PCIe link for GPU←→ CPU memory transfers
- Baseline with performance features (more memory reqd)
- vDNNall: all layers perform an offload/prefetch (least memory)
- vDNNconv: only convolutional layers perform offload/prefetch
- vDNNdyn: explores many configs to identify the few layers that need to be offloaded/prefetched, and the few layers that can use the faster algos; exploration isn't expensive and uses a greedy algorithm

Memory Interference



Early layers need the most offloading and will interfere the most with GDDR access; but early layers also have the least memory accesses for weights; at most, the interference will be 16/336 = 5%

Performance Results



- The largest benchmark has 18% perf loss; others are near zero
- Dynamic scheme gives best perf while fitting in memory
- Power increase of 1-7%

References

- “Neural Networks and Deep Learning,” (Chapter 2) Michael Nielsen
- “SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks,” S. Venkataramani et al., ISCA 2017
- “vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design,” M. Rhu et al., MICRO 2016