TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory

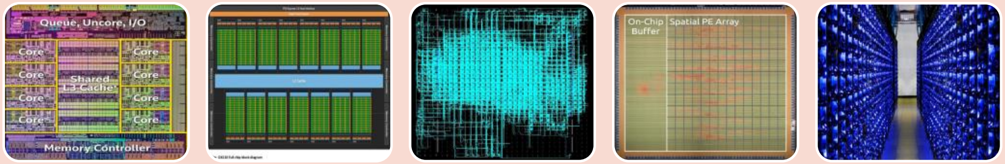
Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, Christos Kozyrakis

Stanford University

Neural Networks (NNs)

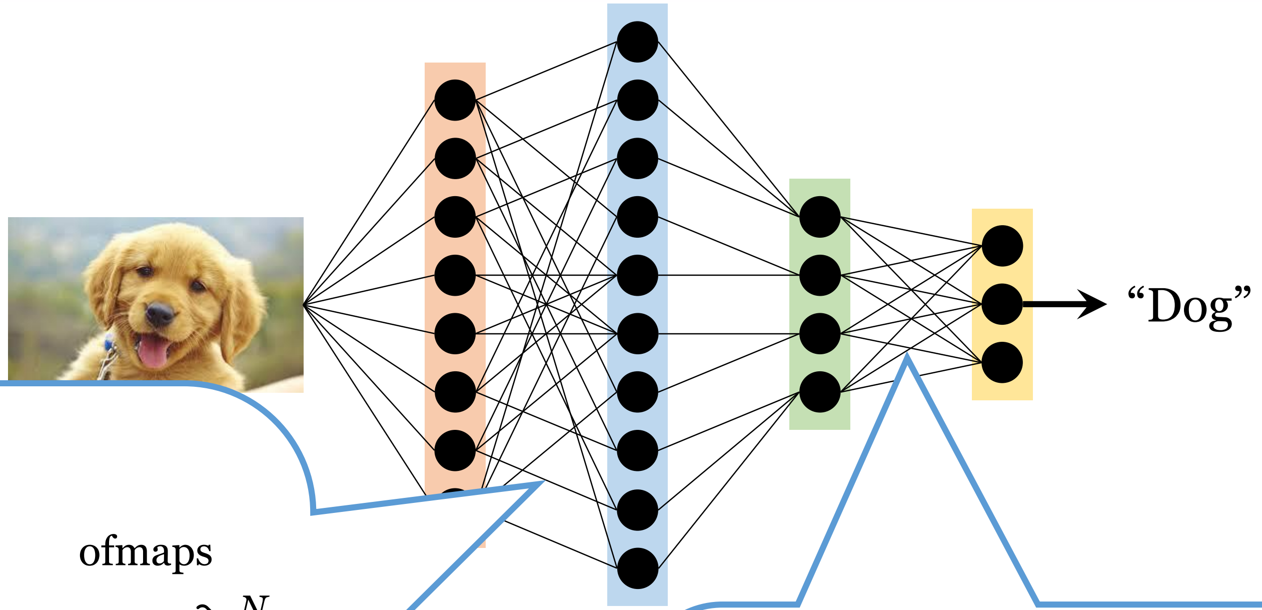
- Unprecedented accuracy for challenging applications
- System perspective: compute *and* memory intensive
  - Many efforts to accelerate with specialized hardware

- Classification
- Recognition
- Control
- Prediction
- Optimization

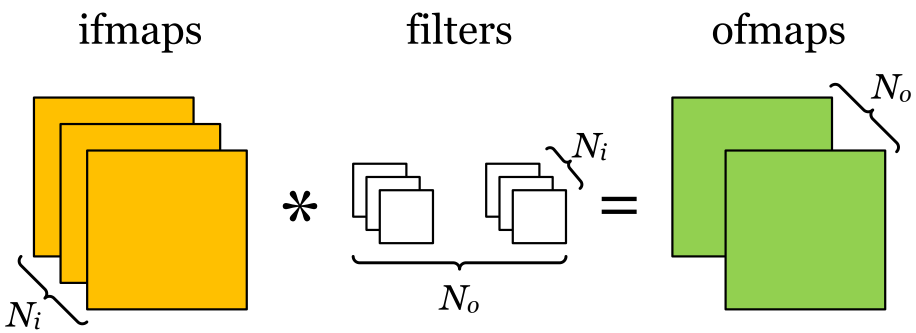


Multi-cores	GPUs	FPGAs	ASICs	Clusters

Neural Networks (NNs)



CONV




```
foreach b in batch Nb
  foreach ifmap u in Ni
    foreach ofmap v in No
      // 2D conv
      O(v,b) += I(u,b) * W(u,v) + B(v)
```

FC

$$O = I \times W$$

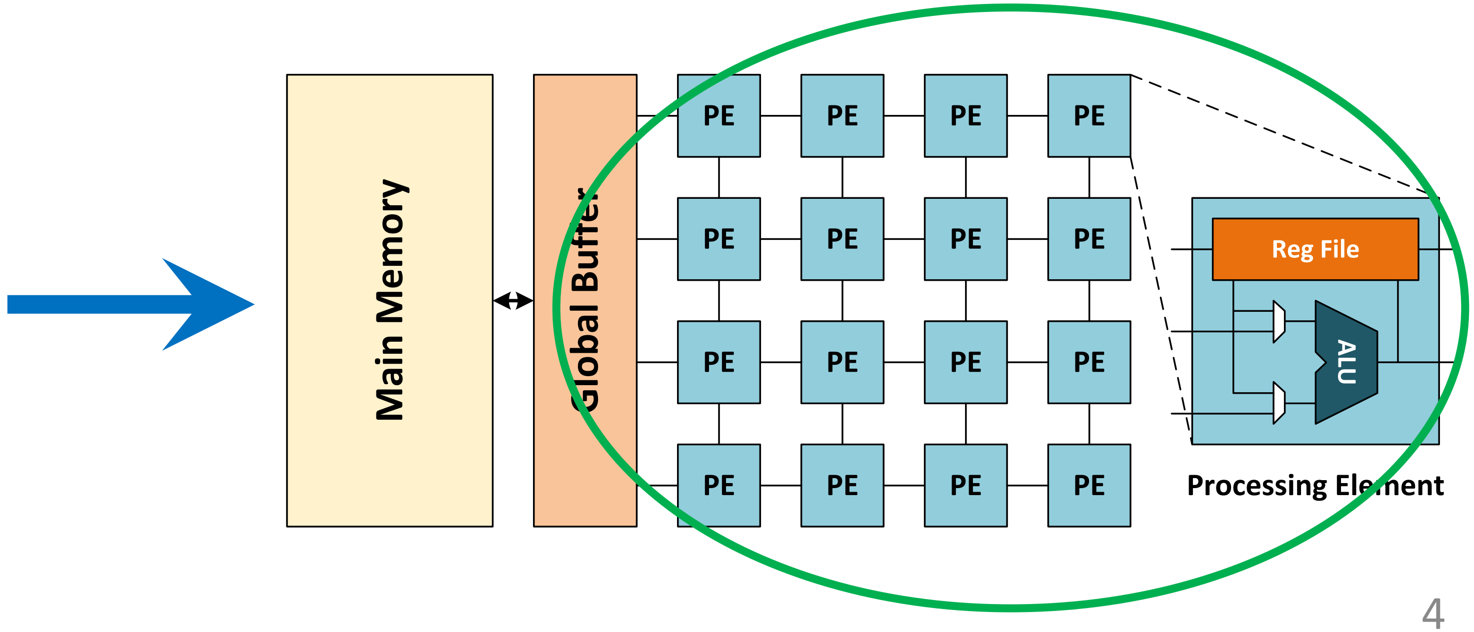
```
foreach b in batch Nb
  foreach neuron x in Nx
    foreach neuron y in Ny
      // Matrix multiply
      O(y,b) += I(x,b) x W(x,y) + B(v)
```

Domain-Specific NN Accelerators

- Spatial architectures of PEs
  - 100x performance and energy efficiency
  - Low-precision arithmetic, dynamic pruning, static compression, …

```
foreach b in batch Nb
  foreach ifmap u in Ni
    foreach ofmap v in No
      // 2D conv
      O(v,b) += I(u,b) * W(u,v) + B(v)

foreach b in batch Nb
  foreach neuron x in Nx
    foreach neuron y in Ny
      // Matrix multiply
      O(y,b) += I(x,b) x W(x,y) + B(v)
```

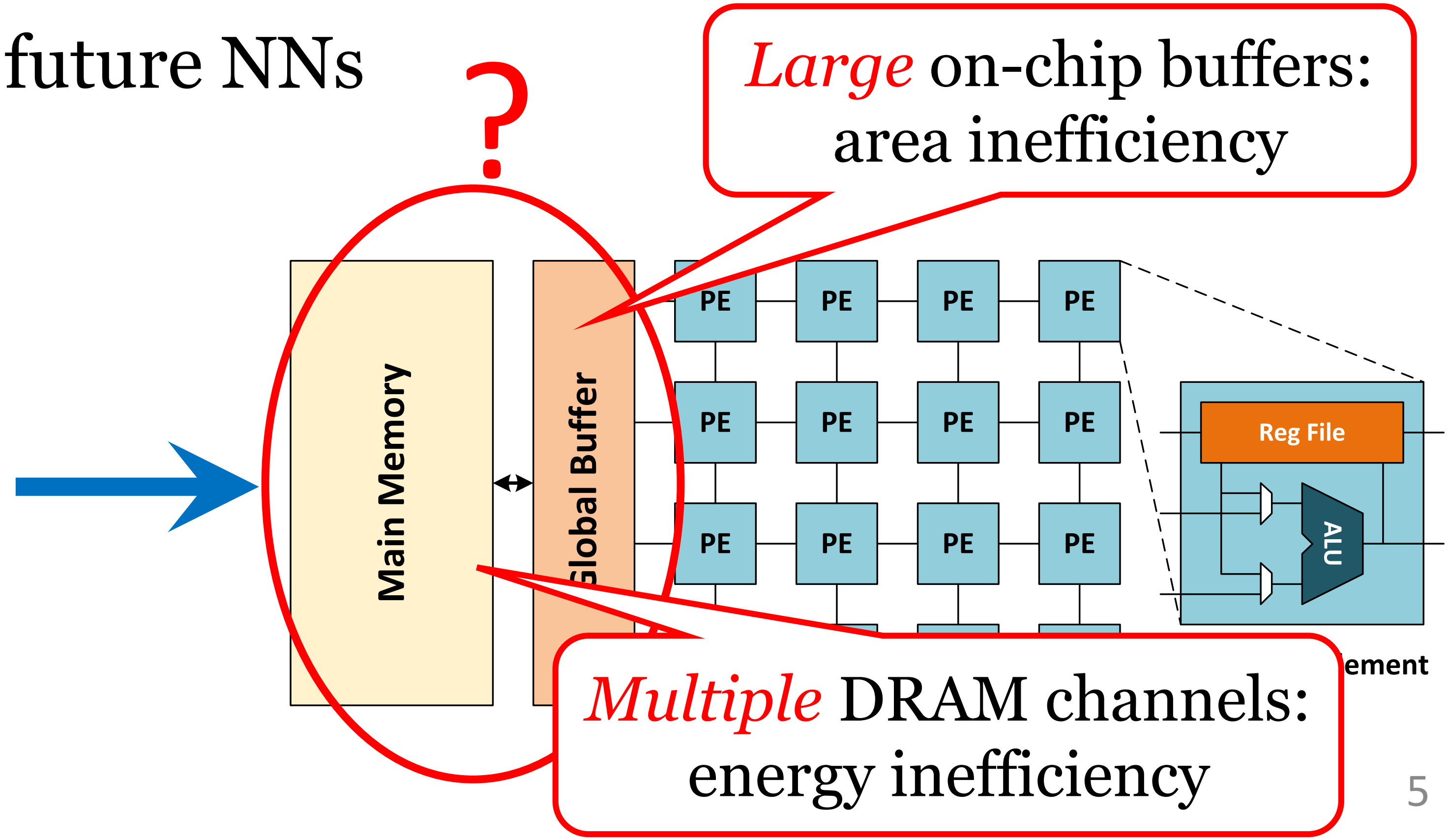


Memory Challenges for Large NNs

- Large footprints and bandwidth requirements
  - Many and large layers, complex neuron structures
  - Efficient computing requires higher bandwidth
- Limit scalability for future NNs

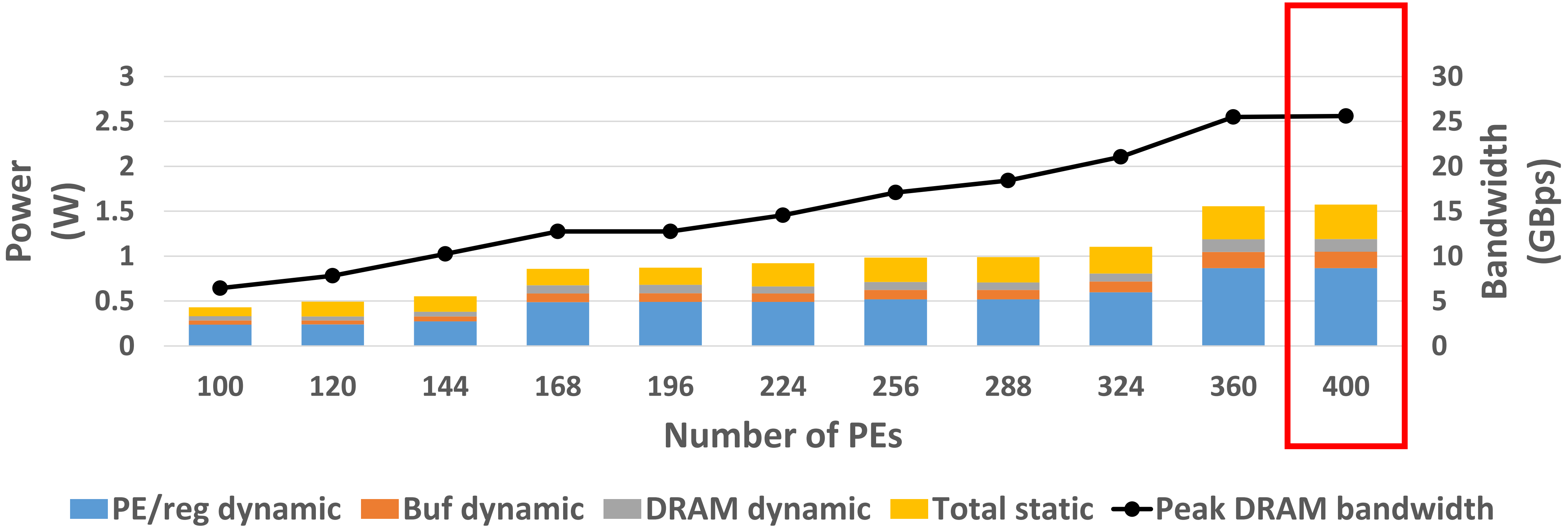
```
foreach b in batch Nb
  foreach ifmap u in Ni
    foreach ofmap v in No
      // 2D conv
      O(v,b) += I(u,b) * W(u,v) + B(v)

foreach b in batch Nb
  foreach neuron x in Nx
    foreach neuron y in Ny
      // Matrix multiply
      O(y,b) += I(x,b) x W(x,y) + B(v)
```



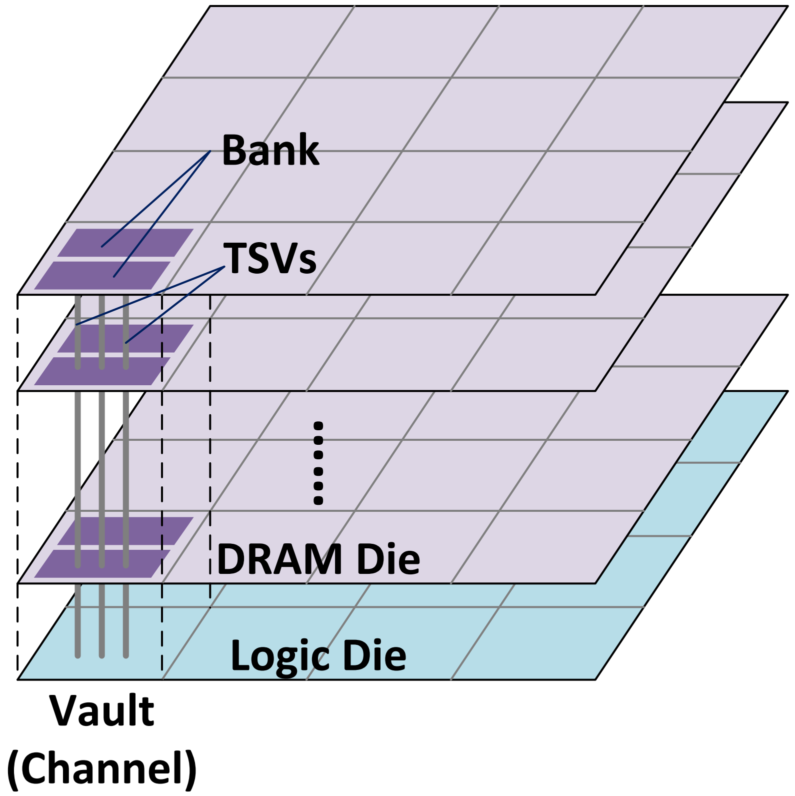
Memory Challenges for Large NNs

- State-of-the-art NN accelerator with 400 PEs
  - 1.5 MB SRAM buffer → 70% area
  - 4 LPDDR3 x32 chips → 45% power in DRAM & SRAM



3D Memory + NN Acceleration

- Opportunities
  - High bandwidth at low access energy
  - Abundant parallelism (vaults, banks)
- Key questions
  - *Hardware* resource balance
  - *Software* scheduling and workload partitioning



Micron's Hybrid Memory Cube

TETRIS

- NN acceleration with 3D memory
  - Improves *performance scalability* by 4.1x over 2D
  - Improves *energy efficiency* by 1.5x over 2D

- Hardware architecture
  - Rebalance resources between PEs and buffers
  - In-memory accumulation

High performance & low energy

Alleviate bandwidth pressure

- Software optimizations
  - Analytical dataflow scheduling for memory hierarchy
  - Hybrid partitioning for parallelism across vaults

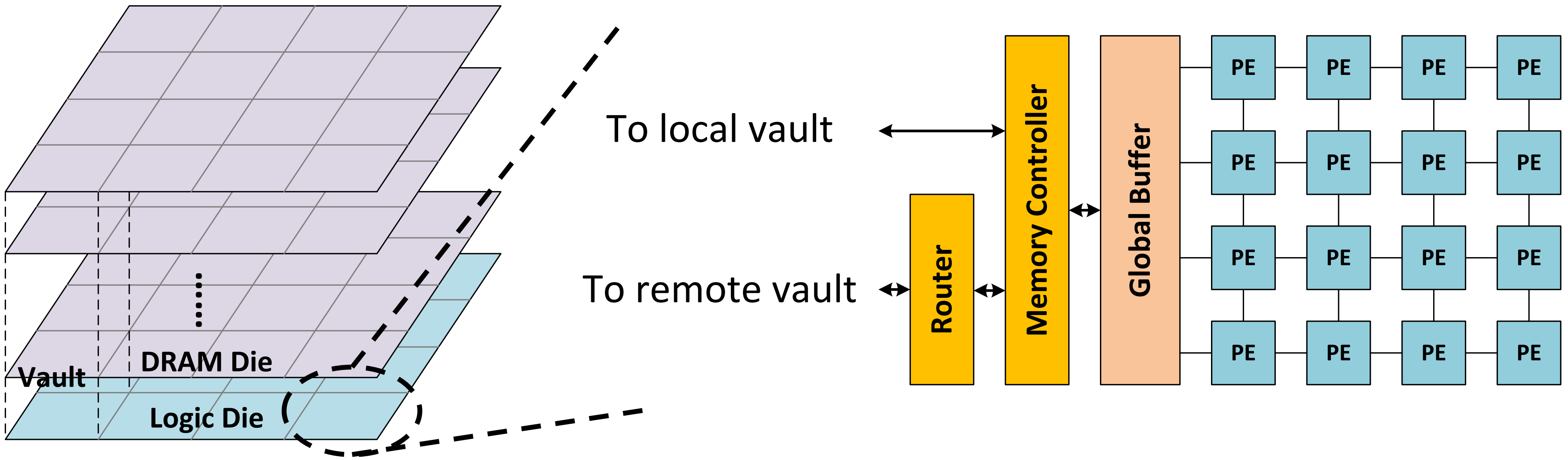
Optimize buffer use

Efficient parallel processing

TETRIS Hardware Architecture

TETRIS Architecture

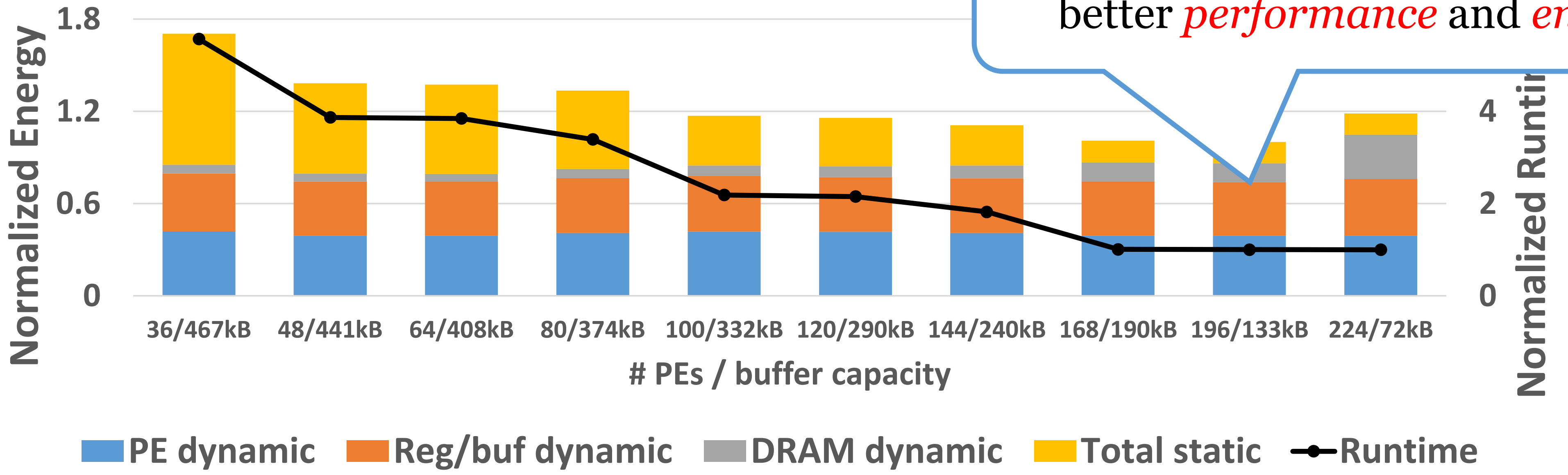
- Associate one NN engine with each vault
  - PE array, local register files, and a shared global buffer
- NoC + routers for accesses to remote vaults
- All vaults can process NN computations in parallel



Resource Balancing

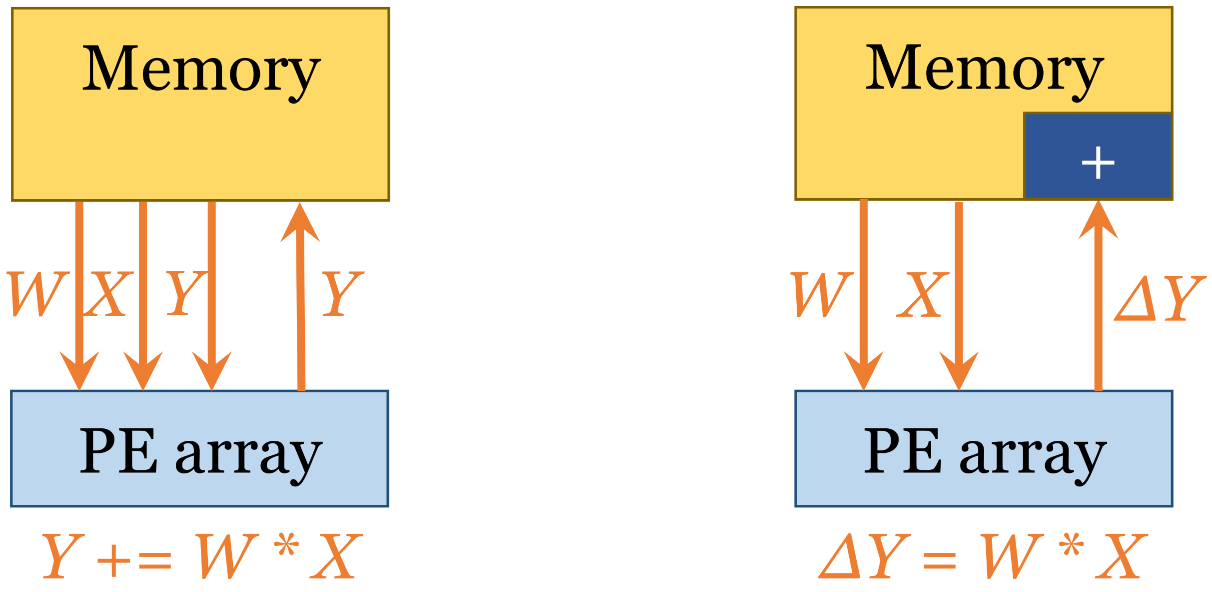
- *Larger* PE arrays with *smaller* SRAM buffers
  - High memory bandwidth → more PEs
  - Low access energy + sequential pattern → smaller buffers

196 PEs with 133 kB buffer (area 1:1) better *performance* and *energy*



In-Memory Accumulation

- Move simple accumulation logic close to DRAM banks
  - 2x bandwidth reduction for output data
  - See paper for discussion of logic placement in DRAM



Scheduling and Partitioning for TETRIS

Dataflow Scheduling

- Critical for maximizing on-chip data reuse to save energy

```
foreach b in batch Nb
  foreach ifmap u in Ni
    foreach ofmap v in No
      // 2D conv
      O(v,b) += I(u,b) * W(u,v) + B(v)
```

Ordering: loop blocking and reordering

- Locality in global buffer
- Non-convex, exhaustive search

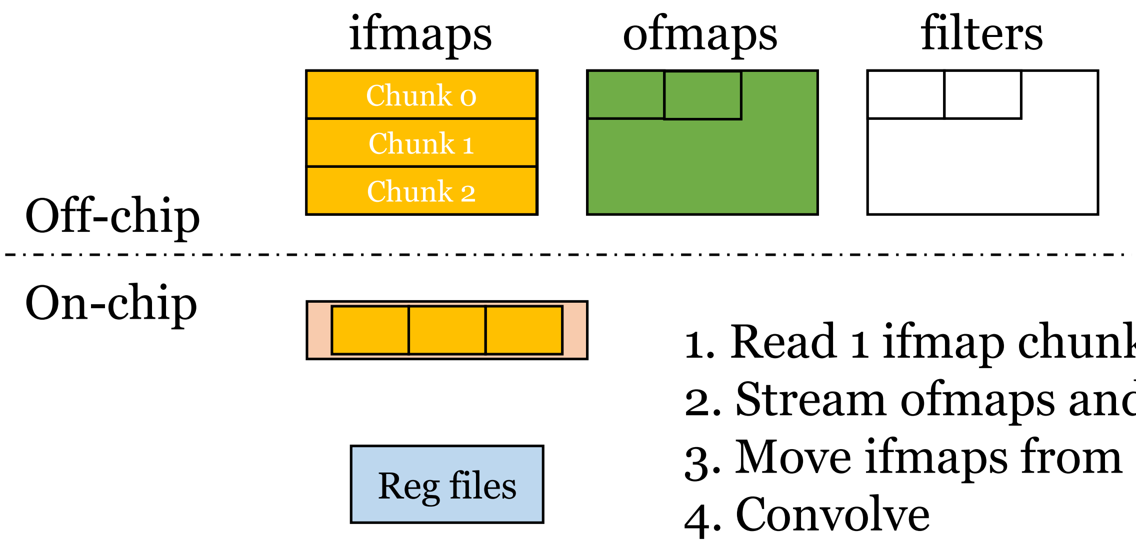
Mapping: execute 2D conv on PE array

- Regfiles and array interconnect
- Row stationary [Chen et al., ISCA'16]

TETRIS Bypass Ordering

- Limited reuse opportunities with small buffers
- IW bypass, OW bypass, IO bypass
  - Use buffer only for one stream for maximum benefit
  - Bypass buffer for the other two to sacrifice their reuse

OW bypass ordering



1. Read 1 ifmap chunk into gbuf
2. Stream ofmaps and filters to regf
3. Move ifmaps from gbuf to regf
4. Convolve
5. Jump to 2

TETRIS Bypass Ordering

- Analytically derived
  - Closed-form solution
  - No need for exhaustive search
- Near-optimal schedules
  - With 2% from schedules derived with exhaustive search

$$\min A_{\mathrm{DRAM}} = 2 \times N_{\mathrm{b}} N_{\mathrm{o}} S_{\mathrm{o}} \times t_{\mathrm{i}} + N_{\mathrm{b}} N_{\mathrm{i}} S_{\mathrm{i}} + N_{\mathrm{o}} N_{\mathrm{i}} S_{\mathrm{w}} \times t_{\mathrm{b}}$$

$$\text{s.t.} \begin{cases} \frac{N_{\mathrm{b}}}{t_{\mathrm{b}}} \times \frac{N_{\mathrm{i}}}{t_{\mathrm{i}}} \times S_{\mathrm{i}} \le S_{\mathrm{buf}} \\ 1 \le t_{\mathrm{b}} \le N_{\mathrm{b}}, \quad 1 \le t_{\mathrm{i}} \le N_{\mathrm{i}} \end{cases}$$

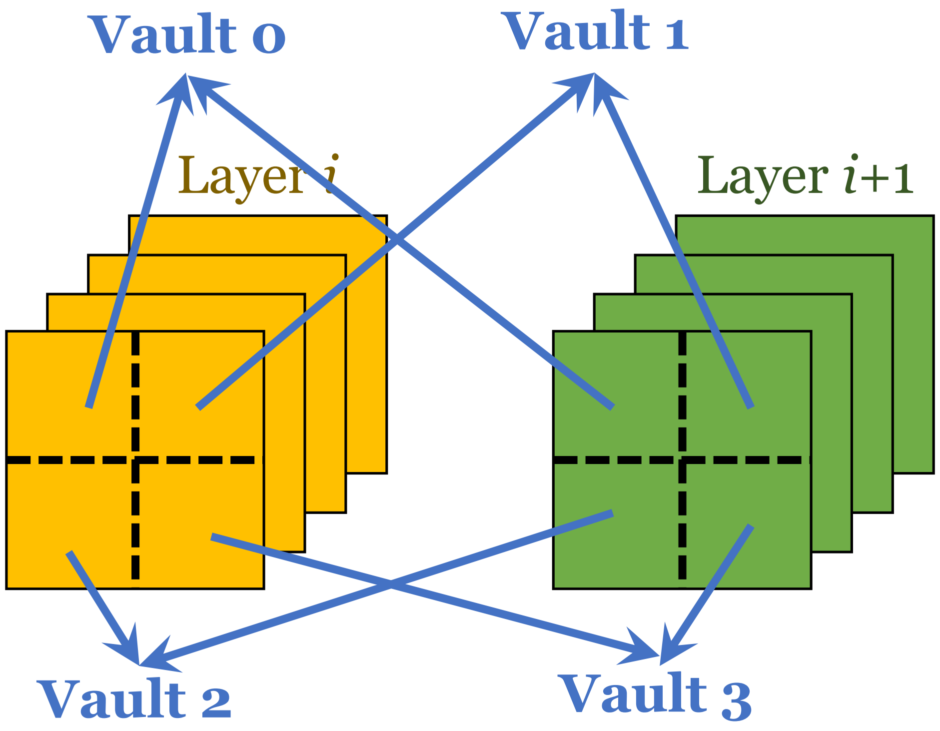
NN	Runtime Gap (w.r.t. optimal)	Energy Gap (w.r.t. optimal)
AlexNet	1.48 %	1.86 %
ZFNet	1.55 %	1.83 %
VGG16	0.16 %	0.20 %
VGG19	0.13 %	0.16 %
ResNet	2.91 %	0.78 %

NN Partitioning

- Process NN computations in parallel in all vaults

- Option 1: fmap partitioning
  - Divide a fmap into tiles
  - Each vault processes one tile

  - Minimum *remote accesses*

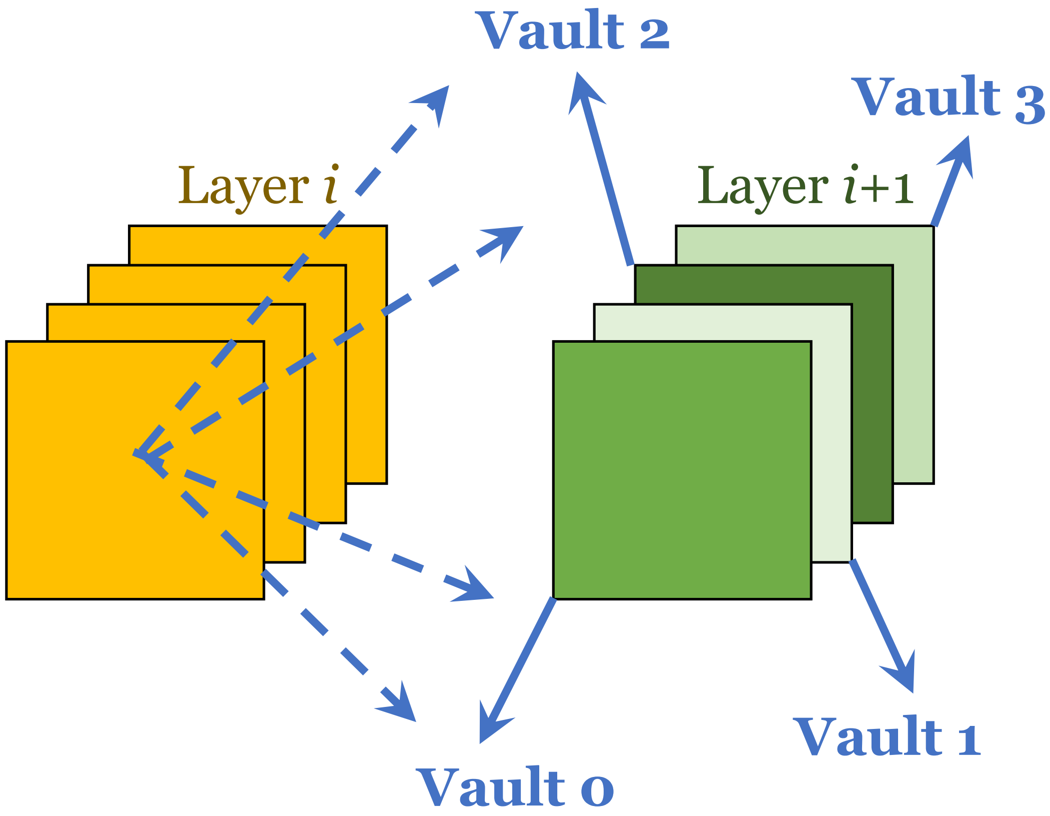


NN Partitioning

- Process NN computations in parallel in all vaults

- Option 2: output partitioning
  - Partition all ofmaps into groups
  - Each vault processes one group

  - Better filter weight reuse
  - Fewer *total memory accesses*



TETRIS Hybrid Partitioning

- Combine fmap partitioning and output partitioning
  - Balance between minimizing *remote accesses* and *total DRAM accesses*
  - Total energy = NoC energy + DRAM energy

- Difficulties
  - Design space exponential to # layers
    → *Greedy algorithm reduces to be linear to # layers*
  - Complex dataflow scheduling to determine total DRAM accesses
    → *Bypass ordering to quickly estimate total DRAM accesses*

TETRIS Evaluation

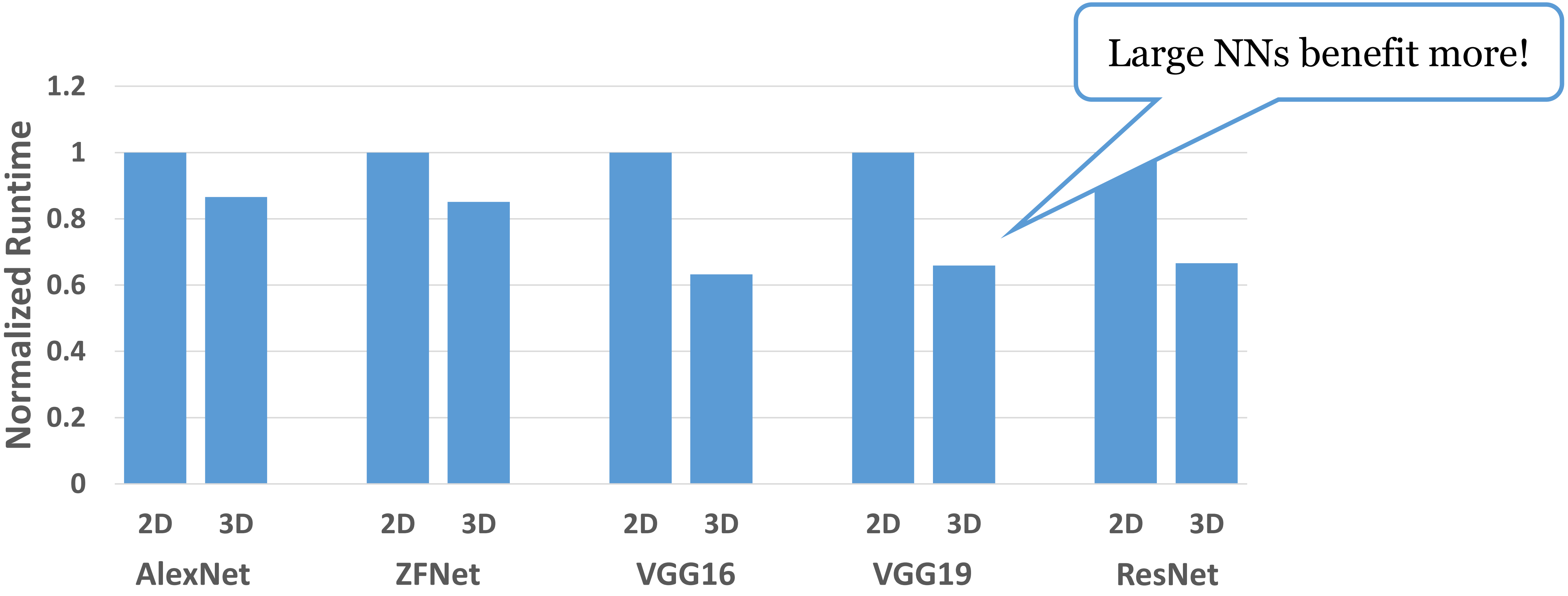
Methodology

- State-of-the-art NNs
  - AlexNet, ZFNet, VGG16, VGG19, ResNet
  - 100—300 MB total memory footprint for each NN
  - Up to 152 layers in ResNet

- 2D and 3D accelerators with ≥1 NN engines
  - 2D engine: 16 x 16 PEs, 576 kB buffer, 1 LPDDR3 channel
    - 8.5 mm^2, 51.2 Gops/sec
    - Bandwidth-constrained
  - 3D engine: 14 x 14 PEs, 133 kB buffer, 1 HMC vault
    - 3.5 mm^2, 39.2 Gops/sec
    - Area-constrained

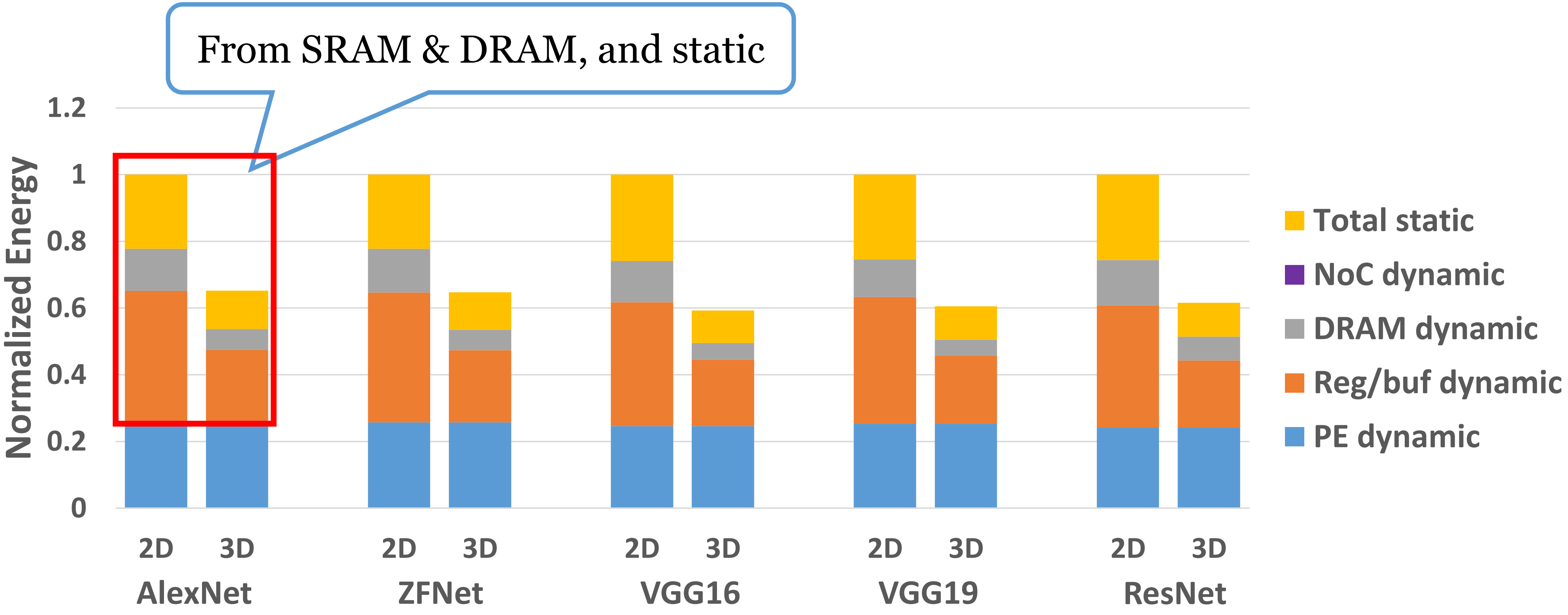
Single-engine Comparison

- Up to 37% performance improvement with TETRIS
  - Due to higher bandwidth despite smaller PE array



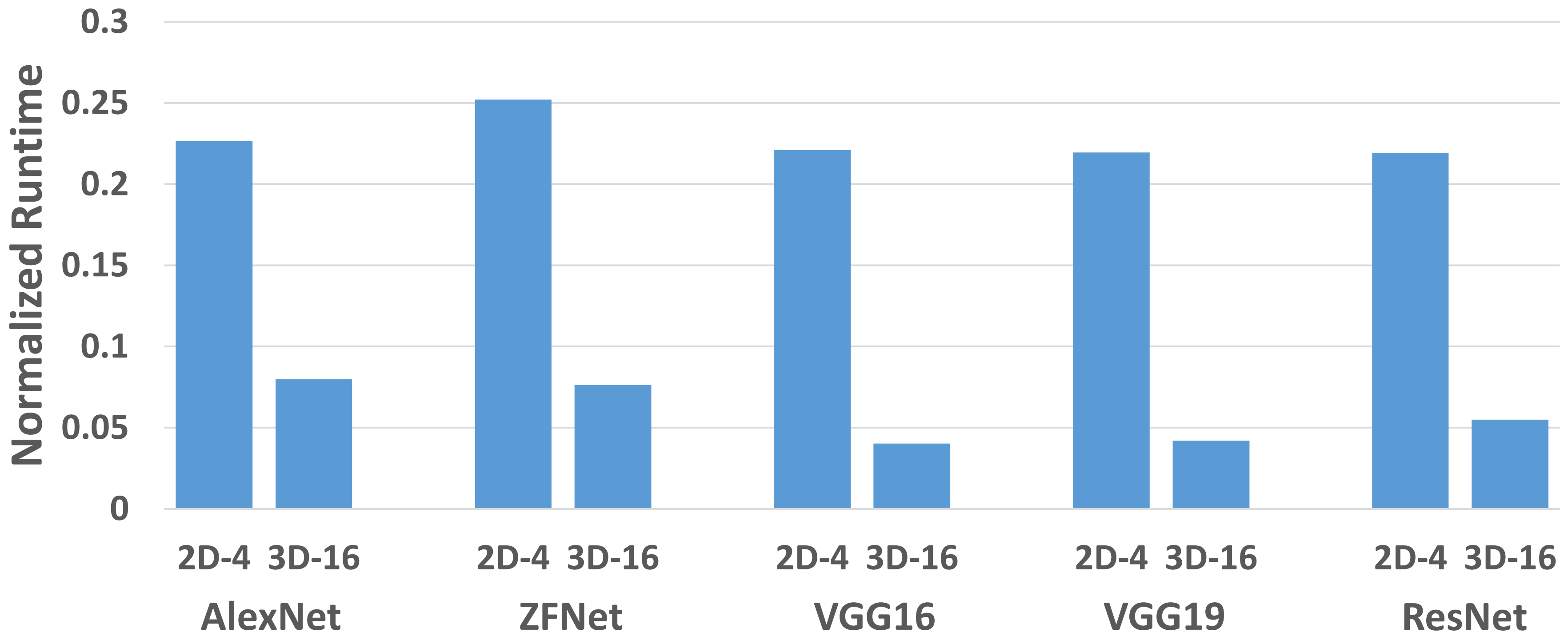
Single-engine Comparison

- 35–40% energy reduction with TETRIS
  - Smaller on-chip buffer, better scheduling



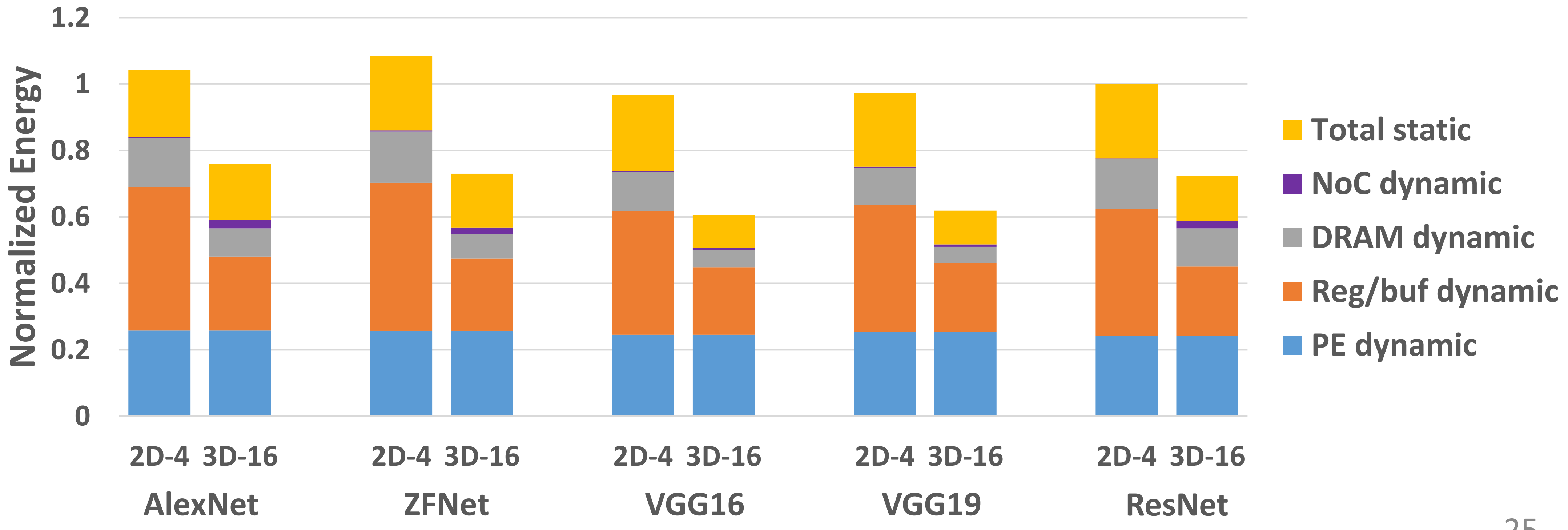
Multi-engine Comparison

- 4 2D engines: 34 mm², pin constrained (4 LPDDR3 channels)
- 16 3D engines: 56 mm², area constrained (16 HMC vaults)
- 4.1x performance gain → *2x compute density*



Multi-Engine Comparison

- 1.5x lower energy
  - 1.2x from better scheduling and partitioning
- *4x computation* only costs *2.7x power*



TETRIS Summary

- A scalable and efficient NN accelerator using 3D memory
  - 4.1x performance and 1.5x energy benefits over 2D baseline

- Hardware features
  - PE/buffer area rebalancing
  - In-memory accumulation
- Software features
  - Analytical dataflow scheduling
  - Hybrid partitioning

- Scheduling exploration tool
  - https://github.com/stanford-mast/nn_dataflow

Thanks!

Questions?