CS5120 VLSI System Design, Spring 2025

DNN Mapping Part 1: Domain-Specific Architectures

黃稚存

Chih-Tsun Huang

cthuang@cs.nthu.edu.tw

國立清華大學
NATIONAL TSING HUA UNIVERSITY

資訊工程學系
Computer Science

Lecture 06

聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容 (例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

Outline

- Source of Inefficiency with General Purpose CPUs
- Core Optimization
  - Optimization of Instruction and Operation Decoding
  - Optimization of Datapath Optimization
  - Optimization of Memory Organization

Domain-Specific Architectures (DSAs)

- In early 2000s, microprocessors moved to manycore architecture
  - Multiple general-purpose cores per die
  - To improve energy efficiency for parallel workloads
- Now, great interest in more specialized architectures to further improve energy efficiency on certain workloads
  - At system-level, usually a combination of

    host processor + co-processor(s)

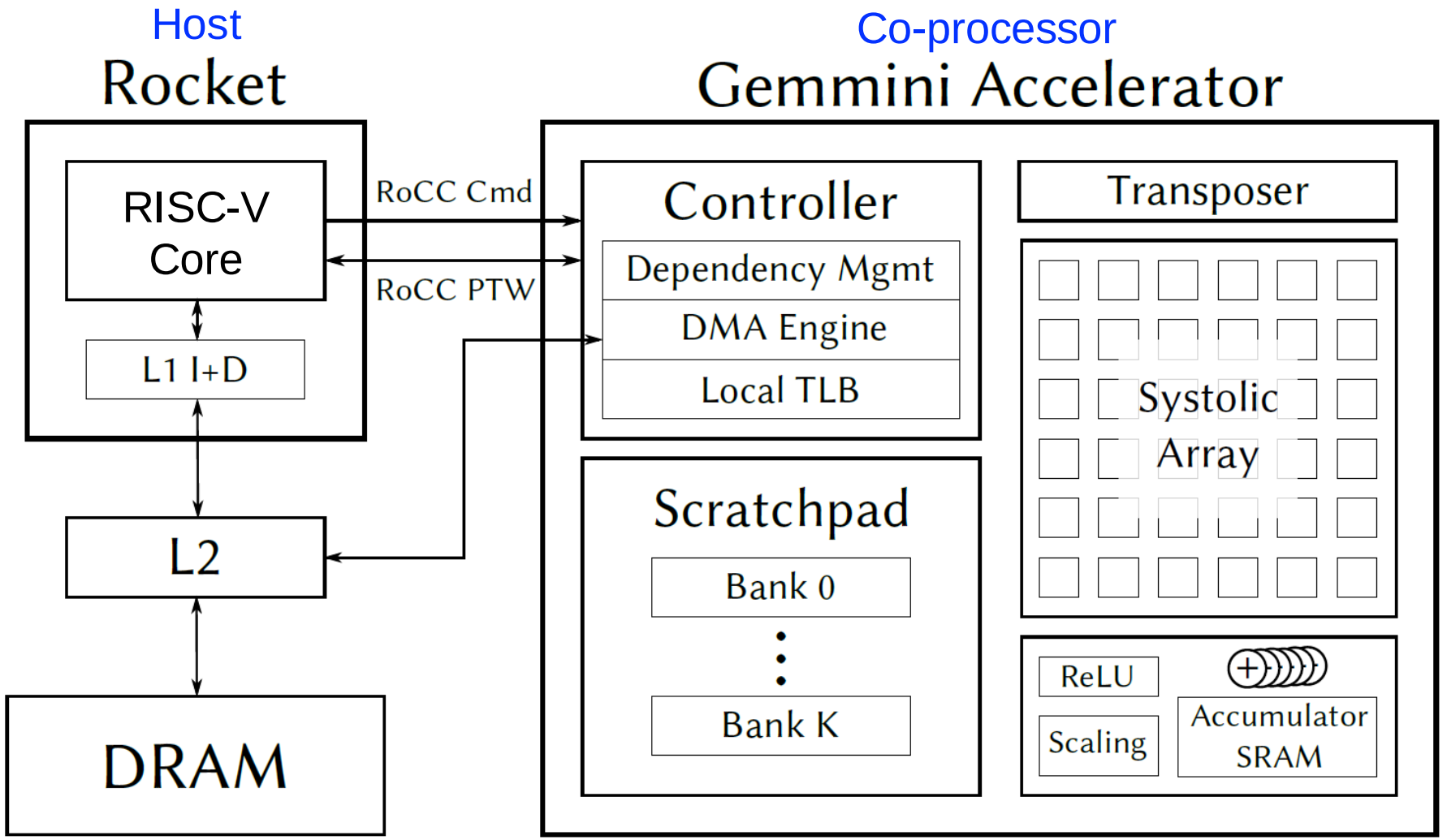
    (General Purpose Microcontroller) (Domain-Specific Accelerator(s))

Guidelines for DSAs

- Use dedicated memories to minimize data movement
  - E.g., a two-way set associative cache uses 2.5X energy as an equivalent software-controlled scratchpad memory
- Invest resources into more arithmetic units or bigger memories
- Use the easiest form of parallelism that matches the domain
  - SIMD, VLIW
- Reduce data size and type to the simplest needed for the domain
  - Applications in many domains are typically memory-bound
  - Narrower data types
    - Increase the effective memory bandwidth and on-chip memory utilization
    - Pack more arithmetic units into the same chip area
- Use a domain-specific programming language
  - Porting applications to your DSA becomes much more feasible

Ex: Gemmini Accelerator Architecture

- Gemmini Project: Systolic-Array Matrix Multiplication Accelerator Generator



*DMA: Direct Memory Access

https://github.com/ucb-bar/gemmini

Optimization of Instruction and Operation Fusion

Instruction Set and Control

- Coarse-Grained, Domain-Specific Instructions
  - Instead of fine-grained general-purpose instructions
    - E.g., RISC-V instructions: `add`, `ld`, `beq`
  - Key instructions: data movement and compute
- Ex: Gemmini ISAs: https://github.com/ucb-bar/gemmini
  - Configurations:
    - config_ex: configures the execute pipeline
    - config_mvin: configures the load pipeline
    - config_mout: configures the store pipeline
    - flush: flush the TLB
  - Data movement:
    - mvin: move data from L2/DRAM to scratchpad
    - mvout: move data from scratchpad to L2/DRAM
  - Compute:
    - matmul.preload: preload weights for weight stationary
    - matmul.compute.preloaded: compute using preloaded values
    - matmul.compute.accumulated: compute using existing values

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Coarse-Grained Operation Optimization

- Operation fusion
  - Merging multiple operators together
  - Also commonly used in software optimization
- Ex:
  - CRP (convolution-ReLU-pooling)
  - Operations fused in TensorFlow graph optimization
    - Conv2D + BiasAdd + <Activation>
    - Conv2D + FusedBatchNorm + <Activation>
    - Conv2D + Squeeze + Activation
    - MatMul + BiasAdd + Activation

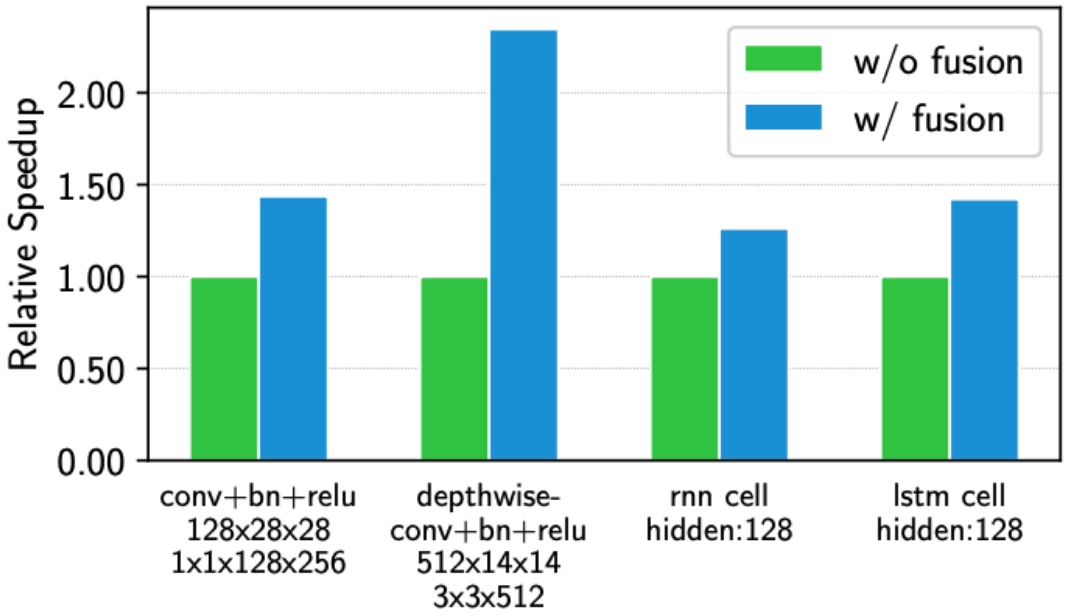
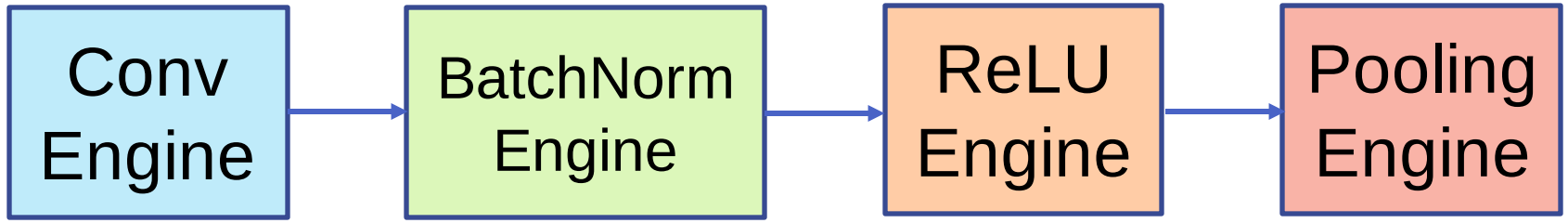


Figure 4: Performance comparison between fused and non-fused operations. TVM generates both operations. Tested on NVIDIA Titan X.

TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

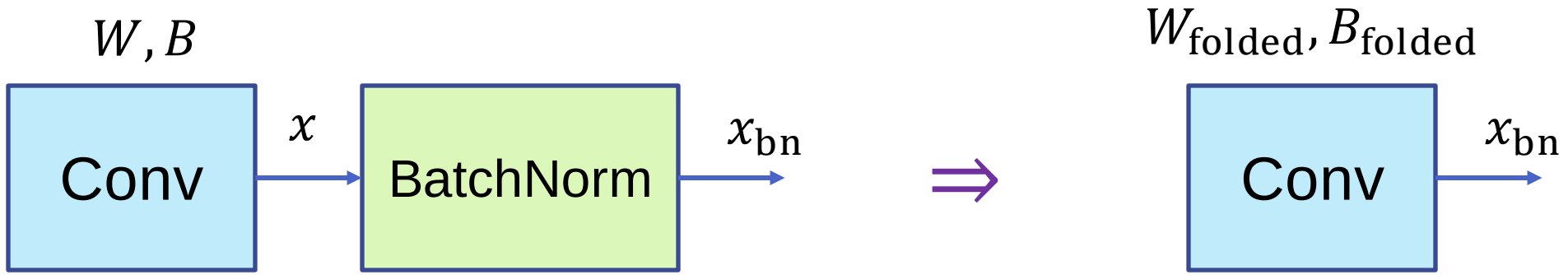
Operation Fusion



- Or sometimes, layer fusion
- Merge multiple operators into one contiguous calculation
  - Arithmetic or structural optimization
  - Reduce # memory accesses
    ➔ reduce energy, storage
  - Streaming engines
    - Pipeline with data forwarding

BatchNorm Folding (Fusion)

- Manipulating the arithmetic algorithm for further optimization
- Simplify the compute for inference



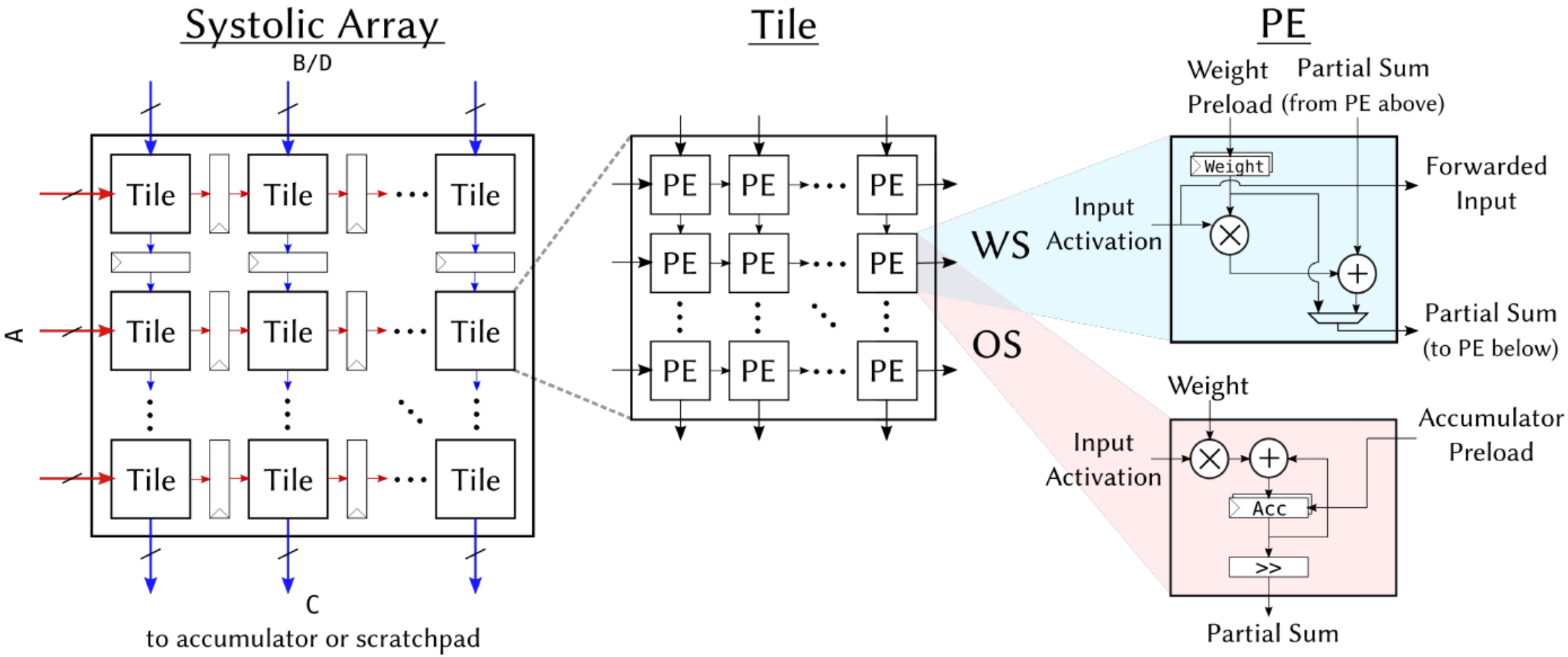
$$x_{\text{bn}} = \gamma\left(\frac{x-\mu}{\sigma}\right) + \beta$$

$$W_{\text{folded}} = \frac{\gamma W}{\sigma}$$

$$B_{\text{folded}} = \beta - \frac{\gamma\mu}{\sigma}$$

Optimization of Datapath

Ex: Gemmini Datapath of Systolic Array



https://github.com/ucb-bar/gemmini

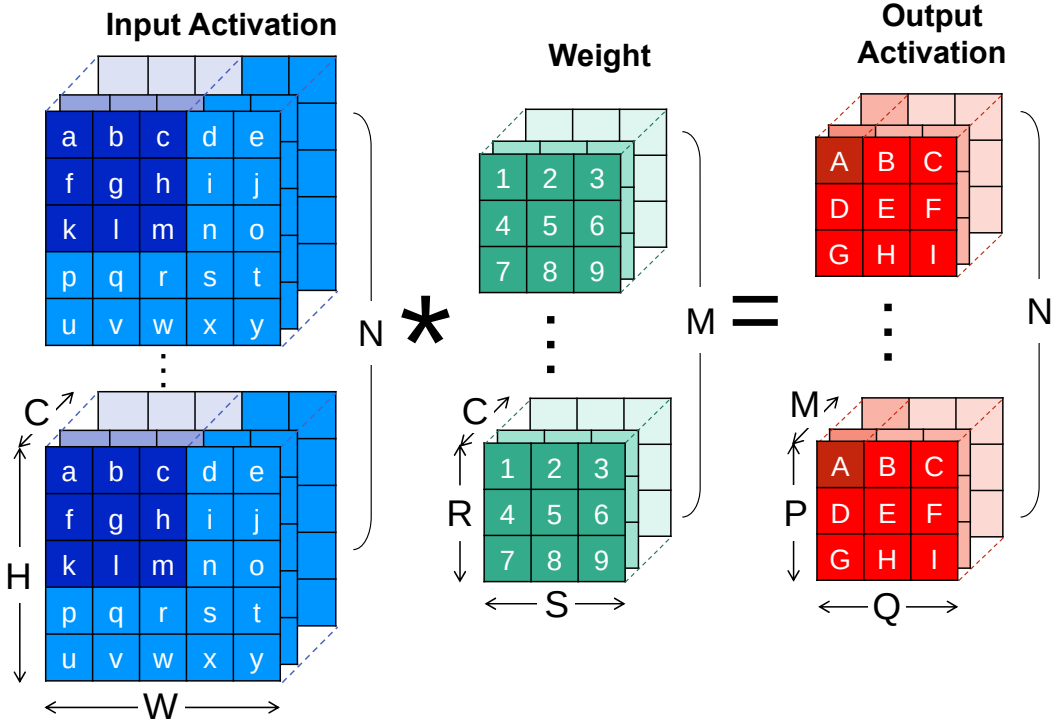
Recap: Dataflow

- Dataflow refers to how data is processed within the hardware architecture
  - Determines the path that data moves and how it is transformed and manipulated through the system
  - Defines the execution order of the DNN operations in hardware
    - Computation order
    - Data movement order
- Loop nest is a compact way to describe the dataflow supported in hardware
  - `for` (temporal for): describes the temporal execution order
  - `parallel_for` (spatial for): describes the parallel execution

Recap: Convolution Layer with 7 Nested Loops

Input Feature Map (ifmap)

Output Feature Map (ofmap)



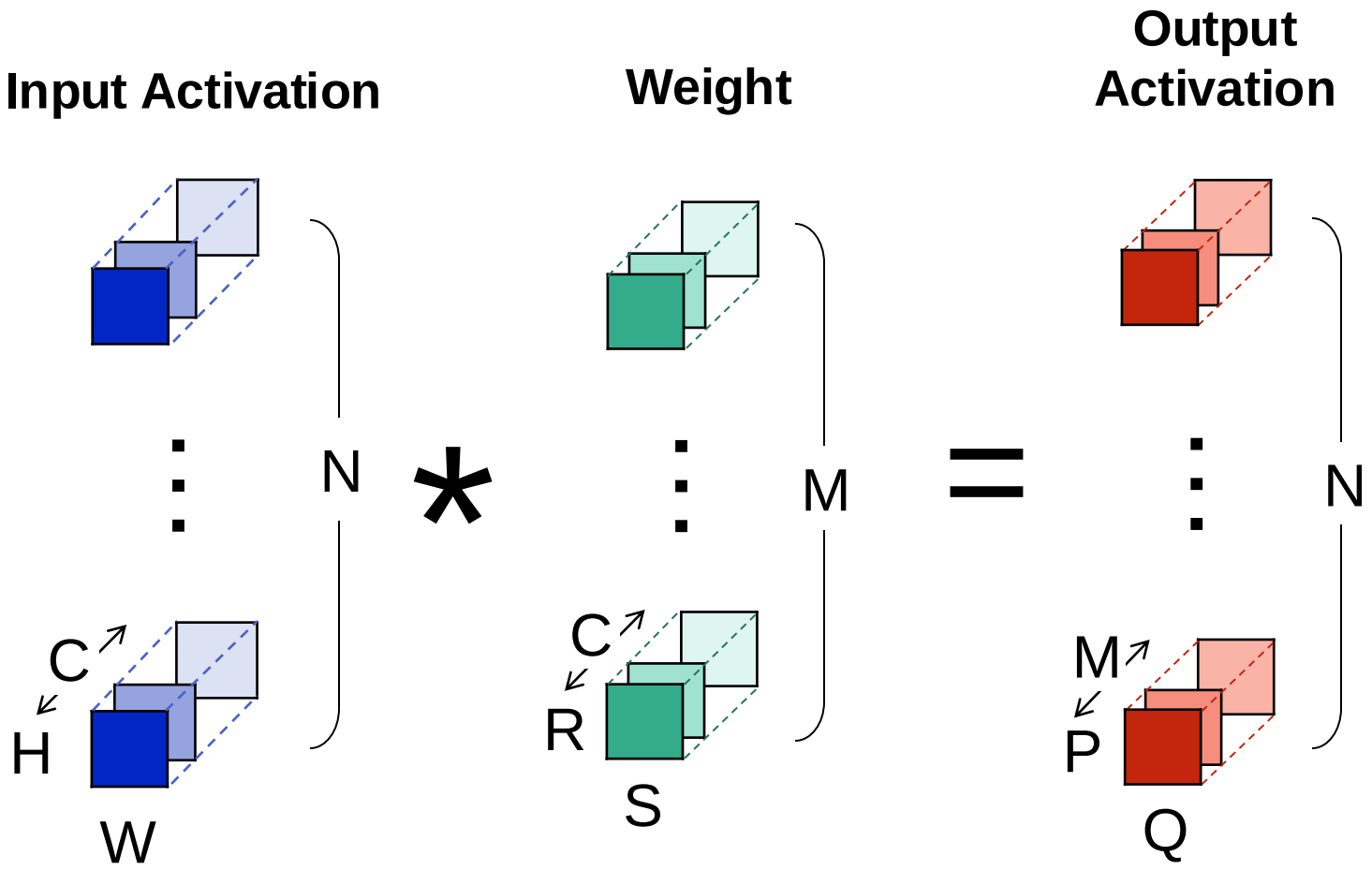

```
for (n=0; n<N; n++) {
  for (m=0; m<M; m++) {
    for (p=0; p<P; p++) {
      for (q=0; q<Q; q++) {
        OA[n][m][p][q] = 0;
        for (r=0; r<R; r++) {
          for (s=0; s<S; s++) {
            for (c=0; c<C; c++) {
              h = p * U – Pad + r;
              w = q * U – Pad + s;
              OA[n][m][p][q] +=
                IA[n][c][h][w] * W[m][c][r][s];
        }}}
        OA[n][m][p][q] = Activation(OA[n][m][p][q]);
}}}}
```

For each output activation

Convolution window

Partial Sum (psum)

Recap: Fully-Connected Layer



H = 1
W = 1
R = 1
S = 1
P = 1
Q = 1

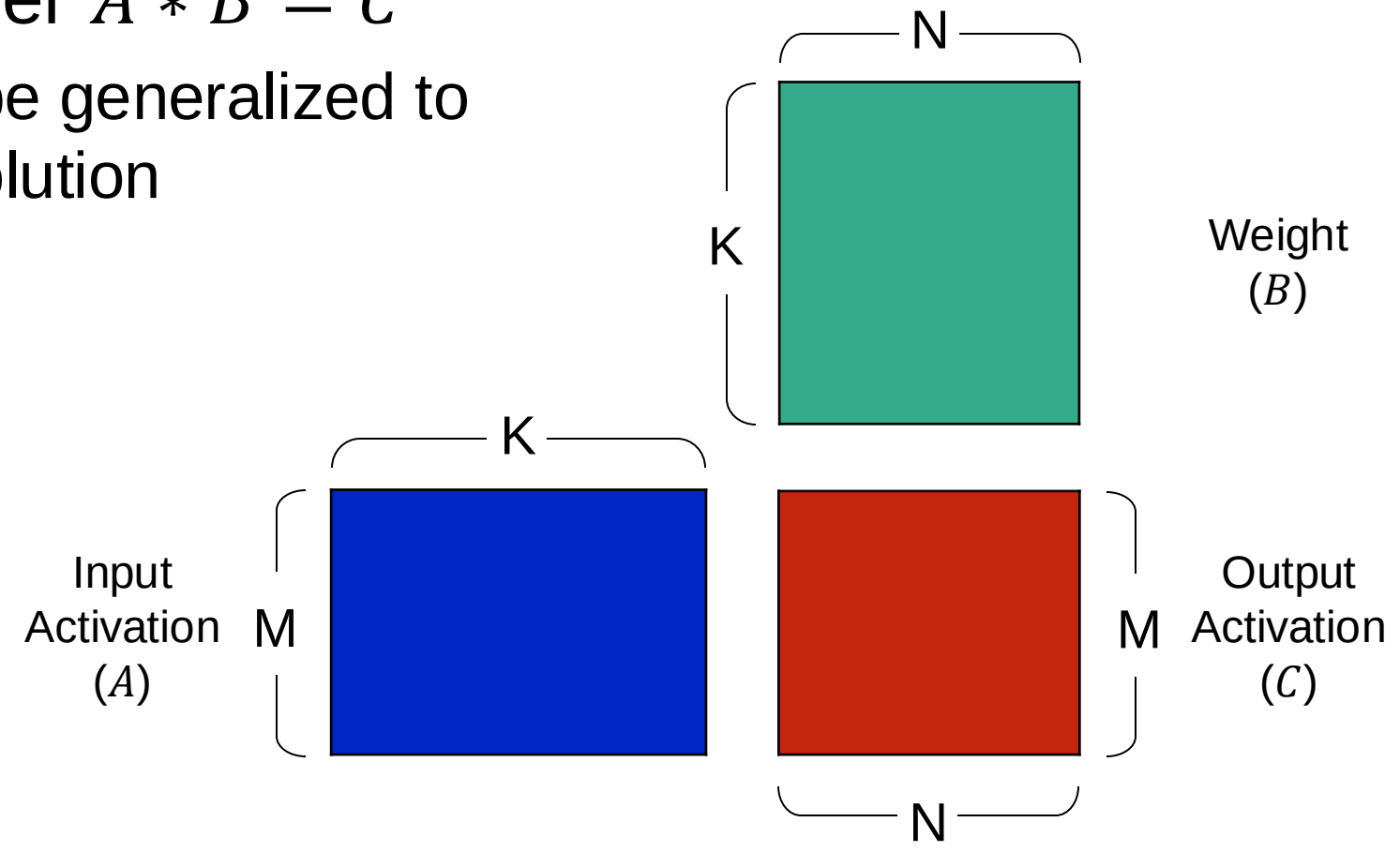
U (Stride)= 1

Pad (Padding) = 0

C: # of Input Channels
M: # of Output Channels
N: Batch size

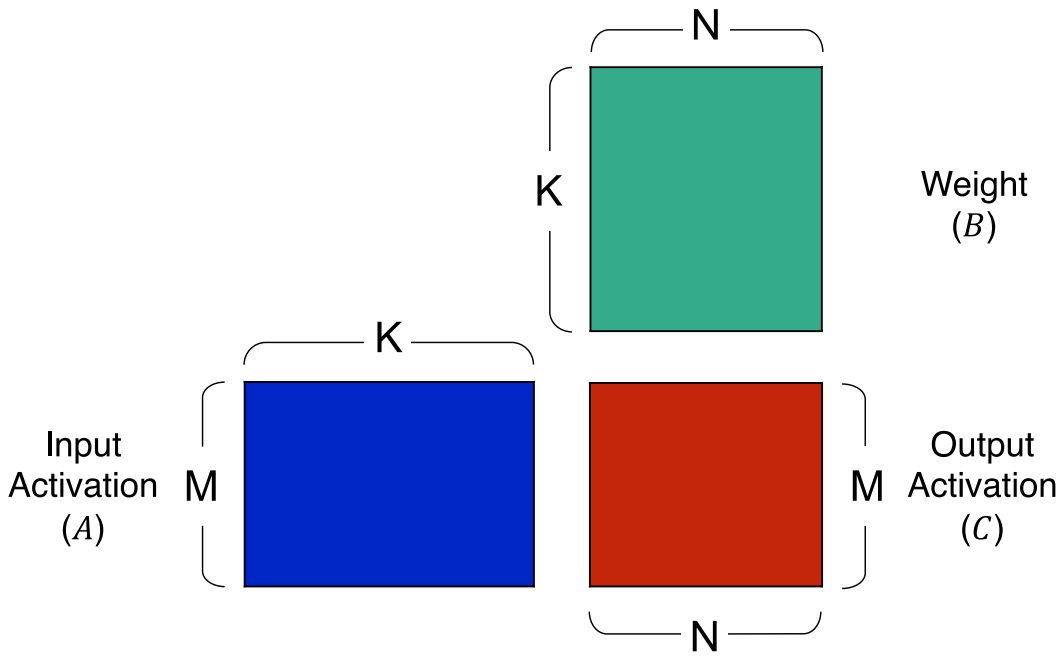
Matrix Multiplication as Case Study

- Consider $A * B = C$
  - Can be generalized to convolution



[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Loop Nest for Matrix Multiplication



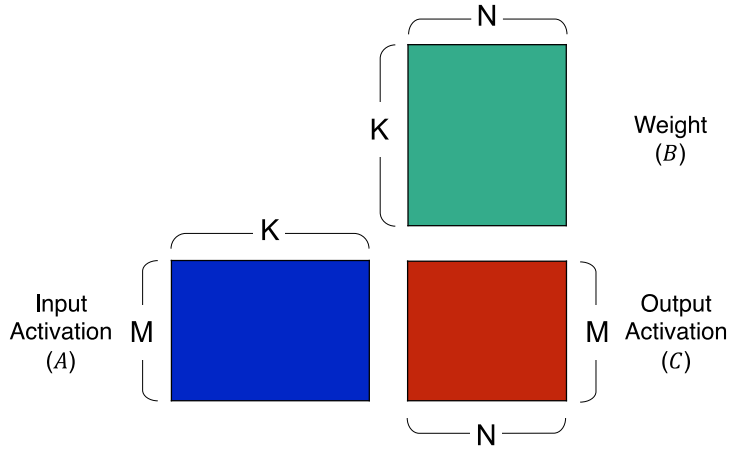

```
for (m=0; m<M; m++) {
  for (n=0; n<N; n++) {
    OA[n,m] = 0;
    for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

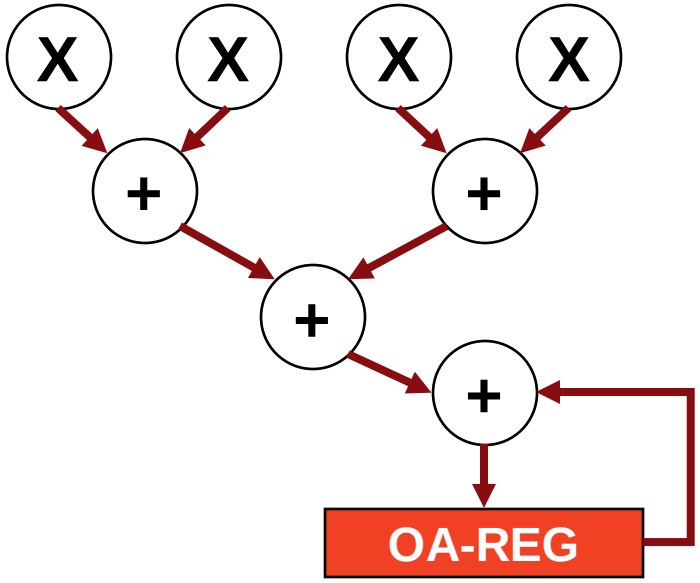
For each output activation

Reduction

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Datapath Optimization for Spatial-K
Type 1: Adder-tree Accumulation (Reduction Tree)



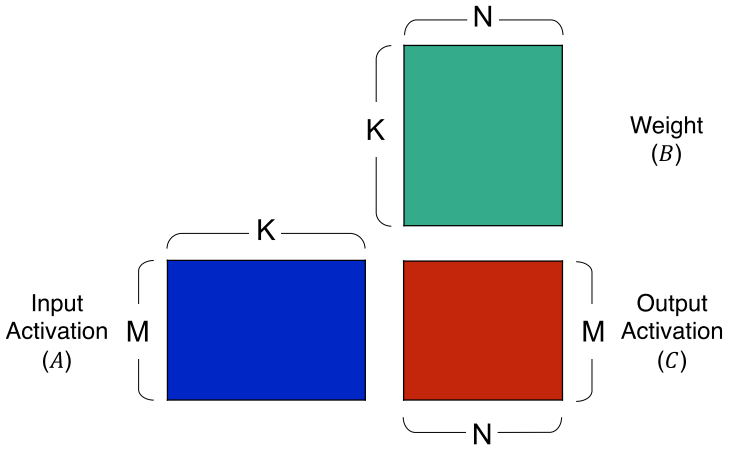


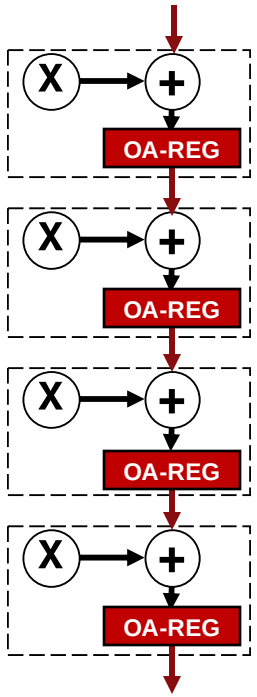

```
for (m=0; m<M; m++) {
  for (n=0; n<N; n++) {
    OA[n,m] = 0;
    parallel_for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Case examples: NVDLA, DianNao
- Typical width: 8-64
- Applicable to any accumulation dimensions
  - E.g., R, S, C in convolution

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Datapath Optimization for Spatial-K
Type 2: Systolic Accumulation



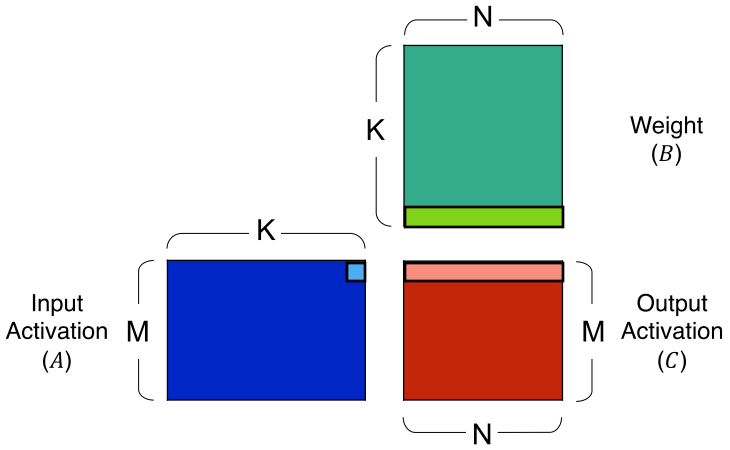


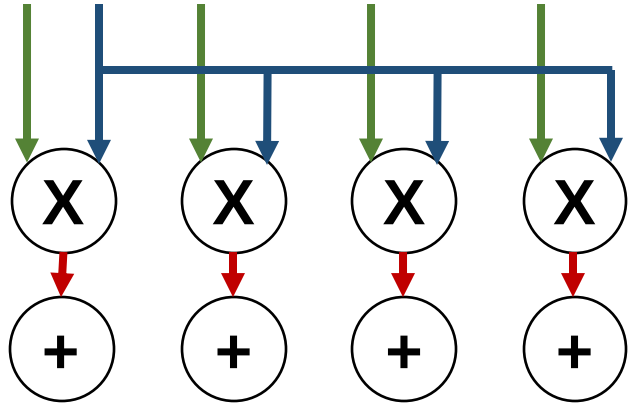

```
for (m=0; m<M; m++) {
  for (n=0; n<N; n++) {
    OA[n,m] = 0;
    parallel_for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Case examples: TPU, Gemmini
- Typical width: 8-256
- Applicable to any accumulation dimensions
  - E.g., R, S, C in convolution

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Datapath Optimization for Spatial-N
Type 1: Direct-wire multicast



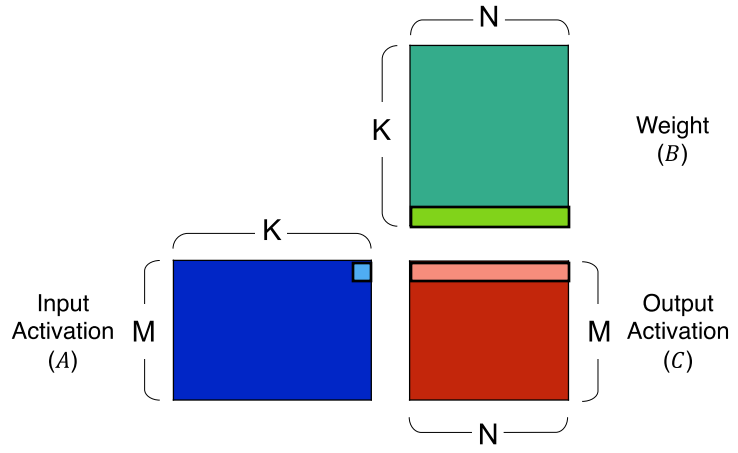


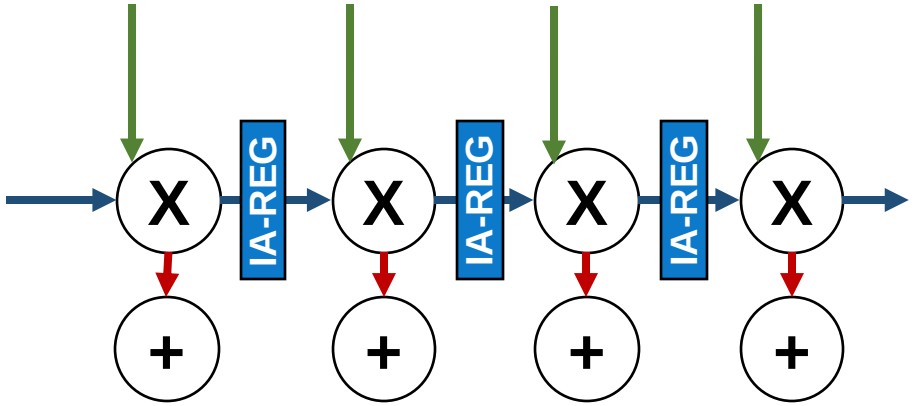

```
for (m=0; m<M; m++) {
  parallel_for (n=0; n<N; n++) {
    OA[n,m] = 0;
    for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Case examples: NVDLA, DianNao
- Typical width: 8-16
- Applicable to any non-accumulation dimensions

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Datapath Optimization for Spatial-N Type 2: Systolic Multicast



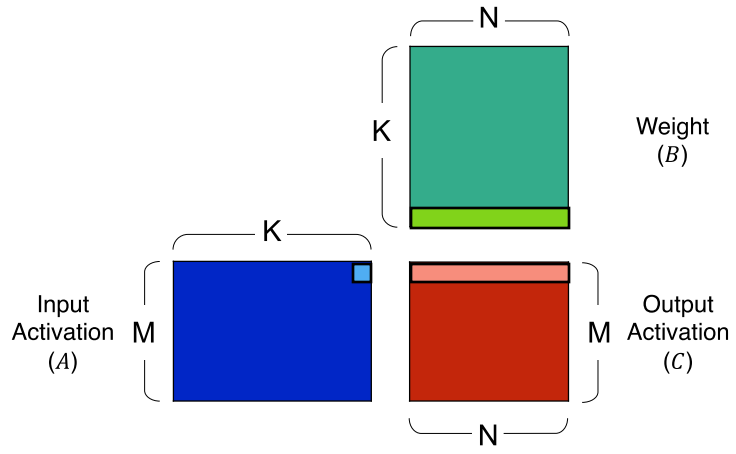


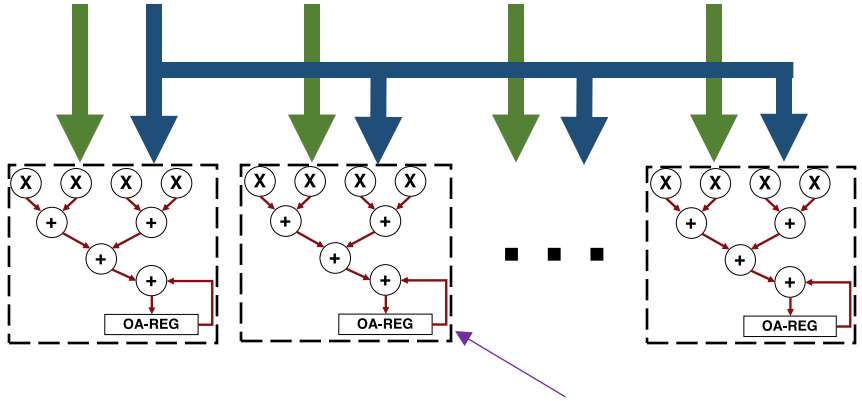

```
for (m=0; m<M; m++) {
  parallel_for (n=0; n<N; n++) {
    OA[n,m] = 0;
    for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Case examples: TPU, Gemmini
- Typical width: 8-256
- Applicable to any non-accumulation dimensions

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Mixed Datapath Optimization: NVDLA





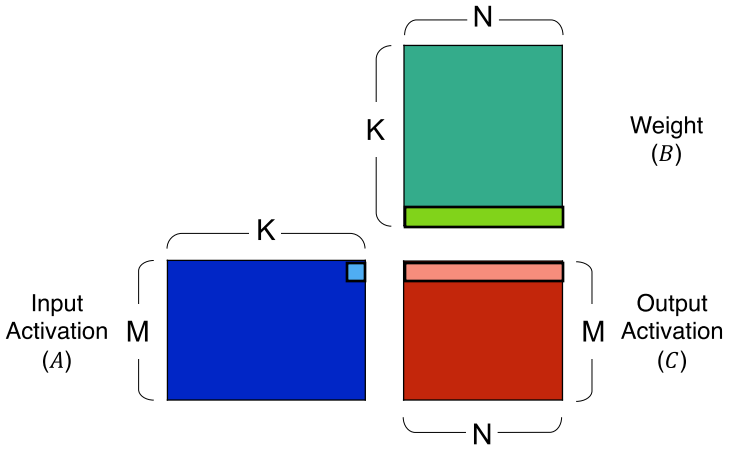
MAC Cell, Tile, or PE

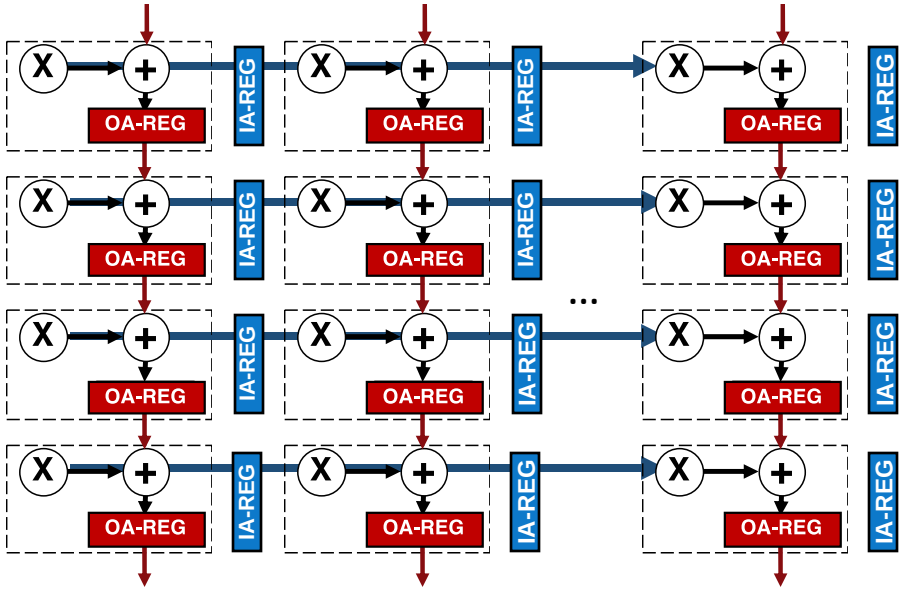
```
for (m=0; m<M; m++) {
  parallel_for (n=0; n<N; n++) {
    OA[n,m] = 0;
    parallel_for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Adder-tree accumulation (reduction tree)
- Direct-wire multicast

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Mixed Datapath Optimization: TPU






```
for (m=0; m<M; m++) {
  parallel_for (n=0; n<N; n++) {
    OA[n,m] = 0;
    parallel_for (k=0; k<K; k++) {
      OA[n,m] +=  IA[m, k] * W[k, n];
    }
    OA[n,m] = Activation(OA[n,m]);
  }
}
```

- Systolic accumulation
- Systolic multicast

→ Area vs. scalability

→ Latency vs. pipeline throughput

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Summary for Datapath Optimization

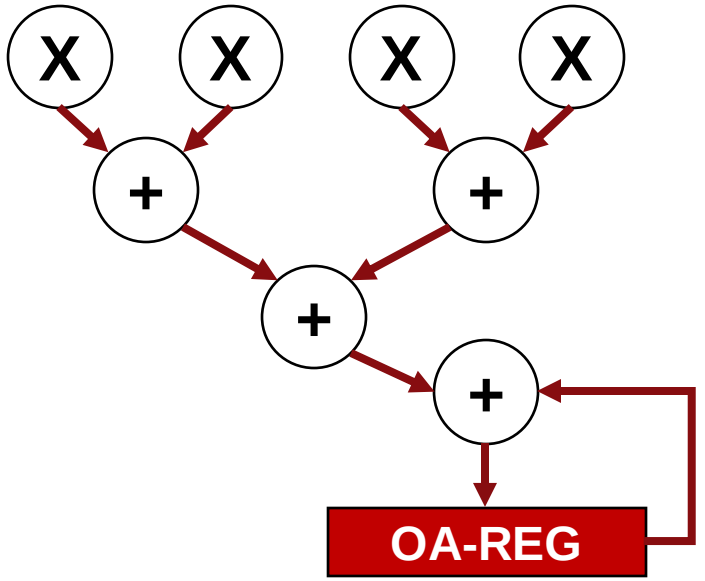
- `parallel_for` to explore parallelism
  - Spatial-K: accumulation dimension
    - Adder (reduction) tree and dot product
    - Systolic accumulation
  - Spatial-M/N: data reuse dimension
    - Direct-wire multicast
    - Systolic multicast
- State-of-the-art accelerators typically use a combination of both at different levels of the hierarchy

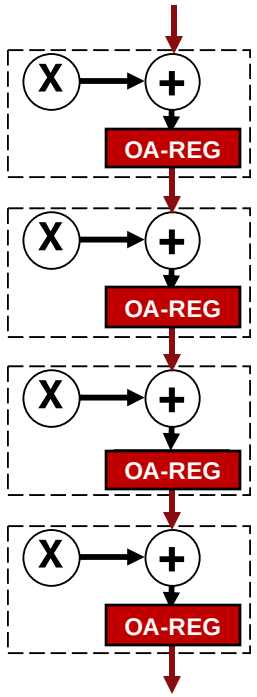
[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Optimization of Memory Organization

Memory Optimization 1: Short-lived Intermediate Results

- Consume the intermediate results directly
  - Instead of accessing SRAM or a shared, large register file
- Example: adder tree and systolic accumulation

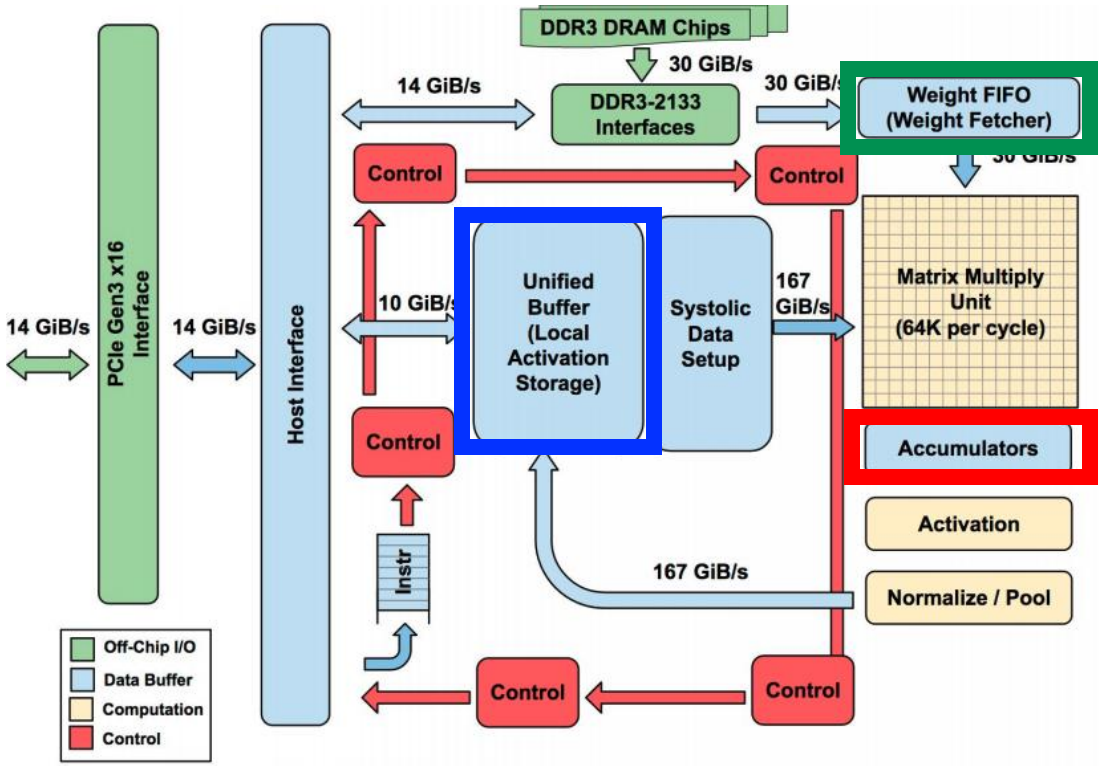




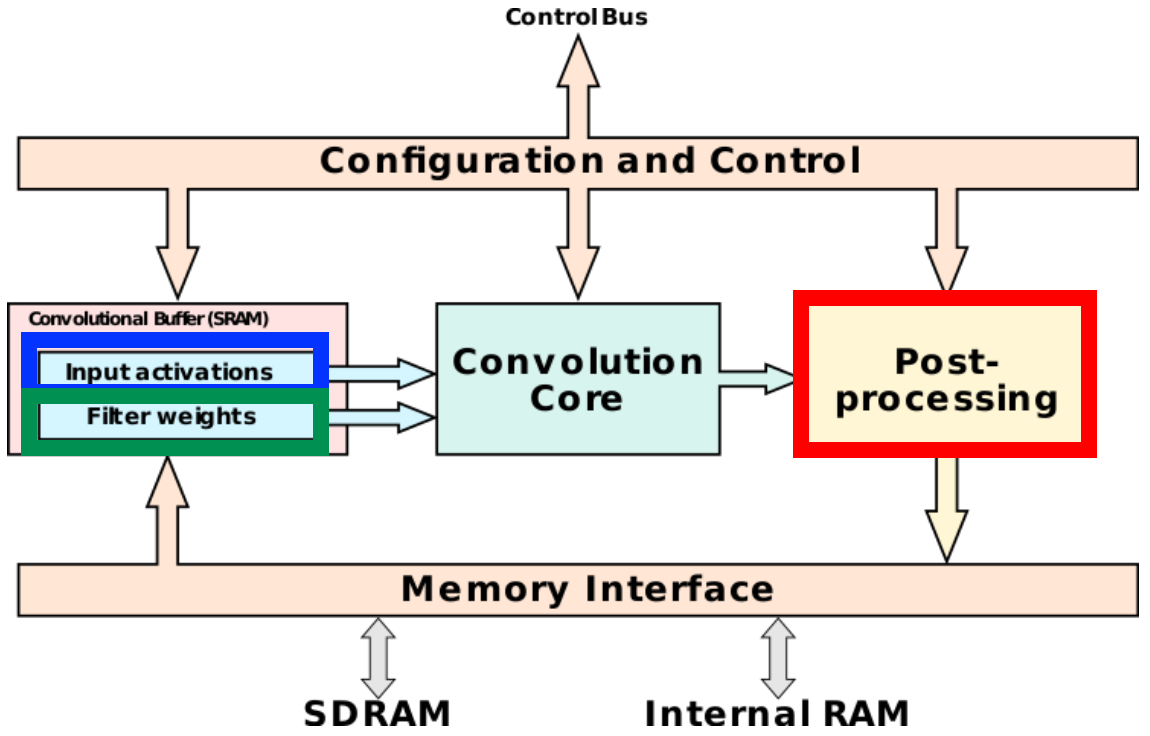
[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Memory Optimization 2: Application-Specific Storage Size/BW

- Dedicated storage (register/buffer) for each operand



TPU

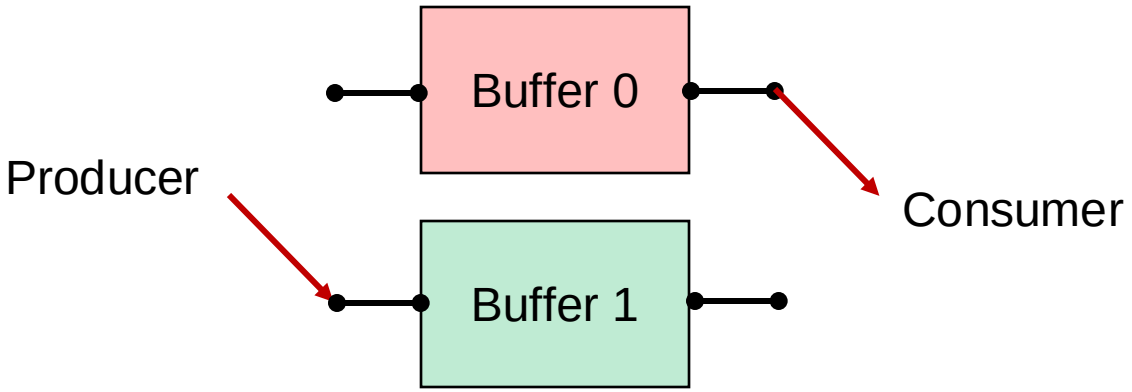


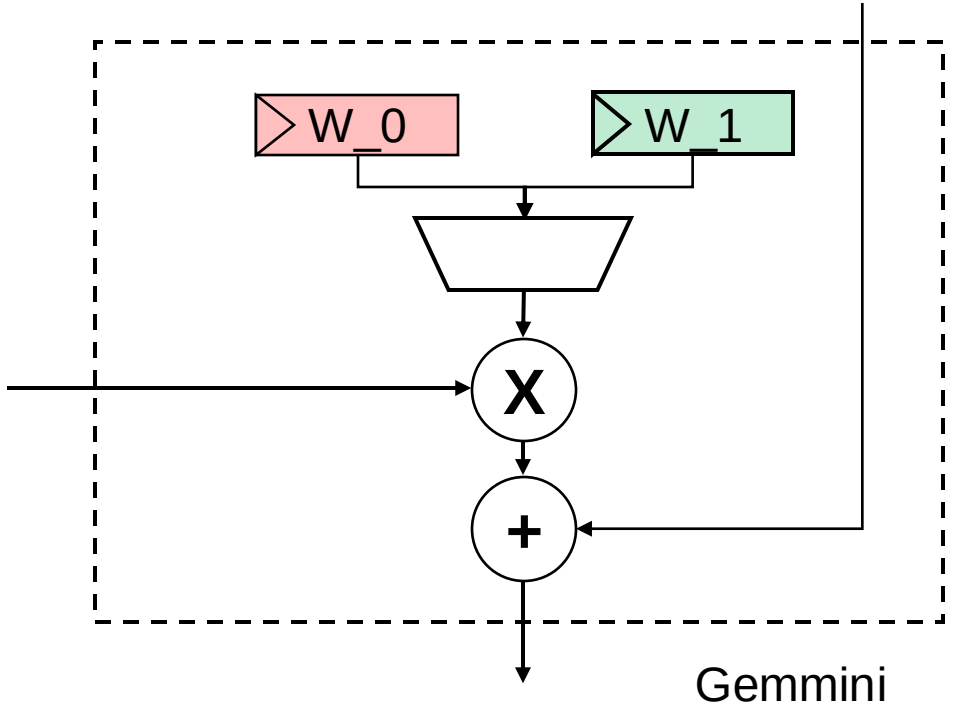
NVDLA

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Memory Optimization 3: Application-Specific Data Delivery

- Double-buffering
  - Goal: Overlap compute and communication by doubling the SRAM sizes.
  - Producer writes to a buffer while Consumer reads from the other buffer.

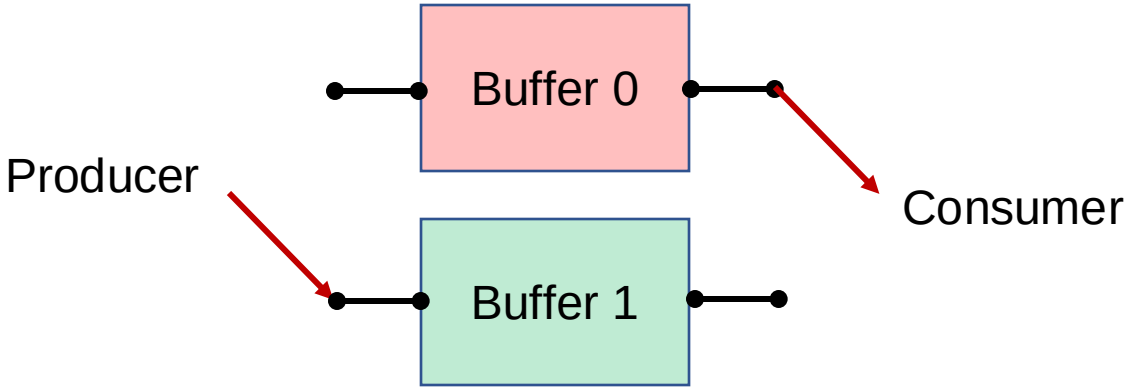


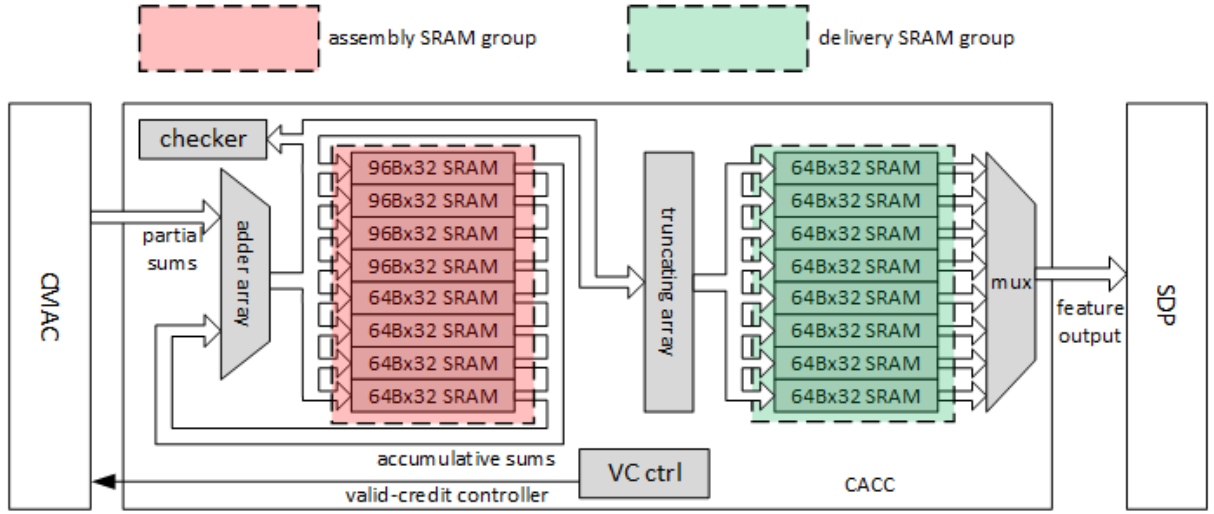


[Source: Prof. Sophia Shao, EE290-2, Berkeley]

Memory Optimization 3: Application-Specific Data Delivery

- Double-buffering
  - Case example: NVDLA





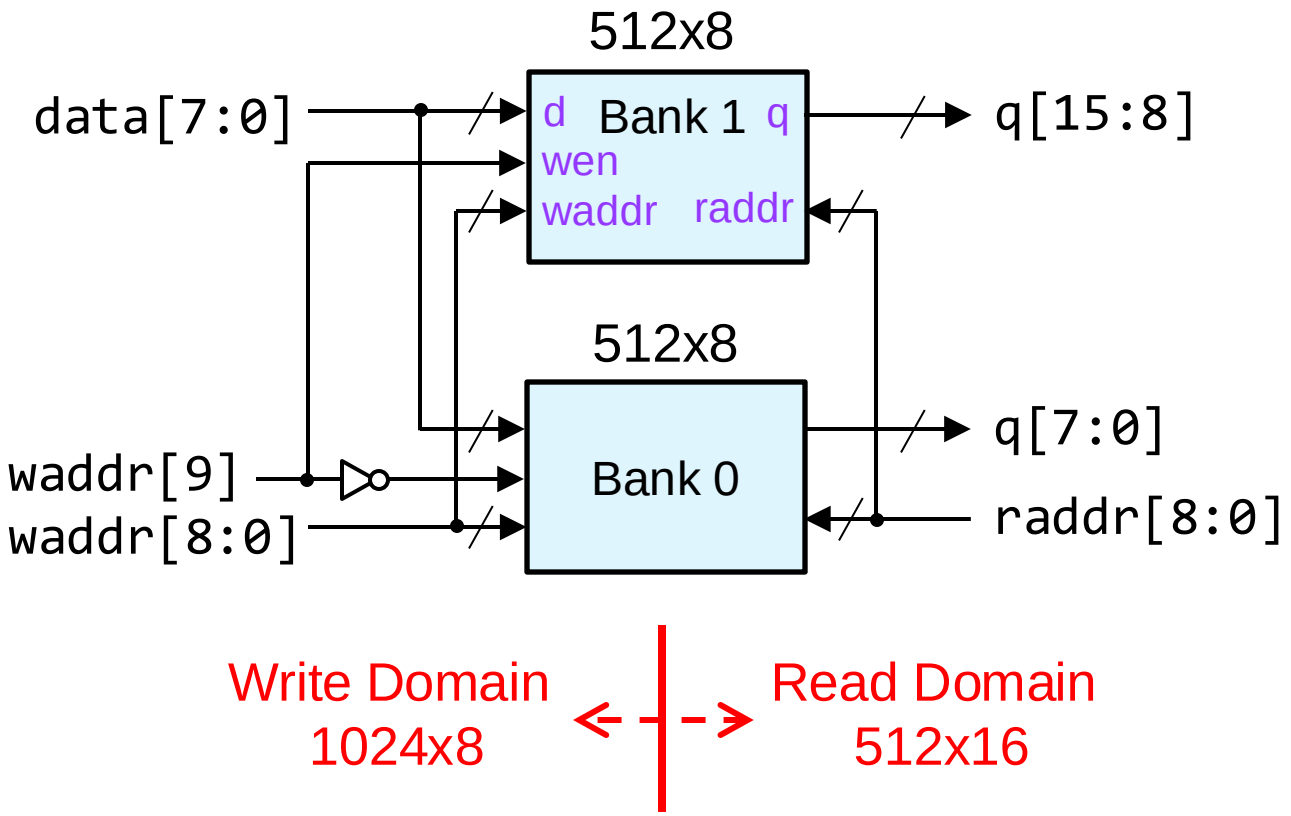
NVDLA

[Source: Prof. Sophia Shao, EE290-2, Berkeley]

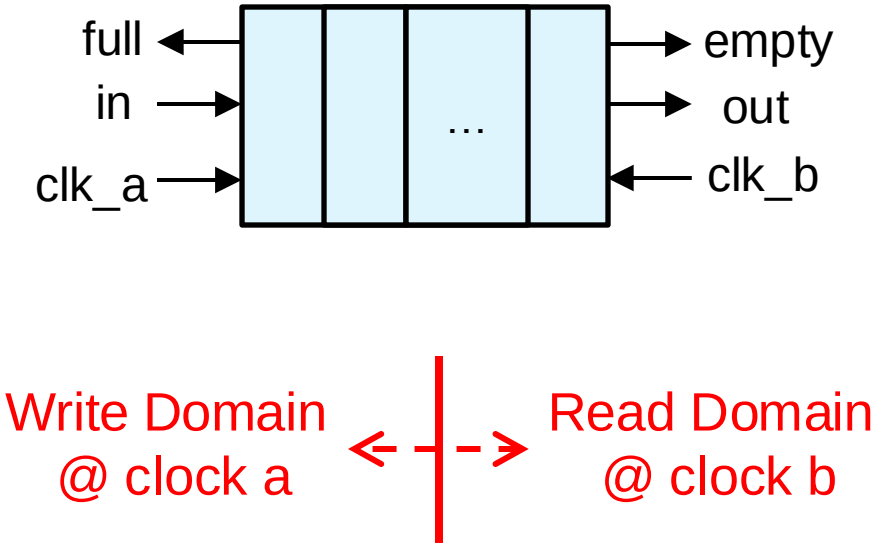
Memory Optimization 4: Application-Specific Data Delivery

- Data-width and clock-rate adoption

Data-width Adoption with Multi-bank Memory



Asynchronous FIFO for Clock-rate adoption



Summary for Memory Optimization

- Consume short-lived intermediate results directly
  - E.g., adder tree for partial sums
- Application-specific size and bandwidth for data storage
  - E.g., dedicated weight, input, output buffers
  - Scratchpad memory explicitly controlled by user HW or SW
- Application-specific data delivering network
  - E.g., double-buffering
  - E.g., data-width and clock-rate adoption