http://inst.eecs.berkeley.edu/~ee290-2/sp21/

Hardware for Machine Learning

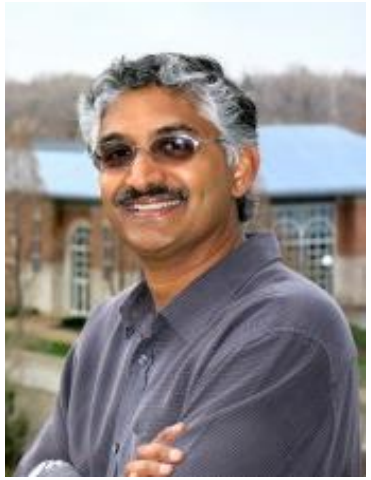
Lecture 11: Mapping

Sophia Shao

The LLVM Compiler Infrastructure (http://llvm.org/)

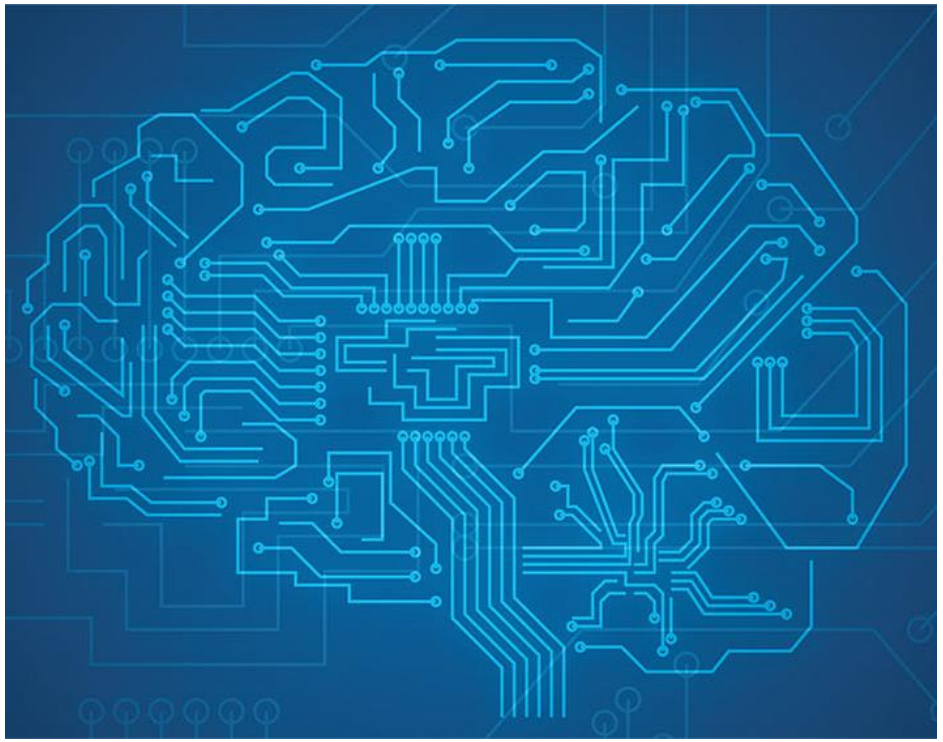
The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

LLVM began as a research project at the University of Illinois, with the goal of providing a modern, SSA-based compilation strategy capable of supporting both static and dynamic compilation of arbitrary programming languages. Since then, LLVM has grown to be an umbrella project consisting of a number of subprojects, many of which are being used in production by a wide variety of commercial and open source projects as well as being widely used in academic research. Code in the LLVM project is licensed under the "Apache 2.0 License with LLVM exceptions"



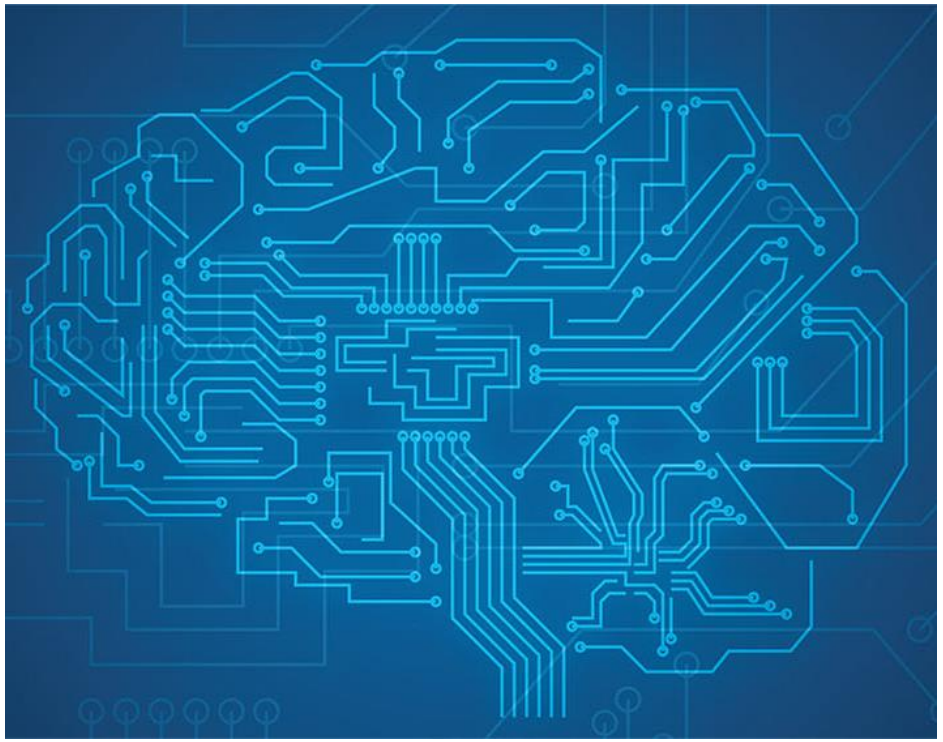
Review

- Core computation in DNN
- Execution order of the core computation
- Hardware realization of the core computation
  - Overview: source of inefficiency of general-purpose processing
  - Core optimizations:
    - Inst. decoding logic: coarse-grained instructions and operation fusion
    - Datapath: spatial-K and spatial-N/M
    - Memory: short-lived value, app-specific storage and data delivery
  - TPU Example (Lab2)
- This lecture: mapping DNNs to hardware



Mapping

- DNN Mapping Problem:
  - Loop nest
  - HW Constraints
- Mapping space:
  - Loop ordering
  - Loop bound
  - Spatial choice
- Tuning



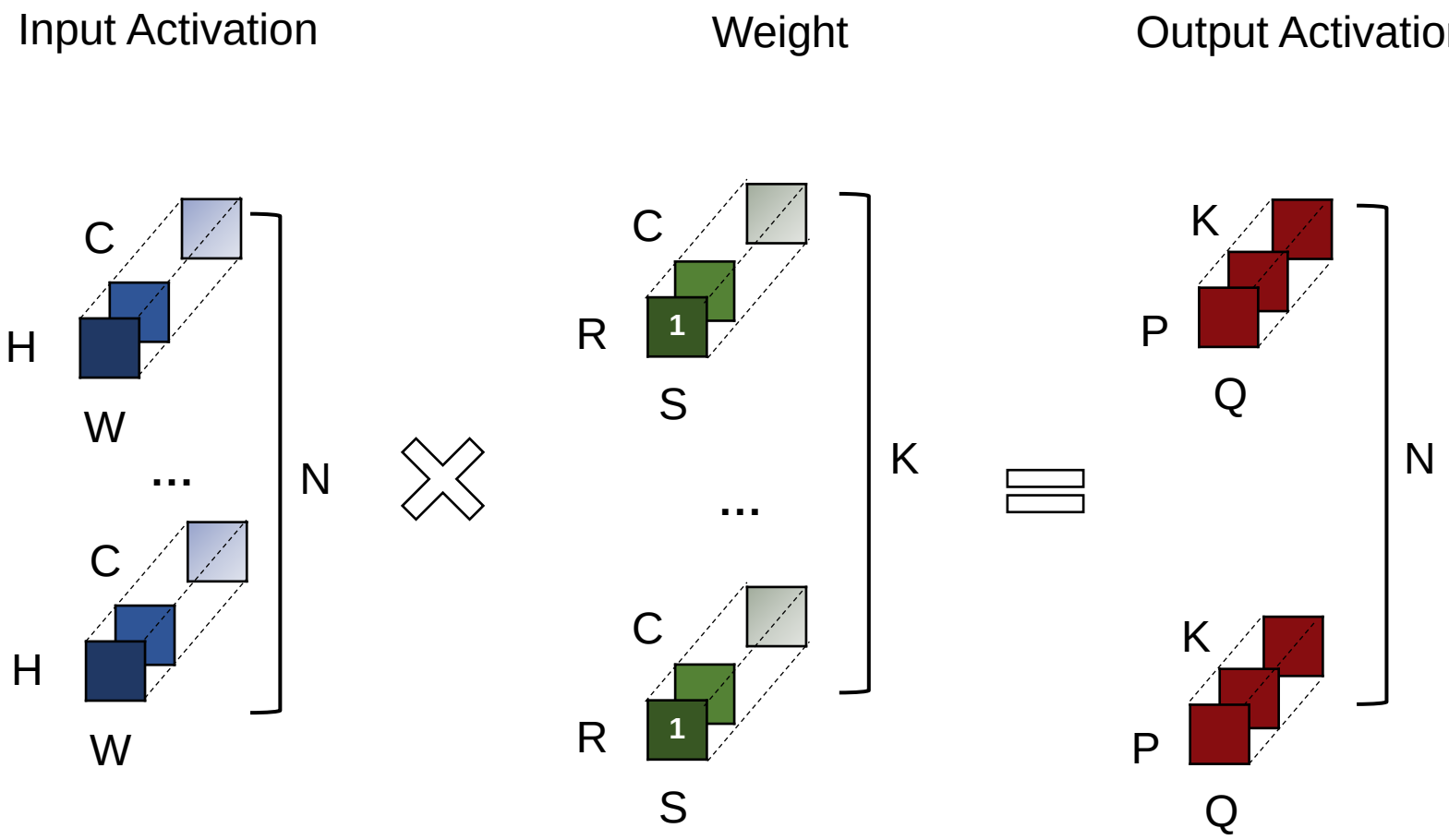
Mapping

- **DNN Mapping Problem:**
  - **Loop nest**
  - **HW Constraints**
- **Mapping space:**
  - **Loop ordering**
  - **Loop bound**
  - **Spatial choice**
- **Tuning**

Dataflow Recap

- Defines the execution order of the DNN operations in hardware.
  - Computation order
  - Data movement order
- Loop nest is a compact way to describe the execution order, i.e., dataflow, supported in hardware.
  - `for`: temporal for, describes the temporal execution order.
  - `spatial_for`: describes parallel execution.

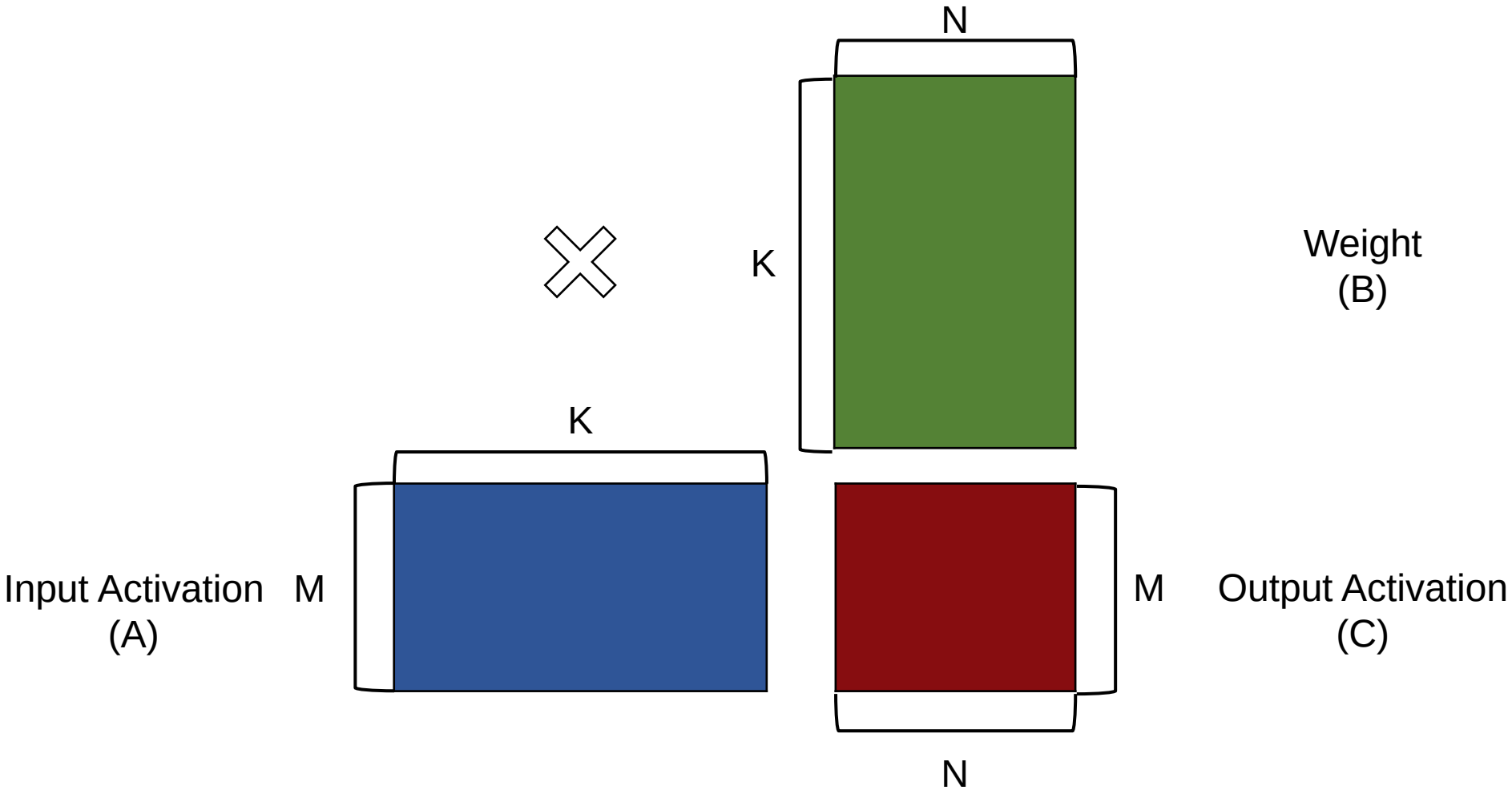
Fully-Connected Layer



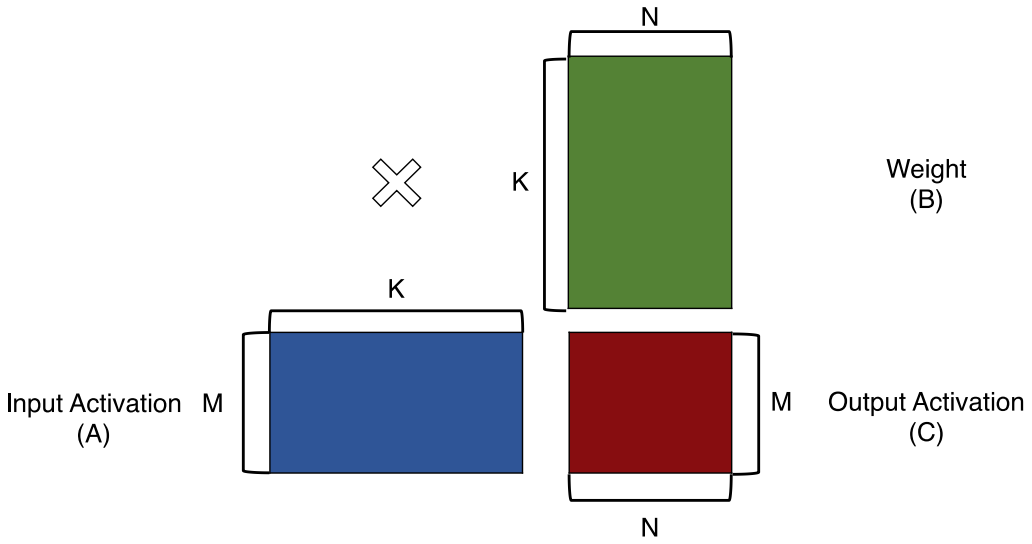
H = 1
W = 1
R = 1
S = 1
P = 1
Q = 1
stride = 1
padding= 0

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

Matrix Multiply



Matrix Multiply

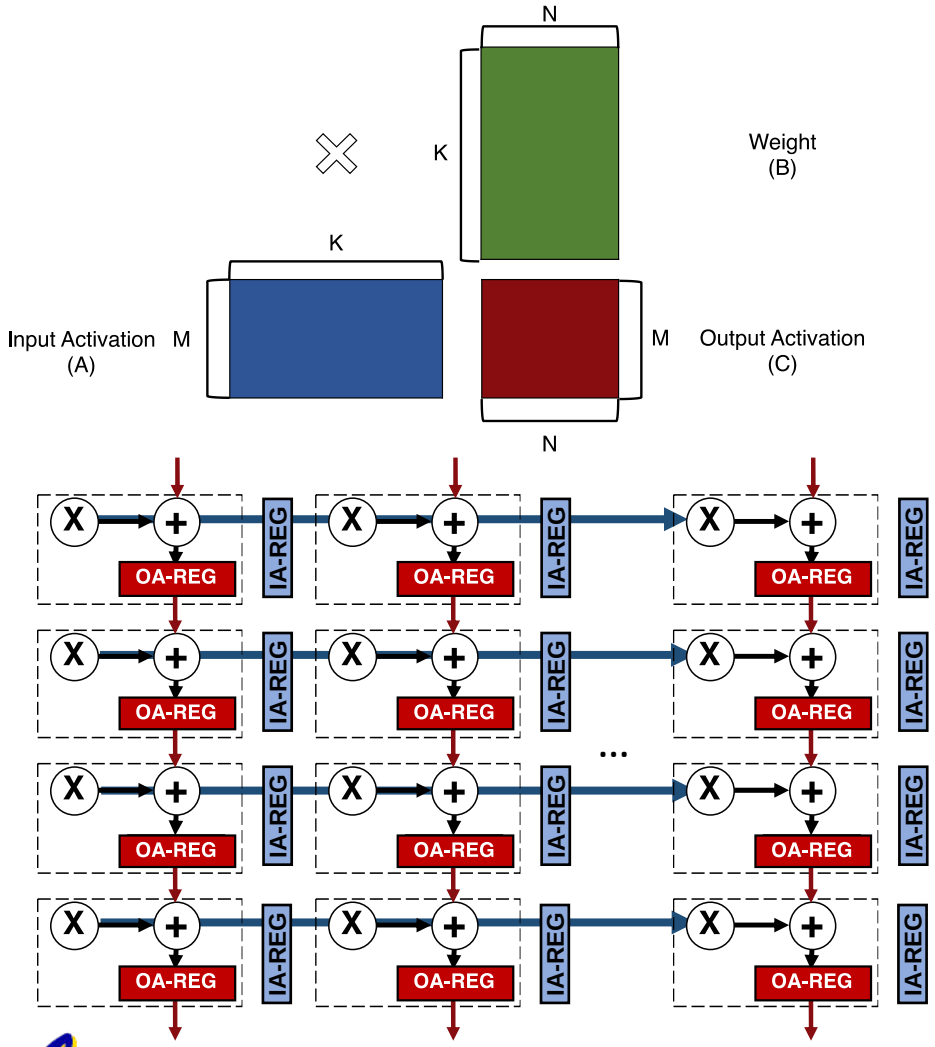



```
for (m=0; m<M; m++) {
  for (n=0; n<N; n++) {
    OA[n,m] = 0;
    for (k=0; k<K; k++) {
      OA[n,m] += IA[m, k]
                 * W[k, n];
    }
  }
}
```

for each output activation

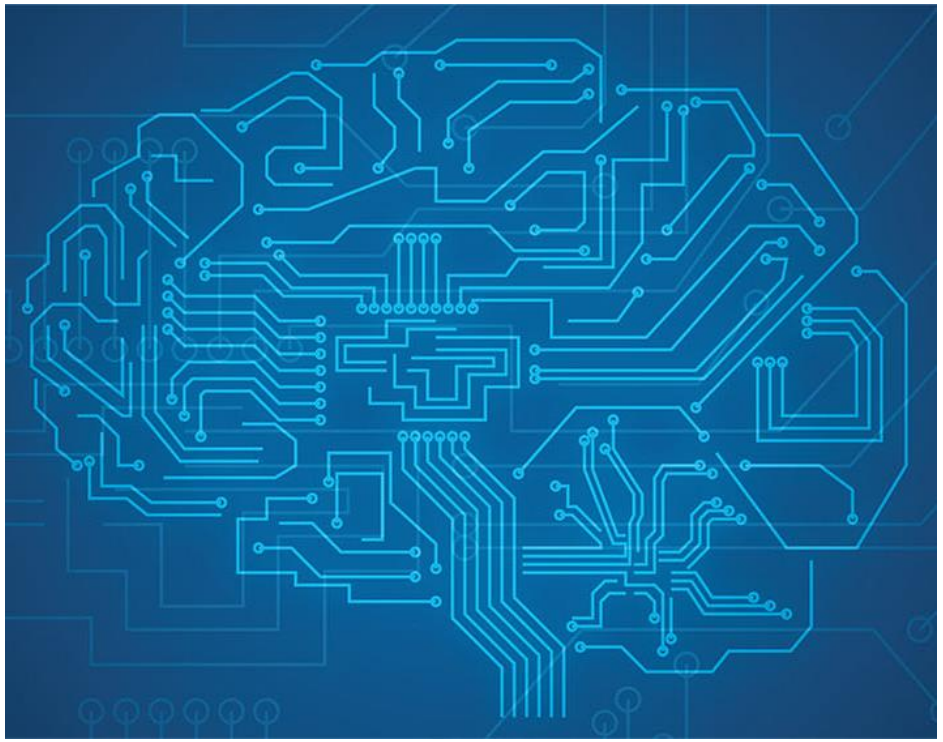
convolution window

Datapath Optimization Combined: TPU




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

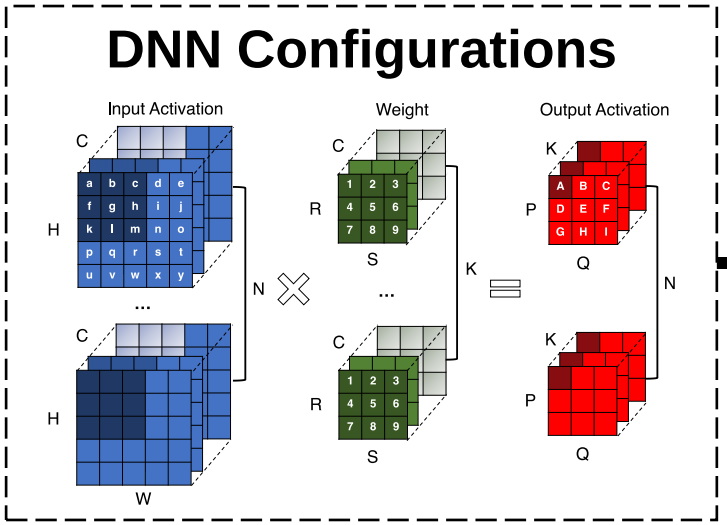
- Systolic accumulation
- Systolic multicast

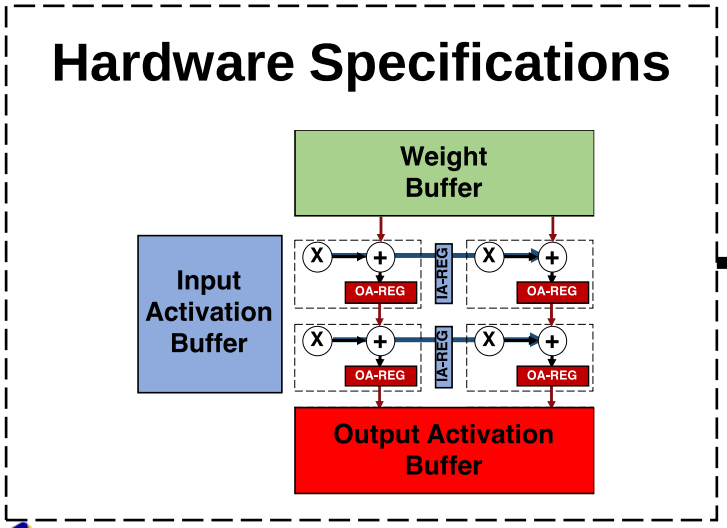


Mapping

- **DNN Mapping Problem:**
  - Loop nest
  - **HW Constraints**
- Mapping space:
  - Loop ordering
  - Loop bound
  - Spatial choice
- Tuning

The DNN Mapping Problem

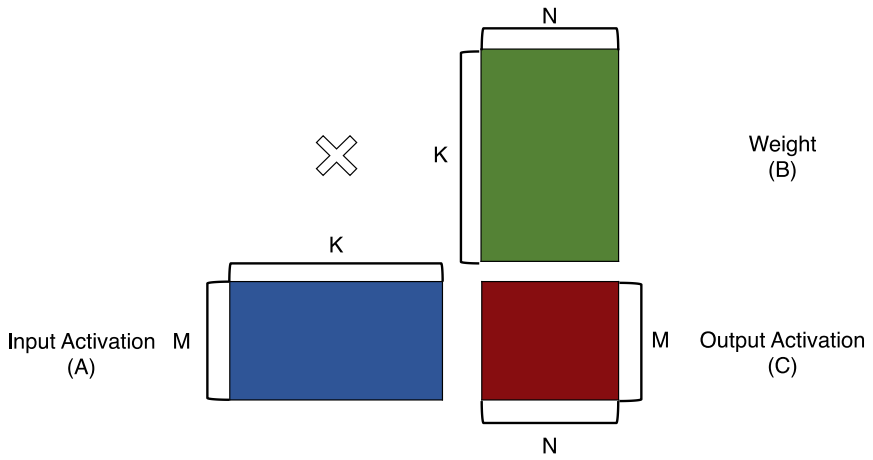


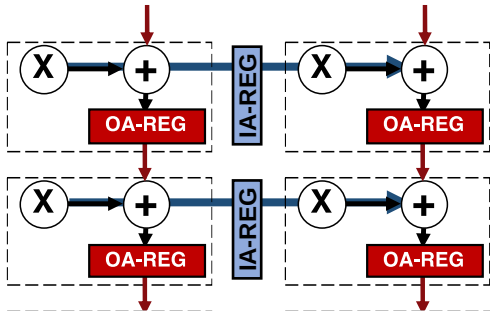


Mapping (Blocking/Tiling/ Compiling)

Execution Order (on the target HW)

Q1: What if the systolic array size < K/N?

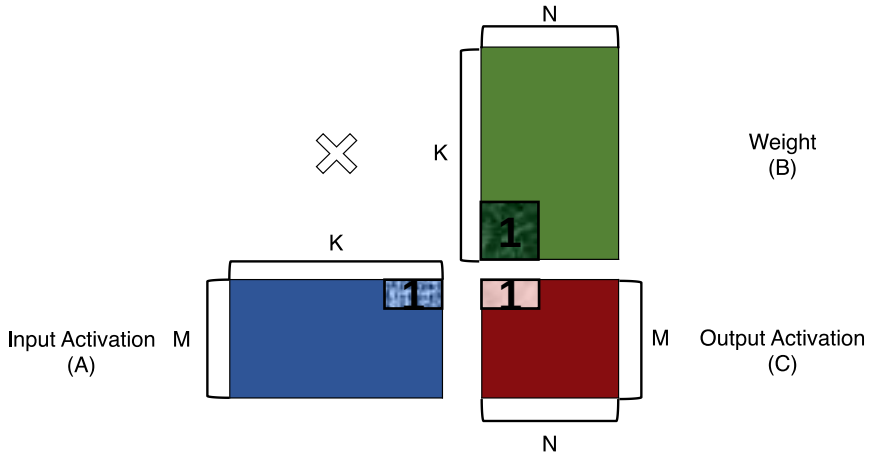


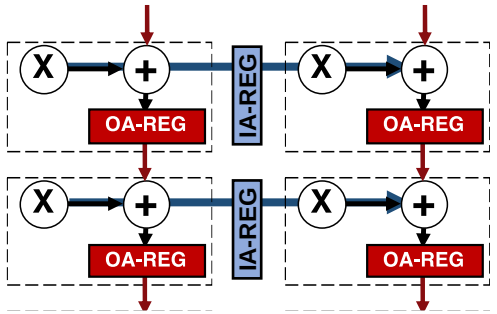


- Hardware Specifications:
  - Systolic array size: 2x2

```
for (m=0; m<M; m++) {
  spatial_for (n=0; n<2; n++) {
    OA[n,m] = 0;
    spatial_for (k=0; k<2; k++) {
      OA[n,m] +=  IA[m, k]
                * W[k, n];
    }
    //OA[n,m] is not done yet.
  }
}
```

Q1: What if the systolic array size < K/N?

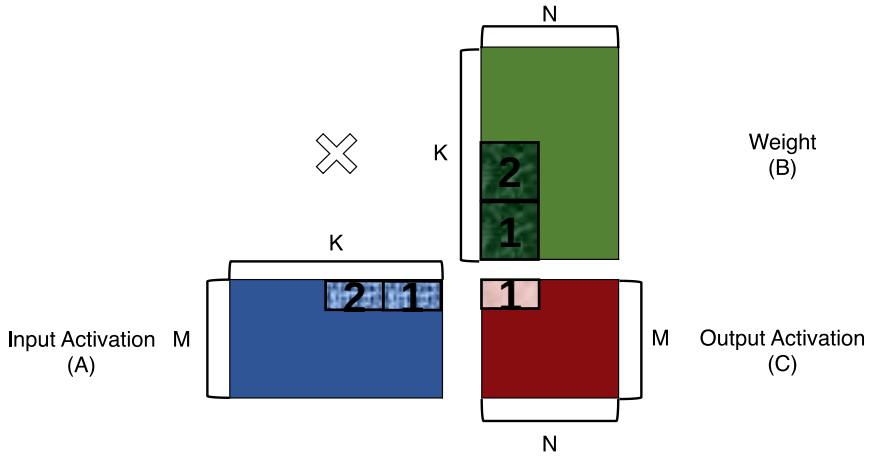


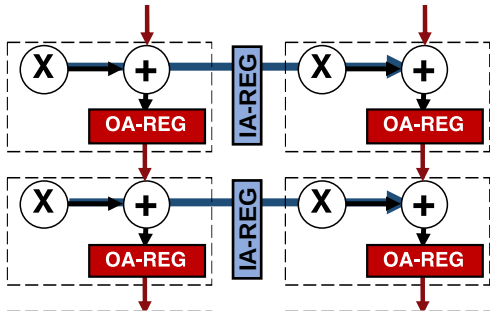


- Hardware Specifications:
  - Systolic array size: 2x2
- Notation:
  - K?/N?: loop bounds

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

Q1: What if the systolic array size < K/N?

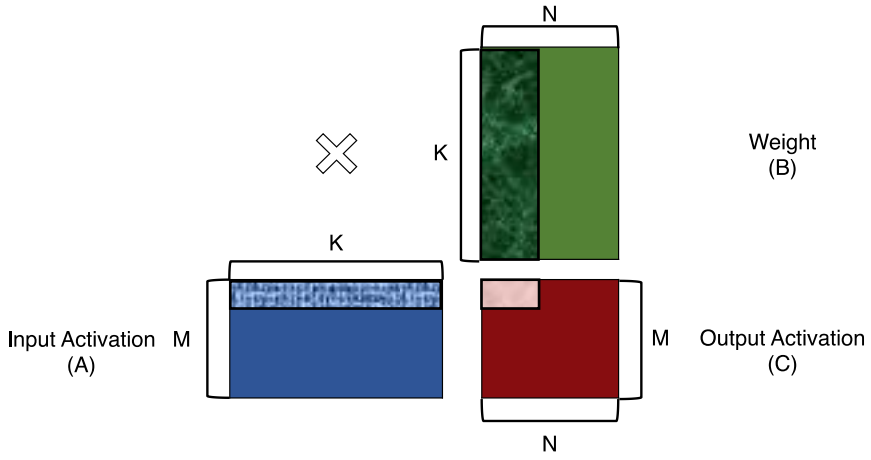


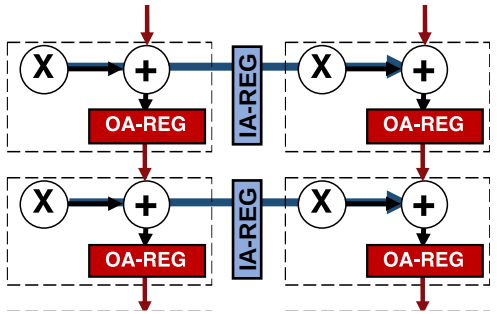


- Hardware Specifications:
  - Systolic array size: 2x2
- Notation:
  - K?/N?: loop bounds

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

Q1: What if the systolic array size < K/N?

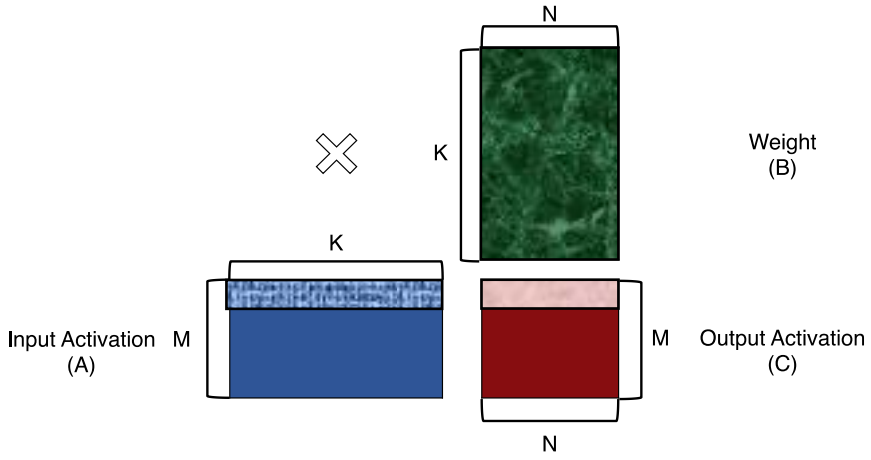


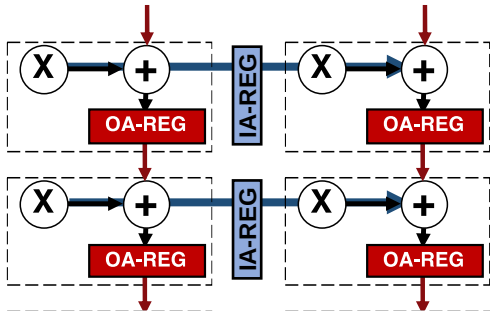


- Hardware Specifications:
  - Systolic array size: 2x2
- Notation:
  - K?/N?: loop bounds

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

Q1: What if the systolic array size < K/N?

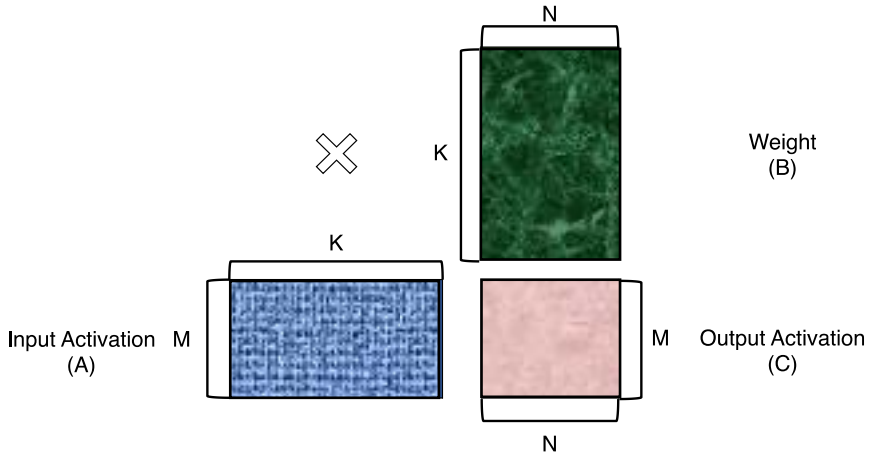


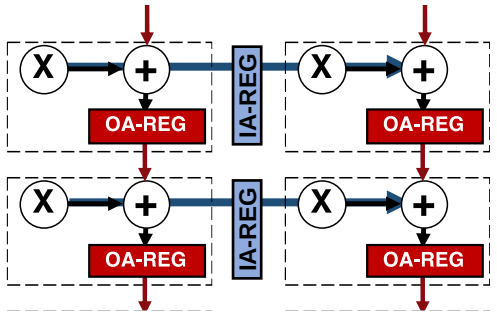


- Hardware Specifications:
  - Systolic array size: 2x2
- Notation:
  - K?/N?: loop bounds

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

Q1: What if the systolic array size < K/N?

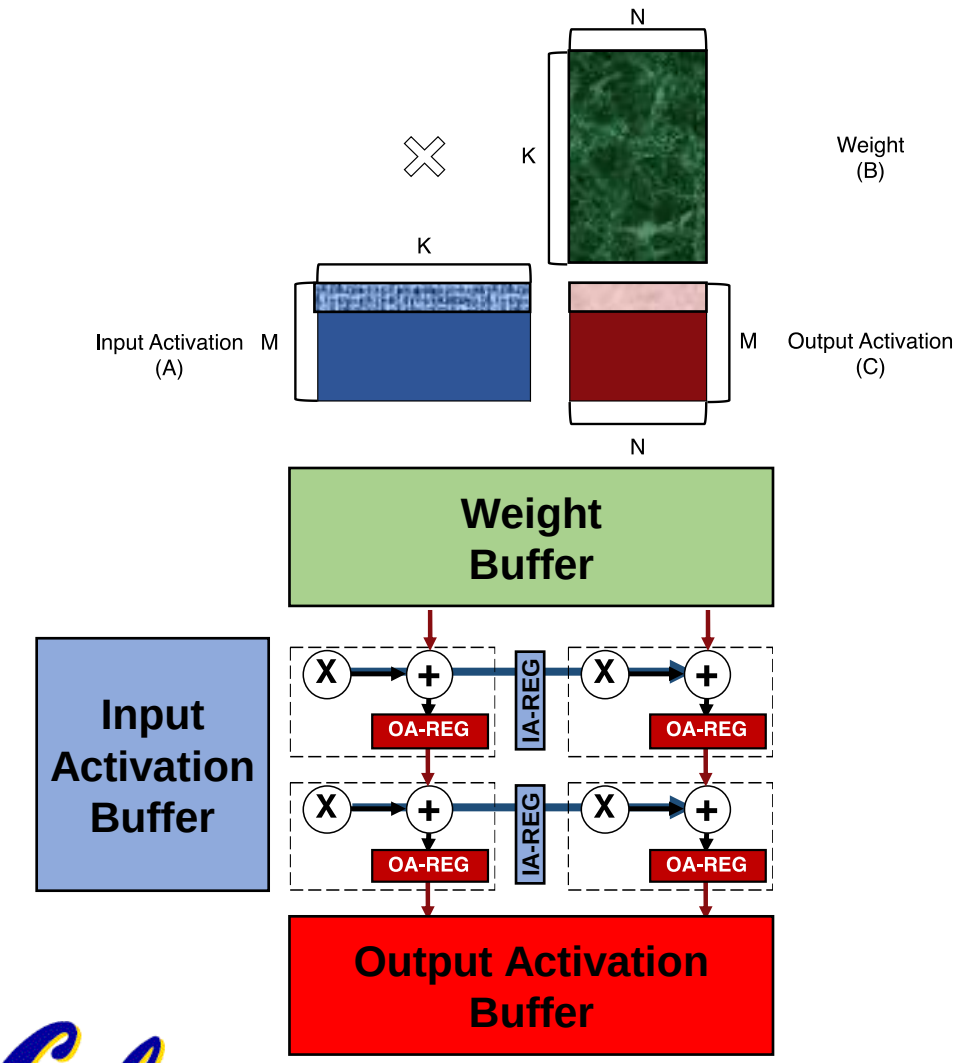




- Hardware Specifications:
  - Systolic array size: 2x2
- Notation:
  - K?/N?: loop bounds

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

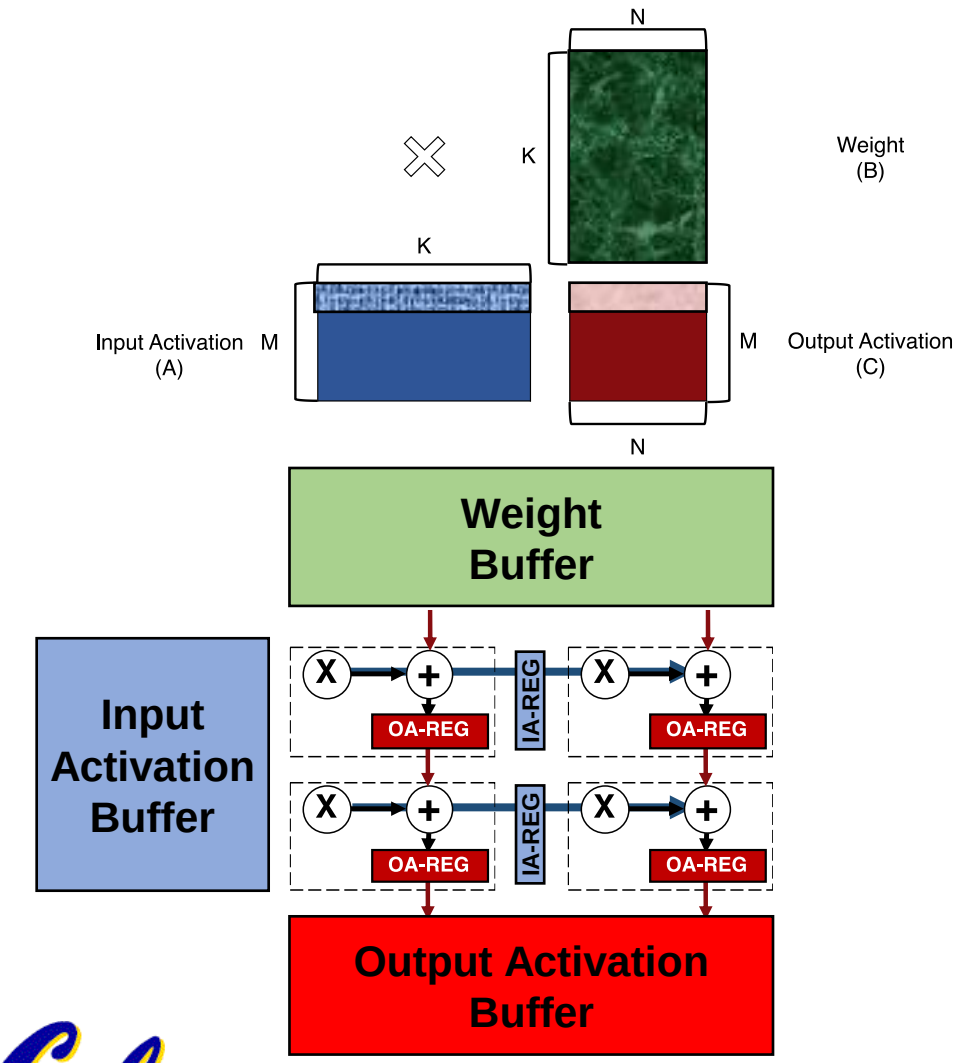
Q2: What if weight buffer size < N*K?



- Hardware Specifications:
  - Systolic array size: 2x2
  - Explicit data movement

```
for (m=0; m<M; m++) {
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m] += IA[m, k1*K0+k0]
                          * W[k1*K0+k0, n1*N0+n0];
}}}}}
```

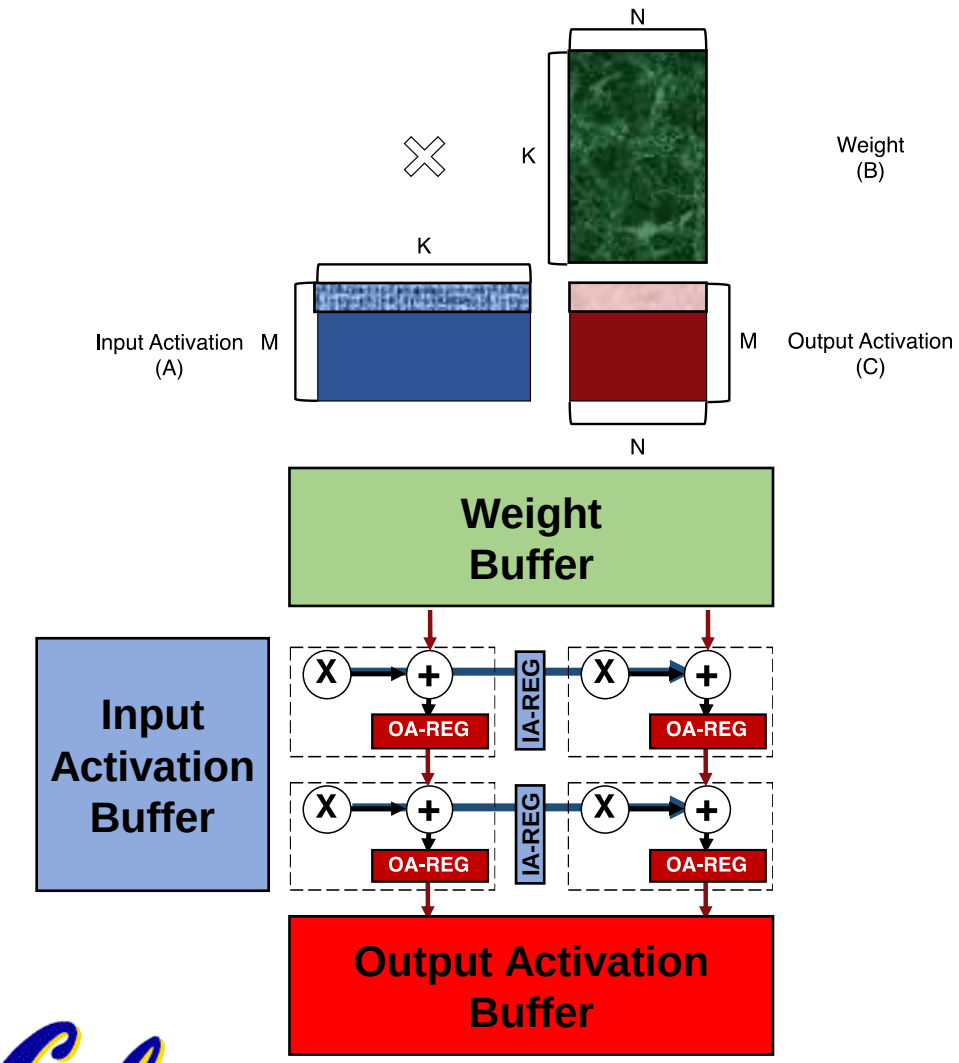
Q2: What if weight buffer size < N*K?



- Hardware Specifications:
  - Systolic array size: 2x2
  - Explicit data movement

```
for (m=0; m<M; m++) {
  mvin(W[0:K,0:N]);
  mvin(IA[m:m+1,0:K]);
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m]+=IA[m, k1*K0+k0]
                        * W[k1*K0+k0, n1*N0+n0];
}}}}
  mvout(OA[0:N,m:m+1]);
}
```

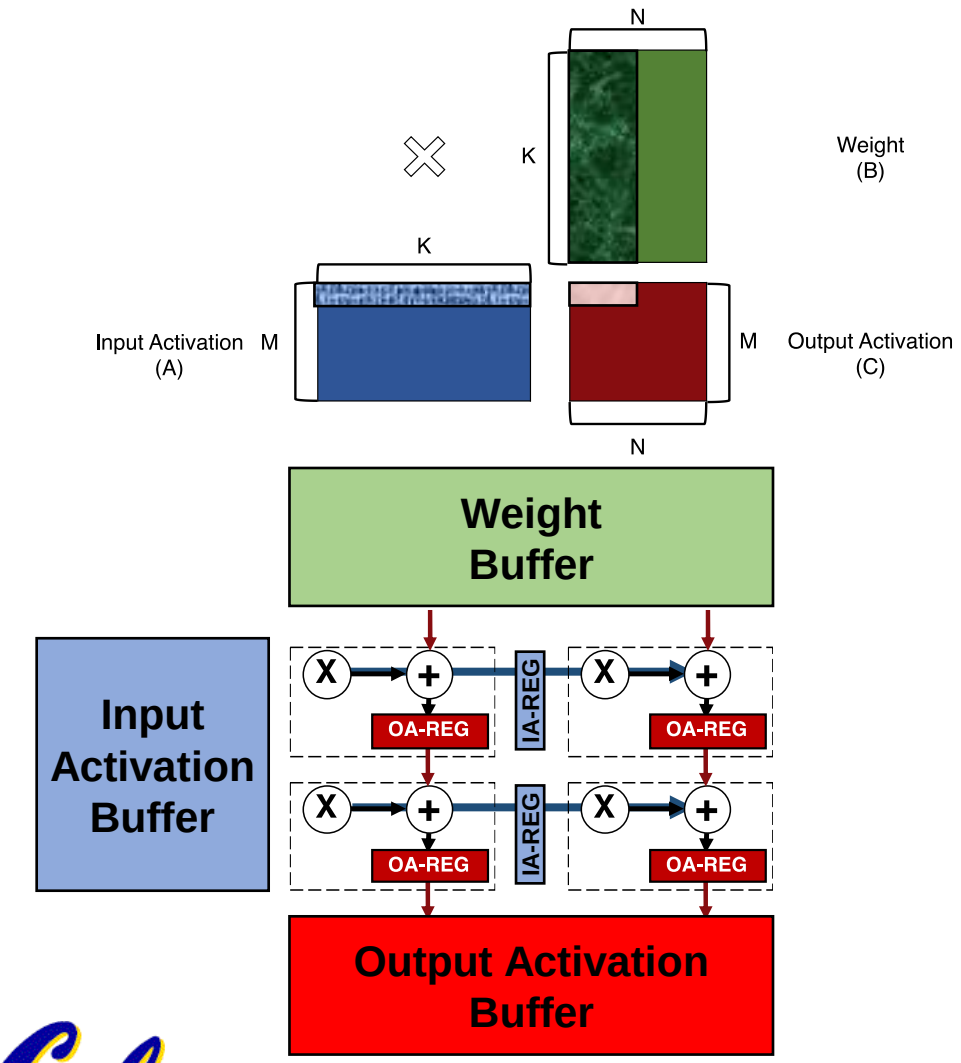
Q2: What if weight buffer size < N*K?



- Hardware Specifications:
  - Systolic array size: 2x2
  - Explicit data movement
  - Output/Weight/Input buffer sizes

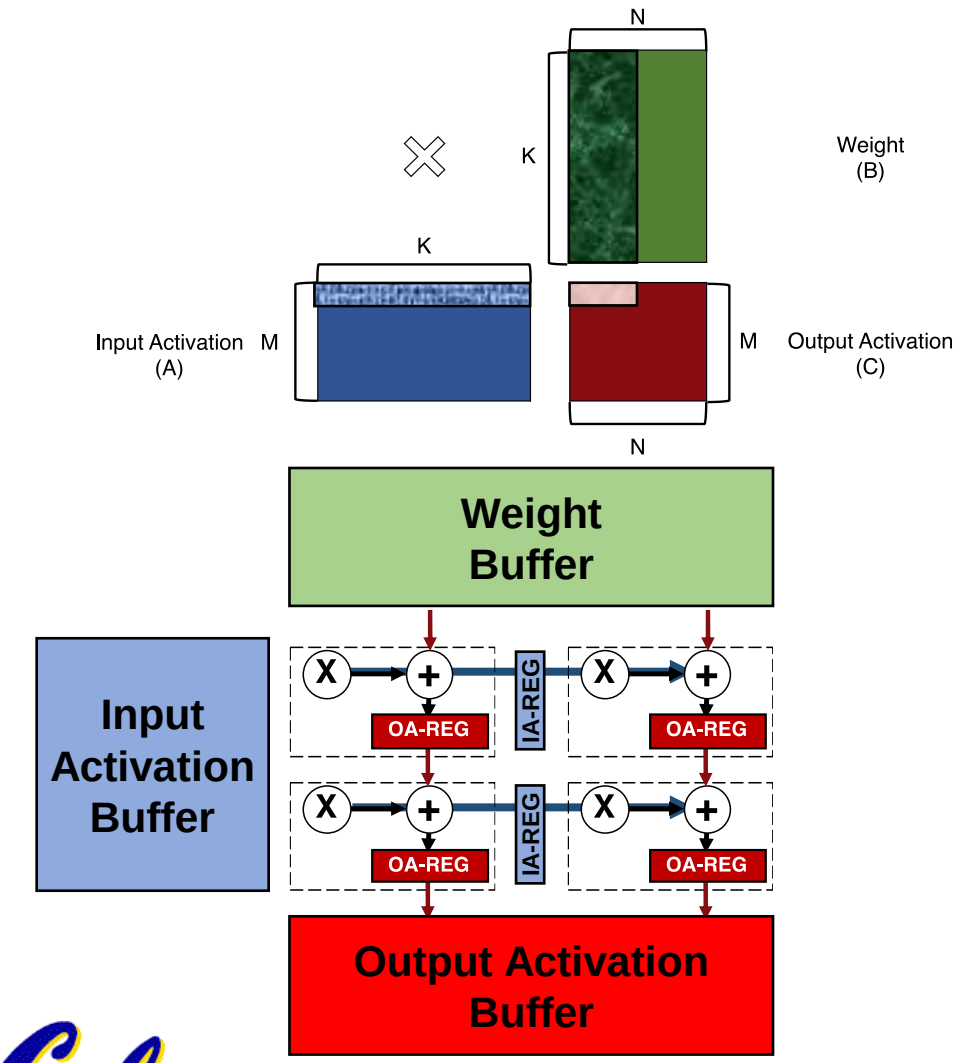
```
for (m=0; m<M; m++) {
  mvin(W[0:K,0:N]);// W buffer >= NK
  mvin(IA[m:m+1,0:K]);// IA buffer >= 1*K
  for (n1=0; n1<N1; n1++) {
    OA[n1*N0:(n1+1)*N0,m] = 0;
    for (k1=0; k1<K1; k1++) {
      spatial_for (n0=0; n0<N0; n0++) {
        spatial_for (k0=0; k0<K0; k0++) {
          OA[n1*N0+n0,m]+=IA[m, k1*K0+k0]
                        * W[k1*K0+k0, n1*N0+n0];
}}}}
  mvout(OA[0:N,m:m+1]);// OA buffer >= 1*N
}
```

Q2: What if weight buffer size < N*K?



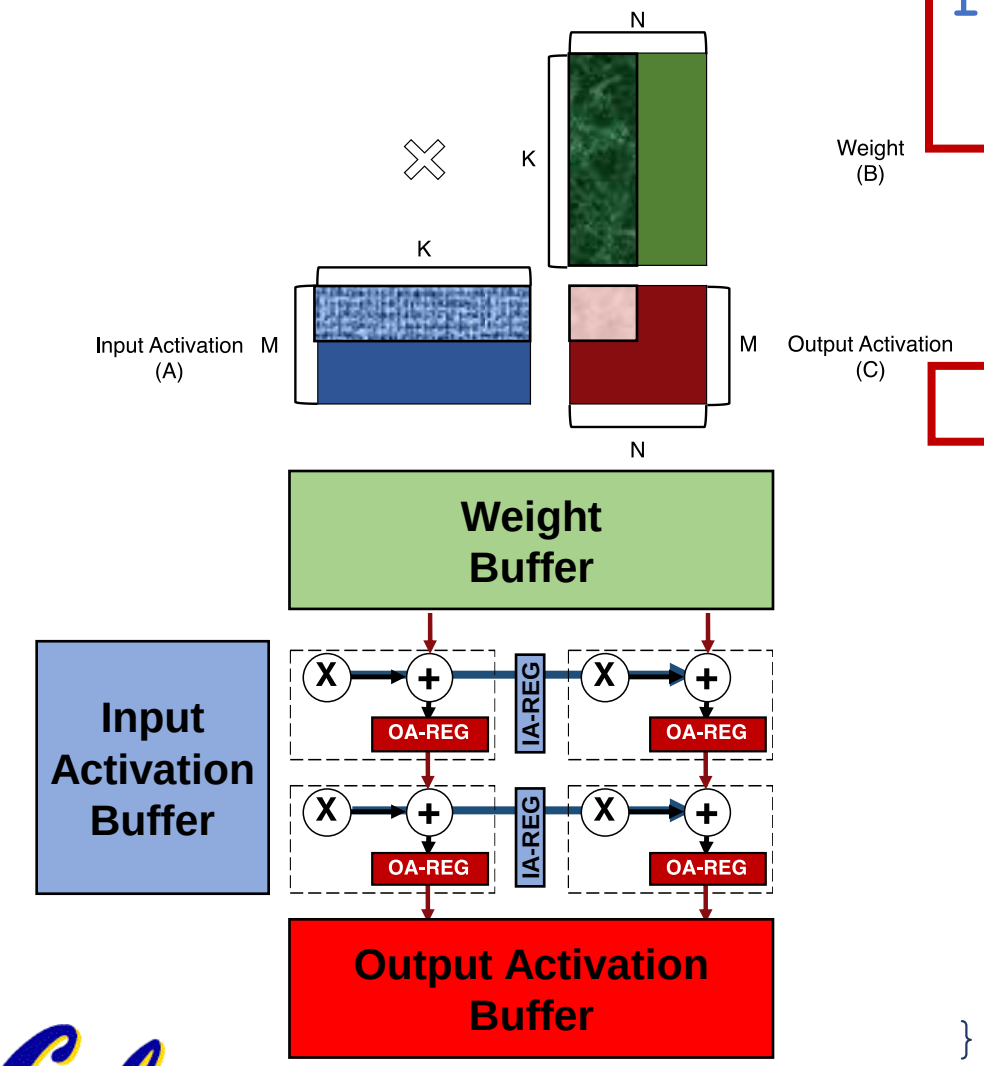

```
for (m=0; m<M; m++) {

  mvin(IA[m:m+1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K < NK
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]=0;
    for (n1=0; n1<N1; n1++) {
      for (k1=0; k1<K1; k1++) {
        spatial_for (n0=0; n0<N0; n0++) {
          spatial_for (k0=0; k0<K0; k0++) {
            OA[n2*N1*N0+n1*N0+n0,m]
           +=IA[m, k1*K1+k0]
           * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]);
  }
}
```

Q3: What if input buffer size > 1*K?



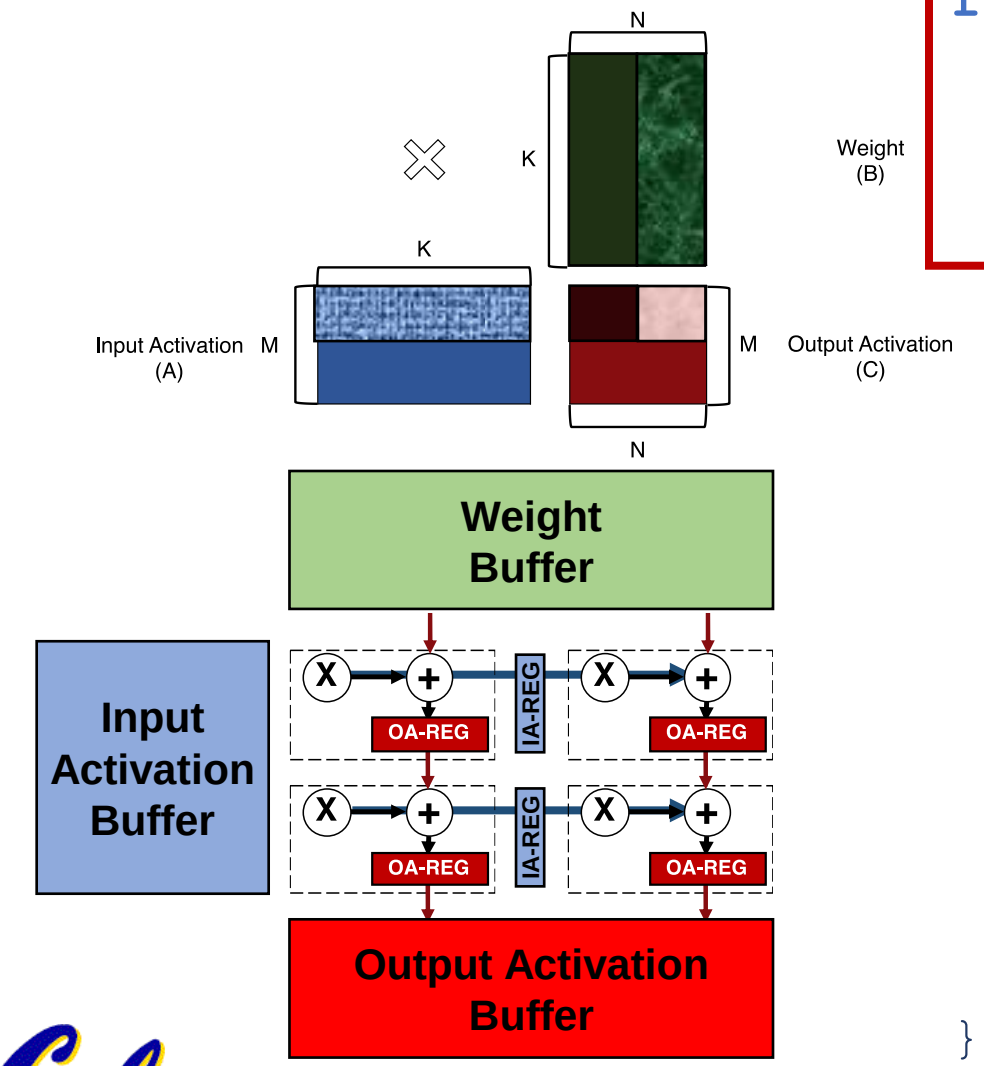

```
for (m=0; m<M; m++) {
  // IA buffer stores: 1*K
  mvin(IA[m:m+1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]=0;
    for (n1=0; n1<N1; n1++) {
      for (k1=0; k1<K1; k1++) {
        spatial_for (n0=0; n0<N0; n0++) {
          spatial_for (k0=0; k0<K0; k0++) {
            OA[n2*N1*N0+n1*N0+n0,m]
           +=IA[m, k1*K1+k0]
           * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0,m:m+1]);
  }
}
```

Q3: What if input buffer size > 1*K?



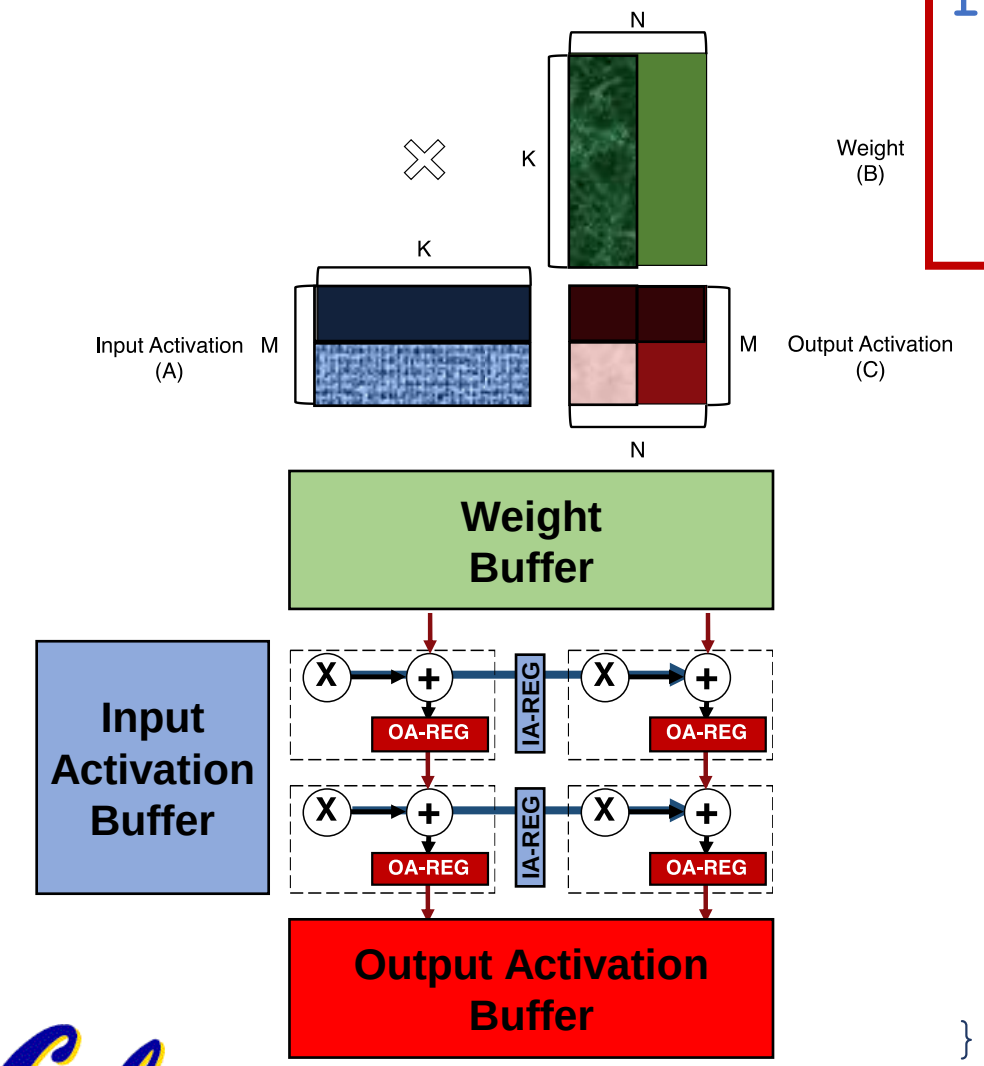

```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

Q3: What if input buffer size > 1*K?



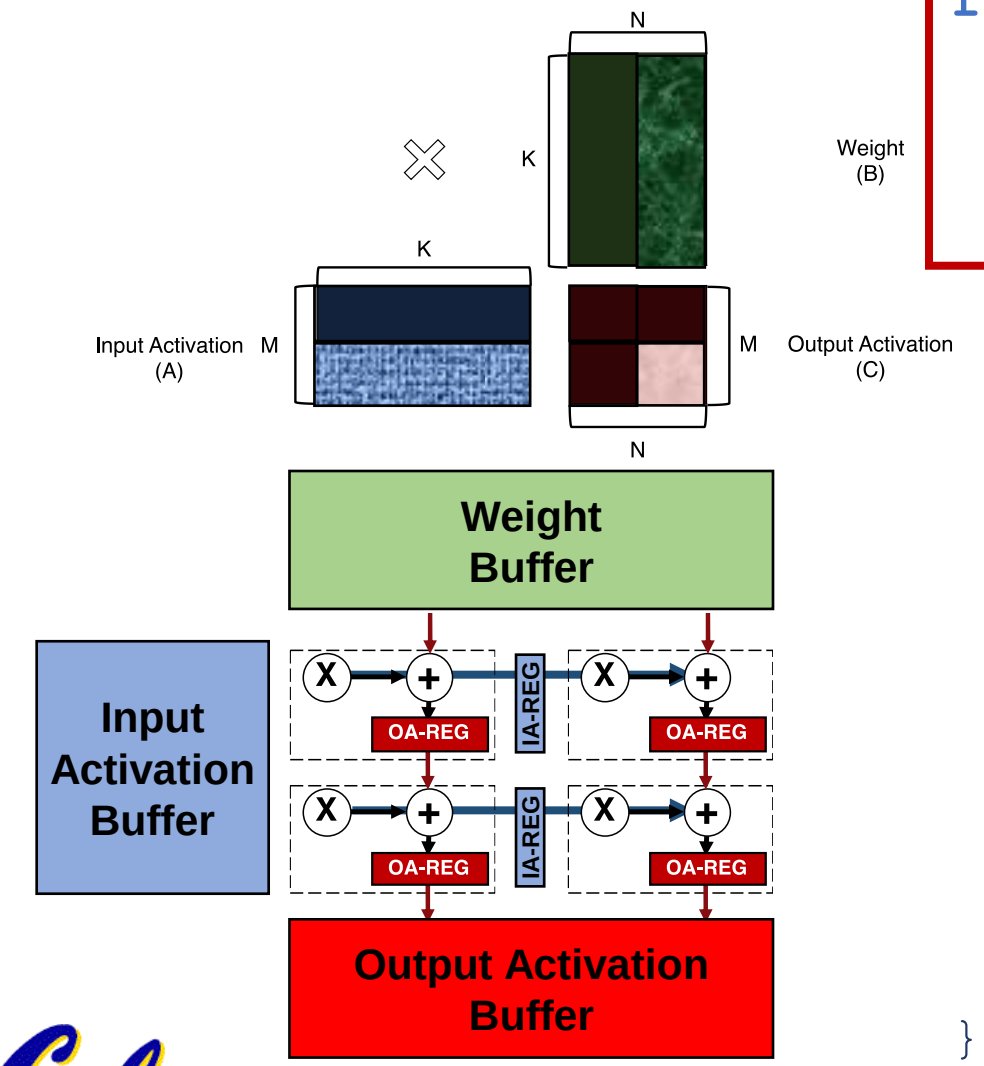

```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

Q3: What if input buffer size > 1*K?



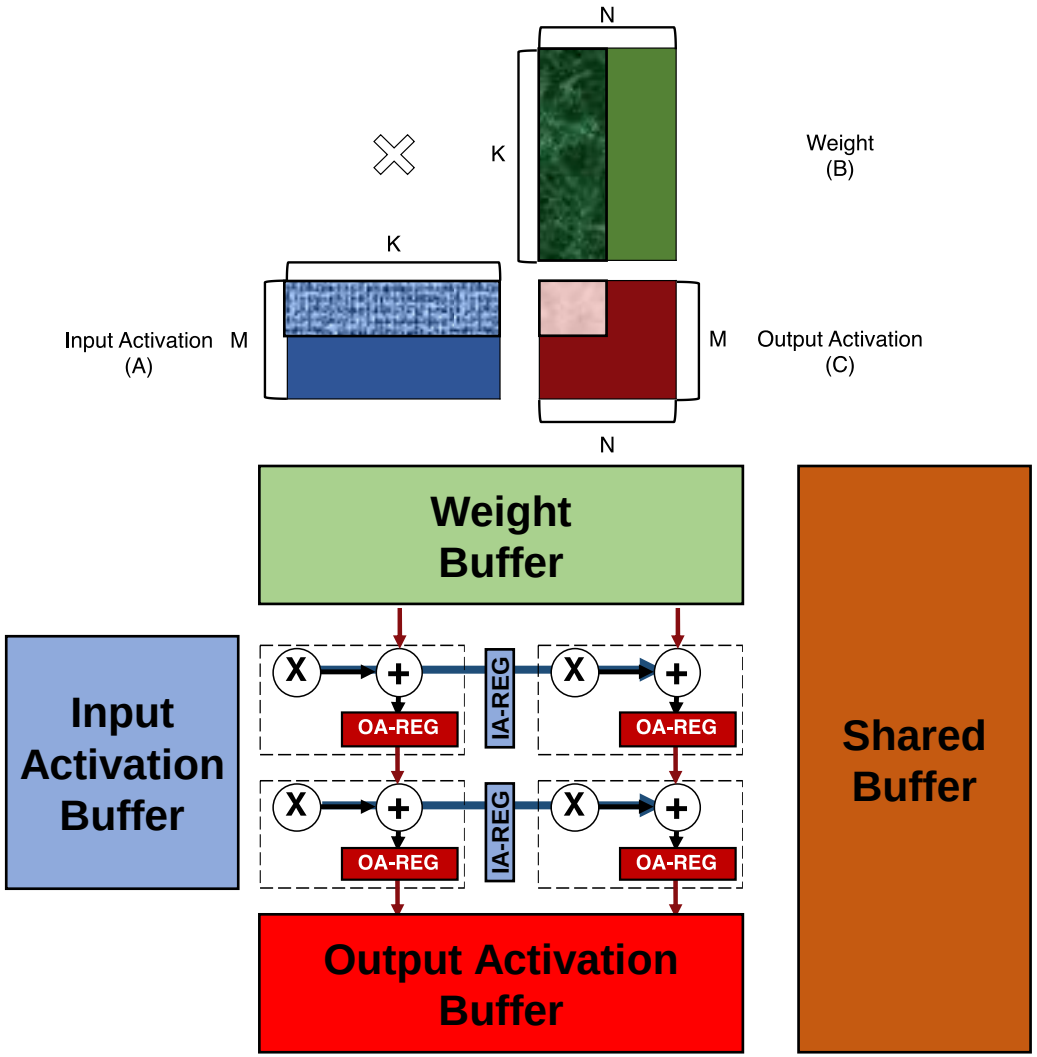

```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

Q3: What if input buffer size > 1*K?




```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

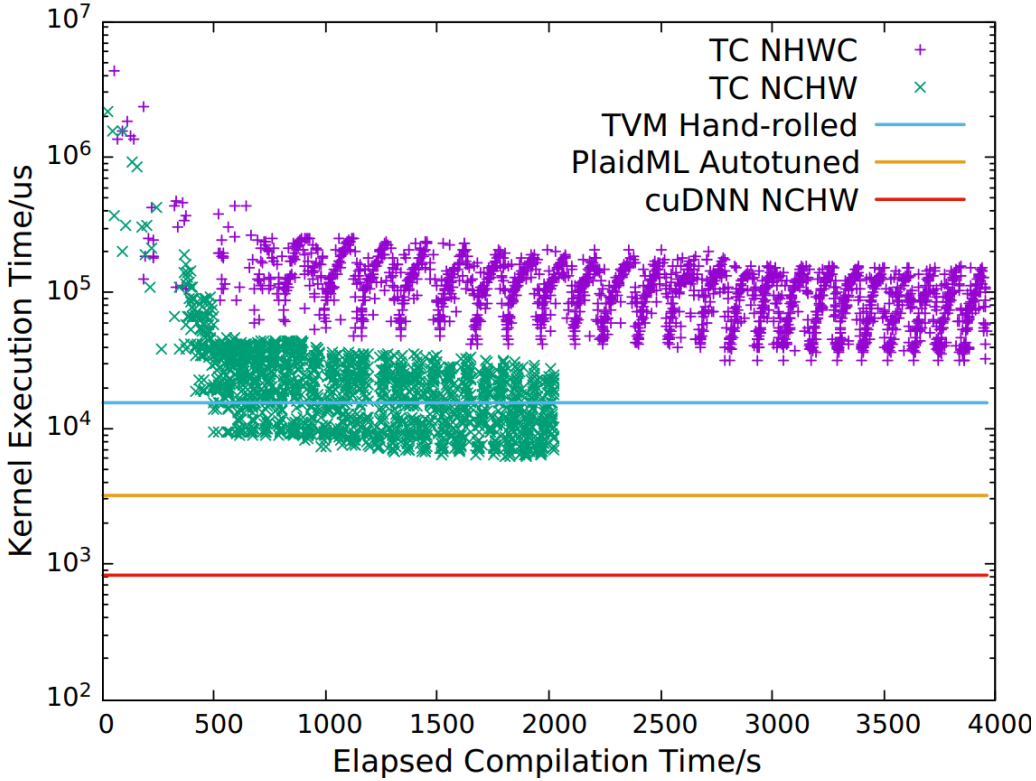
Q4: What if there is another shared buffer?




```
for (m3=0; m3<M3; m3++) {
 for (n3=0; n3<N3; n3++) {
  // Shared buffer blocking
```

```
   for (m2=0; m2<M2; m2++) {
     // IA buffer stores: M1*K
     mvin(IA[…:…]);
     for (n2=0; n2<N2; n2++) {
       // W buffer stores: N1*N0*K
       mvin(W[…:…,…:…]);
       OA[…:…,…:…]=0;
       for (m1=0; m1<M1; m1++) {
         for (n1=0; n1<N1; n1++) {
           for (k1=0; k1<K1; k1++) {
             …
     }}}}}
     mvout(OA[…:…,…:…]);
}}}}
```

The DNN Mapping Problem

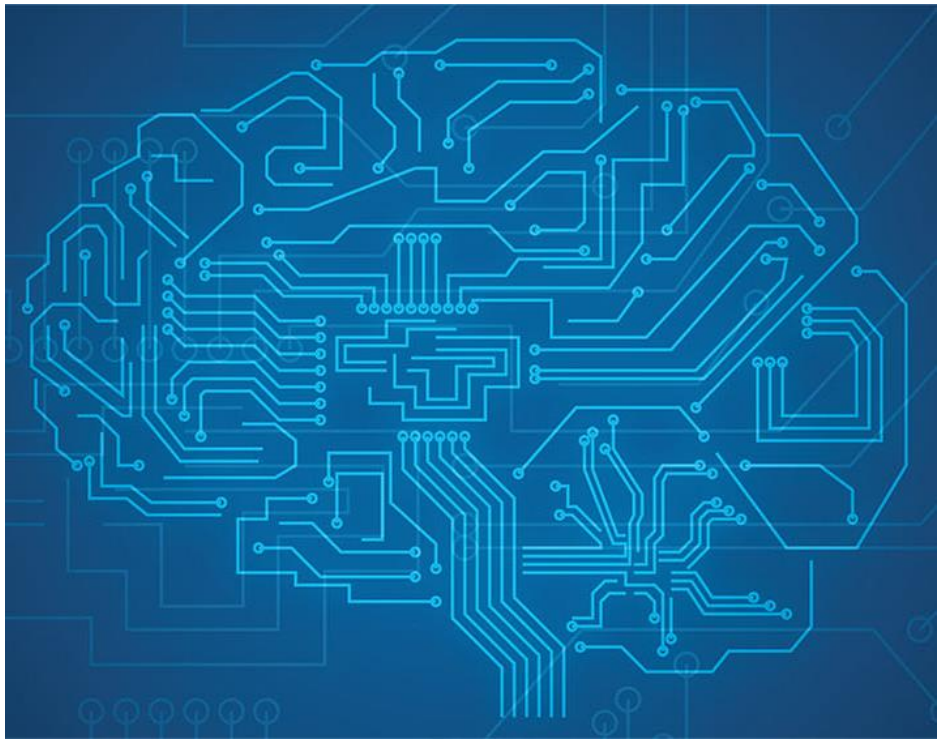


(a) Strided conv2d (batch=32)

Machine Learning Systems are Stuck in a Rut

Administrivia

- Lab 3 is due this week.
  - Have fun!
- Paper reading is NOT required this week.
- Project starts next week:
  - Project meet-up last & this weeks.
    - 3/4 2-3pm
  - Talk to the teaching staff about your project ideas/scope.
  - 3/19 Project Proposal Due



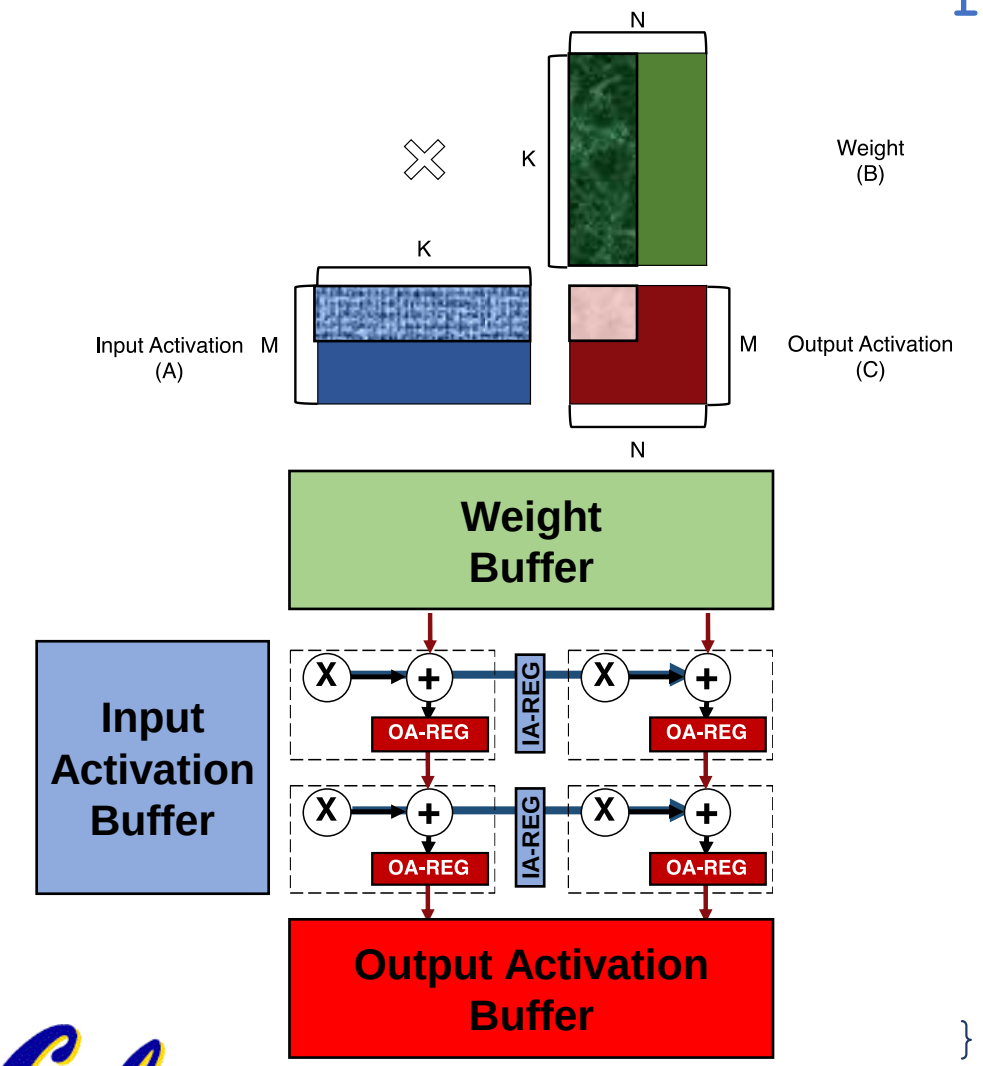
Mapping

- DNN Mapping Problem:
  - Loop nest
  - HW Constraints
- **Mapping space:**
  - **Loop ordering**
  - **Loop bound**
  - **Spatial choice**
- Tuning

Mapping Dimensions

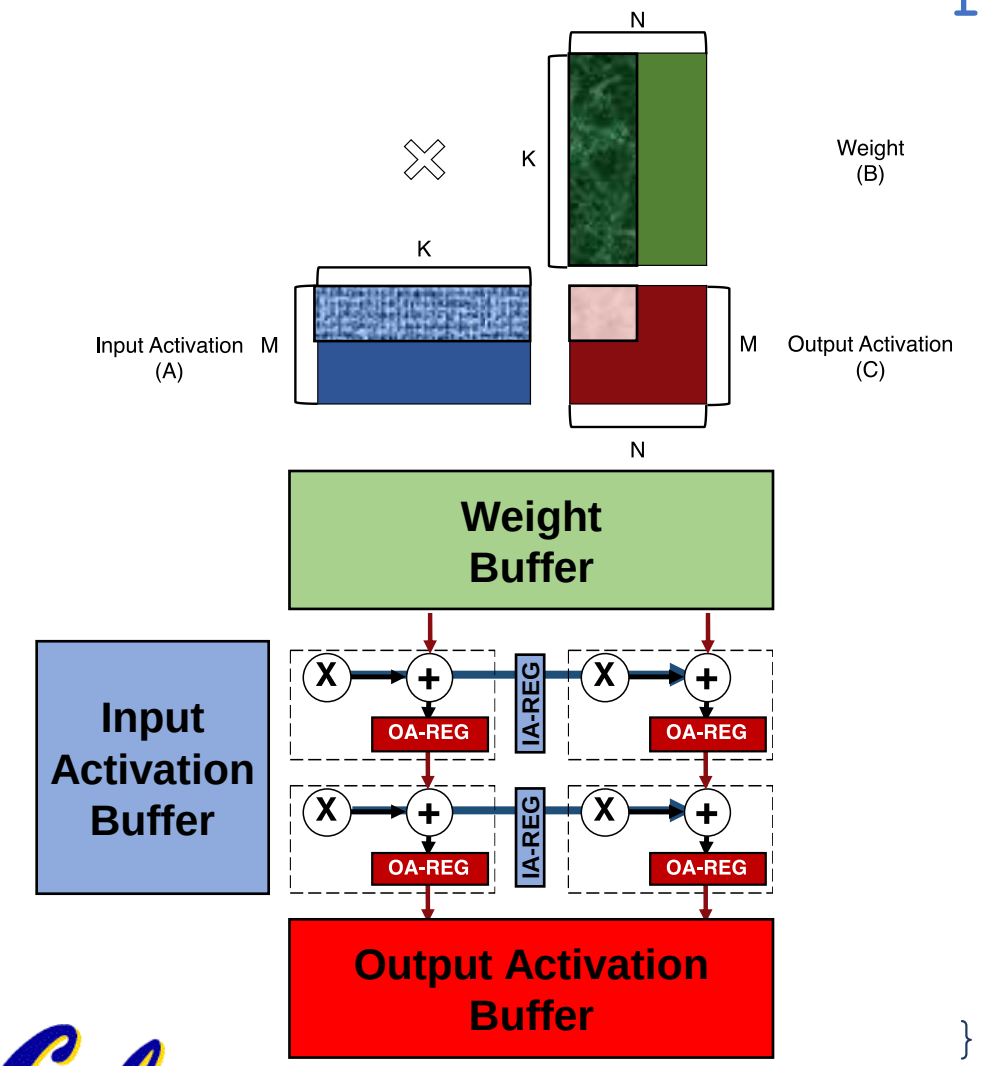
- Loop ordering:
  - Which index goes to the inner/outer loop?
- Loop bounds:
  - What are the loop bounds (i.e., N?/K?/M?) for each loop?
- Spatial choice:
  - Which loop should be spatial/temporal?
    - Data/Model Parallelism

The DNN Mapping Problem




```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

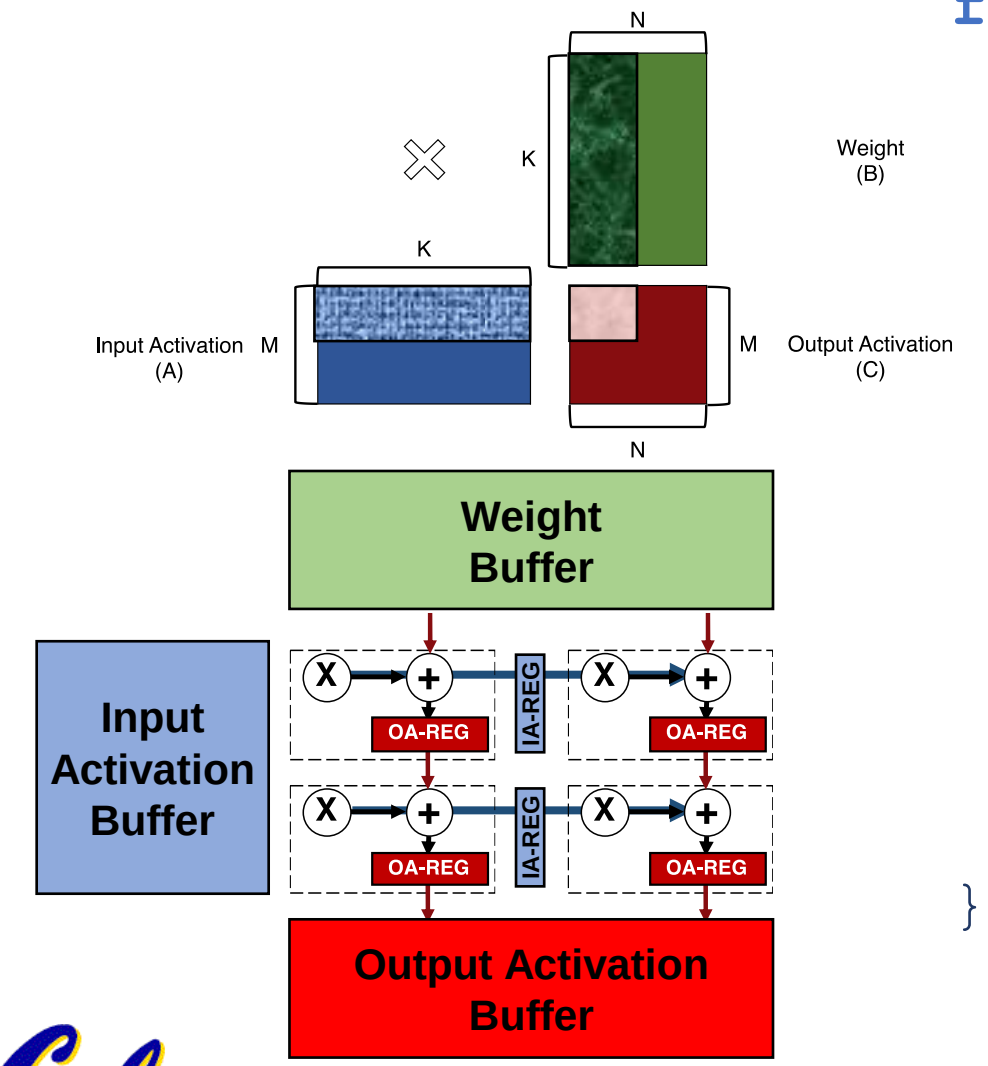
Loop Bounds




```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    OA[n2*N1*N0:(n2+1)*N1*N0,m2*M1:(m2+1)*M1]=0;
    for (m1=0; m1<M1; m1++) {
      for (n1=0; n1<N1; n1++) {
        for (k1=0; k1<K1; k1++) {
          spatial_for (n0=0; n0<N0; n0++) {
            spatial_for (k0=0;k0<K0; k0++){
              OA[n2*N1*N0+n1*N0+n0,m2*M1+m1]
            +=IA[m2*M1+m1, k1*K1+k0]
            * W[k1*K0+k0,n2*N1*N0+n1*N0+n0];
    }}}}}
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

M1*M2=M

Loop Ordering




```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:(n2+1)*N1*N0]);
    compute_matmul(*W, *IA, *OA,
                   N2, N1, N0,
                   M2, M1,
                   K1, K0,
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
             ,m2*M1:(m2+1)*M1]);
}}
```

Loop Ordering

Option 1: Loop_m2 -> Loop_n2

```
for (m2=0; m2<M2; m2++) {
  // IA buffer stores: M1*K
  mvin(IA[m2*M1:(m2+1)*M1,0:K]);
  for (n2=0; n2<N2; n2++) {
    // W buffer stores: N1*N0*K
    mvin(W[0:K,n2*N1*N0:
           (n2+1)*N1*N0]);
    compute_matmul(*W, *IA, *OA,
                   …
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

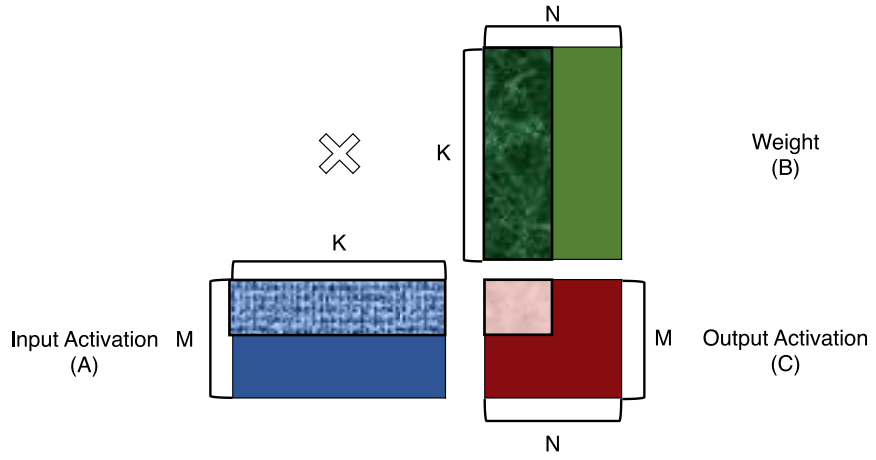
IA movement: M * K
W movement: M2 * N * K

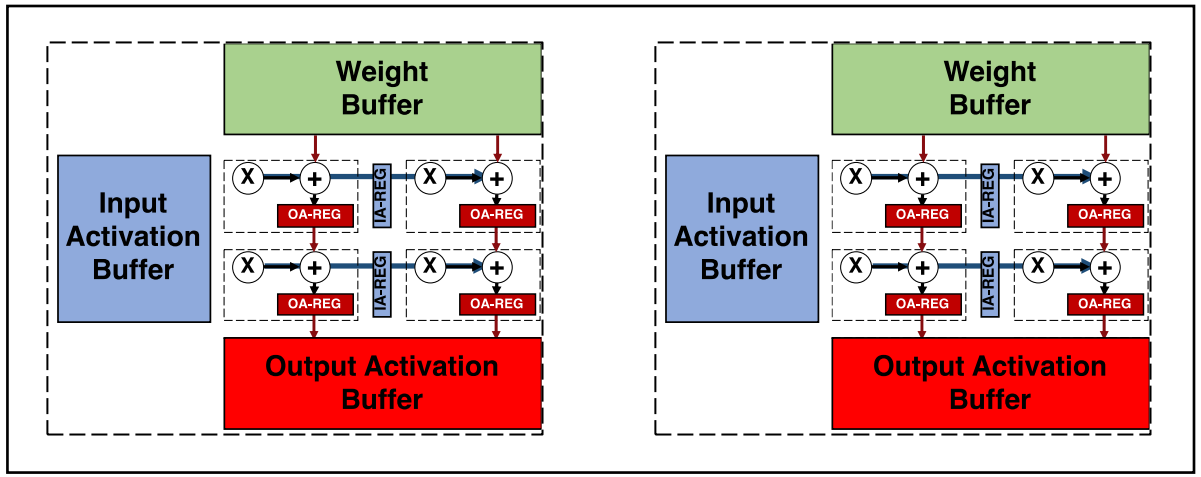
Option 2: Loop_n2 -> Loop_m2

```
for (n2=0; n2<N2; n2++) {
  // W buffer stores: N1*N0*K
  mvin(W[0:K,n2*N1*N0:
         (n2+1)*N1*N0]);
  for (m2=0; m2<M2; m2++) {
    // IA buffer stores: M1*K
    mvin(IA[m2*M1:(m2+1)*M1,0:K]);
    compute_matmul(*W, *IA, *OA,
                   …
                   m2, n2);
    mvout(OA[n2*N1*N0:(n2+1)*N1*N0
          ,m2*M1:(m2+1)*M1]);
}}
```

IA movement: N2 * M* K
W movement: N * K

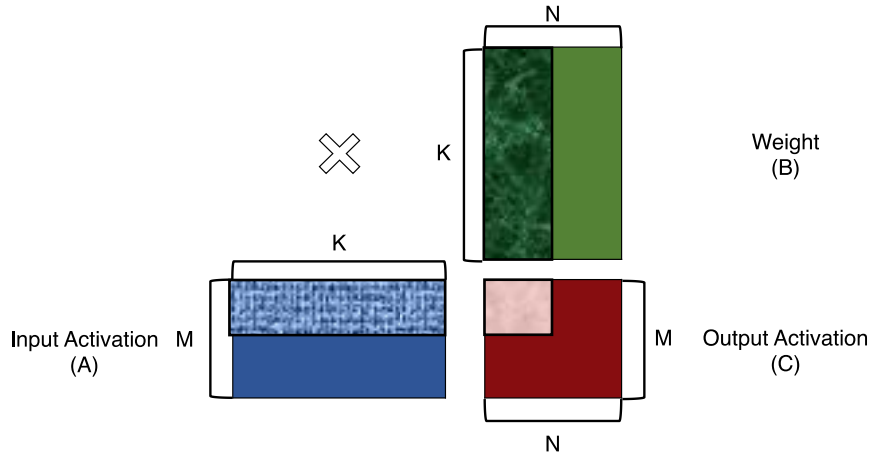
Spatial Choice

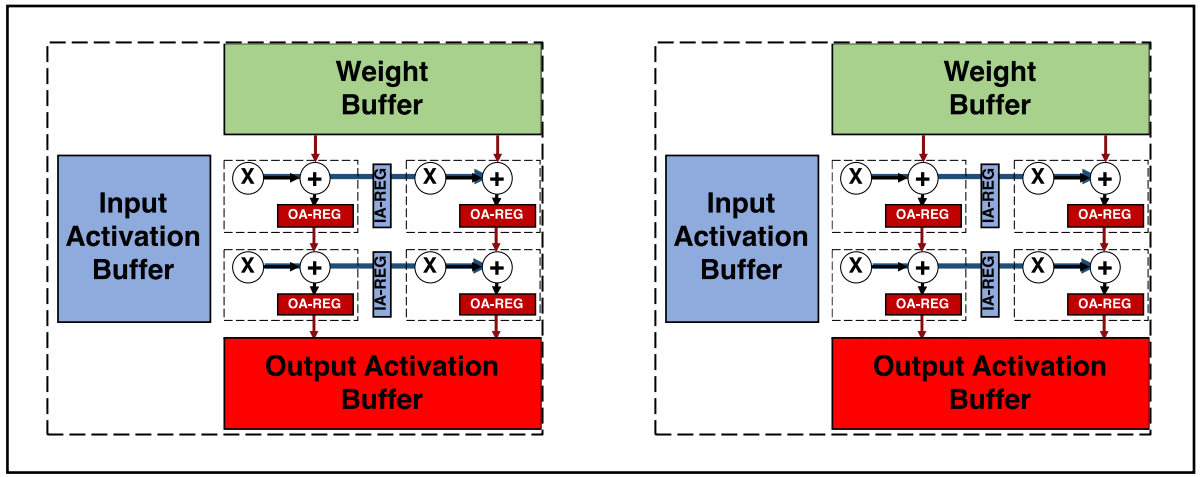




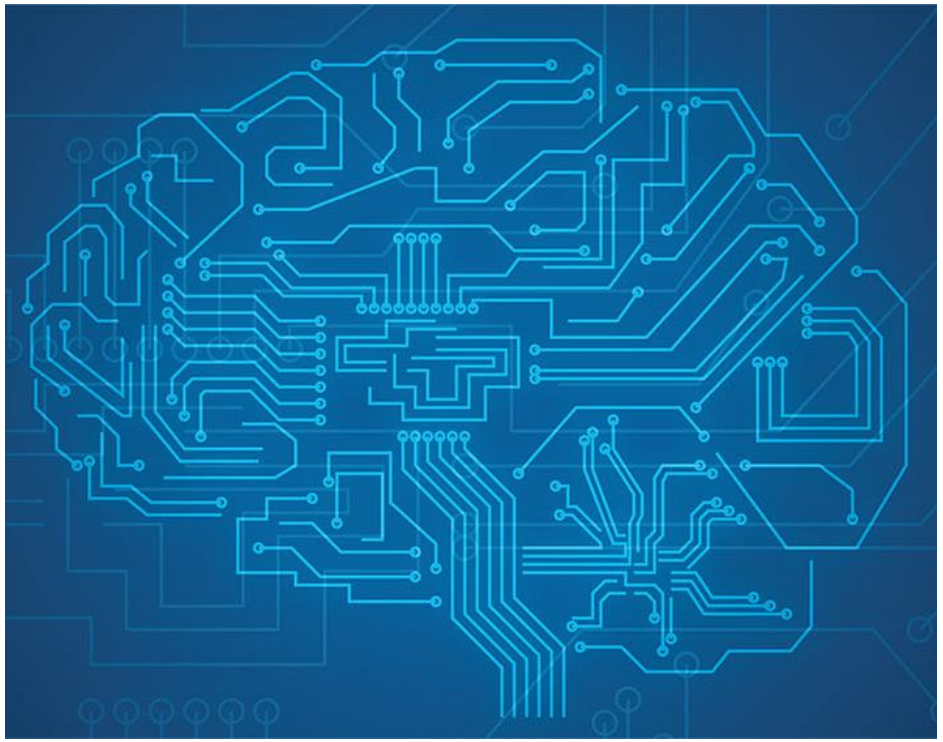

```
spatial_for (n2=0; n2<N2; n2++) {
  // Model Parallelism
    mvin(W[0:K,n2*N1*N0:
                (n2+1)*N1*N0]);
    for (m2=0; m2<M2; m2++) {
      mvin(IA[m2*M1:(m2+1)*M1,0:K]);
      compute_matmul(*W, *IA, *OA,
                                …
                                m2, n2);
      mvout(OA[n2*N1*N0:(n2+1)*N1*N0
              ,m2*M1:(m2+1)*M1]);
}}
```

Spatial Choice






```
spatial_for (m2=0; m2<M2; m2++) {
  // Data Parallelism
    mvin(IA[m2*M1:(m2+1)*M1,0:K]);
    for (n2=0; n2<N2; n2++) {
      mvin(W[0:K,n2*N1*N0:
             (n2+1)*N1*N0]);
      compute_matmul(*W, *IA, *OA,
                     …
                     m2, n2);
      mvout(OA[n2*N1*N0:(n2+1)*N1*N0
           ,m2*M1:(m2+1)*M1]);
}}
```



Mapping

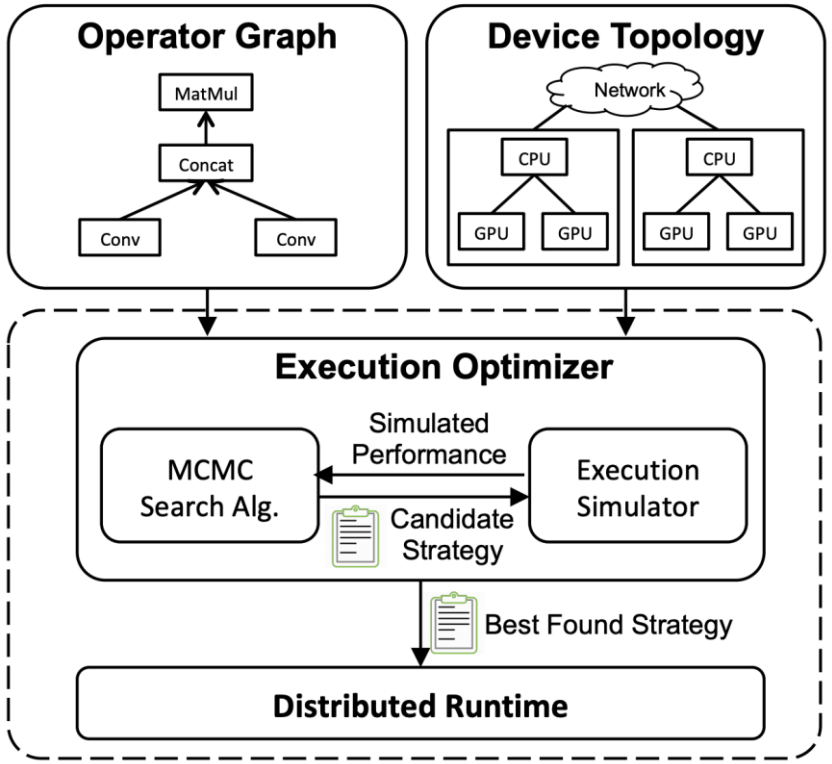
- DNN Mapping Problem:
  - Loop nest
  - HW Constraints
- Mapping space:
  - Loop ordering
  - Loop bound
  - Spatial choice
- **Tuning**

Navigating the Mapping Space

- DNN mapping problem can be viewed as an optimization problem.
- Given:
  - DNN dimensions (N, H, W, C, R, S, K, stride, padding)
  - Hardware specifications (dataflow, memory hierarchy)
- Find:
  - An optimal loop nest that minimizes latency and/or energy.
    - That defines both temporal and spatial execution order
- Approaches:
  - Exhaustive search
  - Random search
  - Learning-based algorithms

Example: FlexFlow, SysML'2018

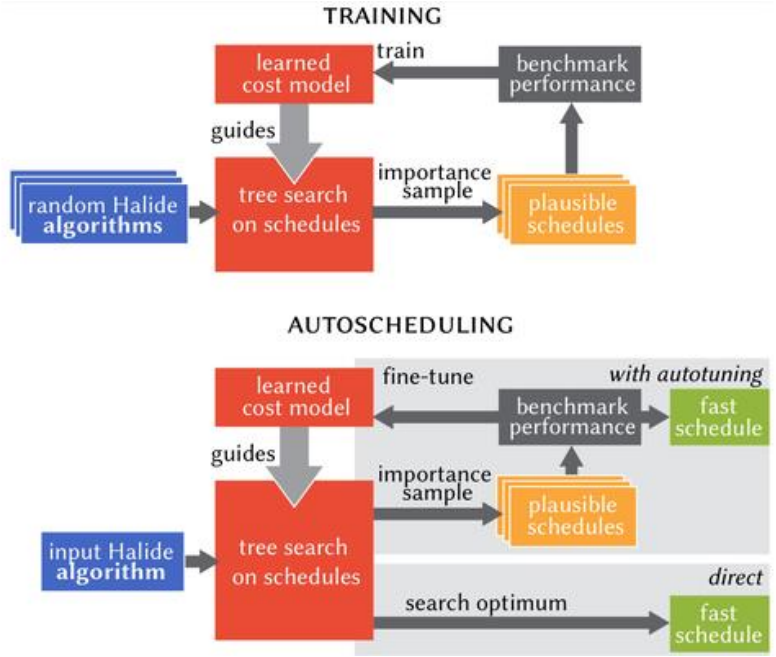
- "The optimizer uses a MCMC search algorithm to explore the space of possible parallelization strategies and iteratively proposes candidate strategies that are evaluated by an execution simulator. "



Beyond Data and Model Parallelism for Deep Neural Networks, SysML 2018

Example: Halide, SIGGRAPH'2019

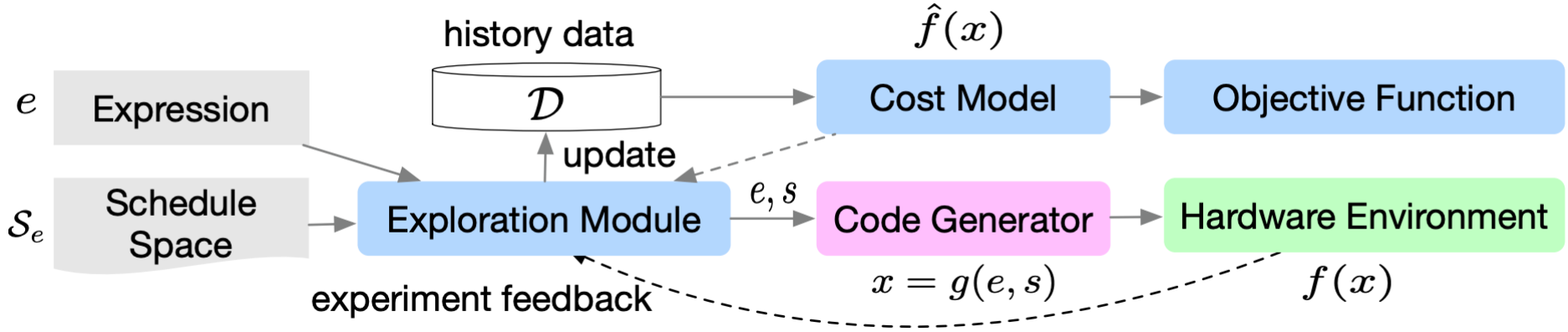
- "We generate schedules for Halide programs using tree search over the space of schedules guided by a learned cost model and optional autotuning. The cost model is trained by benchmarking thousands of randomly-generated Halide programs and schedules. The resulting code significantly outperforms prior work and human experts."



Learning to Optimize Halide with Tree Search and Random Programs, *SIGGRAPH 2019*

Example: TVM, NeurIPS'2018

- "We learn domain-specific statistical cost models to guide the search of tensor operator implementations over billions of possible program variants. We further accelerate the search using effective model transfer across workloads."



Learning to Optimize Tensor Programs, *NeurIPS'2018*

Review

- Core computation in DNN
- Execution order of the core computation
- Hardware realization of the core computation
- This lecture: mapping DNNs to hardware
  - Temporal and Spatial Mapping based on hardware constraints.
    - Memory hierarchy
    - Parallelism
  - Tile the loops to improve reuse and parallelism
    - Loop ordering
    - Loop bounds
    - Spatial choices
  - Navigate the large mapping space