Memory Management
&
System Design

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html



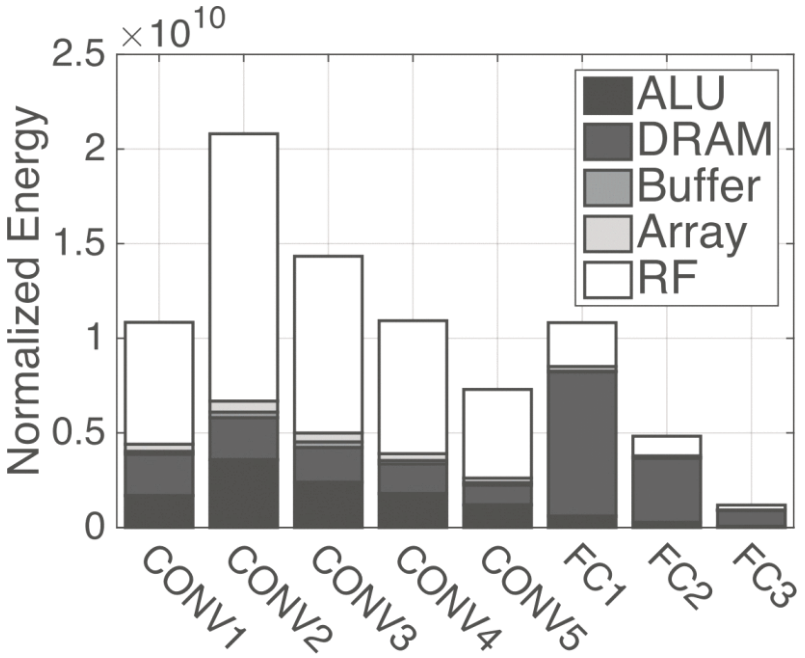
Computer Architecture - EE219

faculty.sist.shanghaitech.edu.cn

立志成才报国裕民

Memory Considerations

Memory	
Cache	(64 bit)
8 KB	10 pJ
32 KB	20 pJ
1 MB	100 pJ
DRAM	1.3-2.6 nJ
FP	
FADD	
16 bit	0.4 pJ
FMULTI	
16 bit	1 pJ

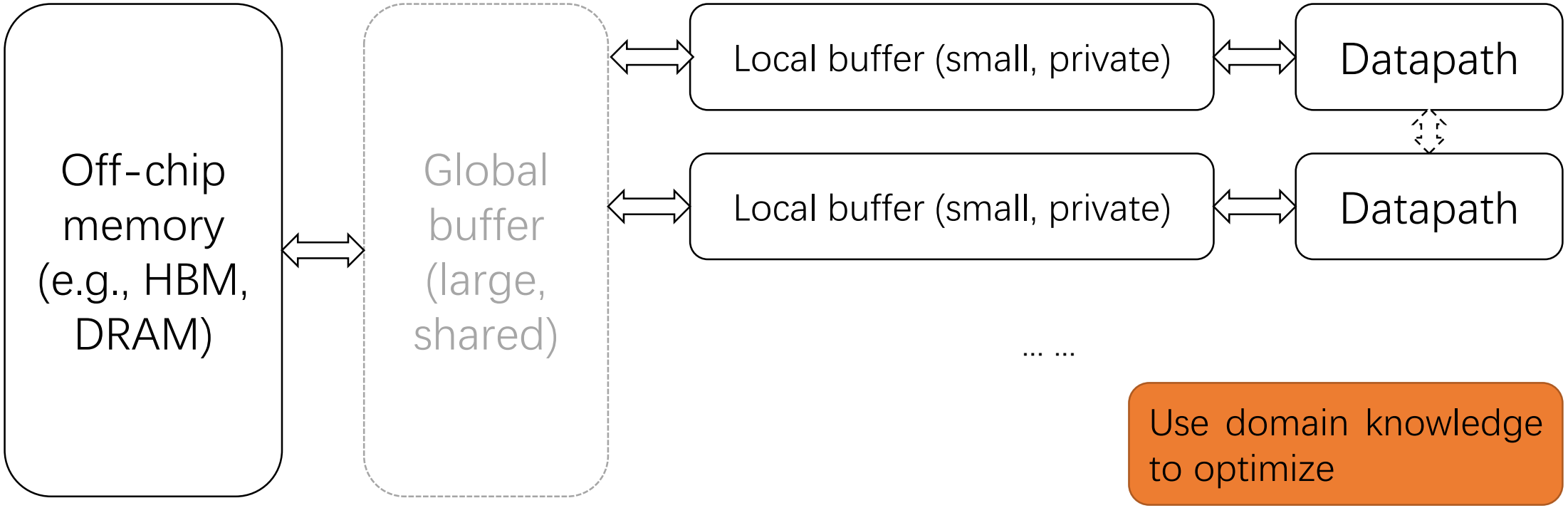


On-chip memory takes generally more than 40% of chip area.

Y. -H. Chen, J. Emer and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in 2016 ISCA.

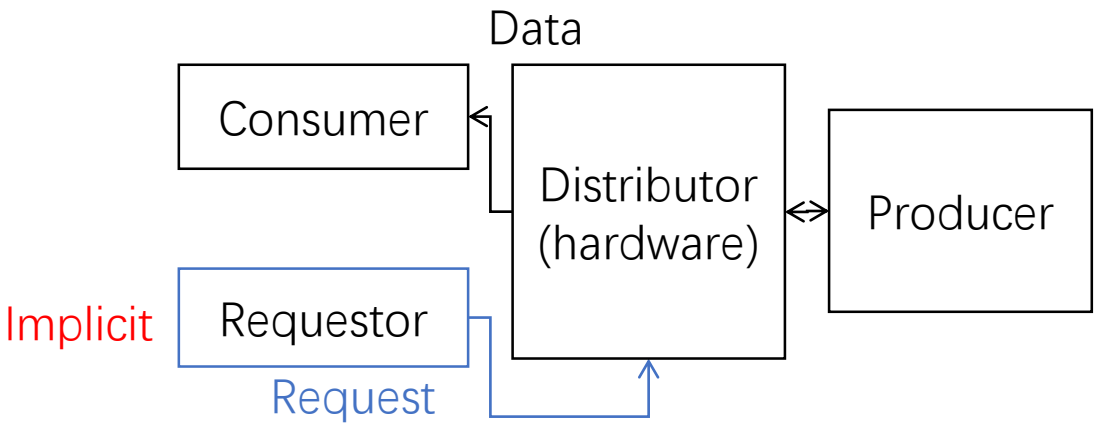
Ideally, …

- Data is ready when the datapath (functional unit or PE) needs it
- Removing data from a buffer when it is not used
- Reduce data movement as much as possible



Old Wisdom

- Cache to coordinates data movements



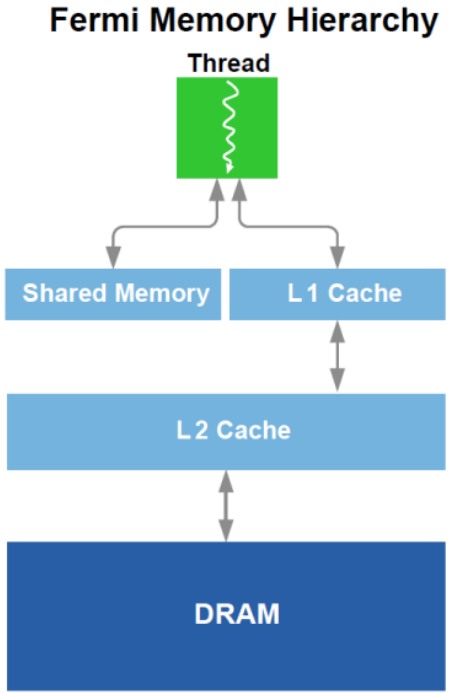
- Easy to program
- Workload agnostic
- Area/energy overhead

- Cache controller receives request and sends it to main memory if necessary.
- Cache provides data to ALU/datapath

立志成才报国裕民

New Wisdom

- Explicitly-managed memory, e.g., shared memory in NVIDIA GPUs
- In Fermi architecture, 64 KB configurable shared memory and L1 cache
- **cudaDeviceSetCacheConfig()**
- **cudaFuncSetCacheConfig()**
  - 48 KB shared memory with 16 KB L1 cache
  - 16 KB shared memory with 48 KB L1 cache
  - Shared memory: memory accesses are known beforehand
  - Synchronize threads by **__syncthreads()**



立志成才报国裕民

Use of Shared Memory in GPU

```
#include <stdio.h>

__global__ void staticReverse(int *d, int
n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
   __syncthreads();
  d[t] = s[tr];
}
```

```
int main(void)
{
  const int n = 64; int a[n], r[n], d[n];
  for (int i = 0; i < n; i++) {
    a[i] = i;
    r[i] = n-i-1;
    d[i] = 0;
  }
  int *d_d; cudaMalloc(&d_d, n * sizeof(int));
  // run version with static shared memory
  cudaMemcpy(d_d, a, n*sizeof(int), cudaMemcpyHostToDevice);
  staticReverse<<<1,n>>>(d_d, n);
  cudaMemcpy(d, d_d, n*sizeof(int), cudaMemcpyDeviceToHost);
  for (int i = 0; i < n; i++)
    if (d[i] != r[i]) printf("Error: d[%d]!=r[%d] (%d, %d)/n", i, i, d[i], r[i]);
}
```

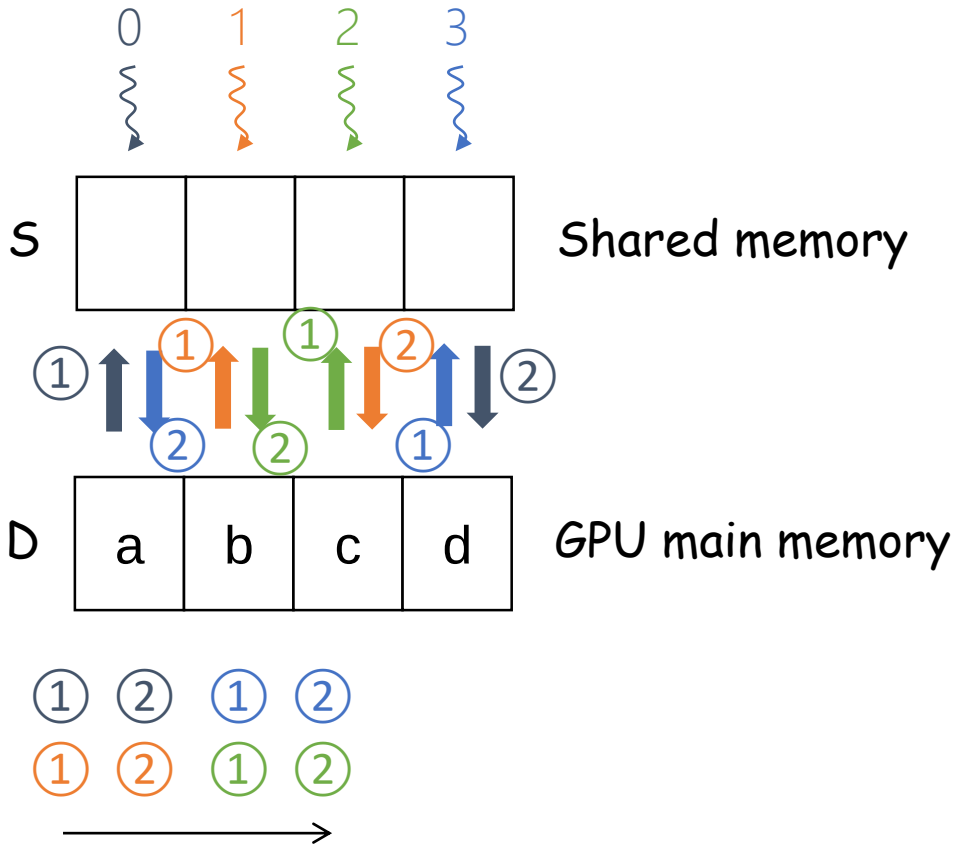
https://github.com/NVIDIA-developer-blog/code-samples/blob/master/series/cuda-cpp/shared-memory/shared-memory.cu

立志成才报国裕民

Manually Synchronize

```
#include <stdio.h>

__global__ void staticReverse(int *d, int
n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
   __syncthreads();
  d[t] = s[tr];
}
```

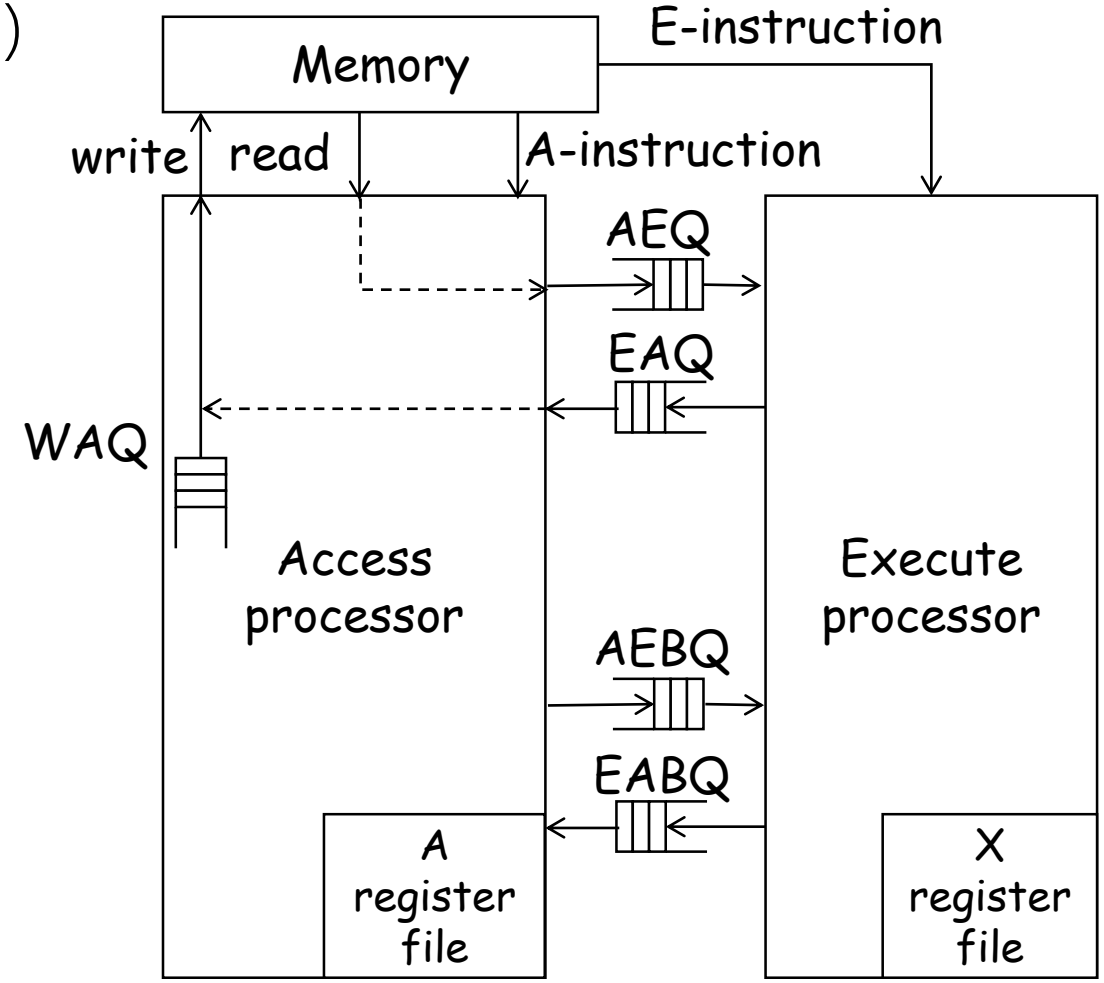


https://github.com/NVIDIA-developer-blog/code-samples/blob/master/series/cuda-cpp/shared-memory/shared-memory.cu

立志成才报国裕民

Decoupled Access-Execute (DAE)

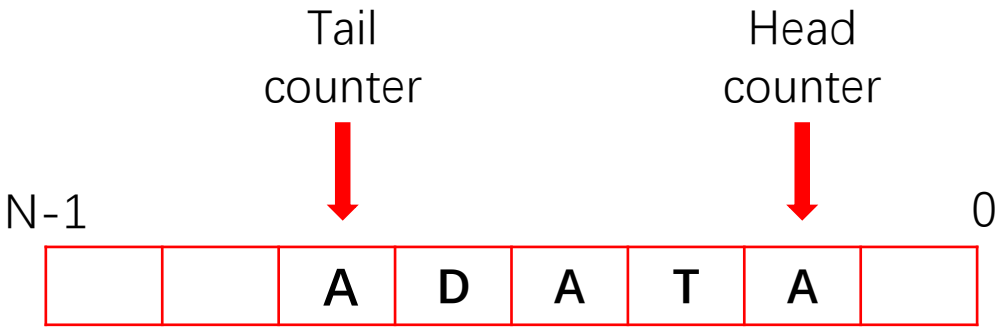
- Decoupled access-execute (DAE, old-fashion)
  - Separate access/execute processor
  - Connected with FIFO queues
  - If data loading is always ahead of execution, zero access time
  - Separate data requestor/consumer
- Reduce the complexity of instruction scheduling (programmer/compiler) at relatively low hardware overhead



Smith, J.E., 1982, April. Decoupled access/execute computer architectures. Proceedings of the 11th International Symposium on Computer Architecture.

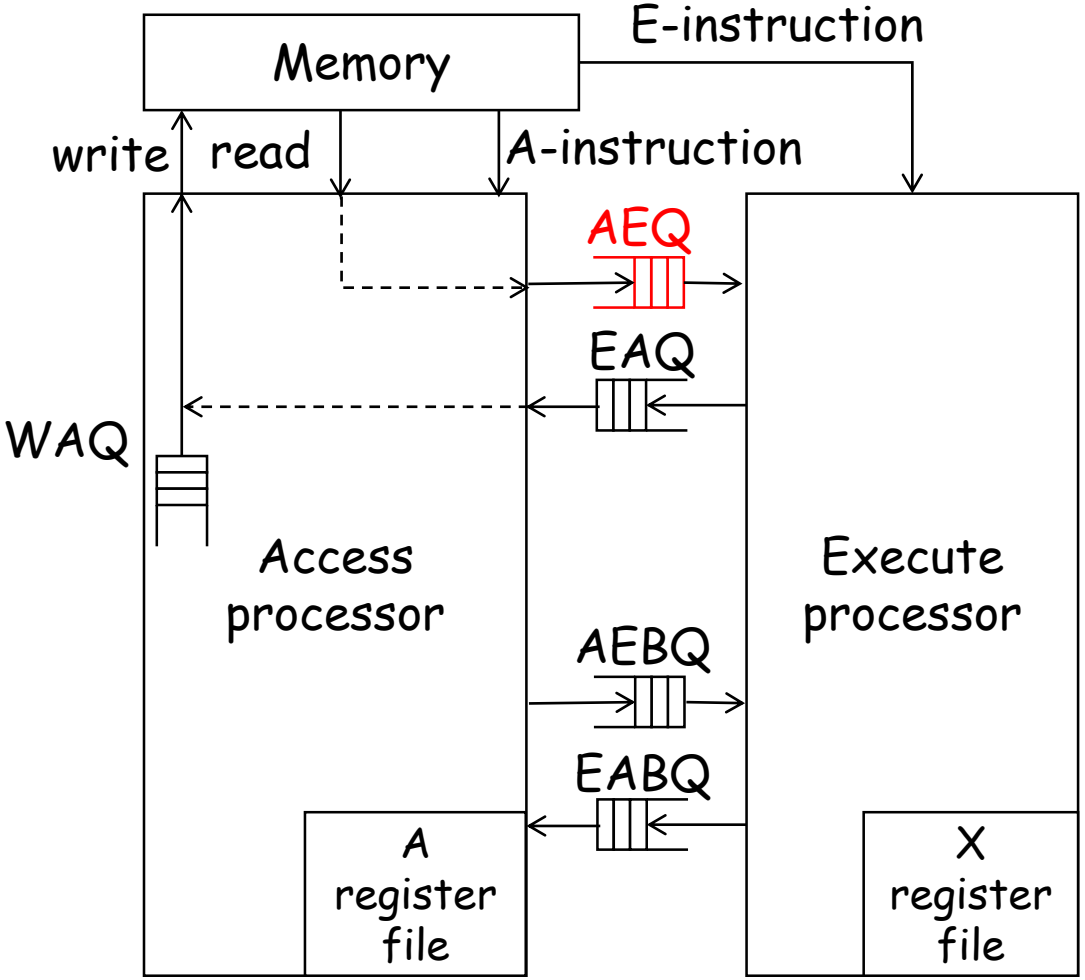
立志成才报国裕民

FIFO Queue



AEQ implemented by a circular buffer

- On a read from E-processor, increase head counter
- On a fill from A-processor, increase tail counter
- When **head counter == tail counter**, queue empty
- When **head counter == tail counter + 1**, queue full



Smith, J.E., 1982, April. Decoupled access/execute computer architectures. Proceedings of the 11th International Symposium on Computer Architecture.

立志成才报国裕民

Decoupled Access-Execute (DAE)

- Example

```
q = 0.0
Do k = 1 to 400
x(k) = q + y(k) * (r * z(k+10) +  t * z(k+11))
```

Regular assembly

```
A7 <- -400          . negative loop count
A2 <- 0             . initialize index
A3 <- 1             . index increment
X2 <- r             . load loop invariants
X5 <- t             . into registers
loop: X3 <- z + 10, A2   . load z(k+10)
    X7 <- z + 11, A2     . load z(k+11)
    X4 <- X2 *f X3       . r*z(k+10)-flt. mult.
    X3 <- X5 *f X7       . t * z(k+11)
    X7 <- y, A2          . load y(k)
    X6 <- X3 +f X4       . r*z(x+10)+t*z(k+11))
    X4 <- X7 *f X6       . y(k) * (above)
    A7 <- A7 + 1         . increment loop counter
    x, A2 <- X4          . store into x(k)
    A2 <- A2 + A3        . increment index
    JAM loop             . Branch if A7 < 0
```

Assembly for DAE architecture (loop body)

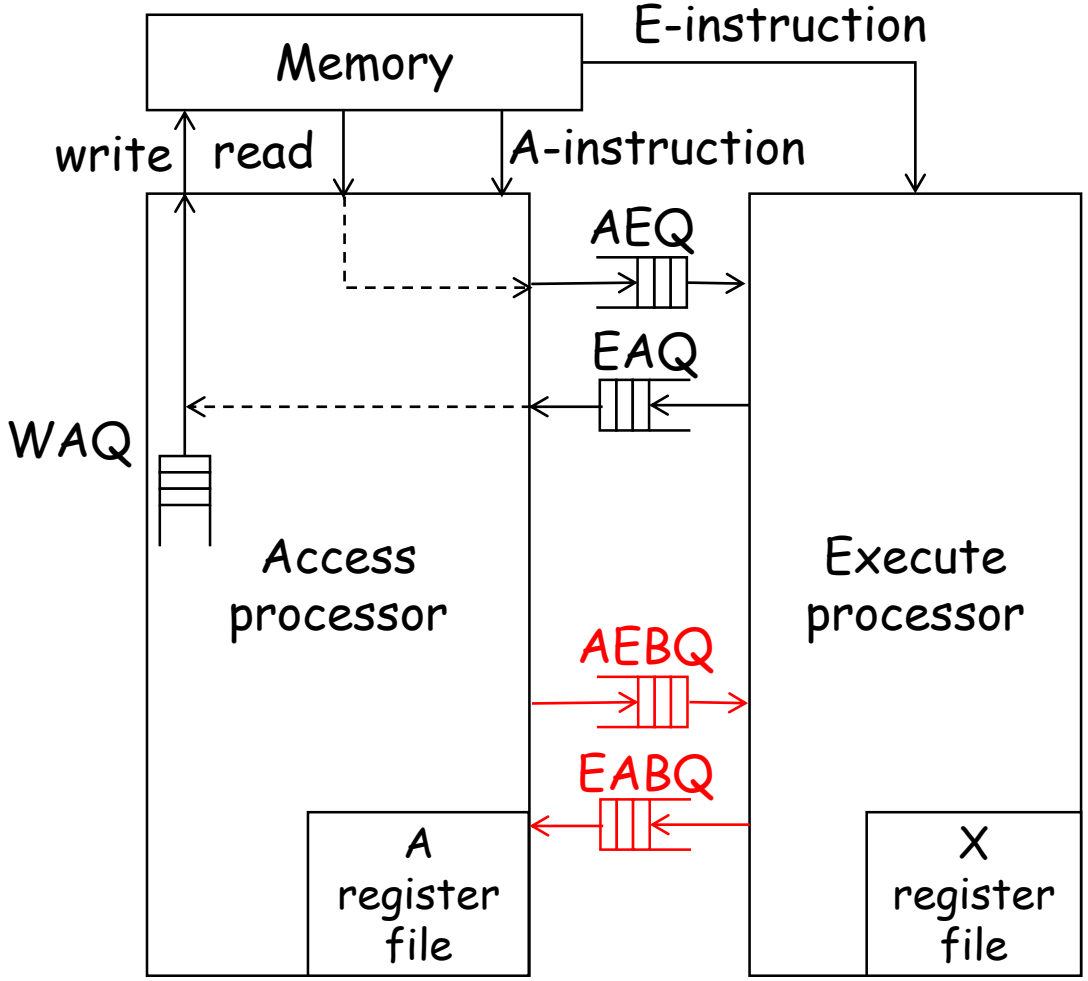
Access stream	Execute
AEQ <- z + 10, A2	X4 ← X2 *f AEQ
AEQ <- z + 11, A2	X3 ← X5 *f AEQ
AEQ <- y, A2	X6 ← X3 +f X4
A7 <- A7 + 1	EAQ ← AEQ *f X6
x, A2 <- EAQ	
A2 <- A2 + A3	

Smith, J.E., 1982, April. Decoupled access/execute computer architectures. Proceedings of the 11th International Symposium on Computer Architecture.

立志成才报国裕民

Conditional Branch

- When a processor determines a conditional branch outcome, it places it on the tail of the branch queue to the opposite processor;
- Let A-processor decides branch as many as possible to run ahead (as if pre-fetching). and AEBQ allows E-processor instruction "pre-fetch"; common in loop counter;



Smith, J.E., 1982, April. Decoupled access/execute computer architectures. Proceedings of the 11th International Symposium on Computer Architecture.

Possible Deadlock

Access stream	Execute
A4 <- EAQ	X3 <- AEQ
AEQ <- A5	EAQ <- X2

An attempted transfer from A5 to X3 and from X2 to A4

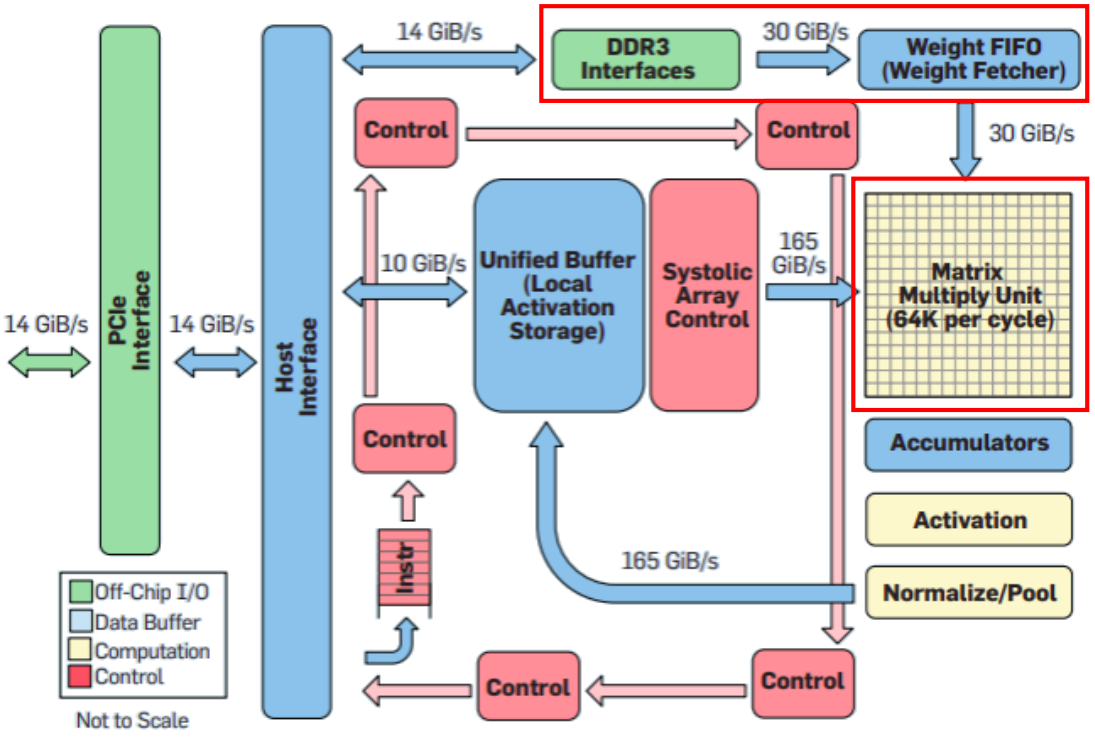


Smith, J.E., 1982, April. Decoupled access/execute computer architectures. Proceedings of the 11th International Symposium on Computer Architecture.

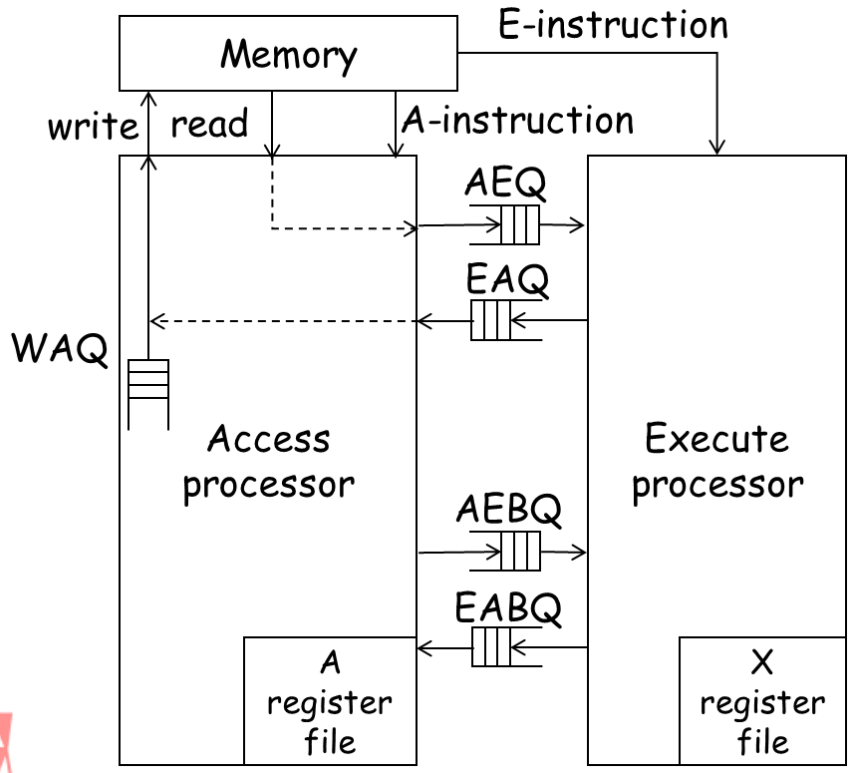
立志成才报国裕民

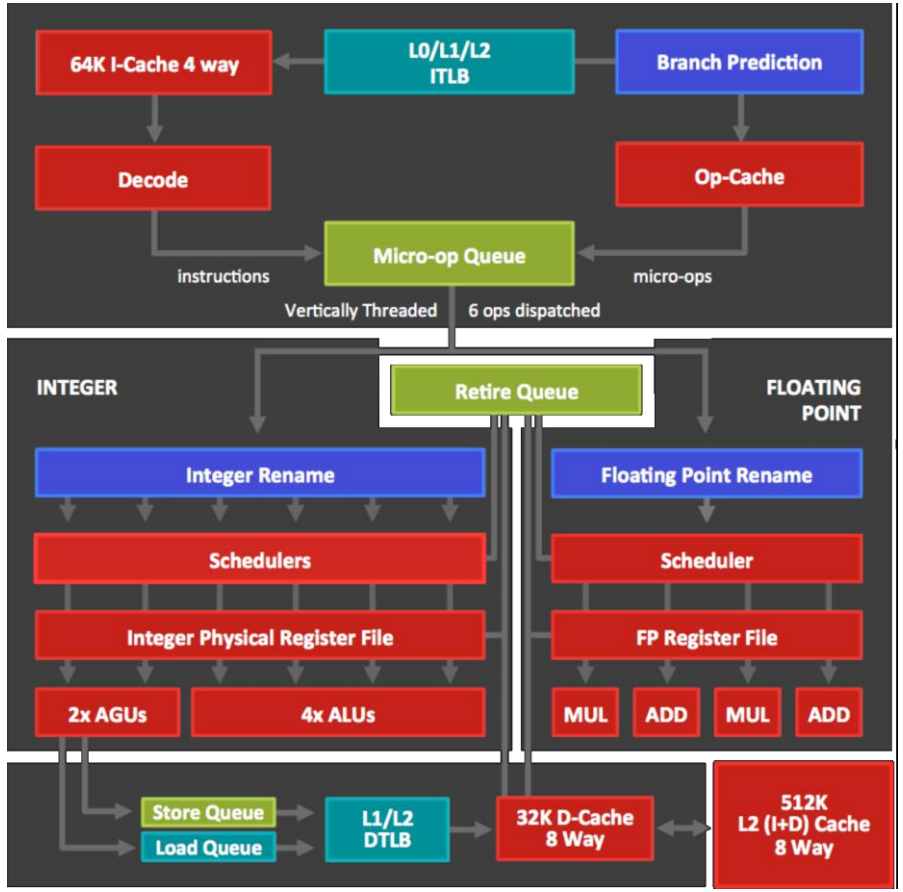
DAE Philosophy in TPUv1

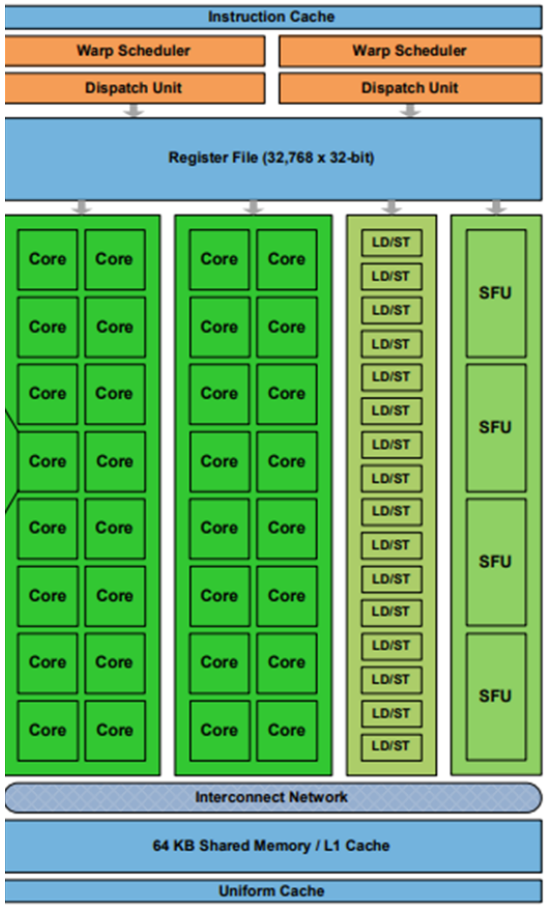
- **Read_Weights** instruction follows the decoupled-access/execute philosophy in that it can complete after sending its address but before the weight is fetched from weight memory.
- The matrix unit will stall if the input activation or weight data is not ready



DAE vs. Modern CPU & GPU

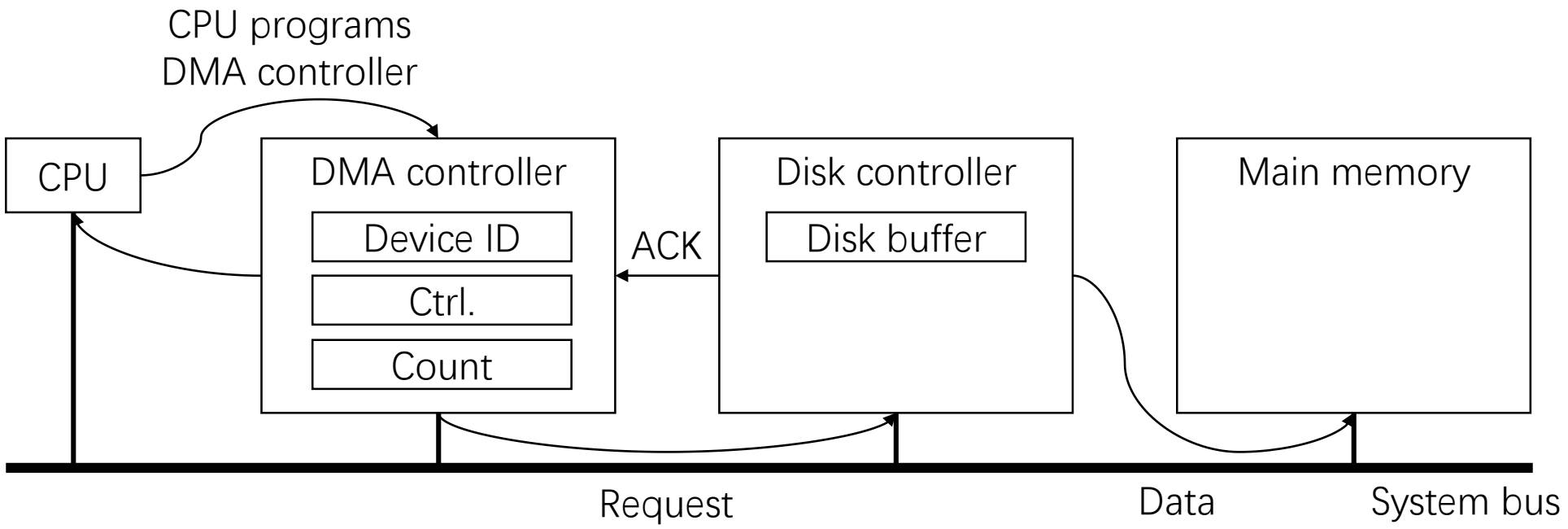




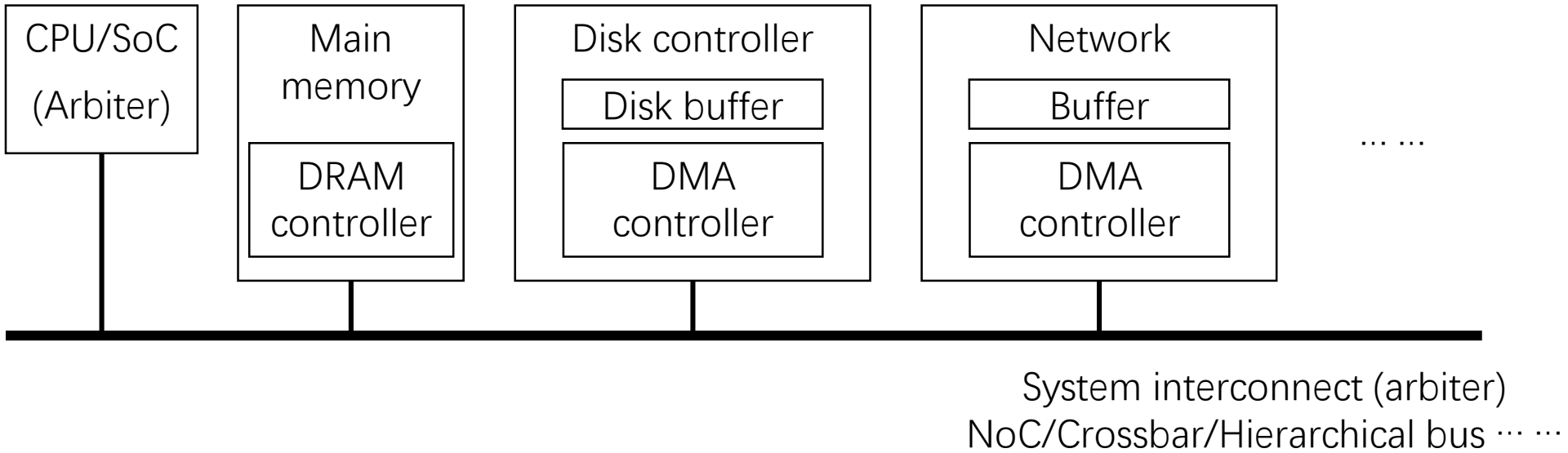


Implementation – DMA (Direct Memory Access)

- DMA in classic computer architecture: data transfer between I/O and main memory without the coordination of CPU



Modern CPU/SoC



立志成才报国裕民

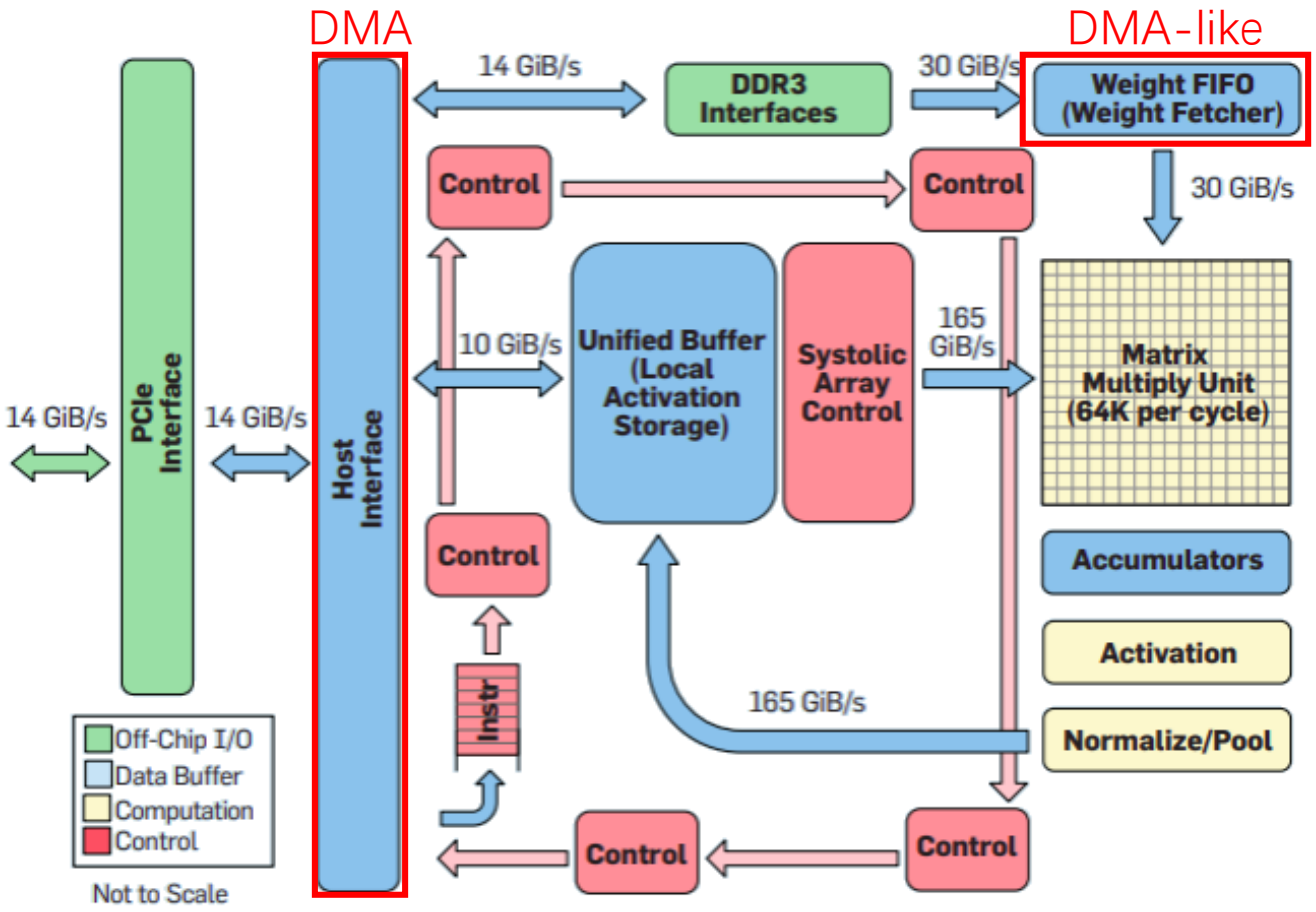
DMA Model

- dma_copy (dst = DST, src = SRC, length, mode = MODE)
- What to consider for hardware design?
- CPU vs. AI accelerator



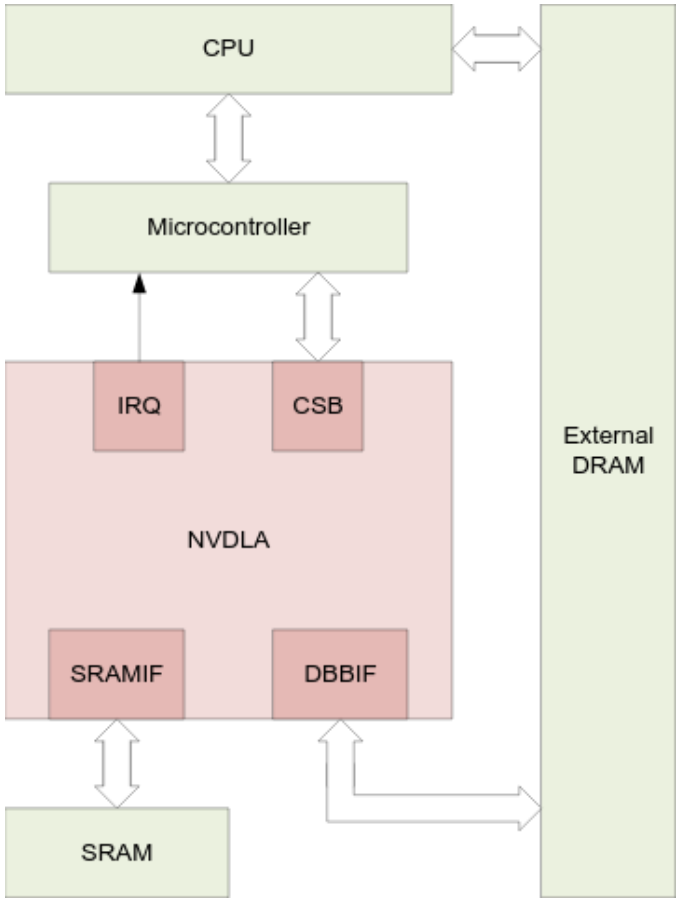
立志成才报国裕民

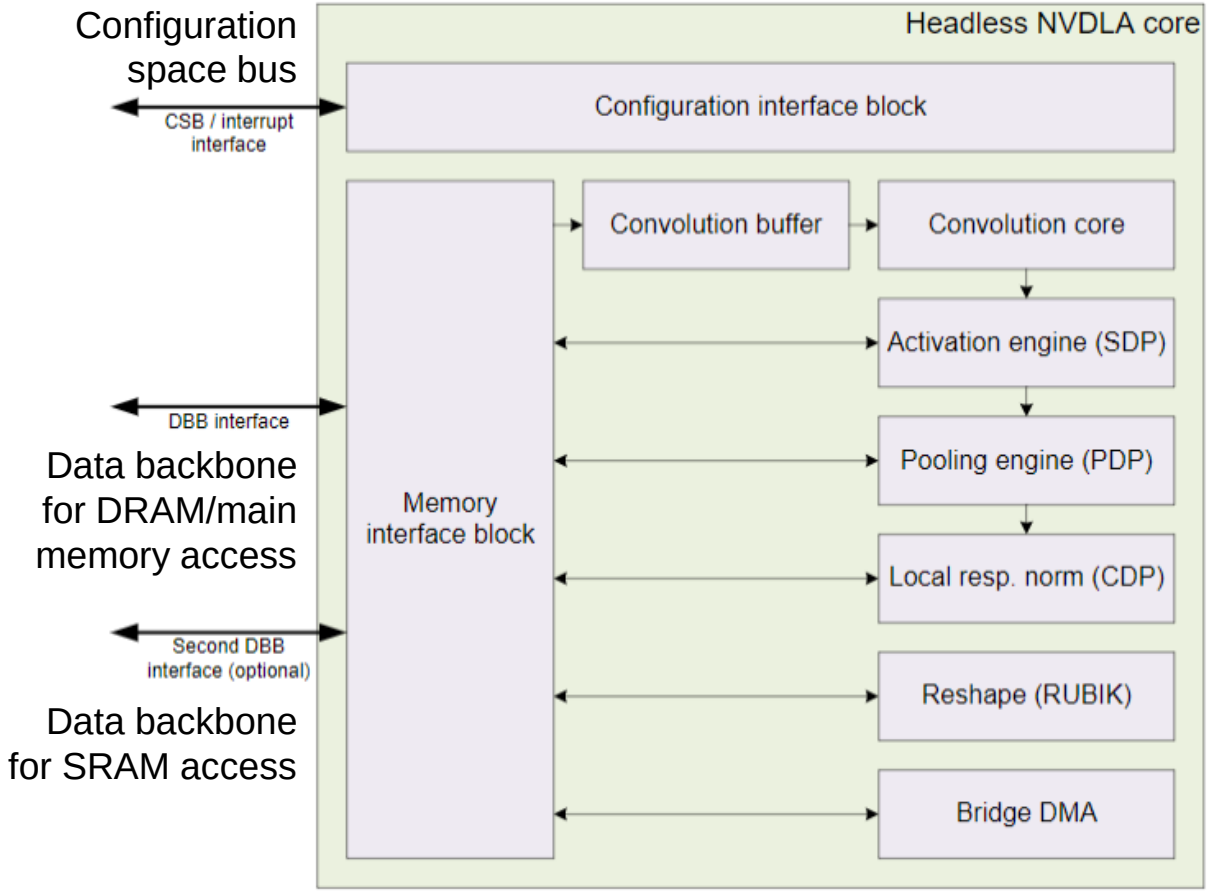
DMA in Neural Network Accelerators



DMA in Neural Network Accelerators (Cont'd)

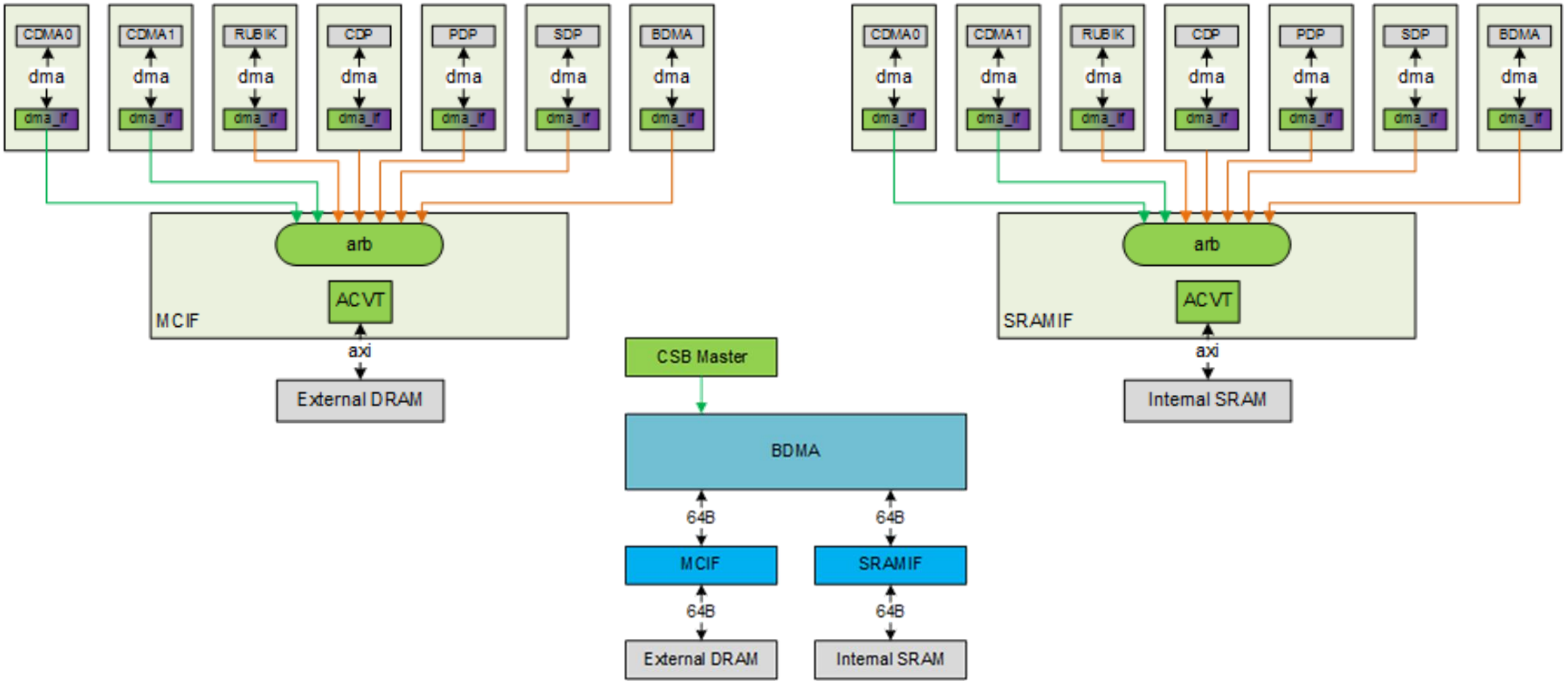
- NVDLA





NVDLA. Unit description. http://nvdla.org/hw/v1/ias/unit_description.html

NVDLA DMAs



NVDLA. Unit description. http://nvdla.org/hw/v1/ias/unit_description.html

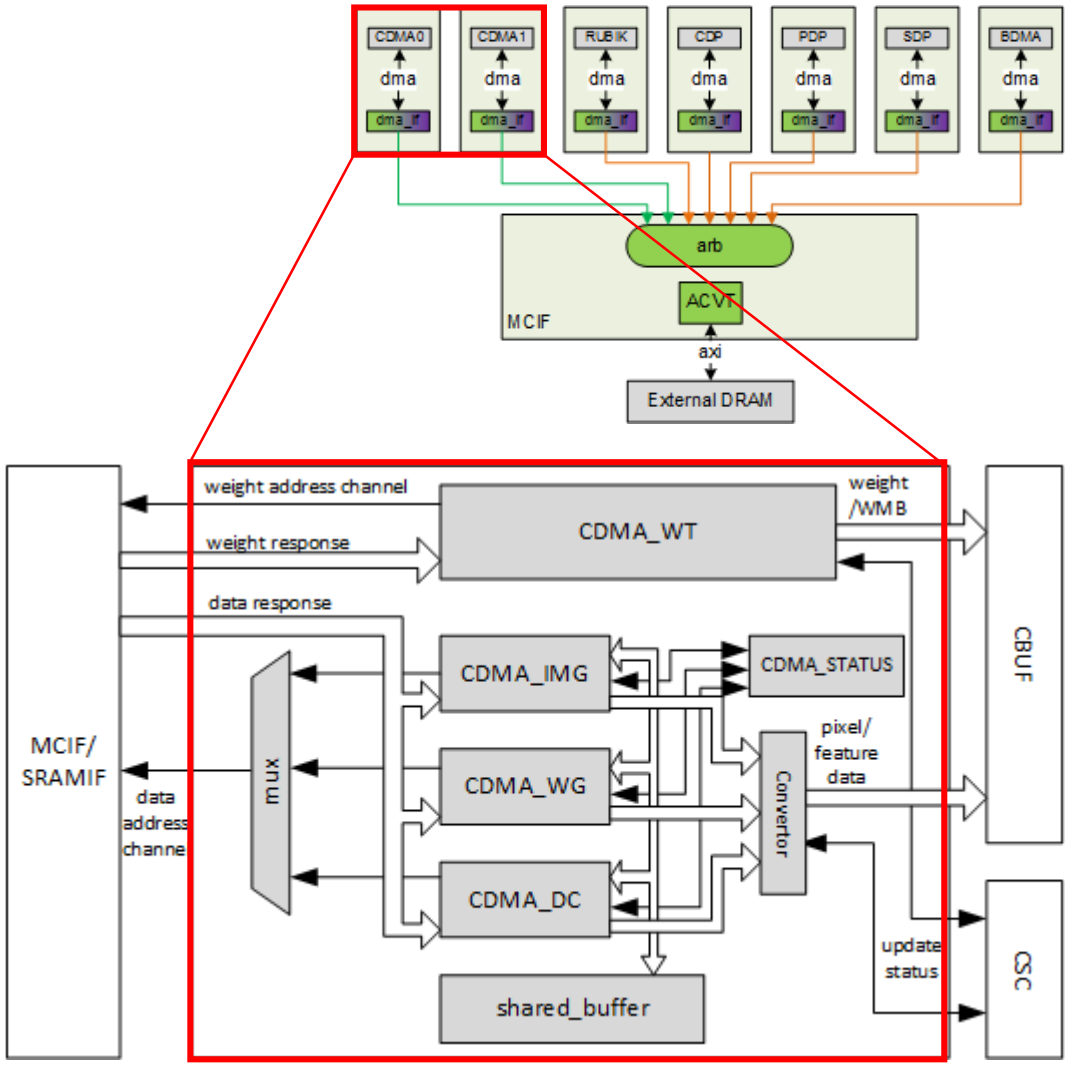
立志成才报国裕民

CDMA as an Example

- CDMA_DC (direct convolution) as example
  - Check status of convolution buffer for enough free space.
  - Generate read transactions
  - Cache feature data in shared buffers
  - Reshape feature cubes into proper order
  - Generate convolution buffer write address
  - Write feature data into convolution buffer
  - Update status of convolution buffer in the CDMA_STATUS sub-module

Convolution datapath pipeline





NVDLA. Unit description. http://nvdla.org/hw/v1/ias/unit_description.html

Takeaways

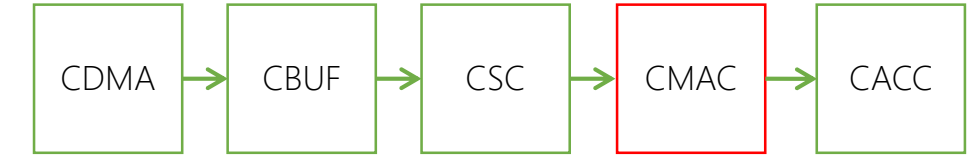
- Most AI chips adopts decoupled-access execute philosophy and DMA to simplify the hardware design;
- Since most AI algorithms are fixed at the first place, programmers or compilers are responsible for explicit memory management and beyond.
- Readings
  - Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration
  - Maia 200: A Software Defined Dataflow System for Large-scale AI Acceleration

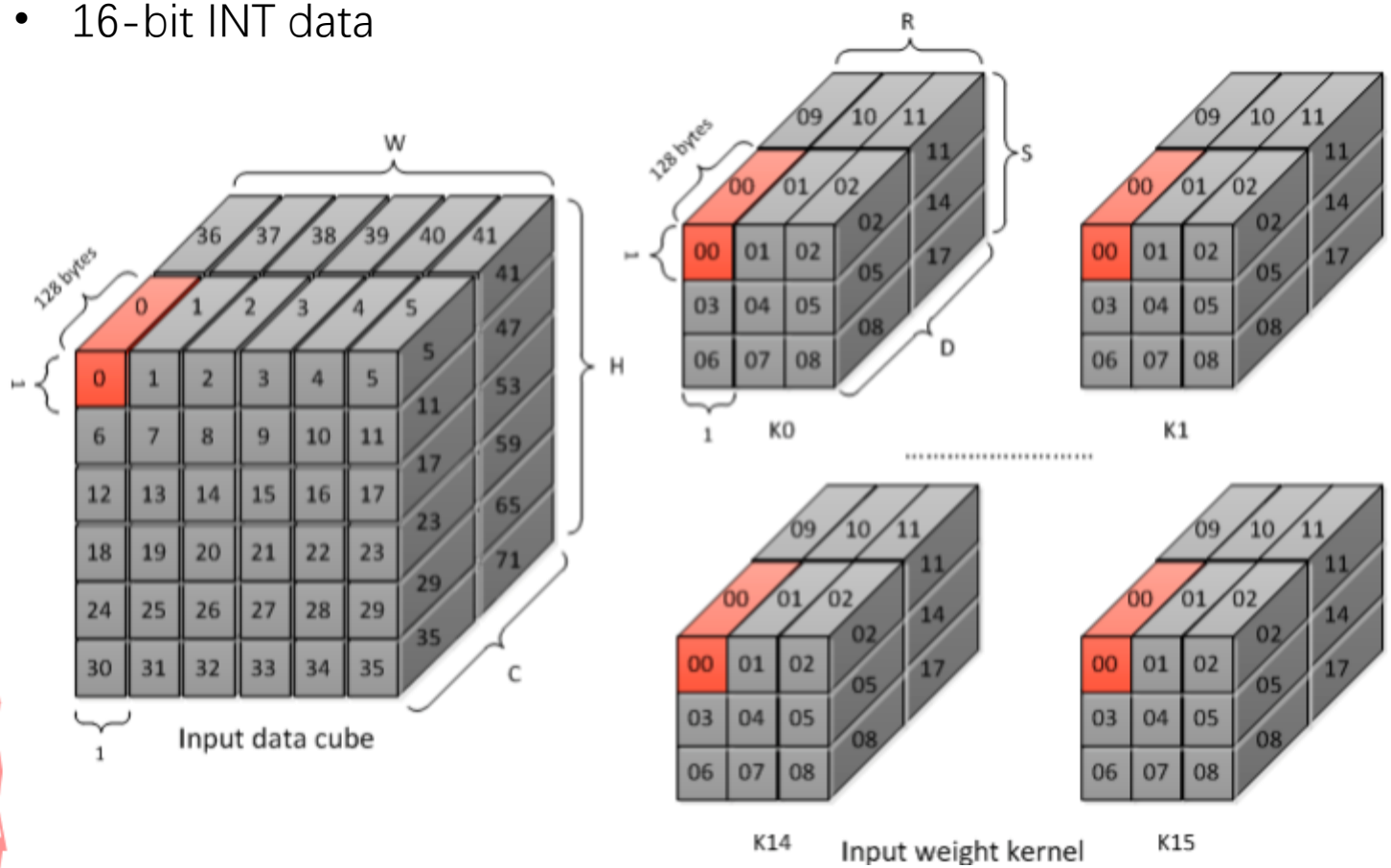


立志成才报国裕民

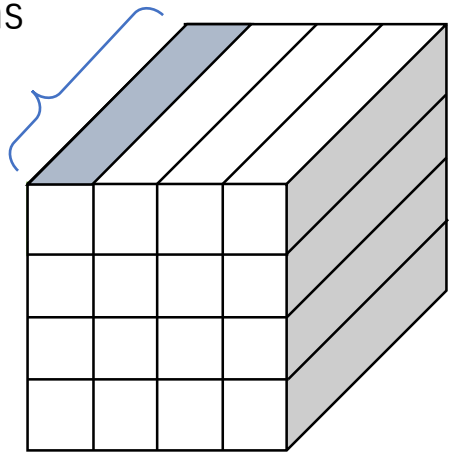
Direct Convolution

- Atomic/basic operation (not the one in CA)
- One clock cycle
- 16-bit INT data



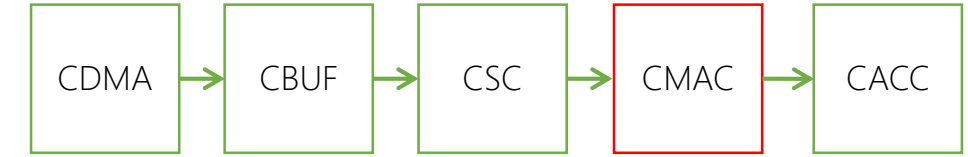


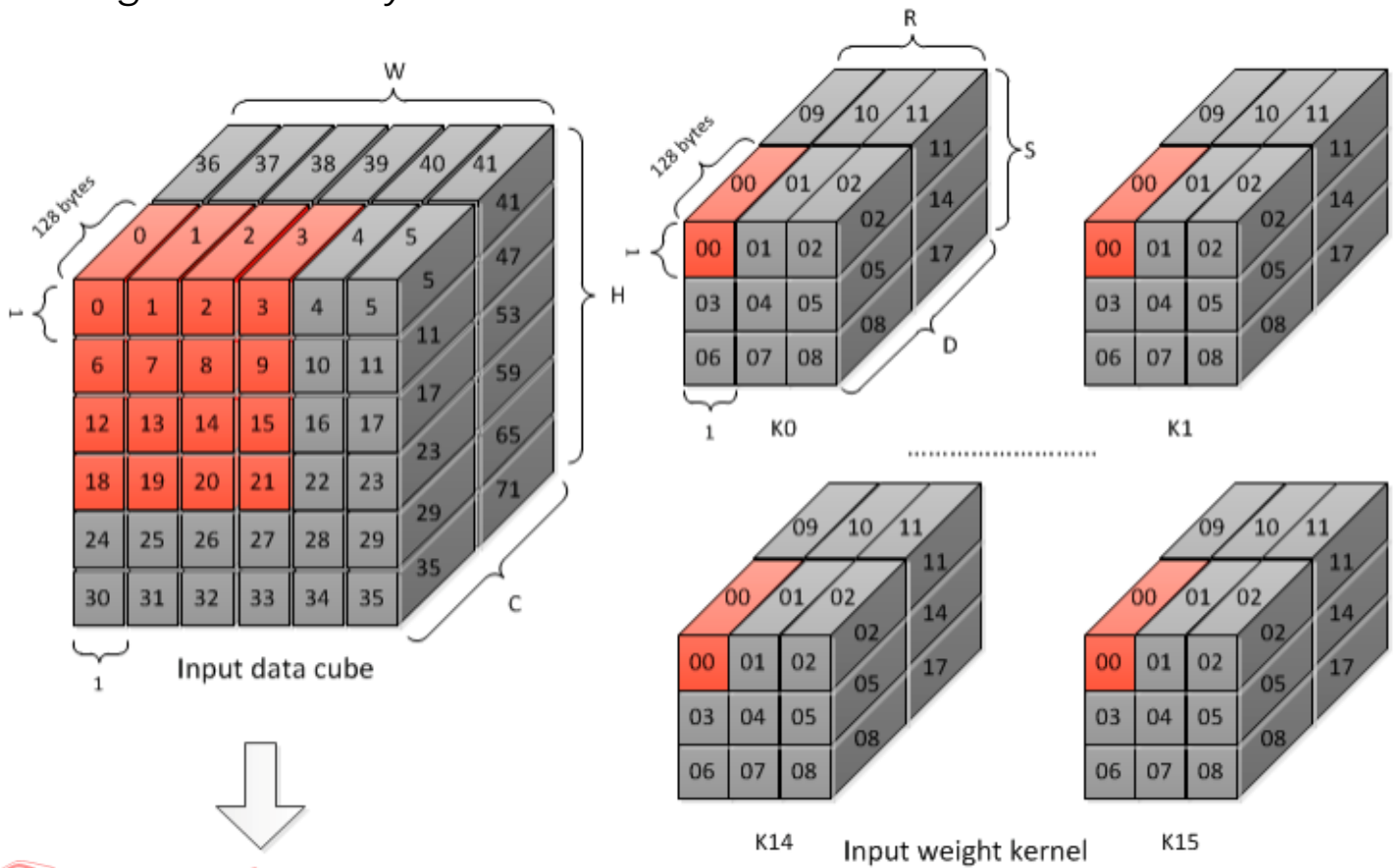
Partial sums
(32 bytes)



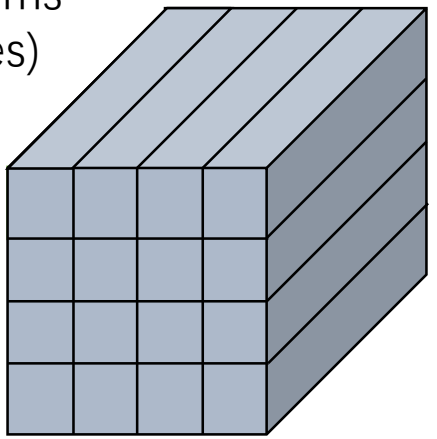
Direct Convolution (Cont'd)

- Stripe operation
- 16 clock cycles
- Weight-stationary





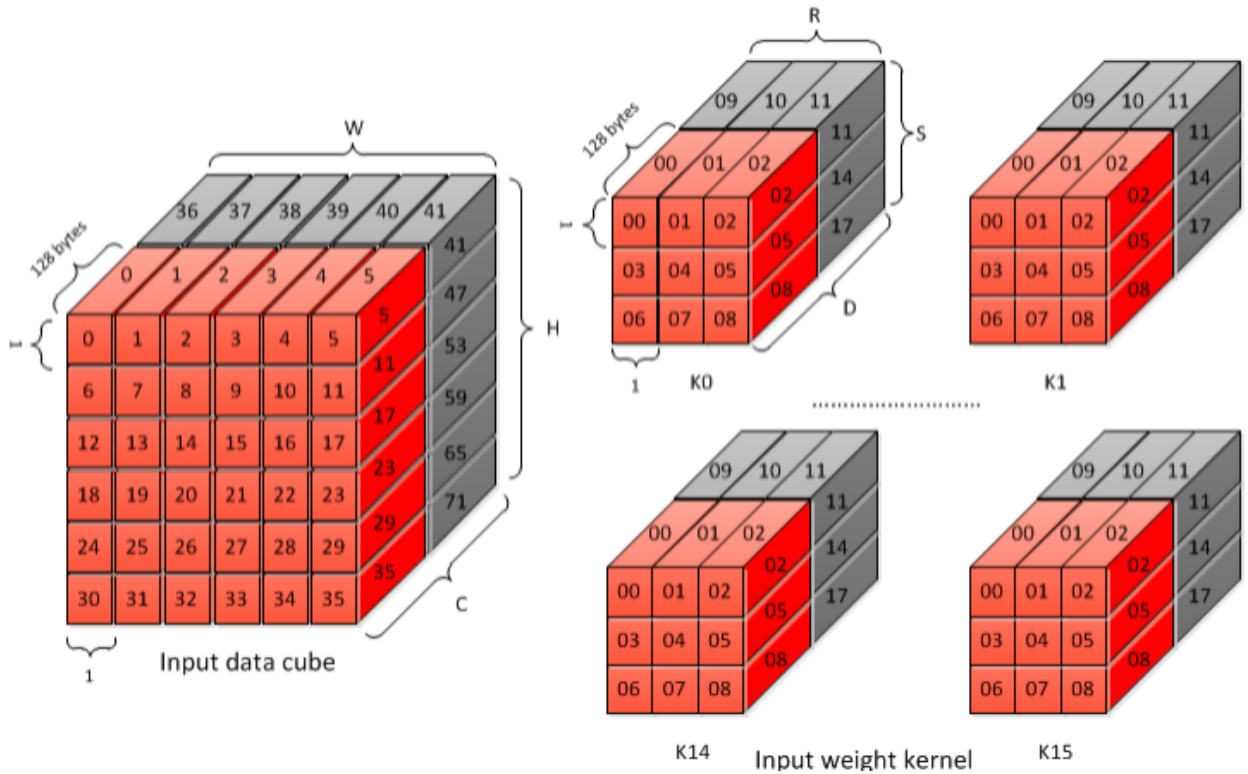
Partial sums
(32 bytes)



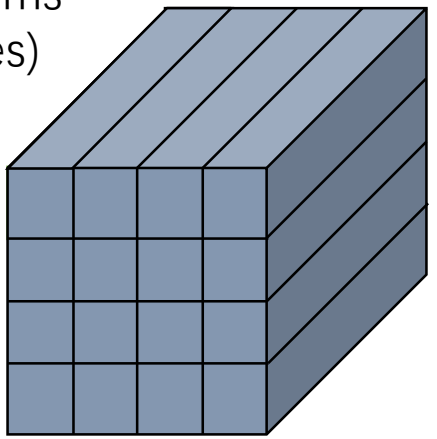
Direct Convolution (Cont'd)



- Block operation
- 16 * 9 clock cycles
- Weight-stationary



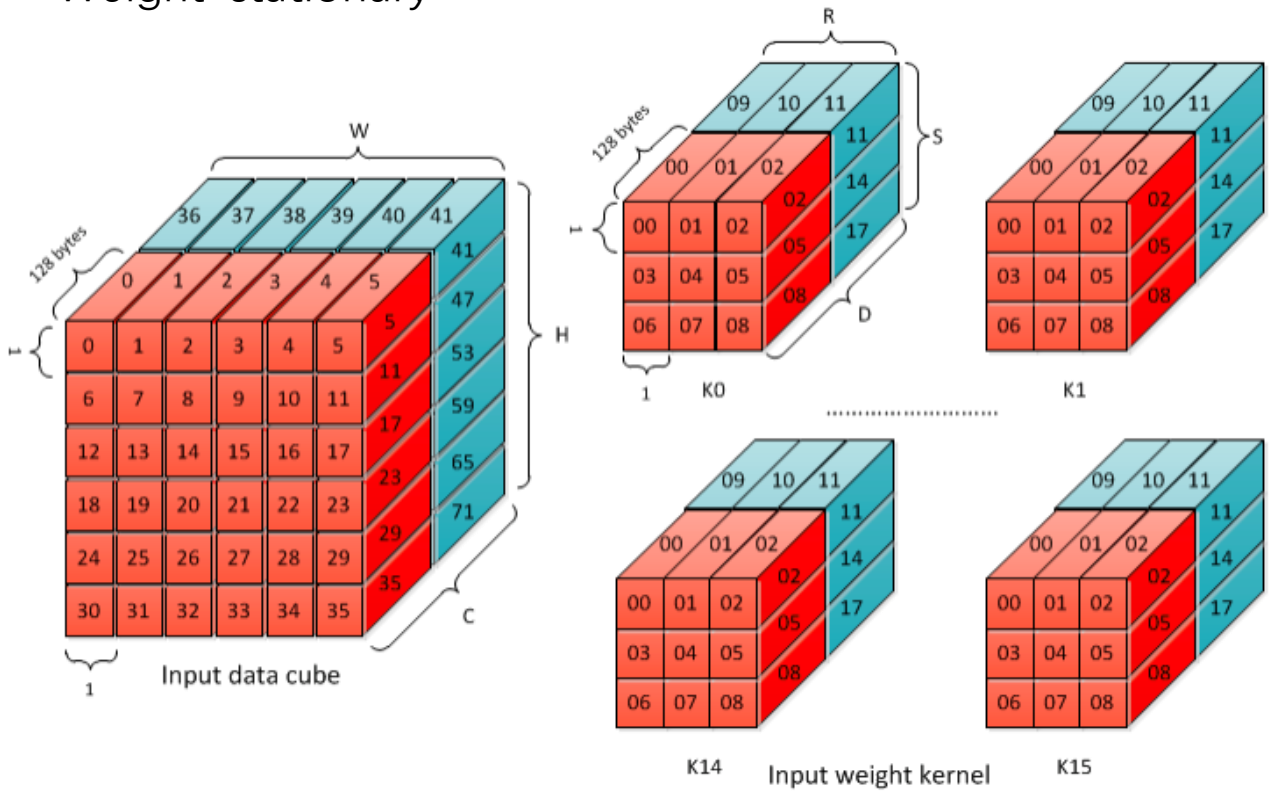
Partial sums
(32 bytes)



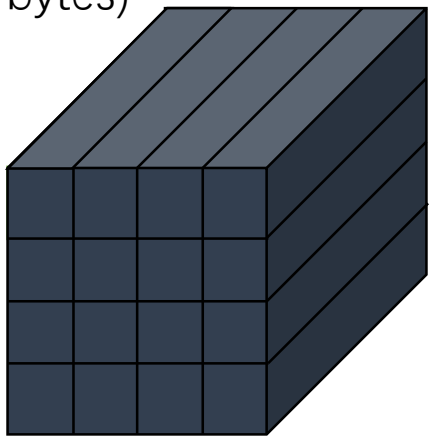
Direct Convolution (Cont'd)

- Channel operation
- 16 * 9 * 2 clock cycles
- Weight-stationary

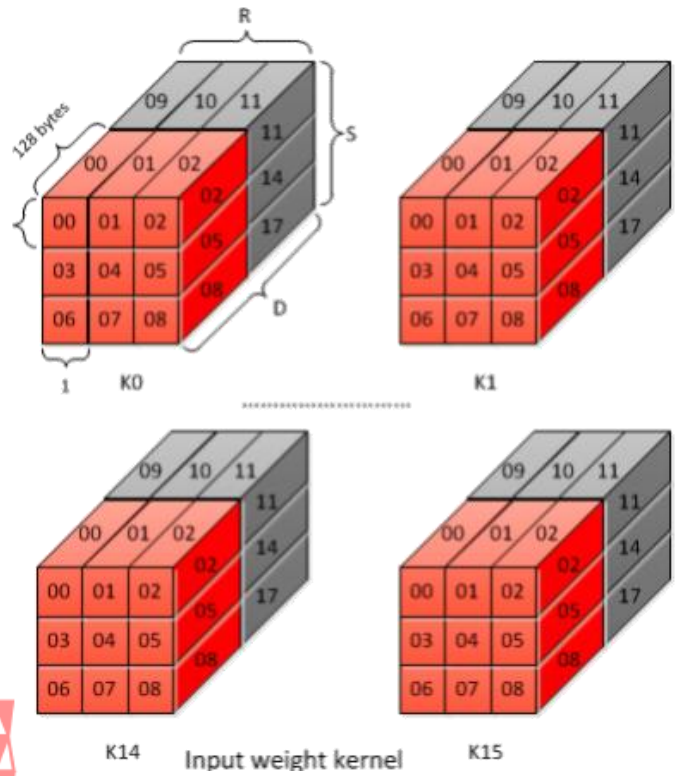


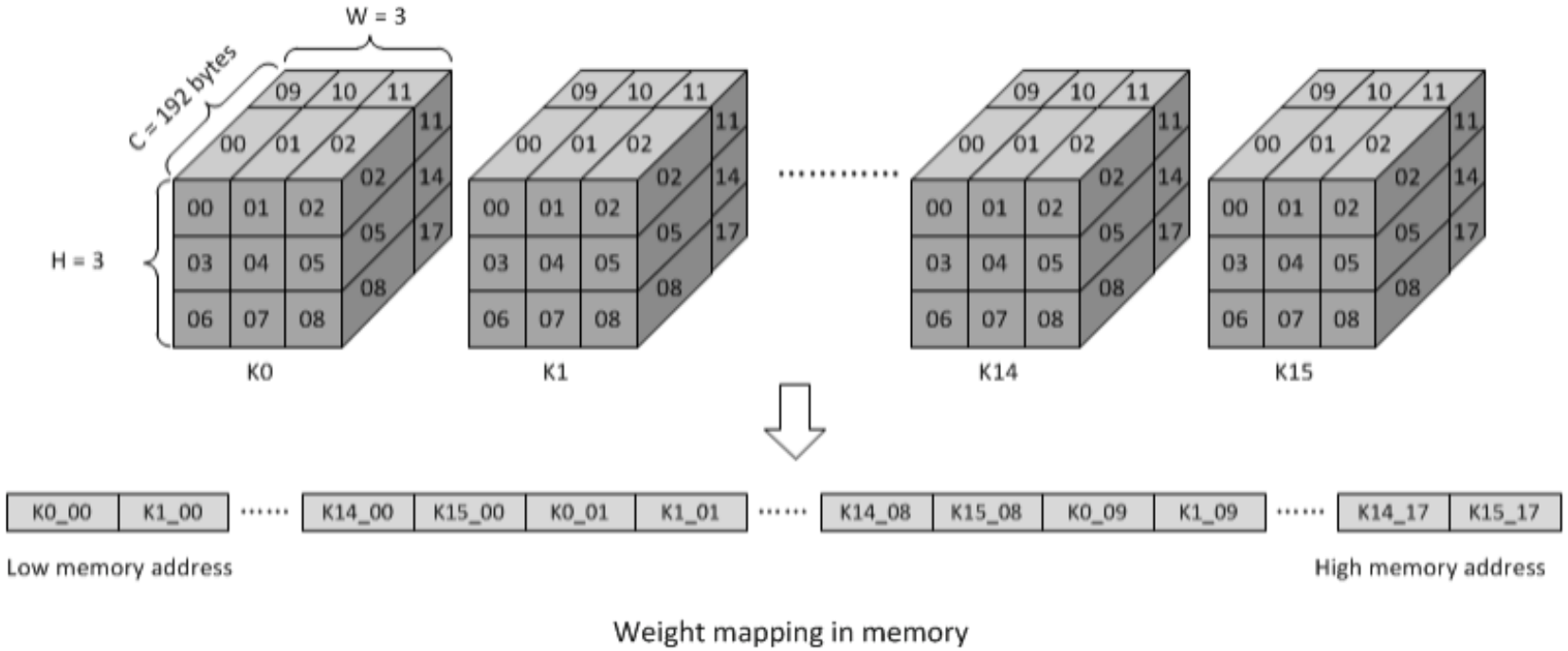


Outputs (32 bytes)



Direct Convolution – Weight Layout





Direct Convolution Output Sequence

