EE-508: Hardware Foundations for Machine Learning

Modeling Accelerators

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

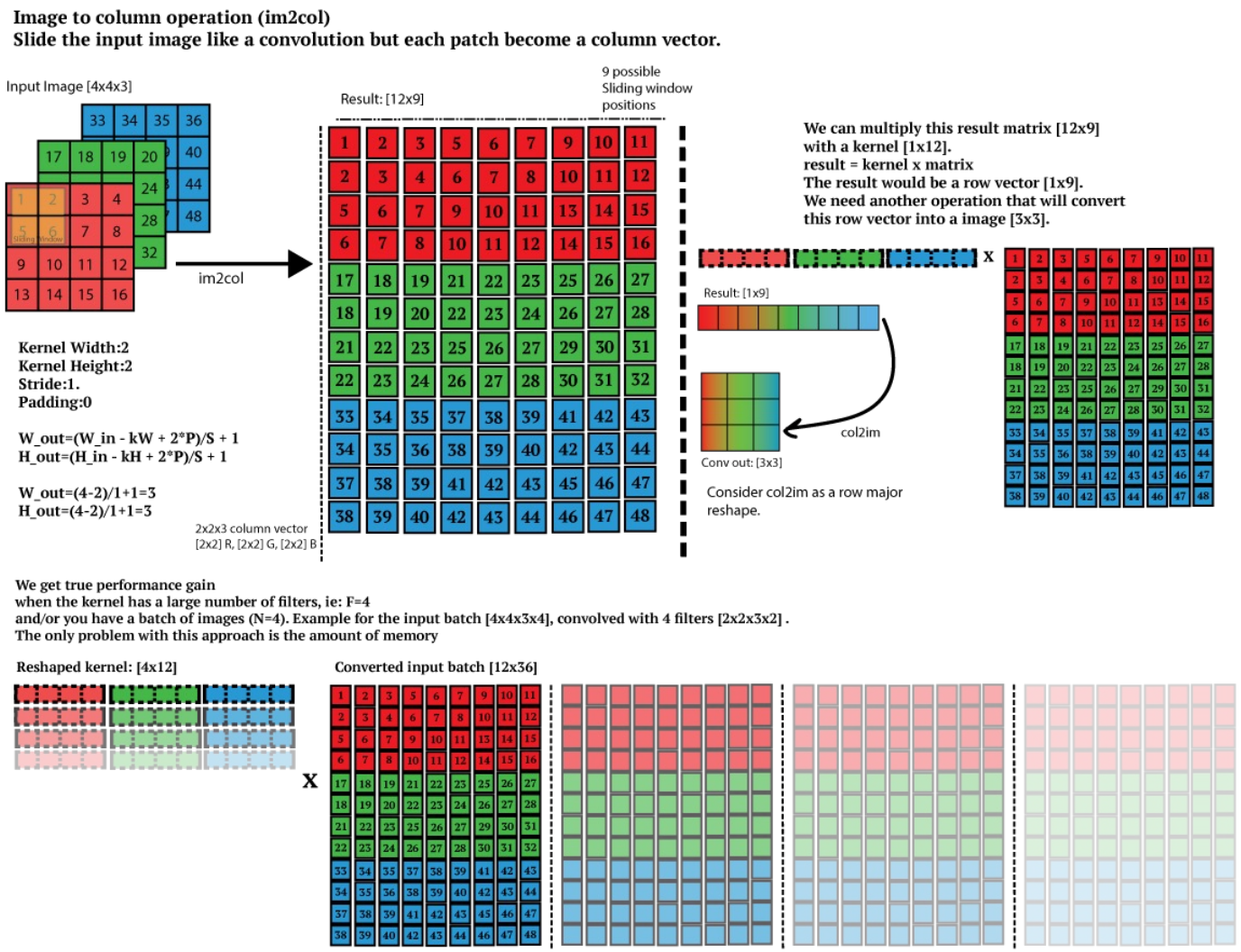
Instructors:
Arash Saifhashemi

Computation Transform Optimizations

General Matrix Multiplication

- We showed that we can map DNNs and CNNs into GEMM.
- How can we perform GEMM Optimally?

Img2Col Operation to Map Conv to GEMM



Source: leonardoaraujosantos.gitbook.io/artificial-inteligence

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) + (c + di)$$

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) \times (c + di)$$

$$z_1 \times z_2 = (ac - bd) + (ad + bc)i$$

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) \times (c + di)$$

$$z_1 \times z_2 = (ac - bd) + (ad + bc)i$$

4 Multiplications
3 Additions

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) \times (c + di)$$

$$z_1 \times z_2 = (ac - bd) + (ad + bc)i$$

4 Multiplications
3 Additions

$$k_1 = (a + b)(c + d)$$
$$k_2 = ac$$
$$k_3 = bd$$

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) \times (c + di)$$

$$z_1 \times z_2 = (ac - bd) + (ad + bc)i$$

4 Multiplications
3 Additions

$$k_1 = (a + b)(c + d)$$
$$k_2 = ac$$
$$k_3 = bd$$

$$z_1 \times z_2 = (k_2 - k_3) + (k_1 - k_2 - k_3)i$$

Gauss's complex multiplication

$$z_1 = a + bi$$
$$z_2 = c + di$$

$$z_1 \times z_2 = (a + bi) \times (c + di)$$

$$z_1 \times z_2 = (ac - bd) + (ad + bc)i$$

4 Multiplications
3 Additions

$$k_1 = (a + b)(c + d)$$
$$k_2 = ac$$
$$k_3 = bd$$

3 Multiplications 😃
6 Additions

$$z_1 \times z_2 = (k_2 - k_3) + (k_1 - k_2 - k_3)i$$

Strassen’s Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

$$\boldsymbol{A} \times \boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & df + dh \end{bmatrix}$$

8 Multiplications
4 Additions

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$A \times B = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \times \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right] = \left[\begin{array}{c|c} ae + bg & af + bh \\ \hline ce + dg & df + dh \end{array}\right]$$

8 Multiplications
4 Additions

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$A \times B = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \times \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right] = \left[\begin{array}{c|c} ae + bg & af + bh \\ \hline ce + dg & df + dh \end{array}\right]$$

8 Multiplications
4 Additions

$$T(n) = 8T(n/2) + O(n^2)$$

Standard Multiplication: Each recursive step performs **8 smaller matrix multiplications**, each on matrices of size **n/2 × n/2**.

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$A \times B = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \times \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right] = \left[\begin{array}{c|c} ae + bg & af + bh \\ \hline ce + dg & df + dh \end{array}\right]$$

8 Multiplications
4 Additions

$$T(n) = 8T(n/2) + O(n^2)$$

Standard Multiplication: Each recursive step performs **8 smaller matrix multiplications**, each on matrices of size **n/2 × n/2**.

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$A \times B = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \times \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right] = \left[\begin{array}{c|c} ae + bg & af + bh \\ \hline ce + dg & df + dh \end{array}\right]$$

8 Multiplications
4 Additions

$$T(n) = 8T(n/2) + O(n^2)$$

Standard Multiplication: Each recursive step performs **8 smaller matrix multiplications**, each on matrices of size **n/2 × n/2**.

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) \\ \Theta\left(n^{\log_b a} \log n\right) \\ \Theta(f(n)) \end{cases}$$

$$\log_b a = \log_2 8 = 3$$

$$f(n) = O(n^{\log_b a - \varepsilon})$$
$$f(n) = \Theta(n^{\log_b a})$$
$$f(n) = \Omega(n^{\log_b a + \varepsilon})$$

$$T(N) = \theta(n^3)$$

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$k_1 = a(f - h)$$
$$k_2 = (a + b)h$$
$$k_3 = (c + d)e$$
$$k_4 = d(g - e)$$
$$k_5 = (a + d)(e + h)$$
$$k_6 = (b - d)(g + h)$$
$$k_7 = (a - d)(e + f)$$

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$\begin{aligned} k_1 &= a(f-h) \\ k_2 &= (a+b)h \\ k_3 &= (c+d)e \\ k_4 &= d(g-e) \\ k_5 &= (a+d)(e+h) \\ k_6 &= (b-d)(g+h) \\ k_7 &= (a-d)(e+f) \end{aligned}$$

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

$$\begin{aligned} k_1 &= a(f-h) \\ k_2 &= (a+b)h \\ k_3 &= (c+d)e \\ k_4 &= d(g-e) \\ k_5 &= (a+d)(e+h) \\ k_6 &= (b-d)(g+h) \\ k_7 &= (a-d)(e+f) \end{aligned}$$

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

Strassen's Matrix Multiplication Transform

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

$$\begin{aligned} k_1 &= a(f-h) \\ k_2 &= (a+b)h \\ k_3 &= (c+d)e \\ k_4 &= d(g-e) \\ k_5 &= (a+d)(e+h) \\ k_6 &= (b-d)(g+h) \\ k_7 &= (a-d)(e+f) \end{aligned}$$

$$A \times B = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

$$T(n) = 7T(n/2) + O(n^2)$$

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$k_1 = a(f - h)$$
$$k_2 = (a + b)h$$
$$k_3 = (c + d)e$$
$$k_4 = d(g - e)$$
$$k_5 = (a + d)(e + h)$$
$$k_6 = (b - d)(g + h)$$
$$k_7 = (a - d)(e + f)$$

$T(N) = \theta(n^3)$

$$\boldsymbol{A}\times\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

$$T(n) = 7T(n/2) + O(n^2)$$

$$\log_b a = \log_2 7 = 2.807$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

$$\begin{aligned} k_1 &= a(f-h) \\ k_2 &= (a+b)h \\ k_3 &= (c+d)e \\ k_4 &= d(g-e) \\ k_5 &= (a+d)(e+h) \\ k_6 &= (b-d)(g+h) \\ k_7 &= (a-d)(e+f) \end{aligned}$$

$$\boldsymbol{A}\times\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

$$T(n) = 7T(n/2) + O(n^2)$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a}\log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

$$T(N) = \theta(n^{2.8074})$$ 😀

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$\begin{aligned} k_1 &= a(f-h) \\ k_2 &= (a+b)h \\ k_3 &= (c+d)e \\ k_4 &= d(g-e) \\ k_5 &= (a+d)(e+h) \\ k_6 &= (b-d)(g+h) \\ k_7 &= (a-d)(e+f) \end{aligned}$$

$T(N) = \theta(n^3)$

$$\boldsymbol{A}\times\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

$$T(n) = 7T(n/2) + O(n^2)$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a}\log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

$T(N) = \theta(n^{2.8074})$ 😀

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$k_1 = a(f - h)$$
$$k_2 = (a + b)h$$
$$k_3 = (c + d)e$$
$$k_4 = d(g - e)$$
$$k_5 = (a + d)(e + h)$$
$$k_6 = (b - d)(g + h)$$
$$k_7 = (a - d)(e + f)$$

Only $\boldsymbol{B}$

$T(N) = \theta(n^3)$

$$\boldsymbol{A} \times \boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
18 Additions

$$T(n) = 7T(n/2) + O(n^2)$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a} \log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

$T(N) = \theta(n^{2.8074})$ 😀

Strassen's Matrix Multiplication Transform

$$\boldsymbol{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$T(N) = \theta(n^3)$$

$$k_1 = a(f-h)$$
$$k_2 = (a+b)h$$
$$k_3 = (c+d)e$$
$$k_4 = d(g-e)$$
$$k_5 = (a+d)(e+h)$$
$$k_6 = (b-d)(g+h)$$
$$k_7 = (a-d)(e+f)$$

Only $\boldsymbol{B}$

When multiplying multiple matrices by $\boldsymbol{B}$, we calculated these terms only once

$$\boldsymbol{A}\times\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

7 Multiplications
~~18~~ 13 Additions

😀

$$T(n) = 7T(n/2) + O(n^2)$$

Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta\left(n^{\log_b a}\log n\right) & f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \end{cases}$$

$$T(N) = \theta(n^{2.8074})$$ 😀

Strassen's Matrix Multiplication Transform

- **Strassen's Algorithm in DNNs**:
  - Theoretically possible to speed up certain DNN operations.

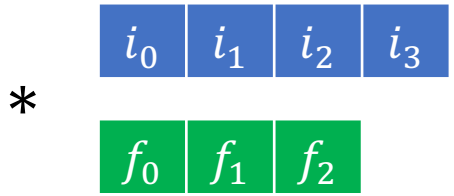
$$\boldsymbol{A}\times\boldsymbol{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

- **Challenges**:
  - **Memory Overhead**: Requires additional memory, which could be a limiting factor
  - **Precision**: Might introduce numerical instability or reduced precision in certain cases.
  - **Complexity**: Implementing Strassen's matrix multiplication can be more complex than the standard method.

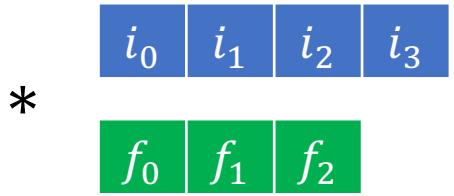
$$\begin{bmatrix} k_5 + k_4 - k_2 + k_6 & k_1 + k_2 \\ k_3 + k_4 & k_1 + k_5 - k_3 - k_7 \end{bmatrix}$$

While Strassen's matrix multiplication isn't commonly used in the mainstream DNN frameworks and libraries for typical networks, it might be explored in research settings or specialized applications where the matrix sizes and problem characteristics make it beneficial.

Winograd Transform for Convolutions



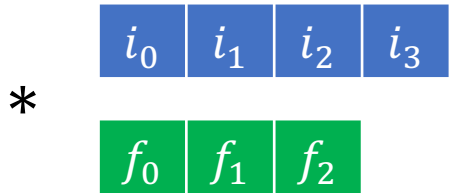
Winograd Transform for Convolutions



img2col

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix}$$

Winograd Transform for Convolutions

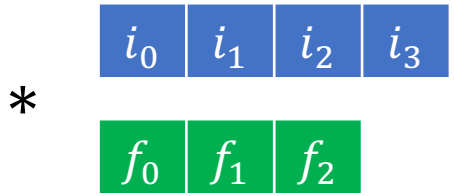


img2col

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix}$$

6 Multiplications
4 Additions

Winograd Transform for Convolutions



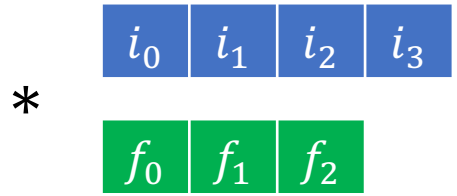
img2col

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 + k_3 \\ k_2 - k_3 - k_4 \end{bmatrix}$$

$$\begin{aligned} k_1 &= (i_0 - i_2) f_0 \\ k_2 &= (i_1 + i_2) \frac{f_0 + f_1 + f_2}{2} \\ k_3 &= (i_2 - i_1) \frac{f_0 - f_1 + f_2}{2} \\ k_4 &= (i_1 - i_3) f_2. \end{aligned}$$

6 Multiplications
4 Additions

Winograd Transform for Convolutions



img2col

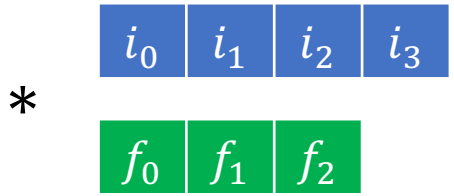
$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 + k_3 \\ k_2 - k_3 - k_4 \end{bmatrix}$$

6 Multiplications
4 Additions

$$\begin{aligned} k_1 &= (i_0 - i_2) f_0 \\ k_2 &= (i_1 + i_2) \frac{f_0 + f_1 + f_2}{2} \\ k_3 &= (i_2 - i_1) \frac{f_0 - f_1 + f_2}{2} \\ k_4 &= (i_1 - i_3) f_2. \end{aligned}$$

4 Multiplications
12 Additions

Winograd Transform for Convolutions



img2col

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 + k_3 \\ k_2 - k_3 - k_4 \end{bmatrix}$$

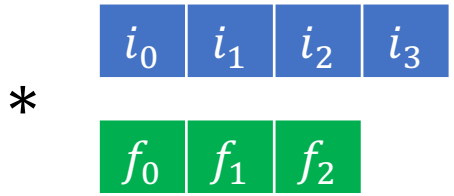
6 Multiplications
4 Additions

When multiplying multiple fmaps $\boldsymbol{F}$, we calculated these terms only once

$$\begin{aligned} k_1 &= (i_0 - i_2) f_0 \\ k_2 &= (i_1 + i_2) \frac{f_0 + f_1 + f_2}{2} \\ k_3 &= (i_2 - i_1) \frac{f_0 - f_1 + f_2}{2} \\ k_4 &= (i_1 - i_3) f_2. \end{aligned}$$

4 Multiplications
12 Additions

Winograd Transform for Convolutions



img2col

$$\begin{bmatrix} i_0 & i_1 & i_2 \\ i_1 & i_2 & i_3 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} o_0 \\ o_1 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 + k_3 \\ k_2 - k_3 - k_4 \end{bmatrix}$$

6 Multiplications
4 Additions

When multiplying multiple fmaps F, we calculated these terms only once

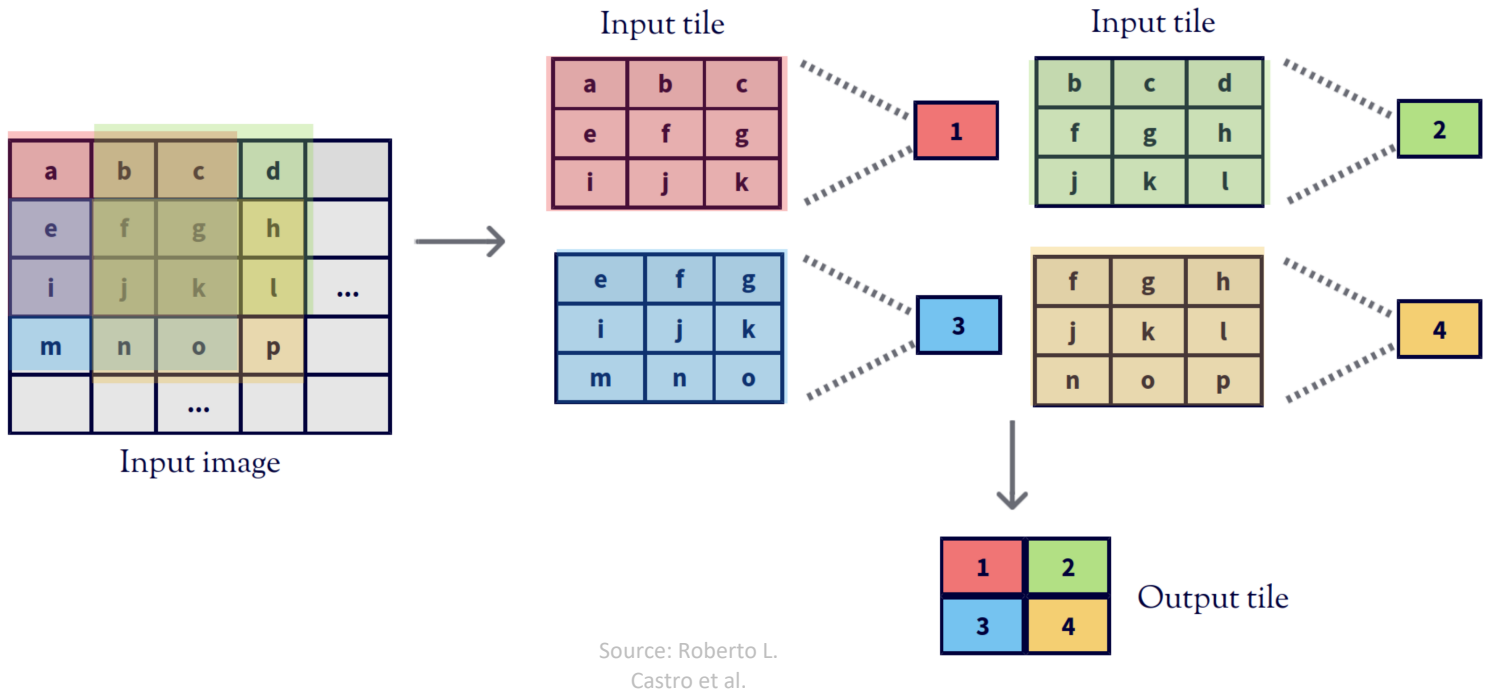
$$k_1 = (i_0 - i_2) f_0$$
$$k_2 = (i_1 + i_2) \frac{f_0 + f_1 + f_2}{2}$$
$$k_3 = (i_2 - i_1) \frac{f_0 - f_1 + f_2}{2}$$
$$k_4 = (i_1 - i_3) f_2.$$

4 Multiplications
8 ~~12~~ Additions

😃

Winograd Transform for Convolutions

- **Key Insight**: Reduces multiplication operations in convolution computations.
- **Performance Boost**: Achieves approximately **2.25x reduction** in multiplications for a 3x3 convolution.
- **Specificity**:
  - Tailored based on **filter size**.
  - Depends on **tile size** of input data.



Source: Roberto L. Castro et al.

Winograd Transform for 2D Convolutions (Linear Algebraic)

Suppose we want to convolve filter f and input i. We first define the following matrices.

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \quad G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

Transform matrices (depend on output tile and filter size)
values shown for tile size 2, and filter size 3x3

$$Y = A^T\Big[\big[GfG^T\big] \odot \big[B^T iB\big]\Big]A$$

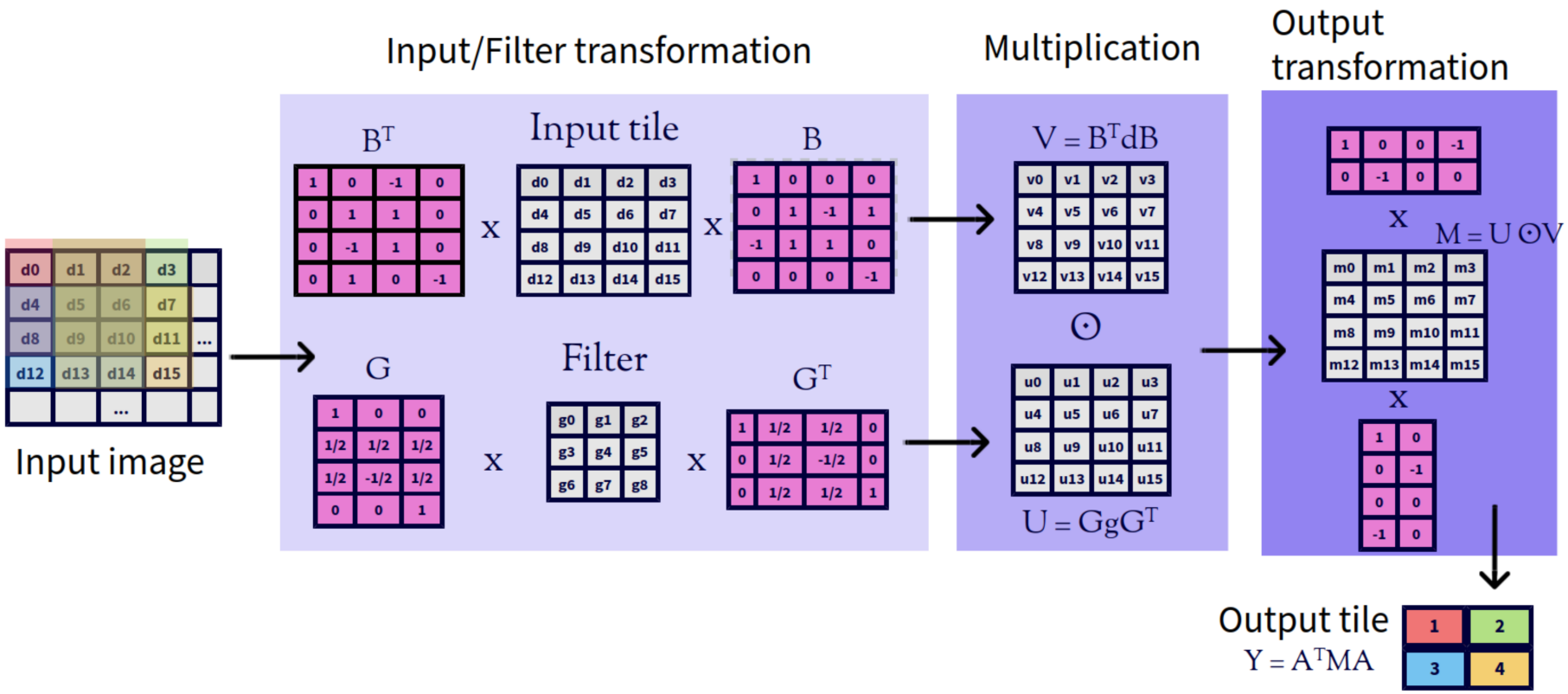
filter

Input

Constructing these matrices in general is explained in
Winograd Convolution for DNNs: Beyond Linear Polynomials. Barabasz, B. et al. , In AI*IA 2019

Winograd Transform for 2-D Convolutions

$$Y = A^T\Big[\big[GfG^T\big] \odot \big[B^T iB\big]\Big]A$$



Source: Roberto L. Castro et al.

Winograd Transform for 2-D Convolutions

$$Y = A^T\Big[\big[GfG^T\big] \odot \big[B^T iB\big]\Big]A$$

Convolution Operations

Convolution operations work on two sets of data: one set of offline-trained "weights" (which remain constant between each run of inference), and one set of input "feature" data (which varies with the network's input). The NVDLA Convolution Engine exposes parameters that enable several different modes of operation. Each of these modes include optimizations that improve performance over a naive convolution implementation:

- Direct
- Image-input
- Winograd
- Batching

Source: http://nvdla.org/hw/v1/hwarch.html

Convolution involves two sets of data:

- **Weights**: Pre-trained, remain constant during inference.
- **Feature Data**: Changes with each input.

Optimization Modes for Efficient Convolutions:

- **Direct**: Standard convolution, directly computing the output.
- **Image-Input**: Optimized for handling input images efficiently.
- **Winograd**: A fast algorithm that reduces the number of multiplications, improving speed.
- **Batching**: Processes multiple inputs together to improve efficiency.

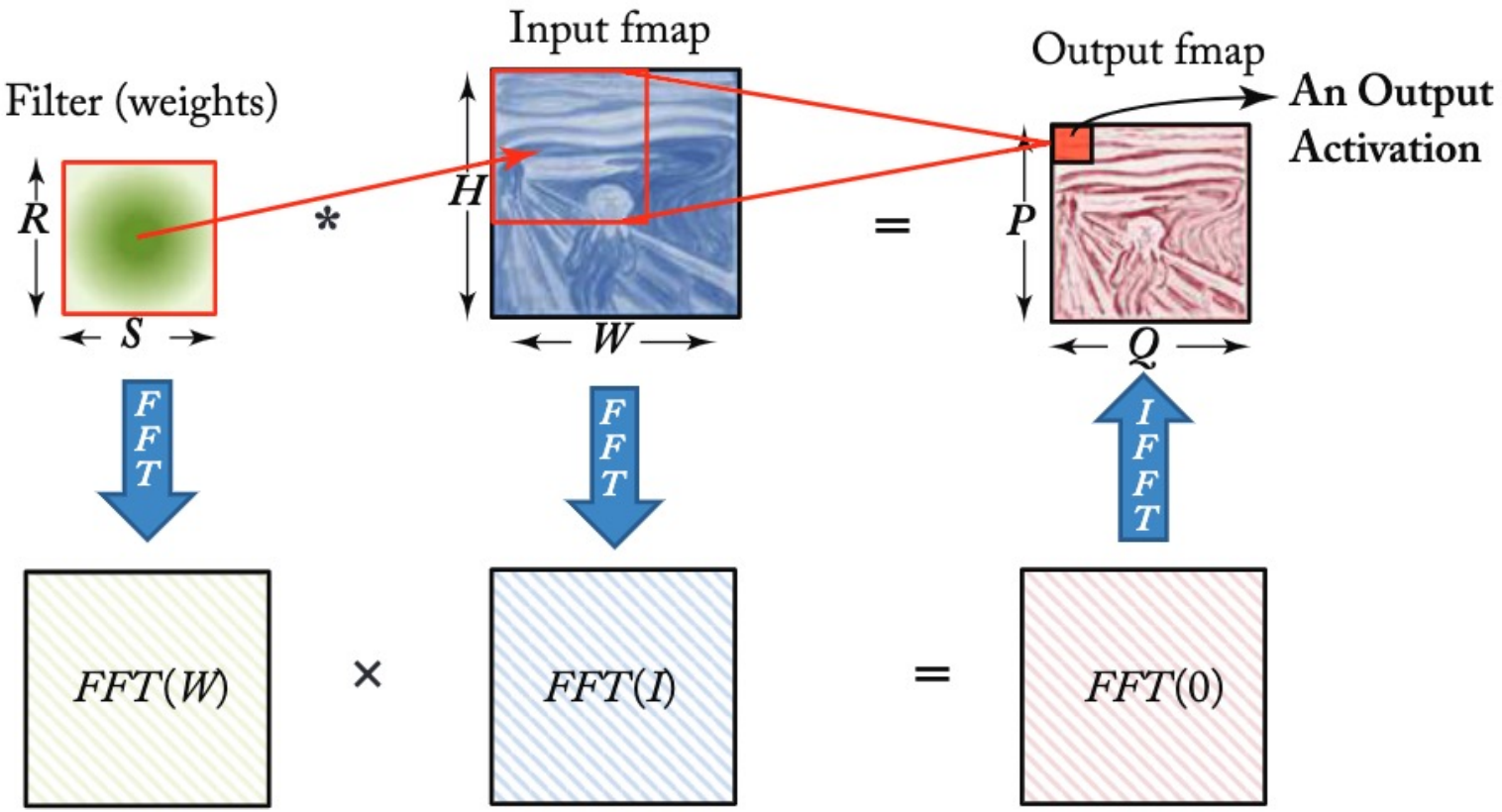
Winograd Transform for 2-D Convolutions

$$Y = A^T\Big[\big[GfG^T\big] \odot \big[B^T iB\big]\Big]A$$

- **Benefits:**
  - **Faster Convolutions**: Reduction in the number of multiplications speeds up computation.
- **Challenges**:
  - **Precision**: Potential minor numerical errors due to transforms.
  - **Kernel Size Limitations**: Primarily for small kernels (e.g., 3x3).
- **Implementation**:
  - Common in deep learning frameworks like cuDNN (used in TensorFlow & PyTorch).

Empirical Observations: In practice, when implemented in deep learning frameworks and libraries, the Winograd algorithm has been observed to give the most performance benefit for small kernels, such as **3x3**. This kernel size is also quite common in many modern CNN architectures (e.g., VGG, ResNet), making the optimization even more relevant.

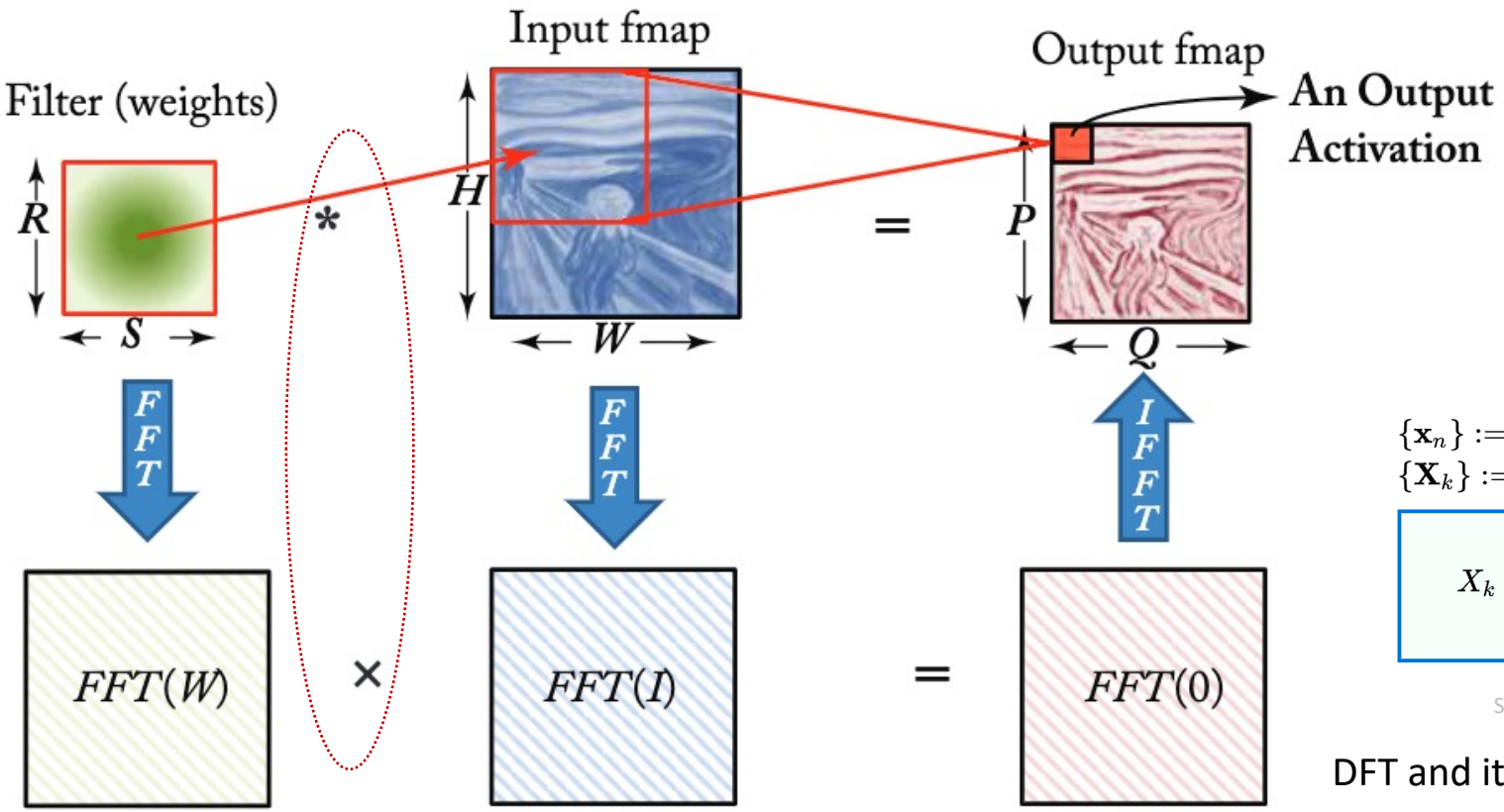
FFT Transform for Convolutions



Source: Roberto L. Castro et al.

FFT Transform for Convolutions

$O(RSPQ)$



$O(PQ \log PQ)$

Source: Roberto L. Castro et al.

$$\{\mathbf{x}_n\} := x_0, x_1, \ldots, x_{N-1}$$
$$\{\mathbf{X}_k\} := X_0, X_1, \ldots, X_{N-1}$$

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \quad \textbf{(Eq.1)}$$

Source: Wikipedia

DFT and its inverse is O(PQlog(PQ))

We gain when

$RS > \log PQ$

Total Complexity

- **Forward FFT:** $O(PQ \log PQ)$
- **Element-wise Multiplication:** $O(PQ)$
- **Inverse FFT:** $O(PQ \log PQ)$

Since $O(PQ)$ is smaller than $O(PQ \log PQ)$, the dominant term is:

$$O(PQ \log PQ)$$

FFT Transform for Convolutions

Given a matrix A of size $M \times N$, where $A_{m,n}$ represents the element of the matrix at row m and column n, the DFT F of the matrix A is given by the formula:

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

where:

- $F(k,l)$ is the element of the Fourier-transformed matrix at row k and column l,
- M and N are the dimensions of the matrix A,
- m and n are indices that run over all the rows and columns of the matrix A,
- k and l are indices that run over all the rows and columns of the transformed matrix F,
- i is the imaginary unit ($i^2 = -1$),
- e is the base of the natural logarithm.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{\text{FFT}} \begin{bmatrix} 45.00 - 0.00j & -4.50 + 2.60j & -4.50 - 2.60j \\ -13.50 + 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \\ -13.50 - 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \xrightarrow{\text{FFT}} \begin{bmatrix} 1.00 - 0.00j & -0.50 - 0.87j & -0.50 + 0.87j \\ -2.00 - 1.73j & -2.00 + 0.00j & -0.50 - 0.87j \\ -2.00 + 1.73j & -0.50 + 0.87j & -2.00 - 0.00j \end{bmatrix}$$

FFT Transform for Convolutions

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad * \quad \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$$

FFT Transform for Convolutions

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad * \quad \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$$

FFT

FFT

$$\begin{bmatrix} 1.00 - 0.00j & -0.50 - 0.87j & -0.50 + 0.87j \\ -2.00 - 1.73j & -2.00 + 0.00j & -0.50 - 0.87j \\ -2.00 + 1.73j & -0.50 + 0.87j & -2.00 - 0.00j \end{bmatrix} \quad x \quad \begin{bmatrix} 45.00 - 0.00j & -4.50 + 2.60j & -4.50 - 2.60j \\ -13.50 + 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \\ -13.50 - 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \end{bmatrix}$$

FFT Transform for Convolutions

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad * \quad \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$$

FFT

FFT

$$\begin{bmatrix} 1.00 - 0.00j & -0.50 - 0.87j & -0.50 + 0.87j \\ -2.00 - 1.73j & -2.00 + 0.00j & -0.50 - 0.87j \\ -2.00 + 1.73j & -0.50 + 0.87j & -2.00 - 0.00j \end{bmatrix} \quad x \quad \begin{bmatrix} 45.00 - 0.00j & -4.50 + 2.60j & -4.50 - 2.60j \\ -13.50 + 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \\ -13.50 - 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \end{bmatrix}$$

$$\begin{bmatrix} 45.00 - 0.00j & 4.50 + 2.60j & 4.50 - 2.60j \\ 40.50 + 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \\ 40.50 - 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \end{bmatrix}$$

FFT Transform for Convolutions

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad * \quad \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$$

FFT

FFT

$$\begin{bmatrix} 1.00 - 0.00j & -0.50 - 0.87j & -0.50 + 0.87j \\ -2.00 - 1.73j & -2.00 + 0.00j & -0.50 - 0.87j \\ -2.00 + 1.73j & -0.50 + 0.87j & -2.00 - 0.00j \end{bmatrix} \quad x \quad \begin{bmatrix} 45.00 - 0.00j & -4.50 + 2.60j & -4.50 - 2.60j \\ -13.50 + 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \\ -13.50 - 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \end{bmatrix}$$

$$\begin{bmatrix} 45.00 - 0.00j & 4.50 + 2.60j & 4.50 - 2.60j \\ 40.50 + 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \\ 40.50 - 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \end{bmatrix}$$

IFFT

$$\begin{bmatrix} 15 & 13 & 14 \\ 0 & -2 & -1 \\ 3 & 1 & 2 \end{bmatrix}$$

FFT Transform for Convolutions

$$F(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{m,n} \cdot e^{-2\pi i \left(\frac{mk}{M} + \frac{nl}{N}\right)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad * \quad \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix} \quad = \quad \begin{bmatrix} -2 & -1 \\ 1 & 2 \end{bmatrix}$$

FFT

FFT

$$\begin{bmatrix} 1.00 - 0.00j & -0.50 - 0.87j & -0.50 + 0.87j \\ -2.00 - 1.73j & -2.00 + 0.00j & -0.50 - 0.87j \\ -2.00 + 1.73j & -0.50 + 0.87j & -2.00 - 0.00j \end{bmatrix} \quad x \quad \begin{bmatrix} 45.00 - 0.00j & -4.50 + 2.60j & -4.50 - 2.60j \\ -13.50 + 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \\ -13.50 - 7.79j & 0.00 + 0.00j & 0.00 - 0.00j \end{bmatrix}$$

$$\begin{bmatrix} 45.00 - 0.00j & 4.50 + 2.60j & 4.50 - 2.60j \\ 40.50 + 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \\ 40.50 - 7.79j & -0.00 + 0.00j & -0.00 + 0.00j \end{bmatrix}$$

IFFT

$$\begin{bmatrix} 15 & 13 & 14 \\ 0 & -2 & -1 \\ 3 & 1 & 2 \end{bmatrix}$$

Efficient Algorithm Selection for Convolutional Layers

- **Adaptive Algorithm Selection**:
  - Different layers might necessitate different algorithms based on their shapes and sizes.
- **Algorithm Criteria**:
  - **FFT (Fast Fourier Transform)**:
    - Suited for larger filters, e.g., greater than 5x5.
  - **Winograd Transform**:
    - Optimal for smaller filters, e.g., 3x3 and below.
- **3. Dynamic Choice by Libraries**:
- Platform libraries such as **MKL** and **cuDNN**:
  - Automatically determine and use the best algorithm for the given layer shape and size.

Efficient Algorithm Selection for Convolutional Layers

5.2.10. cudnnFindConvolutionForwardAlgorithmEx()

This function attempts all algorithms available for `cudnnConvolutionForward()`. It will attempt both the provided `convDesc mathType` and `CUDNN_DEFAULT_MATH` (assuming the two differ).

```
cudnnStatus_t cudnnFindConvolutionForwardAlgorithmEx(
    cudnnHandle_t                       handle,
    const cudnnTensorDescriptor_t       xDesc,
    const void                         *x,
    const cudnnFilterDescriptor_t       wDesc,
    const void                         *w,
    const cudnnConvolutionDescriptor_t  convDesc,
    const cudnnTensorDescriptor_t       yDesc,
    void                               *y,
    const int                           requestedAlgoCount,
    int                                *returnedAlgoCount,
    cudnnConvolutionFwdAlgoPerf_t      *perfResults,
    void                               *workSpace,
    size_t                              workSpaceSizeInBytes)
```

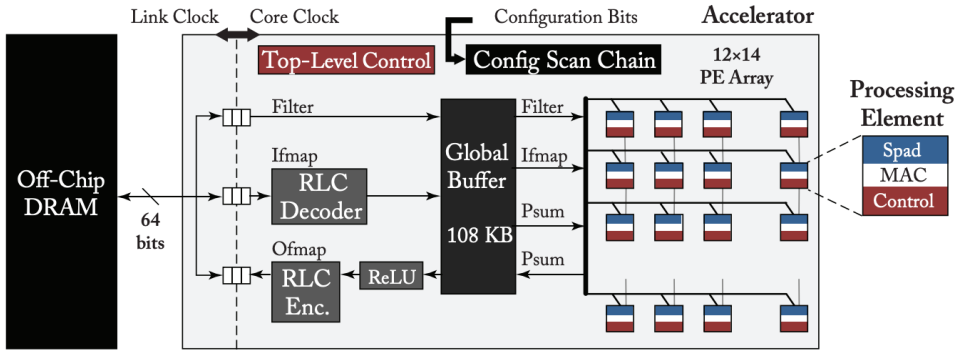
https://docs.nvidia.com/deeplearning/cudnn/api/index.html#cudnnFindConvolutionForwardAlgorithmEx

- This function benchmarks different convolution algorithms on the current hardware and selects the best one according to performance.
- The selection process considers the memory constraints imposed by the workspace size parameter.
- It's designed to optimize the convolution operation for efficiency on different hardware configurations.

Buffer Management

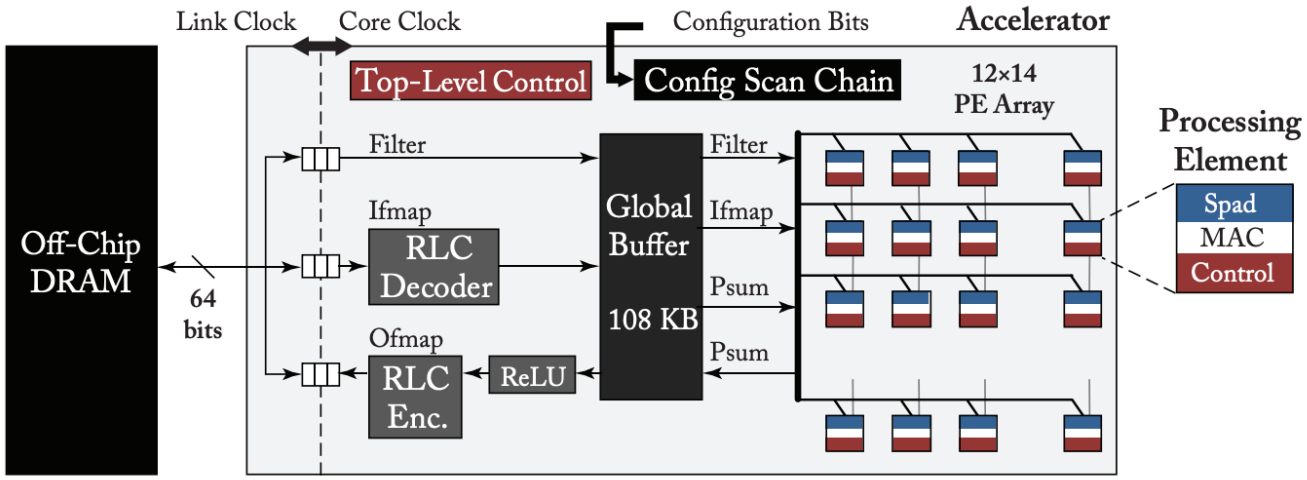
DNN Orchestration

- For efficient DNN accelerator performance.
- Objectives:
  - **Efficient** and **timely** data transfer:
    - Ensure only the necessary future data is transferred to the consumer.
  - Data **overlap**:
    - Overlap **data receipt** for future use with the **consumption** of current data.
  - Timely **data removal**:
    - Discard data precisely when it's no longer required.
  - Precise & cost-effective **synchronization**: Manage data flow seamlessly and affordably.



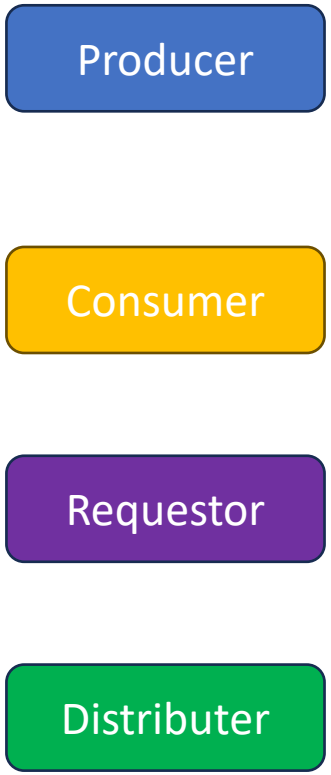
DNN Orchestration

- Affects efficiency and performance of DNN accelerator.
- Providing data when it is needed
- Removing data when it’s no longer needed

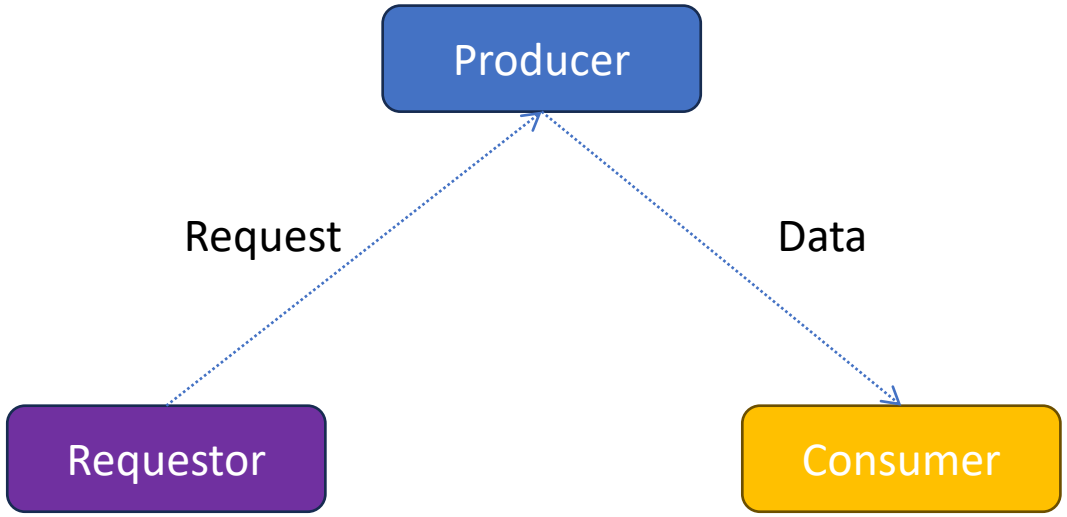


DNN Orchestration Roles

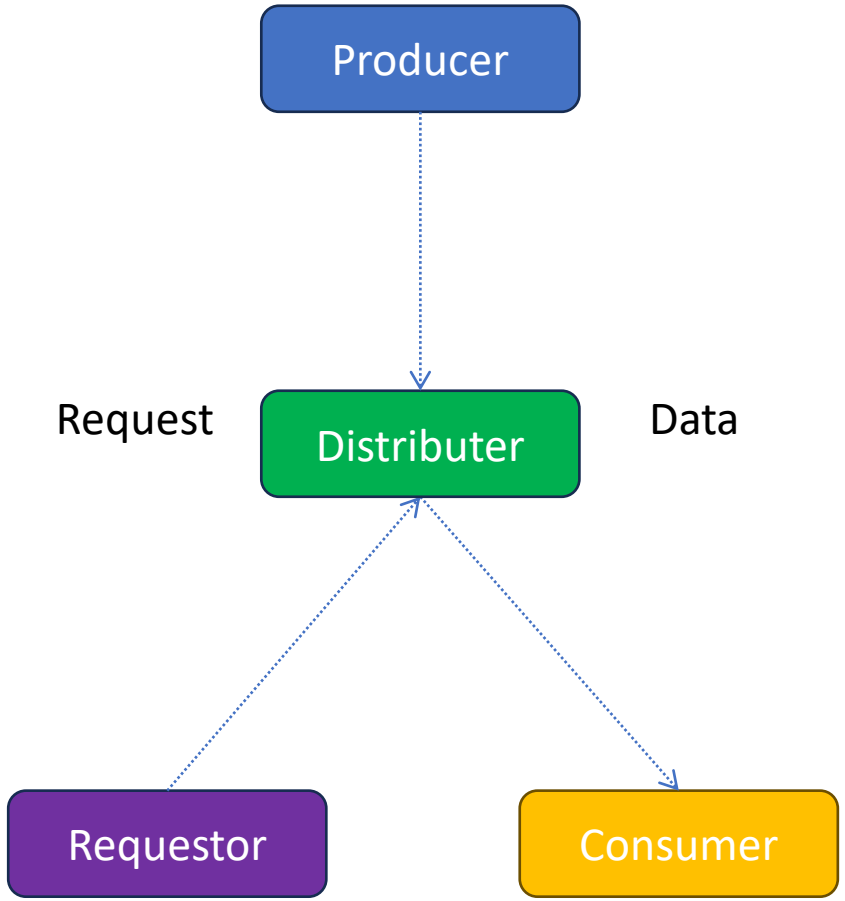
- **Producer**
  - The entity that contains the requested data
- **Consumer**
  - The entity that consumes the requested data
- **Requestor**
  - The entity that requests the data
- **Distributer**
  - The entity that distributes data requests



DNN Orchestration Roles

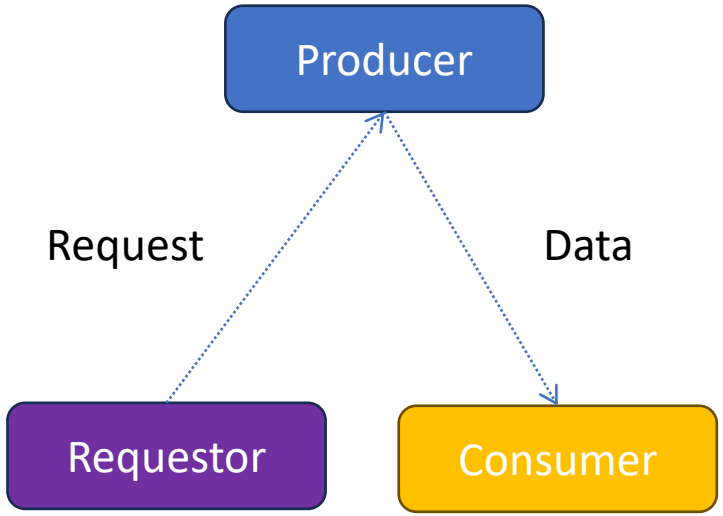


DNN Orchestration Roles

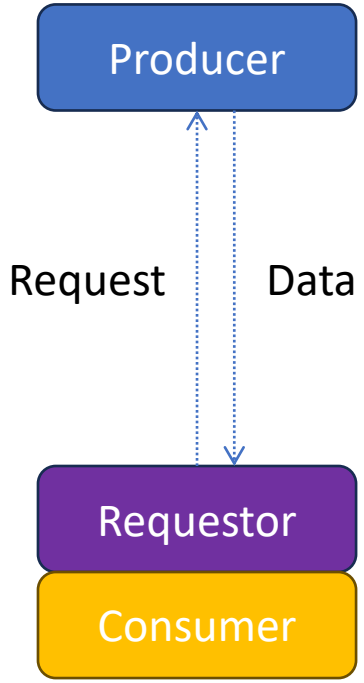


Coupled vs. Decoupled

Is Requestor the same as Consumer?



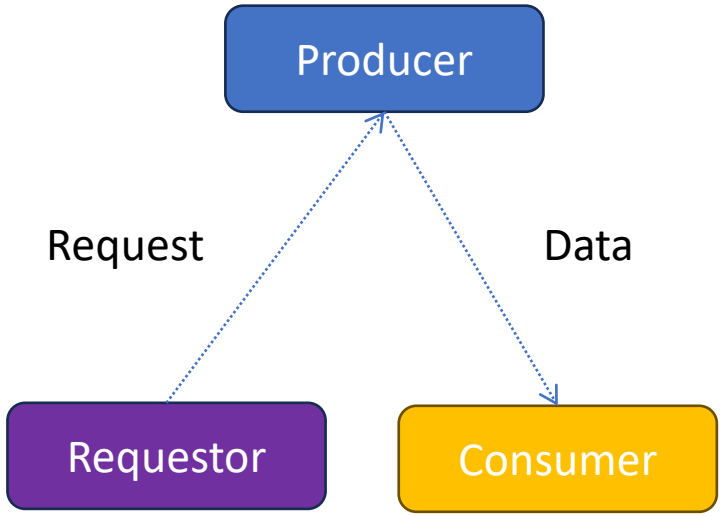
Decoupled



Coupled

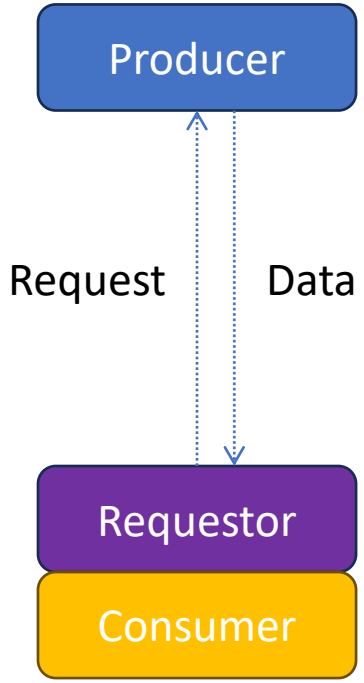
Coupled vs. Decoupled

Is Requestor the same as Consumer?



Decoupled

Requestor can request in advance
Requires synchronization

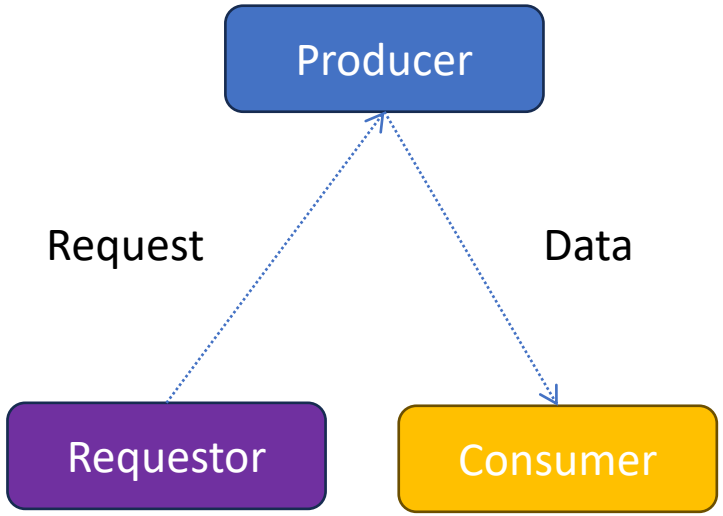


Coupled

Simpler Design

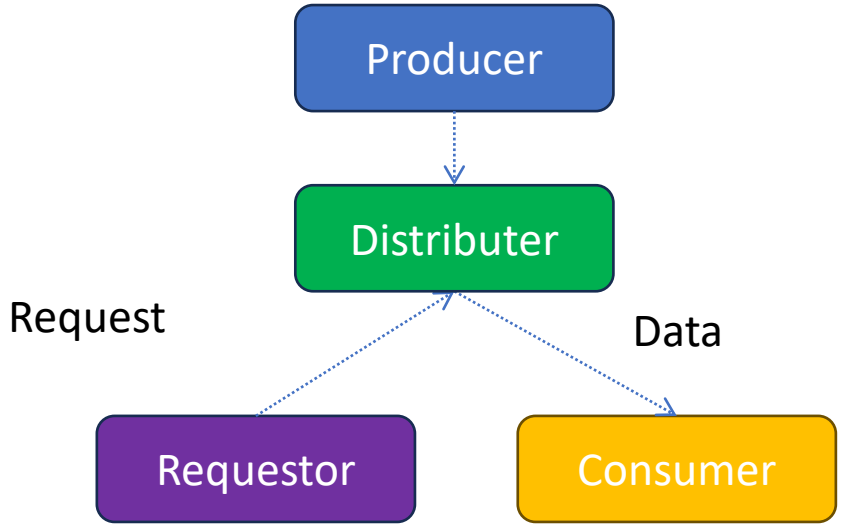
Implicit vs Explicit

Are requests sent to data producer directly?



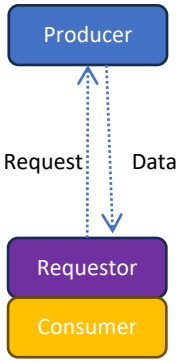
Explicit

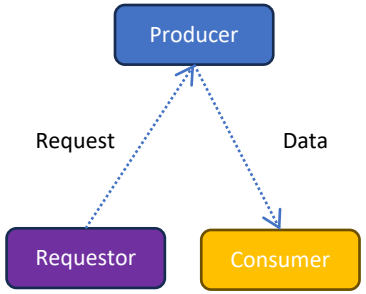
Simpler Design
Harder for programmers

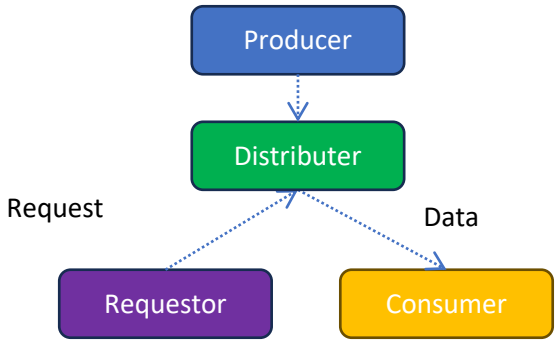


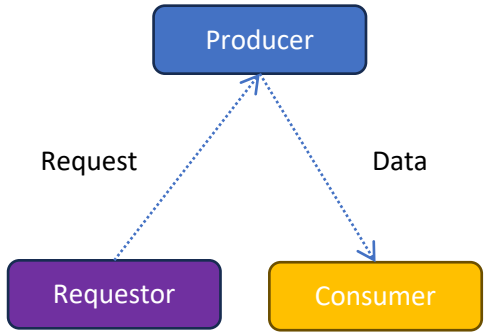
Implicit

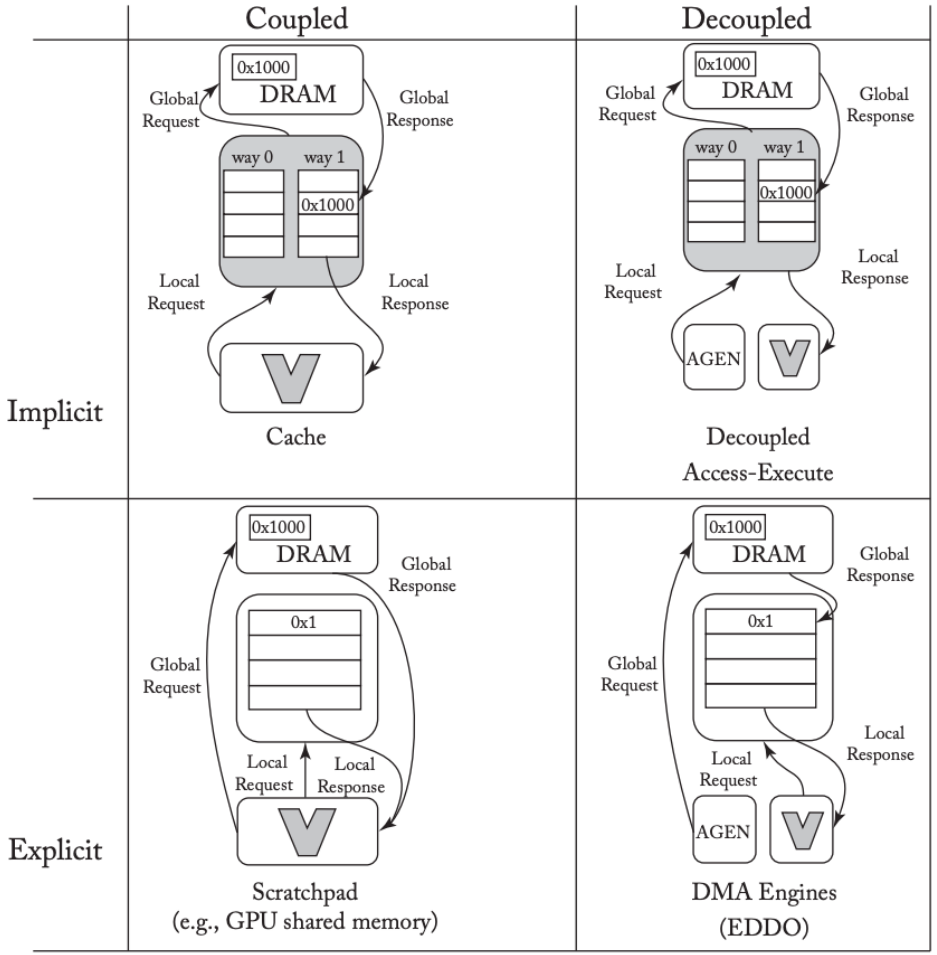
- Requestor does not know where the actual data is
- Easier for programmers
- Higher area and complex hardware


Producer
Request
Data
Requestor
Consumer


Producer
Request
Data
Requestor
Consumer

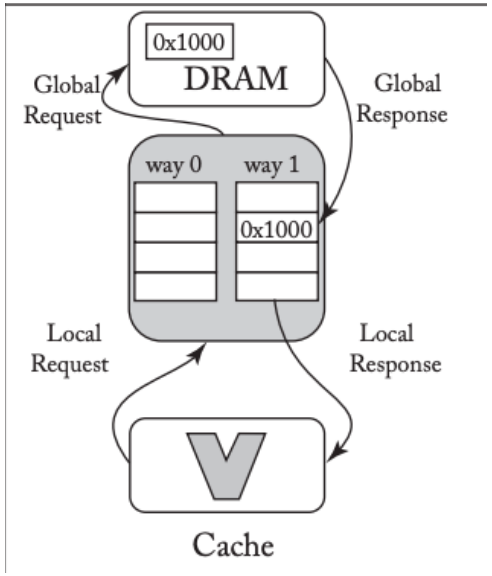

Producer
Distributer
Request
Data
Requestor
Consumer

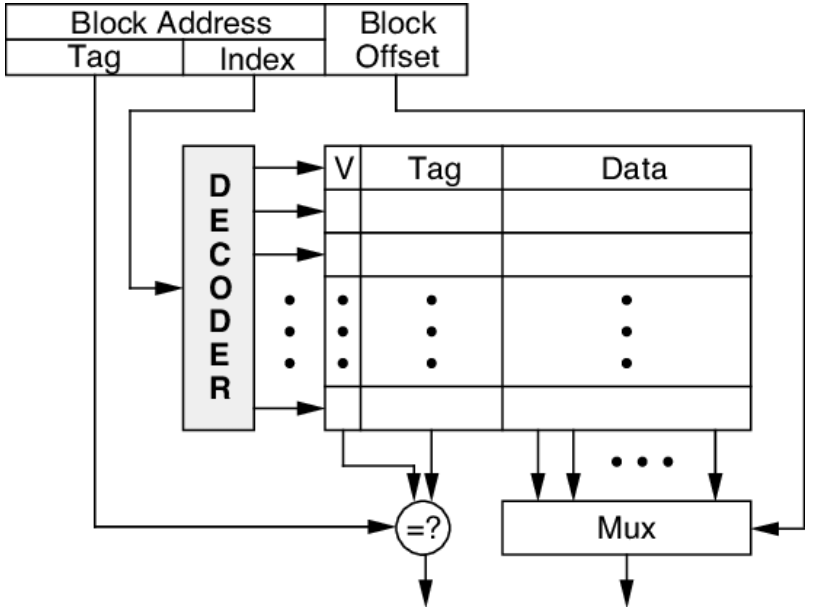

Producer
Request
Data
Requestor
Consumer


Coupled
Decoupled
Implicit
0x1000
DRAM
Global Request
Global Response
way 0
way 1
0x1000
Local Request
Local Response
AGEN
Cache
Decoupled Access-Execute
Explicit
0x1
Scratchpad (e.g., GPU shared memory)
DMA Engines (EDDO)

Implicit-Coupled

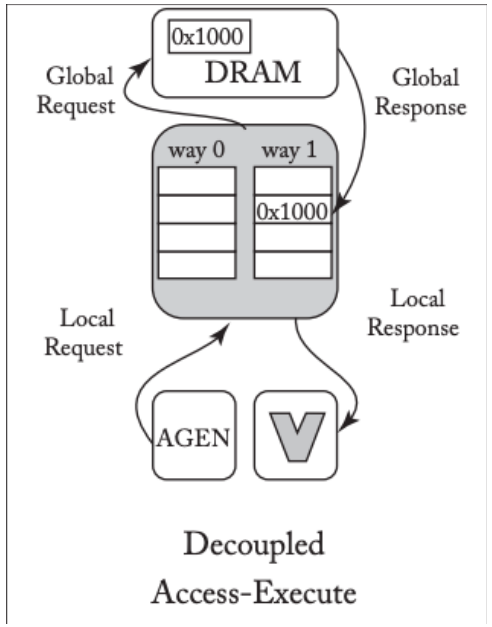
- Cache
  - Popular in general purpose computing
  - **Features:**
    - **Reusable:** can be used multiple times, avoiding frequent data fetches from main memory.
    - **Composable**: Can be combined with other computing elements for optimized performance.
  - **Drawbacks**:
    - High area and energy overhead
- Cache Mechanism:
  - **Implicit**: requestor sends request to L1 cache without knowing where the actual data is.
  - **Coupled**: The requestor and consumer are often the same.

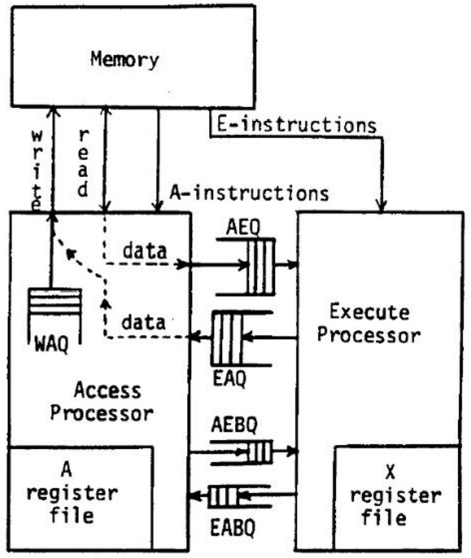




Implicit-Decoupled

- Decoupled Access-Execute
  - Requestor (Address Generator) is separated from consumer
  - Optimized parallelization of each stream.
  - Reduced stall times.
- Disadvantages:
  - Complexity
  - Synchronization
  - OOO and memory hierarchy already hide memory latency
  - Compiler complexity
- Mechanism:
  - **Implicit:** Requestor sends requests to L1 without knowing the exact location of data.
  - **Decoupled:**
    - Separate requestor and consumer

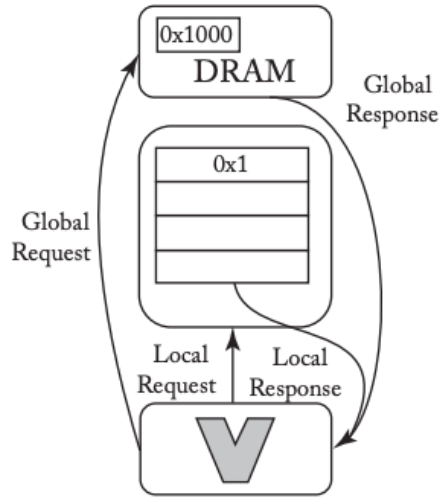


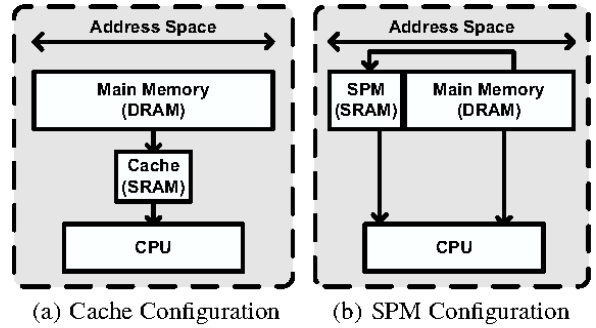


Source: Smith, 1982

Explicit-Coupled

- GPU Shared Memory
- **Scratchpad** in GPU
  - Fast, on-chip memory
  - Can be explicitly managed by the programmer
  - Unlike caches
- Instructions to explicitly move data across memory hierarchy
- Mechanism:
  - **Explicit**: Core explicitly requests data from DRAM
  - **Coupled**





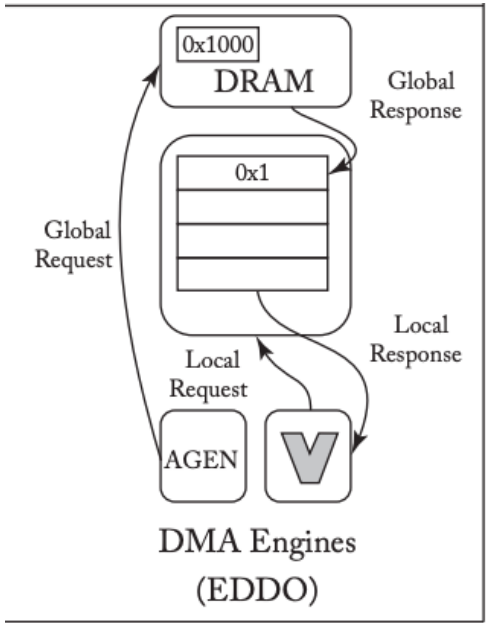
Source: Janapsatya et al.

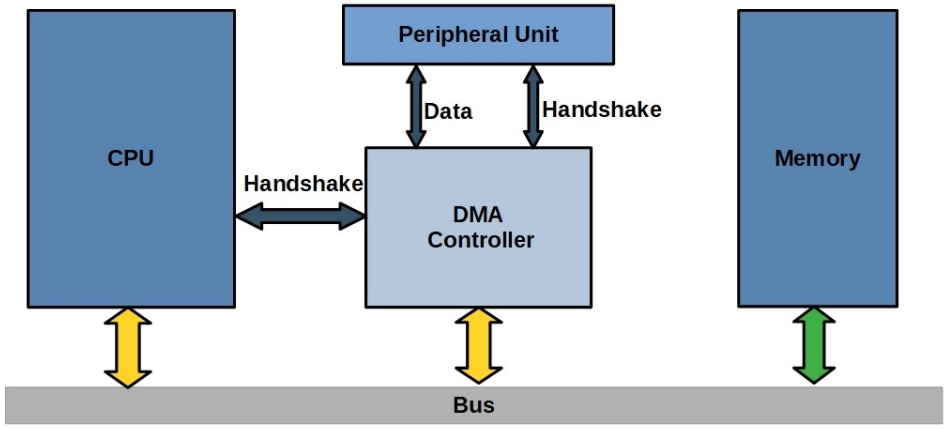
```
__global__ void someKernel(float* input, float* output) {
  // Static allocation of shared memory
  __shared__ float sharedData[256];

  int tid = threadIdx.x;

  // Load data into shared memory
  sharedData[tid] = input[tid];

  // Synchronize all threads in block to ensure all data is loaded
  __syncthreads();

  // Some processing code here, which benefits from the fast access of shared
memory...

  output[tid] = sharedData[tid];
}
```

Simple example in which shared memory is used in a CUDA kernel

Explicit-Decoupled (EDDO)

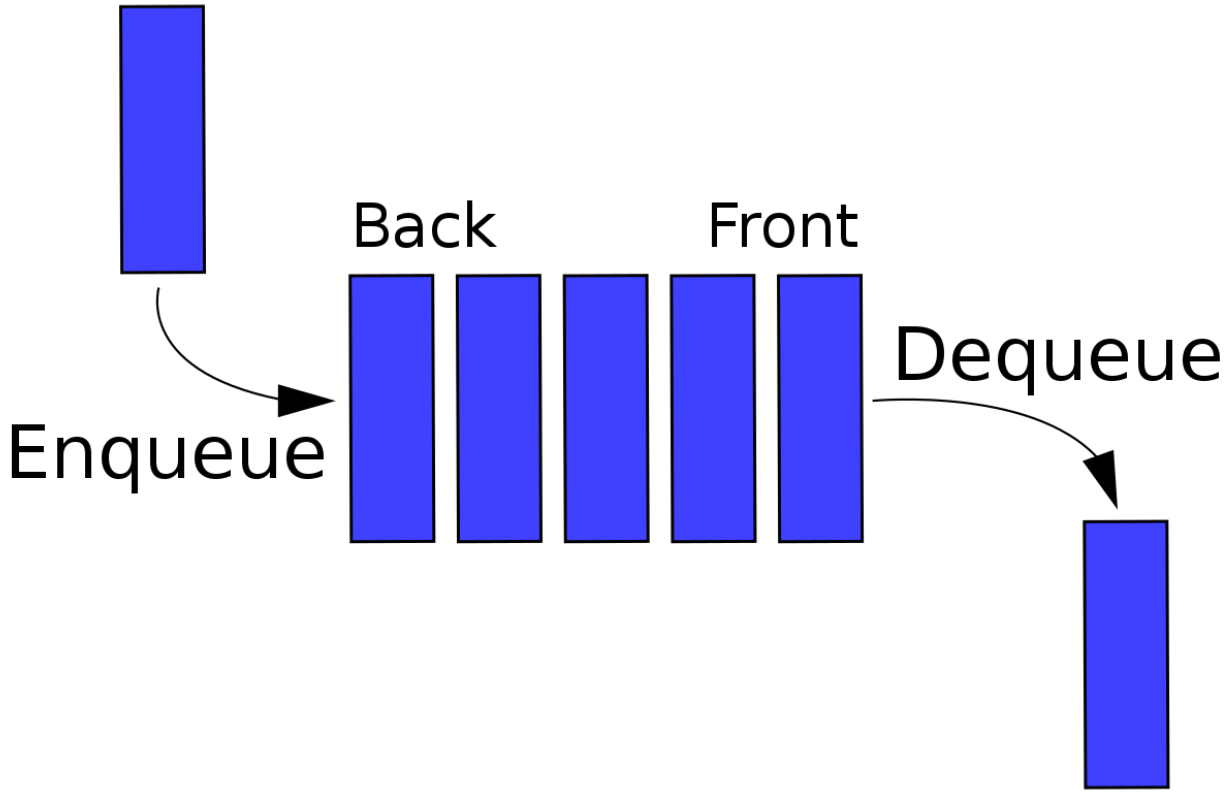
- Direct Memory Access (DMA)
  - Generates addresses
  - Data is sent to the consumer
  - Efficient hardware implementation
  - Requires synchronization
- Mechanism:
  - **Explicit**
  - **DeCoupled**

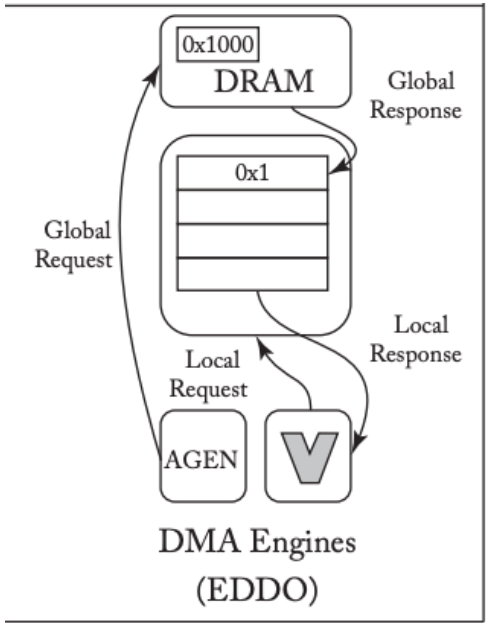


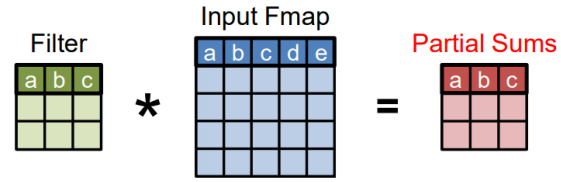


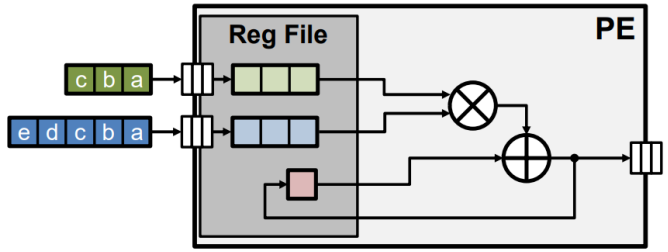
Explicit-Decoupled (EDDO)

- FIFO
  - First in First Out
  - Data cannot be updated



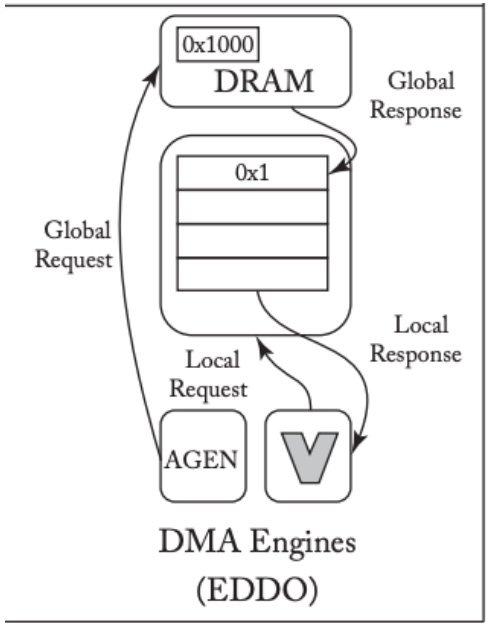


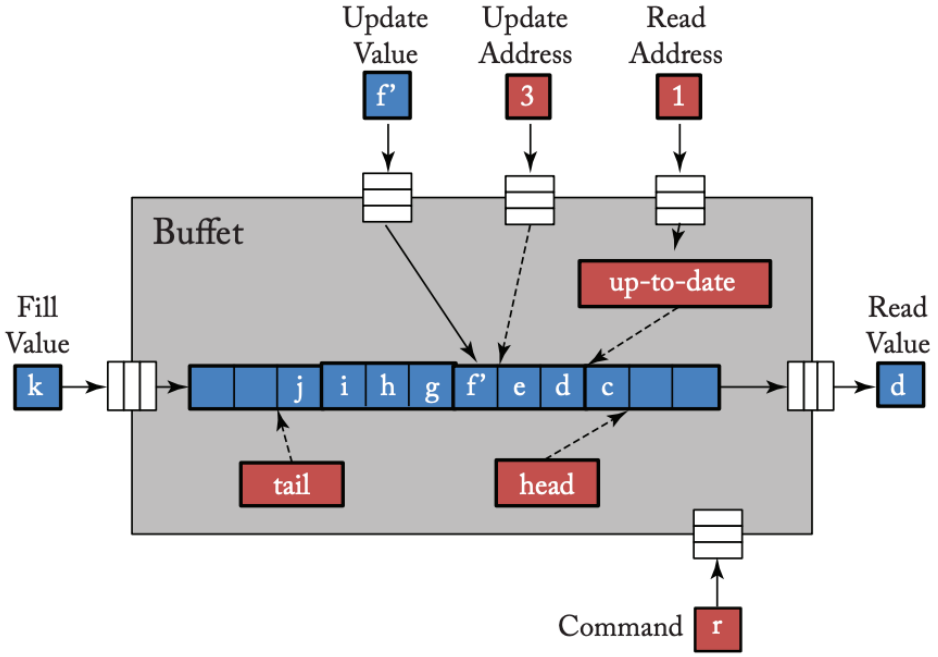




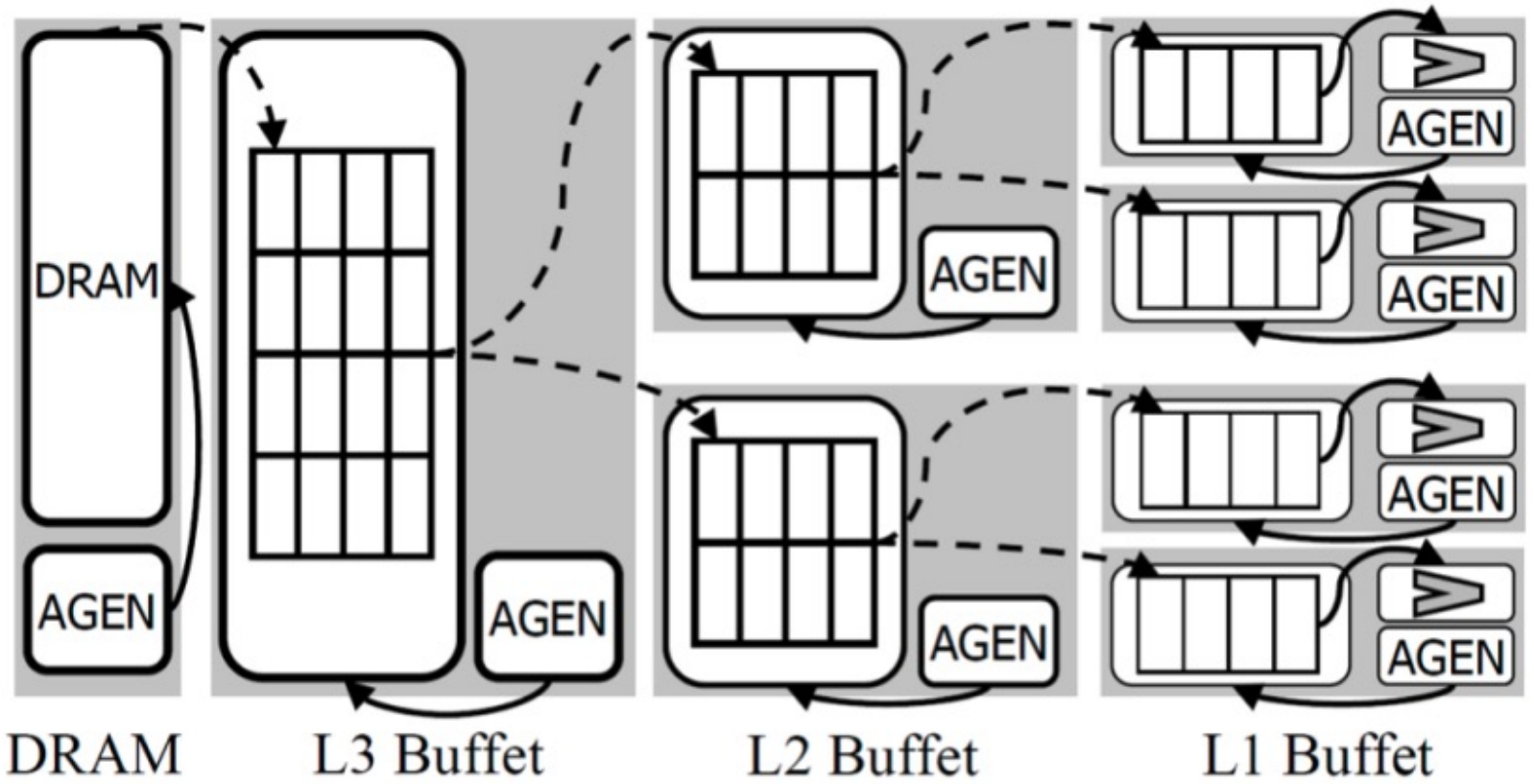
Explicit-Decoupled (EDDO)

- Buffets
  - Enhanced FIFO
  - Data can be updated
  - Commands:
    - Read
    - Read + Update
    - Shrink (Remove k items from head)

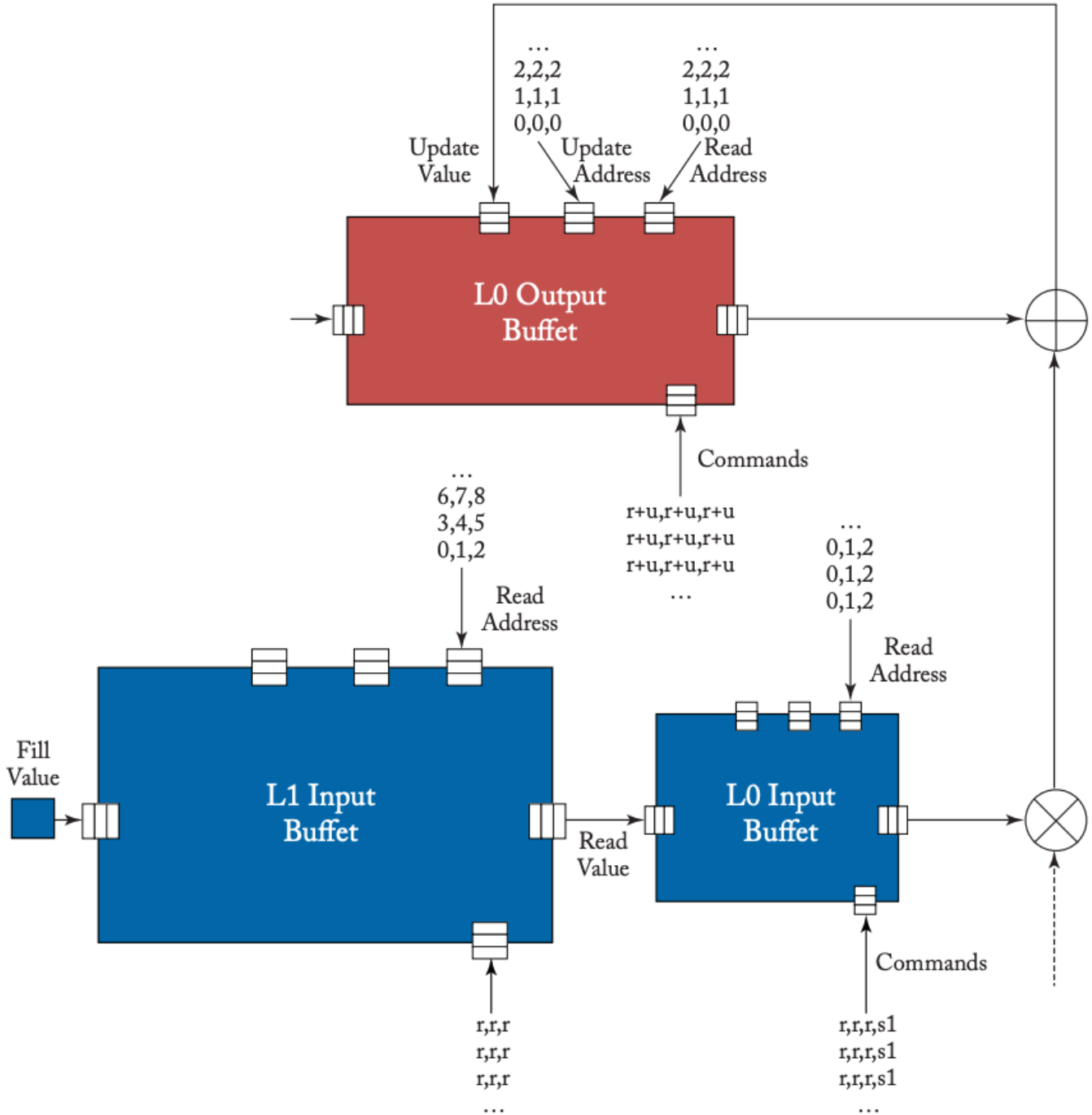




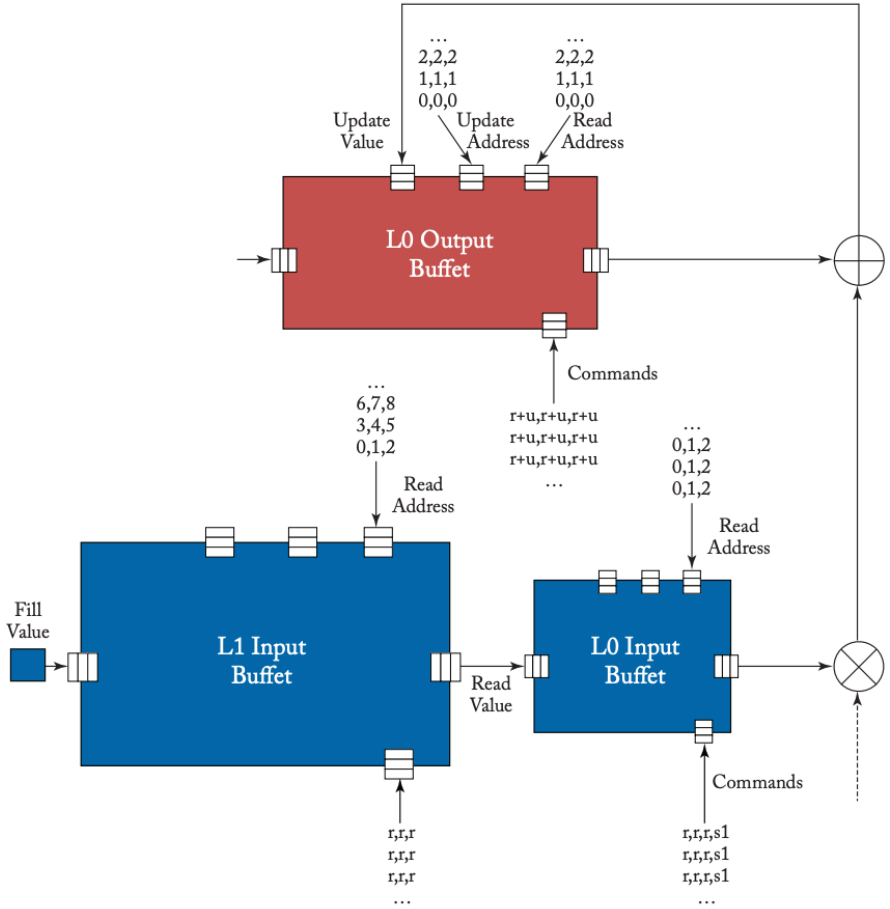
Explicit-Decoupled (EDDO)



Buffet Based Accelerator


...
2,2,2
1,1,1
0,0,0
Update Value
Update Address
...
2,2,2
1,1,1
0,0,0
Read Address
L0 Output Buffet
Commands
r+u,r+u,r+u
r+u,r+u,r+u
r+u,r+u,r+u
...
...
6,7,8
3,4,5
0,1,2
Read Address
...
0,1,2
0,1,2
0,1,2
Read Address
Fill Value
L1 Input Buffet
Read Value
L0 Input Buffet
Commands
r,r,r
r,r,r
r,r,r
...
r,r,r,s1
r,r,r,s1
r,r,r,s1
...

Explicit-Decoupled (EDDO)



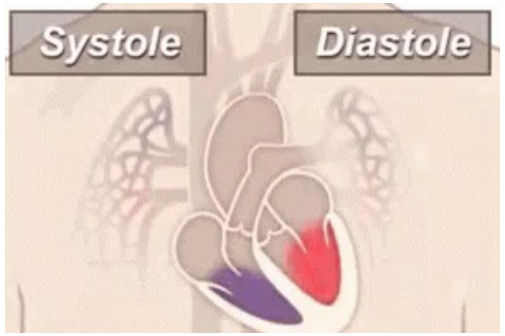
- Eyeriss-like global buffer and PE built with buffets.
- Reads to the L1 Input Buffet fill the L0 Buffet, which performs reads that pass a sliding window of inputs to the multiplier.
- The L0 Output Buffer performs a series of read+update commands to generate the partial sums.

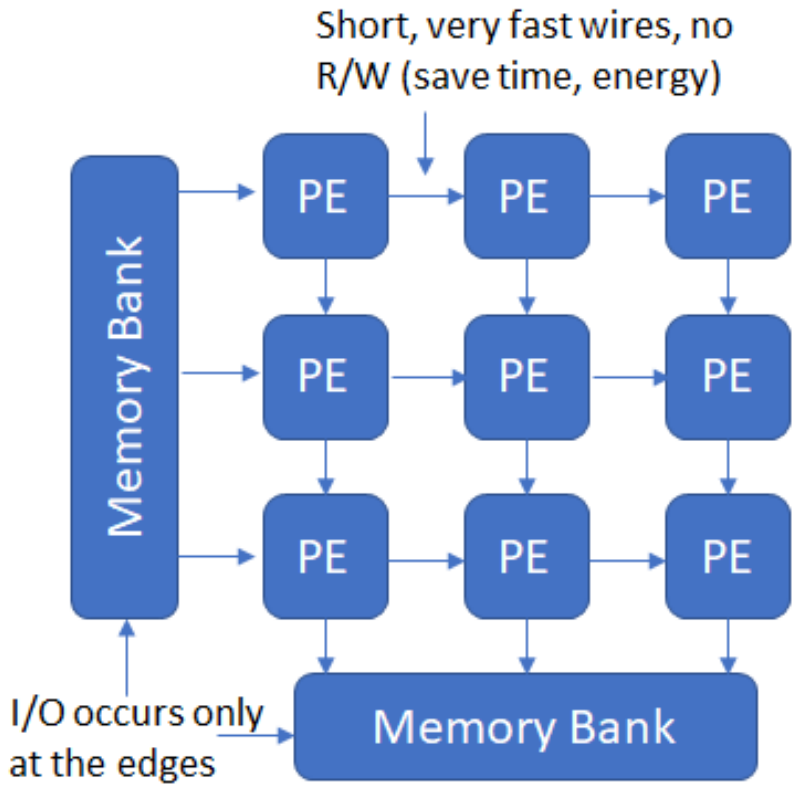
Systolic Array

Matrix Multiplication

Systolic Arrays

- **Definition**: A parallel computing architecture where PEs are arranged in a regular grid pattern.
- **Data Movement**: Data flows rhythmically between PEs, like blood in the heart (hence the name 'systolic').

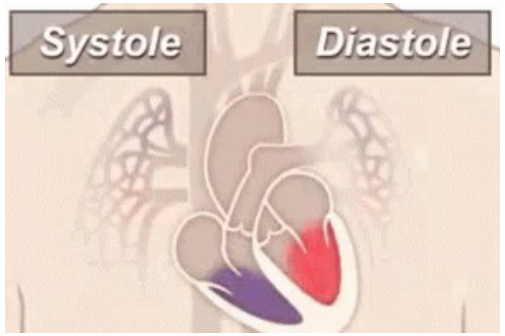


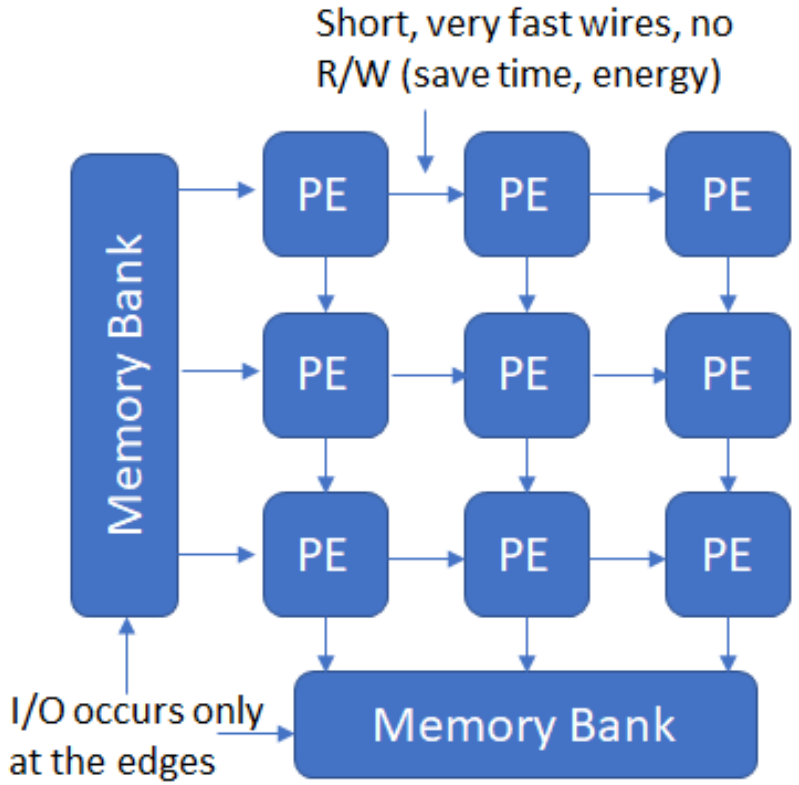


Source: telesens.co

Systolic Arrays

- **Advantages**:
  - High **throughput** due to parallel processing.
  - Efficient local data reuse, reducing memory bandwidth requirements.
  - Predictable performance due to regular data movement patterns.
- **Applications**:
  - Matrix multiplications (used in deep learning).
  - VLSI design for signal processing.





Source: telesens.co

Matrix Multiplication

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix} \quad C = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

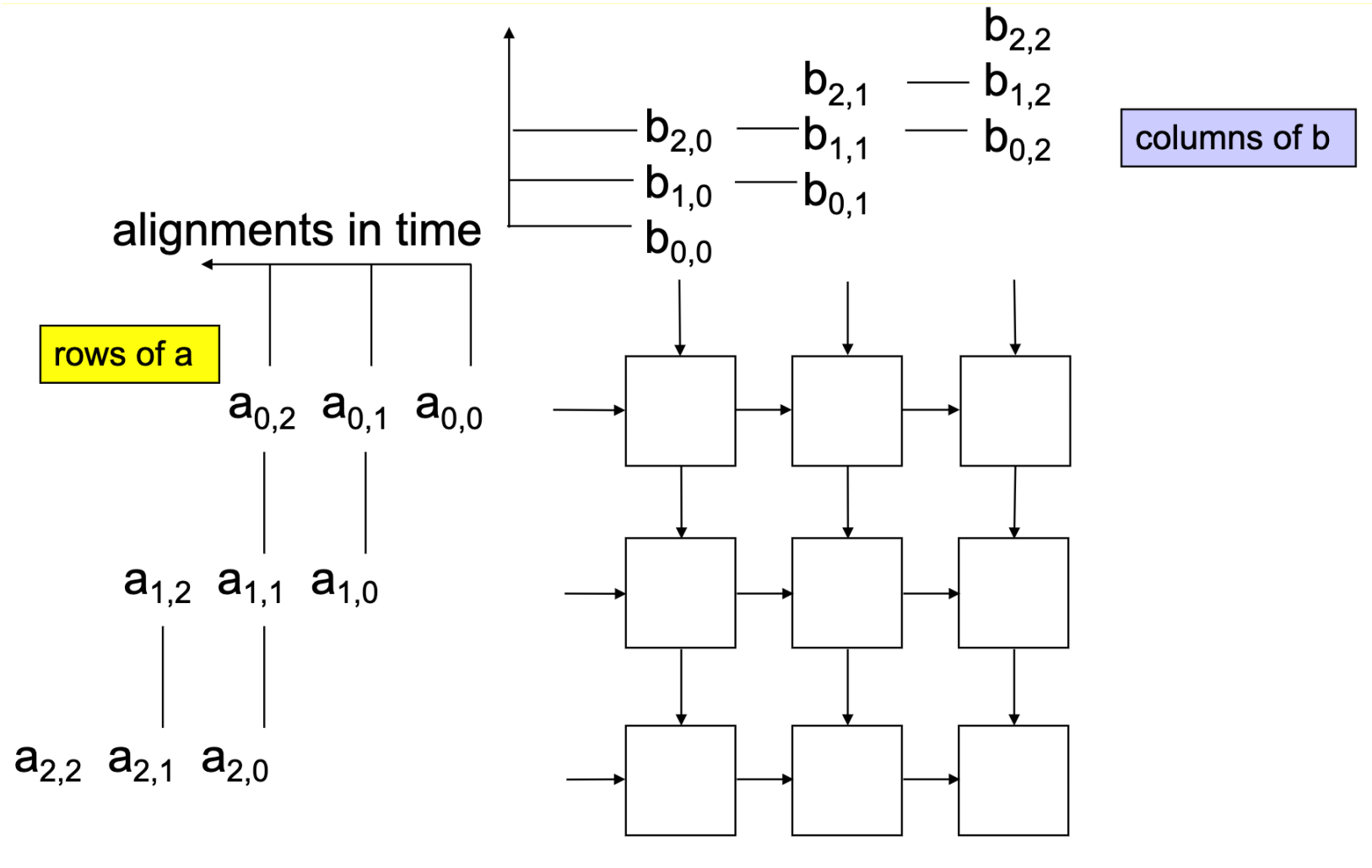
$$\begin{aligned} c_{00} &= a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} \\ c_{01} &= a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} \\ c_{02} &= a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ c_{10} &= a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} \\ c_{11} &= a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ c_{20} &= a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} \\ c_{21} &= a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

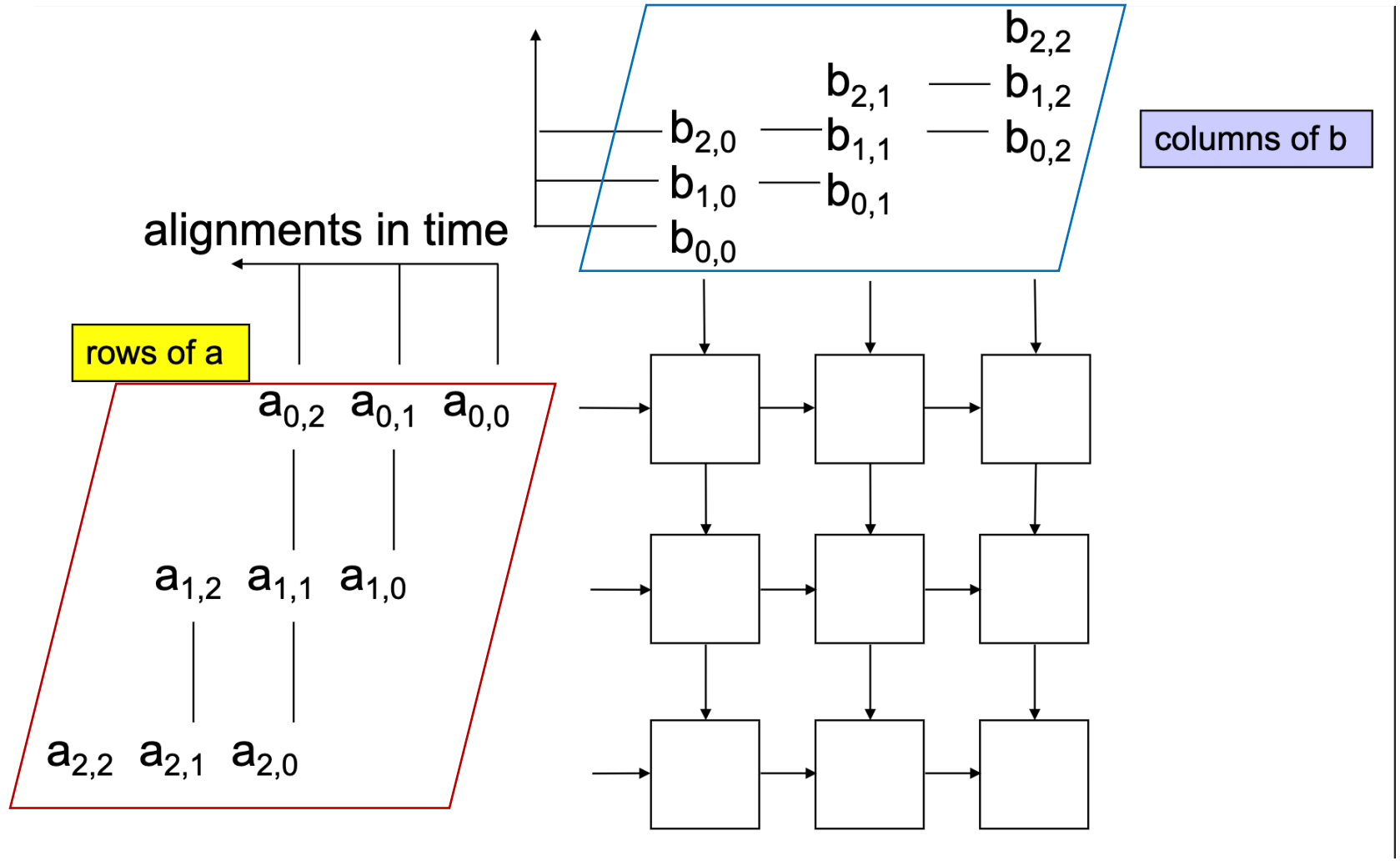


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

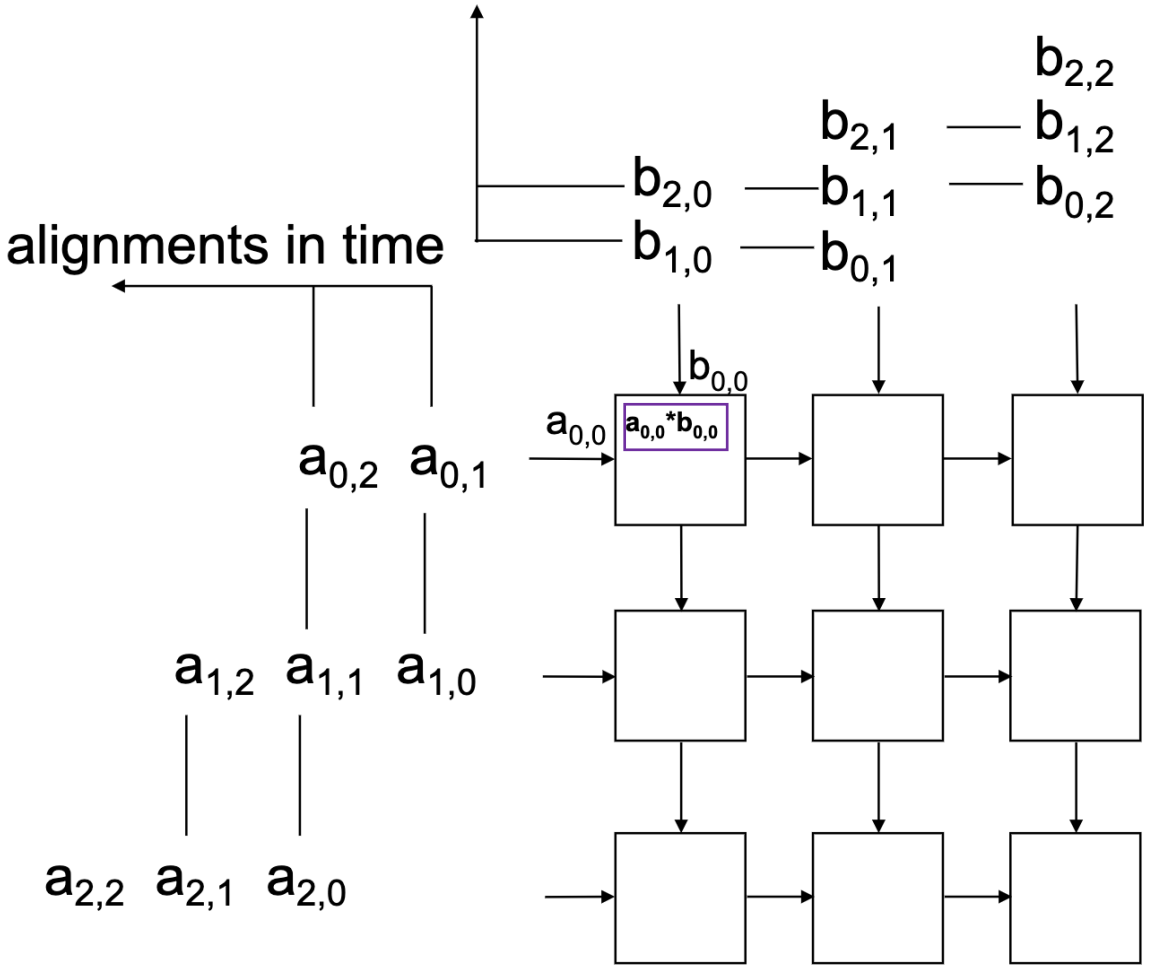


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

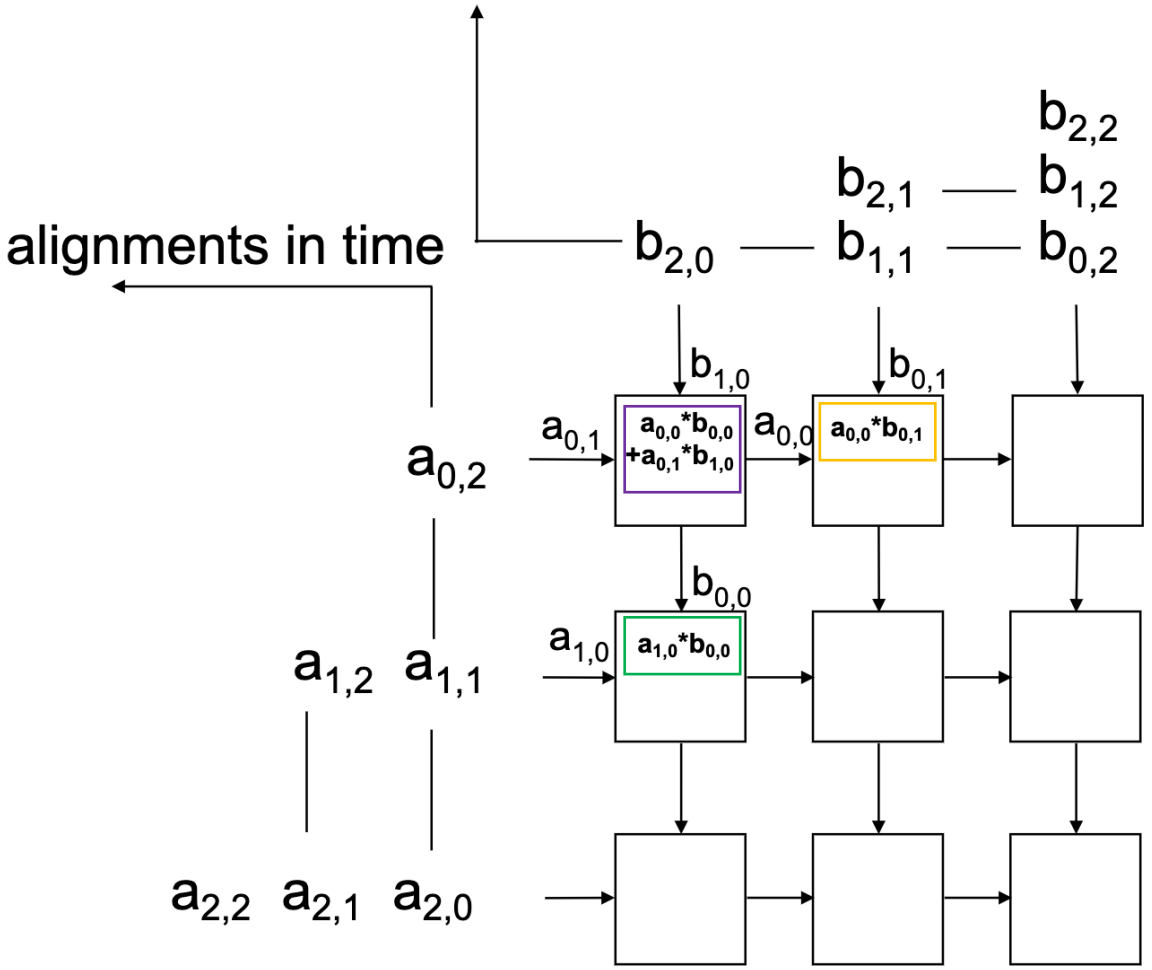


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

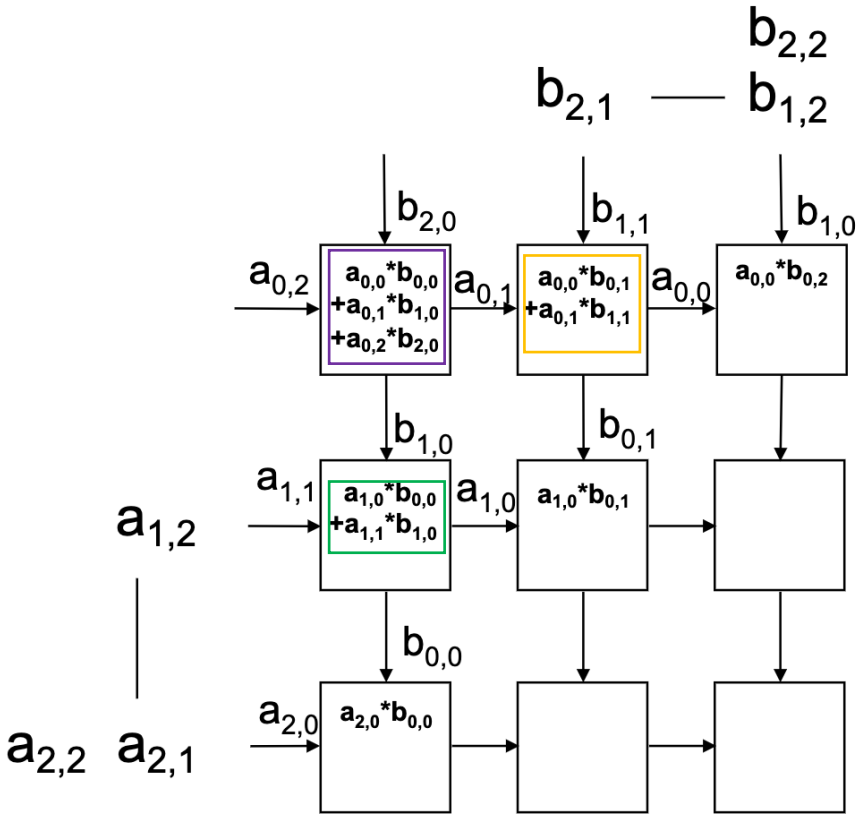


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

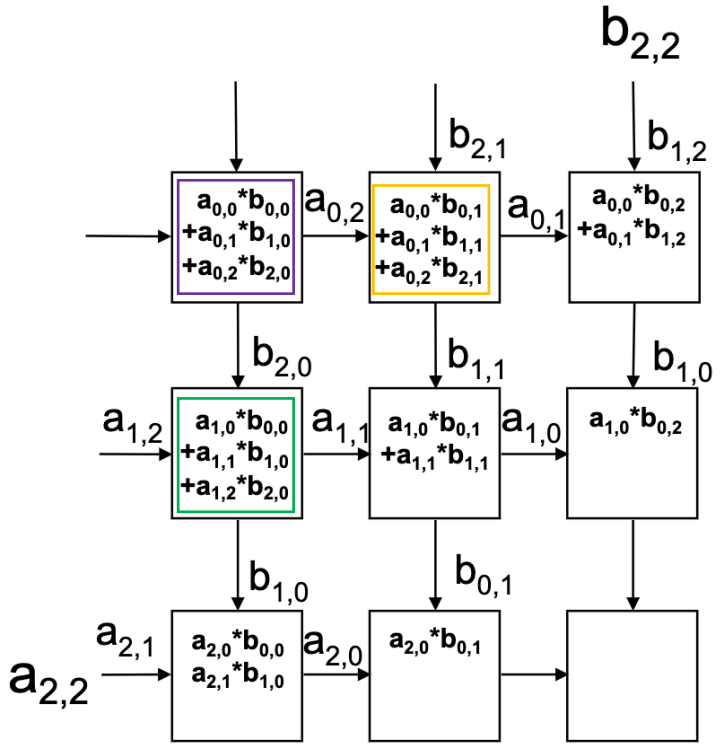


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

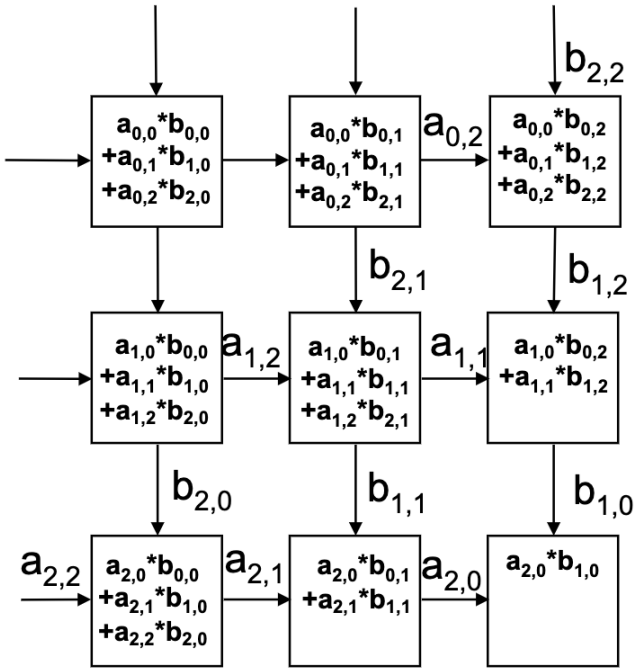


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$

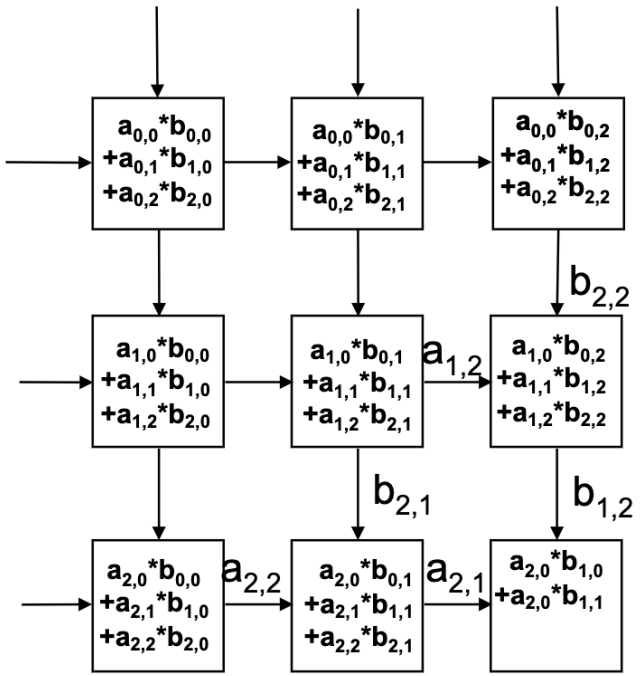


Matrix Multiplication

$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$\begin{aligned} c_{00} &= a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} \\ c_{01} &= a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} \\ c_{02} &= a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ c_{10} &= a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} \\ c_{11} &= a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ c_{20} &= a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} \\ c_{21} &= a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

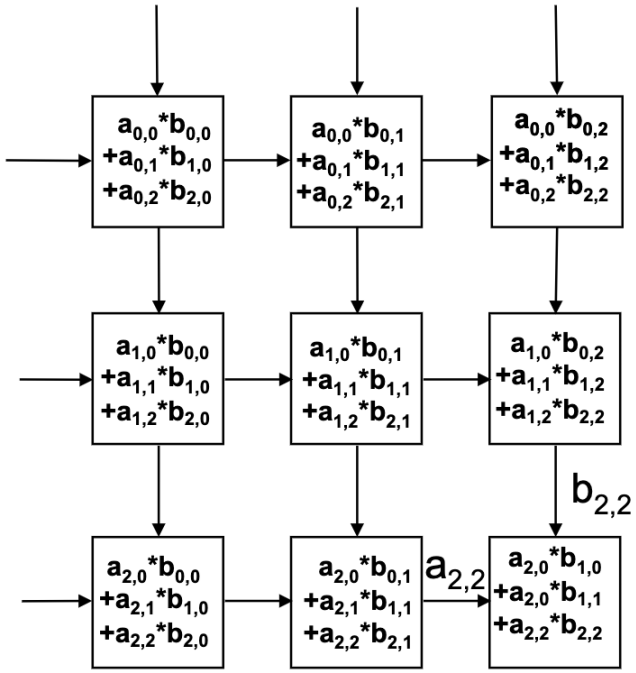


Matrix Multiplication

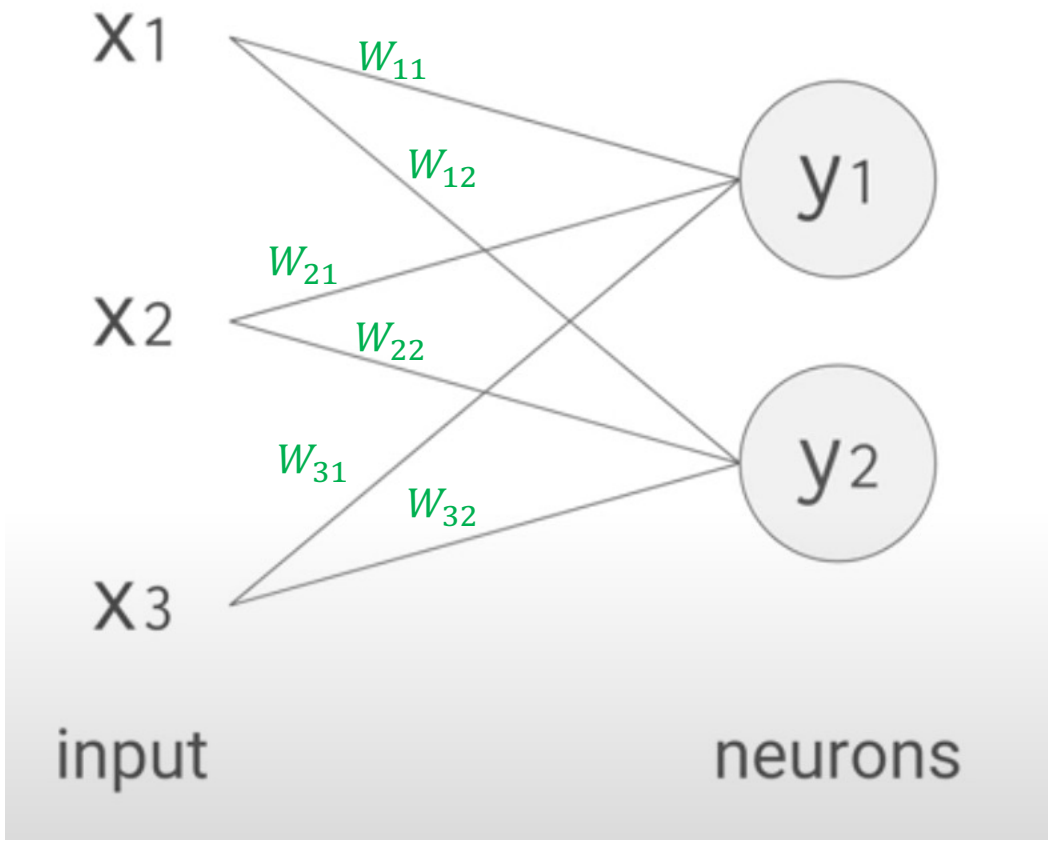
$$C = \begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22} \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22} \\ a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20} & a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21} & a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \quad B = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{bmatrix}$$

$$c_{00} = a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20}$$
$$c_{01} = a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21}$$
$$c_{02} = a_{00}b_{02} + a_{01}b_{12} + a_{02}b_{22}$$
$$c_{10} = a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20}$$
$$c_{11} = a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{10}b_{02} + a_{11}b_{12} + a_{12}b_{22}$$
$$c_{20} = a_{20}b_{00} + a_{21}b_{10} + a_{22}b_{20}$$
$$c_{21} = a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{20}b_{02} + a_{21}b_{12} + a_{22}b_{22}$$



Matrix Multiplication in DNNs

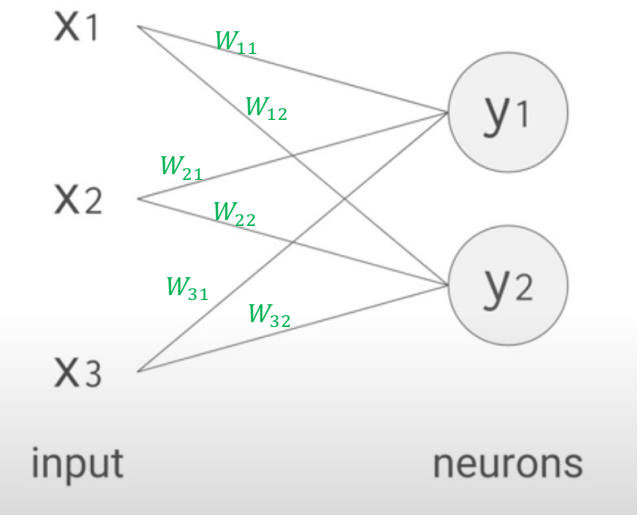


$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$

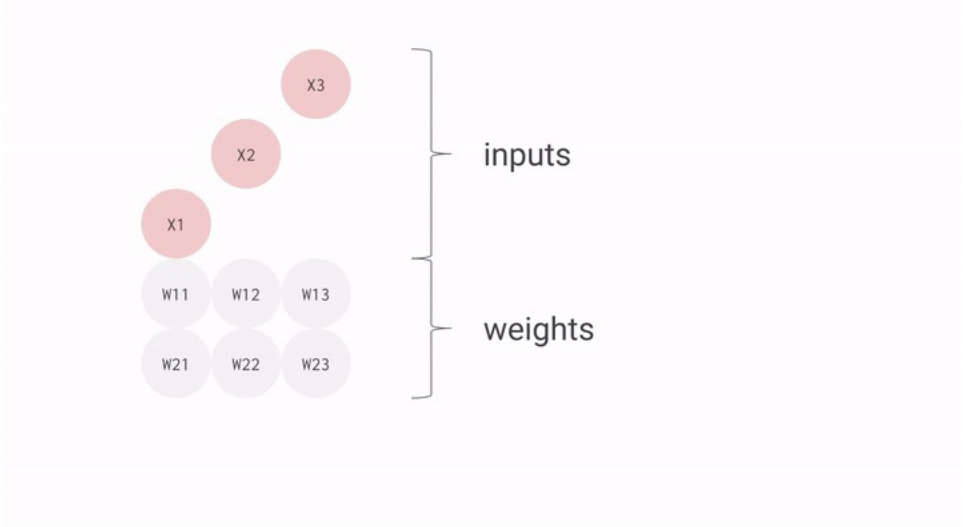
Matrix Multiplication in DNNs



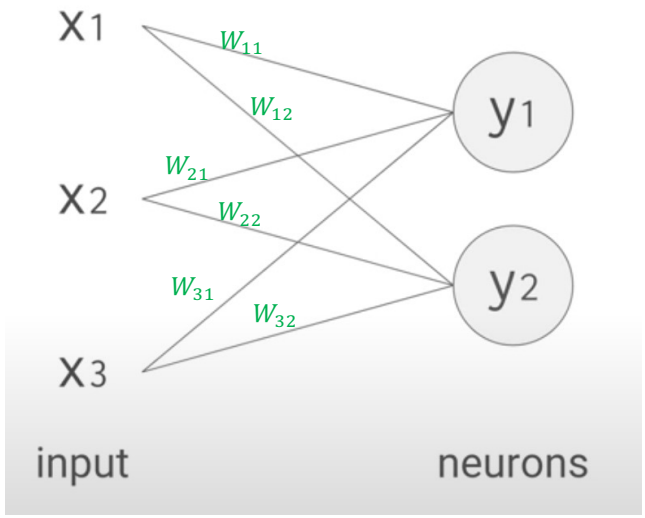
$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



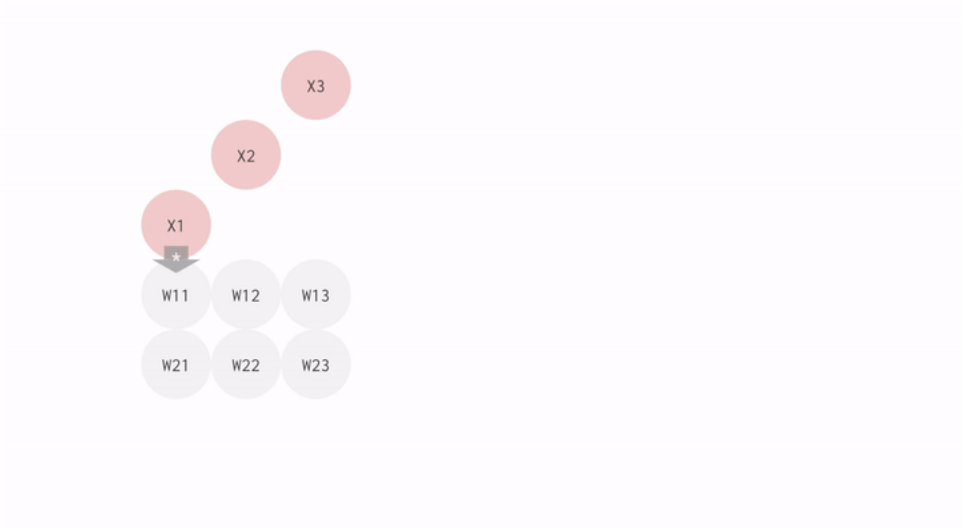
Matrix Multiplication in DNNs



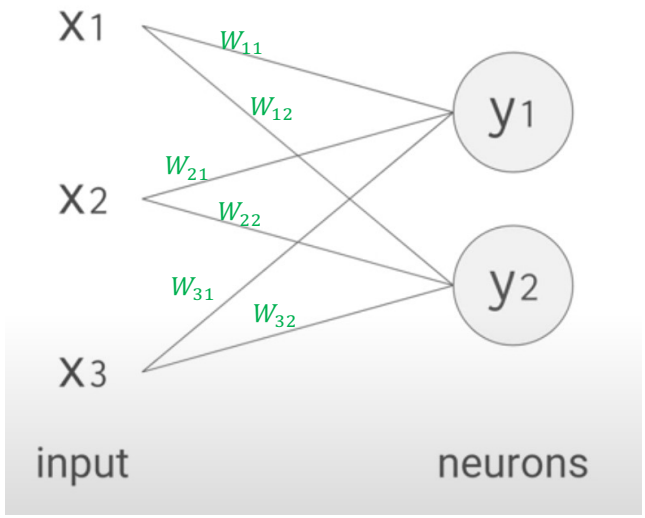
$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



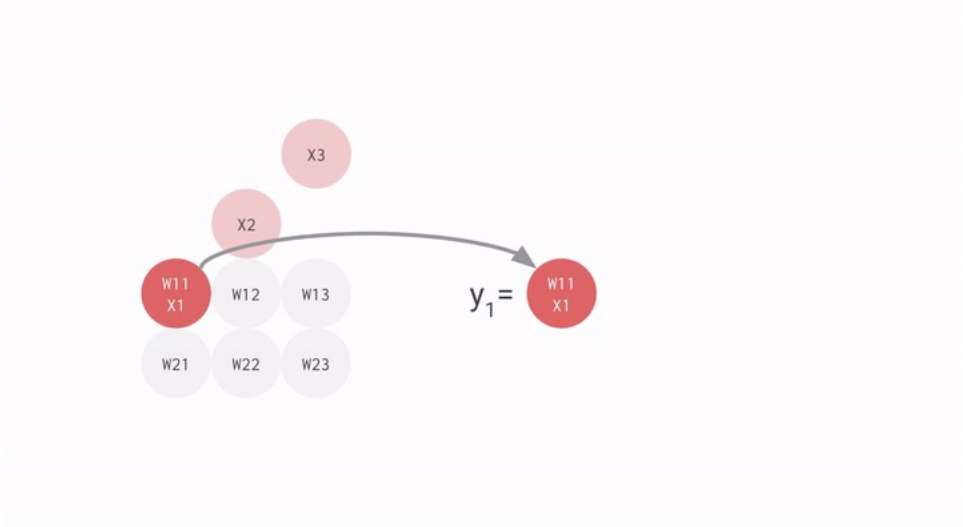
Matrix Multiplication in DNNs



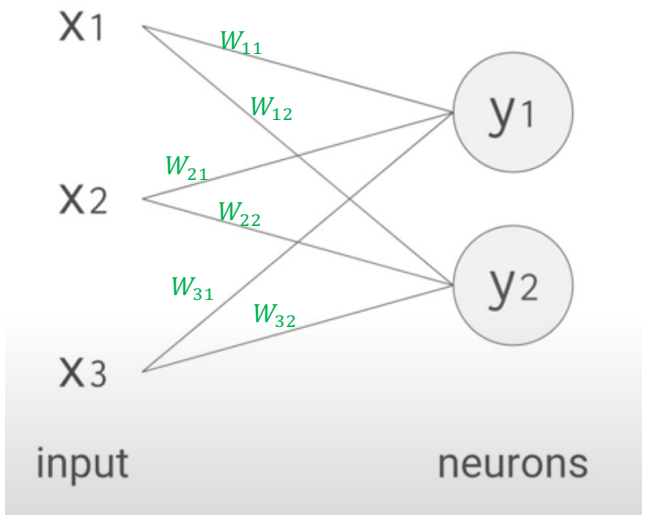
$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



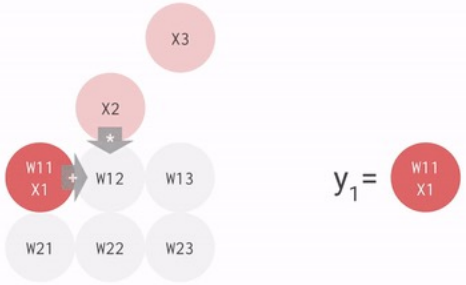
Matrix Multiplication in DNNs



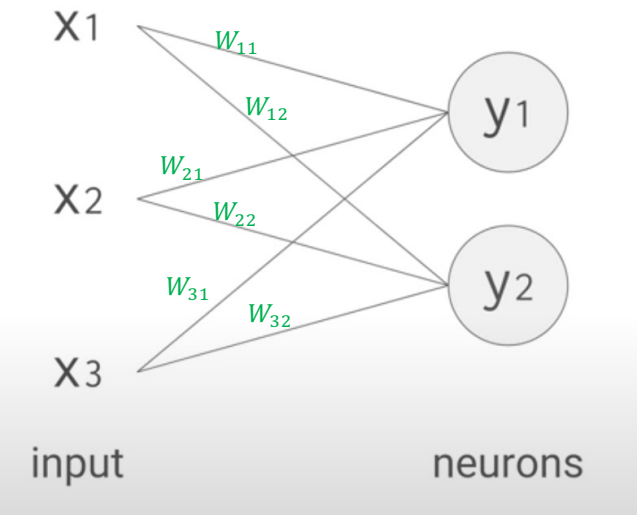
$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



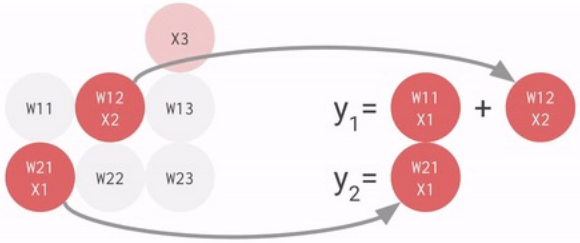
Matrix Multiplication in DNNs



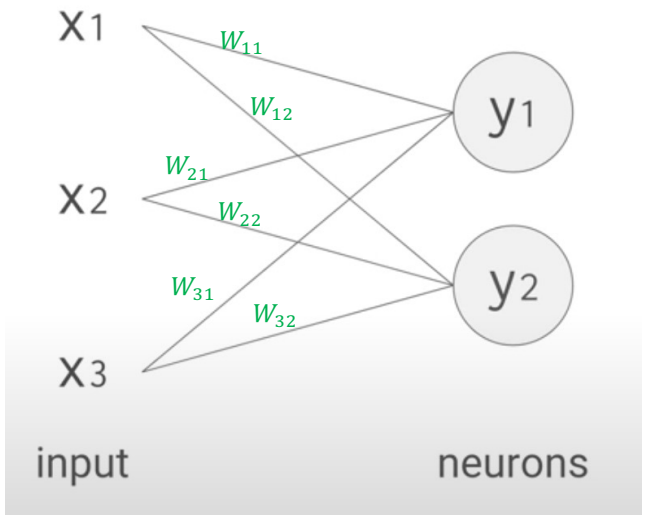
$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



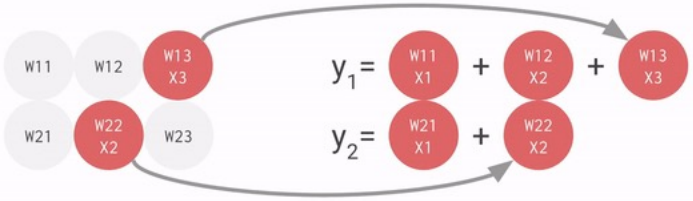
Matrix Multiplication in DNNs



$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$

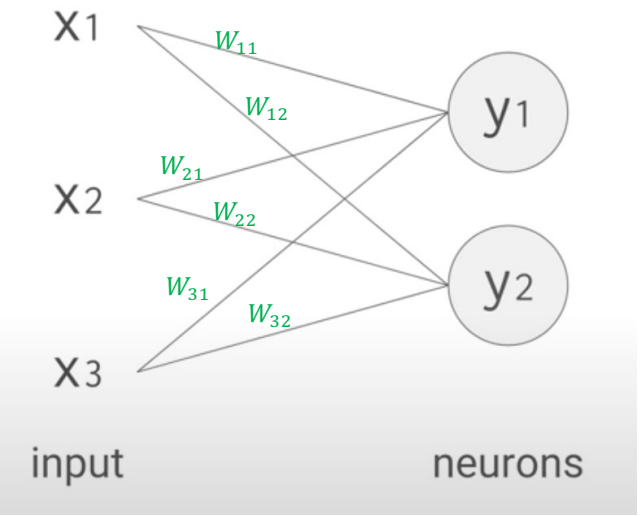


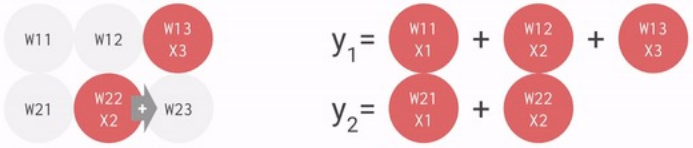
Matrix Multiplication in DNNs

$$\boldsymbol{y = Wx}$$

$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

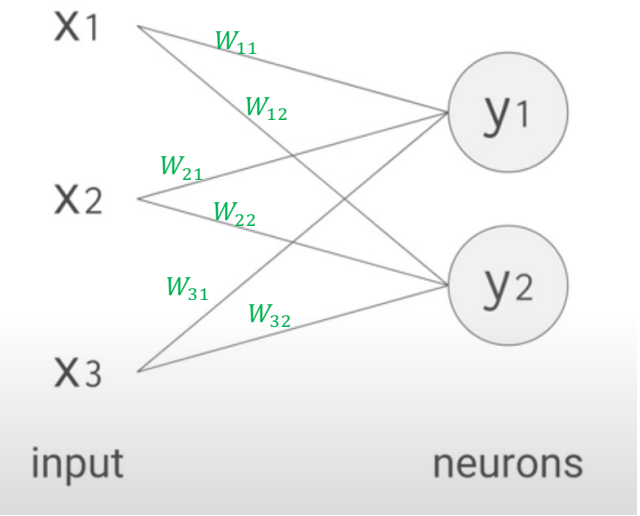
$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$





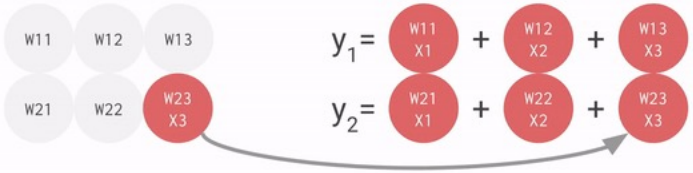
Matrix Multiplication in DNNs

$$\boldsymbol{y = Wx}$$



$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$

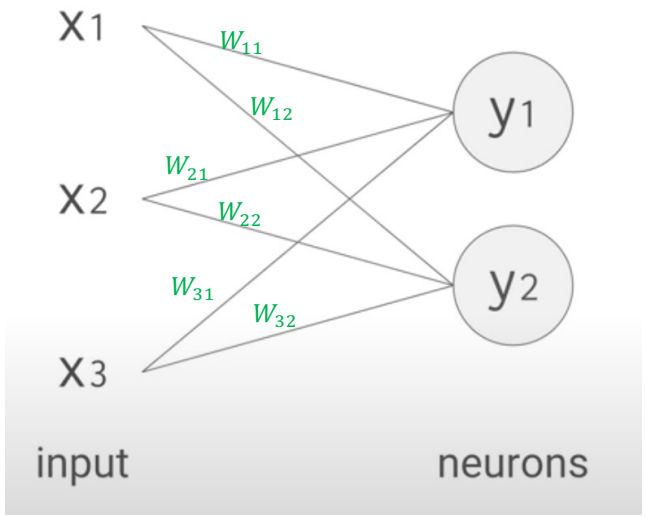


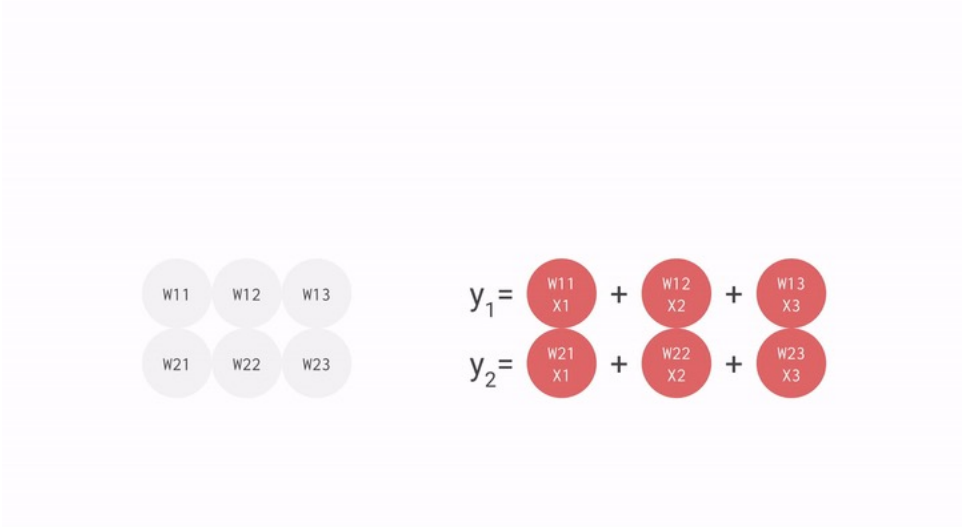
Matrix Multiplication in DNNs

$$\boldsymbol{y = Wx}$$

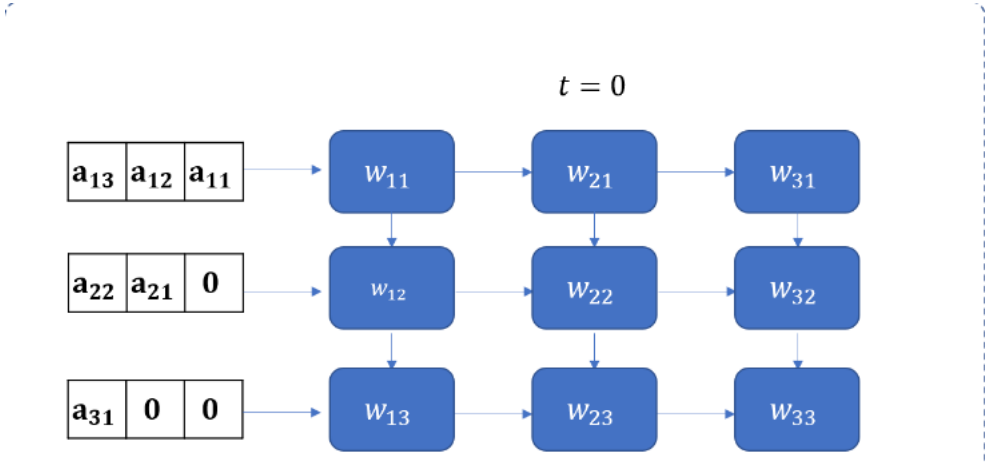
$$y_1 = W_{11}X_1 + W_{21}X_2 + W_{31}X_3$$

$$y_2 = W_{12}X_1 + W_{22}X_2 + W_{32}X_3$$



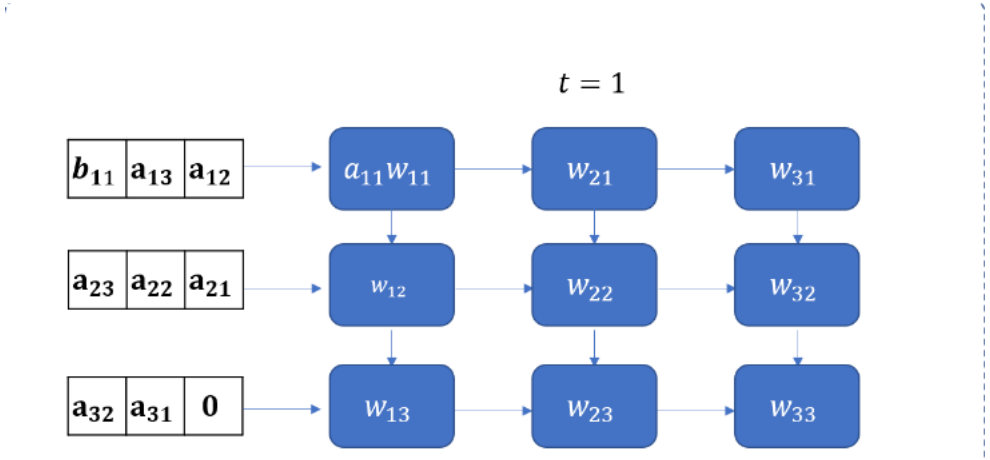


Matrix Multiplication



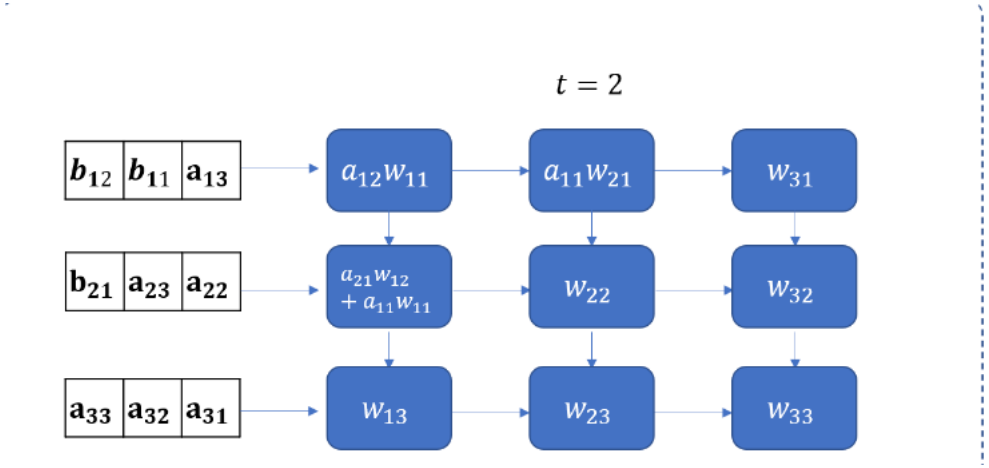
Source: telesens.co

Matrix Multiplication



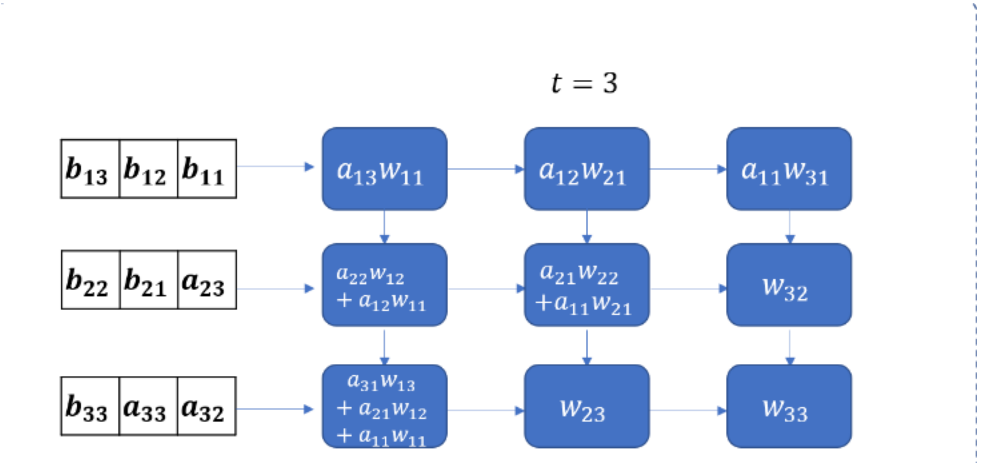
Source: telesens.co

Matrix Multiplication



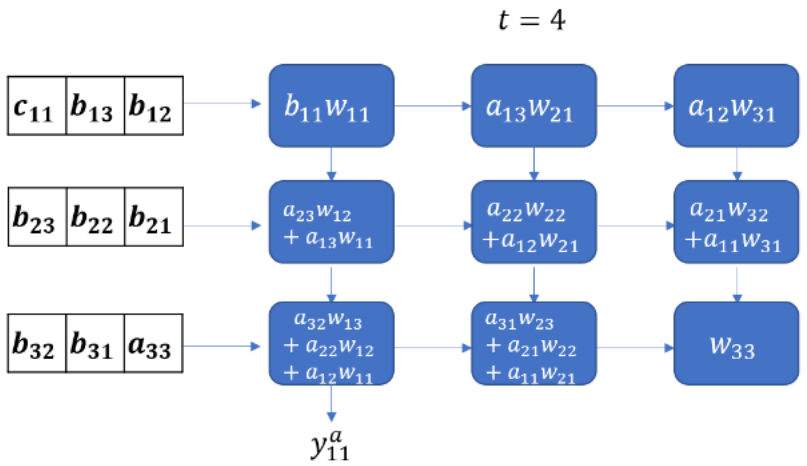
Source: telesens.co

Matrix Multiplication



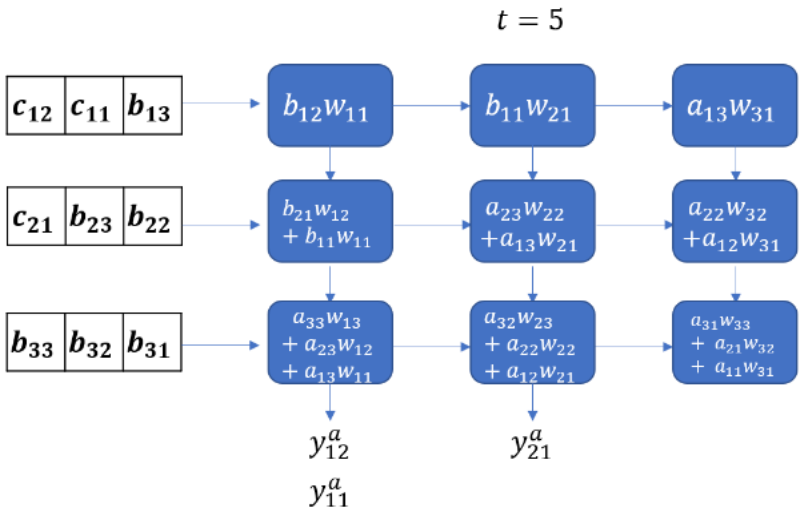
Source: telesens.co

Matrix Multiplication



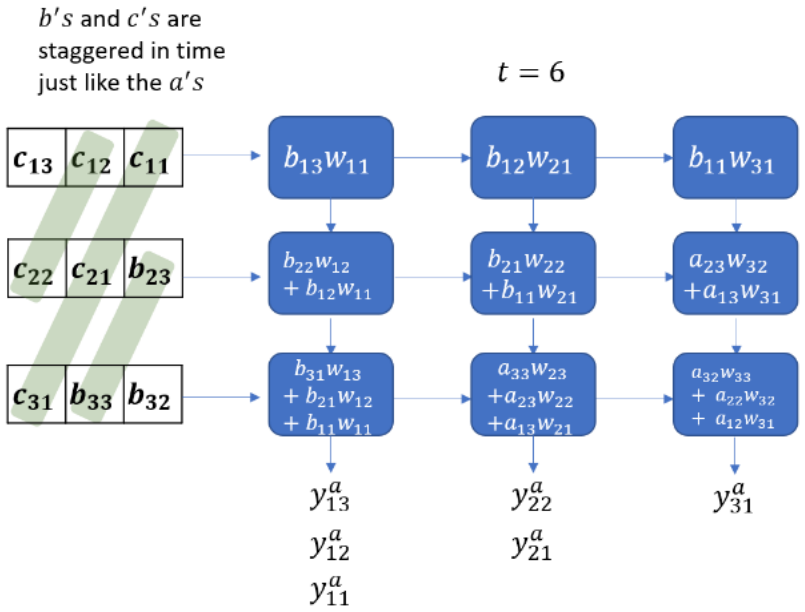
Source: telesens.co

Matrix Multiplication



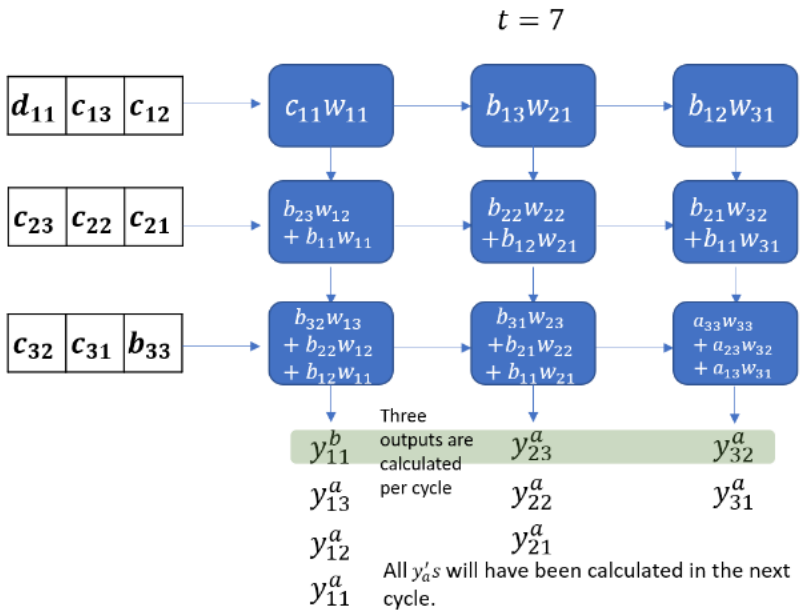
Source: telesens.co

Matrix Multiplication



Source: telesens.co

Matrix Multiplication



Source: telesens.co

Acknowledgement: Most diagrams are taken from V.Sze et al: Efficient Processing of Deep Neural Networks