JO Approach

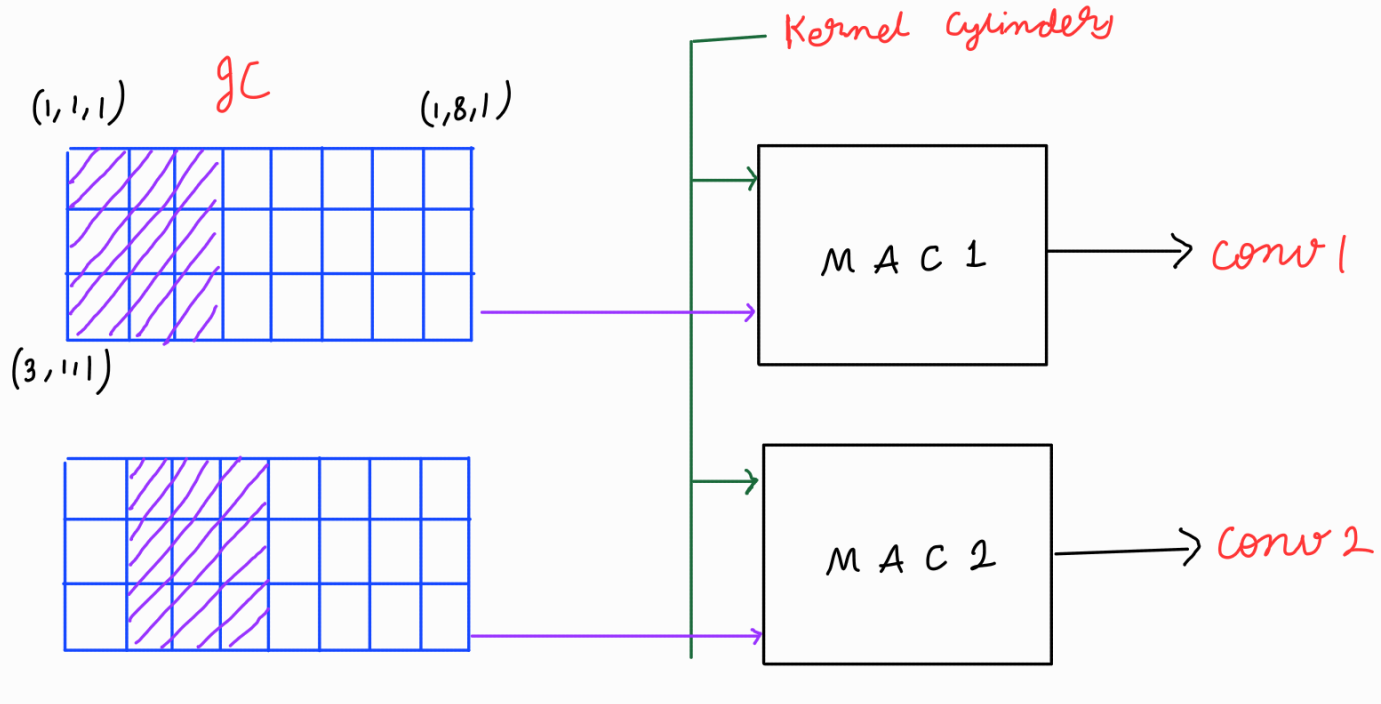
MAC Architecture :-

Let us use some numbering system for input cylinders

	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
Row 1	Cy 1	Cy 4	Cy 7	Cy 10	Cy 13	Cy 16	Cy 19	Cy 22
Row 2	Cy 2	Cy 5	Cy 8	Cy 11	Cy 14	Cy 17	Cy 20	Cy 23
Row 3	Cy 3	Cy 6	Cy 9	Cy 12	Cy 15	Cy 18	Cy 21	Cy 24

* Cy 1 : Input Cylinder 1 = Input[row = 1, col = 1, chn = 1 : 8]

→ Recall from our "Convolution Approach"



∴ so on

So, MAC 1 requires Cy 1 to Cy 9

MAC 2 requires Cy 3 to Cy 12

... so on

→ Let us assume we need 1 clock cycle to get a cylinder.

So, Cy 1 arrives at $t = 1$

Cy 2 arrives at $t = 2$

⋮

Cy x arrives at $t = x$

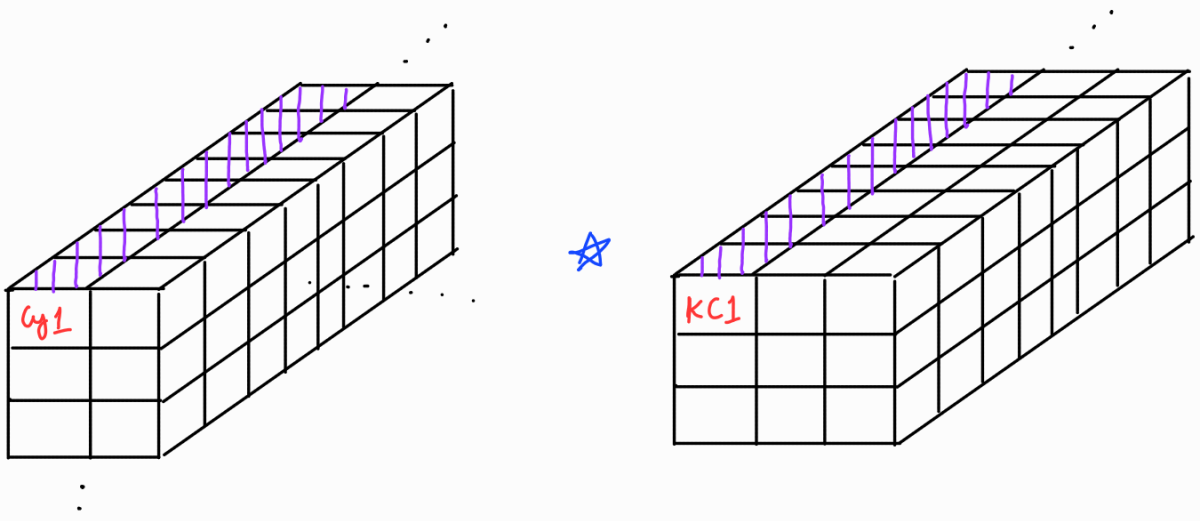
→ Also, no. of clock cycles required for half-precision multiplier } = 2 cycles

no. of clock cycles required for half-precision adder } = 1 cycle

i) Multiplier and Adder structure

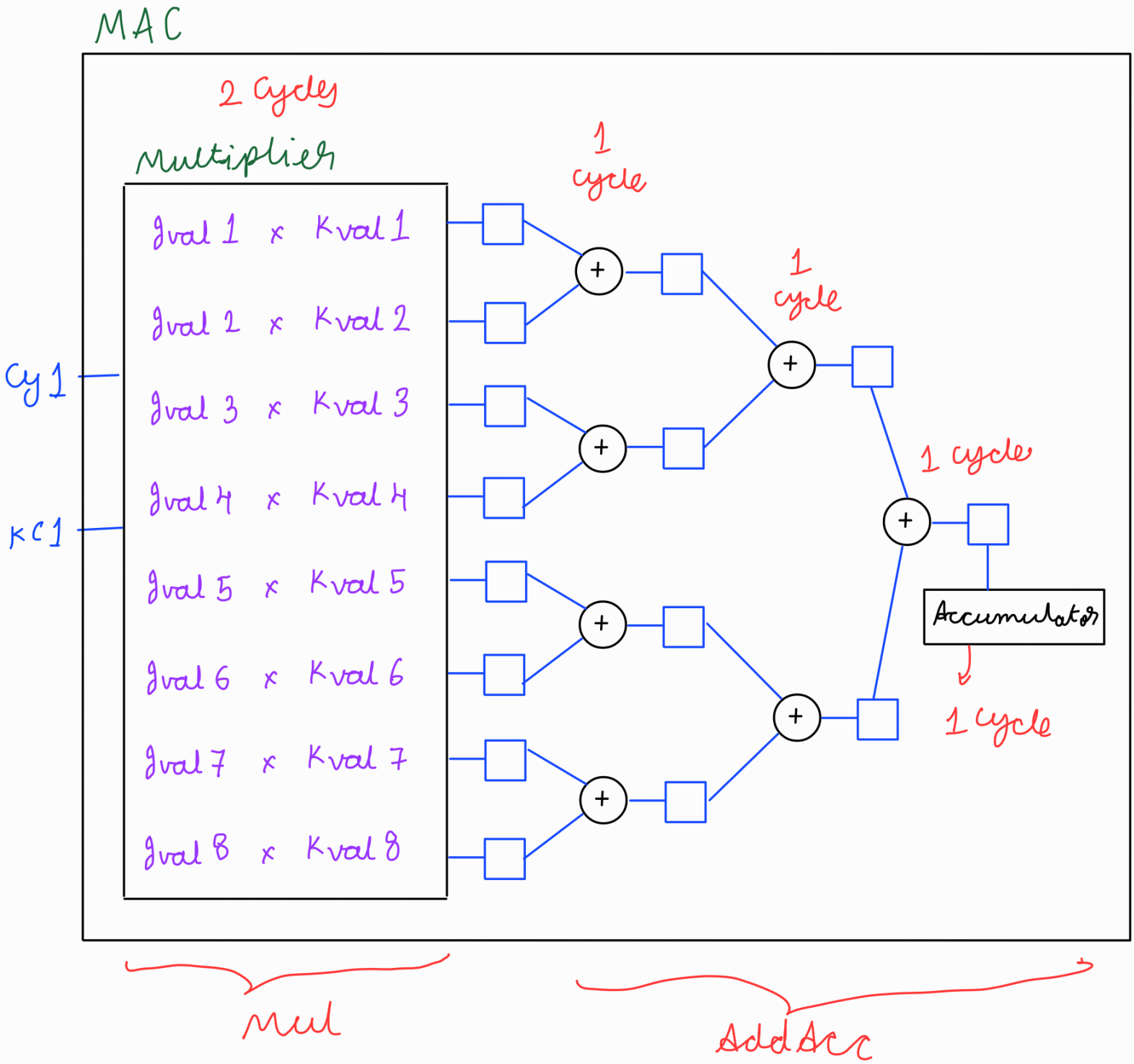
→ Multiplier and adder structure for convolution between "1" Input cylinder and "1" Kernel cylinder

Eg: Cy 1 * KC 1



Let Cy 1 = [fval 1, fval 2, fval 3, fval 4,
fval 5, fval 6, fval 7, fval 8]

KC1 = [Kval 1, Kval 2, KVal 3, Kval 4,
Kval 5, Kval 6, Kval 7, Kval 8]



Mul = 2 cycles

AddAcc = 4 cycles

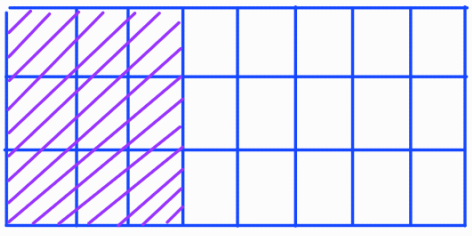
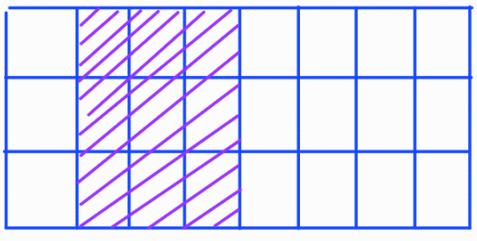
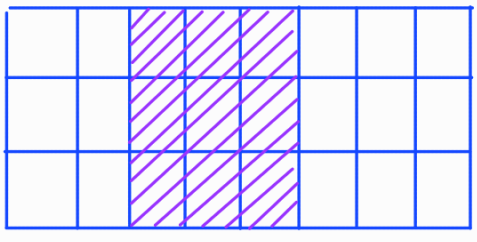
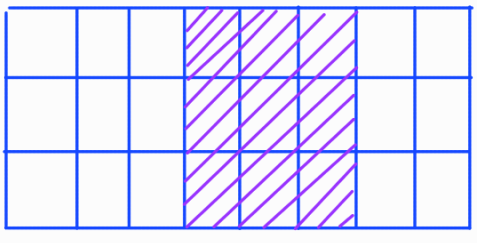
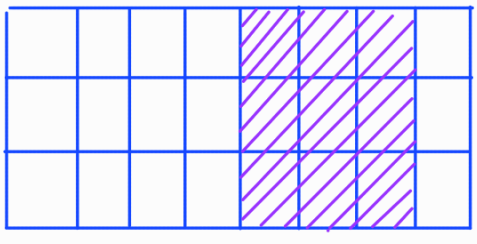
→ For MAC 1, the cylinders Cy 1 to Cy 9 will arrive one after another, so instead of starting MAC 1 after waiting for 9 cylinders, we can pipeline the convolution of 9 cylinders. This will reduce no. of multipliers per MAC to 8 and no. of adders per MAC to 8

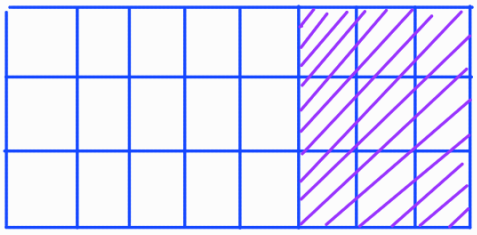
t : 0	Start		
t : 1	Cy 1		
t : 2	Cy 2		
t : 3	Cy 3	Mul 1	
t : 4	Cy 4	Mul 2	
t : 5	Cy 5	Mul 3	
t : 6	Cy 6	Mul 4	
t : 7	Cy 7	Mul 5	Add Acc 1
t : 8	Cy 8	Mul 6	Add Acc 2
t : 9	Cy 9	Mul 7	Add Acc 3
t : 10		Mul 8	Add Acc 4
t : 11		Mul 9	Add Acc 5
t : 12			Add Acc 6
t : 13			Add Acc 7
t : 14			Add Acc 8
t : 15			Add Acc 9

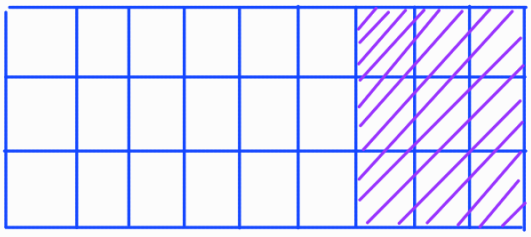
Hence, it takes 15 cycles for a MAC to calculate output once first cylinder arrives.

→ Let us calculate effect of no. of MAC on total cycles required for 36 output values

Case 1 :- 4 MACs

		Start	End
Iter 1			
MAC 1		t = 1	t = 15
MAC 2		t = 4	t = 18
MAC 3		t = 7	t = 21
MAC 4		t = 10	t = 24
Iter 2			
MAC 1		t = 13	t = 27

MAC 2  t = 16 t = 30

MAC 3  t = 19 t = 33

MAC 4 t = 22 t = 36

. . . . . .

iter 9

MAC 4 t = 106 t = 120

→ Hence for 4 MACs, it will take 120 cycles to get 36 output values.

Similarly

For 5 MAC's → 117 cycles for 35 output

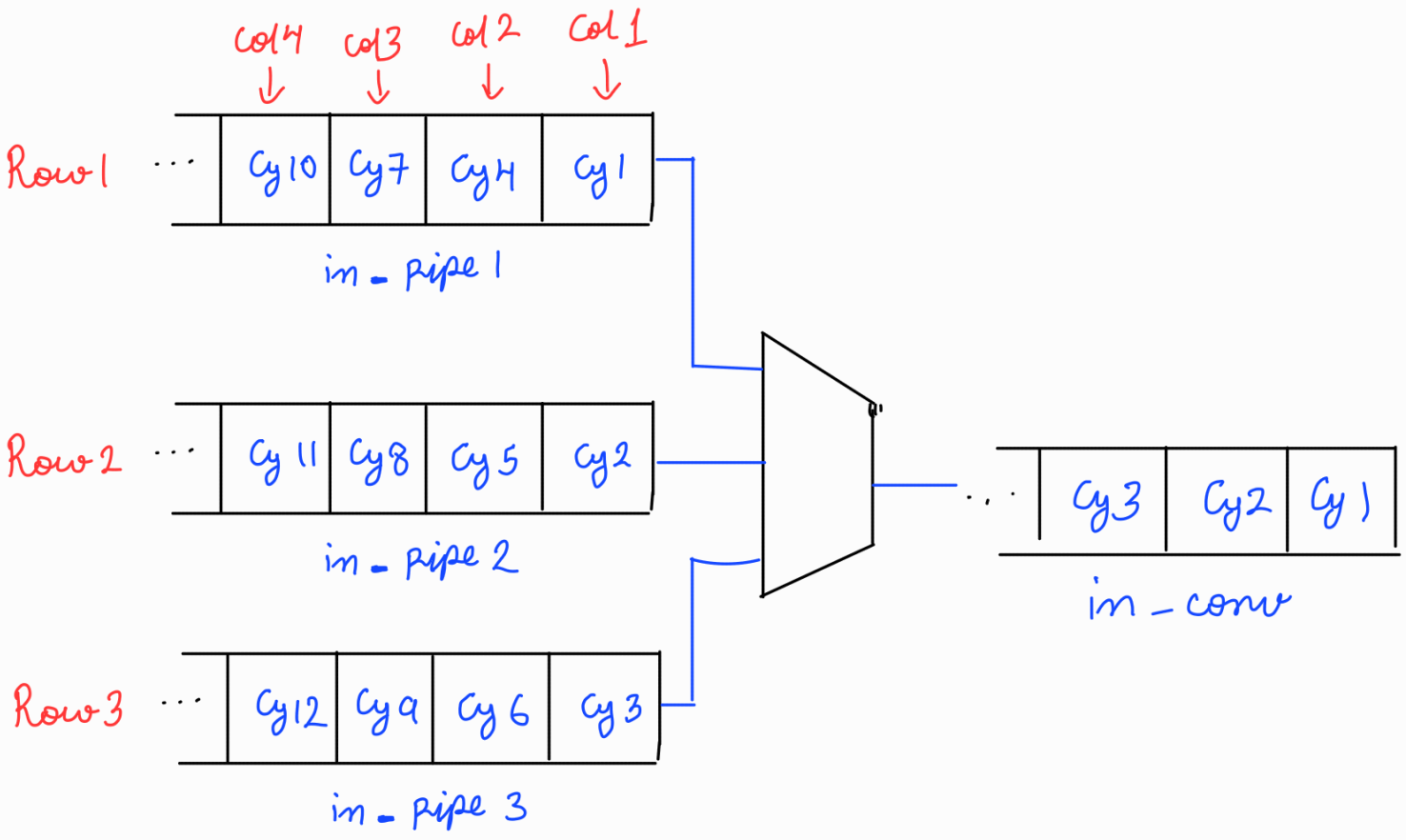
For 6 MAC's → 120 cycles for 36 output

For 7 MAC's → 117 cycles for 35 output

Input Module :-

Let us continue with 4 MAC blocks per engine

For storing input, let us use 3 pipes, each pipe stores the data for 1 row

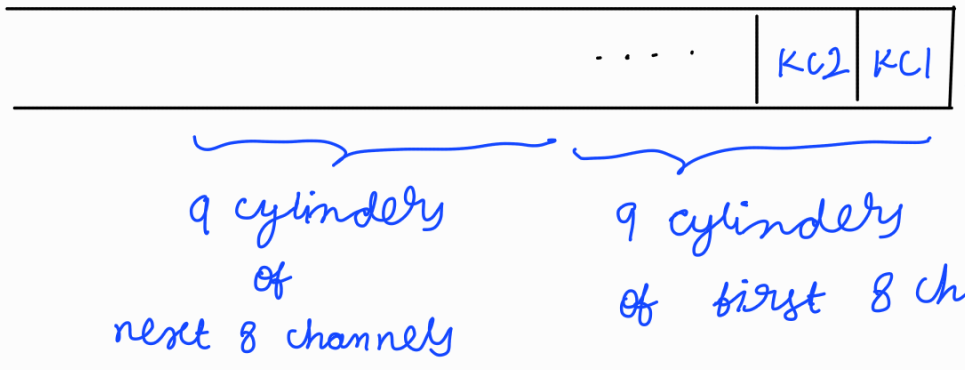


→ A diagramatic representation for distribution of "in_conv" data to MAC buffers

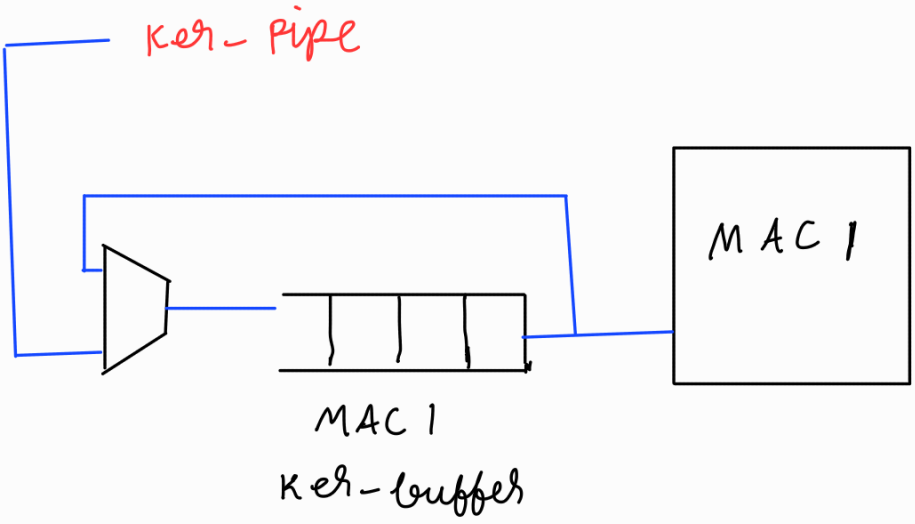
	iter 1	iter 2	iter 3	iter 4	iter 5	iter 6
MAC 1 in_buffer	Cy 1-3	Cy 4-6	Cy 7-9		Cy 13-15	Cy 16-18
MAC 2 in_buffer		Cy 4-6	Cy 7-9	Cy 10-12		Cy 16-18
MAC 3 in_buffer			Cy 7-9	Cy 10-12	Cy 13-15	
MAC 4 in_buffer				Cy 10-12	Cy 13-15	Cy 16-18

Kernel Module :-

→ Similar to Aman's approach, we will store entire kernel in a pipe.



→ Storage of these kernels in MAC buffers.



Output Module :-

→ We have output buffer of depth 36, we can send these values directly to the memory.