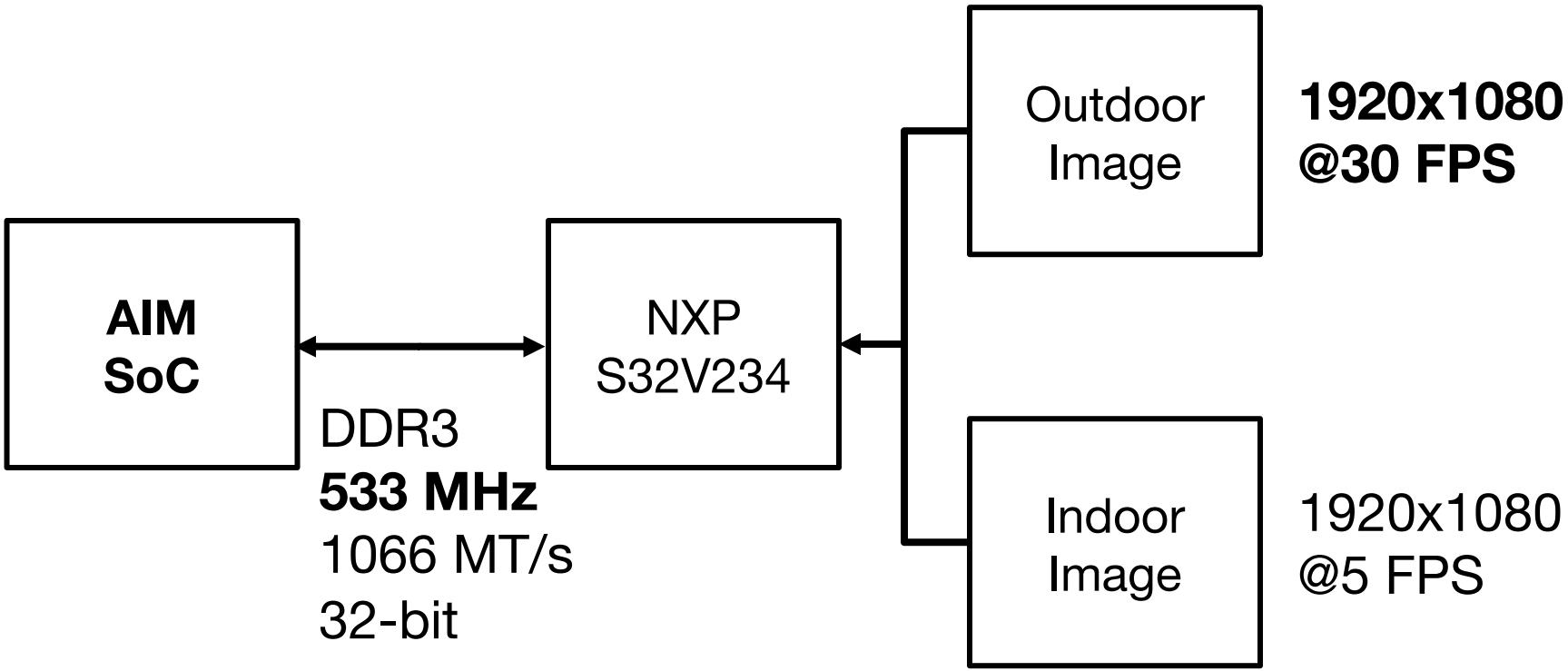
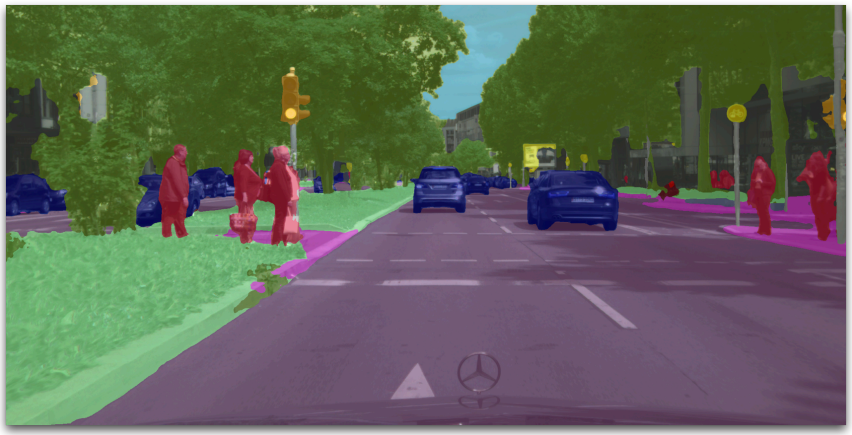
AIM SoC Spec.


AIM
SoC
DDR3
533 MHz
1066 MT/s
32-bit
NXP
S32V234
Outdoor
Image
1920x1080
@30 FPS
Indoor
Image
1920x1080
@5 FPS

Semantic Segmentation

Cityscapes dataset

Inference Result



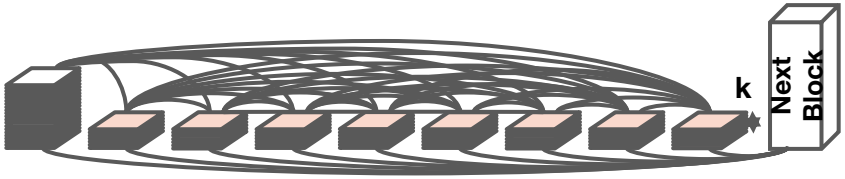
Fully Convolutional Network

FCN Model

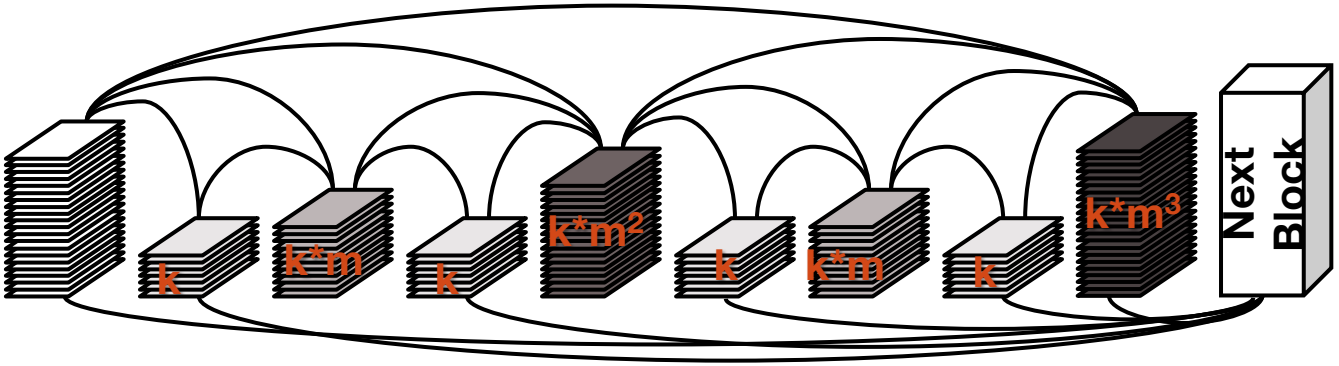
FCN use convolutional layers to classify every pixel and to determine the context of the image.

HarDNet (ICCV 2019)

DenseNet
(CVPR 2017)



- Reduced shortcut connection
- Balanced input/output channel ratio
- **Accuracy over Memory Traffic and Low MACs**

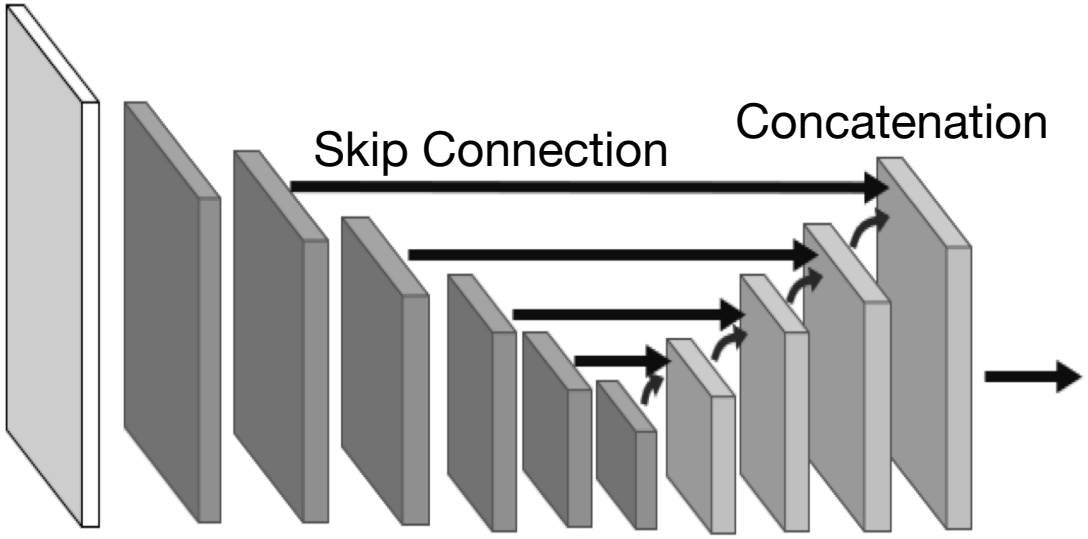


Highlight

Memory Traffic ↘

GPU Runtime ↘

U-HarDNet Model



U-Net Architecture

1,920x1,080

960x540

480x270

240x135

120x67

60x33

30x16

Div 2

Operations:

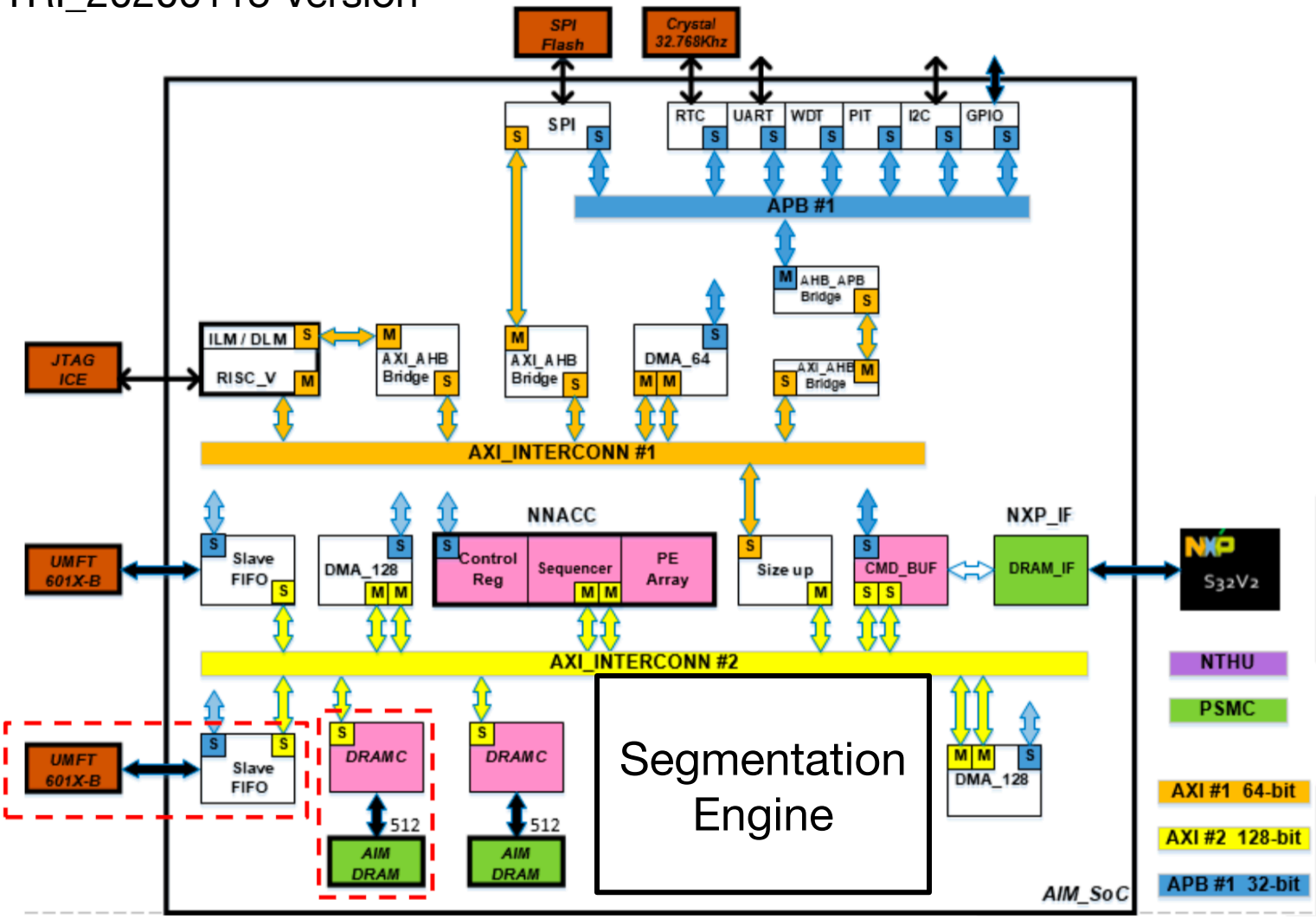
- 3x3, stride 2
- 3x3, stride 1
- 1x1, stride 1
- Average Pooling
- Bilinear Interpolation
- Concatenation

Support Operations

- **Focus on convolution operation, we can support following types:**
  - 1 x 1 filter, stride 1
  - 3 x 3 filter, stride 1, 2
  - 5 x 5 filter, stride 1, 2
  - 7 x 7 filter, stride 1, 2
- **Max Pooling / Average Pooling**
  - 2 x 2 filter, stride 2
  - 3 x 3 filter, stride 2
- **Activation Function**
  - Relu, Tanh, Sigmoid Function
- **Skip Connection**
  - Concatenation

AIM SoC Architecture

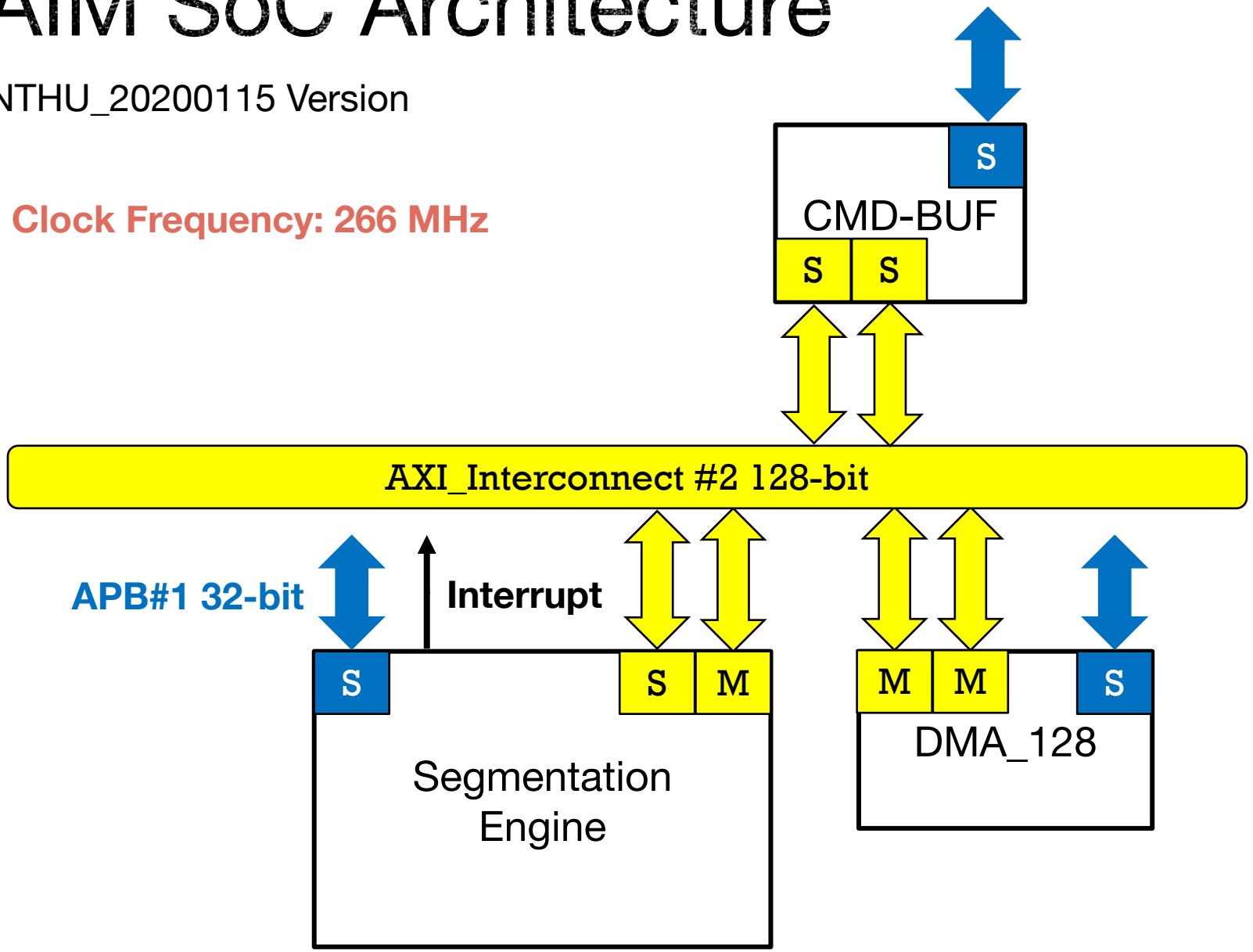
ITRI_20200115 Version



AIM SoC Architecture

NTHU_20200115 Version

Clock Frequency: 266 MHz

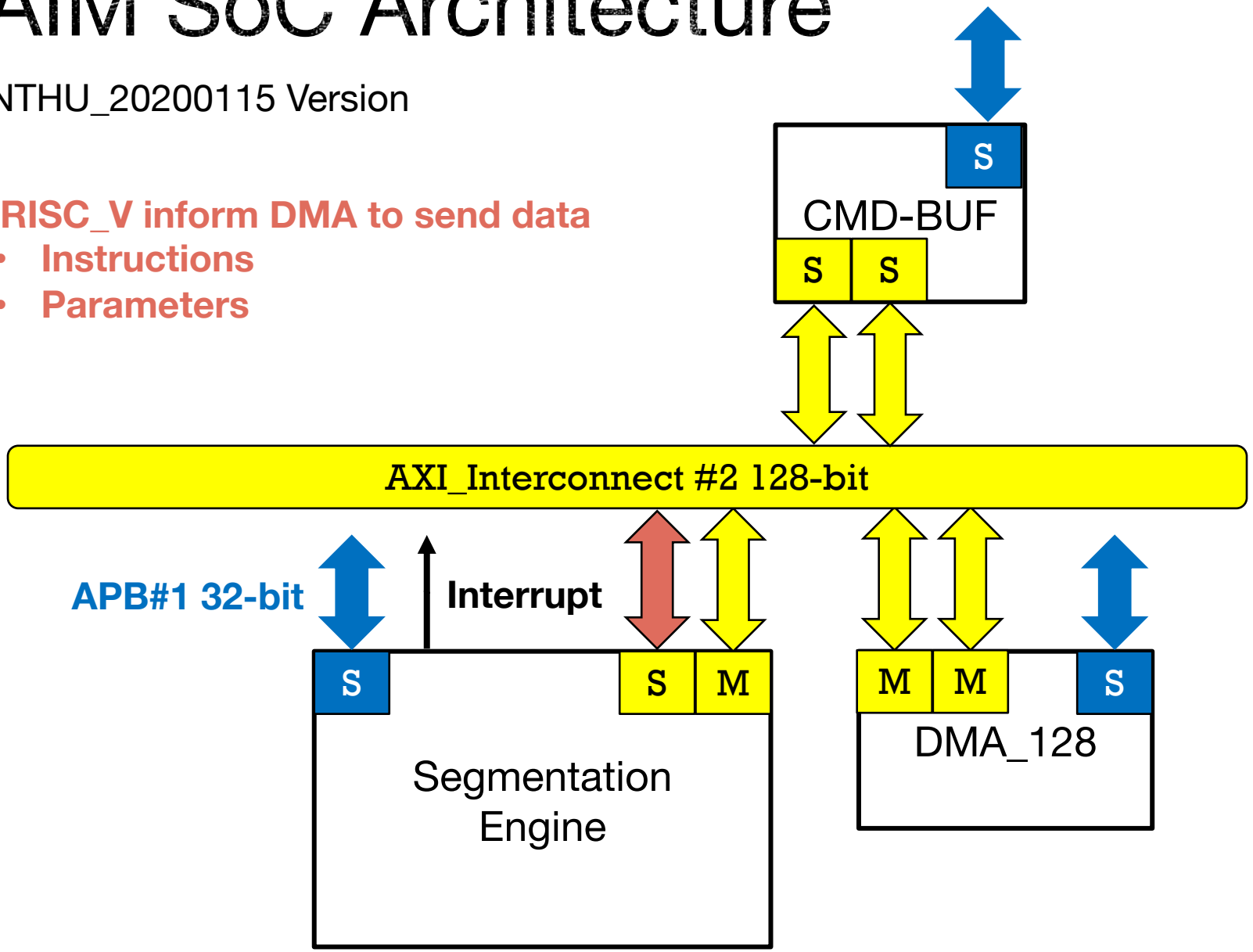


AIM SoC Architecture

NTHU_20200115 Version

RISC_V inform DMA to send data

- **Instructions**
- **Parameters**

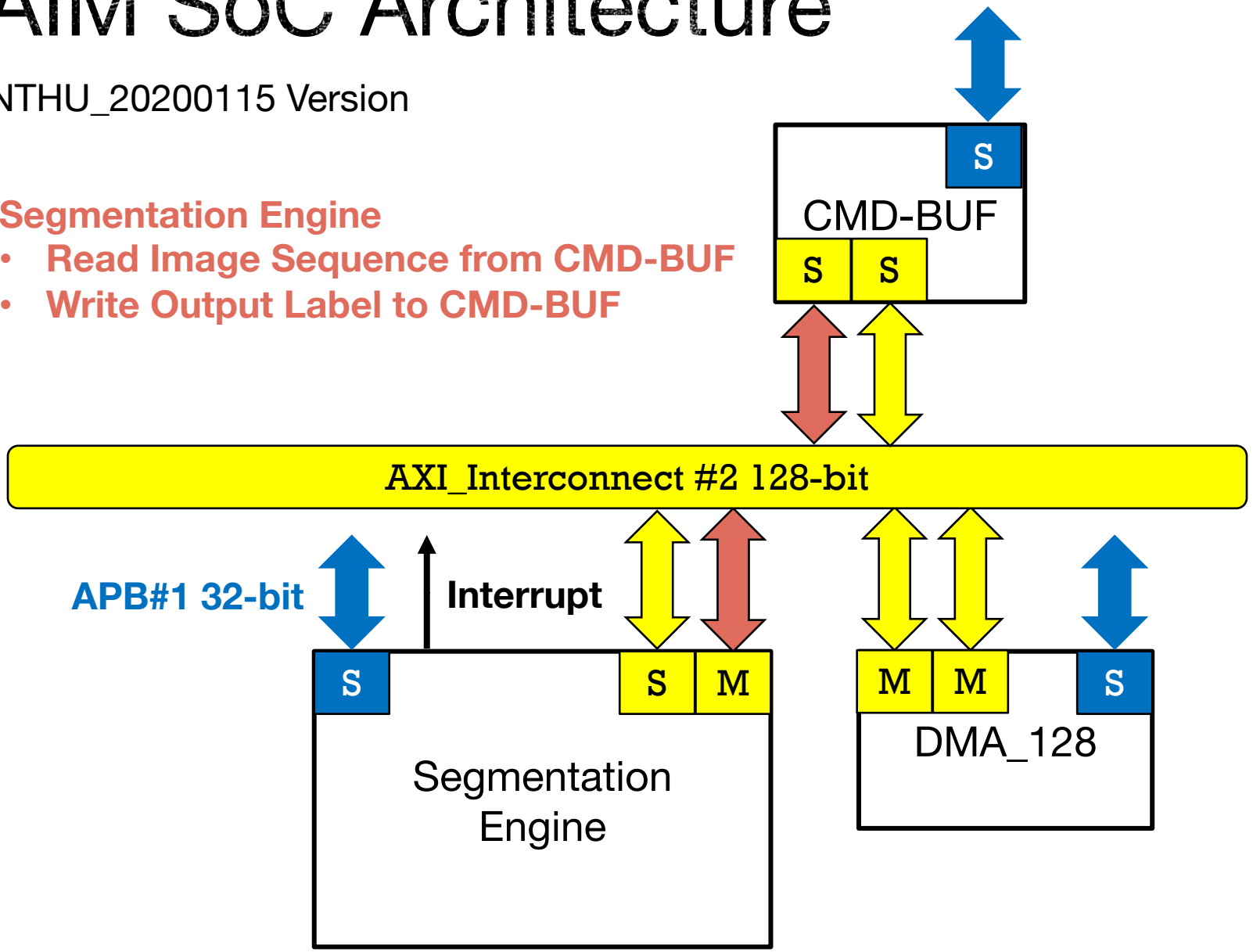


AIM SoC Architecture

NTHU_20200115 Version

Segmentation Engine

- **Read Image Sequence from CMD-BUF**
- **Write Output Label to CMD-BUF**



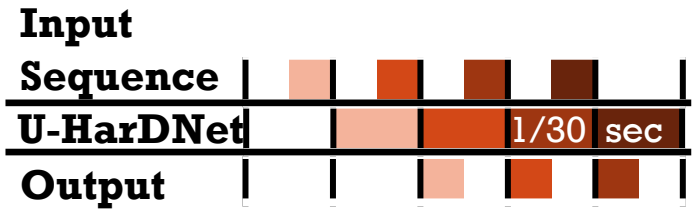
Program Sequence

Start

Initialization Engine

Ctrl Reg:

1. Software RESET
2. AIM - Power On
3. AIM - Base Address Setting
4. Engine Configurations
5. Start U-HarDNet Engine



Segmentation (SEG) Engine

Interrupt

Check SEG Engine Status

Is Done ?

Y

N

Update Data Status

Recv Data

Instructions Parameters

Input Sequence

Output Label

Inform DMA Engine Fetch Data from Flash

Inform SEG Engine Fetch Data from CMD_BUF

Inform SEG Engine Write Data to CMD_BUF

Activate Data Mover

Send/Recv Data

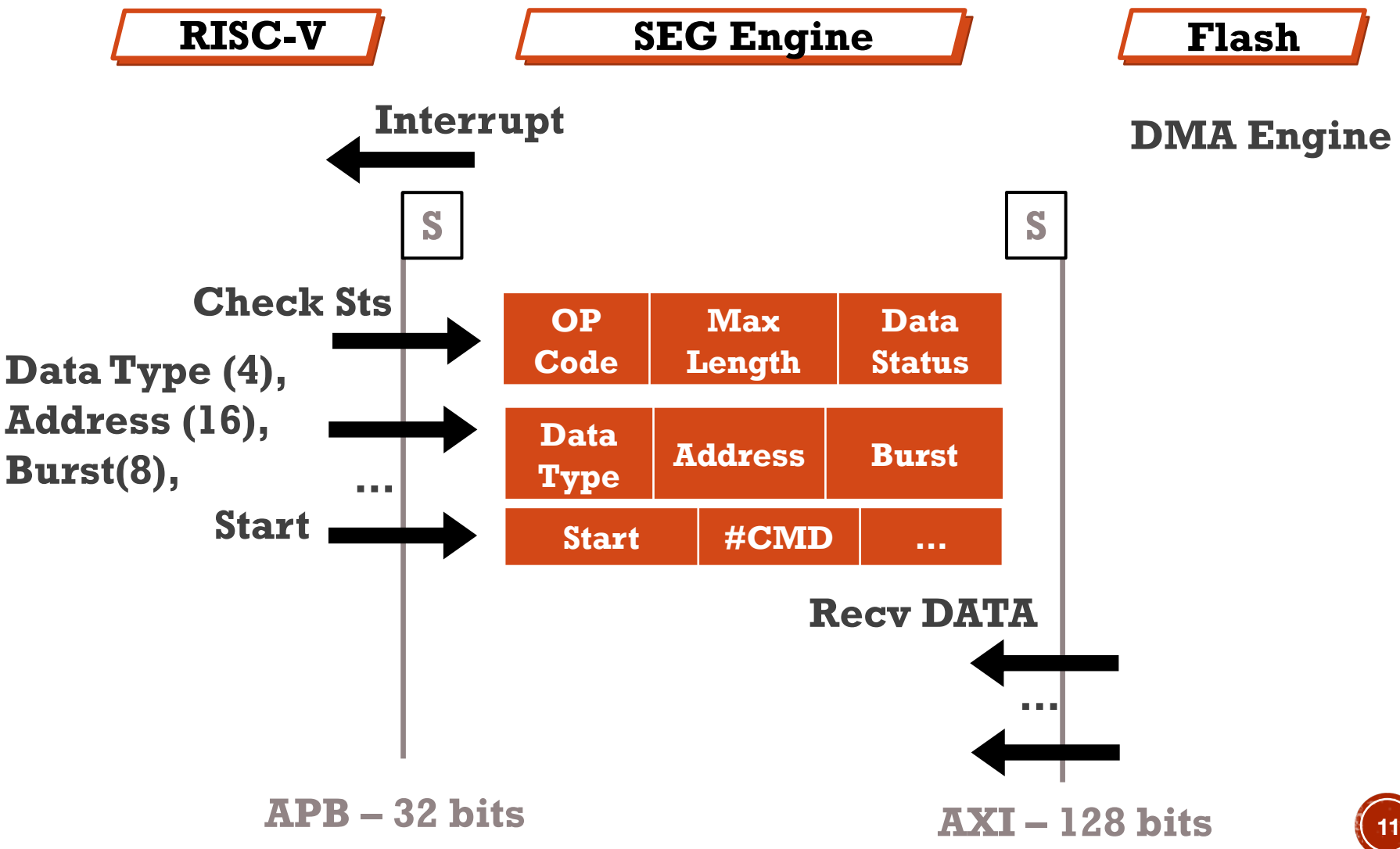
Update Data Status

Is Done ?

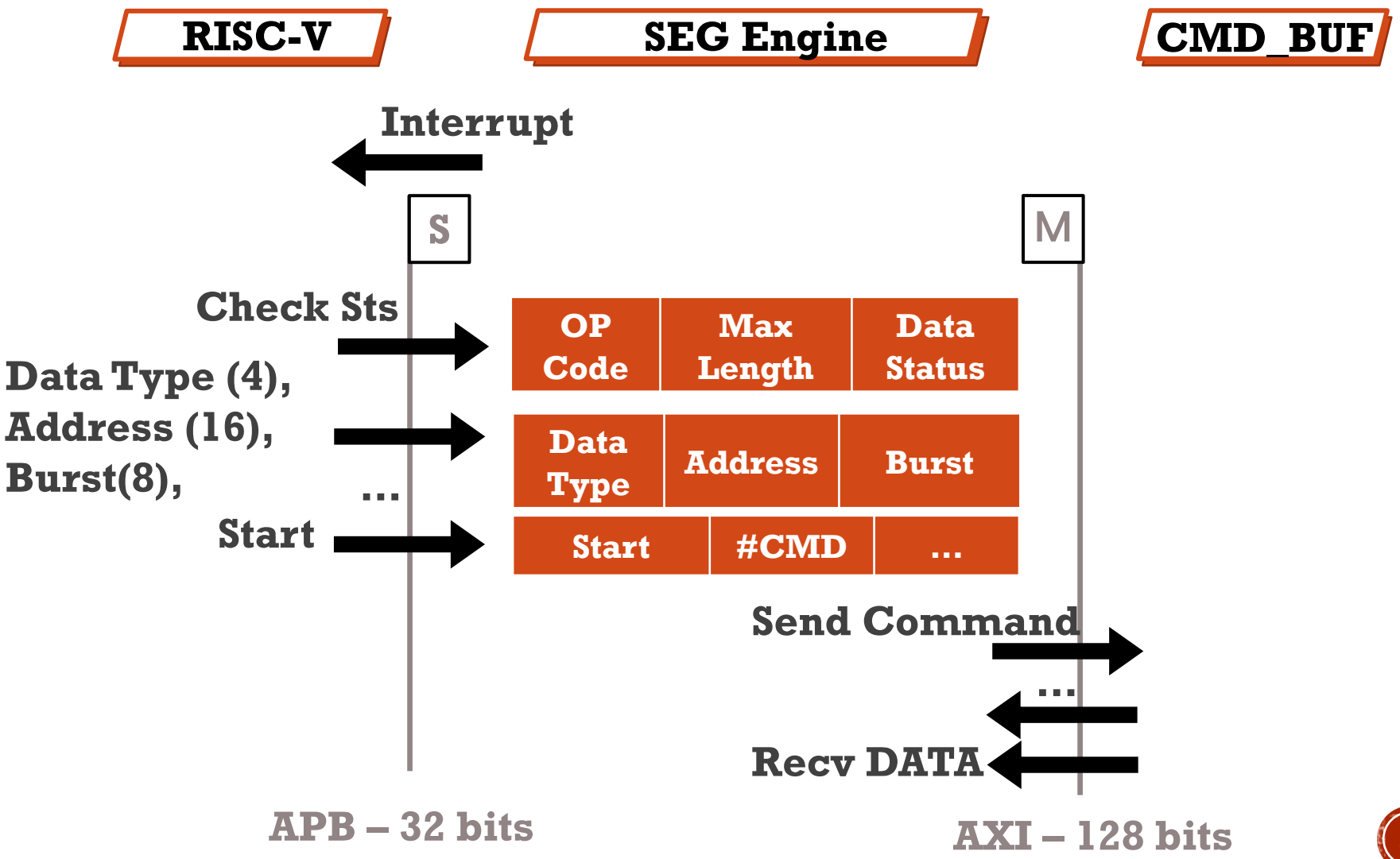
Y

N

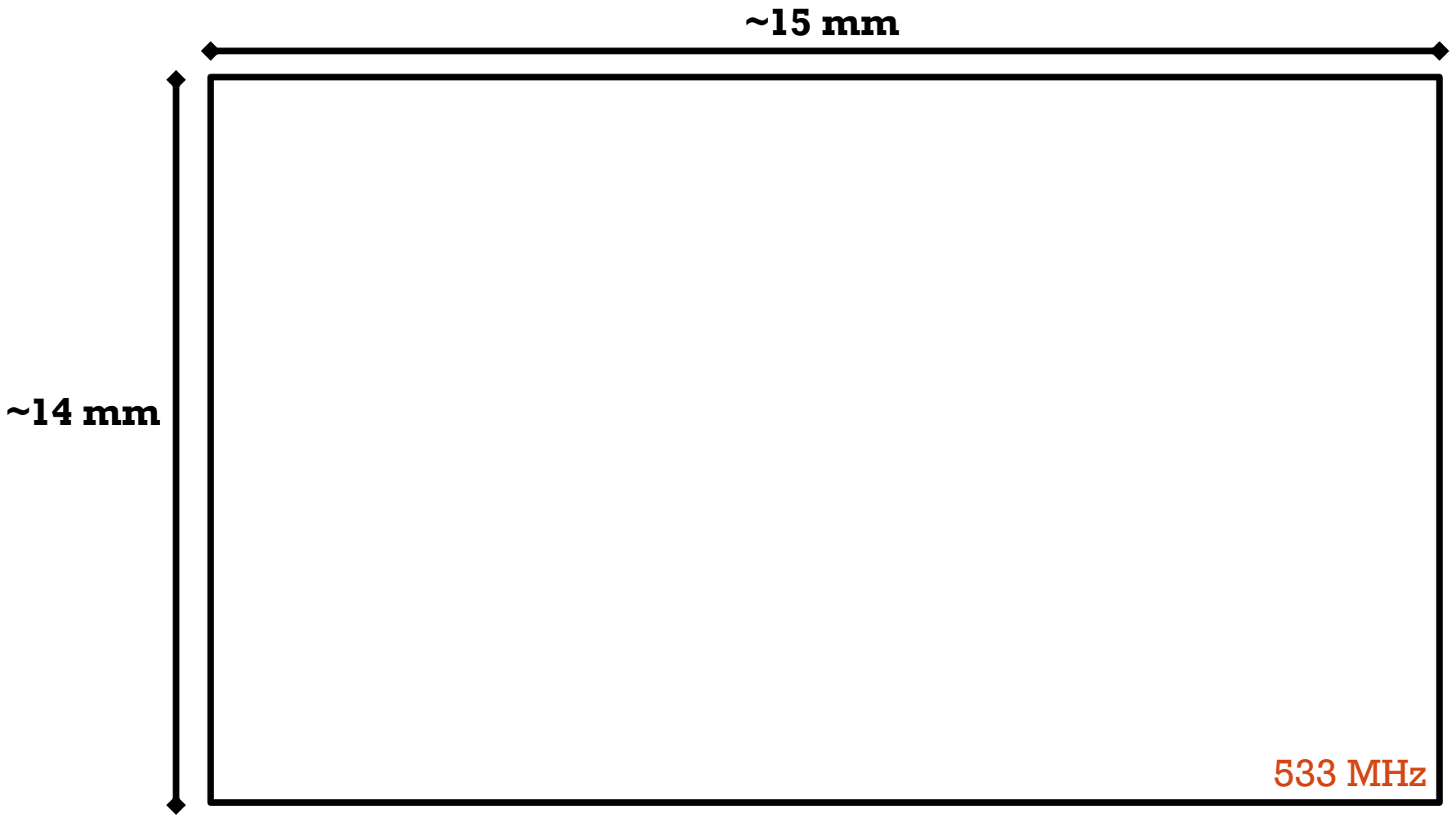
Protocol – Data From Flash

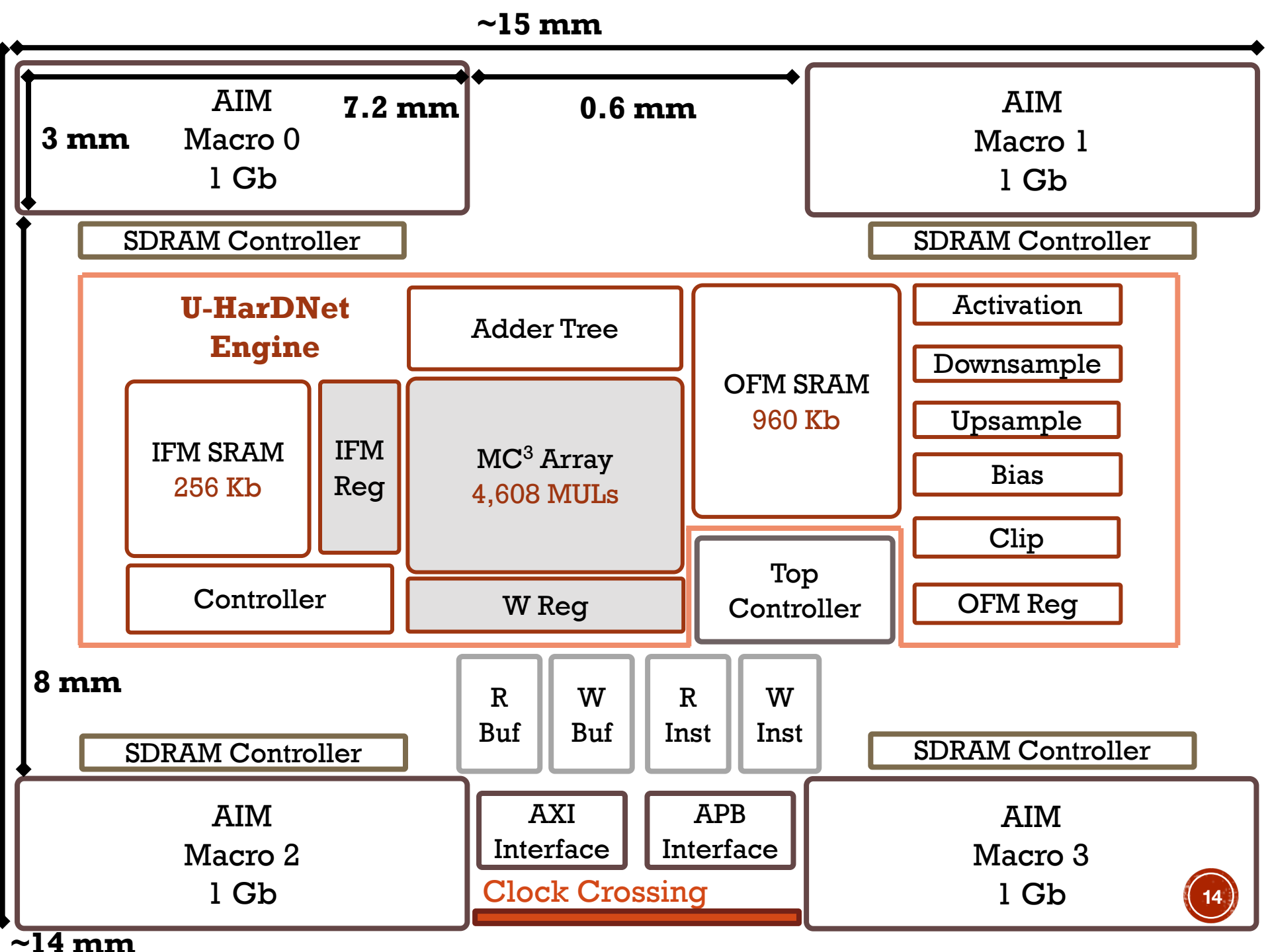


Protocol – Data From CMD-BUF



Segmentation Engine Overview

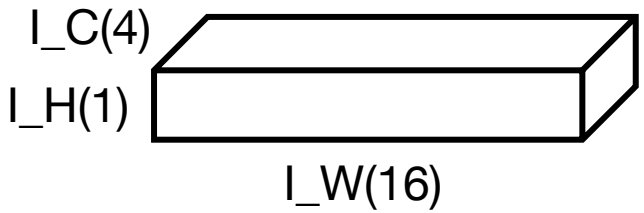



~15 mm
3 mm
AIM
Macro 0
1 Gb
7.2 mm
0.6 mm
AIM
Macro 1
1 Gb
SDRAM Controller
SDRAM Controller
U-HarDNet
Engine
Adder Tree
IFM SRAM
256 Kb
IFM
Reg
MC³ Array
4,608 MULs
OFM SRAM
960 Kb
Activation
Downsample
Upsample
Bias
Clip
OFM Reg
Controller
W Reg
Top
Controller
8 mm
R
Buf
W
Buf
R
Inst
W
Inst
SDRAM Controller
SDRAM Controller
AIM
Macro 2
1 Gb
AXI
Interface
APB
Interface
Clock Crossing
AIM
Macro 3
1 Gb
~14 mm
14

Segmentation Engine – Data Format

Image Sequence

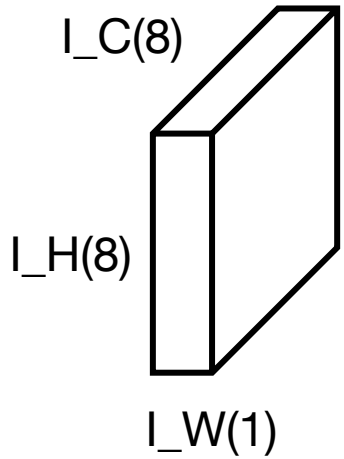
Each data: 8-bit



4 * 16 * 8-bit
= 512 bits

Input Feature Map

Each data: 8-bit



8 * 8 * 8-bit
= 512 bits

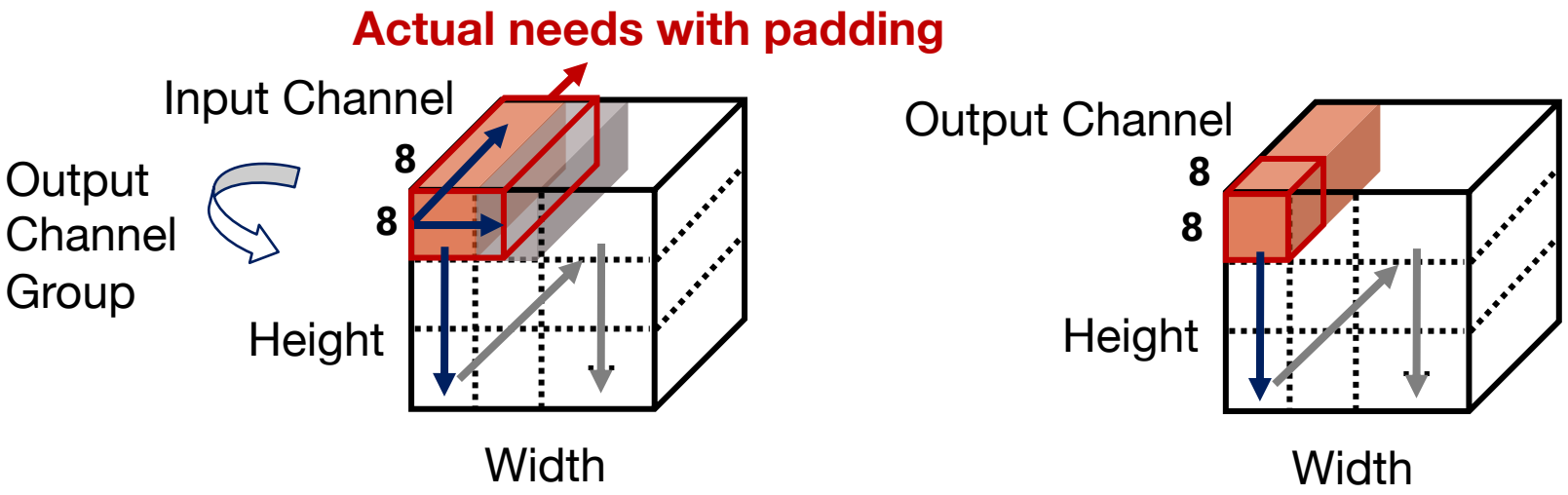
Weight / Bias

Each data: 8-bit

Segmentation Engine – Data Flow

Input Feature Map (IFM)

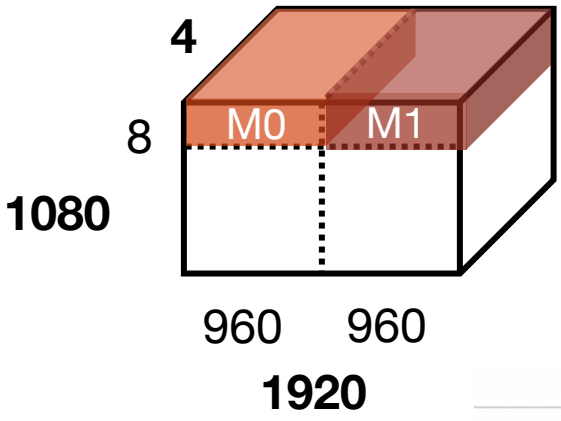
Output Feature Map (OFM)

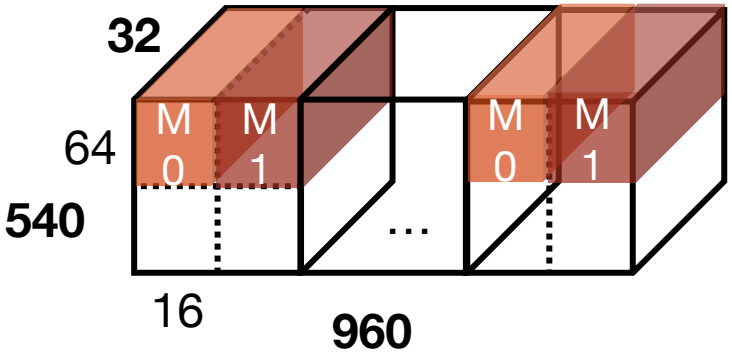


Segmentation Engine Instructions (Appendix: Register Map)

AIM Macro – Data Mapping

Each row
256 Kb





	I_H	I_W			I_C	O_C
Conv1	1024	2048	3	3	3	16
Conv2	512	1024	3	3	16	24
Conv3	512	1024	3	3	24	32
Conv4	256	512	3	3	32	48
Conv5	256	512	3	3	48	10
Conv6	256	512	3	3	58	18
Conv7	256	512	3	3	18	10
Conv8	256	512	3	3	76	28
Conv9	256	512	1	1	48	64

C	H
16	128
32	64
64	32
128	16
256	8

1,920x1,080 (Image Sequence)

960x540

480x270

240x135

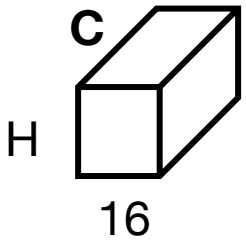
120x67

60x33

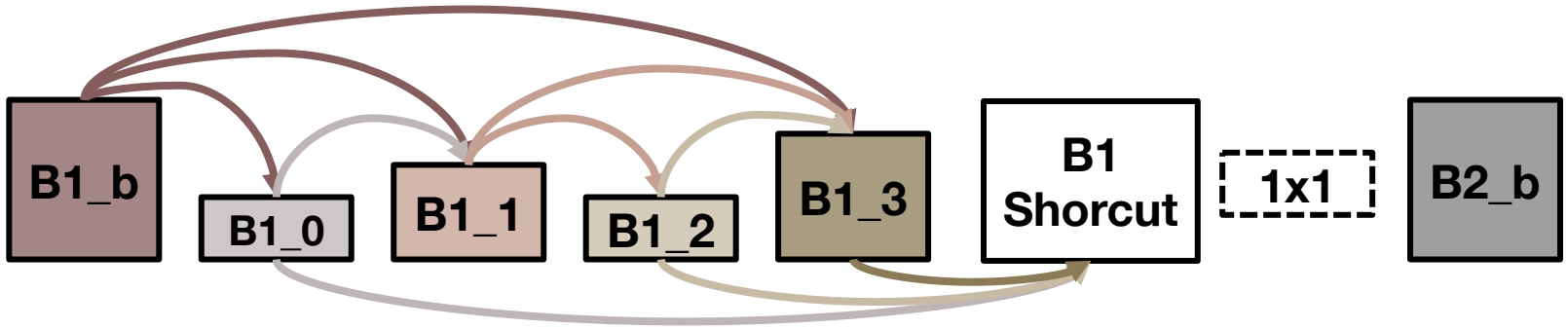
30x16

Div 2

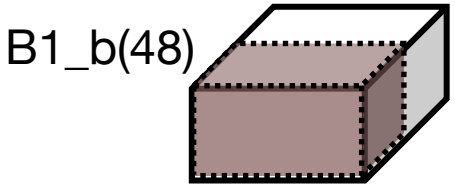
Layer-based Config



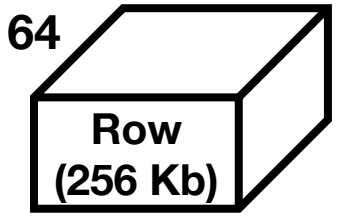
AIM Macro Mapping



FM

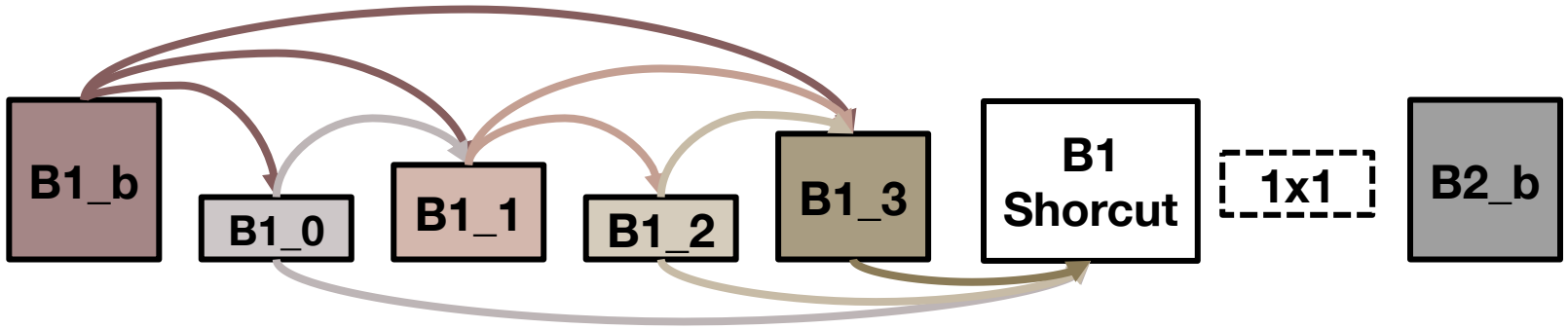


Block_Channel = 64

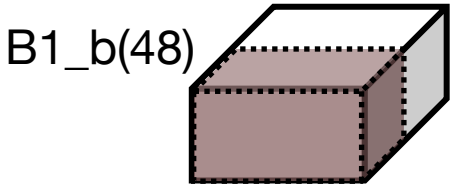


Shortcut

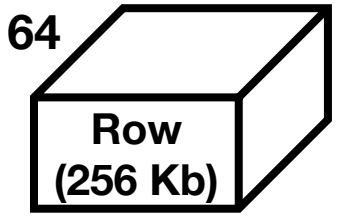
AIM Macro Mapping



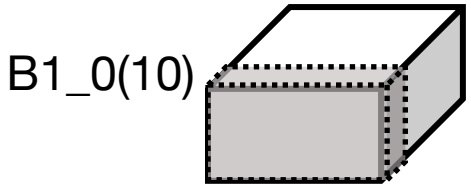
FM



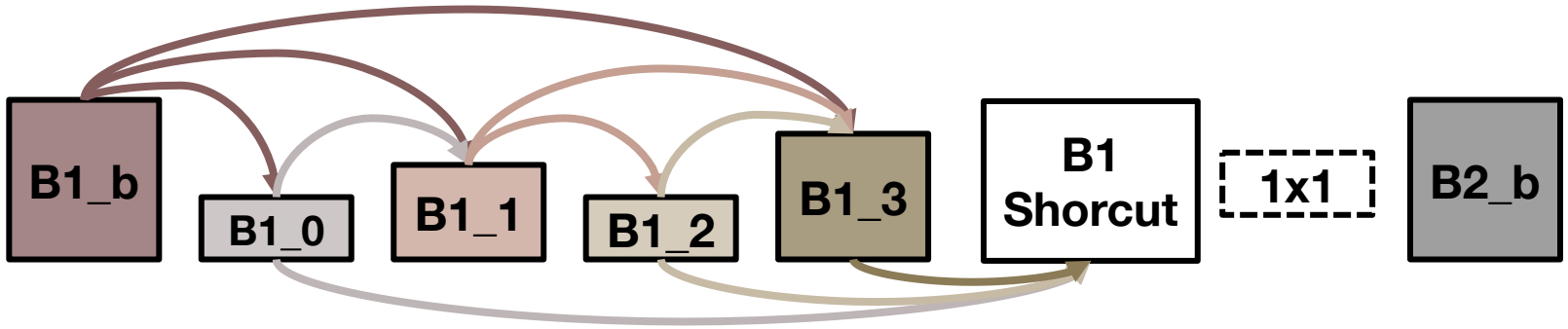
Block_Channel = 64



Shortcut



AIM Macro Mapping

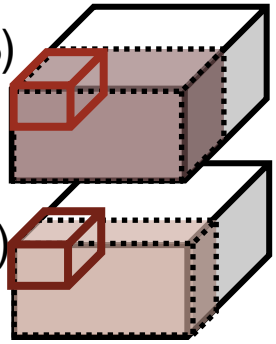


I_C	48	58	18	76	48	48	
O_C	48	10	18	10	28		64

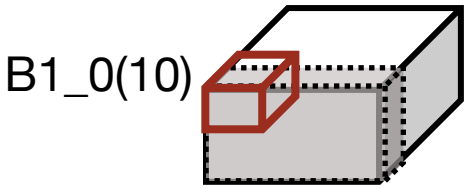
Macro 0/1 **Macro 2/3**

Read B1_b
Read B1_0

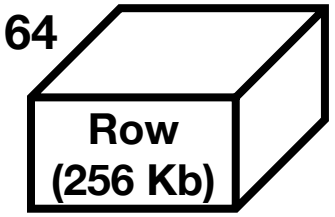
FM



Shortcut



Block_Channel = 64




R_M0/1_B0

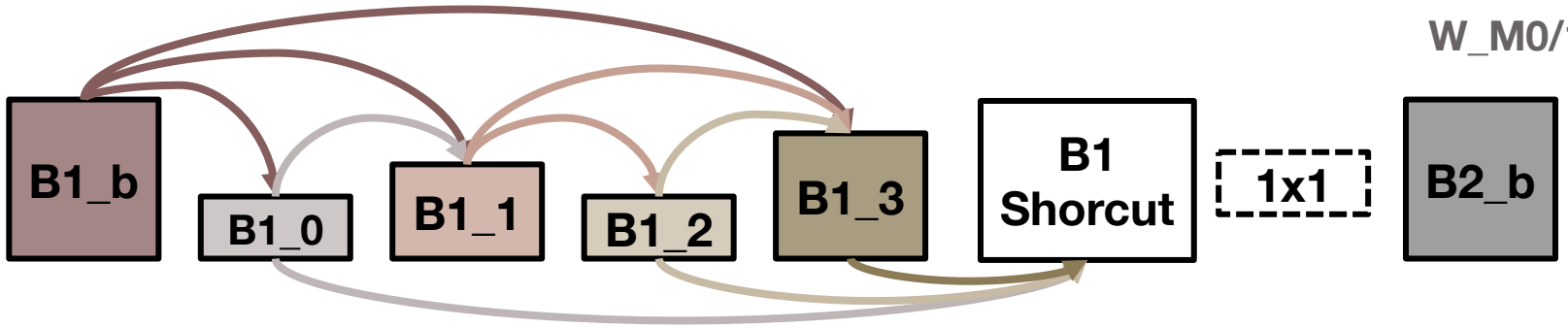

R_M2/3_B0


R_M0/1_B1


R_M2/3_B1

AIM Macro Mapping


W_M0/1_B1

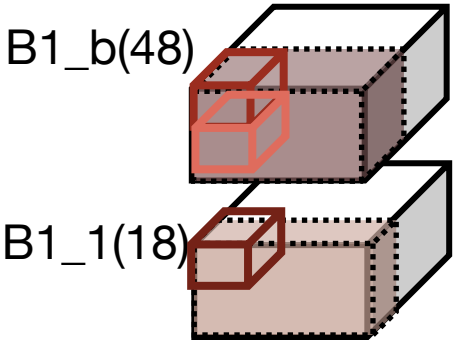


I_C	48	58	18	76	48	48	
O_C	48	10	18	10	28		64

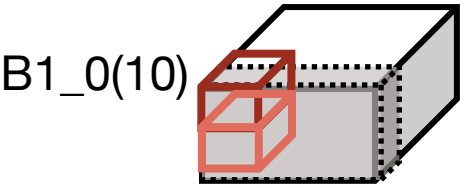
Macro 0/1 **Macro 2/3**


Read B1_b
Read B1_0

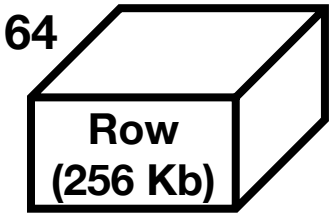
FM



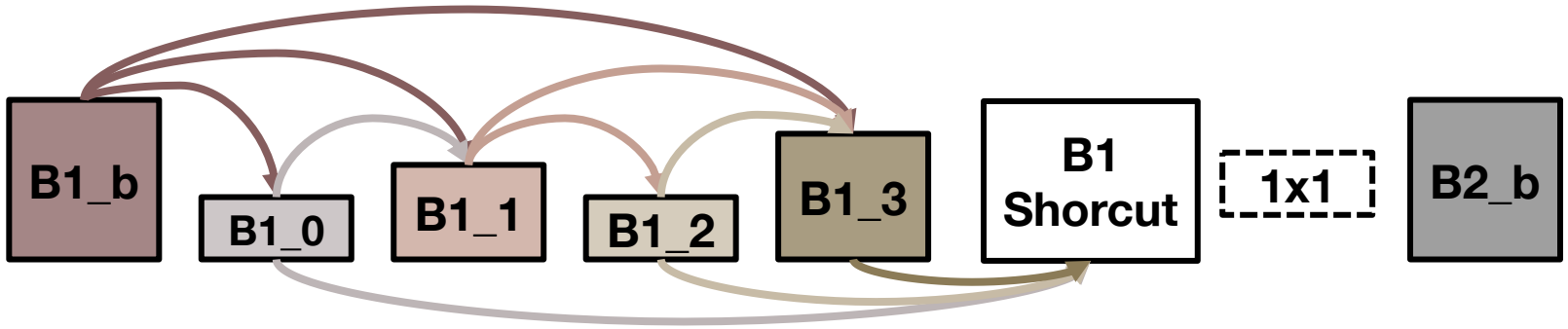
Shortcut

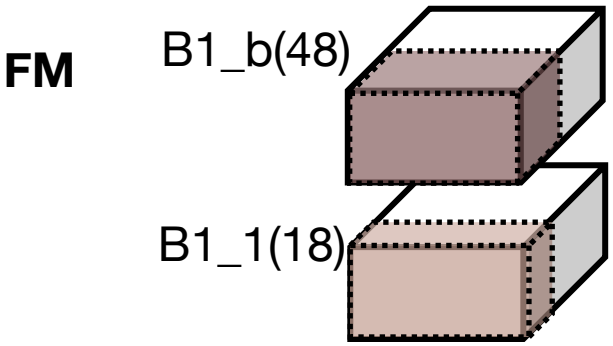


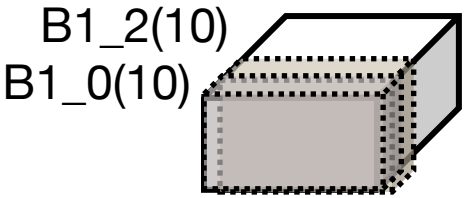
Block_Channel = 64

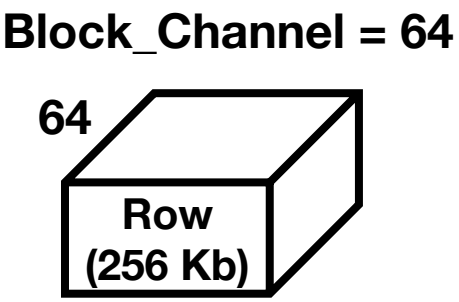


AIM Macro Mapping

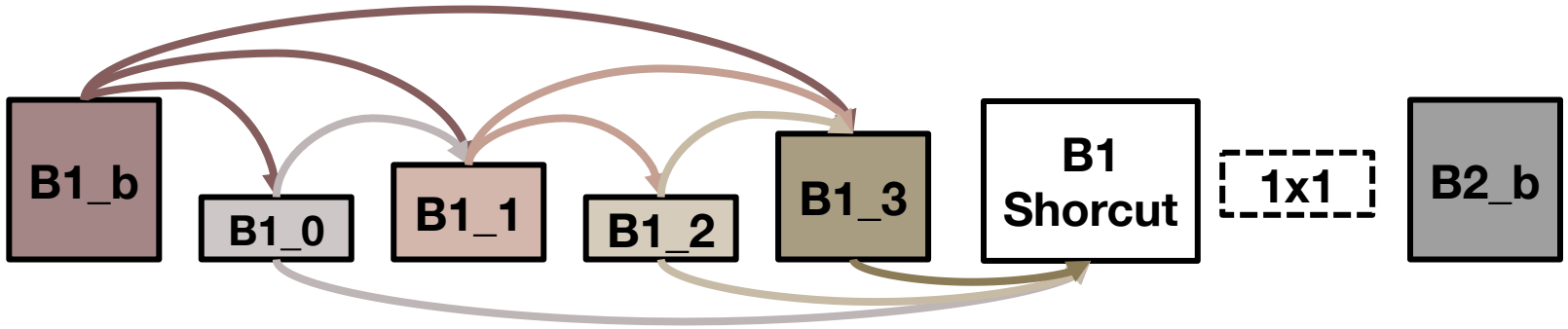








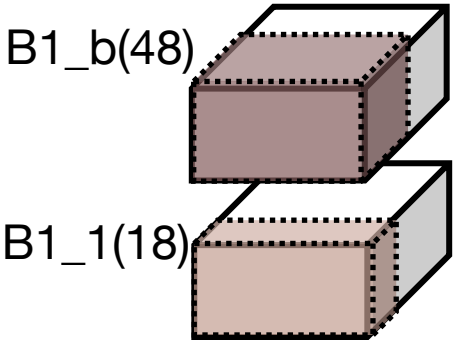
AIM Macro Mapping



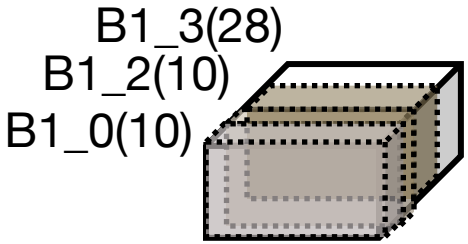
I_C	48	58	18	76	48	48	
O_C	48	10	18	10	28		64

Macro 0/1 **Macro 2/3**

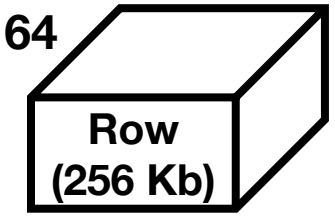
FM



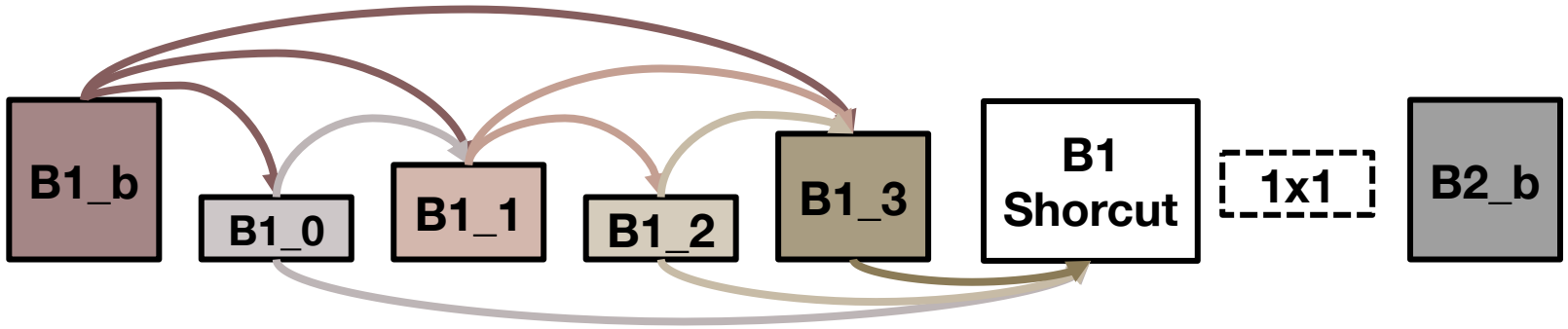
Shortcut



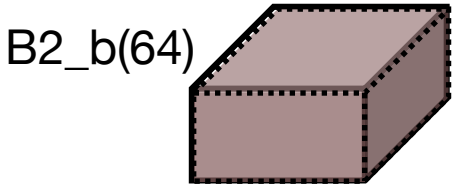
Block_Channel = 64



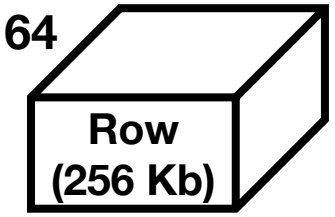
AIM Macro Mapping

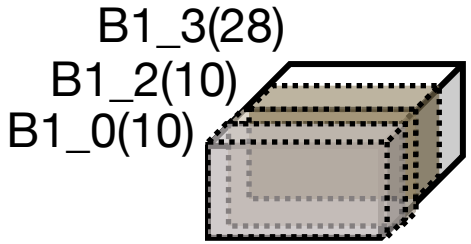


FM



Block_Channel = 64





Shortcut

AIM Macro – Data Allocation

Macro 0, 1, 2, 3	**Bank 0**	**Bank 1**
Current Frame intermediate data	FM 256 Mb	FM 256 Mb
	Skip Connection 128 Mb	Skip Connection 128 Mb
	Weight / Bias 64 Mb	Weight / Bias 64 Mb
1-Frame Buffer	Output Label 8 Mb	Output Label 8 Mb
1-Frame Buffer	Image Sequence 56 Mb	Image Sequence 56 Mb
	512 Mb	**512 Mb**

U-HarDNet Engine Instructions

64-bit/ 128-bit / 256-bit

Opcode (4)	Config (60/124/252)

- 0000: Layer Config - 1
- 0001: Layer Config – 2
- 0010: Operation Setting
- 0011: Data Fetch Configuration
- 0100: Read FM/Read Weight (2R)
- 0101: Read FM/Read Weight/Read Weight (3R)
- 0110: Read FM/Read FM/Read Weight (3R)
- 0111: Read FM/Read FM/Read Weight/Read Weight (4R)
- 1000: Write FM/Write FM/ Write FM/ Write FM (4W)

U-HarDNet Engine Instructions

128-bit

Opcode (4)	Config (124)

Layer Config - 1

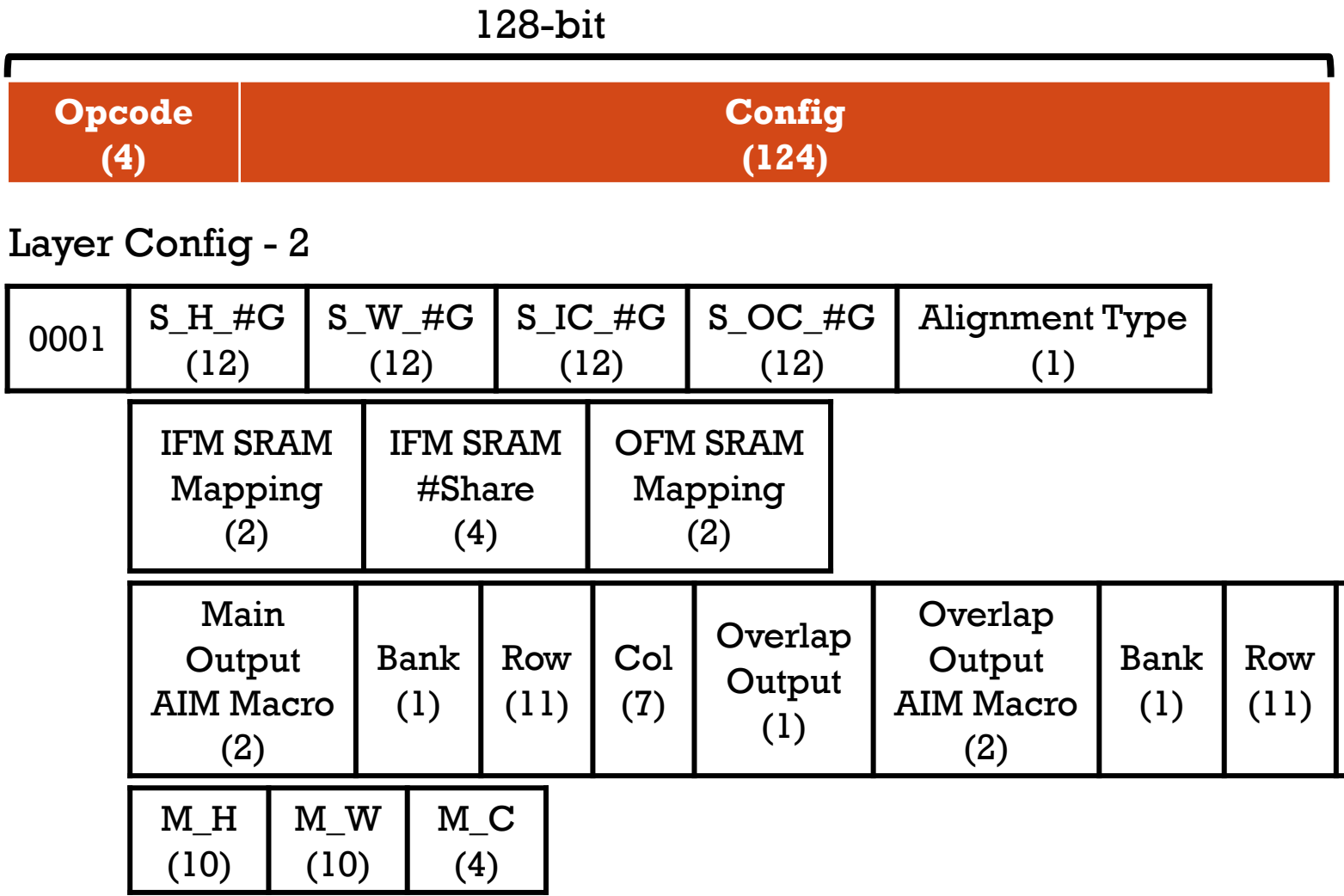
0000	Layer_ID (7)	I_H (12)	I_W (12)	I_C (12)	O_C (12)	S_H (10)	S_W (10)	S_IC (4)	S_OC (4)

R_H (10)	R_W (10)	R_IC (4)	R_OC (4)

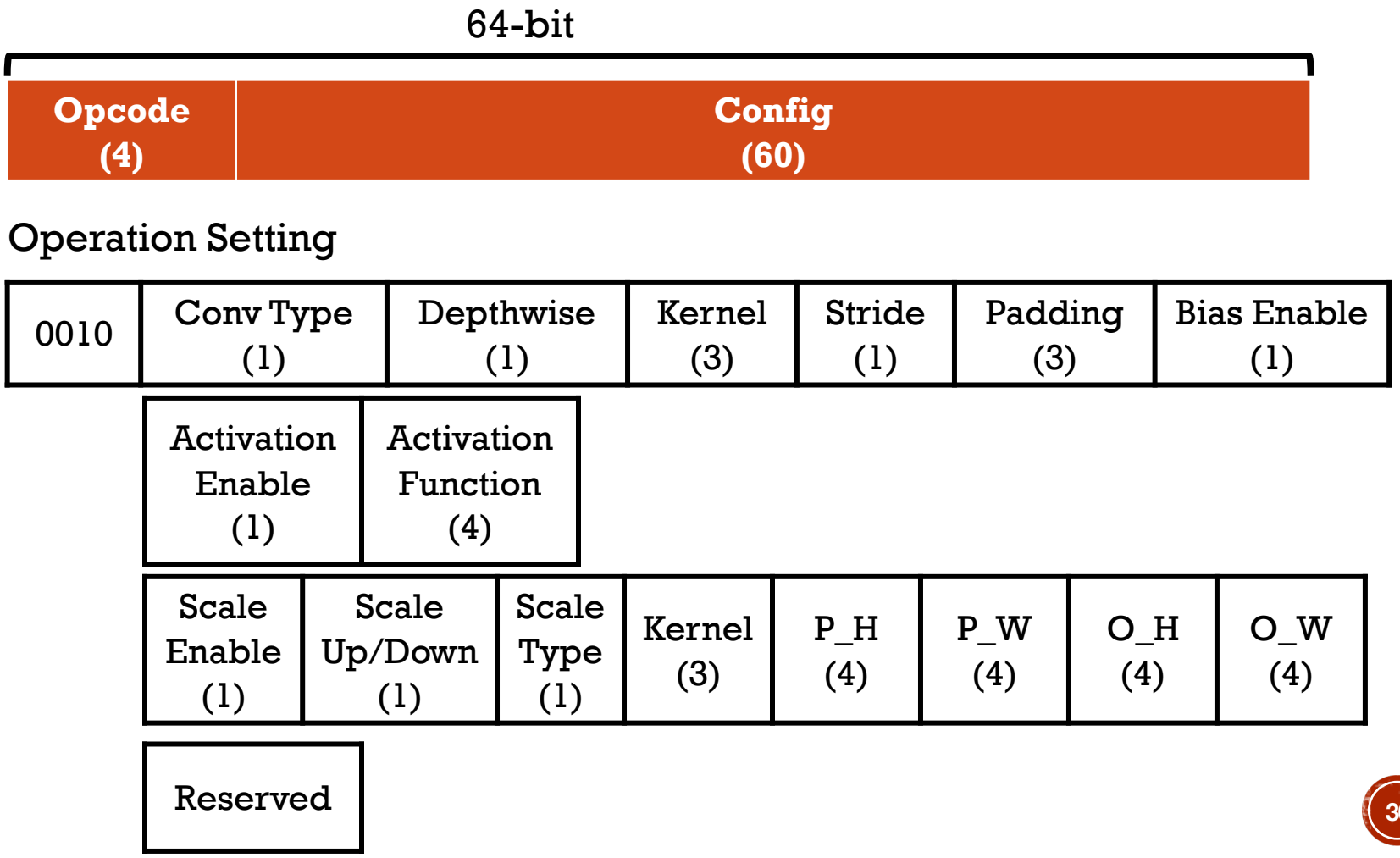
FM_BW_Type (2)	Param_BW_Type (2)

FM_QFormat (3)	Param_QFormat (3)	Bias_QFormat (3)

U-HarDNet Engine Instructions



U-HarDNet Engine Instructions



U-HarDNet Engine Instructions

128-bit

Opcode (4)	Config (124)

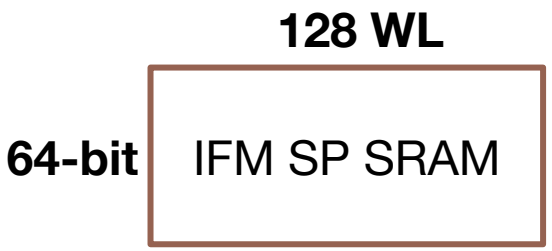
Read FM/Read Weight (2R)

0100	Layer_ID (7)	Block_ID (4)	Inst_ID ()						

FM AIM Macro (2)	Bank (1)	Row (11)	Col (7)	Burst (7)	Action (4)	Remain (1)	Slice Enable (2)

Weight AIM Macro (2)	Bank (1)	Row (11)	Col (7)	Burst (7)	Action (4)	Remain (2)	Slice Enable (2)

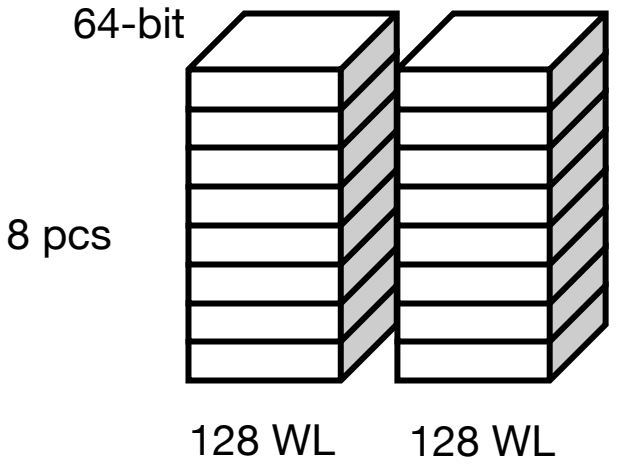
IFM SP SRAM



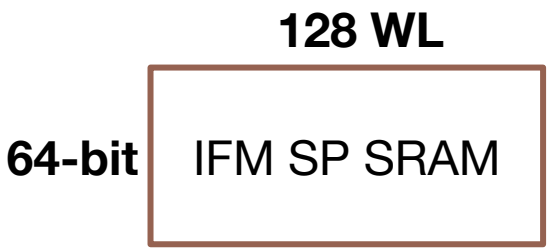
2 * 8 pcs * (64-bit * 128 WL)
= 128 Kb

Each data: 8 bits
Input data: 512 bits

8 pcs * 64-bit = 512 bits



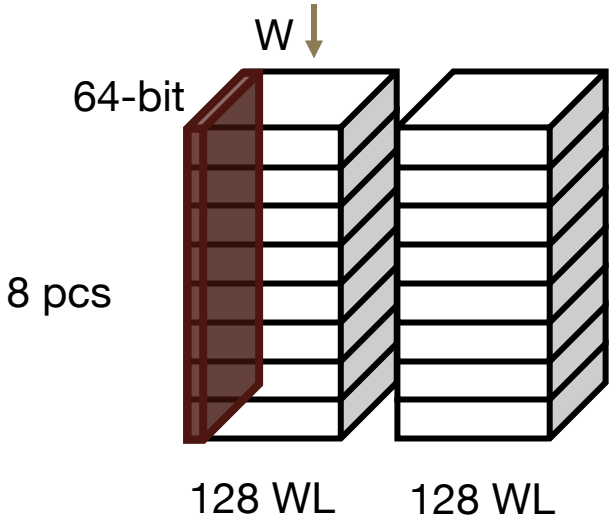
IFM SP SRAM



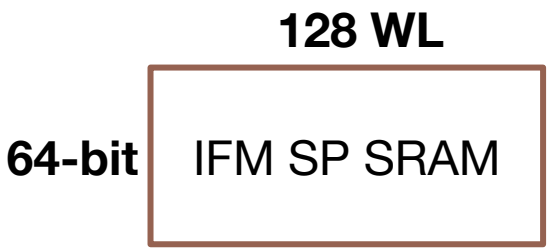
2 * 8 pcs * (64-bit * 128 WL)
= 128 Kb

Each data: 8 bits
Input data: 512 bits

8 pcs * 64-bit = 512 bits



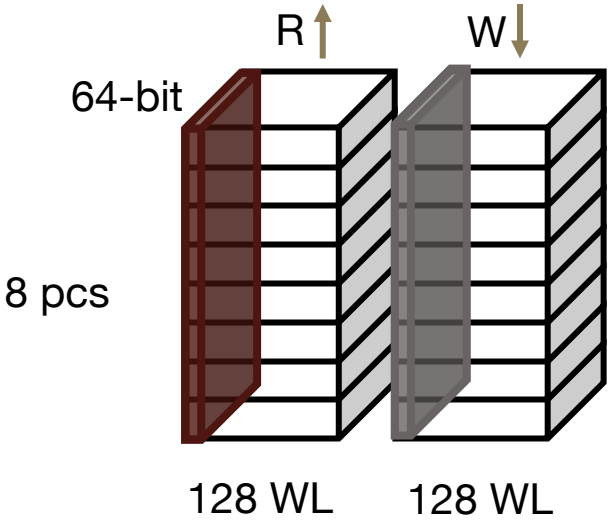
IFM SP SRAM



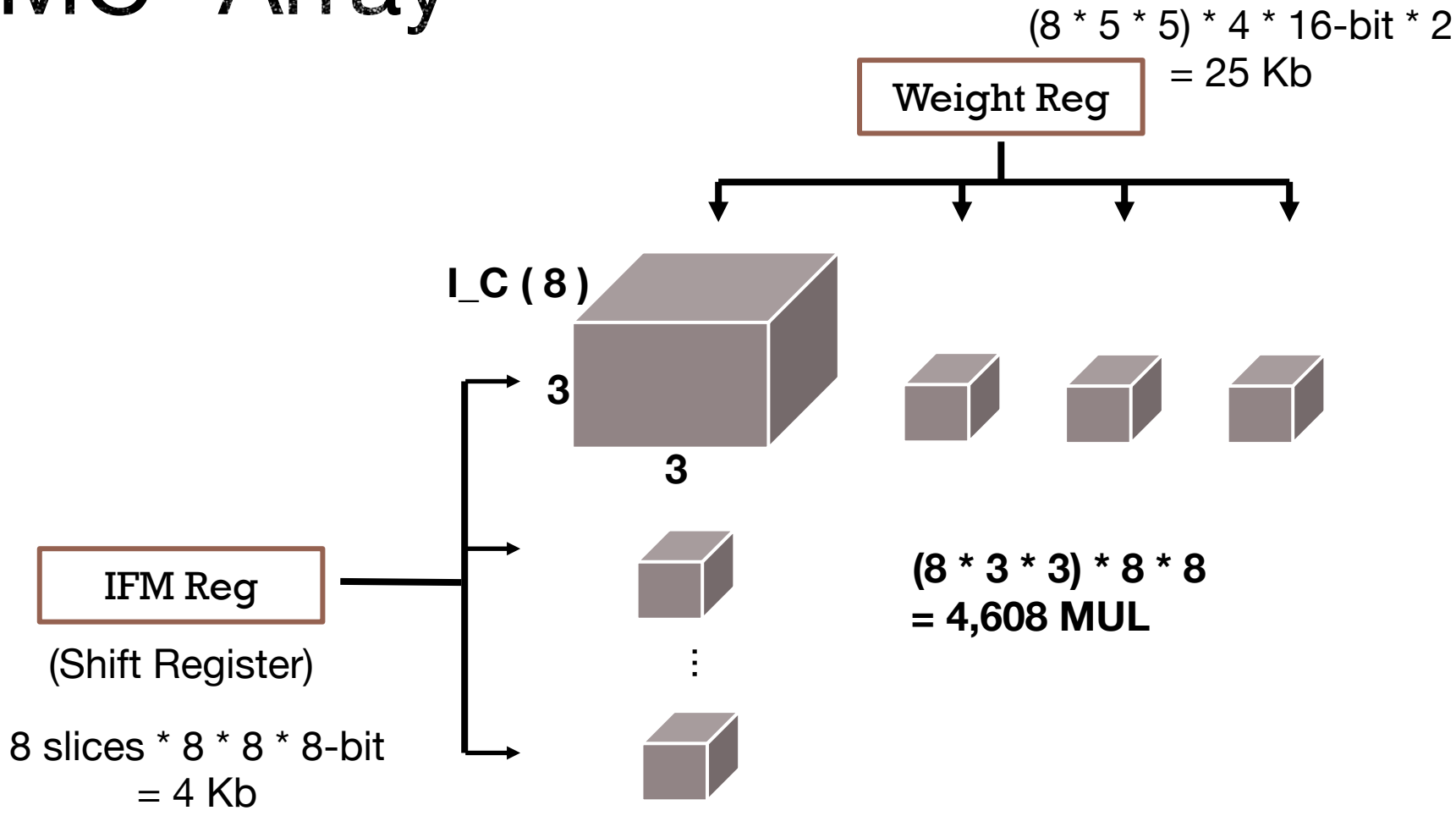
2 * 8 pcs * (64-bit * 128 WL)
= 128 Kb

Each data: 8 bits
Input data: 512 bits

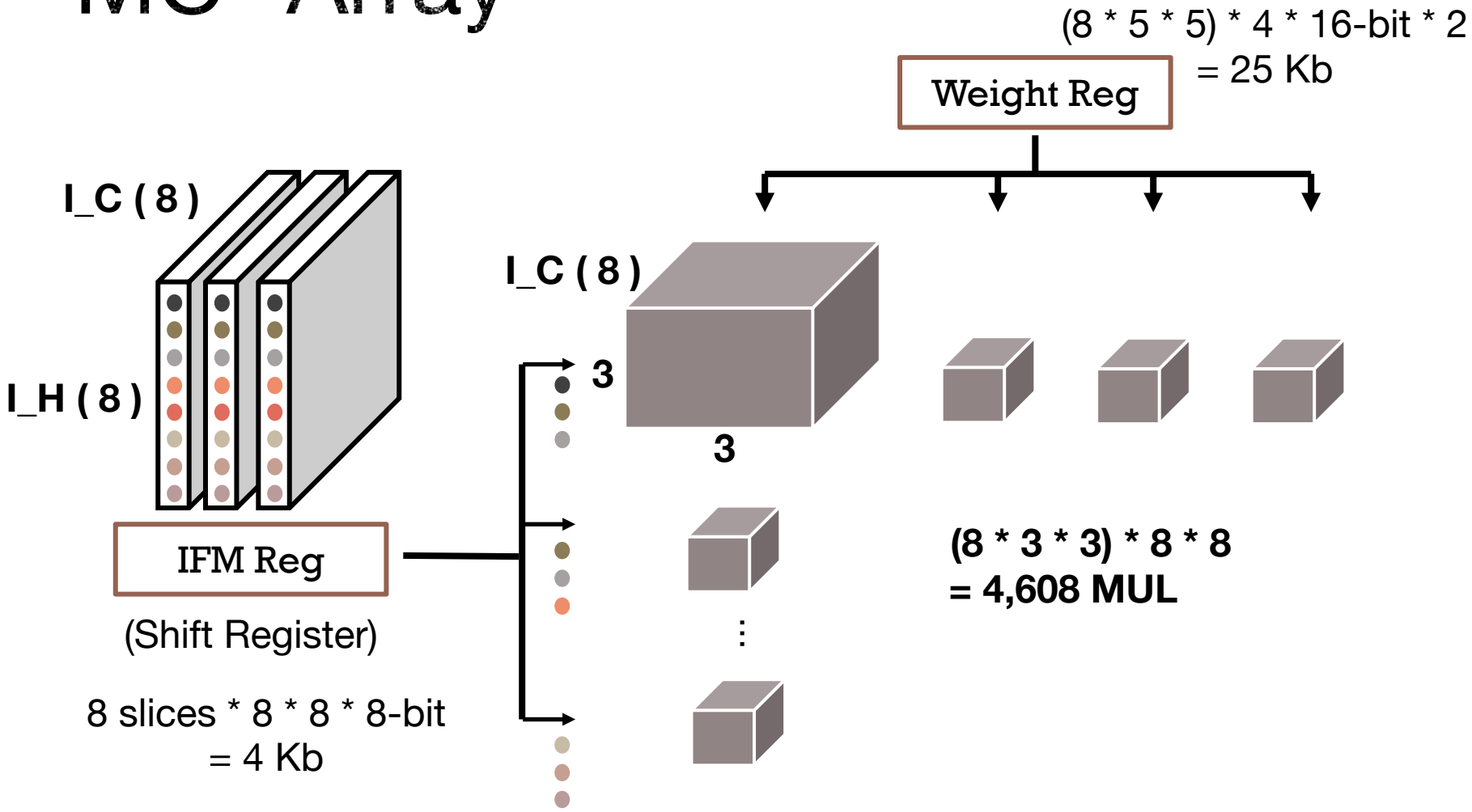
8 pcs * 64-bit = 512 bits



MC^3 Array



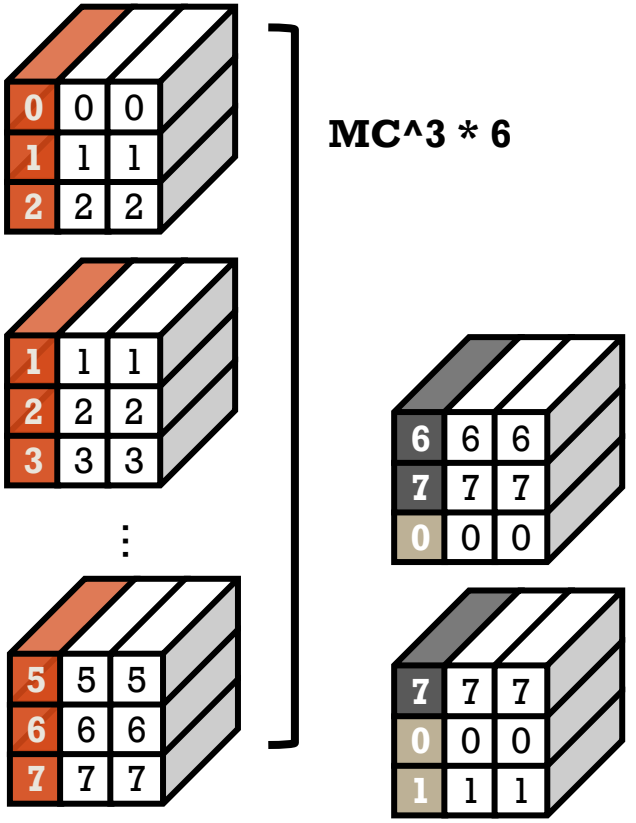
MC³ Array



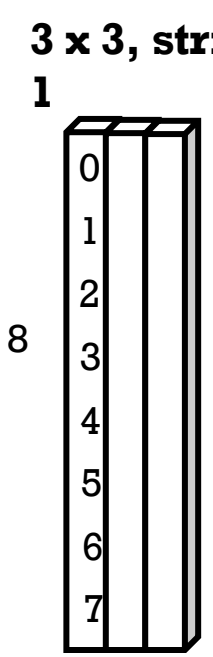
MC^3 Array – 3 x 3, stride 1

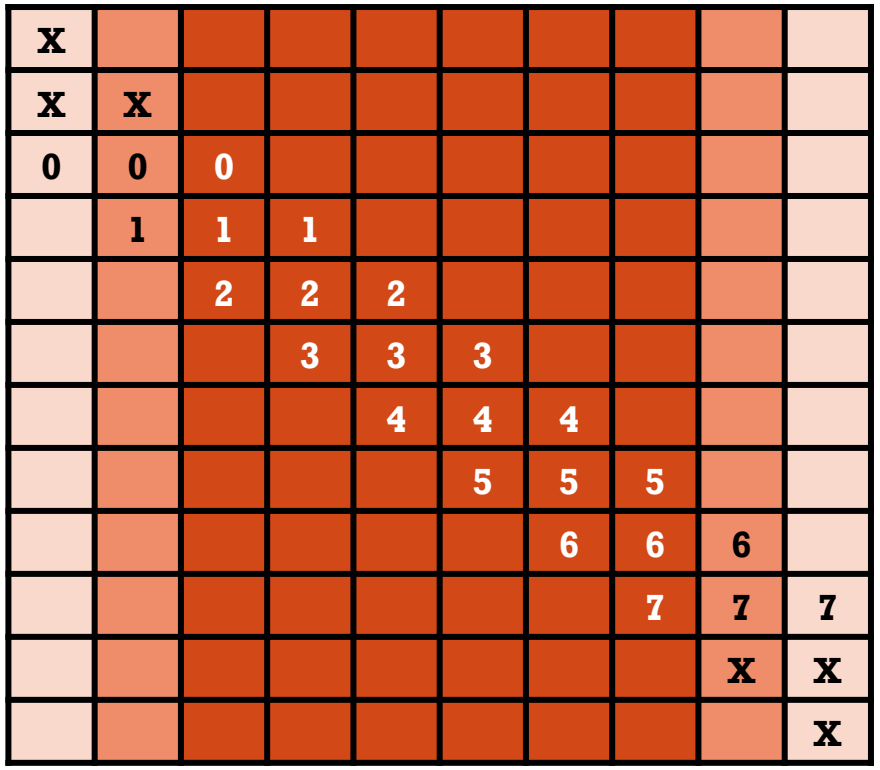
I_H = 8 (0 ~ 7)

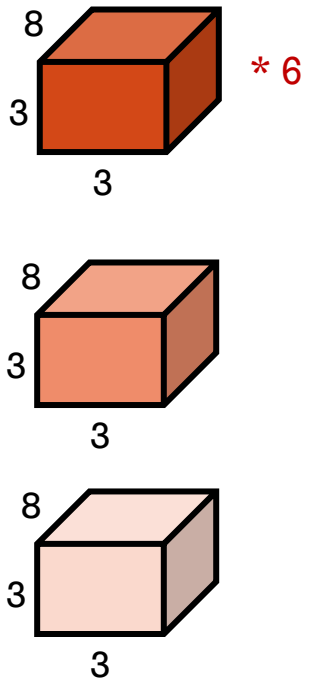
Psum = 6 + 4



Architecture – MC^3 Array



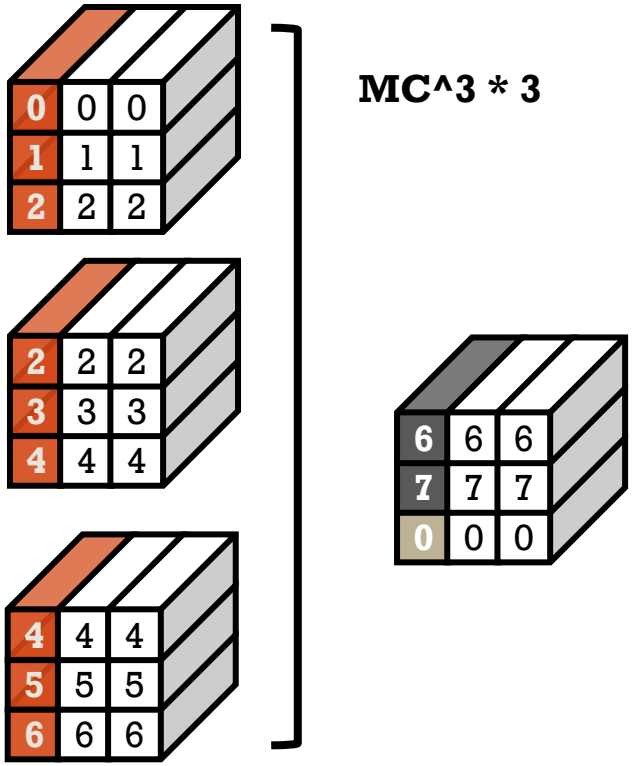




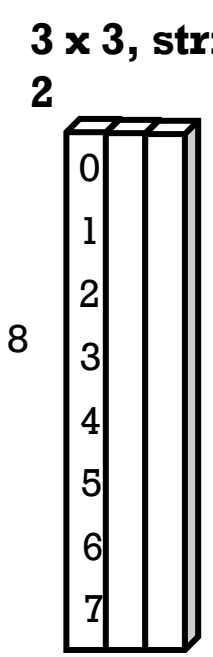
MC^3 Array – 3 x 3, stride 2

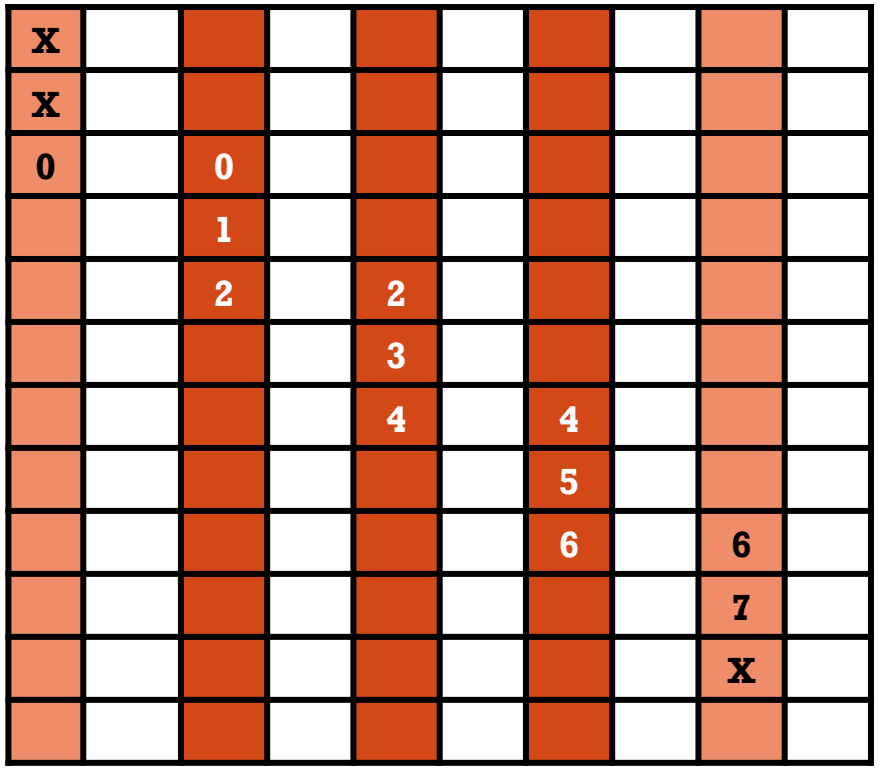
I_H = 8 (0 ~ 7)

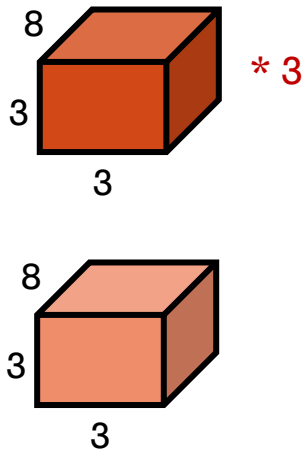
Psum = 3 + 2



Architecture – MC^3 Array



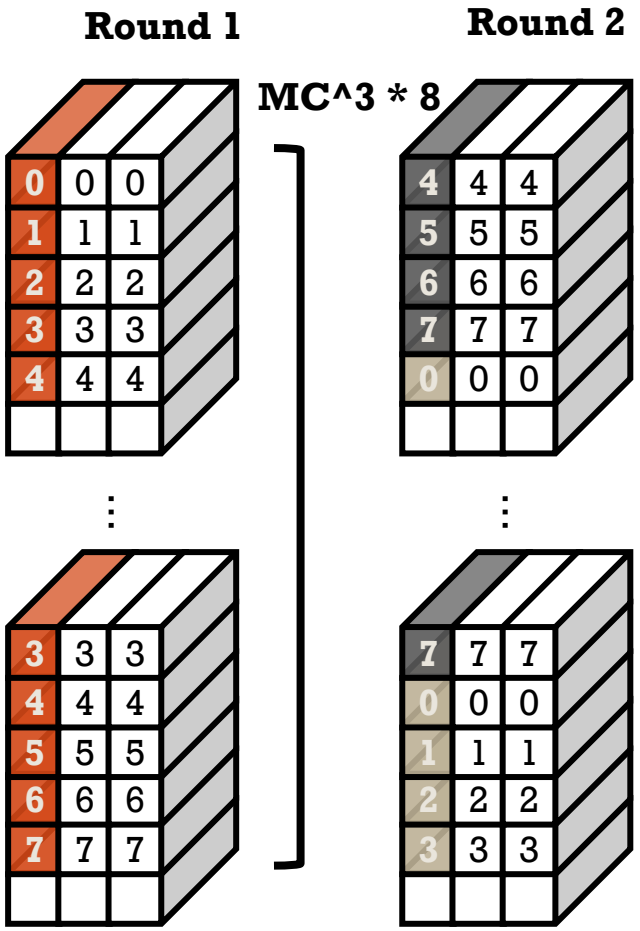




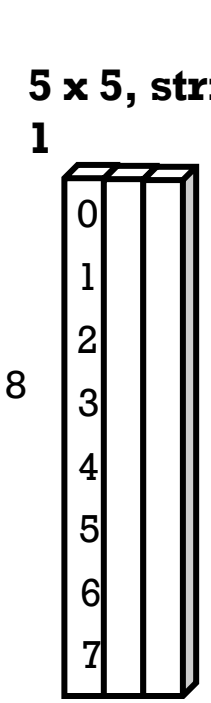
MC^3 Array – 5 x 5, stride 1

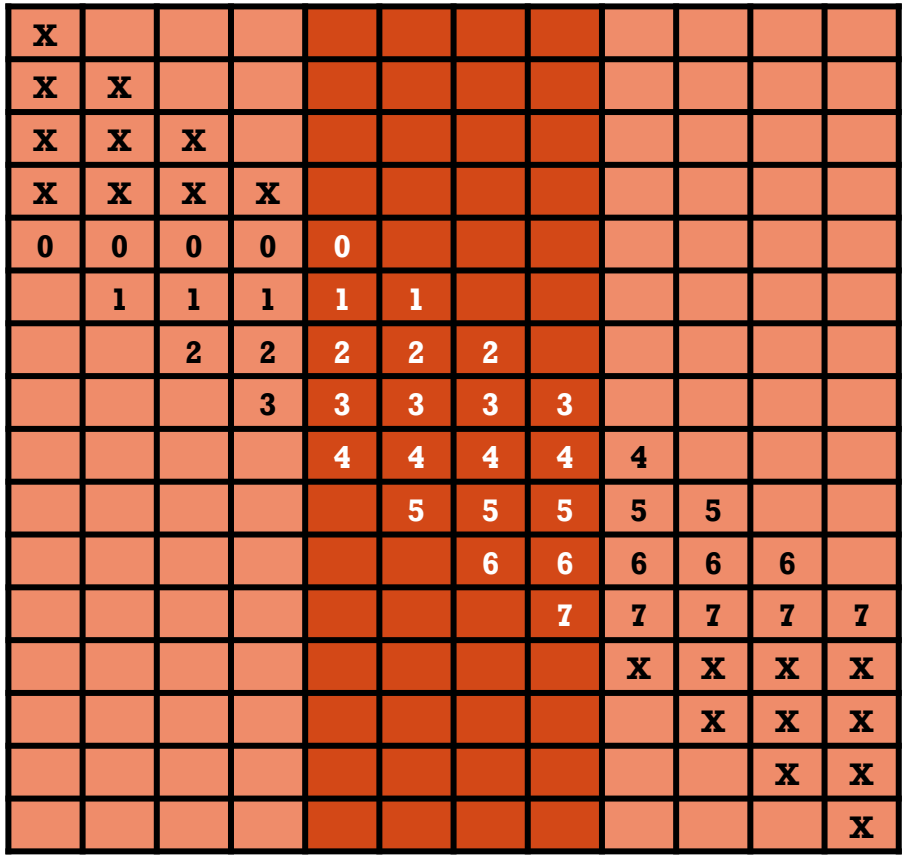
I_H = 8 (0 ~ 7)

Psum = 4 or 8

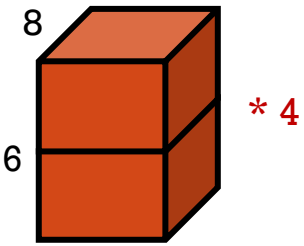


Architecture – MC^3 Array

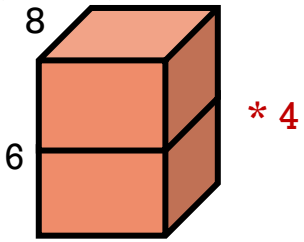




Round 1 (O_C: 8)



Round 2 (O_C:8)

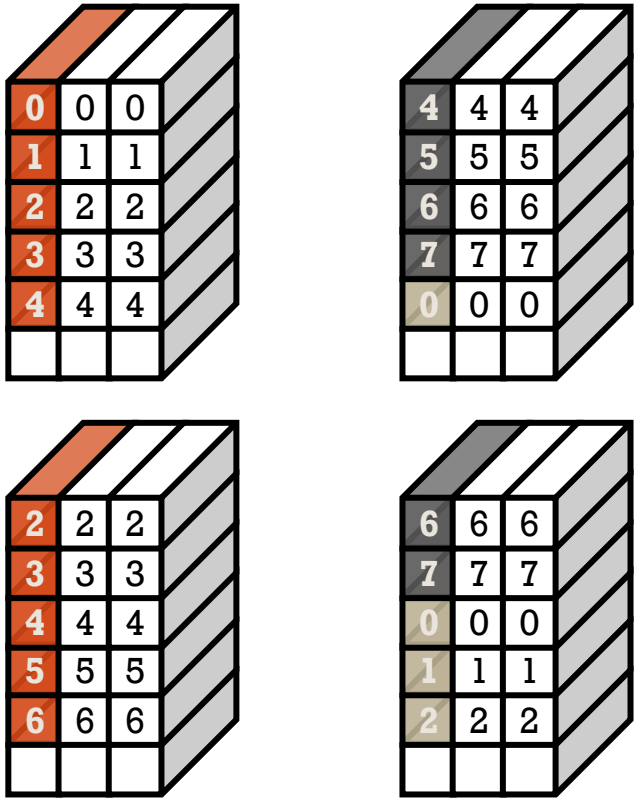


MC^3 Array – 5 x 5, stride 2

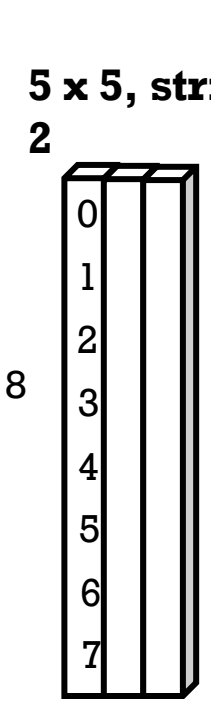
I_H = 8 (0 ~ 7)

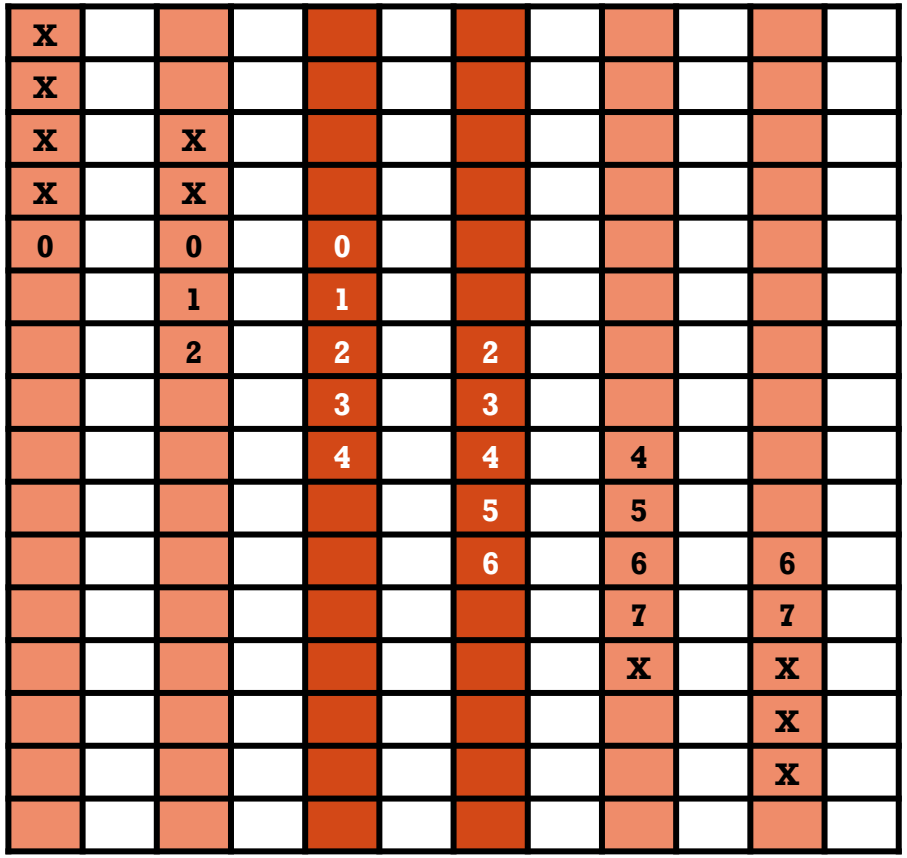
Psum = 2 + 4

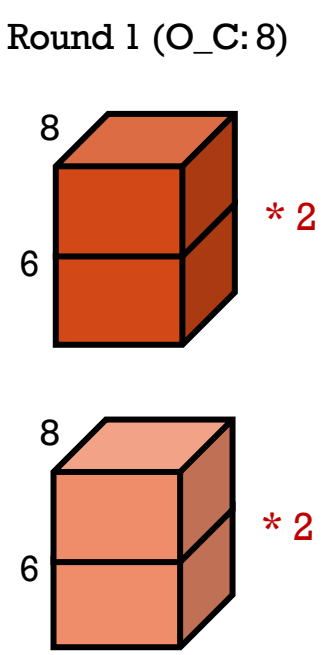
MC^3 * 8



Architecture – MC^3 Array







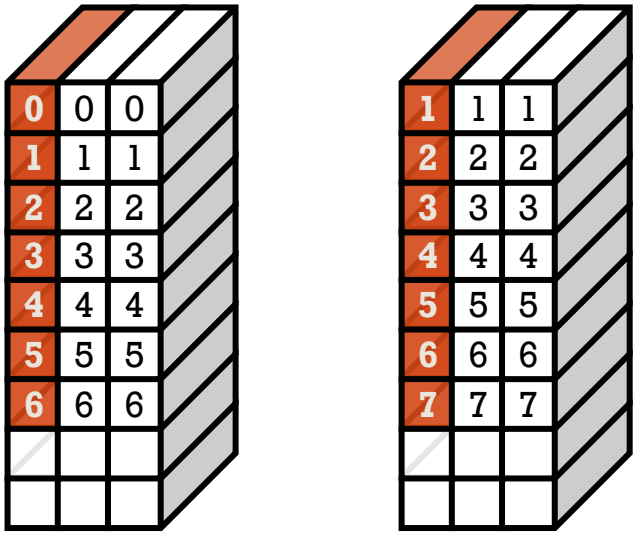
MC^3 Array – 7 x 7, stride 1

I_H = 8 (0 ~ 7)

Psum = 2 or 4

Round 1

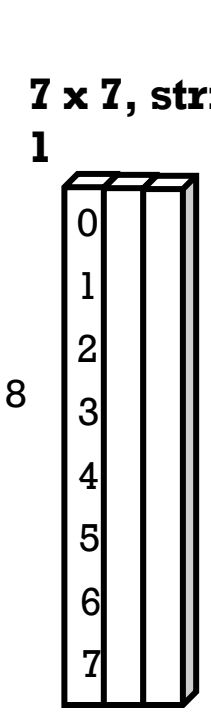
MC^3 * 6

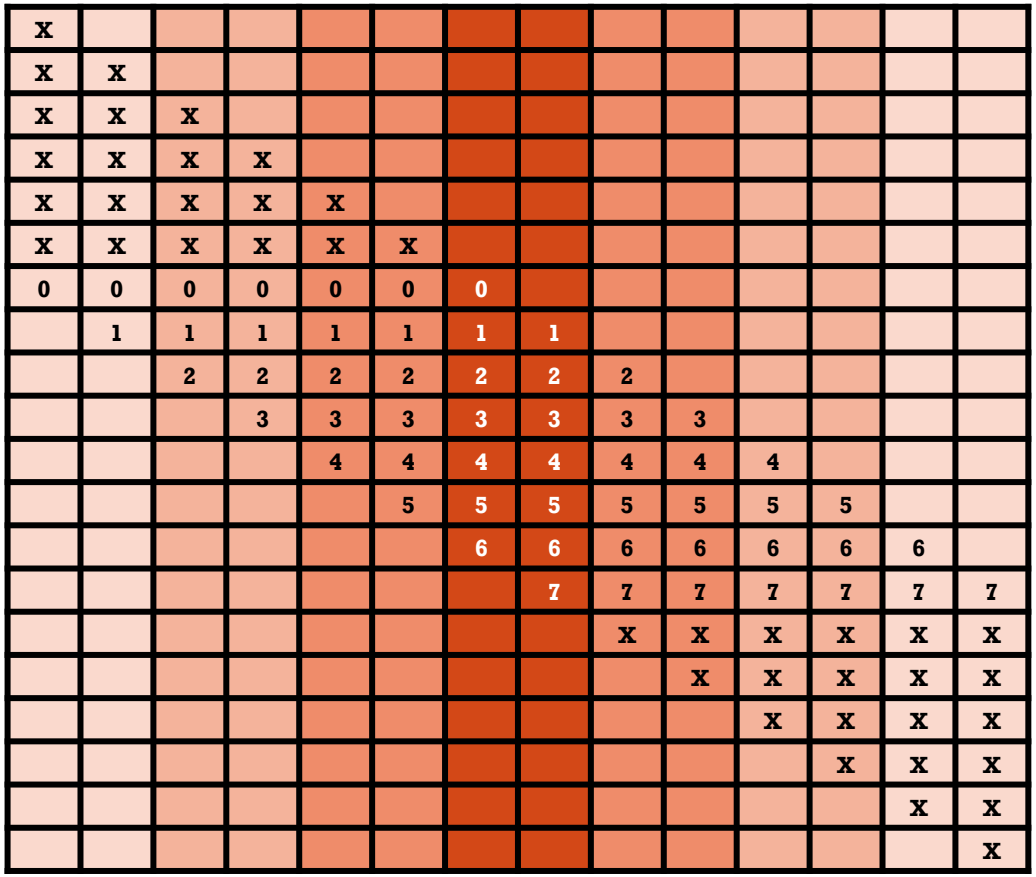


Round 2~4

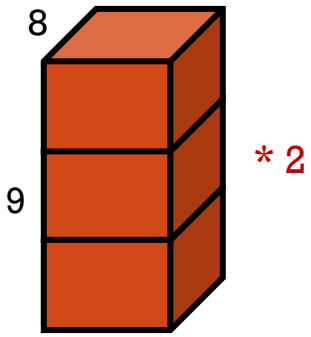
MC^3 * 6

Architecture – MC^3 Array





Round 1 (O_C: 4)



Round 2 (O_C: 4)

Round 3 (O_C: 4)

Round 4 (O_C: 4)

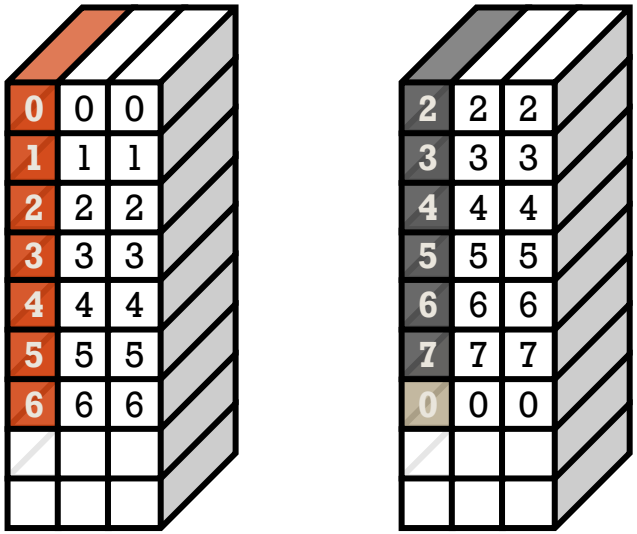
MC^3 Array – 7 x 7, stride 2

I_H = 8 (0 ~ 7)

Psum = 3 or 4

Round 1

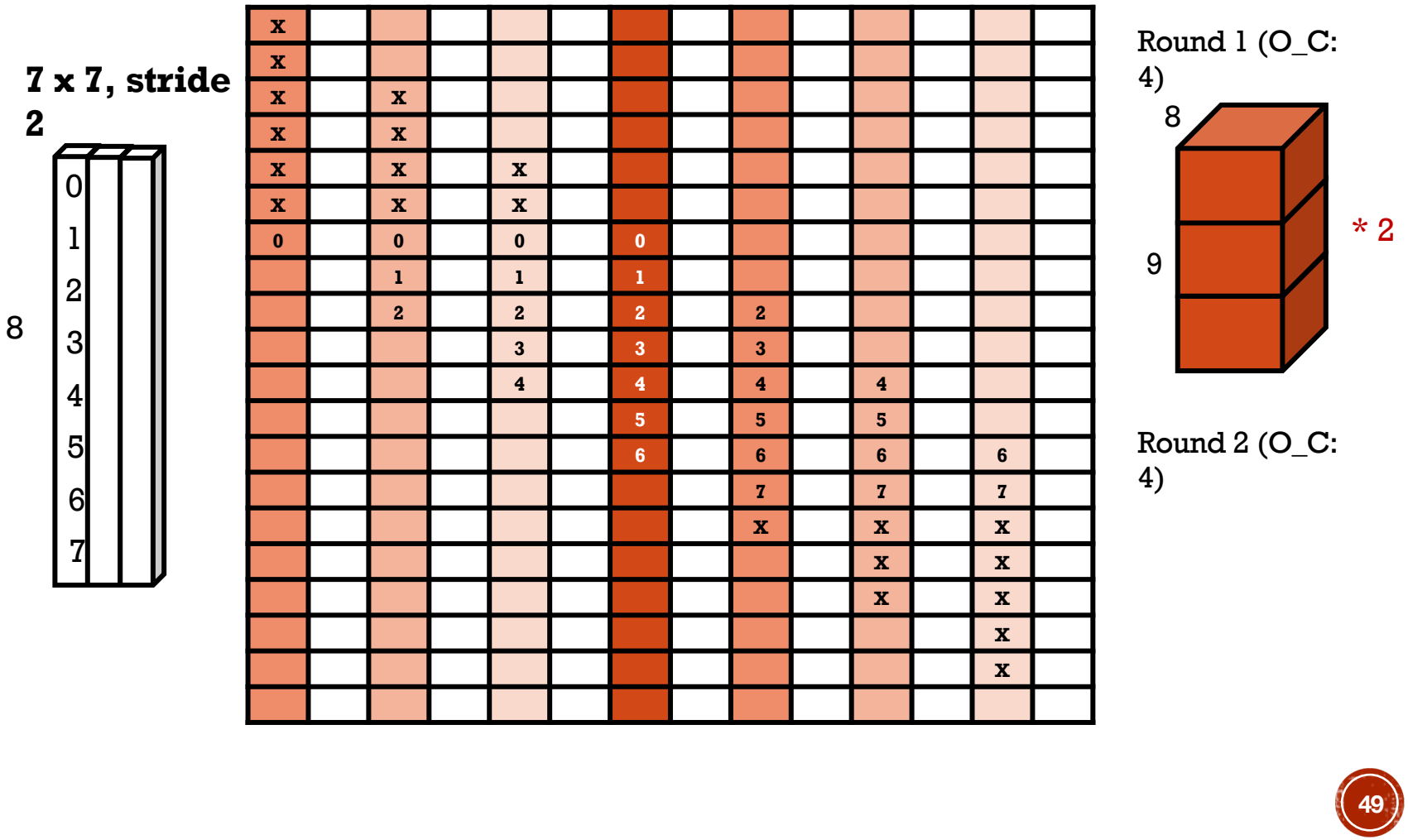
MC^3 * 6



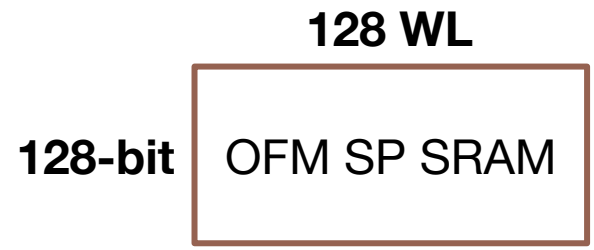
Round 2

MC^3 * 6

Architecture – MC^3 Array

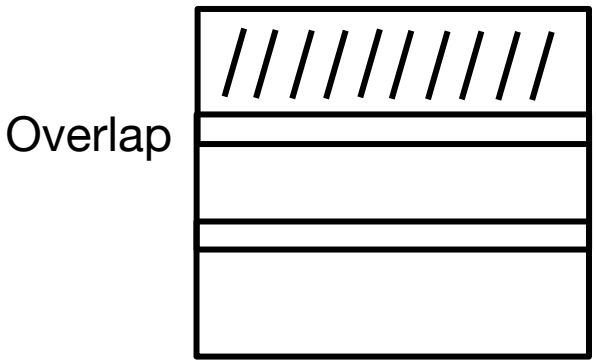


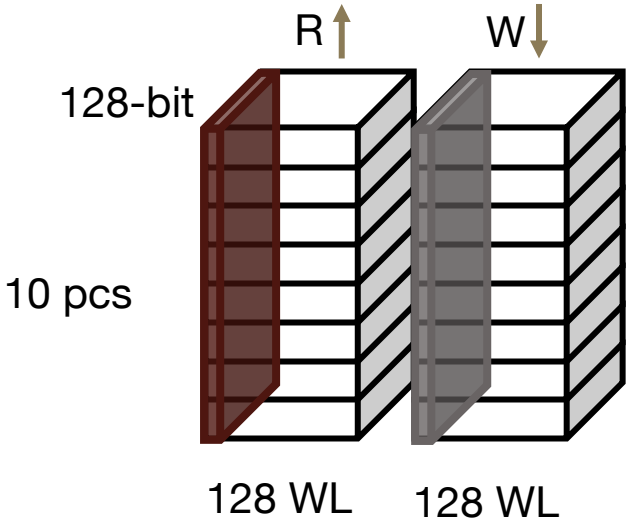
OFM SP SRAM



Each data: 32 bits

3 * 20 pcs * (128-bit * 128 entry)
= 960 Kb

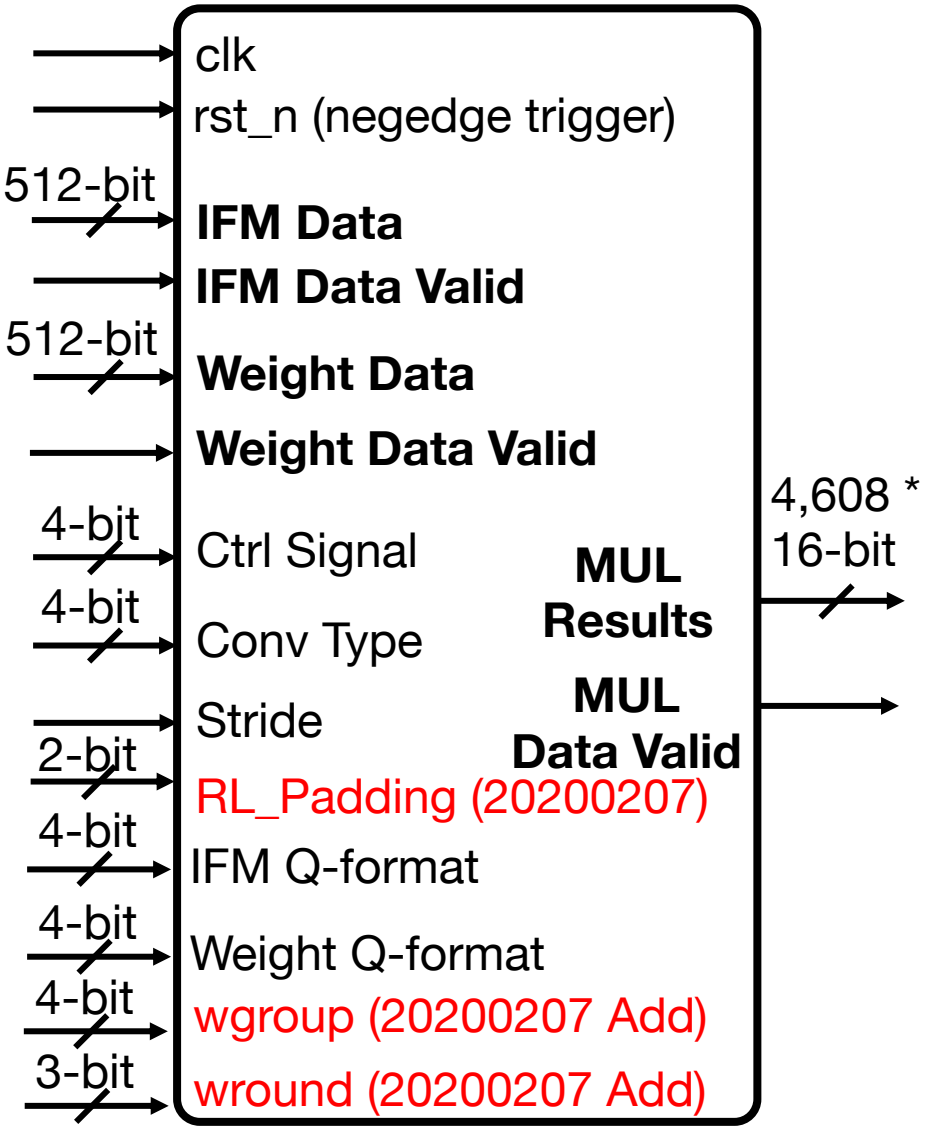




MAC Utilization

Filter type	Multiplier Number	MAC Utilization
3x3, stride = 1	(3 * 3 * 8) * 8 * 4	100%
3x3, stride = 2	(3 * 3 * 8) * 4 * 4	50%
5x5, stride = 1	(5 * 5 * 8) * 4 * 2	69% (25/36)
5x5, stride = 2	(5 * 5 * 8) * 4 * 2	69% (25/36)
7x7, stride = 1	(7 * 7 * 8) * 2 * 1	34%
7x7, stride = 2	(7 * 7 * 8) * 2 * 1	34%
1x1, stride = 1	(1 * 8 * 8) * 8 * 4	88.9% (8/9)

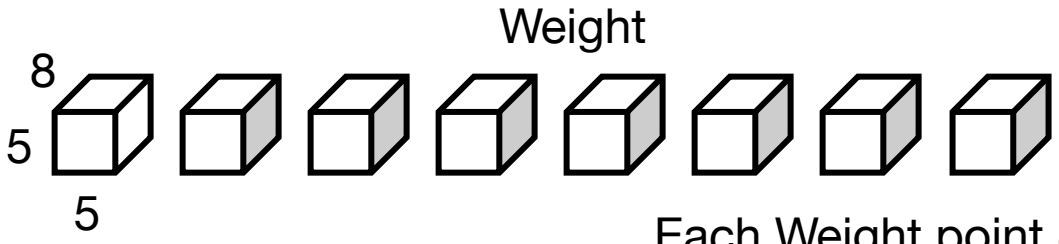
MC^3 – Interface v1 (1/20)



Signal	
IFM Data	8 (I_H) * 8 (I_C) * 8-bit = 512-bit
IFM Data Valid	1-bit
W Data	512-bit = 64 * 8-bit
W Data Valid	1-bit
Ctrl Signal	00: Reserved 01: Start 10: Finish 11: Refresh
Conv type	[3]: Reversed [2]: Traditional Conv (0) or Same Conv (1) [1:0]: 7x7(3), 5x5(2), 3x3(1), 1x1(0)
Stride	1(0) or 2(1)
RL_Padding (左右)	00: w/0 Padding 01: w Right Padding 10: w Left Padding
IFM / W Q-format	[3]:Reserved [2:0]: Q-format Ex. Fixed point xxx.xxxxx (Q5)

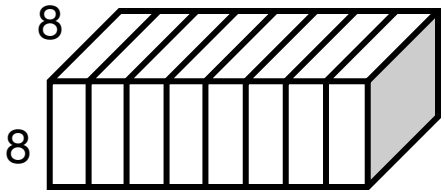
Note: Same Conv => Padding issue

MC^3 Module v1 - 3x3 (1/20)

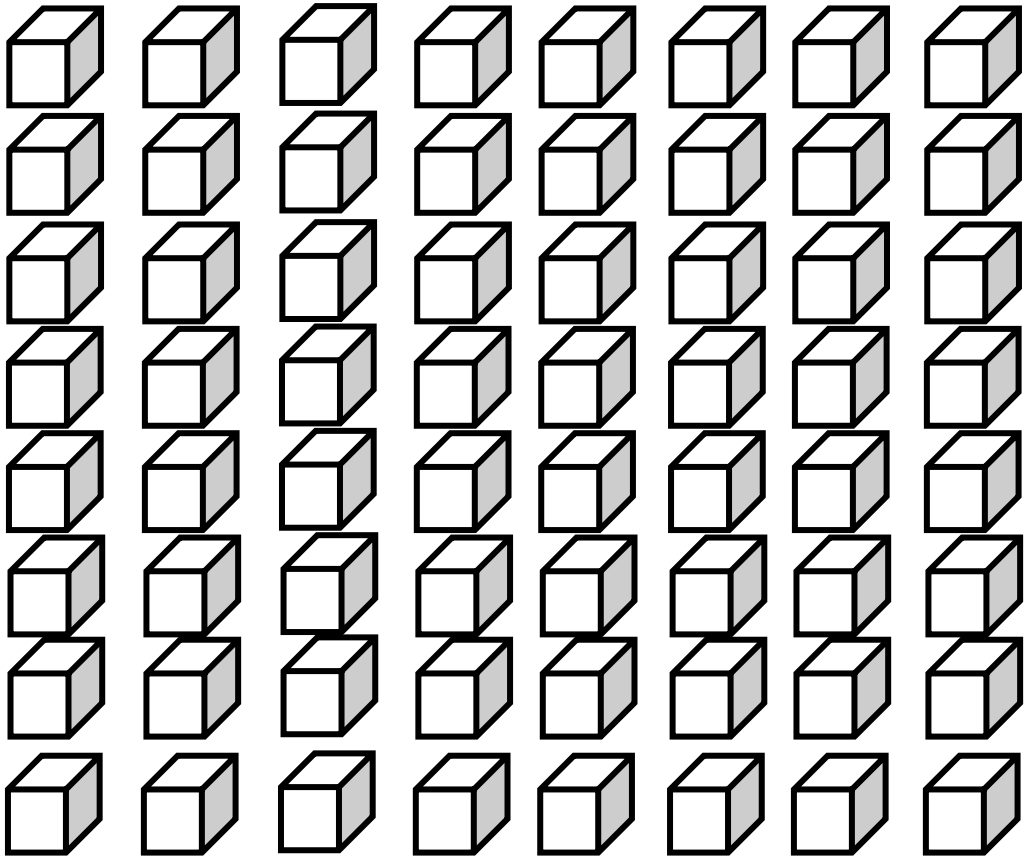


Each Weight point 8-bit (TBD)

IFM

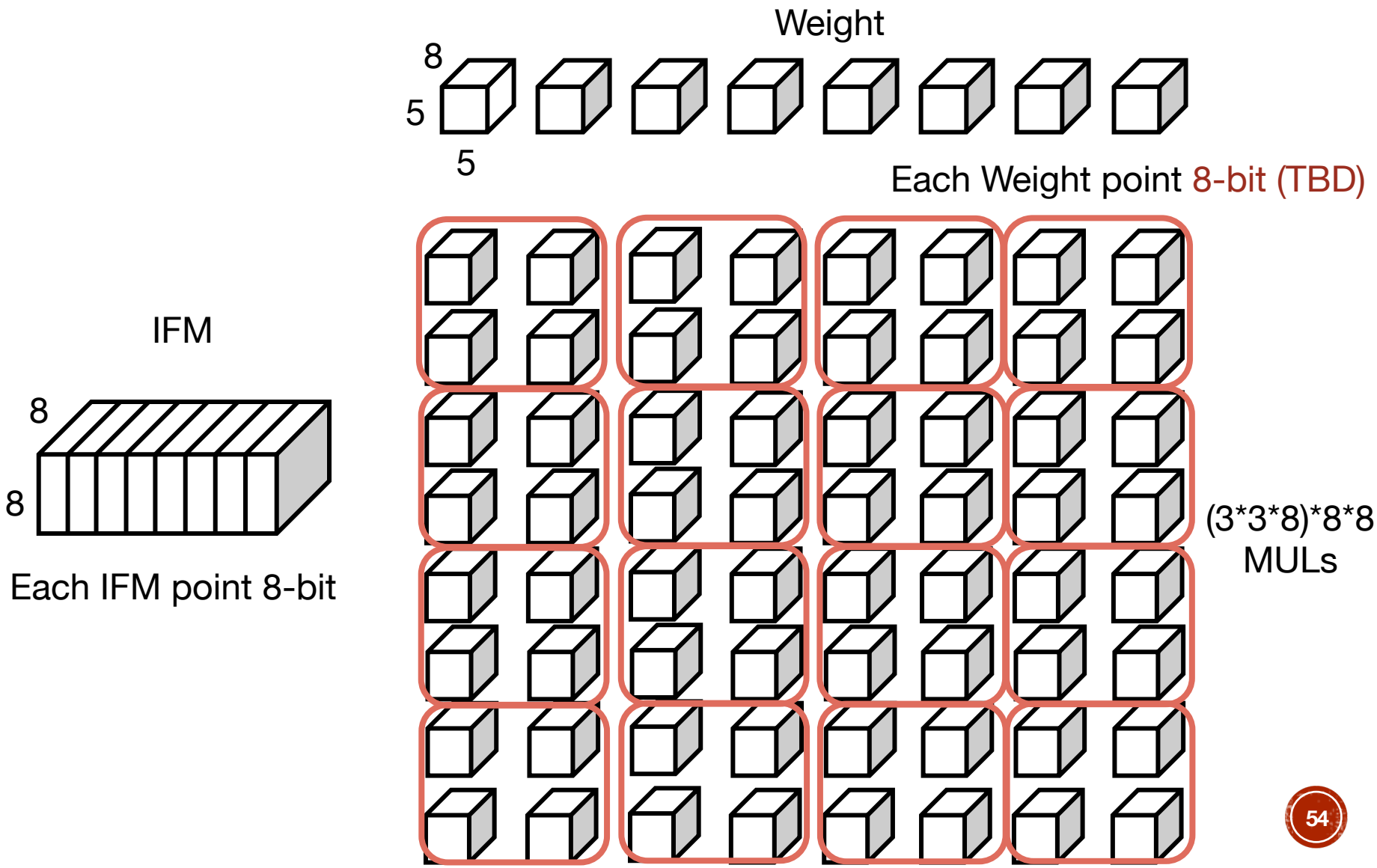


Each IFM point 8-bit

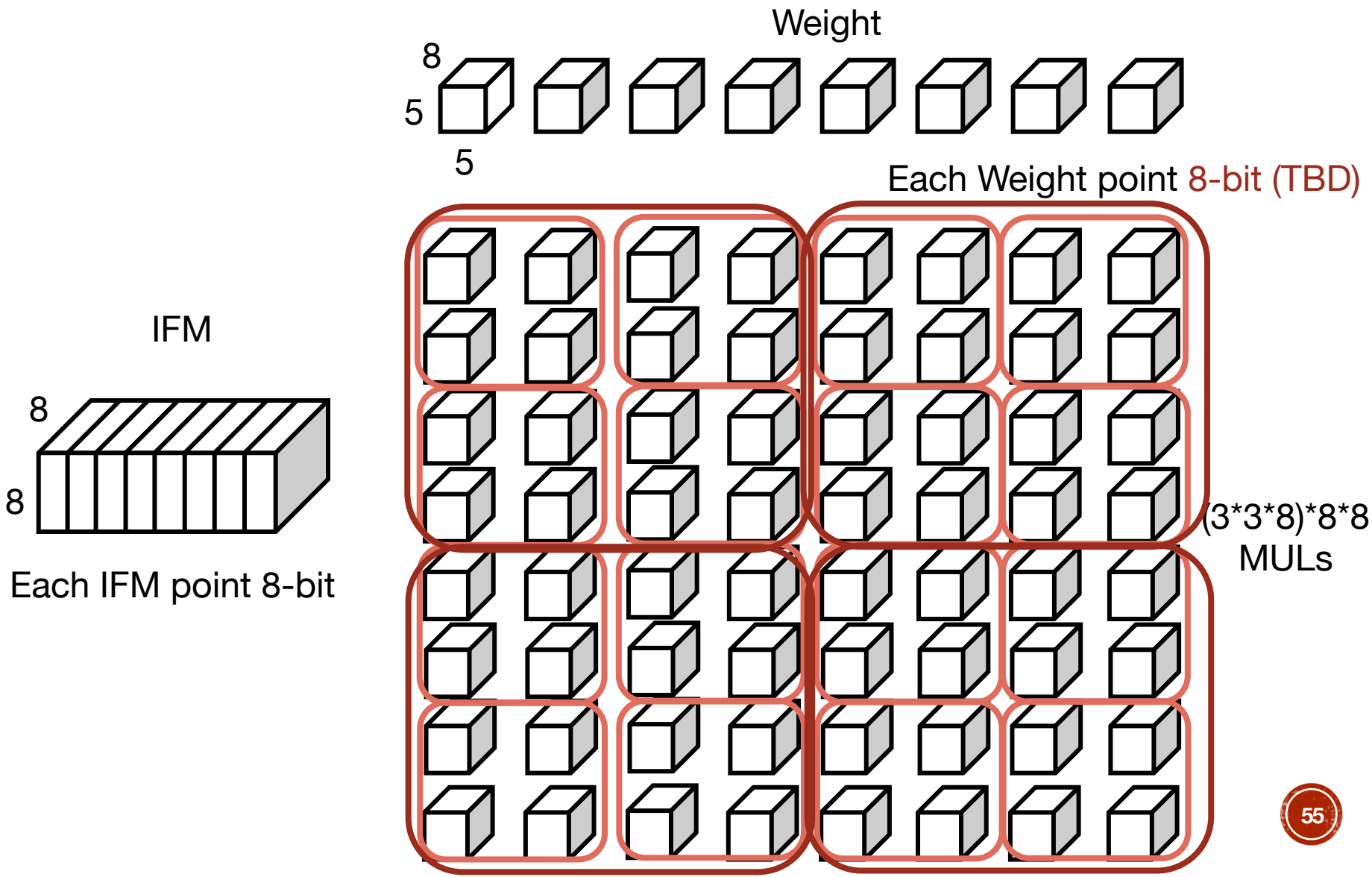


(3*3*8)*8*8 MULs

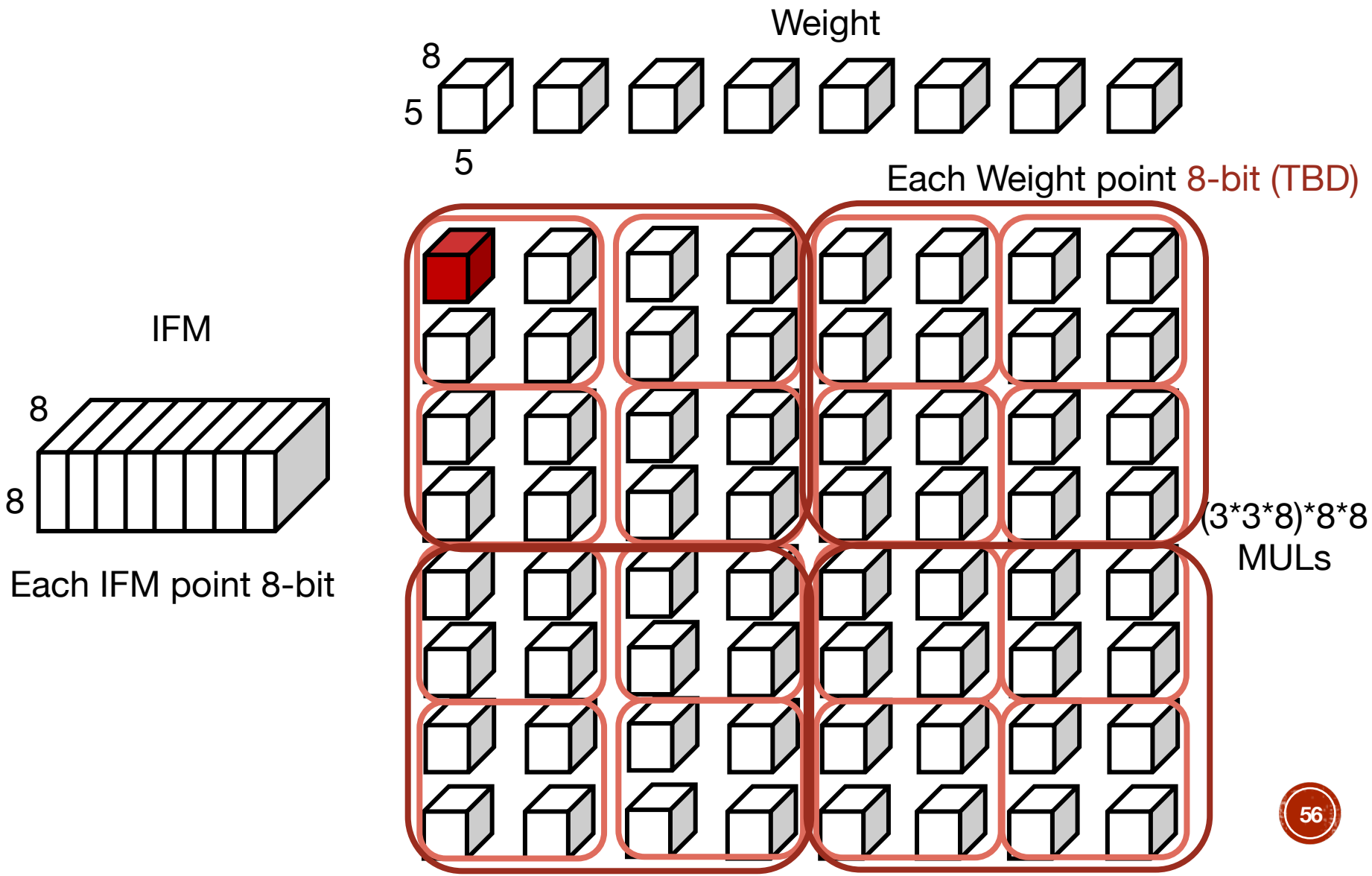
MC^3 Module v1 (1/20)



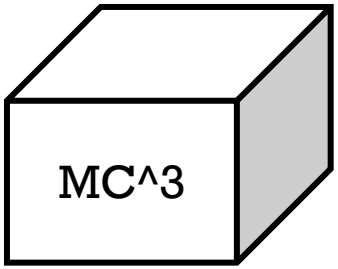
MC^3 Module v1 (1/20)



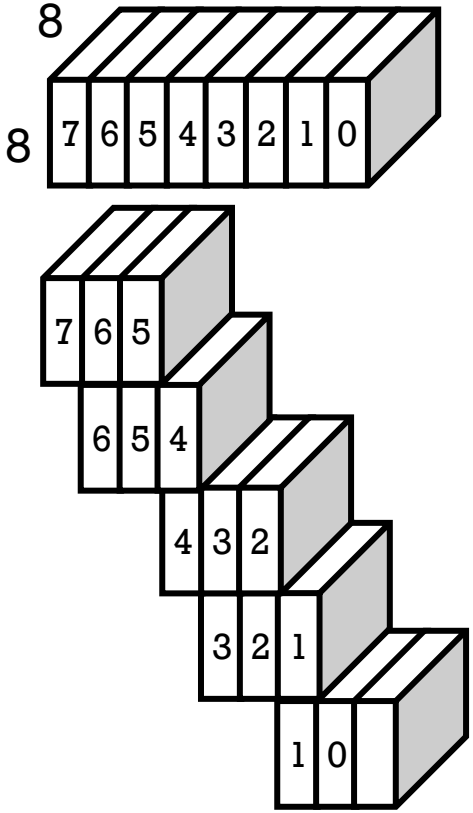
MC^3 Module v1 (1/20)

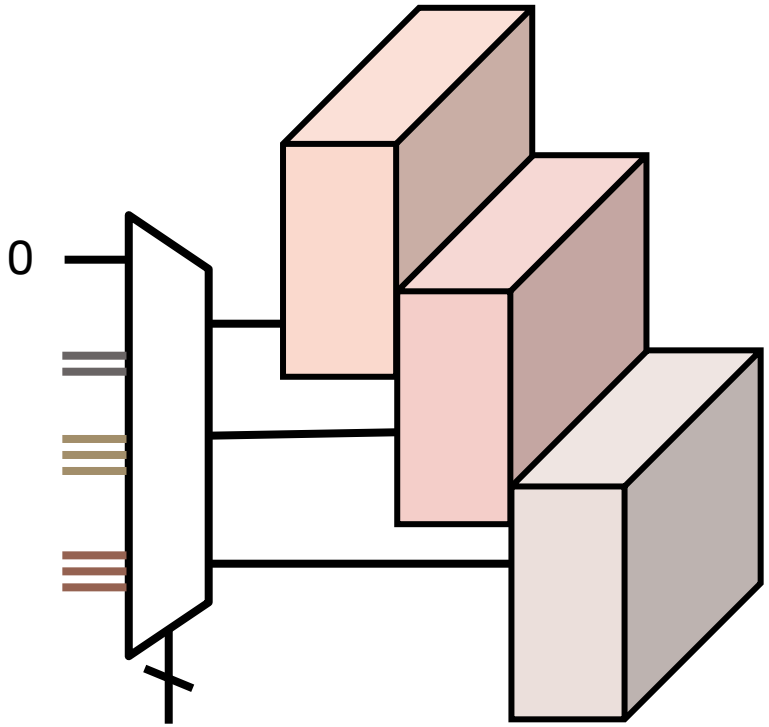


MC^3 Module v1 - IFM (1/20)

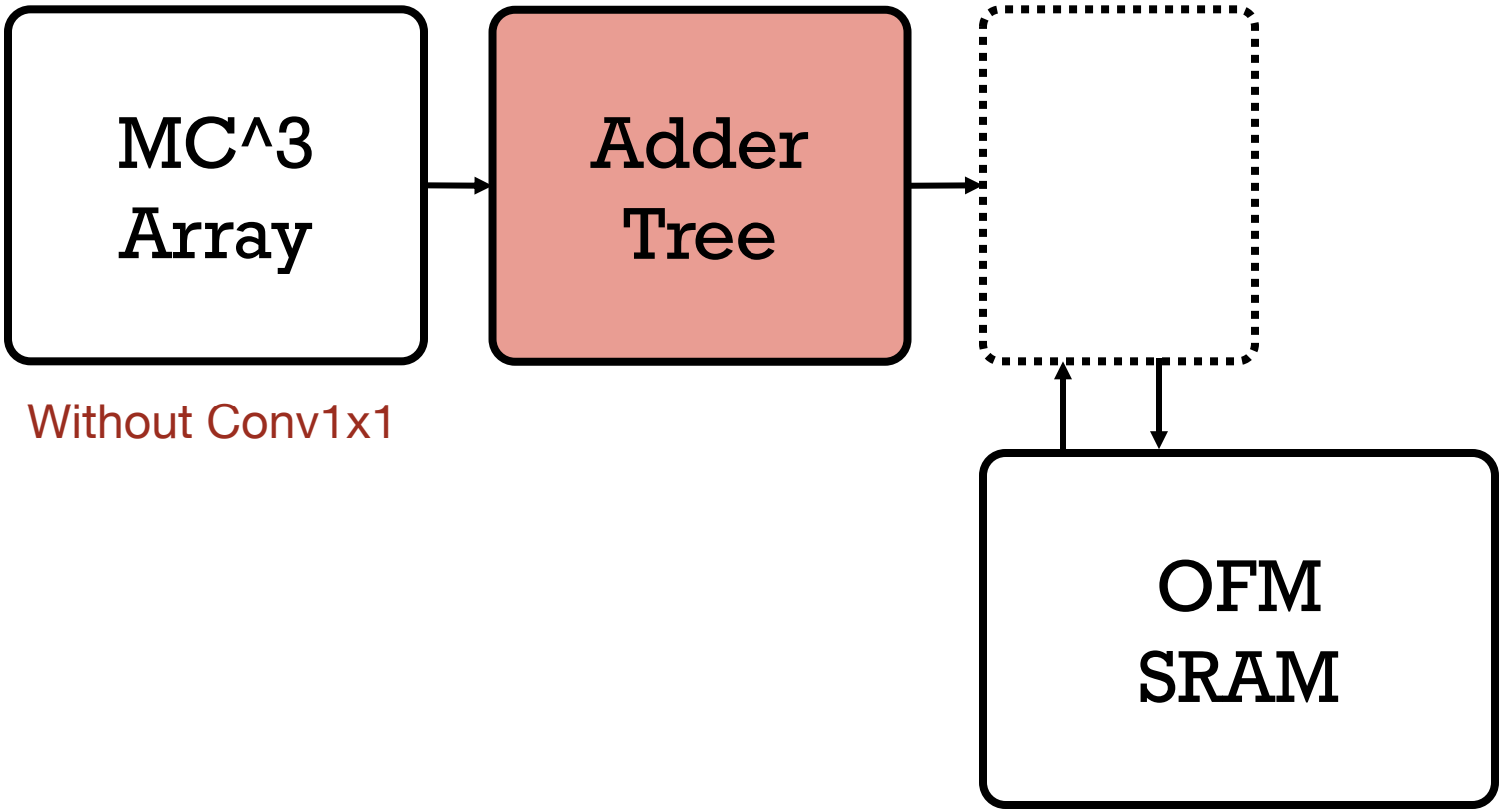


IFM (Each IFM point 8-bit)



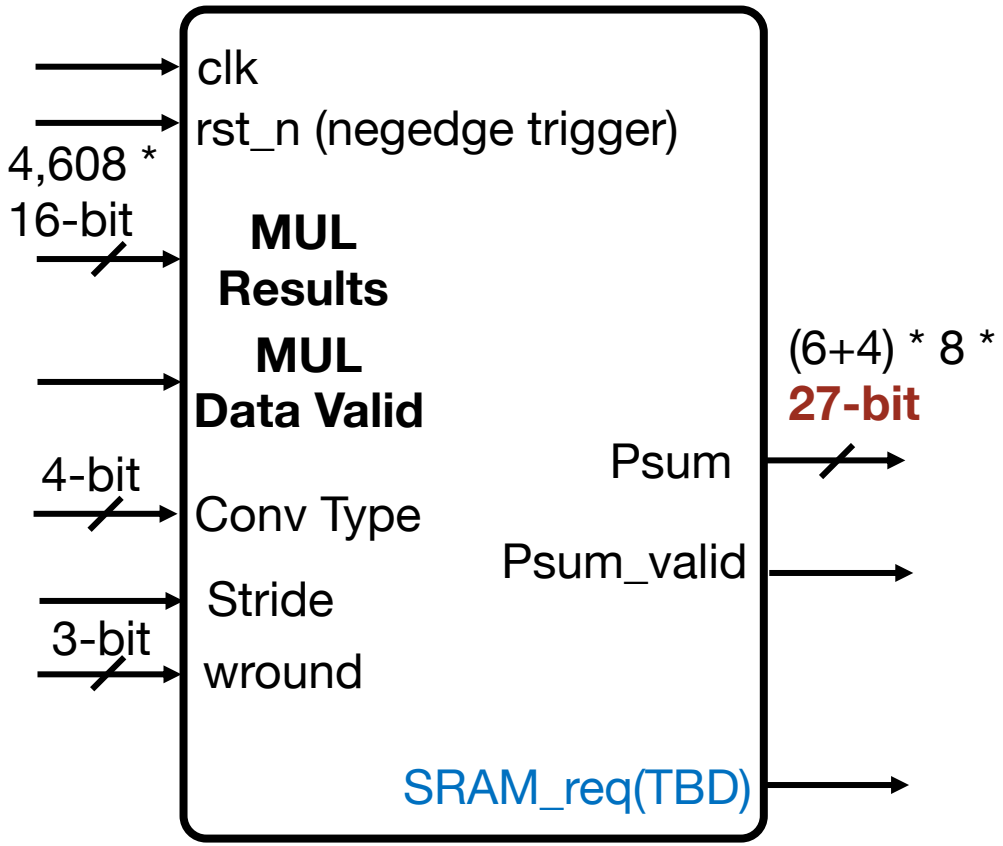


U-HarDNet Engine Overview



Without Conv1x1

Adder Tree – Interface v1

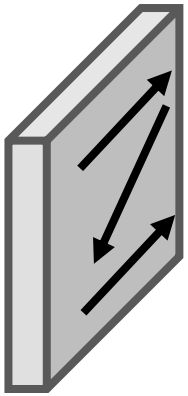


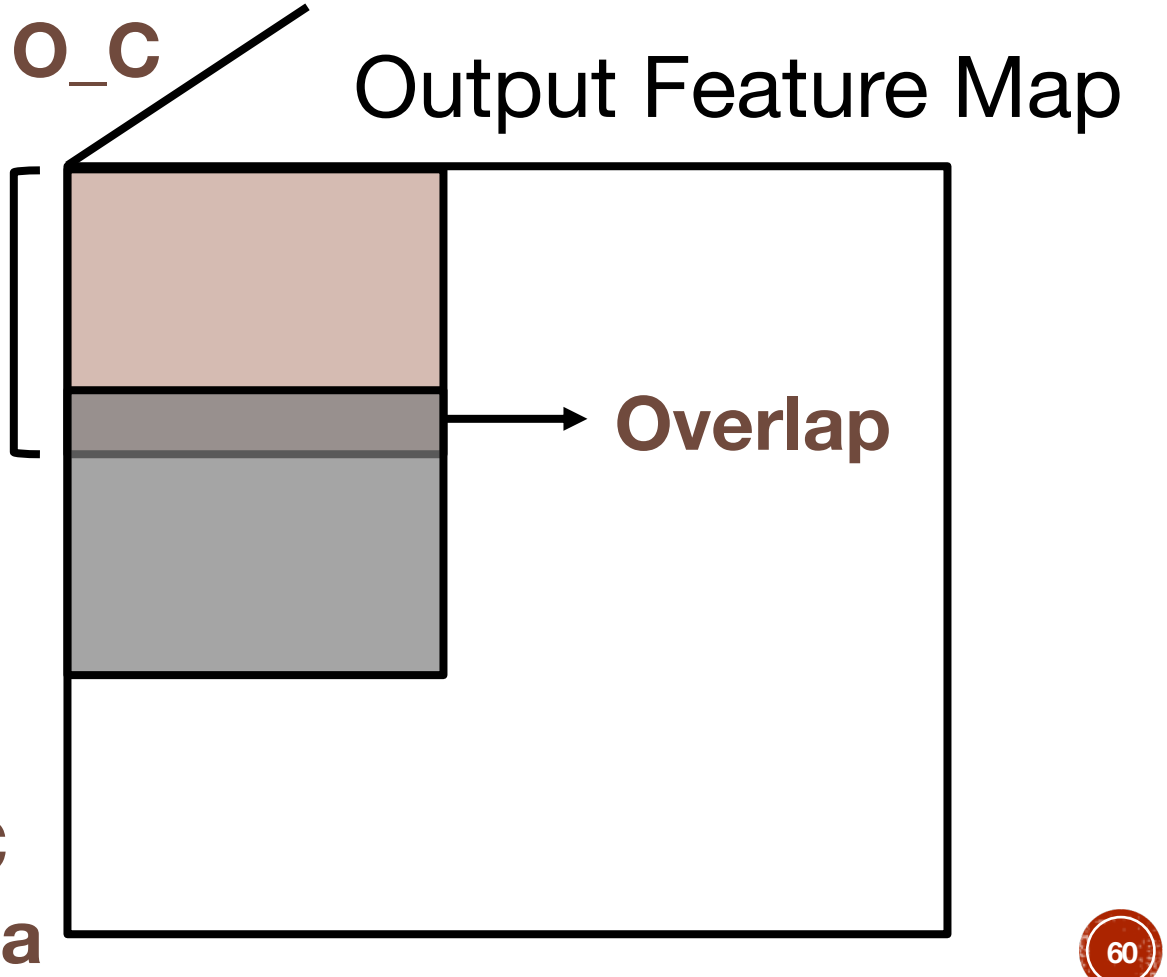
Signal	
MUL Results	4,608 * 16-bit
MUL Data Valid	1-bit
Conv type	[3]: Reversed [2]: Traditional Conv (0) or Same Conv (1) [1:0]: 7x7(3), 5x5(2), 3x3(1), 1x1(0)
Stride	Stride 1: 0 Stride 2: 1
wround	Convolution round id
Psum	(8+4) * 8 * 27-bit
Psum_valid	Conv results 計算好時拉起一個 cycle

Signed Data

Adder Tree Results

Basic Unit
Data Allocation





Psum = O_H * O_C

data

Adder Tree Results

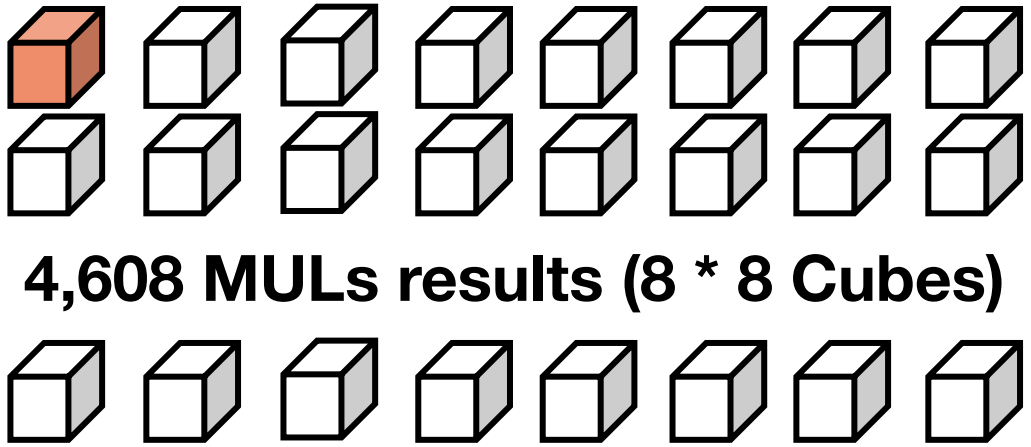
(可以搭配 P.39, P.41, P.43, P.45, P.47, P.49)

Psum = O_H * O_C = (? + ?) * ? //Color Mapping

Stride	1				2	
Round	0	1	2	3	0	1
1x1	8 * 8	x	x	x	4 * 8	x
3x3	(6+4) * 8	x	x	x	(3+2) * 8	x
5x5	4 * 4	8 * 4	x	x	(2+4) * 4	x
7x7	2 * 2	4 * 2	4 * 2	4 * 2	(1+2) * 2	4 * 2

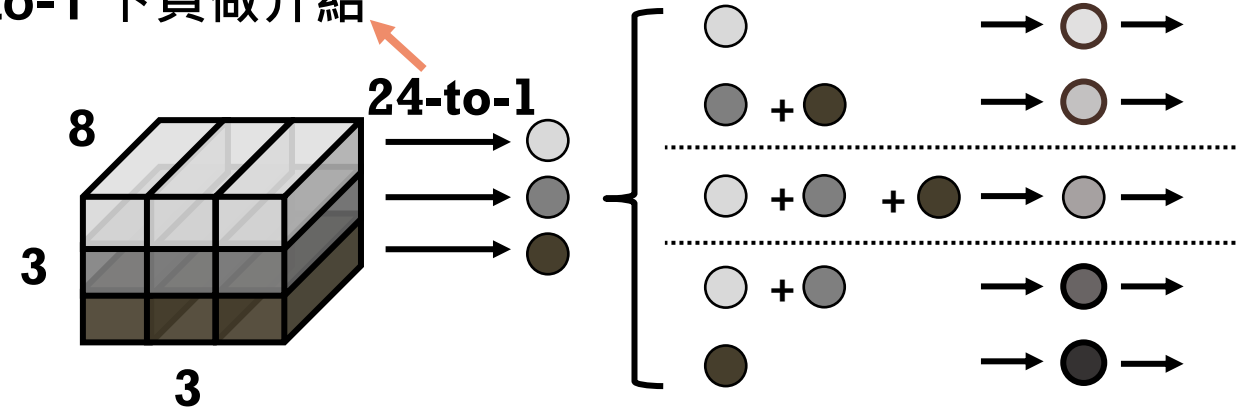
Adder Tree - Step 1

每一個 Cube
先依照 Row 切割，
得三點結果後，
加總重組為五個點．
(如下方圖所示，最終
顏色不同則為不同點)

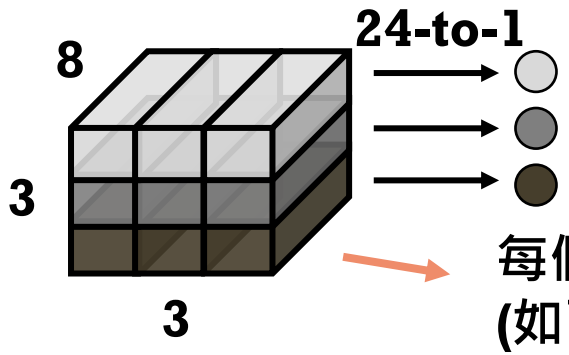


4,608 MULs results (8 * 8 Cubes)

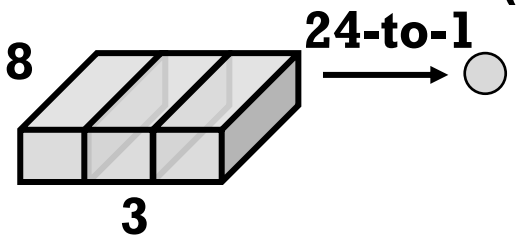
24-to-1 下頁做介紹

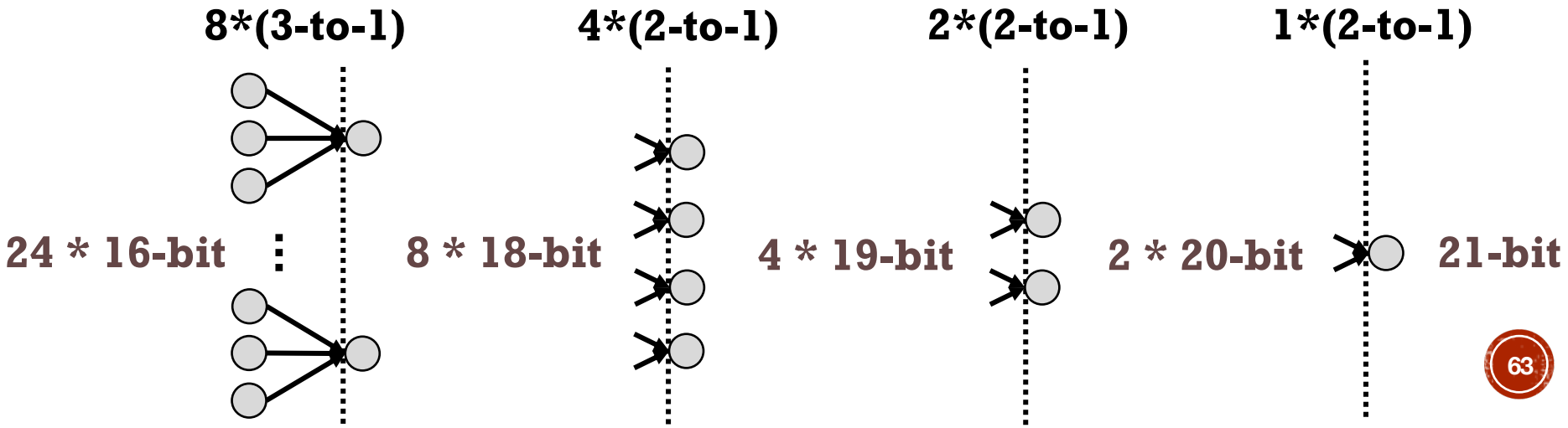


Adder Tree - Step 1

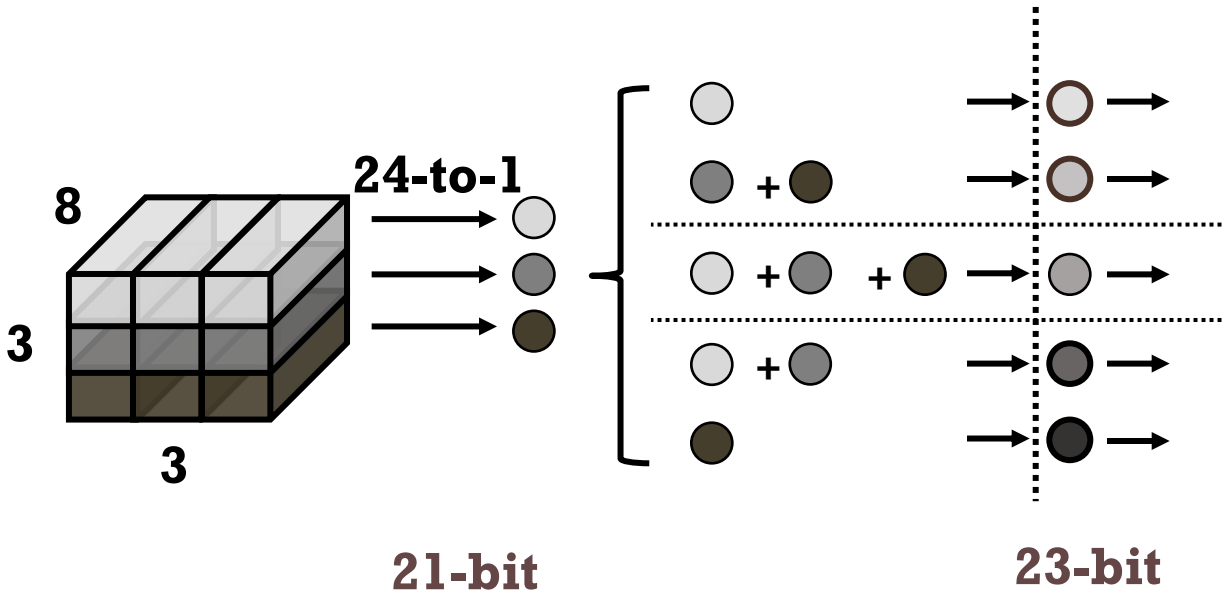


每個 Row 都需各自先加總為一點，以 24-to-1 做說明
(如下方圖所示，切 4 層 Pipe，虛線為 DFF)。





Adder Tree - Step 1



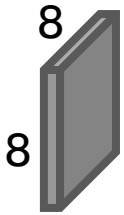
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

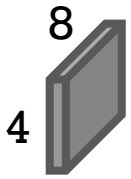
1x1

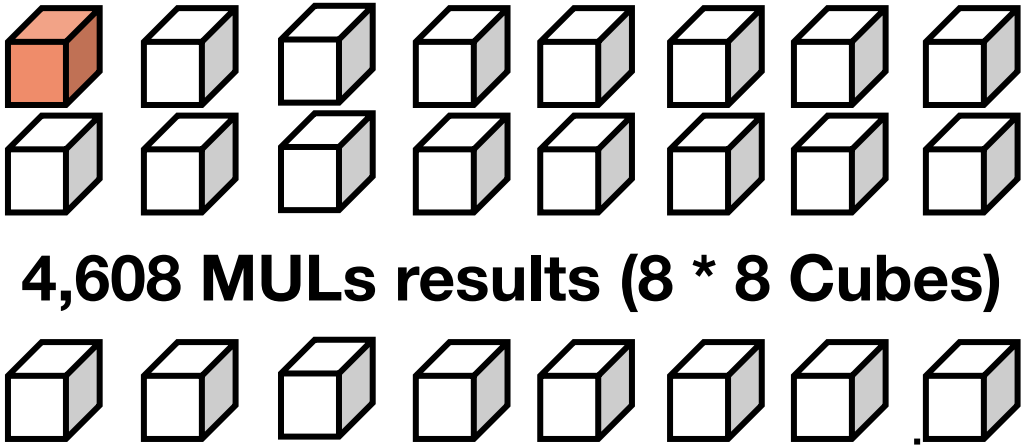
每個 Cube 各自加總為一點。

1. Stride 1

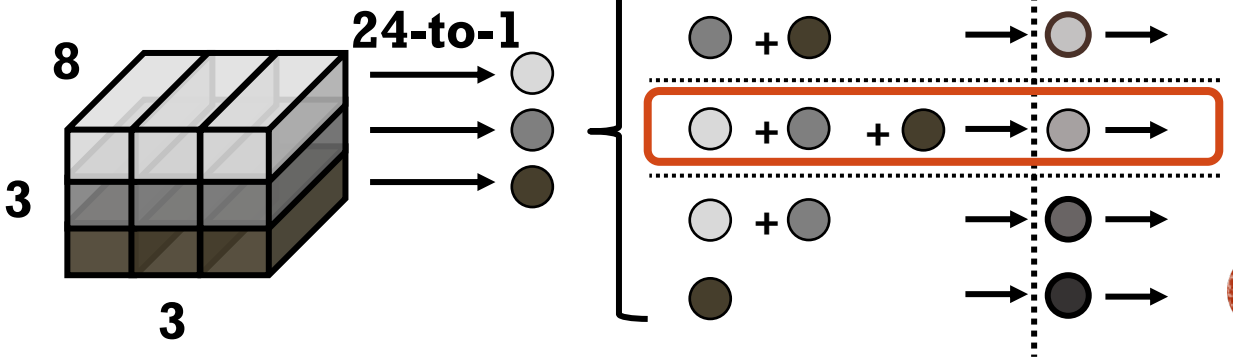


2. Stride 2





4,608 MULs results (8 * 8 Cubes)



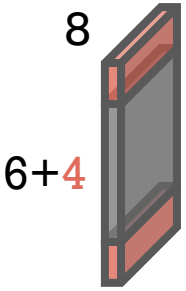
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

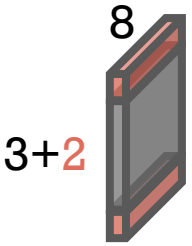
3x3

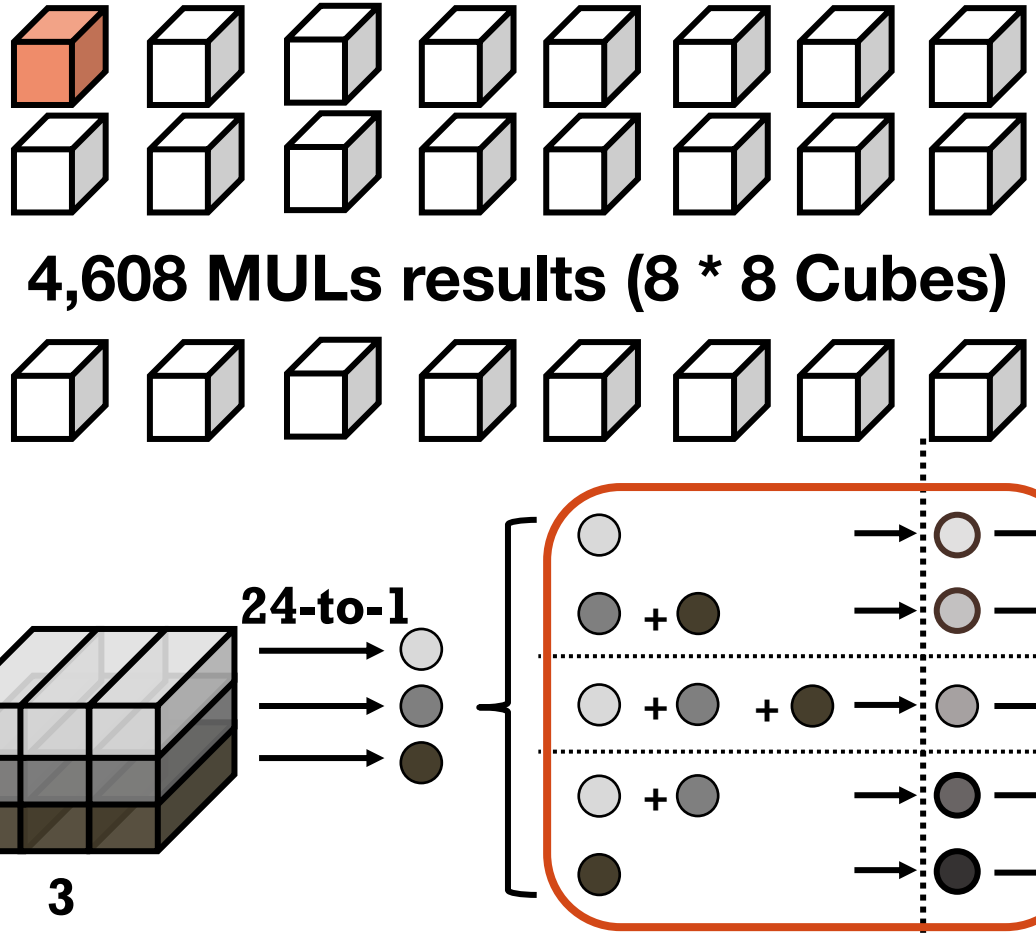
以 **column** 來看，有部分 **Cube** 需拆解。

1. Stride 1



2. Stride 2





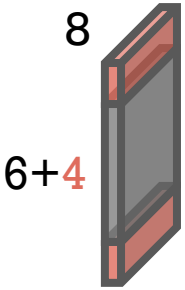
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

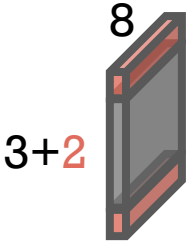
3x3

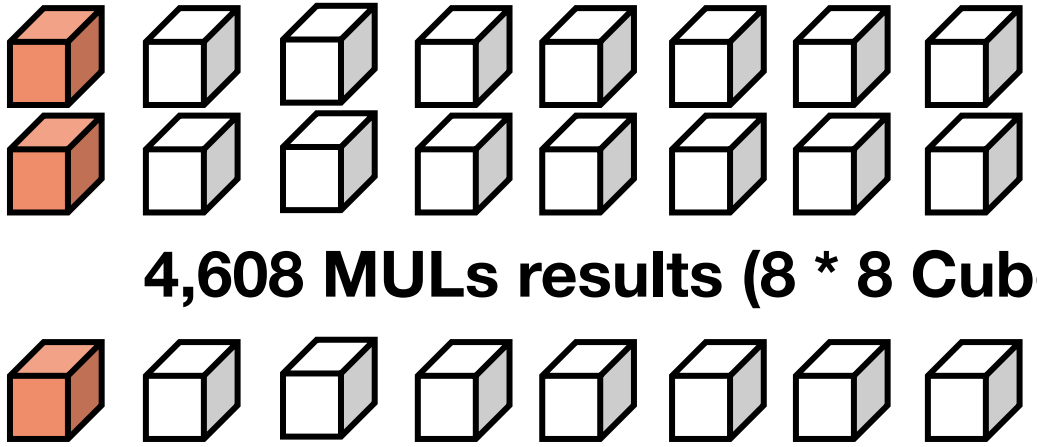
以 **column** 來看，有部分 **Cube** 需拆解。

1. Stride 1



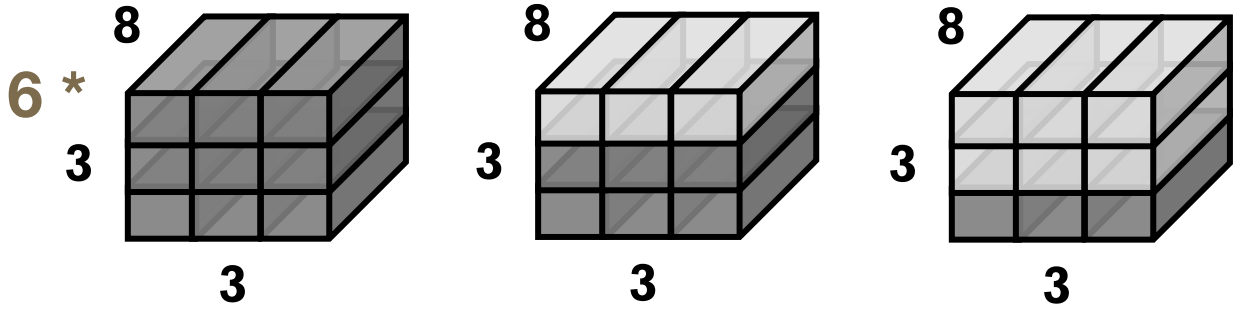
2. Stride 2





4,608 MULs results (8 * 8 Cubes)

以 **column** 來看，六個 **Cube** 為整個加總，另外兩個則會重組為四個結果。



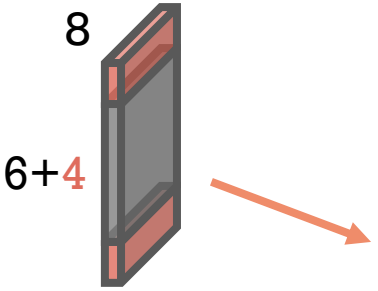
Adder Tree - Step 2

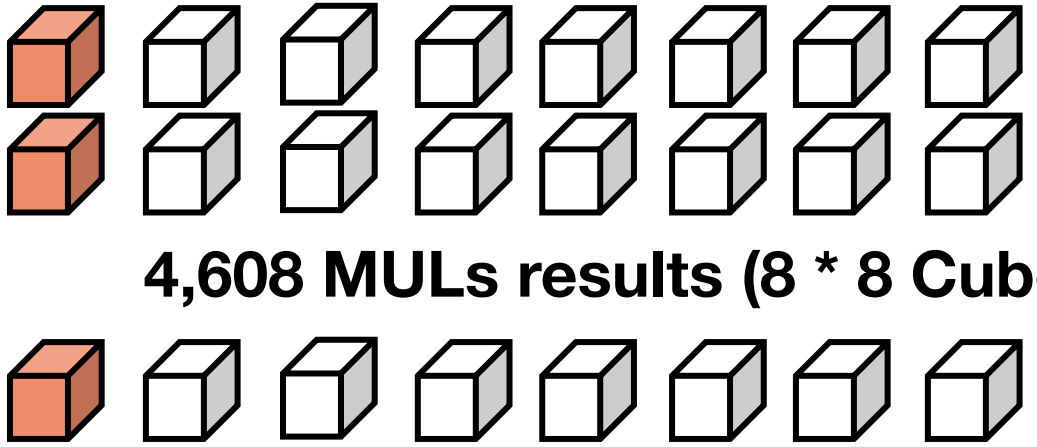
依據不同 **Conv Type, Stride, Round**，會有不同組合需求．

3x3

以 **column** 來看，有部分 **Cube** 需拆解．

1. Stride 1

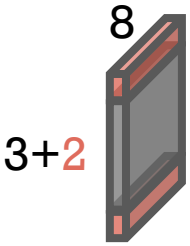


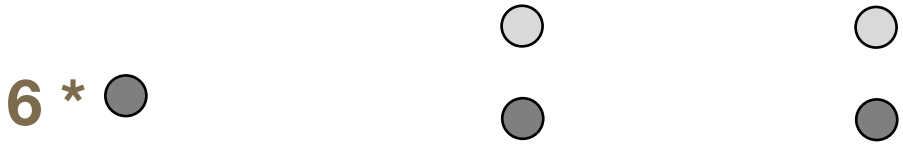


4,608 MULs results (8 * 8 Cubes)

以 **column** 來看，六個 **Cube** 為整個加總，另外兩個則會重組為四個結果．

2. Stride 2





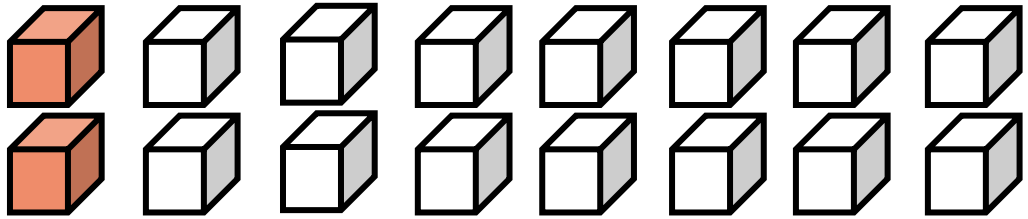
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

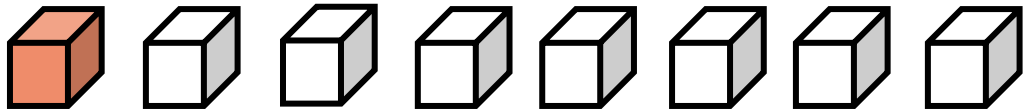
5x5

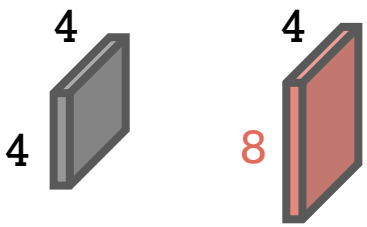
以 column 來看，有些情況下 Cube 需拆解。

1. Stride 1, 分 2 round

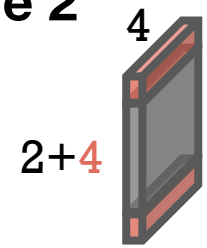


4,608 MULs results (8 * 8 Cubes)





2. Stride 2



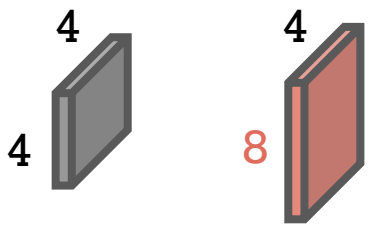
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

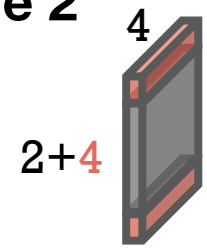
5x5

以 **column** 來看，有些情況下 **Cube** 需拆解。

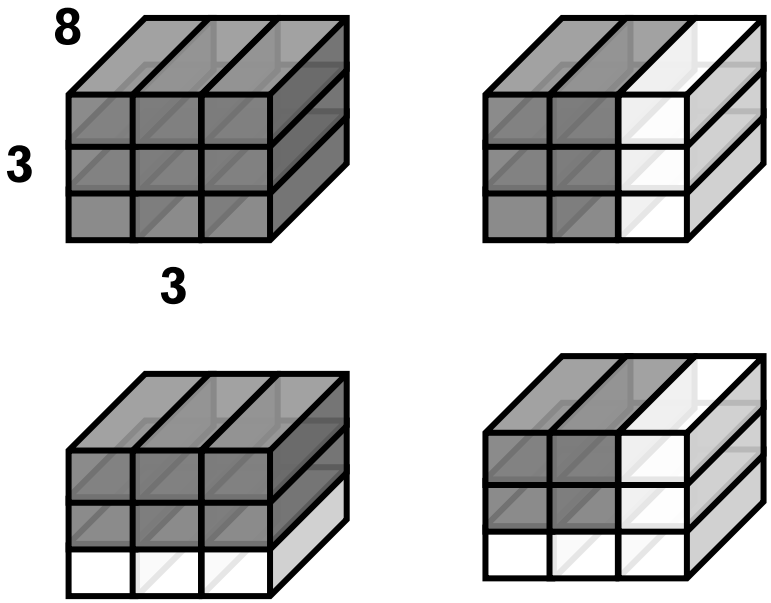
1. Stride 1, 分 2 round



2. Stride 2



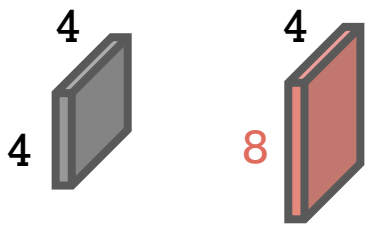
先以 **Stride 1, Round 1,** 只看 **4 Cubes** 狀況，這邊 **Cubes** 取各自加總的值，**4** 的值做相加。



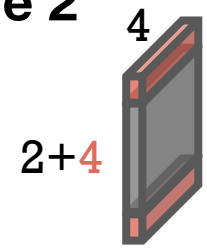
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求．

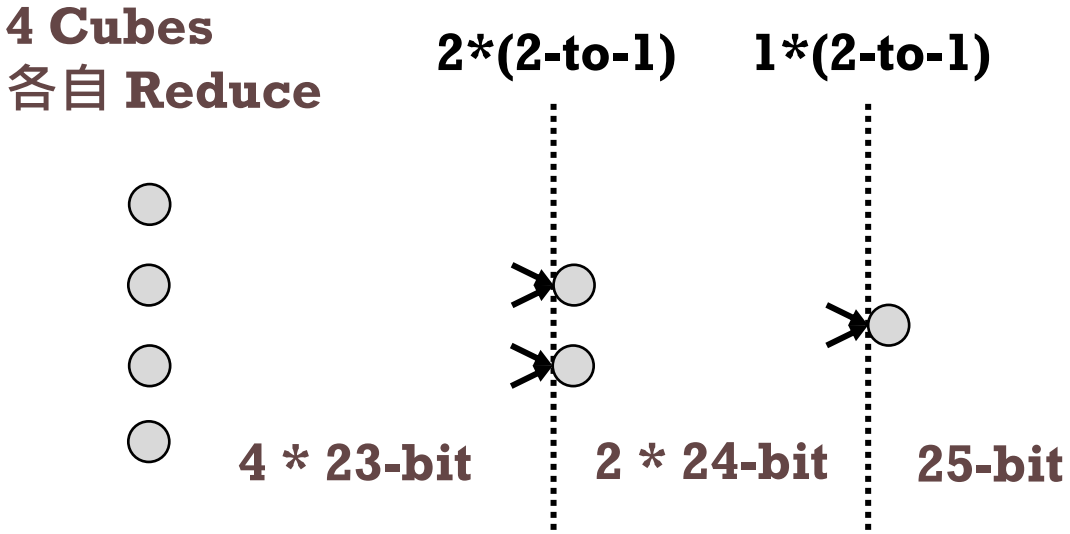
5x5
以 **column** 來看，有些情況下 **Cube** 需拆解．
1. Stride 1, 分 2 round



2. Stride 2



先以 **Stride 1, Round 1,** 只看 **4 Cubes** 狀況，這邊 **Cubes** 取各自加總的值，**4** 的值做相加．



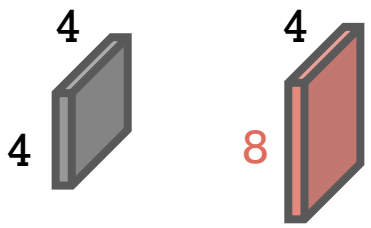
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

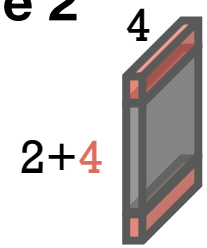
5x5

以 **column** 來看，有些情況下 **Cube** 需拆解。

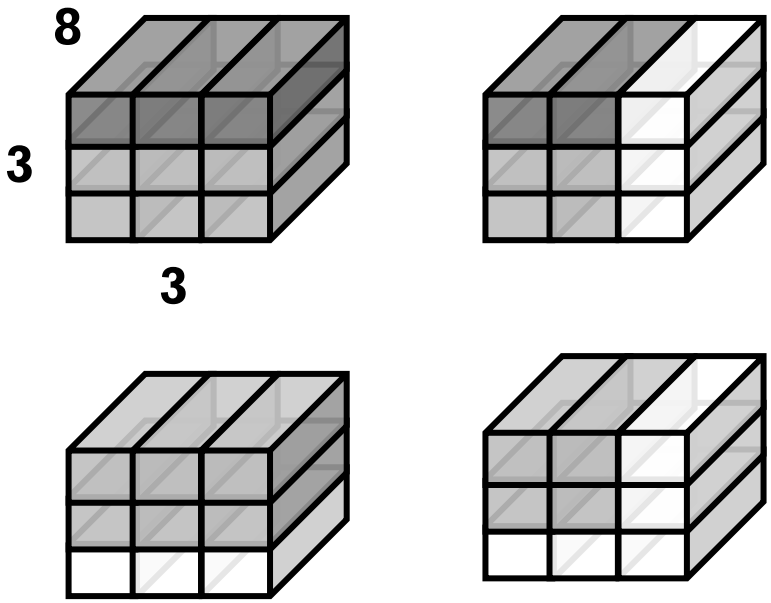
1. Stride 1, 分 2 round



2. Stride 2



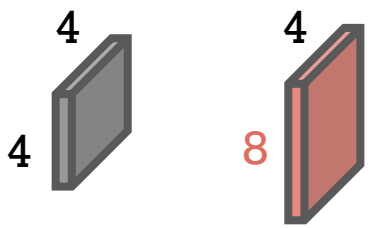
再看 Stride 1, Round 2, 一樣只看 4 Cubes 狀況，這邊牽涉到重組問題，這邊舉其中一種組合(1+4)。



Adder Tree - Step 2

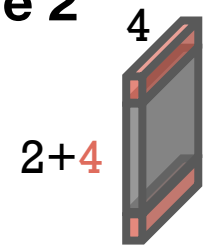
依據不同 **Conv Type, Stride, Round**，會有不同組合需求．

5x5
以 **column** 來看，有些情況下 **Cube** 需拆解．
1. Stride 1, 分 2 round



再看 **Stride 1, Round 2,** 一樣只看 **4 Cubes** 狀況，這邊牽涉到重組問題．

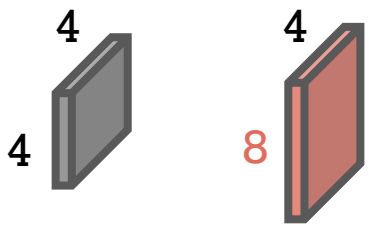
2. Stride 2



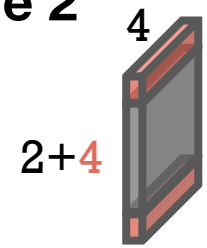
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求．

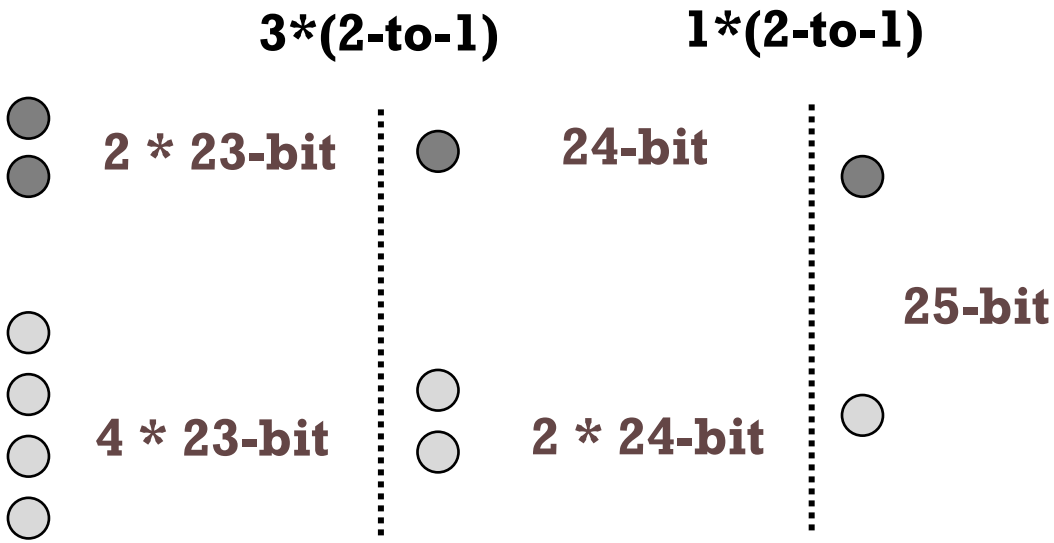
5x5
以 **column** 來看，有些情況下 **Cube** 需拆解．
1. Stride 1, 分 2 round



2. Stride 2



再看 Stride 1, Round 2, 一樣只看 4 Cubes 狀況，這邊牽涉到重組問題．



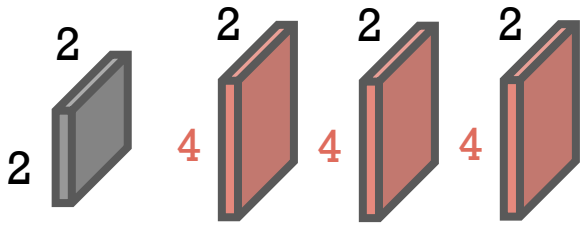
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

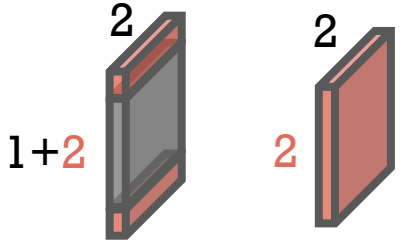
7x7

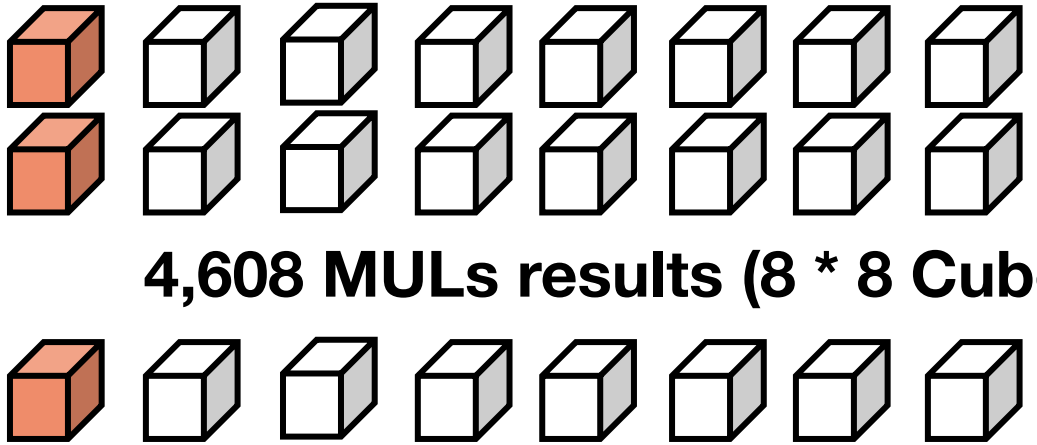
以 column 來看，有些情況下 Cube 需拆解。

1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round





4,608 MULs results (8 * 8 Cubes)

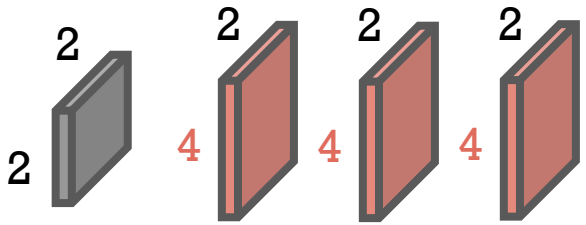
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

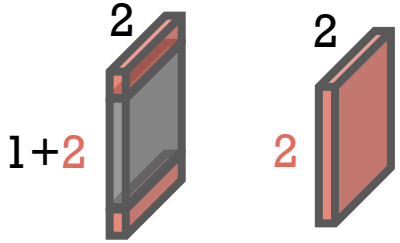
7x7

以 column 來看，有些情況下 Cube 需拆解。

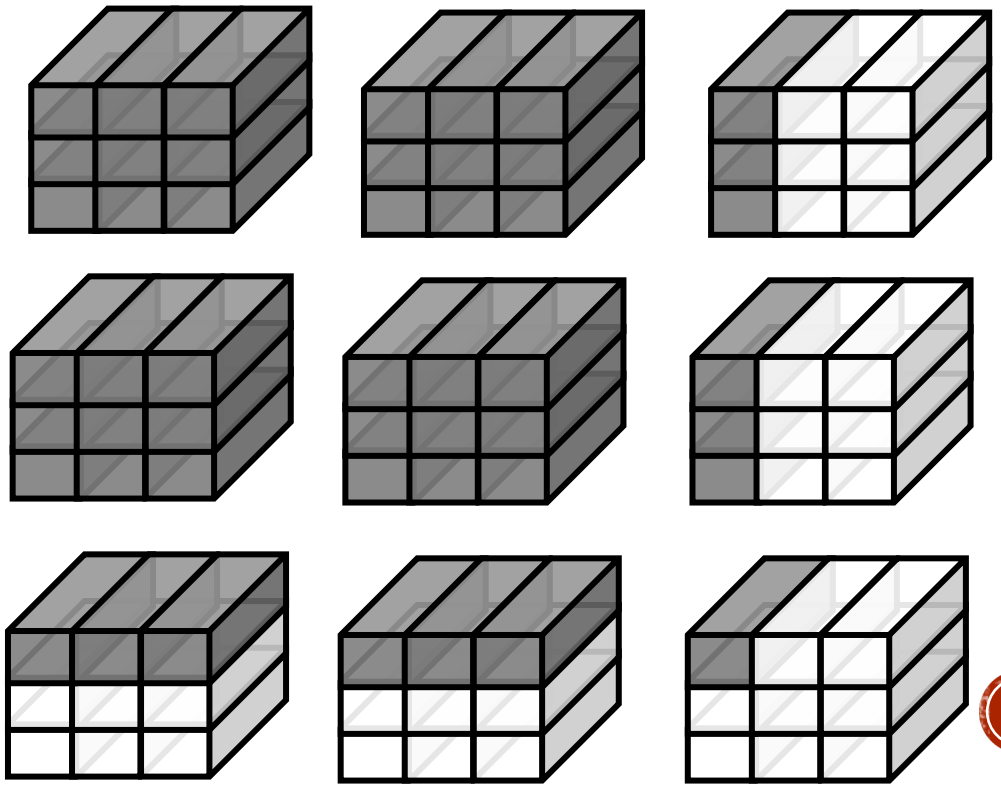
1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round



先以 Stride 1, Round 1, 只看 9 Cubes 狀況，這邊 Cubes 取各自加總的值，最後則有 9 個值需相加，



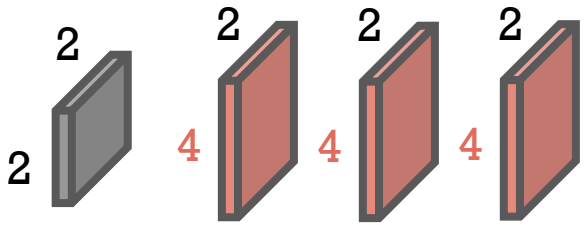
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求．

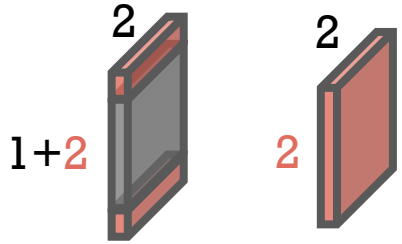
7x7

以 column 來看，有些情況下 Cube 需拆解．

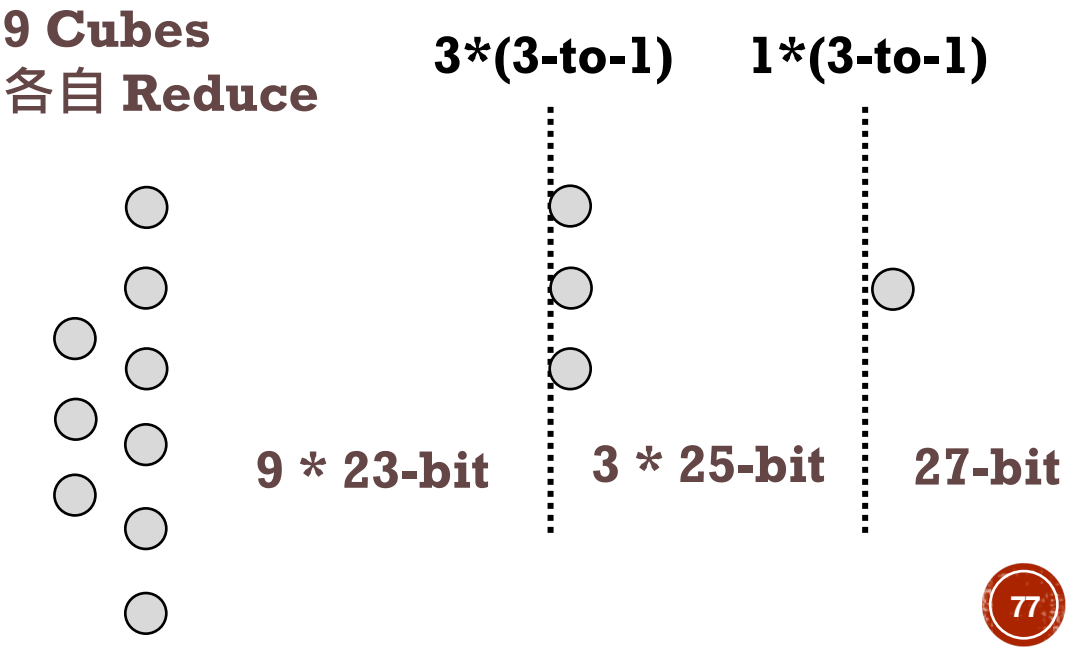
1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round



先以 Stride 1, Round 1, 只看 9 Cubes 狀況，這邊 Cubes 取各自加總的值，最後則有 9 個值需相加，



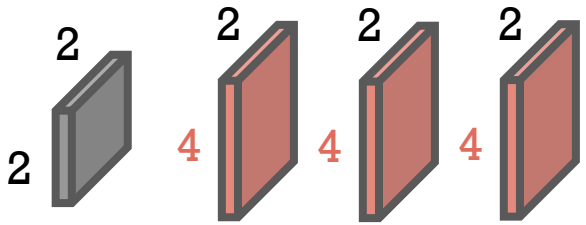
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

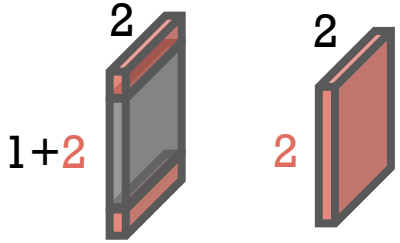
7x7

以 **column** 來看，有些情況下 **Cube** 需拆解。

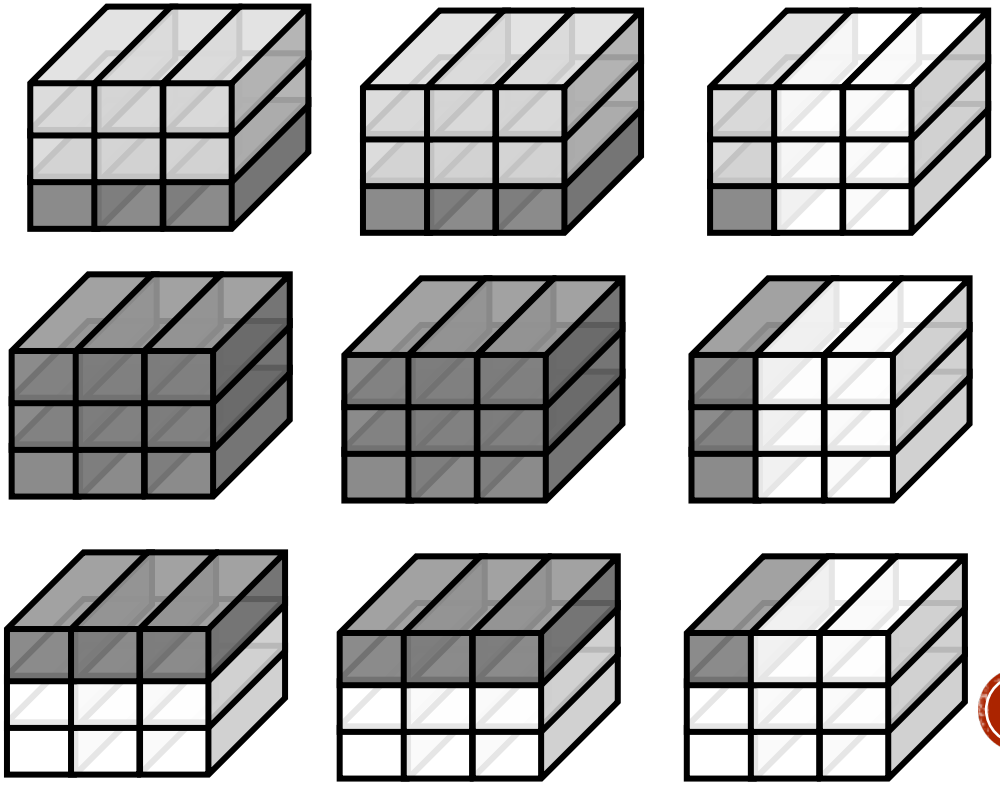
1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round



先以 **Stride 1, Round 2,** 只看 **9 Cubes** 狀況，這邊牽涉到重組問題，這邊舉其中一種組合 **(2+5)**。



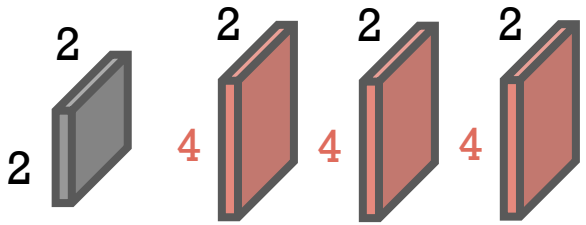
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

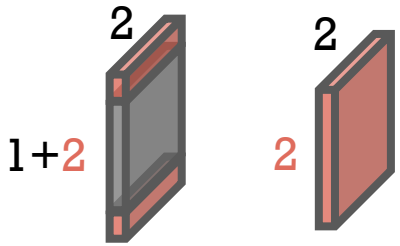
7x7

以 column 來看，有些情況下 Cube 需拆解。

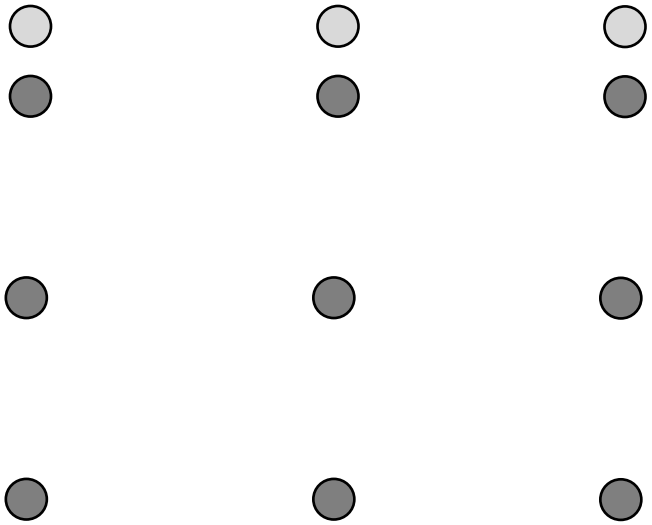
1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round



先以 Stride 1, Round 2, 只看 9 Cubes 狀況，這邊牽涉到重組問題，這邊舉其中一種組合 (2+5)。



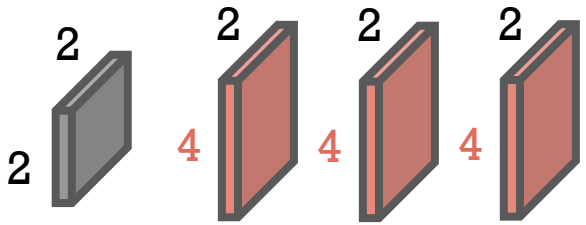
Adder Tree - Step 2

依據不同 **Conv Type, Stride, Round**，會有不同組合需求。

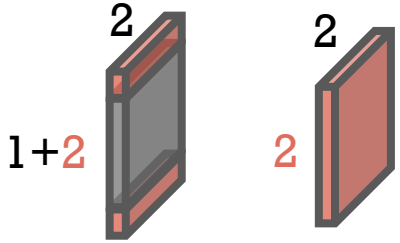
7x7

以 column 來看，有些情況下 Cube 需拆解。

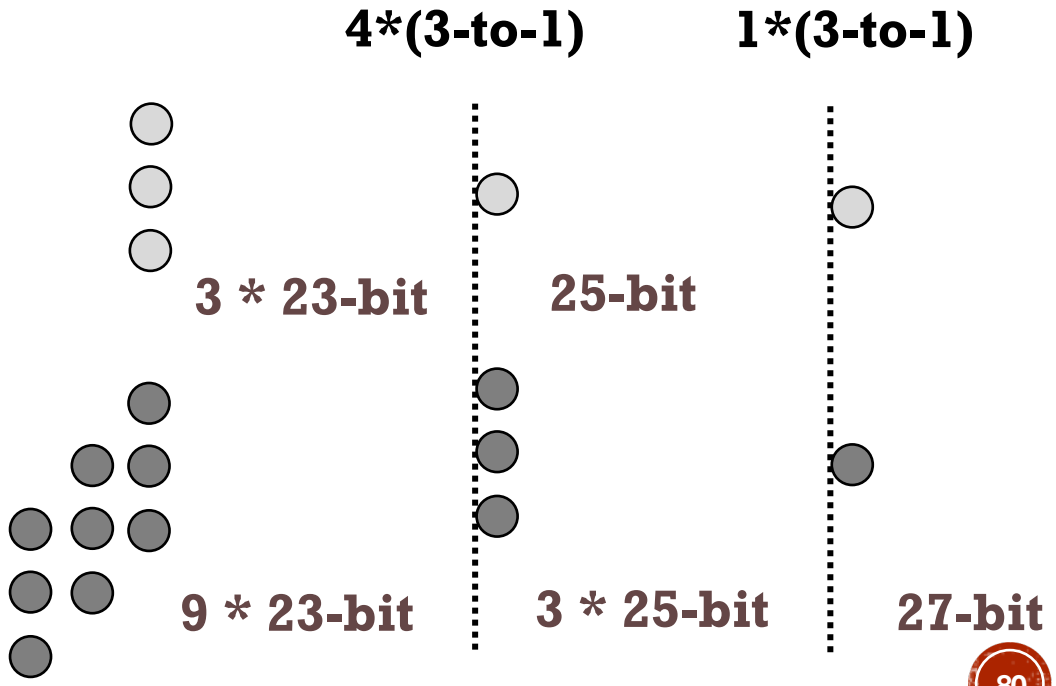
1. Stride 1, 分 4 round



2. Stride 2 , 分 2 round

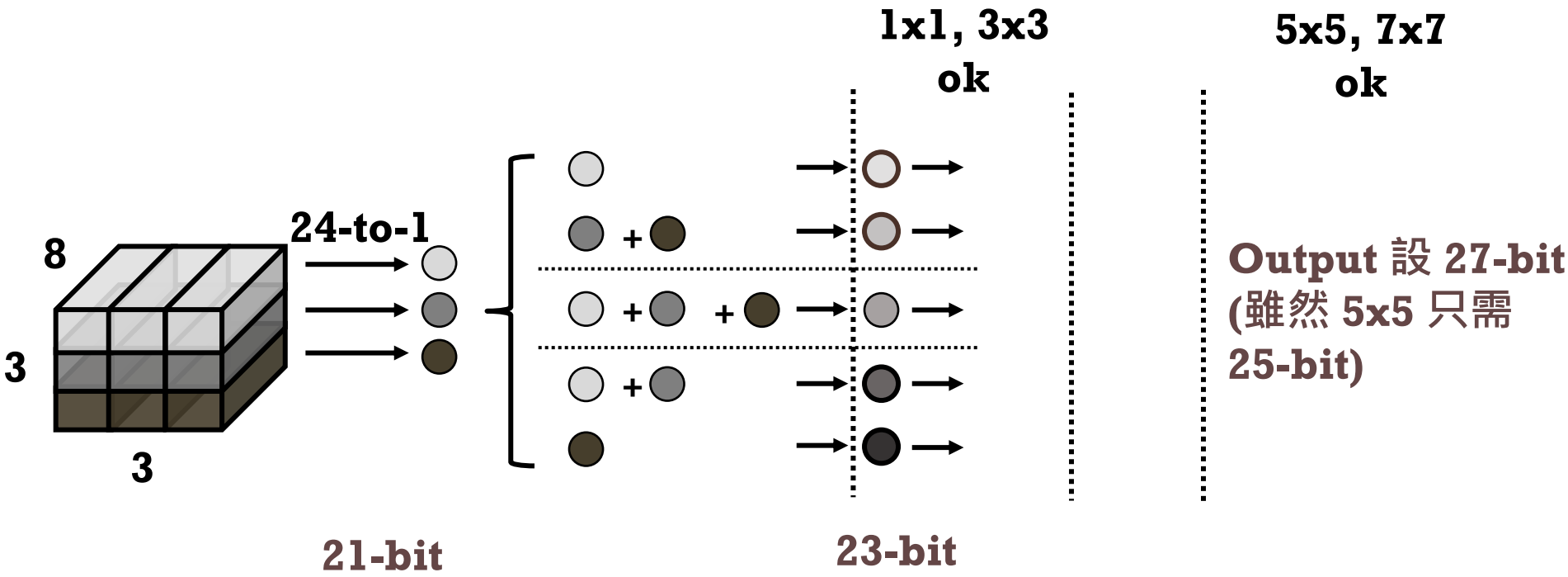


先以 Stride 1, Round 2, 只看 9 Cubes 狀況，這邊牽涉到重組問題，這邊舉其中一種組合 (2+5)。



Adder Tree - Step 3

輸出時，valid 要拉起，在不同 conv type 下，valid 拉起時間不同

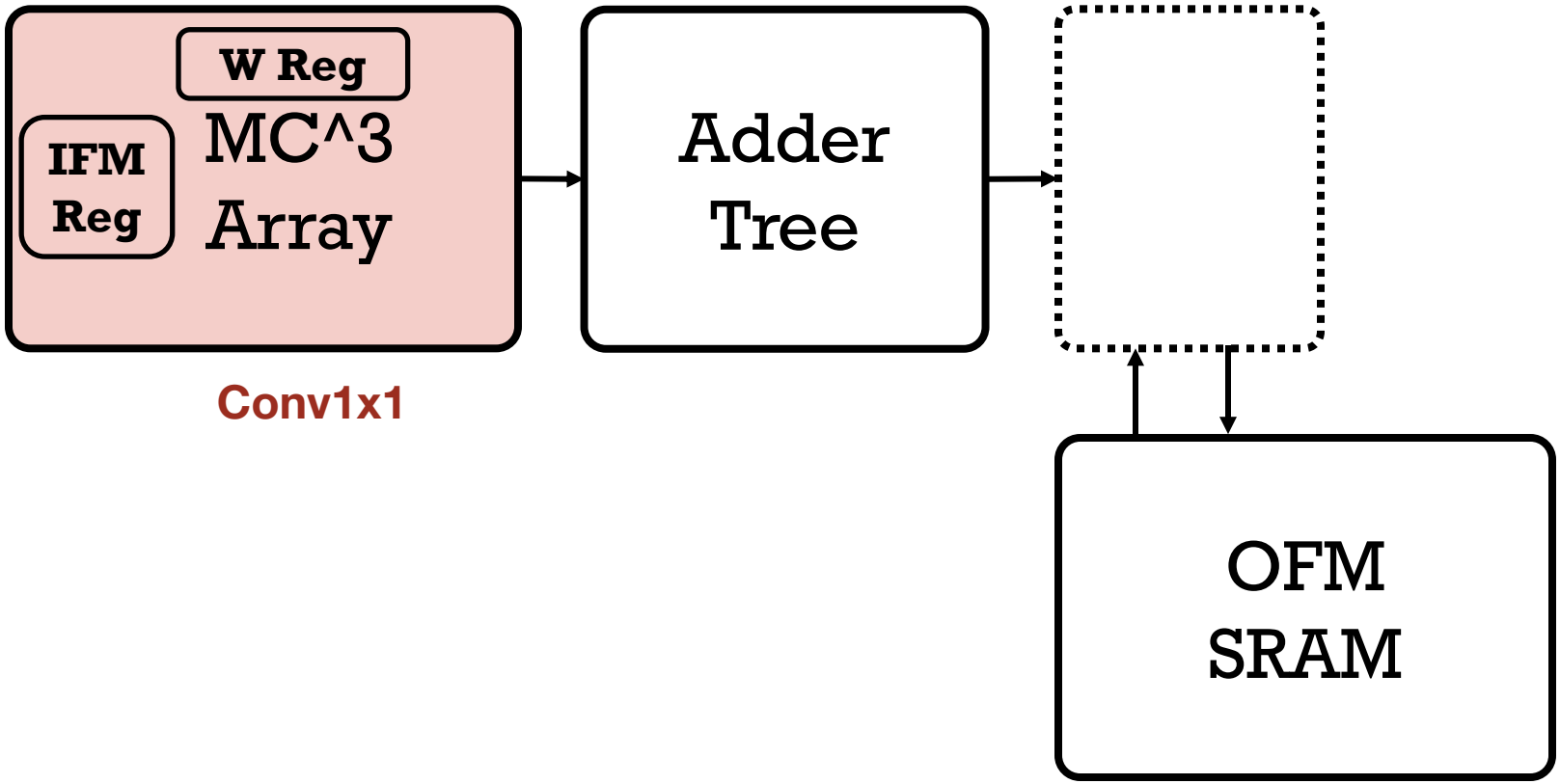


輸出順序由恒軒決定

Adder Tree - Step 4

與前次的結果相加，這部分下一次說明 (與 **SRAM** 之間的溝通)

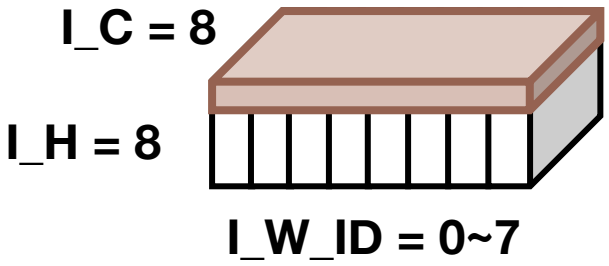
U-HarDNet Engine Overview



Conv 1x1 – Data Mapping

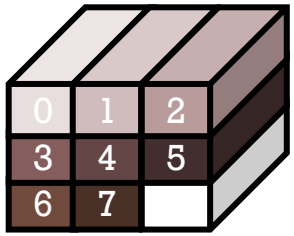
1x1 的部分，因其特性對於 Weight IO 需求量較大，也為了拉高 Cube 使用率，所以在 1x1 時會調整擺放方式．
以 IFM Reg 來看的話，需先填滿 8 slices，以下展示其對應的 Cube．

IFM Reg



假設框好同為 H = 0 的部分
(咖啡色區域)

Cube

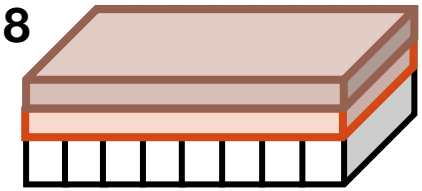


對應到其中一組 Weight，
依序放置 I_W_ID = 0 ~ 7

Conv 1x1 - Data Mapping

再從 O_C(1) * O_H(8) Cubes 來看其分配 ·

IFM Reg

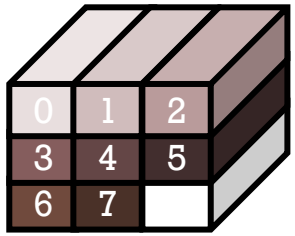


I_C = 8

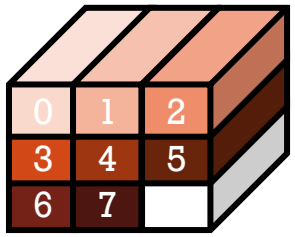
I_H_ID = 0~7

I_W_ID = 0~7

框顏色對應相同
Row_ID 的 Cube



I_H_ID = 0



I_H_ID = 1

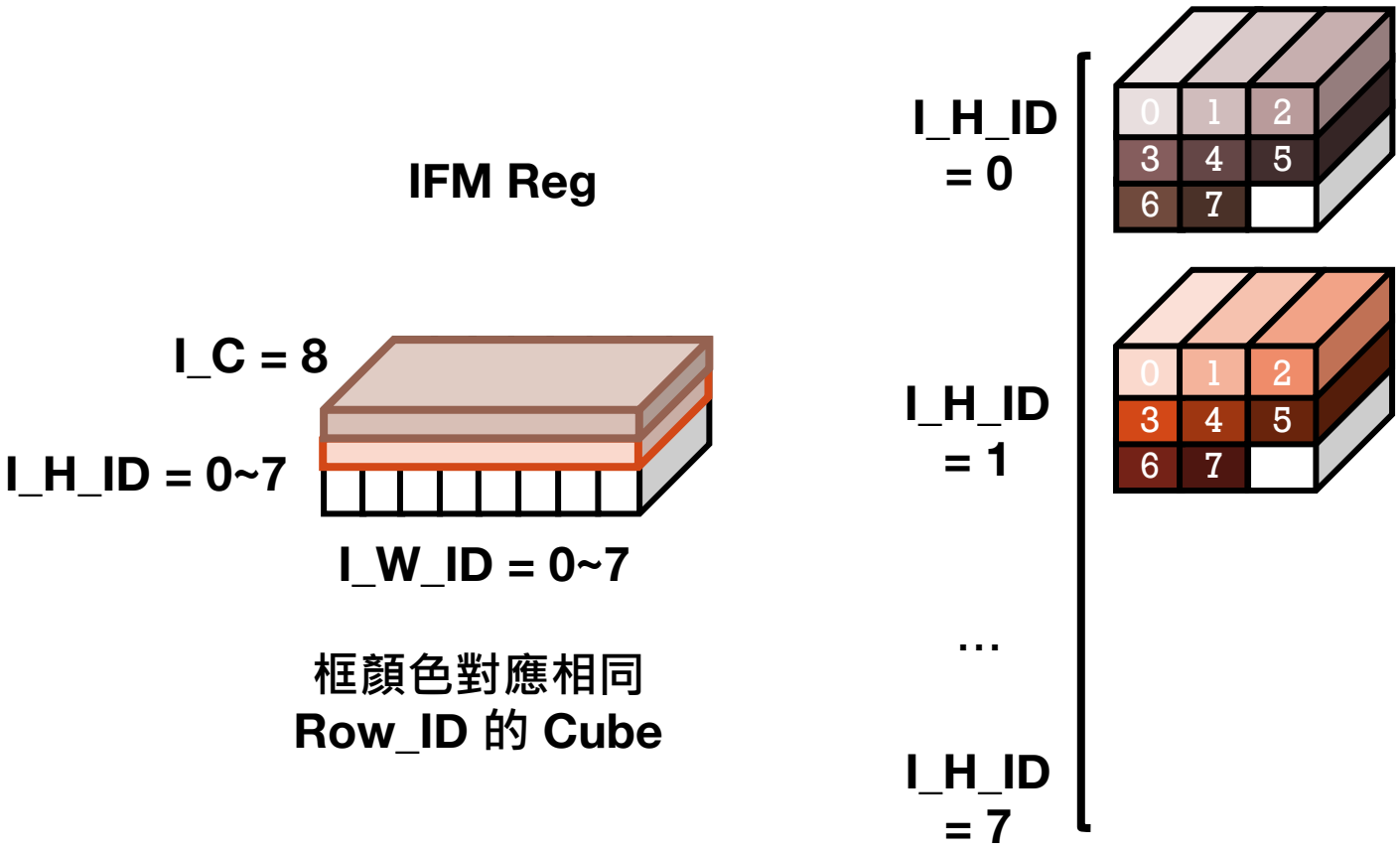
…

I_H_ID = 7

I_H_ID = 0~ 7
會對應同一組 Weight

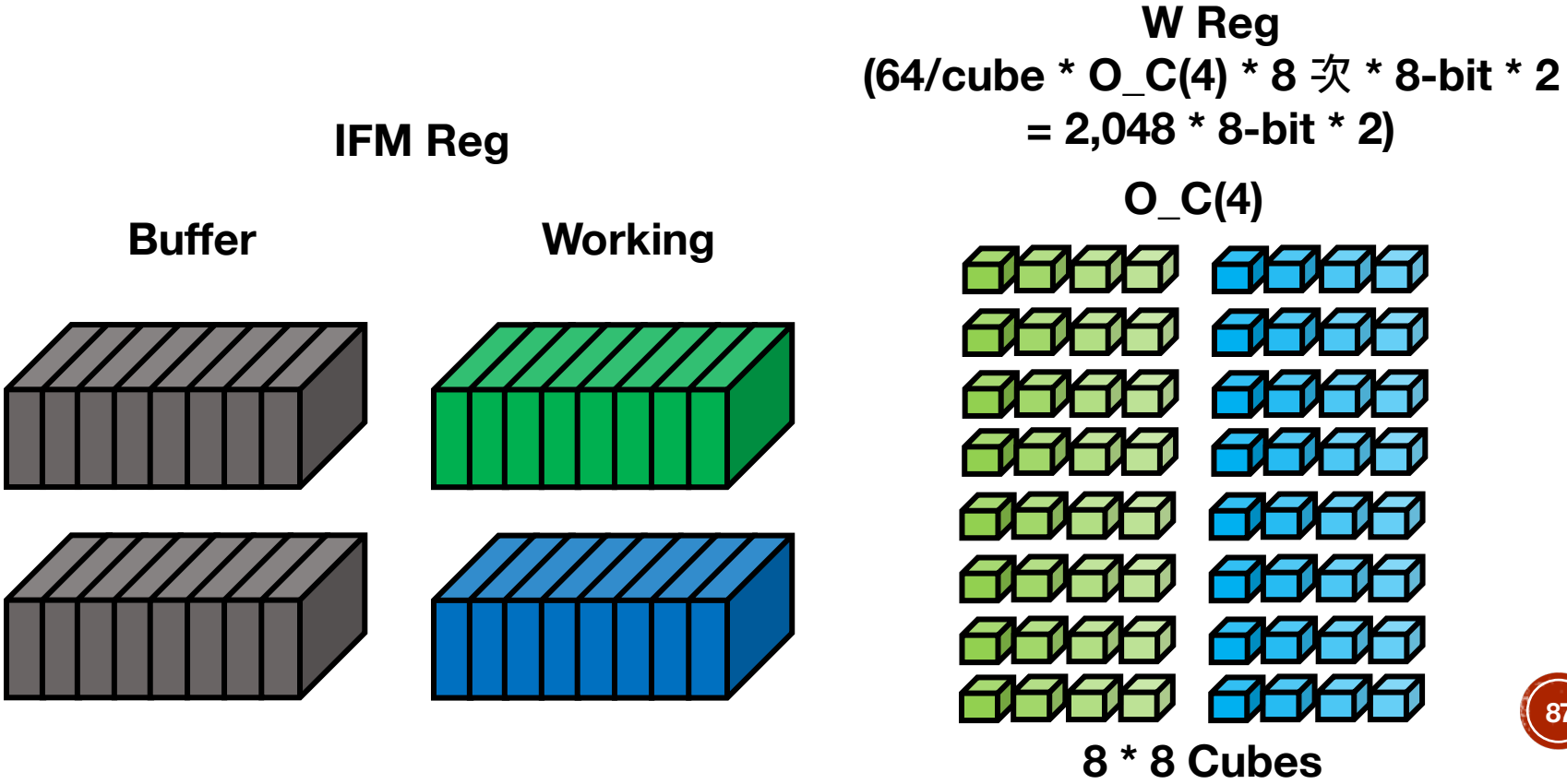
Conv 1x1 - Data Mapping

再從 O_C(1) * O_H(8) Cubes 來看其分配 ·

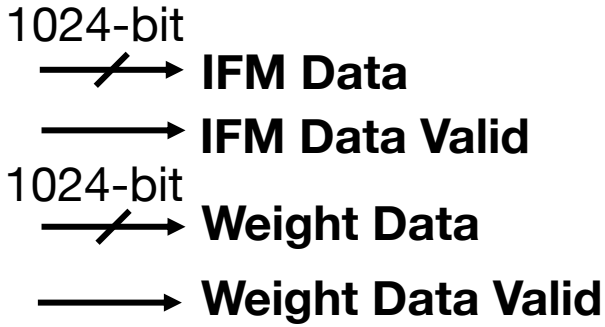


Conv 1x1 – 平衡資料進出

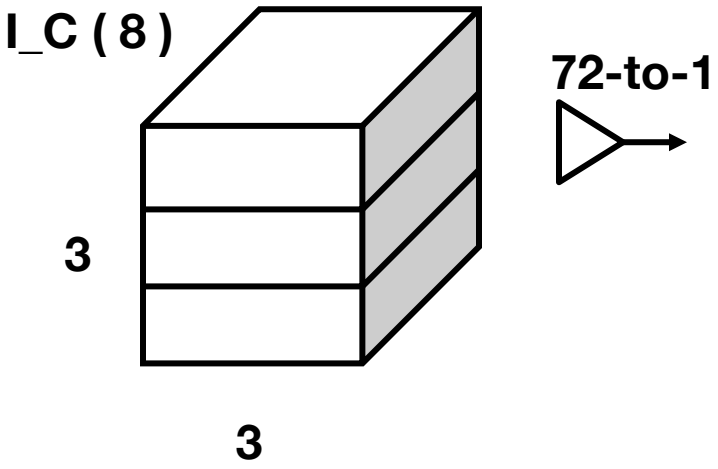
Cube 對應 8 * 8 = 64 個 weight，原先實作的 (5*5*8)*8，如果希望 O_C 維持在 8，則只能存放 3 * O_C (8)，因 IFM Reg 進來時需 8 cycles 才可以填滿，為了有效利用此資料，資料應使用 8 次以上才替換，這邊為了滿足上述需求，增加其 Reg 資源以平衡時間．

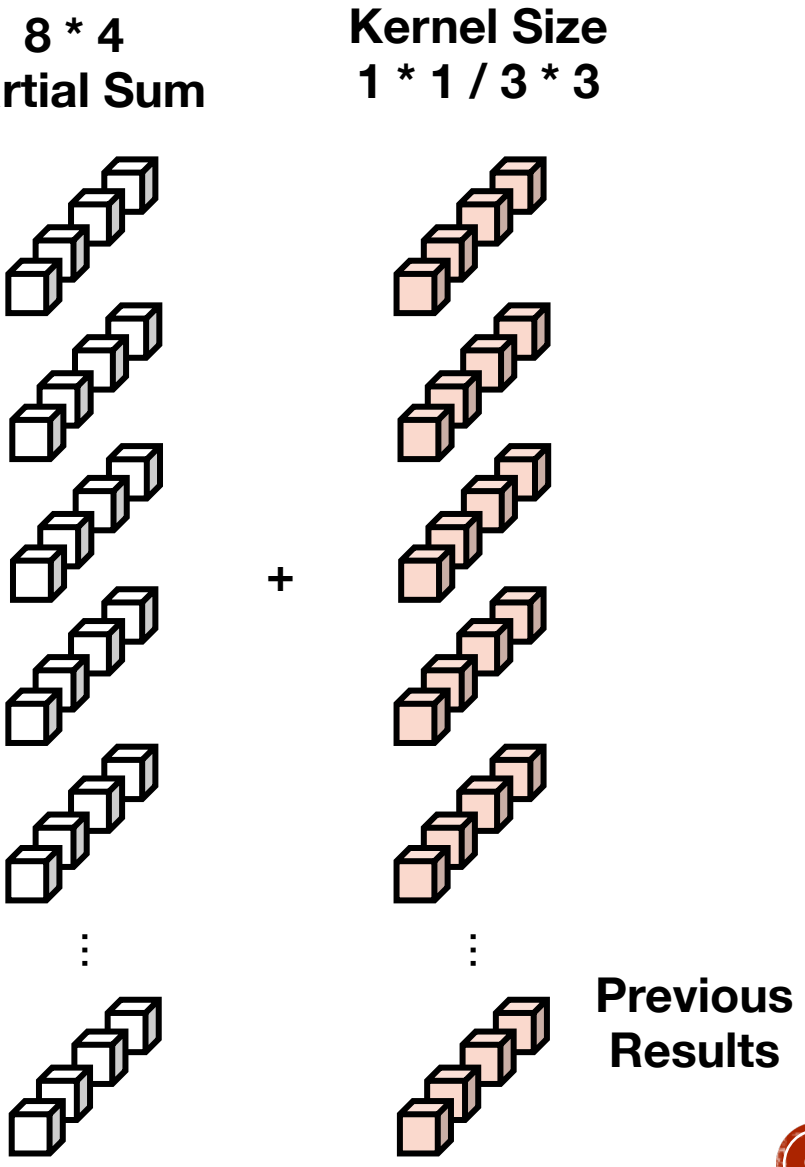


Conv 1x1 – Reg 部分需切開

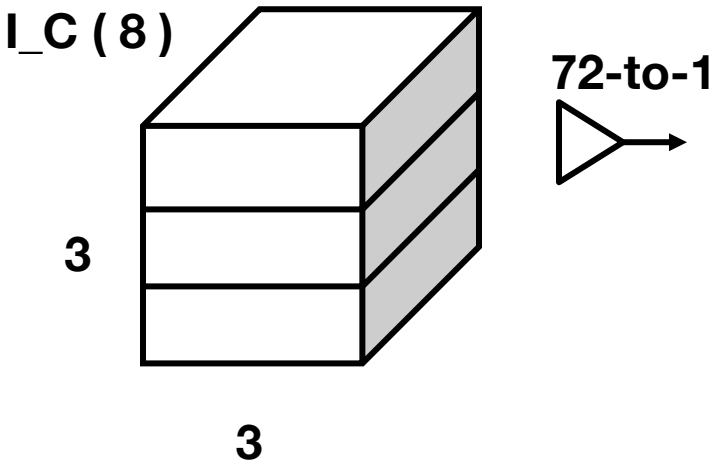


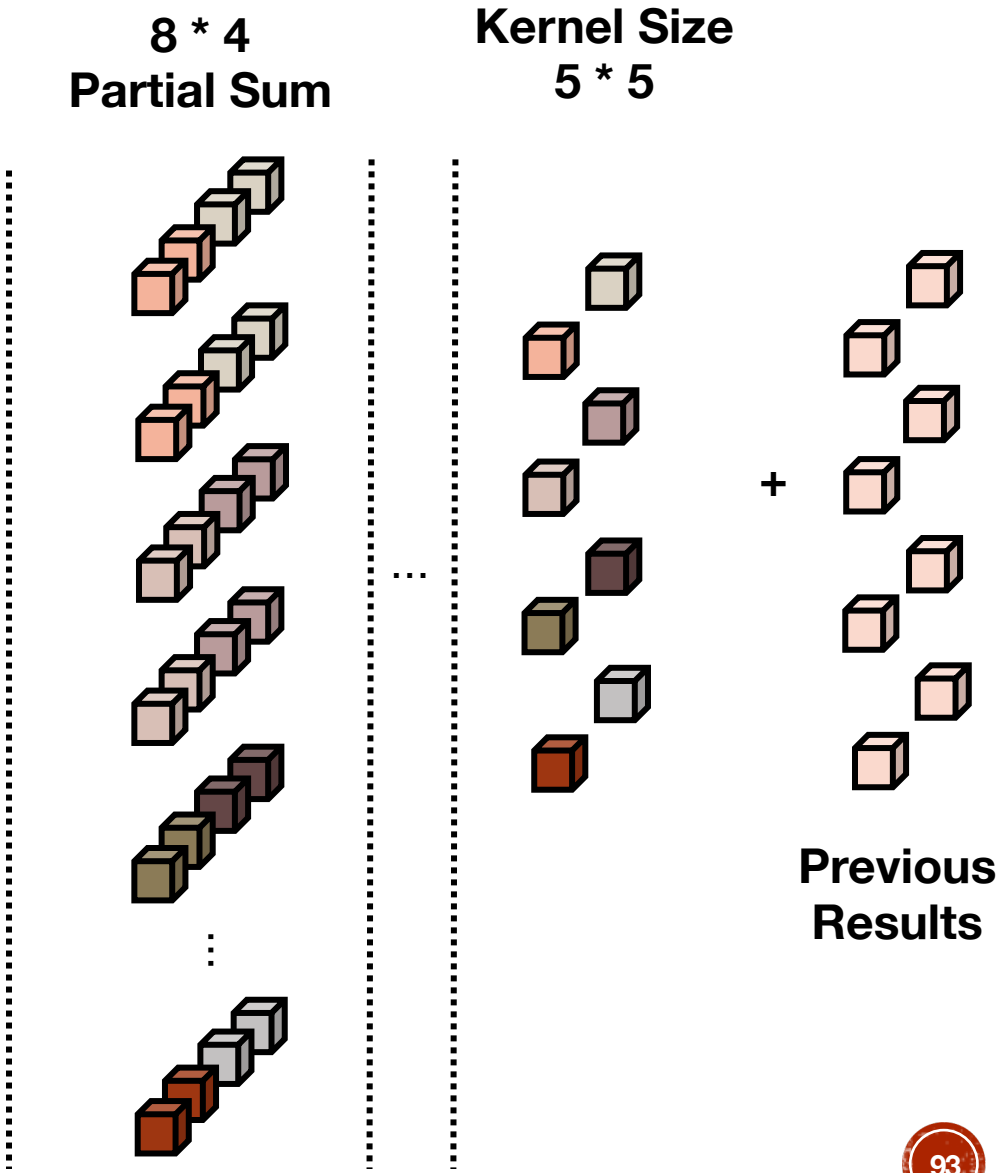
Adder Tree



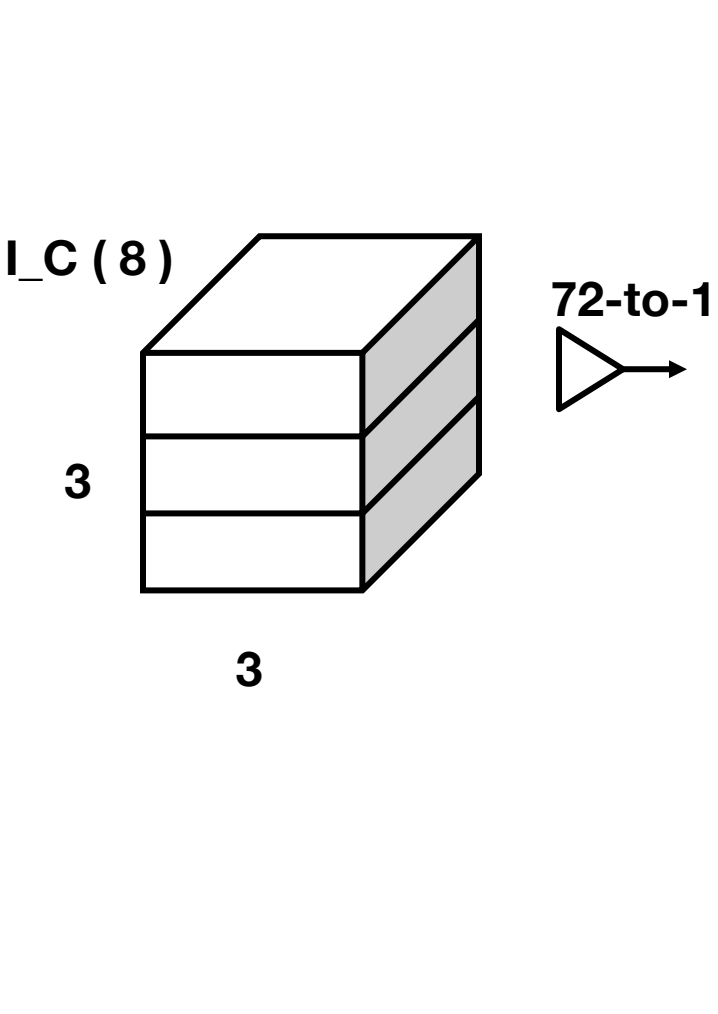


Adder Tree

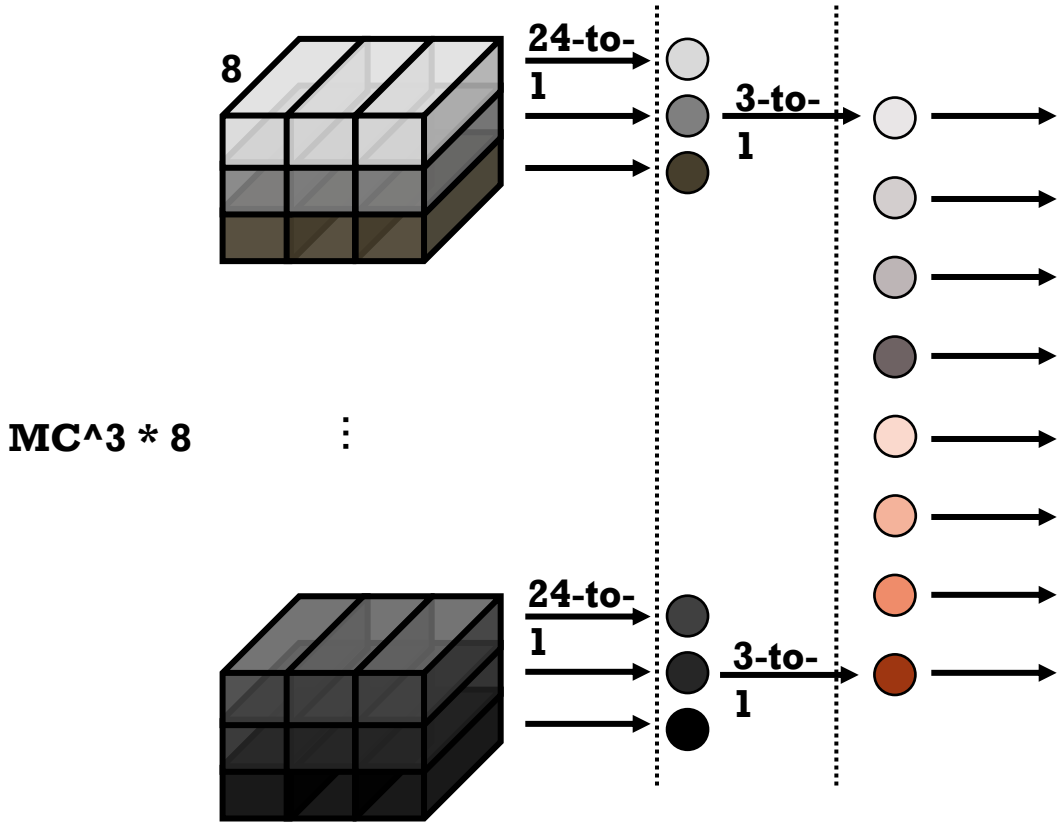




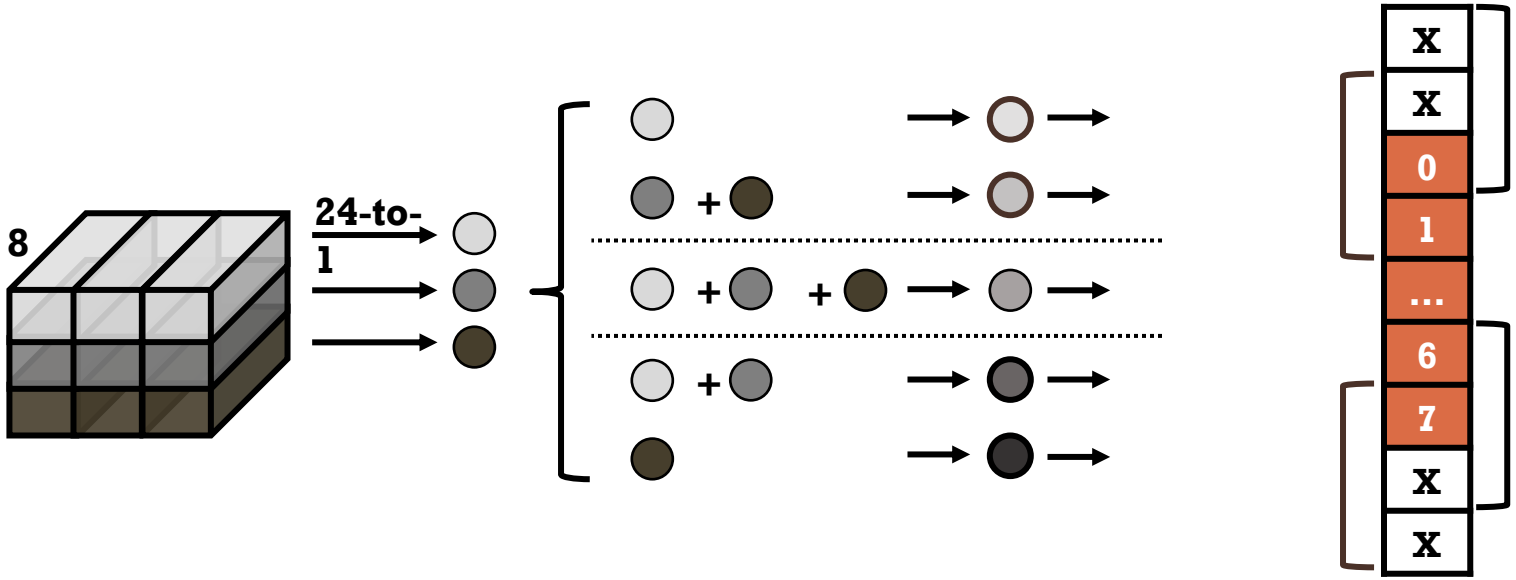
Adder Tree



Adder Tree – 1 x 1



Adder Tree – 3 x 3, 5 x 5, 7 x 7



Project Schedule