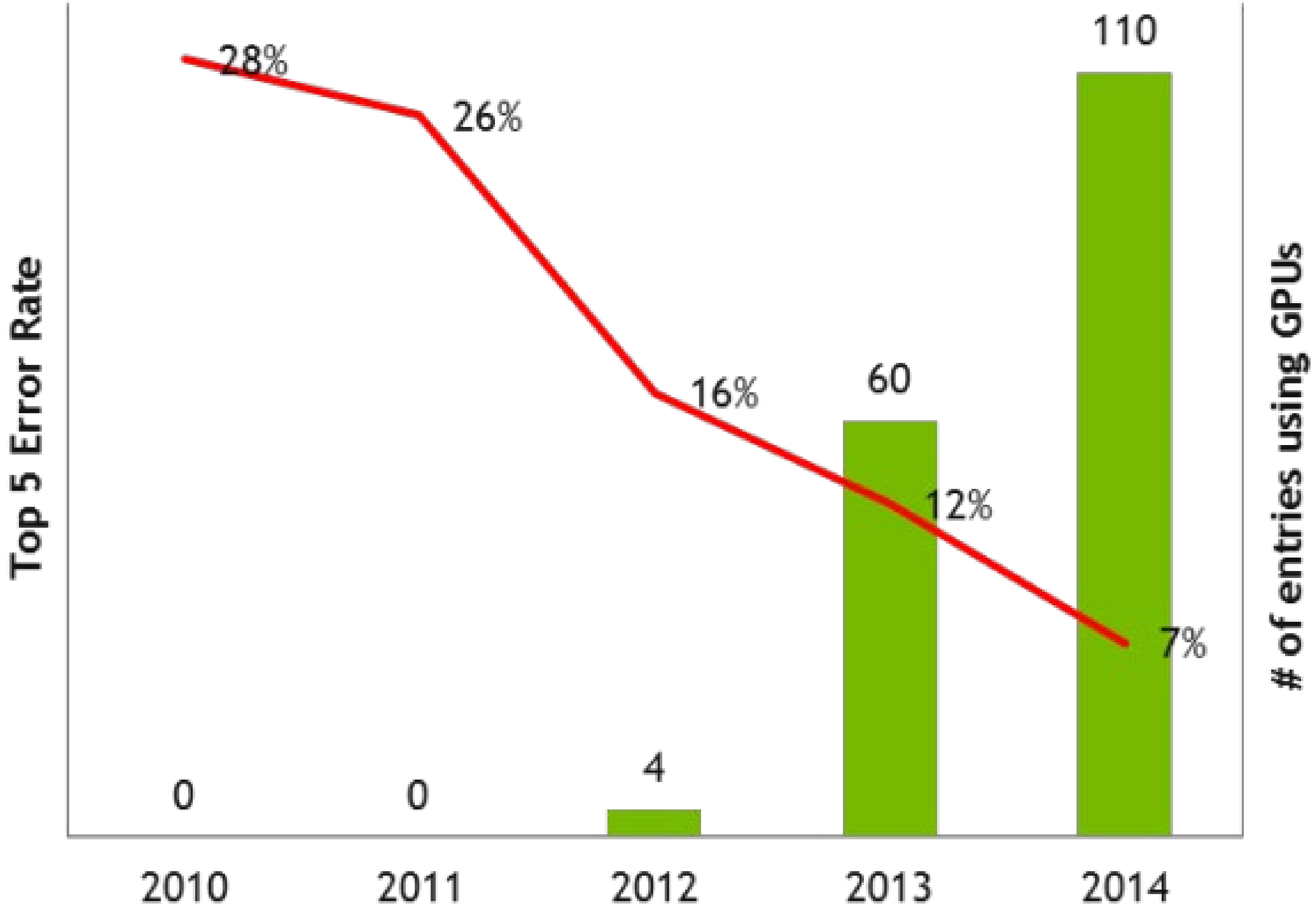
Accelerators (I)

Joel Emer

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

> “Compute has been the oxygen of deep learning”
> – Ilya Sutskever (Open AI)

GPU Usage for ImageNet Challenge



Challenges

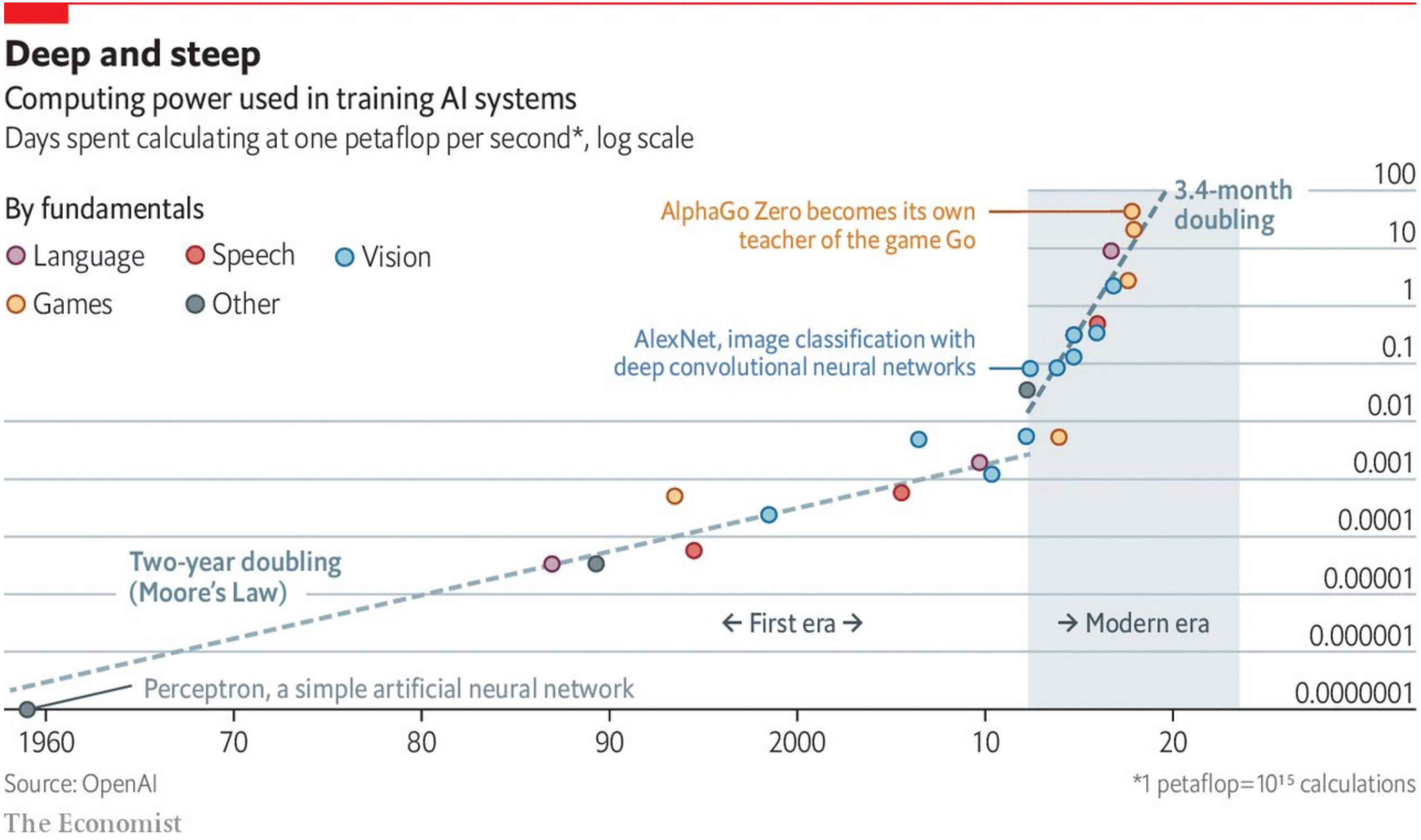
From EE Times – September 27, 2016

“Today the job of training machine learning models is limited by compute, if we had faster processors we’d run bigger models…in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater.”

– Greg Diamos, Senior Researcher, SVAIL, Baidu

Compute Demands Growing Exponentially

AlexNet to AlphaGo Zero: A 300,000x Increase in Compute



Deep and steep

Computing power used in training AI systems

Days spent calculating at one petaflop per second*, log scale

AlphaGo Zero becomes its own teacher of the game Go

AlexNet, image classification with deep convolutional neural networks

Perceptron, a simple artificial neural network

Two-year doubling (Moore's Law)

3.4-month doubling

Source: OpenAI

*1 petaflop=10^{15} calculations

The Economist

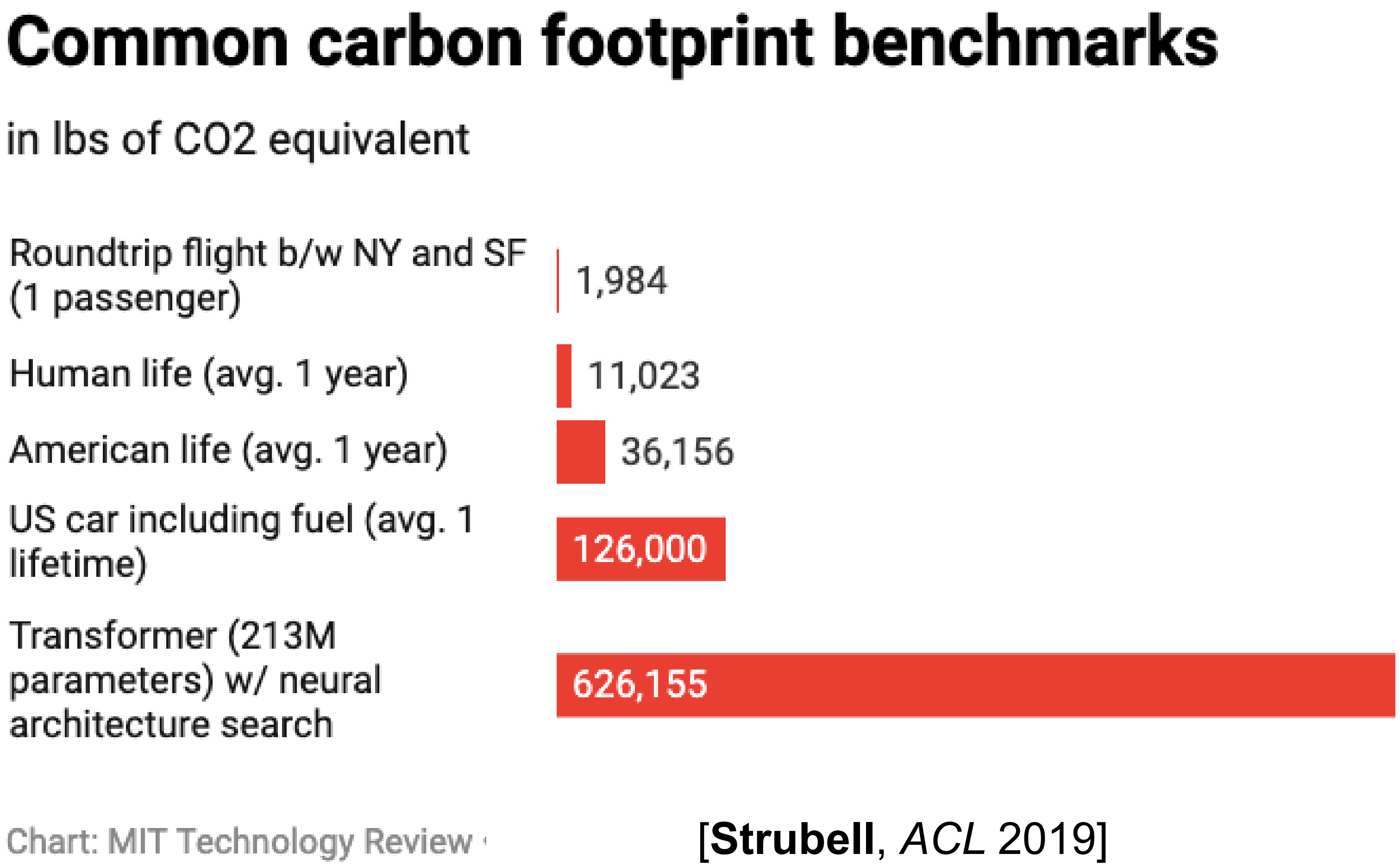
Source: https://www.economist.com/technology-quarterly/2020/06/11/the-cost-of-training-machines-is-becoming-a-problem

What is Moore’s Law

- every two years*
- CPU performance will double every two years*
- Chip performance will double every two years*
- The speed of transistors will double every two years*
- Transistors will shrink to half size every two years*
- Transistors per die will double every two years*
- The economic sweet spot for the number of devices on a chip will double every two years*

* Or 18 months…

Compute Demands for Deep Neural Networks



Common carbon footprint benchmarks

in lbs of CO2 equivalent

Chart: MIT Technology Review

[**Strubell**, *ACL* 2019]

Energy and Power Consumption

- **Energy Consumption** = α x C x V^2

Switching activity factor (between 0 to 1)

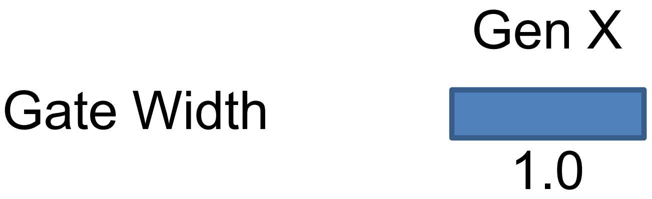
Capacitance

Voltage

- **Power Consumption** = α x C x V^2 x f

Frequency

Dennard Scaling (idealized)



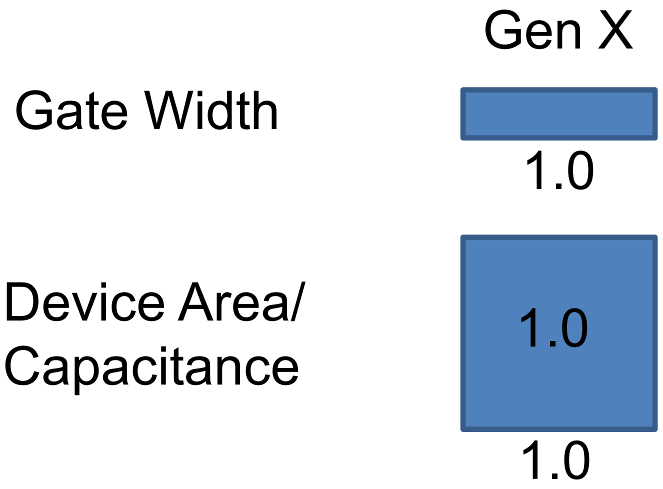
Gen X

Gate Width

1.0

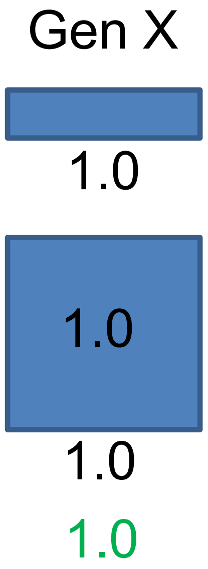
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



Gen X

Gate Width

1.0

Device Area/
Capacitance

1.0

1.0

Voltage

1.0

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

Gen X

Gate Width

1.0

Device Area/
Capacitance

1.0

Voltage 1.0

Energy $\sim \underline{1.0 \text{ x } 1.0^2} = 1.0$

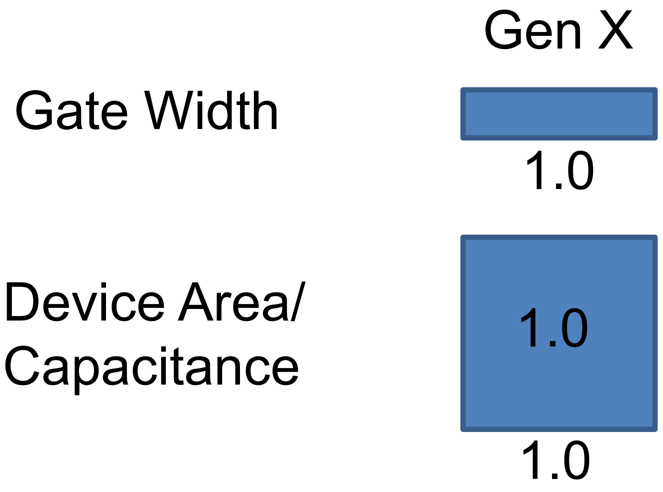
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

	Gen X
Gate Width	1.0
Device Area/ Capacitance	1.0 1.0
Voltage	1.0
Energy	$\sim \underline{1.0\ \text{x}\ 1.0^{2}} = 1.0$
Delay	1.0

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



Voltage 1.0

Energy ~ $\underline{1.0 \times 1.0^2}$ = 1.0

Delay 1.0

Frequency 1/1.0 = 1.0

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

Gen X

Gate Width

Device Area/
Capacitance

	Gen X
Voltage	1.0
Energy	$\sim \underline{1.0 \times 1.0^2} = 1.0$
Delay	1.0
Frequency	$1/1.0 = 1.0$
Power	$\sim \underline{1.0 \times 1.0^2} \times 1.0 = 1.0$

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

	Gen X	Gen X+1
Gate Width	 1.0	
Device Area/ Capacitance	 1.0 1.0	
Voltage	1.0	
Energy	$\sim \underline{1.0 \text{ x } 1.0^2} = 1.0$	
Delay	1.0	
Frequency	1/1.0 = 1.0	
Power	$\sim \underline{1.0 \text{ x } 1.0^2} \text{ x } 1.0 = 1.0$	

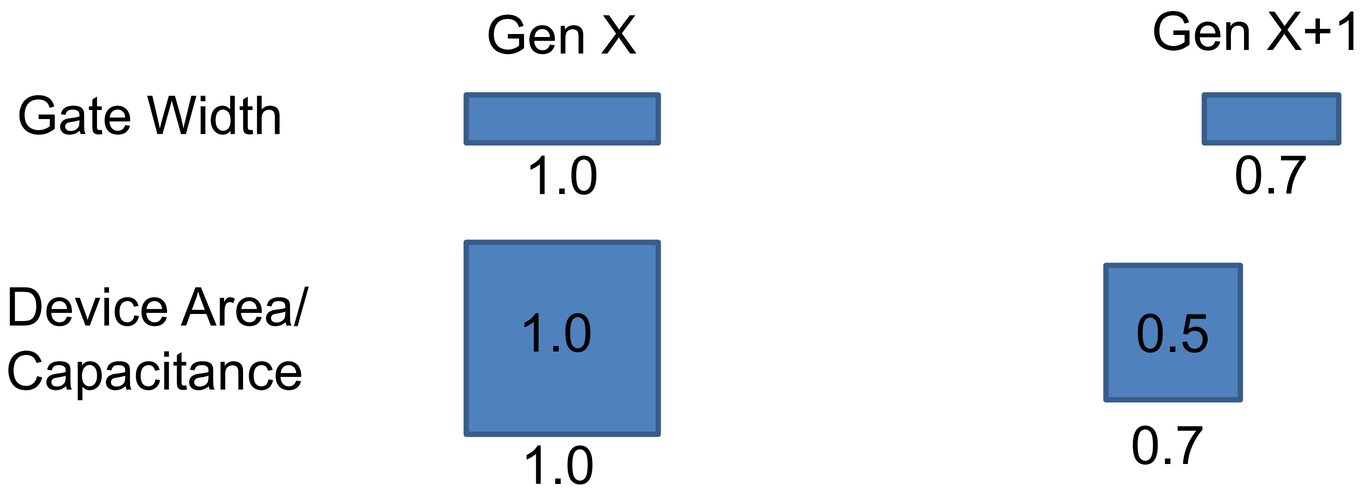
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

	Gen X	Gen X+1
Gate Width	▬ 1.0	▬ 0.7
Device Area/ Capacitance	■ 1.0 1.0	
Voltage	1.0	
Energy	~ $\underline{1.0 \times 1.0^2}$ = 1.0	
Delay	1.0	
Frequency	1/1.0 = 1.0	
Power	~ $\underline{1.0 \times 1.0^2}$ x 1.0 = 1.0	

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



Voltage 1.0

Energy $\sim \underline{1.0 \times 1.0^2} = 1.0$

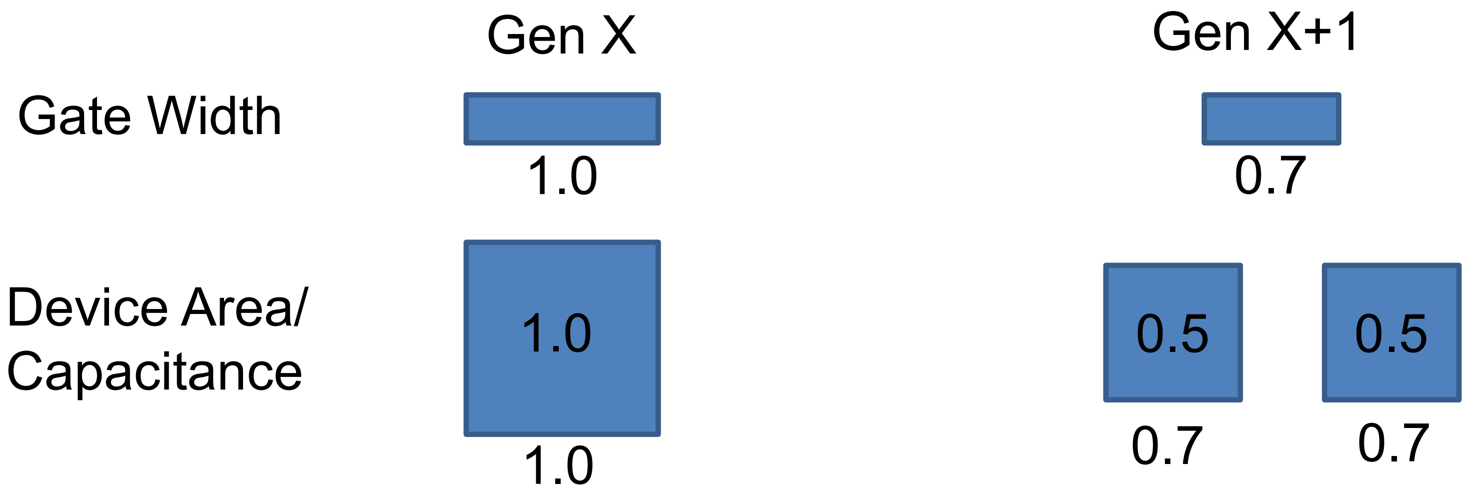
Delay 1.0

Frequency $1/1.0 = 1.0$

Power $\sim \underline{1.0 \times 1.0^2} \times 1.0 = 1.0$

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



Voltage	1.0
Energy	$\sim \underline{1.0 \times 1.0^2} = 1.0$
Delay	1.0
Frequency	$1/1.0 = 1.0$
Power	$\sim \underline{1.0 \times 1.0^2} \times 1.0 = 1.0$

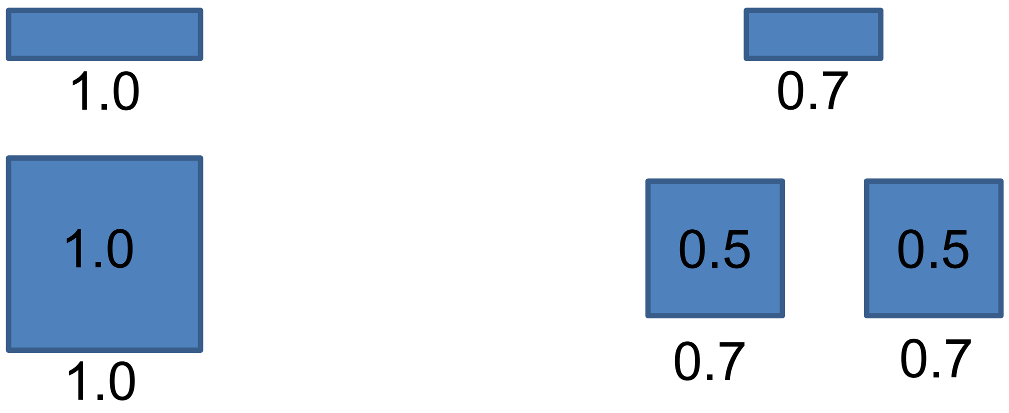
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)

Gen X　　　　Gen X+1

Gate Width

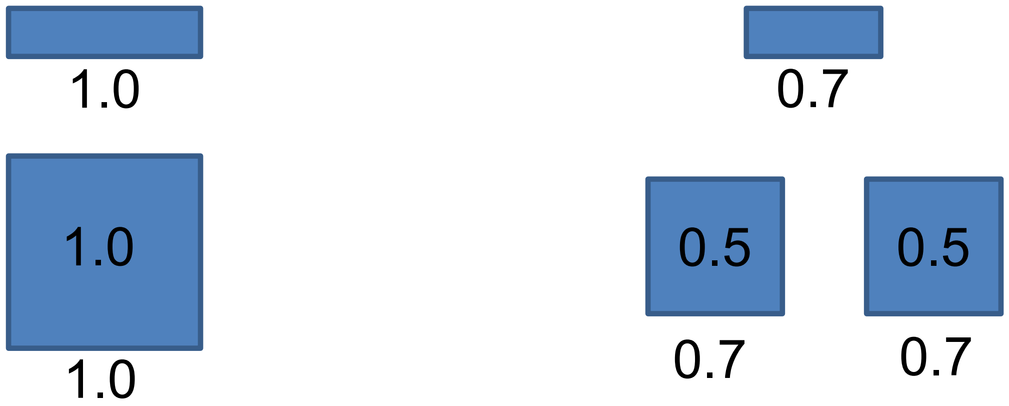
Device Area/
Capacitance



	Gen X	Gen X+1
Voltage	1.0	0.7
Energy	$\sim \underline{1.0\ x\ 1.0^2} = 1.0$	
Delay	1.0	
Frequency	1/1.0 = 1.0	
Power	$\sim \underline{1.0\ x\ 1.0^2}\ x\ 1.0 = 1.0$	

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

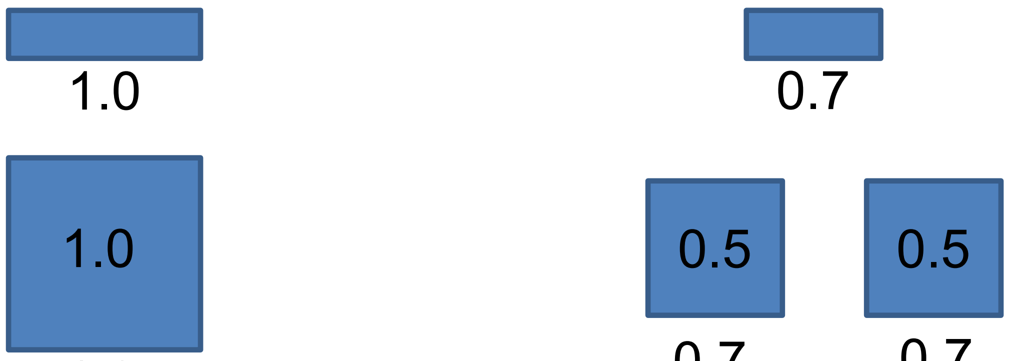
Dennard Scaling (idealized)



	Gen X	Gen X+1
Gate Width	1.0	0.7
Device Area/ Capacitance	1.0 1.0	0.5 0.5 0.7 0.7
Voltage	1.0	0.7
Energy	~ $\underline{1.0 \times 1.0^2} = 1.0$	~ $\underline{2 \times 0.7 \times 0.7^2} = 0.65$
Delay	1.0	
Frequency	1/1.0 = 1.0	
Power	~ $\underline{1.0 \times 1.0^2} \times 1.0 = 1.0$	

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

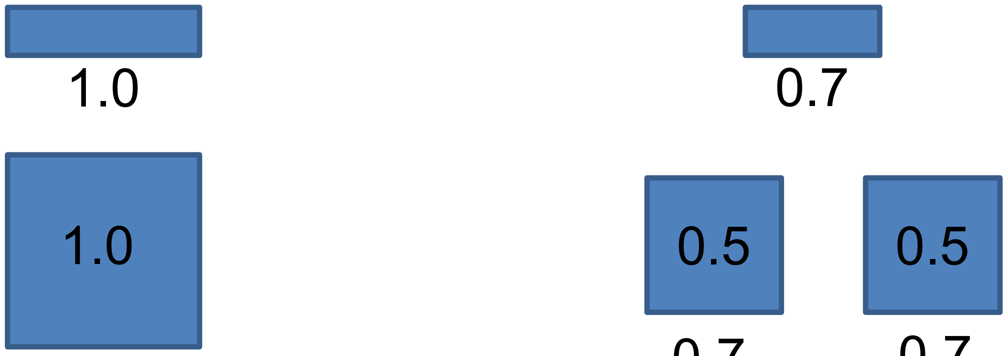
Dennard Scaling (idealized)



	Gen X	Gen X+1
Gate Width	1.0	0.7
Device Area/ Capacitance	1.0 1.0	0.5 0.5 0.7 0.7
Voltage	1.0	0.7
Energy	$\sim \underline{1.0 \times 1.0^2} = 1.0$	$\sim \underline{2 \times 0.7 \times 0.7^2} = 0.65$
Delay	1.0	0.7
Frequency	1/1.0 = 1.0	
Power	$\sim \underline{1.0 \times 1.0^2} \times 1.0 = 1.0$	

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

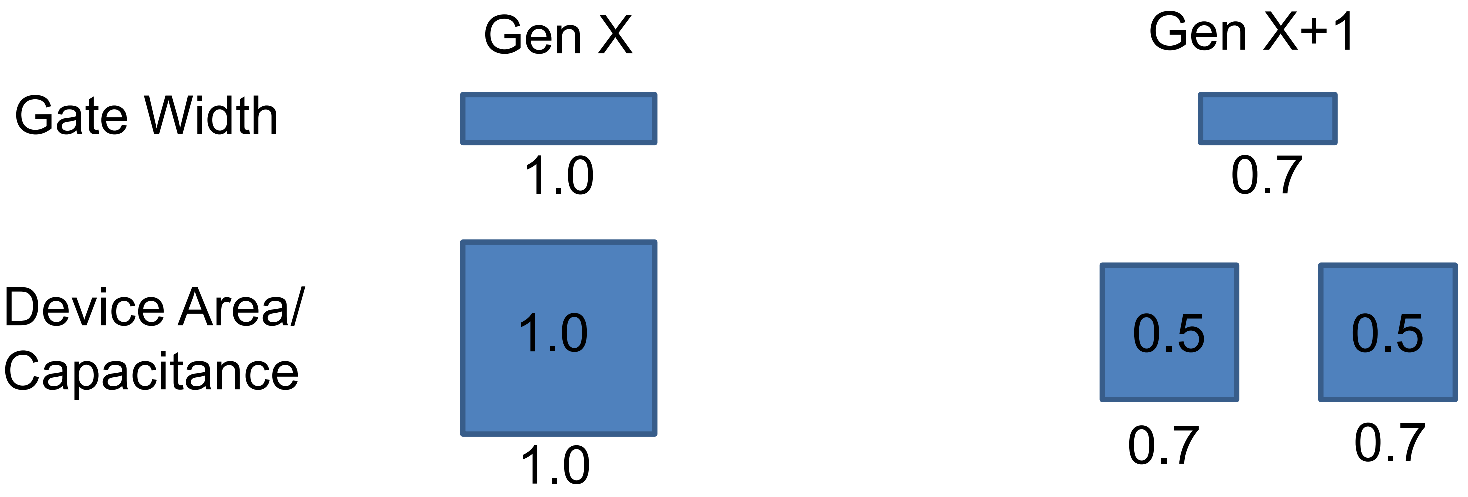
Dennard Scaling (idealized)



	Gen X	Gen X+1
Gate Width	1.0	0.7
Device Area/ Capacitance	1.0 1.0	0.5 0.5 0.7 0.7
Voltage	1.0	0.7
Energy	~ $\underline{1.0 \times 1.0^2}$ = 1.0	~ $\underline{2 \times 0.7 \times 0.7^2}$ = 0.65
Delay	1.0	0.7
Frequency	1/1.0 = 1.0	1/0.7 = 1.4
Power	~ $\underline{1.0 \times 1.0^2} \times 1.0$ = 1.0	

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Dennard Scaling (idealized)



	Gen X	Gen X+1
Gate Width	1.0	0.7
Device Area/ Capacitance	1.0	0.5 0.5
Voltage	1.0	0.7
Energy	$\sim \underline{1.0 \text{ x } 1.0^2} = 1.0$	$\sim \underline{2 \text{ x } 0.7 \text{ x } 0.7^2} = 0.65$
Delay	1.0	0.7
Frequency	$1/1.0 = 1.0$	$1/0.7 = 1.4$
Power	$\sim \underline{1.0 \text{ x } 1.0^2} \text{ x } 1.0 = 1.0$	$\sim \underline{2 \text{ x } 0.7 \text{ x } 0.7^2} \text{ x } 1.4 = 1.0$

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions“, JSSC 1974]

Technology Trends

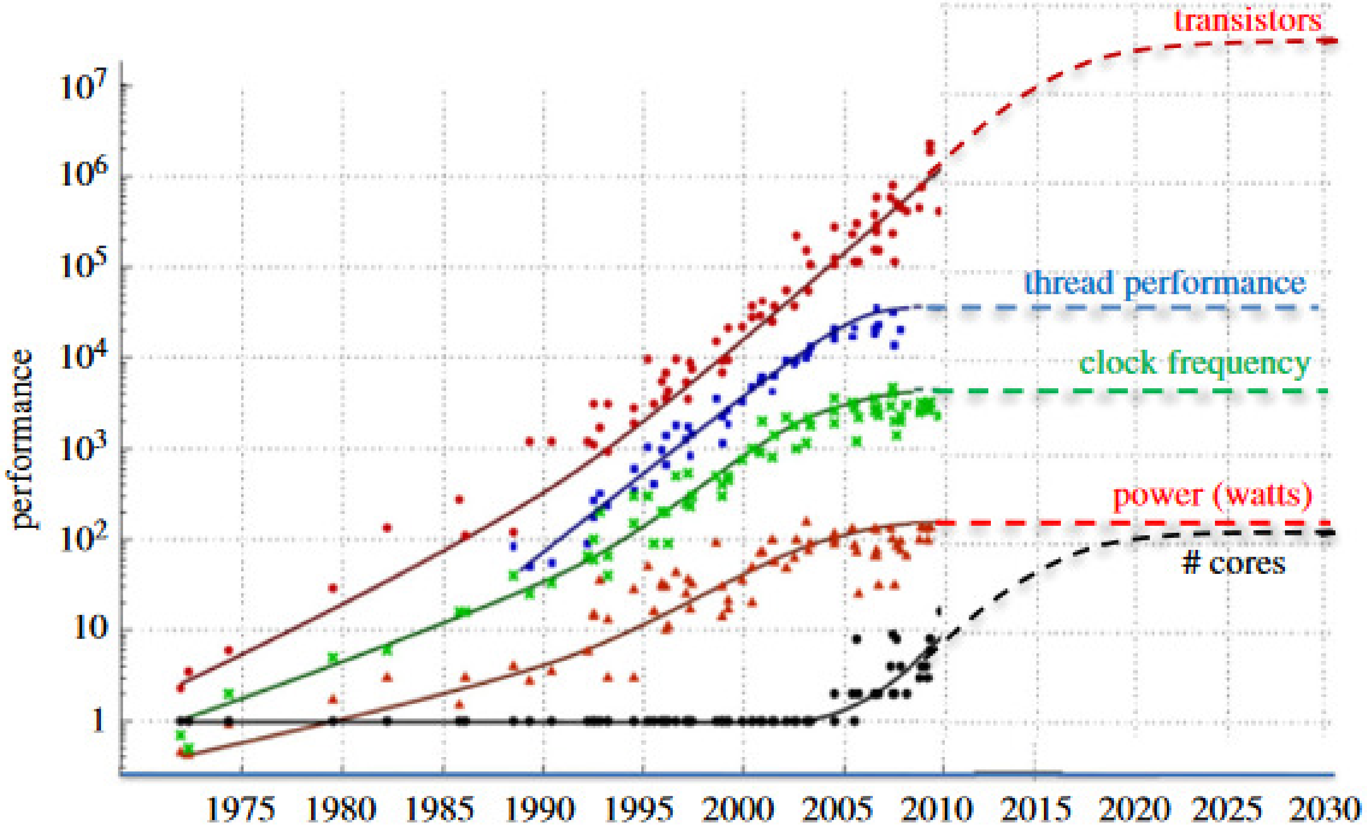
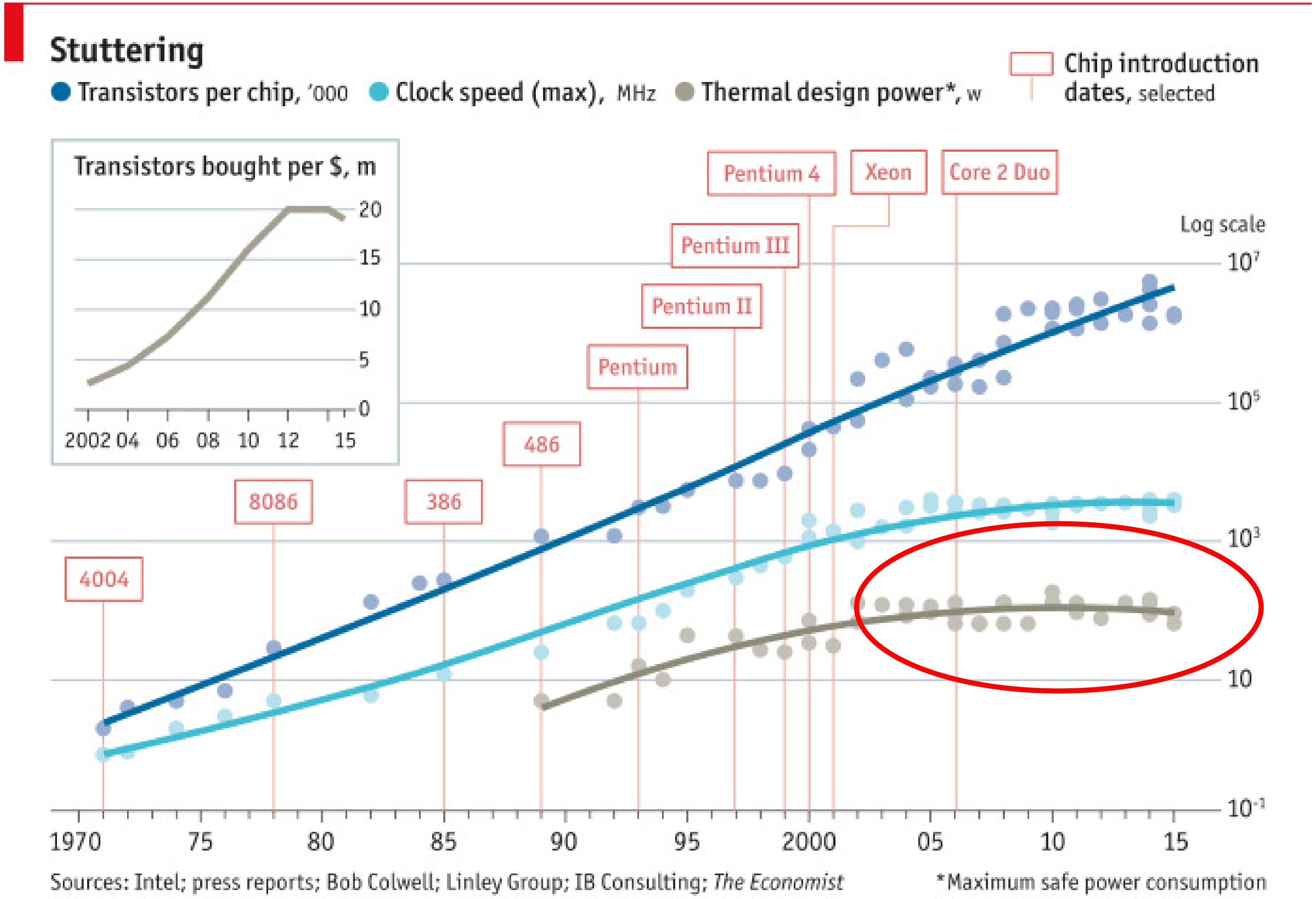


Figure 2. Sources of computing performance have been challenged by the end of Dennard scaling in 2004. All additional approaches to further performance improvements end in approximately 2025 due to the end of the roadmap for improvements to semiconductor lithography. Figure from Kunle Olukotun, Lance Hammond, Herb Sutter, Mark Horowitz and extended by John Shalf. (Online version in colour.)

During the Moore + Dennard's Law Era

- Instruction-level parallelism (ILP) was largely mined out by early 2000s
- Voltage (Dennard) scaling ended in 2005
- Hit the power limit wall in 2005
- Performance is coming from parallelism using more transistors since ~2007
- But….

Technology Trends



Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; *The Economist*

*Maximum safe power consumption

Architecture Metrics

- Speed – The rate at which the hardware finishes tasks. Limited by the number of computation units and their utilization.

- Energy – The total energy, e.g., in Joules, consumed to perform a task. Often constrained by battery capacity or desire to reduce carbon footprint.

- Power – The rate at which energy is consumed, e.g., in Watts. Often limited by delivery or packaging constraints

- Accuracy – The precision of the results produced. Can be dictated by bit width of compute units.

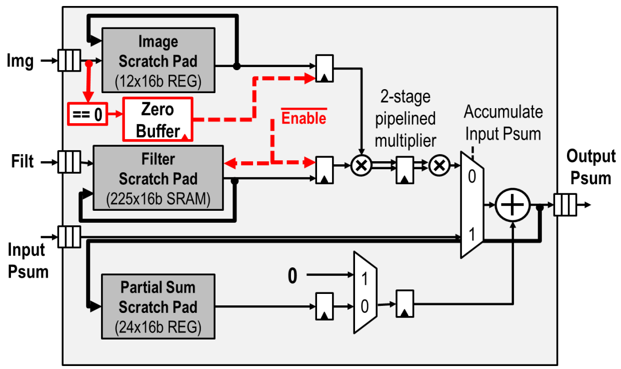
- Flexibility – The range of problems that can be solved, which is constrained by the limitations of the architecture.

Deep Learning Platforms

- CPU
  - Intel, ARM, AMD…
- GPU
  - NVIDIA, AMD…
- Fine Grained Reconfigurable (FPGA)
  - Xilinx, Altera (Microsoft BrainWave)
- Coarse Grained Programmable/Reconfigurable
  - Wave Computing, Graphcore, Samba Nova…
- Application Specific
  - Neuflow, *DianNao, Eyeriss, TPU, Cnvlutin, SCNN, …

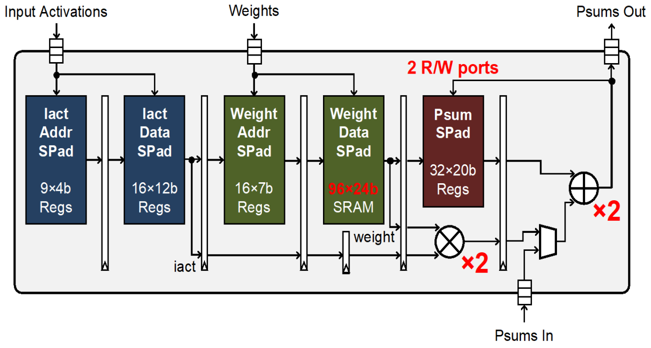
There Are Myriad Tensor Accelerators

Selected tensor accelerator designs

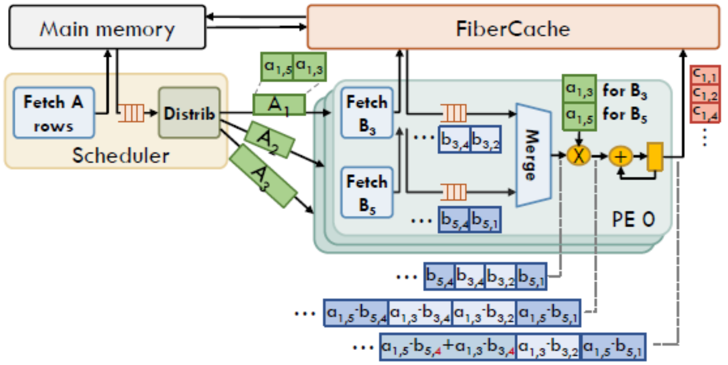


Img
Image Scratch Pad (12x16b REG)
== 0
Zero Buffer
Enable
Filt
Filter Scratch Pad (225x16b SRAM)
Input Psum
2-stage pipelined multiplier
Accumulate Input Psum
Output Psum
Partial Sum Scratch Pad (24x16b REG)
0
1

Eyeriss [JSSC2017]

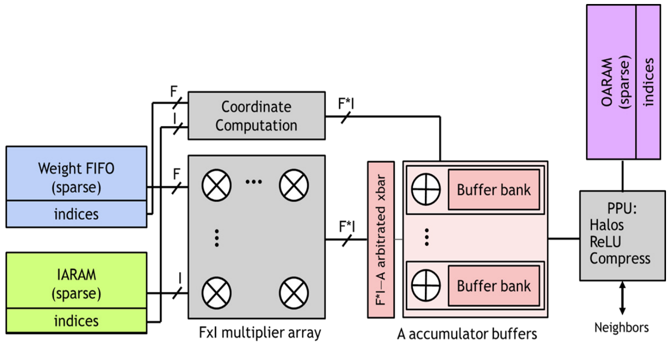


Eyeriss V2 [JETCAS2019]

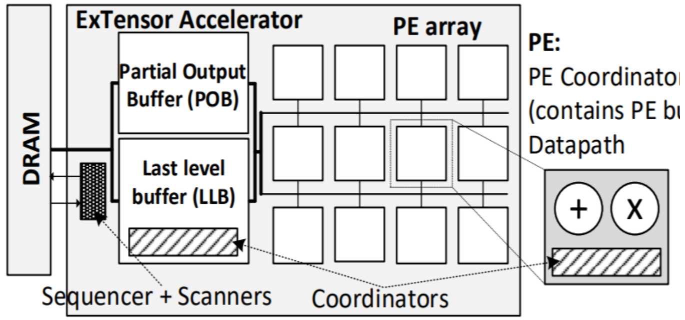


Main memory
FiberCache
Fetch A rows
Distrib
Scheduler
$a_{1,5}$ $a_{1,3}$
A_1
A_2
A_3
Fetch B_3
Fetch B_5
$b_{3,4}$ $b_{3,2}$
$b_{5,4}$ $b_{5,1}$
Merge
$a_{1,3}$ for B_3
$a_{1,5}$ for B_5
PE 0
$c_{1,1}$ $c_{1,2}$ $c_{1,4}$
$b_{5,4}$ $b_{3,4}$ $b_{3,2}$ $b_{5,1}$
$a_{1,5} \cdot b_{5,4}$ $a_{1,3} \cdot b_{3,4}$ $a_{1,3} \cdot b_{3,2}$ $a_{1,5} \cdot b_{5,1}$
$a_{1,5} \cdot b_{5,4} + a_{1,3} \cdot b_{3,4}$ $a_{1,3} \cdot b_{3,2}$ $a_{1,5} \cdot b_{5,1}$

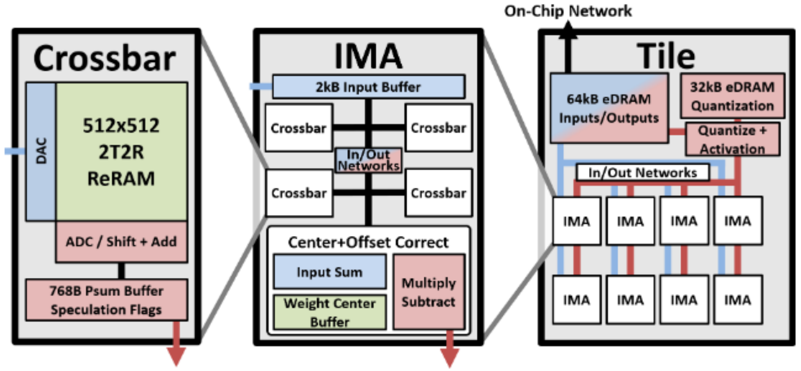
Gamma [ASPLOS2021]



SCNN [ISCA2017]

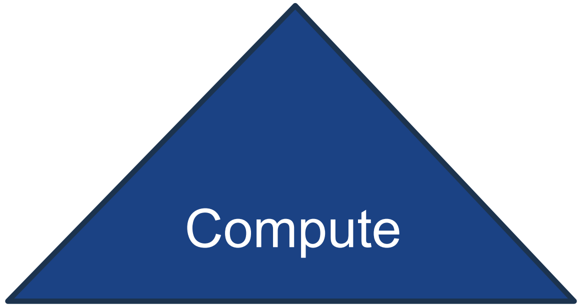


ExTensor [MICRO2019]



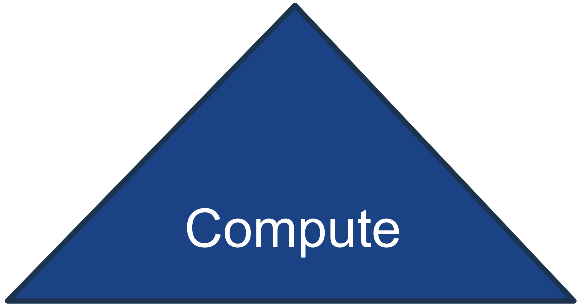
RAELLA [ISCA2023]

Separation of Concerns



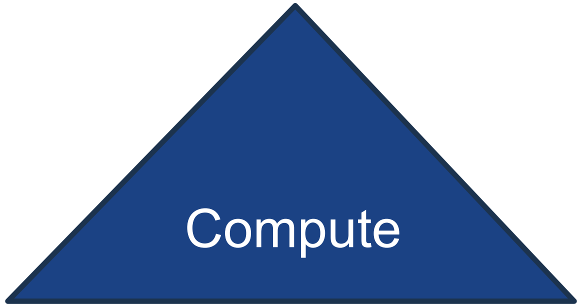
[TeAAL, Nayak et.a. MICRO 2023]

Separation of Concerns



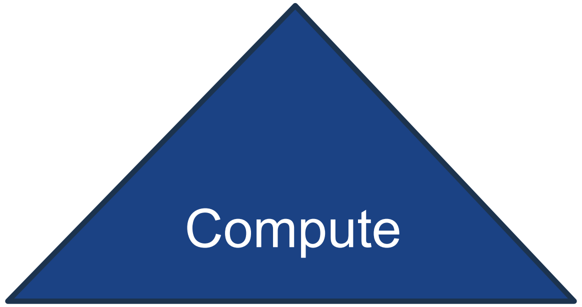
[TeAAL, Nayak et.a. MICRO 2023]

Separation of Concerns



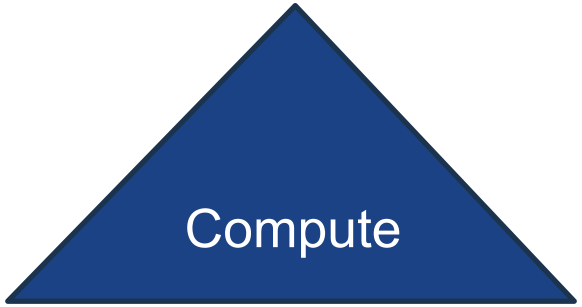
[TeAAL, Nayak et.a. MICRO 2023]

Separation of Concerns



[TeAAL, Nayak et.a. MICRO 2023]

Separation of Concerns



[TeAAL, Nayak et.a. MICRO 2023]

Separation of Concerns

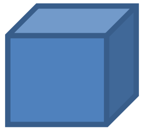
[TeAAL, Nayak et.a. MICRO 2023]

What Can Einsums Do For You?

- Simultaneously more precise and concise representation
- Extends tensor algebra far beyond matrix multiplication
- Now includes algorithms for: AI, graphs, fft, crypto, point cloud
- Intermediate point between GEMM and full programmability
- Allows for bounds analysis (SoL) on compute and data movement
- Admits of optimization via algebraic transformations
- Tip of the pyramid of separation of concerns

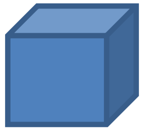
Tensors

Rank-0: Scalar

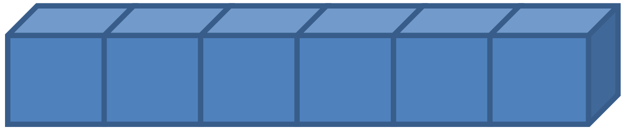


Tensors

Rank-0: Scalar

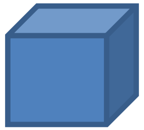


Rank-1: Vector

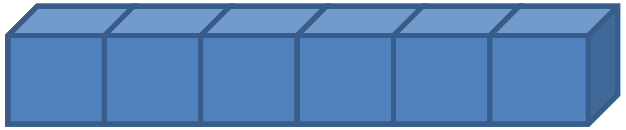


Tensors

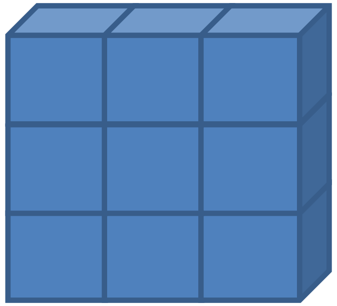
Rank-0: Scalar



Rank-1: Vector

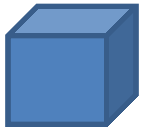


Rank-2: Matrix

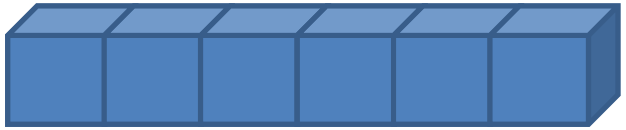


Tensors

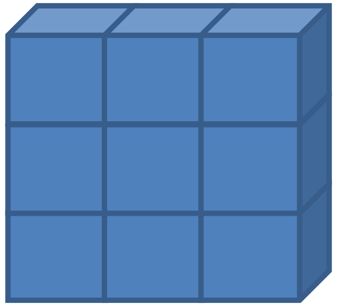
Rank-0: Scalar



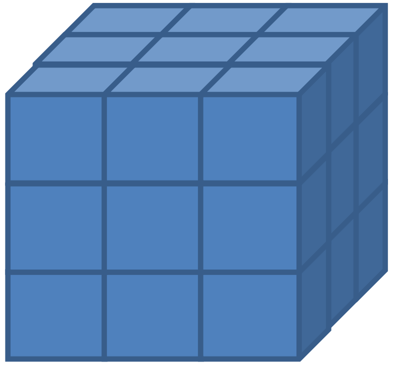
Rank-1: Vector



Rank-2: Matrix

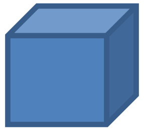


Rank-3: Cube

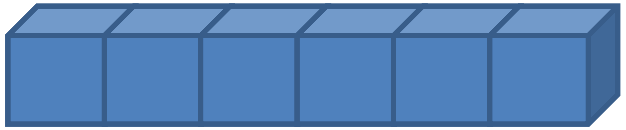


Tensors

Rank-0: Scalar

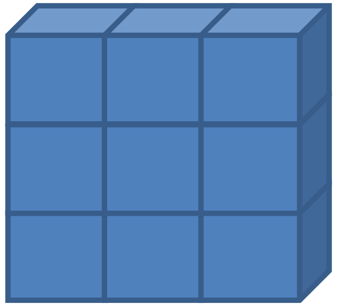


Rank-1: Vector

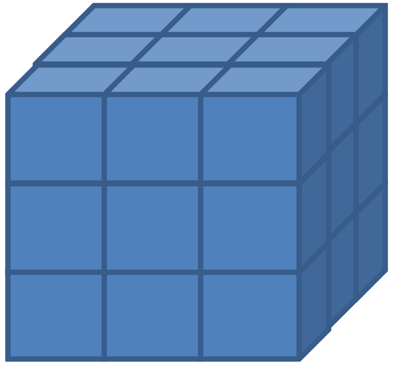


$$\boldsymbol{s}\boldsymbol{v}M\boldsymbol{T}$$

Rank-2: Matrix

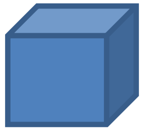


Rank-3: Cube

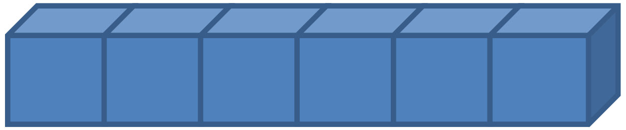


Tensors

Rank-0: Scalar

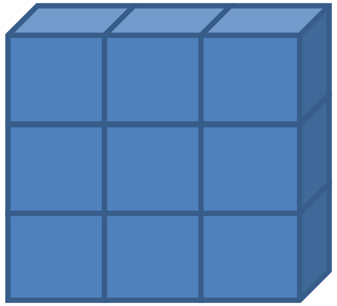


Rank-1: Vector

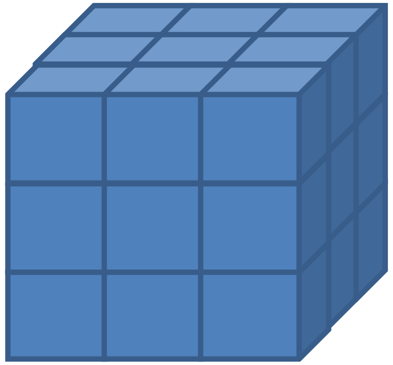


$$s\boldsymbol{v}M\boldsymbol{T}$$

Rank-2: Matrix

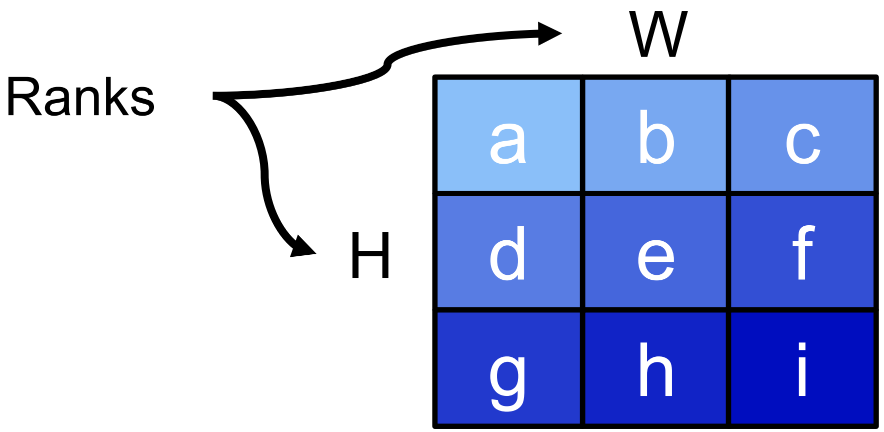


Rank-3: Cube



$$A_{i,j,k}$$

Tensor Data Terminology



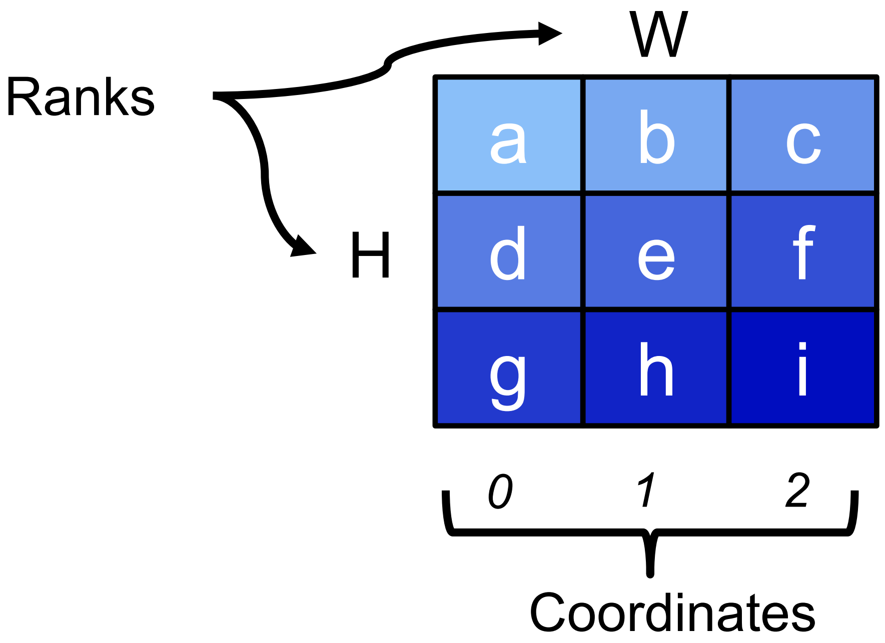
Ranks

W

H

a	b	c
d	e	f
g	h	i

Tensor Data Terminology



Ranks

W

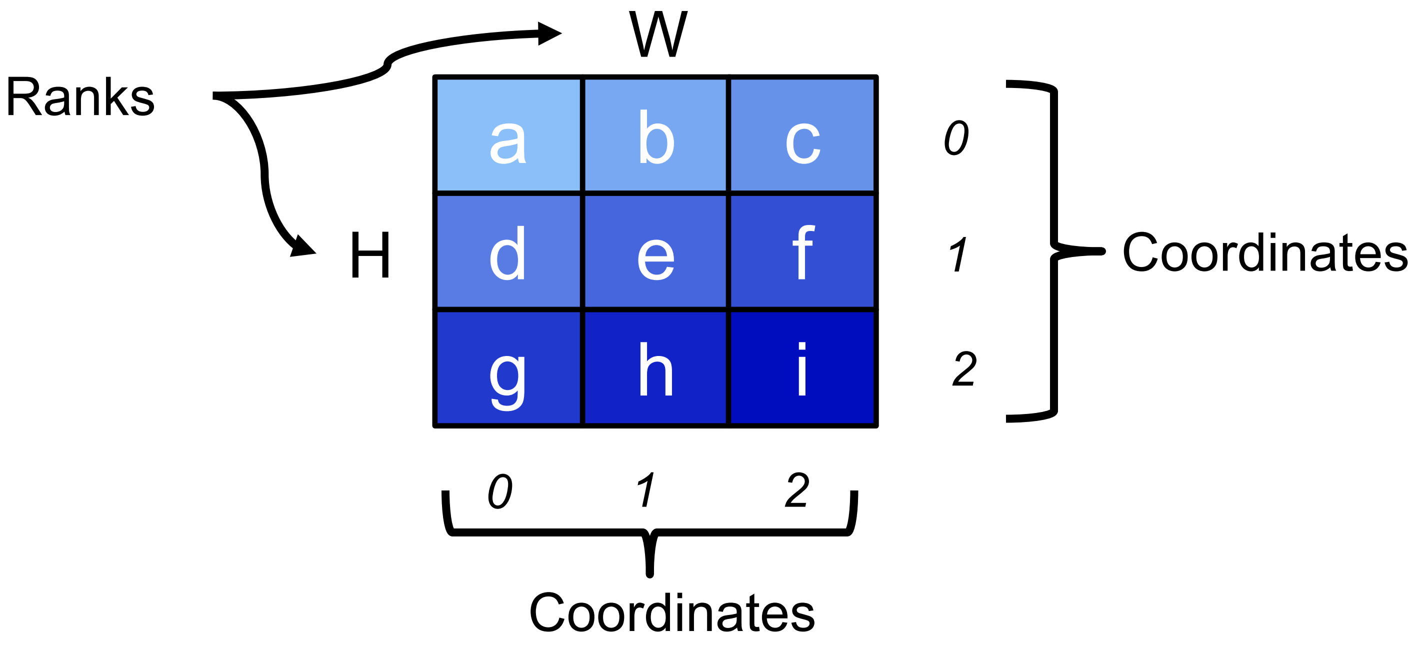
H

a	b	c
d	e	f
g	h	i

0 *1* *2*

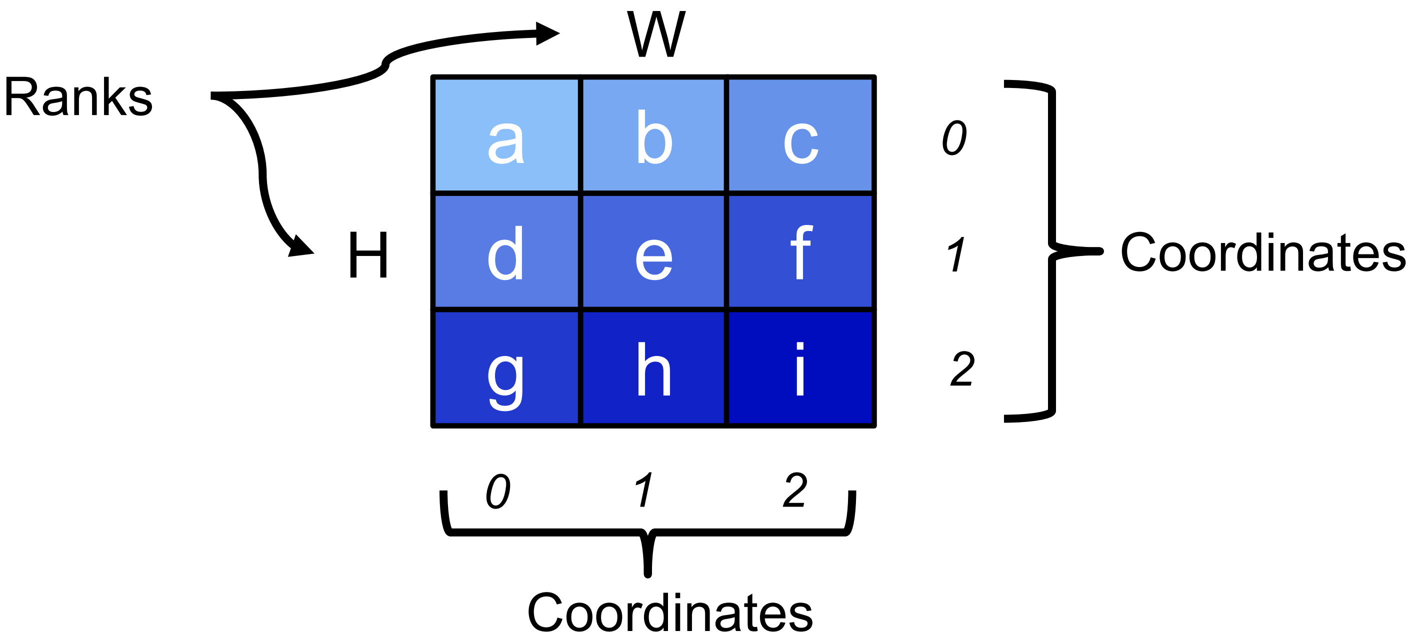
Coordinates

Tensor Data Terminology



	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

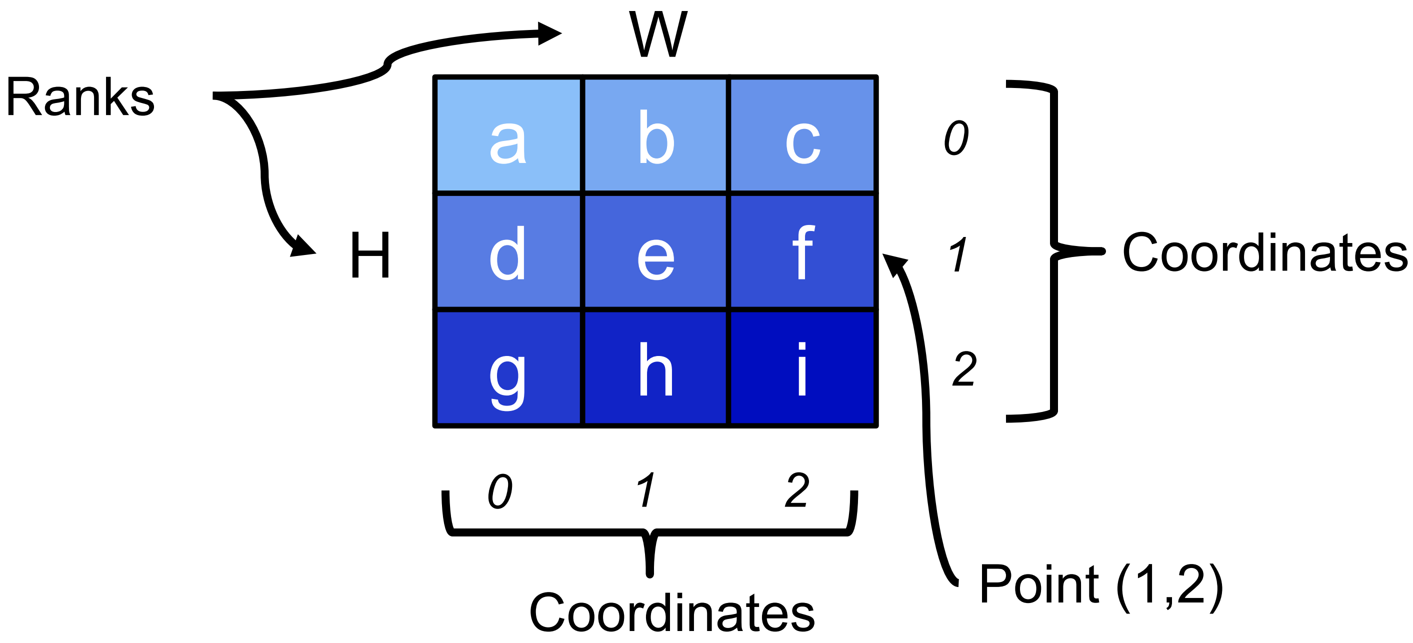
Tensor Data Terminology



	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2

Tensor Data Terminology



Ranks

W

H

	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

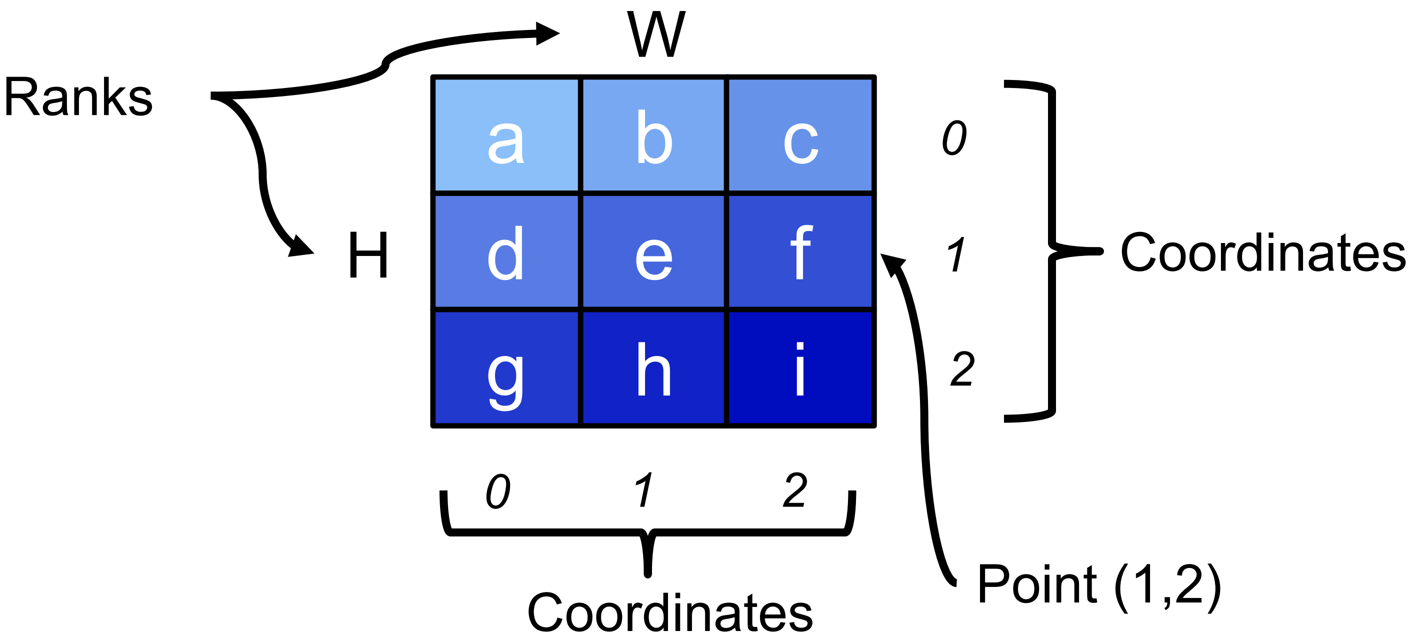
Coordinates

Coordinates

Point (1,2)

- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2

Tensor Data Terminology

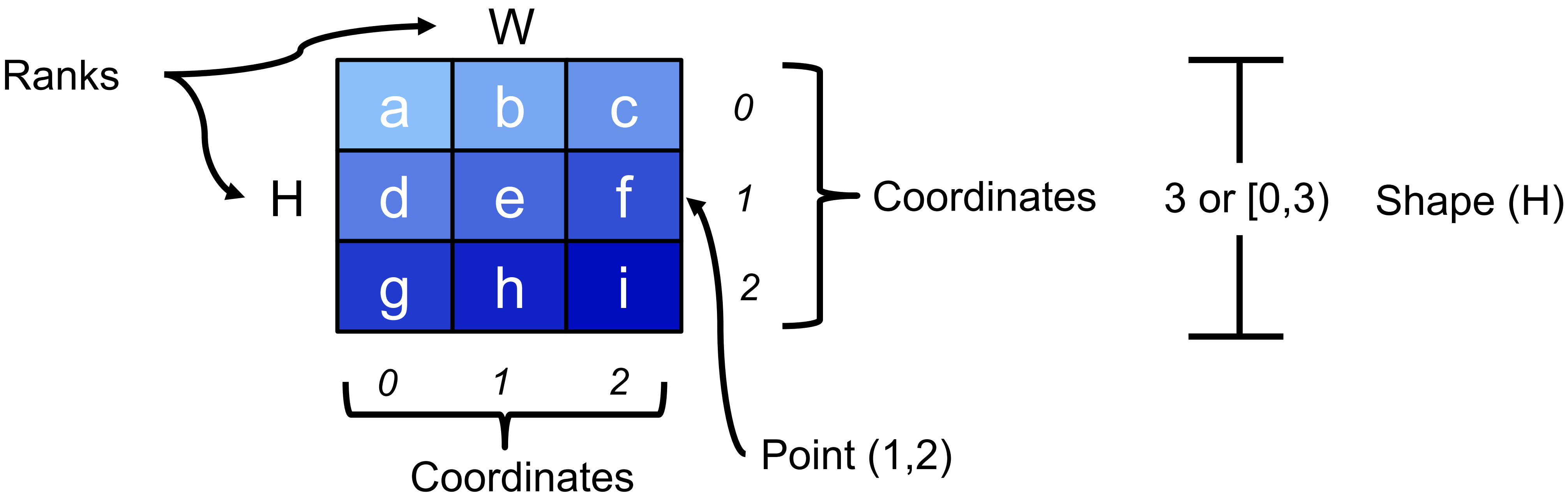


	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2

- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
  So (1,2) -> “f”

Tensor Data Terminology

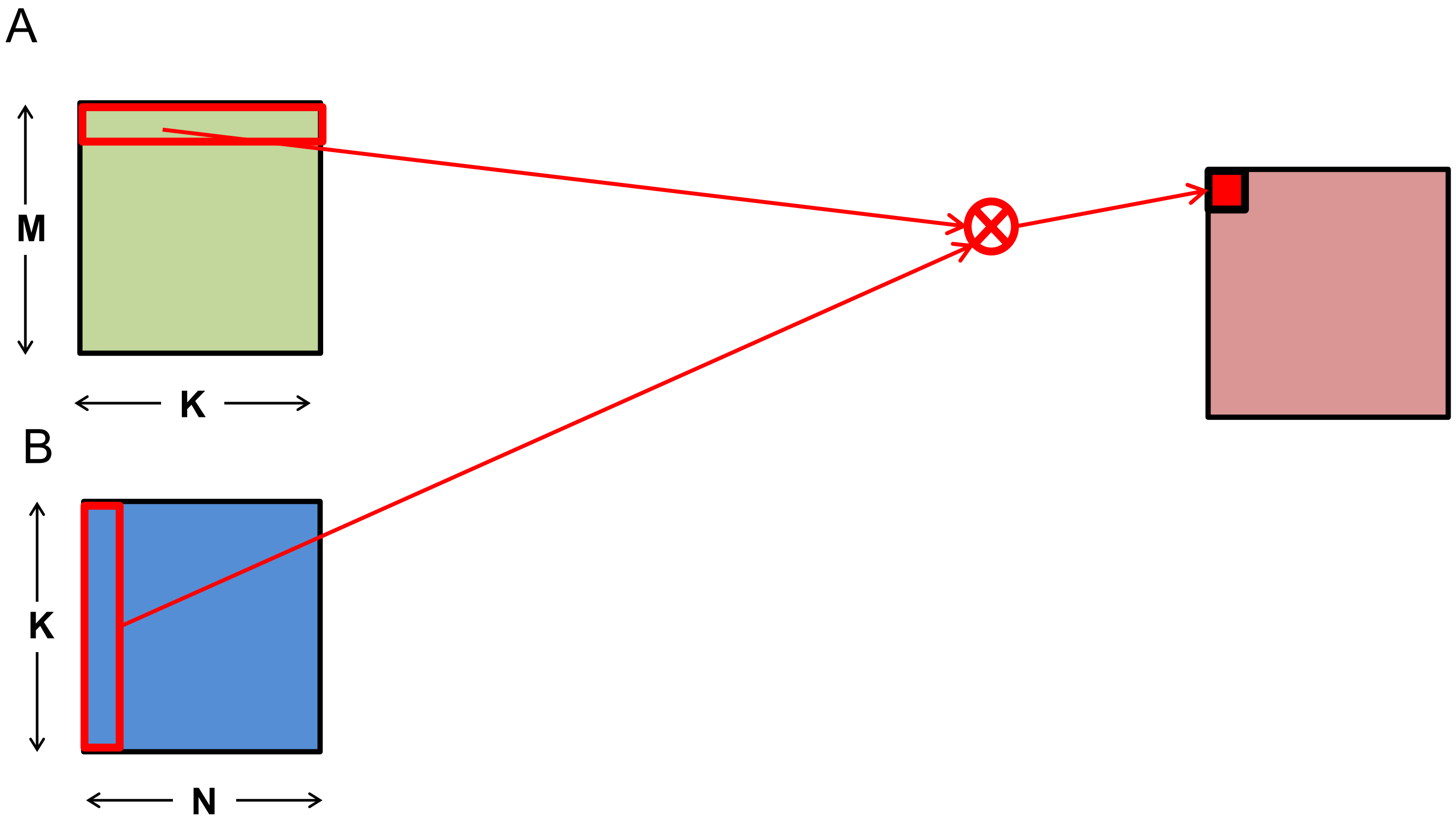


	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

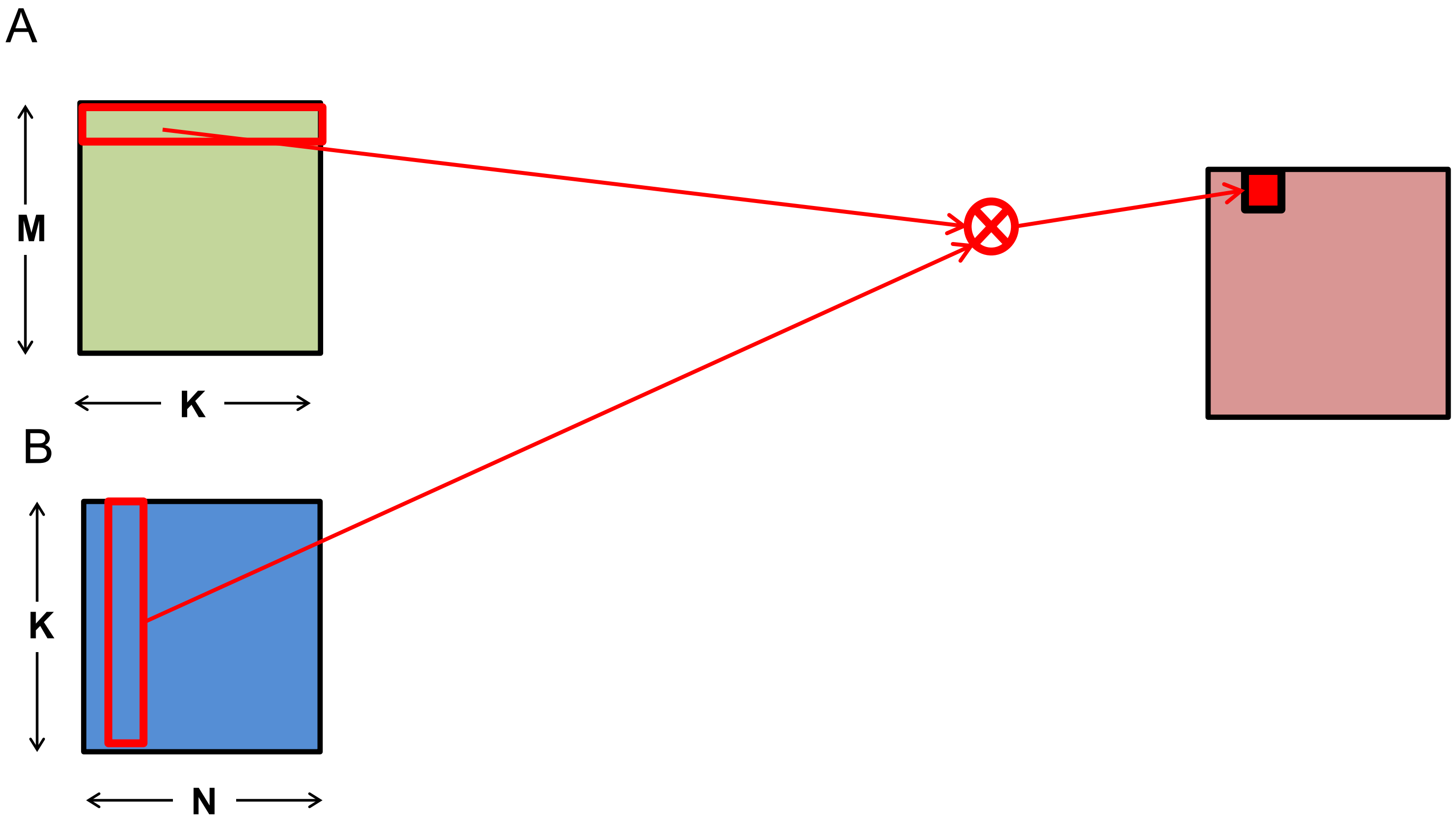
- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2

- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
  So (1,2) -> “f”

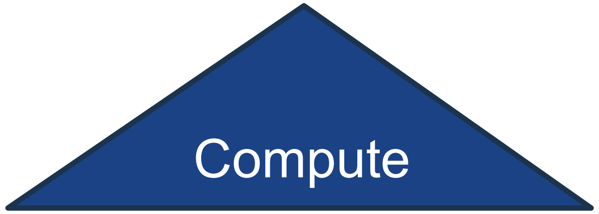
Matrix Multiply



Matrix Multiply



Matrix Multiply – Compute

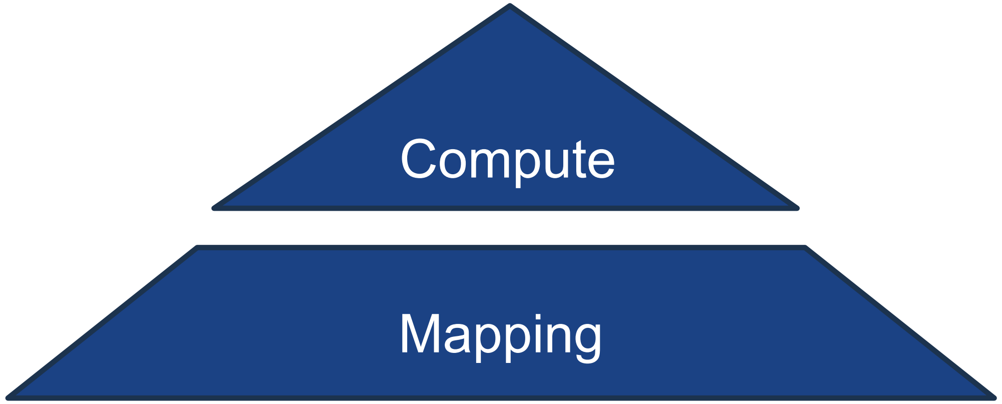


Compute

```
A = Tensor(shape=[M, K])
B = Tensor(shape=[N, K])
Z = Tensor(shape=[M, N])

for n in [0..N):
    for m in [0..M):
        for k in [0..K):
            Z[m][n] += A[m][k] × B[n][k]
```

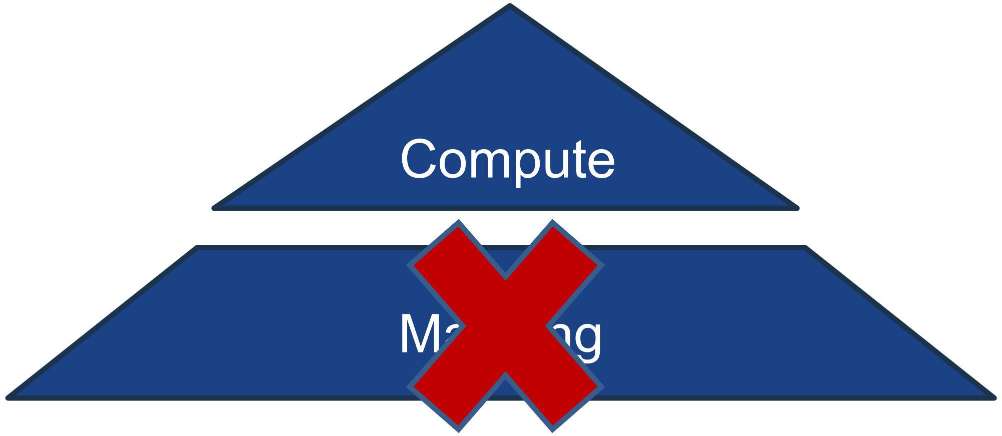
Matrix Multiply – Compute



```
A = Tensor(shape=[M, K])
B = Tensor(shape=[N, K])
Z = Tensor(shape=[M, N])

for n in [0..N):
    for m in [0..M):
        for k in [0..K):
            Z[m][n] += A[m][k] × B[n][k]
```

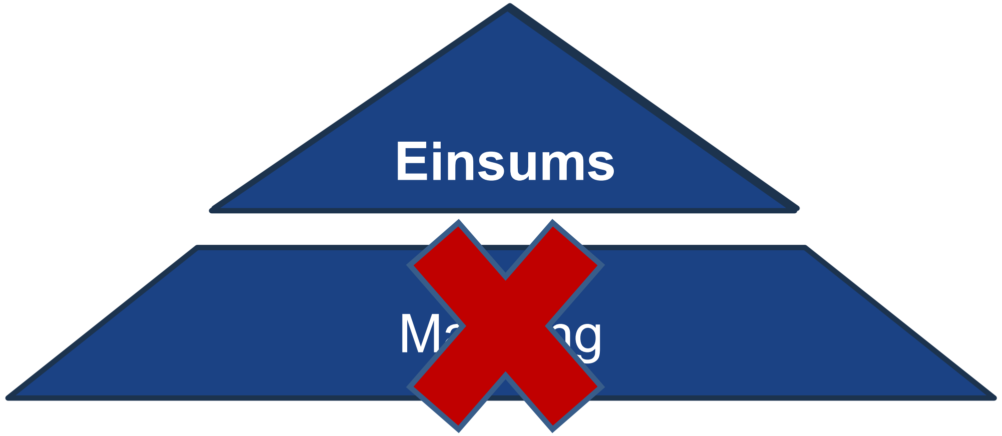
Matrix Multiply – Compute



```
A = Tensor(shape=[M, K])
B = Tensor(shape=[N, K])
Z = Tensor(shape=[M, N])

for n in [0..N):
    for m in [0..M):
        for k in [0..K):
            Z[m][n] += A[m][k] × B[n][k]
```

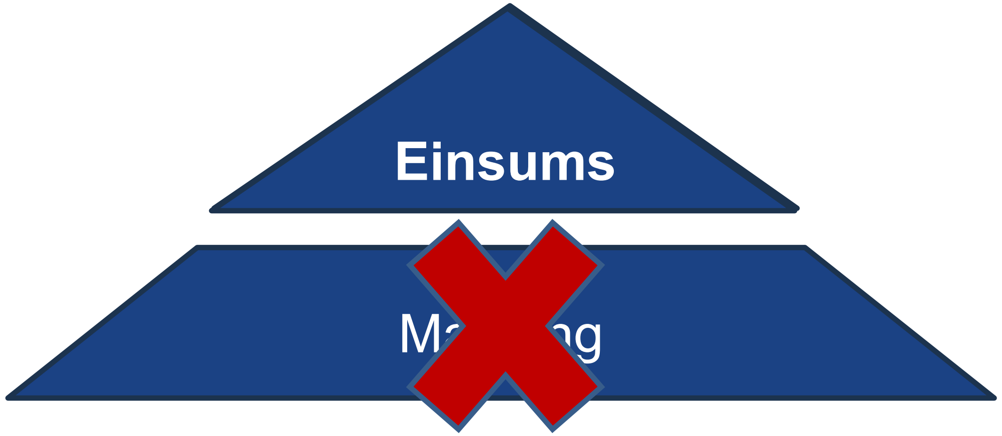
Matrix Multiply – Compute



```
A = Tensor(shape=[M, K])
B = Tensor(shape=[N, K])
Z = Tensor(shape=[M, N])

for n in [0..N):
    for m in [0..M):
        for k in [0..K):
              Z[m][n] += A[m][k] × B[n][k]
```

Matrix Multiply – Compute



```
A = Tensor(shape=[M, K])
B = Tensor(shape=[N, K])
Z = Tensor(shape=[M, N])

for n in [0..N):
    for m in [0..M):
        for k in [0..K):
            Z[m][n] += A[m][k] × B[n][k]
```

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
    - Calculate value on right hand at specified indices for each operand

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
    - Calculate value on right hand at specified indices for each operand
    - Assign value to operand at specified indices on left hand side

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
  - Calculate value on right hand at specified indices for each operand
  - Assign value to operand at specified indices on left hand side
  - Unless that operand is non-zero, then reduce value into it

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = \sum_{k} A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
  - Calculate value on left hand at specified indices for each operand
  - Assign value to operand at specified indices on right hand side
  - Unless that operand is non-zero, then reduce value into it

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – Matrix Multiply

$$Z_{m,n} = \xcancel{\sum_{k}} A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
  - Calculate value on left hand at specified indices for each operand
  - Assign value to operand at specified indices on right hand side
  - Unless that operand is non-zero, then reduce value into it

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Einsum – More notation

$$Z_{m,n}^{M,N} = A_{m,k}^{M,K} \times B_{n,k}^{N,K}$$

Superscript represents the name of rank of the tensor

Einsum – More notation

$$Z_{m,n}^{M,N} = A_{m,k}^{M,K} \times B_{n,k}^{N,K}$$

Superscript represents the name of rank of the tensor

$$Z_{m,n}^{M=3,N=3} = A_{m,k}^{M=3,K=6} \times B_{n,k}^{N=3,K=6}$$

Equals in superscript represents the shape of the rank of the tensor, by default rank shape is assumed to be same as rank name

Einsum-level analysis - MM

$$Z_{m,n}^{M,N} = A_{m,k}^{M,K} \times B_{n,k}^{N,K}$$

$$Number\ of\ multiplies:$$
$$M \times N \times K$$

$$Minimum\ amount\ of\ data\ to\ read:$$
$$M \times K + N \times K$$

$$Minimum\ amount\ of\ data\ to\ write:$$
$$M \times N$$

Einsum-level analysis – M dot products

$$Z_m^M = A_{m,k}^{M,K} \times B_{m,k}^{M,K}$$

$$Number\ of\ multiplies:$$
$$M \times K$$

$$Minimum\ amount\ of\ data\ to\ read:$$
$$M \times K + M \times K$$

$$Minimum\ amount\ of\ data\ to\ write:$$
$$M$$

Einsum – Matrix Multiply

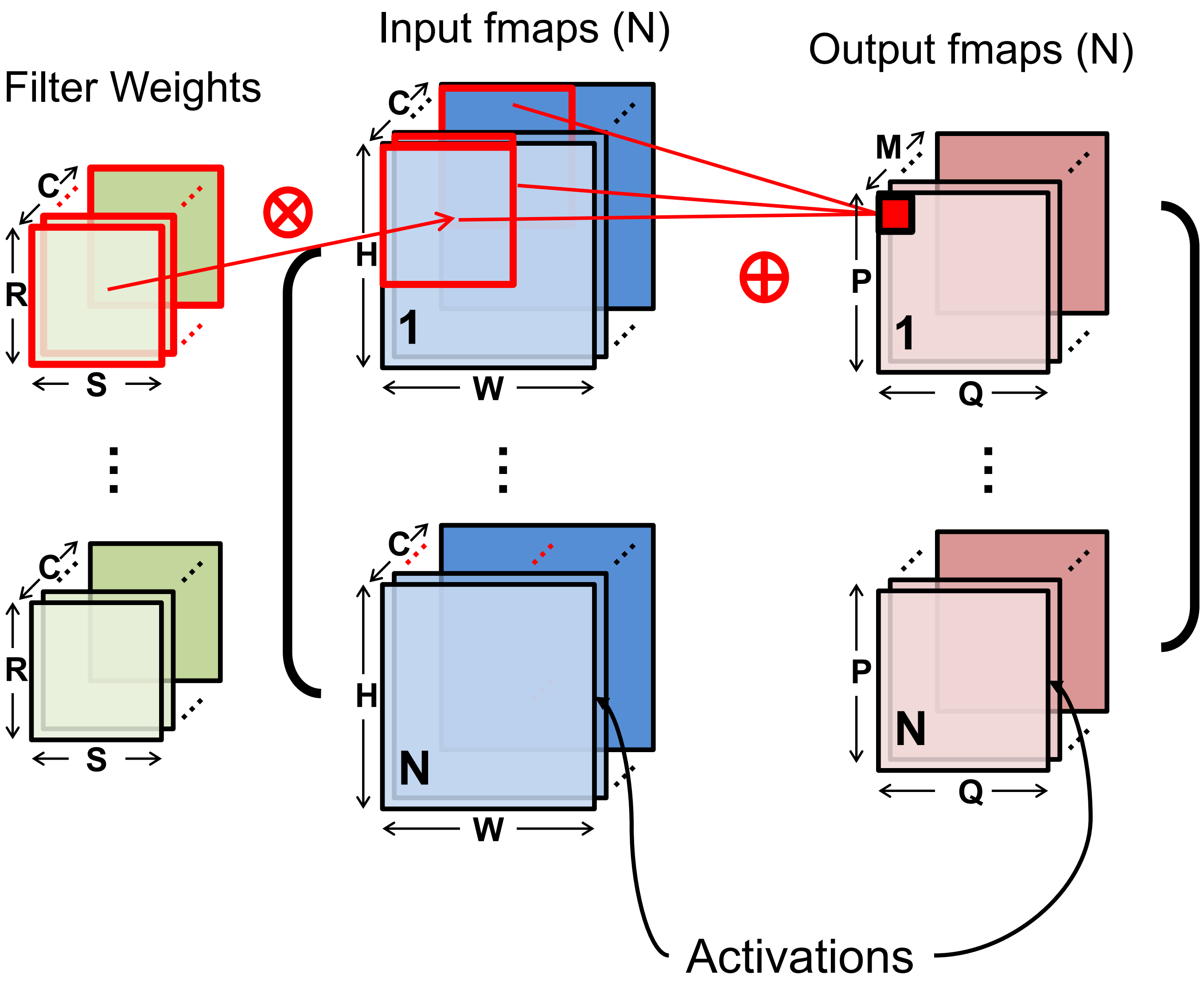
$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Operational Definition for Einsums (ODE):

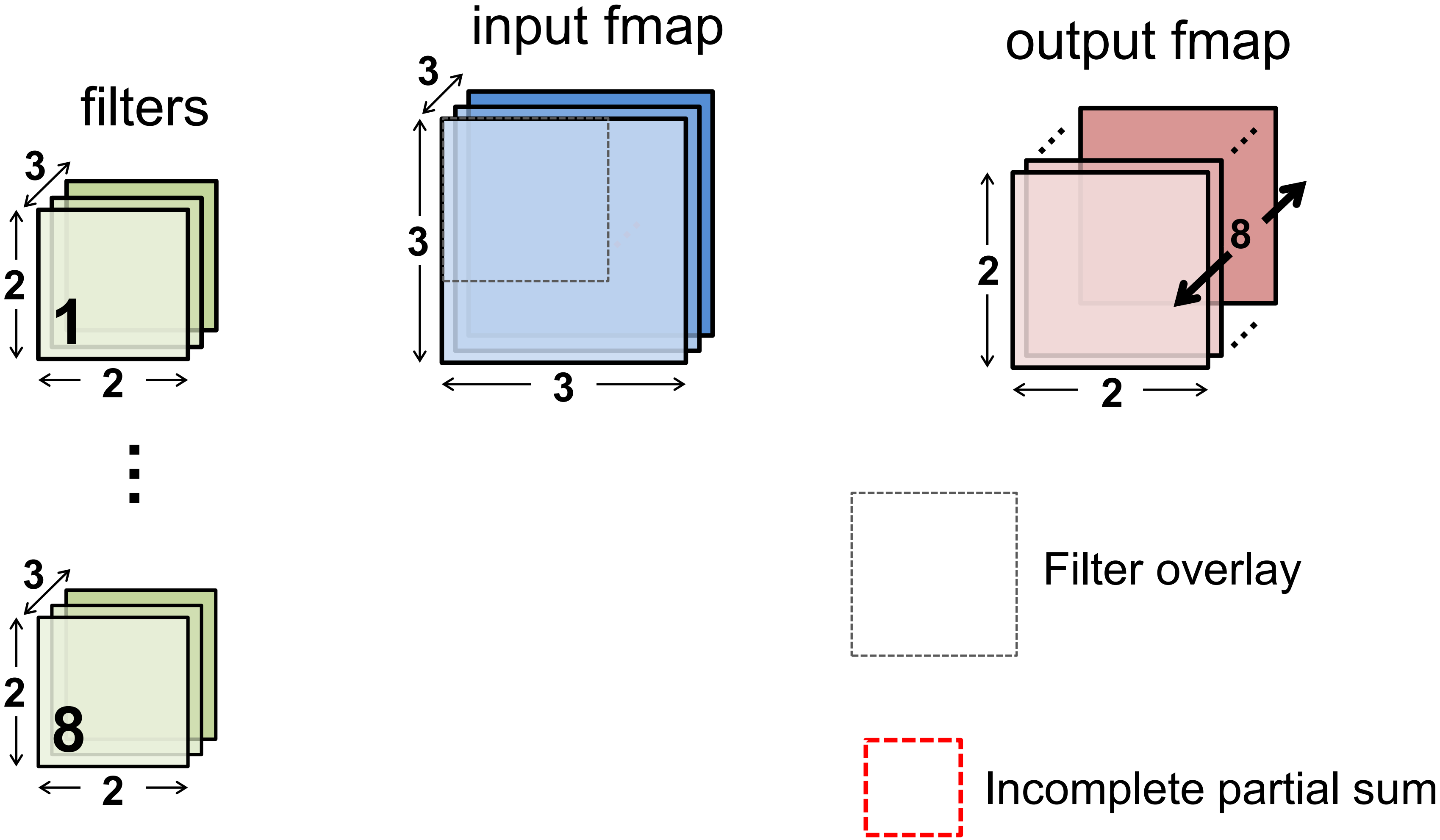
- Traverse all points in space of all legal index values (iteration space)
- At each point in iteration space:
  - Calculate value on right hand at specified indices for each operand
  - Assign value to operand at specified indices on left hand side
  - Unless that operand is non-zero, then reduce value into it

[Relativity, Einstein, Annelen de Physik, 1916]
[Numpy/Einsum Python, ~2015]
[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]
[SAM, Hsu et.al., ASPLOS 2023]

Convolution (CONV) Layer



CONV Computation

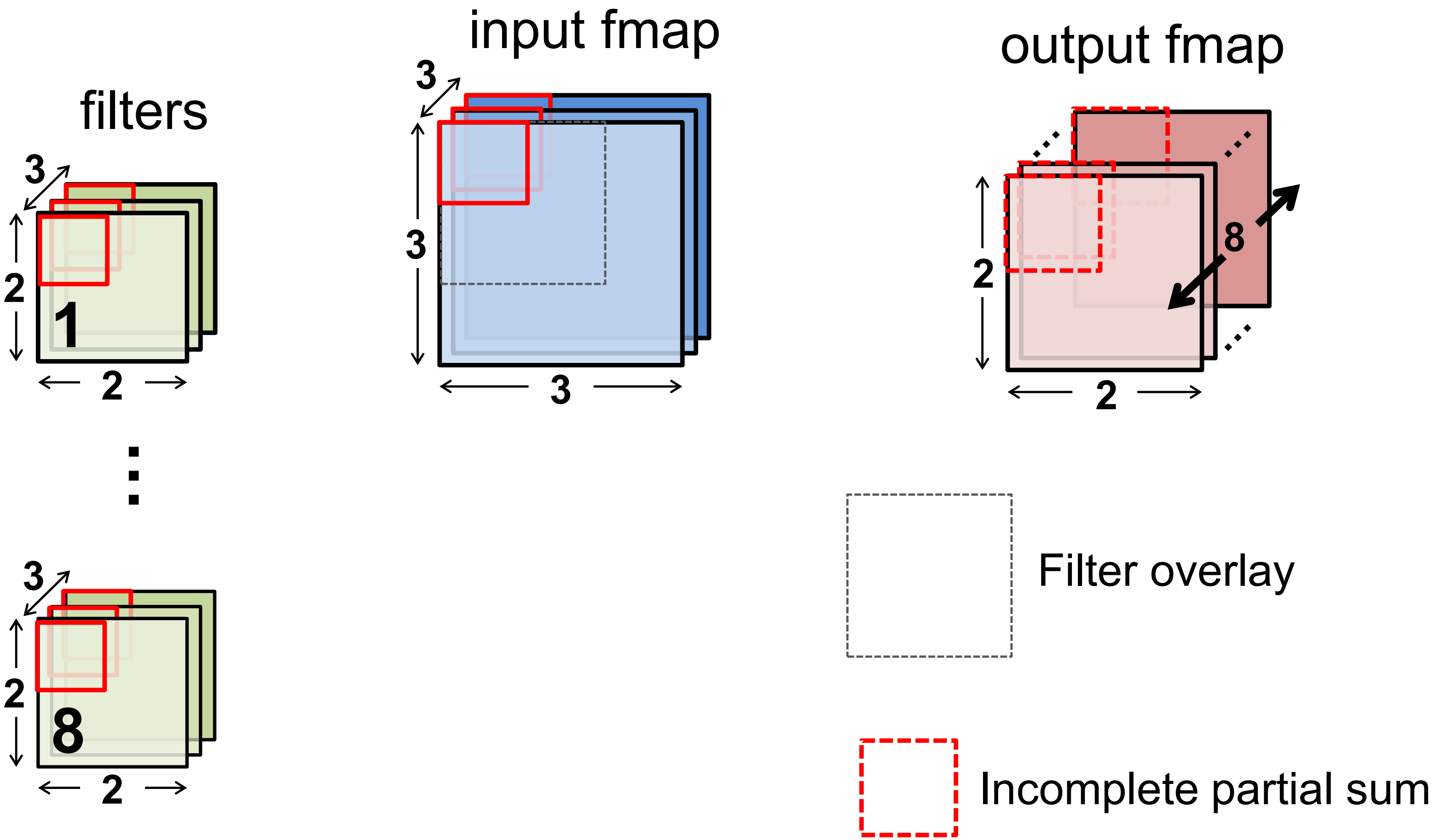


Filter overlay

Incomplete partial sum

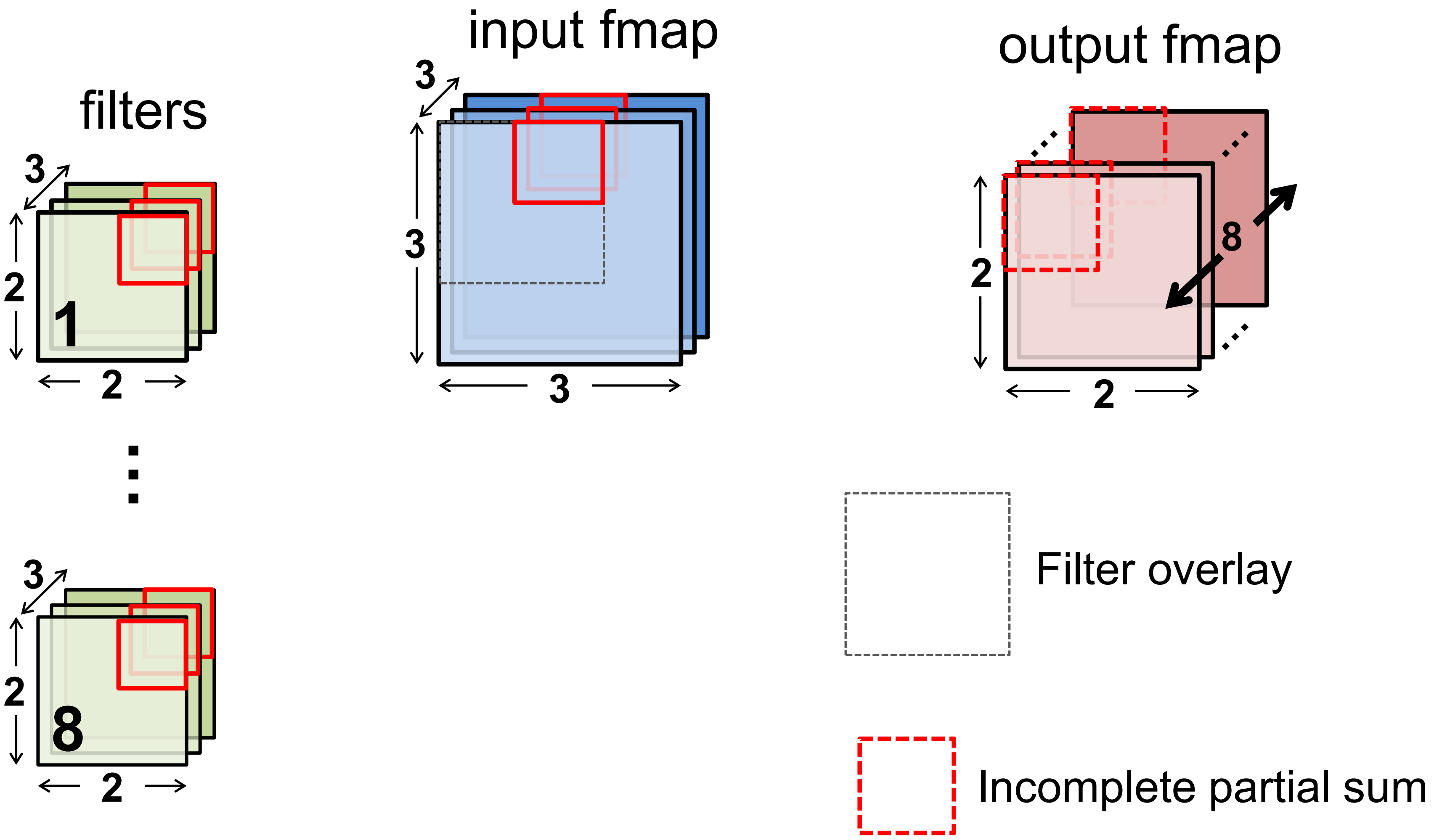
CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



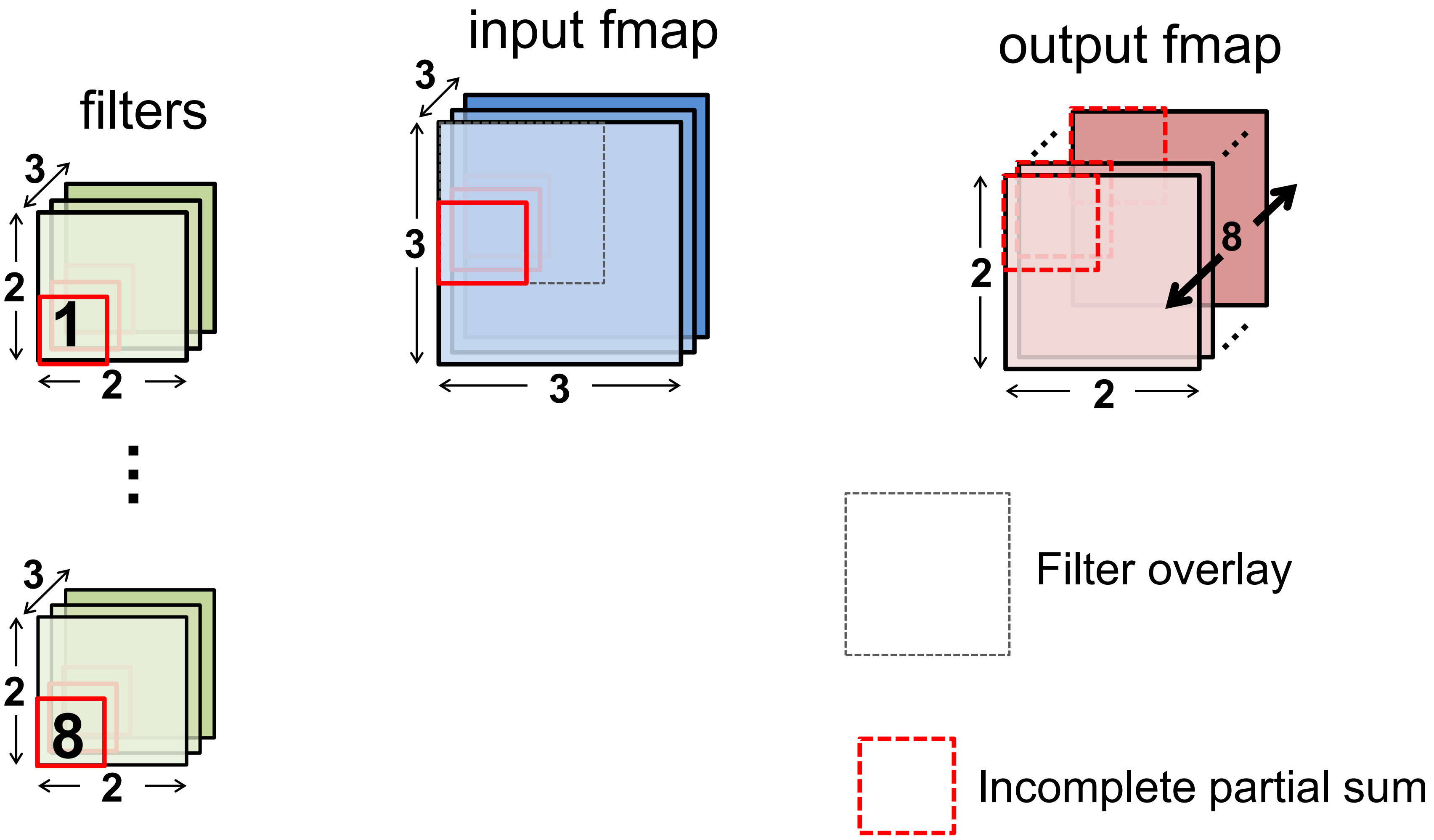
CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



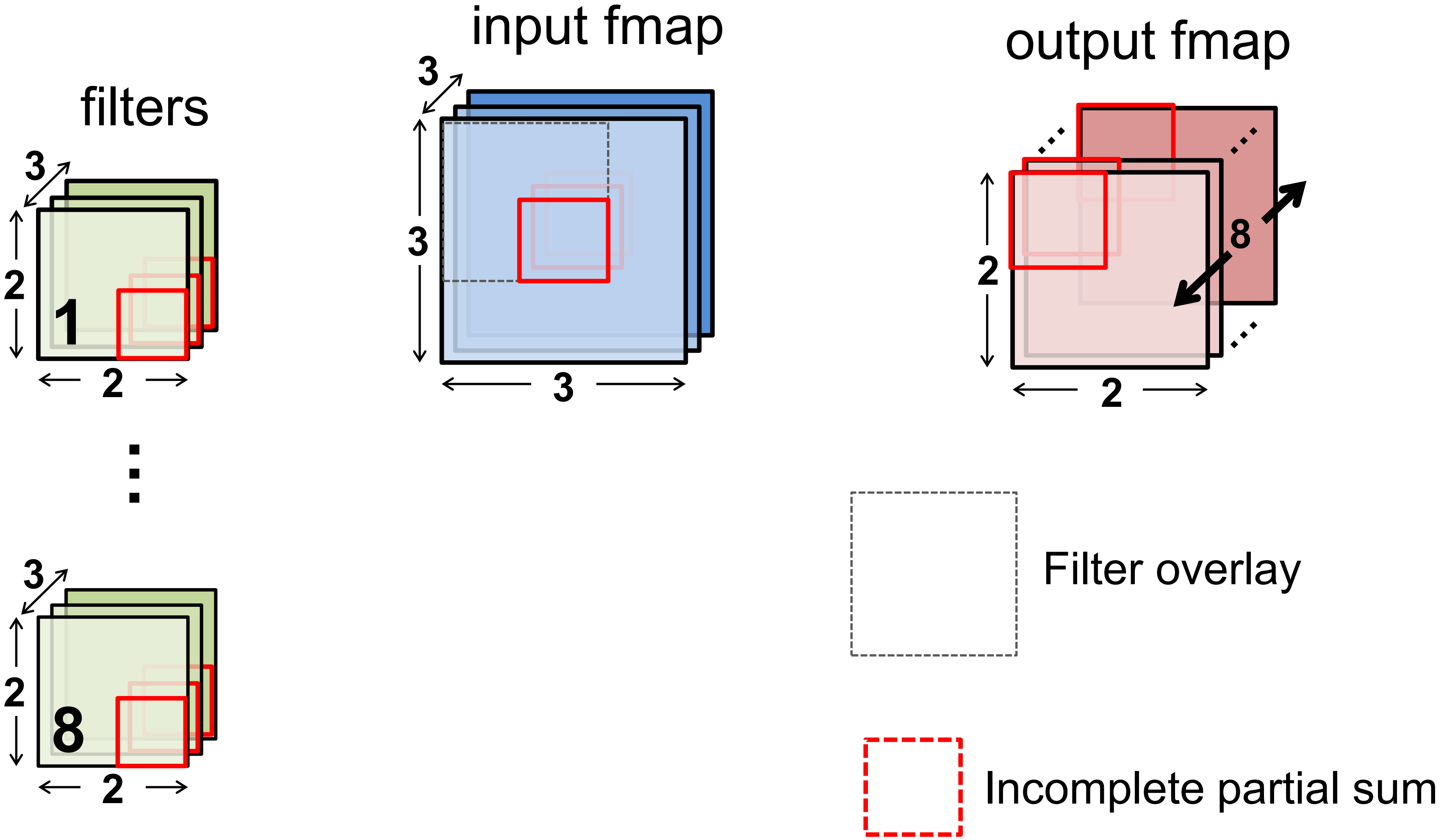
CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



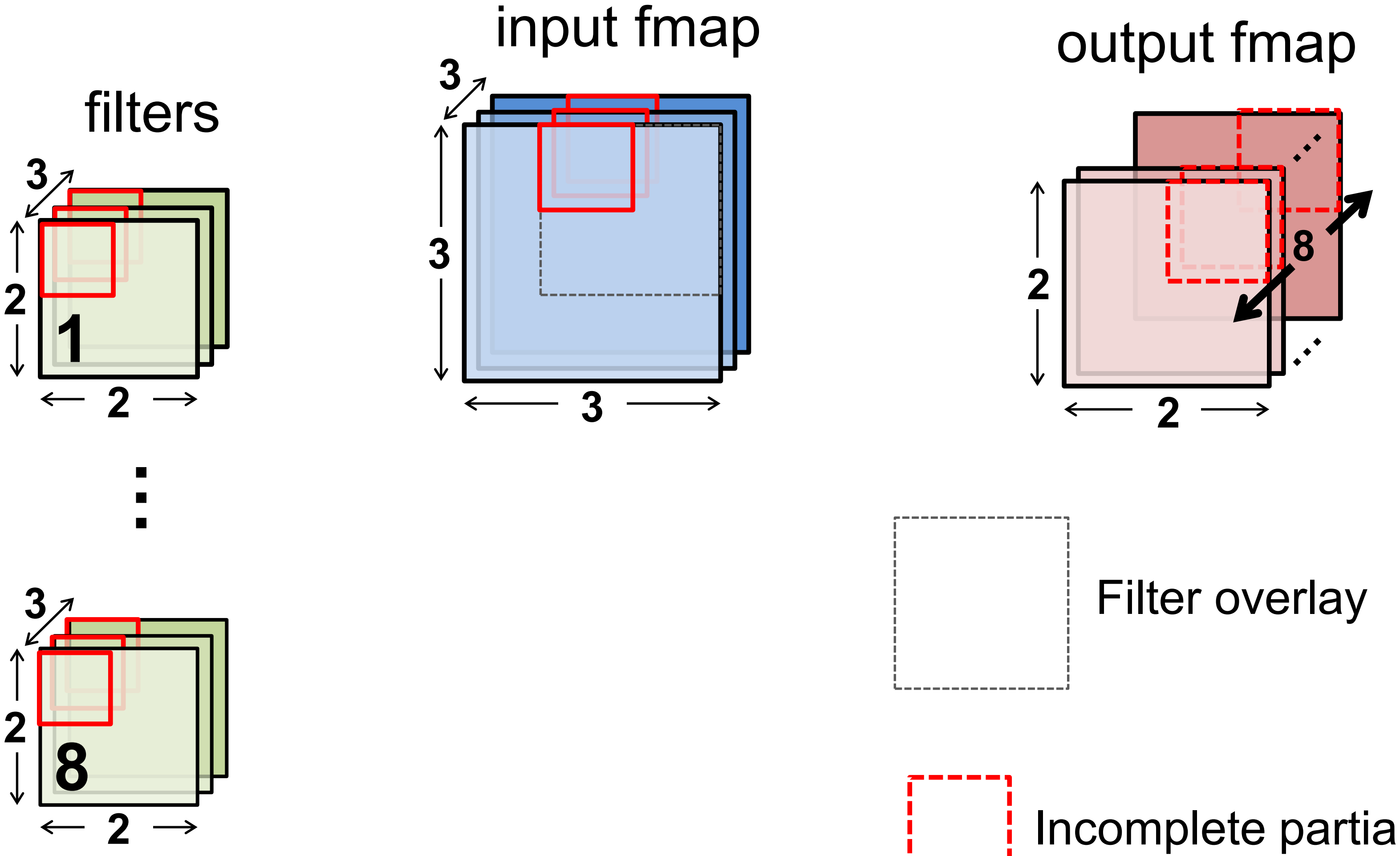
CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)

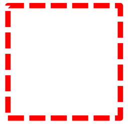


CONV Computation

Start processing next output feature activations



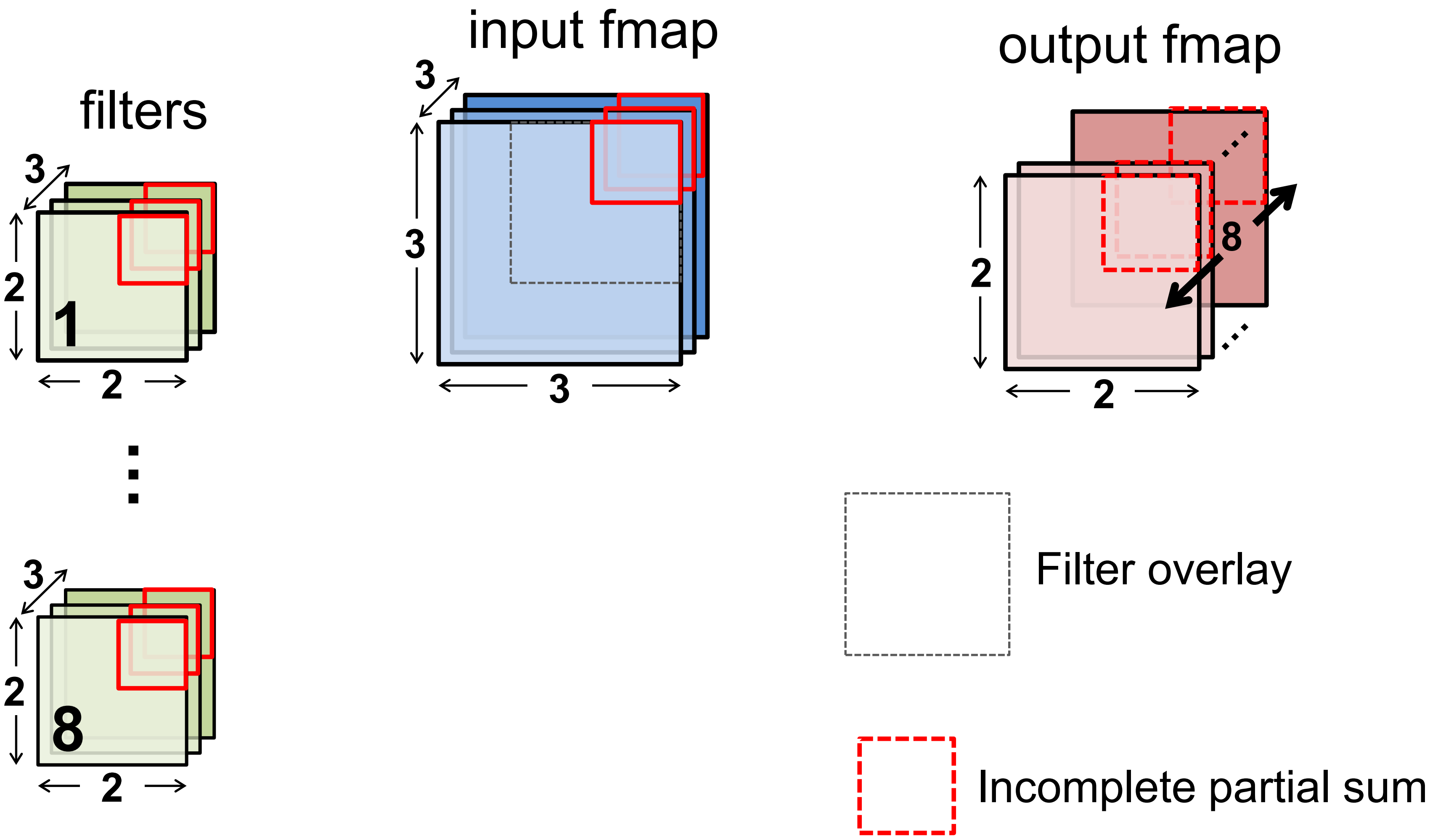
Filter overlay



Incomplete partial sum

CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



filters

input fmap

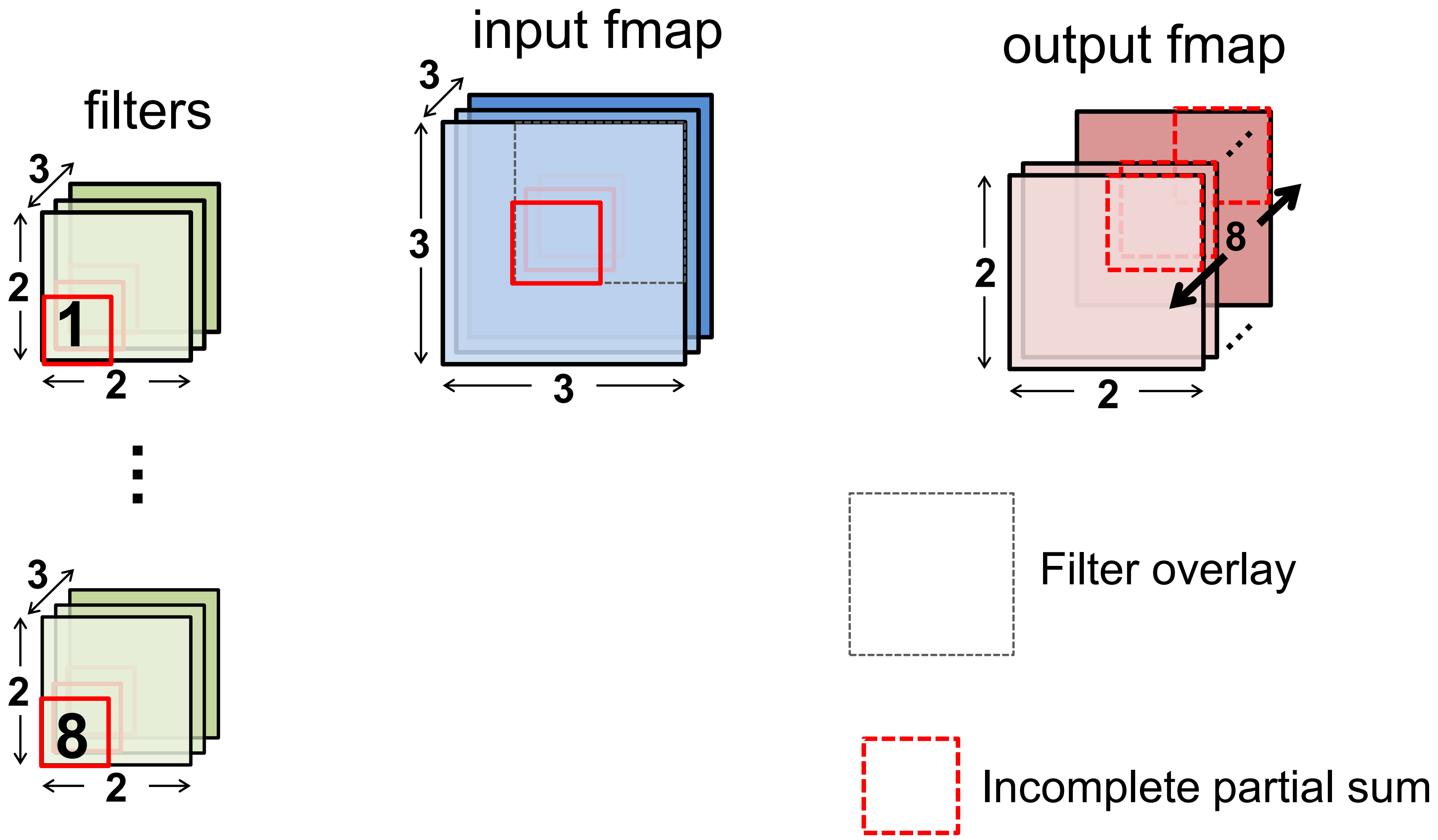
output fmap

Filter overlay

Incomplete partial sum

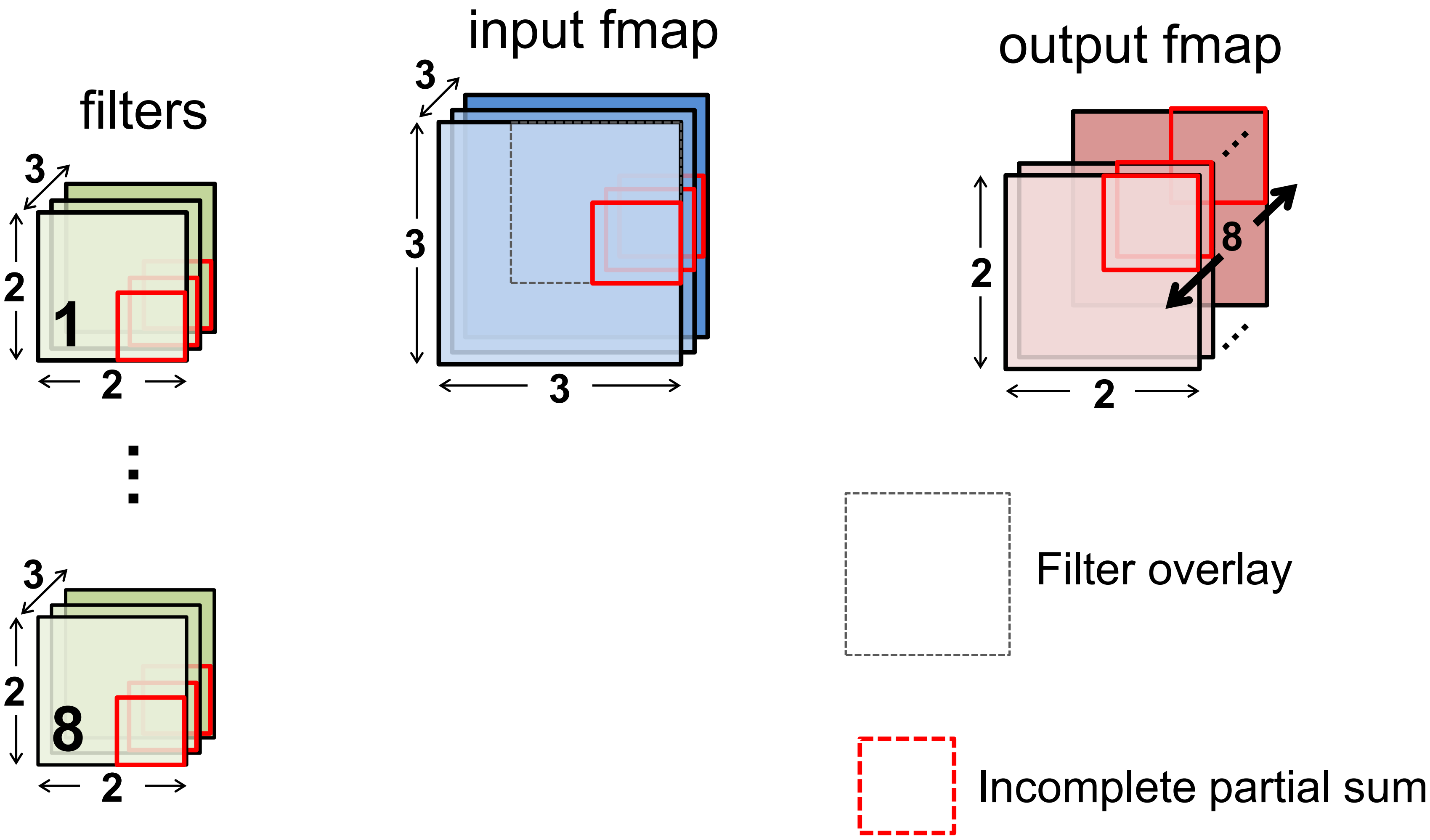
CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



CONV Computation

Cycle through input fmap and weights (hold psum of output fmap)



CONV Layer Implementation

Naïve 7-layer for-loop implementation:

```
for n in [0..N):
    for m in [0..M):
        for q in [0..):
            for p in [0..P):
                O[n][m][p][q] = B[m];
                for r in [0..R):
                    for s in [0..S):
                        for c in [0..C):
                            O[n][m][p][q] += I[n][c][Up+r][Uq+s] × F[m][c][r][s];
                        }
                    }
                }

                O[n][m][p][q] = Activation(O[n][m][p][q]);
            }
        }
    }
}
```

for each output fmap value

convolve
a window
and apply
activation

Einsum - Convolution

$$O_{p,q,m} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of channels in input fmaps (activations) & filters (weights)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of filter (weights)
- S – Width of filter (weights)
- M – Number of channels in output fmaps (activations)
- P – Height of output fmap (activations)
- Q – Width of output fmap (activations)
- U – Stride of convolution

CONV Variants

- Depthwise layer - $M == C \; and \; \forall_{c \,!= m} F_{c,m,r,s} = 0$
- Pointwise layer - $R == S == 1$
- Matrix multiply - $R == S == H == 1$
- Compress (pointwise) - $M < C \; and \; R == S == 1$
- Expand (pointwise) - $M > C \; and \; R == S == 1$

Compress…Expand sequences are called a “bottleneck”

Toeplitz (IM2COL) Cascade

$$O_{p,q,m} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

$$T_{c.p,q,r,s} = I_{c,p+r,q+s}$$

$$O_{p,q,m} = T_{c,p,q,r,s} \times F_{m,c,r,s}$$

Conventional Transformer Diagram

This time, composition of many kernels

Not a precise description of functionality.

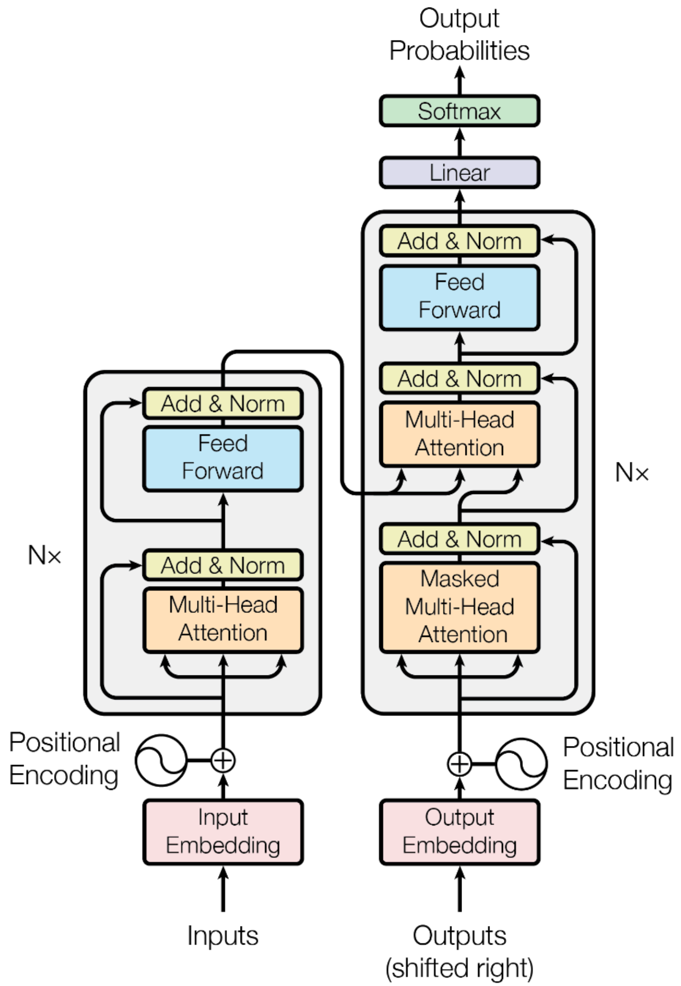


Figure 1: The Transformer - model architecture.

[Attention, Vaswani et al. 2016]

Multi-head Attention (without initial embedding step)

$$K_{b,h,m,e} = I_{b,m,d} \times WK_{d,h,e}$$

$$Q_{b,h,m,e} = I_{b,m,d} \times WQ_{d,h,e}$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = Q_{b,h,p,e}^{B,H,M,E} \times K_{b,h,m,e}$$

$$SN_{b,h,m,p} = exp\left(QK_{b,h,m,p}\right)$$

$$SD_{b,h,p} = SN_{b,h,m,p}$$

$$A_{b,h,m,p} = SN_{b,h,m,p}/SD_{b,h,p}$$

$$V_{b,h,m,f} = I_{b,m,d} \times WV_{d,h,f}$$

$$AV_{b,h,p,f}^{B,H,P=M,F} = A_{b,h,m,p} \times V_{b,h,m,f}$$

$$C_{b,p,h\times F+f}^{B,P=M,G=H\times F} = AV_{b,h,p,f}$$

$$Z_{b,p,d} = C_{b,p,f} \times WZ_{g,d}$$

Separation of Concerns

Ideas for Sparse Attention in Transformers

Base transformer [Vaswani et al. 2016]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK^{B,H,M,P=M}_{b,h,m,p} = \frac{1}{\sqrt{E}} \times Q^{B,E,H,M}_{b,e,h,p} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN^{B,H,M,P=M}_{b,h,m,p} = e^{QK_{b,h,m,p}}$$

$$SD^{B,H,P=M}_{b,h,p} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C^{B,G=H\times F,P=M}_{b,h\times F+f,p} = AV_{b,f,h,p}$$

$$Z_{b,d,p} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

SpAtten [Wang et al. 2020]

$$IP_{i,b,d,m} = MT_{i,b,m} \times IKV_{b,d,m}$$

$$Q_{i,b,e,h,r} = MH_{i,b,h} \times IQ_{b,d,r} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{i,b,e,h,m} = MH_{i,b,h} \times IP_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{i,b,f,h,m} = MH_{i,b,h} \times IP_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK_{i,b,h,m,r} = \frac{1}{\sqrt{E}} \times Q_{i,b,e,h,r} \times K_{i,b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN_{i,b,h,m,r} = e^{QK_{i,b,h,m,r}}$$

$$SD_{i,b,h,r} = SN_{i,b,h,m,r} : \bigvee_{m} +(\cup)$$

$$A_{i,b,h,m,r} = SN_{i,b,h,m,r} / SD_{i,b,h,r}$$

$$AV_{i,b,f,h,r} = A_{i,b,h,m,r} \times V_{i,b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C^{B,G=H\times F,R}_{i,b,h*F+f,r} = AV_{i,b,f,h,r}$$

$$Z_{i,b,d,r} = C_{i,b,g,r} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

$$ZA_{i,b,f,h,r} = Z_{i,b,h*F+f,r}$$

$$MT_{i+1,b,m} = prune(A_{i,b,h,m,r}) \times MT_{i,b,m}$$

$$MH_{i+1,b,h} = prune(ZA_{i,b,f,h,r}) \times MH_{i,b,h}$$

$$ST_{i+1,b,m} = ST_{i,b,m} + A_{i,b,h,m,r}$$

$$SH_{i+1,b,h} = SH_{i,b,h} + ZA_{i,b,f,h,r}$$

Ideas for Sparse Attention in Transformers

Base transformer [Vaswani et al. 2016]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = \frac{1}{\sqrt{E}} \times Q_{b,e,h,p}^{B,E,H,M} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN_{b,h,m,p}^{B,H,M,P=M} = e^{QK_{b,h,m,p}}$$

$$SD_{b,h,p}^{B,H,P=M} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C_{b,h\times F+f,p}^{B,G=H\times F,P=M} = AV_{b,f,h,p}$$

$$Z_{b,d,p} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

Sanger [Lu et al. 2021]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$\hat{Q}_{b,e,h,m} = Qt(Q_{b,e,h,m})$$

$$\hat{K}_{b,e,h,m} = Qt(K_{b,e,h,m})$$

$$\widehat{QK}_{b,h,m,p}^{B,H,M,P=M} = \frac{1}{\sqrt{E}} \times \hat{Q}_{b,e,h,p}^{B,E,H,M} \times \hat{K}_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$\widehat{SN}_{b,h,m,p} = e^{\widehat{QK}_{b,h,m,p}}$$

$$\widehat{SD}_{b,h,p} = \widehat{SN}_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$\hat{A}_{b,h,m,p} = \widehat{SN}_{b,h,m,p} / \widehat{SD}_{b,h,p}$$

$$M_{b,h,m,p} = prune(\hat{A}_{b,h,m,p})$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = \frac{1}{\sqrt{E}} \times M_{b,h,m,p} \times Q_{b,e,h,p}^{B,E,H,M} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN_{b,h,m,p}^{B,H,M,P=M} = e^{QK_{b,h,m,p}}$$

$$SD_{b,h,p}^{B,H,P=M} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p}^{B,H,M,P=M} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p}^{B,F,H,P=M} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C_{b,h*F+f,p}^{B,G=H\times F,P=M} = AV_{b,f,h,p}$$

$$Z_{b,d,p}^{B,D,P=M} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

Ideas for Sparse Attention in Transformers

Base transformer [Vaswani et al. 2016]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK^{B,H,M,P=M}_{b,h,m,p} = \frac{1}{\sqrt{E}} \times Q^{B,E,H,M}_{b,e,h,p} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN^{B,H,M,P=M}_{b,h,m,p} = e^{QK_{b,h,m,p}}$$

$$SD^{B,H,P=M}_{b,h,p} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C^{B,G=H\times F,P=M}_{b,h\times F+f,p} = AV_{b,f,h,p}$$

$$Z_{b,d,p} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

EdgeBERT [Tambe et al. 2021]

$$MH_h = take(M_{h,m,p}, zeroes(m,p), 0)$$

$$Q_{b,e,h,m} = MH_h \times I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = MH_h \times I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = MH_h \times I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK^{B,H,M,P=M}_{b,h,m,p} = \frac{1}{\sqrt{E}} \times Q^{B,E,H,M}_{b,e,h,p} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN^{B,H,M,P=M}_{b,h,m,p} = e^{QK_{b,h,m,p}}$$

$$SD^{B,H,P=M}_{b,h,p} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p} = M_{h,m,p} \times A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C^{B,G=H\times F,P=M}_{b,h*F+f,p} = AV_{b,f,h,p}$$

$$Z^{B,D,P=M}_{b,d,p} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

Ideas for Sparse Attention in Transformers

Base transformer [Vaswani et al. 2016]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = \frac{1}{\sqrt{E}} \times Q_{b,e,h,p}^{B,E,H,M} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN_{b,h,m,p}^{B,H,M,P=M} = e^{QK_{b,h,m,p}}$$

$$SD_{b,h,p}^{B,H,P=M} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C_{b,h\times F+f,p}^{B,G=H\times F,P=M} = AV_{b,f,h,p}$$

$$Z_{b,d,p} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

DOTA [Qu et al. 2022]

$$Q_{b,e,h,m} = I_{b,d,m} \times WQ_{d,e,h} : \bigvee_{d} +(\cup)$$

$$K_{b,e,h,m} = I_{b,d,m} \times WK_{d,e,h} : \bigvee_{d} +(\cup)$$

$$V_{b,f,h,m} = I_{b,d,m} \times WV_{d,f,h} : \bigvee_{d} +(\cup)$$

$$\tilde{Q}_{b,k,h,m} = I_{b,d,m} \times P_{d,j}^{D,K} \times W\tilde{Q}_{j,k,h} : \bigvee_{d,j} +(\cup)$$

$$\tilde{K}_{b,k,h,m} = I_{b,d,m} \times P_{d,j}^{D,K} \times W\tilde{K}_{j,k,h} : \bigvee_{d,j} +(\cup)$$

$$Q\tilde{K}_{b,h,m,p}^{B,H,M,P=M} = \tilde{Q}_{b,k,h,p}^{B,K,H,M} \times \tilde{K}_{b,k,h,m} : \bigvee_{k} +(\cup)$$

$$M_{b,h,m,p} = prune(Q\tilde{K}_{b,h,m,p})$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = \frac{1}{\sqrt{E}} \times M_{b,h,m,p} \times Q_{b,e,h,p}^{B,E,H,M} \times K_{b,e,h,m} : \bigvee_{e} +(\cup)$$

$$SN_{b,h,m,p}^{B,H,M,P=M} = e^{QK_{b,h,m,p}}$$

$$SD_{b,h,p}^{B,H,P=M} = SN_{b,h,m,p} : \bigvee_{m} +(\cup)$$

$$A_{b,h,m,p}^{B,H,M,P=M} = SN_{b,h,m,p} / SD_{b,h,p}$$

$$AV_{b,f,h,p}^{B,F,H,P=M} = A_{b,h,m,p} \times V_{b,f,h,m} : \bigvee_{m} +(\cup)$$

$$C_{b,h*F+f,p}^{B,G=H\times F,P=M} = AV_{b,f,h,p}$$

$$Z_{b,d,p}^{B,D,P=M} = C_{b,g,p} \times WZ_{d,g} : \bigvee_{g} +(\cup)$$

Einsums: Precise and Concise

	Paper Length	Code length	# Einsums
Attention Is All You Need	15 pages	14 python files	14
FlashAttention	34 pages	47 python files	24 (3 changed)
FlashAttention2	14 pages		25 (11 changed)
Spatten [Wang]	15 pages	8 python files + CPP	19 (3 changed)
Sanger [Lu]	15 pages	16 python files + Scala	21 (1 changed)
EdgeBert [Tambe]	16 pages	80+ python files	15 (4 changed)
DOTA [Qu]	13 pages	N/A	17 (1 changed)

Separation of Concerns

The High Cost of Data Movement

Fetching operands more expensive than computing on them

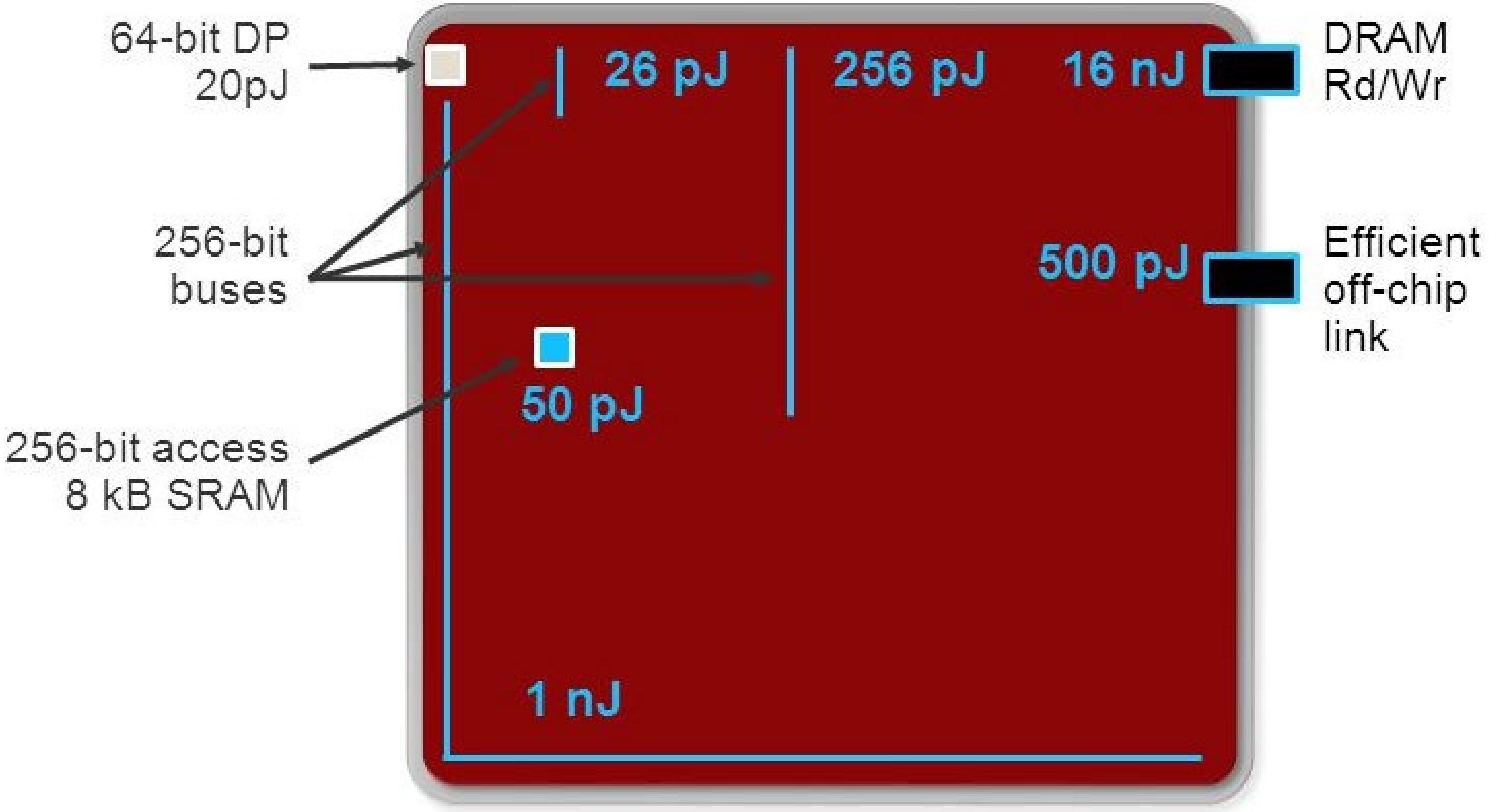
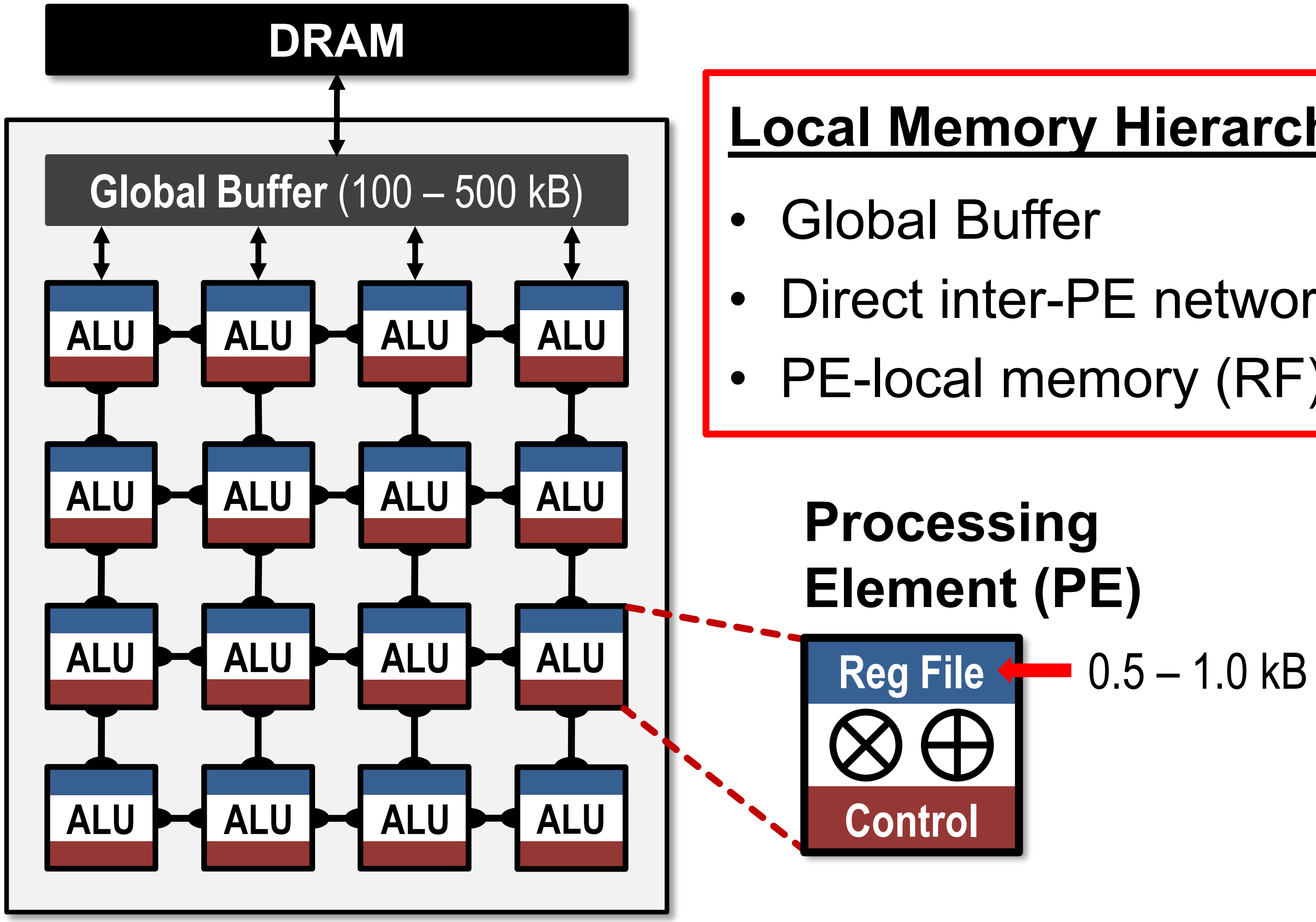


Image source: Bill Daly

Now the key is how we use our transistors most effectively.

Spatial Architecture for DNN



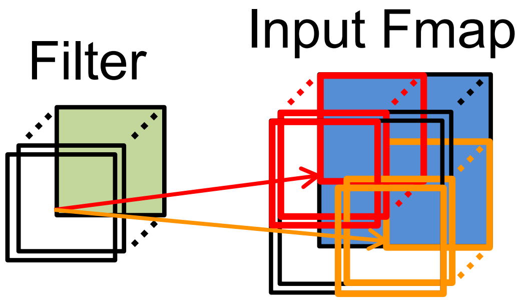
Local Memory Hierarchy

- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)

Types of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

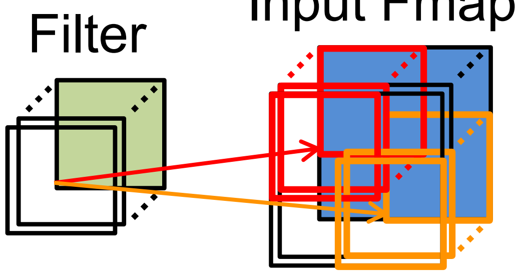


Reuse: Activations
Filter weights

Types of Data Reuse in DNN

Convolutional Reuse

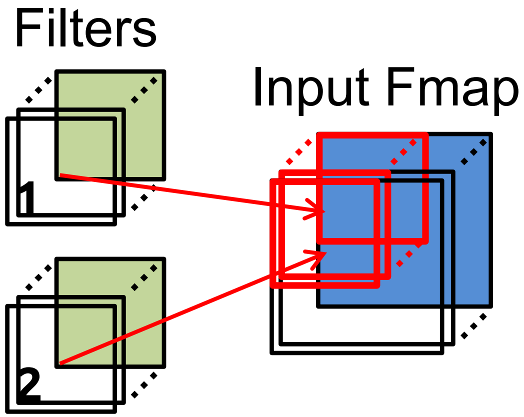
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

CONV and FC layers

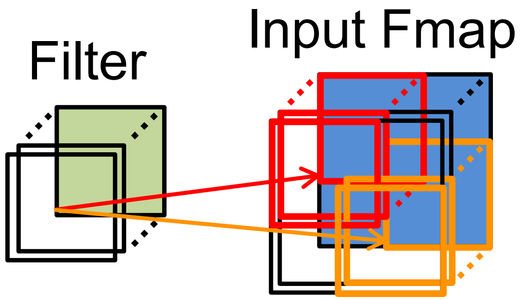


Reuse: Activations

Types of Data Reuse in DNN

Convolutional Reuse

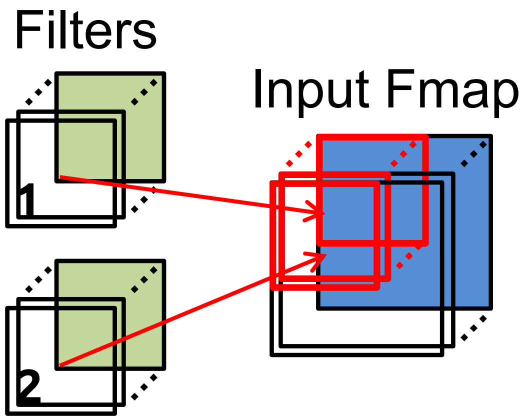
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

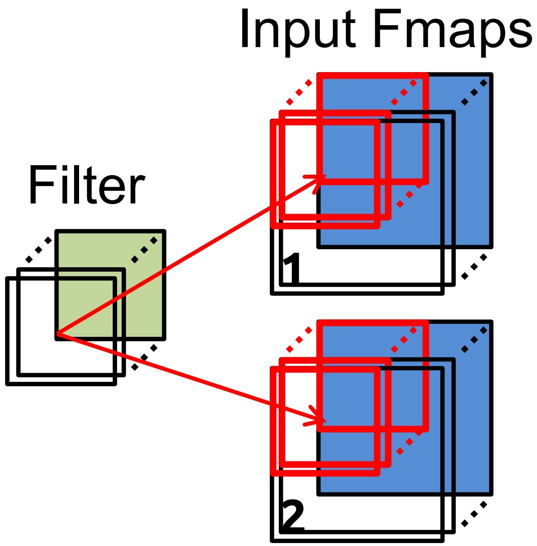
CONV and FC layers



Reuse: Activations

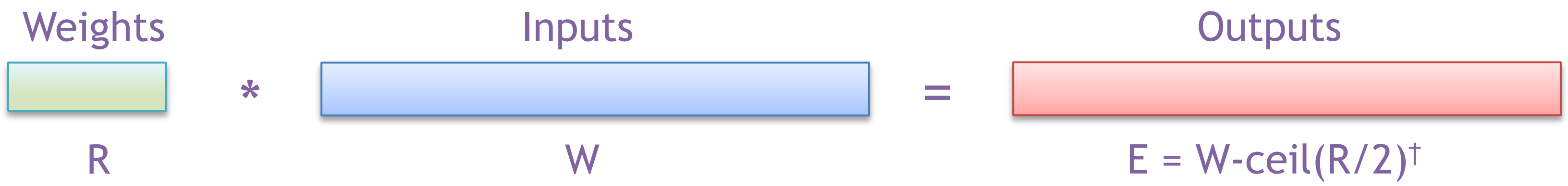
Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

1-D Convolution



Weights * Inputs = Outputs

R

W

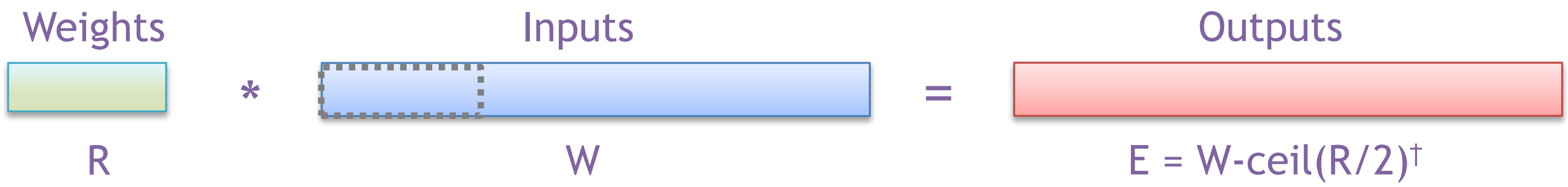
E = W-ceil(R/2)†

```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: 'valid' style convolution

1-D Convolution

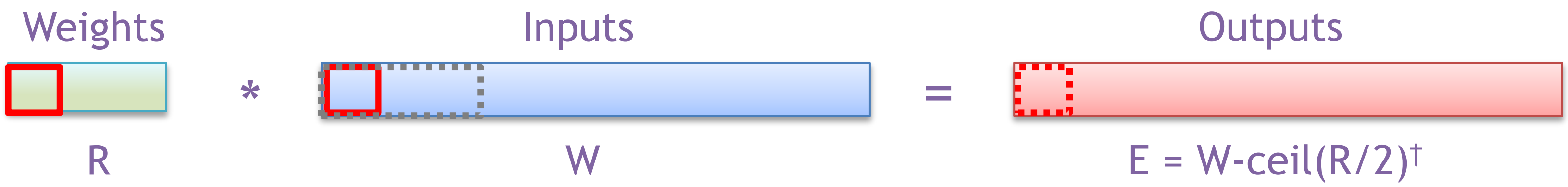


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
          w = q+s
          o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution

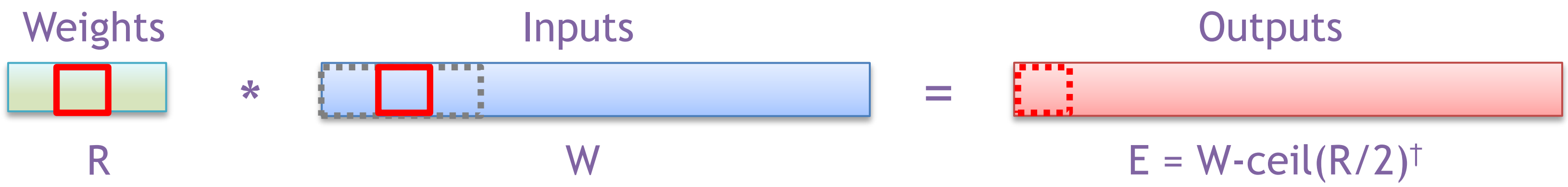


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution

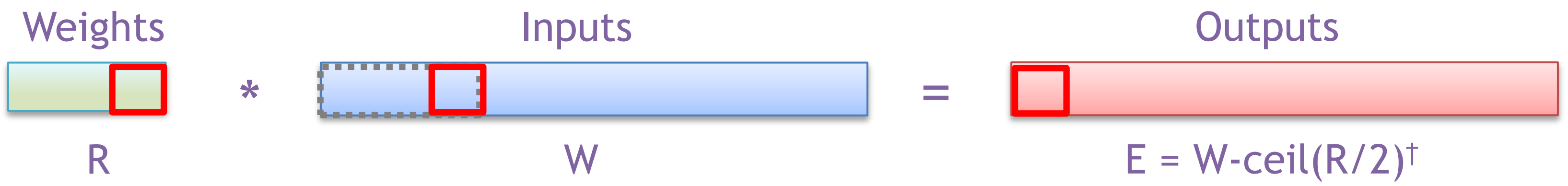


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution

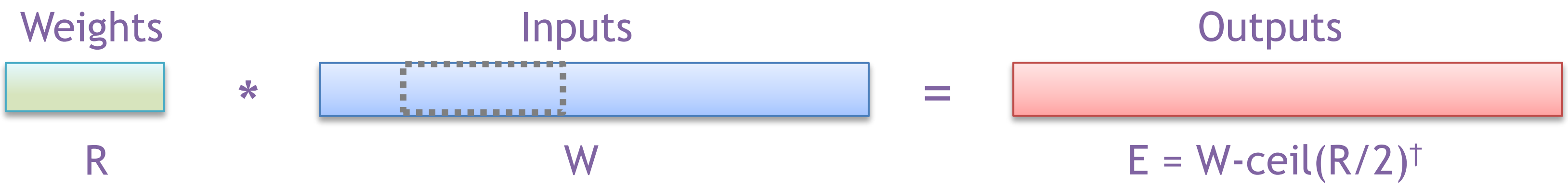


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution



Weights * Inputs = Outputs

R

W

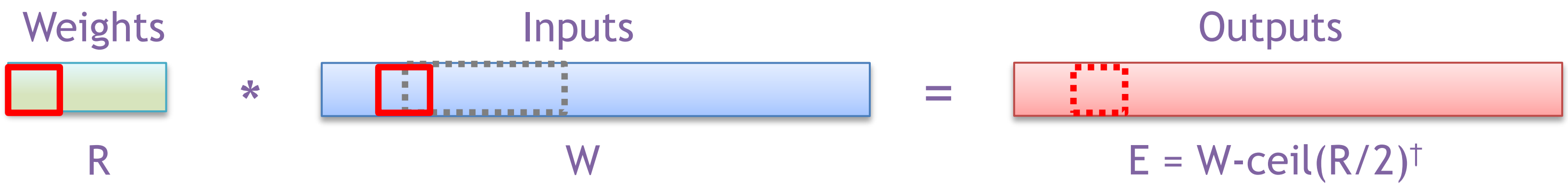
E = W-ceil(R/2)†

```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution

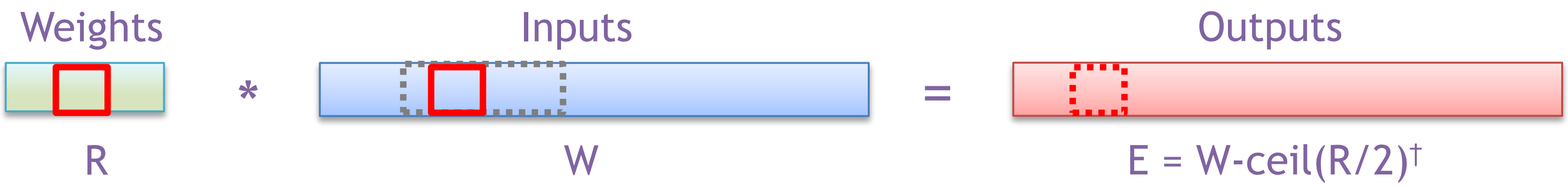


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
        w = q+s
        o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution

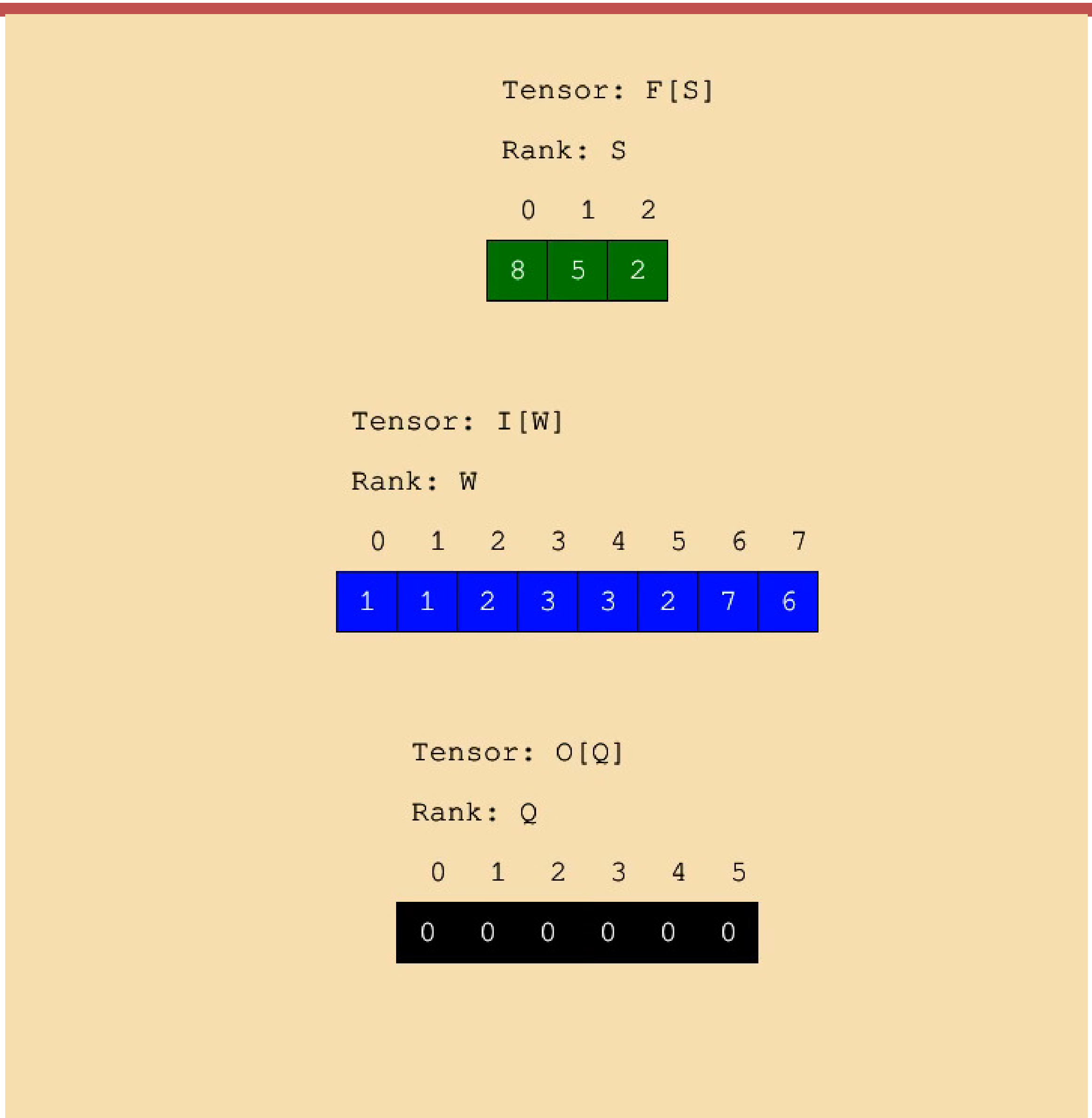


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0, Q):
    for s in [0, S):
          w = q+s
          o[q] += i[w]*f[s];
```

† Assuming: ‘valid’ style convolution

1-D Convolution - Movie



Tensor: F[S]

Rank: S

0	1	2
8	5	2

Tensor: I[W]

Rank: W

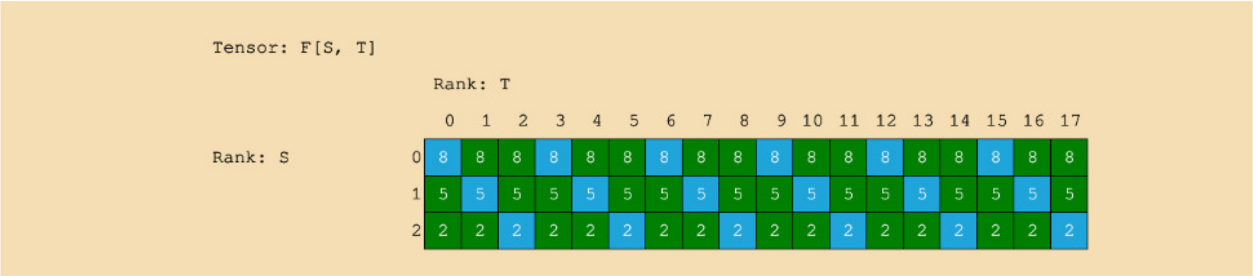
0	1	2	3	4	5	6	7
1	1	2	3	3	2	7	6

Tensor: O[Q]

Rank: Q

0	1	2	3	4	5
0	0	0	0	0	0

Output Stationary – Spacetime View

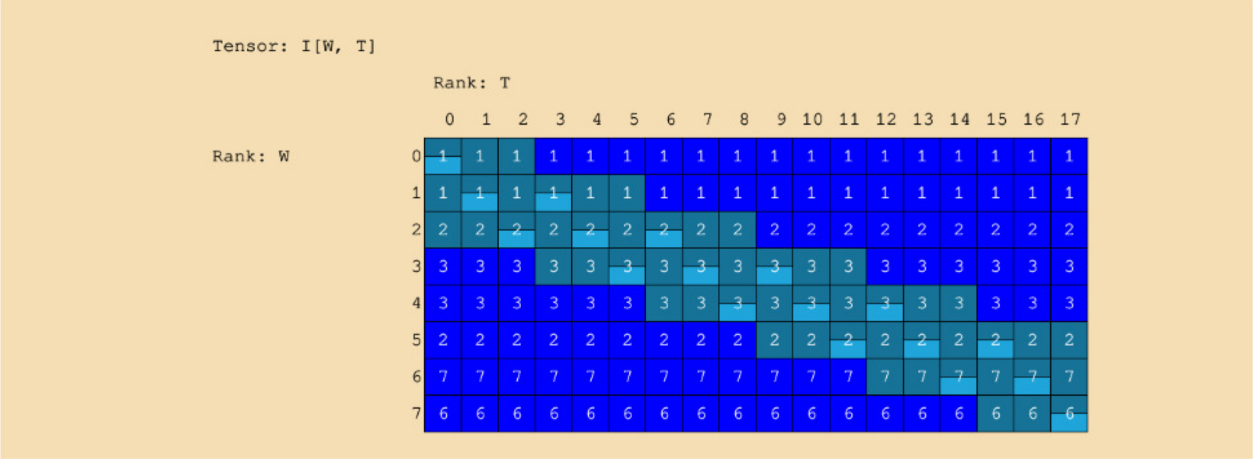


Tensor: F[S, T]

Rank: T

Rank: S

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
1	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

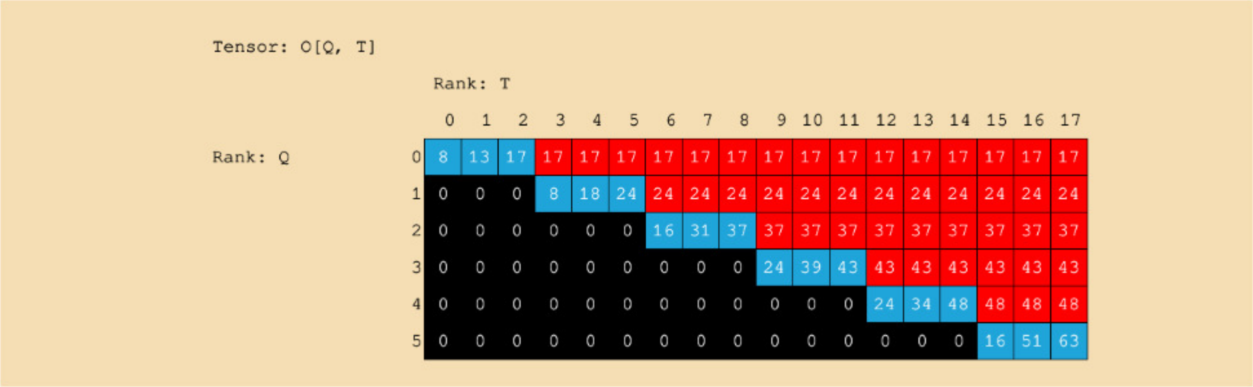


Tensor: I[W, T]

Rank: T

Rank: W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
5	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
7	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6



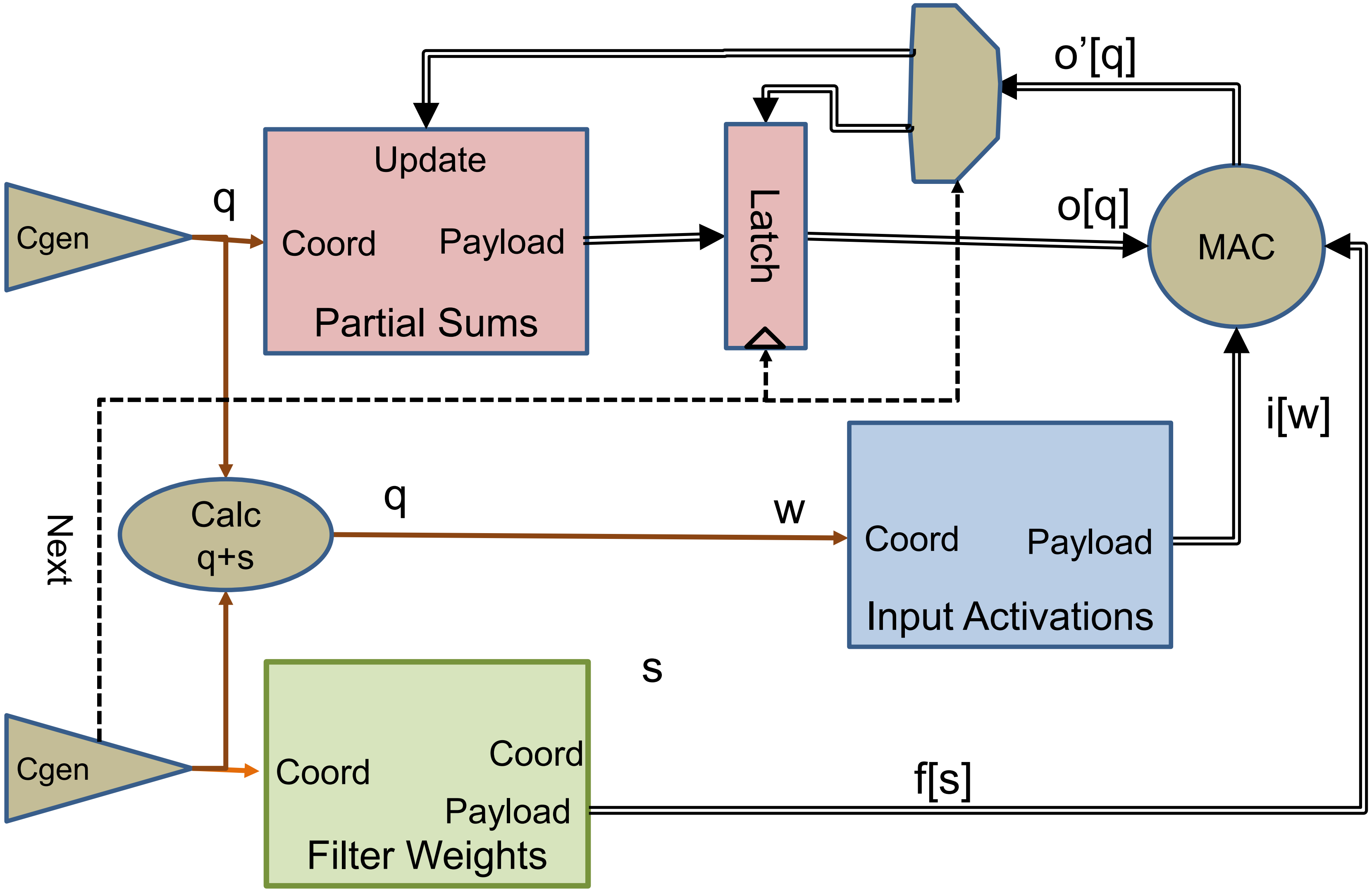
Tensor: O[Q, T]

Rank: T

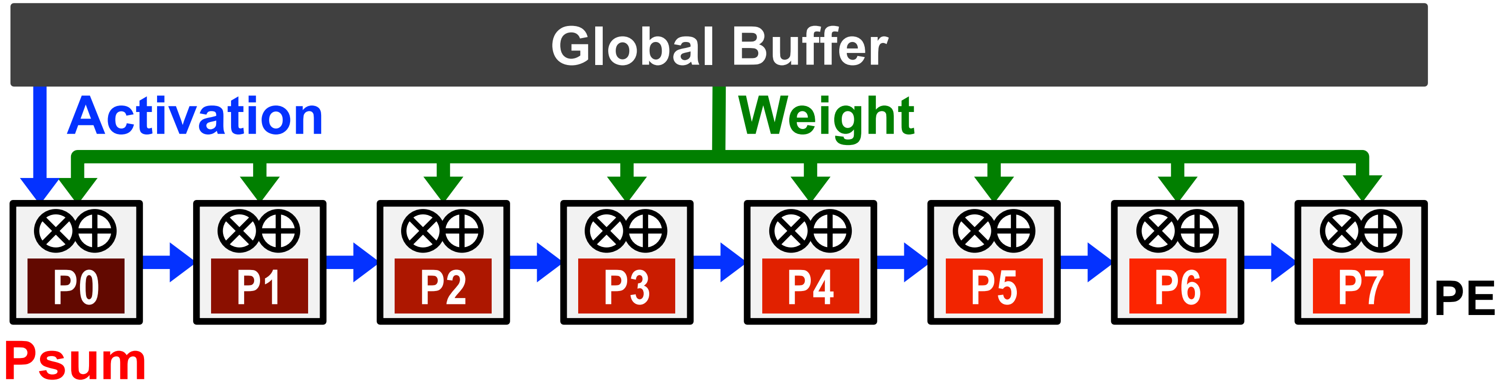
Rank: Q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	8	13	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17
1	0	0	0	8	18	24	24	24	24	24	24	24	24	24	24	24	24	24
2	0	0	0	0	0	0	16	31	37	37	37	37	37	37	37	37	37	37
3	0	0	0	0	0	0	0	0	0	24	39	43	43	43	43	43	43	43
4	0	0	0	0	0	0	0	0	0	0	0	0	24	34	48	48	48	48
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	51	63

1-D Output Stationary



Output Stationary (OS)

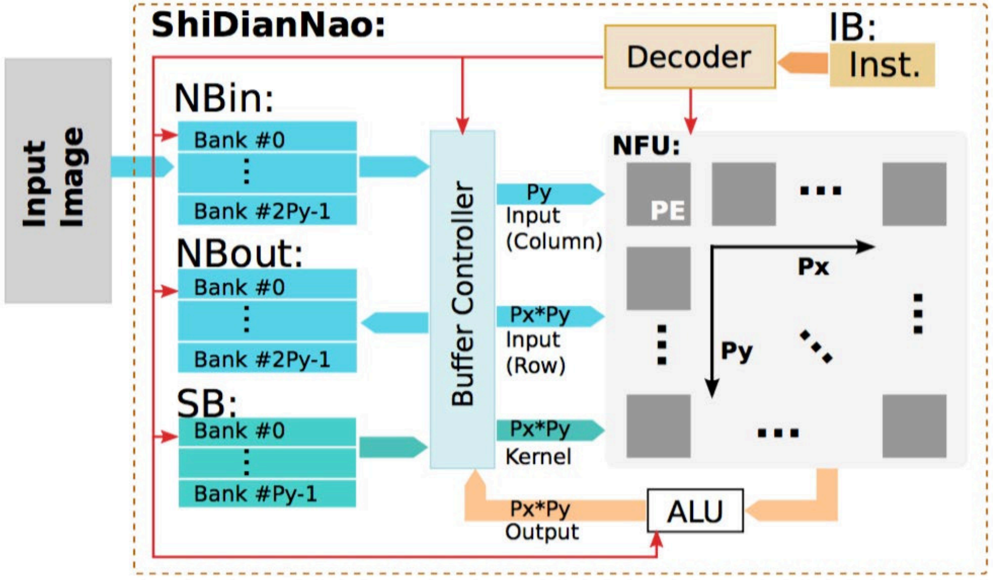


- **Minimize** **partial sum** R/W energy consumption
  - maximize local accumulation

- **Broadcast/Multicast** **filter weights** and **reuse** **activations** **spatially** across the PE array

OS Example: ShiDianNao

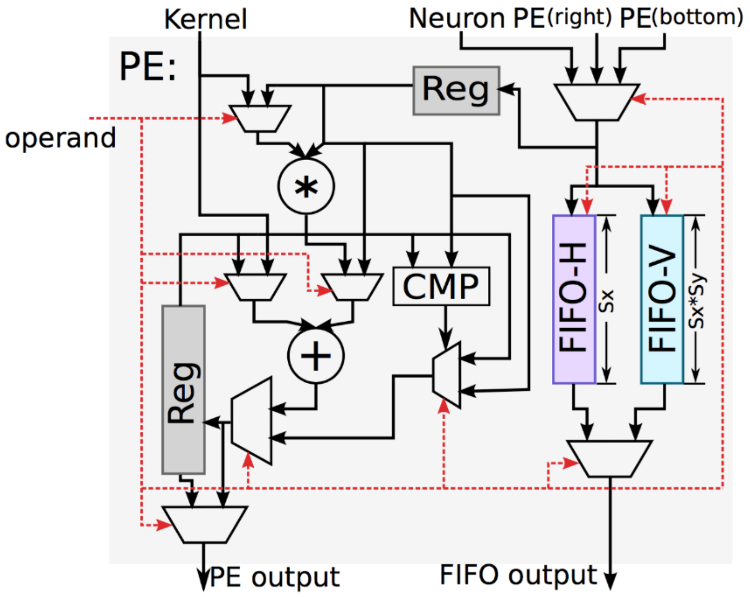
Top-Level Architecture



PE Architecture

weights

activations

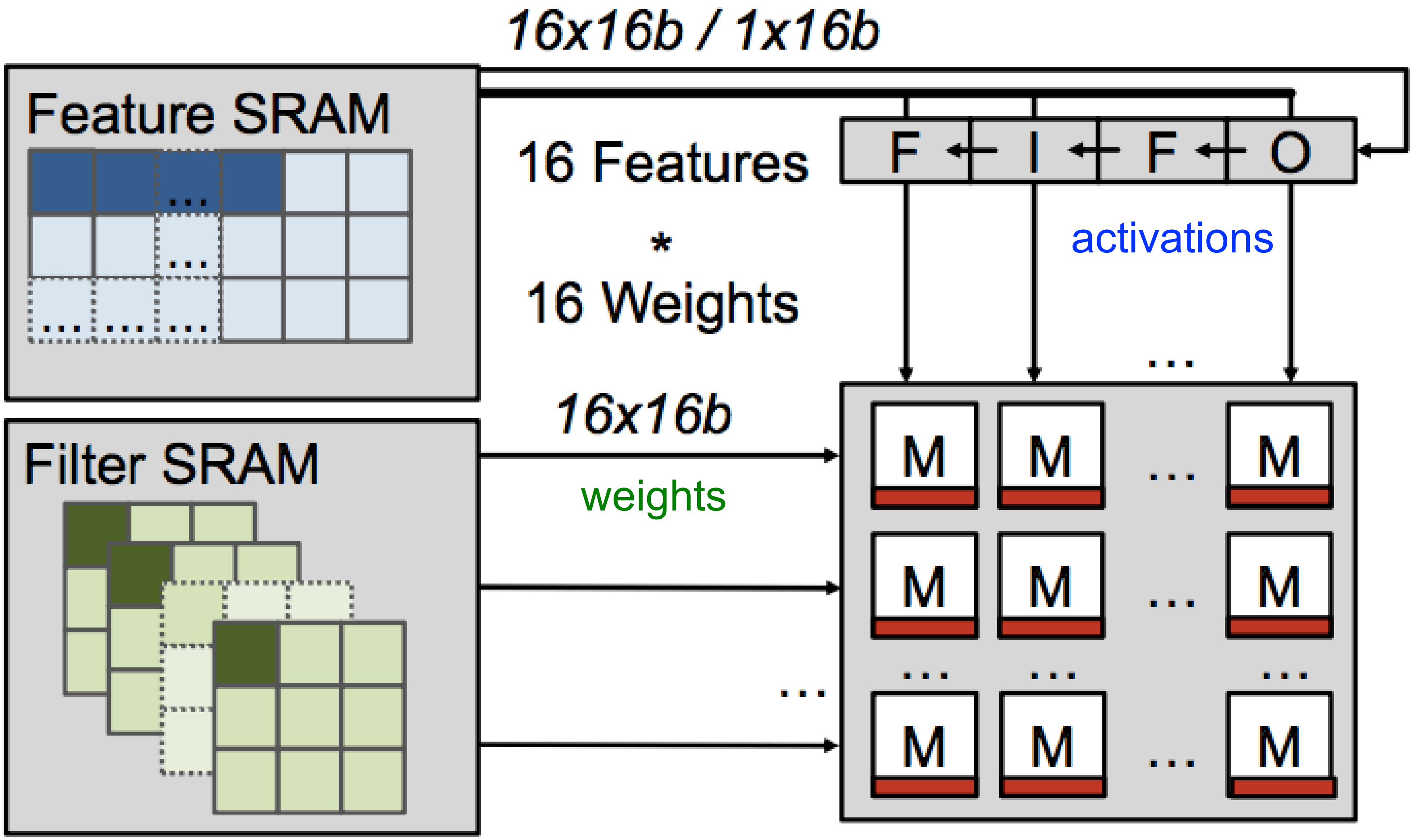


psums

- Inputs streamed through array
- Weights broadcast
- Partial sums accumulated in PE and streamed out

[Du et al., ISCA 2015]

OS Example: KU Leuven



[Moons et al., VLSI 2016, ISSCC 2017]

Many Dataflows

- **Output Stationary (OS)**

  [**Peemen**, *ICCD* 2013] [**ShiDianNao**, *ISCA* 2015]

  [**Gupta**, *ICML* 2015] [**Moons**, *VLSI* 2016] [**Thinker**, *VLSI* 2017]

- **Weight Stationary (WS)**

  [**Chakradhar**, *ISCA* 2010] [**nn-X (NeuFlow)**, *CVPRW* 2014]

  [**Park**, *ISSCC* 2015] [**ISAAC**, *ISCA* 2016] [**PRIME**, *ISCA* 2016]

  [**TPU**, *ISCA* 2017]

- **Input Stationary (IS)**

  [**Parashar (SCNN)**, ISCA 2017]

- **Row Stationary (IS)**

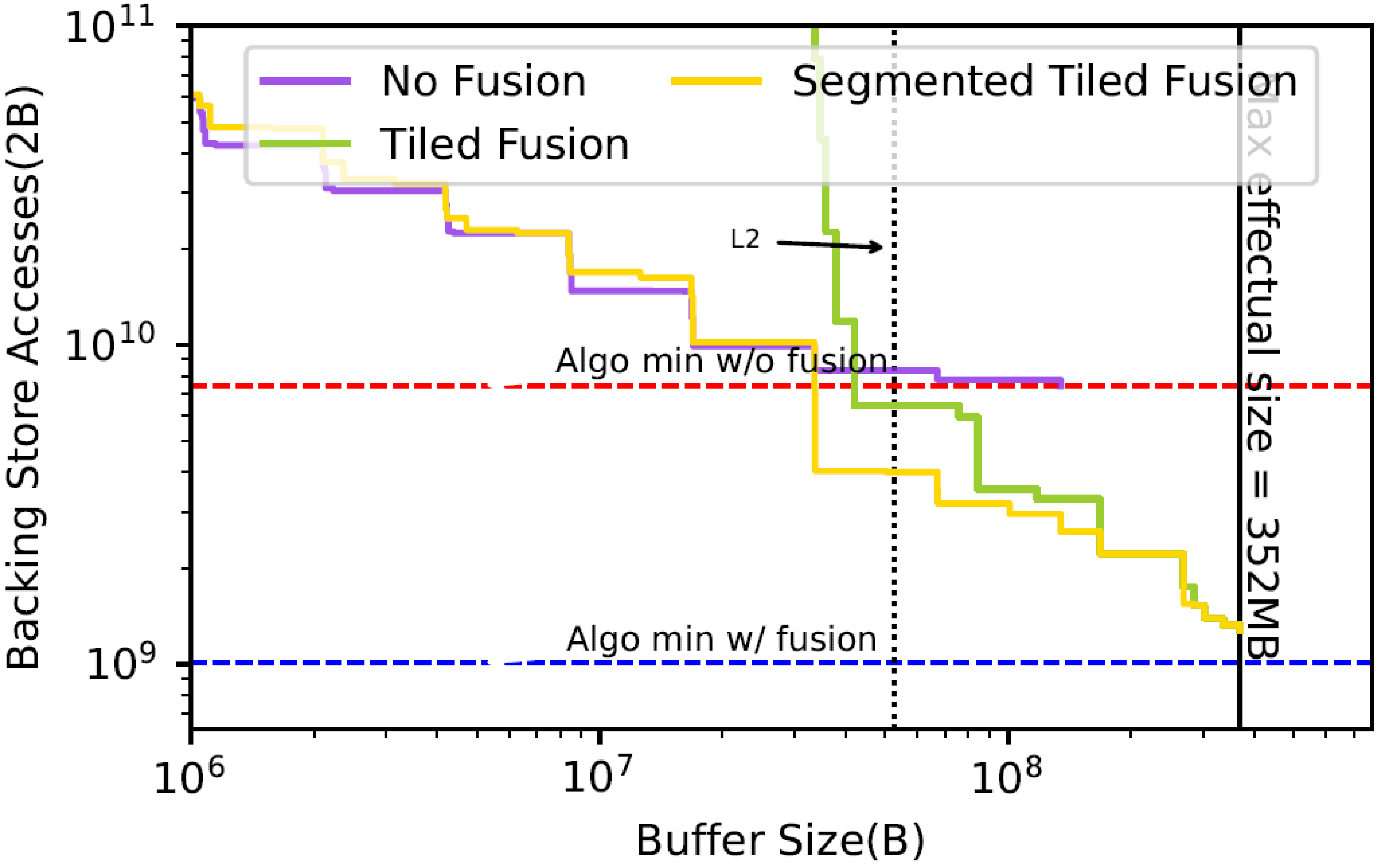
  [**Eyeriss**, ISCA 2016] [**Tetris** ASPLOS 2017] [**Eyeriss2**, JETCAT 2019]

Many Mapping Options

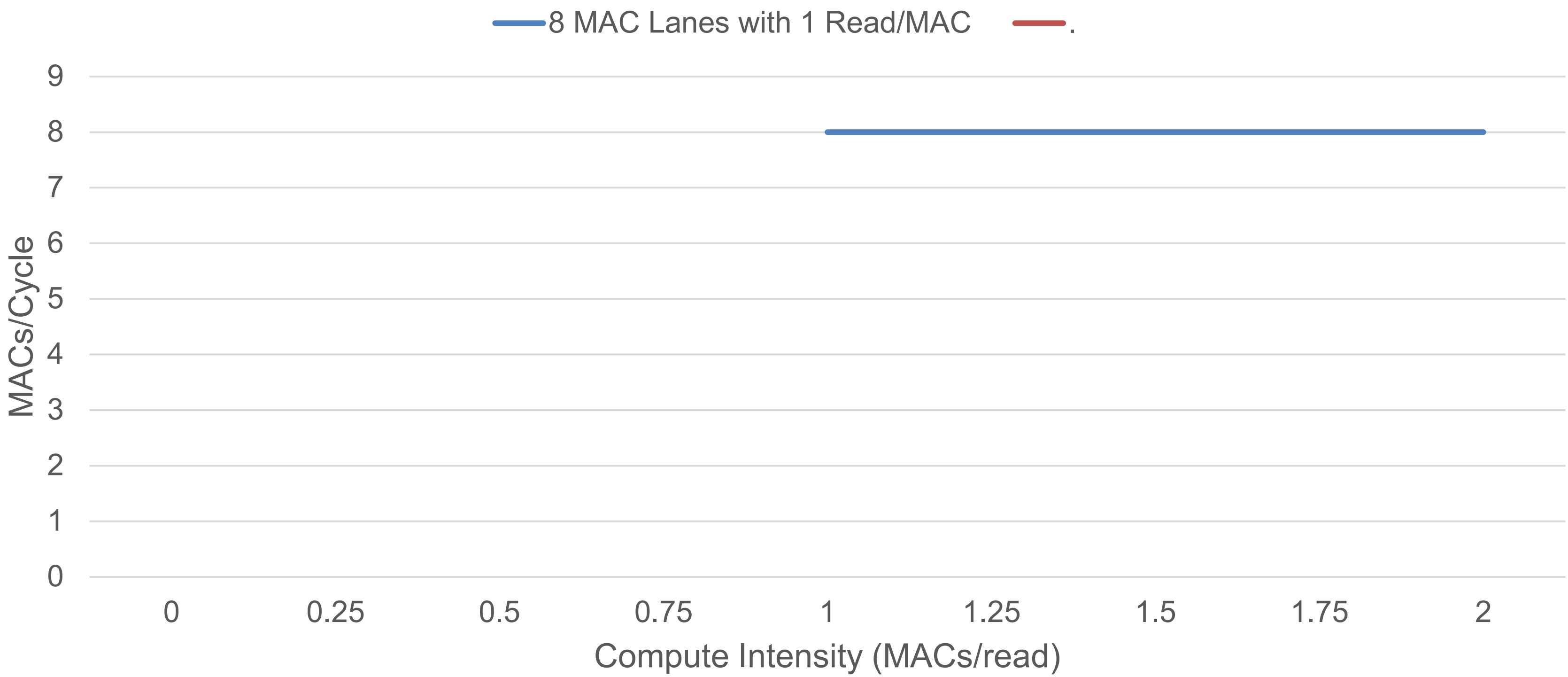
Per storage level cross product of:

- **Dataflow**
- **Tiling in time**
- **Tiling in space**
- **Bypassing**
- **Split/Shared storage**

Variation in Traffic with Mapping

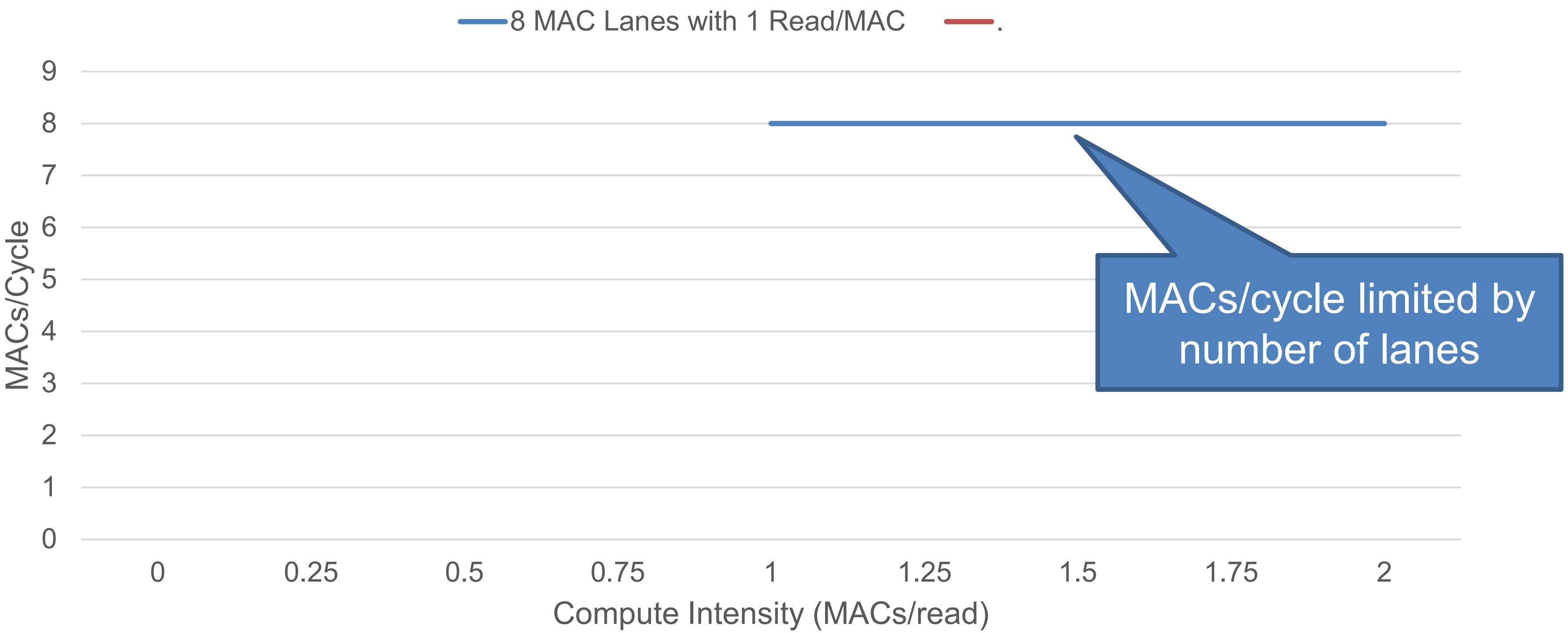


Roofline Model



Williams, Samuel, Andrew Waterman, and David Patterson. "Roofline: an insightful visual performance model for multicore architectures." Communications of the ACM 52.4 (2009): 65-76.

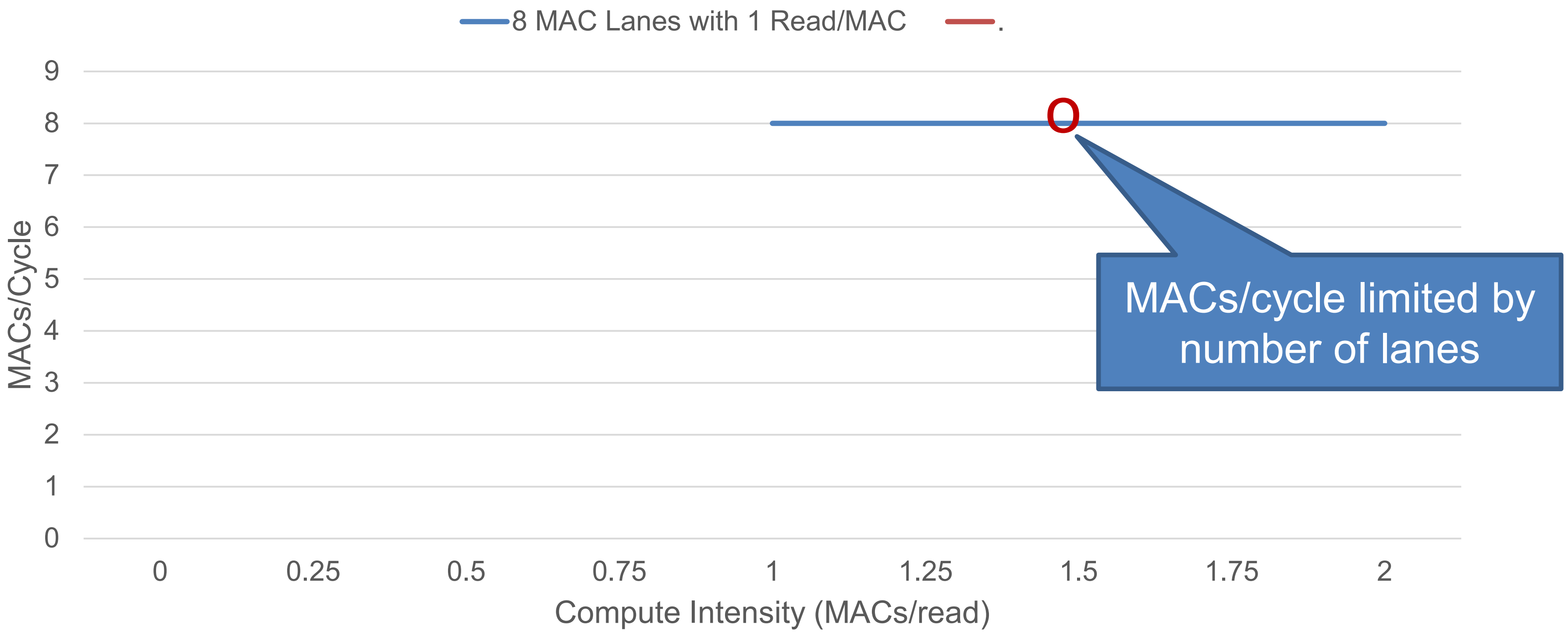
Roofline Model



MACs/cycle limited by number of lanes

Williams, Samuel, Andrew Waterman, and David Patterson. "Roofline: an insightful visual performance model for multicore architectures." Communications of the ACM 52.4 (2009): 65-76.

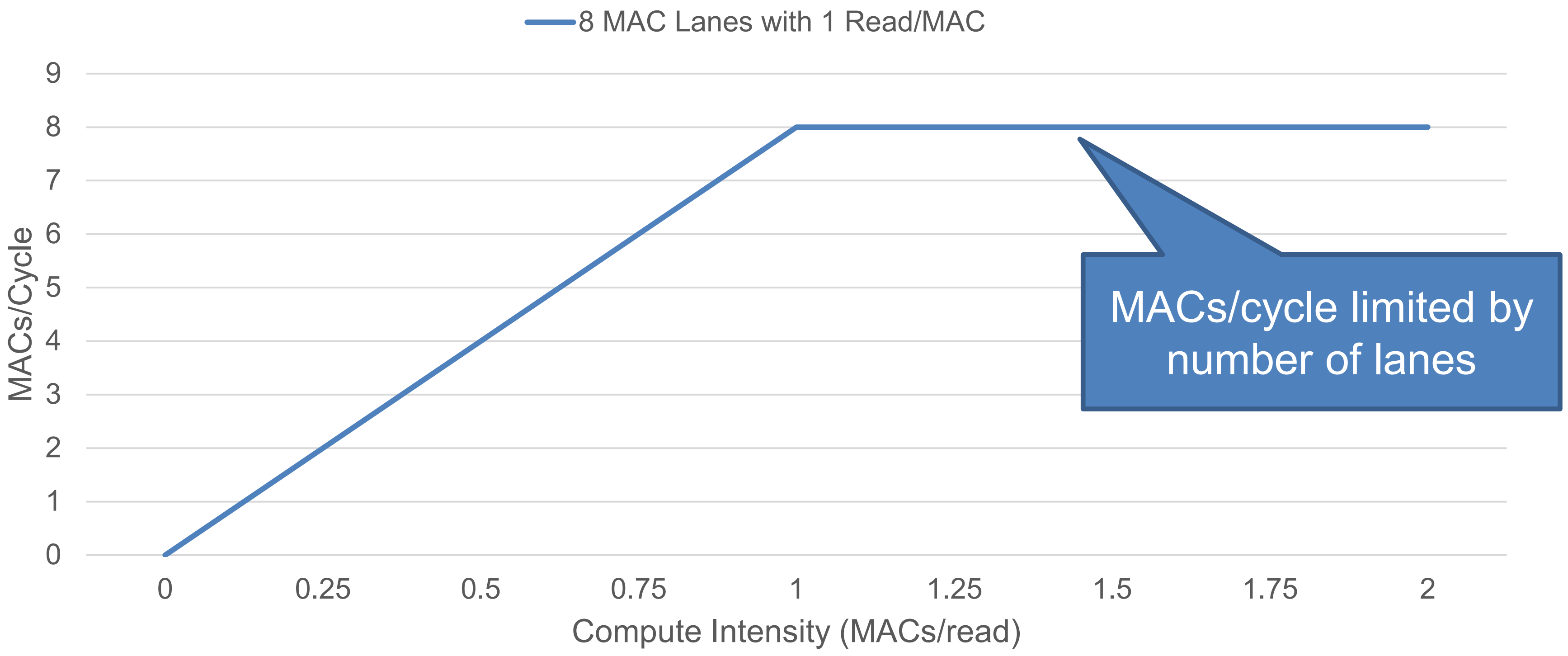
Roofline Model



MACs/cycle limited by number of lanes

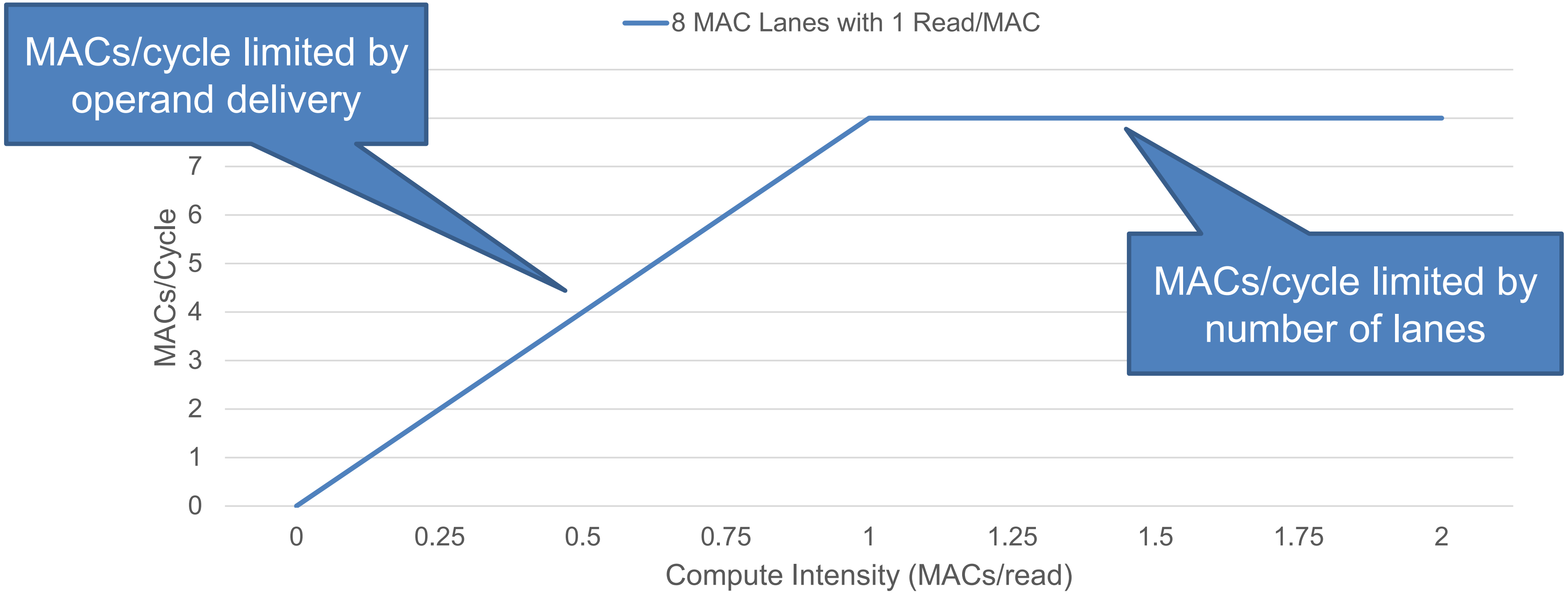
Williams, Samuel, Andrew Waterman, and David Patterson. "Roofline: an insightful visual performance model for multicore architectures." Communications of the ACM 52.4 (2009): 65-76.

Roofline Model



MACs/cycle limited by number of lanes

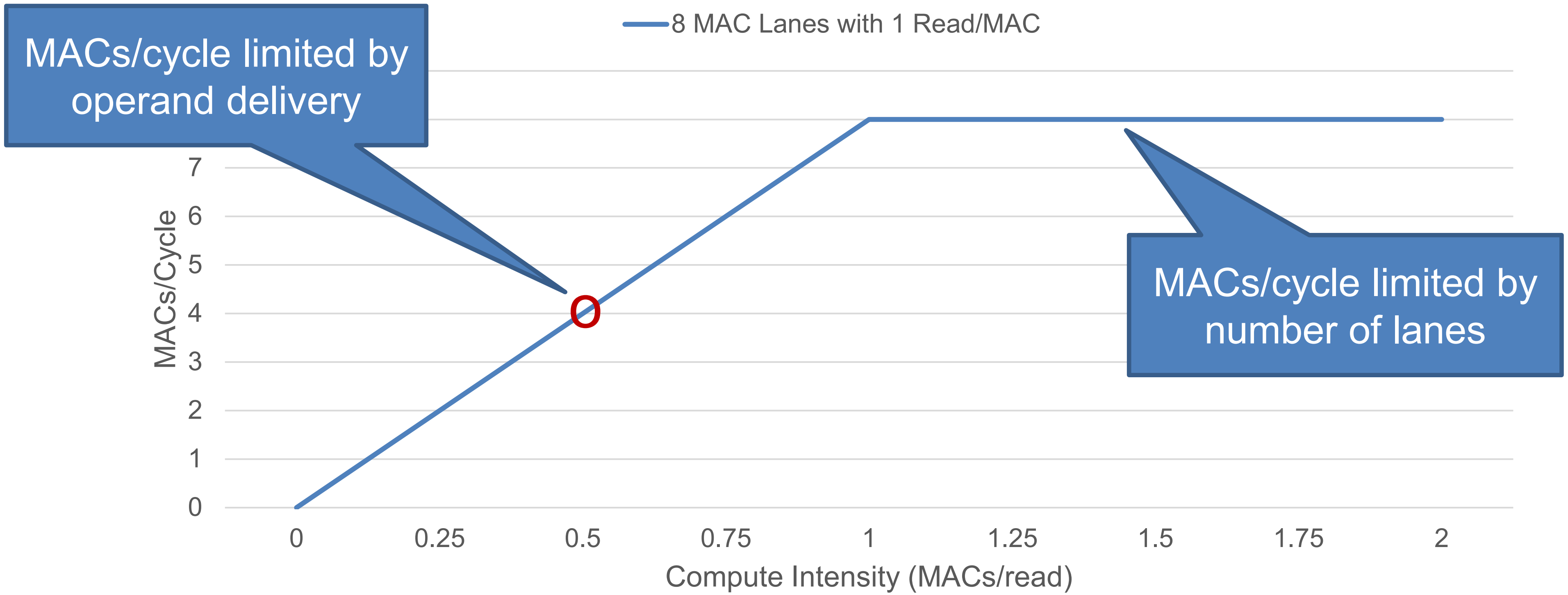
Roofline Model



MACs/cycle limited by operand delivery

MACs/cycle limited by number of lanes

Roofline Model



MACs/cycle limited by operand delivery

MACs/cycle limited by number of lanes

Thank you!

Next Lecture:
Accelerators (II)