

Accelerators-I

Tushar Krishna
Associate Professor @ Georgia Tech
Visiting Professor @ MIT EECS and CSAIL

Outline

- Why do we need accelerators?
- Why now?
- How to design accelerators

Outline

- Why do we need accelerators?
- Why now?
- How to design accelerators

Power Constraints in Modern Computers



<< 1W



~ 1W



~ 15W



~ 50W



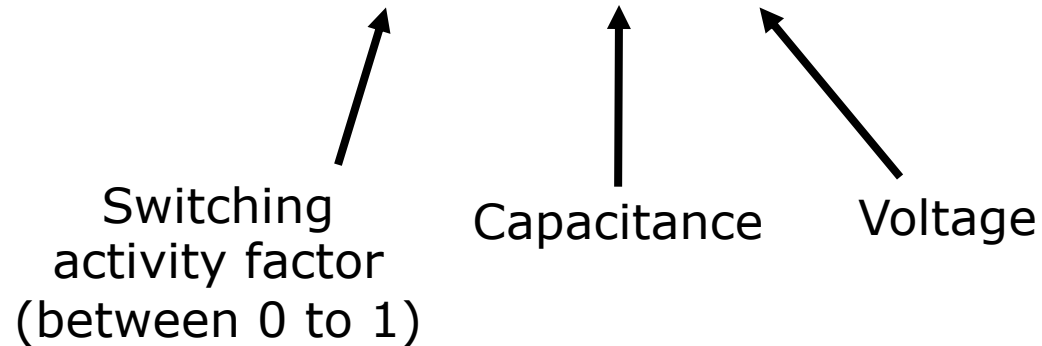
~ 100W



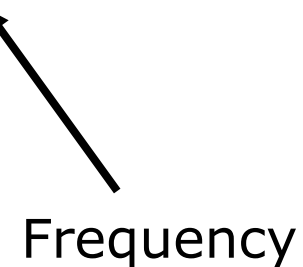
~ 100W

Energy and Power Consumption

- Energy Consumption = $a \times C \times V^2$



- Power Consumption = $a \times C \times V^2 \times f$



CMOS Scaling (idealized)


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

CMOS Scaling (idealized)

Gen X

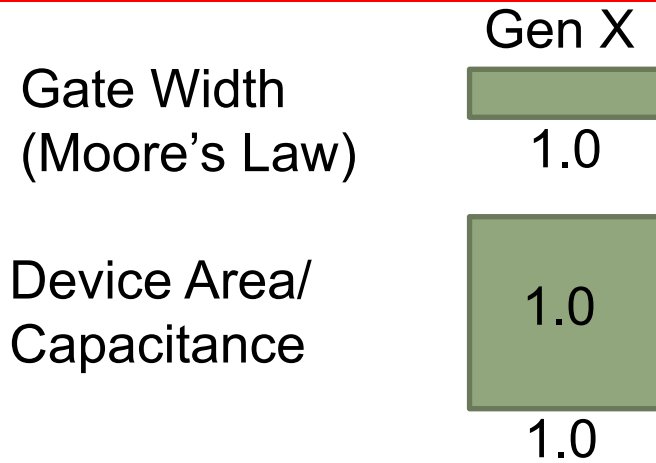
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | |
|-----------------------------|--|
| | Gen X |
| Gate Width (Moore's Law) |  1.0 |



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

CMOS Scaling (idealized)





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

CMOS Scaling (idealized)

| | Gen X |
|-----------------------------|--|
| Gate Width (Moore's Law) |  1.0 |
| Device Area/ Capacitance |  1.0 |
| Voltage (Dennard's) | 1.0 1.0 |



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

CMOS Scaling (idealized)

| | Gen X |
|-----------------------------|--|
| Gate Width (Moore's Law) |  1.0 |
| Device Area/ Capacitance |  1.0 |
| Voltage (Dennard's) | 1.0 |
| Energy | $\sim \underline{1.0} \times \underline{1.0}^2 = 1.0$ |



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

CMOS Scaling (idealized)

| | Gen X |
|-----------------------------|--|
| Gate Width (Moore's Law) |  1.0 |
| Device Area/ Capacitance |  1.0 |
| Voltage (Dennard's) | 1.0 |
| Energy | $\sim \underline{1.0} \times \underline{1.0}^2 = 1.0$ |
| Delay | 1.0 |



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

CMOS Scaling (idealized)

| | Gen X |
|-----------------------------|--|
| Gate Width (Moore's Law) |  1.0 |
| Device Area/ Capacitance |  1.0 |
| Voltage (Dennard's) | 1.0 |
| Energy | $\sim \underline{1.0} \times \underline{1.0}^2 = 1.0$ |
| Delay | 1.0 |
| Frequency | $1/\underline{1.0} = \underline{1.0}$ |

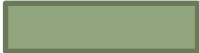

[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X |
|-----------------------------|--|
| Gate Width (Moore's Law) |  1.0 |
| Device Area/ Capacitance |  1.0 |
| Voltage (Dennard's) | 1.0 |
| Energy | $\sim \underline{1.0} \times \underline{1.0}^2 = 1.0$ |
| Delay | 1.0 |
| Frequency | $1/\underline{1.0} = \underline{1.0}$ |
| Power | $\sim \underline{1.0} \times \underline{1.0}^2 \times \underline{1.0} = 1.0$ |




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

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---------|
| Gate Width (Moore's Law) |  1.0 | |
| Device Area/ Capacitance |  1.0 | |
| Voltage (Dennard's) | 1.0 | |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | |
| Delay | 1.0 | |
| Frequency | $1/1.0 = 1.0$ | |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |




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

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|--|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 | |
| Voltage (Dennard's) | 1.0 | |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | |
| Delay | 1.0 | |
| Frequency | $1/1.0 = 1.0$ | |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |

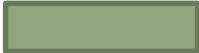
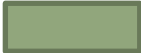


[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|--|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.7 |
| Voltage (Dennard's) | 1.0 | |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | |
| Delay | 1.0 | |
| Frequency | $1/1.0 = 1.0$ | |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |





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

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | 0.7 |
| Delay | 1.0 | 0.7 |
| Frequency | $1/1.0 = 1.0$ | 0.7 |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | 0.7 |




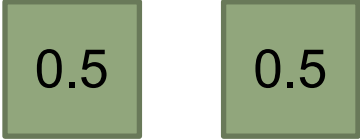
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim \underline{1.0} \times \underline{1.0^2} = 1.0$ | |
| Delay | 1.0 | |
| Frequency | $1/\underline{1.0} = 1.0$ | |
| Power | $\sim \underline{1.0} \times \underline{1.0^2} \times \underline{1.0} = 1.0$ | |

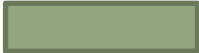
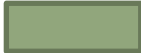

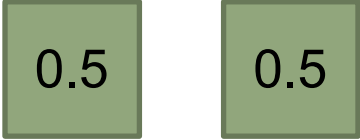
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | $\sim 2 \times 0.7 \times 0.7^2 = 0.65$ |
| Delay | 1.0 | |
| Frequency | $1/1.0 = 1.0$ | |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |




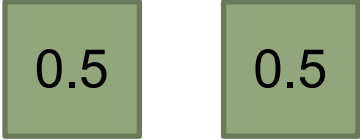
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | $\sim 2 \times 0.7 \times 0.7^2 = 0.65$ |
| Delay | 1.0 | 0.7 |
| Frequency | $1/1.0 = 1.0$ | |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |




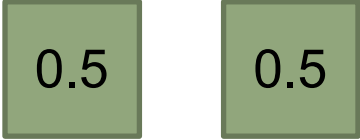
[Dennard et al., "Design of ion-implanted MOSFET's with very small physical dimensions", JSSC 1974]

CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | $\sim 2 \times 0.7 \times 0.7^2 = 0.65$ |
| Delay | 1.0 | 0.7 |
| Frequency | $1/1.0 = 1.0$ | $1/0.7 = 1.4$ |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | |

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

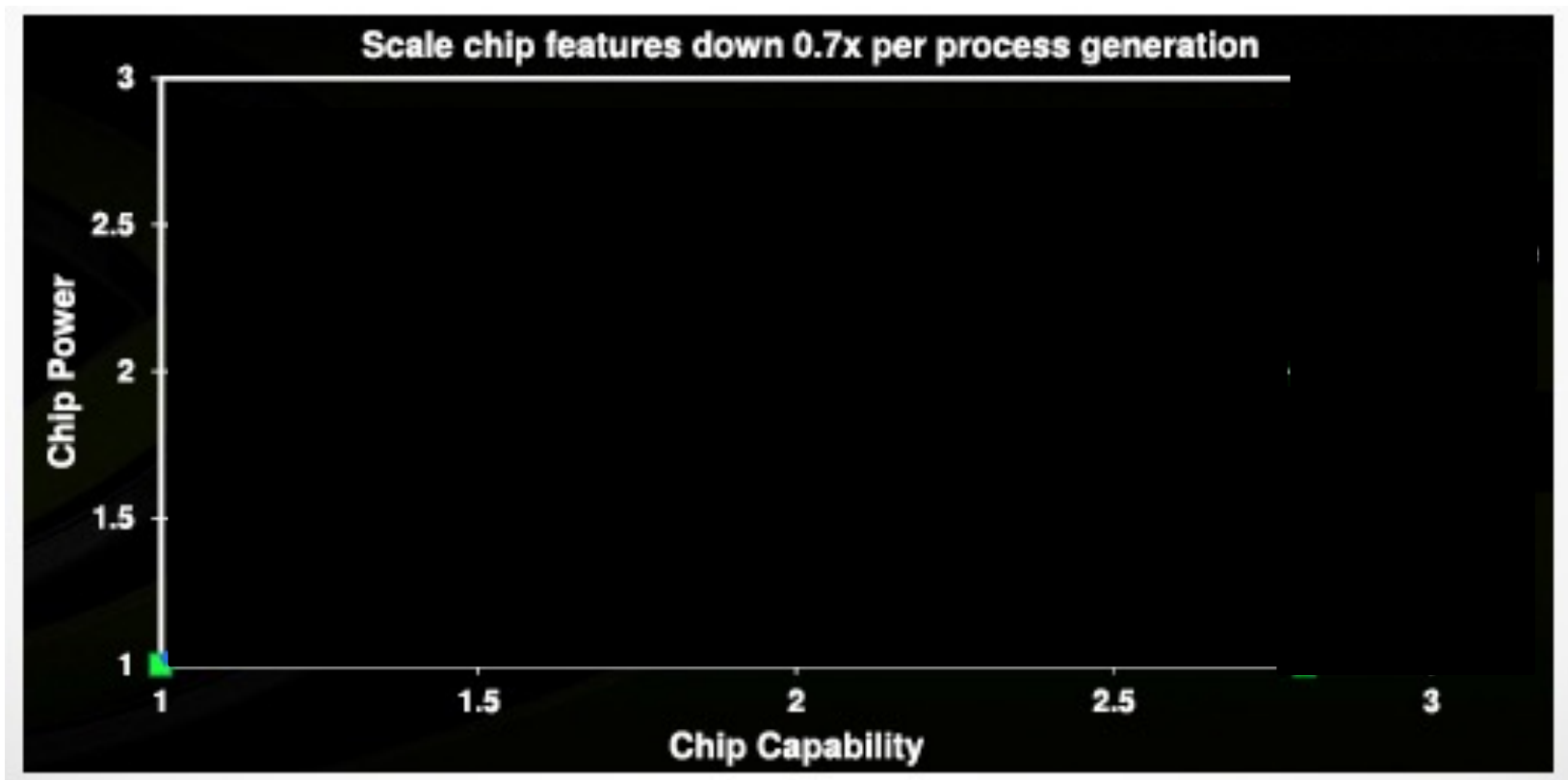
CMOS Scaling (idealized)

| | Gen X | Gen X+1 |
|-----------------------------|--|---|
| Gate Width (Moore's Law) |  1.0 |  0.7 |
| Device Area/ Capacitance |  1.0 |  0.5 0.5 |
| Voltage (Dennard's) | 1.0 | 0.7 |
| Energy | $\sim 1.0 \times 1.0^2 = 1.0$ | $\sim 2 \times 0.7 \times 0.7^2 = 0.65$ |
| Delay | 1.0 | 0.7 |
| Frequency | $1/1.0 = 1.0$ | $1/0.7 = 1.4$ |
| Power | $\sim 1.0 \times 1.0^2 \times 1.0 = 1.0$ | $\sim 2 \times 0.7 \times 0.7^2 \times 1.4 = 1.0$ |

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

CMOS Scaling (idealized)

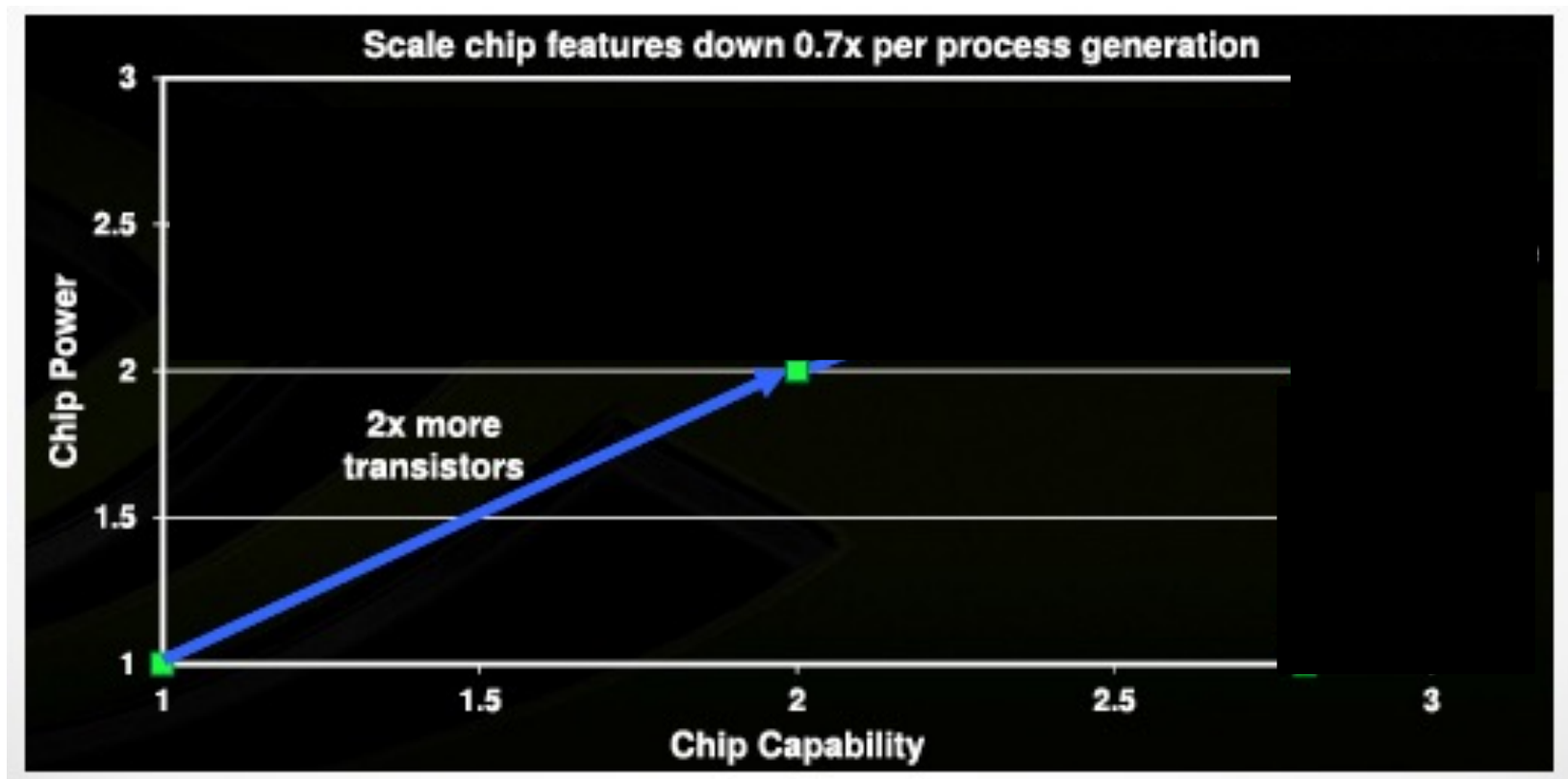
- Moore's Law (transistors) + Dennard's Scaling (voltage)



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012"

CMOS Scaling (idealized)

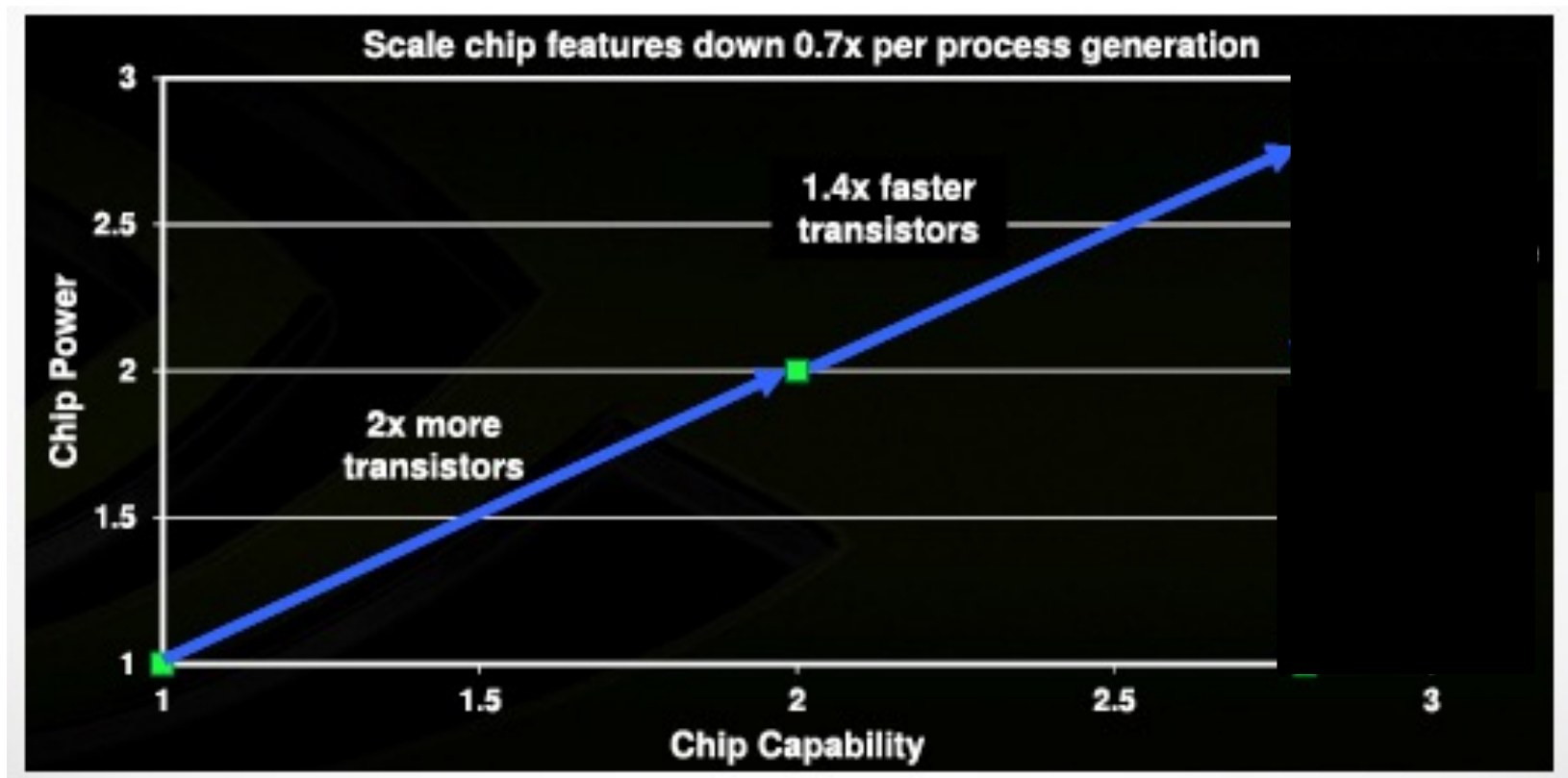
- Moore's Law (transistors) + Dennard's Scaling (voltage)



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012

CMOS Scaling (idealized)

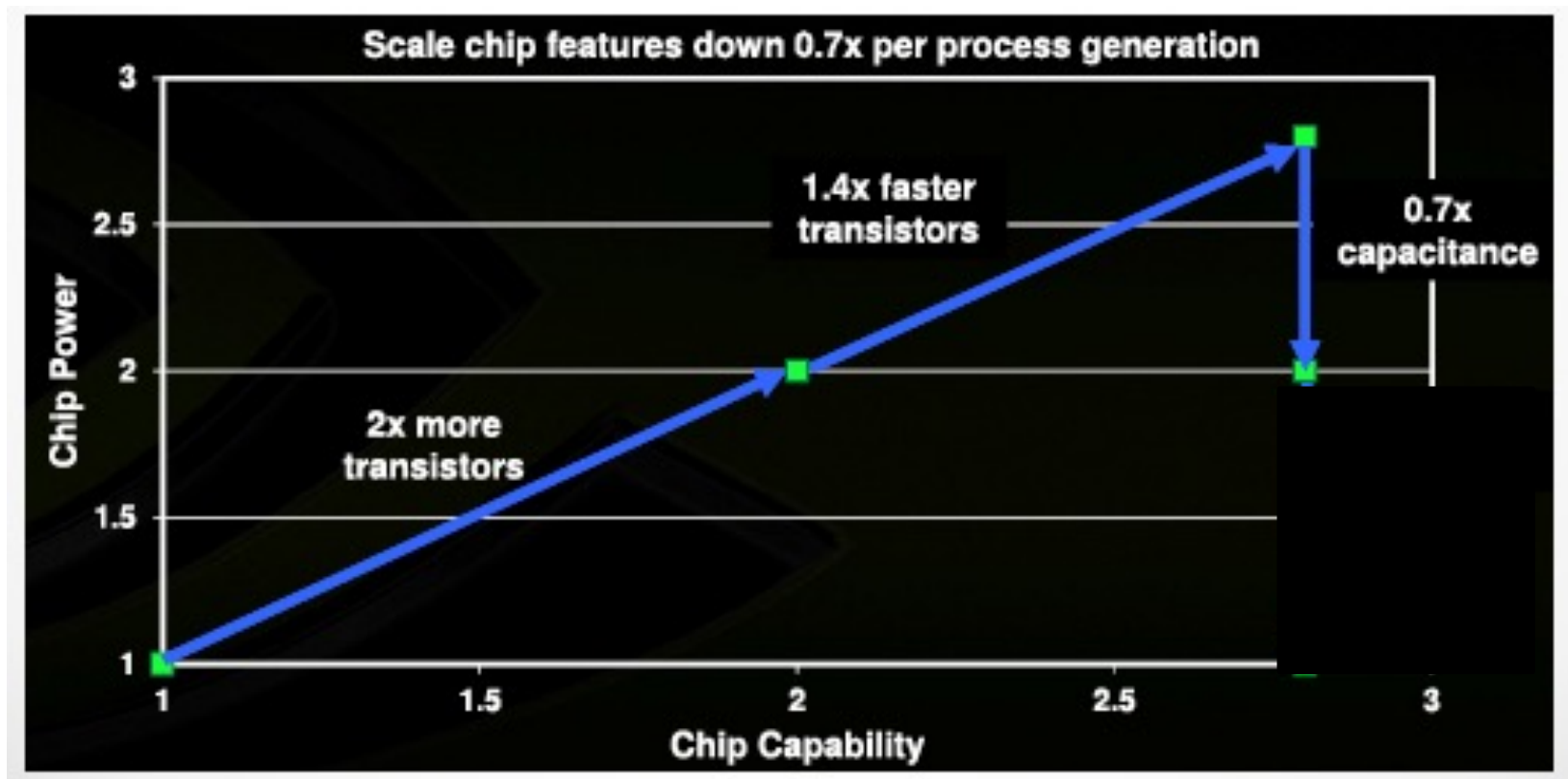
- Moore's Law (transistors) + Dennard's Scaling (voltage)



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012

CMOS Scaling (idealized)

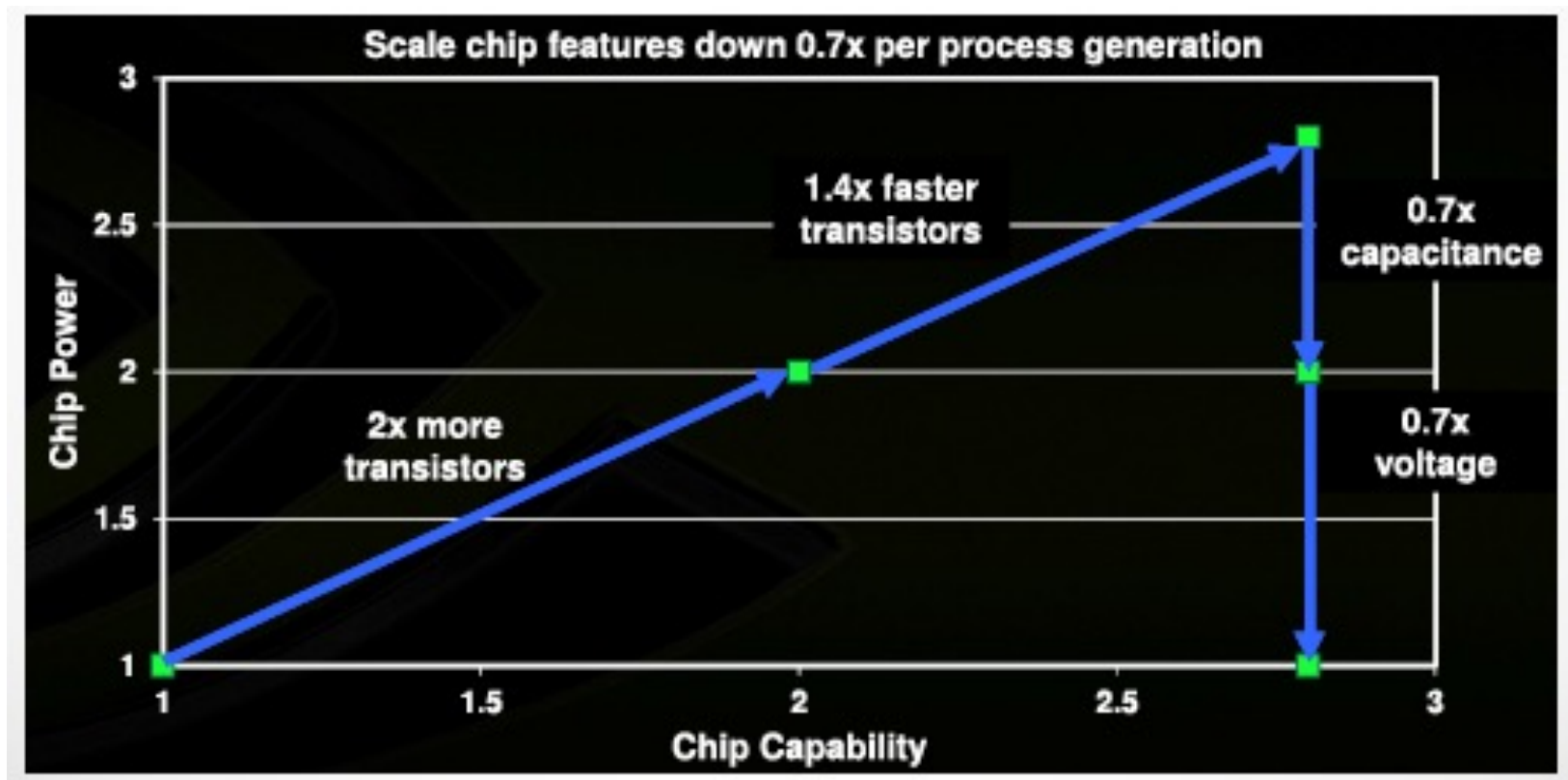
- Moore's Law (transistors) + Dennard's Scaling (voltage)



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012

CMOS Scaling (idealized)

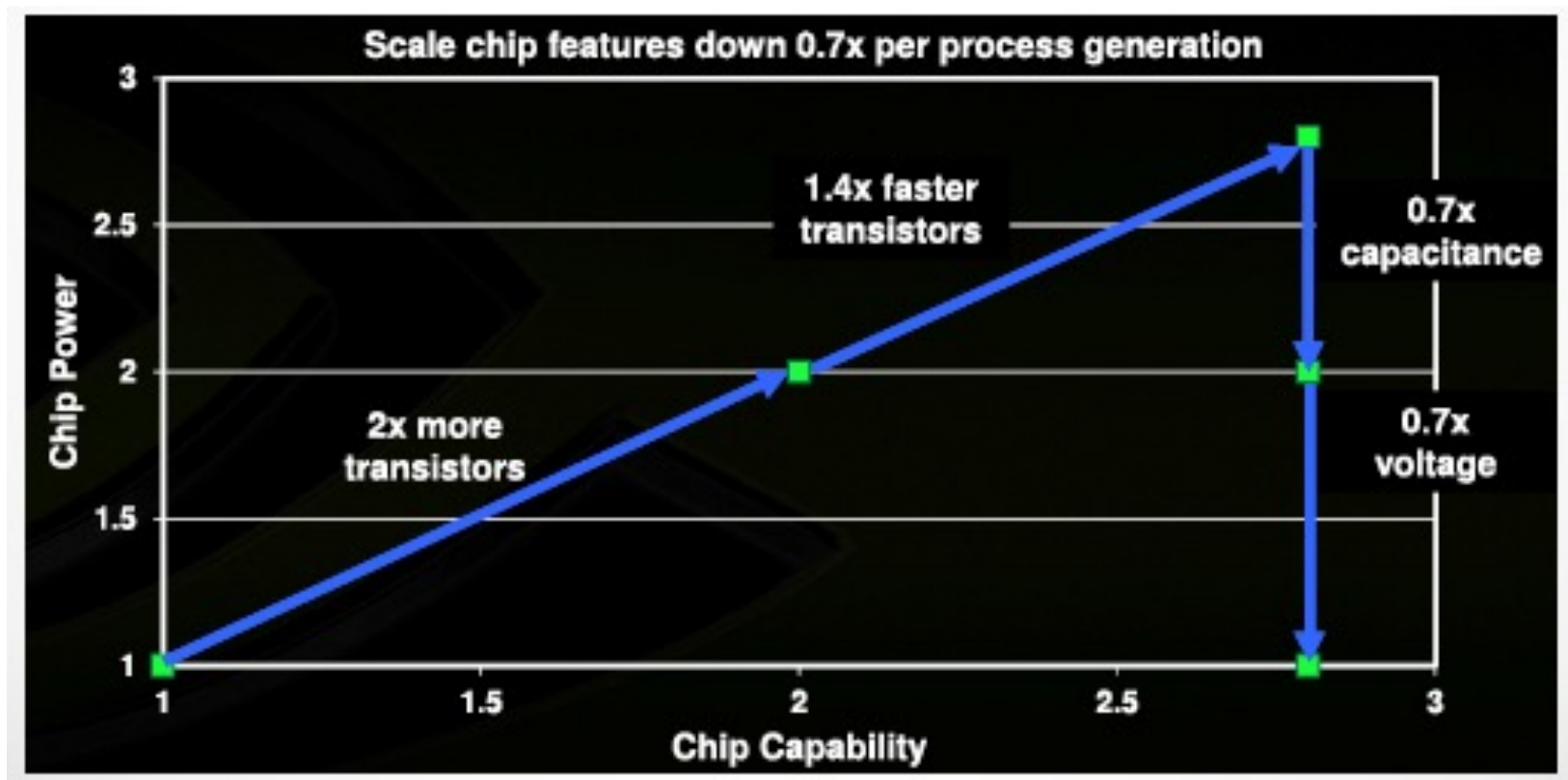
- Moore's Law (transistors) + Dennard's Scaling (voltage)



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012"

CMOS Scaling (idealized)

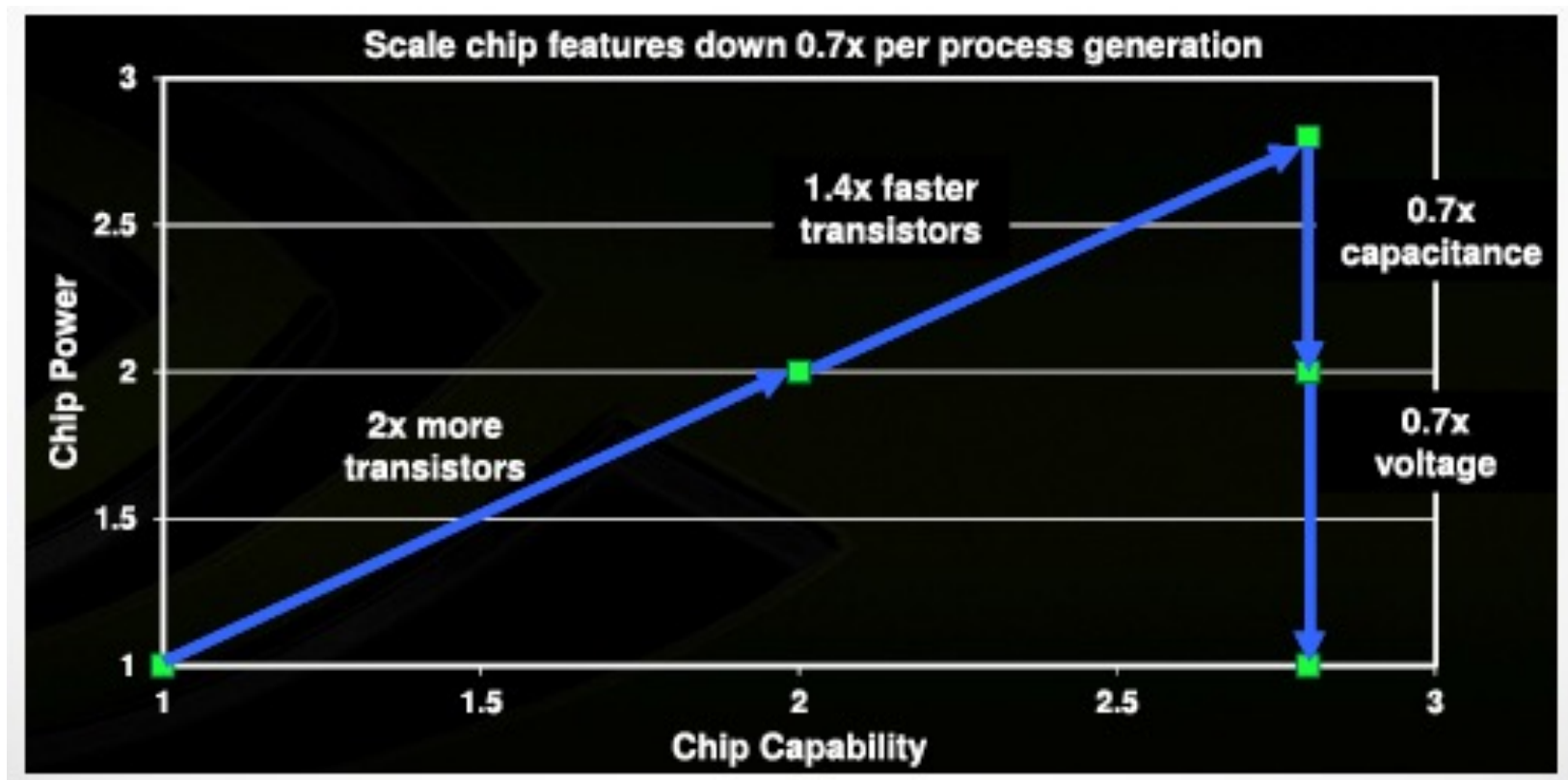
- Moore's Law (transistors) + Dennard's Scaling (voltage)
 - 2.8X in chip capability per generation at constant power



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012"

CMOS Scaling (idealized)

- Moore's Law (transistors) + Dennard's Scaling (voltage)
 - 2.8X in chip capability per generation at constant power
 - ~5000x performance improvement in 20 years



Source: "Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012"

“Power Wall”

- Dennard’s Scaling has stopped
 - Why?

“Power Wall”

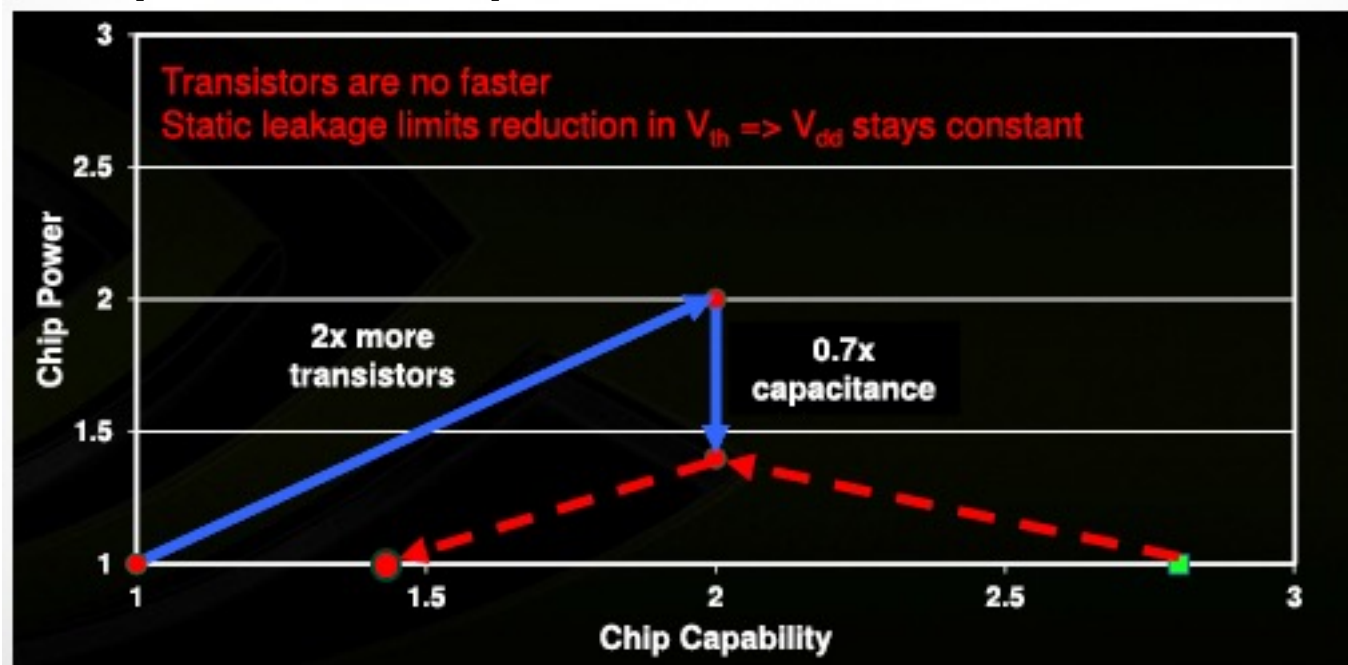
- Dennard’s Scaling has stopped
 - Why? **Already operating close to $V_{\text{threshold}}$**

“Power Wall”

- Dennard’s Scaling has stopped
 - Why? **Already operating close to $V_{\text{threshold}}$**
 - Cannot increase operating frequency
($P = \frac{1}{2} CV^2$)

“Power Wall”

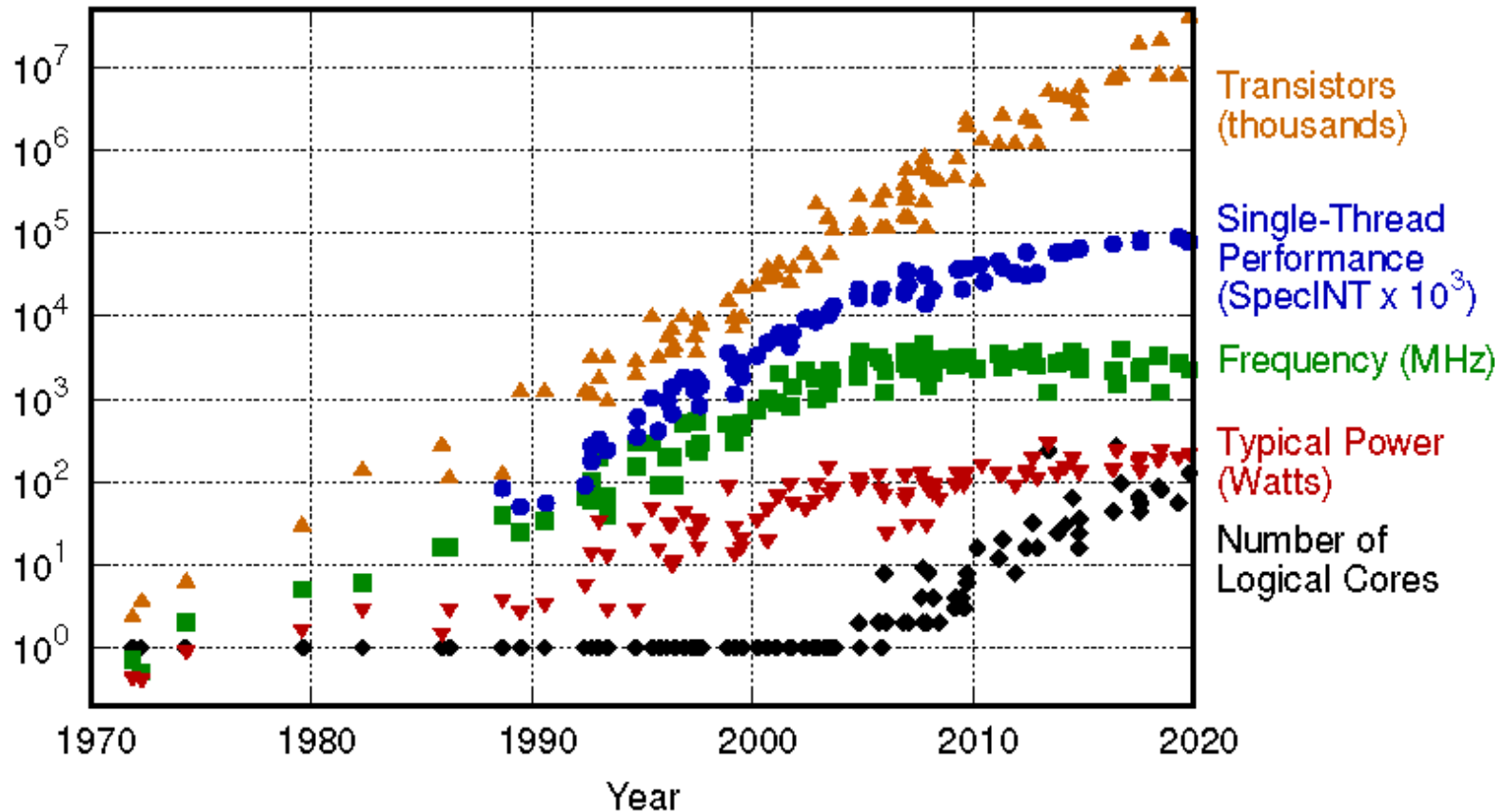
- Dennard’s Scaling has stopped
 - Why? **Already operating close to $V_{\text{threshold}}$**
 - Cannot increase operating frequency
($P = \frac{1}{2} CV^2$)



Source: “Advancing Computer Systems Without Technology Progress, ISAT Outbrief, 2012

Technology Trends

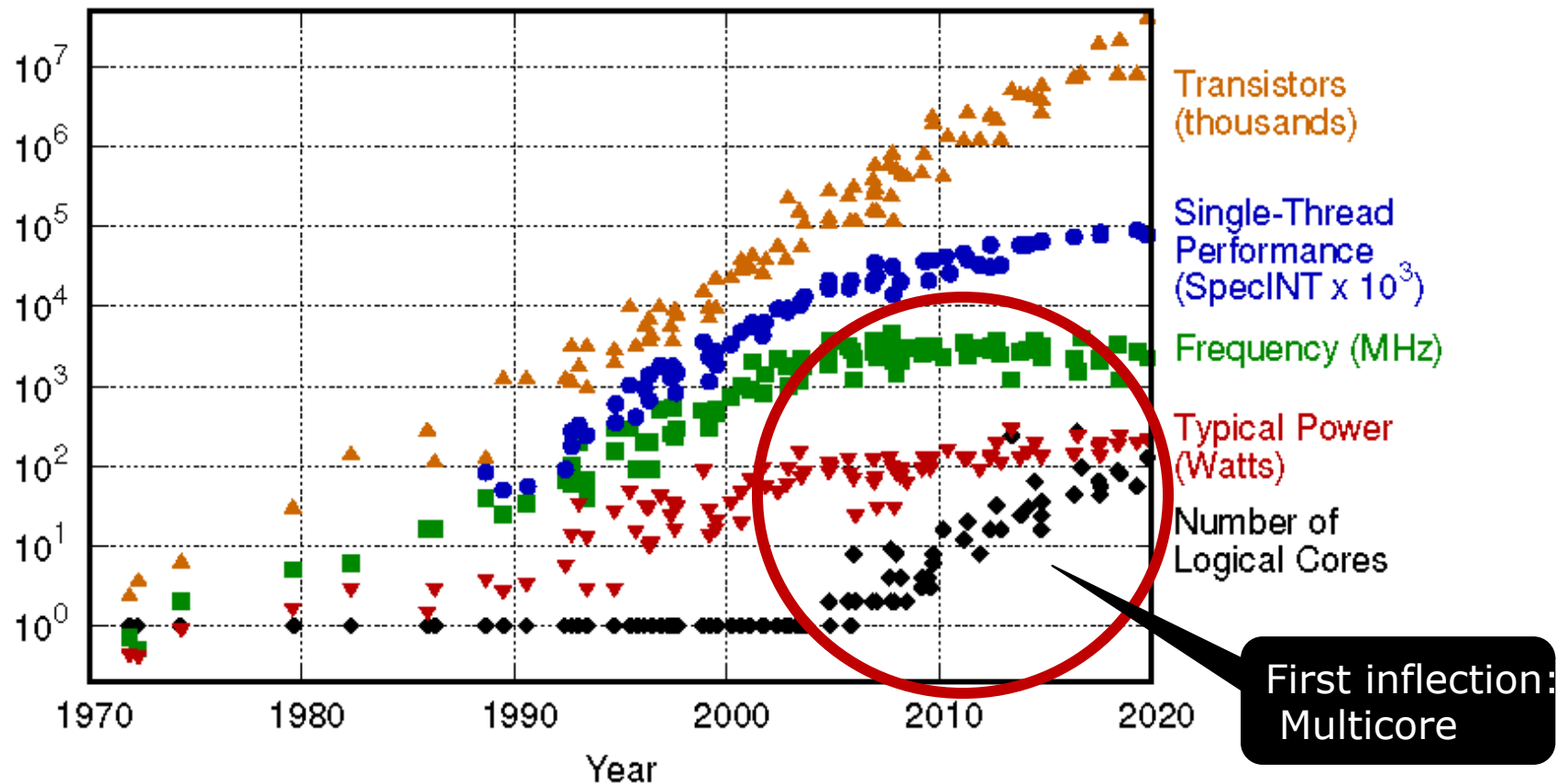
48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Technology Trends

48 Years of Microprocessor Trend Data



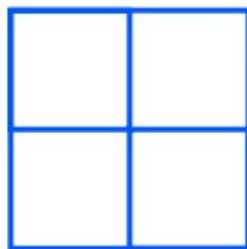
Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Utilization Wall ("Dark Silicon")

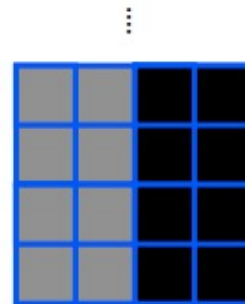
Spectrum of tradeoffs
between # of cores and
frequency

Example:
65 nm \rightarrow 32 nm ($S = 2$)

4 cores @ 1.8 GHz

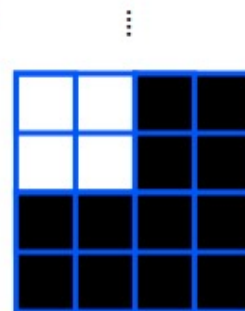


65 nm



2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)

(Industry's Choice)



4 cores @ 2x1.8 GHz
(12 cores dark)

**75% dark after 2 generations;
93% dark after 4 generations**

32 nm

Source: M. Taylor, "Is Dark Silicon Useful? Harnessing the Four Horsemen of the Coming Dark Silicon Apocalypse", DAC 2012

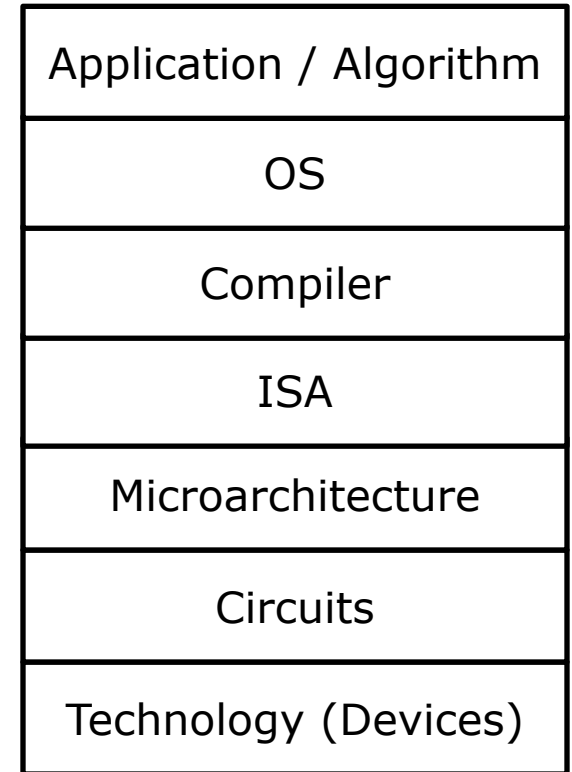
Power Management options?

Power Management options?

| |
|-------------------------|
| Application / Algorithm |
| OS |
| Compiler |
| ISA |
| Microarchitecture |
| Circuits |
| Technology (Devices) |

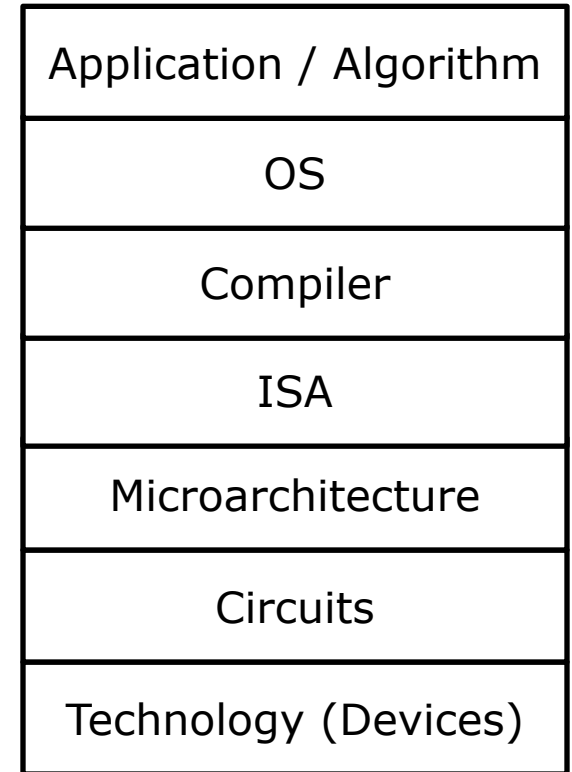
Power Management options?

- Tackle power across all levels of the computing stack



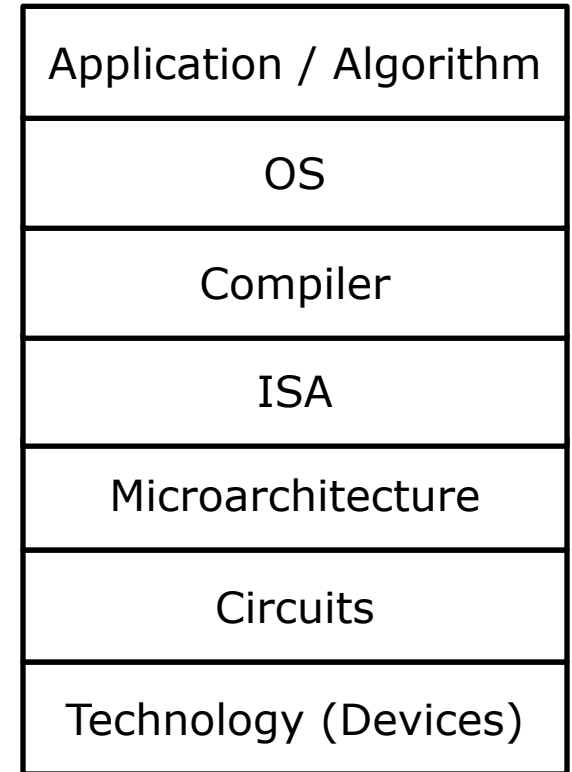
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**



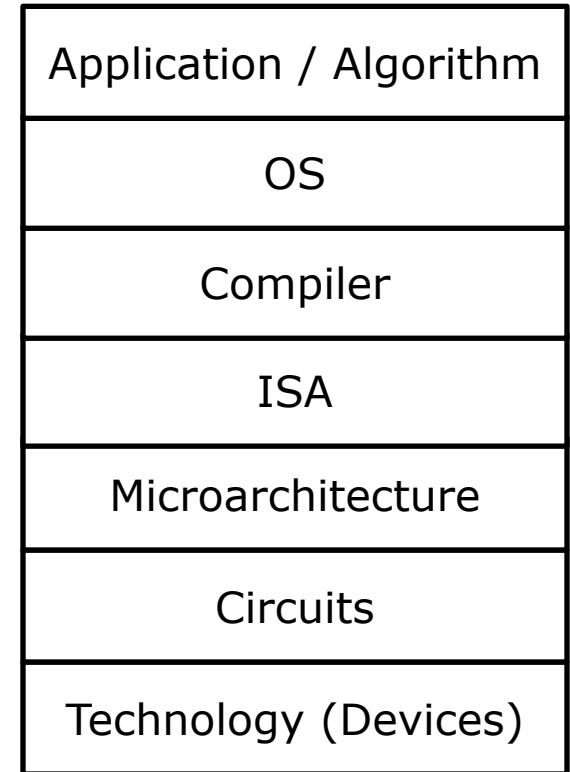
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching



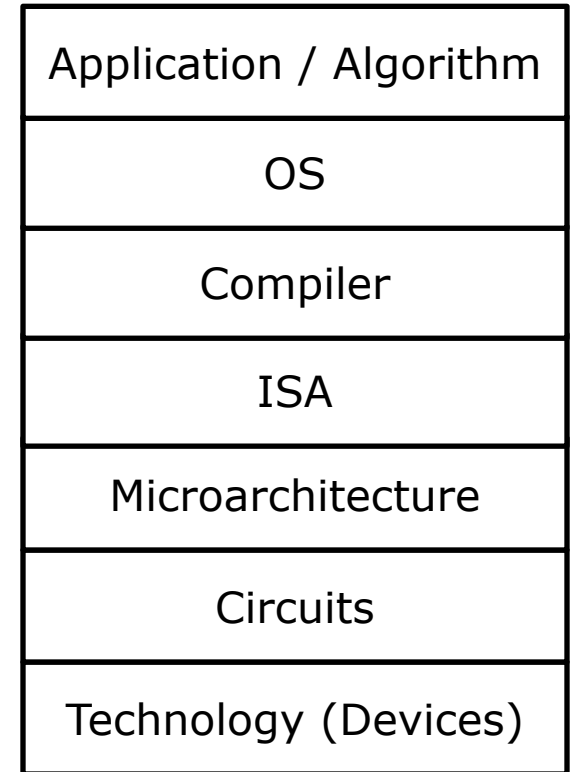
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**



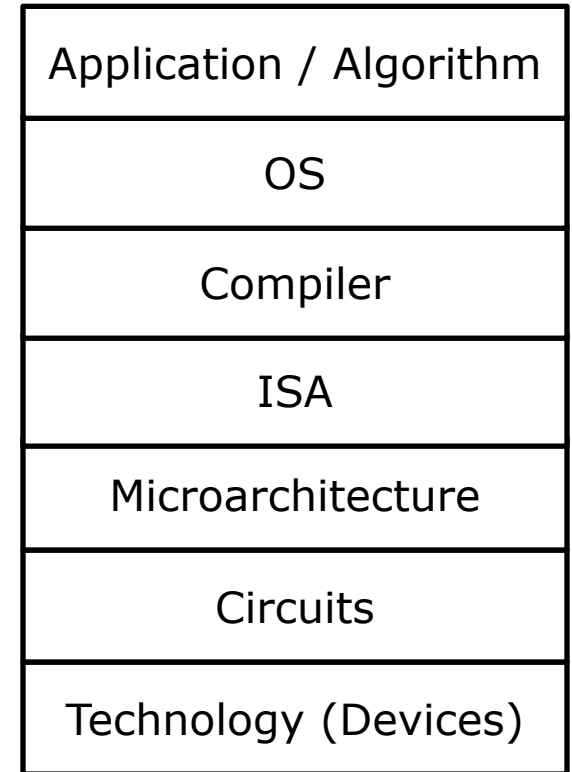
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation



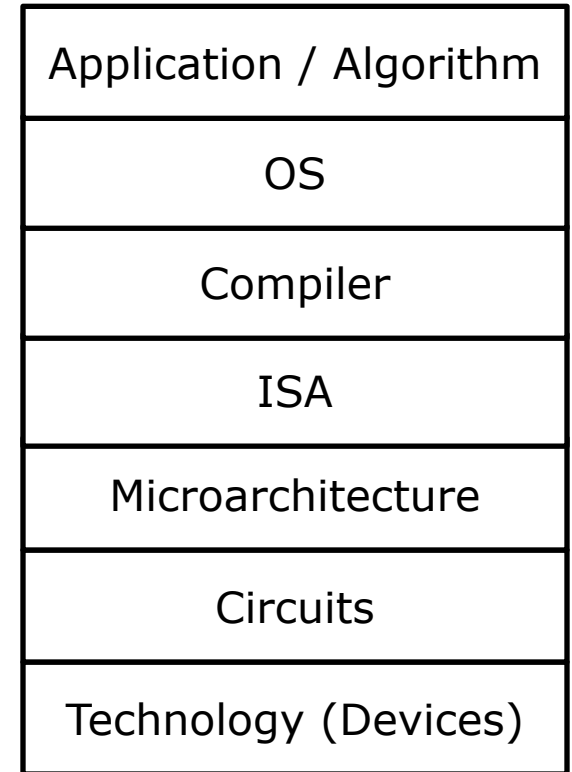
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support



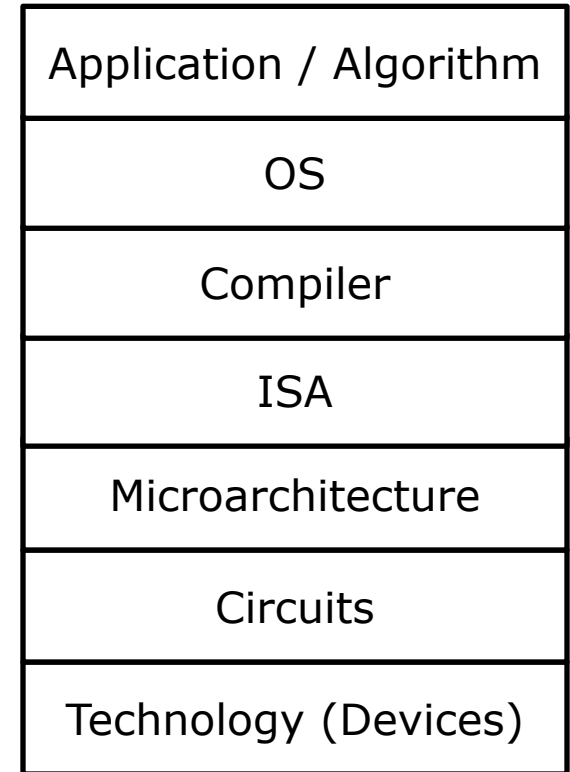
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS



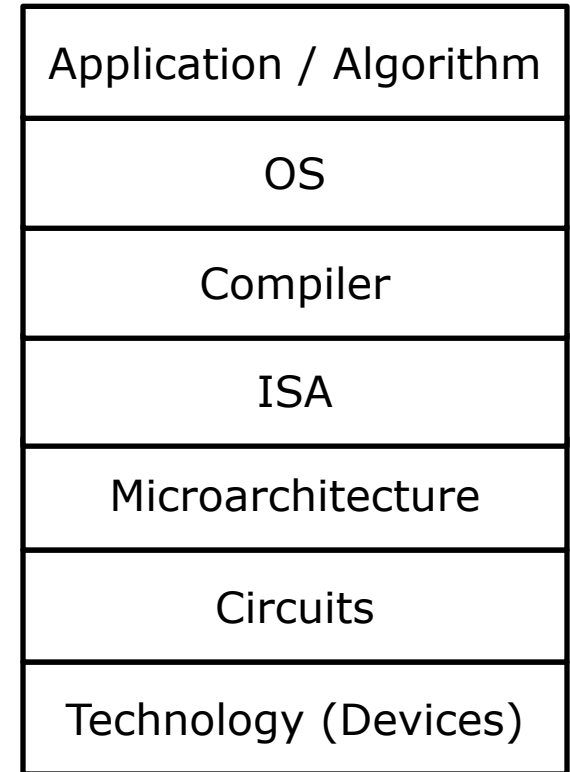
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**



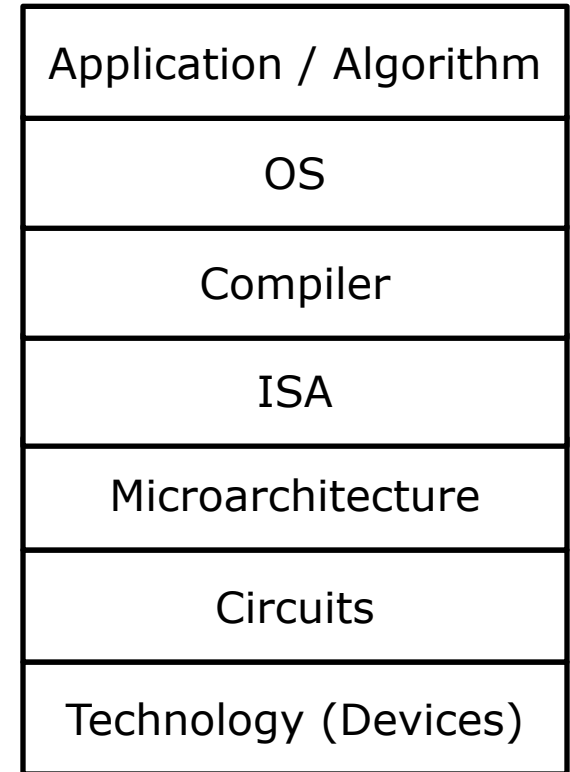
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design



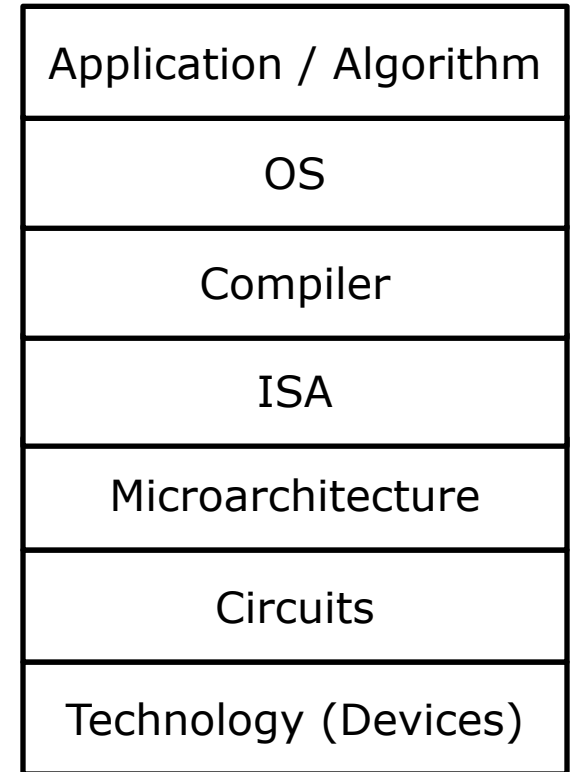
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism



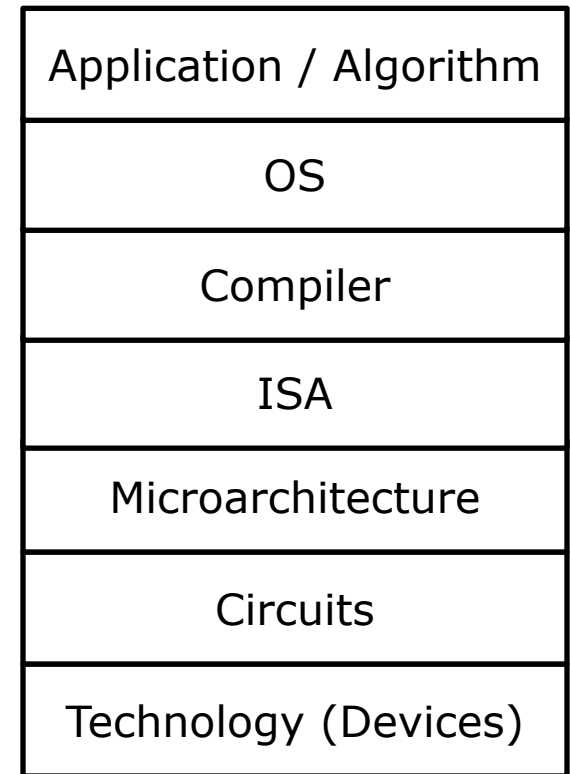
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization



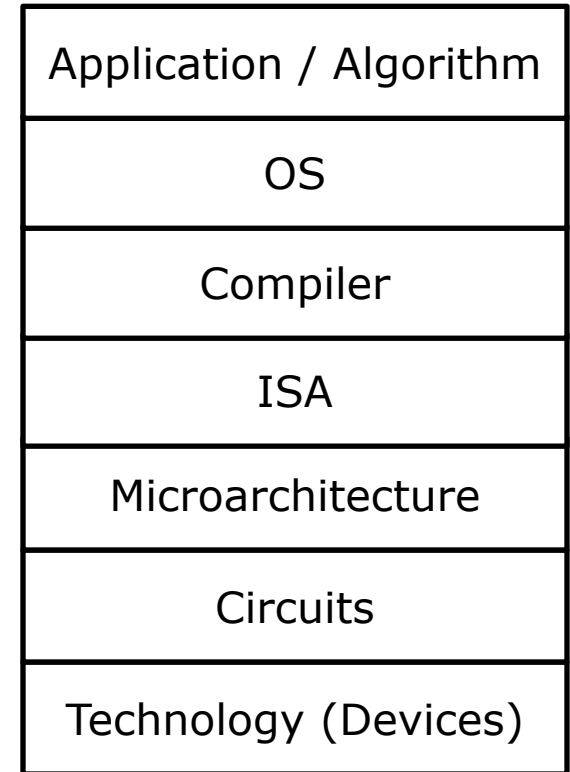
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**



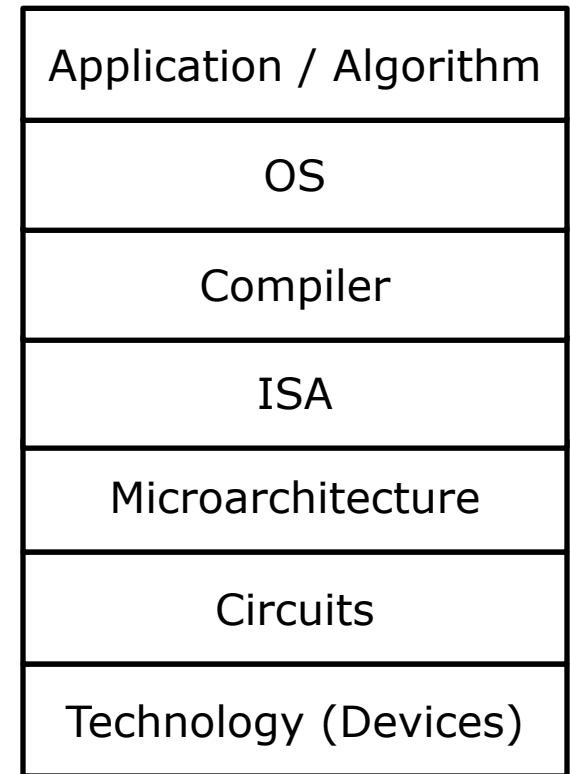
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior



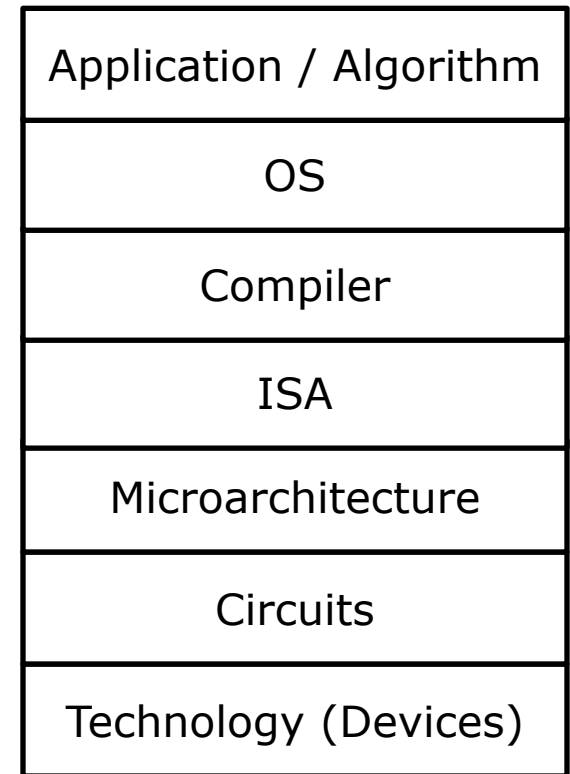
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior
 - **OS**



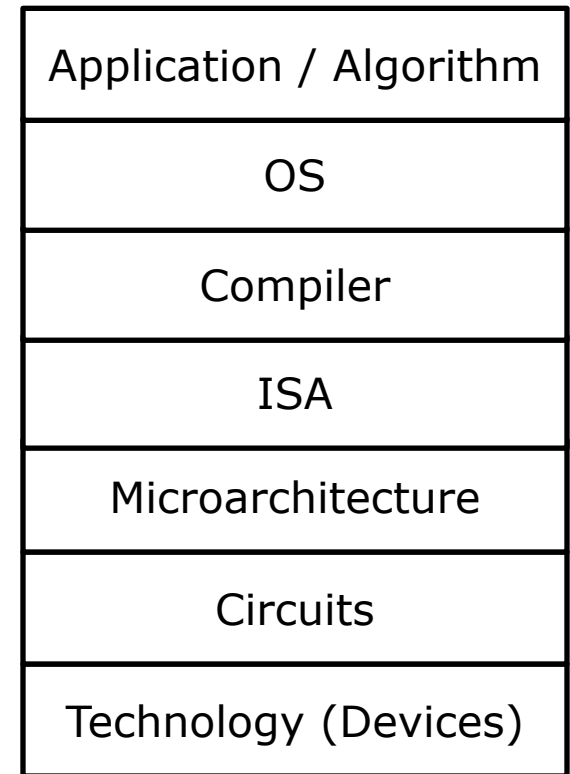
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior
 - **OS**
 - Tune DVFS and power states



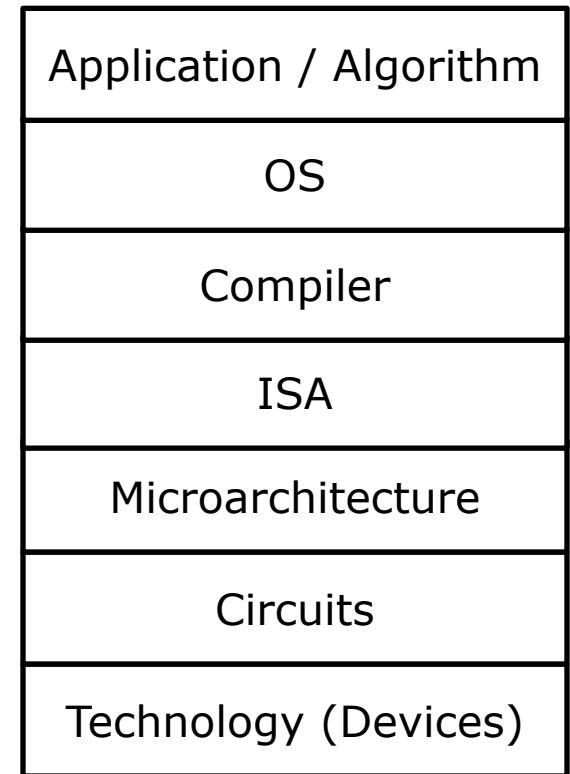
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior
 - **OS**
 - Tune DVFS and power states
 - **Algorithm:**



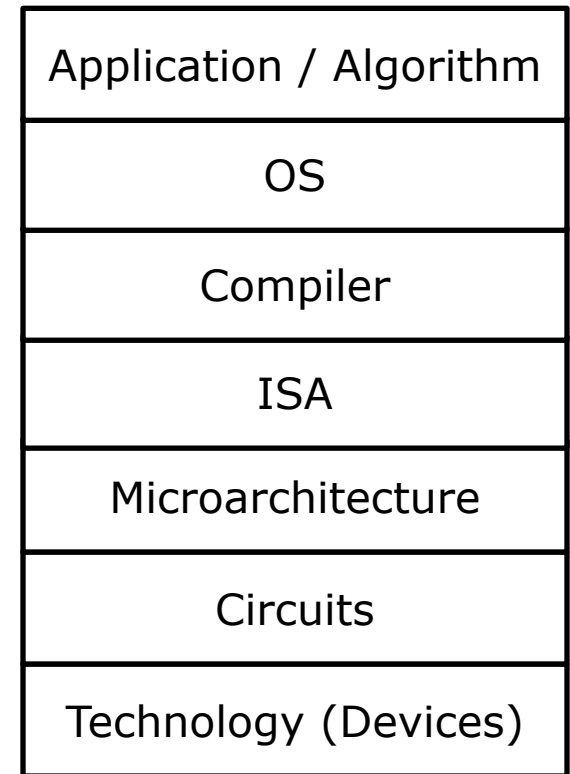
Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior
 - **OS**
 - Tune DVFS and power states
 - **Algorithm:**
 - Switching activity



Power Management options?

- Tackle power across all levels of the computing stack
 - **Technology**
 - Cost of switching
 - **Circuits**
 - High-speed vs low-power implementation
 - Clock gating and power gating support
 - DVFS
 - **Microarchitecture and ISA**
 - Simplify design
 - Parallelism
 - Heterogeneity and Specialization
 - **Compiler**
 - Instruction footprint and Cache behavior
 - **OS**
 - Tune DVFS and power states
 - **Algorithm:**
 - Switching activity



Heterogeneity and Specialization

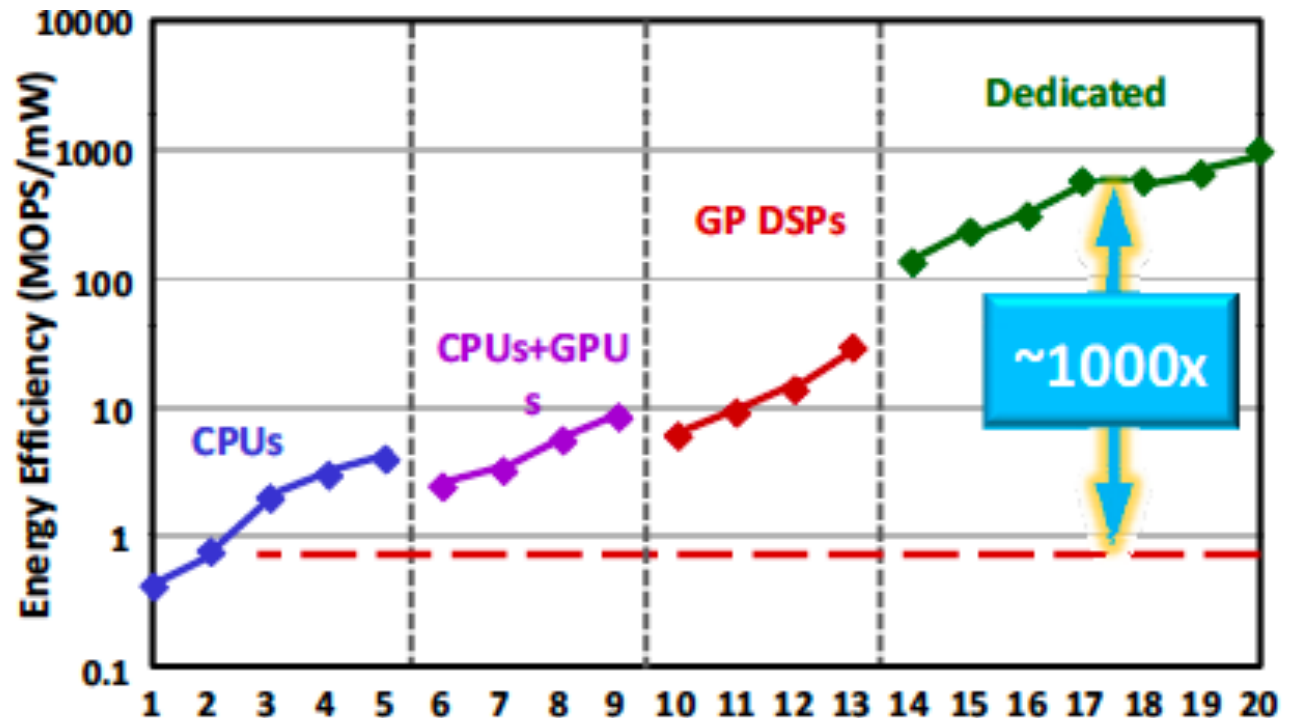
Chip type:

Microprocessor

Microprocessor + GPU

General purpose DSP

Dedicated design



Improve Energy Efficiency via Customization!

Heterogeneity and Specialization

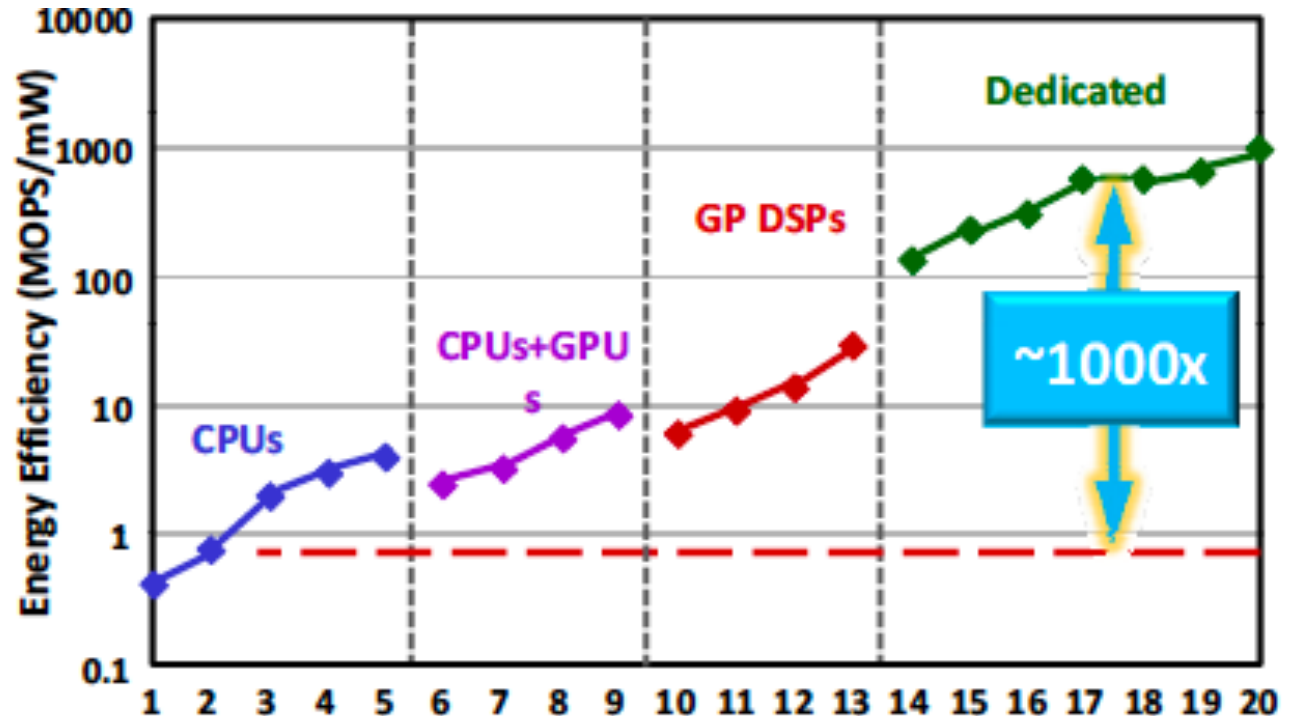
Chip type:

Microprocessor

Microprocessor + GPU

General purpose DSP

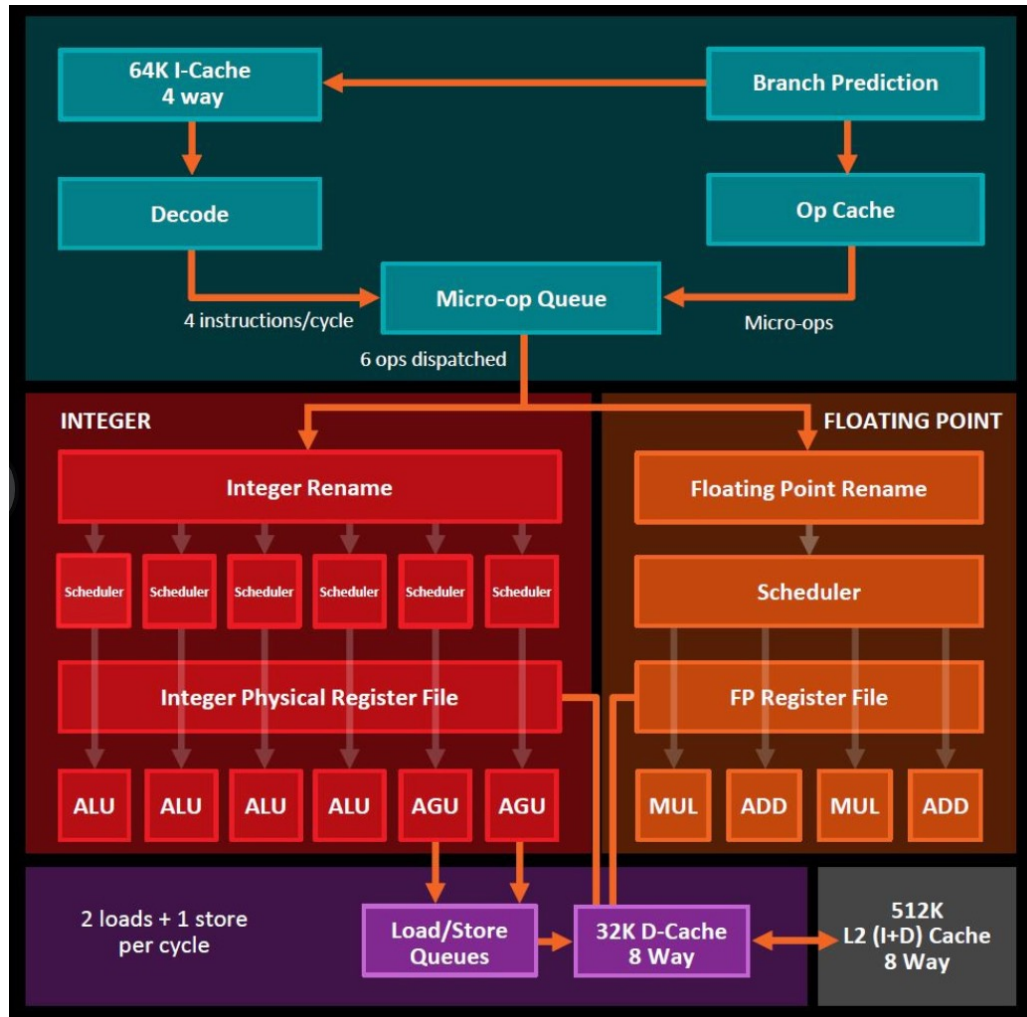
Dedicated design



Improve Energy Efficiency via Customization!

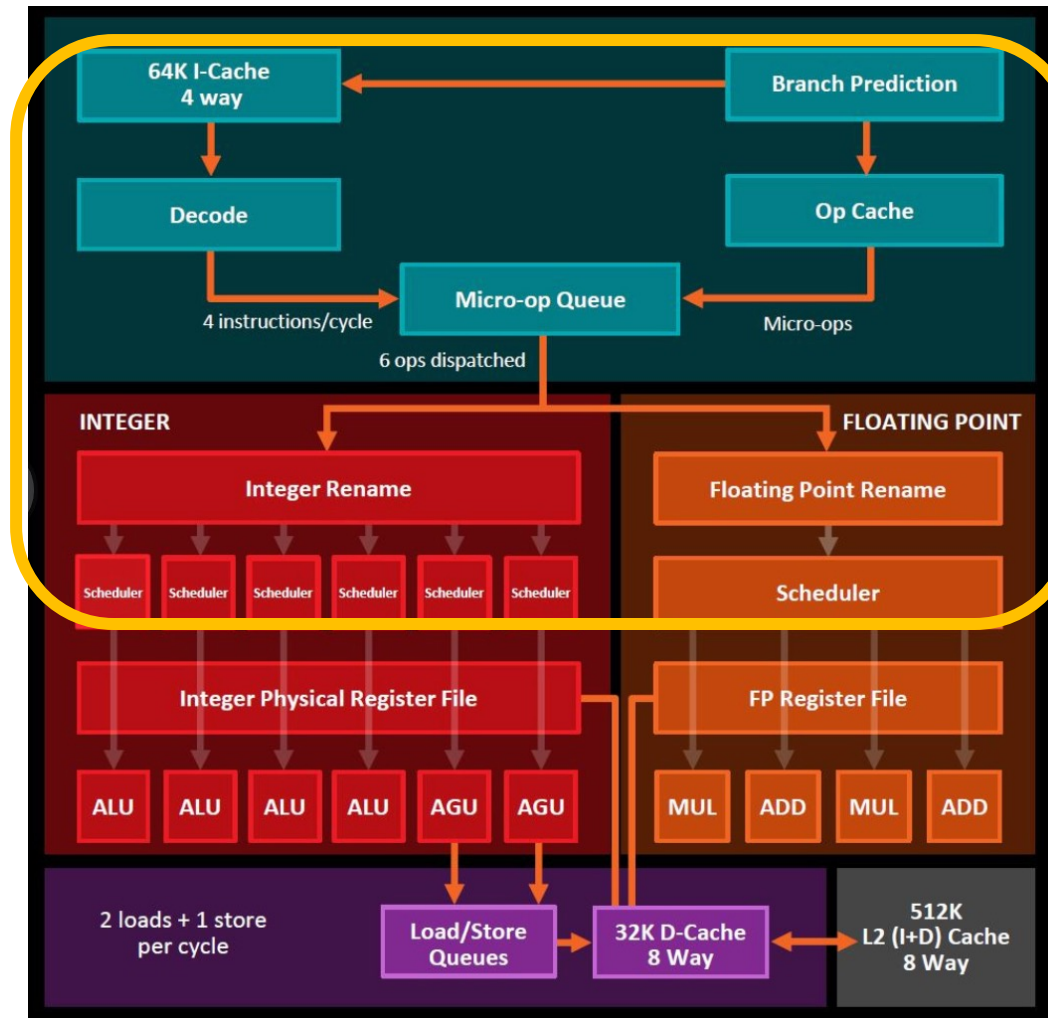
Why?

A modern CPU



AMD Zen (2016)

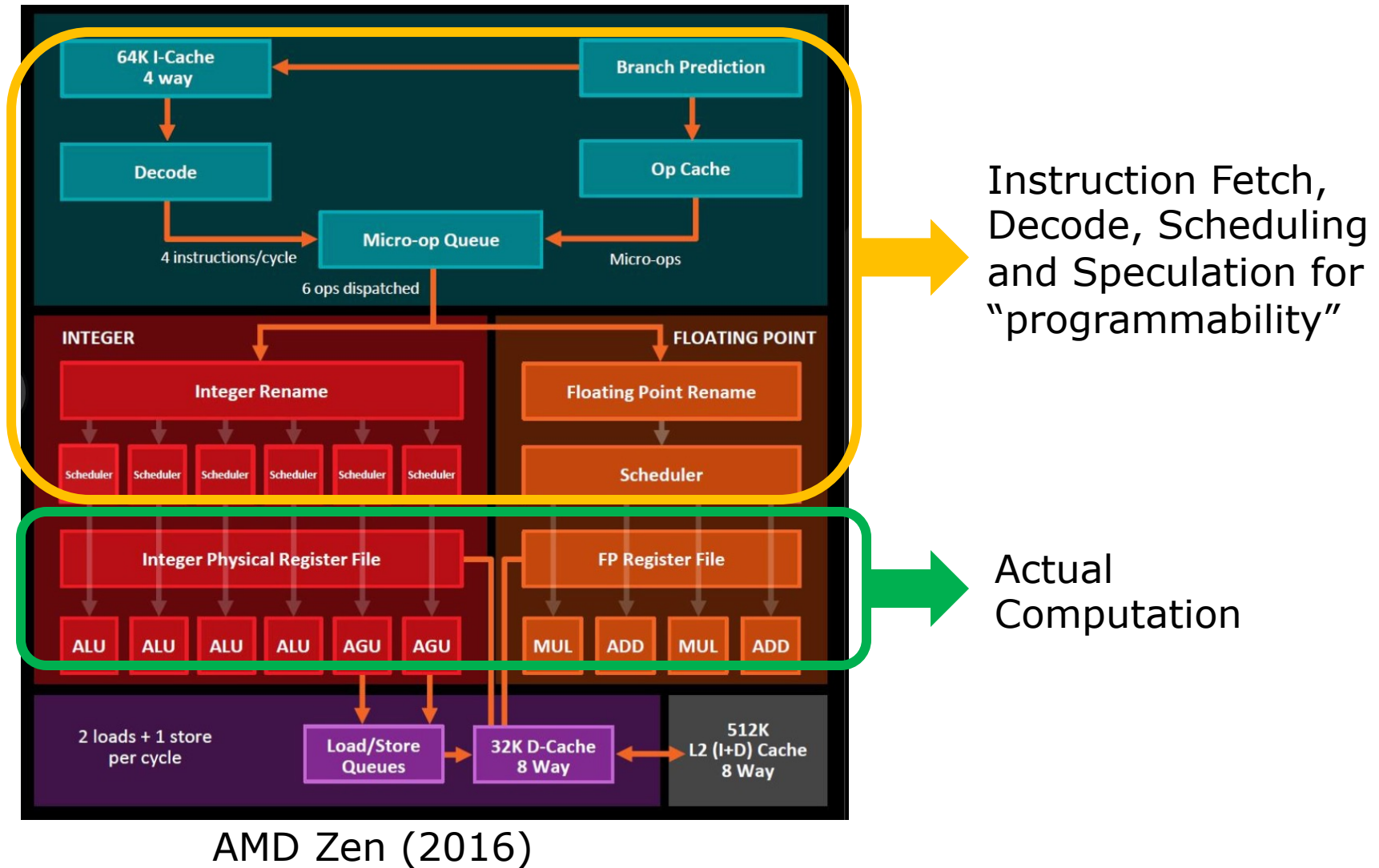
A modern CPU



Instruction Fetch,
Decode, Scheduling
and Speculation for
“programmability”

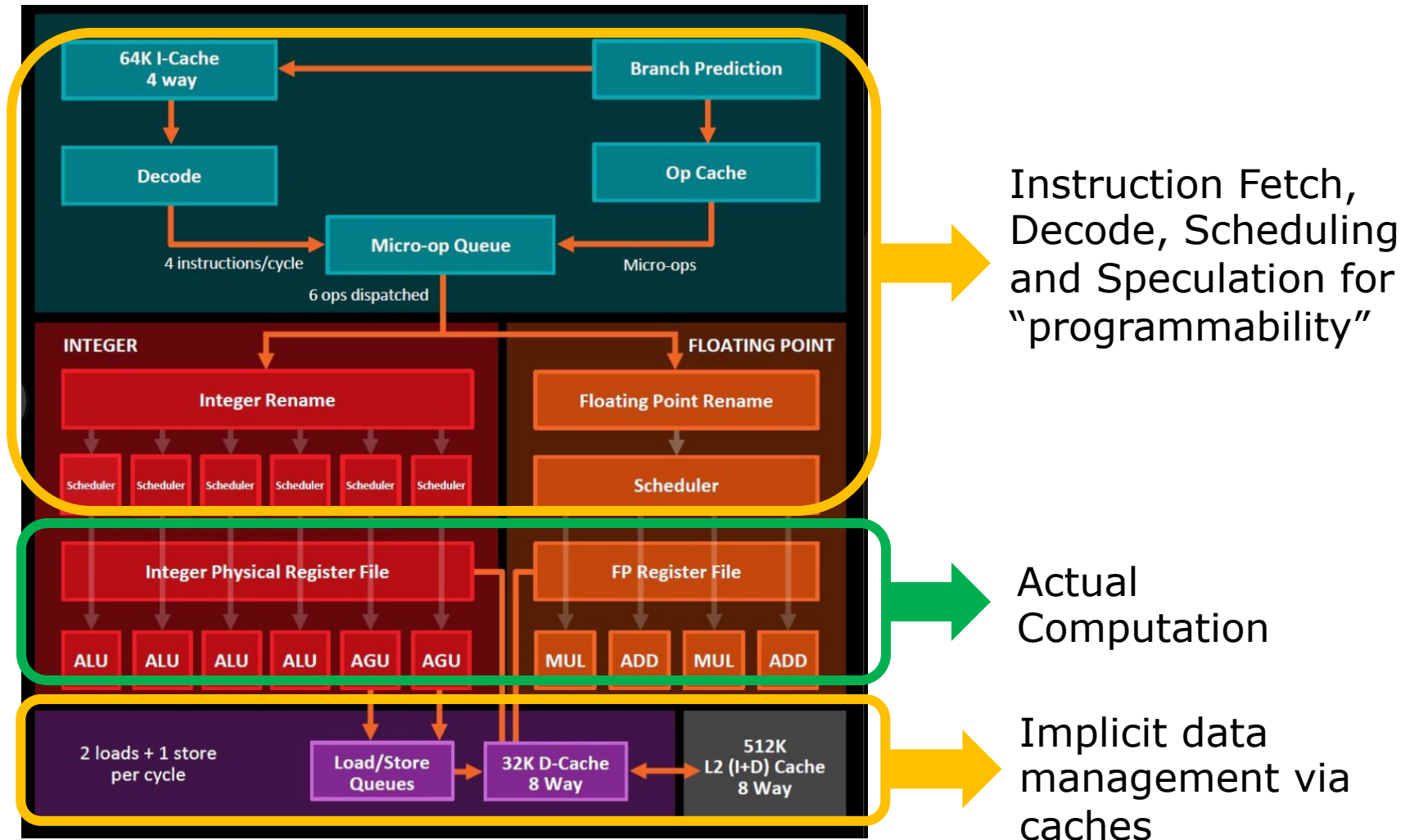
AMD Zen (2016)

A modern CPU



AMD Zen (2016)

A modern CPU

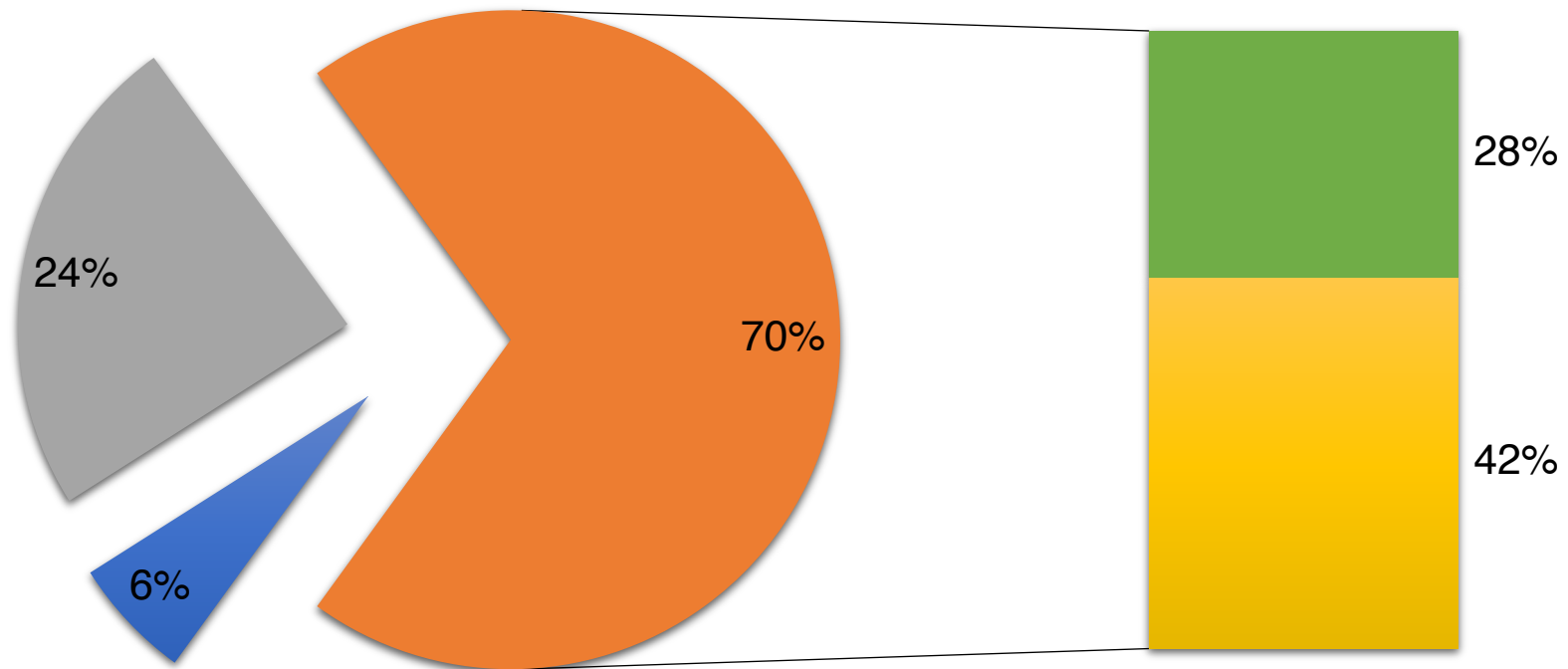


AMD Zen (2016)

Quantifying this overhead

Embedded Processor Energy Breakdown

■ Arithmetic ■ Clock and control ■ Data supply ■ Instruction supply



Source: Dally et al. Efficient Embedded Computing, IEEE'08

Performance/Area Benefits

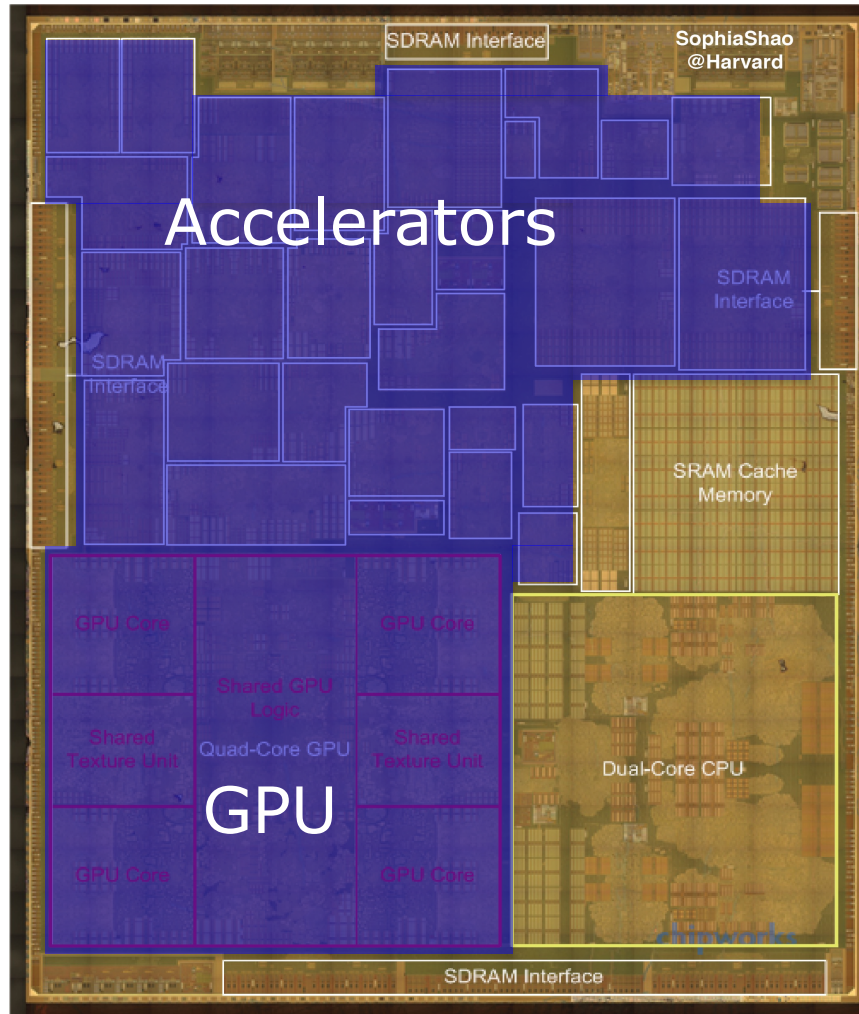
| | | GFLOP/s | (GFLOP/s)/mm ² | GFLOP/J |
|---------------------|--------------------------|---------|---------------------------|----------|
| FFT-2 ¹⁰ | Intel Core i7 (45nm) | 67 | 0.35 | 0.71 |
| | Nvidia GTX285 (55nm) | 250 | 1.41 | 4.2 |
| | Nvidia GTX480 (40nm) | 453 | 1.08 | 4.3 |
| | ATI R5870 (40nm) | - | - | - |
| | Xilinx V6-LX760 (40nm) | 380 | 0.99 | 6.5 |
| | same RTL std cell (65nm) | 952 | 239 | 90 |
| | | Mopt/s | (Mopt/s)/mm ² | Mopt/s/J |
| Black-Scholes | Intel Core i7 (45nm) | 487 | 2.52 | 4.88 |
| | Nvidia GTX285 (55nm) | 10756 | 60.72 | 189 |
| | Nvidia GTX480 (40nm) | - | - | - |
| | ATI R5870 (40nm) | - | - | - |
| | Xilinx V6-LX760 (40nm) | 7800 | 20.26 | 138 |
| | same RTL std cell (65nm) | 25532 | 1719 | 642.5 |

Source: Chung et al., "Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?", MICRO 2010

Outline

- Why do we need accelerators?
- **Why now?**
- How to design accelerators

Domain-specific Accelerators in SoCs



Apple A8 SoC

The rise of AI

Object Detection

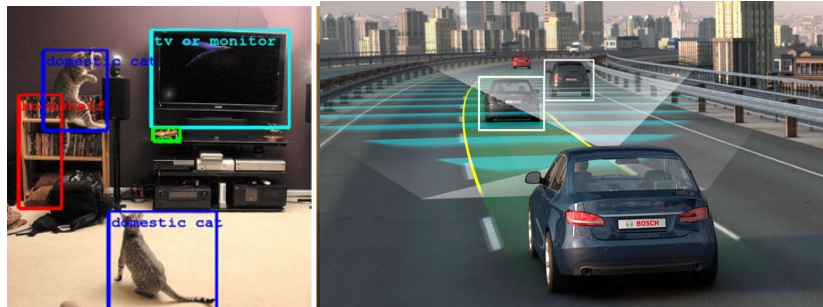
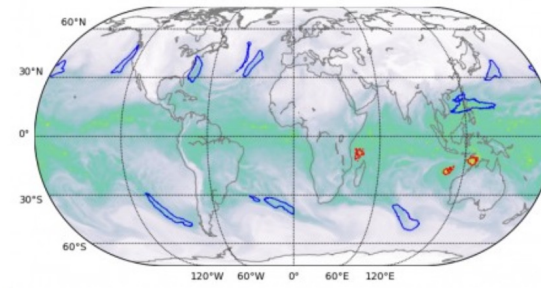
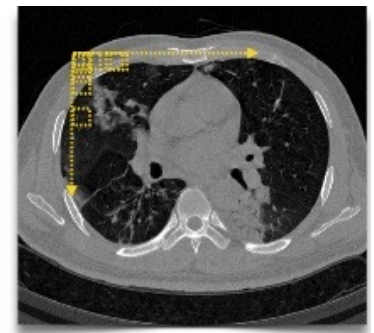


Image Segmentation



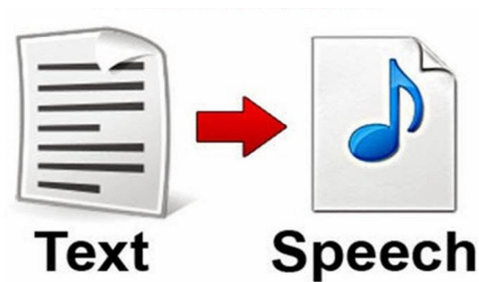
Medical Imaging



Speech Recognition



Text to Speech



Recommendations



Games



The rise of AI

“AI is the new electricity” – Andrew Ng

Object Detection

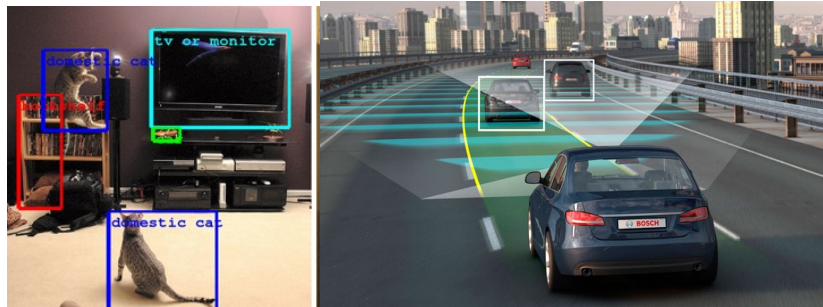
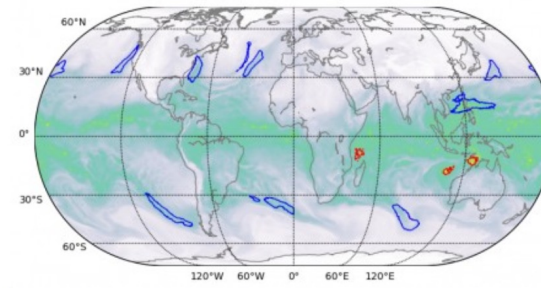
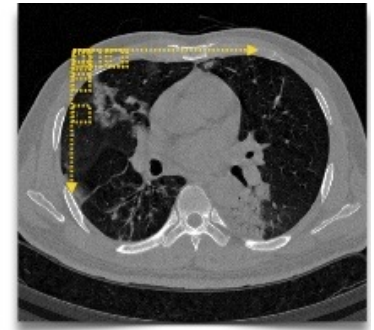


Image Segmentation



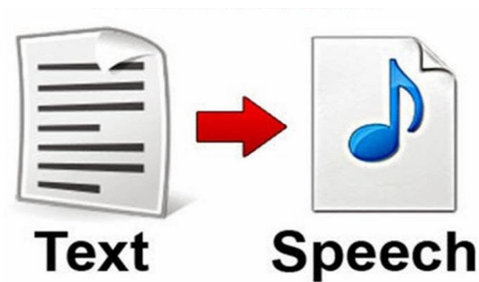
Medical Imaging



Speech Recognition



Text to Speech



Recommendations



Games



AI Compute Demands Growing Exponentially

Deep and steep

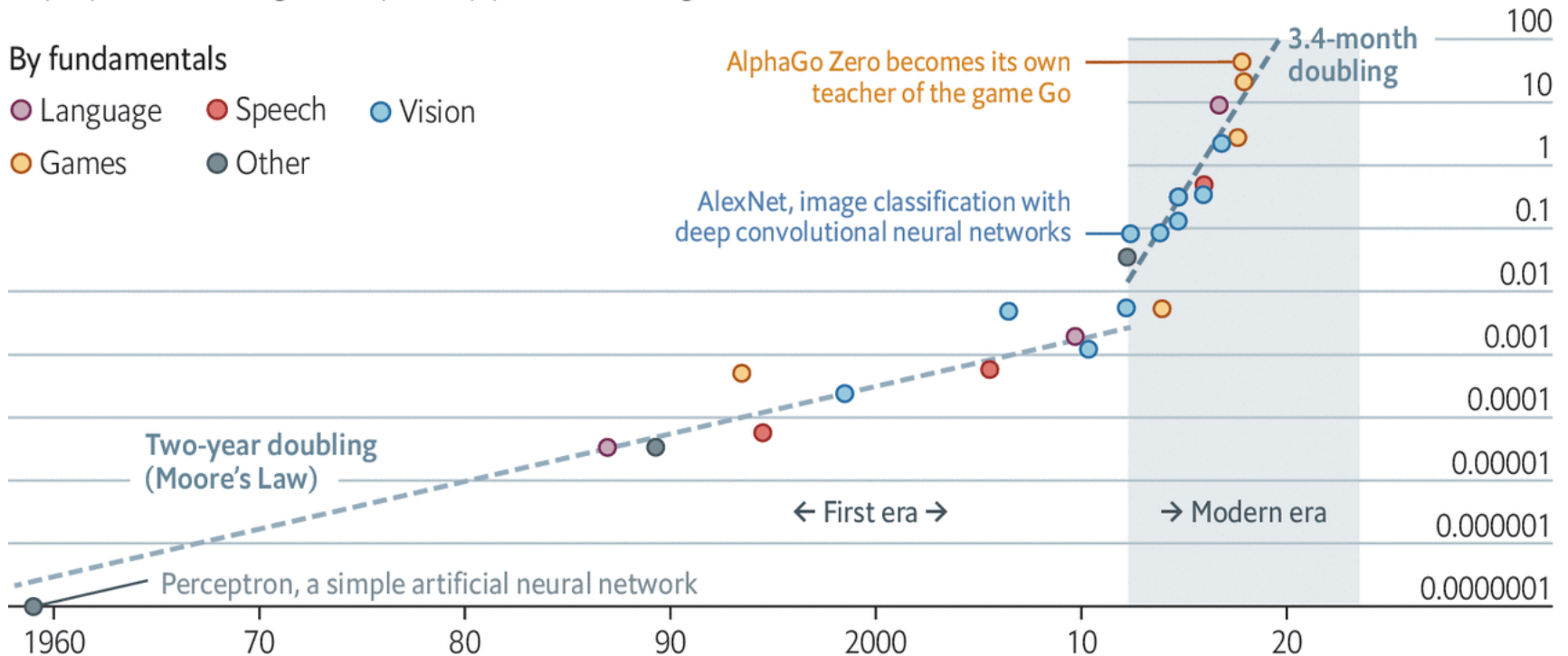
AlexNet to AlphaGo Zero: A 300,000x Increase in Compute

Computing power used in training AI systems

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

By fundamentals

- Language
- Speech
- Vision
- Games
- Other



Source: OpenAI

The Economist

*1 petaflop=10¹⁵ calculations

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

Cloud SW Companies Building HW

Cloud SW Companies Building HW

Google's dedicated TensorFlow processor, or TPU, crushes Intel, Nvidia in inference workloads

By Joel Hruska on April 6, 2017 at 9:48 am | [25 Comments](#)

How Amazon is racing to catch Microsoft and Google in generative A.I. with custom AWS chips

PUBLISHED SAT, AUG 12 2023-9:00 AM EDT | UPDATED MON, AUG 21 2023-7:40 PM EDT

Meta announces AI training and inference chip project

By Katie Paul and Stephen Nellis

May 18, 2023 4:34 PM EDT · Updated 7 months ago

Microsoft announces custom AI chip that could compete with Nvidia

PUBLISHED WED, NOV 15 2023-11:00 AM EST | UPDATED WED, NOV 15 2023-3:05 PM EST

HW Beyond Cloud Computing



Musk Says Tesla Is Building Its Own Chip for Autopilot

TOM SIMONITE BUSINESS 12.08.17 01:09 PM

MUSK SAYS TESLA IS BUILDING ITS OWN CHIP FOR AUTOPILOT



Elon Musk disclosed plans for Tesla to design its own chip to power its self-driving function.

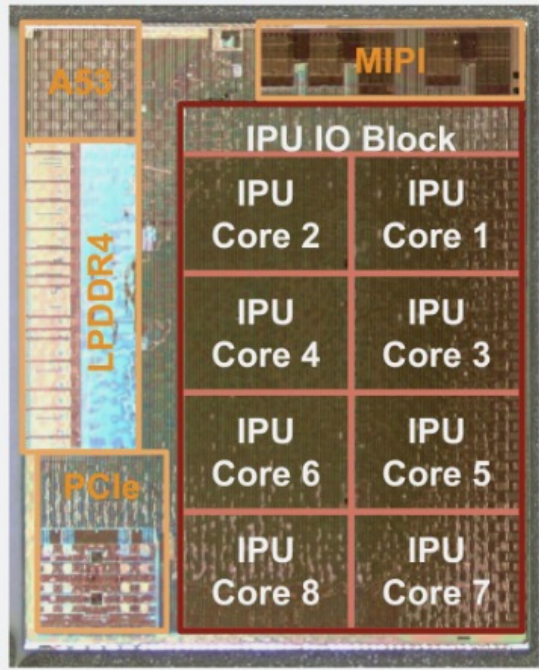
NASA/ALAMY



Surprise! The Pixel 2 is hiding a custom Google SoC for image processing

Google's 8-core Image Processing Unit will be enabled with Android 8.1.

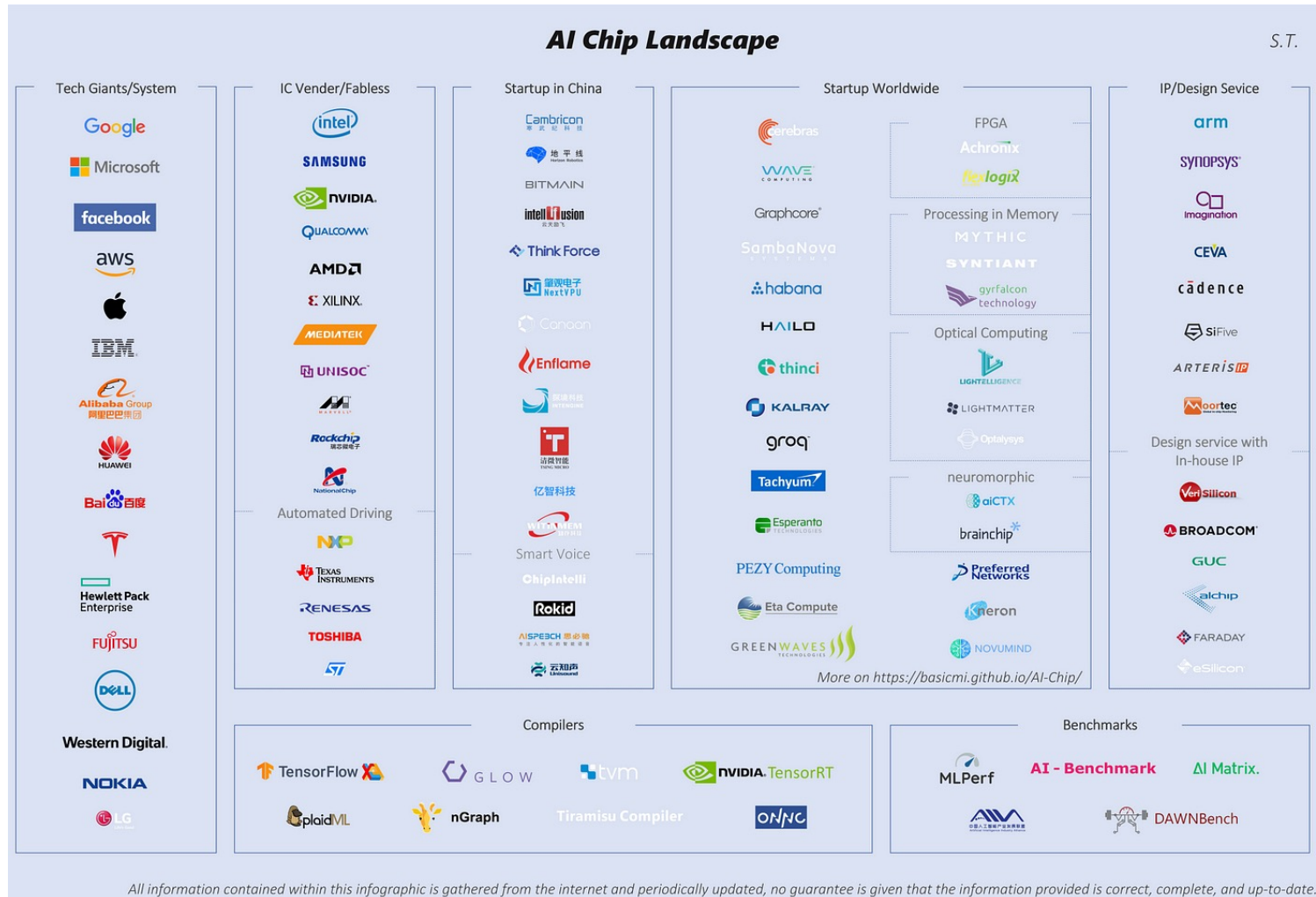
RON AMADEO - 10/17/2017, 9:00 AM



Google

Enlarge / Google's Pixel Visual Core, an SoC designed for image processing and machine learning.

AI Chips ecosystem



Opportunities

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

ACM’s Celebration of 50 Years of the ACM Turing Award (June 2017)

“Compute has been the oxygen of deep learning”

– Ilya Sutskever, Research Director of Open AI

Demand for Computer Architects

Demand for Computer Architects



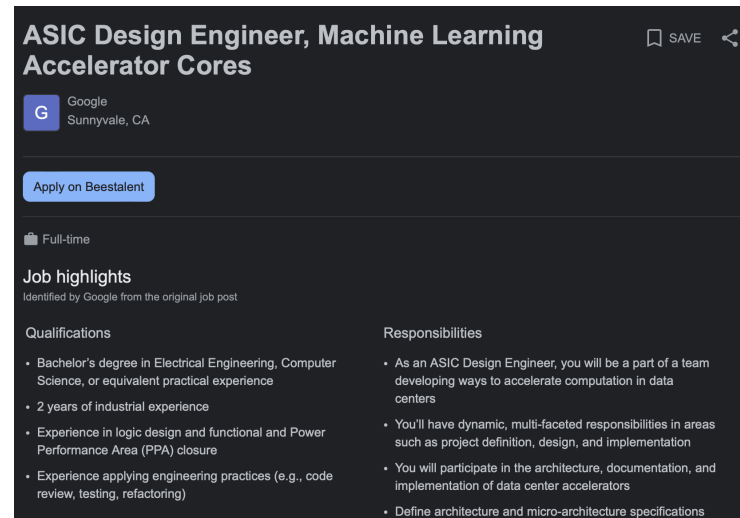
Meta Jobs Areas of Work Locations Career Programs How We Work Blog | Login

ASIC Engineer, Architecture

Sunnyvale, CA | + 3 more

Apply to Job

Meta is seeking an ASIC Engineer, Architecture to join our Infrastructure organization. Our servers and data centers are the foundation upon which our rapidly scaling infrastructure efficiently operates and upon which our innovative services are delivered. By holding this role, you will be an integral member of an ASIC team to build accelerators for some of our top workloads enabling our data centers to scale efficiently. You will have an opportunity to work with AI/Machine Learning (ML) and video codec experts in the company, help architect state-of-the-art machine learning accelerators and video transcoders and contribute to modeling these accelerators. Come work and learn alongside our expert engineers to build "Green" data center accelerators.



ASIC Design Engineer, Machine Learning Accelerator Cores

Google Sunnyvale, CA

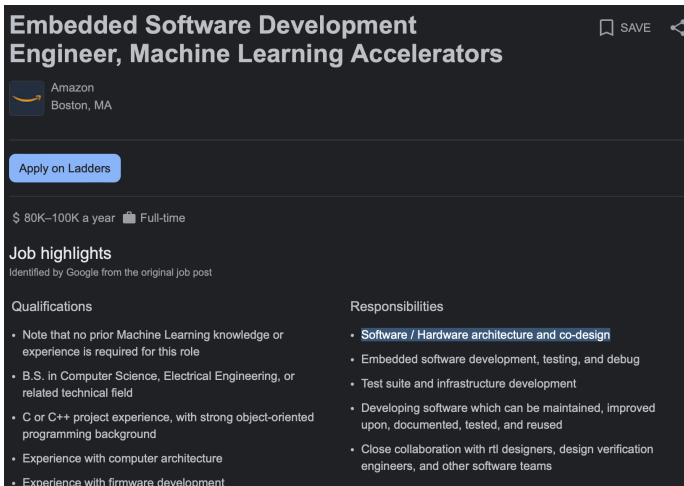
Apply on Beestalent

Full-time

Job highlights

Identified by Google from the original job post

| Qualifications | Responsibilities |
|---|--|
| <ul style="list-style-type: none">Bachelor's degree in Electrical Engineering, Computer Science, or equivalent practical experience2 years of industrial experienceExperience in logic design and functional and Power Performance Area (PPA) closureExperience applying engineering practices (e.g., code review, testing, refactoring) | <ul style="list-style-type: none">As an ASIC Design Engineer, you will be a part of a team developing ways to accelerate computation in data centersYou'll have dynamic, multi-faceted responsibilities in areas such as project definition, design, and implementationYou will participate in the architecture, documentation, and implementation of data center acceleratorsDefine architecture and micro-architecture specifications |



Embedded Software Development Engineer, Machine Learning Accelerators

Amazon Boston, MA

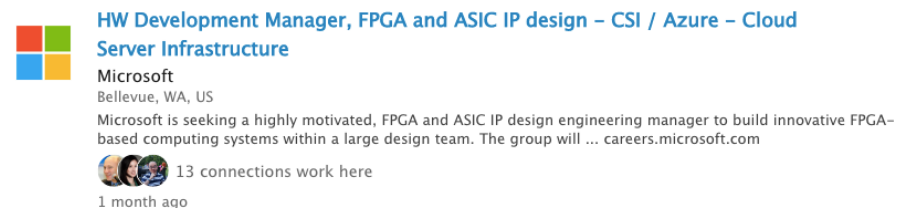
Apply on Ladders

\$ 80K–100K a year Full-time

Job highlights

Identified by Google from the original job post

| Qualifications | Responsibilities |
|---|--|
| <ul style="list-style-type: none">Note that no prior Machine Learning knowledge or experience is required for this roleB.S. in Computer Science, Electrical Engineering, or related technical fieldC or C++ project experience, with strong object-oriented programming backgroundExperience with computer architectureExperience with firmware development | <ul style="list-style-type: none">Software / Hardware architecture and co-designEmbedded software development, testing, and debugTest suite and infrastructure developmentDeveloping software which can be maintained, improved upon, documented, tested, and reusedClose collaboration with rtl designers, design verification engineers, and other software teams |

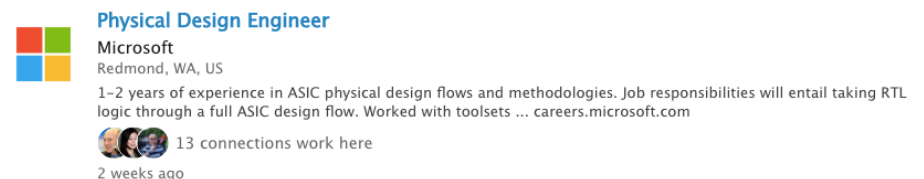


HW Development Manager, FPGA and ASIC IP design – CSI / Azure – Cloud Server Infrastructure

Microsoft Bellevue, WA, US

Microsoft is seeking a highly motivated, FPGA and ASIC IP design engineering manager to build innovative FPGA-based computing systems within a large design team. The group will ... careers.microsoft.com

13 connections work here
1 month ago



Physical Design Engineer

Microsoft Redmond, WA, US

1–2 years of experience in ASIC physical design flows and methodologies. Job responsibilities will entail taking RTL logic through a full ASIC design flow. Worked with toolsets ... careers.microsoft.com

13 connections work here
2 weeks ago

Why do we need DNN accelerators?

Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**

Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**
 - Billions of computations

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**

- Billions of computations → **Need lots of parallel compute**

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

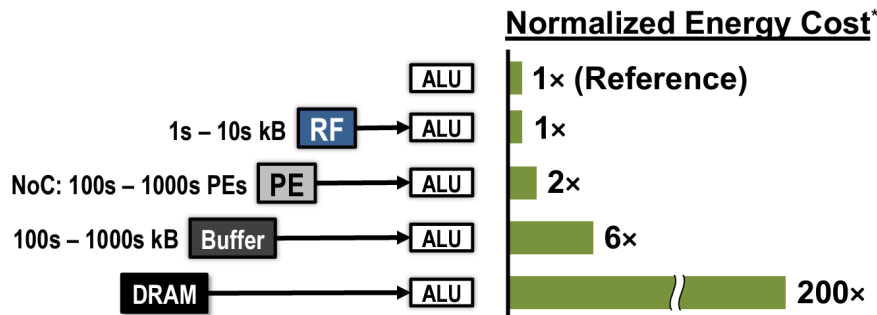
Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**

- Billions of computations → **Need lots of parallel compute**

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

- Heavy data movement



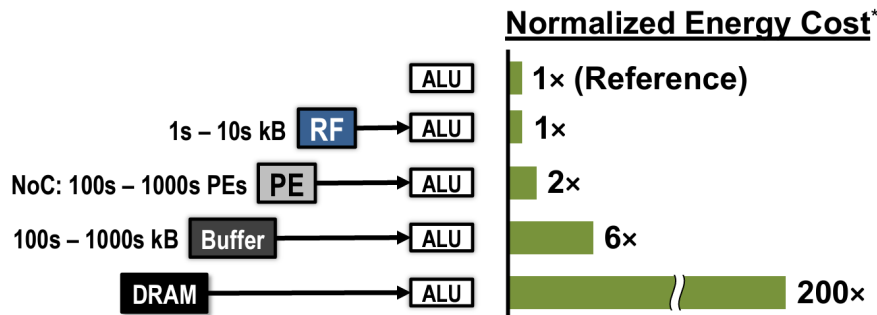
Why do we need DNN accelerators?

- **Millions of Parameters (i.e., weights)**

- Billions of computations → **Need lots of parallel compute**

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

- Heavy data movement → **Need to reduce energy**



Why do we need DNN accelerators?

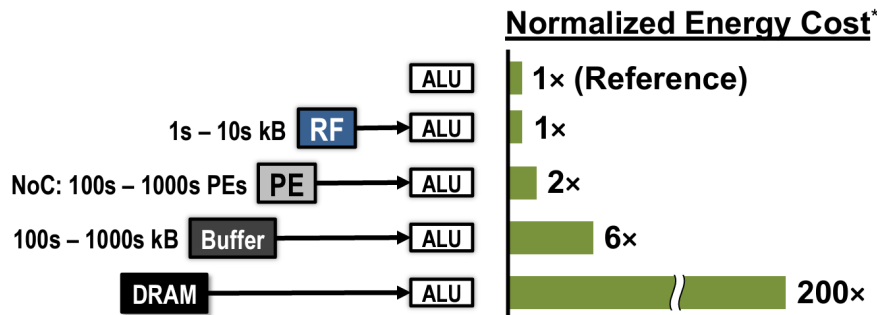
- **Millions of Parameters (i.e., weights)**

- Billions of computations → Need lots of parallel compute

| DNN Topology | Number of Weights |
|------------------|-------------------|
| AlexNet (2012) | 3.98M |
| VGGnet-16 (2014) | 28.25M |
| GoogleNet (2015) | 6.77M |
| Resnet-50 (2016) | 23M |
| DLRM (2019) | 540M |
| Megatron (2019) | 8.3B |

This makes CPUs inefficient

- Heavy data movement → Need to reduce energy

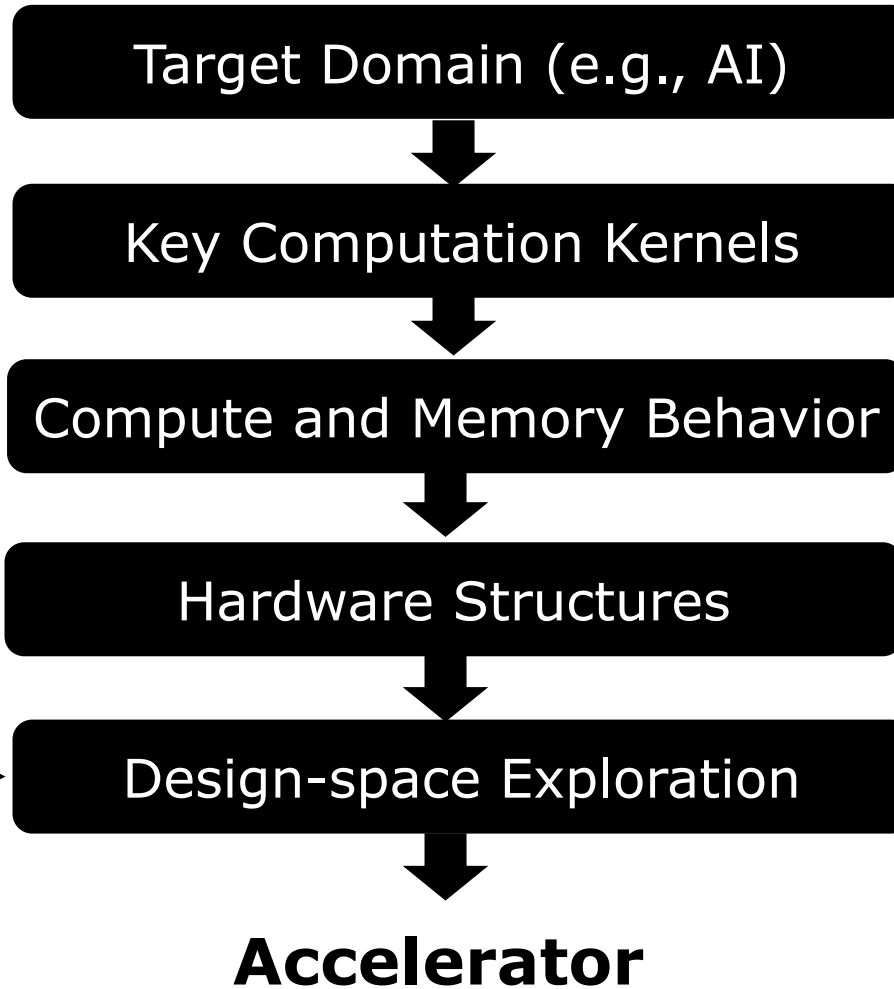


This makes GPUs inefficient

Outline

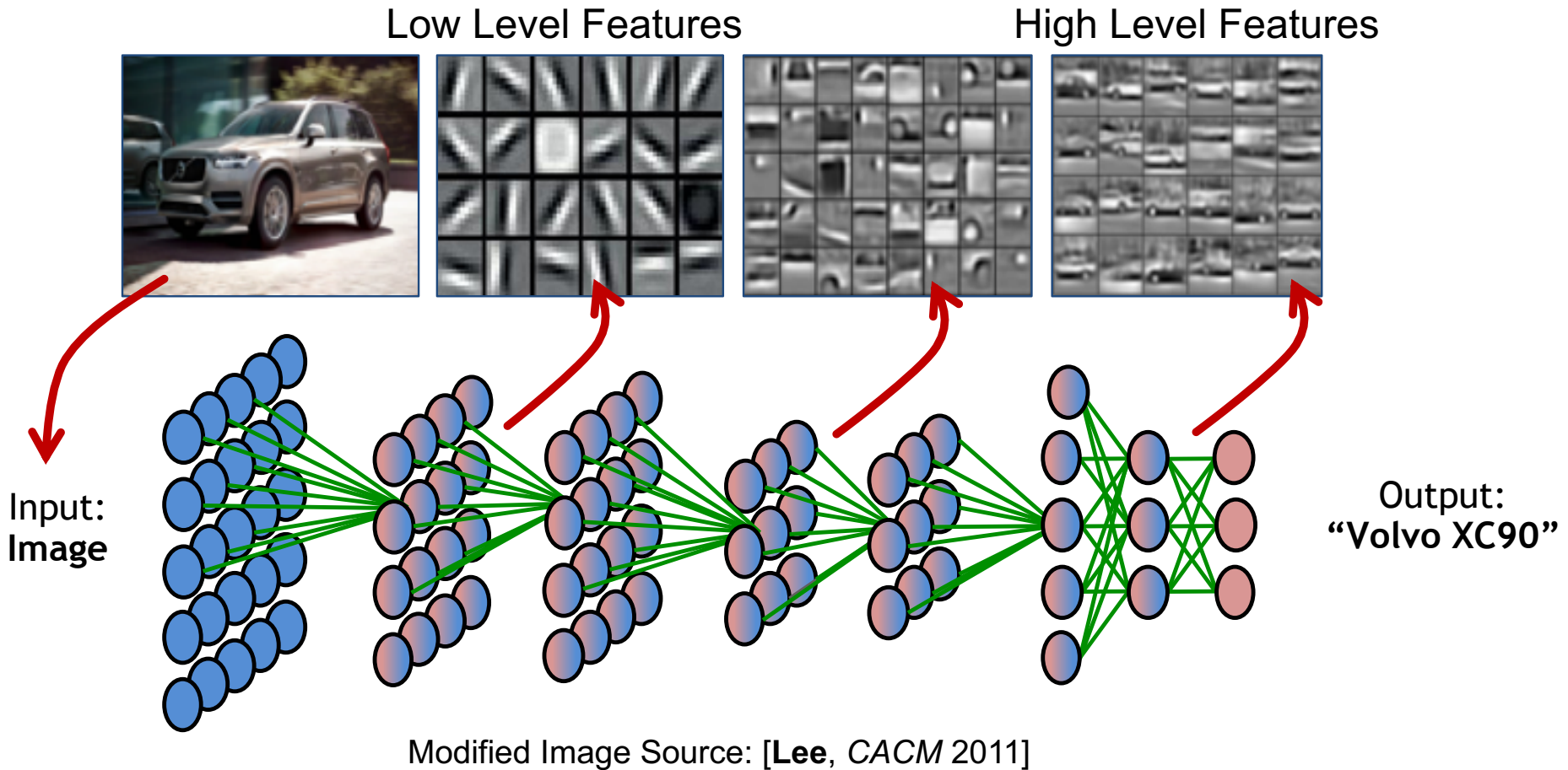
- Why do we need accelerators?
- Why now?
- How to design accelerators

HW-SW Co-Design

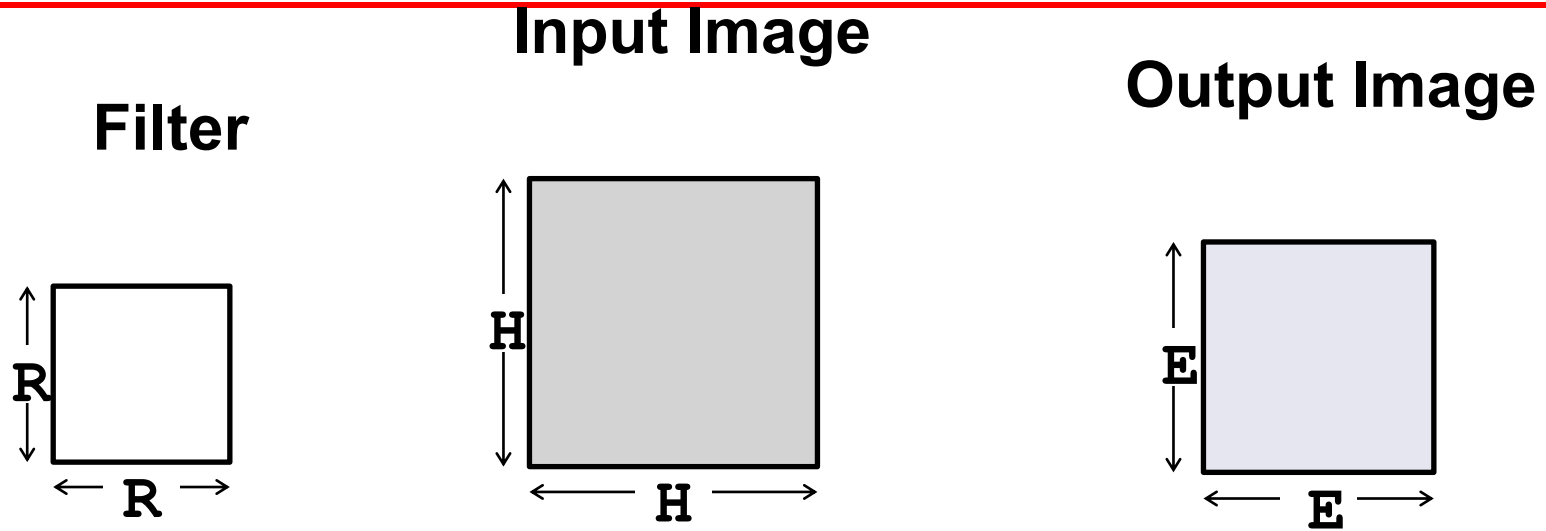


*Constraints
(e.g, Area and
Power)*

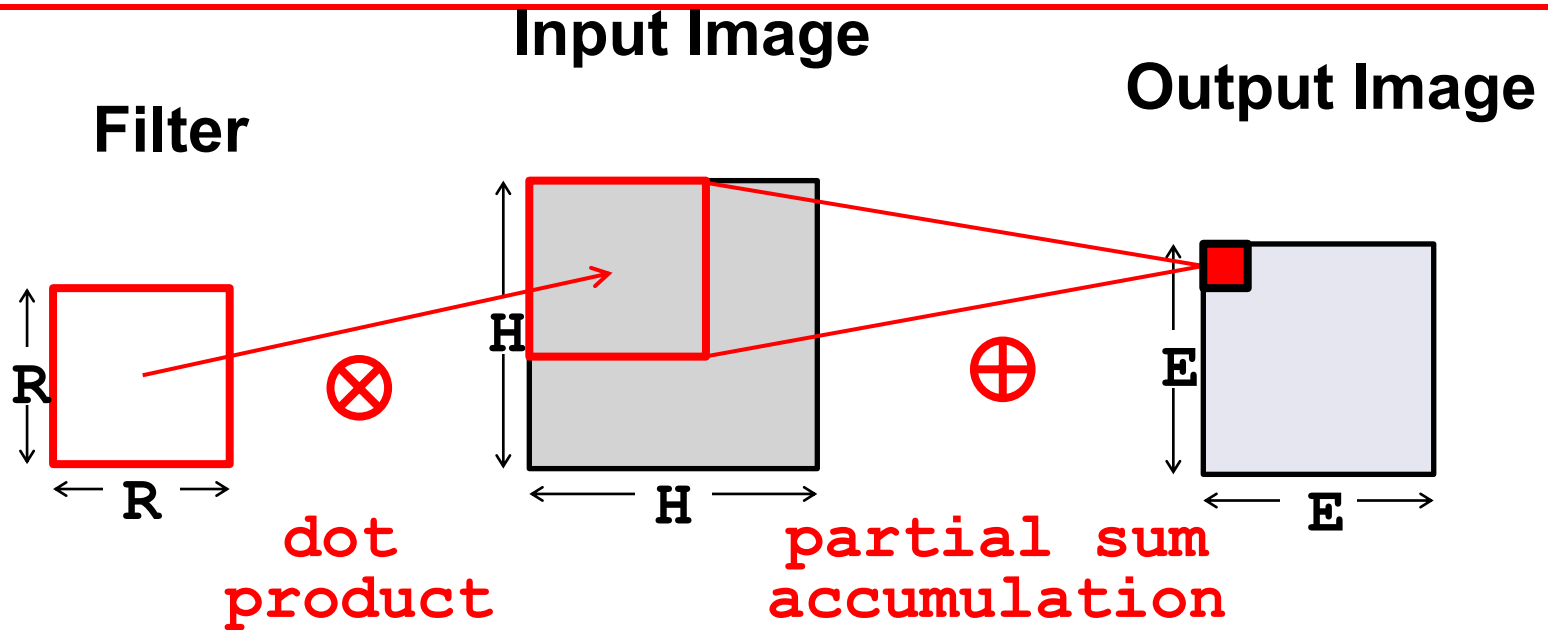
Understanding Deep Neural Networks



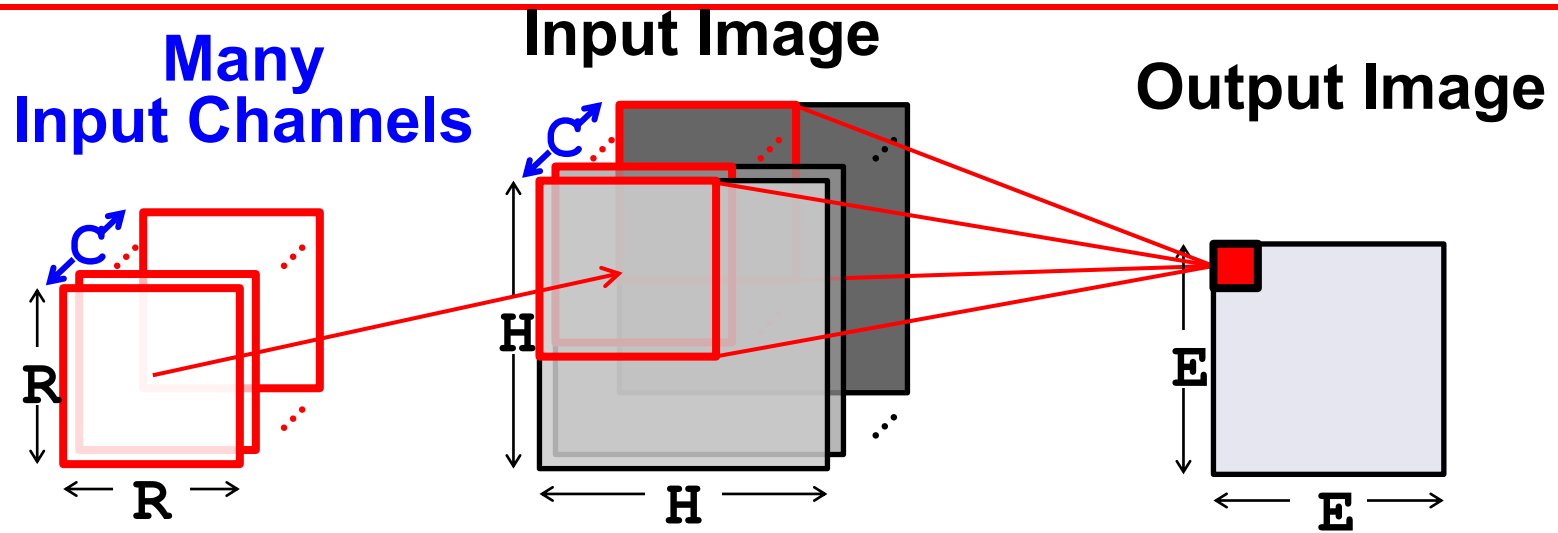
Convolution Operation



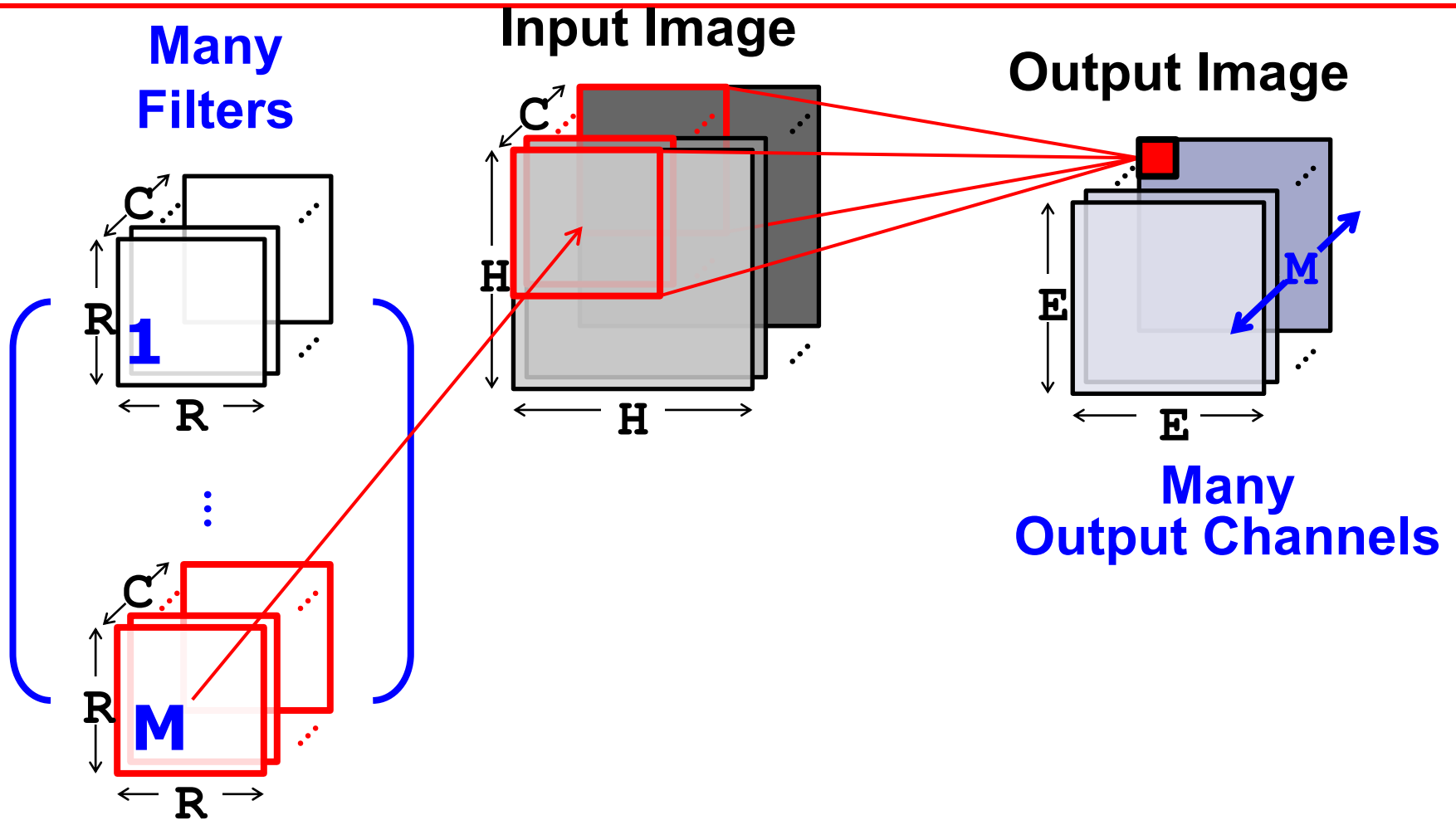
Convolution Operation



Convolution Operation



Convolution Operation

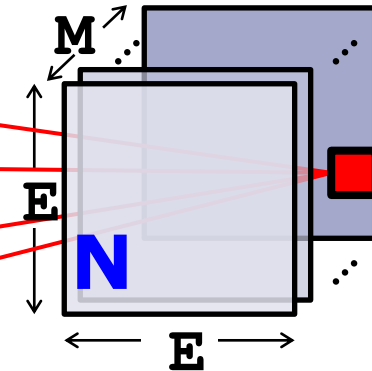
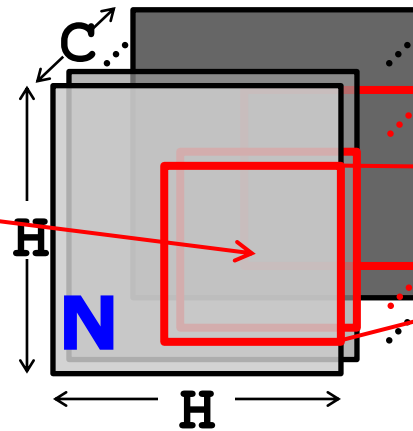
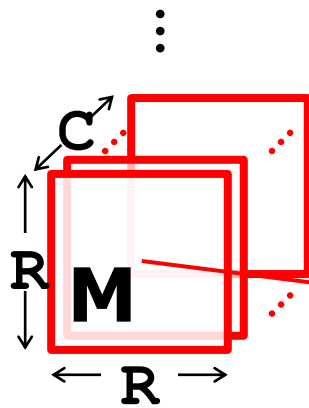
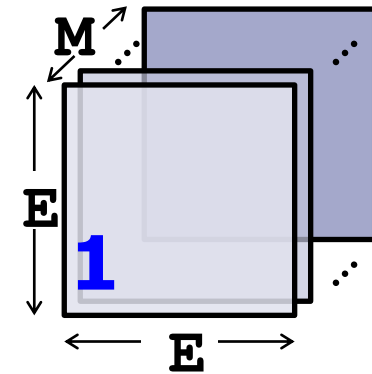
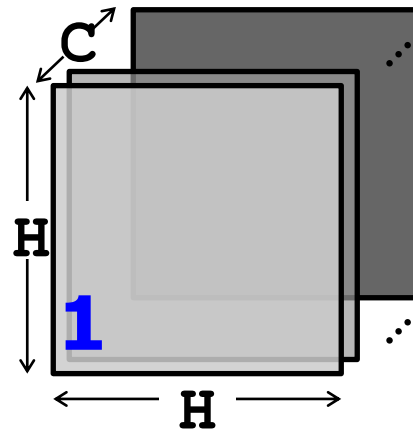
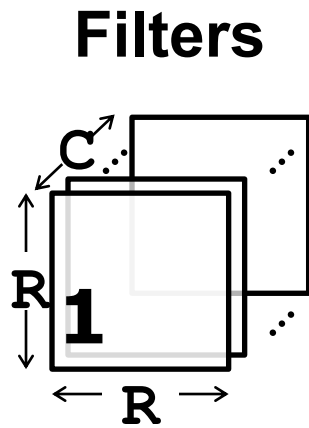


Convolution Operation

Many

Input Images

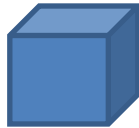
Many
Output Images



Representation: Tensors

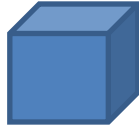
Representation: Tensors

Rank-0 - Scalar



Representation: Tensors

Rank-0 - Scalar

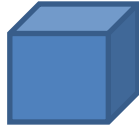


Rank-1 - Vector



Representation: Tensors

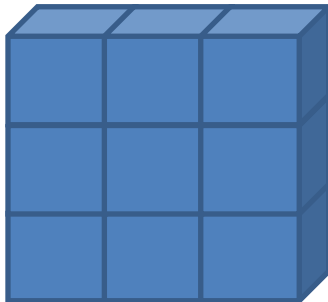
Rank-0 - Scalar



Rank-1 - Vector

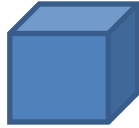


Rank-2 - Matrix



Representation: Tensors

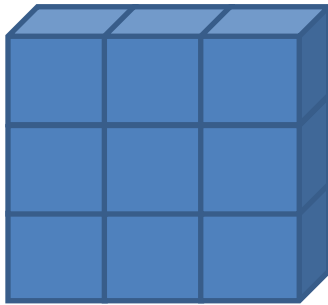
Rank-0 - Scalar



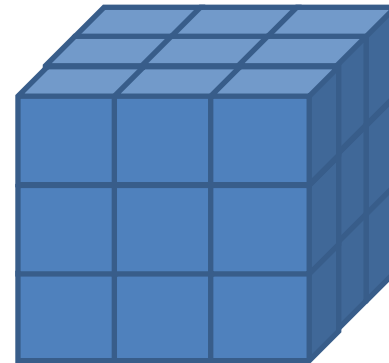
Rank-1 - Vector



Rank-2 - Matrix

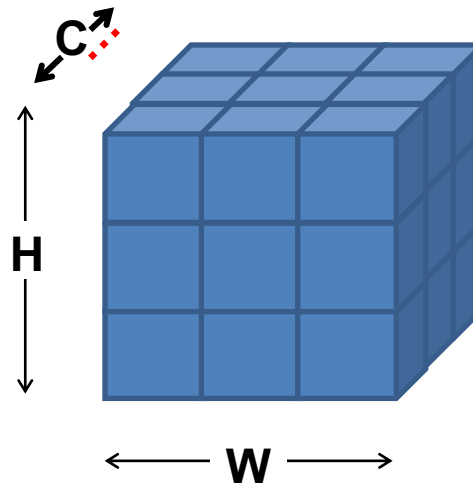


Rank-3 - Cube



Example: Input Act/Fmap Tensor

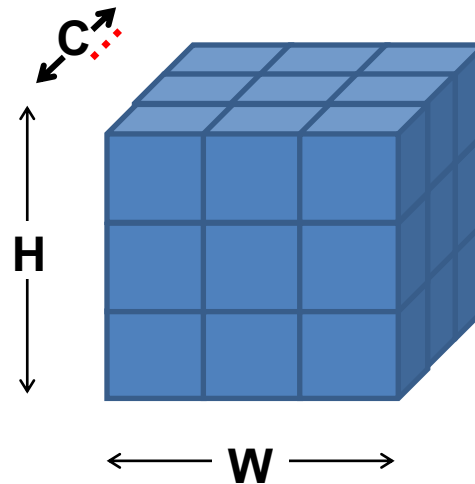
input fmap



$I[C][H][W]$

Example: Input Act/Fmap Tensor

input fmap

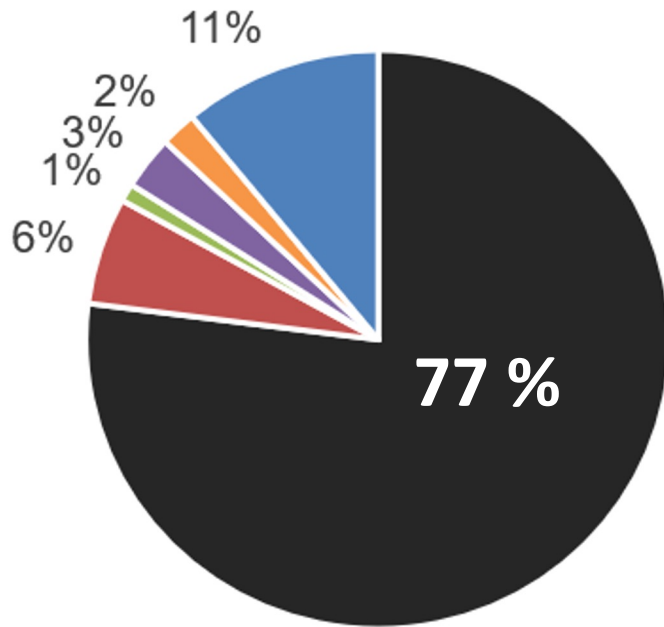


*The compiler
"lowers" high-
rank tensors into
appropriate HW
structures*

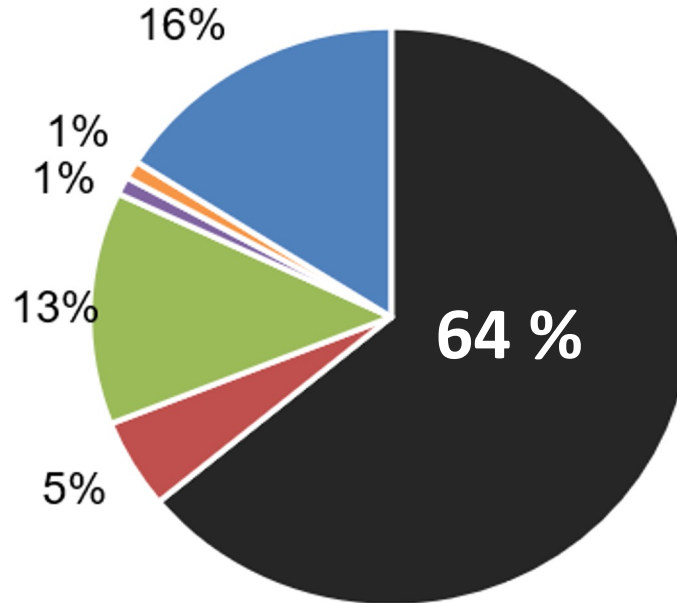
$I[C][H][W]$

What to accelerate?

What to accelerate?



Transformer
(Language Understanding)

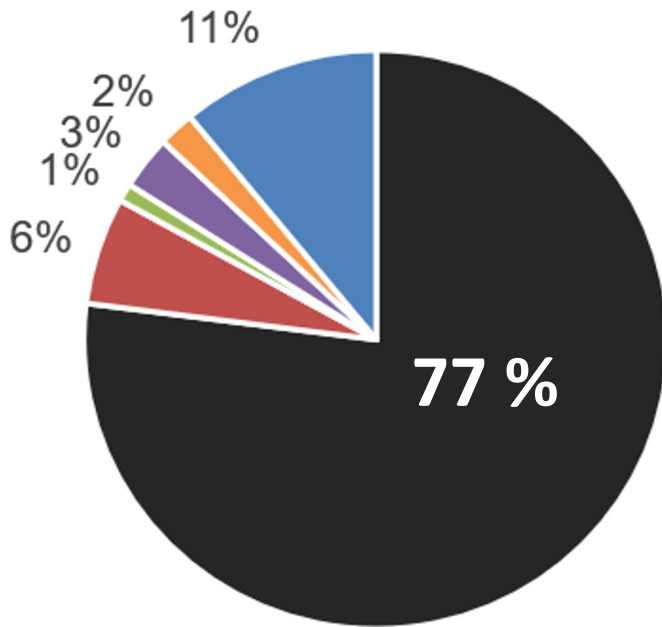


GNMT
(Machine Translation)

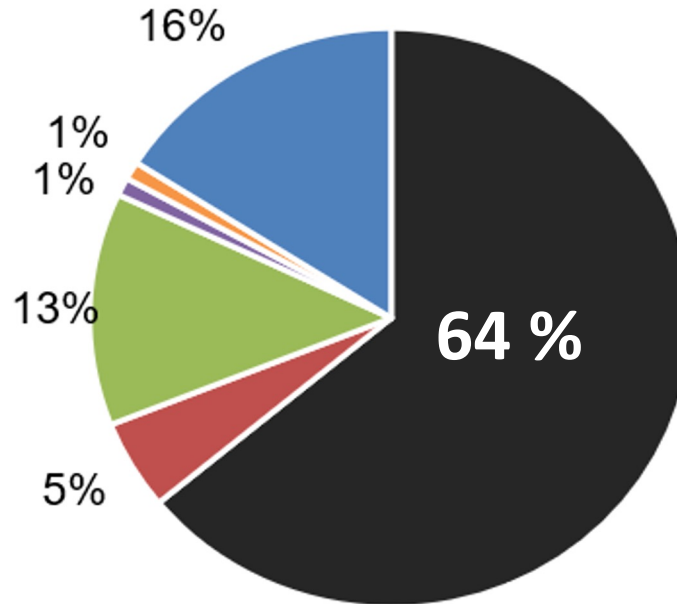
Runtime breakdown on V100 GPU

- MatMul
- Mul
- Add
- SoftmaxCrossEntropy
- BatchMatMul
- Rest

What to accelerate?



Transformer
(Language Understanding)



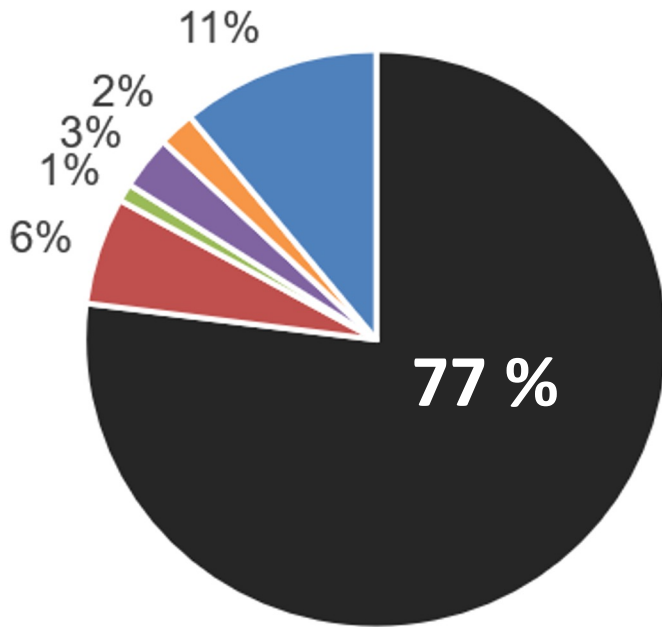
GNMT
(Machine Translation)

- MatMul
- Mul
- Add
- SoftmaxCrossEntropy
- BatchMatMul
- Rest

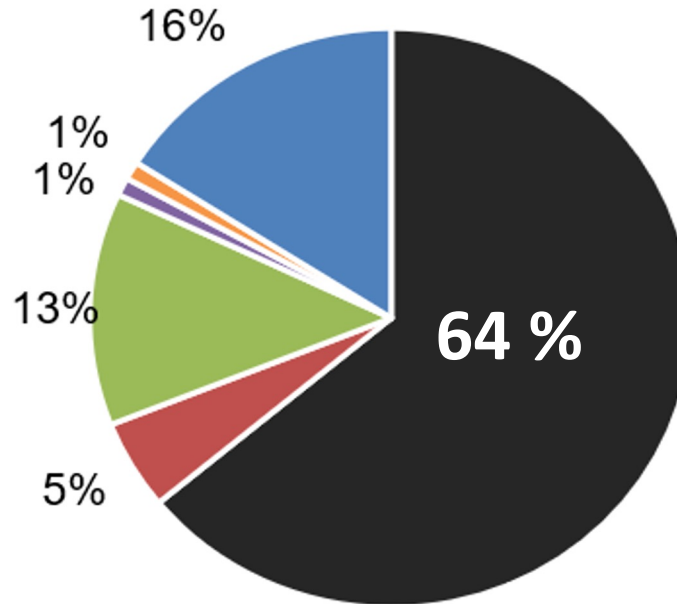
Runtime breakdown on V100 GPU

Matrix multiplications (GEMMs) consume around 70% of the total runtime on modern deep learning workloads.

What to accelerate?



Transformer
(Language Understanding)



GNMT
(Machine Translation)

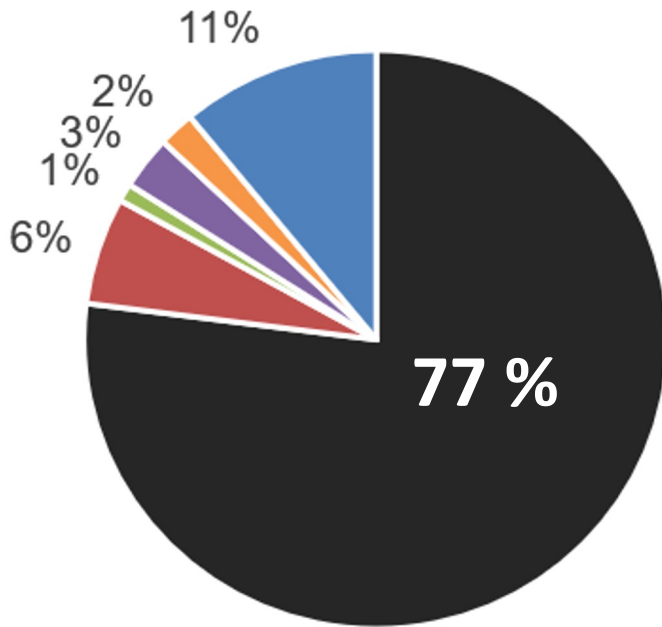
- MatMul
- Mul
- Add
- SoftmaxCrossEntropy
- BatchMatMul
- Rest

Runtime breakdown on V100 GPU

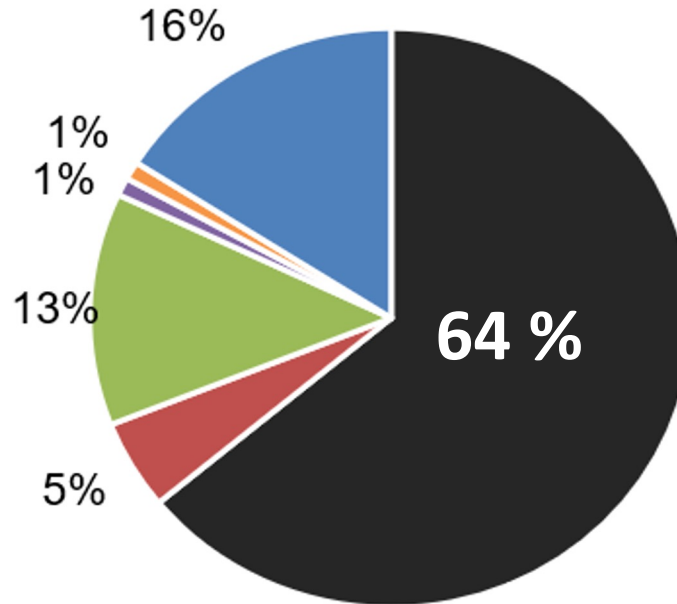
Matrix multiplications (GEMMs) consume around **70%** of the total runtime on modern deep learning workloads.

Prime candidate for acceleration

What to accelerate?



Transformer
(Language Understanding)



GNMT
(Machine Translation)

- MatMul
- Mul
- Add
- SoftmaxCrossEntropy
- BatchMatMul
- Rest

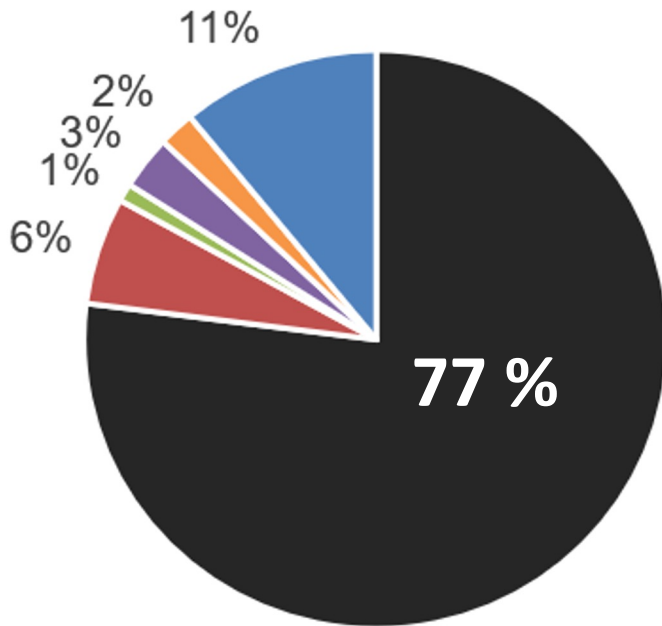
Runtime breakdown on V100 GPU

Matrix multiplications (GEMMs) consume around **70%** of the total runtime on modern deep learning workloads.

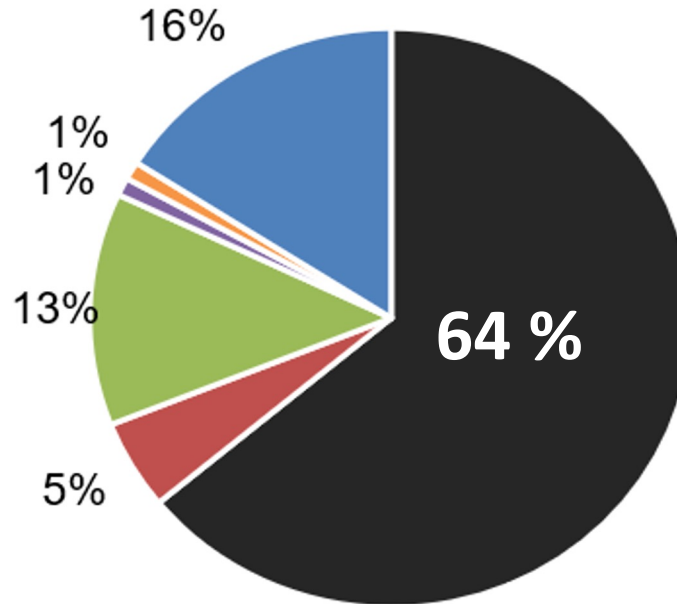
Prime candidate for acceleration

Possible Pitfall?

What to accelerate?



Transformer
(Language Understanding)



GNMT
(Machine Translation)

- MatMul
- Mul
- Add
- SoftmaxCrossEntropy
- BatchMatMul
- Rest

Runtime breakdown on V100 GPU

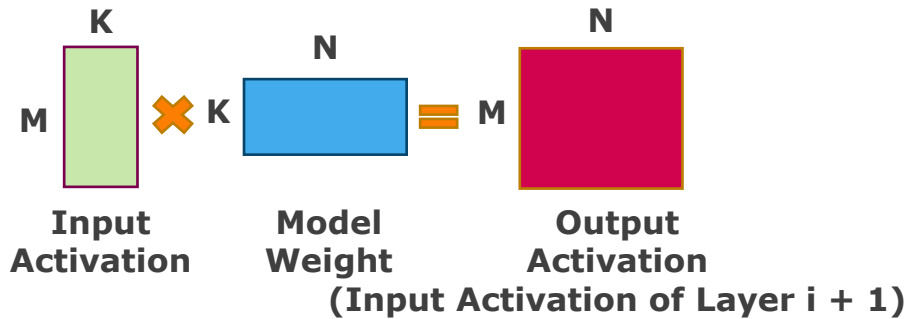
Matrix multiplications (GEMMs) consume around **70%** of the total runtime on modern deep learning workloads.

Prime candidate for acceleration

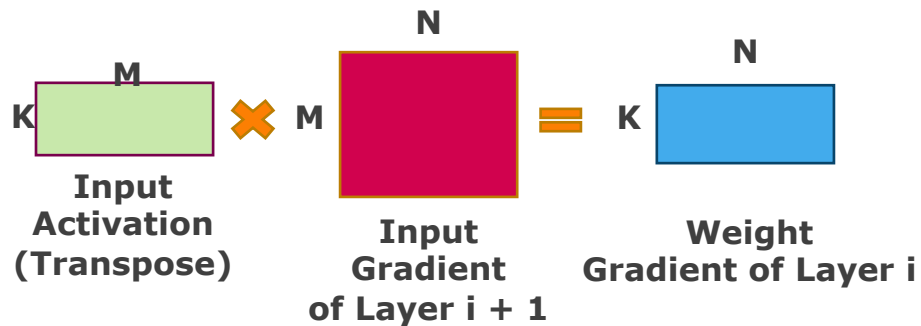
Possible Pitfall?

Beware of Amdahl's Law

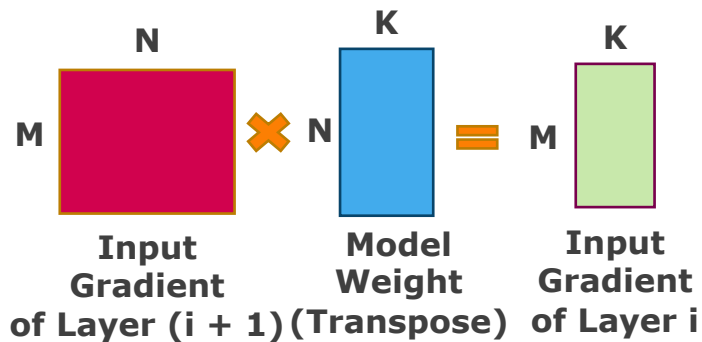
GEMMs in Deep Learning



Forward Pass



Backward Pass

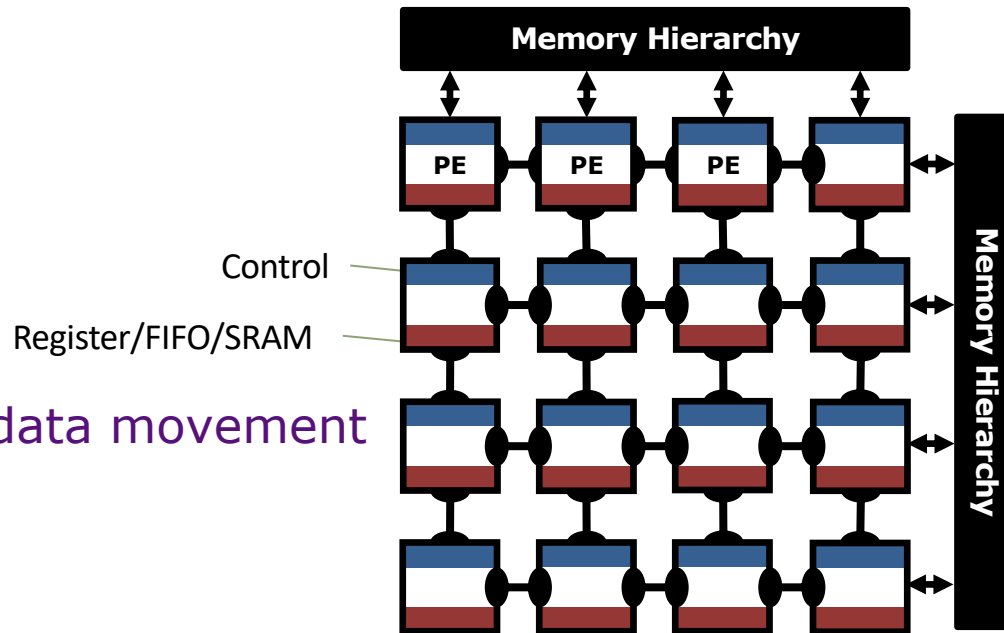


Spatial (or Dataflow) Accelerators

- **Millions of Parameters (i.e., weights)**

- Billions of computations

- Heavy data movement



Features

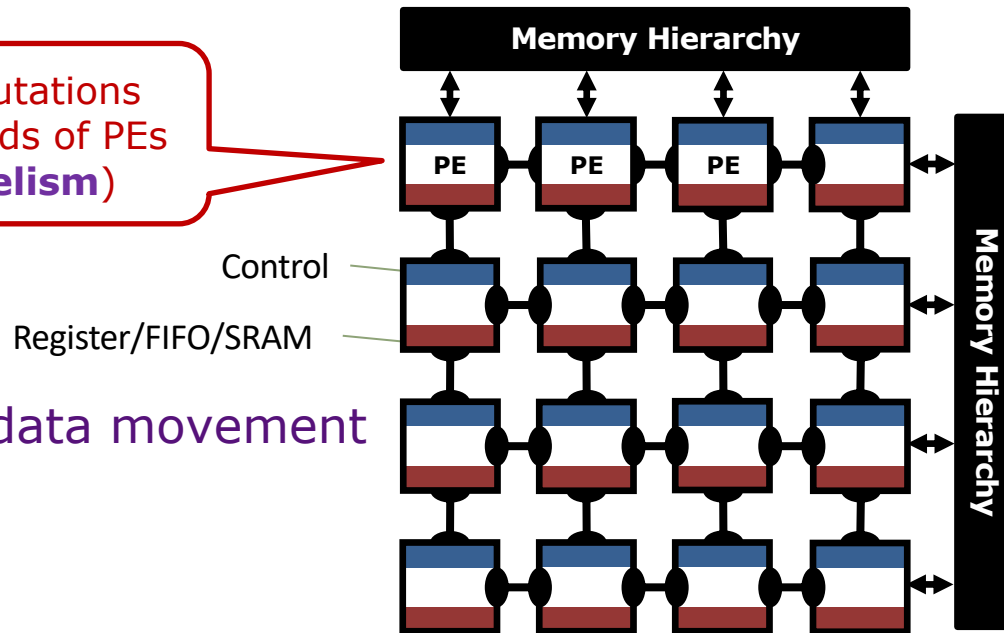
- Thousands of Processing Elements (PEs)
- Custom Memory Hierarchy (typically scratchpads, no caches)
- Custom NoCs

Spatial (or Dataflow) Accelerators

- **Millions of Parameters (i.e., weights)**

- Billions of computations

Spread computations
across thousands of PEs
(i.e., **parallelism**)



- Heavy data movement

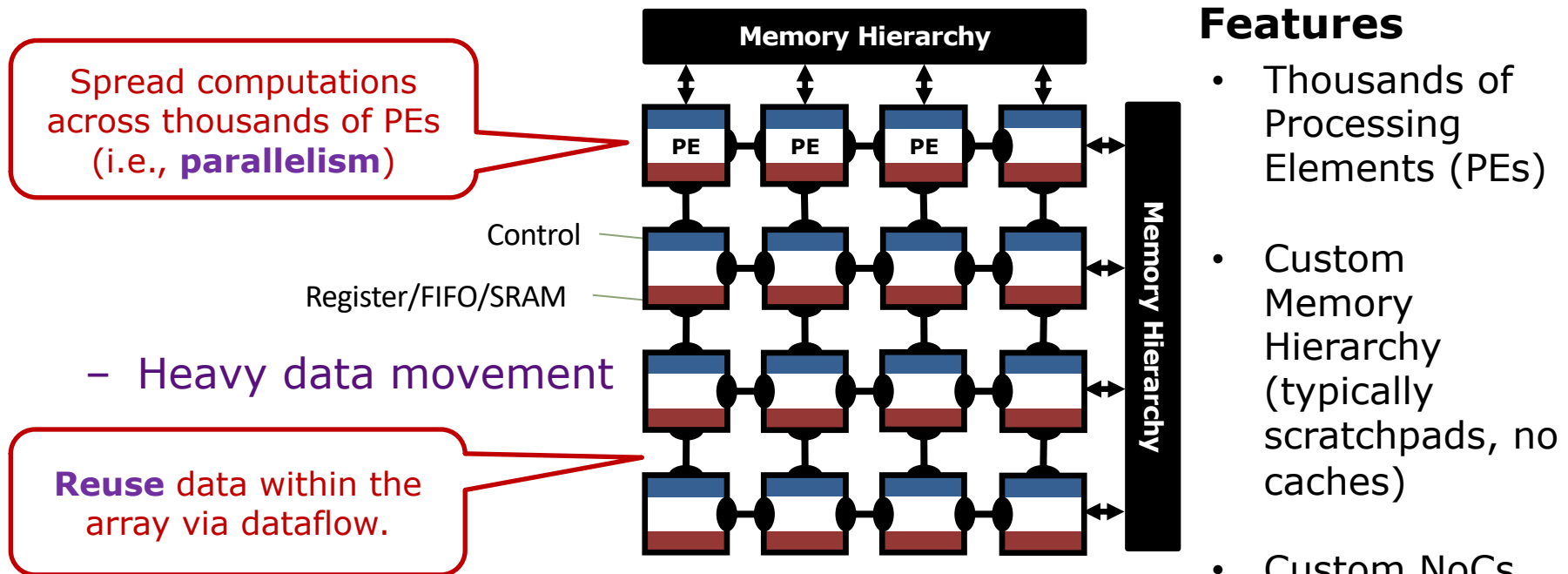
Features

- Thousands of Processing Elements (PEs)
- Custom Memory Hierarchy (typically scratchpads, no caches)
- Custom NoCs

Spatial (or Dataflow) Accelerators

- **Millions of Parameters (i.e., weights)**

- Billions of computations



What is Reuse?

| | | | |
|-----|-----|-----|-----|
| A1 | A2 | A3 | A4 |
| A5 | A6 | A7 | A8 |
| A9 | A10 | A11 | A12 |
| A13 | A14 | A15 | A16 |

Matrix A

x

| | | | |
|-----|-----|-----|-----|
| B1 | B2 | B3 | B4 |
| B5 | B6 | B7 | B8 |
| B9 | B10 | B11 | B12 |
| B13 | B14 | B15 | B16 |

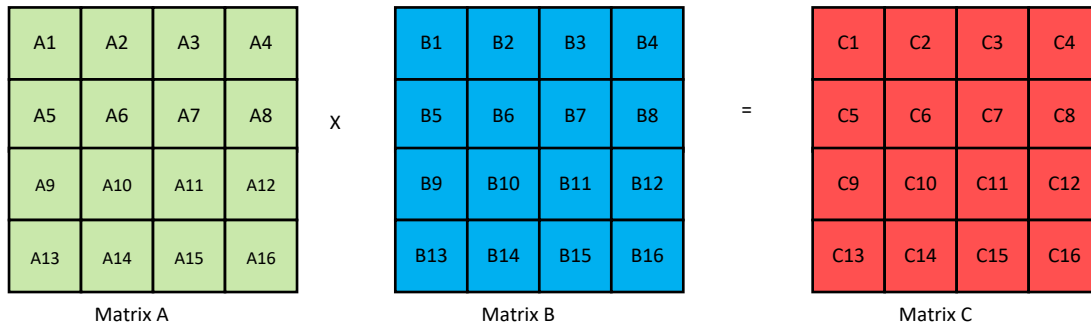
Matrix B

=

| | | | |
|-----|-----|-----|-----|
| C1 | C2 | C3 | C4 |
| C5 | C6 | C7 | C8 |
| C9 | C10 | C11 | C12 |
| C13 | C14 | C15 | C16 |

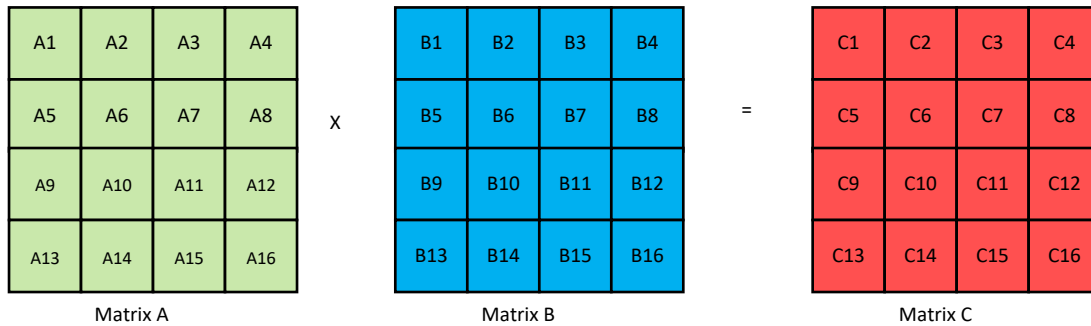
Matrix C

What is Reuse?

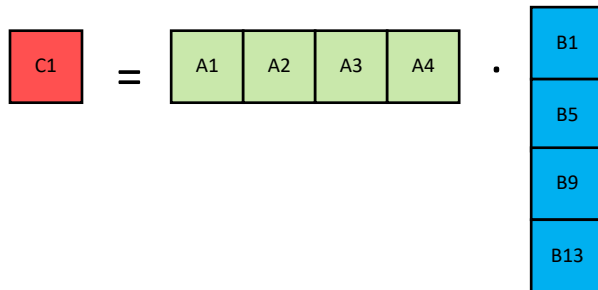


Example Operations

What is Reuse?



Example Operations

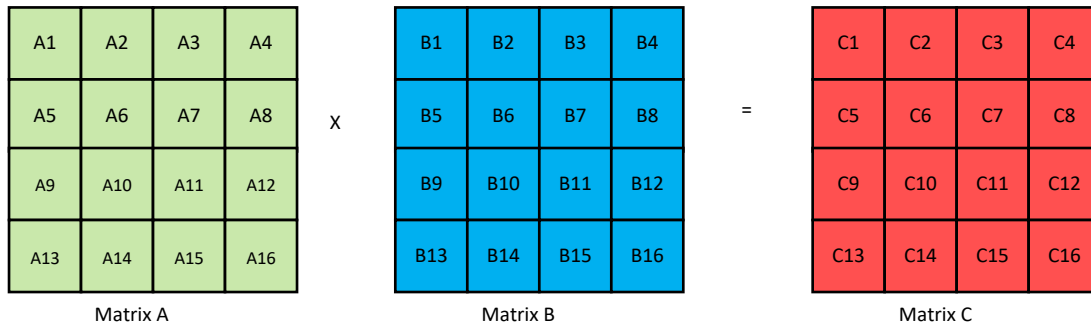


| |
|----|
| C1 |
|----|

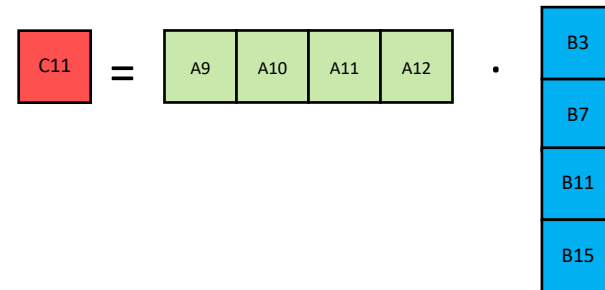
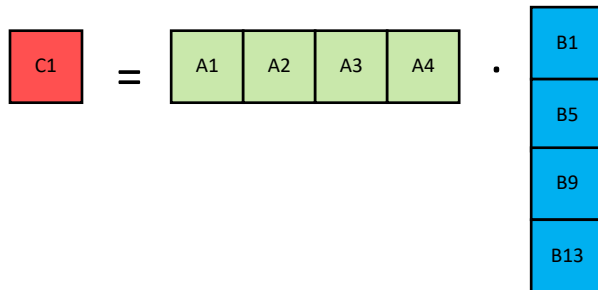
=

$$A1 * B1 + A2 * B5 + A3 * B9 + A4 * B13$$

What is Reuse?



Example Operations



| |
|----|
| C1 |
|----|

=

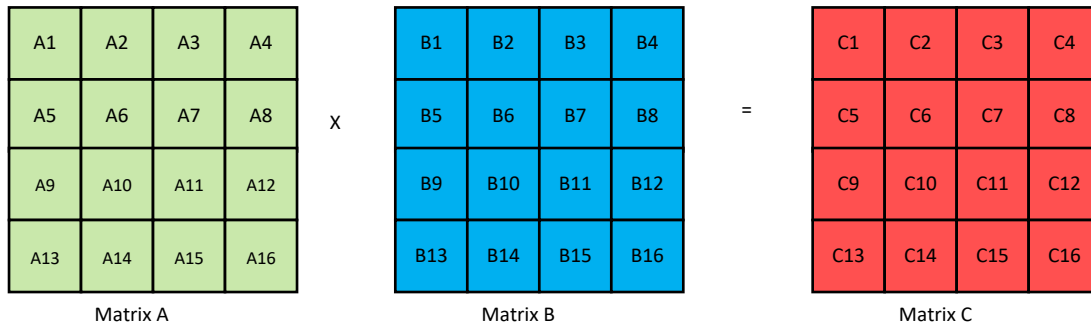
$$A1 \cdot B1 + A2 \cdot B5 + A3 \cdot B9 + A4 \cdot B13$$

| |
|-----|
| C11 |
|-----|

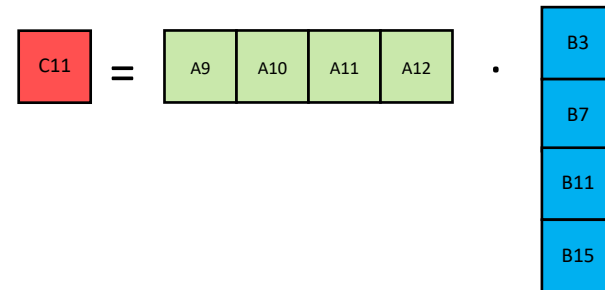
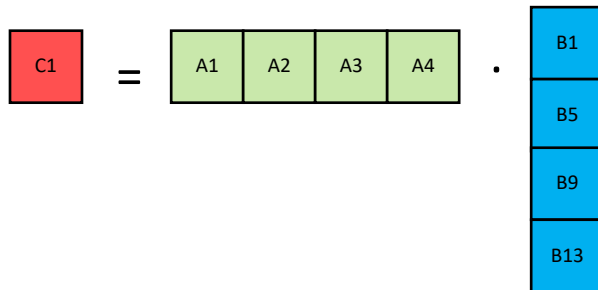
=

$$A9 \cdot B3 + A10 \cdot B7 + A11 \cdot B11 + A12 \cdot B15$$

What is Reuse?



Example Operations



| |
|----|
| C1 |
|----|

=
 $A1 \cdot B1 + A2 \cdot B5 + A3 \cdot B9 + A4 \cdot B13$

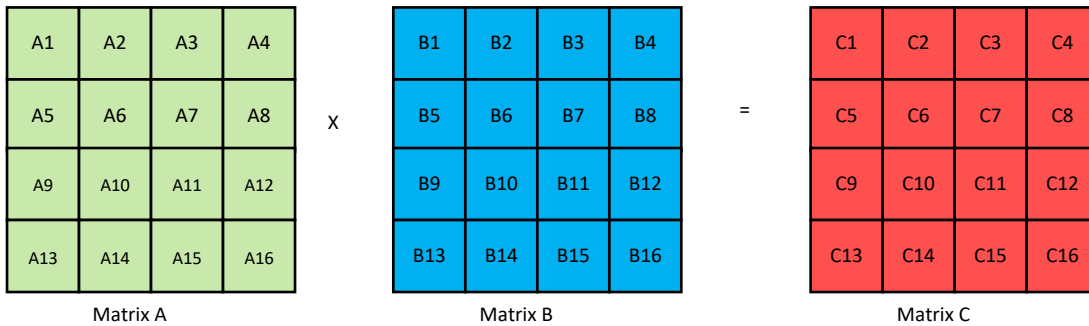
| |
|-----|
| C11 |
|-----|

=
 $A9 \cdot B3 + A10 \cdot B7 + A11 \cdot B11 + A12 \cdot B15$

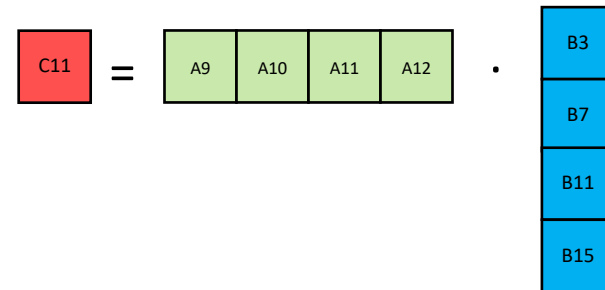
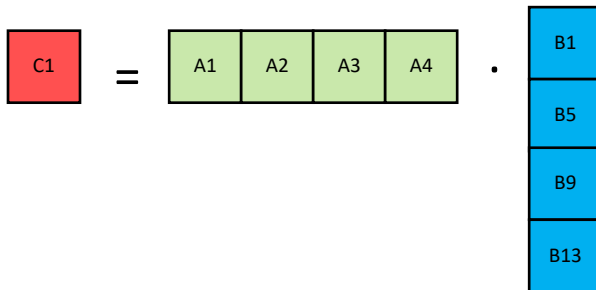
| |
|----|
| C5 |
|----|

=

What is Reuse?



Example Operations



| |
|----|
| C1 |
|----|

=
 $A1 \cdot B1 + A2 \cdot B5 + A3 \cdot B9 + A4 \cdot B13$

| |
|-----|
| C11 |
|-----|

=
 $A9 \cdot B3 + A10 \cdot B7 + A11 \cdot B11 + A12 \cdot B15$

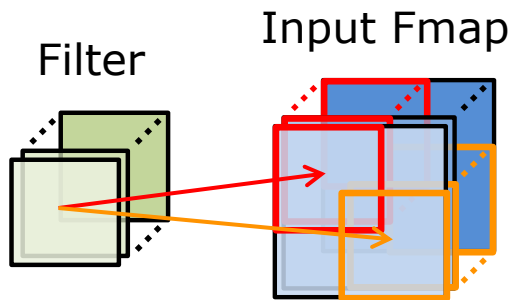
| |
|----|
| C5 |
|----|

=
 $A5 \cdot B1 + A6 \cdot B5 + A7 \cdot B9 + A8 \cdot B13$

Examples of Data Reuse in DNN

Convolutional Reuse

CONV layers only
(sliding window)

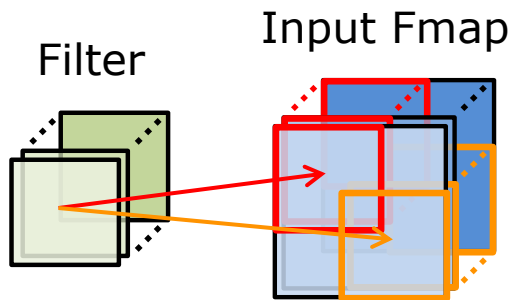


Reuse: **Activations**
Filter weights

Examples of Data Reuse in DNN

Convolutional Reuse

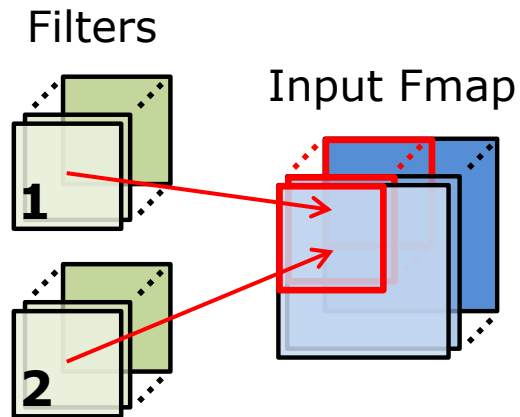
CONV layers only
(sliding window)



Reuse: **Activations**
Filter weights

Fmap Reuse

CONV and FC layers

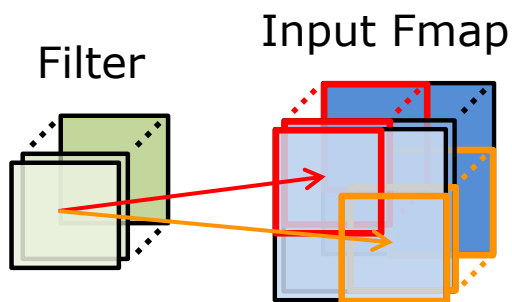


Reuse: **Activations**

Examples of Data Reuse in DNN

Convolutional Reuse

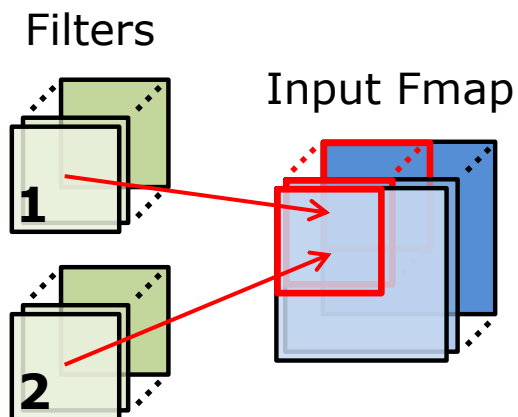
CONV layers only
(sliding window)



Reuse: **Activations**
Filter weights

Fmap Reuse

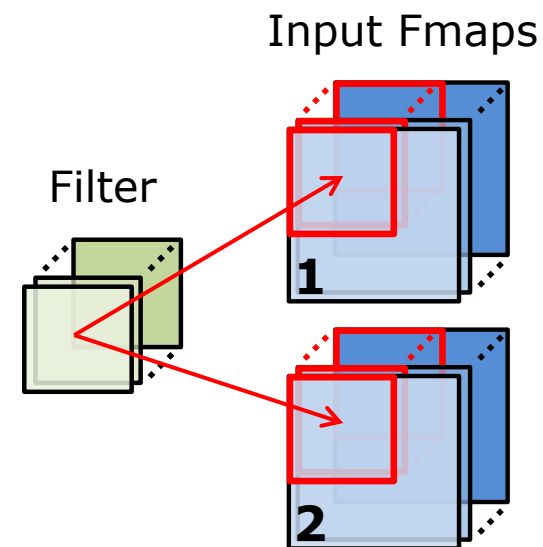
CONV and FC layers



Reuse: **Activations**

Filter Reuse

CONV and FC layers
(batch size > 1)



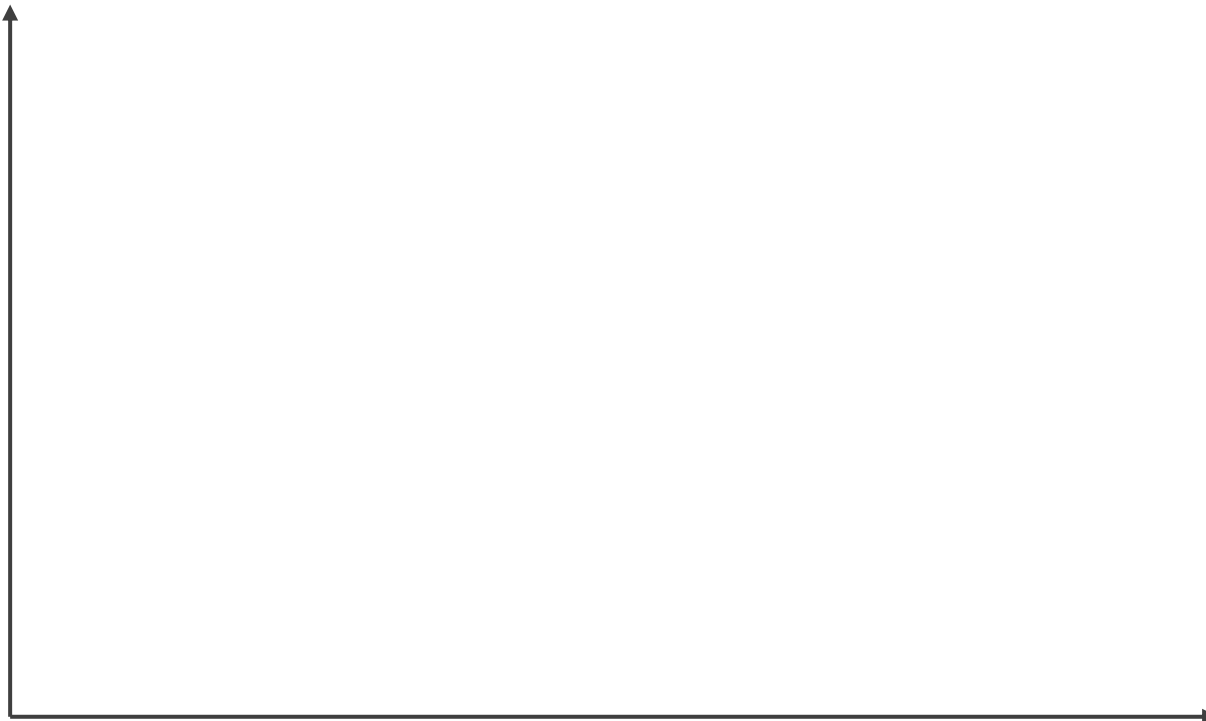
Reuse: **Filter weights**

Why does reuse help?

Why does reuse help?

**Attainable
Performance
(GFLOPS)**

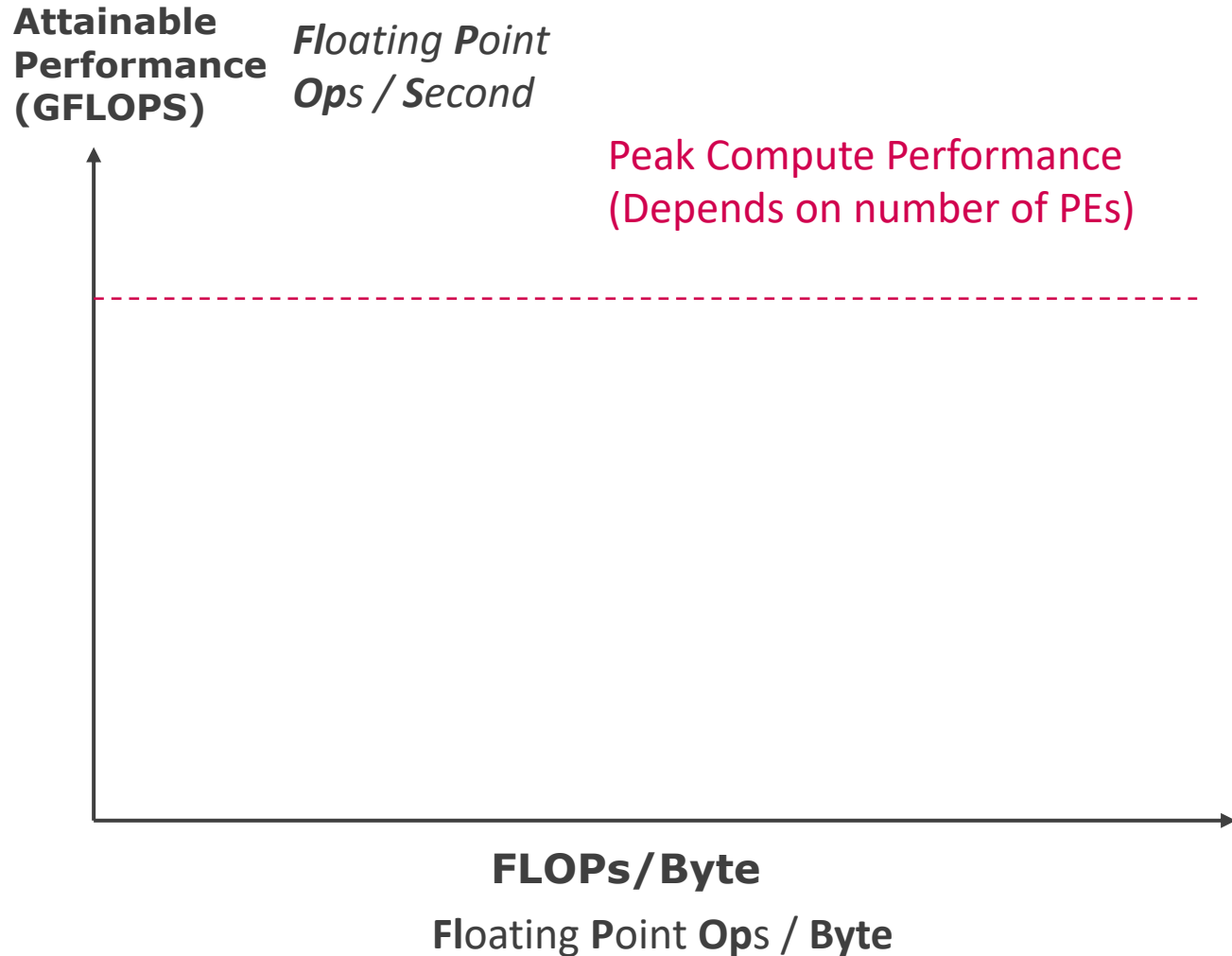
*Floating Point
Ops / Second*



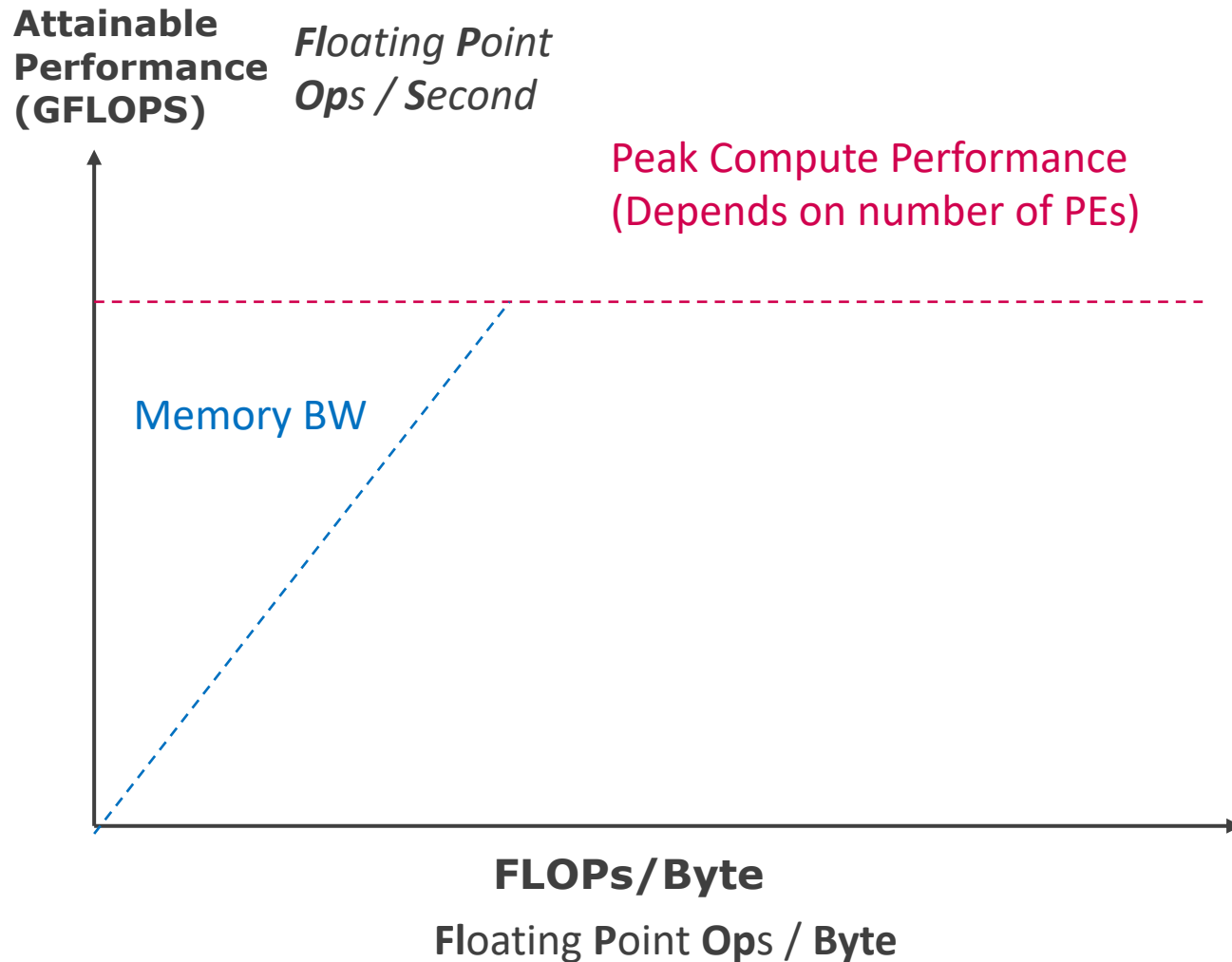
FLOPs/Byte

Floating Point Ops / Byte

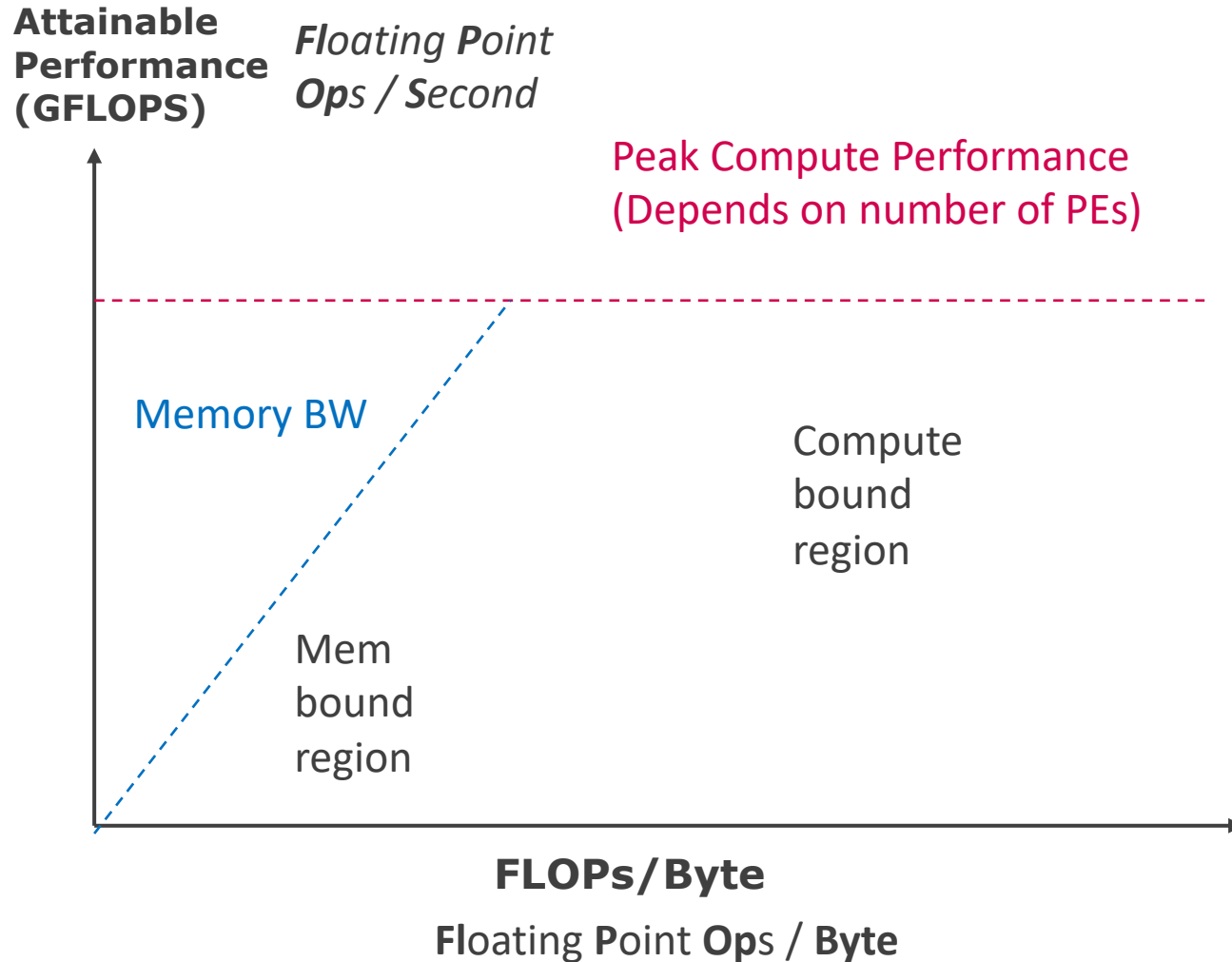
Why does reuse help?



Why does reuse help?



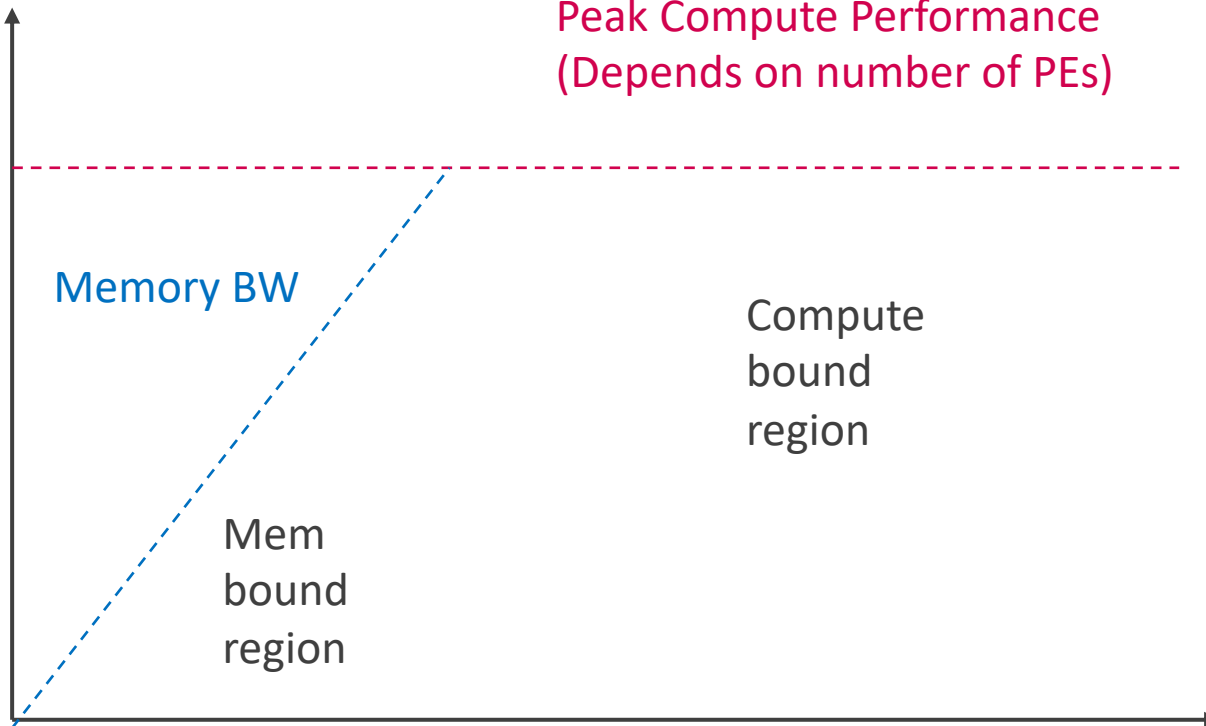
Why does reuse help?



Why does reuse help?

Attainable Performance (GFLOPS)

Floating Point Ops / Second



Peak Compute Performance
(Depends on number of PEs)

Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Memory BW

Compute bound region

Mem bound region

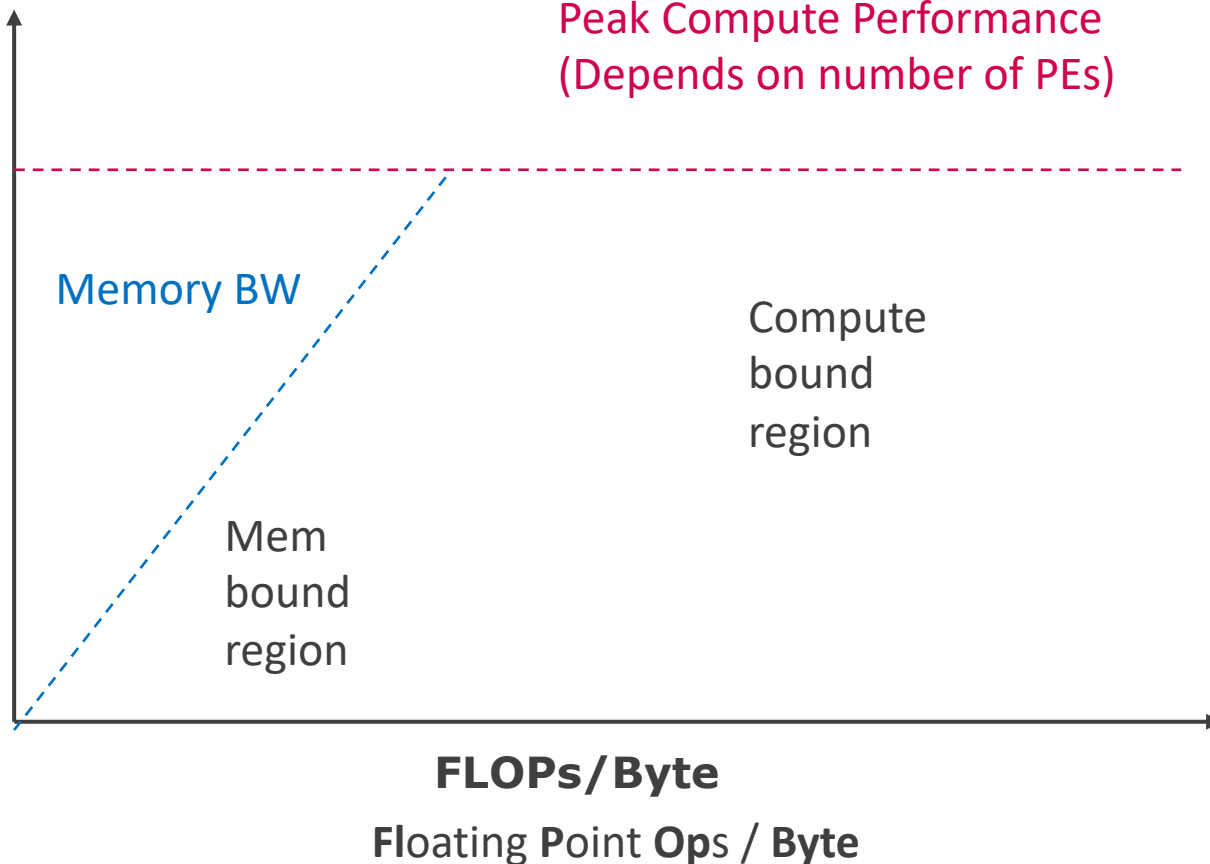
FLOPs/Byte

Floating Point Ops / Byte

Why does reuse help?

Attainable
Performance
(GFLOPS)

*Floating Point
Ops / Second*



Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

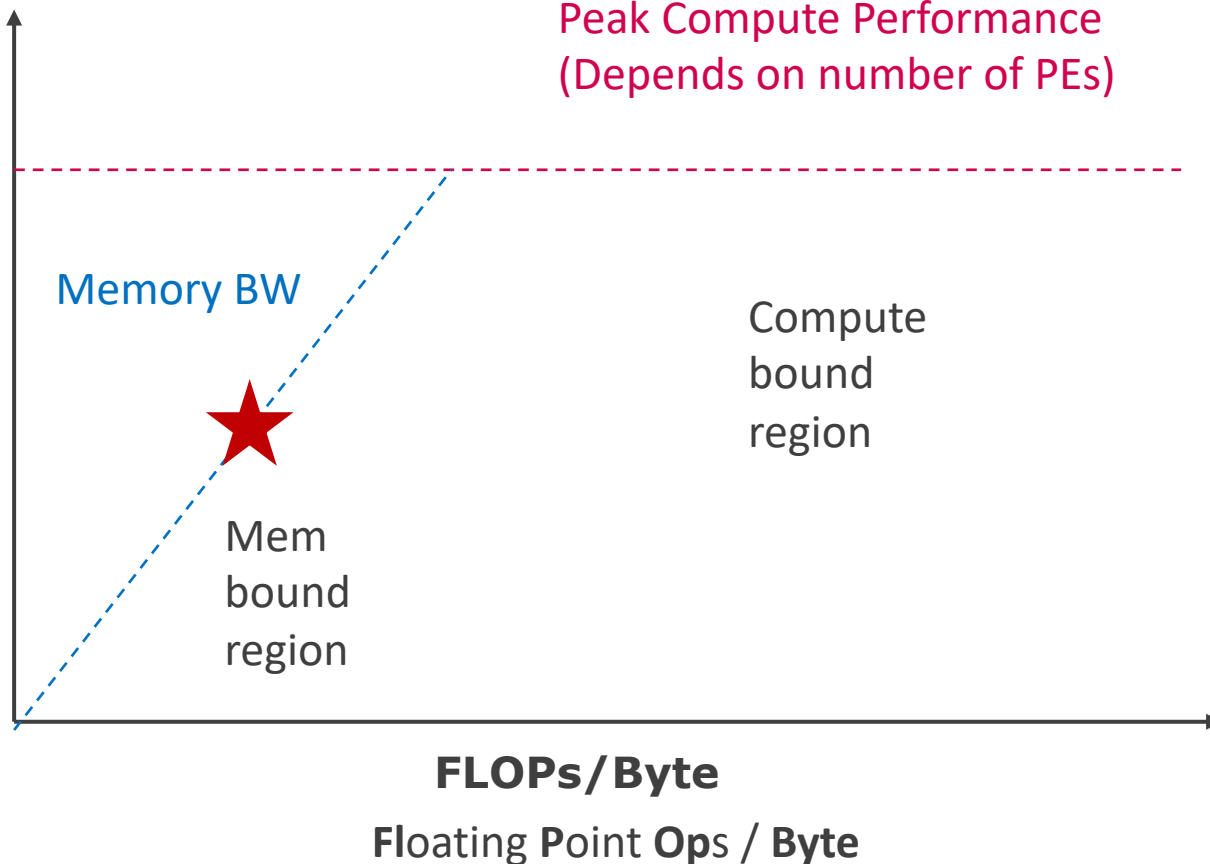
Simple Accelerator:

BW requirement: 3
bytes/cycle => 300 GBps
--> Mem Bound!

Why does reuse help?

Attainable Performance (GFLOPS)

Floating Point Ops / Second



Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

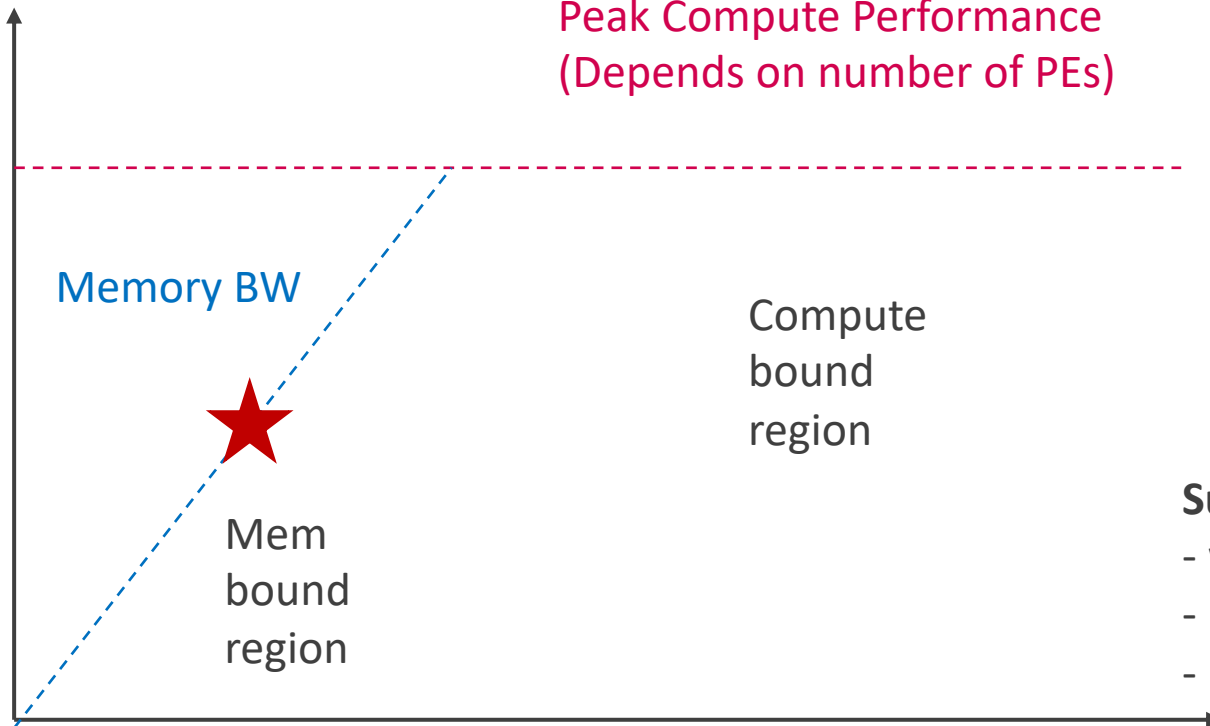
Simple Accelerator:

BW requirement: 3 bytes/cycle => 300 GBps
--> Mem Bound!

Why does reuse help?

Attainable
Performance
(GFLOPS)

*Floating Point
Ops / Second*



Peak Compute Performance
(Depends on number of PEs)

Memory BW

Compute
bound
region

Mem
bound
region

FLOPs/Byte

Floating Point Ops / Byte

Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Simple Accelerator:

BW requirement: 3
bytes/cycle => 300 GBps
--> Mem Bound!

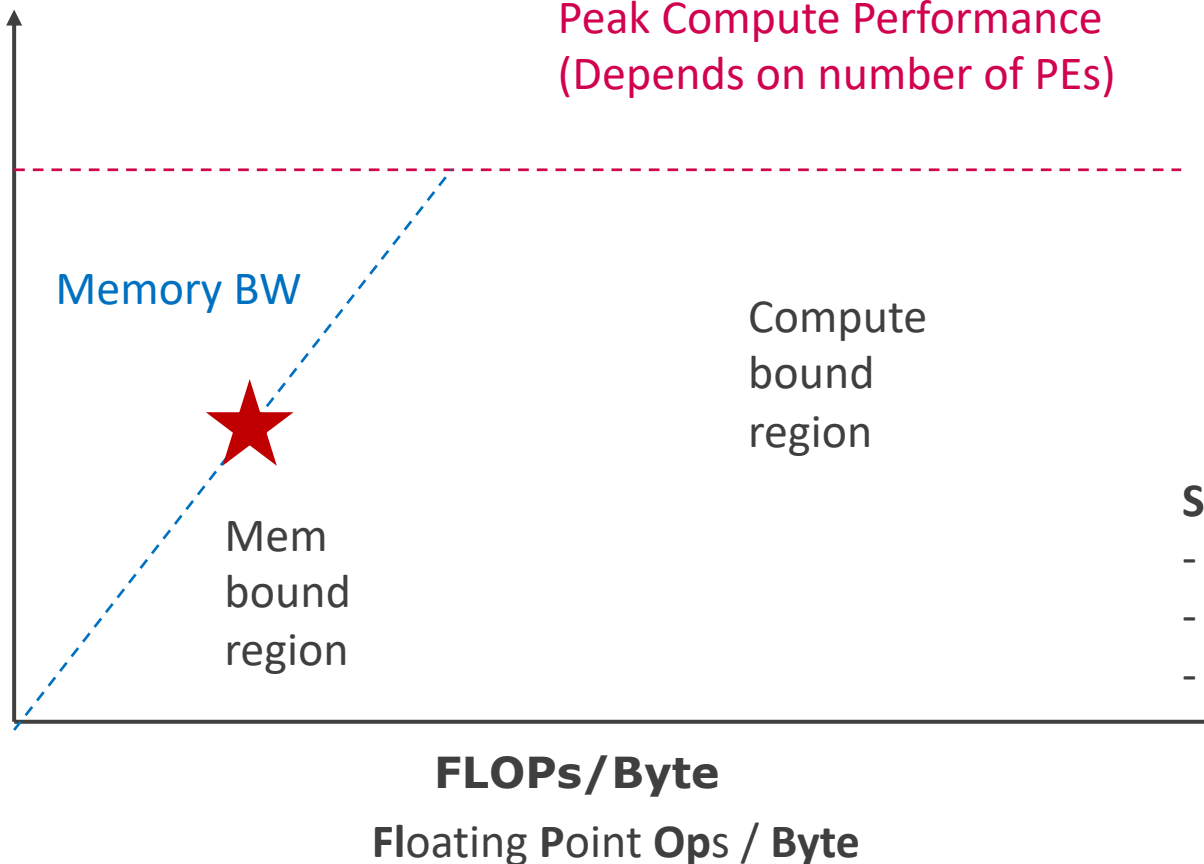
Suppose we have data reuse

- weight reused completely
- input reused for 10 cycles
- psums reused for 10 cycles

Why does reuse help?

Attainable Performance (GFLOPS)

Floating Point Ops / Second



Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Simple Accelerator:

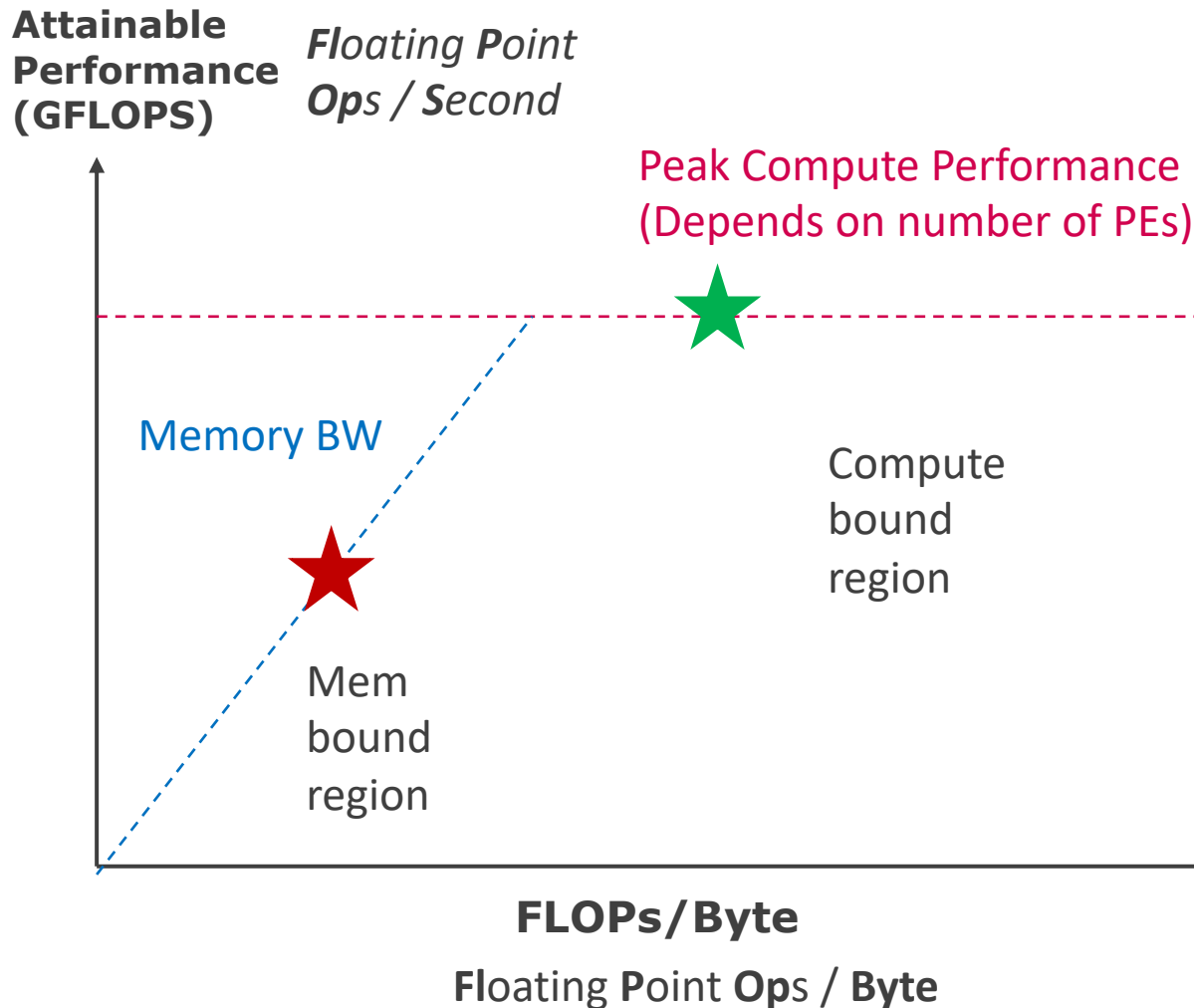
BW requirement: 3 bytes/cycle => 300 GBps
--> Mem Bound!

Suppose we have data reuse

- weight reused completely
- input reused for 10 cycles
- psums reused for 10 cycles

BW requirement: 20 GBps
→ Compute Bound

Why does reuse help?



Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Simple Accelerator:

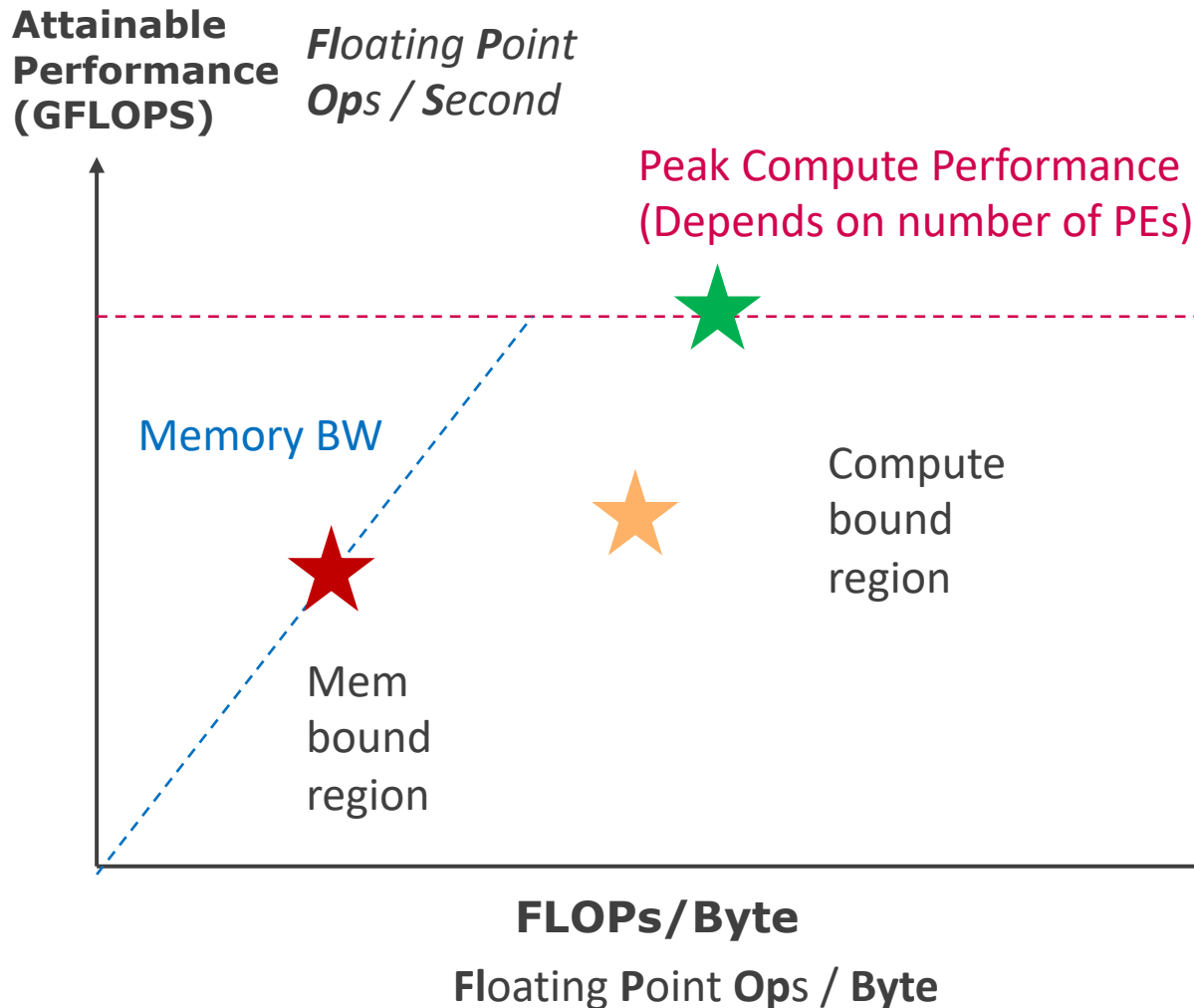
BW requirement: 3 bytes/cycle => 300 GBps
--> Mem Bound!

Suppose we have data reuse

- weight reused completely
- input reused for 10 cycles
- psums reused for 10 cycles

BW requirement: 20 GBps
→ Compute Bound

Why does reuse help?



Suppose

- 100 PEs operating at 1 GHz => 100 GFLOPs/sec
- Each PE needs 2 bytes of read and 1 byte of write.
- DRAM BW ~25 GBps

Simple Accelerator:

BW requirement: 3 bytes/cycle => 300 GBps
--> Mem Bound!

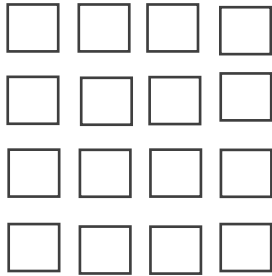
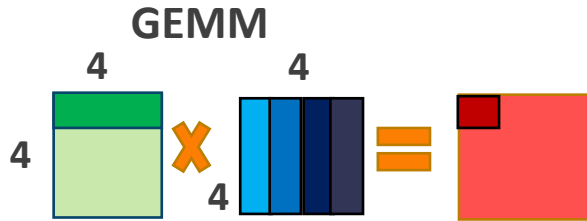
Suppose we have data reuse

- weight reused completely
- input reused for 10 cycles
- psums reused for 10 cycles

BW requirement: 20 GBps
→ Compute Bound

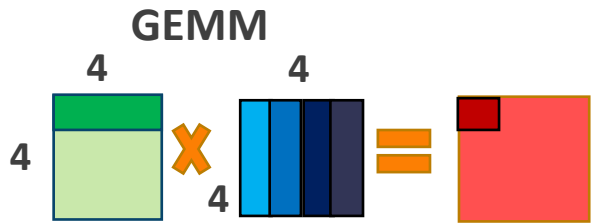
Realistic?

How to exploit Reuse?

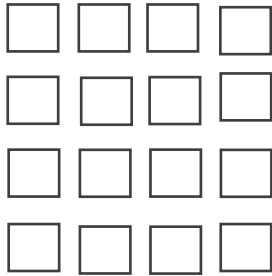


Accelerator

How to exploit Reuse?

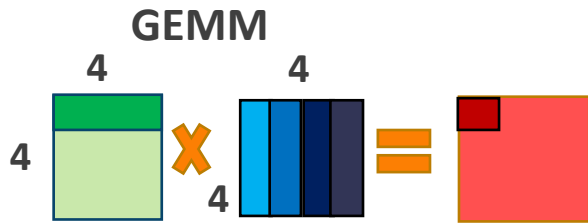


↓
“Mapping”

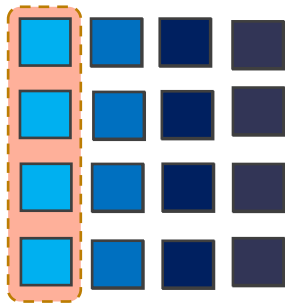


Accelerator

How to exploit Reuse?

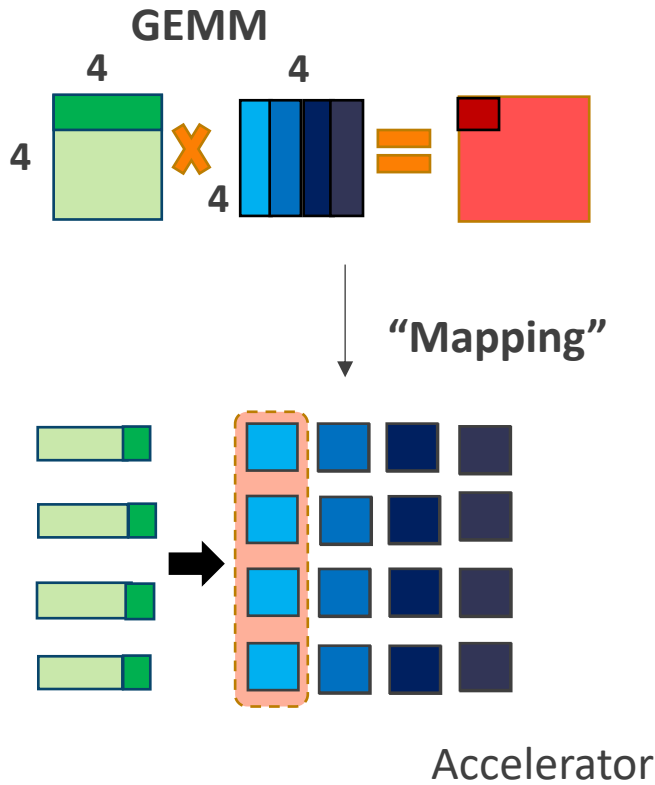


“Mapping”

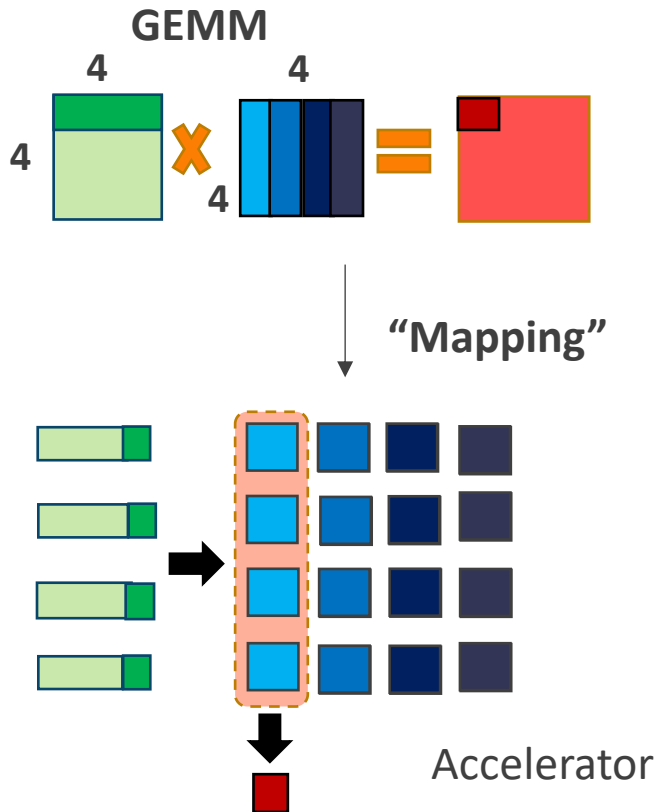


Accelerator

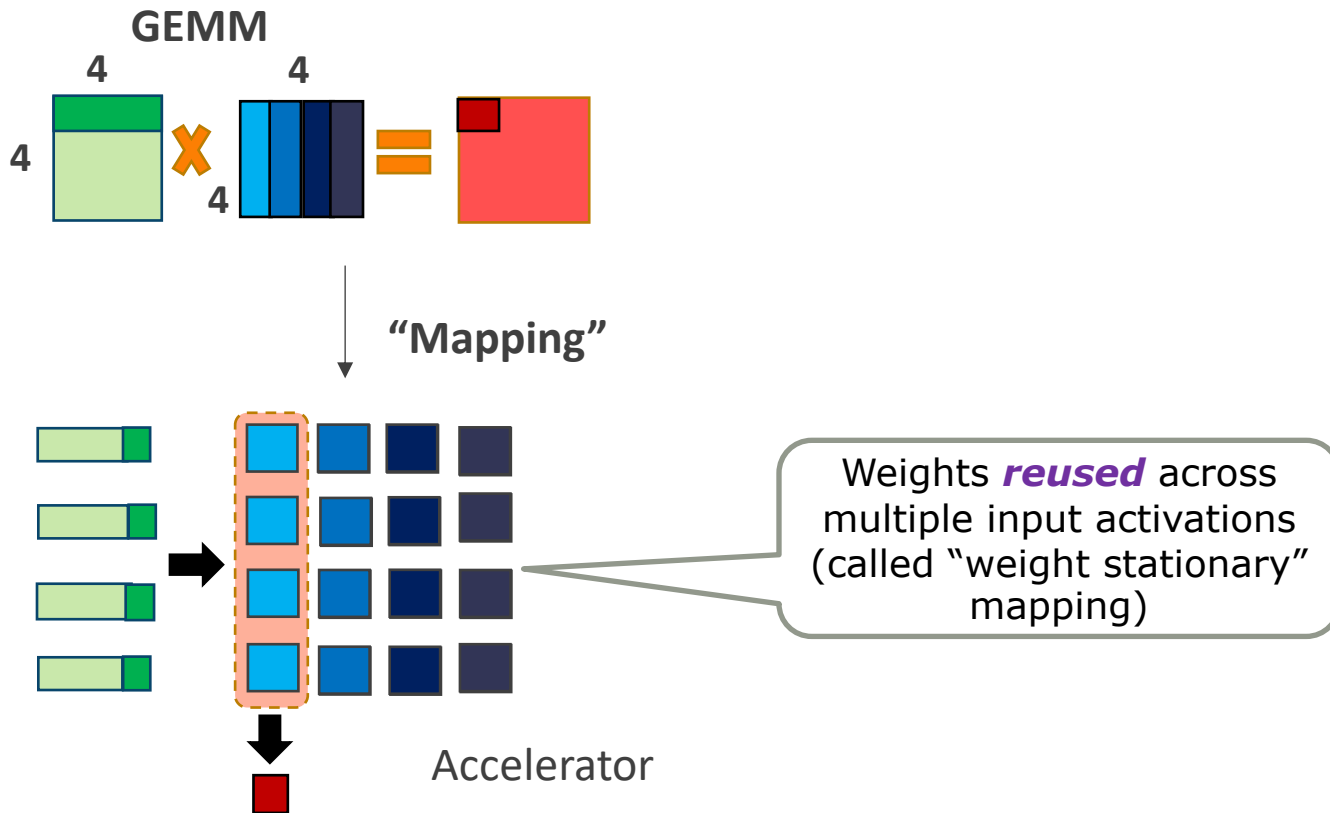
How to exploit Reuse?



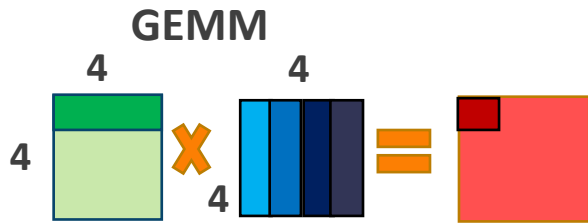
How to exploit Reuse?



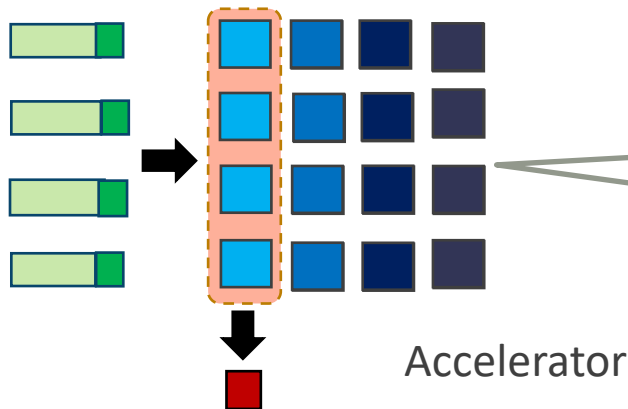
How to exploit Reuse?



How to exploit Reuse?



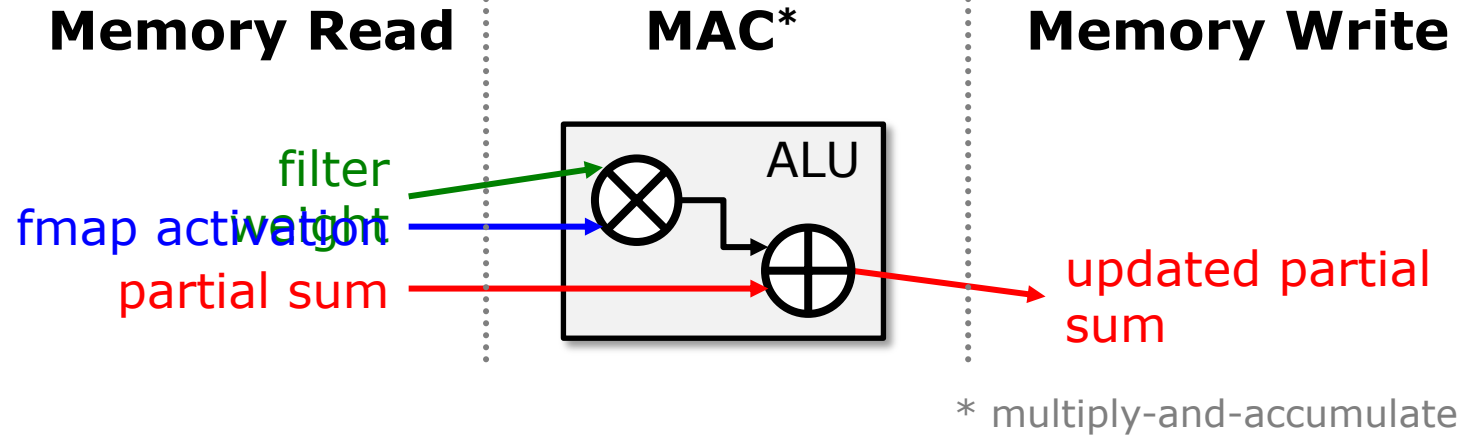
“Mapping”



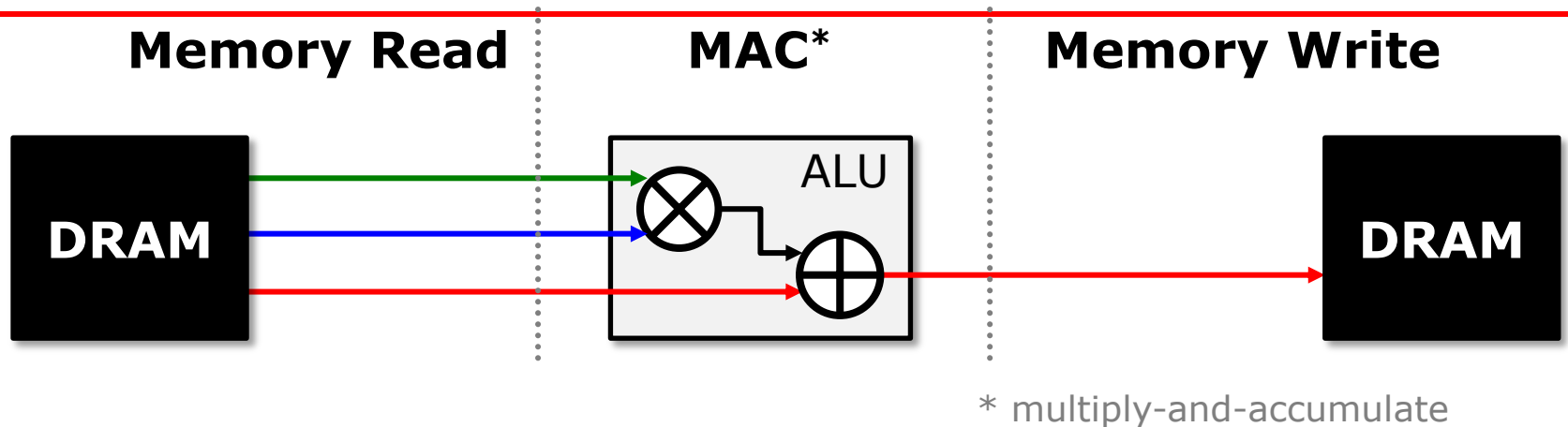
Weights *reused* across multiple input activations (called “weight stationary” mapping)

What HW structures would you need?

Building a DNN Accelerator



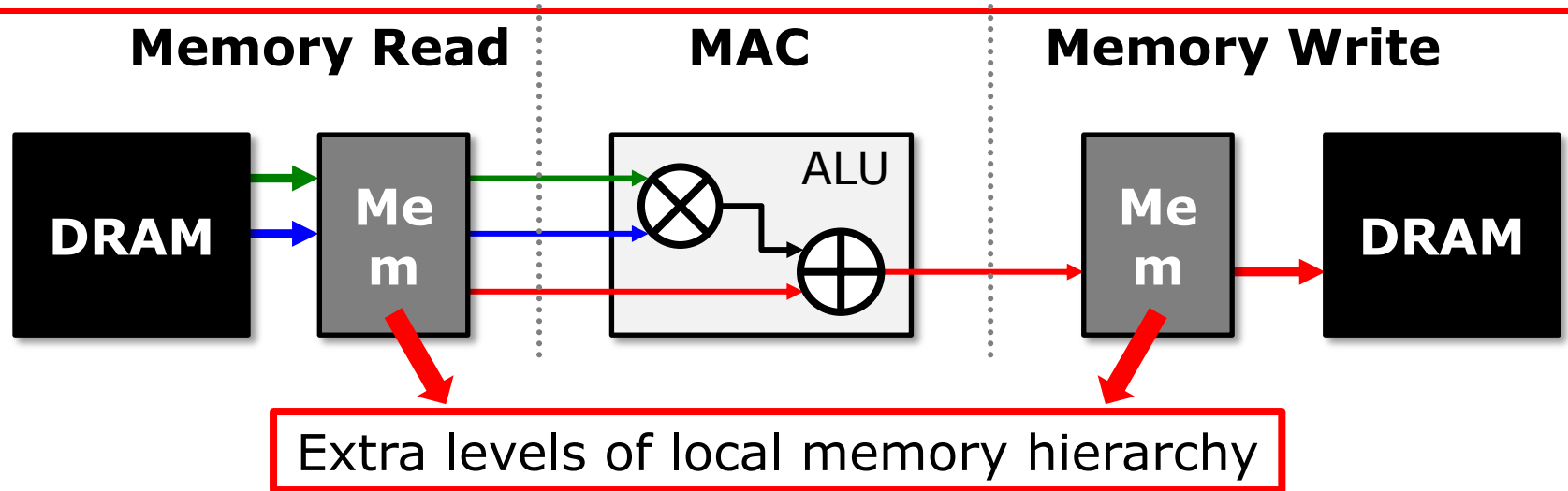
Building a DNN Accelerator



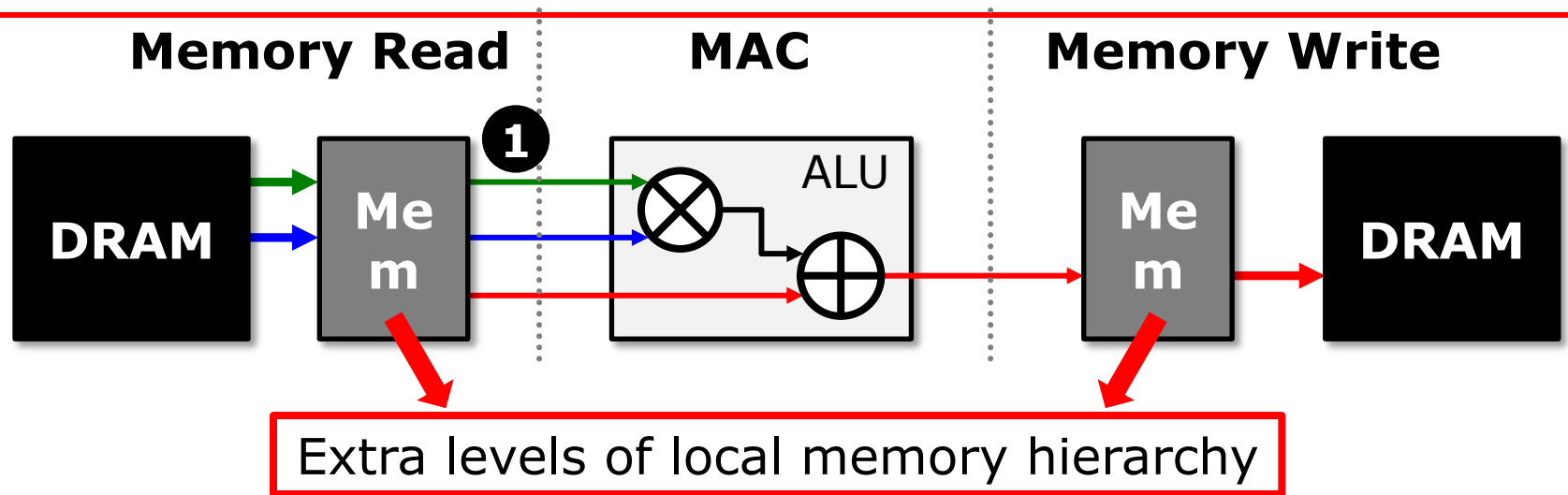
Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NeurIPS 2012] has **724M** MACs
→ **2896M** DRAM accesses required

Building a DNN Accelerator

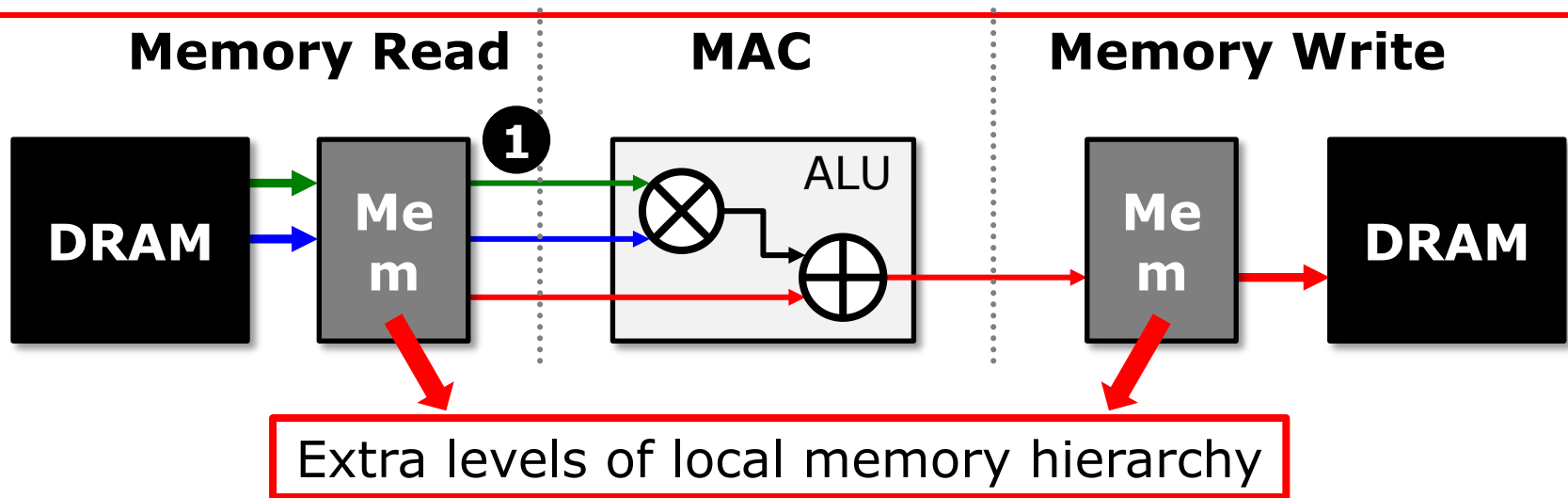


Building a DNN Accelerator



Opportunities: **1 data reuse**

Building a DNN Accelerator

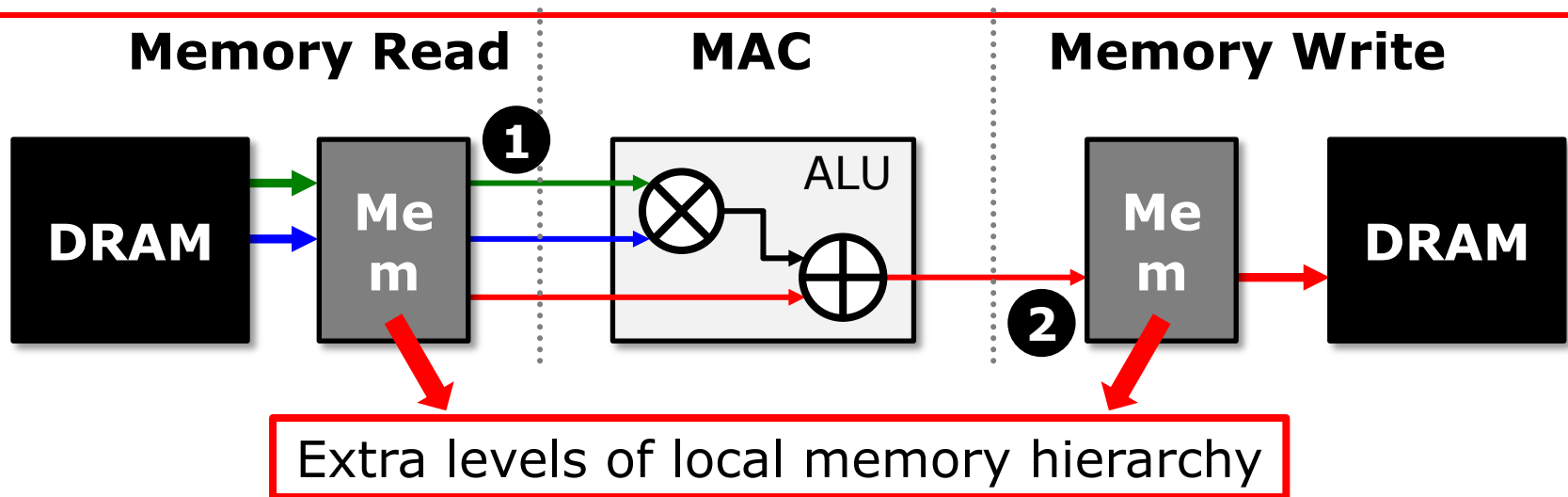


Opportunities: **1** data reuse

- 1** Can reduce DRAM reads of **filter/fmap** by up to **500x****

** AlexNet CONV layers

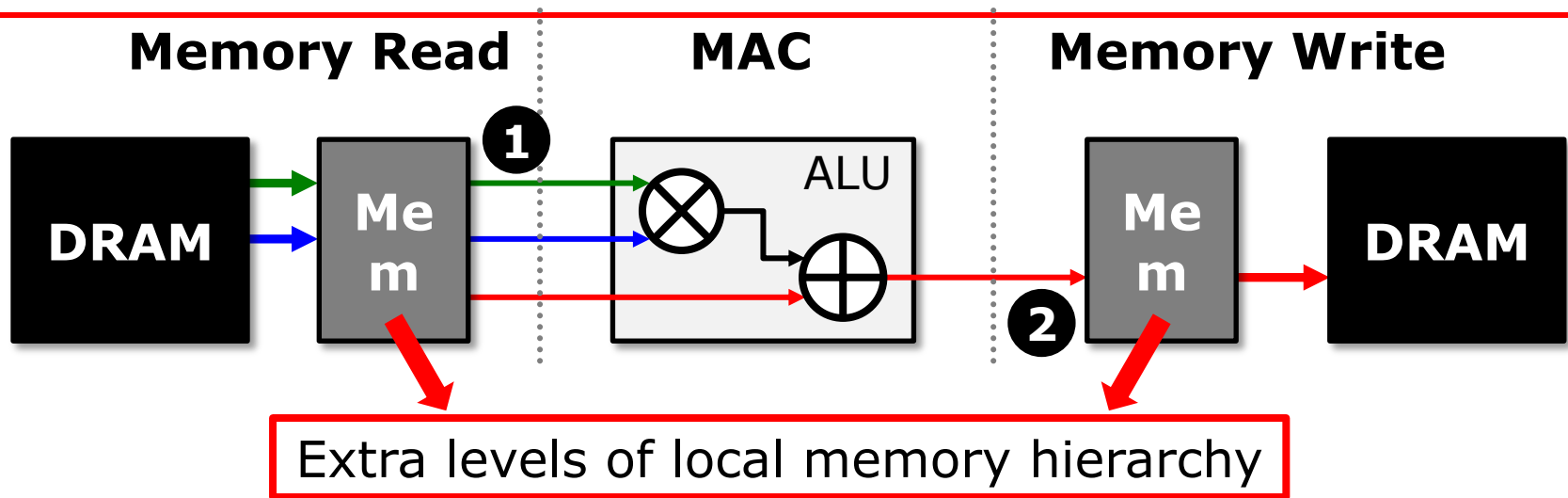
Building a DNN Accelerator



Opportunities: **1 data reuse** **2 local accumulation**

- 1** Can reduce DRAM reads of **filter/fmap** by up to **500×**
- 2** **Partial sum** accumulation does **NOT** have to access DRAM

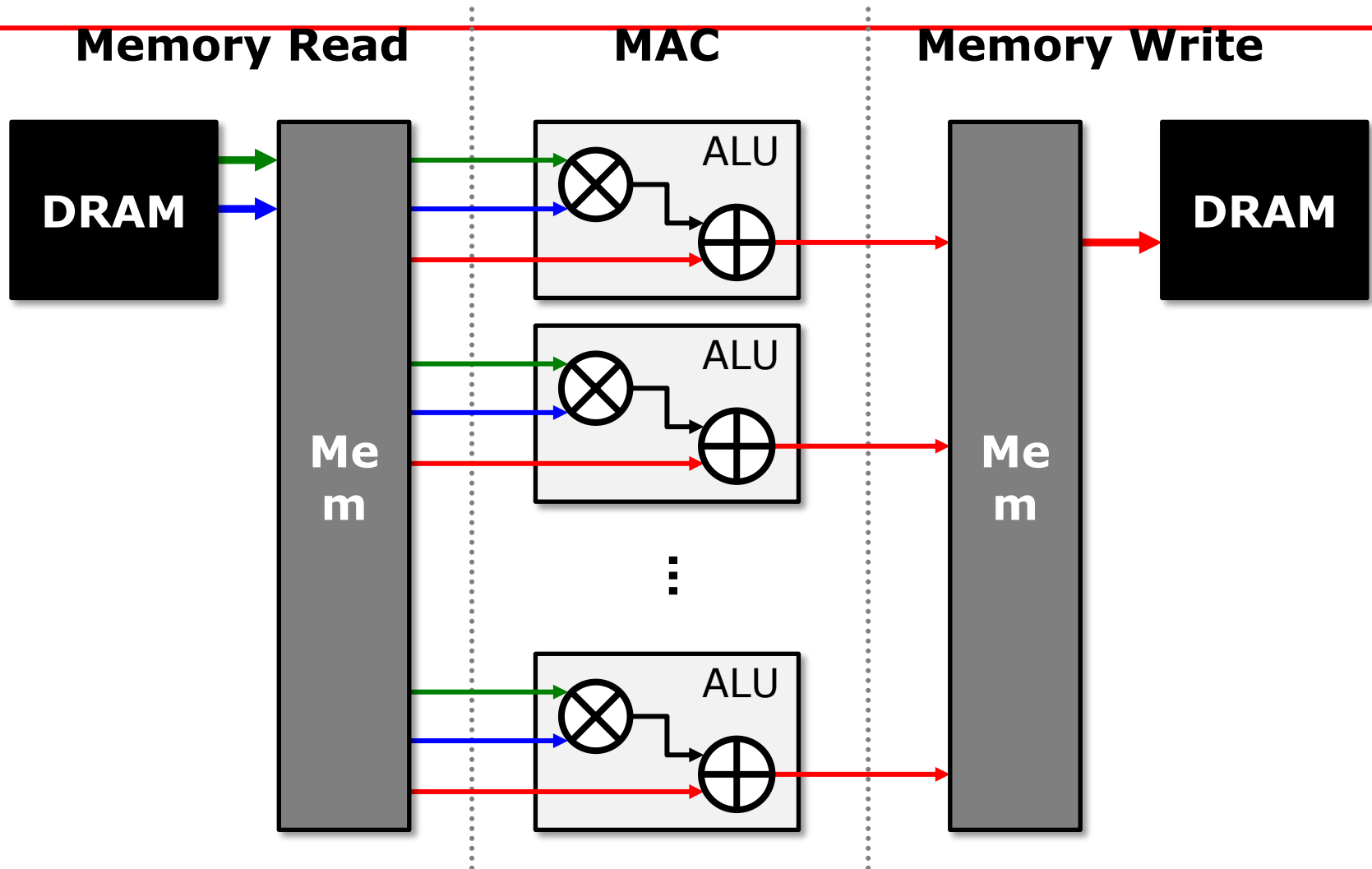
Building a DNN Accelerator



Opportunities: **1 data reuse** **2 local accumulation**

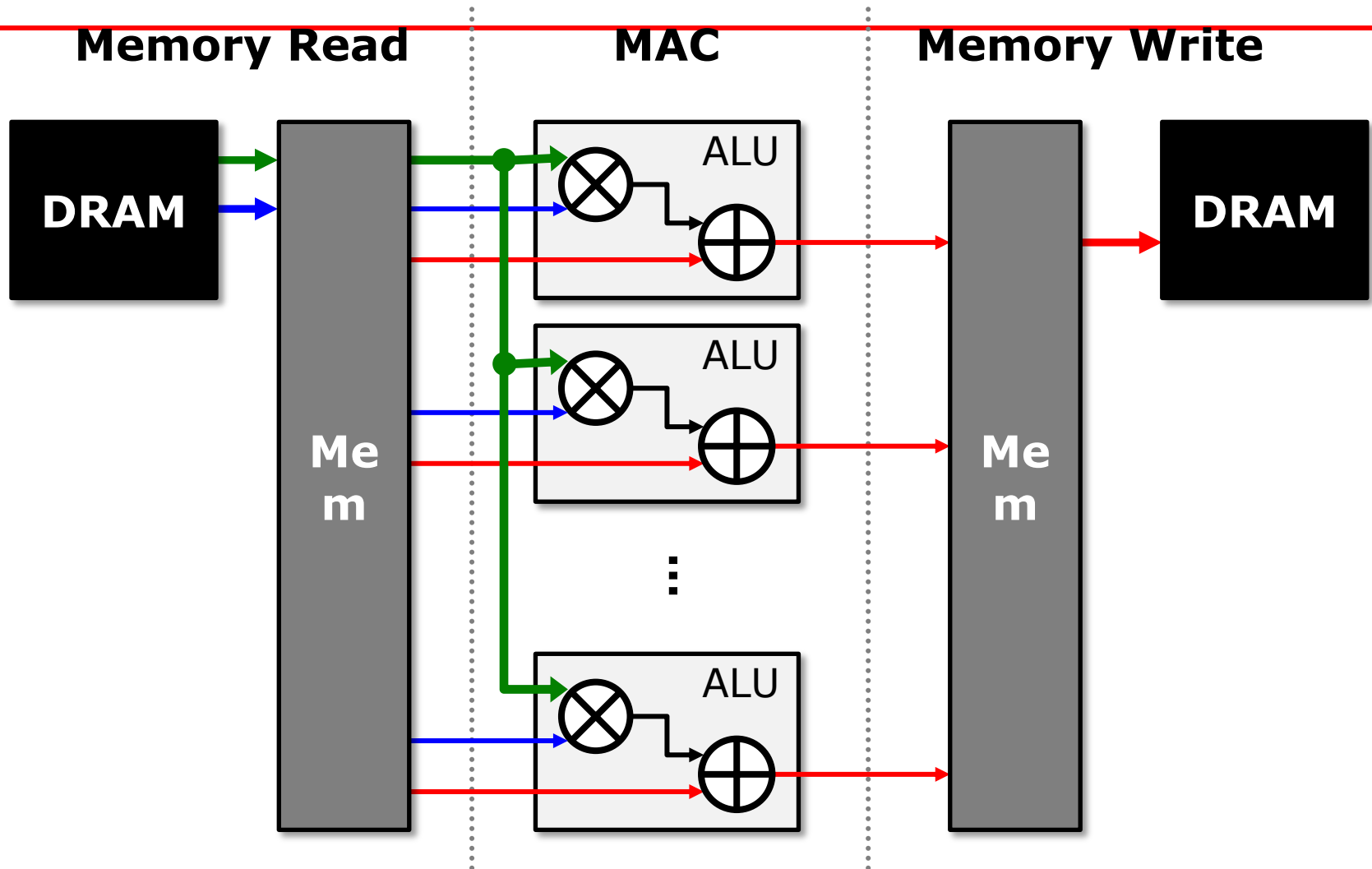
- 1** Can reduce DRAM reads of **filter/fmap** by up to **500x**
- 2** **Partial sum** accumulation does **NOT** have to access DRAM
 - Example: DRAM access in AlexNet can be reduced from **2896M** to **61M** (best case)

Building a DNN Accelerator



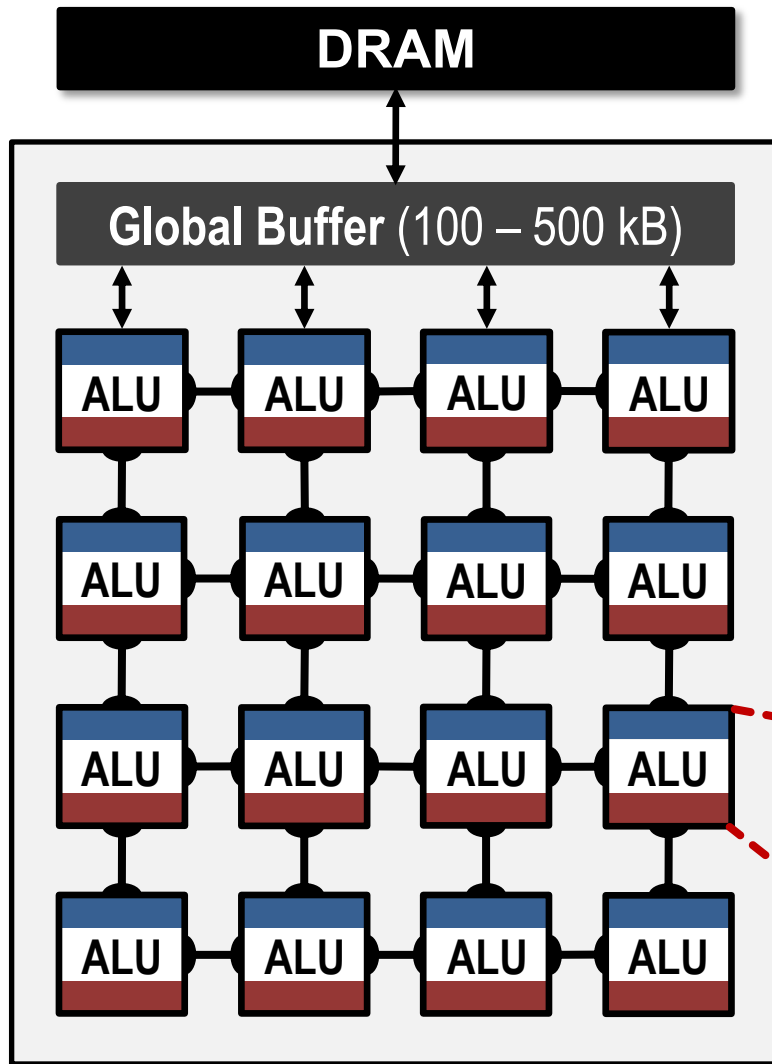
Leverage Parallelism for Throughput!

Building a DNN Accelerator



Leverage Parallelism for *spatial* data reuse!

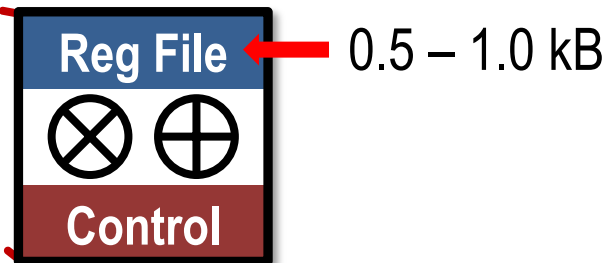
Spatial DNN Accelerator



Local Memory Hierarchy

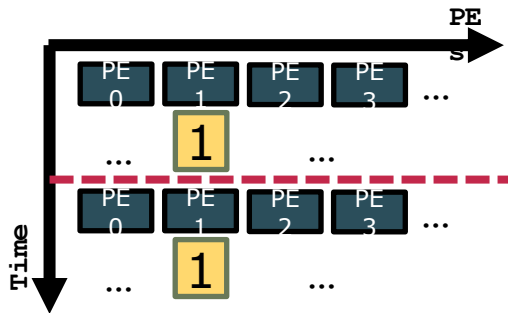
- Global Buffer
- Direct inter-PE network
- PE-local memory (RF)

Processing Element (PE)

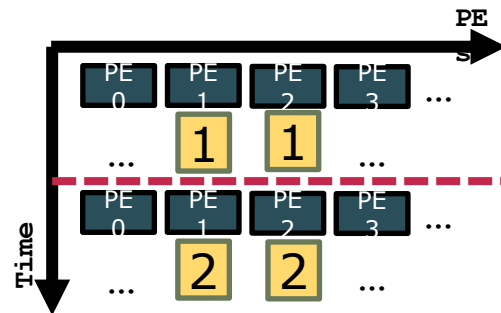


Hardware structures to exploit reuse

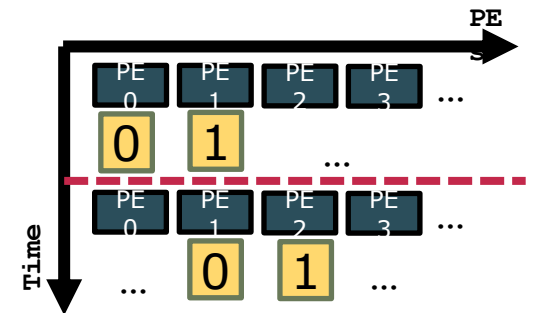
Temporal Reuse



Spatial Reuse

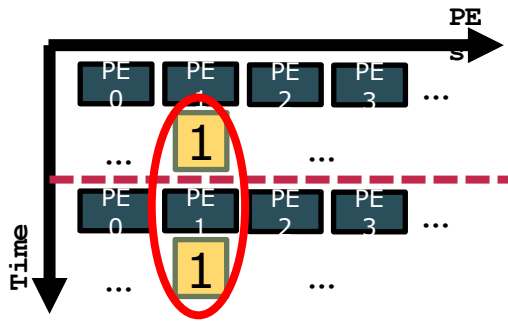


Spatio-Temporal Reuse

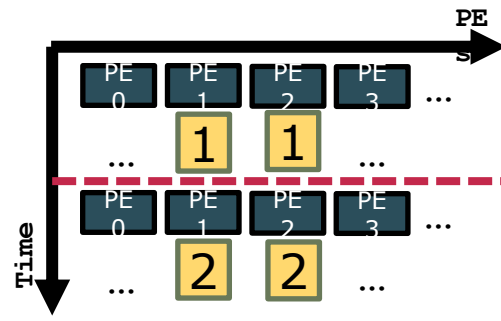


Hardware structures to exploit reuse

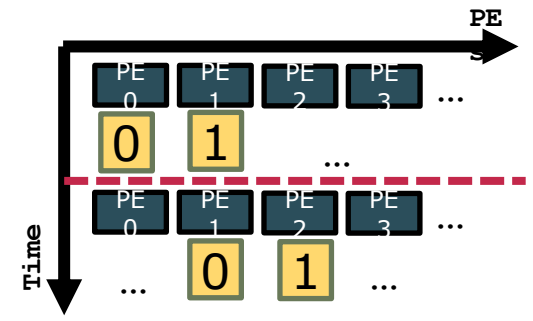
Temporal Reuse



Spatial Reuse

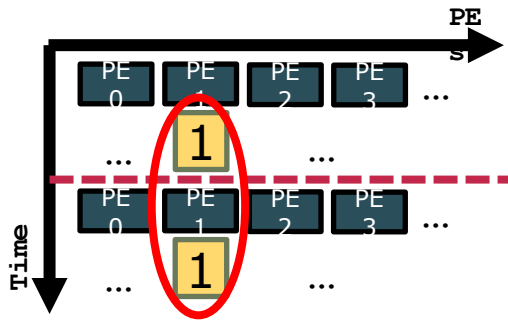


Spatio-Temporal Reuse

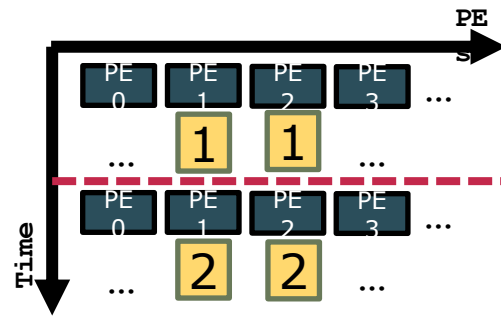


Hardware structures to exploit reuse

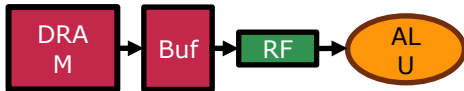
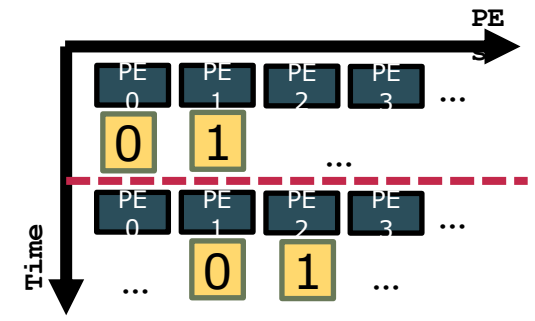
Temporal Reuse



Spatial Reuse



Spatio-Temporal Reuse

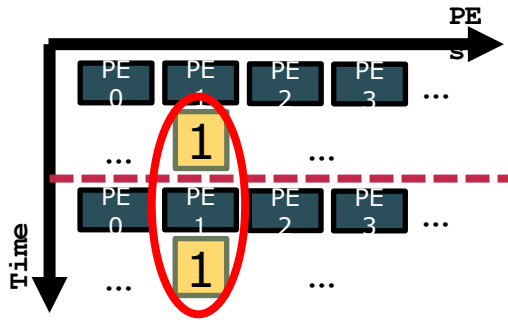


Memory Hierarchy /
Staging Buffers

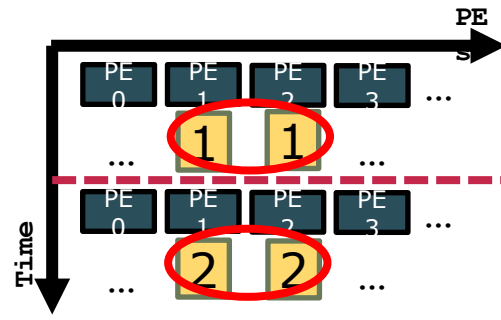
E.g., Custom memory hierarchies in accelerators.

Hardware structures to exploit reuse

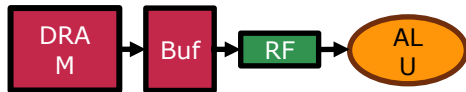
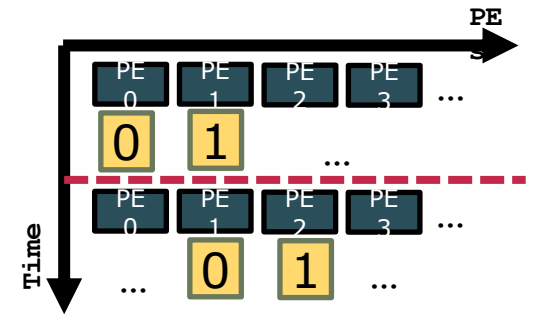
Temporal Reuse



Spatial Reuse



Spatio-Temporal Reuse

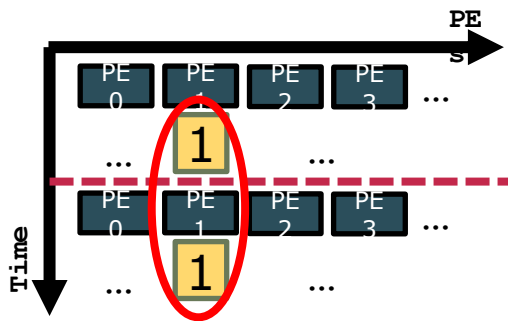


Memory Hierarchy /
Staging Buffers

E.g., Custom memory hierarchies in
accelerators.

Hardware structures to exploit reuse

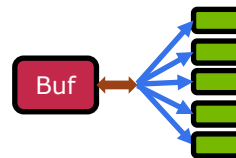
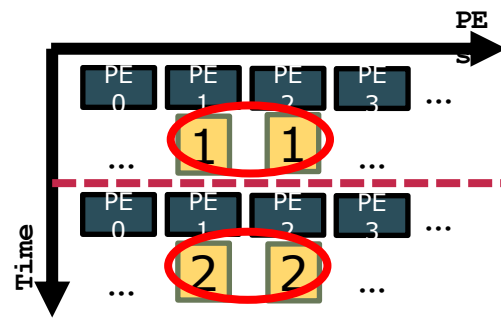
Temporal Reuse



Memory Hierarchy / Staging Buffers

E.g., Custom memory hierarchies in accelerators.

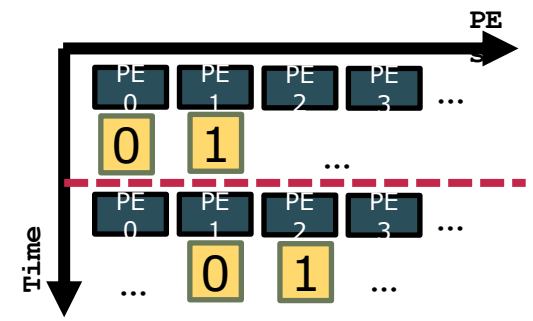
Spatial Reuse



Multicasting-support NoCs

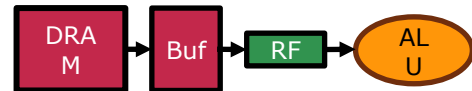
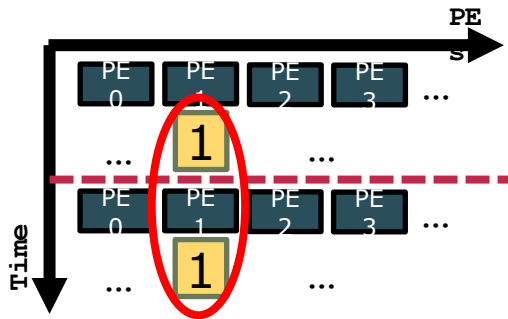
E.g., Hierarchical Bus in Eyeriss (ISCA 2016), Tree in MAERI (ASPLOS 2018)

Spatio-Temporal Reuse



Hardware structures to exploit reuse

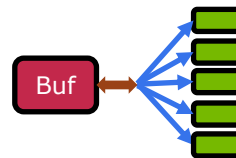
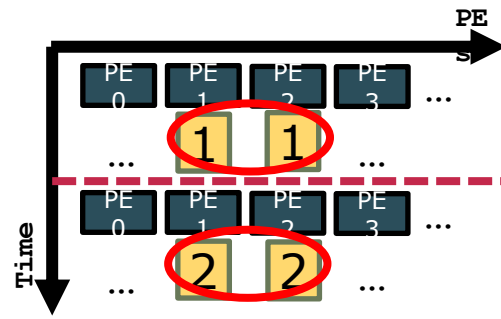
Temporal Reuse



Memory Hierarchy /
Staging Buffers

E.g., Custom memory hierarchies in accelerators.

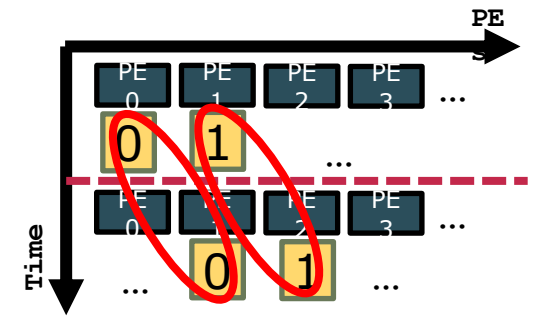
Spatial Reuse



Multicasting-support NoCs

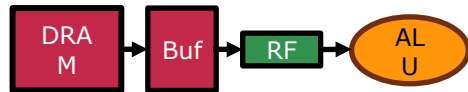
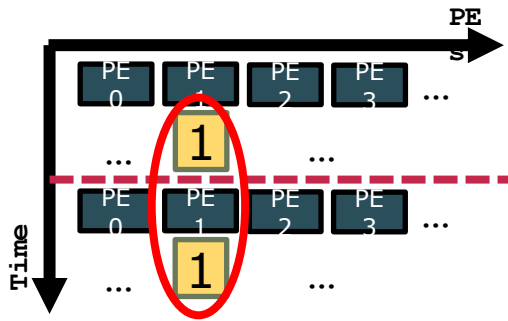
E.g., Hierarchical Bus in Eyeriss (ISCA 2016),
Tree in MAERI (ASPLOS 2018)

Spatio-Temporal Reuse



Hardware structures to exploit reuse

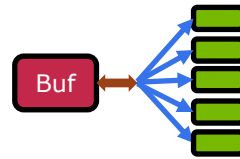
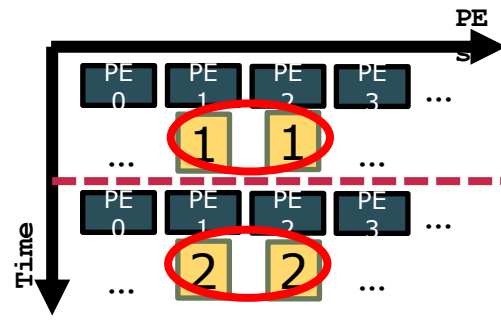
Temporal Reuse



Memory Hierarchy / Staging Buffers

E.g., Custom memory hierarchies in accelerators.

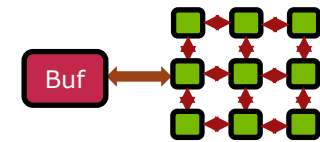
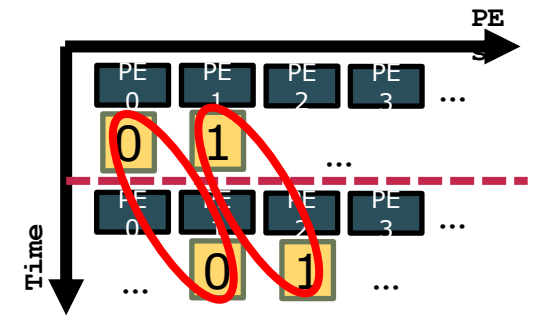
Spatial Reuse



Multicasting-support NoCs

E.g., Hierarchical Bus in Eyeriss (ISCA 2016), Tree in MAERI (ASPLOS 2018)

Spatio-Temporal Reuse

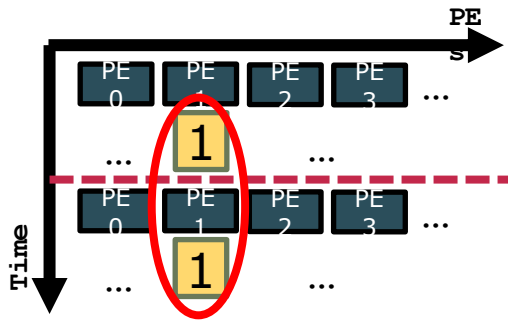


Neighbor-to-Neighbor Connections

E.g., TPU (ISCA 2017), local network in Eyeriss (ISCA 2016)

Hardware structures to exploit reuse

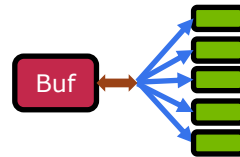
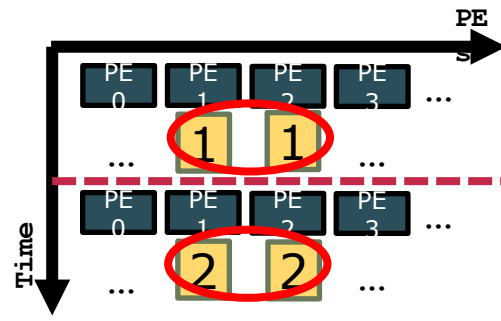
Temporal Reuse



Memory Hierarchy / Staging Buffers

E.g., Custom memory hierarchies in accelerators.

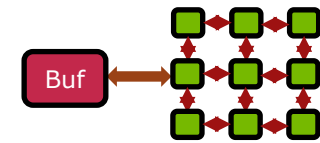
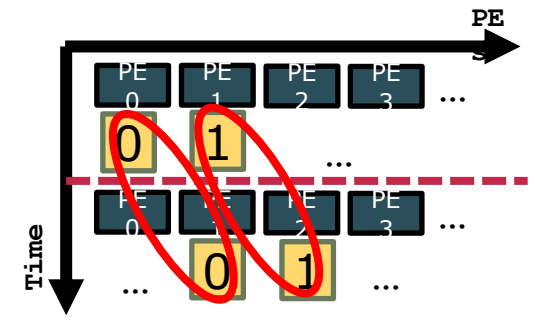
Spatial Reuse



Multicasting-support NoCs

E.g., Hierarchical Bus in Eyeriss (ISCA 2016), Tree in MAERI (ASPLOS 2018)

Spatio-Temporal Reuse



Neighbor-to-Neighbor Connections

E.g., TPU (ISCA 2017), local network in Eyeriss (ISCA 2016)

The availability of the hardware structure limits the "mapping-space"

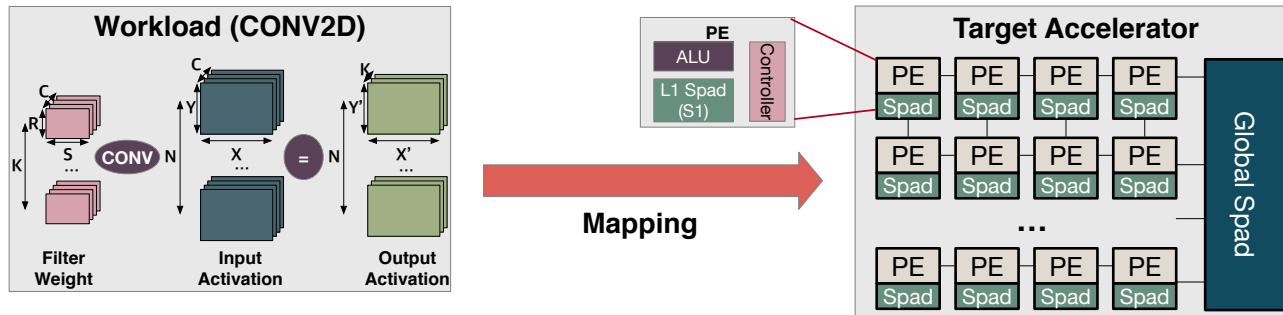
Design-space of a DNN Accelerator

HW Design-Space

Mapping Design-Space aka Map Space

HW or SW depending on flexibility

Design-space of a DNN Accelerator

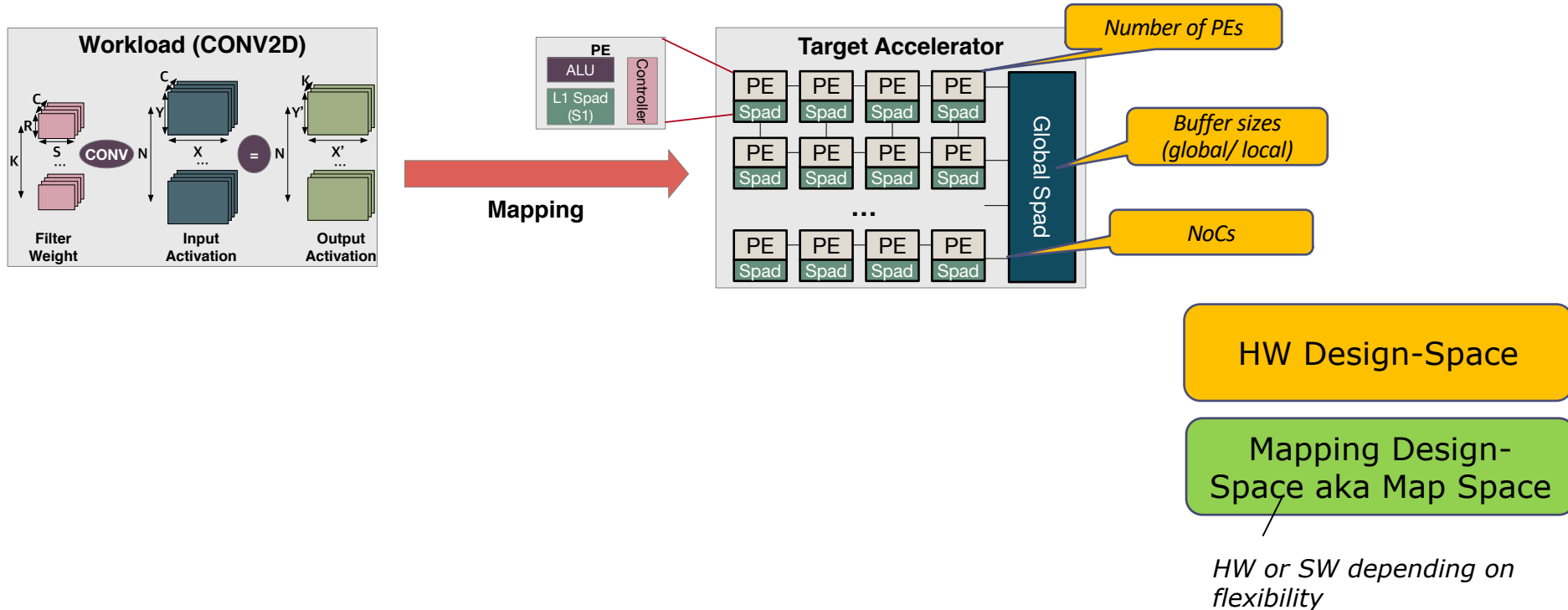


HW Design-Space

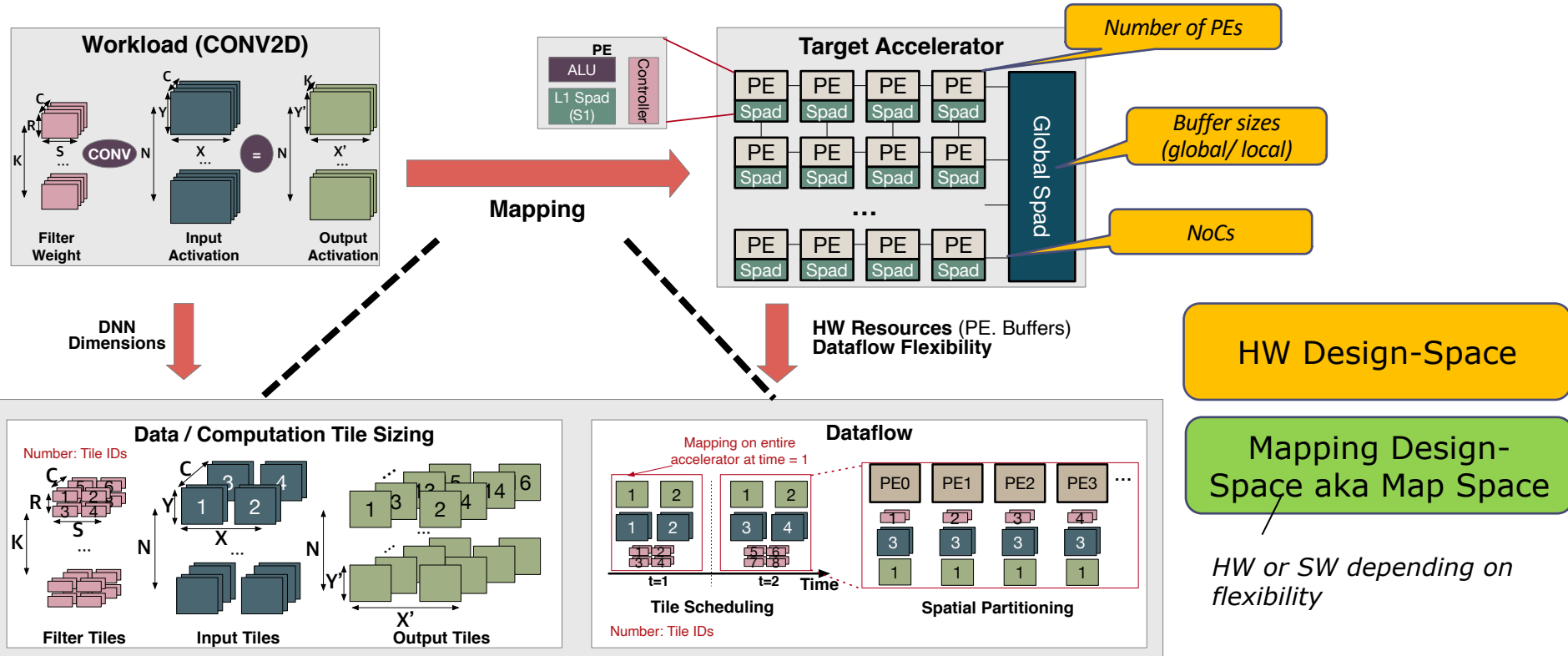
Mapping Design-Space aka Map Space

HW or SW depending on flexibility

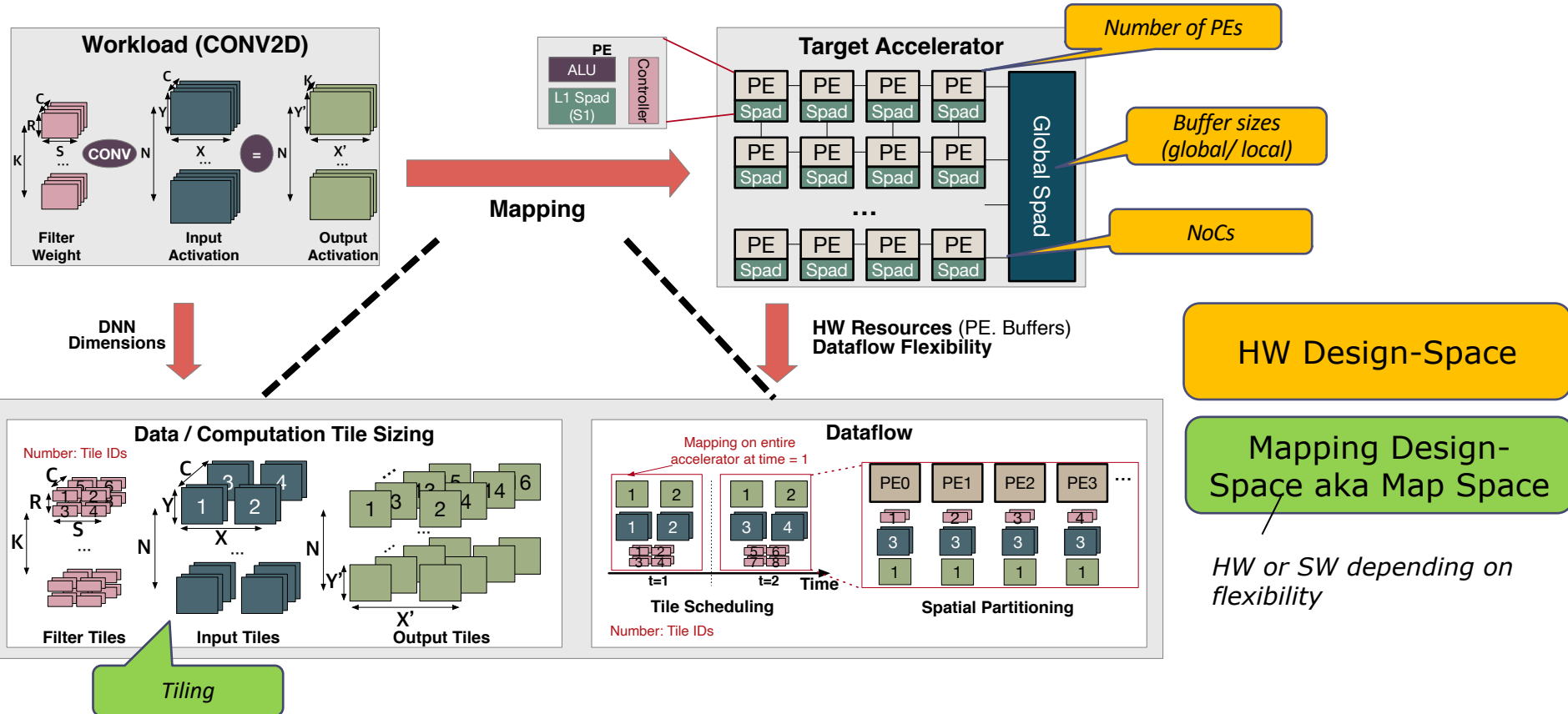
Design-space of a DNN Accelerator



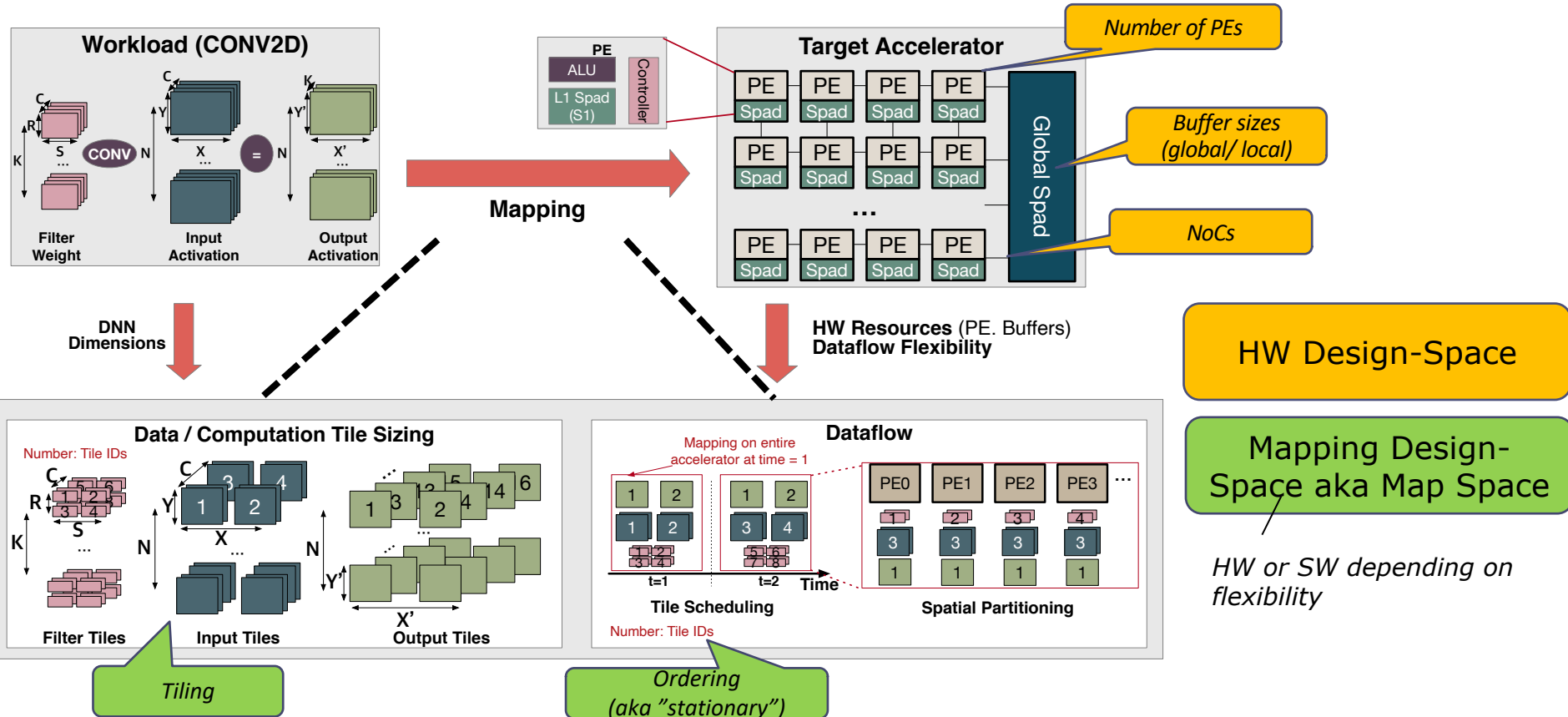
Design-space of a DNN Accelerator



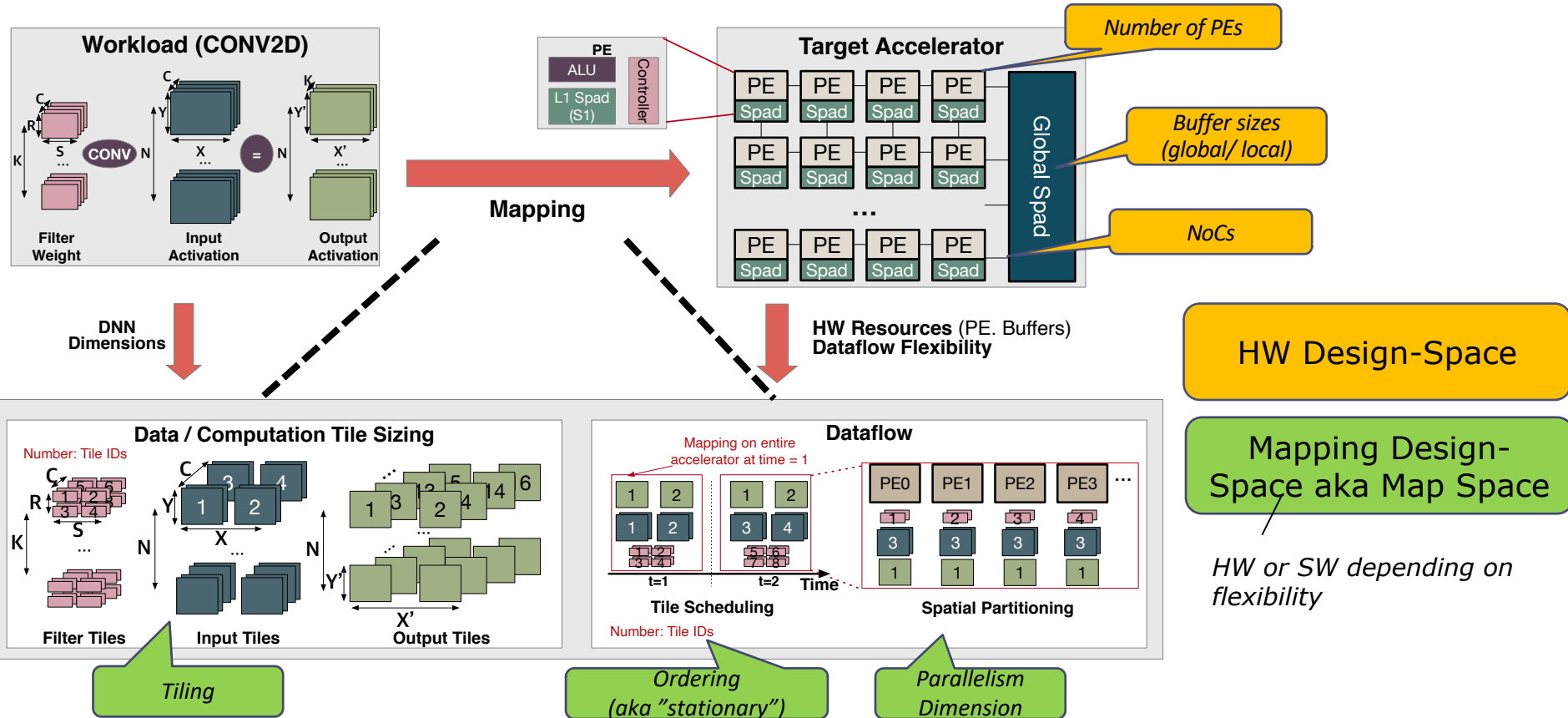
Design-space of a DNN Accelerator



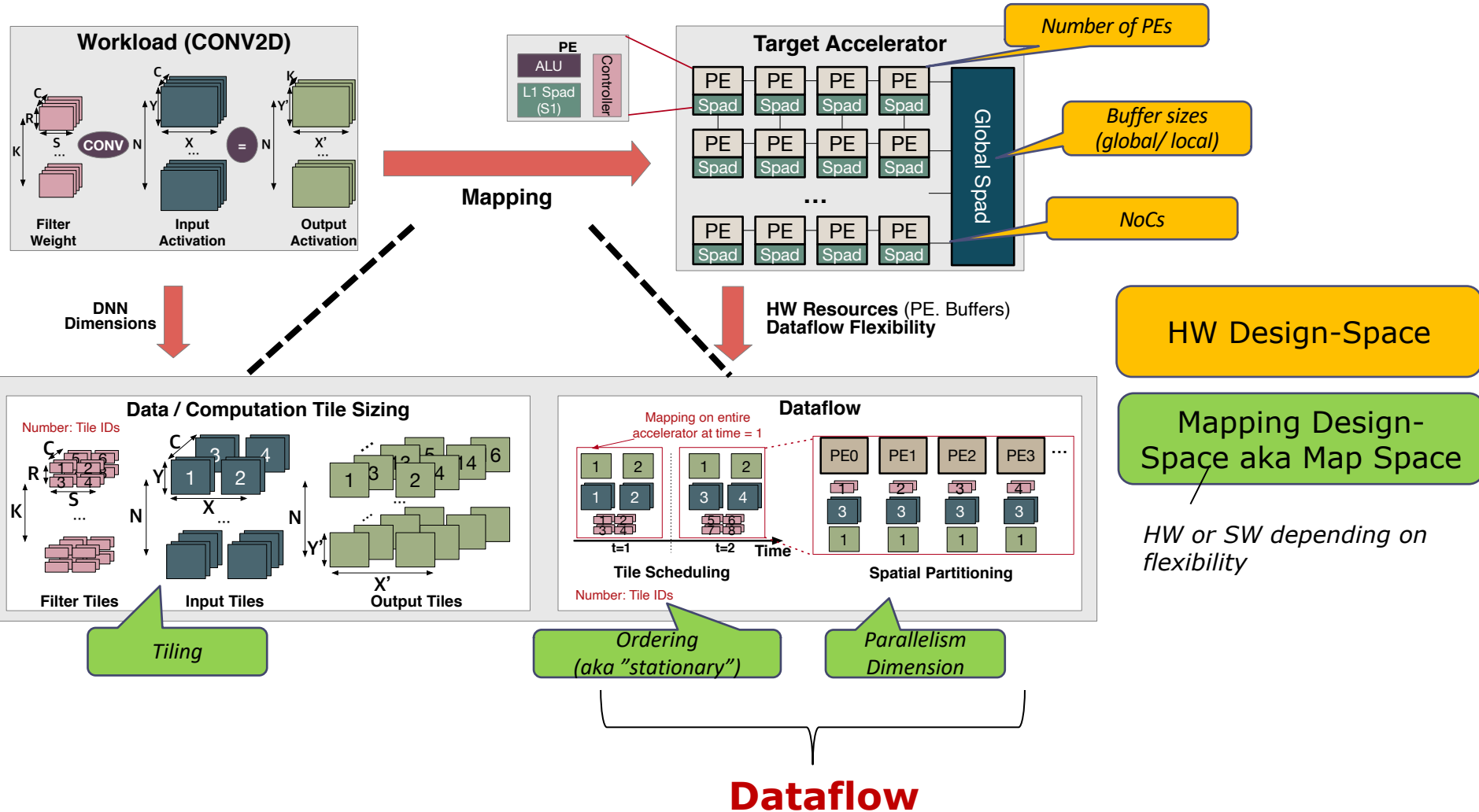
Design-space of a DNN Accelerator



Design-space of a DNN Accelerator



Design-space of a DNN Accelerator

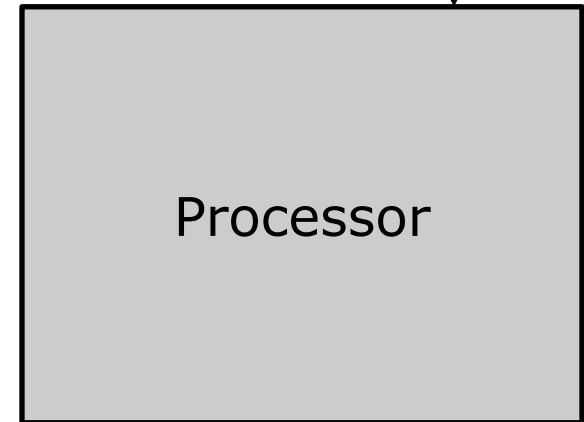


CPU Compute Model

Program



Input
Data

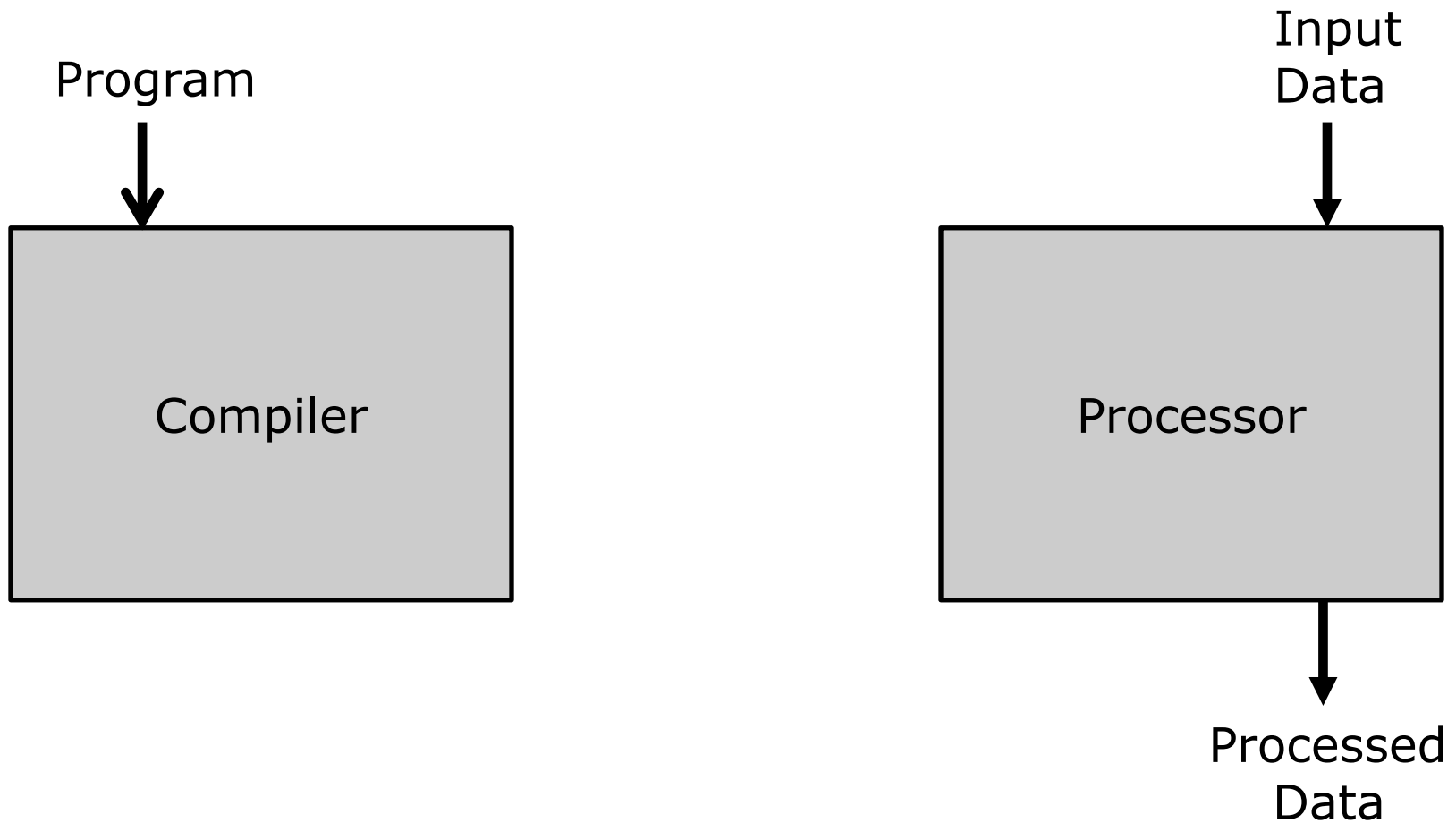


Processor

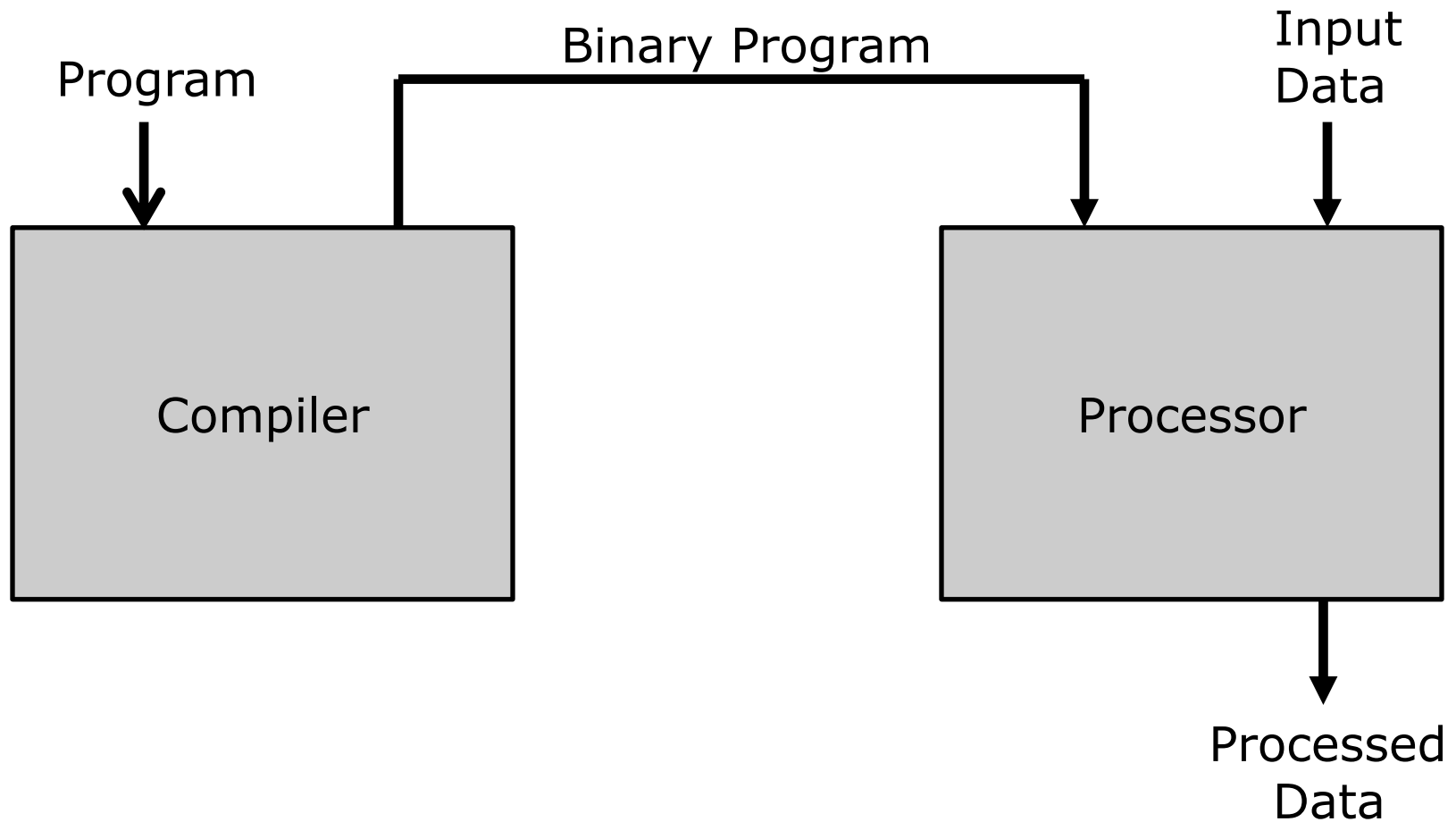


Processed
Data

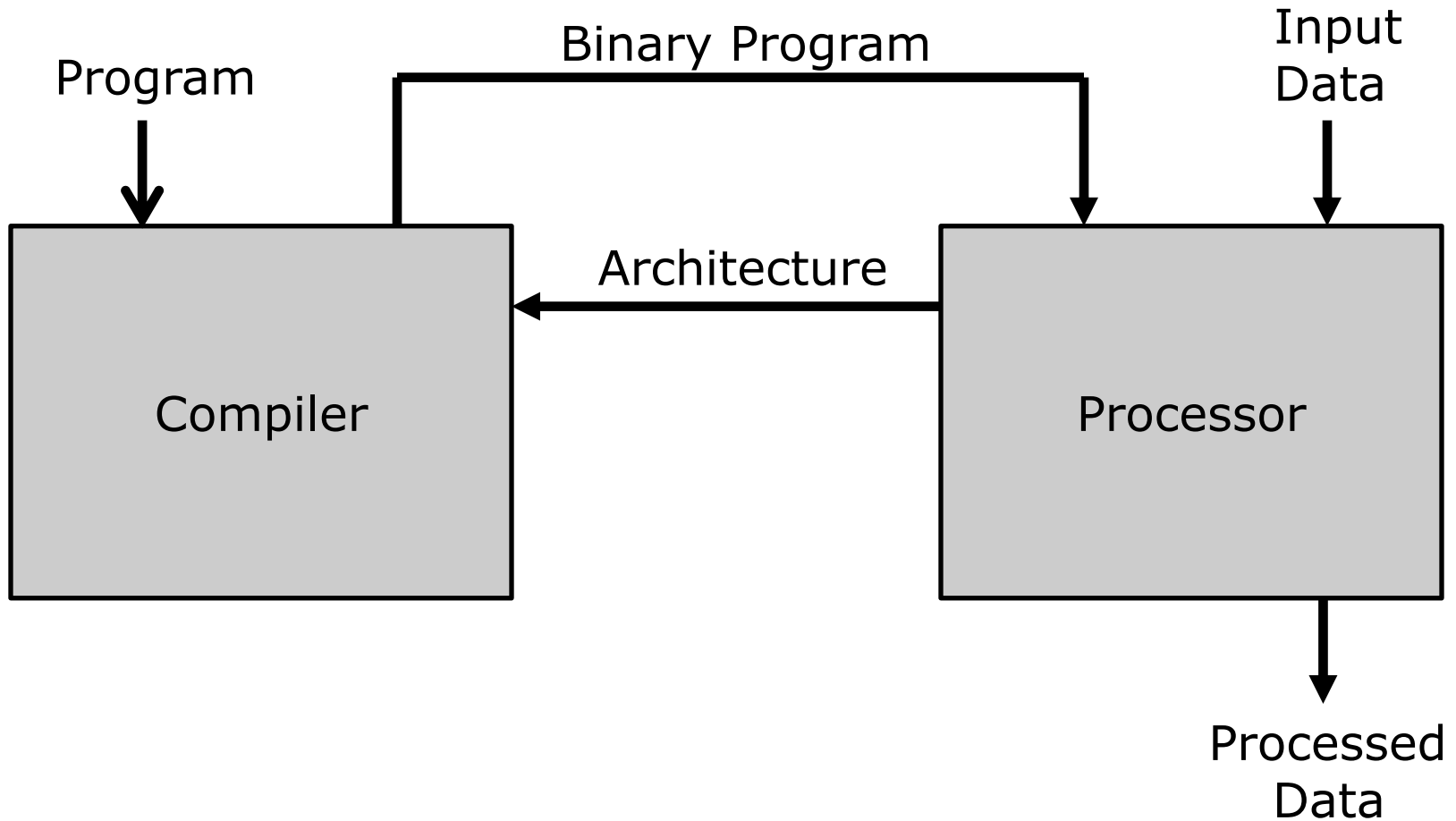
CPU Compute Model



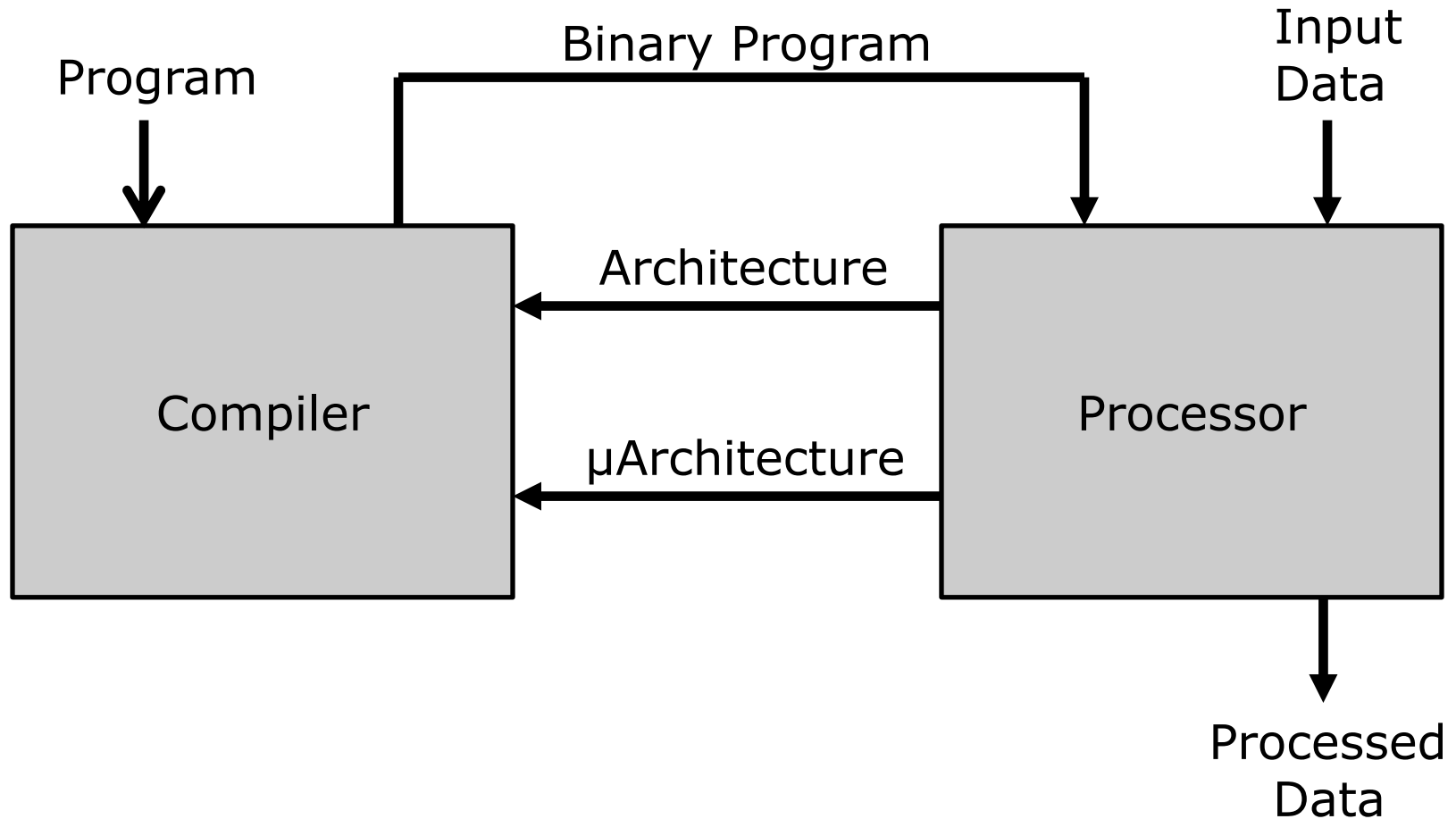
CPU Compute Model



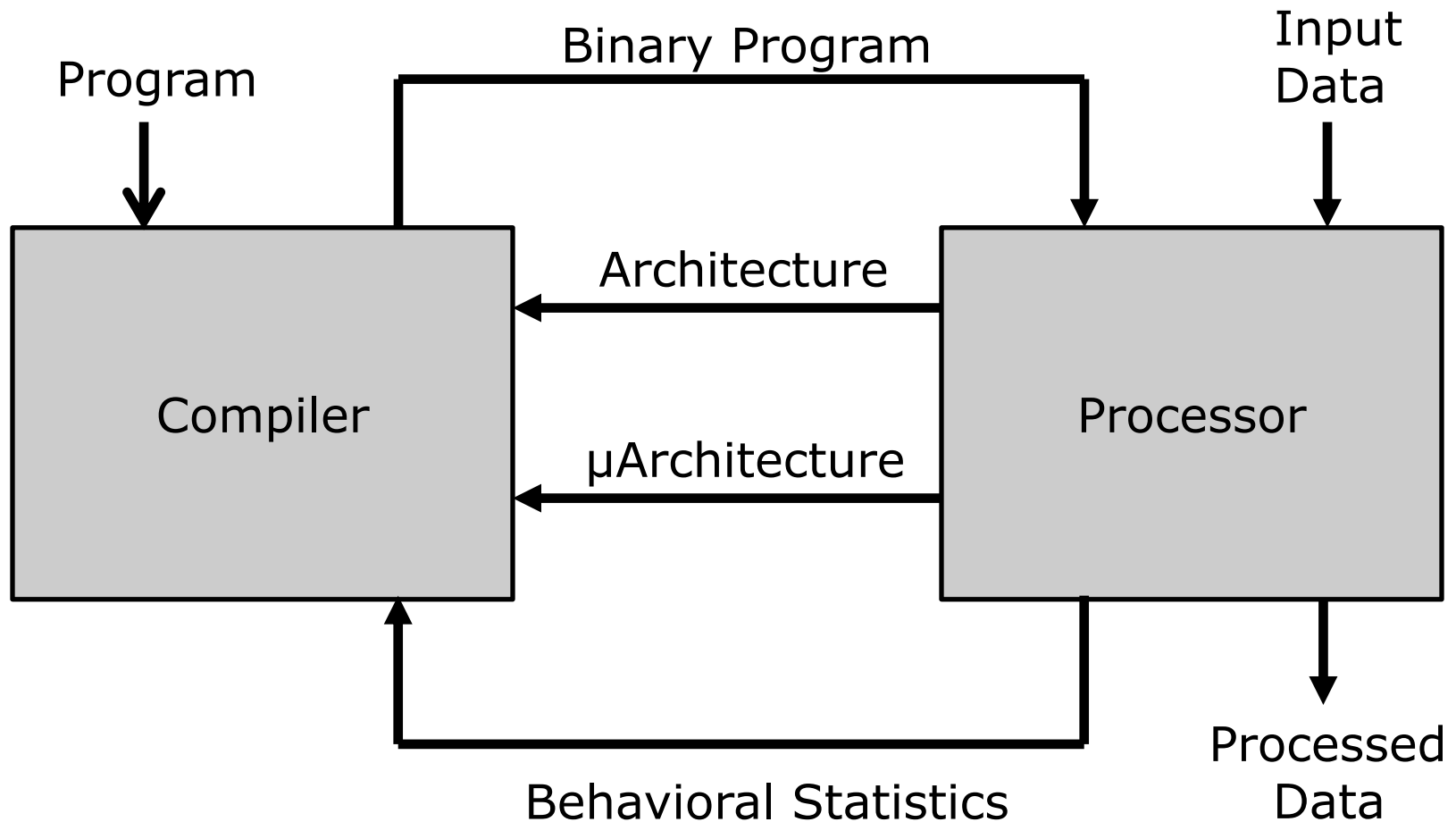
CPU Compute Model



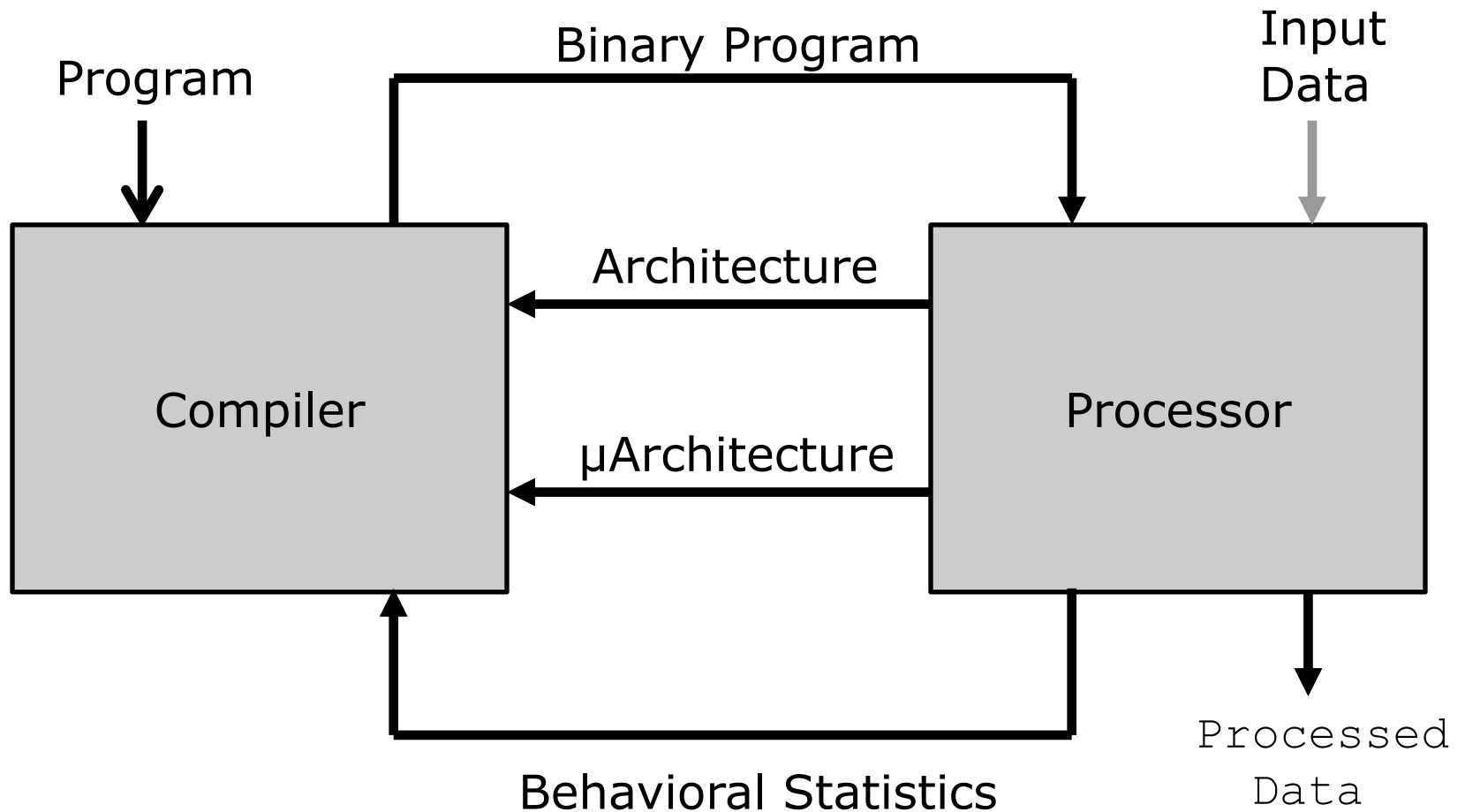
CPU Compute Model



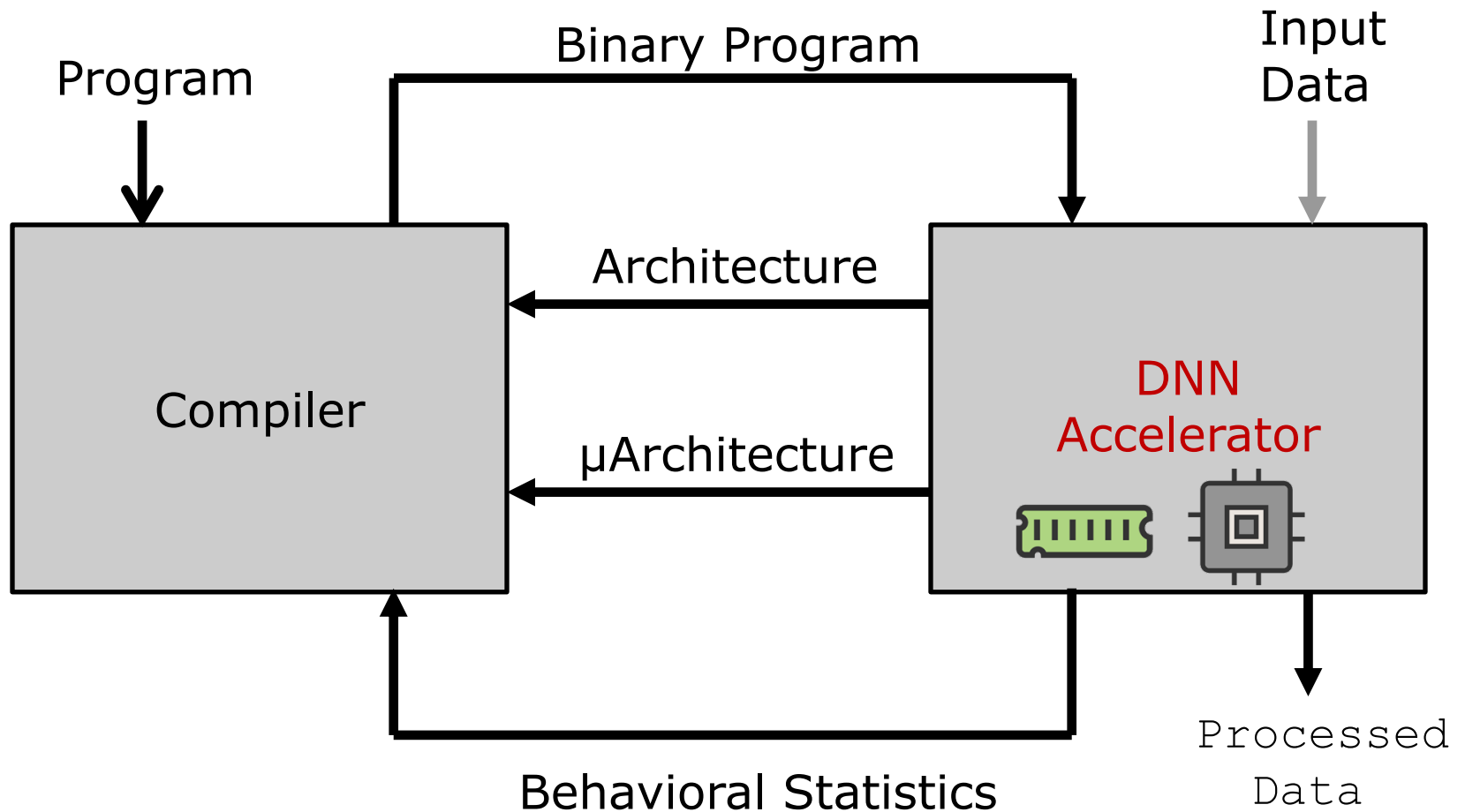
CPU Compute Model



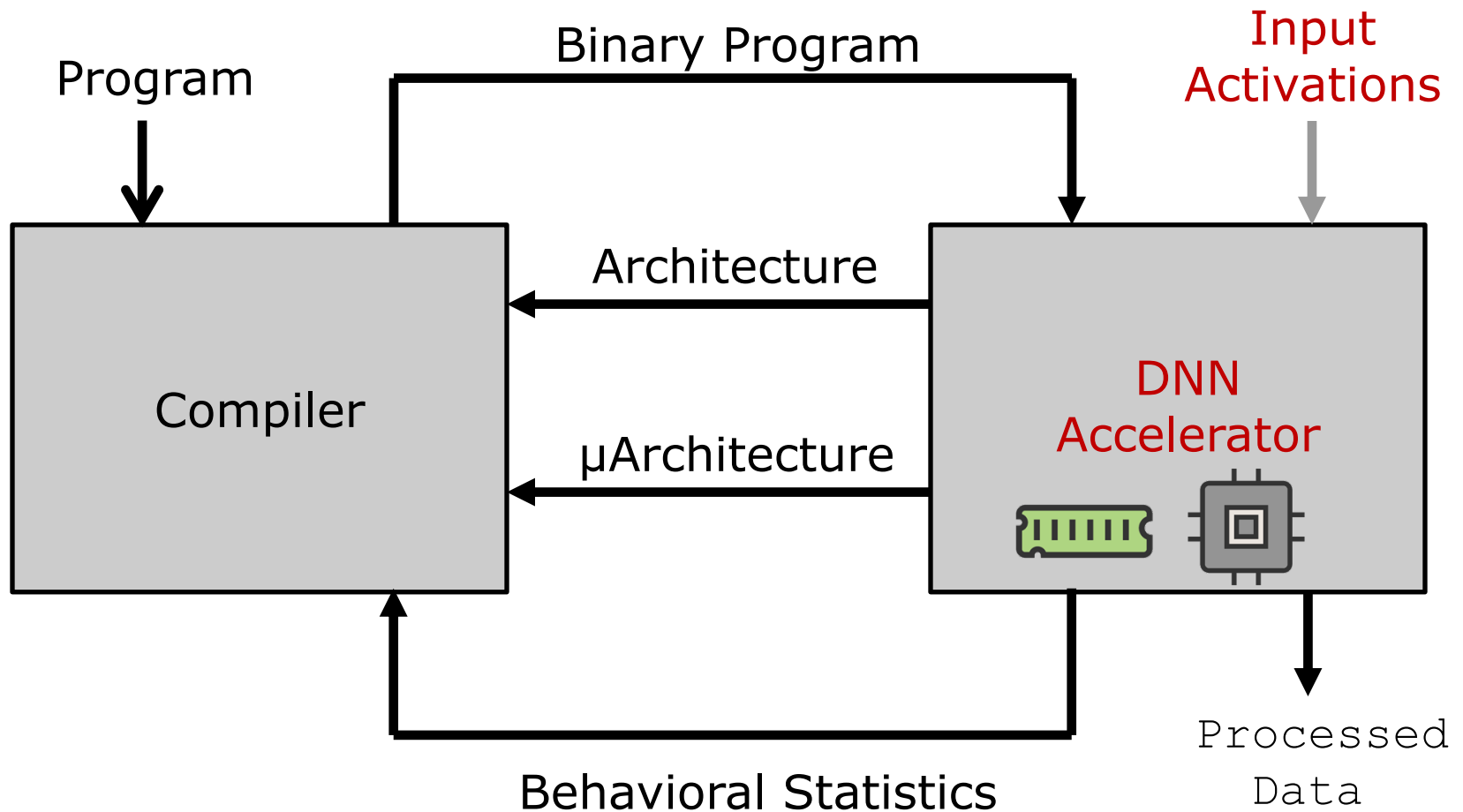
DNN Compute Model



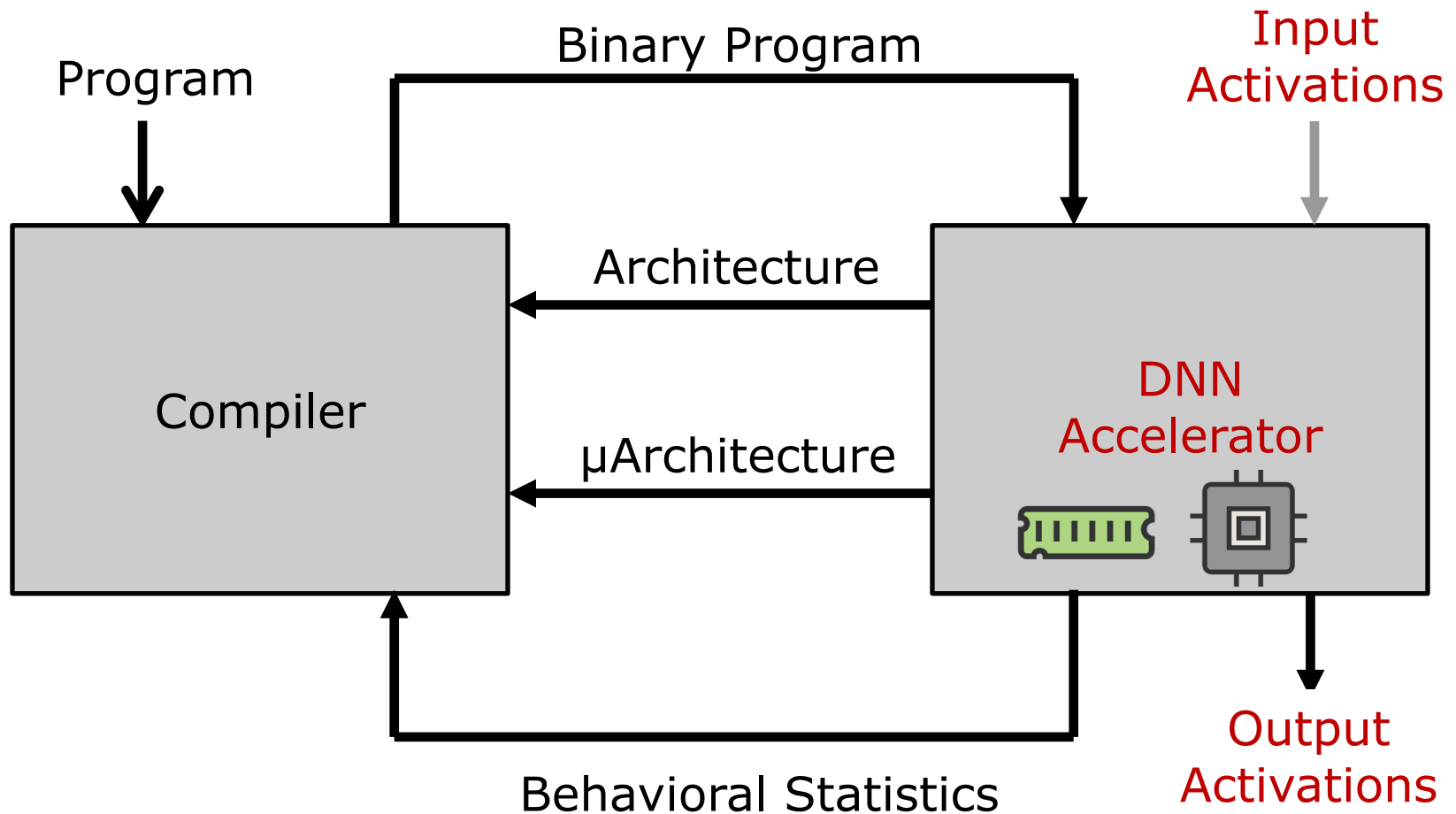
DNN Compute Model



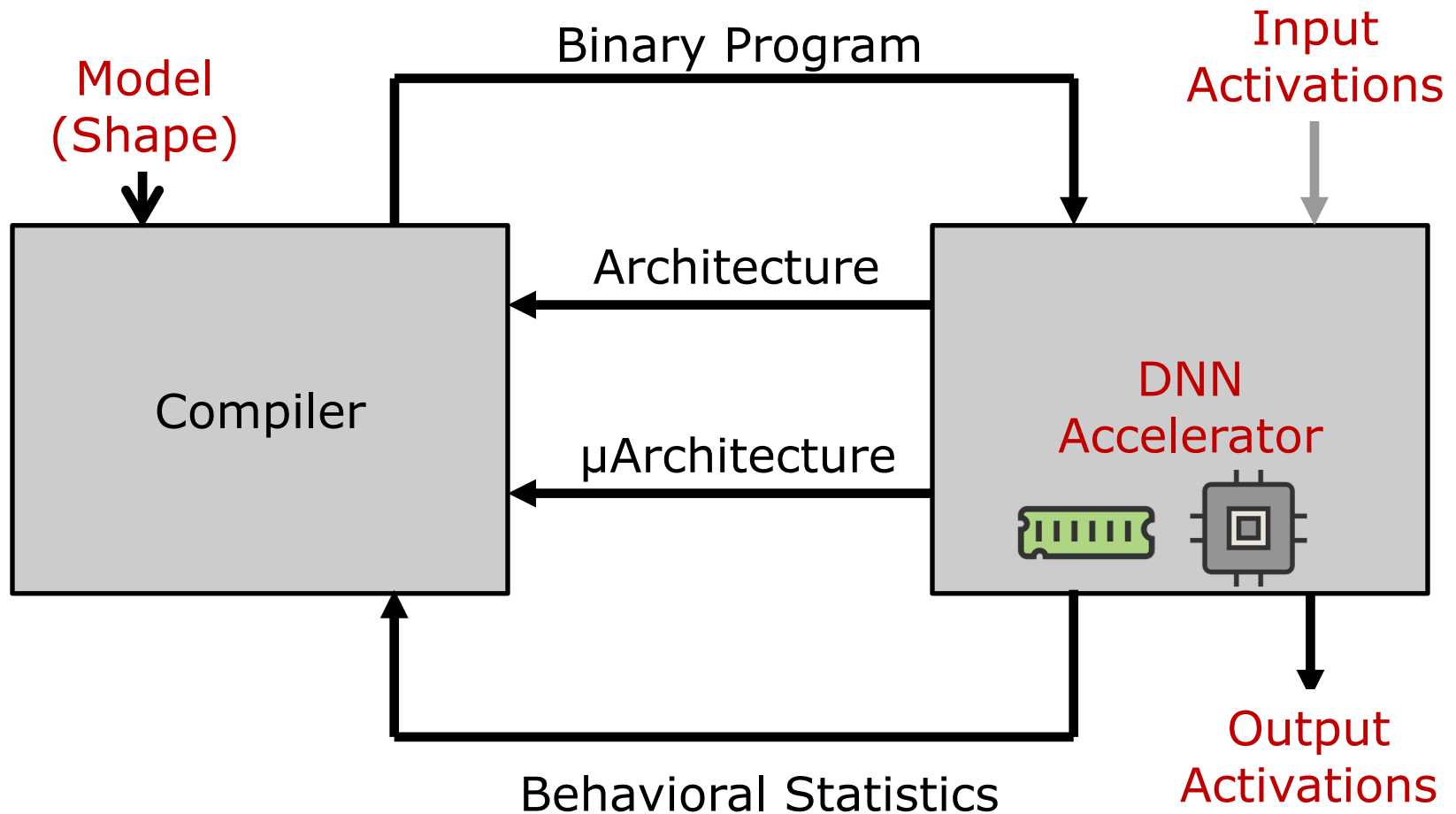
DNN Compute Model



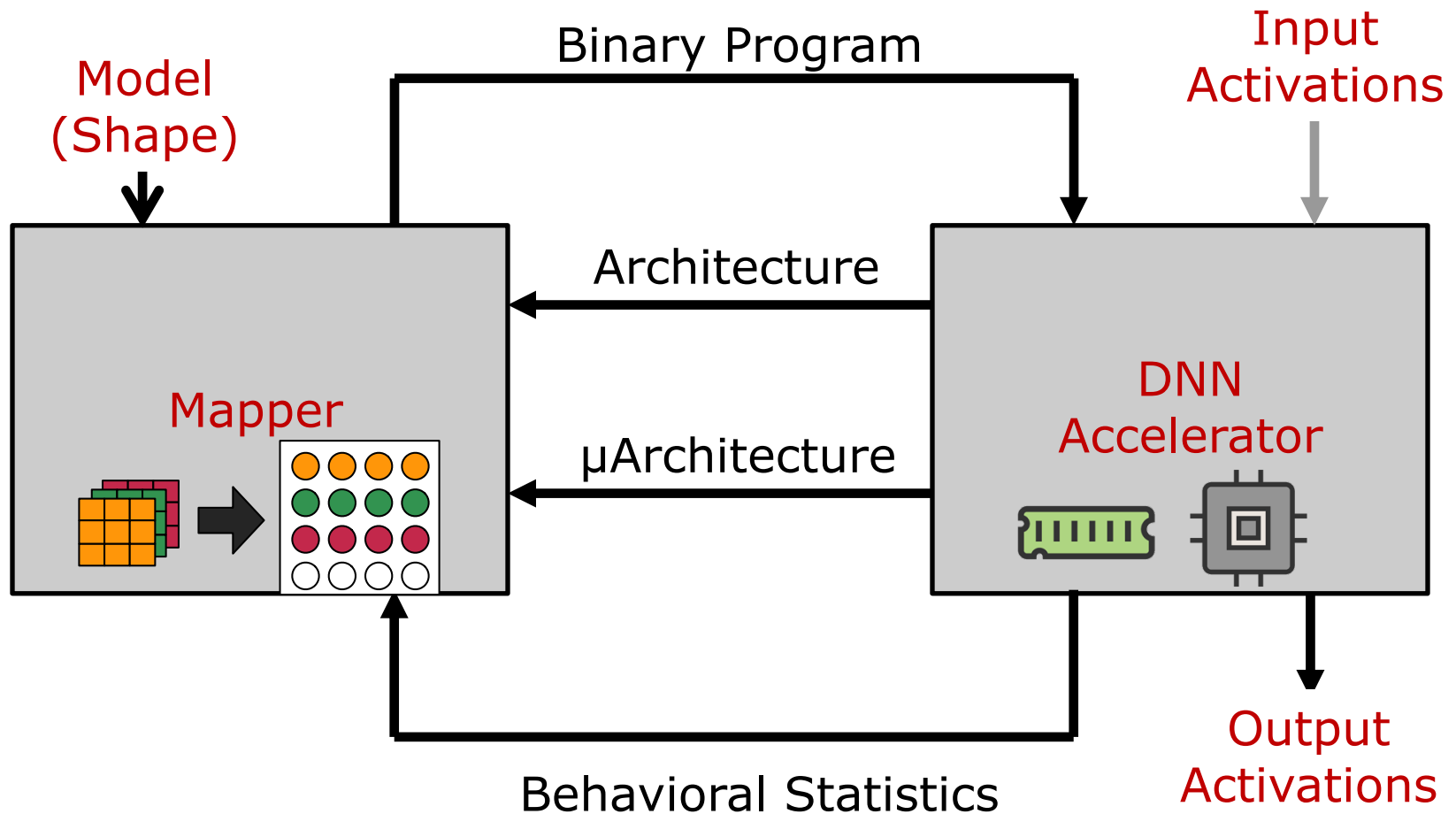
DNN Compute Model



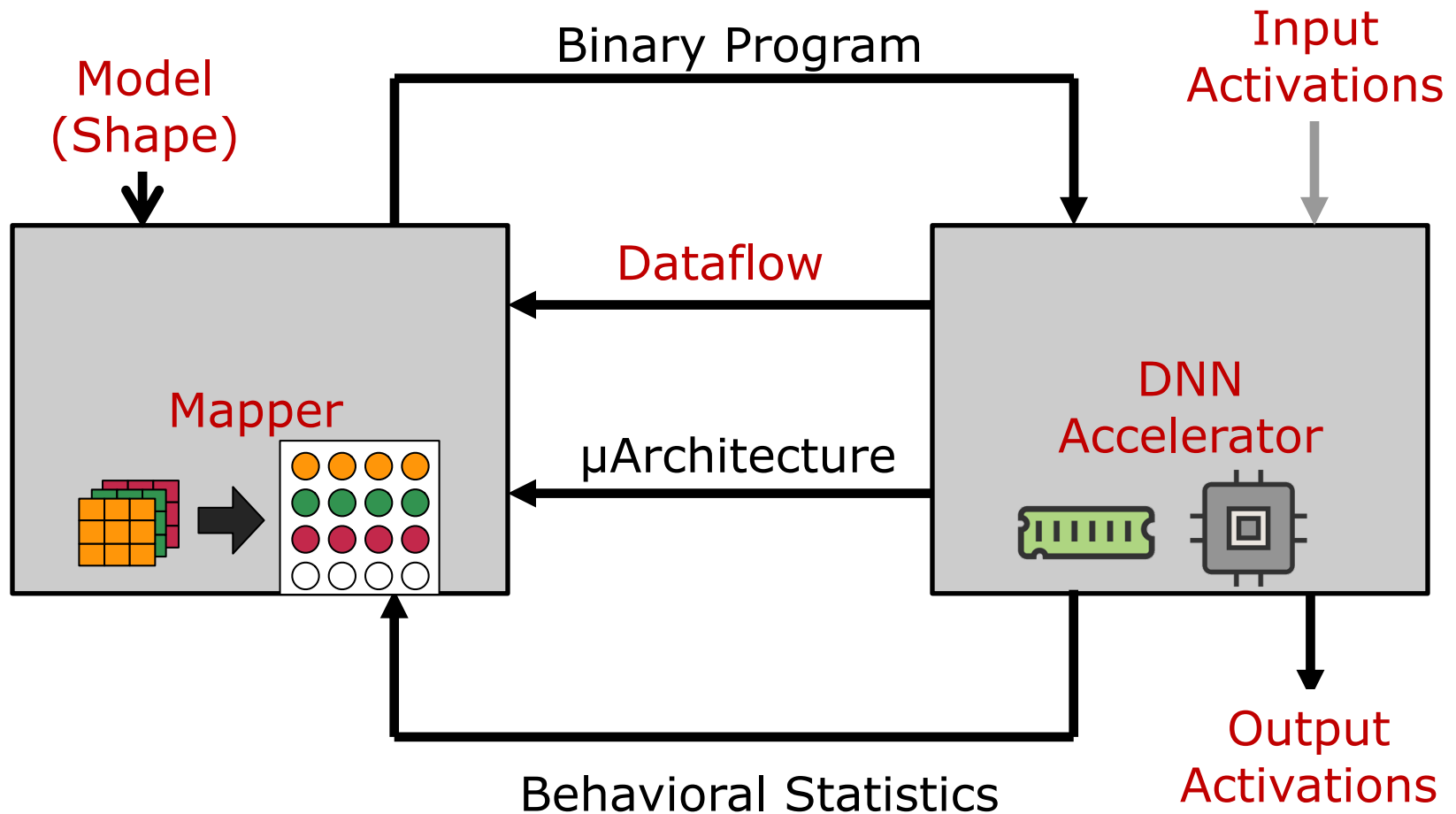
DNN Compute Model



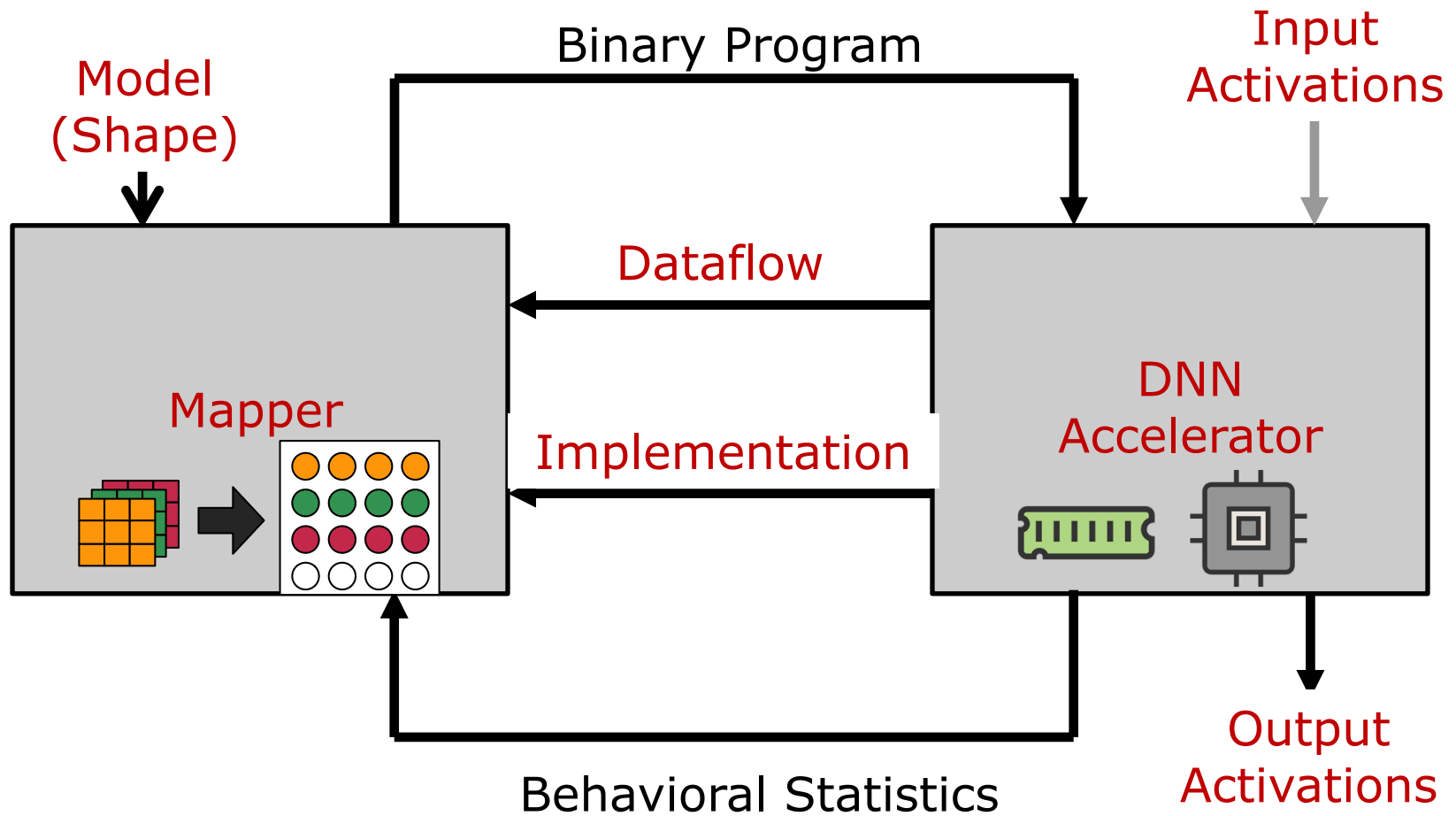
DNN Compute Model



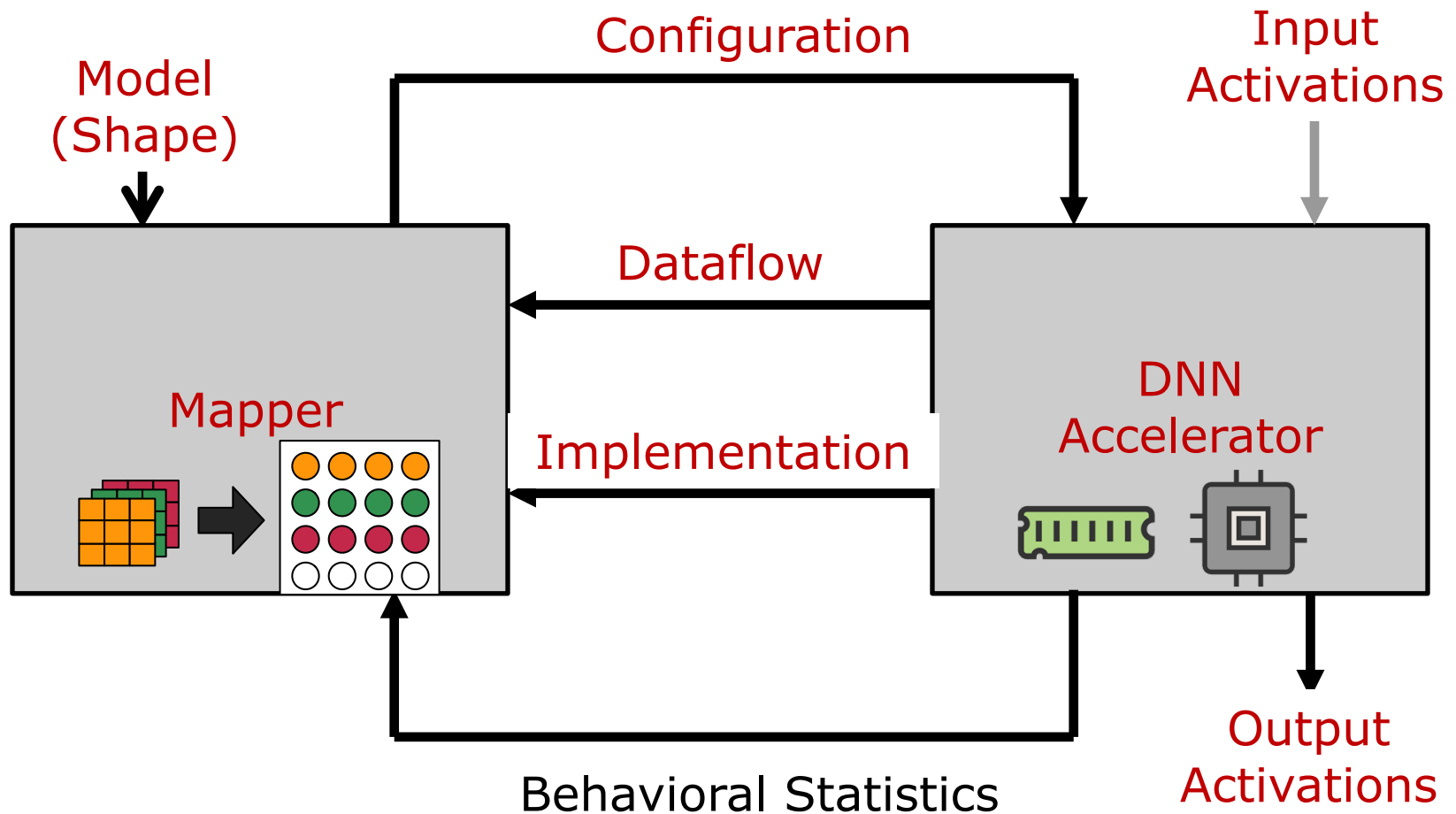
DNN Compute Model



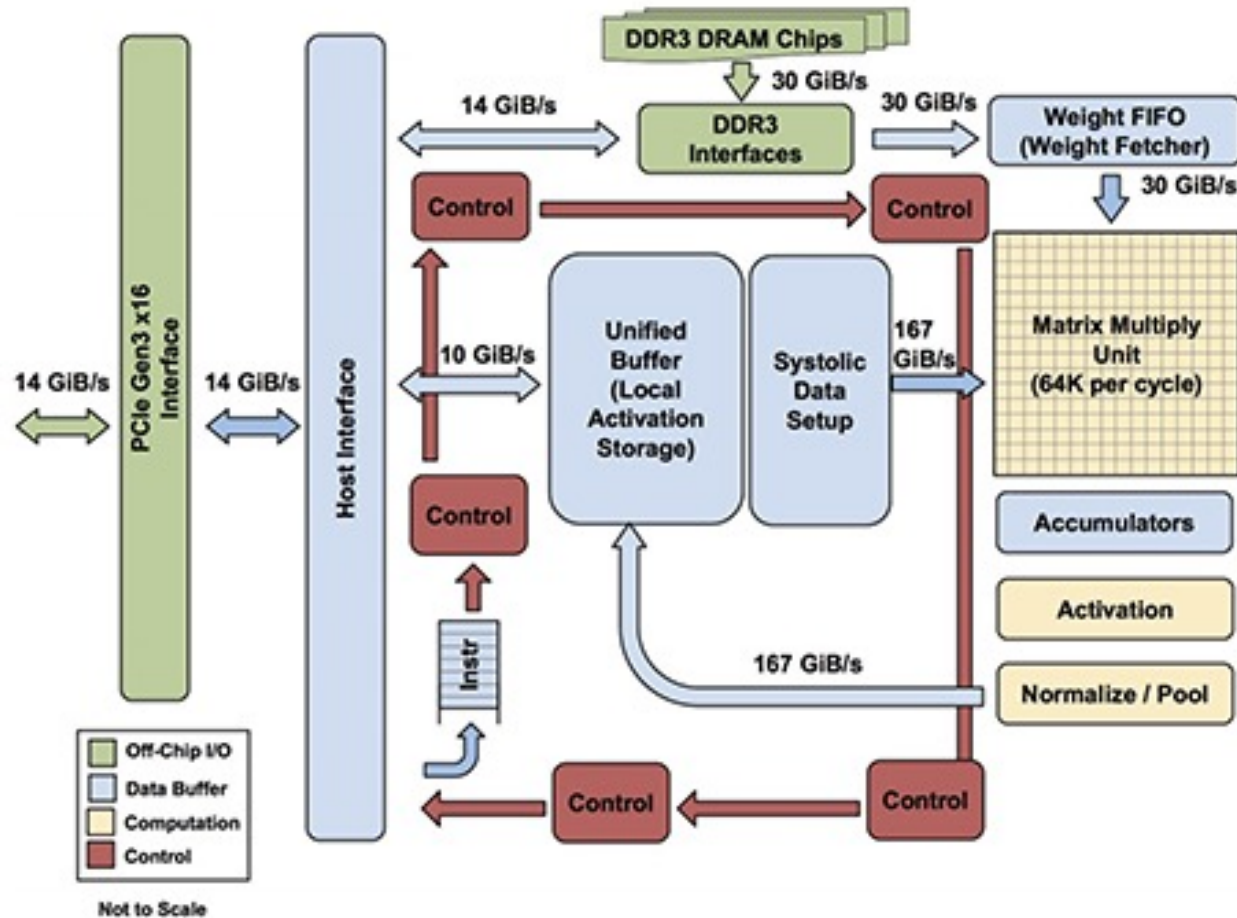
DNN Compute Model



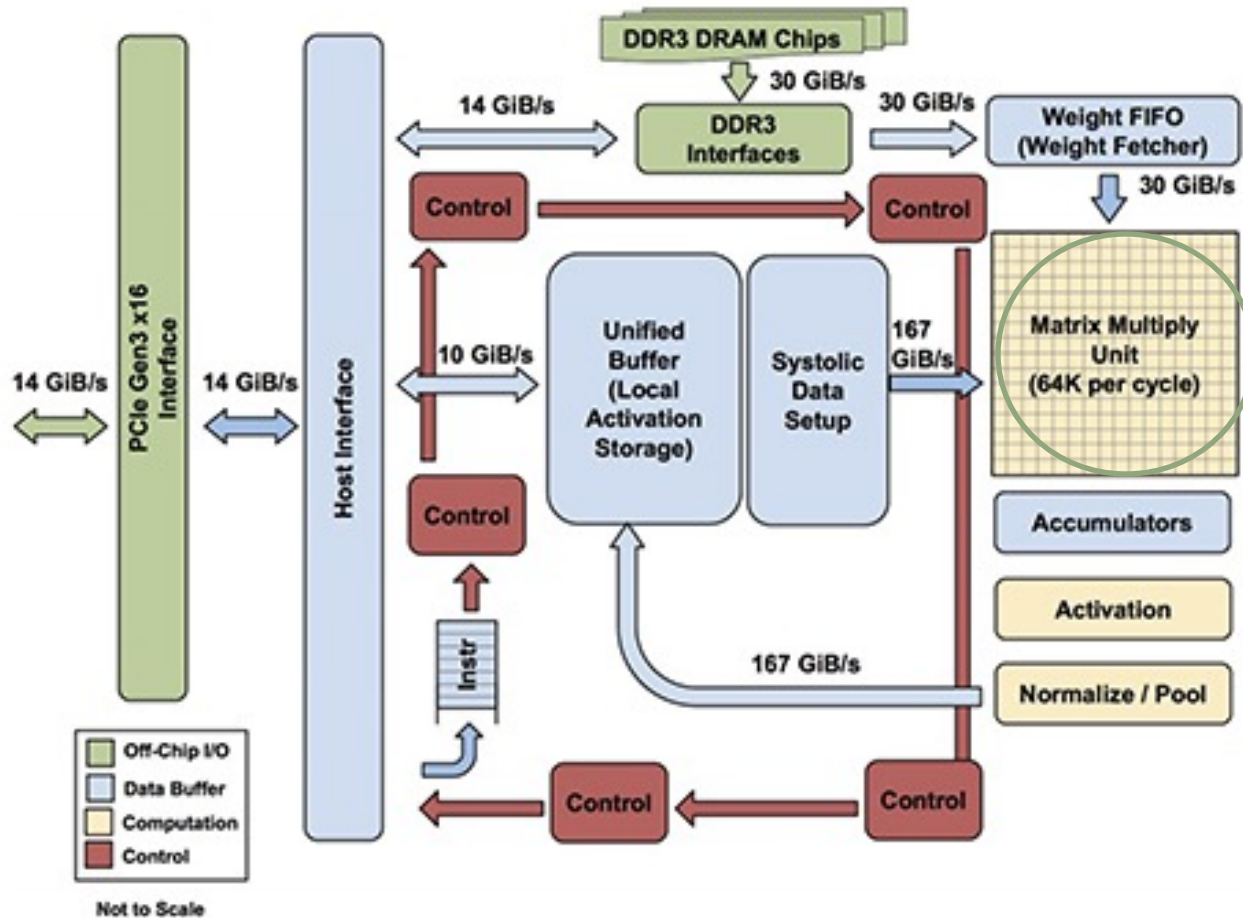
DNN Compute Model



Putting it all together: Case Study of Google TPUv1 (2016)

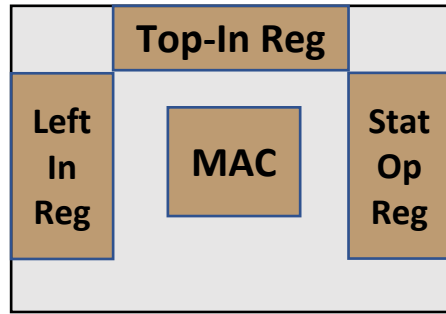


Putting it all together: Case Study of Google TPUv1 (2016)

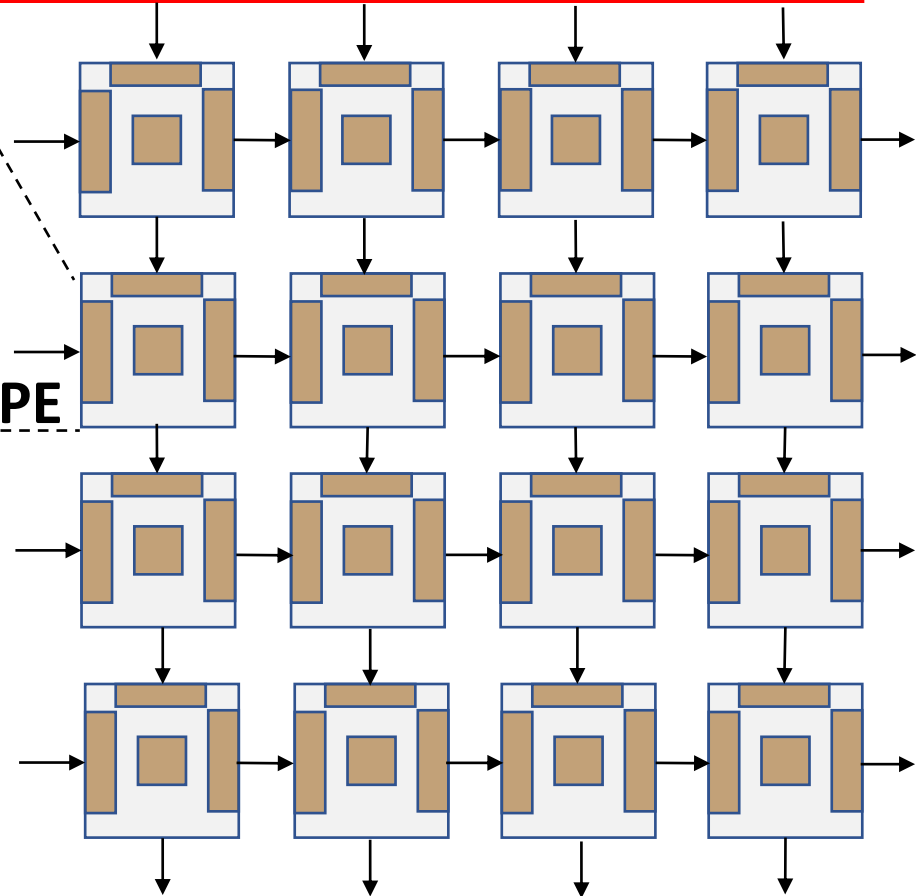


"Systolic Array"

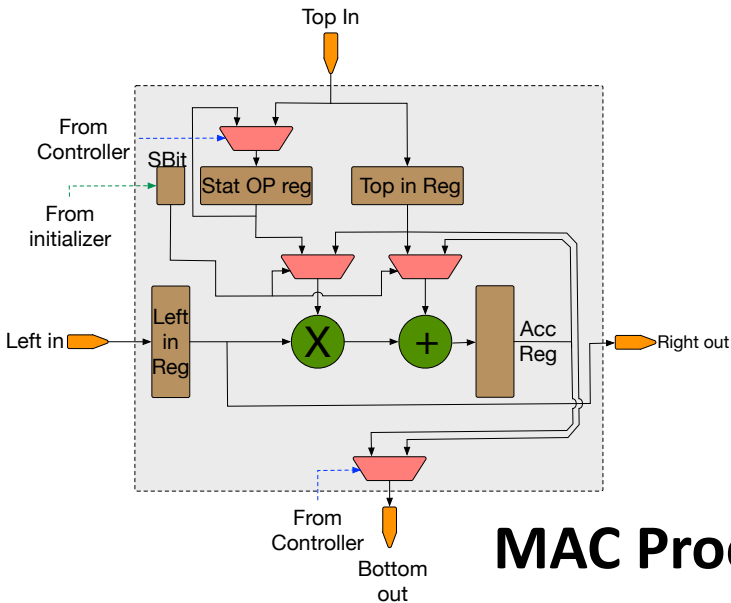
Systolic Array Structure



Schematic of MAC PE



Systolic Array

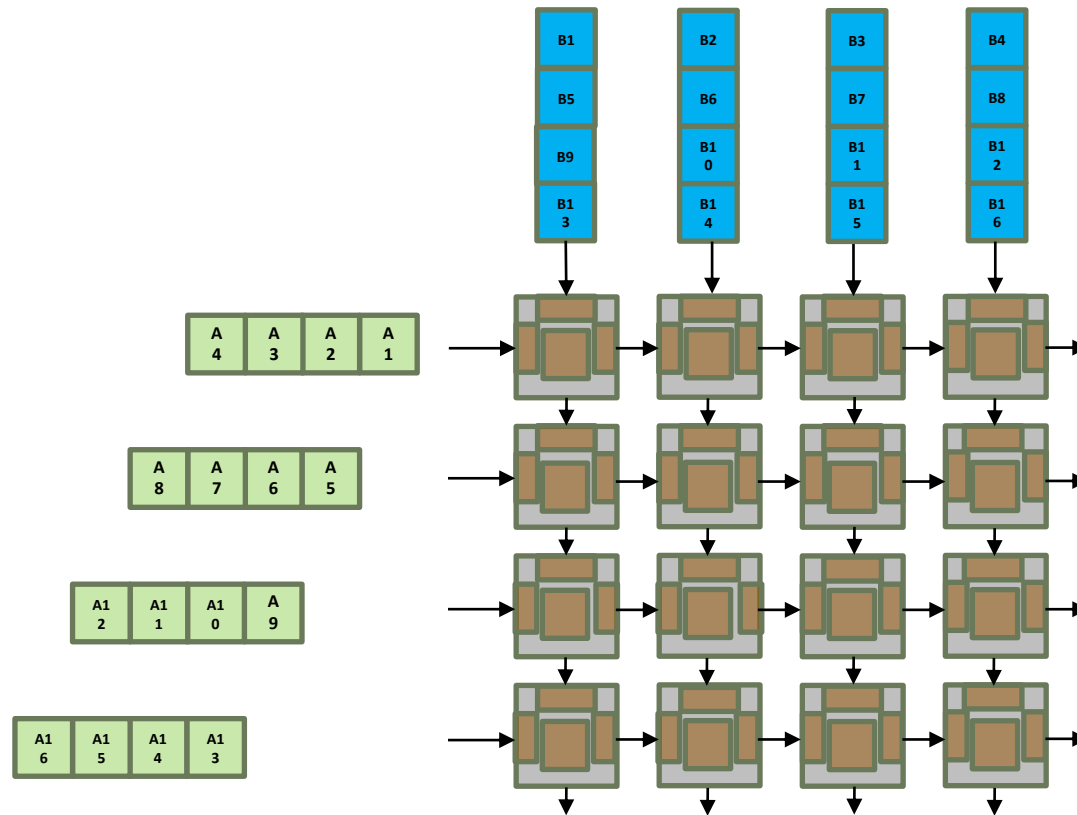


MAC Processing Element (PE)

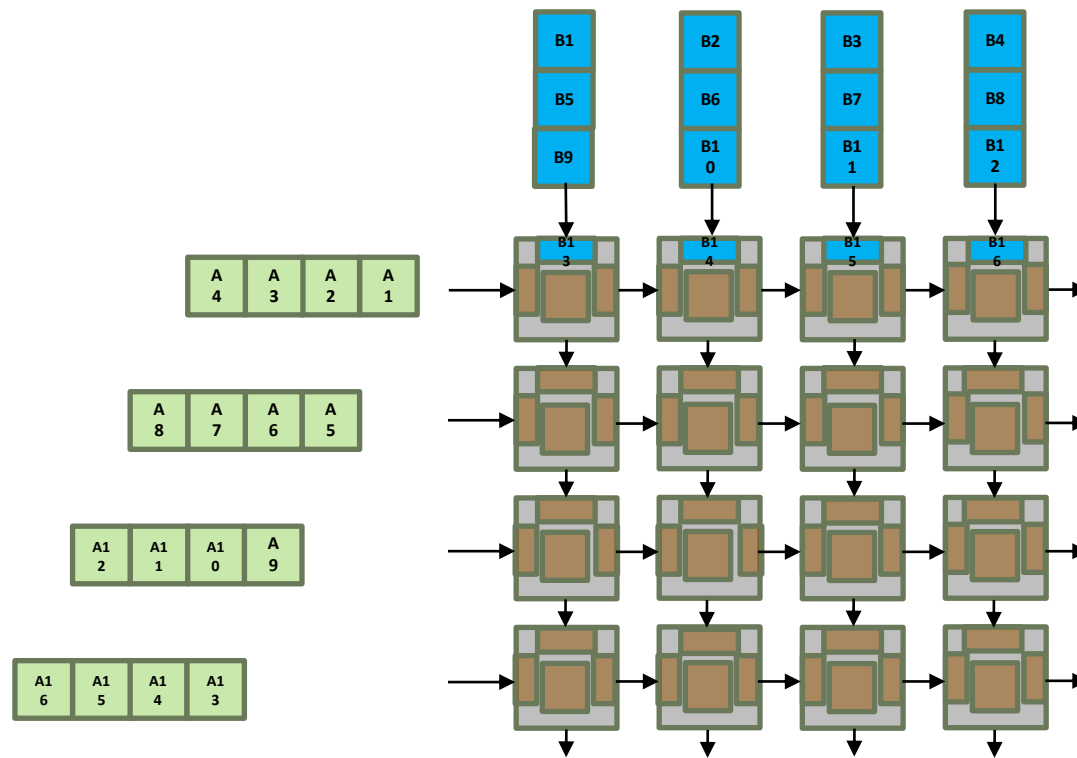
Difference from CPU Architecture

- Registers distributed across PEs
 - One per operand per PE
 - Can be more than one as well
- Operands “forwarded” from one register to the other
 - More energy-efficient than reading large register file
- Stage data through array in deterministic manner
 - No need for hazard checks etc

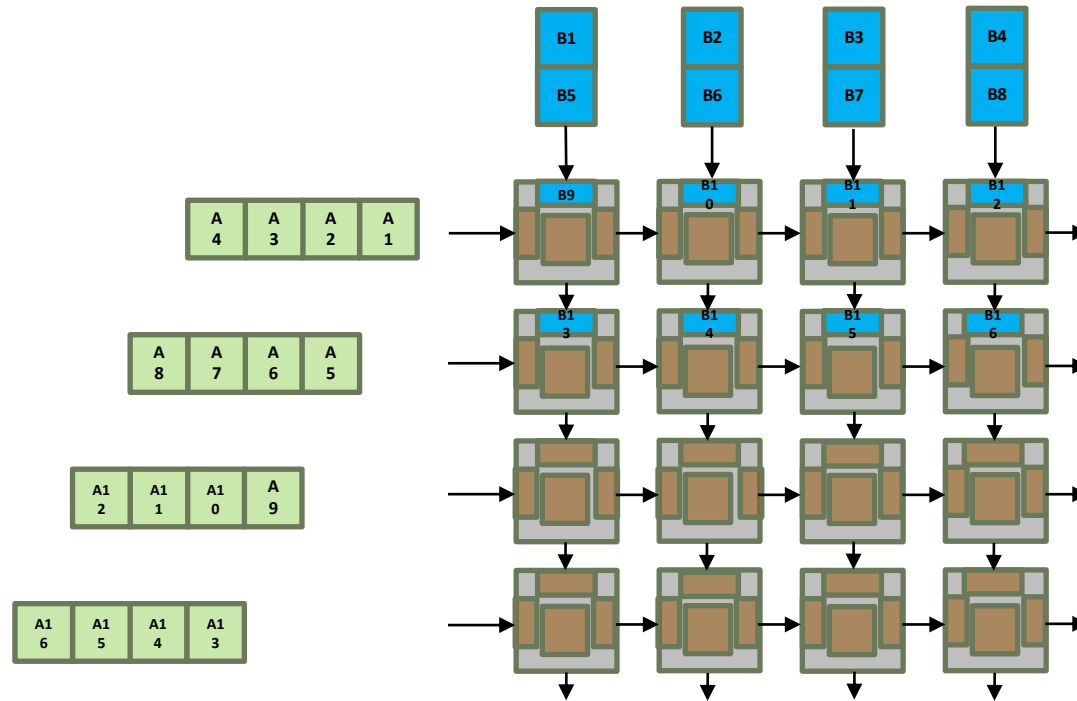
Walkthrough Example



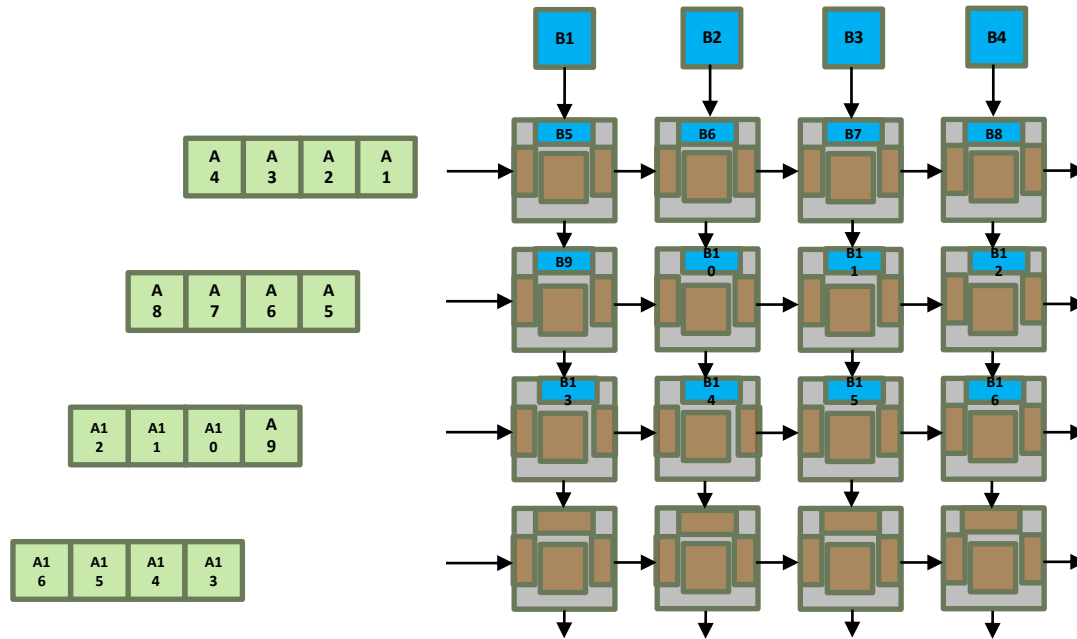
Walkthrough Example



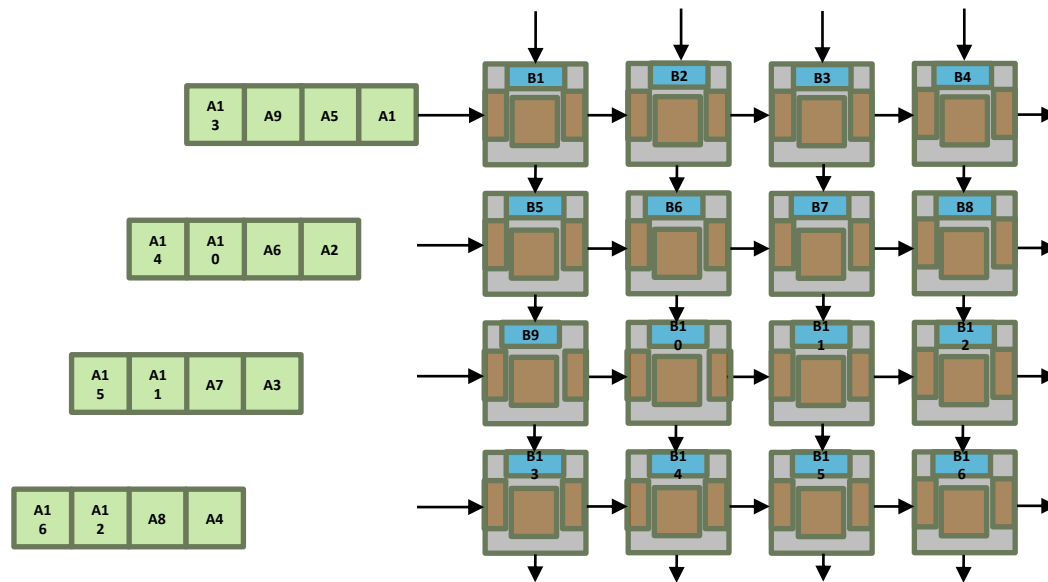
Walkthrough Example



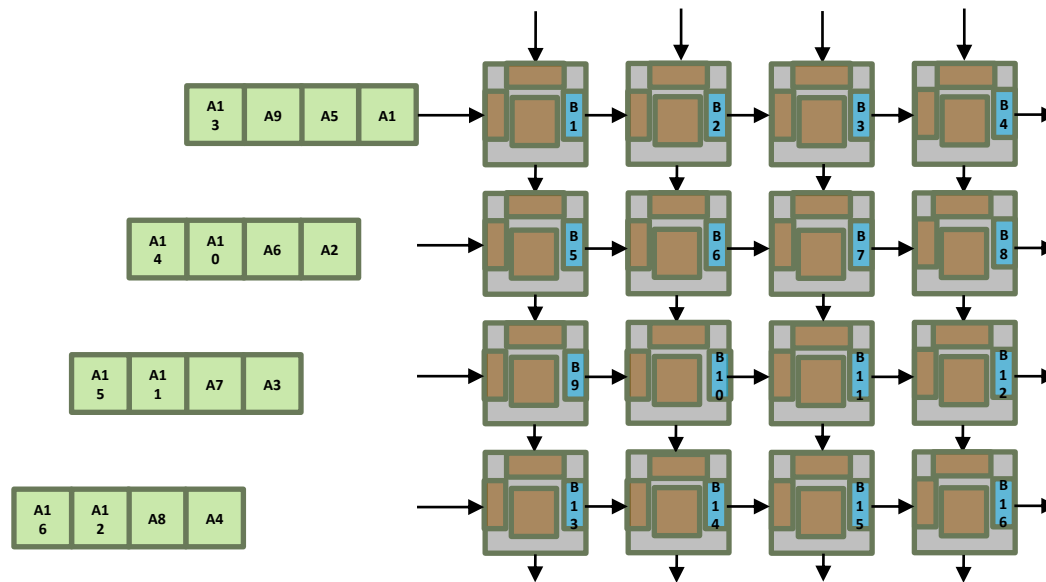
Walkthrough Example



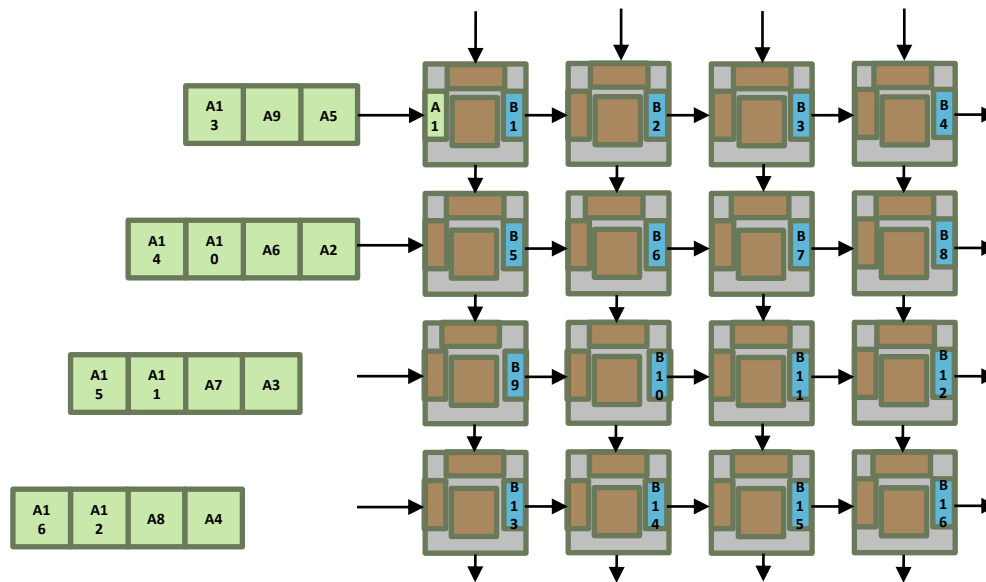
Walkthrough Example



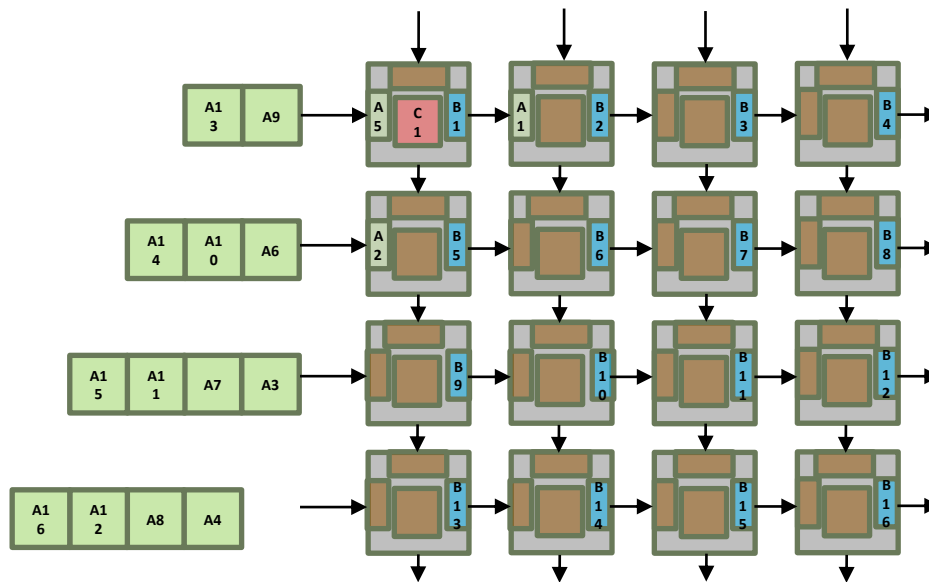
Walkthrough Example



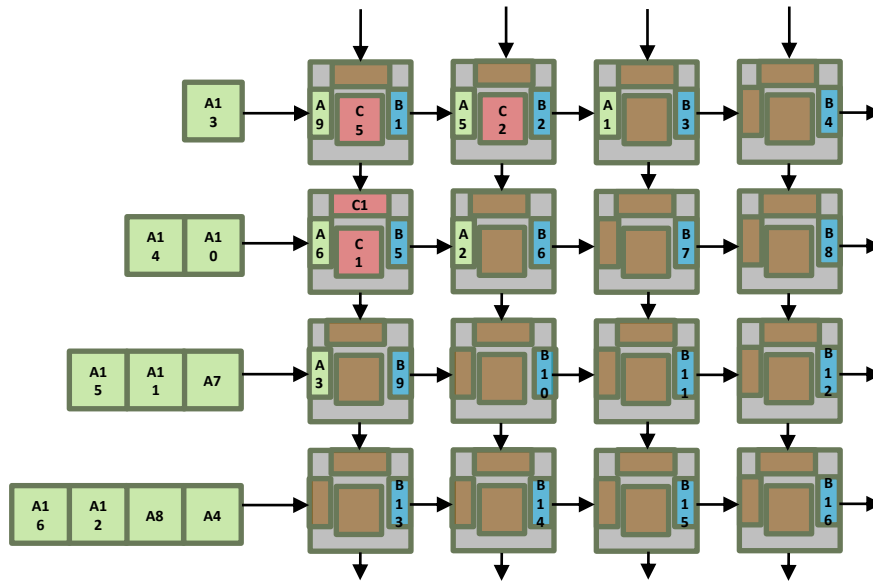
Walkthrough Example



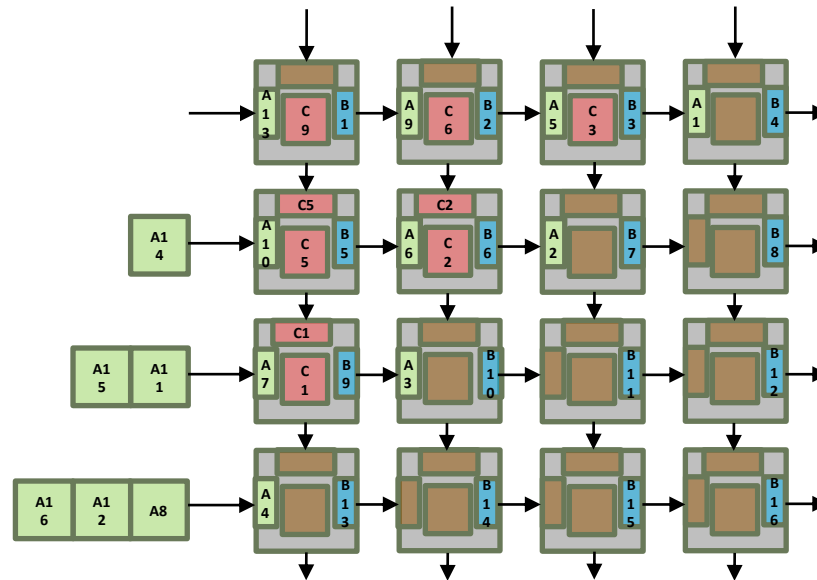
Walkthrough Example



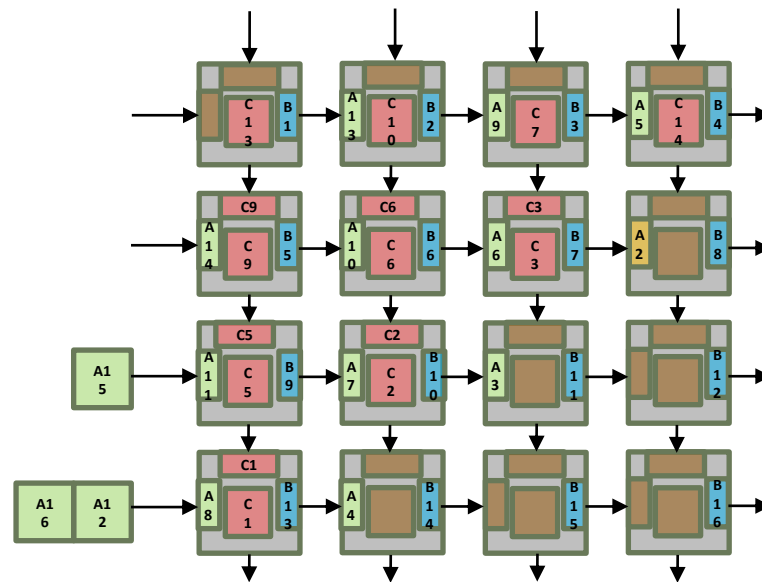
Walkthrough Example



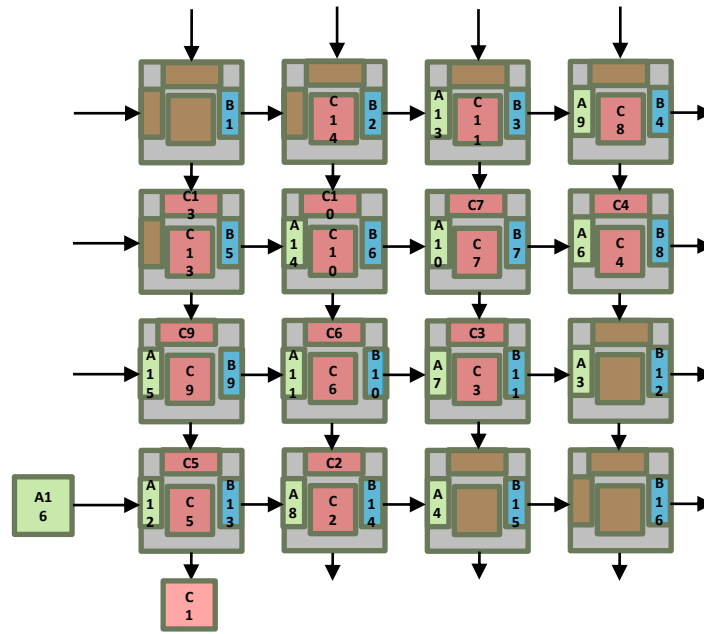
Walkthrough Example



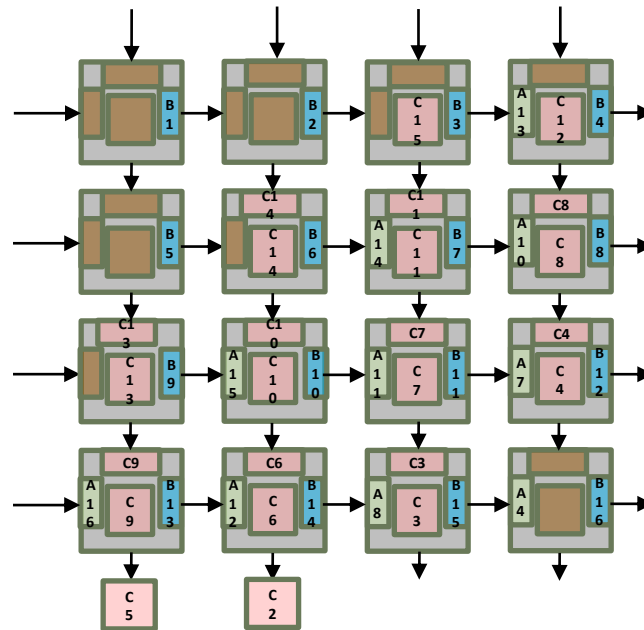
Walkthrough Example



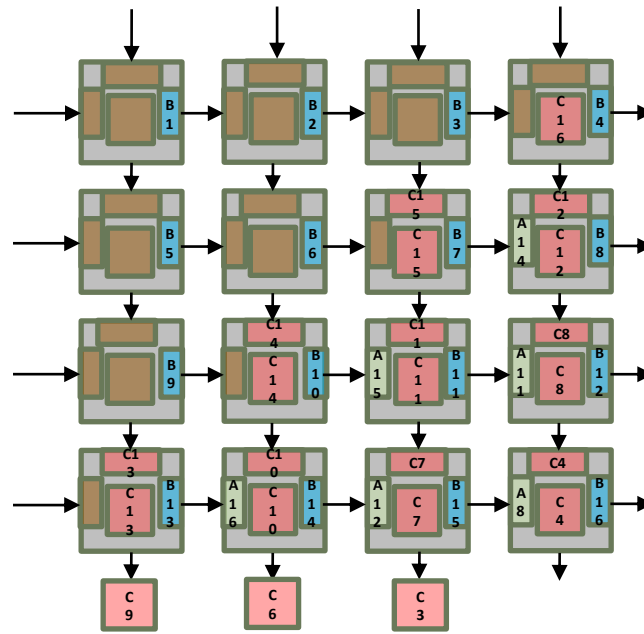
Walkthrough Example



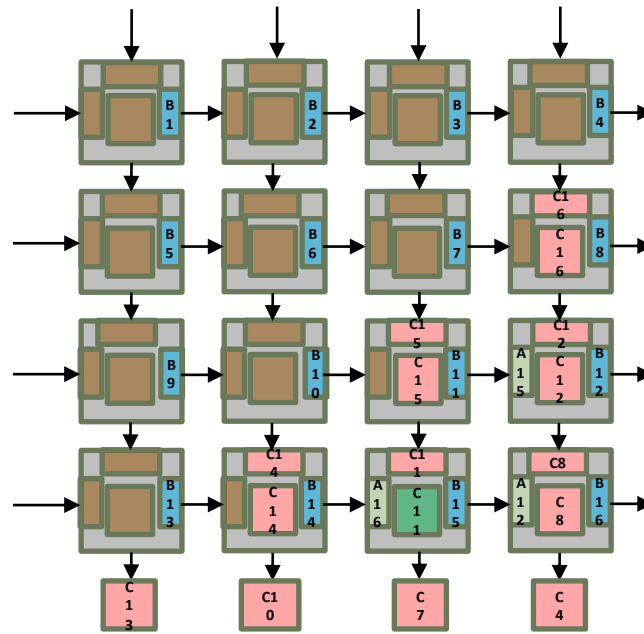
Walkthrough Example



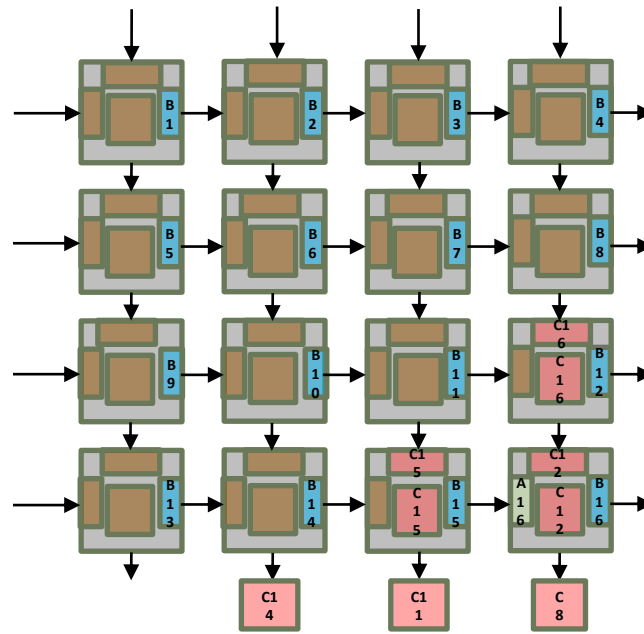
Walkthrough Example



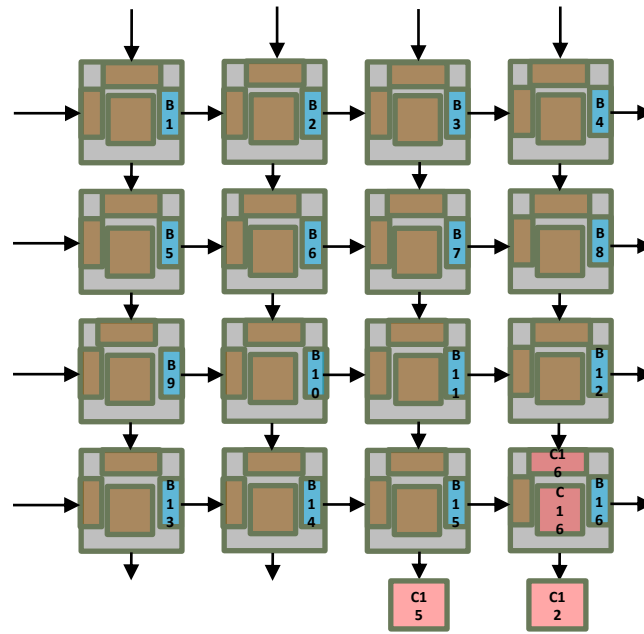
Walkthrough Example



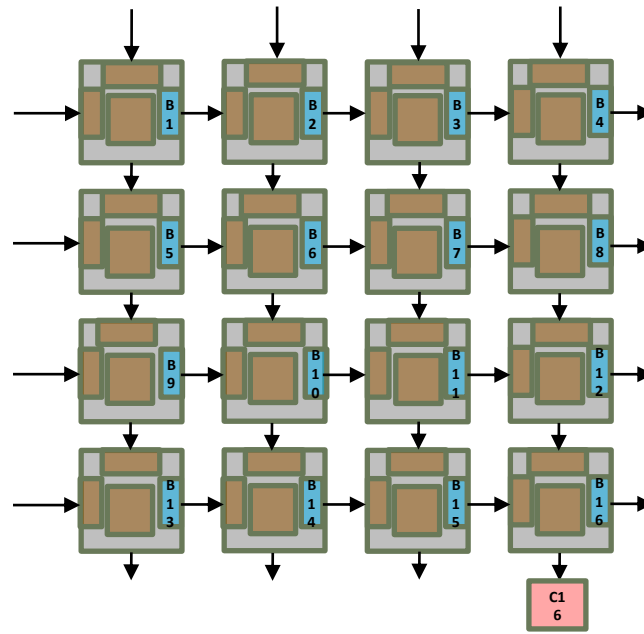
Walkthrough Example



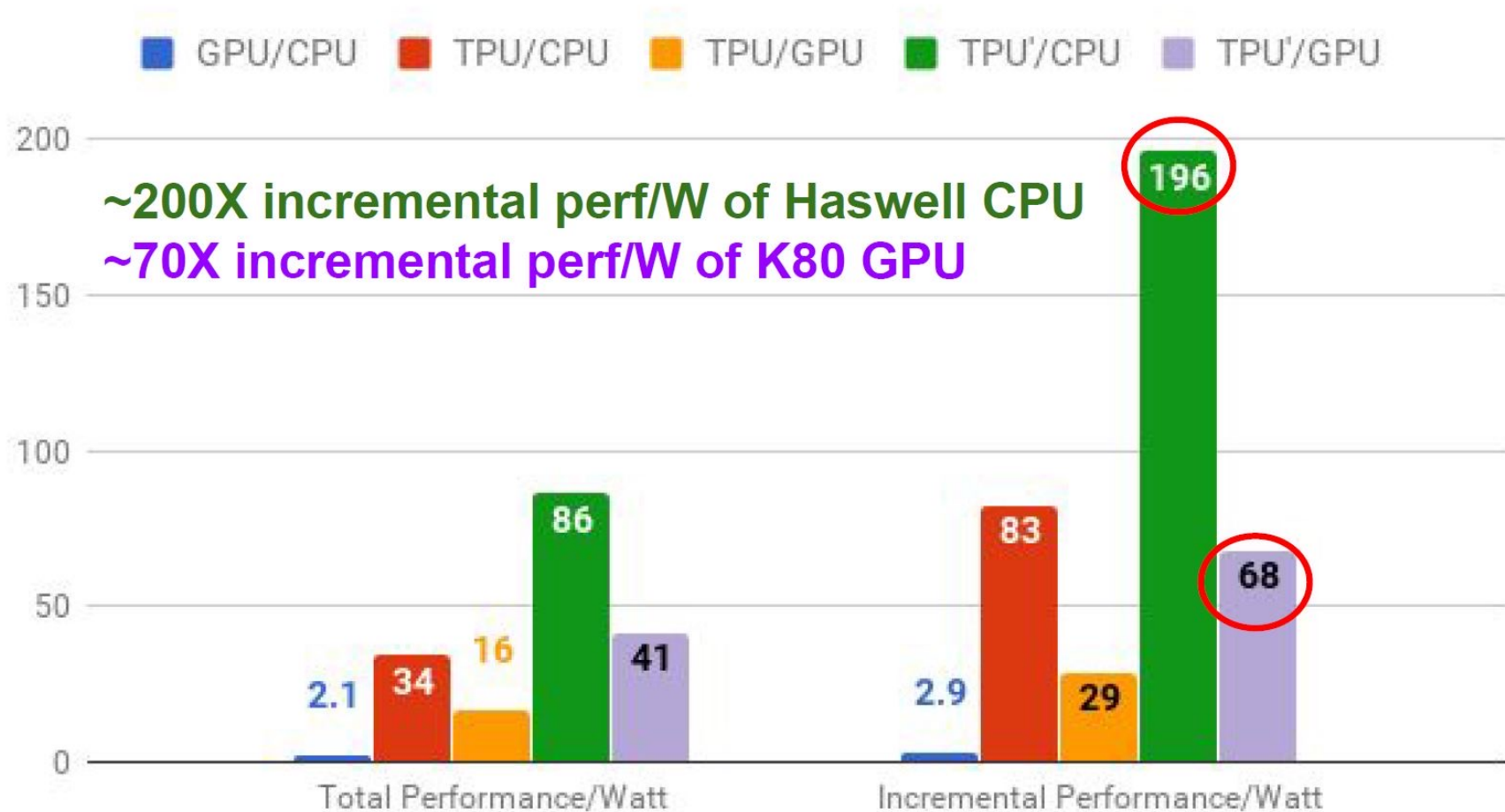
Walkthrough Example



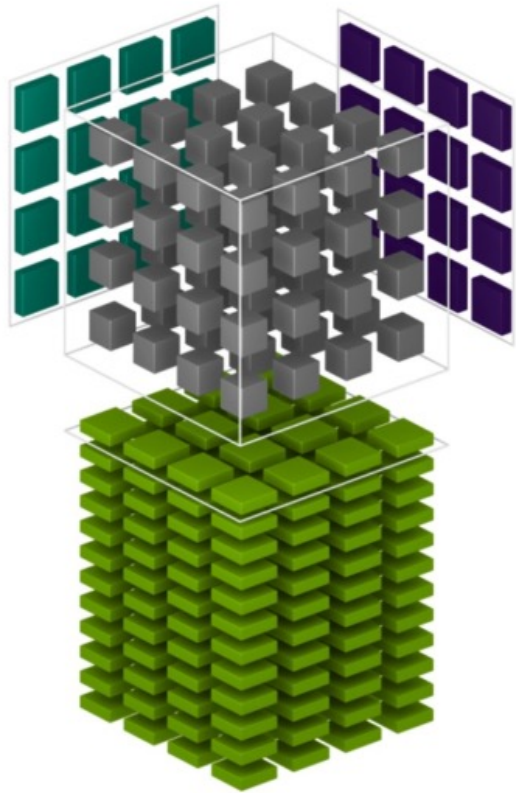
Walkthrough Example



Performance/watt

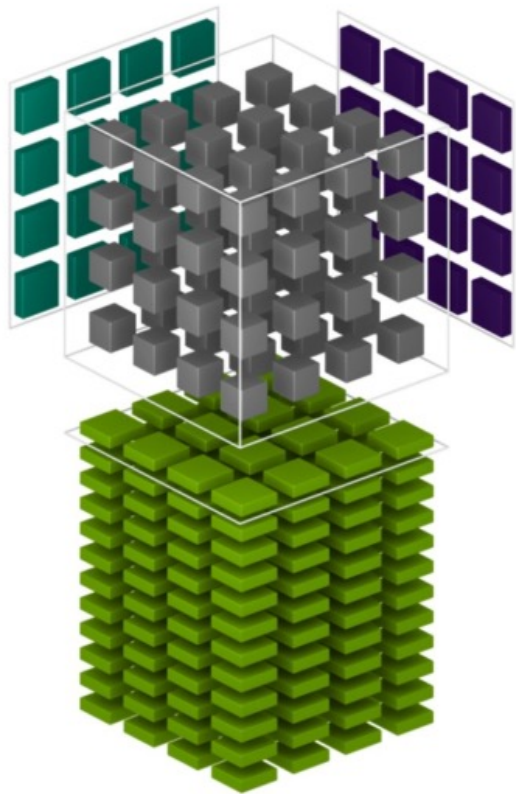


NVIDIA Response: Tensor Cores



$$D = \begin{matrix} \text{FP16 or FP32} & \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} & \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} & \text{FP16} & + & \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix} & \text{FP16 or FP32} \end{matrix}$$

NVIDIA Response: Tensor Cores



$$D = \begin{matrix} \text{FP16 or FP32} & \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} & \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} & + & \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix} \\ & \text{FP16} & \text{FP16} & & \text{FP16 or FP32} \end{matrix}$$

*Google TPU – large tensor engine vs NVIDIA:
multiple tiny tensor engines.
Trade-off?*

Summary

- High Throughput requirements and Energy costs of Data Movement are key drivers towards the trend towards custom accelerators
- Heavy HW-SW Co-Design is used in practice to design accelerators
- Open questions: how to avoid “over”-specialization

Thank you!

Next Lecture: Accelerators-II