# CSC 252: Computer Organization
# Fall 2021: Lecture 26

Accelerators

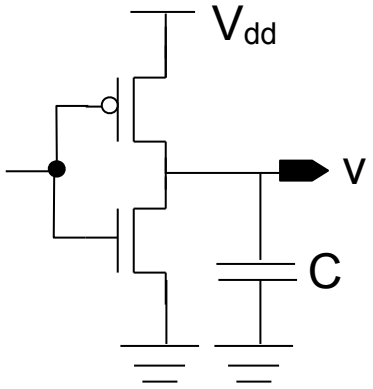
Instructor: Alan Beadle

Department of Computer Science
University of Rochester

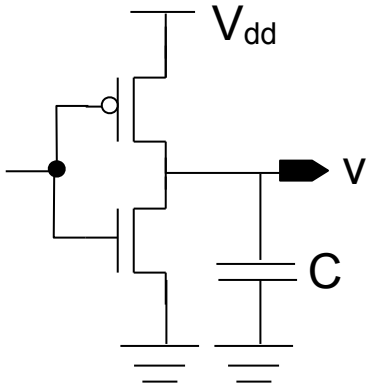
# Announcements

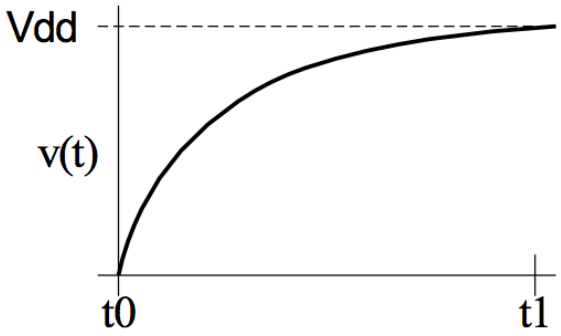
- A5 due Dec 9th (day after last lecture)
  - Let Abhishek know about slip days
- Next lecture is 100% review

# Dynamic Power

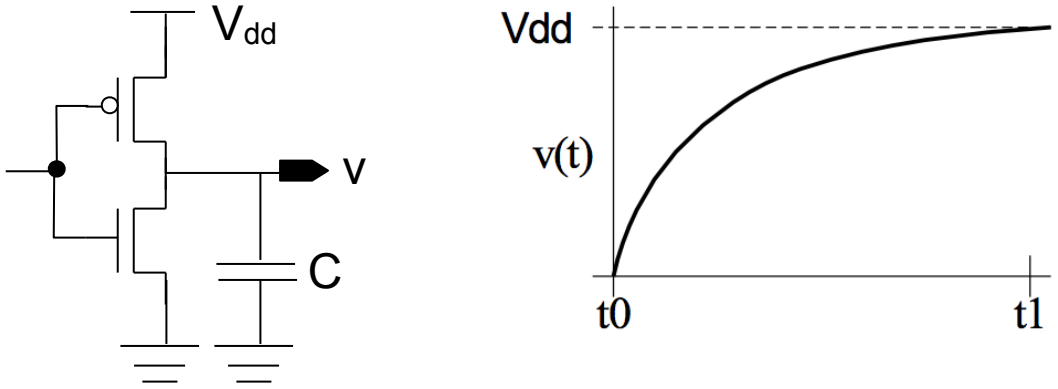


# Dynamic Power



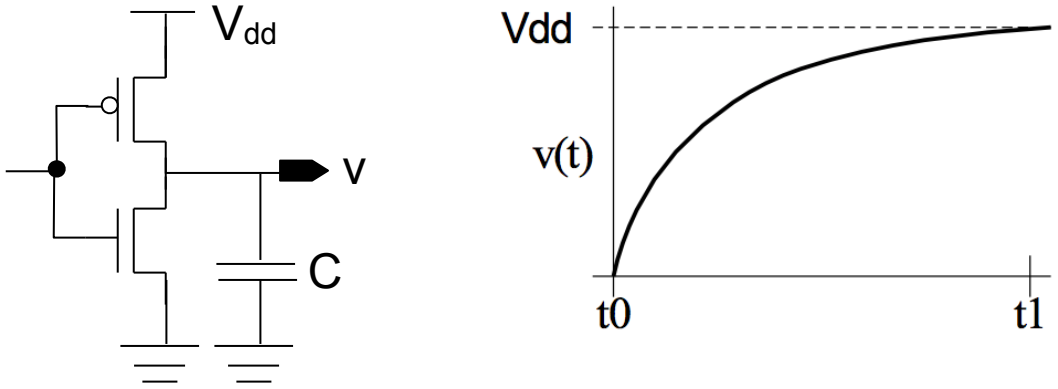
# Dynamic Power




$$E_{sw} = \int_{t_0}^{t_1} P(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot i(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot c\,(dv/dt)\,dt =$$

$$= cV_{dd} \int_{t_0}^{t_1} dv - c\int_{t_0}^{t_1} v \cdot dv = cV_{dd}^{\,2} - 1/2\,cV_{dd}^{\,2} = \boxed{1/2\,cV_{dd}^{\,2}}$$

# Dynamic Power

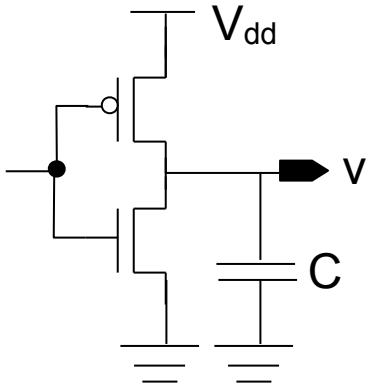


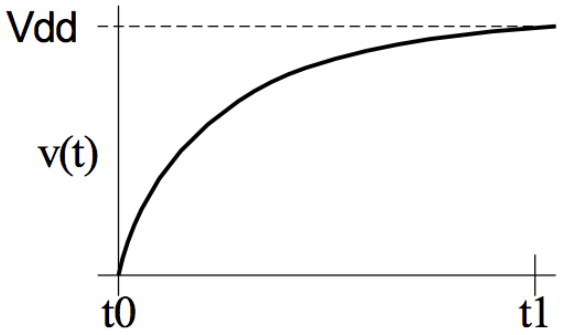

Energy dissipated for every transition (0->1 or 1->0)

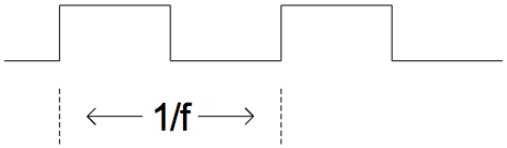
$$E_{sw} = \int_{t_0}^{t_1} P(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot i(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot c\,(dv/dt)\,dt =$$

$$= cV_{dd} \int_{t_0}^{t_1} dv - c\int_{t_0}^{t_1} v \cdot dv = cV_{dd}^{\,2} - 1/2\,cV_{dd}^{\,2} = \boxed{1/2\,cV_{dd}^{\,2}}$$

# Dynamic Power




**Average dynamic power of a transistor:**

$$P = \alpha \cdot (E / T) = \alpha \cdot E f = \tfrac{1}{2} \alpha C V_{dd}^2 f$$

α: switch activity factor. No switching, no dynamic power consumption

# Dynamic Power

$$P = k\,C\,V^2\,f$$

# Dynamic Power

$$P = k\,C\,V^2\,f$$

- Increasing f requires V to be increased proportionally

# Dynamic Power

$$P = k\,C\,V^2\,f$$

- Increasing f requires V to be increased proportionally
  - Intuitively: a higher frequency means a shorter cycle time, which means the critical path of your processor needs to be shorter, which requires faster transistors, which you get by increasing the voltage

# Dynamic Power

$$P = k\ C\ V^2\ f$$

- Increasing f requires V to be increased proportionally
  - Intuitively: a higher frequency means a shorter cycle time, which means the critical path of your processor needs to be shorter, which requires faster transistors, which you get by increasing the voltage
  - “Overclocking” just increases the clock speed without increasing voltage => machine might crash (cycle time shorter than the critical path delay)

# Dynamic Power

$$P = k\,C\,V^2\,f$$

- Increasing f requires V to be increased proportionally
  - Intuitively: a higher frequency means a shorter cycle time, which means the critical path of your processor needs to be shorter, which requires faster transistors, which you get by increasing the voltage
  - “Overclocking” just increases the clock speed without increasing voltage => machine might crash (cycle time shorter than the critical path delay)
  - Corollary: reducing voltage requires reducing frequency

# Dynamic Power

$$P = k\,C\,V^2\,f$$

- Increasing f requires V to be increased proportionally
  - Intuitively: a higher frequency means a shorter cycle time, which means the critical path of your processor needs to be shorter, which requires faster transistors, which you get by increasing the voltage
  - “Overclocking” just increases the clock speed without increasing voltage => machine might crash (cycle time shorter than the critical path delay)
  - Corollary: reducing voltage requires reducing frequency
  - 15% reduction in voltage requires about 15% slow down in frequency

# Dynamic Power

$$P = k\,C\,V^2\,f$$

- Increasing f requires V to be increased proportionally
  - Intuitively: a higher frequency means a shorter cycle time, which means the critical path of your processor needs to be shorter, which requires faster transistors, which you get by increasing the voltage
  - "Overclocking" just increases the clock speed without increasing voltage => machine might crash (cycle time shorter than the critical path delay)
  - Corollary: reducing voltage requires reducing frequency
  - 15% reduction in voltage requires about 15% slow down in frequency
  - What's the impact on dynamic power? $0.85^3 \approx 60\%$ -> 40% dynamic power reduction.

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

- Dynamic power favors parallel processing over higher clock rate

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

- Dynamic power favors parallel processing over higher clock rate
  - Take a core and replicate it 4 times: 4x speedup & **4x power**

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

- Dynamic power favors parallel processing over higher clock rate
  - Take a core and replicate it 4 times: 4x speedup & **4x power**
  - Take a core and clock it 4 times faster: 4x speedup but **64x dynamic power**!

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

- Dynamic power favors parallel processing over higher clock rate
  - Take a core and replicate it 4 times: 4x speedup & **4x power**
  - Take a core and clock it 4 times faster: 4x speedup but **64x dynamic power**!
- Another way to think about this

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

- Dynamic power favors parallel processing over higher clock rate
  - Take a core and replicate it 4 times: 4x speedup & **4x power**
  - Take a core and clock it 4 times faster: 4x speedup but **64x dynamic power**!
- Another way to think about this
  - If a task can be perfectly parallelized by 4 cores, we can reduce the clock frequency of each core to 1/4 while retaining the same performance

# Dynamic Power Favors Parallelisms

$$P = k\ C\ f^3$$

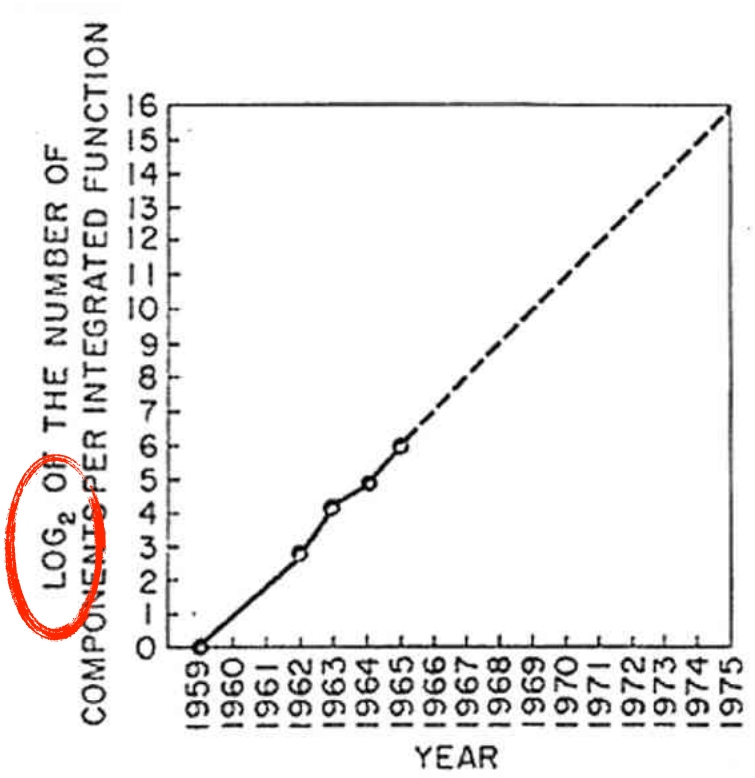
- Dynamic power favors parallel processing over higher clock rate
  - Take a core and replicate it 4 times: 4x speedup & **4x power**
  - Take a core and clock it 4 times faster: 4x speedup but **64x dynamic power**!
- Another way to think about this
  - If a task can be perfectly parallelized by 4 cores, we can reduce the clock frequency of each core to 1/4 while retaining the same performance
  - Dynamic power becomes 4 x $(1/4)^3$ = 1/16

# Moore’s Law

- Gordon Moore in 1965 predicted that the number of transistors doubles every year

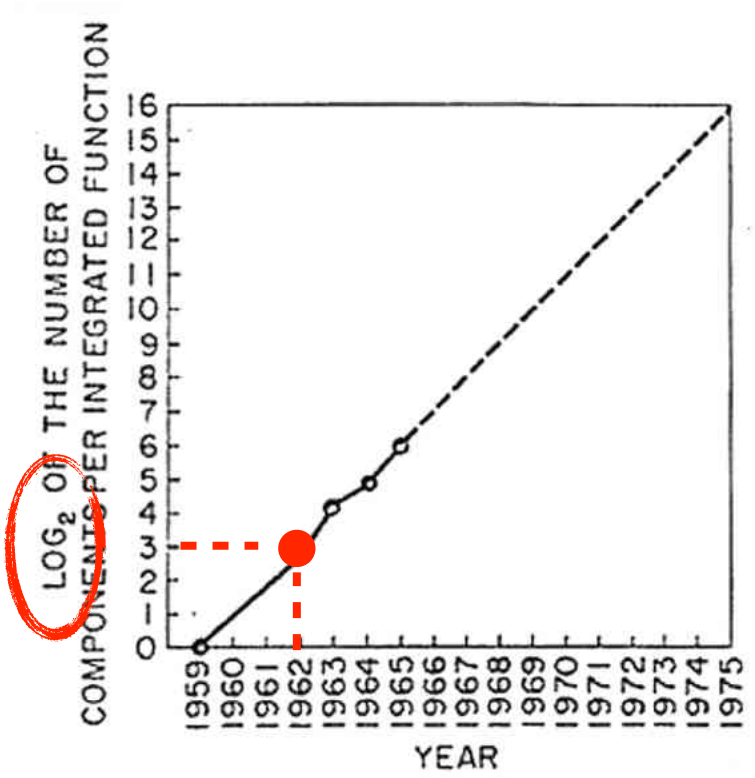
# Moore's Law

- Gordon Moore in 1965 predicted that the number of transistors doubles every year



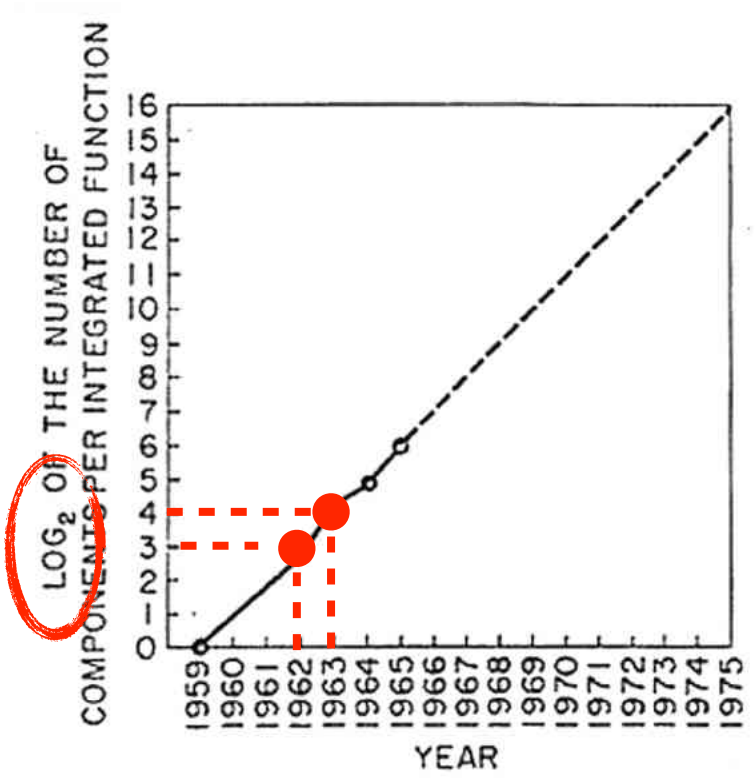

# Moore's Law

- Gordon Moore in 1965 predicted that the number of transistors doubles every year



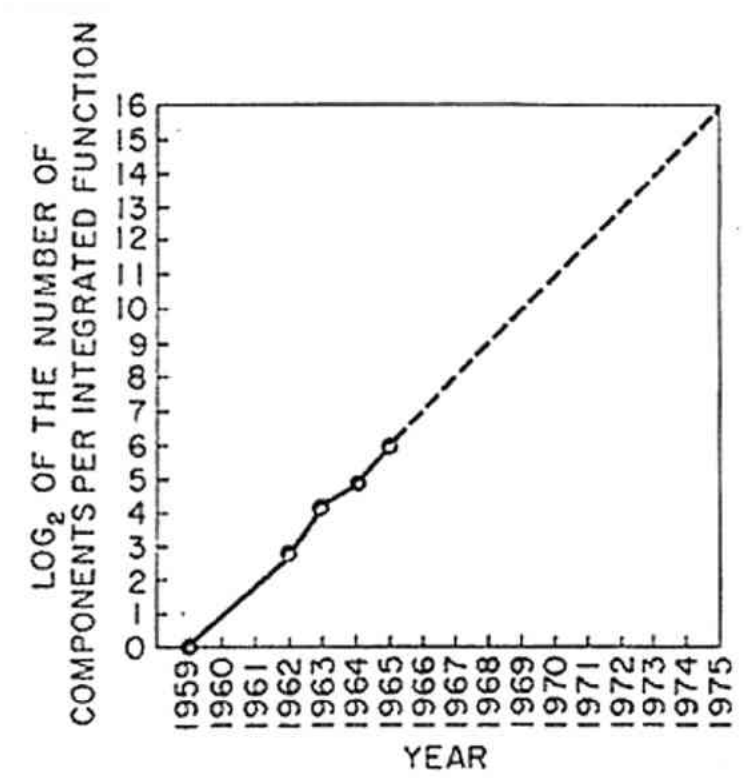

# Moore's Law

- Gordon Moore in 1965 predicted that the number of transistors doubles every year



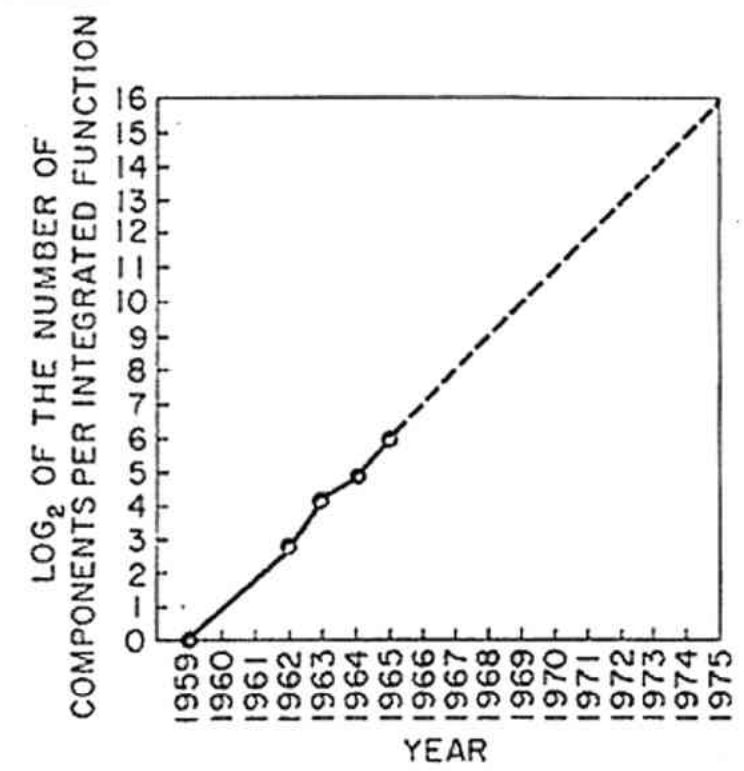

# Moore's Law

- Gordon Moore in 1965 predicted that the number of transistors doubles every year
- In 1975 he revised the prediction to doubling every 2 years

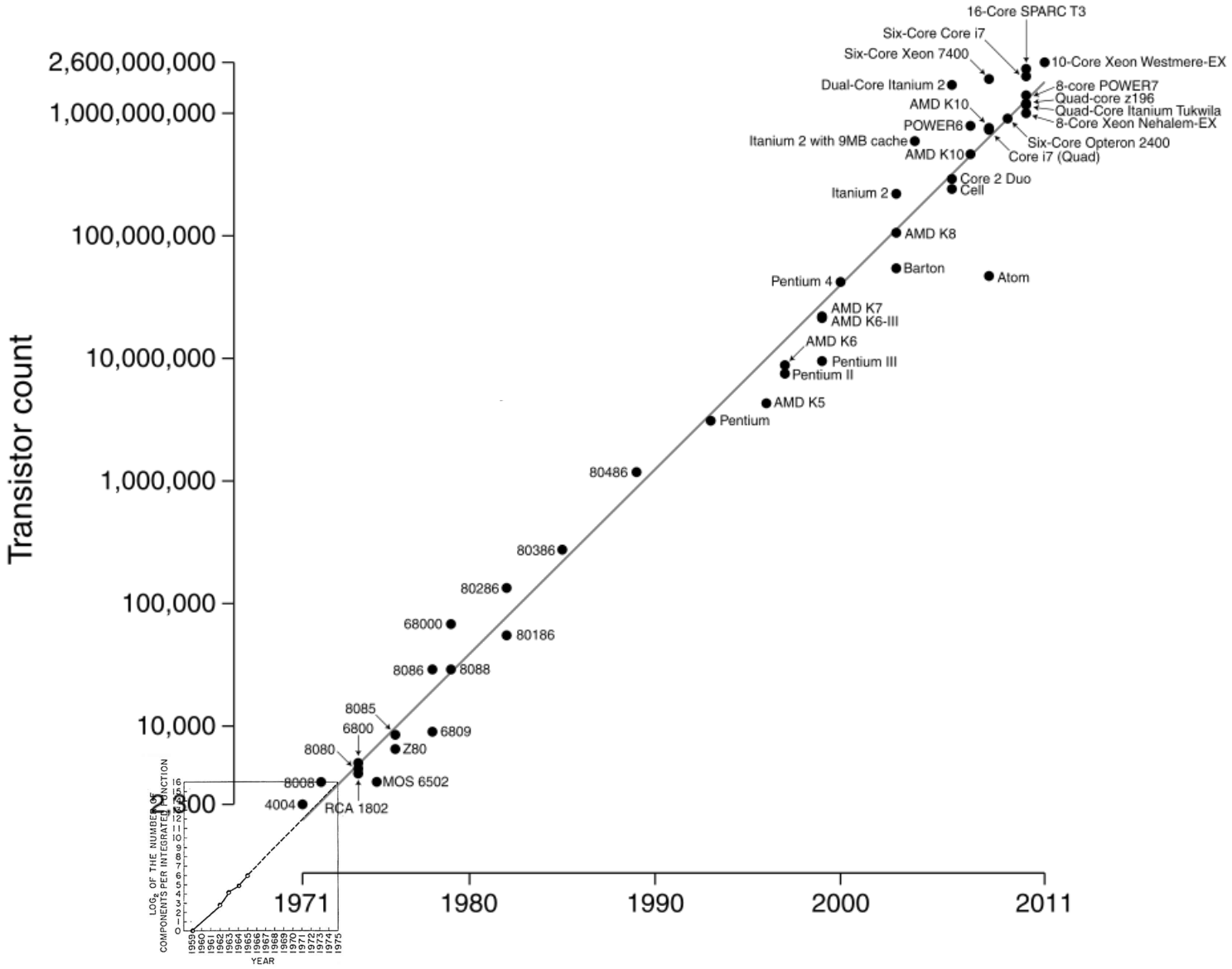


# Moore's Law

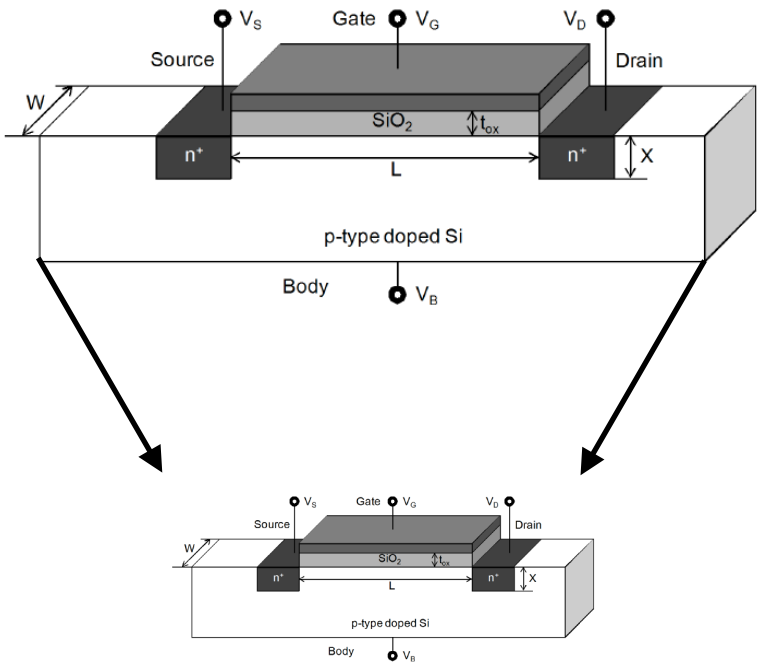
- Gordon Moore in 1965 predicted that the number of transistors doubles every year
- In 1975 he revised the prediction to doubling every 2 years
- Today's widely-known Moore's Law: number of transistors double about every 18 months (Moore never used the number 18…)



# Moore's Law


Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
1971
1980
1990
2000
2011
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
AMD K10
Quad-Core Itanium Tukwila
POWER6
8-Core Xeon Nehalem-EX
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Itanium 2
Cell
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K6-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
8080
Z80
8008
MOS 6502
4004
RCA 1802
LOG2 OF THE NUMBER OF COMPONENTS PER INTEGRATED FUNCTION
YEAR

# Dennard Scaling

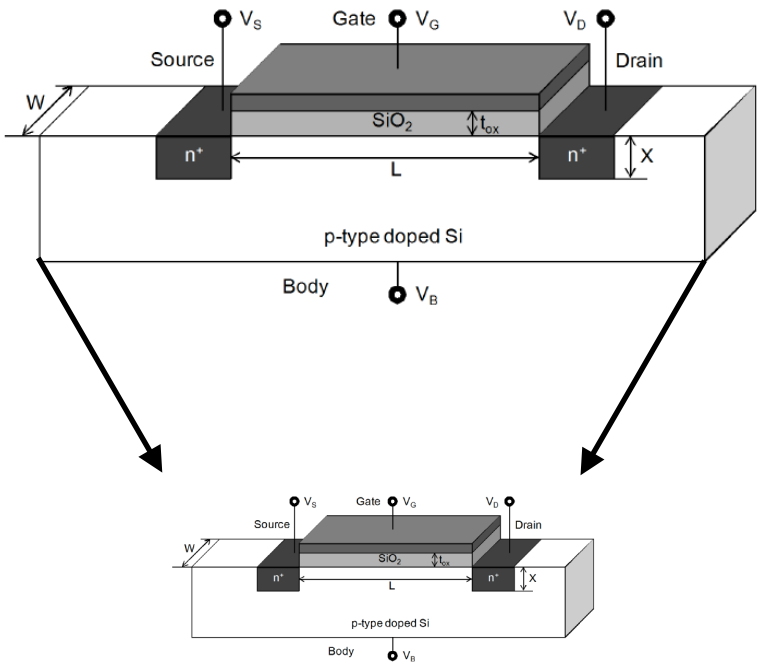



*Scale factor $\alpha$<1*

$\alpha = 0.7$ => 2X more transistors!

Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proc. of IEEE. Dennart et al. 1974.

# Dennard Scaling



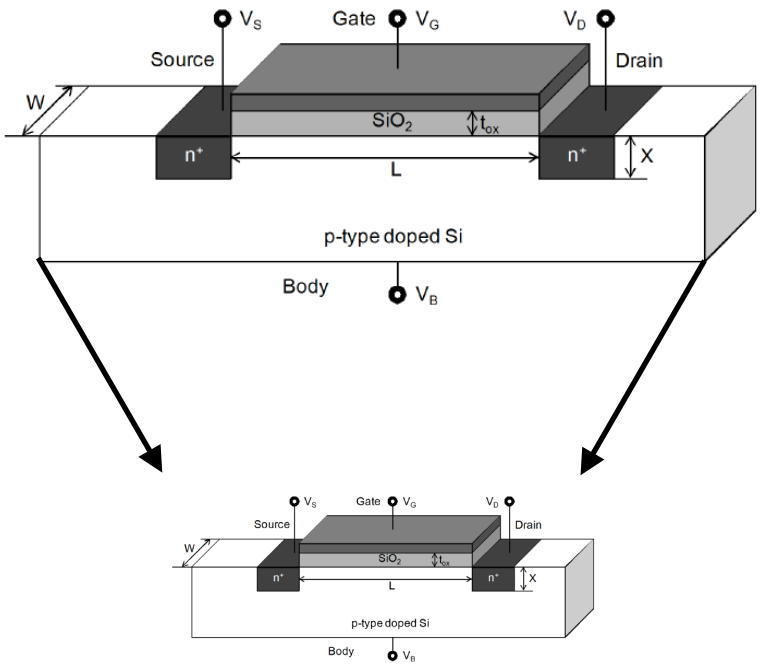

| Parameter | Value | Scaled Value |
|---|---|---|
| Dopant concentrations | Na, Nd | Na/α, Nd/α |
| Dimensions | L, W, Tox | αL, αW, αTox |
| Field | E | E |
| Voltage | V | αV |
| Capacitance | C | αC |
| Current | I | αI |

*Scale factor $\alpha$<1*

$\alpha = 0.7$ => 2X more transistors!

Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proc. of IEEE. Dennart et al. 1974.

# Dennard Scaling



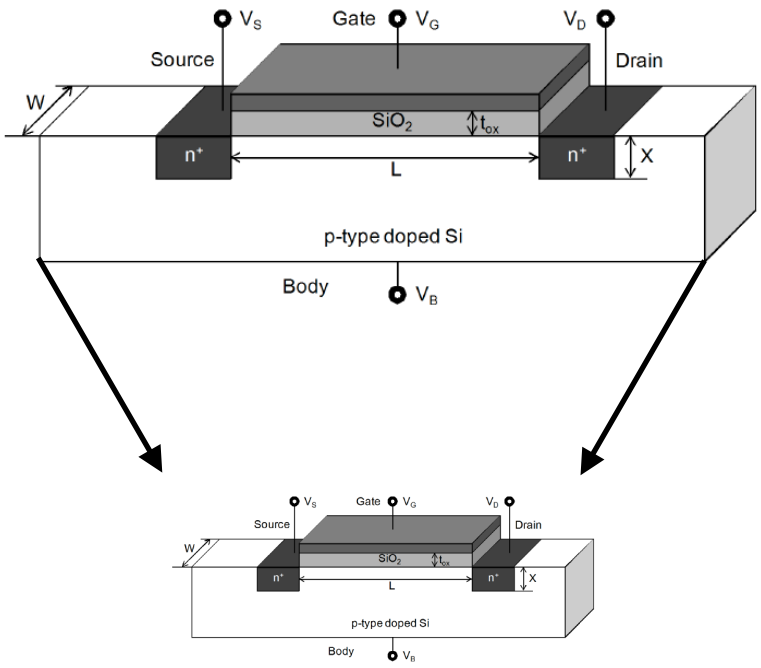

| Parameter | Value | Scaled Value |
|---|---|---|
| Dopant concentrations | Na, Nd | Na/α, Nd/α |
| Dimensions | L, W, Tox | αL, αW, αTox |
| Field | E | E |
| Voltage | V | αV |
| Capacitance | C | αC |
| Current | I | αI |
| Transistors/Area | d | $d/\alpha^2$ |

*Scale factor $\alpha$<1*

$\alpha = 0.7$ => 2X more transistors!

Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proc. of IEEE. Dennart et al. 1974.

# Dennard Scaling



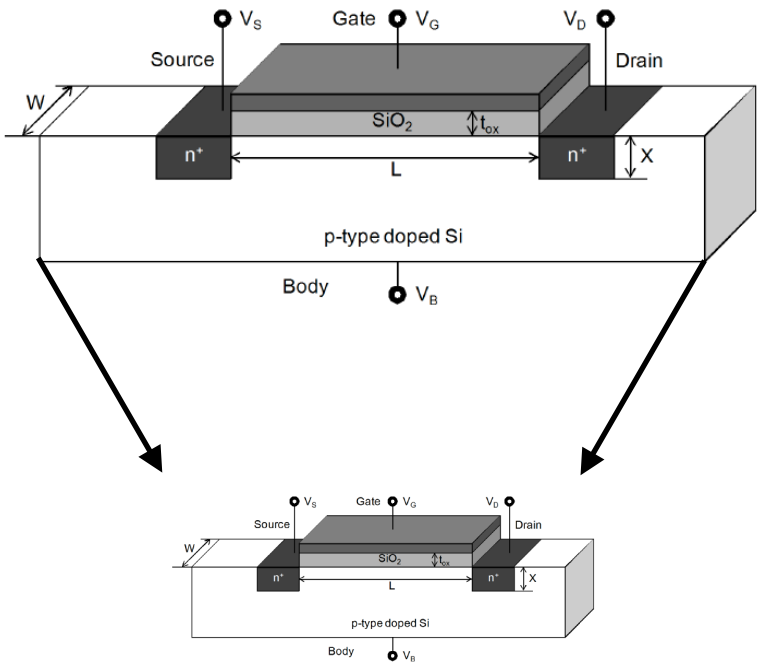

| Parameter | Value | Scaled Value |
|---|---|---|
| Dopant concentrations | Na, Nd | Na/α, Nd/α |
| Dimensions | L, W, Tox | αL, αW, αTox |
| Field | E | E |
| Voltage | V | αV |
| Capacitance | C | αC |
| Current | I | αI |
| Transistors/Area | d | $d/\alpha^2$ |
| Propagation time (~CV/I) | t | αt |
| Frequency (1/t) | f | f/α |

*Scale factor α<1*

$\alpha = 0.7$ => 2X more transistors!

Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proc. of IEEE. Dennart et al. 1974.

# Dennard Scaling




| Parameter | Value | Scaled Value |
|---|---|---|
| Dopant concentrations | Na, Nd | Na/α, Nd/α |
| Dimensions | L, W, Tox | αL, αW, αTox |
| Field | E | E |
| Voltage | V | αV |
| Capacitance | C | αC |
| Current | I | αI |
| Transistors/Area | d | $d/\alpha^2$ |
| Propagation time (~CV/I) | t | αt |
| Frequency (1/t) | f | f/α |
| Power ($CV^2f$) | P | $\alpha^2 P$ |
| **Power/area (Power density)** | **Pd** | **Pd** |

*Scale factor α<1*

$\alpha = 0.7 \Rightarrow 2X$ more transistors!

Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions. Proc. of IEEE. Dennart et al. 1974.

# Implications of Dennard Scaling and Moore's Law

- Each new processor generation can have more transistors (Moore's Law), and will run at higher frequency but won't consume more power under the same area budget.

# Implications of Dennard Scaling and Moore's Law

- Each new processor generation can have more transistors (Moore's Law), and will run at higher frequency but won't consume more power under the same area budget.
- More transistors means better microarchitecture, which leads to better performance even under the same frequency.

# Implications of Dennard Scaling and Moore's Law

- Each new processor generation can have more transistors (Moore's Law), and will run at higher frequency but won't consume more power under the same area budget.
- More transistors means better microarchitecture, which leads to better performance even under the same frequency.
- Higher frequency means better performance even under the same microarchitecture.

# Implications of Dennard Scaling and Moore's Law

- Each new processor generation can have more transistors (Moore's Law), and will run at higher frequency but won't consume more power under the same area budget.
- More transistors means better microarchitecture, which leads to better performance even under the same frequency.
- Higher frequency means better performance even under the same microarchitecture.
- Overall, software gets a free ride: wait for the next generation of hardware and performance will naturally increase without consuming more power.

# Implications of Dennard Scaling and Moore's Law

- Each new processor generation can have more transistors (Moore's Law), and will run at higher frequency but won't consume more power under the same area budget.
- More transistors means better microarchitecture, which leads to better performance even under the same frequency.
- Higher frequency means better performance even under the same microarchitecture.
- Overall, software gets a free ride: wait for the next generation of hardware and performance will naturally increase without consuming more power.

> Moore's law gave us more transistors;
> Dennard scaling made them useful.
>
> Bob Colwell, DAC 2013, June 4, 2013

# 2005: End of Dennard Scaling

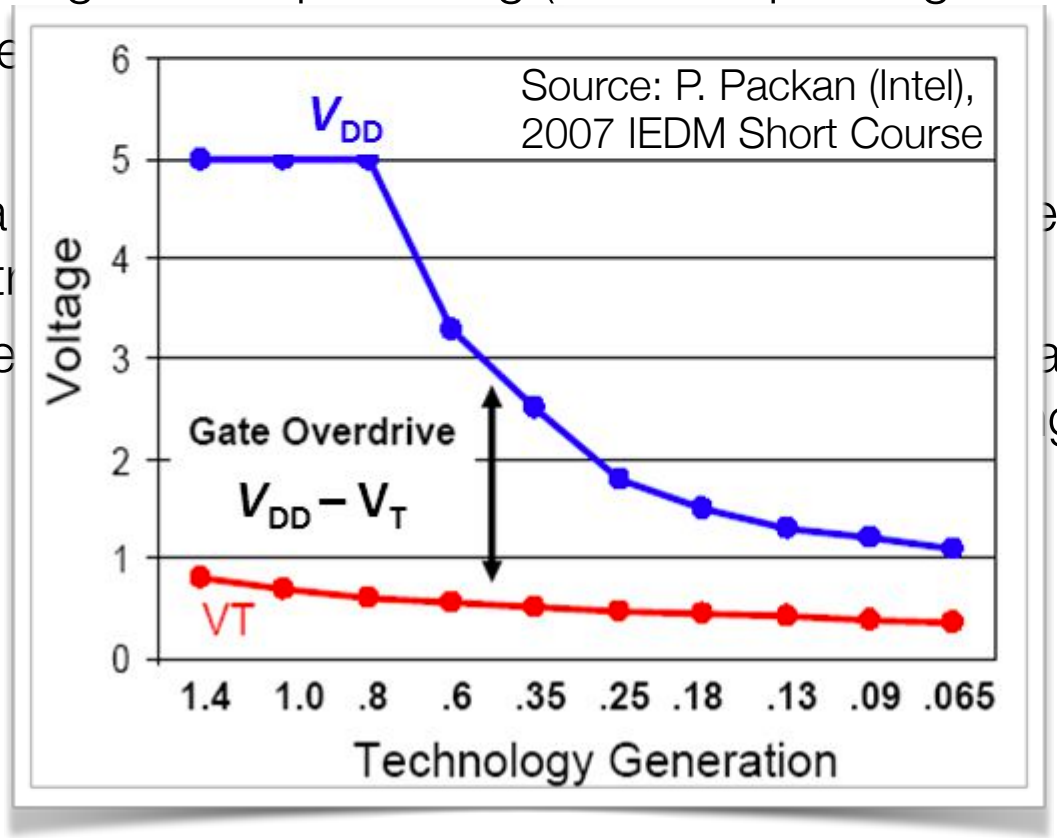
- What Happened?
  - Supply voltage $V_{dd}$ stops scaling (Can't drop voltage below ~1 V)
  - Remember Power = $CV^2f$

# 2005: End of Dennard Scaling

- What Happened?
  - Supply voltage $V_{dd}$ stops scaling (Can't drop voltage below ~1 V)
  - Remember Power = $CV^2f$
- Why?
  - There is a fundamental limit as to how much voltage we need to switch a transistor, called threshold voltage ($V_{th}$).
  - $V_{th}$ stopped scaling because leakage power/reliability/variation becomes huge issues, and accordingly $V_{dd}$ stops scaling

# 2005: End of Dennard Scaling

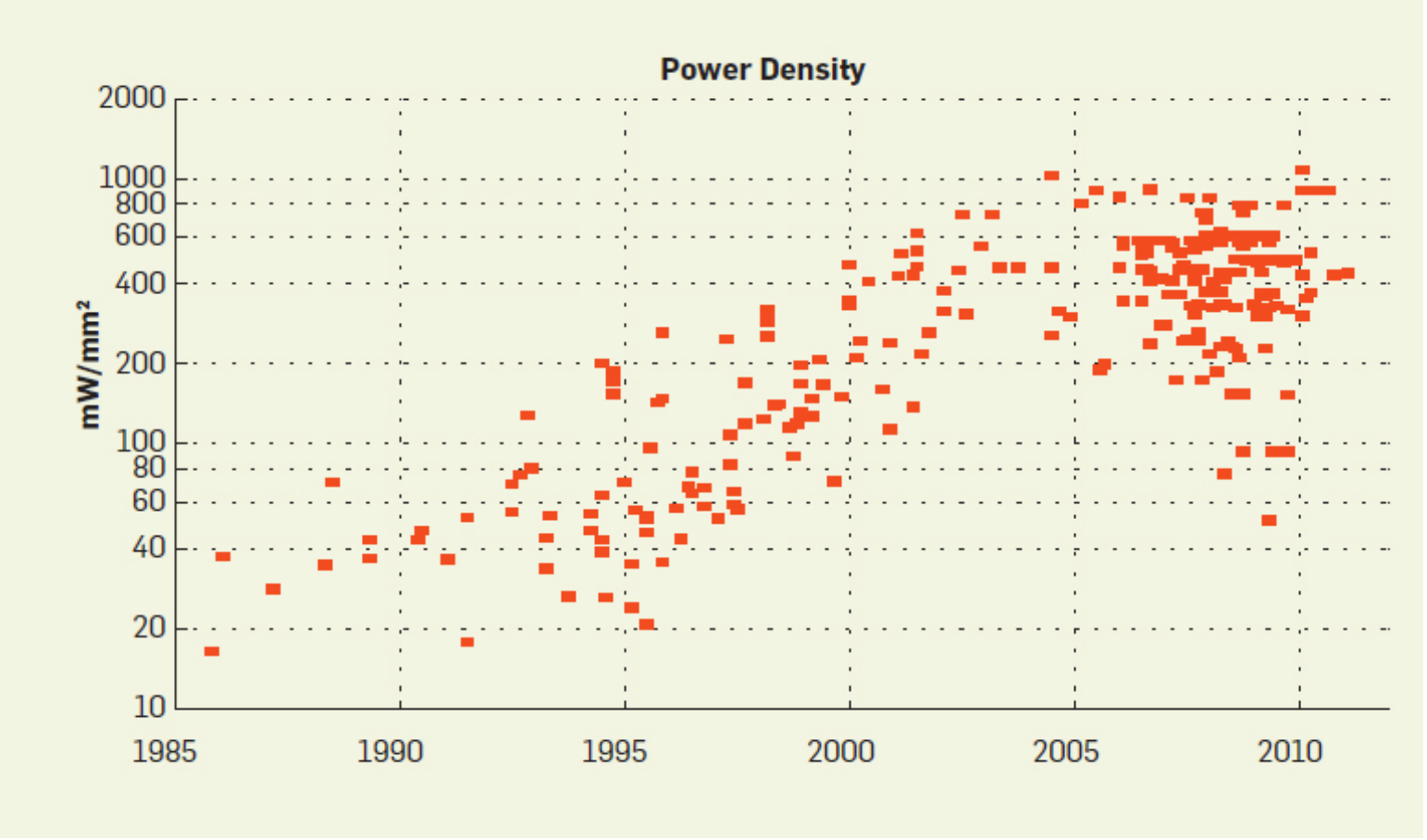
- What Happened?
  - Supply voltage $V_{dd}$ stops scaling (Can't drop voltage below ~1 V)
  - Remembe
- Why?
  - There is a  need to switch a tr
  - $V_{th}$ stoppe ariation becomes  g



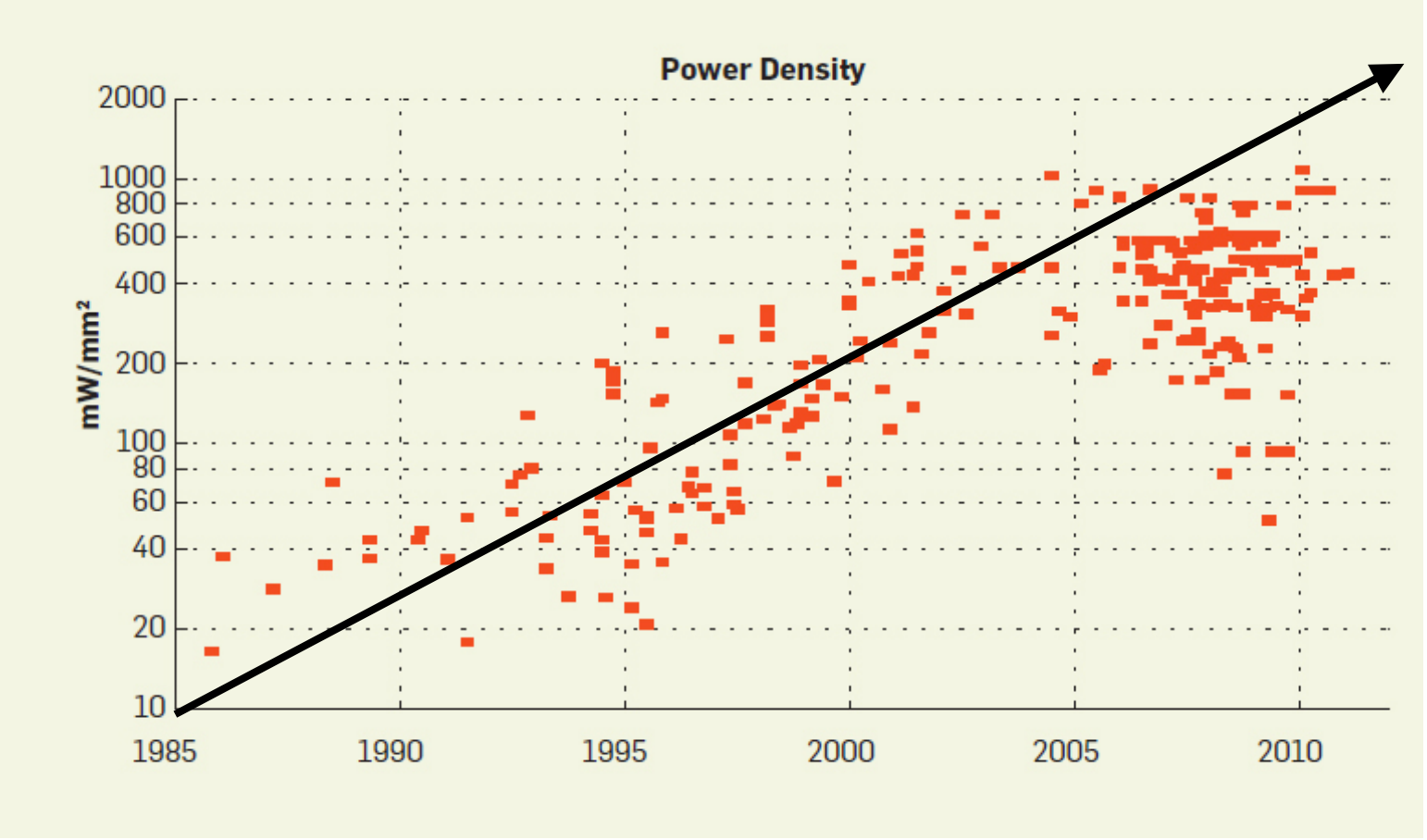
# 2005: End of Dennard Scaling

- What Happened?
  - Supply voltage $V_{dd}$ stops scaling (Can't drop voltage below ~1 V)
  - Remember Power = $CV^2f$
- Why?
  - There is a fundamental limit as to how much voltage we need to switch a transistor, called threshold voltage ($V_{th}$).
  - $V_{th}$ stopped scaling because leakage power/reliability/variation becomes huge issues, and accordingly $V_{dd}$ stops scaling
- The demise of Dennard Scaling means the power density (power consumption per unit area) will increase rather than staying stable.

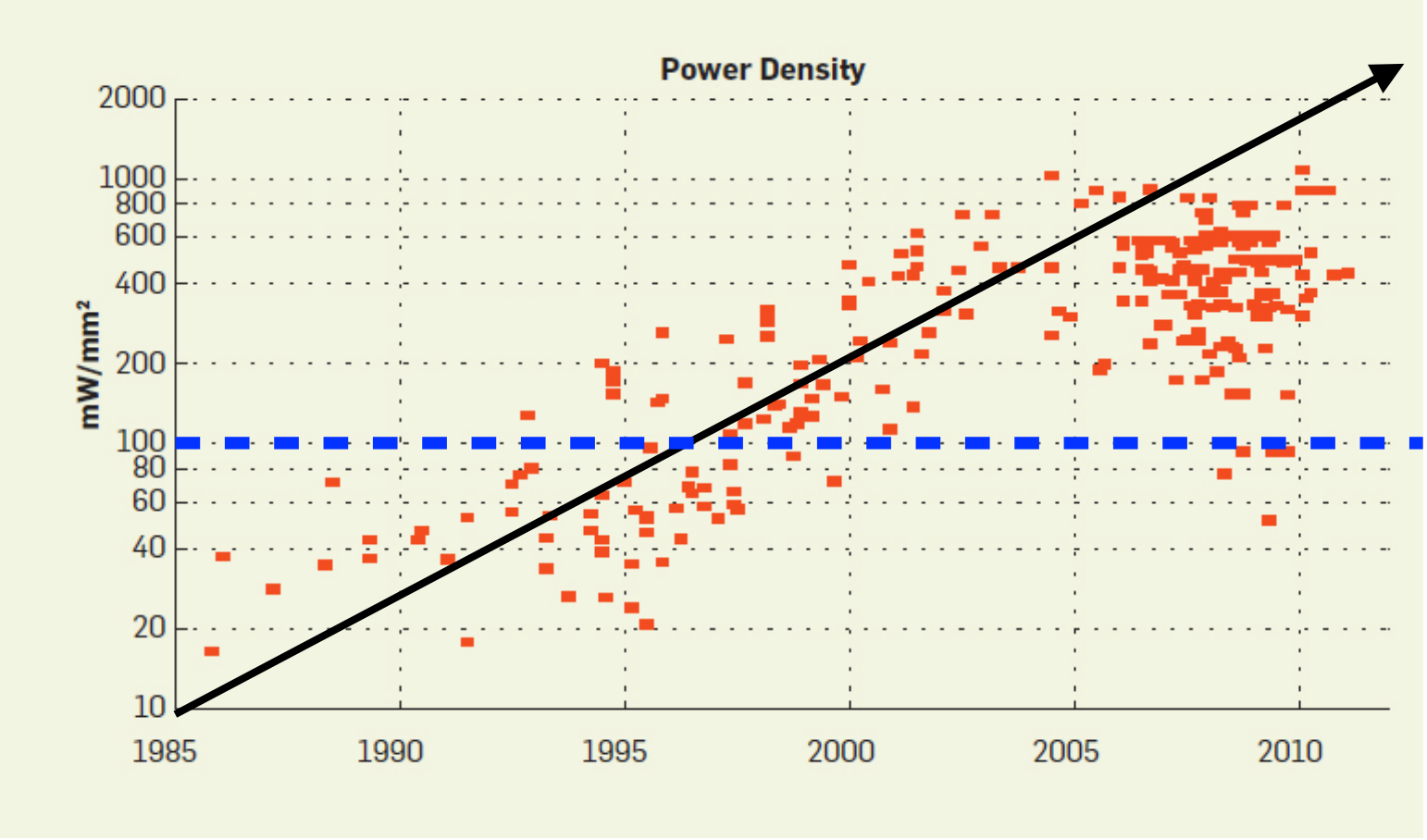
# 2005: End of Dennard Scaling



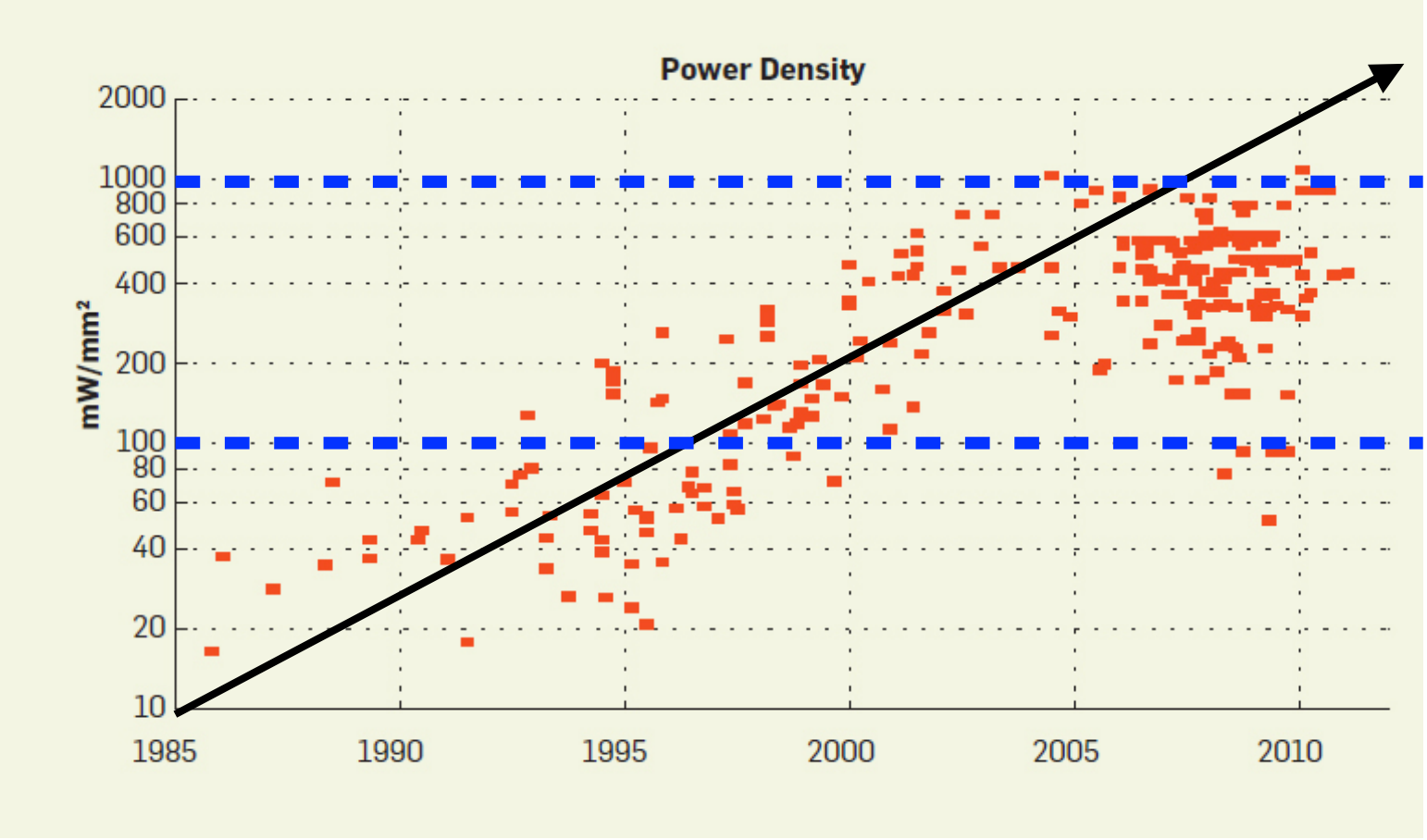
# 2005: End of Dennard Scaling



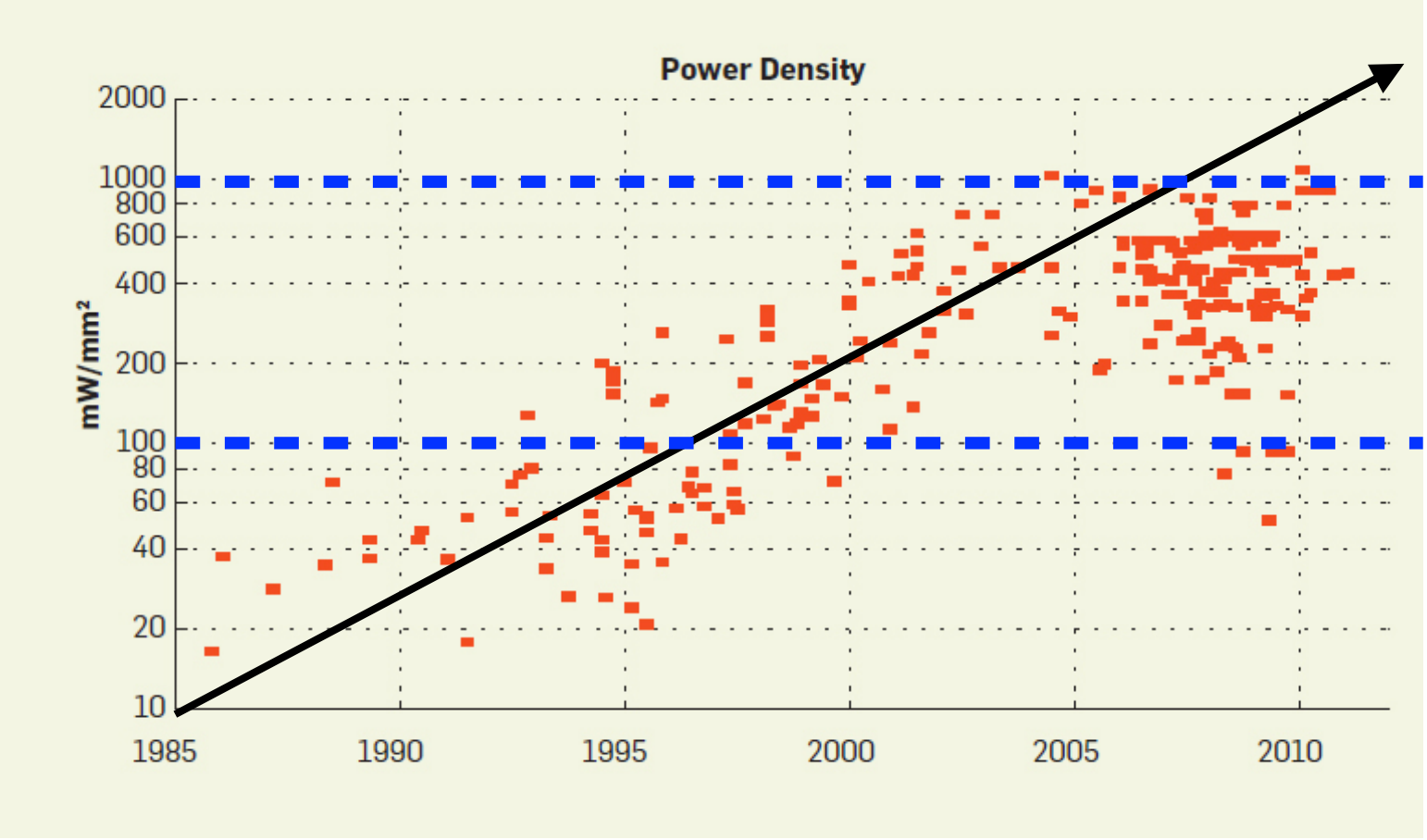
# 2005: End of Dennard Scaling



# 2005: End of Dennard Scaling



# 2005: End of Dennard Scaling



# Dark Silicon

*n.* [därk, sĭl′ĭ-kən, -kŏn′]

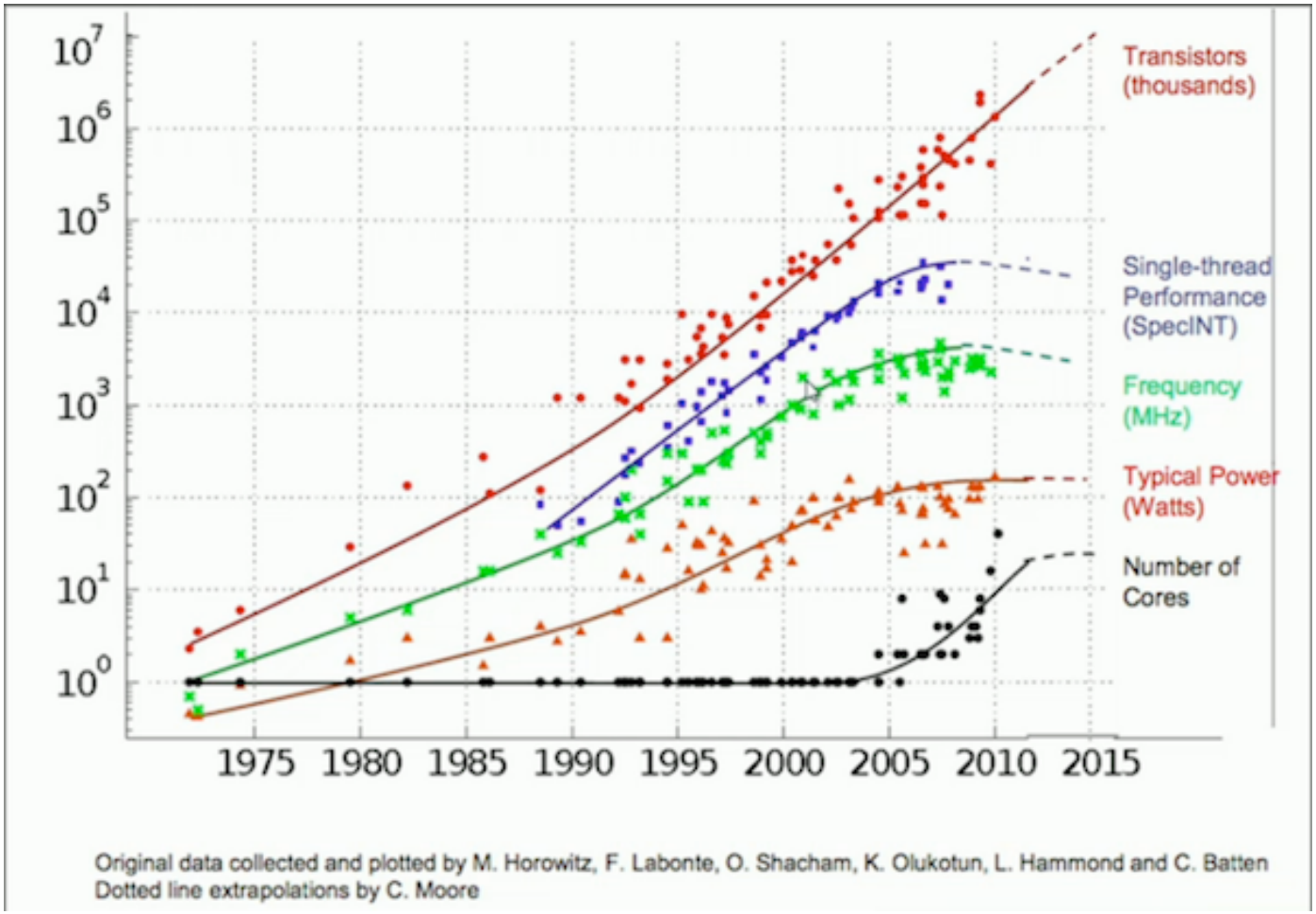
More transistors on chip (due to Moore's Law), but a growing fraction cannot actually be used due to power limits (due to the end of Dennard Scaling).

# 2005: End of Dennard Scaling

- Initial response has been to lower frequency and increase cores / chip

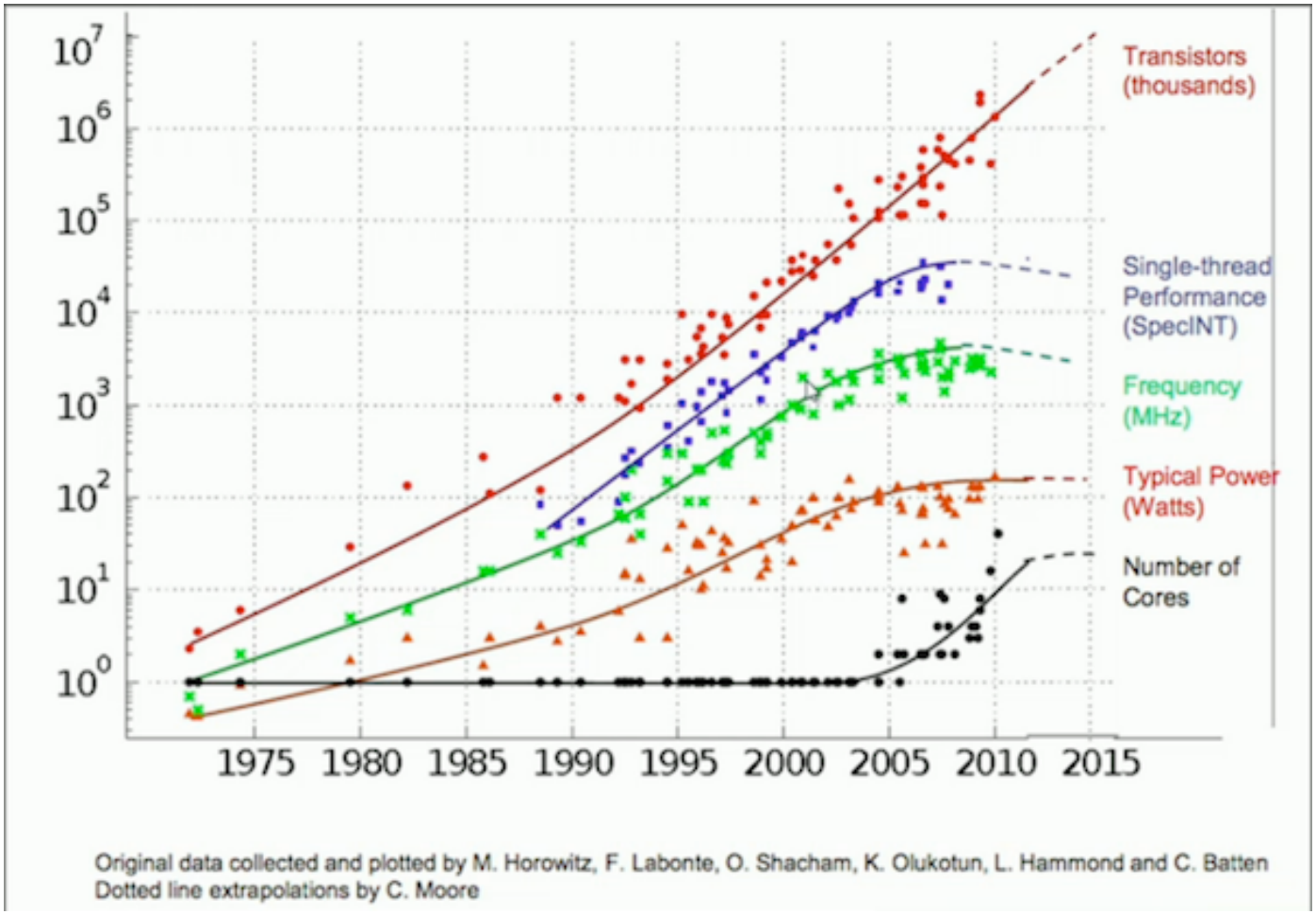
# 2005: End of Dennard Scaling

- Initial response has been to lower frequency and increase cores / chip



# 2005: End of Dennard Scaling

- Initial response has been to lower frequency and increase cores / chip
- There is a limit to core scaling. Why?



# 2007: A Revolutionary New Computer

OPINION

# No Moore's Law for batteries

**Fred Schlachter**[1]
*American Physical Society, Washington, DC 20045*

The public has become accustomed to rapid progress in mobile phone technology, computers, and access to information; tablet computers, smart phones, and other powerful new devices are familiar to most people on the planet.

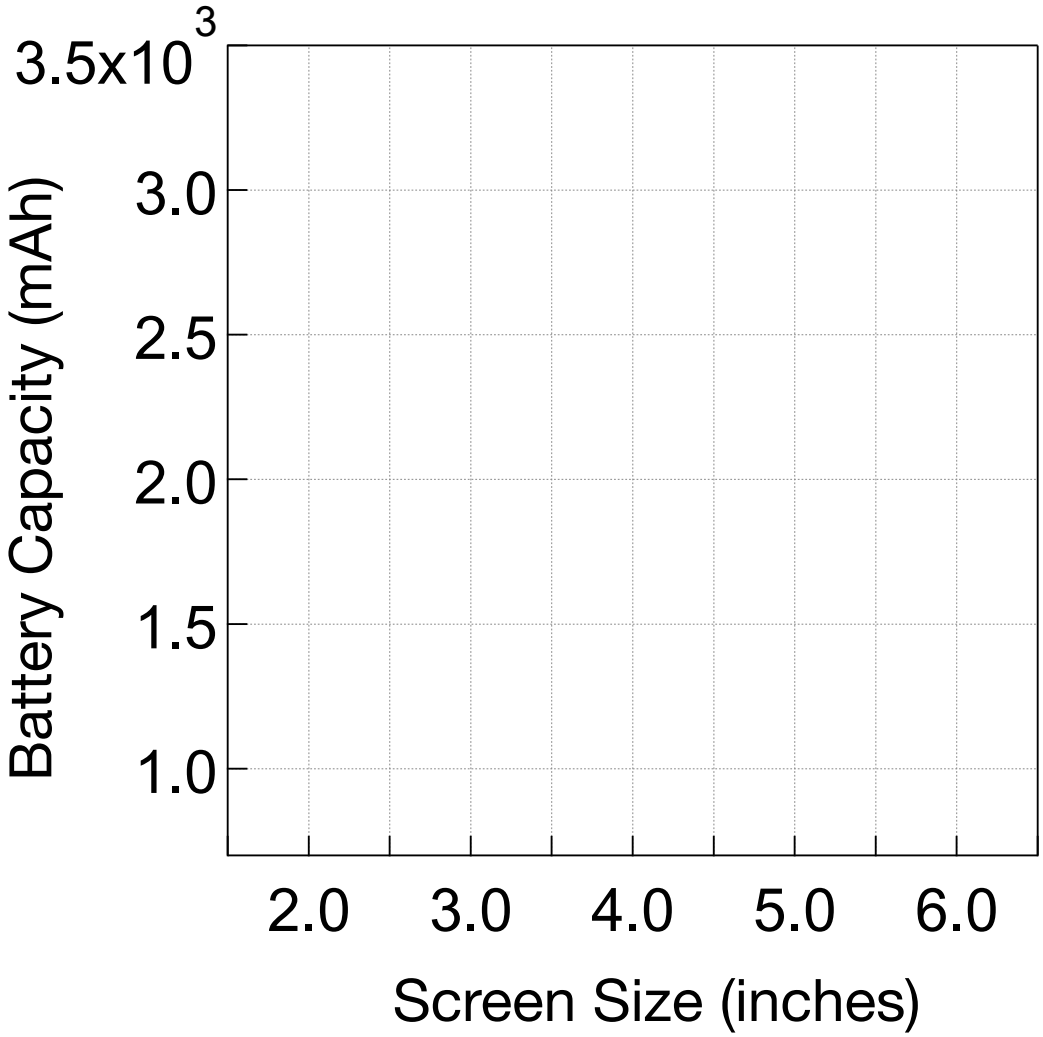
These developments are due in part to the ongoing exponential increase in computer processing power, doubling approximately every 2 years for the past several decades. This pattern is usually called Moore's Law and is named for Gordon Moore, a cofounder of Intel. The law is not a law like that for gravity; it is an empirical observation, which has become a self-fulfilling prophecy. Unfortunately, much of the public has come to expect that all technology does, will, or should follow such a law, which is not consistent with our everyday observations: For example, the maximum there is a Moore's Law for computer processors is that electrons are small and they do not take up space on a chip. Chip performance is limited by the lithography technology used to fabricate the chips; as lithography improves ever smaller features can be made on processors. Batteries are not like this. Ions, which transfer charge in batteries, are large, and they take up space, as do anodes, cathodes, and electrolytes. A D-cell battery stores more energy than an AA-cell. Potentials in a battery are dictated by the relevant chemical reactions, thus limiting eventual battery performance. Significant improvement in battery capacity can only be made by changing to a different chemistry.

Scientists and battery experts, who have been optimistic in the recent past about improving lithium-ion batteries and about de-

# “Improving” Energy Capacity

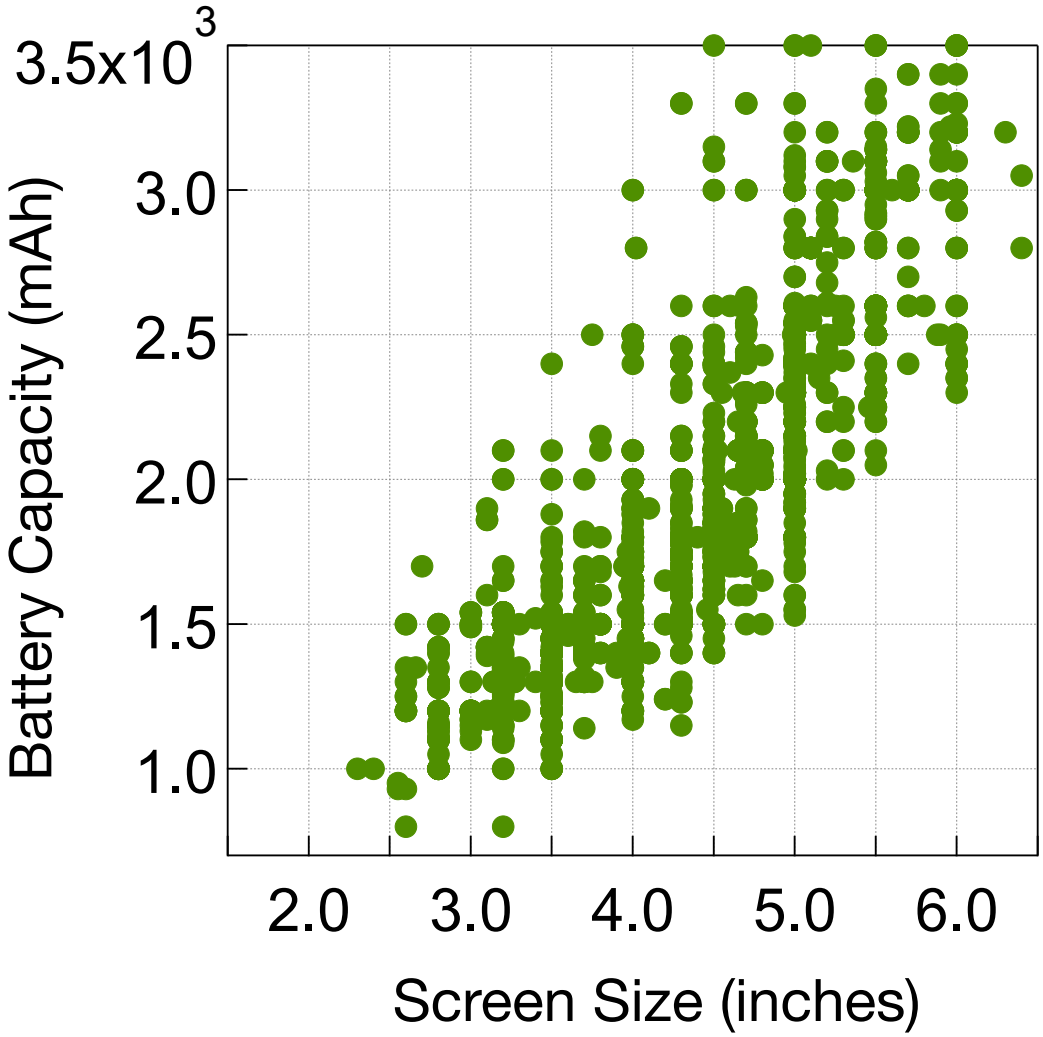
600 smartphone from 2006 to 2014 on http://www.gsmarena.com/makers.php3

# “Improving” Energy Capacity



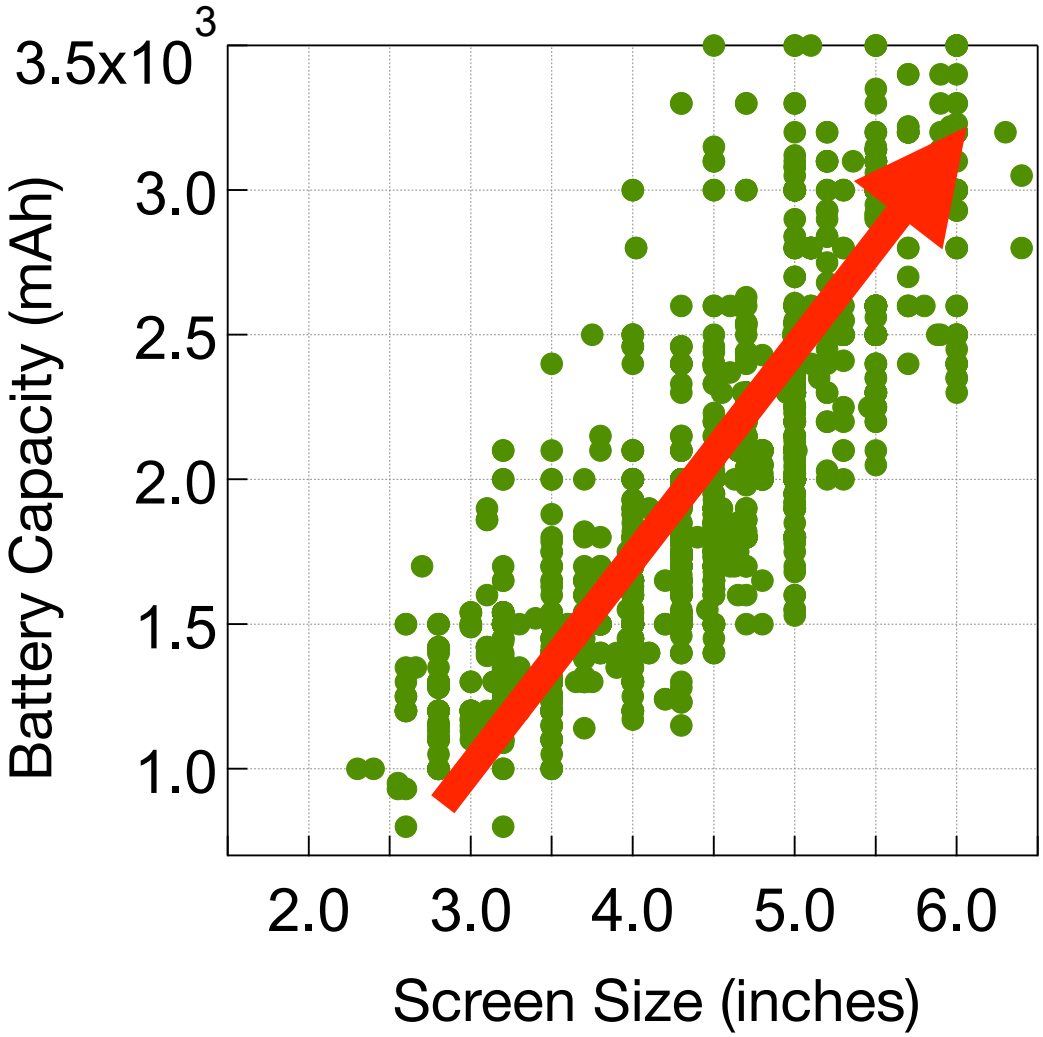

600 smartphone from 2006 to 2014 on http://www.gsmarena.com/makers.php3

# “Improving” Energy Capacity




600 smartphone from 2006 to 2014 on http://www.gsmarena.com/makers.php3

# “Improving” Energy Capacity




600 smartphone from 2006 to 2014 on http://www.gsmarena.com/makers.php3

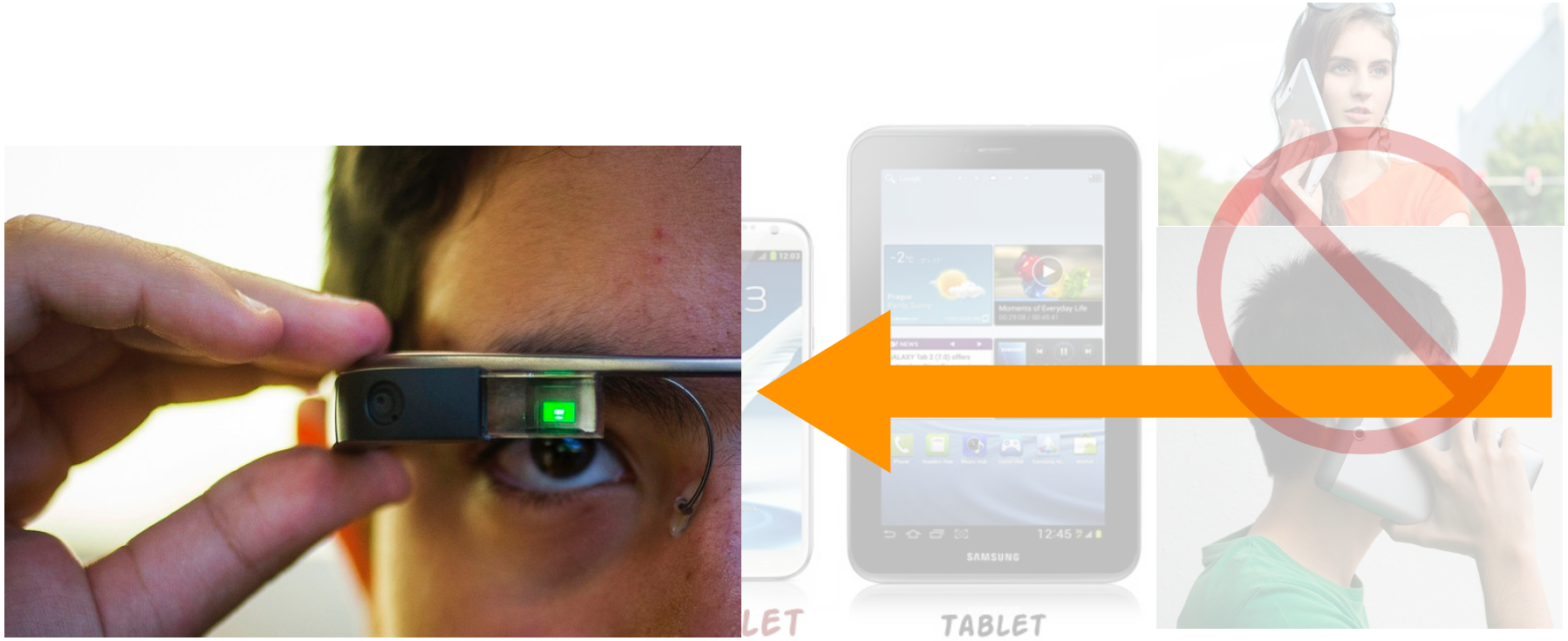
# “Improving” Energy Capacity

# “Improving” Energy Capacity

# “Improving” Energy Capacity

# “Improving” Energy Capacity

# “Improving” Energy Capacity



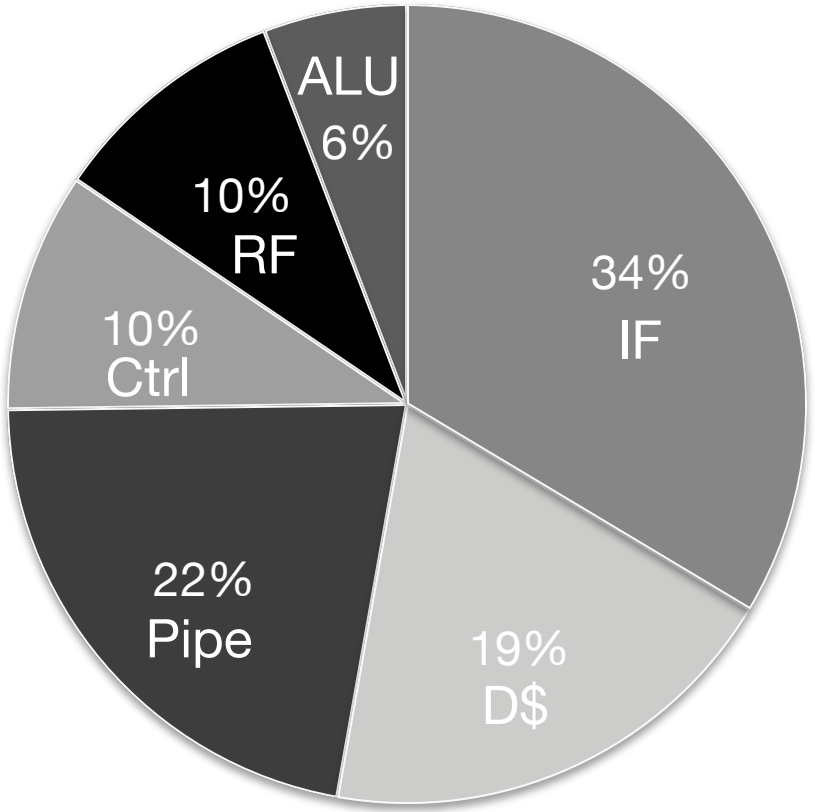
# Sources of Energy-Inefficiencies

General-Purpose CPU = Instruction Delivery + Data Feeding + Execution + Control, where instruction delivery, data feeding & control are pure overhead

Understanding sources of inefficiency in general-purpose chips, Hameed et al., ISCA 2010

# Sources of Energy-Inefficiencies

General-Purpose CPU = Instruction Delivery + Data Feeding + Execution + Control, where instruction delivery, data feeding & control are pure overhead




**IF**: Instruction fetch

**Ctrl**: Other control logics

**Pipe**: Pipeline reg, bus, clock

**D$**: Data cache

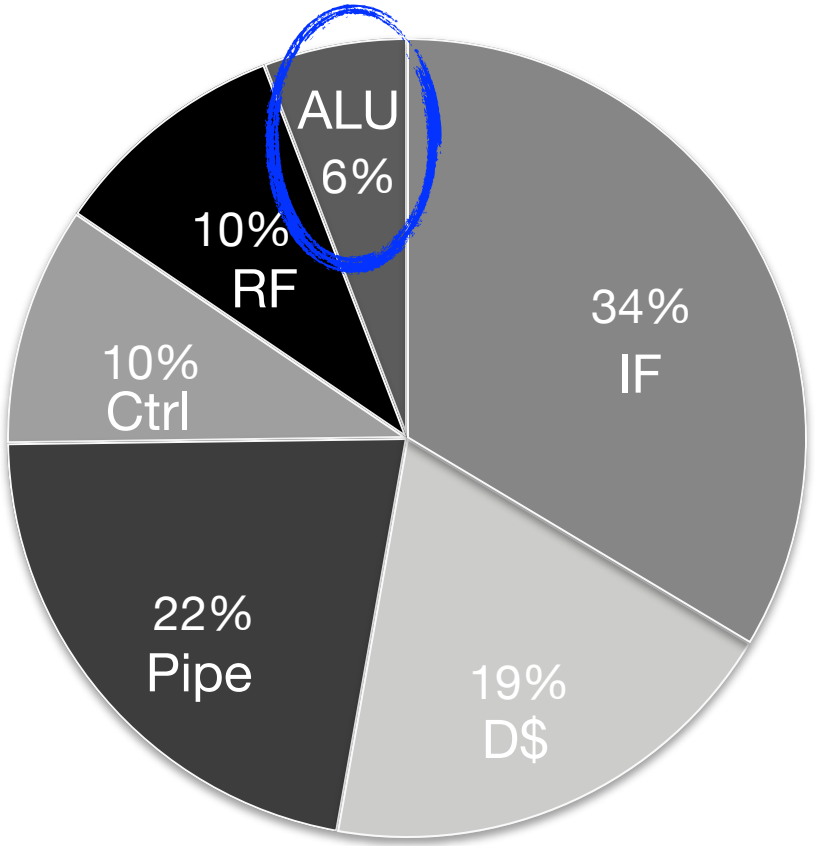
**RF**: Register file

**ALU**: Functional units

Understanding sources of inefficiency in general-purpose chips, Hameed et al., ISCA 2010

# Sources of Energy-Inefficiencies

General-Purpose CPU = Instruction Delivery + Data Feeding + Execution + Control, where instruction delivery, data feeding & control are pure overhead




**IF**: Instruction fetch

**Ctrl**: Other control logics

**Pipe**: Pipeline reg, bus, clock

**D$**: Data cache

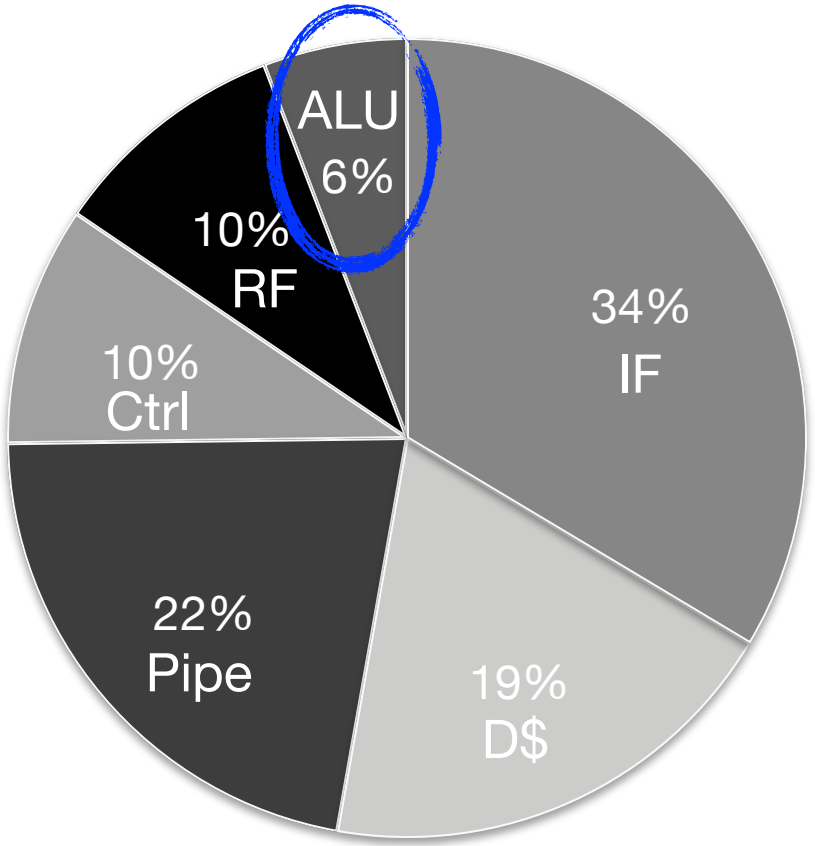
**RF**: Register file

**ALU**: Functional units

Understanding sources of inefficiency in general-purpose chips, Hameed et al., ISCA 2010

# Sources of Energy-Inefficiencies

General-Purpose CPU = Instruction Delivery + Data Feeding + Execution + Control, where instruction delivery, data feeding & control are pure overhead




**Pure Overhead**

**IF**: Instruction fetch

**Ctrl**: Other control logics

**Pipe**: Pipeline reg, bus, clock

**D$**: Data cache

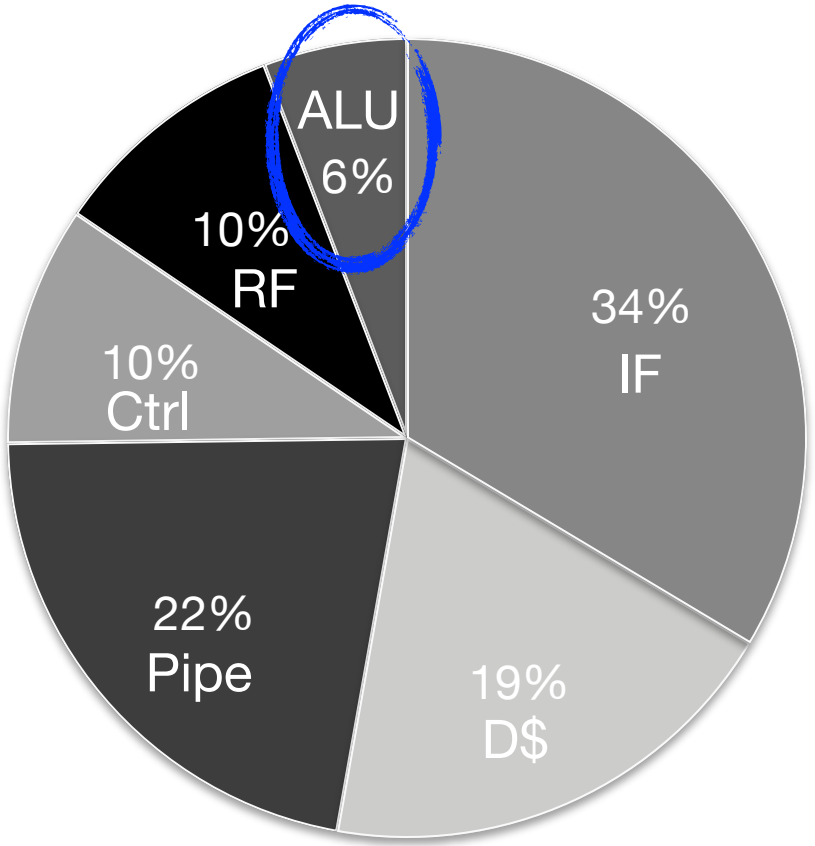
**RF**: Register file

**ALU**: Functional units

Understanding sources of inefficiency in general-purpose chips, Hameed et al., ISCA 2010

# Sources of Energy-Inefficiencies

General-Purpose CPU = Instruction Delivery + Data Feeding + Execution + Control, where instruction delivery, data feeding & control are pure overhead




**Pure Overhead**

**IF**: Instruction fetch

**Ctrl**: Other control logics

**Pipe**: Pipeline reg, bus, clock

**D$**: Data cache

**RF**: Register file

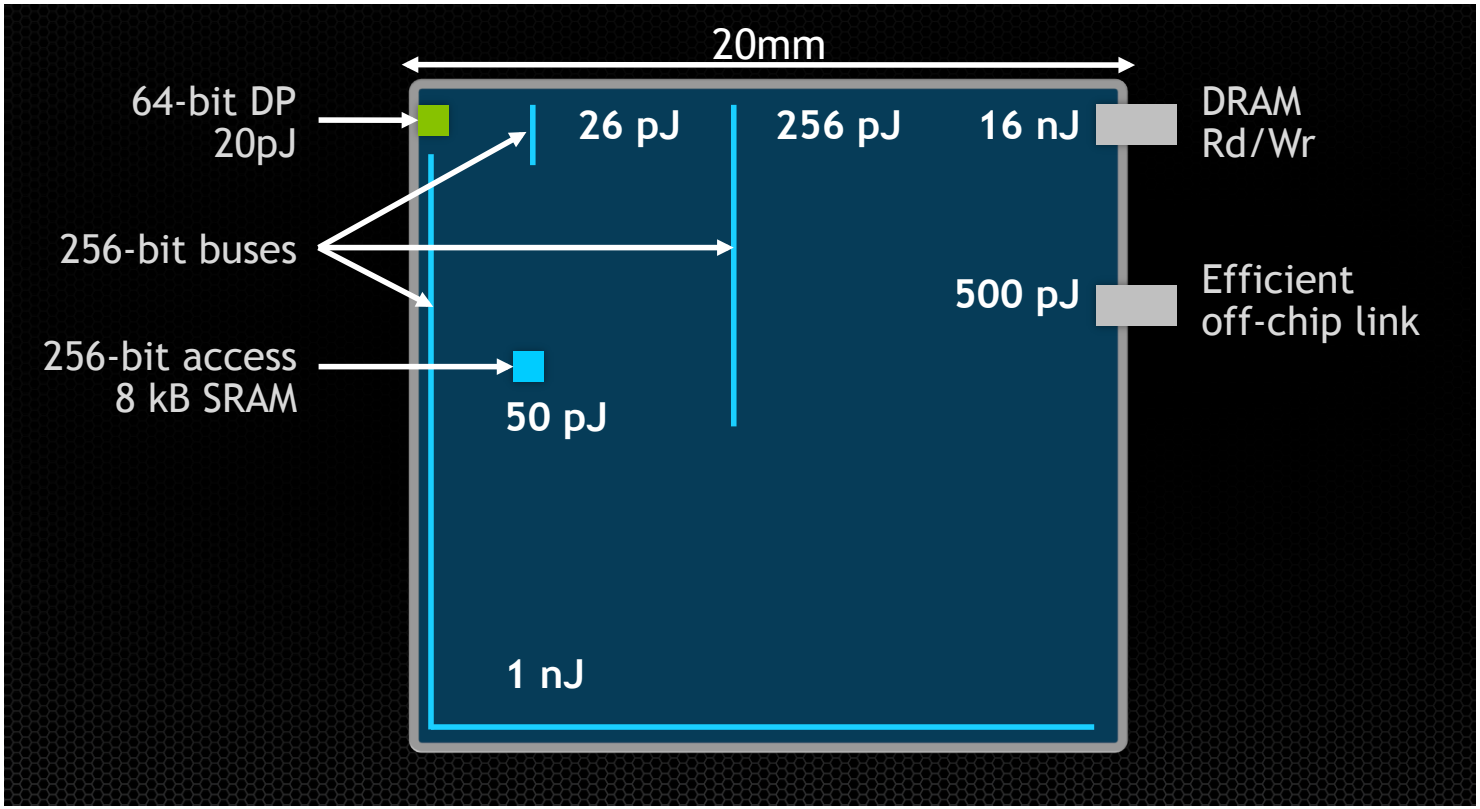
**ALU**: Functional units

**Doing Actual Work**

Understanding sources of inefficiency in general-purpose chips, Hameed et al., ISCA 2010

# Computation vs. Data Movement

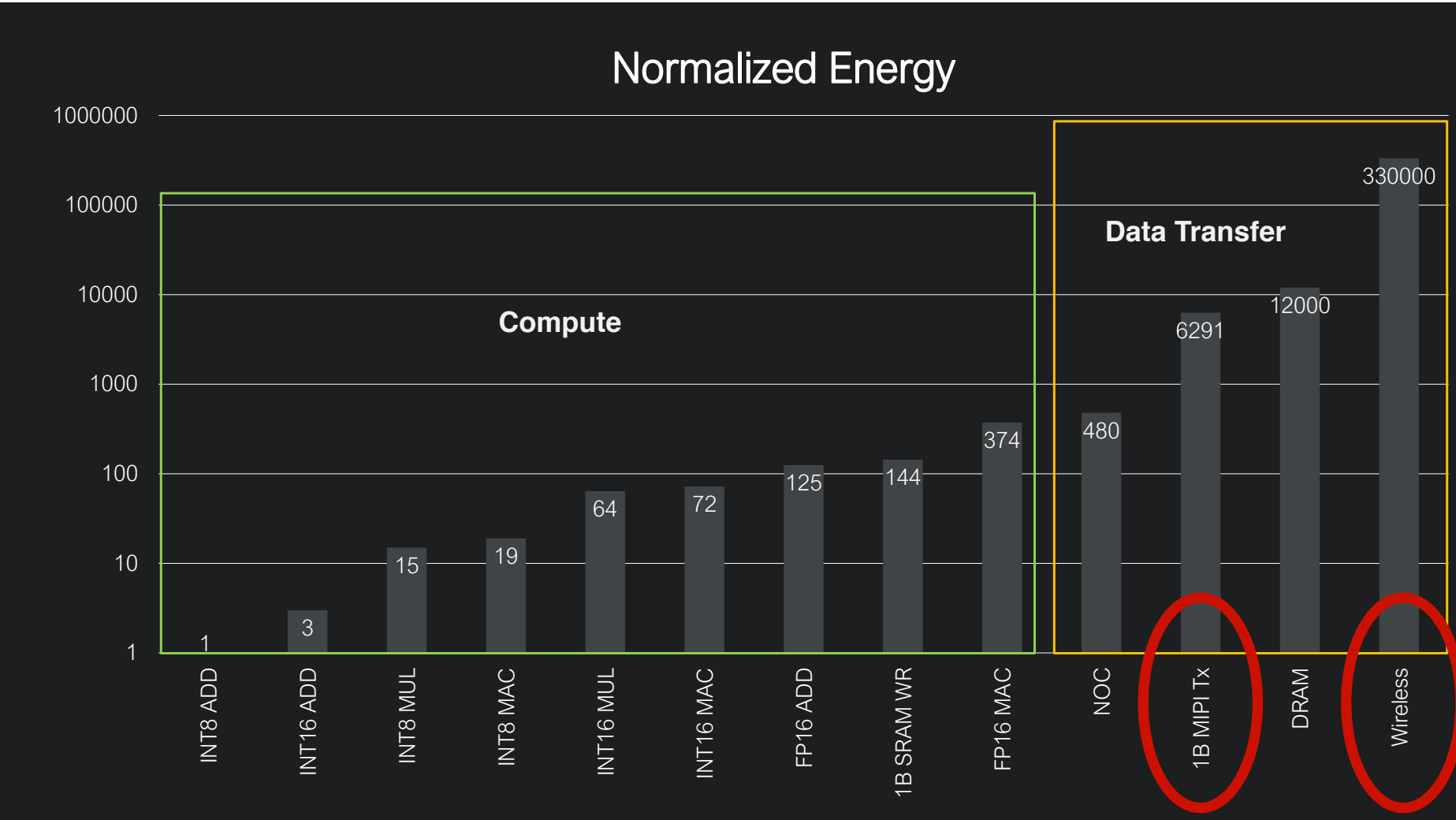
Data movement energy >> computation energy




Challenges for future computing systems, Bill Dally, 2015

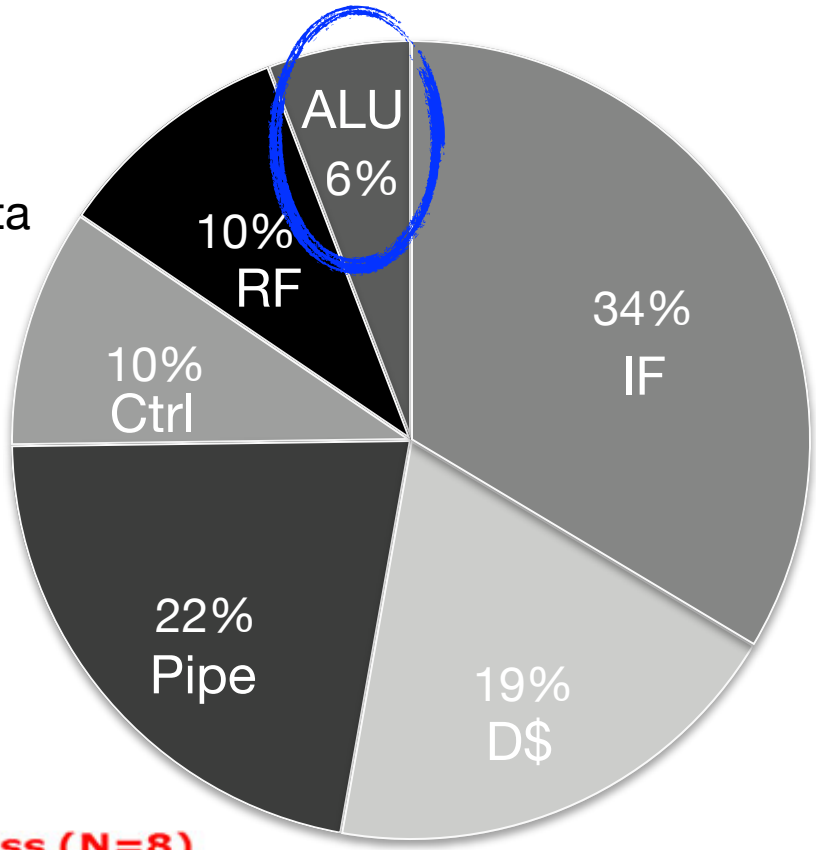
# Computation vs. Data Movement

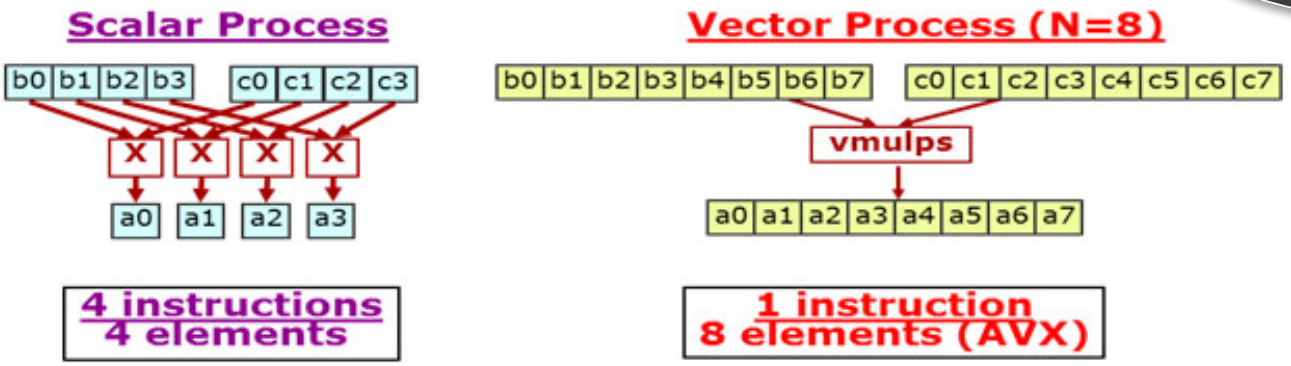
Data movement energy >> computation energy



# SIMD

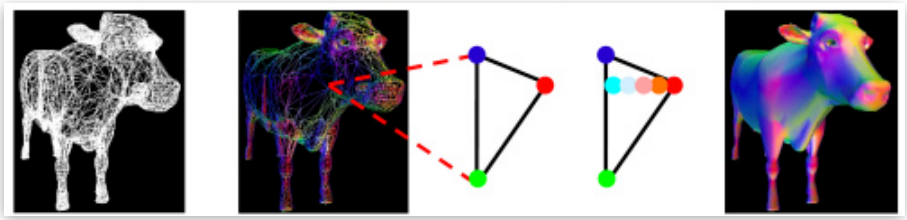
- Single Instruction (operating on) Multiple Data
- Amortizing the cost of instruction delivery/ control across many execution units (even cores).
- Almost all modern ISAs provide such instructions:
  - x86: MMX/SSE/AVX
  - Arm: Neon



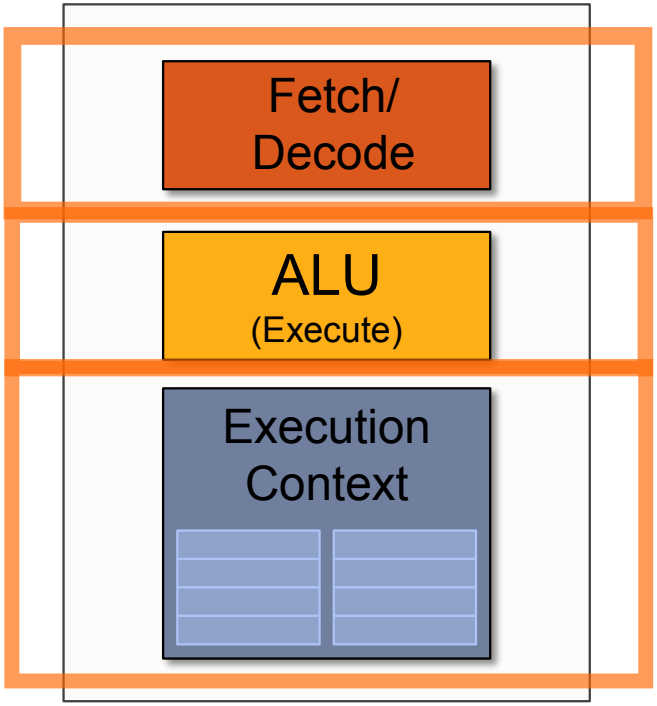



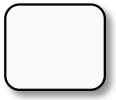
# Graphics Processing Units/GPUs (SIMT)

- Designed for graphics rendering, which is massively parallel.



# Execute shader



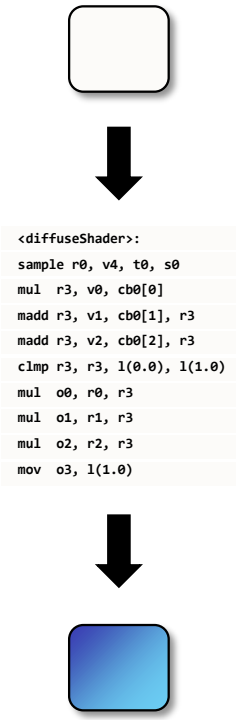



```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```

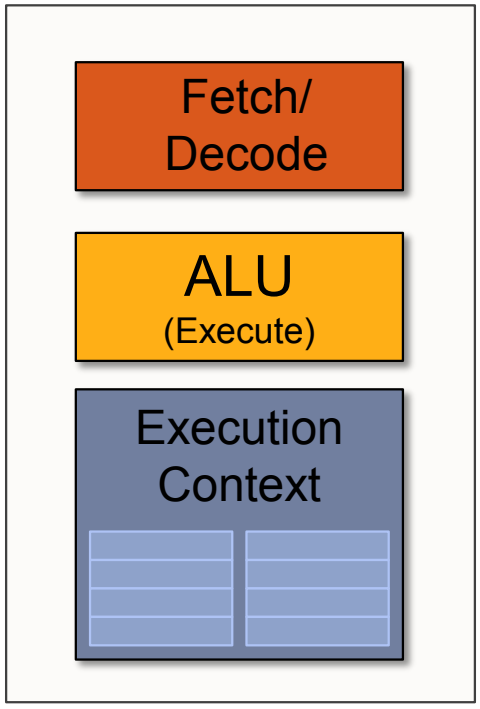


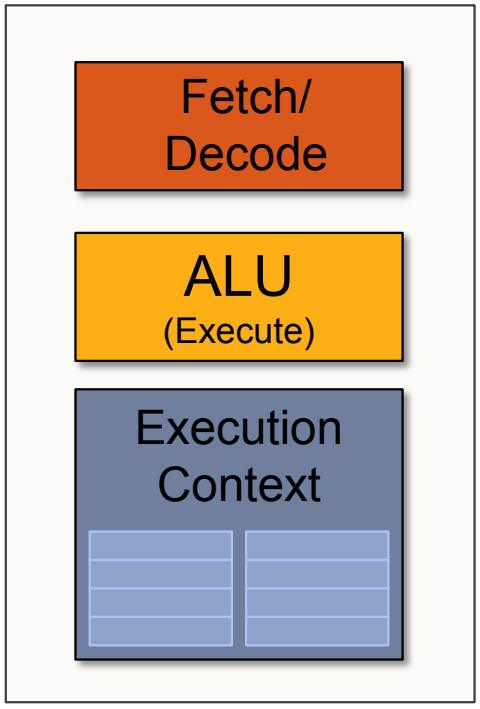
# Two cores (two fragments in parallel)

fragment 1

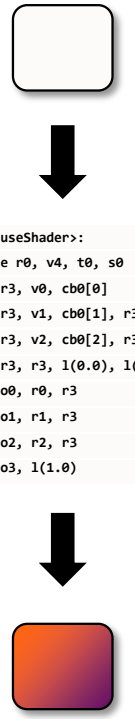


```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```



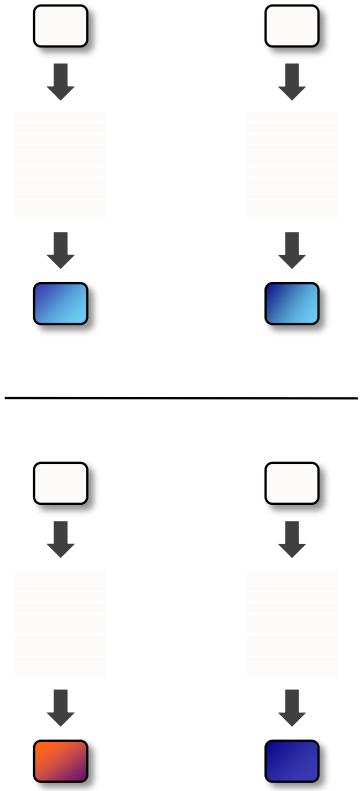

fragment 2

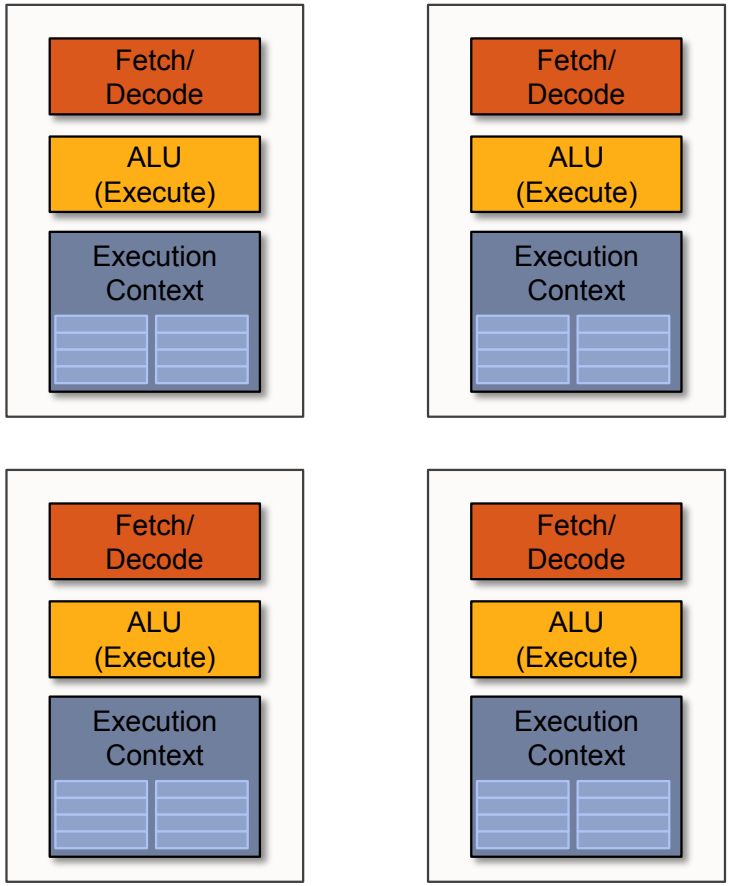


```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```



# Four cores (four fragments in parallel)



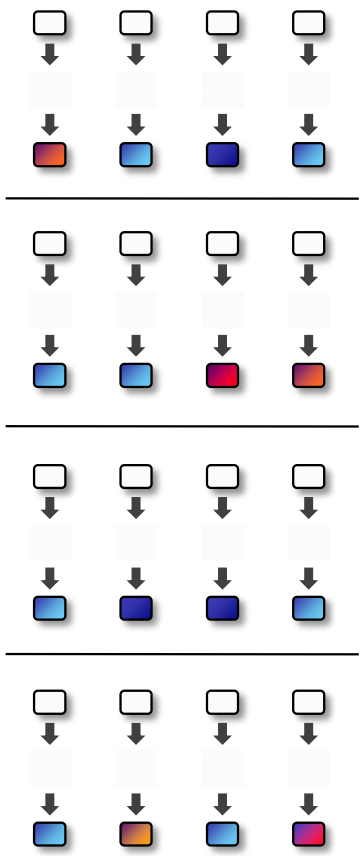





# Sixteen cores (sixteen fragments in parallel)


16 cores = 16 simultaneous instruction streams

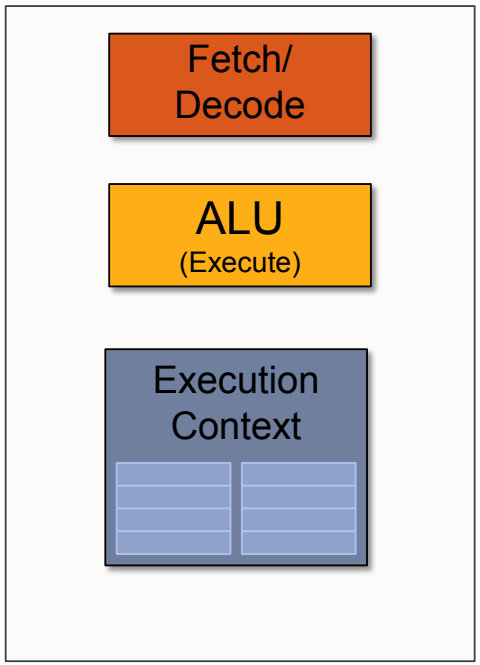
# Instruction stream coherence

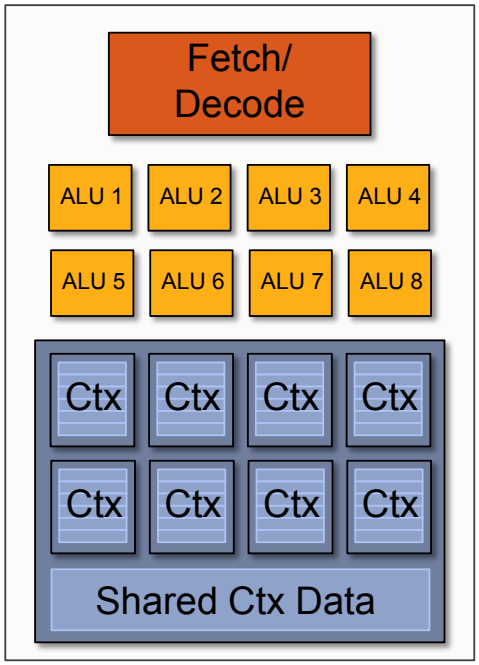


But… many fragments should be able to share an instruction stream!

```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```



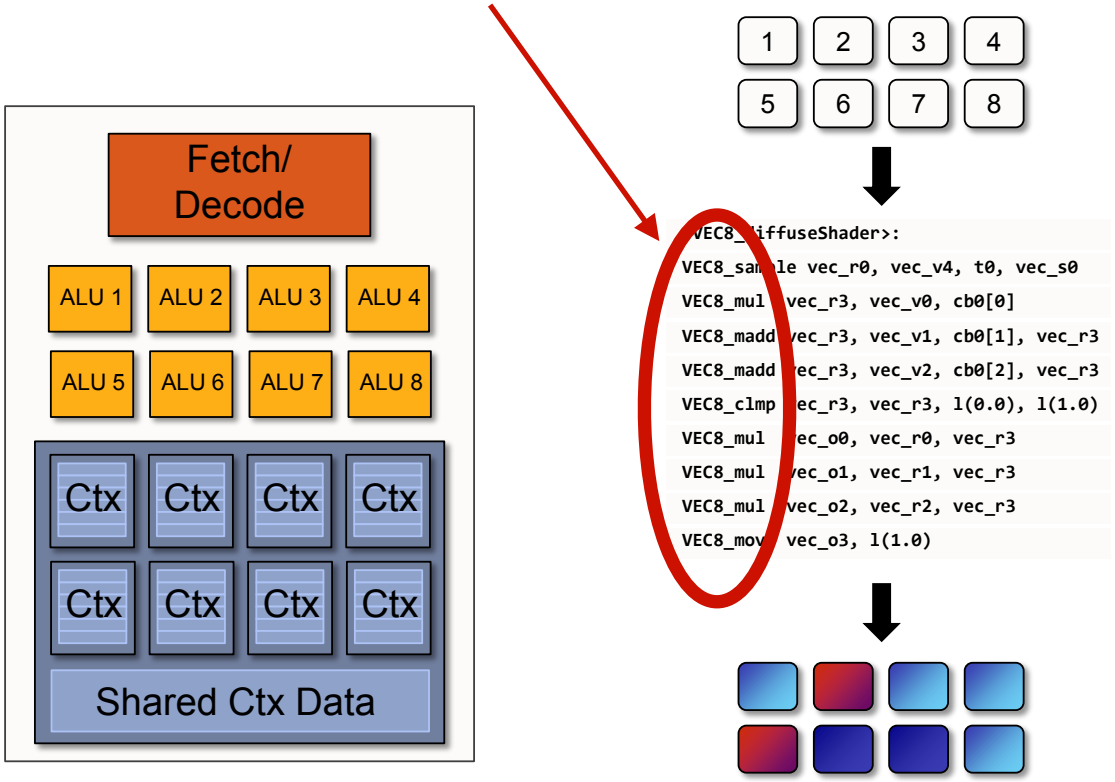


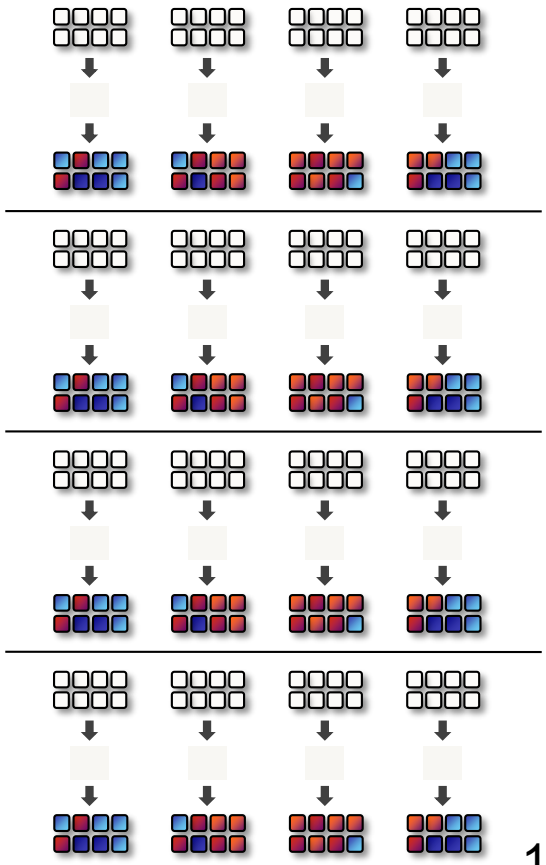

Amortize cost/complexity of managing an instruction stream across many ALUs

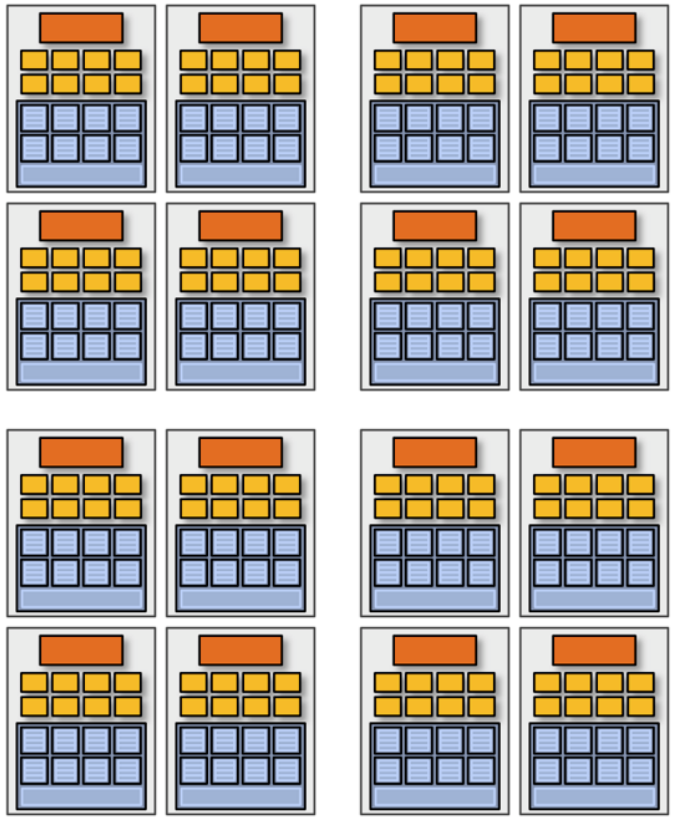
# SIMD processing

SIMD/vector instructions, each operates on a vector of 8 elements here.



# 16 cores, each with 8 ALUs. Each core here runs the same program (fragment shader)

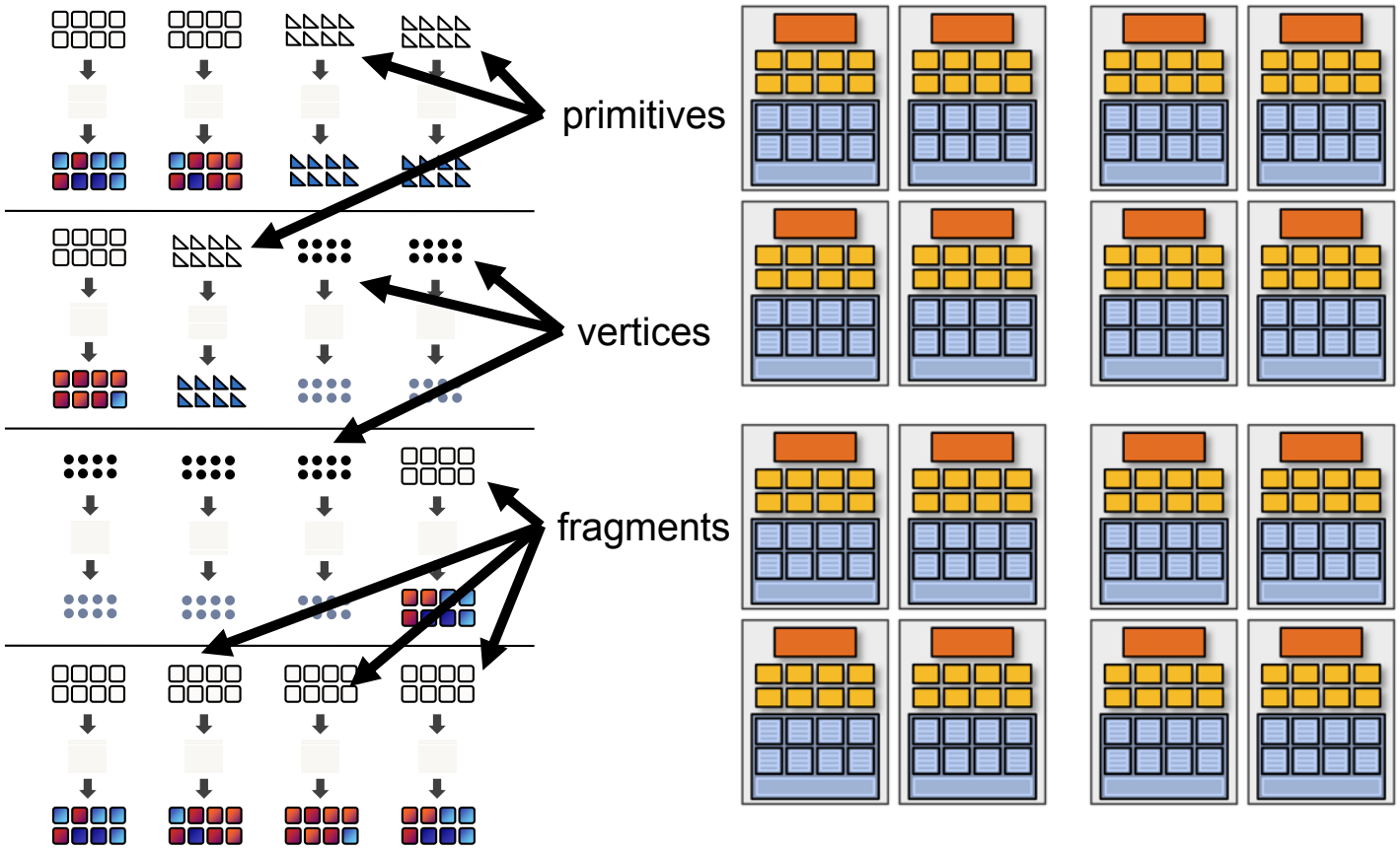




16 cores = 128 ALUs
= 16 simultaneous instruction streams

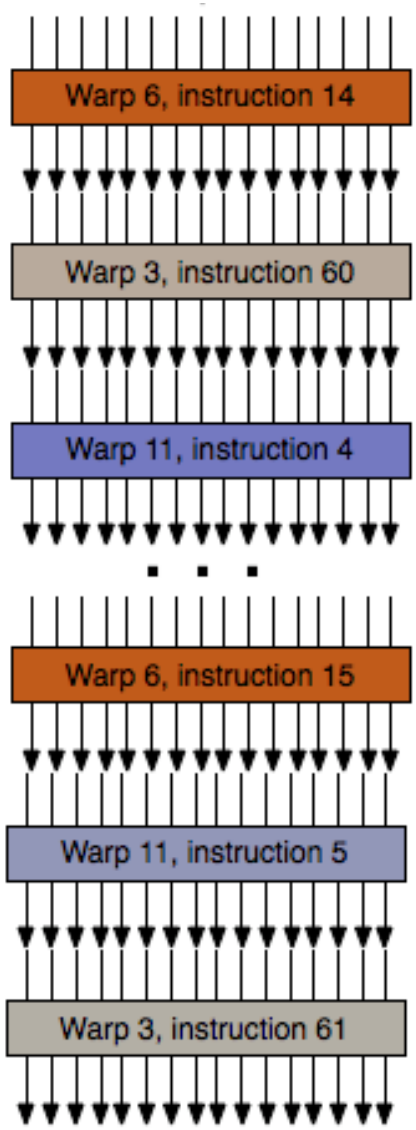


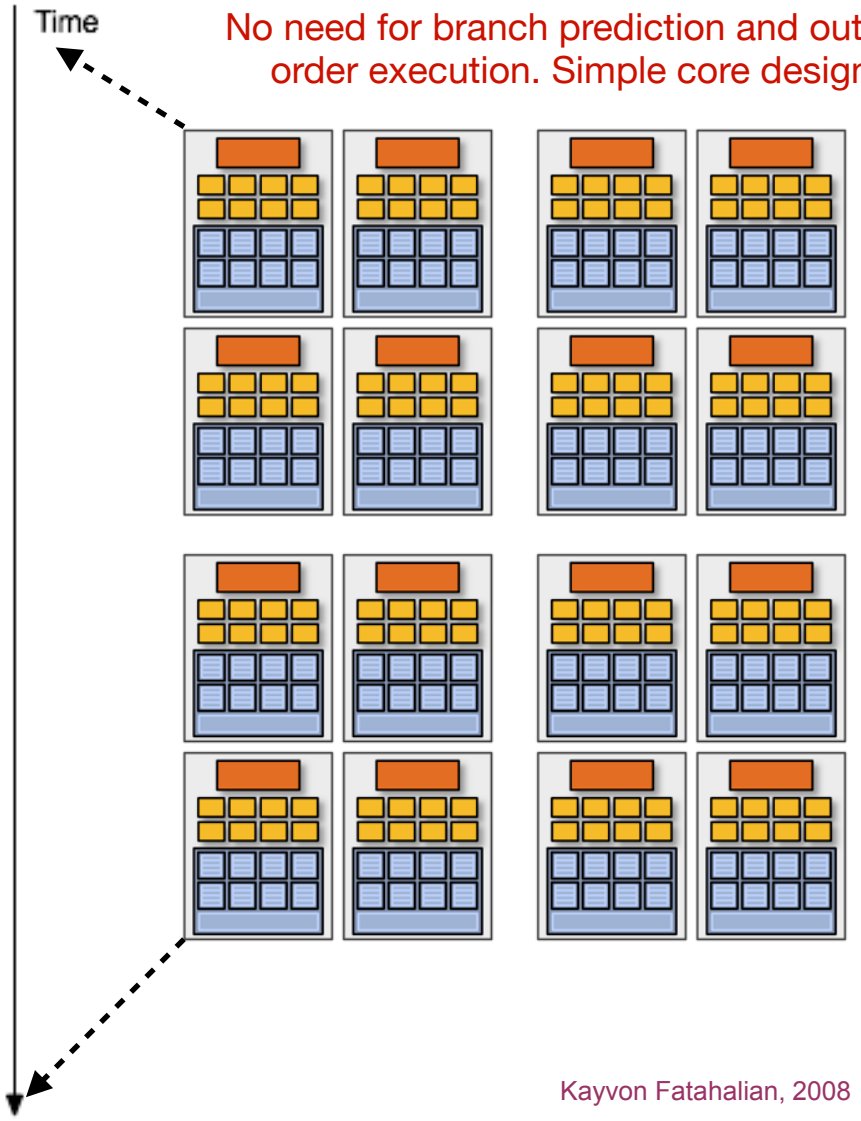
**16 cores, each with 8 ALUs. Cores here run different programs**
(some are processing vertices, some are processing fragments)



# Each Core Does Fine-Grained Multi-threading

Wrap: a group of threads (8 here)




No need for branch prediction and out-of-order execution. Simple core design.

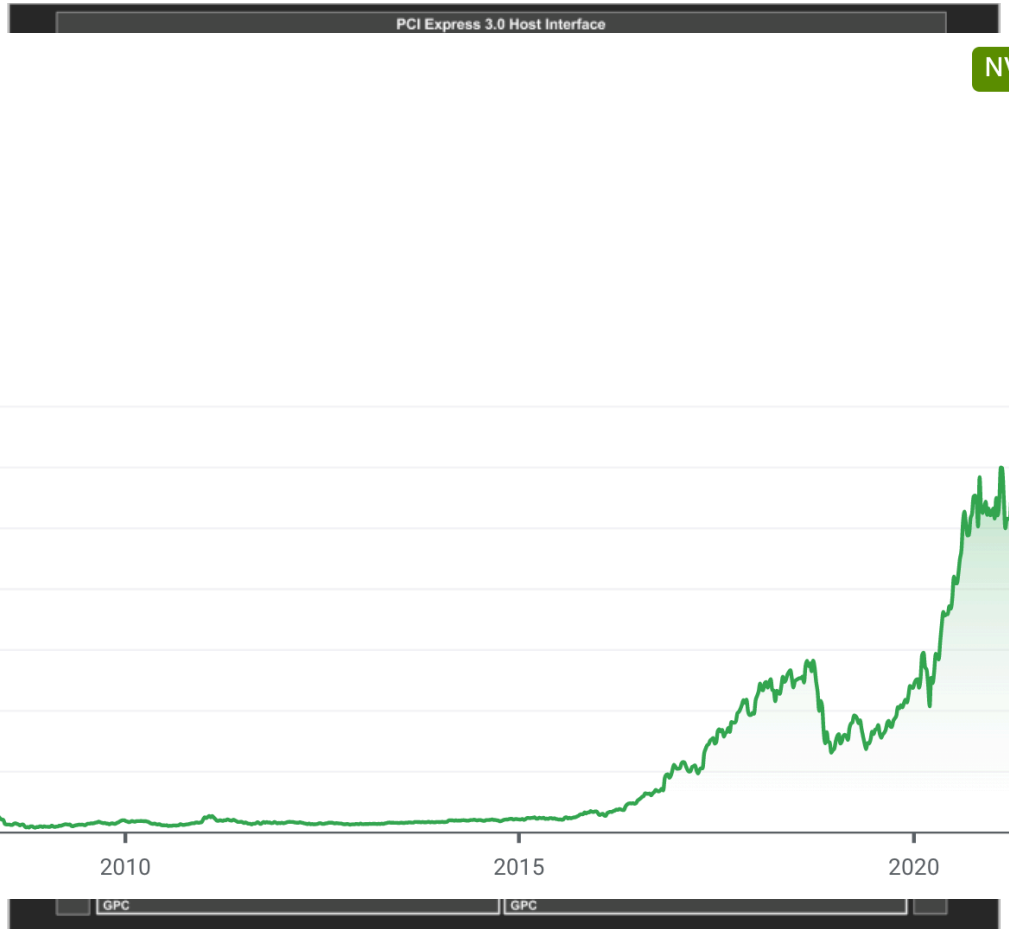
Kayvon Fatahalian, 2008

# Nvidia Maxwell GPU (2014)

- Today: General Purpose GPU (GPGPU), used for any massive parallel applications:
  - Physics simulation
  - Deep learning
  - Computer vision

# Nvidia Maxwell GPU (2014)

- Today: General Purpose GPU (GPGPU), used for any massive parallel applications:



# Entering the Era of Specialization

# Entering the Era of Specialization

- GPUs are very efficient for massively parallel program

# Entering the Era of Specialization

- GPUs are very efficient for massively parallel program
- But are still fairly general, so there are still many inefficiencies
  - Still need to fetch and decode instructions
  - Still have (very large) caches, so data delivery isn't efficient

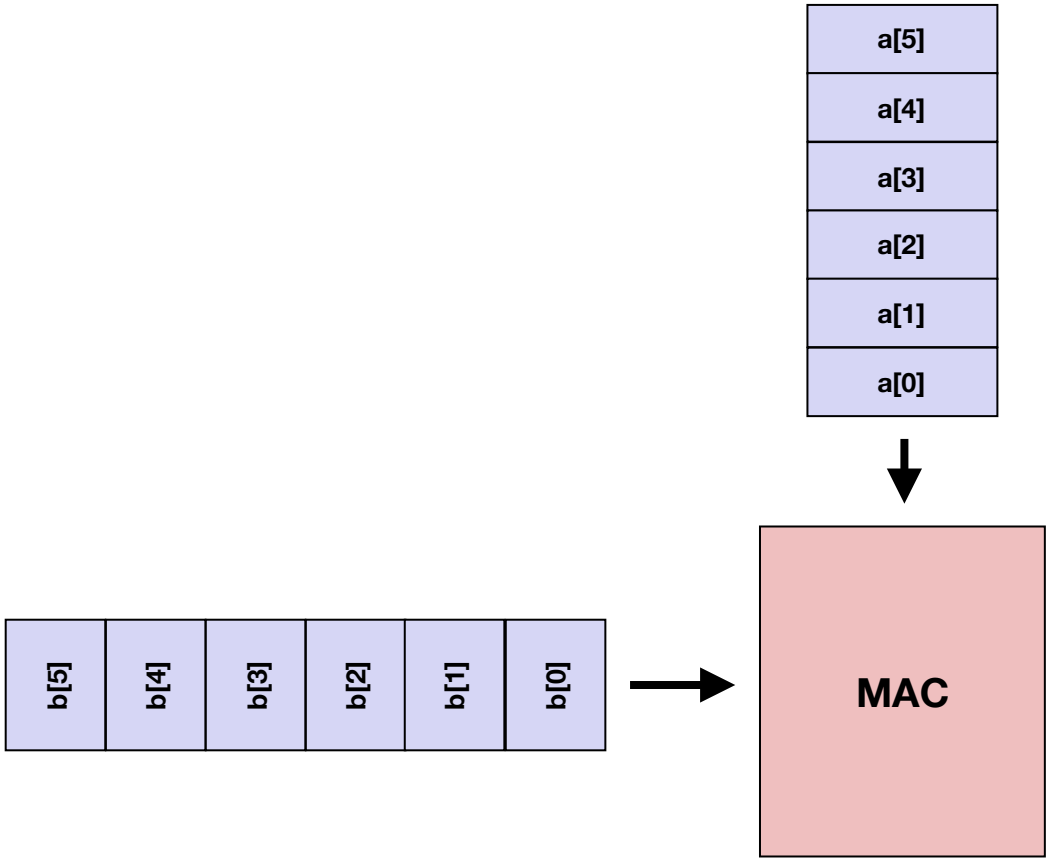
# Entering the Era of Specialization

- GPUs are very efficient for massively parallel program
- But are still fairly general, so there are still many inefficiencies
  - Still need to fetch and decode instructions
  - Still have (very large) caches, so data delivery isn't efficient
- Idea: instead of building general-purpose processors that can do everything, but inefficiently, let's build specialized processors that can only do limited things, but extremely efficiently.

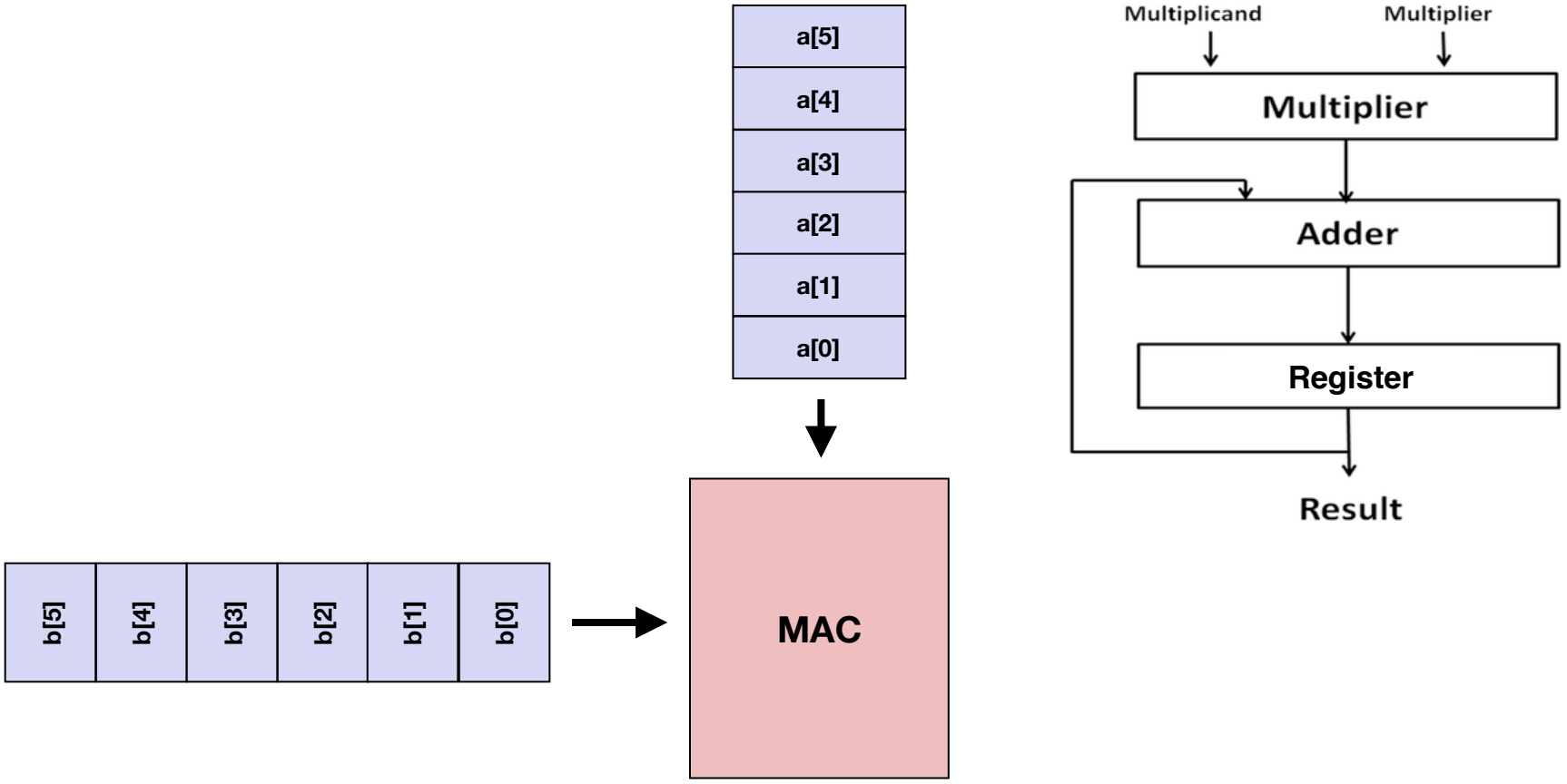
# Entering the Era of Specialization

- GPUs are very efficient for massively parallel program
- But are still fairly general, so there are still many inefficiencies
  - Still need to fetch and decode instructions
  - Still have (very large) caches, so data delivery isn't efficient
- Idea: instead of building general-purpose processors that can do everything, but inefficiently, let's build specialized processors that can only do limited things, but extremely efficiently.
- A.k.a., domain-specific accelerators

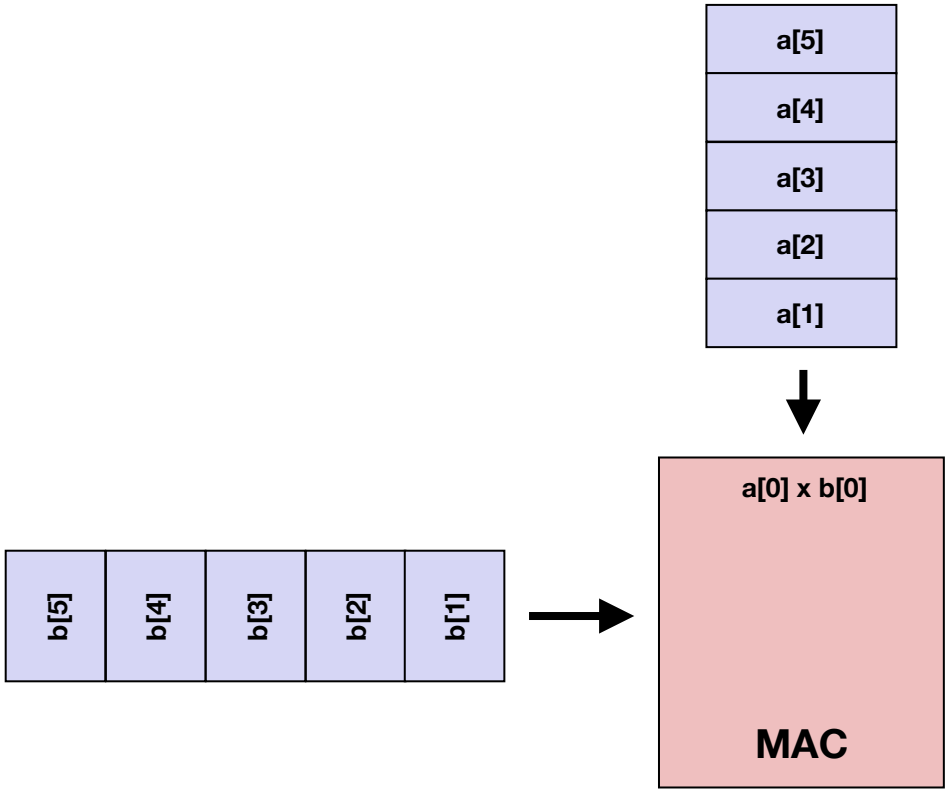
# Example: Vector Dot Product

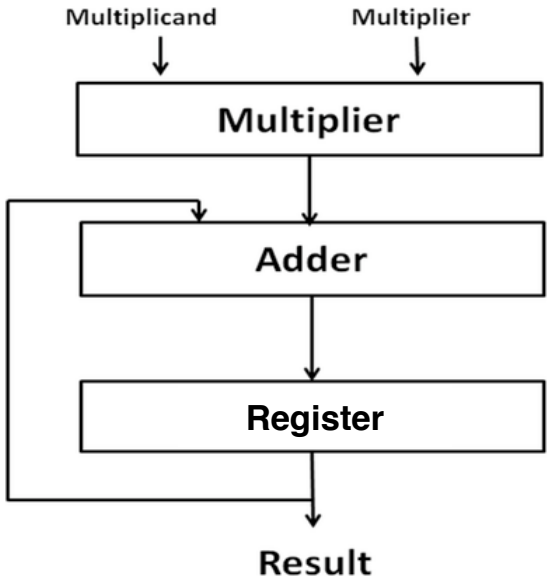


# Example: Vector Dot Product

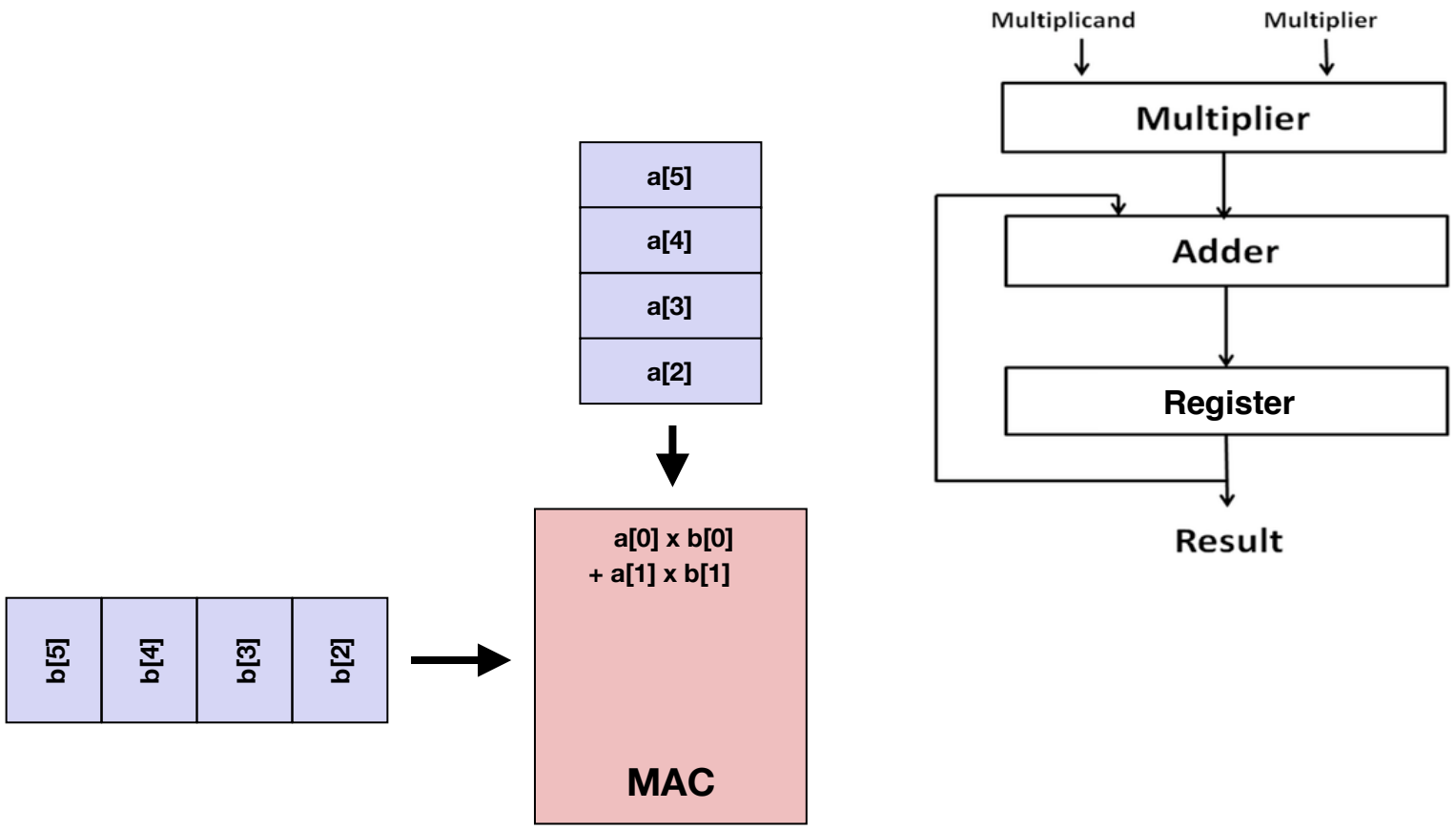


# Example: Vector Dot Product

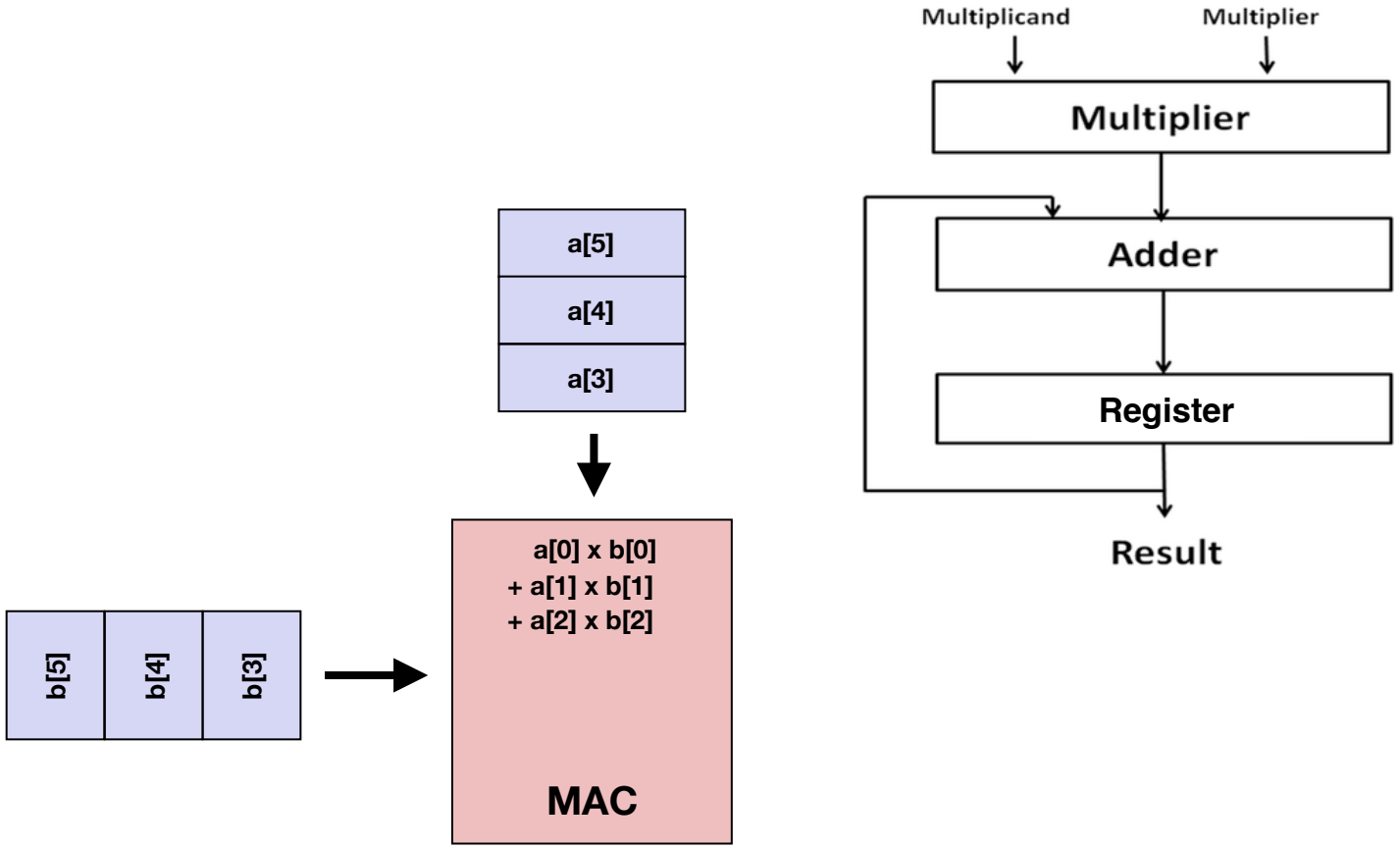


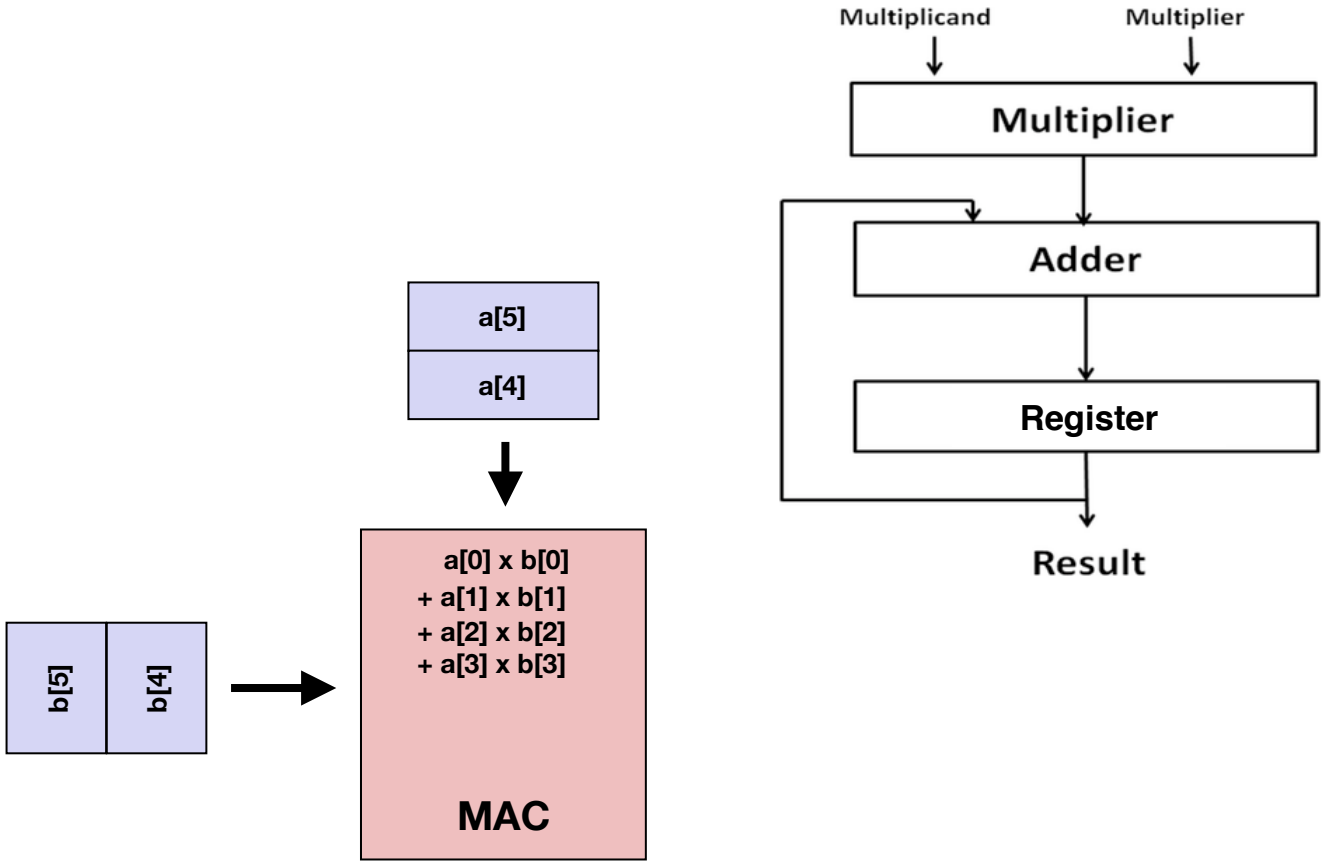
# Example: Vector Dot Product



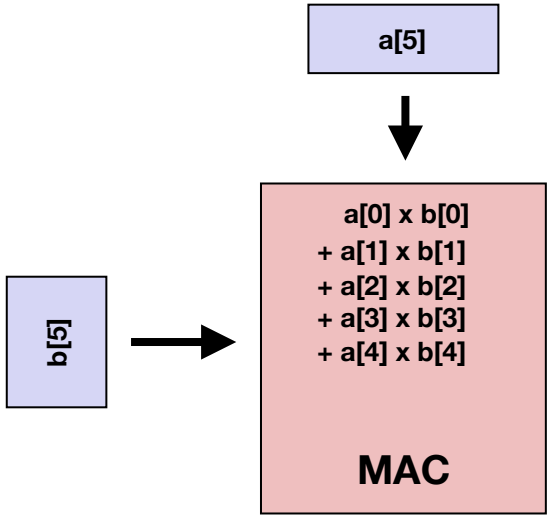
# Example: Vector Dot Product

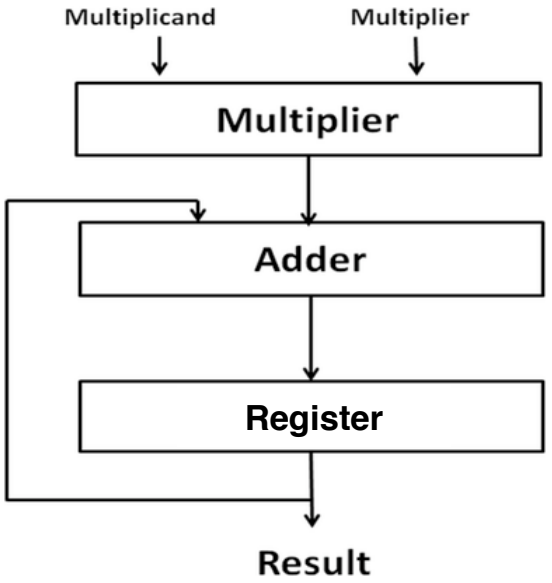


# Example: Vector Dot Product

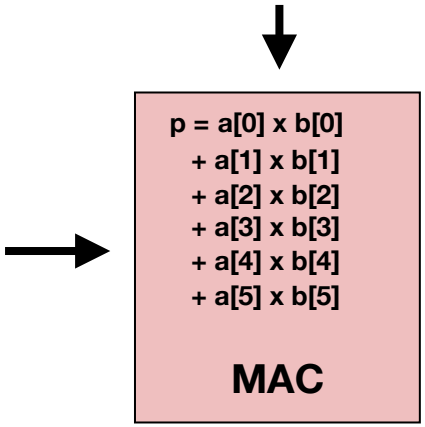


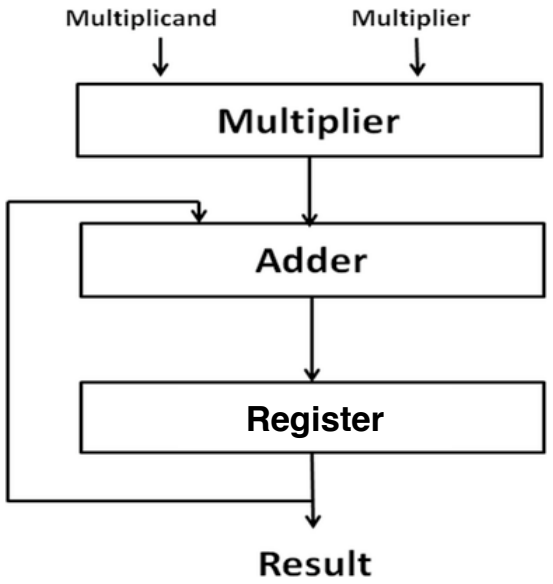
# Example: Vector Dot Product



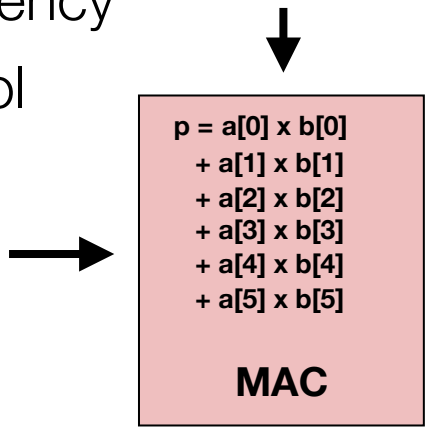
# Example: Vector Dot Product

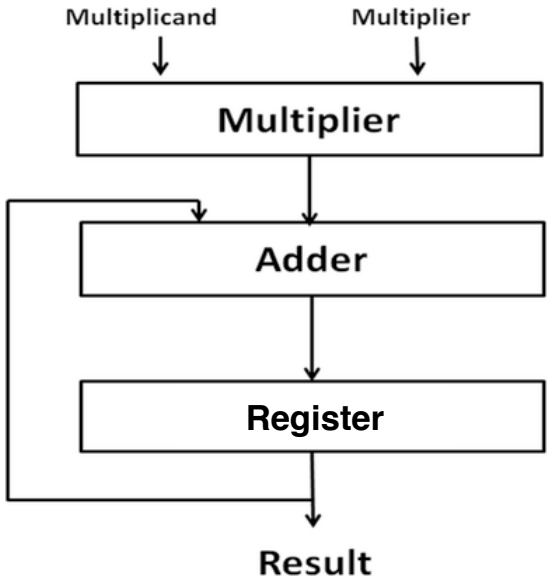


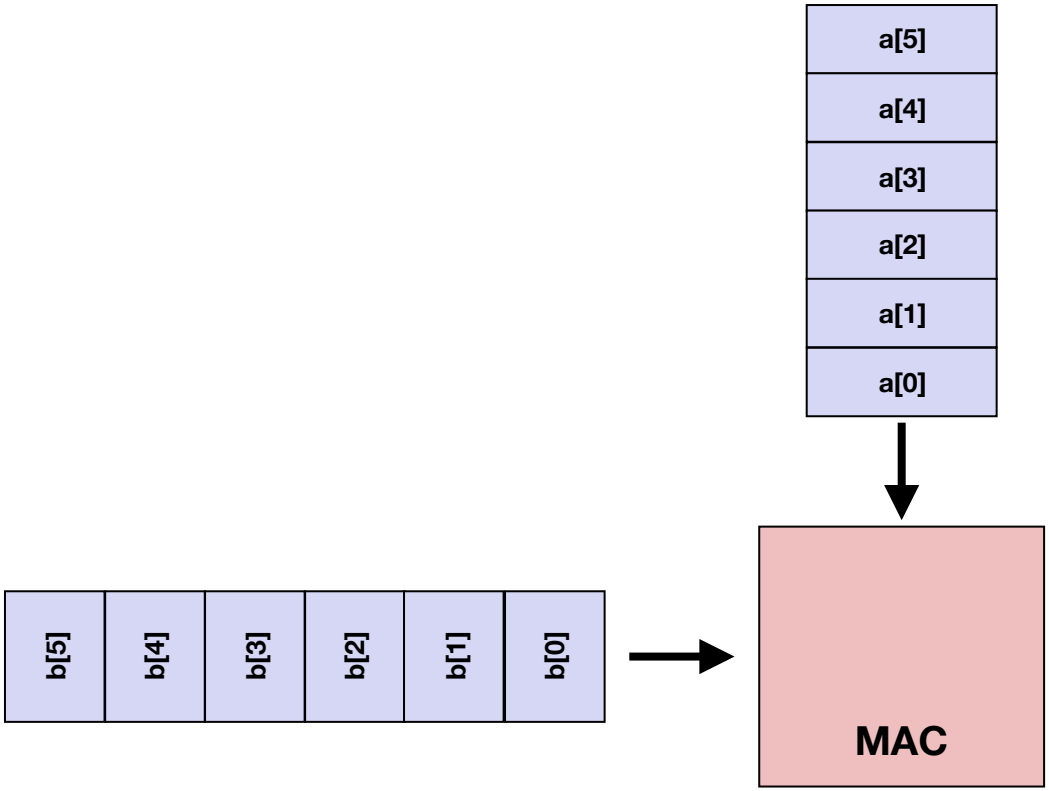
# Example: Vector Dot Product

- Does nothing but vector dot product
  - No instruction fetch and decode (there is no instruction)
  - The register is close to the ALU and gets reused over and over: good data delivery efficiency
  - Very simple control

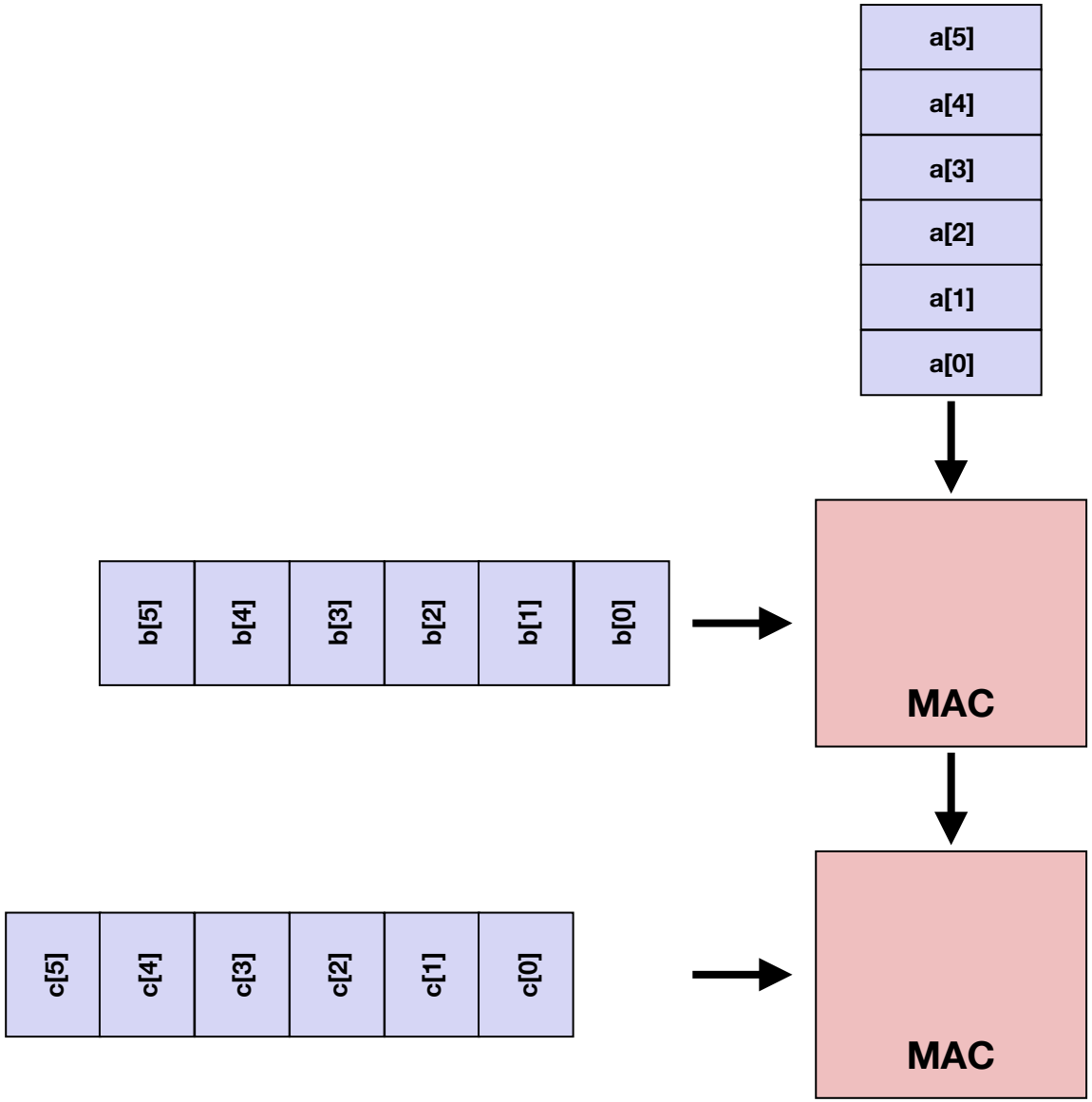


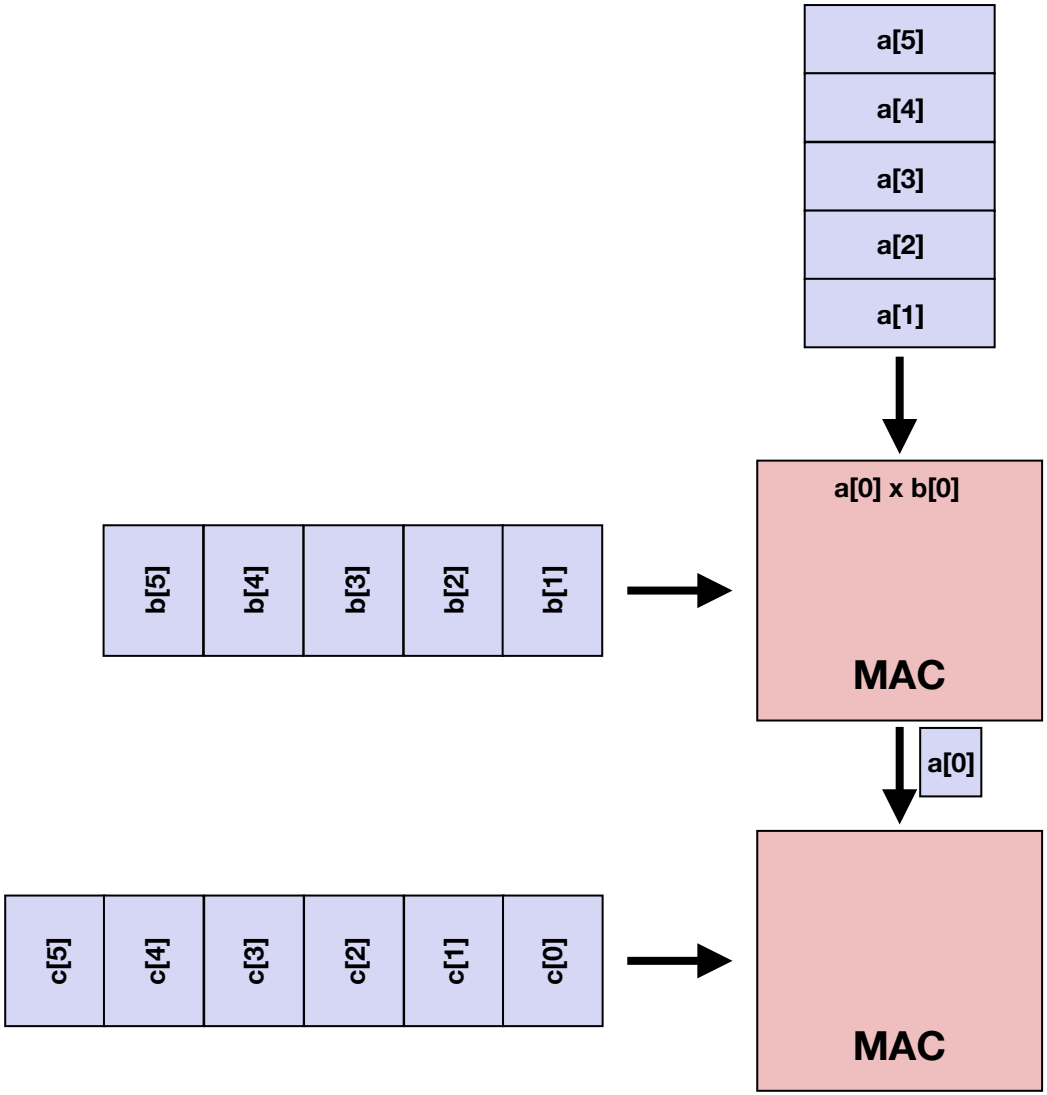
# Matrix Vector Multiplication



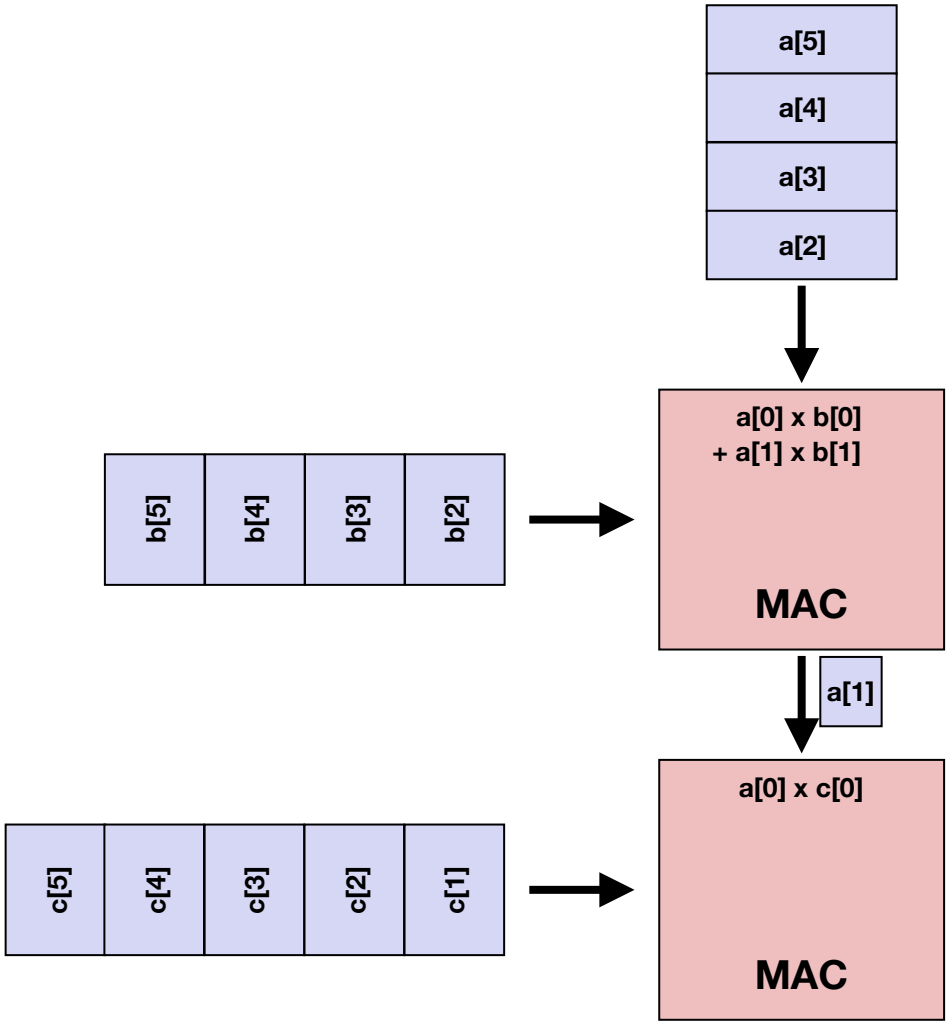
# Matrix Vector Multiplication



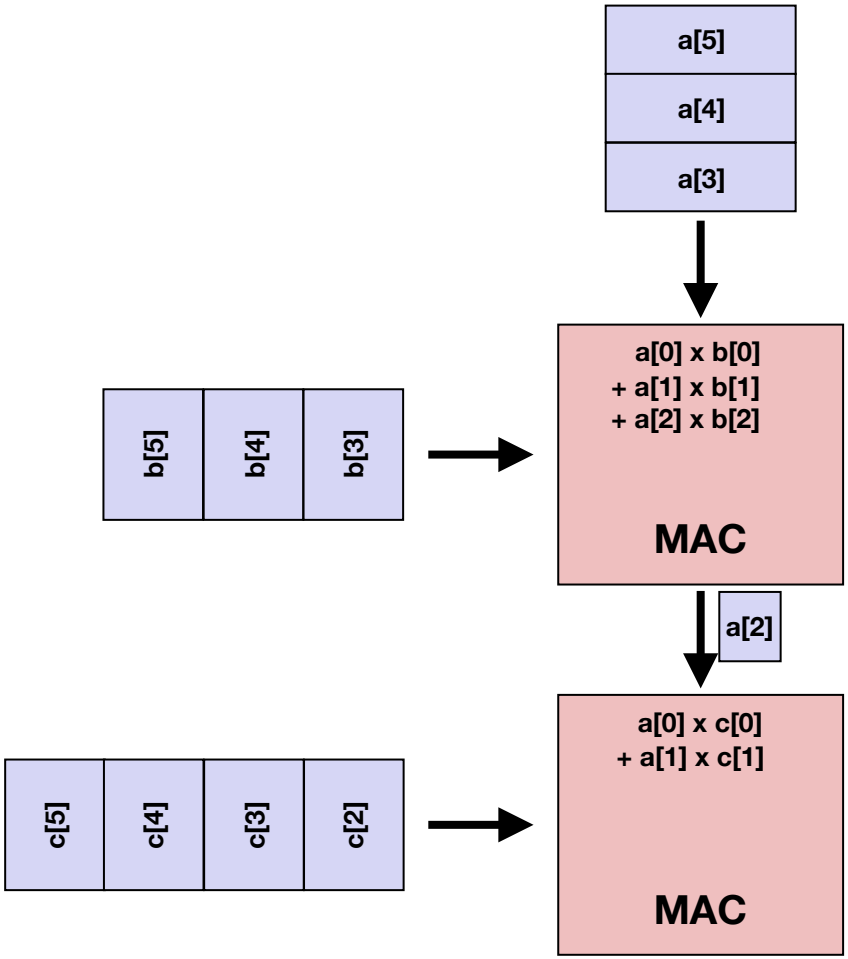
# Matrix Vector Multiplication



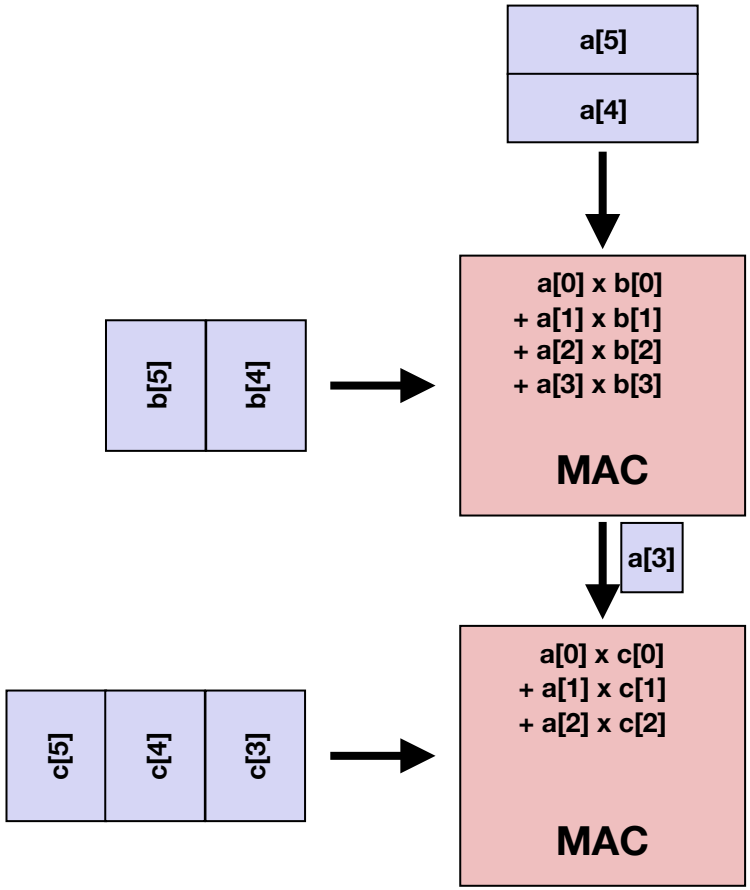
# Matrix Vector Multiplication



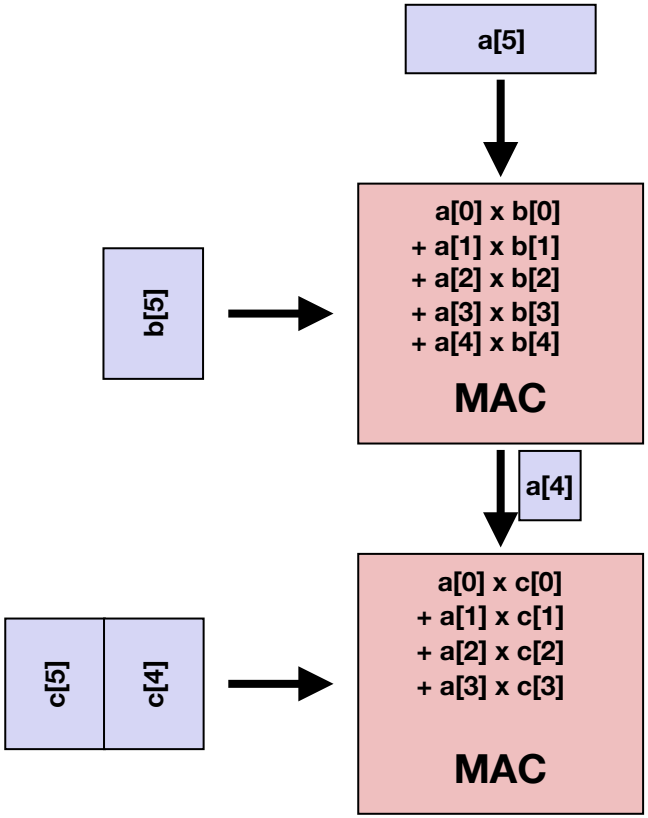
# Matrix Vector Multiplication



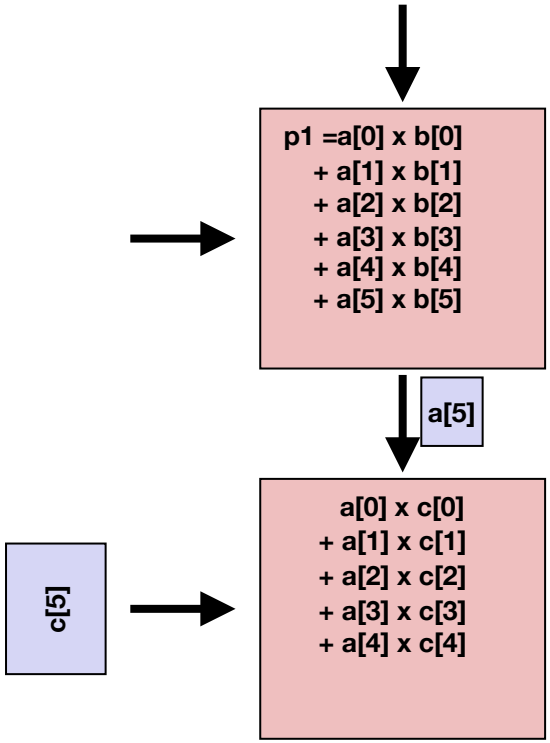
# Matrix Vector Multiplication



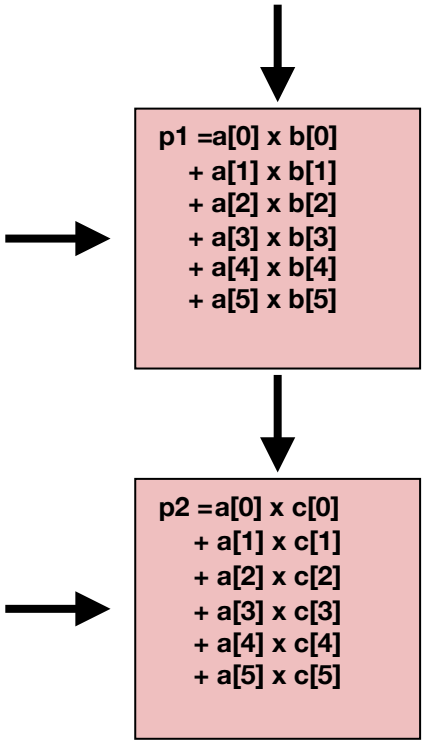
# Matrix Vector Multiplication



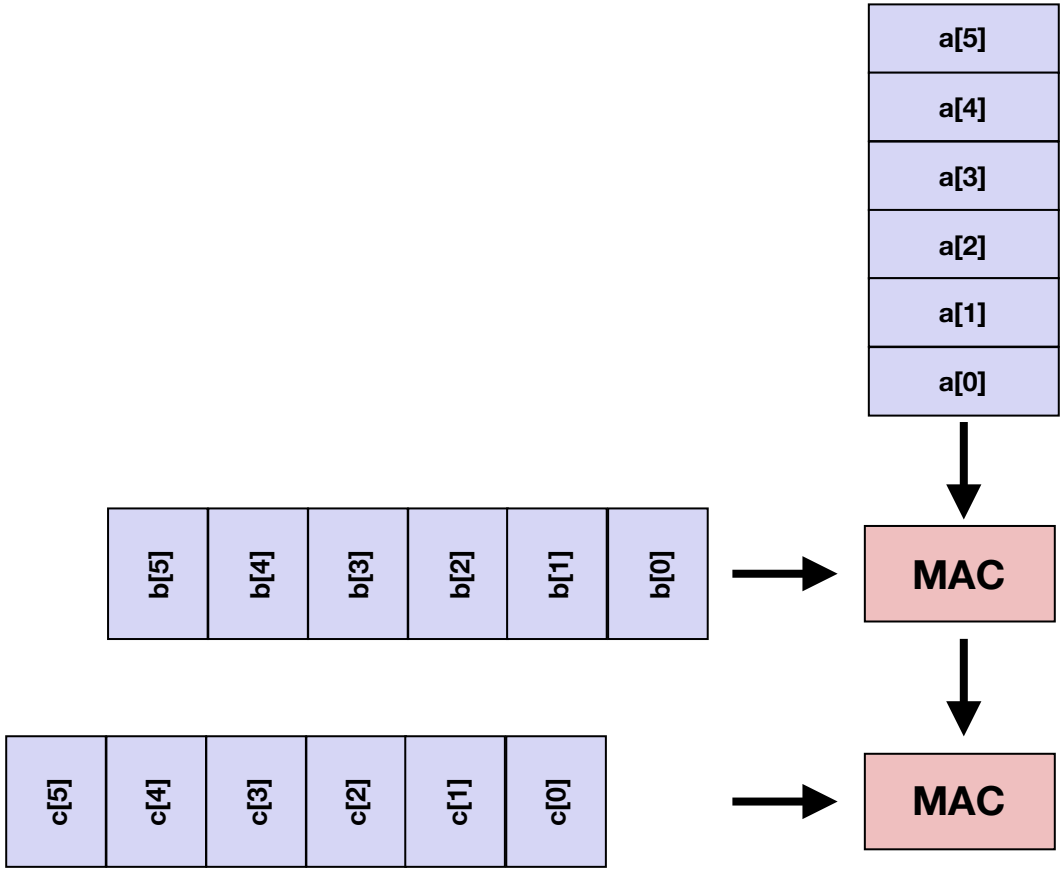
# Matrix Vector Multiplication



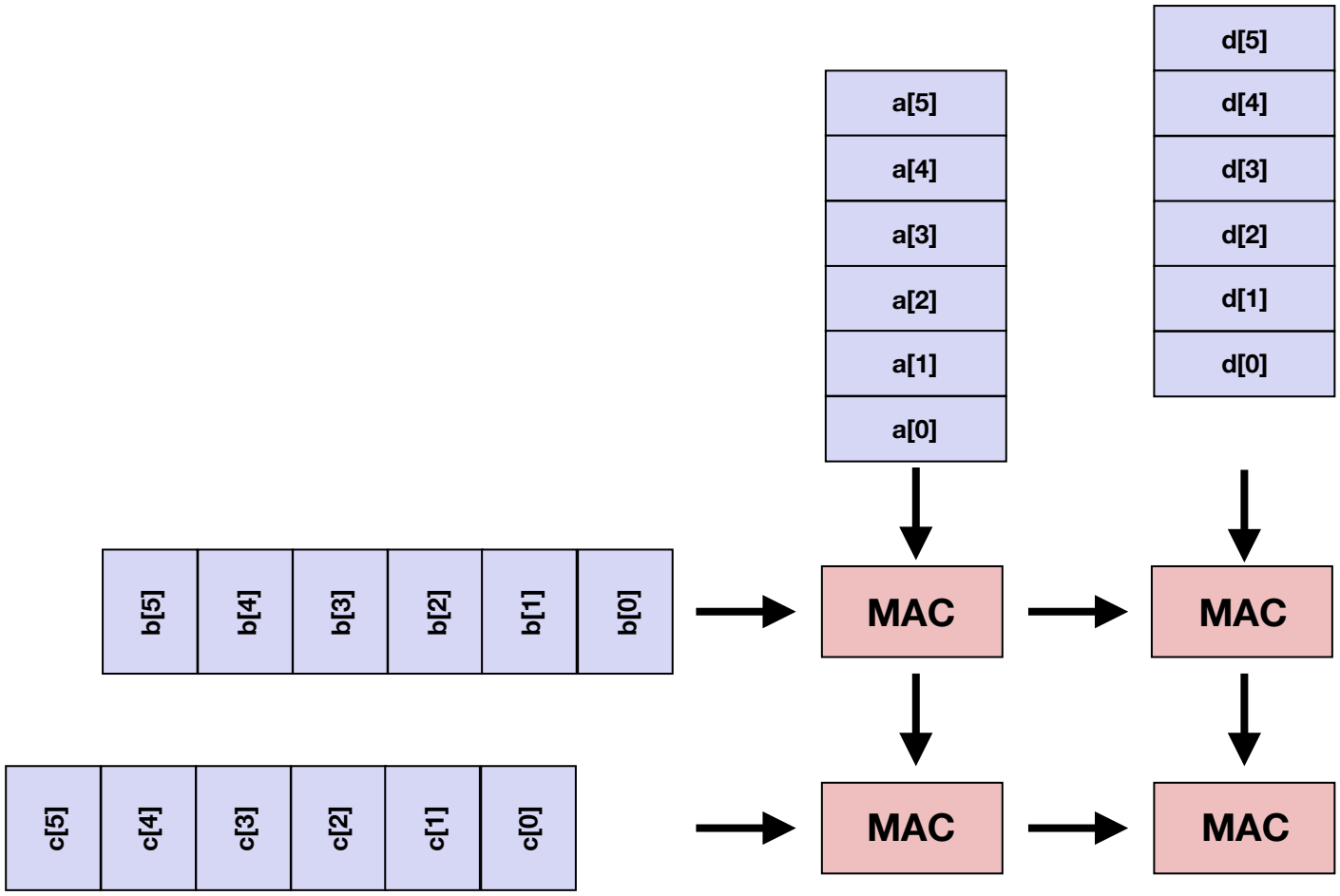
# Matrix Vector Multiplication



# Matrix Matrix Multiplication

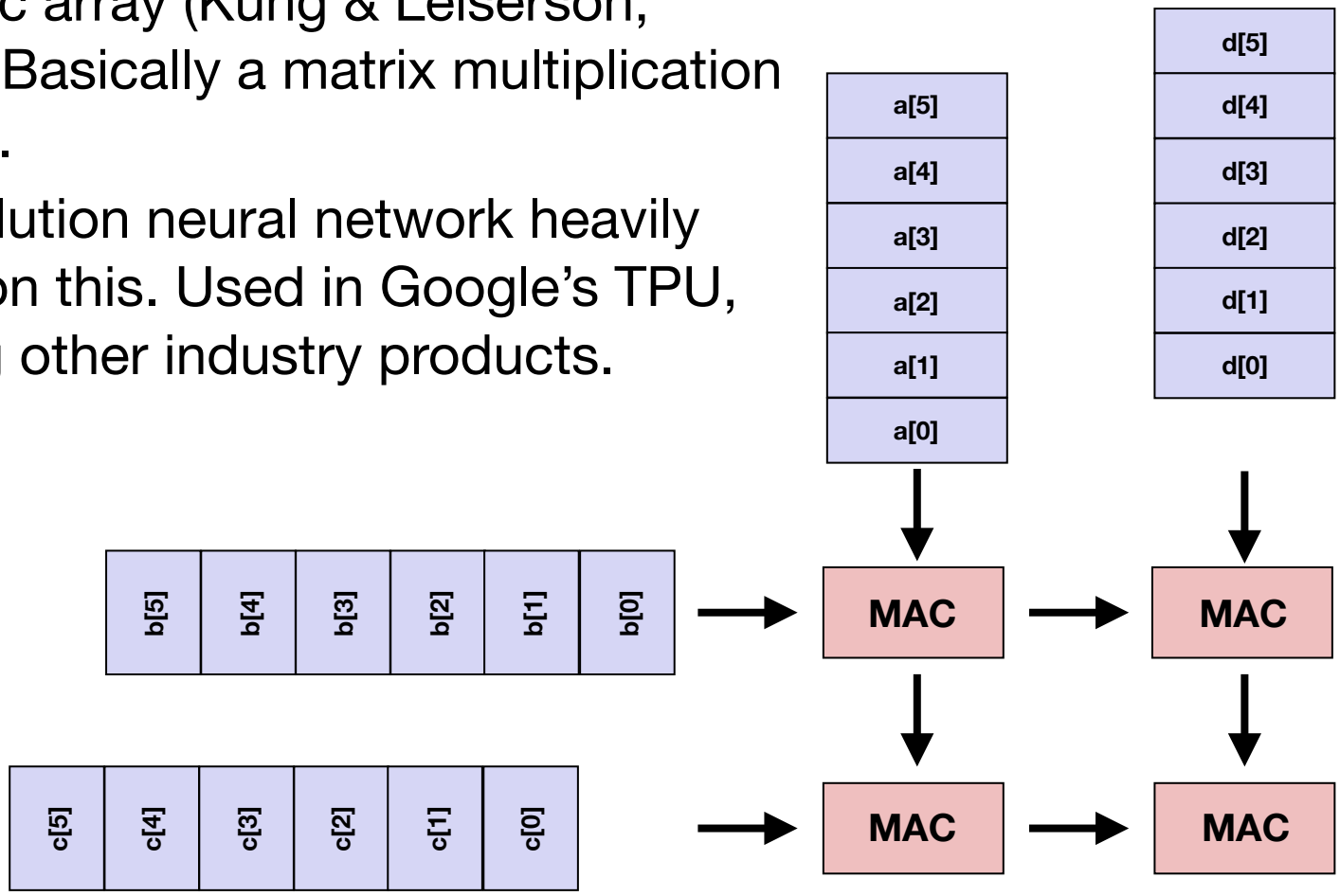


# Matrix Matrix Multiplication



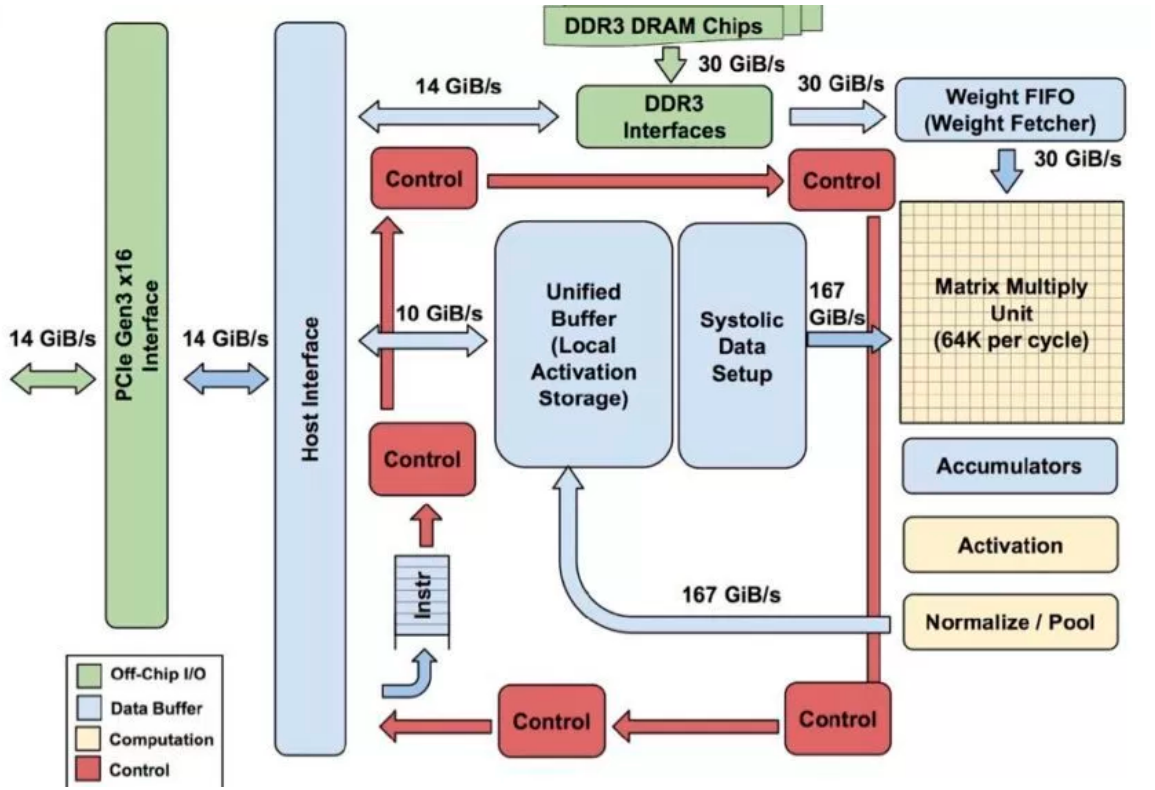
# Matrix Matrix Multiplication

- Systolic array (Kung & Leiserson, 1978). Basically a matrix multiplication engine.
- Convolution neural network heavily relies on this. Used in Google's TPU, among other industry products.



# Google Tensor Processing Unit

- Convolution in deep learning can be transformed to matrix multiplication.
- TPU: specialized processor (i.e., systolic array architecture) for tensor processing (matrix multiply)
  - 30x~80x more power-efficient than GPU

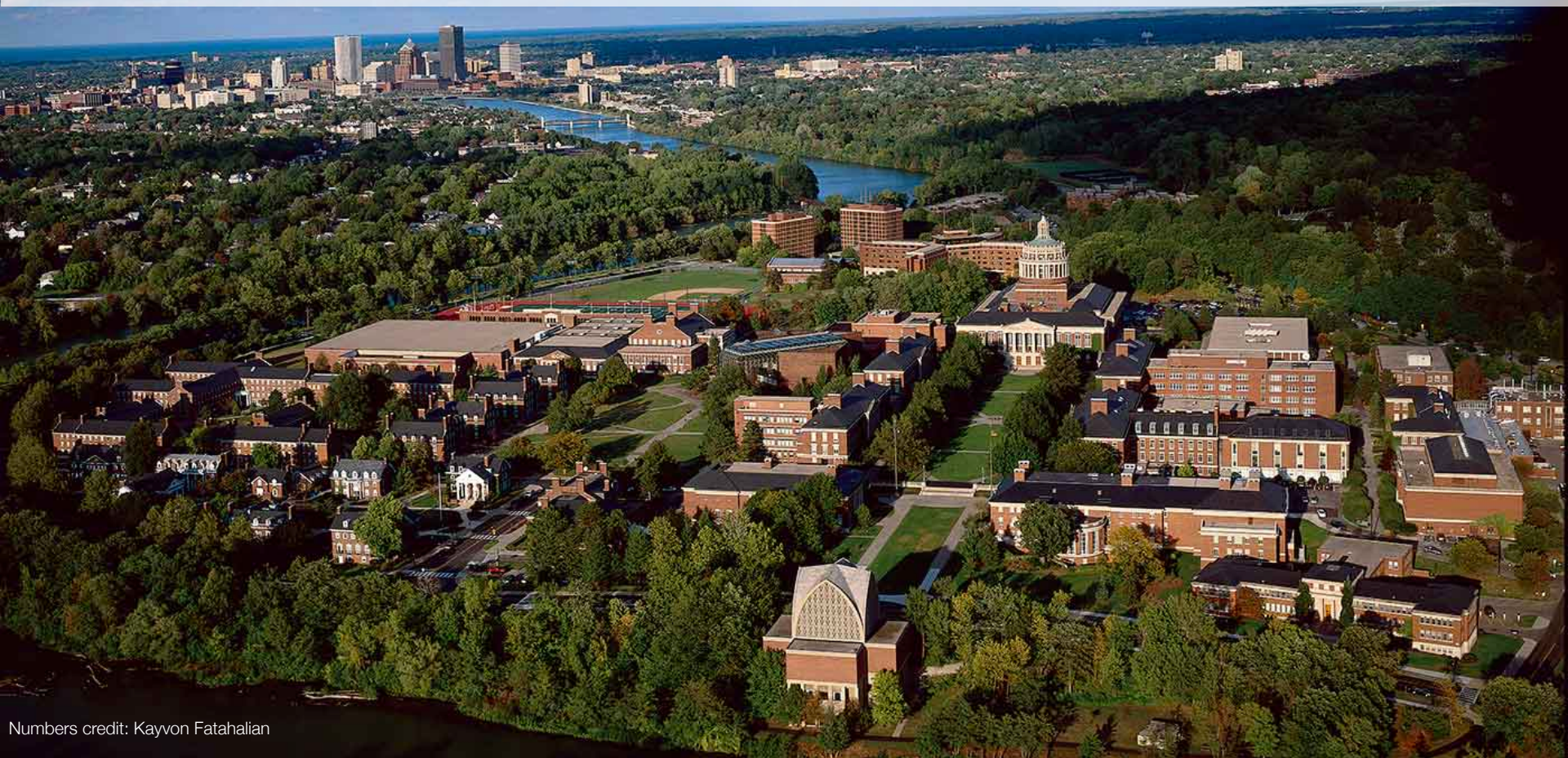


# Another Domain: Video Compression

Numbers credit: Kayvon Fatahalian

# Another Domain: Video Compression

30-second video @ 1080p resolution (1920 x 1080 pixels per frame) @ 30 frames per second (FPS)
3 colors per pixel + 1 byte per color → 6.2 MB/frame → 6.2 MB x 30 s x 30 FPS = 5.2 GB total size
Actual H.264 video file size: 65.4 MB (**80-to-1 compression ratio**).
**Compression/encoding done in real-time without you even realizing it!**



Numbers credit: Kayvon Fatahalian

# Another Domain: Computational Photography

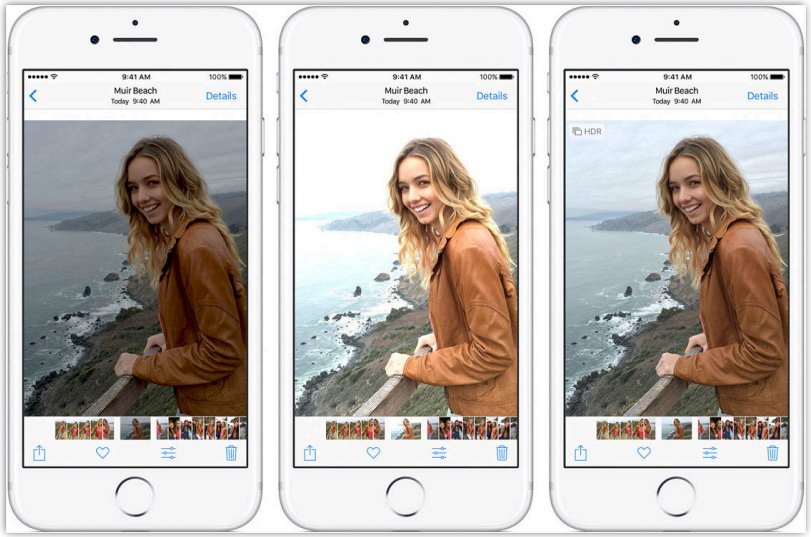
- Use computational algorithms to mimic a DSLR.
- Must be done in real-time. Executed on a dedicated Image Signal Processor (ISP).


**Conventional cameras**

**Today's "cameras"**

# Another Domain: Computational Photography

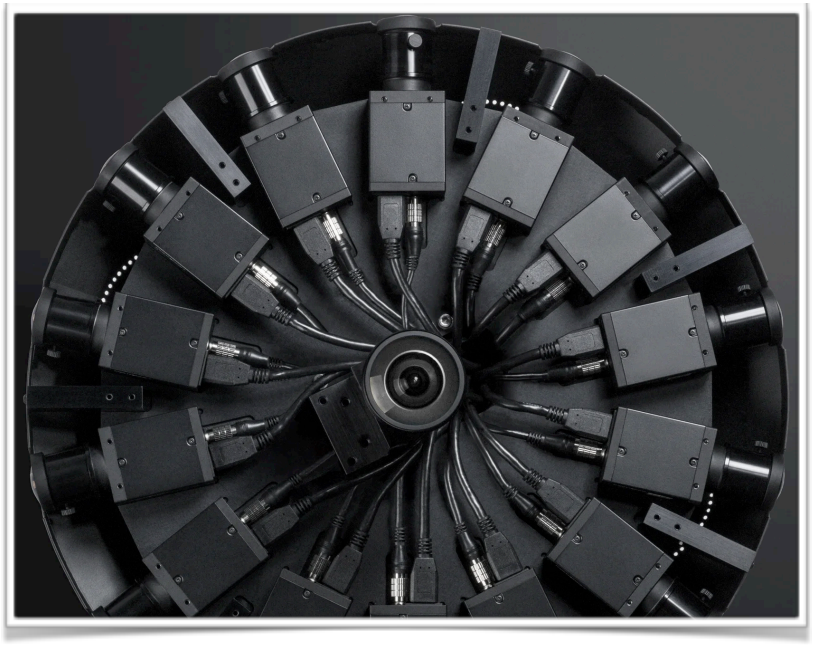
**Portrait mode: simulate a large aperture**




**HDR mode: simulate a high dynamic range sensor**

# 360° (VR/Panoramic) Videos and Photos

# VR Video Capturing



Facebook Surround 360

Google Jump VR

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45617.pdf
https://www.wired.com/2016/04/heres-360-degree-video-camera-facebook-giving-away/
https://twitter.com/yitechnology/status/918116570559336448

# Autonomous Machines


https://www.wired.com/story/news-rules-clear-way-self-driving-cars/

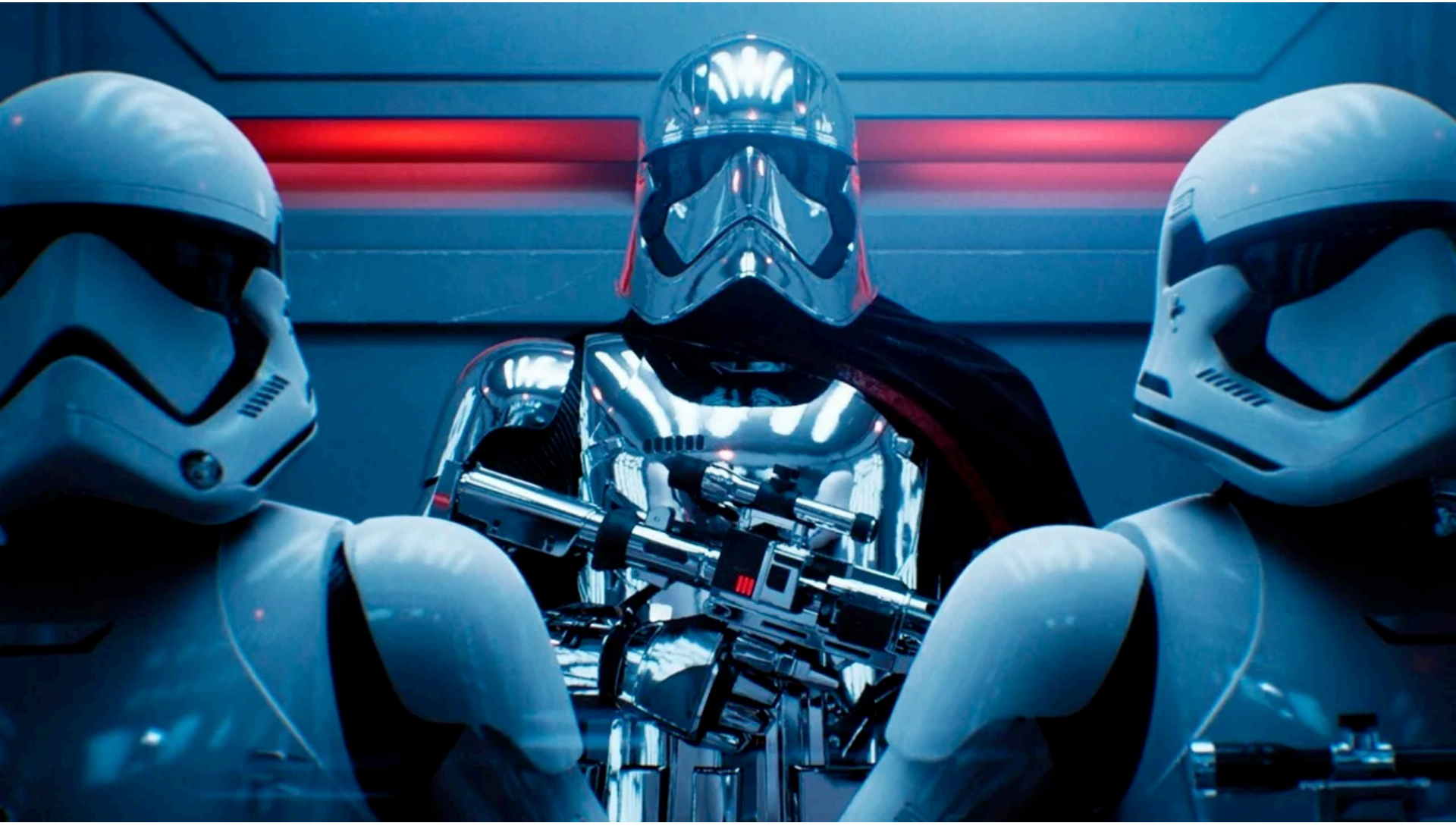
# Autonomous Machines

https://www.wired.com/2017/05/the-physics-of-drones/

# Photorealistic Rendering

https://notrianglestudio.com/all-categories-blog/why-photorealistic-renderings-should-be-part-of-your-real-estate-marketing-strategy

# Photorealistic Rendering

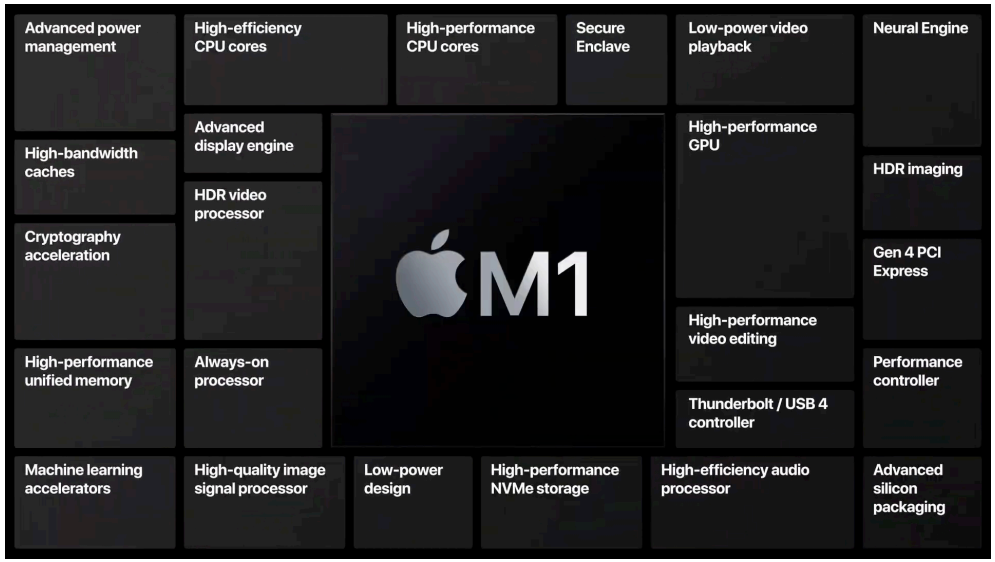


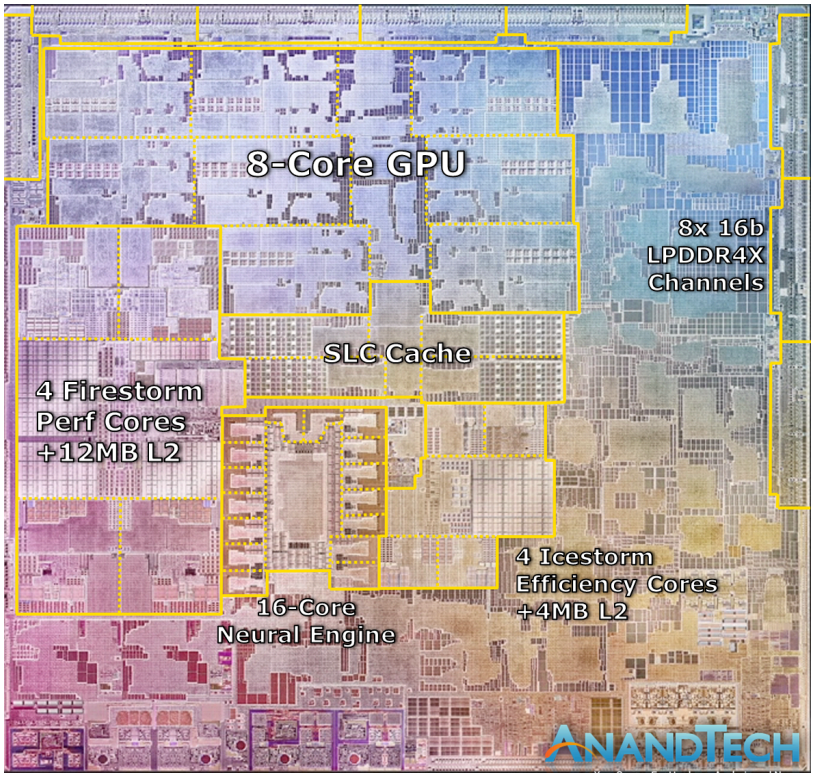
https://www.youtube.com/watch?v=uY4cE_nq2IY

# Photorealistic Rendering


https://www.digitaltrends.com/gaming/battlefield-v-dxr-ray-tracing-tested/

# Today's Processor Chips are Full of Accelerators

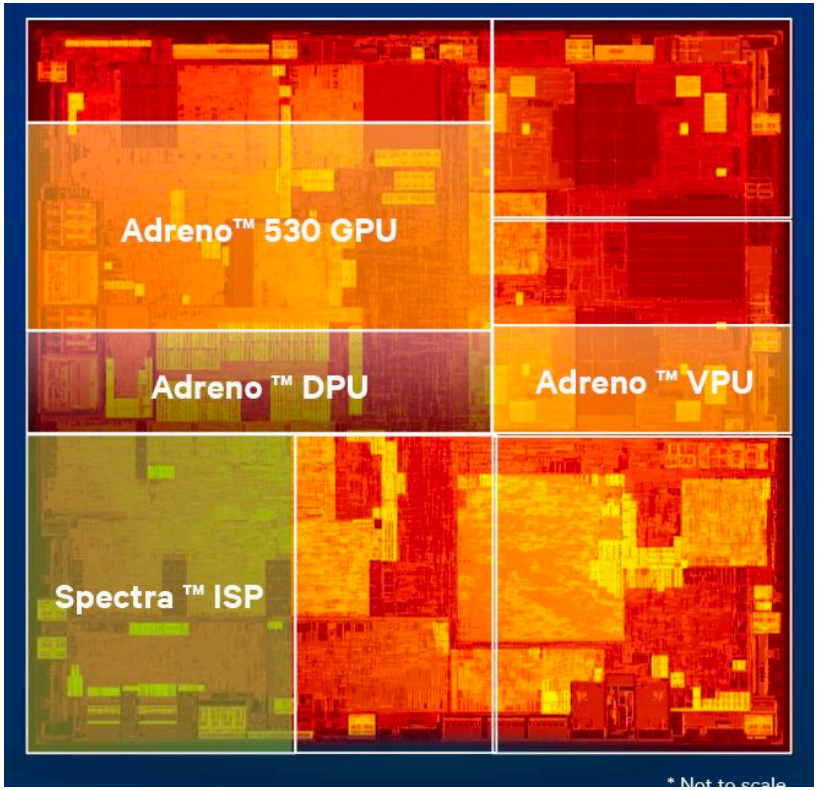


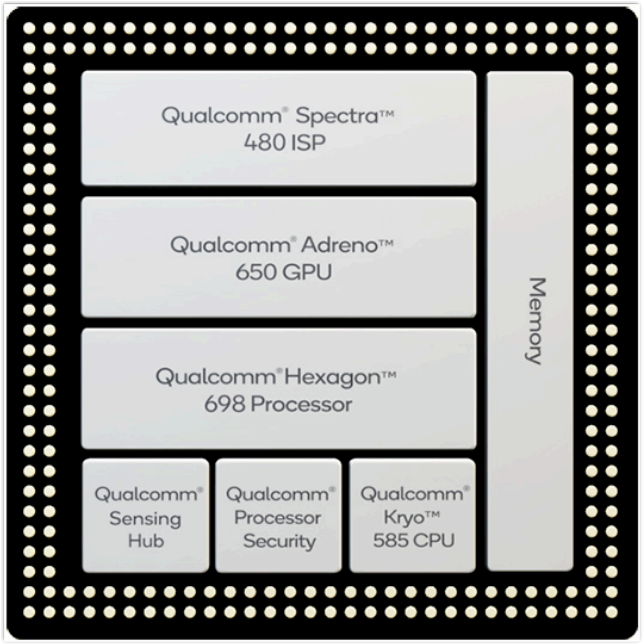

https://www.techradar.com/news/apple-m1-chip https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive

# Today's Processor Chips are Full of Accelerators

Qualcomm
Snapdragon
820 SoC

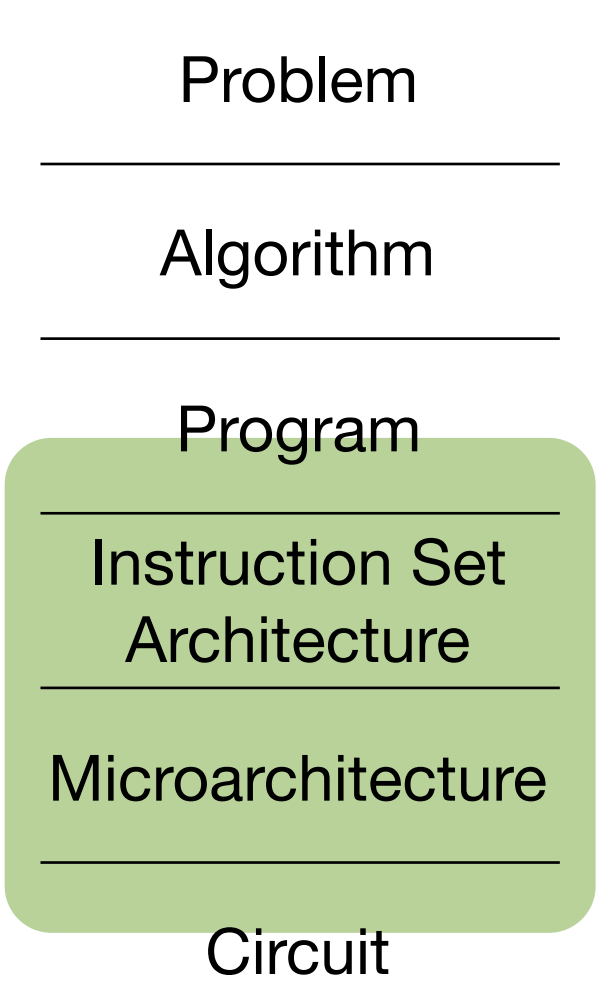



Qualcomm
Snapdragon
835 SoC



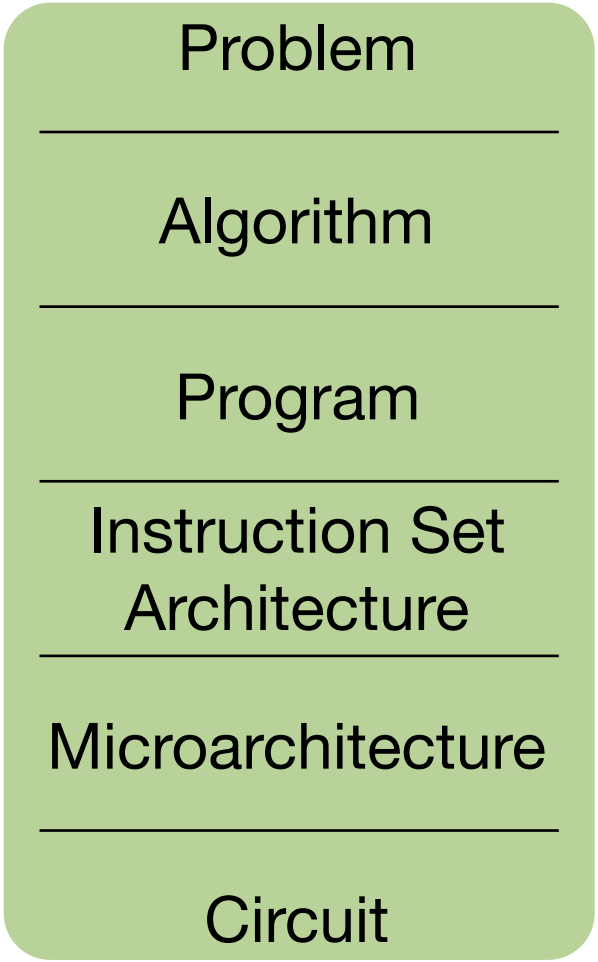
# Traditional Scope of Computer Systems

- Take a program and try to figure out how to best execute on the hardware

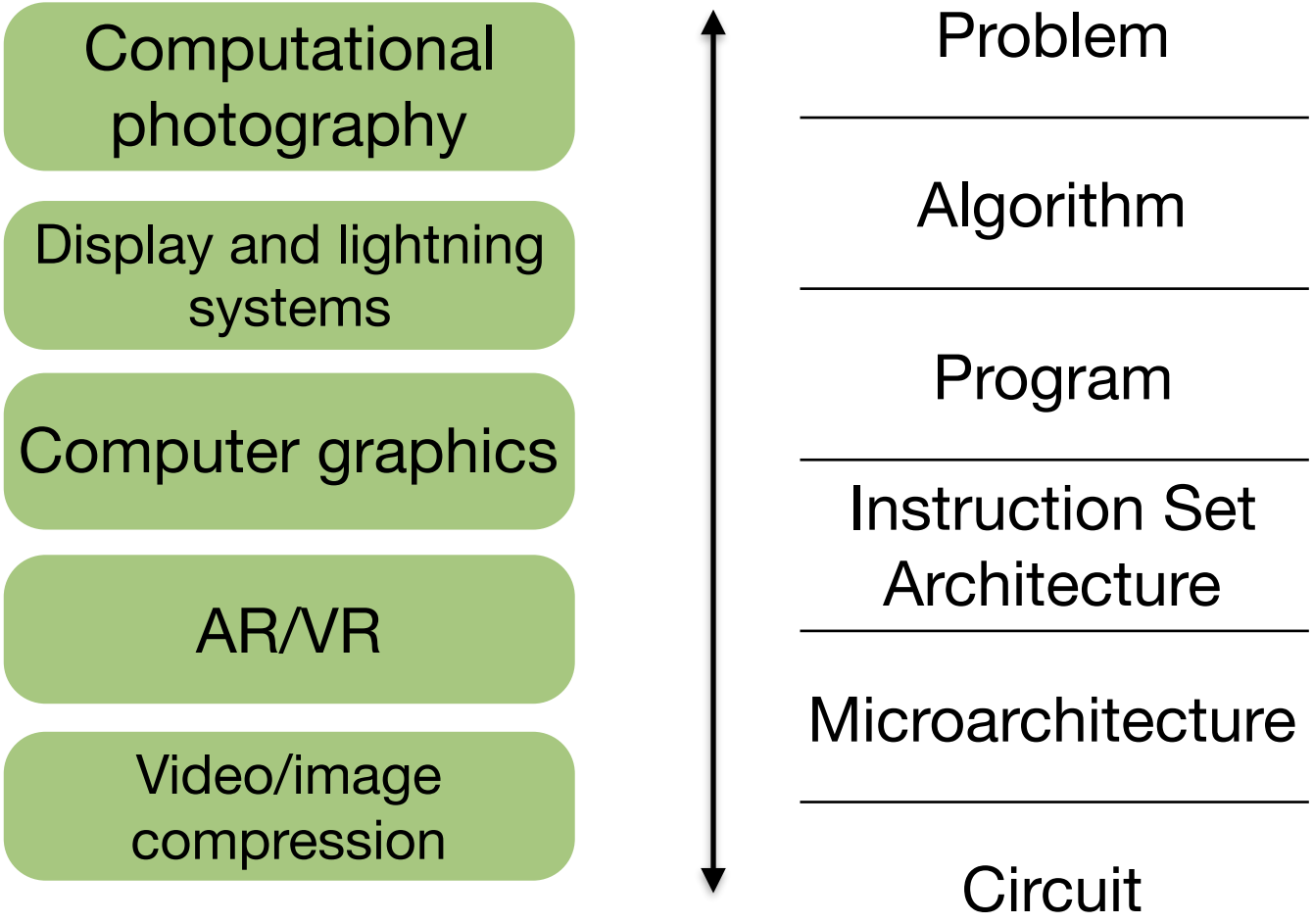


# Real Scope of Computer Systems

- Understand the problem to be solved, design algorithms, understand algorithms characteristics to design the best computer systems.
- It's no longer enough to work with a given program without understanding it.



# CSC 292/572: Mobile Visual Computing



# The Most Important Take Away of 252

- “There is no magic.”
- Every thing can be derived from first principles. Trust your logical reasoning.
- Apply to virtually everything in science and engineering.

# The Second Most Important Take Away of 252

- “Things don’t have to be this way.”
- As long as you don’t violate physics, you can design a computer however you want.
- But every design decision you make usually involves certain trade-offs. Be clear what your design goal is.

# The Third Most Important Take Away of 252

- Virtual all computer system design practices follow a small set of basic principles.
- It is these basic principles that are important, not the practices.