The inexorable rise of hardware specialization

Nathan Beckmann, CMU

~~CSD faculty meeting, Fall '21~~

15-418/618 Fall '25

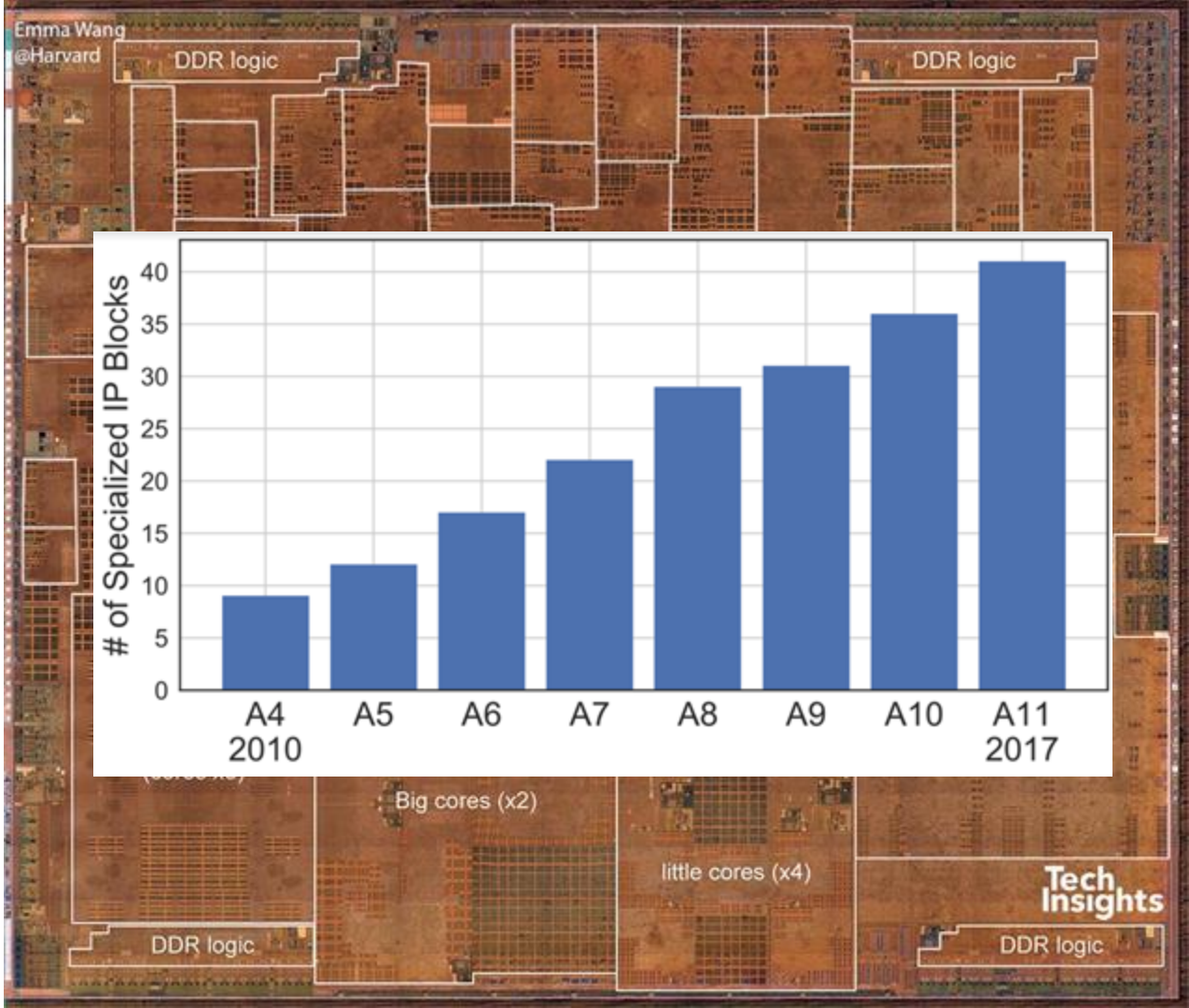
The world today:

Apple A12 (2019)

- 42 different “IP blocks” on one processor chip
- High-performance CPUs
- Energy-efficient CPUs
- GPU
- “Neural engine”
- + 38 other specialized widgets

Traditional CPUs are a small fraction of the chip!

Credit: Emma Wang & Sophia Shao, http://vlsiarch.eecs.harvard.edu/research/accelerators/die-photo-analysis/

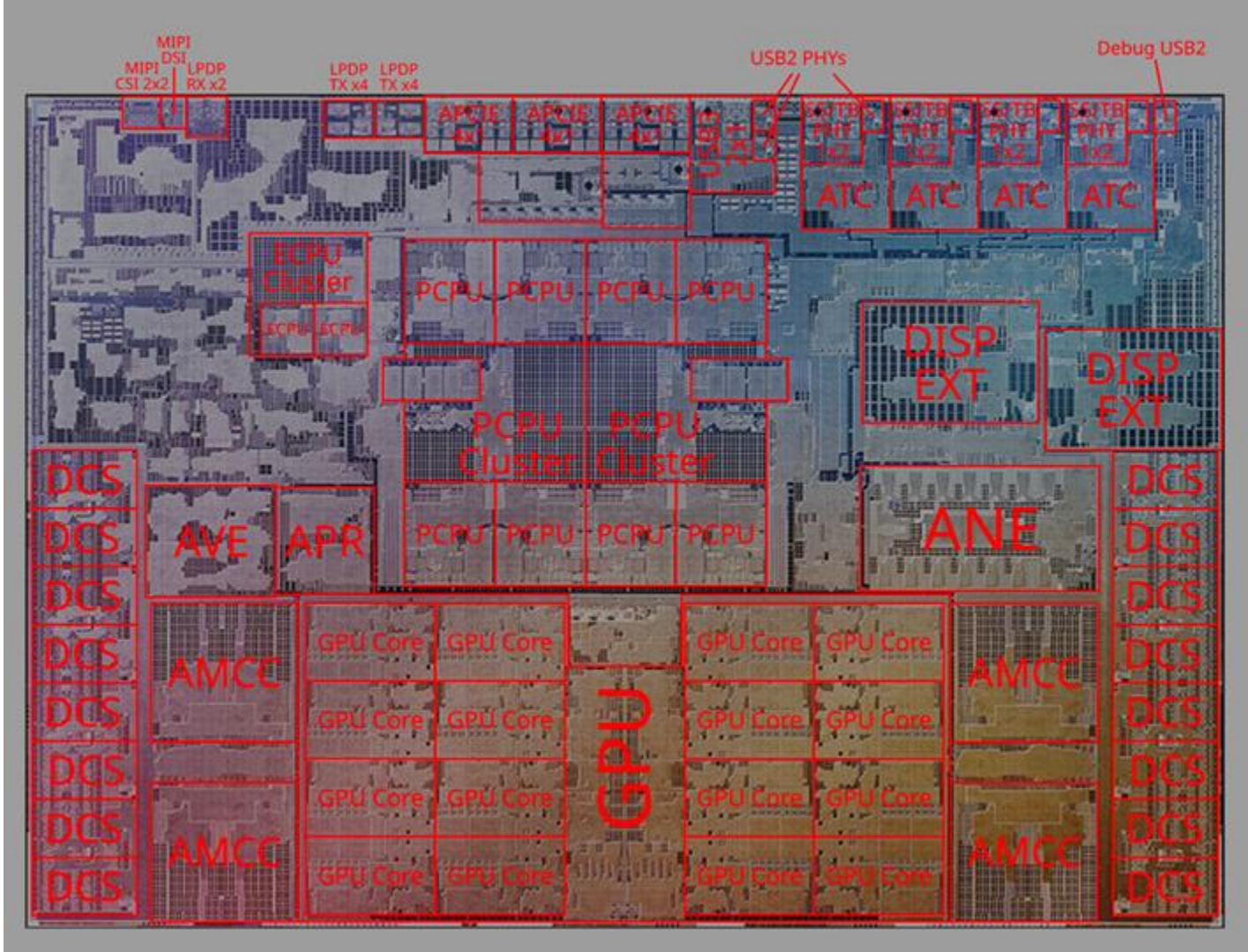


Laptops, too

Apple M1 chip

- Big CPUs
- Small CPUs
- GPU
- “Neural engine”
- Video encoder
- PCIe controller
- Etc.

Credit: Hector Martin
https://twitter.com/marcan42/



Also, Intel!

Intel Alder Lake

- Big CPUs
- Little CPUs
- GPU
- Etc

Credit: Nemez
https://twitter.com/GPUsAreMagic

What’s going on with hardware?

Takeaways

1. The world is moving away from traditional, fully programmable cores
2. “Non-von Neumann” computing seems to be the way to continue scaling
3. There isn’t a very compelling story for how one should write programs
4. The best balance between programmability & efficiency is an open question

Why should you care about hardware?

Goal: Performance & efficiency scaling

Hardware performance has grown exponentially for decades

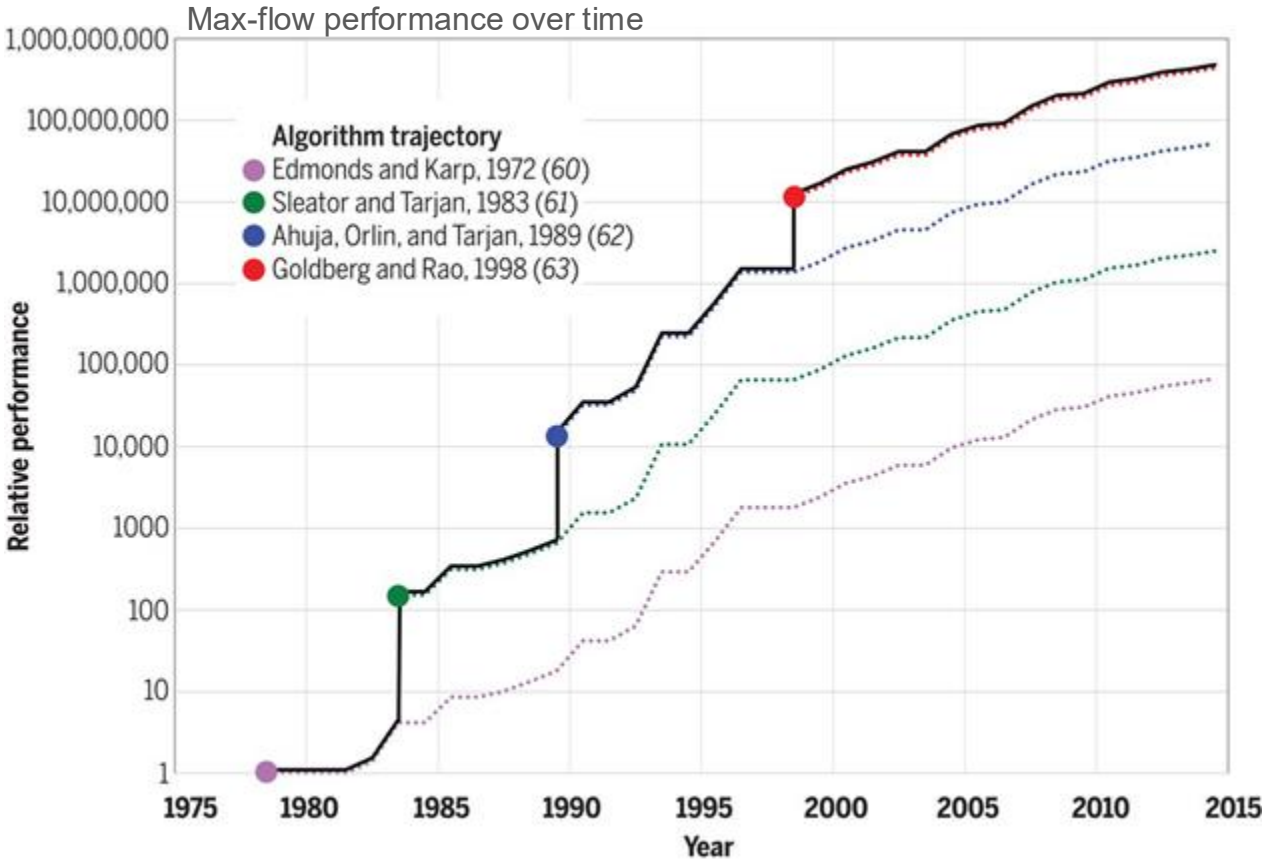
Basic assumption: *This is good, and we want it to continue*

Algorithms, hardware, or both?

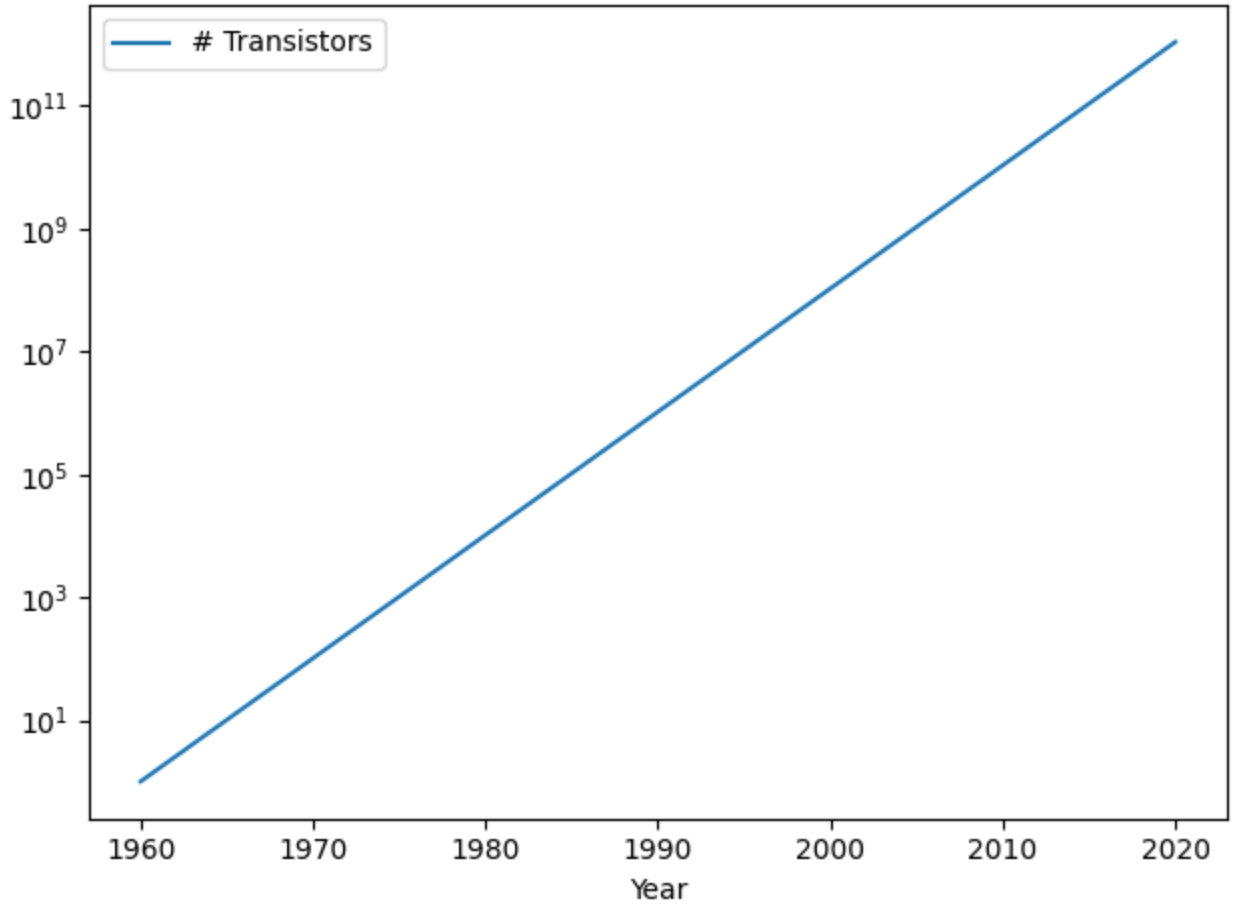
Similar order-of-magnitude improvements from algorithm & hardware improvements

- Diminishing returns from both!

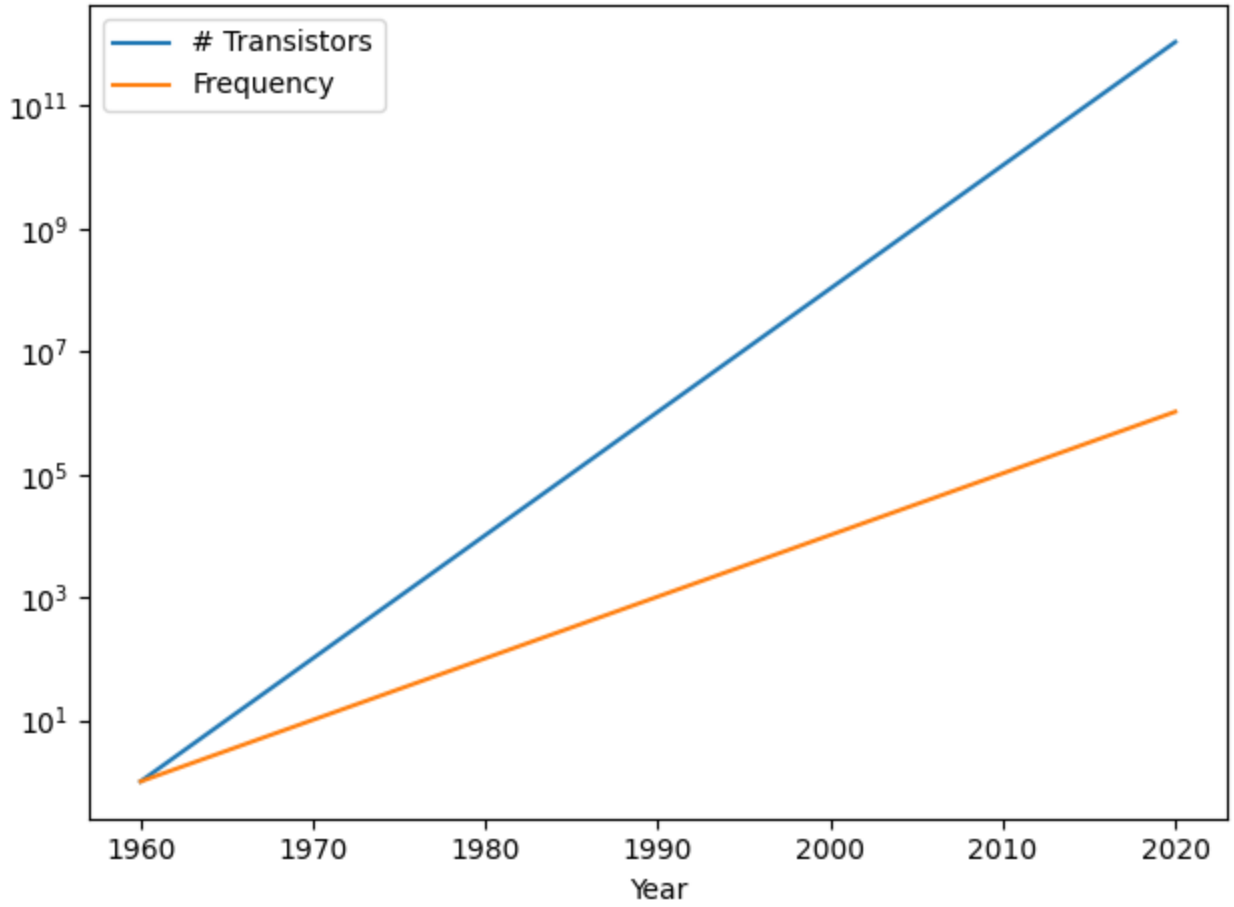
Credit: “There’s plenty of room at the top”, Leiserson+, Science ‘20



OK, so what’s going on with hardware?

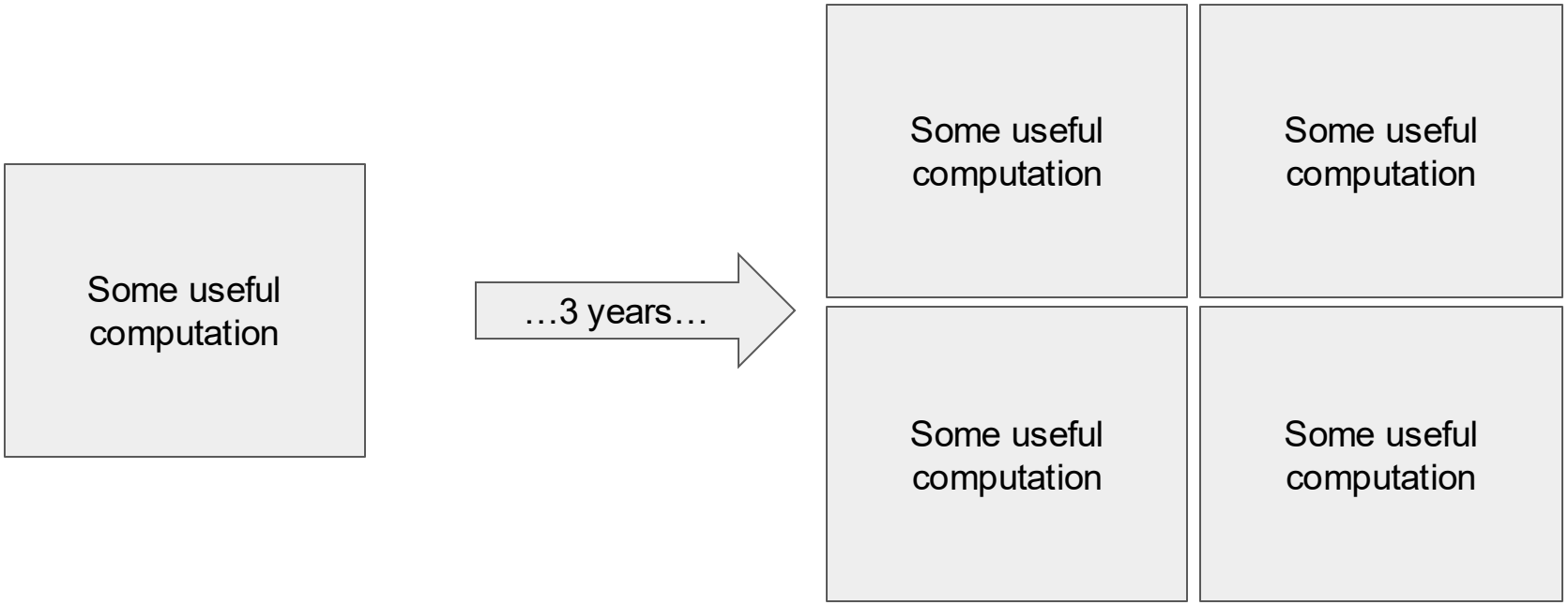


Moore’s Law: 2x transistors every 18 months



Smaller transistors are faster, too!

Optimistic performance model



How does performance scale?

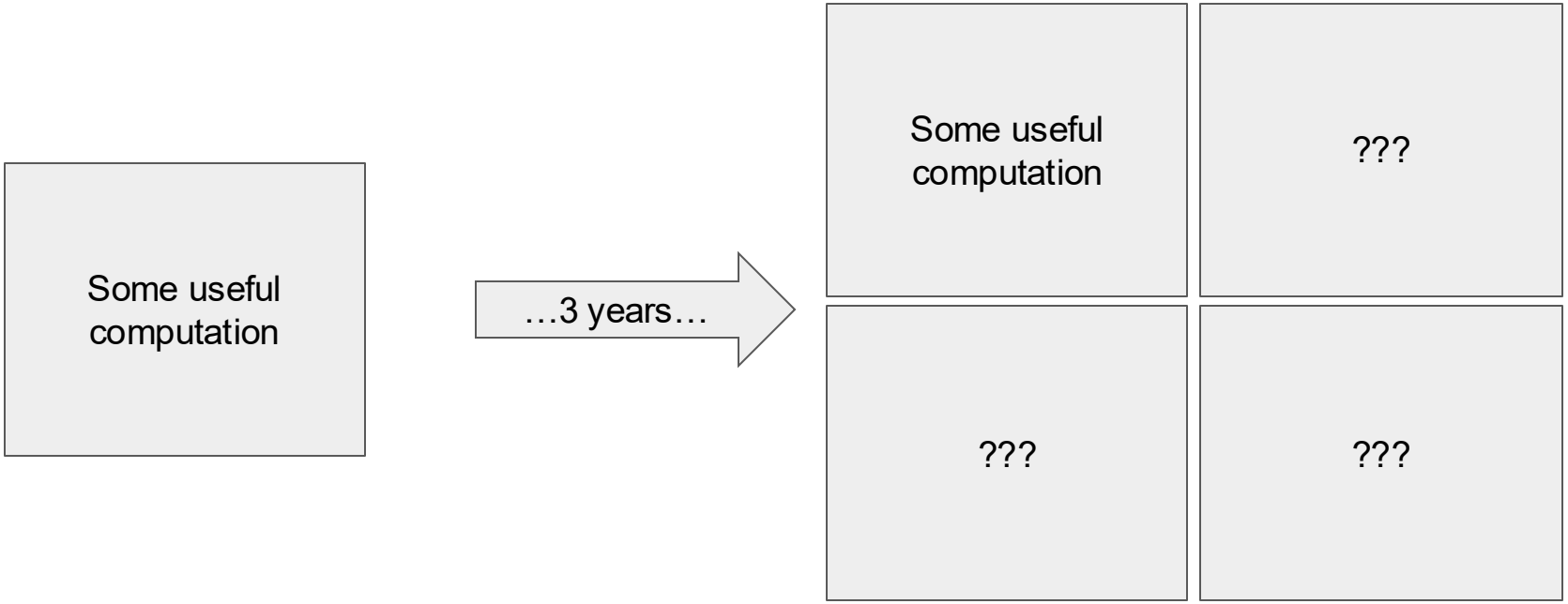
Optimistic performance = Number of transistors × frequency = N f

N’ = 2 N

f’ = √2 f

⇒ Optimistic scaling factor = 2√2× per technology generation

Pessimistic performance model

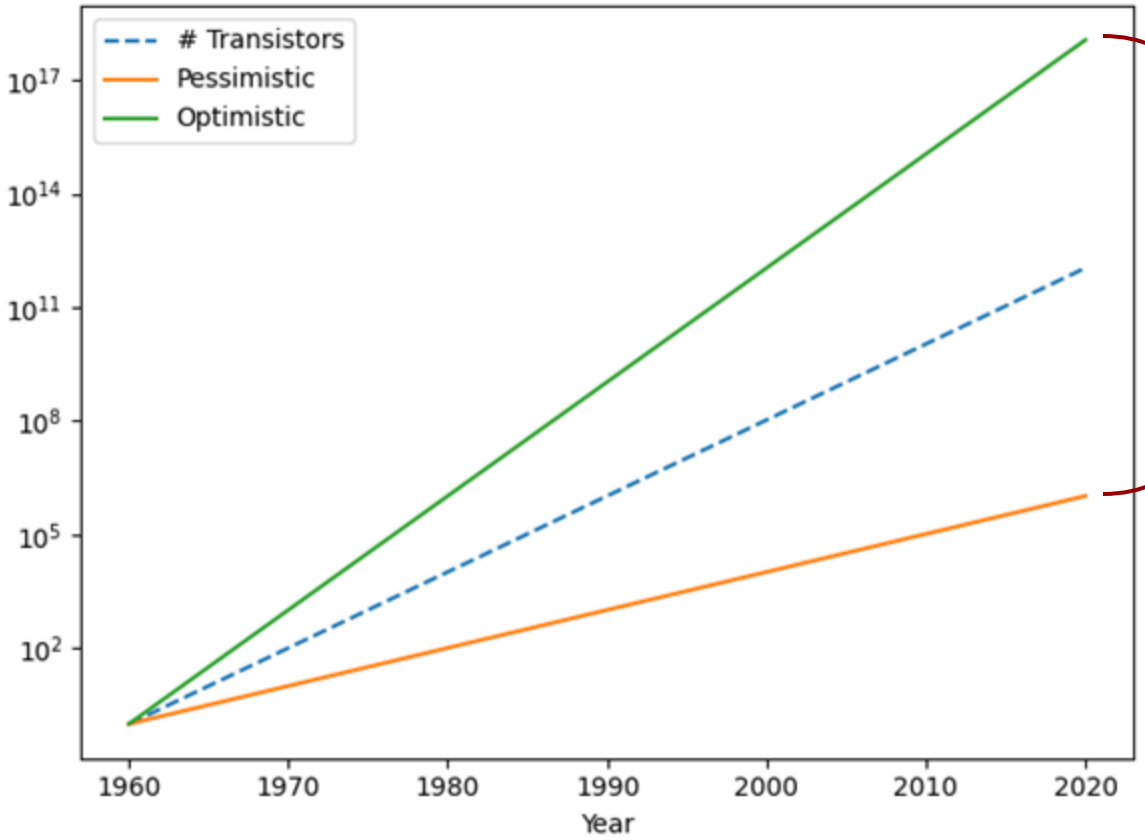


How does performance scale?

Pessimistic performance = frequency = f

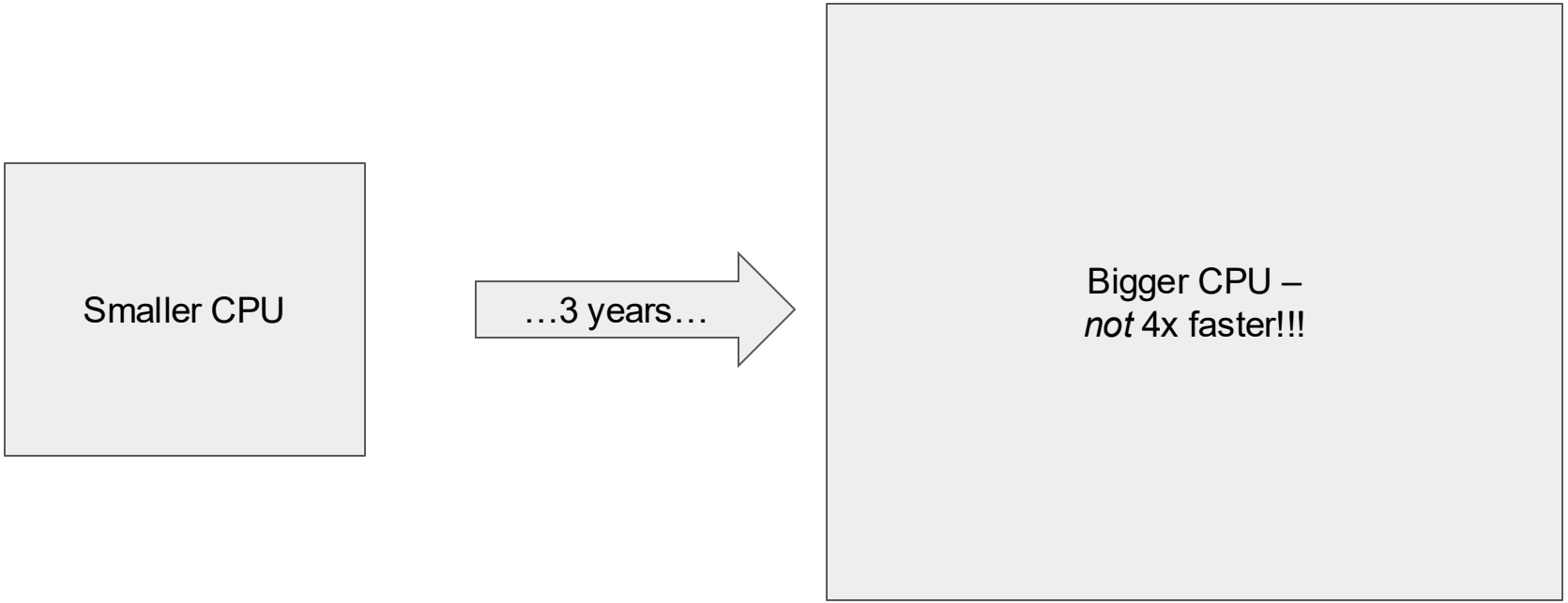
$f' = \sqrt{2}\, f$

$\Rightarrow$ Pessimistic scaling factor = $\sqrt{2}\times$ per technology generation



Huge performance gap! *Need to put extra transistors to good use!*

Better historical performance model



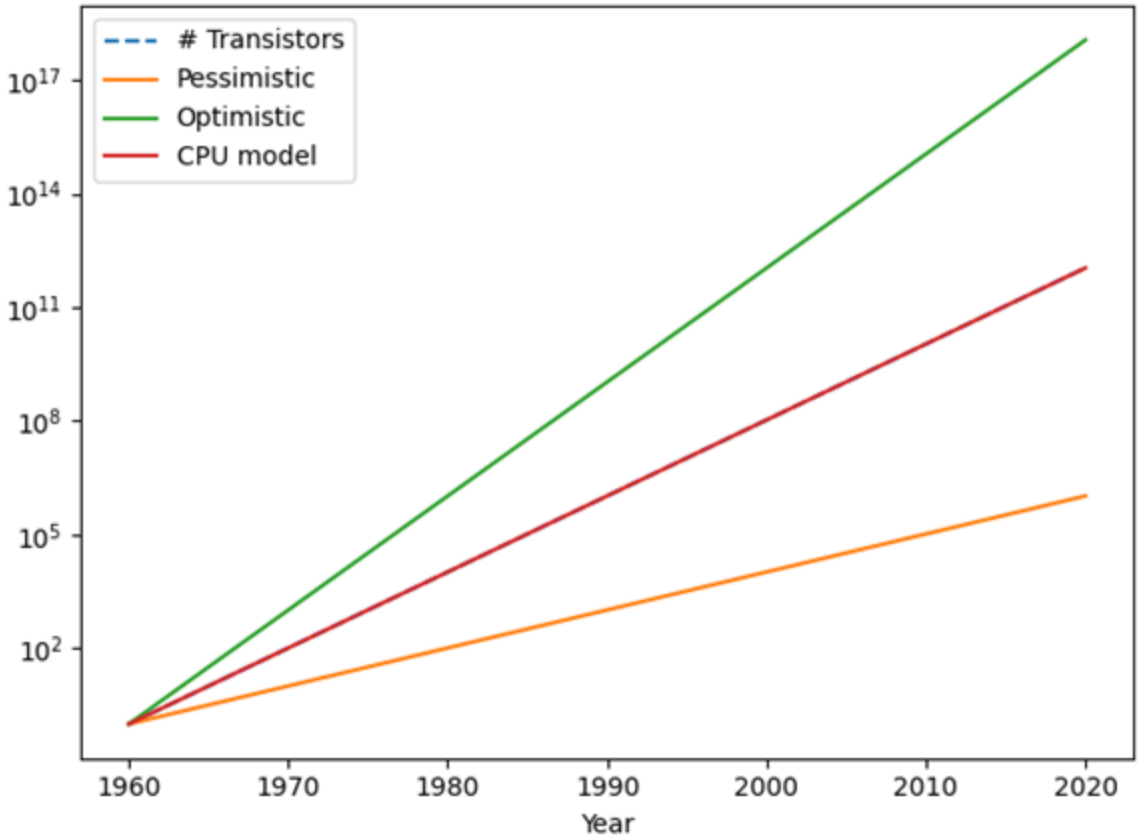
Better historical performance model

Realistic performance ≈ $\sqrt{N}$ f (slightly optimistic)

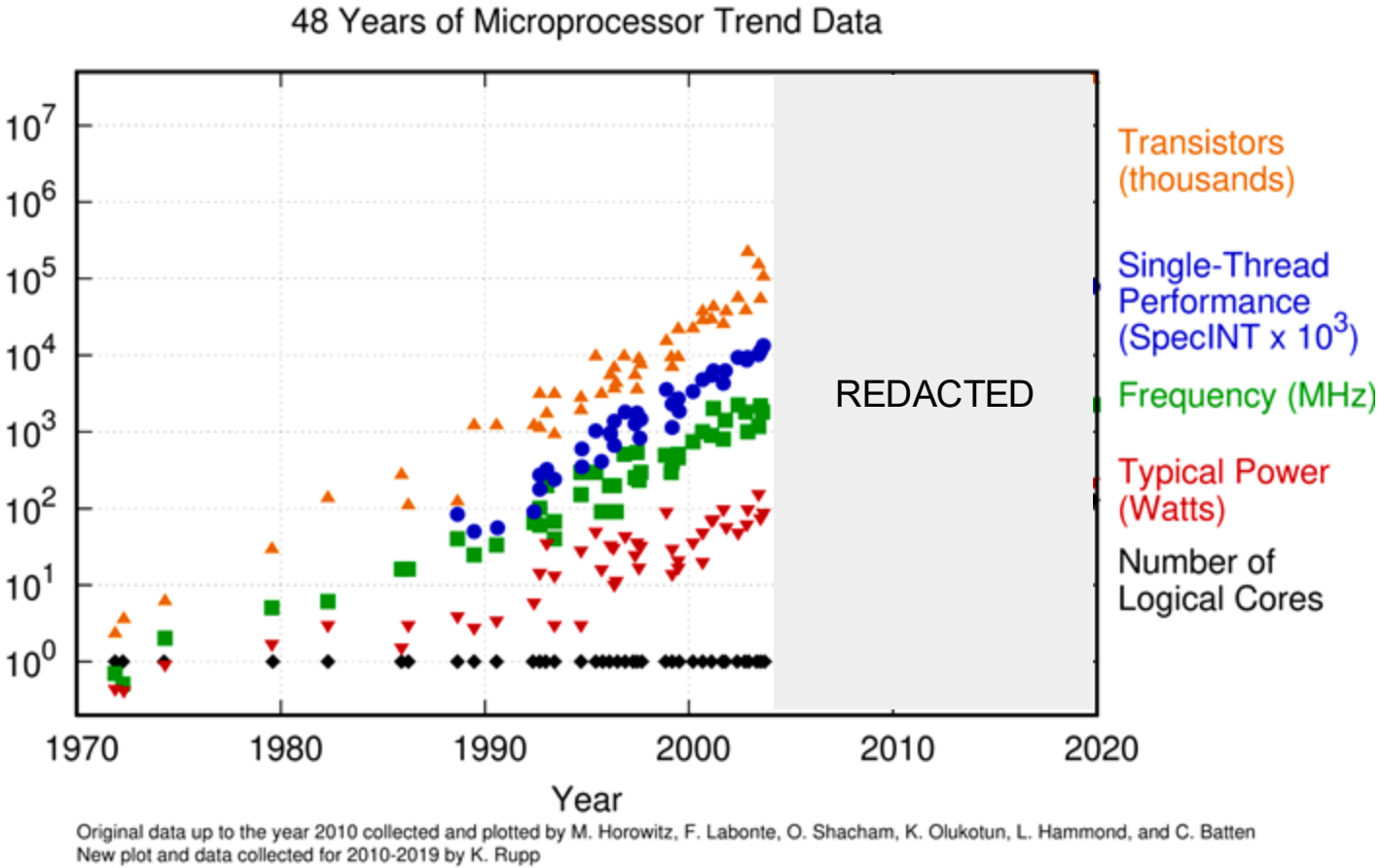
N’ = 2N

f’ = $\sqrt{2}$f

⇒ Scaling factor = 2× per technology generation



CPUs close ~half the gap & are easy to program



Cartoons aren't far from reality!

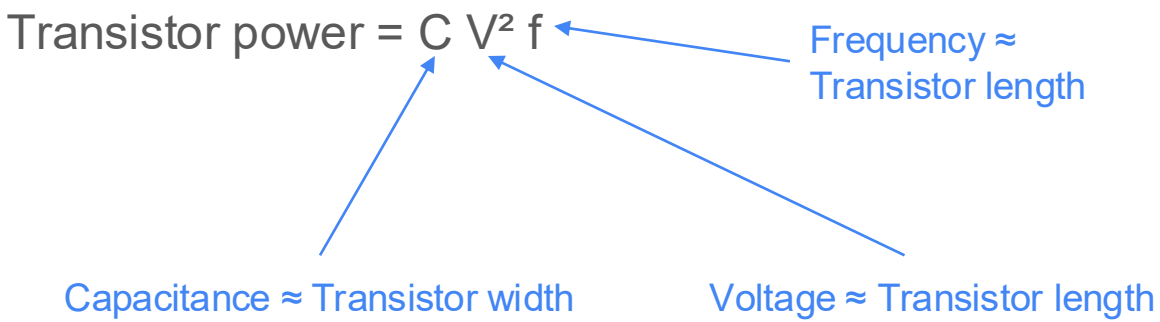
The elephant in the room

What about power & energy?

Power = Number of transistors $\times$ Transistor power

What about power & energy?

Power = Number of transistors × Transistor power



How do these parameters scale?

After one step of Moore's Law…

$N' = 2\,N$ — Area × 1/2

$C' = C / \sqrt{2}$

$V' = V / \sqrt{2}$

$f' = \sqrt{2}\,f$

} Linear dimension × $1/\sqrt{2}$

(All of the above just from the geometry of a smaller transistor!)

Dennard scaling: Power is constant!

$$\text{Power}' = N' C' V'^2 f'$$

$$= 2N (C/\sqrt{2}) (V / \sqrt{2})^2 \sqrt{2}f$$

$$= N C V^2 f$$

$$= \text{Power}$$

Power is **constant!**

(Even with $2\sqrt{2}\times$ more transistor toggling)

Dennard scaling ended in ~2003.

(Voltage can only drop up to a point. Band gap, and all that.)

What does this mean for performance scaling?

Without Dennard scaling…

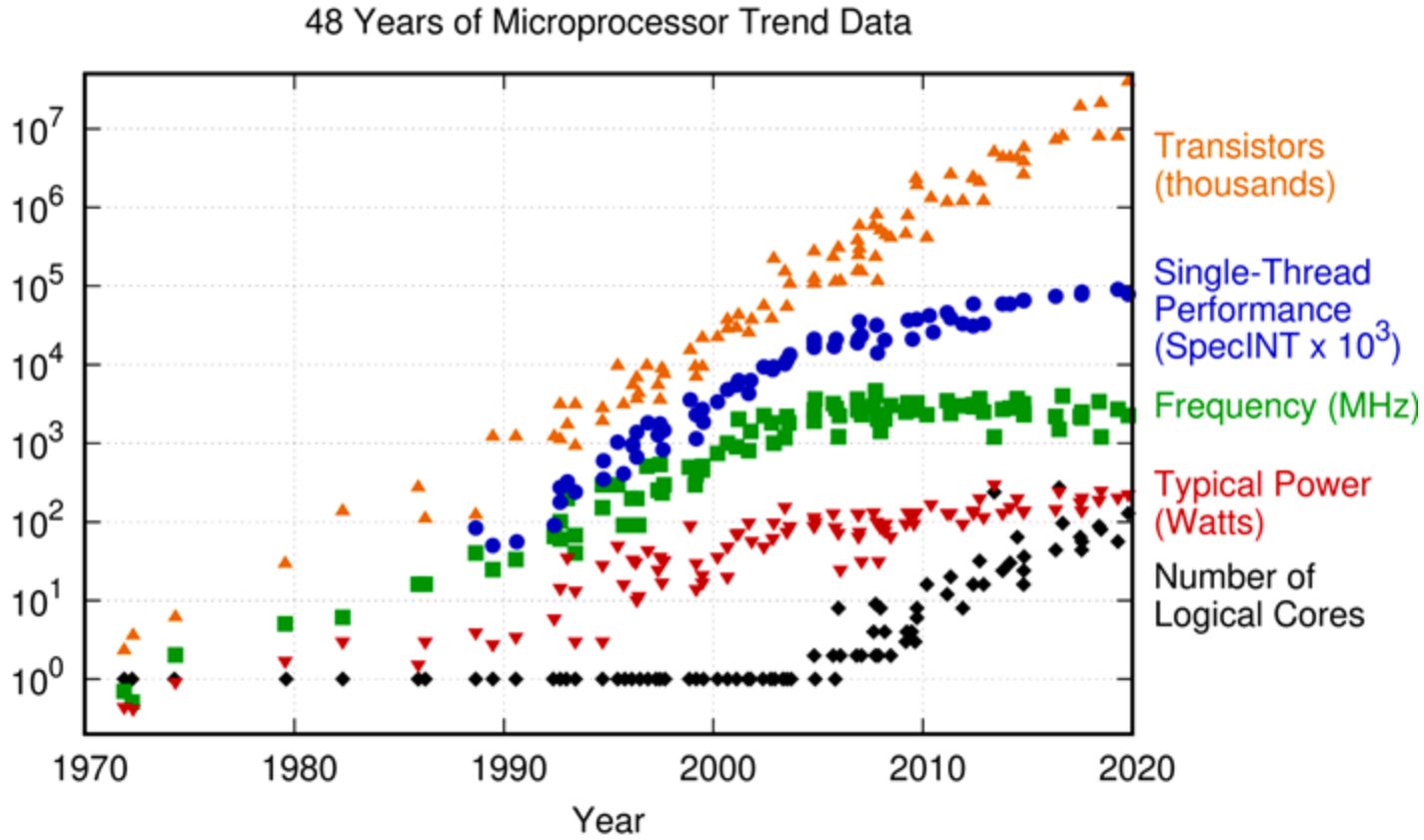
uh-oh! 😟

$$\text{Power'} = N' C' V'^2 f' = 2N (C/\sqrt{2}) V^2 \sqrt{2} f = \mathbf{2\times}\ \text{Power}$$

But power *must* remain ~constant! (heat sinks have limits)

$$(\text{Power'} / \text{Power}) = 1 \quad \Rightarrow \quad N' f' = \sqrt{2}\, N f$$

Steep decrease in performance scaling (lose 2× off optimistic)


48 Years of Microprocessor Trend Data
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1970
1980
1990
2000
2010
2020
Year
Transistors (thousands)
Single-Thread Performance (SpecINT x 10^3)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
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 scaling recap

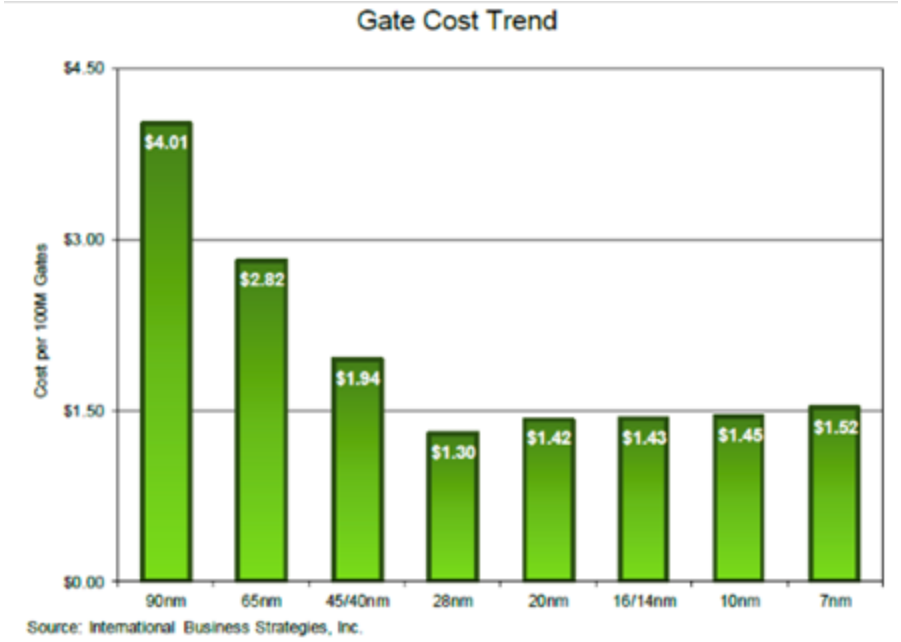
Moore's Law isn't dead

- At least not in Taiwan & Korea

But!

Cost scaling has stopped

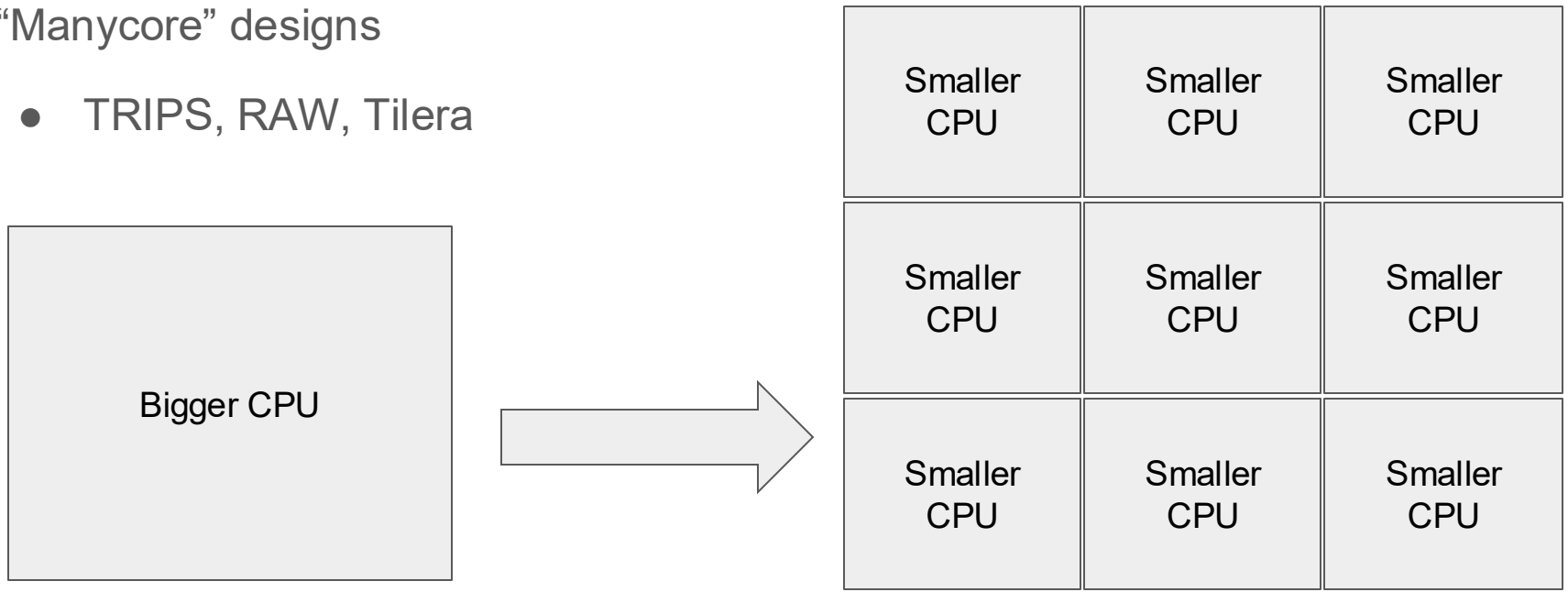
Dennard (power) scaling has stopped



Back to the drawing board…

“Manycore” designs

- TRIPS, RAW, Tilera



“Amdahl’s Law in the Multicore Era”

Mark Hill, Wisconsin
Eckhert-Mauchly ‘19

Model programs as sequential & parallel regions

Same CPU model (perf = $\sqrt{N}$), but now splitting N across different #s of cores

Amdahl’s Law: Speedup = 1 / ((1 - s) + s / P)

- s = fraction of program that can run in parallel
- P = number of processors
- (assumes ideal speedup in parallel region)

“Amdahl’s Law in the Multicore Era”

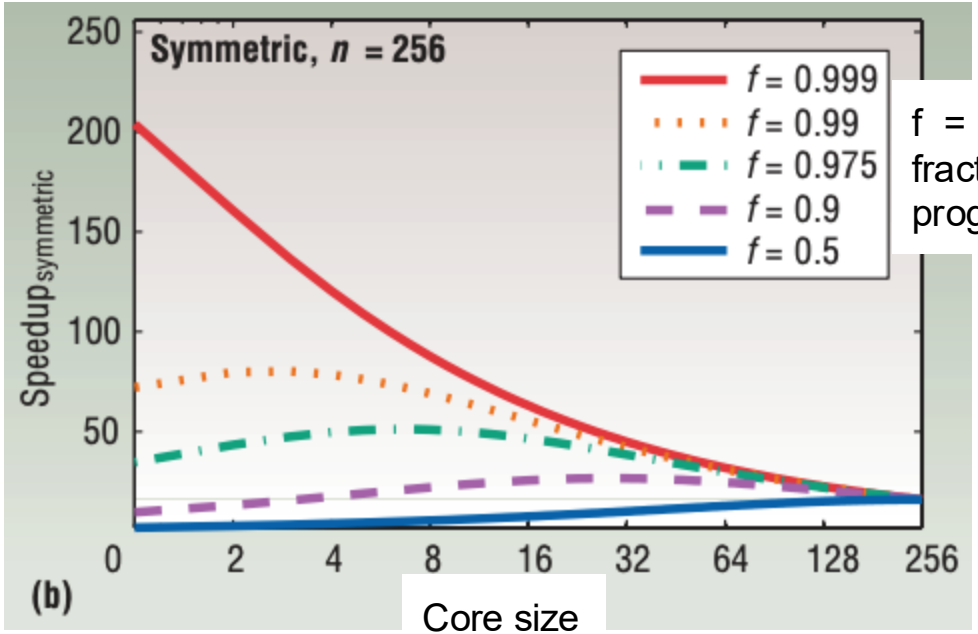
Mark Hill, Wisconsin
Eckhert-Mauchly ‘19

Speedup from a manycore design:

x = CoreSize

P = n / x

Speedup =
√x / ((1-s)+s/P)



f = Parallel fraction of program

Speedup is surprisingly awful unless program is ≥99.9% parallel!

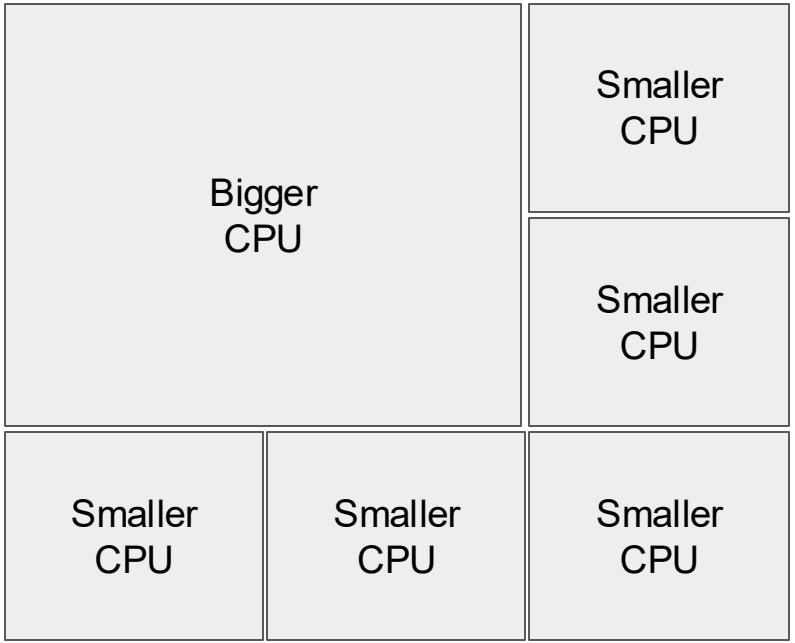
No good single choice for core size!

Back to the drawing board again…

Problem is *sequential bottleneck*

⇒ Heterogeneous multicore design

Big CPUs are locally inefficient, but globally efficient



Mark Hill, Wisconsin
Eckhert-Mauchly ‘19

“Amdahl’s Law in the Multicore Era”

Speedup from a heterogeneous/asymmetric design:

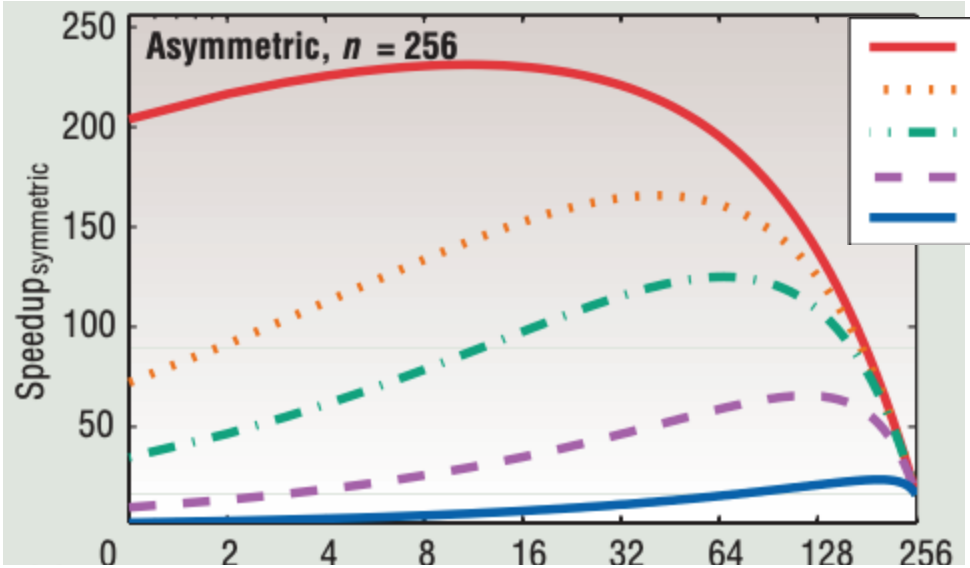
x = BigCore

PSmall = (n - x) + 1

T = PSmall + √x

Speedup =

1/((1-s)/√x + s/T)



Big core size (small core size = 1)

f = Parallel fraction of program

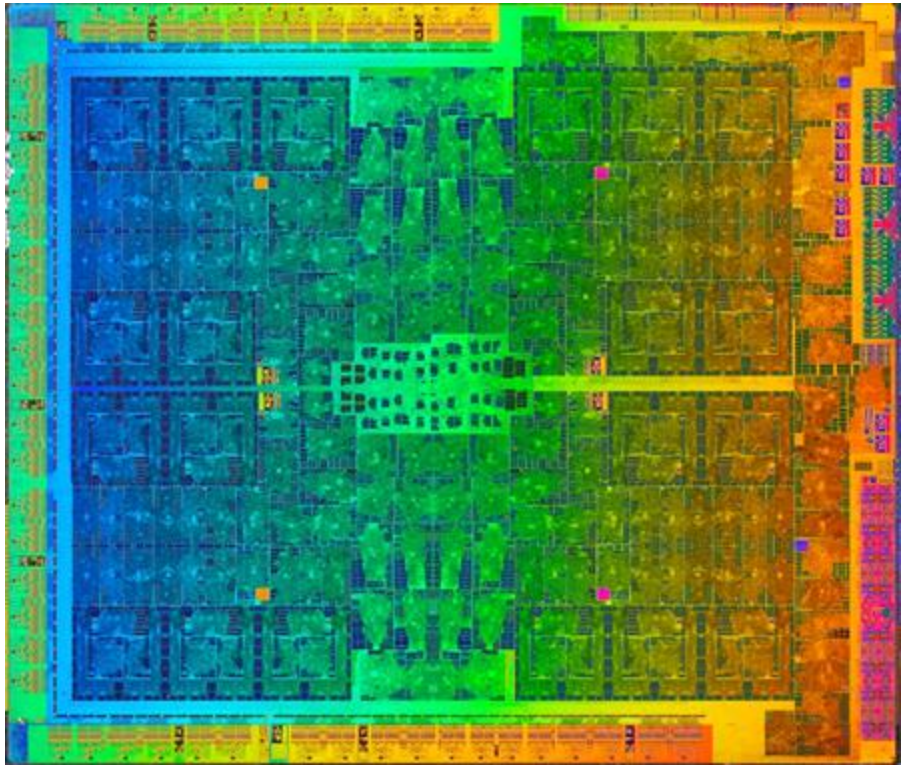
Speedup is much better & less sensitive to core size!
(But still limited by parallel fraction)

Heterogeneity & manycore in practice

Common in mobile (e.g., big.LITTLE)

Common in server systems:

- Intel Xeon is the “big CPU”
- Nvidia GPU is the “manycore” — design looks a little different, but the principle is the same
- Important: GPU is *not* an “accelerator” in this story; it is a programmable manycore (really, its somewhere in between)



Nvidia GP104

Why specialization?

“Dark silicon”

$N' = 2N$

but

$N' f' = \sqrt{2}\, N f$

⇒ We cannot use 30% of the new transistors each generation!

(Or we must run everything 30% slower…)

Dark silicon ⇒ Specialized hardware

Mounting pressure to use transistors efficiently

Conventional designs will have idle transistors

Why not add custom hardware for important computations?

“Dark silicon and the end of multicore scaling”, Esmaeilzadeh+ ISCA’11

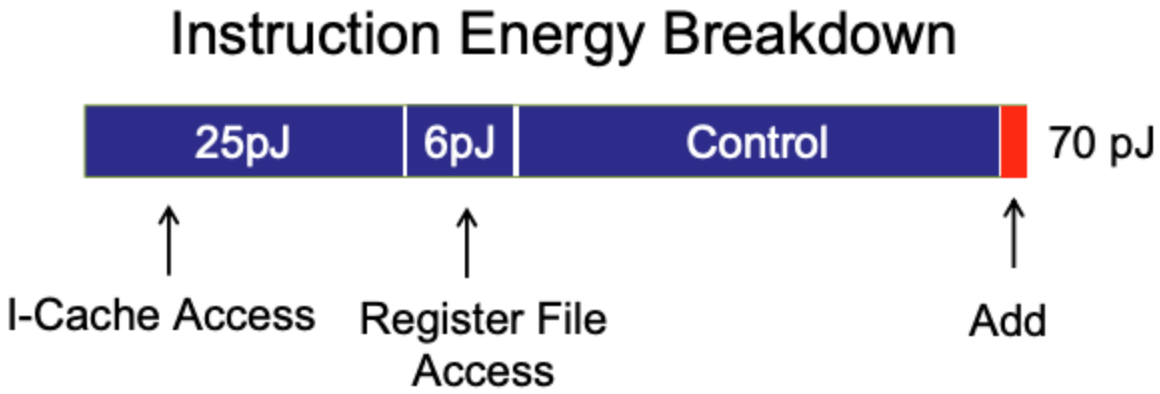
“Conservation cores: Reducing the energy of mature computations”, Venkatesh+ ASPLOS’10

Mark Horowitz, Stanford
ISSCC'14 keynote

CPUs are very inefficient

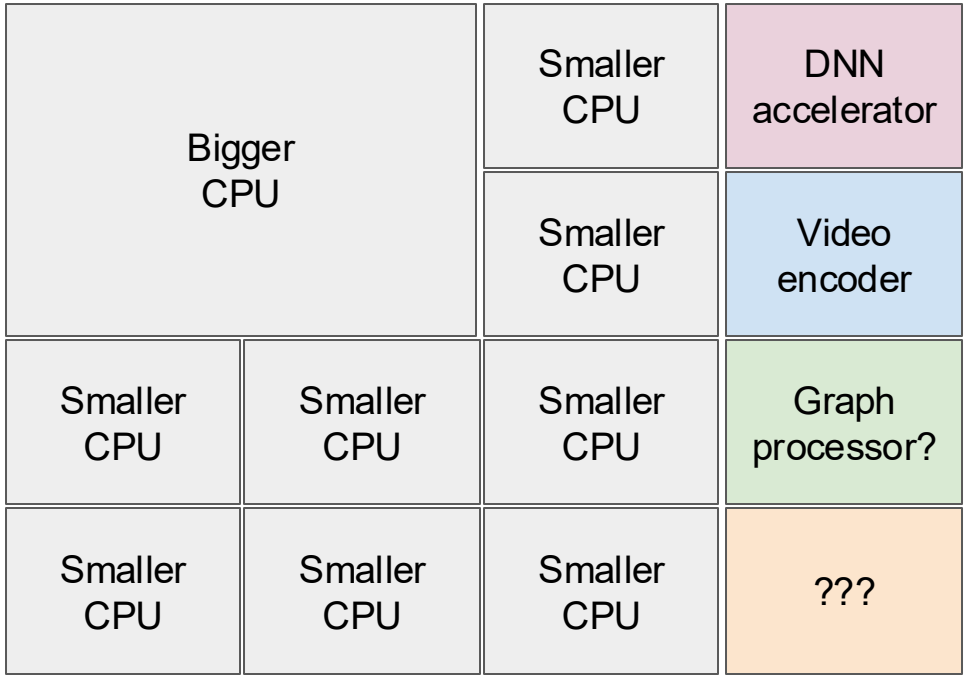
CPU spends ≈1% energy on useful work

⇒ 100× opportunity!



Cartoon of a system on a chip (SoC)

Replaces some small CPUs in heterogeneous multicore with specialized accelerators that are ~100x more energy-efficient

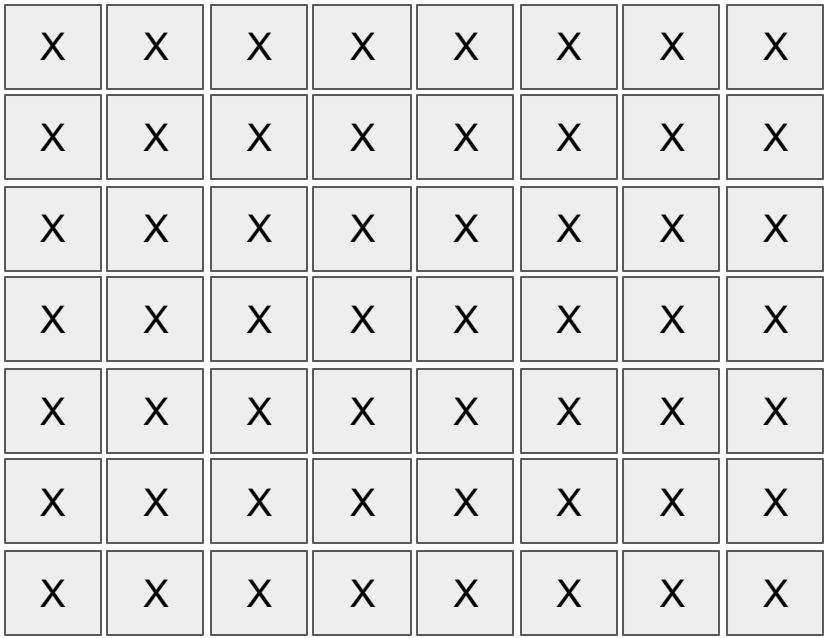


Under the hood: Basics of accelerator design

Goal: Spend all energy on useful work

Replicate minimal, custom circuit many times

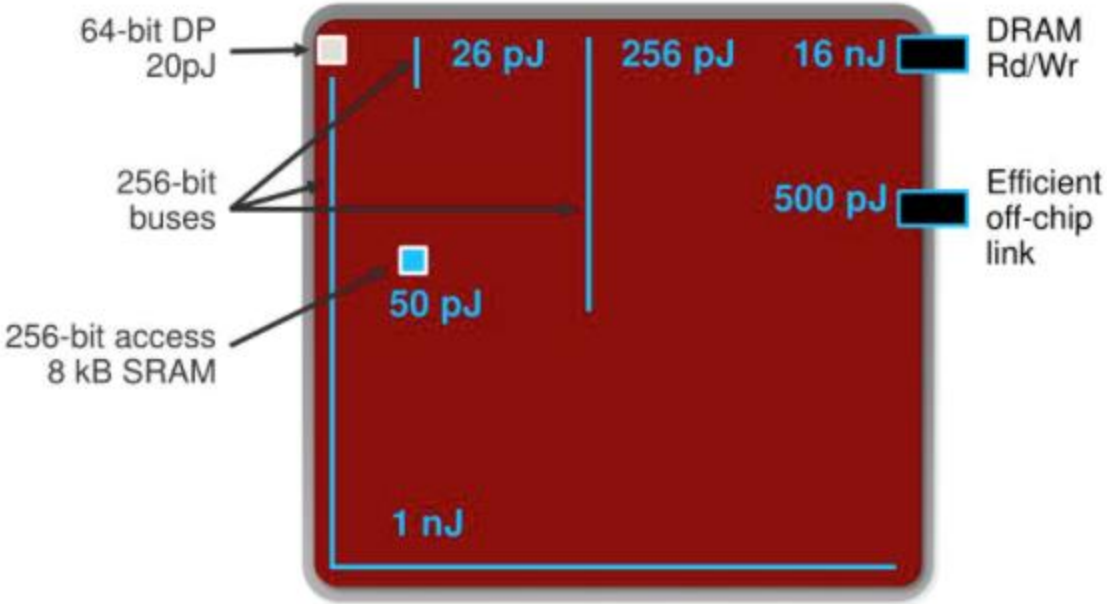
- Maximize parallelism
- Eliminate overhead
- Back to ‘optimistic’ scaling regime



Bill Dally, Nvidia/Stanford
SC'15 keynote

Accelerator design: Data movement

With CPU inefficiency removed, *data movement* consumes most energy



Bill Dally, Nvidia/Stanford
CACM'20

Accelerators as algorithm design

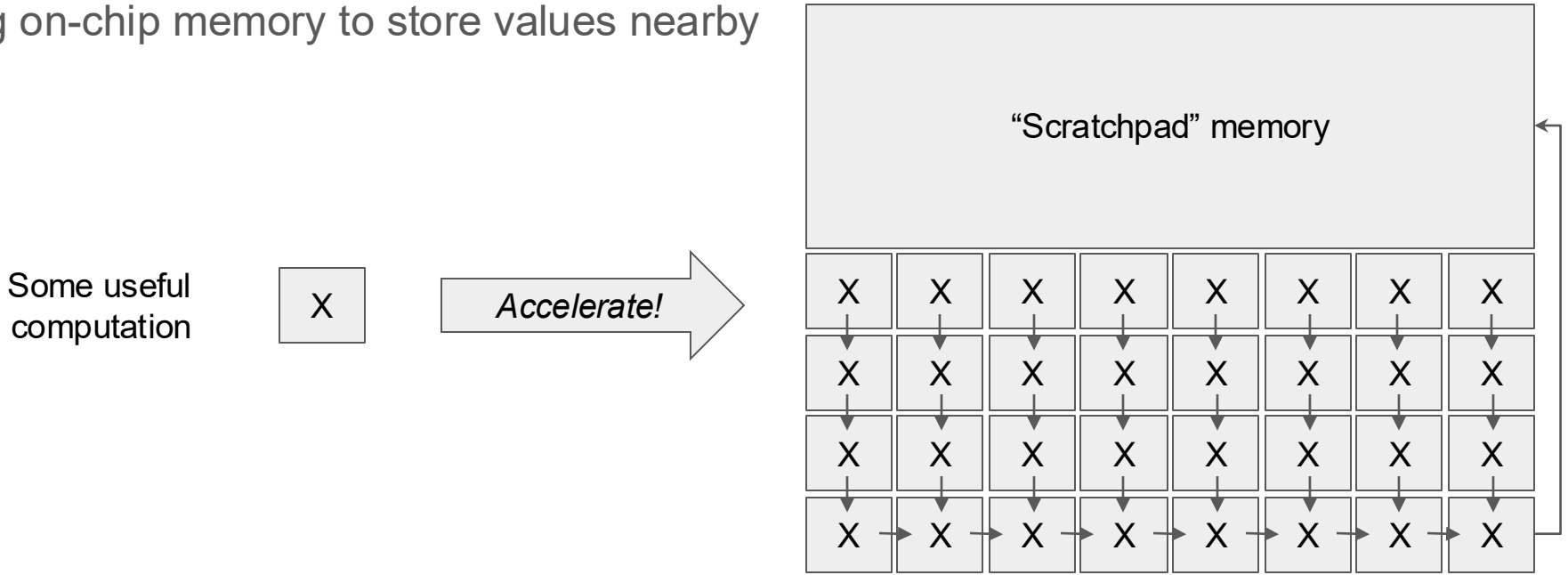
"Achieving high speedups and gains in efficiency from specialized hardware usually requires modifying the underlying algorithm…

"**When logic is free, memory dominates.** Because the area and power of most accelerators are memory dominated, a reasonable first estimate of these costs can be made by considering only the memory…"

Accelerator design: Data movement

Data values routed directly from one operation to another – no register file!

Big on-chip memory to store values nearby

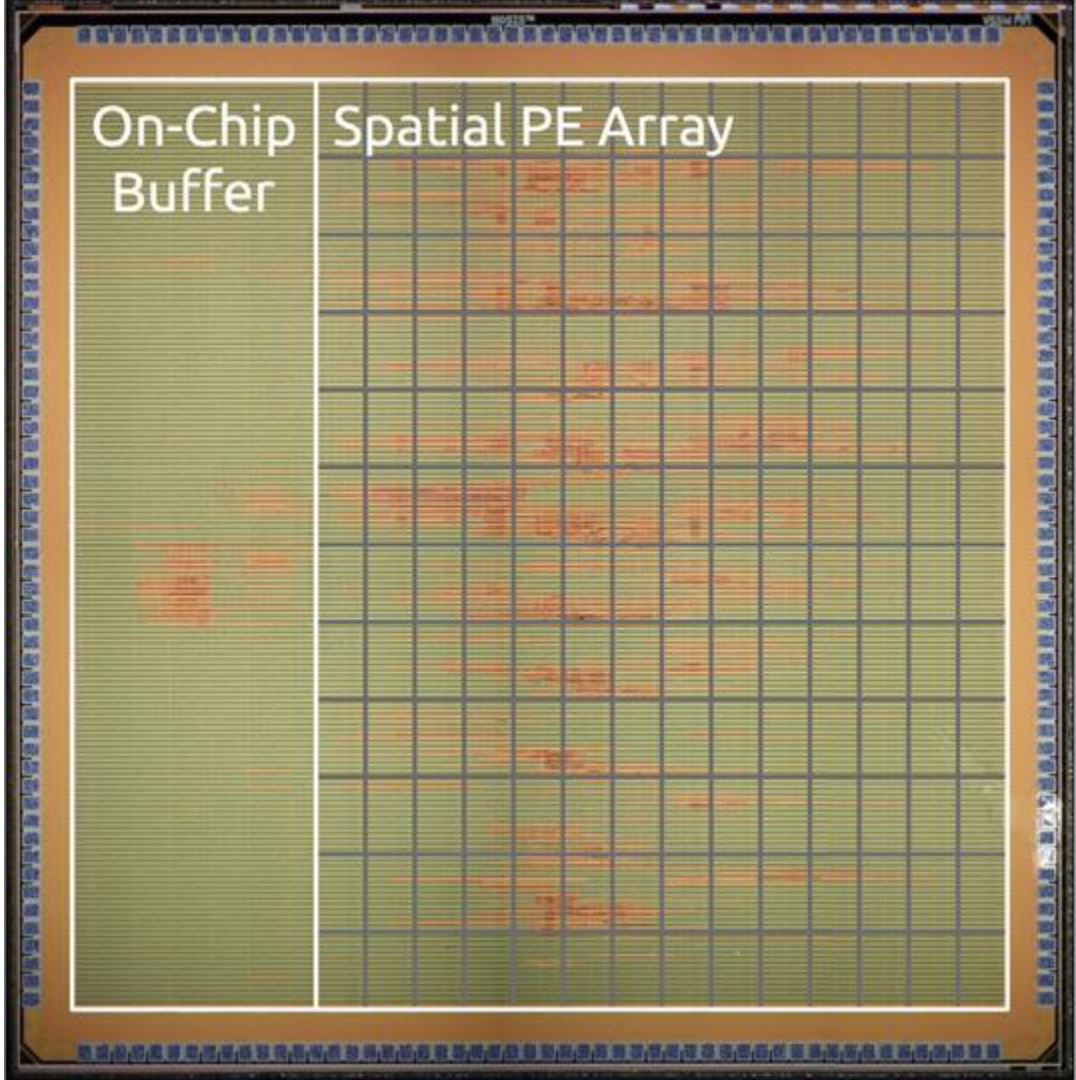


Eyeriss

Groundbreaking CNN accelerator

- Big on-chip memory
- Custom on-chip network topology for CNNs

Yu-Hsin Chen (MIT), Tushar Krishna (MIT/GTech), Joel Emer (MIT/Nvidia), Vivienne Sze (MIT) at ISCA'16



Challenges and open problems

Challenges and open problems

Amdahl's Law

- To scale, ≥99.9% of the program must run on the accelerator (deja vu…)
- How many computations are worth accelerating anyway?

Programming

- How to code & compile to accelerators?
- Is there any hope for programmability?

Amdahl’s Law in the Accelerator Era

All the same problems as multicore, plus…

How many accelerators do we need?

- Rapidly diminishing returns
- Large up-front engineering costs

Are accelerators a scaling strategy or a one-off boost?

Programmability vs specialization

How do you write programs for an accelerator?

- OS challenge: How to publish what accelerators are available?
- PL/compiler challenge: How to map code onto an accelerator?

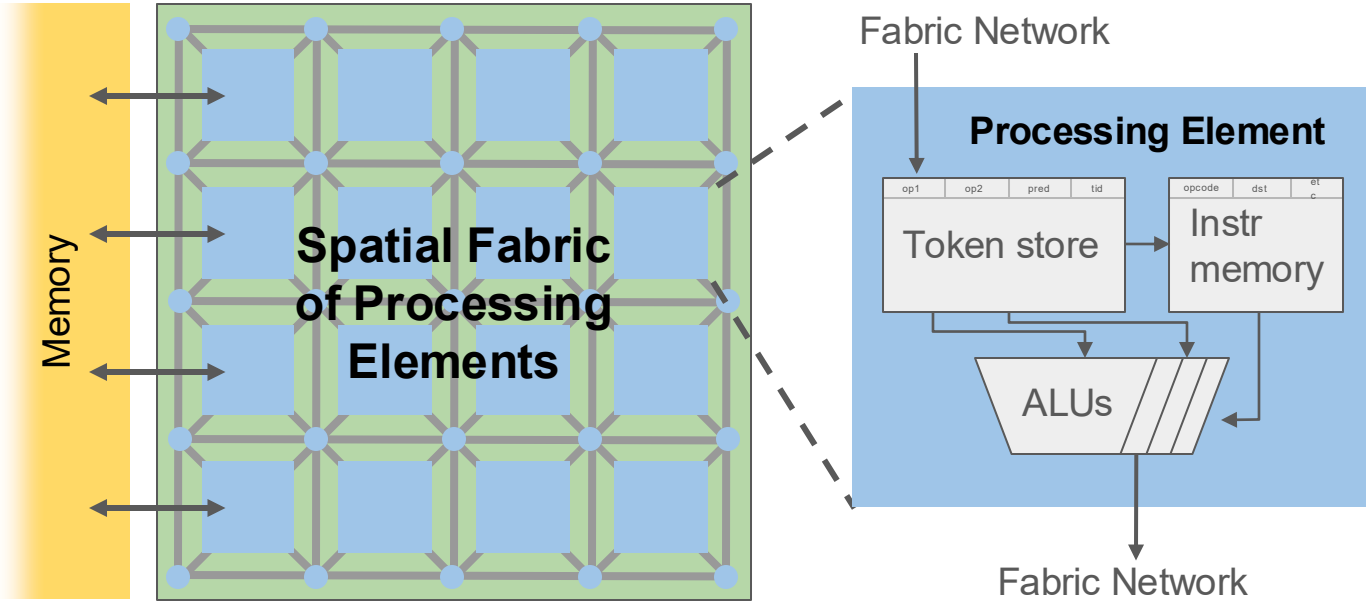
Common approach: Domain-specific languages & frameworks

- Restrict programmers to high-level operations that the framework understands
- Problem: Non-library code becomes Amdahl bottleneck
- Problem: What if an important operation is missing?

General-purpose acceleration?

Rising interest in programmable, non-von Neumann architectures

E.g., coarse-grained reconfigurable arrays (CGRAs)



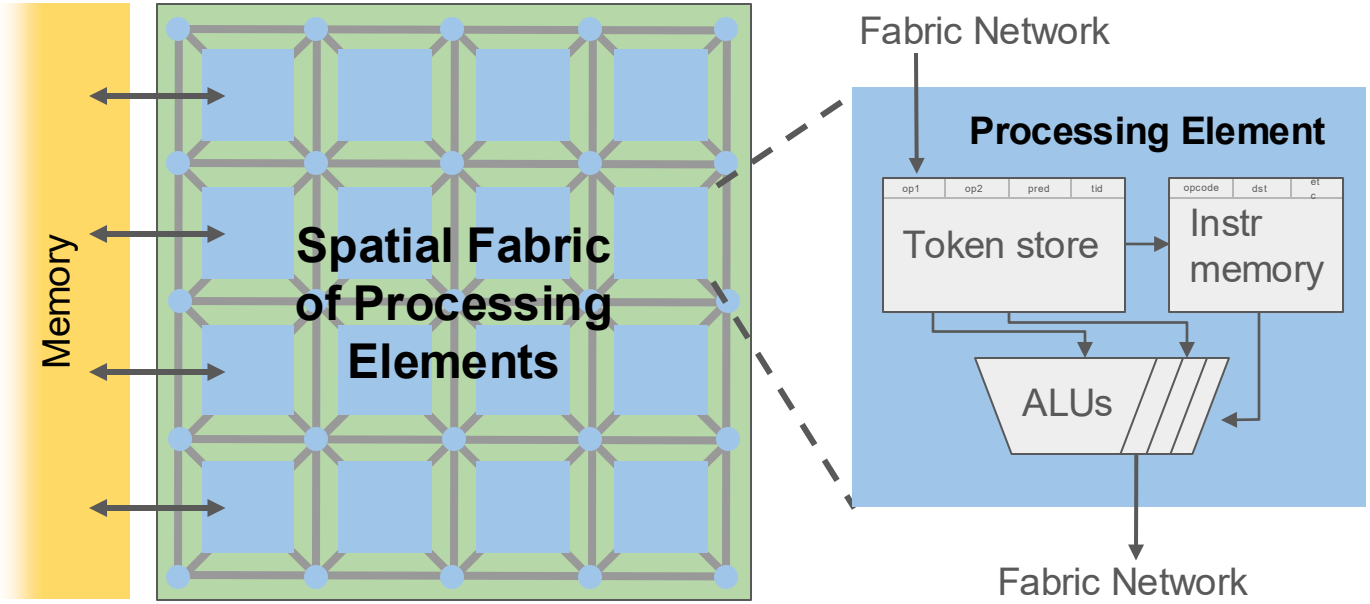
Processor consists of replicated “PEs” (necessary to get efficient scaling)

Compiler maps instructions to PEs, and they stay there!

- Directly encodes dataflow graph
- Can compile from high-level code; much simpler synthesis problem

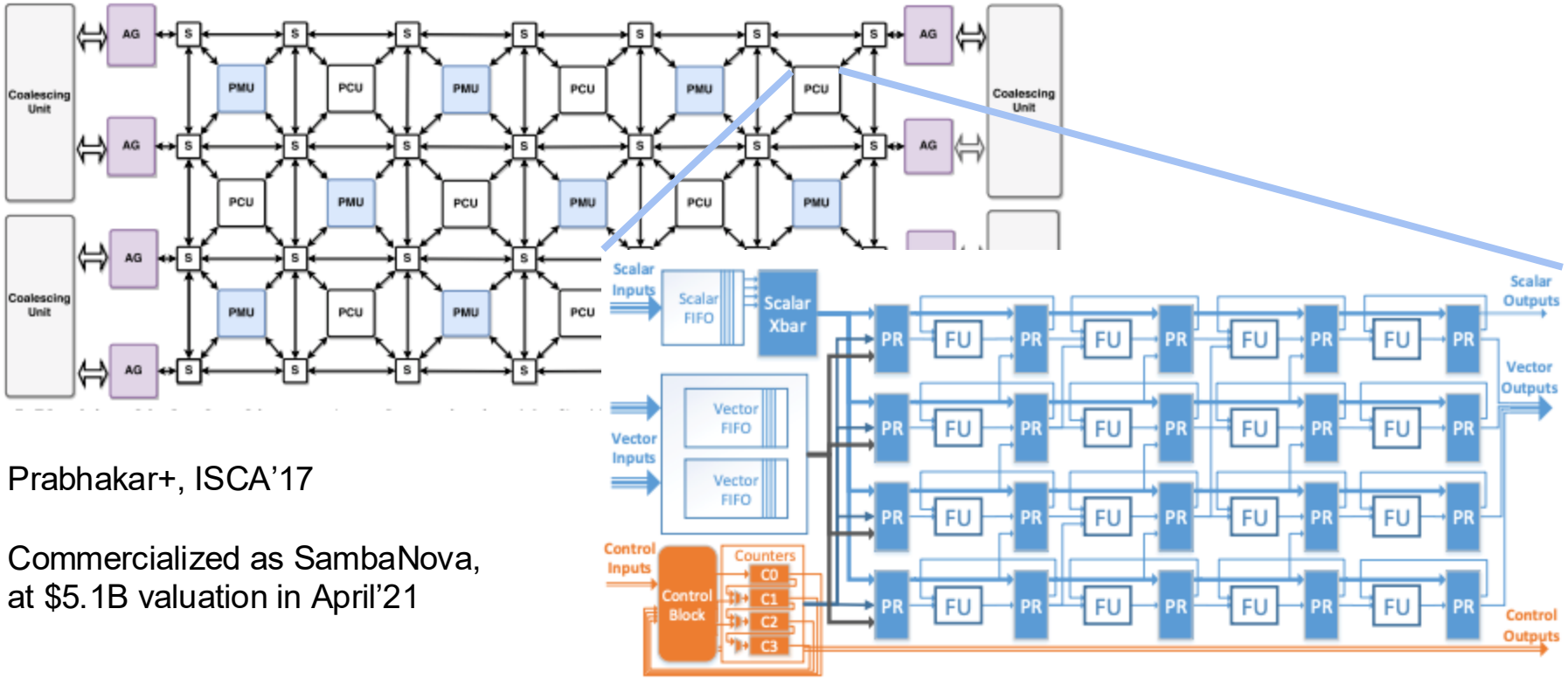
Old idea:

TRIPS, Garp, Piperench, Dyser, etc



Kunle Olukotun, Stanford

Example CGRA: Plasticine



Prabhakar+, ISCA'17

Commercialized as SambaNova, at $5.1B valuation in April'21