LECTURE 6 PHYD57

- Exponential growth of our technical civilization (1974-2004) and the recent limitations and changes: Moore's law
- Physics as a reason for the end of the glorious path of ICs: Dennard's law
- Cuncurrent execution and Amdahl's law
- Programming and parallelism: C, Fortran, OpenMP

- Simple benchmark loop (simpler-nb-3da and aa) C, F95 heap vs. stack variables, comp.-bound and comm.-bound programs; timing and benchmarking programs

Literature: see links on our course home page, the references page, and the coding page available from the course page.

TECHNICAL CIVILIZATION of the 20th / 21st century

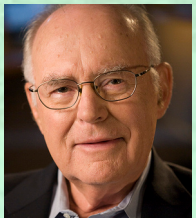
The physics of transistors over the last half-century allowed a complete and unprecedented transformation of the way we store, process, transfer and use information in modern society.

Hardware was historically ahead of software.
Sometimes, vary rarely, our bad programming is actually dangerous (we lost to it space probes; two B737 MAX airplanes crashed recently) but these are exceptions.

Our bicycles, cars, ships & airplanes do not travel twice as fast every year...
(estimate how fast we'd travel if they did!)

But soon only bicycles may do well without the help of computers, although even that is changing. There are computer-equipped gear changers on modern bikes :-|

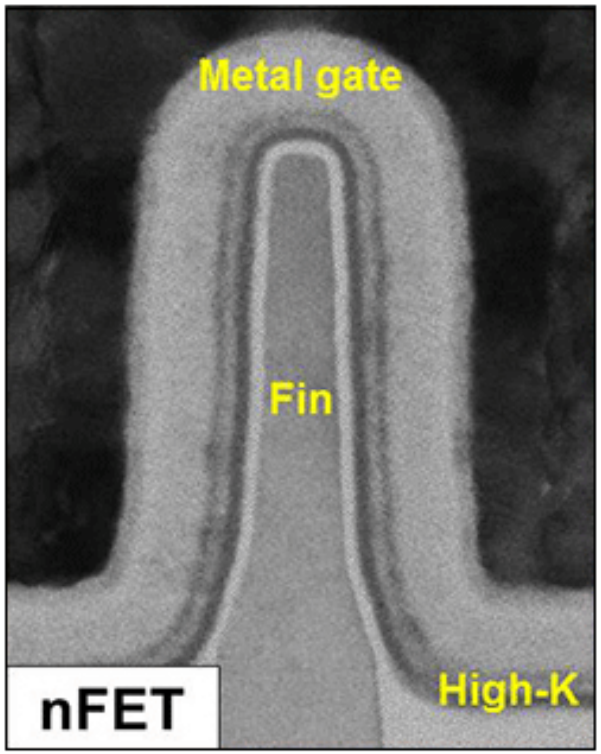
An empirical observation made in 1964 by Fairchild Semiconductor & Intel Corp. co-founder Gordon Moore spelled out the past rate of development of computing hardware from 1959. It deals with # of transistors per unit surface of a microchip. Similar laws described similar, exponential, growth of bandwidth and other aspects of computing.

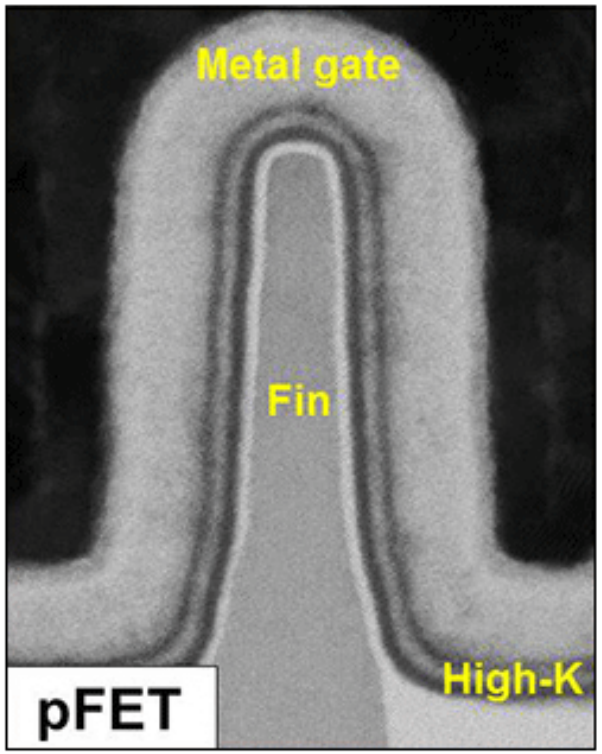


https://www.cs.utexas.edu/~fussell/courses/cs352h/papers/moore.pdf

This is how individual electronic elements look inside an Integr. Circuit (IC)
Their scale was deceasing by a factor of
$S=\sqrt{2}$
in each new generation if ICs, introduced roughly every 2 years.

The name like “100 nm” or “22 nm” or “14 nm technology” is related to the width of conductors. The spacing between transistors is ~10 times bigger than the indicated number.





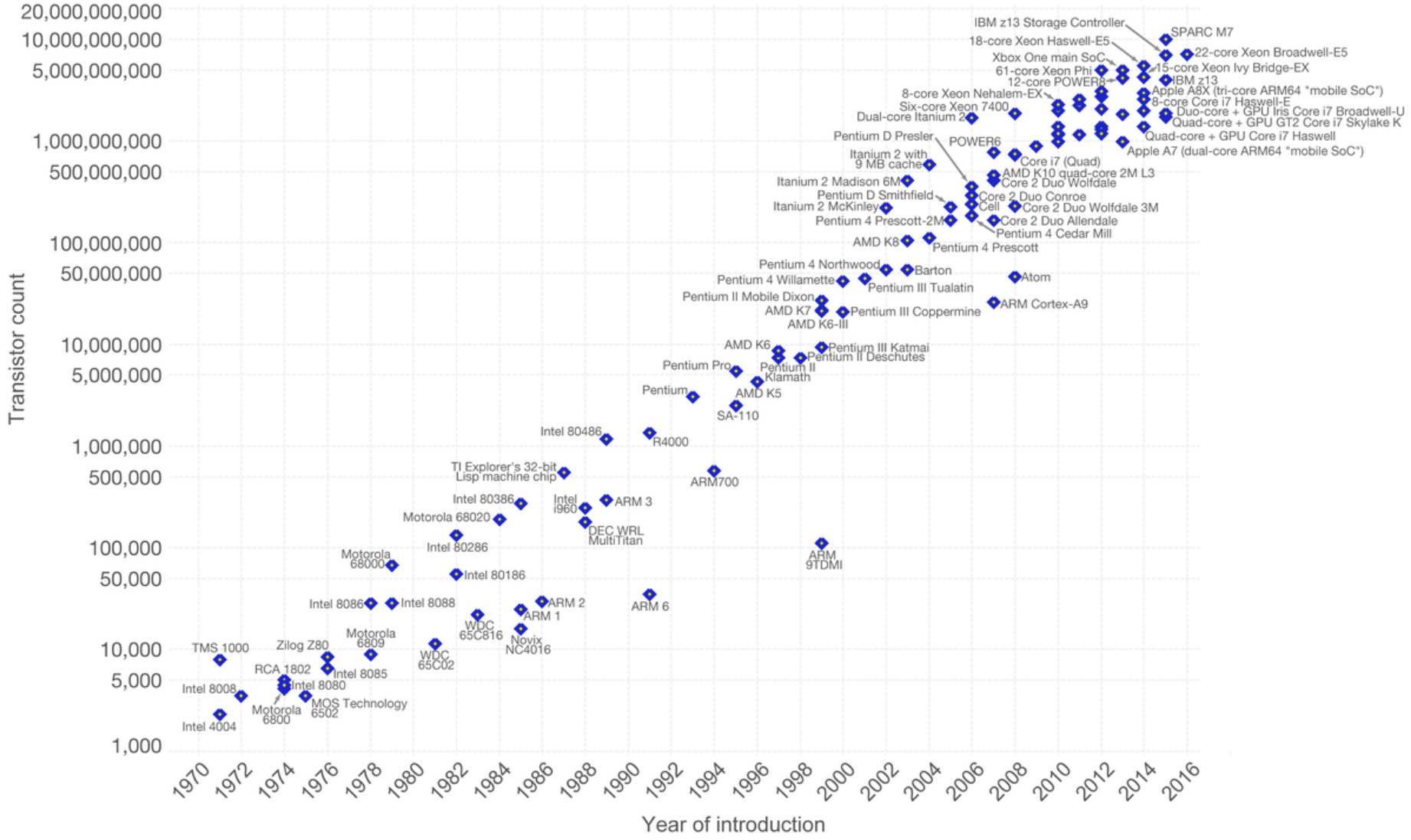
MOSFET = Metal-Oxide Semiconductor Field-Effect Transistor

technology also called CMOS = Complementary Metal-Oxide Semicond.

A self-fulfilling prophecy of the chip industry

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

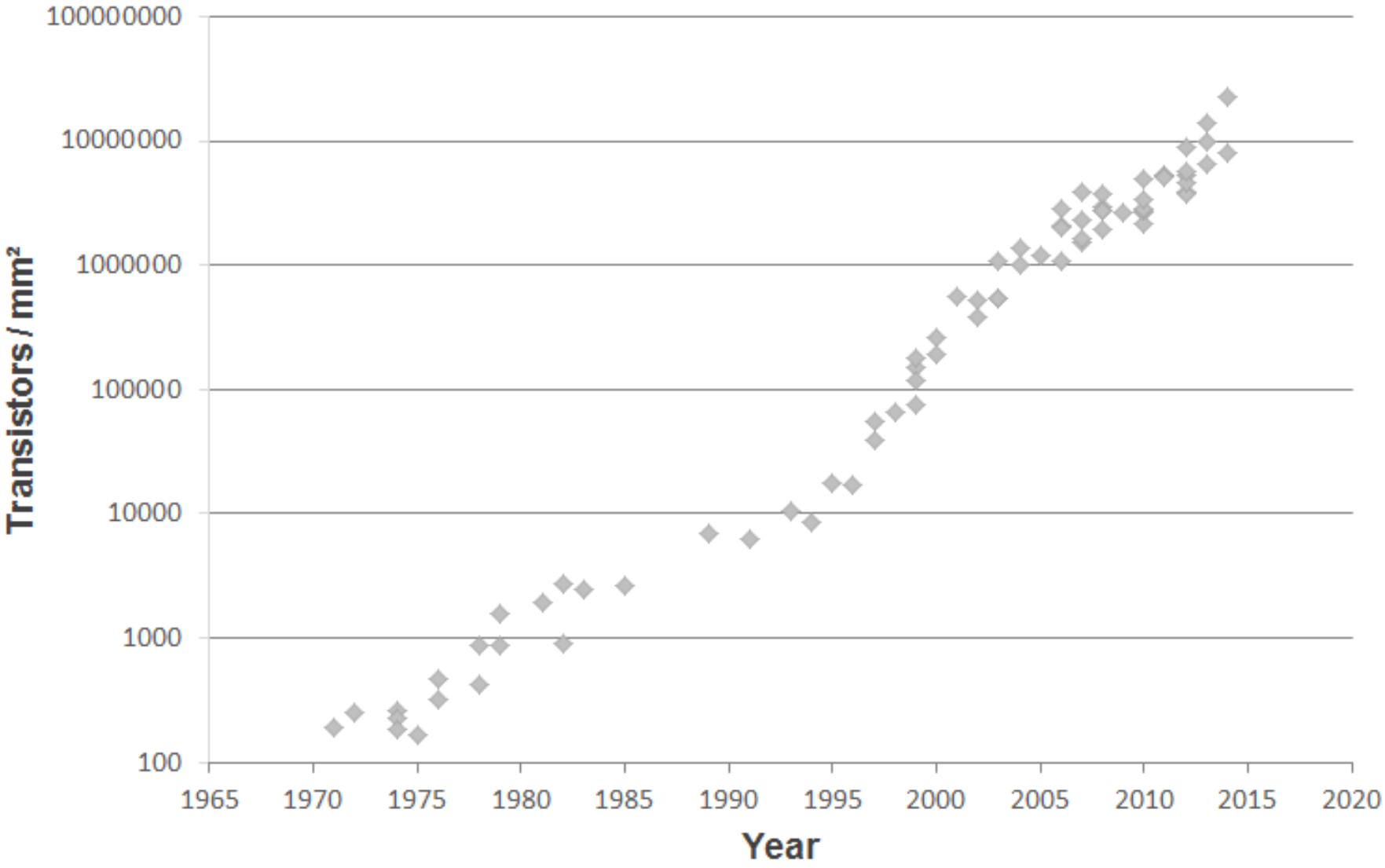
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



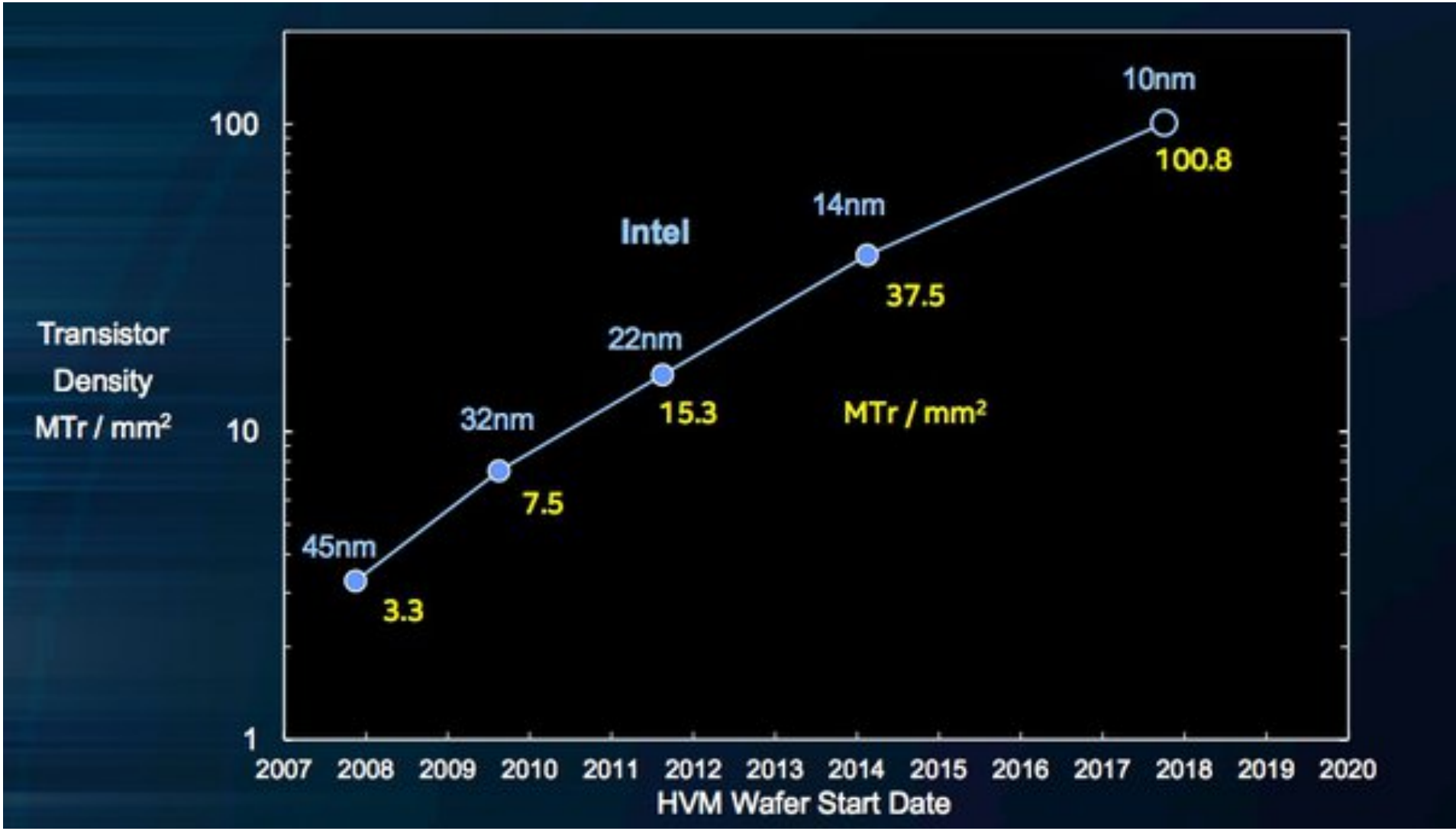
Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Moore's law: Transistor density grew exponentially.
It used to double every 2 years, partly a self-fulfilling prophecy of the chip industry.

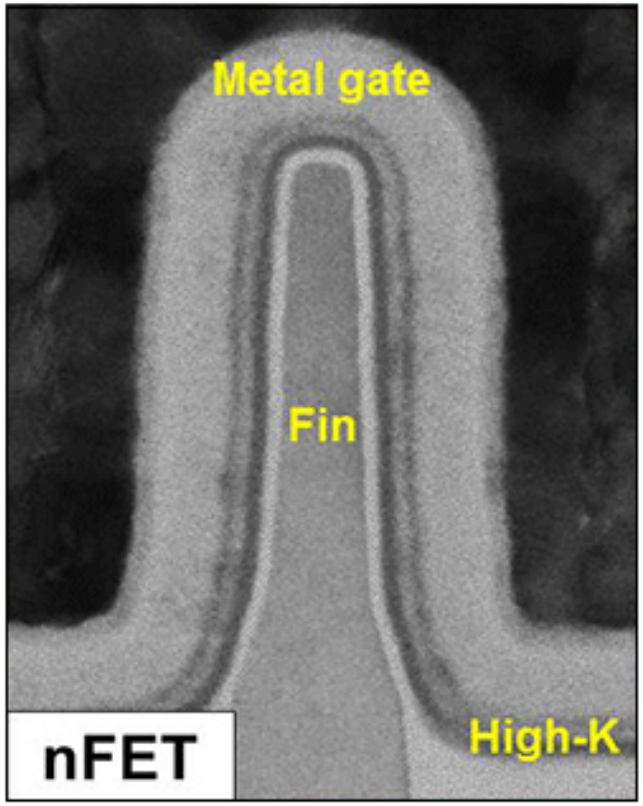


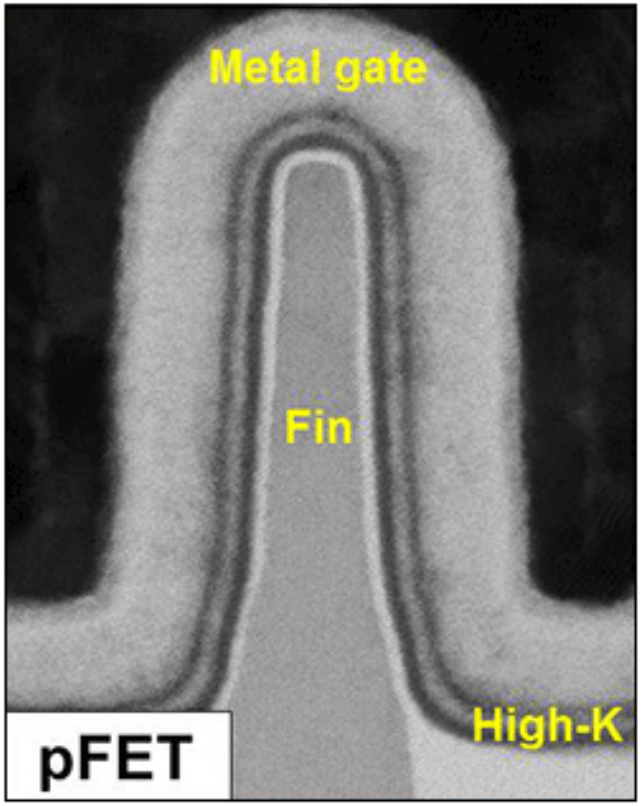
Moore's law: Transistor density grew exponentially.
It used to double every 2 years, partly a self-fulfilling prophecy of the chip industry.



This is how individual electronic elements look.
Their scale was deceasing by a factor of
$S=\sqrt{2}$
in each new generation if ICs, introduced roughly every 2 years.

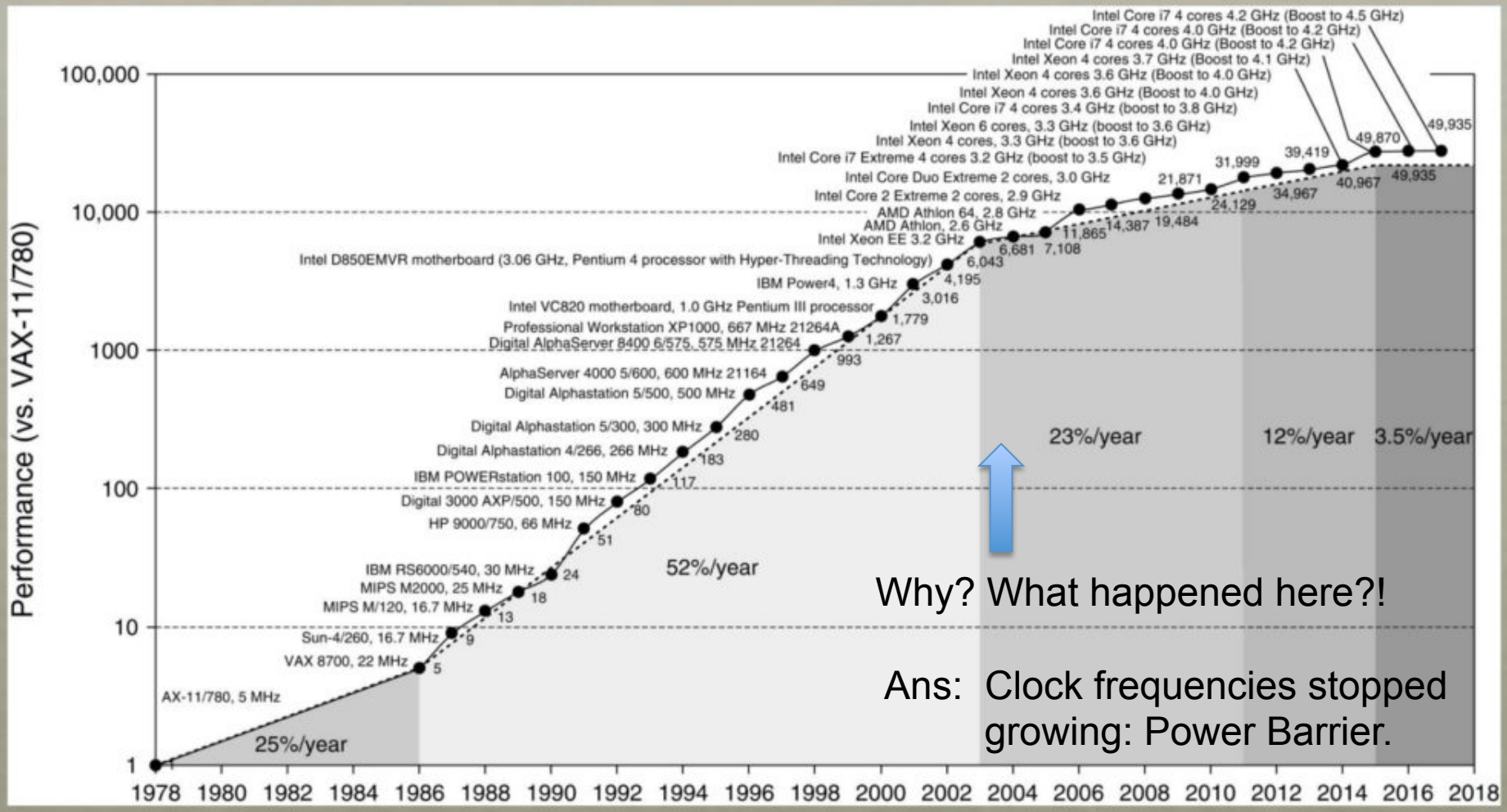
The name like “100 nm” or “22 nm” or “14 nm technology was related to the width of conductors. The spacing between transistors was ~10 times bigger than the indicated number.





MOSFET = Metal-Oxide Semiconductor Field-Effect Transistor

Uniprocessor Performance (Single Core)

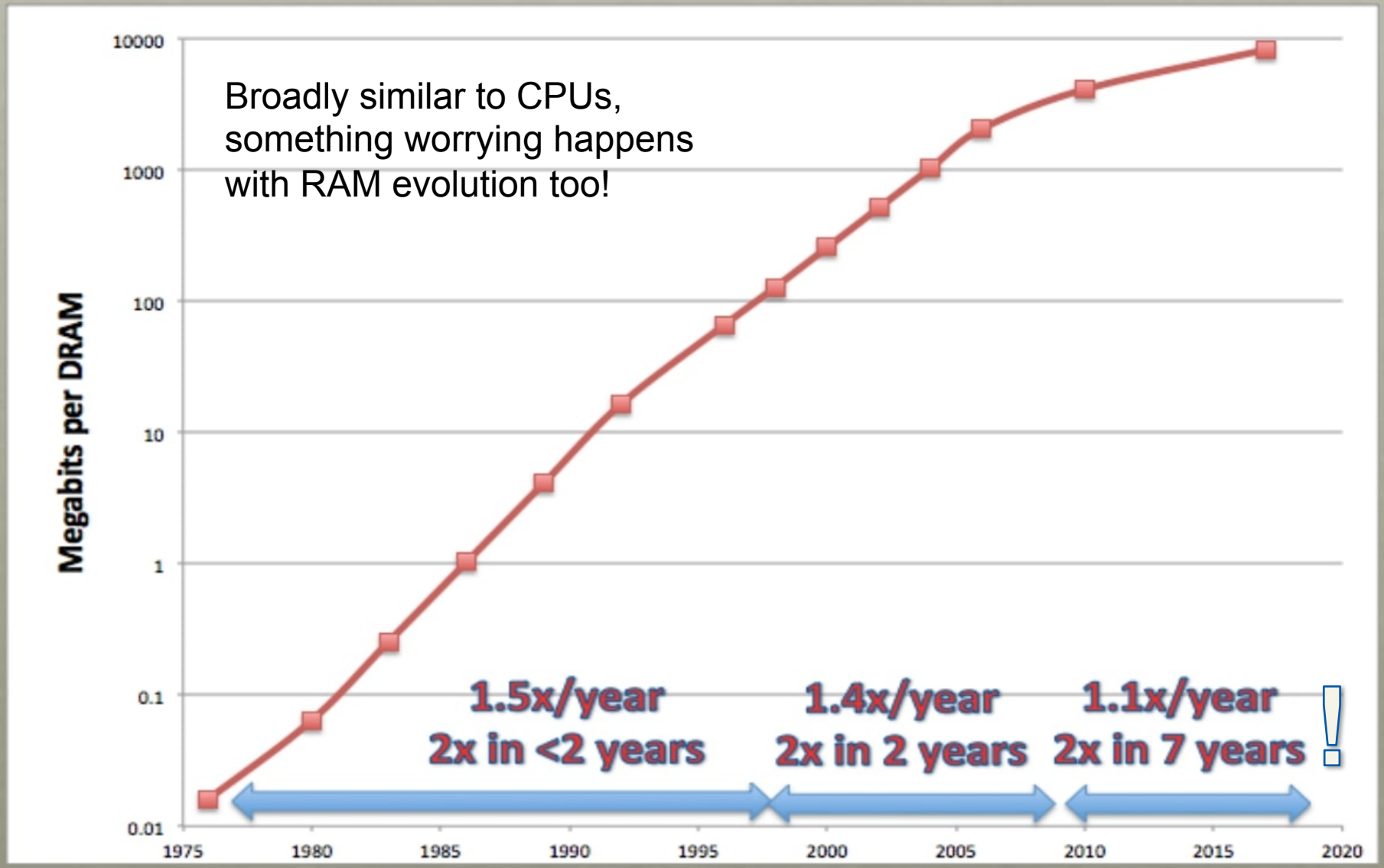


Why? What happened here?!

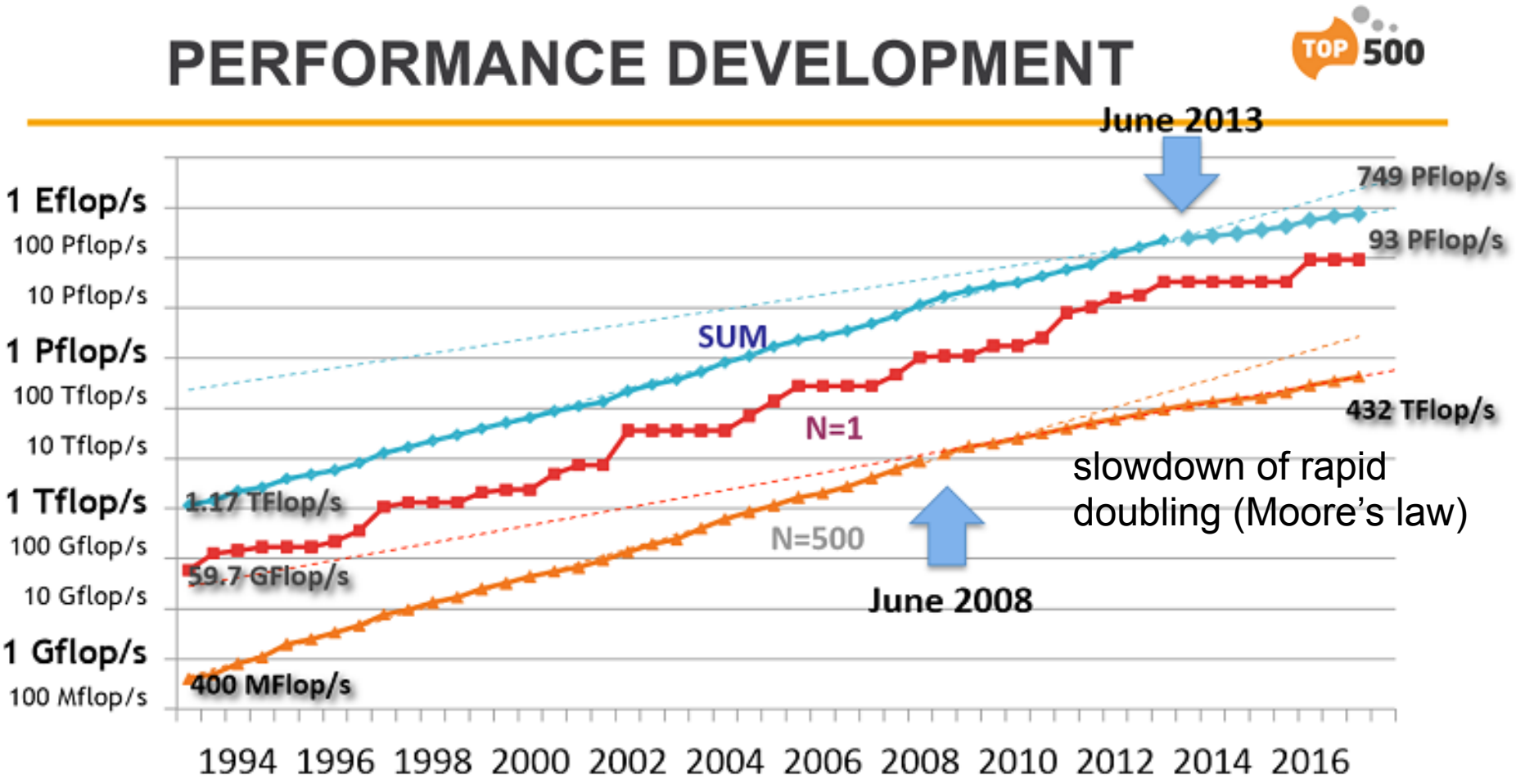
Ans: Clock frequencies stopped growing: Power Barrier.

Moore's Law in DRAMs (memory)

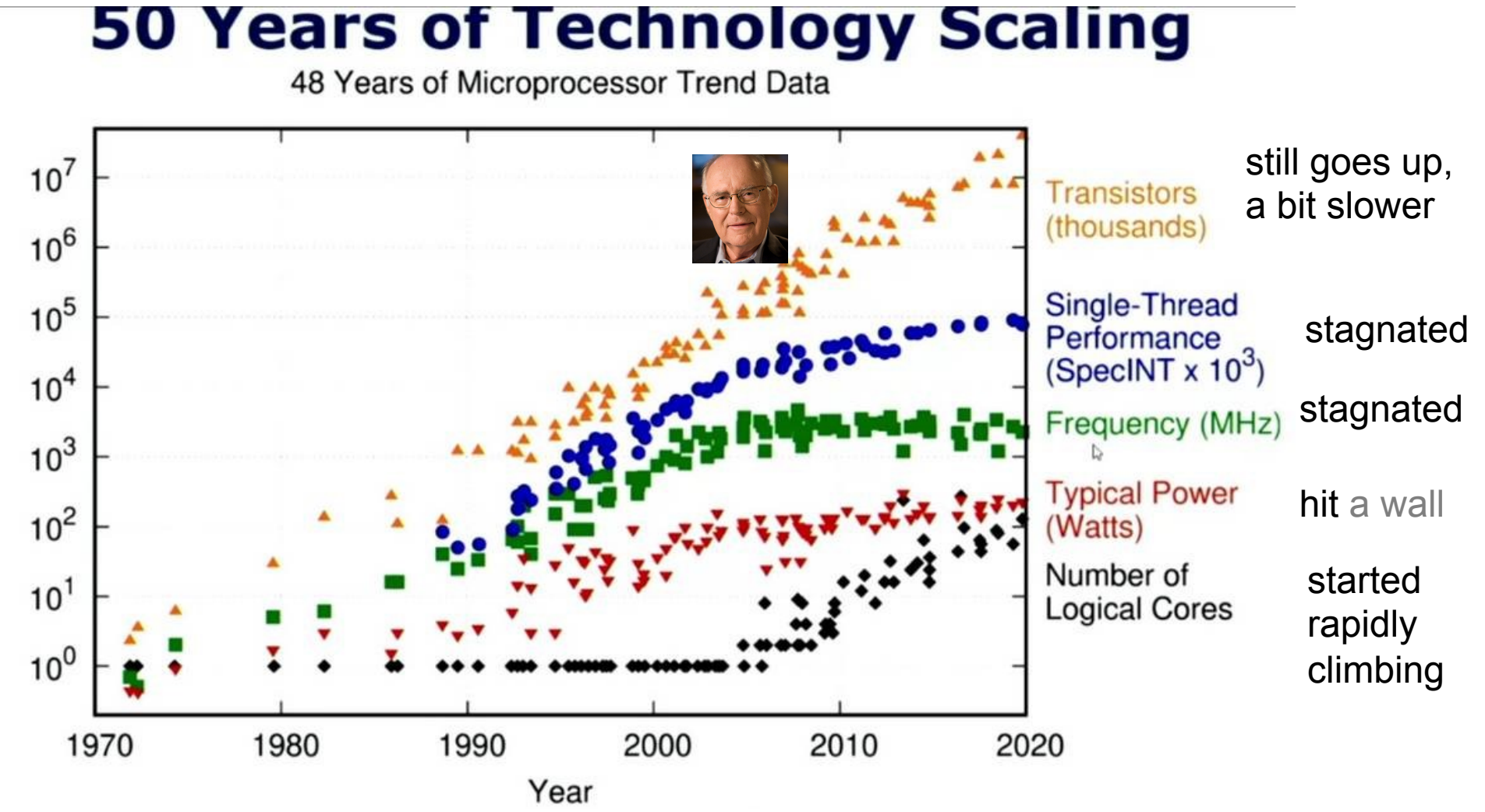
https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html = a DARPA talk



Top 500 supercomputers in the world since ~2008 grow in speed slower than before



after ~2004:



still goes up, a bit slower

stagnated

stagnated

hit a wall

started rapidly climbing

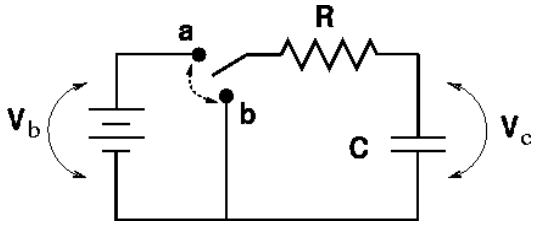
Moore's law says that the number of elements (logical gates, transistors) on an integrated circuit (on 1 cm^2 area), grows exponentially with time, such that it doubles every ~2 years. It was an empirical observation.

This was not a coincidence, but a combination of:

(i) repeated shrinkage cycles of circuitry etched onto silicon wafers, &

(ii) physics of materials, which after the shrinkage allowed to keep the *same amount* of electric power to the increasing number of logical gates

Let's prove it using physics of RC circuits.



A transistor on an Integrated Circuit has capacitance **C** and some stored charge Q. It also has resistance R, on which the current's energy can be dissipated. Transistors discharge and re-charge, under voltage **V**.

The process of charging/discharging is not for free; it dissipates energy equal to $\Delta E = CV^2$, half of it during charging, which is like pushing electrons up the growing potential hill. Energy drawn from voltage source by charge Q is $\sim QV = (CV)V = CV^2$. Usually all of that energy must be supplied but is wasted as heat during discharge through resistance R, however ΔE does not depend on R.

If the chip works with clock frequency **f**, then the rate of energy dissipation (power) is

$P \sim N C V^2 f$ + N*(leakage current*V) *Dennard's law*

where **N** is the number of transistors.

Now let's **shrink** the circuitry on an IC. Each side of the chip is constant but the pattern of circuits is shrunk by a factor $\sqrt{2}$ **=1.4128..** on a side, as it was in fact done every 2 years or so for 4 or 5 decades.

Q: What is happening to quantities C, V, and f of a transistor, and their number N?

N grows by a factor 2

[since there are 1.41 more transistors along each side of the chip

C grows by a factor $1/\sqrt{2}$ ~ 0.707

[this follows from electrostatics; C ~ area of capacitor/spacing between electrodes]

V was being increased by a factor $1/\sqrt{2}$

[i.e. decreased by 1.41..]

f was increased by a factor $\sqrt{2}$

[because a smaller transistor can switch its state proportionally faster]

We have estimated that power needed to do calculations scales like $P \sim N\ C\ V^2\ f$,

which grows by a factor of: $2\ [1/\sqrt{2}]\ [1/\sqrt{2}]^2\ \sqrt{2}$ **== 1 (!)**

We have shown why 15+ cycles of shrinkage of transistors & circuits on a microchip over 4 decades did not require much more cooling + much more power supply to the chip. (Electric power requirement did in fact grow, as IC's area slowly grew.)

The exponentially growing number N was utilized to complicate the logical structure of CPUs, e.g., by introducing more and more levels of *cache memory* inside the processor. This memory is much fast but much smaller than RAM. It can be used as a buffer between CPU and the outside operational memory (RAM).

This allowed to mask the fact that CPU-RAM communication and the speed of the RAM (memory) was growing slower than the processing needs of CPU. As a result, frequency f grew ~2 times, instead of just 1.41 in each new generation of processors.

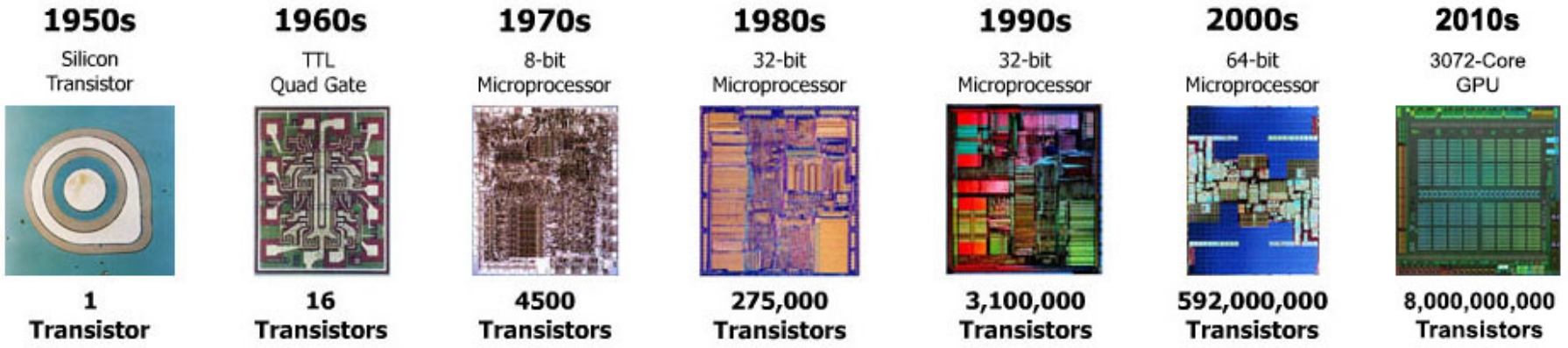
The progress *was* phenomenal. Among others, it allowed a similar exponential Internet bandwidth growth with doubling time ~18 months.

If our cars were increasing in speed at the rate equal to the increases in computing power, then we would now drive at cosmic speeds close to the speed of light, or about once around the Earth per second.

No other area of human activity saw a million-, let alone a billion-fold quantitative improvement in one human generation, but computing did.

Physics allowed us to keep providing a similar power to faster, smaller processors.

So why did the size reduction and exponential growth of clock speed have to end around 2004?



Transistors inside the microchips, now counted in billions, became so small that, due to microscopic material imperfections, electric power gets dissipated in new, unavoidable way, via the *leakage current* in volatile memory (RAM, CPU).

Transistor should either block or transmit current, but now the blocked state became not 100% blocked. Electric energy is slowly going to waste, without carrying out any computation. That's how the doped semiconductors work.
Leaks occurred before, but were masked by much larger heat dissipation due to (useful) toggling from state "0" to state "1", and from "1" to "0", during binary data processing.

Now: $\mathbf{P} \sim \mathbf{N}\,C\,V^2\,f + \mathbf{N}\,(\mathbf{I}_{\mathbf{leakage}}\mathbf{V})$ *Dennard's law*

(i) size of circuitry still diminishes (with serious manufacturing problems! ASML wins)
(ii) number of transistors in new generations of processors still grows (slower)
(iii) f ~ const. ~3-4 GHz (speed of 1 core of CPU inches forward *very* slowly)
in order to try to keep energy supply and dissipation at a reasonable level!
(iv) Cost of manufacturing next chip technology increases very rapidly, and the number of microchip foundries decreased to 2 or 3 (The 5nm technology may be the last (?))

Post-Dennard era, leakage current contributes about the same as re-charging power losses:

Let's **shrink** the circuitry as before by a factor **$\sqrt{2}$ =1.4128..**
Q: What is happening to quantities C, V, and f of a transistor, and their number N?

N still grows by a factor 2

C grows by a factor $1/\sqrt{2}$
[this follows from electrostatics; C ~ area of capacitor/spacing, as before]

V has to be kept constant (~1 Volt) 1

f increases only slightly ~1
[because a smaller transistor now incurs more power losses]

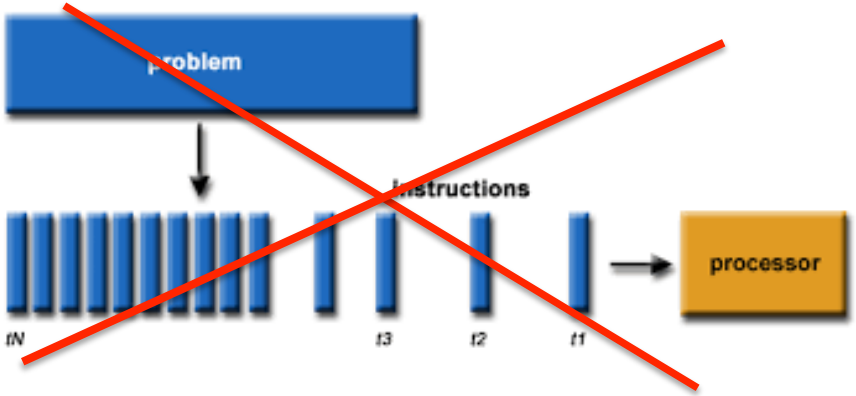
We have estimated that power needed to do calculations scales like
$P \sim N\,C\,V^2\,f \quad + N\,V\,I_{leakage}$

1st term grows by a factor $2\,[1/\sqrt{2}]\;1^2\;(\sim 1)$ **~ 1.4...1.5 (!)**

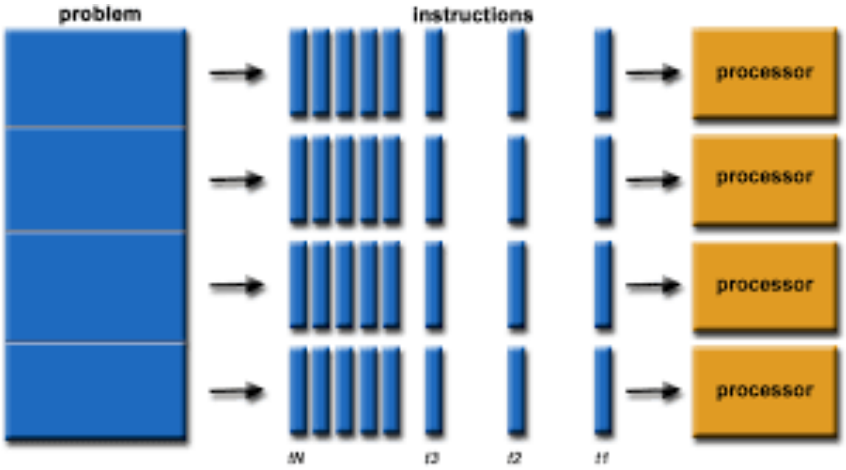
Dark silicon: switching off more and more transitors at any moment → ~1 (?)

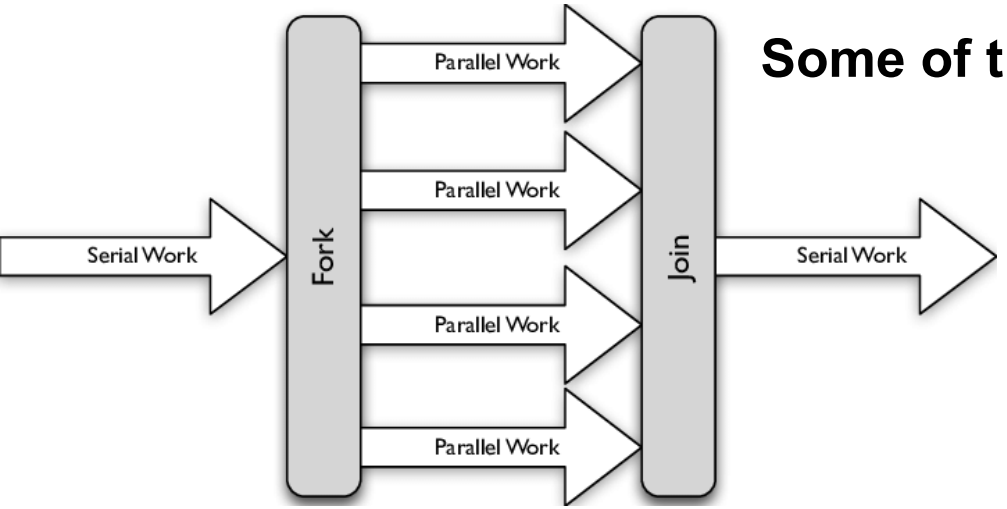
- The problems boil down to the power barrier, although there is also a rising manufacturing difficulty and cost. The wait for doubling of performance lengthens...
- It is no longer possible to play the game of “increase clock speed and keep one core”, as before. On the contrary, increasing the number of cores in a processor comes cheap (power-wise), and therefore is now practiced by manufacturers.
- We are (and will be, in the near future) getting processors with more and more cores. Each core will NOT be much more capable to crunch numbers than in the previous hardware generation. Hardware engineers can’t keep up doing miracles. They hope that **you** as a programmer will do them, adjusting to the changing world.
- Power barrier forces us to use more and more cores **in parallel**, if we hope to have practice HPC
- This task is significantly more difficult in Python than e.g. in C, Fortran, & Julia. Therefore we are going to learn parallelization and other speedup methods in this course.

serial program & execution 😐
(try to avoid!)



parallel program, concurrent execution ☺





Some of the modern programming methods:

If you find a fork in a road, take it.

'Yogi' Berra

1. One of the most successful **parallelization** paradigms is the fork-join scheme (on all platforms; Message Passing Interface = MPI, essential on supercomputers, OpenMP mostly in shared memory sys.: **multithreading** on many cores).

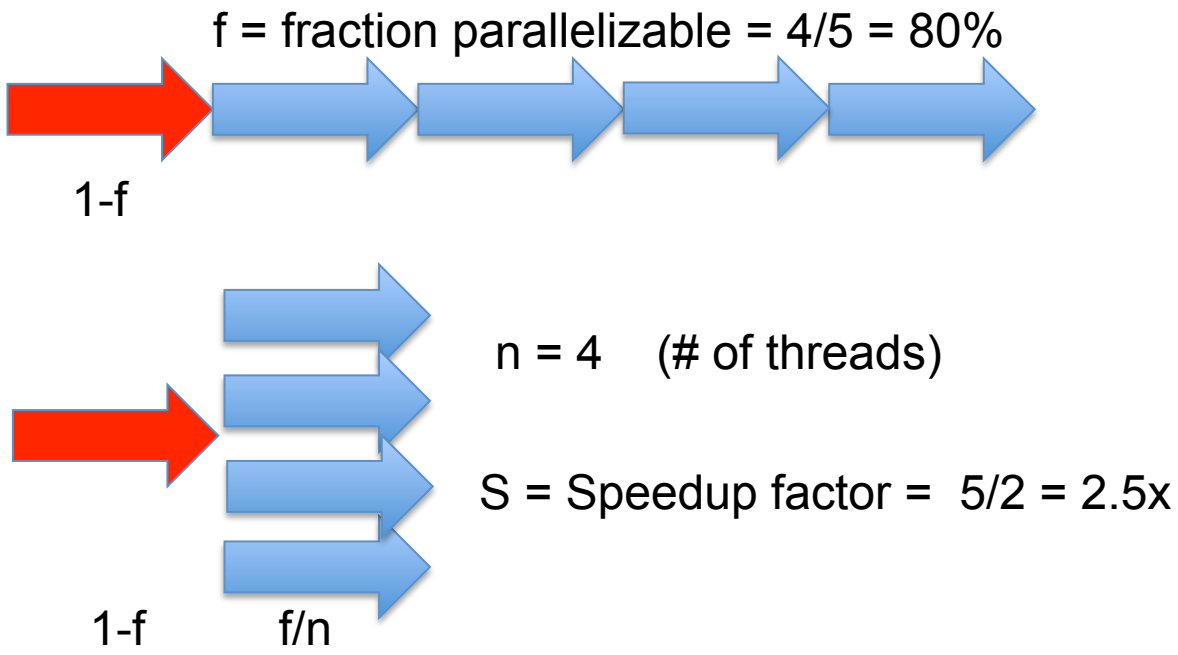
2. Every thread may, in addition, be **vectorized,** using a growing set of long AVX registers (on CPU, MIC, currently mostly 256 bit = 8 single prec. or 4 dp floats)

Vectorization:

- manual (intrinsic C/C++ functions)

- automatic (compiler + SIMD directives)

Derivation of **Amdahl's law** of parallel program's speedup



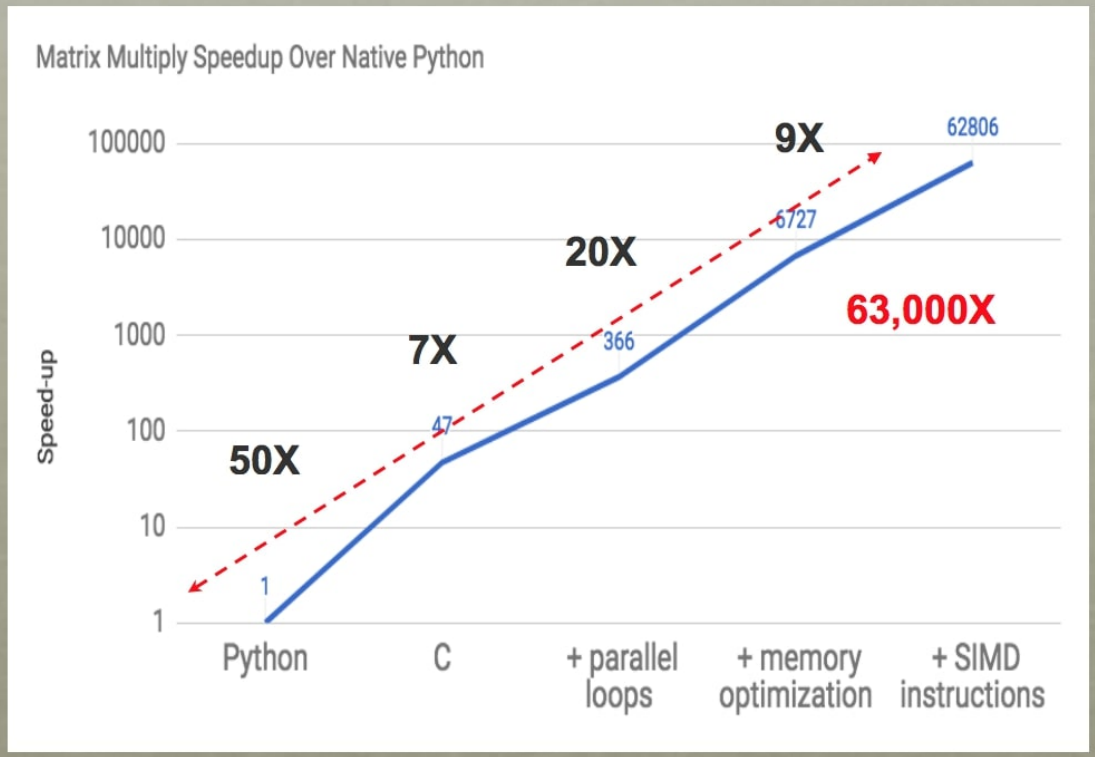
In general: $S = 1 : (1-f + f/n) < 1/(1-f)$

e.g., if $1-f = 1/20$ then $S < 20$ even if $n = 512$

We have already seen an example of 40x speedup over Python possible with Numpy, and 40x over Numpy with Fortran (OpenMP & CUDA), Here's someone's success story with the matrix multiplication algorithm, which has f~1 (is ~fully parallelizable)

WHAT'S THE OPPORTUNITY?

Matrix Multiply: relative speedup to a Python version (18 core Intel)

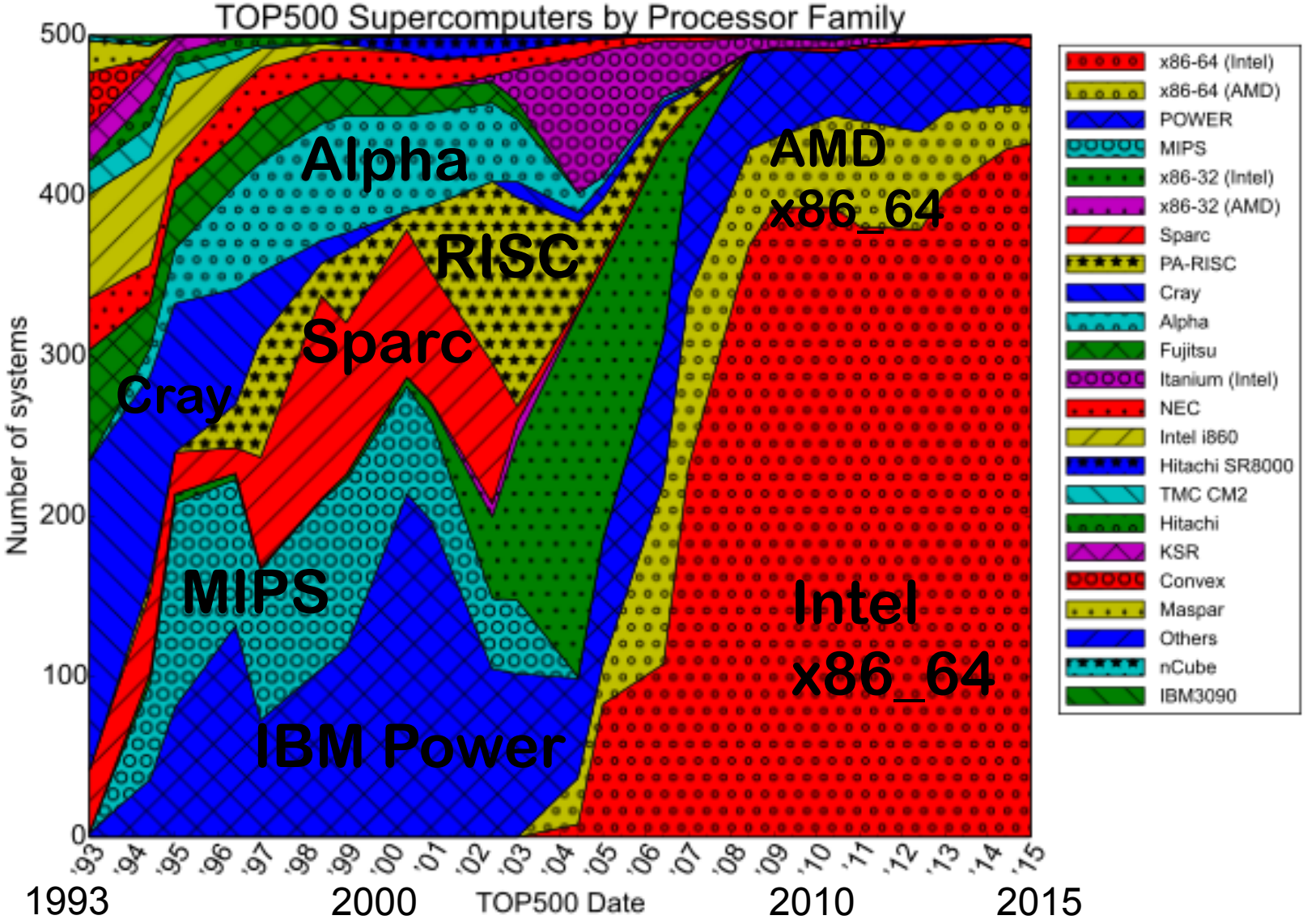


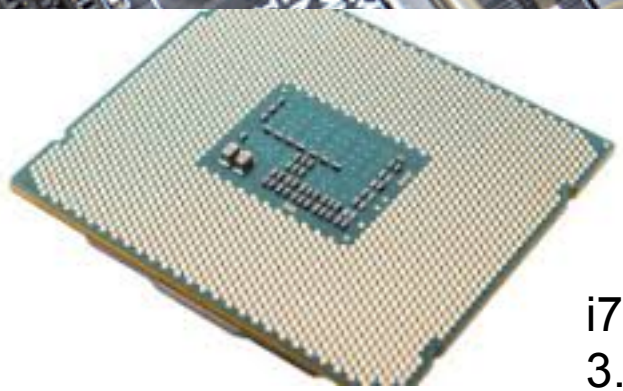
from: "There's Plenty of Room at the Top," Leiserson, et. al., *to appear*.

CLUSTERS, SUPERCOMPUTERS:
additional, final boost by their hardware's massive scale.
But these beasts should run efficient, smart codes.

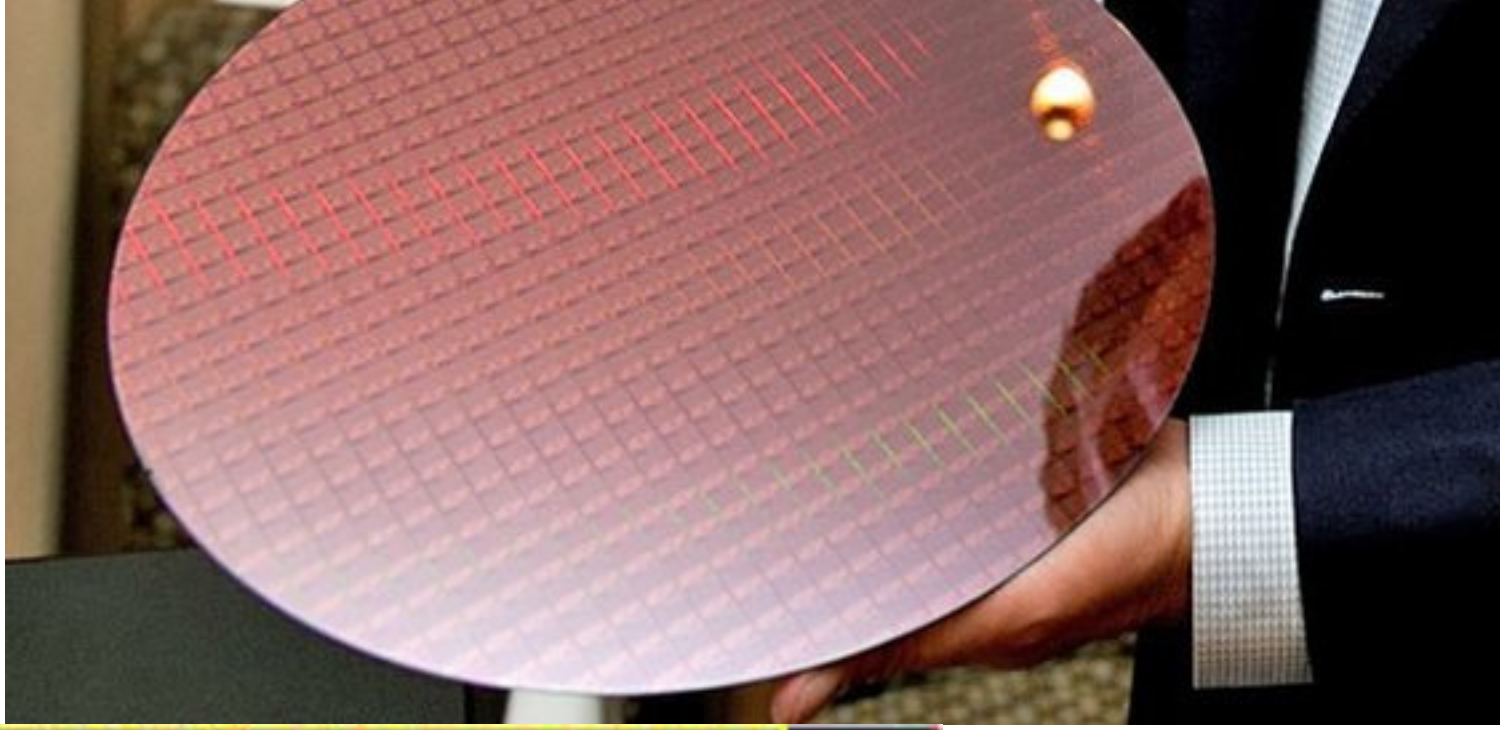
stars ~ # cores
in supercomputer

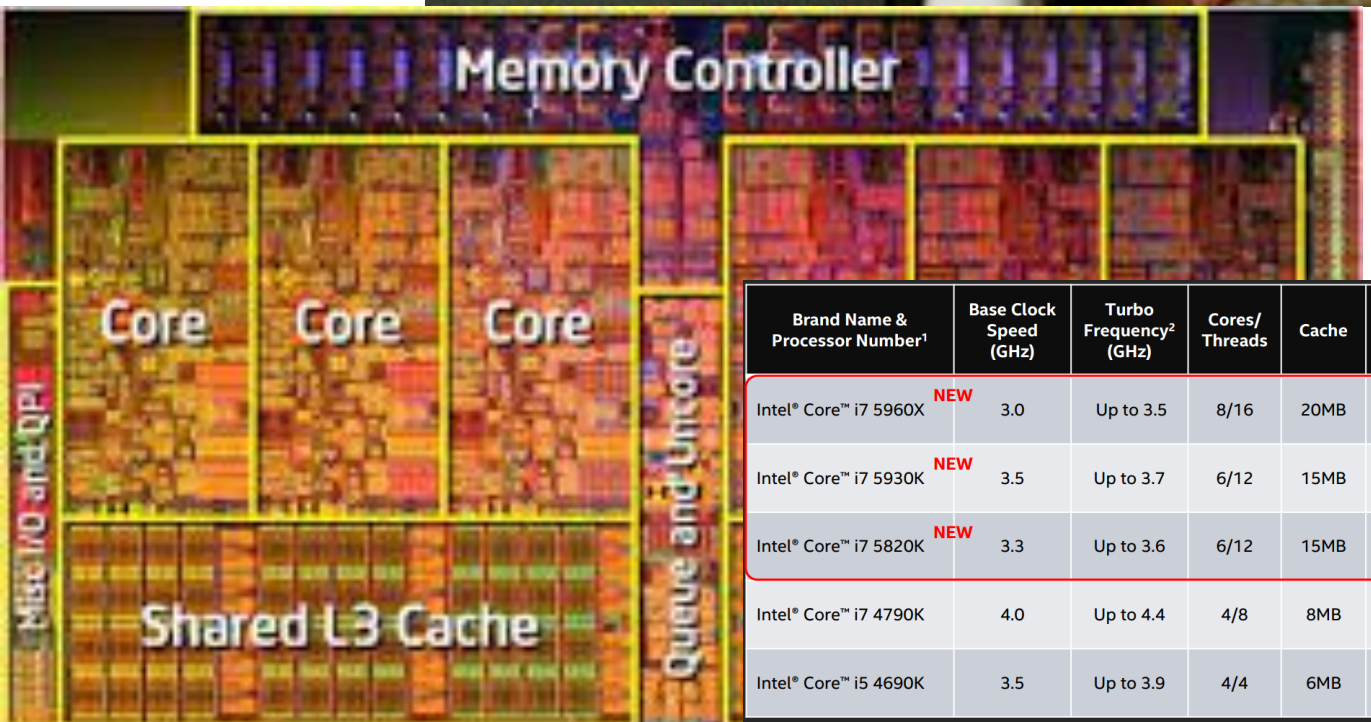
Processor architecture & instruction set: Out of creative chaos of different CPUs before ~2004, Intel's x86_64 processors emerged as a big winner. Your CPU and CPUs in supercomputers, until 2019 were likely all x86_64 microprocessors. They understand the same instruction set. Here is the full story, in 1993-2015:





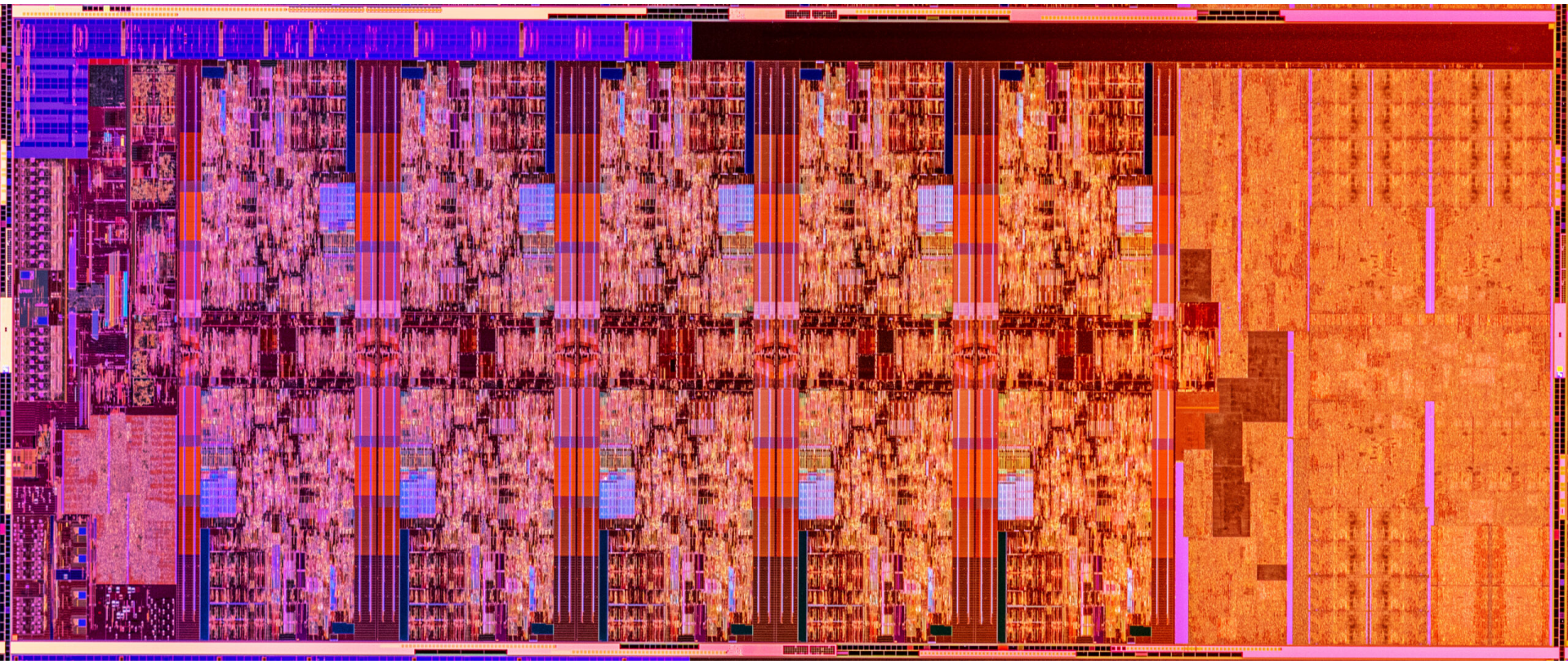
i7-5820K Haswell CPU 6 cores, 12 threads
3.3 GHz, cache L1/L2/L3 = 64k/1.5MB/15MB, 140W



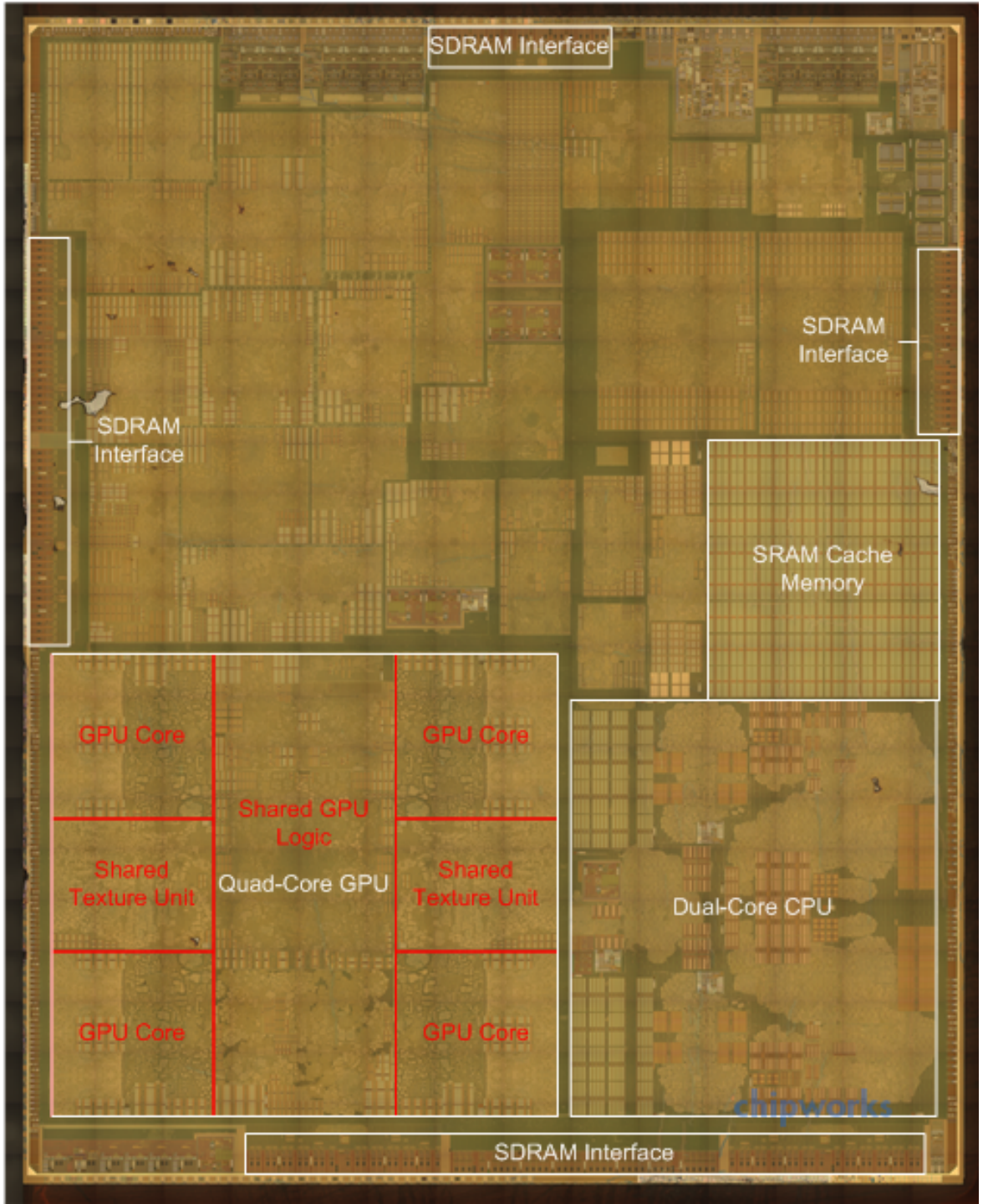


Brand Name & Processor Number[1]	Base Clock Speed (GHz)	Turbo Frequency[2] (GHz)	Cores/ Threads	Cache	PCI Express* 3.0 Lanes	Memory Support	TDP	Socket (LGA)	Pricing (1K USD)
NEW Intel® Core™ i7 5960X	3.0	Up to 3.5	8/16	20MB	40	4 channels DDR4-2133	140W	2011-v3	$999
NEW Intel® Core™ i7 5930K	3.5	Up to 3.7	6/12	15MB	40	4 channels DDR4-2133	140W	2011-v3	$583
NEW Intel® Core™ i7 5820K	3.3	Up to 3.6	6/12	15MB	28	4 channels DDR4-2133	140W	2011-v3	$389
Intel® Core™ i7 4790K	4.0	Up to 4.4	4/8	8MB	16	2 channels DDR3-1600	88W	1150	$339
Intel® Core™ i5 4690K	3.5	Up to 3.9	4/4	6MB	16	2 channels DDR3-1600	88W	1150	$242

Intel Alder Lake 10-core desktop CPU



socket : LGA 1200
area : 200 mm^2



A8 SoC designed by Apple for:
iPad mini 4, iPhone 6

produced by TSMC
(Taiwan Semiconductor)

process: 20 nm

area : 89 mm^2
# of transistors: 3 billion

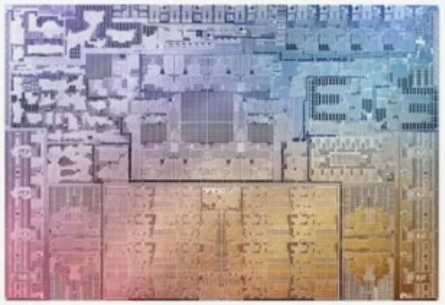
instruction set: ARMv8-A

dual CPU: 64 bit, 1.5 GHz
caches: L1 / L2 / L3
64+64kB/2MB/4MB

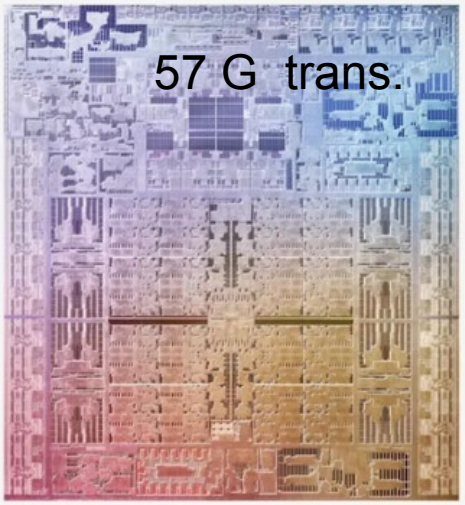
GPU: 8 cores,
148/296 GFlops (FP32/FP16)

16 G trans., 32 G trans. 57 G trans.

M1



M1 Pro

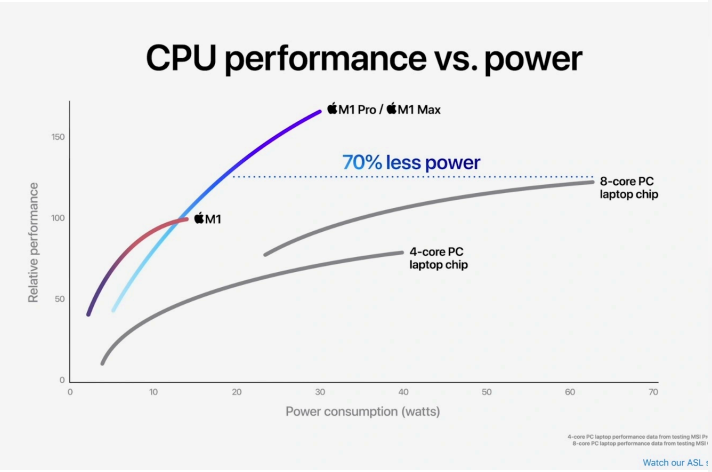


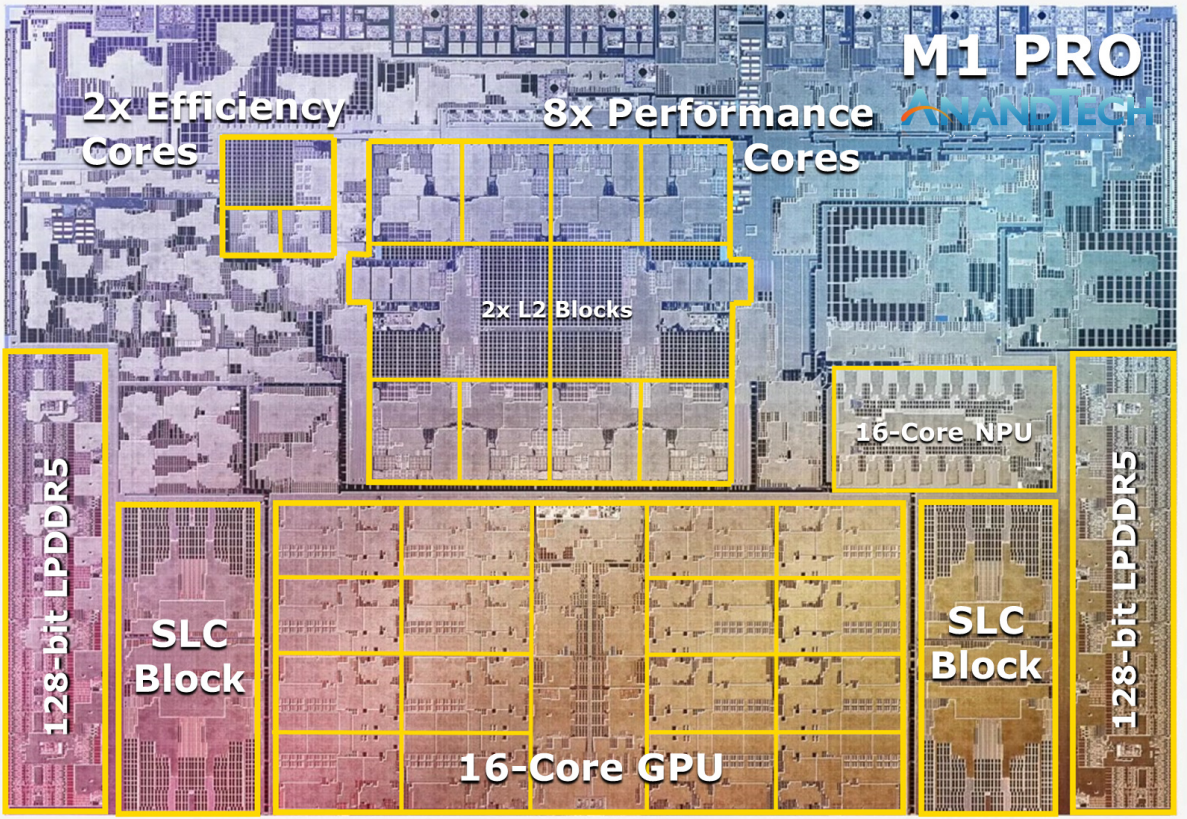
M1 Max

M1 Max processor
5 nm process (N5)

10-core CPU,
32-core GPU

64 GB unified memory
400 GB/s bandwidth





Modern supercomputing clusters compute in parallel. They consist of many nodes (workstations running Linux operating system), connected by a fast network in order to work in parallel on a single problem (or on many problems simultaneously). Hierarchical hardware inside as well.

Even though the modern networks (Ethernet, Infiniband) transfer from 1 to 20 GB/s between arbitrary two nodes using networking switches (at the top of the picture, there is 2 GB/s Infiniband switch), data can flow much faster, at 250-500 GB/s, inside the computational cards.

Node connectivity can therefore be a bottleneck for some (fortunately not all) parallel computer applications.

Picture shows some of the 28 nodes of the UTSC supercomputer designed and built in 2017 by prof. P. Artymowicz + u/g student.
Computations are done by:

- CPUs (6-core, 4GHz overclock),
- GPUs (graphics cards), and/or
- Intel Xeon Phi cards (57-core processors)

Each node contains CPU + 2 Nvidia graphics cards capable of combined ~10 TFLOPs in single precision.

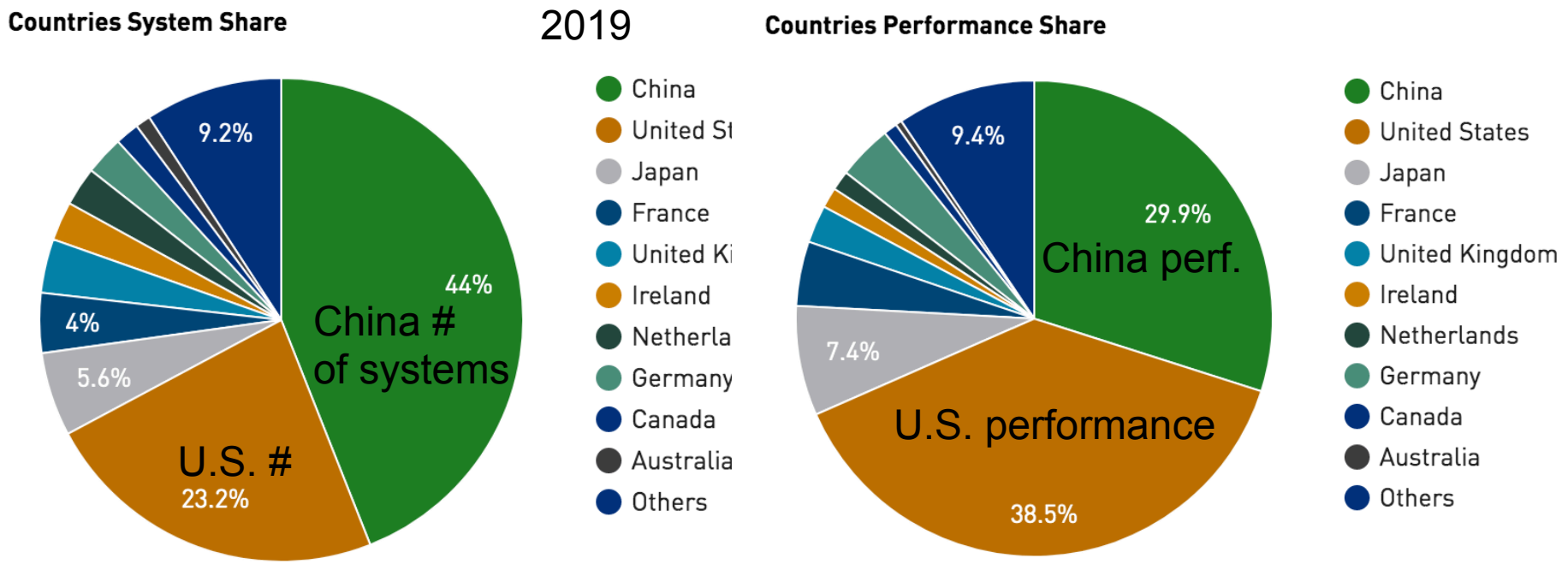
This science cluster can compute 10^{12} (trillion) times faster than ENIAC in 1946, and

~6 billion times faster than similarly sized PDP-11 (1970), using
~10 million times larger memory (RAM, disks) than PDP-11/45.

It costs ~10% of PDP-11.

Who has the fastest supercomputers today? US & China.

(U.S. used to be the dominant player for 65 years, 1945...2010)



Title of media report in November **2017**: "China overtakes U.S in the Top 500 Supercomputers List"

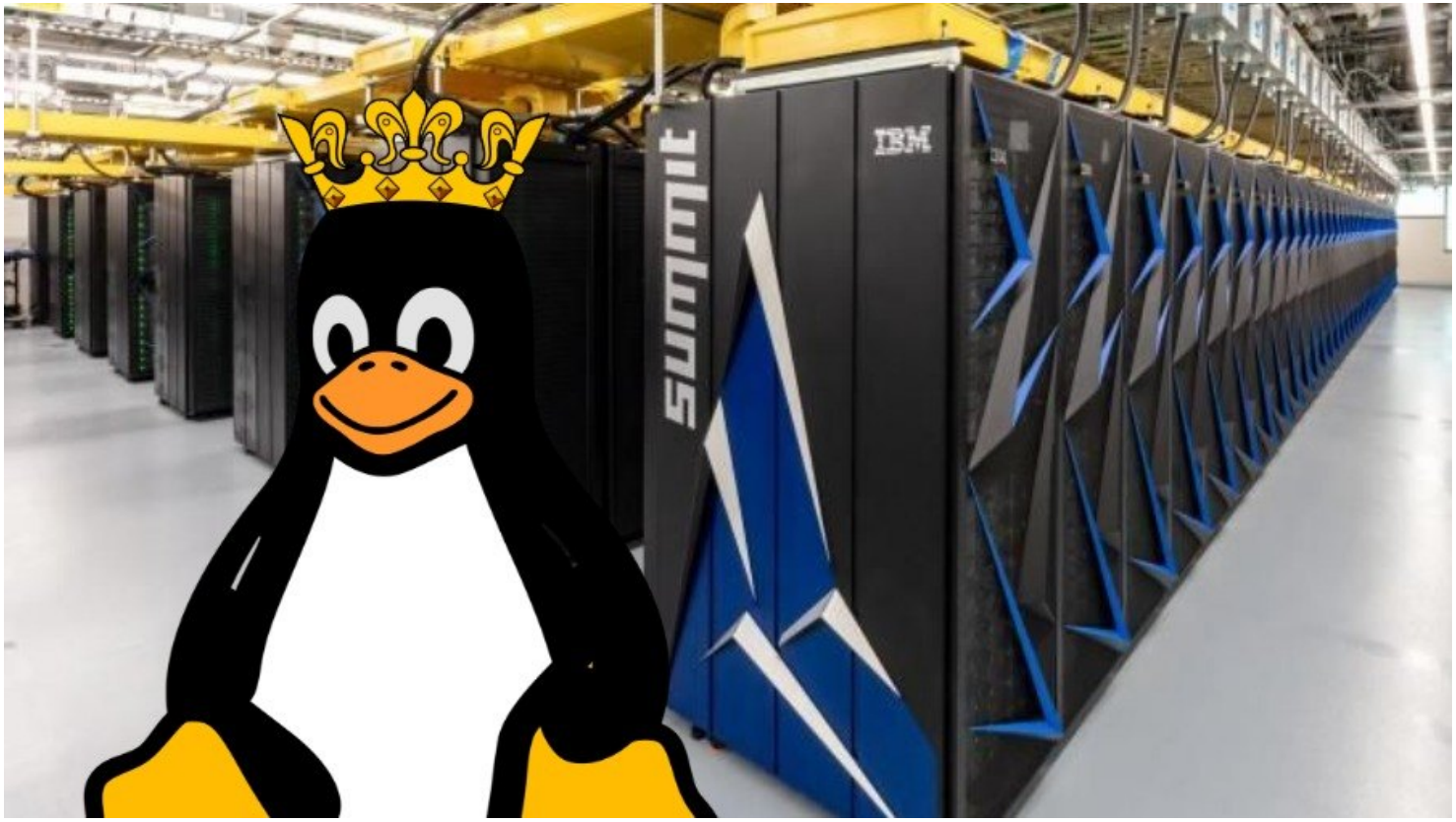
China also had two fastest supercomputers in 2017 (Sunway & Tianhe-2), including one based on own 240-core processors and one on Intel Xeon Phi processors (60-core).

Up to date statistics available on top500.org

0.2% of top 500 systems were Chinese, 54% were American in 1999

4% --- ,, --- 55% --- ,, --- in 2009

44% --- ,, --- 29% --- ,, --- in 2019



Title of article in June **2018**:
"Linux Powers ALL Top 500 Supercomputers in the World. U.S. beats China for #1"
Classified computer called *Summit* in the U.S. is now the fastest machine in the world.

Linux followed Unix in its domination of HPC (High Performance Computing); but MacOS (Linux derivative, has no role in HPC, while Windows OS was never a good choice.

https://www.top500.org/statistics/sublist/

Modern top supercomputers:
rows of 19”-wide racks filling basketball fields, 1000s of nodes (workstations), consume up to ~23 MW of electric power and emit heat at that rate.
Cost ~$300 million.
Not environment-friendly (e.g. UofTs SciNet is warming/damaging env., whose warming it was meant to study.)

However, let’s put things into perspective! That’s about the price of *ONE* Boeing 777 airplane, whose engines also produce 23 MW of power in cruise & much more while climbing.
There are 1600 B777s, and 10000 Airbuses! ea. costs as much as a supercomputer center

Summit has 4608 nodes, 9216 IBM POWER9 CPUs and 27648 Nvidia Tesla GPUs.
Most computational power is from those Tesla V100 graphics cards.

The combined performance is measured at ~150 PFLOPS, out of the theoretical number of 200 PFLOPS (Summit can perform 150,000,000,000,000,000 arithmetic ops/s).

Modern top supercomputer:

150,000,000,000,000,000 arithmetic operations/second.

Who'd ever need that?

No single user is allowed to hog the whole supercomputer, there are thousands of users at national supercomputing centers.

We will do a realistic estimate of how long one scientific model runs on Summit.

[*Do not memorize it! It's informative, but not exact to better than on order of magnitude. Read this & the next 2 slides for fun.]*

Suppose 1000 scientists want to do high-resolution simulations in 3D, simultaneously. Each of them divides simulated object or region of space into, say, 2000x2000x2000 cells, or 8G cells [$(2K)^3$ = 8G]. Each cell must hold 5-10 floating point numbers of length 8 Bytes (double precision). For instance, in a fluid simulation those would be density and pressure, as well as 3 components of velocity vector in each cell. Storage of at least 40 to 80 B/cell is usually needed.

The total volume of simulated data may thus be ~1000* 8G*(40-80 B) ~ 500 TB.

Modern top supercomputer:

150,000,000,000,000,000 arithmetic ops/s. Is this not an overkill and a waste of money?!

If the total volume of simulated data is ~500 TB, and ~30K graphics cards do simultaneous number crunching on Summit, then

~500GB/30 ~16 GB

of this data must be processed by each GPU in each time step. (Today, 16 GB can fit on the biggest RAM available on GPUs, avoiding transfer from/to CPU RAM, a good thing!) Constantly shuffling data back and forth to RAM at a bandwidth of ~300 GB/s, the GPU card takes at least

~(16 GB) /(300 GB/s) = ~0.05 s

to process its allotted data in a given time step of the simulation (inter-processor and inter-node communication takes additional time, sometimes majority of time, but let's assume this does not apply to CFD = Computational Fluid Dynamics.)

Next, how many time steps are needed?

Modern supercomputers:

Large-scale simulations take a lot of time & use all available resources! There is never "too much resolution" or "too much precision" in science or engineering.
Think of simulating a new passenger jet flying with complicated wing flaps. Is 2000x2000x2000 resolution sufficient? (Assume the length of aircraft is 70 m. Estimate average cell size. Boundary layer of air is a few mm thick.)

A high resolution simulation may need ~1 million time steps to complete. (That's because the computational cells are small and modeled physical signals propagating through the grid of cells cannot cross more than 1 cell per time step. In practice, a physical disturbance of some sort crossing 50 times a grid of 2000 cells in both directions needs a minimum of 500K steps, in agreement with the 1M estimate).

If so, then each of our hypothetical 1000 researchers have to wait a *minimum* of

1M * (0.05s) ~ 14 hours

for their simulations to complete, if everything works at 100% efficiency.
So the seemingly 'ridiculously large' number of arithmetic ops per second (>10^{17}), of which Summit is capable, is not so ridiculous after all when shared among scientists.

- Taking into account *bandwidth limitations* (decisive for most computations!), Summit is of course fast but not excessively fast. Neither will have a 1 EFLOP performance.

1 EFLOPS = ExaFLOPS = 10^{18} FLOP/s in double precision (8B/floating point). Such computers were supposed to arrive by 2020. Now... in 2022 or later.

OpenMP = Open Multiprocessing (cf. wiki)

OpenMP, "Hands-on intro to OpenMP", a slide show by Intel programmers from SC08, Austin, TX

https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf

Full description of OpenMP (fairly tedious reading but good to have for reference)

https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

Examples of good and erroneous application of OMP. Lots of interesting code snippets in C/C++ and Fortran:

https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf

Intel's writeup about performance limitations of OpenMP-instrumented codes.

https://software.intel.com/en-us/articles/performance-obstacles-for-threading-how-do-they-affect-openmp-code

Our code page http://planets.utsc.utoronto.ca/~pawel/PHYD57/
has links to program tetra*.f90/.f95 which solves a comp. demanding task

- Tutorials for Fortran and C are on our web page.
  We used the last link in the Languages section to review the usage of C
- Programs are placed in /progD57 directory and in art-2 program directory

- We looked at C-version of arithmetic/bandwidth demonstration program simpler-nb-3ra.c, which did not work (produced segmentation fault, problem with memory management, we say it 'segfaulted'). The reason was that the program tried to deposit results in a non-existing array element tab[6][k] . This needs to be changed to tab[5][k] because 6 indices of the 1st dimension are: 0,1,..,5. (C like Numpy has 0-base arrays.)
- simpler-nb-3ra.f90 is the corrected program, and the version with swapped indices is called simpler-nb-3da.c
- simpler-nb-3da.f90 and simpler-nb-3ra.f90 are the Fortran versions, which we analyzed during lecture.

OpenMP = Open Multiprocessing (cf. wiki)

OpenMP, "Hands-on intro to OpenMP", a slide show by Intel programmers from SC08, Austin, TX

https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf

Full description of OpenMP (fairly tedious reading but good to have for reference)

https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf

Examples of good and erroneous application of OMP. Lots of interesting code snippets in C/C++ and Fortran:

https://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf

Intel's writeup about performance limitations of OpenMP-instrumented codes.

https://software.intel.com/en-us/articles/performance-obstacles-for-threading-how-do-they-affect-openmp-code

Our code page http://planets.utsc.utoronto.ca/~pawel/PHYD57/
has links to program tetra*.f90/.f95 which solves a comp. demanding task