CSCI2467: Systems Programming Concepts

Slideset 14: Review and future of systems

Source: Prof. Randal Bryant, Carnegie Mellon Univ.

Course Instructors:

Matthew Toups
Caitlin Boyce

Course Assistants:

Saroj Duwal
David McDonald

Spring 2020

THE UNIVERSITY of NEW ORLEANS

Final exam

20 out of 100 points will be assigned on Friday

- Friday, May 8
- 10:00am start
- 5 questions regarding shell lab:
  - fork and processes
  - process groups and signals
  - race conditions
  - input/output redirection

Final exam

The other 80 points: work on it now!

- Most of your exam will be the two take-home parts:
- Systems topics (40 points, due Thursday):
  - cachelab (C code)

    or
  - review of research talks
- Security topics (40 points, due Wednesday):
  - pwntools/attacklab (Python code)

    or
  - review of security research talks

Today

1. Review
   - 2nd half of semester
   - 3: Machine-level programs
     - Procedures / functions
     - Arrays, structs, pointers
   - 6: Memory, locality, caching
     - Memory hierarchy
   - 8: Exceptional Control Flow
     - Processes
     - System calls
     - Signals
   - 9: Virtual Memory
   - 10: Input/Output
2. Systems in the future
   - Moore's law
   - What would this look like?
   - Challenges ahead
3. Fin

Final exam is really just exam #2

Like the last exam, it contributes 100 points toward total grade.
Material: second half of course (see schedule)

	10	*Fall break (no class)*			
	15	Exceptional Control Flow: Introducing processes (slides)	209	8.1,8.2	
	17	Exceptional Control Flow: process control and shells (activity code)	209	8.3,8.4	shell lab out
	22	more ECF: fork() and your shell (example: forks.c)	209		
	24	demos, reaping, and signals (next steps)	209	8.5	shell lab **demo**
	29	Signals, process graphs and ECF conclusion (activity, doit.c)	209		
	31	File Input/Output and redirection (slides, activity, comment solutions, graph solution, forkSig.c)	209	10	
	5	Memory, locality, caches and Memory Hierarchy (slides)	209	6.1-6.3	
	7	Caches, procfs (activity)	209		shell lab due
November	12	Processes, caching and Virtual Memory (slides)	209	9.1-9.5	attack lab out
	14	Machine-level: procedures and call stack (slides, activity)	209	3.7	
	19	Machine-level: Stack attacks and defenses (slides)	209	3.10	
	21	Machine-level: pointers and data structures (slides, activity)	209	3.9	
	26	Exam review and future of systems	209		attack lab due
	28	*Holiday (no class)*			

Procedures (3.7)

- When `call` assembly instruction executes, what happens...
  - to `$rip` (instruction pointer or program counter)?
  - to `$rsp` (stack pointer)?
- What happens to the above during a `ret` ?
- How is the return address stored?

Practice Problem 3.32 (page 244) combines many of these skills

Practice problem 3.32 (p.244)

```
0000000000400450 <last>:
 400540: 48 89 f8        mov  rax,rdi      % L1: u
 400543: 48 pf af c6     imul rax, rsi     % L2: u*v
 400547: c3              ret
0000000000400548 <first>:
 400548: 48 8d 77 01     lea  rsi, [rdi+1] % F1: x+1
 40054c: 48 83 ef 01     sub  rdi, 1       % F2: x-1
 400550: e8 eb ff ff ff  call 400540 <last>% F3: call
 400555: f3 c3           rep ret           % F4: return
 .
 .
 .
 400560: e8 e3 ff ff ff  call 400548 <first> % M1:call
 400565: 48 89 c2        mov rdx, rax      % M2: resume
```

Practice problem 3.32 (p.244)

	Instruction		State values (at beginning)					
Label	PC	Instruction	`rdi`	`rsi`	`rax`	`rsp`	`*rsp`	Description
M1	0x400560	`call`	10			0x7fffffffe820		Call `first(10)`
F1								
F2								
F3								
L1								
L2								
L3								
F4								
M2								

Practice problem 3.32: Solution (p.339)

	Instruction		State values (at beginning)					
Label	PC	Instruction	`rdi`	`rsi`	`rax`	`rsp`	`*rsp`	Description
M1	0x400560	`call`	10	–	–	0x7fffffffe820	–	Call `first(10)`
F1	0x400548	`lea`	10	–	–	0x7fffffffe818	0x400565	Entry of `first`
F2	0x40054c	`sub`	10	11	–	0x7fffffffe818	0x400565	
F3	0x400550	`call`	9	11	–	0x7fffffffe818	0x400565	Call `last(9,11)`
L1	0x400540	`mov`	9	11	–	0x7fffffffe810	0x400555	Entry of `last`
L2	0x400543	`imul`	9	11	9	0x7fffffffe810	0x400555	
L3	0x400547	`ret`	9	11	99	0x7fffffffe810	0x400555	Return 99 from `last`
F4	0x400555	`rep ret`	9	11	99	0x7fffffffe818	0x400565	Return 99 from `first`
M2	0x400565	`mov`	9	11	99	0x7fffffffe820	–	Resume `main`

Arrays (3.8)

- How large is an array (given type of element and number of elements)?
- How are elements indexed?
- How does this change when dealing with an array of pointers?
- See struct activity, lecture notes

Buffer overflow: example

Great example: Practice Problem 3.46 (p.282)

`get_line()` is called with a return address....

- A: Stack diagram, in `get_line` before `gets(buf)`:
- (show what `push` does, where `rsp` points (and where `buf` is stored)
- B: Stack diagram after `gets(buf)`:
- Where the input bytes are stored
- Remember, memory contains ASCII values for each character typed (followed by NULL byte: `00`)
- C: What happens when `ret` (return) instruction is reached?
- Program attempts to return to value stored on stack (which has been partially changed by last 2 bytes of user input)
- D: What happens to `rbx` after a `pop` restores its value?
- E: What else is bad about the C code?
- `malloc()` needs an extra byte to store NULL character
- check return value on system call `malloc()`

Defenses and ROP

- What are the defenses to stack overflows we've seen?
  - Randomized stack location
  - Nonexecutable stack memory
  - Stack canaries

  Be able to explain these!
- ROP: return-oriented programming attacks (remember rtarget?)
  - How is it a response to the above?
  - How does it work? What is a "gadget"?

Today

1. Review
   - 2nd half of semester
   - 3: Machine-level programs
     - Procedures / functions
     - Arrays, structs, pointers
   - 6: Memory, locality, caching
     - Memory hierarchy
   - 8: Exceptional Control Flow
     - Processes
     - System calls
     - Signals
   - 9: Virtual Memory
   - 10: Input/Output
2. Systems in the future
   - Moore's law
   - What would this look like?
   - Challenges ahead
3. Fin

Locality Example

- **Question:** Does this function have good locality with respect to array `a`?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Qualitative Estimates of Locality

- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

- **Question:** Does this function have good locality with respect to array `a`?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Locality Example

- **Question: Can you permute the loops so that the function scans the 3-d array `a` with a stride-1 reference pattern (and thus has good spatial locality)?**

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

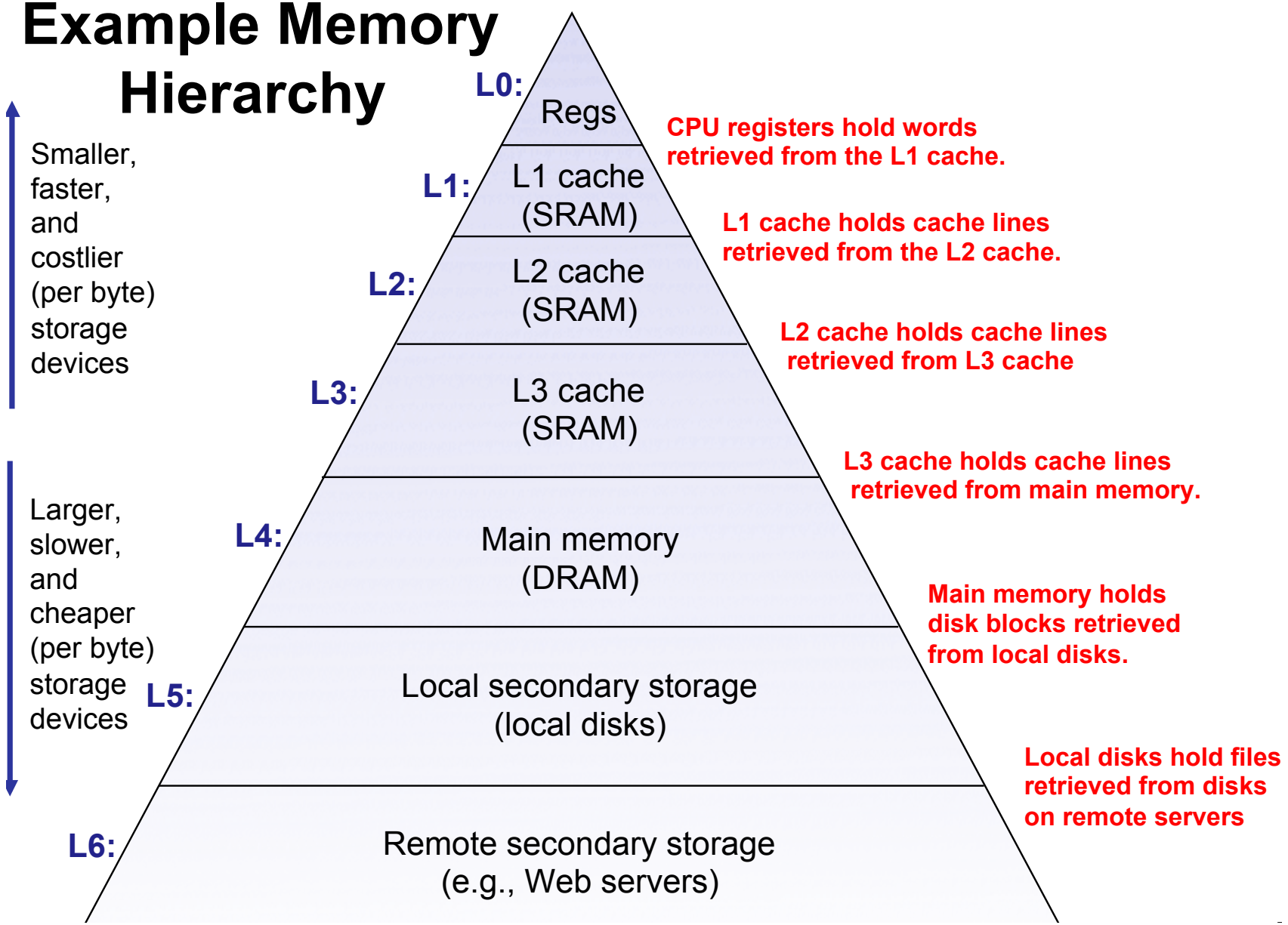
Memory Hierarchies

- **Some fundamental and enduring properties of hardware and software:**
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.

- **These fundamental properties complement each other beautifully.**

- **They suggest an approach for organizing memory and storage systems known as a memory hierarchy.**

Example Memory Hierarchy



CS:APP3e Figure 6.21

Explain the pyramid

Can you explain the roles of the following in the memory hierarchy? Describe them in terms: **size, cost, and access time / latency**. Which are largest/fastest/costliest?

- CPU registers
- SRAM cache
- DRAM
- Disk storage
- Network storage

Processes: a crucial concept in systems

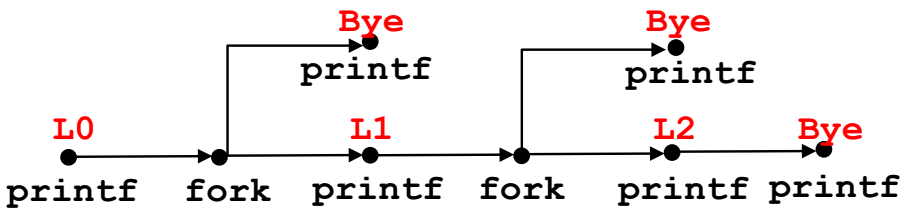
- What is a process?
- What abstractions does a process provide?
  - Control flow
  - Address space
- Explain these and explain how context switching and virtual memory systems enable these abstractions.

Process graphs and feasibility

- What output is possible and what is not possible?
- `forks.c` examples
- No guarantees of what processes get scheduled before others, but you do know that certain things in the code must happen before others.

Process graphs and feasibility

```c
void fork4()
{
    printf("L0\n");
    if (fork() != 0) {
        printf("L1\n");
        if (fork() != 0) {
            printf("L2\n");
        }
    }
    printf("Bye\n");
}
```
forks.c



Feasible output:

L0
L1
Bye
Bye
L2
Bye

Infeasible output:

L0
Bye
L1
Bye
Bye
L2

`fork()` and `execve()`

- `fork()` / `execve()`
  - what does it do? How does it return?
  - (be ready to explain the unusual return)
- Given example code, show what the output is.
  - after some `fork()` calls, how many lines of output?

  (with and without `if(fork == 0)`)
- Use a process graph to model these things.

signal handlers and `waitpid()`

- What do signal handlers do? What happens when they return?
- When is `SIGCHLD` received? (shell lab experiences!)
- What does `waitpid()` do?
- Given some code that does these things, can you predict output?

VM and its relationship to previous topics

- What are the benefits of Virtual Memory?
  - VM allows for virtual address space larger than physical memory size
  - Memory management: each process has its own independent address space
  - Protection/isolation: processes can't interfere with each other, access to memory can be restricted with privileges
- How does it relate to caching, locality, processes?

File I/O

- Input/Output redirection using `dup2()` function
- See `do_redirect()` in shell lab

Today

1. Review
   - 2nd half of semester
   - 3: Machine-level programs
     - Procedures / functions
     - Arrays, structs, pointers
   - 6: Memory, locality, caching
     - Memory hierarchy
   - 8: Exceptional Control Flow
     - Processes
     - System calls
     - Signals
   - 9: Virtual Memory
   - 10: Input/Output
2. Systems in the future
   - Moore's law
   - What would this look like?
   - Challenges ahead
3. Fin

Origins of Moore's Law

April 19, 1965

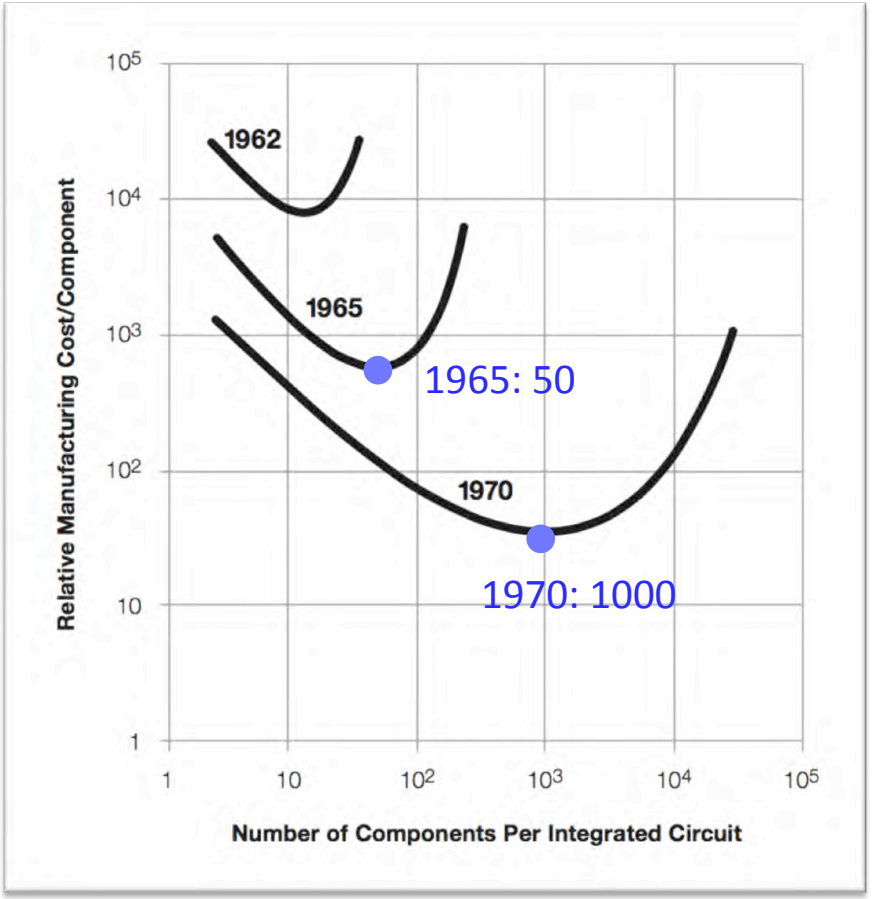
Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

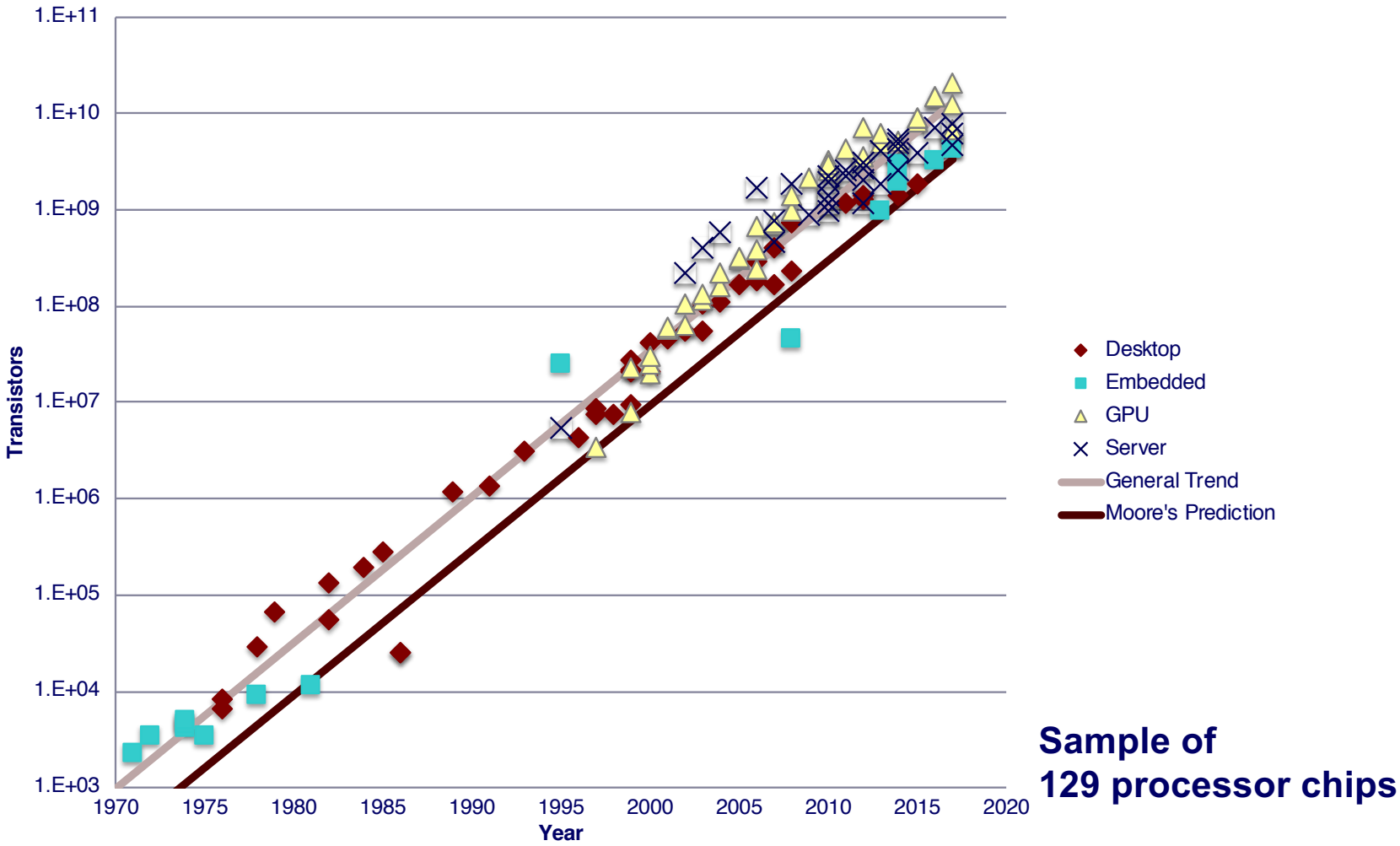
Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

Moore's observation



- **Moore's Thesis**
  - Minimize price per device
  - Optimum number of devices / chip increasing 2x / year
- **Later**
  - 2x / 2 years
  - "Moore's Prediction"

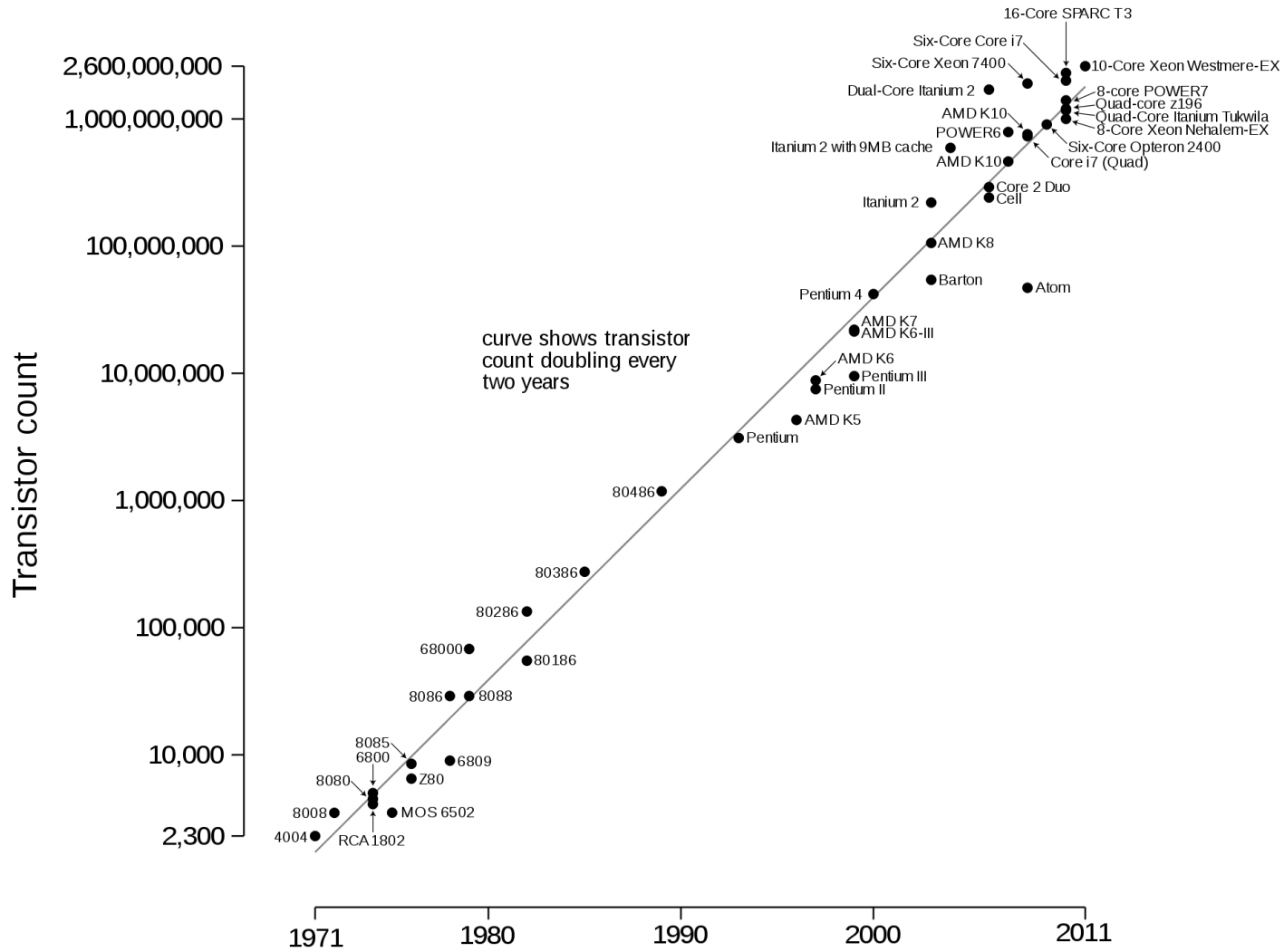
50-year perspective



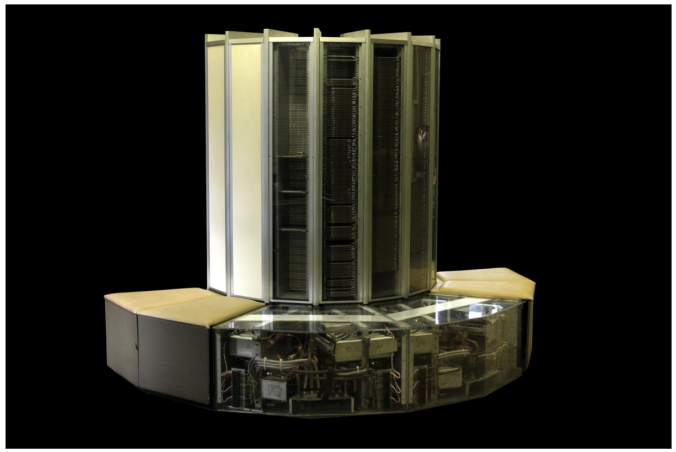
Sample of 129 processor chips

Transistor count

see: https://en.wikipedia.org/wiki/Transistor_count



Moore's Law Benefits



1976 Cray 1

2018 iPhone XS

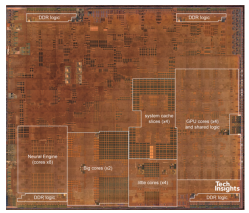
Moore's Law Benefits

What Moore's law has wrought

1965 Consumer Product

2018 Consumer Product

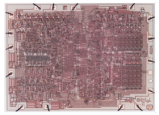
Apple A12
6.9 B transistors
(not to scale)



Visualizing Moore's law to date

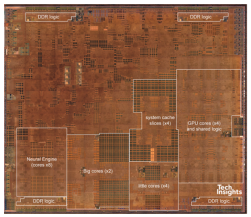
If transistors were the size of a grain of sand

Intel 4004
1970
2,300 transistors



0.1 g

Apple A12
2018
6.9 B transistors



300 kg

What Moore's law could mean?

- **2015 Consumer Product**

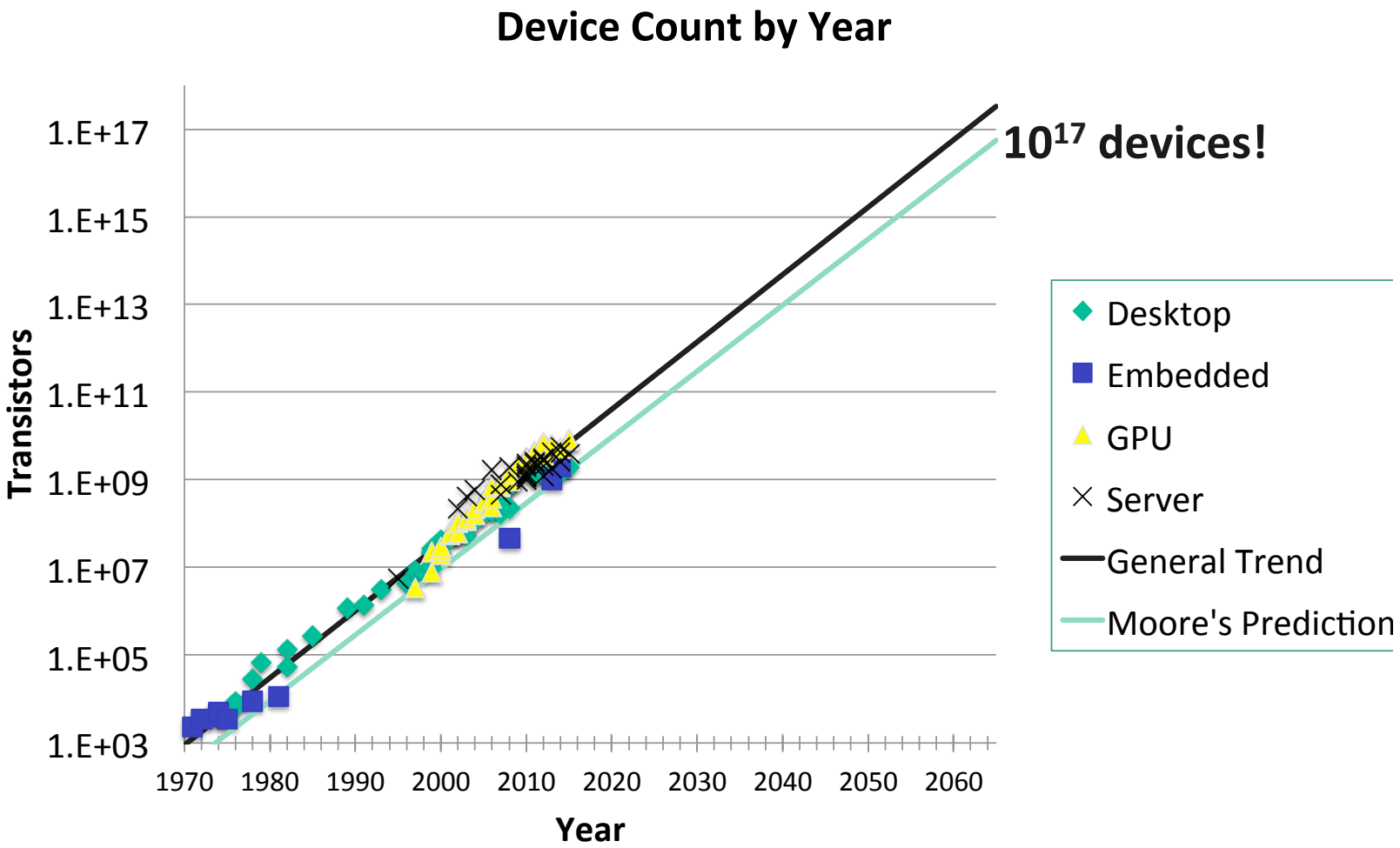
- **2065 Consumer Product**

  - Portable
  - Low power
  - Will drive markets & innovation

Requirements for future tech

- **Must be suitable for portable, low-power operation**
  - Consumer products
  - Internet of Things components
  - Not cryogenic, not quantum
- **Must be inexpensive to manufacture**
  - Comparable to current semiconductor technology
    - O(1) cost to make chip with O(N) devices
- **Need not be based on transistors**
  - Memristors, carbon nanotubes, DNA transcription, ...
  - Possibly new models of computation
  - But, still want lots of devices in an integrated system

Moore's law: 100 years?



Devices == transistors (so far...)

Visualizing 10^{17} devices

If devices were the size of a grain of sand

0.1 m^3

3.5 X 10^9 grains

1 million m^3

0.35 X 10^{17} grains

Increasing transistor counts

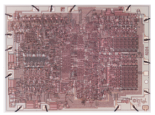
1. **Chips have gotten bigger**
   - 1 area doubling / 10 years
2. **Transistors have gotten smaller**
   - 4 density doublings / 10 years

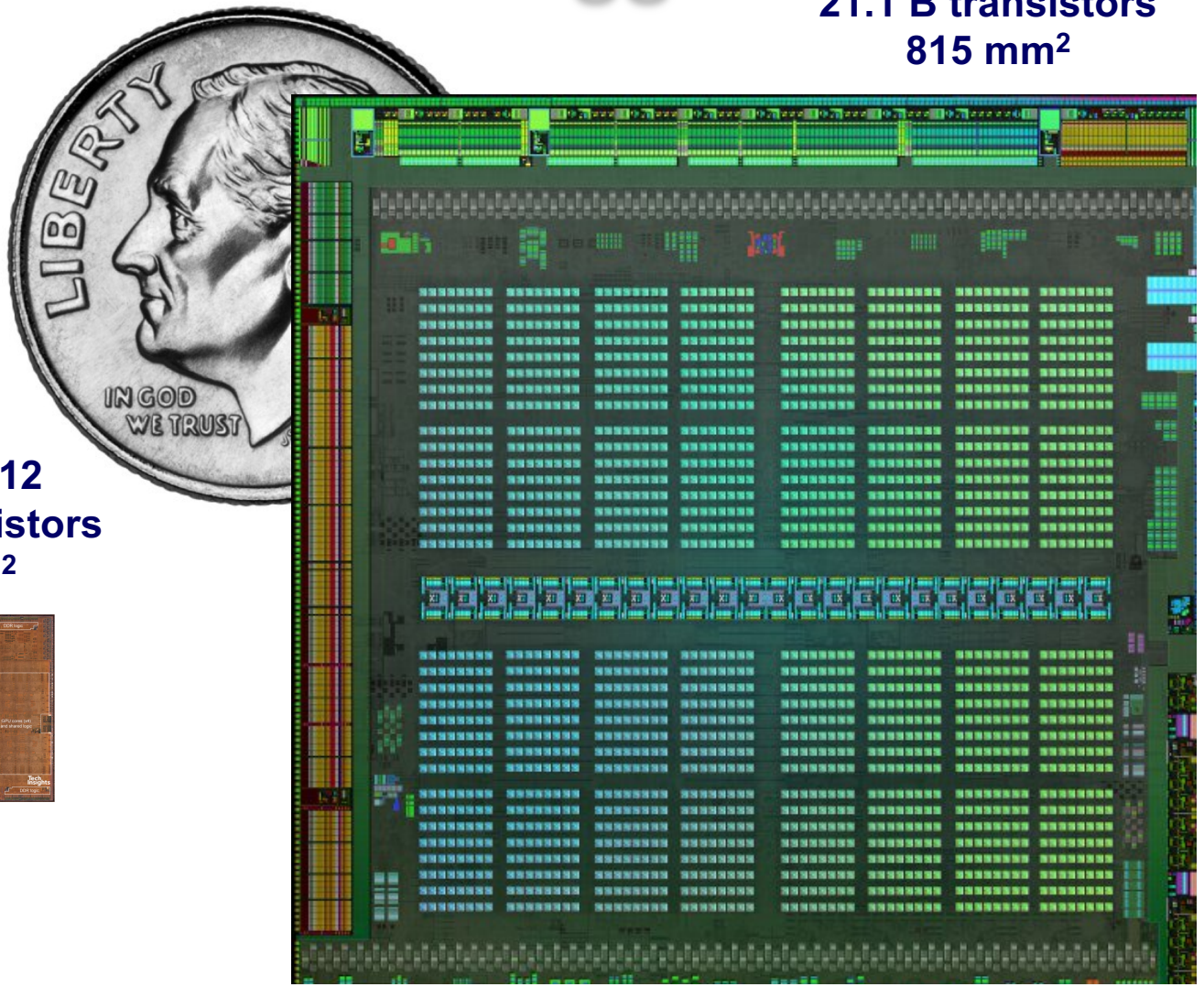
Will these trends continue?

Chips Have Gotten Bigger

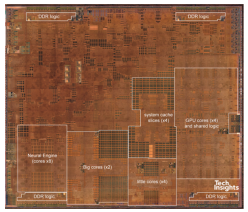
NVIDIA GV100 Volta
2017
21.1 B transistors
815 mm^2

Intel 4004
1970
2,300 transistors
12 mm^2

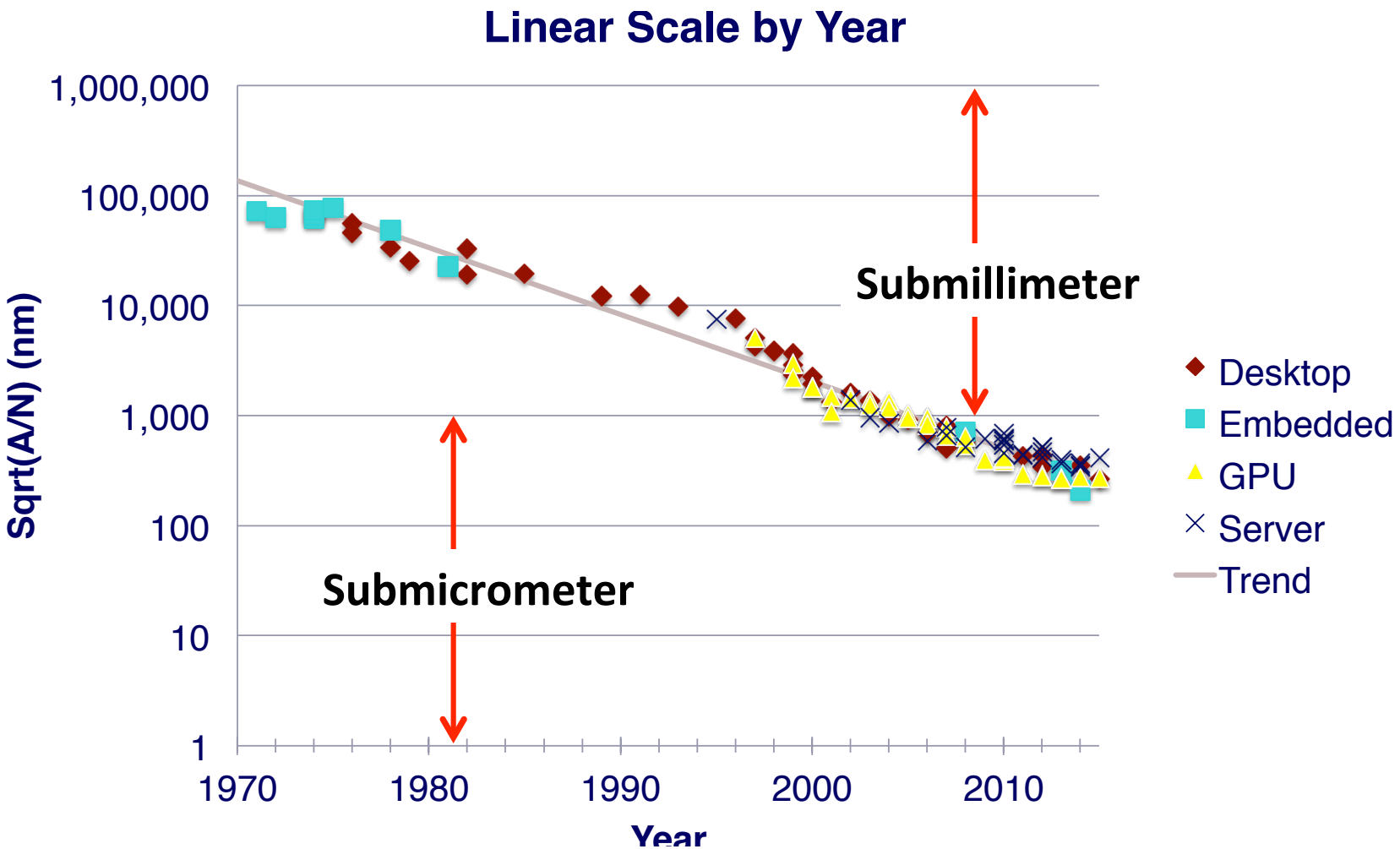




Apple A12
6.9 B transistors
83 mm^2



Linear scaling trend

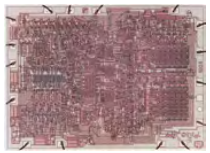


Submillimeter dimensions

10^{-3} 1 millimeter (mm)

500µm: Length of amoeba

10^{-4}



72µm: Intel 4004 linear scale

50µm: Average size of cell in human body

10^{-5}

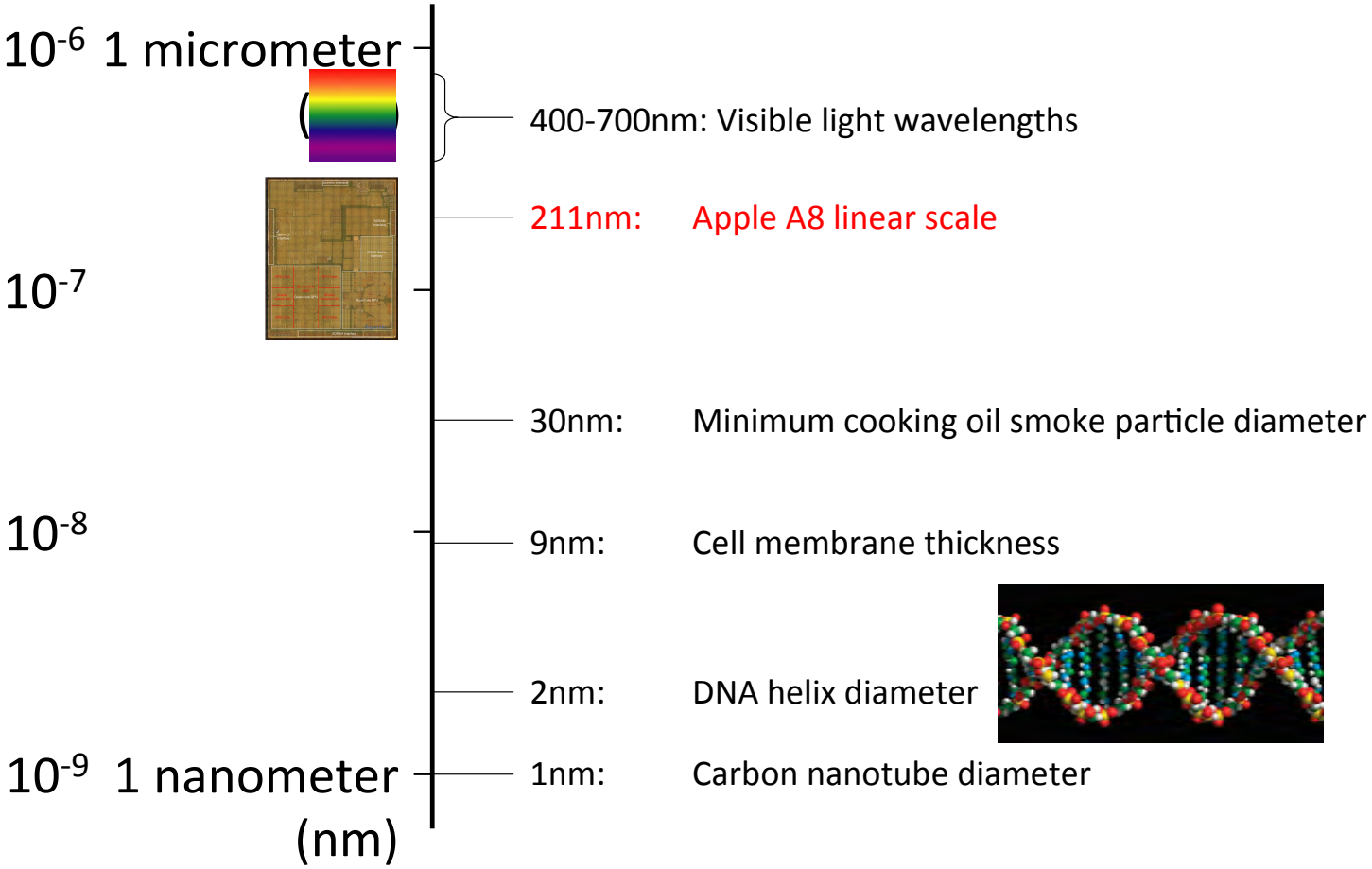
10µm: Thickness of sheet of plastic food wrap

5µm: Spider silk thickness

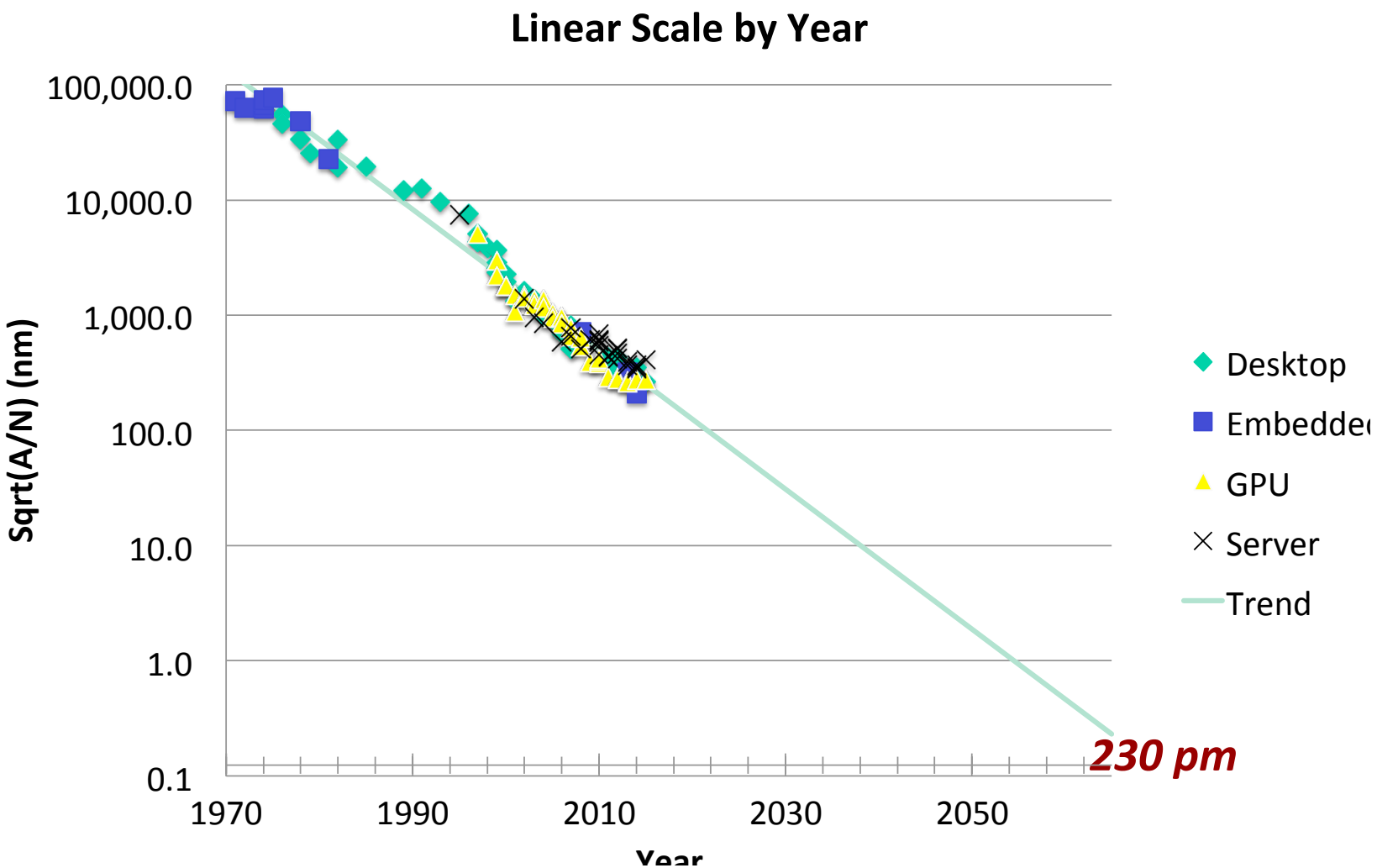
2µm: E coli bacterium length

10^{-6} 1 micrometer (µm)

Submicrometer dimensions



Linear scaling extrapolation



Subnanometer dimensions

10^{-9} 1 nanometer (nm)

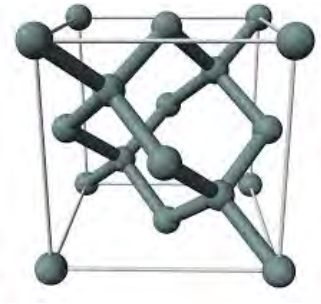
10^{-10}

10^{-11}

10^{-12} 1 picometer (pm)

1nm: Carbon nanotube diameter

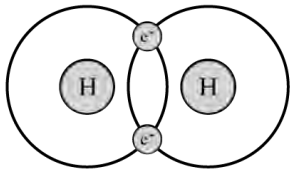
543pm: Silicon crystal lattice spacing



230pm: 2065 linear scale projection

74pm: Spacing between atoms in hydrogen molecule

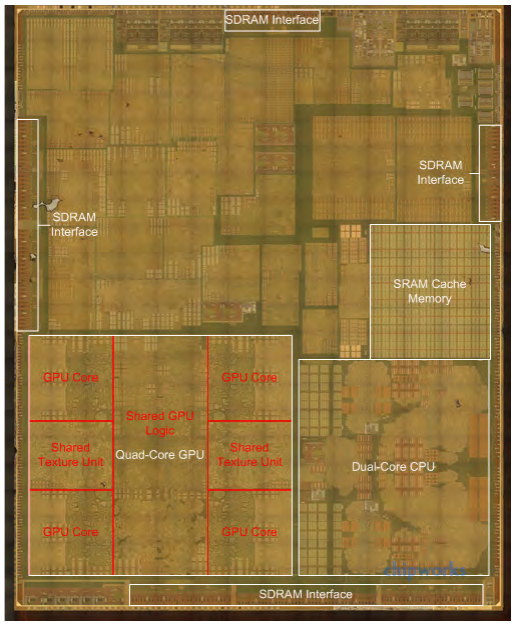
53pm: Electron-proton spacing in hydrogen (Bohr radius)



2.4pm: Electron wavelength (Compton wavelength)

Can the next 50 years work like the last 50?

- **Target**
  - 10^{17} devices
  - 400 mm^2
  - L = 63 pm

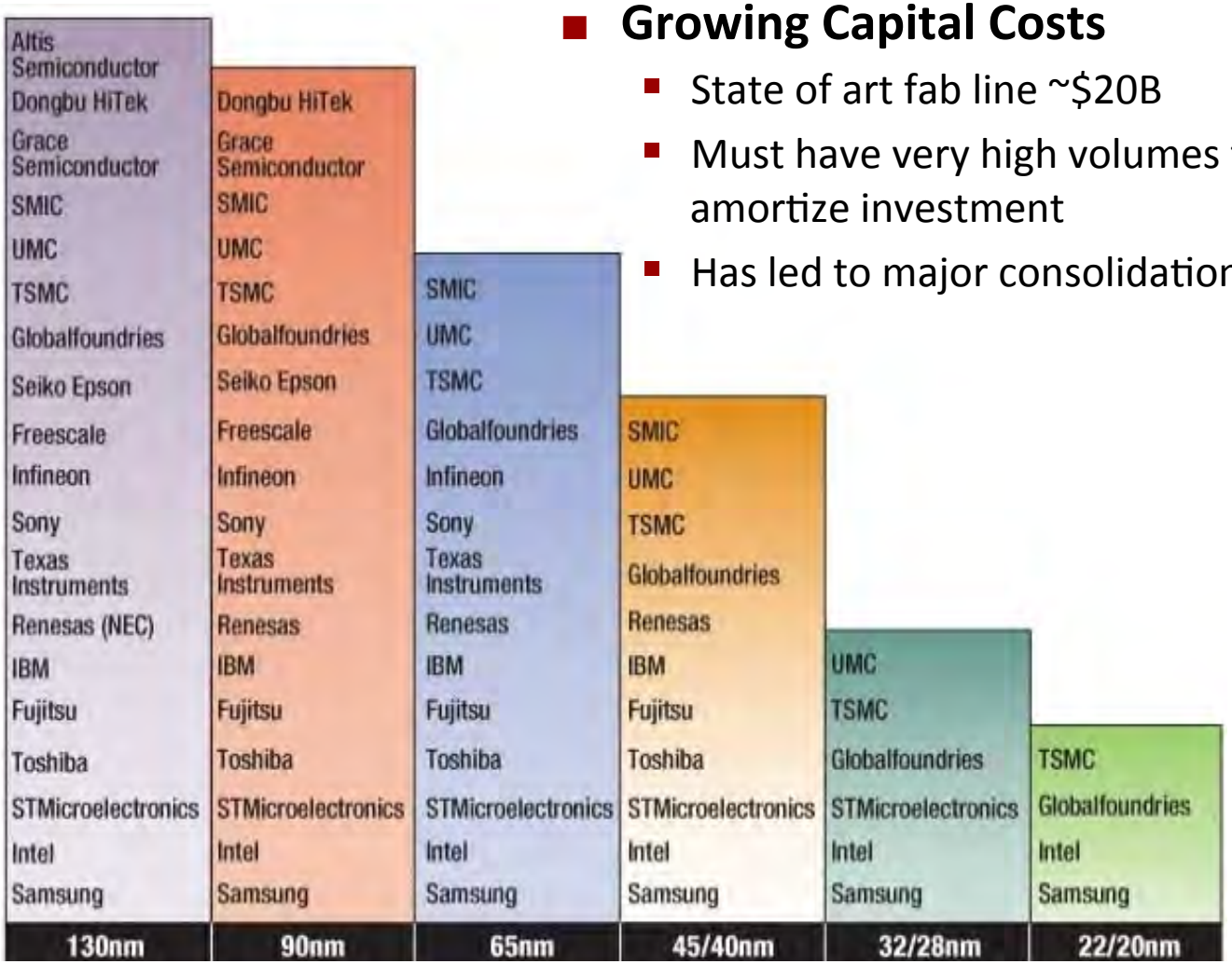


- **Is this possible?**

No!

Not with 2-d fabrication

Challenges to Moore's Law



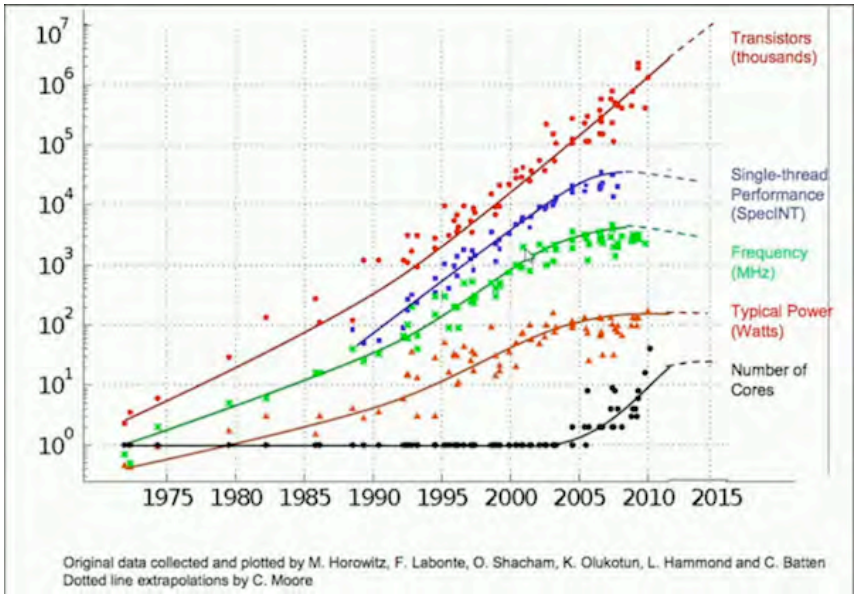
- **Growing Capital Costs**
  - State of art fab line ~$20B
  - Must have very high volumes to amortize investment
  - Has led to major consolidations

Dennard scaling

- Due to Robert Dennard, IBM, 1974
- Quantifies benefits of Moore's Law

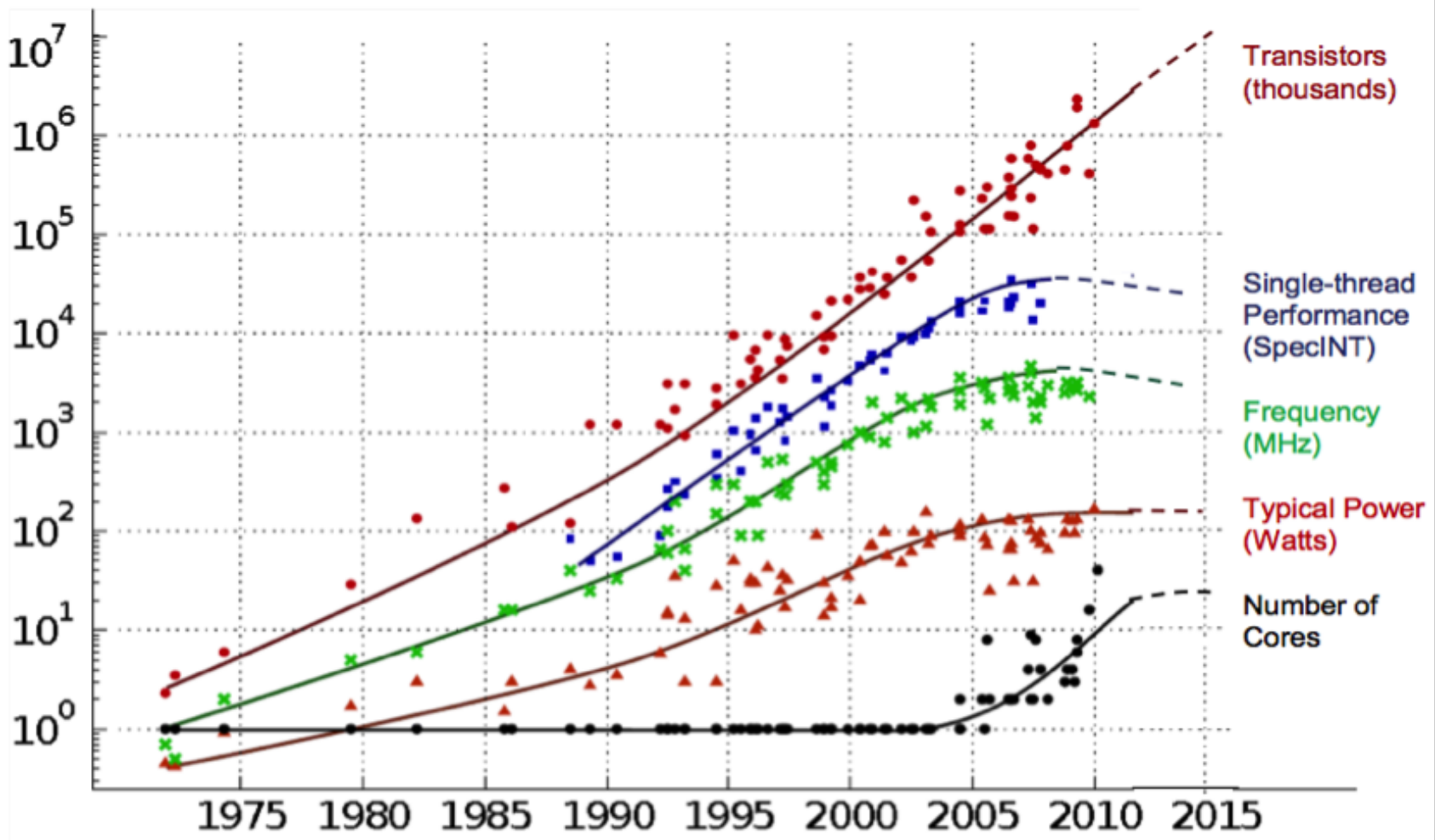
- **How to shrink an IC Process**
  - Reduce horizontal and vertical dimensions by k
  - Reduce voltage by k
- **Outcomes**
  - Devices / chip increase by k^2
  - Clock frequency increases by k
  - Power / chip constant
- **Significance**
  - Increased capacity and performance
  - No increase in power

End of Dennard scaling



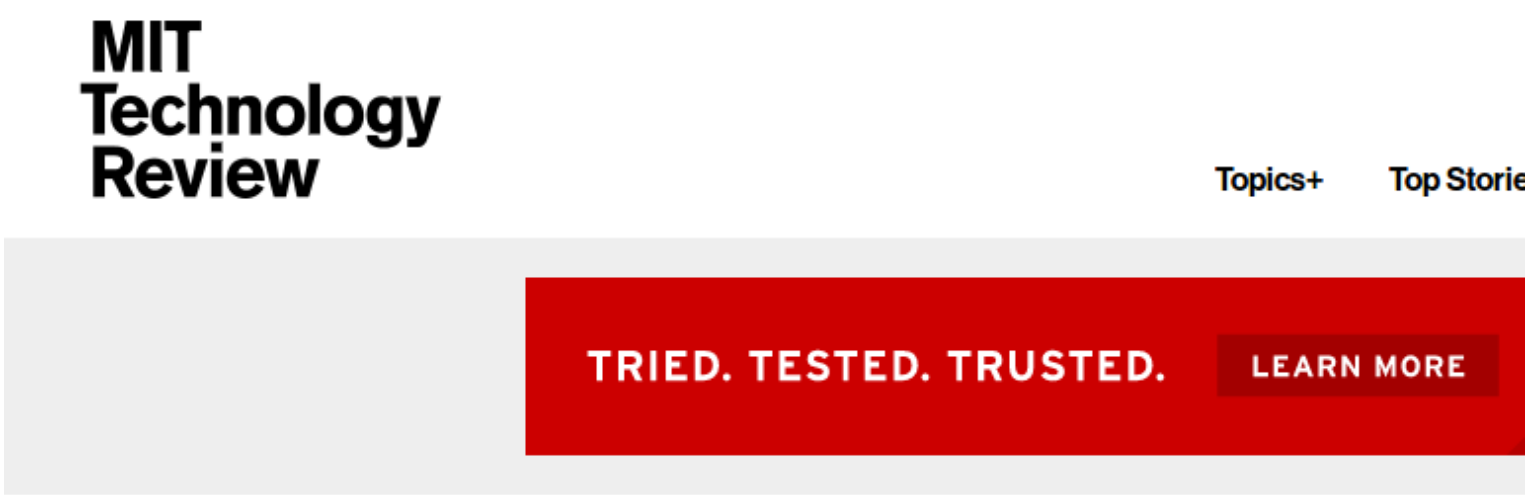
- **What Happened?**
  - Can't drop voltage below ~1V
  - Reached limit of power / chip in 2004
  - More logic on chip (Moore's Law), but can't make them run faster
    - Response has been to increase cores / chip

End of Dennard scaling



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten, dotted line extrapolations by C. Moore

"Death" of Moore's Law



Computing

Moore's Law Is Dead. Now What?

Shrinking transistors have powered 50 years of advances in computing—but now other ways must be found to make computers more capable.

by Tom Simonite May 13, 2016

Mobile apps, video games, spreadsheets, and accurate weather

"Death" of Moore's Law

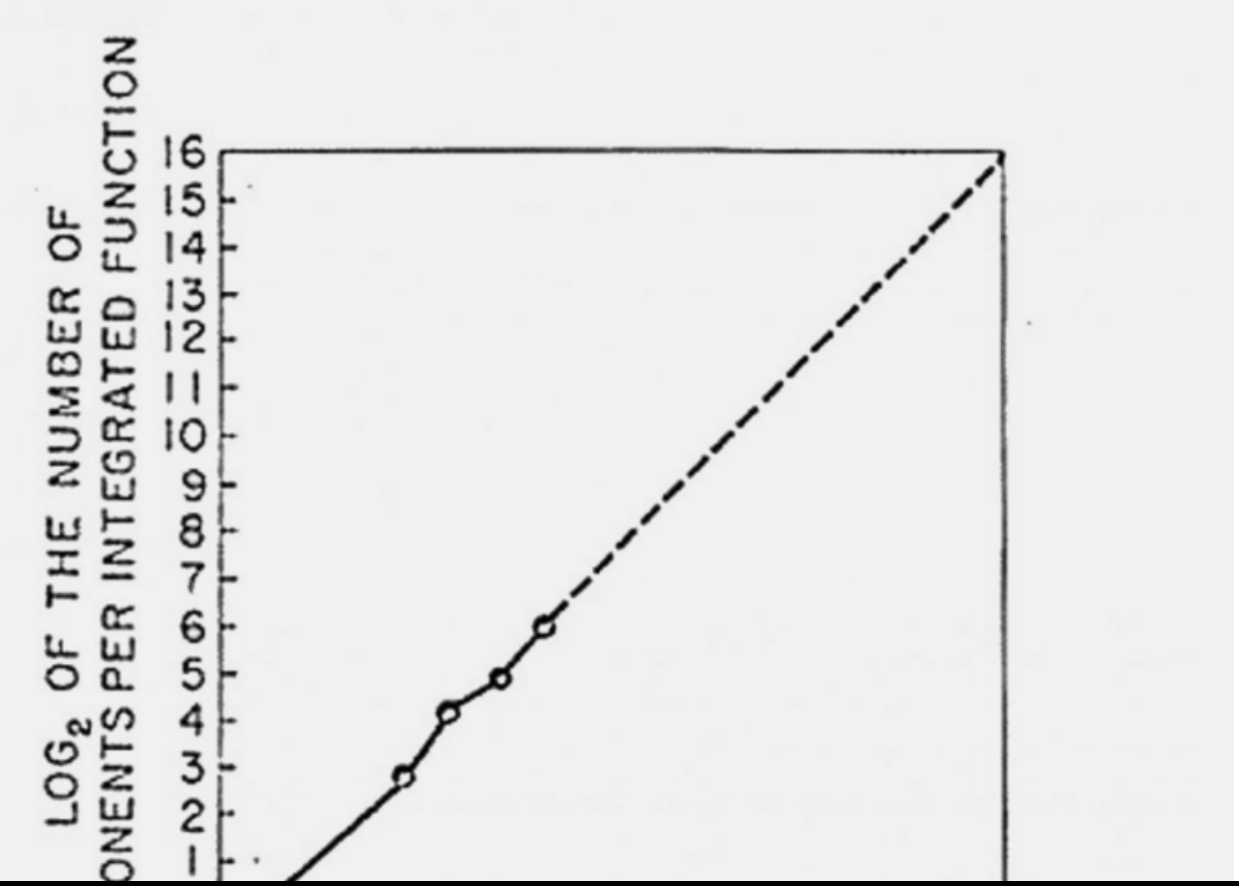
ars TECHNICA BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE FORUMS

TECHNOLOGY LAB —

Moore's law really is dead this time

The chip industry is no longer going to treat Gordon Moore's law as the target to aim for.

PETER BRIGHT - 2/10/2016, 7:22 PM



“Death” of Moore’s Law

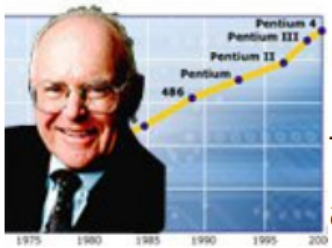
Home › News › Operating Systems News › Moore's Law is dead, says Gordon Moore

Moore's Law is dead, says Gordon Moore

Legendary chip man reviews the past, present and future.

By Manek Dubash | Apr 13, 2010

Share



Moore's Law is dead, according to Gordon Moore, its inventor.

The extrapolation of a trend that was becoming clear even as long ago as 1965, and has been the pulse of the IT industry ever since will eventually end, said Moore, who is now retired from Intel.

Forty years after the publication of his law, which states that transistor density on integrated circuits doubles about every two years, Moore said this morning: "It can't continue forever. The nature of exponentials is that you push them out and eventually disaster happens.

"In terms of size [of transistor] you can see that we're approaching the size of atoms which is a fundamental barrier, but it'll be two or three generations before we get that far - but

Systems and your future

- In this course you have seen many ways that systems have evolved over the past 50 years alongside the transistor-based computer
- This will inform how future systems evolve
- The systems in your daily lives will be based on these designs
- Perhaps future systems will be very different!
- You may shape the new designs

  . . . with the benefits of your understanding of the past

Writing the paper on systems topics for your final? (Or just interested?) See: No Moore Left to Give

Interested in Quantum computing? Sign up

Also see "No Moore Left to Give"

☆ ☆ ☆ NEW COURSE ☆ ☆ ☆

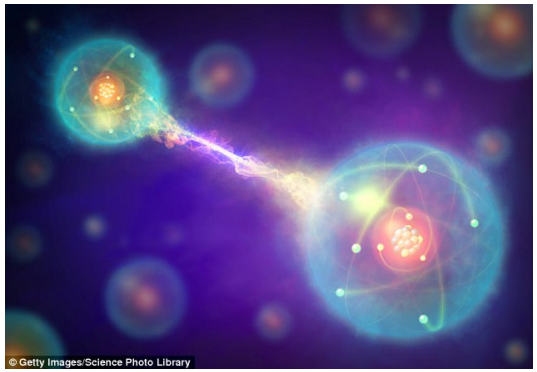
Math 4410/5410

Introduction to Quantum Nonlocality and Quantum Computing

Instructor: Dr. Peter Bierhorst

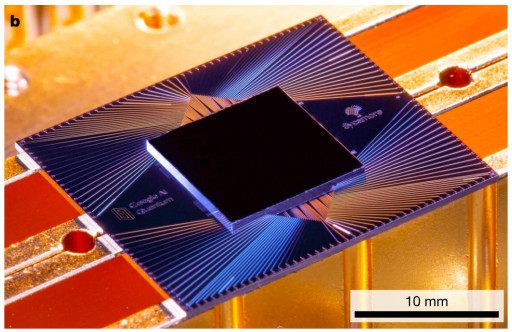
Fall Semester 2020 - 3 Credit Hours - Meets MWF 1-1:50

Quantum Nonlocality



What is Quantum Entanglement and Einstein's "Spooky Action at a Distance?"

Quantum Computing



Google announced they built a quantum computer in late 2019. What can it do?

Dr. Peter Bierhorst's research on quantum nonlocality and random number generation has been profiled in Wired Magazine and on NPR.

Final exam is Friday May 8

Don't forget to prepare for your exam!
Also finish up your take-home parts by Wed/Thurs.

- Why Mock exam?
- For Friday: make sure you have a working camera (laptop/smartphone) so we can see your face when you start your exam.
- Also you will screen-share so we can monitor your progress.
- (We will use "break-out rooms" in Zoom)
- Contact course staff *before* Friday if any of this is a problem.