University of Pittsburgh

# CS 1541 Introduction

## Instructor Introduction

## Course Introduction

Wonsun Ahn
Department of Computer Science
School of Computing and Information


# *Instructor Introduction*

# My Technical Background

- Wonsun Ahn
  - First name is pronounced *one-sun* (if you can manage)
  - Or you can just call me Dr. Ahn (rhymes with *naan*)
- PhD in CPU Design and Compilers
  - University of Illinois at Urbana Champaign
- Industry Experience
  - Software engineer, field engineer, technical lead, manager
  - Bluebird Corporation (70-person startup company)
    - Manufactures industrial hand-held devices from top to bottom
    - Me: Built software stack based on Windows Embedded
  - IBM Research (thousands of people)
    - Does next-gen stuff like carbon nanotubes, quantum computers
    - Me: Designed supercomputers for ease of parallel programming

# My World View

- Everything is connected
  - Pandemic: If my neighbors catch the virus, so will I
  - Environment: If my neighbors pollute, I will feel the effects
  - Economy: Think of how the subprime mortgage crisis spread
- Zero-sum thinking (old way of thinking)
  - “If you get a larger slice of the pie, I get a smaller slice.”
  - Therefore, if you lose, I win (and vice versa)
- Zero-sum thinking no longer works
  - If you catch the virus, do I become safer from the virus?
- Collaboration is replacing competition

# Collaboration is Replacing Competition


- Is happening in all spheres of life
- Collaboration is also happening in the IT industry
  - The *open source* movement
  - Increasing importance of the software/hardware *ecosystem*
  - Increasing importance of the developer *community*
- Collaboration is also important for learning
  - During my undergrad years, what do I remember best?
  - Stuff that I explained to my classmates
  - Stuff that my classmates taught me

# Supporting Collaborative Learning

- I *never* grade on a curve
  - You will not be competing against your classmates
  - You are graded on your own work on an absolute scale

- You will be a member of a Team
  - You are already a member of the class on Microsoft Teams
  - I encourage you to be on Teams at most times (I will too)
    - You can install app on both laptop and cell phone
  - If you have a question, you can ask in the Team “Posts” tab
    - Either your classmate or your instructor will answer
  - You can chat with any individual on the Team
    - “Manage Team” item in the “…” Team context menu

# Supporting Collaborative Learning

- You will be a member of a Group
  - On Teams, you are part of a chat group of 7~8 members
  - Members are a good mix of in-person and online students
  - Your instructor is also a member of each Group
  - It is a smaller support group where you can talk more freely

- You are allowed to discuss TopHat lecture questions
  - The goal is no-student-left-behind (pun intended)
  - Discuss answers on Teams even before submitting them
  - Form a basis of knowledge for doing homeworks and exams

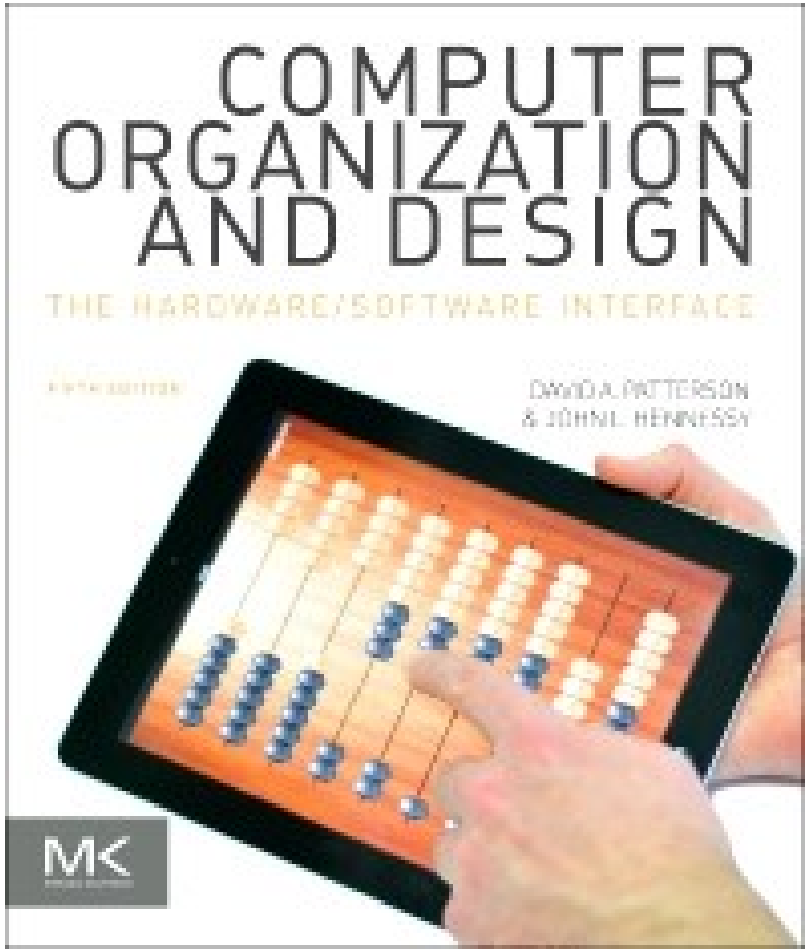

# *Course Introduction*

# Structure of the Course

- (45% of grade) Three midterms
- (20% of grade) Two projects
  - Implementing a CPU simulator using C programming language
- (20% of grade) Four homeworks
- (15% of grade) Participation
  - TopHat lecture questions, Teams participation
- Class resources:
  - Canvas: announcements, Zoom meetings, recorded lectures
  - GitHub: syllabus, lectures, homeworks, projects
  - Tophat: online lecture questions
  - GradeScope: homework / projects submission, grading and feedback
  - Microsoft Teams: Online / off-line communication

# Textbook (You Probably Have it)

- "Computer Organization and Design - The Hardware/Software Interface" by David Patterson and John Hennessy Fifth Edition - Morgan & Kaufmann.



# For More Details

- Please refer to the course info page: https://github.com/wonsunahn/CS1541_Fall2020/blob/master/course-info.md

- Please follow the course schedule syllabus: https://github.com/wonsunahn/CS1541_Fall2020/blob/master/syllabus.md

# TODO



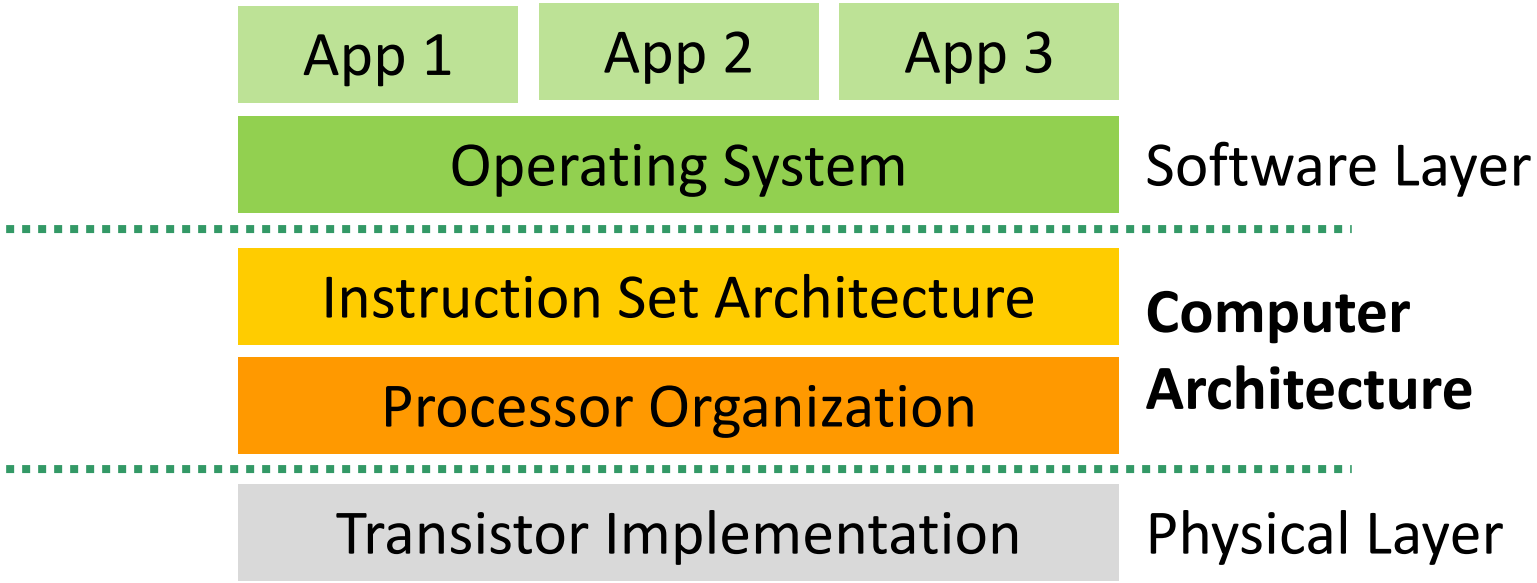
- Submit the TopHat survey questions (due 8/21)

# What is Computer Architecture?



- At a high-level: how a computer is built
  - Computer here meaning the processor (CPU)
- You probably heard of a similar term before: ISA
  - ISA (Instruction Set Architecture)
- Review: what is defined by an ISA?
  - Set of instructions usable by the computer
  - Set of registers available in the computer
  - Functional attributes are clearly defined (it's the interface)
- What is *not* defined by an ISA?
  - *Speed* of computer
  - *Energy efficiency* of computer
  - *Reliability* of computer
  - Performance attributes are undefined (intentionally so)

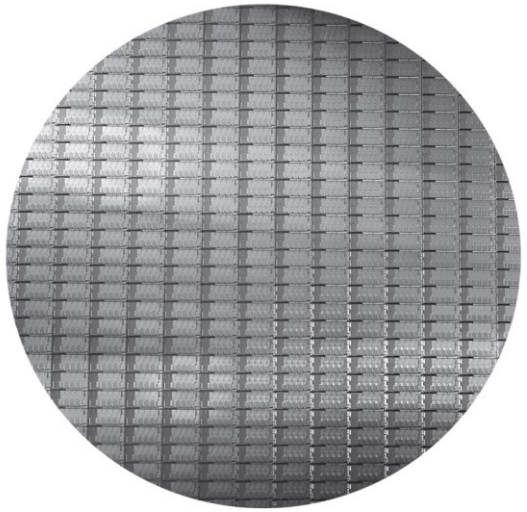
# Computer Architecture Defined




- Computer Architecture = ISA + Processor Organization
  - Processor organization is also called *Microarchitecture*
- Performance is decided by:
  - Processor organization (internal design of the processor)
  - Transistor implementation (semiconductor technology)

# Scope of Class

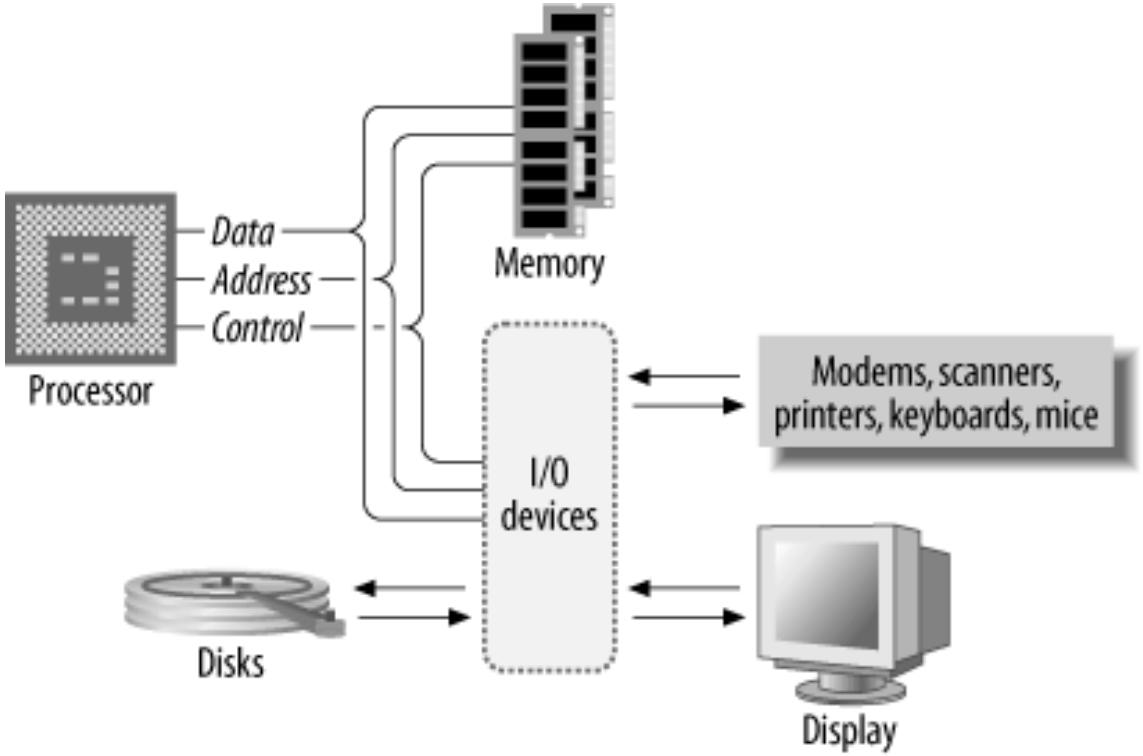
- Physical layer is beyond the scope of the class



- We will focus mostly on processor organization
  - And how performance goals are achieved

# Scope of Class

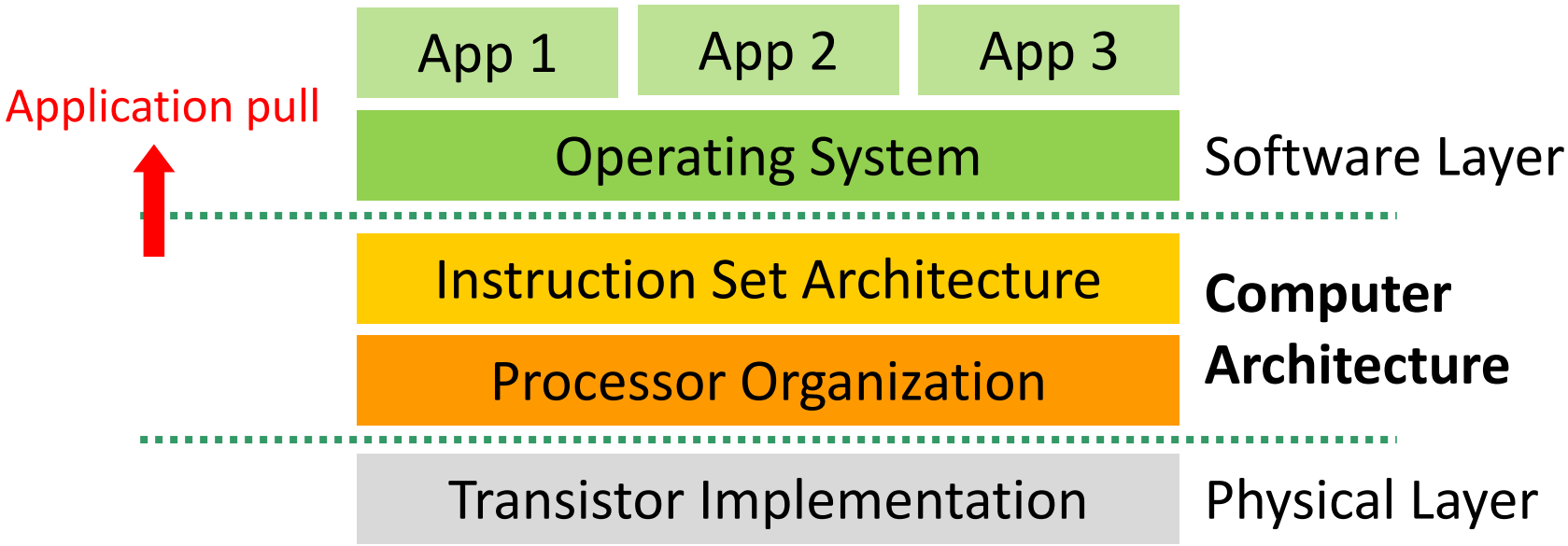
- Computer architecture is part of system architecture




- Other components beside processor is beyond the scope

# Application Pull

- Different applications pull in different directions



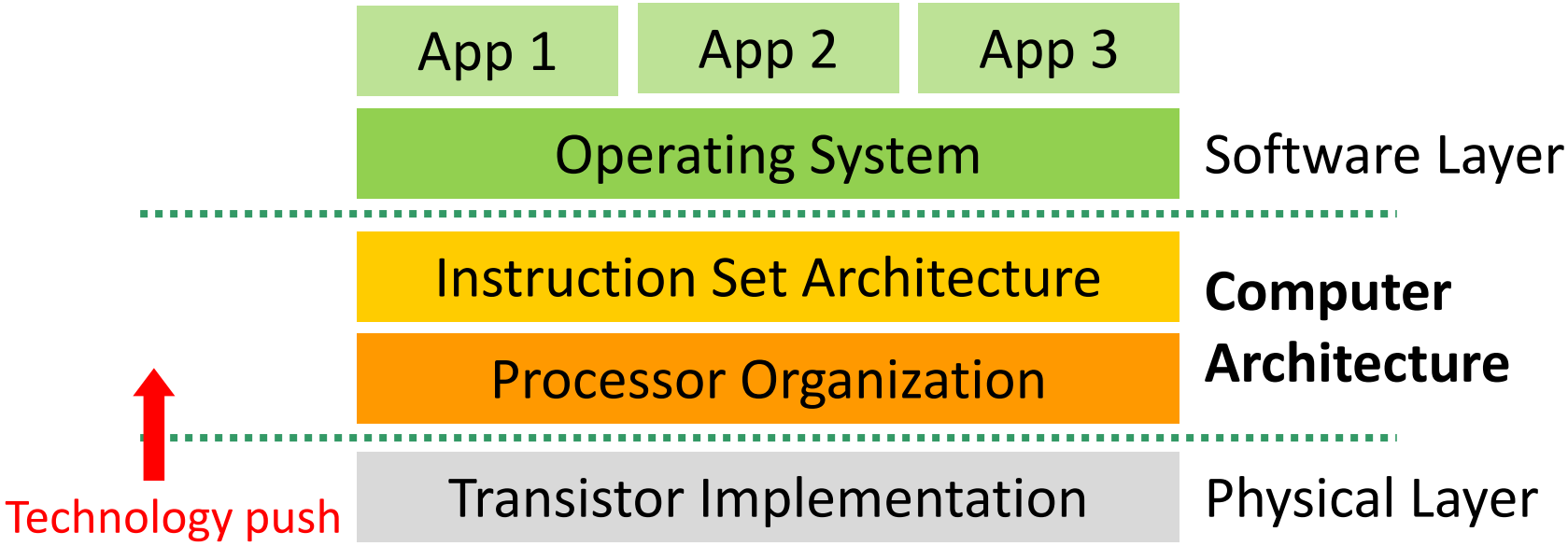

  - Real-time app (e.g. Game): *Short latency*
  - Server app: *High throughput*
  - Mobile app: *High energy-efficiency* (battery life)
  - Mission critical app: *High reliability*
- An app typically has multiple goals that are important

# Application Pull

- Some goals can be incompatible
  - E.g. Speed and energy-efficiency are incompatible
    - Running is faster than walking but uses more energy
    - A Ferrari is faster than a Prius but has worse fuel efficiency
  - E.g. Reliability is incompatible with many other goals
    - If you use redundancy, you use twice the amount of energy
- Even when sharing a goal, apps have unique needs
  - Scientific apps need lots of *floating point units* to go fast
  - Database apps need lots of *memory cache* to go fast
- An architecture is a compromise among all the apps
  - When app achieves market critical mass, designs diverge (Mobile chips / Server chips / GPUs / TPUs diverged)
  - Sometimes even ISAs diverge (GPUs and TPUs)

# Technology Push

- Trends in technology pushes architecture too



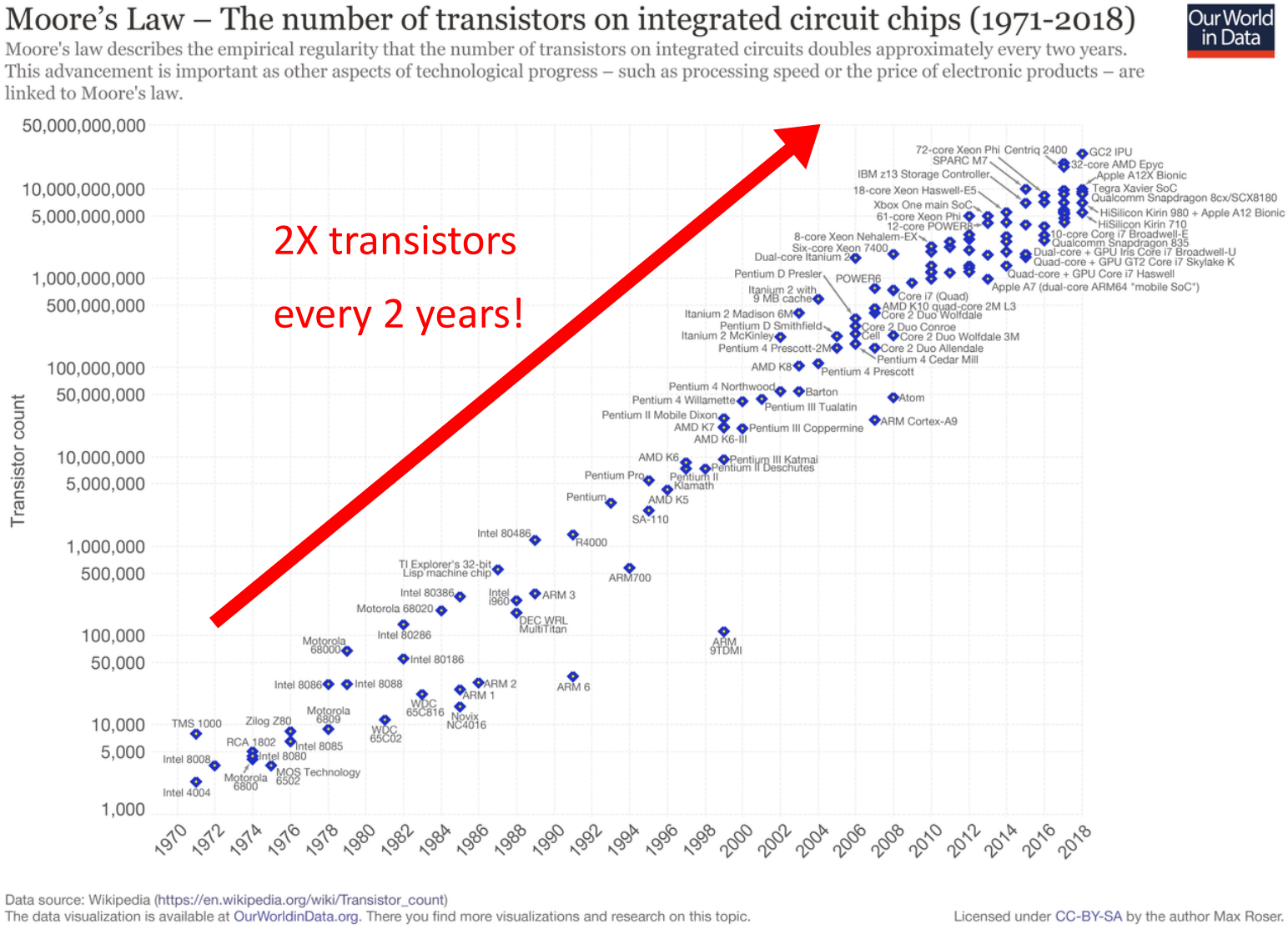

  - Trends can be *advances* in technology
  - Trends can be *constraints* technology couldn't overcome
- "Technology" in CPU design refers to the physical layer
  - Manufacturing technology used for transistor implementation


# Advances in Technology

- Technology has been advancing at lightning speed
- Architecture and IT as a whole were beneficiaries

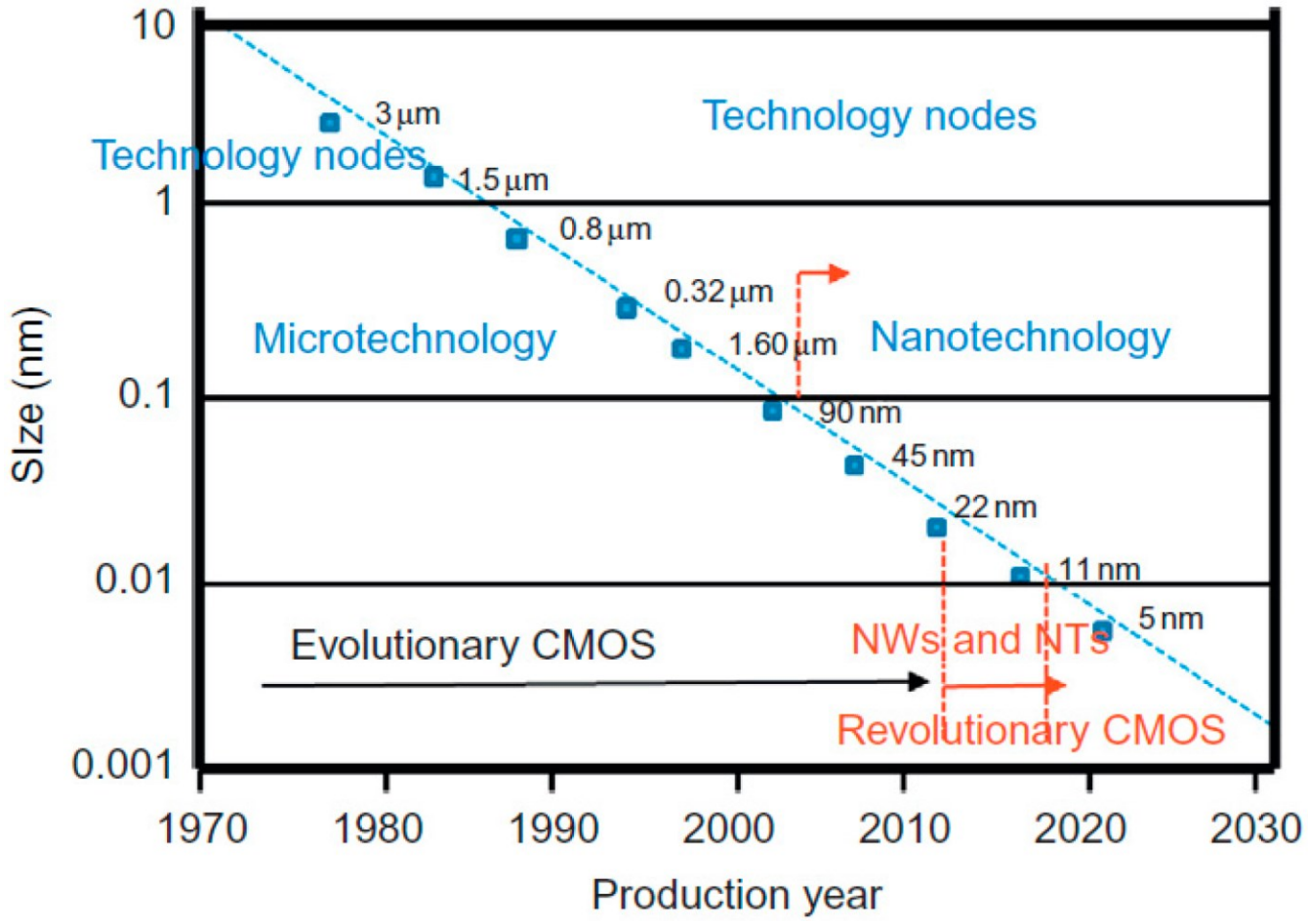
- Technology advance is summarized by *Moore's Law*
  - You probably heard of it at some point. Something about …
  - "X doubles every 18-24 months at constant cost"
- Is X:
  - Computer performance?
  - CPU clock frequency?
  - Transistors per chip?

# 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.

# Miniaturization of Transistors




Data source: Radamson, H.H.; He, X.; Zhang, Q.; Liu, J.; Cui, H.; Xiang, J.; Kong, Z.; Xiong, W.; Li, J.; Gao, J.; Yang, H.; Gu, S.; Zhao, X.; Du, Y.; Yu, J.; Wang, G. Miniaturization of CMOS. *Micromachines* **2019**, *10*, 293.

- Moore's Law has been driven by transistor miniaturization

# Future of Moore's Law

- The semiconductor industry has produced roadmaps
  - Semiconductor Industry Association (SIA): 1977~1997
  - International Technology Roadmap for Semiconductors (ITRS): 1998~2016
  - International Roadmap for Devices and Systems (IRDS): 2017~Present
- IRDS Lithography Projection (2020)

| Year of Production | 2018 | 2020 | 2022 | 2025 | 2028 | 2031 | 2034 |
|---|---|---|---|---|---|---|---|
| Technology Node (nm) | 7 | 5 | 3 | 2.1 | 1.5 | 1.0 | 0.7 |

  - Looks like Moore's Law will continue into foreseeable future
  - IRDS does not project significant increase in CPU chip size
  - Increases in transistors will come from *transistor density*

# Moore's Law and Performance


- Million-dollar question:
  Did Moore's Law result in higher performance CPUs?

- What do you think? Are computers getting faster?
  - If not, why do you think so? What examples support your view?
  - If yes, is speed of improvement accelerating or decelerating?

# What happened to Performance?

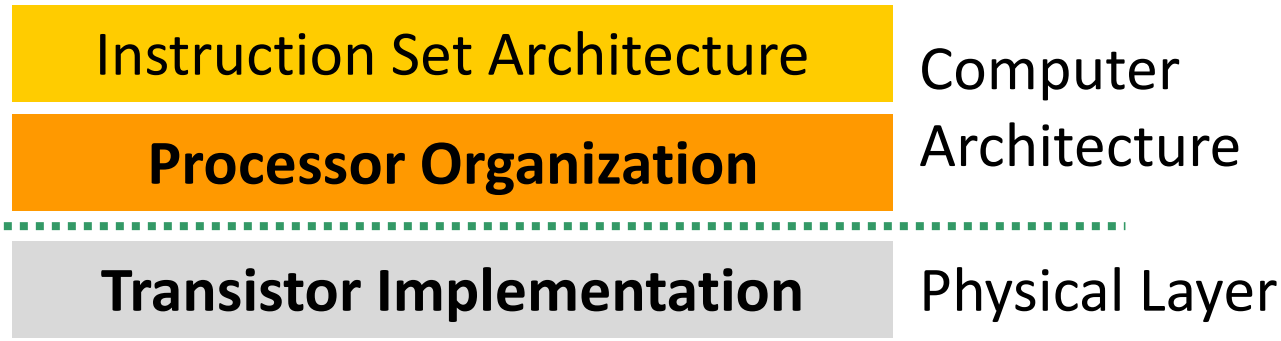

- Did it improve exponentially like Moore's law?
  - Did transistor count translate to performance?

- What do you think?  Are computers getting faster?

# Moore's Law and Performance

- Million-dollar question:
  Did Moore's Law result in higher performance CPUs?

- Law impacts both architecture and physical layer




  - Processor Organization: many more transistors to use in design
  - Transistor Implementation: smaller, more efficient transistors

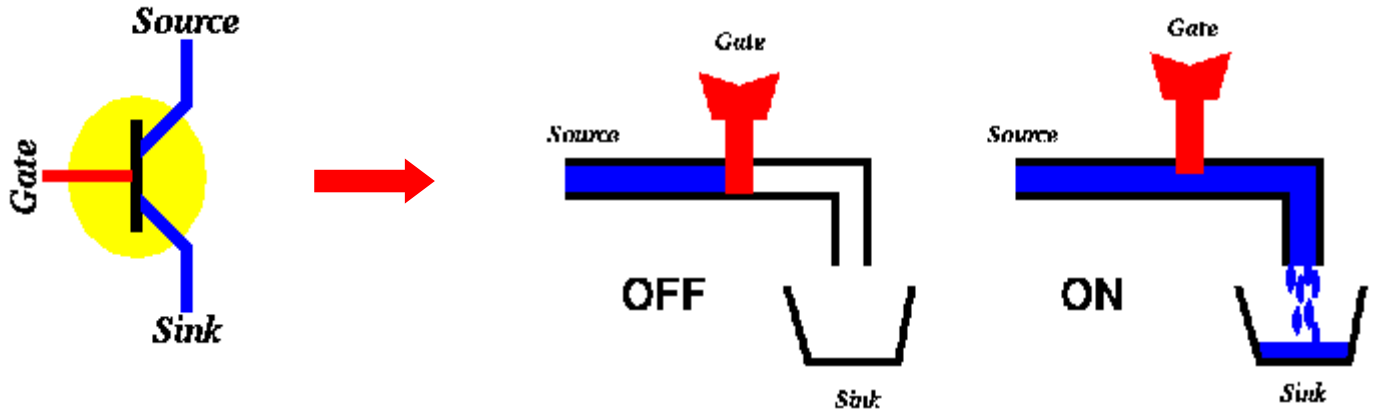
# Moore's Law Impact on Architecture

- So where did architects use all those transistors?
- Well, we will learn this throughout the semester ☺
  - Pipelining
  - Parallel execution
  - Prediction of values
  - Speculative execution
  - Memory caching
  - In short, they put it to good use to improve performance
- Let's go on to impact on the physical layer for now

# Moore's Law Impact on Physical Layer

- So did Moore's Law result in faster transistors?
  - Or did it result in (God forbid) slower transistors?

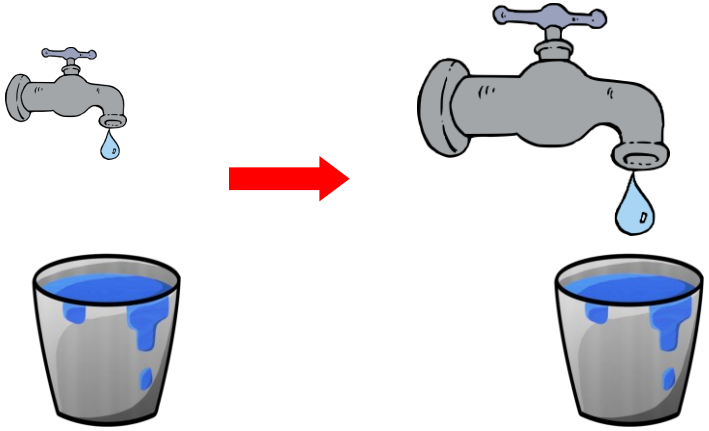
- What decides the speed of transistors anyway?
- Transistors are like faucets!



# Speed of Transistors

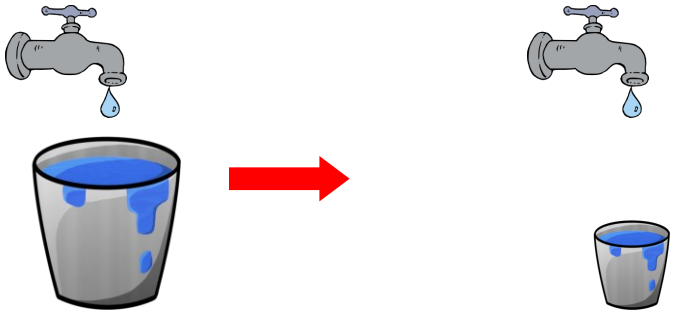
- So how do you make a transistor go faster?

1. *Increase* the water pressure (*voltage*)



- a.k.a DVFS (Dynamic Voltage Frequency Scaling)
- a.k.a. Overclocking

2. *Decrease* the bucket size (*transistor size*)



- What happens according to Moore's Law at each generation

# Thermal Design Power Limits Speed


- But there is a limit: *Thermal Design Power* (TDP)
- TDP: Maximum heat that cooling system can handle
  - Since heat is proportional to power, this limits power
  - Cooling system hasn't improved much over generations (Typically a CPU cooling fan attached with thermal paste)

- CPU Power = $N * CFV^2$
  - N = Number of transistors
  - C = Capacitance (proportional to transistor size)
  - F = Frequency (CPU frequency)
  - V = Voltage (CPU operating voltage)

# 1. TDP Impact on DVFS

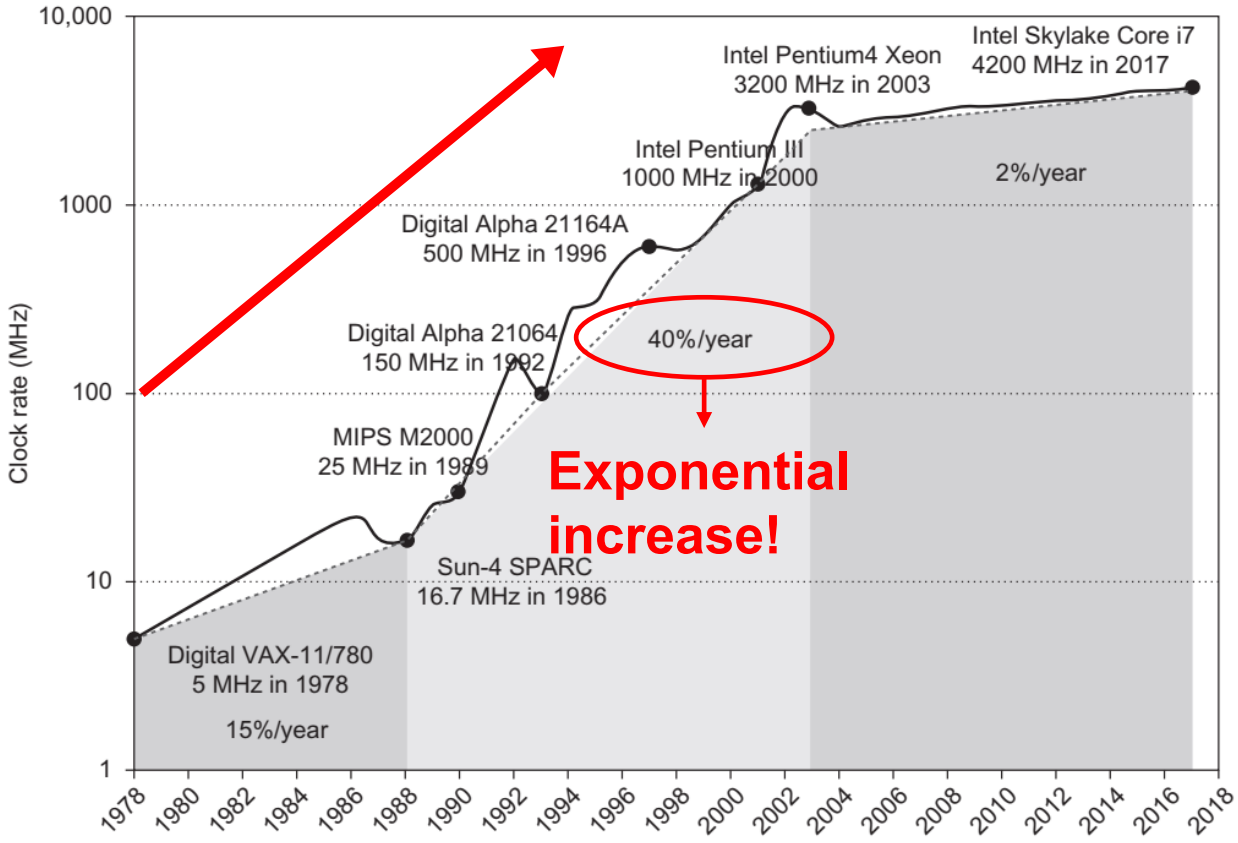
1. Increasing voltage / increasing frequency (DVFS)
   - CPU Power = N * $CFV^2$
     - N = Number of transistors ⇔ (same number of transistors)
     - C = Capacitance ⇔ (same transistor size)
     - F = Frequency ⇧ (faster transistor due to higher voltage)
     - V = Voltage ⇧
   - CPU power / heat always increases with DVFS
   - Can't overclock unless there is headroom in TDP

# 2. TDP Impact on Moore's Law

2. Transistor miniaturization (Moore's Law)
   - CPU Power = N * CFV$^2$
     - N = Number of transistors ⇧ (increase in transistor density)
     - C = Capacitance ⇩ (smaller transistors)
     - F = Frequency ⇧ (faster transistor due to smaller size)
     - V = Voltage ⇩ (lower to meet TDP while maintaining frequency)
   - All factors balance each other out so power remains constant
   - Important since TDP does not improve across generations

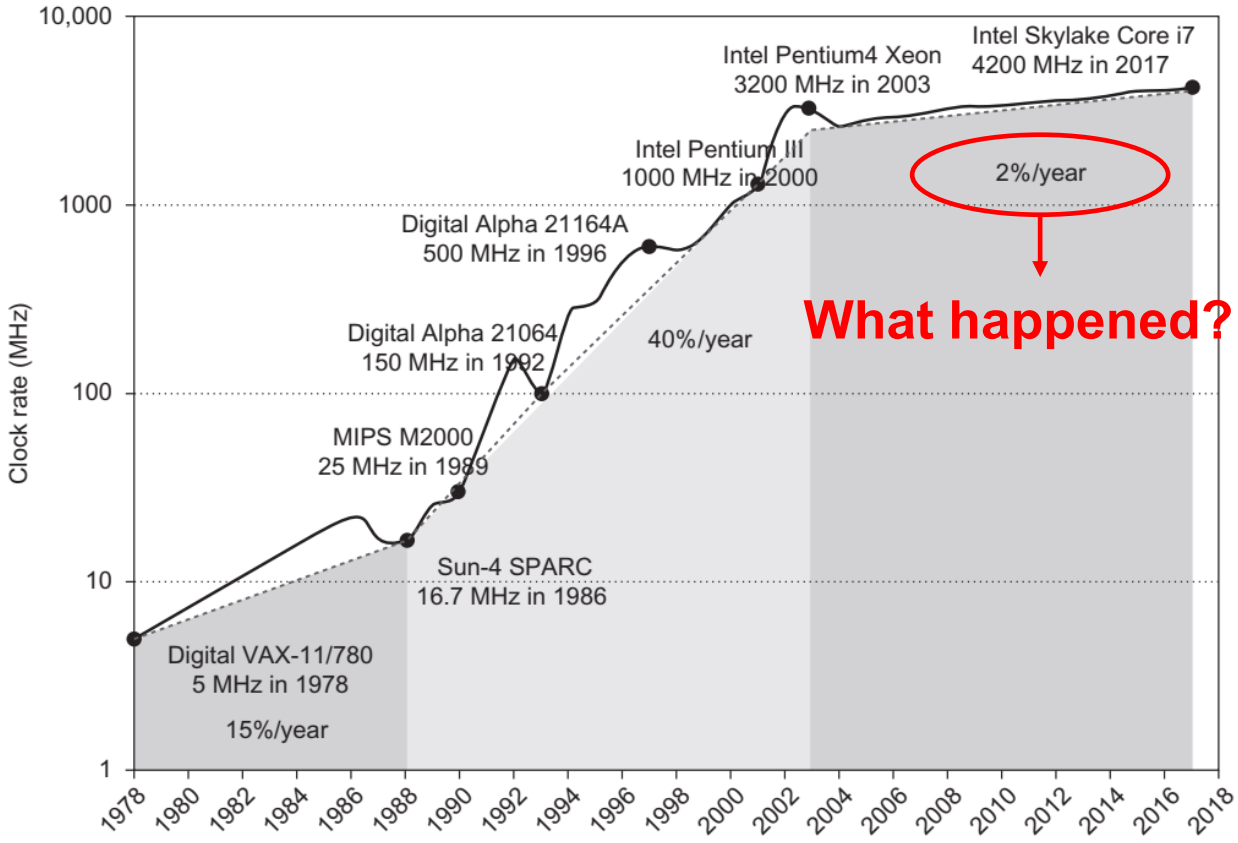
- Result: transistor speed improves under Moore's Law
  - While power remains constant (stays within TDP)
  - This transistor scaling trend is called *Dennard Scaling*

# Improvements to CPU Frequency




- Improvements in large part due to Dennard Scaling
  - Processor design also contributed but we'll discuss later

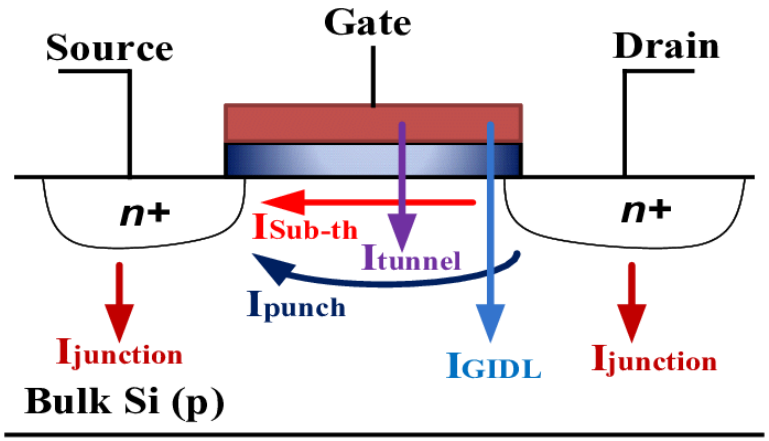
# End of Dennard Scaling




- Suddenly around 2003, frequency scaling stops
  - This is called the *End of Dennard Scaling*

# End of Dennard's Scaling Cause

- Why did it end?
  - When transistors got extremely small, they became leaky
  - Small amount of electricity still flowed even when switch is off




  - This "leakage power" in itself caused CPU power to increase
  - Needed to maintain operating voltage to stop leakage
    (Why that stops leakage is too complicated to explain here)

# End of Dennard Scaling Impact

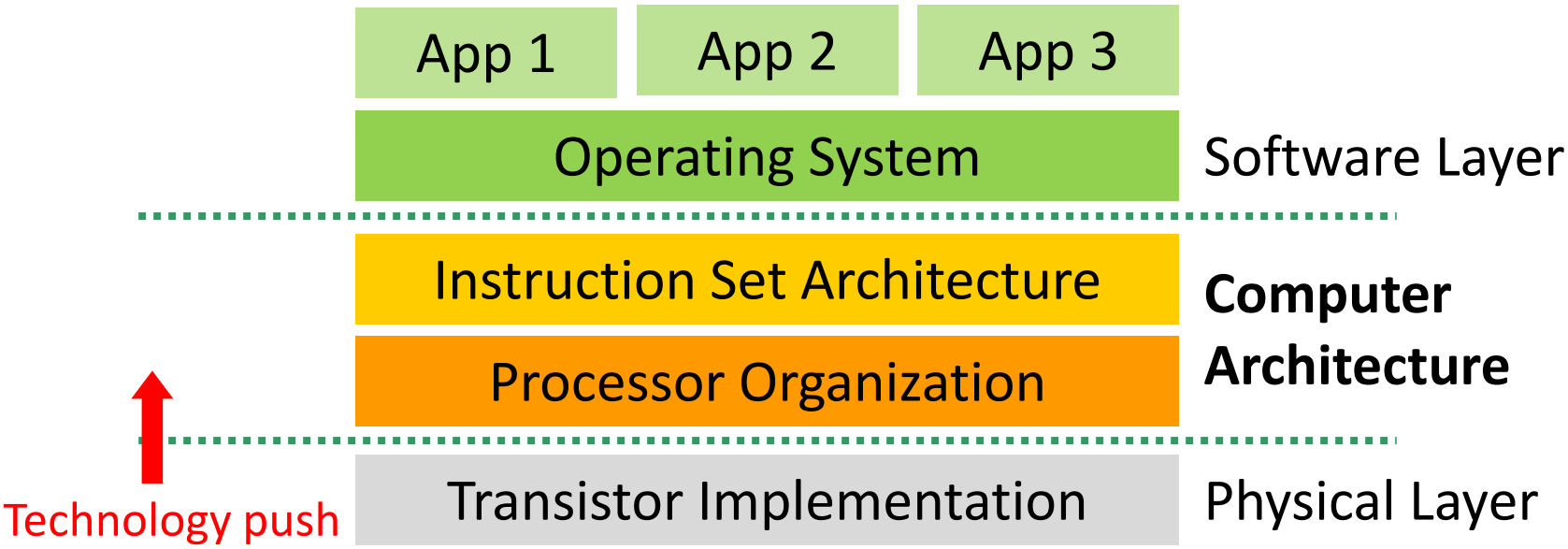
- Transistor miniaturization (Moore's Law)
  - CPU Power = N * CFV$^2$
    - N = Number of transistors ⇧ (increase in transistor density)
    - C = Capacitance ⇩ (smaller transistors)
    - F = Frequency ⇧ (faster transistor due to smaller size)
    - V = Voltage ⇔ (due to end of Dennard's scaling)
  - Now power increases at each generation
  - That is unsustainable due to TDP, so architects must either
    - Reduce frequency (this has already happened)
    - Reduce number of transistors (this may yet happen)
    - Do some smart processor design that reduces power otherwise
  - Reduce transistor numbers!? Are you serious? Yes.
    - To be precise, reduce transistors that are active at a time
    - The parts that are inactive are called *dark silicon*

# Free Ride is Over


- "Free" speed improvements from physical layer is over
  - How many times do I have to say it: Dennard Scaling is dead
- Now it's up to architects to improve performance
  - Remember Moore's Law is still alive and well
  - Architects are flooded with extra transistors each generation
- Now is a good time to discuss technology constraints
  - Since we already mentioned one: Thermal Design Power (TDP)

# Technology Constraints

- Constraints in technology push architecture too

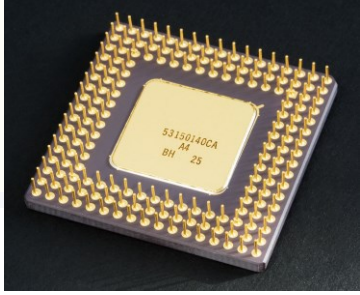



  - *Power Wall*: Thermal Design Power (TDP) constraint
  - *Memory Wall*: Constraint in bandwidth to memory
  - …
- Processor must be designed to meet all constraints

# Power Wall

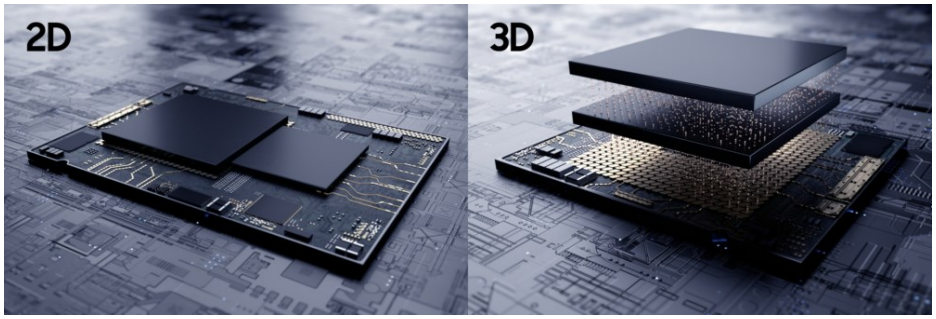
- The discrepancy between power demand and TDP
  - Architects have plenty of transistors but not enough power
- To eke out extra performance out of a program
  - Architects must use increasingly complex logic
  - Complex logic is very power hungry
  - Diminishing returns on performance when power is invested
- Which direction did the Power Wall push architecture?
  - *Multi-cores*: Run multiple programs (threads) on simple cores
  - *GPUs*: Run simple programs on massively parallel compute units
  - *Caches*: Memory caches are very power efficient structures
  - Will discuss all later in the lecture!

# Memory Wall



- Discrepancy in memory *bandwidth* demand and supply
- Bandwidth demand increases on each CPU generation
  - Bandwidth demand is proportional to compute power of CPU
- Bandwidth supply is mostly constant across generations
  - Supply is determined by number of CPU pins (limited by CPU size)

- Which direction did this push architecture?
  - *More Caches*: Memory caching reduces bandwidth demand
  - *3D Cubes*: Stack CPU on top of memory → increases bandwidth

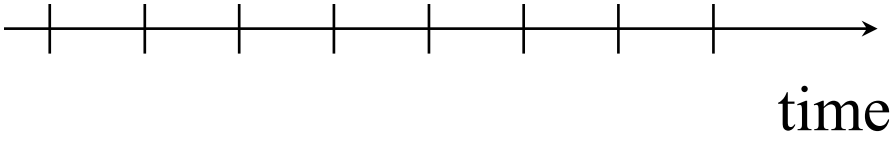


# Opportunities for Speed Improvement


- So Dennard Scaling is dead
- And we are walled in by technology constraints
- Where do architects go look for performance?

# Components of Execution Time

- Processor activity happens on clock "ticks" or cycles




  - On each tick, bits flow through logic gates and are latched

- Execution time = $\frac{\text{seconds}}{\text{program}}$

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

$$= \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instructions}} \times \frac{\text{seconds}}{\text{cycle}}$$

# Improving Execution Time

$$\frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instructions}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Improving $\frac{\text{seconds}}{\text{cycle}}$:
  - Clock frequency = $\frac{\text{cycles}}{\text{second}}$ = reverse of $\frac{\text{seconds}}{\text{cycle}}$
  - Higher clock frequency (GHz) leads to shorter exec time
  - More *pipelining* can lead to higher frequencies
- Improving $\frac{\text{cycles}}{\text{instructions}}$:
  - Also known as CPI (Cycles Per Instruction)
  - IPC (Instructions Per Cycle) = $\frac{\text{instructions}}{\text{cycles}}$ = reverse of $\frac{\text{cycles}}{\text{instructions}}$
  - *Superscalars* can execute multiple instructions per cycle
- Improving $\frac{\text{instructions}}{\text{program}}$:
  - Can modify ISA to more efficiently encode computation
  - *GPUs* can do a lot of work with much less instructions

# What about Other Performance Goals?

- We talked a lot about execution speed
- But there are other performance goals such as:
  - Energy efficiency
  - Reliability
  - Security
  - …
- In this class, we will mainly focus on speed
  - Not that other goals are not important; we don't have time
  - Performance will be used synonymously as speed