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

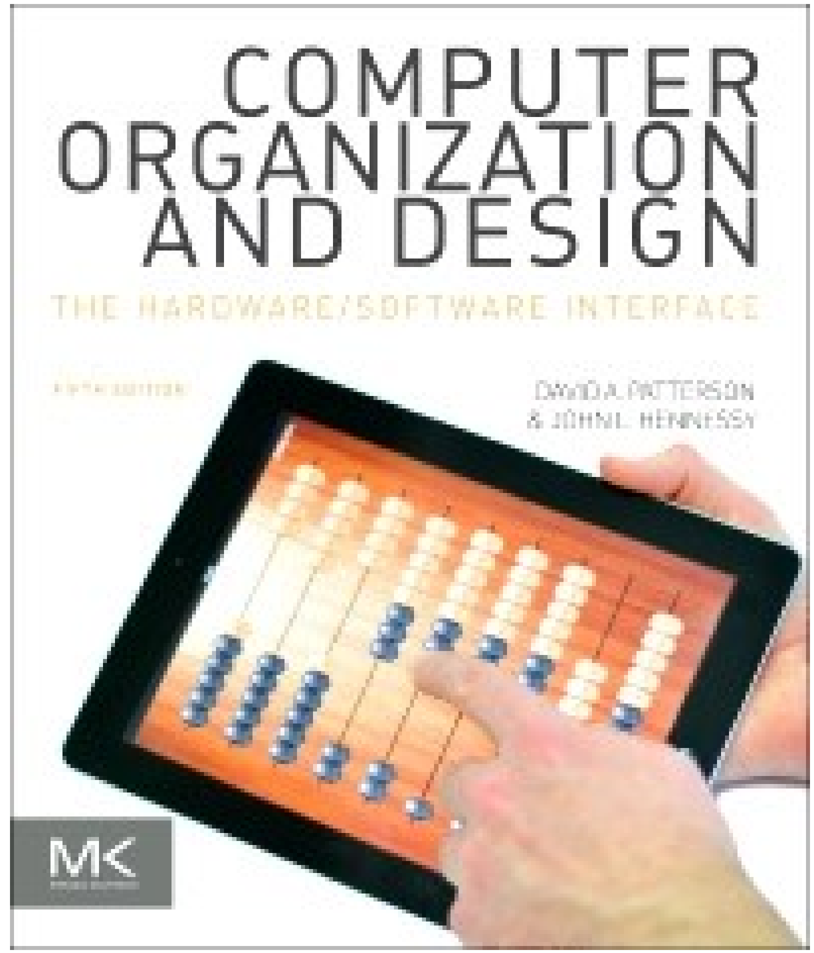
Course Introduction

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

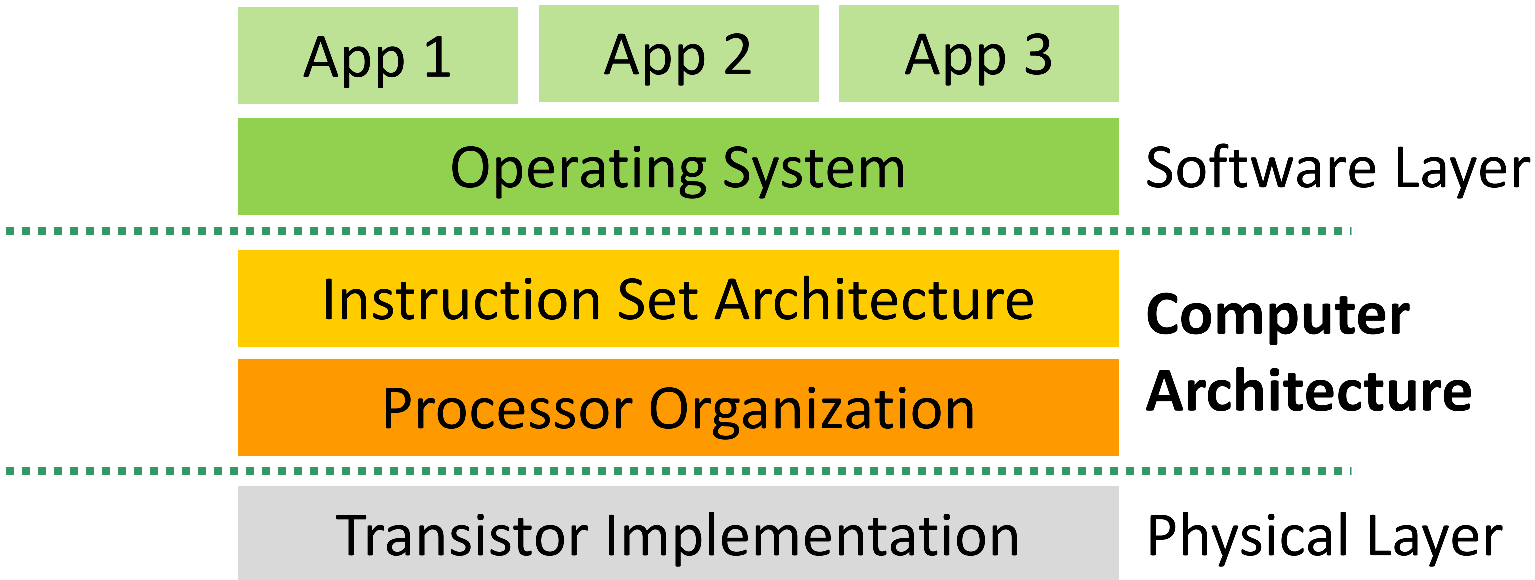
TODO

- 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)

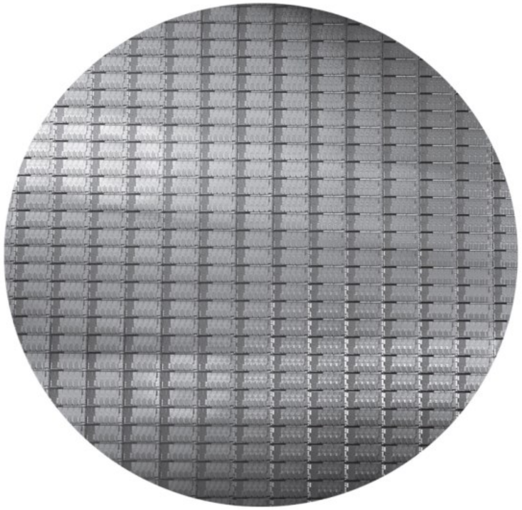
Computer Architecture Defined



- 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

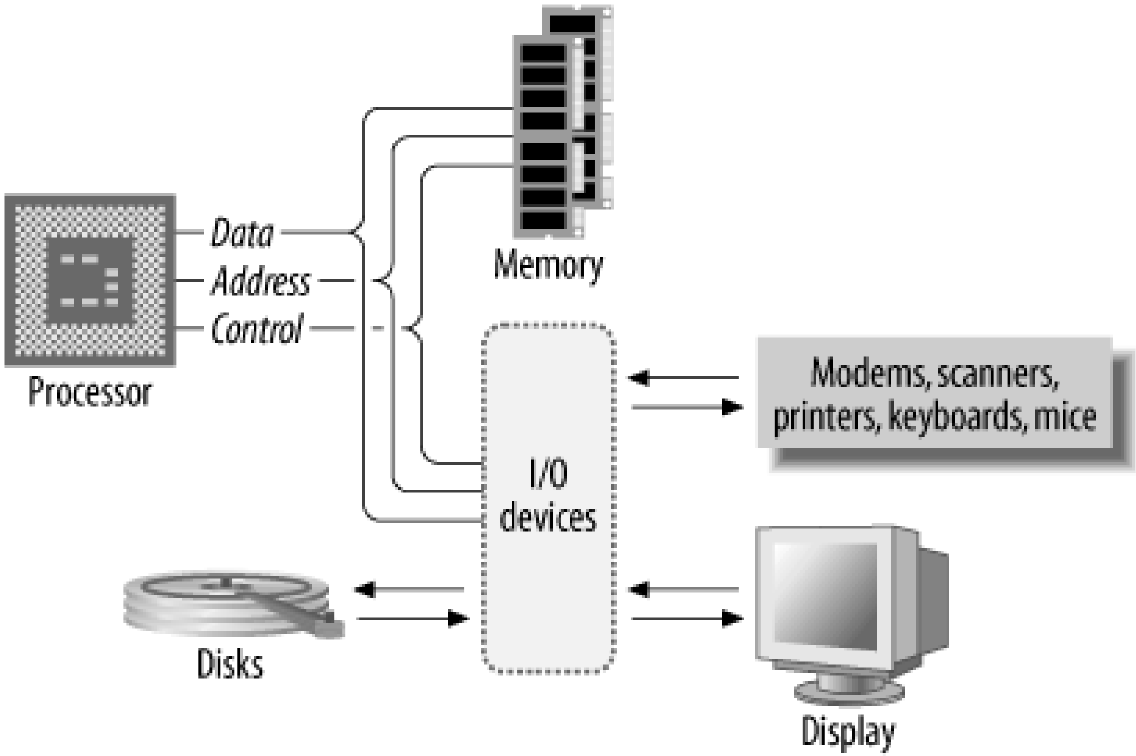
- Physical layer is beyond the scope of the class



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

Scope of Class

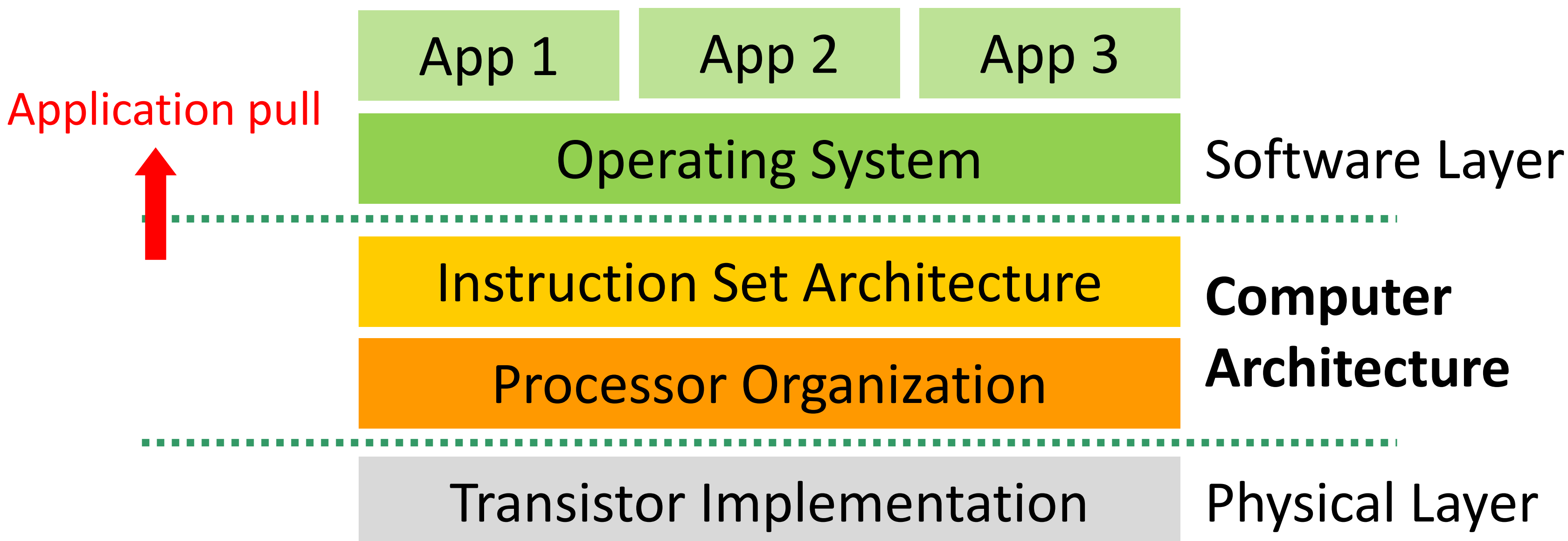
- Computer architecture is part of system architecture



- Other components beside processor is beyond the scope

Application Pull

- Different applications pull in different directions



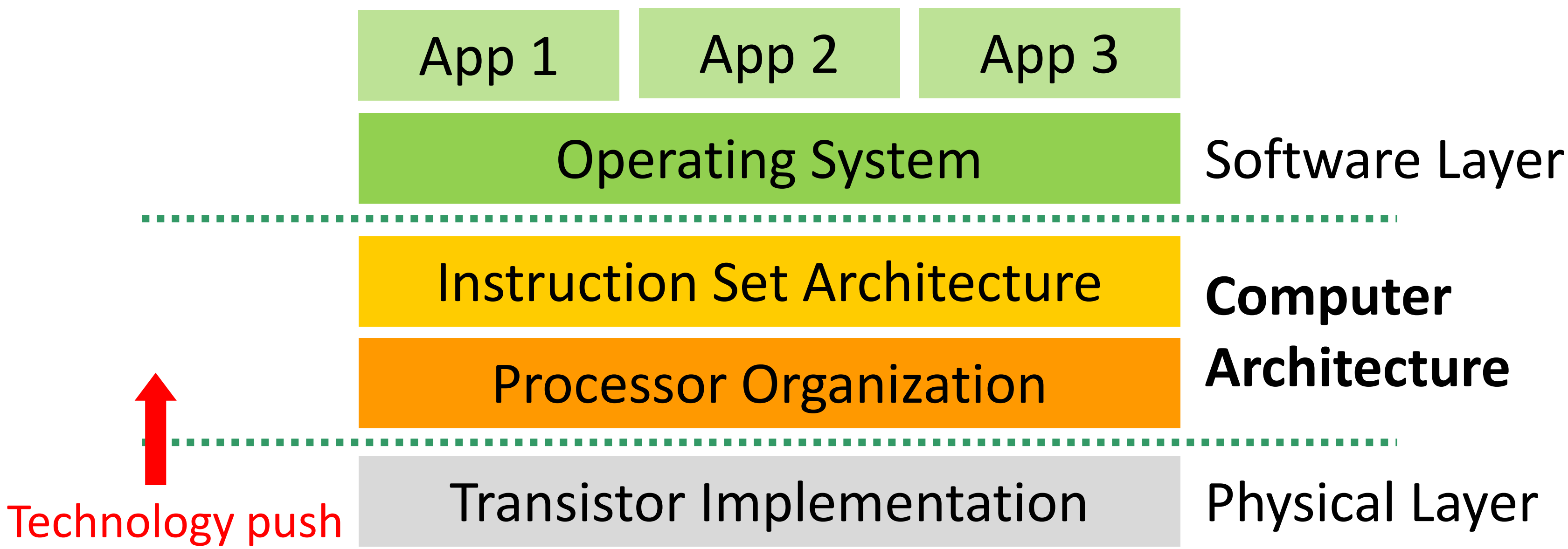
  - Real-time app (e.g. Game): *Short latency*
  - Server app: *High throughput*
  - Mobile app: *High energy-efficiency* (battery life)
  - Mission critical app: *High reliability*
- An app typically has multiple goals that are important

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

Technology Advances

Advances in Technology

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

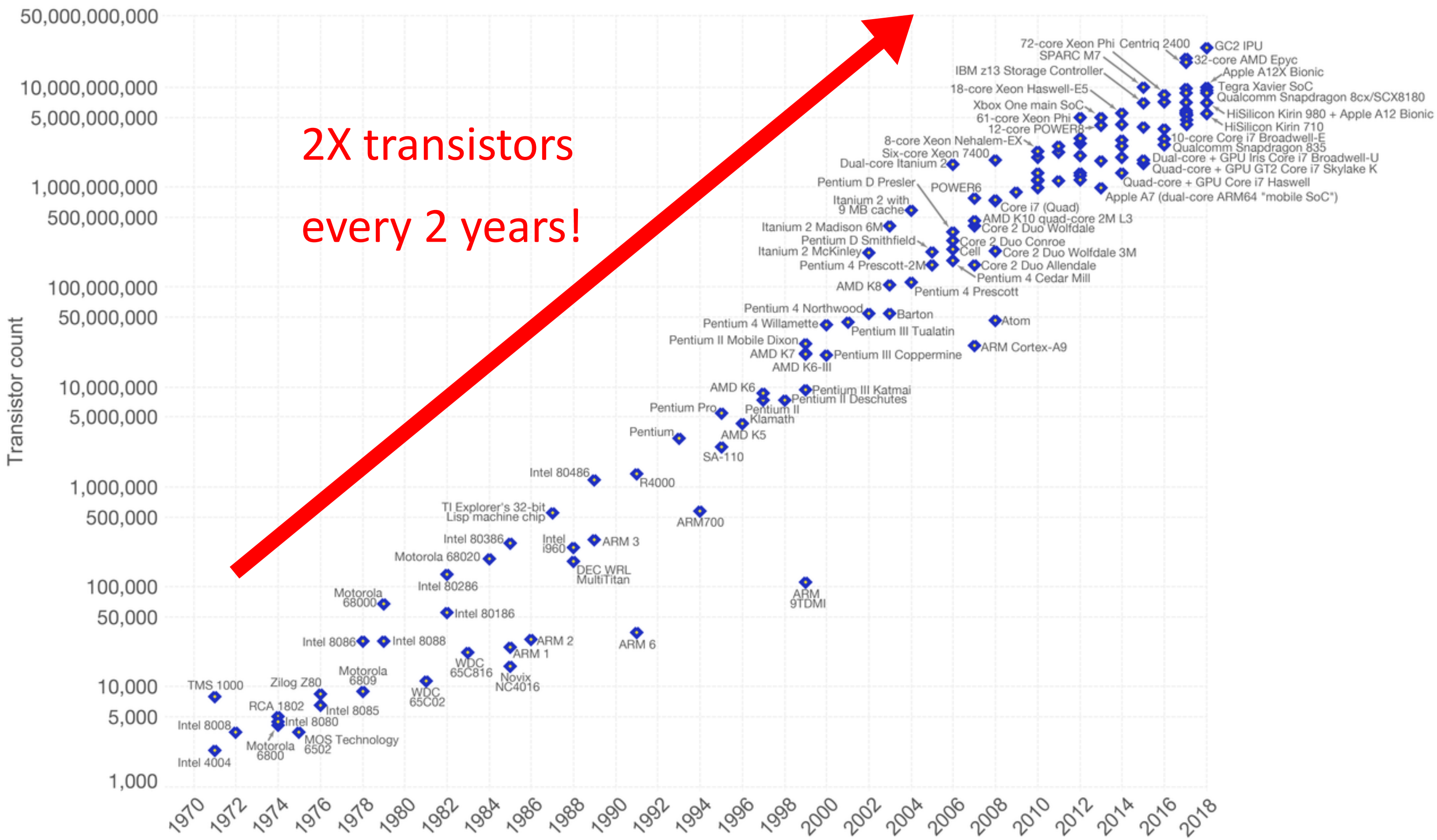
- Technology advance is summarized by *Moore's Law*
  - You probably heard of it at some point. Something about …
  - "X doubles every 18-24 months at constant cost"
- Is X:
  - CPU performance?
  - CPU clock frequency?
  - Transistors per CPU chip?
  - Area of CPU chip?

Moore’s Law

Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

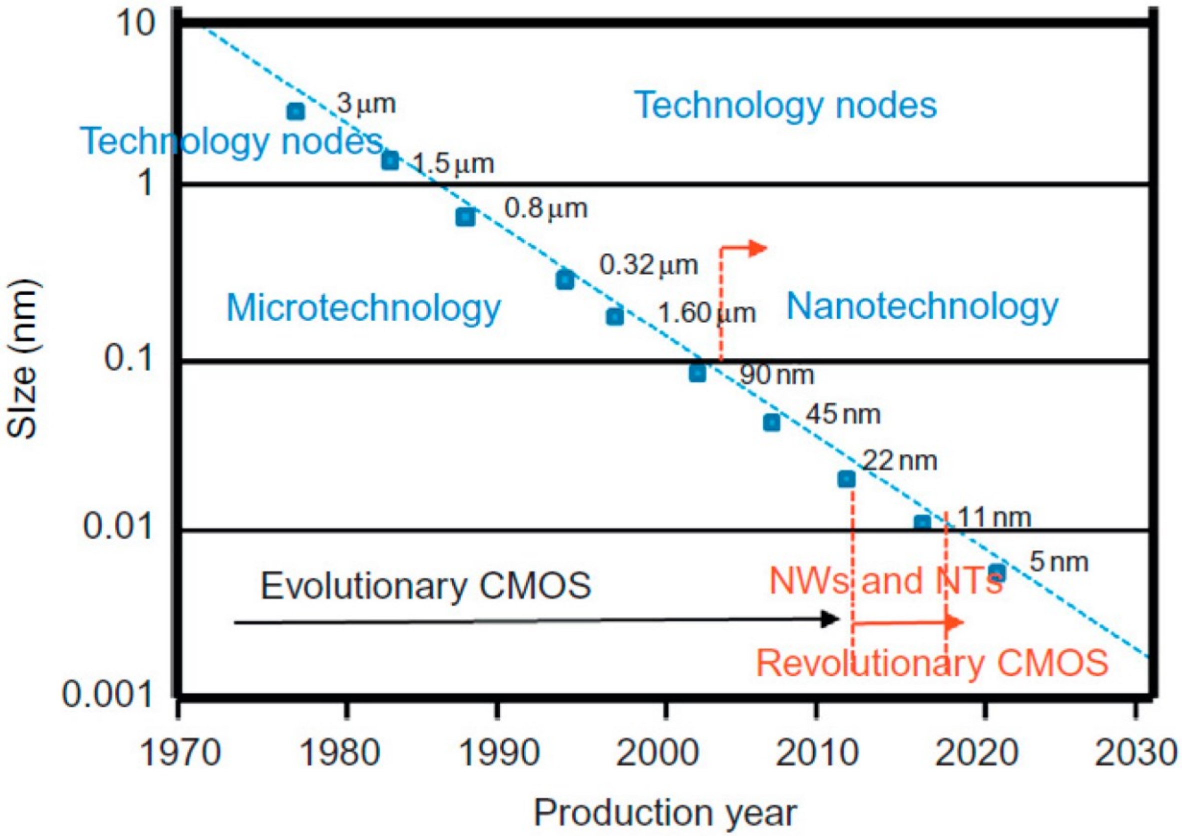
Our World in Data

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 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.

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
  - CPU chip area hasn’t changed much

Future of Moore's Law

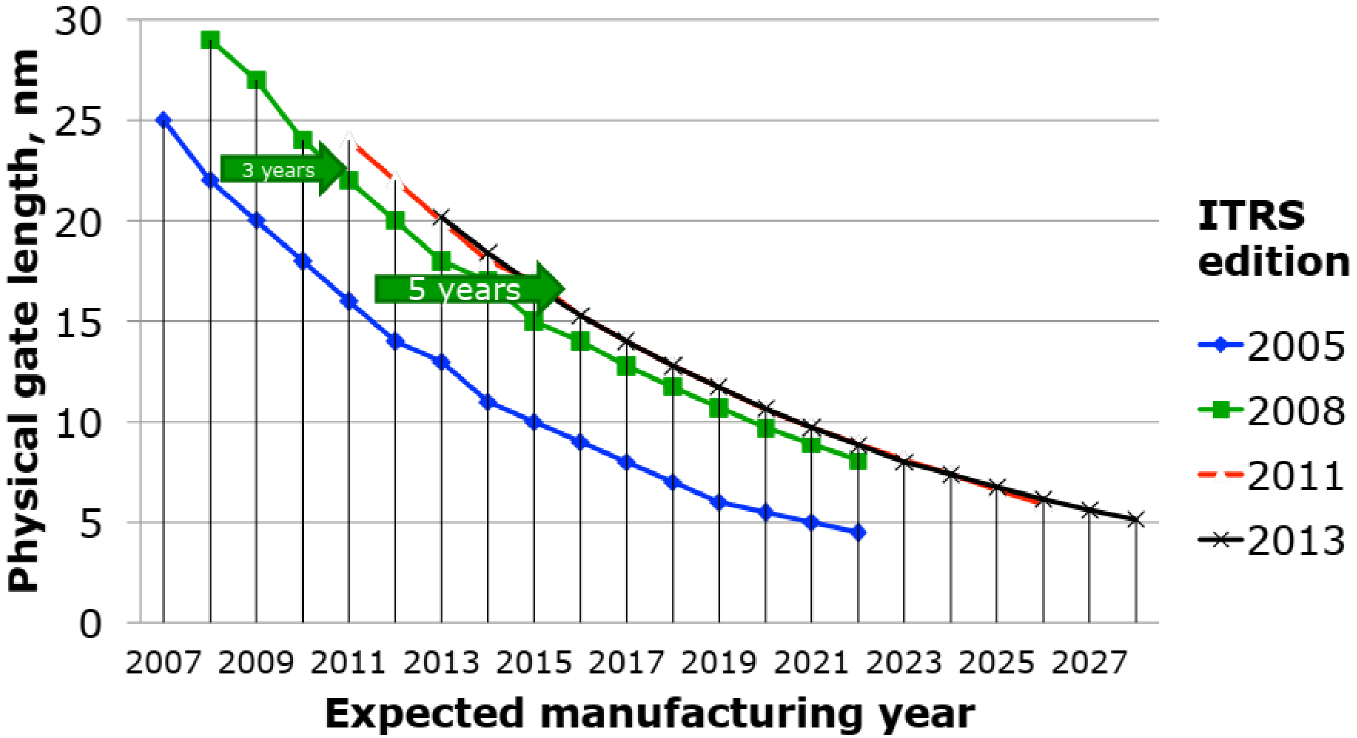
- The semiconductor industry has produced roadmaps
  - Semiconductor Industry Association (SIA): 1977~1997
  - International Technology Roadmap for Semiconductors (ITRS): 1998~2016
  - International Roadmap for Devices and Systems (IRDS): 2017~Present
- IRDS Lithography Projection (2020)

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*

IRDS isn't Perfect

- ITRS (predecessor of IRDS) has made corrections before



  - After all, you are trying to predict the future
  - But architects rely on the roadmap to design future processors

Moore’s Law and Performance

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

- Please go to your respective Teams chat groups
  - But stay in the Zoom room and use only chat on Teams
  - To have chat content accessible to asynchronous students

1. Get to know each other
2. And then try to answer the following questions:
   - What do you think? Are CPUs getting faster?
   - If not, why do you think so? If yes, again why do you think so?
3. After 10 minutes, we will share discussions with class

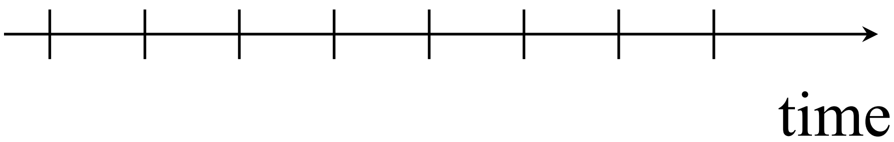
Are CPUs getting Faster?

- Yes!
- Clock speeds are increasing, power draw is decreasing – Andrew
- More cores – Jason
- Visual Studio compilation is actually faster - Nick

- No!
- Clock speeds are plateauing recently – Jason
- Seems stagnant from user's perspective e.g. Visual Studio - Josh

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 = $\dfrac{\text{seconds}}{\text{program}}$

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

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

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

- Improving $\frac{\text{cycles}}{\text{instructions}}$:
  - Also known as CPI (Cycles Per Instruction)
  - IPC (Instructions Per Cycle) = $\frac{\text{instructions}}{\text{cycles}}$ = reverse of $\frac{\text{cycles}}{\text{instructions}}$
  - Higher IPC leads to shorter execution time

- Improving $\frac{\text{instructions}}{\text{program}}$:
  - Less instructions leads to shorter execution time
  - ISAs that do a lot of work with one instruction shortens time

Moore’s Law and Performance

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

- Law impacts both architecture and physical layers

Instruction Set Architecture	Computer Architecture
Processor Organization	
Transistor Implementation	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 were used to improve frequency or IPC

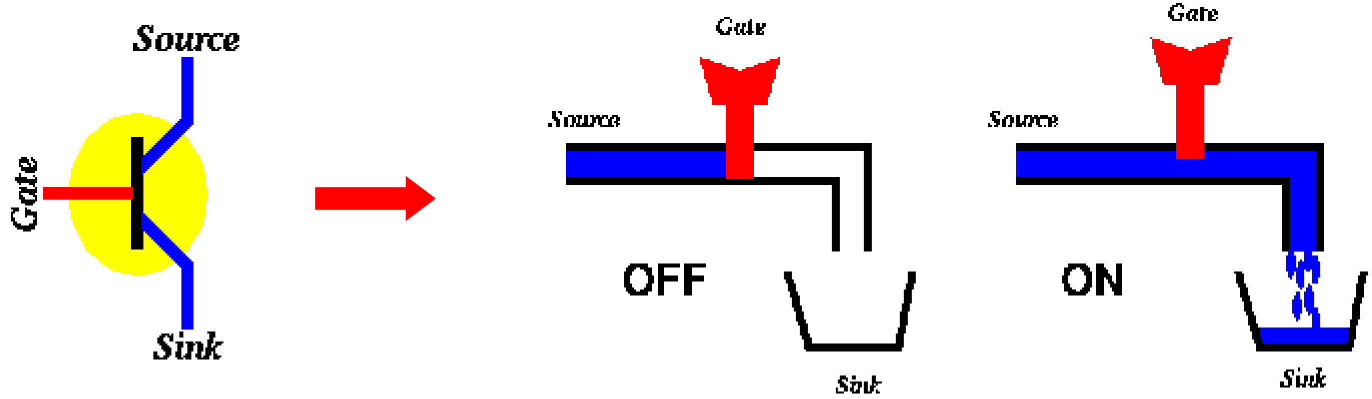
- Let’s go on to impact on the physical layer for now

Moore’s Law Impact on Physical Layer

- CPU frequency is also impacted by transistor speed
  - As well as how many transistors are in between clock ticks (which is determined by processor organization)
- So did Moore’s Law result in faster transistors?
  - In other words, are smaller transistors faster?

Speed of Transistors

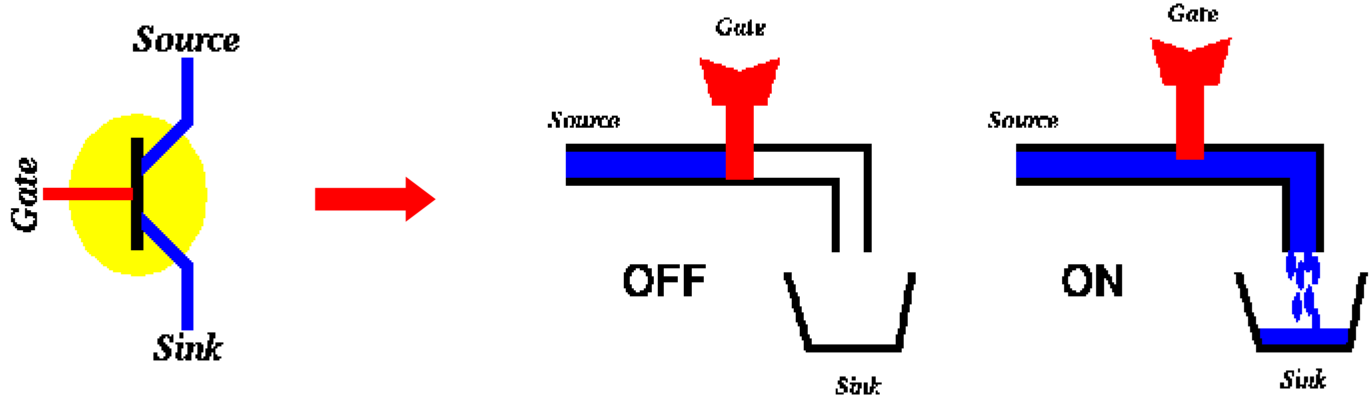
- Transistor 101: Transistors are like faucets!



- To make a transistor go fast, do one of the following:
  - Reduce distance from source to sink (*channel length*) ⇩
  - Reduce bucket size (*capacitance*) ⇩
  - Increase water pressure (*supply voltage*) ⇧

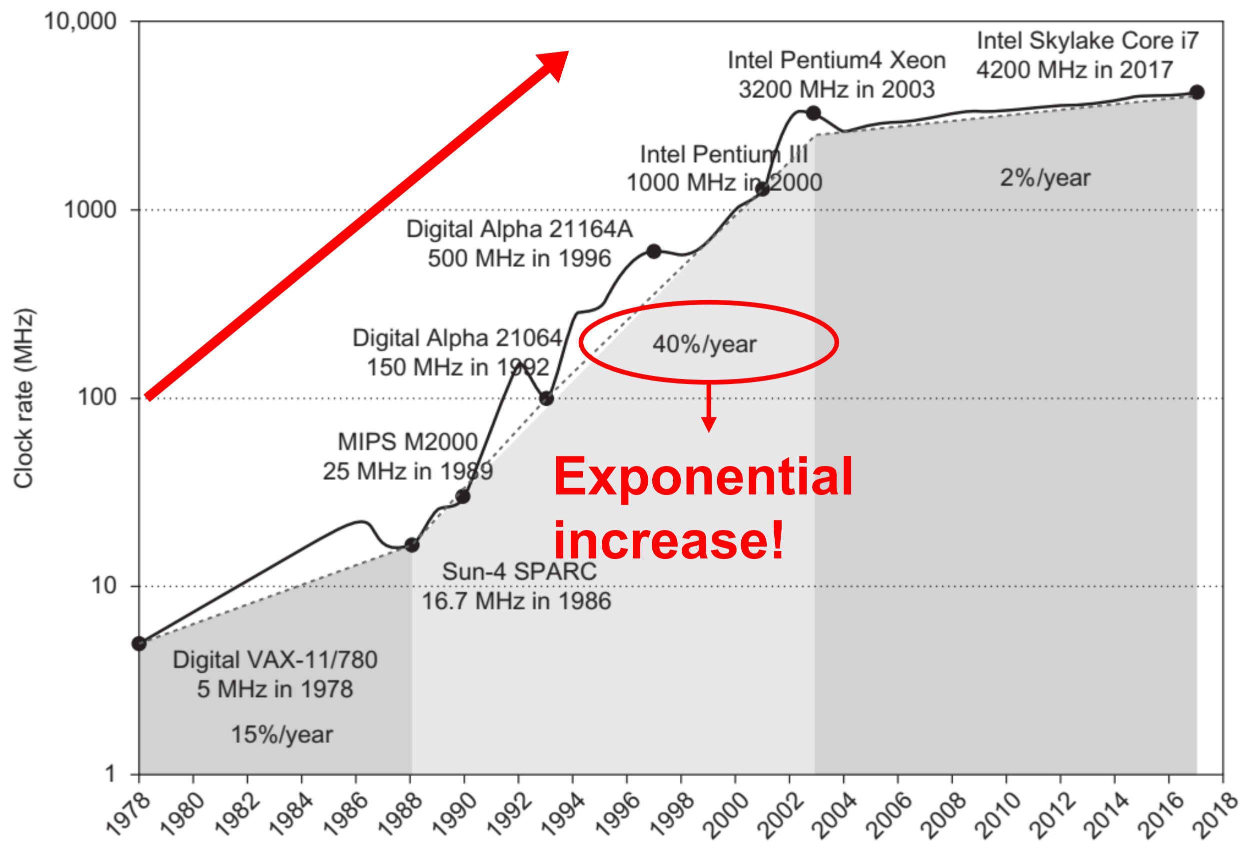
Smaller Transistors are Faster!

- Transistor 101: Transistors are like faucets!



- When a transistor gets smaller:
  - *Channel length* (channel resistance) is reduced ⇩
  - *Capacitance* is reduced ⇩
- So, given the same *supply voltage*, smaller is faster!
- So, did Moore's Law enjoy faster and faster frequencies?

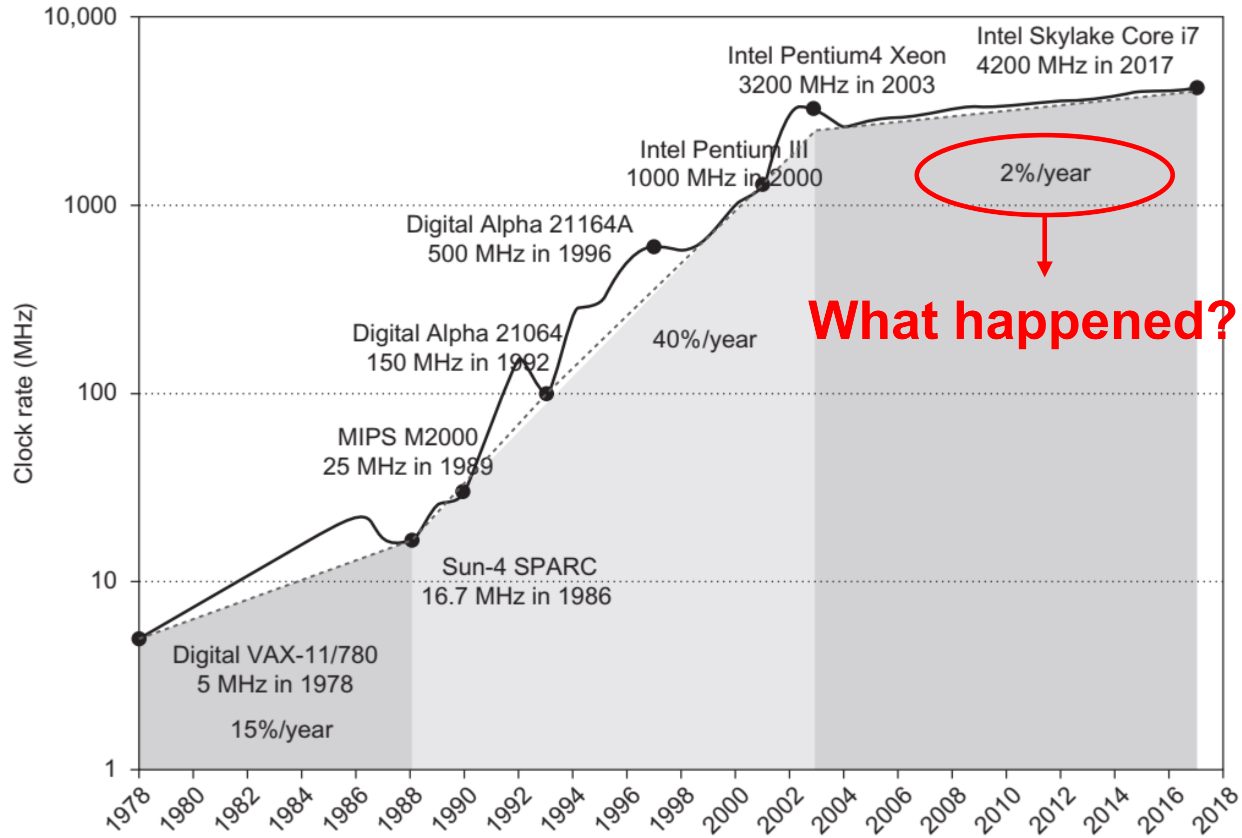
Yes, for a while …



Source: Computer Architecture, A Quantitative Approach (6th ed.) by John Hennessy and David Patterson, 2017

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

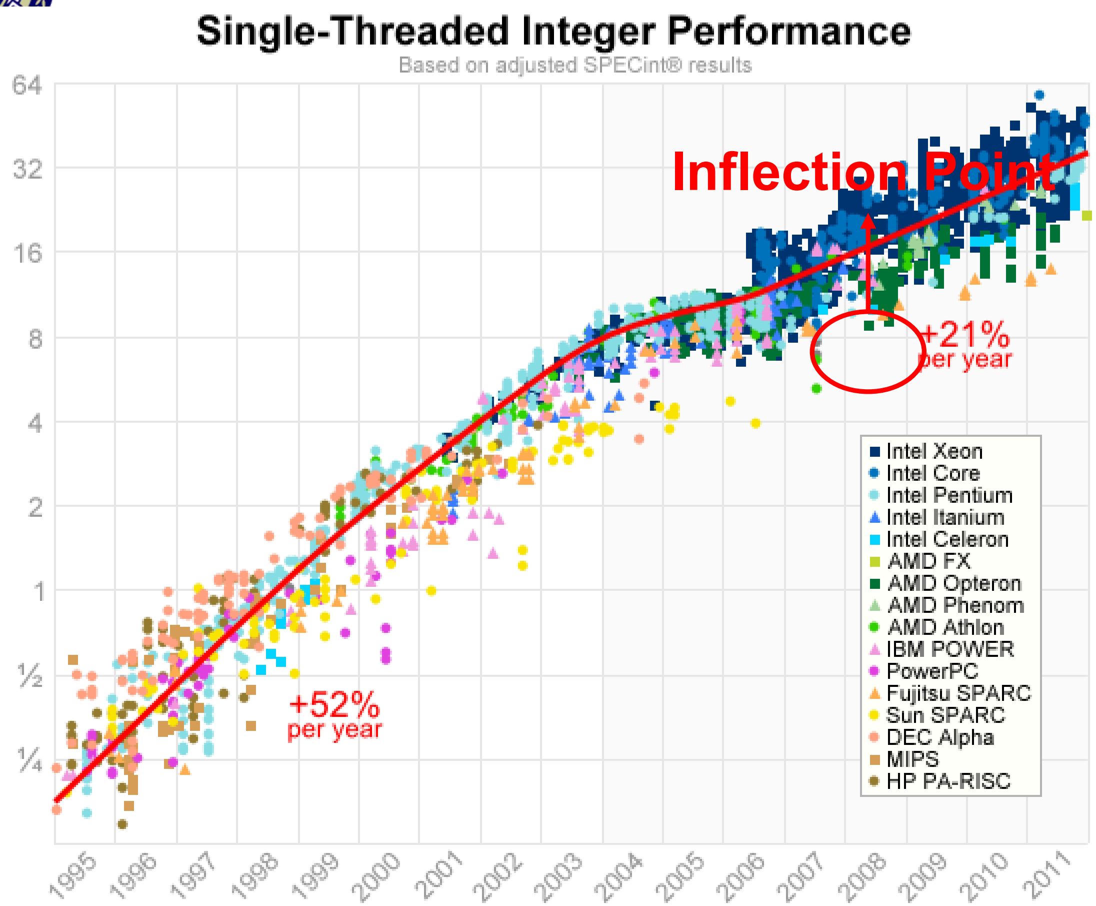
But not so much lately



Source: Computer Architecture, A Quantitative Approach (6th ed.) by John Hennessy and David Patterson, 2017

- Suddenly around 2003, frequency scaling stops

Dent in CPU Performance



Source: https://preshing.com/20120208/a-look-back-at-single-threaded-cpu-performance/

- This caused a big dent in CPU performance at 2003
- Improvements henceforth only came from architecture
  - From improvements to IPC (instructions per cycle)

So What Happened? TDP.

- *TDP* (*Thermal Design Power*):
  - Maximum heat (power) that cooling system can handle
  - Cooling system hasn't improved much over generations (Typically a CPU cooling fan attached with thermal paste)
- CPU Power = A * N * CFV^2 must be < TDP
  - A = Activity factor (% of transistors with activity)
  - N = Number of transistors
  - C = Capacitance ($\propto$ transistor size)
  - F = Frequency
  - V = Supply Voltage

- What happens to each factor with Moore's Law?

TDP and Moore's Law

- CPU Power = A * N * CFV² with Moore's Law
  - A = Activity factor
  - N = Number of transistors ⇧ ⇧
  - C = Capacitance (∝ transistor size) ⇩
  - F = CPU frequency ⇧
  - V = CPU Supply Voltage

- Reductions in C cannot offset increases in N and F

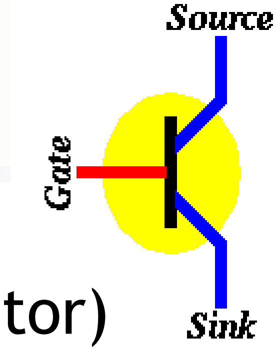
  Q) How did CPU frequency keep increasing up to 2003?

  A) By maintaining power through reductions in Voltage ⇩

  Q) Wait! Voltage reduction reduces frequency! (Transistor 101)

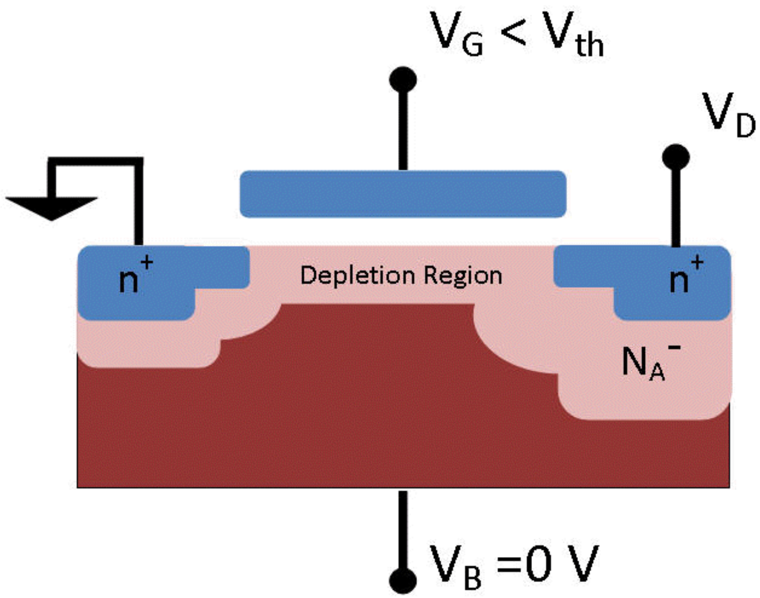
  A) Alright, time to do MOSFET 101

MOSFET 101

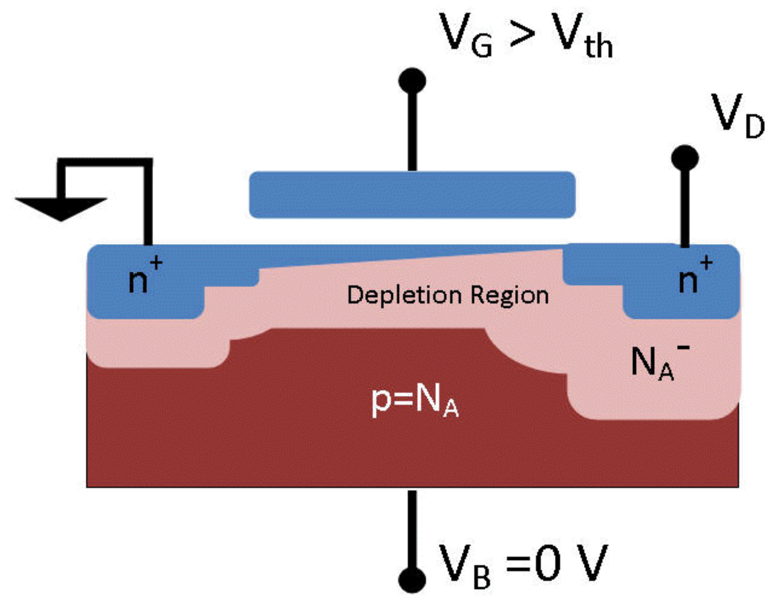


- MOSFET (Metal Oxide Silicon Field Effect Transistor)

[A MOSFET transistor switched off]



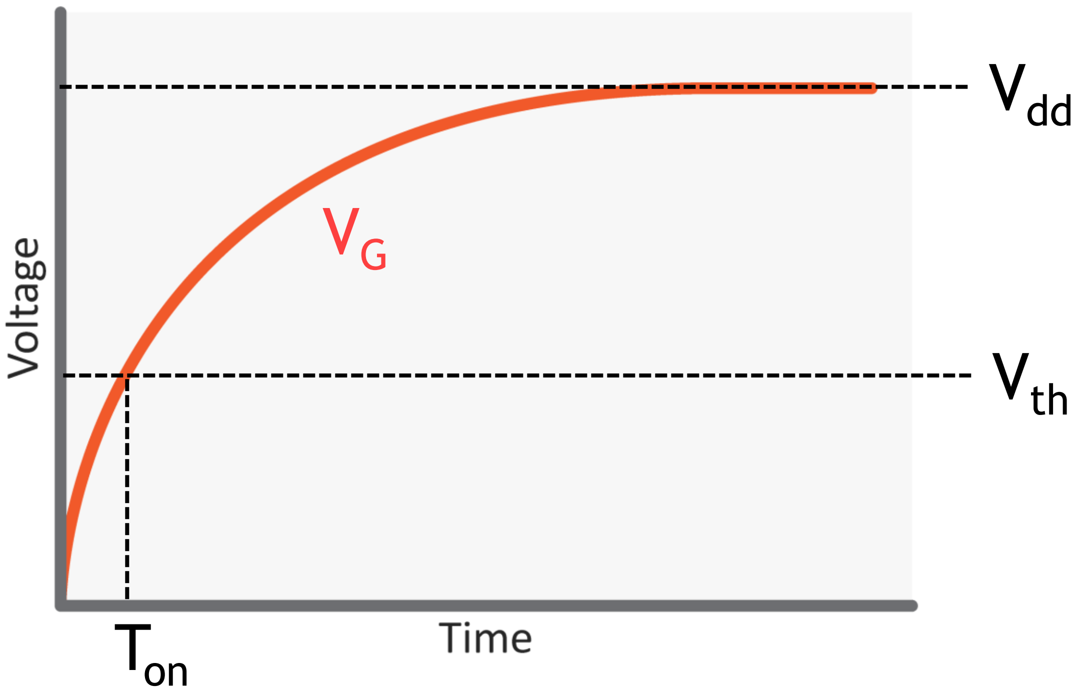
[A MOSFET transistor switched on]



- Gate is switched on when V_G reaches a threshold V_{th}
  - By creating a channel in depletion region through field effect
  - V_{th}: threshold voltage (minimum voltage to create channel)

MOSFET 101

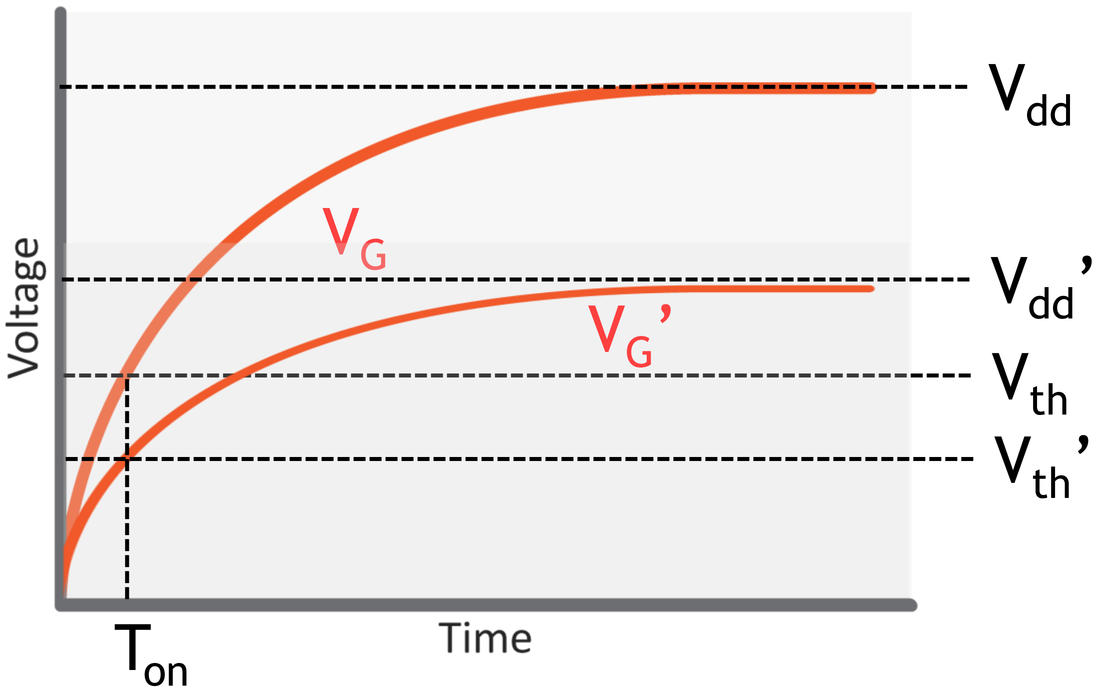
- RC charging curve of V_G



- Speed (T_{on}) is determined by V_{dd} if V_{th} is fixed
  - V_{dd} is the CPU supply voltage (the water pressure)
  - If V_{dd} is lower, V_G will reach V_{th} more slowly (low pressure)

MOSFET 101

- RC Charging Curve of V_G

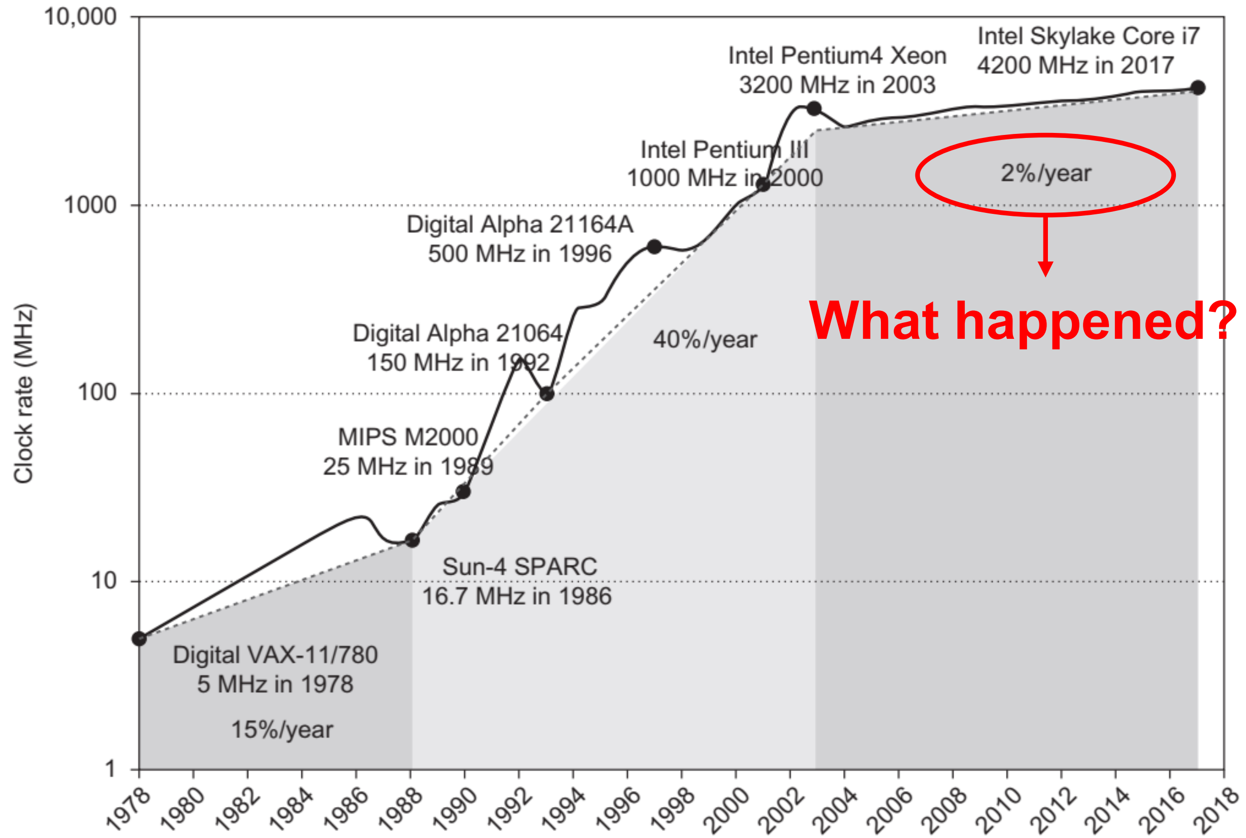


- Speed (T_{on}) is maintained while reducing V_{dd} to V_{dd}', if V_{th} is also reduced to V_{th}'

Dennard Scaling

- So in the end, this is what happens …
- CPU Power = A * N * CFV^2 with Moore's Law
  - A = Activity factor
  - N = Number of transistors ⇧ ⇧
  - C = Capacitance (∝ transistor size) ⇩
  - F = CPU frequency ⇧
  - V = CPU Supply Voltage ⇩ (thanks to reductions in V_{th})
- Factors balance each other out to make power constant
- This recipe for scaling frequency while keeping power constant is called Dennard Scaling

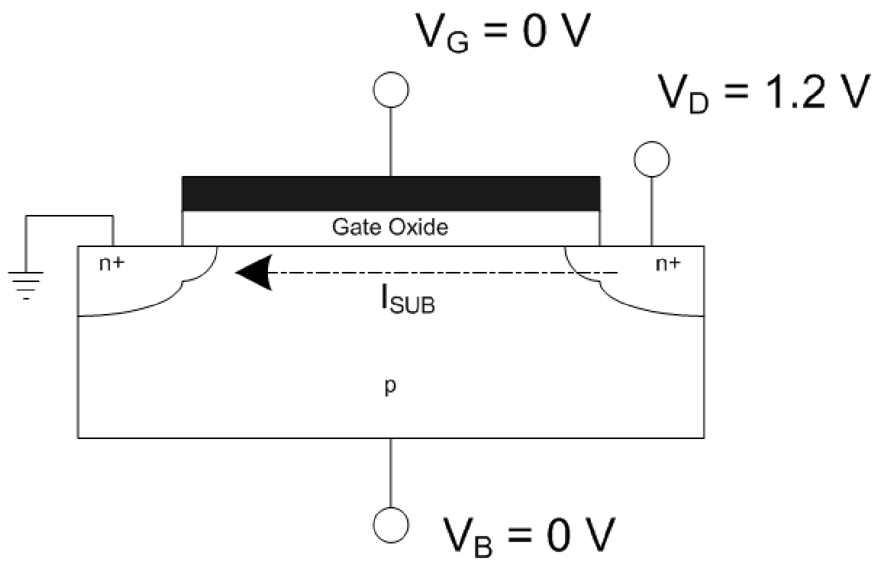
End of Dennard Scaling



- And around 2003 is when Dennard Scaling ended

Limits to Dropping V_{th}

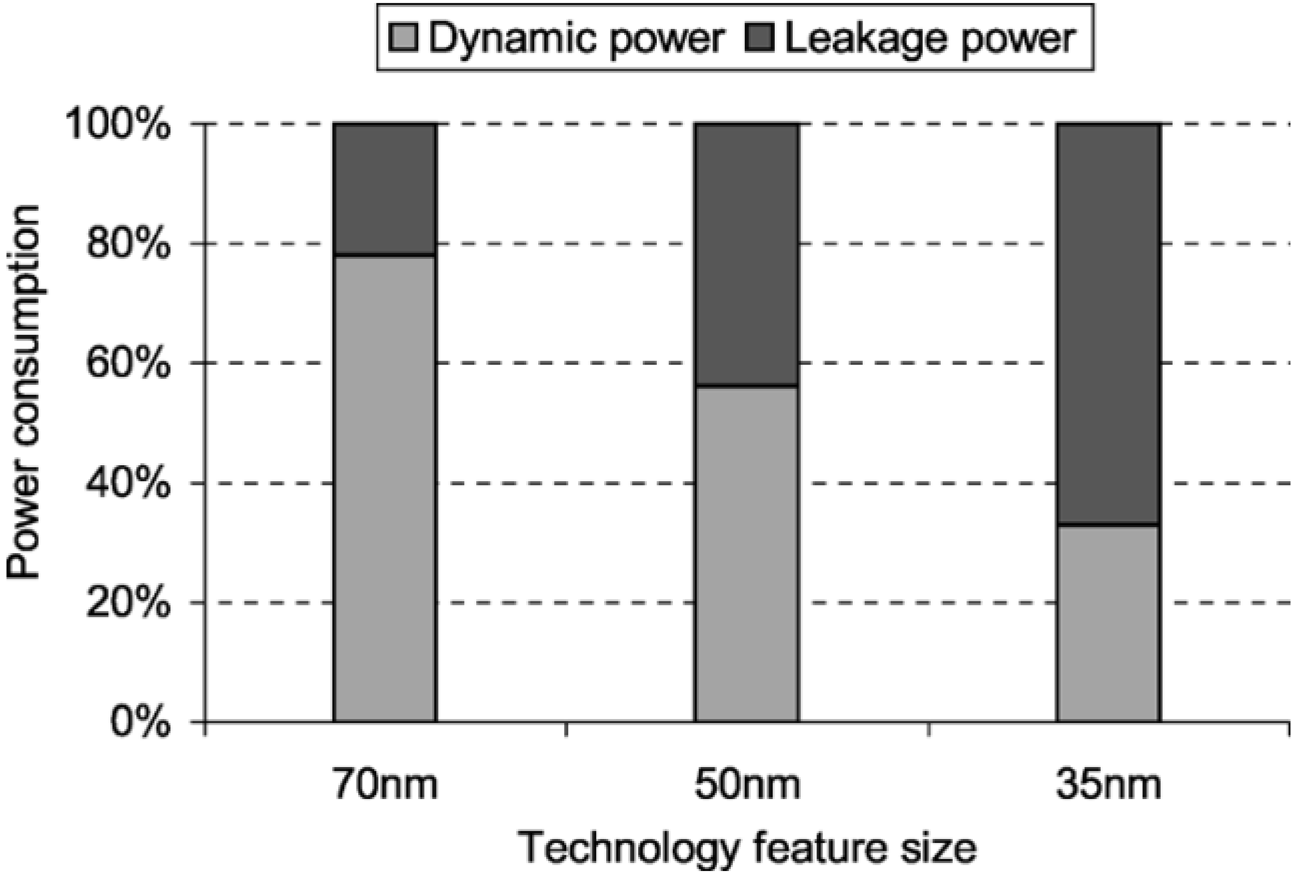
- Subthreshold leakage
  - Transistor leaks current even when gate is off ($V_G = 0$)



  - This leakage current translates to leakage power
  - Leakage worsens when V_{th} is dropped (related to oxide thickness)

Leakage Power across Generations

- Leakage power has increased across technology nodes



Source: L. Yan, Jiong Luo and N. K. Jha, "Joint dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1030-1041, July 2005

End of Dennard Scaling

- $Power_{CPU} = Power_{dynamic} + Power_{leakage}$
  - $Power_{dynamic} = A * N * CFV^2$
  - $Power_{leakage} = f(N, V, V_{th}) = N * V * e^{-Vth}$
  - Leakage worsens *exponentially* when V_{th} is dropped
  - Catch-22: when dropping V_{th}, $Power_{dynamic}$ ⇩ but $Power_{leakage}$ ⇧⇧
  - That means V_{th} can't be reduced and V can't be reduced
- CPU Power = $A * N * CFV^2 + N * V * e^{-Vth}$
  - A = Activity factor
  - N = Number of transistors ⇧ ⇧
  - C = Capacitance ($\propto$ transistor size) ⇩
  - F = CPU frequency ⬄ (Can't increase without violating TDP)
  - V = CPU Supply Voltage ⬄ (Due to fixed V_{th})
  - V_{th} = CPU Threshold Voltage ⬄ (Due to leakage power)

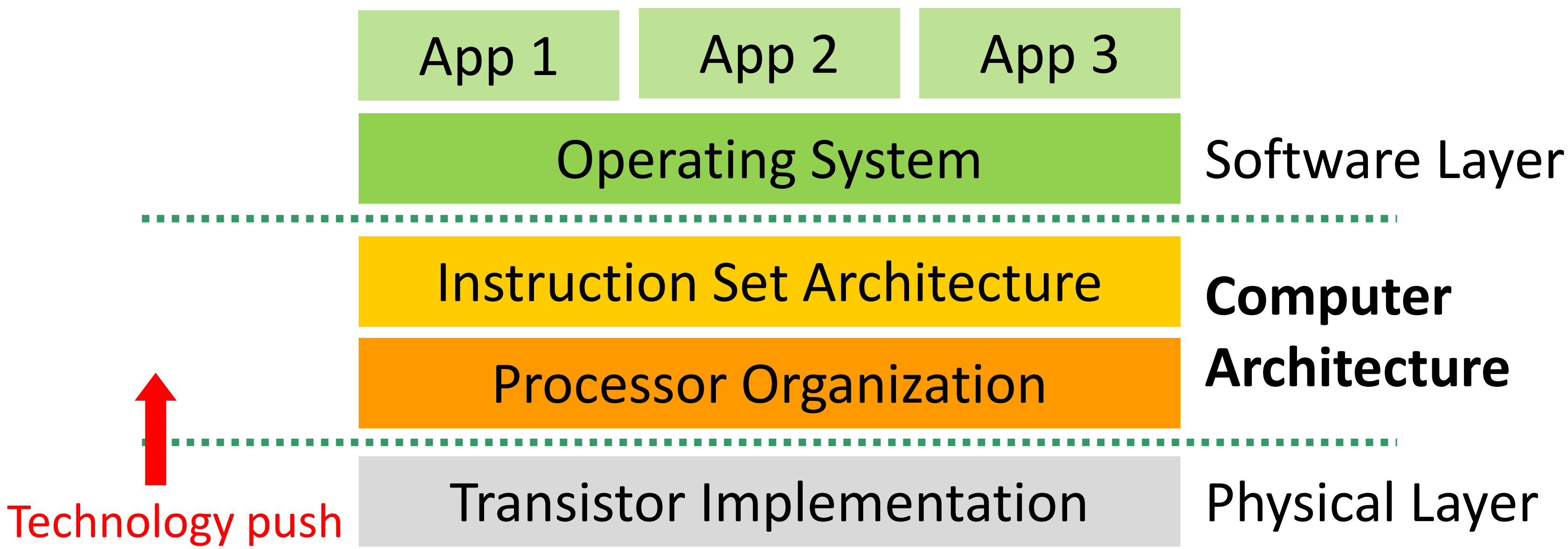
Free Ride is Over

- "Free" speed improvements from transistors is over
- 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 a big one: TDP

Technology Constraints

Technology Constraints

- Constraints in technology push architecture too



  - *Power Wall*: Thermal Design Power (TDP) constraint
  - *Memory Wall*: Constraint in bandwidth to memory
  - *Variability*: Limits in the precision of manufacturing technology
- Processor must be designed to meet all constraints

Power Wall

- $Power_{CPU} = Power_{dynamic} + Power_{leakage}$
  $= A * N * CFV^2 + N * V * e^{-Vth}$
  - Leakage power is also called *static power*
  - This total CPU power cannot exceed TDP
- Moore's Law transistor scaling means two things:
  - N = Number of transistors ⇧ ⇧
  - C = Capacitance (∝ transistor size) ⇩
  - Reductions in C does not compensate for increases in N
- Architects must use tricks to keep power in check
  - To keep packing more transistors to increase performance

1. Reducing Dynamic Power

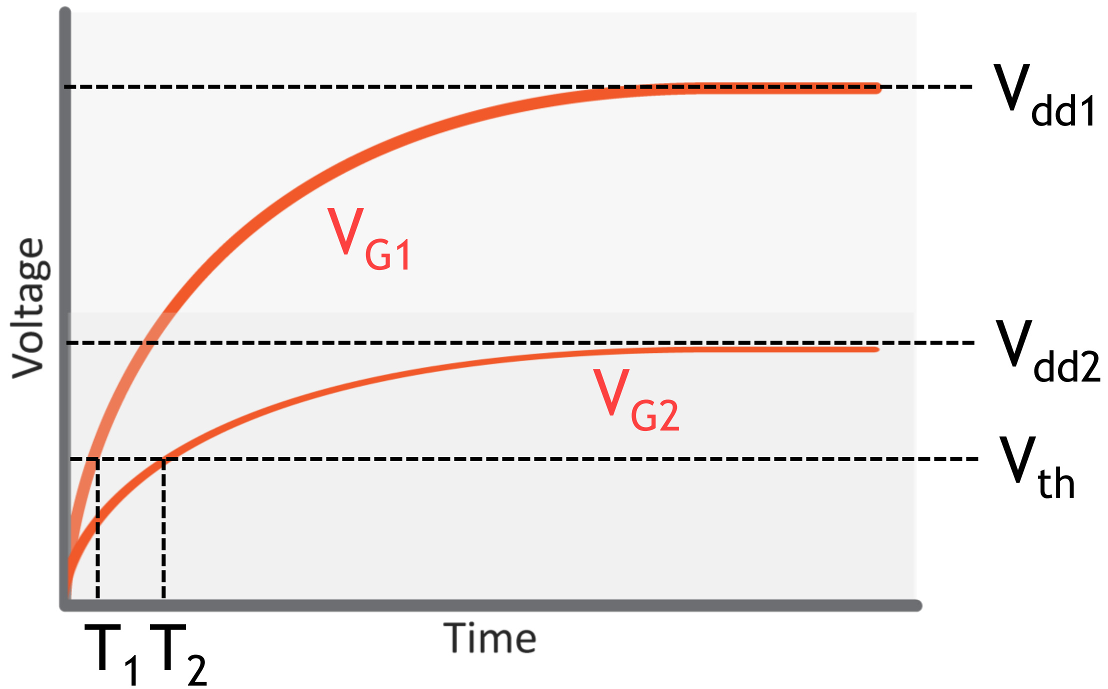
- CPU Power = $A * N * CFV^2 + N * V * e^{-Vth}$

- Reducing A (Activity): *Clock gating*
  - Disables the clock signal to unused parts of the chip (idle cores)
  - Wake-up is instantaneous (the moment clock signal goes in)

- Reducing F (Frequency) and V (Voltage)
  - When F is reduced, V can also be reduced
    (Transistor 101 and water pressure, remember?)
  - *Dynamic Voltage Frequency Scaling (DVFS)* done on multi-cores
    - Slow down low-priority cores, speed up high-priority cores

DVFS and Clock Speed

- RC Charging Curve of V_G



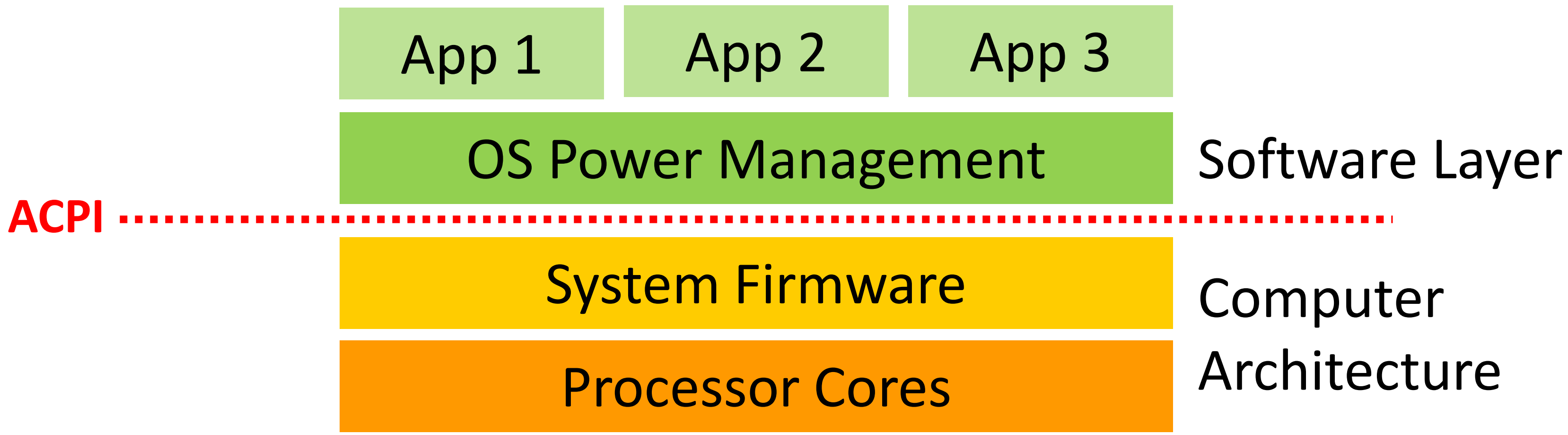
- $V_{dd1} \rightarrow V_{dd2}$ saves power, but slows $T_1 \rightarrow T_2$
- $V_{dd2} \rightarrow V_{dd1}$ uses more power, but speeds up $T_2 \rightarrow T_1$

2. Reducing Leakage Power

- CPU Power = A * N * CFV^2 + N * V * e^{-Vth}
- Reducing N (Transistor Number): *Power gating*
  - Disables power to unused parts of the chip (unused cores)
  - Eliminates dynamic power *and* leakage power to those parts
  - Drawback: wake-up takes a much longer time than clock gating
    - Delay for supply voltage to stabilize
    - Delay to backup and restore CPU state to/from memory
- Reducing V (Voltage): *DVFS* also helps here

OS Manages Power

- Who decides which cores to clock gate and power gate?
- Who decides how to apply DVFS to the cores?
- *ACPI* (*Advanced Configuration and Power Interface*)
  - OS performs power management using this interface
    - OS knows best which threads to prioritize for best user experience
  - Open standard interface to system firmware
    - Firmware sends signals to processor cores to control them



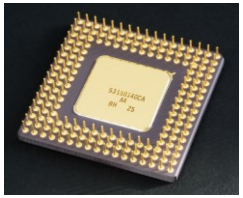
3. Simpler Processor Design

- Plenty of transistors but not enough power
  - Power becomes the ultimate currency in processor design
- To eke out the last bit of performance out of a thread
  - Architects must use increasingly complex logic (more power)
  - Diminishing returns on performance for power investment
- Push towards simpler architectures:
  - *Multi-cores*: Run multiple programs (threads) on simple cores
  - *GPUs*: Run each instruction on massively parallel compute units
  - *Caches*: Memory caches are power efficient (low dynamic power)

Memory Wall

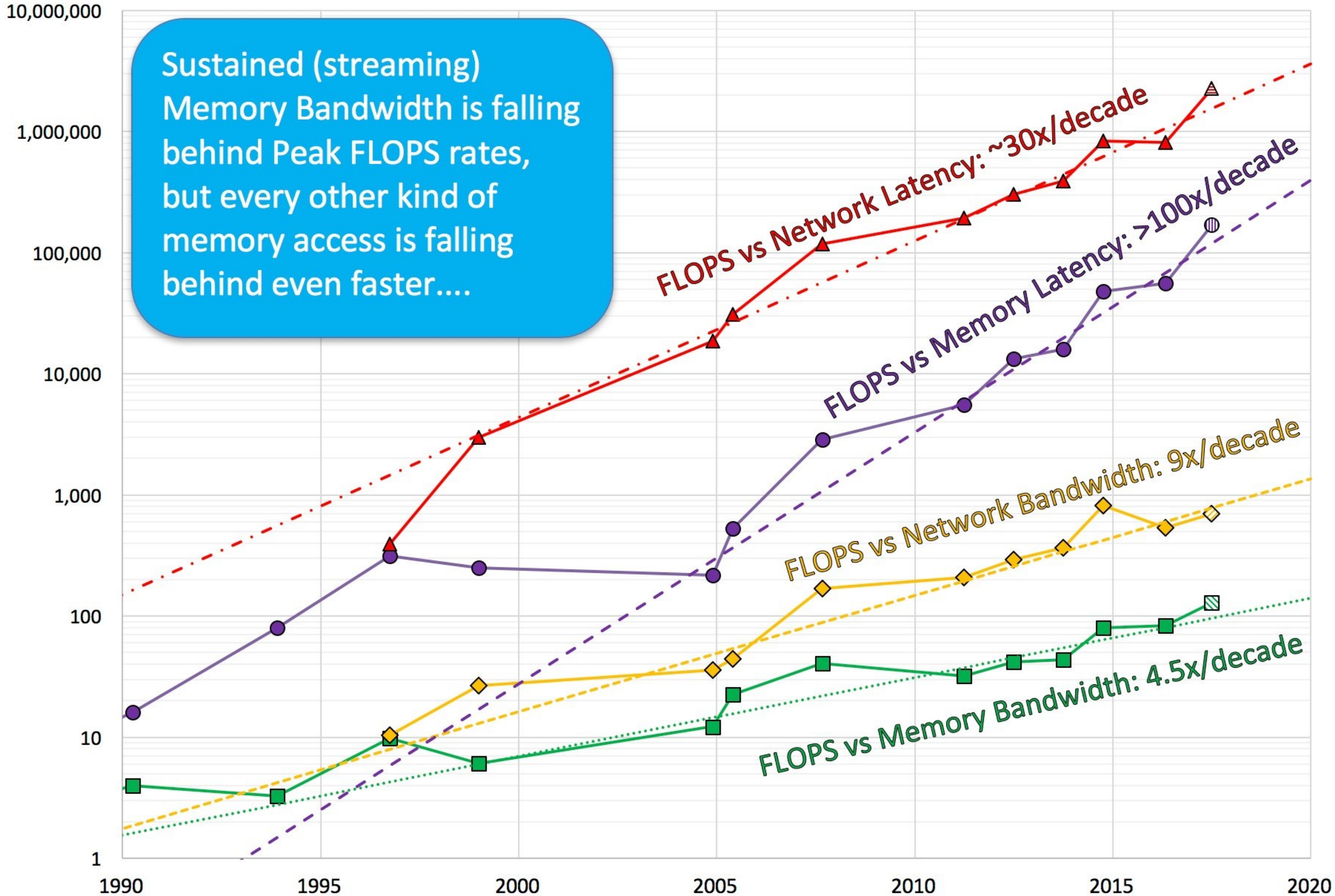
- Refers to both *latency (ns)* and *bandwidth (GB/s)*
  - CPU cycle time and overall performance increased dramatically
  - Memory (DRAM) latency and bandwidth have lagged far behind

- Why?
  - Limit on the number of CPU / DRAM pins that can be soldered on



  - DRAM manufacturers have traditionally prioritized capacity
  - DDR1 (1998): 1.6 GB/s → DDR4 (2014): 25.6 GB/s
    (Impressive? Not so much compared to CPU performance)

Memory Wall



Source: SC16 Invited Talk ""Memory Bandwidth and System Balance in HPC Systems" by John D. McCalpin

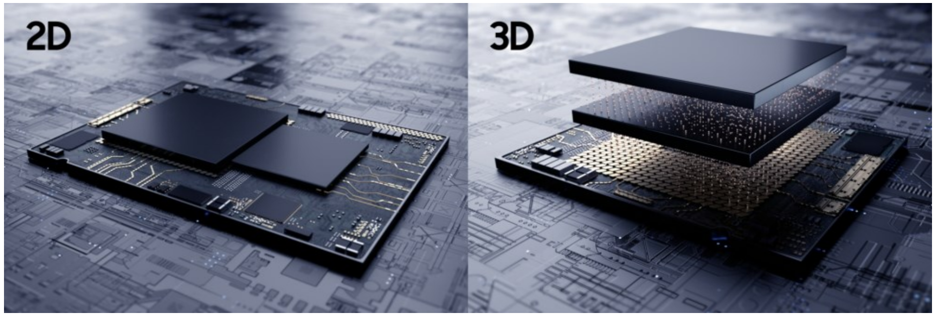
- FLOPS = floating point operations per second (performance)

Memory Wall

- Where did the Memory Wall push architecture?

- *Caches*: If hit in cache, no need to go to memory
  - Caching reduces both data access latency/bandwidth

- *3D-Stacked Memory*: Stack CPU on top of memory
  - Drill vias, or holes, through silicon to bond CPU with memory
  - Through silicon vias (TSVs) have low latency / high bandwidth

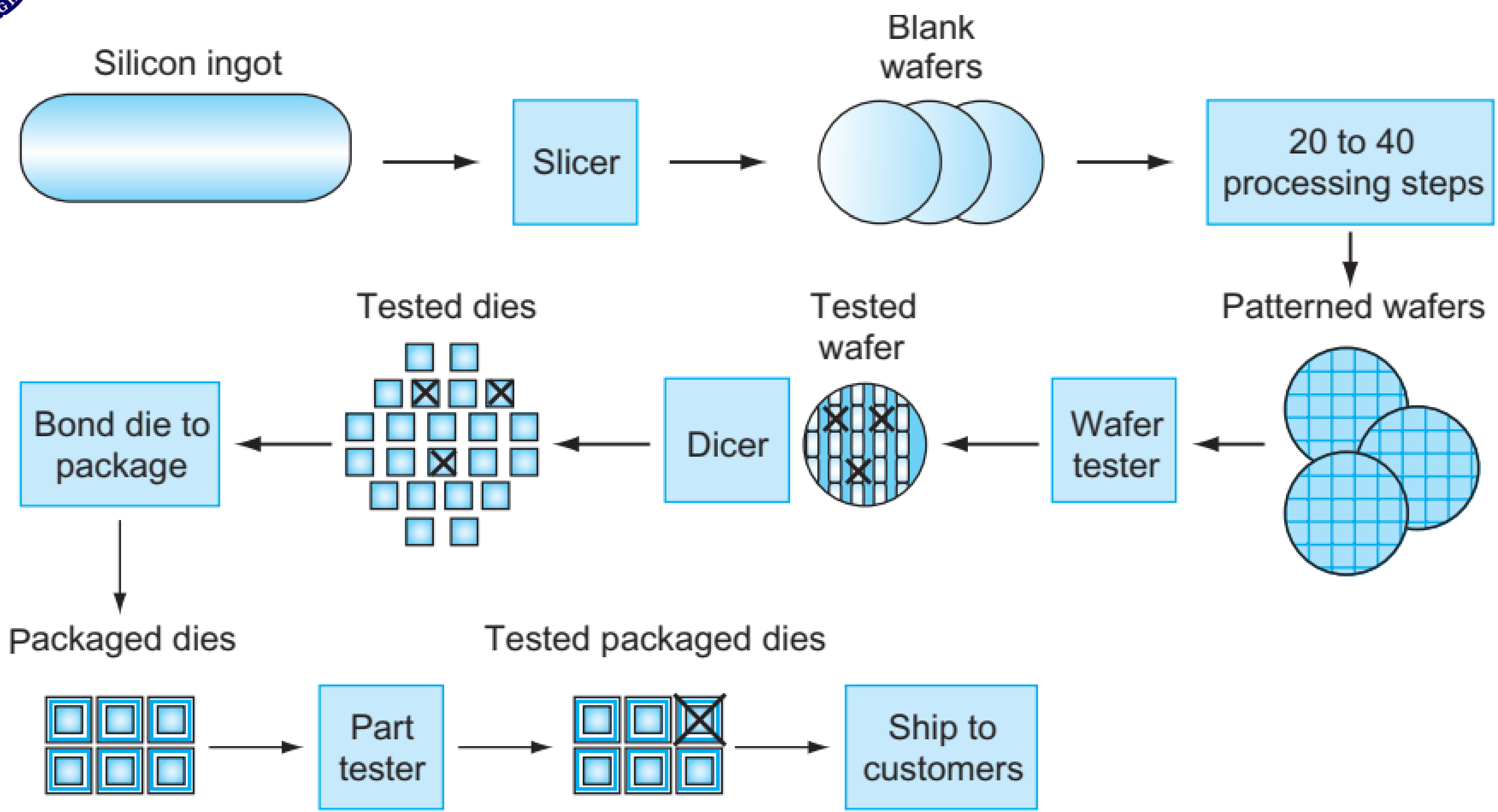


Variability

- *Variability*: differences in speed of individual transistors
  - Nowadays, transistor dimensions measure just dozens of atoms!
  - If fab can't ensure uniformity of transistors, speeds will differ
  - Speed differences come from variations in V_{th}:
    low V_{th} → cycle time ⇩ but power ⇧
    high V_{th} → cycle time ⇧ but power ⇩

- If unlucky and a logic path has lots of slow transistors
  → CPU may miss clock cycle time if path is exercised
  → CPU must be discarded, since it malfunctions
  → Leads to low chip *yield*

Wafer Yield



- Yield is 85% (17 out of 20 tested dies or chips)
- Lower yield leads to higher production cost

Variability

- Where did Variability push architecture?

- *Product binning*: Sell slower CPUs at a cheaper “bin”
  - And rate slower CPUs at a lower CPU frequency
  - Instead of discarding them as “malfunctioning”
- *Multi-cores*: Easy to disable one or two buggy cores
  - Compared to single core where subcomponents must be disabled
  - Used when one or two cores are extremely slow
- *Limited pipelining*: pipelining exacerbates variability
  - With long stages, many transistors so tend to even each other out
  - With short stages, few transistors so probable all are slow

Intel i9 Product Binning

Model	# Cores	# Threads	Base Clock	All Core Turbo	Turbo Boost	Total L3 Cache	PL1 TDP
i9-10900K	10	20	3.7	4.8	5.1	20	$\boxed{125}$
i9-10900KF	10	20	3.7	4.8	5.1	20	125
i9-10900	10	20	2.8	4.5	5.0	20	65
i9-10900F	10	20	2.8	4.5	5.0	20	65
i9-10900T	10	20	1.9	3.7	4.5	20	$\boxed{35}$
i7-10700K	8	16	3.8	4.7	5.0	16	125
i7-10700KF	8	16	3.8	4.7	5.0	16	125
i7-10700	8	16	2.9	4.6	7.7	16	65
i7-10700F	8	16	2.9	4.6	4.7	16	65
i7-10700T	8	16	2.0	3.7	4.4	16	35
i5-10600K	6	12	4.1	4.5	4.8	12	125
i5-10600K	6	12	4.1	4.5	4.8	12	125
i5-10600	6	12	3.3	4.4	4.8	12	65
i5-10600T	6	12	2.4	3.7	4.0	12	35
i5-10500	6	12	3.1	4.2	4.5	12	65
i5-10500T	6	12	2.3	3.5	3.8	12	35
i5-10400	6	12	2.9	4.0	4.3	12	65
i5-10400F	6	12	2.9	4.0	4.3	12	65
i5-10400T	6	12	2.0	3.2	3.6	12	35

Why the close to 4X difference?

Produced from one wafer

Source: https://www.techspot.com/article/2039-chip-binning/

* TDP is calculated using the Base Clock frequency at a nominal supply voltage

Opportunities for Speed Improvement

- So Dennard Scaling is dead
  - Free CPU frequency gains are no longer there

- And we are walled in by technology constraints
  - Power wall
  - Memory wall
  - Variability
  - …

- Where do architects go look for performance?

Improving Execution Time

- Execution time = $\frac{\text{instructions}}{\text{program}}$ X $\frac{\text{cycles}}{\text{instructions}}$ X $\frac{\text{seconds}}{\text{cycle}}$

- Improving $\frac{\text{seconds}}{\text{cycle}}$:
  - *Pipelining* can lead to higher frequencies (by having short stages separated by latches)

- Improving $\frac{\text{cycles}}{\text{instructions}}$:
  - *Superscalars* can execute multiple instructions per cycle
  - *Multi-cores* execute multi-instructions from multi-threads

- Improving $\frac{\text{instructions}}{\text{program}}$:
  - *GPU*s are *SIMD (Single Instruction Multiple Data)* processors

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 will touch upon other goals when relevant
  - Performance will be used synonymously with speed

Textbook Chapters

- Please review Chapter 1 of the textbook.