CS/ECE 752: Advanced Computer Architecture I

Lecture 2 : Technology & Performance

Prof. Matthew D. Sinclair

Slide History/Attribution Diagram:


REMEMBER: WITH GREAT POWER COMES GREAT CURRENT SQUARED TIMES RESISTANCE.
OHM NEVER FORGOT HIS DYING UNCLE'S ADVICE.

Source: XKCD

Announcements

- Advanced Topics Poll Due Tuesday
  - Very even race so far!
- HW0 and Review 1 grading tonight (hopefully)
- HW1 Assigned today (tonight)
- Next Monday: virtual lecture
- Next Wednesday: no class

In-Class Activity

skeptical? Data backs up?

- With a partner, answer the following: But real examples
  - If you were reading this article in 1965, would you believe in Moore's Law? Why?
  - What does the publication source say about the paper? trade magazine
  - Why did Moore's "Law" change from every year to every 18-24 months? physics are/is hard, especially as size shrinks
- In 3 minutes we'll come back together and discuss as a class

Law is more of an observation

In-Class Activity

- With a partner, answer the following:
  - If you were reading this article in 1965, would you believe in Moore's Law? Why?
    - Moore was pretty optimistic but gave examples
    - Only predicted it would last for 10 years
    - Maybe not: technology changes so fast ; 1->2 simple, bigger harder → maybe something to be intrigued by vs a believer (problems like heat may come up as scale increases)
  - What does the publication source say about the paper?
    - Electronics **magazine**
    - Internal tech report?
  - Why did Moore's Law change from every year to every 18-24 months?
    - Observation, not a law
    - Physics / Dennard's Scaling
    - Packaging

How do we build a high-**performance** computer?

Subject to: Cost,
Power,
Energy,
Reliability,
Security,
Generality

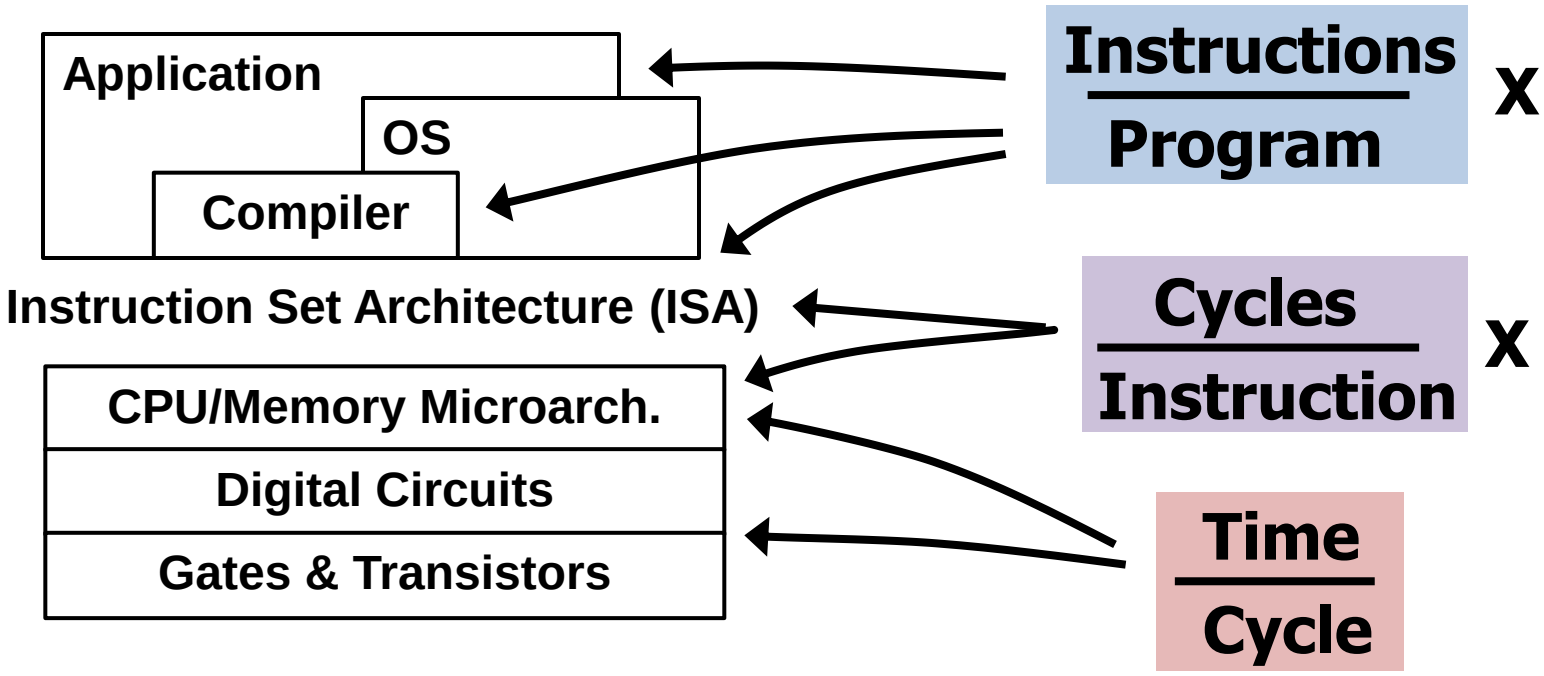
Iron Law of Performance

clock (freq.)

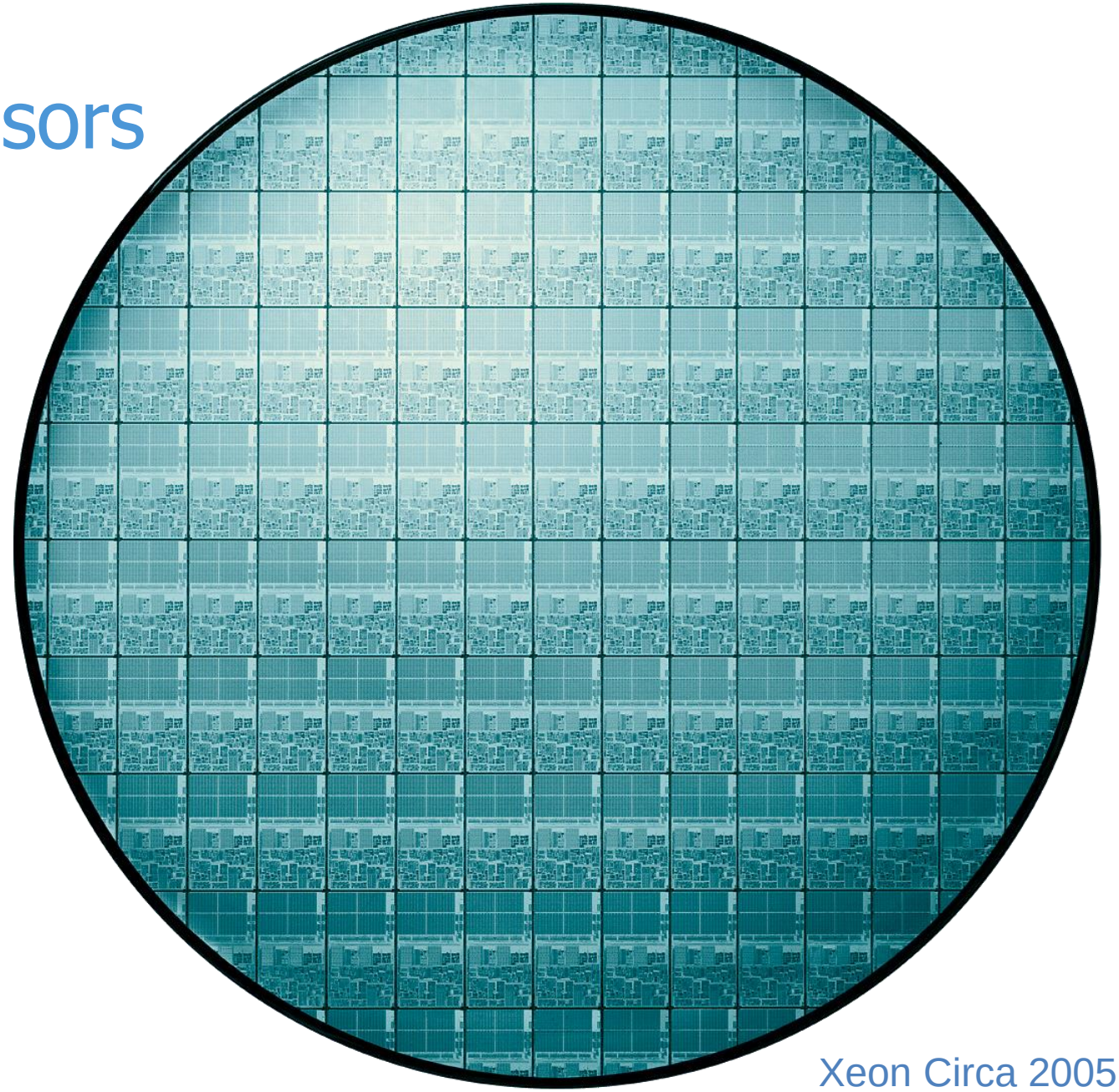
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

- Programs consist of simple operations (instructions)
  - Add two numbers, fetch data value from memory, etc.
- **Instructions per program**: "dynamic instruction count"
  - Runtime count of instructions executed by the program
  - Determined by program, compiler, instruction set architecture (ISA)
- **Cycles per instruction**: "CPI" (typical range: 2 to 0.5)
  - On average, how many *cycles* does an instruction take to execute?
  - Determined by program, compiler, ISA, micro-architecture
- **Seconds per cycle**: clock period, length of each cycle
  - Inverse metric: cycles per second (Hertz) or cycles per ns (Ghz)
  - Determined by micro-architecture, **technology parameters**

Computer System Layers



What are microprocessors made of?

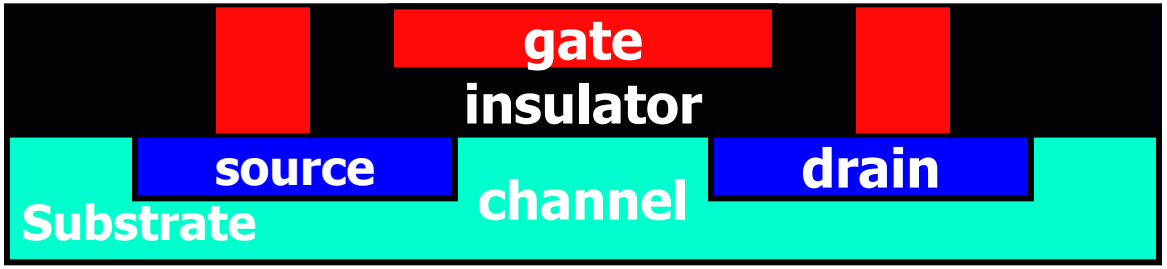


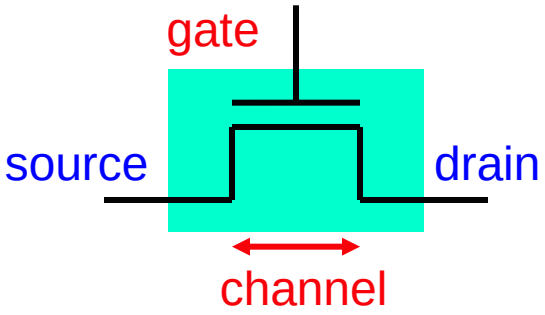
Xeon Circa 2005

Computer Elements

- Transistors (computing)
  - How can they be connected to do something useful?
  - How do we evaluate how fast a logic block is?
- Wires (transporting)
  - What and where are they?
  - How can they be modeled?
- Higher level things…
- Memories (storing)
  - SRAM (speed) vs. DRAM (density)
  - CAMs: Memories enabling associative lookups (lookup by value rather than by index)
- Compute in Memory /NDP/PIM/PNM
  - Memristors: Memories with programmable resistance

Basic technology element: **MOSFET**



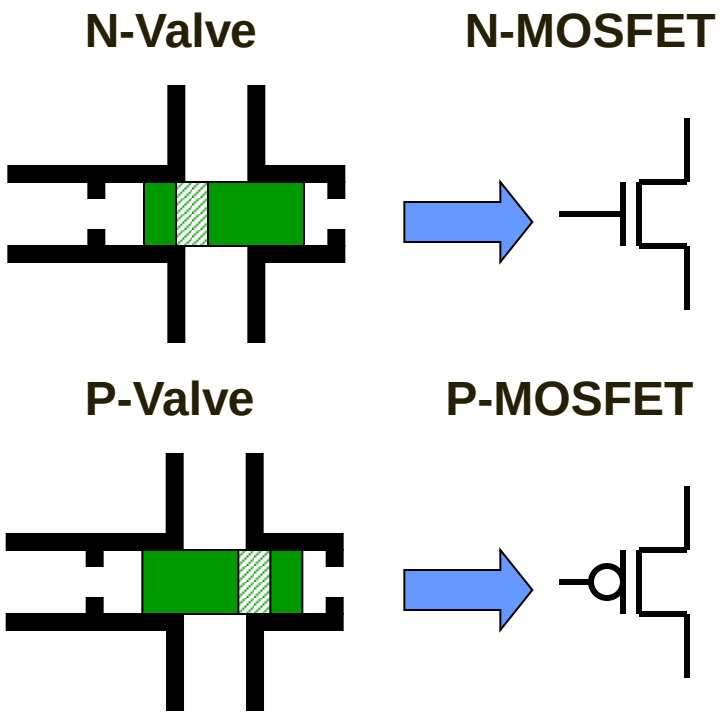


- Solid-state component acts like electrical switch
- **MOS:** three materials needed to make a transistor
  - **Metal:** Aluminum, Tungsten, Copper: conductor
  - **Oxide:** Silicon Dioxide (SiO 2): insulator
  - **Semiconductor:** doped Si: conducts under certain conditions
- **FET**: field-effect (the mechanism) transistor
  - Voltage on gate: current flows source to drain (transistor on)
  - No voltage on gate: no current (transistor off)
- **Channel length**: characteristic parameter (short → fast)
  - Aka "feature size" or "technology"
  - Currently: 7-10 nm (0.007 – 0.010 micron)
  - Continued miniaturization (scaling) known as "**Moore's Law**"

How do Transistors work?

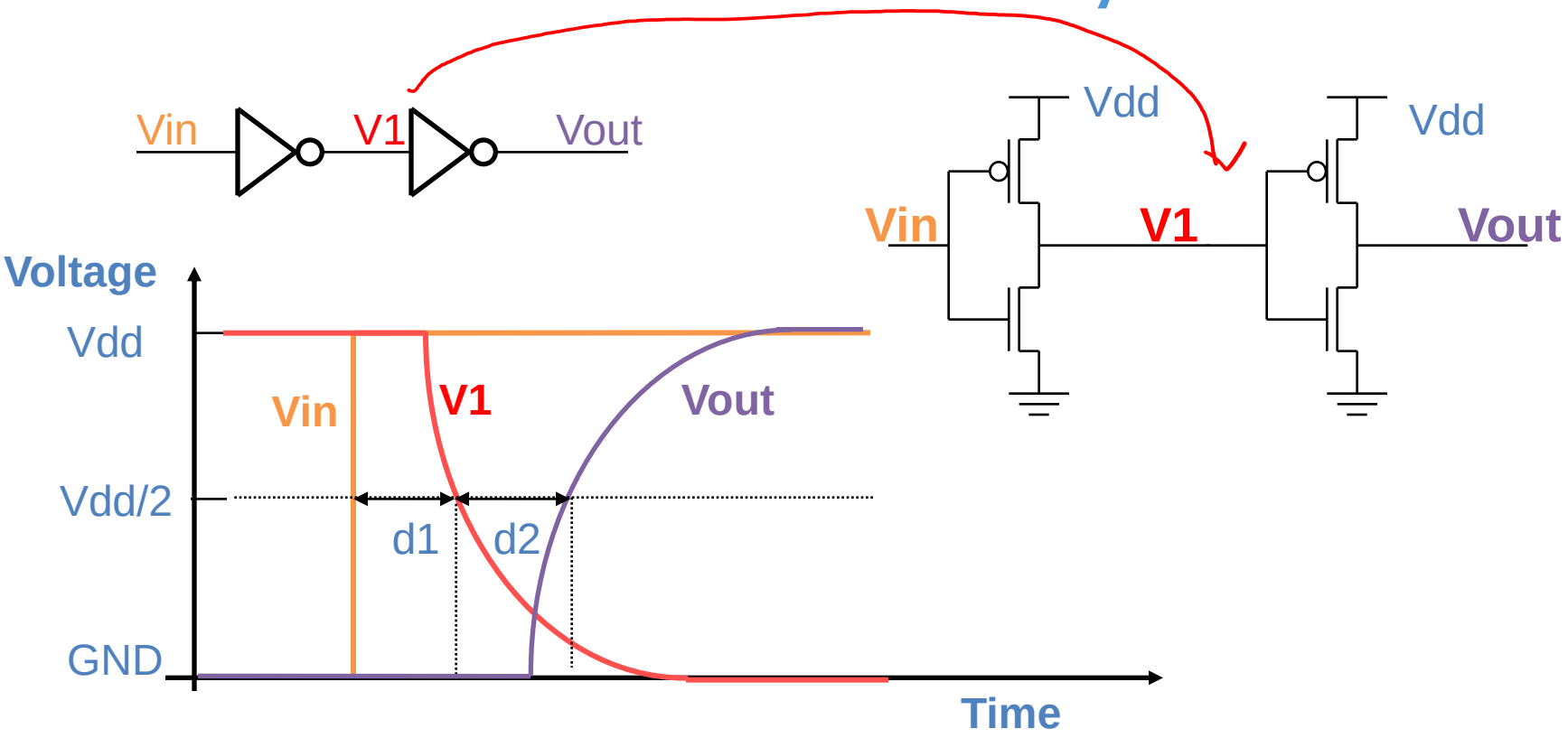
Transistors as Switches

- Two types
  - N-type: Conduct when gate voltage is 1
  - P-type: Conduct when gate voltage is 0
- Properties
  - Solid state (no moving parts)
  - Reliable (low failure rate)
  - Small (10nm channel length)
  - Fast (<0.1ns switch latency)
  - **CMOS**: complementary n-/p-networks form Boolean logic



Circuit Timing & Critical Path

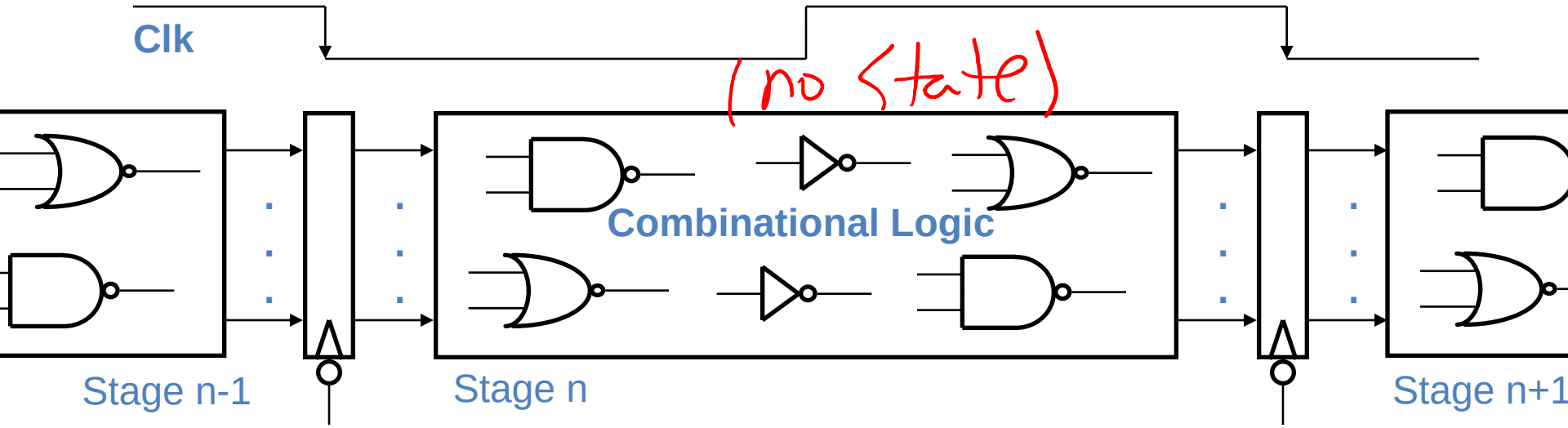
What about Transistor Delay



- Total Propagation Delay = Sum of individual delays = d1 + d2

Synchronous Design

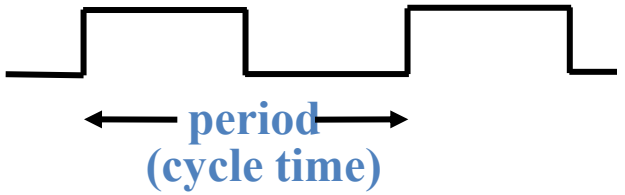
- Hard to coordinate design components with logic only
  - (can be done, called asynchronous design)
- Synchronous: State elements synchronized by clock



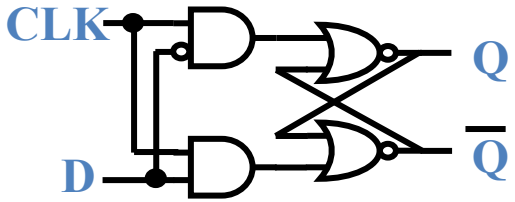
- All storage elements are clocked by the same clock edge
- The combination logic block's:
  - Inputs are updated at each clock tick
  - All outputs MUST be stable before the next clock tick

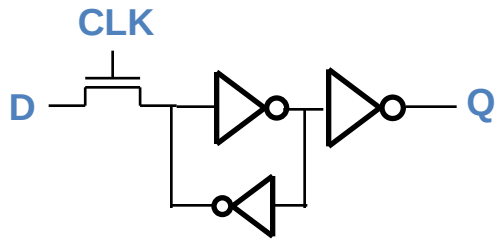
Clocking and Clocked Elements

- Simple Clock
  - 1Hz = 1 cycle per second
  - 1Ghz = billion cycles per second
    - (1ns period)



- Transparent Latch

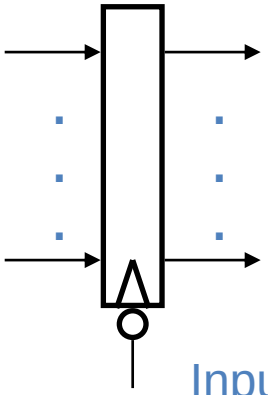


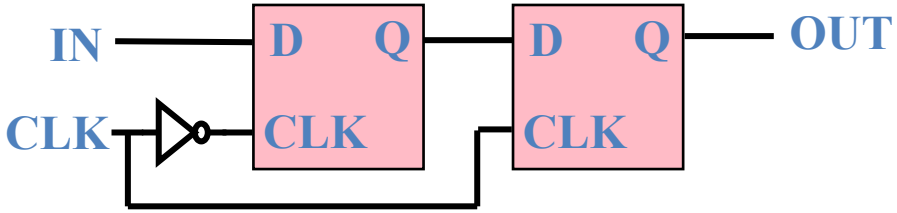


CLK=0, Q=oldQ
CLK=1, Q=D

2 D-latches → 1 D-FF

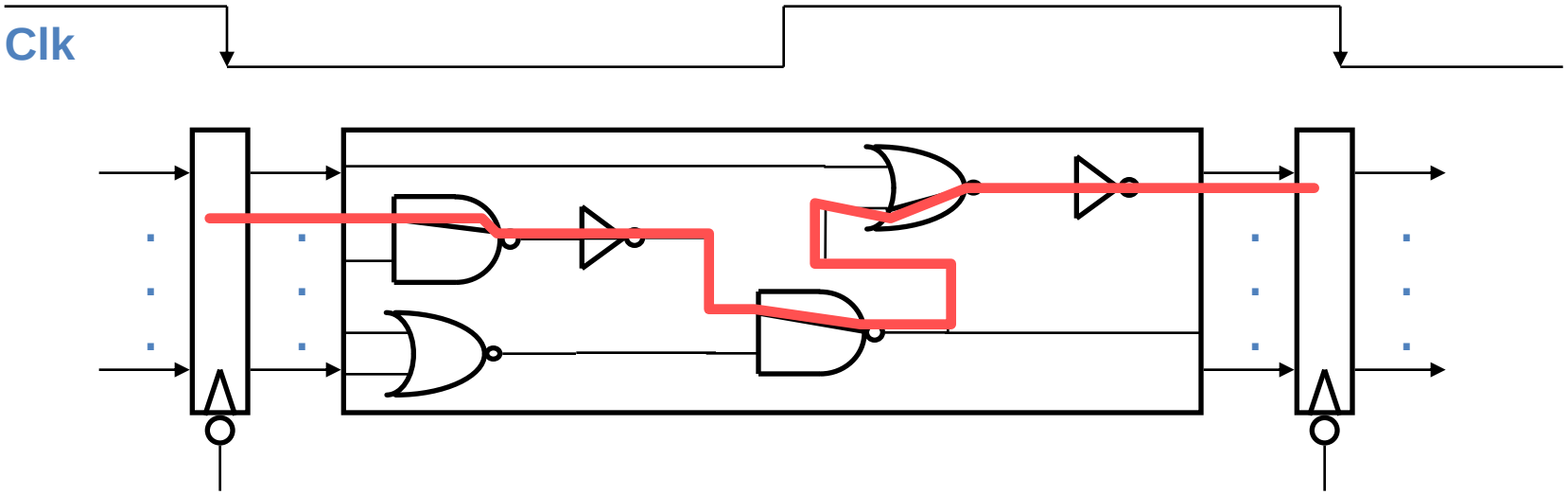
- Edge Triggered Flip-Flop





Inputs at end of cycle are outputs for next cycle

Critical Path & Cycle Time

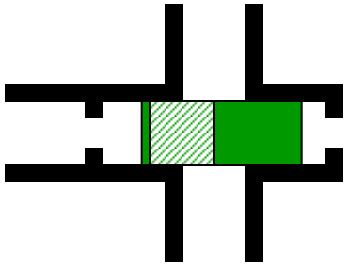


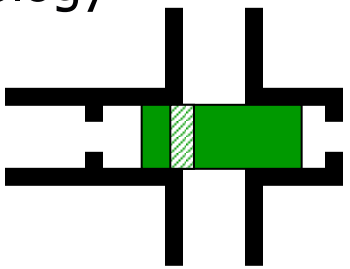
- Critical path: the slowest path between any two storage devices
- Cycle time > time critical path
- **This is where frequency comes from :)**

Technology Basis of Transistor Speed

- Physics 101: delay through an electrical component ~ **RC**
  - **Resistance (R)**  ~ length / cross-section area
    - Slows rate of charge flow
  - **Capacitance (C)**  ~ length * area / distance-to-other-plate
    - Stores charge
  - **Voltage (V)**
    - Electrical pressure
  - **Threshold Voltage (V_t)**
    - Voltage at which a transistor turns "on"
    - Property of transistor based on fabrication technology
  - **Switching time ~ to (R * C) / (V – V_t)**

Low Vt (faster, more power)





High Vt (slower, less power)

(ignoring static P)

- Two kinds of electrical components
  - CMOS transistors (gates)
  - Wires

Increasing Problem: Wire Delay

- RC Delay of wires
  - **Resistance** proportional to: resistivity * length / (cross section)
    - Wires with smaller cross section have higher resistance
    - Resistivity (type of metal, copper vs aluminum)
  - **Capacitance** proportional to length
    - And wire spacing (closer wires have large capacitance)
    - Permittivity or "dielectric constant" (of material between wires)
- Result: delay of a wire is **quadratic** in length
  - Insert "inverter" repeaters for long wires 

  - Why? To bring it back to linear delay… but repeaters still add delay
- Trend: wires are getting relatively slow to transistors
  - And relatively longer time to cross relatively larger chips
- **Arch. Implication:**
  - **Avoid limit the amount of communication**
  - **Keep high-bandwidth communication as local as possible**

Power & Energy

Power/Energy Are Increasingly Important

- **Battery life** for mobile devices
  - Laptops, phones, cameras
- **Tolerable temperature** for devices without active cooling
  - Power means temperature, active cooling means **cost**
  - No room for a fan in a cell phone, no market for a hot cell phone
- **Electric bill** for compute/data centers
  - Pay for power twice: once in, once out (to cool)
- **Environmental concerns**
  - "Computers" account for growing fraction of energy consumption
  - But let's be realistic:
    - Gallon of gas: ~120 million Joules
    - iPhone 10 battery: ~40 thousand Joules (3000x times less)

LLMs driving

Energy & Power

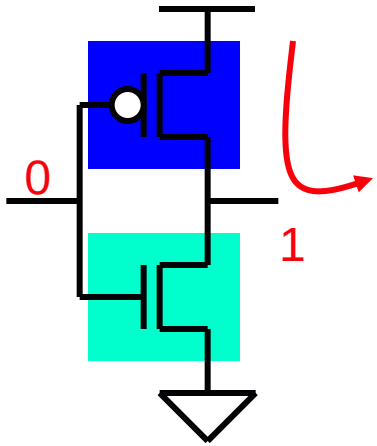
- **Energy**: measured in Joules or Watt-seconds
  - Total amount of energy stored/used
  - Battery life, electric bill, environmental impact
  - Joules per Instruction (car analogy: gallons per mile)
- **Power**: energy per unit time (measured in Watts)
  - Joules per second (car analogy: gallons per hour)
  - Related to "performance" (which is also a "per unit time" metric)
  - Power impacts power supply and cooling requirements (cost)
    - Power-density (Watt/mm^2): important related metric
  - Peak power vs average power
    - E.g., camera, power "spikes" when you actually take a picture
- Two sources:
  - **Dynamic power**: active switching of transistors
  - **Static power**: leakage of transistors even while inactive

Recall: Tech. Basis of Transistor Speed

- Physics 101: delay through an electrical component ~ **RC**
  - **Resistance (R)**  ~ length / cross-section area
    - Slows rate of charge flow
  - **Capacitance (C)**  ~ length * area / distance-to-other-plate
    - Stores charge
  - **Voltage (V)**
    - Electrical pressure
  - **Threshold Voltage (V_t)**
    - Voltage at which a transistor turns "on"
    - Property of transistor based on fabrication technology
  - **Switching time ~ to (R * C) / (V – V_t)**

Dynamic Power

- **Dynamic power ($P_{dynamic}$)**: aka switching or active power
  - Energy to switch a gate (0 to 1, 1 to 0)
  - Each gate has capacitance (C)
    - Energy to charge/discharge a capacitor is ~ to C * V^2
    - Why (Energy Q = C*V, E ~ Q*V)



  - **$P_{dynamic}$ ~ N * C * V^2 * f * A**
    - N: number of transistors
    - C: capacitance per transistor (size of transistors)
    - V: voltage (supply voltage for gate)
    - f: frequency (transistor switching freq. is ~ to clock freq.)
    - A: activity factor (not all transistors may switch this cycle)

Reducing Dynamic Power

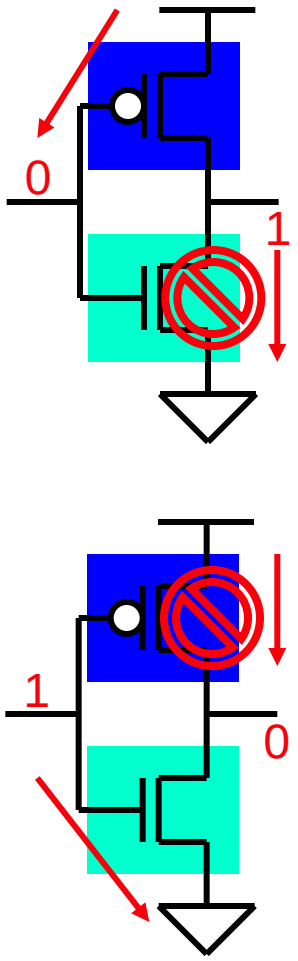
- Target each component: $\mathbf{P_{dynamic} \sim N * C * V^2 * f * A}$
- **Reduce number of transistors** (N)
  - Use fewer transistors/gates (better design; specialized hardware)
- **Reduce capacitance** (C)
  - Smaller transistors (Moore's law)
- **Reduce voltage** (V)
  - Quadratic reduction in energy consumption!
  - But also slows transistors (recall transistor speed is ~ to V)
- **Reduce frequency** (f)
  - Slower clock frequency (reduces power but not energy) Why?
- **Reduce activity** (A)
  - "Clock gating" disable clocks to unused parts of chip
  - Don't switch gates unnecessarily

Announcements 9/13

- HW1 Due Tomorrow (on Canvas)
- I am working through emails form you all while I was gone
  - If you enrolled late in the course I will handle your Review 1/HW0 accordingly
- Review 1 grading will be done tonight or tomorrow
  - Sorry for the delay
- HW2 Assigned Tonight or Tomorrow

Static Power

- **Static power (P_{static})**: aka idle or leakage power
  - Transistors don't turn off all the way
  - Transistors "leak"
    - Analogy: leaky valve
  - $P_{static} \sim N * V * e^{-V_t}$
  - N: number of transistors
  - V: voltage
  - **V_t (threshold voltage)**: voltage at which transistor conducts (begins to switch)
- Switching speed vs leakage trade-off
- The lower the **V_t**:
  - Faster transistors (linear)
    - Transistor speed ~ to $V - V_t$
  - Leakier transistors (exponential)



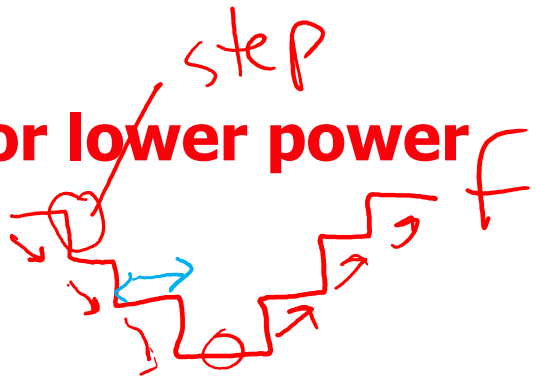
Reducing Static Power

- Target each component: $P_{static} \sim N * V * e^{-Vt}$
- **Reduce number of transistors** (N)
  - Use fewer transistors/gates
- **Disable transistors** (also targets N)
  - "Power gating" disable power to unused parts (long latency to power up)
  - Power down units (or entire cores) not being used
- **Reduce voltage** (V)
  - Linear reduction in static energy consumption
  - But also slows transistors (transistor speed is ~ to V)
- **Dual V_t** – use a mixture of high and low V_t transistors
  - Use slow, low-leak transistors in SRAM arrays
  - Requires extra fabrication steps (cost)
- **Low-leakage transistors**
  - High-K/Metal-Gates in Intel's 45nm process
- Note: reducing frequency can actually hurt static energy. Why?

Scale Voltage & Frequency Together

- Recall: $P_{dynamic}$ ~ N * C * **V^2** * **f** * A
  - Because max frequency ~ to V…
  - $P_{dynamic}$ ~ to **V^3**
- Reduce both voltage and frequency linearly
  - **Cubic decrease in dynamic power**
  - Linear decrease in performance (sub-linear if memory bound)
    - Thus, only about quadratic in energy
  - Linear decrease in static power
    - Thus, only modest static energy improvement
- Example: Intel Xscale
  - 1 GHz → 200 MHz reduces energy used by 30x
    - But around 5x slower
  - 5 x 200 MHz in parallel, use **1/6th the energy**
- **Power-limited designs favor multi-cores!** multi-accel

Dynamic Voltage/Frequency Scaling (DVFS)

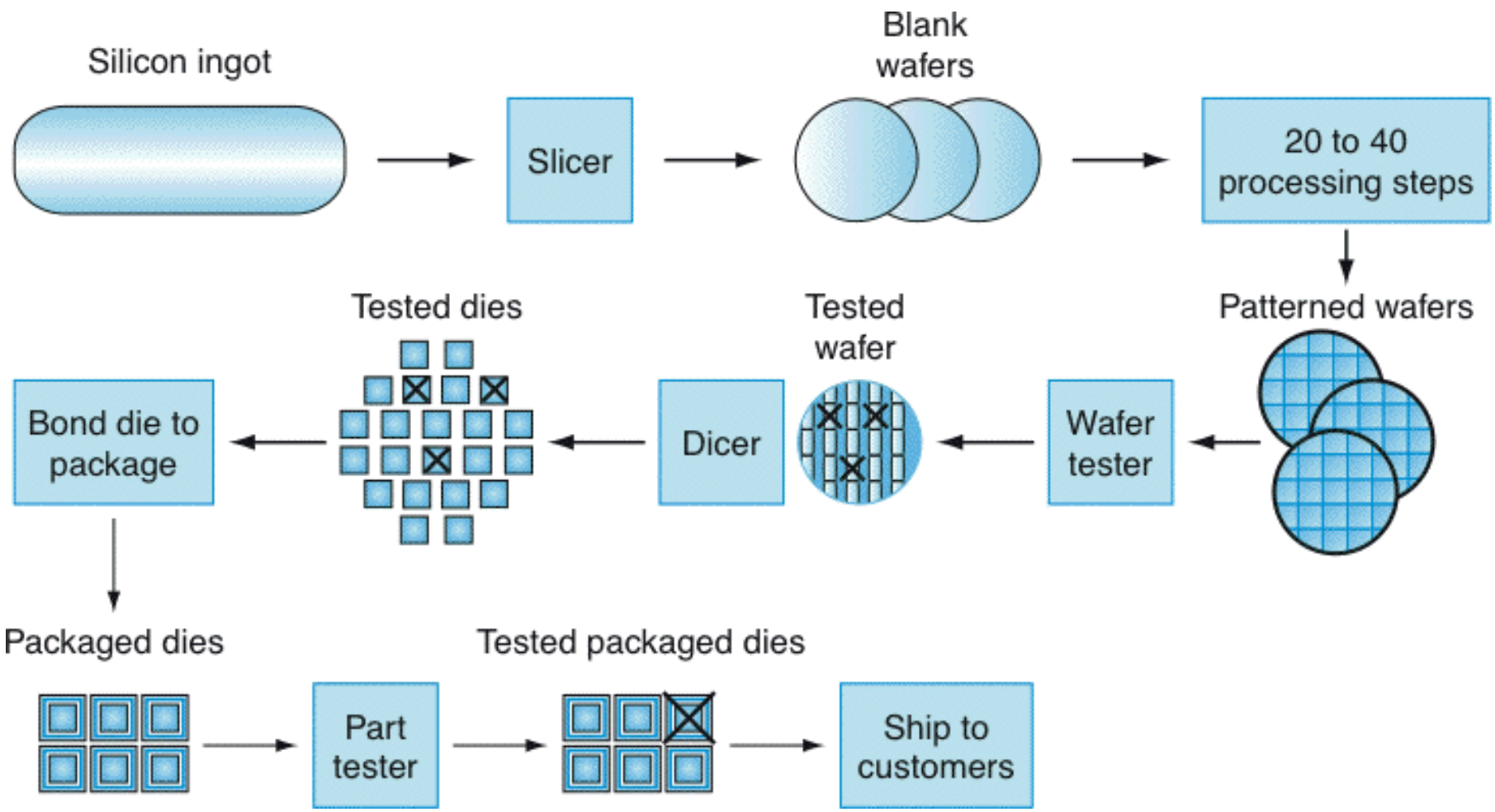


- **Dynamically trade-off performance for lower power**
  - Change the voltage and frequency at runtime
  - Under control of operating system
  - Best of both worlds?
- Modern chips can adjust frequency on a per-core basis

	Mobile PentiumIII "**SpeedStep**"	Transmeta 5400 "LongRun"	Intel X-Scale (StrongARM2)
f (MHz)	300–1000 (step=50)	200–700 (step=33)	50–800 (step=50)
V (V)	0.9–1.7 (step=0.1)	1.1–1.6V (cont)	0.7–1.65 (cont)
High-speed	3400MIPS @ 34W	1600MIPS @ 2W	800MIPS @ 0.9W
Low-power	1100MIPS @ 4.5W	300MIPS @ 0.25W	62MIPS @ 0.01W

Manufacturing:
Get those transistors on a chip!

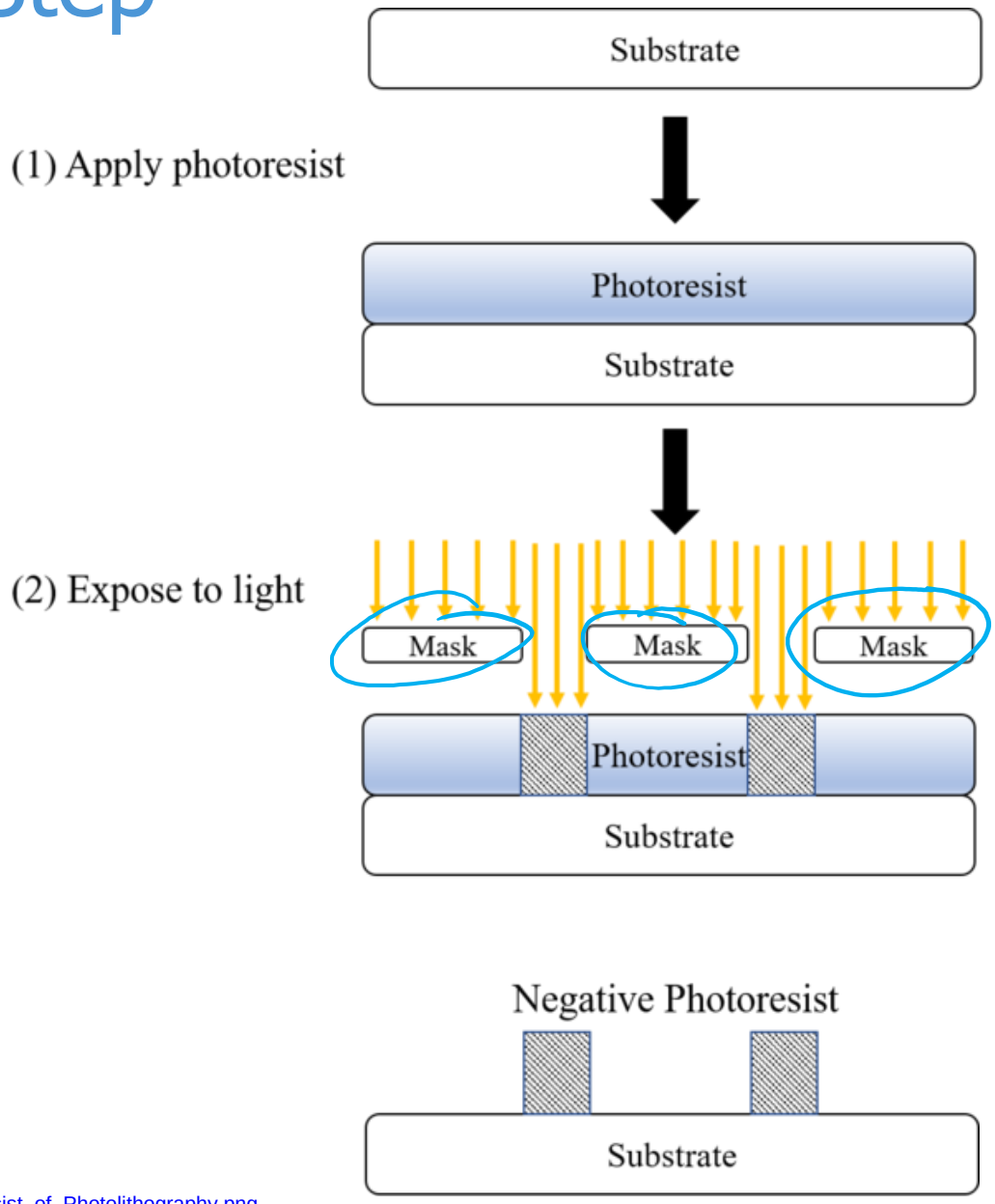
Manufacturing Steps



"floor scrubbing"

Source: P&H

Lithography Step

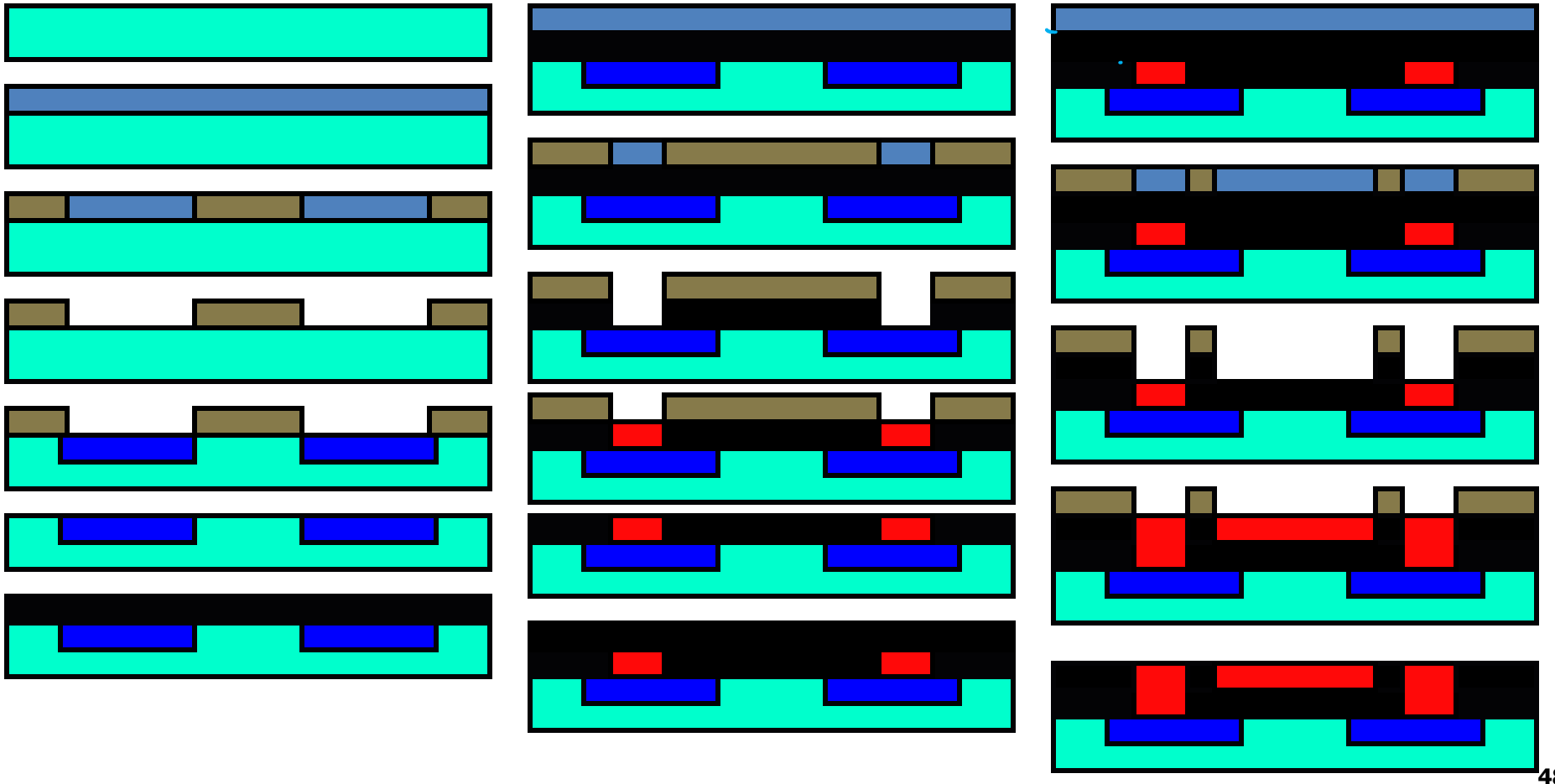


https://en.wikipedia.org/wiki/Photoresist#/media/File:Photoresist_of_Photolithography.png

Manufacturing Steps

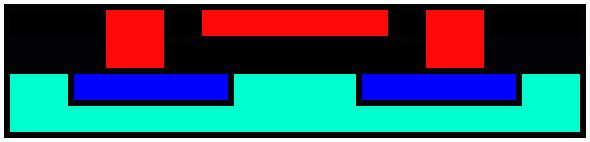
- Multi-step photo-/electro-chemical process
  - More steps, higher unit cost

+ Fixed cost mass production ($1 million+ for “mask set”)

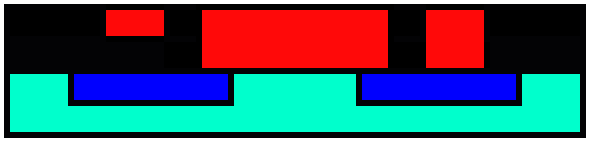


Manufacturing Defects

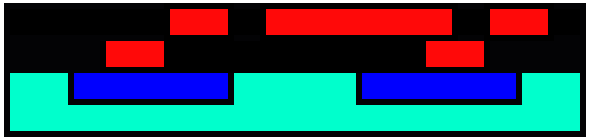
Correct:



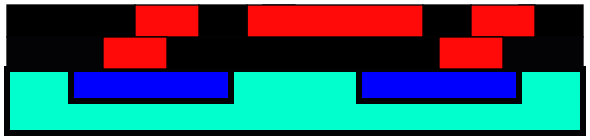
Defective:



Defective:



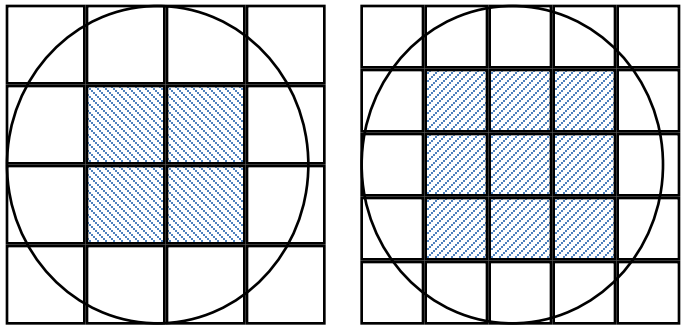
Slow:

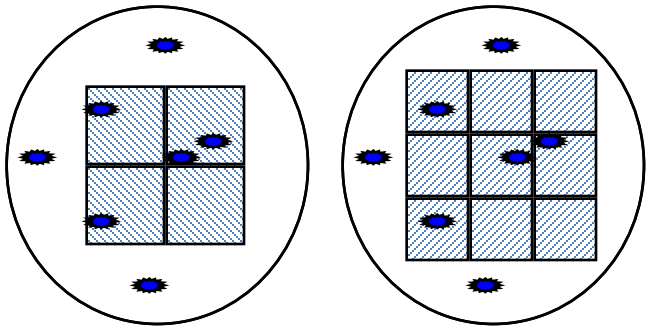


- Defects can arise
  - Under-/over-doping
  - Over-/under-dissolved insulator
  - Mask mis-alignment
  - Particle contaminants
- Try to minimize defects
  - Process margins
  - Design rules
    - Minimal transistor size, separation
- Or, tolerate defects
  - Redundant or "spare" memory cells
  - Can substantially improve yield

Cost Implications of Defects

- Chips built in multi-step chemical processes on **wafers**
  - Cost / wafer is constant, f(wafer size, number of steps)
- Chip (die) cost is related to **area**
  - Larger chips means fewer of them
- Cost is more than linear in area
  - Why? random defects
  - Larger chip: more chance of defect





- **Wafer yield**: % wafer that is chips
- **Die yield**: % chips that work
- Yield is increasingly non-binary - fast vs slow chips

All Roads Lead To Multi-Core (Multi-Accel.)

+ Multi-cores reduce unit costs
  - Higher yield than same-area single-cores
  - Why? Defect on one of the cores? Sell remaining cores for less
  - IBM manufactured CBE ("cell processor") with eight cores
    - But PlayStation3 software was often written for seven cores
    - Yield for eight working cores was too low
  - Sun manufactures Niagara's (UltraSparc T1) with eight cores
    - Also sells six- and four- core versions (for less)

+ Multi-cores can reduce design costs too
  - Replicate existing designs rather than re-design larger single-cores

Fixed Costs

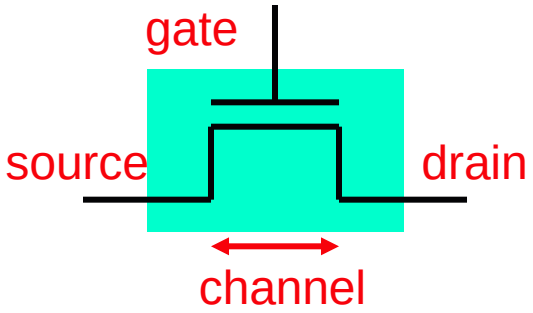
- For new chip design
  - Design & verification: ~$100M (500 person-years @ $200K per)
  - Amortized over "proliferations", e.g., Core i3, i5, i7 variants

- For new (smaller) technology generation
  - ~$3B for a new fab (future fabs probably a lot more)
  - Amortized over multiple designs
  - Amortized by "rent" from companies that don't fab themselves
    - Xilinx invented this approach ⇒ like TSMC do now
    - E.g. Qualcomm, Apple, NVIDIA, Broadcom, MediaTek

– Moore's Law generally increases startup cost
  - More expensive fabrication equipment
  - More complex chips take longer to design and verify

Technology Scaling

Moore's Law: Technology Scaling



- **Moore's Law**: aka "technology scaling"
  - Continued miniaturization (esp. reduction in channel length)
  - \+ Improves switching speed, area(cost)/transistor
  - – Reduces transistor reliability
  - Literally: DRAM density (transistors/area) doubles every 18 months
- Dennard Scaling: power/transistor
  - Corollary to Moore's law that has more-or-less ended
- Public interpretation: performance doubles every 18 months

Moore's Effect #1: Transistor Count

- Linear shrink in each dimension
  - 180nm, 130nm, 90nm, 65nm, 45nm, 32nm, …
  - Each generation is a 1.414 linear shrink ($\sqrt{2}$)
    - Shrink each dimension (2D)
  - Results in 2x more transistors (1.414*1.414)

- Reduces cost per transistor

- More transistors can increase performance
  - Job of a computer architect: use the ever-increasing number of transistors
  - Examples: caches, exploiting parallelism at all levels

Moore's Effect #2: RC Delay

- **First-order: speed scales proportional to gate length**
  - Has provided much of the performance gains in the past
- Scaling helps wire and gate delays in some ways…
  + Transistors become shorter (Resistance↓), narrower (Capacitance↓)
  + Wires become shorter (Length↓ → Resistance↓)
  + Wire "surface areas" become smaller (Capacitance↓)
- Hurts in others…
  – Transistors become narrower (Resistance↑)
  – Gate insulator thickness becomes smaller (Capacitance↑)
  – Wires becomes thinner (Resistance↑)
- What to do?
  - Take the good, use wire/transistor sizing & repeaters to counter bad
  - Exploit new materials: Aluminum → Copper, metal gate, high-K
- **Used to get much faster, not as much any more…**

Moore's Effect #3: Cost

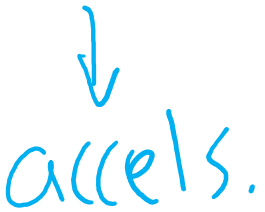
- Mixed impact on unit integrated circuit cost
  - + Either lower cost for same functionality…
  - + Or same cost for more functionality
  - – Difficult to achieve high yields

– Increases startup cost
  - More expensive fabrication equipment
  - Takes longer to design, verify, and test chips

– Process variation across chip increasing
  - Some transistors slow, some fast
  - Increasingly active research area: dealing with this problem

Moore's Effect #4: Power

+ Technology scaling reduces power/transistor…
  - Reduced sizes and surface areas reduce capacitance (C)
– …but increases power density and total power
  - By increasing transistors/area and total transistors
  - Faster transistors → higher frequency → more power
  - Hotter transistors leak more (thermal runaway)
- The end of voltage scaling & "dark silicon"



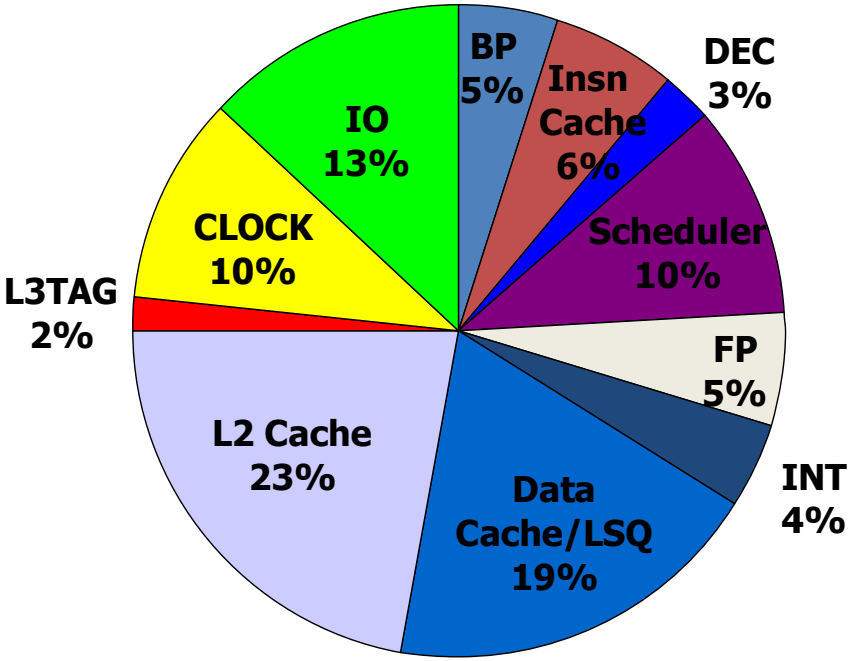
Trends in Power

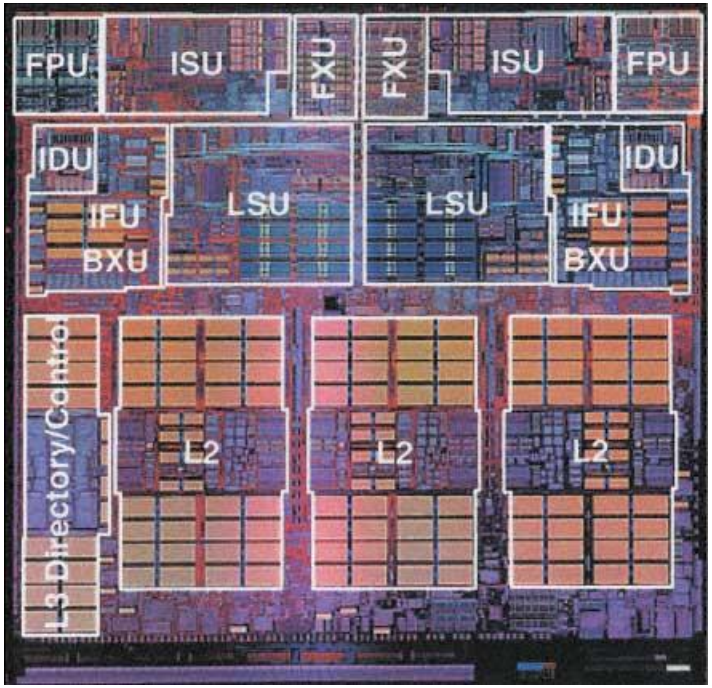
	386	Pentium	Pentium II	Pentium 4	Core2	Core i7	Cascade Lake
Year	1985	1993	1998	2001	2006	2009	2019
Technode (nm)	1500	350	180	130	65	45	14
Transistors (M)	0.3	3.1	5.5	42	291	731	8000
Voltage (V)	**5**	**3.3**	**2.9**	**1.7**	**1.3**	**1.2**	**1.2**
Clock (MHz)	16	66	200	1500	3000	3300	2.6-3.8K
Power (W)	1	16	35	**80**	**75**	**130**	**100-400**
Peak MIPS	6	132	600	4500	24000	52,800	582,400
MIPS/W	6	8	17	56	320	406	1,456

- Supply voltage decreasing over time
  - But "voltage scaling" is perhaps reaching its limits
- Emphasis on power starting around 2000
  - Resulting in slower frequency increases
  - Also note number of cores increasing (2 in Core 2, 4 in Core i7)

Processor Power Breakdown

- Power breakdown for IBM POWER4
  - Two 4-way superscalar, 2-way multi-threaded cores, 1.5MB L2
  - Big power components are L2, data cache, scheduler, clock, I/O
  - Implications on "complicated" versus "simple" cores





Moore's Effect #5: Psychological/Economic

- **Moore's Curve**: common interpretation of Moore's Law
  - "CPU performance doubles every 18 months"
  - Self fulfilling prophecy: 2X every 18 months is ~1% per week
    - Q: Would you add a feature that improved performance 20% if it would delay the chip 8 months?
  - Processors under Moore's Curve (arrive too late) fail spectacularly
    - E.g., Intel's Itanium, Sun's Millennium

Moore's Law in the Future

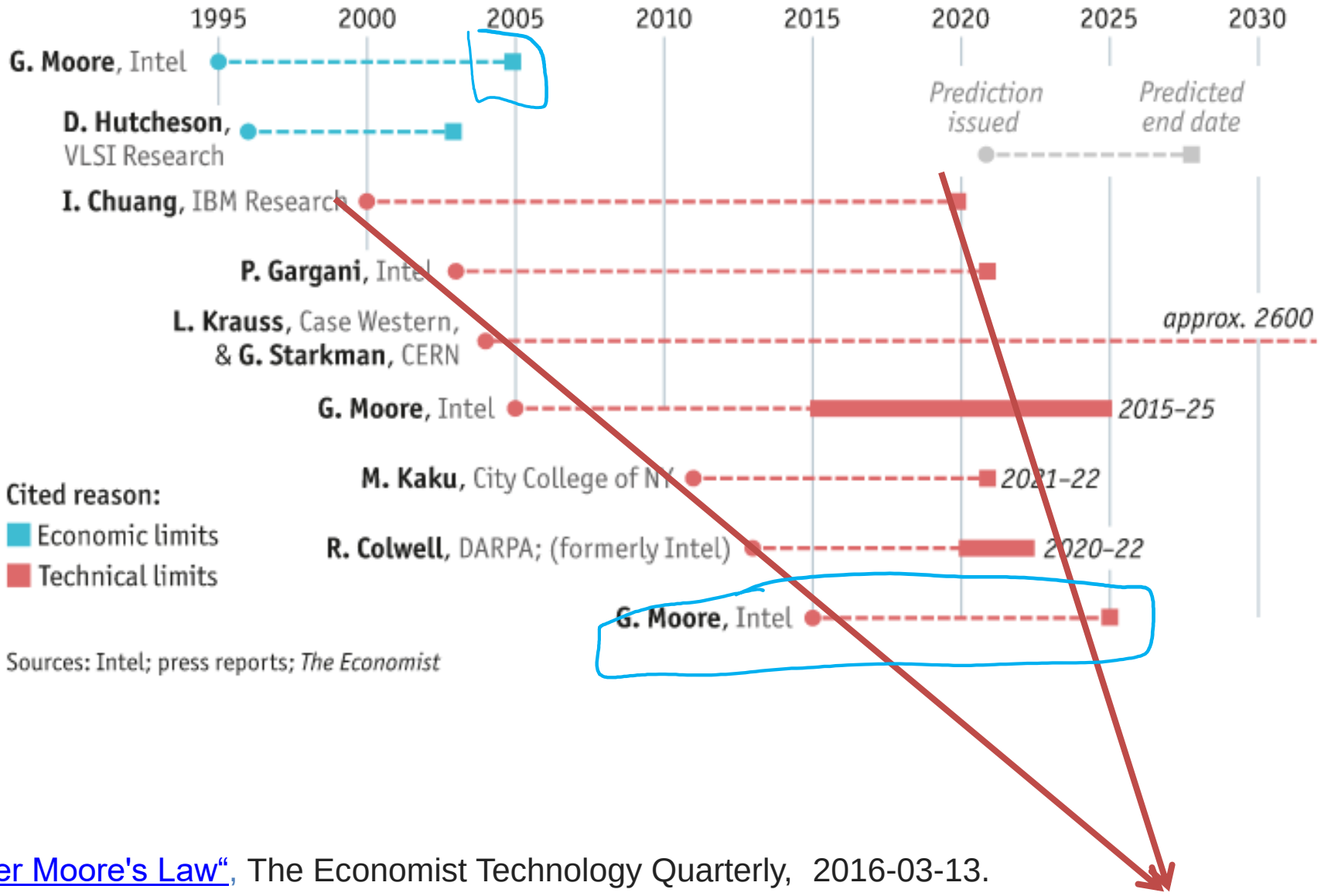
- Won't last forever, approaching physical limits
  - "If something must eventually stop, it can't go on forever"
  - But betting against it has proved foolish in the past
  - Perhaps will "slow" rather than stop abruptly

- Transistor count will likely continue to scale
  - "Die stacking" is on the cusp of becoming main stream
    - Uses the third dimension to increase transistor count (mostly for integrating logic/memory)
  - Wafer-scale computing – don't use packages, just use wafer

- But transistor performance scaling?
  - Running into physical limits
  - Example: gate oxide is less than 10 silicon atoms thick!
    - Can't decrease it much further
  - Power is becoming a limiting factor

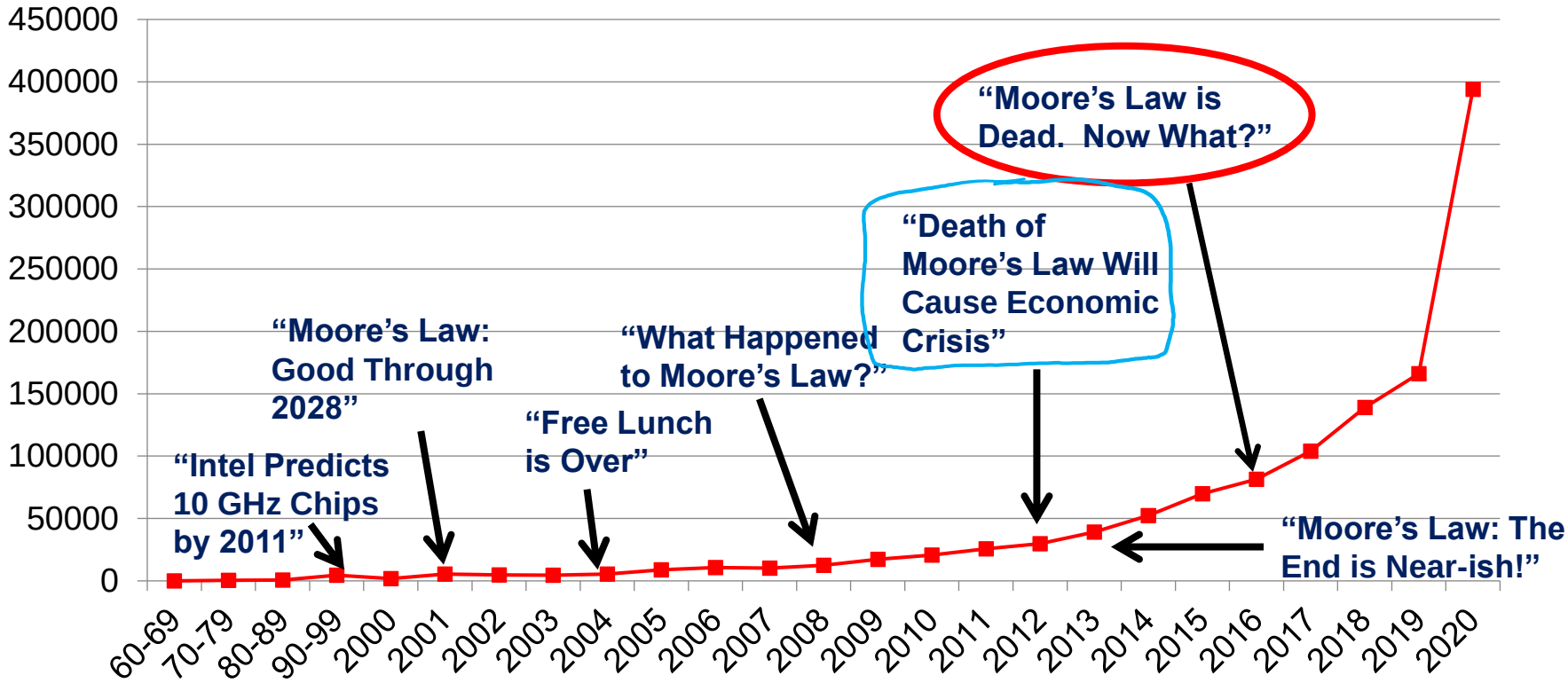
Faith no Moore

Selected predictions for the end of Moore's law



"After Moore's Law", The Economist Technology Quarterly, 2016-03-13.

Google Hits on End of Moore's Law



Summary: A Global Look at Moore

- Device scaling (Moore's Law)
  - Increases performance
    - Reduces transistor/wire delay
    - Gives us more transistors with which to reduce CPI
  - Reduces local power consumption
    - Which is quickly undone by increased integration
    - Aggravates power-density and temperature problems
  - Reduces unit cost
    - But increases startup cost
  - Aggravates reliability problem
    - But gives us the transistors to solve it via redundancy
  - Aggravates power problem

- (When) will we fall off Moore's Cliff (for real, this time?)
  - What's next: nanotubes, quantum-dots, optical, spin-tronics, DNA?

Technology Summary

- Has a first-order impact on computer architecture
  - Cost (die area)
  - Performance (transistor delay, wire delay)
  - **Changing rapidly**
- Most significant trends for architects (and thus 752)
  - More and more transistors
    - What to do with them? → integration → **parallelism**
  - Logic is improving faster than memory & cross-chip wires
    - "Memory wall" → caches, more integration

Rest of semester

Performance

Iron Law of Performance

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

CPI

(microarchitecture value added)

Performance

- Two definitions
  - **Latency (execution time)**: time to finish a fixed task
  - **Throughput (bandwidth)**: number of tasks in fixed time
  - Very different: throughput can exploit parallelism, latency cannot
    - Ex: baking bread
  - Often contradictory
  - Choose definition of performance that matches your goals

- Example: move people from A → B, 10 miles
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: **car = 10 min**, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

Performance Improvement

- Given a program of interest P…
- Processor A is X times faster than processor B if
  - Latency(P, A) = Latency(P, B) / X
  - Throughput(P, A) = Throughput(P, B) * X
- Processor A is X% faster than processor B if
  - Latency(P, A) = Latency(P, B) / (1+X/100)
  - Throughput(P, A) = Throughput(P, B) * (1+X/100)

- Car/bus example
  - Latency? Car is 3 times (and 200%) faster than bus
  - Throughput? Bus is 4 times (and 300%) faster than car

What is 'P' in Latency(P, A)?

- Program
  - Latency(A) makes no sense, processor executes **some** program
  - But which one?
- Actual target workload?
  - \+ Accurate
  - – Not portable/repeatable, overly specific, hard to pinpoint problems
- – Some representative benchmark program(s)?
  - \+ Portable/repeatable, pretty accurate
  - – Hard to pinpoint problems, may not be exactly what you run
- – Some small kernel benchmarks (micro-benchmarks)
  - \+ Portable/repeatable, easy to run, easy to pinpoint problems
  - – Not representative of complex behaviors of real programs

SPEC Benchmarks

- SPEC (Standard Performance Evaluation Corporation)
  - http://www.spec.org/
  - Consortium that collects, standardizes, and distributes benchmarks
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
  - Benchmark suites for CPU, Java, I/O, Web, Mail, etc.
  - Updated every few years: so companies don't target benchmarks

- SPEC CPU 2017 – most recent SPEC
  - 10 "integer" (e.g. gcc)
  - 13 "floating point" benchmarks (e.g. pov-ray -- ray tracing)
  - Written in C/C++ and Fortran

Other CPU Benchmarks

- Parallel benchmarks
  - SPLASH2: Stanford Parallel Applications for Shared Memory
  - NAS: another parallel benchmark suite
  - SPECopenMP: parallelized versions of SPECfp 2000)
  - SPECjbb: Java multithreaded database-like workload
- Transaction Processing Council (TPC)
  - TPC-C: On-line transaction processing (OLTP)
  - TPC-H/R: Decision support systems (DSS)
  - TPC-W: E-commerce database backend workload
  - Have parallelism (intra-query and inter-query)
  - Heavy I/O and memory components
- Last 10 years: GPU benchmarks (e.g., Parboil, Rodinia, …)
- Last 2 years: MLPerf, Embench

Pitfalls of Partial Performance Metrics

Danger: Partial Performance Metrics

- General public often equates performance with particular architecture aspects, **neglecting CPI**!
  - ISA Bitwidth (16,32,64,128 bit, yay!)
  - **Frequency**
  - # Cores
- Which processor would you buy?
  - Processor A: CPI = 2, clock = 5 GHz
  - Processor B: CPI = 1, clock = 3 GHz
  - Probably A, but B is faster (assuming same ISA/compiler)
- Classic example
  - 800 MHz PentiumIII faster than 1 GHz Pentium4!

- **Meta-point: danger of partial performance metrics!**

MIPS Metric (ignores instr. count)

- (Micro) architects often ignore dynamic instruction count
  - Typically work in one ISA/one compiler → treat it as fixed

- Without inst/program, Iron Law becomes:
  - Latency: seconds / insn = (cycles / insn) * (seconds / cycle)
  - Throughput: **insn / second** = (insn / cycle) * (cycles / second)

- **MIPS** (millions of instructions per second)
  - **Instructions/second * 10^{-6}**
  - **Cycles / second**: clock frequency (in MHz)
  - Example: CPI = 2, clock = 500 MHz → 0.5 * 500 MHz = 250 MIPS

- Pitfall of MIPS: may vary inversely with actual performance
  - Compiler removes insns, program gets faster, MIPS goes down…
  - Other problems:
    - Some optimizations actually add instructions
    - Work per instruction varies (e.g., multiply vs. add, FP vs. integer)
    - ISAs are not equivalent

MFLOPS (MegaFLOPS)

- **MFLOPS**: like MIPS, but counts only FP ops, because…
  - + FP ops can't be optimized away (by compiler)
  - + FP ops have longest latencies (FP Units are larger)
  - + FP ops are same across machines
- Pitfalls:
  - – Many CPU programs are "integer" – light on FP
  - – Loads from memory can take much longer than even an FP divide
  - – Even FP instruction sets are not equivalent
- Upshot: Neither MIPS nor MFLOPS are universal

Cycles per Instruction (CPI)

- This course is mostly about improving **CPI**
  - Cycle/instruction for average instruction
  - IPC = 1/CPI
    - Used more frequently than CPI, but harder to compute with
  - Different instructions have different cycle costs
    - E.g., integer add typically takes 1 cycle, FP divide takes > 10
  - Assumes you know something about instruction frequencies

- CPI example
  - A program executes equal integer, FP, and memory operations
  - Cycles per instruction type: integer = 1, memory = 2, FP = 3
  - What is the CPI? (0.33 * 1) + (0.33 * 2) + (0.33 * 3) = 2
  - **Caveat**: this sort of calculation ignores dependencies completely
    - Kind-of useful for back-of-the-envelope arguments?

Another CPI Example

- Assume a processor with instruction frequencies and costs
  - Integer ALU: 50%, 1 cycle
  - Load: 20%, 5 cycles
  - Store: 10%, 1 cycle
  - Branch: 20%, 2 cycles
- For now, lets not assume any parallelism (in-order core?)
- Which change would improve performance more?
  - A: Branch prediction to reduce branch cost to 1 cycle?
  - B: A bigger data cache to reduce load cost to 3 cycles?

- Compute CPI
  - Base: 0.5*1 + 0.2*5 + 0.1*1 + 0.2*2 = 2
  - A = 0.5*1 + 0.2*5 + 0.1*1 + 0.2*1 = 1.8
  - B = 0.5*1 + 0.2*3 + 0.1*1 + 0.2*2 = 1.6 (**winner**!)

Improving CPI

- This course focuses on improve CPI instead of frequency
  - Historically, clock accounts for 70%+ of performance improvement
    - Achieved via deeper pipelines
  - That has been forced to change
    - Deep pipelining is **not** power efficient
    - Physical speed limits are approaching
    - 1 GHz: 1999, 2 GHz: 2001, 3 GHz: 2002, 3.8 GHz: 2004, 5 GHz: 2008
    - Intel Core 2: 1.8 – 3.2 GHz in 2008
  - **Techniques we'll look at:**
    - **Caching, speculation, multiple issue, out-of-order issue**
    - **Vectors, multiprocessing, more …**

- Moore helps because CPI reduction requires transistors
  - The definition of parallelism is "more transistors"
  - But best example is caches

Performance Rules of Thumb

- Make common case fast
  - "**Amdahl's Law**"
    - $Time_{optimized} = Time_{orig} * fraction_x / Speedup_x + Time_{orig} * (1\text{-}fraction_x)$
    - $Speedup = Time_{orig} / Time_{optimized}$
    - $Speedup = 1/((1 - fraction_x) + fraction_x/Speedup_x)$
  - Corollary: don't optimize 5% to the detriment of the other 95%
  - $Speedup_{overall} = 1 / ((1 - 5\%) + 5\%/\infty) \sim= 1.05$

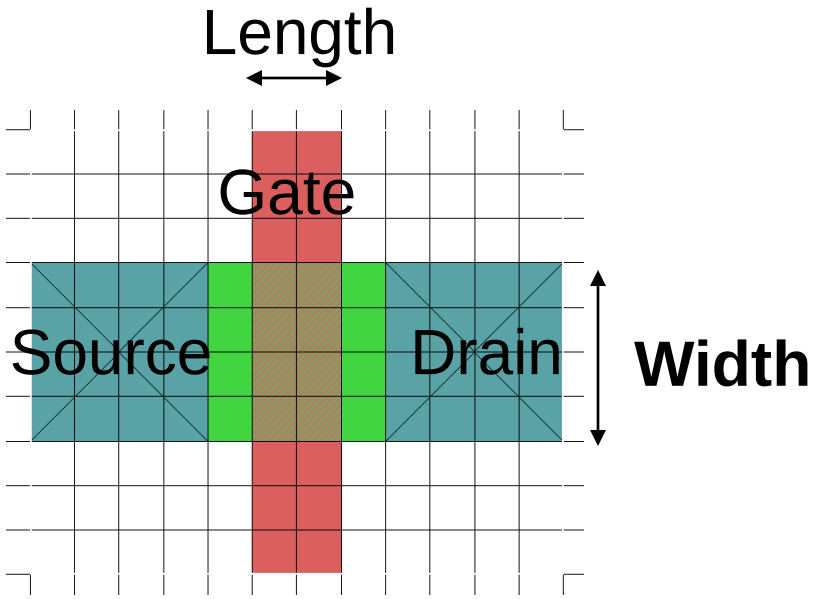
- Build a balanced system
  - Don't over-engineer capabilities that cannot be utilized
  - Try to be "bound" by the most expensive resource (if not everywhere)

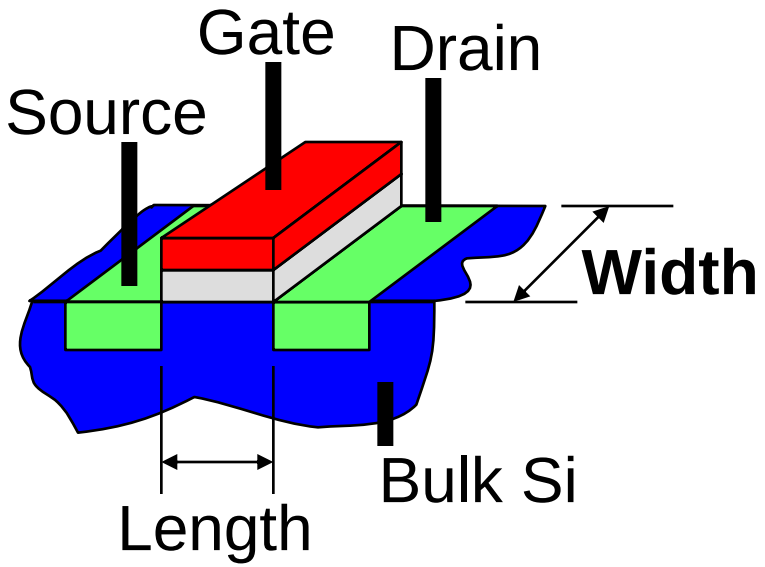
- Design for actual, not peak performance
  - For actual performance X, machine capability must be > X

Next Time…

Hidden Bonus Slides

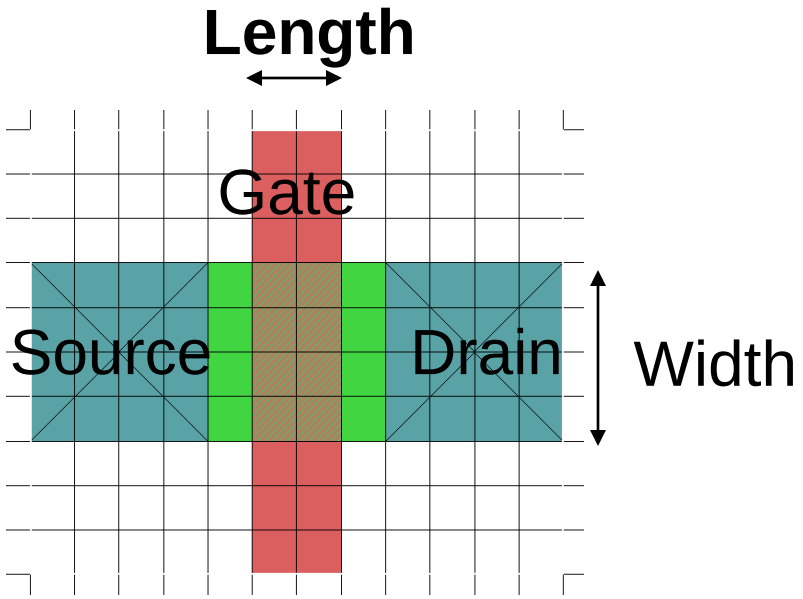
Transistor Geometry: Width

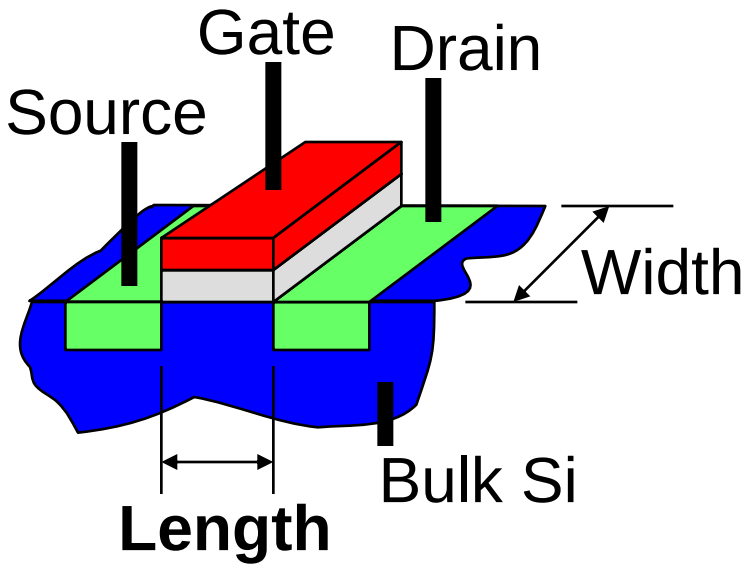




- **Transistor width**, set by designer for each transistor
- Wider transistors:
  - **Lower resistance** of channel (increases drive strength) – good!
  - But, **increases capacitance** of gate/source/drain – bad!
- Result: set width to balance these conflicting effects

Transistor Geometry: Length & Scaling

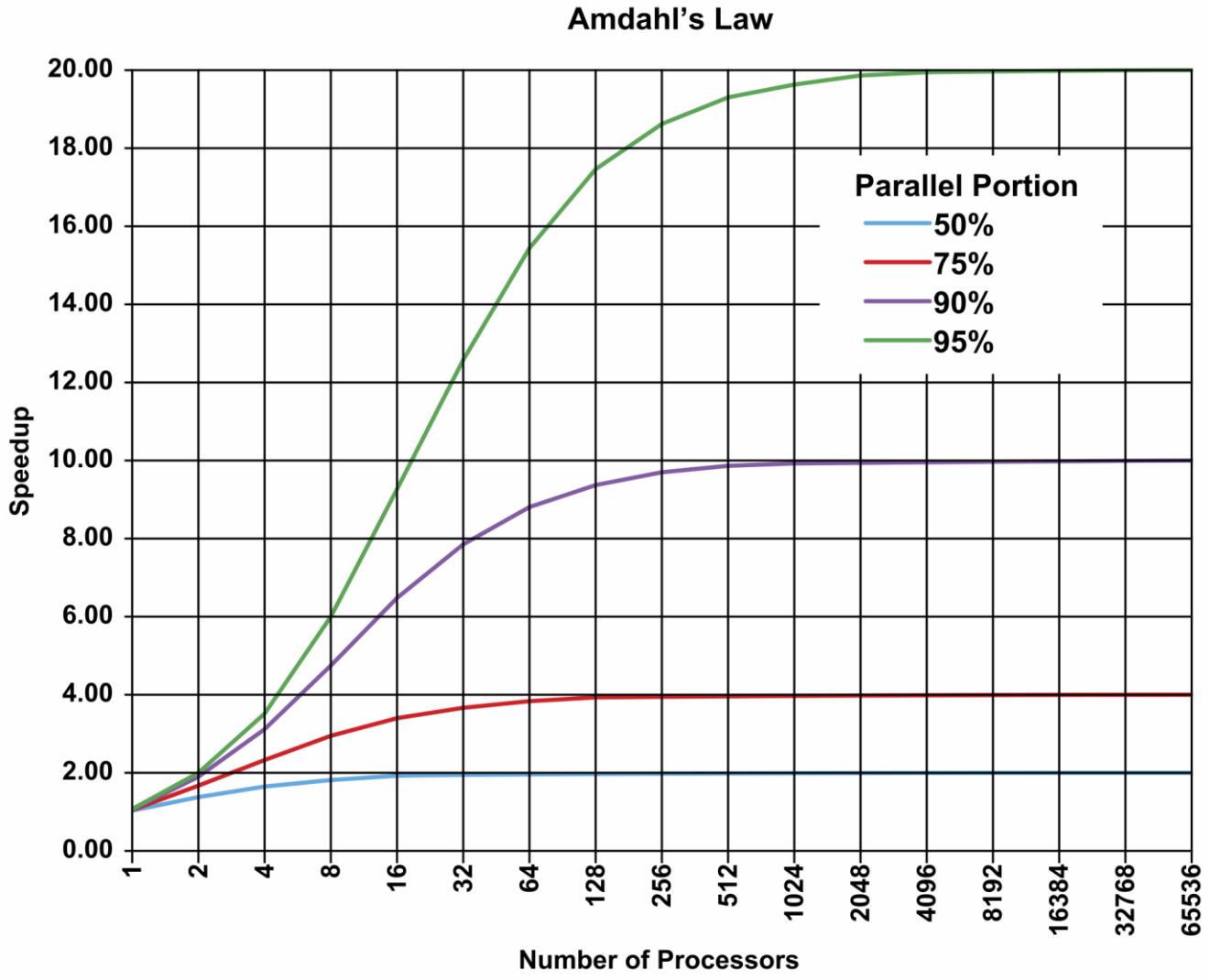




Diagrams © Krste Asanovic, MIT

- **Transistor length**: characteristic of "process generation"
  - 45nm refers to the transistor gate length, same for all transistors
- Shrink transistor length:
  - Lower resistance of channel (shorter) – good!
  - Lower gate/source/drain capacitance – good!
- Result: switching speed improves linearly as gate length shrinks

Amdahl's Law Graph



Source: Wikipedia

Little's Law

- Key Relationship between latency and bandwidth:
  - Average number in system = arrival rate * avg holding time

- Example:
  - How big a wine cellar should I build?
  - My family drinks (and buys) an average of 4 bottles per week
  - On average, I want to age my wine 5 years

  - # bottles in cellar = 4 bottles/week * 52 weeks/year * 5 years
    = 1040 bottles (!)

More Little's Law

- How many outstanding cache misses?
  - Want to sustain 5 GB/s bandwidth
  - 64 byte blocks
  - 100ns miss latency
- Requests in system = arrival rate * time in system
  = (5 GB/s / 64 byte blocks) * 100 ns
  = 8 misses
- That's an **AVERAGE**.
  - Need to support many more if we hope to sustain this bandwidth
  - Rule of thumb: 2X

Reliability

Technology Basis for Reliability

- As transistors get smaller, they are less reliable
  - Wasn't a problem a few years ago, becoming a big problem
  - Small capacitance means fewer electrons represent 1 or 0

- **Transient faults**
  - A bit "flips" randomly, **temporarily**
  - Cosmic rays and such (more common at higher altitudes!)
  - Memory cells (especially memory) vulnerable today, logic soon

- **Permanent (hard) faults**
  - A gate or memory cell wears out, **breaks and stays broken**
  - Temperature & electromigration gradually deform components

- Solution for both: use **redundancy** to detect and tolerate

Aside: Memory Technology Families

- **SRAM**: "static" RAM
  - Used on processor chips (same transistors as used for "logic")
  - Storage implemented as 6 transistors per bit
    - An inverter pair (2 transistors each) + two control transistors
  - Optimized for speed first, then secondarily density and power
- **DRAM (volatile memory)**: "dynamic" RAM
  - Different manufacturing steps, not typically used on processor chips
  - Storage implemented as one capacitor + 1 transistor per bit
  - Optimized for density and cost
- **Flash (non-volatile memory)**:
  - Used for solid state storage
  - Slower than DRAM, but non-volatile
- Disk is also a "technology", but isn't transistor-based

Memory Error Detection

- Idea: add extra state to memory to detect a bit flip

- **Parity**: simplest scheme
  - One extra bit, detects any single bit flip
  - Parity bit = XOR($data_{N-1}$, …, $data_1$, $data_0$)

- Example:
  - 010101 0^1^0^1^0^1 = "1" so parity is "odd" (versus "even")
  - So, store "010101 **1**" in memory
  - When you read the data, and re-calculate the parity, say
    - 01**1**101 **1**, if the parity bit doesn't match, error detected

- Multiple bit errors? more redundancy can detect more

Memory Error Detection

- What to do on a parity error?

- **Crash**
  - **Dead programs tell no lies**
  - Fail-stop is better than silent data corruption
  - Avoiding writing that "$1m check"

- For user-level data, OS can kill just the program
  - Not the whole system, unless it was OS data

- Alternative: correct the error

SEC Error Correction Code (ECC)

- **SEC**: single-error correct (a hamming code)

- Example: Four data bits, three "code" bits
  - $d_1\ d_2\ d_3\ d_4\ \mathbf{c_1\ c_2\ c_3} \rightarrow \mathbf{c_1\ c_2}\ d_1\ \mathbf{c_3}\ d_2\ d_3\ d_4$
  - $c_1 = d_1 \wedge d_2 \wedge d_4$, $c_2 = d_1 \wedge d_3 \wedge d_4$, $c_3 = d_2 \wedge d_3 \wedge d_4$
  - Syndrome: $c_i \wedge c'_i = 0$? no error
    - Otherwise, then $c_3'\ c_2'$, c_1' points to flipped-bit

- Working example
  - Original data = 0110 $\rightarrow$ $c_1 = 1$, $c_2 = 1$, $c_3 = 0$
  - Flip d_2 = 0010 $\rightarrow$ $c'_1 = 0$, $c'_2 = 1$, $c'_3 = 1$
    - Syndrome = 101 (binary 5) $\rightarrow$ 5th bit? D_2
  - Flip c_2 $\rightarrow$ $c'_1 = 1$, $c'_2 = 0$, $c'_3 = 0$
    - Syndrome = 010 (binary 2) $\rightarrow$ 2nd bit? c_2

SECDED Error Correction Code (ECC)

- **SECDED**: single error correct, double error detect

- Example: D = 4 → C = 4
  - $d_1\ d_2\ d_3\ d_4\ \mathbf{c_1\ c_2\ c_3} \rightarrow \mathbf{c_1\ c_2}\ d_1\ \mathbf{c_3}\ d_2\ d_3\ d_4\ \mathbf{c_4}$
  - $c_4 = c_1 \wedge c_2 \wedge d_1 \wedge c_3 \wedge d_2 \wedge d_3 \wedge d_4$
  - Syndrome == 0 and $c'_4 == c_4$ → no error
  - Syndrome != 0 and $c'_4 != c_4$ → 1-bit error
  - Syndrome != 0 and $c'_4 == c_4$ → 2-bit error
  - Syndrome == 0 and $c'_4 != c_4$ → c_4 error
  - **In general: $C = \log_2 D + 2$**
- Many machines today use 64-bit SECDED code
  - C = 8 (64bits + 8bits = 72bits, 12% overhead)
  - ChipKill - correct any aligned 4-bit error
    - If an entire memory chips dies, the system still works!

Another Issue: Process Variability

- As transistors get smaller…
  - Small geometric variations have relatively larger impact

- Example: Gate oxide thickness
  - In Intel's 65nm process: only 1.2 nm, just a few molecules thick!
  - Small variation in gate oxide thickness impacts speed and energy
    - Too thick: slow transistor
    - Too thin: exponential increase in leakage (static power)

- Some parts of the chip slow than others (impacts yield)
- Complicates high-speed memory designs
- Limits circuit techniques ("dynamic" versus "static" circuits)
  - Intel's Nehalem (Core i7) moved to all static circuits

Reliability

- **Mean Time Between Failures (MTBF)**
  - How long before you have to reboot or buy a new one

- CPU reliability is small in the grand scheme
  - Software most unreliable component in a system
    - Much more difficult to specify & test
    - Much more of it
  - Most unreliable hardware component: disk
    - Subject to mechanical wear

Moore's Bad Effect on Reliability

- Wasn't a problem until 5-10 years ago…
  - Except for transient-errors on chips in orbit (satellites)
  - …a problem already and getting worse all the time
  - Transient faults:
    - Small (low charge) transistors are more easily flipped
    - Even low-energy particles can flip a bit now
  - Permanent faults:
    - Small transistors and wires deform and break more quickly
    - Higher temperatures accelerate the process
- Progression of transient faults
  - Memory (DRAM) was hit first: denser, smaller devices than SRAM
  - Then on-chip memory (SRAM)
  - Logic is starting to have problems…

Moore's Good Effect on Reliability

- The key to providing reliability is **redundancy**
  - The same scaling that makes devices less reliable…
  - Also increase device density to enable redundancy

- Examples
  - Error correcting code for memory (DRAM) and caches (SRAM)
  - Core-level redundancy: paired-execution, hot-spare, etc.
  - Intel's Core i7 (Nehalem) uses 8 transistor SRAM cells
    - Versus the standard 6 transistor cells

- Big open questions
  - Can we protect logic efficiently? (without 2x or 3x overhead)
  - Can architectural techniques help hardware reliability?
  - Can software techniques help?

Power Implications on Software

- Software-controlled dynamic voltage/frequency scaling
  - OS? Application?
  - Example: video decoding
    - Too high a clock frequency – wasted energy (battery life)
    - Too low a clock frequency – quality of video suffers
- Managing low-power modes
  - Don't want to "wake up" the processor every millisecond
- Tuning software
  - Faster algorithms can be converted to lower-power algorithms
  - Via dynamic voltage/frequency scaling
- Exploiting parallelism & heterogeneous cores
  - NVIDIA Tegra 3: 5 cores (4 "normal" cores & 1 "low power" core)
- Specialized hardware accelerators