National Yang Ming Chiao Tung University
Computer Architecture & System Lab

# Lecture 4: Technology Trends and Quantitative Design and Analysis for Performance

## CS10014 Computer Organization

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao University

# Acknowledgements and Disclaimer

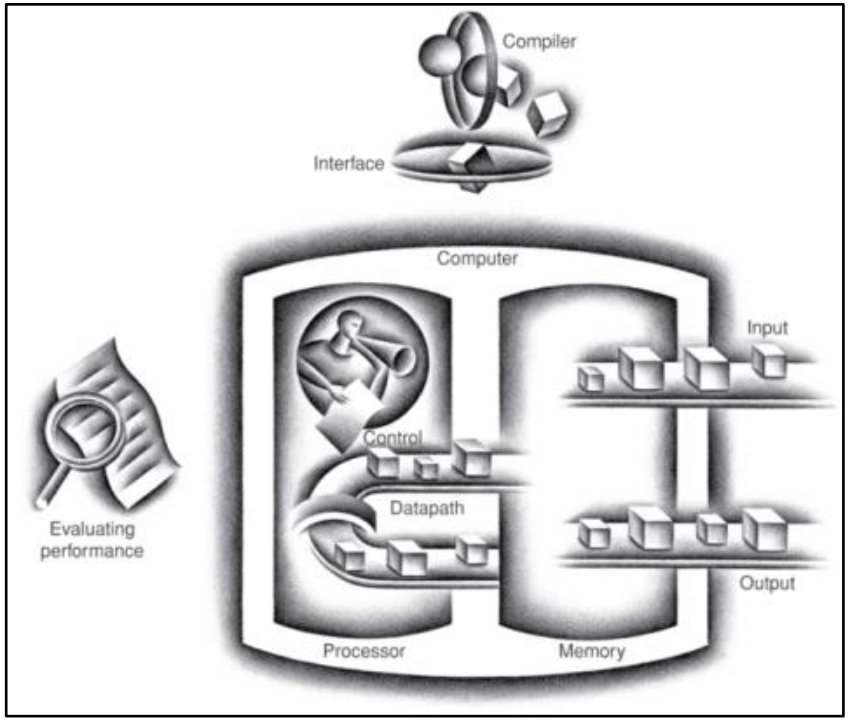
- Slides were developed in the reference with
  - CS 61C at UC Berkeley
    - https://inst.eecs.berkeley.edu/~cs61c/sp23/
  - CS 252 at UC Berkeley
    - https://people.eecs.berkeley.edu/~culler/courses/cs252-s05/
  - CSCE 513 at University of South Carolina
    - https://passlab.github.io/CSCE513/

# Outline

- Computer organization & Architecture
- Great ideas of computer architecture
- Performance Analysis & Technology trends

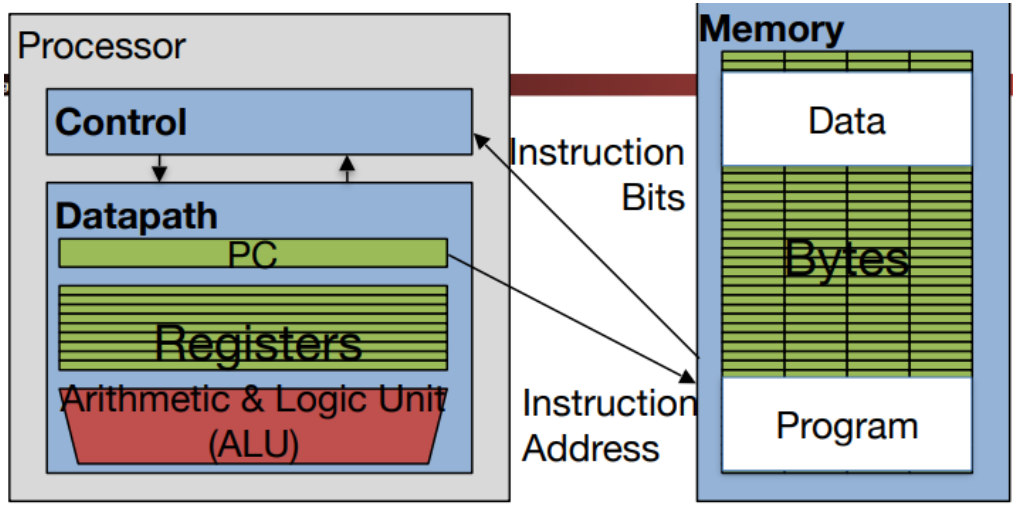
# Components of a Computer

- Two core parts
  - Processor and memory
- Input/output systems
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, flash drive
  - Network adapters
    - For communicating with other computers



# Inside the Processor (CPU)

- Datapath
  - Performs operations on data
- Control
  - Sequences datapath, memory, …
- Cache memory
  - Small fast SRAM memory for immediate access to data



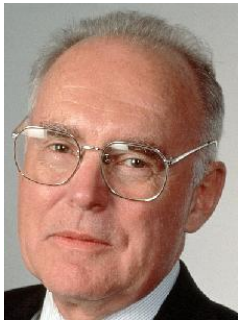
# Great Ideas in Computer Architectures

- **Abstraction** (Layers of representation/interpretation)
- **Moore's** Law
- Performance via **pipelining**
- Performance via **parallelism**
- **Hierarchy** of Memories
- **Dependability** via Redundancy

# Great Ideas: “Abstraction”

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

# Great Ideas: “Moore’s Law”

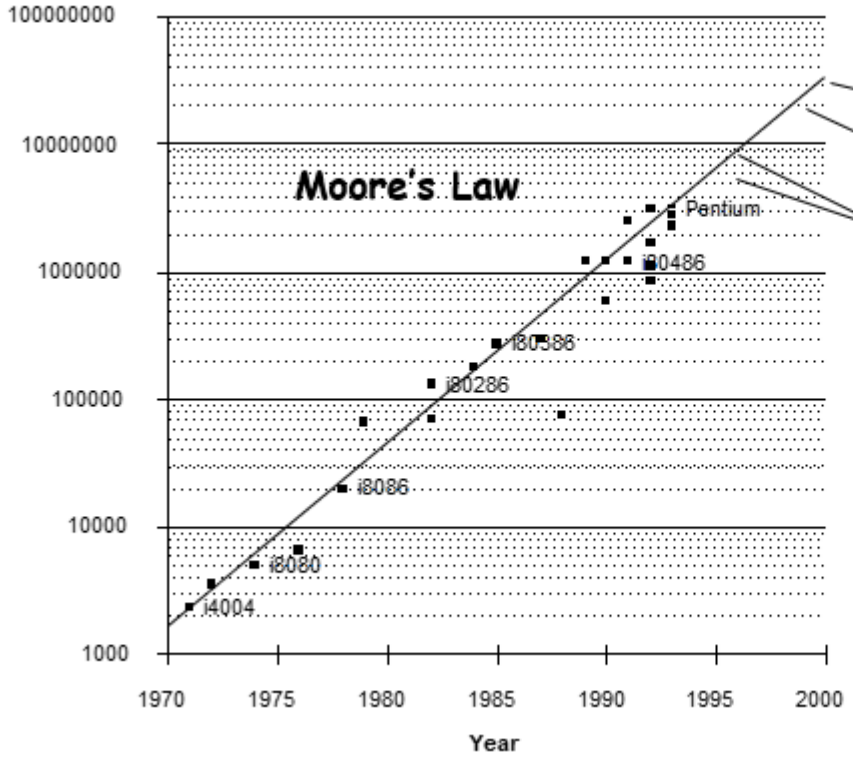


Gordon Moore
Intel
Cofounder
B.S. Cal 1950!

- Predicted 2x transistors/chip every 2 years (1965)
  - This trend would continue for the foreseeable future
- **Increasing circuit density~= increasing frequency ~= increasing performance**
  - Buying faster processors (higher frequency)

# Great Ideas: “Moore’s Law”

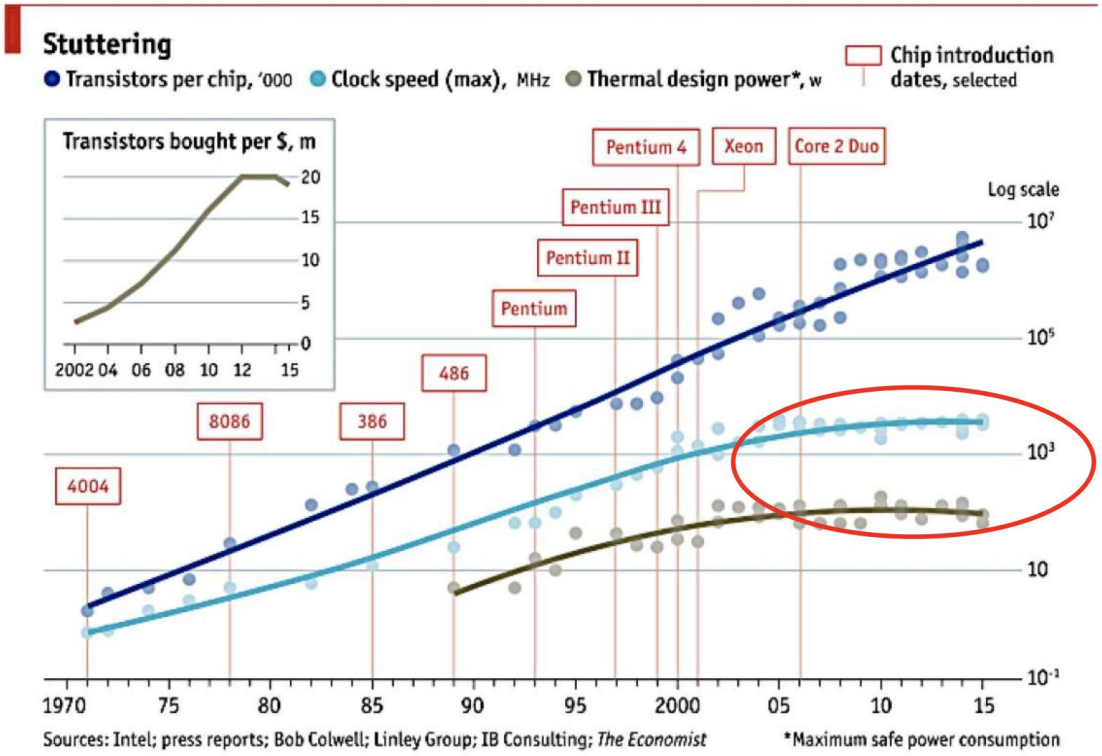
- # of transistors on an integrated circuit will double every 18 months




Itanium II: 241 million
Pentium 4: 55 million
Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

**CMOS improvements:**
- **Die size: 2X every 3 yrs**
- **Line width: halve / 7 yrs**

# Increasing transistors is not getting efficient



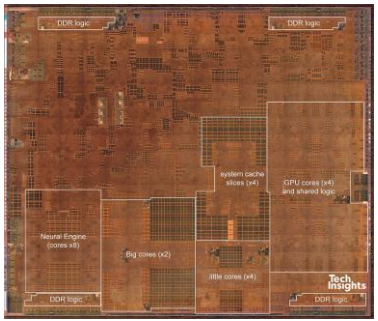

General purpose processor is not getting faster and power-efficient because of **Slowdown of Moore's Law and Dennard Scaling**

# Domain-Specific Accelerators

- **Domain-Specific Accelerators (DSAs)**
  - A suite of accelerators on chip are rising
  - Applications will only use a subset of processors/accelerators at a time
  - Employs DSAs to keep the performance improvement of hardware


**2010 Apple A4**
65 nm TSMC 53 mm$^2$
**4 accelerators**


**2014 Apple A8**
20 nm TSMC 89 mm$^2$
**28 accelerators**

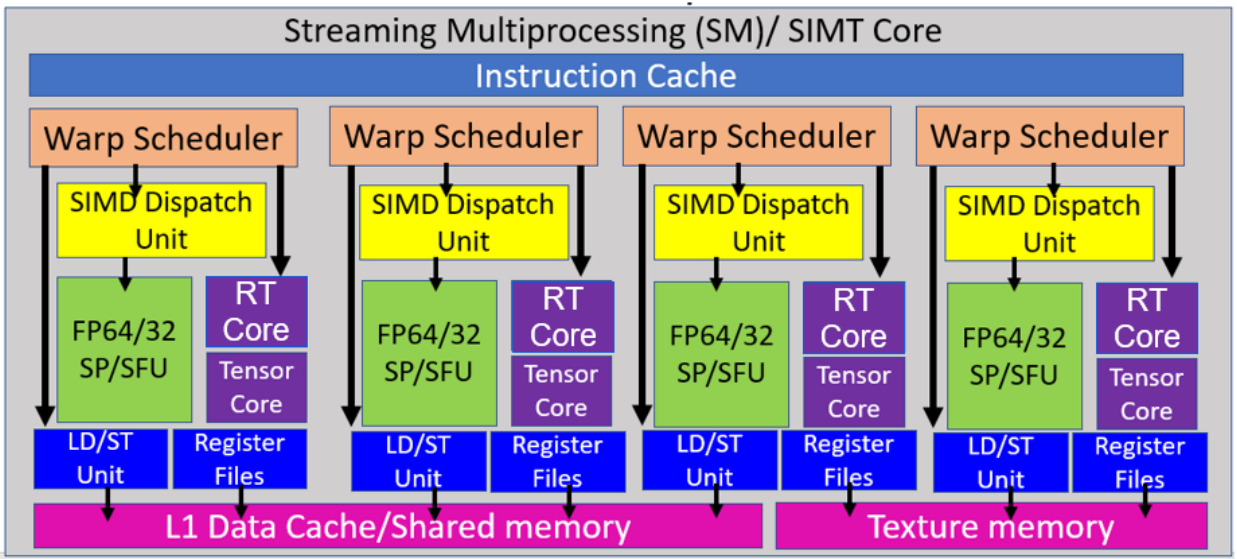


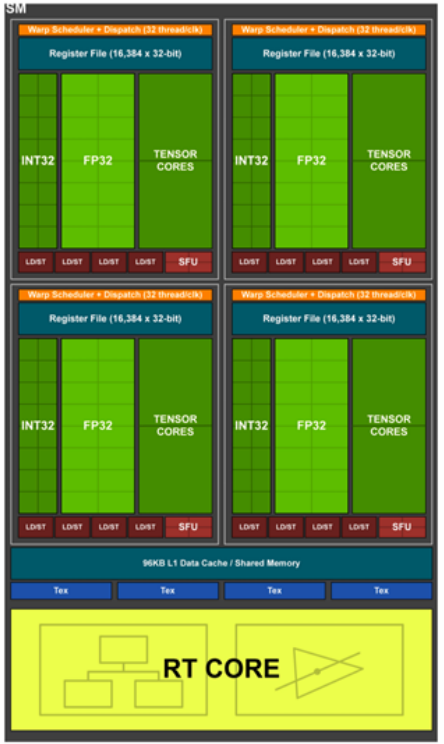

**2019 Apple A12**
7 nm TSMC 83 mm$^2$
**42 accelerators**

https://edge.seas.harvard.edu/files/edge/files/alp.pdf

# Graphic Processing Unit (GPU)

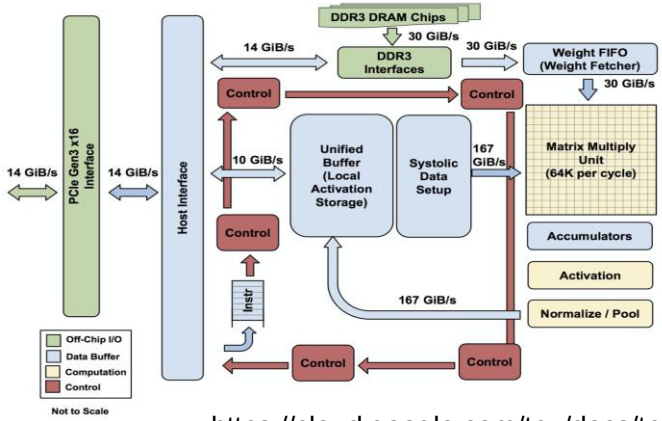
- GPU includes FP, SFU (Special Functional Unit), Ray Tracing (RT) Core, and Tensor Core



# Google Tensor Processing Unit (TPU)

- Systolic-array accelerator
  - Data-flow Accelerator
  - V1: Inference only
  - V2: Training with bfloat
  - V3: 2X powerful than v2
- Edge TPU
  - Coral Dev Board
  - 4 TOPS
  - 2 TOPS/Watt
  - Support TensorFlow Lite

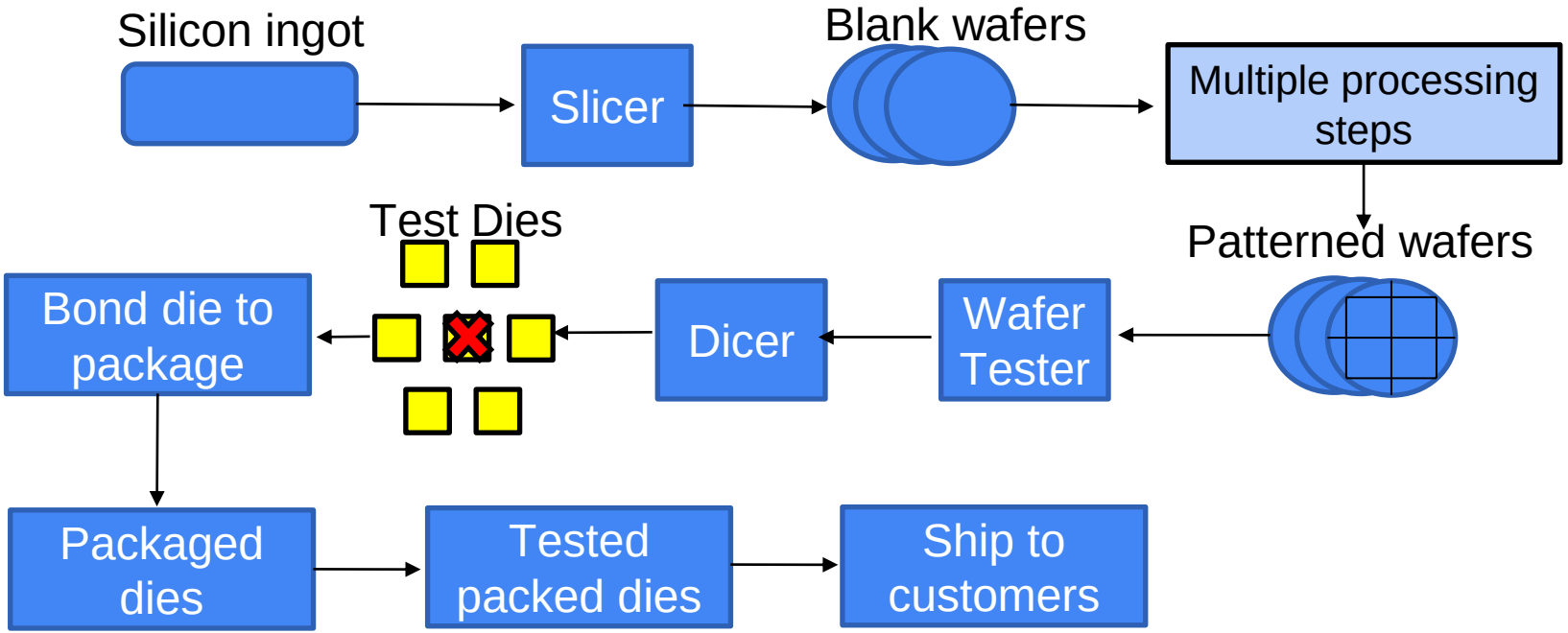



https://cloud.google.com/tpu/docs/tpus


https://coral.ai/products/

# Manufacturing ICs

- Yield: proportion of working dies per wafer



# Integrated Circuit Cost

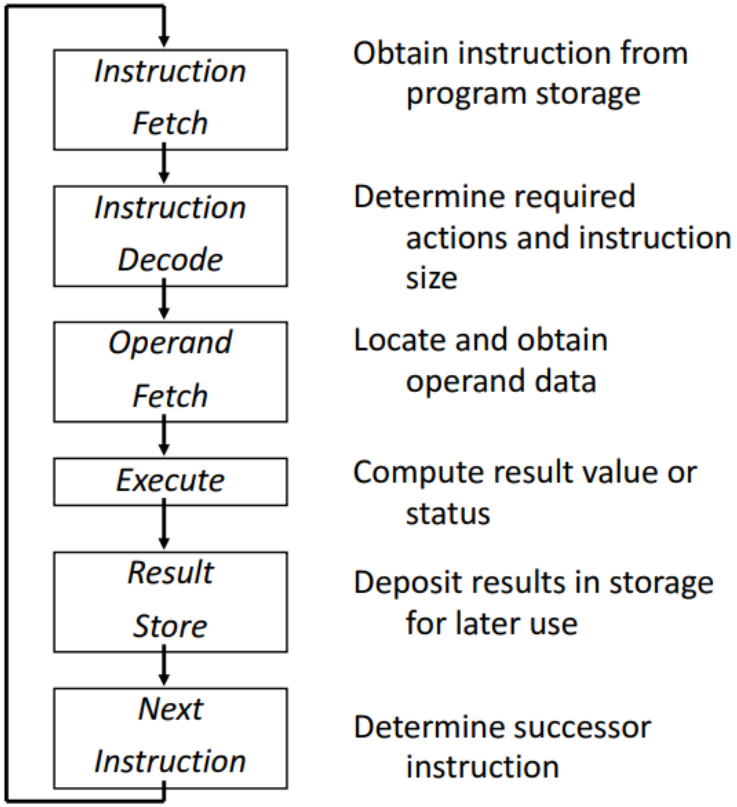
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing processing
  - Die area determined by architecture and circuit design

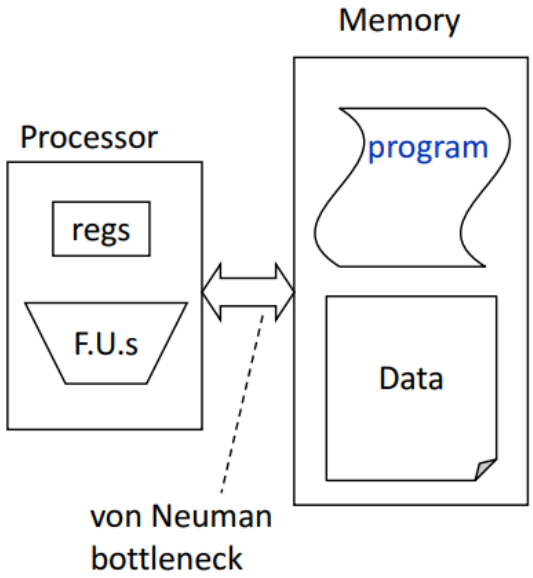
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

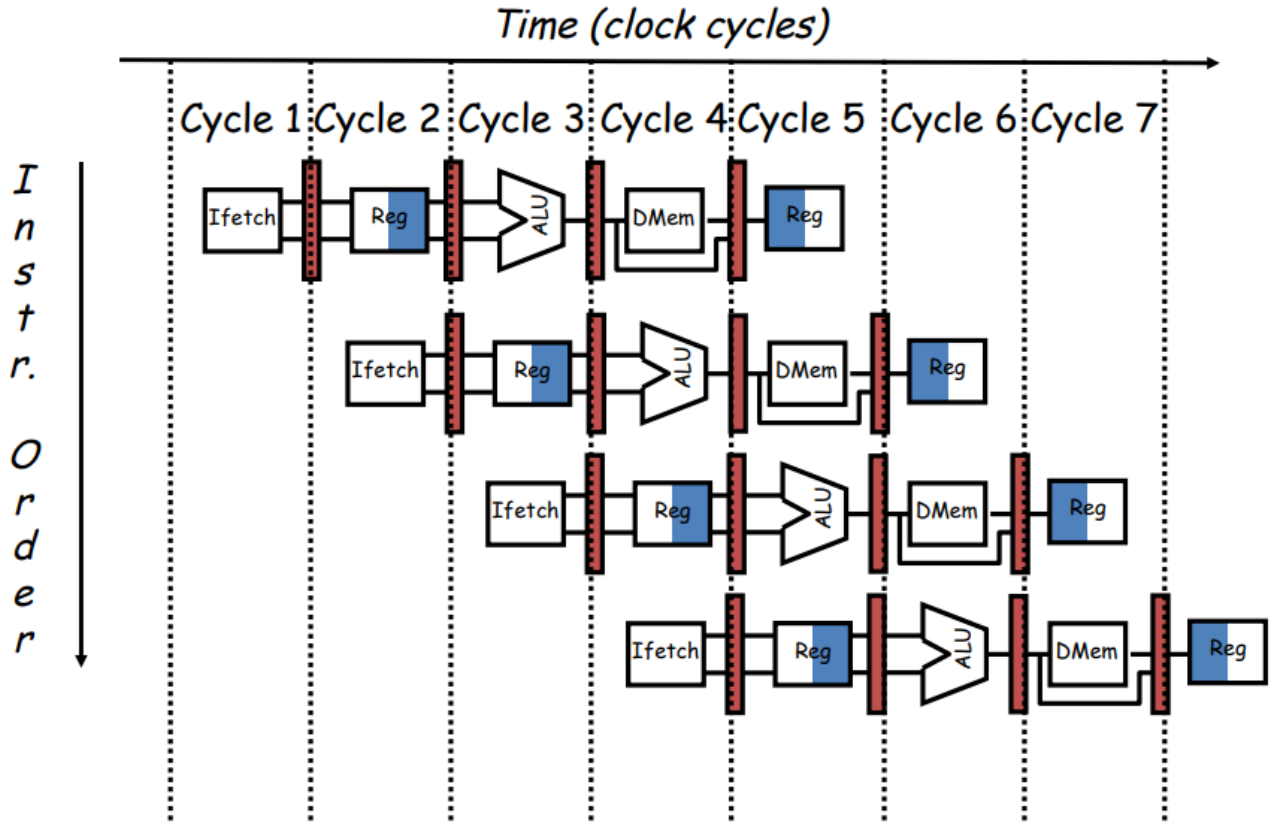
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

# Great Ideas: “Pipeline”

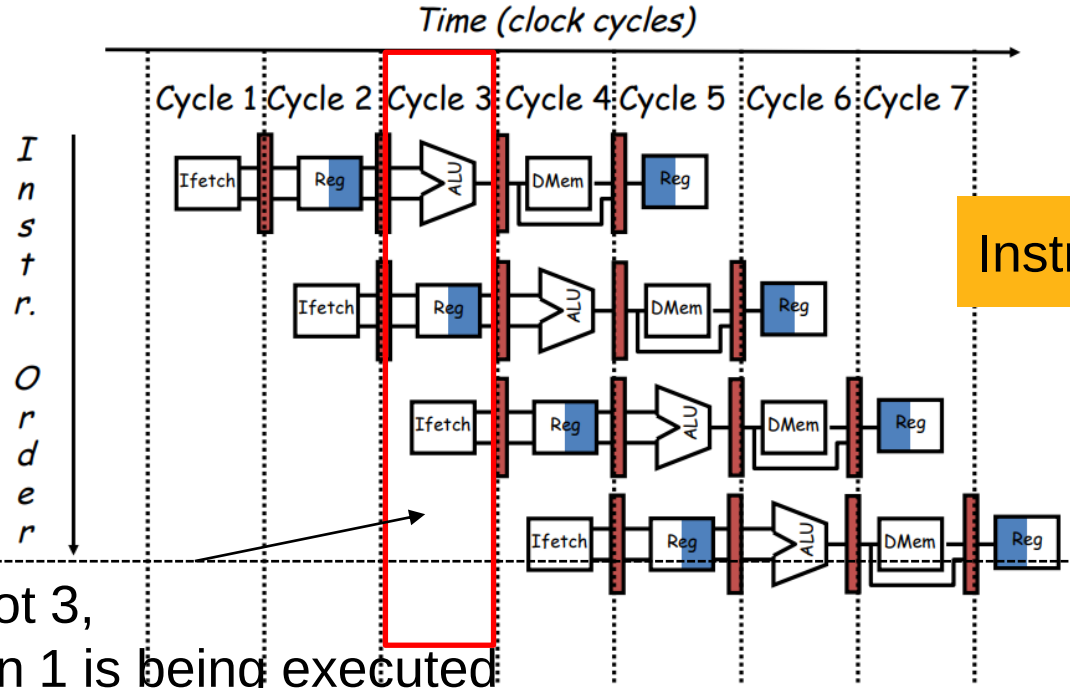


# Pipelined Instruction Execution



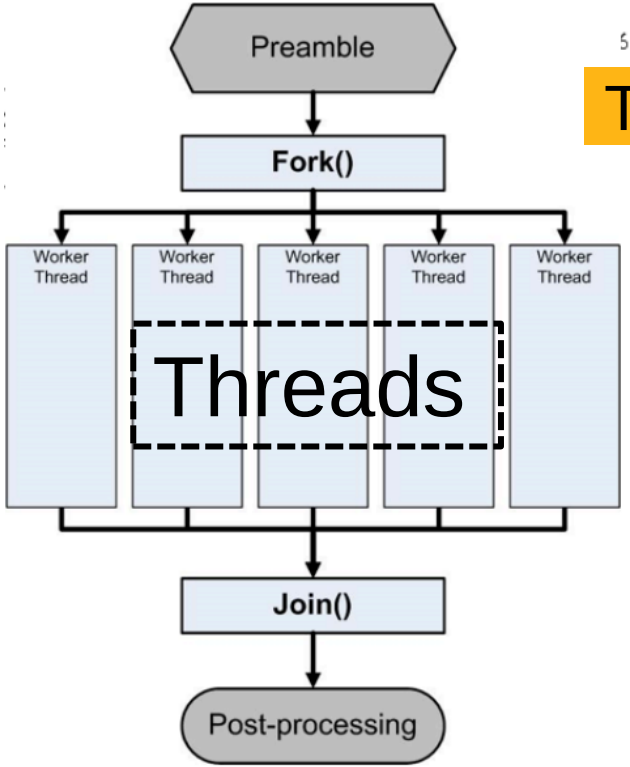
# Great Ideas: “Parallelism”




Instruction-Level Parallelism

In time slot 3,
Instruction 1 is being executed
Instruction 2 is being decoded
And Instruction 3 is being fetched from memory

# Great Ideas: “Parallelism”

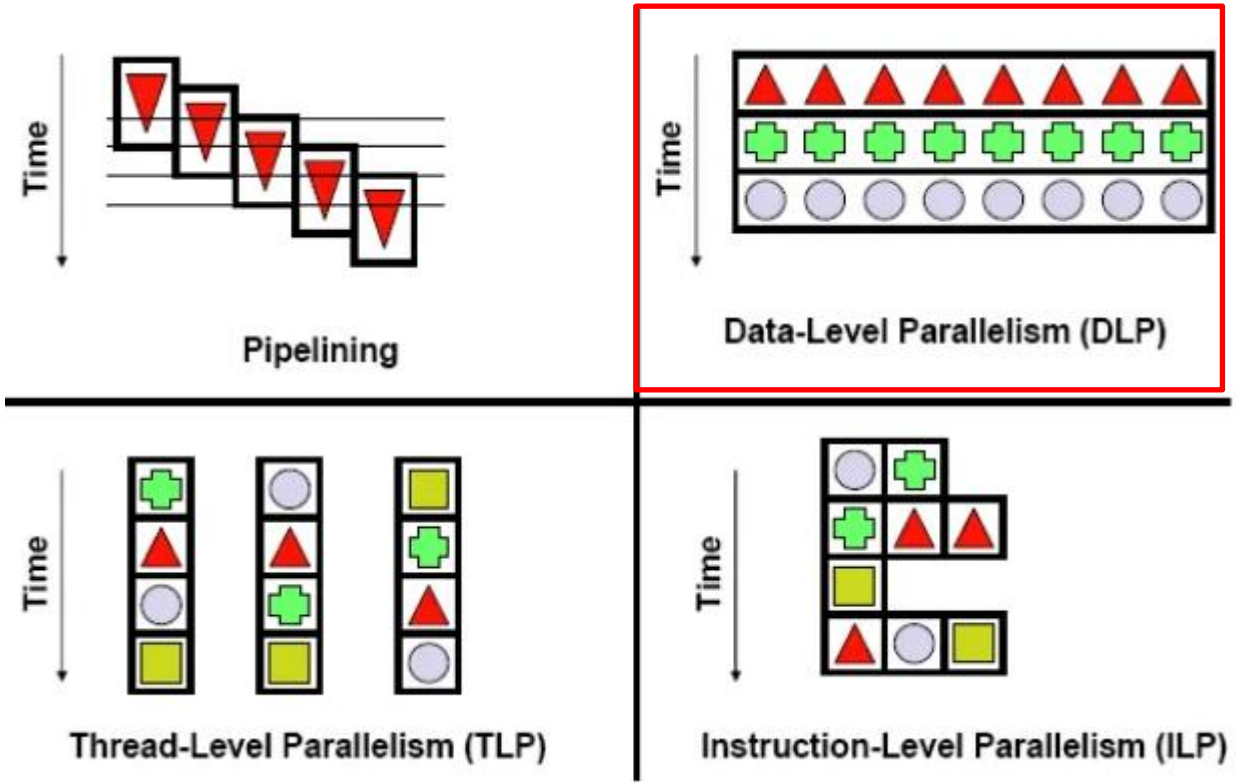



5

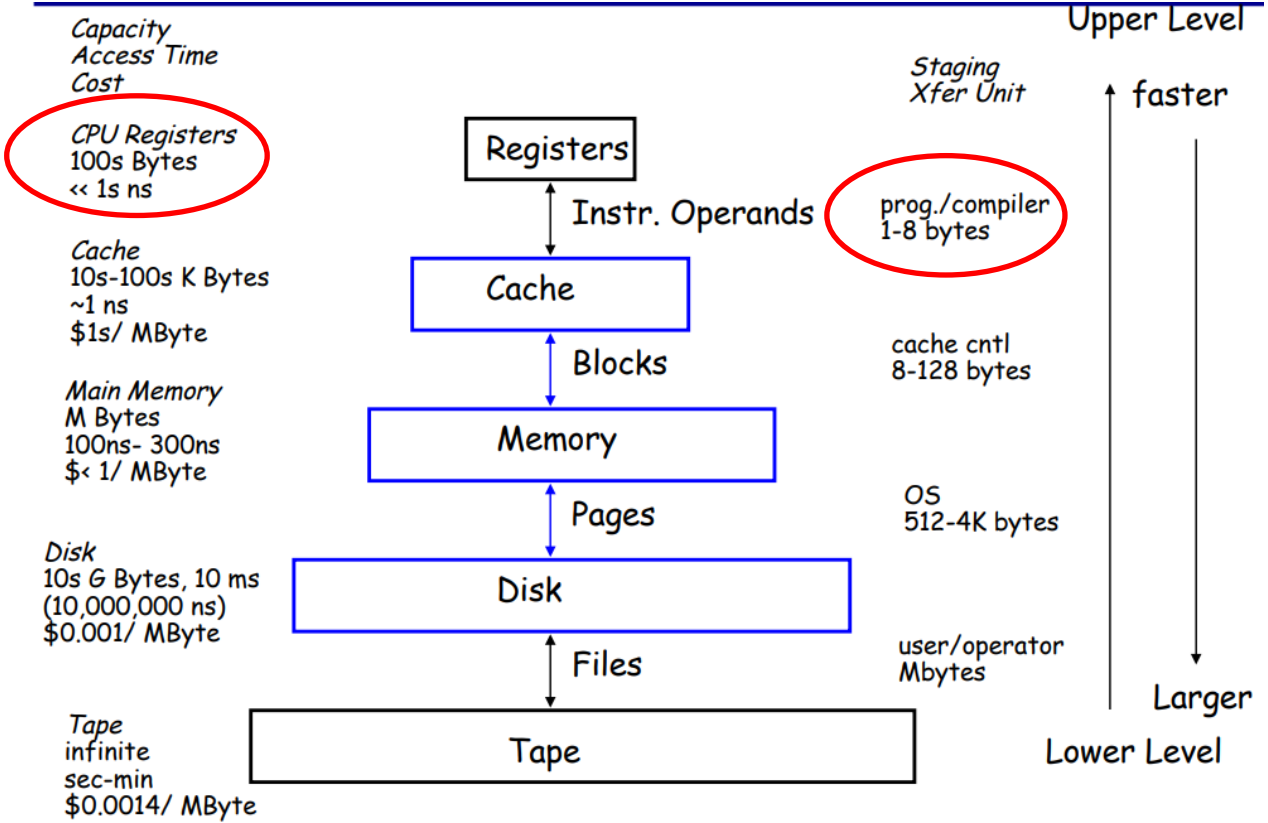
## Thread-Level Parallelism

- Each thread executes a different instruction stream

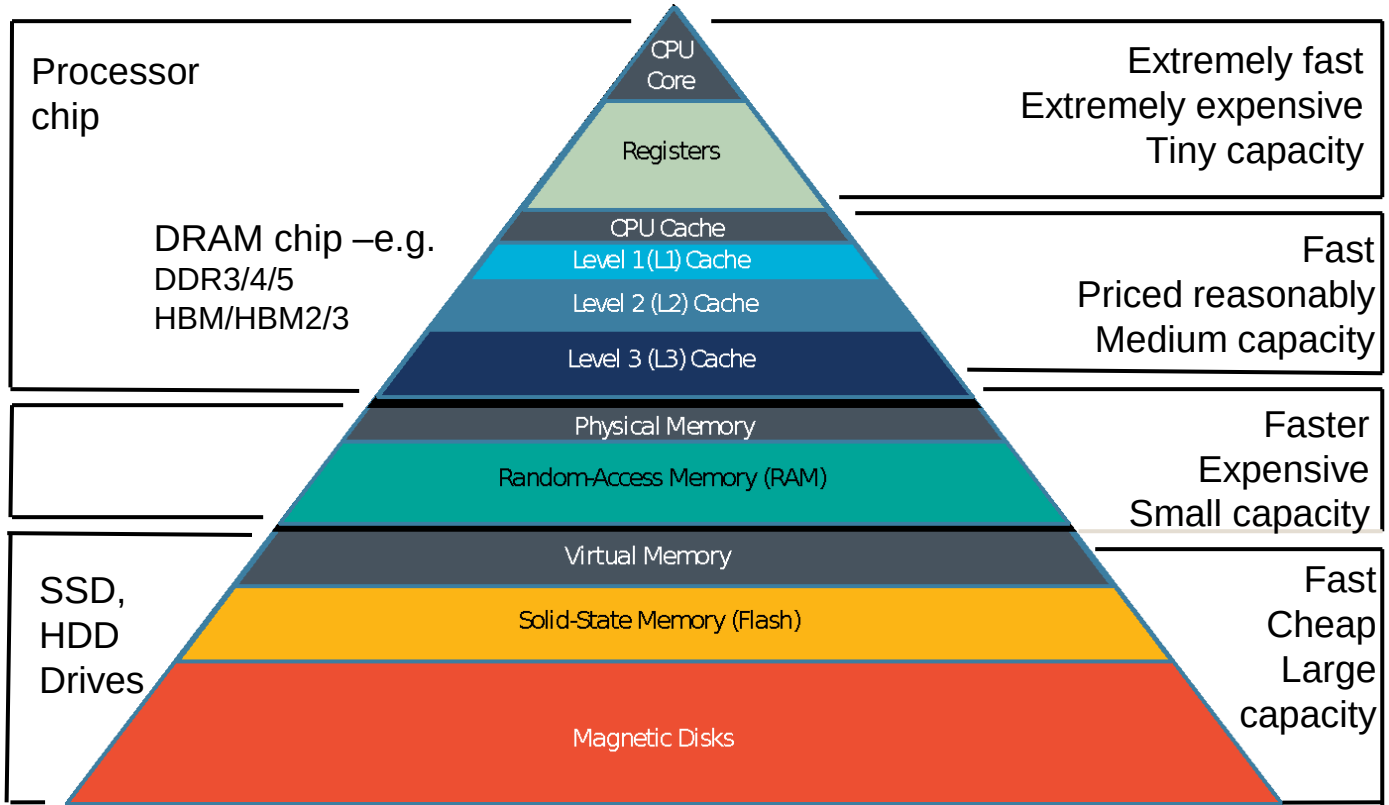
# Great Ideas: “Parallelism”



# Great Ideas: “Hierarchy of Memories”



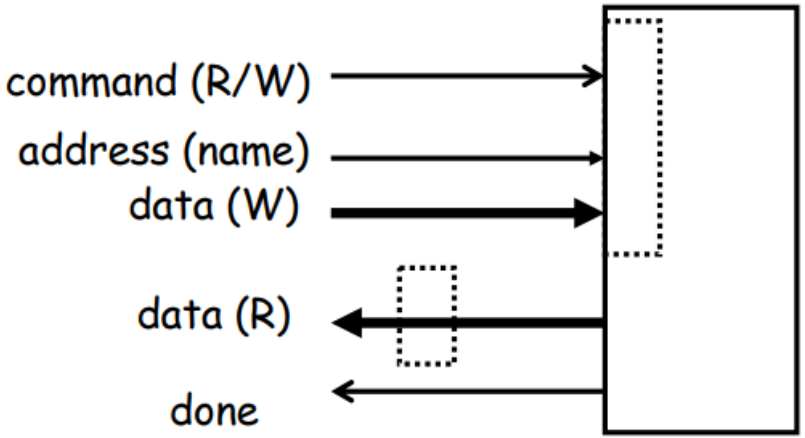
# Great Ideas: “Hierarchy of Memories”



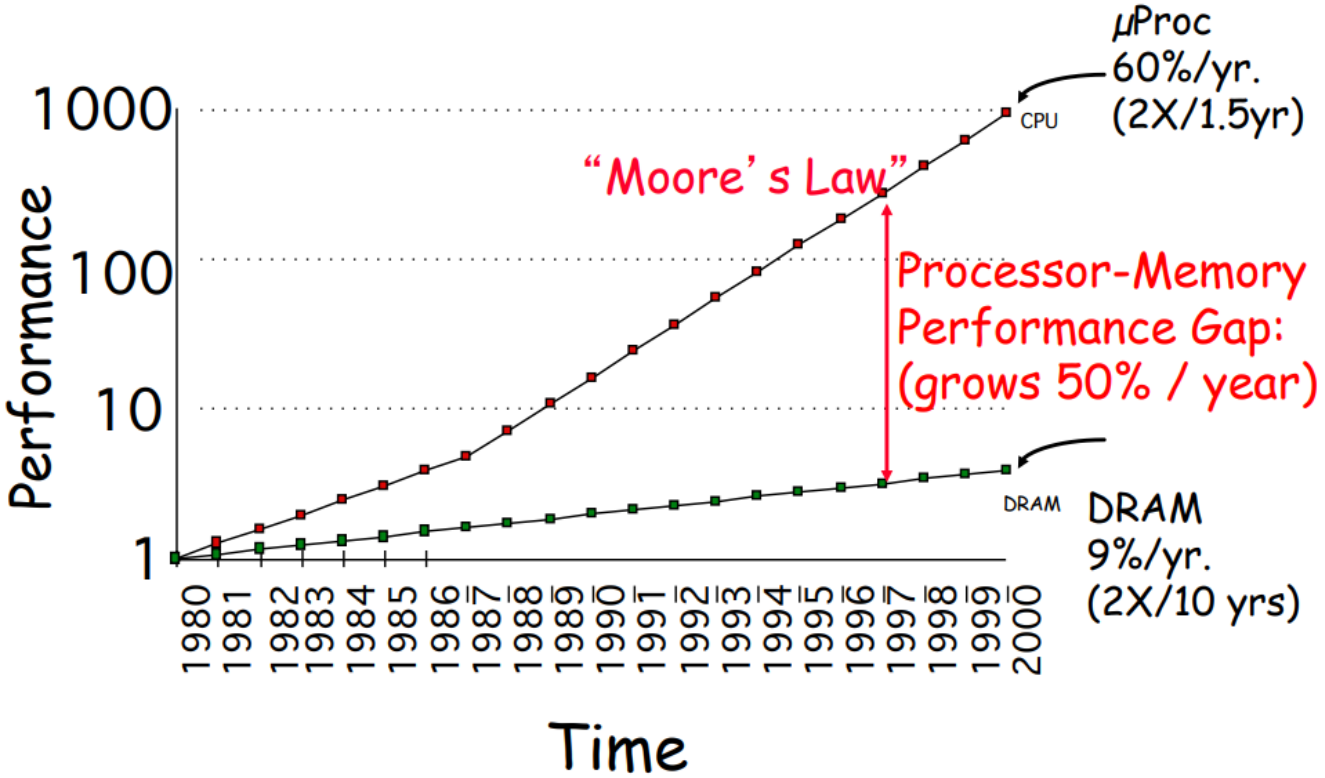
# The Memory Abstraction

- Association of <name, value> pairs
  - Name as byte addresses
  - Values aligned on multiples of size
- Sequence of Reads and Writes
- Write binds a value to an address
  - Left value
- Read address returns most recently written value bound to that address
  - Right value

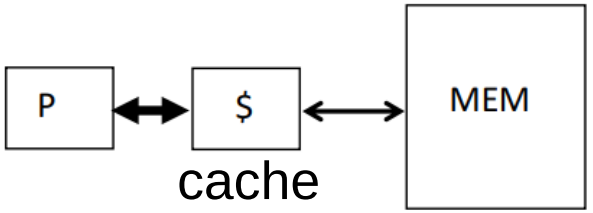
**int a = b;**



# Processor-DRAM Memory Gap (latency)



# The Principle of Data Locality




- The principle of locality
  - Program access a relatively small portion of the address space at any instance of time
- Two different types of locality
  - **Temporal locality** (locality in time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial locality** (locality in space): If an item is referenced, close-by items tend to be referenced soon (e.g., array access)
- HW often relies on locality for speed

# Great Ideas: “Dependability via Redundancy”

- The insecure problem in computer organization
  - Unintended electron flow from cosmic rays will cause unintended transistor behavior
- Redundant Arrays of Independent Disks (RAID)
  - Redundant disks that can lose 1 disk but not lose data
- Error Correcting Code (ECC) Memory
  - Redundant memory bits that can be tolerant of 1 bit of data lost

# Takeaway Questions

- What kinds of components are within a CPU?
  - (A) Functional unit
  - (B) Control unit
  - (C) Memory/storage unit
- Why do we need the abstraction of instruction sets?
  - (A) Improve the performance of the CPU
  - (B) Provides convenient functionality to higher levels
  - (C) Lower the power consumption of the CPU

# Takeaway Questions

- What is the purpose of the memory hierarchy?
  - (A) Save the cost of a computer
  - (B) Shorten the memory access latency by using data locality
  - (C) Raise the storage capacity of a computer

# Understanding Performance

- Algorithm
  - Determines the number of operations executed
- Programming language, compiler, architecture
  - Determine the number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

# Trends in Technology

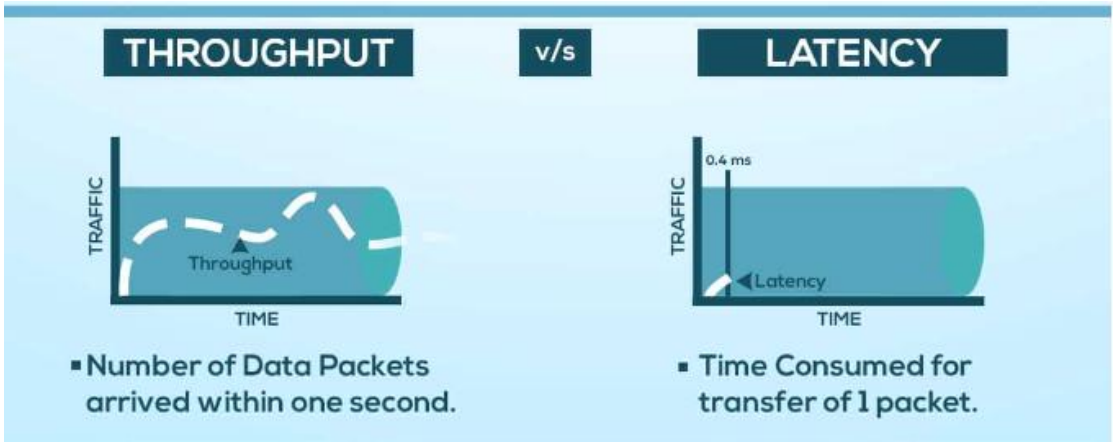
- Integrated circuit technology (Moore's Law)
  - Transistor density: 35% per year
  - Die size: 10-20% per year
  - Integration overall: 40-50% per year
- DRAM capacity
  - 25-40% per year
- Flash memory capacity
  - 50-60% per year, 8-10X cheaper/bit than DRAM
- Magnetic disk capacity
  - 8-10X cheaper/bit than Flash, 200-300X cheaper/bit than DRAM

# Measuring Performance

- Typical performance metrics
  - Response time
  - Throughput
- Speedup of X relative to Y
  - Execution time of Y / Execution time of X
  - E.g. time taken to run a program, 10s on X, 15s on Y
  - Speedup: 15s/10s = 1.5 -> X is 1.5 times faster than Y

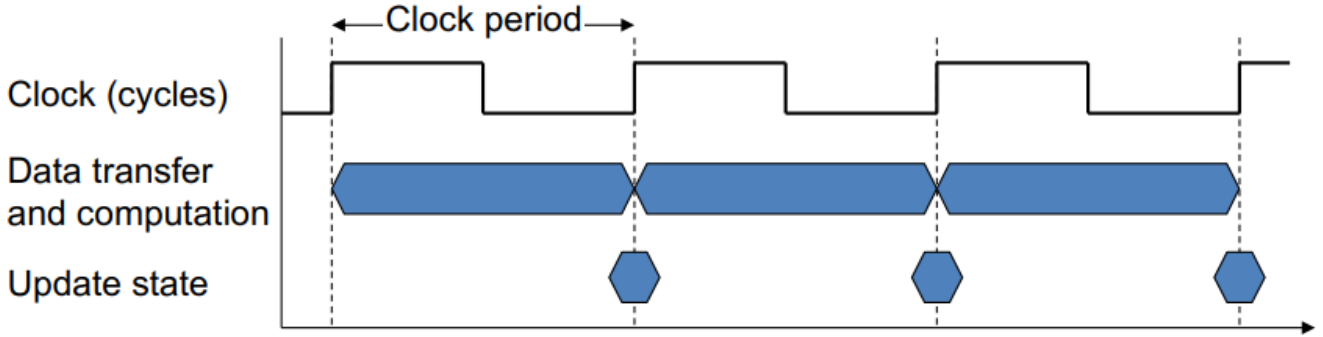
# Response Time and Throughput

- Bandwidth or throughput
  - Total work done in a given time
  - E.g. GFLOPs
- Latency or response time
  - Time between the start and completion of an event



# CPU Clocking

- Digital hardware operations governed by a constant-rate clock

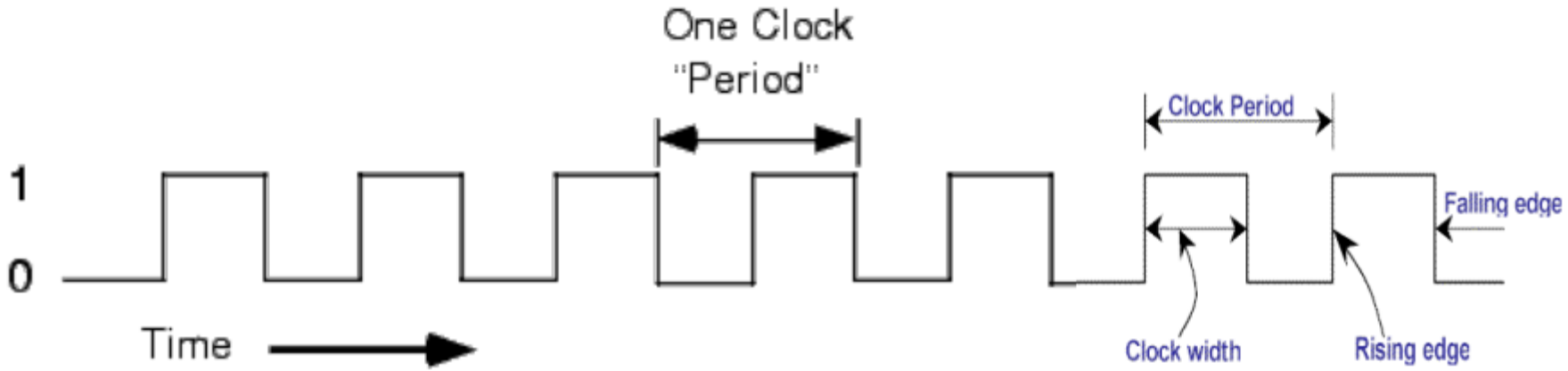



  - Clock period: duration of a clock cycle
    - E.g., 250 ps = 0.25 ns = 250 x $10^{-12}$ s
  - Clock frequency (rate): cycles per second
    - E.g., 4.0 GHz = 4000 MHz = 4.0 x $10^{9}$ Hz
    - Clock period: 1 / (4.0 x $10^{9}$)s = 0.25 ns

# CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing the number of clock cycles
  - Increasing clock rate (frequency)



# CPU Time Example

- Computer A: 2GHz clock rate, 10s CPU time
- Designing Computer B
  - Aim to reduce the CPU time from 10s to 6s, but will cause 1.2X more clock cycle of A (how?)
  - How fast must the computer B clock rate be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

# Instruction Count and CPI

- Instruction count for a program
  - Determined by program, ISA, and compiler
- Average cycles per instruction
  - Determine by the CPU hardware; difference when changing ISAs

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

# CPI Example

- Computer A: Cycle Time = 250 ps, CPI = 2.0
- Computer B: Cycle Time = 500 ps, CPI = 1.2
- Computer A and B use the same ISA
- Which is faster, and by how much ?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster...

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

...by this much

# CPI in More Detail

- The number of cycles varies across different kinds of instructions

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, and C

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| | | | |
| IC in sequence #1 | 2 | 1 | 2 |
| IC in sequence #2 | 4 | 1 | 1 |

- Sequence #1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence #2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

# Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affect IC, possibly CPI
  - Programming language: affect IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affect IC, CPI, $T_c$

# Performance Summary

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

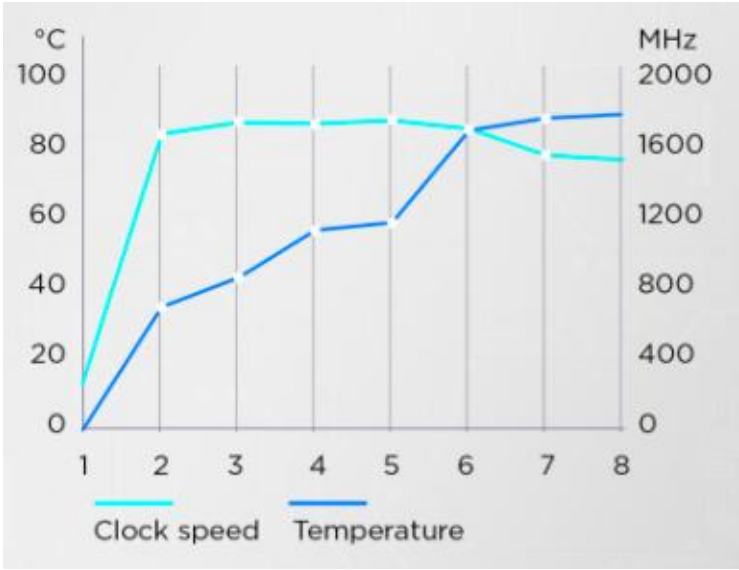
| | Inst Count | CPI | Clock Rate |
|---|---|---|---|
| Program | X | | |
| Compiler | X | (X) | |
| Inst. Set. | X | X | |
| Architecture | X | | X |
| Technology | | | X |

# Power Wall

- Power: energy per unit of time
  - 1 watt = 1 joule per second
  - Energy = joule
- Thermal Design Power (TDP)
  - in watts
  - Refers to CPU/GPU power consumption & the amount of heat produced
  - Impact the processor speed

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000



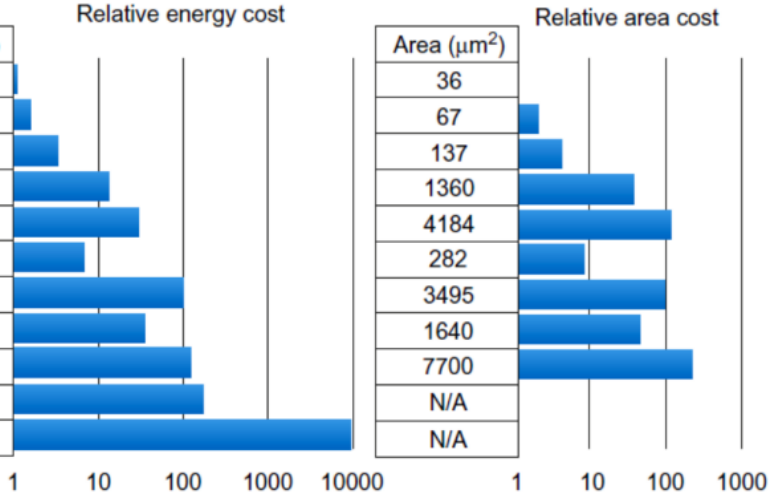

https://www.cgdirector.com/cpu-tdp-thermal-design-power-explained/

# Static Power

- Power includes both dynamic power and static power
- Static power consumption
  - 25-50% of total power
  - Scales with number of transistors
  - Using power gating (turn off power of inactive modules) to reduce static power

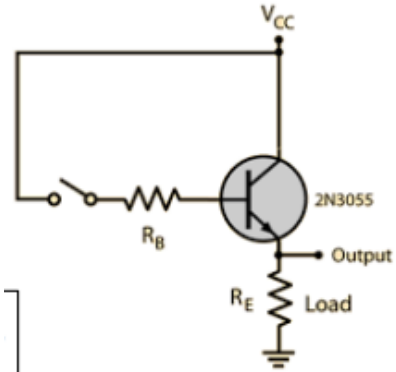
$$\text{Power}_{\text{static}} \propto \text{Current}_{\text{static}} \times \text{Voltage}$$

| Operation: | Energy (pJ) | Area ($\mu m^2$) |
|---|---|---|
| 8b Add | 0.03 | 36 |
| 16b Add | 0.05 | 67 |
| 32b Add | 0.1 | 137 |
| 16b FB Add | 0.4 | 1360 |
| 32b FB Add | 0.9 | 4184 |
| 8b Mult | 0.2 | 282 |
| 32b Mult | 3.1 | 3495 |
| 16b FB Mult | 1.1 | 1640 |
| 32b FB Mult | 3.7 | 7700 |
| 32b SRAM Read (8KB) | 5 | N/A |
| 32b DRAM Read | 640 | N/A |




[Horowitz, ISSCC 2014, 65nm]

# Dynamic Power and Energy




- Dynamic energy
  - Transistor switch from 0->1 or 1->0
  - The capacitive load
    - Include energy stored in materials and devices
    - Causes changes in voltage to lag behind changes in current

$$\text{Energy}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2$$

- Dynamic power
  - Reducing clock rate reduces power, not energy

$$\text{Power}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

# Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We cannot reduce voltage further
  - We cannot remove more heat
  - How else can we improve performance?

# Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# SPEC CPU Benchmark

- Programs used to measure performance
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of program
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

# Geometric Mean Performance $\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$

- Advantage
  - Provides a realistic “average” of compounding growth
- When to use
  - Ideal for multiplicative, compounding, or ratio-based data
  - Ex. Benchmark improvements
- Example: A 10% gain (1.1) followed by a 10% loss (0.9) over two periods
  - Arithmetic mean suggests 0% change
  - Geometric mean sqrt(1.1 x 0.9) = 0.995, correctly shows a 0.5% loss (compounding effect)

# Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5x overall?

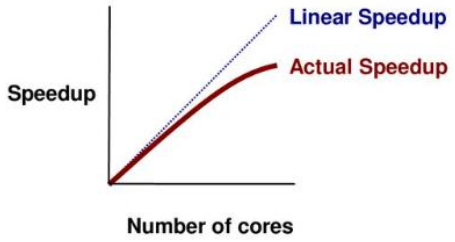
$$20 = \frac{80}{n} + 20$$

Can't be done !

# Amdahl's Law

- The time to use Amdahl's law
  - Only can use Amdahl's law to estimate performance improvement when we know the time consumed for some function and its potential speedup
- Speedup is defined as
  - The execution time on a single core ($T_1$) over the execution time on **p** cores ($T_p$) (Amdahl, 1967)
  - Linear or ideal speedup is reached when $S_p = p$

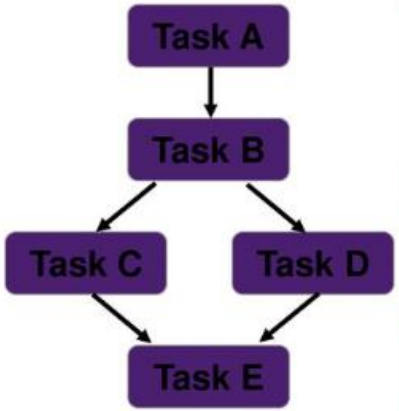
$$S_p = \frac{T_1}{T_p}$$

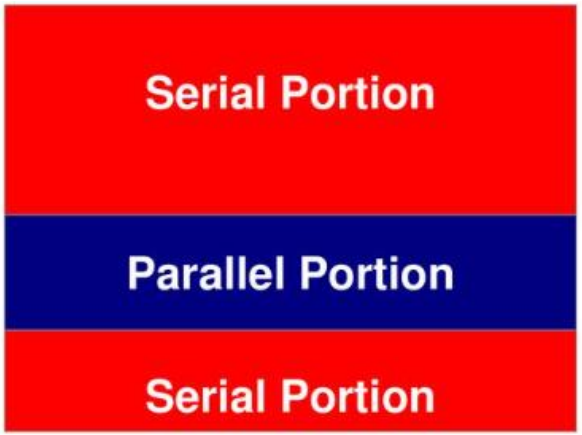


# Amdahl's Law: Theoretical Speedup

- Assume P is the parallel portion of a parallel program, then (1-P) is the serial portion
- Amdahl's law states that the maximum speedup on N processors is:

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$



# Amdahl's Law: Examples

- As N tends to infinity, S(N) tends to be 1 / (1-P)

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}}$$

| Parallel Portion | Maximum Speedup* |
|---|---|
| 99% | 100 |
| 95% | 20 |
| 90% | 10 |
| 75% | 4 |
| 50% | 2 |
| 25% | 1.3 |

# Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[ \left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

- Best we could hope to do

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$

# Amdahl's Law: Examples

Overall speedup if we make 90% of a program run 10 times faster.

$F = 0.9 \quad S = 10$

$$\text{Overall Speedup} = \frac{1}{(1-0.9)+\frac{0.9}{10}} = \frac{1}{0.1+0.09} = 5.26$$

Overall speedup if we make 80% of a program run 20% faster.

$F = 0.8 \quad S = 1.2$

$$\text{Overall Speedup} = \frac{1}{(1-0.8)+\frac{0.8}{1.2}} = \frac{1}{0.2+0.66} = 1.153$$

# Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Difference in complexity between instructions
  - CPI varies between programs on a given CPU

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

# Takeaway Questions

- What kind of components will affect the instruction count?
  - (A) The implementation of a program
  - (B) The compiler
  - (C) The semiconductor processing technology
- A program spends 10% of its time on the serial codes. What is the speedup when running this program on a 100-core CPU using Amdahl's Law?
  - (A) 9.17 X
  - (B) 100 X
  - (C) 5.26 X