

# **ECE552 / CPS550**

## **Advanced Computer Architecture I**

### **Lecture 1**

### **Introduction**

Benjamin Lee  
Electrical and Computer Engineering  
Duke University

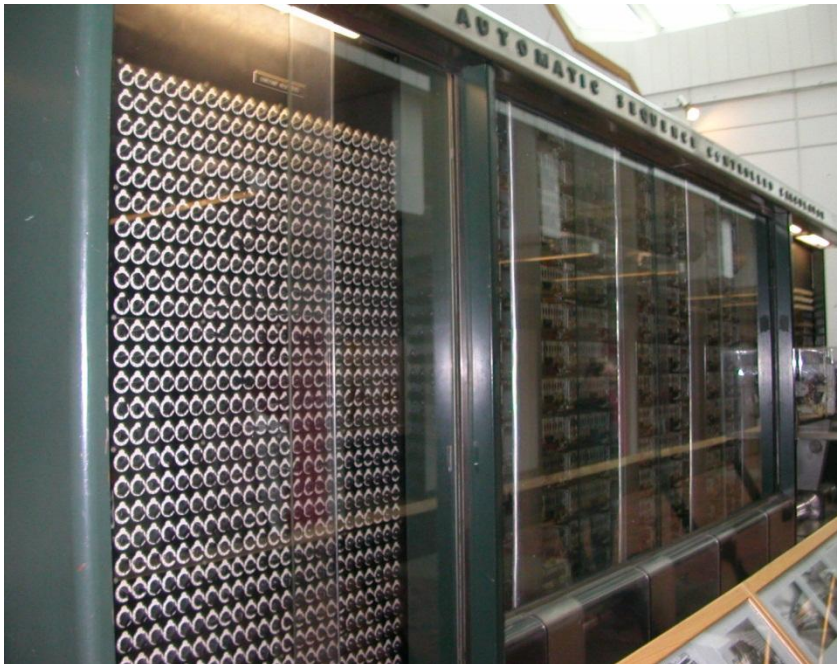
[www.duke.edu/~bcl15](http://www.duke.edu/~bcl15)  
[www.duke.edu/~bcl15/class/class\\_ece552fall12.html](http://www.duke.edu/~bcl15/class/class_ece552fall12.html)



# Computing Devices (Then)

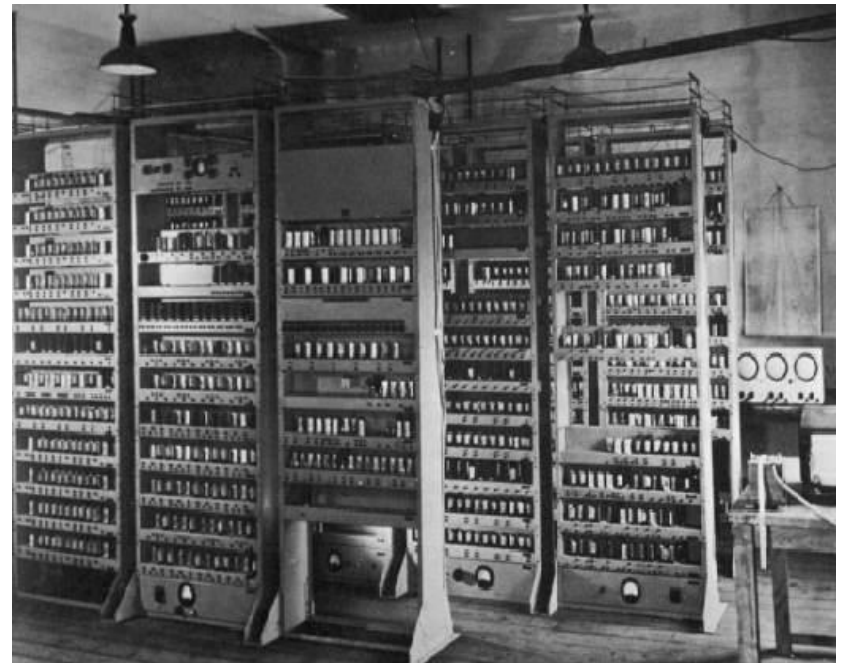
Mark I

Harvard University, 1944



EDSAC

University of Cambridge, 1949





# Computing Devices (Now)

iPad

Apple/ARM, 2010



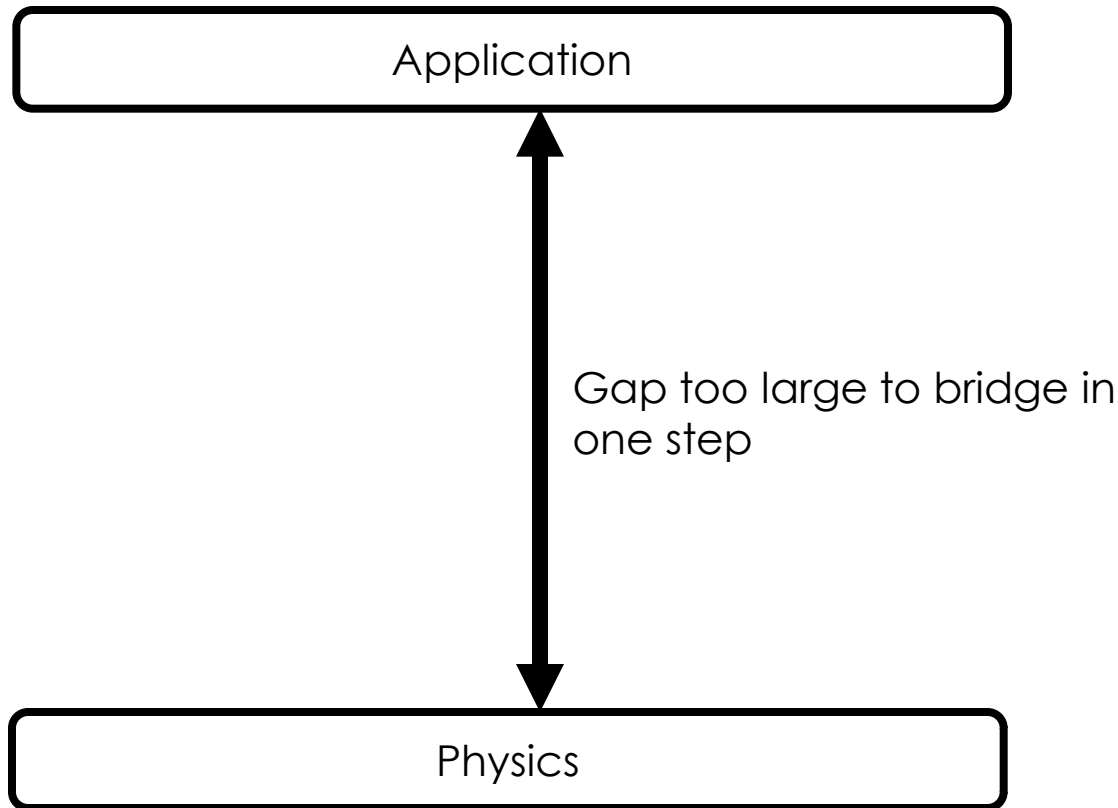
Blue Gene/P

IBM, 2007





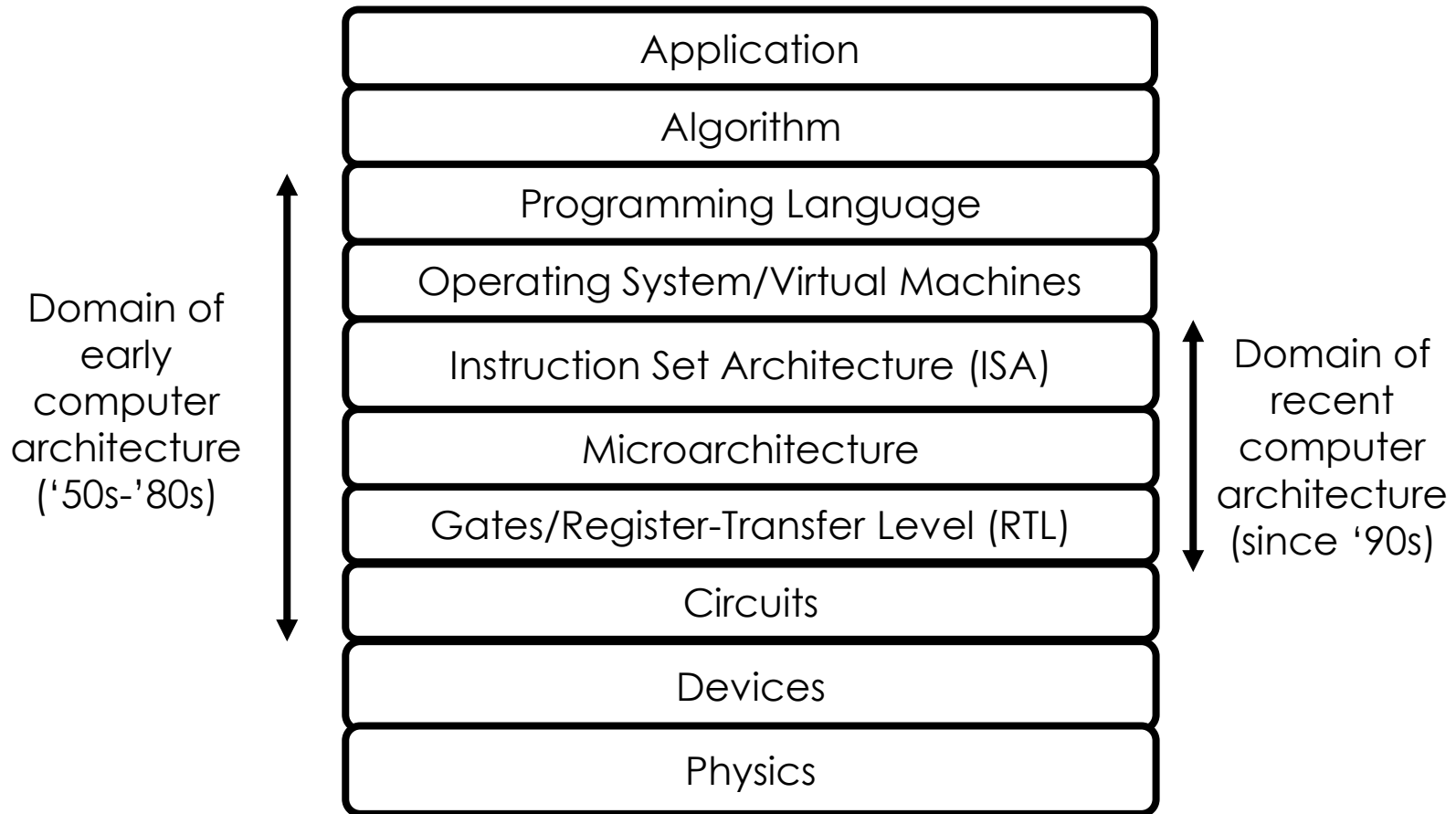
# Computer Architecture



Computer architecture is the design of abstraction layers, which allow efficient implementations of computational applications on available technologies



# Abstraction Layers





# An Integrated Approach

## Architect Systems

- Coordinate technology, hardware, run-time software, compilers, apps
- Responsible for end-to-end functionality

## Design and Analyze

- Search the space of possible designs at all levels in computer system
- Evaluate designs with quantitative metrics (performance, power, cost)

## Navigate Computing Landscape

- Architects work at the hardware-software interface
- Technologies are emerging
- Applications are demanding
- Systems are scaling

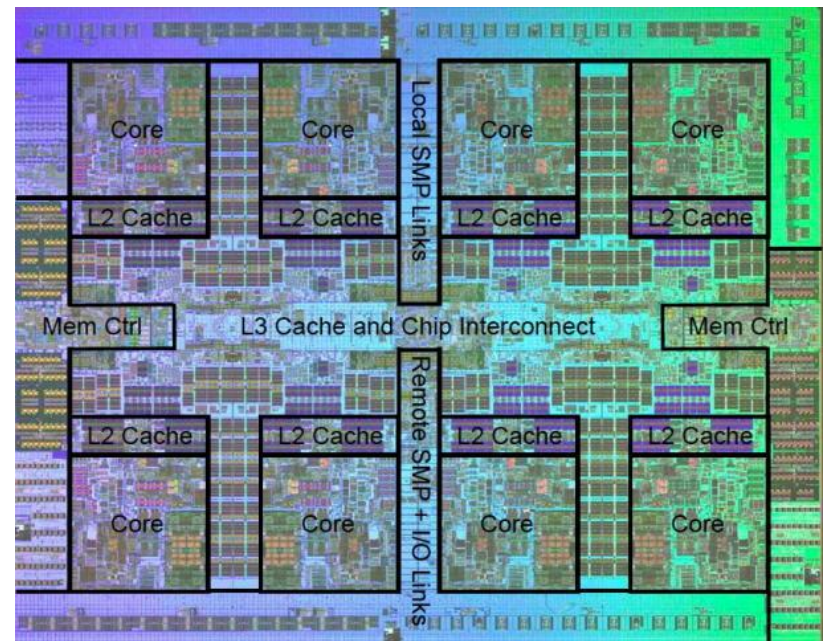




# ECE 552 Executive Summary

In-order Datapath  
(built, ECE152)

Chip Multiprocessors  
(understand, experiment ECE552)





# ECE 552 Administrivia

Instructor	Prof. Benjamin Lee benjamin.c.lee@duke.edu Office Hours: Tu 4-5pm, Fr 4-5pm, 210 Hudson
Teaching Assistants	Marisabel Guevara, mg@cs.duke.edu Office Hours: Tu 12:25-1:25pm, W 4-5pm, TBD  Weidan Wu, ww53@duke.edu Office Hours: M 4-5pm, Th 2:40-3:40pm, TBD
Lectures	Tu/Th 1:25-2:40PM, 208 Hudson
Text	Computer Architecture: A Quantitative Approach, 5 <sup>th</sup> Edition (2012). Do not use earlier editions
Web	<a href="http://www.duke.edu/~BCL15/class/class_ece552fall12.html">http://www.duke.edu/~BCL15/class/class_ece552fall12.html</a>





# ECE 552 Prerequisites

## Participation

- Electrical and Computer Engineering, Computer Science
- PhD, MS, Undergraduates

## Prerequisites

- Introduction to computer architecture (CPS 104, ECE 152, or equiv.)
- Programming (homework/projects in C, C++)

## Background Knowledge

- Instruction sets, computer arithmetic, assembly programming
- D.A. Patterson and J.L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*, 5<sup>th</sup> Edition.

## Dropping the Course

- if you are going to drop, please do so early



# ECE 552 Syllabus

## 1. Design Metrics

1. **Performance**
2. **Power**
3. **Early machines**

## 2. Simple Pipelining

- 1 **Multi-cycle machines**
- 2 **Branch Prediction**
- 3 **In-order Superscalar**
- 4 **Optimizations**

## 3. Complex Pipelining

- 1 **Score-boarding, Tomasulo Algorithm**
- 2 **Out-of-order Superscalar**

Midterm Exam

Fall Break

## 4. Memory Systems

- 1 **Caches**
- 2 **DRAM**
- 3 **Virtual Memory**

## 5. Explicitly Parallel Architectures

- 1 **VLIW**
- 2 **Vector machines**
- 3 **Multi-threading**

## 6. Multiprocessors

- 1 **Memory Models**
- 2 **Coherence Protocols**

## 7. Advanced Topics

- 1 **Emerging Technologies**
- 2 **Specialized Architectures**
- 3 **Datacenter Architectures**



# ECE 552 Components

## 30% Homework and Readings

- Homework done in teams of 3
- 5 classes dedicated to paper discussions

## 15% Midterm exam

- 75 minutes (in class), closed book

## 25% Final exam

- 3 hours, closed-book
- based on lectures, problem sets, readings

## 30% Term project/paper

- Project done in teams of 3

## Academic Policy

University policy as codified by Duke Undergraduate Honor Code will be strictly enforced. Zero tolerance for cheating and/or plagiarism.



# ECE 552 Academic Policy

University policy as codified by the Duke Undergraduate Honor Code will be strictly enforced. Zero tolerance for cheating and/or plagiarism.

If a student is suspect of academic dishonesty (e.g., cheating on an exam, copying a lab report, collaborating inappropriately on an assignment), faculty are required to report the matter to the Office of Student Conduct.

A student found responsible for academic dishonesty faces formal disciplinary action, which may include suspension. A student suspended twice for academic dishonesty automatically faces a minimum 5-year separation from Duke University.



# ECE 552 Term Project

## Scope

- Semester-long research project
- Teams of 3
- Students propose project ideas (Oct 14)

## Final Paper

- 6-12 page research paper
- Evaluate research idea quantitatively
- Survey and cite related work



# ECE 552 Upcoming Deadlines

## 11 September – Homework #1 Due

Assignment on web page. Teams of 2-3.

Submit hard copy in class. Email code to TA's

## 11 September – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Hill et al. "Classic machines: Technology, implementation, and economics"
2. Moore. "Cramming more components onto integrated circuits"
3. Radin. "The 801 minicomputer"
4. Patterson et al. "The case for the reduced instruction set computer"
5. Colwell et al. "Instruction sets and beyond: Computers, complexity, controversy"



# Performance

## Definitions

- Latency: time to finish given task (a.k.a. execution time)
- Throughput: number of tasks in given time (a.k.a. bandwidth)
- Throughput can exploit parallelism while latency cannot

## Example: Move people from Duke to UNC, 10 miles

- Car: capacity = 5, speed = 60 miles/hour
- Bus: capacity = 60, speed = 20 miles/hour
  
- Latency(car) = (10 miles @ 60 miles/hour) = 10 minutes
- Latency(bus) = (10 miles @ 20 miles/hour) = 30 minutes
  
- Throughput(car) = (3 trips @ 60 miles per hour) = 15 people/hour
- Throughput(bus) = (1 trip @ 20 miles per hour) = 60 people/hour





# Benchmarking

## Measuring Performance

- Target Workload: accurate but not portable
- Representative Benchmark: portable but not accurate
- Microbenchmark: small, fast code sequences but incomplete

## Representative Benchmarks

- SPEC (Standard Performance Evaluation Corporation, [www.spec.org](http://www.spec.org))
- Collects, standardizes, distributes benchmark programs
- Parallel Benchmarks
- Scientific and commercial computing
- SPLASH-2, NAS, SPEC OpenMP, SPECjbb
- Transaction Processing Council (TPC)
- Online transaction processing (OLTP) with heavy I/O, memory
- TPC-C, TPC-H, TPC-W



# Aggregating Performance

## Addition

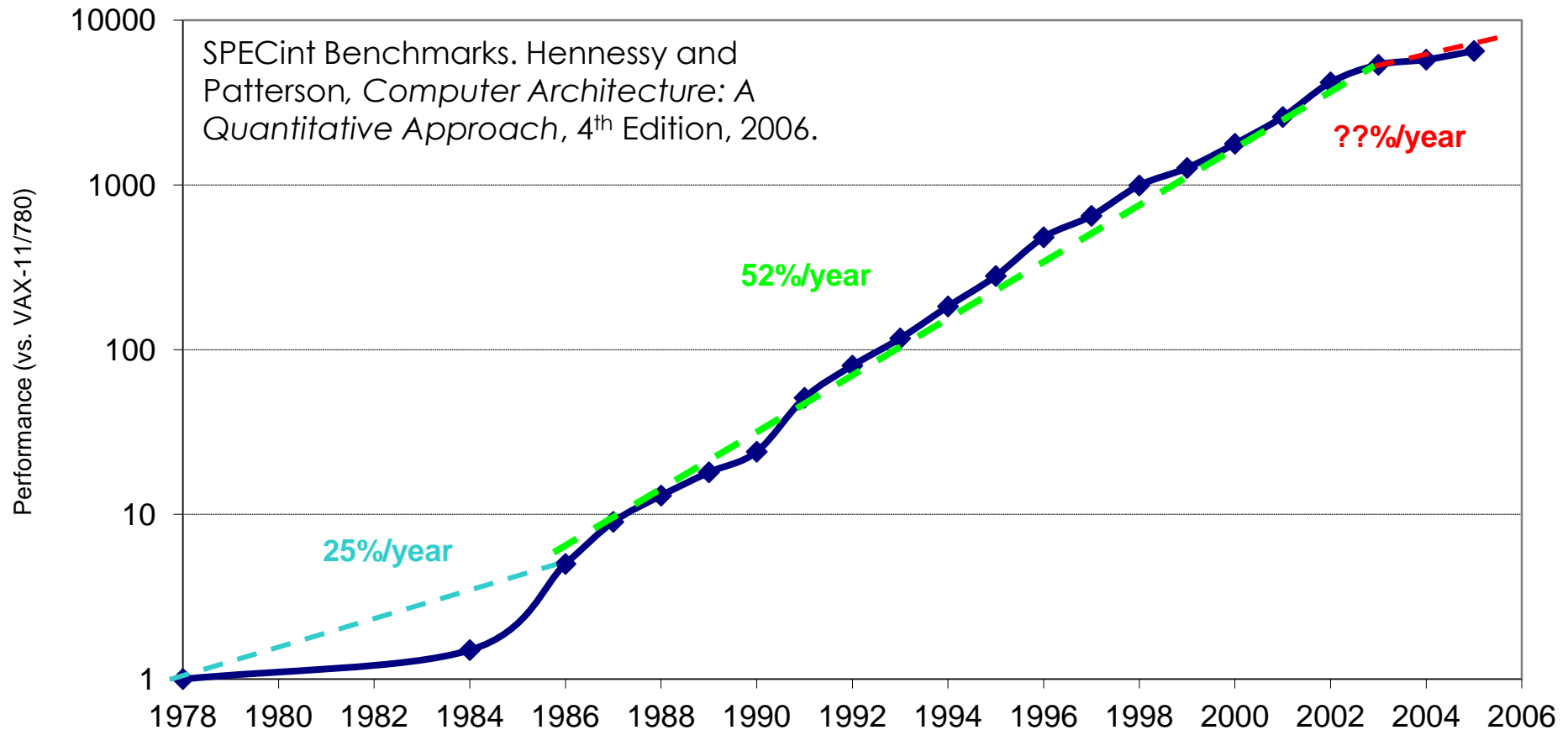
- Latency is additive but throughput is not
- Example: Consider applications A1 and A2 on processor P
- $\text{Latency}(A1, A2) = \text{Latency}(A1) + \text{Latency}(A2)$
- $\text{Throughput}(A1, A2) = 1 / [1/\text{Throughput}(A1) + 1/\text{Throughput}(A2)]$

## Averages

- Arithmetic Mean:  $(1/N) * \sum_{P=1..N} \text{Latency}(P)$
- For measures that are proportional to time (e.g., latency)
- Harmonic Mean:  $N / \sum_{P=1..N} 1/\text{Throughput}(P)$
- For measures that are inversely proportional to time (e.g., throughput)
- Geometric Mean:  $(\prod_{P=1..N} \text{Speedup}(P))^{(1/N)}$
- For ratios (e.g., speed-ups)



# Processor Performance





# Performance Factors

$$\text{Latency} = (\text{Instructions} / \text{Program}) \times (\text{Cycles} / \text{Instruction}) \times (\text{Seconds} / \text{Cycle})$$

## Seconds / Cycle

- Technology and architecture
- Transistor scaling
- Processor microarchitecture

## Cycles / Instruction (CPI)

- Architecture and systems
- Processor microarchitecture
- System balance (processor, memory, network, storage)

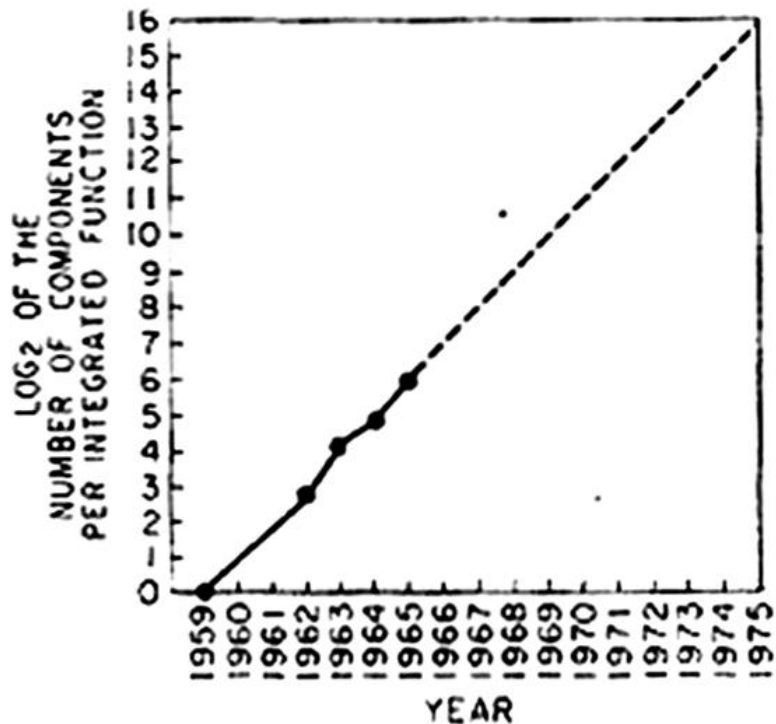
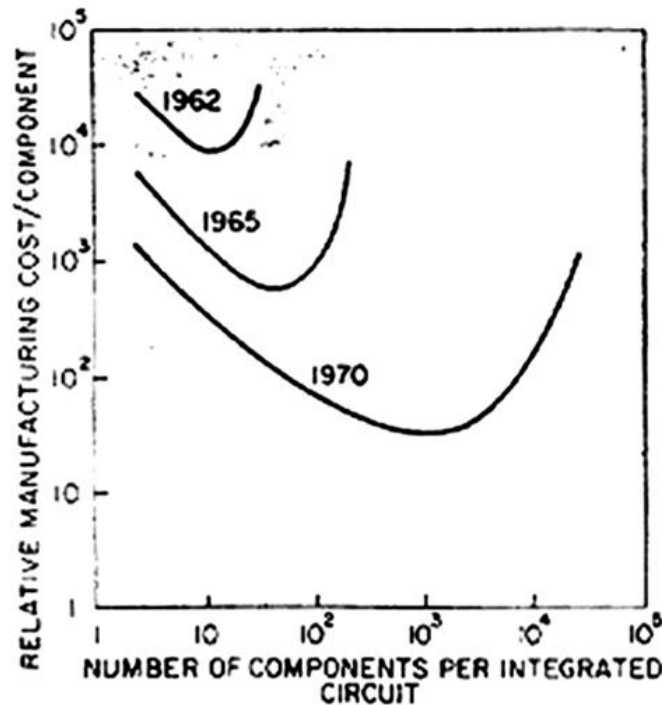
## Instructions / Program

- Algorithm and applications
- Compiler transformations, optimizations
- Instruction set architecture



# Moore's Law

- Moore. "Cramming more components onto integrated circuits." Electronics, Vol 38, No. 8, 1965.
- As integration increases and packaging cost decrease
- How does Moore's Law impact performance?

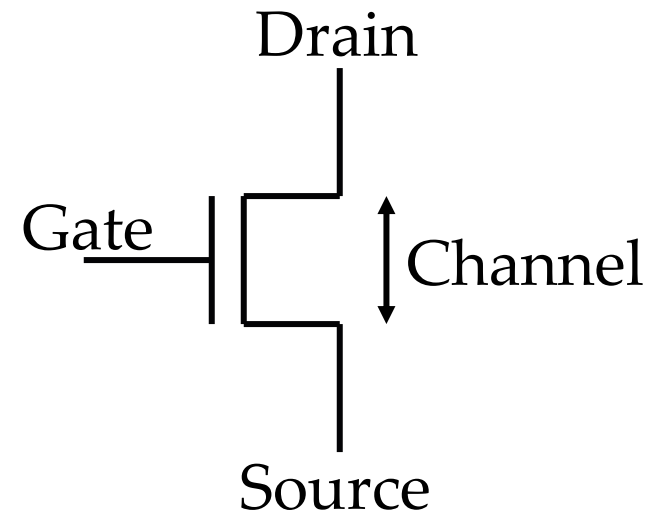
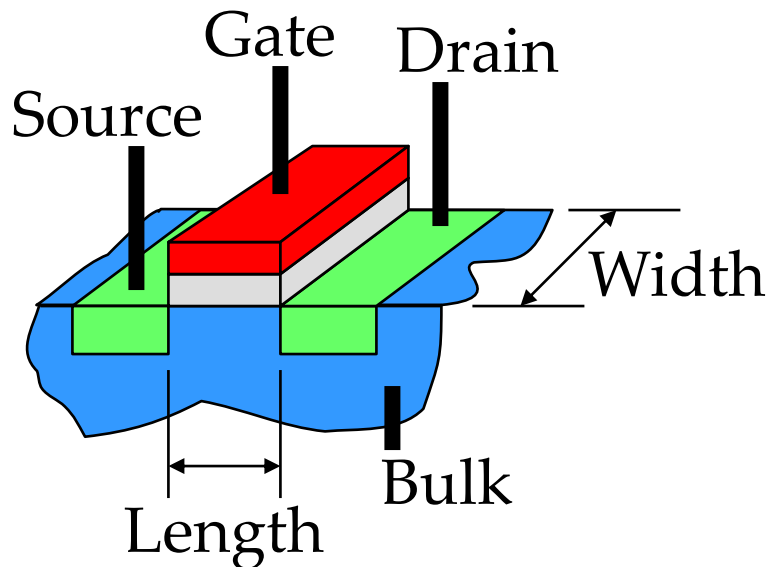




# Field-Effect Transistors

## MOSFET

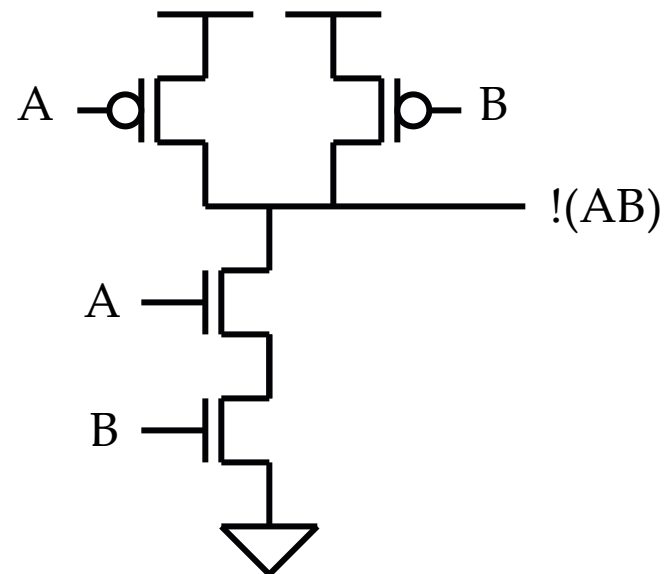
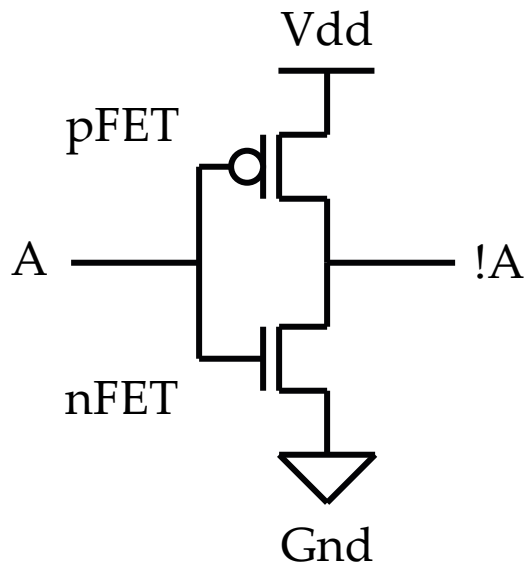
- MOS: metal-oxide semiconductor
- FET: field-effect transistor
- Charge carriers flow between source-drain
- Flow controlled by gate voltage
- Abstract MOSFET as electrical switch





# Complementary MOS (CMOS)

- Voltages map to logical values ( $V_{dd}=1$ ,  $Gnd=0$ )
- Implement complementary Boolean logic
- nFET: conduct charge when  $V_g = V_{dd}$ , used in pull-down network
- pFET: conduct charge when  $V_g = Gnd$ , used in pull-up network
- Examples: Inverter, NAND (universal, any logic function via De Morgan's Law)

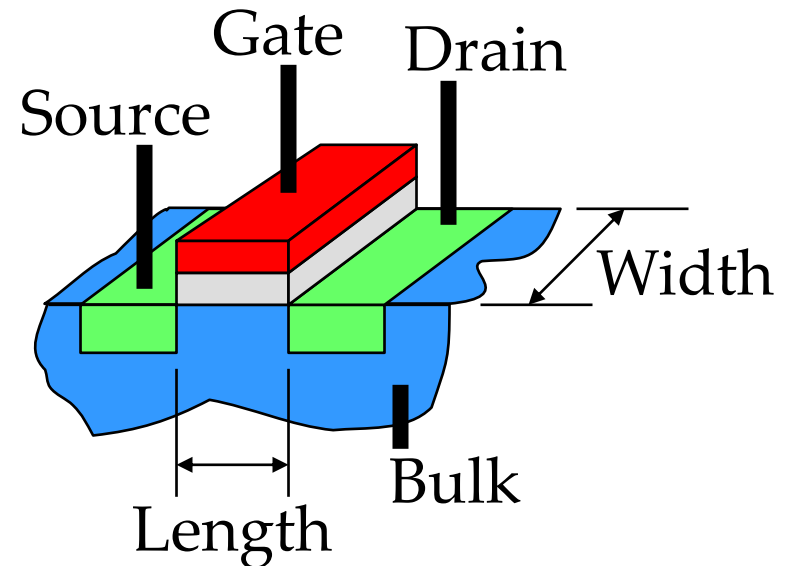
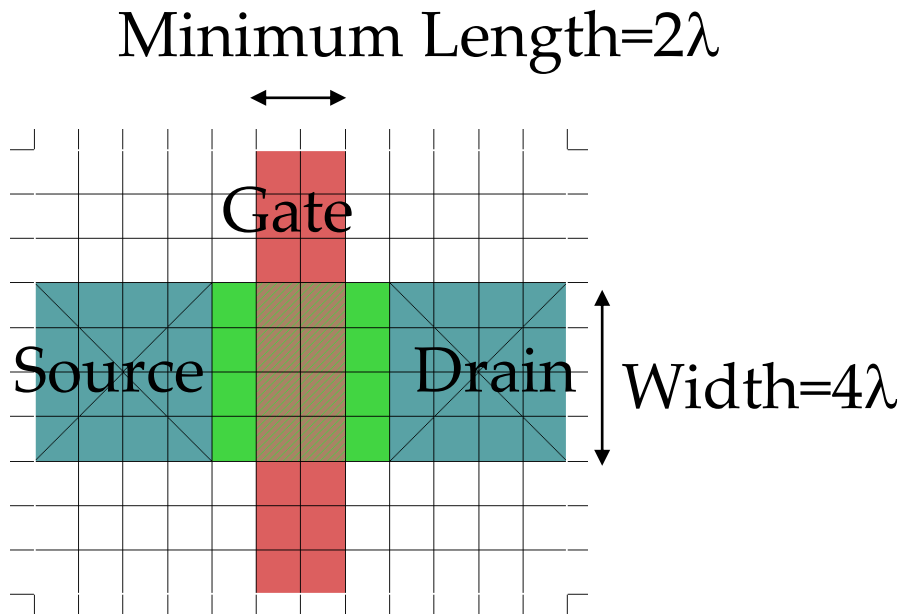






# Transistor Dimensions

- Process defined by feature size ( $F$ ), layout design ( $\lambda = F/2$ )
- Example:  $F=2\lambda = 45\text{nm}$  process technology
- Transistor dimensions determine technology performance
- Transistor drive strength (i.e., performance) increases as channel length shrinks





# Dennard Scaling

- Dennard et al. "Design of ion-implanted MOSFETs with very small physical dimensions," Journal Solid State Circuits, 1974.
- Scale not only dimensions but also doping concentration and voltage
- Transistors become faster (1.4x)
- Applied to Moore's Law:  $k=1.4$ ,  $1/k = 0.7$  every 18-24 months

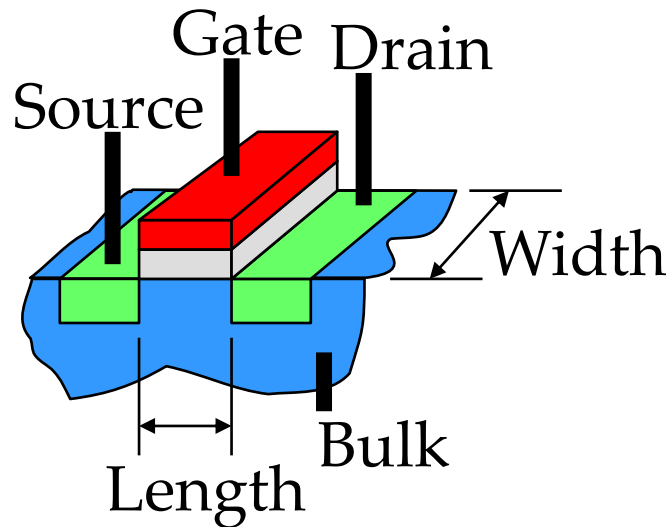


TABLE I  
SCALING RESULTS FOR CIRCUIT PERFORMANCE

Device or Circuit Parameter	Scaling Factor
Device dimension $t_{ox}$ , $L$ , $W$	$1/\kappa$
Doping concentration $N_a$	$\kappa$
Voltage $V$	$1/\kappa$
Current $I$	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit $VC/I$	$1/\kappa$
Power dissipation/circuit $VI$	$1/\kappa^2$
Power density $VI/A$	1



# Dennard Scaling Limits

- Horowitz et al. "Scaling, power, and the future of CMOS." IEDM, 2005.
- Classical Dennard scaling ended at 130nm in 2000-2001.
- Oxide Thickness: How to manage increasing leakage? Use high-K dielectrics
- Channel Length: How to manage increasing leakage? Stop scaling  $L$
- Doping Concentration: How to handle imprecise doping? Manage variability
- Voltage: How to manage increasing leakage? Stop scaling  $V$
- Current: How to increase current with shrinking channels? Stress silicon
- Example: Intel 22nm process technology with FinFET

TABLE I  
SCALING RESULTS FOR CIRCUIT PERFORMANCE

Device or Circuit Parameter	Scaling Factor
Device dimension $t_{ox}$ , $L$ , $W$	$1/\kappa$
Doping concentration $N_a$	$\kappa$
Voltage $V$	$1/\kappa$
Current $I$	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit $VC/I$	$1/\kappa$
Power dissipation/circuit $VI$	$1/\kappa^2$
Power density $VI/A$	1

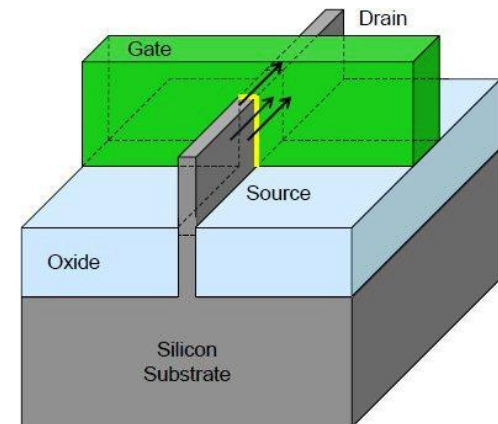
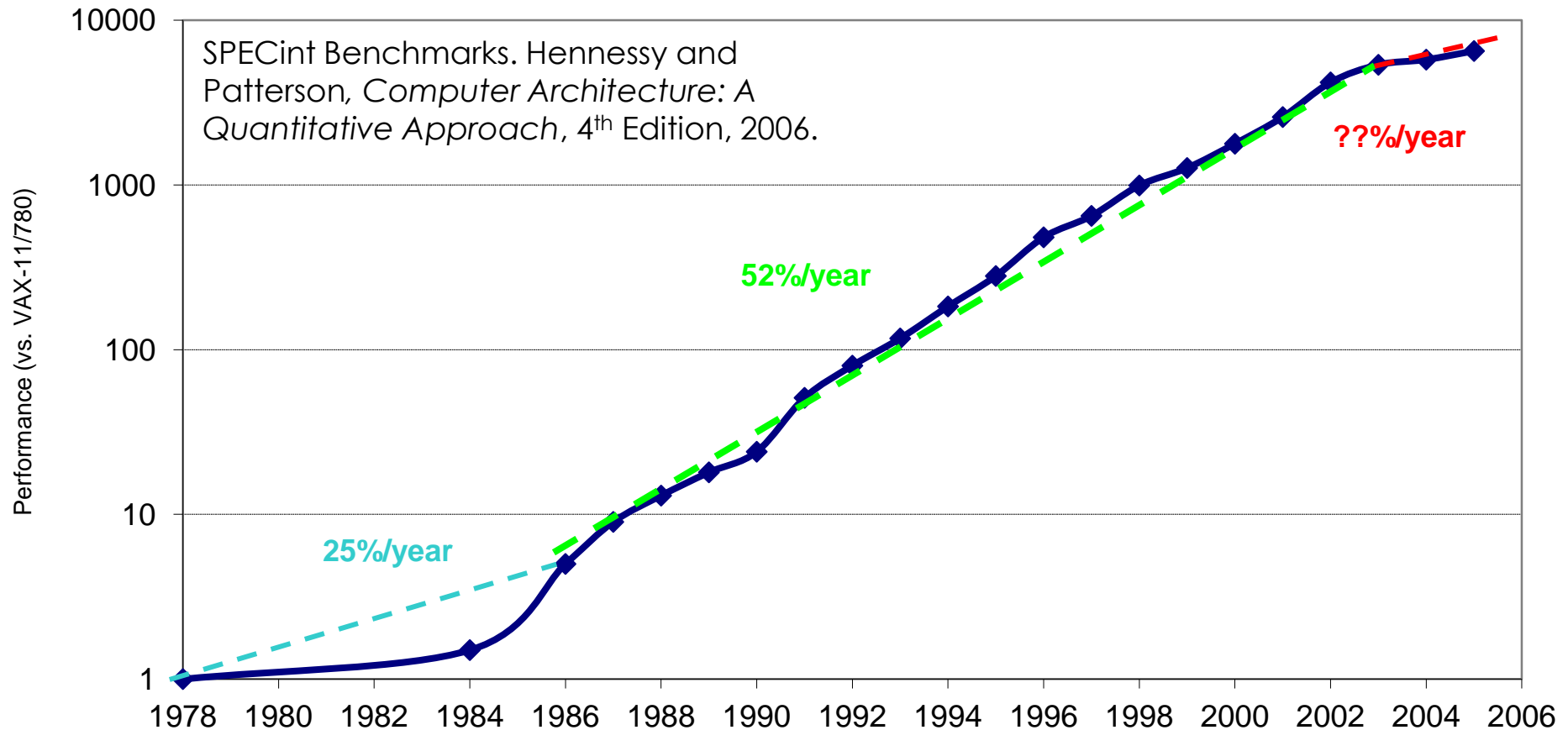


Image: Courtesy Intel Corp.



# Processor Performance





# Performance Factors

$$\text{Latency} = (\text{Instructions} / \text{Program}) \times (\text{Cycles} / \text{Instruction}) \times (\text{Seconds} / \text{Cycle})$$

## Seconds / Cycle

- Technology and architecture
- Transistor scaling
- Processor microarchitecture

## Cycles / Instruction (CPI)

- Architecture and systems
- Processor microarchitecture
- System balance (processor, memory, network, storage)

## Instructions / Program

- Algorithm and applications
- Compiler transformations, optimizations
- Instruction set architecture



# Cycles per Instruction (CPI)

## Average Instruction Latency

- Examine instruction frequency
- Different instructions require different number of cycles
- Example: Integer instructions (1 cy), Floating-point instruction (>10 cy)
- CPI is slightly easier to calculate than IPC (time versus rate)

## Example

- Instruction frequency: 1/3 INT, 1/3 FP, 1/3 MEM operations
- Instruction cycles: 1cy INT, 3cy FP, 2cy MEM
- $CPI = (1/3 \times 1) + (1/3 \times 3) + (1/3 \times 2)$

## Caveat

- CPI provides high-level, quick estimates of performance
- Does not account for details (e.g., instruction dependences)



# CPI and Design

## Baseline Processor / Application

- Integer ALU: 50%, 1 cycle
- Load: 20%, 5 cycle
- Store: 10%, 1 cycle
- Branch: 20%, 2 cycle

## Possible Enhancements

- Option 1: Branch prediction to reduce branch cost to 1 cycle
- Option 2: Bigger data cache to reduce load cost to 3 cycles
- Which enhancement would we prefer?

## Cycles Per Instruction

- Base =  $(0.5 \times 1) + (0.2 \times 5) + (0.1 \times 1) + (0.2 \times 2) = 2$  cycles
- Option 1 =  $(0.5 \times 1) + (0.2 \times 5) + (0.1 \times 1) + (0.2 \times 1) = 1.8$  cycles
- Option 1 =  $(0.5 \times 1) + (0.2 \times 3) + (0.1 \times 1) + (0.2 \times 2) = 1.6$  cycles





# Measuring CPI

## Physical Measurements

- Measure wall clock time as application runs
- Multiply time by clock frequency to get cycles
- Profile application with hardware counters (e.g., Intel VTune)

## Simulated Measurements

- Cycle-level, microarchitectural simulation (e.g., SimpleScalar)
- Run applications on simulated hardware
- Track instructions as they progress through the design



# Pitfall: Partial Performance Metrics

## Ignoring Instructions per Program

- Neglect dynamic instruction count
- Misleading if working in algorithms, compilers, or ISA

## Using Instructions per Second

- $\text{MIPS} = (\text{Instructions} / \text{Cycle}) \times (\text{Cycles} / \text{Second}) \times 1\text{E-}6$
- FLOPS: considers only floating-point instructions
- Example: CPI = 2, clock frequency = 500MHz, 250 MIPS
- Example: compiler removes instructions, latency falls, MIPS increases

## Using Clock Frequency

- Cannot equate clock frequency with performance
- Proc A: CPI = 2,  $f = 500\text{MHz}$
- Proc B: CPI = 1,  $f = 300\text{MHz}$
- Given the same ISA and compiler, B is faster



# Pitfall & Amdahl's Law

- Amdahl. "Validity of the single-processor approach..." AFIPS, 1967.

## Make Common Case Fast

Consider improving fraction F of system with a speedup S.

$$\begin{aligned} T(\text{new}) &= T(\text{base}) \times (1-F) + T(\text{base}) \times F / S \\ &= T(\text{base}) \times [(1-F) + F/S] \end{aligned}$$

$$\text{Speedup} = 1 / [(1-F) + F/S] = T(\text{base})/T(\text{new})$$

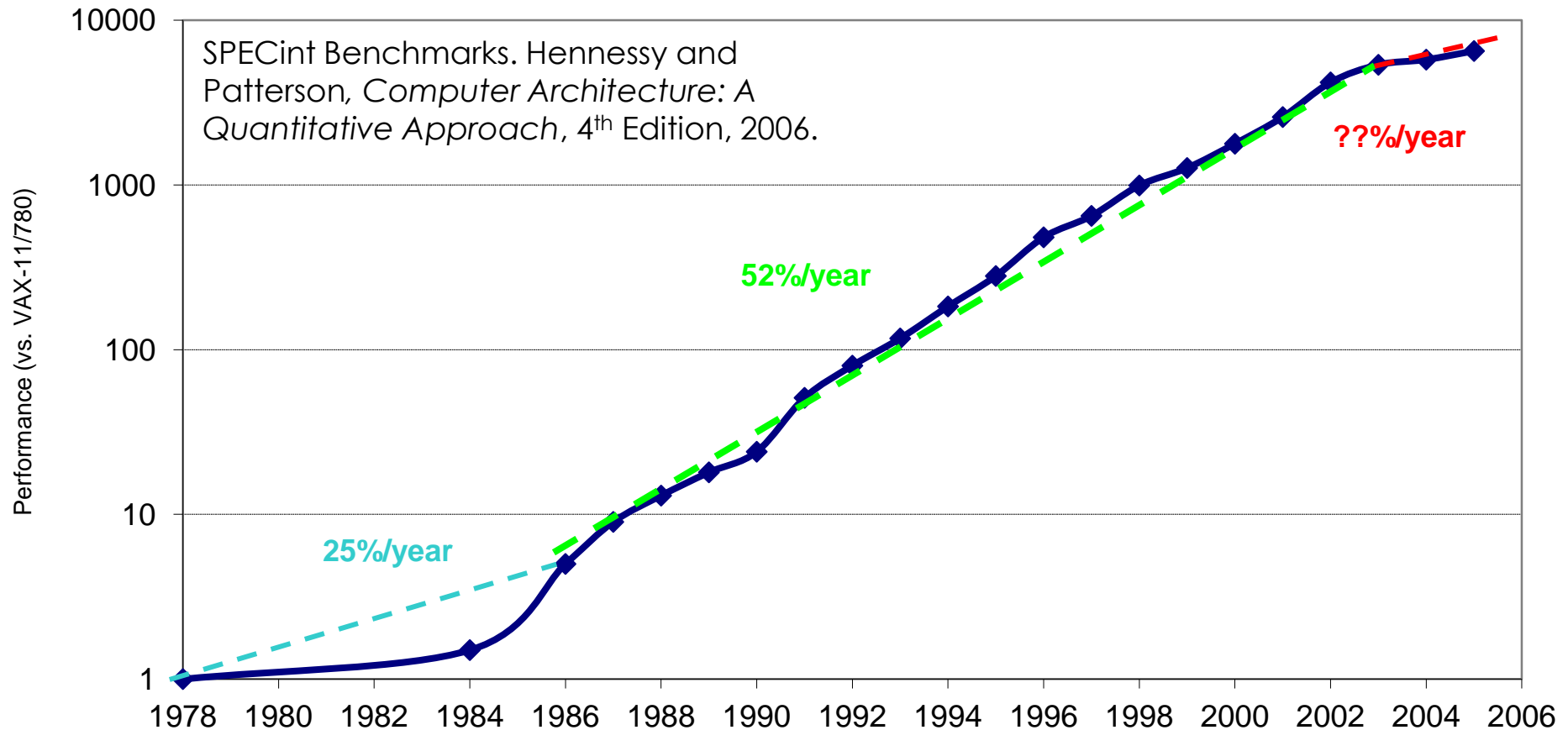
$$\text{Max Speedup} = 1 / (1 - F)$$

## Example

- Suppose FP computation is 1/4 of an application's execution time
- Maximum benefit from optimizing FP unit is 1.3x (=1/0.75)
- Multiprocessor systems were original application of this law
- Accounts for diminishing marginal returns



# Processor Power





# Power and Energy

## Definitions

- Energy (Joules) =  $a \times C \times V^2$
- Power (Watts) =  $a \times C \times V^2 \times f$

## Power Factors and Trends

- activity (a): function of application resource usage
- capacitance (C): function of design; scales with area
- voltage (V): constrained by leakage, which increases as V falls
- frequency (f): varies with pipelining and transistor speeds
- Models in cycle-accurate simulators (e.g., Princeton Wattch)

## Dynamic Voltage and Frequency Scaling (DVFS)

- P-states: move between operational modes with different V, f
- Intel TurboBoost: increase V, f for short durations without violating thermal design point (TDP)



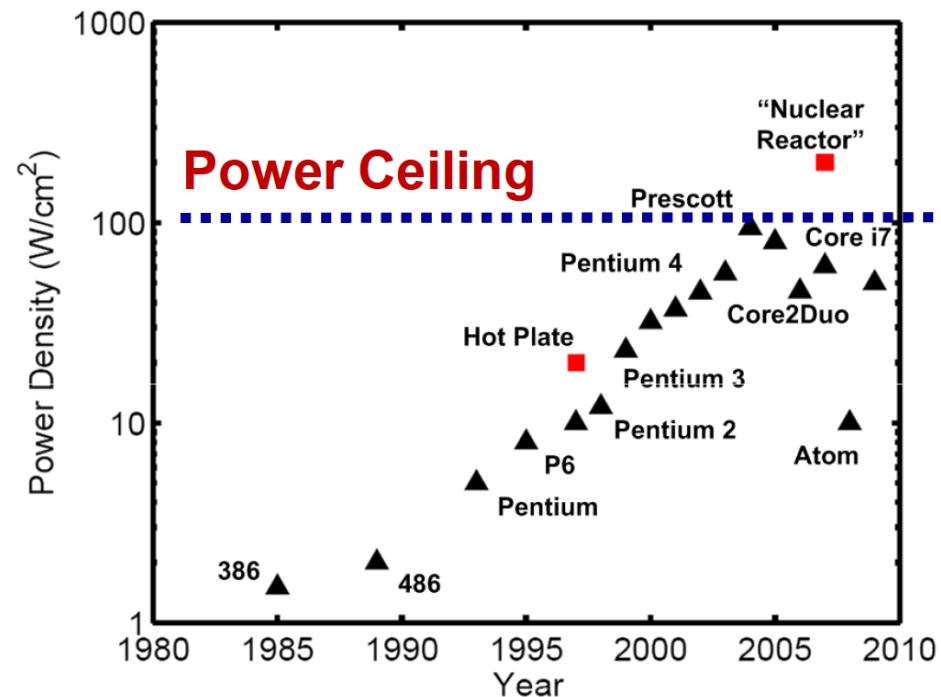
# Power and Temperature

## Temperature

- Power density (Watts / sq-mm) is proxy for thermal effects
- Estimate thermal conductivity and resistance to understand processor hot spots (e.g., University of Virginia, HotSpot simulator)

## Power Budgets

- Higher power budgets increase packaging cost
- 130W servers, 65W desktops, 10-30W laptops, 1-2W hand-held





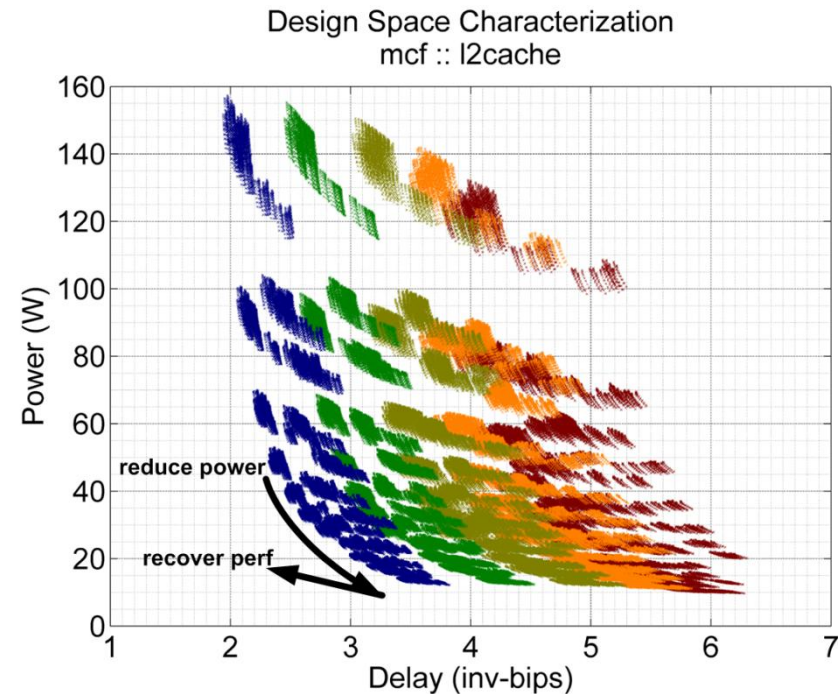
# Power and Chip-Multiprocessors

## Definitions

- Historically, multiprocessors use multiple packages (e.g., IBM Power 3)
- Chip multi-processor integrates multiple cores on the same die

## Multiprocessor Efficiency

- Reduce power with simpler cores
- Recover lost performance with many core parallelism (e.g., IBM Power 4)







# Power and Chip-Multiprocessors

## Lower voltages, frequencies

- Voltage, frequency scale together (approximately)
- Power proportional to  $V^2$ ,  $f$  (falls cubically)
- Performance proportional to  $f$  (falls linearly)

## Example

- Baseline: 1-core at  $V$ ,  $f$
- Multiprocessor: 4-cores at  $0.85V$ ,  $0.85f$ ; program is 75% parallel
- 1-Core Power  $0.61x = 0.85^3$
- 1-Core Performance  $0.85x$
- Power impact  $2.44x = 0.61x \cdot 4$
- Performance adjusted for parallelism  $2.28x = 1/[0.25 + (0.75 / 4)]$
- Performance adjusted for freq slowdown  $1.94x = 2.28 \times 0.85$
- Multiprocessor: 1.5% power per 1% performance  $(=144\%/94\%)$
- Higher  $V$ ,  $f$ : 3% power per 1% performance  $(=(1.01^3-1)/(1.01-1))$



# Cost

## Non-recurring Engineering (NRE)

- Dominated by engineer-years (\$200K per engineer-year)
- Mask costs (>\$1M per spin)

## Chip Cost

- Depends on wafer and chip size, process maturity

## Packaging Cost

- Depends on number of pins (e.g., signal + power/ground)
- Depends on thermal design point (e.g., heat sink)

## Total Cost of Ownership

- Capital costs (e.g., server procurement cost)
- Operating costs (e.g., electricity)



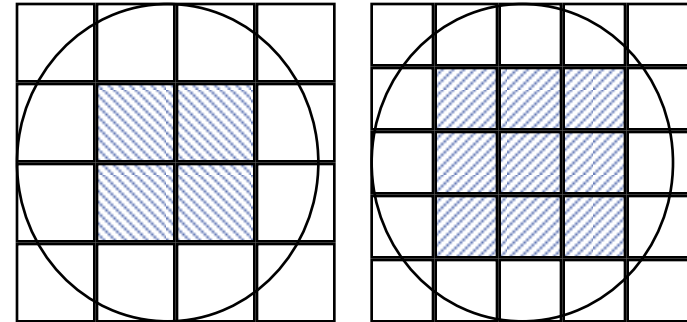
# Yield

## Wafers

- Integrated circuits built with multi-step chemical process on wafers
- Cost per wafer depends on wafer size, number of steps

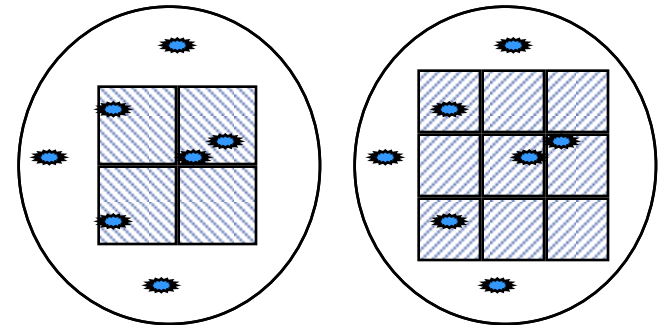
## Chip (a.k.a. Die)

- If chips are large, fewer chips per wafer
- Larger chips have lower yield
- Uniform defect density
- Chip cost is proportional to  $\text{area}^{2-3}$



## Process Variability

- Yield is non-binary
- Binning for speed grades
- Binning for core count
- Post-fabrication tuning with spares





# Acknowledgements

These slides contain material developed and copyright by

- Arvind (MIT)
- Krste Asanovic (MIT/UCB)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- John Kubiatowicz (UCB)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)