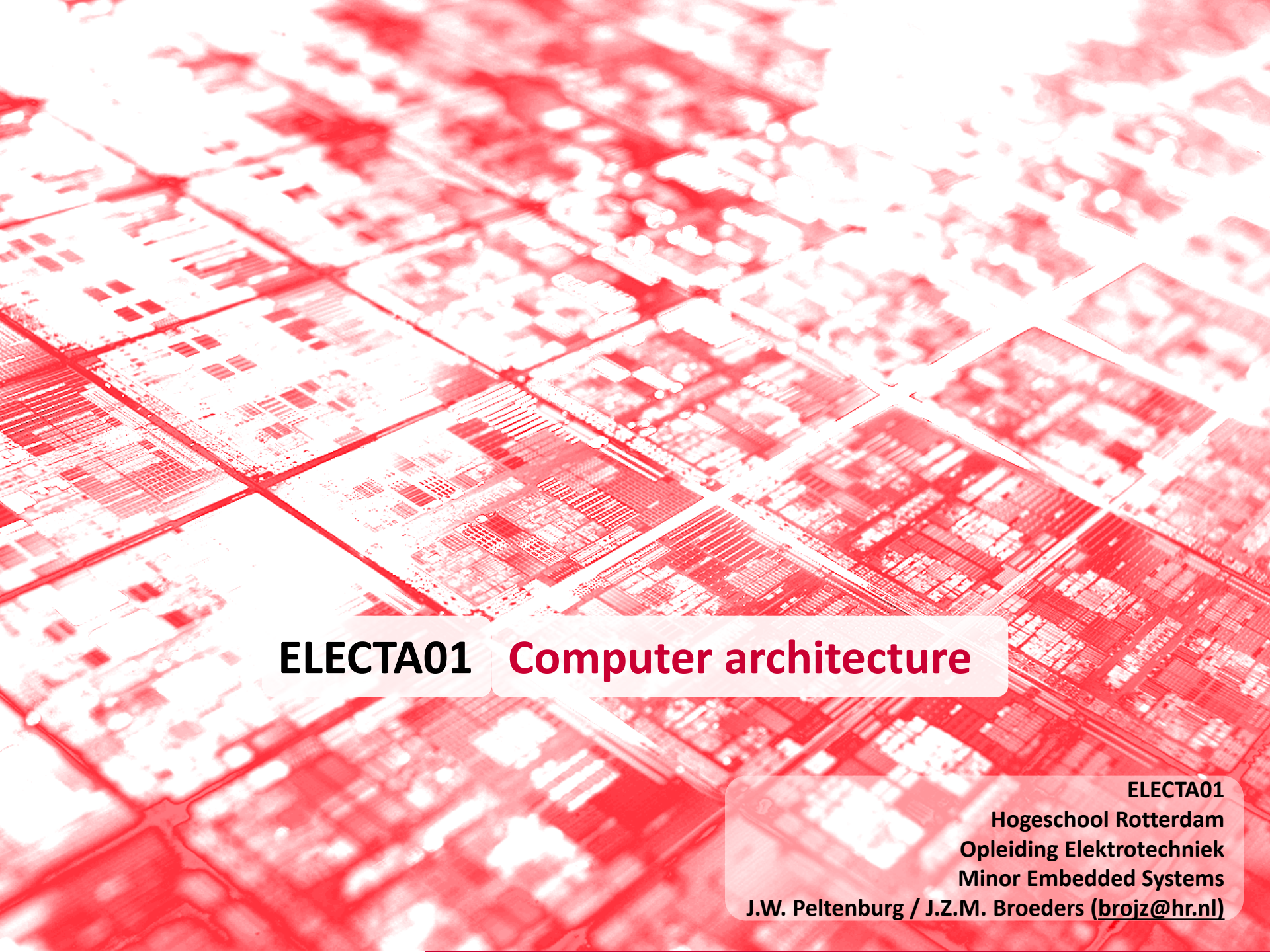


ELECTA01 Computer architecture

ELECTA01
Hogeschool Rotterdam
Opleiding Elektrotechniek
Minor Embedded Systems
J.W. Peltenburg / J.Z.M. Broeders (brojz@hr.nl)

Organization: Lectures & Homework

- Lectures
  - 8 lectures

    I talk about the theory.
    We do some exercises.
    Repeat.

    Interrupt based
    If questions arise: ask.

- Homework
  - Read the book
  - Do exercises from the book.

Organization: examination

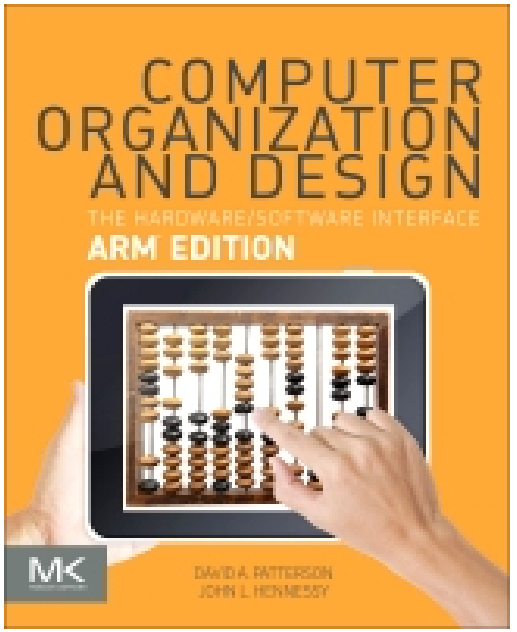
- Homework Lab (Group of maximal 2 students. 25% of final grade)
  - Writing assembly for the ARMv8 processor
    - You get some C code or description of an algorithm
    - “Compile” manually into assembly
    - Run in a simulator

- Written exam (75% of final grade)
  - 135 minutes in week 9 (T1). Resit at end of quarter 2 (HT1).
    - 60 minutes formative exam in week 4.

See “Curushandleiding” for details.

Organization: Book

- Book
  - Computer Organization and Design
    - The Hardware / Software Interface
      - ARM Edition
      - David A. Patterson, John L. Hennessy
      - ISBN: 9780128017333
    - Standard textbook on the subject
    - Good availability
    - Lots of exercises
    - **Mandatory for this course** (must be used during the exam)



Planning CTA01

- **Week 1: Introduction, Performance & Parallelism**
- Week 2: Instructions for arithmetic and memory
- Week 3: Instructions for decisions and procedures
- Week 4: Computer arithmetic
- Week 5: Single cycle LEGv8, intro pipelining
- Week 6: Pipelined LEGv8, hazards and forwarding
- Week 7: Memory architecture
- Week 8: Guest lecture (multicore and parallelism)

Preliminary Knowledge

- Required to have knowledge of:
  - Elementary Digital Circuits
  - C programming

Why Computer Architecture?

- Exploit your knowledge about the hardware when writing software
  - Make code run faster and more efficient
  - Writing inline assembly is sometimes necessary (but rarely)
  - Can my µC run my code fast enough? Why (not)?
- Connect superfast hardware peripheral to a processor
- More and more custom So(P)C designs
  - Also small design companies
- It's everywhere!
  - New automobile: 150+ microprocessors
  - Average household: 50+ microprocessors
  - Supercomputer: 25000+ processors
  - Your gaming GPU: 500+ cores

And what is more…

- After completing HWP01 and CTA01 you should be able to, with some effort, write your own simple ARM-like processor in VHDL (although this is outside the scope of this course).

Agenda

- Introduction
- **History**
- Layers and abstraction
- Components of a computer
- Performance analysis

Computing technology history

- 1947: First electrical and digital computer: ENIAC (von Neumann)
  - Around that time the *transistor* was invented at Bell Labs: components became faster and faster and cheaper and cheaper.
- 1972: First commercially available microprocessor *in one chip* by Intel (Faggin, Hoff, Mazer, Shima)
  - Around this time Texas Instruments and Pico/General Instruments were also building single chip processors.
  - Single chip designs made everything *much* faster.

The Zuse Z4 was the first commercial digital computer.

Computer science & scientists

Computer science: the formalisms and mathematics behind solving problems by computing

- George Boole:

  Boolean algebra
- Ada Lovelace:

  Algorithms for machines
- Alan Turing:

  Formal mathematical framework of computation
- Claude Shannon:

  Information theory, Boolean description of digital circuits
- John von Neumann:

  Stored-program concept

Chapter 1

Computer Abstractions and Technology

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Smartphones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

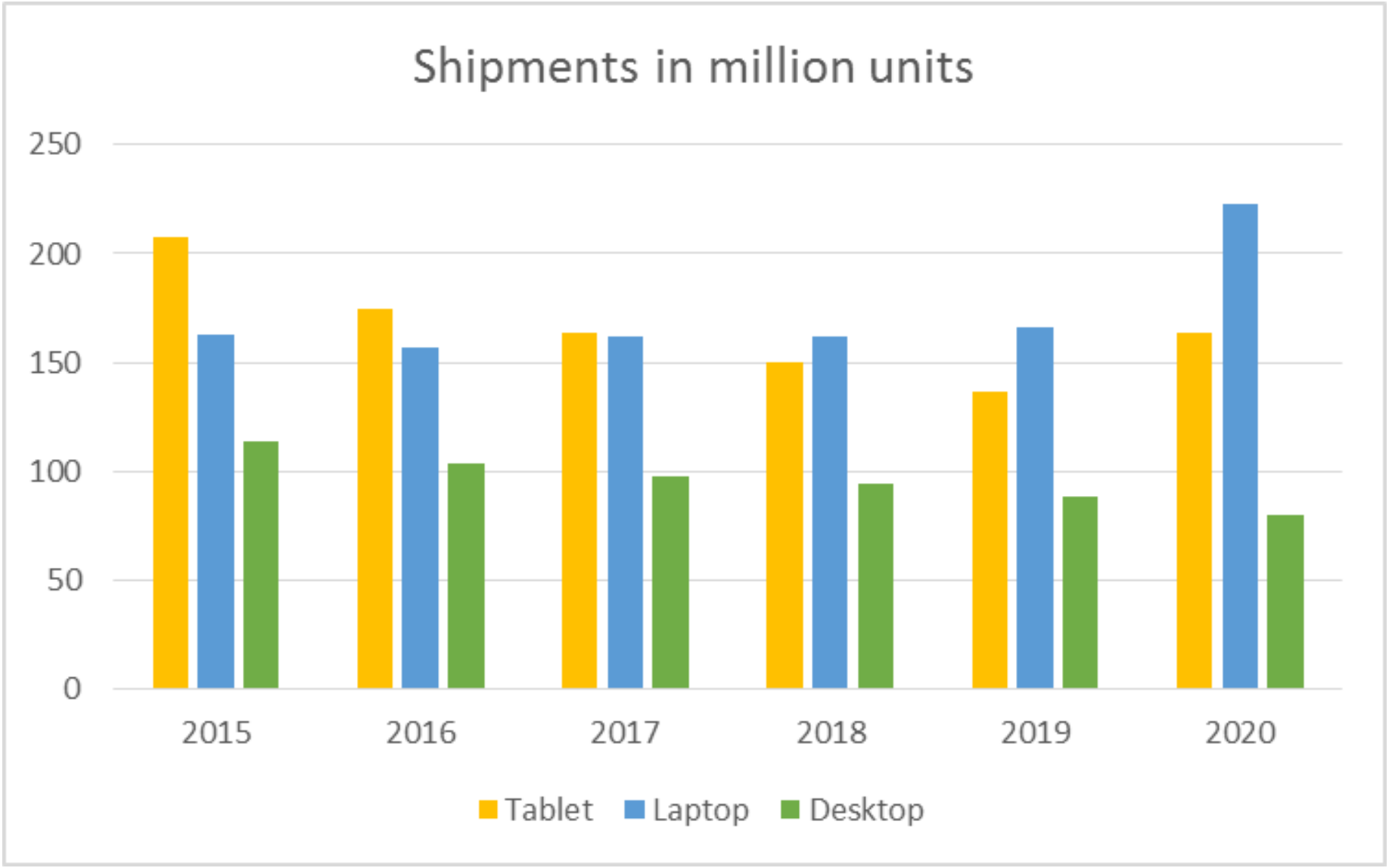
Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

Classes of Computers

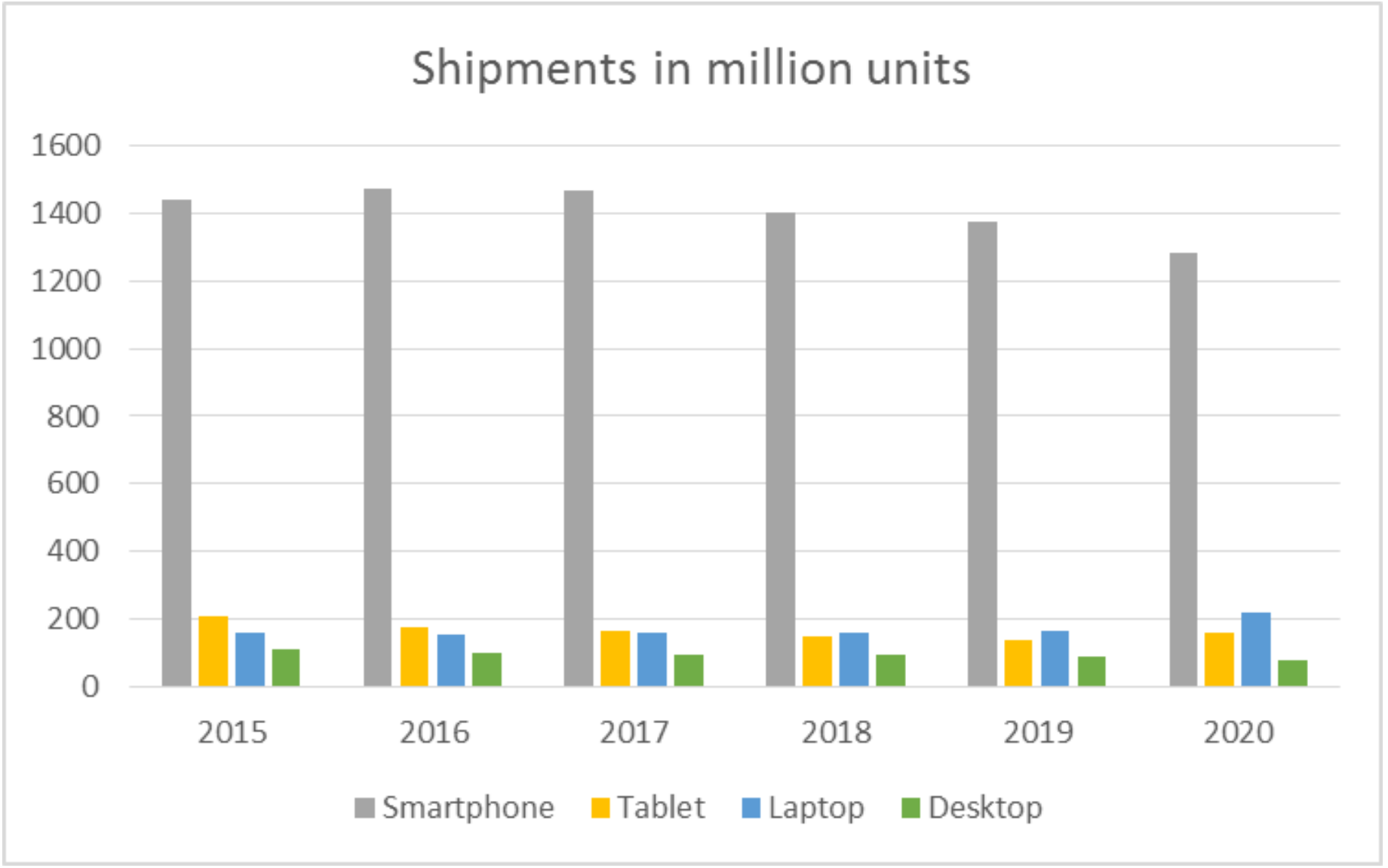
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The PostPC Era



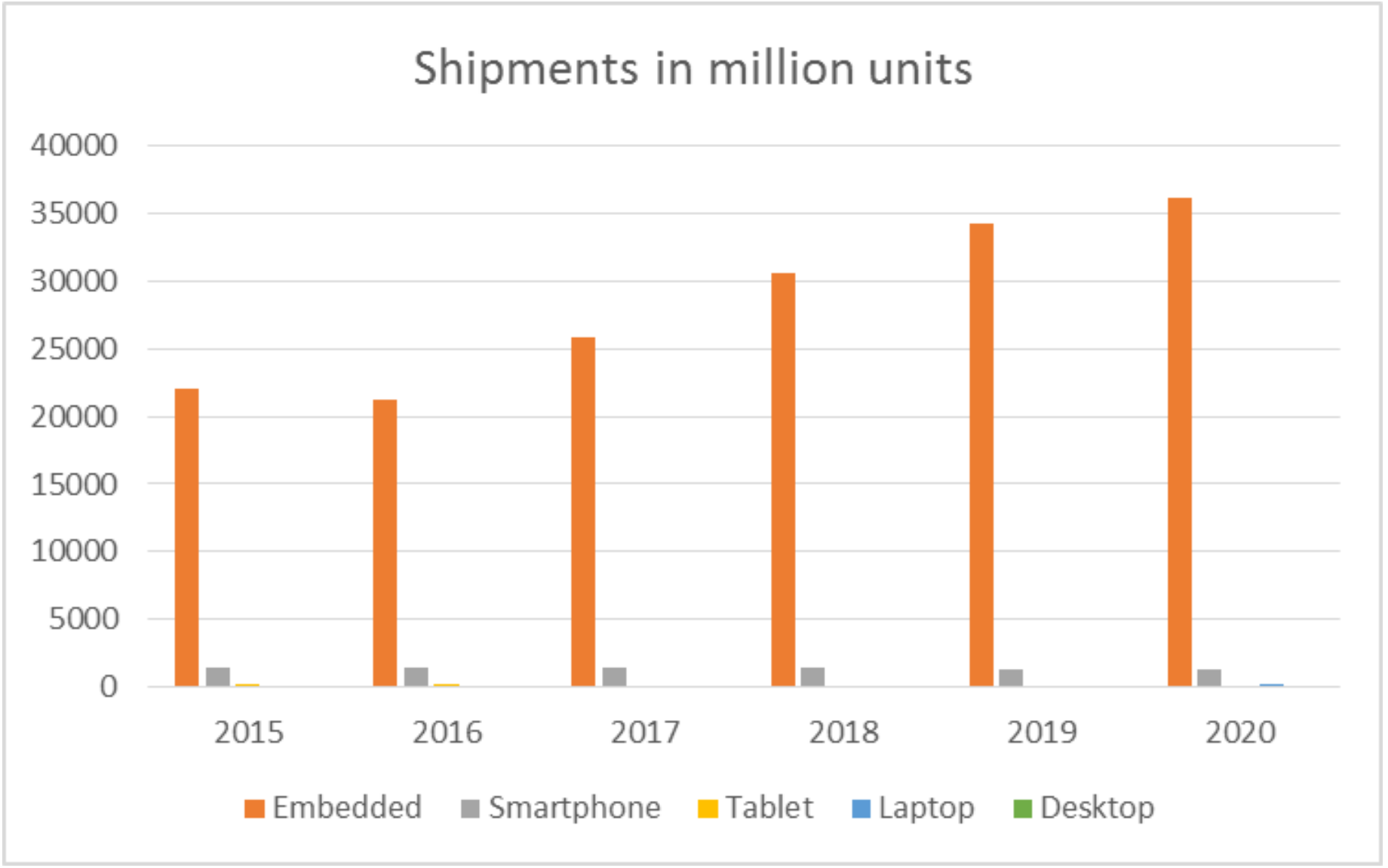
Bron: https://www.statista.com/statistics/272595/global-shipments-forecast-for-tablets-laptops-and-desktop-pcs/

The PostPC Era



Bron: https://www.statista.com/statistics/263441/global-smartphone-shipments-forecast/

The PostPC Era



Bron: https://www.icinsights.com/news/bulletins/MCUs-Sales-To-Reach-RecordHigh-Annual-Revenues-Through-2022/

The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, smart watches
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

What You Will Learn

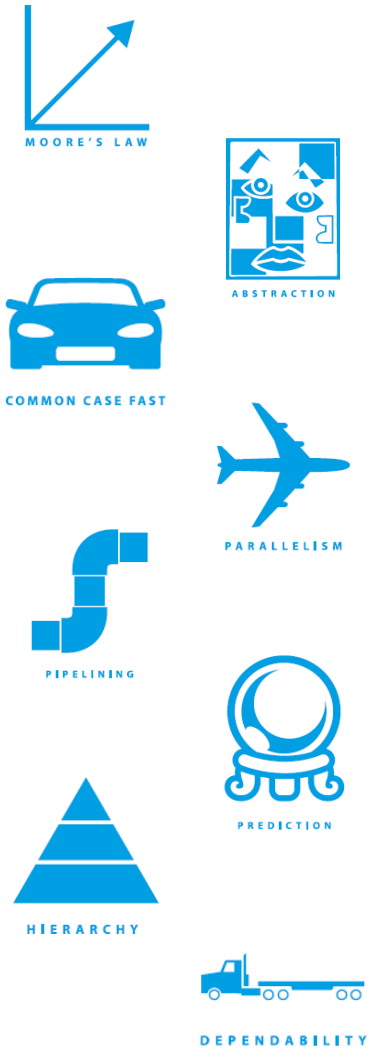
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

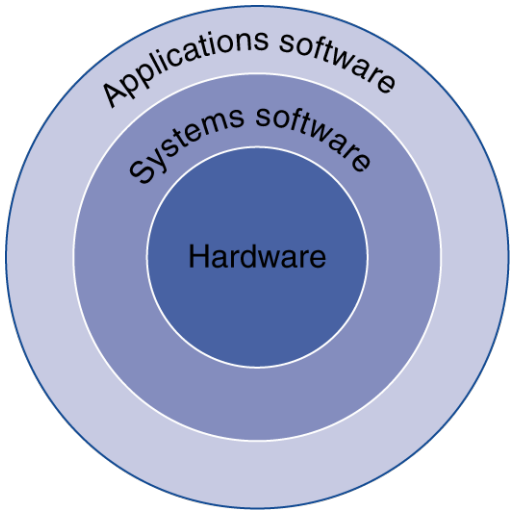
- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast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

Eight Great Ideas

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



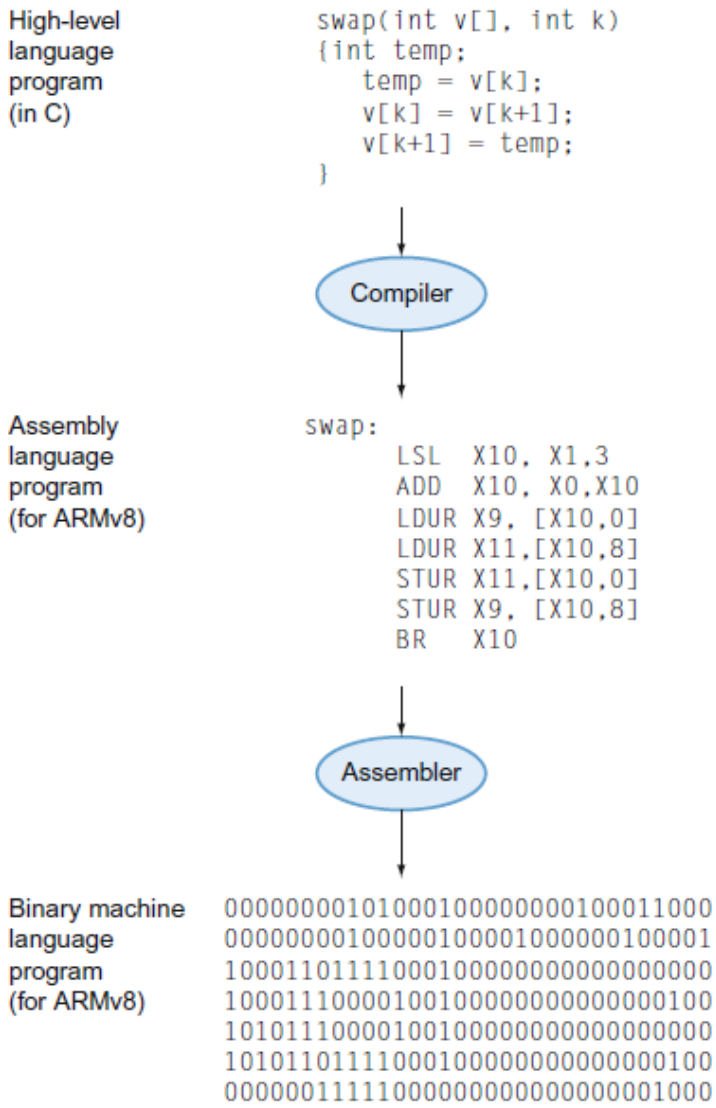
Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

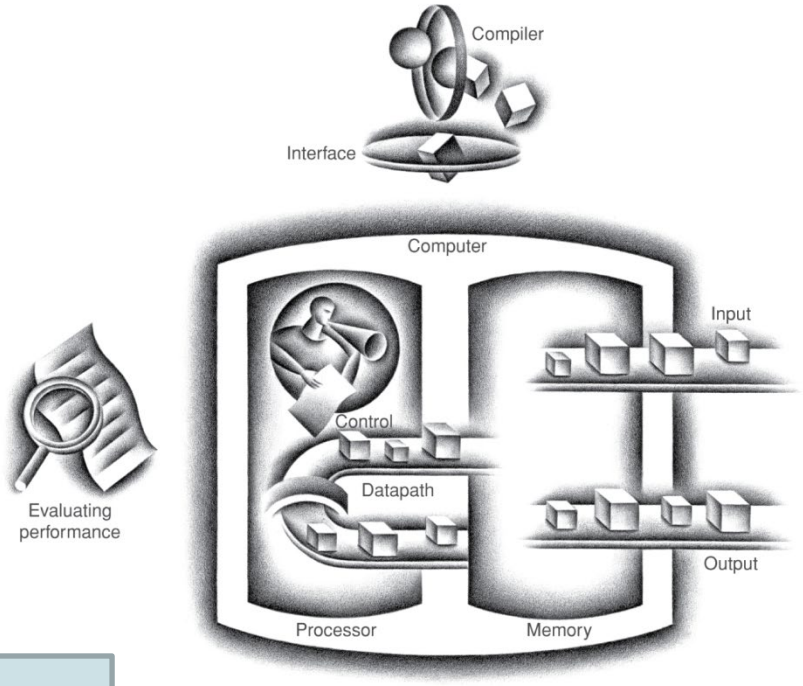


Components of a Computer

- **I** skip this part of the lecture.
- **You** have to study paragraph 1.4 yourself!

Components of a Computer

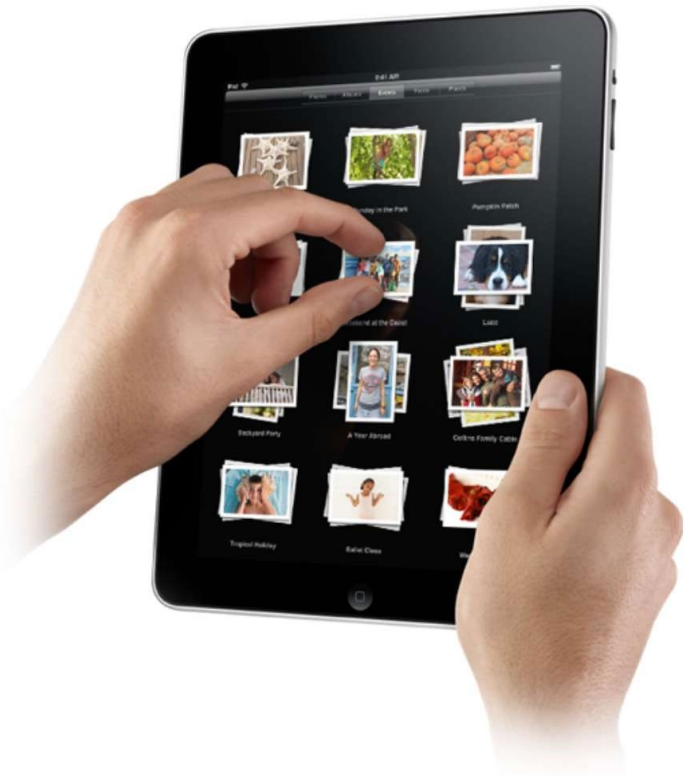
The BIG Picture



- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

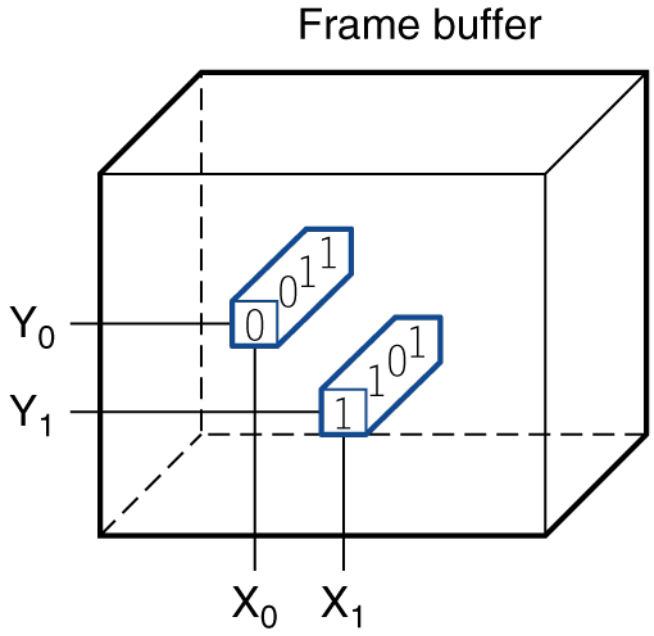
Touchscreen

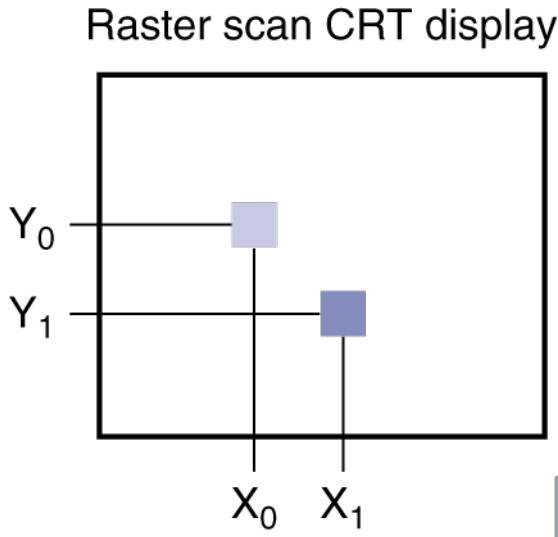
- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



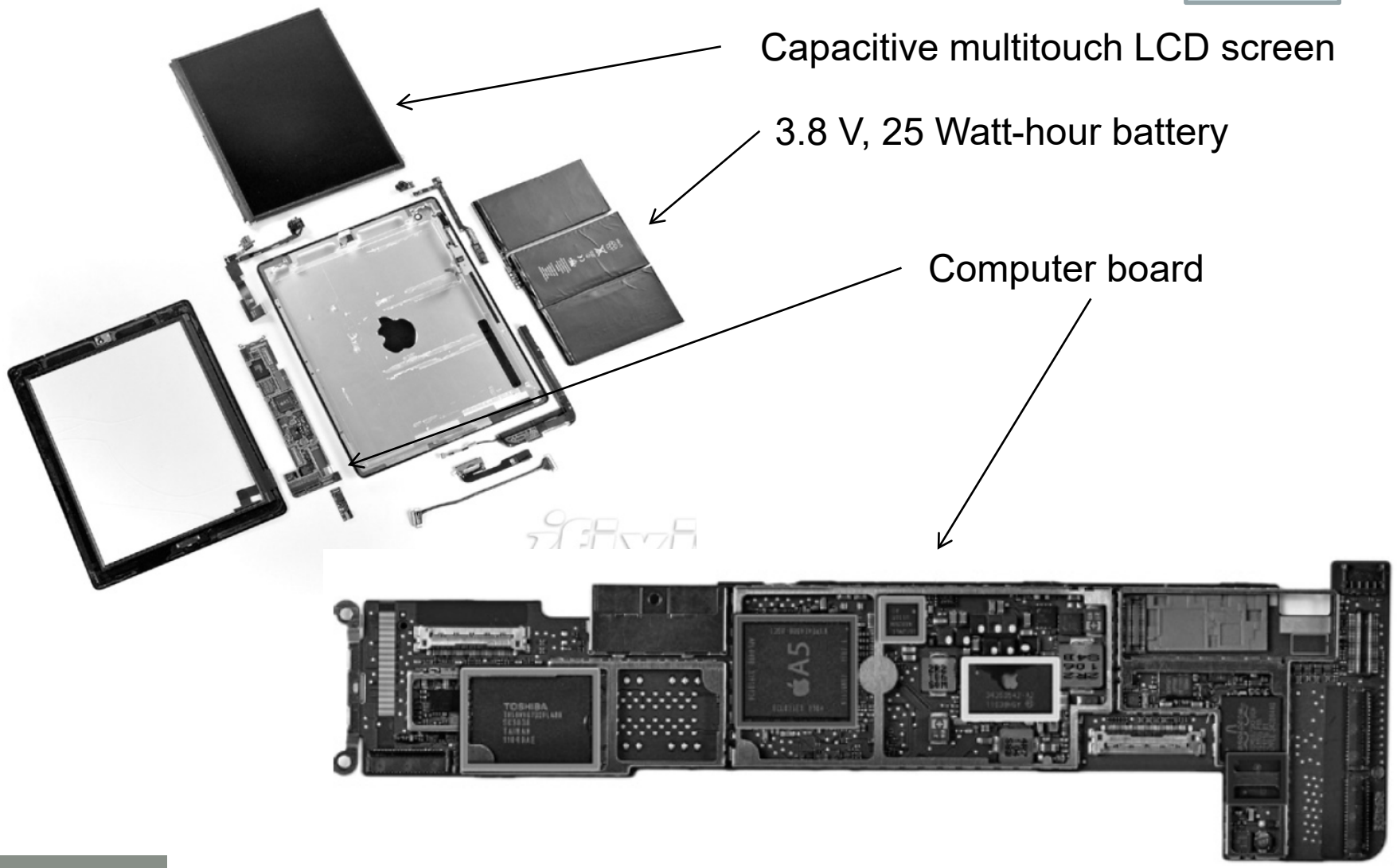
Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



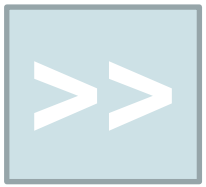


Opening the Box



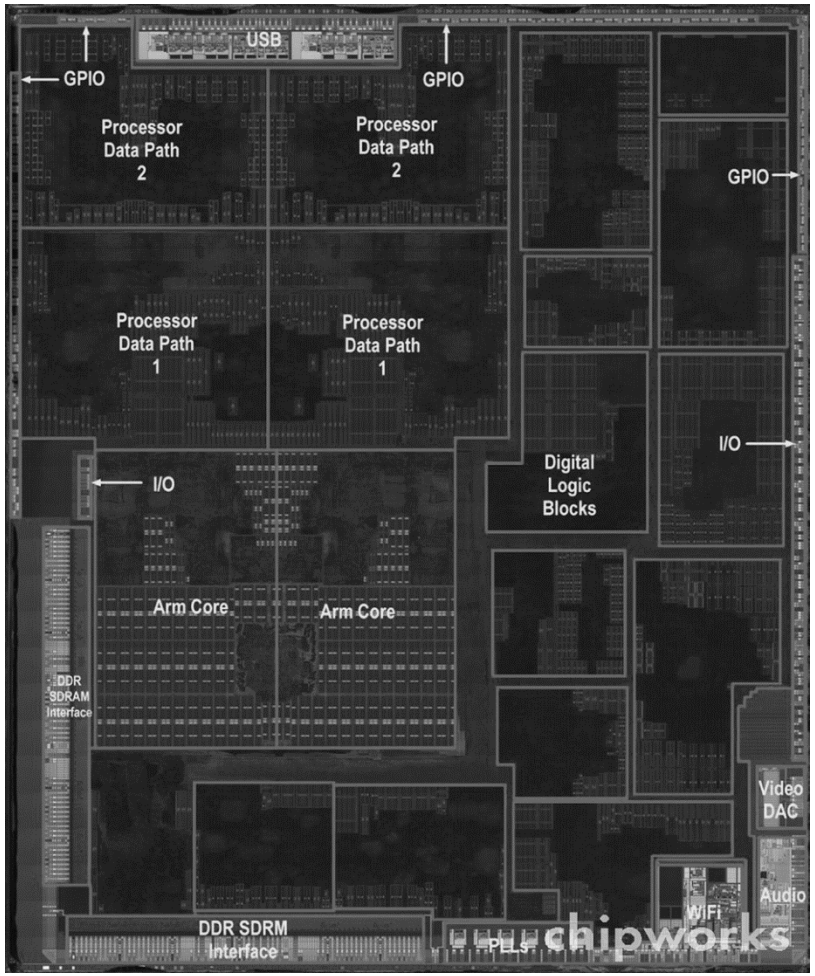
Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

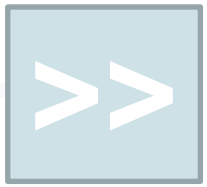


Inside the Processor

- Apple A5



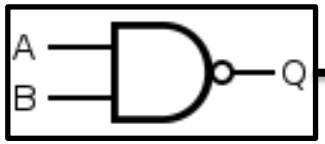
Abstractions

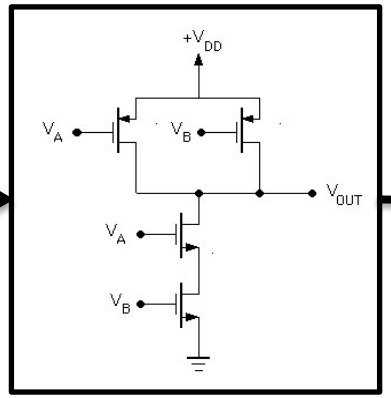


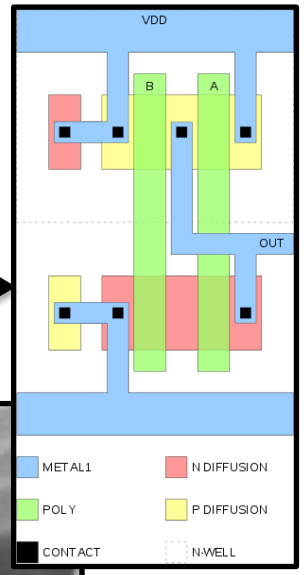
The BIG Picture

- 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

Abstraction example AND gate







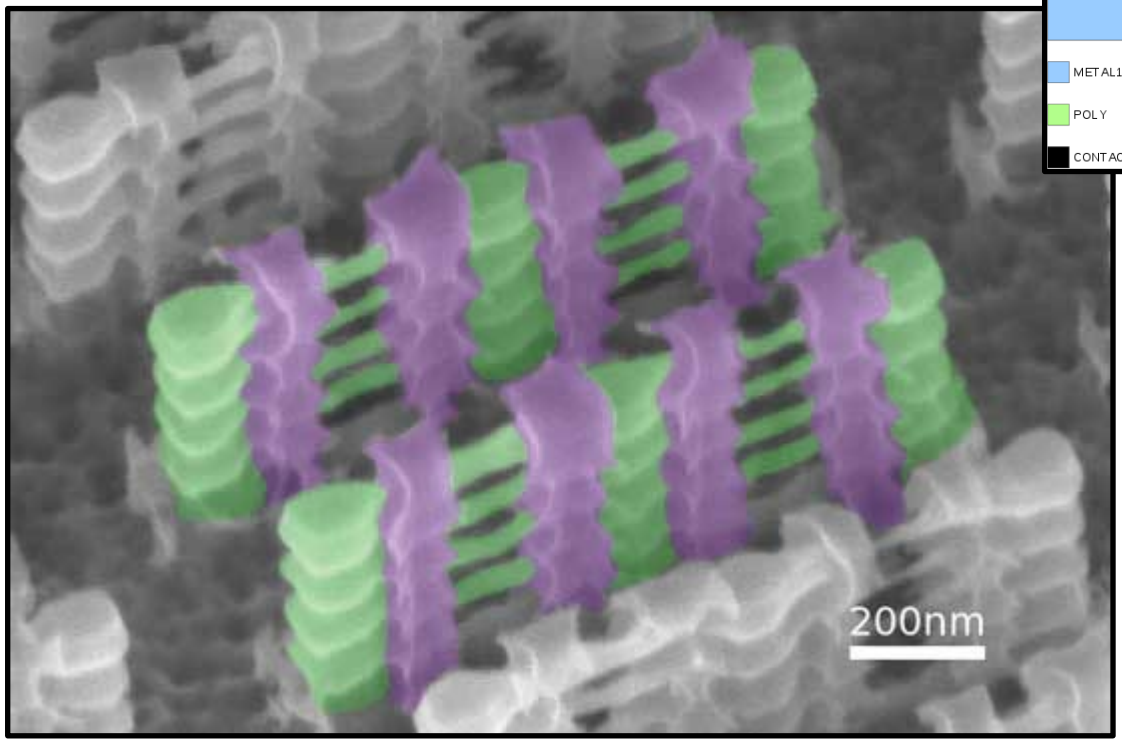
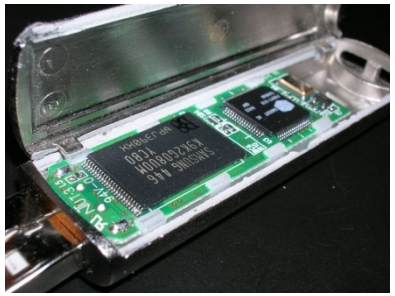


Image Sources: http://people.seas.harvard.edu/~jones/es154/lectures/lecture_7/MOS/mos_logic.html
http://www.creativecommons.org
http://www.electroiq.com/articles/sst/print/vol-55/issue-8/features/cover-articles/a-review-of-recent-advances-in-electronic.html

HOGESCHOOL ROTTERDAM

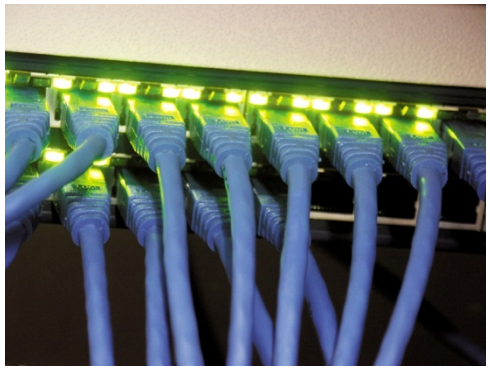
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth

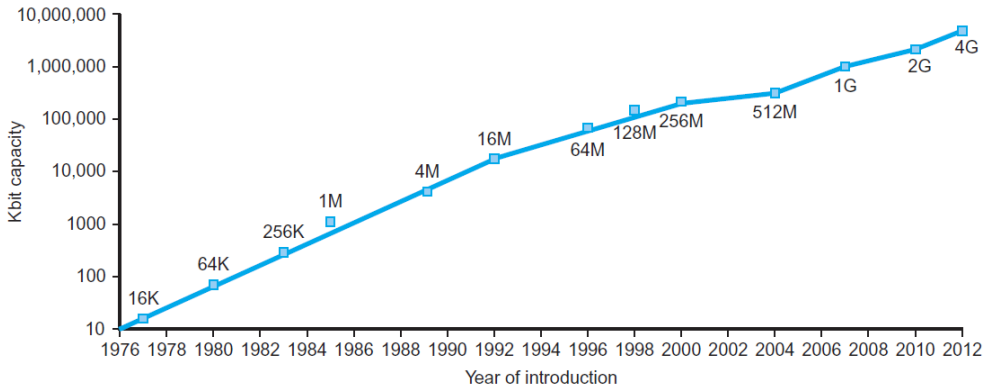


Semiconductor Technology

- **I** skip this part of the lecture.
- **You** have to study paragraph 1.5 yourself!

Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

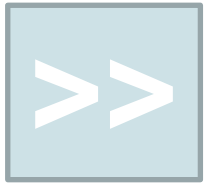


DRAM capacity

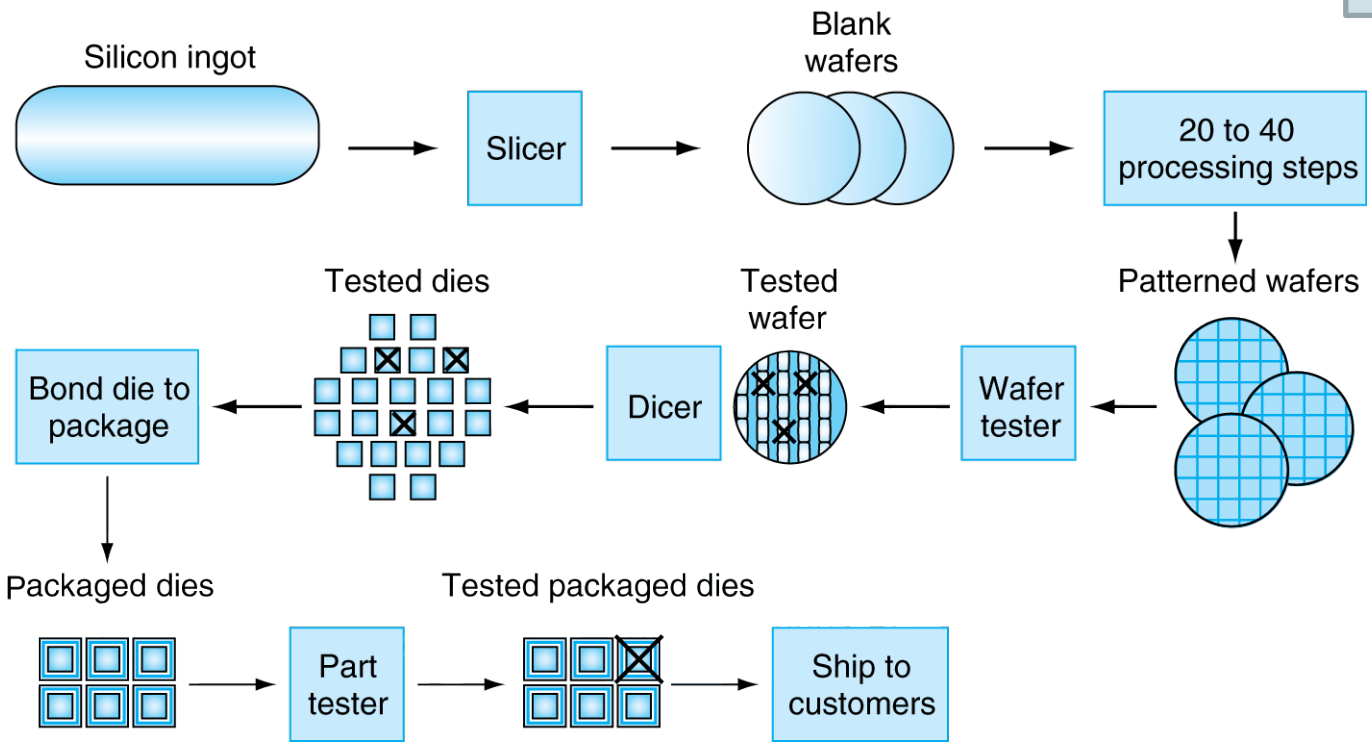
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

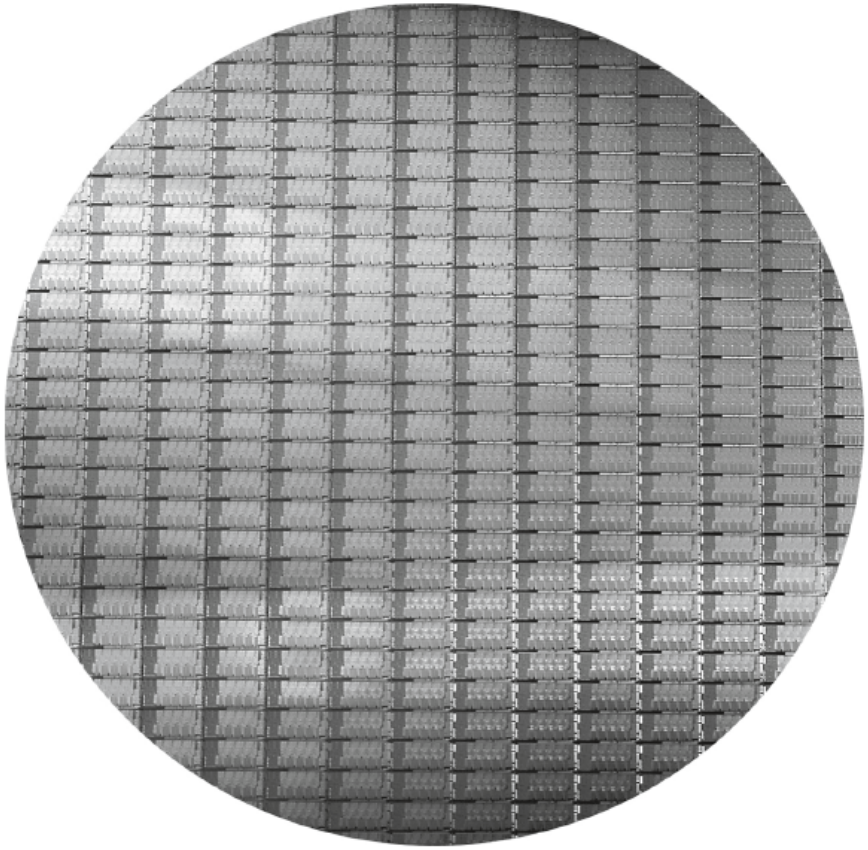


Manufacturing ICs



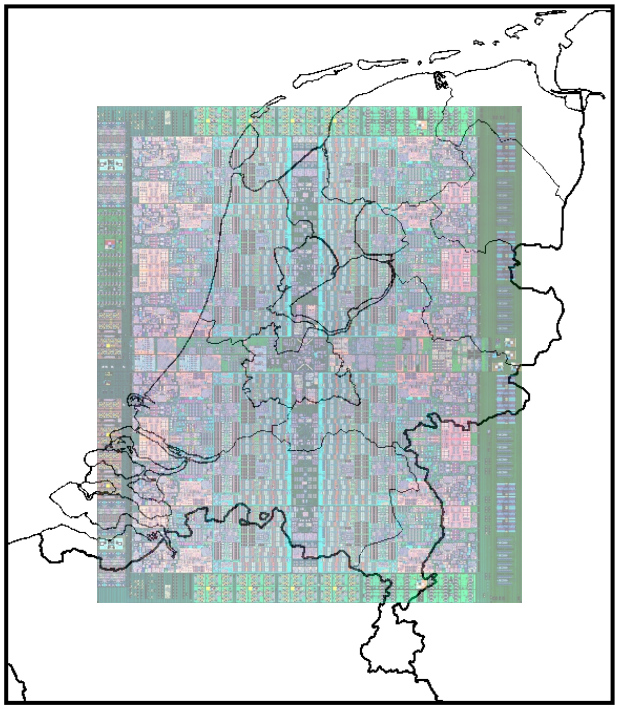
- Yield: proportion of working dies per wafer

Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

Semiconductor fabrication technology (1)



Compare IBM's 12-core POWER8 chip (2013) to a map of The Netherlands

Process used:	22nm
Chip area:	approx. 6.5 cm^2
Netherlands area:	approx. 40,000 km^2

Scale:
2.6 cm / 200 km → 1:7,700,000 (approx.)

Compares to the Netherlands filled with roads

- 0.17 meters wide
- 0.17 meters apart
- 15 layers

>>

Inspiration for this slide is taken with permission from the digital IC design course of Nick van der Meijs at: http://cas.et.tudelft.nl/~nick/courses/digic/

Semiconductor fabrication technology (2)

- Wafer diameter: 300 mm
- Netherlands area: approx. 40,000 km^2
- Equivalent diameter: approx. 225 km

300 mm	:	225 km
1	:	750000
22 nm	:	~1.7 cm

A special wafer stepper could print The Netherlands with a resolution of 1.7 cm in 20 seconds!

Integrated Circuit Cost

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

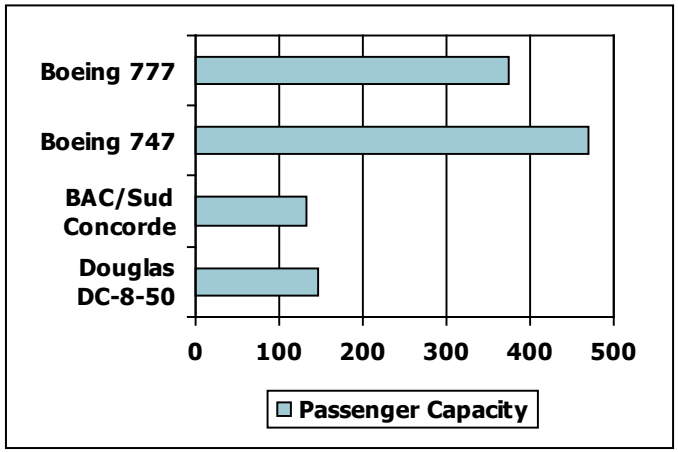
$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

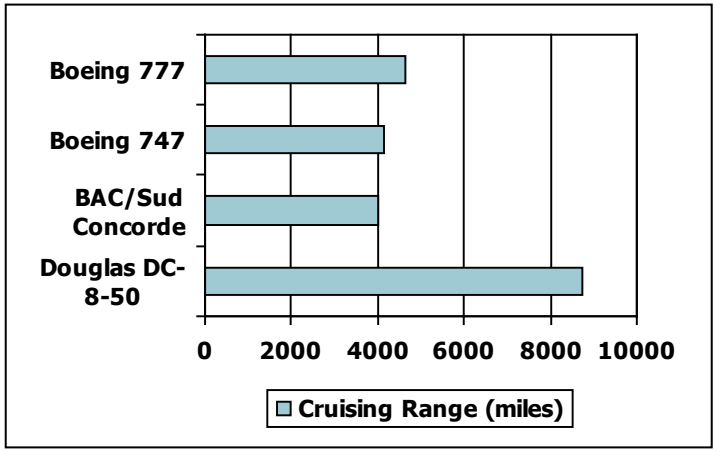
$$\text{Yield} = \frac{1}{(1+(\text{Defects per area} \times \text{Die area}/2))^2}$$

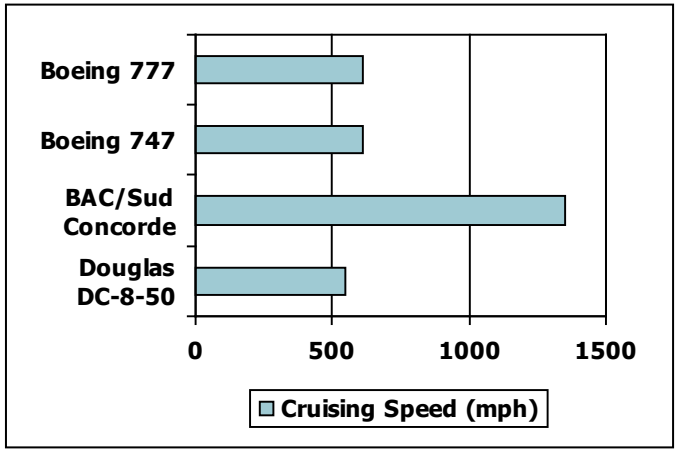
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

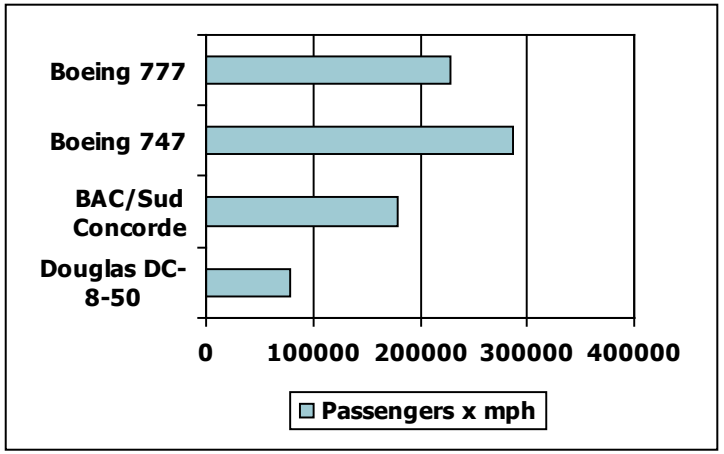
Defining Performance

- Which airplane has the best performance?









Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

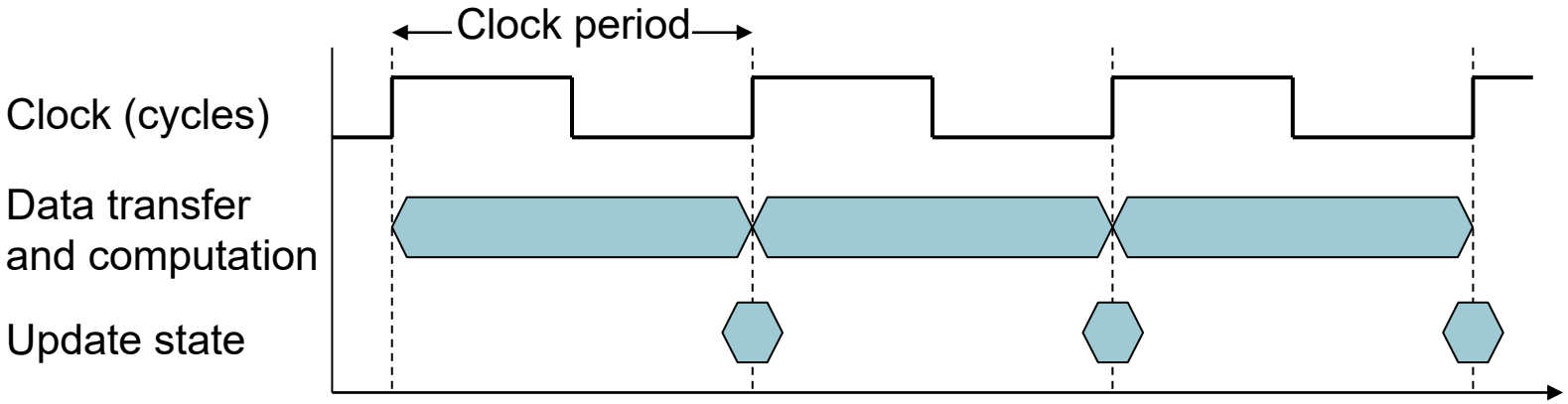
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



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

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 number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock 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

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

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

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

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- 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

- If different instruction classes take different numbers of cycles

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

  - 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

Calculate Average CPI

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

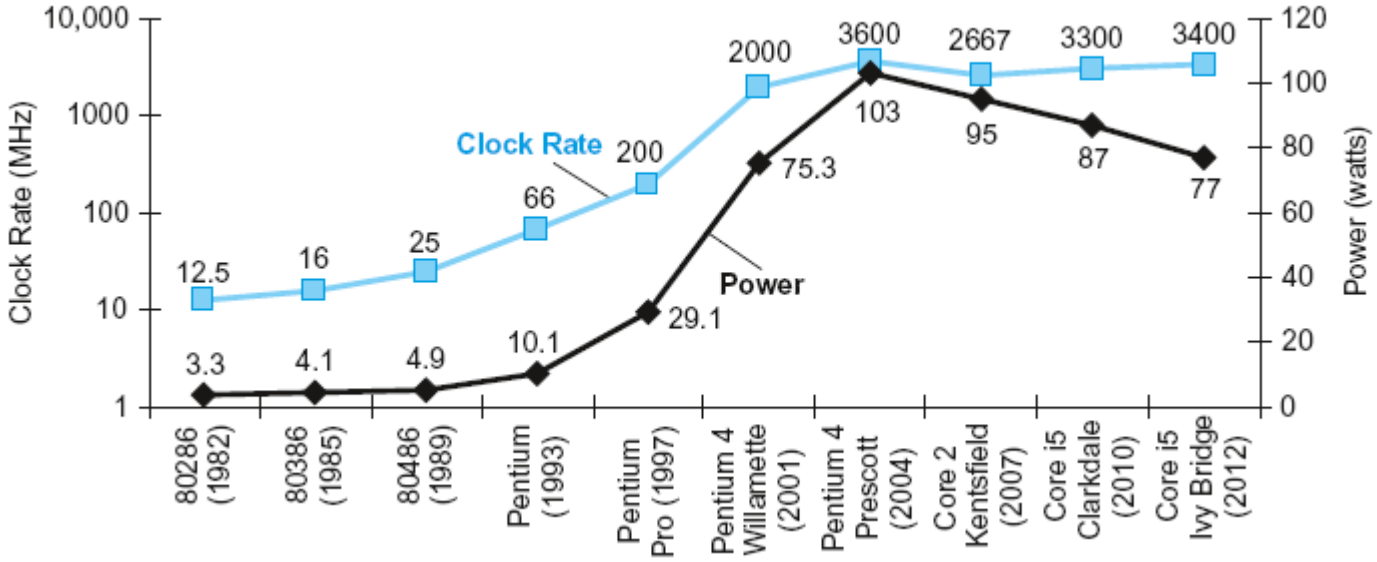
Performance Summary

The BIG Picture

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

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

Power Trends



- In CMOS IC technology

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

×30 | Proportional to | 5V → 1V | ×1000

Reducing Power

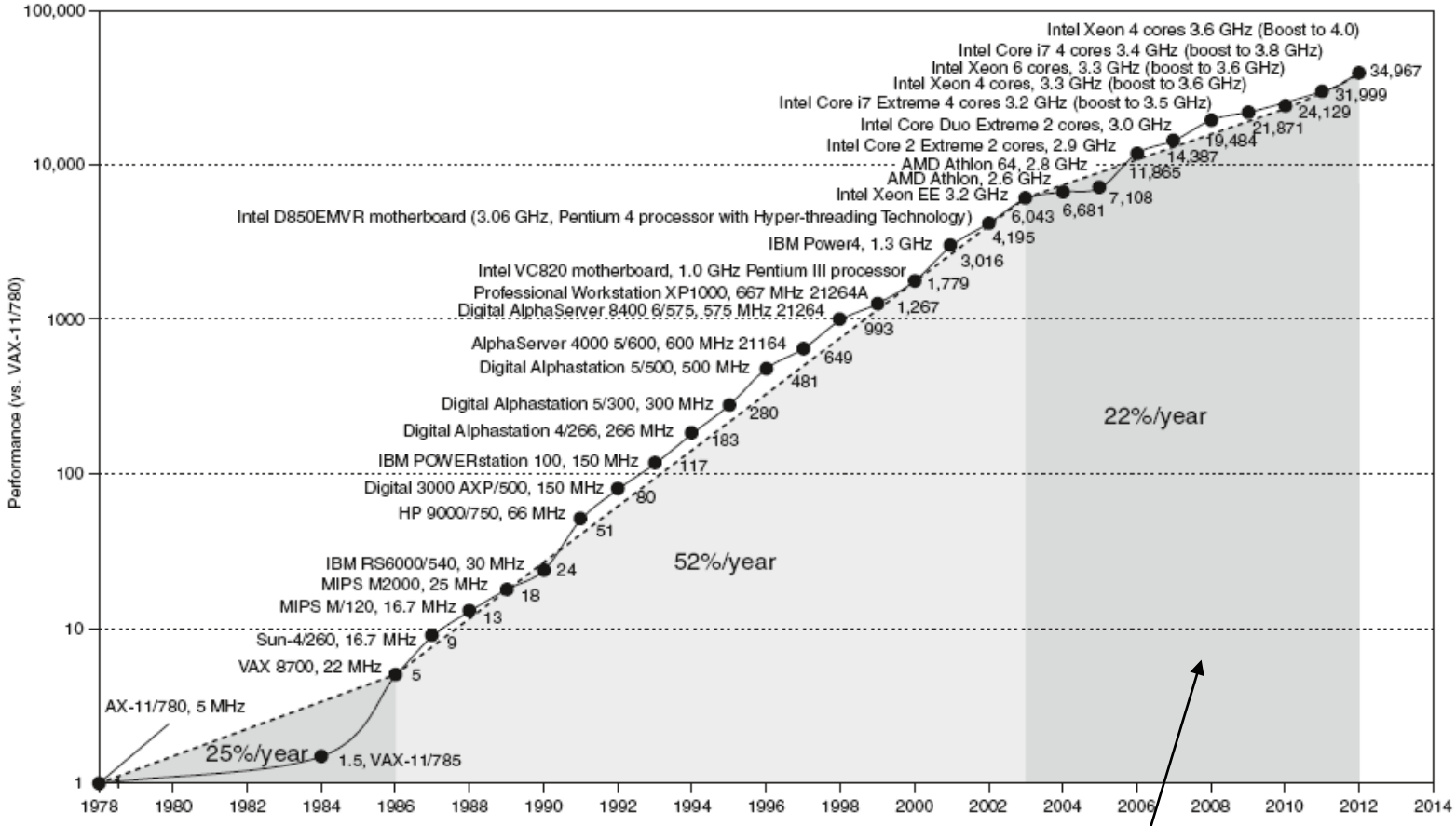
Calculate P_{new}/P_{old}

- 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 can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

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

Pitfall: Amdahl's Law

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

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

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

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

  - Can't be done!
- Corollary: make the common case fast

Amdahl's Law (1/2)

- Speedup of a system with 1 'processor':

$$S(1) = 1$$

- Suppose some application has
  - a portion of P that can be parallelized and 1-P that cannot be parallelized
  - N 'processors'

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

Exercise

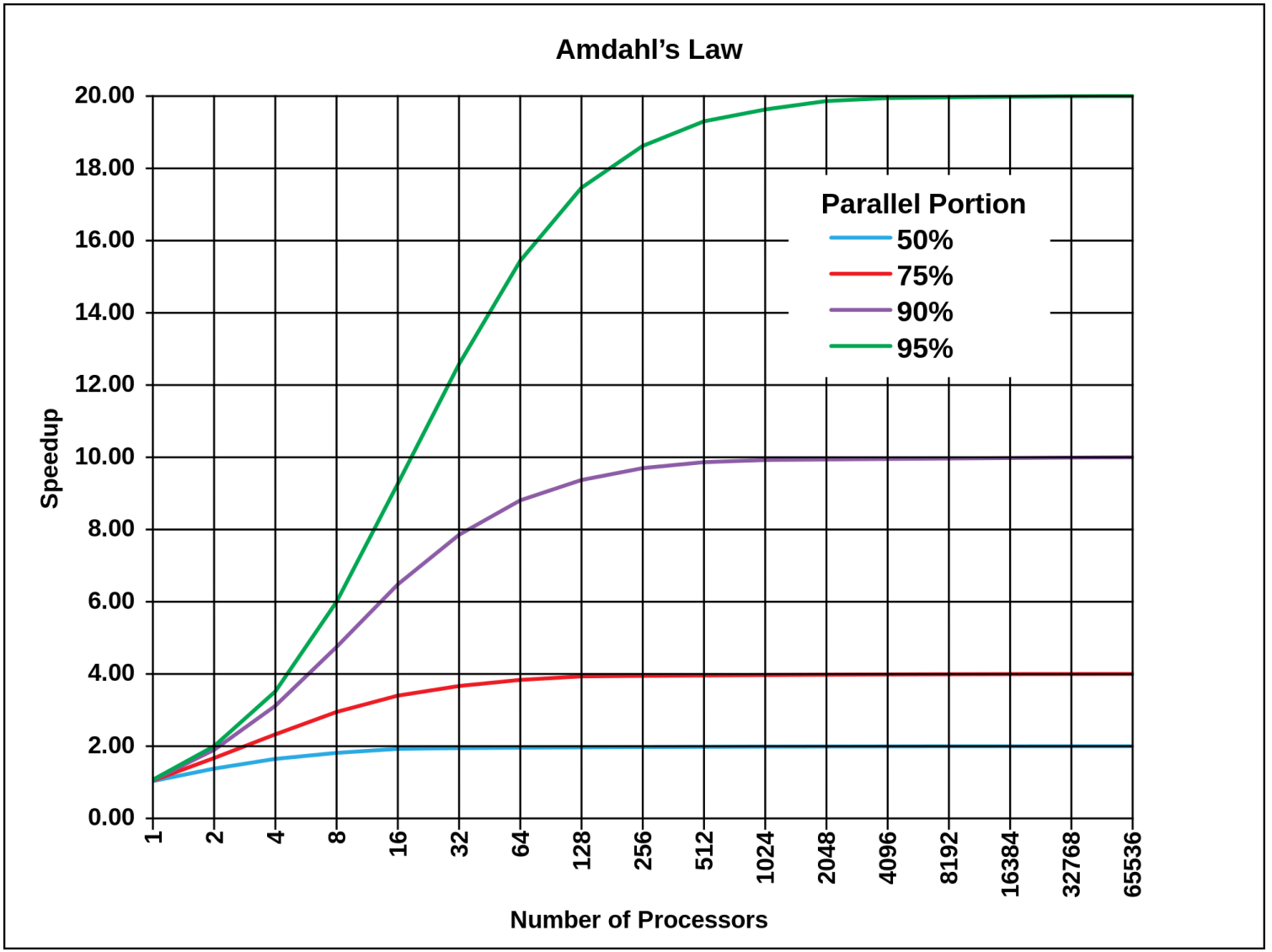
- Some application exhibits 90% parallelism.
- Assume perfect scalability.
- What is the maximum speed-up given:
  a) 4 processors?
  b) 16 processors?
  c) 256 processors?
  d) Unlimited processors?

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

- Note that:

$$\lim_{N \to \infty} S(N) = \frac{1}{1-P}$$

Amdahl's Law (2/2)



Source: http://commons.wikimedia.org/wiki/File:AmdahlsLaw.svg

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

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

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

  - CPI varies between programs on a given CPU

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance