# Then and Now

Fundamentals and Performance Analysis

- Building Blocks
- Logic Design Conventions
- Building a Datapath
- A Simple Implementation
- An Overview of Pipelining
- Pipelined Datapath and Control

# Catalog Description

- Advanced computer architecture, including discussion of instruction set design, virtual memory system design, memory hierarchies, cache memories, pipelining, vector processing, I/O subsystems, coprocessors, and multiprocessor architectures.  Case studies of current systems.
- Main objective:
  - Attain a strong foundation to understand and analyze computer architecture and to apply these insights and principles to future computer designs.

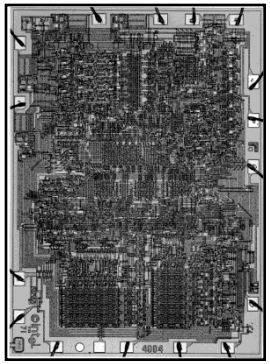
# Course Outline

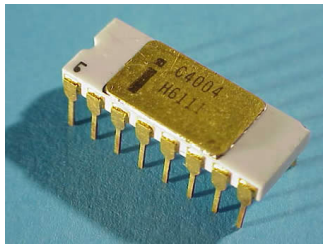
- **Fundamental principles**
  - Roadmap, performance analysis
- **Fundamental processors**
  - Instructions, datapath, controller, pipelining, resolving hazards, I/O, storage
- **Fundamental memories**
  - Memory technology, cache coherency, main memory, virtual memory
- **Parallel systems**
  - **Instruction-level parallelism**
    - Static and dynamic scheduling, superscalar architectures, branch prediction, software pipelining
  - **Data-level parallelism**
    - Vector, SIMD, GPU, loop optimizations
  - **Thread-level parallelism**
    - Multithreaded processors, centralized-memory multiprocessors, multicore systems
  - **Parallel systems**
    - Distributed-memory multiprocessors, memory synchronization and consistency
  - **Request-level parallelism**
    - Interconnection networks, distributed computers, cloud computing

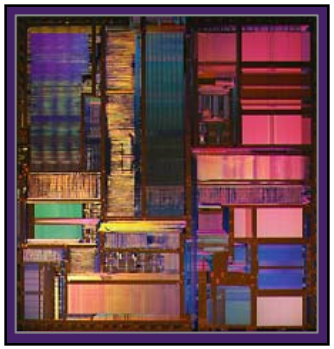
# Course Overview

- *Computer Architecture: A Quantitative Approach*. 5th ed. Elsevier, 2012.
  - And notes, articles, links…
- Four homework assignments: 10 percent each.
  - Book problems, exercises, paper review, drills...
- Term paper: 20 percent.
  - Specs given after midterm exam
- Midterm exam: 10 percent (online).
- Final exam: 10 percent (online) and 10 percent (take-home).
- Participation: 10 percent.
  - Attendance, posting current articles, reviewing content
- Academic honesty:
  - All work submitted must be the sole work of the author. Cheating in any form is not tolerated, nor is assisting another person to cheat. Such offense will result in a failing grade “F” and a letter of reprimand in your permanent student file.

# A Little History







ENIAC, 1946

17,000 vacuum tubes

5,000 calculations

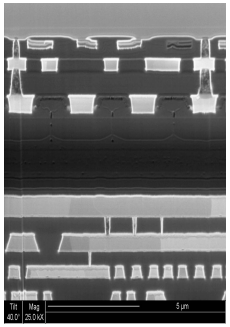
Mots: Debugging!

Intel 4004, 1971

2,300 transistors

60,000 calculations

10 micron transistors



Intel Pentium 4, 2000

55 million transistors

100 million calculations

0.09 micron = 90 nm

Hair: 100,000 nm
Cell: 7,000 nm
Virus: 100 nm
Atom: 0.5 nm

# Time Line

1943
*"I think there is a world market for maybe five computers."*

1946
Early electronic computers used glass valves.

1947
First transistor developed at Bell Labs.

1949
*"Computers in the future may weigh no more than 1.5 tons."*

1954
First fully transistorized Computer.

1958
First integrated circuit (IC).

1971
First microprocessor.

1975
*"Chip complexity will double every 1.5 years"*

1977
*"There is no reason anyone would want a computer in their home."*

1983
Apple introduces the first "user-friendly" computer.

1998
Pentium II.

Today?

# Innovations

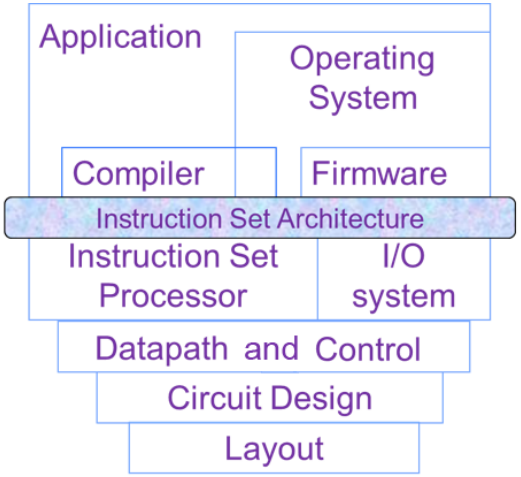
ENGINEERING@SYRACUSE

# Design Flow

# Requirements to Implementation

- Application Requirements → Technology Constraints
  - Abstraction layers needed to close the gap.
  - Computer architects are in between.
    - Feedback to application and technology directions
- Computer architecture is the development of the abstraction/implementation layers that allow us to:
  - Execute information-processing applications efficiently using available manufacturing technologies

# Abstraction Hides Complexity




Coordination of many *levels of abstraction*

High-Level Language Program

*Compiler*

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Assembly Language Program

*Assembler*

```
lw   $15, 0($2)
lw   $16, 4($2)
sw   $16, 0($2)
sw   $15, 4($2)
```

Machine Language Program

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

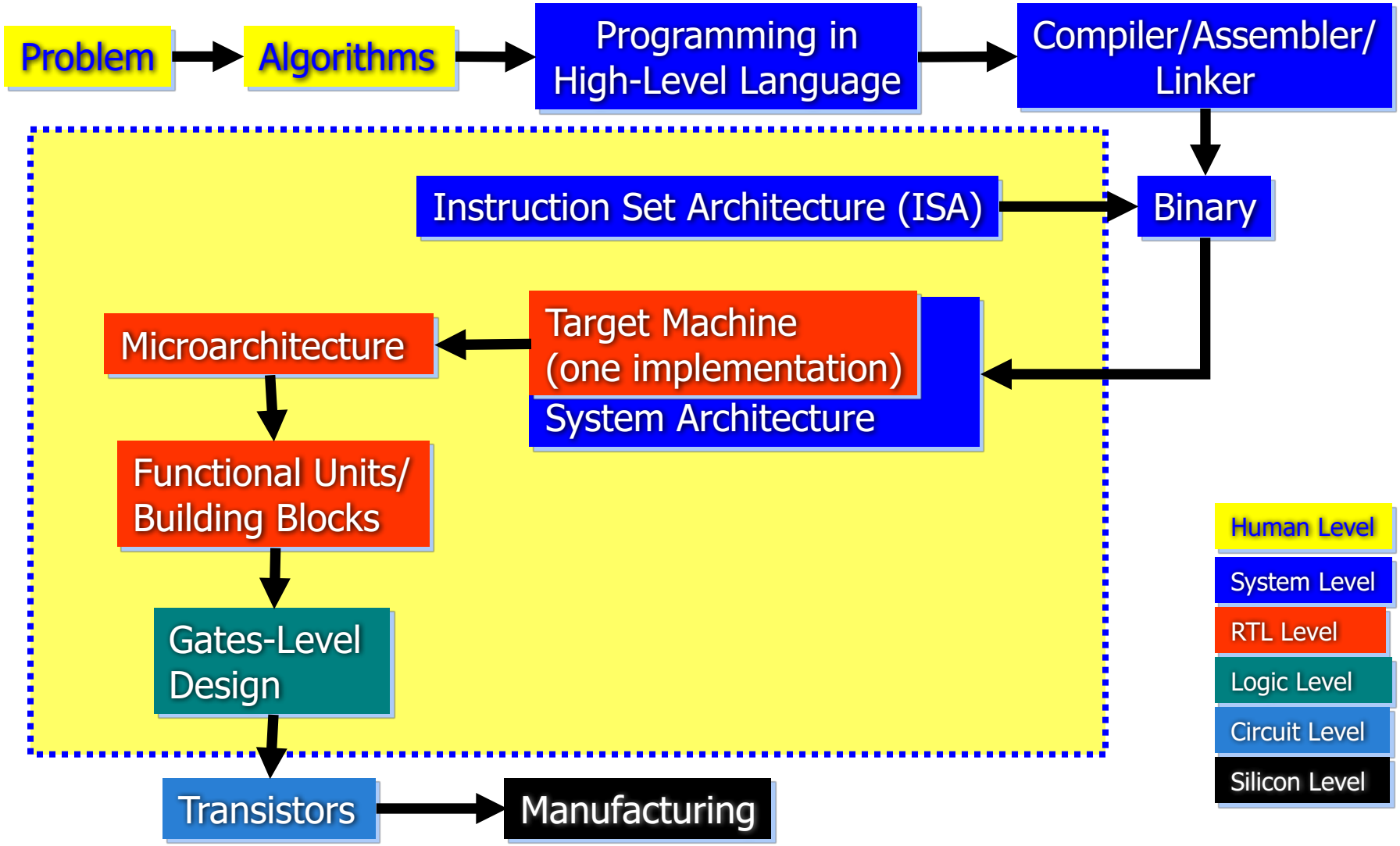
*Machine Interpretation*

Control Signal Specification

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```

°
°

# Hierarchy of Computation



# Representations in Hardware

Truth Table

| Input X | Input Y | Output Ci | Output S | Output Co |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

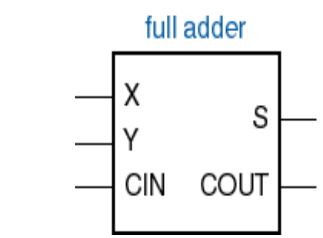
Boolean Equations

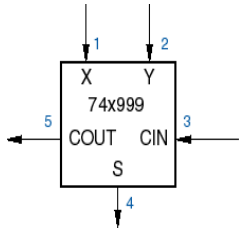
$$S = X'Y'C_{IN} + X'YC_{IN}' + XY'C_{IN}' + XYC_{IN}$$
$$S = X \oplus Y \oplus C_{IN}$$
$$C_{OUT} = XY + XC_{IN} + YC_{IN}$$

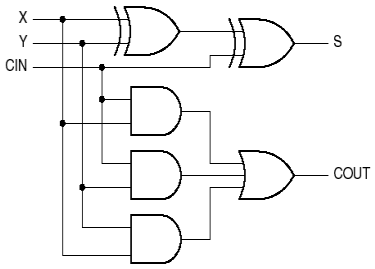
VHDL

```
ARCHITECTURE half_adder_a OF
half_adder IS
   BEGIN
        PROCESS (x, y, enable)
            BEGIN
            IF enable = '1' THEN
                    result <= x XOR y;
                    carry <= x AND y;
            ELSE
                    carry <= '0';
                    result <= '0';
            END IF;
            END PROCESS;
  END half_adder_a;
```

## Logic Schematics

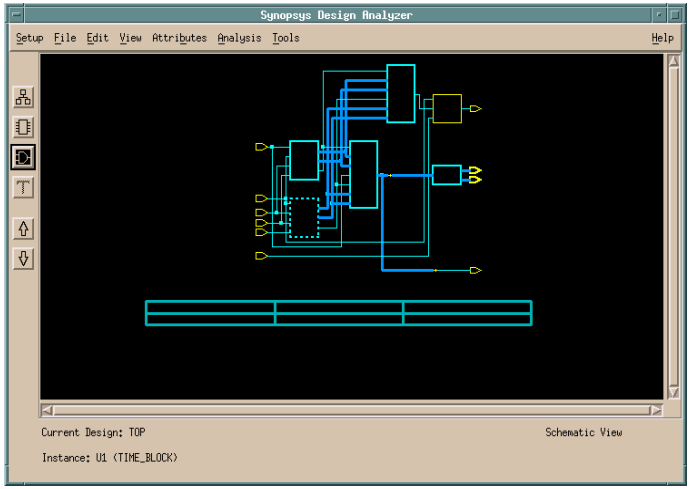


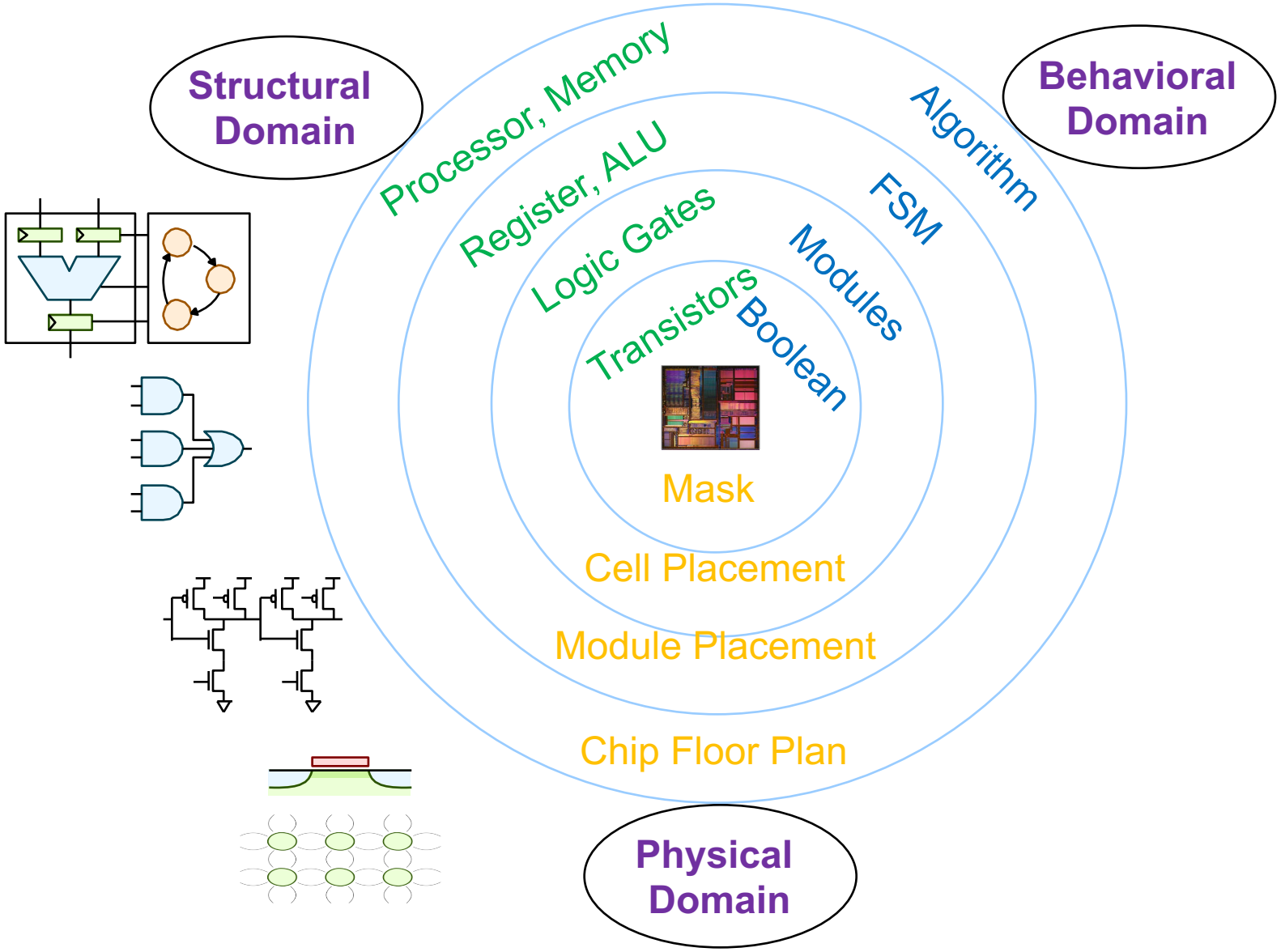

## Logic Gates

- AND
- OR
- XOR




Electronic Design Automation (EDA) Tool

# Systems at the Crossroads



# Methodology Is the Key!

- Hardware
  - Processors
- Software
  - Processes

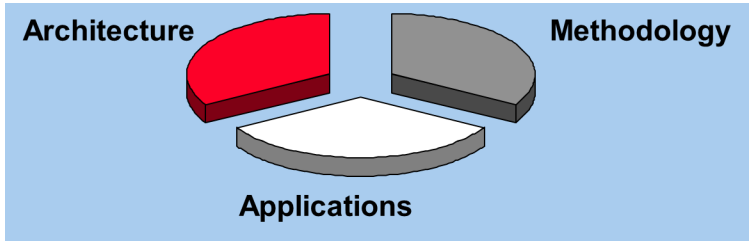
Architecture 

- Requirements
- Specifications
- Platforms

Applications 

- Modeling
- Analysis
- Synthesis
- Verification

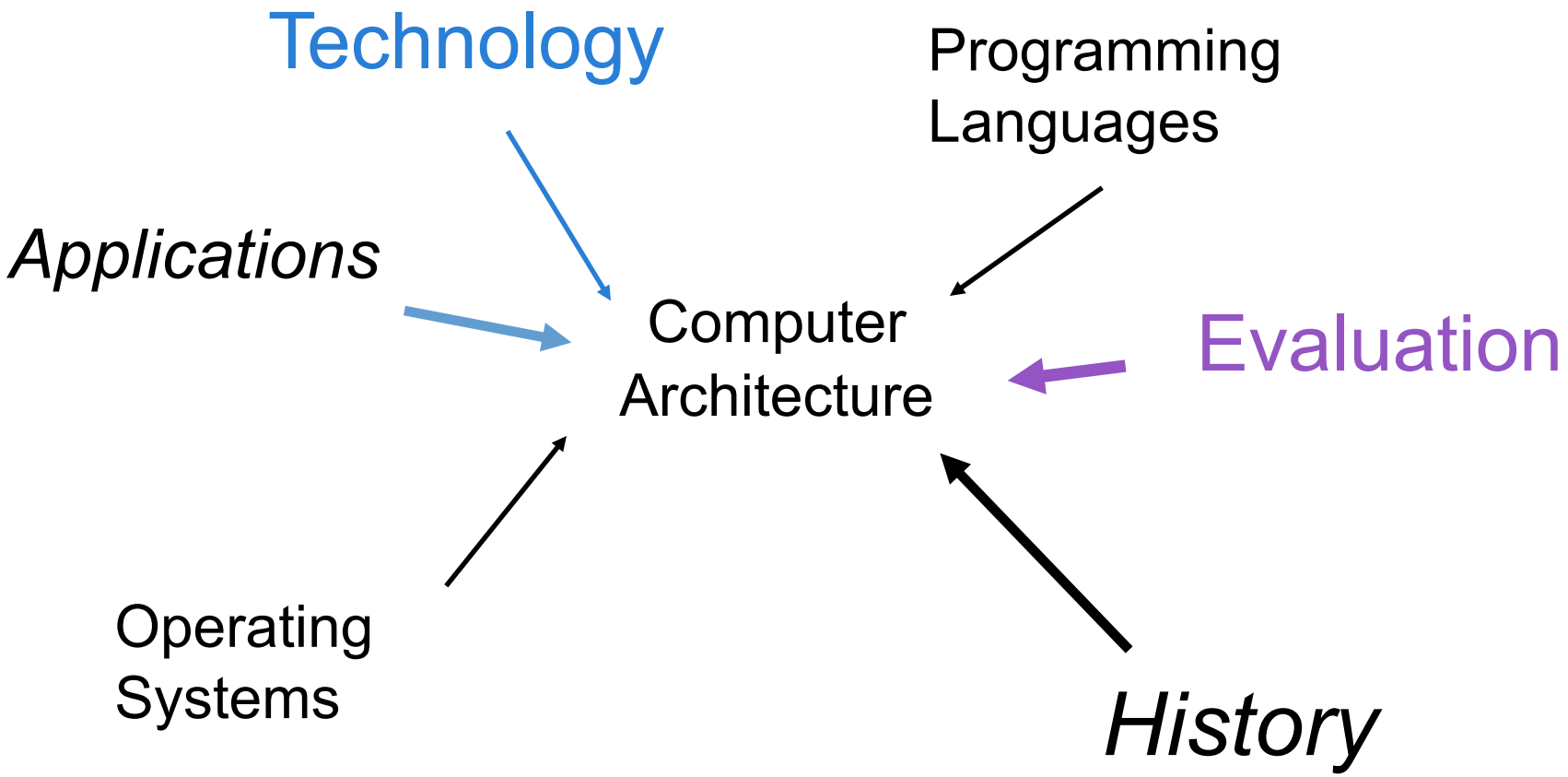
Methodologies 



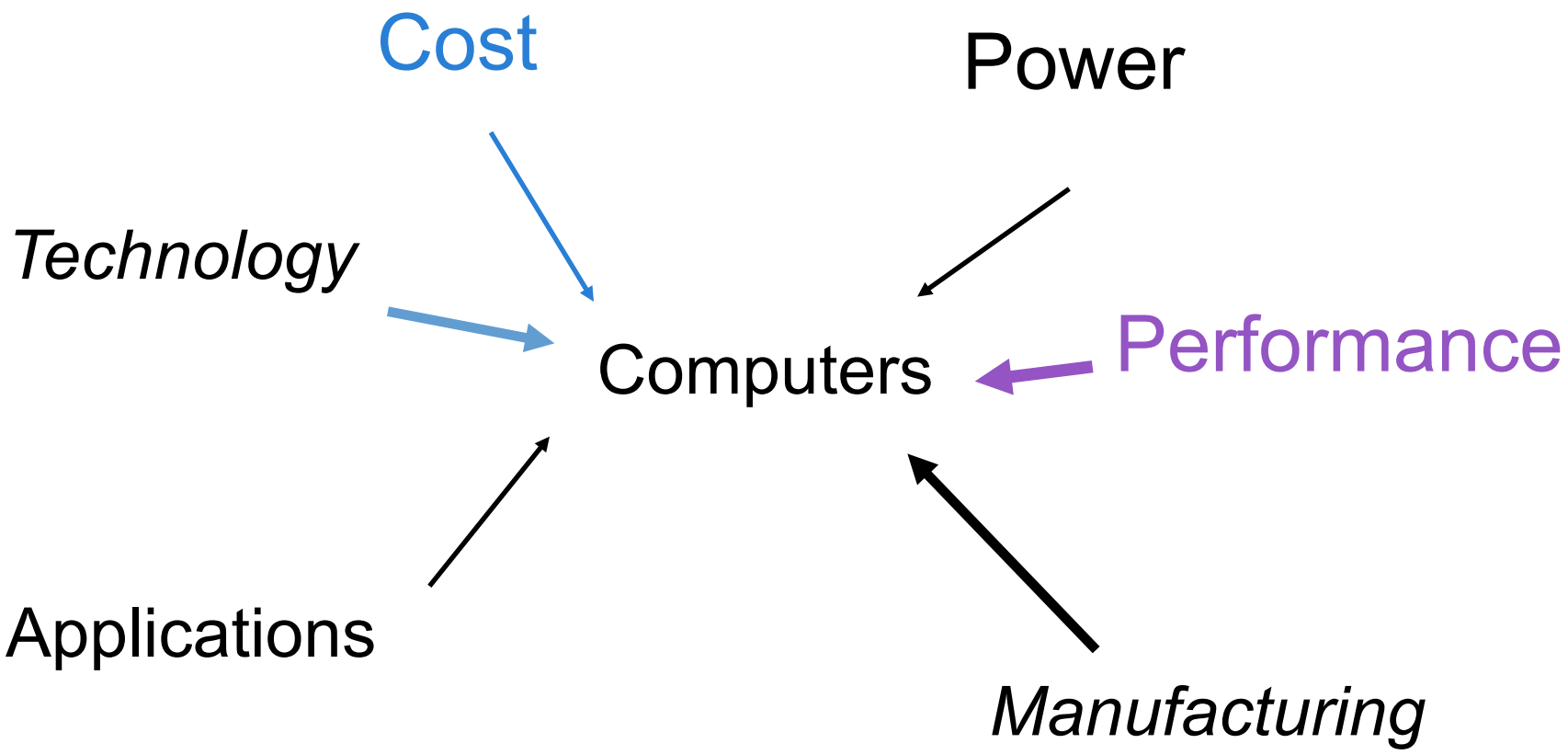
# Models to Architectures

- Models
  - Conceptual views of system functionality
  - A set of objects and rules for choosing the objects
- Architectures
  - Abstract views of system implementation
  - A set of implementation components and their connections
- Computer Architecture
  - The process of building computing systems to meet given application requirements within physical technology constraints

# Design Forces on CA



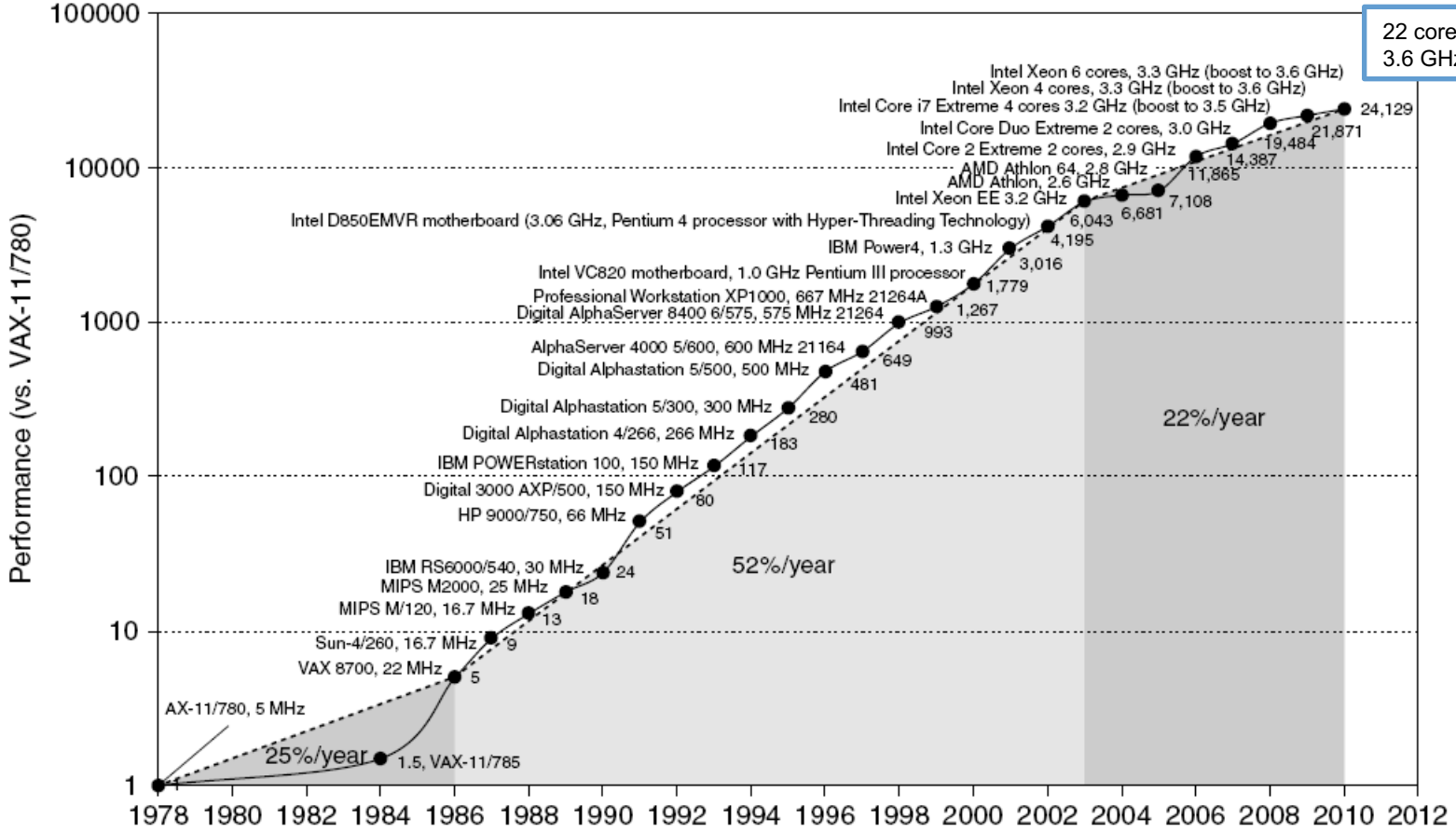
# Implementation Forces on CA



# Computer Technology

- Performance improvements
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by HLL compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

# Single-Processor Performance




22 core Xeon Broadwell E5, 3.6 GHz, 145W, $4K, **2016**

# Current Trends in Architecture

- Cannot continue to leverage instruction-level parallelism (ILP).
  - Single-processor performance improvement ended in 2003.
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application.

ENGINEERING@SYRACUSE

# Classes of Computers

# Classes of Computers

- Personal mobile device (PMD)
  - E.g., smartphones, tablet computers
  - Emphasis on energy efficiency and real time
- Desktop computing
  - Emphasis on price performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters/warehouse-scale computers
  - Used for “Software as a Service (SaaS)”
  - Emphasis on availability and price performance
  - Subclass: supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded computers
  - Emphasis: price

# Parallelism

- Classes of parallelism in applications
  - Data-level parallelism (DLP)
  - Task-level parallelism (TLP)

- Classes of architectural parallelism
  - Instruction-level parallelism (ILP)
  - Vector architectures/graphic processor units (GPUs)
  - Thread-level parallelism
  - Request-level parallelism

# Flynn's Taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly coupled MIMD
  - Loosely coupled MIMD

# Defining Computer Architecture

- "Old" view of computer architecture
  - Instruction set architecture (ISA) design
  - I.e. decisions regarding:
    - Registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding
- "Real" computer architecture
  - Specific requirements of the target machine
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware

ENGINEERING@SYRACUSE

# Trends in Technology

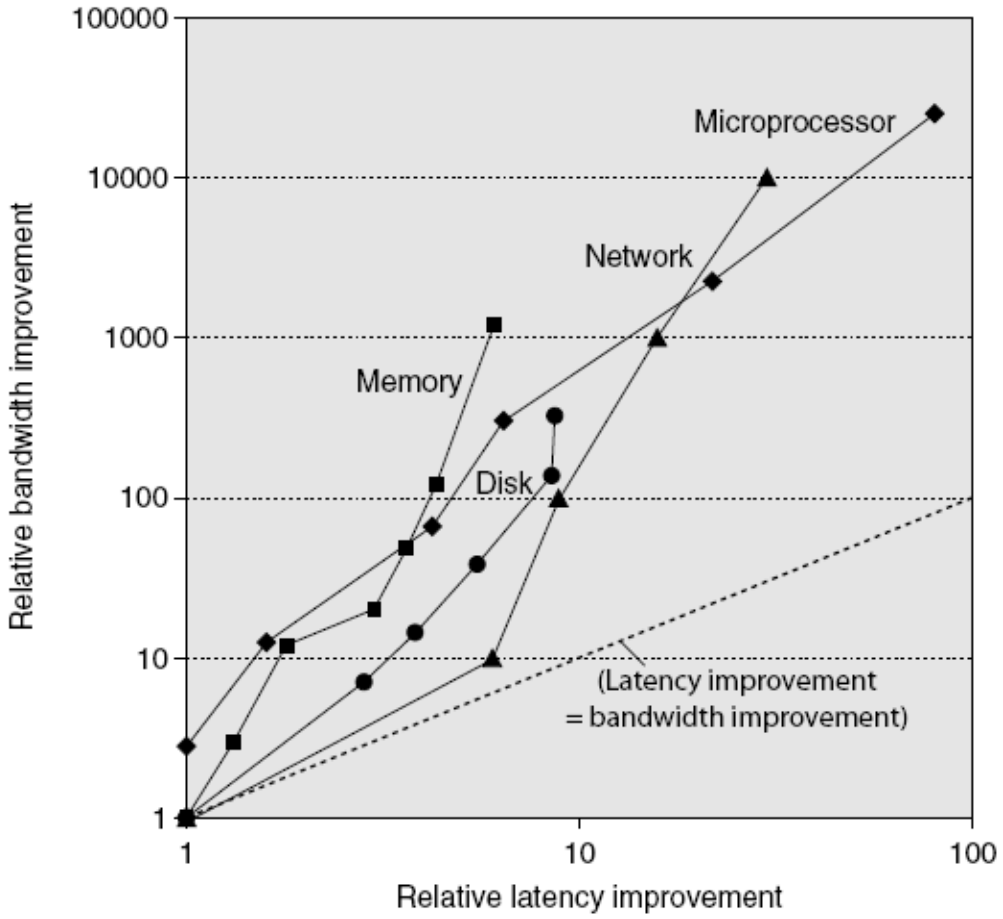
# Trends in Technology

- Integrated circuit technology
  - Transistor density:  35 percent/year
  - Die size:  10–20 percent/year
  - Integration overall:  40–55 percent/year
- DRAM capacity:  25–40 percent/year (slowing)
- Flash capacity:  50–60 percent/year
  - 15–20X cheaper/bit than DRAM
- Magnetic disk technology:  40 percent/year
  - 15–25X cheaper/bit than flash
  - 300–500X cheaper/bit than DRAM

# Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
  - 10,000–25,000X improvement for processors
  - 300–1200X improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
  - 30–80X improvement for processors
  - 6–8X improvement for memory and disks

# Bandwidth and Latency (cont.)




Bandwidth improves by more than the square of the improvement in latency.

# Transistors and Wires

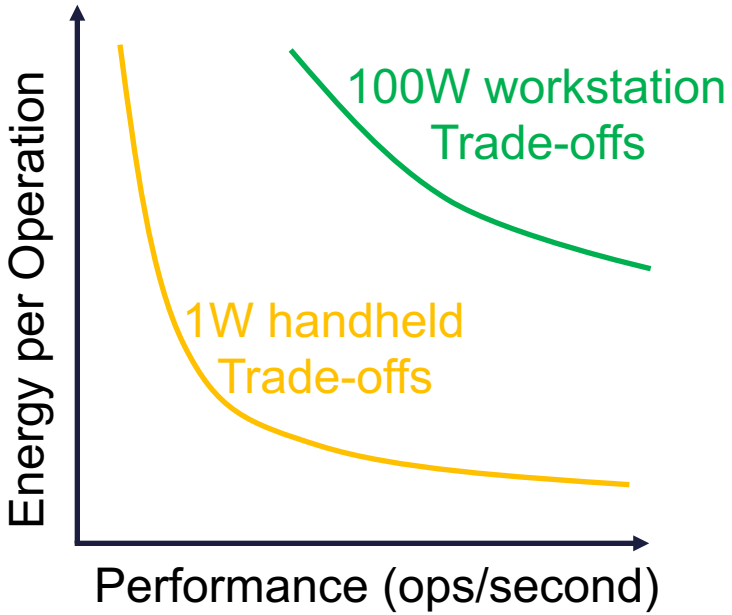
- Feature size
  - Minimum size of transistor or wire in x or y dimension.
  - 10 microns in 1971 to .01 microns (10 nm) in 2017.
    - 5 nm in 2021
  - Transistor performance scales linearly.
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically.
    - 2,300 in 1971 to 30,000,000,000 in 2017

ENGINEERING@SYRACUSE

# Trends in Energy, Power, and Cost

# Power and Energy

- Power = energy/second = energy/op x ops/seconds
- Power
  - Chip packaging
  - System noise
  - Chip and system cooling
- Energy
  - Joules per task
  - Battery life
  - Electricity bill
  - Device weight



# Power and Energy (cont.)

- Problem:  Get power in, get power out.
- Thermal design power (TDP):
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption.
- Energy per task is often a better measurement.

# Example

- Suppose a 15 percent reduction in voltage results in a 15 percent reduction in frequency. What is the impact on dynamic power?

$$
\begin{aligned}
Power_{dynamic} &= 1/2 \cdot CapacitiveLoad \cdot Voltage^2 \cdot FrequencySwitched \\
&= 1/2 \cdot .85 \cdot CapacitiveLoad \cdot (.85 \cdot Voltage)^2 \cdot FrequencySwitched \\
&= (.85)^3 \cdot OldPower_{dynamic} \\
&\approx 0.6 \cdot OldPower_{dynamic}
\end{aligned}
$$

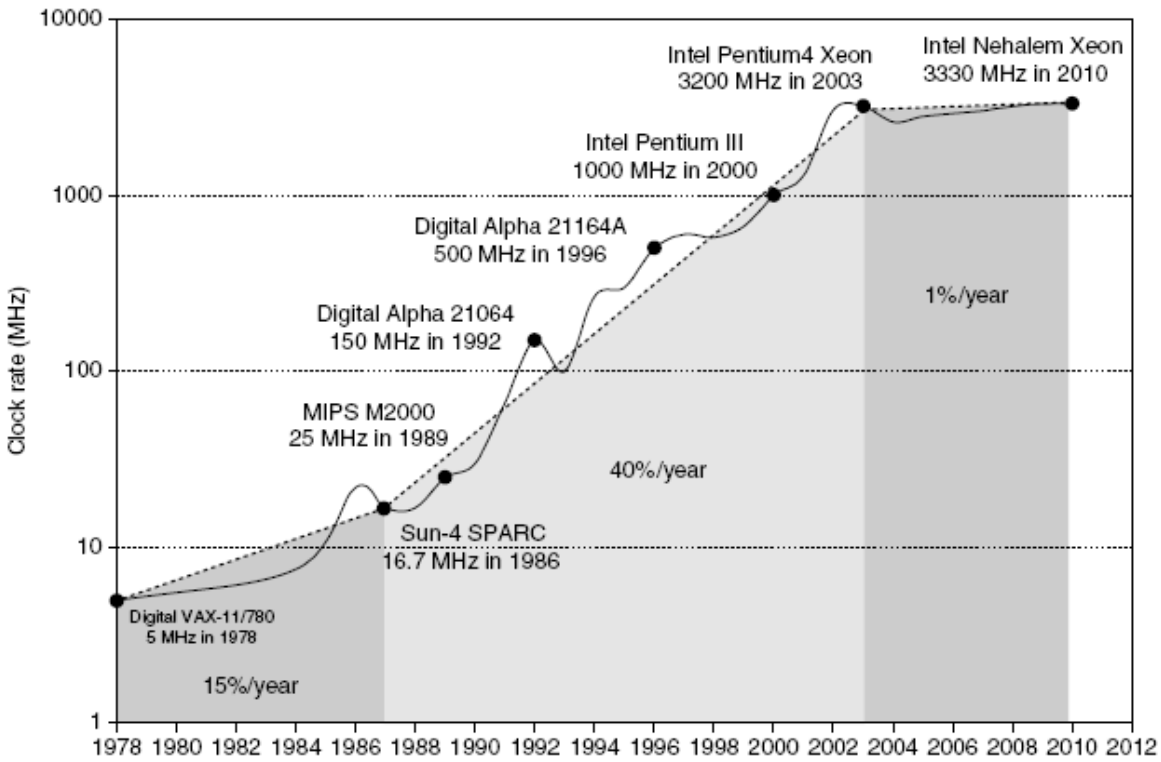
# Dynamic Energy and Power

- Dynamic energy:
  - Transistor switch from 0 → 1 or 1 → 0
  - ½ x capacitive load x voltage$^2$
- Dynamic power:
  - ½ x capacitive load x voltage$^2$ x frequency switched
- Reducing clock rate reduces power, not energy.

# Power

- Intel 80386 consumed ~ 2 W.
- 3.3 GHz Intel Core i7 consumes 130 W.
- Heat must be dissipated from 1.5 x 1.5 cm chip.
- This is the limit of what can be cooled by air.

22 core Xeon Broadwell E5, 3.6 GHz, 145W, **2016**



# Reducing Power

- Techniques for reducing power:
  - Do nothing well.
  - Dynamic voltage-frequency scaling
  - Low power state for DRAM, disks.
  - Overclocking, turning off cores.

# Static Power

- Because leakage current flows even when a transistor is off, now static power is important too.
  - $Current_{static}$ x voltage.
  - Scales with number of transistors.
  - Leakage current increases in processors with smaller transistor sizes.
  - Goal for leakage is 35 percent of total power consumption.
- To reduce: power gating.

# Trends in Cost

- Cost driven down by learning curve.
  - Yield
- DRAM: Price closely tracks cost.
- Microprocessors: Price depends on volume.
  - 10 percent less for each doubling of volume

# Integrated Circuit Cost

- Integrated circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- Bose-Einstein formula:

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- Defects per unit area = 0.016–0.057 defects per square cm (2010)
- N = process-complexity factor = 11.5–15.5 (40 nm, 2010)

ENGINEERING@SYRACUSE

# Measuring Performance and Dependability

# Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput
- Speedup of X relative to Y
  - Execution time$_Y$/execution time$_X$
- Execution time
  - Wall clock time:  includes all system overheads
  - CPU time:  only computation time
- Benchmarks
  - Kernels (e.g., matrix multiply)
  - Toy programs (e.g., sorting)
  - Synthetic benchmarks (e.g., Dhrystone)
  - Benchmark suites (e.g., SPEC06fp, TPC-C)

# Dependability

- How to decide when a system is operating properly?
- Infrastructure providers now offer service-level agreements (SLA) to guarantee that their networking or power service would be dependable.
- Systems alternate between two states of service with respect to an SLA:
  1. Service accomplishment, where the service is delivered as specified in SLA
  2. Service interruption, where the delivered service is different from the SLA
- Failure = transition from state 1 to state 2.
- Restoration = transition from state 2 to state 1.

# Dependability Metrics

- Module reliability = measure of continuous service accomplishment (or time to failure).
  - Mean time to failure (MTTF) measures reliability.
  - Failures in time (FIT) = 1/MTTF, the rate of failures.
    - Traditionally reported as failures per billion hours of operation
  - Mean time to repair (MTTR) measures service interruption.

- Mean time between failures (MTBF) = MTTF + MTTR

- Module availability measures service as alternate between the two states of accomplishment and interruption (number between 0 and 1, such as 0.9).
  - Module availability = MTTF / ( MTTF + MTTR)

# Example

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules.
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), one disk controller (0.5M hour MTTF), and one power supply (0.2M hour MTTF):

$$
\begin{aligned}
FailureRate &= 10 \cdot (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\
&= 10 + 2 + 5/1{,}000{,}000 \\
&= 17/1{,}000{,}000 \\
&= 17{,}000 FIT \\
MTTF &= 1{,}000{,}000{,}000/17{,}000 \\
&\quad 59{,}000 hours
\end{aligned}
$$

ENGINEERING@SYRACUSE

# Principles

# Principles of Computer Design

- Take advantage of parallelism.
  - E.g., multiple processors, disks, memory banks, pipelining, multiple functional units
- Principle of locality.
  - Reuse of data and instructions
- Focus on the common case.
  - Amdahl's law

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left( (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

# Principles of Computer Design (cont.)

- The processor performance equation

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

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

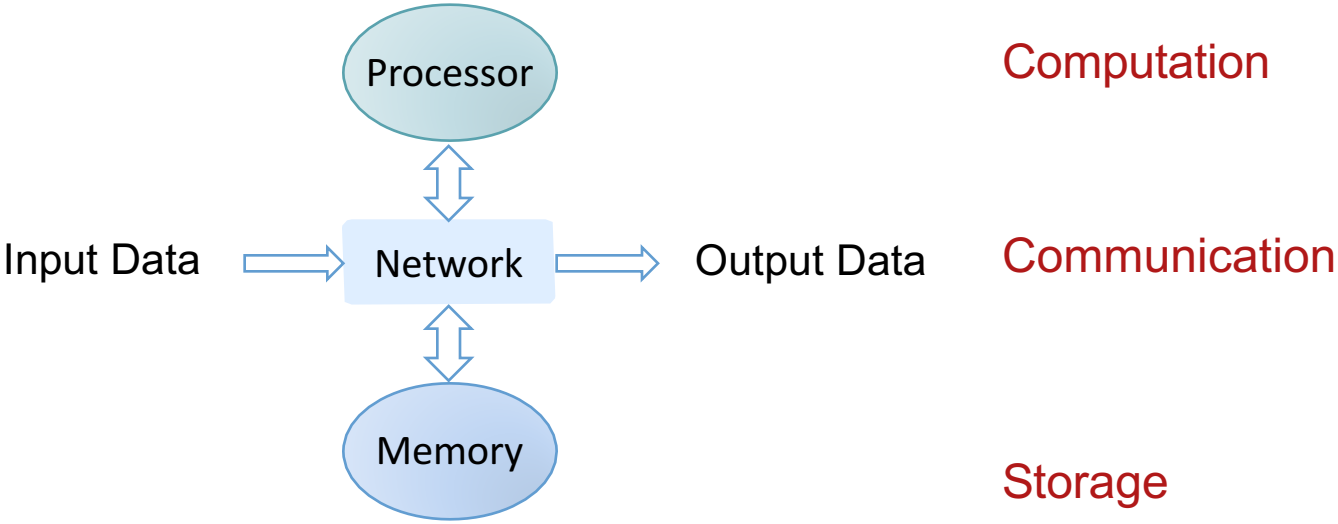
# Principles of Computer Design (cont.)

- Different instruction types having different CPIs

$$\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left( \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

# Three Building Blocks




- Trends
  - Growing diversity in application requirements.
  - Energy and power constrain systems.
  - Multiple cores integrated onto a single chip.
  - Heterogeneous systems-on-chip.
  - New device technologies for three basic blocks.

ENGINEERING@SYRACUSE

# Crossroads

# Crossroads

- Old: Power is free; transistors expensive.
  - New: “Power wall”—power expensive; transistors free.
- Old: Instruction-level parallelism (ILP) via compilers.
  - New: “ILP wall” law of diminishing returns on more HW for ILP.
- Old: Multiplies are slow; memory access is fast.
  - New: “Memory wall”—memory slow, multiplies fast.
    (200 clock cycles to DRAM memory, four clocks for multiply)
- Old: Uniprocessor performance 2X/1.5 years.
  - New: Power wall + ILP wall + memory wall = brick wall.
  - Uniprocessor performance now 2X / 5(?) years.

⇒Sea change in chip design: multiple “cores.”

- 2X processors per chip / around two years

⇒More simpler processors are more power efficient.

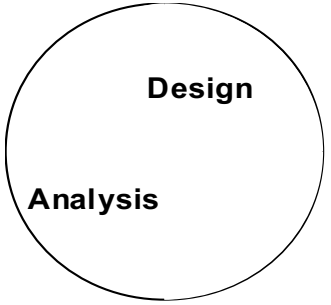
# Problems with Sea Change

- Algorithms
- Programming languages
- Compilers
- Operating systems
- Architectures
- Libraries

⇒To supply thread-level parallelism or data-level parallelism for 1,000 CPUs / chip.

# CA Is an Integrated Approach

- What really matters is the functioning of the complete system.
  - Hardware, runtime system, compiler, operating system, and application
- Computer architecture is not just about transistors, individual instructions, or particular implementations; it is an iterative process.
  - Searching the space of possible designs at all levels of computer systems ➔ trade-offs



# Computer Architecture

- Other fields often borrow ideas from architecture.
- Quantitative principles of design:
  1. Take advantage of parallelism. (pipelining, …)
  2. Principle of locality. (cache, …)
  3. Focus on the common case. (faster I/O, …)
  4. Amdahl's law. (bottlenecks, …)
  5. The processor performance equation. (IC*CPI*CCT, …)
- Careful, quantitative comparisons: Define and quantify.
  - Relative performance ($C \times V^2 \times f$, …)
  - Relative cost
  - Dependability (MTTF, availability, …)
  - Define and quantify power (ratios, mean, …)
- Culture of anticipating and exploiting advances in technology.
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked.

ENGINEERING@SYRACUSE

# Conclusions

# In Conclusion

- An exciting new era of computer architecture with growing diversity in applications and systems from mainstream parallel processing to SoCs.
- Computer architecture skill sets are different.
  - Quantitative principles of design
    - Parallelism, locality, common case, bottlenecks, equations
  - Quantitative approach to design
    - Performance, cost, dependability, power
  - Solid interfaces that really work
  - Technology tracking and anticipation

ENGINEERING@SYRACUSE