Chapter 1

Computer Abstractions and Technology
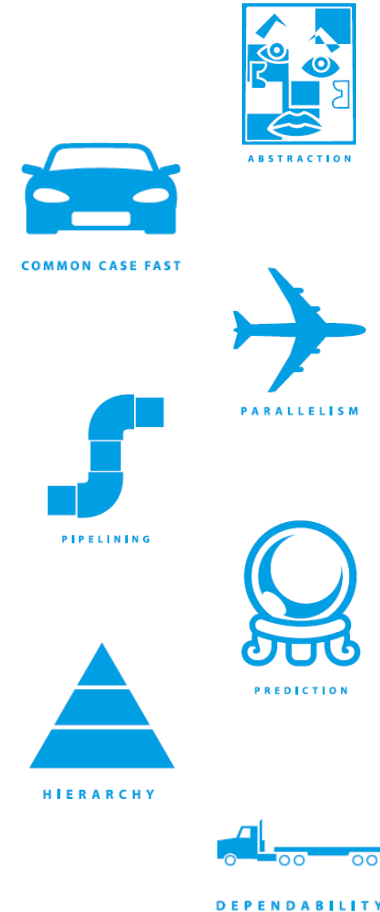
Adapted by Prof. Gheith Abandah

Content

Seven Great Ideas

- 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

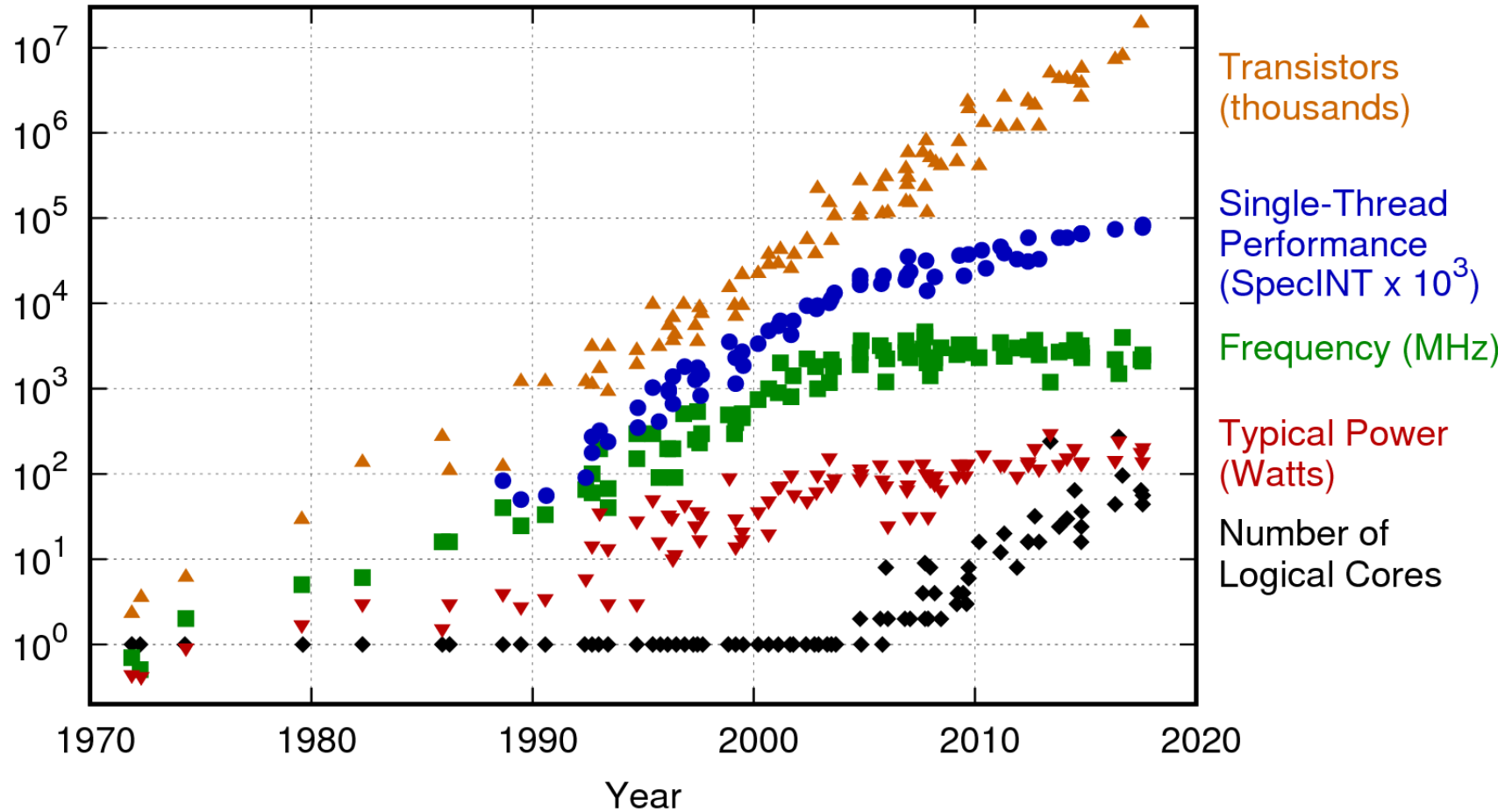ABSTRACTION

COMMON CASE FAST

PARALLELISM

PIPELINING

PREDICTION

HIERARCHY

DEPENDABILITY

Content

Technology Trends

42 Years of Microprocessor Trend Data

Technology Trends

40 Years of Microprocessor Trend Data

$$P_{dynamic} \approx N * C * V^2 f * A$$

Transistors (Thousands)

Single-Thread Performance (SpecINT × 10³)

Frequency (MHz)

Typical Power (W)

Number of Logical Cores
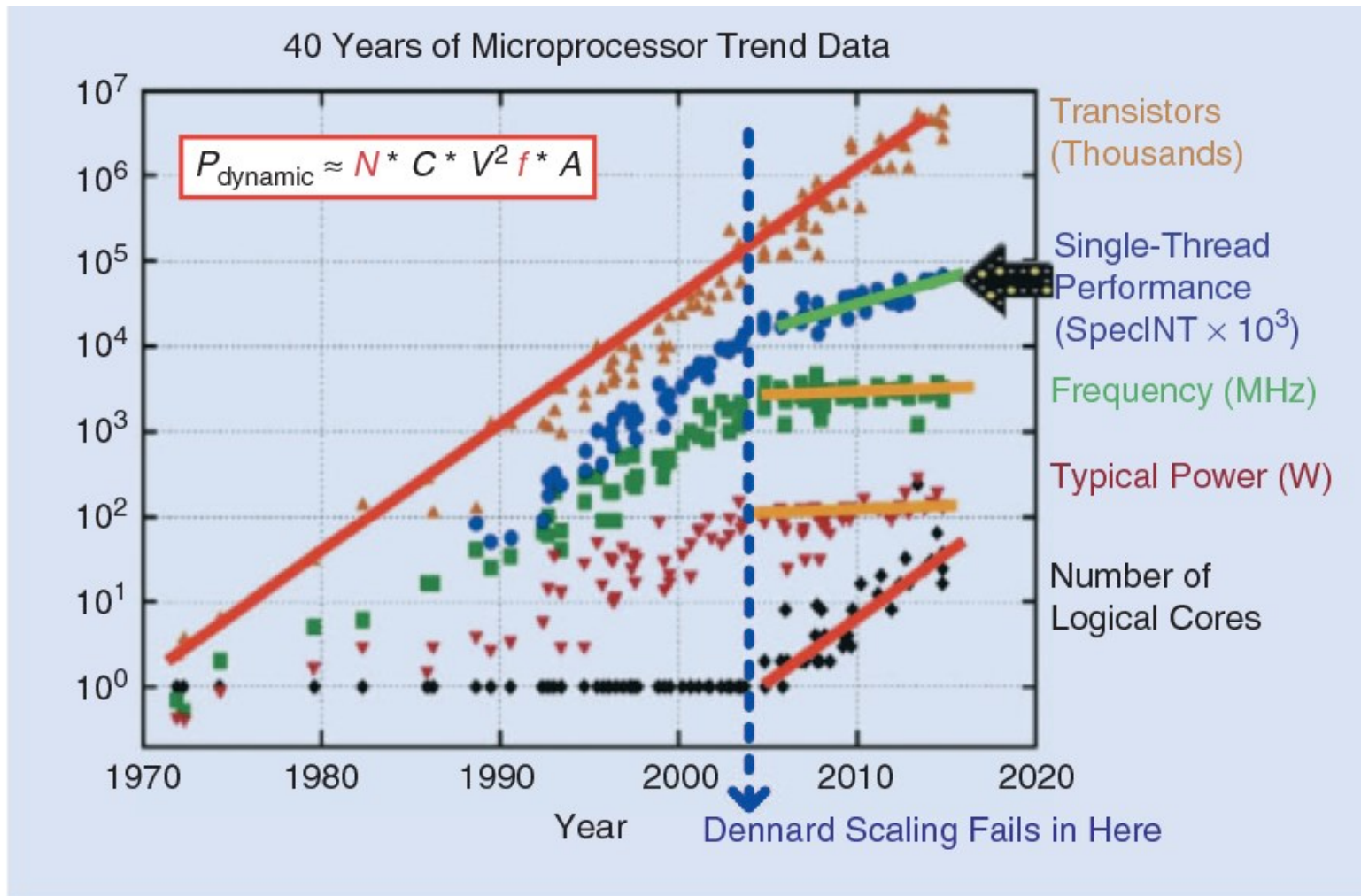
Year

Dennard Scaling Fails in Here

FIGURE 1: The Dennard scaling failed around the middle of the 2000s [24].

Technology Trends

DRAM capacity

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost
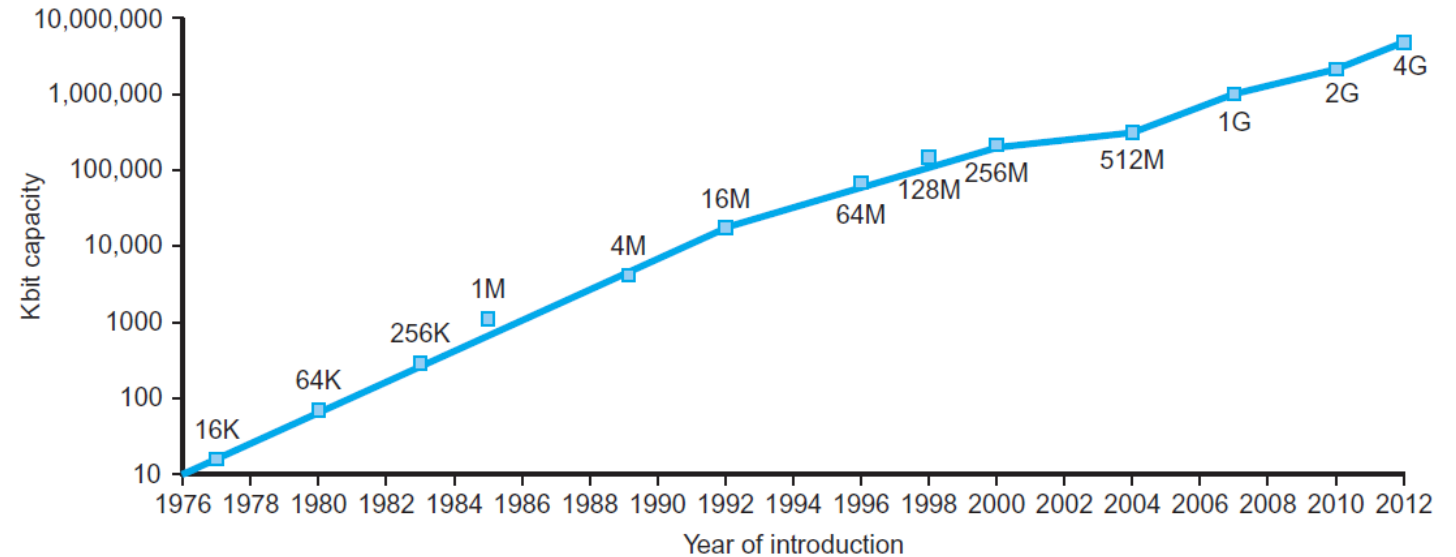
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

- Silicon: semiconductor

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

- YouTube Video: VLSI Fabrication Process

  https://youtu.be/fwNkg1fsqBY

Manufacturing ICs
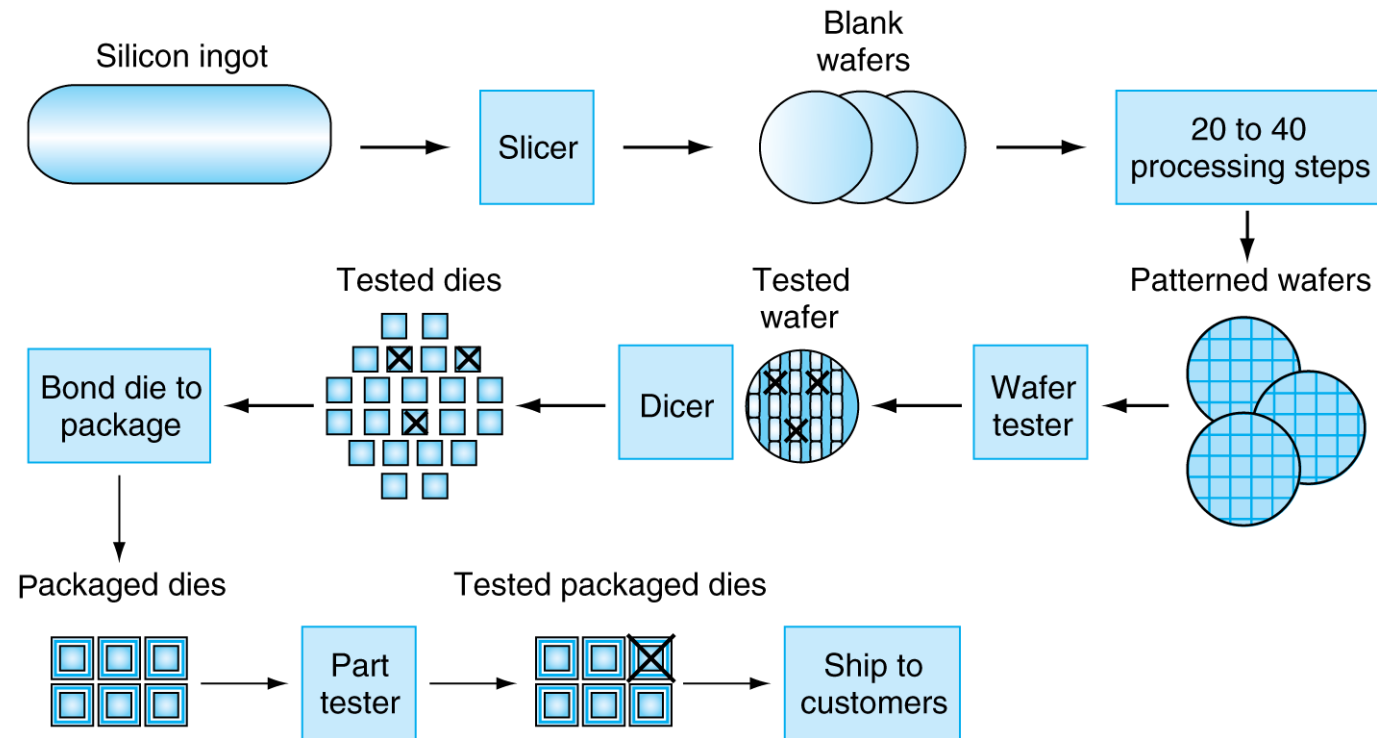
- Yield: proportion of working dies per wafer

Intel Core 10th Gen

- 300mm wafer, 506 chips, 10nm technology
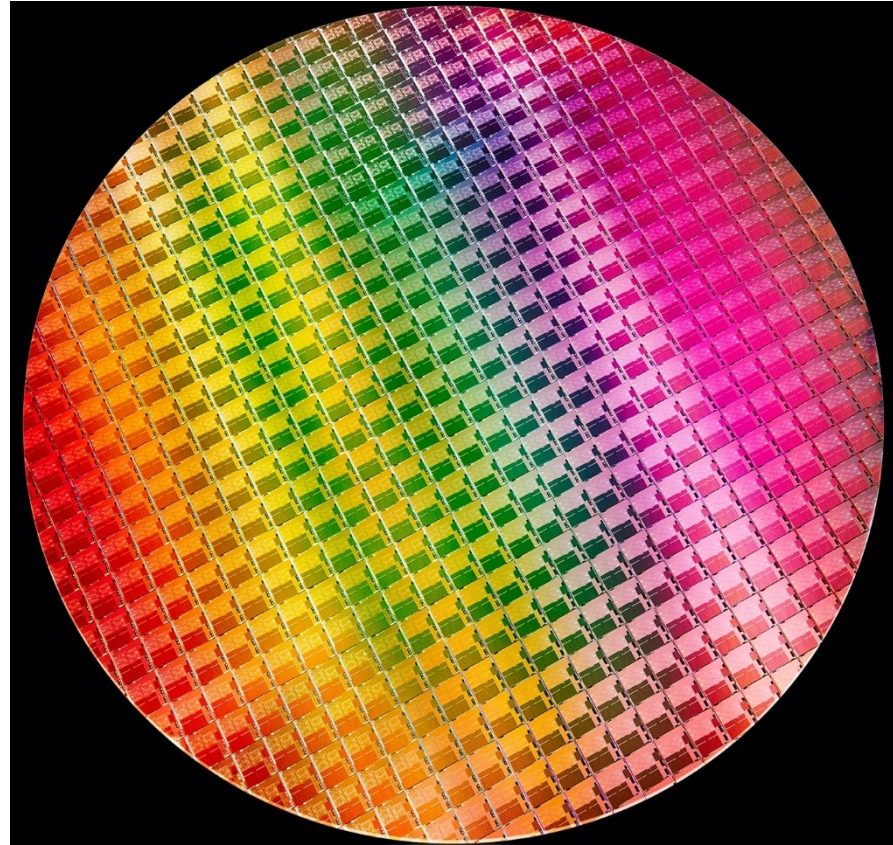- Each chip is 11.4 x 10.7 mm

Integrated Circuit Cost

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

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

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

- 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

Content

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"

$$Performance_X / Performance_Y$$
$$= Execution\ time_Y / Execution\ time_X = n$$

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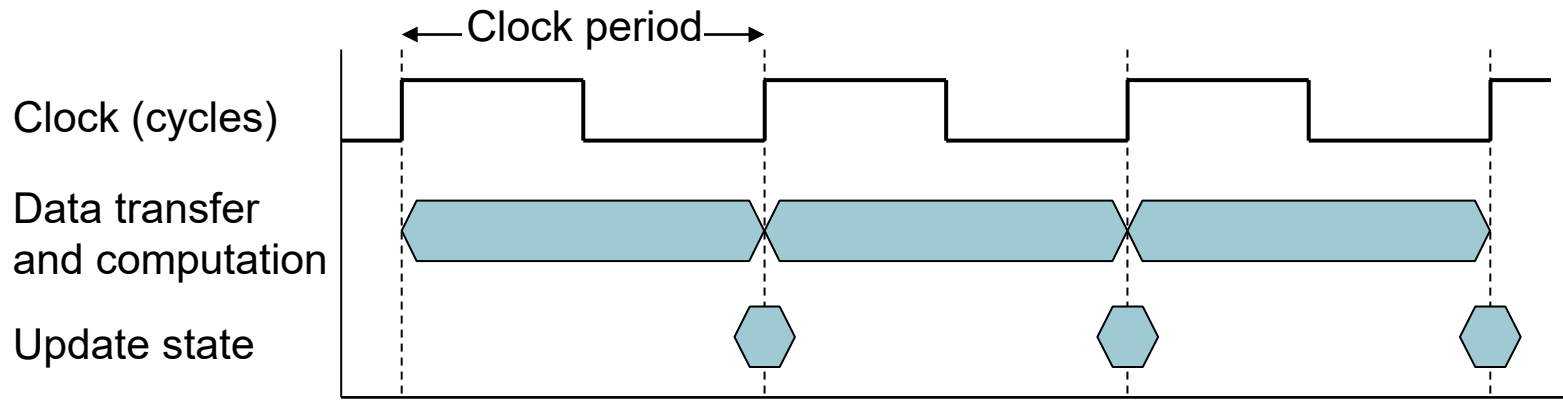
- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^9Hz

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

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 in More Detail

- If different instruction classes take different numbers of cycles

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

- Weighted average CPI

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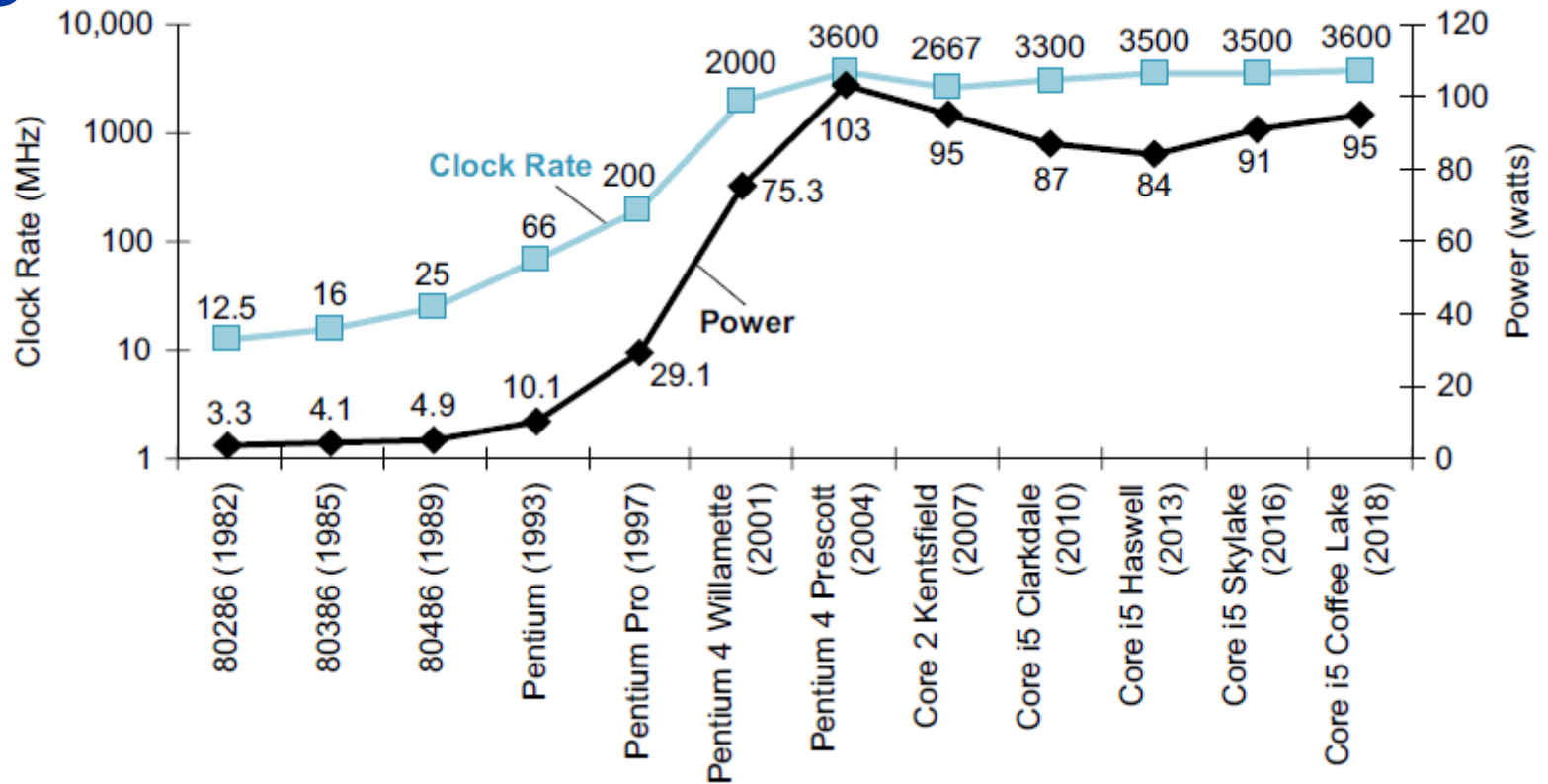
Performance Summary

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

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

Content

Power Trends

- In CMOS IC technology

$$\text{Power} = \frac{1}{2}\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

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 can't reduce voltage further

  - ### We can't remove more heat

- ## How else can we improve performance?

Content

Uniprocessor Performance
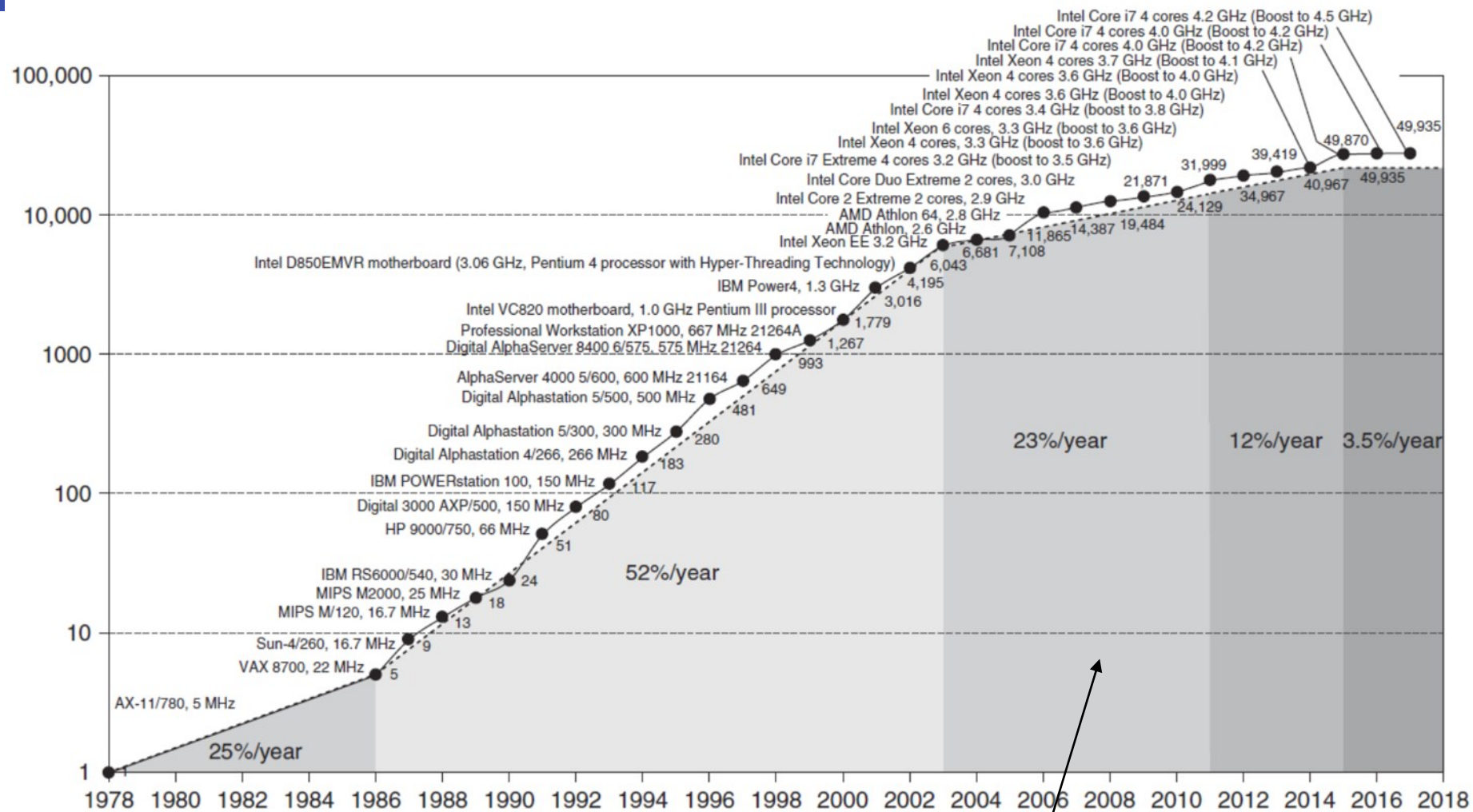
Constrained by power, instruction-level parallelism, memory latency

Uniprocessor Performance

40 years of Processor Performance

CISC 2X / 3.5 yrs (22%/yr)

RISC 2X / 1.5 yrs (52%/yr)

End of Dennard Scaling ⇒ Multicore 2X / 3.5 yrs (23%/yr)

Am-dahl's Law ⇒ 2X / 6 yrs (12%/yr)

End of Moore's Law ⇒ 2X / 20 yrs (3%/yr)
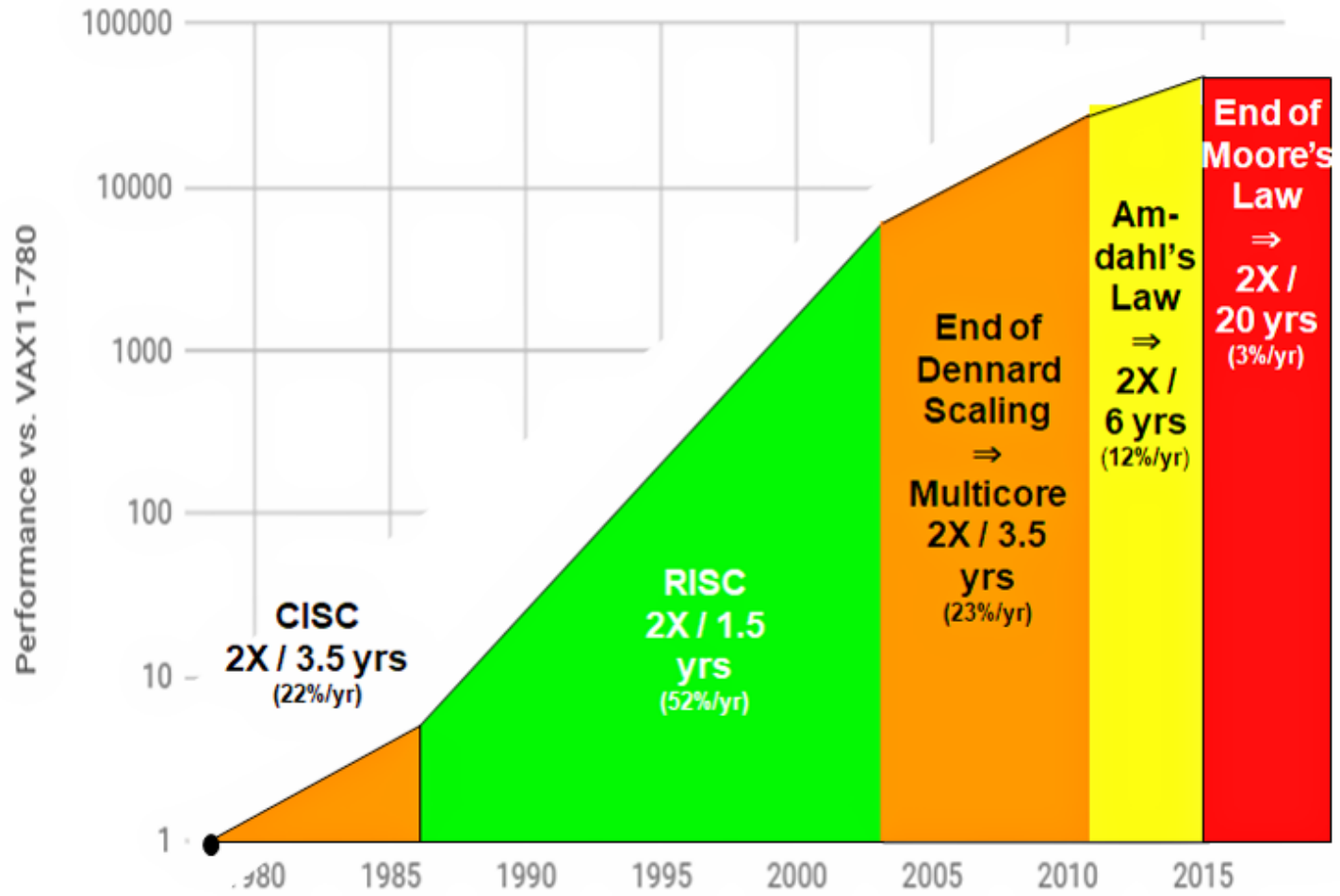
Multiprocessors

- 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

Content

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …

- SPEC CPU2017
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2017 (integer) and CFP2017 (floating-point)

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

SPECspeed 2017 Integer benchmarks on a 1.8 GHz Intel Xeon E5-2650L

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^-9)	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Perl interpreter	perlbench	2684	0.42	0.556	627	1774	2.83
GNU C compiler	gcc	2322	0.67	0.556	863	3976	4.61
Route planning	mcf	1786	1.22	0.556	1215	4721	3.89
Discrete Event simulation - computer network	omnetpp	1107	0.82	0.556	507	1630	3.21
XML to HTML conversion via XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Video compression	x264	4488	0.32	0.556	813	1763	2.17
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Artificial Intelligence: Monte Carlo tree search (Go)	leela	2236	0.79	0.556	987	1703	1.73
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
General data compression	xz	8533	1.32	0.556	6290	6182	0.98
Geometric mean	–	–	–	–	–	–	2.36

SPEC Power Benchmark

- Power consumption of server at different workload levels
    - Performance: ssj_ops/sec
    - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i \right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i \right)$$

SPECpower_ssj2008 for Xeon E5-2650L

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	4,864,136	347
90%	4,389,196	312
80%	3,905,724	278
70%	3,418,737	241
60%	2,925,811	212
50%	2,439,017	183
40%	1,951,394	160
30%	1,461,411	141
20%	974,045	128
10%	485,973	115
0%	0	48
Overall Sum	26,815,444	2,165
$\sum$ssj_ops / $\sum$power =		12,385

Content

Pitfall: Amdahl's Law

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

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

- 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

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

Content

Concluding Remarks

- ## Cost/performance is improving
  - ### Due to underlying technology development
- ## Execution time: the best performance measure
- ## Power is a limiting factor
  - ### Use parallelism to improve performance