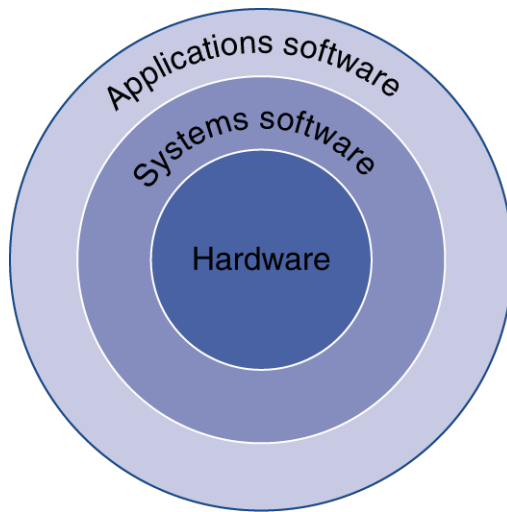# Between Your Program and Hardware

- **Application software**
  - ◆ Written in high-level language (HLL)

- **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

Applications software

Systems software

Hardware

1

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

High-level
language
program
(in C)

```c
swap(int v[], int k)
{int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

Assembly
language
program
(for MIPS)

```
swap:
    muli $2, $5,4
    add  $2, $4,$2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```
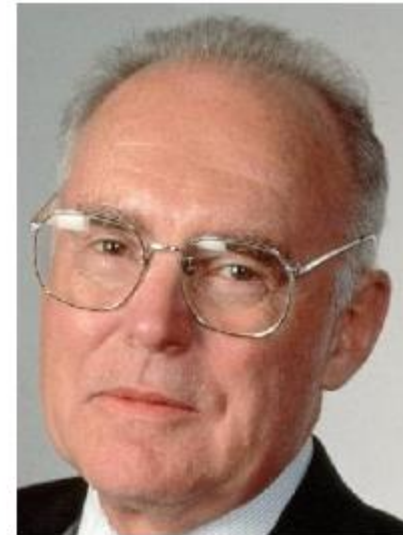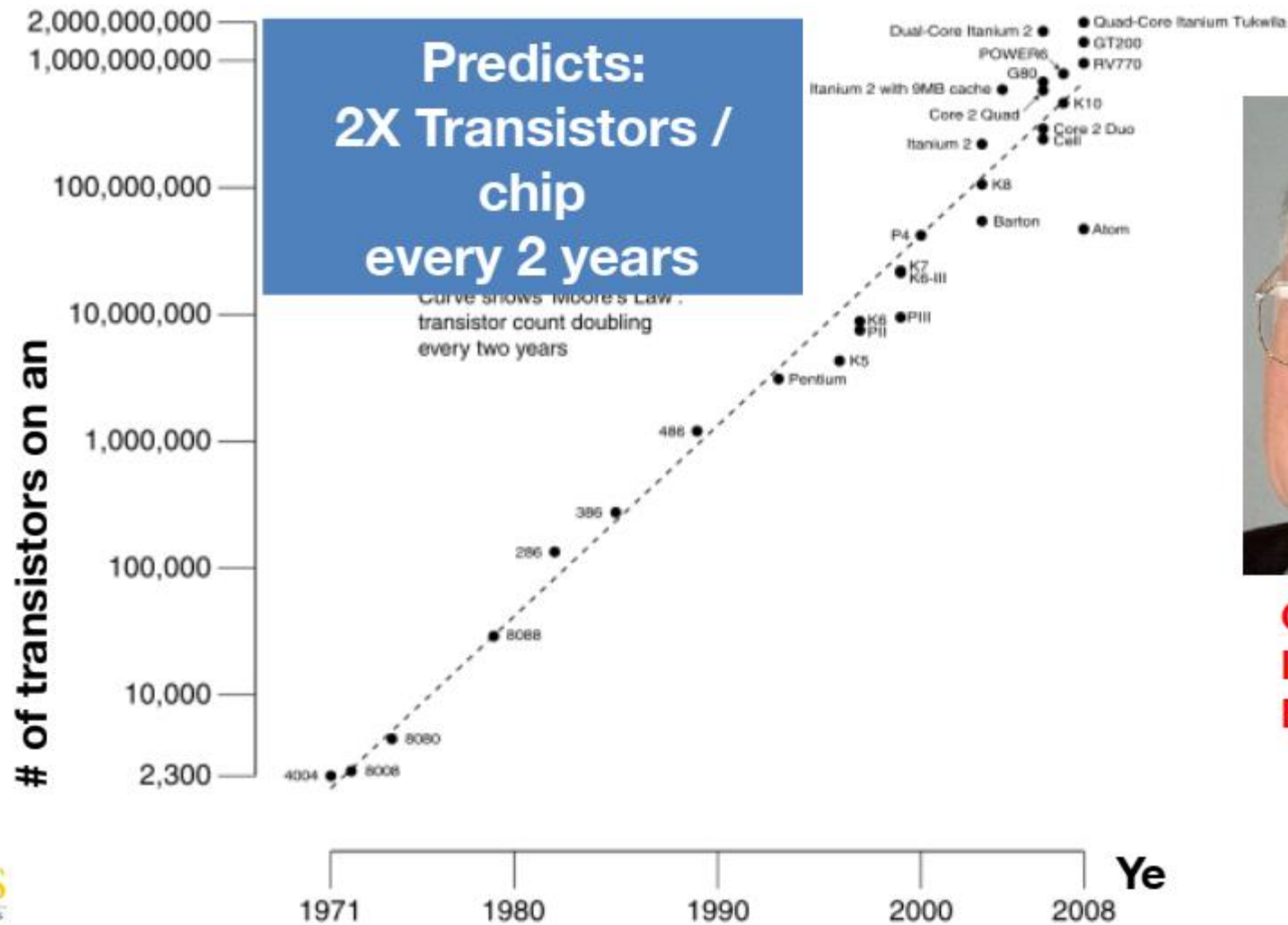
Assembler

Binary machine
language
program
(for MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

2

2

Eight Great Ideas

design for hardware in the future

- 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

MOORE'S LAW

ABSTRACTION

COMMON CASE FAST

PARALLELISM

PIPELINING

PREDICTION

HIERARCHY

DEPENDABILITY

Moore's Law

Predicts:
2X Transistors /
chip
every 2 years

Curve shows Moore's Law:
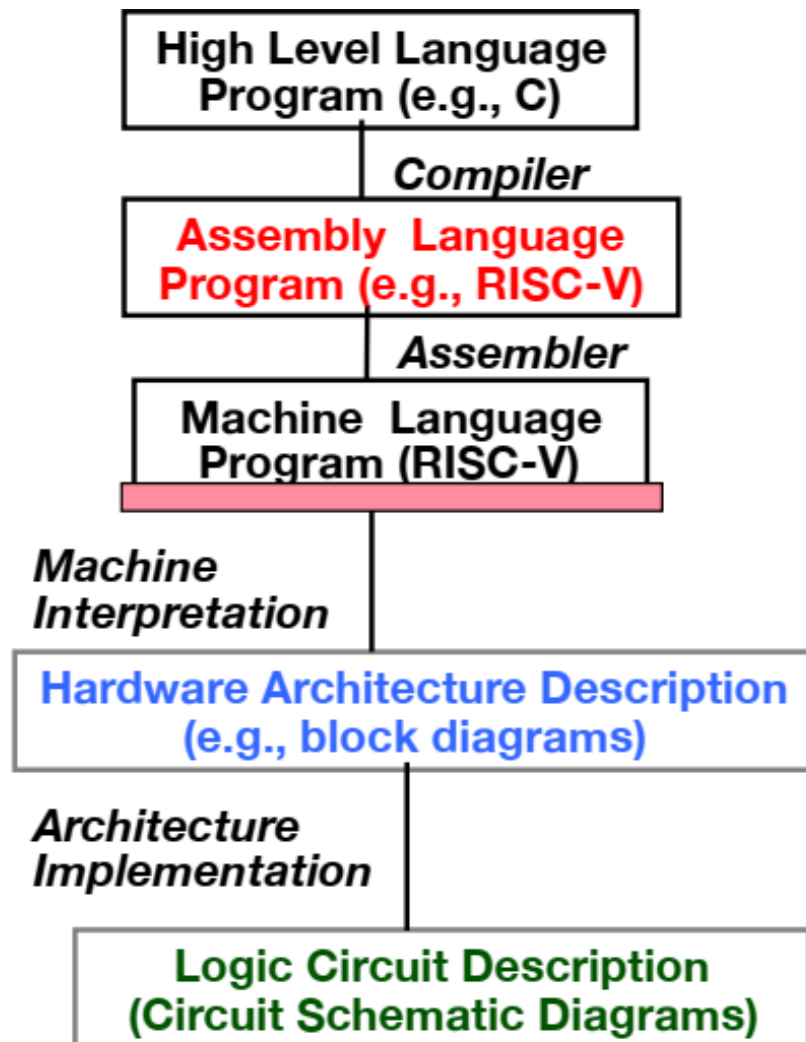transistor count doubling
every two years

**Gordon Moore
Intel Cofounder
B.S. Cal 1950!**

4

Post Moore's Law

- Moore's Law meant that the cost of transistors scaled down as technology scaled to smaller and smaller feature sizes.
  - And the resulting transistors resulted in increased single-task performance
- But single-task performance improvements hit a brick wall years ago...
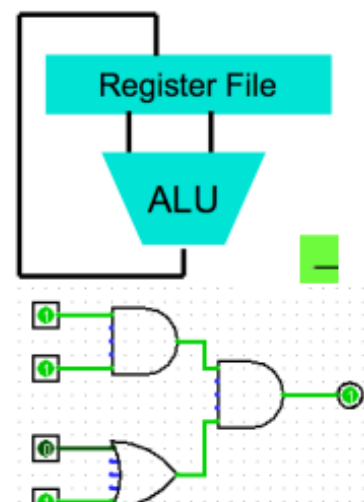- And now the newest, smallest fabrication processes <14nm, might have greater cost/transistor!!!! So, why shrink????

Abstraction

High Level Language Program (e.g., C)

Compiler

Assembly Language Program (e.g., RISC-V)

Assembler

Machine Language Program (RISC-V)

Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)

Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```
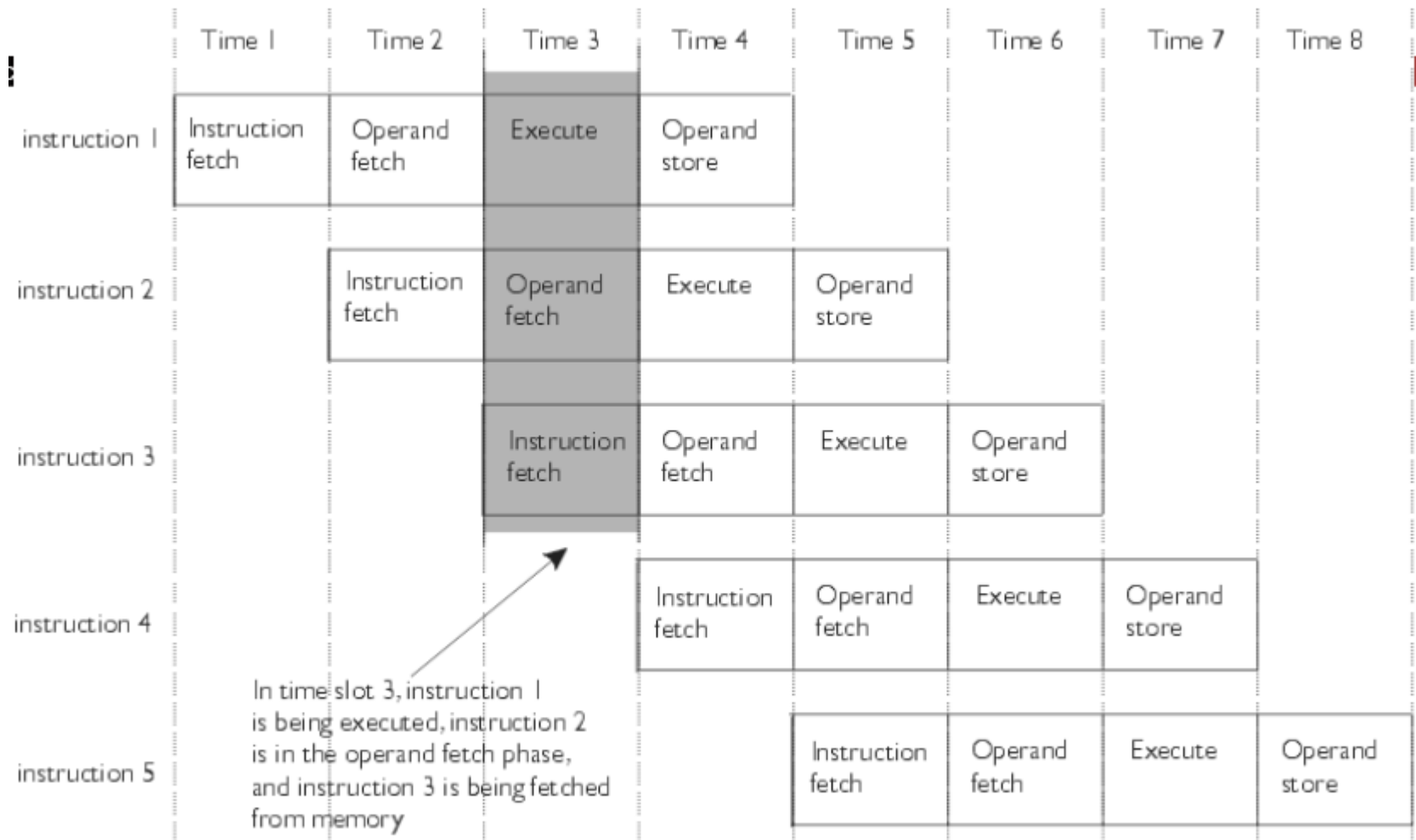
Anything can be represented as a *number*, i.e., data or instructions

```
0000 1001 1100 0110 1010        )0
1010 1111 0101 1000 0000        .0
1100 0110 1010 1111 0101        )1
0101 1000 0000 1001 1100        .1
```

Register File

ALU

Pipeline

no more hardware resources.

	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8
instruction 1	Instruction fetch	Operand fetch	Execute	Operand store				
instruction 2		Instruction fetch	Operand fetch	Execute	Operand store			
instruction 3			Instruction fetch	Operand fetch	Execute	Operand store		
instruction 4				Instruction fetch	Operand fetch	Execute	Operand store	
instruction 5					Instruction fetch	Operand fetch	Execute	Operand store

In time slot 3, instruction 1 is being executed, instruction 2 is in the operand fetch phase, and instruction 3 is being fetched from memory
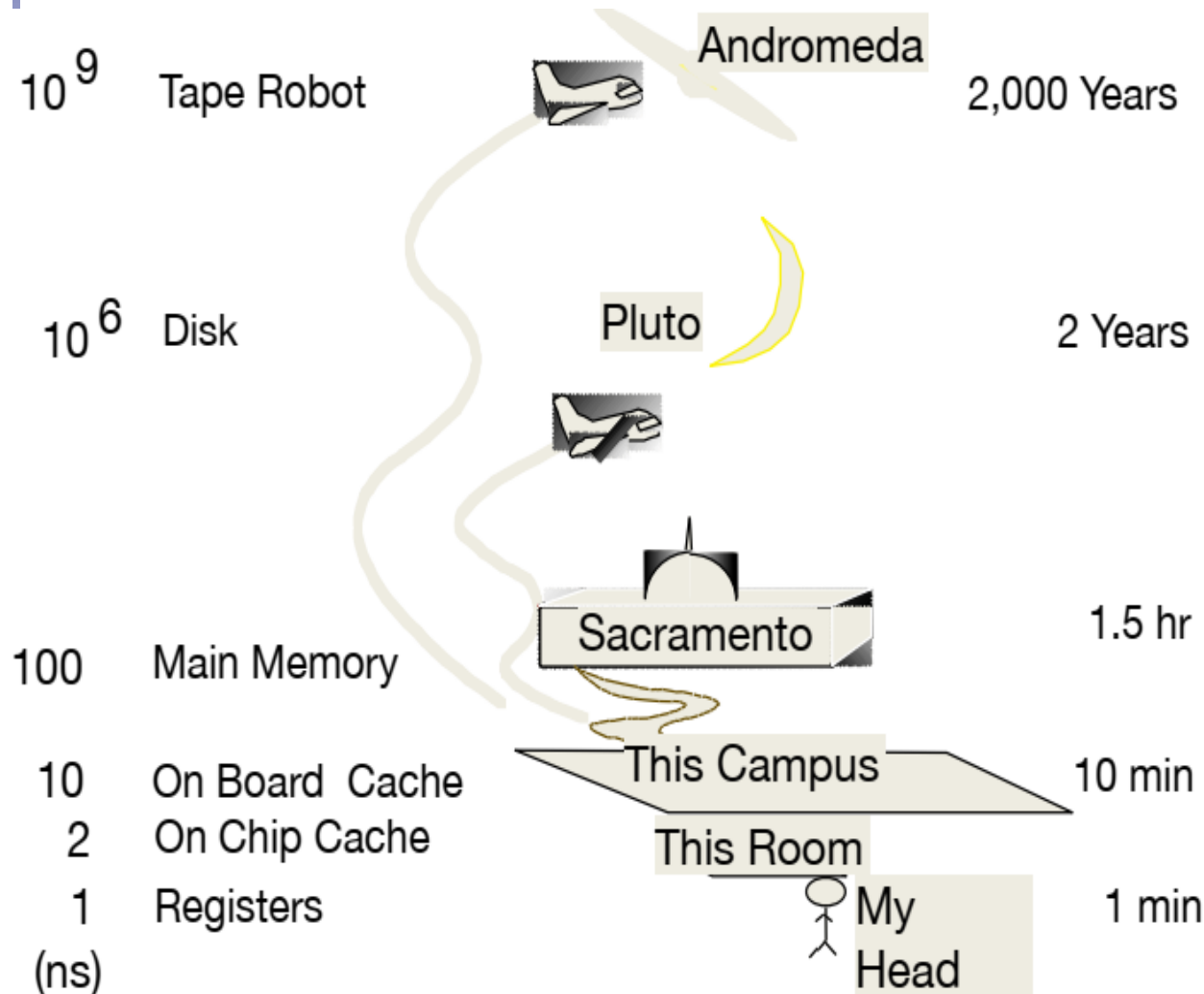
Parallelism 并行

- Raspberry Pi 4 B+
  - Quad core processor at 1.5 GHz
    - Each core is 3-issue, out-of-order superscalar
  - Plus GPU, 1-4 GB RAM, 2x USB3, 2x USB2, Gigabit Ethernet, 2x HDMI...
  - $35-55
    - Nick is working on a board to turn one of these to power a fully autonomous, vision-guided drone
- Compare with a Cray-1 from 1975:
  - 8 MB RAM, 80 MHz processor, 300MB storage, $5M+
- Or modern high end servers:
  - You can get a 2u server which supports 4 processors
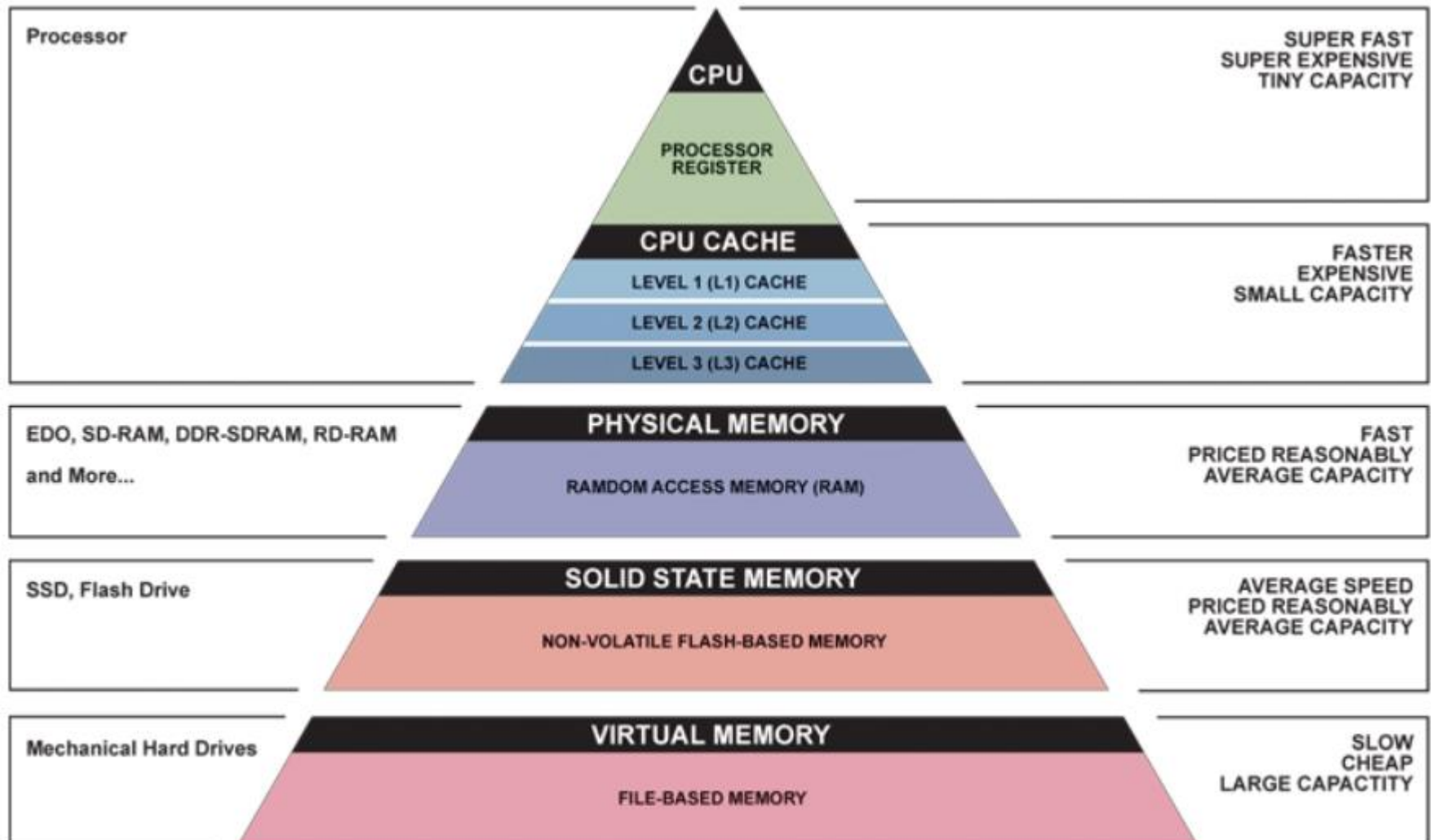   - And each processor can have 20+ cores, so 80 processor cores!

Jim Gray's Storage Latency Analogy

- How far away is the Data?

10^9	Tape Robot		Andromeda	2,000 Years
10^6	Disk		Pluto	2 Years
100	Main Memory		Sacramento	1.5 hr
10	On Board Cache		This Campus	10 min
2	On Chip Cache		This Room	
1	Registers		My Head	1 min
(ns)				

Jim Gray
Turing Award
B.S. Cal 1966
Ph.D. Cal 1969!

35

Memory Hierarchy

Processor

| | | SUPER FAST SUPER EXPENSIVE TINY CAPACITY |

CPU

PROCESSOR REGISTER

FASTER EXPENSIVE SMALL CAPACITY

CPU CACHE

LEVEL 1 (L1) CACHE

LEVEL 2 (L2) CACHE

LEVEL 3 (L3) CACHE

EDO, SD-RAM, DDR-SDRAM, RD-RAM and More...

PHYSICAL MEMORY

RAMDOM ACCESS MEMORY (RAM)

FAST PRICED REASONABLY AVERAGE CAPACITY

SSD, Flash Drive

SOLID STATE MEMORY

NON-VOLATILE FLASH-BASED MEMORY

AVERAGE SPEED PRICED REASONABLY AVERAGE CAPACITY

Mechanical Hard Drives

VIRTUAL MEMORY

FILE-BASED MEMORY

SLOW CHEAP LARGE CAPACTITY

Fails Happen, so?

- 4 disks/server, 50,000 servers
- Failure rate of disks: 2% to 10% / year
  - Assume 4% annual failure rate
- On average, how often does a disk fail?
  a) 1 / month
  b) 1 / week
  c) 1 / day
  d) 1 / hour

50,000 x 4 = 200,000 disks
200,000 x 4% = 8000 disks fail
365 days x 24 hours = 8760 hours

Reliability via Redundancy

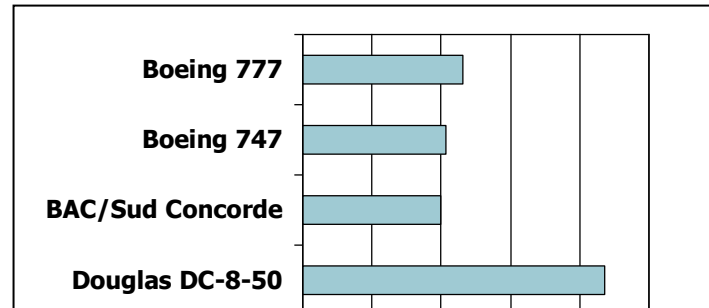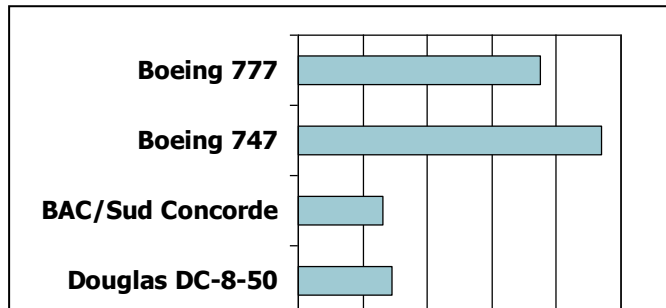Redundancy so that a failing piece doesn't make the whole system fail
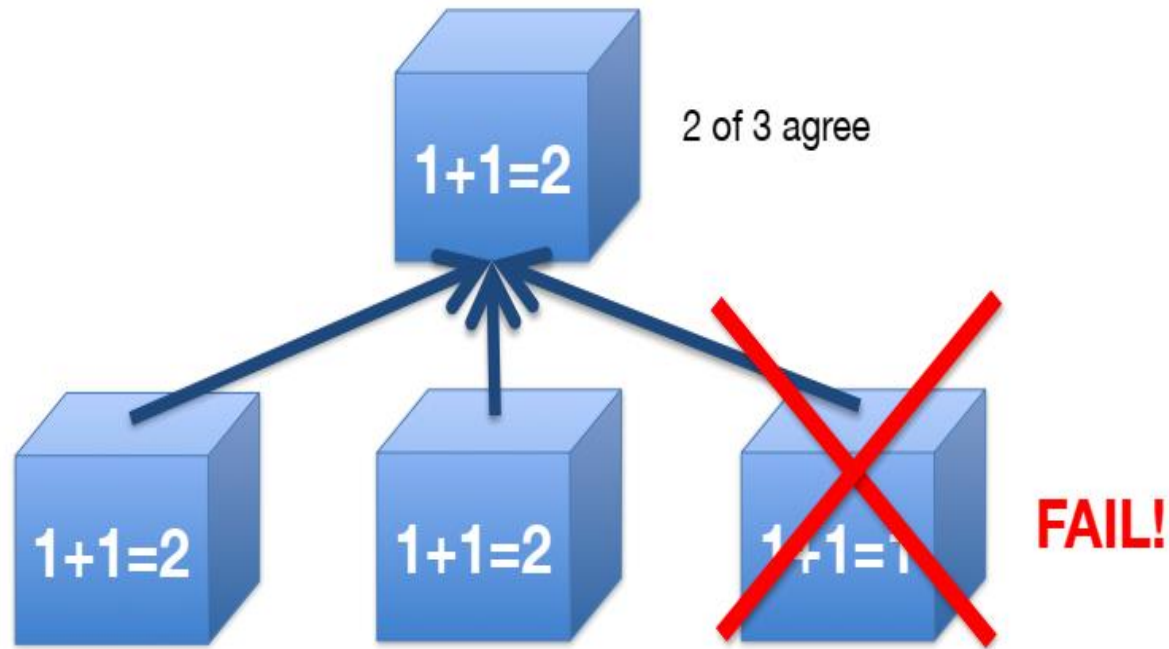
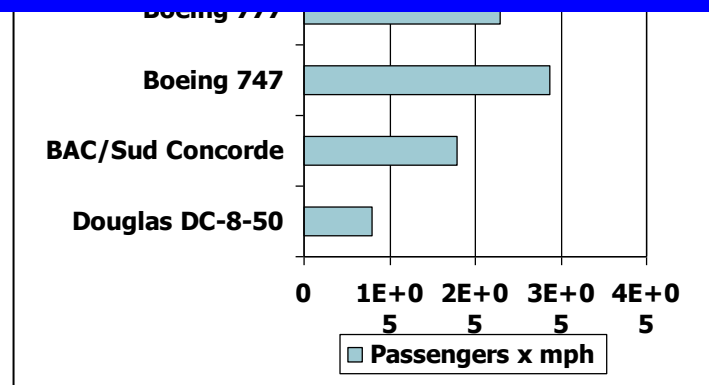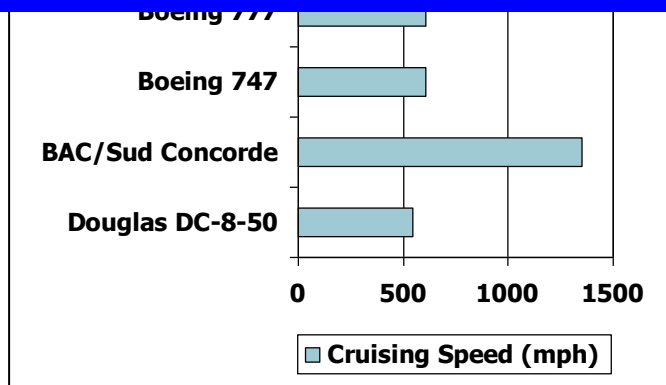Increasing transistor density reduces the cost of redundancy

Lecture 2

Performance

Defining Performance

- Which airplane has the best performance?

To evaluate the performance, we must define the metric first!

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? { response ↓ through ×2

  - Adding more processors? { response 不变 (depend on division). throughput ×2

- We'll focus on response time for now...

Relative Performance

- Define Performance = 1/Execution Time

- "X is n time as fast as 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 of B / Execution Time of A
    = 15s / 10s = 1.5

  - So A is 1.5 times as fast as B

- Increase performance = decrease execution time

  → "**improve**" performance/execution time.

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
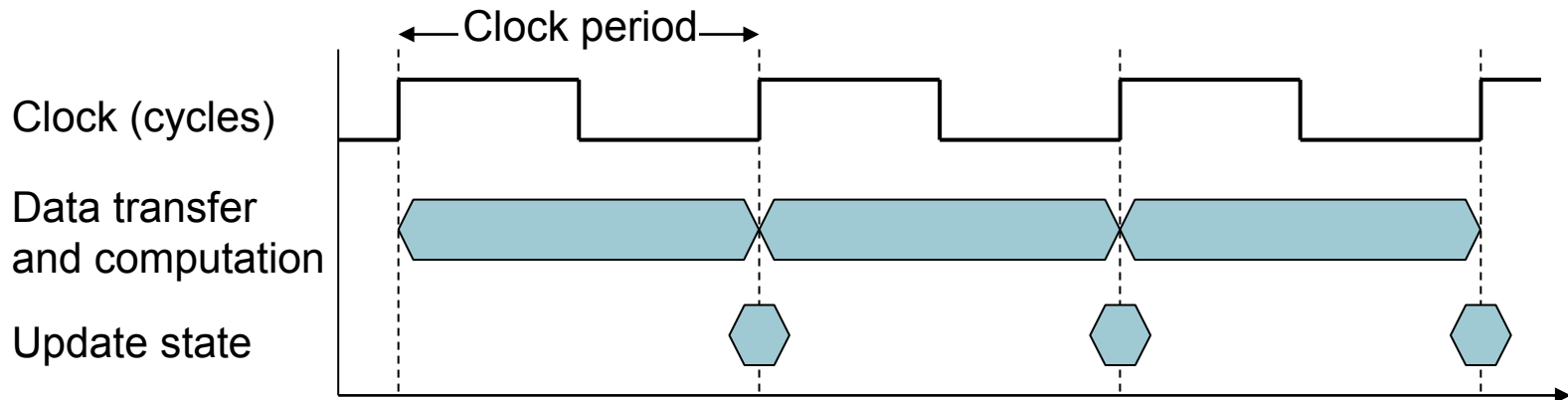
- Clock period: duration of a clock cycle

  - 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

18

CPU Time

$$\text{CPU Time} = \text{No. of Clock Cycles} \times \text{Clock Period}$$

$$= \frac{\text{No. of Clock Cycles}}{\text{Clock Rate}}$$

- **Performance improved by**

  - ◆ Reducing number of clock cycles (cycle count)

  - ◆ 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 $1.2 \times$ number (#) of clock cycles

- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{No.\,of\ \text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times No.\,of\ \text{Clock Cycles}_A}{6s}$$

$$No.\,of\ \text{Clock Cycles}_A = \text{CPU Time}_A \times No.\,of\ \text{Clock Rate}_A$$
$$= 10s \times 2\text{GHz} = 20 \times 10^9$$

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

Instruction Count and CPI

$No. of$ Clock Cycles = Instruction Count × Cycles per Instruction (CPI)

CPU Time $\quad$ = Instruction Count × CPI × Clock Period

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

1- cycle

CPI = 1

different clock time

pipeline

CPI = 4

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

21

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?

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

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps} \quad \longleftarrow \boxed{\text{A is faster…}}$$

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

$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2 \quad \longleftarrow \boxed{\text{…by this much}}$$

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

- Alternative compiled code sequences using instructions in classes A, B, C. IC is short for "instruction count".

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
    $= 2 \times 1 + 1 \times 2 + 2 \times 3$
    $= 10$
  - Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
  - Clock Cycles
    $= 4 \times 1 + 1 \times 2 + 1 \times 3$
    $= 9$
  - Avg. CPI = 9/6 = 1.5

Performance Summary

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

$$= \text{IC} \times \text{CPI} \times \text{Tc}$$

- 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

Energy Consumption of a chip

- Energy consumption = dynamic energy + static energy

  - Dynamic energy (energy spent when transistors switch from $0\rightarrow1$ $1\rightarrow0$) is primary

  - Static energy is the energy cost when no transistor switches

- Energy for $0\rightarrow1\rightarrow0$: $Energy \propto Capacitive\ load \times Voltage^2$

- Energy for $0\rightarrow1$ or $1\rightarrow0$: $Energy \propto 1/2 \times Capacitive\ load \times Voltage^2$

- Energy per second (power):

$$Power \propto 1/2 \times Capacitive\ load \times Voltage^2 \times Frequency\ switched$$

Power Trends
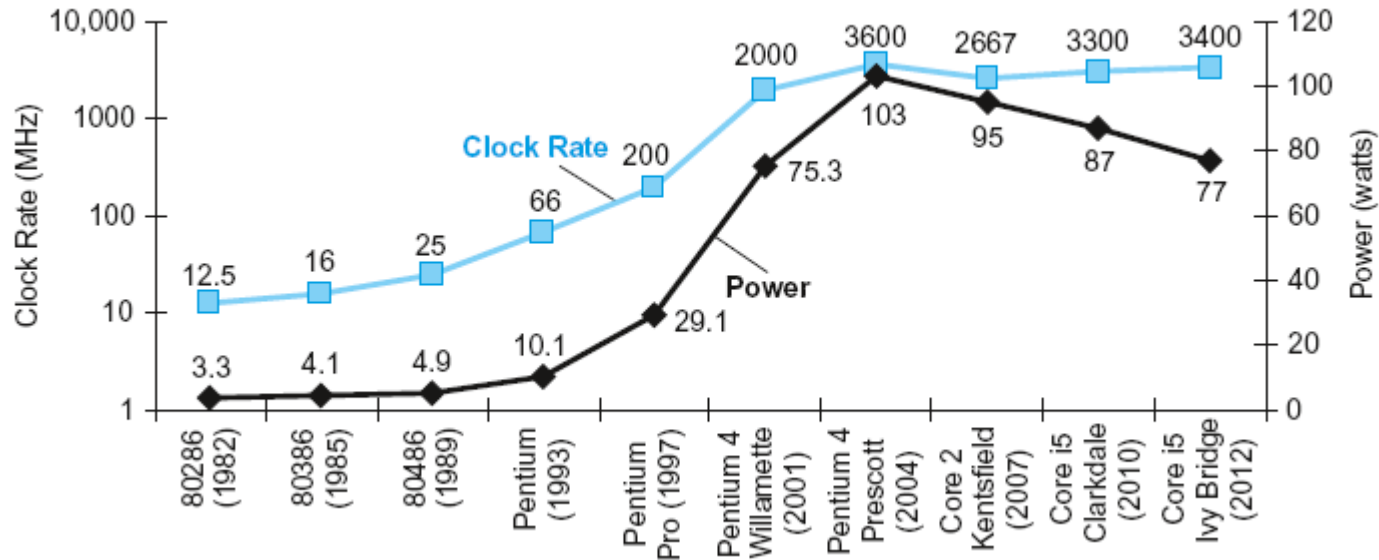
- In CMOS IC technology

$$\text{Power} \propto \frac{1}{2}\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$
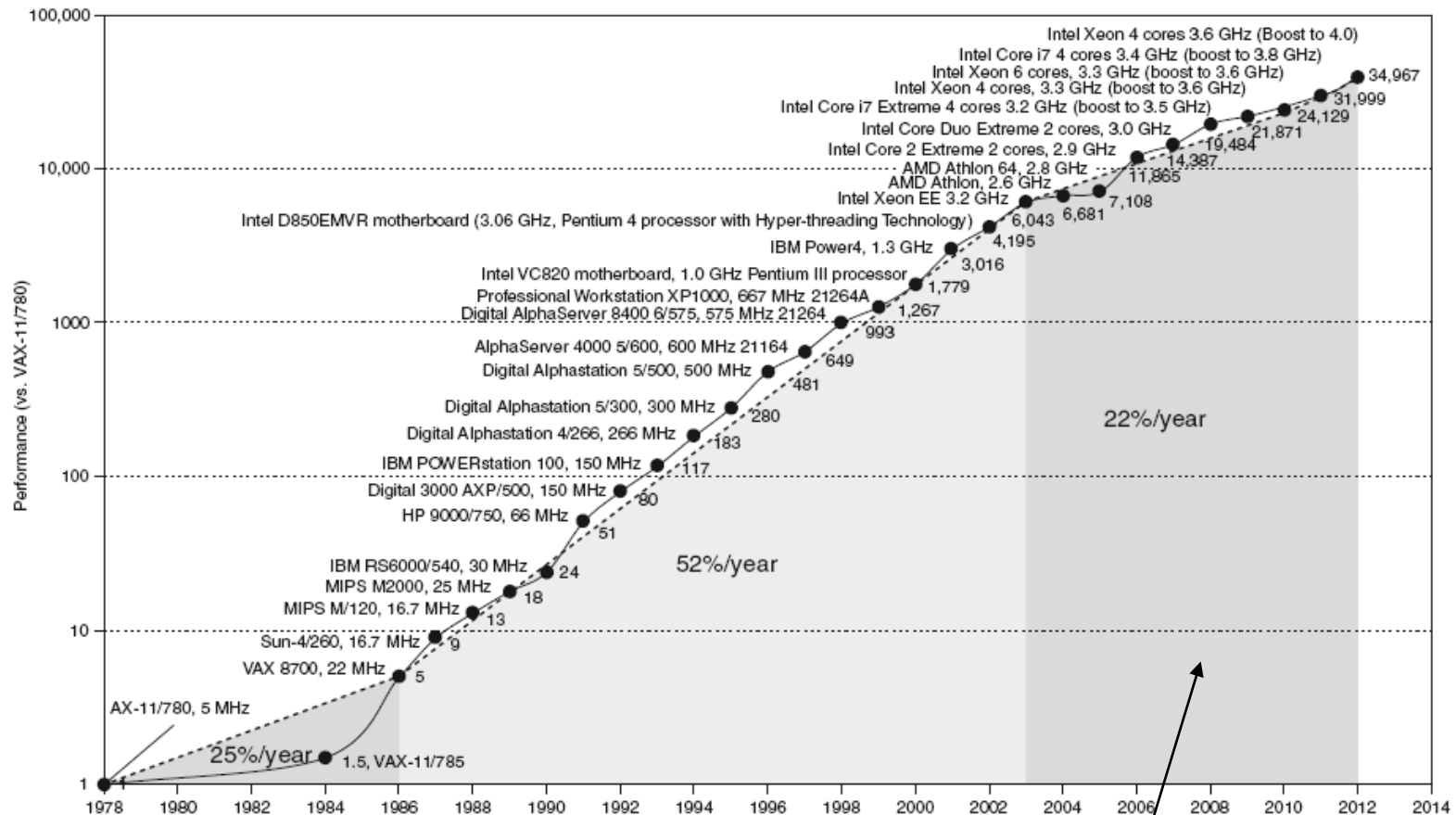
$\times 23$

$5V \rightarrow 1.5V$

$\times 270$

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?

Uniprocessor Performance

Constrained by power, instruction-level parallelism, memory latency

29

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

Benchmark Suites

- Each vendor announces a SPEC (Standard Performance Evaluation Cooperative) rating for their system

  - a measure of execution time for a fixed collection of programs

  - is a function of a specific CPU, memory system, IO system, operating system, compiler

  - enables easy comparison of different systems

- The key is coming up with a collection of relevant programs

SPEC CPU Benchmark

- Programs used to measure performance

  - Supposedly typical of actual workload

- Standard Performance Evaluation Cooperative (SPEC)

  - Develops benchmarks for CPU, I/O, Web, …

- SPEC CPU2006

  - Elapsed time to execute a selection of programs

    - Negligible I/O, so focuses on CPU performance

  - Normalized relative to reference machine

  - Summarize as geometric mean of performance ratios

    - CINT2006 (integer) and CFP2006 (floating-point)

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

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC Power Benchmark

- Power consumption of the server at different workload levels

  - Performance: ssj_ops (server side Java operations per second)

  - 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 X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Amdahl's Law

- Architecture design is very bottleneck-driven – make the common case fast, do not waste resources on a component that has little impact on overall performance/power

- Amdahl's Law: performance improvements through an enhancement is limited by the fraction of time the enhancement comes into play 时间占比更大

- Example: multiply accounts for 80s/100s

  - How much improvement in multiply performance to get $5\times$ 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

Concluding Remarks

- Knowledge of hardware improves software quality:
  - compilers, OS, threaded programs, memory management
- Important trends:
  - growing transistors
  - move to multi-core
  - slowing rate of performance improvement
  - power/thermal constraints
- Reasoning about performance: clock speeds, CPI, benchmark suites, performance equations
- Next: assembly instructions