CS6L047: Advanced Computer Architecture

(Chapter 1)
Fundamentals of Quantitative Design and Analysis

- Trends
- Power
- Measuring Performance

Research Trends in Systems World

- IISc Systems Workshop Link:

- My Talk: https://www.youtube.com/watch?v=Hvf9ST8JFAo

  https://www.youtube.com/watch?v=xQcYnFof5nc

- Schedule Invited Talks from Industry Experts (Google, Micron, Intel, AMD)
  - Check Availability, Time Zone

AI models Challenges and Limitations

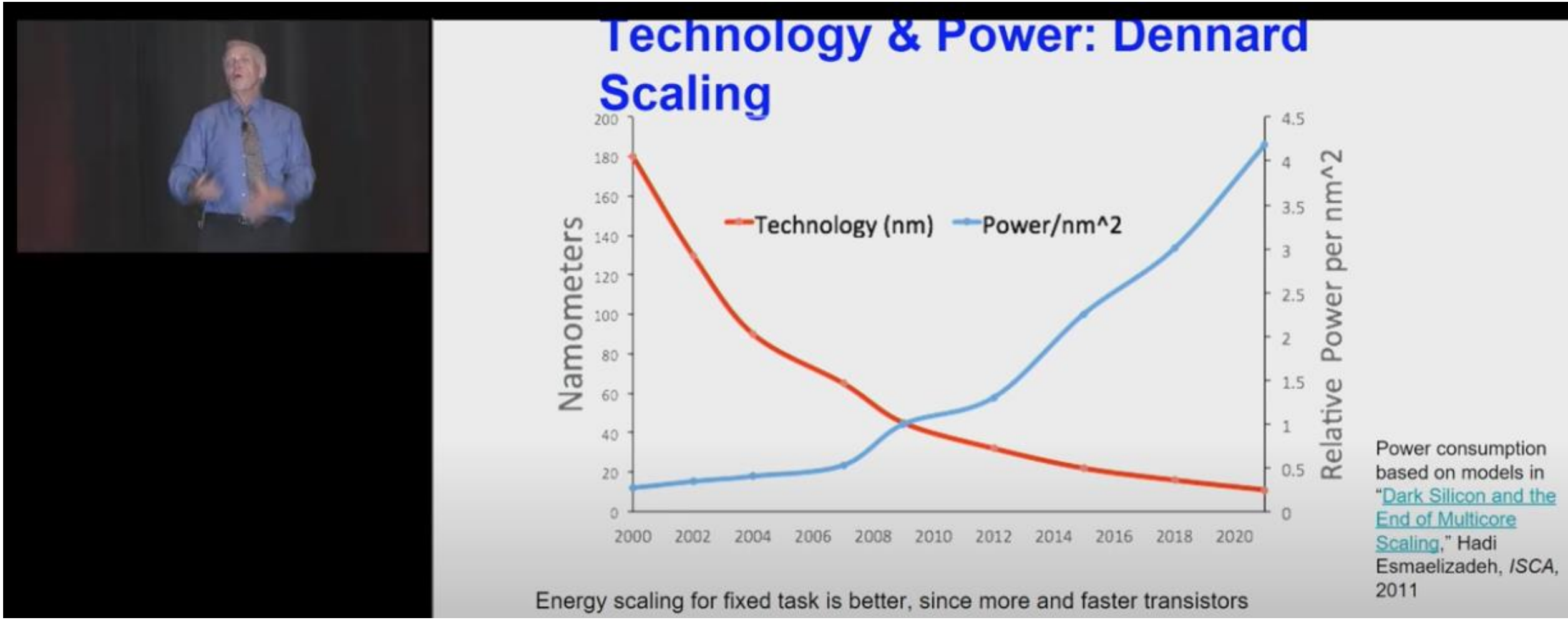
of a Computer Architect, Challenges in LLM Optimization:

- Memory constraints and management.
- Computational complexity and resource requirements.
- Latency and throughput considerations.

Problem: Data and Compute requirement of LLM/AI models

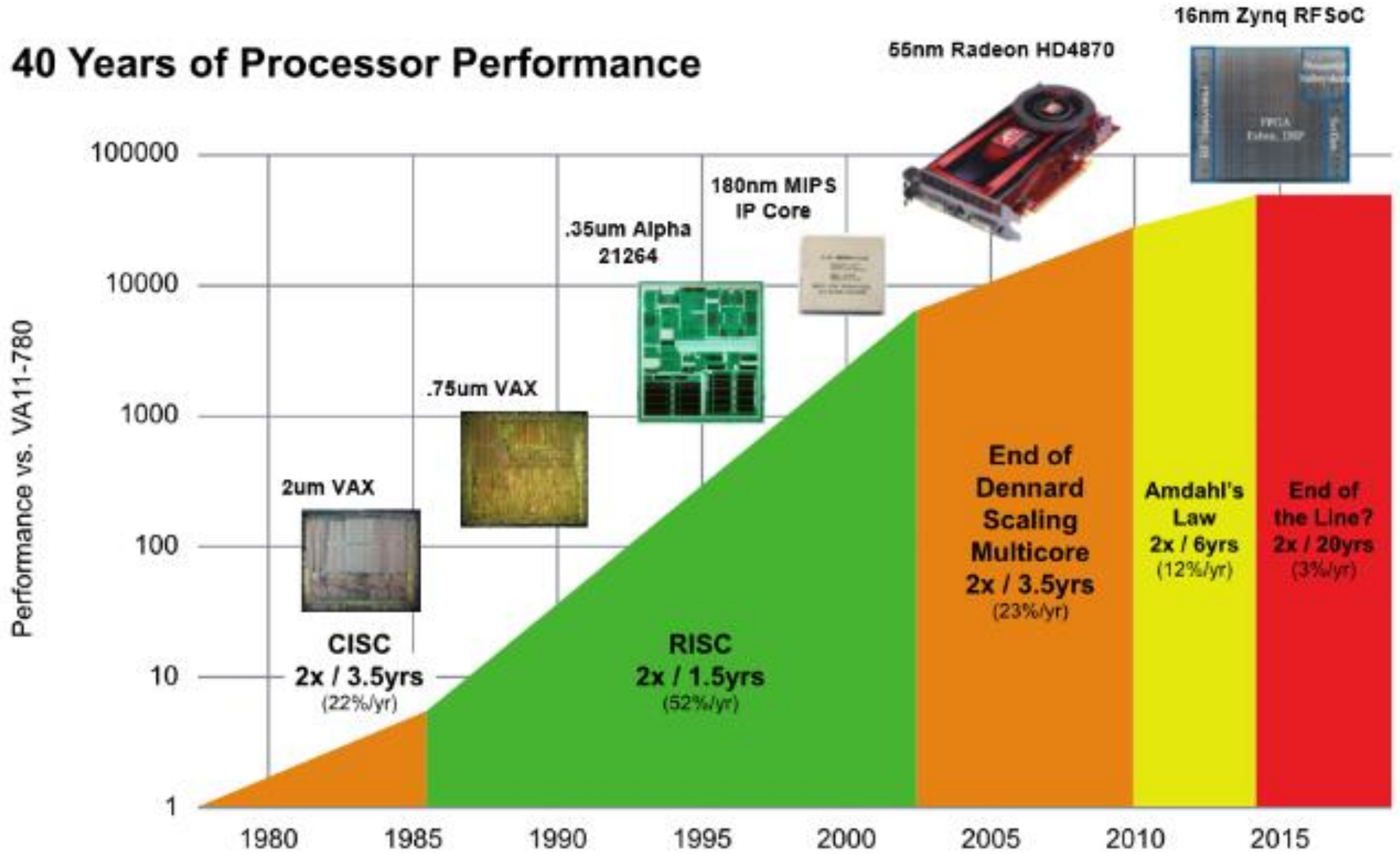
Solution:

Advancements in Hardware, Algorithms and Datasets.



John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture

40 Years of Processor Performance



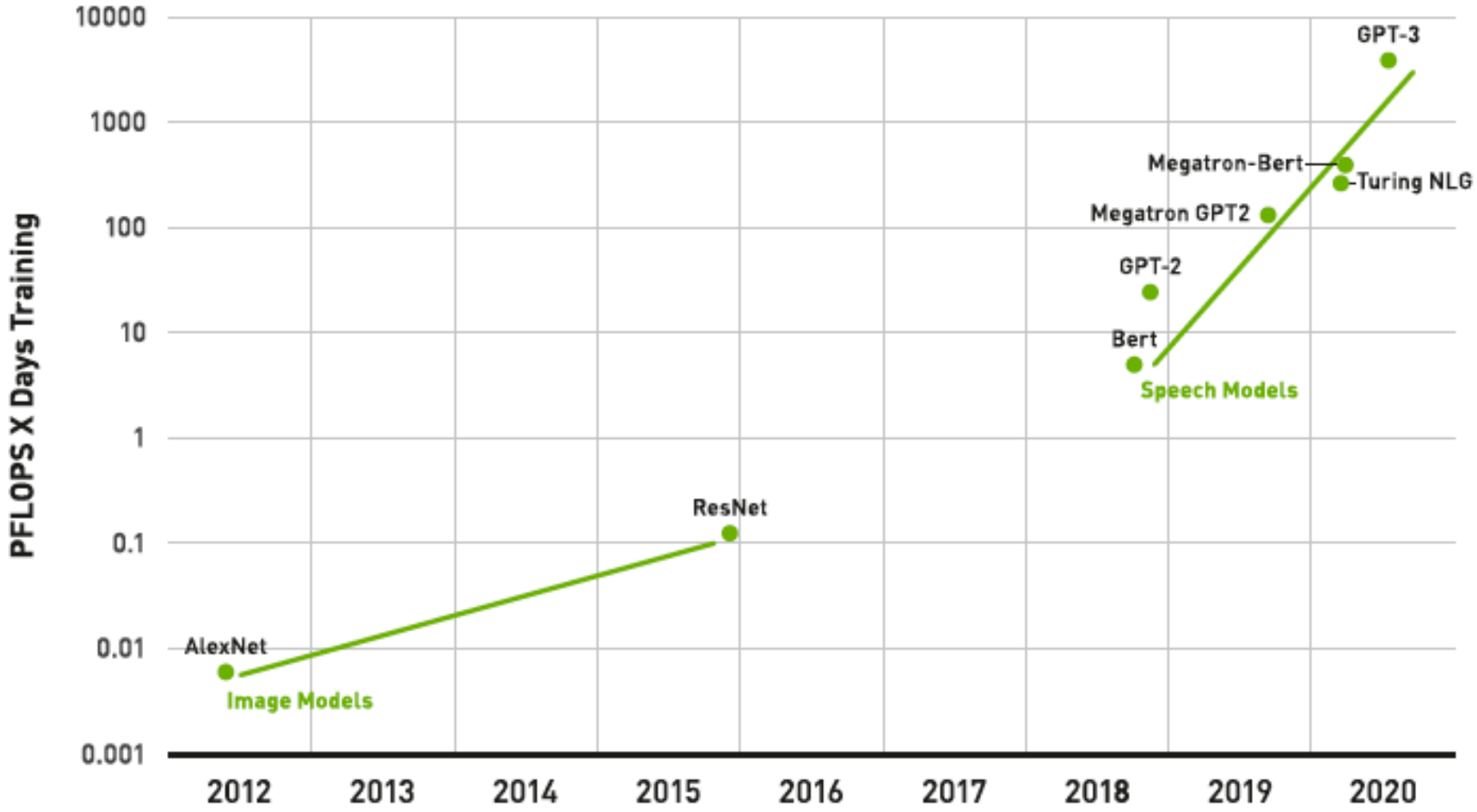
Domain Specific Architectures (DSAs)

- Achieve higher efficiency by tailoring the architecture to characteristics of the domain
  - Not one application, but a domain of applications
    -Different from strict ASIC
  - Requires more domain-specific knowledge then general purpose processors need
- Examples:
  - Neural network processors for machine learning
  - GPUs for graphics, virtual reality
  - Programmable network switches and interfaces

John Hennessy and David Patterson 2017 ACM A.M. Turing Award Lecture

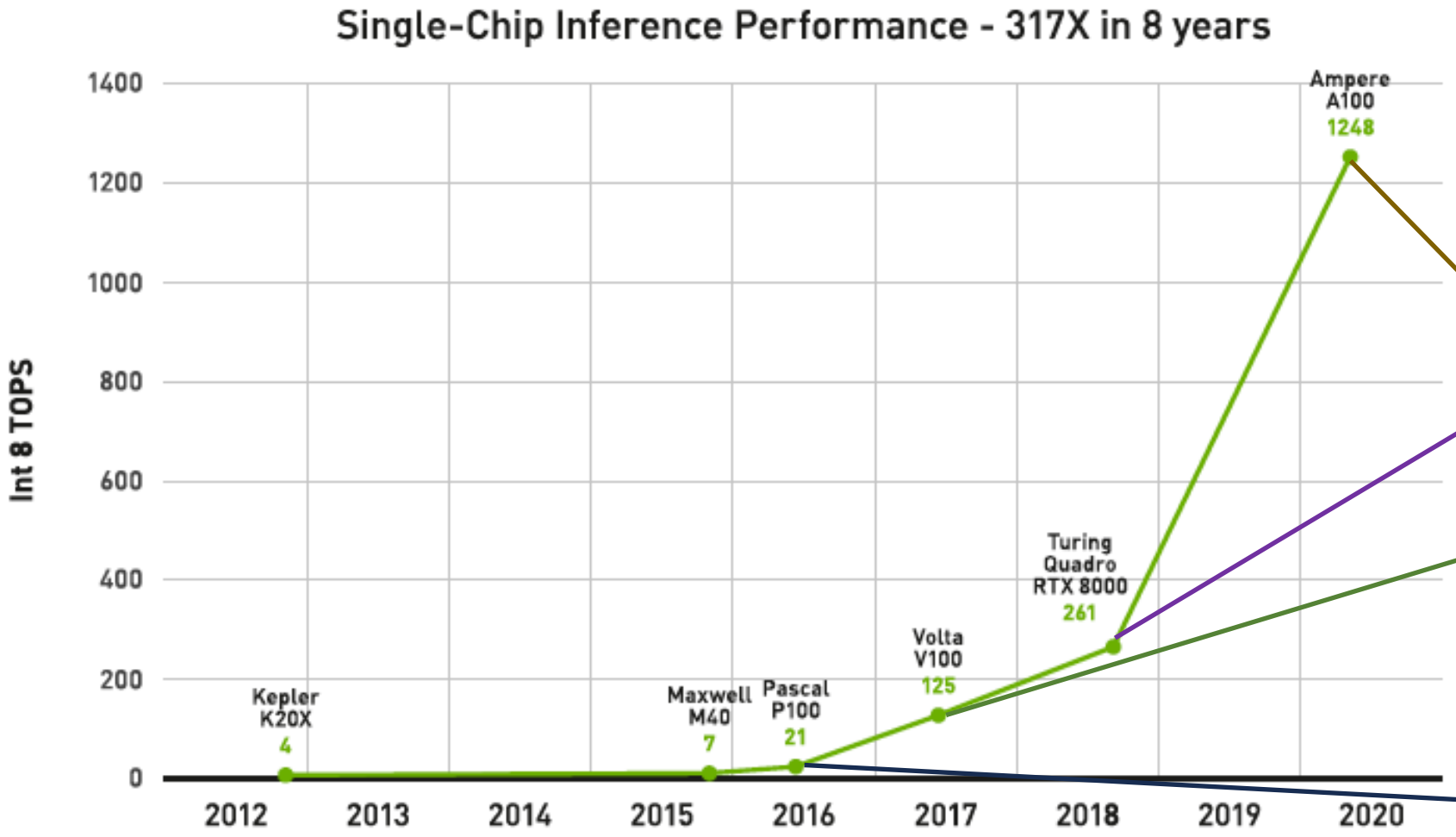
How has the
Deep Learning Hardware
evolved over the years?

Scaling of DL depends on Hardware Performance



Source: Dally.et.al, Evolution of the graphics processing Unit, NVIDIA, 2021

Huang's Law: GPU Inference Performance is >2x every year



Tensor Core support for Sparse data computation

Integer Tensor cores (INT8)

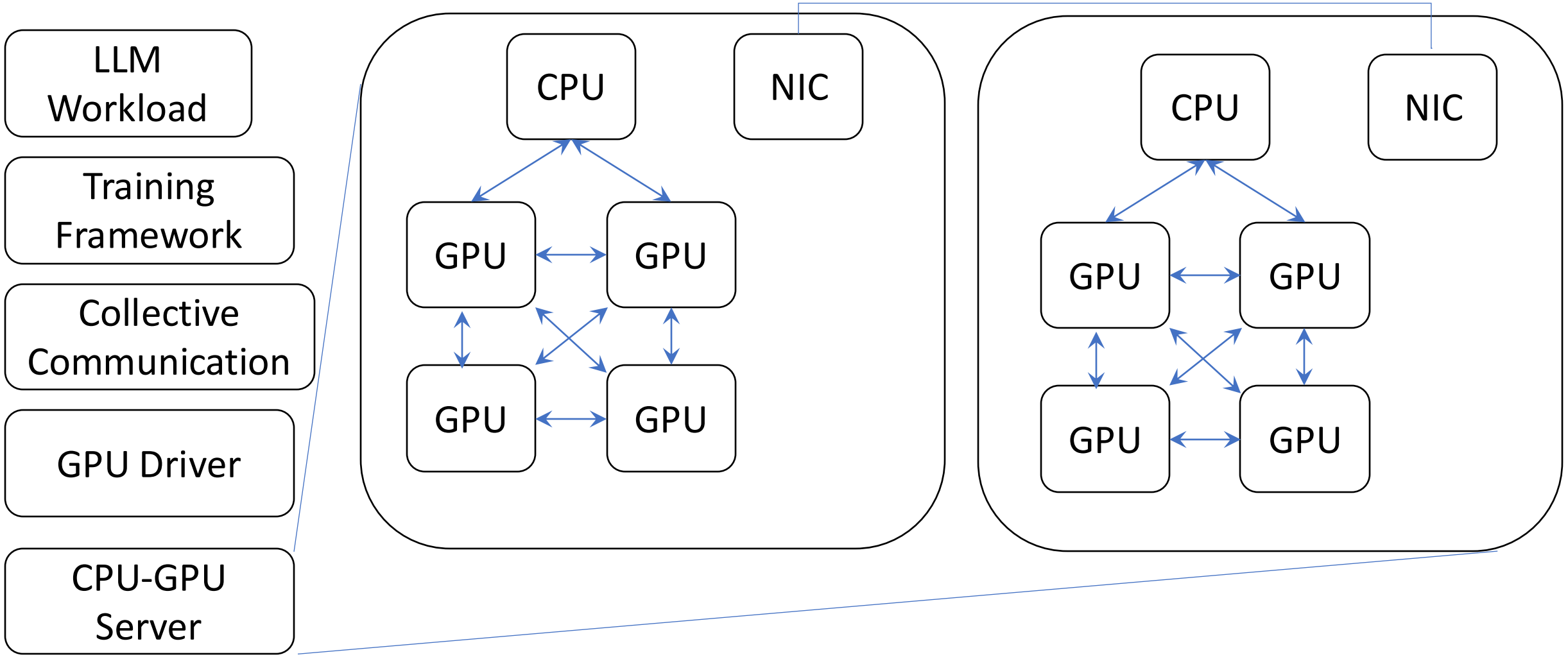
1. NVSwitch: High Performance Networks, infiniband connected GPUs
2. Tensor Core architecture: Flexibility of Programmable Architecture (FP16)

NVLink: GPU-GPU communication measured on FP32

Source: Dally.et.al, Evolution of the graphics processing Unit, NVIDIA, 2021

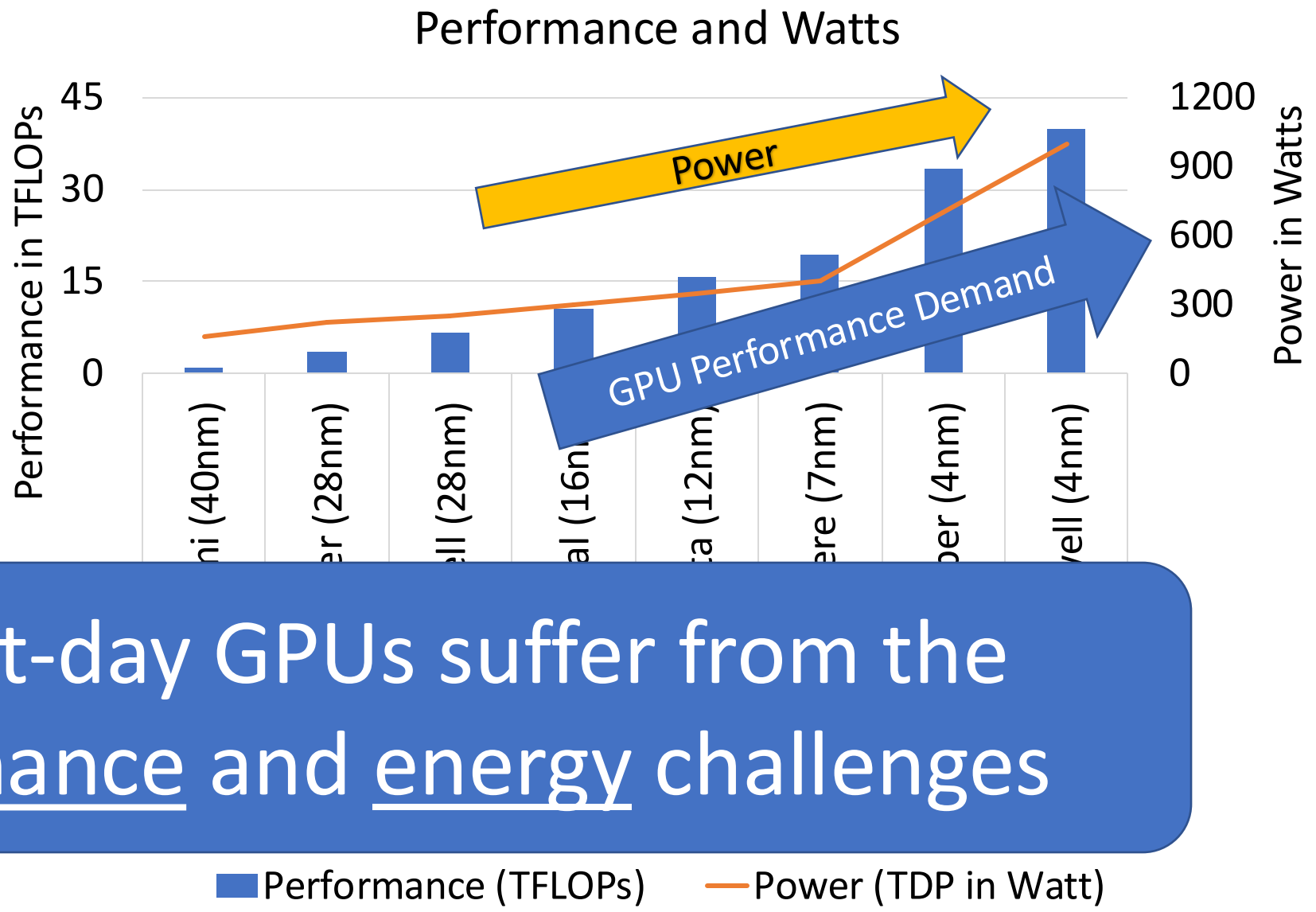
Key Components in Accelerating DL on GPUs

- Hardware Advancements (NVLink, NVSwitch, FP16, Tensor Cores, …)
- Software Advancements:
  - cuDNN
  - NVIDIA’s Tensor RT (TRT): optimize inference code, schedules multiple models.
  - TensorFlow, PyTorch, Caffe2
  - DALI: training pipeline


LLM Workload
Training Framework
Collective Communication
GPU Driver
CPU-GPU Server
CPU
NIC
GPU
GPU
GPU
GPU
CPU
NIC
GPU
GPU
GPU
GPU

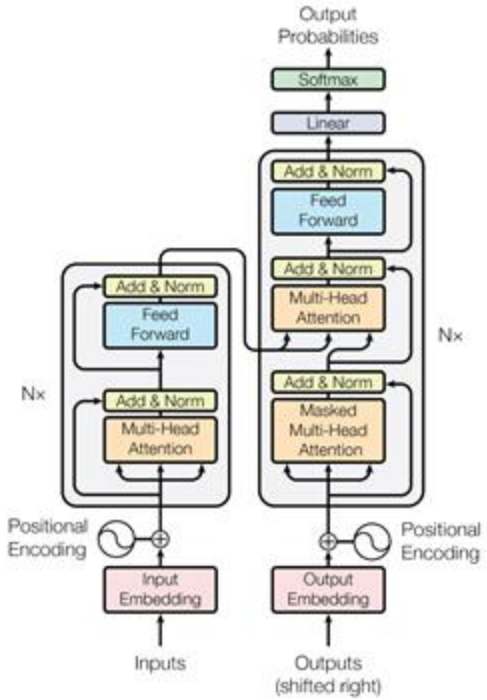
GPU Performance increases with Power

- GPUs are one of the de-facto accelerator for high-performance computing, machine learning.
- Used for data-parallel tasks



Present-day GPUs suffer from the performance and energy challenges

Gen AI is Gaining Attention



OpenAI

Gemini

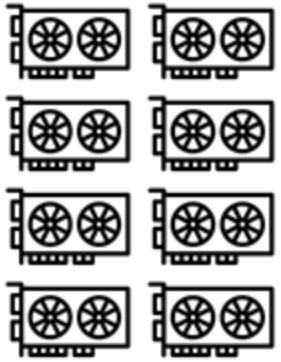
Claude

AGI?
(Artificial General Intelligence)

Ashish Vaswani.et al “Attention Is All You Need” NIPS 2017

A Lot of Energy and Carbon Emission

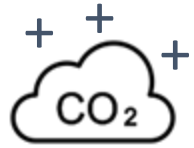
AI Chatbot is a trending hot app



xAI data center has 100k GPUs

GPU shipped by Nvidia can use 14.4 TWh power for a year

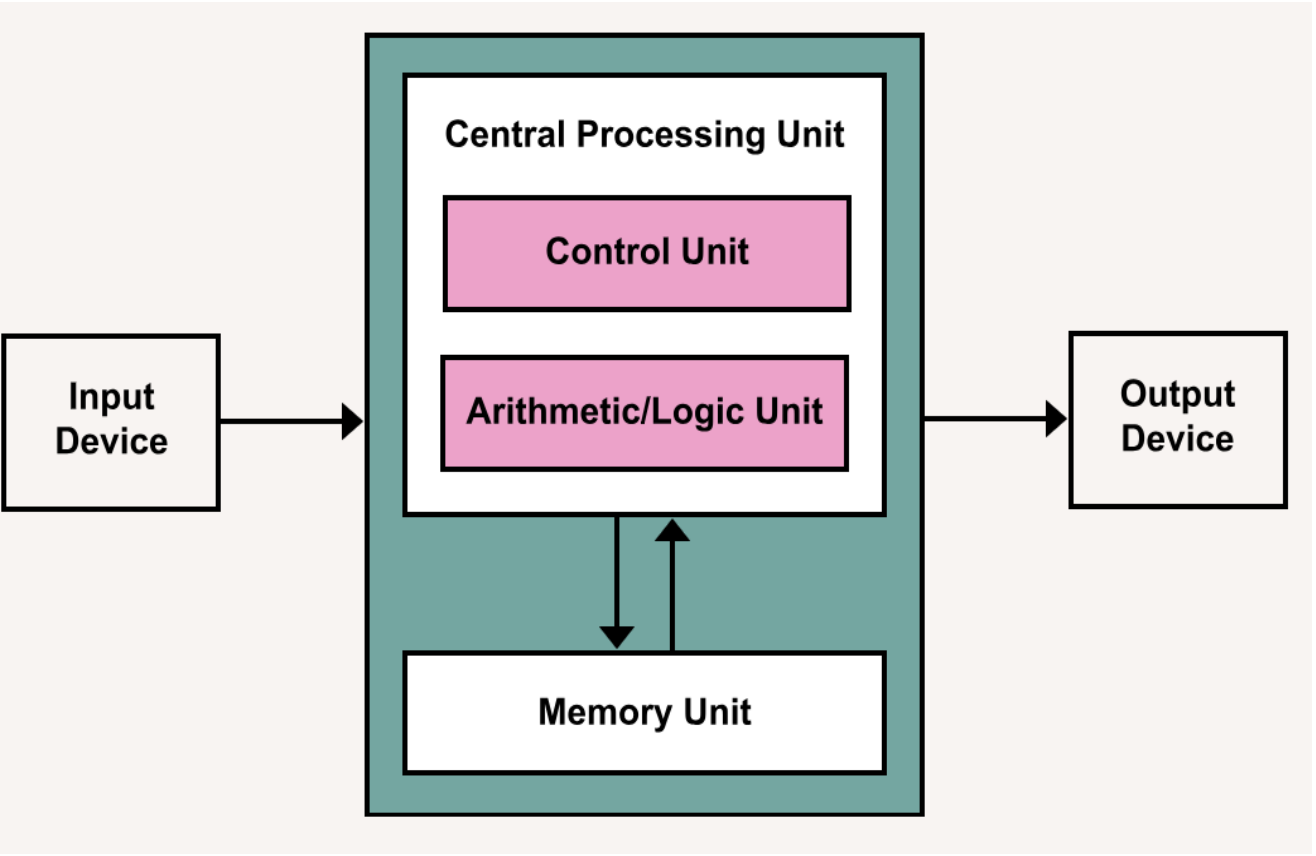
=



Data center accounts for up to 3.7% of global greenhouse gas emissions

Can we reduce the power/energy used for LLM?

The von Neumann Computer



- Von Neumann Computer: A program is written as a sequence of instructions, represented by binary numbers. The instructions are stored in the memory just as data. They are read one by one, decoded and then executed by the CPU.

- Stored-Program Concept – Storing programs as numbers – by John von Neumann – Eckert and Mauchly worked in engineering the concept.

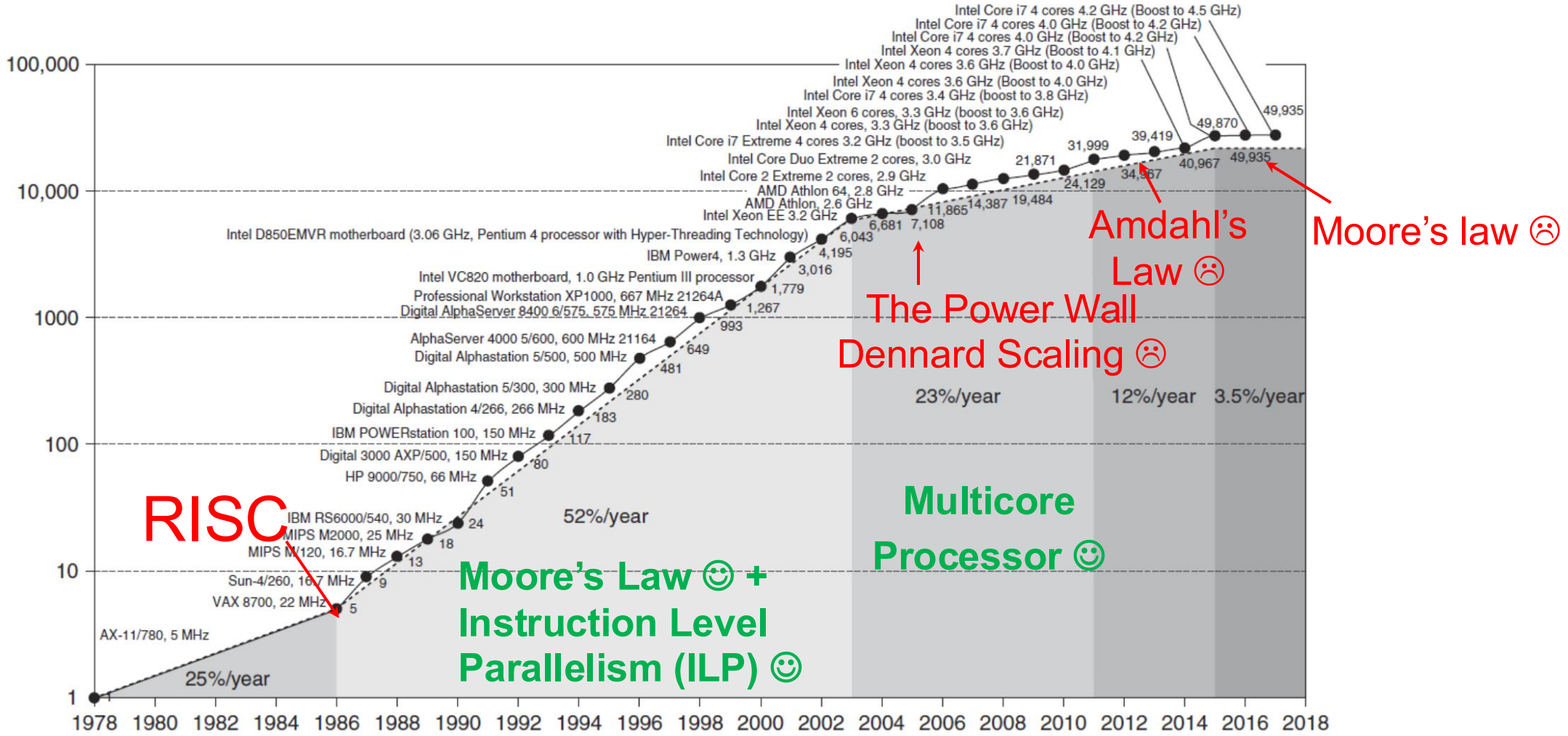
Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, 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
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Internet of Things/Embedded Computers
  - Emphasis: price

Classes of Computers

Classes	Examples	Key Focus/Emphasis	Why It Matters	Analogy
Personal Mobile Devices	Smartphones, tablets	Energy efficiency & instant response	Battery-powered; must react quickly to user input	Smart assistant in your pocket
Desktop Computing	Home, school, office PCs	Price vs. performance	Not mobile, but powerful & cost-effective	Reliable all-rounder
Servers	Web & app backend machines	Availability, scalability, throughput	Must stay online and handle increasing demand	24/7 worker with flexible backup
Clusters / Warehouse-Scale	Google, Microsoft, Netflix servers	Availability & price-performance	Large-scale computing to deliver cloud software	Massive team powering online tools
Supercomputers	Scientific research systems	Speed & complex calculations	Solves tough problems like climate models or DNA study	Brainiac of computers
IoT / Embedded Devices	Smart fridges, watches, sensors	Low price	Task-specific, tiny, low-cost, pervasive	Tiny helpers doing one job really well

Growth in Processor Performance



CISC vs RISC

Feature	CISC	RISC
Instruction Length	Longer and complex	Short and simple
Execution Time	Often slower (but fewer lines)	Faster per instruction
Memory Use	Less (fewer instructions)	More (more instructions)
Hardware Complexity	Higher	Lower
Assembly	ADD A, B This single instruction: Loads A and B from memory Adds them Stores the result	LOAD R1, A ; Load value A into register R1 LOAD R2, B ; Load value B into register R2 ADD R3, R1, R2 ; Add R1 and R2, store result in R3 STORE R3, C ; Save result from R3 into memory RISC requires multiple instructions, each doing one specific job.

Which architecture does your laptop have?

Processor Type	Architecture	Example Brands	Typical Use
Intel Core i3/i5/i7	CISC (x86)	Dell, HP, Lenovo, Asus	General-purpose laptops
AMD Ryzen Series	CISC (x86)	Acer, HP, Lenovo	Budget and performance laptops
Apple M1/M2/M3 chips	RISC (ARM)	MacBook Air, MacBook Pro	Apple laptops (macOS)
Chromebooks (ARM-based)	RISC (ARM)	Lenovo, Asus, HP	Lightweight, web-based tasks

Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
  - **Power and heat constraints**: Faster CPUs consumed too much energy.
  - **Diminishing returns**: Complex ILP hardware gave smaller performance boosts.
  - **Software bottlenecks**: Most programs didn't have enough independent instructions

- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)

- These require explicit restructuring of the application

ILP in Image Processing Pipeline

- Task: Build app that applies filters to a batch of images — say, resizing, sharpening, and color correction.

With ILP:

- You optimize a single-threaded loop:

```
for (int i = 0; i < N; i++) {
    process_image(images[i]);
}
```

The CPU tries to parallelize instructions inside process_image, but:

- Each image is processed sequentially.
- Memory access latency and dependencies stall execution.

DLP in Image Processing Pipeline

To go faster, you need to **restructure the code** to use:

Data-Level Parallelism (DLP) 🧮

- Apply the same operation to multiple data items at once — great for vectorized tasks.

```
// Using SIMD to process 4 images at once
vector<Image> batch = load_batch(images[i..i+3]);
vector<Image> result = apply_filter(batch);
```

- ✅ Ideal for scientific computing, graphics, and batch operations.

TLP in Image Processing Pipeline

- Split the work across multiple threads or cores.

```
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    process_image(images[i]);
}
```

- ✅ Perfect for multicore CPUs and workloads with independent tasks.

RLP in Image Processing Pipeline

- Handle many independent requests — like user uploads or API calls — in parallel.

```
// Each image upload is handled by a separate thread or service
handle_upload(user1_image);
handle_upload(user2_image);
handle_upload(user3_image);
```

- ✅ Common in cloud services, web servers, and microservices.

Code Restructuring for different parallelism

- These models **don’t work automatically** like ILP did. You need to:
- Write parallel code (e.g., using OpenMP, CUDA, or threads).
- Design algorithms that avoid shared state and bottlenecks.
- Use hardware features like vector units or multiple cores.

Model	What It Does	Example Use Case
ILP	Parallel instructions in one thread	Optimized loops
DLP	Parallel data operations	Image filters, matrix math
TLP	Parallel threads	Multicore search, simulations
RLP	Parallel requests	Web servers, cloud APIs

Parallelism

- Classes of **architectural** parallelism:
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism (Multiprocessor)
  - Request-Level Parallelism
- Classes of parallelism in **applications**:
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)

Code Example: Processing an Array

Data-Level Parallelism

- Same operation on different data chunks:

```
// Add two arrays in parallel
#pragma omp parallel for
for (int i = 0; i < N; i++) {
C[i] = A[i] + B[i];
}
```

✅ Efficient when the same computation is applied to large datasets.

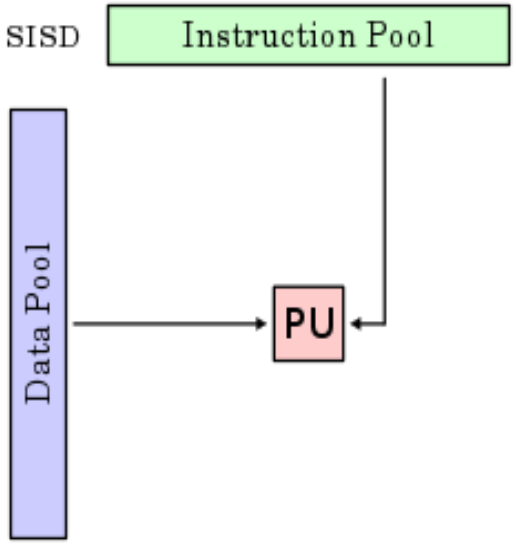
Task-Level Parallelism

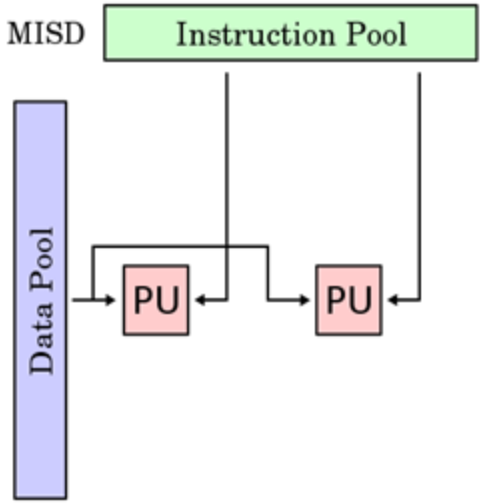
- Different operations on possibly same or different data:

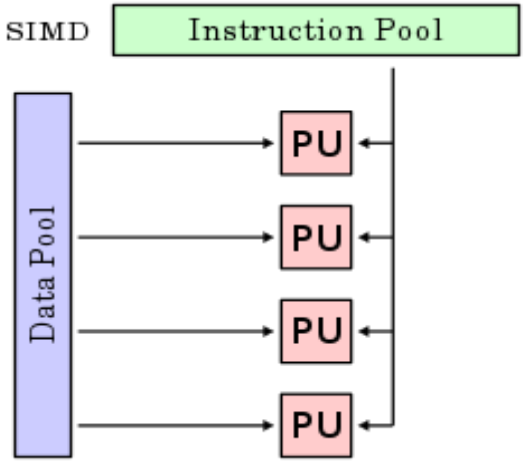
```
#pragma omp parallel sections {
 #pragma omp section
process_audio(audio_data);
 #pragma omp section
process_video(video_data);
 #pragma omp section
apply_effects(effect_data); }
```

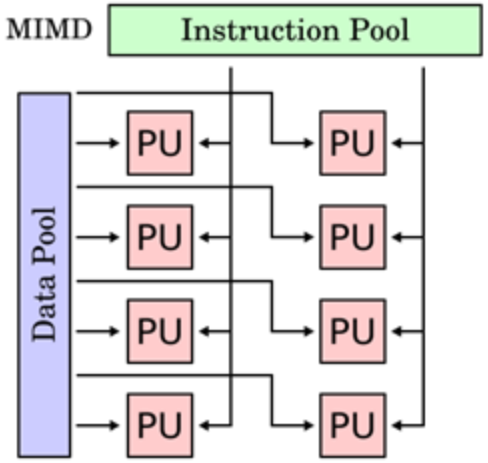
✅ Great for multimedia apps, simulations, or workflows with distinct stages.

Popular Flynn Categories









Popular Flynn Categories

- **SISD (Single Instruction Single Data stream)**
  - A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single Instruction Stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) to operate on single Data Stream (DS) i.e. one operation at a time.

  - **What it means**: One processor executes one instruction at a time on one piece of data.
  - **Example**: A basic laptop running a single-threaded program.
  - **Analogy**: One chef making one dish using one recipe

- **MISD (Multiple Instruction Single Data stream)**
  - multiple processors on a single data stream
  - Multiple instructions operate on a single data stream. Uncommon architecture
  - **What it means**: Multiple processors execute different instructions on the same data.
  - **Example**: Rare in practice, but used in **fault-tolerant systems** where the same data is processed in different ways to detect errors.
  - **Analogy**: Several chefs taste the same soup and each gives a different opinion or analysis.

SIMD (Single Instruction Multiple Data)

A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized.

Examples: Illiac-IV (1st machine), Array processor, GPU, Simple programming model, Low overhead, Flexibility, suitable for most scientific applications, not suitable for general-purpose applications

- **What it means**: Multiple processors execute the same instruction on different pieces of data.
- **Example**: A GPU applying the same filter to every pixel in an image.
- **Analogy**: Many chefs chopping different vegetables using the same chopping technique.

MIMD (Multiple Instruction Multiple Data stream)

Multiple autonomous processors simultaneously executing different instructions on different data.

Examples: C.mmp (1st machine), All modern Multicore machines, Clusters, Top 500 supercomputers, Data centers

Flexible

- **What it means**: Multiple processors execute different instructions on different data.
- **Example**: A multicore server handling different user requests simultaneously.
- **Analogy**: A team of chefs each cooking a different dish with different ingredients.

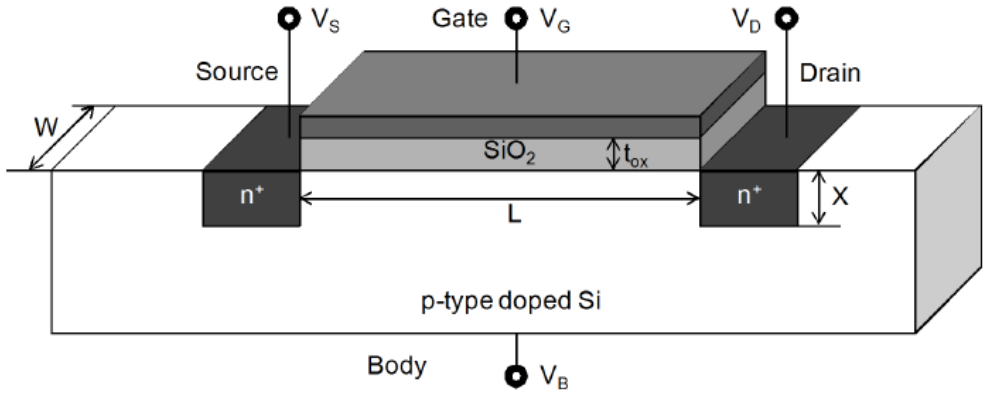
Flynn's Taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units (GPU)
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD: exploit thread-level parallelism because multiple cooperating threads operate in parallel.
  - Loosely-coupled MIMD: clusters and warehouse-scale computers—that exploit request-level parallelism, where many independent tasks can proceed in parallel naturally with little need for communication or synchronization.

Transistors

Transistors and Wires

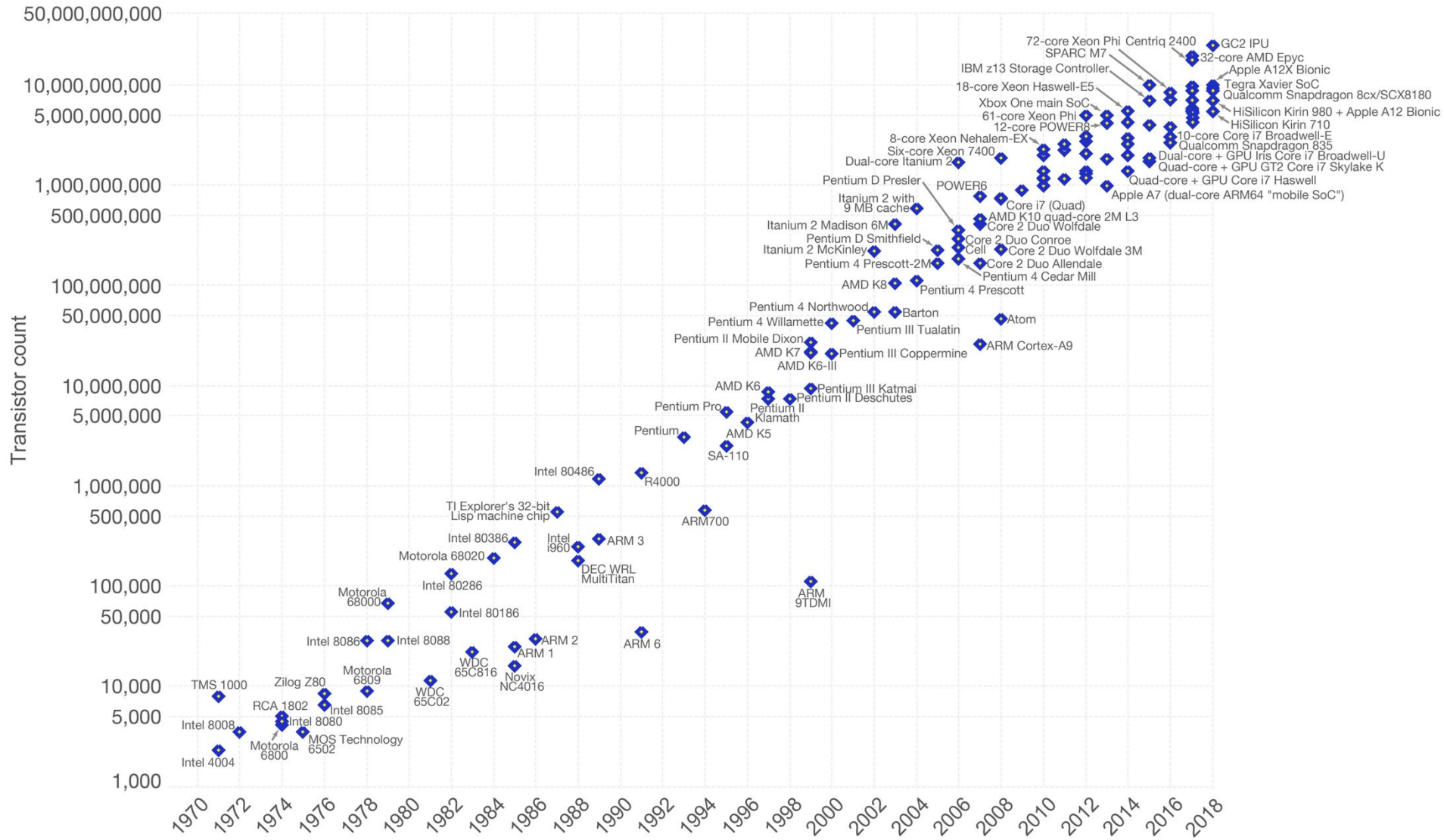
- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - Starting with 10µm in 1971
  - 32nm (‘09) → 22nm (‘12) → 14nm (‘14) → 10nm (’16) → 7nm (‘18) → 5nm
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically



Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

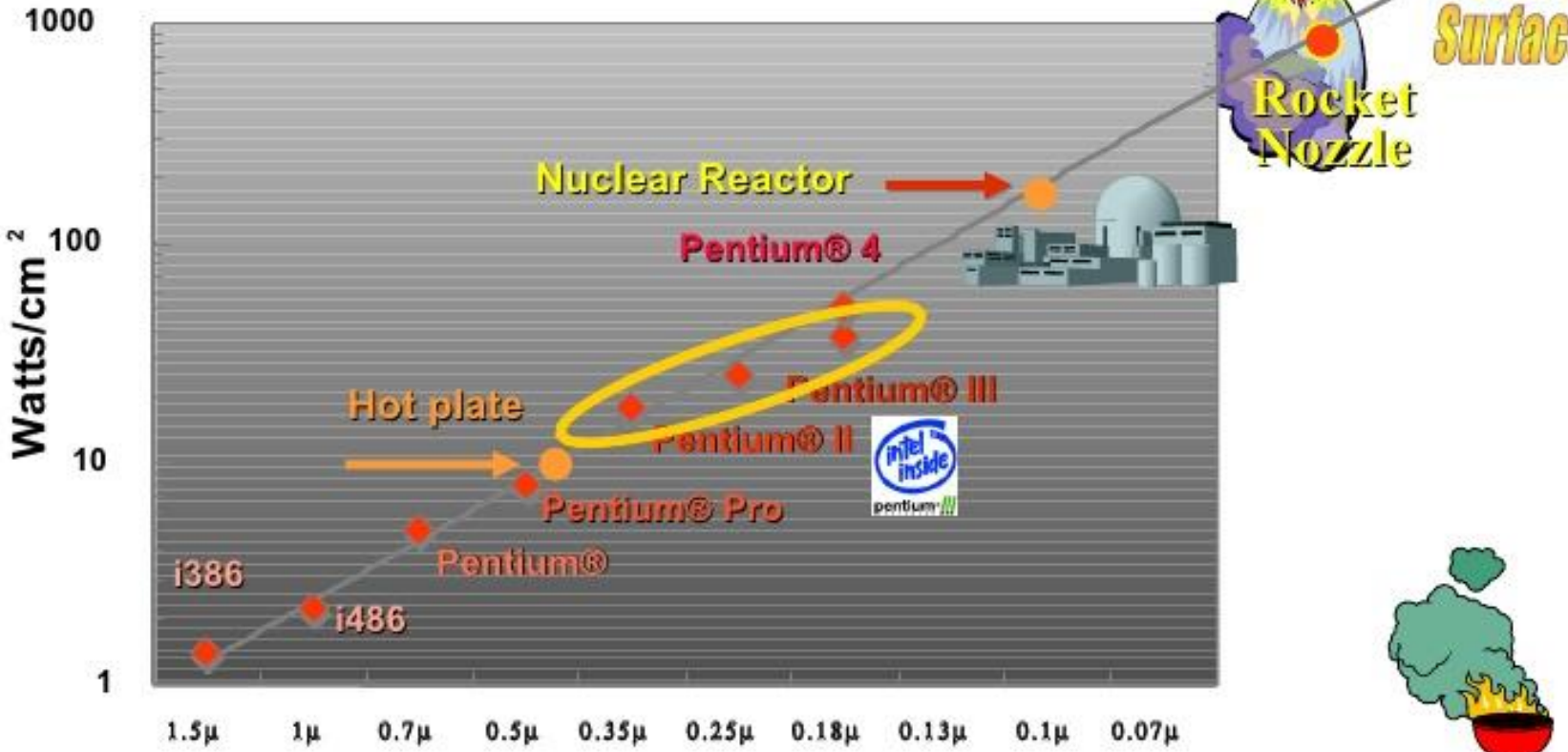
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

Our World in Data



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Power Density



* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference keynote - 1999.

Technology Trends

- Wire delays don’t scale like logic delays
  - Processor structures must expand to support more instructions
  - Thus wire delays dominate the cycle time; slow wires must be local
- Design complexity
  - Processors are becoming so complex that a large fraction of the development of a processor or system is dedicated to verification
  - Chip density is increasing much faster than the productivity of verification engineers (new tools, speed of systems)
- CMOS endpoint
  - CMOS is rapidly reaching the limits of miniaturization
  - Feature sizes will reach atomic dimensions in less than 15 years
- Options????
  - Quantum computing
  - Analog computing
  - Superconducting, spintronics, carbon nanotube, graphene, and more!
- Performance remains a critical design factor

Power and energy

Scenario	Type of Power	Action/Activity	Analogy
Streaming music	Dynamic Power	Circuit switching, CPU active	Dancing to a beat
Phone in standby mode	Static Power	Leakage currents still flowing	Speaker humming in silence

Power

- Total power: dynamic + static (leakage)

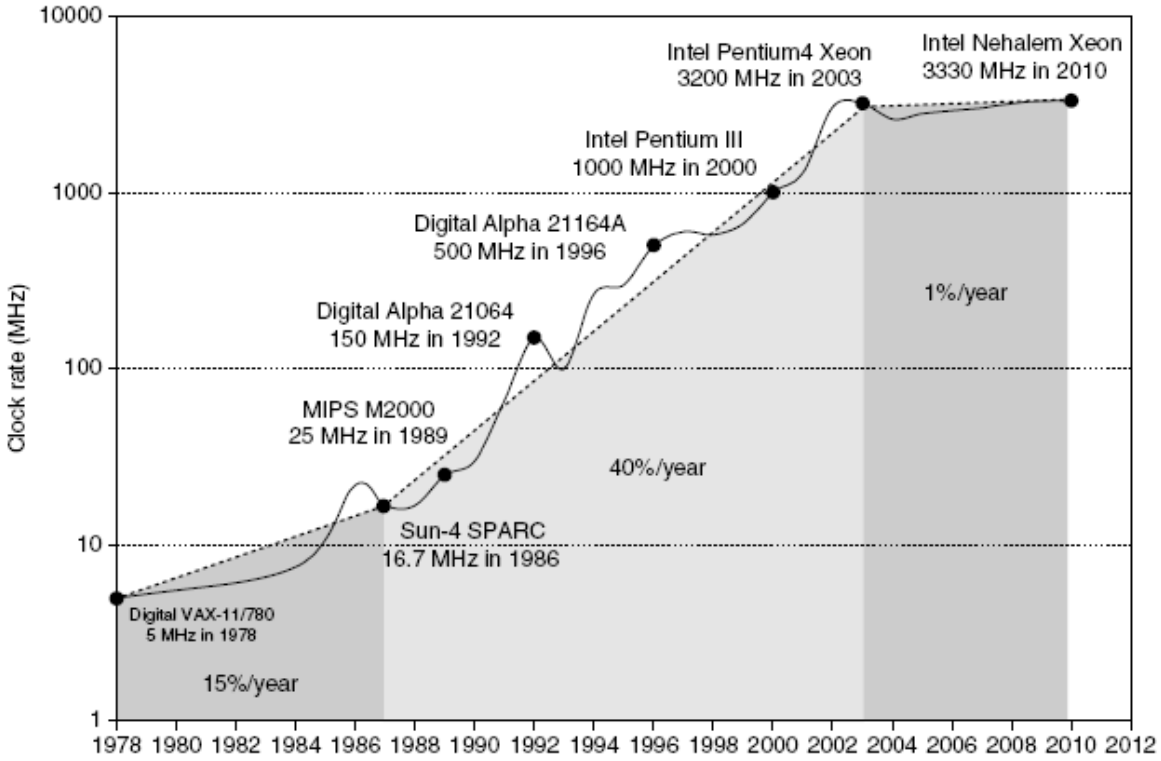
$$P_{dynamic} = \alpha C V^2 f$$

$$P_{static} = V I_{sub} \approx V e^{-KVt/T}$$

- Power/energy are critical problems
  - Power (immediate energy dissipation) must be dissipated
    - Otherwise temperature goes up (affects performance, correctness and may possibly destroy the circuit, short term or long term)
    - Effect on the supply of power to the chip
  - Energy (depends on power and speed)
    - Costly; global problem
    - Battery operated devices

Power Trends

- Intel 80386 consumed ~ 2 W [1985]
- 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



Why multi-core?

$P_{dynamic} = \alpha C V^2 f$

- What uses less power?
  - A) Take a processor and clock it 4 times faster?

    Or

  - B) Take a processor and replicate it 4 times?

Why multi-core?

- $P_{dynamic} = \alpha C V^2 f$

- Dynamic power favors parallel processing over higher clock rate
  - Dynamic power roughly proportional to f^3
  - Take a proc and clock it 4 times faster: 4x speedup but 64x dynamic power!
  - Take a proc. and replicate it 4 times: 4x speedup & 4x power

Static Power

- Because leakage current flows even when a transistor is off, now static power important too

$$Power_{static} = Current_{static} \square Voltage$$

- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- Very low power systems gate voltage to inactive modules to control loss due to leakage

Reducing Power

- Techniques for reducing power:
  - Do nothing well (Idle low power modes)
    - Ex. Processor sleep states (C-states)
  - Dynamic Voltage-Frequency Scaling
    - Ex. Processor P-states
  - Low power state for Caches, DRAM, Disks, etc.
    - Many non-processor components have energy states
  - Overclocking, turning off cores (Race to halt)

Trends in Technology - Density

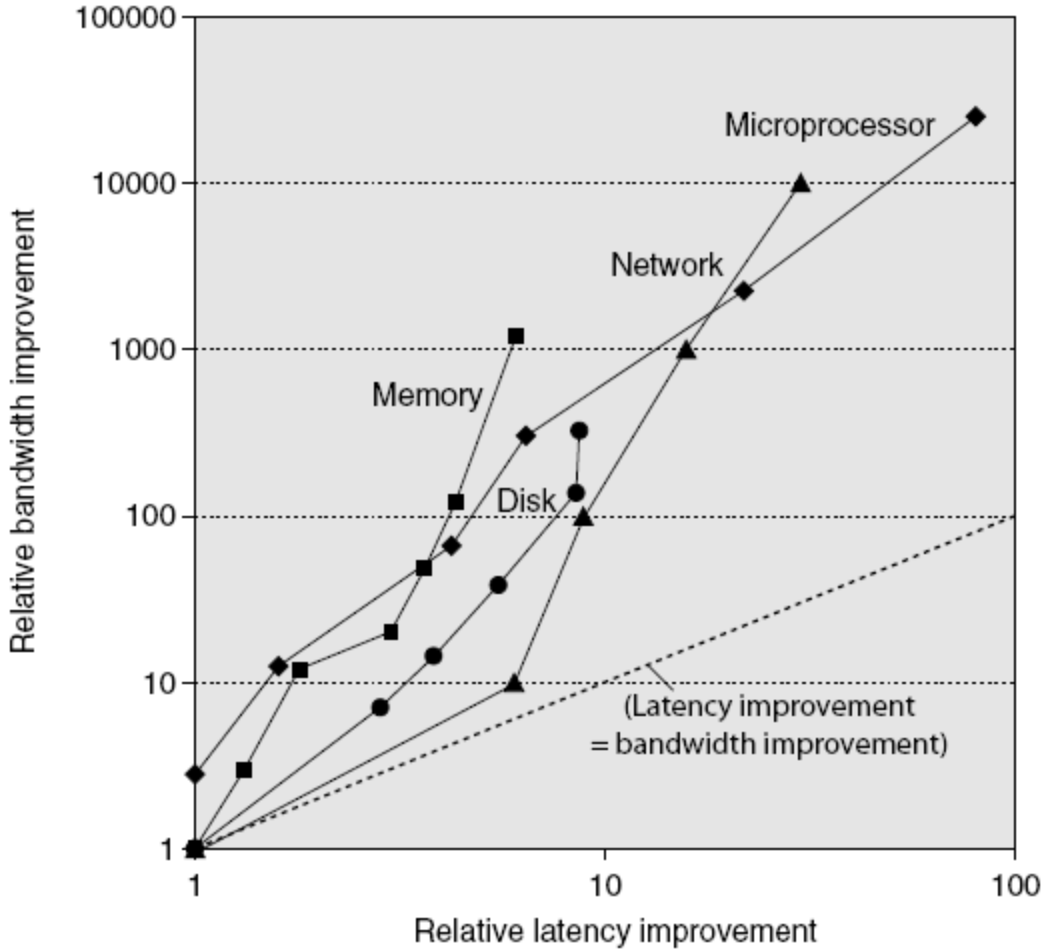
- Integrated circuit technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year
  - 15-25X cheaper/bit then Flash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

Bandwidth and Latency Trends

- 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



Log-log plot of bandwidth and latency milestones

What is performance?

Depends on what you're measuring…

- 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

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…
  aka Execution time

What is performance?

- Performance is defined as:
  - Performance = 1 / Execution Time

- More generally:
  - Speedup = $\text{Performance}_{\text{New}}$ / $\text{Performance}_{\text{Old}}$
    = $\text{Execution}_{\text{Old}}$ / $\text{Execution}_{\text{New}}$

Execution time

Performance

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
- Two common measurements:
  - Wall Clock Time
  - CPU Time

Measuring Execution Time

- Wall Clock Time
  - Real time to complete job (seconds)

- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time

```
$ time make > /dev/null 2>&1
real     1m14.115s
user     0m57.853s
sys      0m10.853s
```

Relative Performance

- "X is n time faster than 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

Relative Performance

- "X is n time faster than Y"
- Execution time_Y / Execution time_X = n

  - Example: time taken to run a program
    - 60s on A, 30s on B
    - Execution Time_B / Execution Time_A= 30s / 60s = 0.5
      So A is 0.5 times faster than B
    - or B is 2 times faster than A

CPI Equation

Performance

CPU Time

- We have seen How to decide between feasible optimizations
- Next question:
  - How do we measure execution time?
  - Which are the determinant factors?

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

  - Clock cycle time is the reciprocal of frequency or 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?

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

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?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

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

CPI Equation

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

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

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

- Execution time equally depends on three components
- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware or architecture
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Performance Summary

The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructio ns}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instructio n}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on

	IC	CPI	T_c
Programming language, Algorithm, Compiler	x		
Instruction set architecture	x	x	
Semiconductor Technology used			x
Processor Architecture		x	x

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?

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

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

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

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 types take different numbers of cycles

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

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instructio n Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \frac{\text{Instructio n Count}_i}{\text{Instructio n Count}} \right)$$

Relative frequency

CPI Example

- Alternative compiled code sequences using instructions in type INT, FP, MEM

Type	INT	FP	MEM
CPI for type	1	2	3
IC in Program 1	2	1	2
IC in Program 2	4	1	1

CPI Example

- Alternative compiled code sequences using instructions in type INT, FP, MEM

Type	INT	FP	MEM
CPI for type	1	2	3
IC in Program 1	2	1	2
IC in Program 2	4	1	1

- Program 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0

CPI Example

- Alternative compiled code sequences using instructions in type INT, FP, MEM

Type	INT	FP	MEM
CPI for type	1	2	3
IC in Program 1	2	1	2
IC in Program 2	4	1	1

- Program 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Program 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

CPI Equation Example: Which one is better?

- The Current GPU doesnot have FPSQRT instruction and FPSQRT is emulated on software (large CPI)
- Frequency of FP instructions is 25%
- Average CPI of these instructions is 4.0
- Average CPI of non-FP instructions is 1.33
- Frequency of FPSQRT operations is 2%
- CPI of FPSQRT operation is 20.0
- One Design alternative is to reduce CPI of FPSQRT to 2
- Other Design alternative is to reduce CPI of all FP operations to 2.0.

CPI Equation Example: Which one is better?

- The Current GPU doesnot have FPSQRT instruction and FPSQRT is emulated on software (large CPI)
- Frequency of FP instructions is 25%
- Average CPI of these instructions is 4.0
- Average CPI of non-FP instructions is 1.33
- Frequency of FPSQRT operations is 2%
- CPI of FPSQRT operation is 20.0
- One Design alternative is to reduce CPI of FPSQRT to 2
- Other Design alternative is to reduce CPI of all FP operations to 2.0.

Instruction Type	all FP	non- FP	FPSQRT
CPI for type	4 (Design$_1$: 2)	1.33	20 (Design$_2$: 2)
IC or frequency	25%	? = 75%	2%

$Time_{default}$ =	0.23 x 4 x t (= 0.92t) +	0.75 x 1.33 x t (= x) +	0.02 x 20 x t (= 0.4t)	
$Time_{Design\ 1}$ =	0.23 x 2 x t (=0.46t) +	x +	0.4t	0.86t + x
$Time_{Design\ 2}$ =	0.92t +	x +	0.02 x 2 x t (=0.04t)	0.96t + x

CPI Equation Example: Which one is better?

- The Current GPU doesnot have FPSQRT instruction and FPSQRT is emulated on software (large CPI)
- Frequency of FP instructions is 25%
- Average CPI of these instructions is 4.0
- Average CPI of non-FP instructions is 1.33
- Frequency of FPSQRT operations is 2%
- CPI of FPSQRT operation is 20.0
- One Design alternative is to reduce CPI of FPSQRT to 2
- Other Design alternative is to reduce CPI of all FP operations to 2.0.

Instruction Type	all FP	non- FP	FPSQRT
CPI for type	4 (Design$_1$: 2)	1.33	20 (Design$_2$: 2)
IC or frequency	25%	? = 75%	2%

$Time_{default}$ =	0.23 x 4 x t (=0.92t) +	0.75 x 1.33 x t +	0.02 x 20 x t (=0.4t)
$Time_{Design\ 1}$ =	Least Execution Time => Better Design		
$Time_{Design\ 2}$ =	0.92t +	0.75 x 1.33 x t +	0.02 x 2 x t (=0.04t)

Amdahl's Law

- Gives theoretical speedup of a fixed program when a resource is improved
- Speedup is due to enhancement (E)

Let f be the fraction where enhancement is applied
s is speedup by enhancement

- f = parallel fraction and (1-f) = serial fraction

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Amdahl's Law

- Assume we have a program that runs for 10 seconds.
  20% of the execution time is spent doing multiply operations.
  You design a multiply unit that can speed up multiply operations by 4x.

  What is the theoretical speedup that you can expect?

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Amdahl's Law

$ExecutionTime_{baseline}$

- Assume we have a program that runs for 10 seconds. 20% of the execution time is spent doing multiply operations. You design a multiply unit that can speed up multiply operations by 4x.

  What is the theoretical speedup that you can expect?

$$Speedup_{enhanced} = \frac{1}{(1-0.2)+\frac{0.2}{4}} = 1.176$$

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Amdahl's Law

- Assume we have a program that runs for 10 seconds.
  20% of the execution time is spent doing multiply operations.
  You want to design a new multiply unit so that your overall speedup is 4x.

  What is the speedup that your multiply unit need to achieve?

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Amdahl's Law

- Assume we have a program that runs for 10 seconds.
  20% of the execution time is spent doing multiply operations.
  You want to design a new multiply unit so that your overall speedup is 4x.

  What is the speedup that your multiply unit need to achieve?

$$Speedup_{enhanced} = \frac{1}{(1-0.2)+\frac{0.2}{s}} = 4$$

s = -0.3636, can't be done!

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1-f)+\frac{f}{s}}$$

Amdahl's Law

- Law of diminishing returns
  - Program with execution time T
  - Fraction f of the program can be sped up by a factor s
  - New execution time, enhanced, is T_e

$$T_e = T\left[(1-f)+\frac{f}{s}\right]$$

$$Speedup = \frac{T}{T_e} = \frac{T}{T\left[(1-f)+\frac{f}{s}\right]} = \frac{1}{\left[(1-f)+\frac{f}{s}\right]} \quad s \square \ \times, Speedup \square \ \frac{1}{1-f}$$

- Optimize the common case!

Amdahl's Law

- Makes the common case fast
- Intuition: no point investing time and effort on the portion of the code that is invoked only 1% of time.

t = time taken to execute program on a processor

x = fraction of the program to be optimized

y = speed up of the optimized part of the program.

Amdahl's Law

$$\text{Speed up} = \frac{t}{t_{new}} = \frac{t}{t - tx + tx/y} = \frac{1}{1 - x + x/y}$$

$$\text{Speed up} = \frac{1}{1 - x + x/y}$$

Amdahl's Law: Choose Design Alternative

- Look for portions of the program that takes maximum amount of time to execute.
- Allocate resources and design time proportional to execution time
- Used to compare design alternatives i.e. Which Design would bring more performance?

$$\text{Speed up} = \frac{1}{1 - x + x/y}$$

Amdahl's Law: Example

Q: FP square root is critical in graphics applications:

Two Design choices: implement a FP square root hardware to improve sqrt execution by 10 times or improve all FP instructions by 2 times; suppose FP sqrt takes 20% of execution time while 50% time is spent in all FP instructions in the current processor; which design is better?

$$\text{Speed up} = \frac{1}{1 - x + x/y}$$

Amdahl's Law: Example

Q: FP square root is critical in graphics applications:

Two Design choices: implement a FP square root hardware to improve sqrt execution by 10 times or improve all FP instructions by 2 times; suppose FP sqrt takes 20% of execution time while 50% time is spent in all FP instructions in the current processor; which design is better?

Design 1: **FP sqrt 20% time, 10x speedup**	**Design 2:** **all FP 50% time, 2x speed up**
x= 0.2, y=10 Speed up = $\frac{1}{1 - 0.2 + 0.2/10} = \frac{1}{0.82}$	x= 0.5, y=2 Speed up = $\frac{1}{1 - 0.5 + 0.5/2} = \frac{1}{0.75}$

Better choice

Amdahl's Law: Get max speedup in a parallel computer

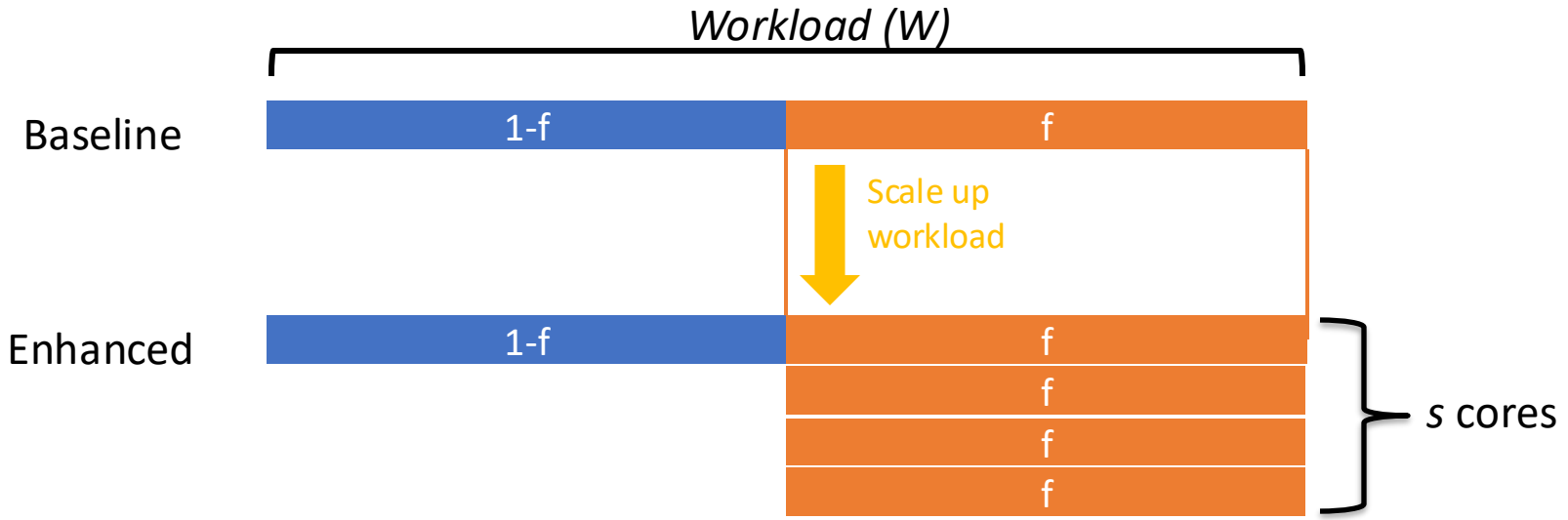
- Suppose a sequential program takes time "**t**" to run on a single processor
- A profiler shows that a fraction "**s**" of this time is spent in executing inherently sequential portions of the program
- The remaining time can be perfectly parallelized on arbitrary number of processors **"P"**
- Maximum achievable speedup

$$\text{Speed up} = \frac{t}{s*t+(1-s)t/P} = \frac{1}{s+(1-s)/P}$$

  - When P $\rightarrow \infty$, $\text{Speedup} \rightarrow 1/s$
  - For a 95% parallelizable program, s = 0.05, Speedup < 20 (ignoring the communication overhead.)

Gustafson's Law

- Gives theoretical speedup of a program at *fixed execution time* when a resource is improved.
  - Ex. As more cores are integrated, the workload is also growing!



Let f be the fraction where enhancement is applied
s is speedup by enhancement

$$Speedup_{enhanced} = \frac{Workload_{enhanced}}{Workload_{baseline}} = \frac{(1-f)W+fWs}{(1-f)W+fW} = 1 - f + fs$$

Gustafson's Law

- Assume we have a program that runs for 10 seconds.
  20% of the execution time is spent doing multiply operations.
  You decide to increase the number of multiply units in the process to 4 multiply units from 1 multiply unit.

  What is the theoretical speedup that you can expect?

$$Speedup_{enhanced} = \frac{Workload_{enhanced}}{Workload_{baseline}} = \frac{(1-f)W+fWs}{(1-f)W+fW} = 1 - f + fs$$

Gustafson's Law

- Assume we have a program that runs for 10 seconds.
  20% of the execution time is spent doing multiply operations.
  You decide to increase the number of multiply units in the process to
  4 multiply units from 1 multiply unit.

  What is the theoretical speedup that you can expect?

$$Speedup_{enhanced} = 1 - 0.2 + 0.2 \times 4 = 1.6$$

$$Speedup_{enhanced} = \frac{Workload_{enhanced}}{Workload_{baseline}} = \frac{(1-f)W+fWs}{(1-f)W+fW} = 1 - f + fs$$

Amdahl’s Law vs Gustafson’s Law

Amdahl’s Law: The Bottleneck Perspective

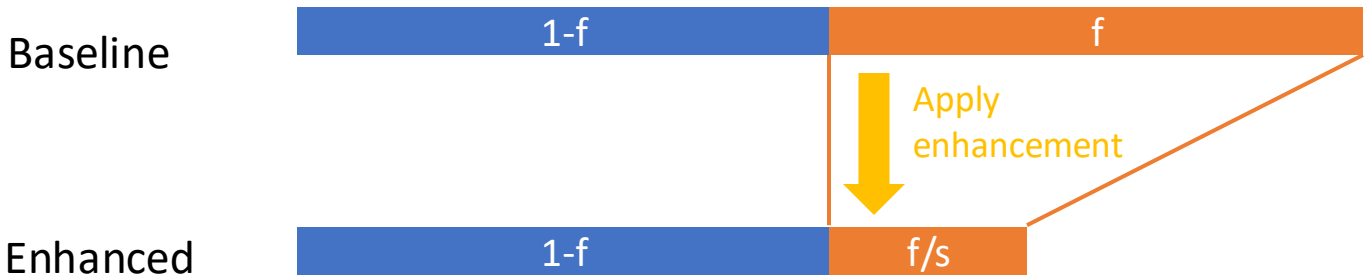
- **Analogy:** Imagine you're organizing a group of people to paint a house. Most of the house can be painted by many people at once (parallel work), but the trim around the windows can only be done by one person at a time (serial work). No matter how many painters you add, the trim still takes the same amount of time. That trim becomes the bottleneck.
- **Key Idea:** Amdahl’s Law says the **maximum speedup** of a task is limited by the portion that **cannot be parallelized**.
- **Formula:** Speedup = 1/ (1-P + P/N)
  - P = parallelizable portion of the task
  - N = number of processors
- **Example:** If 90% of a program can be parallelized and you use 10 processors:
- Speedup = 1 / (0.1 + 0.9/10) ≈ 5.26
- Even with 10 processors, you don’t get a 10x speedup because of 10% that’s serial.

Gustafson's Law: The Scaling Perspective

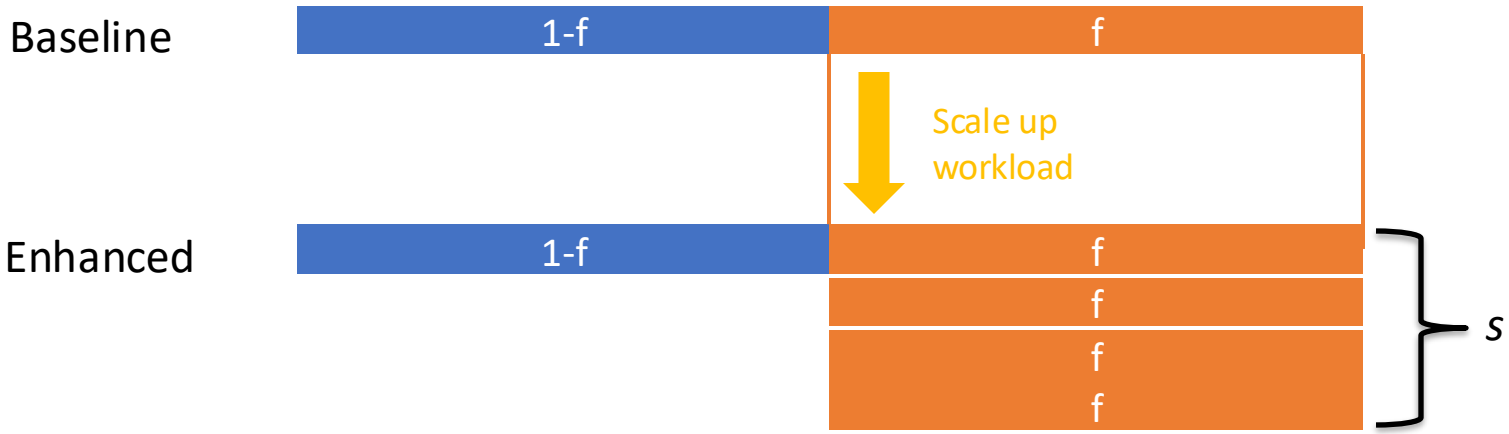
- **Analogy:** Now imagine instead of painting a small house faster, you decide to paint a **mansion** with your team. The trim still takes one person, but now the walls are much larger, and your team can paint more of them in parallel. The serial part stays the same, but the parallel part grows—so your team's impact scales up.
- **Key Idea:** Gustafson's Law says that as **problem size increases**, the **parallel portion dominates**, making parallel computing more effective.
- **Formula:** Speedup = 1-P + P * N = S + P * N
  - P = parallel fraction or parallelizable portion of the task
  - S = serial fraction = 1 - P
  - N = number of processors
- **Example:** If only 10% is serial and you use 10 processors:
- Speedup = 0.1+0.9×10 = 0.1+9 = 9.1
- Much better than Amdahl's result for the same setup!

Amdahl's Law vs Gustafson's Law

- Amdahl's Law



- Gustafson's Law



Concluding Remarks

- Execution time: the best performance measure
- Amdahl's law helps choose a better design alternative and derive upper bound of maximum achievable speedup in a parallel computer.
- Execution time equally depends on three components: IC, CPI and T_c