

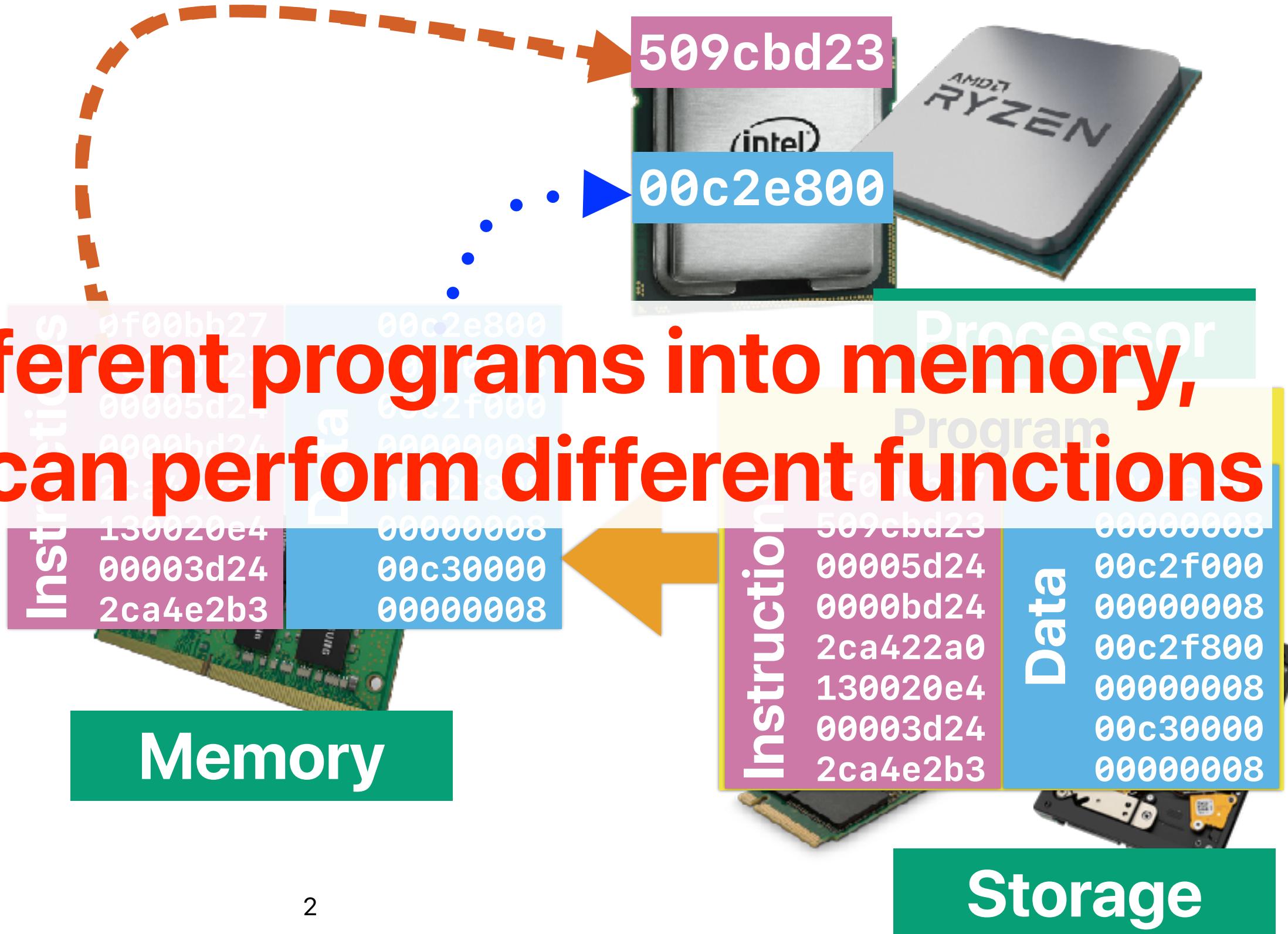
Performance

Hung-Wei Tseng

Recap: von Neumann Architecture



By loading different programs into memory,
your computer can perform different functions



Recap: Start with this simple program in C

```
int A[] =  
{1,2,3,4,5,6,7,8,9,10,1,2,3,4  
,5,6,7,8,9,10};
```

Compiler

Contents of section .data:
0000 01000000 02000000 03000000 04000000
0010 05000000 06000000 07000000 08000000
0020 09000000 0a000000 0b000000 0c000000
0030 03000000 04000000 05000000 06000000
0040 07000000 08000000 09000000 0a000000

control flow
operations
logical operations

```
int main()  
{  
    int i=0, sum=0;  
    for(i = 0; i < 20; i++)  
    {  
        sum += A[i];  
    }  
    return 0;  
}
```

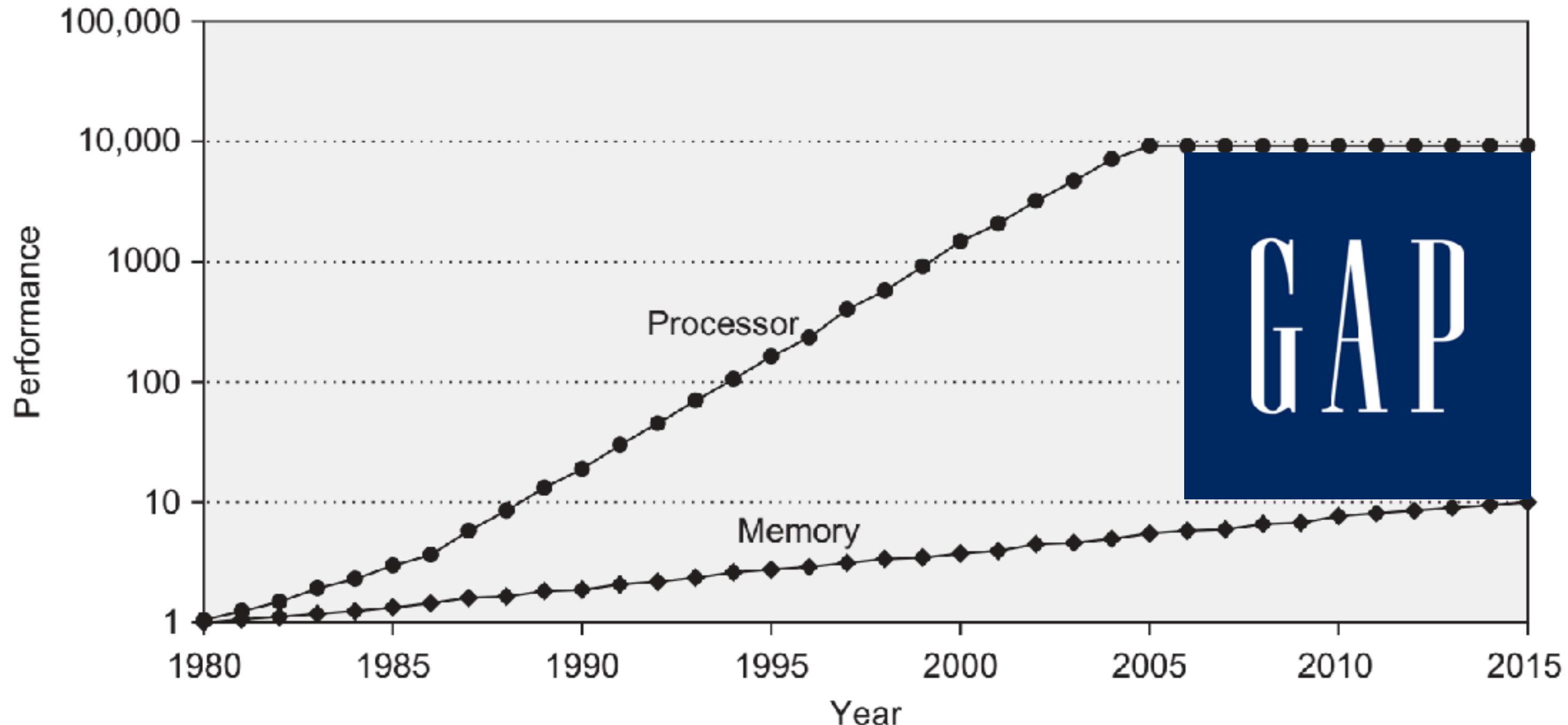
memory access
arithmetic operations

Compiler

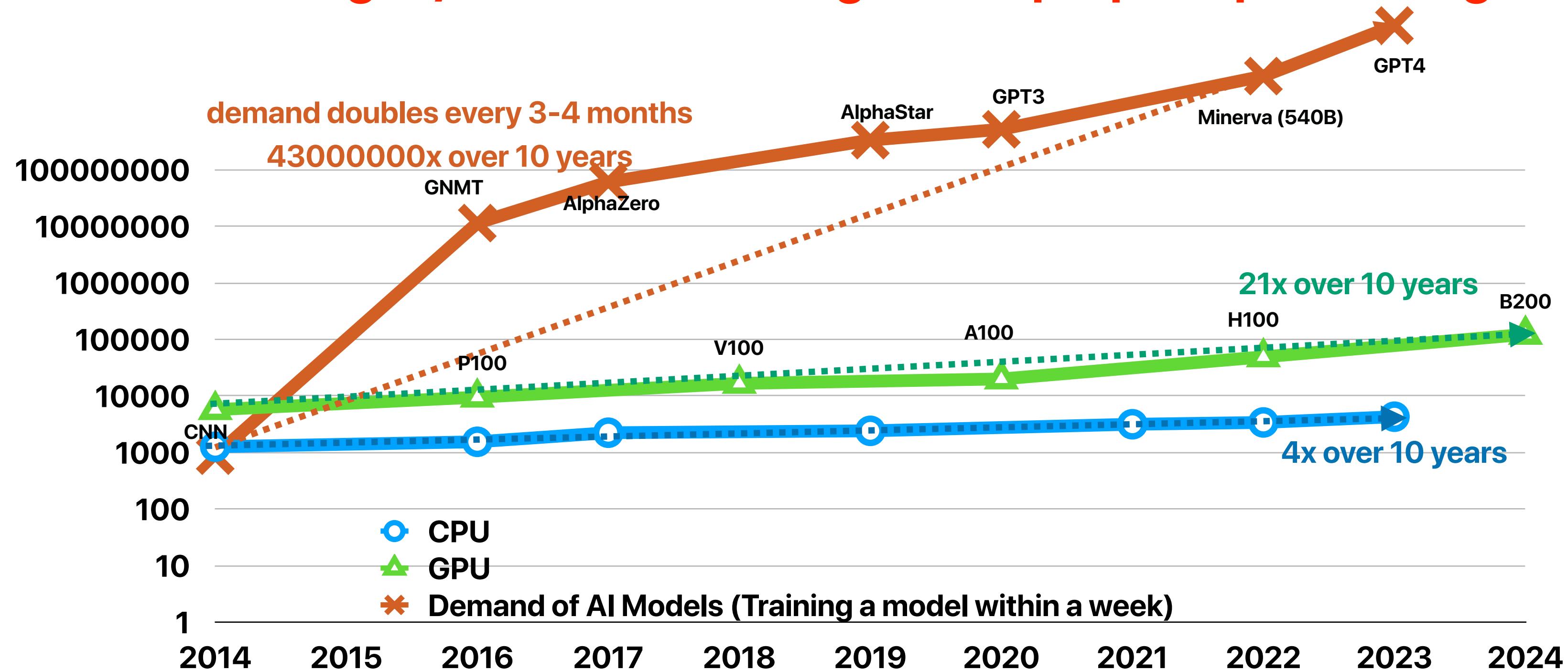
```
main:  
.LFB0:  
    endbr64  
    pushq %rbp  
    movq %rsp, %rbp  
    movl $0, -8(%rbp)  
    movl $0, -4(%rbp)  
    movl $0, -8(%rbp)  
    jmp .L2  
.L2:  
    cmpl $19, -8(%rbp)  
    jle .L3  
    movl $0, %eax  
    popq %rbp  
    ret  
.L3:  
    movl -8(%rbp), %eax  
    cltq  
    leaq 0(%rax,4), %rdx  
    leaq A(%rip), %rax
```

Contents of section .text:
0000 f30f1efa 554889e5 c745f800 000000c7
0010 45fc0000 0000c745 f8000000 00eb1e8b
0020 45f84898 488d1405 00000000 488d0500
0030 0000008b 04020145 fc8345f8 01837df8
0040 137edcb8 00000000 5dc3

Recap: Performance gap between Processor/Memory



Mis-matching AI/ML demand and general-purpose processing



<https://ourworldindata.org/grapher/artificial-intelligence-training-computation>

**What does it really mean by
“better” performance**

ChatGPT

chat.openai.com

ChatGPT 3.5

You
What are the most popular topics in computer science?

ChatGPT

Message ChatGPT...

ChatGPT can make mistakes. Consider checking important information.

Gemini

gemini.google.com...

Gemini

See the latest updates to the Gemini Apps Preview Hub

Gemini

Hello, Hung-Wei
How can I help you today?

Revise my writing and fix my grammar

Teach me the concept of game theory in simple terms

Help me plan a game night with 5 friends for under \$100

Your conversations are processed by human reviewers to improve the technologies powering Gemini Apps. Don't enter anything you wouldn't want reviewed or used.

How it works Dismiss

What are the most popular topics in computer science?

Gemini may display inaccurate info, including about people, so double-check its responses. Your privacy Apps

Submit

Important performance metrics

- End-to-end latency — how much time the program/operation takes from the beginning to the end
- Response time — how much time the user starts to feel the program is running/finishing
- Throughput/bandwidth — the average amount of work/data can the program/system deliver within the execution time
- Energy consumption — the aggregated power during the execution time
- Cost of operation — the amount of money necessary for finishing an operation
- Quality of results — the human perception of the execution result
- Power consumption — the heat generation produced by the circuit

Important performance metrics

- End-to-end latency — how much **time** the program/operation takes from the beginning to the end
- Response time — how much **time** the user starts to feel the program is running/finishing
- Throughput/bandwidth — the average amount of work/data can the program/system deliver within the execution **time**
- Energy consumption — the aggregated power during the execution **time**
- Cost of operation — the amount of money necessary for finishing an operation (related to **time**)
- Quality of results — the human perception of the execution result
- Power consumption — the heat generation produced by the circuit

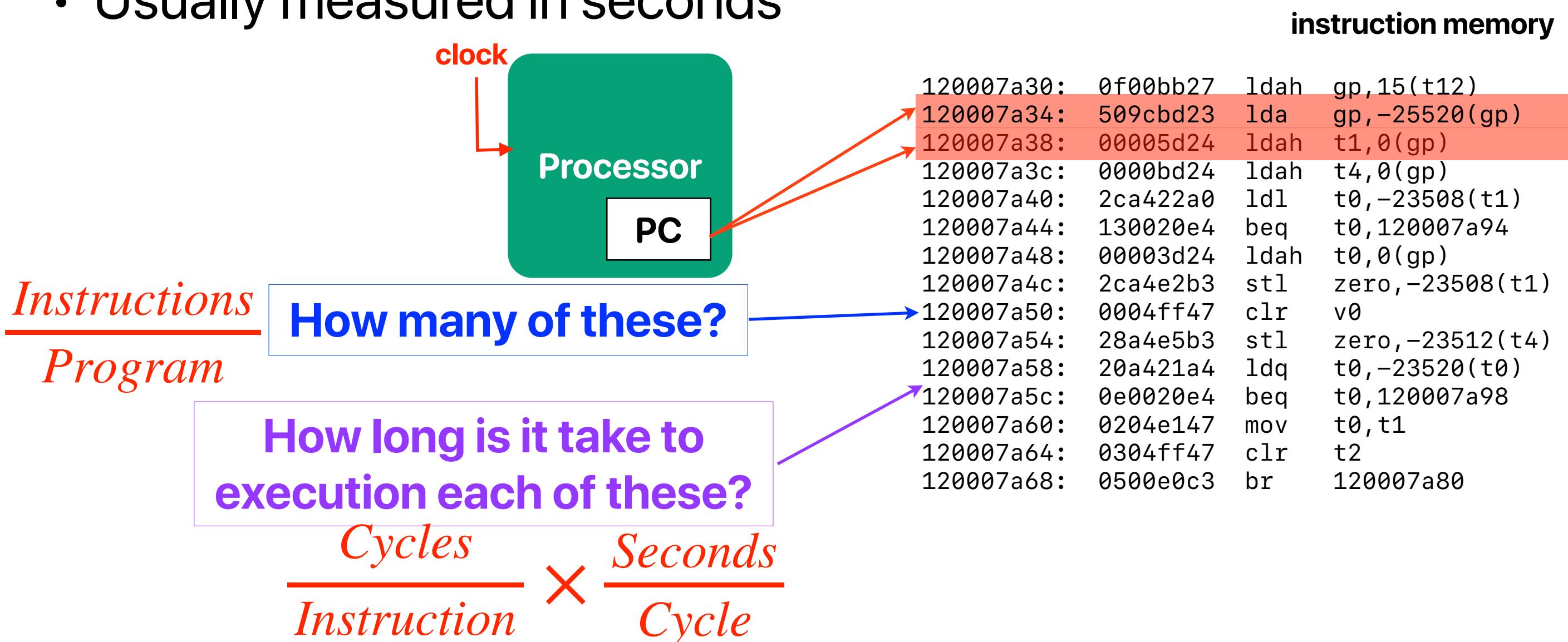
Takeaways: What does “perfect” mean?

- Latency is the most fundamental performance metric

**Let's start with “end-to-end latency”
as the default metric — how long it
takes to execute a program?**

Execution Time

- The simplest kind of performance
- Shorter execution time means better performance
- Usually measured in seconds



CPU Performance Equation

$$Performance = \frac{1}{Execution\ Time}$$

$$Execution\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

$$ET = IC \times CPI \times CT$$

$$1GHz = 10^9Hz = \frac{1}{10^9}sec\ per\ cycle = 1\ ns\ per\ cycle$$

$\frac{1}{Frequency(i.e.,\ clock\ rate)}$

Takeaways: What does “perfect” mean?

- Latency is the most fundamental performance metric
- Instruction count, cycles per instruction, cycle time define the latency of execution on CPUs

**Choose the right metric — Latency
v.s. Throughput/Bandwidth**



Artificial Intelligence Computing Leadership from NVIDIA

CLOUD & DATA CENTER

PRODUCTS ▾

SOLUTIONS ▾

APPS ▾

FOR DEVELOPERS

TECHNOLOGIES ▾

Tesla V100

AI TRAINING

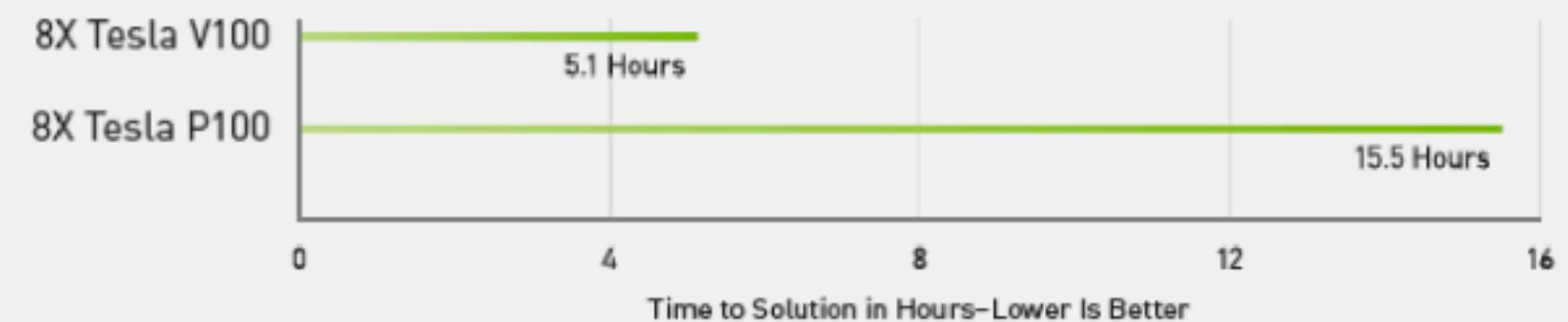
AI INFERENCE

HPC

DATA CENTER GPUs

SPECIFICATIONS

Deep Learning Training in Less Than a Workday



Server Config: Dual Xeon E5-2699 v4 2.6 GHz | 8X NVIDIA® Tesla® P100 or V100 | ResNet-50 Training on MXNet for 90 Epochs with 1.28M ImageNet Dataset.

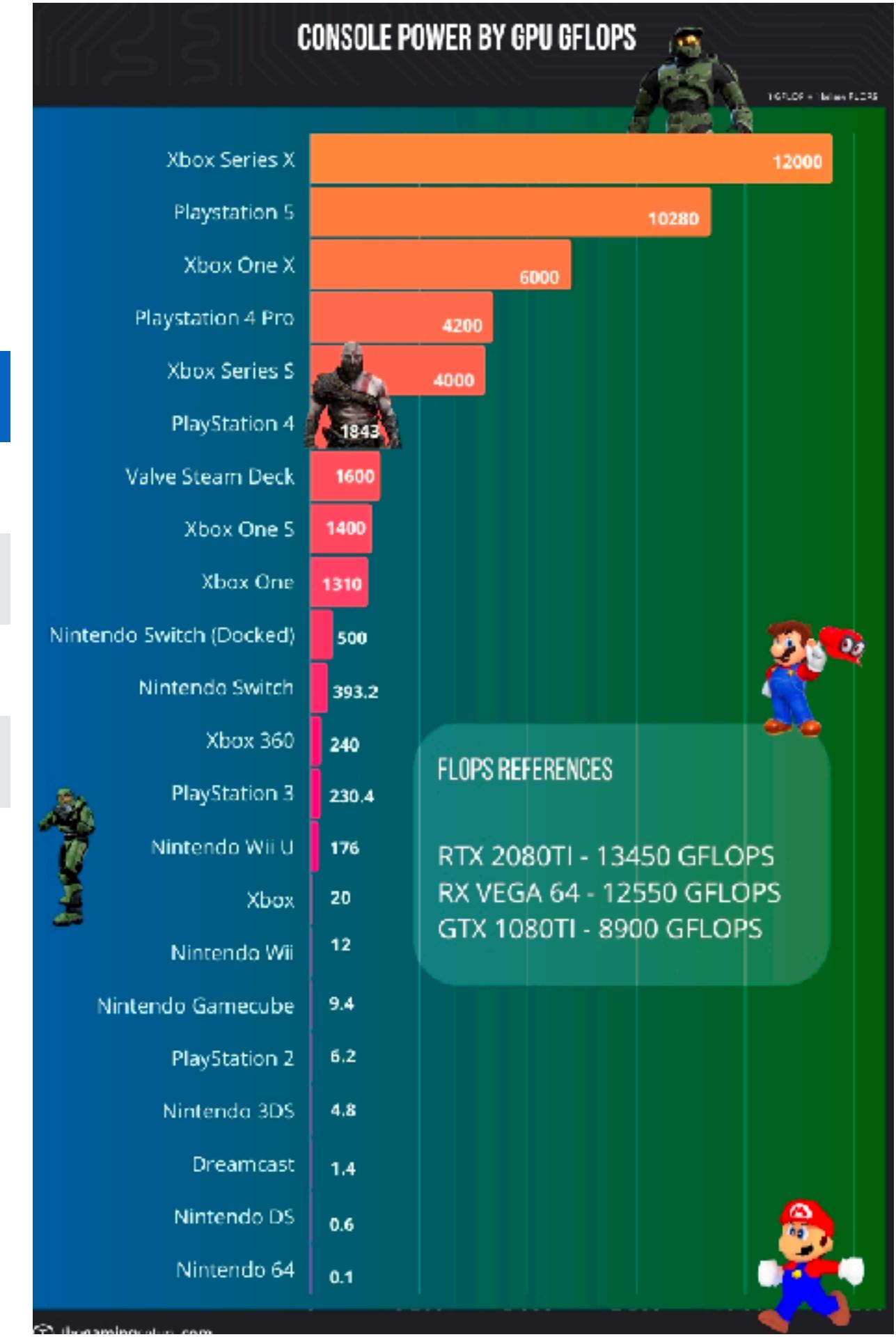
AI TRAINING

From recognizing speech to training virtual personal assistants and teaching autonomous cars to drive, data scientists are taking on increasingly complex challenges with AI. Solving these kinds of problems requires training deep learning models that are exponentially growing in complexity, in a practical amount of time.

With 640 **Tensor Cores**, Tesla V100 is the world's first GPU to break the 100 teraFLOPS (**TFLOPS**) barrier of deep learning performance. The next generation of **NVIDIA NVLink™** connects multiple V100 GPUs at up to 300 GB/s to create the world's most powerful computing servers. AI models that would consume weeks of computing resources on previous systems can now be trained in a few days. With this dramatic reduction in training time, a whole new world of problems will now be solvable with AI.

TFLOPS (Tera FLoating-point Operations Per Second)

	TFLOPS	clock rate
Switch	1	921 MHz
PS5	10.28	2.23 GHz
XBox Series X	12	1.825 GHz
GeForce RTX 3090	40	1.395 GHz



Let's measure the FLOPS of matrix multiplications

```
for(i = 0; i < ARRAY_SIZE; i++) {  
    for(j = 0; j < ARRAY_SIZE; j++) {  
        for(k = 0; k < ARRAY_SIZE; k++) {  
            c[i][j] += a[i][k]*b[k][j];  
        }  
    }  
}
```

Floating point operations per second (FLOP"S"):

Floating point operations (FLOP"s"):

$$i \times j \times k \times 2$$

Given $i = j = k = 2048$

$$2^{3 \times 11} \times 2 = 2^{34} \text{ FLOPs in total}$$

$$FLOPS = \frac{2^{34}}{ET_{seconds}}$$

How reflective is FLOPS?

- If you're given the FLOPS of an underlying GPU, how many situations below can the FLOPS be representative to the real performance?
 - ① The FLOPS remains the same on the same GPU even if we change the data size
 - ② The FLOPS remains the same on the same GPU even if we change the data type to double
 - ③ The FLOPS remains the same on the same GPU if we change the algorithm implementation
 - ④ The ratio of FLOPS on two different GPUs reflects the ratio execution on these two GPUs when executing floating point applications
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

TFLOPS (Tera FLoating-point Operations Per Second)

$$TFLOPS = \frac{\# \text{ of floating point instructions} \times 10^{-12}}{\text{Execution Time}}$$

Demo: matmul on GPU

Size	Latency	Relative Latency	Throughput (Output Numbers Per Second)	Relative Throughput
16x16x16	~ 0.09ms	1	0.09ms/256	1
32x32x32	~ 0.09ms	1	0.09ms/1024	4
64x64x64	~ 0.09ms	1	0.09ms/4096	16

Larger throughput doesn't mean shorter latency!

Is TFLOPS (Tera FLoating-point Operations Per Second) a good metric?

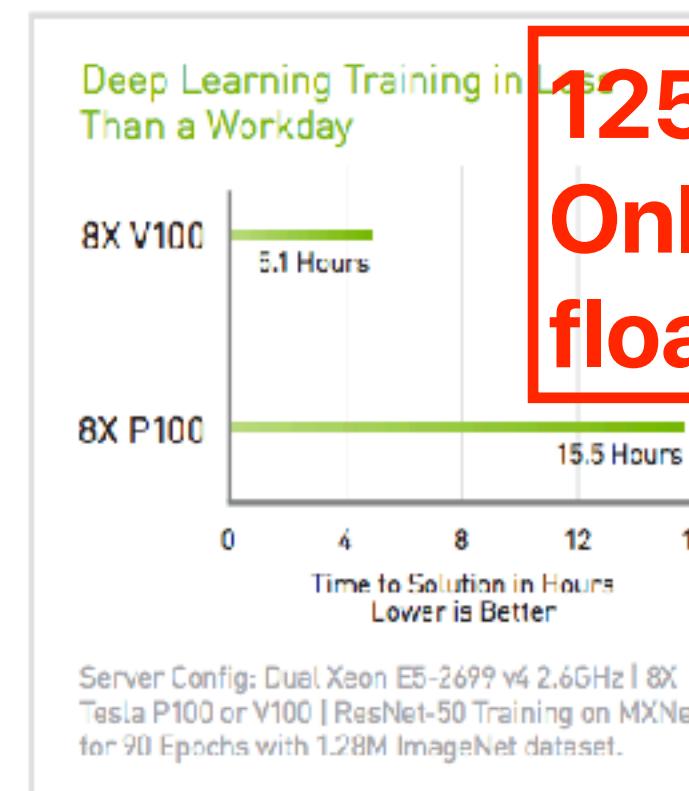
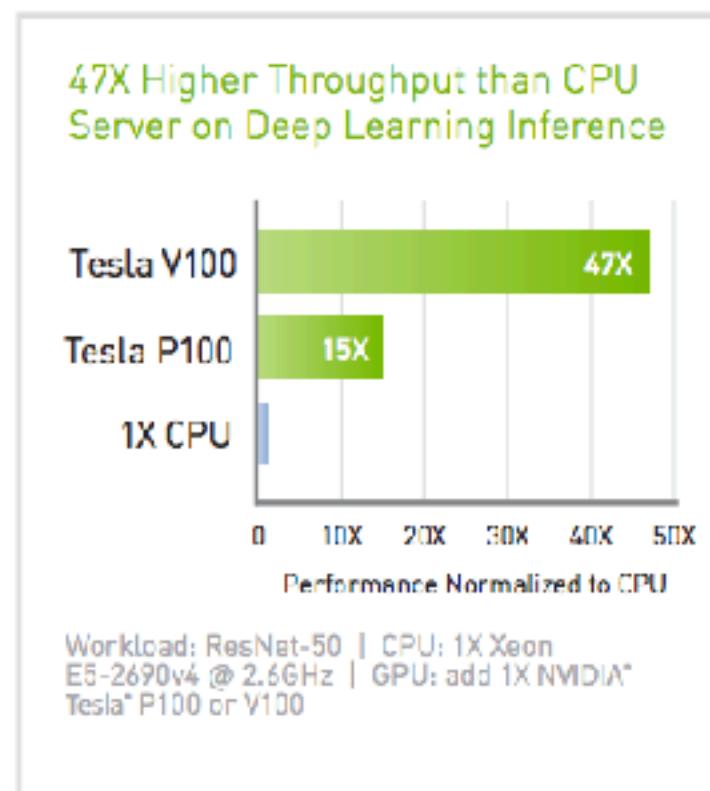
$$\begin{aligned}TFLOPS &= \frac{\# \text{ of floating point instructions} \times 10^{-12}}{\text{Execution Time}} \\&= \frac{IC \times \% \text{ of floating point instructions} \times 10^{-12}}{IC \times CPI \times CT} \\&= \frac{\% \text{ of floating point instructions} \times 10^{-12}}{CPI \times CT}\end{aligned}$$

IC is gone!

- Cannot compare different ISA/compiler
 - What if the compiler can generate code with fewer instructions?
 - What if new architecture has more IC but also lower CPI?
- Does not make sense if the application is not floating point intensive

The Most Advanced Data Center GPU Ever Built.

NVIDIA® Tesla® V100 is the world's most advanced data center GPU ever built to accelerate AI, HPC, and graphics. Powered by NVIDIA Volta, the latest GPU architecture, Tesla V100 offers the performance of up to 100 CPUs in a single GPU—enabling data scientists, researchers, and engineers to tackle challenges that were once thought impossible.



**125 TFLOPS
Only @ 16-bit floating point**

SPECIFICATIONS



**Tesla V100
PCIe**



**Tesla V100
SXM2**

GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power	375W	300W

1 GPU Node Replaces Up To 54 CPU Nodes

Node Replacement: HPC Mixed Workload

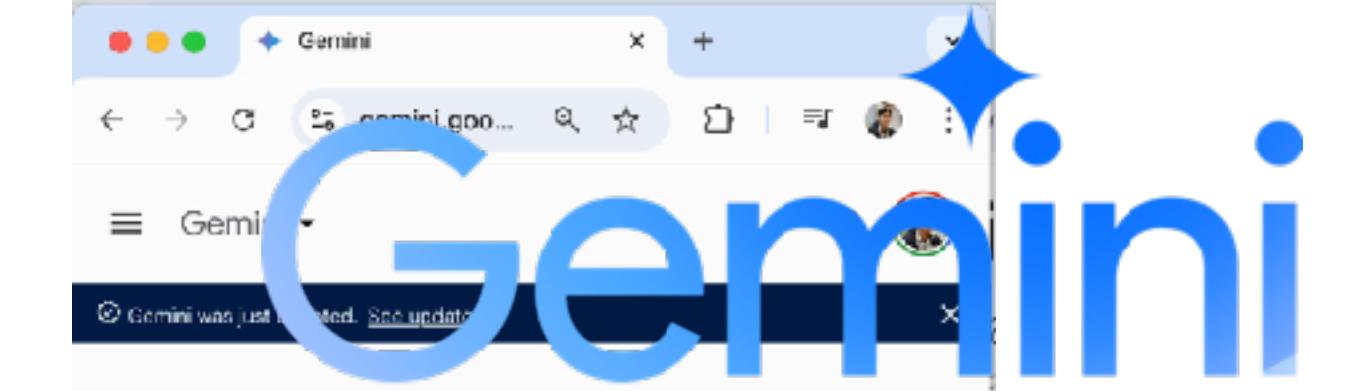


Doesn't matter — they're both wrong :)

1 answer, 9 seconds

Who is teaching CSE142 at UCSD this summer?

ChatGPT 可能會發生錯誤。請查核重要資訊。



Who is teaching CSE142 at UCSD this summer?

Enter a prompt here

Gemini may display inaccurate info, including about people, so double-check its responses.
[Your privacy & Gemini Apps](#)

Nvidia accused of cheating in 3DMar

Futuremark, the maker of 3DMark 03, releases a patch for the game to correct the way Nvidia drivers manipulate results in the popular benchmark.

By **David Becker** on May 23, 2003 at 4:27PM PDT

Saratoga, California-based Futuremark on Friday said in a statement that Nvidia tweaked the software needed to run its new GeForce FX 5900 processor to distort performance in Futuremark's 3DMark 03 testing application. The company said drivers for the new Nvidia chip were altered to detect activity characteristic of a benchmark and adjust performance accordingly.

Recently, there have been questions and some confusion regarding 3DMark 03 results obtained with certain Nvidia products, Futuremark said in the statement. **"We have now established that Nvidia's Detonator FX drivers contain certain detection mechanisms that cause an artificially high score when using 3DMark 03,"** the statement read.

Futuremark has released the version 330 patch for 3DMark 03, which prevents the Nvidia drivers from identifying the benchmark. By Futuremark's measure, the performance of the Nvidia GeForce FX 5900 Ultra drops 24 percent with the patch, compared with a drop of less than 2 percent with ATI's Radeon 9800 Pro with the latest ATI drivers.

A representative at Nvidia questioned the validity of Futuremark's conclusions. "Since Nvidia is not part of the Futuremark beta program (a program which costs of hundreds of thousands of dollars to participate in), we do not get a chance to work with Futuremark on writing the shaders like we would with a real applications developer," the representative said. "We don't know what they did, but it looks like they have intentionally tried to create a scenario that makes our products look bad."

<https://www.gamespot.com/articles/nvidia-accused-of-cheating-in-3dmark-03/1100-6028894/>

#:~:text=%22We%20have%20now%20established%20that,drivers%20from%20identifying%20the%20benchmark.

Nvidia is amid a hard-fought battle with rival ATI Technologies to claim the performance lead in PC graphics processors. After years of Nvidia dominating both in market share and performance, ATI

Latency v.s. Bandwidth/Throughput

- Latency — the amount of time to finish an operation
 - End-to-end execution time of “something”
 - Access time
 - Response time
- Throughput — the amount of work can be done within a given period of time (typically “something” per “timeframe” or the other way around)
 - Bandwidth (MB/Sec, GB/Sec, Mbps, Gbps)
 - IOPs (I/O operations per second)
 - FLOPs (Floating-point operations per second)
 - IPS (Inferences per second)
 - FPS (Frames per second)

Fair comparison in computer architectures

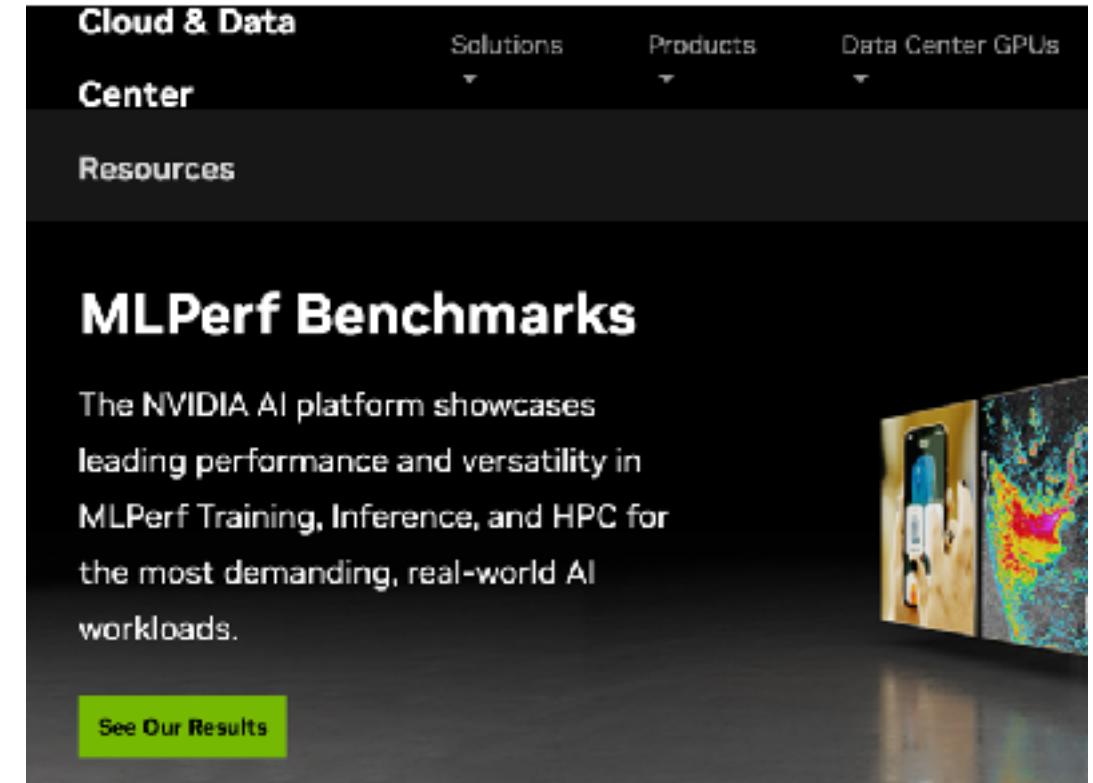
- Metrics: you must consider the fact that performance is composed of IC, CPI, and CT. — any metric that misses one of them is misleading
- Only one variation in each comparison
 - Only change the processor, but not ISA (related to IC) and others
 - Only change the algorithm, but not others
 - The same dataset, must be the same outcome

The reason of “Benchmark Suites”

- Allowing people evaluate systems with exactly the same program and the same inputs and validate results from different machines
- Popular benchmark suites
 - SPEC — CPU benchmark
 - MLPerf — ML systems



The screenshot shows the homepage of the SPEC Standard Performance Evaluation Corporation. The main navigation bar includes links for Home, Benchmarks, Tools, Results, Contact, Blog, Join Us, Search, and Help. A sidebar on the left has sections for Results (Published Results, Results Search, Fair Use Policy), Information (CPU2017, Documentation Overview, System Requirements, Run & Reporting Rules, Using SPEC CPU2017, Resources, Technical Support, Support, FAQ), and Press & Publications. The central content area features a section for SPEC CPU® 2017, describing it as a next-generation, industry-standardized benchmark package for measuring compute intensive performance.

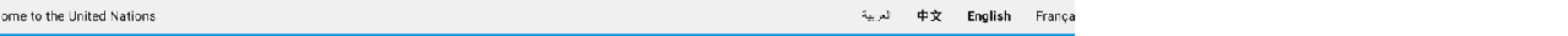


The screenshot shows the NVIDIA Cloud & Data Center website. The top navigation bar includes links for Solutions, Products, and Data Center GPUs. The main content area features a section for MLPerf Benchmarks, stating that the NVIDIA AI platform showcases leading performance and versatility in MLPerf Training, Inference, and HPC for demanding real-world AI workloads. A green button labeled "See Our Results" is visible at the bottom.

Takeaways: What does “perfect” mean?

- Latency is the most fundamental performance metric
- Classic CPU performance equation — Instruction count (IC), cycles per instruction (CPI), cycle time (CT) define the latency of execution on CPUs
- Performance metrics without considering all three factors in the classic performance equation can mislead — anything throughput typically miss one of them

An emerging important performance metric



For a livable climate: Net-zero commitments must be

What is net zero?

Put simply, net zero means cutting carbon emissions to a small amount from nature and other carbon dioxide removal measures, leaving zero in the atmosphere.

Why is net zero important?

The science shows clearly that in order to avert the worst impacts of climate change, global temperature increase needs to be limited to 1.5°C above pre-industrial levels. Current projections show that by 2050, temperatures will have risen by about 1.1°C since the 1800s, and emissions continue to rise. To keep global warming to no more than 1.5°C, greenhouse gas emissions need to be reduced by 45% by 2030 and reach net zero by 2050.

How can net zero be achieved?

Transitioning to a net-zero world is one of the greatest challenges humanity faces. It requires a fundamental transformation of how we produce, consume, and move about. The energy system must be transformed to reduce greenhouse gas emissions today and holds the key to averting the worst effects of climate change. Switching from fossil fuels to energy from renewable sources, such as wind or solar, would dramatically reduce emissions.

Is there a global effort to reach net zero?

Yes, a growing coalition of countries, cities, businesses and other institutions are committing to net-zero. As of January 2021, over 100 countries, including the biggest polluters – China, the United States, India, and the European Union – have committed to net-zero. These countries represent about 88% of global emissions. More than 9,000 companies, over 1000 financial institutions have joined the Race to Zero, pledging to take rigorous action to reach net-zero by 2050 or earlier.



Net-zero carbon



Power v.s. Energy

- Power is the direct contributor of “heat”
 - Packaging of the chip
 - Heat dissipation cost
 - Dynamic power
 - Leakage power
- $\text{Energy} = \text{Power} \times \text{Execution_Time}$
 - The electricity bill and battery life is related to energy!
 - Lower power does not necessarily means better battery life if the processor slow down the application too much

Dynamic/Active Power

- The power consumption due to the switching of transistor states
- Dynamic power per transistor

$$P_{dynamic} \sim \alpha \times C \times V^2 \times f \times N$$

- α : average switches per cycle
- C : capacitance
- V : voltage
- f : frequency, usually linear with V
- N : the number of transistors

Static/Leakage Power

- The power consumption due to leakage — transistors do not turn all the way off during no operation
- Becomes the **dominant** factor in the most advanced process technologies.

$$P_{leakage} \sim N \times V \times e^{-V_t}$$

- N : number of transistors
- V : voltage
- V_t : threshold voltage where transistor conducts (begins to switch)

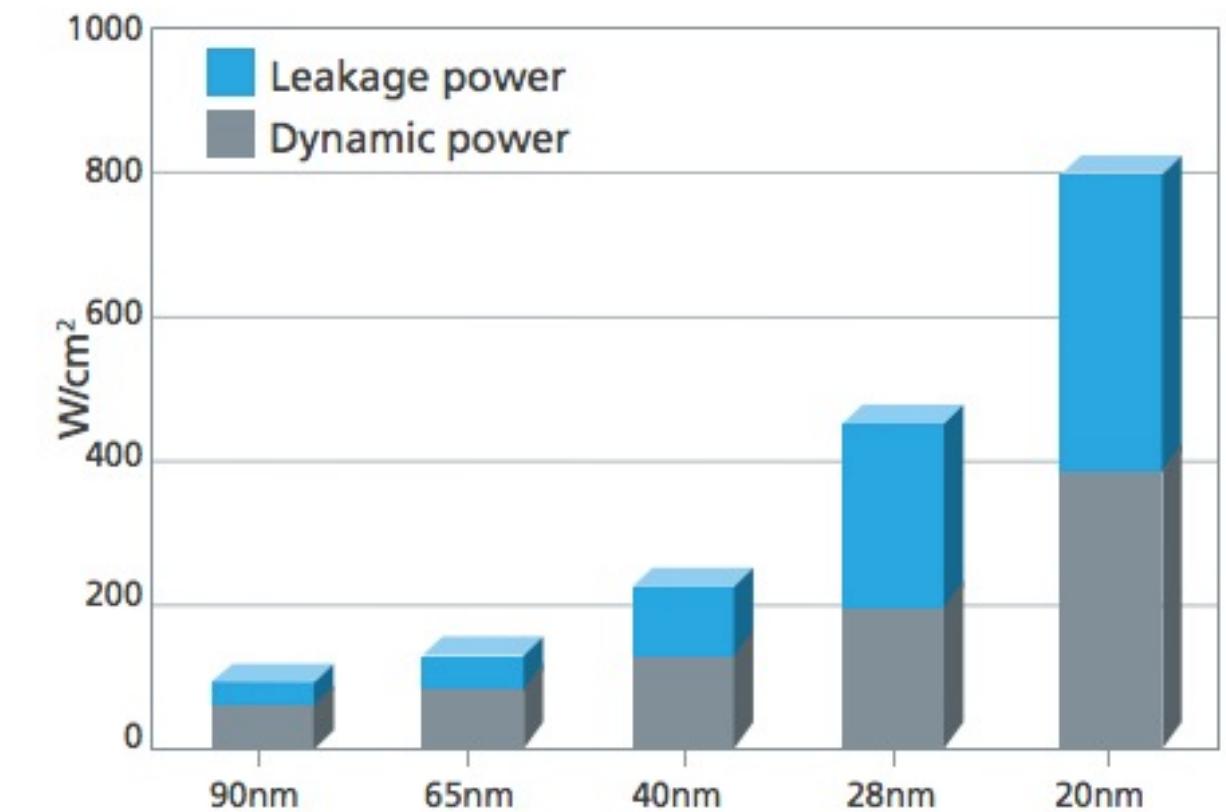


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Demo — changing the max frequency and performance

- Change the maximum frequency of the intel processor — you learned how to do this when we discuss programmer's impact on performance
- LIKWID a profiling tool providing power/energy information
 - likwid-perfctr -g ENERGY [command_line]
 - Let's try blockmm and popcorn and see what's happening!

Takeaways: What does “perfect” mean?

- Latency is the most fundamental performance metric
- Classic CPU performance equation — Instruction count (IC), cycles per instruction (CPI), cycle time (CT) define the latency of execution on CPUs
- Performance metrics without considering all three factors in the classic performance equation can mislead — anything throughput typically miss one of them
- Energy efficiency and power become increasingly important — but latency/execution time still matters as

$$Energy = Power \times Execution_Time$$

What Affects Each Factor in Performance Equation

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)  
{  
    for(j = 0; j < ARRAY_SIZE; j++)  
    {  
        c[i][j] = a[i][j]+b[i][j];  
    }  
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)  
{  
    for(i = 0; i < ARRAY_SIZE; i++)  
    {  
        c[i][j] = a[i][j]+b[i][j];  
    }  
}
```

$O(n^2)$

Complexity

$O(n^2)$

Instruction Count?

Clock Rate

CPI

Use “performance counters” to figure out!

- Modern processors provides performance counters
 - instruction counts
 - cache accesses/misses
 - branch instructions/mis-predictions
- How to get their values?
 - You may use “perf stat” in linux
 - You may use Instruments —> Time Profiler on a Mac
 - Intel’s vtune — only works on Windows w/ intel processors
 - You can also create your own functions to obtain counter values

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)  
{  
    for(j = 0; j < ARRAY_SIZE; j++)  
    {  
        c[i][j] = a[i][j]+b[i][j];  
    }  
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)  
{  
    for(i = 0; i < ARRAY_SIZE; i++)  
    {  
        c[i][j] = a[i][j]+b[i][j];  
    }  
}
```

$O(n^2)$

Same

Same

Better

Complexity

Instruction Count?

Clock Rate

CPI

$O(n^2)$

Same

Same

Worse

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
    for(j = 0; j < ARRAY_SIZE; j++)
    {
        c[i][j] = a[i][j]+b[i][j];
    }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
    for(i = 0; i < ARRAY_SIZE; i++)
    {
        c[i][j] = a[i][j]+b[i][j];
    }
}
```

How many of the following make(s) the performance different between version A & version B?

① IC
 CPI

③ CT

A. 0

B. 1

C. 2

D. 3

Programmer's impact

- By adding the “sort” in the following code snippet, what changes in the performance equation to achieve **better** performance?

```
std::sort(data, data + arraySize);
```

```
for (unsigned c = 0; c < arraySize*1000; ++c) {
    if (data[c%arraySize] >= INT_MAX/2)
        sum++;
}
```

A. CPI

B. IC ←

C. CT

D. IC & CPI

E. CPI & CT

**programmer changes IC as well, but
not in the positive direction**

Programmers can also set the cycle time

<https://software.intel.com/sites/default/files/comment/1716807/how-to-change-frequency-on-linux-pub.txt>

```
=====
Subject: setting CPU speed on running linux system
```

If the OS is Linux, you can manually control the CPU speed by reading and writing some virtual files in the "/proc"

1.) Is the system capable of software CPU speed control?

If the "directory" /sys/devices/system/cpu/cpu0/cpufreq exists, speed is controllable.

-- If it does not exist, you may need to go to the BIOS and turn on EIST and any other C and P state control and vi:

2.) What speed is the box set to now?

Do the following:

```
$ cd /sys/devices/system/cpu
$ cat ./cpu0/cpufreq/cpuinfo_max_freq
3193000
$ cat ./cpu0/cpufreq/cpuinfo_min_freq
1596000
```

3.) What speeds can I set to?

Do

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

It will list highest settable to lowest; example from my NHM "Smackover" DX58SO HEDT board, I see:

```
3193000 3192000 3059000 2926000 2793000 2660000 2527000 2394000 2261000 2128000 1995000 1862000 1729000 1596000
```

You can choose from among those numbers to set the "high water" mark and "low water" mark for speed. If you set "h

4.) Show me how to set all to highest settable speed!

Use the following little sh/ksh/bash script:

```
$ cd /sys/devices/system/cpu # a virtual directory made visible by device drivers
$ newSpeedTop=`awk '{print $1}' ./cpu0/cpufreq/scaling_available_frequencies`
$ newSpeedLcw=$newSpeedTop # make them the same in this example
$ for c in ./cpu[0-9]* ; do
>   echo $newSpeedTop >${c}/cpufreq/scaling_max_freq
>   echo $newSpeedLow >${c}/cpufreq/scaling_min_freq
> done
$
```

5.) How do I return to the default - i.e. allow machine to vary from highest to lowest?

Edit line # 3 of the script above, and re-run it. Change the line:

```
$ newSpeedLcw=$newSpeedTop # make them the same in this example
```

To read

How programmer affects performance?

- Performance equation consists of the following three factors

① IC

② CPI

③ CT

How many can a **programmer** affect?

- A. 0
- B. 1
- C. 2
- D. 3

Takeaways: What matters?

- Programmers can control all three factors in the classic performance equation

Programming languages

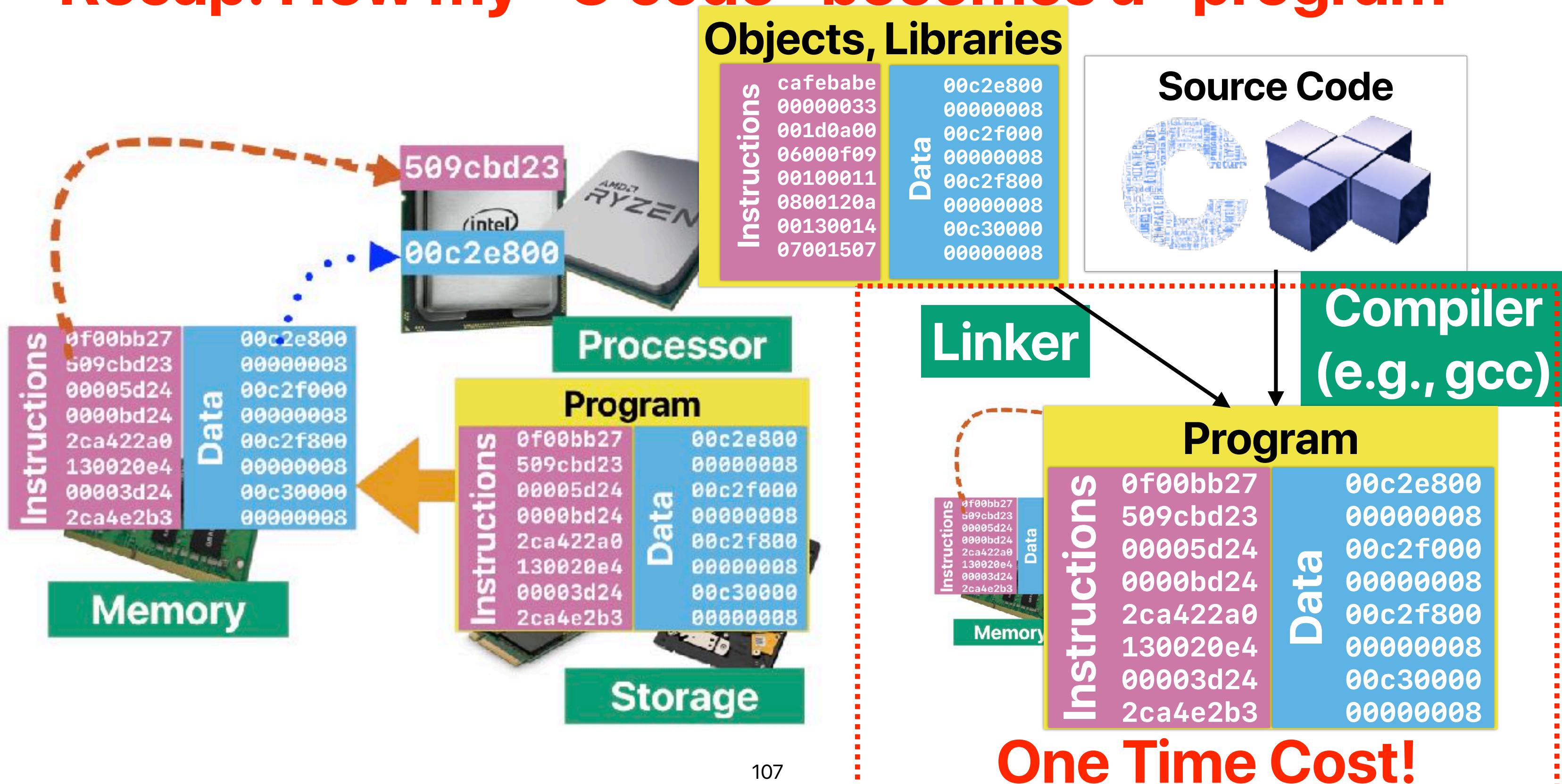
- How many instructions are there in “Hello, world!”

	Instruction count	LOC	Ranking
C	600k	6	1
C++	3M	6	2
Java	~145M	8	5
Perl	~12M	4	3
Python	~33M	1	4
GO (Interpreter)	~1200M	1	6
GO (Compiled)	~1.7M	1	
Rust	~1.4M	1	

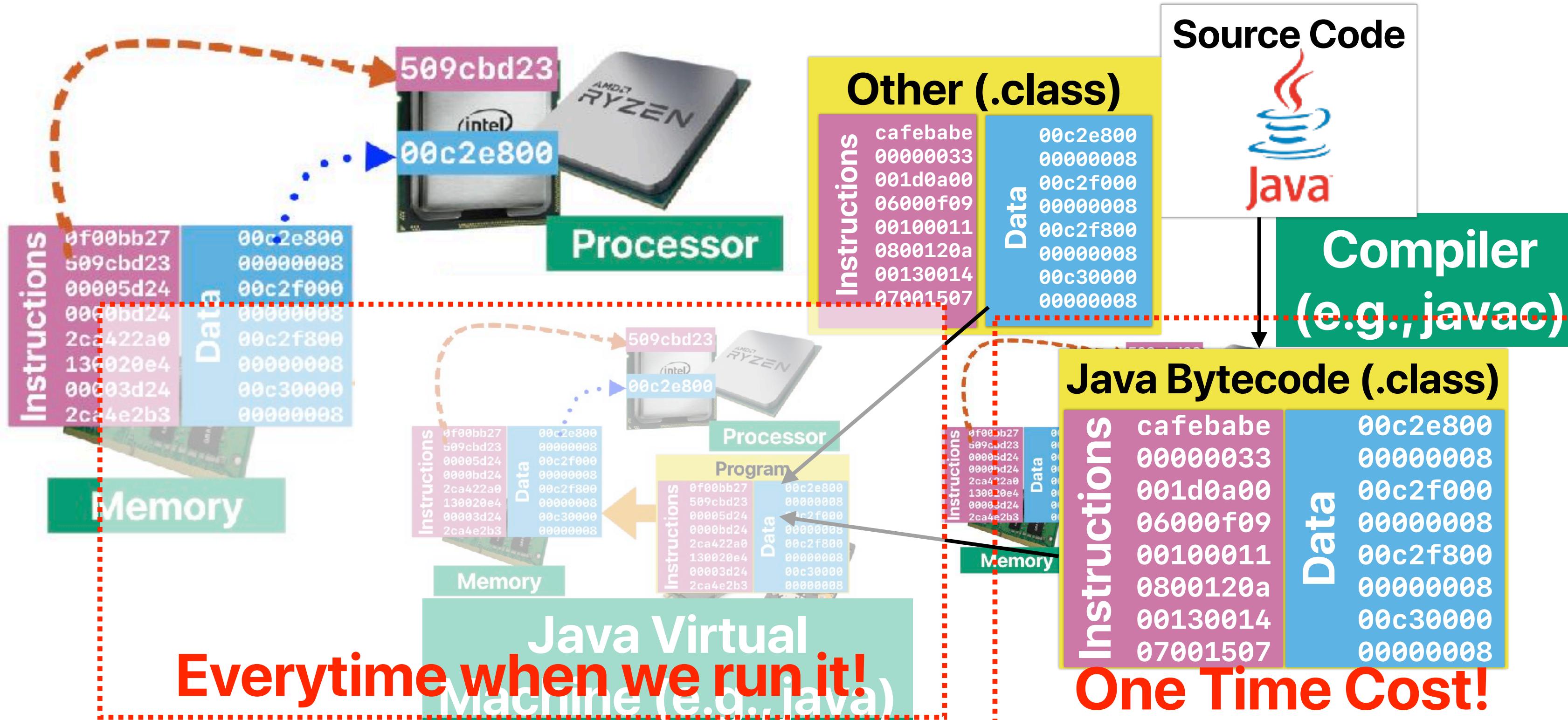
Programming languages

- Which of the following programming language needs to highest instruction count to print “Hello, world!” on screen?
 - A. C
 - B. C++
 - C. Java
 - D. Perl
 - E. Python

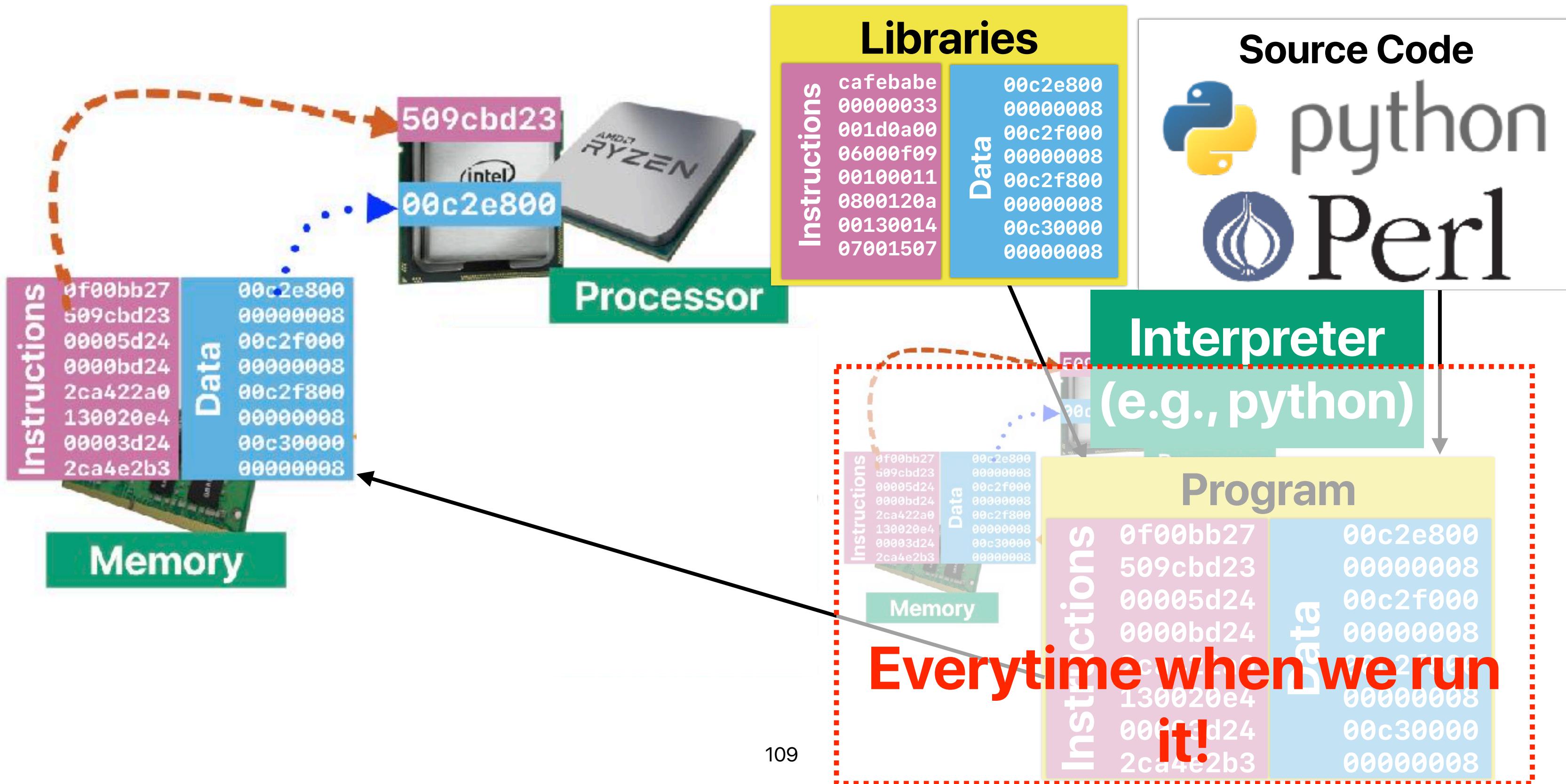
Recap: How my “C code” becomes a “program”



Recap: How my “Java code” becomes a “program”



Recap: How my “Python code” becomes a “program”



Takeaways: What matters?

- Programmers can control all three factors in the classic performance equation
- Different programming languages can generate machine operations with different orders of magnitude performance — programmers need to make wise choice of that!

Takeaways: What matters?

- Programmers can control all three factors in the classic performance equation
- Different programming languages can generate machine operations with different orders of magnitude performance — programmers need to make wise choice of that!
- Compiler optimization can help — only if programmers write code in a way facilitating optimizations!

How about complexity?

How about “computational complexity”

- Algorithm complexity provides a good estimate on the performance if —
 - Every instruction takes exactly the same amount of time
 - Every operation takes exactly the same amount of instructions

These are unlikely to be true

Summary of CPU Performance Equation

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$ET = IC \times CPI \times CT$$

- IC (Instruction Count)
 - ISA, Compiler, algorithm, programming language, **programmer**
- CPI (Cycles Per Instruction)
 - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, **programmer**
- Cycle Time (Seconds Per Cycle)
 - Process Technology, microarchitecture, **programmer**

Takeaways: What matters?

- Programmers can control all three factors in the classic performance equation
- Different programming languages can generate machine operations with different orders of magnitude performance — programmers need to make wise choice of that!
- Compiler optimization can help — only if programmers write code in a way facilitating optimizations!
- Complexity does not provide good assessment on real machines due to the idealized assumptions

Quantitative Analysis of “Better”

Speedup

- The relative performance between two machines, X and Y. Y is n times faster than X

$$n = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}$$

- The speedup of Y over X

$$\text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}$$

Takeaways: What matters?

- Programmers can control all three factors in the classic performance equation
- Different programming languages can generate machine operations with different orders of magnitude performance — programmers need to make wise choice of that!
- Compiler optimization can help — but programmers need to write code in a way facilitate optimizations!
- Complexity does not provide good assessment on real machines due to the idealized assumptions
- The only definition of Y is Speedup times faster than X —

$$\text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}$$

Amdahl's Law — and It's Implication in the Multicore Era

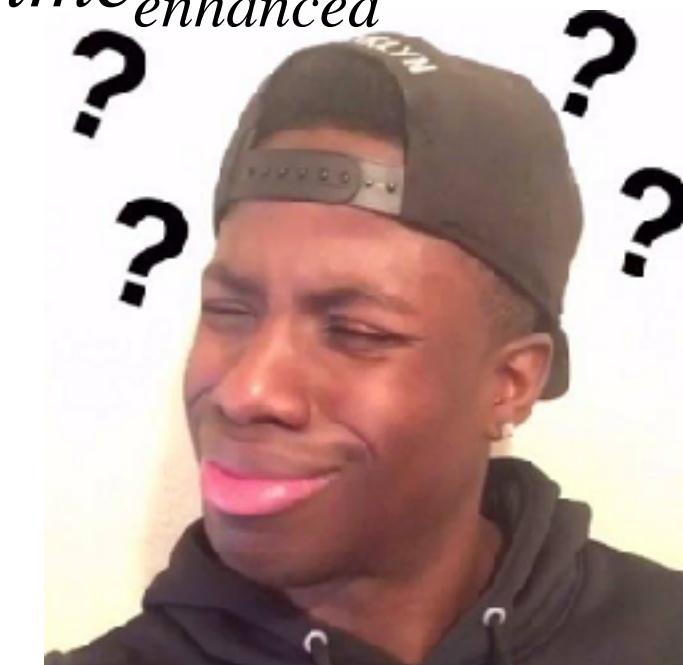
Amdahl's Law



$$\text{Speedup}_{\text{enhanced}}(f, s) = \frac{1}{(1-f) + \frac{f}{s}}$$

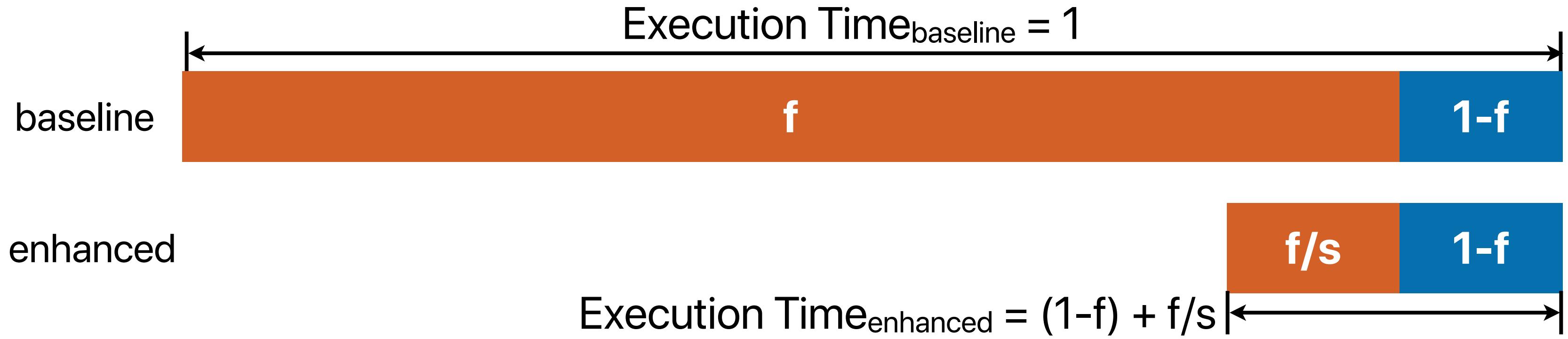
- f — The fraction of time in the original program
 s — The speedup we can achieve on f

$$\text{Speedup}_{\text{enhanced}} = \frac{\text{Execution Time}_{\text{baseline}}}{\text{Execution Time}_{\text{enhanced}}}$$



Amdahl's Law

$$Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}$$



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

Amdahl's Law on Multiple Optimizations

- We can apply Amdahl's law for multiple optimizations
- These optimizations must be dis-joint!
 - If optimization #1 and optimization #2 are dis-joint:



$$Speedup_{enhanced}(f_{Opt1}, f_{Opt2}, s_{Opt1}, s_{Opt2}) = \frac{1}{(1 - f_{Opt1} - f_{Opt2}) + \frac{f_{Opt1}}{s_{Opt1}} + \frac{f_{Opt2}}{s_{Opt2}}}$$

- If optimization #1 and optimization #2 are not dis-joint:



$$Speedup_{enhanced}(f_{OnlyOpt1}, f_{OnlyOpt2}, f_{BothOpt1Opt2}, s_{OnlyOpt1}, s_{OnlyOpt2}, s_{BothOpt1Opt2}) = \frac{1}{(1 - f_{OnlyOpt1} - f_{OnlyOpt2} - f_{BothOpt1Opt2}) + \frac{f_{BothOpt1Opt2}}{s_{BothOpt1Opt2}} + \frac{f_{OnlyOpt1}}{s_{OnlyOpt1}} + \frac{f_{OnlyOpt2}}{s_{OnlyOpt2}}}$$

Amdahl's Law Corollary #1

- The maximum speedup is bounded by

$$\text{Speedup}_{max}(f, \infty) = \frac{1}{(1-f) + \frac{f}{\infty}}$$

$$\text{Speedup}_{max}(f, \infty) = \frac{1}{(1-f)}$$

Intel kills the remnants of Optane memory

The speed-boosting storage tech was already on the ropes.



By [Michael Crider](#)

Staff Writer, PCWorld | JUL 29, 2022 6:59 AM PDT



Image: Intel

MILLIPORE
SIGMA



MISSION® es

targeting mo.
2010204k13r

Corollary #1 on Multiple Optimizations

- If we can pick just one thing to work on/optimize



$$Speedup_{max}(f_1, \infty) = \frac{1}{(1 - f_1)}$$

$$Speedup_{max}(f_2, \infty) = \frac{1}{(1 - f_2)}$$

$$Speedup_{max}(f_3, \infty) = \frac{1}{(1 - f_3)}$$

$$Speedup_{max}(f_4, \infty) = \frac{1}{(1 - f_4)}$$

The biggest f_x would lead to the largest $Speedup_{max}$!

Corollary #2 — make the common case fast!

- When f is small, optimizations will have little effect.
- Common == **most time consuming** not necessarily the most frequent
- The uncommon case doesn't make much difference
- The common case can change based on inputs, compiler options, optimizations you've applied, etc.

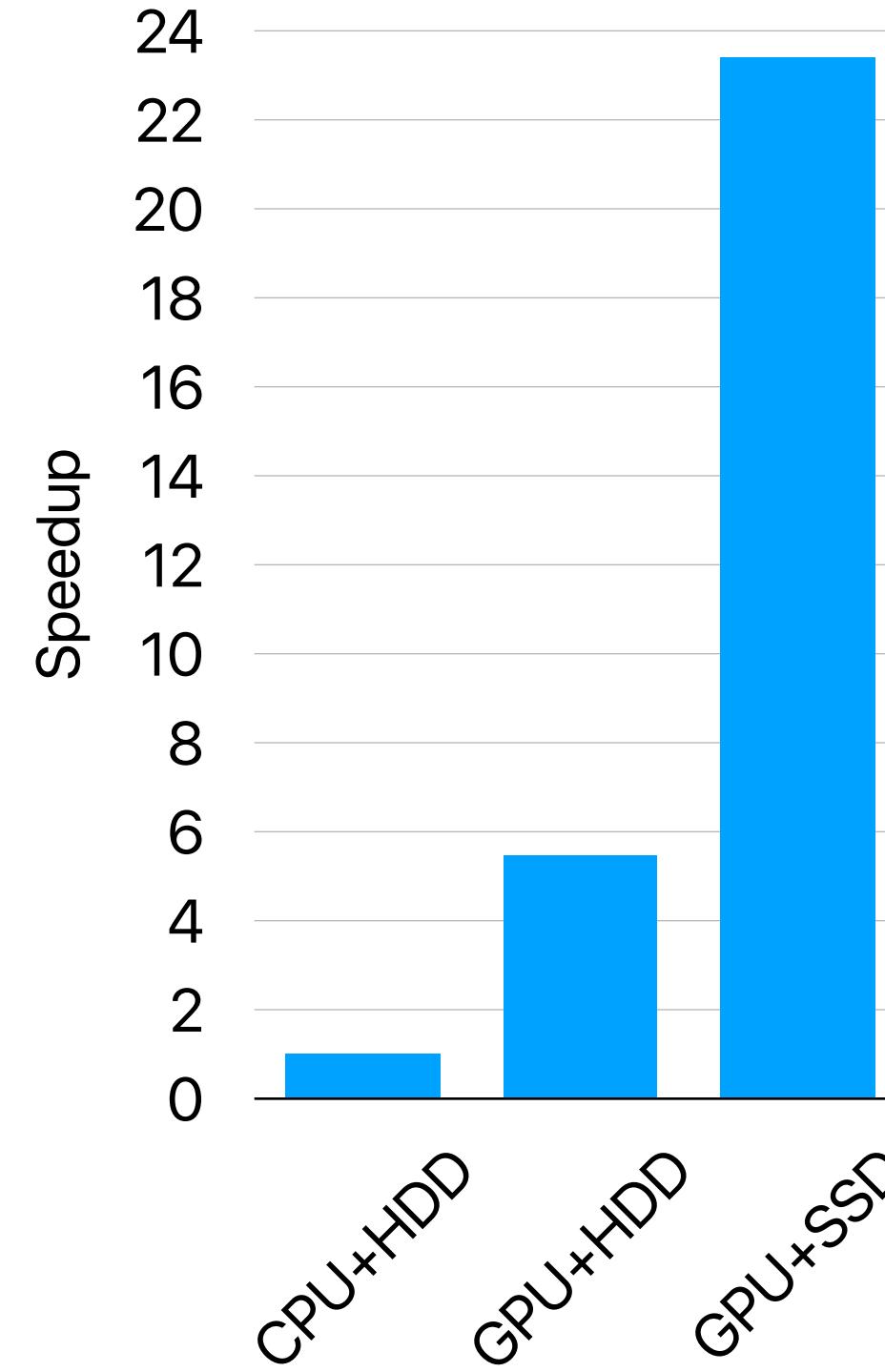
Identify the most time consuming part

- Compile your program with -pg flag
- Run the program
 - It will generate a gmon.out
 - `gprof your_program gmon.out > your_program.prof`
- It will give you the profiled result in `your_program.prof`

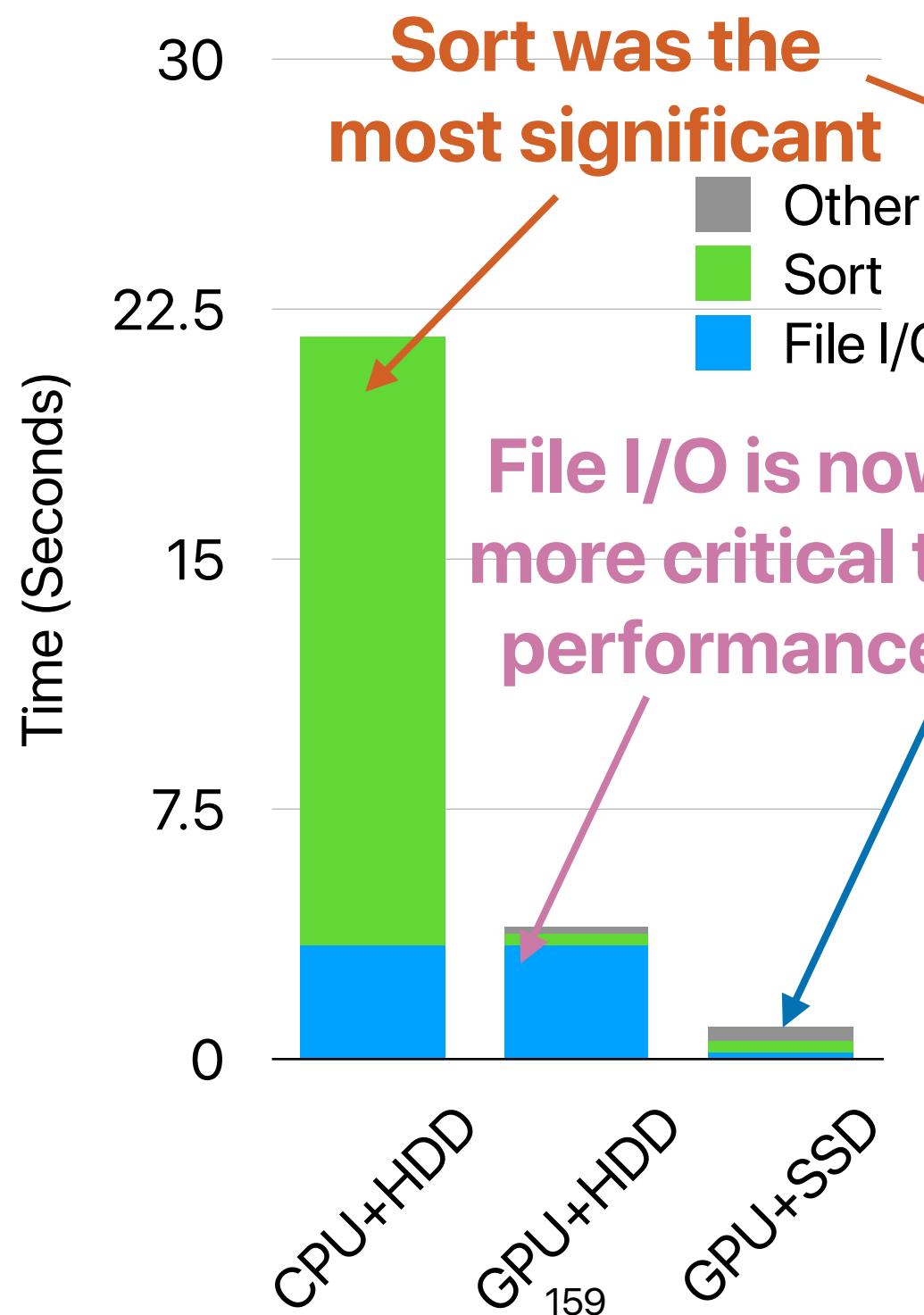
Demo — sort

Something else (e.g., data movement) matters more

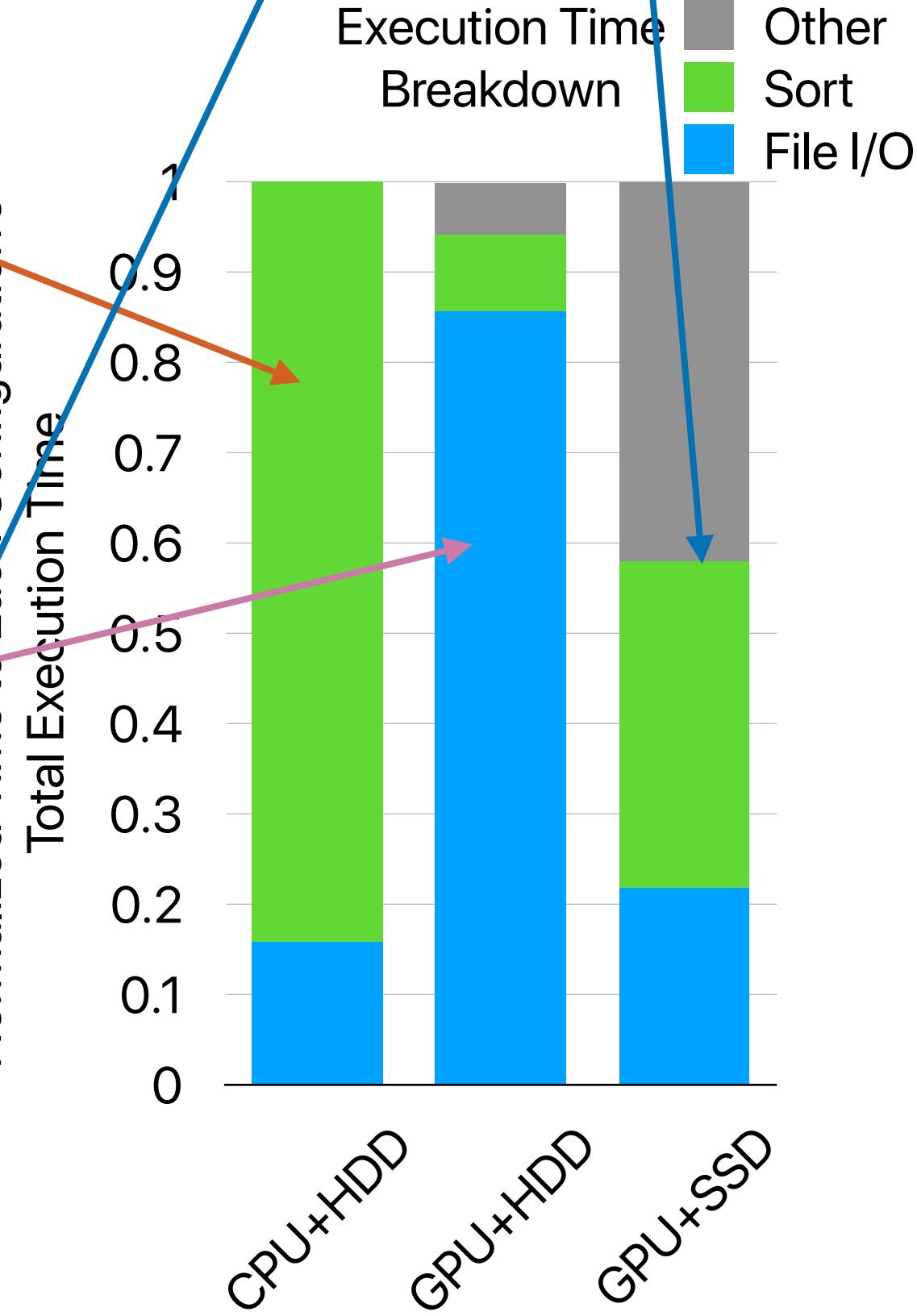
Speedup



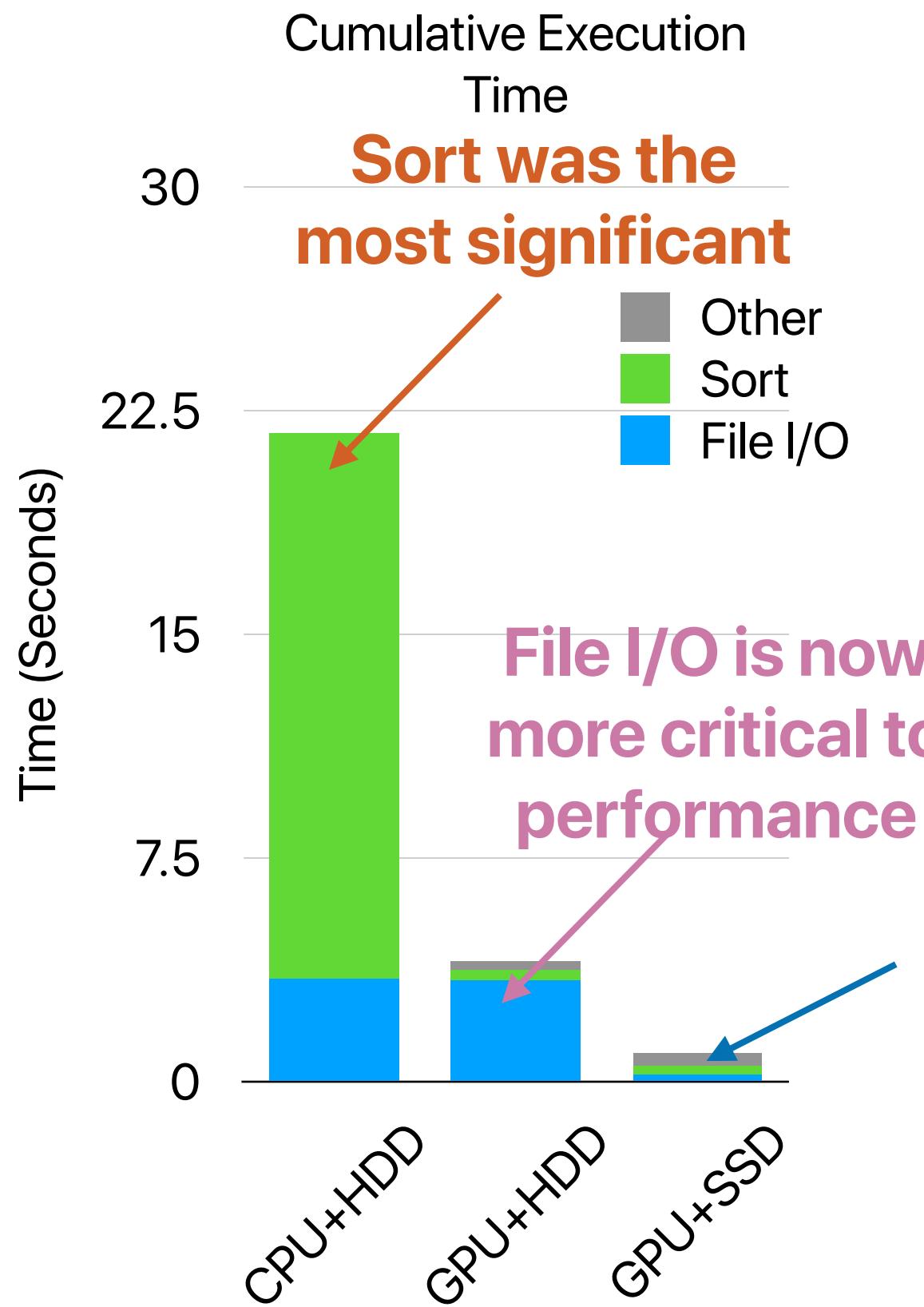
Cumulative Execution Time



Normalized Time to Each Configuration's Total Execution Time



If we repeatedly optimizing our design based on Amdahl's law...



- With optimization, the common becomes uncommon.
- An uncommon case will (hopefully) become the new common case.
- Now you have a new target for optimization — You have to revisit “Amdahl’s Law” every time you applied some optimization

Something else (e.g.,
data movement)
matters more now

Amdahl's Law on Multicore Architectures

- Symmetric multicore processor with n cores (if we assume the processor performance scales perfectly)

$$\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, n) = \frac{1}{(1 - f_{\text{parallelizable}}) + \frac{f_{\text{parallelizable}}}{n}}$$

Demo — merge sort v.s. bitonic sort on GPUs

Merge Sort

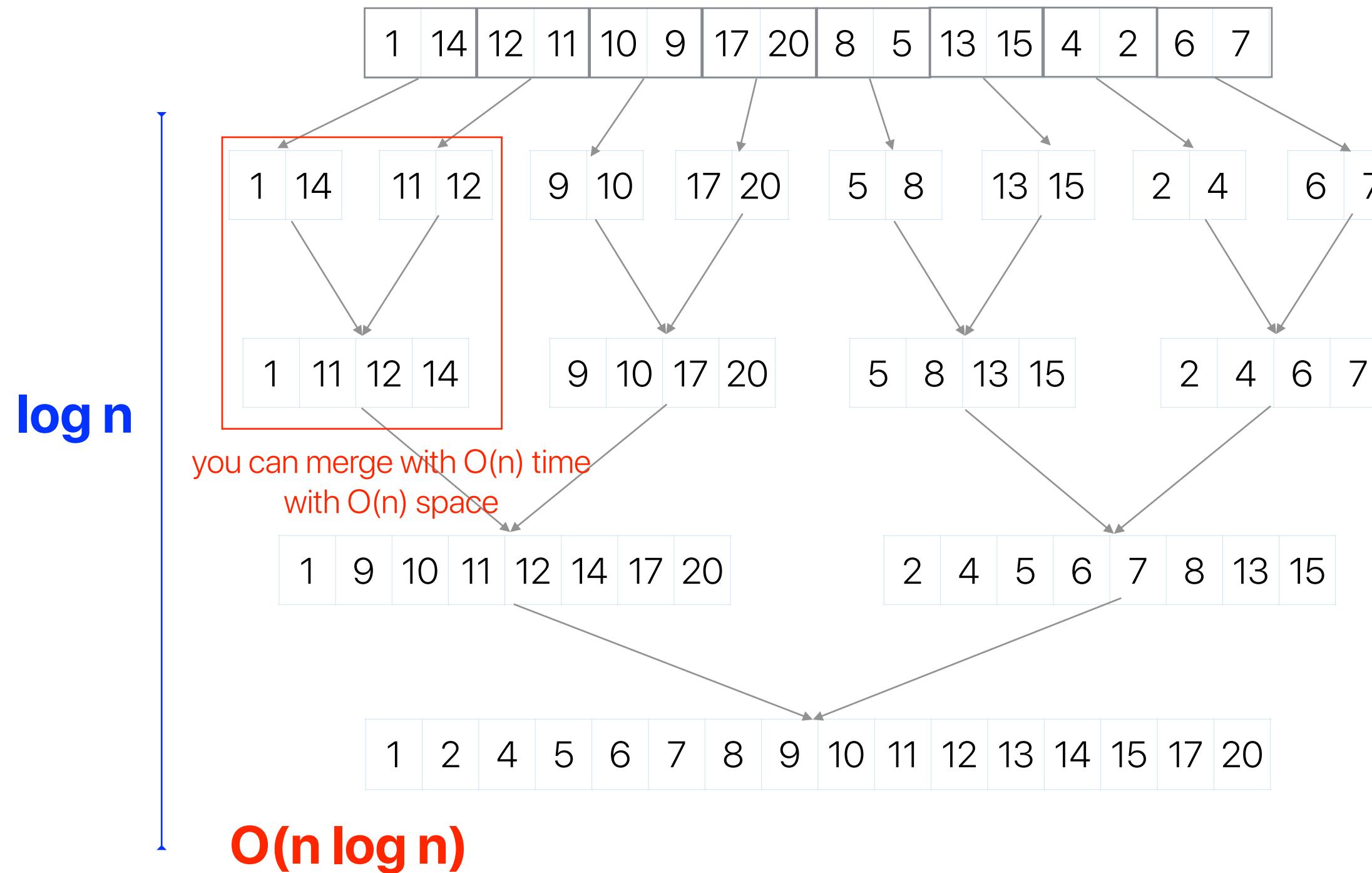
$$O(n \log_2 n)$$

Bitonic Sort

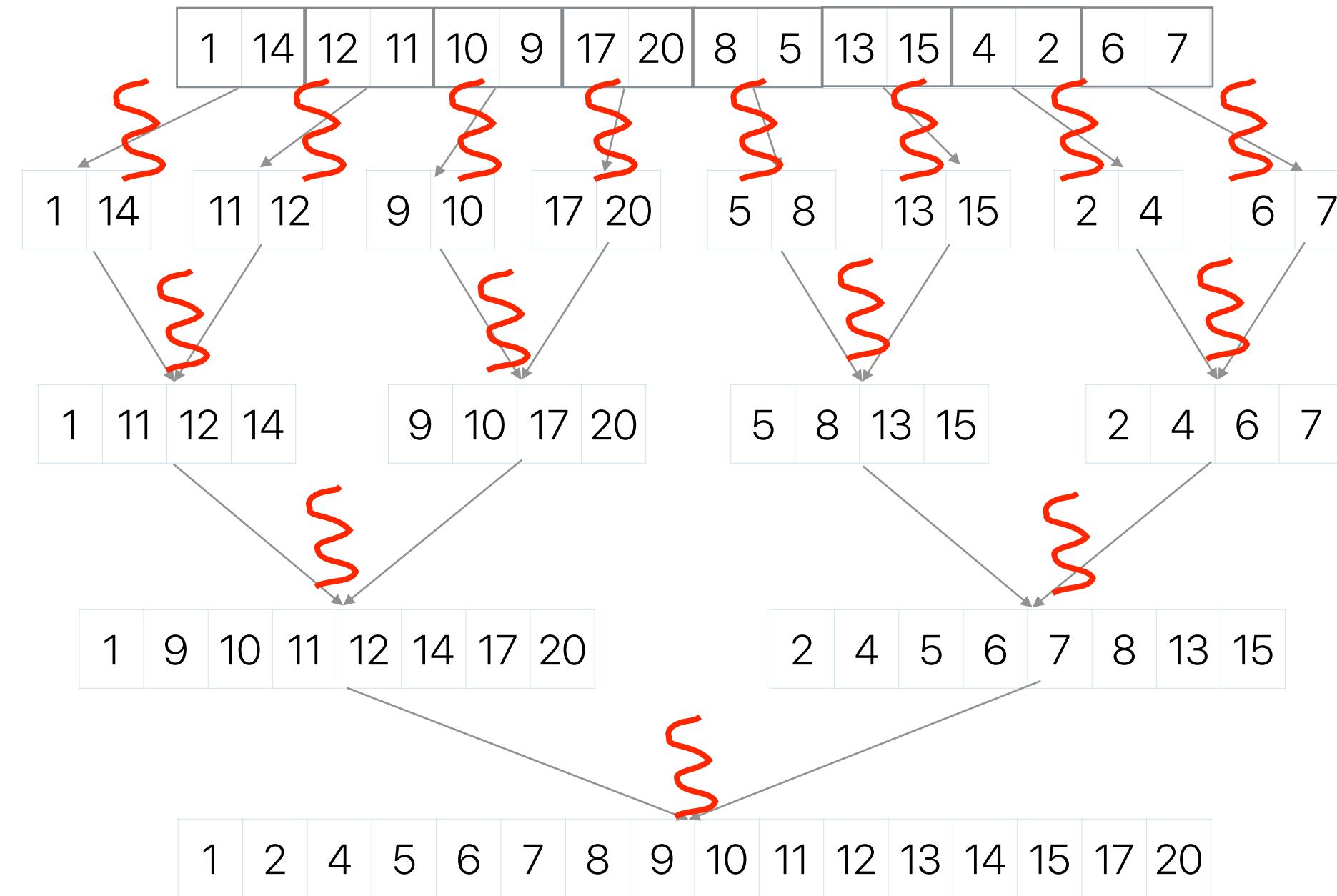
$$O(n \log_2^2 n)$$

```
void BitonicSort() {  
    int i,j,k;  
  
    for (k=2; k<=N; k=2*k) {  
        for (j=k>>1; j>0; j=j>>1) {  
            for (i=0; i<N; i++) {  
                int ij=i^j;  
                if ((ij)>i) {  
                    if ((i&k)==0 && a[i] > a[ij])  
                        exchange(i,ij);  
                    if ((i&k)!=0 && a[i] < a[ij])  
                        exchange(i,ij);  
                }  
            }  
        }  
    }  
}
```

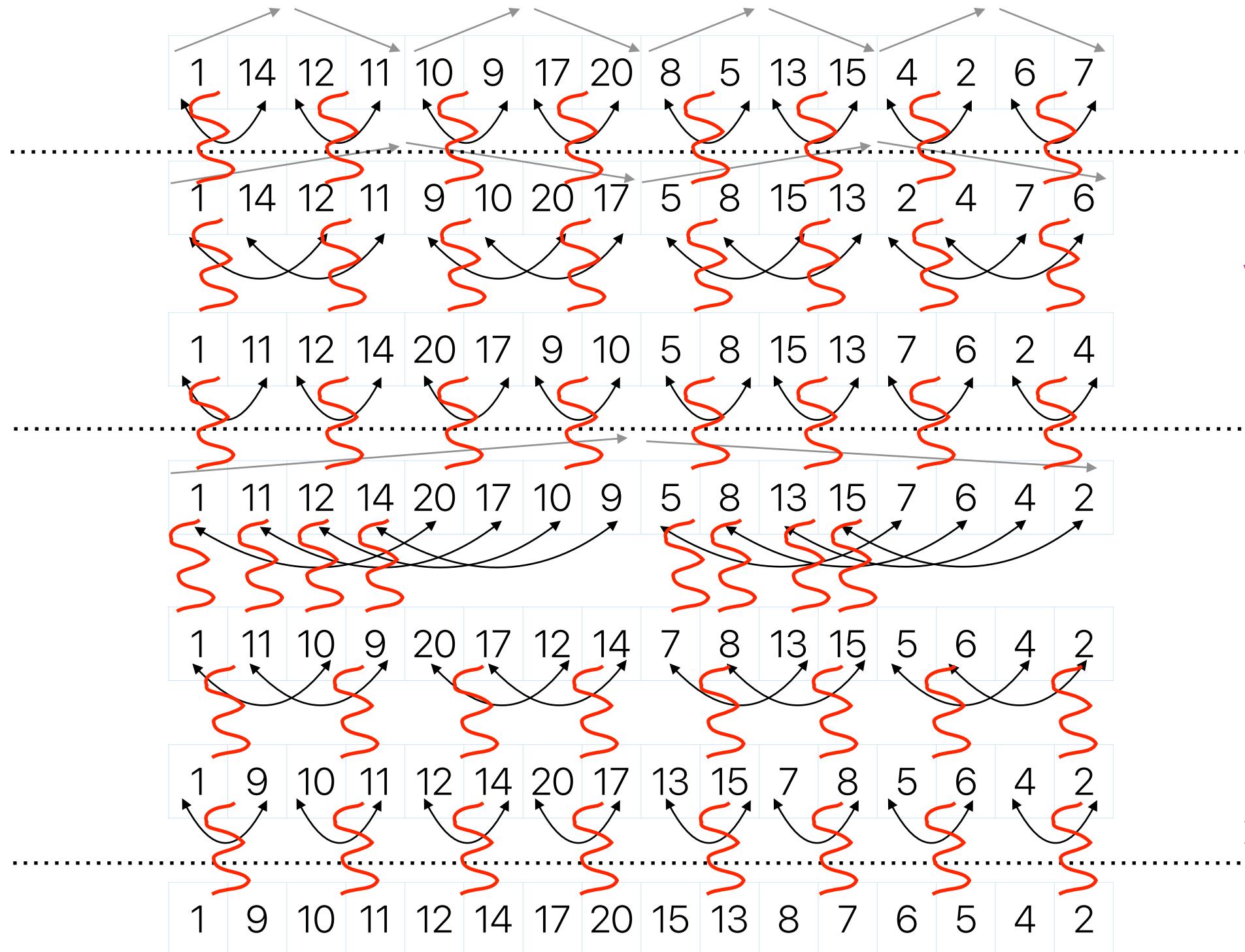
Merge sort



Parallel merge sort

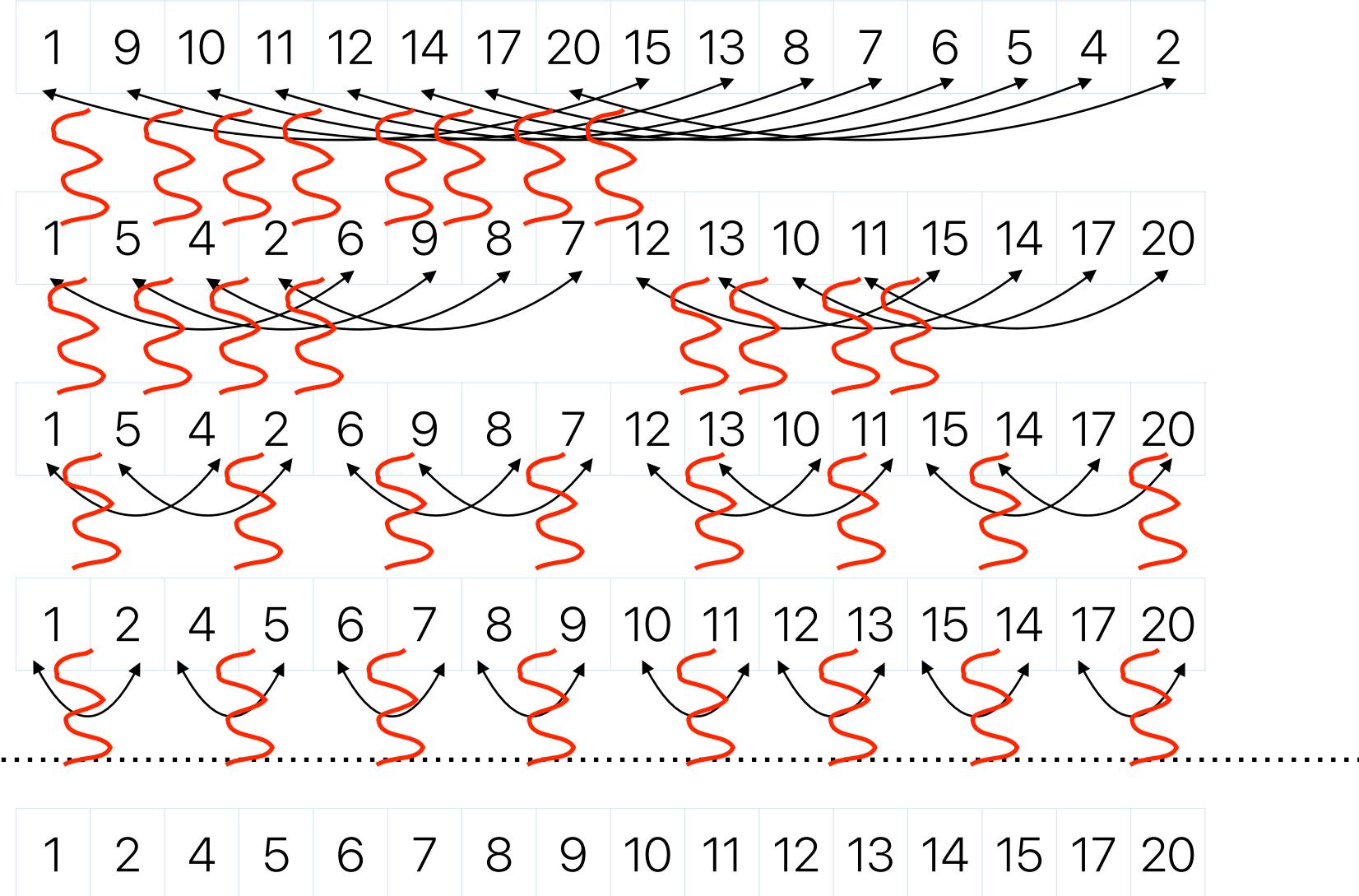


Bitonic sort



```
void BitonicSort() {  
    int i,j,k;  
  
    for (k=2; k<=N; k=2*k) {  
        for (j=k>>1; j>0; j=j>>1) {  
            for (i=0; i<N; i++) {  
                int ij=i^j;  
                if ((ij)>i) {  
                    if ((i&k)==0 && a[i] > a[ij])  
                        exchange(i,ij);  
                    if ((i&k)!=0 && a[i] < a[ij])  
                        exchange(i,ij);  
                }  
            }  
        }  
    }  
}
```

Bitonic sort (cont.)



```
void BitonicSort() {  
  
    int i, j, k;  
  
    for (k=2; k<=N; k=2*k) {  
        for (j=k>>1; j>0; j=j>>1) {  
            for (i=0; i<N; i++) {  
                int ij=i^j;  
                if ((ij)>i) {  
                    if ((i&k)==0 && a[i] > a[ij])  
                        exchange(i,ij);  
                    if ((i&k)!=0 && a[i] < a[ij])  
                        exchange(i,ij);  
                }  
            }  
        }  
    }  
}
```

benefits — in-place merge (no additional space is necessary), very stable comparison patterns

$O(n \log^2 n)$ — hard to beat $n(\log n)$ if you can't parallelize this a lot!

Corollary #4

$$\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}}) + \frac{f_{\text{parallelizable}}}{\infty}}$$

$$\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}})}$$

- If we can build a processor with unlimited parallelism
 - The complexity doesn't matter as long as the algorithm can utilize all parallelism
 - That's why bitonic sort or MapReduce works!
- **The future trend of software/application design is seeking for more parallelism rather than lower the computational complexity**

**Is it the end of computational
complexity?**

Corollary #5

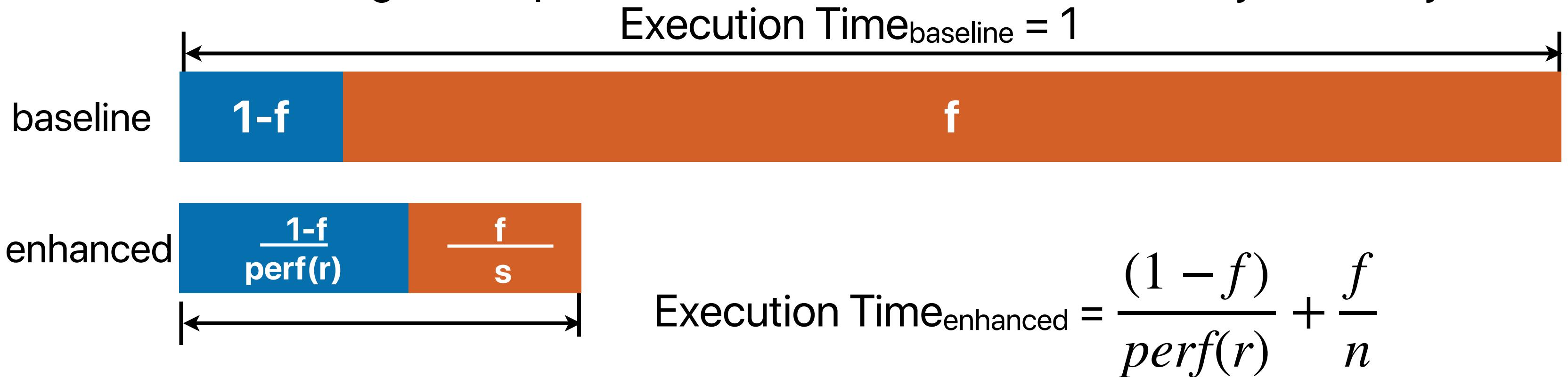
$$\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}}) + \frac{f_{\text{parallelizable}}}{\infty}}$$

$$\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}})}$$

- Single-core performance still matters
 - It will eventually dominate the performance
 - If we cannot improve single-core performance further, finding more “parallelizable” parts is more important
 - Algorithm complexity still gives some “insights” regarding the growth of execution time in the same algorithm, though still not accurate

However, parallelism is not “tax-free”

- Synchronization
- Preparing data
- Addition function calls
- Data exchange if the parallel hardware has its own memory hierarchy



Don't hurt non-common part too much

- If the program spend 90% in A, 10% in B. Assume that an optimization can accelerate A by 9x, by hurts B by 10x...
- Assume the original execution time is T. The new execution time

$$ET_{new} = \frac{ET_{old} \times 90\%}{9} + ET_{old} \times 10\% \times 10$$

$$ET_{new} = 1.1 \times ET_{old}$$

$$Speedup = \frac{ET_{old}}{ET_{new}} = \frac{ET_{old}}{1.1 \times ET_{old}} = 0.91 \times \dots \text{slowdown!}$$

You may not use Amdahl's Law for this case as Amdahl's Law does NOT
(1) consider overhead
(2) bound to slowdown

Takeaways: Are we there yet?

- Definition of “Speedup of Y over X” or say Y is n times faster than X:

$$speedup_{Y_over_X} = n = \frac{Execution\ Time_X}{Execution\ Time_Y}$$

- Amdahl's Law — $Speedup_{enhanced}(f, s) = \frac{1}{(1-f) + \frac{f}{s}}$

- Corollary 1 — each optimization has an upper bound

$$Speedup_{max}(f, \infty) = \frac{1}{(1-f)}$$

- Corollary 2 — make the common case (the most time consuming case) fast!

$$Speedup_{max}(f_1, \infty) = \frac{1}{(1-f_1)}$$

$$Speedup_{max}(f_2, \infty) = \frac{1}{(1-f_2)}$$

$$Speedup_{max}(f_3, \infty) = \frac{1}{(1-f_3)}$$

$$Speedup_{max}(f_4, \infty) = \frac{1}{(1-f_4)}$$

- Corollary 3: Optimization has a moving target

- Corollary 4: Exploiting more parallelism from a program is the key to performance gain in modern architectures

$$Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1-f_{parallelizable})}$$

- Corollary 5: Single-core performance still matters

$$Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1-f_{parallelizable})}$$

- Corollary 6: Don't hurt the non-common case too much

$$Speedup_{enhanced}(f, s, r) = \frac{1}{(1-f) + perf(r) + \frac{f}{s}}$$