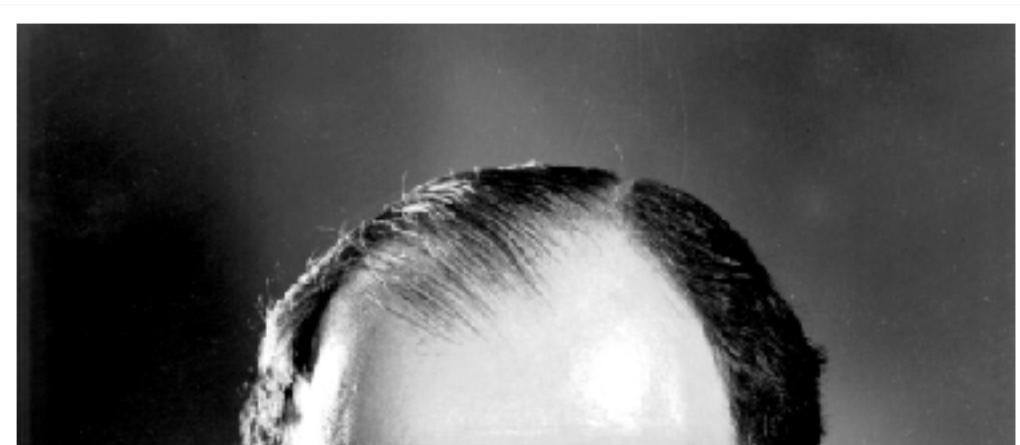


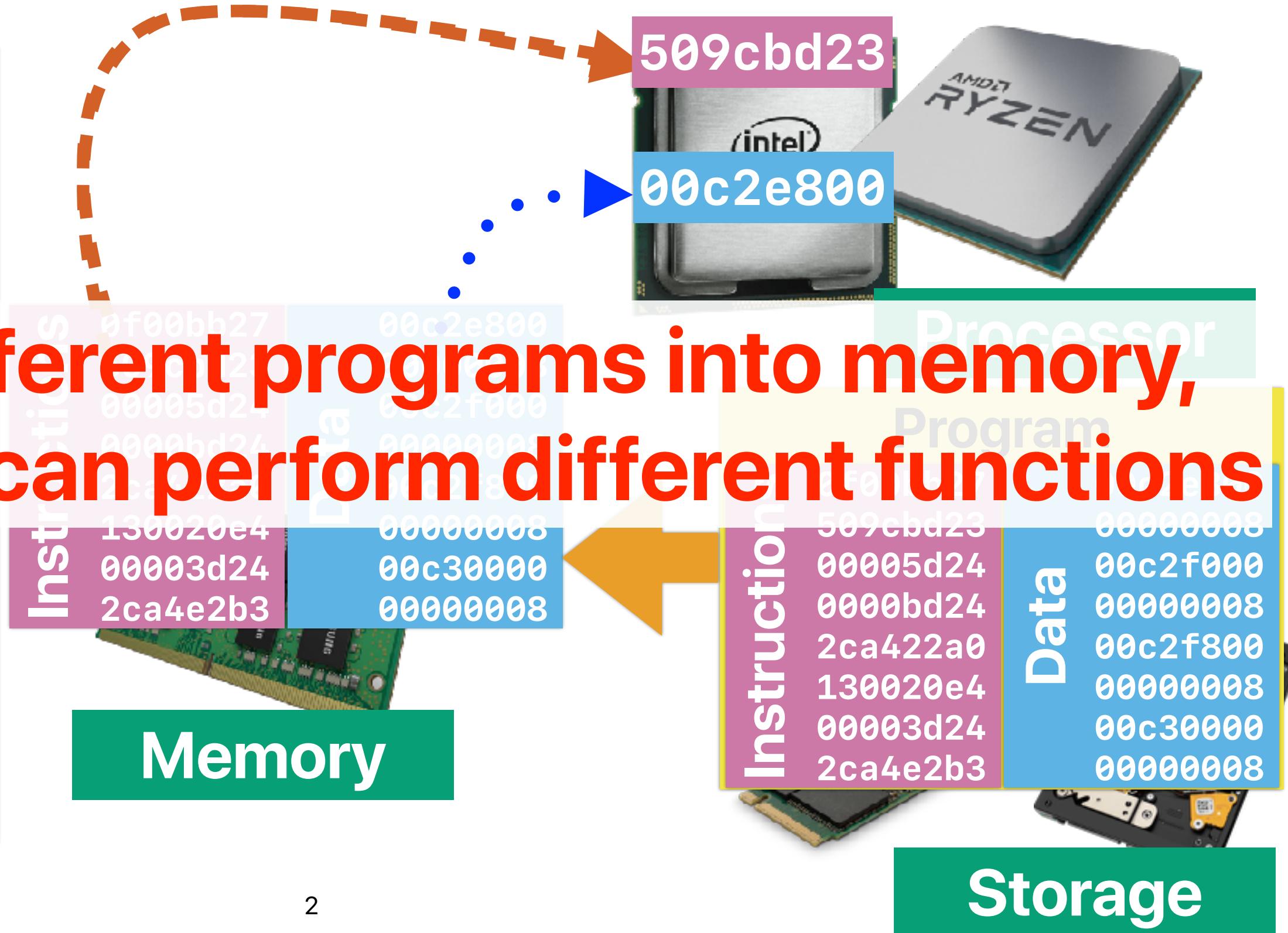
# **Performance (1): How Good Is “Good”?**

Hung-Wei Tseng

# Recap: von Neumann Architecture



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



# Recap: Demo

```
if(option)
    std::sort(data, data + arraySize);  $O(n \log_2 n)$ 
```

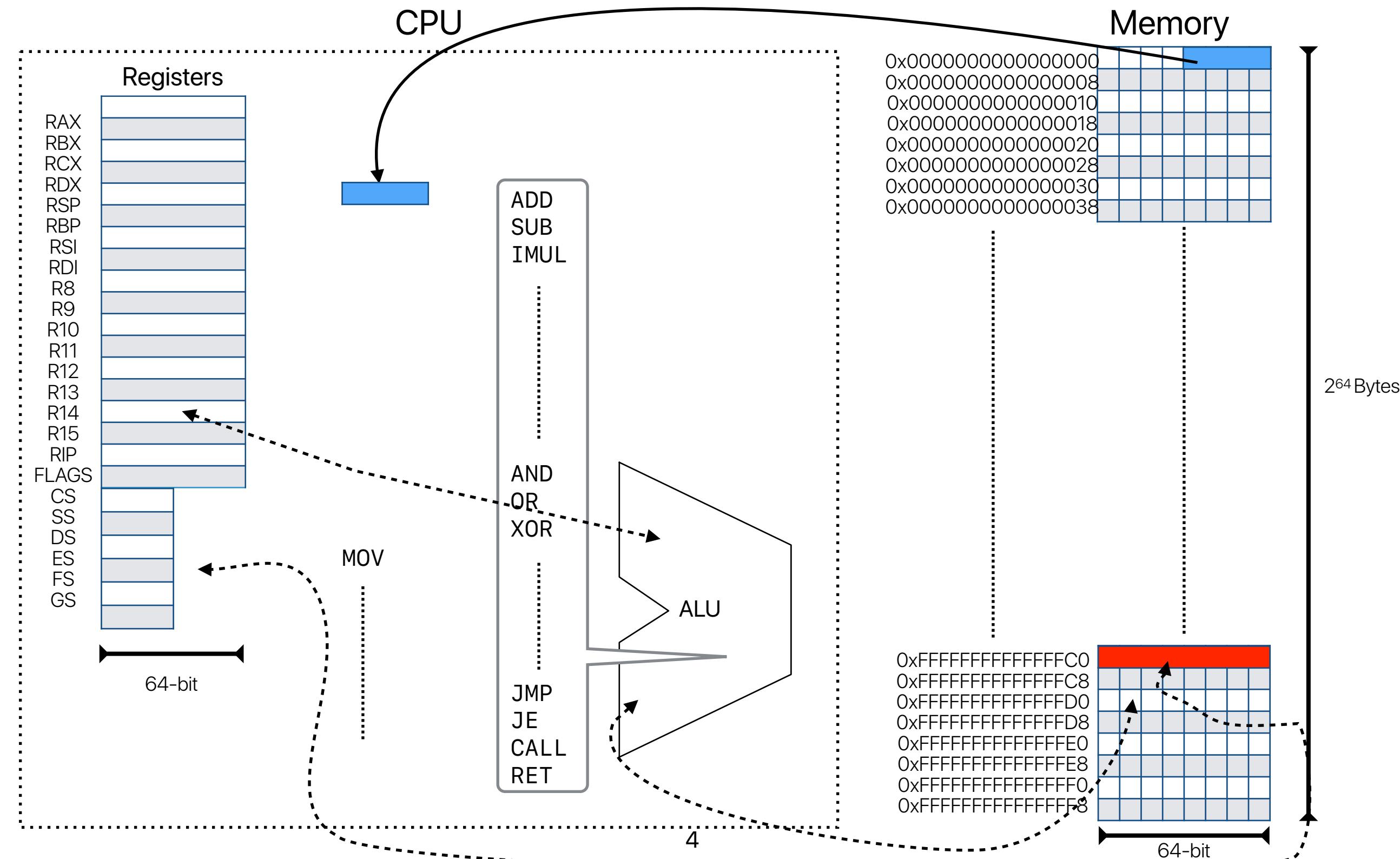
```
for (unsigned c = 0; c < arraySize*1000; ++c) {
    if (data[c%arraySize] >= INT_MAX/2)
        sum++;
}
```

$O(n)$

**if option is set to 1:**  $O(n \log_2 n)$

**otherwise, O(n):**  $O(n)$

# The abstracted x86 machine



# Outline

- Definition of “Performance”
- The performance equation
- What affects each factor in “Performance Equation”

## Best National

Schools in the National University are a full range of undergraduate research producing groundbreaking results.

To unlock full rankings, SAT/ACT scores

SUMMARY ▾



443

Scho

Sc

Loca

Cit

All

Rank

Nat

## Best Computer Science Schools

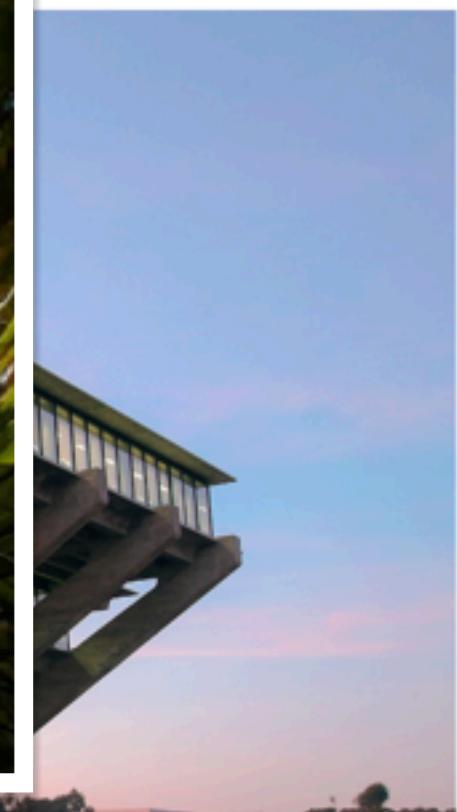
Ranked in 2022, part of Best Science Schools

Earning a graduate degree in computer technology companies and colleges are reflects its average rating on a scale from institutions. [Read the methodology](#) ▾



# UC San Diego Ranked No. 1 Public University by Washington Monthly

Campus celebrated as a leader in social mobility, research and public



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

# Bard



Who is Hung-Wei Tseng?

View other drafts

Hung-Wei Tseng is an associate professor in the Department of Electrical and Computer Engineering and a cooperating faculty of the Department of Computer Science and Engineering at University of California, Riverside. He is now leading the Extreme Storage & Computer Architecture Laboratory. He is interested in diverse research topics that allow applications or programmers to more efficiently use modern heterogeneous hardware components. Together with his students, they have demonstrated how intelligent storage devices can help improve performance, power and energy for data-intensive systems.

Their work has been recognized by IEEE Micro "Top Picks from the 2019 Computer Architecture Conferences" (IEEE MICRO Top Picks 2020) and Facebook Research Award, 2018. Prior to joining UCR, he served as an assistant professor for the Department of Computer Science and the Department of Electrical and Computer Engineering at NC State University. He was a PostDoc of the Non-volatile Systems Laboratory and a lecturer of the Department of Computer Science and Engineering at University of California, San Diego with Professor Steven Swanson.

Sources - Learn more  
1. [ira.ece.ucr.edu/~htseng/](http://ira.ece.ucr.edu/~htseng/)  
2. [www.nvsl.io/portfolio/hung-wei-tseng/](http://www.nvsl.io/portfolio/hung-wei-tseng/)

Like Dislike Google it

How important is Computer Architecture in Computer Science? ▶

182 words/6 secs

I am unable to determine who Hung-Wei Tseng is. Can you please provide me with more context or details about the individual in question?

HT Who is Hung-Wei Tseng at UCR?

Hung-Wei Tseng is an Associate Professor of Computer Science and Engineering at the University of California, Riverside (UCR). His research interests are in the area of computer networks and distributed systems, with a focus on improving network performance and security. Some of his recent research work has focused on topics such as network measurement and monitoring, network optimization, and network security. Hung-Wei Tseng has been a faculty member at UCR since 2013, and he has received several awards and recognitions for his research contributions.

Regenerate response

How important is Computer Architecture in Computer Science? ▶

170 words/18 secs

# Performance

- The right metric — latency? throughput? quality of results?
- The quantitative comparison — A is better than B by “how much”

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

# CPU Performance Equation

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

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

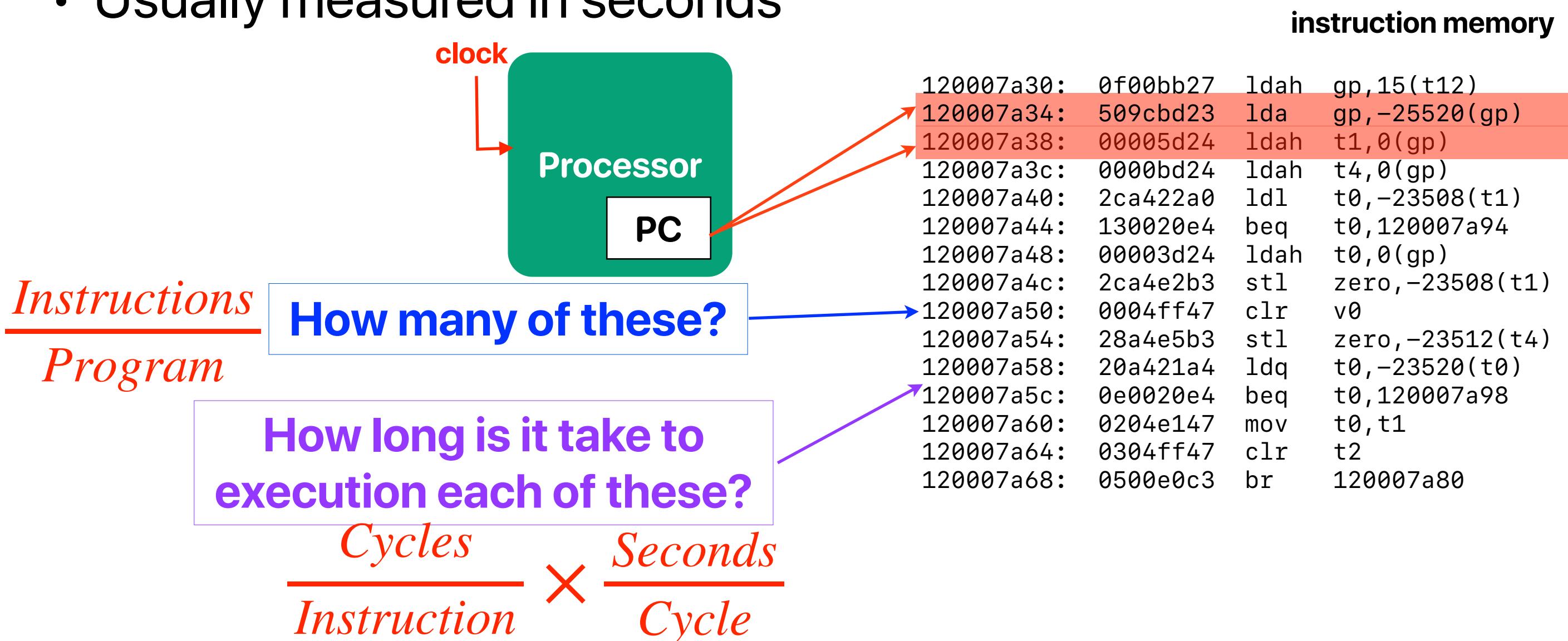
$$ET = IC \times CPI \times CT$$

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

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

# Execution Time

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



# 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

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

# **What Affects Each Factor in Performance Equation**

# 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

# 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
$
```

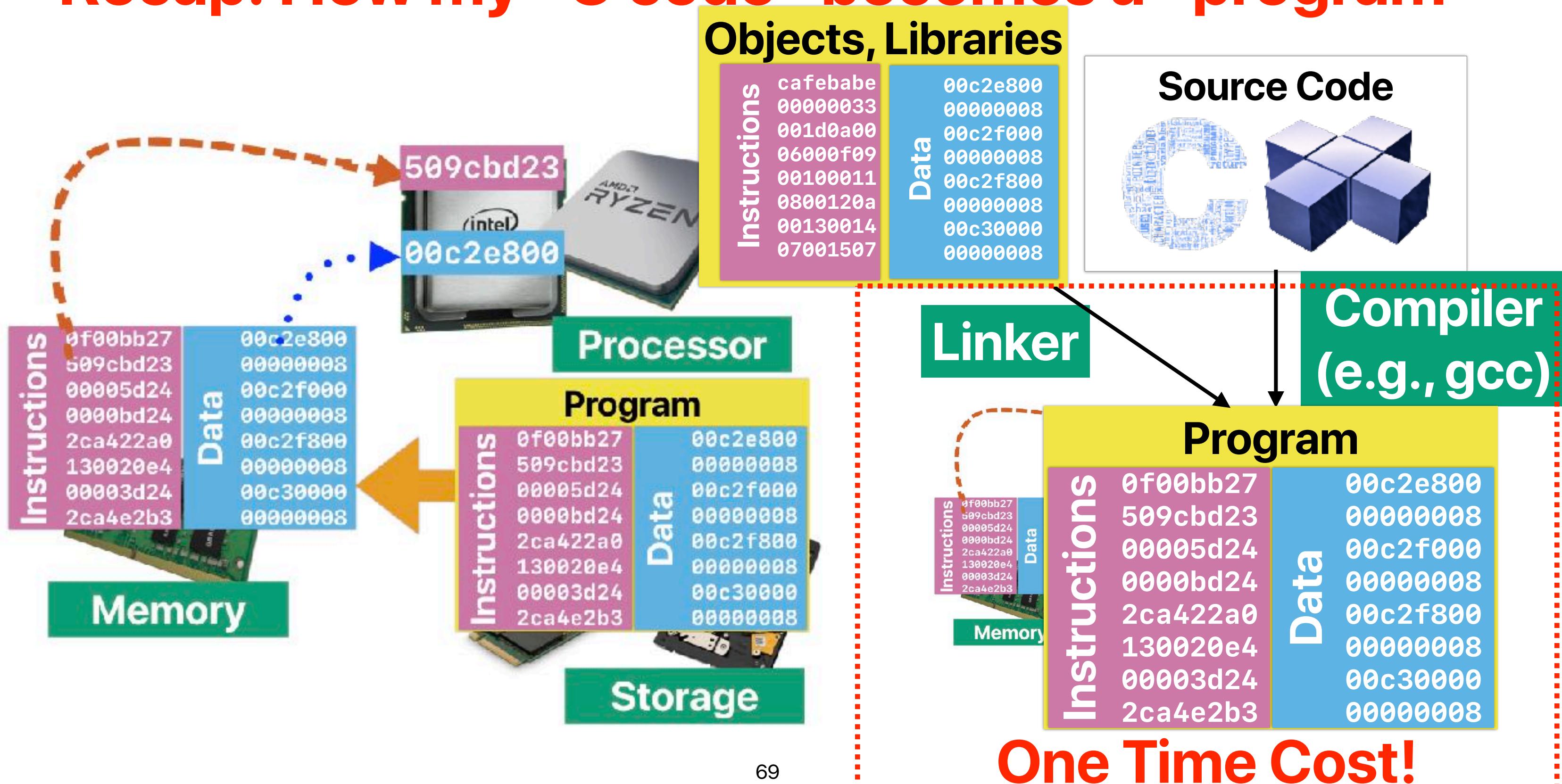
5.) How do I return to the default - i.e. allow machine to vary from highest to lowest?

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

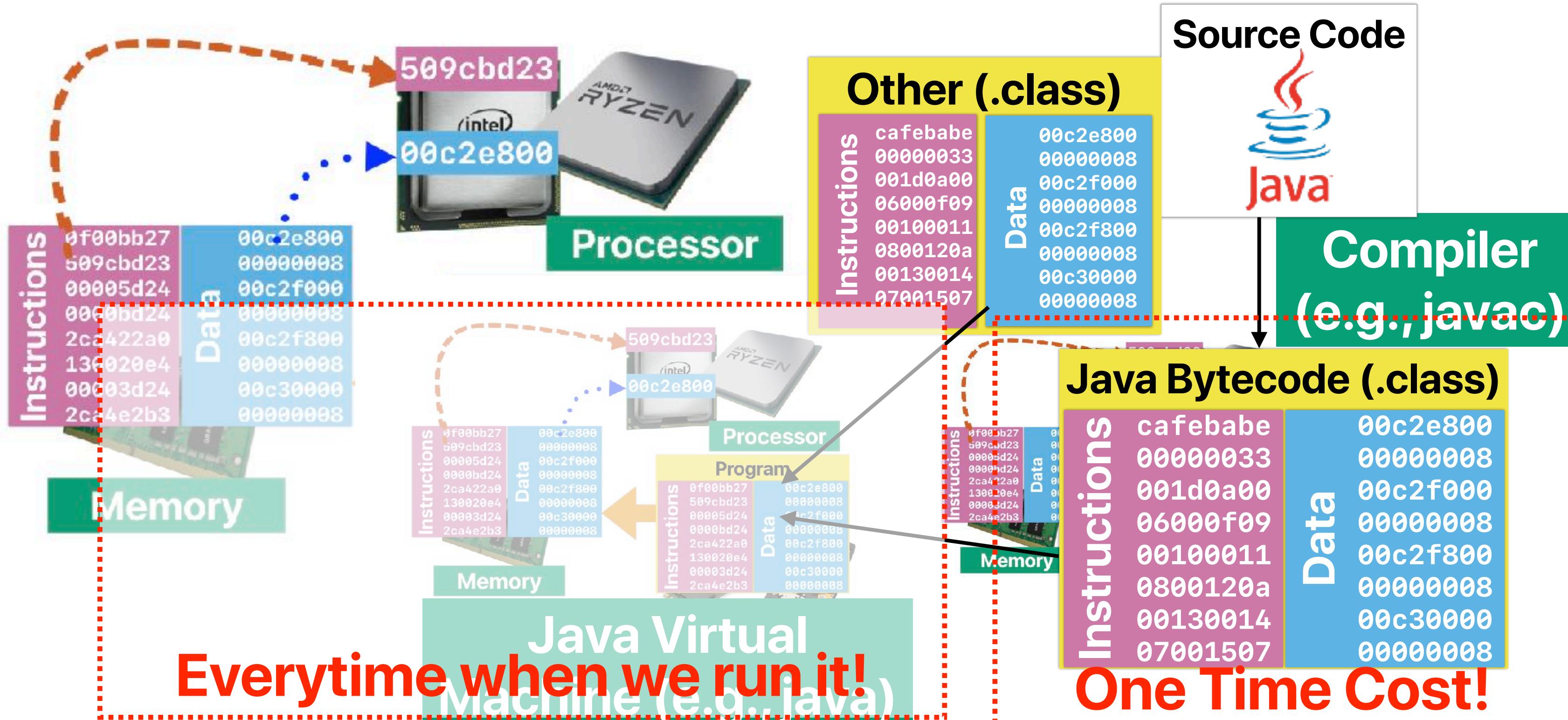
```
$ newSpeedLcw=$newSpeedTop # make them the same in this example
```

To read

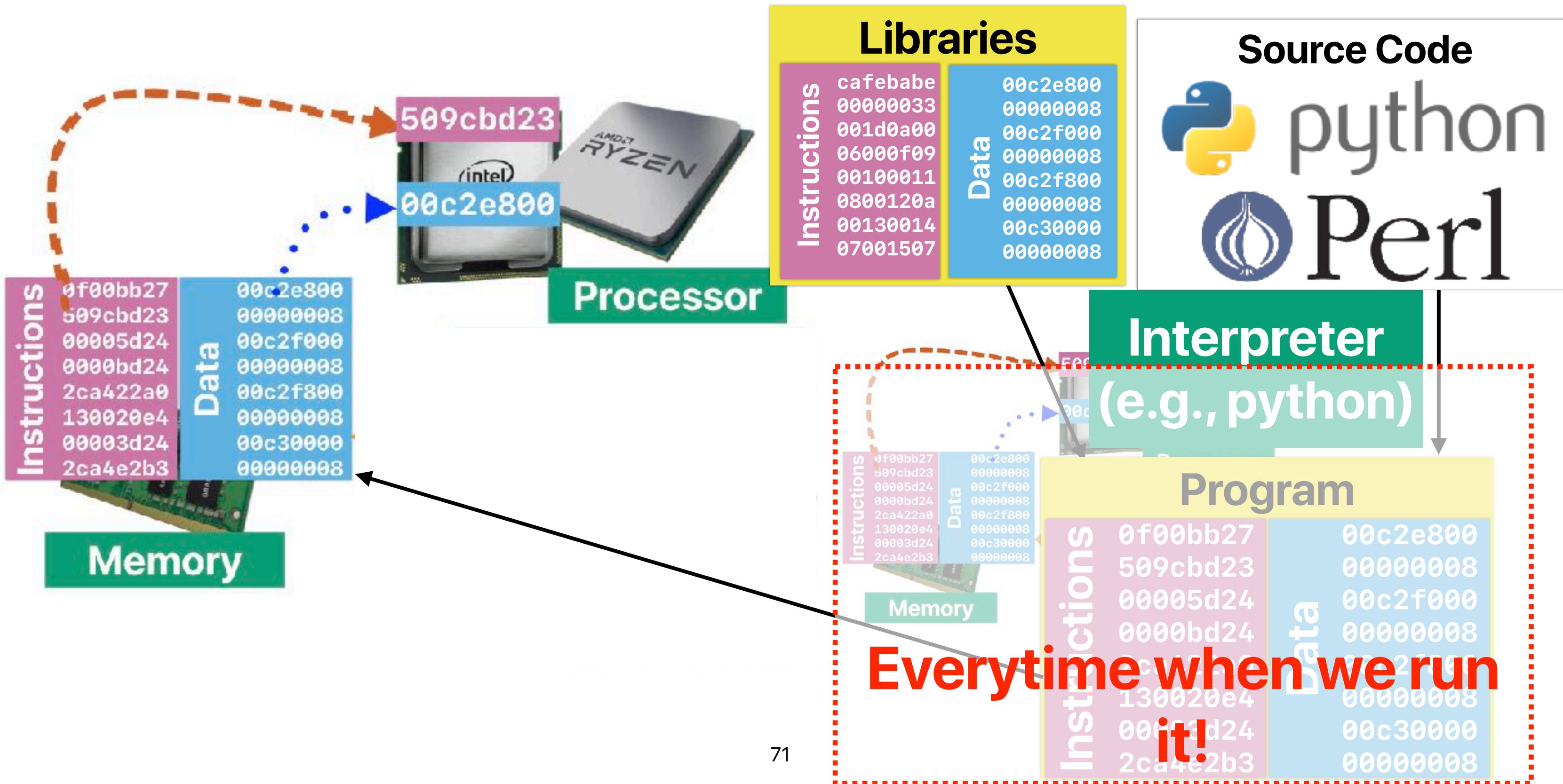
# Recap: How my “C code” becomes a “program”



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



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



# Revisited the demo with compiler optimizations!

- gcc has different optimization levels.
  - -O0 — no optimizations
  - -O3 — typically the best-performing optimization

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 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**

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

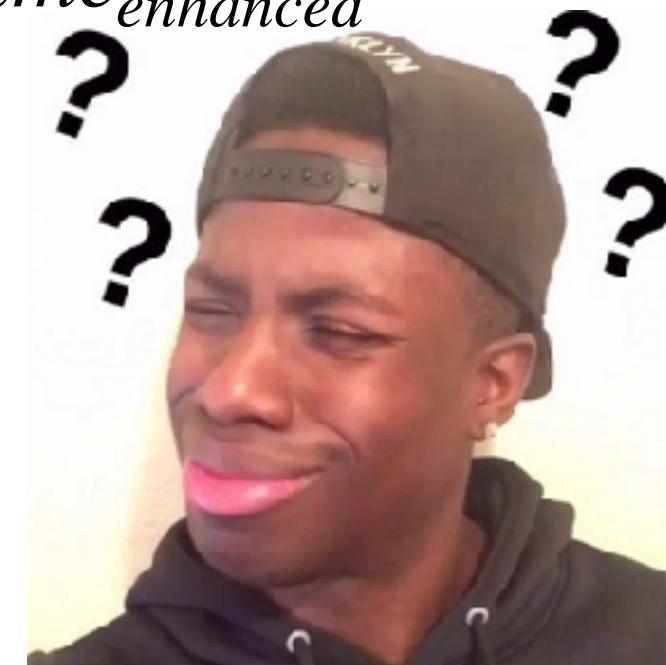
# Amdahl's Law



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

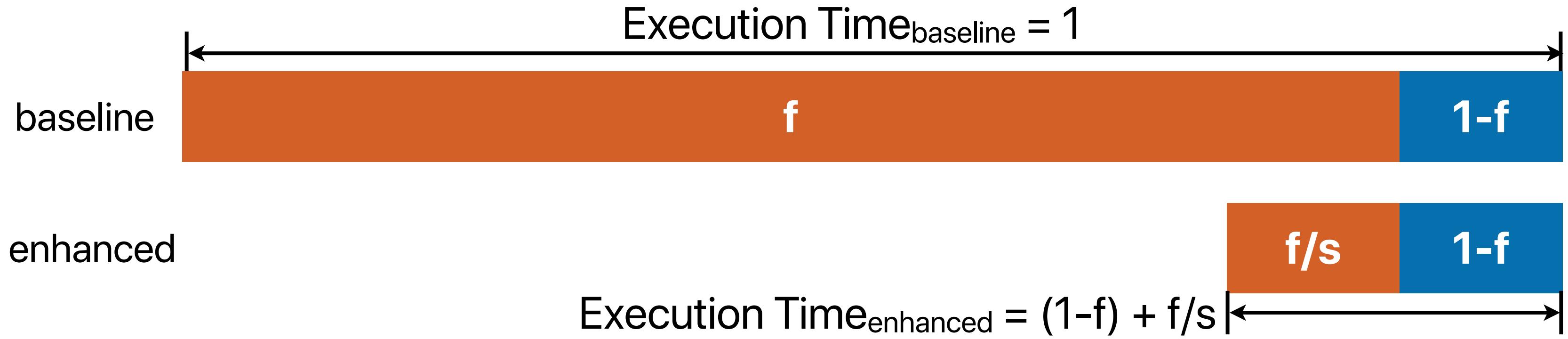
- $f$  — The fraction of time in the original program  
 $s$  — The speedup we can achieve on  $f$

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



# 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

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

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

# Corollary #1 on Multiple Optimizations

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



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

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

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

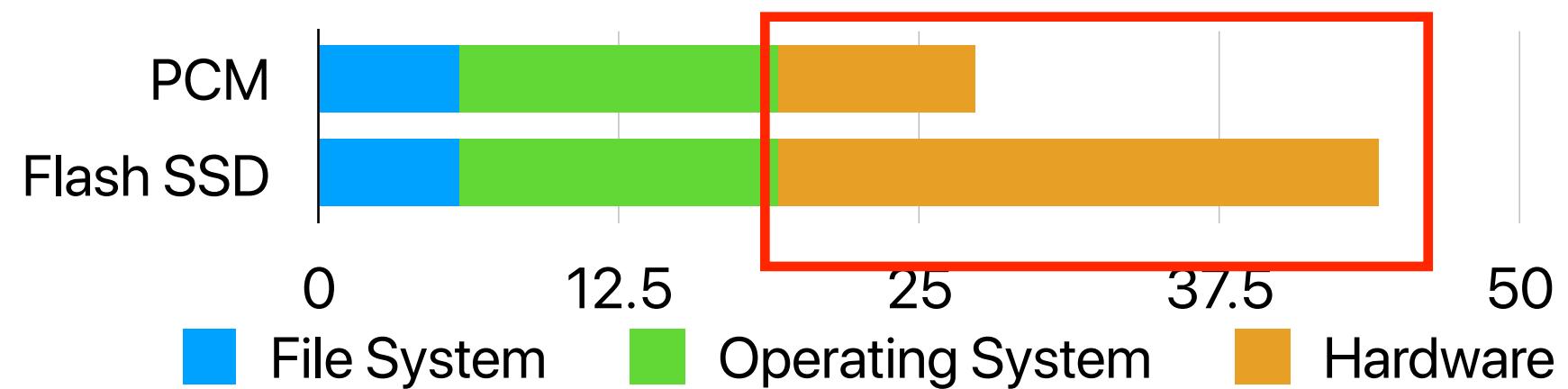
$$Speedup_{max}(f_4, \infty) = \frac{1}{(1 - f_4)}$$

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.

# Speedup further!



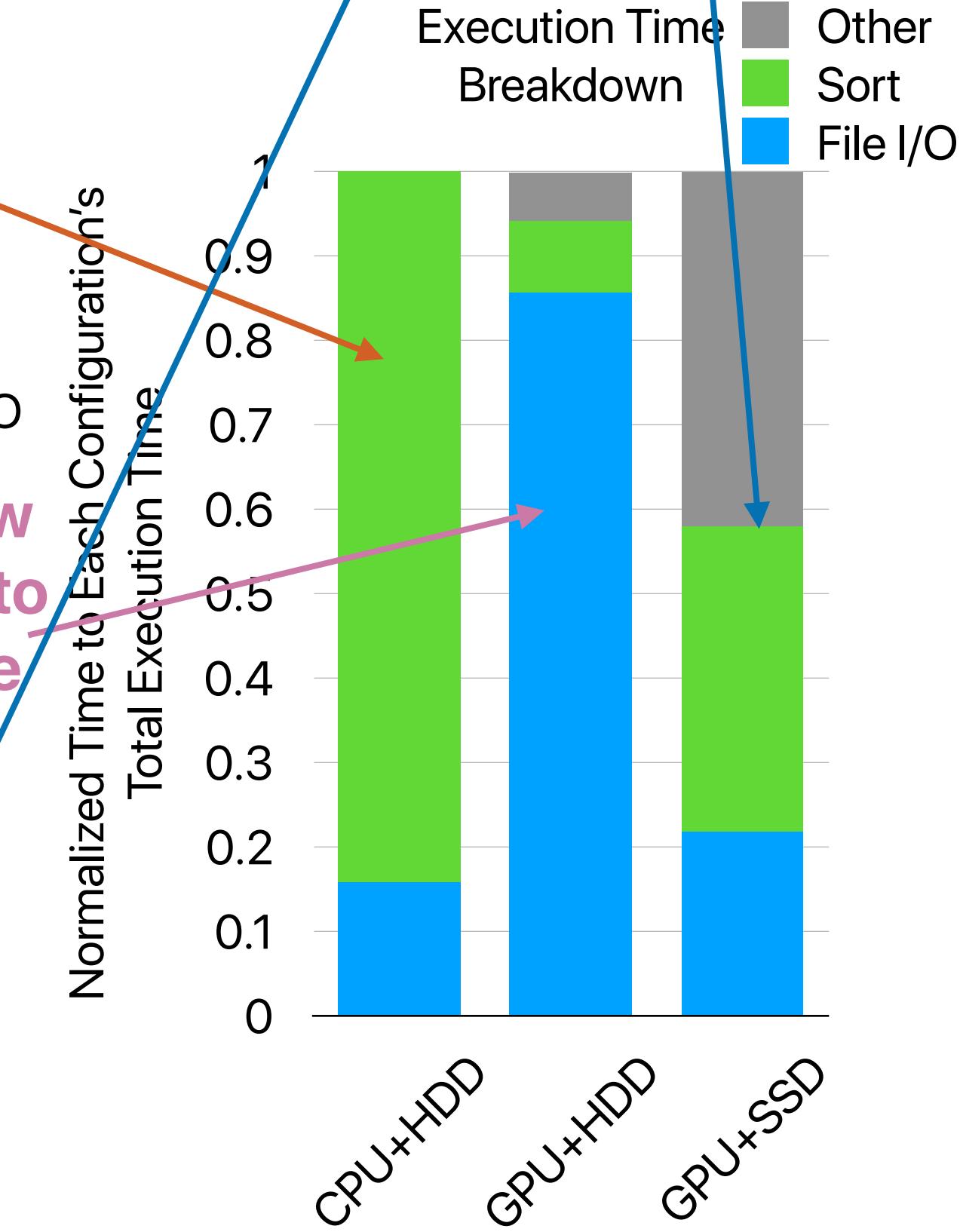
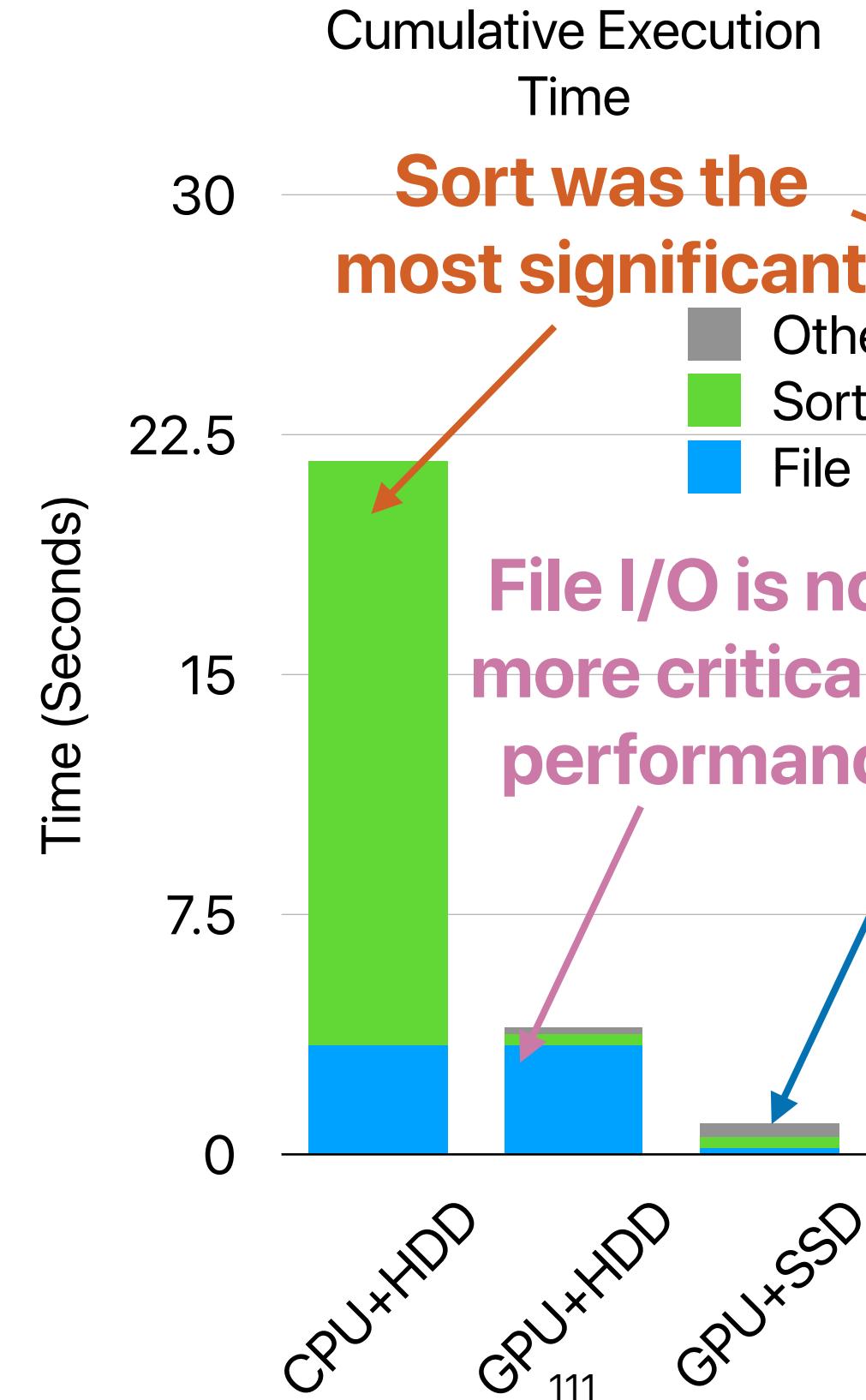
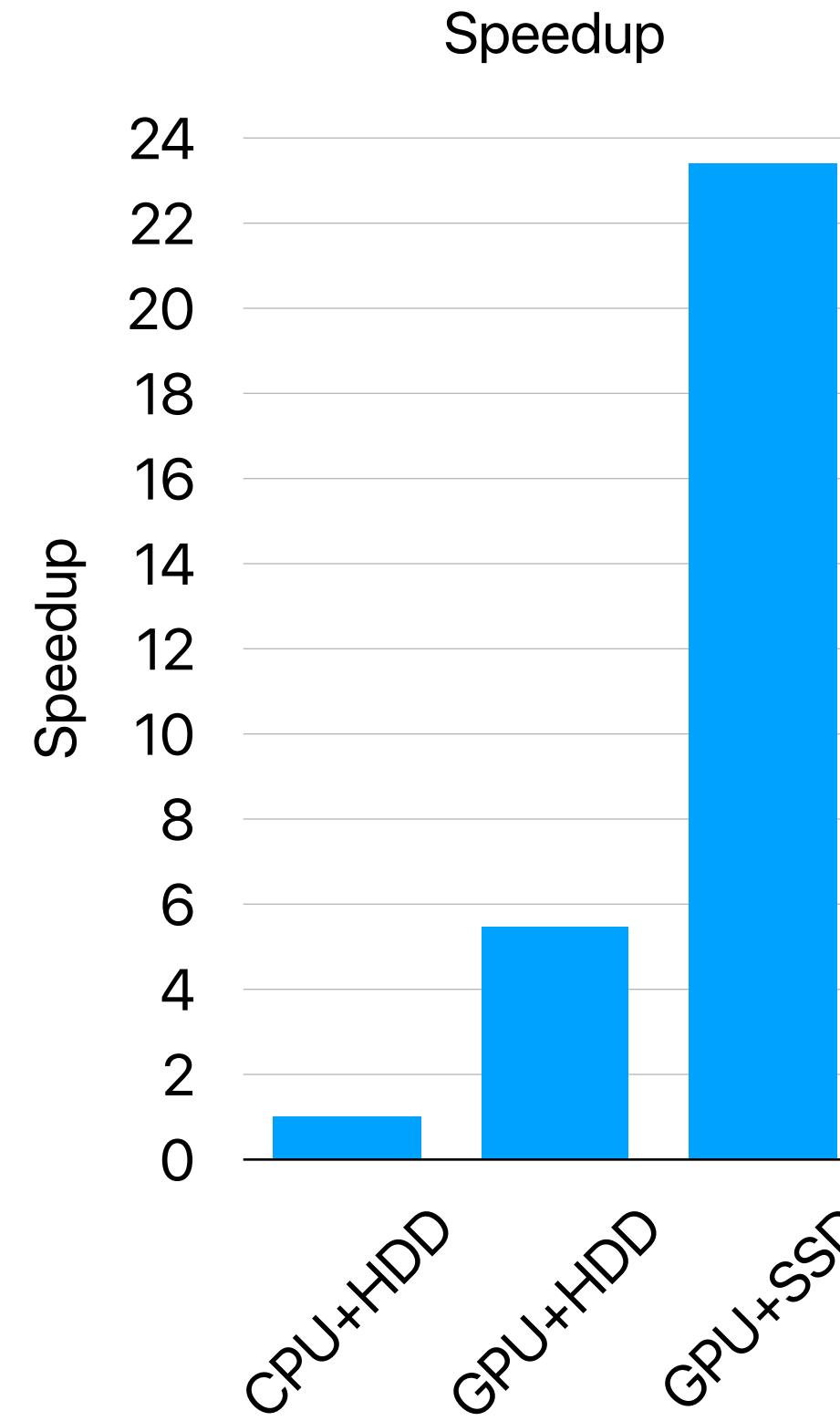
**With optimization, the common becomes uncommon.**

# Identify the most time consuming part

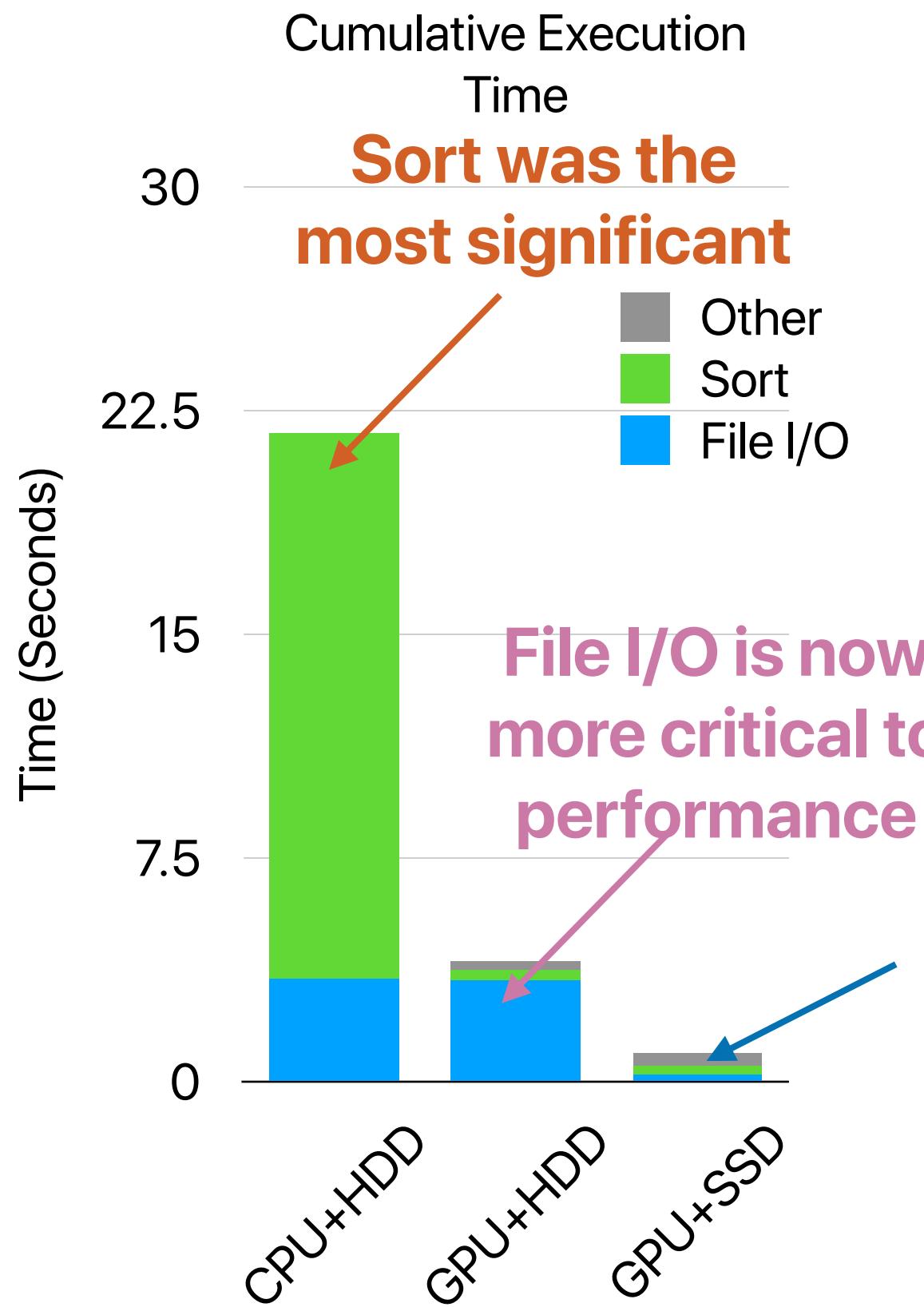
- Compile your program with -pg flag
- Run the program
  - It will generate a gmon.out
  - `gprof your_program gmon.out > your_program.prof`
- It will give you the profiled result in `your_program.prof`

# Demo — sort

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



# 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

# 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

# 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

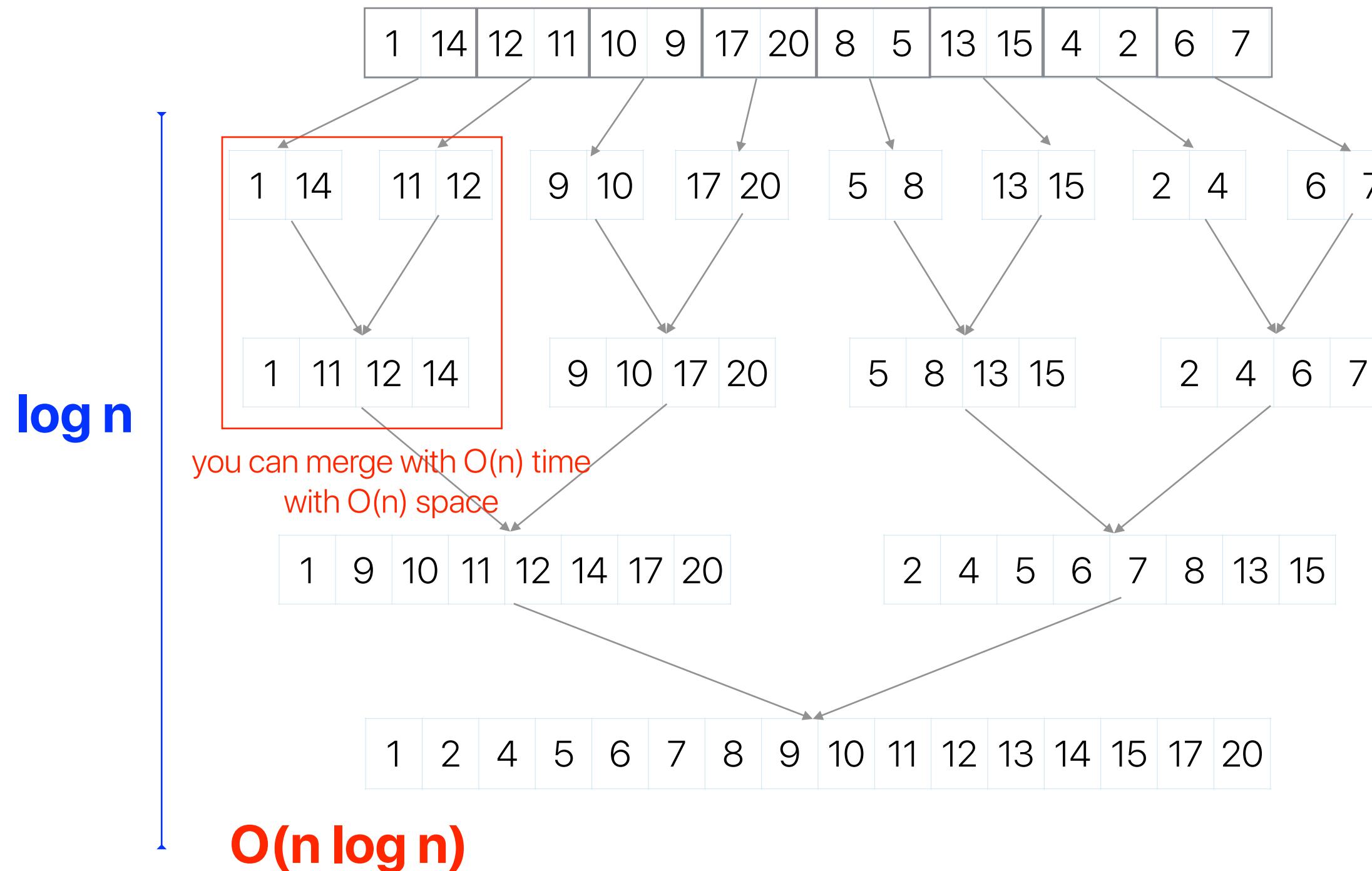
$$O(n \log_2 n)$$

## Bitonic Sort

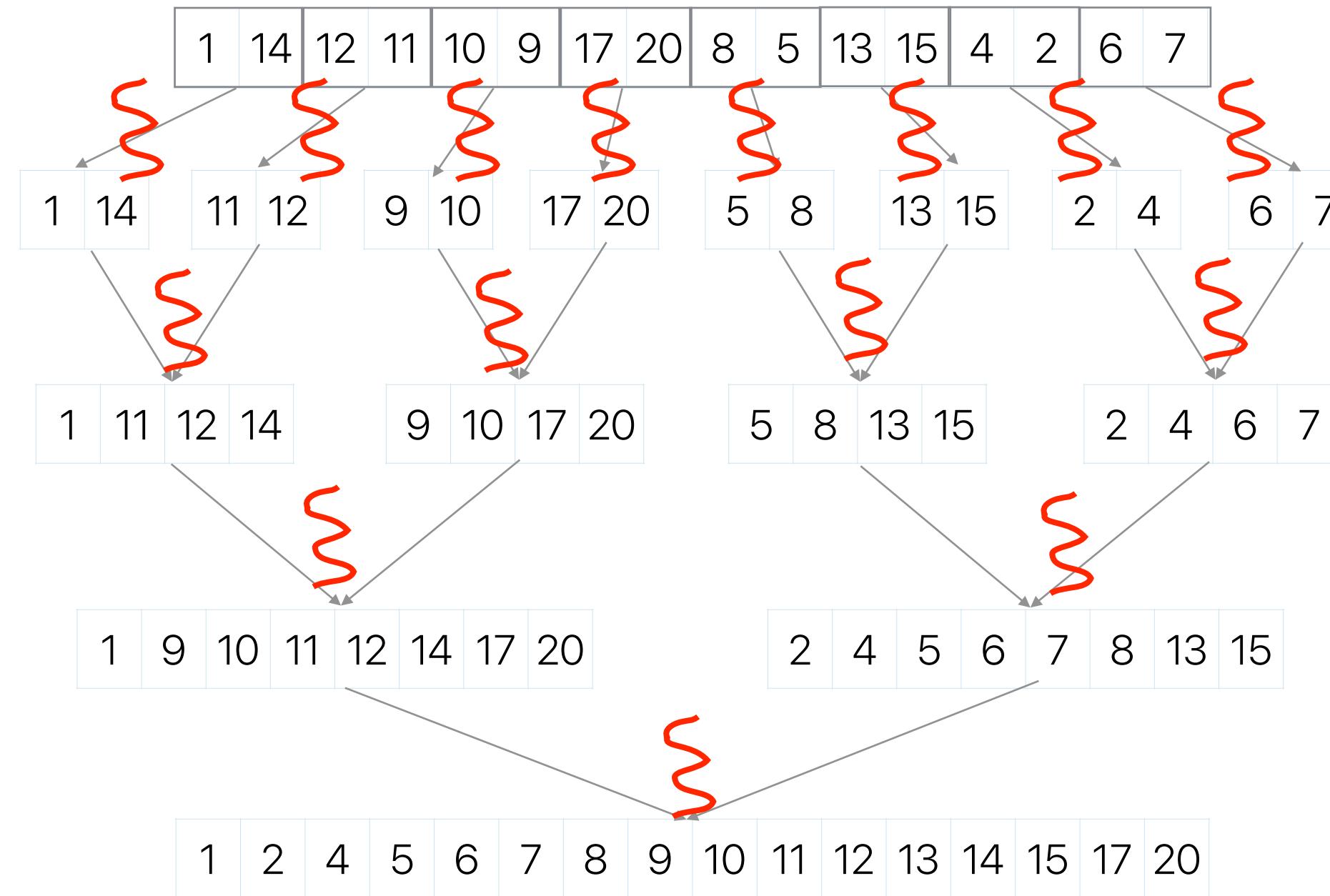
$$O(n \log_2^2 n)$$

```
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);  
                }  
            }  
        }  
    }  
}
```

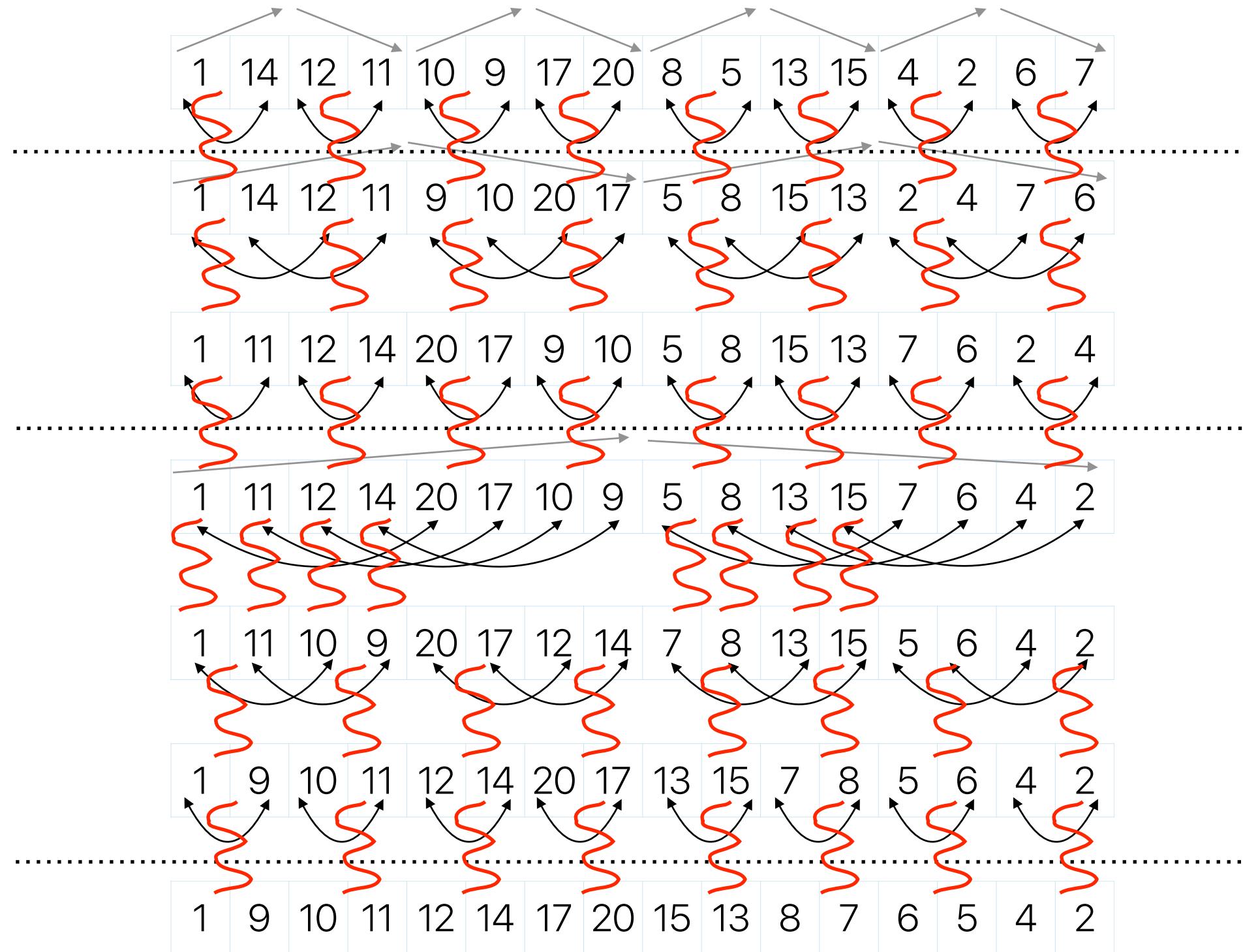
# Merge sort



# Parallel merge sort

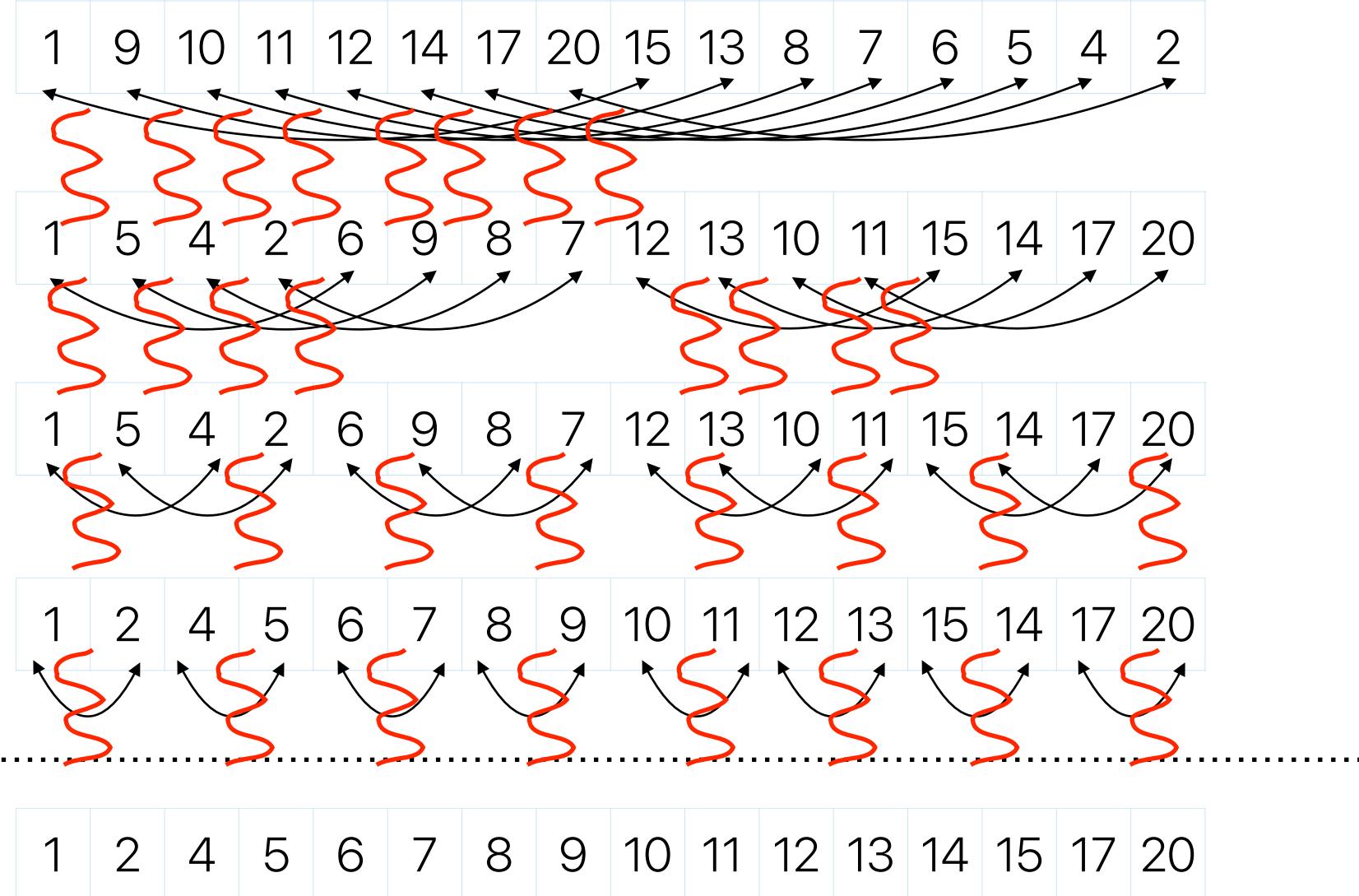


# Bitonic sort



```
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



# Lessons learned from Amdahl's Law

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

- Corollary #1: Maximum speedup
- Corollary #2: Make the common case fast
  - Common case changes all the time
- Corollary #3: Optimization is a moving target
- Corollary #4: Exploiting more parallelism from a program is the key to performance gain in modern architectures
- Corollary #5: Single-core performance still matters

$$\begin{aligned}Speedup_{max}(f, \infty) &= \frac{1}{(1 - f)} \\Speedup_{max}(f_1, \infty) &= \frac{1}{(1 - f_1)} \\Speedup_{max}(f_2, \infty) &= \frac{1}{(1 - f_2)} \\Speedup_{max}(f_3, \infty) &= \frac{1}{(1 - f_3)} \\Speedup_{max}(f_4, \infty) &= \frac{1}{(1 - f_4)}\end{aligned}$$

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

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

# Round #1

# Bard

Who is Hung-Wei Tseng?

Hung-Wei Tseng is an associate professor in the Department of Electrical and Computer Engineering and a cooperating faculty of the Department of Computer Science and Engineering at University of California, Riverside. He is now leading the Extreme Storage & Computer Architecture Laboratory. He is interested in diverse research topics that allow applications or programmers to more efficiently use modern heterogeneous hardware components. Together with his students, they have demonstrated how intelligent storage devices can help improve performance, power and energy for data-intensive systems.

Their work has been recognized by IEEE Micro "Top Picks from the 2019 Computer Architecture Conferences" (IEEE MICRO Top Picks 2020) and Facebook Research Award, 2018. Prior to joining UCR, he served as an assistant professor for the Department of Computer Science and the Department of Electrical and Computer Engineering at NC State University. He was a PostDoc of the Non-volatile Systems Laboratory and a lecturer of the Department of Computer Science and Engineering at University of California, San Diego with Professor Steven Swanson.

Sources - Learn more  
1. [ictra.ugr.es/~htseng/](http://ictra.ugr.es/~htseng/)  
2. [www.nvsl.io/portfolio/hung-wei-tseng/](http://www.nvsl.io/portfolio/hung-wei-tseng/)

Like Dislike Share Google It

How important is Computer Architecture in Computer Science?

1 answer/6 secs



I am unable to determine who Hung-Wei Tseng is. Can you please provide me with more context or details about the individual in question?

HT Who is Hung-Wei Tseng at UCR?

Hung-Wei Tseng is an Associate Professor of Computer Science and Engineering at the University of California, Riverside (UCR). His research interests are in the area of computer networks and distributed systems, with a focus on improving network performance and security. Some of his recent research work has focused on topics such as network measurement and monitoring, network optimization, and network security. Hung-Wei Tseng has been a faculty member at UCR since 2013, and he has received several awards and recognitions for his research contributions.

Regenerate response

How important is Computer Architecture in Computer Science?

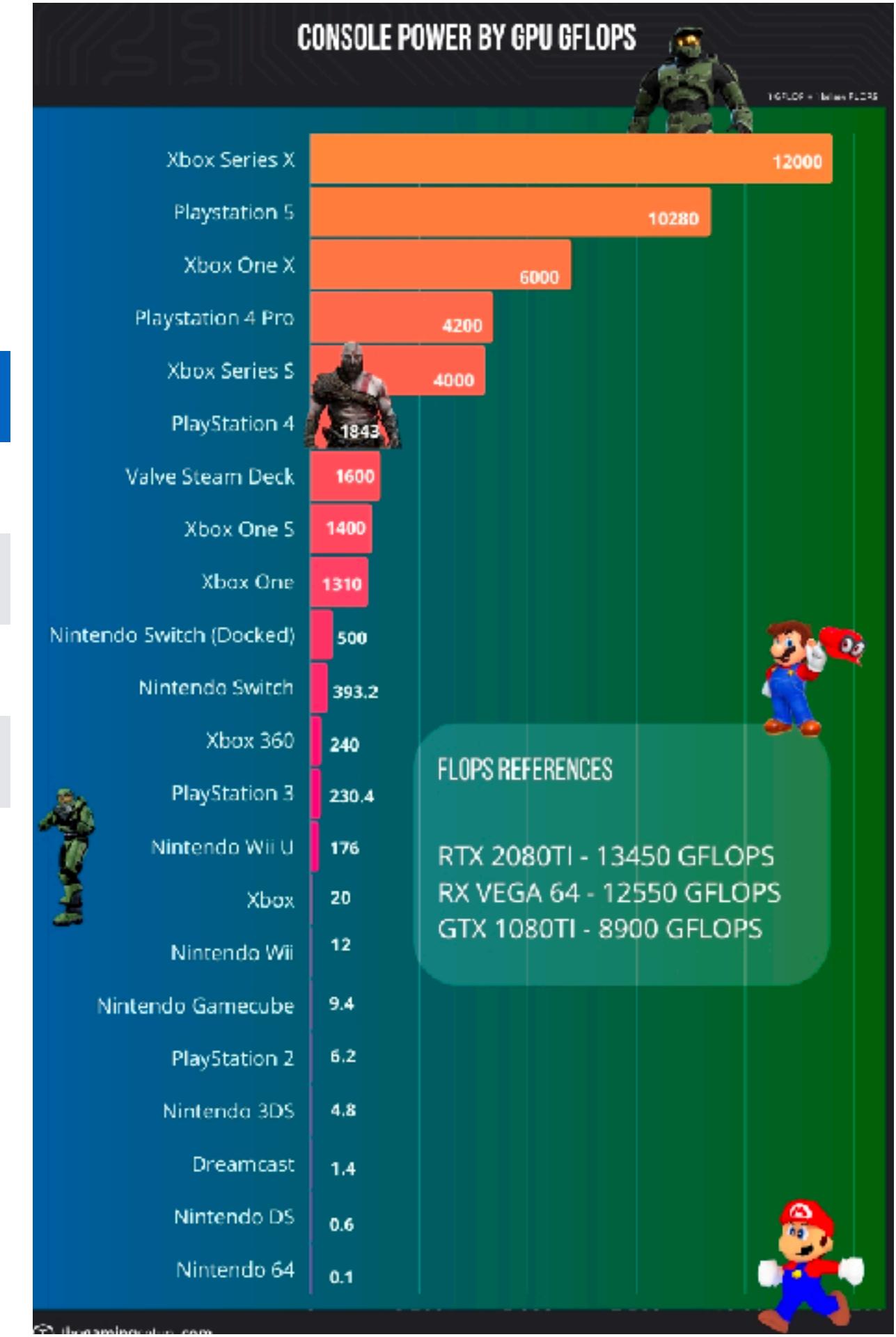
1 answer/18 secs

# 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)

# 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:**

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

**Given**  $i = j = k = 2048$

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

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

nvidia.com

# NVIDIA

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

Configuration	Time to Solution in Hours
8X Tesla V100	5.1 Hours
8X Tesla P100	15.5 Hours

Time to Solution in Hours—Lower Is Better

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.

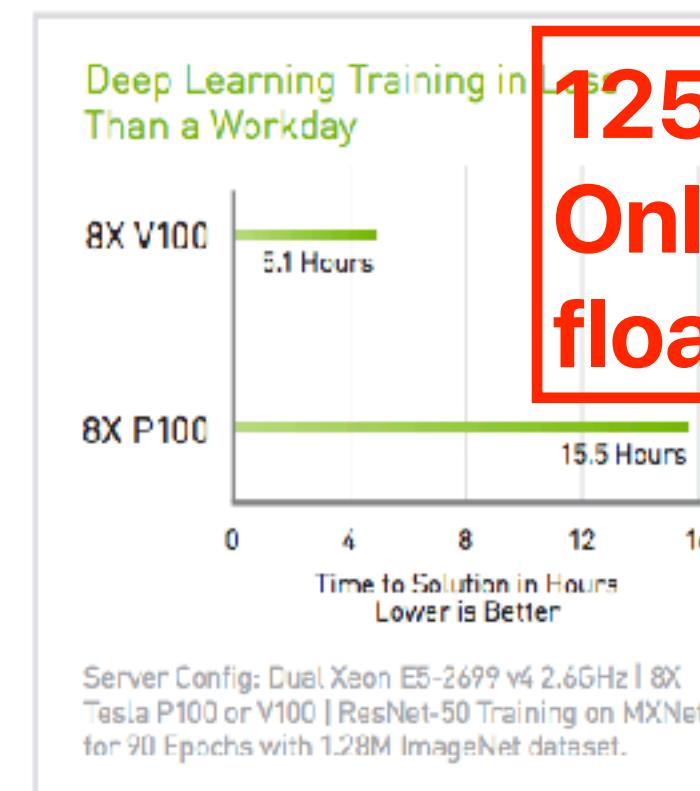
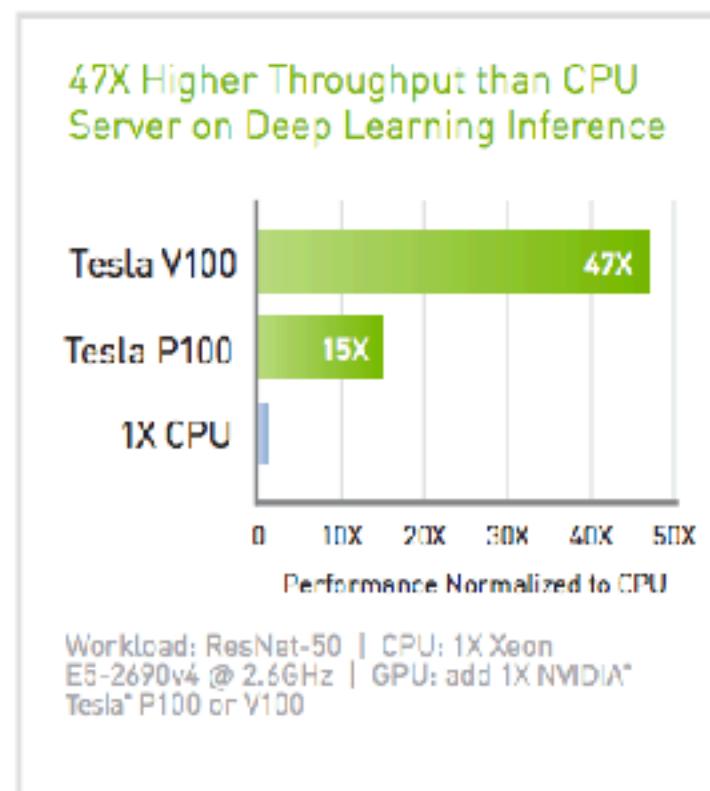
## AI TRAINING

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.

# 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

# They try to tell it's the better AI hardware

<https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/>

	K80 2012	TPU 2015	P40 2016
<b>Inferences/Sec &lt;10ms latency</b>	$^{1/13}X$	1X	2X
<b>Training TOPS</b>	6 FP32	NA	12 FP32
<b>Inference TOPS</b>	6 FP32	90 INT8	48 INT8
<b>On-chip Memory</b>	16 MB	24 MB	11 MB
<b>Power</b>	300W	75W	250W
<b>Bandwidth</b>	320 GB/S	34 GB/S	350 GB/S

# Round #2

Bard



A screenshot of the Bard web interface. On the left, a sidebar lists options: 'Reset chat', 'Bard Activity', 'FAQ', and 'Help &amp; support'. The main area shows a search result for 'Who is teaching CS203 at UCR?'. It includes a profile picture of Hung-Wei Tseng, a brief bio stating he is an assistant professor in the Department of Computer Science and Engineering, and contact information: Room 406, Winston Cheng Hall, 900 University Ave., Riverside, CA 92521; phone (951) 827-4222 ext 28047; and email htseng@ucr.edu. Below the search bar, it says 'Don't see the display you like or prefer a different interface? Click here to view Google's demo.'

A screenshot of the ChatGPT web interface. On the left, a sidebar shows a list of recent chats: 'UCR CS203 Instructor', 'Hung-Wei Tseng', and 'HW Memory Adoption Best'. The main area shows a search result for 'Who is teaching CS203 at UCR?'. It includes a note that as an AI language model, ChatGPT does not have access to real-time information about who is teaching CS203 at the University of California, Riverside (UCR). It advises users to visit UCR's official website or contact the university's department of computer science directly for the most up-to-date information on course instructors. Below the search bar, it says 'Who is Hung-Wei Tseng?' and 'Create a new service'.

1 answer/6 secs

1 answer/6 secs

# Inference per second

$$\frac{\text{Inferences}}{\text{Second}} = \frac{\text{Inferences}}{\text{Operation}} \times \frac{\text{Operations}}{\text{Second}}$$

$$= \frac{\text{Inferences}}{\text{Operation}} \times [\frac{\text{operations}}{\text{cycle}} \times \frac{\text{cycles}}{\text{second}} \times \#_{\_PEs} \times \text{Utilization}_{\_PEs}]$$

	Hardware	Model	Input Data
Operations per inference		v	
Operations per cycle	v		
Cycles per second	v		
Number of PEs	v		
Utilization of PEs	v	v	
Effectual operations out of (total) operations		v	v
Effectual operations plus unexploited ineffectual operations per cycle	v		

# What's wrong with inferences per second?

- There is no standard on how they inference — but these affect!
  - What model?
  - What dataset?
  - Quality?
- That's why Facebook is trying to promote an AI benchmark — MLPerf

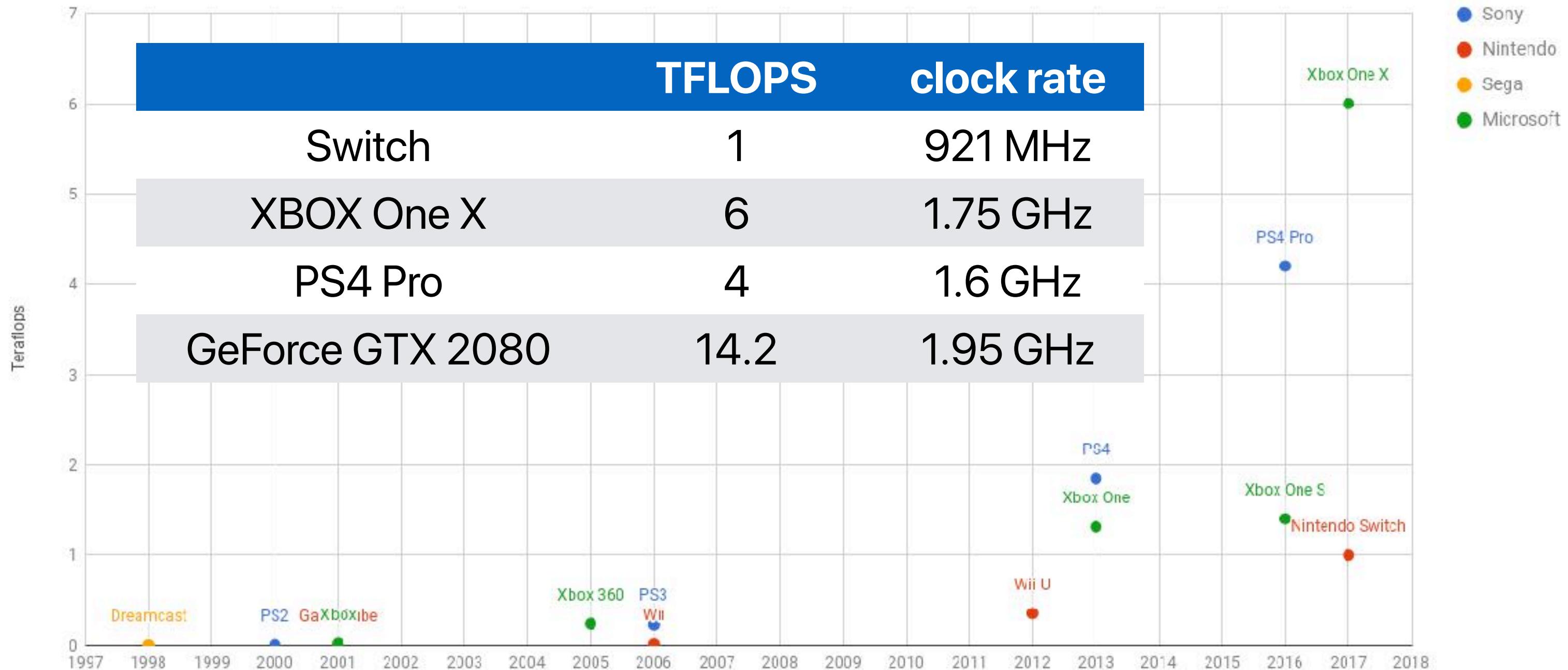
- *Pitfall: For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric.*

Our results show that IPS is a poor overall performance summary for NN hardware, as it's simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers). For example, the TPU runs the 4-layer MLP1 at 360,000 IPS but the 89-layer CNN1 at only 4,700 IPS, so TPU IPS vary by 75X! Thus, using IPS as the single-speed summary is *even more misleading* for NN accelerators than MIPS or FLOPS are for regular processors [23], so IPS should be even more disparaged. To compare NN machines better, we need a benchmark suite written at a high-level to port it to the wide variety of NN architectures. Fathom is a promising new attempt at such a benchmark suite [3].

# **“Fair” Comparisons**

# TFLOPS (Tera FLoating-point Operations Per Second)

Console Teraflops



## 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

# TFLOPS (Tera FLoating-point Operations Per Second)

- 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

	TFLOPS	clock rate
Switch	1	921 MHz
XBOX One X	6	1.75 GHz
PS4 Pro	4	1.6 GHz
GeForce GTX 2080	14.2	1.95 GHz

nvidia.com

NVIDIA

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

8X Tesla V100 5.1 Hours

8X Tesla P100 15.5 Hours

Time to Solution in Hours—Lower Is Better

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.

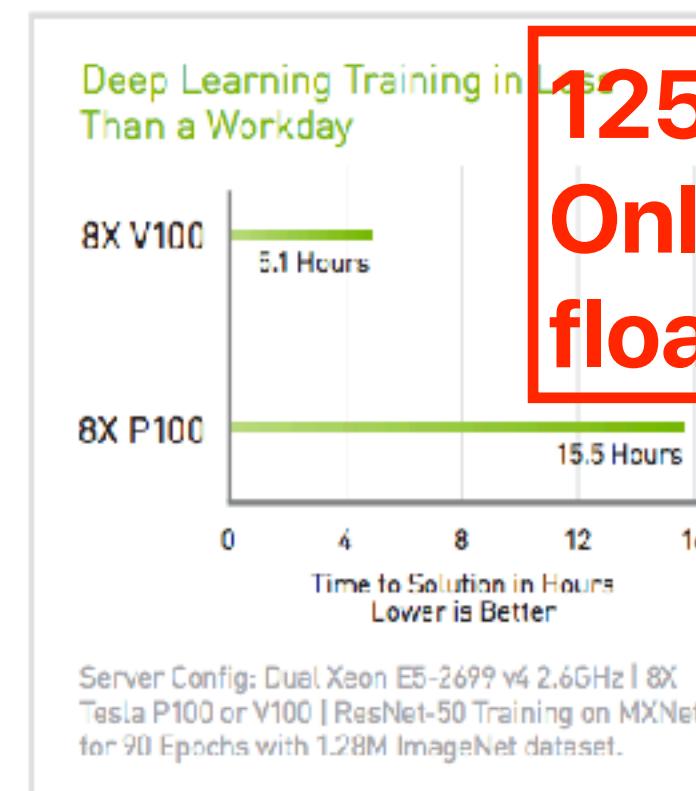
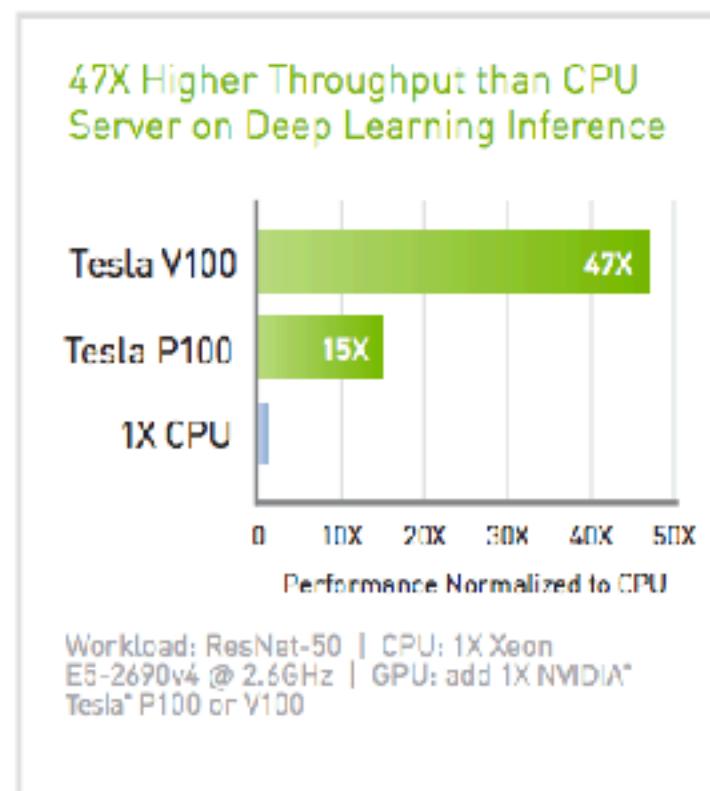
## AI TRAINING

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.

# 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

# They try to tell it's the better AI hardware

<https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/>

	K80 2012	TPU 2015	P40 2016
<b>Inferences/Sec &lt;10ms latency</b>	$^{1/13}X$	1X	2X
<b>Training TOPS</b>	6 FP32	NA	12 FP32
<b>Inference TOPS</b>	6 FP32	90 INT8	48 INT8
<b>On-chip Memory</b>	16 MB	24 MB	11 MB
<b>Power</b>	300W	75W	250W
<b>Bandwidth</b>	320 GB/S	34 GB/S	350 GB/S

# Inference per second

$$\frac{\text{Inferences}}{\text{Second}} = \frac{\text{Inferences}}{\text{Operation}} \times \frac{\text{Operations}}{\text{Second}}$$

$$= \frac{\text{Inferences}}{\text{Operation}} \times [\frac{\text{operations}}{\text{cycle}} \times \frac{\text{cycles}}{\text{second}} \times \#_{\_PEs} \times \text{Utilization}_{\_PEs}]$$

	Hardware	Model	Input Data
Operations per inference		v	
Operations per cycle	v		
Cycles per second	v		
Number of PEs	v		
Utilization of PEs	v	v	
Effectual operations out of (total) operations		v	v
Effectual operations plus unexploited ineffectual operations per cycle	v		

# What's wrong with inferences per second?

- There is no standard on how they inference — but these affect!
  - What model?
  - What dataset?
- That's why Facebook is trying to promote an AI benchmark —  
MLPerf
  - *Pitfall: For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric.*

Our results show that IPS is a poor overall performance summary for NN hardware, as it's simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers). For example, the TPU runs the 4-layer MLP1 at 360,000 IPS but the 89-layer CNN1 at only 4,700 IPS, so TPU IPS vary by 75X! Thus, using IPS as the single-speed summary is *even more misleading* for NN accelerators than MIPS or FLOPS are for regular processors [23], so IPS should be even more disparaged. To compare NN machines better, we need a benchmark suite written at a high-level to port it to the wide variety of NN architectures. Fathom is a promising new attempt at such a benchmark suite [3].

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



# Extreme Multitasking Performance

- Dual 4K external monitors
- 1080p device display
- 7 applications

# What's missing in this video clip?

- The ISA of the “competitor”
- Clock rate, CPU architecture, cache size, how many cores
- How big the RAM?
- How fast the disk?