

Performance (3): Great Pretender

Hung-Wei Tseng

N SERIES

GREAT PRETENDER

Great Pretender

2020 | **TV-MA** | 1 Season | Drama Anime

Supposedly Japan's greatest swindler, Makoto Edamura gets more than he bargained for when he tries to con Laurent Thierry, a real world-class crook.

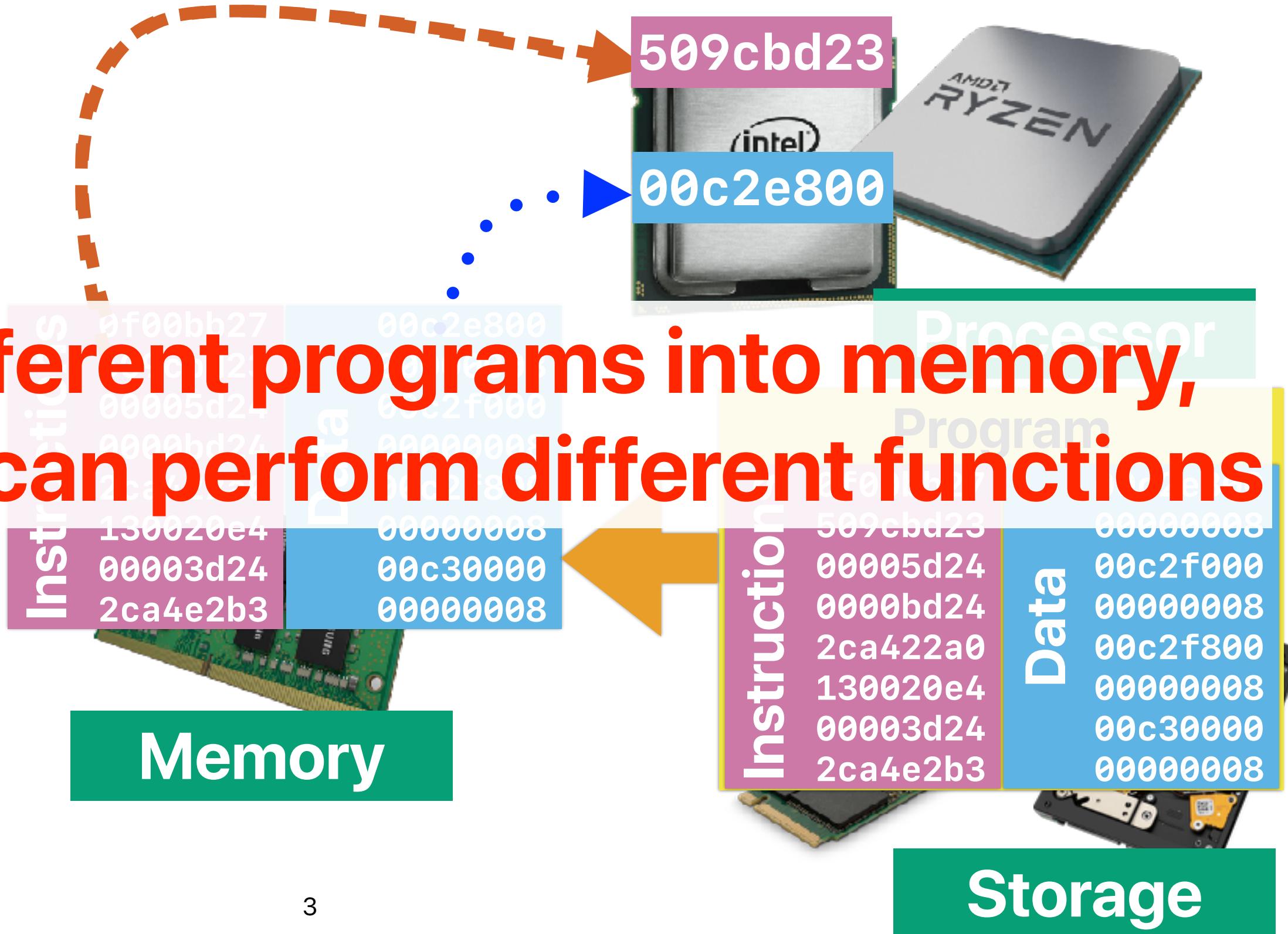
Starring: Chiaki Kobayashi, Junichi Suwabe, Natsumi Fujiwara



Recap: von Neumann Architecture



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



Recap: Summary of CPU Performance Equation

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

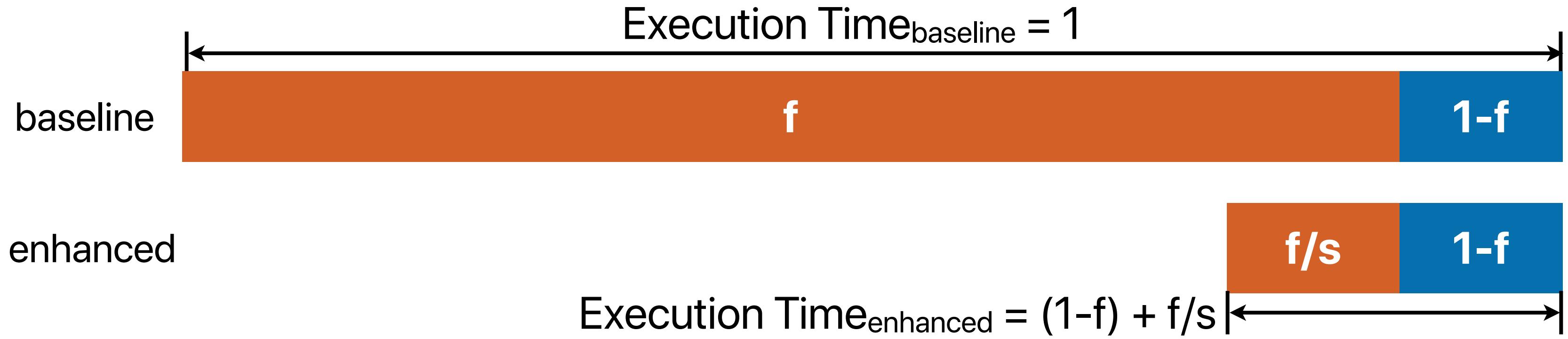
$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

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

- 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**

Recap: Amdahl's Law

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



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

Have you ever been fooled by commercials? How and why?

Outline

- Amdahl's Law and its implication in the multicore-era
- Fair comparisons or great pretenders?

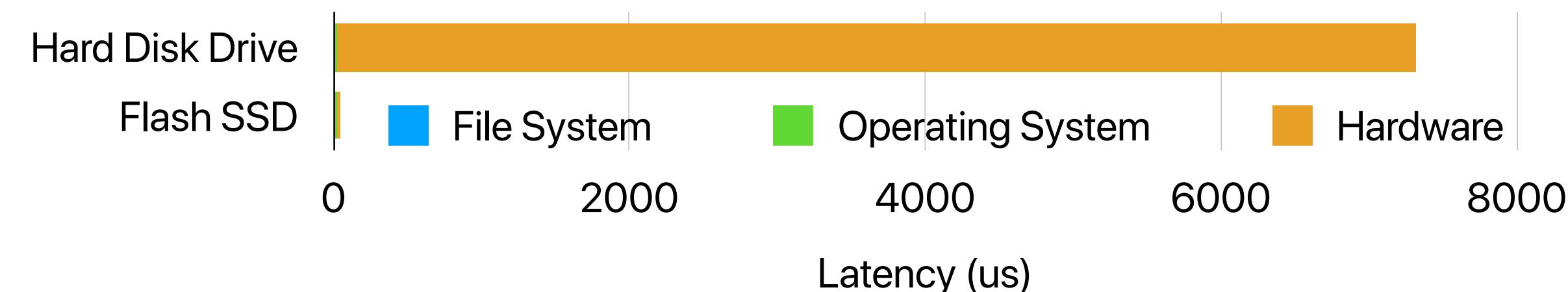
Amdahl's Law — and It's Implication in the Multicore Era



Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If your company ask you and your team to invent a new memory technology that replaces flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

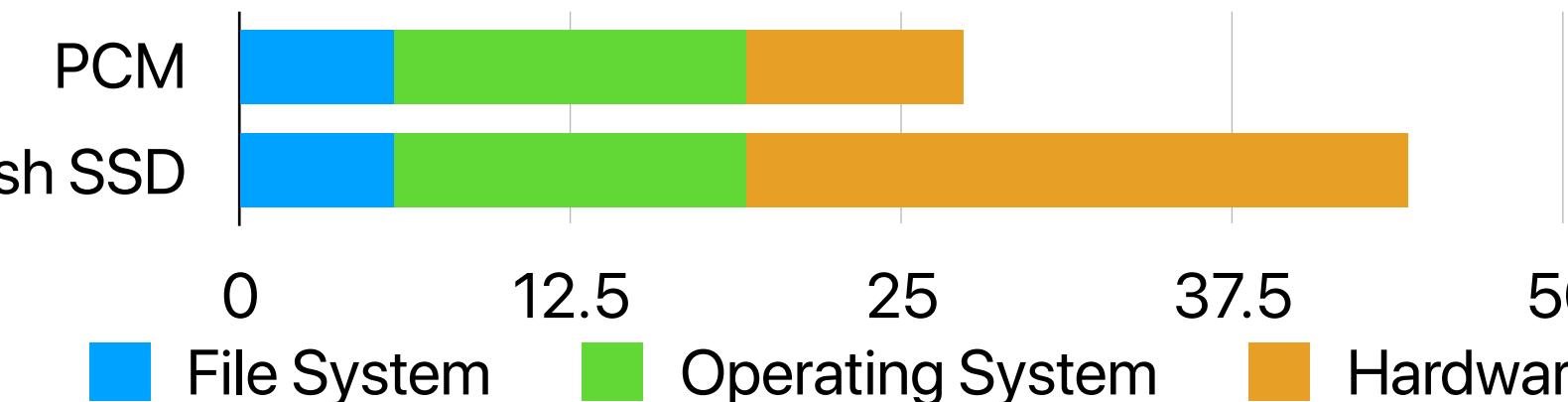
- A. ~5x
- B. ~10x
- C. ~20x
- D. ~100x
- E. None of the above



Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If your company ask you and your team to invent a new memory technology that replaces flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. ~5x



C. ~20x

D. ~100x

E. None of the above

$$\text{Speedup}_{\text{enhanced}}(16\%, x) = \frac{1}{(1 - 16\%) + \frac{16\%}{x}} = 2$$
$$x = 0.47$$

Amdahl's Law Corollary #1

- The maximum speedup is bounded by

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

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

Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If your company ask you and your team to invent a new memory technology that replaces flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

A. ~5x



B. ~10x Flash SSD

C. ~20x

D. ~100x

E. None of the above

$$\text{Speedup}_{max}(16\%, \infty) = \frac{1}{(1 - 16\%)} = 1.19$$

2x is not possible

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 mol
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
 - $ET = IC \times CPI \times CT$
 - The uncommon case doesn't make much difference
 - The common case can change based on inputs, compiler options, optimizations you've applied, etc.

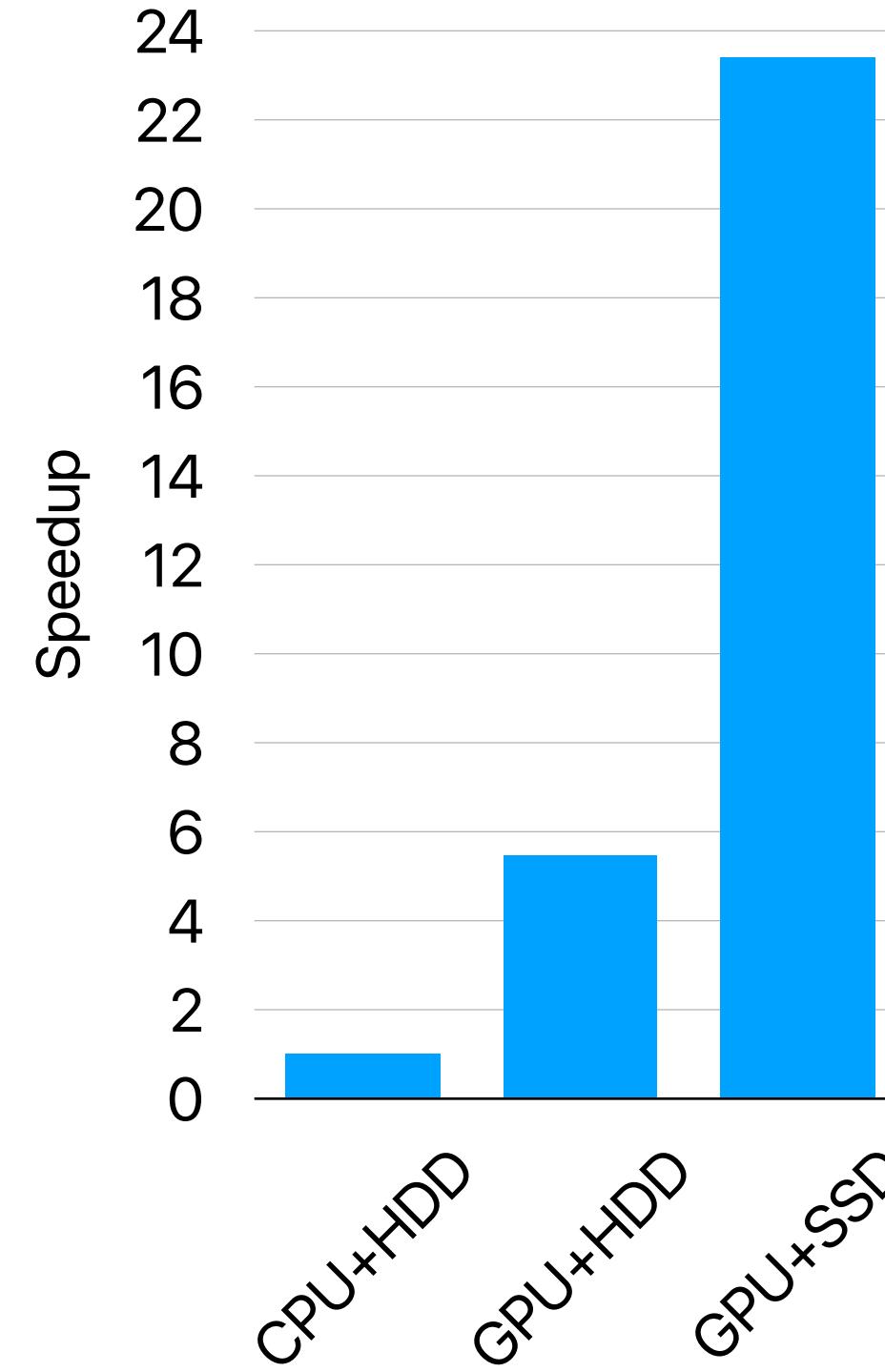
Identify the most time consuming part

- 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`

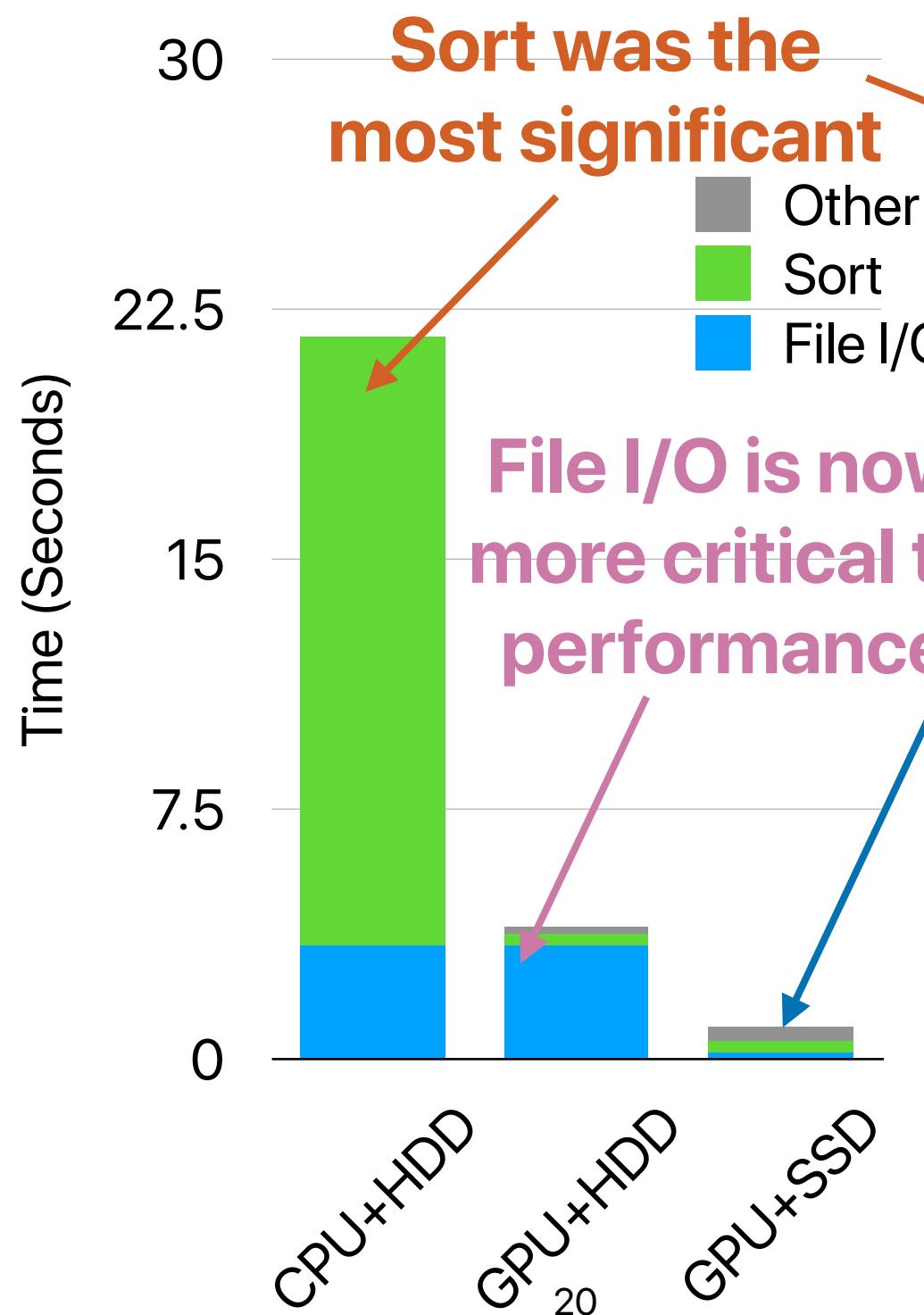
Demo — sort

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

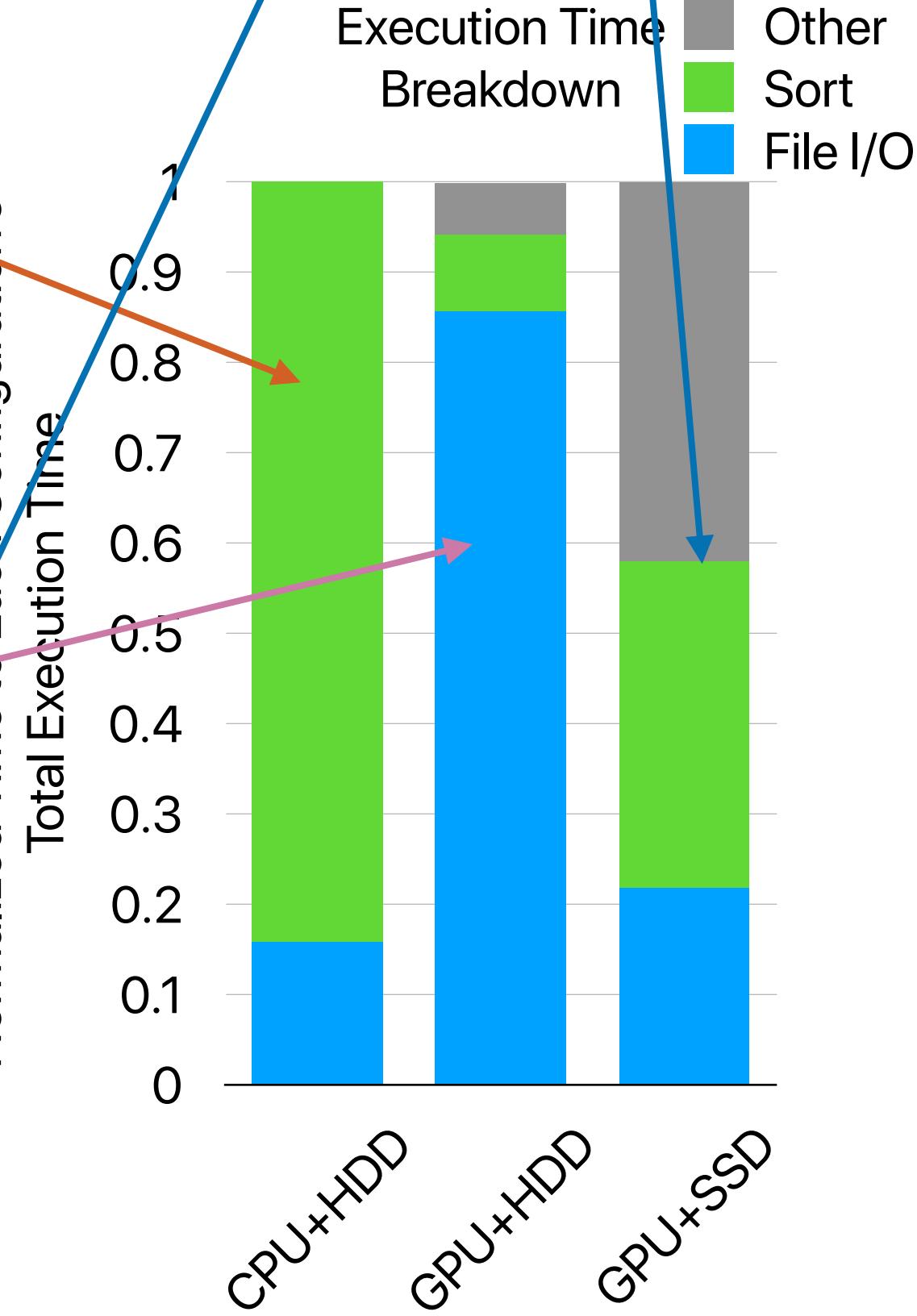
Speedup



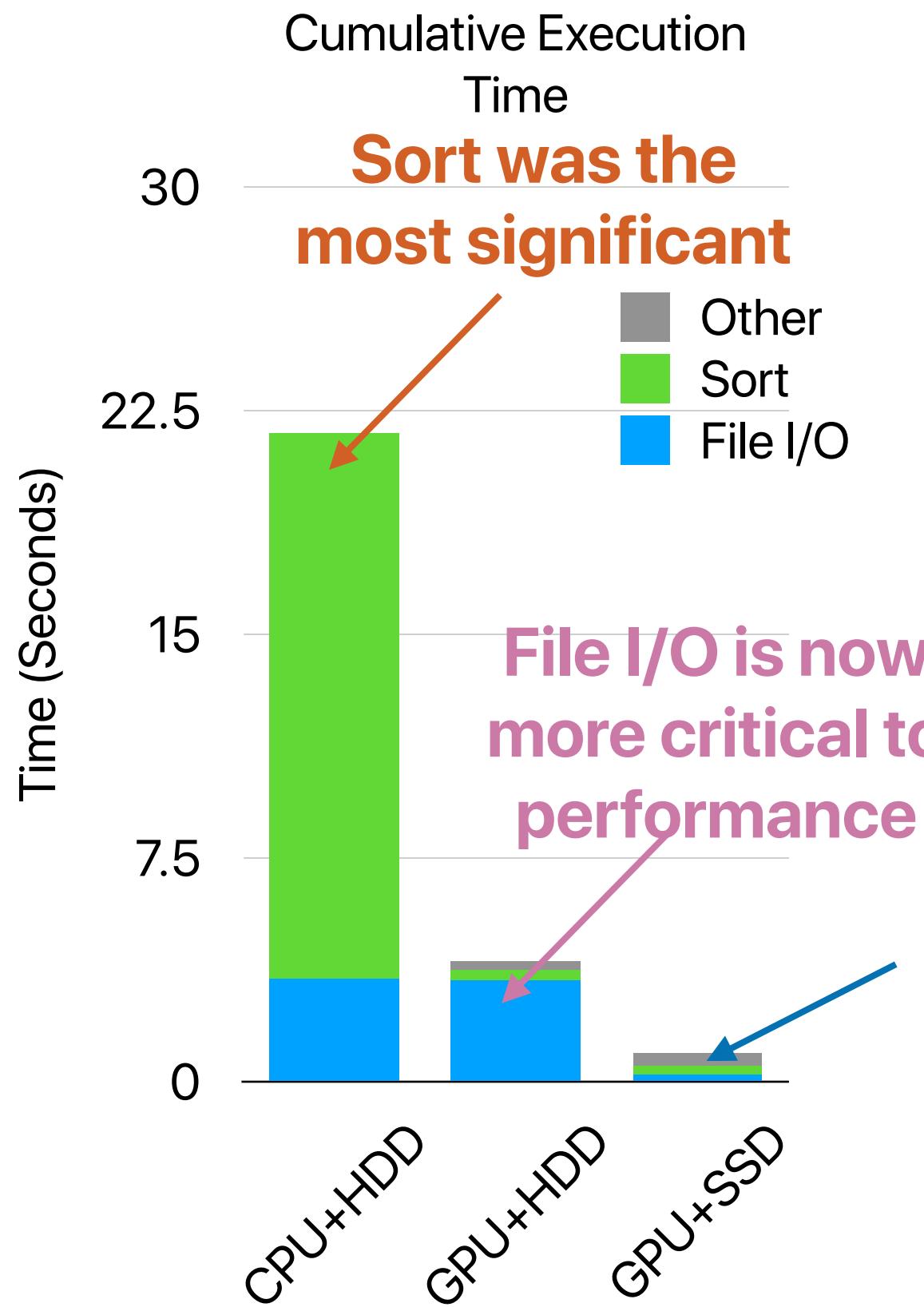
Cumulative Execution Time



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)

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



Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
 - If we have unlimited parallelism, the performance of executing each parallel partition does not matter as long as the performance slowdown in each piece is bounded
 - With unlimited amount of parallel hardware units, single-core performance does not matter anymore
 - With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
 - With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

A. 0
B. 1
C. 2
D. 3
E. 4

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?

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

- If we have unlimited parallelism, the performance of executing each parallel partition does not matter as long as the performance slowdown in each piece is bounded
 - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
 - With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
 - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor
- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

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

Merge Sort

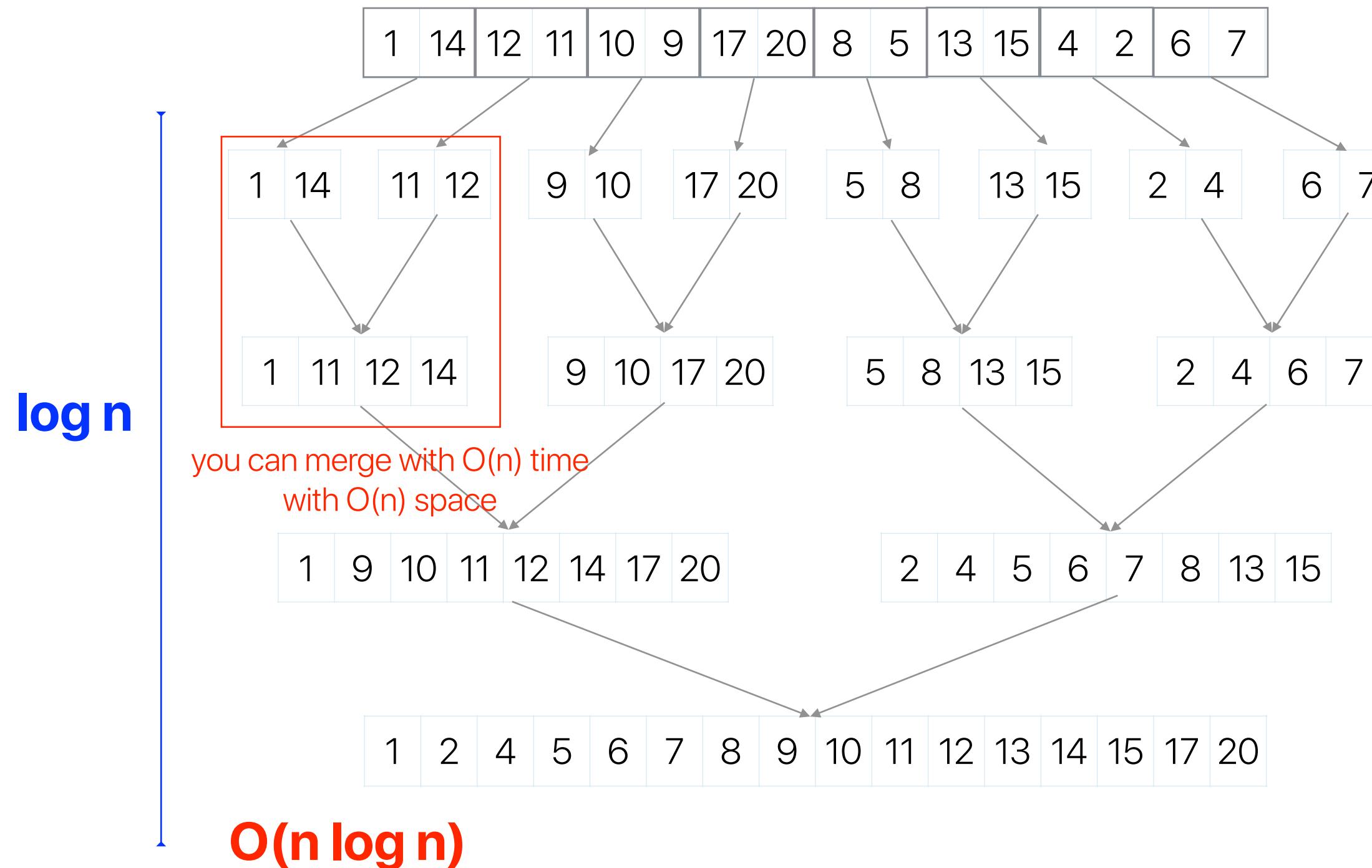
$$O(n \log_2 n)$$

Bitonic Sort

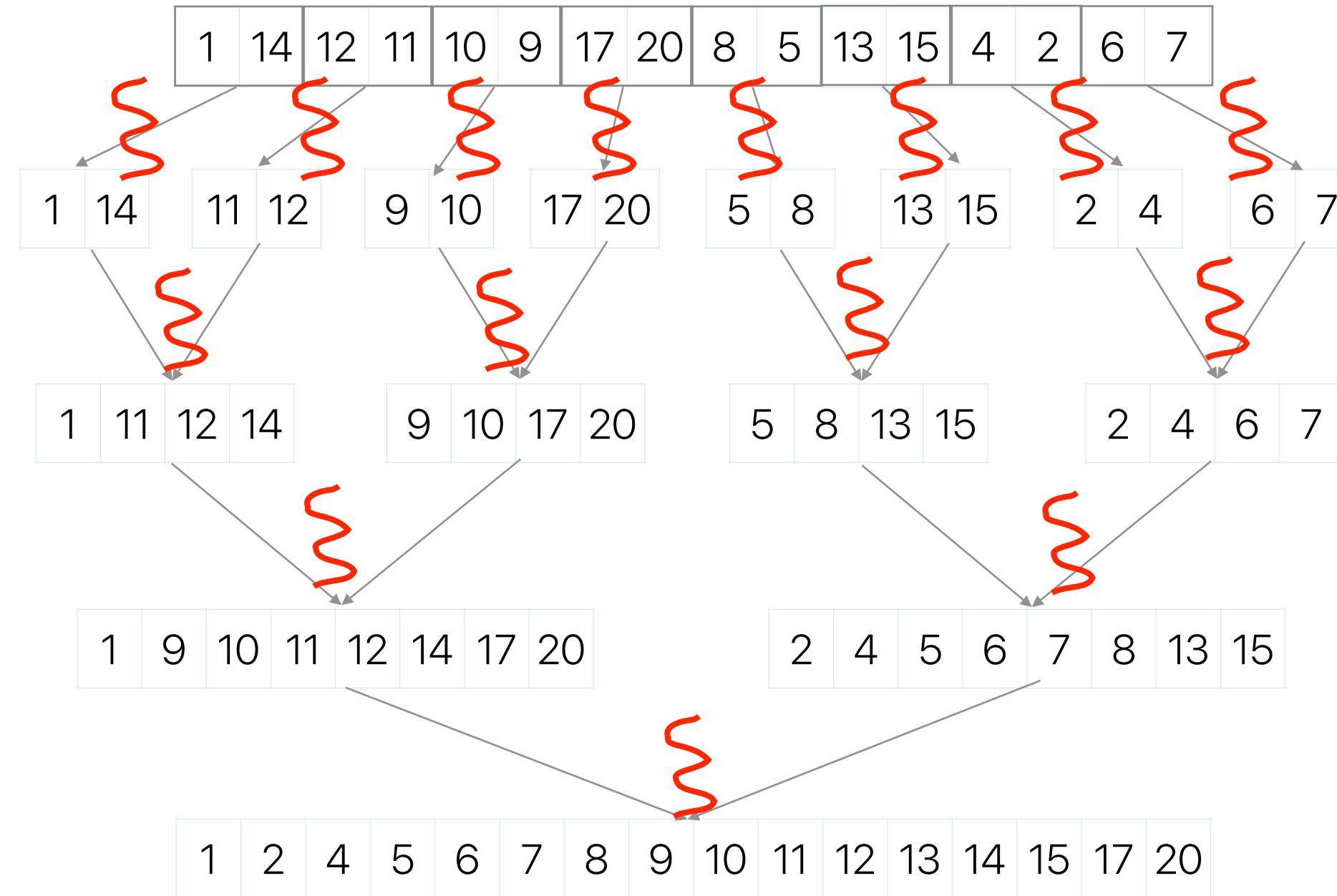
$$O(n \log_2^2 n)$$

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

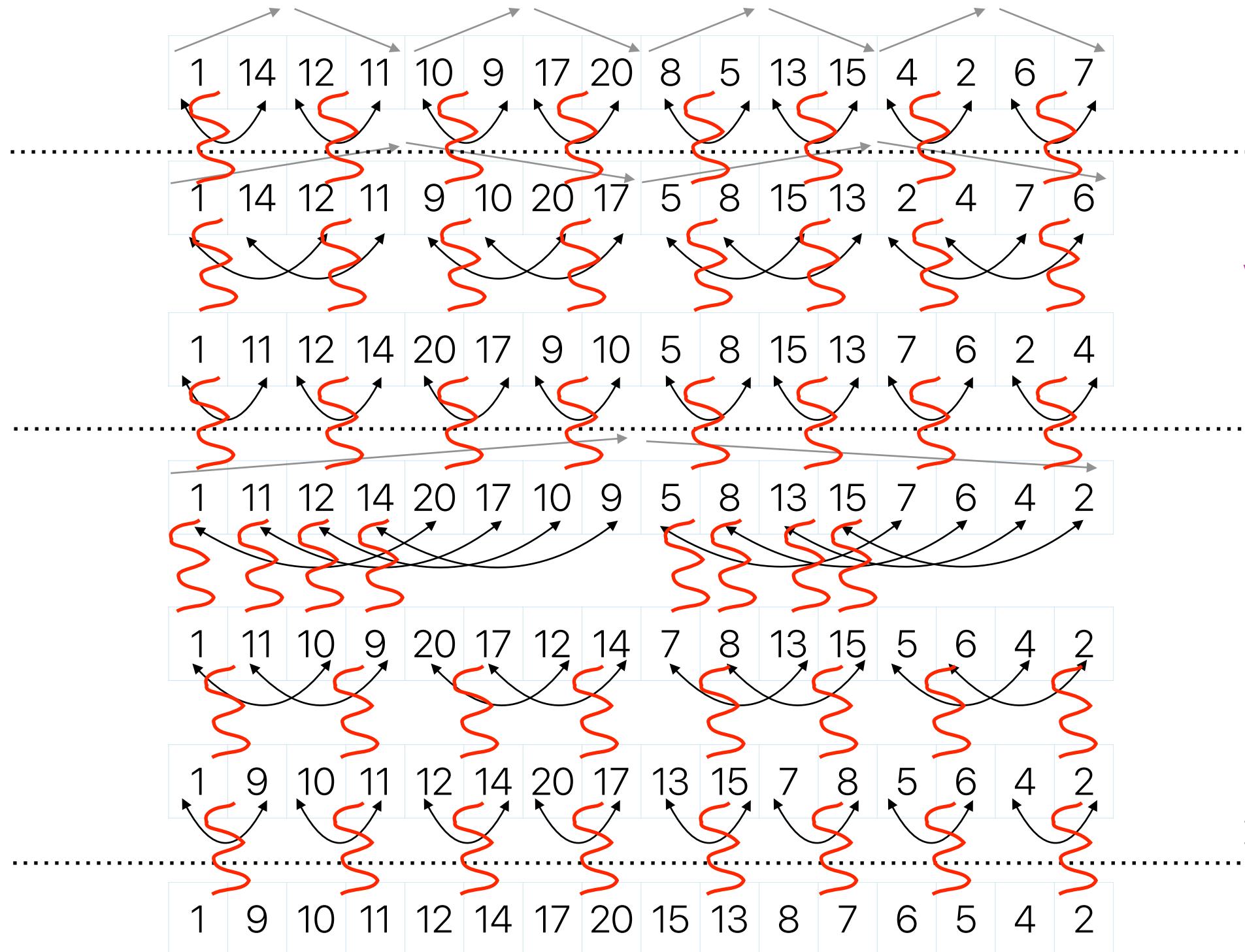
Merge sort



Parallel merge sort

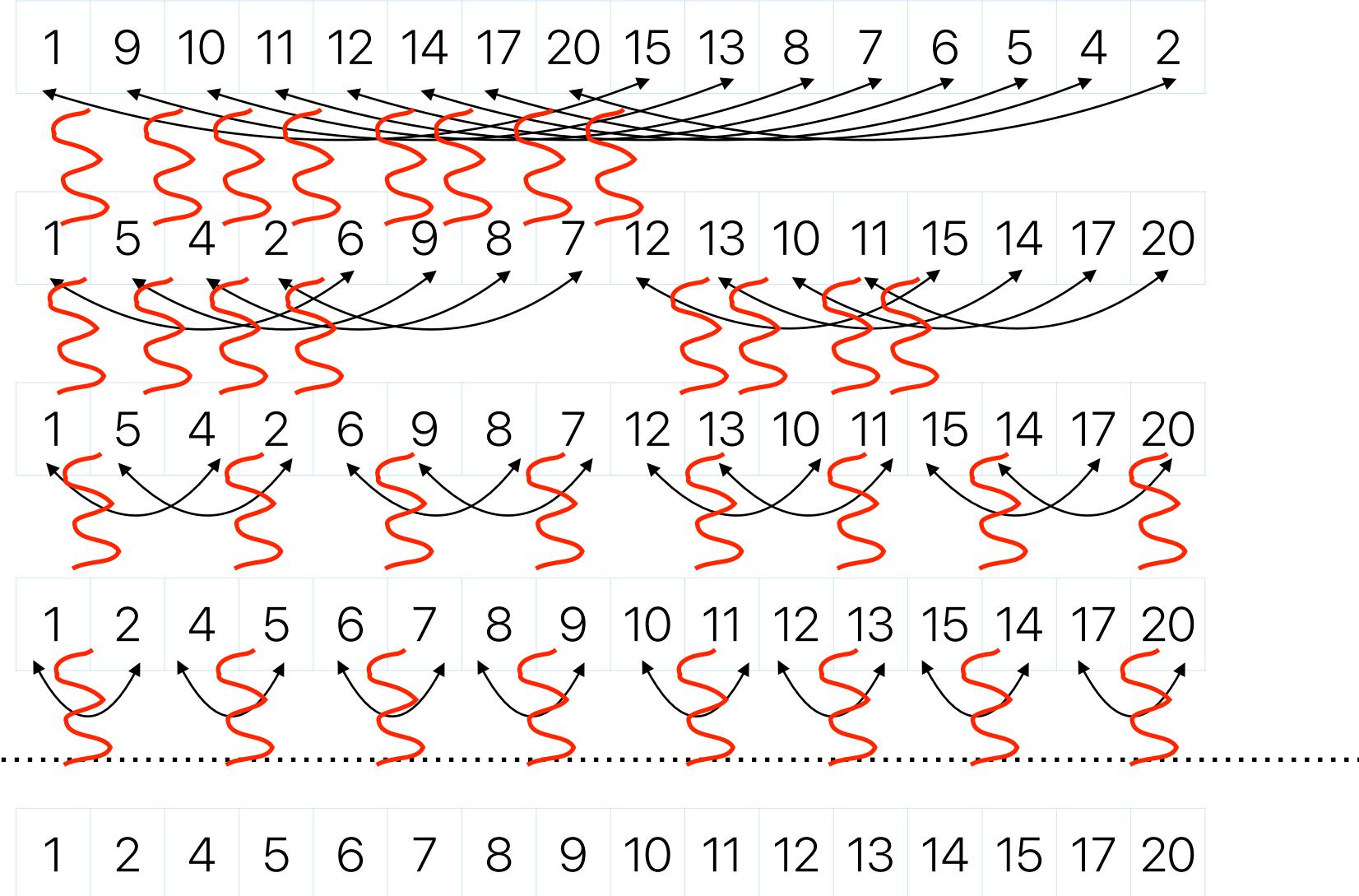


Bitonic sort



```
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”

However, parallelism is not “tax-free”

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



Amdahl's Law considering overhead

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

- r is some other parameter that affects the overhead
 - input size?
 - degree of parallelism?
- The overhead may scale differently than the original problem
 - that's why we introduce "perf()" function

Corollary #6: 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...

$$\text{Speedup} = \frac{1}{(1-f) + perf(r) + \frac{f}{s}} = \frac{1}{(1-0.9) + (1-0.9) \times 9 + \frac{0.9}{9}} = 0.91 \times$$

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

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

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

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

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

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

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

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

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

**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/
2. 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?

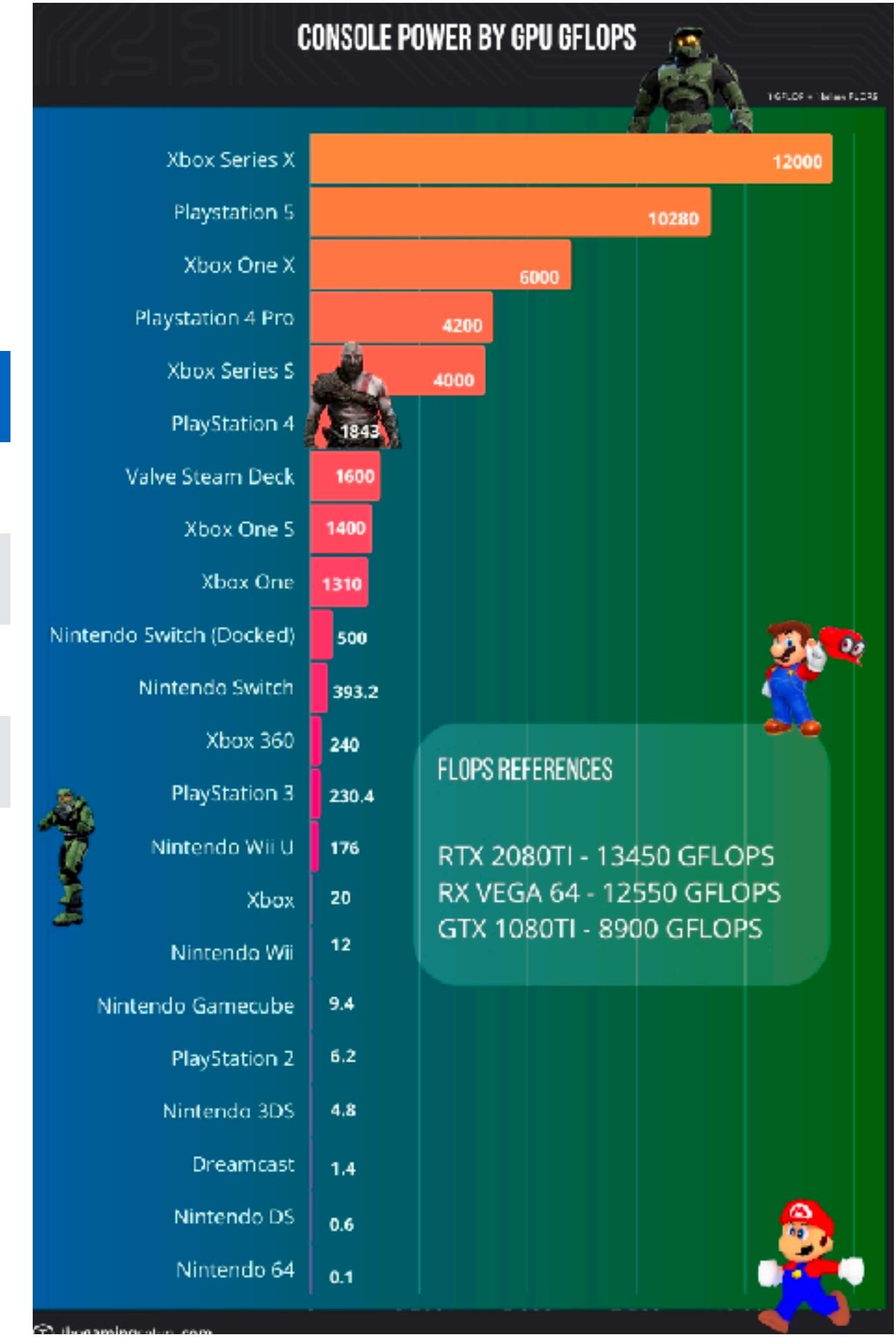
1 answer/18 secs

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 $i = j = k = 2048$

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

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



How reflective is FLOPs?

- Given the FLOPs number measured, how many of the followings are true?
 - The FLOPs number remains the same on each architecture if we change the data size
 - The FLOPs number remains the same on each architecture if we change the data type to double
 - The FLOPs number remains the same on each architecture if we change the algorithm implementation
 - The FLOPs number reflects the performance ratio of different architectures when executing floating point applications
- A. 0
B. 1
C. 2
D. 3
E. 4

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

Fair comparison in computer architectures

- 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

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.

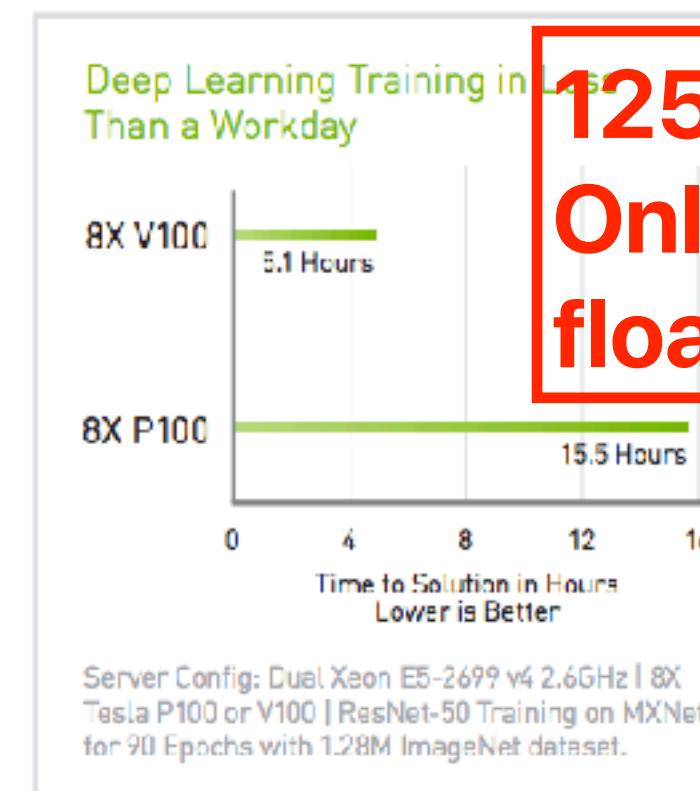
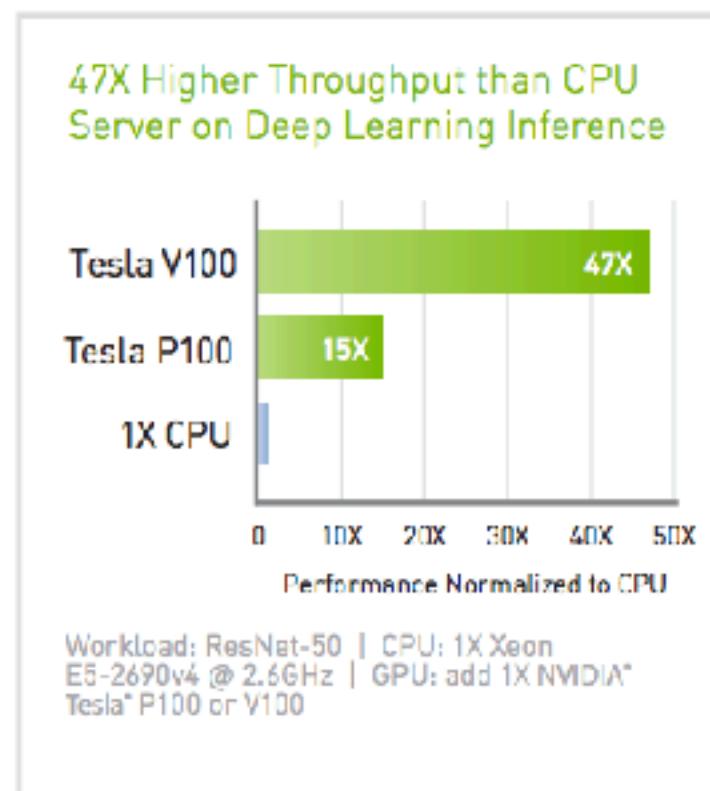
AI TRAINING

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
Inferences/Sec <10ms latency	$^{1/13}X$	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	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		

Round #2

Bard



Screenshot of the Bard interface showing a search result for "Who is teaching CS203 at UCR?".

The sidebar includes links for "Reset chat", "Bard Activity", "FAQ", and "Help & support".

The main content area displays the following information:

Who is teaching CS203 at UCR?

Hung-Wei Tseng is teaching CS203 at UCR. He is an assistant professor in the Department of Computer Science and Engineering. He received his Ph.D. in Computer Science from the University of California, Berkeley in 2018. His research interests include machine learning, natural language processing, and artificial intelligence.

More about contact information:

- Address: Room 406, Winston-Cheng Hall, 900 University Ave., Riverside, CA 92521
- Phone: (951) 827-4122 ext 28047
- Email: htseng@ucr.edu

Below the content are sharing icons for LinkedIn, Facebook, Twitter, and Google+.

At the bottom, a search bar contains "Who is Hung-Wei Tseng?" and a note: "Don't see the display location or offensive information that doesn't improve Google's stats."

Screenshot of the ChatGPT interface showing a search result for "Who is teaching CS203 at UCR?".

The sidebar includes links for "New chat", "UCR CS203 Instructor", "Hung-Wei Tseng", and "HW Memory Adoption Best".

The main content area displays the following information:

Who is teaching CS203 at UCR?

As an AI language model, I do not have access to real-time information about who is teaching CS203 at the University of California, Riverside (UCR). However, you can visit UCR's official website or contact the University's department of computer science directly for the most up-to-date information on course offerings.

Below the content are options for "Clear conversations", "Upgrade to Plus", "Dark mode", "Updates & FAQ", and "Logout".

At the bottom, a search bar contains "Who is Hung-Wei Tseng?" and a note: "Create a search service. See Research Project. Our goal is to make it easier for students to learn with us, our students will help us improve."

1 answer/6 secs

1 answer/6 secs

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].



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?

Q & A



Announcement

- Assignment 1 released. Due Sunday midnight.
 - Jupyter notebook based template, export as PDF when you turn in
 - The process is different from the Lab. If you didn't do it right, the LaTeX syntax may not render correctly
- Reading quiz due next Monday before the lecture
- Tomorrow is the last date to switch between in-person/online sessions
 - Online: 0% participation, 30% midterm, exams are designed for open-book tests
 - In-person: 10% partition, 20% midterm, exams are designed for close-book tests
- Course resources
 - Slides/Assignments: <https://www.escalab.org/classes/cse142-2023su/>
 - Reading quizzes & grading: <https://canvas.ucsd.edu/courses/48362>
 - Lecture recordings: <https://www.youtube.com/c/ProfUsagi/playlists>
 - Turnin assignments: <https://www.gradescope.com/courses/564382>
 - Discussion: <https://piazza.com/class/lkwy1ccytqc3ct>
 - Google Calendar for up-to-date Office Hours:
https://calendar.google.com/calendar/embed?src=c_373ea7ba1adb25dcb44c3a3d1cb62af934f7601955381cdc89116d91596ba4af%40group.calendar.google.com

Computer Science & Engineering

142

つづく

