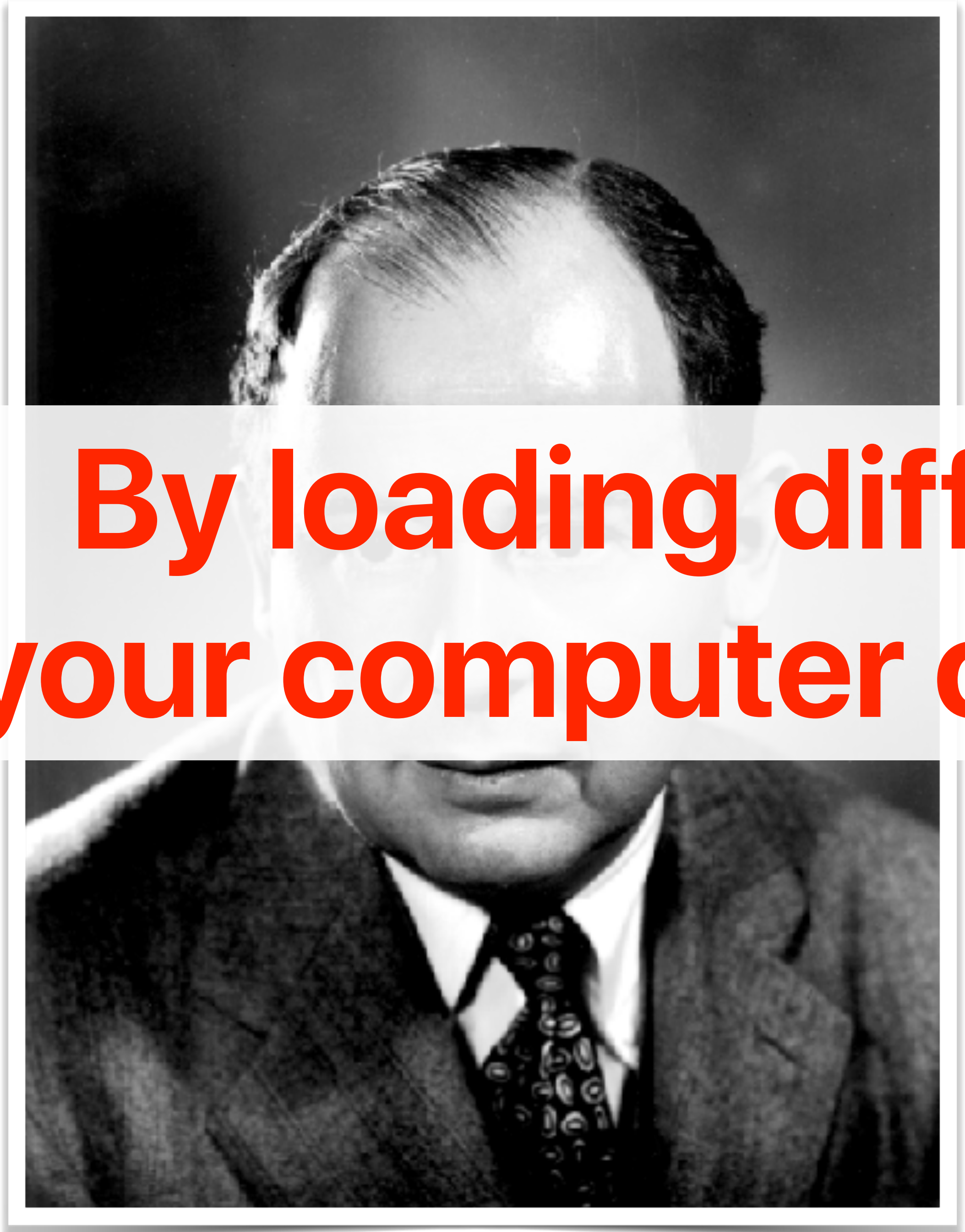
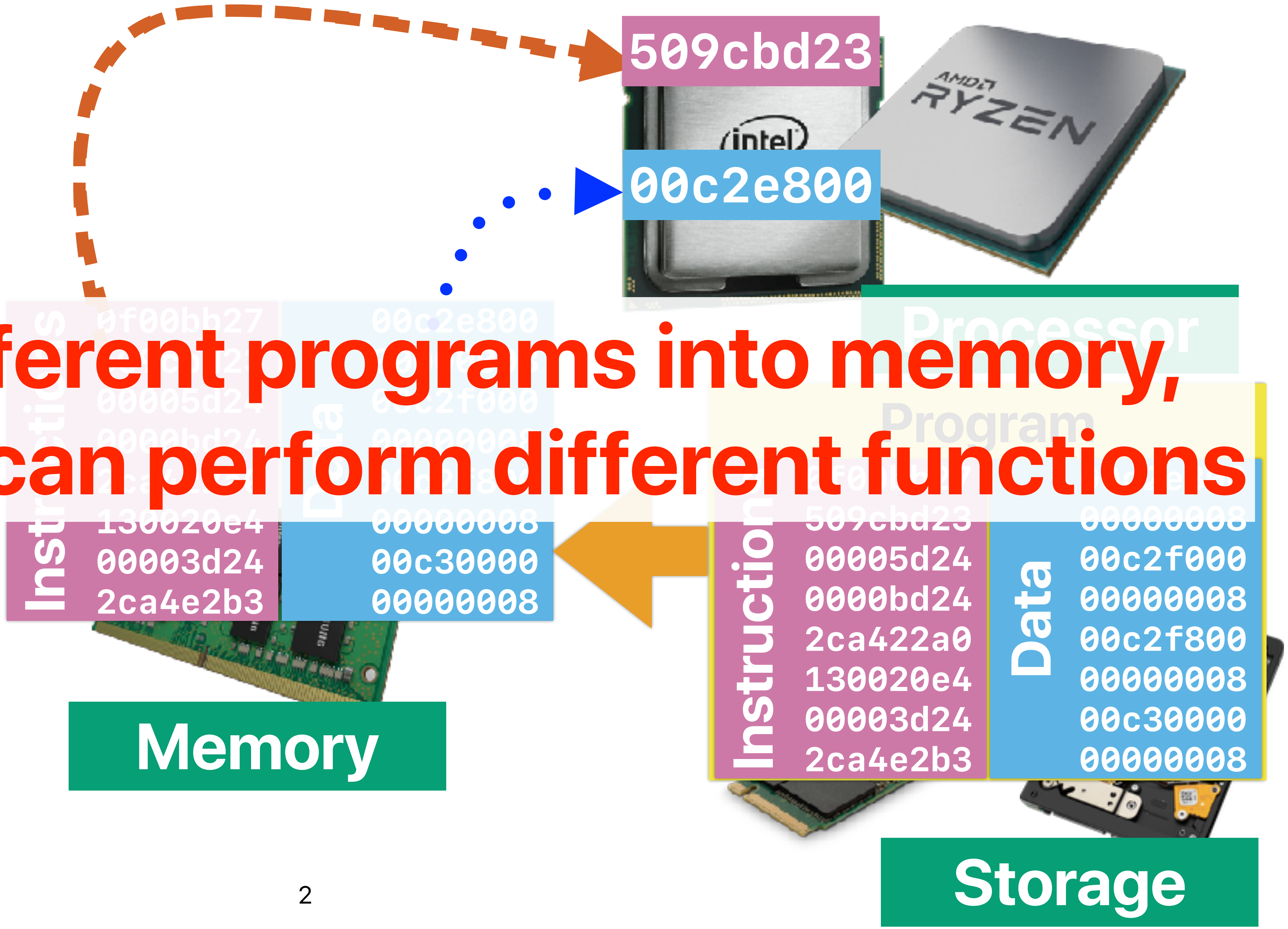
Performance (3): One thing right

Hung-Wei Tseng

Recap: von Neumman Architecture





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

Recap: Summary of CPU Performance Equation

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

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

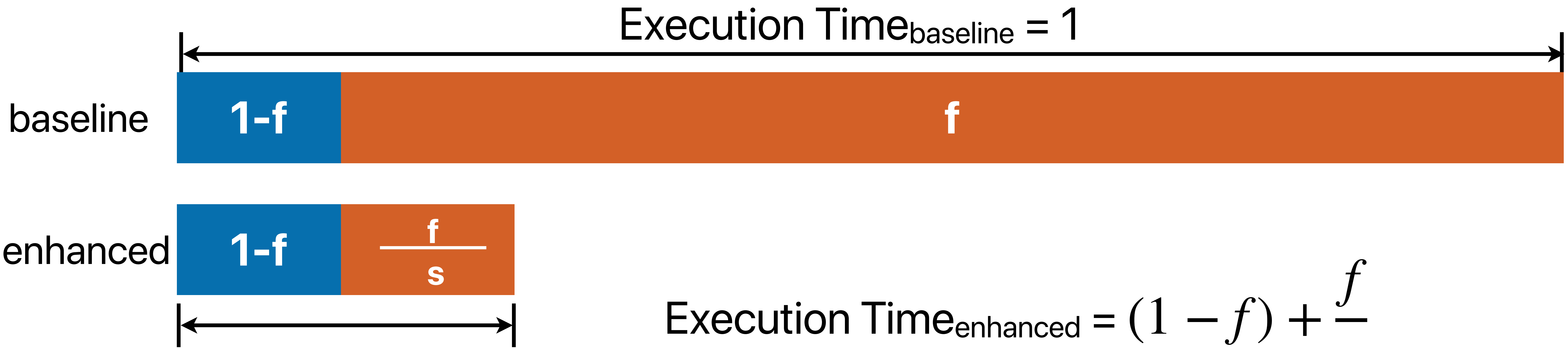
$$ET = IC \times CPI \times CT$$

$$Speedup = \frac{Execution\ Time_X}{Execution\ Time_Y}$$

- 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

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



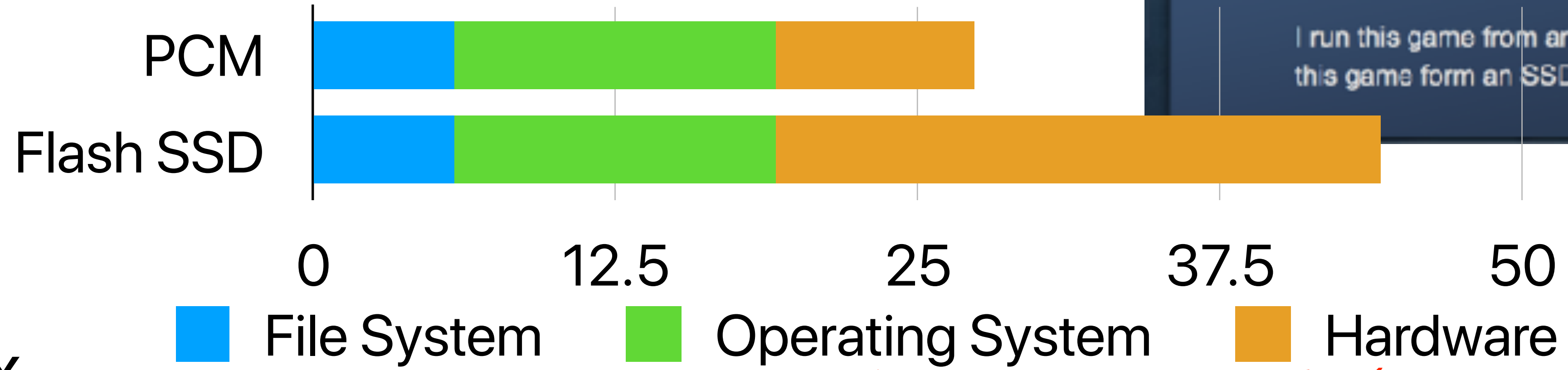
Execution Time$_{enhanced}$ = $(1-f) + \dfrac{f}{s}$

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

Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessi the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faste the new technology needs to be?

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



$$Speedup_{enhanced}(16\%, x) = \frac{1}{(1-16\%) + \frac{16\%}{x}} = 2$$

$$x = 0.47$$

Does this make sense?

Lessons learned from Amdahl's Law

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

- Corollary #1: Maximum speedup $Speedup_{max}(f, \infty) = \frac{1}{(1-f)}$

Outline

- Amdahl’s Law and it’s implications (cont.)

Amdahl’s Law — and It’s Implication in the Multicore Era (cont.)

H&P Chapter 1.9

M. D. Hill and M. R. Marty. Amdahl’s Law in the Multicore Era. In Computer, vol. 41, no. 7, pp. 33-38, July 2008.

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:

f_{Opt1}	f_{Opt2}	$1-f_{Opt1}-f_{Opt2}$

$$Speedup_{enhanced}(f_{Opt1}, f_{Opt2}, s_{Opt1}, s_{Opt2}) = \frac{1}{(1 - f_{Opt1} - f_{Opt2}) + \frac{f\_Opt1}{s\_Opt1} + \frac{f\_Opt2}{s\_Opt2}}$$

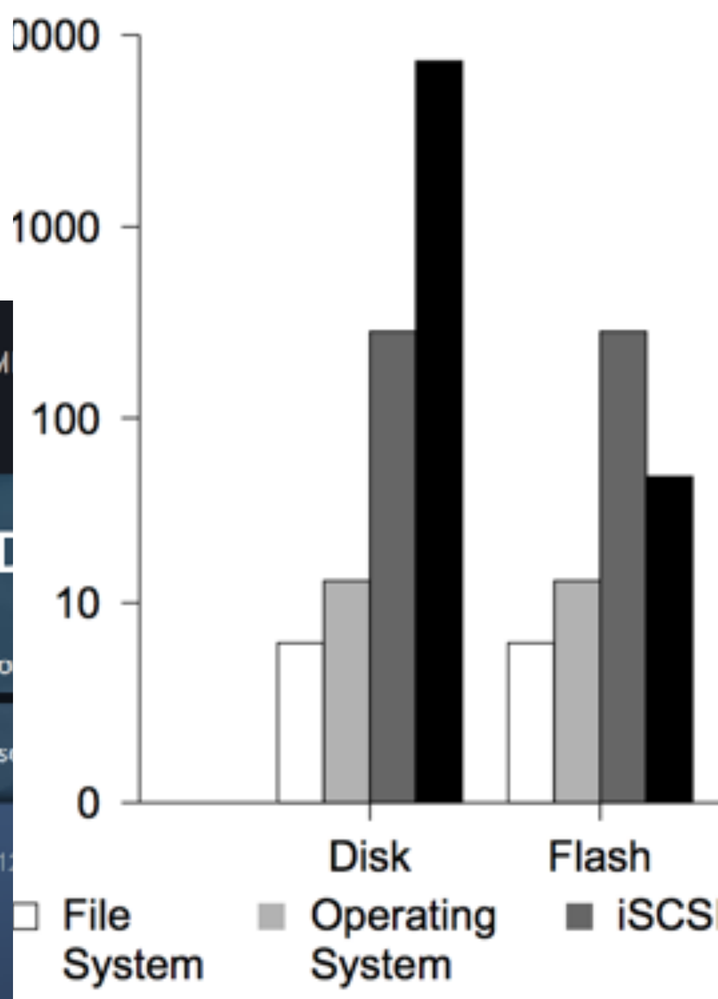
  - If optimization #1 and optimization #2 are not dis-joint:

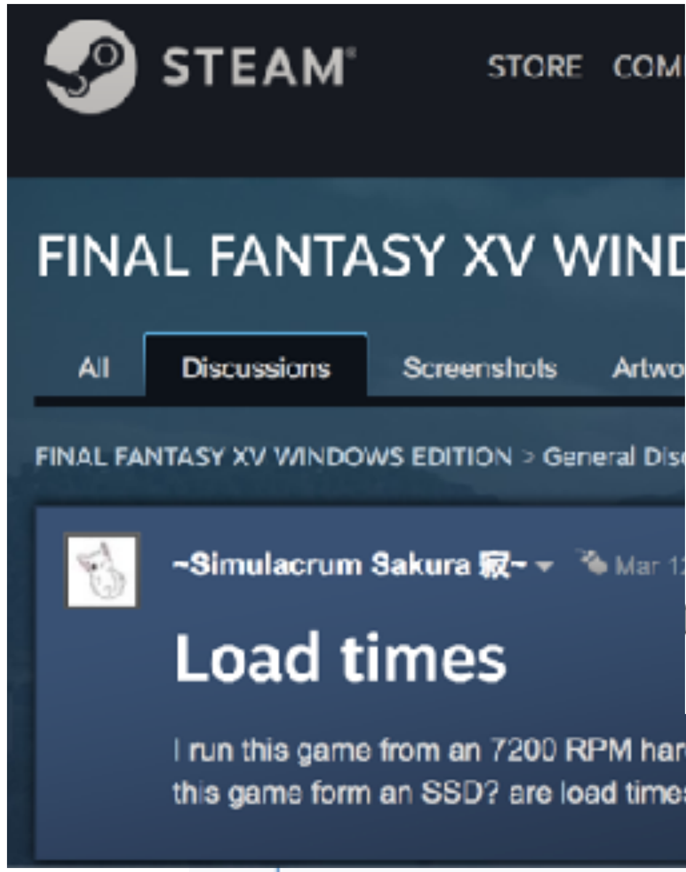
$f_{OnlyOpt1}$	$f_{OnlyOpt2}$	$f_{BothOpt1Opt2}$	$1-f_{OnlyOpt1}-f_{OnlyOpt2}-f_{BothOpt1Opt2}$

$$Speedup_{enhanced}(f_{OnlyOpt1}, f_{OnlyOpt2}, f_{BothOpt1Opt2}, s_{OnlyOpt1}, s_{OnlyOpt2}, s_{BothOpt1Opt2})$$
$$= \frac{1}{(1 - f_{OnlyOpt1} - f_{OnlyOpt2} - f_{BothOpt1Opt2}) + + \frac{f\_BothOpt1Opt2}{s\_BothOpt1Opt2} + \frac{f\_OnlyOpt1}{s\_OnlyOpt1} + \frac{f\_OnlyOpt2}{s\_OnlyOpt2}}$$

Practicing Amdahl's Law (2)

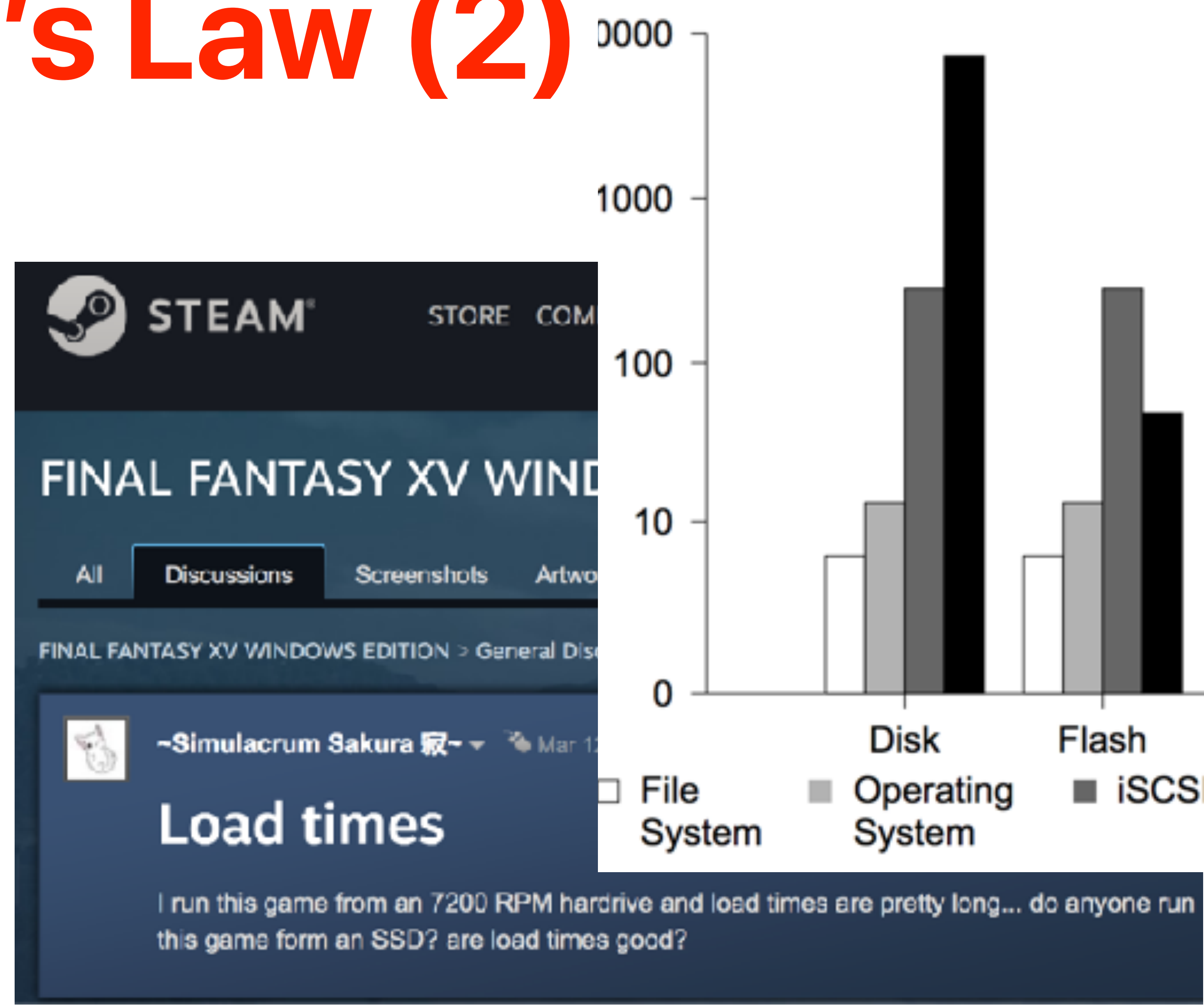
- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?
  A. ~7x
  B. ~10x
  C. ~17x
  D. ~29x
  E. ~100x





Practicing Amdahl's Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?
    A. ~7x
    B. ~10x
    C. ~17x
    D. $\boxed{\text{~29x}}$
    E. ~100x



$$Speedup_{enhanced}(95\%, 5\%, 100, 2) = \frac{1}{(1-95\%-5\%)+\frac{95\%}{100}+\frac{5\%}{2}} = 28.98\times$$

Corollary #1 on Multiple Optimizations

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

f_1	f_2	f_3	f_4	$1-f_1-f_2-f_3-f_4$

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

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

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

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

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

Corollary #2.1 Don't hurt non-common part too mach

- 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

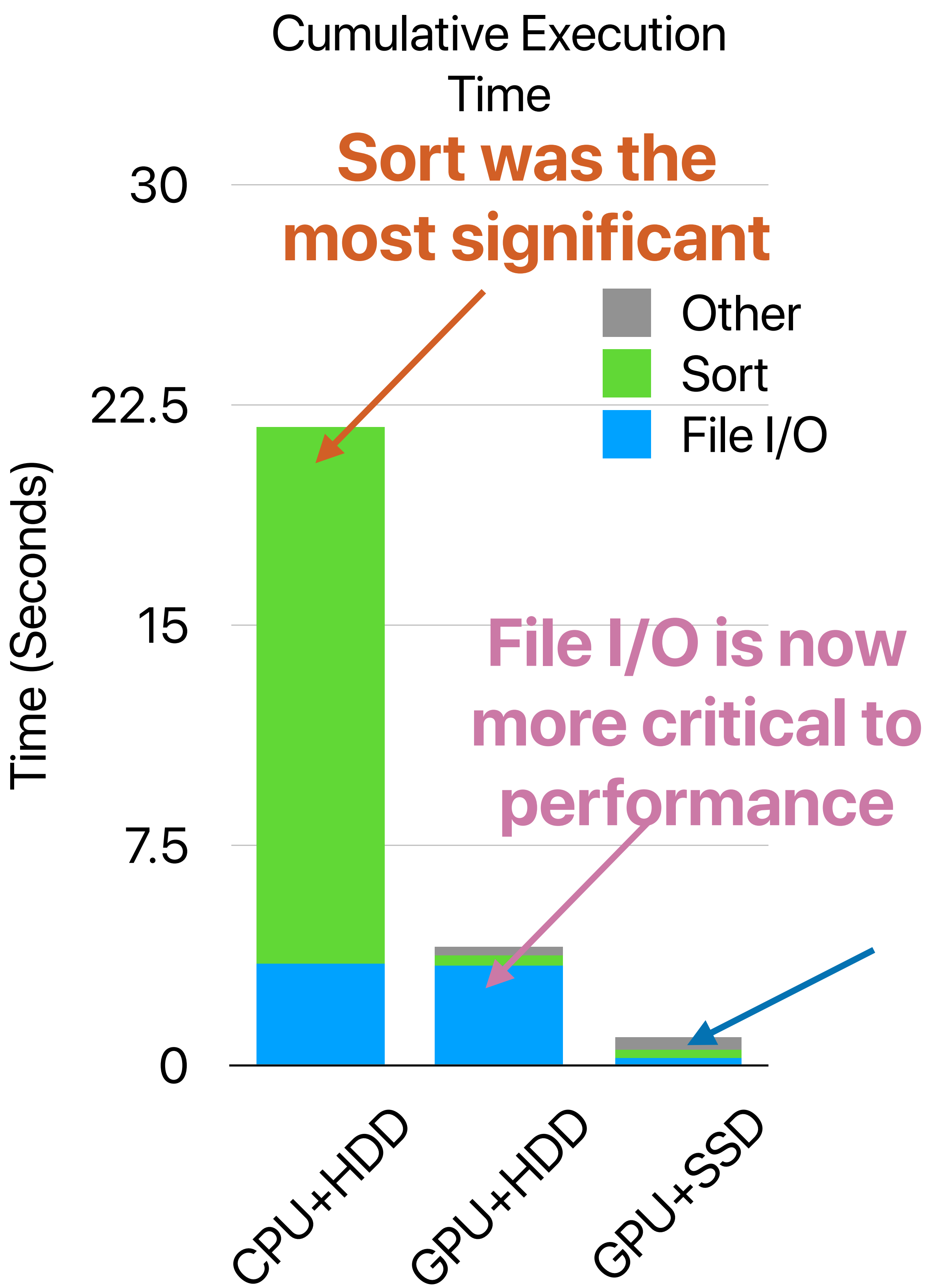
$$ET_{new} = \frac{ET_{old} \times 90\%}{9} + ET_{old} \times 10\% \times 10$$

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

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

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

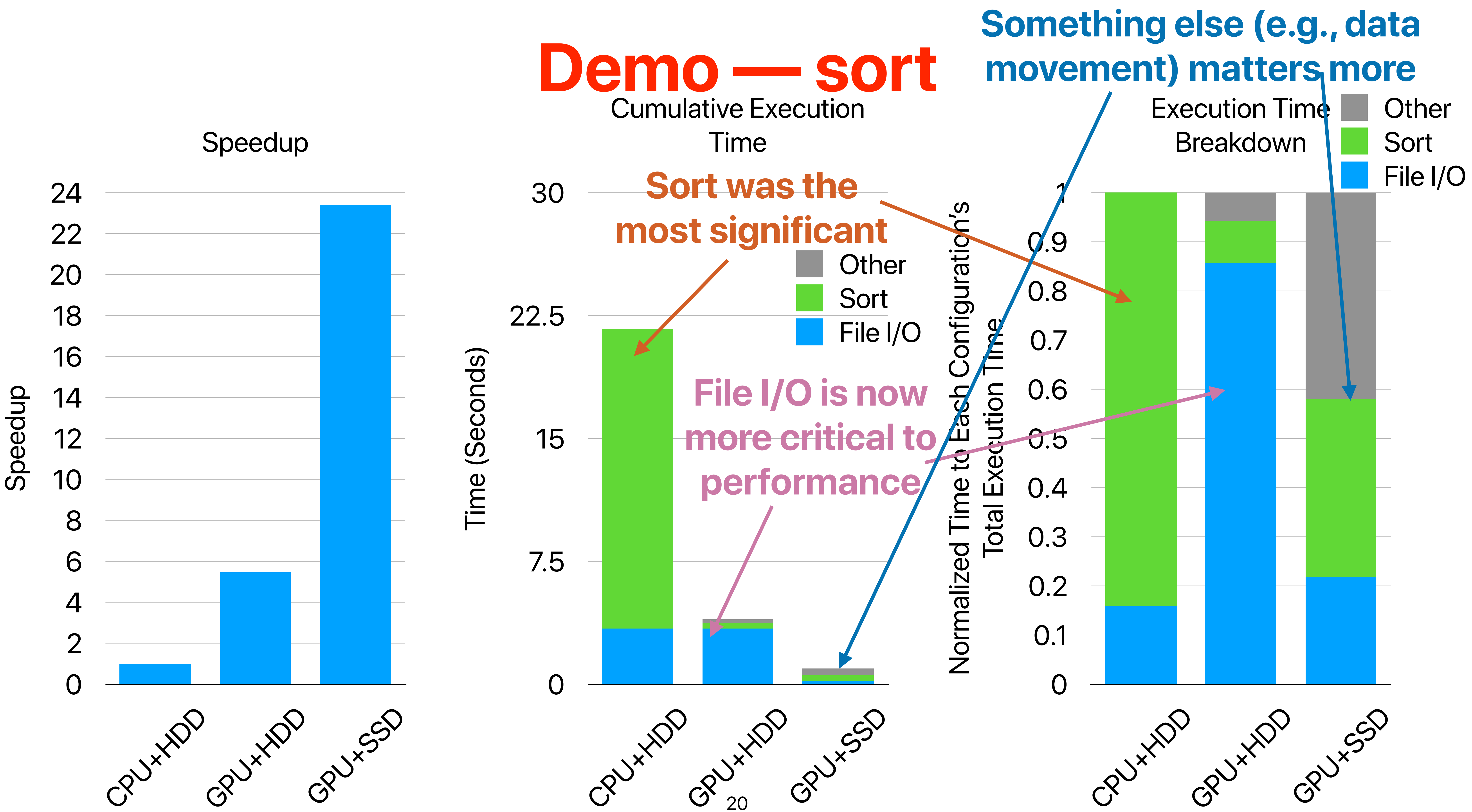
Corollary #3 — optimization has a moving target



- 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

Demo — sort



Something else (e.g., data movement) matters more
Speedup
Speedup
24
22
20
18
16
14
12
10
8
6
4
2
0
CPU+HDD
GPU+HDD
GPU+SSD
Cumulative Execution Time
Time (Seconds)
30
22.5
15
7.5
0
Sort was the most significant
Other
Sort
File I/O
File I/O is now more critical to performance
Execution Time Breakdown
Normalized Time to Each Configuration's Total Execution Time
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Amdahl's Law on Multicore Architectures

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

$$Speedup_{parallel}(f_{parallelizable}, n) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable}{n}}$$

https://www.pollev.com/hungweitseng close in 1:30

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  ① If we have unlimited parallelism, the performance of each parallel piece 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
  ③ 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

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct? $Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable \times Speedup(<1)}{\infty}}$
  ① If we have unlimited parallelism, the performance of each parallel piece 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 $Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1 - f_{parallelizable})}$ **speedup is determined by 1-f**
  ③ 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

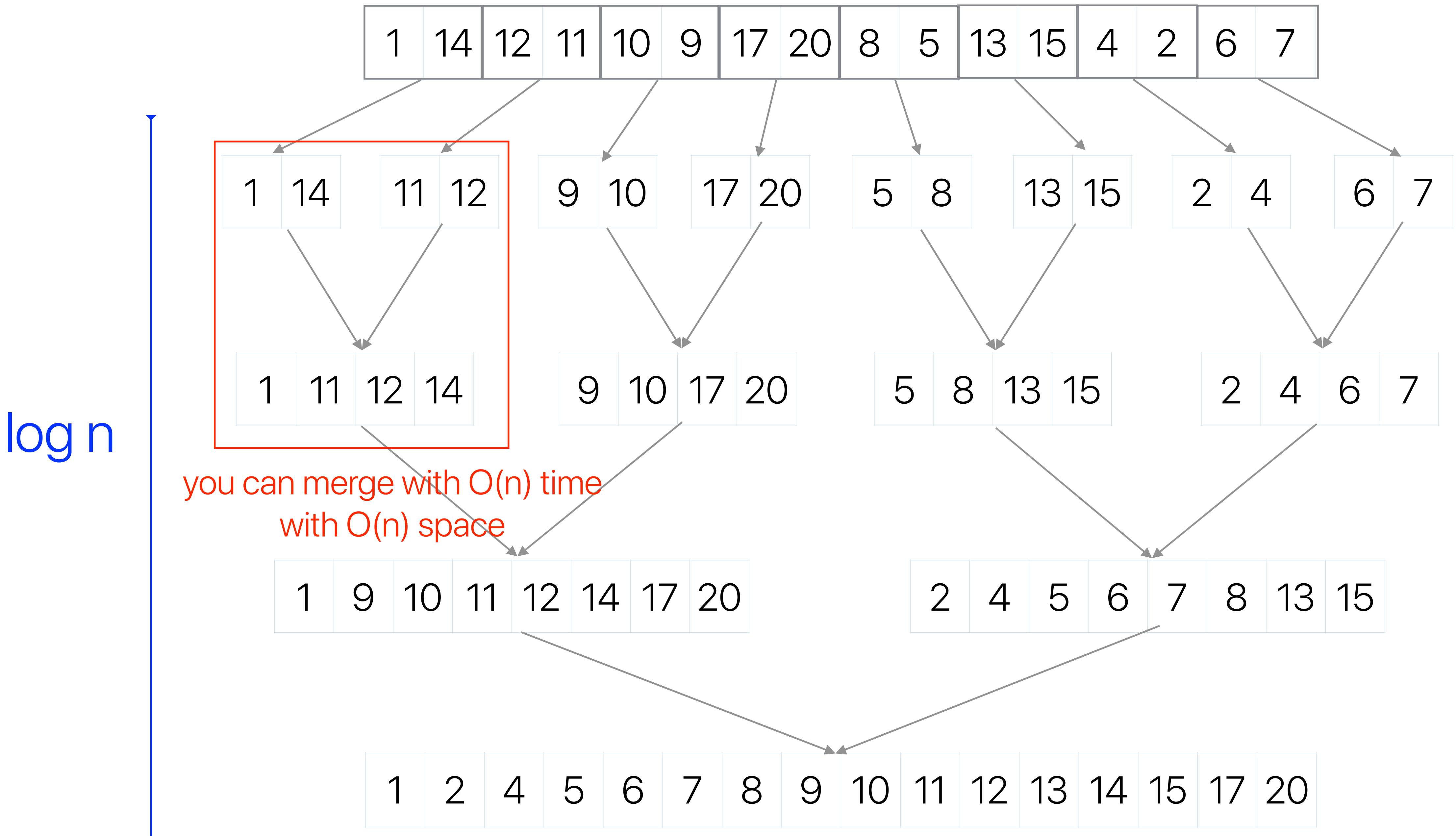
$O(nlog_2n)$

Bitonic Sort

$O(nlog_2^2n)$

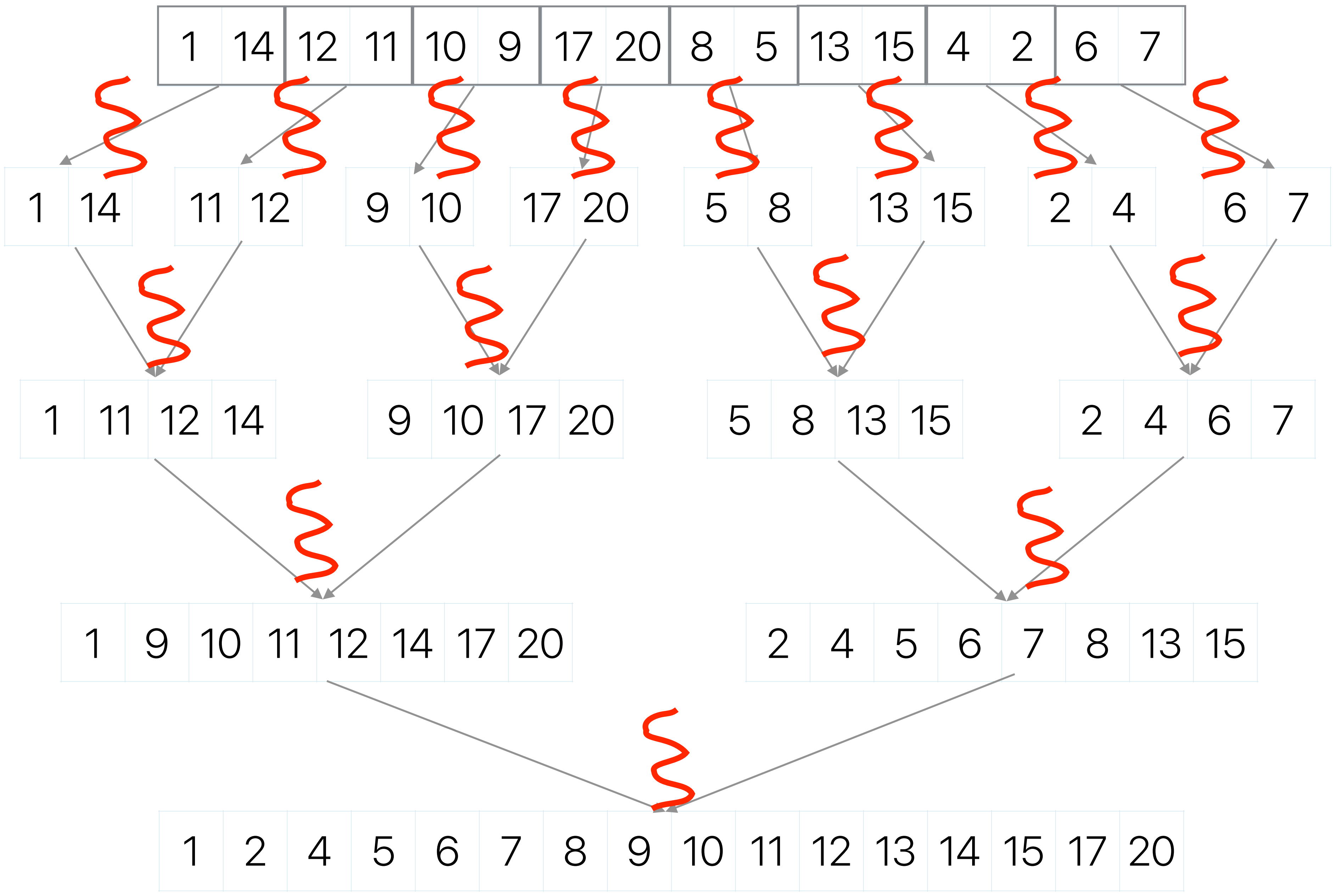
```
void BitonicSort() {

    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

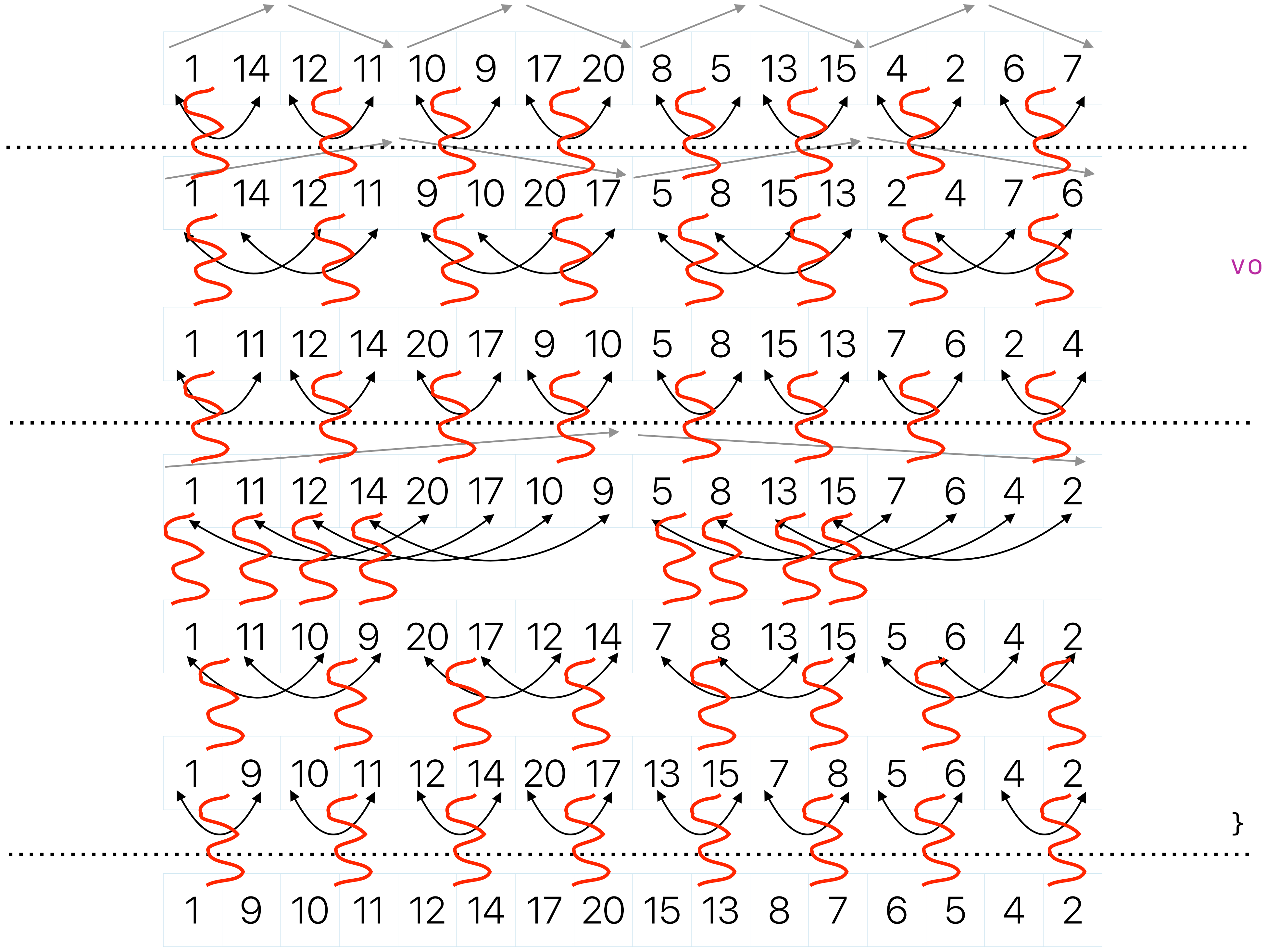


O(n log n)

Parallel merge sort

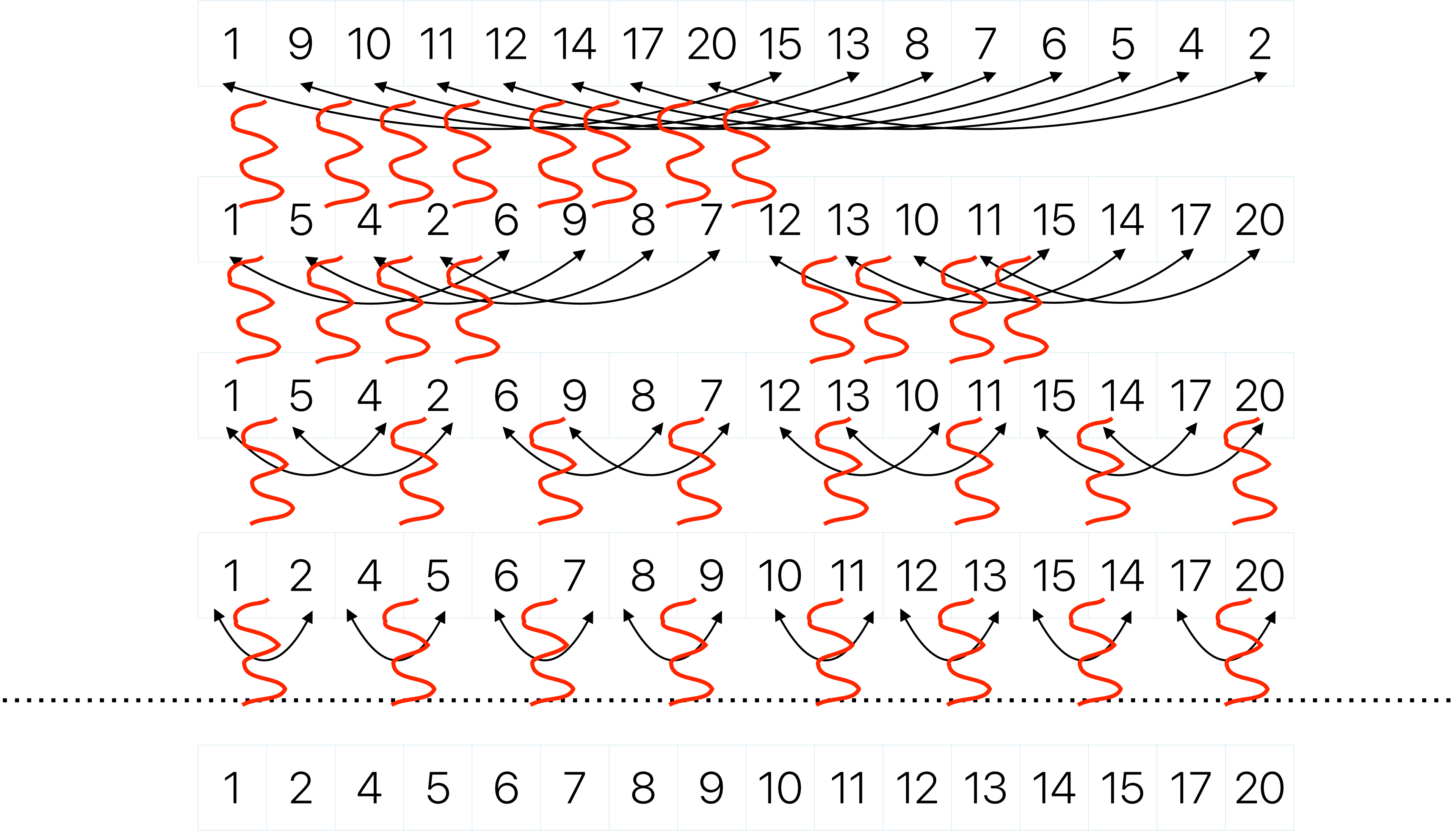


Bitonic sort




```
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 log² n) — hard to beat n(log n) if you can't parallelize this a lot!

Corollary #4

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

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

- 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

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

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

- 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

$$\text{Execution Time}_{enhanced} = \frac{(1-f)}{perf(r)} + \frac{f}{n}$$

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

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

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

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

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

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

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

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

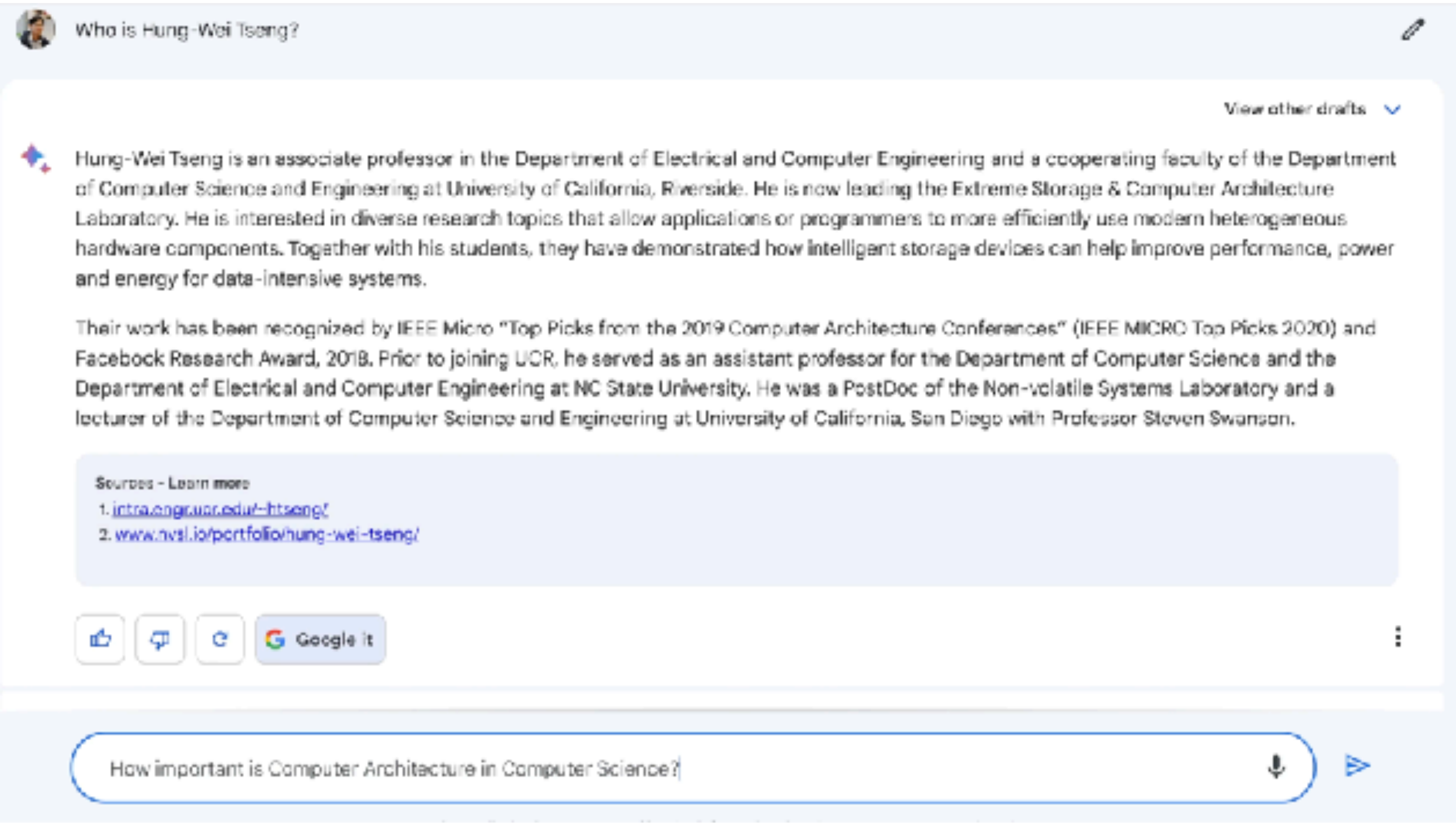
Choose the right metric — Latency v.s. Throughput/Bandwidth

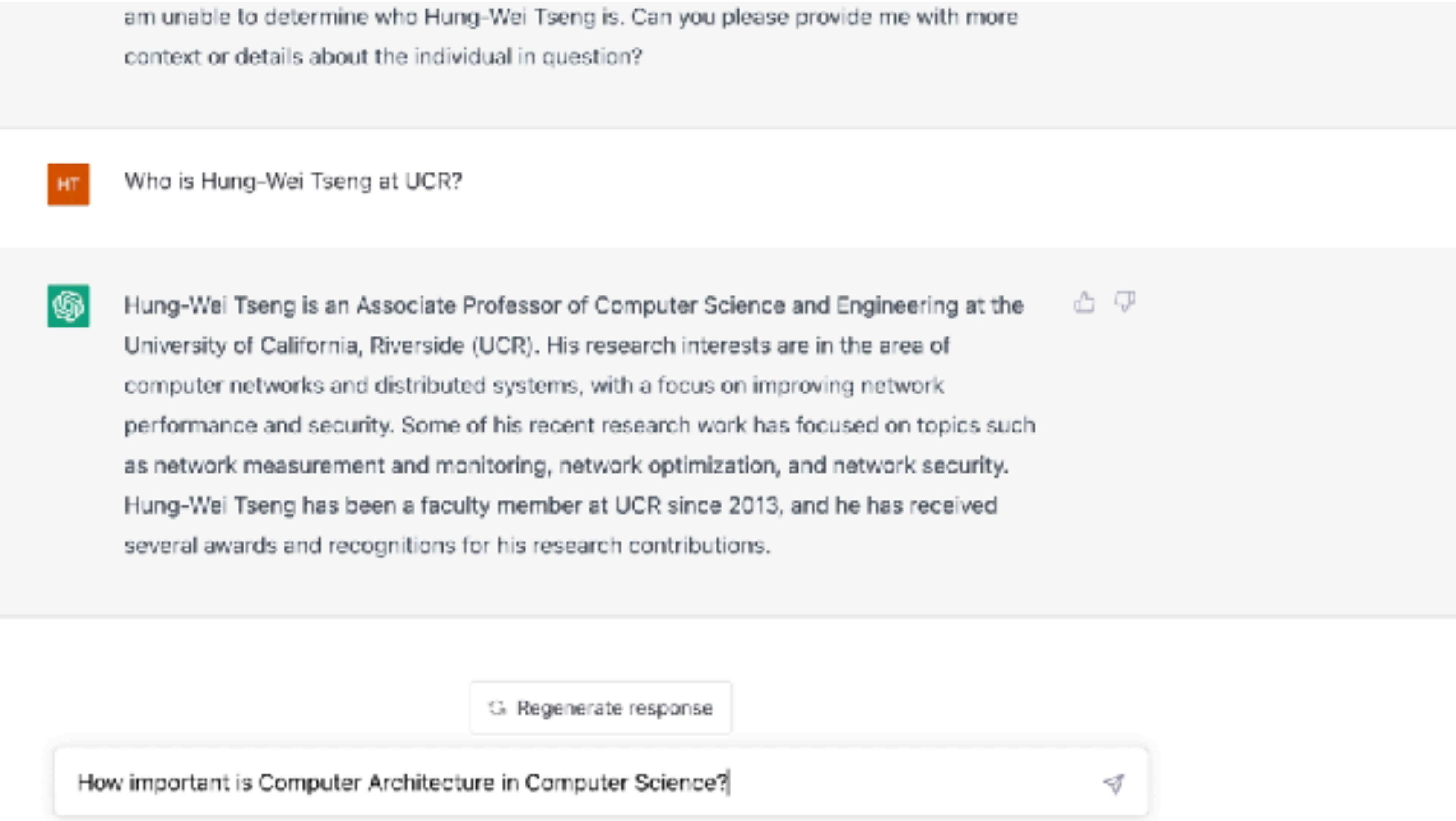
Latency v.s. Bandwidth/Throughput

- 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)

Round #1

Bard



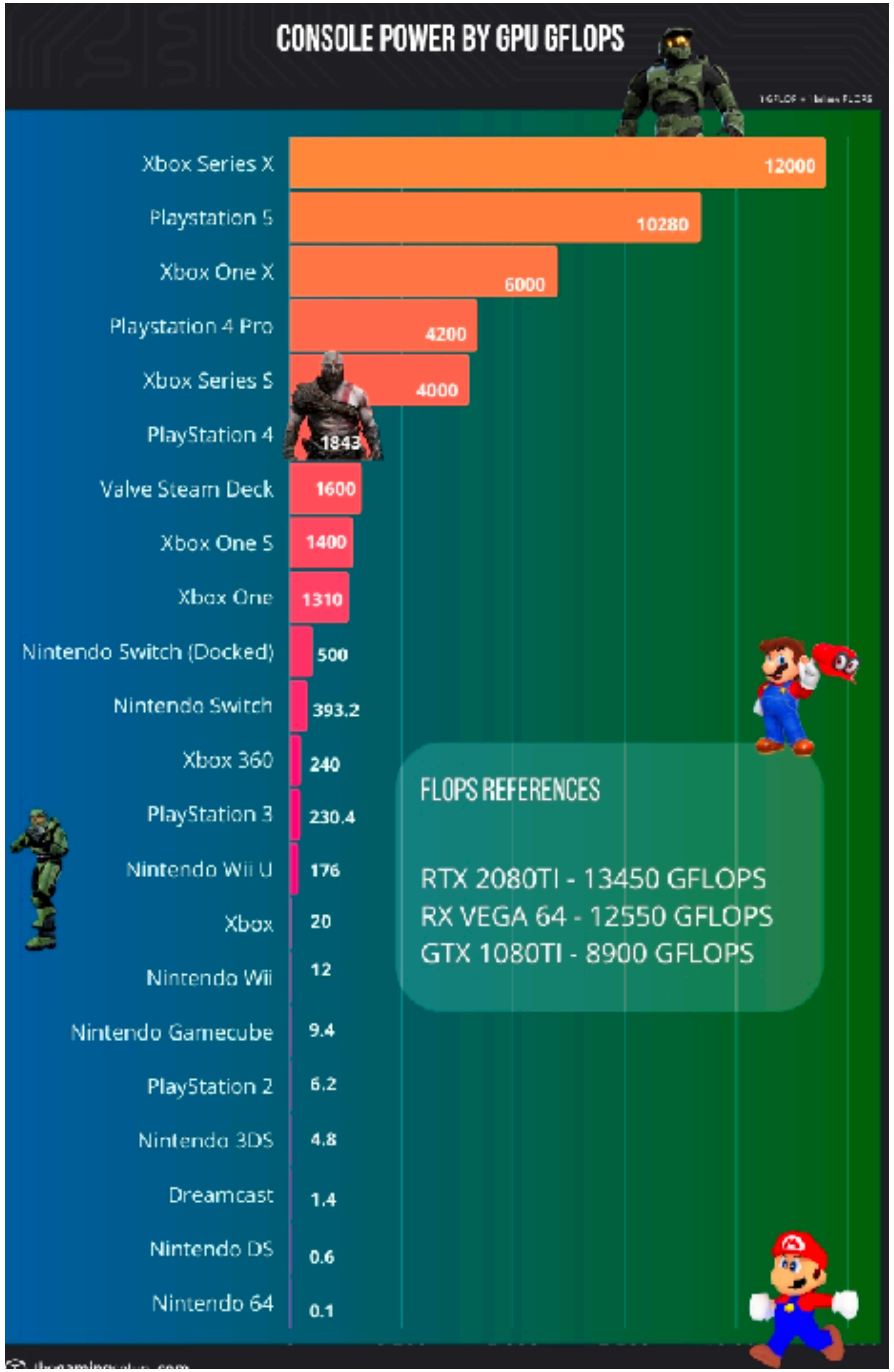


1 answer/6 secs

1 answer/18 secs

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 $i = j = k = 2048$

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

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

Announcement

- Reading quiz due next Tuesday BEFORE the lecture
    - We will drop two of your least performing reading quizzes
    - You have two shots, both unlimited time
- Assignment #1 released
    - We typically give you two weeks to work on an assignment
    - We never allow late submission and we will never have deadline extension
    - Due on 4/20
- Assignment #0 due on tonight
- Check our website for slides, eLearn for quizzes, piazza for discussions
- Youtube channel for lecture recordings: https://www.youtube.com/c/ProfUsagi/playlists

Computer
Science &
Engineering

203

つづく