

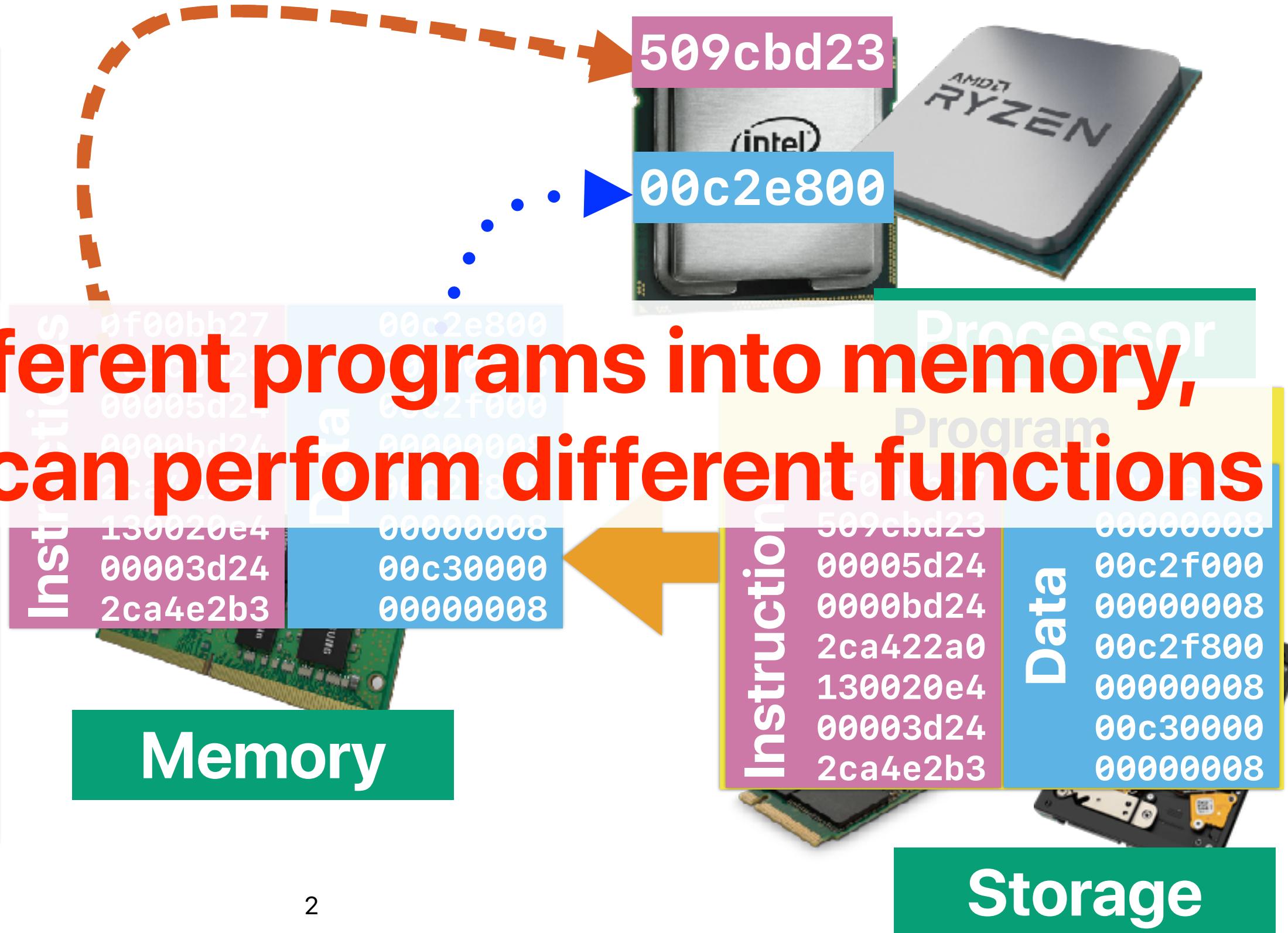
Performance

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): $O(n)$

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

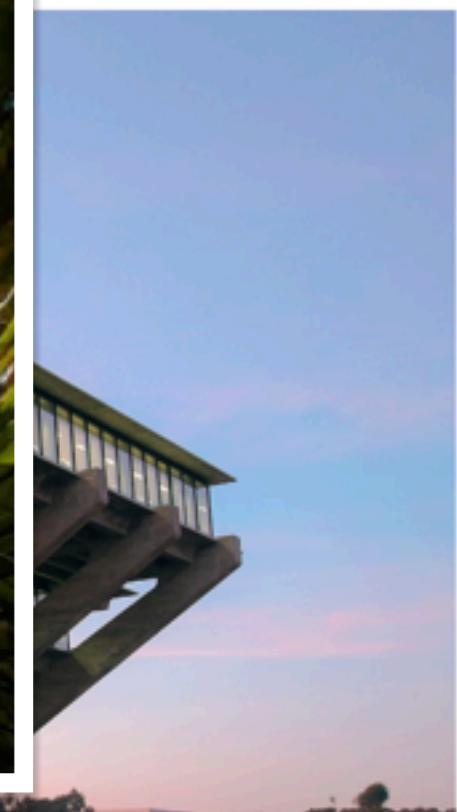
Ranked in 2022, part of Best Science Schools

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



Outline

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

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

Peer instruction

- Before the lecture — You need to complete the required **reading**
- During the lecture — I'll bring in activities to ENGAGE you in exploring your understanding of the material
 - Popup questions
 - Individual **thinking** — use polls in Zoom to express your opinion
 - Group **discussion**
 - Breakout rooms based on your residential colleges!
 - Use polls in Zoom to express your group's opinion
 - Whole-classroom **discussion** — we would like to hear from you

Read

Think

Discuss

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 matters?

- Execution time is the most essential performance metric

**“Execution time” is the most
important metric**

CPU Performance Equation

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

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

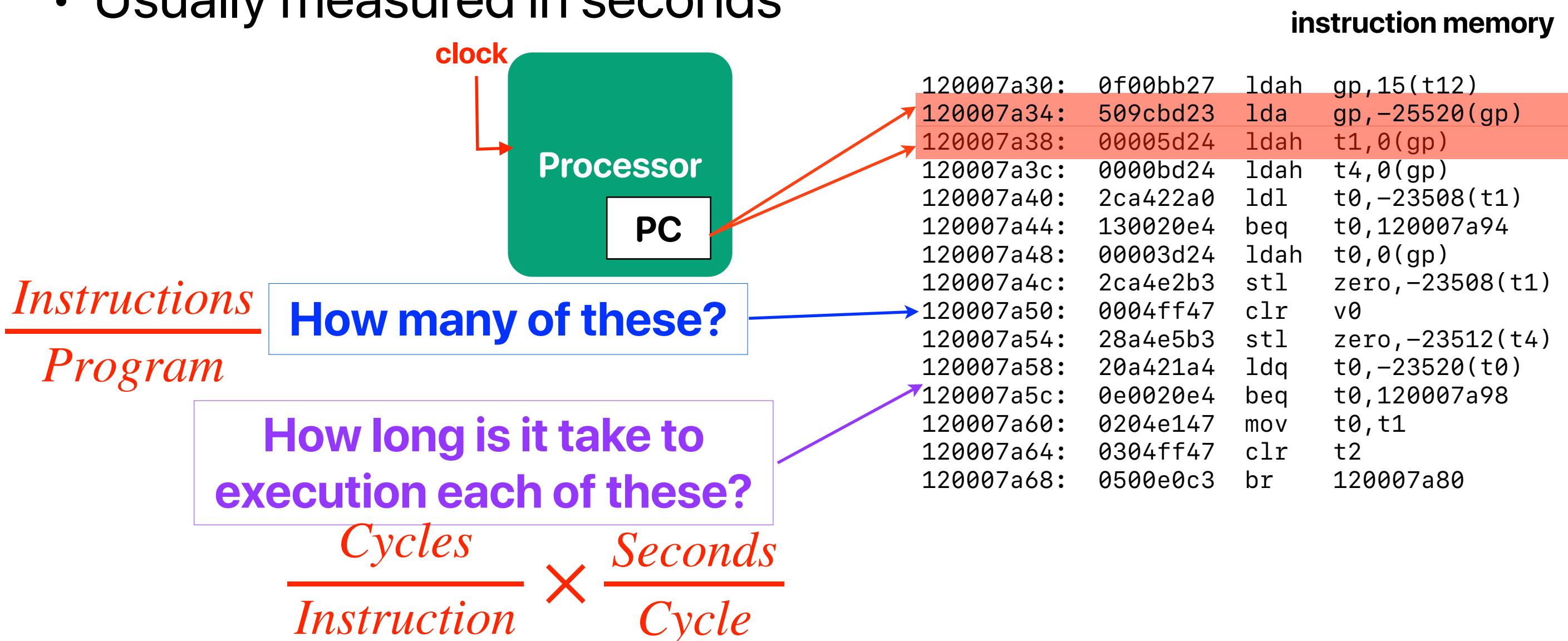
$$ET = IC \times CPI \times CT$$

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

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

Execution Time

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



Evaluating the CPU program “execution time”

- Assume that we have an application composed with a total of **5,000,000,000** instructions, in which **20%** of them are “Type-A” instructions with an average **CPI of 4** cycles, **20%** of them are “Type-B” instructions with an average **CPI of 3** cycles and **the rest** instructions are “Type-C” instructions with average **CPI of 1** cycle. If the processor runs at **4 GHz**, how long is the execution time?

A. 1.25 sec

B. 2.5 sec

C. 3.75 sec

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

D. 7.5 sec

$$ET = (5 \times 10^9) \times (20\% \times 4 + 20\% \times 3 + 60\% \times 1) \times \frac{1}{4 \times 10^9} \text{ sec} = 2.5 \text{ sec}$$

E. 40 sec

average CPI

Takeaways: What matters?

- Execution time is the most essential performance metric
- The following three factors define the execution time of a program
 - Instruction count
 - Cycles per instruction
 - Cycle 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

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  
$
```

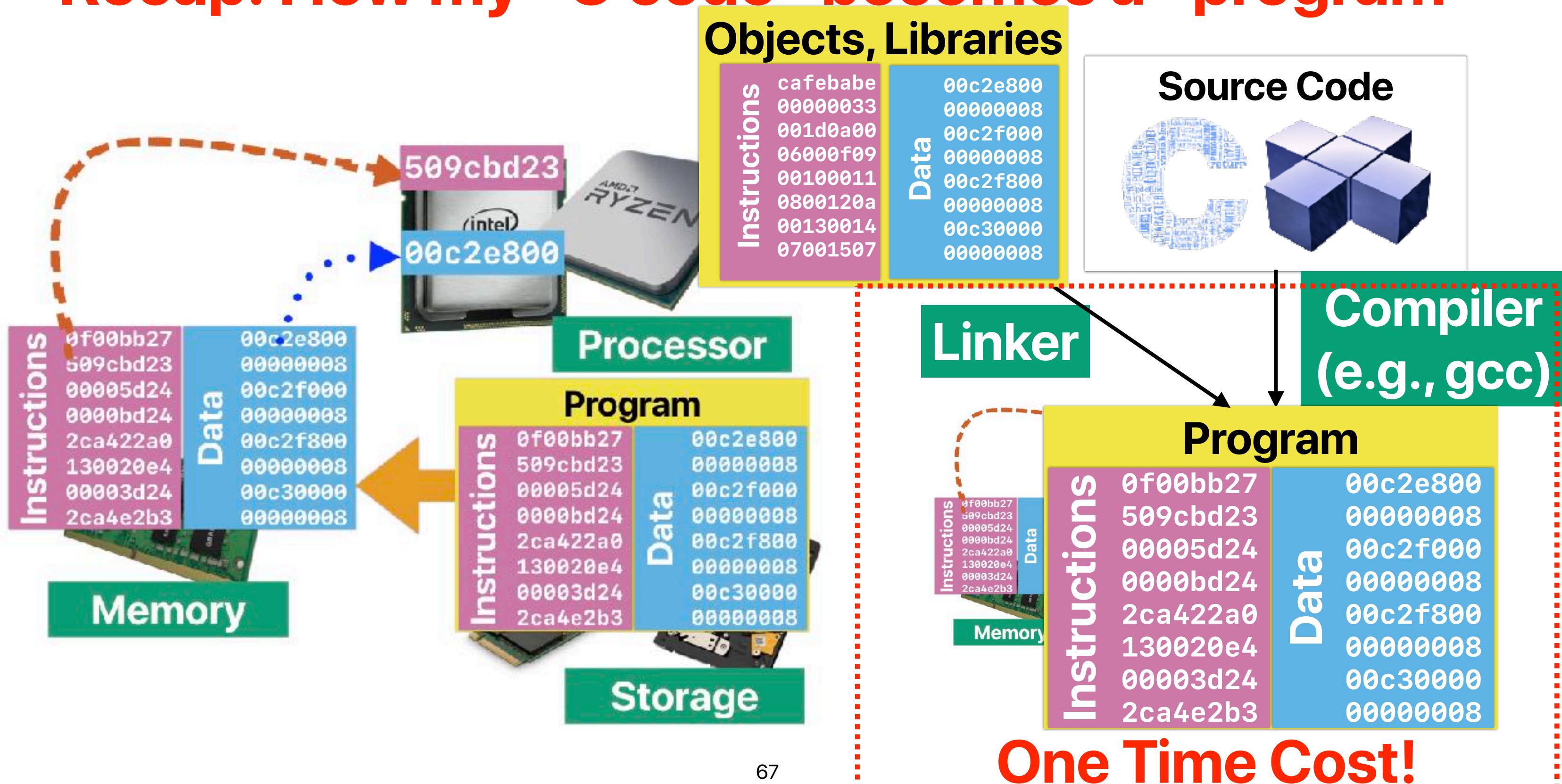
5.) How do I return to the default - i.e. allow machine to vary from highest to lowest?

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

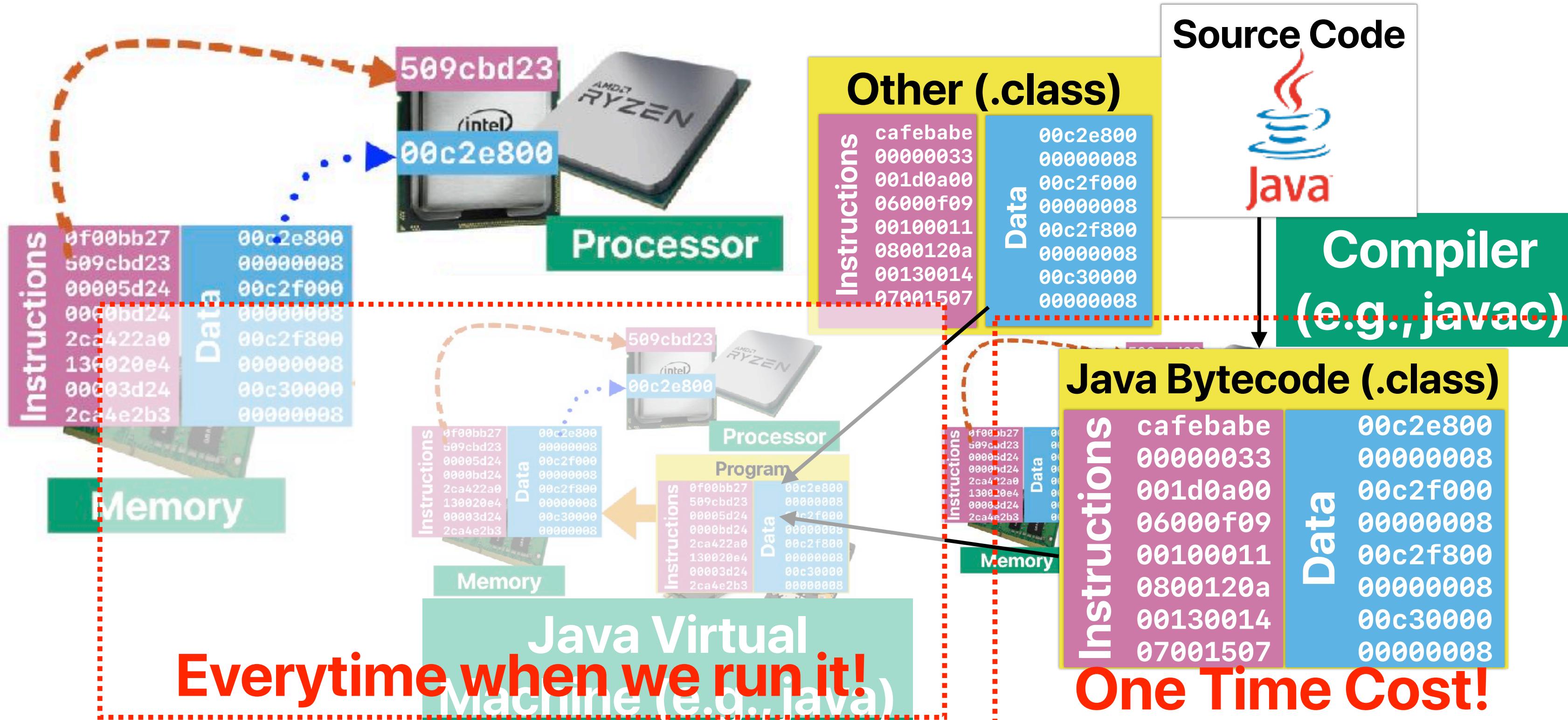
```
$ newSpeedLcw=$newSpeedTop # make them the same in this example
```

To read

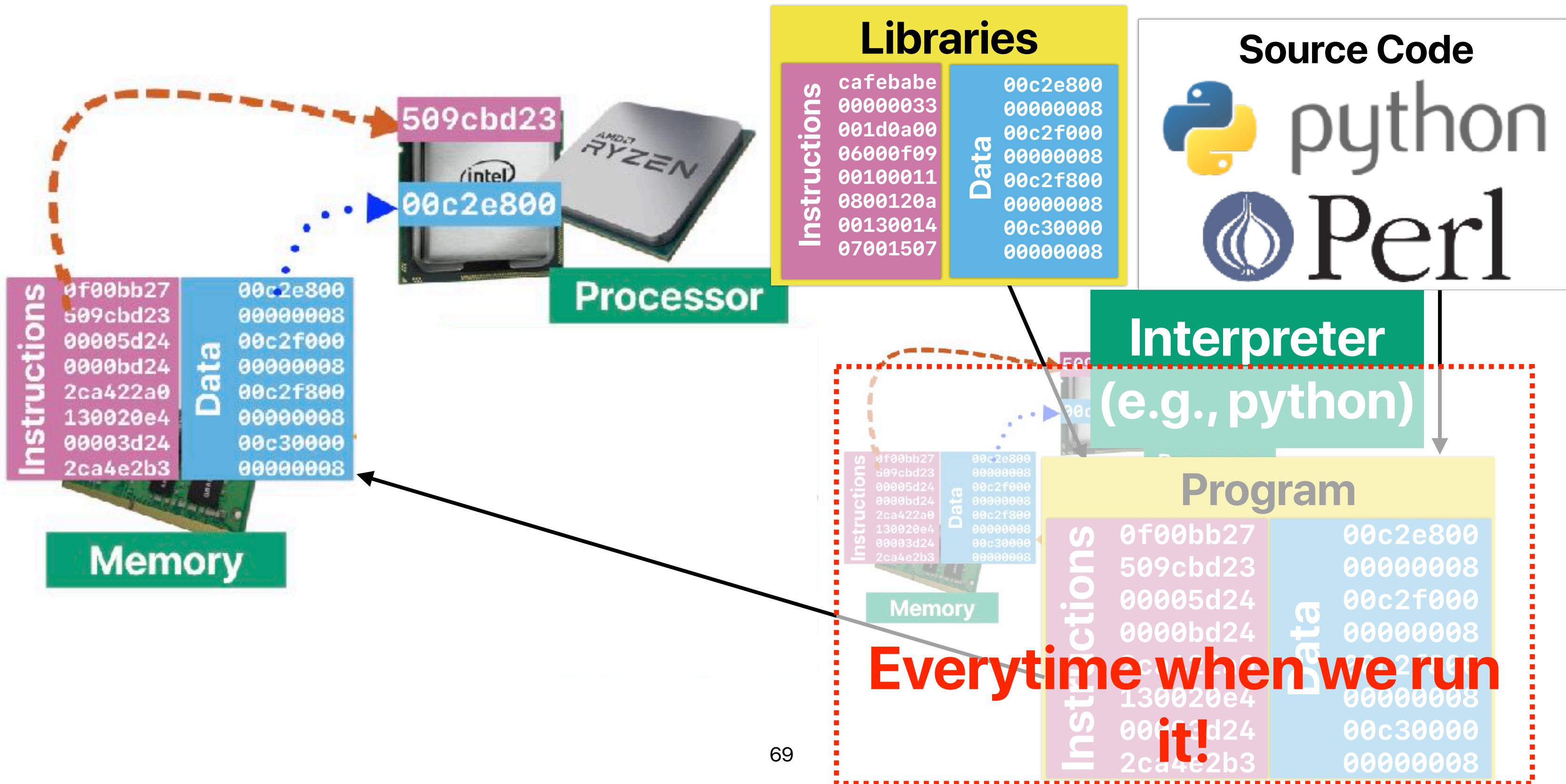
Recap: How my “C code” becomes a “program”



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



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



How programming languages affect performance

- Performance equation consists of the following three factors

① IC

② CPI

③ CT



Programmer uses programming languages to create library/programs that changes the CT, not the programming language itself makes the change

How many can the **programming language** affect?

A. 0

B. 1

C. 2

D. 3



How compilers affect performance

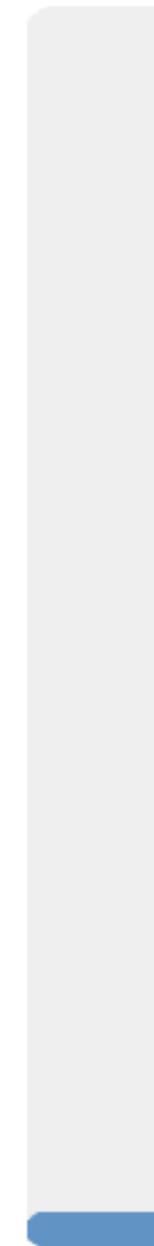
- Performance equation consists of the following three factors
 - ① IC
 - ② CPI
 - ③ CT

How many can the **compiler** affect?

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

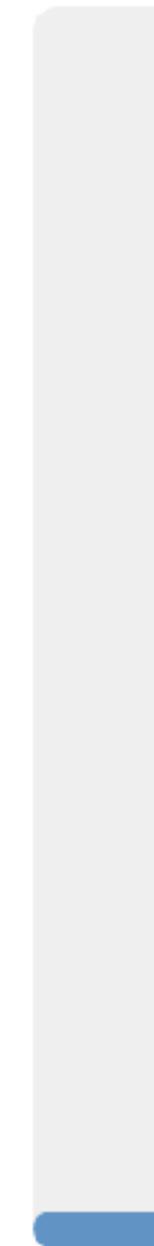
 0

0



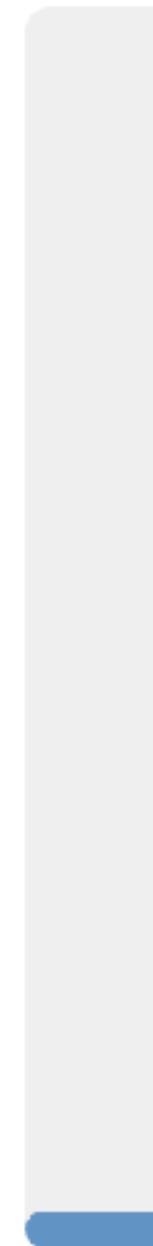
A

0



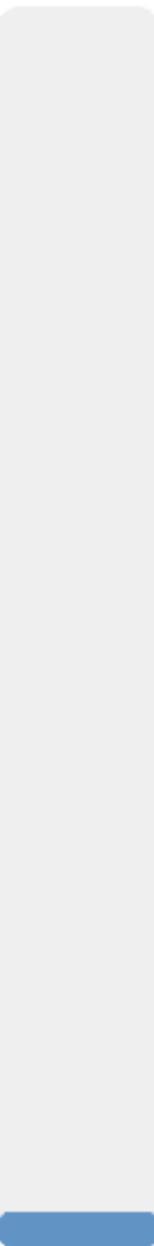
B

0



C

0



D



How compilers affect performance

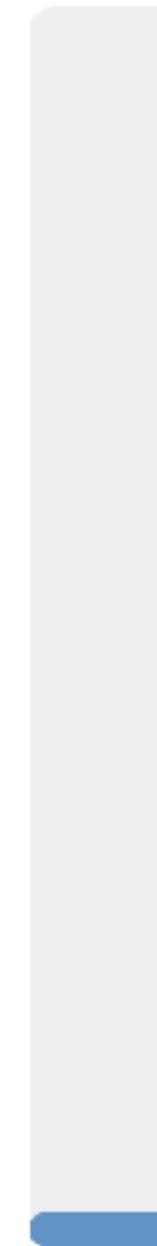
- Performance equation consists of the following three factors
 - ① IC
 - ② CPI
 - ③ CT

How many can the **compiler** affect?

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

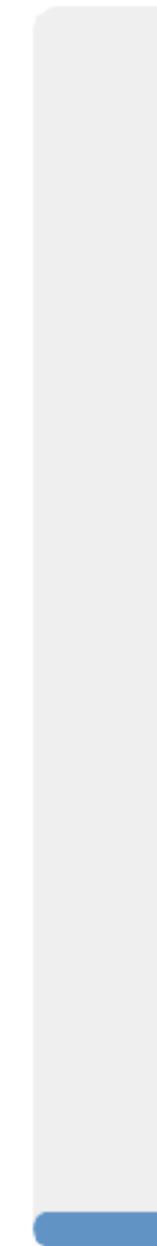
 0

0



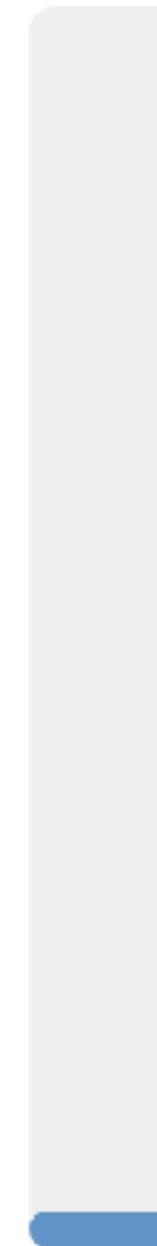
A

0



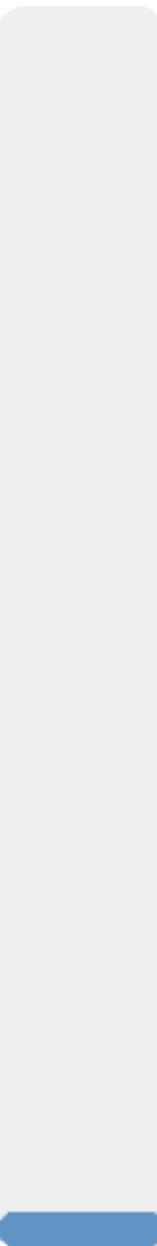
B

0



C

0



D

How compilers affect performance

- Performance equation consists of the following three factors

- ① IC
- ② CPI
- ③ CT

Compiler is less effective than programmer if the programmer wisely tweaked the code based on architecture!

How many can the **compiler** affect?

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

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];
    }
}
```



Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

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];
    }
}
```



Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

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];
    }
}
```

- Compiler has limited effect if the programmer did not do it right

Revisited the demo with compiler optimizations!

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



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



```
for(i = 0; i < 4*QUARTER_SIZE; i++)
{
    for(j = 0; j < 4*QUARTER_SIZE; j++)
    {
        c[i][j] = a[i][j]+b[i][j];
    }
}
```

- Compiler has limited effect if the programmer did not do it right



Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

Takeaways: What matters?

- Execution time is the most essential performance metric
- The following three factors define the execution time of a program
 - Instruction count
 - Cycles per instruction
 - Cycle time
- Programmers can control all factors that affect performance
- 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!

Instruction Set Architecture (ISA) & Performance

Recap: ISA — the interface b/w processor/software

- Operations
 - Arithmetic/Logical, memory access, control-flow (e.g., branch, function calls)
 - Operands
 - Types of operands — register, constant, memory addresses
 - Sizes of operands — byte, 16-bit, 32-bit, 64-bit
- Memory space
 - The size of memory that programs can use
 - The addressing of each memory locations
 - The modes to represent those addresses

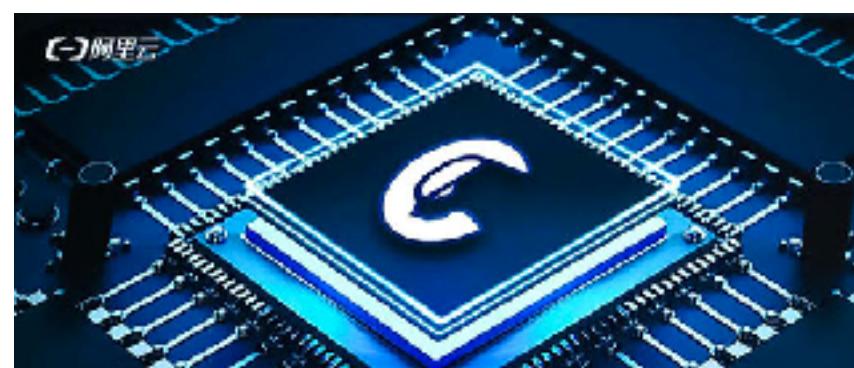
Popular ISAs



x86

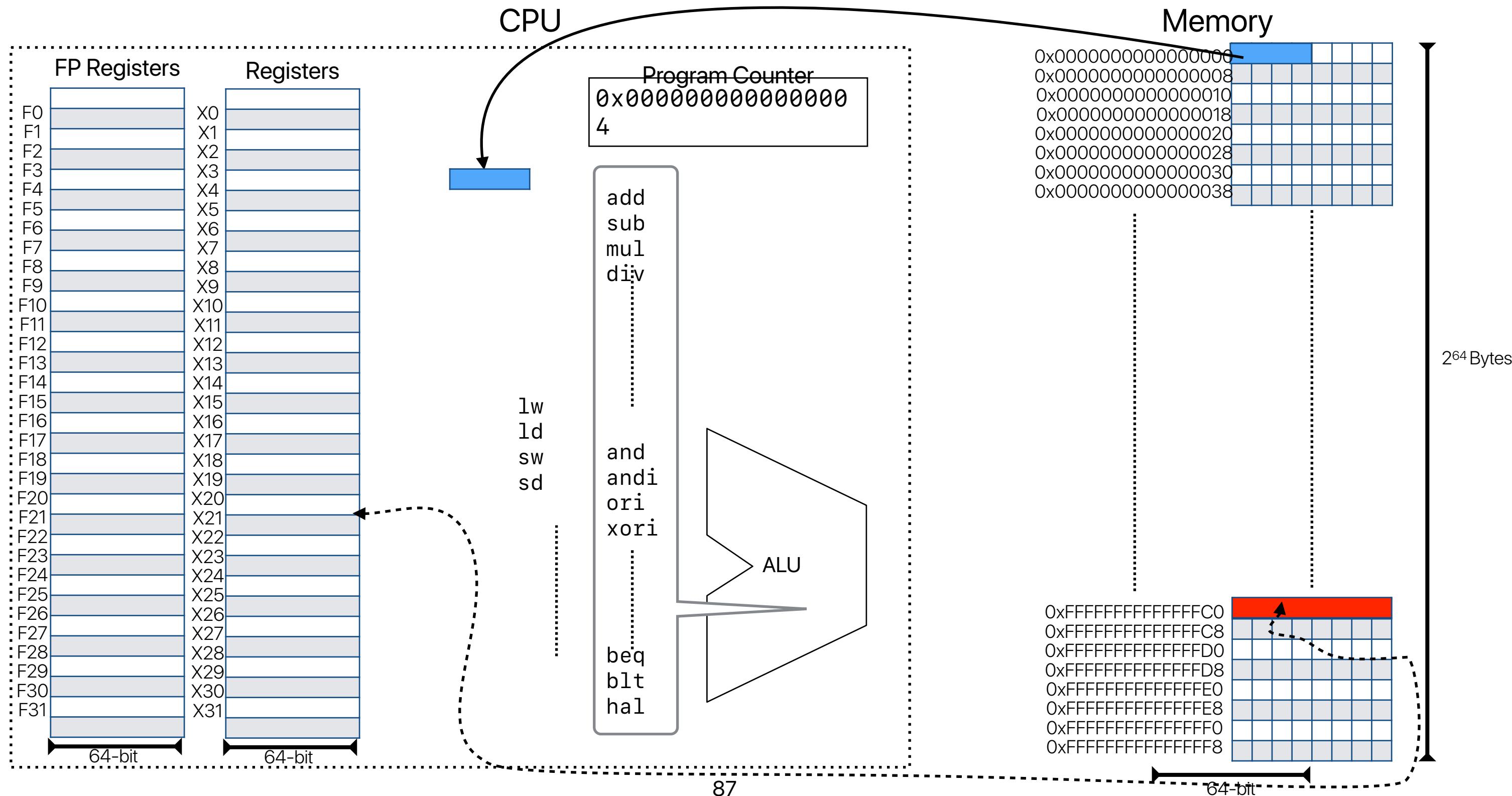


arm



RISC-V

The abstracted “RISC-V” machine

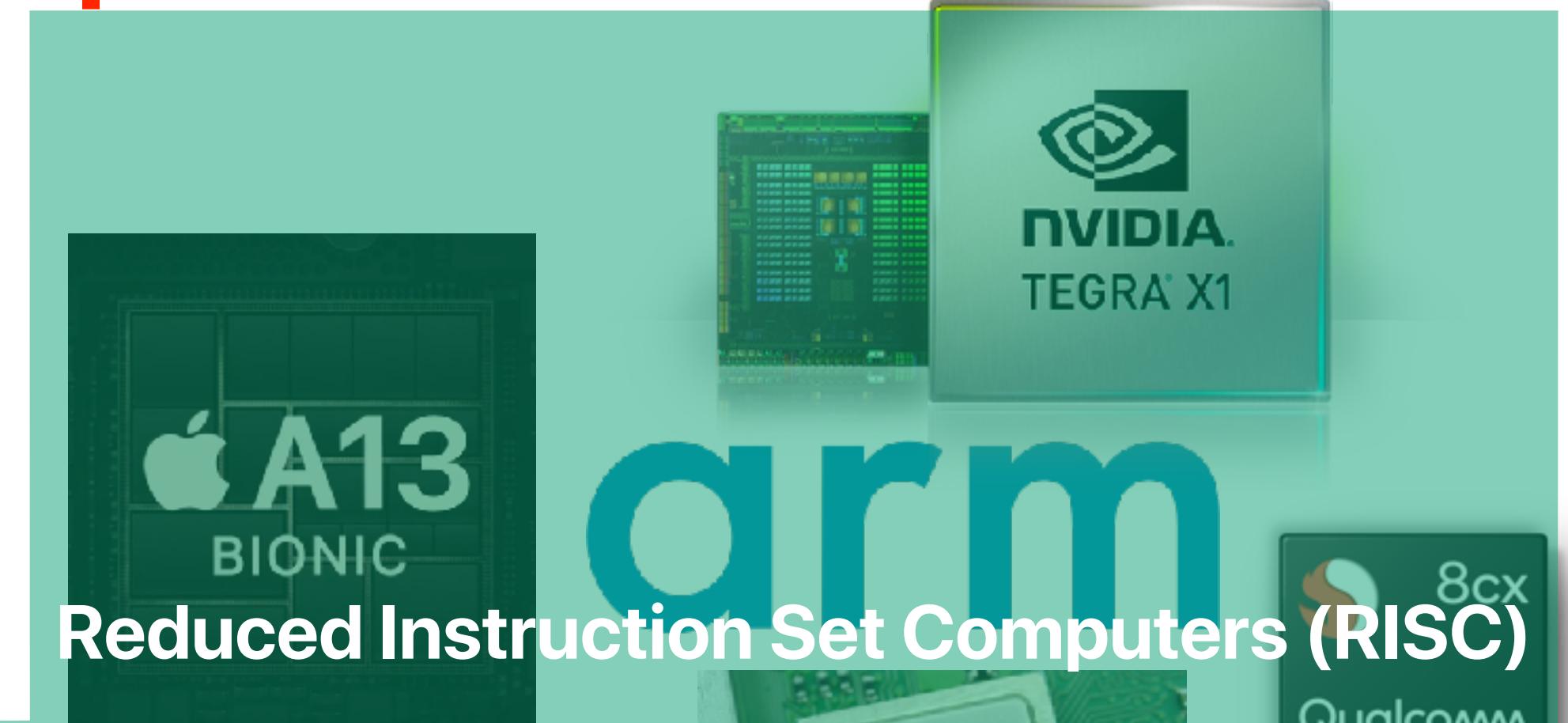


Subset of RISC-V instructions

Category	Instruction	Usage	Meaning
Arithmetic	add	add x1, x2, x3	$x1 = x2 + x3$
	addi	addi x1, x2, 20	$x1 = x2 + 20$
	sub	sub x1, x2, x3	$x1 = x2 - x3$
Logical	and	and x1, x2, x3	$x1 = x2 \& x3$
	or	or x1, x2, x3	$x1 = x2 x3$
	andi	andi x1, x2, 20	$x1 = x2 \& 20$
	sll	sll x1, x2, 10	$x1 = x2 * 2^{10}$
	srl	srl x1, x2, 10	$x1 = x2 / 2^{10}$
Data Transfer	ld	ld x1, 8(x2)	$x1 = \text{mem}[x2+8]$
	sd	sd x1, 8(x2)	$\text{mem}[x2+8] = x1$
Branch	beq	beq x1, x2, 25	if($x1 == x2$), PC = PC + 100
	bne	bne x1, x2, 25	if($x1 != x2$), PC = PC + 100
Jump	jal	jal 25	\$ra = PC + 4, PC = 100
	jr	jr \$ra	PC = \$ra

The only type of instructions can access memory

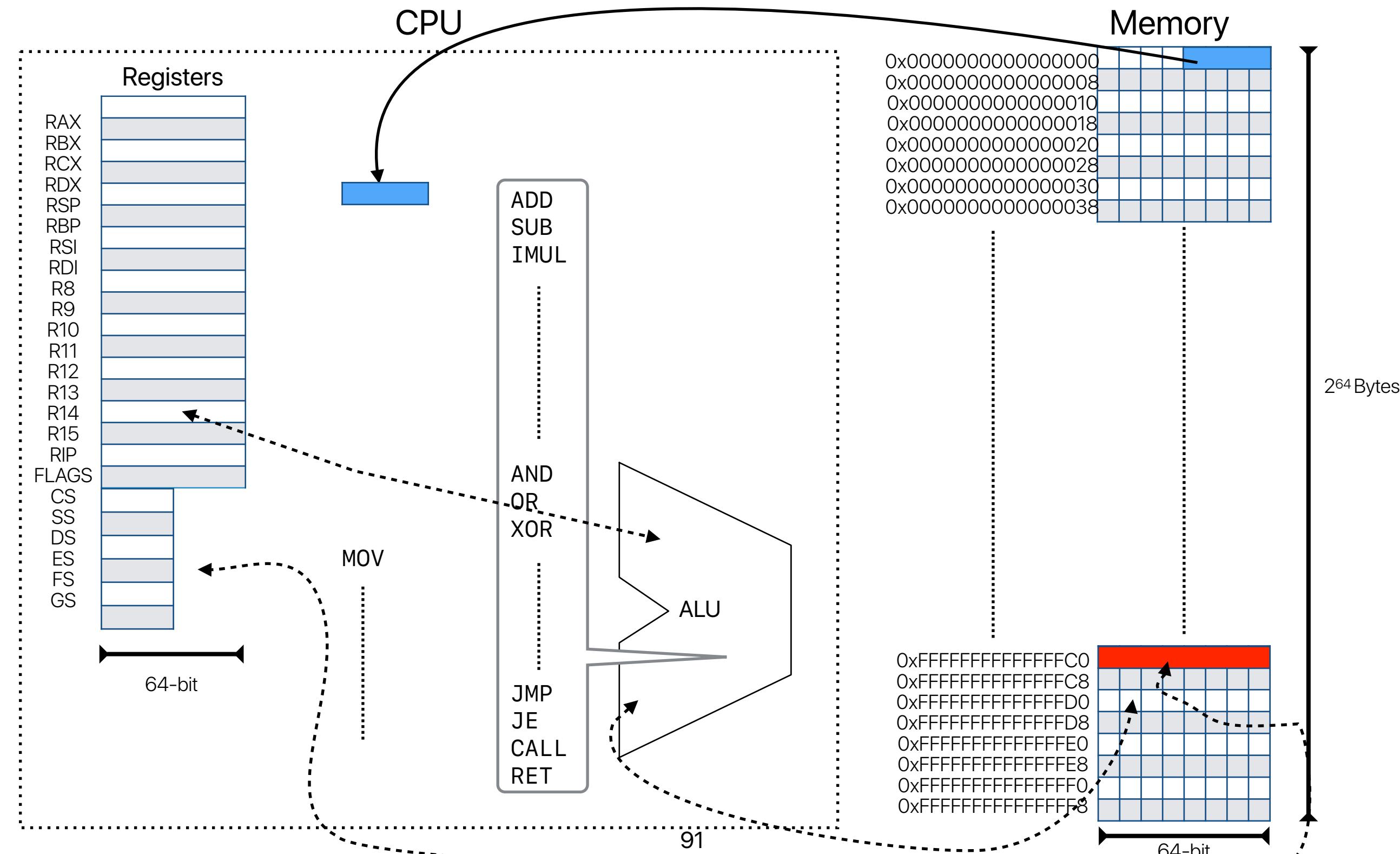
Popular ISAs



How many operations: CISC v.s. RISC

- CISC (Complex Instruction Set Computing)
 - Examples: x86, Motorola 68K
 - Provide **many powerful/complex** instructions
 - Many: more than 1503 instructions since 2016
 - Powerful/complex: an instruction can perform both ALU and memory operations
 - Each instruction takes more cycles to execute
- RISC (Reduced Instruction Set Computer)
 - Examples: ARMv8, RISC-V, MIPS (the first RISC instruction, invented by the authors of our textbook)
 - Each instruction only performs simple tasks
 - Easy to decode
 - Each instruction takes less cycles to execute

The abstracted x86 machine



RISC-V v.s. x86

	RISC-V	x86
ISA type	Reduced Instruction Set Computers (RISC)	Complex Instruction Set Computers (CISC)
instruction width	32 bits	1 ~ 17 bytes
code size	larger	smaller
registers	32	16
addressing modes	reg+offset	base+offset base+index scaled+index scaled+index+offset
hardware	simple	complex

Takeaways: What matters?

- Execution time is the most essential performance metric
- The following three factors define the execution time of a program
 - Instruction count
 - Cycles per instruction
 - Cycle time
- Programmers can control all factors that affect performance
- 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!
- ISA can affect performance, again, the programmer/designer needs to know

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

Speedup

- The relative performance between two machines, X and Y. Y is n times faster than X, or say the speedup of Y over X

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

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 — and It's Implication in the Multicore Era

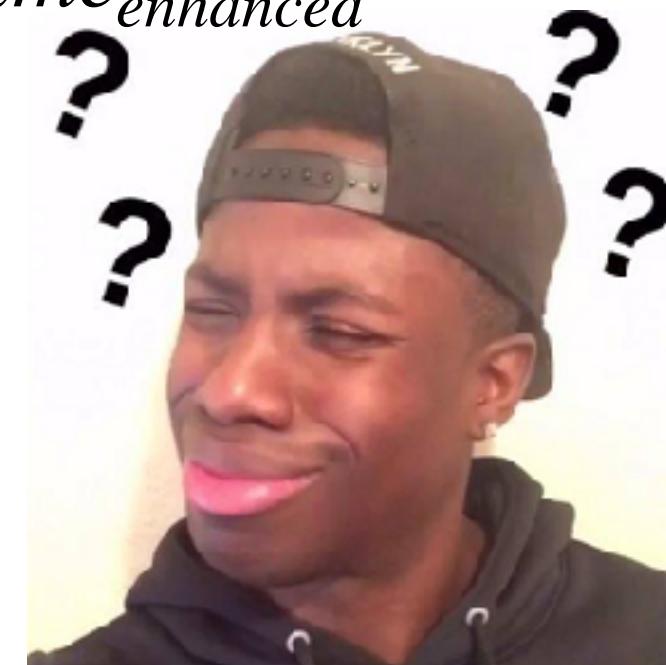
Amdahl's Law



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

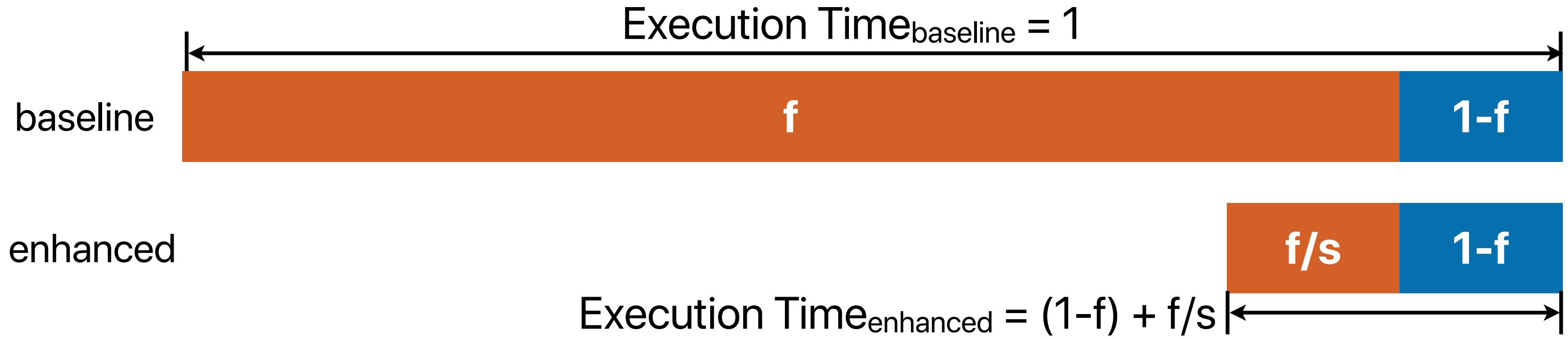
f — The fraction of time in the original program
 s — The speedup we can achieve on f

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



Amdahl's Law

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



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

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



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 of an optimization is bounded by

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

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

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 — $Speedup_{enhanced}(f, s) = \frac{1}{(1-f) + \frac{f}{s}}$ $Speedup_{max}(f, \infty) = \frac{1}{(1-f)}$
 - Corollary 1 — each optimization has an upper bound

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.

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

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

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

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

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

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

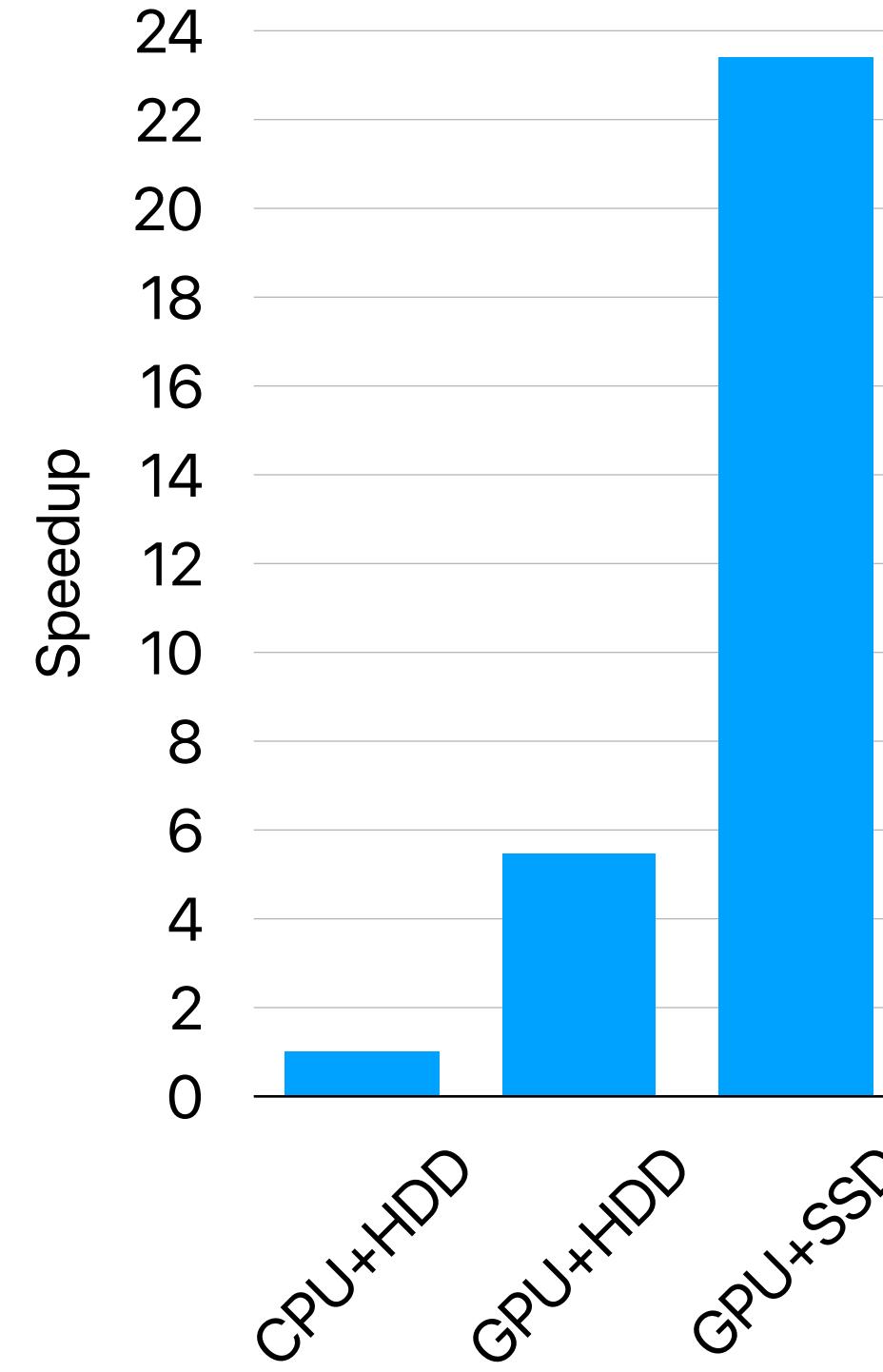
Identify the most time consuming part

- 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`

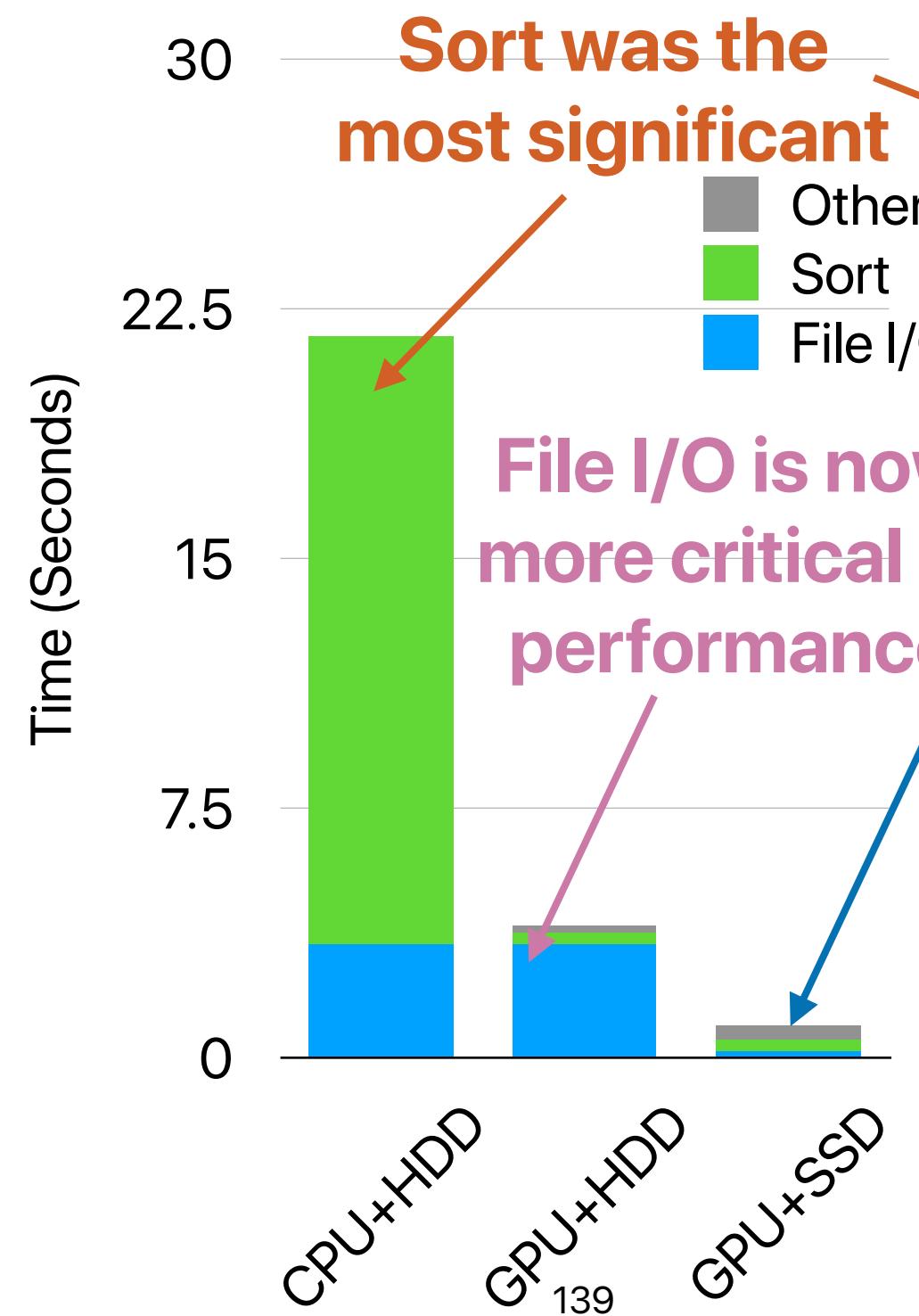
Demo — sort

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

Speedup



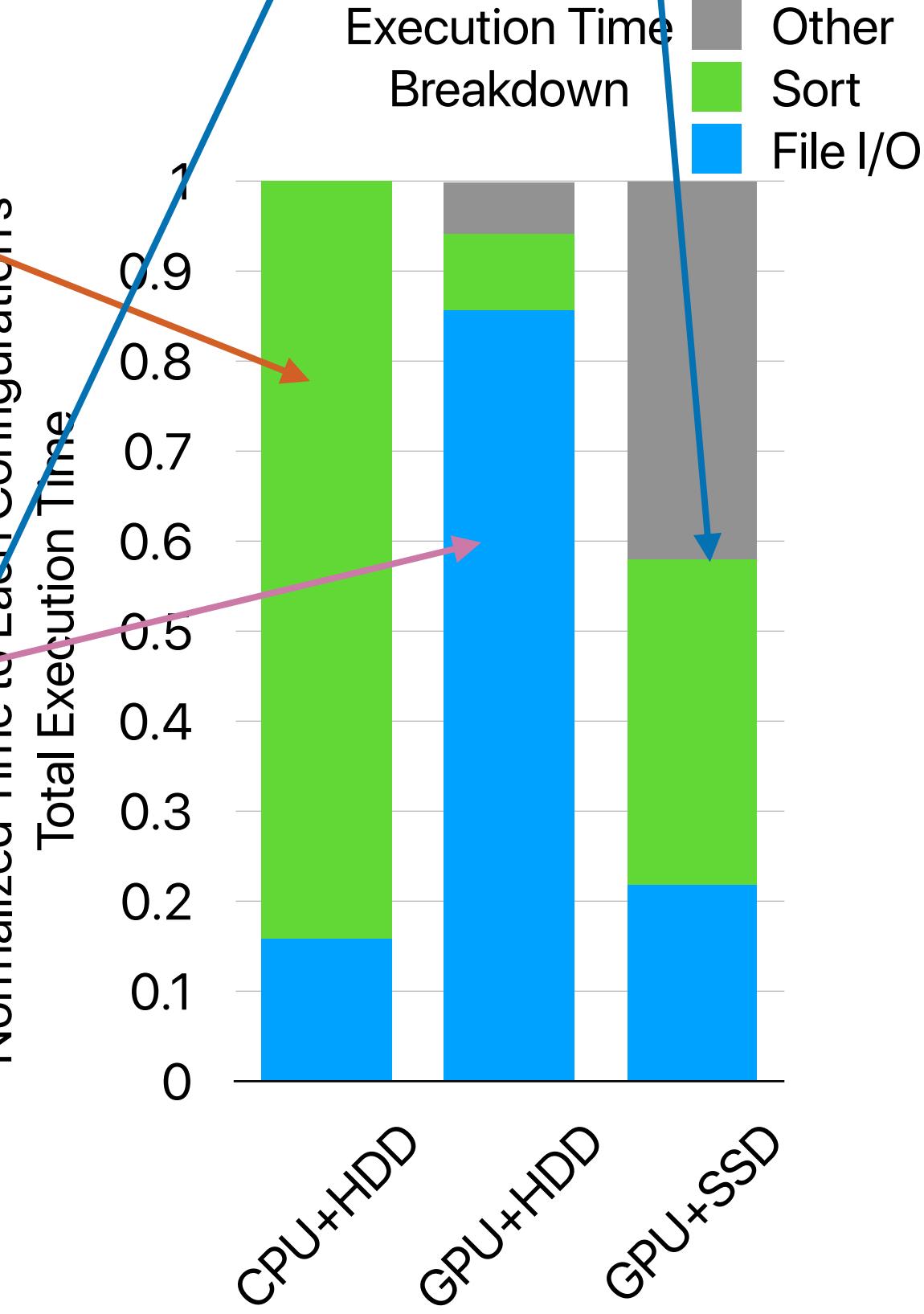
Cumulative Execution Time



Sort was the most significant

File I/O is now more critical to performance

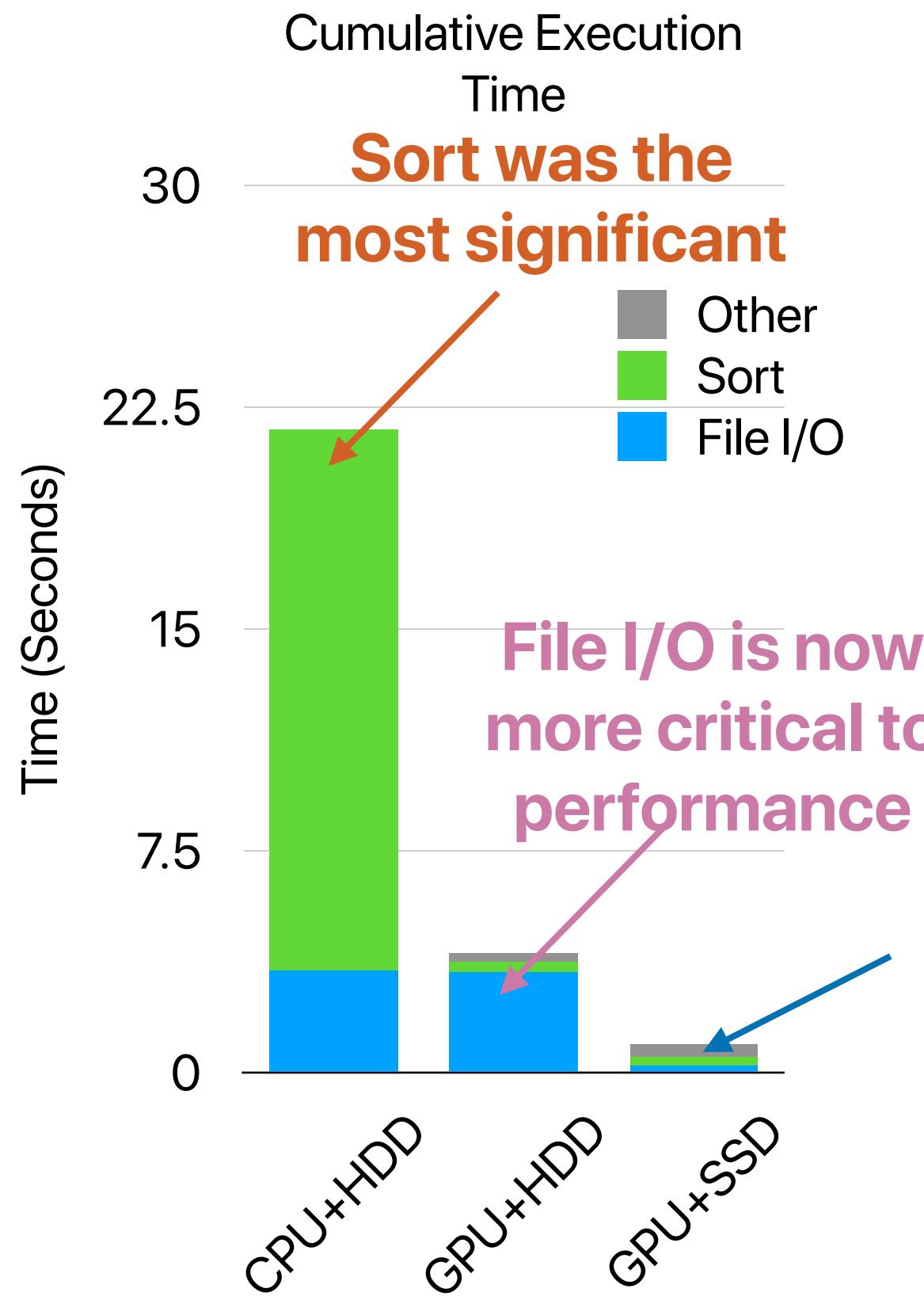
Normalized Time to Each Configuration's Total Execution Time



Execution Time Breakdown

Other
Sort
File I/O

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

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

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

- Corollary 1 — each optimization has an upper bound

- 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

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

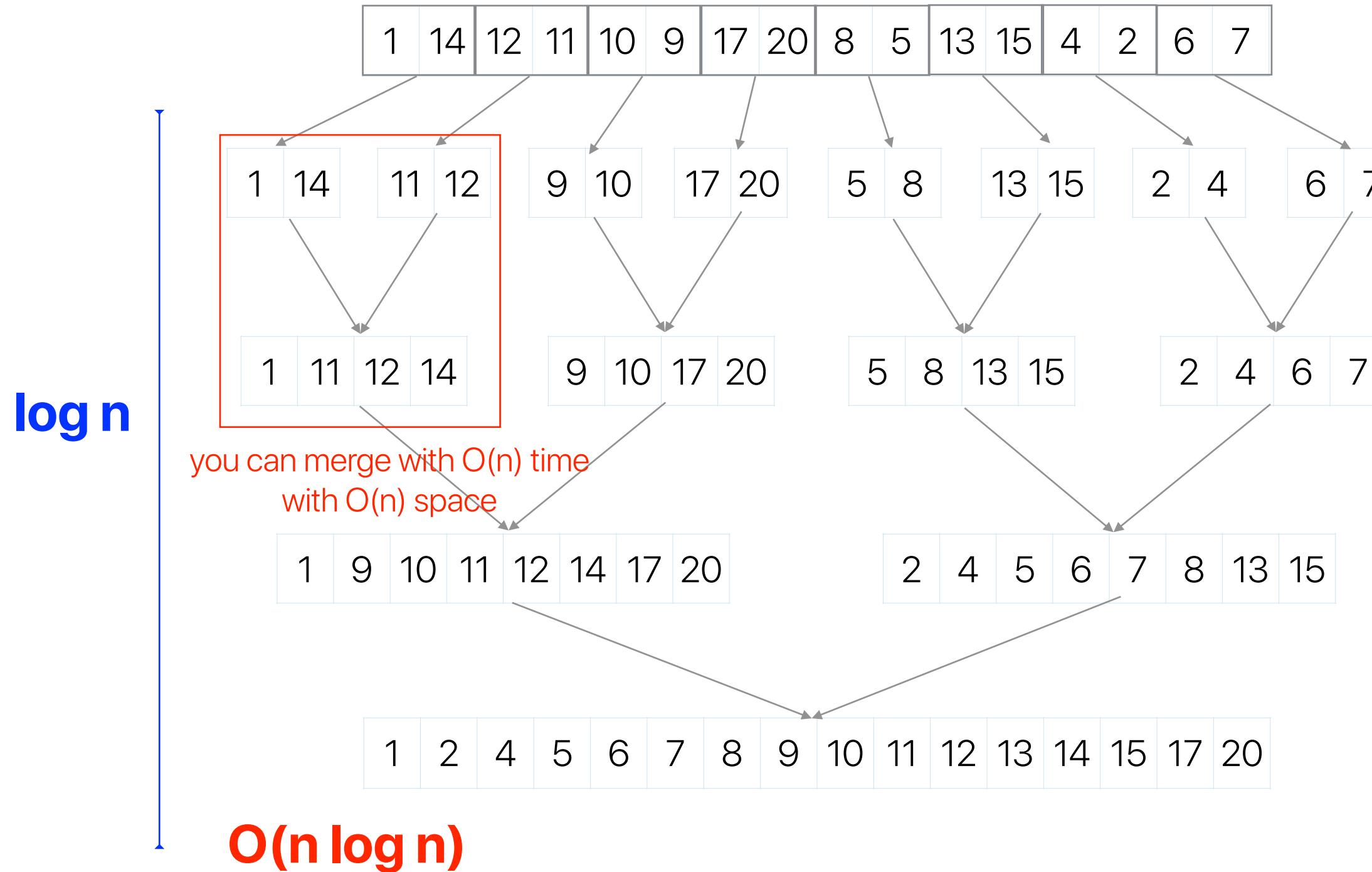
$$O(n \log_2 n)$$

Bitonic Sort

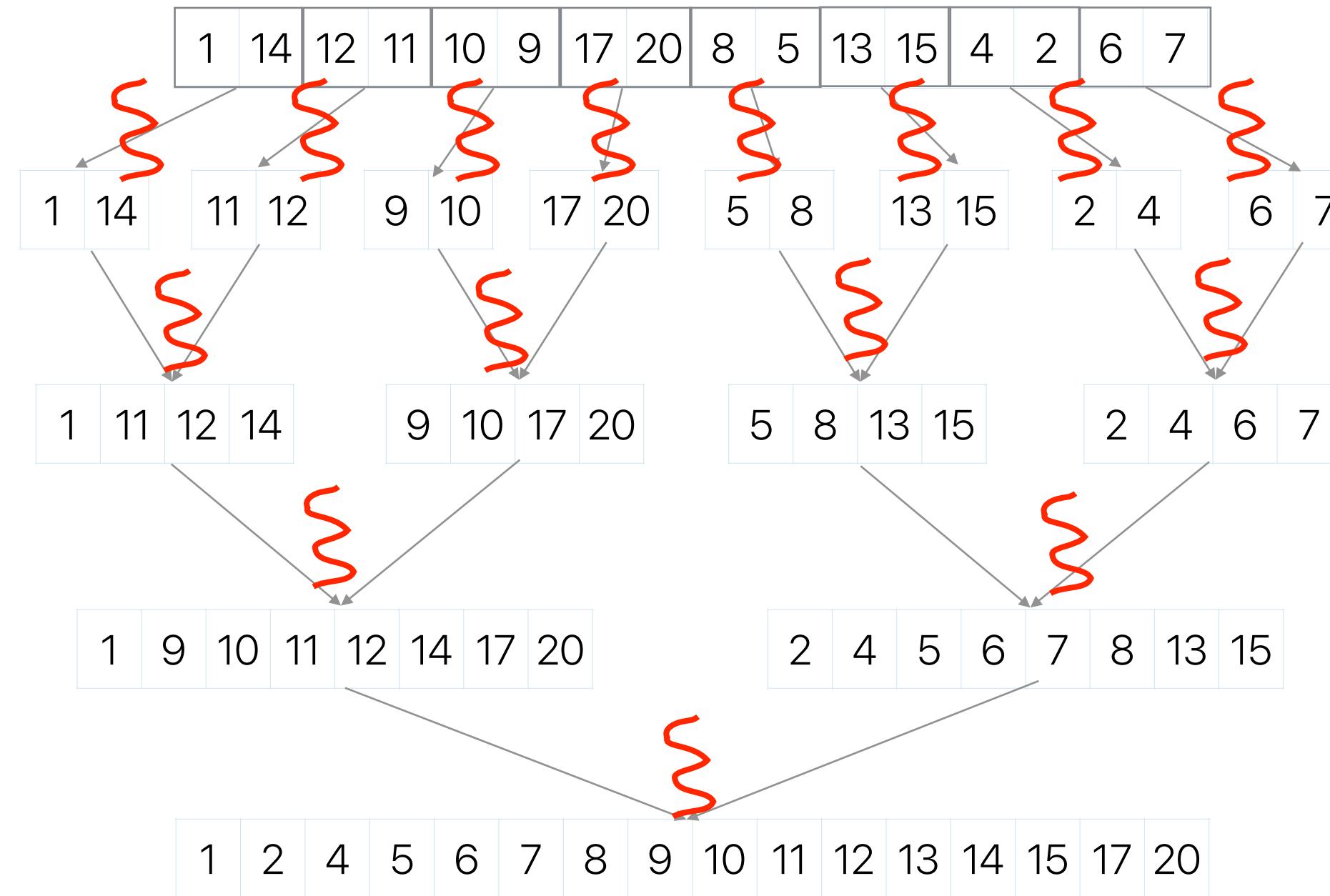
$$O(n \log_2^2 n)$$

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

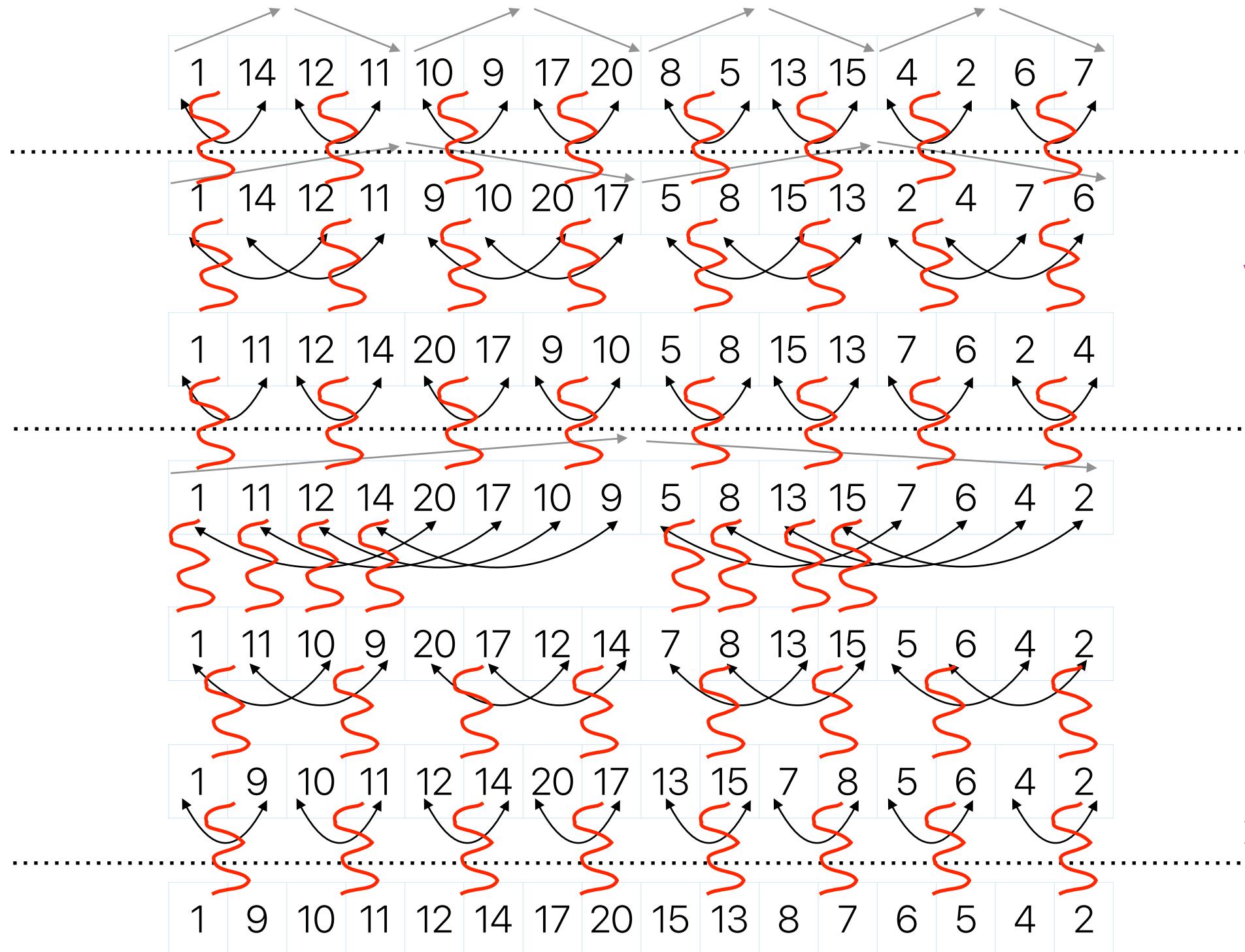
Merge sort



Parallel merge sort

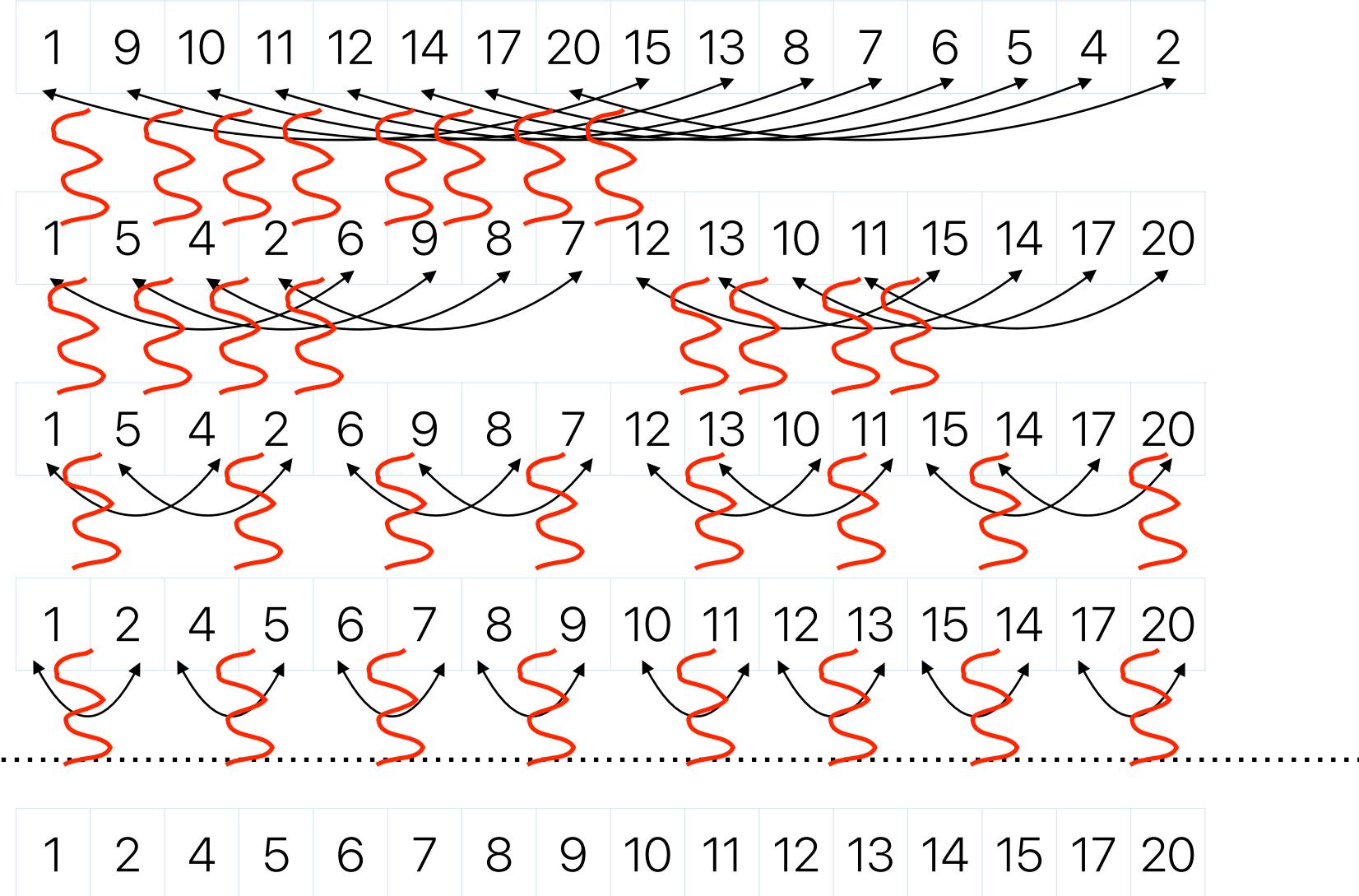


Bitonic sort



```
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**

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

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

- Corollary 1 — each optimization has an upper bound

- 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

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

**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

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

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

- Corollary 1 — each optimization has an upper bound

- 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

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

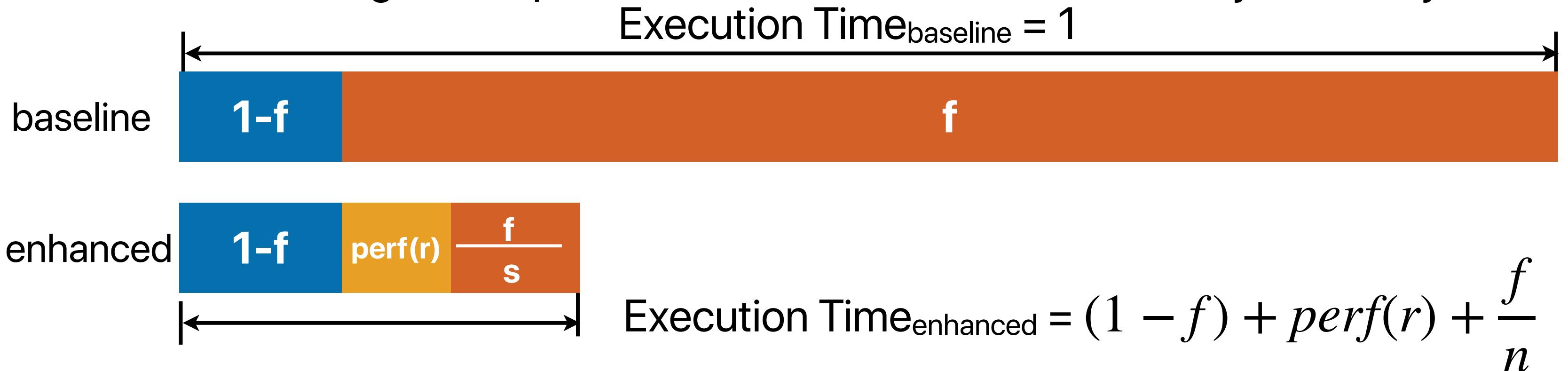
- Corollary 5: Single-core performance still matters

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

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 (i.e., an overhead that is 9x longer than the original execution time)...

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

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

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

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

V. Sze, Y.-H. Chen, T.-J. Yang and J. S. Emer. How to Evaluate Deep Neural Network Processors: TOPS/W (Alone) Considered Harmful. In IEEE Solid-State Circuits Magazine, vol. 12, no. 3, pp. 28-41, Summer 2020.

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)

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!



Artificial Intelligence Computing Leadership from NVIDIA

CLOUD & DATA CENTER

PRODUCTS ▾

SOLUTIONS ▾

APPS ▾

FOR DEVELOPERS

TECHNOLOGIES ▾

Tesla V100

AI TRAINING

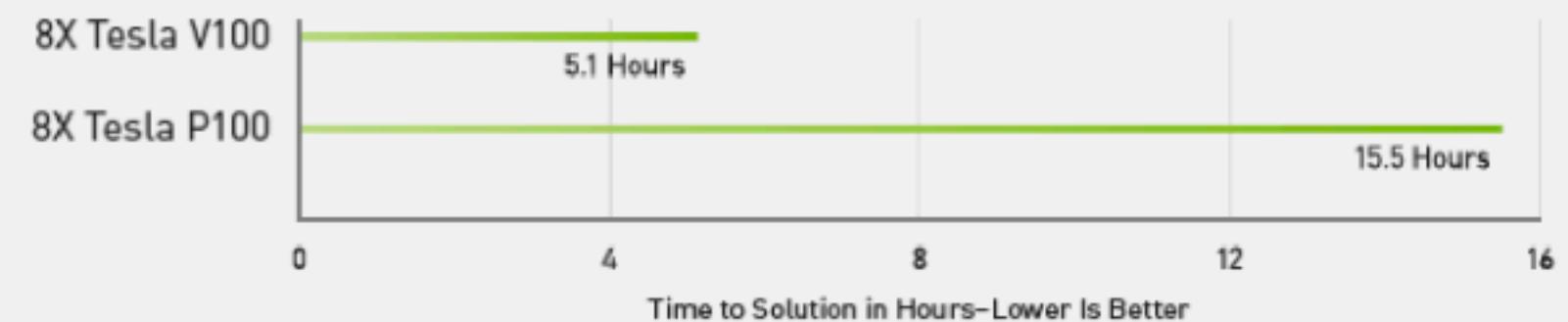
AI INFERENCE

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.

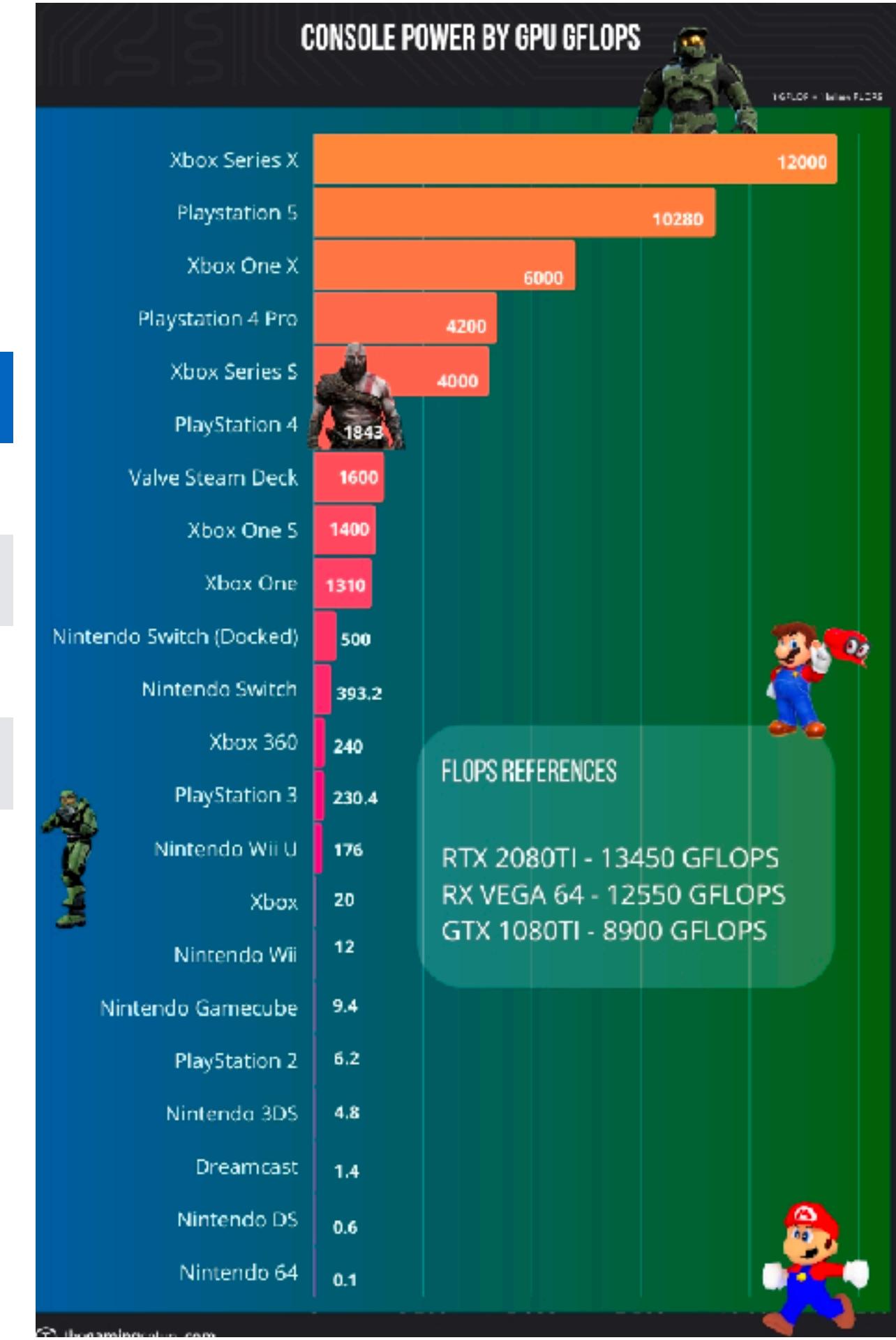
AI TRAINING

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



TFLOPS (Tera FLoating-point Operations Per Second)

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

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 $i = j = k = 2048$

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

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

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

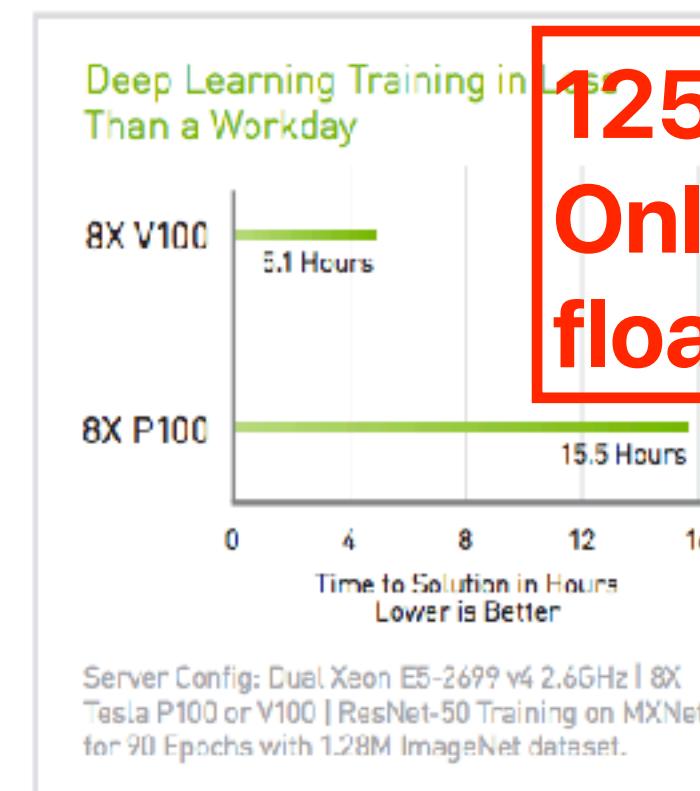
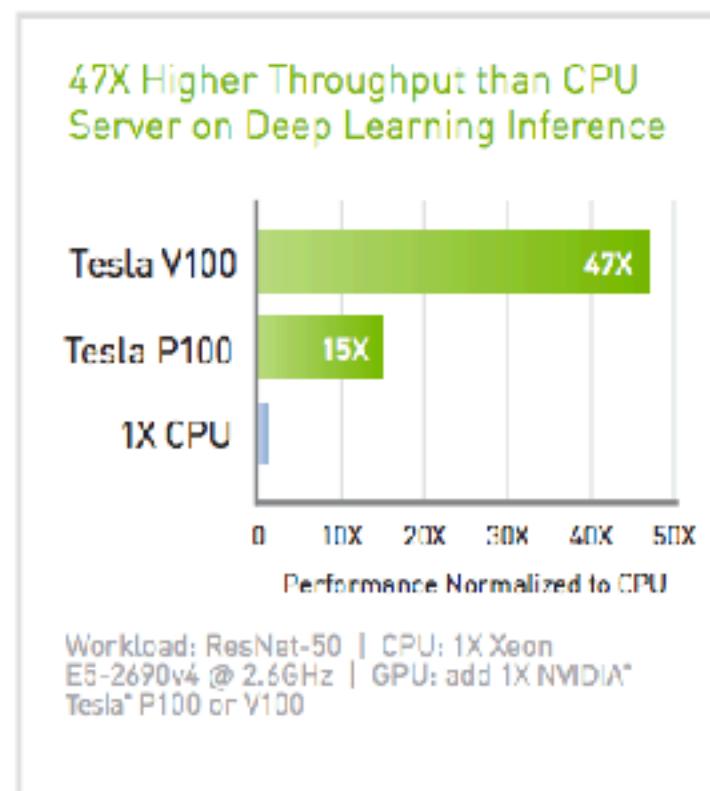
$$\begin{aligned}TFLOPS &= \frac{\# \text{ of floating point instructions} \times 10^{-12}}{\text{Execution Time}} \\&= \frac{IC \times \% \text{ of floating point instructions} \times 10^{-12}}{IC \times CPI \times CT} \\&= \frac{\% \text{ of floating point instructions} \times 10^{-12}}{CPI \times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

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**



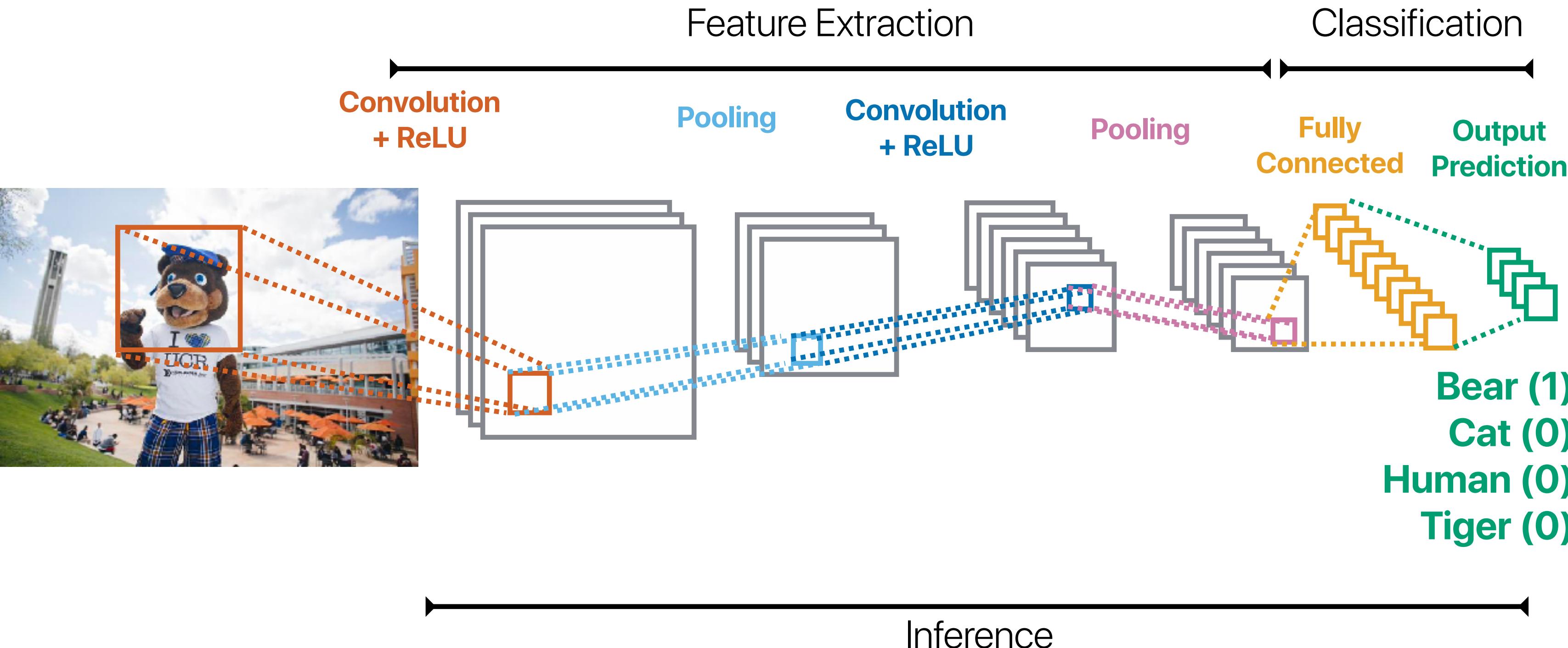
**Tesla V100
SXM2**

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

The Machine Learning Inference Pipeline



Bard v.s. ChatGPT 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/

Facebook icon | Twitter icon | Email icon | Google It

How important is Computer Architecture in Computer Science? ▶

1 answer(inference)/6 secs



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(inference)/18 secs

Inference per second is intensively used

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

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	1/13X	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

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

Bard v.s. ChatGPT Round #2

Bard



Screenshot of the Bard web interface. The title bar says "Bard". The main content area shows a card for "Who is teaching CS203 at UCR?". It includes a profile picture of Hung-Wei Tseng, a brief bio, and contact information. A sidebar on the left lists "Recent chats", "Bard Activity", "FAQ", and "Help & support". At the bottom, there's a search bar with "Who is Hung-Wei Tseng?" and a note about Google's data.

Screenshot of the ChatGPT web interface. The title bar says "ChatGPT". The main content area shows a card for "Who is teaching CS203 at UCR?". It includes a note that ChatGPT is an AI language model and does not have real-time access to course information. A sidebar on the left shows a list of recent chats and a dropdown menu with options like "Clear conversations", "Update to Plus", "Dark mode", "Updates & FAQ", and "Log out". At the bottom, there's a search bar with "Who is Hung-Wei Tseng?" and a note about ChatGPT's mission.

1 answer/6 secs

1 answer/6 secs

What's (not) considered by Inference per second

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

$$= \frac{\text{Inferences}}{\text{Operation}} \times \left[\frac{\text{operations}}{\text{cycle}} \times \frac{\text{cycles}}{\text{second}} \times \#_{\text{of_PEs}} \times \text{Utilization_of_PEs} \right]$$

	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?

Takeaways: Do the right thing

- Choose the right performance metric
 - A good metric should consider all factors of performance equation
 - Performance is not determined by just a single factor

“Fair” Comparisons

Andrew Davison. Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers.
In Humour the Computer, MITP, 1995
Norman P. Jouppi et. al.. In-Datacenter Performance Analysis of a Tensor Processing Unit
<https://doi.org/10.1145/3079856.3080246>



Extreme Multitasking Performance

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

Is this video convincing? Why or
why not?

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?

12 ways to Fool the Masses When Giving Performance Results on Parallel Computers

- Quote only 32-bit performance results, not 64-bit results.
- Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
- Quietly employ assembly code and other low-level language constructs.
- Scale up the problem size with the number of processors, but omit any mention of this fact.
- Quote performance results projected to a full system.
- Compare your results against scalar, unoptimized code on Crays.
- When direct run time comparisons are required, compare with an old code on an obsolete system.
- If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
- Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar.
- Mutilate the algorithm used in the parallel implementation to match the architecture.
- Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
- If all else fails, show pretty pictures and animated videos, and don't talk about performance.

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

Why we need benchmark suites

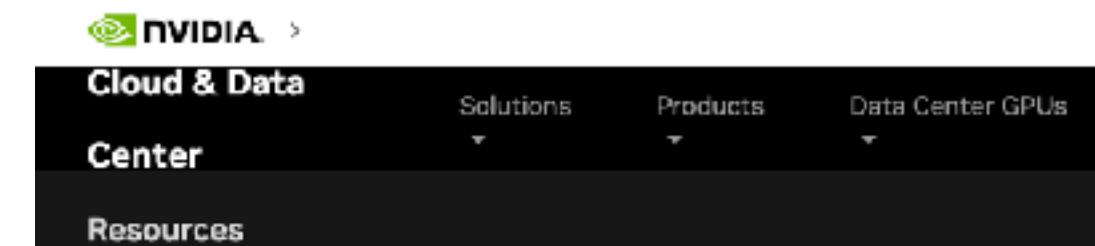
- Allowing people evaluate systems with exactly the same program and the same inputs and validate results from different machines
- Popular benchmark suites
 - SPEC — CPU benchmark
 - MLPerf — ML systems



The screenshot shows the homepage of the SPEC Standard Performance Evaluation Corporation. The header features the SPEC logo and the text "Standard Performance Evaluation Corporation". Below the header is a navigation bar with links for Home, Benchmarks, Tools, Results, Contact, Blog, Join Us, Search, and Help. A sidebar on the left contains links for Published Results, Results Search, Fair Use Policy, and Information, which includes sections for CPURE17, Documentation, Overview, System Requirements, Run & Reporting Rules, Using SPEC-CPURE17, Definitions, Technical Report, and FAQ. The main content area is titled "SPEC CPU® 2017" and describes the benchmark package.

- *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].

In-Datacenter Performance Analysis of a Tensor Processing Unit
<https://doi.org/10.1145/3079856.3080246>



The screenshot shows the NVIDIA Cloud & Data Center Resources page. The top navigation bar includes the NVIDIA logo, Cloud & Data Center, Solutions, Products, and Data Center GPUs. Below the navigation is a section titled "MLPerf Benchmarks" with the subtext: "The NVIDIA AI platform showcases leading performance and versatility in MLPerf Training, Inference, and HPC for the most demanding, real-world AI workloads." On the right side, there is a thumbnail image of a server rack.

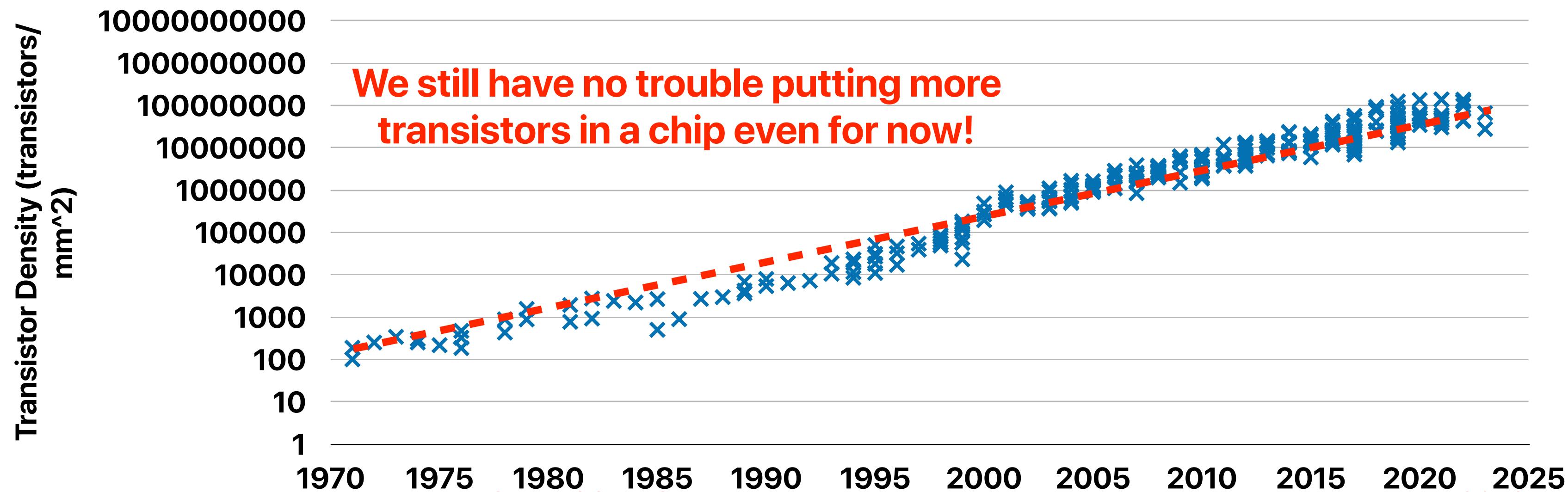
Takeaways: Do the right thing

- Choose the right performance metric
 - A good metric should consider all factors of performance equation
 - Performance is not determined by just a single factor
- Don't fool others but also fool yourself
 - Fair comparison matters

The limiting factor of modern processor performance

Moore's Law⁽¹⁾

- The number of transistors we can build in a fixed area of silicon doubles every 12 ~ 24 months.
- Moore's Law "was" the most important driver for historic CPU performance gains



(1) Moore, G. E. (1965), 'Cramming more components onto integrated circuits', Electronics 38 (8).

What happens if power doesn't scale with process technologies?

- If we are able to cram more transistors within the same chip area (Moore's law continues), but the power consumption per transistor remains the same. Right now, if put more transistors in the same area because the technology allows us to. How many of the following statements are true?
 - ① The power consumption per chip will increase
 - ② The power density of the chip will increase
 - ③ Given the same power budget, we may not be able to power on all chip area if we maintain the same clock rate
 - ④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area

A. 0

B. 1

C. 2

D. 3

E. 4

Power v.s. Energy

- Power is the direct contributor of “heat”
 - Packaging of the chip
 - Heat dissipation cost
- $\text{Energy} = P * ET$
 - 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)

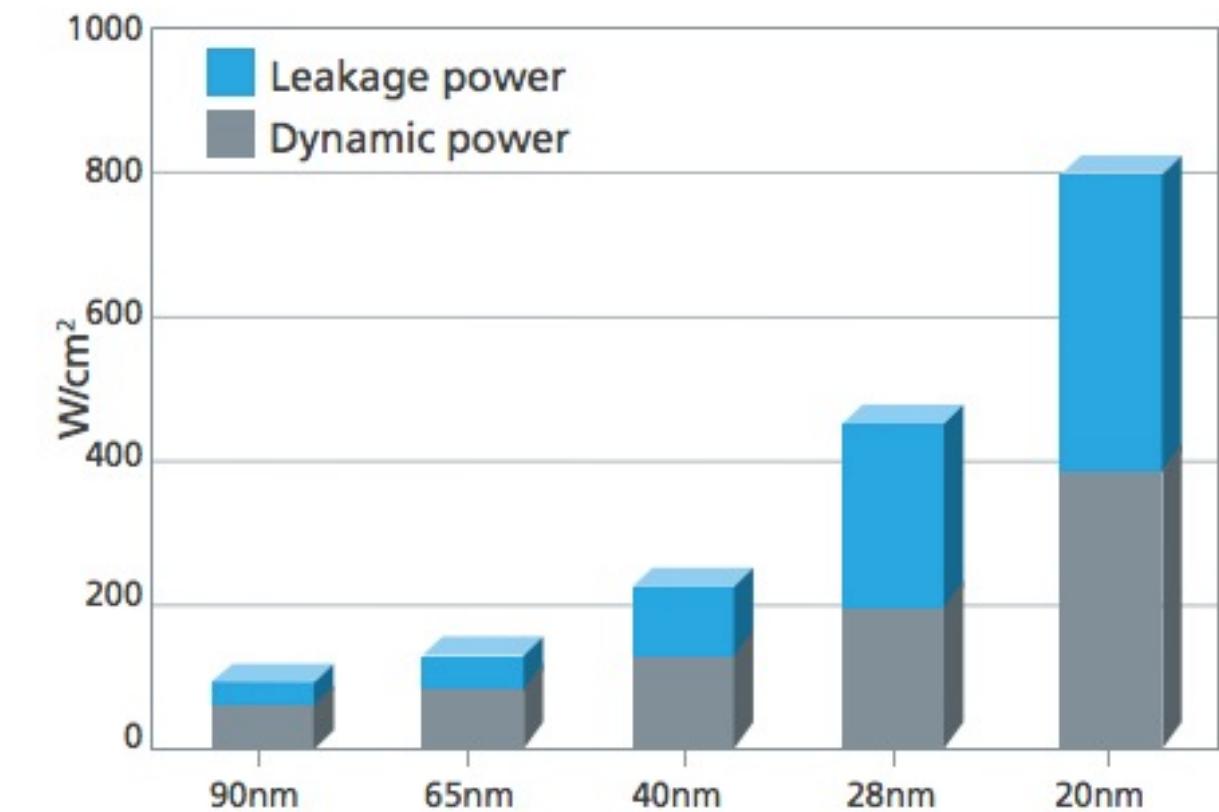


Figure 1: Leakage power becomes a growing problem as demands for more performance and functionality drive chipmakers to nanometer-scale process nodes (Source: IBS).

Power consumption & power density

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

- α : average switches per cycle

- C : capacitance

- V : voltage

- f : frequency, usually linear with V

- N : the number of transistors

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

- N : number of transistors

- V : voltage

- V_t : threshold voltage where transistor conducts (begins to switch)

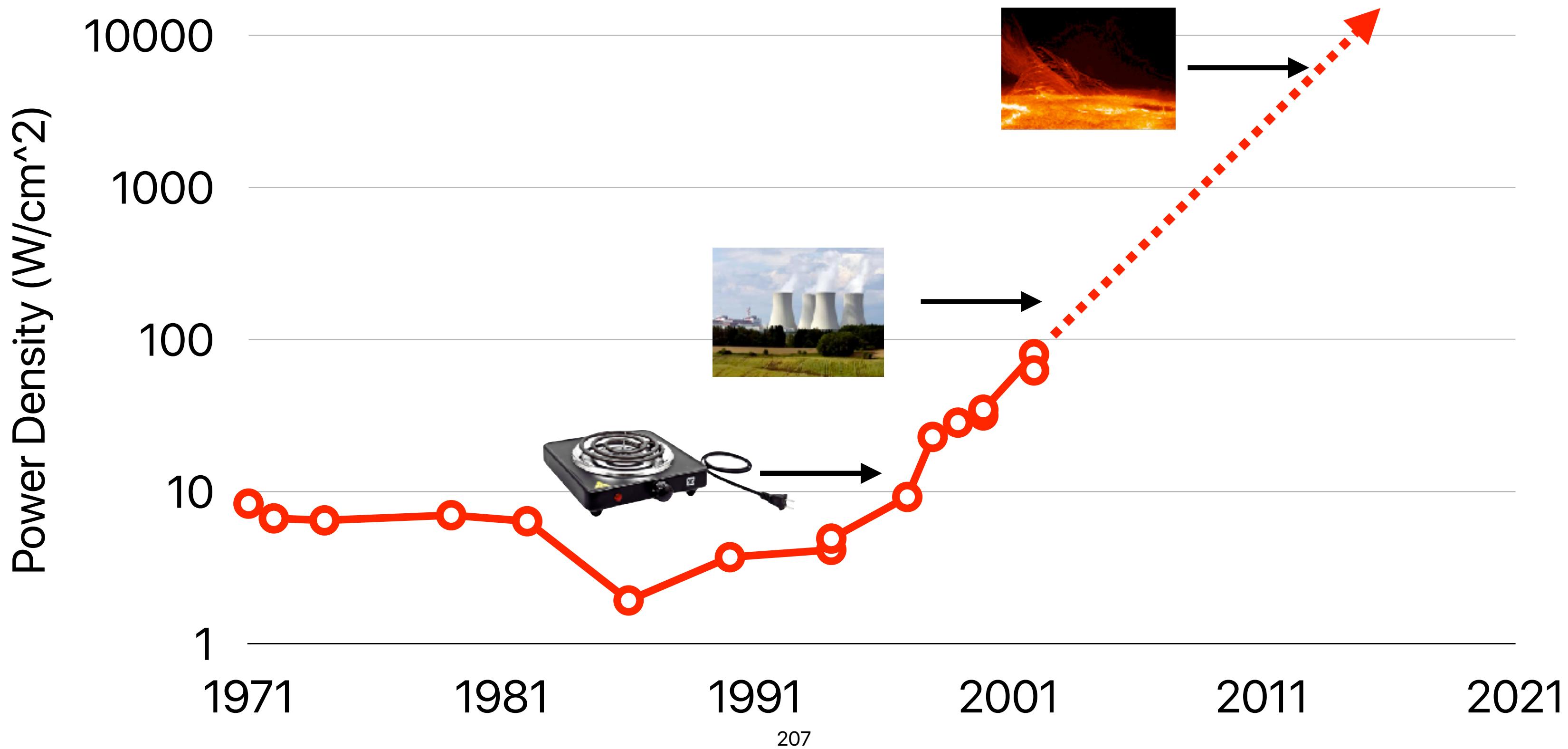
- Power density:

$$P_{density} = \frac{P}{area}$$

Dennard scaling discontinued —
we cannot make voltage lower

Moore's Law allows higher frequencies as transistors are smaller
Moore's Law makes this smaller

Power Density of Processors



Takeaways: Do the right thing

- Choose the right performance metric
 - A good metric should consider all factors of performance equation
 - Performance is not determined by just a single factor
- Don't fool others but also fool yourself
 - Fair comparison matters
- Power and energy become increasingly important
 - Power affects how fast a transistor can run
 - Energy affects the cost and carbon footprint

Computer Science & Engineering

203

つづく

