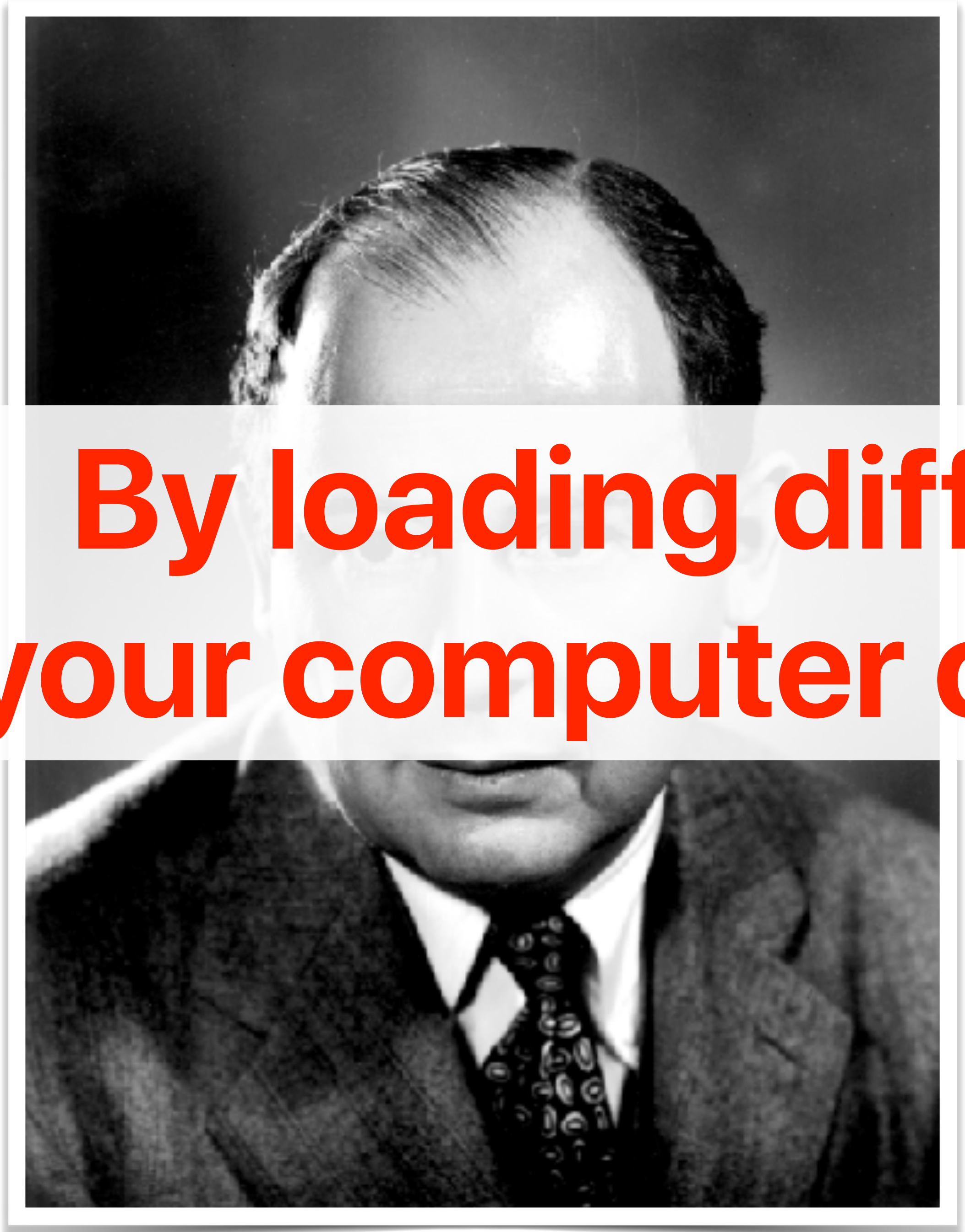
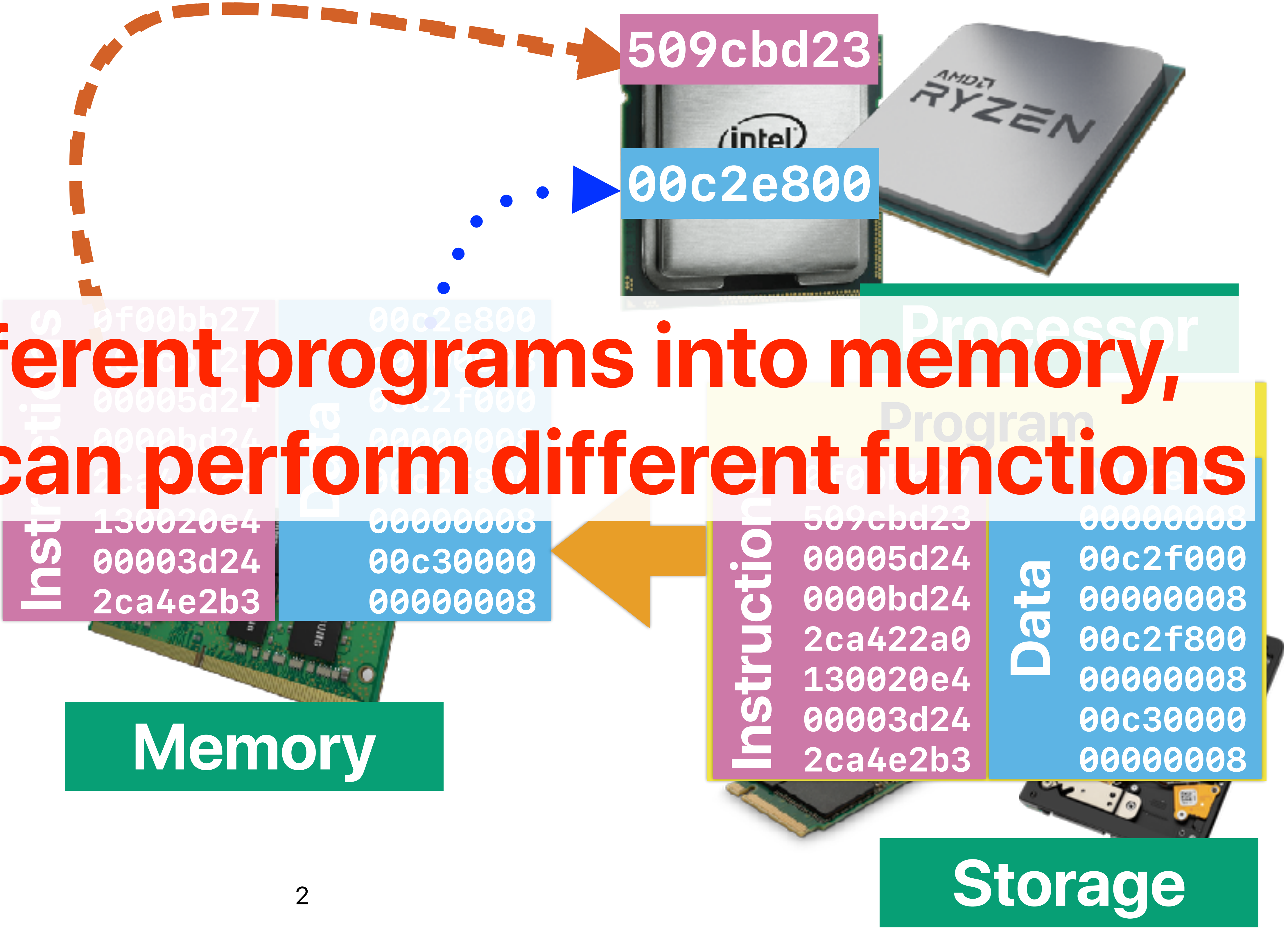
Performance (II): Amdahl’s Law and it’s implications

Hung-Wei Tseng

Recap: von Neumman Architecture

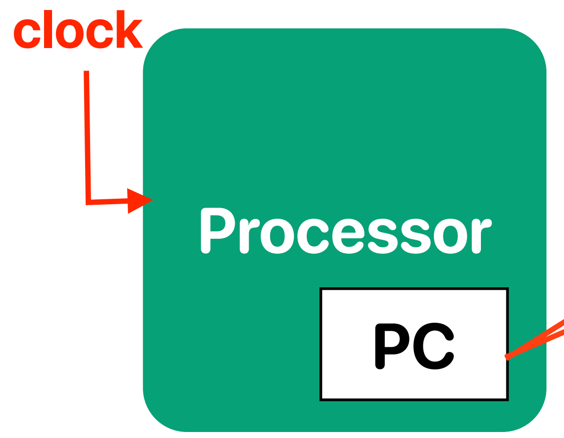




By loading different programs into memory, your computer can perform different functions

Recap: Execution Time

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



instruction memory

```
120007a30:  0f00bb27  ldah  gp,15(t12)
120007a34:  509cbd23  lda   gp,-25520(gp)
120007a38:  00005d24  ldah  t1,0(gp)
120007a3c:  0000bd24  ldah  t4,0(gp)
120007a40:  2ca422a0  ldl   t0,-23508(t1)
120007a44:  130020e4  beq   t0,120007a94
120007a48:  00003d24  ldah  t0,0(gp)
120007a4c:  2ca4e2b3  stl   zero,-23508(t1)
120007a50:  0004ff47  clr   v0
120007a54:  28a4e5b3  stl   zero,-23512(t4)
120007a58:  20a421a4  ldq   t0,-23520(t0)
120007a5c:  0e0020e4  beq   t0,120007a98
120007a60:  0204e147  mov   t0,t1
120007a64:  0304ff47  clr   t2
120007a68:  0500e0c3  br    120007a80
```

$$\frac{Instructions}{Program}$$

How many of these?

How long is it take to execution each of these?

$$\frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Recap: CPU Performance Equation

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

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

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

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

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

Recap: Speedup

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

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

- The speedup of Y over X

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

Recap: How programmer affects performance?

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

How many can a **programmer** affect?

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

Recap: How programming languages affect performance

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

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

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

Team scores

0	1	1	1

Outline

- What affects each factor in “Performance Equation” (cont.)
- Amdahl’s law and it’s implications

Poll close in 1:30

1 — Instinct

Programming languages

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

Programming languages

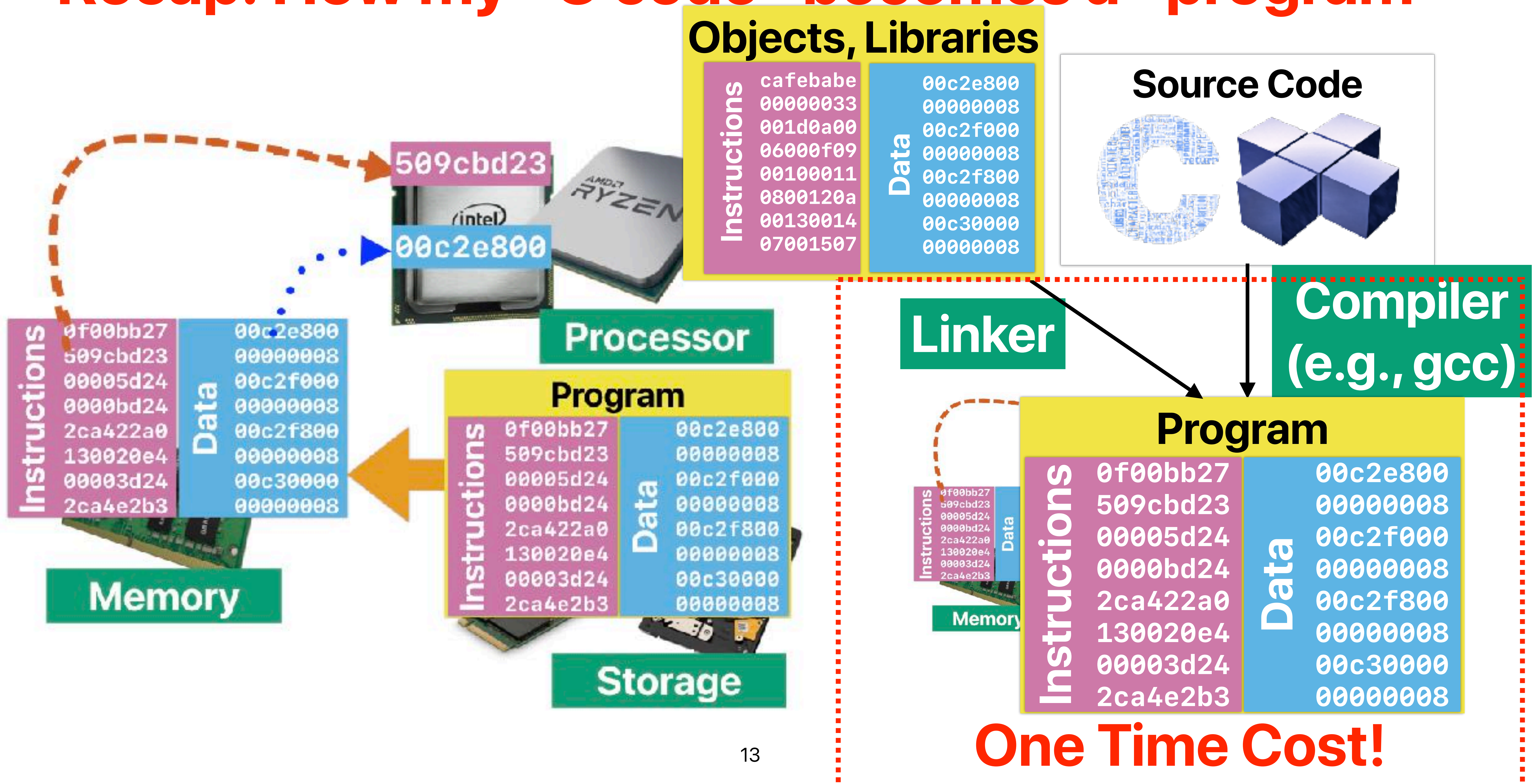
- How many instructions are there in "Hello, world!"

	Instruction count	LOC	Ranking
C	600k	6	1
C++	3M	6	2
Java	~210M	8	5
Perl	10M	4	3
Python	~30M	1	4

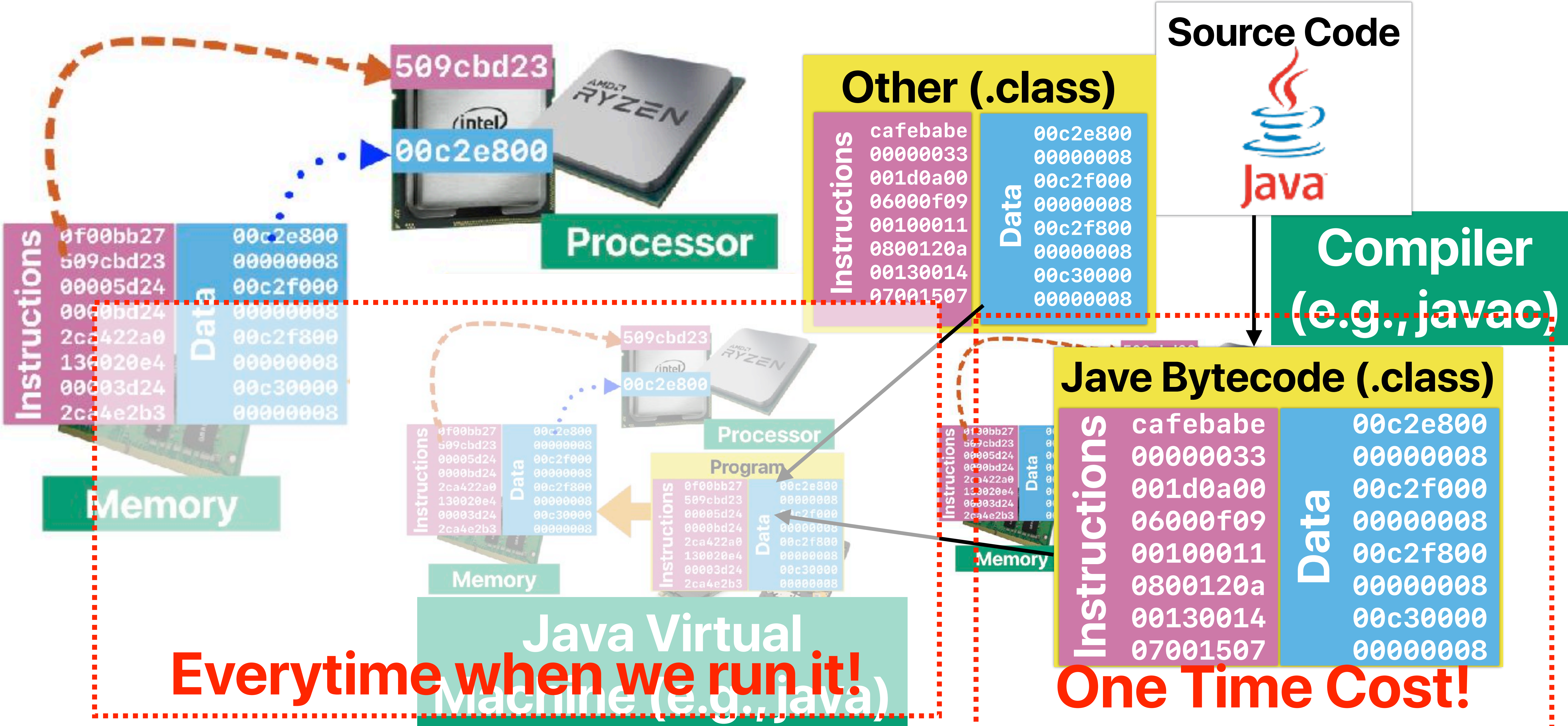
Programming languages

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

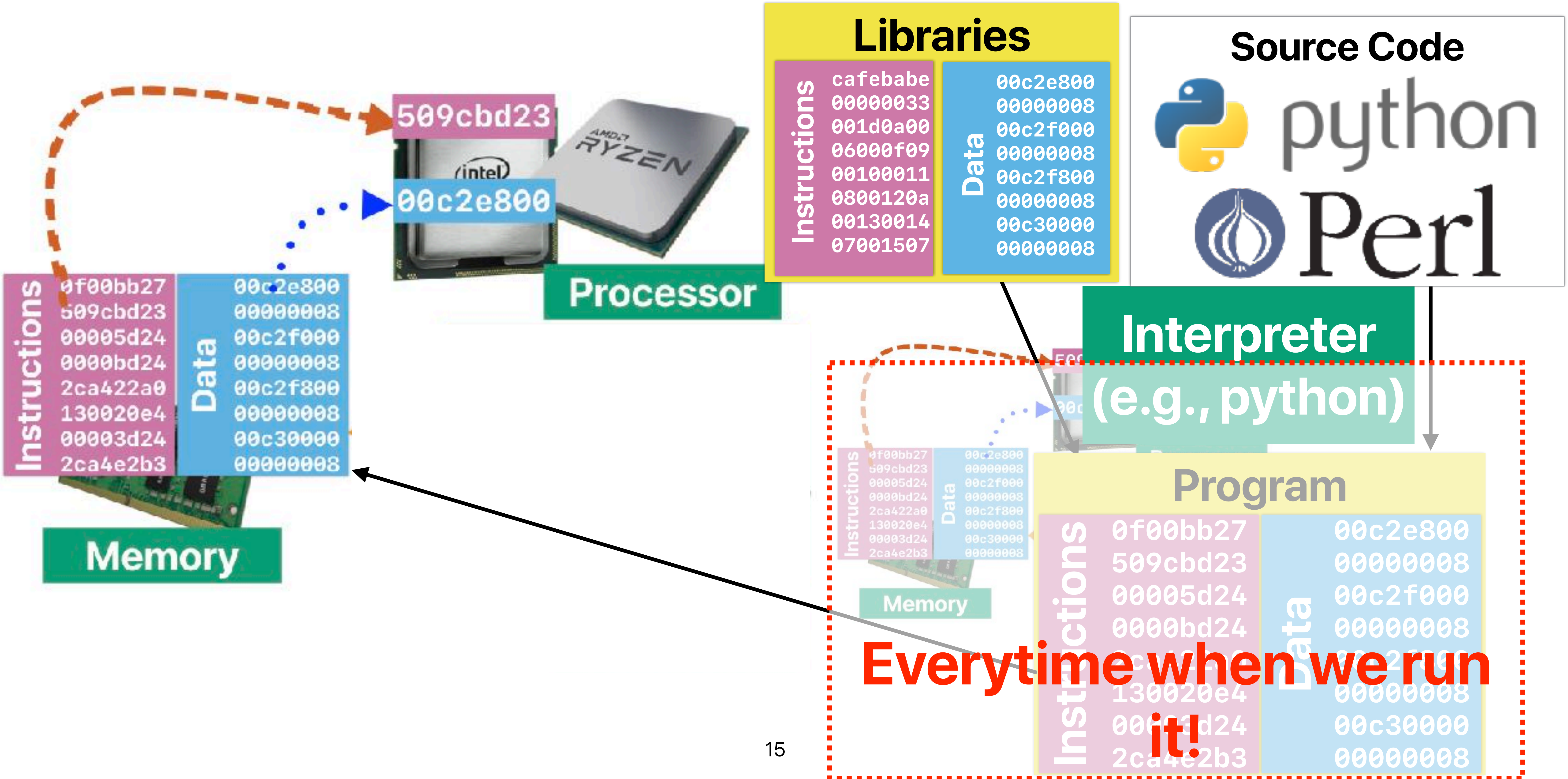
Recap: How my "C code" becomes a "program"



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



Recap: How my "Python code" becomes a "program"



How programming languages affect performance

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

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

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

Poll close in 1:30

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

Poll close in 1:30

1 — Instinct

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

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];
  }
}
```

Demo revisited — compiler optimization

- Compiler can reduce the instruction count, change CPI — with "limited scope"
- Compiler CANNOT help improving "crummy" source code

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

Compiler can never add this — only the programmer can!

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

How about "computational complexity"

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

These are unlikely to be true

Summary of CPU Performance Equation

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

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

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

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

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

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.

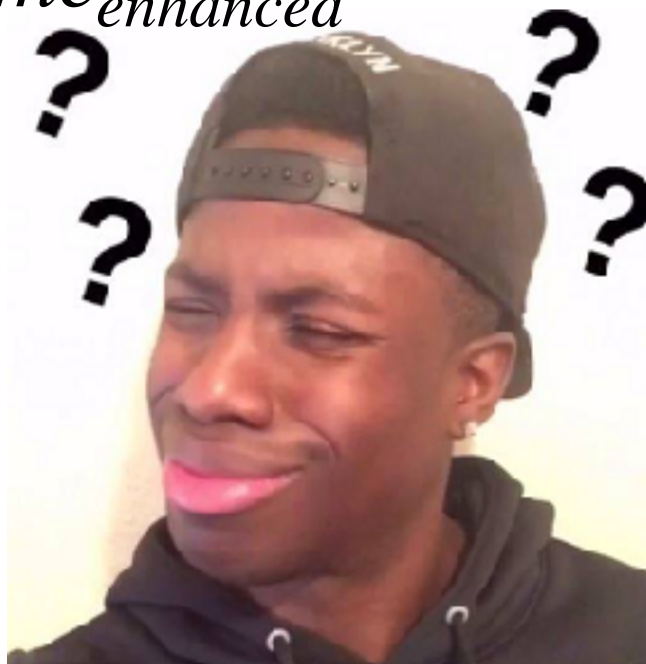
Amdahl's Law

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

f — The fraction of time in the original program

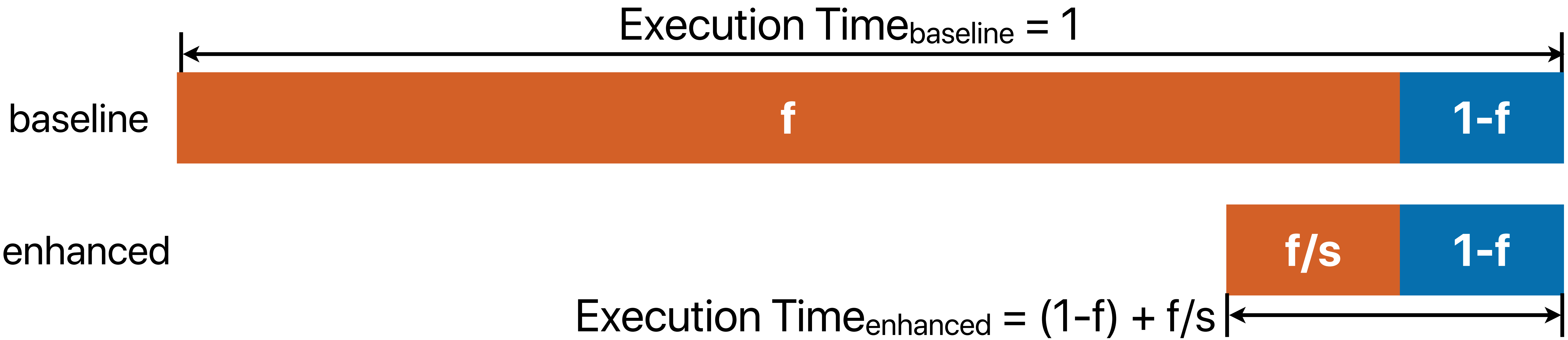
s — The speedup we can achieve on f

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}}$$



Amdahl's Law

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



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

Recap: Speedup

- Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.
  - If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What's the performance improvement after this change?

A. No change
B. 1.25
C. 1.5
D. 2
E. None of the above

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

$$ET_{baseline} = (5 \times 10^5) \times (20\% \times 6 + 80\% \times 1) \times \frac{1}{2 \times 10^{-9}} sec = 5^{-3}$$

$$ET_{enhanced} = (5 \times 10^5) \times (20\% \times 12 + 80\% \times 1) \times \frac{1}{4 \times 10^{-9}} sec = 4^{-3}$$

$$Speedup = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}}$$

$$= \frac{5}{4} = 1.25$$

Replay using Amdahl's Law

- Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.
  - If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What's the performance improvement after this change?

How much time in load/store? $500000 \times (0.2 \times 6) \times 0.5\ ns = 300000\ ns \rightarrow 60\,\%$

How much time in the rest? $500000 \times (0.8 \times 1) \times 0.5\ ns = 200000\ ns \rightarrow 40\,\%$

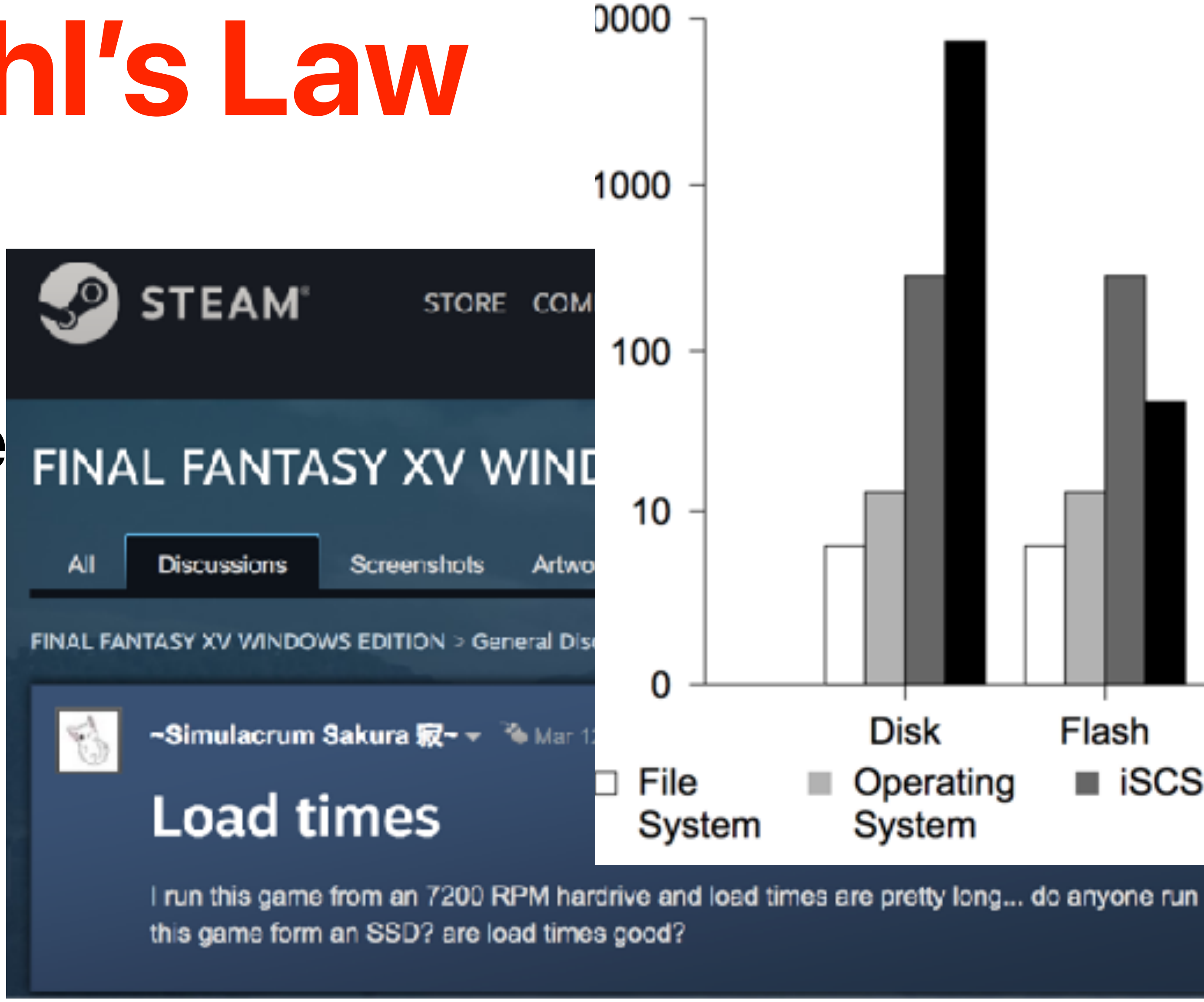
$$Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}$$

$$Speedup_{enhanced}(40\,\%, 2) = \frac{1}{(1 - 40\%) + \frac{40\,\%}{2}} = 1.25 \times$$

Poll close in 1:30

Practicing Amdahl's Law

- 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. By how much can we speed up the map loading process?
    A. ~7x
    B. ~10x
    C. ~17x
    D. ~29x
    E. ~100x

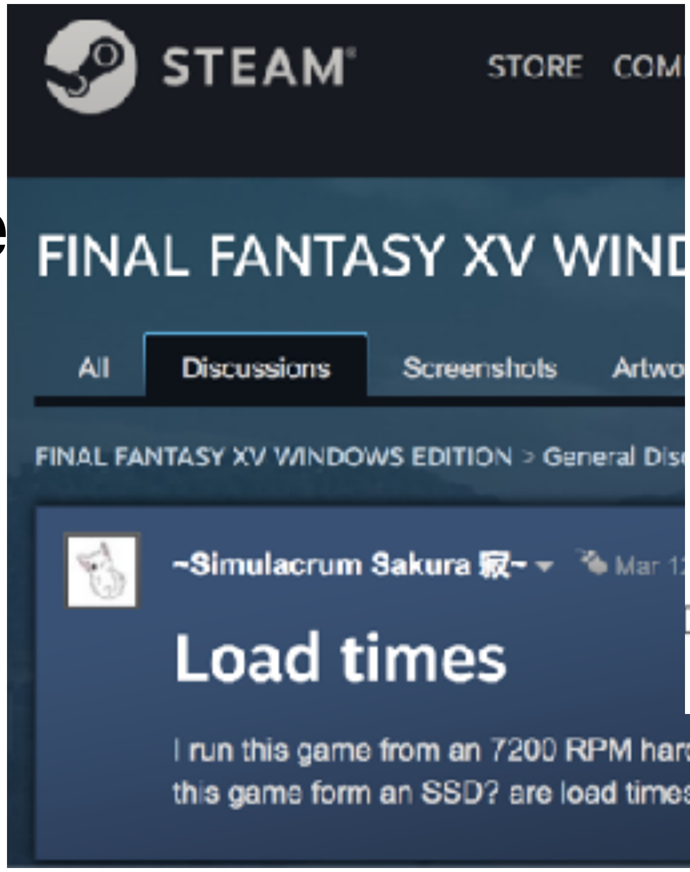


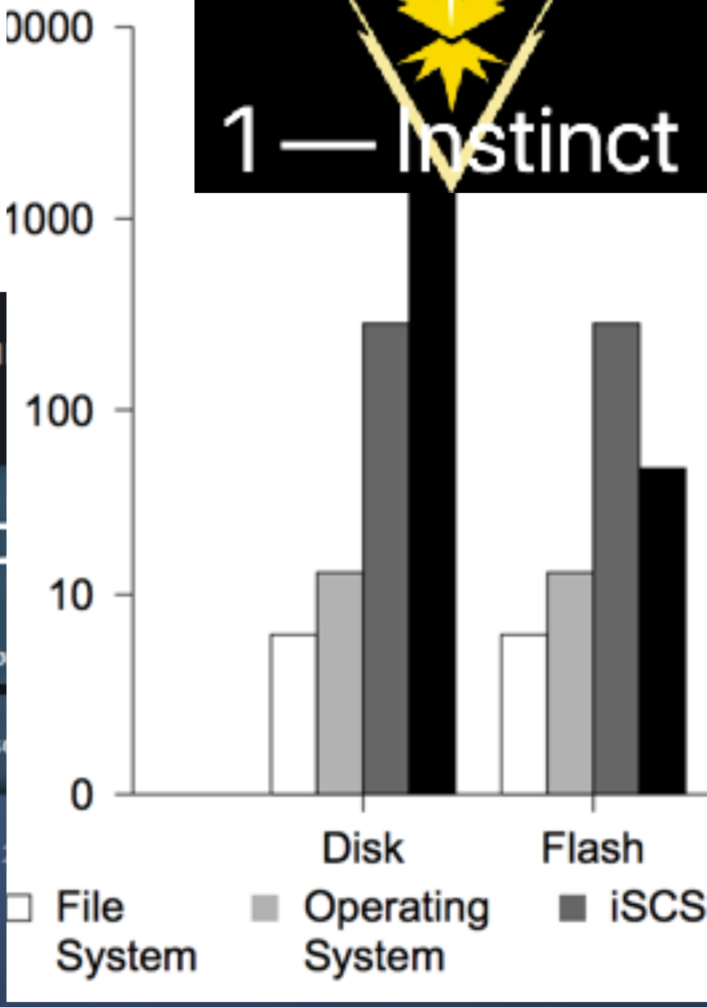
Poll close in 1:30

Practicing Amdahl's Law

1 — Instinct

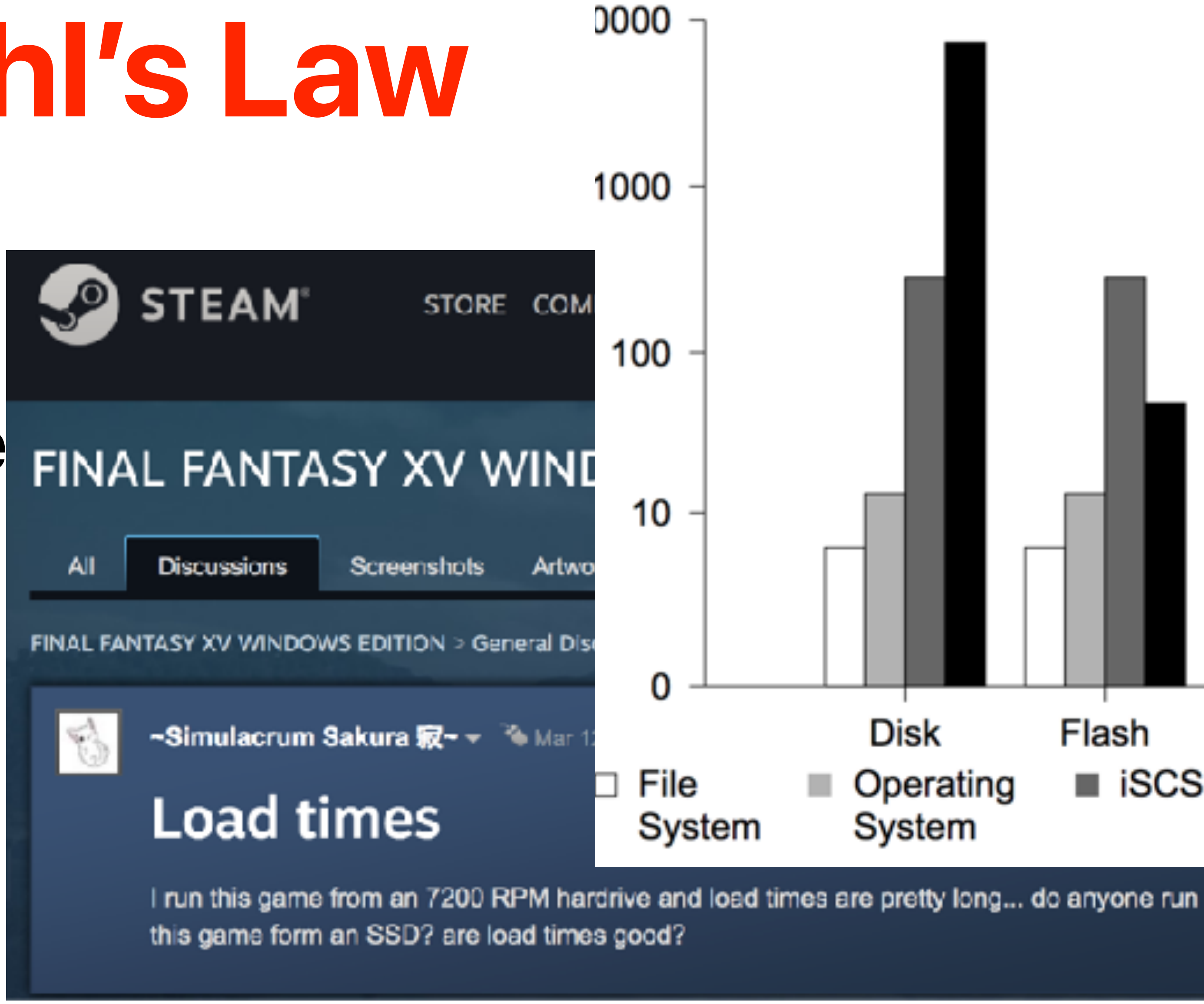
- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed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





Practicing Amdahl's Law

- 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. By how much can we speed up the map loading process?
    A. ~7x
    B. ~10x
    C. ~17x
    D. ~29x
    E. ~100x



$$Speedup_{enhanced}(95\%, 100) = \frac{1}{(1-95\%) + \frac{95\%}{100}} = 16.81\times$$

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:

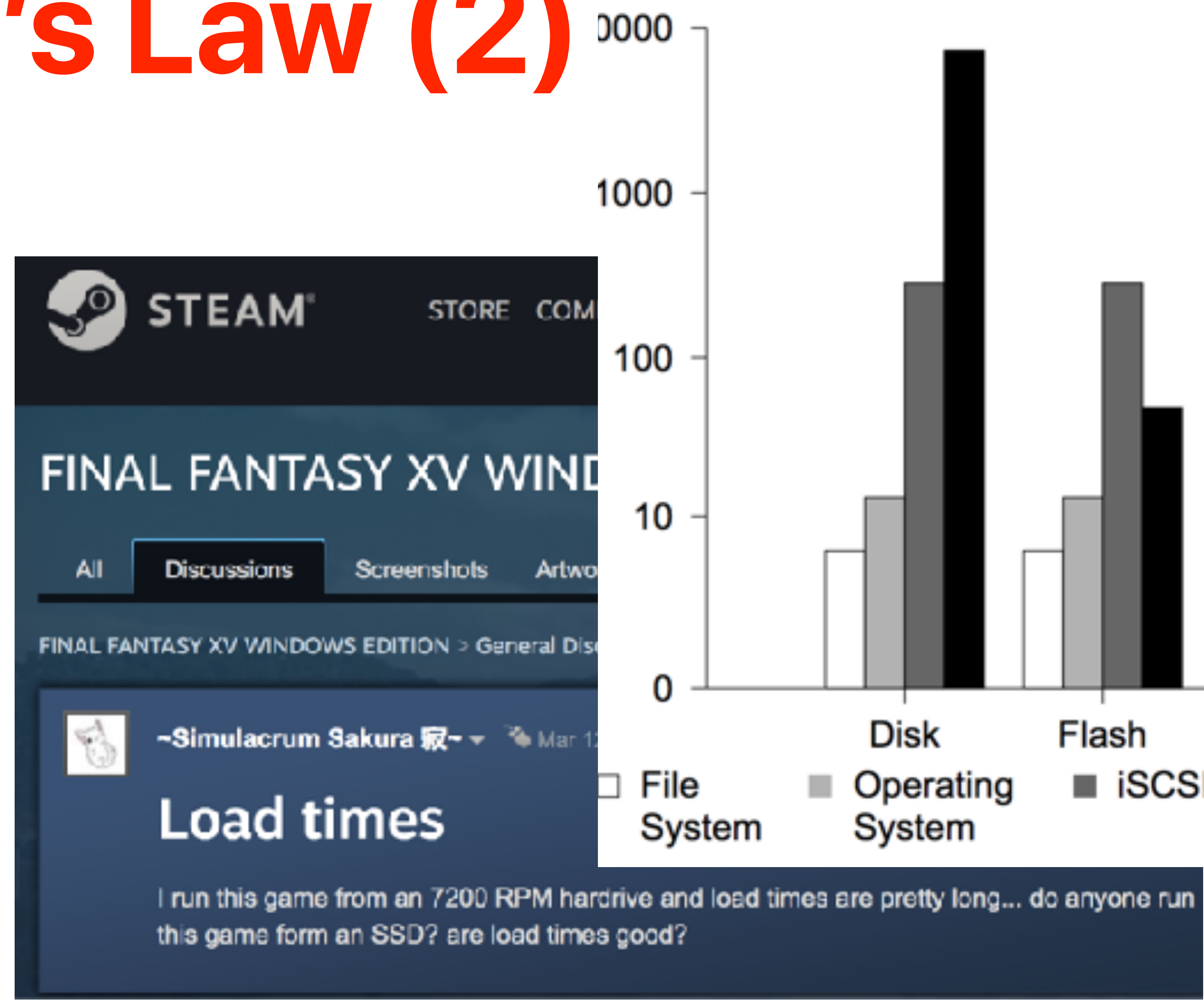
$$Speedup_{enhanced}(f_{Opt1}, f_{Opt2}, s_{Opt1}, s_{Opt2}) = \frac{1}{(1 - f_{Opt1} - f_{Opt2}) + \frac{f\_Opt1}{s\_Opt1} + \frac{f\_Opt2}{s\_Opt2}}$$

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

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

Practicing Amdahl's Law (2)

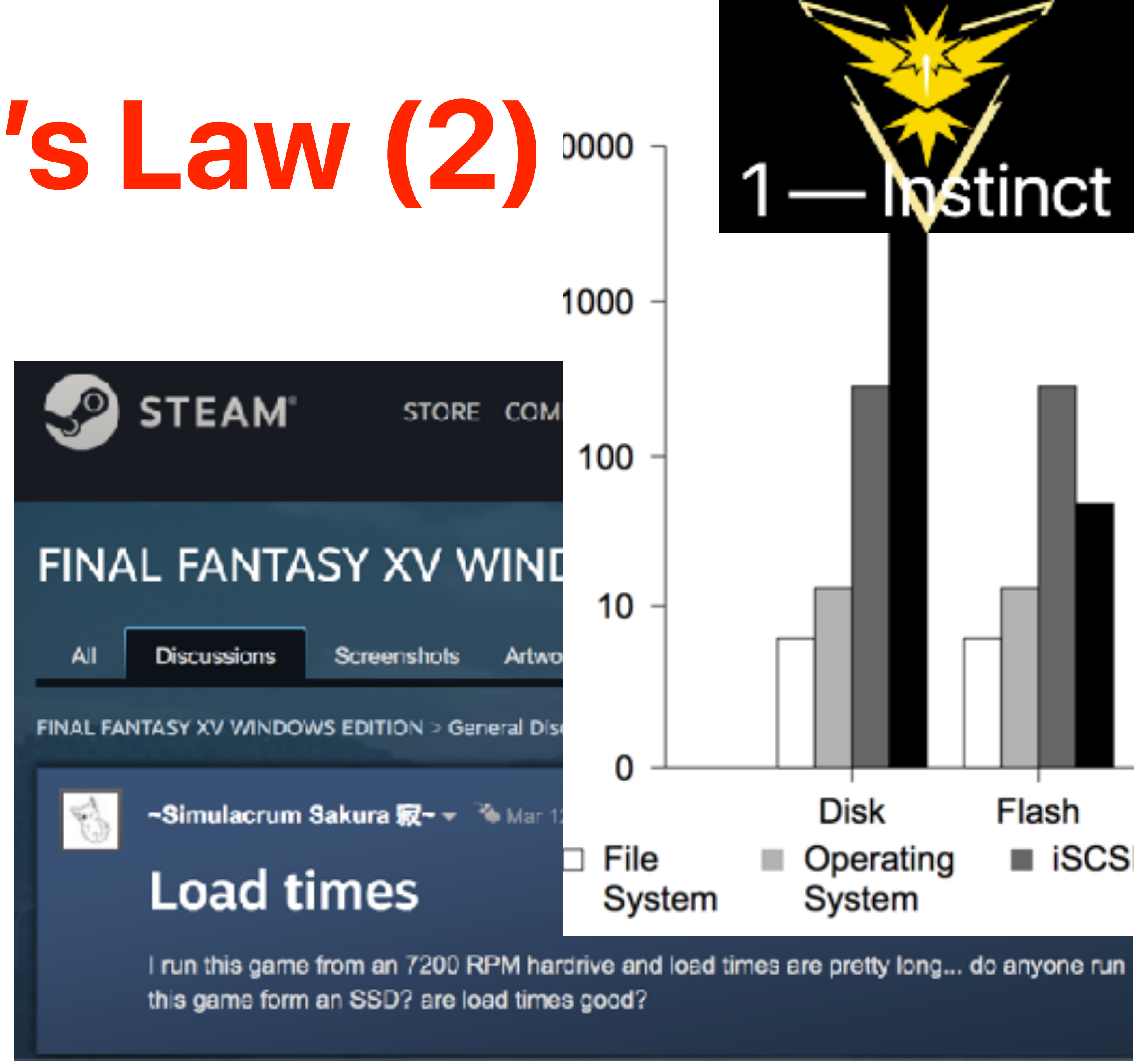
- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to 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



Poll close in 1:30

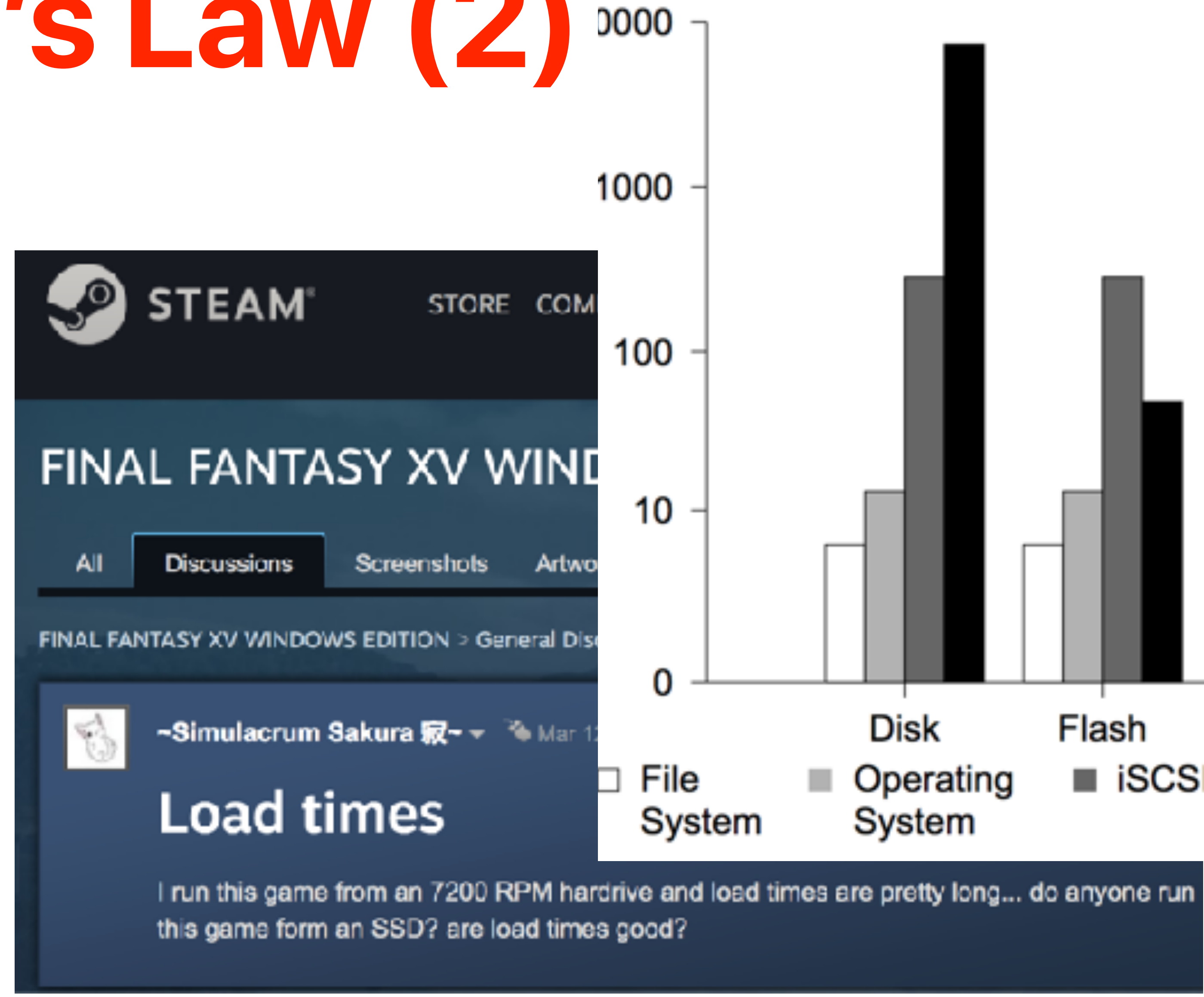
Practicing Amdahl's Law (2)

- 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)

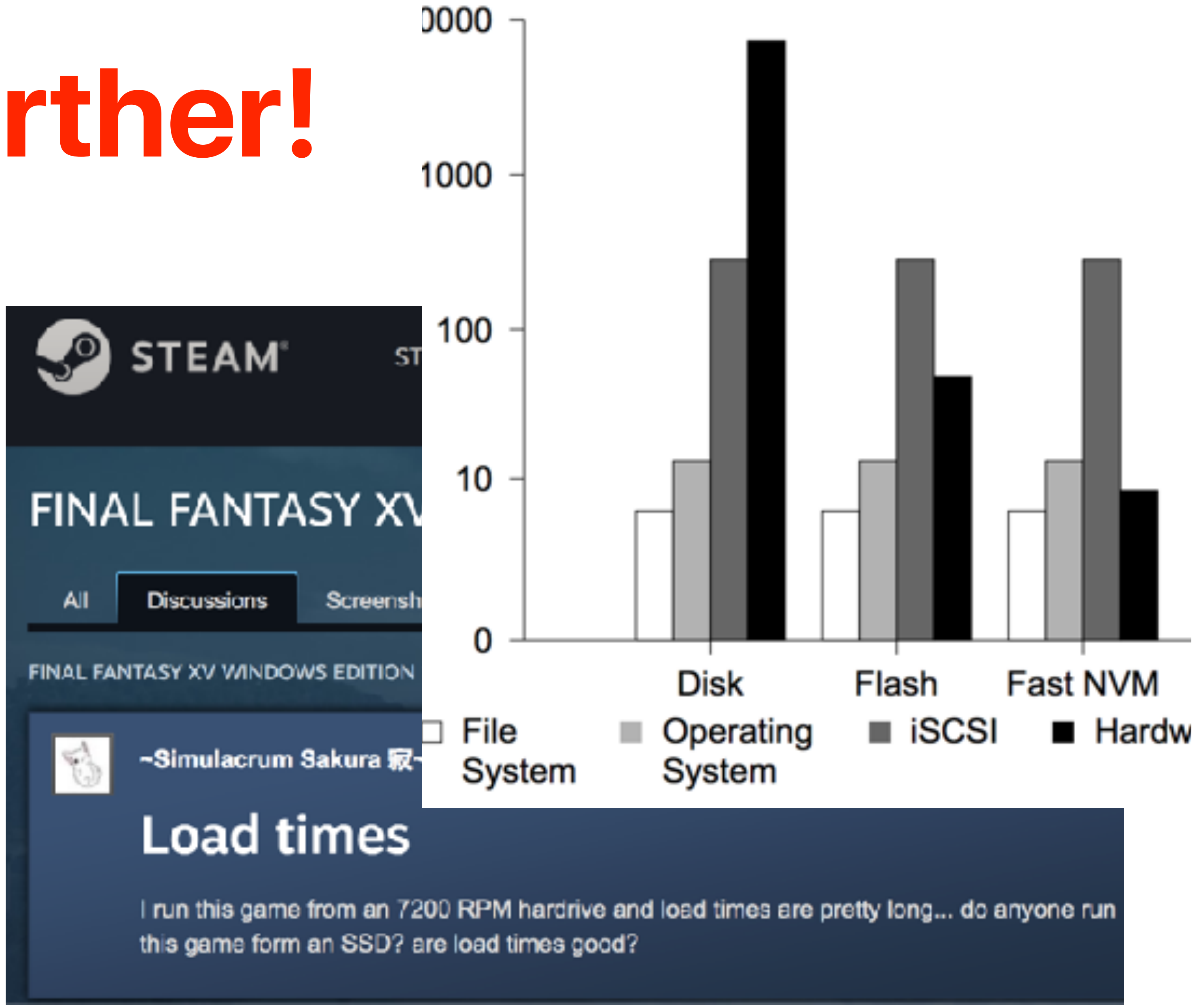
- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to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



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

Speedup further!

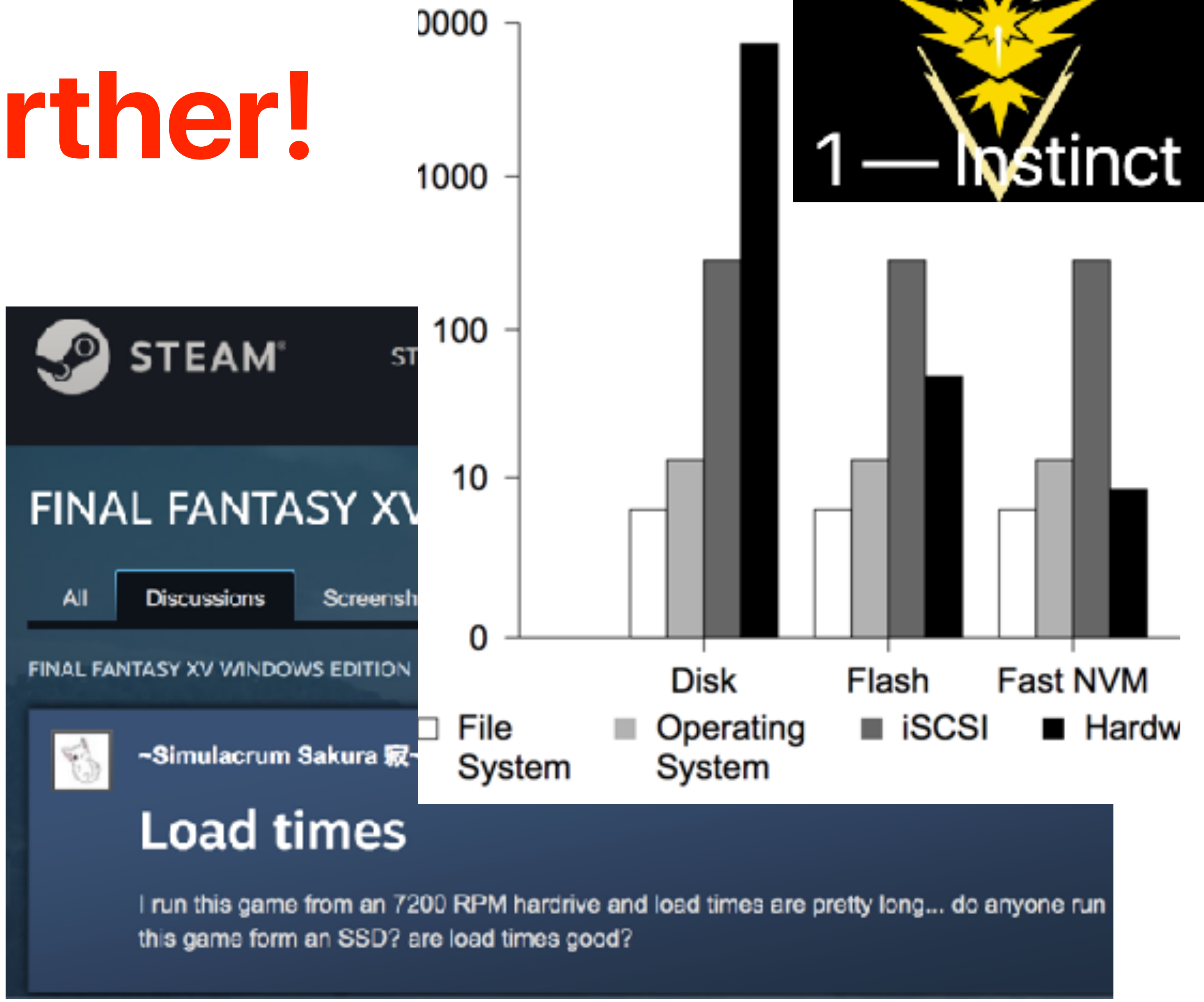
- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?
    A. ~5x
    B. ~10x
    C. ~20x
    D. ~100x
    E. None of the above



Poll close in 1:30

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 we want to adopt a new memory technology to replace 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



1 — Instinct

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 we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?

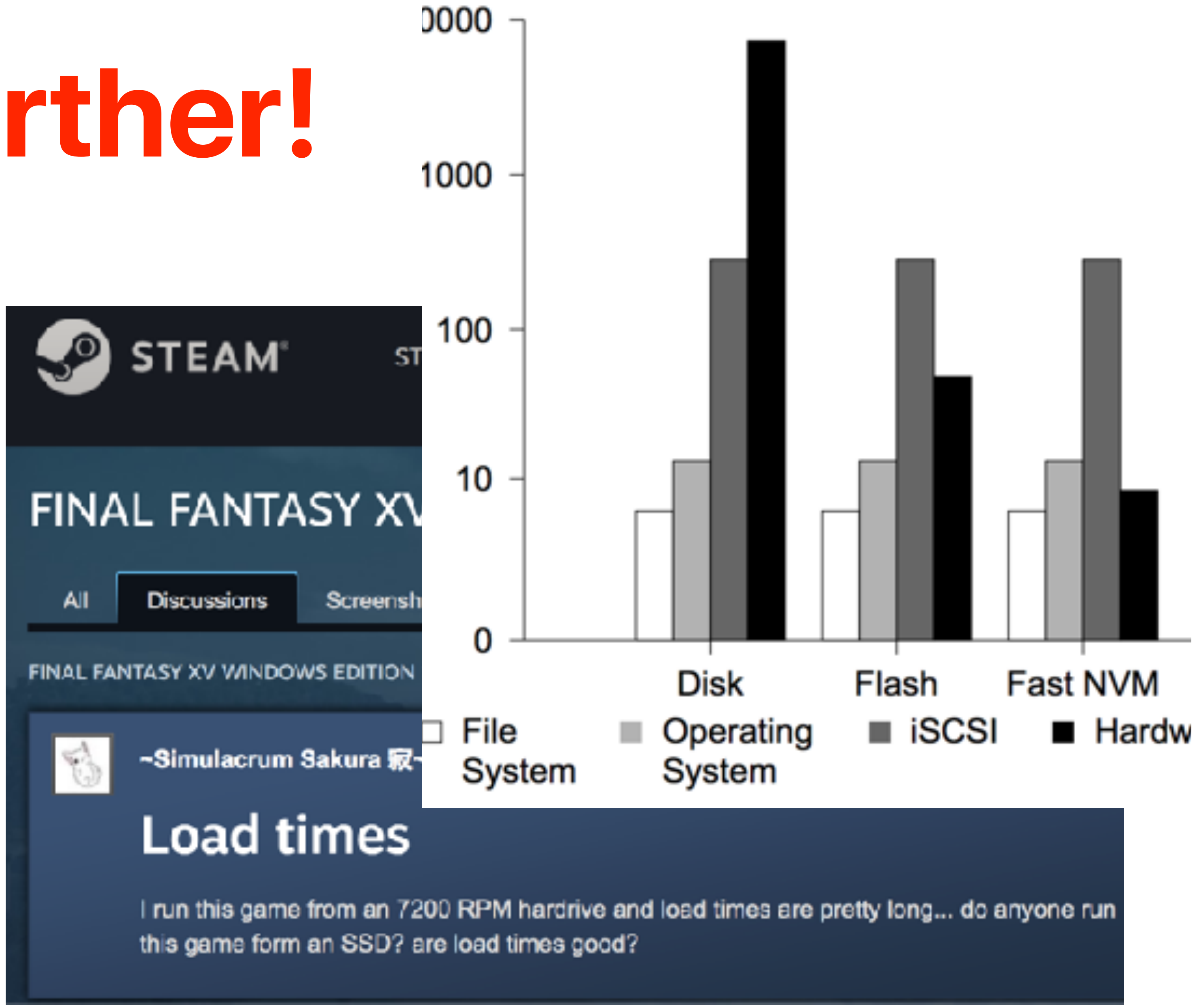
  A. ~5x

  B. ~10x

  C. ~20x

  D. ~100x

  E. None of the above



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

$$x = 0.47$$

Does this make sense?

Amdahl's Law Corollary #1

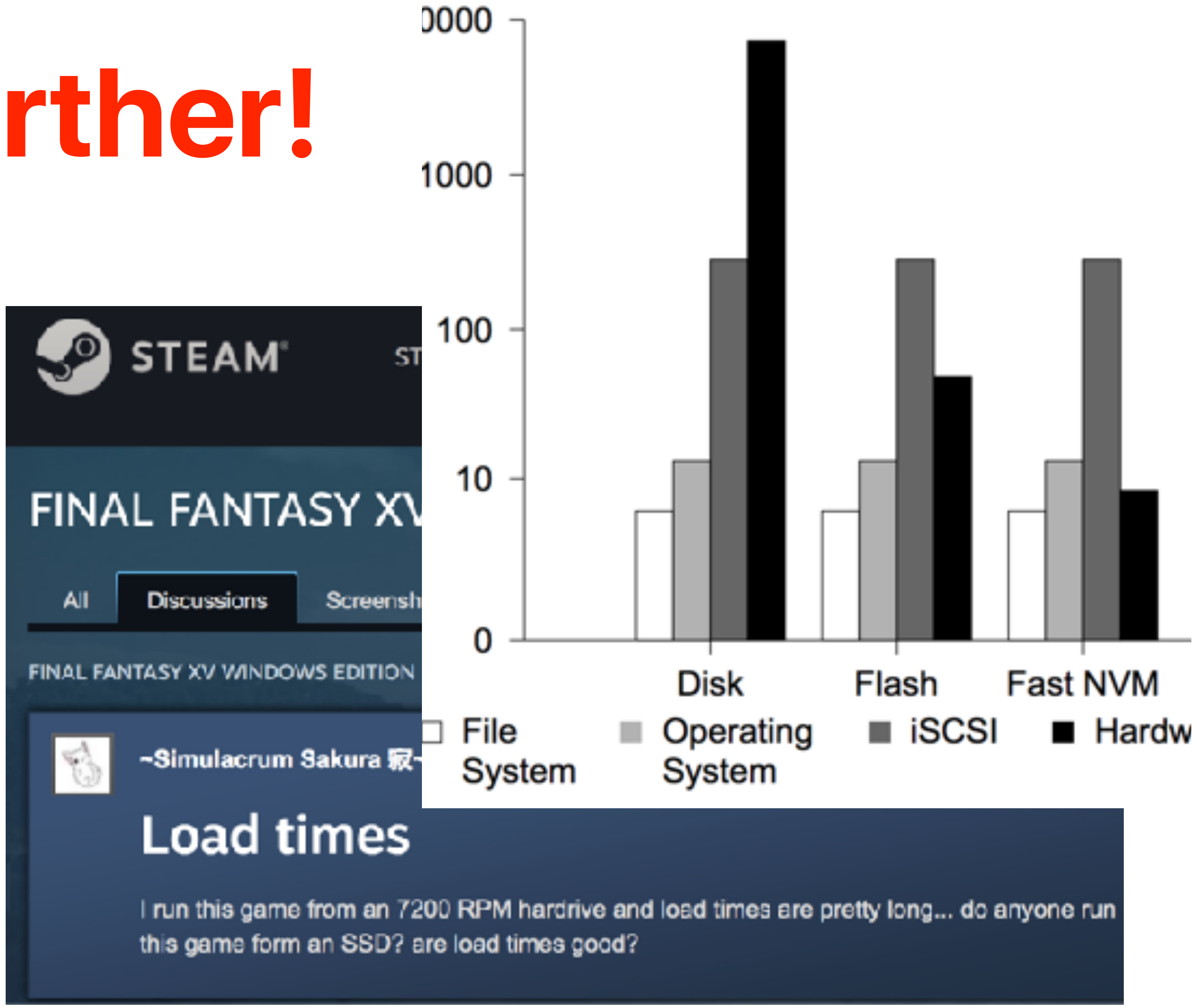
- The maximum speedup is bounded by

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

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

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 we want to adopt a new memory technology to replace 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



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

2x is not possible

Corollary #1 on Multiple Optimizations

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

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

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

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

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

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

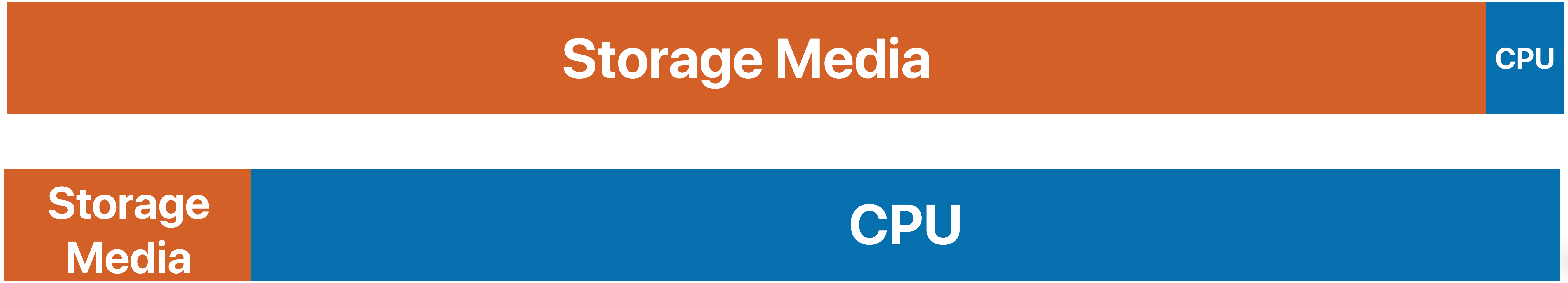
Corollary #2 — make the common case fast!

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

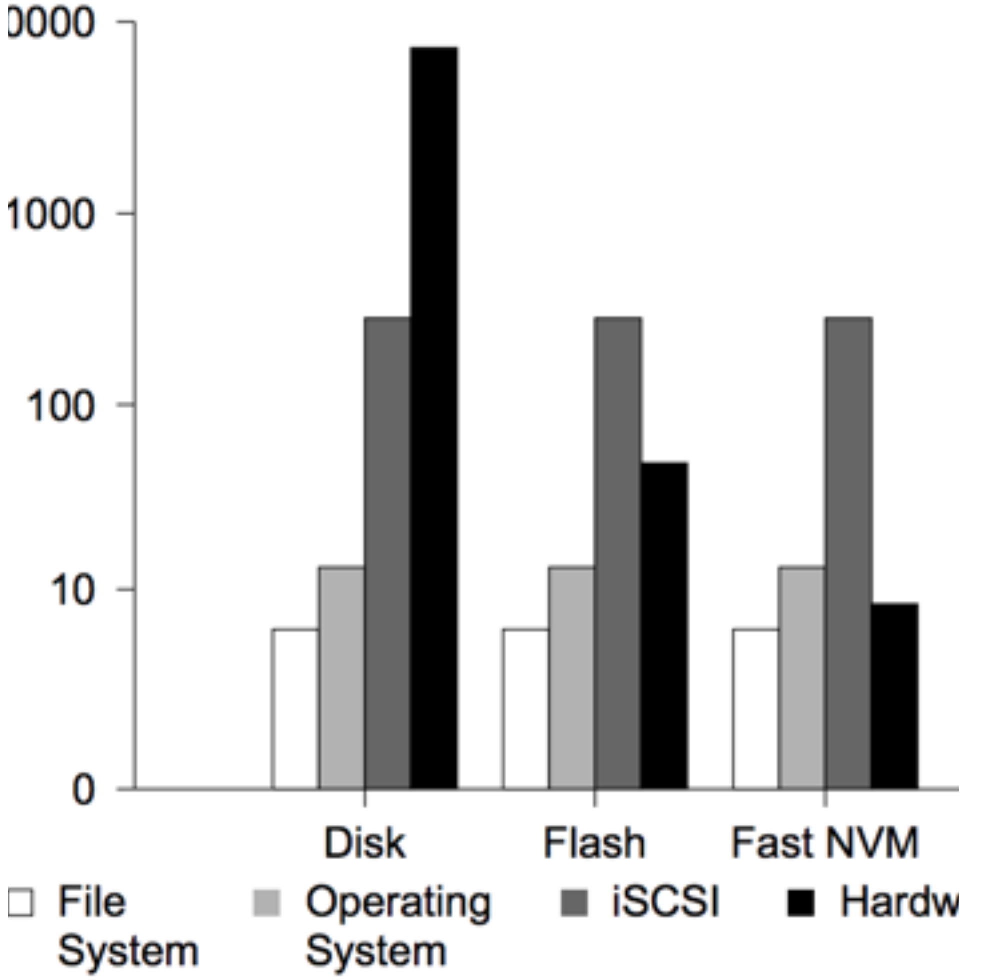
Identify the most time consuming part

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

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



Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010.

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

Announcement

- Reading quiz due next Monday before the lecture
  - We will drop two of your least performing reading quizzes
  - You have two shots, both unlimited time
- Check our website for slides, iLearn for quizzes/assignments, piazza for discussions
  - Assignment #1 due 10/19
  - Assignments SHOULD BE done individually
  - We will drop your least performing assignment as well
  - Attendance counts as one assignment

Computer
Science &
Engineering

203

つづく