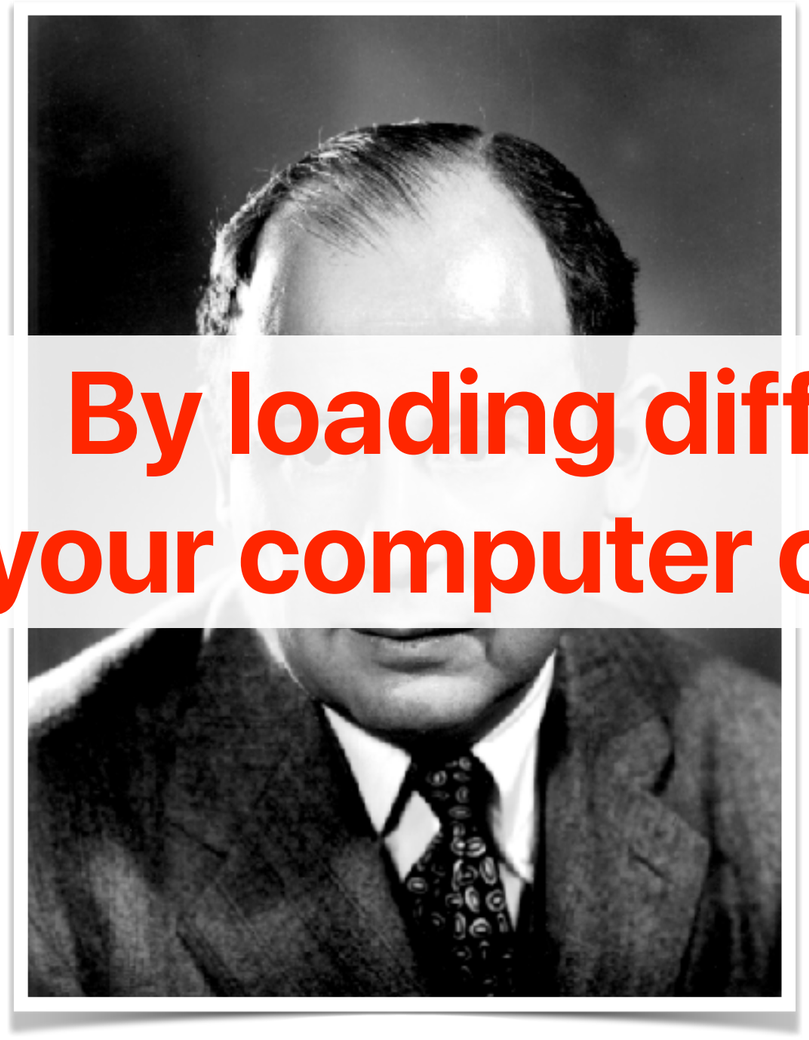
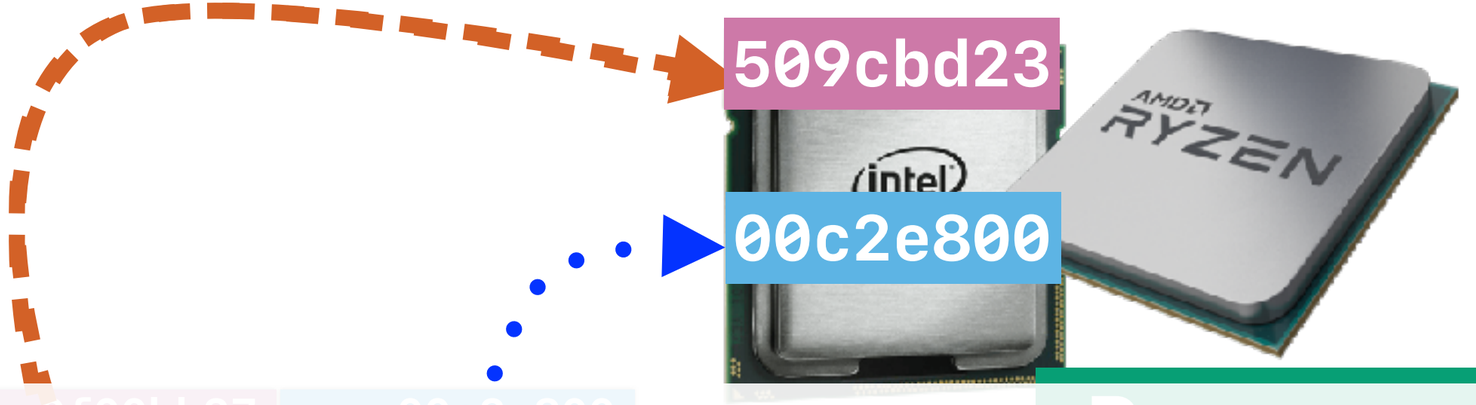
Performance (II): Amdahl's Law and its Implications

Hung-Wei Tseng

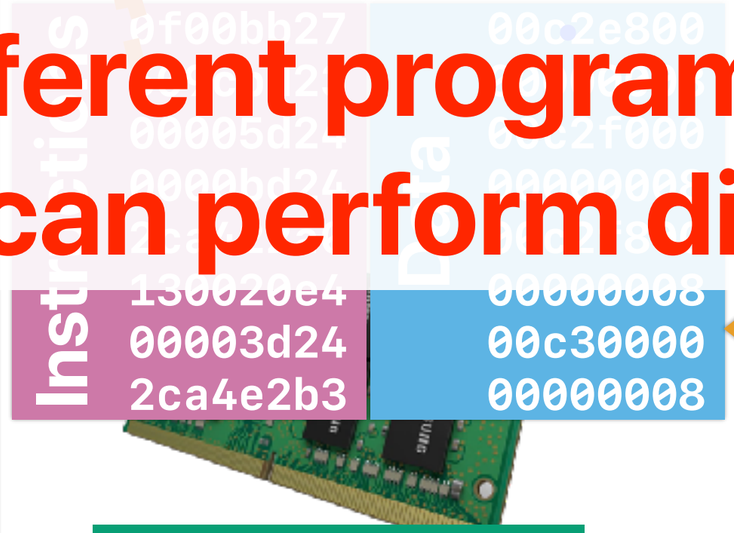
Recap: von Neumman Architecture





Processor

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

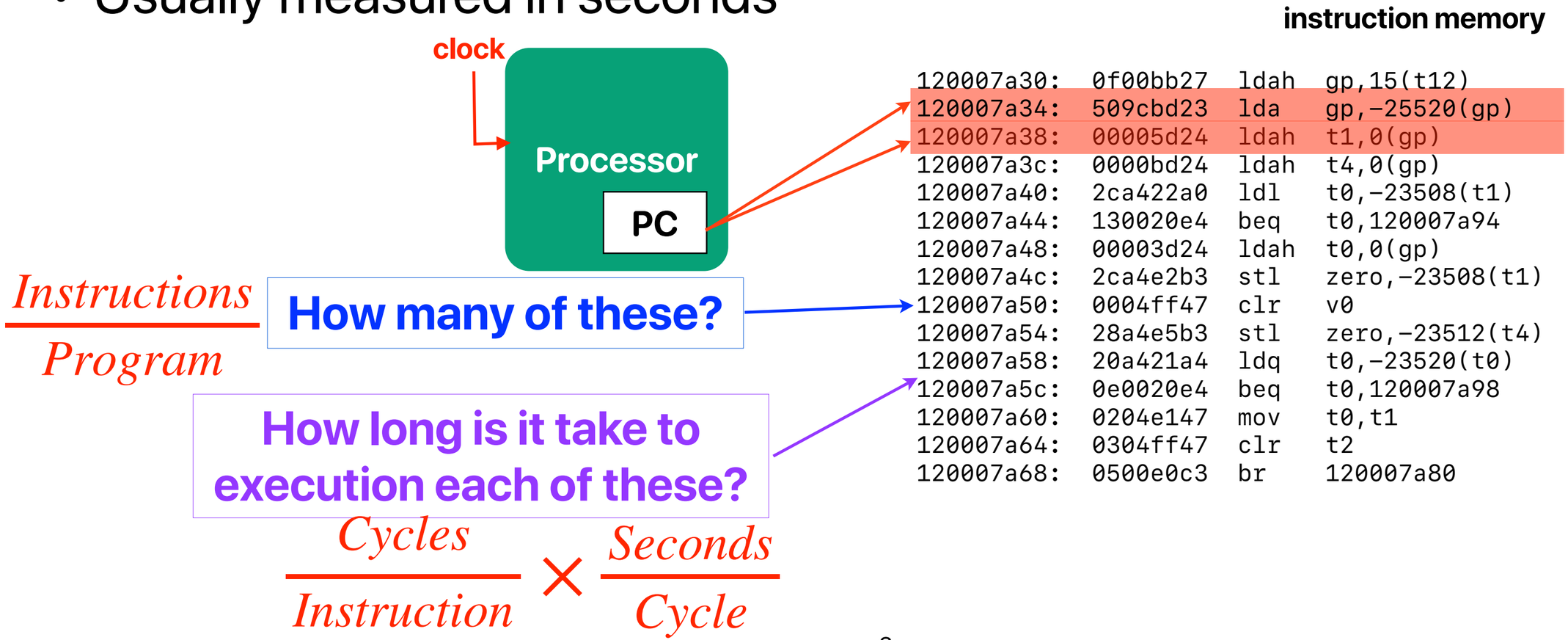


Memory

Storage

Recap: Execution Time

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



Recap: CPU Performance Equation

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

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

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

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

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

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

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

The only "authentic definition" of speedup in Computer Architecture

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

Outline

- What affects each factor in “performance” (cont.)
- Amdahl’s law

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

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

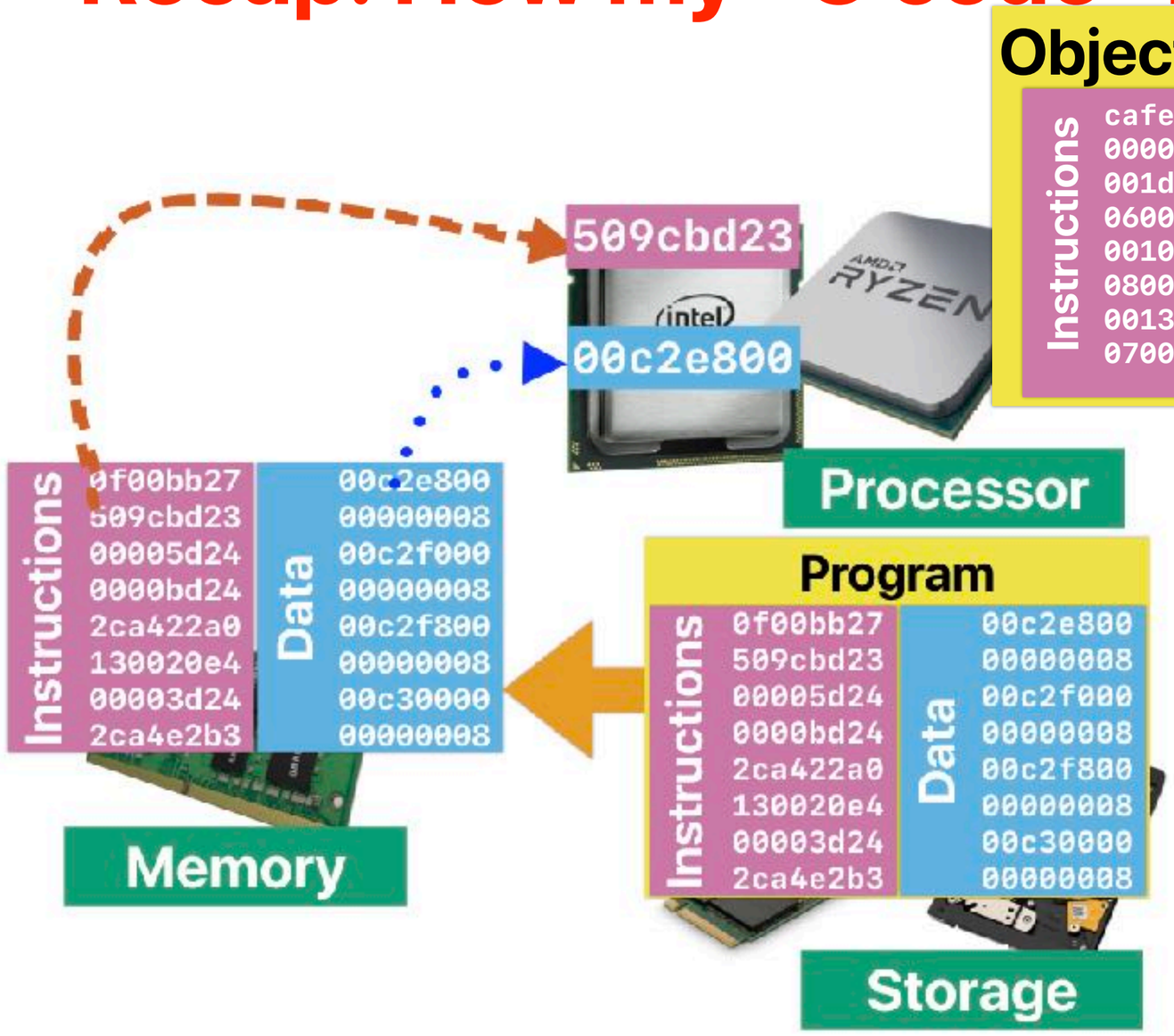
- How many instructions are there in “Hello, world!”

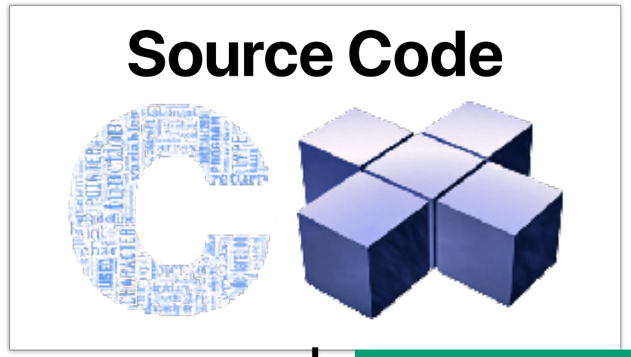
	Instruction count	LOC	Ranking
C	600k	6	1
C++	3M	6	2
Java	~210M	8	5
Perl	10M	4	3
Python	~30M	1	4

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

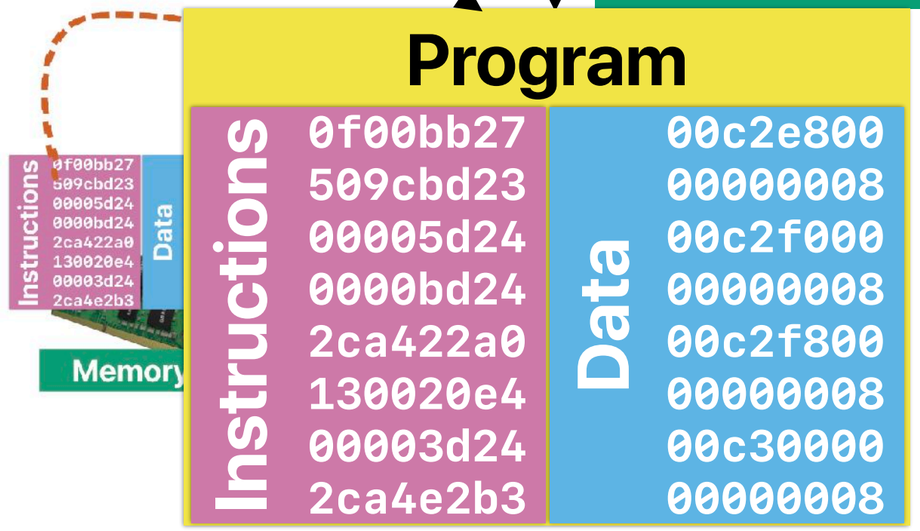
Recap: How my "C code" becomes a "program"





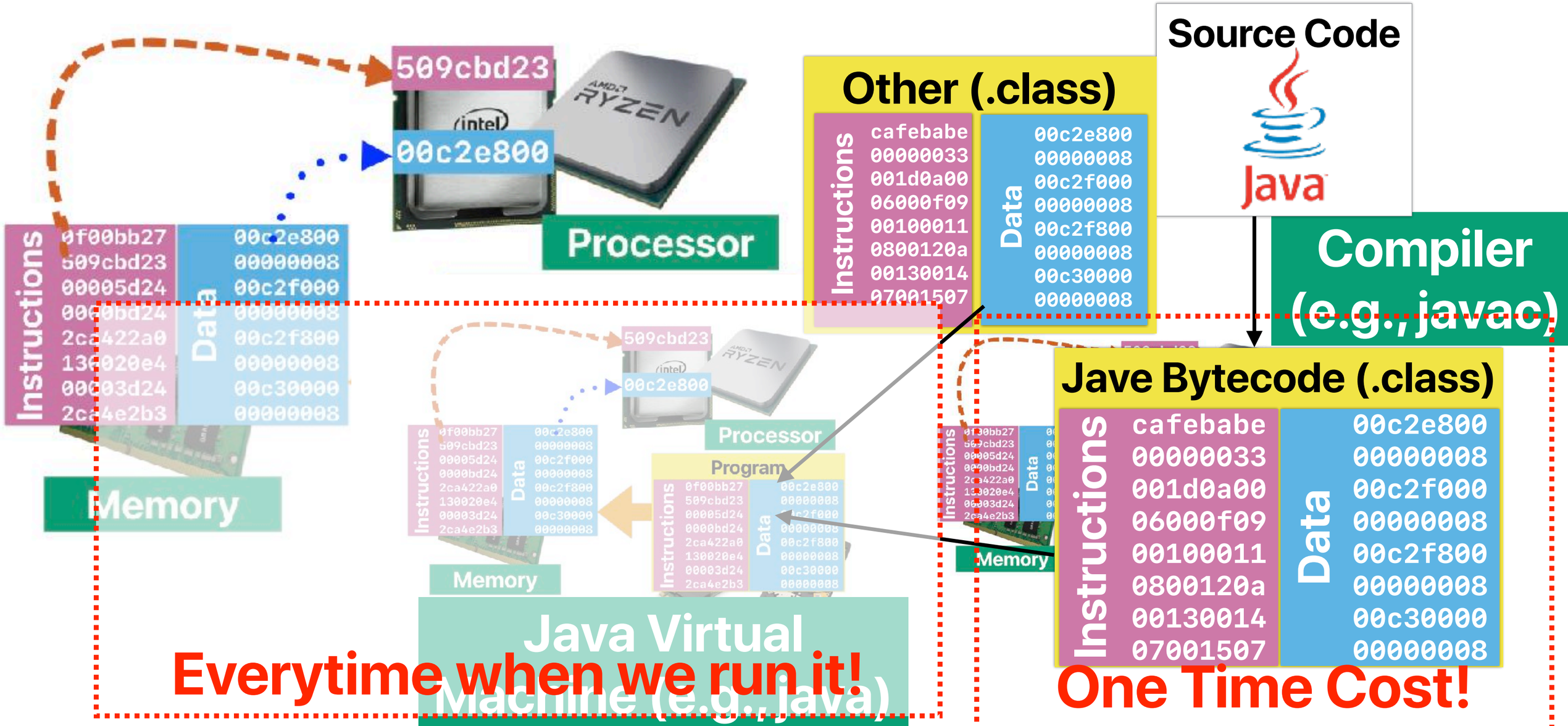
Linker

Compiler (e.g., gcc)

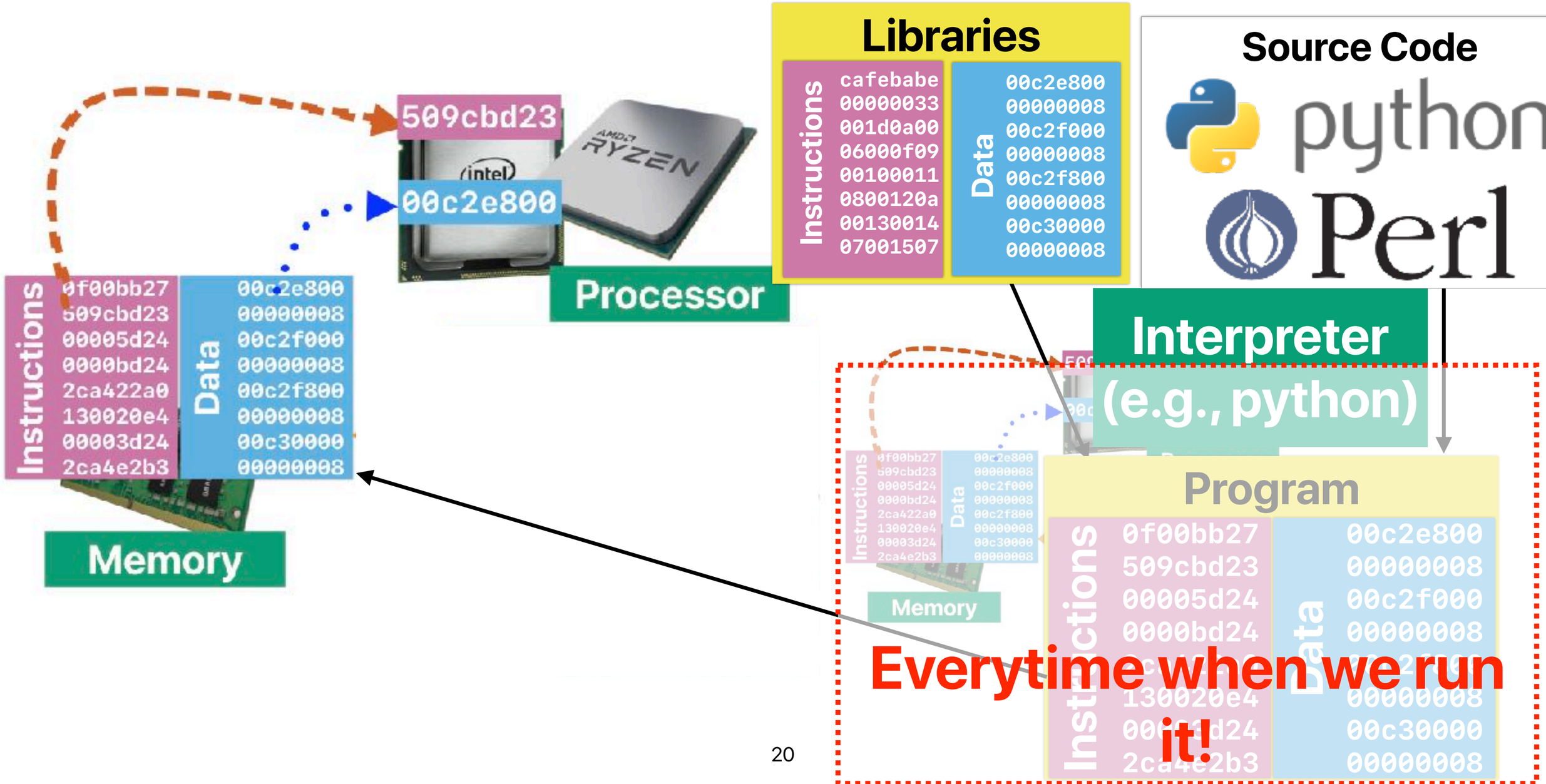


One Time Cost!

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

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.

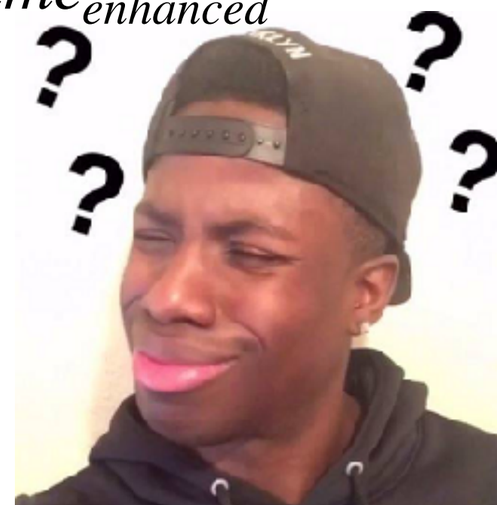
Amdahl's Law

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

f — The fraction of time in the original program

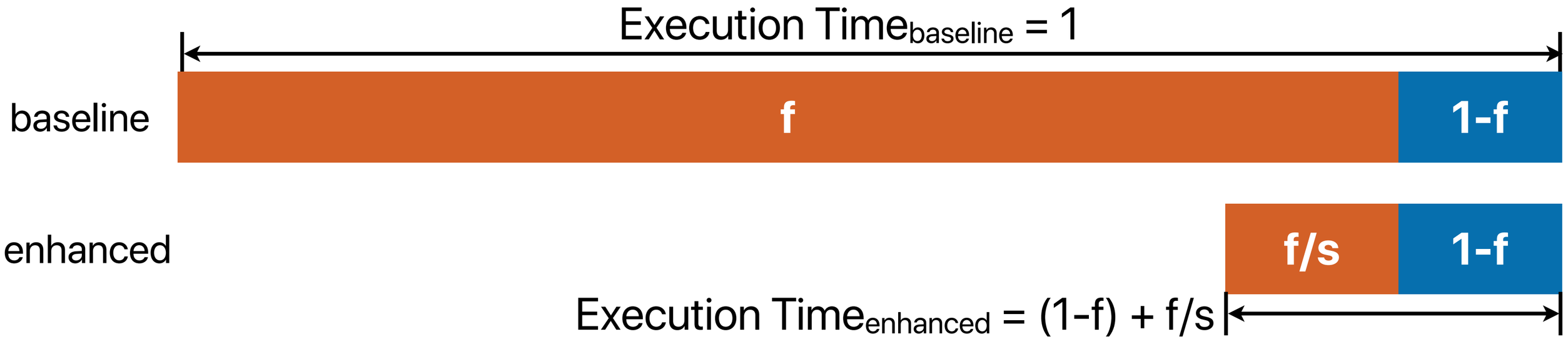
s — The speedup we can achieve on f

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



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? $500000 \times (0.8 \times 1) \times 0.5\ ns = 200000\ ns \rightarrow 40\,\%$

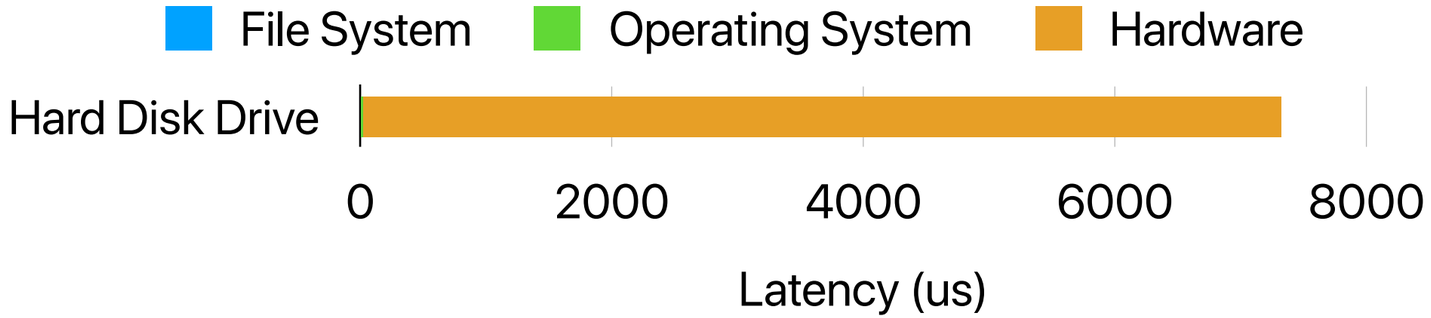
$$Speedup_{enhanced}(f, s) = \frac{1}{(1-f) + \frac{f}{s}}$$

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

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

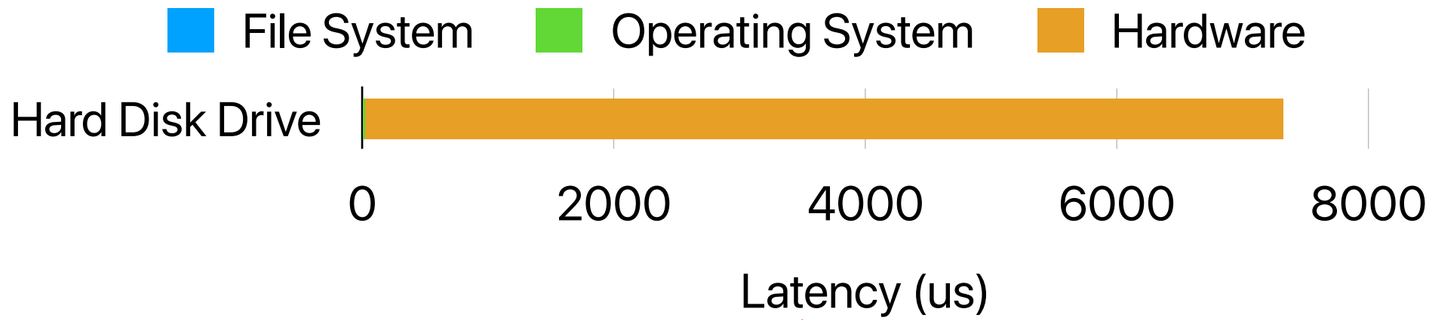
Practicing Amdahl's Law

- 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



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:

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


WHITE HO
WASHINGTON
Q & A
NBC NEWS
NOW

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
  - The commentary question in Quiz #2 needs manual grading — don't be panic
- Check our website for slides, eLearn for quizzes/assignments, piazza for discussions
- Youtube channel for lecture recordings: https://www.youtube.com/c/ProfUsagi/playlists

Computer
Science &
Engineering

203

つづく