

Memory Hierarchy: Basics

Hung-Wei Tseng



Disney·PIXAR INSIDE OUT

GET DISNEY+

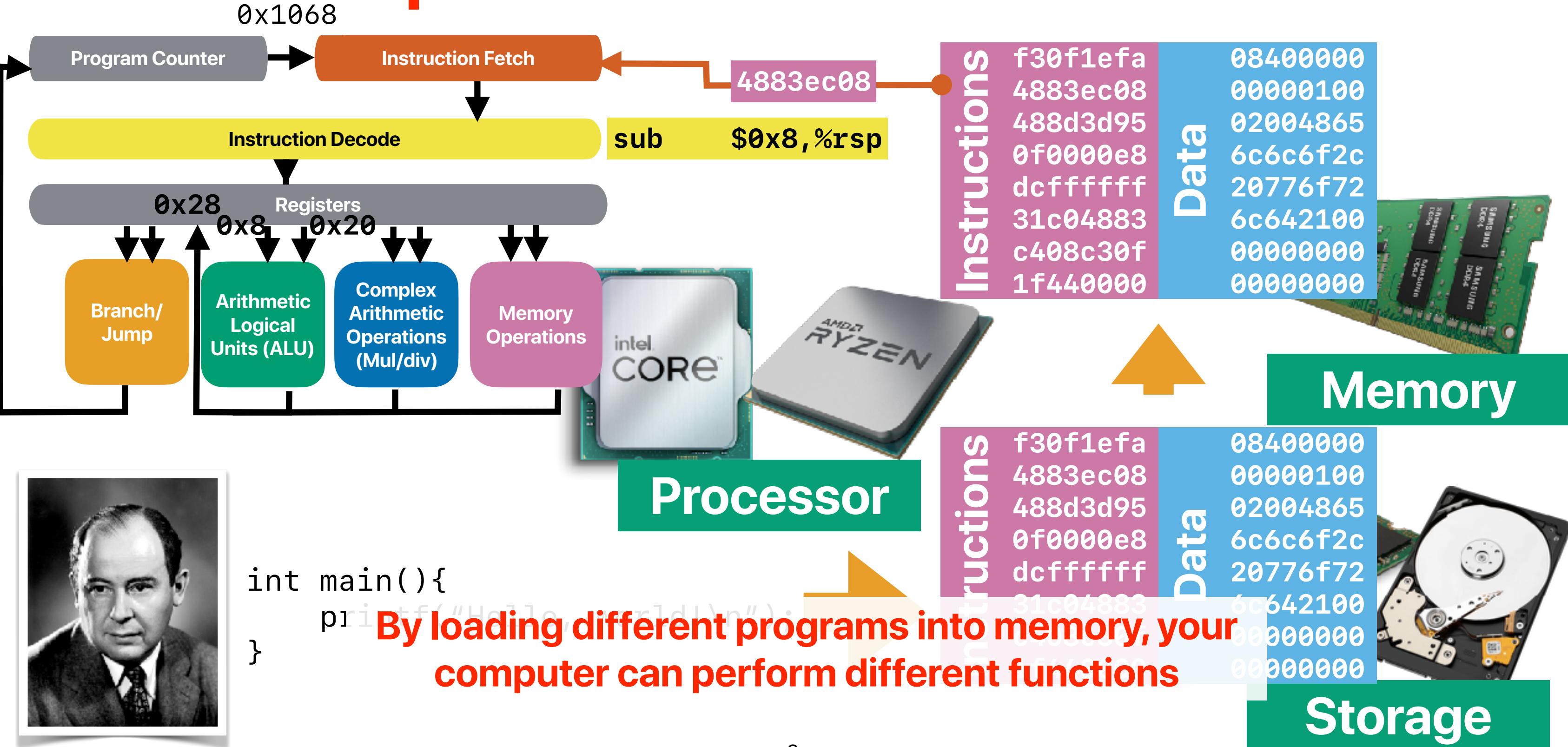
► TRAILER

PG 2015 • 1h 35m • Coming of age, Family, Animation

When 11-year-old Riley moves to a new city, her Emotions team up to help her through the transition. Joy, Fear, Anger, Disgust and Sadness work together, but when Joy and Sadness get lost, they must journey through unfamiliar places to get back home.



Recap: von Neumann architecture



**What percentage of “English words” from
the dictionary do you think someone has
to know for their everyday life?**

federalreserve.gov/releases/z1/datavizdfa/di...

crn.ccm/news/security/18821726/microsofts-ceo-80-20-rule-applies-to-bugs-not-just-features

Board of Governors of the Federal Reserve System

Sections

Wealth by wealth percentile group

Trillions of Dollars

The chart displays the total wealth in Trillions of Dollars over time, broken down into five percentile groups. The legend indicates:

- Top 0.1%: \$20.66 T
- 99-99.9%: \$25.52 T
- 90-99%: \$55.44 T
- 50-90%: \$46.28 T
- Bottom 50%: \$3.78 T

Total: \$151.68 T

| Percentile Group | Value (Trillions of Dollars) |
|------------------|------------------------------|
| Top 0.1% | \$20.66 T |
| 99-99.9% | \$25.52 T |
| 90-99% | \$55.44 T |
| 50-90% | \$46.28 T |
| Bottom 50% | \$3.78 T |
| Total | \$151.68 T |

2010 2012 2014 2016 2018 2020 2022 2024

Top 10% own 67% of the wealth in the U.S.

Microsoft's CEO: 80-20 Rule Applies To Bugs, Not Just Features

BY PAULA ROONEY ▶ OCTOBER 03, 2002, 03:56 PM EDT

Microsoft

One common adage in the IT industry is that 80 percent of all end users generally use only 20 percent of a software application's features.

In recent months, Microsoft has learned that 80 percent of the errors and crashes in Windows and Office are caused by 20 percent of the entire pool of bugs detected, and that more than 50 percent of the headaches derive from a mere 1 percent of all flawed code.

In an e-mail update sent out broadly to enterprise customers on Oct. 2, Microsoft CEO Steve Ballmer highlighted critical progress being made on the company's Trustworthy Computing initiative, an effort to expand by the end of next January to improve its reputation in reliability and security arenas. For one thing, there will be faster bug-fixing as a result of an error-reporting facility embedded in Office and Windows. And that error-reporting tool will be part of the forthcoming Windows Server 2003.

80% of users use only 20% of features

https://en.wikipedia.org/wiki/Pareto_principle

5

 AI Overview[Learn more](#) 

Native English speakers use around 1,200–3,000 words in daily life, which account for about 80% of their communication. These words include common verbs like "eat," "sleep," "work," "talk," and "walk," as well as pronouns like "I," "you," "he," and "she," and basic nouns like "house," "car," "food," and "water". The most commonly used words in English are "the," "be," and "to". 

 Eton Institute · LinkedIn · 1y

How many words are in the English language (95/5 RULE)?

How Many Words Do Native Speakers Use In Daily Life? Native speakers of...

 GeeksforGeeks

Daily Used English Words: List of 100+ Most Common Words

May 6, 2024 — Words used in daily life often include common verbs like...

Show more 

With 2,500 to 3,000 words, you can understand 90% of everyday English conversations, English newspaper and magazine articles, and English used in the workplace.



ef.edu

<http://www.ef.edu/english-vocabulary/1000-3000-w...>

3000 most common words in English | EF English Charts

 AI Overview

The number of words in an English dictionary depends on the dictionary, whether it includes obsolete words, combinations, and phrases:

Oxford English Dictionary

The second edition of this 20-volume dictionary includes 171,476 words in current use, 47,156 obsolete words, and around 9,500 derivative words. It also includes 169,000 combinations and derivatives, and over 600,000 word forms. The dictionary is updated annually to include new and new meanings for existing words.

Webster's Third New International Dictionary

This dictionary, along with its 1993 Addenda Section, includes around 470,000 entries. 

You only need to know 2% English words
to understand 90% of conversations

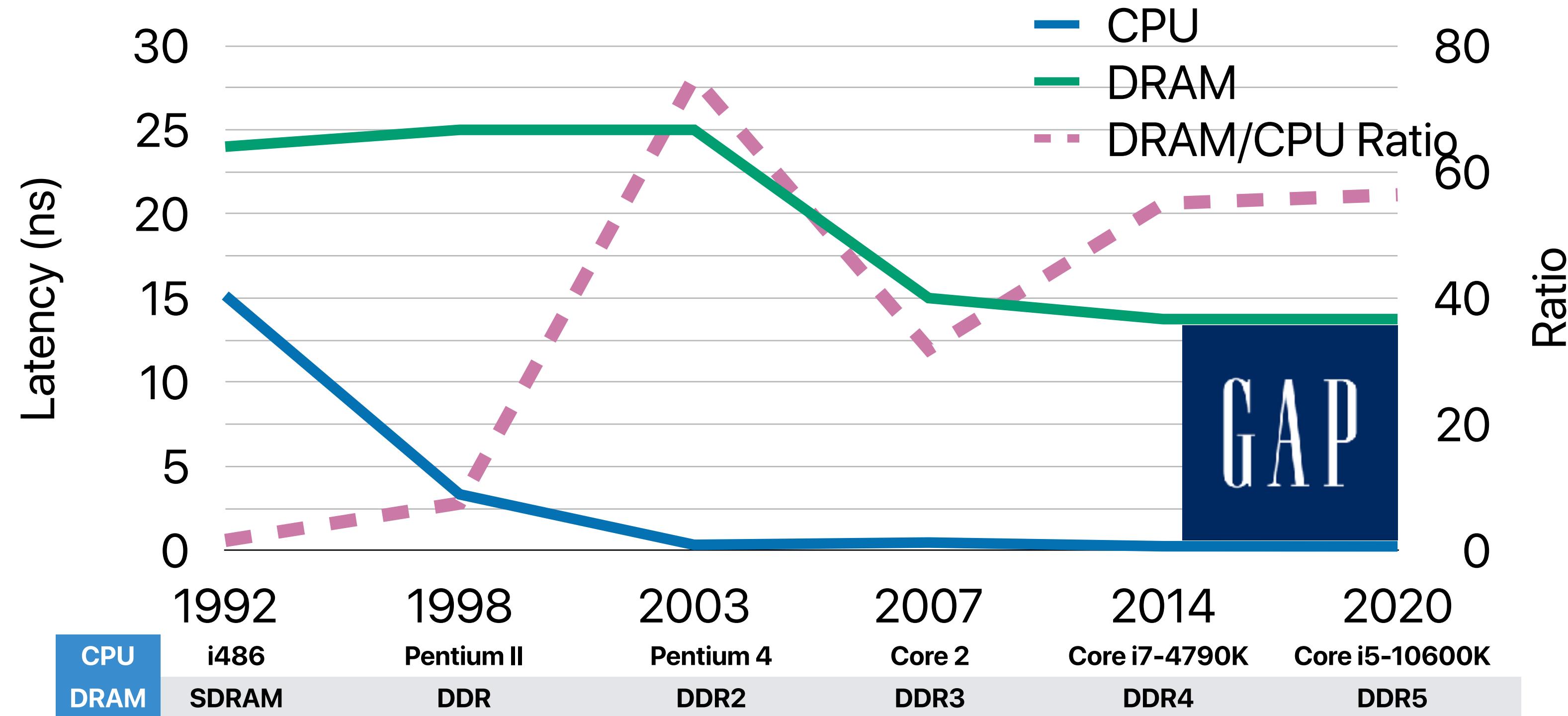
Outline

- The memory-wall problem
- The “predictable” code behavior
- Designing a cache that captures the predictability

Modern DRAM performance

| SDRAM | | | | | DDR | | | | |
|-------------------|-------------------|--------------|-----------------|-------------|-------------------|-------------------|--------------|-----------------|-------------|
| Data Rate MT/s | Bandwidth GB/s | CAS (clk) | Latency (ns) | Year | Data Rate MT/s | Bandwidth GB/s | CAS (clk) | Latency (ns) | Year |
| 100 | 0.80 | 3 | 24.00 | 1992 | 400 | 3.20 | 5 | 25.00 | 1998 |
| 133 | 1.07 | 3 | 22.50 | | 667 | 5.33 | 5 | 15.00 | |
| | | | | | 800 | 6.40 | 6 | 15.00 | |
| DDR 2 | | | | | DDR 3 | | | | |
| 400 | 3.20 | 5 | 25.00 | 2003 | 800 | 6.40 | 6 | 15.00 | 2007 |
| 667 | 5.33 | 5 | 15.00 | | 1066 | 8.53 | 8 | 15.00 | |
| 800 | 6.40 | 6 | 15.00 | | 1333 | 10.67 | 9 | 13.50 | |
| | | | | | 1600 | 12.80 | 11 | 13.75 | |
| | | | | | 1866 | 14.93 | 13 | 13.93 | |
| | | | | | 2133 | 17.07 | 14 | 13.13 | |
| DDR 4 | | | | | DDR 5 | | | | |
| 1600 | 12.80 | 11 | 13.75 | 2014 | 3200 | 25.60 | 22 | 13.75 | 2020 |
| 1866 | 14.93 | 13 | 13.92 | | 3600 | 28.80 | 26 | 14.44 | |
| 2133 | 17.07 | 15 | 14.06 | | 4000 | 32.00 | 28 | 14.00 | |
| 2400 | 19.20 | 17 | 14.17 | | 4400 | 35.20 | 32 | 14.55 | |
| 2666 | 21.33 | 19 | 14.25 | | 4800 | 38.40 | 34 | 14.17 | |
| 2933 | 23.46 | 21 | 14.32 | | 5200 | 41.60 | 38 | 14.62 | |
| 3200 | 25.20 | 22 | 13.75 | | 5600 | 44.80 | 40 | 14.29 | |
| | | | | | 6000 | 48.00 | 42 | 14.00 | |
| | | | | | 6400 | 51.20 | 46 | 14.38 | |

The “latency” gap between CPU and DRAM





The impact of “slow” memory

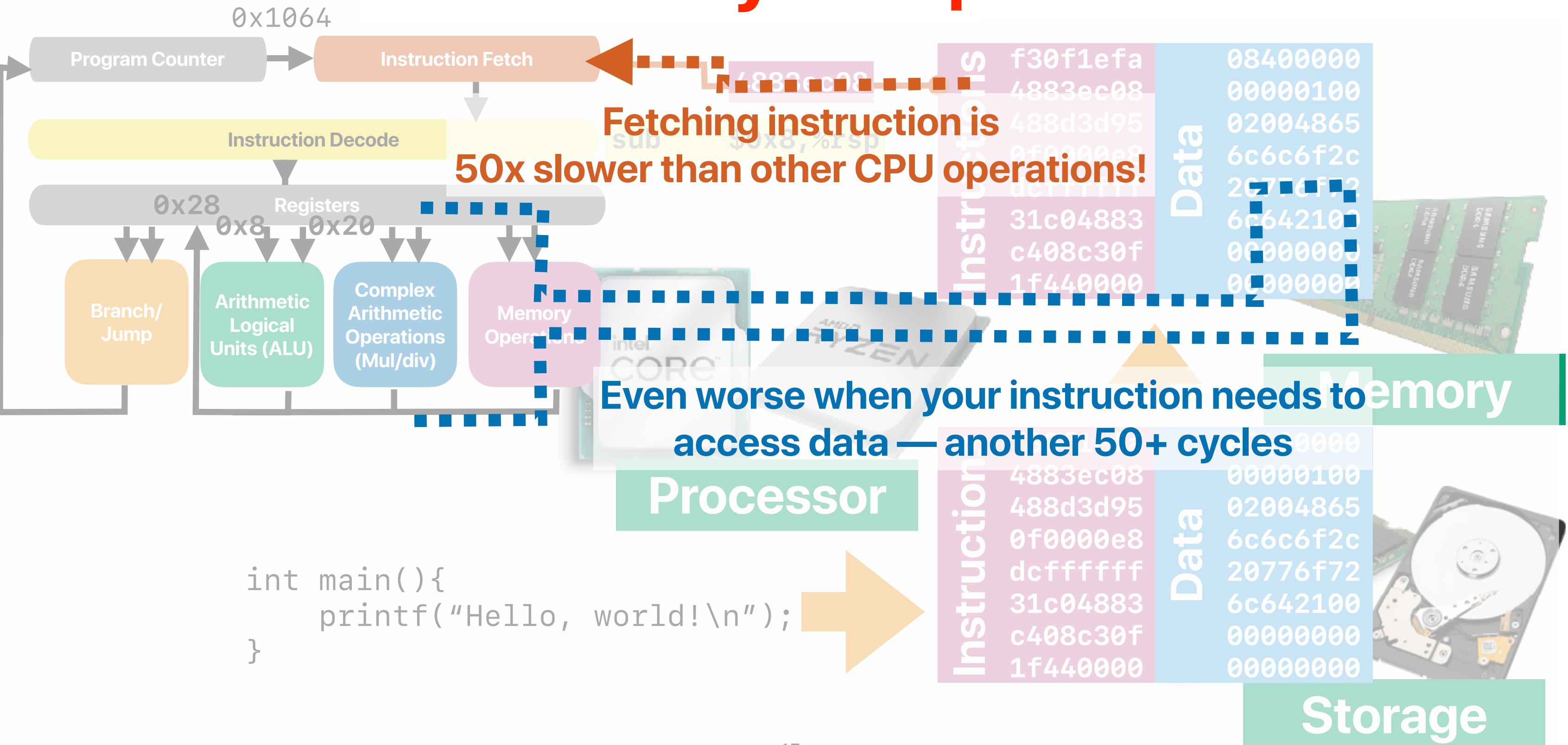
- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access and the processor already fetches the instruction, the CPI is just 1. Now, consider we have DDR5. The program is well-optimized so precharge is never necessary — the memory access latency is 13.75 ns. What's the average CPI (pick the closest one)?

- A. 9
- B. 12
- C. 15
- D. 56
- E. 67

The impact of “slow” memory

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access and the processor already fetches the instruction, the CPI is just 1. Now, consider we have DDR5. The program is well-optimized so precharge is never necessary — the memory access latency is 13.75 ns. What's the average CPI (pick the closest one)?
 - A. 9
 - B. 12
 - C. 15
 - D. 56
 - E. 67

The memory-wall problem



The impact of “slow” memory

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access and the processor already fetches the instruction, the CPI is just 1. Now, consider we have DDR5. The program is well-optimized so precharge is never necessary — the memory access latency is 13.75 ns. What's the average CPI (pick the closest one)?

A. 9

$$CPU \text{ cycle time} = \frac{1}{4 \times 10^9} = 0.25 \text{ ns}$$

B. 12

$$Each \text{ DRAM access} = \frac{13.75}{0.25} = 55 \text{ cycles}$$

C. 15

$$CPI_{average} = 1 + 100\% \times 55 + 20\% \times 55 = 67 \text{ cycles}$$

D. 56

Don't forget, instructions are also from “memory”

E. 67

$$\frac{66}{67} = 98.5 \% \text{ of time, we're dealing with memory accesses!}$$

20% is under-estimating ...

| Instruction class | MIPS examples | HLL correspondence | Frequency | |
|--------------------|--------------------------------------|--|-----------|---------|
| | | | Integer | Ft. pt. |
| Arithmetic | add, sub, addi | Operations in assignment statements | 16% | 48% |
| Data transfer | lw, sw, lb, lbu, lh, lhu, sb, lui | References to data structures, such as arrays | 35% | 36% |
| Logical | and, or, nor, andi, ori, sll, srl | Operations in assignment statements | 12% | 4% |
| Conditional branch | beq, bne,slt, slti, sltiu | If statements and loops | 34% | 8% |
| Jump | jr, jal | Procedure calls, returns, and case/switch statements | 2% | 0% |

FIGURE 2.48 MIPS instruction classes, examples, correspondence to high-level program language constructs, and percentage of MIPS instructions executed by category for the average integer and floating point SPEC CPU2006 benchmarks.

Figure 3.24 in Chapter 3 shows average percentage of the individual MIPS instructions executed.

Recap: Speedup and Amdahl's Law?

- 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

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

Take-aways: inside out our memory hierarchy

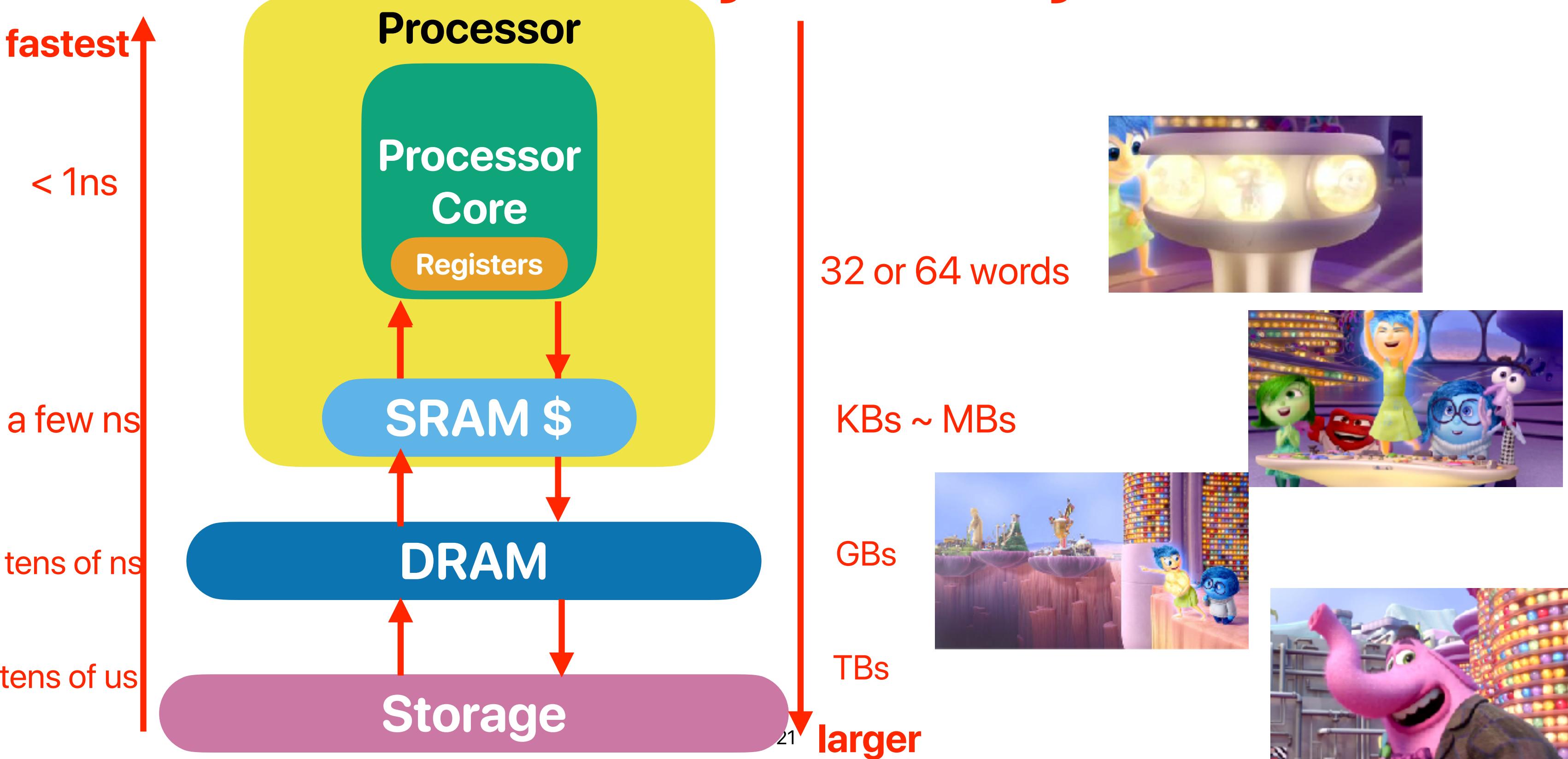
- Memory access time is the most critical performance problem
 - One memory operation is as expensive as 50 arithmetic operations
 - Processor has to fetch instructions from memory
 - We have an average of 33% of data memory access instructions!

Alternatives?

| Memory technology | Typical access time | \$ per GiB in 2012 |
|----------------------------|-------------------------|--------------------|
| SRAM semiconductor memory | 0.5–2.5 ns | \$500–\$1000 |
| DRAM semiconductor memory | 50–70 ns | \$10–\$20 |
| Flash semiconductor memory | 5,000–50,000 ns | \$0.75–\$1.00 |
| Magnetic disk | 5,000,000–20,000,000 ns | \$0.05–\$0.10 |

Fast, but expensive \$\$\$

Memory Hierarchy





How can “memory hierarchy” help in performance?

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to we DDR5, whose latency 13.75 ns, we also got an SRAM cache with latency of just at 0.5 ns and can capture **90%** of the desired data/instructions. what's the average CPI (pick the closest one)?

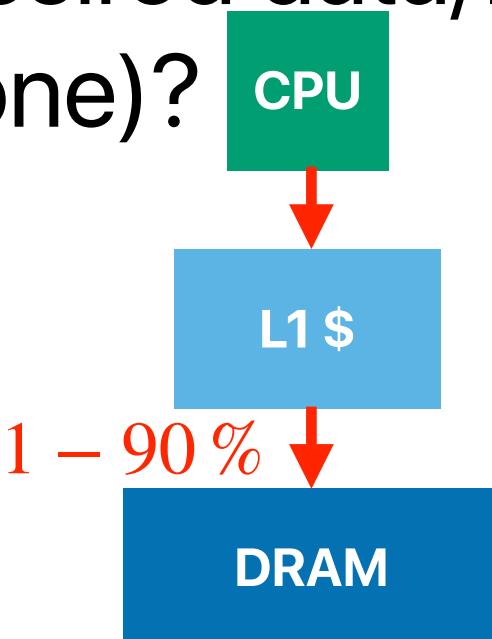
- A. 6
- B. 8
- C. 10
- D. 12
- E. 67



How can “memory hierarchy” help in performance?

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to we DDR5, whose latency 13.75 ns, we also got an SRAM cache with latency of just at 0.5 ns and can capture **90%** of the desired data/instructions. what's the average CPI (pick the closest one)?

- A. 6
- B. 8
- C. 10
- D. 12
- E. 67



$$CPU \text{ cycle time} = \frac{1}{4 \times 10^9} = 0.25\text{ns}$$

$$\text{Each \$ access} = \frac{0.5}{0.25} = 2 \text{ cycles}$$

$$\text{Each DRAM access} = \frac{13.75}{0.25} = 55 \text{ cycles}$$

$$CPI_{average} = 1 + 100\% \times [2 + (1 - 90\%) \times 55] + 20\% \times [2 + (1 - 90\%) \times 55] = 10 \text{ cycles}$$

L1? L2? L3?

CPU-Z - ID : vfljg

CPU Mainboard Memory SPD Graphics Bench About

Processor

| | | | |
|------------|----------------------|--------------|---------|
| Name | AMD Ryzen 7 7700X | | |
| Code Name | Raphael | Max TDP | 105 W |
| Package | Socket AM5 (LGA1718) | | |
| Technology | 5 nm | Core Voltage | 1.288 V |

Specification

| | | | |
|------------------------------------|----|------------|--------|
| AMD Ryzen 7 7700X 8-Core Processor | | | |
| Family | F | Model | 1 |
| Ext. Family | 19 | Ext. Model | 61 |
| Stepping | 2 | Revision | RPL-B2 |

Instructions

| | | | |
|--|--|--|--|
| MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A x86-64, AMD-V, AES, AVX, AVX2, AVX512, FMA3, SHA | | | |
|--|--|--|--|

Clocks (Core #0)

| | |
|------------|---------------------|
| Core Speed | 5188.99 MHz |
| Multiplier | x 52.0 (4 - 55.5) |
| Bus Speed | 99.79 MHz |
| Rated FSB | |

Cache

| | |
|----------|-------------|
| L1 Data | 8 x 32 KB |
| L1 Inst. | 8 x 32 KB |
| Level 2 | 8 x 1024 KB |
| Level 3 | 32 MBytes |

Selection Socket #1 Cores 8 Threads 16

CPU-Z Ver. 2.09.0.x64 Tools Validate Close

CPU-Z - ID : pk15b

CPU Mainboard Memory SPD Graphics Bench About

Processor

| | | | |
|------------|----------------------|--------------|---------|
| Name | Intel Core i7 14700K | | |
| Code Name | Raptor Lake | Max TDP | 125 W |
| Package | Socket 1700 LGA | | |
| Technology | 10 nm | Core Voltage | 1.412 V |

Specification

| | | | |
|-----------------------------|---|------------|----|
| Intel(R) Core(TM) i7-14700K | | | |
| Family | 6 | Model | 7 |
| Ext. Family | 6 | Ext. Model | B7 |
| Stepping | 1 | Revision | B0 |

Instructions

| | | | |
|--|--|--|--|
| MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T VT-x, AES, AVX, AVX2, FMA3, SHA | | | |
|--|--|--|--|

Clocks (Core #0)

| | |
|------------|-------------------|
| Core Speed | 5287.07 MHz |
| Multiplier | x 53.0 (8 - 55) |
| Bus Speed | 99.76 MHz |
| Rated FSB | |

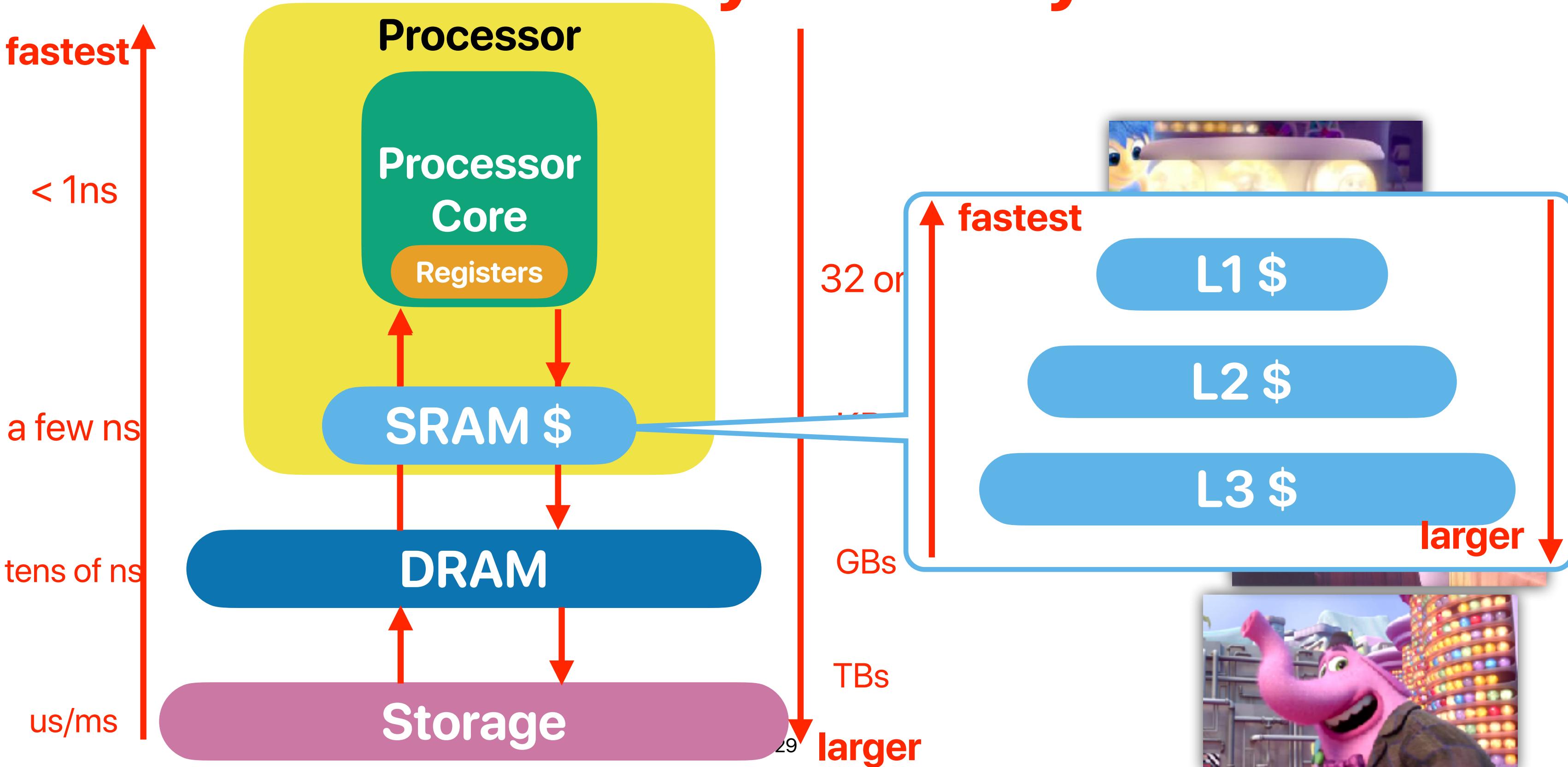
Cache

| | |
|----------|------------------------|
| L1 Data | 8 x 48 KB + 12 x 32 KB |
| L1 Inst. | 8 x 32 KB + 12 x 64 KB |
| Level 2 | 8 x 2 MB + 3 x 4 MB |
| Level 3 | 33 MBytes |

Selection Socket #1 Cores 8 + 12 Threads 28

CPU-Z Ver. 2.08.0.x64 Tools Validate Close

Memory Hierarchy





How can a deeper memory hierarchy help in performance?

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to the DDR5, whose latency is 13.75 ns, we also got 2 levels SRAM caches where
 - the 1st-level one at latency of 0.5ns and can capture 90% of the desired data/instructions.
 - the 2nd-level at latency of 5 ns and can capture 60% of the desired data/instructions

What's the average CPI (pick the closest one)?

- A. 6
- B. 8
- C. 10
- D. 12
- E. 67

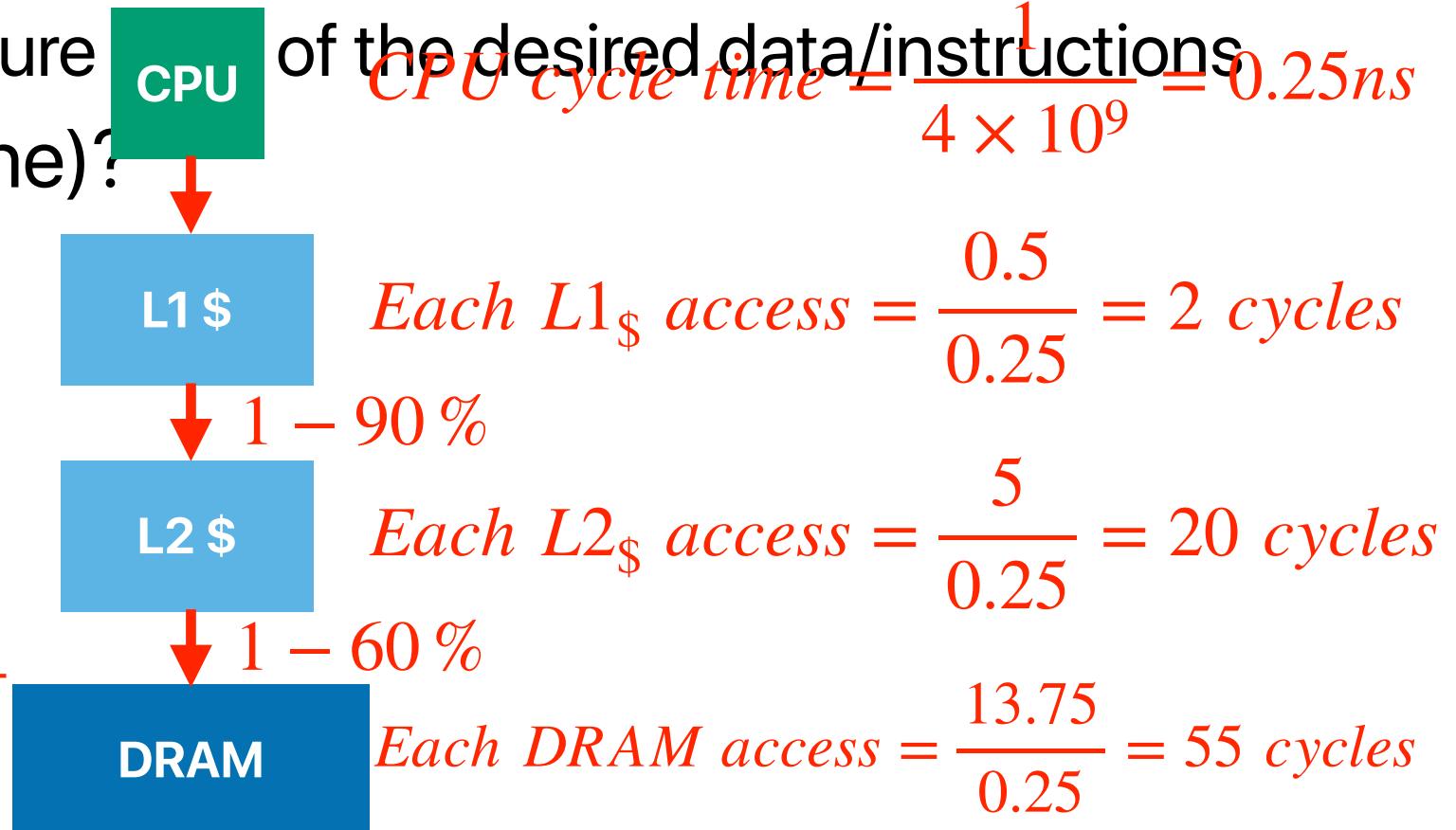


How can a deeper memory hierarchy help in performance?

- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to the DDR5, whose latency is 13.75 ns, we also got 2 levels SRAM caches where
 - the 1st-level one at latency of 0.5ns and can capture 90% of the desired data/instructions.
 - the 2nd-level at latency of 5 ns and can capture $\frac{1}{CPU\ cycle\ time} = \frac{1}{4 \times 10^9} = 0.25\ ns$ of the desired data/instructions.
- What's the average CPI (pick the closest one)?

- A. 6
- B. 8
- C. 10
- D. 12

$$CPI_{average} = 100\% \times [2 + (1 - 90\%) \times (20 + (1 - 60\%) \times 55)] + 20\% \times [2 + (1 - 90\%) \times (20 + (1 - 60\%) \times 55)] = 8.44\ cycles$$



L1? L2? L3?

Can we really “predict” upcoming data accurately (e.g., 90%) with such small caches?

The image shows two side-by-side CPU-Z software interfaces. Both are titled "CPU-Z - ID : vfljg" and "CPU-Z - ID : pk15b". Each interface has tabs for CPU, Mainboard, Memory, SPD, Graphics, Bench, and About. The "Processor" section displays the processor name, code name, max TDP, package, technology, specification, ext. family, instructions, and clock information. The "Cache" section lists L1 Data, L1 Inst., Level 2, and Level 3 cache sizes. A large red box highlights the Cache sections of both processors.

Processor

| Name | AMD Ryzen 7 7700X | Intel Core i7 14700K |
|------------------|---|---|
| Code Name | Raphael | Raptor Lake |
| Max TDP | 105 W | 125 W |
| Technology | 5 nm | 18 nm |
| Core Voltage | 1.283 V | 1.112 V |
| Specification | AMD Ryzen 7 7700X 8-Core Processor | Intel(R) Core(TM) i7-14700K |
| Ext. Family | 19 | 6 |
| Ext. Model | 1 | 67 |
| Revision | RPL-12 | B0 |
| Instructions | MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A x86-64, AMD-V, AES, AVX, AVX2, AVX512, FMA3, SHA | MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T VT-x, AES, AVX, AVX2, FMA3, SHA |
| Clocks (Core #0) | | |
| Core Speed | 5188.99 MHz | 5287.07 MHz |
| Multiplier | x 52.0 (4 - 55.5) | x 53.0 (8 - 55) |
| Bus Speed | 99.79 MHz | 99.76 MHz |
| Rated FSB | | |
| Cache | | |
| L1 Data | 8 x 32 KB | 8 x 48 KB + 12 x 32 KB |
| L1 Inst. | 8 x 32 KB | 8 x 32 KB + 12 x 64 KB |
| Level 2 | 8 x 1024 KB | 8 x 2 MB + 3 x 4 MB |
| Level 3 | 32 MBytes | 33 MBytes |
| Selection | Socket #1 | Socket #1 |
| Cores | 8 | 8 + 12 |
| Threads | 16 | 28 |

CPU-Z Ver. 2.09.0.x64 Tools Validate Close

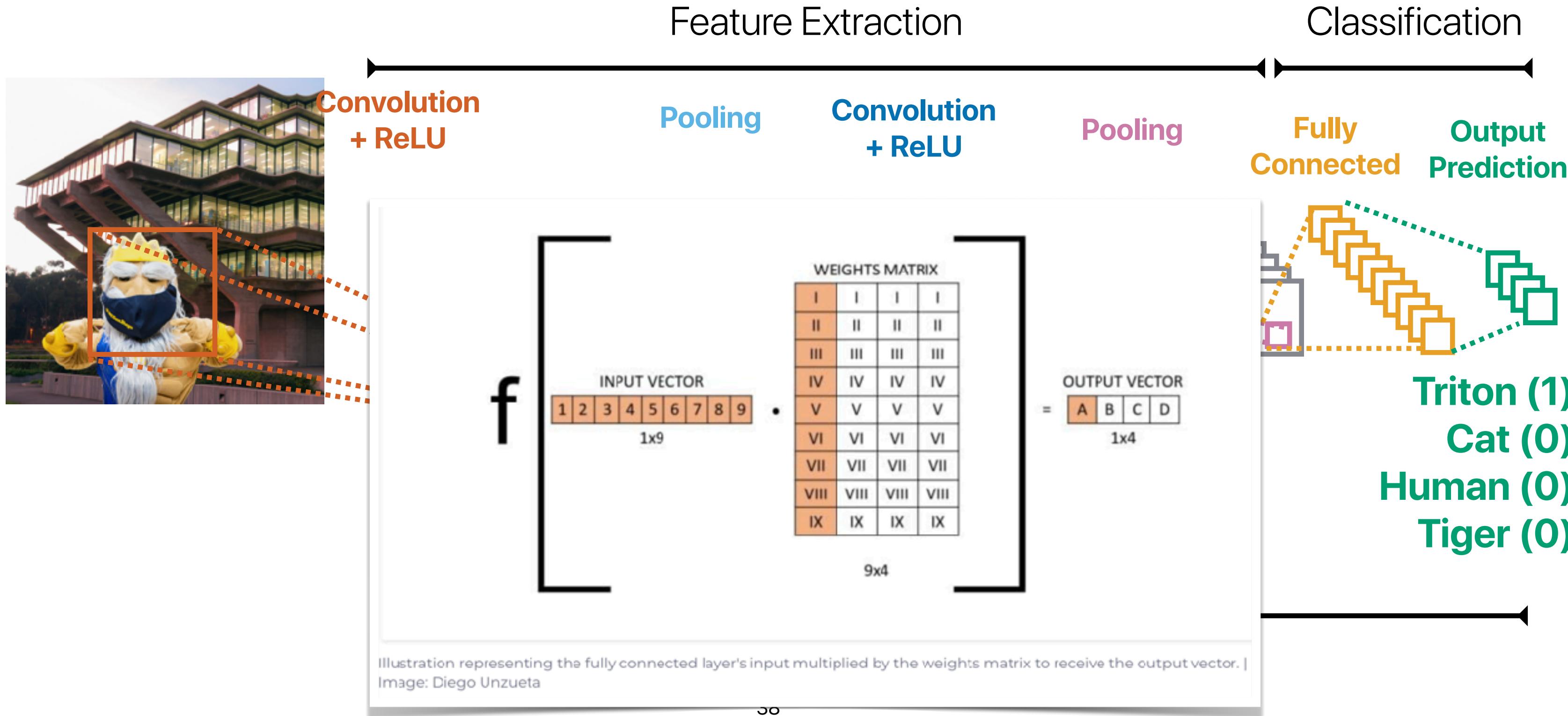
CPU-Z Ver. 2.08.0.x64 Tools Validate Close

Take-aways: inside out our memory hierarchy

- Memory access time is the most critical performance problem
 - One memory operation is as expensive as 50 arithmetic operations
 - Processor has to fetch instructions from memory
 - We have an average of 33% of data memory access instructions!
- Hierarchical caching with small amount of SRAMs will work if we can efficiently capture data and instructions

The predictability of your code

The Machine Learning Inference Pipeline





Locality of data

- Which description about locality of arrays `matrix` and `vector` in the following code is the **most accurate**?

```
for(uint64_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint64_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```



- A. Access of `matrix` has temporal locality, `vector` has spatial locality
- B. Both `matrix` and `vector` have temporal locality, and `vector` also has spatial locality
- C. Access of `matrix` has spatial locality, `vector` has temporal locality
- D. Both `matrix` and `vector` have spatial locality and temporal locality
- E. Both `matrix` and `vector` have spatial locality, and `vector` also has temporal locality

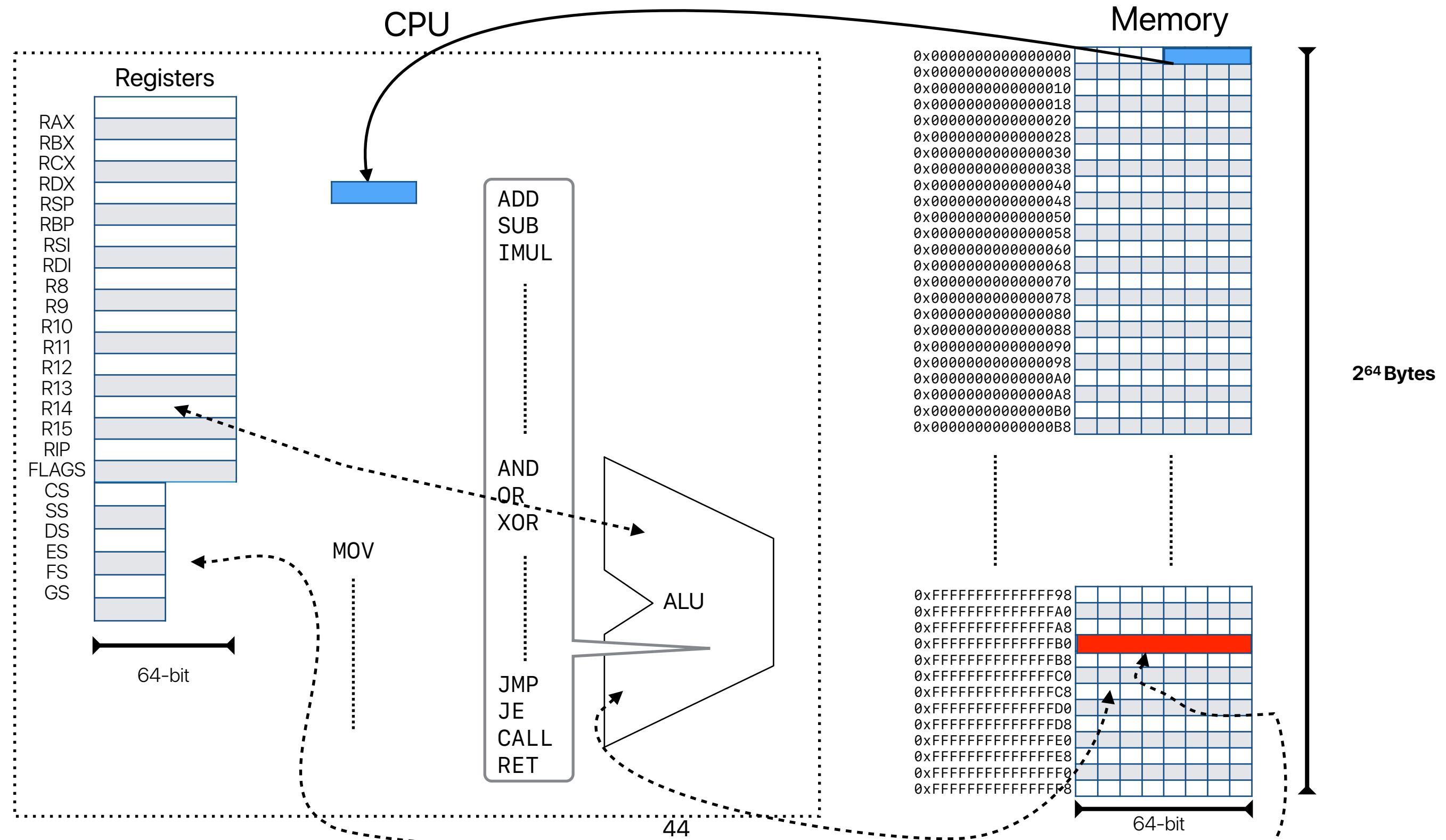
Data locality

- Which description about locality of arrays `matrix` and `vector` in the following code is the **most accurate**?

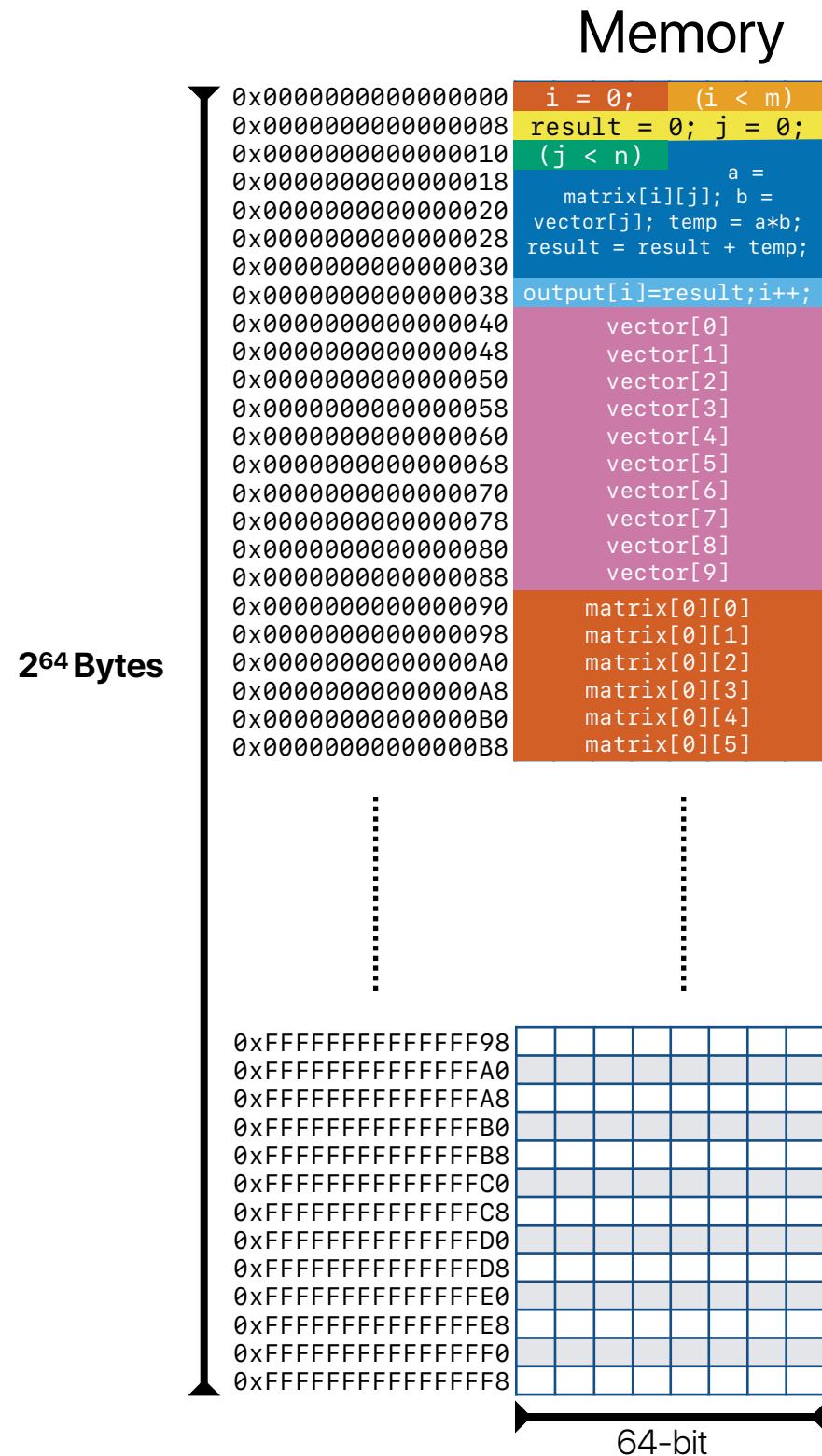
```
for(uint64_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint64_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```

- A. Access of `matrix` has temporal locality, `vector` has spatial locality
- B. Both `matrix` and `vector` have temporal locality, and `vector` also has spatial locality
- C. Access of `matrix` has spatial locality, `vector` has temporal locality
- D. Both `matrix` and `vector` have spatial locality and temporal locality
- E. Both `matrix` and `vector` have spatial locality, and `vector` also has temporal locality

x86 ISA: the abstracted machine



Data/instructions in the memory



Code also has locality

```
for(uint64_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint64_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```

repeat many times —
temporal locality!

```
i = 0;  
while(i < m) {  
    result = 0;  
    j = 0;  
    while(j < n) {  
        a = matrix[i][j];  
        b = vector[j];  
        temp = a*b;  
        result = result + temp;  
    }  
    output[i] = result;  
    i++;
```

keep going to the
next instruction —
spatial locality



Data locality

- Which description about locality of arrays `matrix` and `vector` in the following code is the **most accurate**?

```
for(uint64_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint64_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```

spatial locality:
`matrix[0][0], matrix[0][1], matrix[0][2], ...`
`vector[0], vector[1], ..., vector[n]`
temporal locality:
`reuse of vector[0], vector[1], ...`

- A. Access of `matrix` has temporal locality, `vector` has spatial locality
- B. Both `matrix` and `vector` have temporal locality, and `vector` also has spatial locality
- C. Access of `matrix` has spatial locality, `vector` has temporal locality
- D. Both `matrix` and `vector` have spatial locality and temporal locality
- E. Both `matrix` and `vector` have spatial locality, and `vector` also has temporal locality

Locality

- Spatial locality — application tends to visit nearby stuffs in the memory
 - Code — the current instruction, and then PC + 4
 - Data — the current element in an array, then the next
- Temporal locality — application revisit the same thing again and again
 - Code — loops, frequently invoked functions
 - Data — the same data can be read/write many times

Locality

- Spatial locality — application tends to visit nearby stuffs in the memory
 - Code — the current instruction, and then the next instruction

Most of time, your program is just visiting a very small amount of data/instructions within a given window

- Temporal Locality — application revisit the same thing again and again
 - Code — loops, frequently invoked functions
 - Typically tens of static instructions — at most several KBs
 - Data — program can read/write the same data many times (e.g., vectors in matrix-vector product)

Take-aways: inside out our memory hierarchy

- Memory access time is the most critical performance problem
 - One memory operation is as expensive as 50 arithmetic operations
 - Processor has to fetch instructions from memory
 - We have an average of 33% of data memory access instructions!
- Hierarchical caching with small amount of SRAMs will work if we can efficiently capture data and instructions
- Caching is possible! Most of time, we only work on a small amount of data!
 - Spatial locality
 - Temporal locality

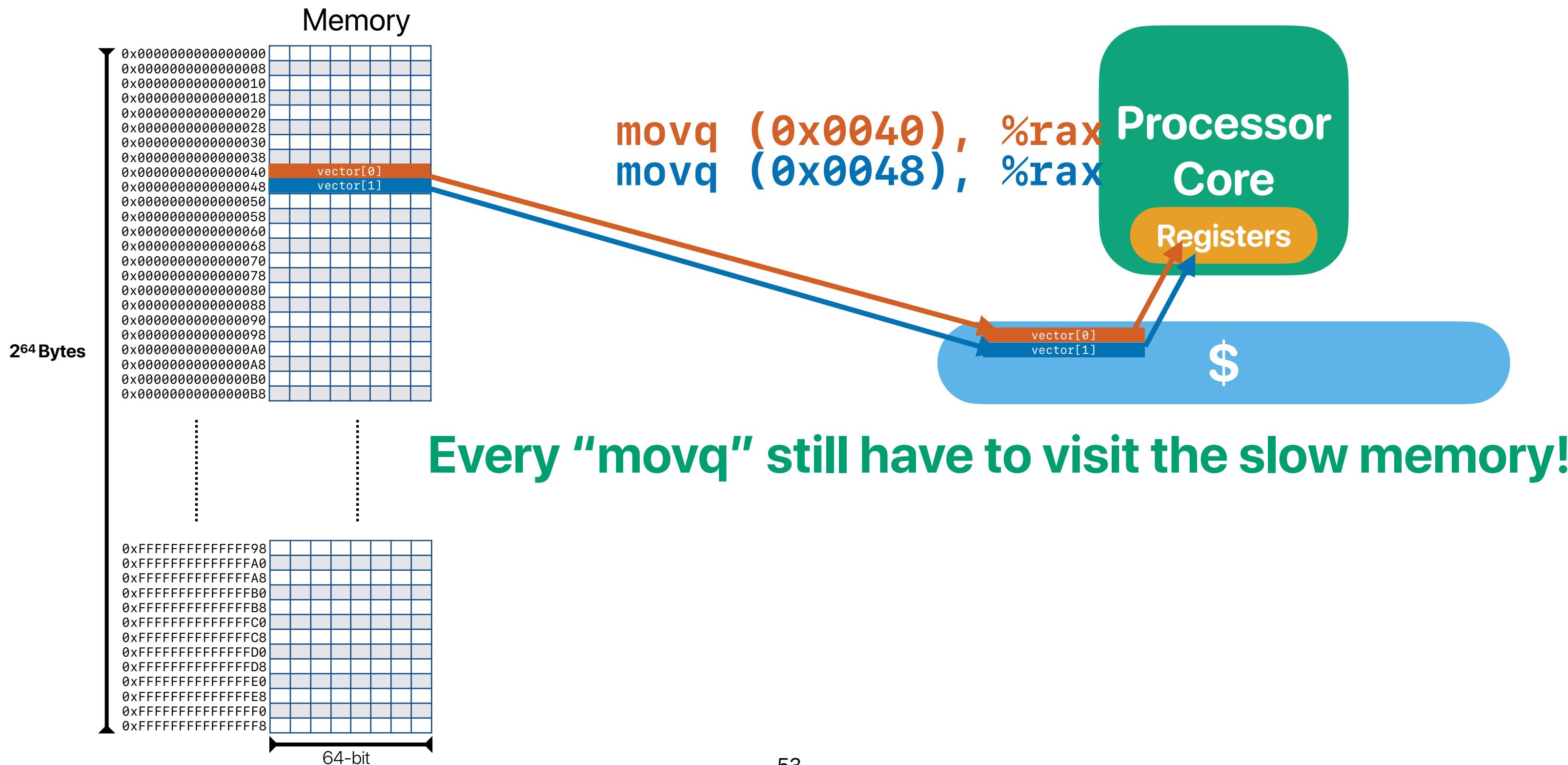
Designing a hardware to exploit locality

- Spatial locality — application tends to visit nearby stuffs in the memory

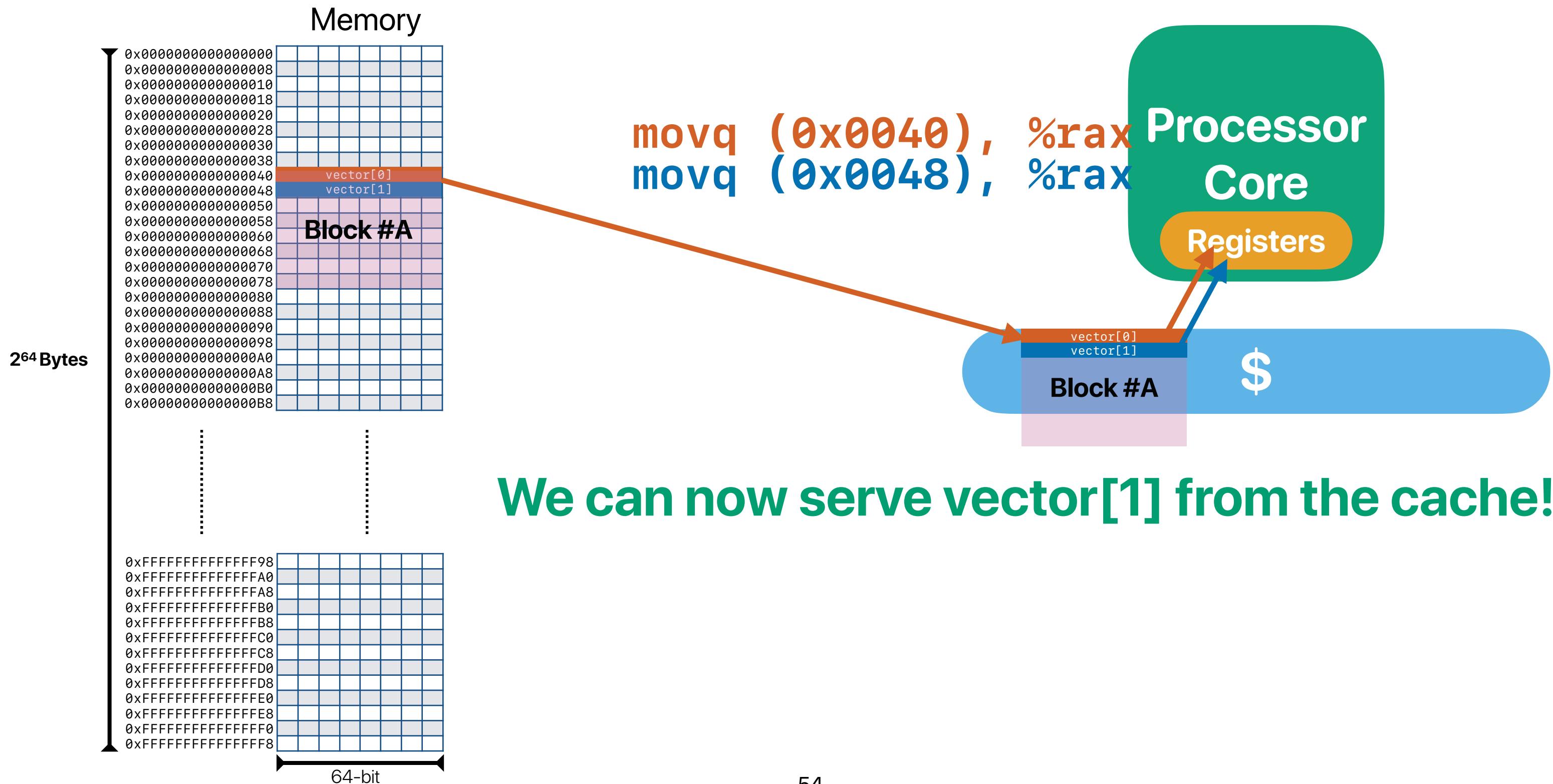
**We need to “cache consecutive memory locations” every time
— the cache should store a “block” of code/data**

- Temporal locality — application revisit the same thing again and again
 - Code — loops, frequently invoked functions
 - Typically tens of static instructions — at most several KBs
 - Data — program can read/write the same data many times (e.g., vectors in matrix-vector product)

If we just cache what has requested



If we can cache a block of data



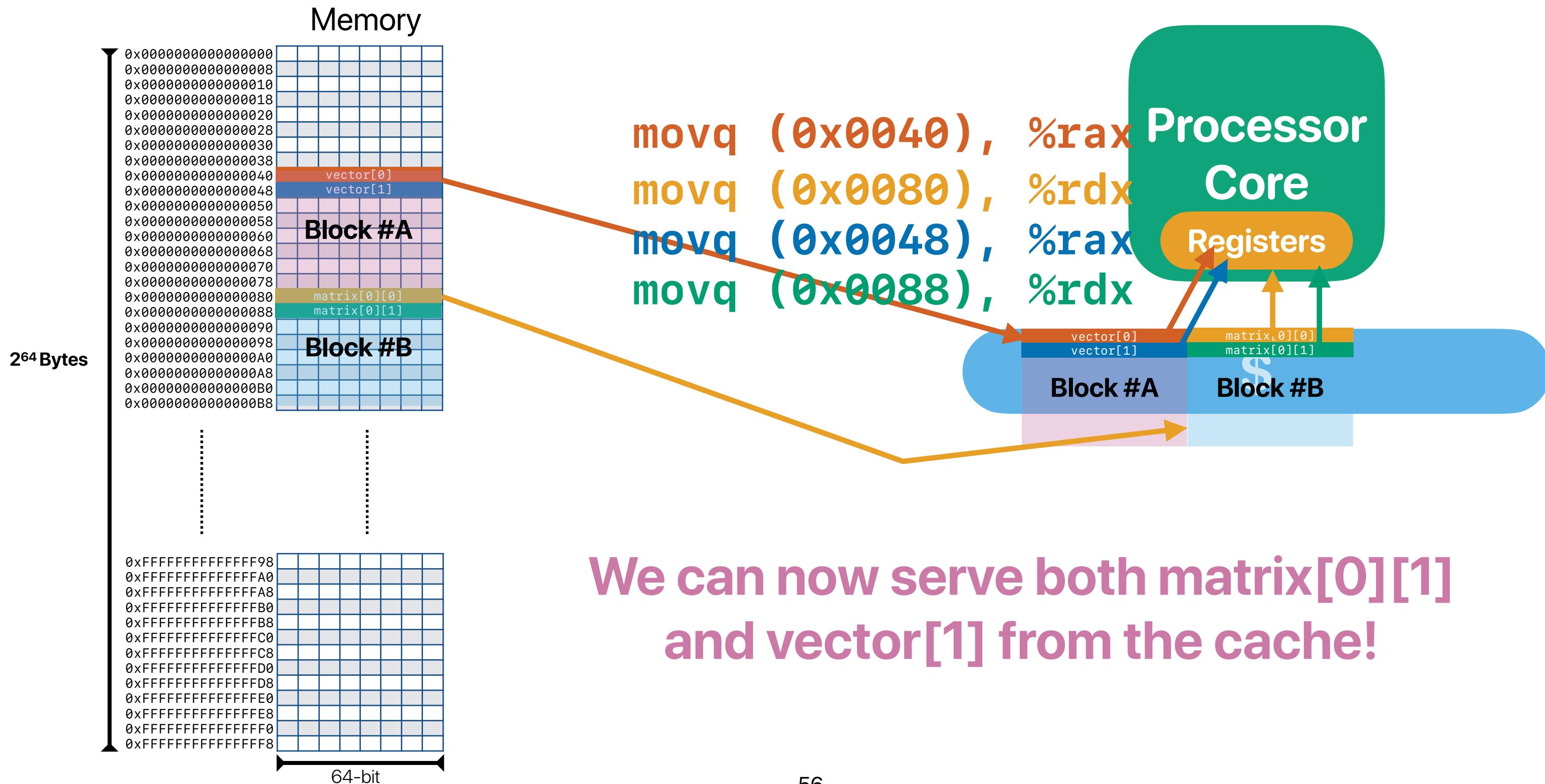
Recap: Locality

- Which description about locality of arrays `matrix` and `vector` in the following code is the **most accurate**?

```
for(uint64_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint64_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```

Simply caching one block
isn't enough

We need to cache multiple blocks of data!



Designing a hardware to exploit locality

- Spatial locality — application tends to visit nearby stuffs in the memory

We need to “cache consecutive memory locations” every time
—**the cache should store a “block” of code/data**

- Temporal locality — application revisit the same thing again and again

We need to “cache frequently used memory blocks”

- Typically tens of static instructions — at most several KBs
- the cache should store a few blocks**
- Data — program can read/write the same data many times (e.g., vectors in matrix-vector product)
- the cache must be able to distinguish blocks**

How to tell who is there?

?

?

0123456789ABCDEF

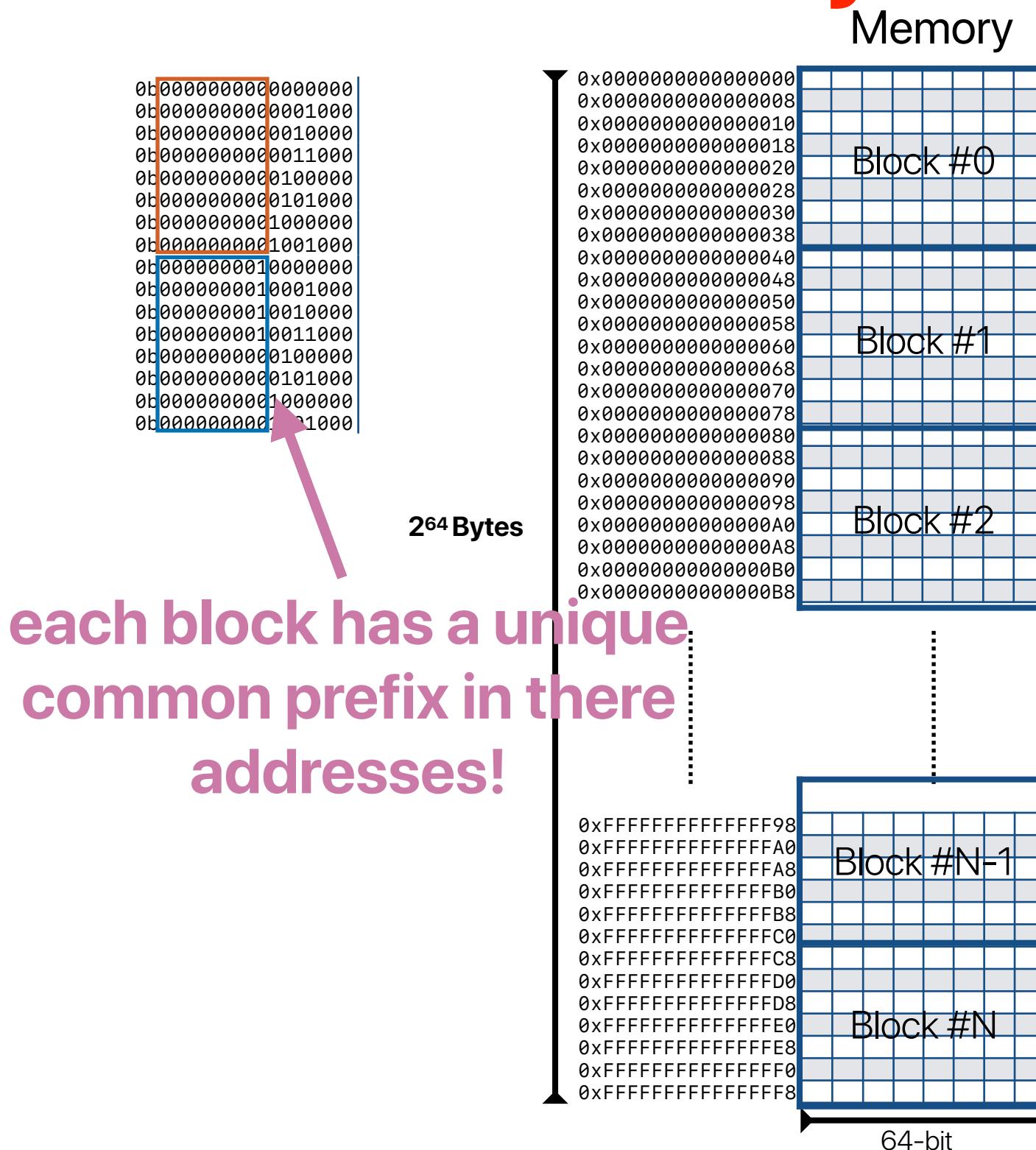
This is CS 203:

Advanced Compute

r Architecture!

This is CS 203:

Partition memory addresses into fix-sized chunks



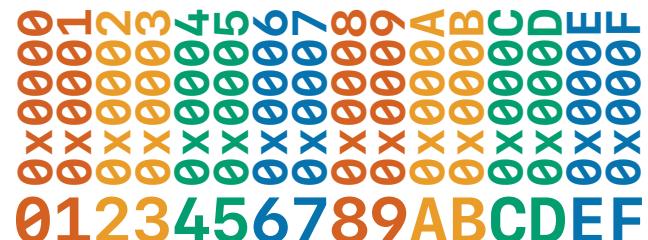
\$

How to tell who is there?

the common address
prefix in each block

tag array

| | tag array |
|-------|------------------|
| 0x000 | This is CS 203: |
| 0x001 | Advanced Compute |
| 0xF07 | r Architecture! |
| 0x100 | This is CS 203: |
| 0x310 | Advanced Compute |
| 0x450 | r Architecture! |
| 0x006 | This is CS 203: |
| 0x537 | Advanced Compute |
| 0x266 | r Architecture! |
| 0x307 | This is CS 203: |
| 0x265 | Advanced Compute |
| 0x80A | r Architecture! |
| 0x620 | This is CS 203: |
| 0x630 | Advanced Compute |
| 0x705 | r Architecture! |
| 0x216 | This is CS 203: |



Processor Core

Registers

How to tell whether

block offset
tag

1w 0x0008

1w 0x4048

0x404 not found,
go to lower-level memory

| | | Valid Bit | Dirty Bit | Tag | Data |
|---|---|-----------|-----------|-------|------------------|
| 1 | 1 | | | 0x000 | This is CSE13: |
| 1 | 1 | | | 0x001 | Advanced Compute |
| 1 | 0 | | | 0xF07 | r Architecture! |
| 0 | 1 | | | 0x100 | This is CS 203: |
| 1 | 1 | | | 0x310 | Advanced Compute |
| 1 | 1 | | | 0x450 | r Architecture! |
| 0 | 1 | | | 0x006 | This is CS 203: |
| 0 | 1 | | | 0x537 | Advanced Compute |
| 1 | 1 | | | 0x266 | r Architecture! |
| 1 | 1 | | | 0x307 | This is CS 203: |
| 0 | 1 | | | 0x265 | Advanced Compute |
| 0 | 1 | | | 0x80A | r Architecture! |
| 1 | 1 | | | 0x620 | This is CS 203: |
| 1 | 1 | | | 0x630 | Advanced Compute |
| 1 | 0 | | | 0x705 | r Architecture! |
| 0 | 1 | | | 0x216 | This is CS 203: |

Tell if the block here can be used

Tell if the block here is modified

Blocksize == Linesize

```
[5]: # Your CS203 Cluster
! cs203 demo "lscpu | grep 'Model name'; getconf -a | grep CACHE"
ssh htseng@horsea " srun -N1 -p datahub lscpu | grep 'Model name'"
Model name:                                     12th Gen Intel(R) Core(TM) i3-12100F
ssh htseng@horsea " srun -N1 -p datahub  getconf -a | grep CACHE"
LEVEL1_ICACHE_SIZE                      32768
LEVEL1_ICACHE_ASSOC                     8
LEVEL1_ICACHE_LINESIZE                  64
LEVEL1_DCACHE_SIZE                     49152
LEVEL1_DCACHE_ASSOC                    12
LEVEL1_DCACHE_LINESIZE                64
LEVEL2_CACHE_SIZE                      1310720
LEVEL2_CACHE_ASSOC                     10
LEVEL2_CACHE_LINESIZE                 64
LEVEL3_CACHE_SIZE                      12582912
LEVEL3_CACHE_ASSOC                     12
LEVEL3_CACHE_LINESIZE                 64
LEVEL4_CACHE_SIZE                      0
LEVEL4_CACHE_ASSOC                     0
LEVEL4_CACHE_LINESIZE                 0
```

Take-aways: inside out our memory hierarchy

- Memory access time is the most critical performance problem
 - One memory operation is as expensive as 50 arithmetic operations
 - Processor has to fetch instructions from memory
 - We have an average of 33% of data memory access instructions!
- Hierarchical caching with small amount of SRAMs will work if we can efficiently capture data and instructions
- Caching is possible! Most of time, we only work on a small amount of data!
 - Spatial locality
 - Temporal locality
- Basic cache structures —
 - Caching in granularity of a block to capture spatial locality
 - Caching multiple blocks to keep frequently used data — temporal locality
 - Tags to distinguish cached blocks

Announcement

- Reading quiz #3 due next **Tuesday** before the lecture
- Assignment #1 due **this Thursday**
- Office Hour:
Wednesdays 2p-4p, WCH 406

Computer Science & Engineering

203

つづく

