

Memory Hierarchy (1): Inside Out the Computer Memory

Hung-Wei Tseng

Disney · PIXAR

INSIDE OUT

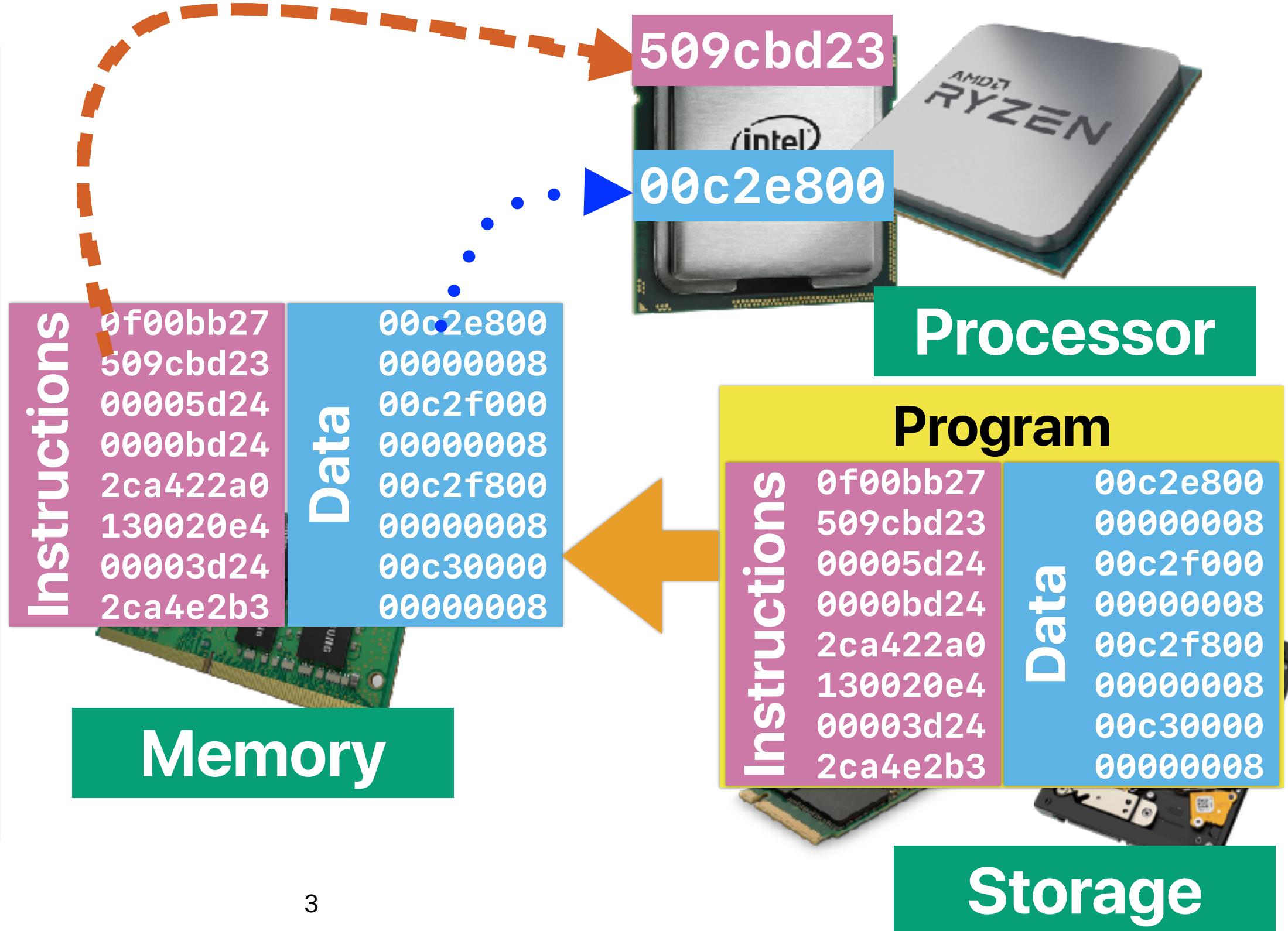
[GET DISNEY+](#)[▶ TRAILER](#)

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.



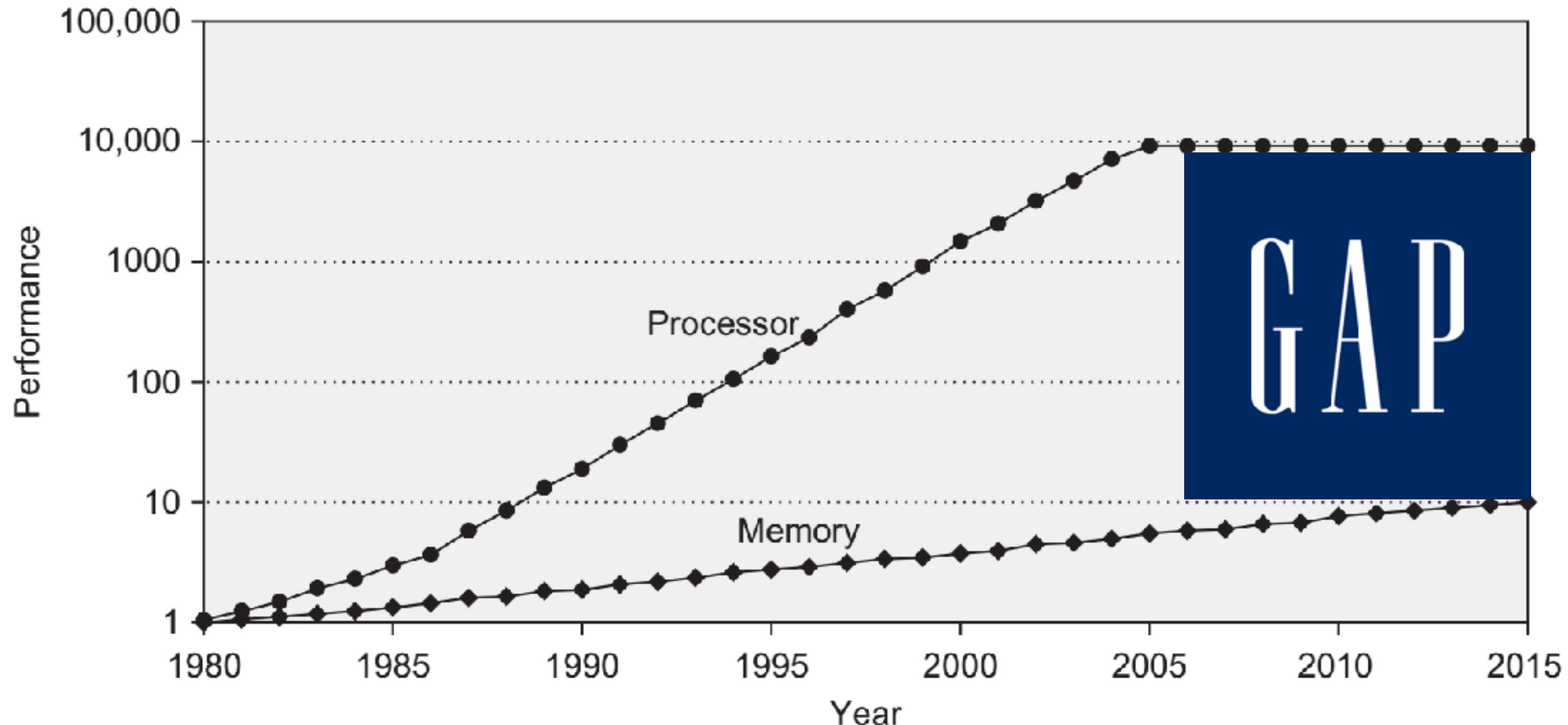
von Neumann Architecture



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	

Recap: Performance gap between Processor/Memory



**How did you prepare an examine
when you're short on time?**

Outline

- The Basic Idea behind Memory Hierarchy
- How cache works



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, 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)?
 - 9
 - 12
 - 15
 - 56
 - 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, 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

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

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

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

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

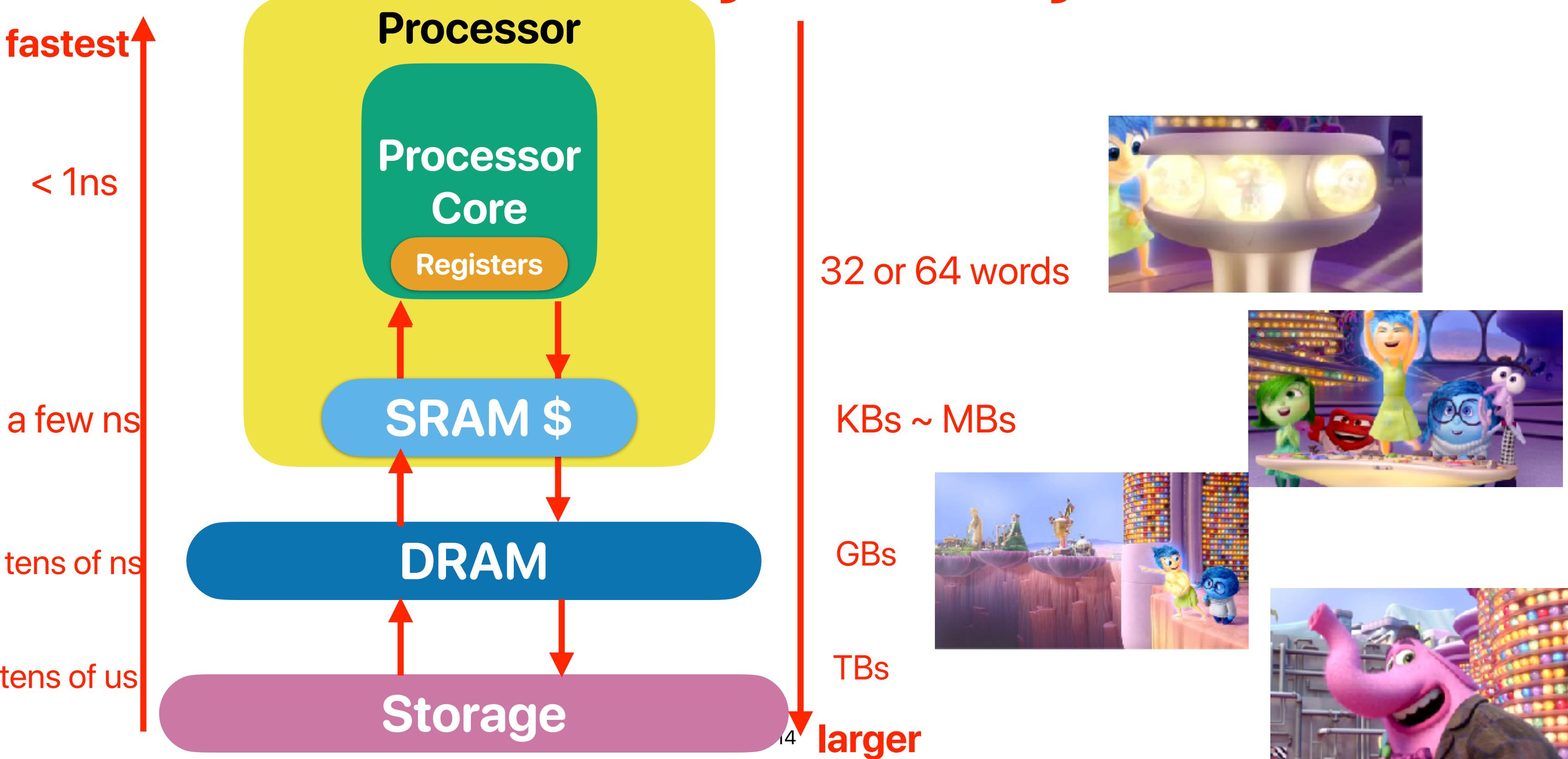
$$\frac{66}{67} = 98.5 \% \text{ of time, we're dealing with memory accesses!}$$

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?

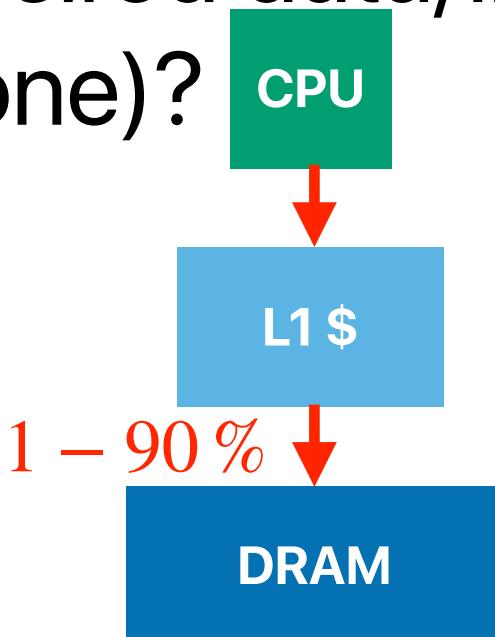
- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to we DDR5, whose latency 13.75 ns, we also got an SRAM cache with latency of just at 0.5 ns and can capture 90% of the desired 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 : wswpbb

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name	AMD Ryzen 7 2700X		
Code Name	Pinnacle Ridge	Max TDP	105 W
Package	Socket AM4 (1331)		
Technology	12 nm	Core Voltage	1.36 V
Specification	AMD Ryzen 7 2700X Eight-Core Processor		
Family	F	Model	8
Ext. Family	17	Ext. Model	8
Instructions	MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA		

Clocks (Core #0)

Core Speed	4290.73 MHz	
Multiplier	x 43.0	
Bus Speed	99.78 MHz	
Rated FSB		

Cache

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 64 KBytes	4-way
Level 2	8 x 512 KBytes	8-way
Level 3	2 x 8192 KBytes	16-way

Selection Processor #1 Cores 8 Threads 16

CPU-Z Ver. 1.86.0.x64 Tools Validate Close

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name	Intel Core i7 9700K		
Code Name	Coffee Lake	Max TDP	95.0 W
Package	Socket 1151 LGA		
Technology	14 nm	Core Voltage	0.737 V
Specification	Intel® Core™ i7-9700K CPU @ 3.60GHz (ES)		
Family	6	Model	E
Ext. Family	6	Ext. Model	9E
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX		

Clocks (Core #0)

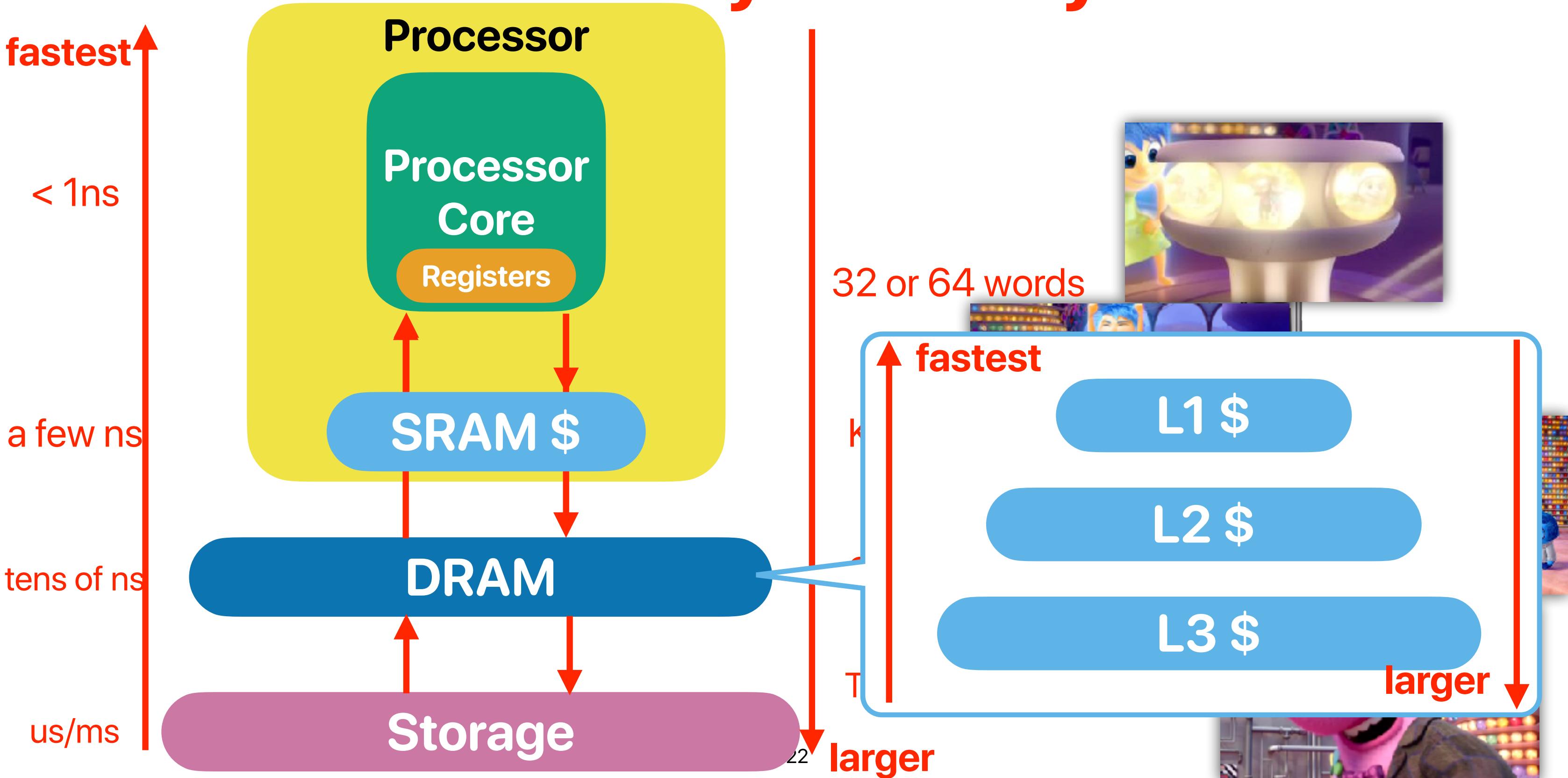
Core Speed	4798.85 MHz	
Multiplier	x 48.0 (8 - 49)	
Bus Speed	99.98 MHz	
Rated FSB		

Cache

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 32 KBytes	8-way
Level 2	8 x 256 KBytes	4-way
Level 3	12 MBytes	12-way

Selection Socket #1 Cores 8 Threads 8

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 we DDR5, whose latency 13.75 ns, we also got a 2-level SRAM caches with
 - it's 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

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 a deeper memory hierarchy help in performance?

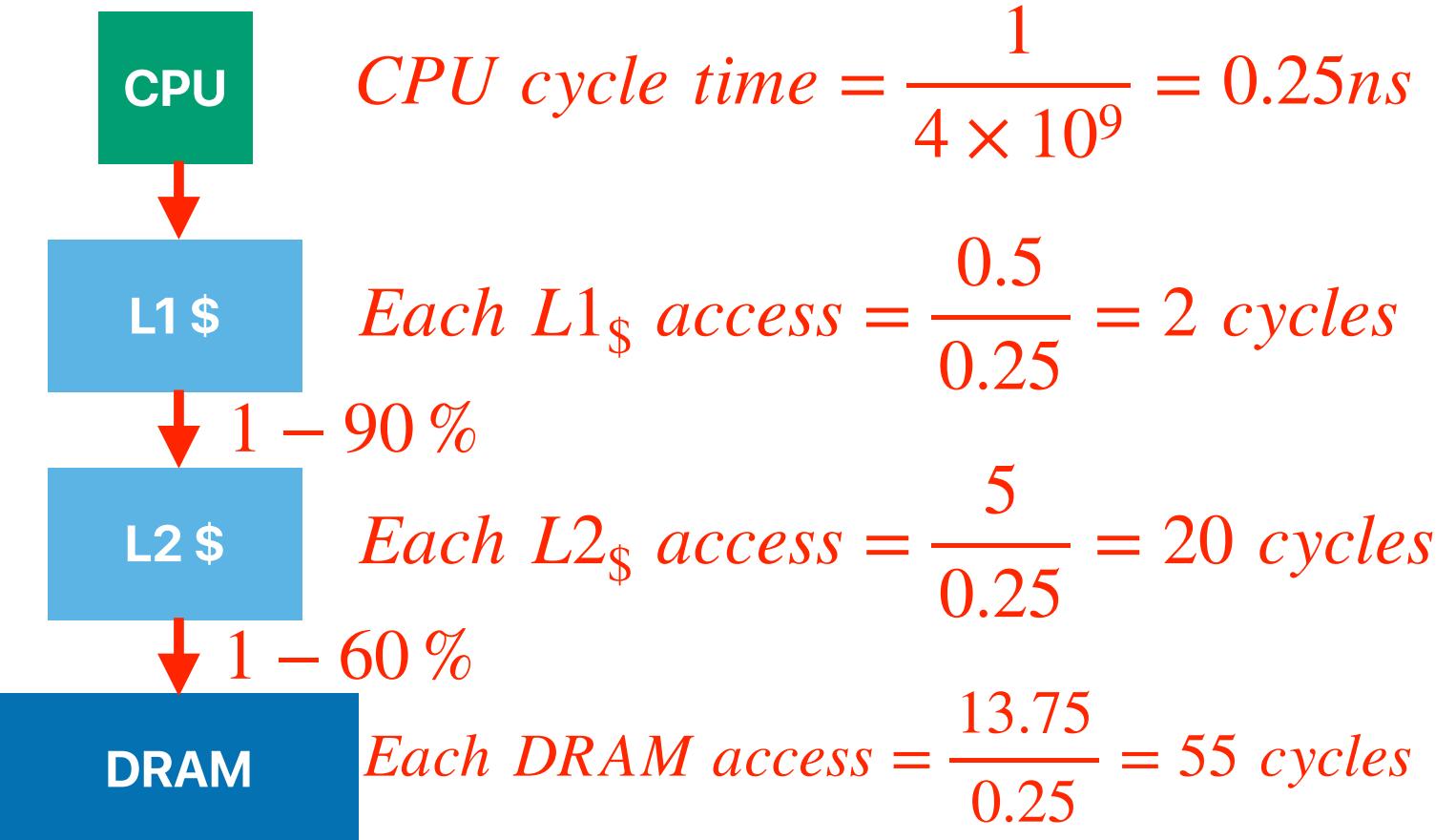
- Assume that we have a processor running @ 4 GHz and a program with 20% of load/store instructions. If the instruction has no memory access, the CPI is just 1. Now, in addition to we DDR5, whose latency 13.75 ns, we also got a 2-level SRAM caches with
 - it's 1st-level one at latency of 0.5ns and can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

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

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



L1? L2? L3?

These are very small compared with your system main memory and program footprint. How does that work?

The image shows two CPU-Z interface windows side-by-side. Both windows have a top navigation bar with tabs: CPU, Caches, Mainboard, Memory, SPD, Graphics, Bench, and About. The left window is for an AMD Ryzen 7 2700X processor, and the right window is for an Intel Core i7 9700K processor.

AMD Ryzen 7 2700X Cache Details:

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 64 KBytes	4-way
Level 2	8 x 512 KBytes	8-way
Level 3	2 x 8192 KBytes	16-way

Intel Core i7 9700K Cache Details:

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 32 KBytes	8-way
Level 2	8 x 256 KBytes	4-way
Level 3	12 MBytes	12-way

Both tables are highlighted with red boxes.

At the bottom of each window, there are selection dropdowns for "Processor #1" and "Socket #1", and buttons for "Tools", "Validate", and "Close".

The “theory” behind caching



Locality of data

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

```
for(uint32_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint32_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(uint32_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint32_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

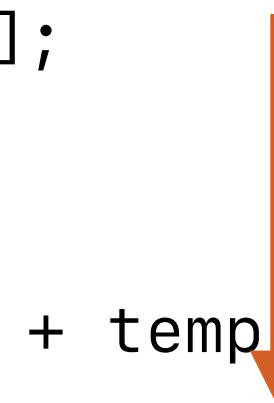
Code also has locality

```
for(uint32_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint32_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



Locality

- Spatial locality — application tends to visit nearby stuffs in the memory
 - Code — the current instruction, and then PC + 4

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

- Data — the same data can be read/write many times

Cache design principles — exploit localities

- The cache must be able to get chunks of near-by items every time to exploit spatial locality
- The cache must be able to keep a frequently used block for a while to exploit temporal locality

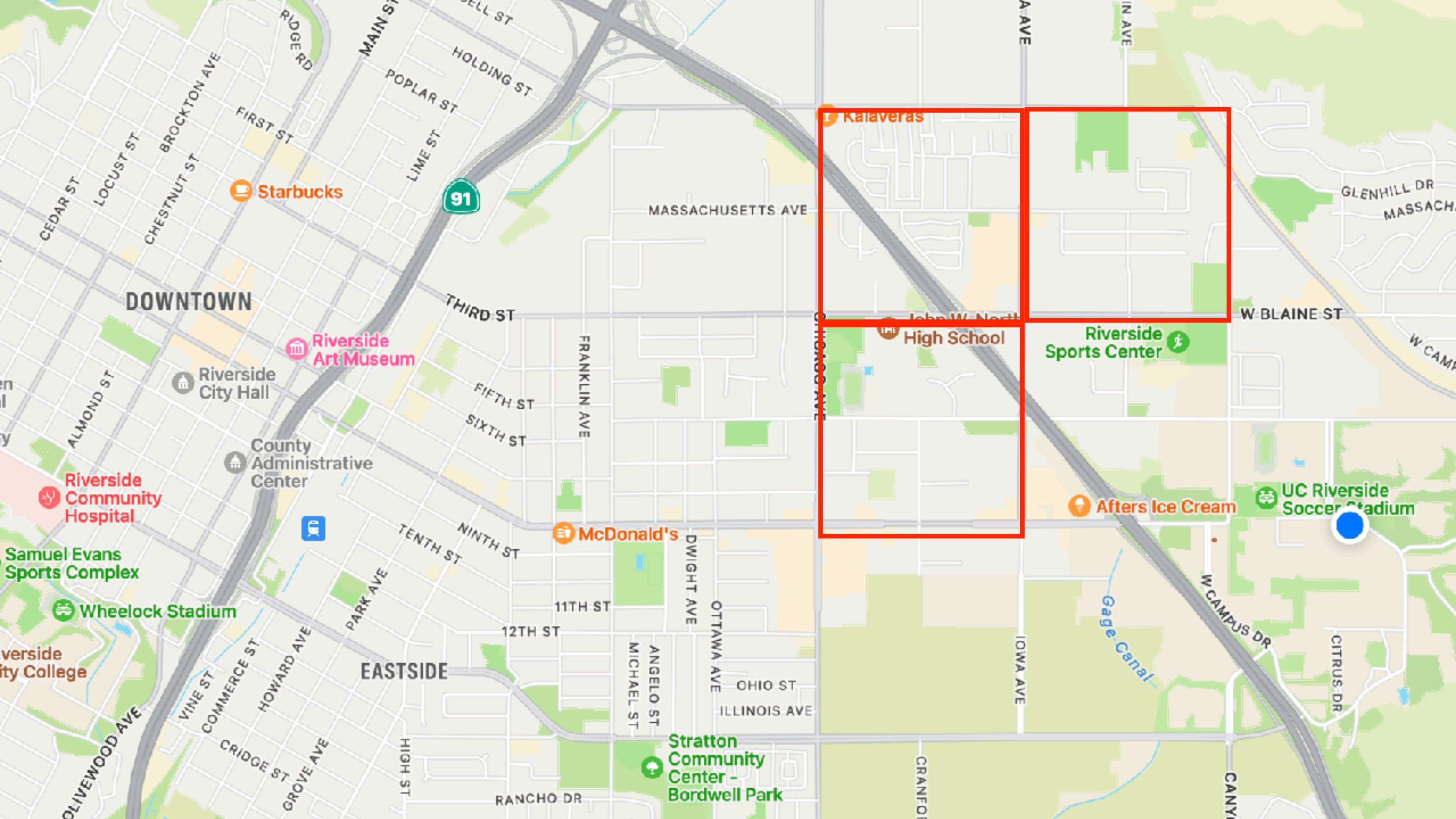
Cache design principles — exploit localities

- The cache must be able to get chunks of near-by items every time to exploit spatial locality

We need to “cache consecutive data elements” every time

— the cache should store a “block” of data

- The cache must be able to keep a frequently used block for a while to exploit temporal locality



Block and the memory space

Each byte of memory location has an “address” Partition the space with fixed-size
“blocks” (e.g., 16-byte)

0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 001A 001B 001C 001D 001E 001F

AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003A 003B 003C 003D 003E 003F

AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH	AA	BB	CC	DD	EE	FF	GG	HH
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

0040 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E 004F 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B 005C 005D 005E 005F

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 006A 006B 006C 006D 006E 006F 0070 0071 0072 0073 0074 0075 0076 0077 0078 0079 007A 007B 007C 007D 007E 007F

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

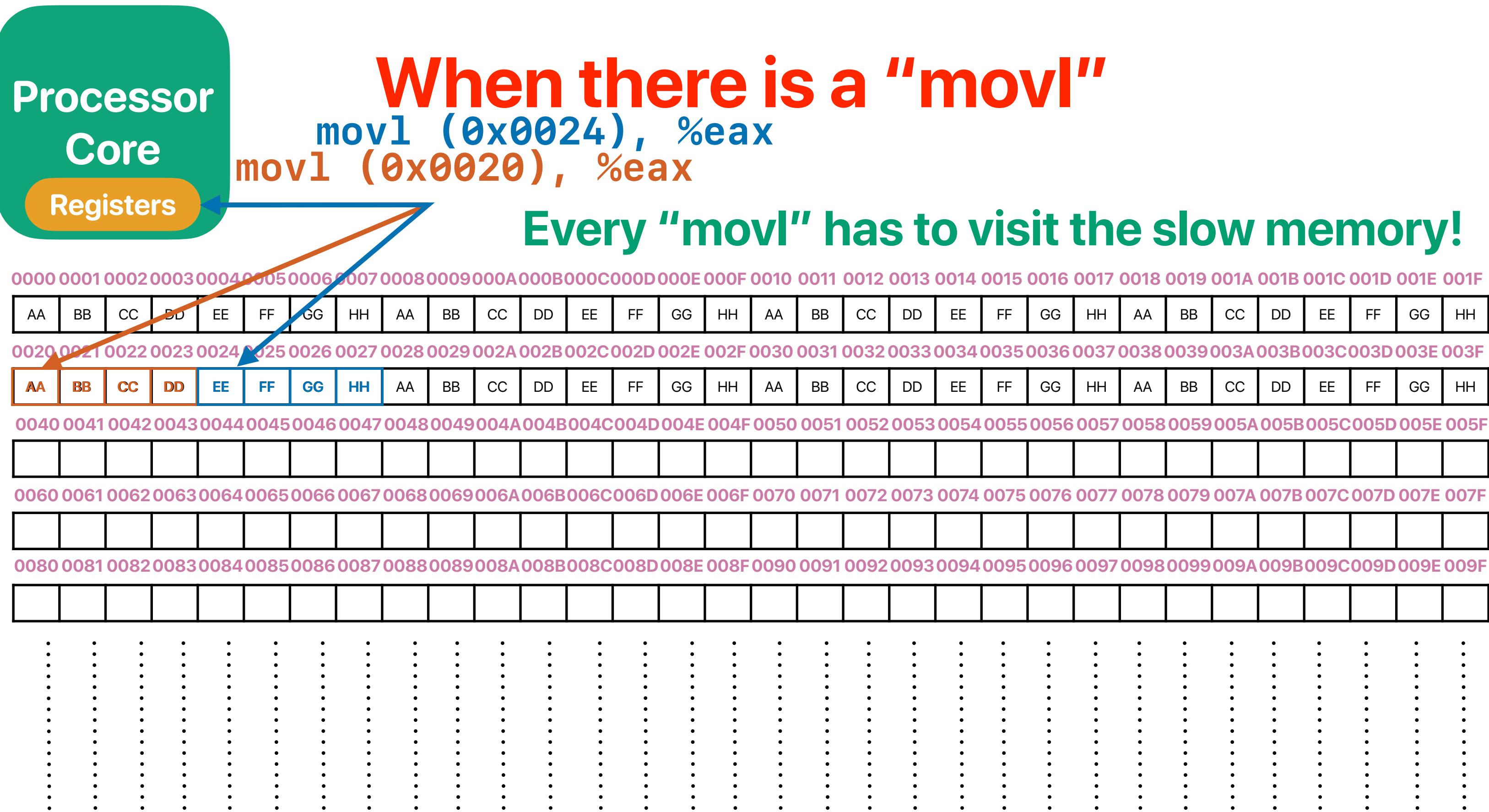
0080 0081 0082 0083 0084 0085 0086 0087 0088 0089 008A 008B 008C 008D 008E 008F 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099 009A 009B 009C 009D 009E 009F

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

.
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

FFE0 FFE1 FFE2 FFE3 FFE4 FFE5 FFE6 FFE7 FFE8 FFE9 FFEA FFEB FFEC FFED FFEE FFFF FFF0 FFF1 FFF2 FFF3 FFF4 FFF5 FFF6 FFF7 FFF8 FFF9 FFFA FFFB FFFC FFFD FFFE FFFF

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

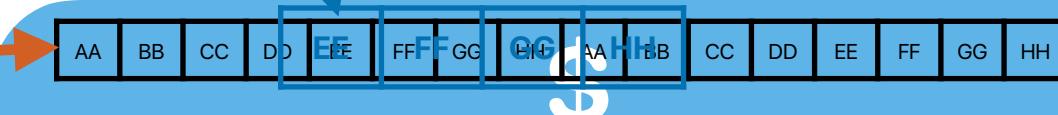


Processor
Core
Registers

Let's cache a "block"!

movl (0x0024), %eax

movl (0x0020), %eax



Caching a block helps exploit "spatial locality"!

0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F

AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH

0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003A 003B 003C 003D 003E 003F

AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH

0040 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E 004F 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B 005C 005D 005E 005F

0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 006A 006B 006C 006D 006E 006F 0070 0071 0072 0073 0074 0075 0076 0077 0078 0079 007A 007B 007C 007D 007E 007F

0080 0081 0082 0083 0084 0085 0086 0087 0088 0089 008A 008B 008C 008D 008E 008F 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099 009A 009B 009C 009D 009E 009F

Recap: Locality

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

```
for(uint32_t i = 0; i < m; i++) {  
    result = 0;  
    for(uint32_t j = 0; j < n; j++) {  
        result += matrix[i][j]*vector[j];  
    }  
    output[i] = result;  
}
```



**Simply caching one block
isn't enough**

Cache design principles — exploit localities

- The cache must be able to get chunks of near-by items every time to exploit spatial locality

We need to “cache consecutive data elements” every time

— the cache should store a “block” of data

- The cache must be able to keep a frequently used block for a while to exploit temporal locality

We need to store multiple blocks

— the cache must be able to distinguish blocks

Processor
Core
Registers

How to tell who is there?

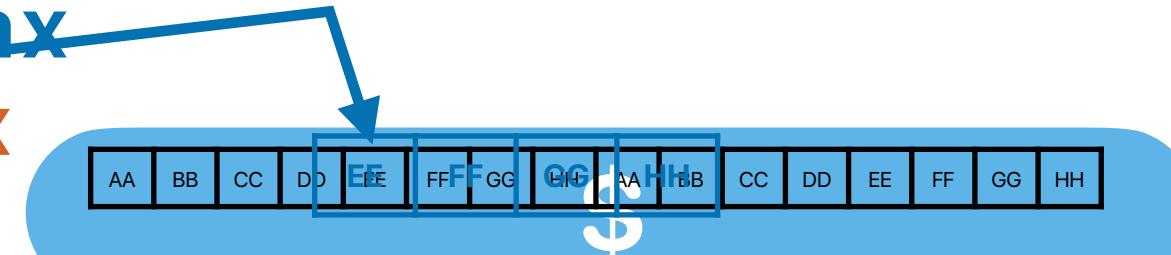
?

?	0123456789ABCDEF
?	This is CS 203:
	Advanced Compute
	r Architecture!
	This is CS 203:
	Advanced Compute
	r Architecture!
	This is CS 203:
	Advanced Compute
	r Architecture!
	This is CS 203:
	Advanced Compute
	r Architecture!
	This is CS 203:
	Advanced Compute
	r Architecture!
	This is CS 203:

Processor
Core
Registers

Let's cache a "block"!

movl (0x0024), %eax
movl (0x0020), %eax



0000 0001 0002 0003 0004 0005 0006 0007 0008 0009 000A 000B 000C 000D 000E 000F 0010 0011 0012 0013 0014 0015 0016 0017 0018 0019 001A 001B 001C 001D 001E 001F

AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH

0020 0021 0022 0023 0024 0025 0026 0027 0028 0029 002A 002B 002C 002D 002E 002F 0030 0031 0032 0033 0034 0035 0036 0037 0038 0039 003A 003B 003C 003D 003E 003F

AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH AA BB CC DD EE FF GG HH

0040 0041 0042 0043 0044 0045 0046 0047 0048 0049 004A 004B 004C 004D 004E 004F 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 005A 005B 005C 005D 005E 005F

0060 0061 0062 0063 0064 0065 0066 0067 0068 0069 006A 006B 006C 006D 006E 006F 0070 0071 0072 0073 0074 0075 0076 0077 0078 0079 007A 007B 007C 007D 007E 007F

0080 0081 0082 0083 0084 0085 0086 0087 0088 0089 008A 008B 008C 008D 008E 008F 0090 0091 0092 0093 0094 0095 0096 0097 0098 0099 009A 009B 009C 009D 009E 009F

the address in each block starts
with the same "prefix"

How to tell who is there?

the common address prefix in each block

tag array

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



Processor Core

Registers

How to tell whether

block offset
tag

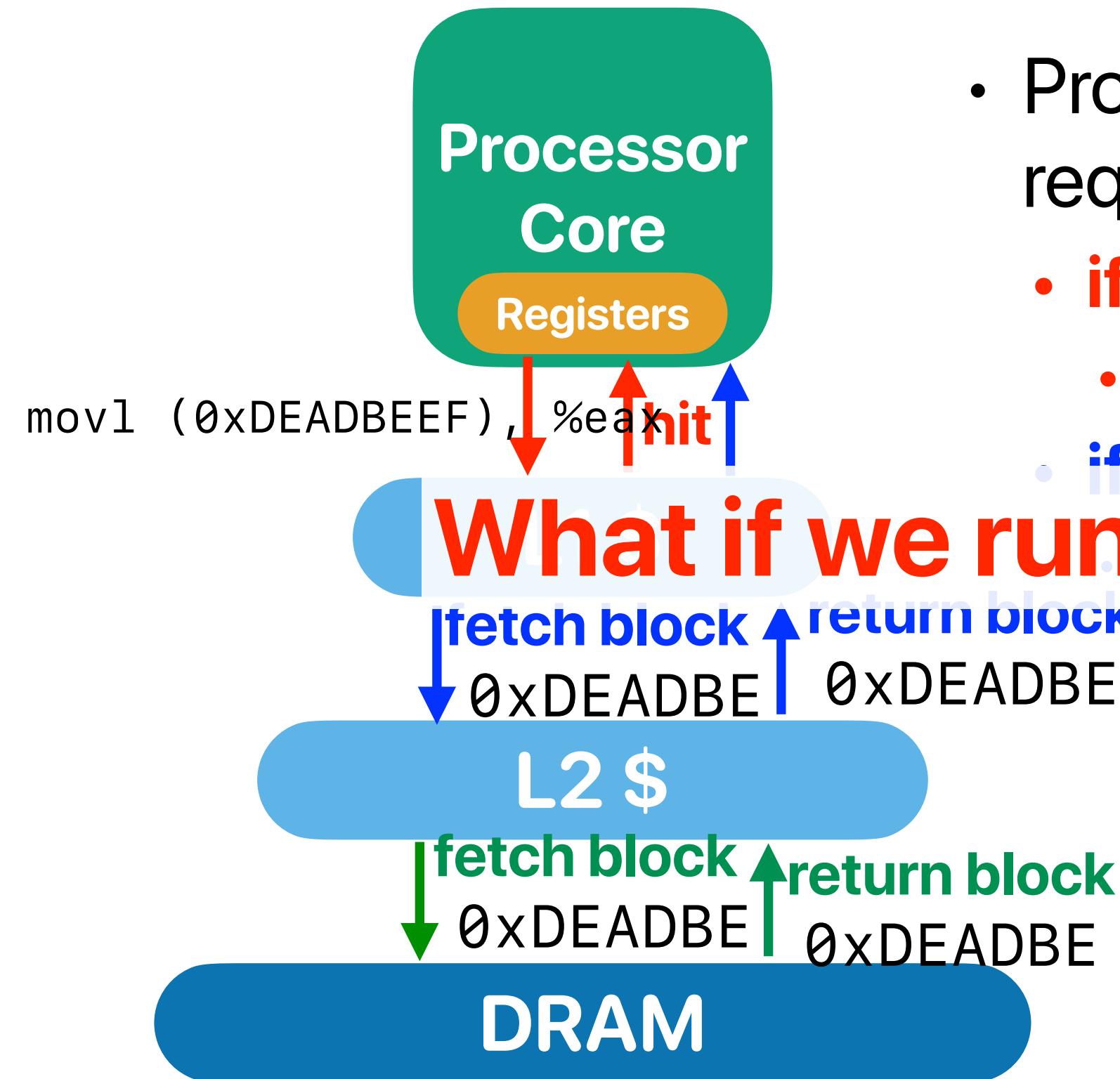
1w 0x0008

1w 0x4048

0x404 not found,
go to lower-level memory

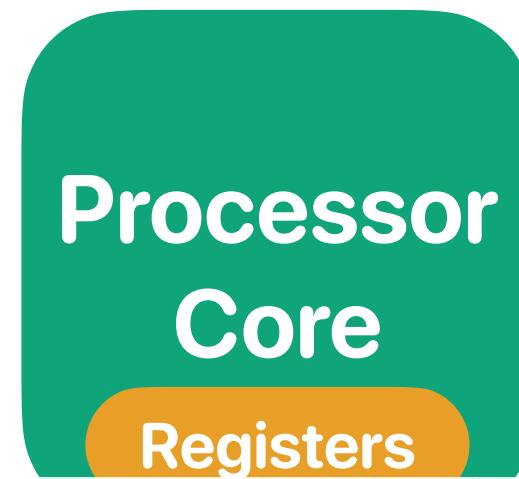
		Valid Bit	Dirty Bit	data
		tag		0123456789ABCDEF
1	1	0x000		This is CSE13:
1	1	0x001		Advanced Compute
1	0	0xF07		r Architecture!
0	1	0x100		This is CS 203:
1	1	0x310		Advanced Compute
1	1	0x450		r Architecture!
0	1	0x006		This is CS 203:
0	1	0x537		Advanced Compute
1	1	0x266		r Architecture!
1	1	0x307		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:

Processor/cache interaction



- Processor sends memory access request to L1-\$
 - if hit
 - return data
 - if miss
 - Fetch the requested block from lower-level memory hierarchy and place in the cache

Considering we have limited space in \$

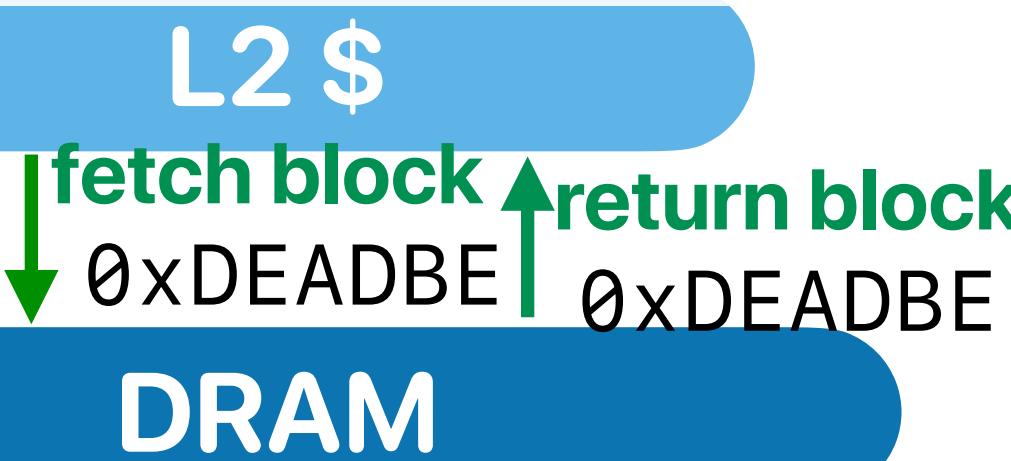


- Processor sends memory access request to L1-\$
 - if hit

movl (0xDEADBEAD) into \$

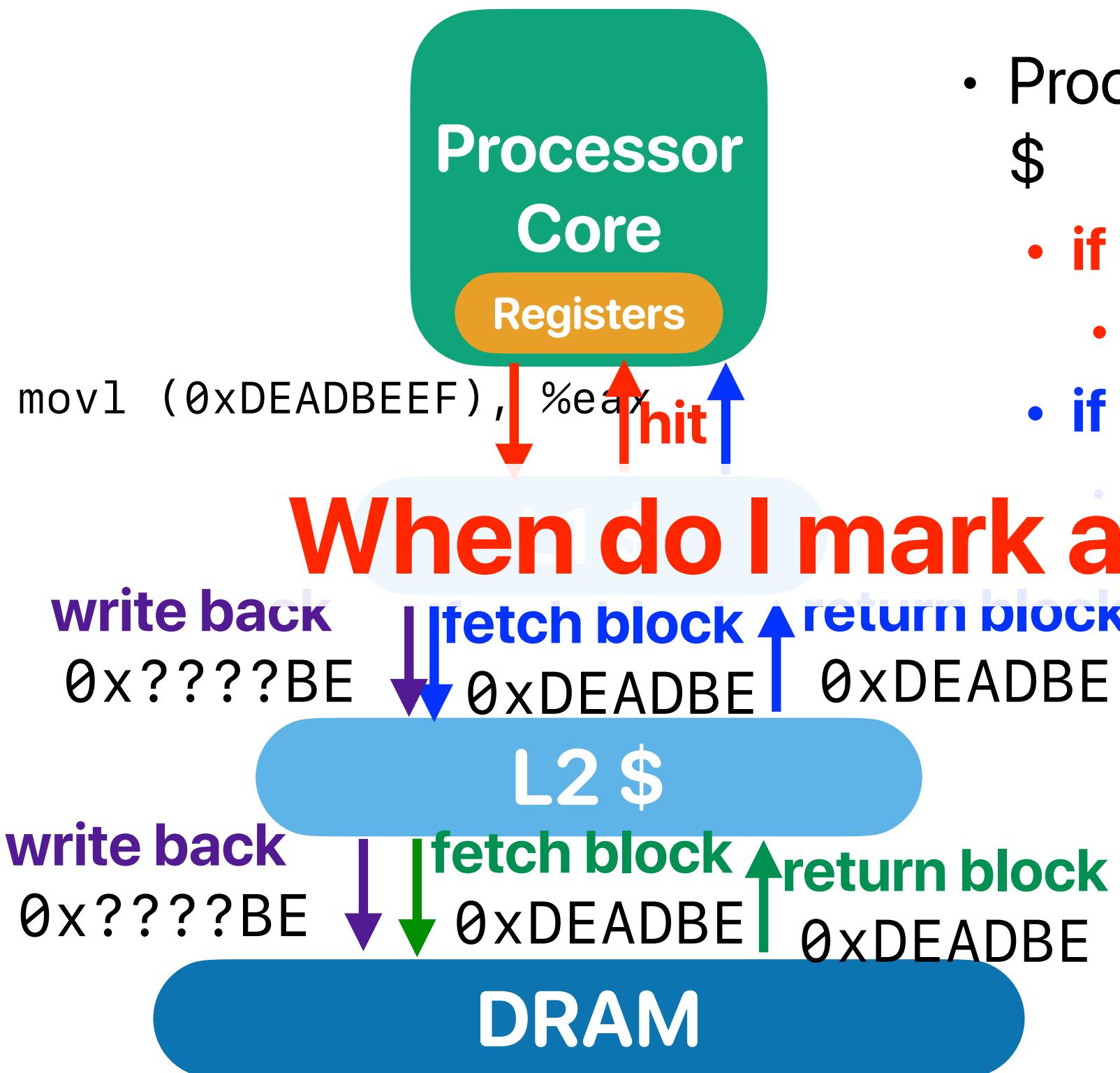
• return data
• if miss

— ignoring the update is not acceptable!



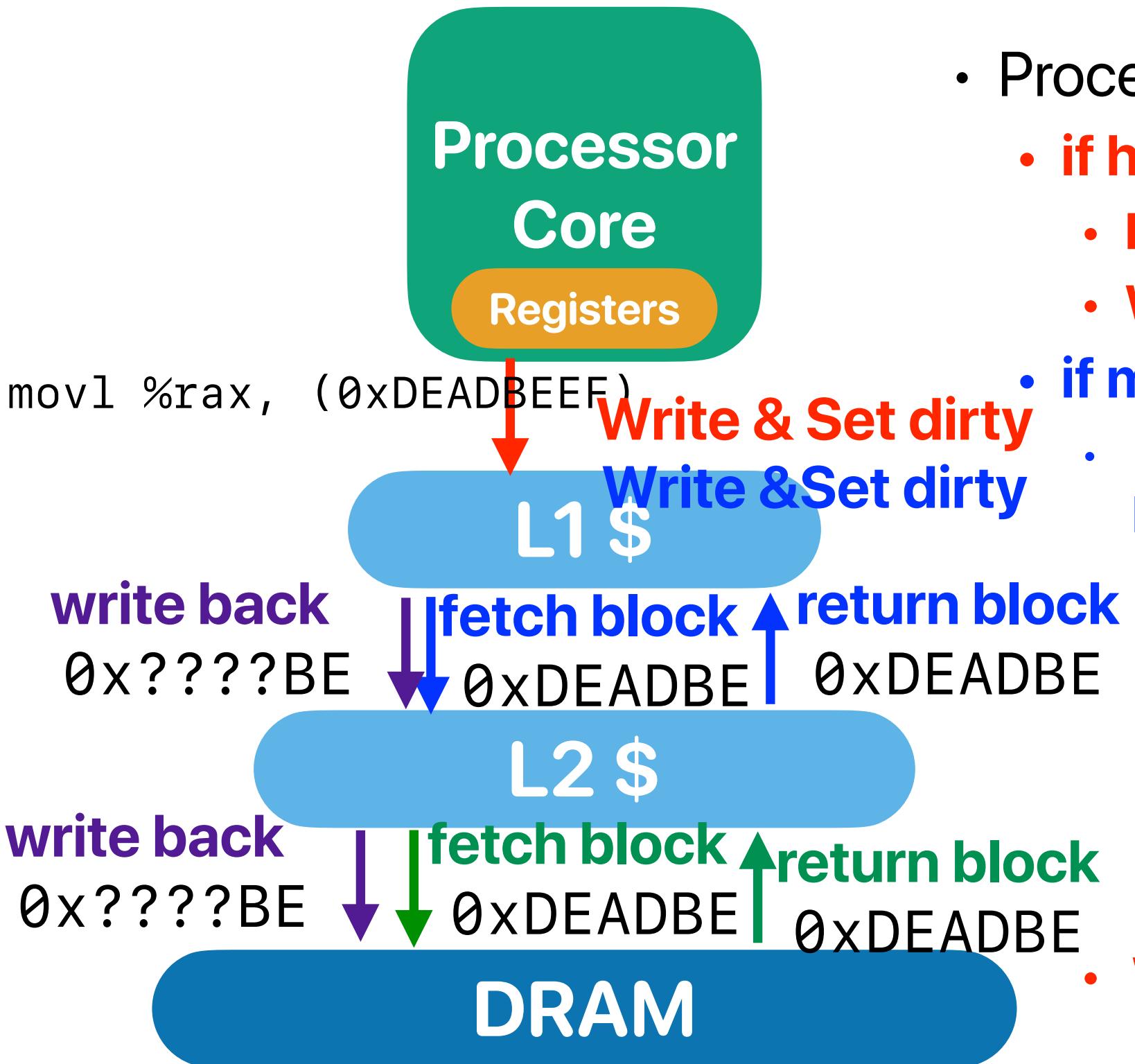
- If there an empty block — place the data there
- If NOT (most frequent case) — select a **victim block**
 - Least Recently Used (LRU) policy

Considering a victim block may be modified



- Processor sends memory access request to L1-\$
 - **if hit**
 - **return data**
 - **if miss**
 - Fetch the requesting block from lower-level memory hierarchy and place it in the cache
 - **return data**
- If there is an empty block — place the data there
- If NOT (most frequent case) — select a **victim block**
 - Least Recently Used (LRU) policy
- If the victim block is “dirty” & “valid”
 - **Write back** the block to lower-level memory hierarchy

Considering we have “writes”



- Processor sends memory access request to L1-\$
 - **if hit**
 - **Read: Return data**
 - **Write: Update “ONLY” in L1 and set DIRTY**
 - **if miss**
 - Fetch the requesting block from lower-level memory hierarchy and place in the cache and then return data
 - If there an empty block — place the data there
 - If NOT (most frequent case) — select a **victim block**
 - Least Recently Used (LRU) policy
 - If the victim block is “dirty” & “valid”
 - **Write back** the block to lower-level memory hierarchy
 - **Write: Update “ONLY” in L1 and set DIRTY**

Processor Core

Registers

How to tell whether

block offset
tag

1w 0x0008

1w 0x4048

0x404 not found,
go to lower-level memory

The complexity of search the matching tag—

$O(n)$ —will be slow if our cache size grows!

Can we search things faster?

—hash table! $O(1)$

		Valid Bit	Dirty Bit	Tag	Data
1	1	0x000		This is CSE13:	0123456789ABCDEF
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:	

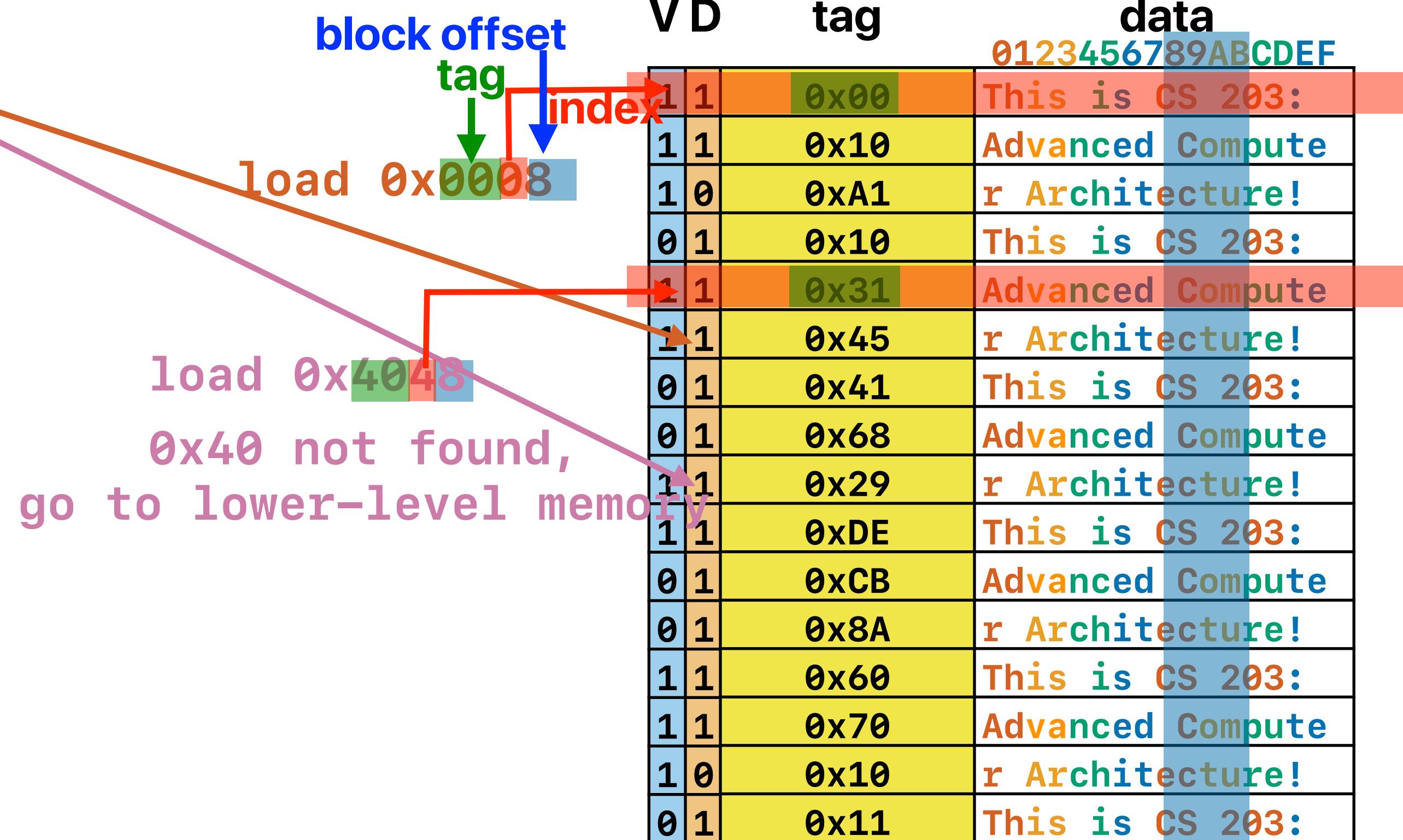
Tell if the block here can be used

Tell if the block here is modified

Processor Core

Registers

Hash-like structure — direct-mapped cache



Way-associative cache

memory address:

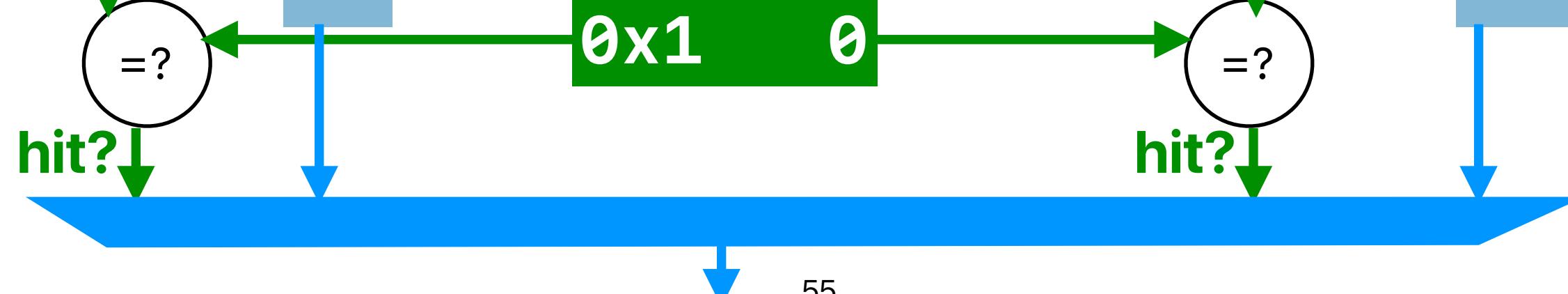
0x0 8 2 4
 set block
 index offset

memory address:

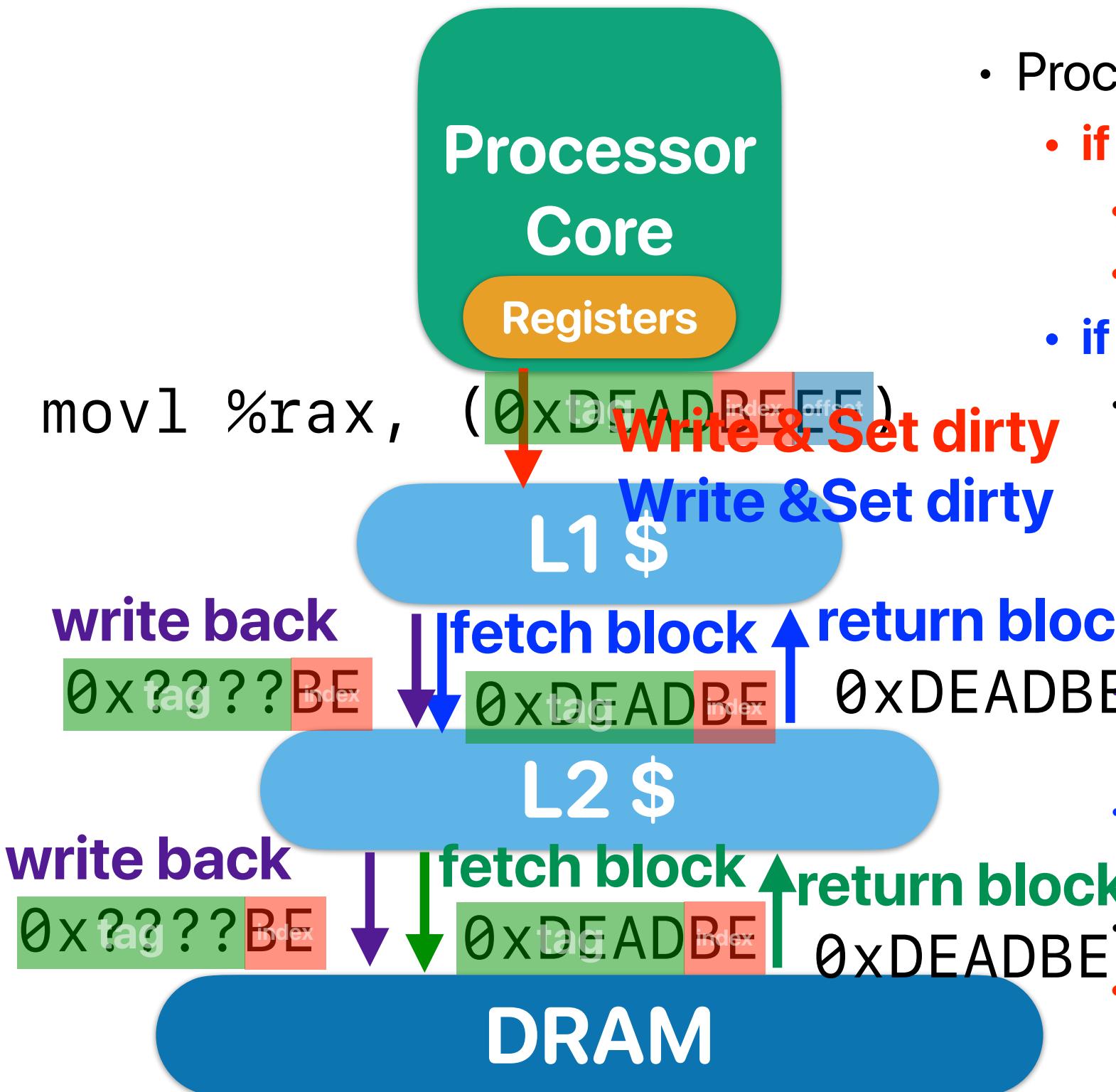
0b0000100000100100

V	D	tag	data
1	1	0x29	r Architecture!
1	1	0xDE	This is CS 203:
1	0	0x10	Advanced Compute
0	1	0x8A	r Architecture!
1	1	0x60	This is CS 203:
1	1	0x70	Advanced Compute
0	1	0x10	r Architecture!
0	1	0x11	This is CS 203:

V	D	tag	data
1	1	0x00	This is CS 203:
1	1	0x10	Advanced Compute
1	0	0xA1	r Architecture!
0	1	0x10	This is CS 203:
1	1	0x31	Advanced Compute
1	1	0x45	r Architecture!
0	1	0x41	This is CS 203:
0	1	0x68	Advanced Compute



The complete picture



- Processor sends memory access request to L1-\$
 - if hit**
 - Read - return data
 - Write - update & set **DIRTY**
 - if miss**
 - Select a victim block
 - If the target “set” is not full — select an empty/invalidated block as the victim block
 - If the target “set” is full — select a victim block using some policy
 - LRU is preferred — to exploit temporal locality!
 - If the victim block is “dirty” & “valid”
 - Write back** the block to lower-level memory hierarchy
 - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
 - If write-back or fetching causes any miss, repeat the same process
 - Present the write “ONLY” in L1 and set DIRTY**

Ways?

CPU-Z - ID : wswpbb

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name	AMD Ryzen 7 2700X		
Code Name	Pinnacle Ridge	Max TDP	105 W
Package	Socket AM4 (1331)		
Technology	12 nm	Core Voltage	1.36 V
Specification	AMD Ryzen 7 2700X Eight-Core Processor		
Family	F	Model	8
Ext. Family	17	Ext. Model	8
Instructions	MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA		

Clocks (Core #0)

Core Speed	4290.73 MHz	
Multiplier	x 43.0	
Bus Speed	99.78 MHz	
Rated FSB		

Cache

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 64 KBytes	4-way
Level 2	8 x 512 KBytes	8-way
Level 3	2 x 8192 KBytes	16-way

Selection Processor #1 | Cores 8 | Threads 16

CPU-Z Ver. 1.86.0.x64 Tools Validate Close

CPU | Caches | Mainboard | Memory | SPD | Graphics | Bench | About

Processor

Name	Intel Core i7 9700K		
Code Name	Coffee Lake	Max TDP	95.0 W
Package	Socket 1151 LGA		
Technology	14 nm	Core Voltage	0.737 V
Specification	Intel® Core™ i7-9700K CPU @ 3.60GHz (ES)		
Family	6	Model	E
Ext. Family	6	Ext. Model	9E
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX		

Clocks (Core #0)

Core Speed	4798.85 MHz	
Multiplier	x 48.0 (8 - 49)	
Bus Speed	99.98 MHz	
Rated FSB		

Cache

L1 Data	8 x 32 KBytes	8-way
L1 Inst.	8 x 32 KBytes	8-way
Level 2	8 x 256 KBytes	4-way
Level 3	12 MBytes	12-way

Selection Socket #1 | Cores 8 | Threads 8

Way-associative cache

memory address:

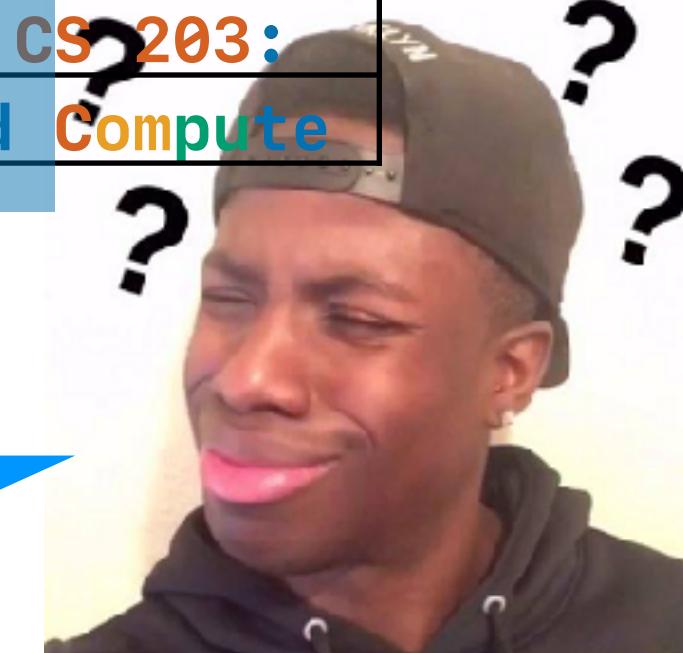
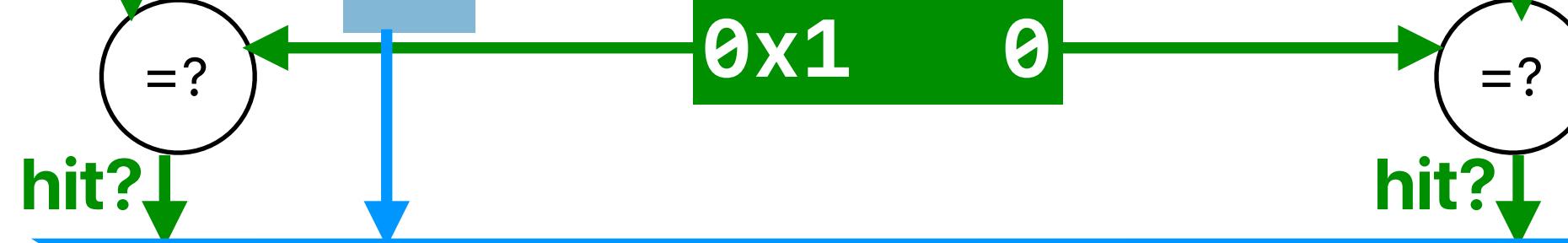
0x0 8 2 4
 set block
 index offset

memory address:

0b0000100000100100

V	D	tag	data
1	1	0x29	r Architecture!
1	1	0xDE	This is CS 203:
1	0	0x10	Advanced Compute
0	1	0x8A	r Architecture!
1	1	0x60	This is CS 203:
1	1	0x70	Advanced Compute
0	1	0x10	r Architecture!
0	1	0x11	This is CS 203:

V	D	tag	data
1	1	0x00	This is CS 203:
1	1	0x10	Advanced Compute
1	0	0xA1	r Architecture!
0	1	0x10	This is CS 203:
1	1	0x31	Advanced Compute
1	1	0x45	r Architecture!
0	1	0x41	This is CS 203:
0	1	0x68	Advanced ?



Announcement

- Reading quiz #3 due **next Tuesday** before the lecture
 - Please do read the textbook before/while taking the quiz
 - We take the “average”
- Assignment #2 due tonight
 - Review the “demo”s of previous lectures if you need inspiration for programming assignment
- Assignment #3 will be up tomorrow. Due in 2 weeks

Computer Science & Engineering

203

つづく

