Memory Hierarchy (1): Inside Out the Computer Memory

Hung-Wei Tseng

Disney+

Disney · PIXAR

INSIDE OUT

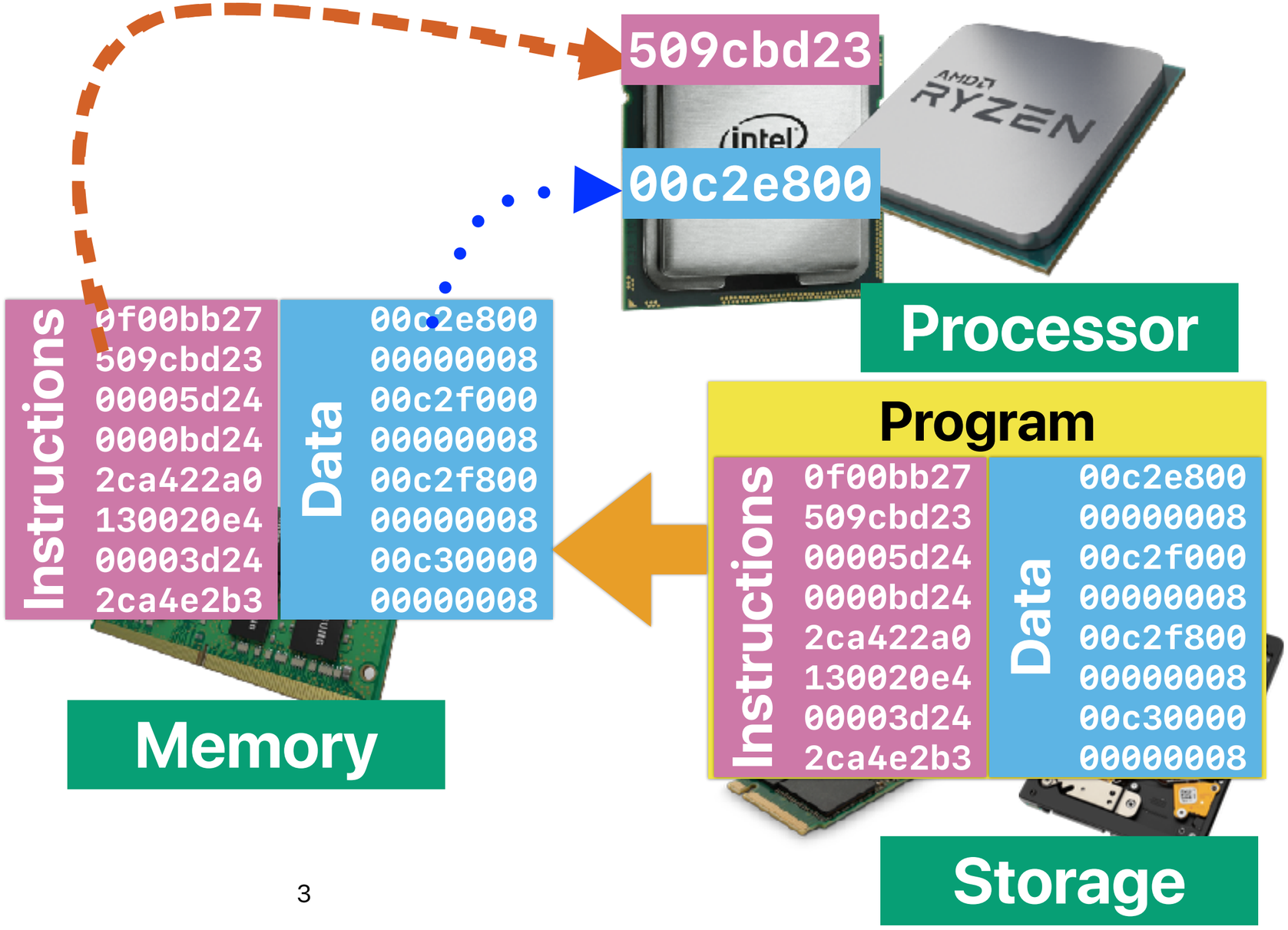
GET DISNEY+

TRAILER

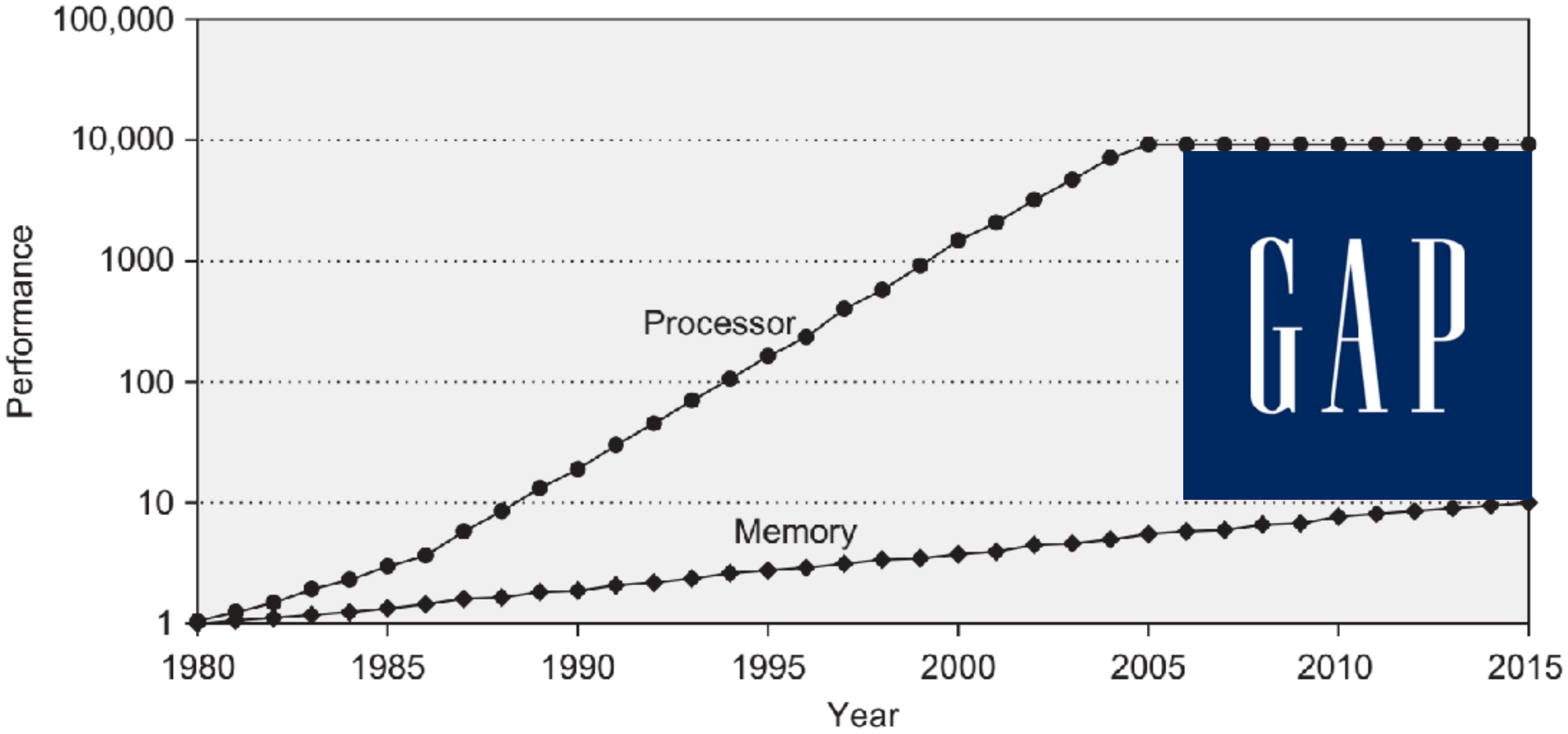
PG 2015 • 1h 35m • Coming of age, Family, Animation

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 Neumman Architecture



Recap: Performance gap between Processor/Memory



Outline

- The Basic Idea behind Memory Hierarchy
- How cache works

Modern DRAM performance

SDRAM Data Rate MT/s	Bandwidth GB/s	CAS (clk)	Latency (ns)	Year	DDR 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	

https://www.anandtech.com/show/16143/insights-into-ddr5-subtimings-and-latencies

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

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

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

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

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

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

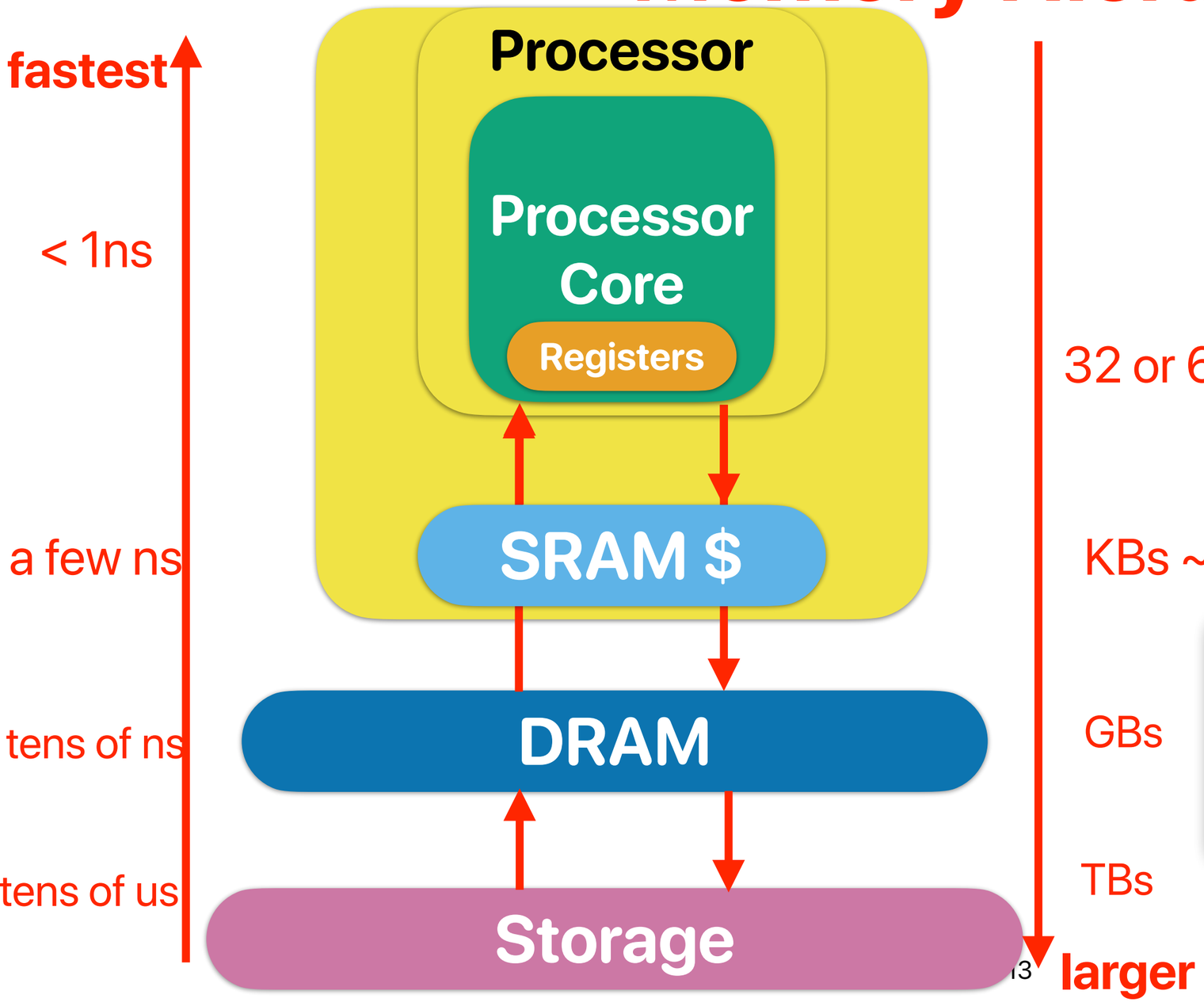
Don't forget, instructions are also from "memory"

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

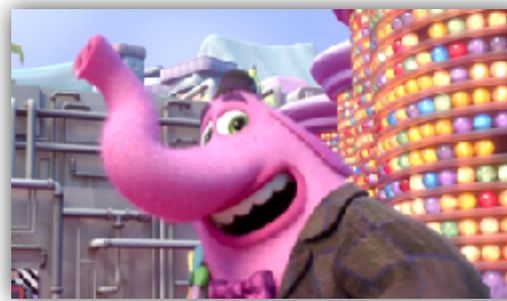


32 or 64 words

KBs ~ MBs

GBs

TBs



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

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)?

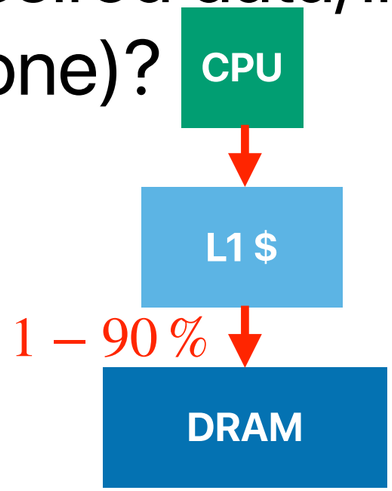
  A. 6

  B. 8

  C. 10

  D. 12

  E. 67



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

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

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

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

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	Stepping	2
Ext. Family	17	Ext. Model	8	Revision	PiR-B2
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	Stepping	C
Ext. Family	6	Ext. Model	9E	Revision	P0
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX		

Clocks (Core #0)

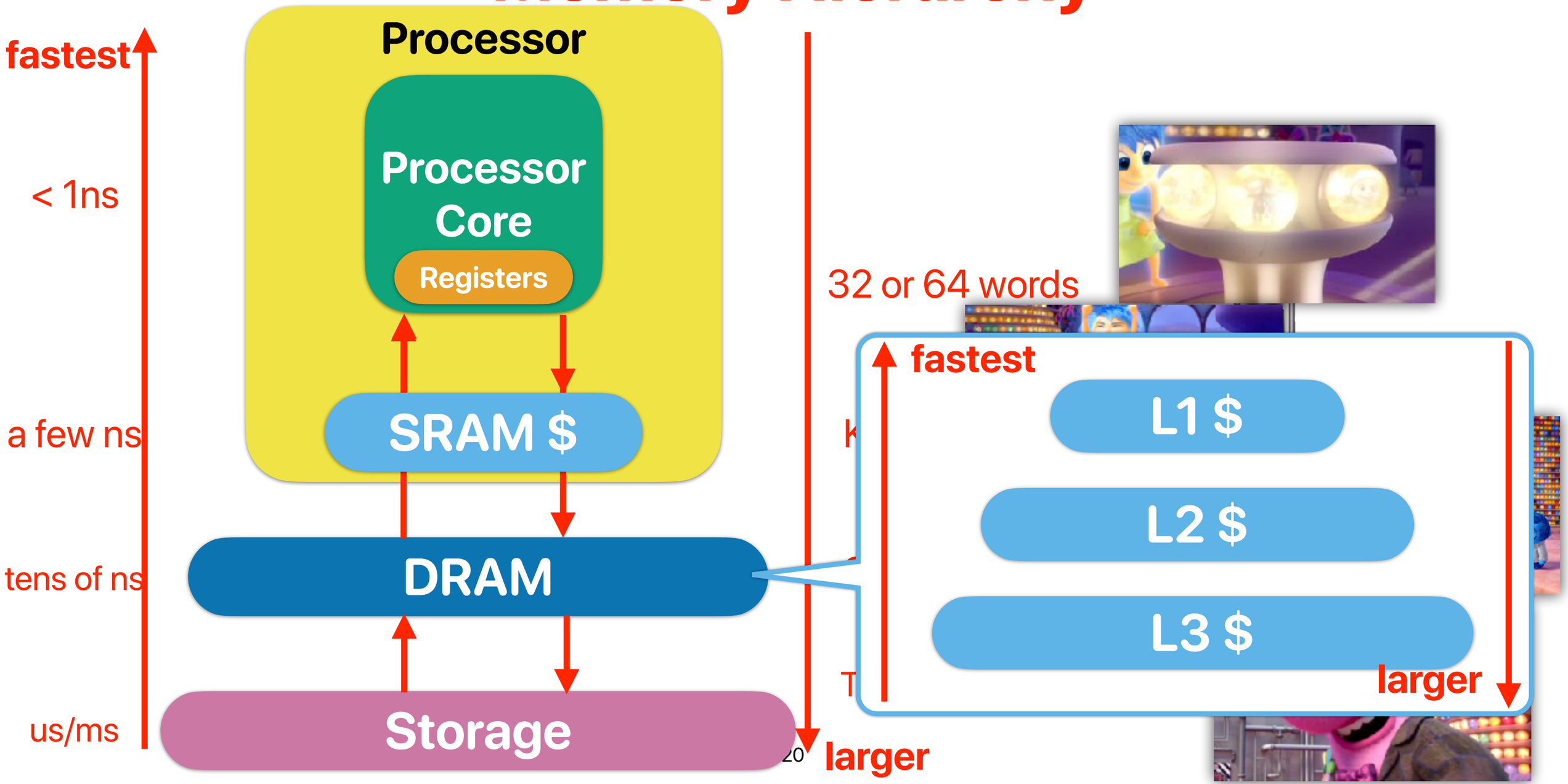
Core Speed	4798.85 MHz
Multiplier	x 48.0 (8 - 49)
Bus Speed	99.98 MHz
Rated FSB	

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



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

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)?

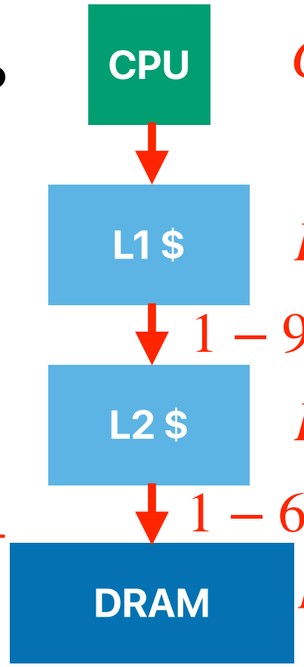
  A. 6
  B. 8
  C. 10
  D. 12
  E. 67

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



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

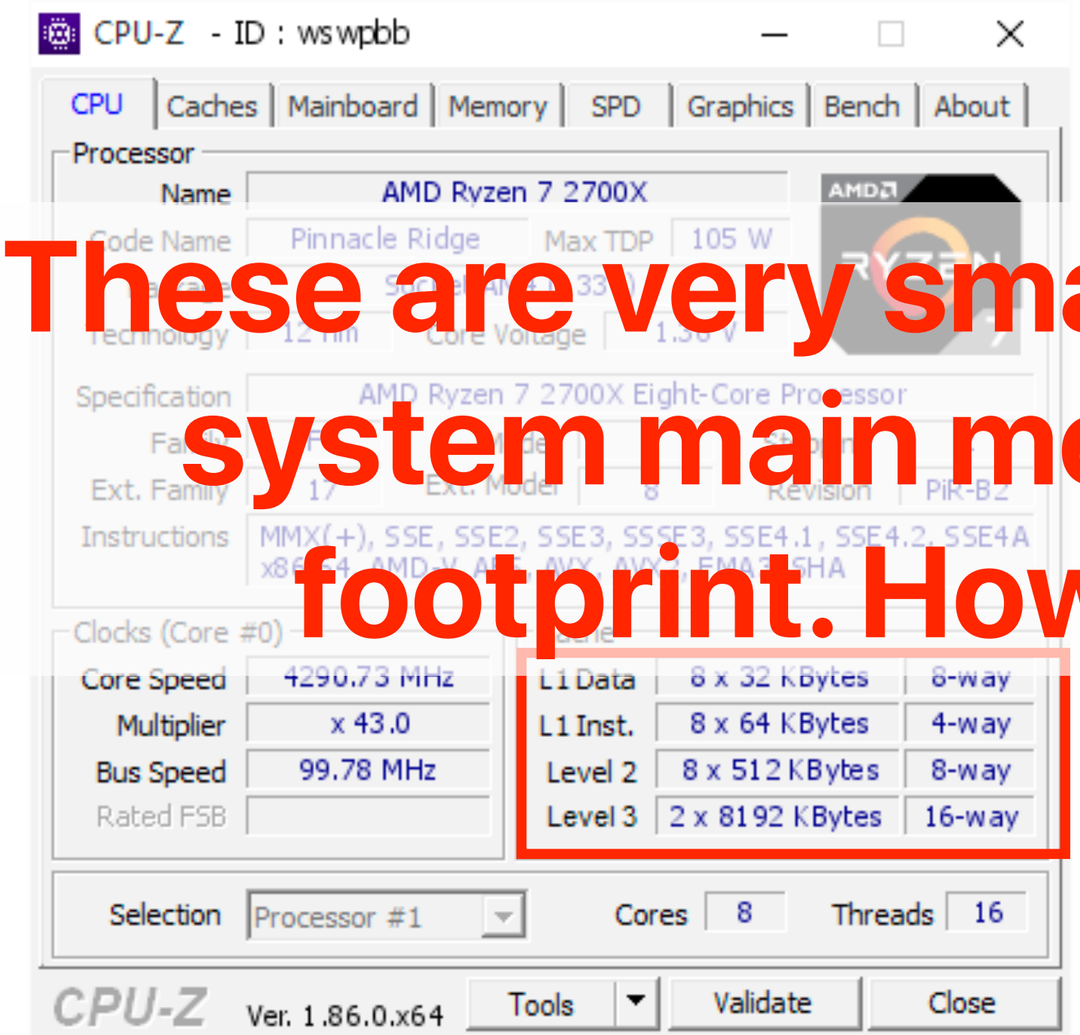
$$Each\ L1_{\$}\ access = \frac{0.5}{0.25} = 2\ cycles$$

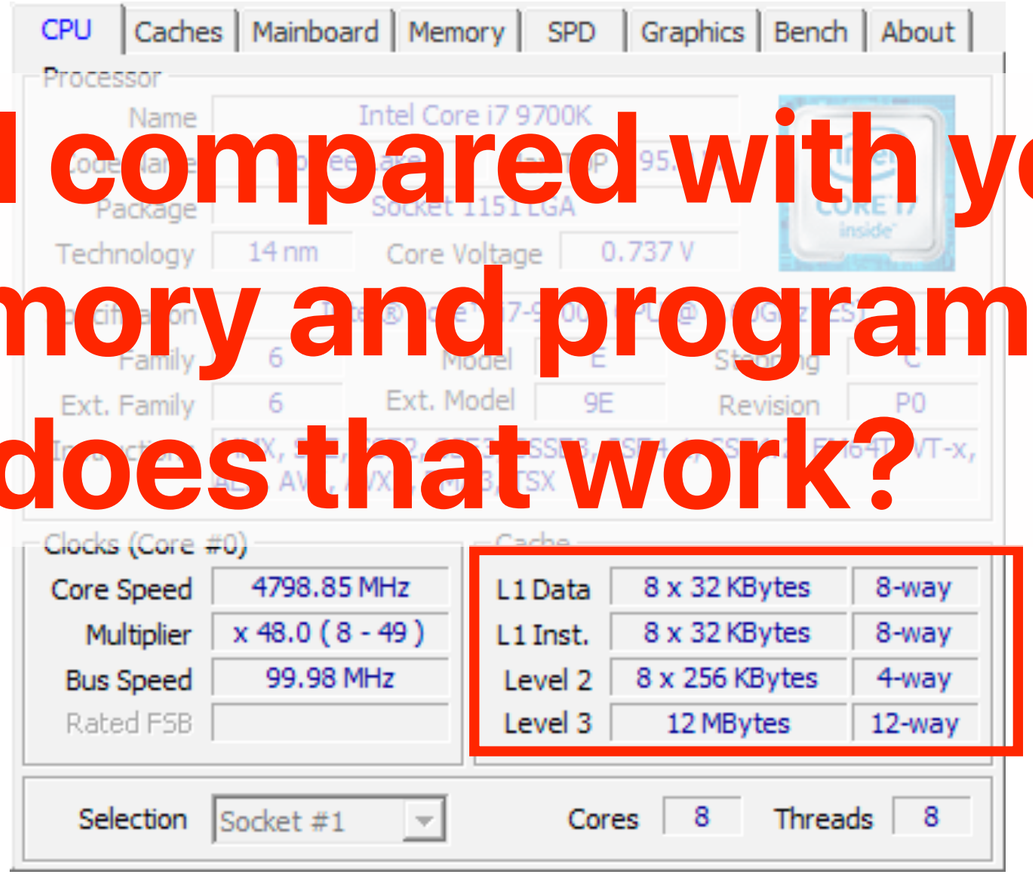
$$Each\ L2_{\$}\ access = \frac{5}{0.25} = 20\ cycles$$

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

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

L1? L2? L3?





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

Why adding small SRAMs would work?

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

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 te

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;
}
```

```
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

repeat many times — 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 data, and 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

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

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

Architecting the Cache

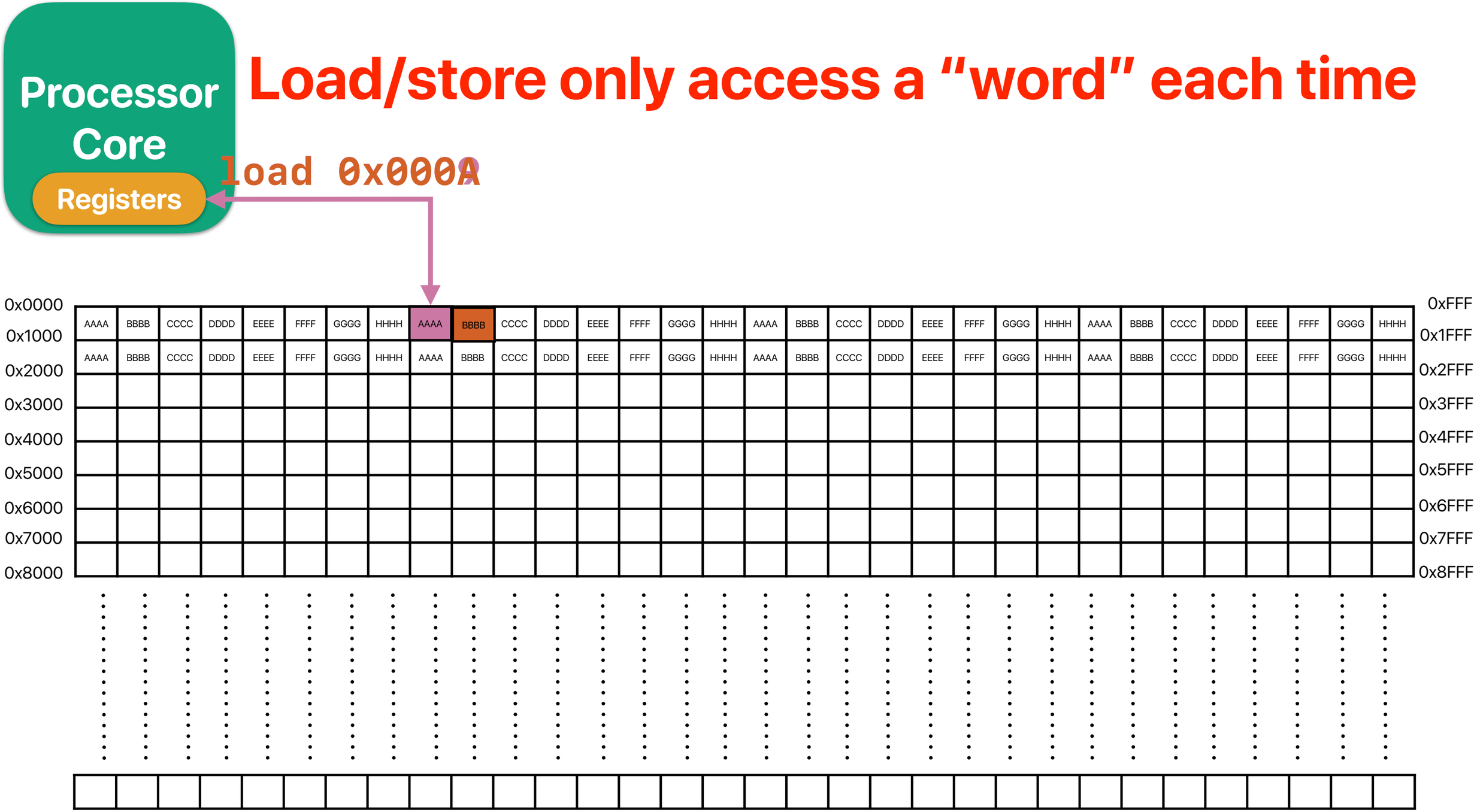
Cache design principles — exploit localities

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

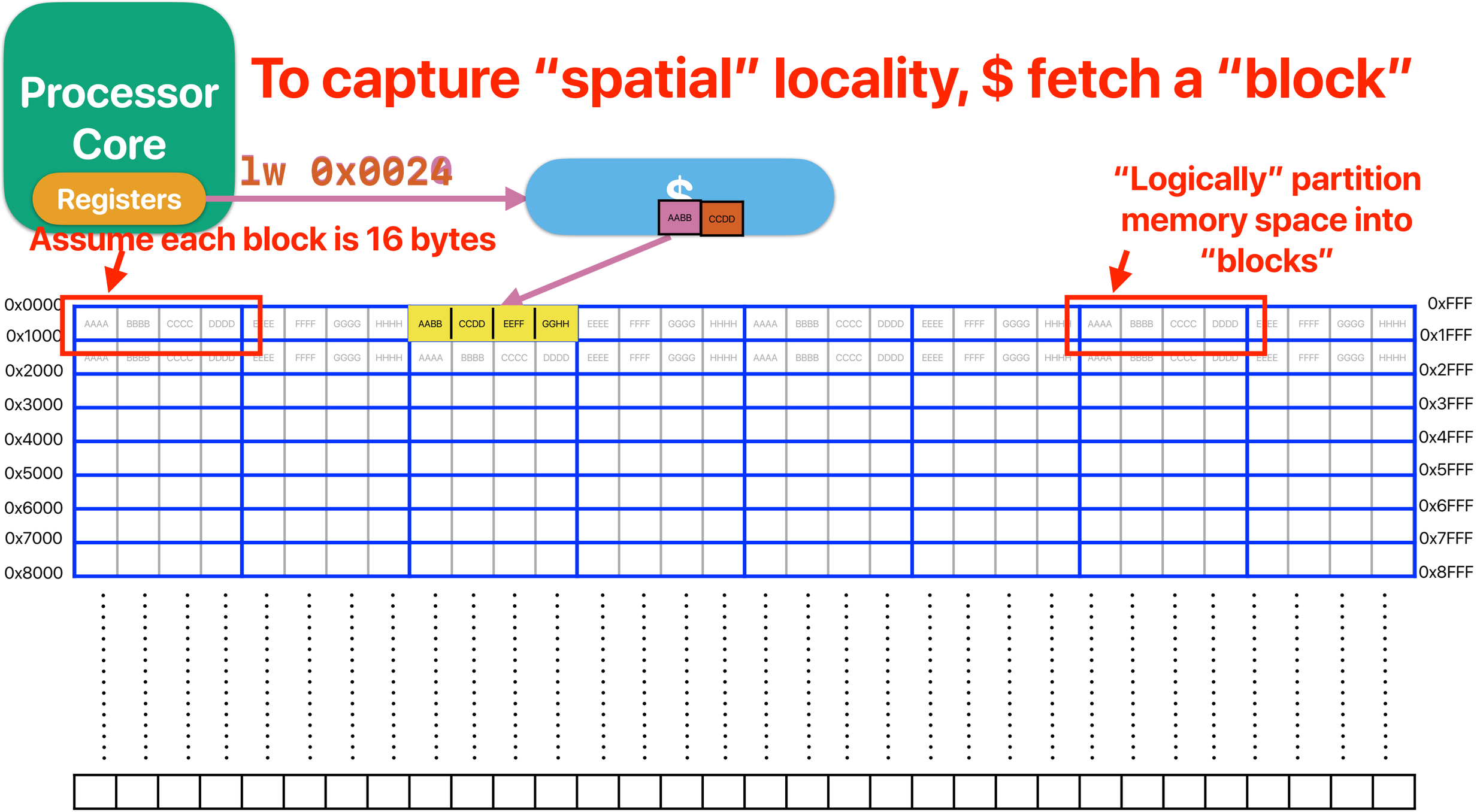
  We need to "cache consecutive data elements" every time — the cache should store a "block" of data

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

Load/store only access a “word” each time



To capture "spatial" locality, $ fetch a "block"



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

What's a block?

the offset of the byte within a block

the data in memory

```
0123456789ABCDEF
This is CSE142:
Advanced Compute
```

```
0x0000, 0x0001, 0x0002, …, 0x000F
0x0010, 0x0011, 0x0012, …, 0x001F
```

the byte addresses of each byte in the block

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

How to tell who is there?

tag	0x000 0x001 0x002 0x003 0x004 0x005 0x006 0x007 0x008 0x009 0x00A 0x00B 0x00C 0x00D 0x00E 0x00F 0123456789ABCDEF
0x000	This is CSE142:
	Advanced Compute
	r Architecture!
	This is CSE142:
	Advanced Compute
	r Architecture!
	This is CSE142:
	Advanced Compute
	r Architecture!
	This is CSE142:
	Advanced Compute
	r Architecture!
	This is CSE142:
	Advanced Compute
	r Architecture!
	This is CSE142:

How to tell who is there?

the common address prefix in each block

tag array	0123456789ABCDEF
0x000	This is CSE142:
0x001	Advanced Compute
0xF07	r Architecture!
0x100	This is CSE142:
0x310	Advanced Compute
0x450	r Architecture!
0x006	This is CSE142:
0x537	Advanced Compute
0x266	r Architecture!
0x307	This is CSE142:
0x265	Advanced Compute
0x80A	r Architecture!
0x620	This is CSE142:
0x630	Advanced Compute
0x705	r Architecture!
0x216	This is CSE142:

How to tell w

Processor Core

Registers

block offset

tag

lw 0x0008

lw 0x4048

0x404 not found,
go to lower-level memory

Tell if the block here can be used

Tell if the block here is modified

Valid Bit	Dirty Bit	tag	data 0123456789ABCDEF
1	1	0x000	This is CSE1 2:
1	1	0x001	Advanced Compute
1	0	0xF07	r Architecture!
0	1	0x100	This is CSE142:
1	1	0x310	Advanced Compute
1	1	0x450	r Architecture!
0	1	0x006	This is CSE142:
0	1	0x537	Advanced Compute
1	1	0x266	r Architecture!
1	1	0x307	This is CSE142:
0	1	0x265	Advanced Compute
0	1	0x80A	r Architecture!
1	1	0x620	This is CSE142:
1	1	0x630	Advanced Compute
1	0	0x705	r Architecture!
0	1	0x216	This is CSE142:

The complexity of search the matching tag—

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

Can we search things faster?

—hash table! $O(1)$

Hash-like structure — direct-mapped cache

block offset

tag

index

load 0x0008

load 0x4048

0x40 not found,
go to lower-level memory

V	D	tag	data (0123456789ABCDEF)
1	1	0x00	This is CSE142:
1	1	0x10	Advanced Compute
1	0	0xA1	r Architecture!
0	1	0x10	This is CSE142:
1	1	0x31	Advanced Compute
1	1	0x45	r Architecture!
0	1	0x41	This is CSE142:
0	1	0x68	Advanced Compute
1	1	0x29	r Architecture!
1	1	0xDE	This is CSE142:
0	1	0xCB	Advanced Compute
0	1	0x8A	r Architecture!
1	1	0x60	This is CSE142:
1	1	0x70	Advanced Compute
1	0	0x10	r Architecture!
0	1	0x11	This is CSE142:

Way-associative cache

memory address: 0x0 8 2 4

set block

tag index offset

memory address: 0b00001000010100100

V	D	tag	data
1	1	0x29	r Architecture!
1	1	0xDE	This is CSE142:
1	0	0x10	Advanced Compute
0	1	0x8A	r Architecture!
1	1	0x60	This is CSE142:
1	1	0x70	Advanced Compute
0	1	0x10	r Architecture!
0	1	0x11	This is CSE142:

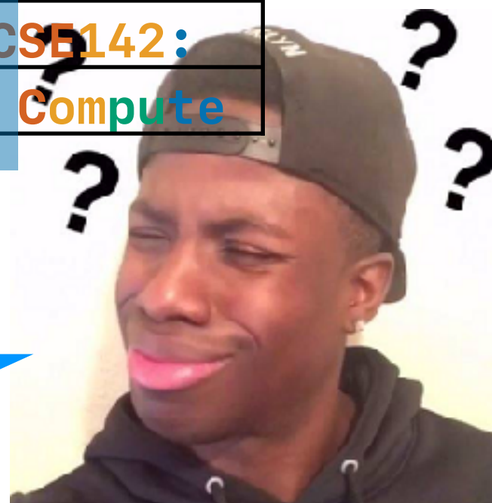
V	D	tag	data
1	1	0x00	This is CSE142:
1	1	0x10	Advanced Compute
1	0	0xA1	r Architecture!
0	1	0x10	This is CSE142:
1	1	0x31	Advanced Compute
1	1	0x45	r Architecture!
0	1	0x41	This is CSE142:
0	1	0x68	Advanced Compute

Set

0x1 0

=? =?

hit? hit?



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	Stepping	2
Ext. Family	17	Ext. Model	8	Revision	PiR-B2
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	Stepping	C
Ext. Family	6	Ext. Model	9E	Revision	P0
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

Computer
Science &
Engineering

つづく