Cache & CPU Performance

Huỳnh Ngọc Thi
so61pi.re@gmail.com

Agenda

1. Questions
2. What is cache?
3. Why CPU needs cache?
4. How does cache work?
5. Cache line & MESI protocol
6. False sharing
7. Answer questions
8. Recommendations
9. Demo

Questions – #1

- Which one is faster?

```c
int matrix[R][C] = {0};
int sum = 0;
```

```c
// row-major
for (int r = 0; r < R; ++r)
  for (int c = 0; c < C; ++c)
    sum += matrix[r][c];
```

```c
// column-major
for (int c = 0; c < C; ++c)
  for (int r = 0; r < R; ++r)
    sum += matrix[r][c];
```

Questions – #2

- Which one is faster?

```
std::vector<int> v(BIG);
std::forward_list<int> l(BIG);

auto va = std::accumulate(begin(v), end(v), 0);
auto la = std::accumulate(begin(l), end(l), 0);
```

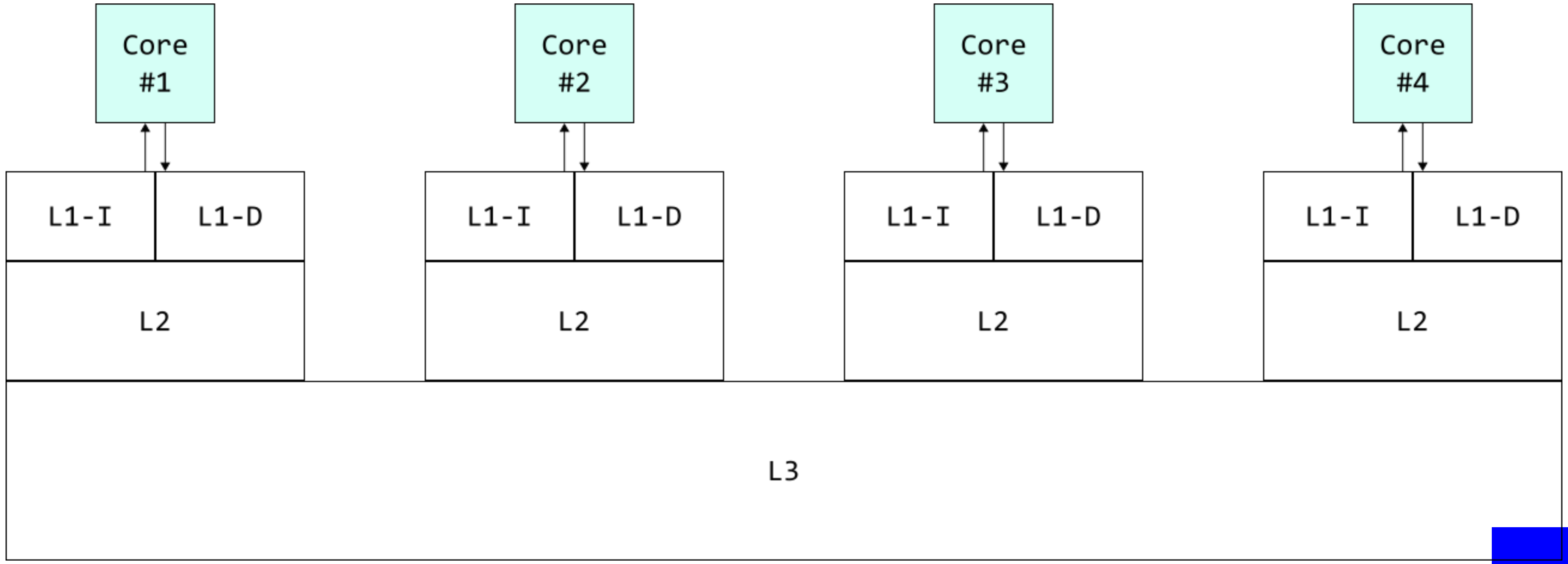
Questions – #3

- How many cycles does the CPU consume to execute this code?

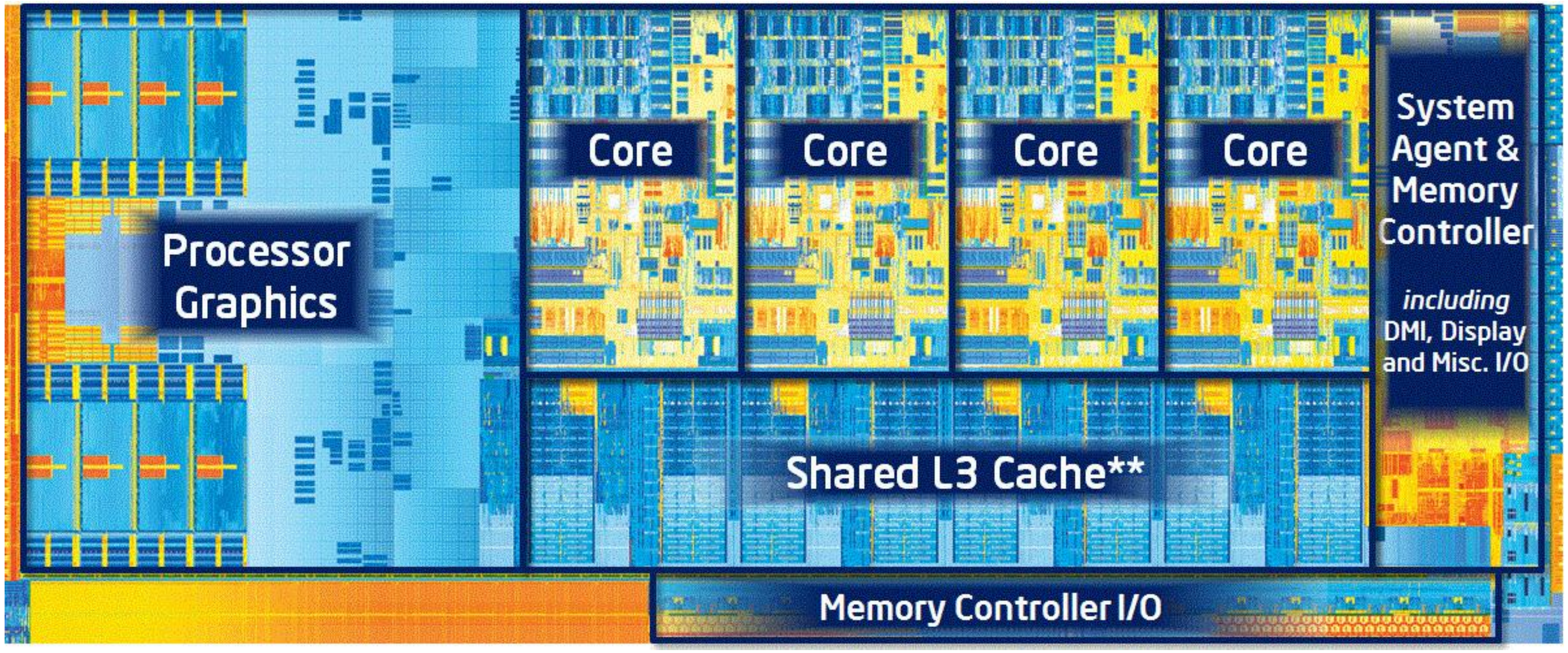
```
mov eax, dword ptr [0x12345678]
```

What is cache?

- Cache is small high speed memory usually Static RAM (SRAM) that contains the most recently accessed pieces of main memory.
- From: http://download.intel.com/design/intarch/papers/cache6.pdf



What is cache?



Ivy-Bridge Die

L1-I: 32K
L1-D: 32K
L2: 256K
L3: 8M

Why CPU needs cache?

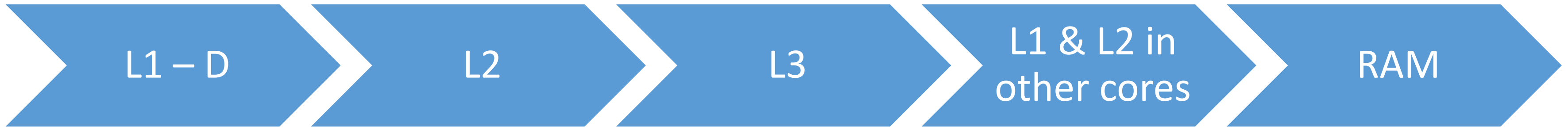
- A trip to RAM takes a very long time.

Latency	CoreSpeed = 2.5GHz
L1 – 4 cycles	1.6 ns
L2 – 12 cycles	4.8 ns
L3 – 26-31 cycles	10.4-12.4 ns
L2 & L1 in other cores	
Clean hit – 43 cycles	17.2 ns
Dirty hit – 60 cycles	24 ns

  - Main memory – 100 ns
  - From: Intel Optimization Reference Manual + internet.

How does cache work – Read

- Cache-hit
  - Requested data is currently in the cache.
  - Returned back to CPU quickly.
- Cache-miss
  - Requested data is not in the cache.
  - Must be loaded from RAM (a long trip).
- Lookup rule (Intel processors)

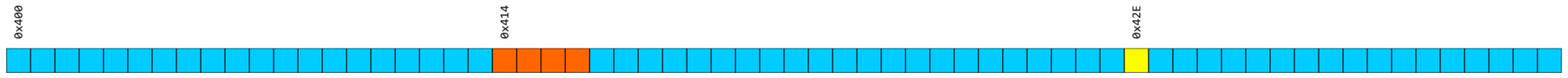


How does cache work – Write

- Write-back
  - Cache acts like a buffer.
  - Data is written to cache, then transferred to RAM later.
  - Usually comes with write-allocate.
    - Write-allocate: Data at the missed-write location is loaded to cache.
- Write-through
  - Data is written immediately to RAM.
  - Usually comes with write-no-allocate.
    - Write-no-allocate: Data at the missed-write location is not loaded to cache.
- Intel processors use write-back policy.

Cache line

- Program tends to use adjacent data.
  - local variables.
  - array elements.
- A trip to RAM will load a cache line.
  - A cache line is 64 bytes, aligned.



MESI protocol

- Each cache line has one of the following states:
  - Exclusive
    - Set when only one Core has the cache line & it's *clean*.
  - Shared
    - Set when more than one core has the cache line & it's *clean*.
  - Modified
    - Set when a write to the cache line happens, makes it *dirty*.
  - Invalid
    - Set when one of the cores writes to a Shared cache line.

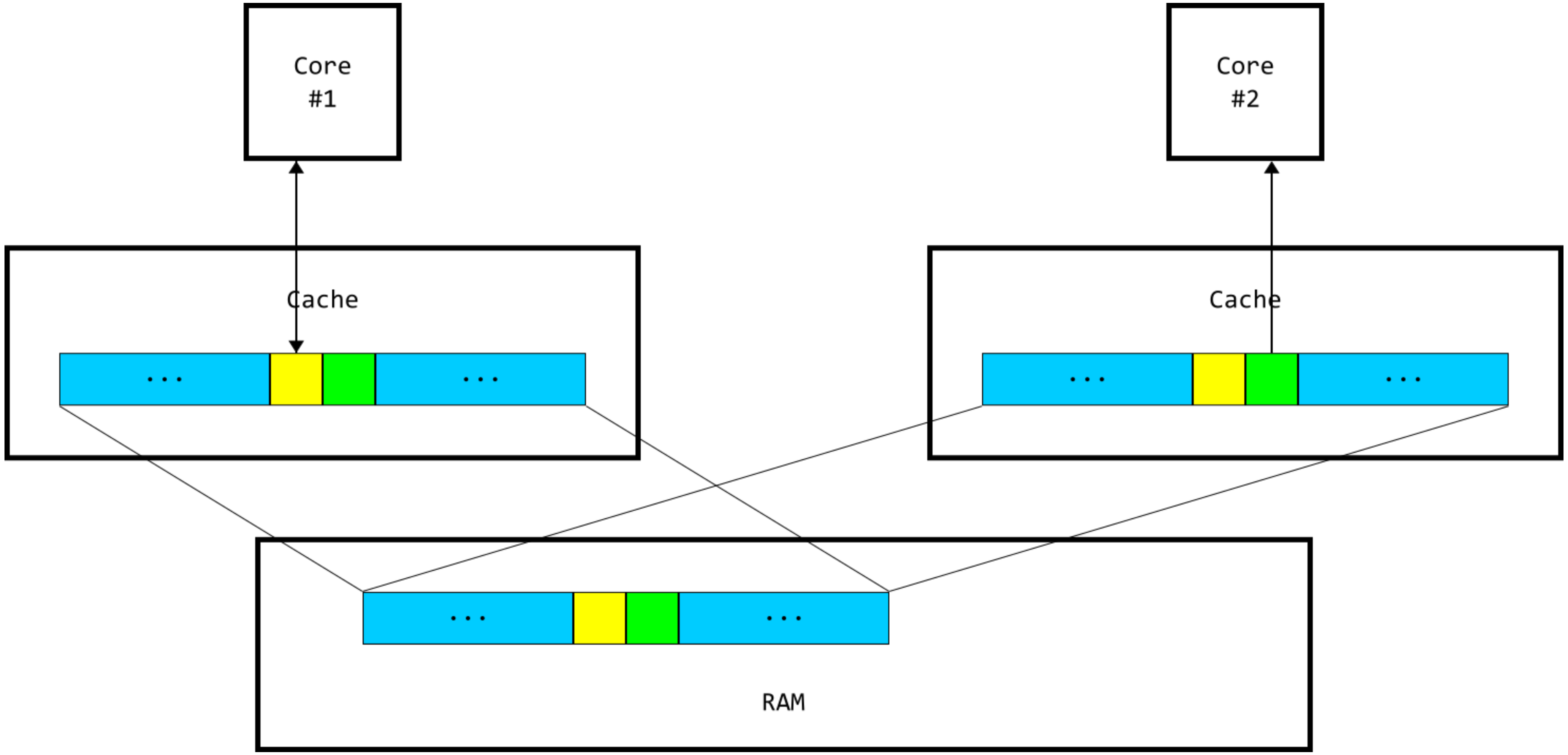
False sharing

```
struct { int x; int y; } f;
```

```
void core_1() {
  for (int i = 0; i++ < 100;)
    ++f.x;
}
```

```
int core_2() {
  int s = 0;
  for (int i = 0; i++ < 100;)
    s += f.y;
}
```

False sharing



Answer questions – #1

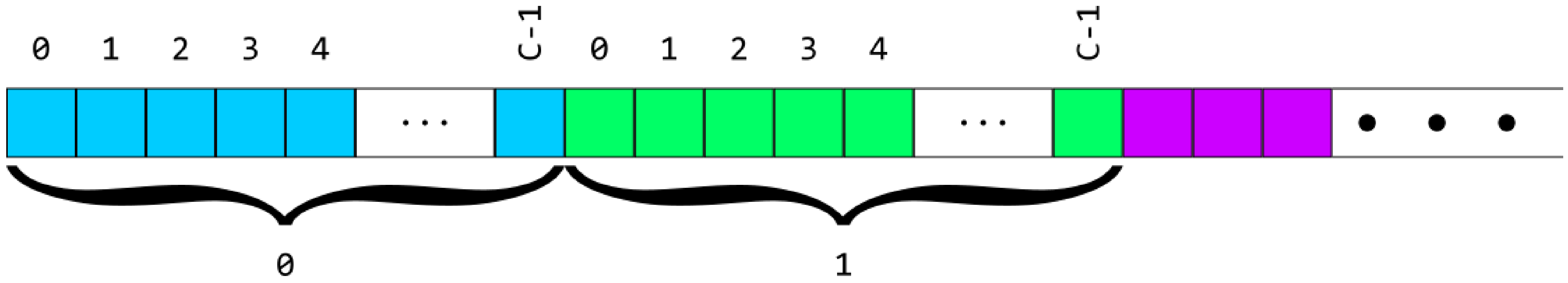
- Which one is faster?

```
int matrix[R][C] = {0};
int sum = 0;
```

```
// row-major                            // column-major
for (int r = 0; r < R; ++r)             for (int c = 0; c < C; ++c)
  for (int c = 0; c < C; ++c)             for (int r = 0; r < R; ++r)
    sum += matrix[r][c];                    sum += matrix[r][c];
```

Answer questions – #1

- Memory layout of matrix



- Row-major order traversal uses cache line better.
- Column-major's creates more cache misses.

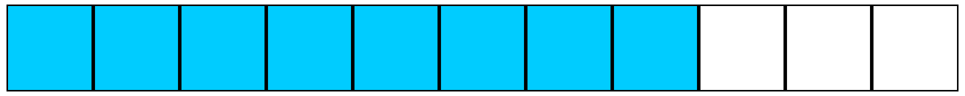
Answer questions – #2

- Which one is faster?

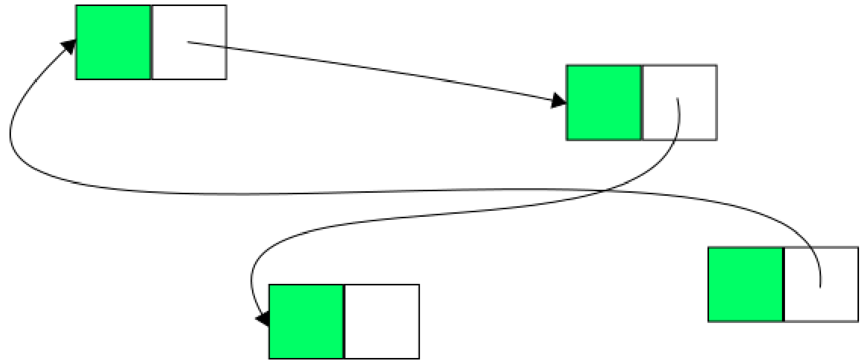
```
std::vector<int> v(BIG);
std::forward_list<int> l(BIG);

auto va = std::accumulate(begin(v), end(v), 0);
auto la = std::accumulate(begin(l), end(l), 0);
```

Answer questions – #2

- std::vector memory layout



- std::forward_list memory layout



- std::vector uses cache line better.
- std::forward_list creates more cache misses.

Answer questions – #3

- How many cycles does the CPU consume to execute this code?

```
mov eax, dword ptr [0x12345678]
```

- It depends on whether the data @ 0x12345678 is currently in cache.

Recommendations

- Use array-based data structure (std::vector, std::array, plain array).
- Arrange frequently used data close to each other.
- Avoid false sharing.
- Smaller is faster.

Demo

Thanks for listening!

Questions?