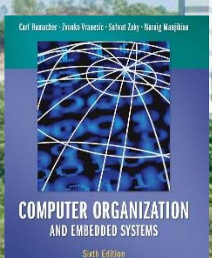
香港中文大學
The Chinese University of Hong Kong

CSCI2510 Computer Organization

Lecture 08: Memory Performance

Ming-Chang YANG

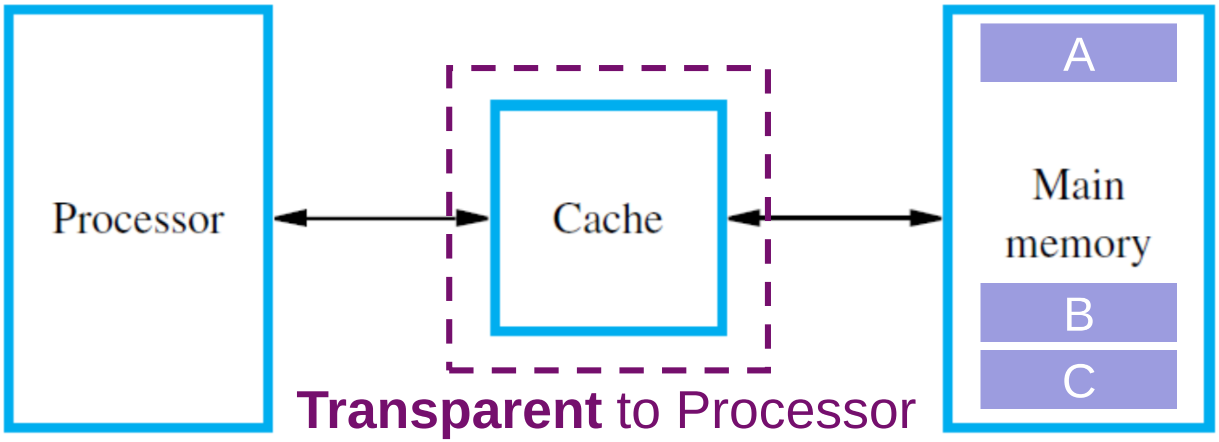
mcyang@cse.cuhk.edu.hk



Reading: Chap. 8.7

Recall: Cache is Fast but Small

- The cache is a small but very fast memory.
  - Interposed between the processor and main memory.



- Its purpose is to make the main memory appear to the processor to be **much faster than it actually is**.
  - The processor does not need to know explicitly about the existence of the cache, but just feels faster!
- How to? Exploit the locality of reference to "properly" load some data from the main memory into the cache.

Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

- Performance Enhancements
  - Prefetch
  - Load-Through
  - Memory Module Interleaving

Cache Hit/Miss Rate and Miss Penalty

- **Cache Hit**:
  - The access can be done in the cache.
  - **Hit Rate**: The ratio of number of hits to all accesses.
    - Hit rates over 0.9 are essential for high-performance PCs.

- **Cache Miss**:
  - The access can not be done in the cache.
  - **Miss Rate**: The ratio of number of misses to all accesses.
  - **Miss Penalty**: the total access time passed through (seen by the processor) when a cache miss occurs.
    - When cache miss occur, extra time is needed to bring blocks from the slower main memory to the faster cache.
    - During that time, the processor is stalled.

Class Exercise 8.1

Student ID: ____________ Date:
Name: ________________ ________

- In the working examples discussed in Class Exercise 7.6~7.8, the program will lead to
  - <u>**2**</u> cache hits (when i = 9 and 8) for direct mapping.
  - <u>**8**</u> cache hits (when i = 9, 8, …, 2) for associative mapping.
  - <u>**4**</u> cache hits (when i = 9, 8, …, 6) for 4-way set associative.
- What are the cache hit rates for each case?

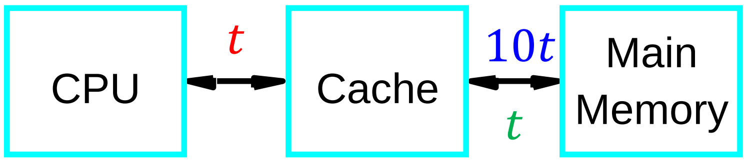
Simplified Program

`j` `i`

A[0][0]: (7A00)
A[1][0]: (7A04)
A[2][0]: (7A08)
A[3][0]: (7A0C)
A[4][0]: (7A10)
A[5][0]: (7A14)
A[6][0]: (7A18)
A[7][0]: (7A1C)
A[8][0]: (7A20)
A[9][0]: (7A24)

An Example of Miss Penalty

- **Miss Penalty**: the total access time passed through (seen by the processor) when a cache miss occurs.
- Consider a system with only one level of cache with following parameters:



  - Word access time to the cache: t
  - Word access time to the main memory: $10t$
  - When a cache miss occurs, a cache block of 8 words will be transferred from the main memory to the cache.
    - Time to transfer the first word: $10t$
    - Time to transfer each of the rest words: t (a.k.a. *fast page mode*)
- The miss penalty can be derived as:

$$t + 10t + 7 \times t + t = 19t$$

The initial cache access that results in a miss.

CPU access the requested data in the cache.

Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time
- Performance Enhancements
  - Prefetch
  - Load-Through
  - Memory Module Interleaving

Average Memory Access Time

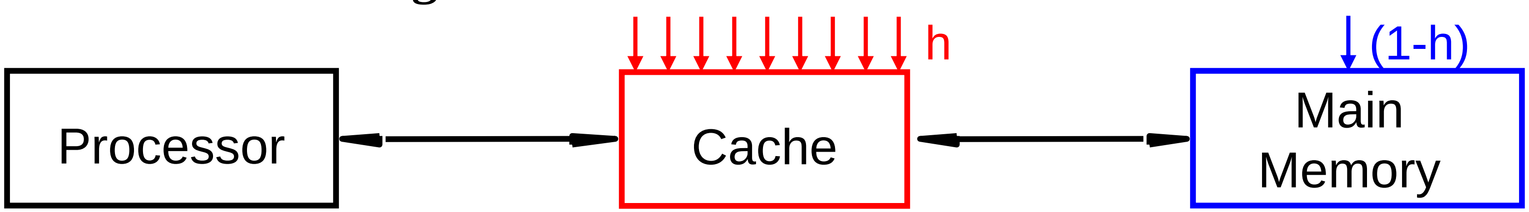
- Consider a system with only one level of cache:
  - h: Cache Hit Rate
  - $1 - h$: Cache Miss Rate
  - C: Cache Access Time
  - M: Miss Penalty
    - It mainly consists of the time to access a block in the main memory.

Expected Value in Probability

$$E[X] = \sum_i x_i \times f(x_i)$$

- The average memory access time can be defined as:

$$t_{avg} = h \times C + (1 - h) \times M$$



– For example, given $h = 0.9,\ C = 1\ cycle,\ M = 19\ cycles$:

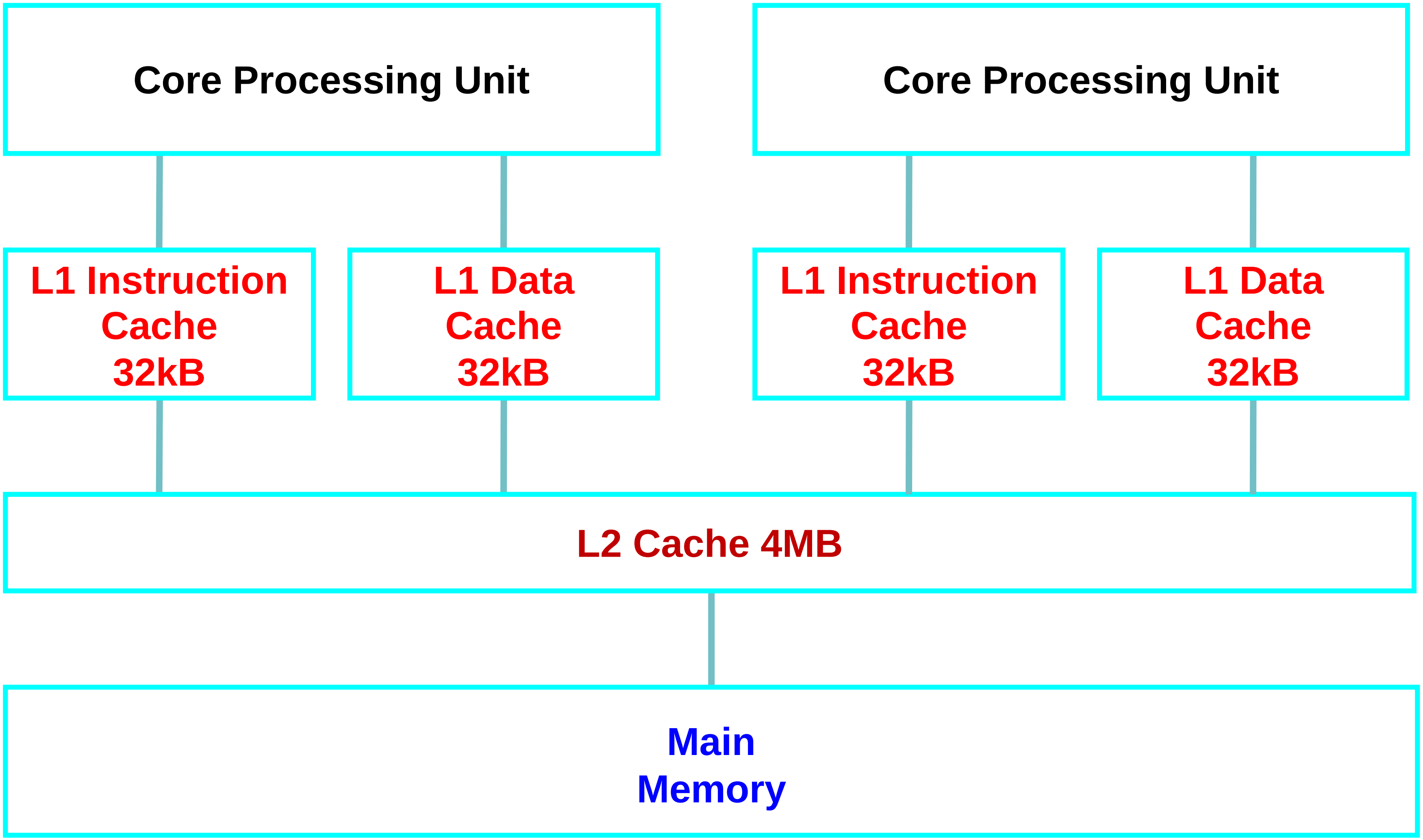
→ Avg. memory access time: $0.9 \times 1 + 0.1 \times 19 = 2.8\ (cycles)$

Real-life Example: Intel Core 2 Duo

Number of Processors	: 1
Number of Cores	: **2** per processor
Number of Threads	: 2 per processor
Name	: **Intel Core 2 Duo E6600**
Code Name	: Conroe
Specification	: Intel(R) Core(TM)2 CPU 6600@2.40GHz
Technology	: 65 nm
Core Speed	: 2400 MHz
Multiplier x Bus speed	: 9.0 x 266.0 MHz = 2400 MHz
Front-Side-Bus speed	: 4 x 266.0MHz = 1066 MHz
Instruction Sets	: MMX, SSE, SSE2, SSE3, SSSE3, EM64T

- **L1 Cache**
  - **Data Cache** : **2 x** 32 KBytes, 8-way set associative, 64-byte line size
  - **Instruction Cache** : **2 x** 32 KBytes, 8-way set associative, 64-byte line size
- **L2 Cache** : 4096 KBytes, 16-way set associative, 64-byte line size

Real-life Example: Intel Core 2 Duo



Multi-Level Caches

- In high-performance processors, two levels of caches are normally used, L1 and L2.
  - **L1 Cache**: Must be very fast as they determine the memory access time seen by the processor.
  - **L2 Cache**: Can be slower, but it should be much larger than the L1 cache to ensure a high hit rate.
- The avg. memory access time of two levels of caches:

$$t_{avg} = h_1 \times C_1 + (1 - h_1) \times [h_2 \times M_{L1} + (1 - h_2) \times M_{L2}],$$

  - h_1 / h_2: hit rate of L1 cache / L2 cache
  - $C_1 / C_2 / Mem$: access time to L1 cache / L2 cache / memory
  - M_{L1}: miss penalty of L1 miss & L2 hit
    - E.g., $M_{L1} = C1 + C2 + C1$
  - M_{L2}: miss penalty of L1 miss & L2 miss
    - E.g., $M_{L2} = C1 + C2 + Mem + C2 + C1$

Avg. memory access time of one level of cache:

$$t_{avg} = h \times C + (1 - h) \times M$$

Class Exercise 8.2

- Given a system with one level of cache, and a system with two level of caches.
- Assume the hit rates of L1 cache and L2 cache (if any) are both 0.9.
- What are the probabilities that miss penalty must be paid to read a block from memory in both systems?

Separate Instruction/Data Caches (1/2)

- Consider the system with only one level of cache:
  - Word access time to the cache: 1 cycle
  - Word access time to the main memory: $10\ cycles$
  - When a cache miss occurs, a cache block of 8 words will be transferred from the main memory to the cache.
    - Time to transfer the first word: $10\ cycles$
    - Time to transfer one word of the rest 7 words: 1 cycle
  - Miss Penalty: $1 + 10 + 7 \times 1 + 1 = 19\ (cycles)$

- Assume there are total 130 memory accesses:
  - 100 memory accesses for instructions with hit rate 0.95
  - 30 memory access for data (operands) with hit rate = 0.90

Separate Instruction/Data Caches (2/2)

- Total execution cycles **without** cache:

$$t_{without} = 100 \times 10 + 30 \times 10 = 1300\ cycles$$

  - All of the memory accesses will result in a reading of a memory word (of latency 10 cycles).

- Total execution cycles **with** cache:

Avg. memory access time for instructions: $h \times C + (1-h) \times M$

$$t_{with} = 100 \times (0.95 \times 1 + 0.05 \times 19) + 30 \times (0.9 \times 1 + 0.1 \times 19) = 274\ cycles$$

Avg. memory access time for data: $h \times C + (1-h) \times M$

- The performance improvement:

$$\frac{t_{without}}{t_{with}} = \frac{1300}{274} = 4.74\ (speed\ up!)$$

Class Exercise 8.3

- Consider the same system with one level of cache.
  - Word access time to the cache: $1\ cycle$
  - Word access time to the main memory: $10\ cycles$
  - Miss Penalty: $1 + 10 + 7 \times 1 + 1 = 19\ (cycles)$
  - 100 memory accesses for instructions with hit rate 0.95
  - 30 memory access for data (operands) with hit rate = 0.90
- What is the performance difference between this cache and an ideal cache?
  - **Ideal Cache**: All the accesses can be done in cache.

Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

- Performance Enhancements
  - Prefetch
  - Load-Through
  - Memory Module Interleaving

How to Improve the Performance?

- Recall the system with only one level of cache:
  - h: Cache Hit Rate
  - $1 - h$: Cache Miss Rate
  - C: Cache Access Time
  - M: Miss Penalty
    - It mainly consists of the time to access a block in the main memory.
- The average memory access time can be defined as:

$$t_{avg} = h \times C + (1 - h) \times M$$

- Possible ways to further reduce t_{avg}?
  ① Use faster cache (i.e., $C \downarrow$)? $$$...
  ② Improve the hit rate (i.e., $h \uparrow$)?
  ③ Reduce the miss penalty (i.e., $M \downarrow$)?

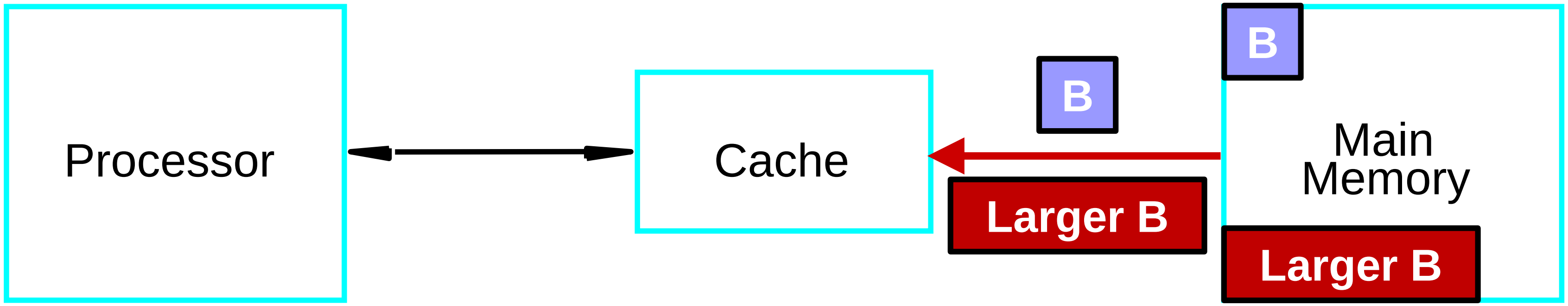
Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

- Performance Enhancements
  - Prefetch (i.e., $h \uparrow$)
  - Load-Through
  - Memory Module Interleaving

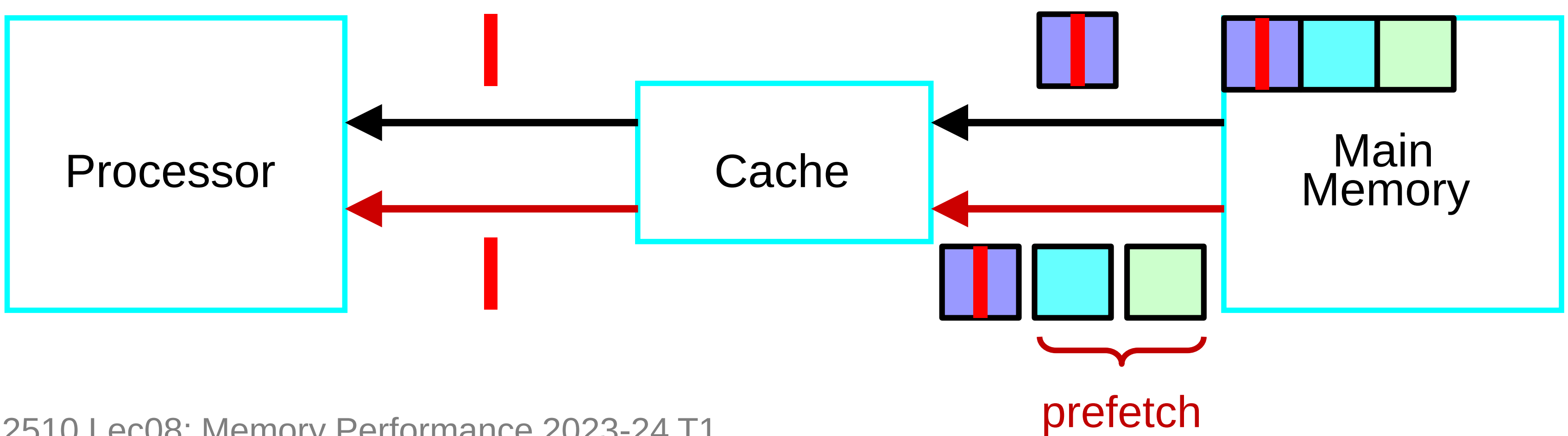
How to Improve Hit Rate?

- How about larger block size?
  - Larger blocks take more advantage of the spatial locality.
    - **Spatial Locality**: If all items in a larger block are needed in a computation, it is better to load them into cache in a single miss.
  - Larger blocks are effective only up to a certain size:
    - Too many items will remain unused before the block is replaced.
    - It takes longer time to transfer larger blocks and may also increase the miss penalty.
  - Block sizes of 16 to 128 bytes are most popular.



Prefetch: More rather than Larger

- **Prefetch**: Load **more** (rather than ~~larger~~) blocks with consecutive memory addresses into the cache before they are needed, while CPU is busy.
  - Prefetch instruction can be put by programmer or compiler.
- Some data may be loaded into the cache without being used, before the prefetched data are replaced.
  - The overall effect on performance is still positive.
  - Most processors support the prefetch instruction.



Revisit: Will Prefetch Help with This?

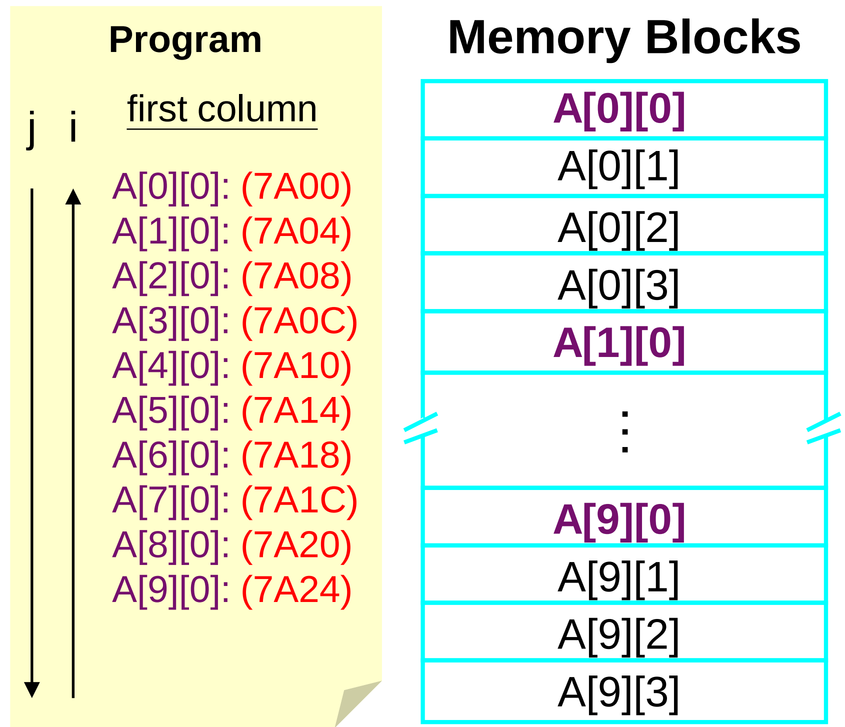
- Cache Configuration:
  - Cache has 8 blocks.
  - A block is of 1 (= 2^0) word.
  - A word is of 16 bits.

```
short  A[10][4];
int    sum = 0;
int    j, i;
double mean;

// 1) forward loop
for (j = 0; j <= 9; j++)
  sum += A[j][0];
mean = sum / 10.0;

// 2) backward loop
for (i = 9; i >= 0; i--)
  A[i][0] = A[i][0] / mean;
```

- **A[10][4]** is an array of words located at the memory word addresses $(7A00)_{16}$~$(7A27)_{16}$ in **row-major** order.



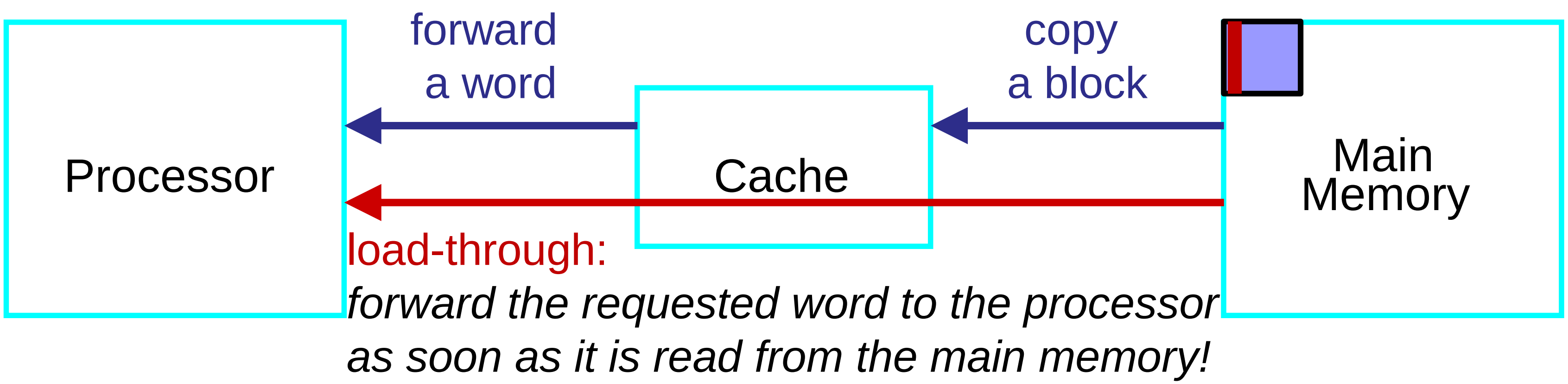
Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

- Performance Enhancements
  - Prefetch
  - Load-Through (i.e., $M \downarrow$)
  - Memory Module Interleaving

Load-through

- Consider a read cache miss:
  - Copy the block containing the requested word to the cache.
  - Then forward to CPU after the entire block is loaded.
- **Load-through**: Instead of waiting the whole block to be transferred, send the requested word to the processor as soon as it is ready.
  - **Pros**: Opportunistically reduce the miss paneity
  - **Cons**: At the expense of more complex circuitry ($)



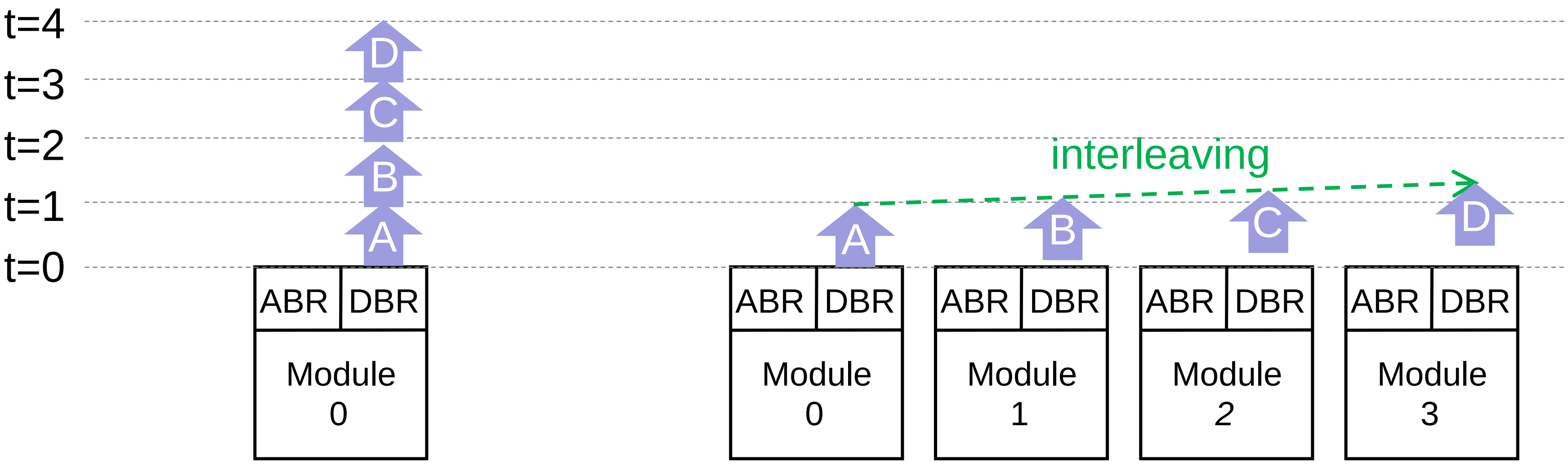
Outline

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

- Performance Enhancements
  - Prefetch
  - Load-Through
  - Memory Module Interleaving (i.e., $M \downarrow$)

Memory Module Interleaving (1/3)

- How to substantially reduce the miss penalty?
  - The main memory is slow in essence ...
- **Idea**: Hide the memory access latency by interleaving memory accesses across several memory modules.
  - Each module has own **Address Buffer Register (ABR)** and **Data Buffer Register (DBR)** to access memory contents.

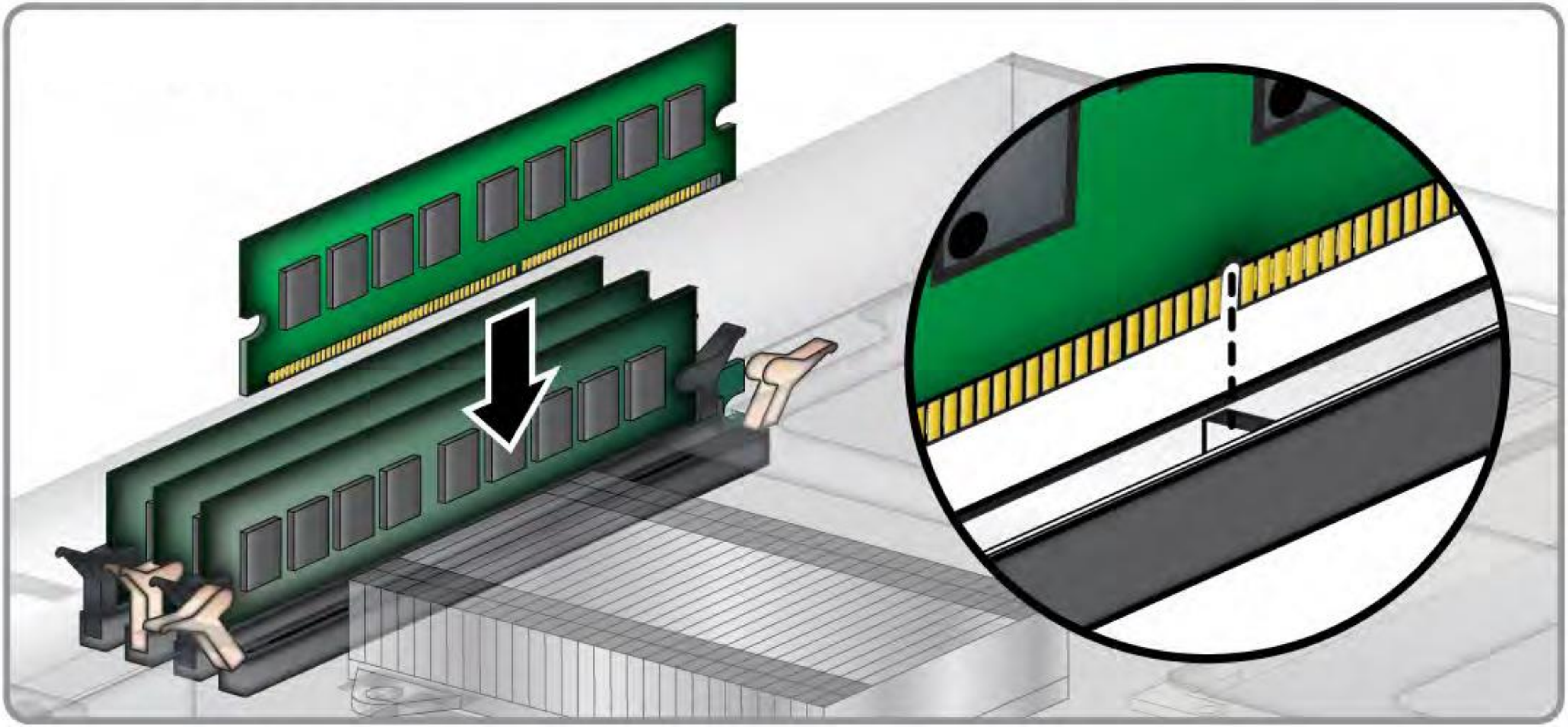


Without Memory Module Interleaving

With Memory Module Interleaving

Memory Module Interleaving (2/3)

- Multiple memory modules (usually a multiple of 2) can be installed in modern computers.



https://origin-symwisedownload.symantec.com/resources/webguides/cacheflow/3x/3_4/webguide/Content/Hardware/CF500_Memory.htm

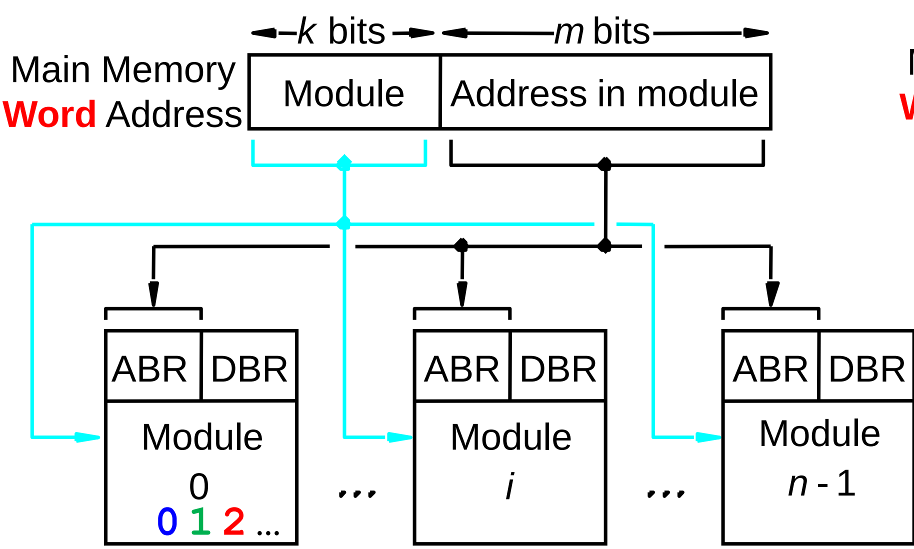
Memory Module Interleaving (3/3)

- Which scheme below can be better interleaved?
  - **Scheme (a)**: Consecutive words in the same module.
  - **Scheme (b)**: Consecutive words in successive module.

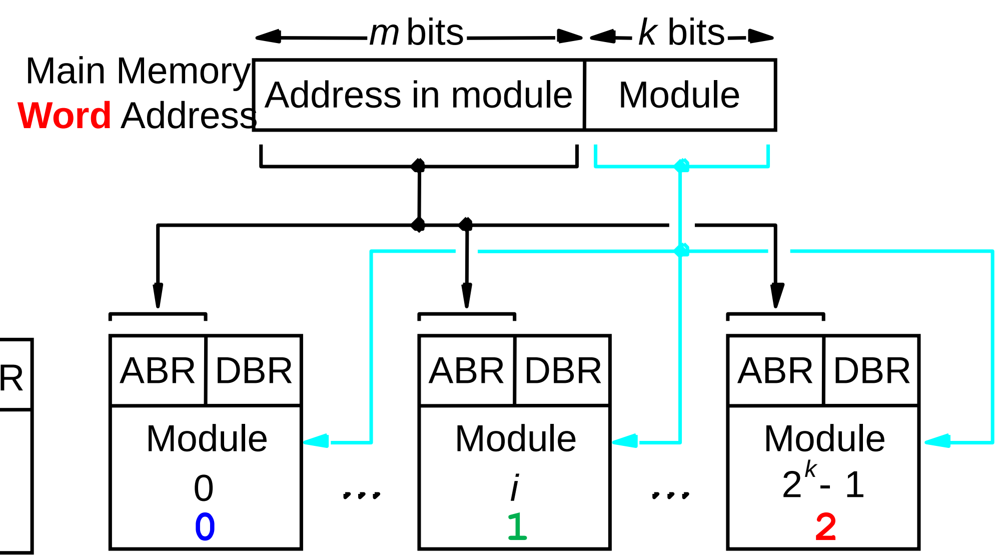
$(0 \ldots 000 \quad 0000 \ldots 0010)_2 = (2)_{10}$ $(0 \ldots 000 \quad 0000 \ldots 0010)_2 = (2)_{10}$

$(0 \ldots 000 \quad 0000 \ldots 0001)_2 = (1)_{10}$ $(0 \ldots 000 \quad 0000 \ldots 0001)_2 = (1)_{10}$

$(0 \ldots 000 \quad 0000 \ldots 0000)_2 = (0)_{10}$ $(0 \ldots 000 \quad 0000 \ldots 0000)_2 = (0)_{10}$



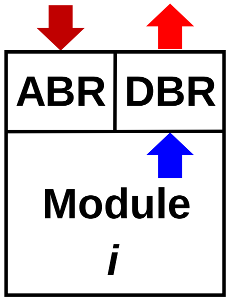
(a) Consecutive words in the same module



(b) Consecutive words in successive modules

Example of Memory Module Interleaving

- Consider a cache read miss, and we need to load a block of 8 words from main memory to the cache.
- Assume consecutive words are in successive modules for the better interleaving (i.e., Scheme (b)).
- For every memory module:
  - **Address Buffer Register** & **Data Buffer Register**
  - Module Operations:
    - Send an address to ABR: **1** cycle
    - Read the first word from module into DBR: **6** cycles
    - Read a subsequent word from module into DBR: **4** cycles
    - Read the data from DBR: **1** cycle



The shared buses only allow accessing the **ABR** of any module one at a time and accessing the **DBR** of any module one at a time.

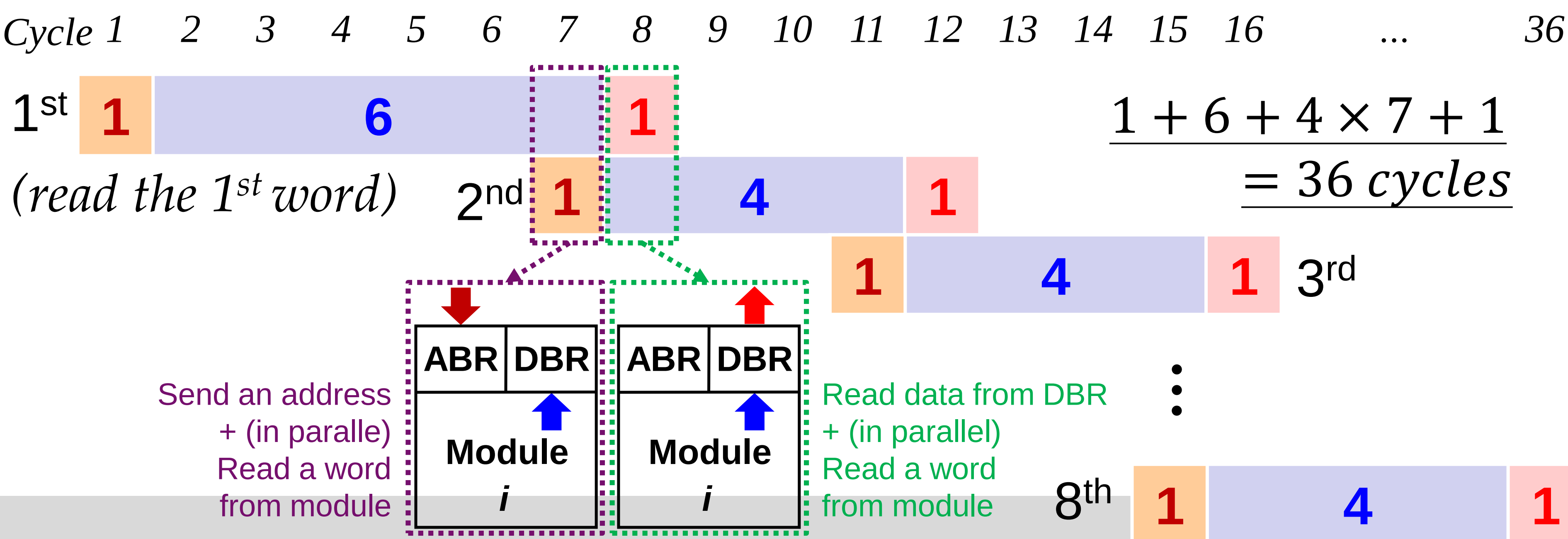
Without Interleaving (Single Module)

- Total cycles to read a single word from the module:

Send an address to ABR: **1** cycle
Read the first word: **6** cycles
Read a subsequent word: **4** cycles
Read the data from DBR: **1** cycle

  - **1** cycle to send the address
  - **6** cycles to read the first word
  - **1** cycle to read the data from DBR → $\underline{1+6+1=8\ cycles}$

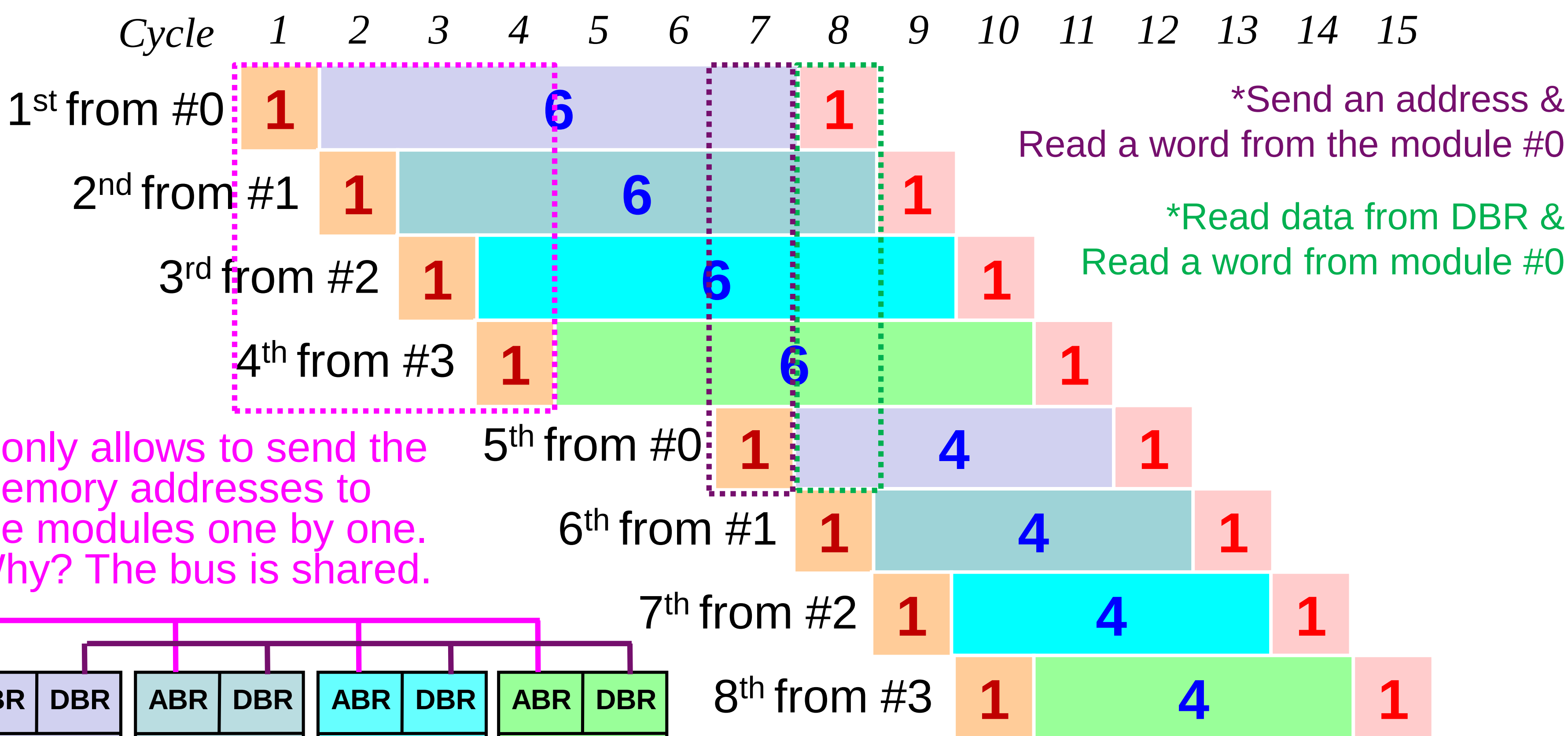
- Total cycles to read an 8-word block from the module:



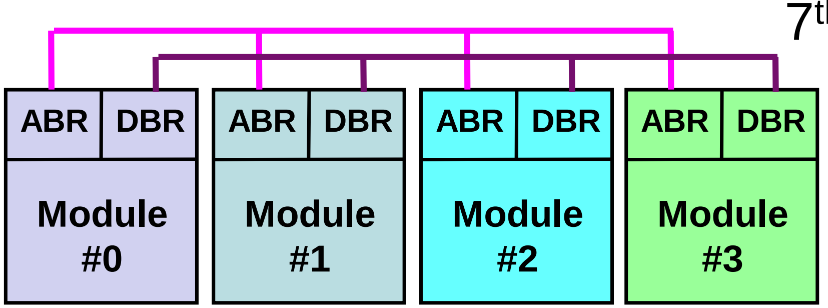
With Interleaving

- Send an address to ABR: **1** cycle
- Read the first word: **6** cycles
- Read a subsequent word: **4** cycles
- Read the data from DBR: **1** cycle

- Total cycles to read a 8-word block from **four** interleaved memory modules:



It only allows to send the memory addresses to the modules one by one. Why? The bus is shared.

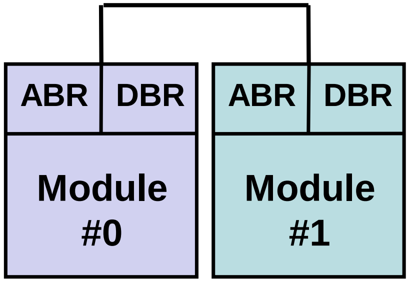


$$1 + 6 + 1 \times 8 = 15\ cycles$$

Class Exercise 8.4

- What is the number of total cycles to read an 8-word block from **two** interleaved memory modules?

- Send an address to ABR: **1** cycle
- Read the first word: **6** cycles
- Read a subsequent word: **4** cycles
- Read the data from DBR: **1** cycle



Summary

- Performance Evaluation
  - Cache Hit/Miss Rate and Miss Penalty
  - Average Memory Access Time

$$t_{avg} = h \times C + (1 - h) \times M$$

- Performance Enhancements
  - Prefetch (i.e., $h \uparrow$)
  - Load-Through (i.e., $M \downarrow$)
  - Memory Module Interleaving (i.e., $M \downarrow$)