BIG DATA MANAGEMENT

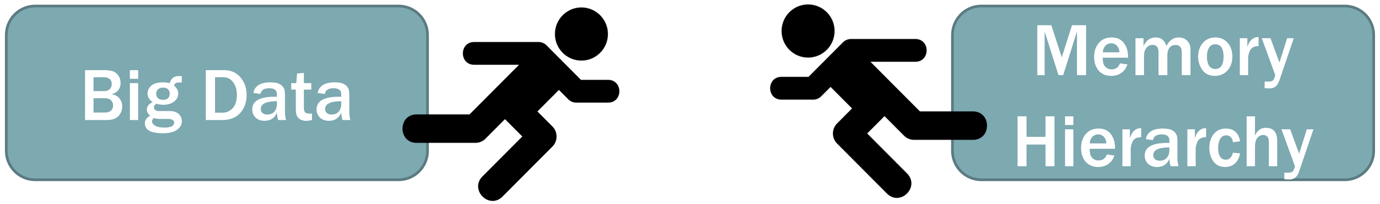
CZ4123

MEMORY HIERARCHY (PART II)

Siqiang Luo

Assistant Professor

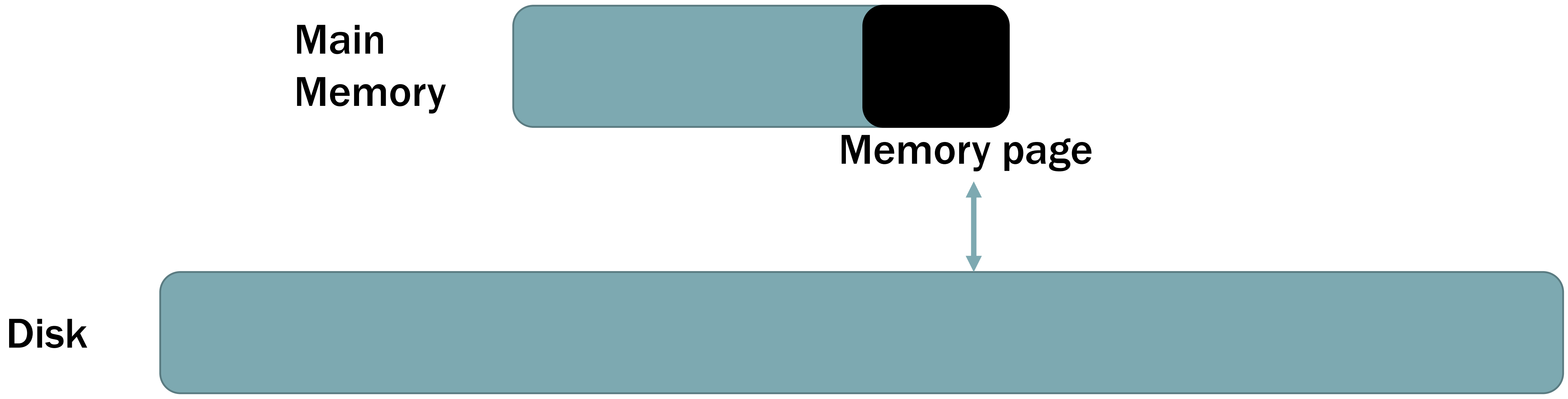
Data in Memory Hierarchy: Basic Cost Analysis



Big data cannot be fully loaded into main memory. Hence, often there are data movements between main memory and disks.

DATA ACCESS IN MEMORY HIERARCHY

- Big data cannot be fully loaded into main memory. Hence, often there are data movements between main memory and disks.



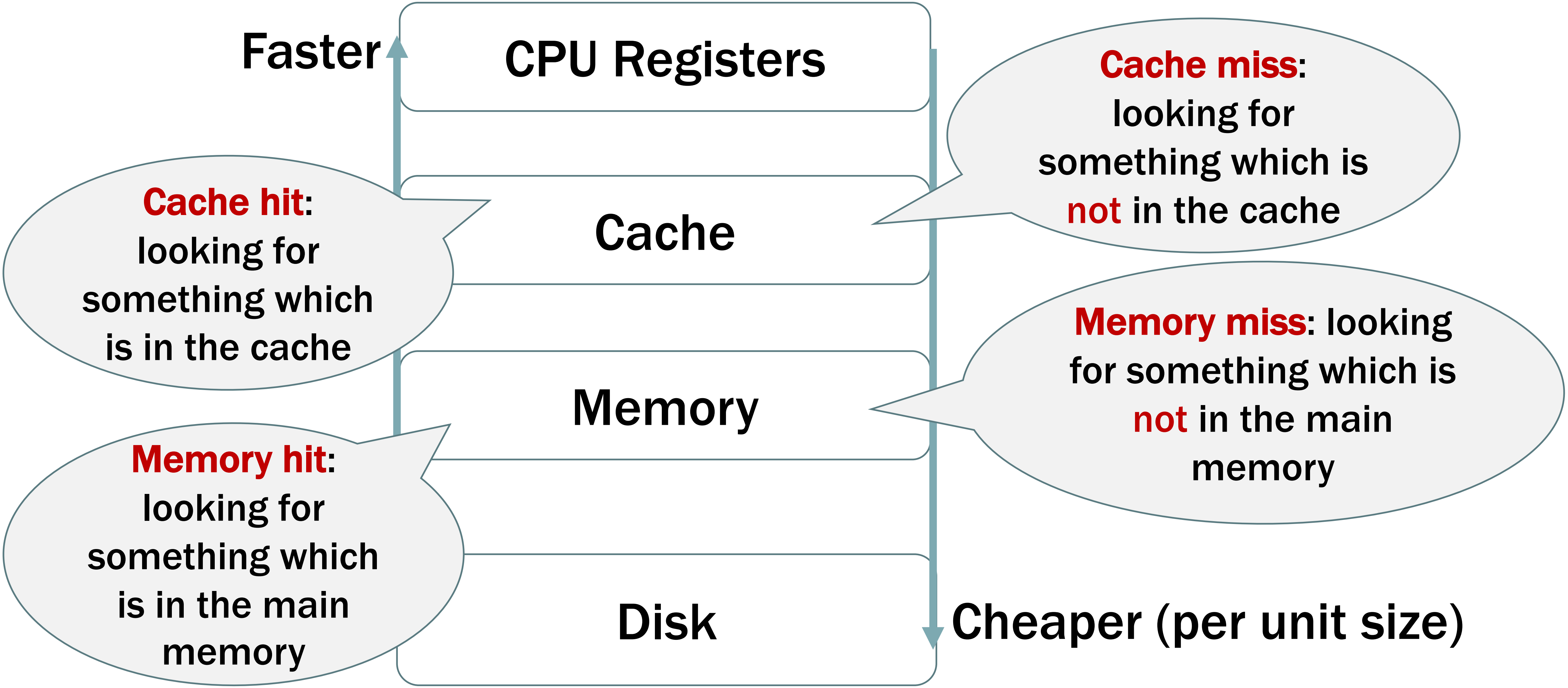
DATA ACCESS IN MEMORY HIERARCHY

- We often analyze the data access cost considering two consecutive layers: Layer i and i+1.
  - (Layer i, Layer i+1) can be (cache, memory)
  - (Layer i, Layer i+1) can also be (disk-cache, disk), where disk-cache is part of memory.

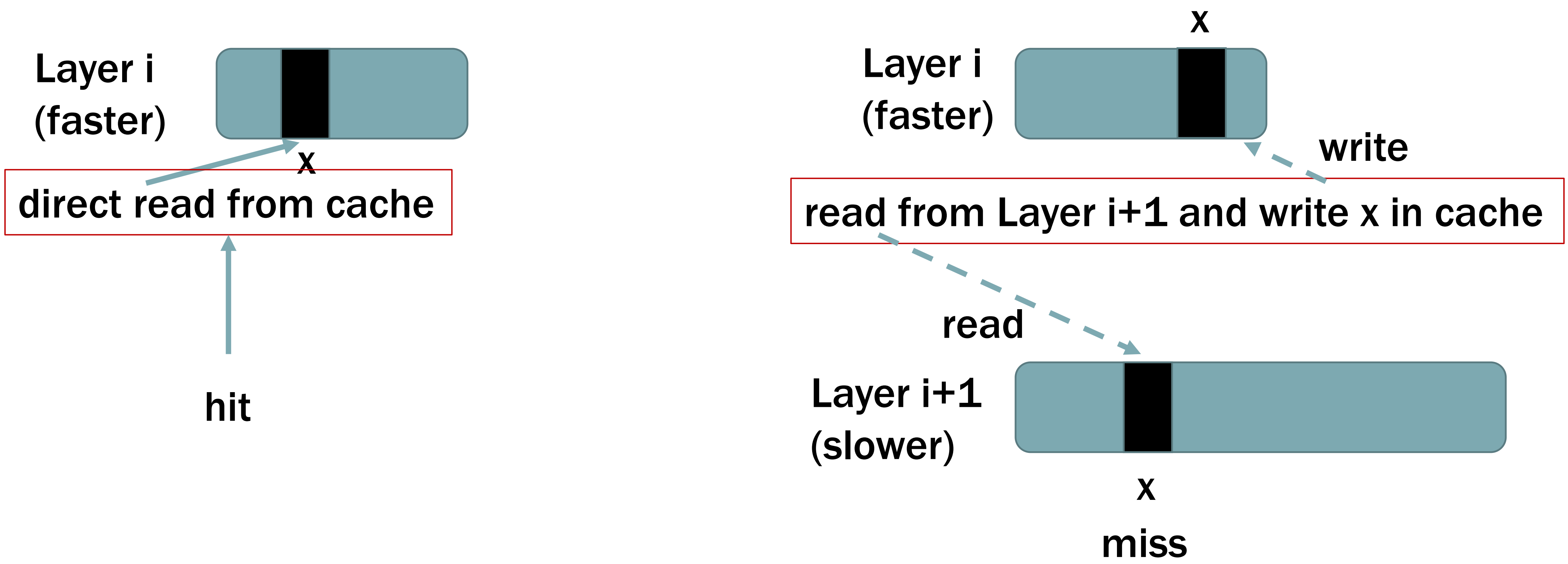
DATA ACCESS IN MEMORY HIERARCHY

- Some data item in Layer i+1 has been cached in Layer i
- When reading a data item in Layer i+1, it will first check whether the data item is cached in Layer i.
  - If yes, directly read the data from the faster Layer i
  - If no, read the data from Layer i+1 and write the data into Layer i+1's cache (usually located in Layer i).

CACHE HIT/MISS AND MEMORY HIT/MISS



DATA ACCESS IN MEMORY HIERARCHY



COST OF A CACHE MISS

Question:
Suppose a data item is located in **main memory** and can be **cached** in the **cache memory**.

1. accessing cache memory once incurs *cost_access_cache*;
2. accessing main memory once incurs *cost_access_mem*;
3. cache miss rate is h (0<=h<=1).

How to estimate the cost of reading data item?

COST OF A CACHE MISS

- If cache hit: cost_access_cache x (1-h)

- If cache miss: $(\underbrace{\text{cost\_access\_cache} + \text{cost\_access\_mem})}_{\textbf{A simplified estimation}} \text{ x } h$

- Overall: cost_access_mem_overall
  =cost_access_cache x (1-h) + (cost_access_cache + cost_access_mem) x h

COST OF A CACHE MISS

- Overall: cost_access_mem_overall
  =cost_access_cache x (1-h) + (cost_access_cache + cost_access_mem) x h

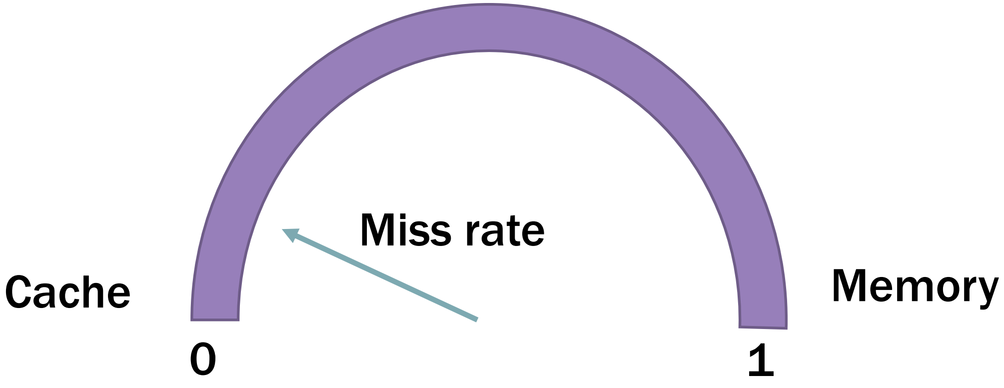
- If h = 1 (cache miss rate is high), then the cost is
  cost_access_cache + cost_access_mem $\approx$ cost_access_mem
  - The above estimation is due to the fact that *cost_access_mem* is significantly higher (say, 1000x) than *cost_access_cache*

- If h = 0 (cache miss rate is low), then the cost is *cost_access_cache*

COST OF A CACHE MISS

- Overall: cost_access_mem_overall
  =cost_access_cache x (1-h) + (cost_access_cache + cost_access_mem) x h

- Summary
  - If cache miss rate is low, then data read performance is close to cache read performance.
  - If cache miss rate is high, then data read performance is close to memory read performance.



COST OF A CACHE MISS

It is important to minimize the cache miss rate!

COST OF A MEMORY MISS

Question:
Suppose data items are located in the **disk** and can be cached in **main memory** as well.

(We only consider two layers)
1. Accessing disk once incurs a cost of *cost_access_disk*;
2. Accessing main memory once incurs a cost of *cost_access_mem*;
3. Memory miss rate is h* (0<=h*<=1).

How to estimate the cost of reading the data Item?

COST OF A MEMORY MISS

- If memory hit: cost_access_mem x (1-h*)

- If memory miss: (cost_access_disk + cost_access_mem) x h*

- Overall: cost_access_disk_overall

=cost_access_mem x (1-h*) + (cost_access_disk + cost_access_mem) x h*

COST OF A MEMORY MISS

- cost_access_disk_overall =
  cost_access_mem x (1-h*) + (cost_access_disk + cost_access_mem) x h*

- If h*=1(high miss rate):
  cost_access_disk_overall
  = cost_access_disk + cost_ access _mem $\approx$ cost_ access _disk

  The above step is due to the fact that cost_ access _disk significantly larger than (e.g., 1000x) cost_ access _mem

- If h*=0: cost_ access _disk_overall = cost_ access _mem

COST OF A MEMORY MISS

- cost_ access _disk_overall =
  cost_ access _mem x (1-h*) + (cost_ access _disk + cost_ access _mem) x h*

- Summary
  - If h*=1(high miss rate), then the read performance is close to the read performance of disk
  - If h*=0(low miss rate), then the read performance is close to the read performance of memory.

What about considering more than two layers?

We will see this in tutorial questions.

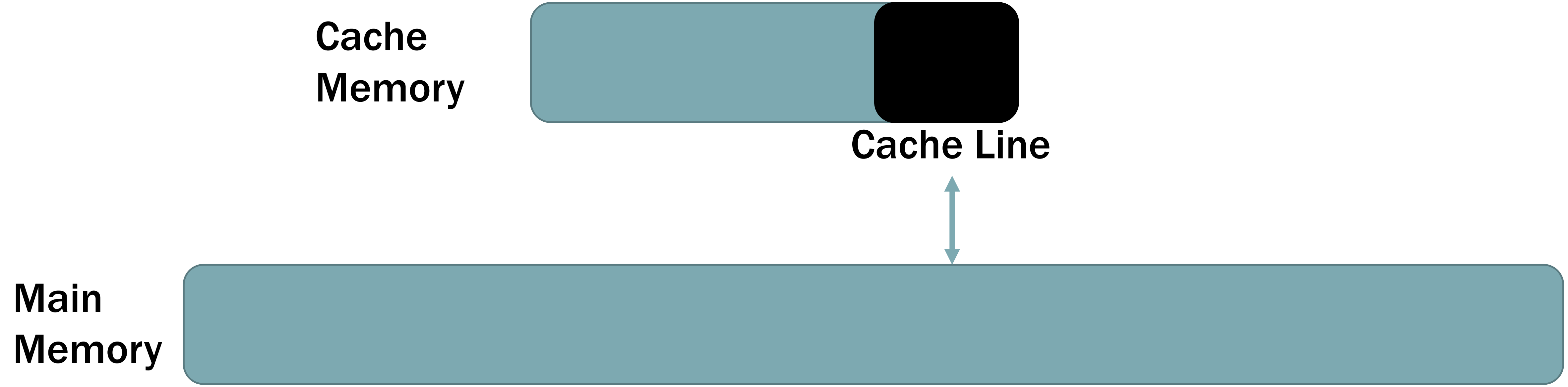
How to obtain a smaller cache miss rate?

We will see this in the coming lectures about cache conscious designs.

Data in Memory Hierarchy: Page-based Access

SMALLEST UNIT TRANSFERRED BETWEEN TWO LAYERS

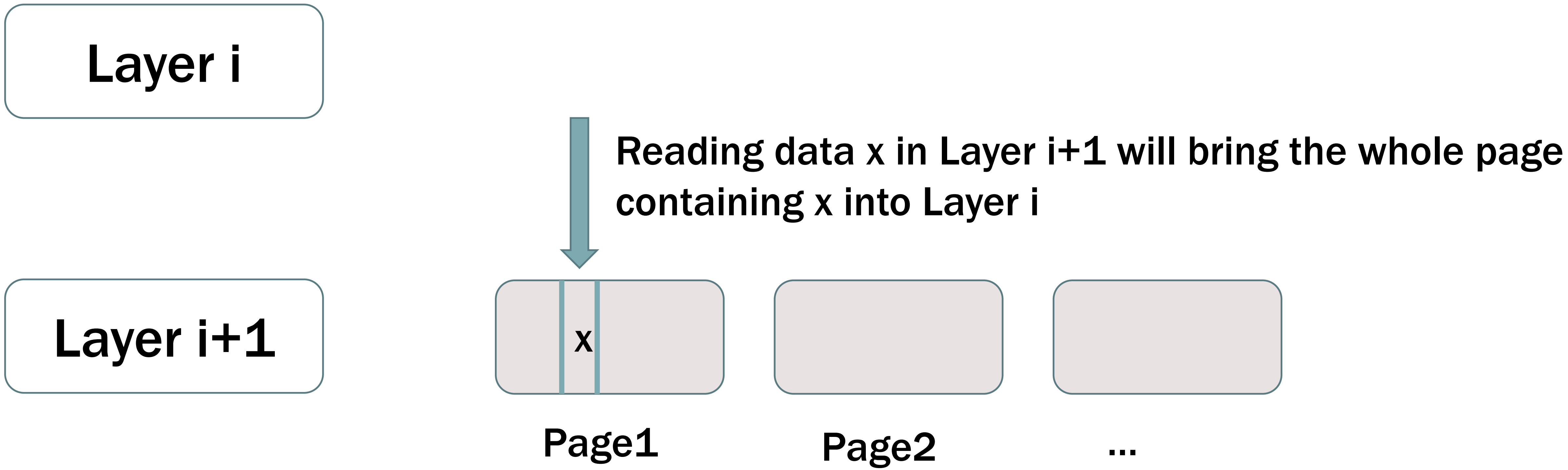
- The data unit swapped between cache memory and main memory is called cache line
  - Cache line size is usually 32, 64 and 128 bytes
  - A cache line contains continuous memory segment



SMALLEST UNIT TRANSFERRED BETWEEN TWO LAYERS

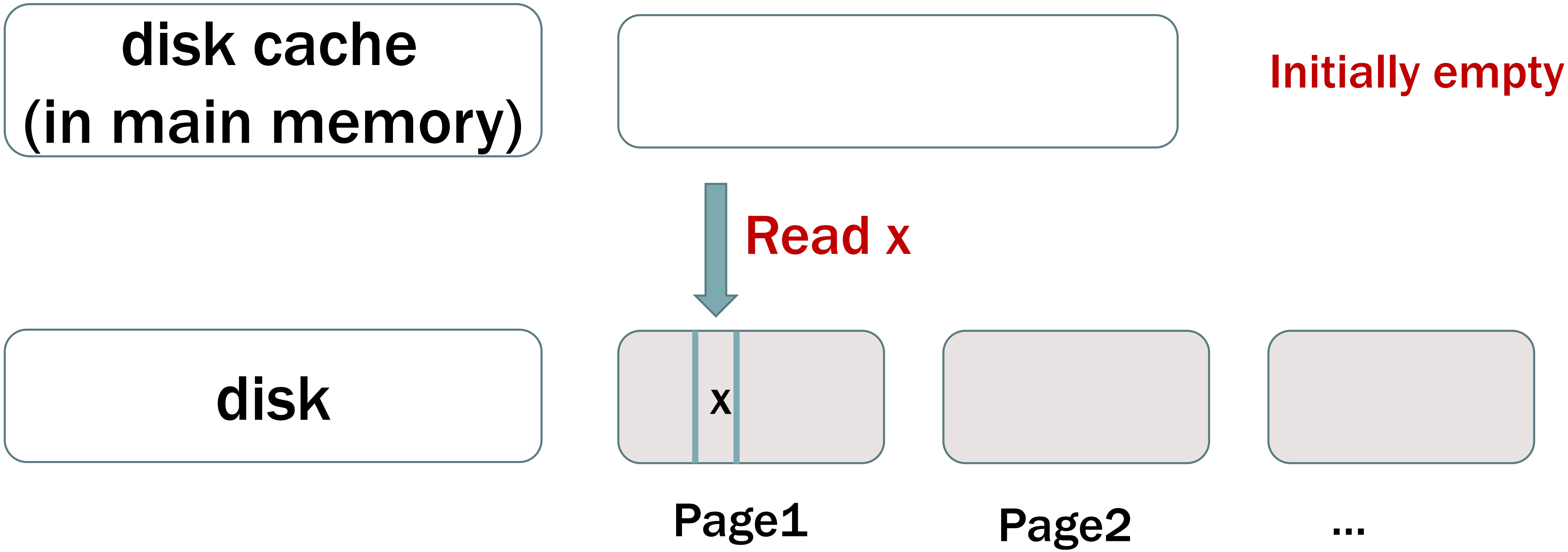
- ❑ The data unit swapped between cache memory and main memory is called cache line
  - ❑ Cache line size is usually 32, 64 and 128 bytes
  - ❑ A cache line contains continuous memory segment
- ❑ The data unit swapped between main memory and disk is called (memory) page.
  - ❑ Page size is usually about 4KB
  - ❑ A page contains continuous memory segment
- ❑ We use “page” as a common term to refer to the transfer data unit between Layer i and Layer i+1.

EXAMPLE OF PAGE-BASED ACCESS

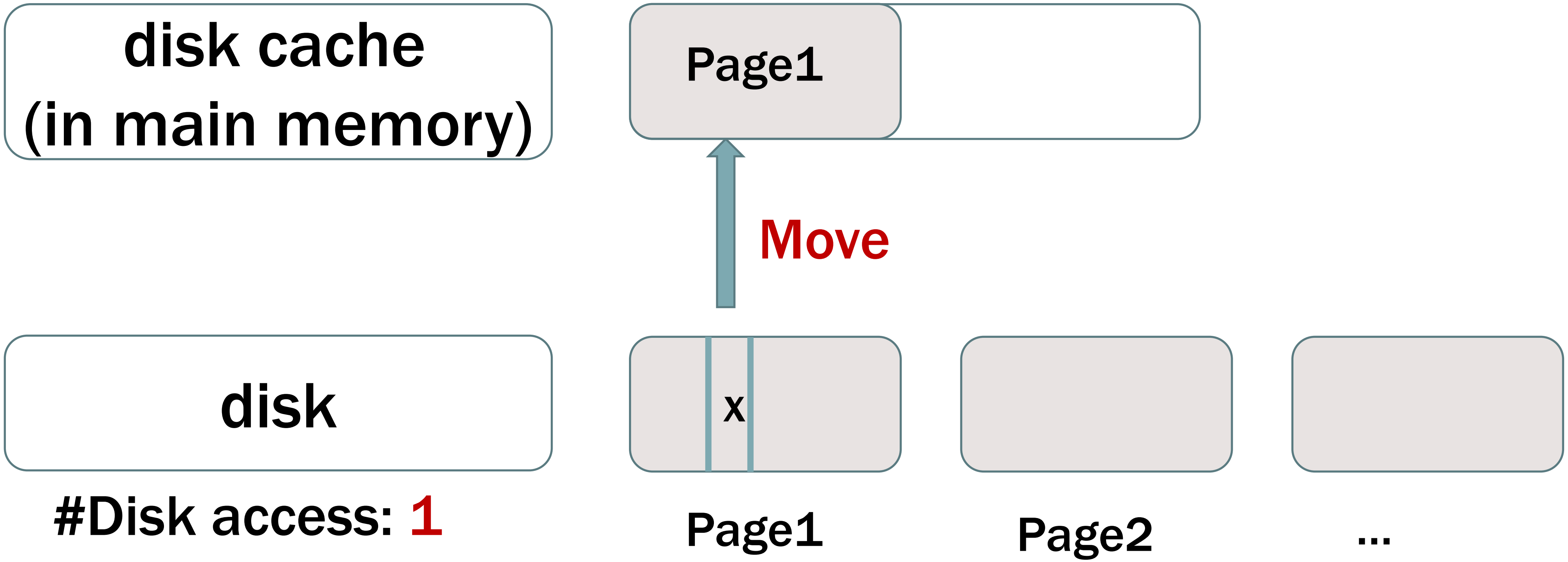


Note: A page can contain many data items.

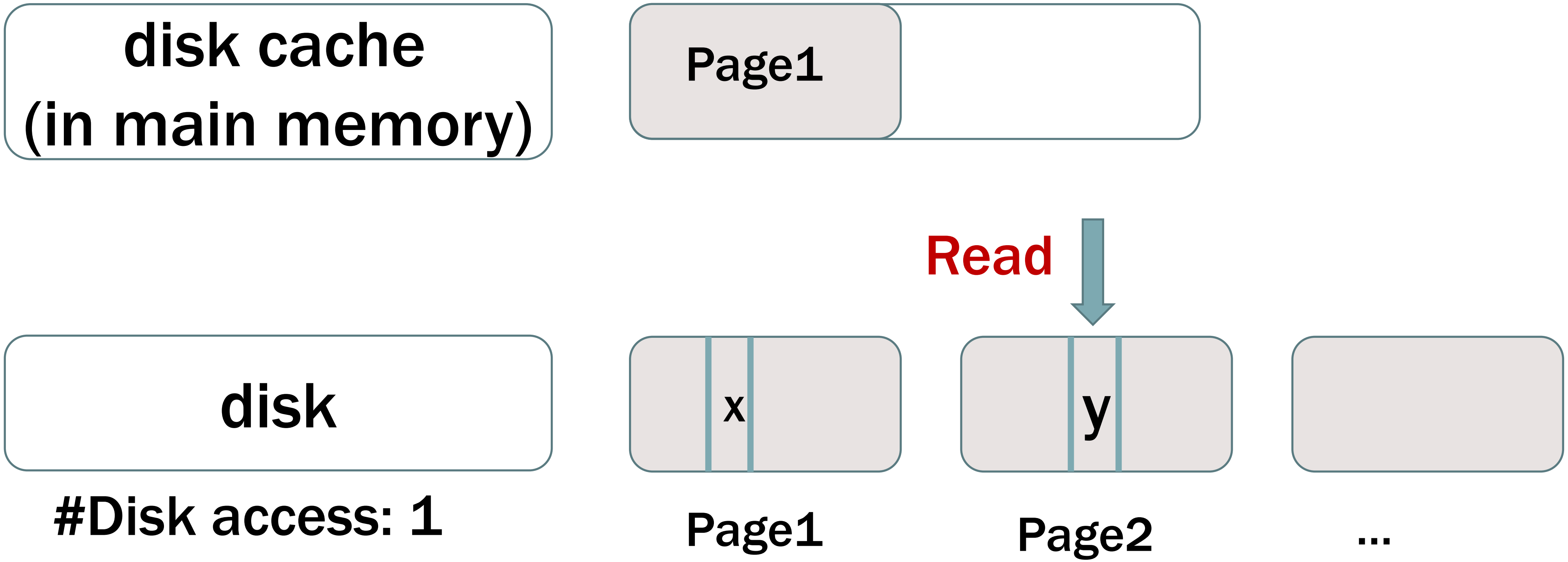
EXAMPLE OF PAGE-BASED ACCESS



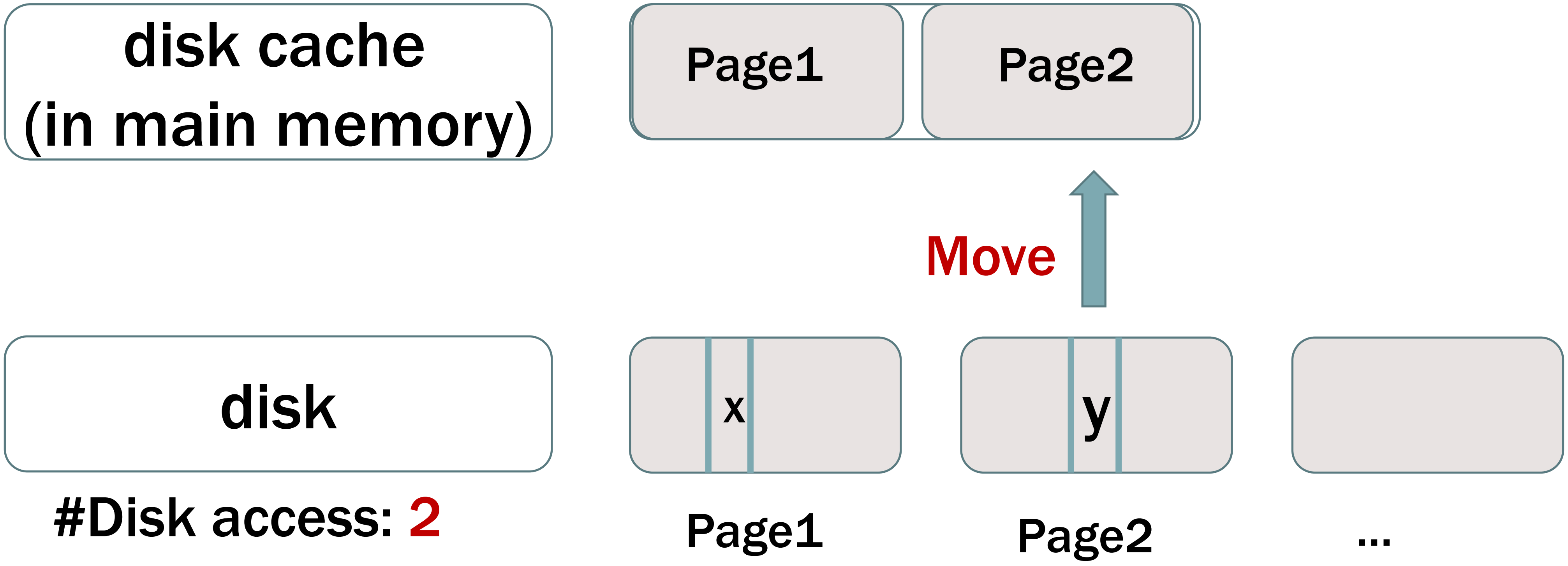
EXAMPLE OF PAGE-BASED ACCESS



EXAMPLE OF PAGE-BASED ACCESS

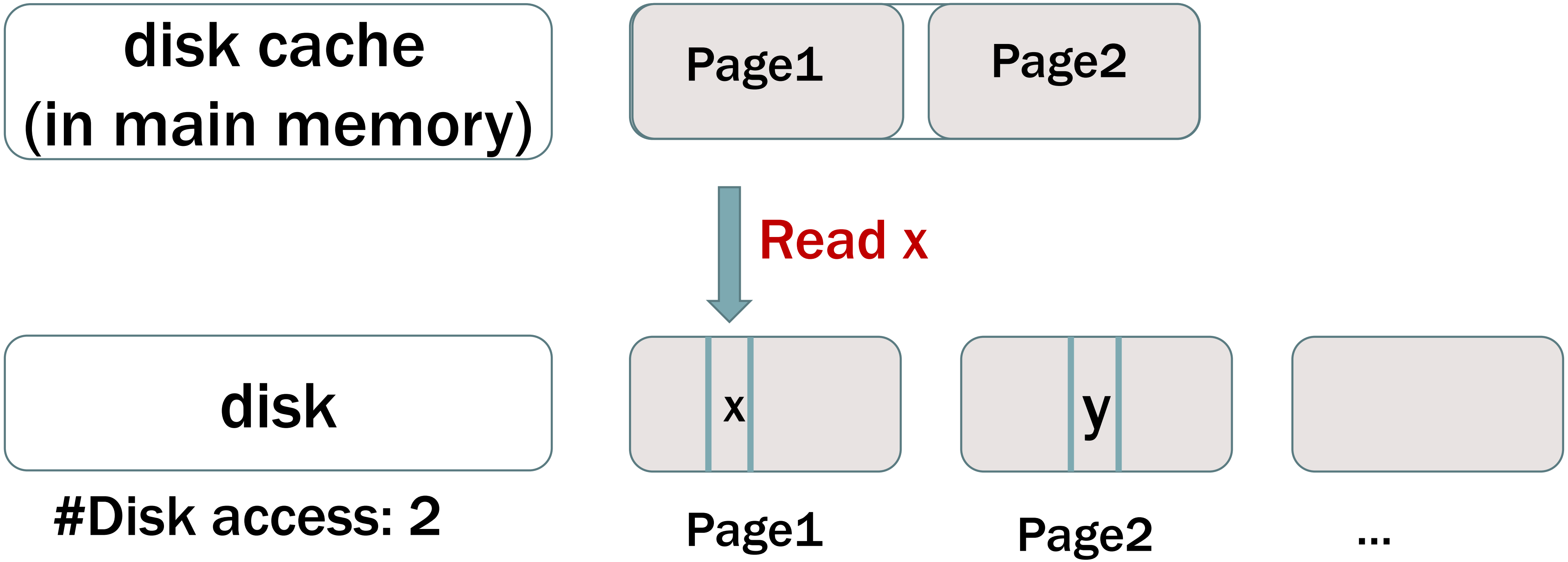


EXAMPLE OF PAGE-BASED ACCESS



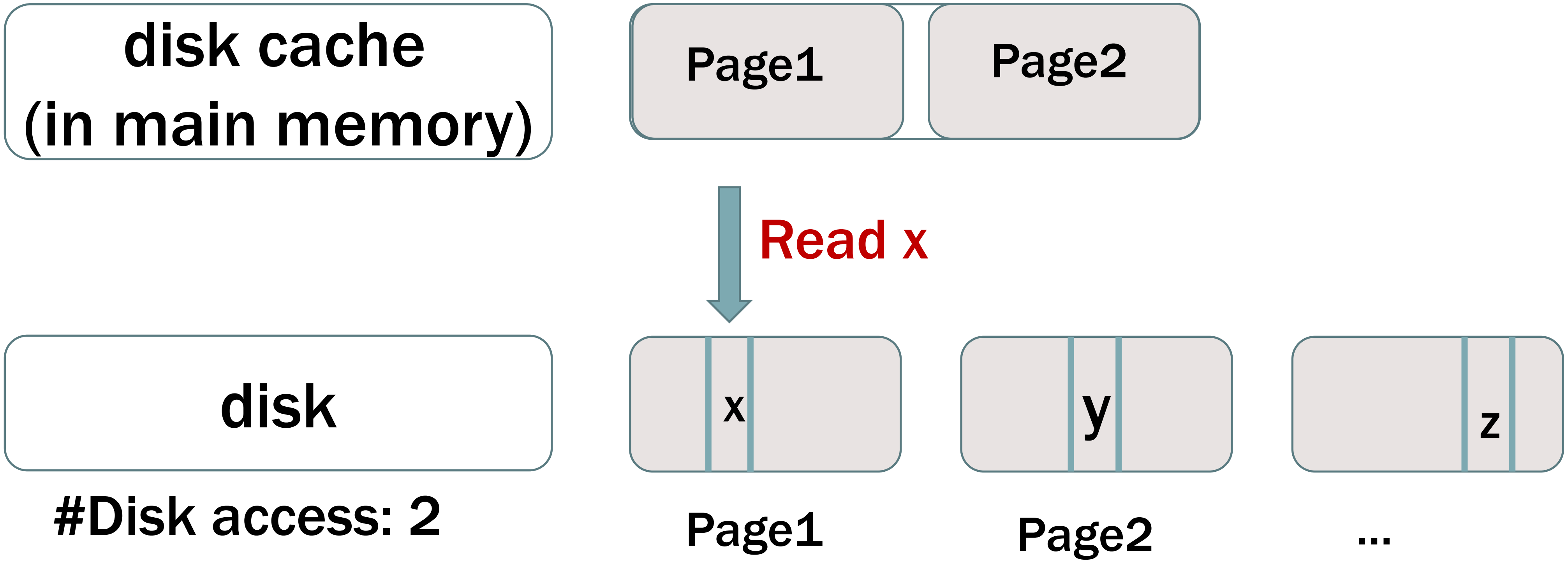
EXAMPLE OF PAGE-BASED ACCESS

Now, if we further read x, no additional disk accesses because it is in main memory already.



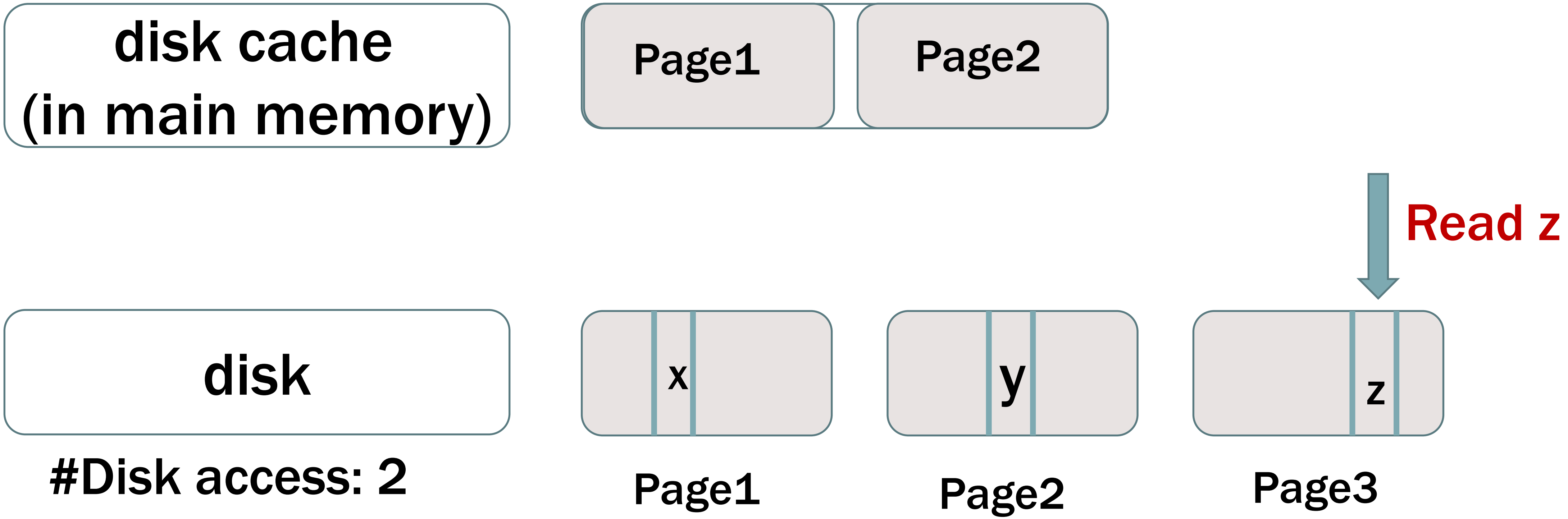
EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page?



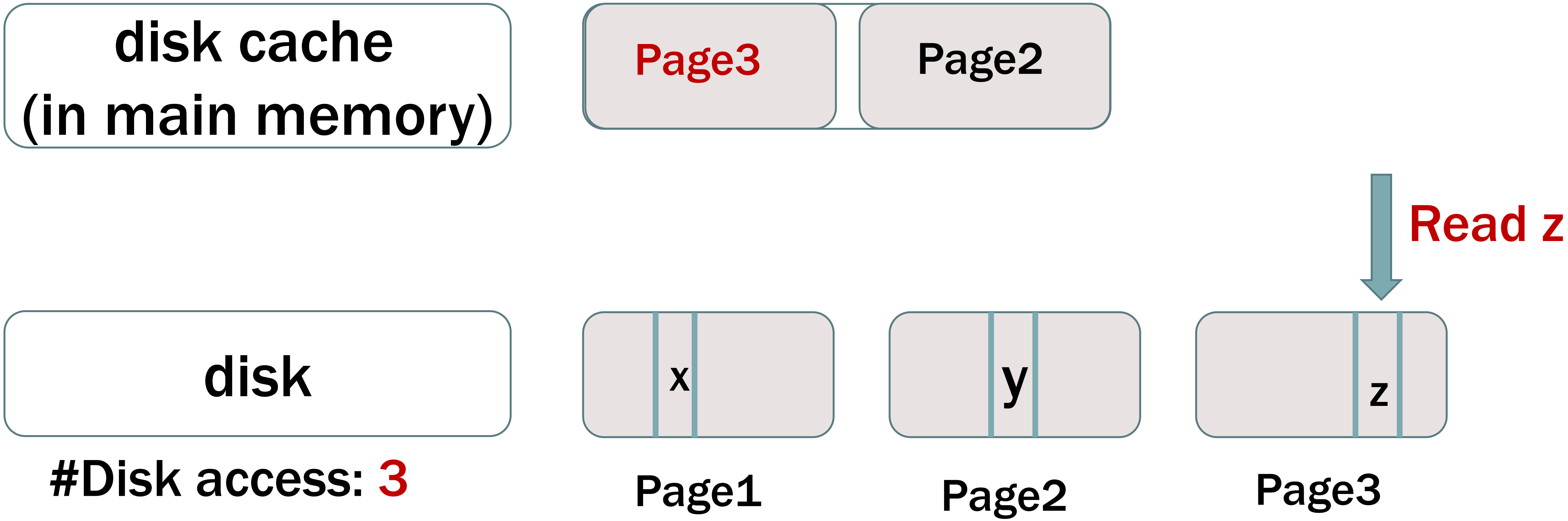
EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page? Now disk cache is full.



EXAMPLE OF PAGE-BASED ACCESS

Now, what if we read the third page? Now disk cache is full.



EXAMPLE: SCANNING ARRAYS

Query x < 4 from the following data

(size=120 bytes)

Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: 40 Bytes

Query x < 4 from the following data

Scan

Qualified results

(size=120 bytes)

Memory Layer i

5, 10, 7, 4, 12

Memory Layer i+1

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

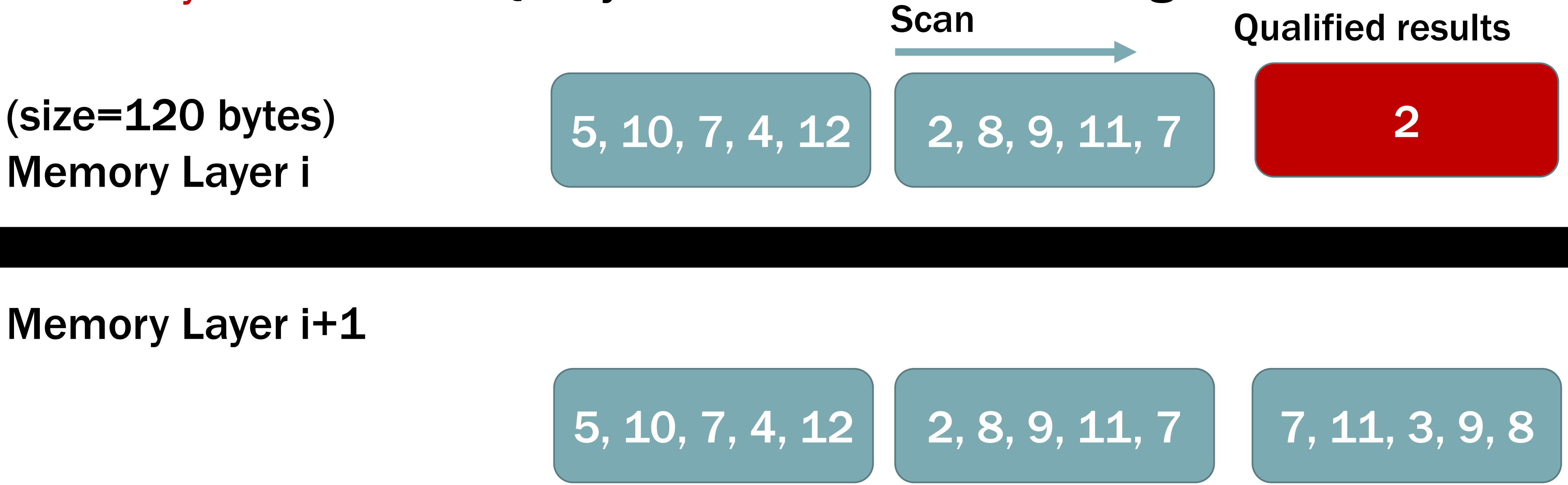
Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: 80 Bytes

Query x < 4 from the following data



Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

EXAMPLE: SCANNING ARRAYS

Cost: 120 Bytes

Query x < 4 from the following data

Scan

Qualified results

(size=120 bytes)
Memory Layer i

7, 11, 3, 9, 8	2, 8, 9, 11, 7	2, 3

Memory Layer i+1

5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

CAN WE DO BETTER ABOUT SCANNING?

- ❑ Consider a cost-free magic function for the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., x<4) within the page.

- ❑ Consider another cost-free magic function:

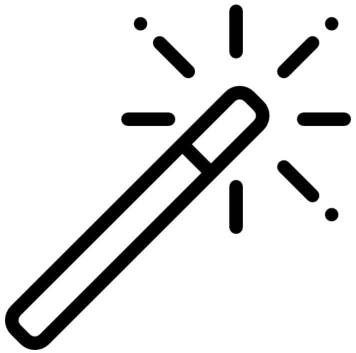
The function tells you **which pages** will contain the qualified keys (i.e., x<4).

- ❖ Can we do better with such a function?
- ❖ Which one do you think is more useful?

TRY THE FIRST MAGIC FUNCTION

❑ Consider a cost-free magic function for the example query:

> By accessing a page, the function immediately tells you the positions of qualified keys (i.e., x<4) within the page.



WITH MAGIC FUNCTION (FIRST)

Query x < 4 from the following data

(size=120 bytes)
Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 40 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)

Memory Layer i

Memory Layer i+1

Magic function read

5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 40 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

5, 10, 7, 4, 12

Memory Layer i+1

Magic function read

bring

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 80 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

5, 10, 7, 4, 12

2

Memory Layer i+1

Magic function read

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 80 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

5, 10, 7, 4, 12

2, 8, 9, 11, 7

2

Memory Layer i+1

Magic function read

bring

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 120 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i	7, 11, 3, 9, 8	2, 8, 9, 11, 7	2, 3

Memory Layer i+1

Magic function read

Memory Layer i+1	5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (FIRST)

Cost: 120 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

7, 11, 3, 9, 8

2, 8, 9, 11, 7

2, 3

Memory Layer i+1

bring

Magic function read

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

CAN WE DO BETTER ABOUT SCANNING?

- ❑ Consider a cost-free magic function regarding the example query:

By accessing a page, the function immediately tells you the positions of qualified keys (i.e., $x<4$) within the page.

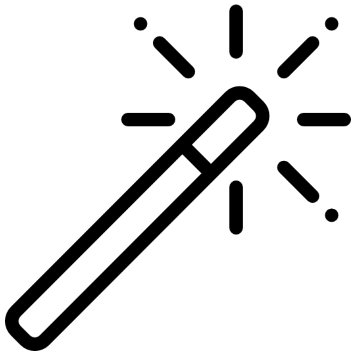
This seemingly good function does not help much. The reason is that the data movement is based on pages.

TRY THE SECOND MAGIC FUNCTION

- ❑ Consider another cost-free magic function:

The function tells you **which pages** will contain the qualified keys (i.e., x<4).
Can we do better with such a function?

In some situations, yes!



WITH MAGIC FUNCTION (SECOND)

Query x < 4 from the following data

(size=120 bytes)

Memory Layer i

Memory Layer i+1

5, 10, 7, 4, 12

2, 8, 9, 11, 7

7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 40 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

Memory Layer i+1

Magic function read Page 2

Page 1	Page 2	Page 3
5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 40 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)

Memory Layer i

2, 8, 9, 11, 7

2

bring

Memory Layer i+1

Magic function read Page 2

5, 10, 7, 4, 12

Page 1

2, 8, 9, 11, 7

Page 2

7, 11, 3, 9, 8

Page 3

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

WITH MAGIC FUNCTION (SECOND)

Cost: 80 Bytes

Query x < 4 from the following data

Qualified results

(size=120 bytes)
Memory Layer i

2, 8, 9, 11, 7	7, 11, 3, 9, 8	2, 3

bring

Memory Layer i+1

Magic function read Page 3

5, 10, 7, 4, 12	2, 8, 9, 11, 7	7, 11, 3, 9, 8
Page 1	Page 2	Page 3

Each integer: 8 bytes

Each page: 8 bytes x 5 = 40 bytes

SUMMARY OF MAGIC FUNCTION EXERCISE

Take-away message

In general, if there is a magic function telling **which pages locate the qualified data**, and if these pages are only a subset of all pages, the read cost will be smaller than directly scanning all pages.

- In practice, we do not have such a magic function with a free cost. Typically, we design indexes, and put the indexes to a faster memory-layer so that its cost is minor compared with the slower memory-layer.