15-213 Recitation
Caches & Blocking

Your TAs

Friday, October 3rd

Reminders

- **`cachelab`** is due ***Thursday (October 9th)***.
- **`malloc`** lab will be released on the same day.
- Take Home midterm is due ***Wednesday (Oct 8th)***
  - Should roughly take 1 hour 20 minutes of your time!
- In Class midterm on ***Tuesday (October 21st)***.
- Drop Deadline: ***Monday (October 6th)***

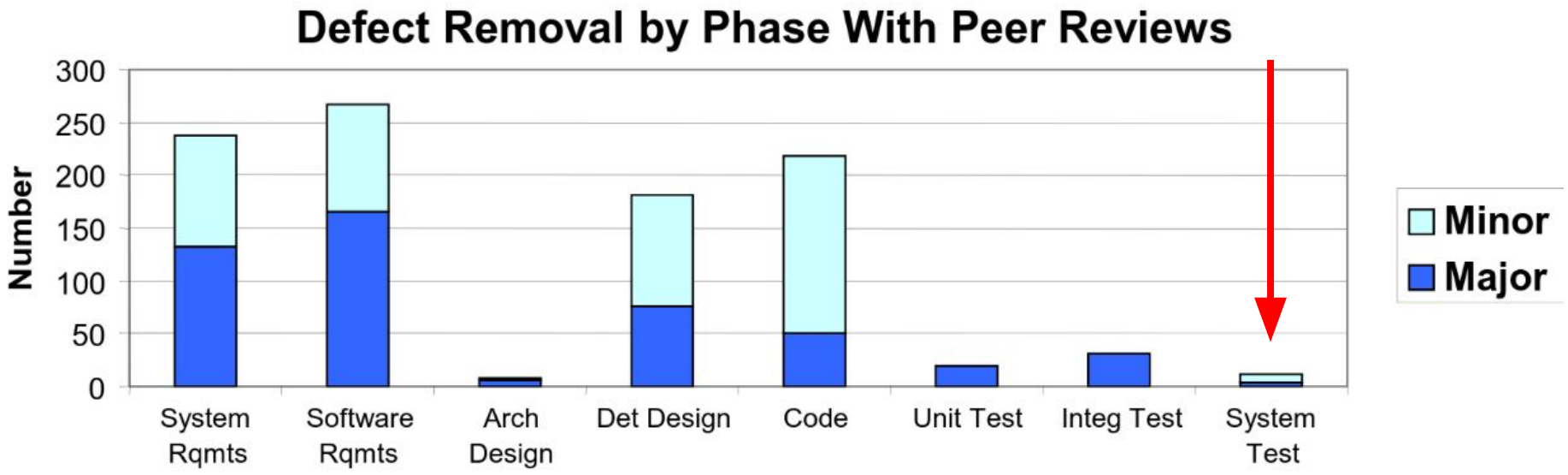
Agenda

- Code Reviews
- Writing Cache-Friendly Code
- Blocking
- Activity: More Blocking!

Code Reviews

Why Code Reviews?

- Used in industry
  - Nearly all companies use code reviews
  - Effective at finding bugs
- Sets you up for success in future systems courses!



Roger G. [Aug 2005]

Code Reviews: Logistics

- Each of you will be assigned a Code Review TA.
- Starting with **`cachelab`**, you will receive style points (0-4) on each lab.
- Watch for an email from your TA so that you can sign up for a meeting slot!
  - Meetings are short (<= 15 minutes)!

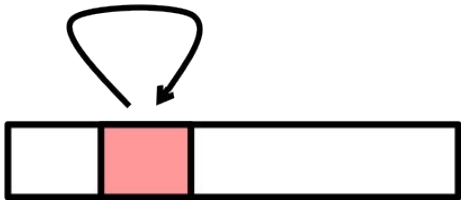
Code Reviews: Guidelines

- First: look at Official 213 Style Guide!
- **Documentation** (comments, *file header*)
- **Modularity**
  - Use helper functions!
  - Avoid magic numbers (use **`#define`** or **`static const`**)
- **Correctness**
  - **`malloc`** can fail! Library functions can fail!
  - Are you leaking memory/file descriptors?

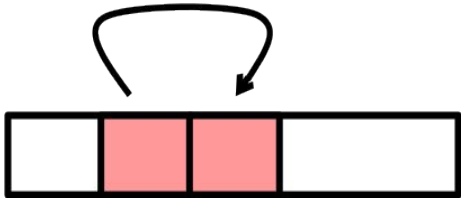
Writing Cache-Friendly Code

Recall: Temporal and Spatial Locality

- ***Temporal Locality:***
  - *Recently referenced* items are likely to be referenced again soon!

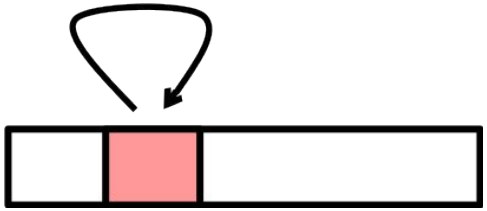


- ***Spatial Locality:***
  - Items with *nearby addresses* tend to be referenced close together in time.



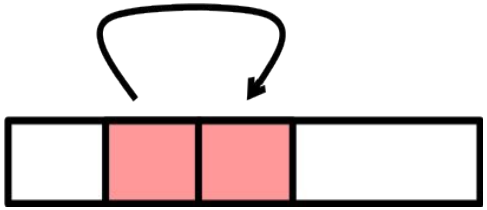
Optimizing for Locality

Temporal



- *Recently referenced* items are likely to be referenced again soon!
- To optimize: try to use data objects as often as possible once they're read from memory.
- Lecture Example: *Blocking*.

Spatial



- Items with *nearby addresses* tend to be referenced close together in time.
- To optimize: read objects sequentially, and with smaller stride.
- Lecture Example: *Rearranging loops*.

Review: Cache Configurations

Cache Configurations

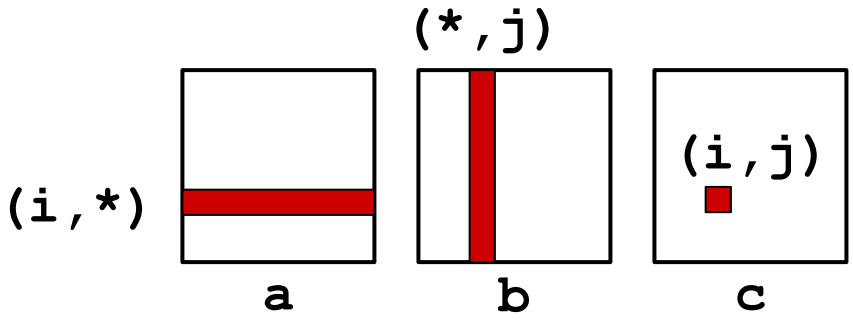
- **Direct Mapped**: Cache with **`E=1`**
  - one line per set
- **Fully Associative**: Cache with **`s=0`**
  - all lines in one set
- **k-way Associative**: Cache with **`E=k`**
  - k lines per set

Blocking

Example: Matrix Multiplication

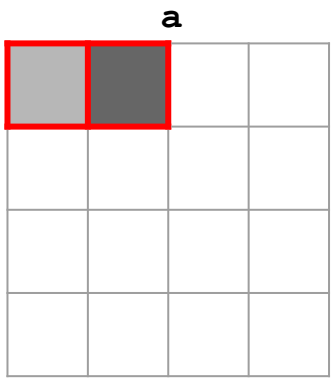
```
/* Multiply 4x4 matrices */
void mm(int a[4][4], int b[4][4], int c[4][4]) {
    int i, j, k;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            for (k = 0; k < 4; k++)
                c[i][j] += a[i][k] * b[k][j];
```

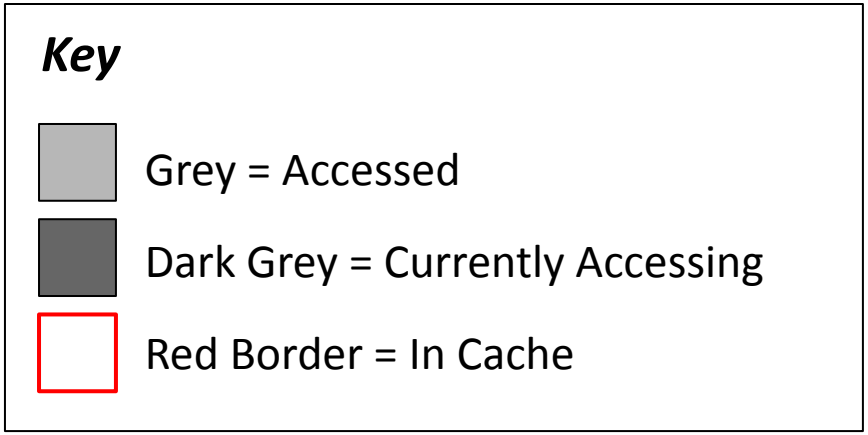
- "Standard" way of doing matrix multiplication (**ijk**):
  - **c[i][j]** is given by taking "dot product" of **i**-th row of **a** with **j**-th column of **b**.



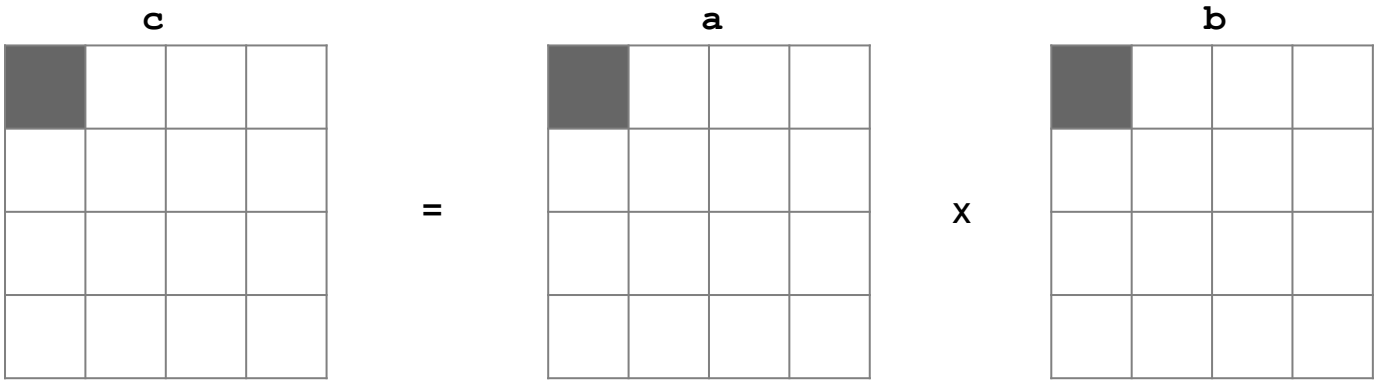
Example: Matrix Multiplication

- Assume a tiny cache with 4 lines of 8 bytes (2 **`int`**s each)
  - **`S`** = 1, **`E`** = 4, **`B`** = 8
- We'll use the following key:





- Let's see what happens if we don't use blocking…

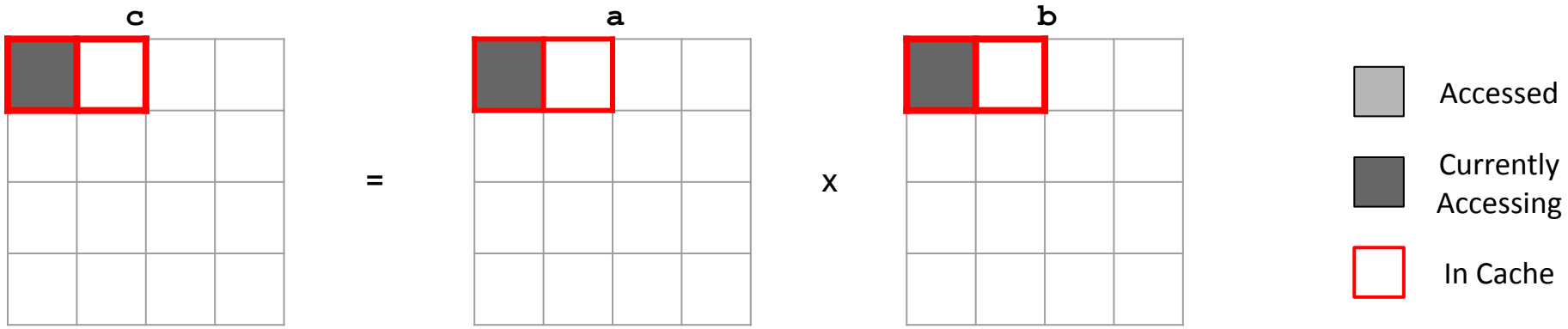


Accessed

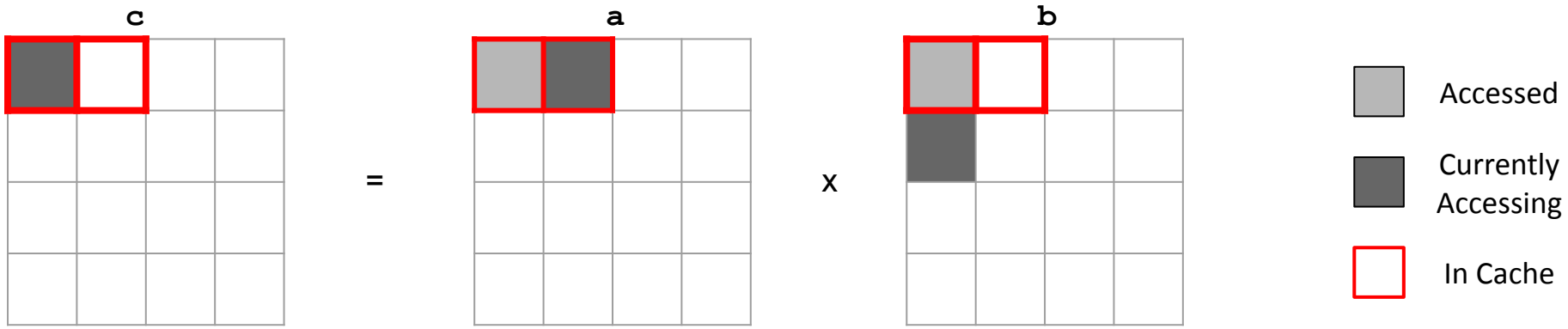
Currently Accessing

In Cache

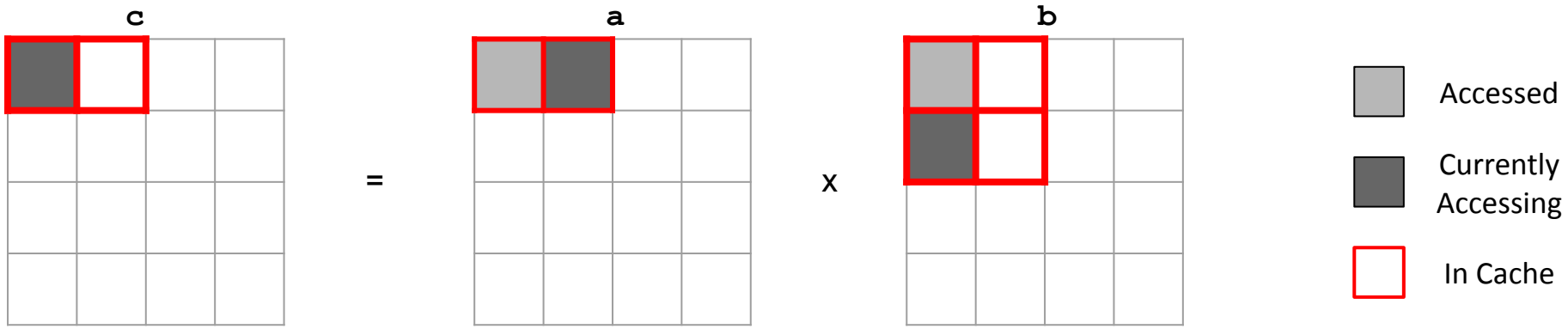
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	???



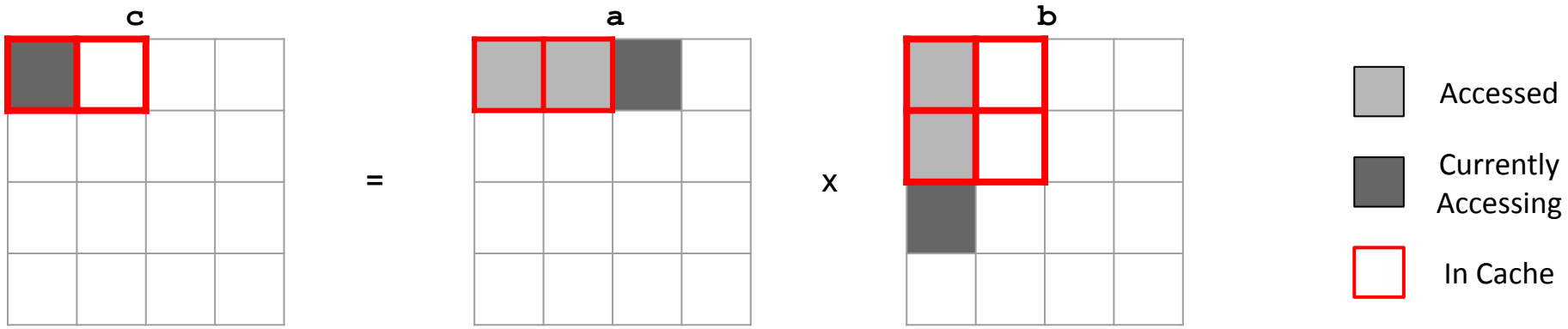
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)



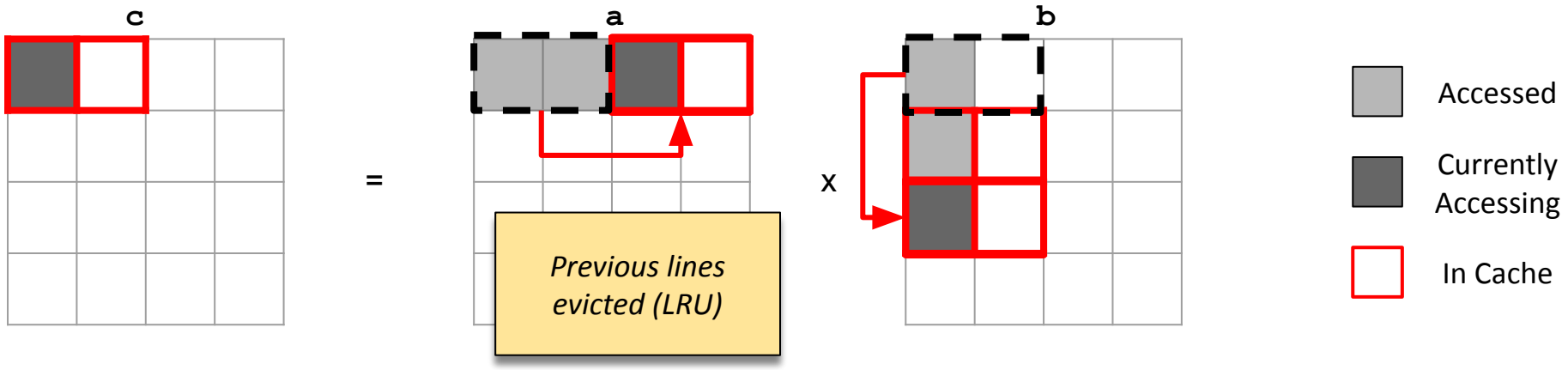
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	???



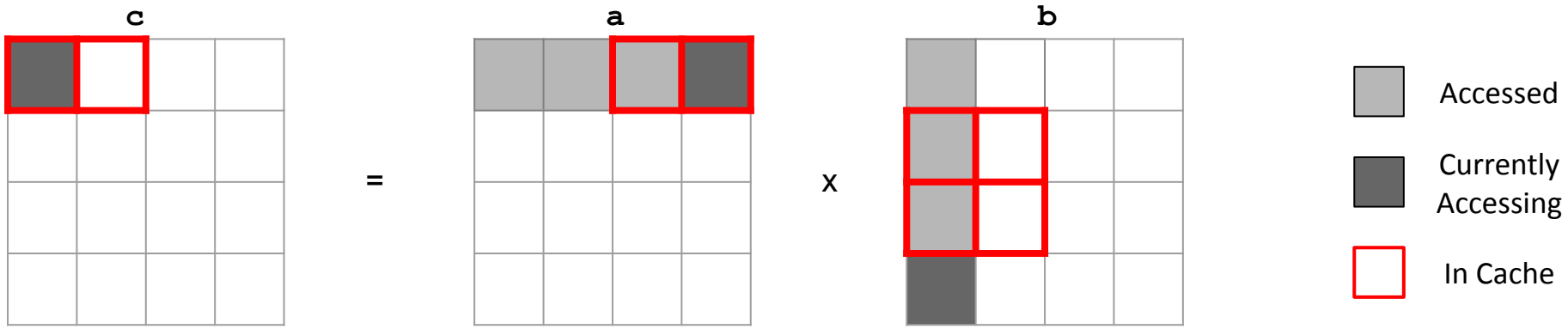
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)



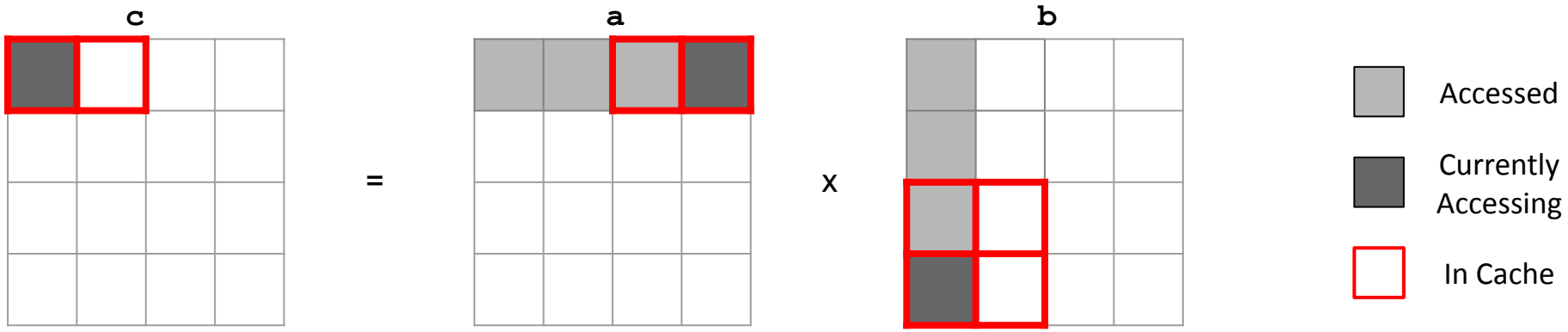
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	???



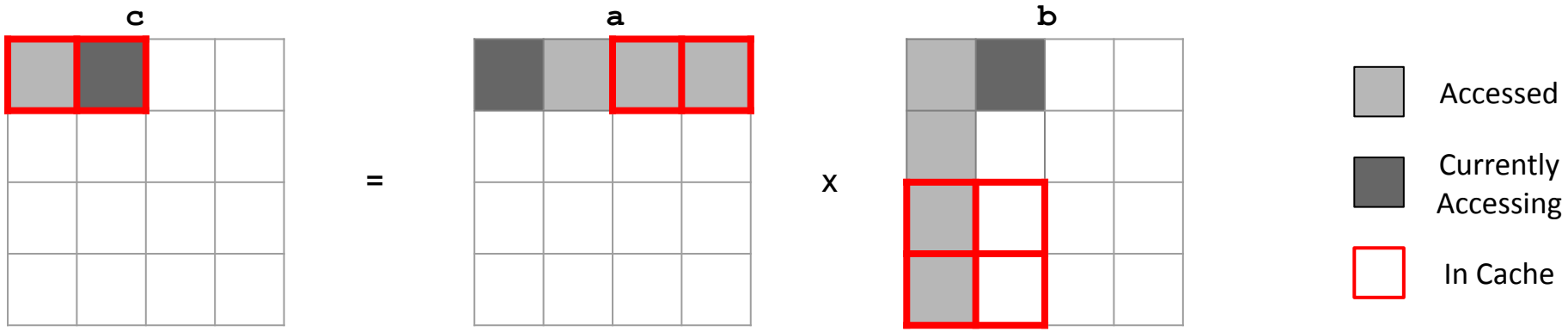
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)



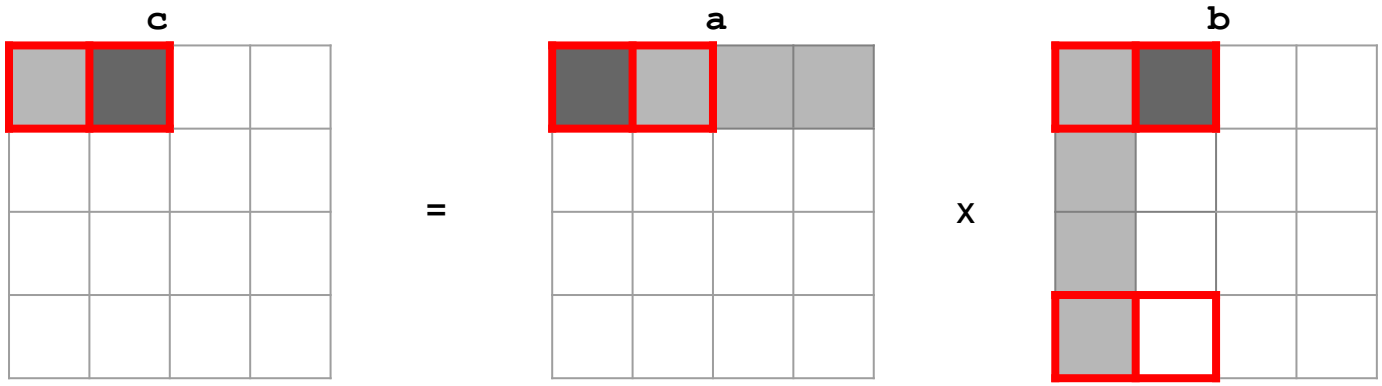
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	???



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)

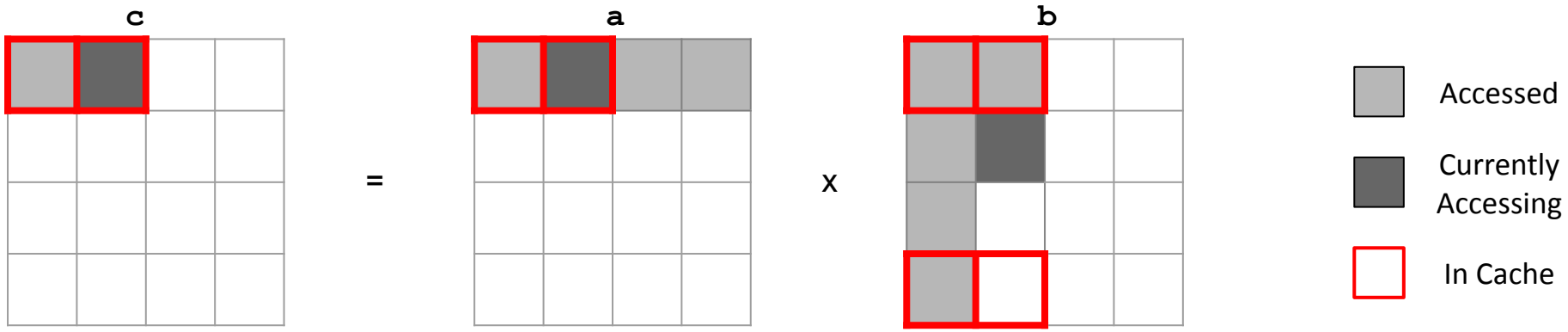


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	???

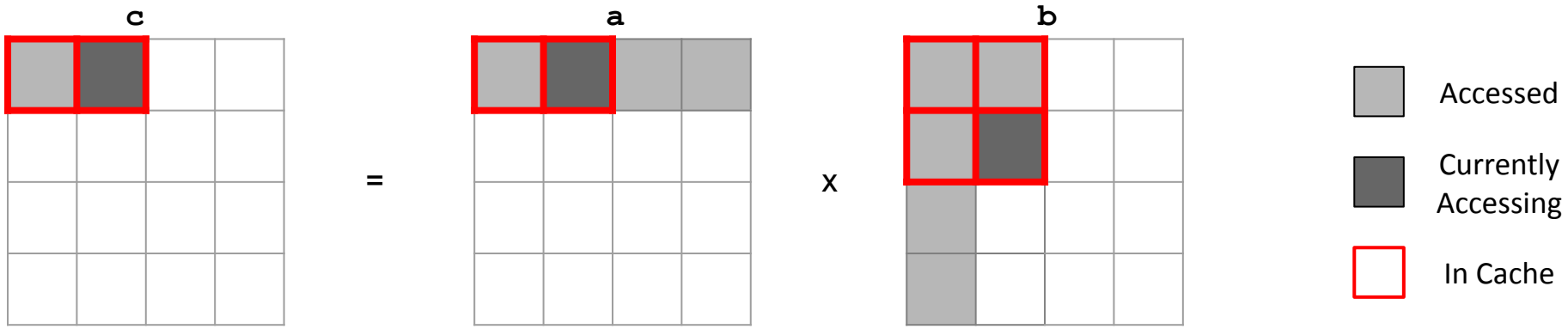


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)

Have these blocks been in the cache before?

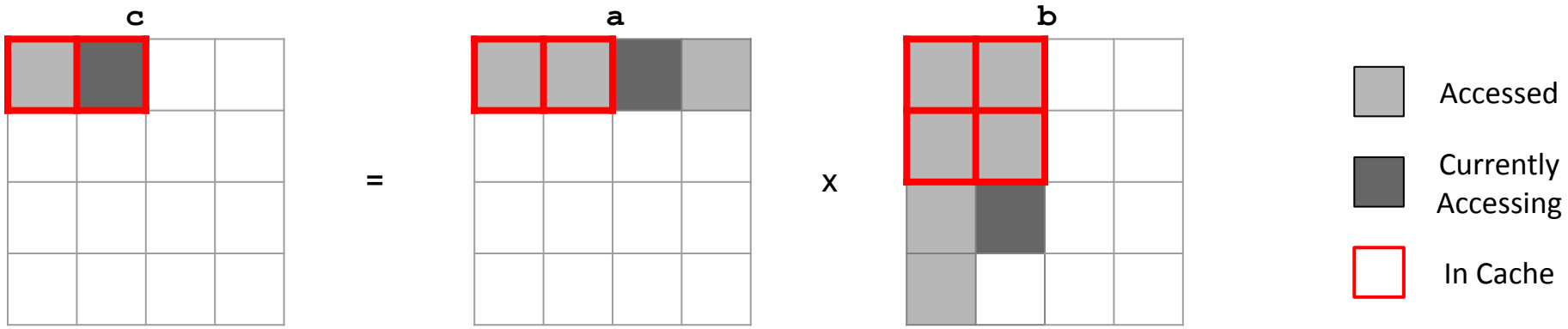


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	???

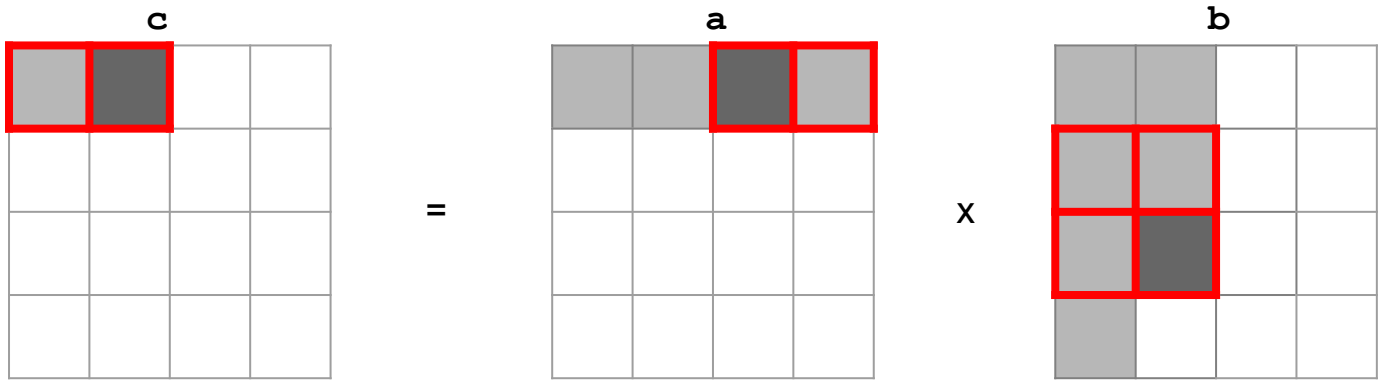


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, m)

Has this block been in cache before?

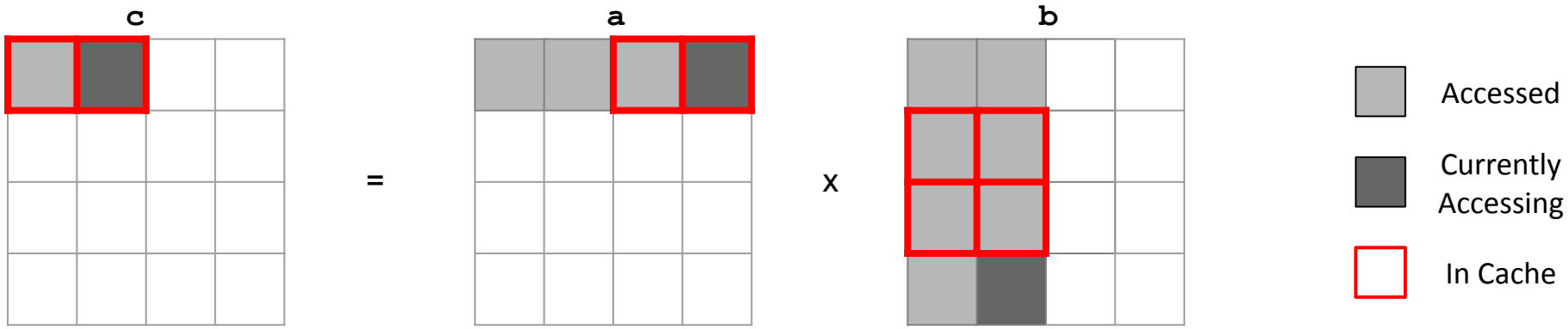


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, m)
6	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	???

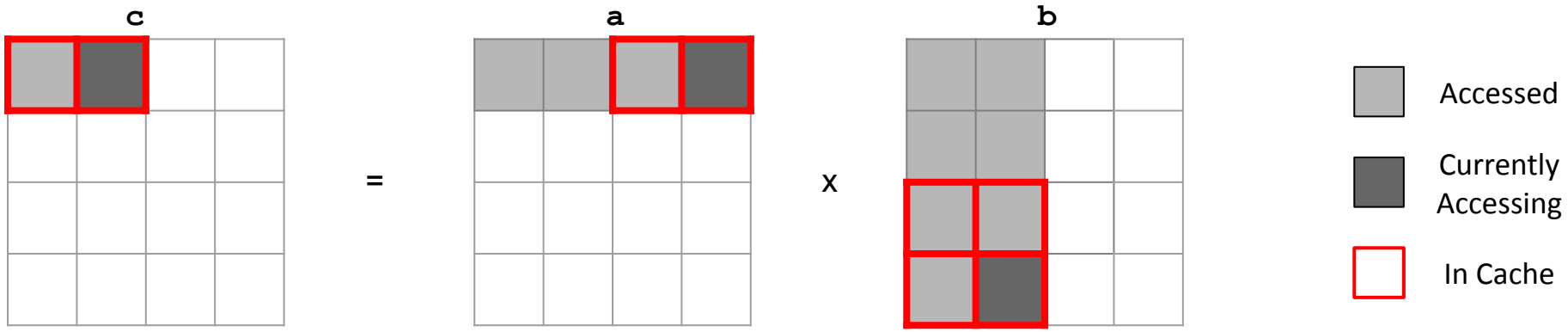


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, m)
6	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(m, m)

Have these blocks been in the cache before?



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, m)
6	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(m, m)
7	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	???



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, m)
6	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(m, m)
7	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, m)

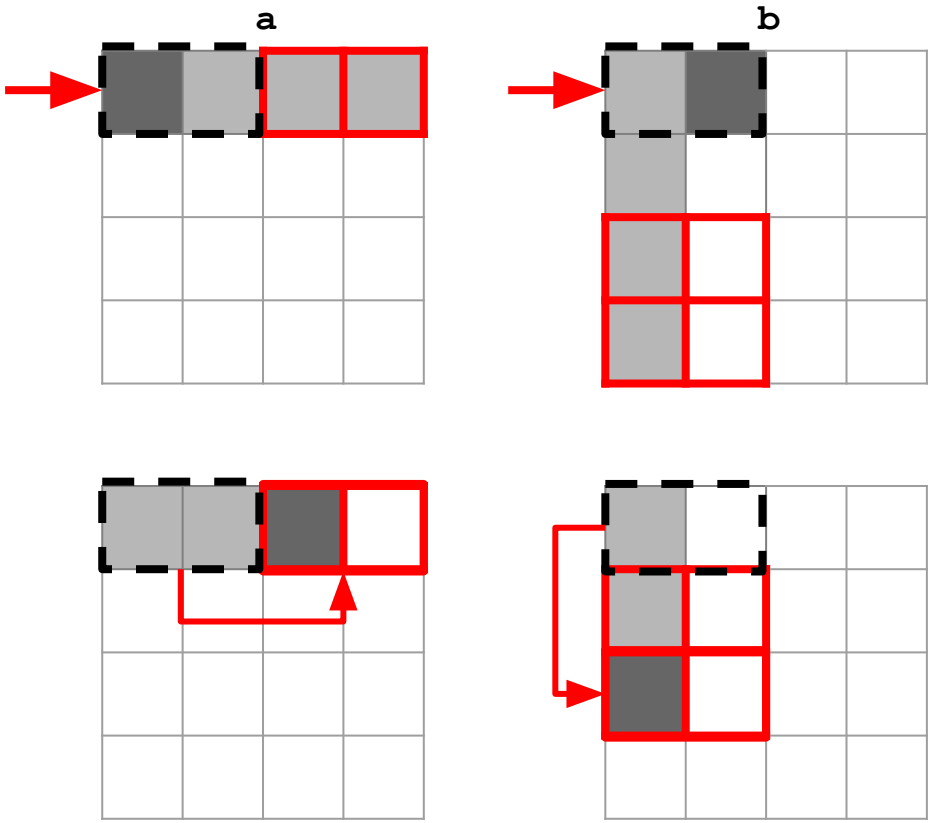
No Blocking: Analyzing Miss Rate

Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, m)
6	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(m, m)
7	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, m)

- What is the miss rate of **`a`**?
  - 4/8 = ***50%***
- What is the miss rate of **`b`**?
  - 8/8 = ***100%***

No Blocking: What went Wrong?

- Bad temporal locality!
- Blocks are used multiple times, but are never in cache when we need them.

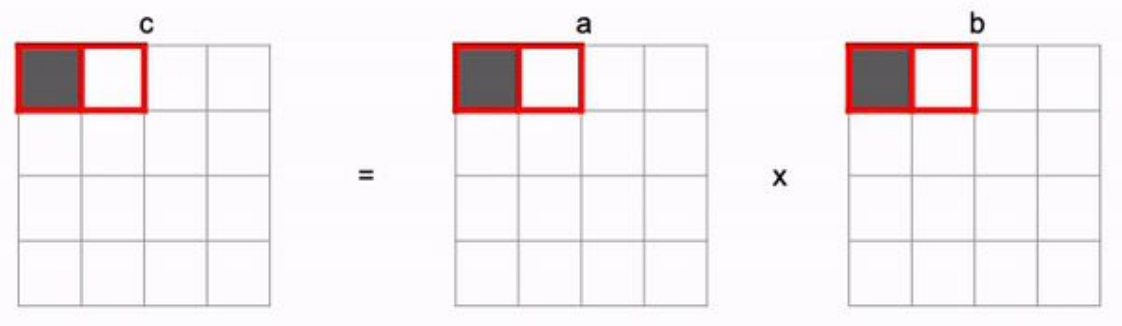


Misses on Iteration 4

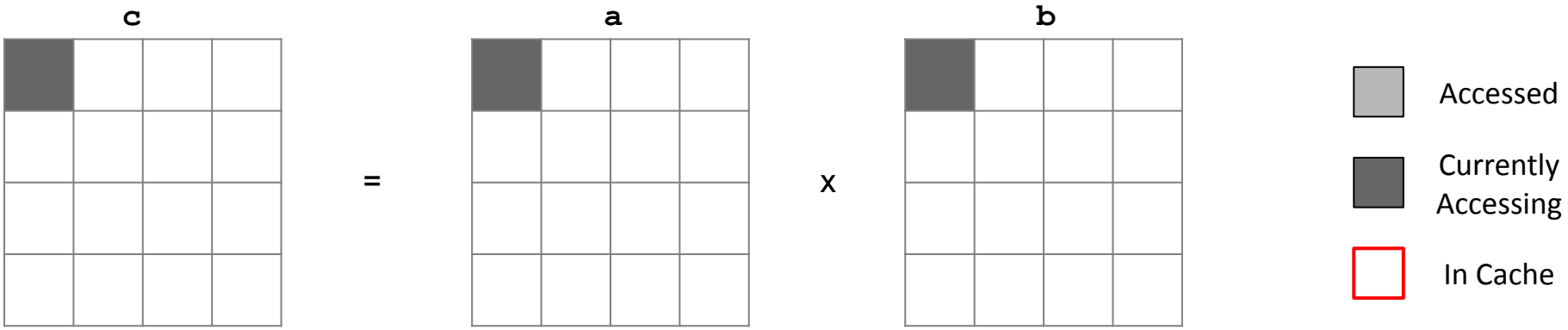
Evictions on Iteration 2

Example: Matrix Multiplication (with Blocking)

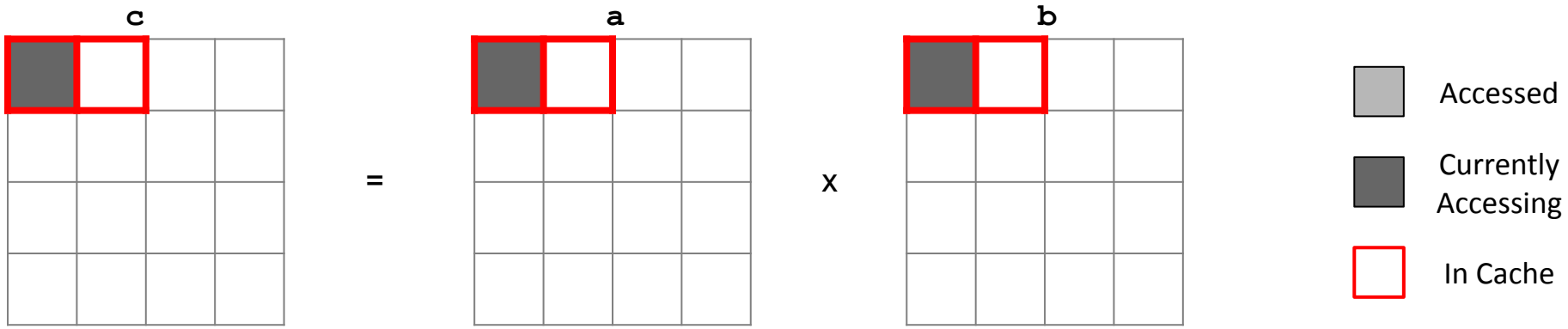
```
/* multiply 4x4 matrices using blocks of size 2 */
void mm_blocking(int a[4][4], int b[4][4], int c[4][4]) {
    int i, j, k;
    int i_c, j_c, k_c;
    int B = 2;
    // control loops
    for (i_c = 0; i_c < 4; i_c += B)
        for (j_c = 0; j_c < 4; j_c += B)
            for (k_c = 0; k_c < 4; k_c += B)
                // block multiplications
                for (i = i_c; i < i_c + B; i++)
                    for (j = j_c; j < j_c + B; j++)
                        for (k = k_c; k < k_c + B; k++)
                            c[i][j] += a[i][k] * b[k][j];
```



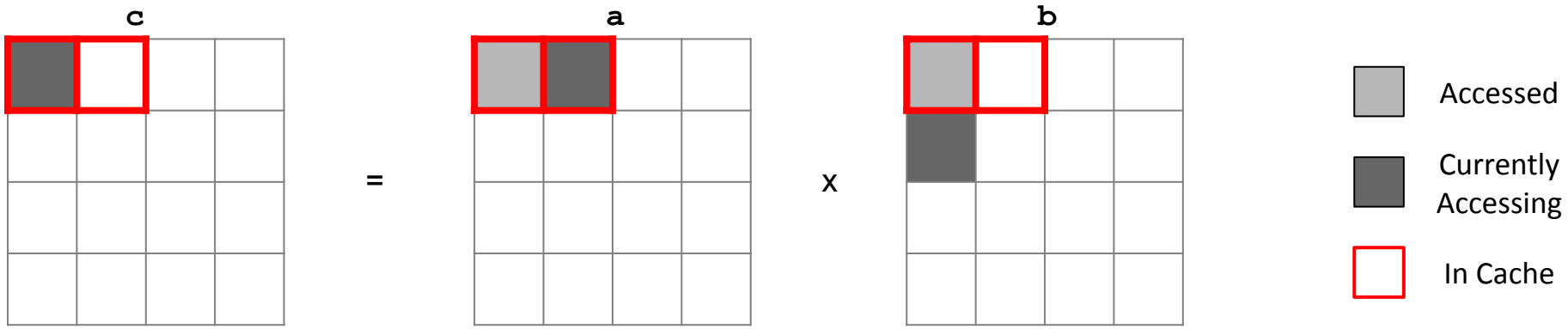
- Let's see what happens if we use blocking!



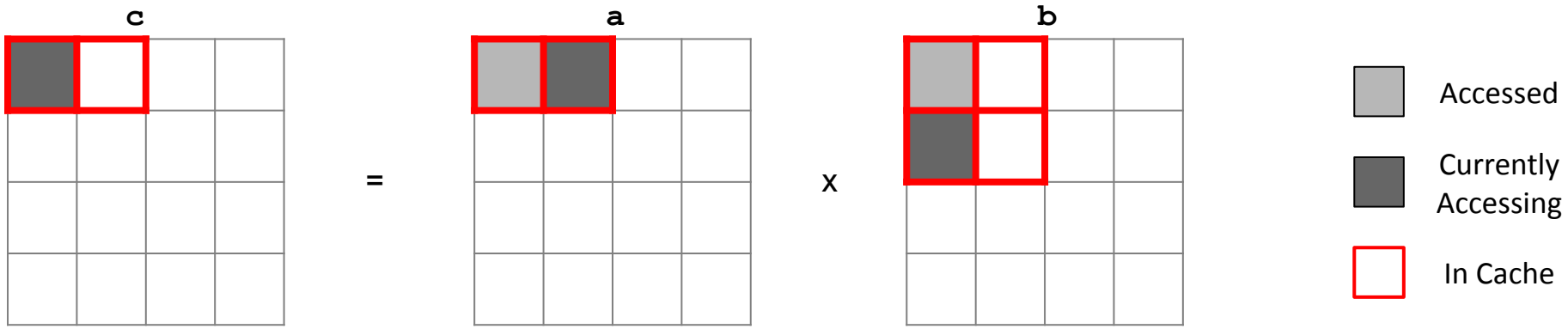
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	???



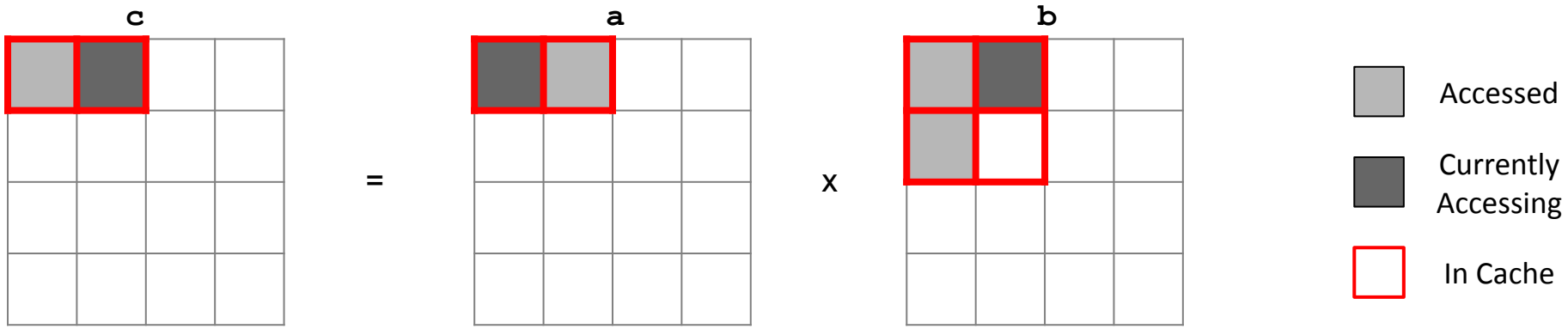
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)



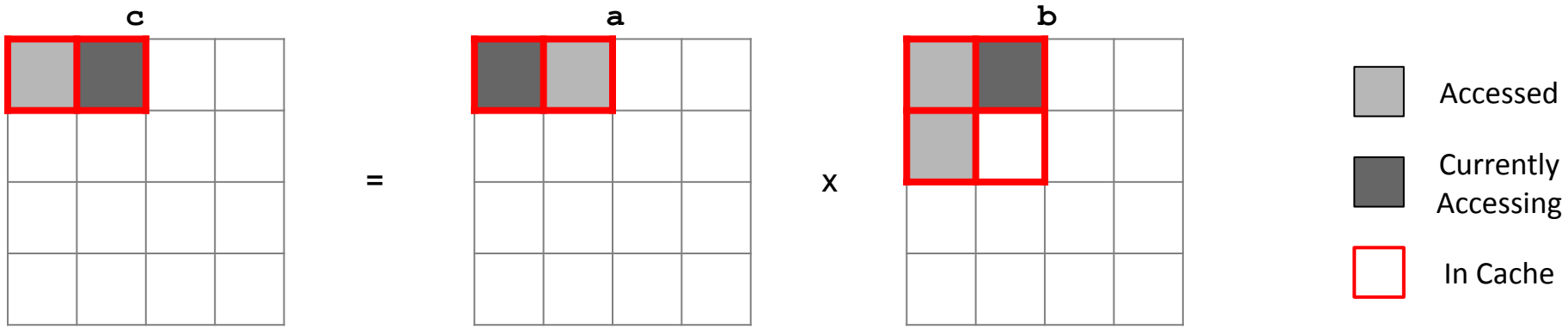
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	???



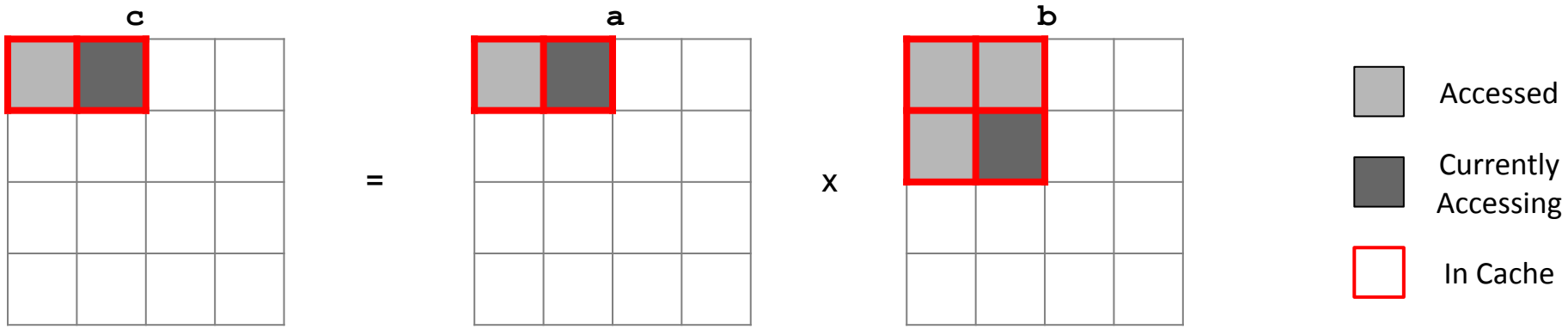
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)



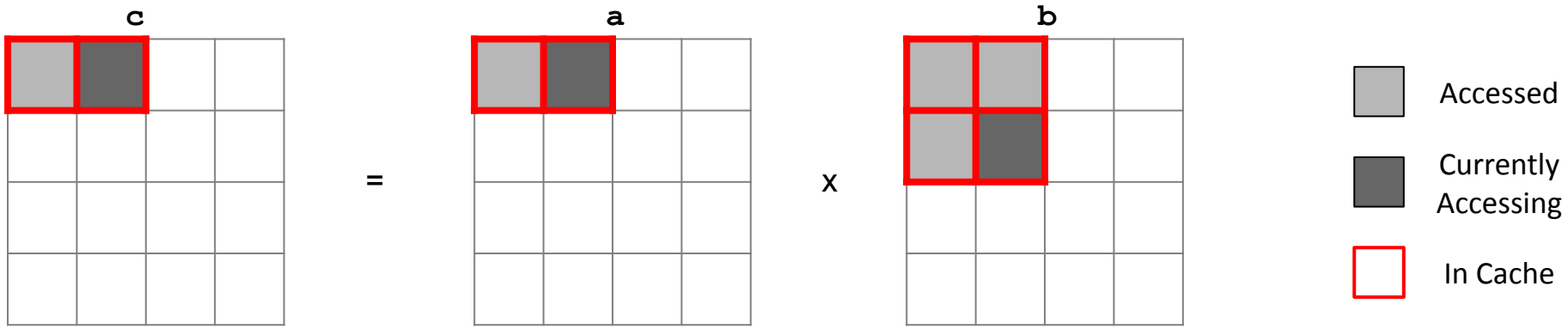
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	???



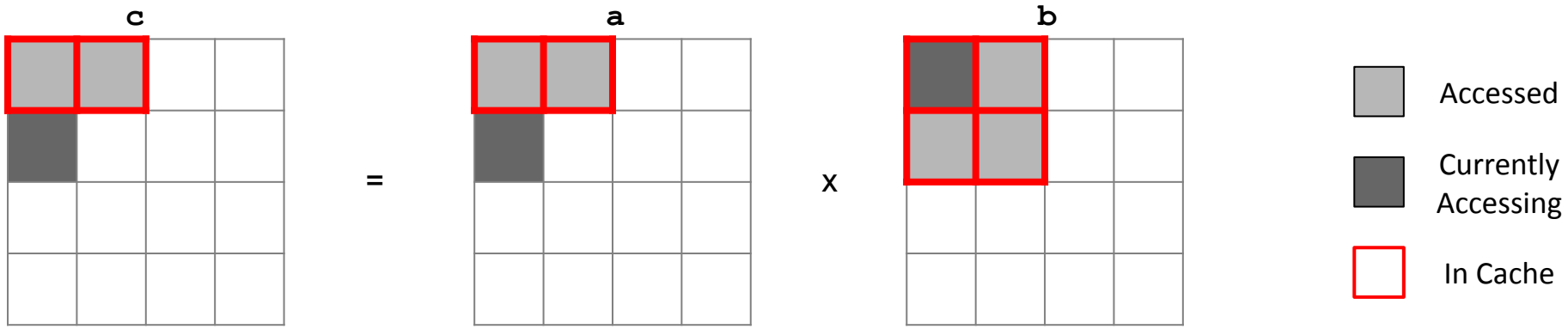
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)



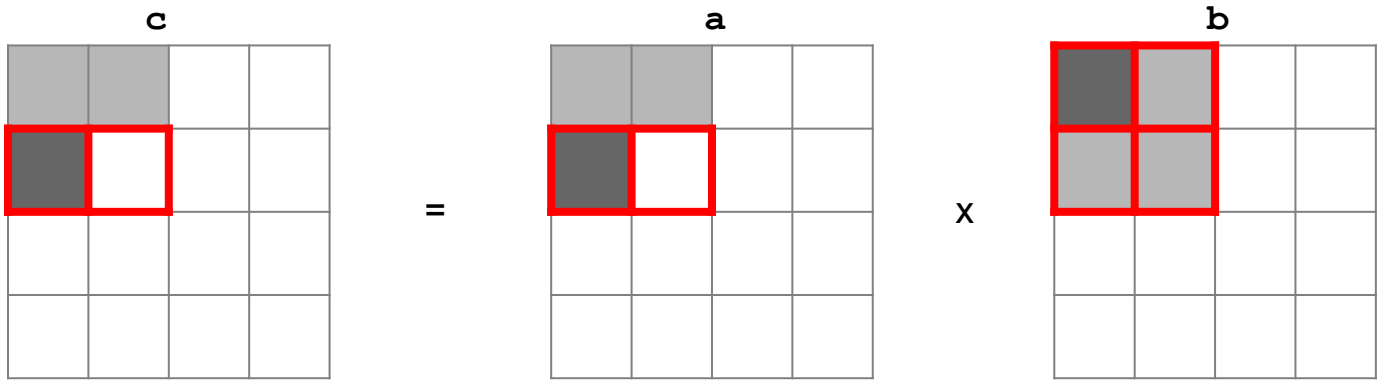
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	???



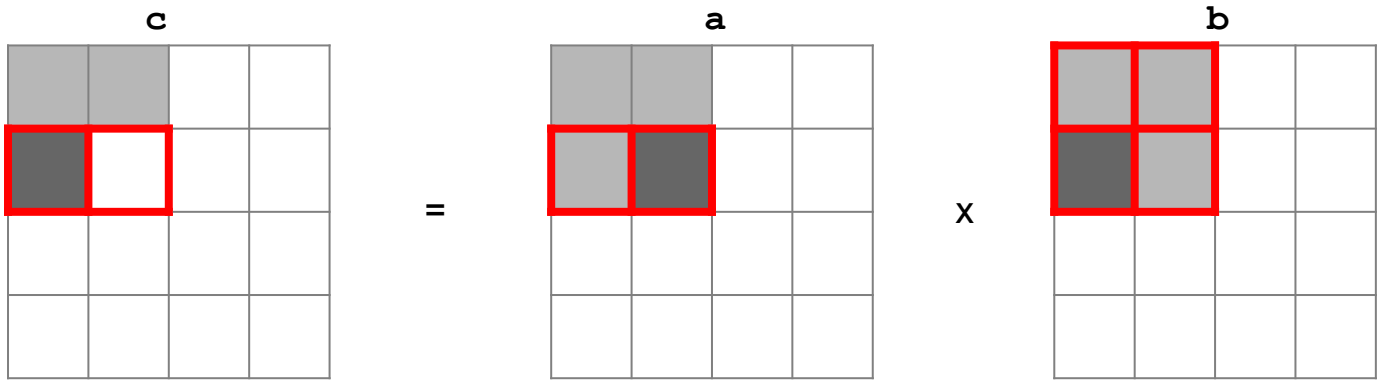
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)



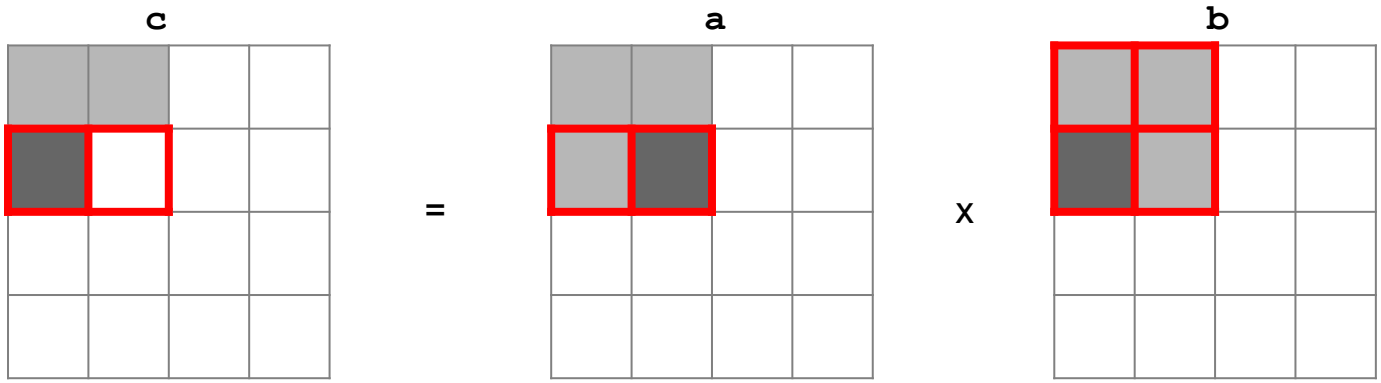
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	???



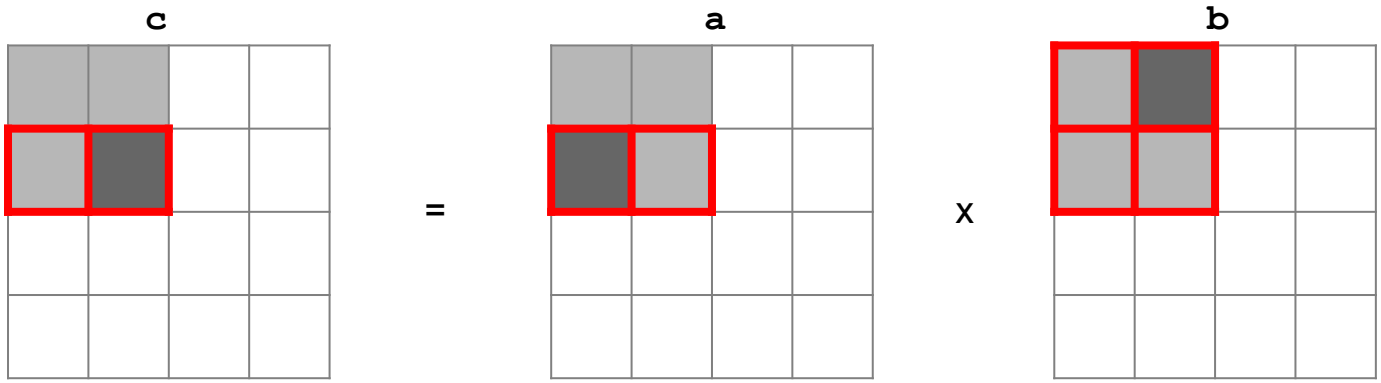
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)



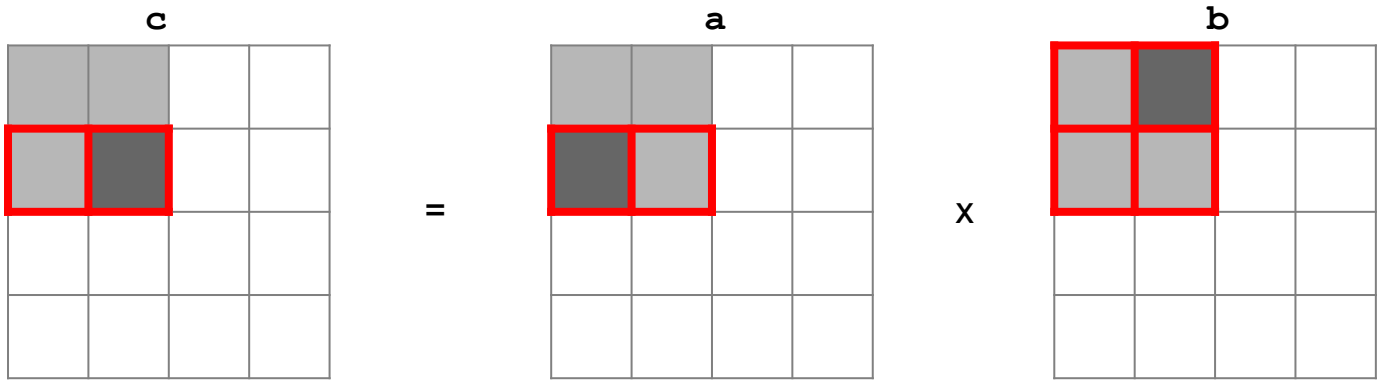
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	???



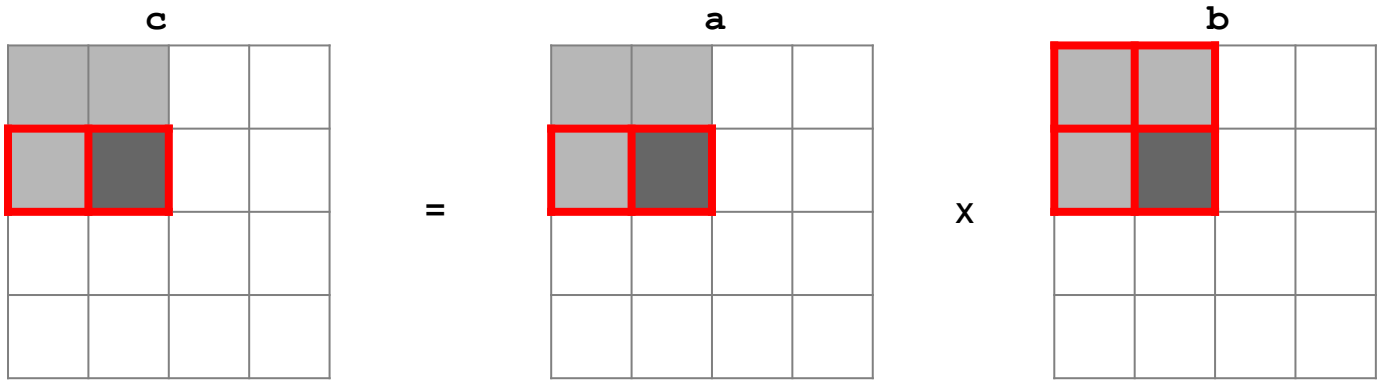
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	(h, h)



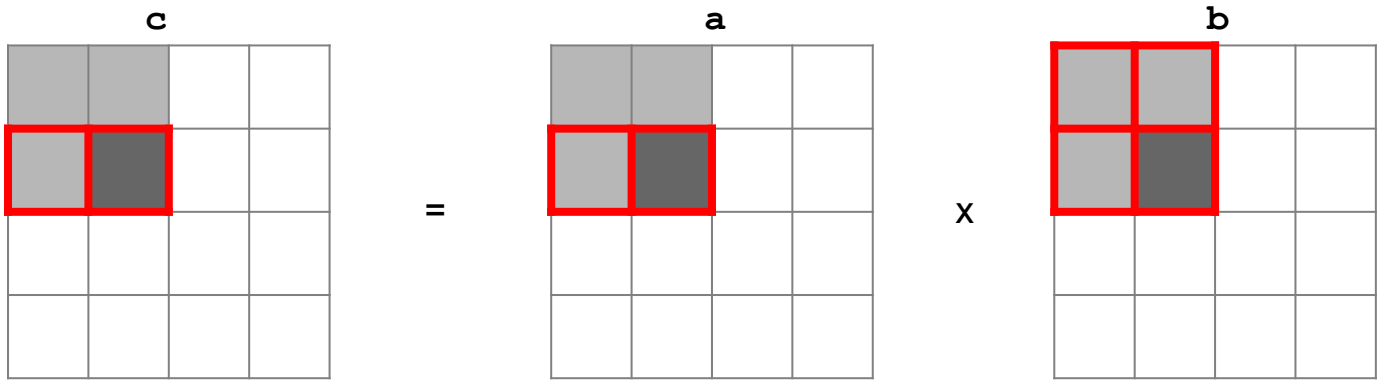
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	(h, h)
6	1	1	0	`c[1][1] += a[1][0] * b[0][1]`	???



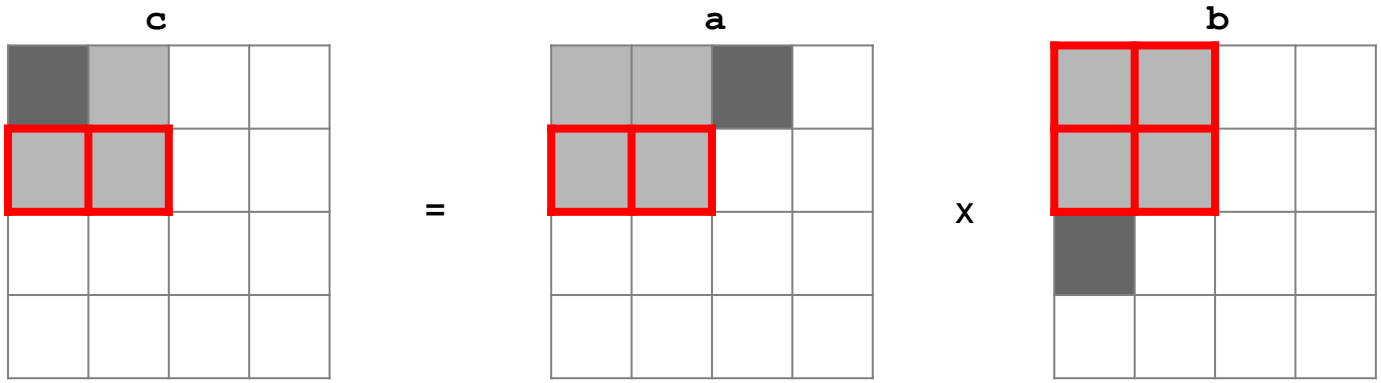
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	(h, h)
6	1	1	0	`c[1][1] += a[1][0] * b[0][1]`	(h, h)



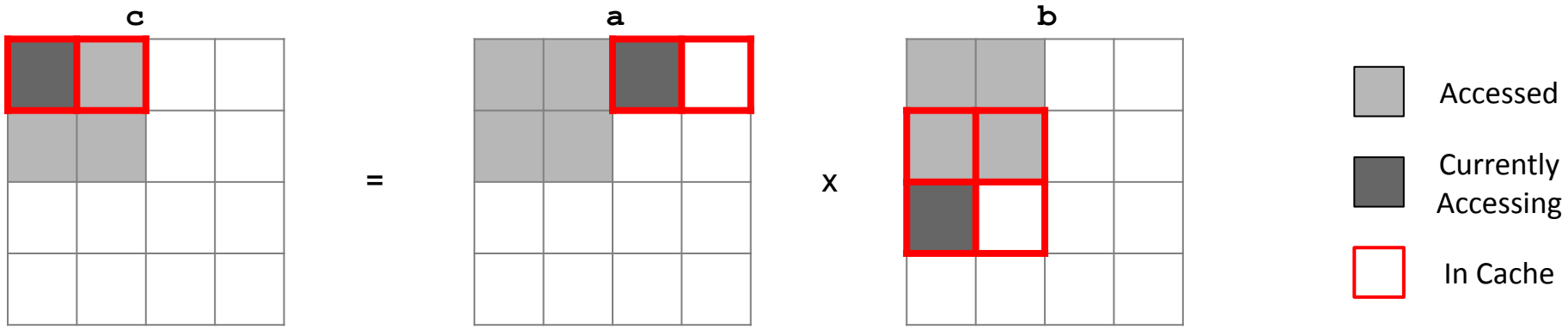
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	(h, h)
6	1	1	0	`c[1][1] += a[1][0] * b[0][1]`	(h, h)
7	1	1	1	`c[1][1] += a[1][1] * b[1][1]`	???



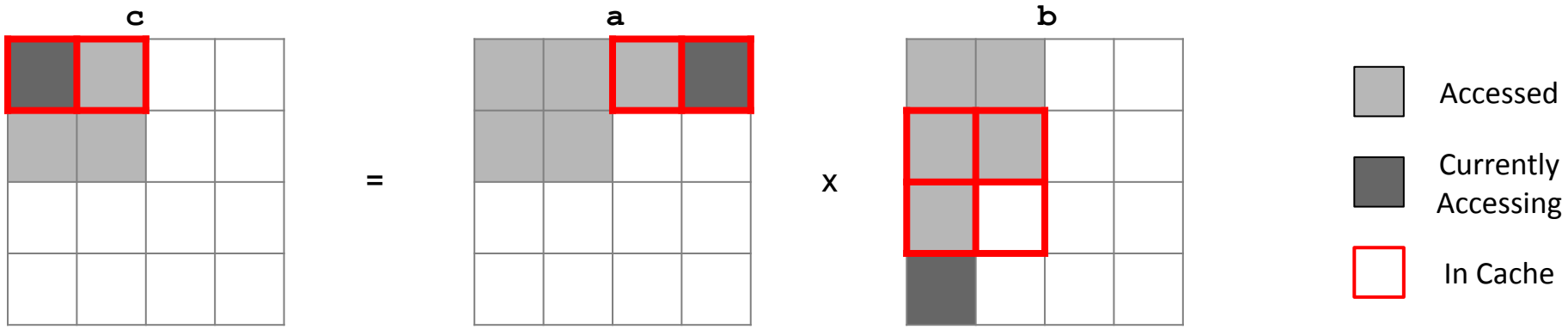
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] * b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] * b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] * b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] * b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] * b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] * b[1][0]`	(h, h)
6	1	1	0	`c[1][1] += a[1][0] * b[0][1]`	(h, h)
7	1	1	1	`c[1][1] += a[1][1] * b[1][1]`	(h, h)



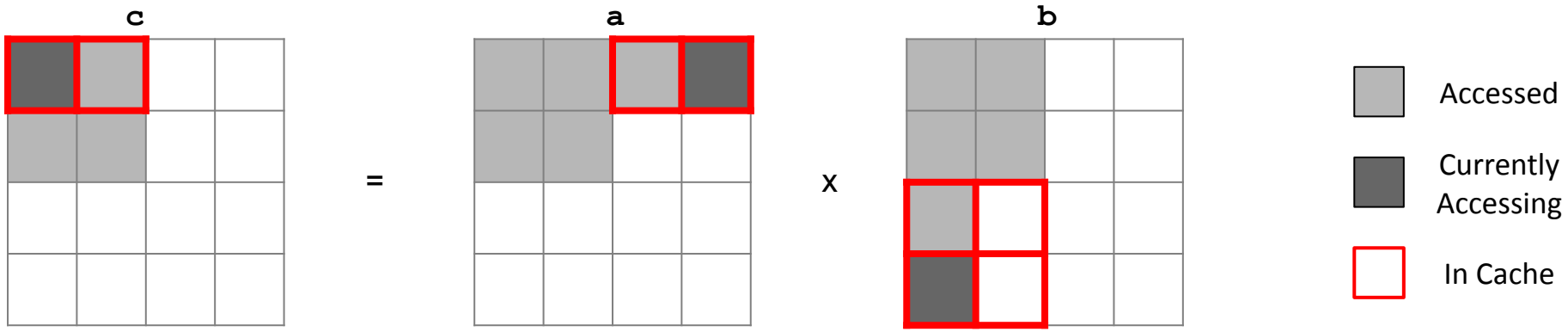
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	???



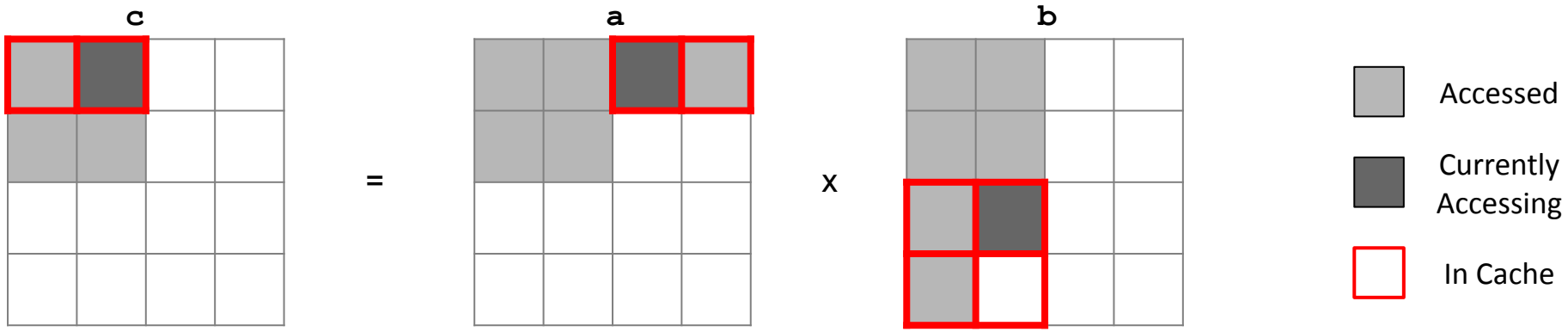
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)



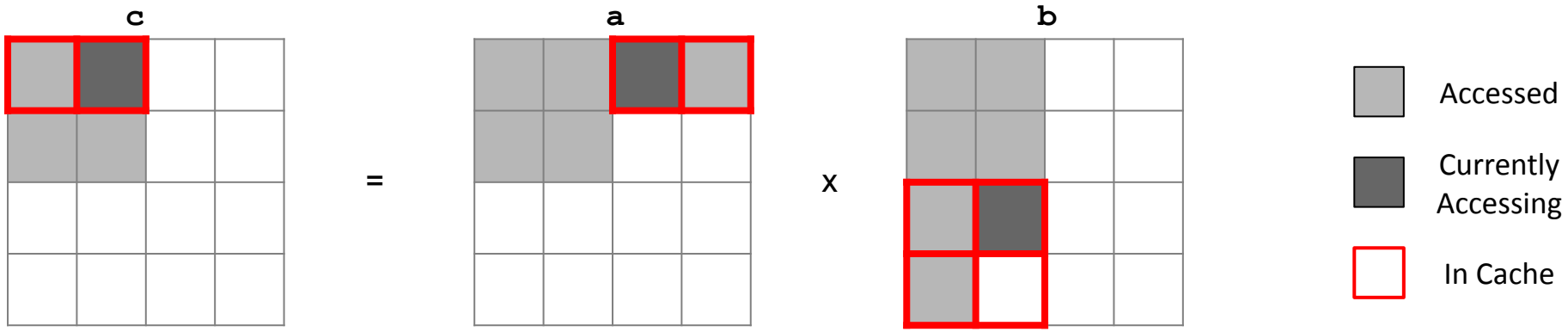
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	???



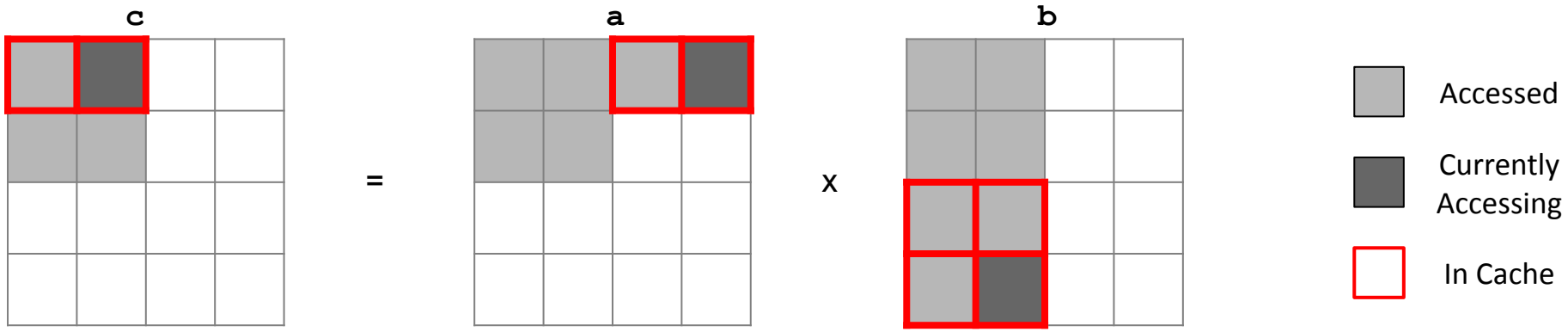
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)



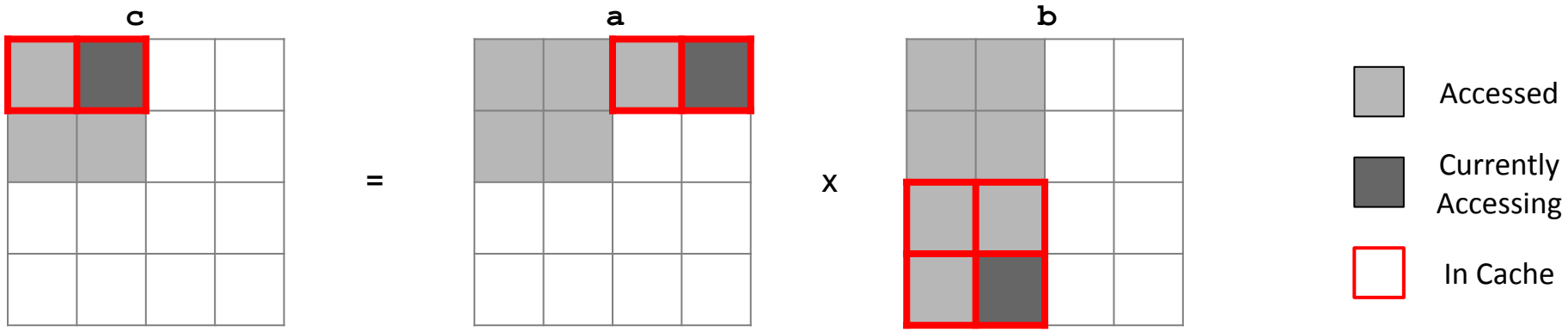
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	???



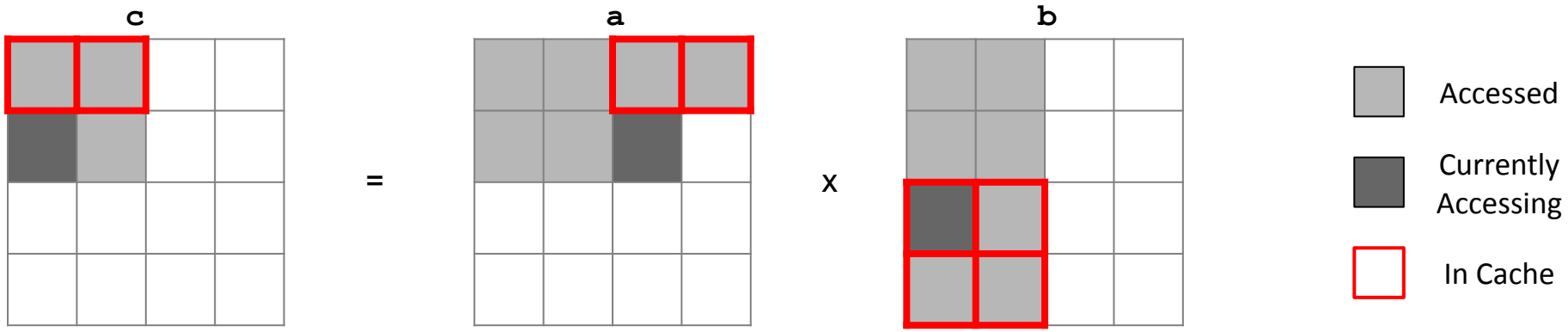
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)



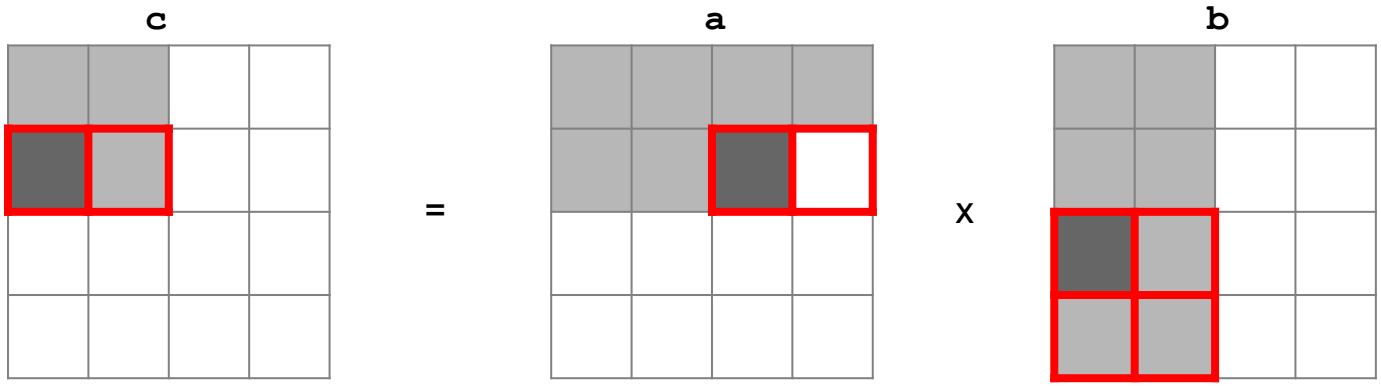
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	???



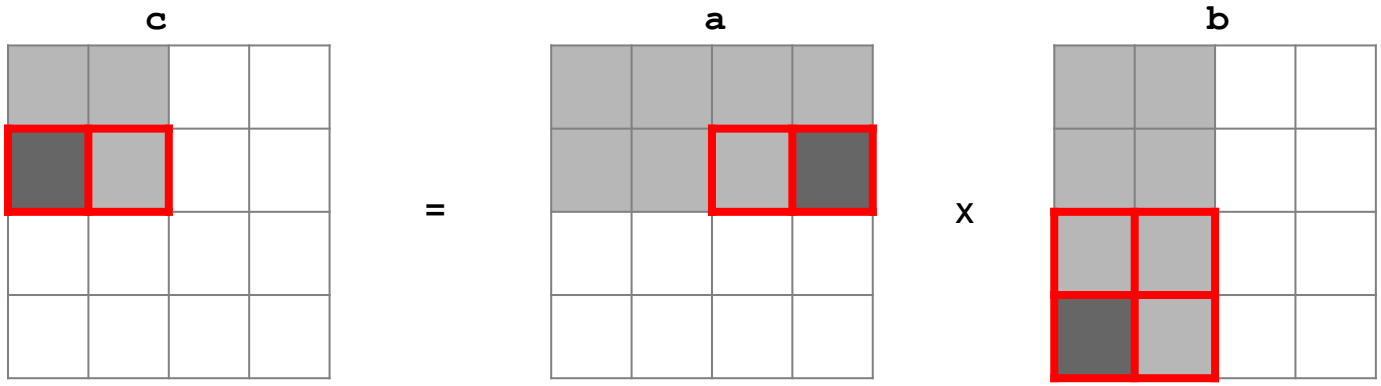
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)



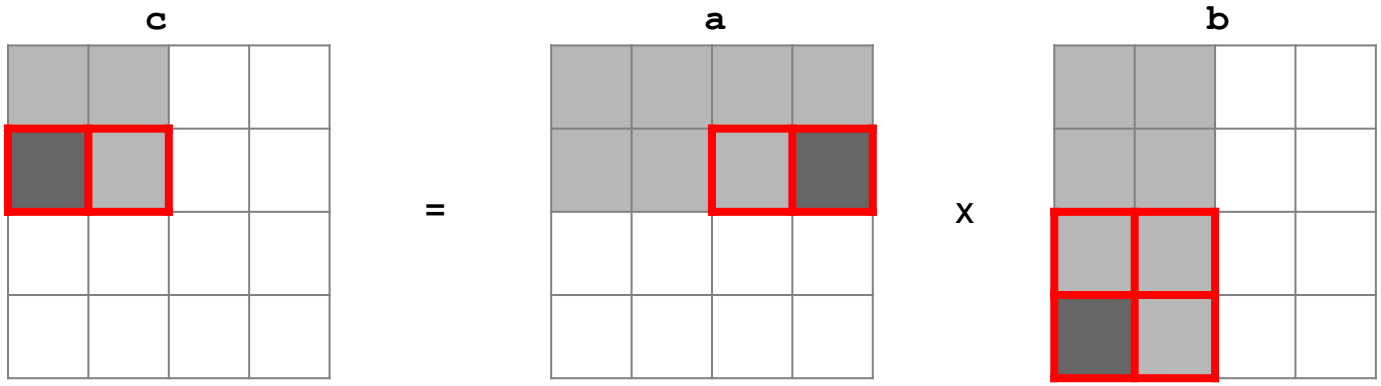
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	???



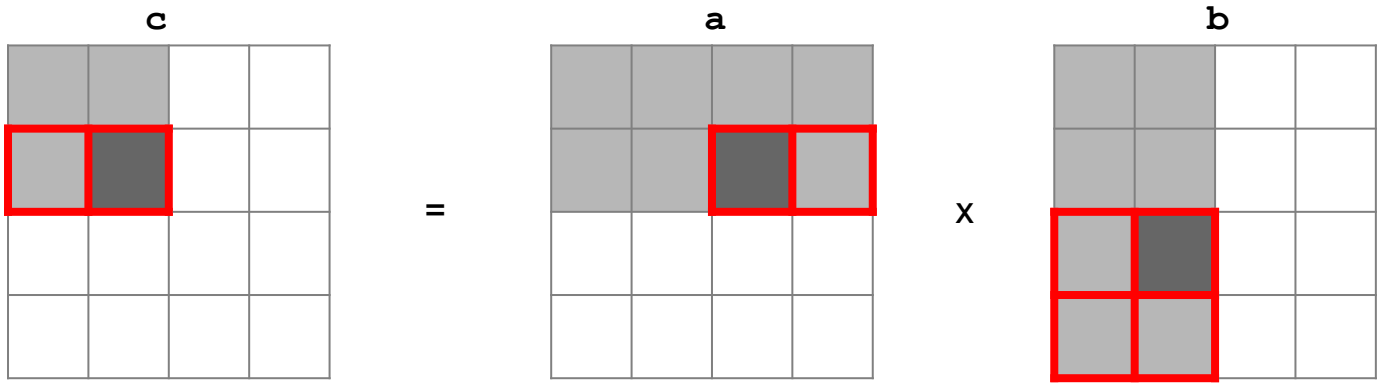
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)



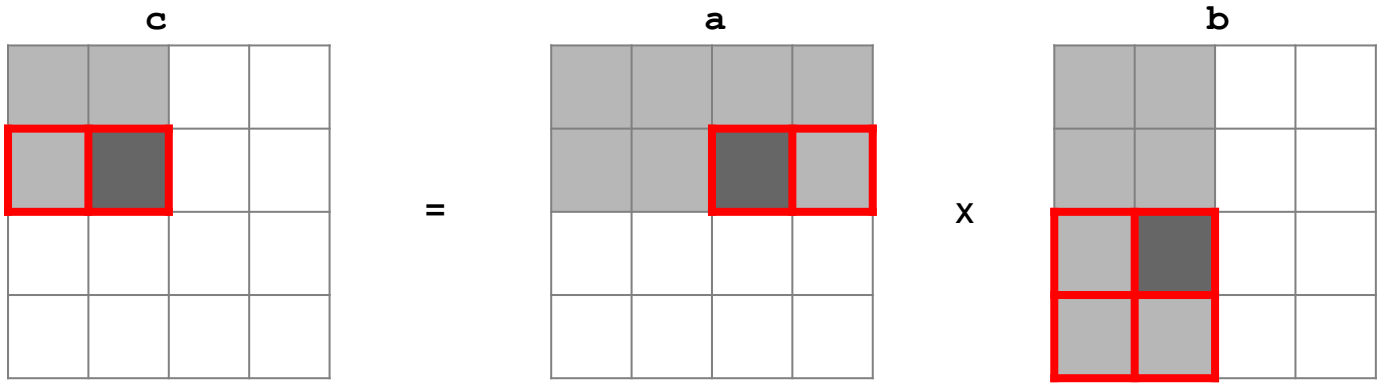
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	???



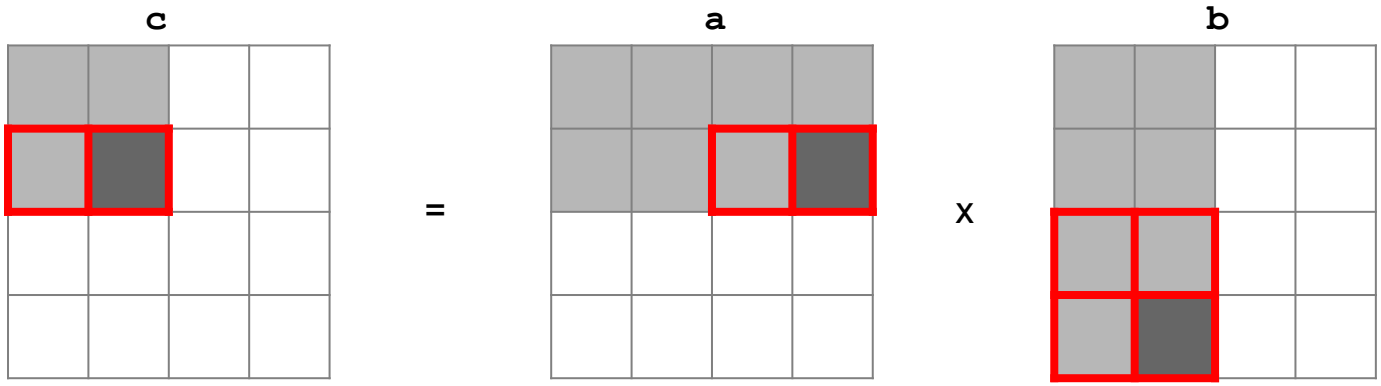
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	(h, h)



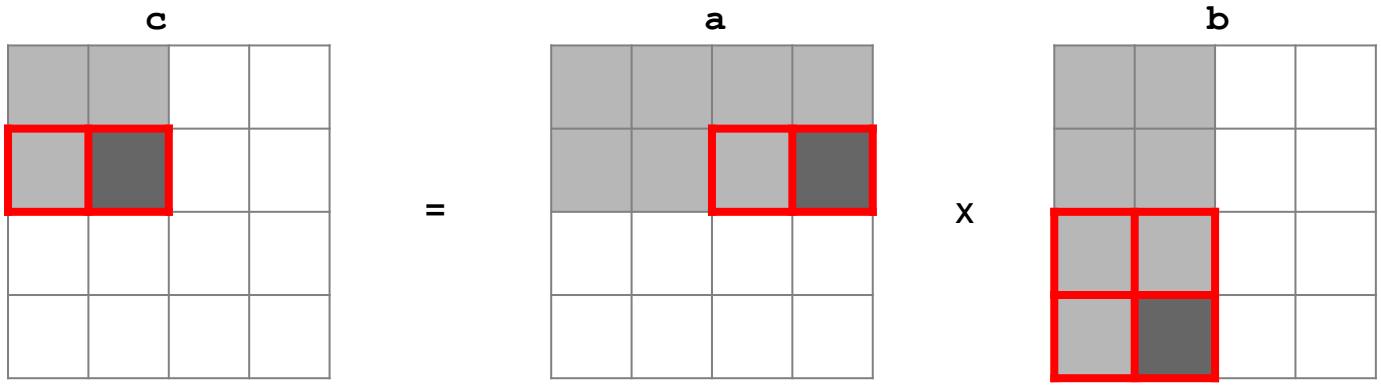
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	(h, h)
14	1	1	2	`c[1][1] += a[1][2] * b[2][0]`	???



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	(h, h)
14	1	1	2	`c[1][1] += a[1][2] * b[2][1]`	(h, h)



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	(h, h)
14	1	1	2	`c[1][1] += a[1][2] * b[2][1]`	(h, h)
15	1	1	3	`c[1][1] += a[1][3] * b[3][1]`	???



Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] * b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] * b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] * b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] * b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] * b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3] * b[3][0]`	(h, h)
14	1	1	2	`c[1][1] += a[1][2] * b[2][1]`	(h, h)
15	1	1	3	`c[1][1] += a[1][3] * b[3][1]`	(h, h)

Blocking: Analyzing Miss Rate

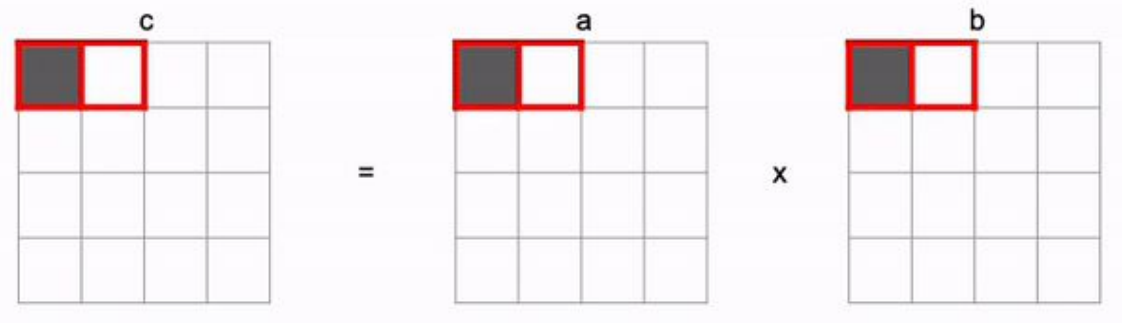
Iteration	*Miss?*
0	(m, m)
1	(h, m)
2	(h, h)
3	(h, h)
4	(m, h)
5	(h, h)
6	(h, h)
7	(h, h)

Iteration	*Miss?*
8	(m, m)
9	(h, m)
10	(h, h)
11	(h, h)
12	(m, h)
13	(h, h)
14	(h, h)
15	(h, h)

- What is the miss rate of **`a`**?
  - ***25%***
- What is the miss rate of **`b`**?
  - ***25%***

Blocking: What Happened?

- Good temporal locality!
- Blocks are re-used while they are still in the cache.



Blocking: What could go wrong?

Blocking with Different Cache

- Great - blocking allows us to better leverage locality! But does this behavior always appear…?
- Let's test it out by apply blocking to a cache that is NOT fully associative!

Blocking Continued

- Suppose our cache is now two-way associative with 2 sets
  - Notice we still have a total of 4 cache lines
- Assuming `C` does not interact with the cache (eg. in a register) and the start of `A,B` point to set 0, we will observe if there is any new behavior.
- Try to pay attention to iterations that leverages locality well versus iterations that are locality's enemy.

Group Activity: B-Accesses

- Let's revisit why we introduced the idea of blocking.
- Ignore all other accesses other than **`B`** and first observe the access pattern and hit/miss pattern for a non-blocking implementation.
  - Where do we see missed opportunities for locality?
- Now do the same analysis but for a blocking implementation. Do you notice anything "better"?

Group Activity: Adding A Back In

- Now we know how blocking works! Let's return to the original problem and introduce accesses to **A** on top of accesses to **B**.
- What do you observe that is different from a fully associative cache?
- Can you identify cases of good locality? What about bad locality?
- Make sure to draw out the cache/matrix states!

Wrapping Up

- **`cachelab`** is due ***Thursday (October 9th)***.
- Take Home midterm is due ***Wednesday (Oct 8th)***
- Make sure to leave time for both.
- Keep an eye out for an email from your Code Review TA!

The End