15-213 Recitation
Caches & Blocking

Your TAs

Friday, October 4th

Reminders

- **`cachelab`** is due ***Thursday (October 10th)***.
- **`malloc`** lab will be released on the same day.
- Written 5 ("Midterm") is due ***Wednesday (October 9th)***
  - Roughly twice the length of a normal written, so plan your time accordingly!
- Drop Deadline: ***Monday (October 7th)***

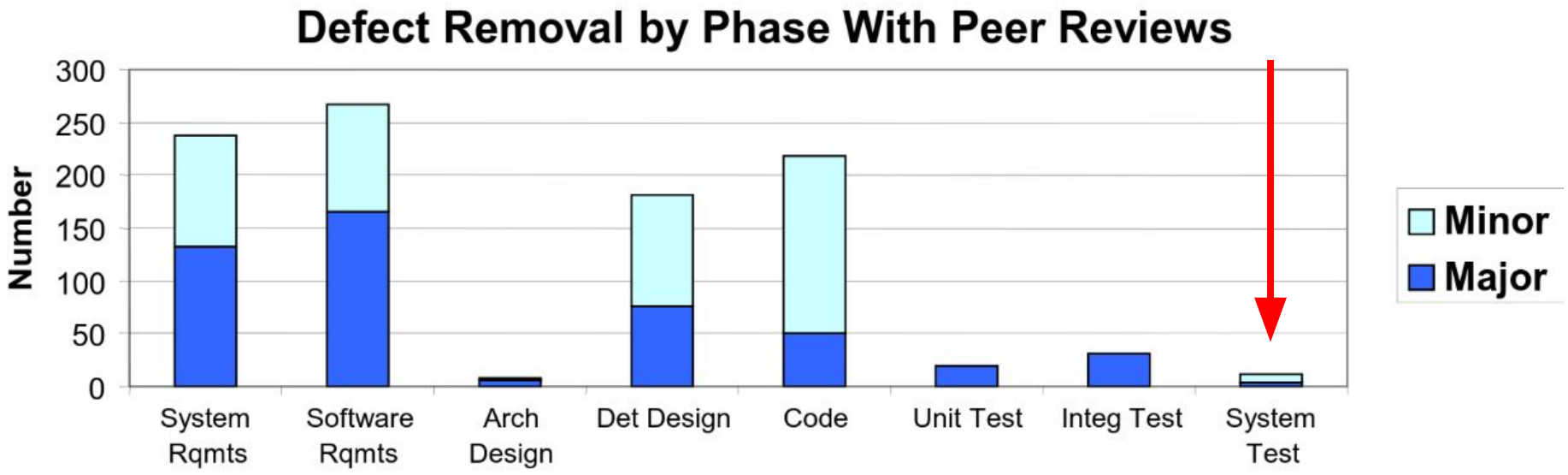
Agenda

- Code Reviews
- Writing Cache-Friendly Code
- Blocking
- Virtual Memory

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)!

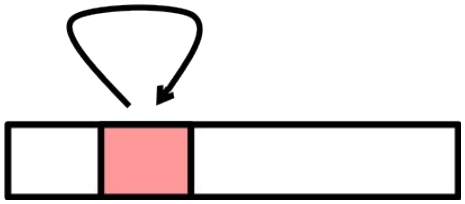
Code Reviews: Guidelines

- 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?

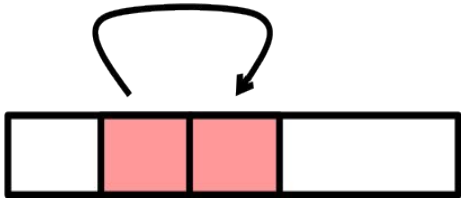
Writing Cache-Friendly Code

Recall: Temporal and Spatial Locality

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

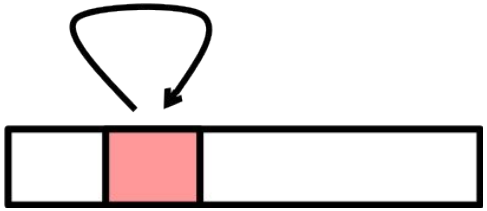


- ***Spatial Locality:***
  - Items with *nearby addresses* tend to be referenced close together in time.



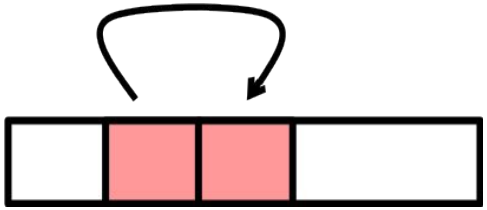
Optimizing for Locality

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



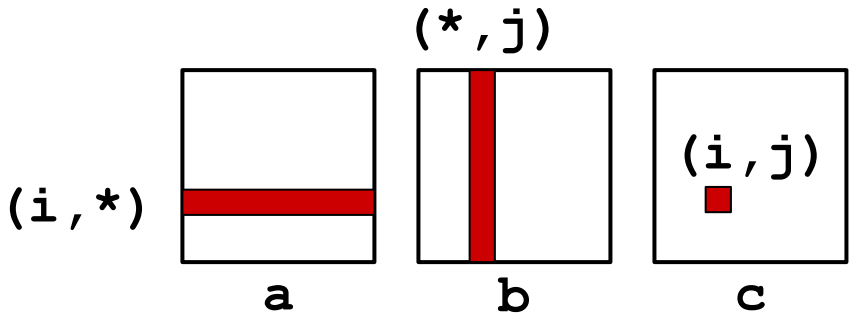
- Items with *nearby addresses* tend to be referenced close together in time.
- To optimize: read objects sequentially, and with smaller stride.
- Lecture Example: *Rearranging loops*.

Blocking

Example: Matrix Multiplication

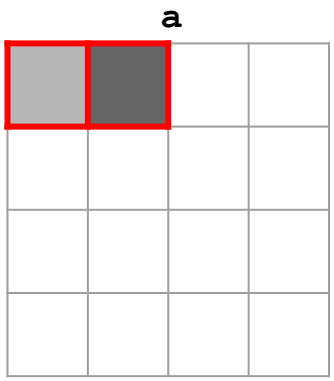
```
/* Multiply 4x4 matrices */
void mm(int a[4][4], int b[4][4], int c[4][4]) {
    int i, j, k;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            for (k = 0; k < 4; k++)
                c[i][j] += a[i][k] * b[k][j];
```

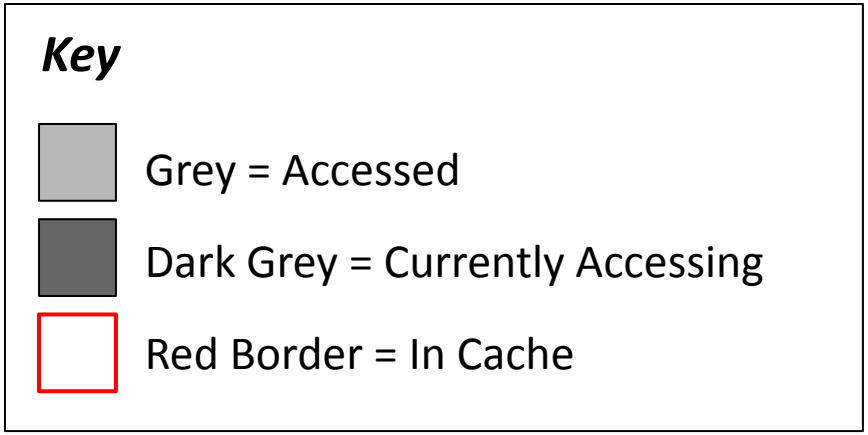
- "Standard" way of doing matrix multiplication (**ijk**):
  - **c[i][j]** is given by taking "dot product" of **i**-th row of **a** with **j**-th column of **b**.



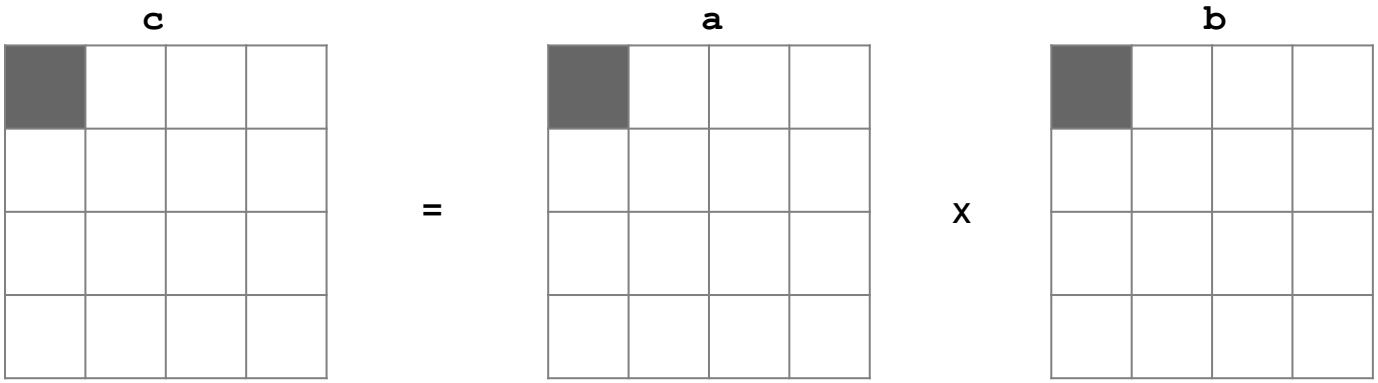
Example: Matrix Multiplication

- 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…

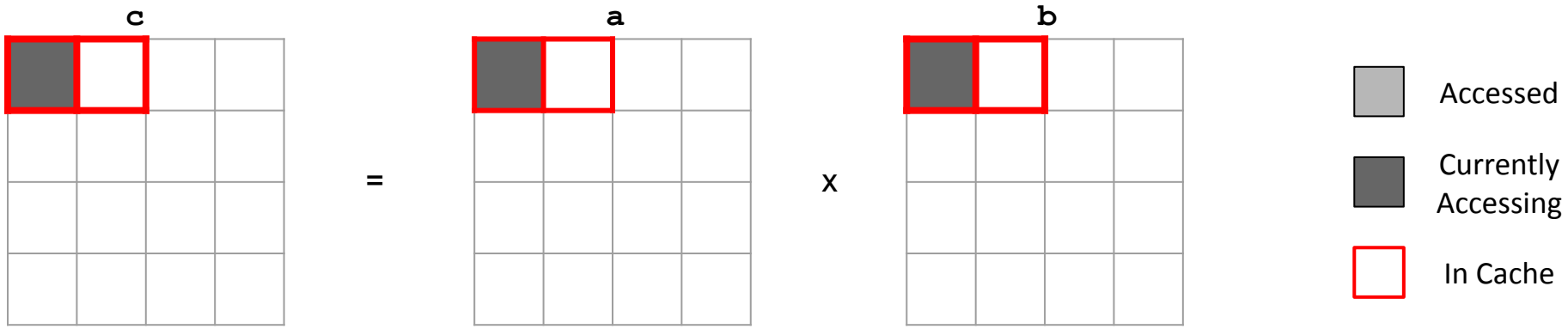


Accessed

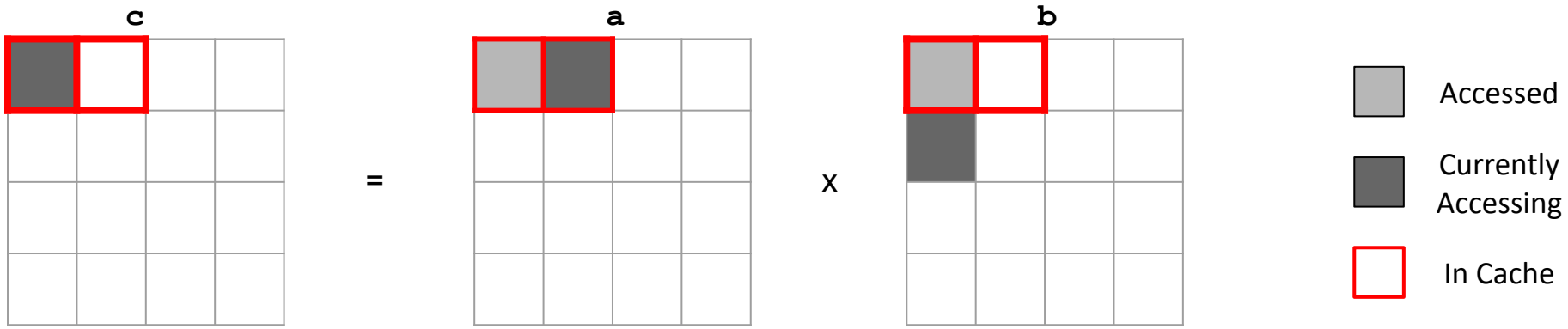
Currently Accessing

In Cache

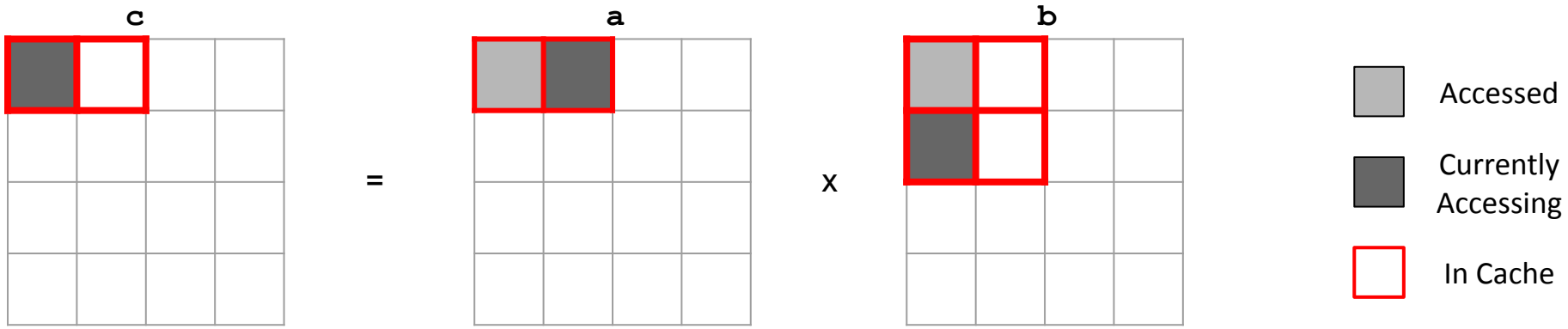
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	???



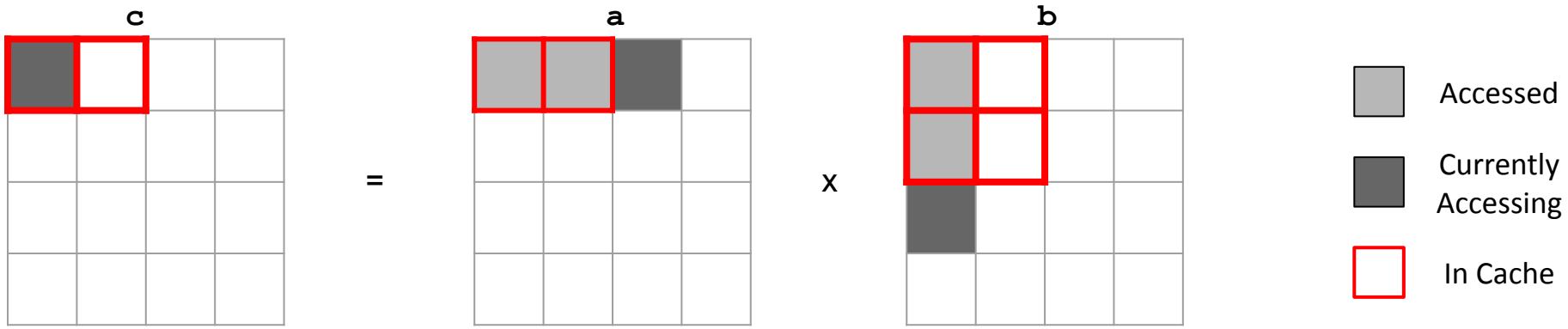
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)



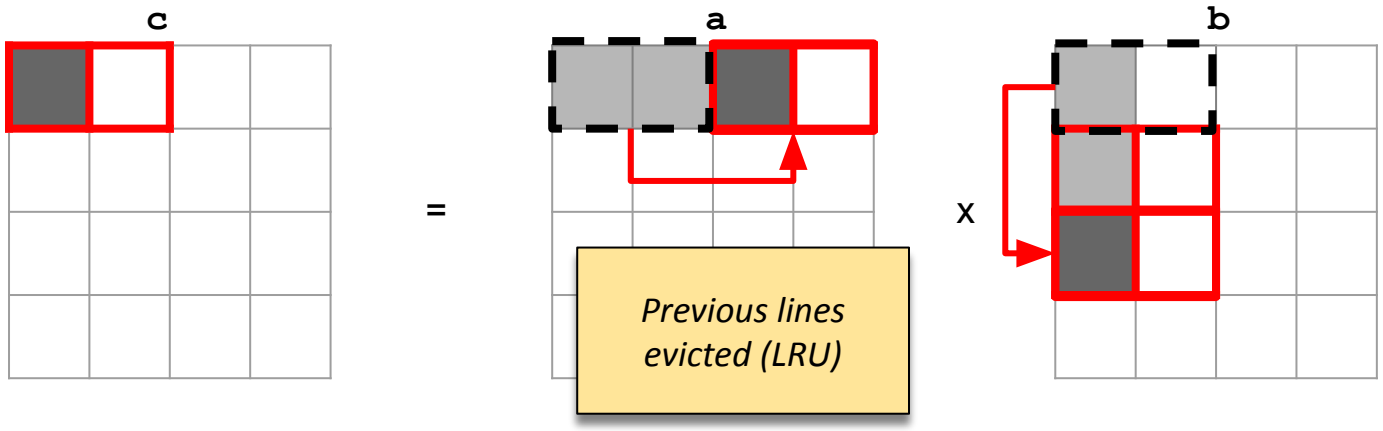
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	???



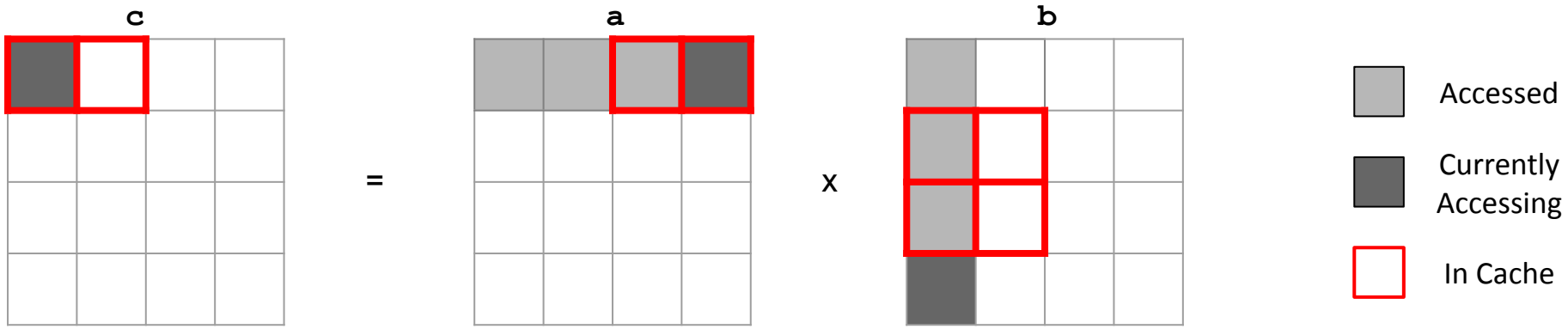
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)



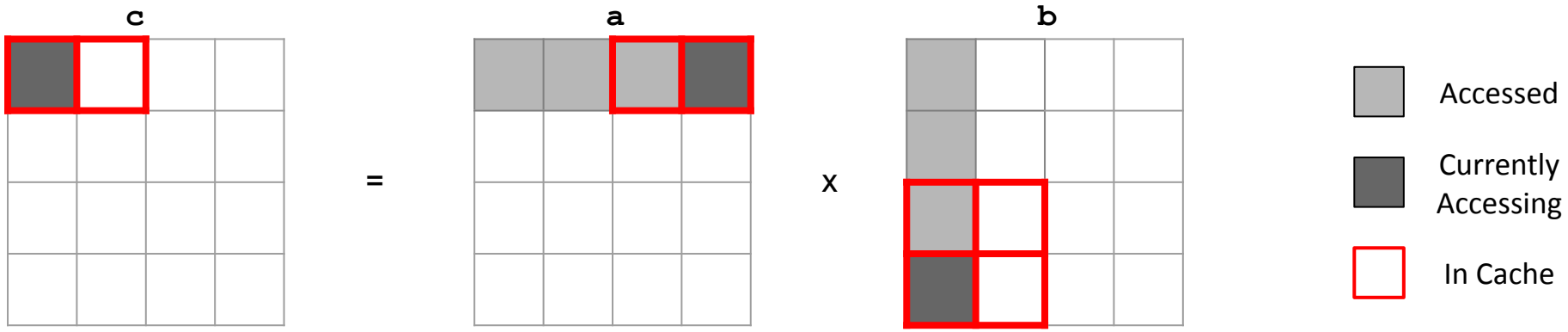
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	???



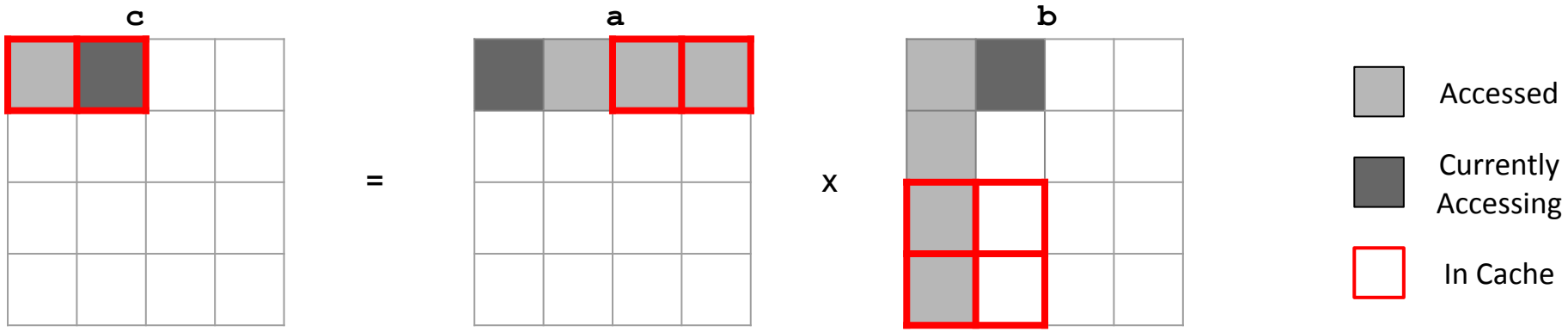
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)



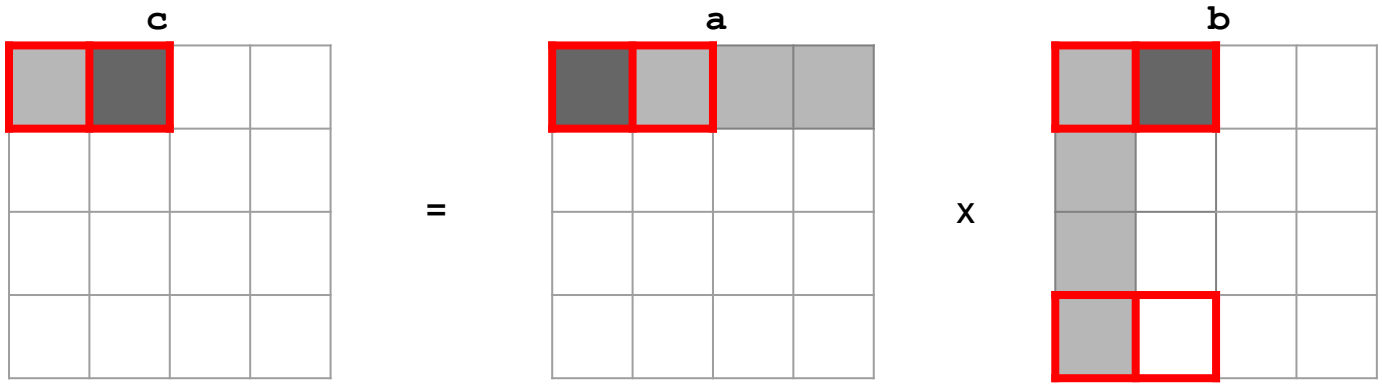
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	???



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)

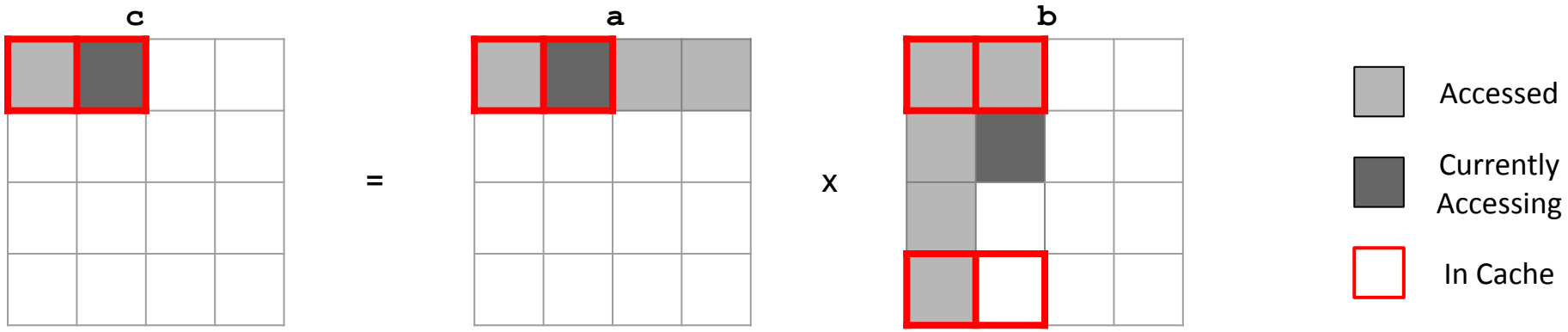


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	???

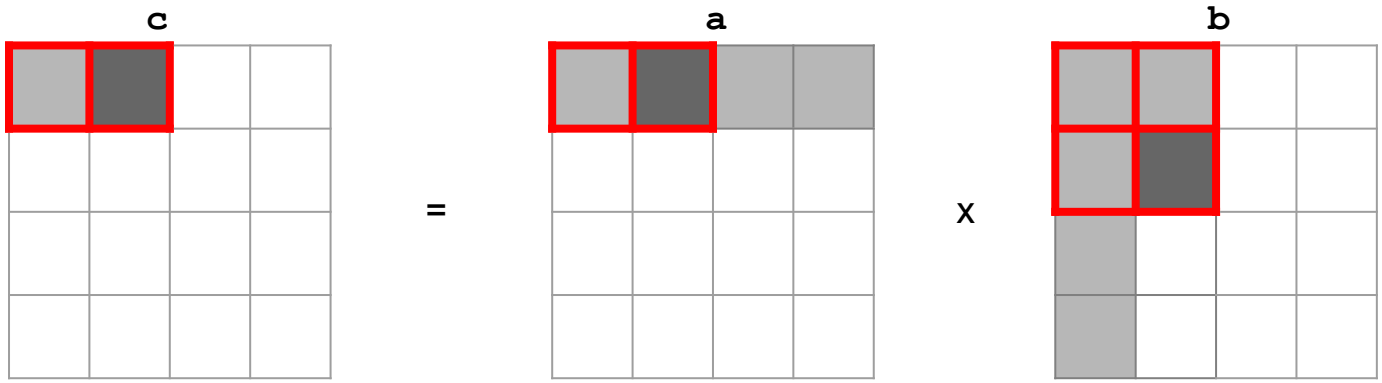


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)

Have these blocks been in the cache before?

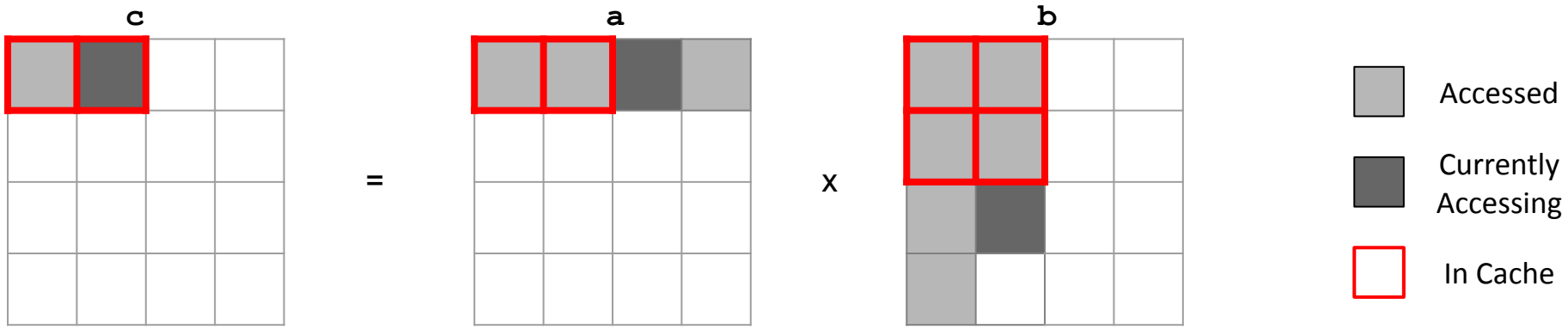


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] +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

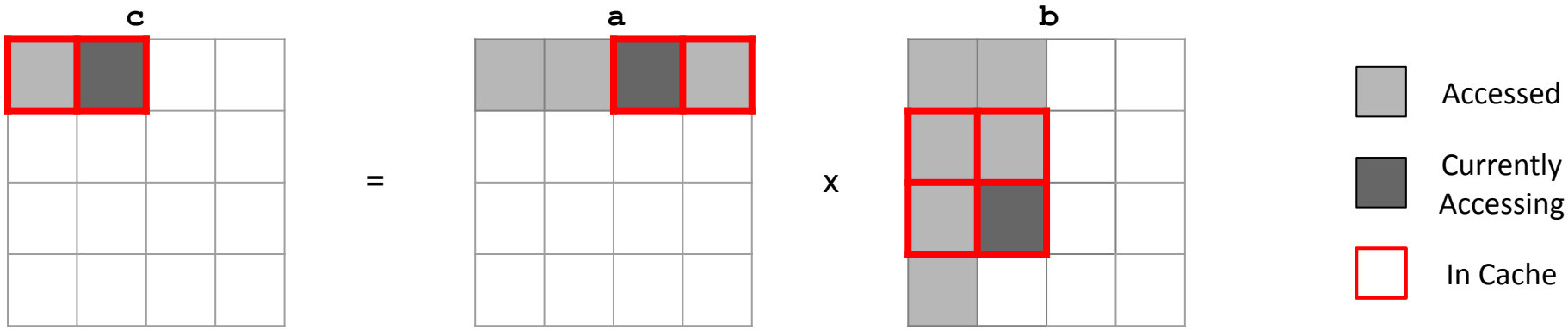


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] +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

Has this block been in cache before?

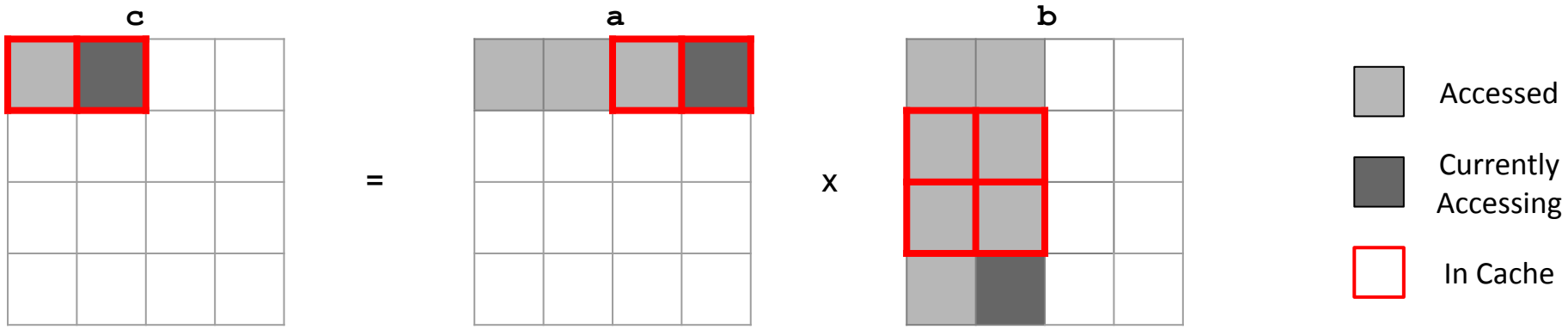


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] +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

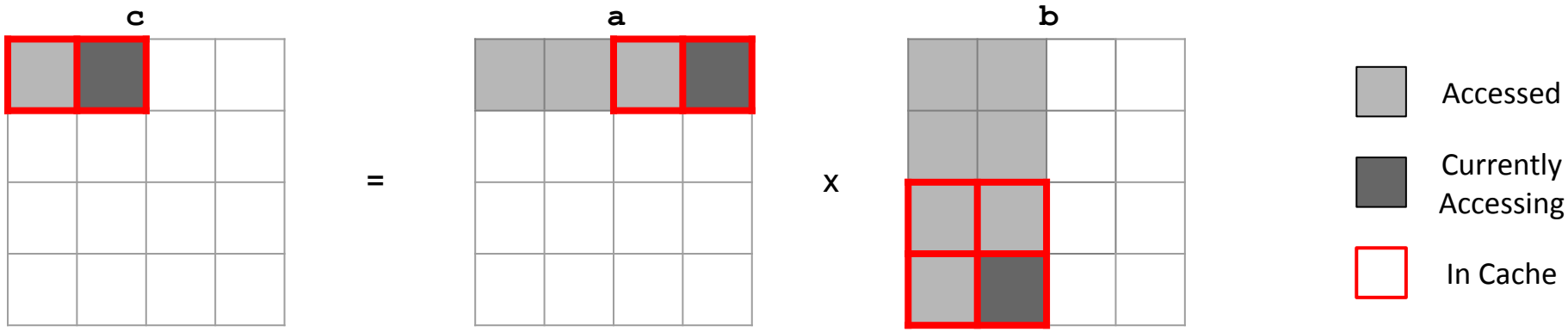


Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
3	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
4	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(m, m)
5	0	1	1	`c[0][1] += a[0][1] +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

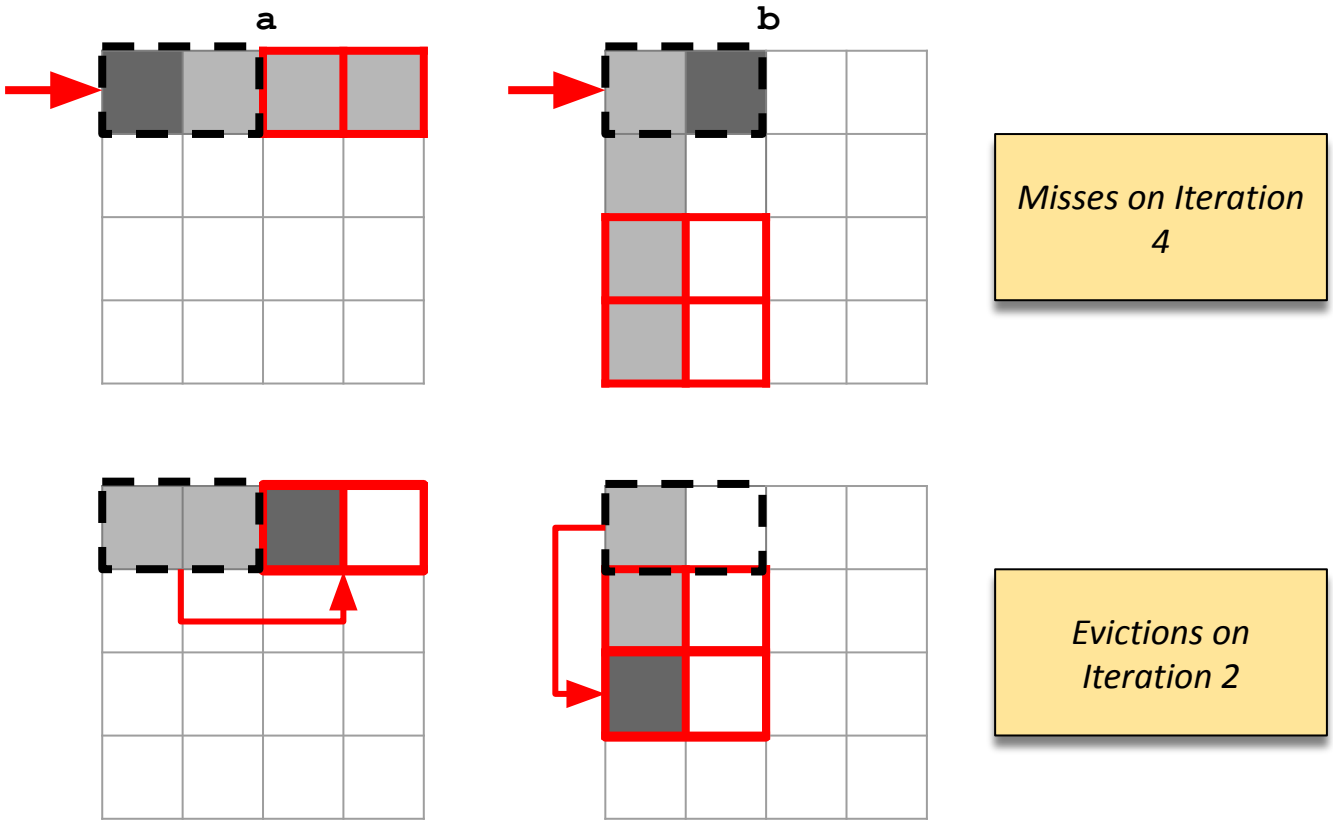
No Blocking: Analyzing Miss Rate

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%***

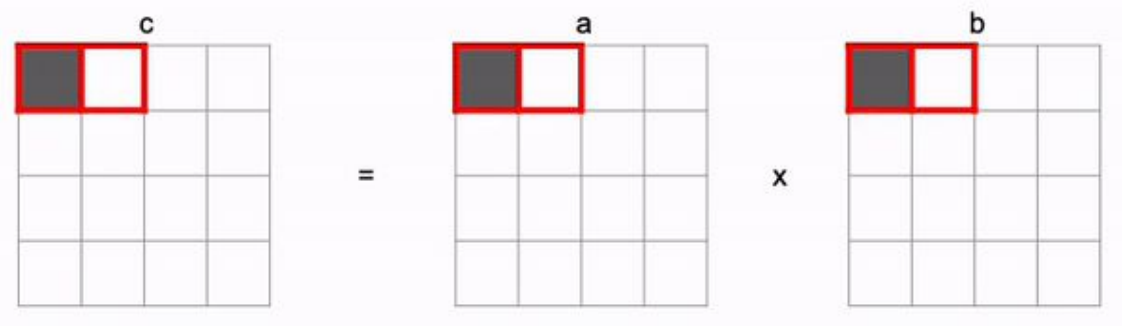
No Blocking: What went Wrong?

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

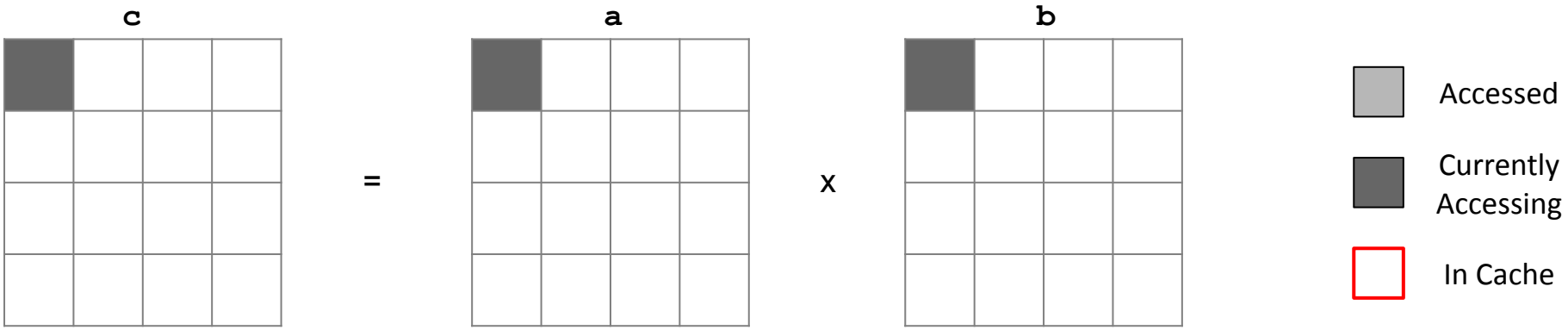


Example: Matrix Multiplication (with Blocking)

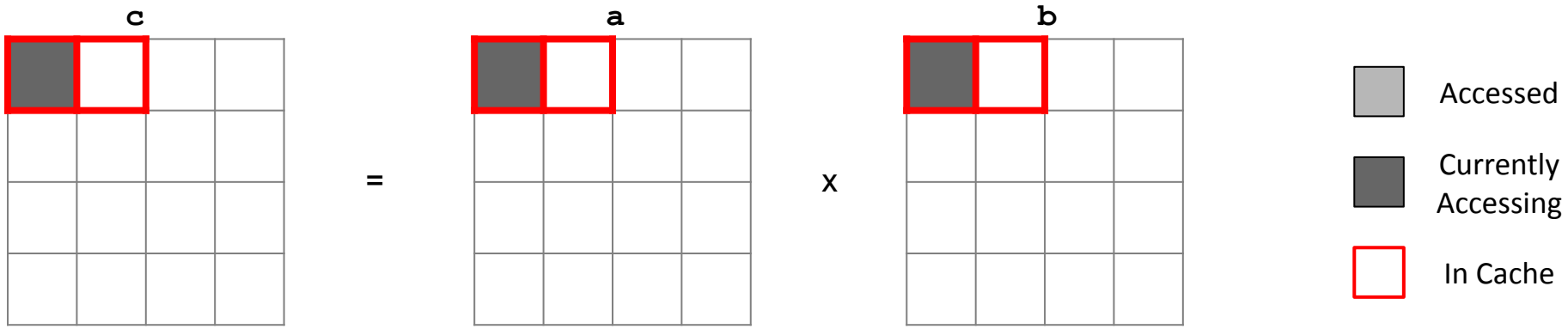
```
/* multiply 4x4 matrices using blocks of size 2 */
void mm_blocking(int a[4][4], int b[4][4], int c[4][4]) {
    int i, j, k;
    int i_c, j_c, k_c;
    int B = 2;
    // control loops
    for (i_c = 0; i_c < 4; i_c += B)
        for (j_c = 0; j_c < 4; j_c += B)
            for (k_c = 0; k_c < 4;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



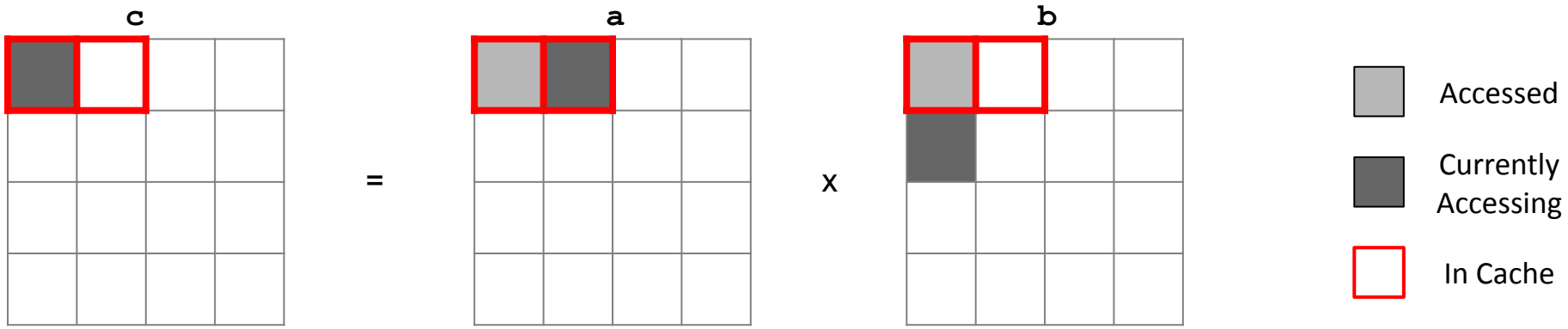
- Let's see what happens if we use blocking!



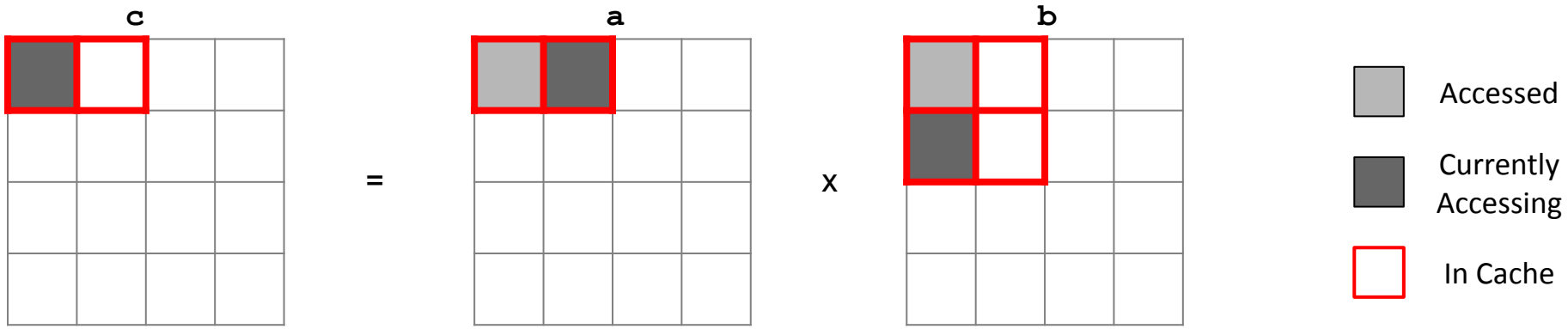
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	???



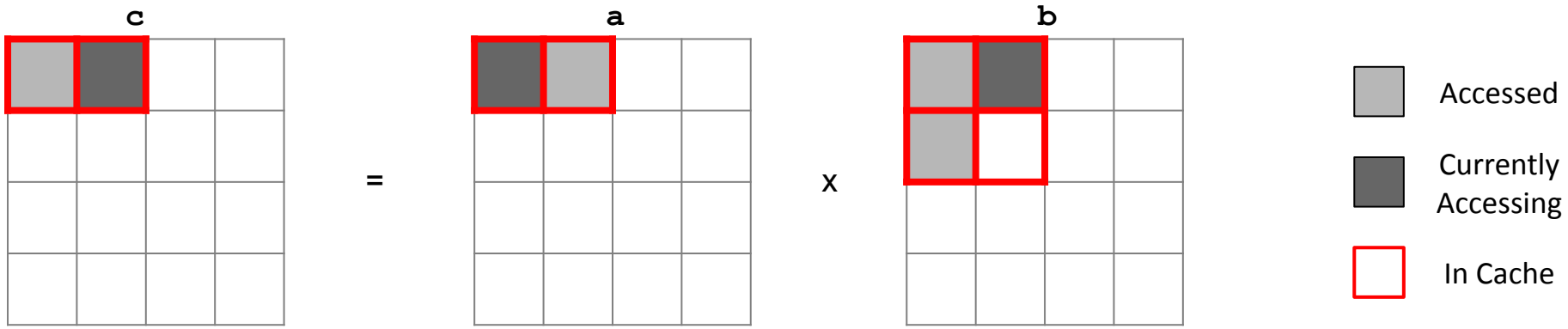
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)



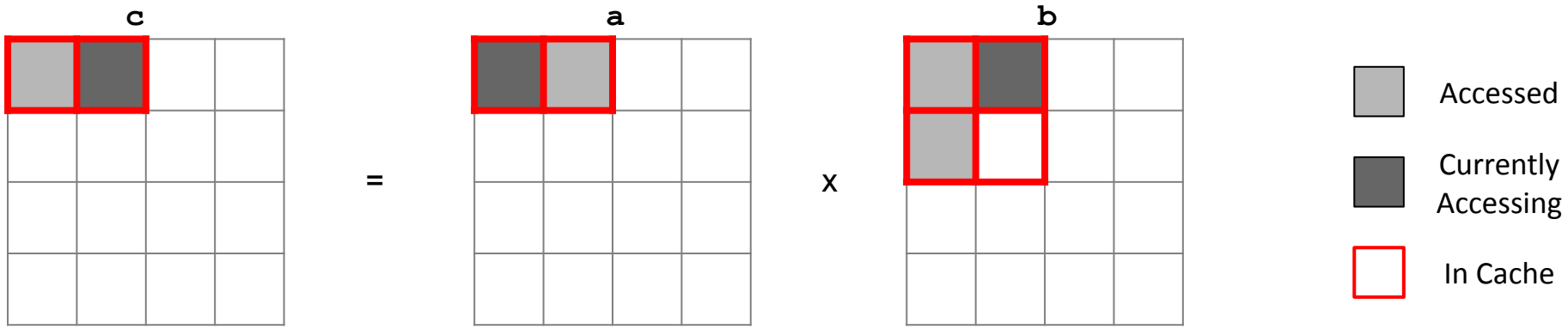
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	???



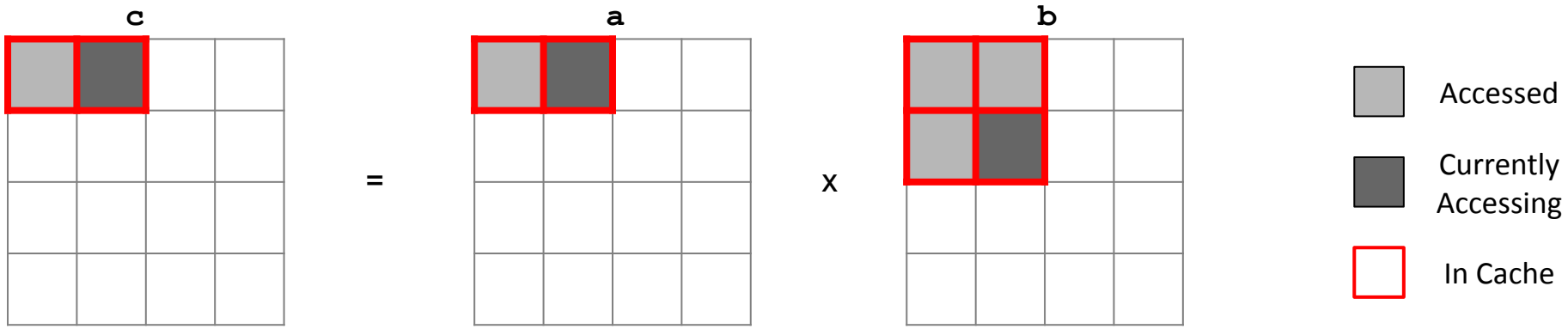
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)



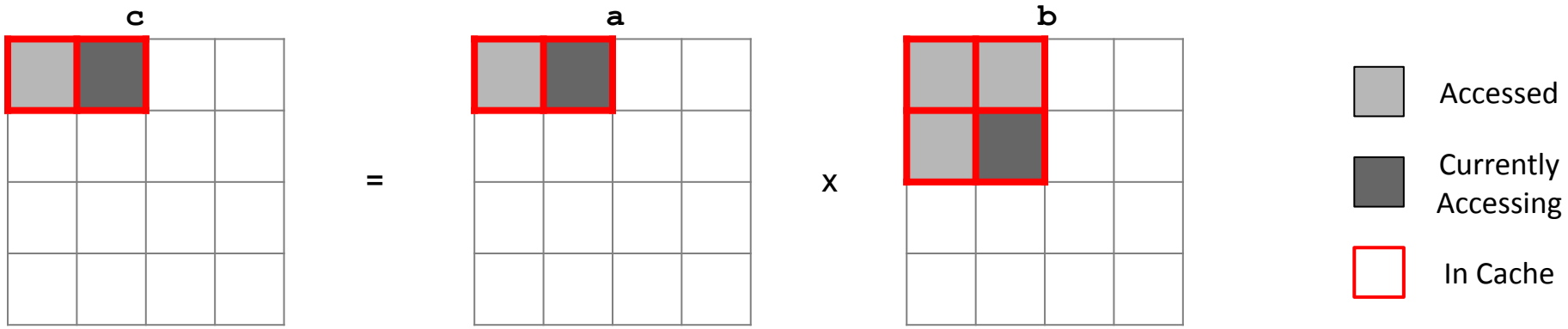
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	???



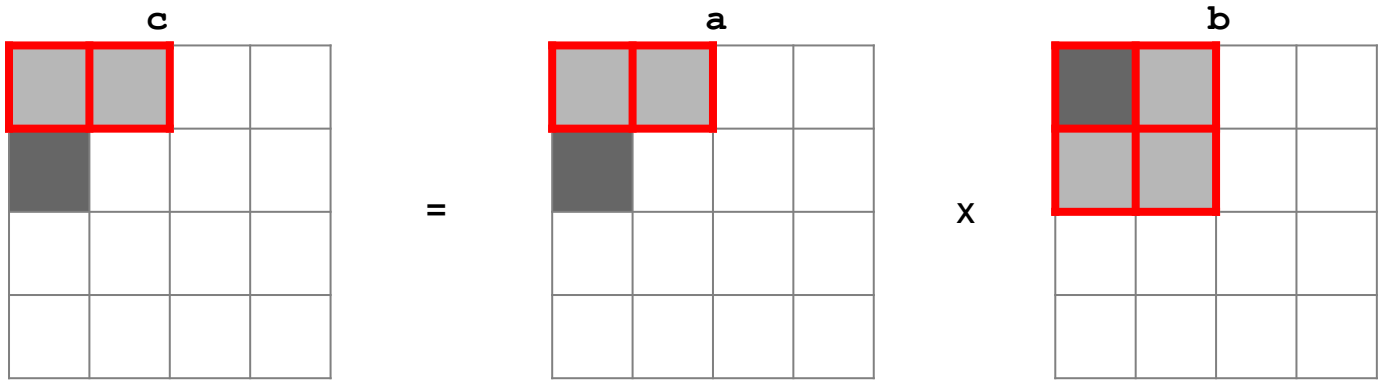
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)



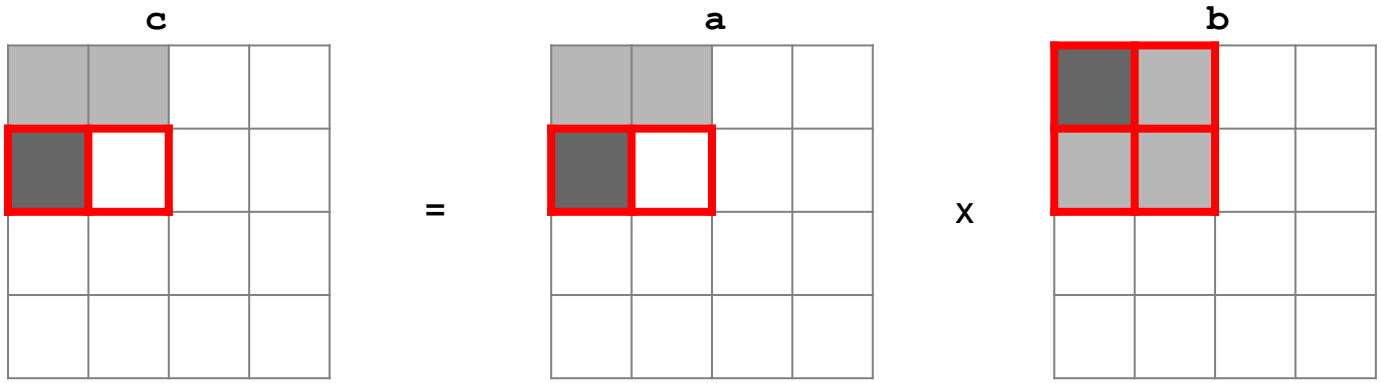
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	???



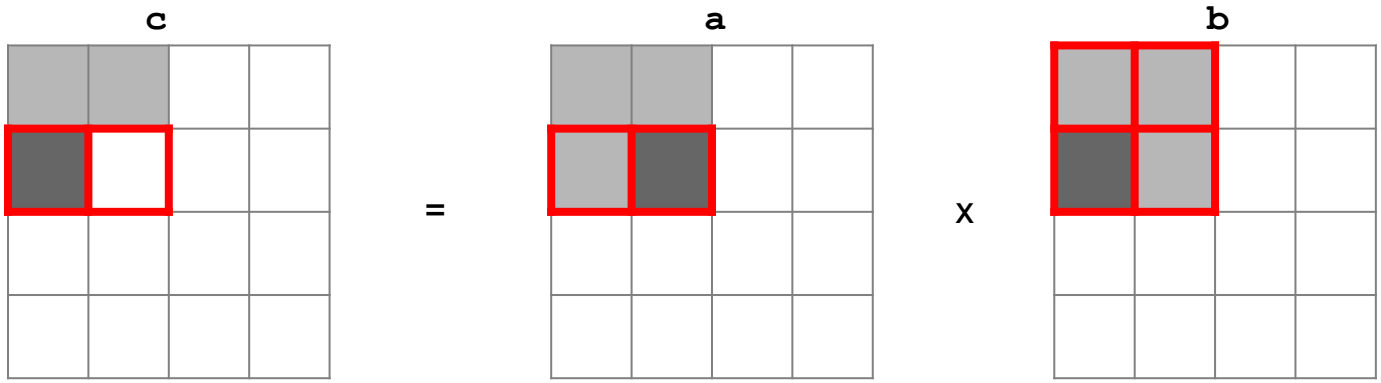
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)



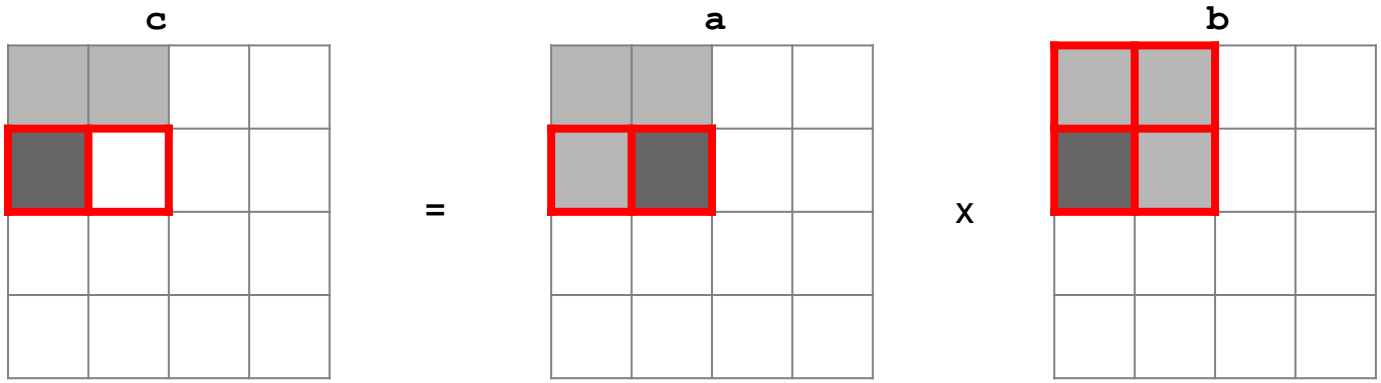
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	???



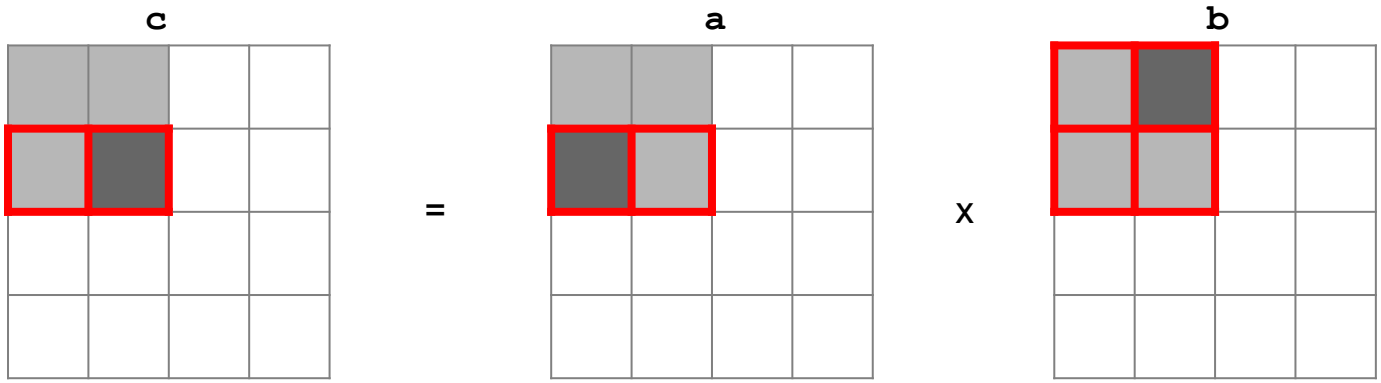
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)



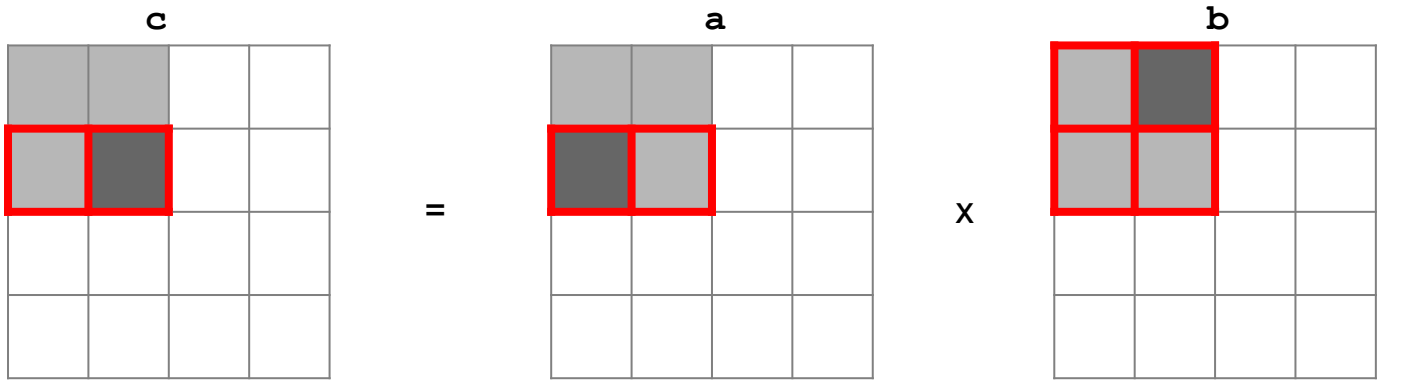
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



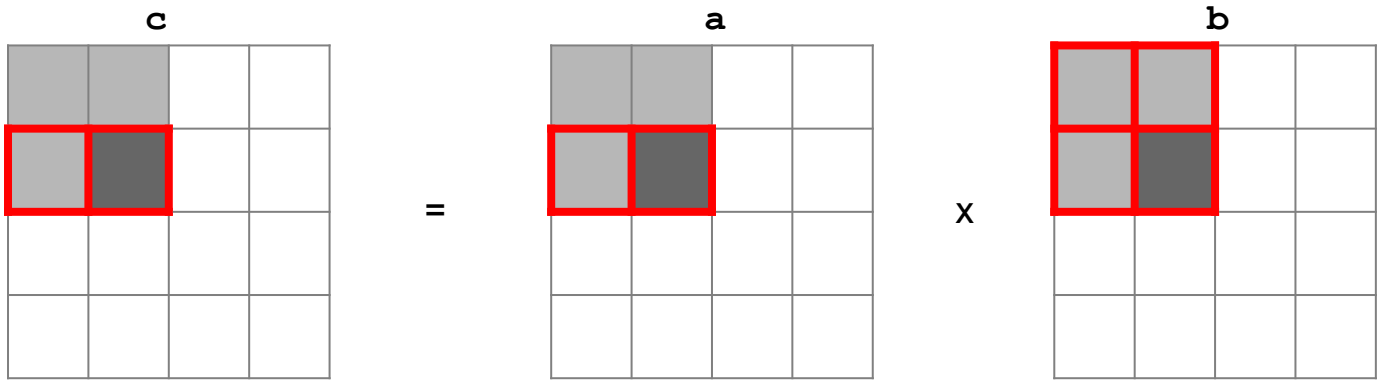
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



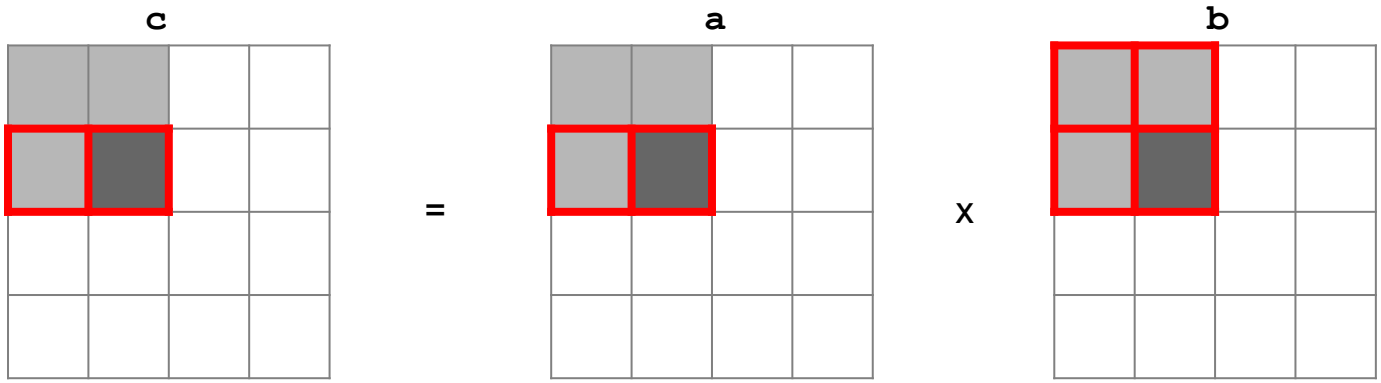
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



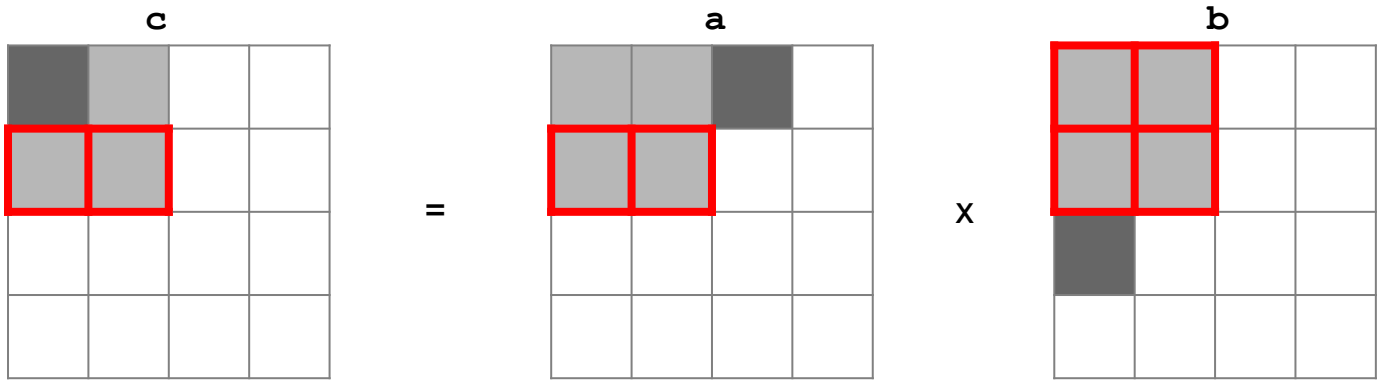
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



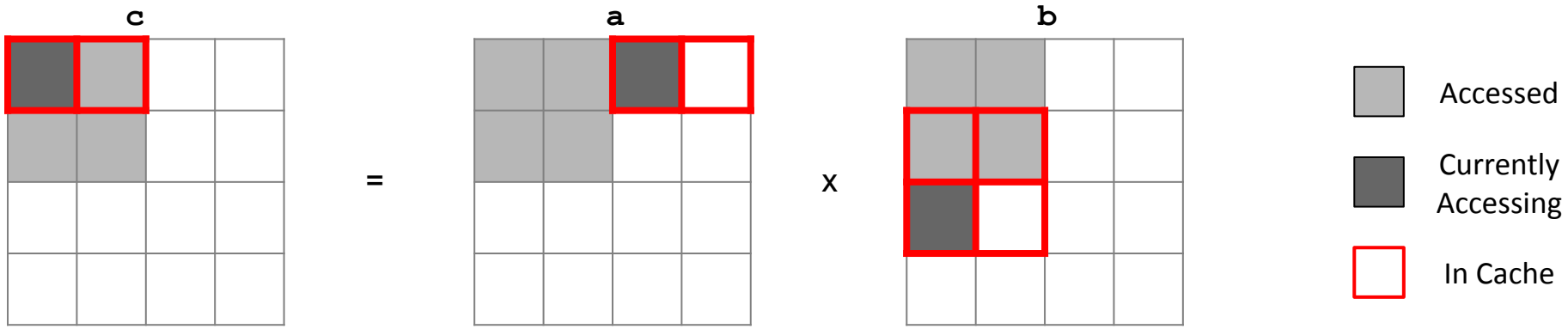
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



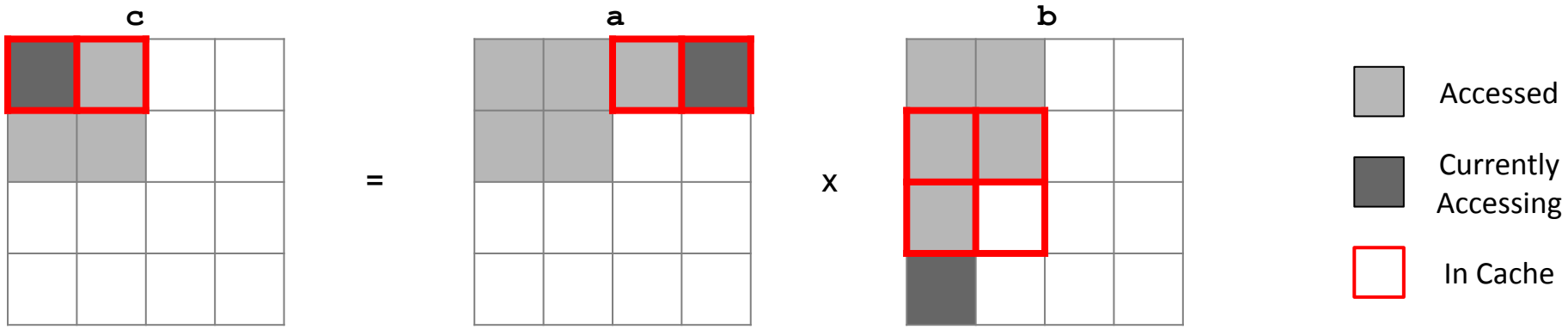
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
0	0	0	0	`c[0][0] += a[0][0] + b[0][0]`	(m, m)
1	0	0	1	`c[0][0] += a[0][1] + b[1][0]`	(h, m)
2	0	1	0	`c[0][1] += a[0][0] + b[0][1]`	(h, h)
3	0	1	1	`c[0][1] += a[0][1] + b[1][1]`	(h, h)
4	1	0	0	`c[1][0] += a[1][0] + b[0][0]`	(m, h)
5	1	0	1	`c[1][0] += a[1][1] +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



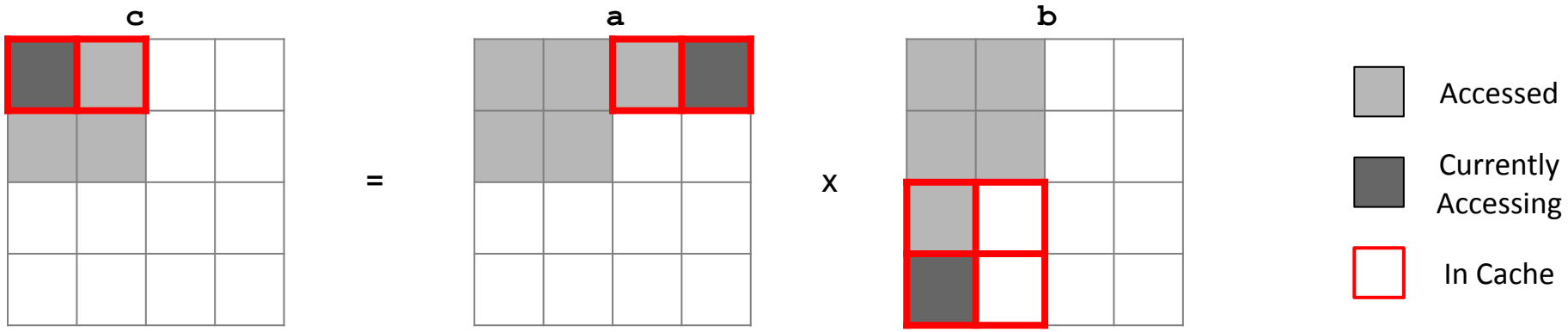
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	???



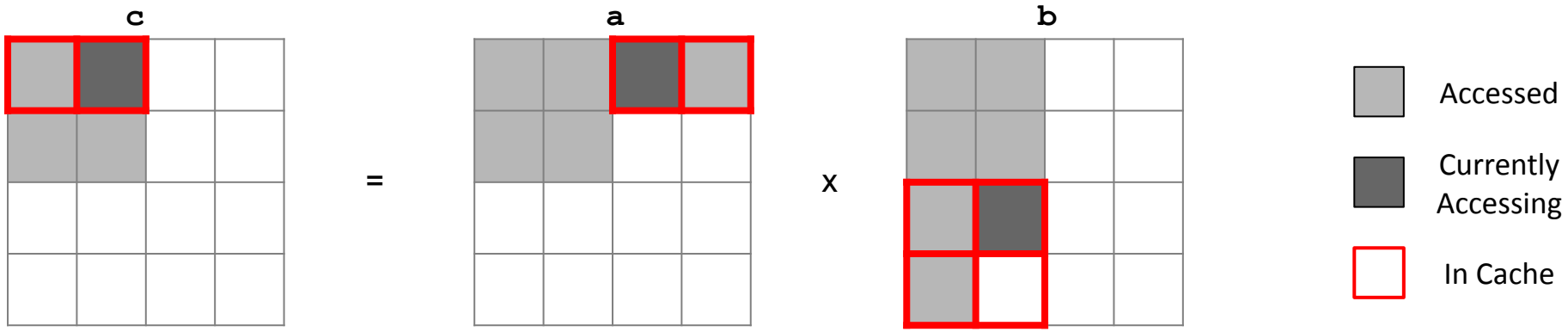
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)



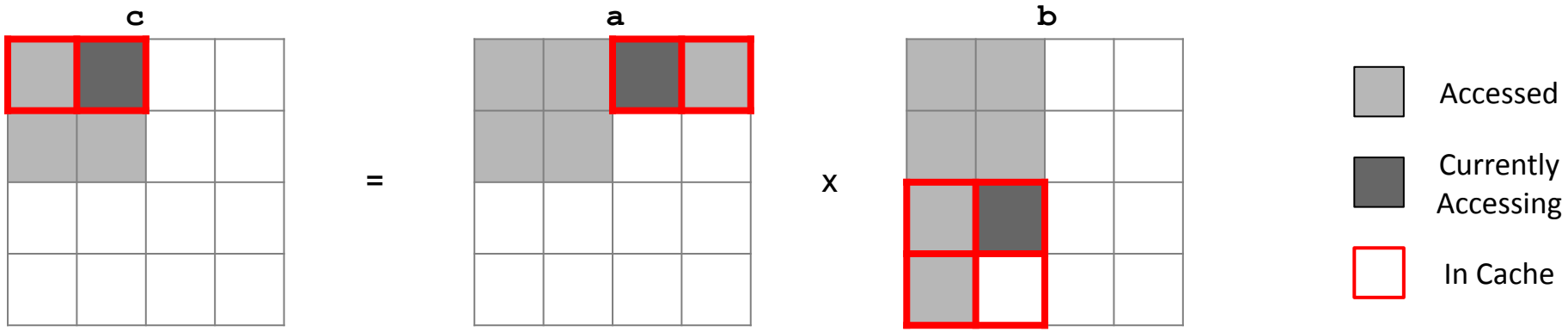
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	???



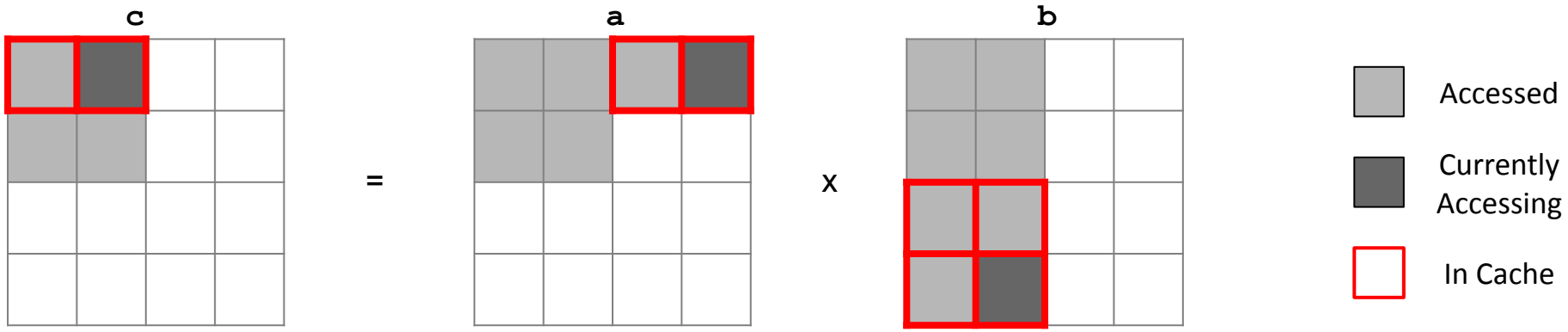
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)



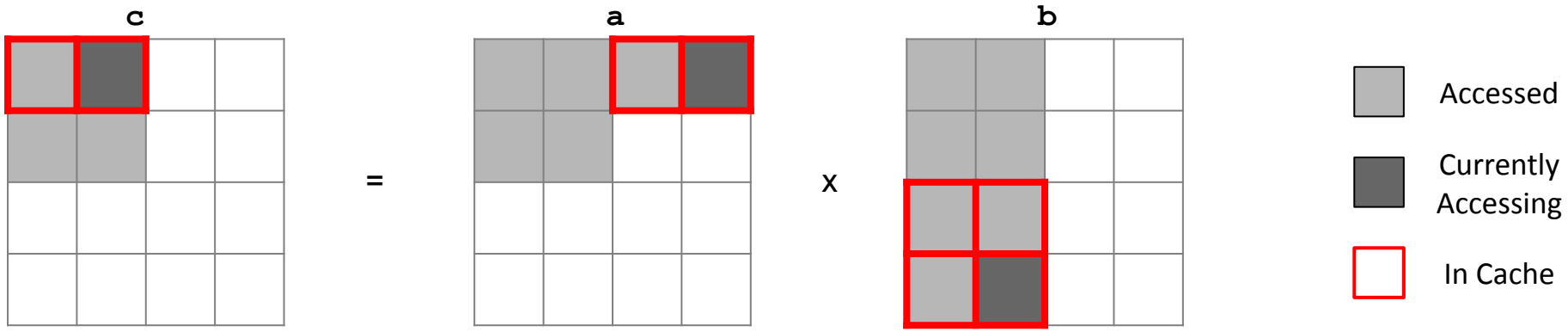
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	???



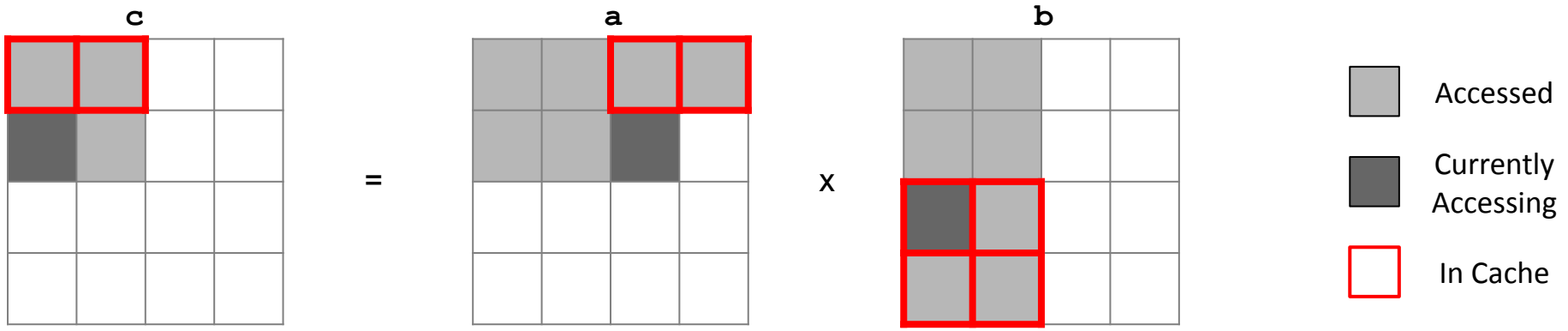
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)



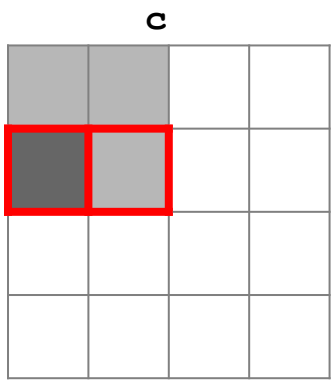
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	???



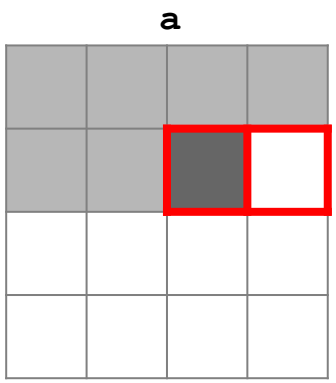
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)



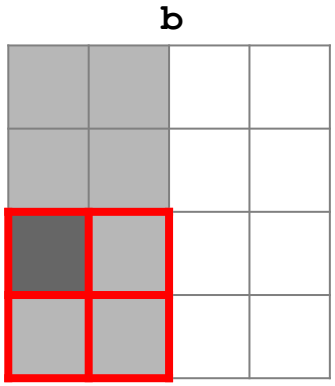
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] + b[2][0]`	???

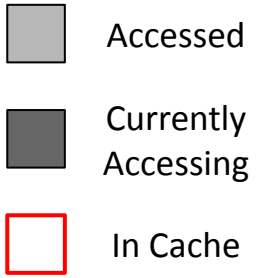


=

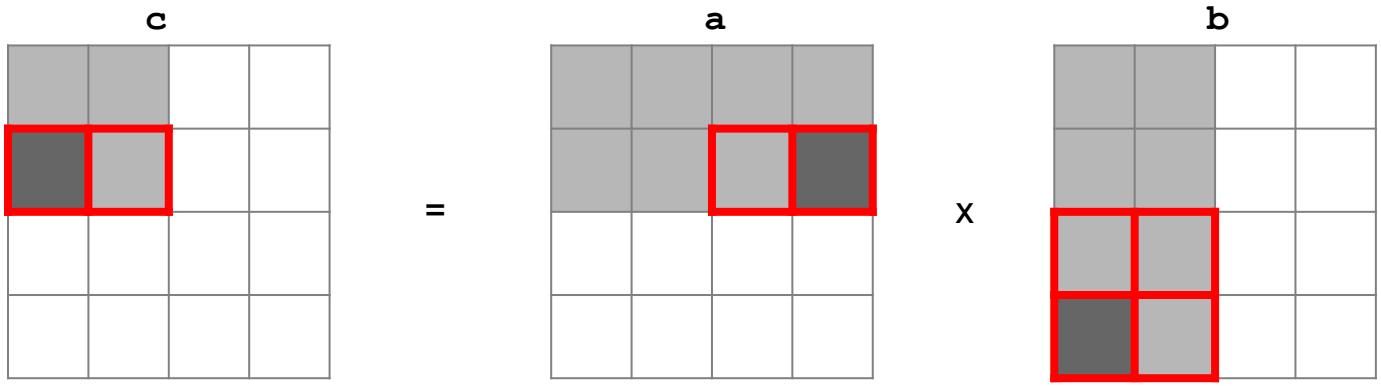


x

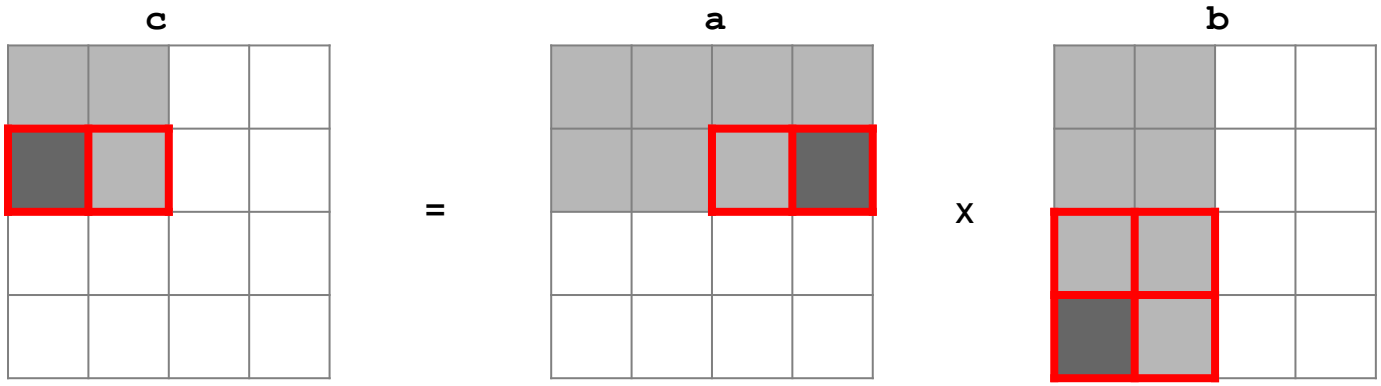




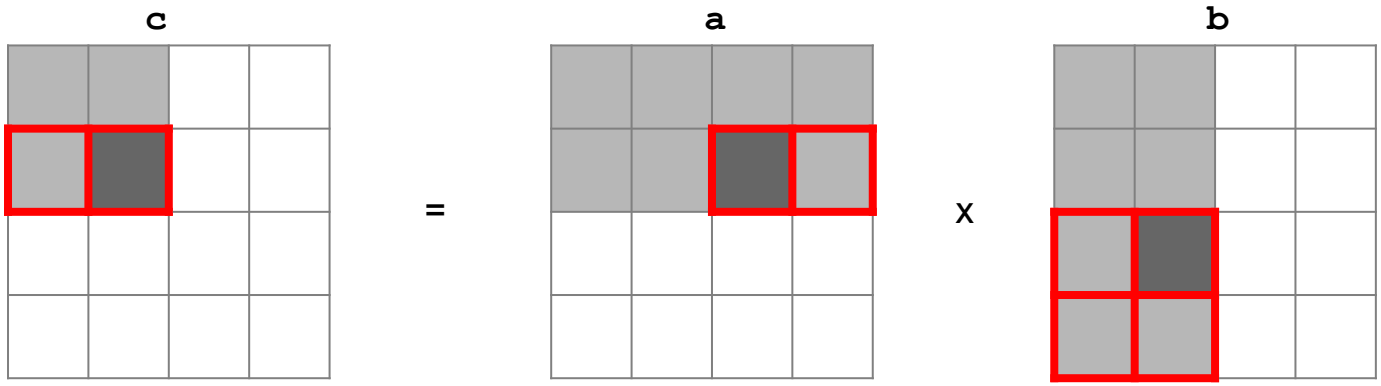
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] + b[2][0]`	(m, h)



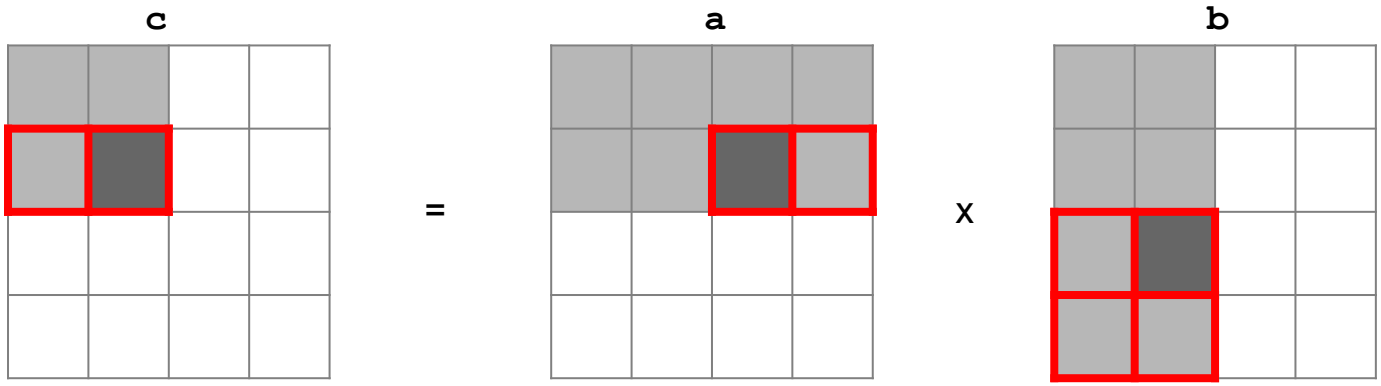
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] + b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3]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



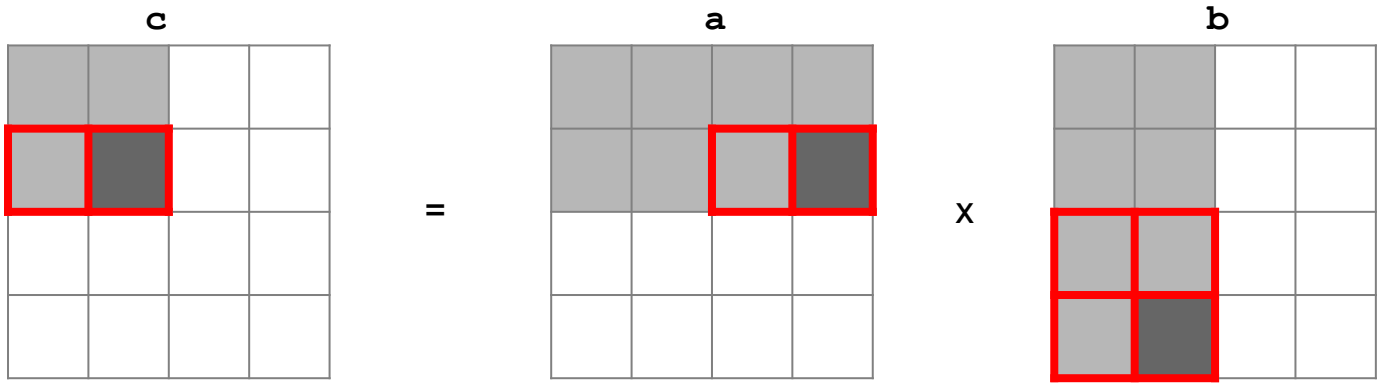
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] + b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3]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



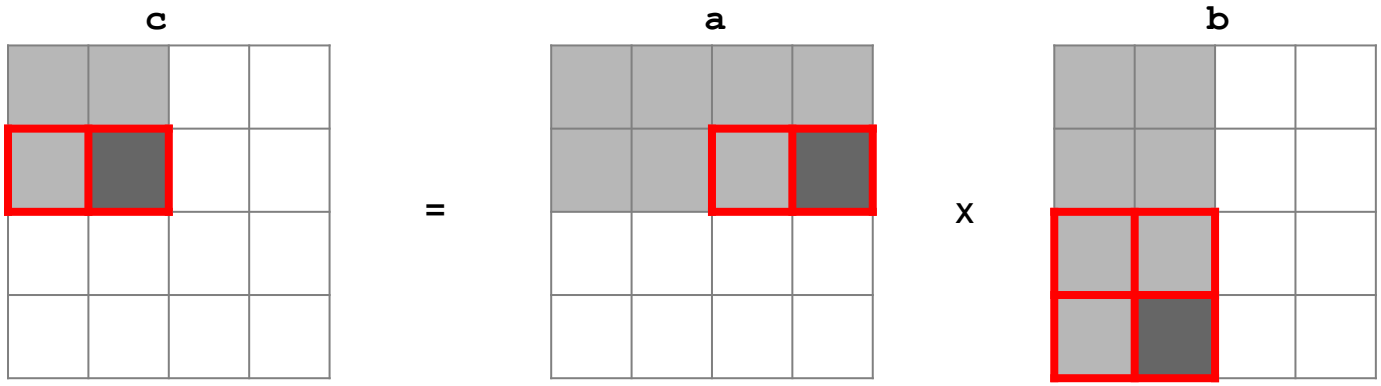
Iteration	`i`	`j`	`k`	*Operation*	*Miss?*
8	0	0	2	`c[0][0] += a[0][2] + b[2][0]`	(m, m)
9	0	0	3	`c[0][0] += a[0][3] + b[3][0]`	(h, m)
10	0	1	2	`c[0][1] += a[0][2] + b[2][1]`	(h, h)
11	0	1	3	`c[0][1] += a[0][3] + b[3][1]`	(h, h)
12	1	0	2	`c[1][0] += a[1][2] + b[2][0]`	(m, h)
13	1	0	3	`c[1][0] += a[1][3]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

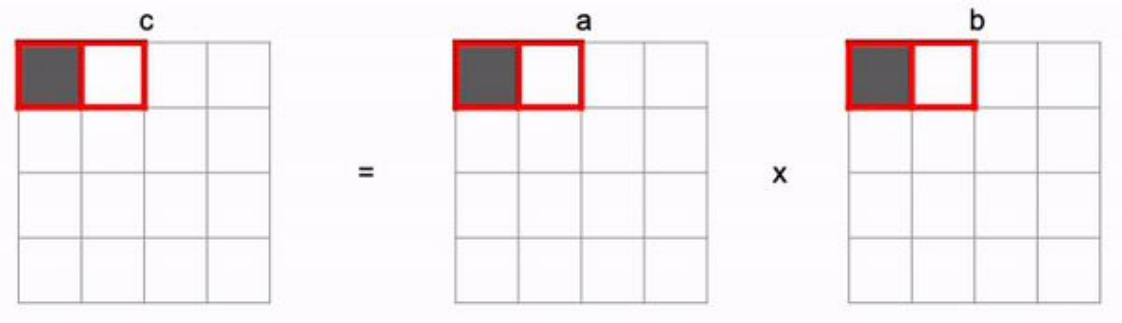
Iteration	*Miss?*
0	(m, m)
1	(h, m)
2	(h, h)
3	(h, h)
4	(m, h)
5	(h, h)
6	(h, h)
7	(h, h)

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?

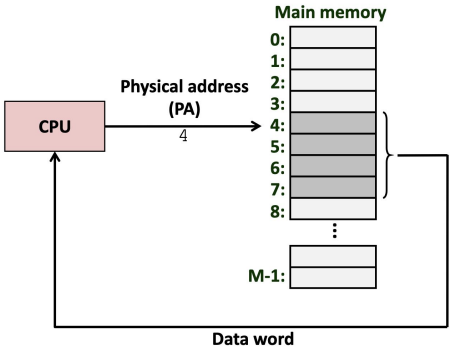
- Good temporal locality!
- Blocks are re-used while they are still in the cache.



Virtual Memory

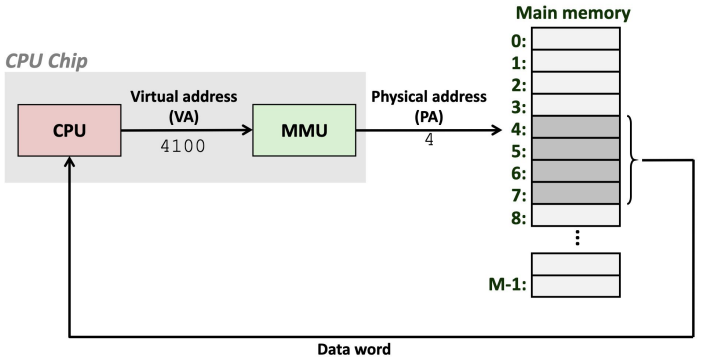
Review: What Is Virtual Memory?

Physical Addressing



Memory address refers to an exact location in memory—only used in simple systems

Virtual Addressing



Memory address refers to a process-specific address, mapped to physical memory via the hardware memory management unit.

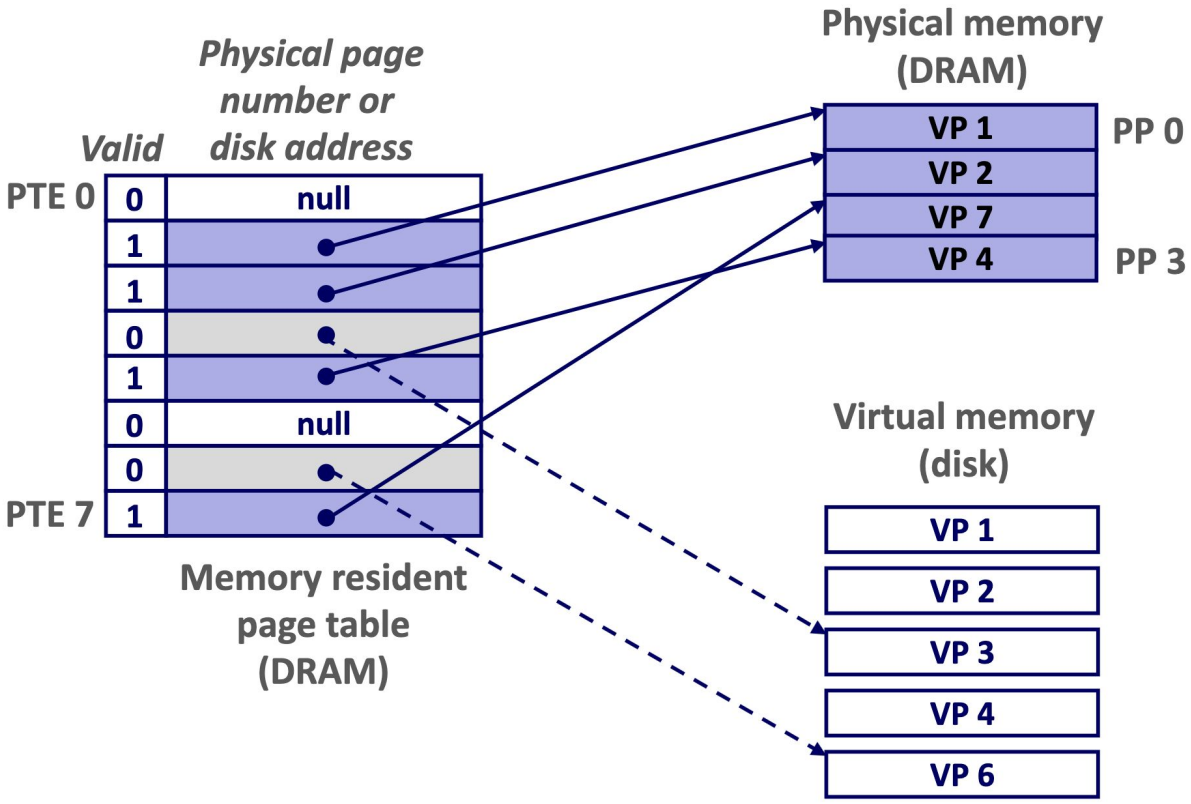
One of the Great Ideas Of Computer Science™

Page Table

Virtual addresses are mapped to physical addresses in the page table. Each entry is called a page table entry.

Pages are in memory, like a cache. If they are not available in memory, we have a page miss.

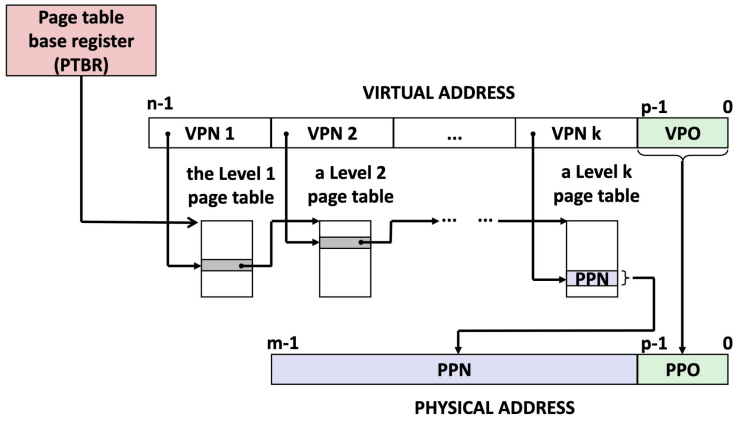
A page miss causes a page fault, which causes the OS to fetch the page from disk and evict a page from DRAM.

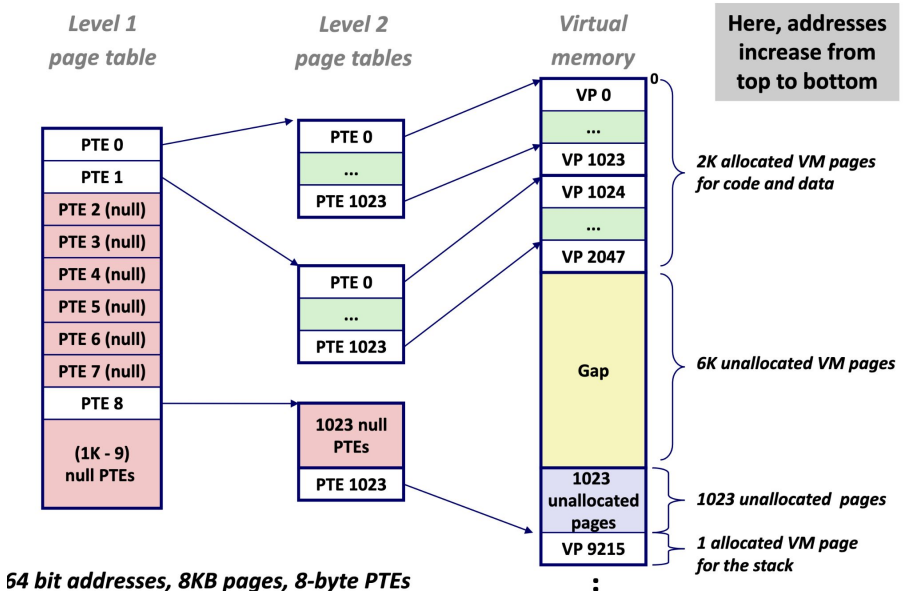


Multi-Level Page Tables

The size of a page table quickly gets out of control when we have to address large addresses space.

The solution is to nest page tables. The VPO/PPO acts as the pseudo-"block offset"





Example - Multi-Level Page Table

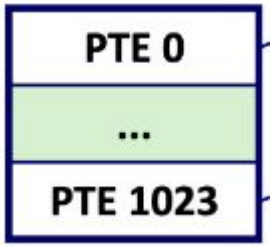
Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

Question of interest : How would we map the virtual address space? Is a single-level page table enough? Do we need more levels? Let's dive into it….

Simpler question : How many PTEs (page table entries) fit inside a single page?

# of PTEs in a page = size of a page / size of a PTE

# of PTEs in a page = size of a page / size of a PTE = 4KB/4B = $2^{12}/2^{2} = 2^{10}$ = 1024



Example - Multi-Level Page Table

Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

Question of interest : How would we map the virtual address space? Is a single-level page table enough? Do we need more levels? Let's dive into it….

Simpler question : How many bits in the virtual/physical address for page offset?

VPO = PPO = $\log_2$(page size)

VPO = PPO = $\log_2(2^{12})$ = 12 bits

20 bits	12 bits
to be discussed in later slides	offset (VPO = PPO)

Example - Multi-Level Page Table

Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

Question of interest : How would we map the virtual address space? Is a single-level page table enough? Do we need more levels? Let's dive into it….

Simpler question : How many PTEs required for mapping the entire VA space?

# of PTEs for VA space = size of VA space/size of a page

# of PTEs for VA space = size of VA space/size of a page = $2^{32}/2^{12} = 2^{20}$

Example - Multi-Level Page Table

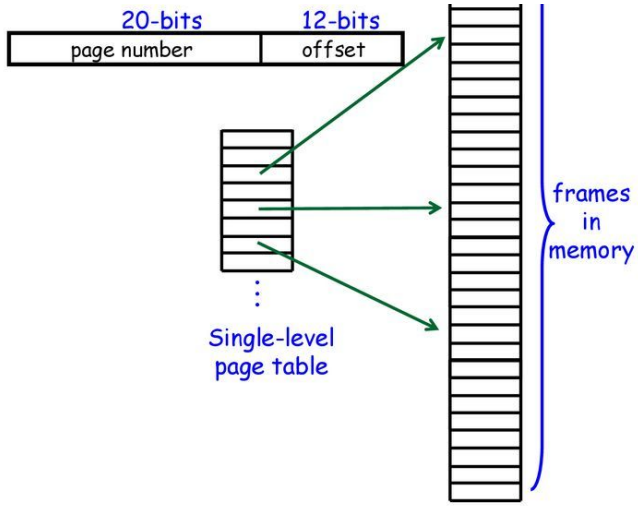
Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

Question of interest : How would we map the virtual address space? Is a single-level page table enough? Do we need more levels? Let's dive into it….

Simpler question : How many pages for a single-level page table?

# of pages for VA space = # of PTEs to map VA space/# of PTEs in a page

# of pages for VA space = $2^{20}/2^{10} = 2^{10}$



Example - Multi-Level Page Table

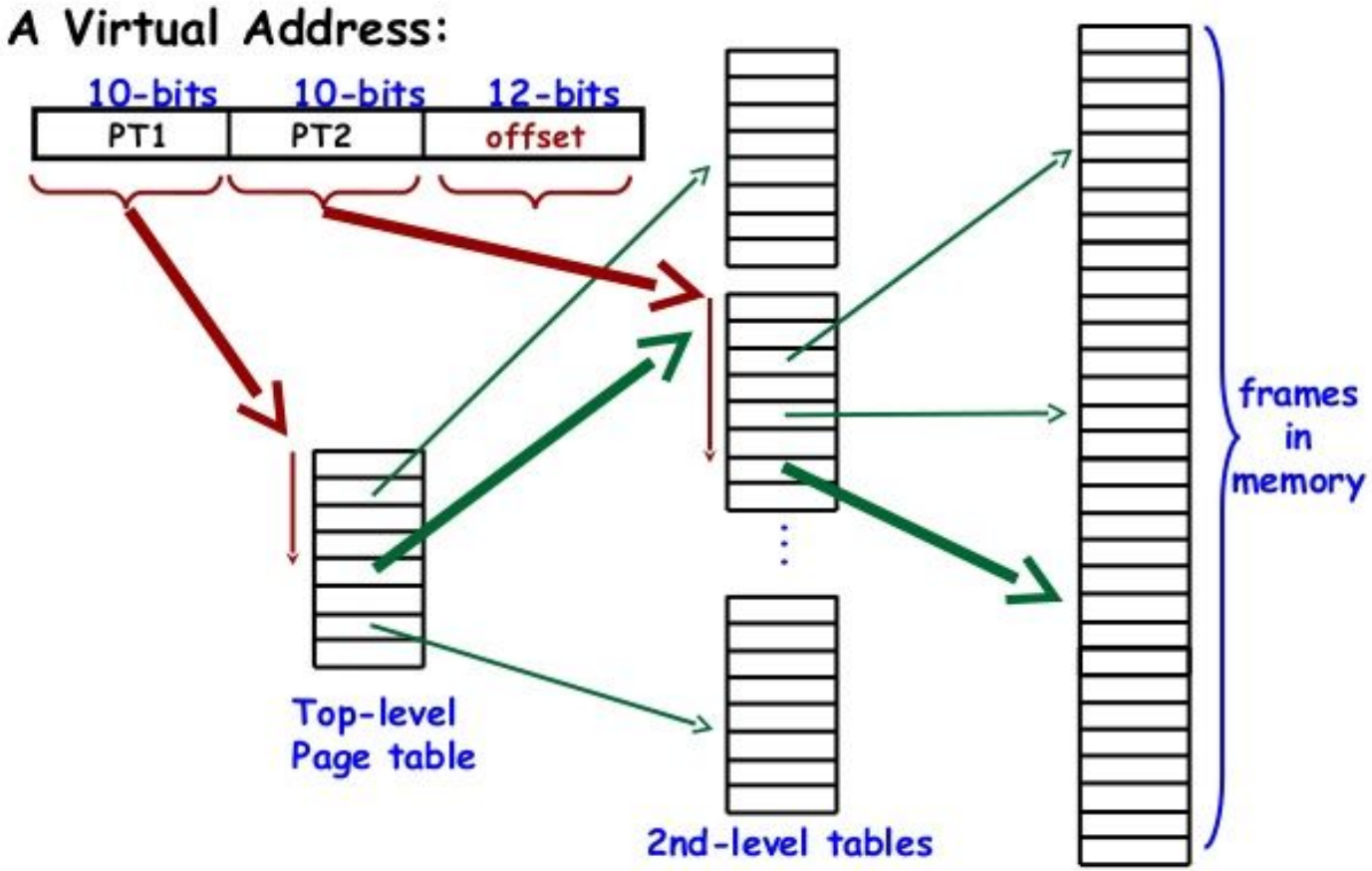
Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

Question of interest : How would we map the virtual address space? Is a single-level page table enough? Do we need more levels? Let's dive into it….

Now that we know that we need 2^{10} pages for the page table, can we add another level?

YES!
We can have one page at the outer level (because a page can hold 1024 PTEs, which is how many pages we need for the inner level)

Example - Multi-Level Page Table



Advantage of multi-level page tables

Consider a system with 32 bit virtual address space and a 24 bit physical address space. Page Size is 4KB. Assume the size of entries in the Page Table is 4 bytes.

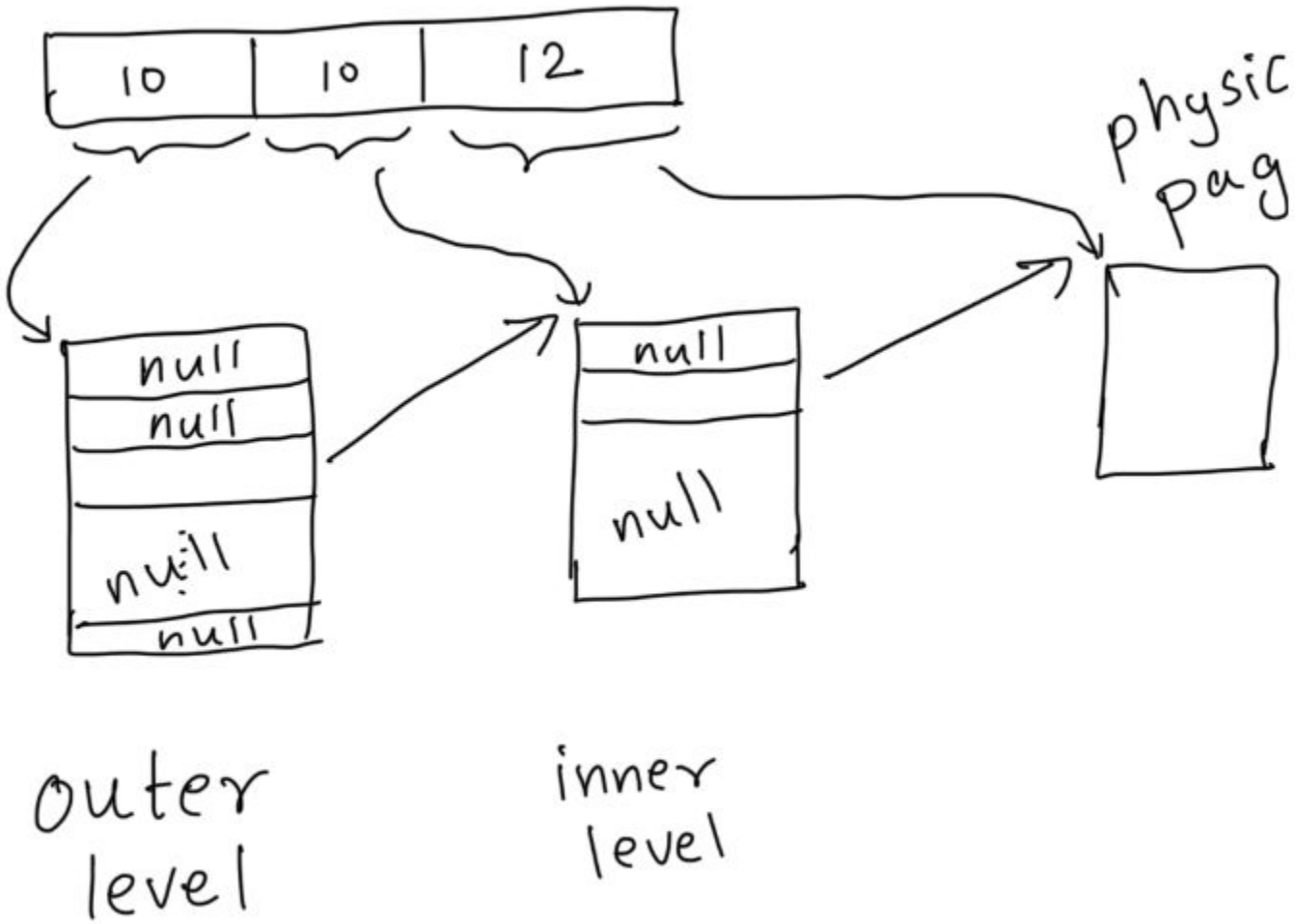
If we think about a single memory access (assuming the page table was empty at the start), how many pages would be require?

We know that we need one page for the outer level.

Since we know it's a single memory access, we only need one valid PTE at the outer level. This also implies we need one page at the inner level

Therefore, we only need 2 pages, saving a huge chunk of space.

Advantage of multi-level page tables



Wrapping Up

- **`cachelab`** is due ***Thursday (October 10th)***
- Written 5 ("Midterm") is due ***Wednesday (October 9th)***
  - Twice the length of a regular written!
- Make sure to leave time for both.
- Keep an eye out for an email from your Code Review TA!

The End