Memory Hierarchy (4): Cache Misses and How to Address Them (cont.)

Hung-Wei Tseng

Outline

- Writing cache-friendly code
- Virtual memory
- Sample Midterm

How can programmer improve memory performance? (cont.)

Data structures

Loop optimizations

Loop interchange

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Loop fission

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

Loop fusion

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity

Tiling/Blocking Algorithm

What is an M by N "2-D" array in C?

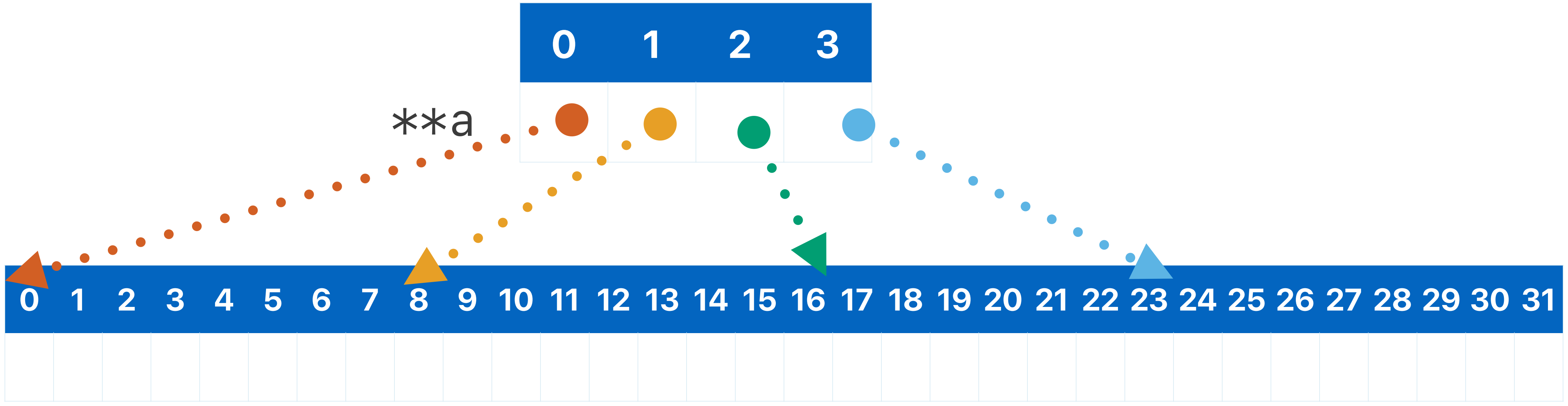
```
a = (double **)malloc(M*sizeof(double *));
for(i = 0; i < N; i++)
{
   a[i] = (double *)malloc(N*sizeof(double));
}
```

a[i][j] is essentially a[i*N+j]

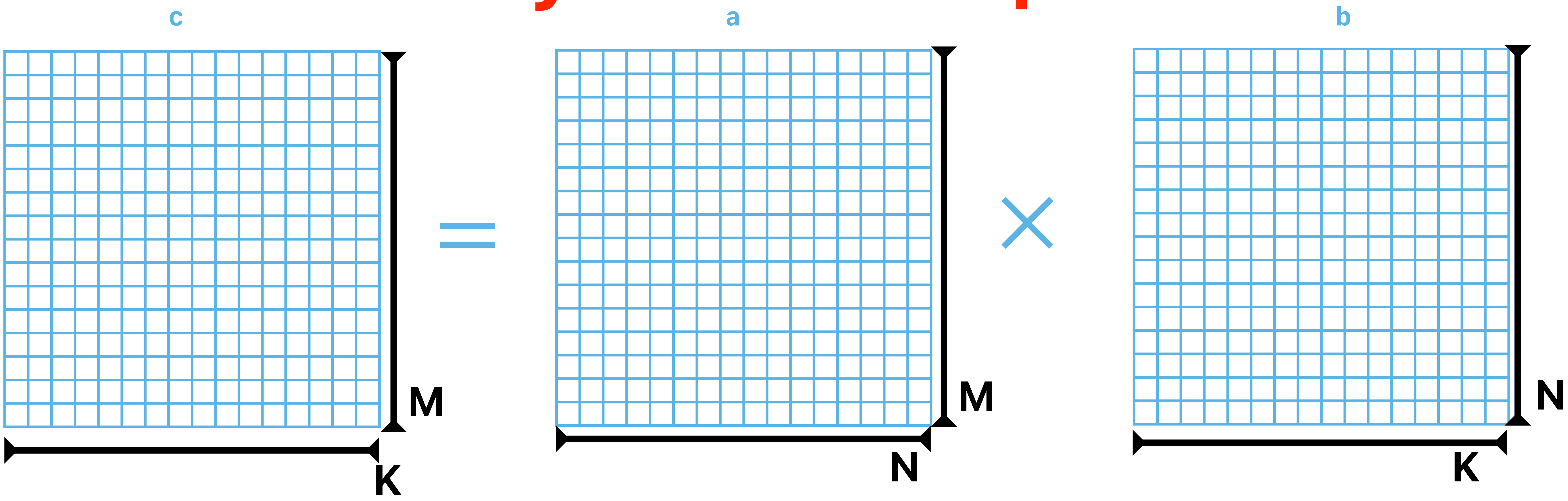
abstraction

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

physical implementation



Case Study: Matrix Multiplications



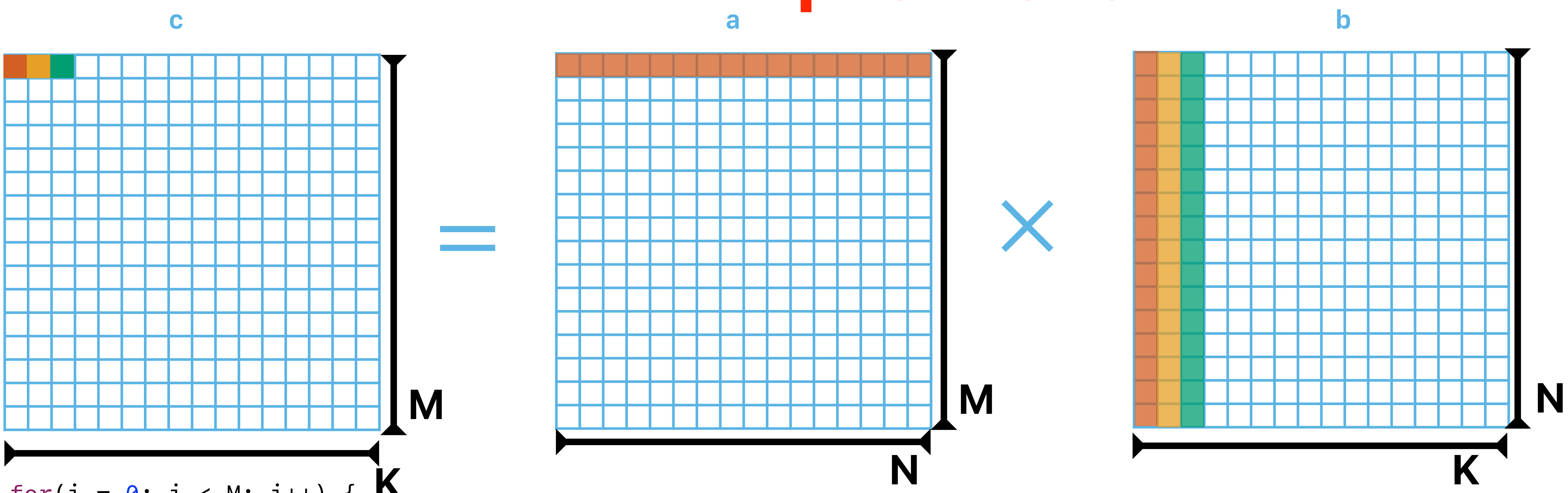

```
for(i = 0; i < M; i++) {
  for(j = 0; j < K; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Algorithm class tells you it's $O(n^3)$

If M=N=K=1024, it takes about 1 sec

How long is it take when M=N=K=2048?

Matrix Multiplications




```
for(i = 0; i < M; i++) {
  for(j = 0; j < K; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Matrix Multiplication — let's consider "b"

```
for(i = 0; i < M; i++) {
  for(j = 0; j < K; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

	Address	Tag	Index
b[0][0]	0x20000	0x20	0x0
b[1][0]	0x24000	0x24	0x0
b[2][0]	0x28000	0x28	0x0
b[3][0]	0x2C000	0x2C	0x0
b[4][0]	0x30000	0x30	0x0
b[5][0]	0x34000	0x34	0x0
b[6][0]	0x38000	0x38	0x0
b[7][0]	0x3C000	0x3C	0x0
b[8][0]	0x40000	0x40	0x0
b[9][0]	0x44000	0x44	0x0
b[10][0]	0x48000	0x48	0x0
b[11][0]	0x4C000	0x4C	0x0
b[12][0]	0x50000	0x50	0x0
b[13][0]	0x54000	0x54	0x0
b[14][0]	0x58000	0x58	0x0
b[15][0]	0x5C000	0x5C	0x0
b[16][0]	0x60000	0x60	0x0

- If the row dimension (N) of your matrix is 2048, each row element with the same column index is
  $$2048 \times 8 = 16384 = 0x4000$$
  away from each other

Each set can store only 12 blocks! So we will start to kick out b[0][0-7], b[1][0-7] …

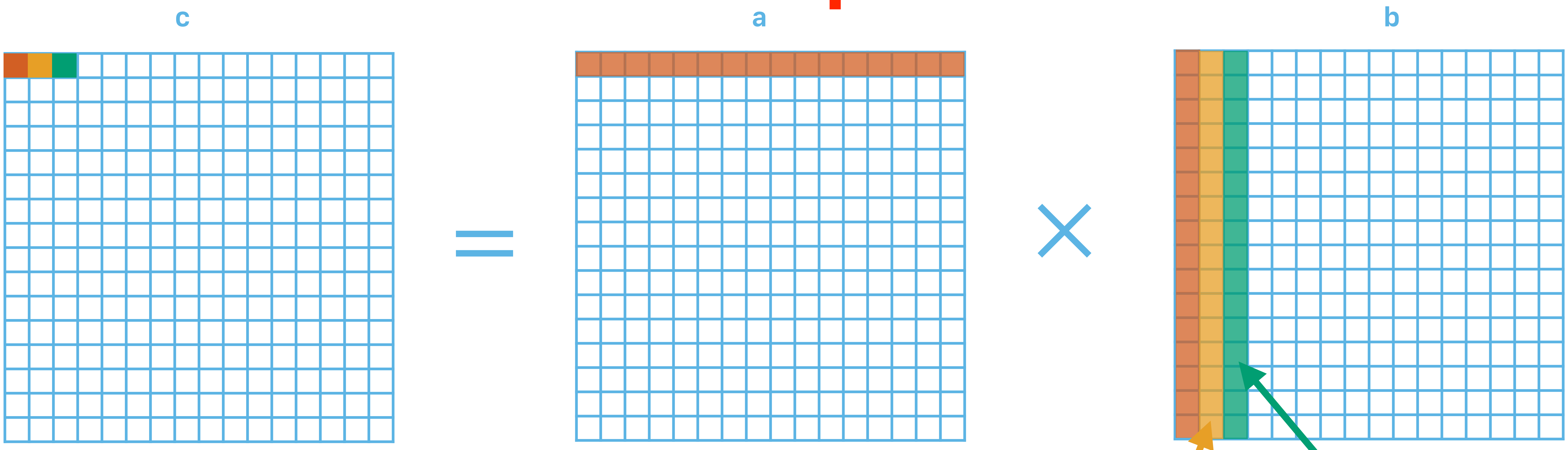
Now, when we work on c[0][1]

	Address	Tag	Index
b[0][0]	0x20000	0x20	0x0
b[1][0]	0x24000	0x24	0x0
b[2][0]	0x28000	0x28	0x0
b[3][0]	0x2C000	0x2C	0x0
b[4][0]	0x30000	0x30	0x0
b[5][0]	0x34000	0x34	0x0
b[6][0]	0x38000	0x38	0x0
b[7][0]	0x3C000	0x3C	0x0
b[8][0]	0x40000	0x40	0x0
b[9][0]	0x44000	0x44	0x0
b[10][0]	0x48000	0x48	0x0
b[11][0]	0x4C000	0x4C	0x0
b[12][0]	0x50000	0x50	0x0
b[13][0]	0x54000	0x54	0x0
b[14][0]	0x58000	0x58	0x0
b[15][0]	0x5C000	0x5C	0x0
b[16][0]	0x60000	0x60	0x0

	Address	Tag	Index	
b[0][1]	0x20008	0x20	0x0	Conflict Miss
b[1][1]	0x24008	0x24	0x0	Conflict Miss
b[2][1]	0x28008	0x28	0x0	Conflict Miss
b[3][1]	0x2C008	0x2C	0x0	Conflict Miss
b[4][1]	0x30008	0x30	0x0	Conflict Miss
b[5][1]	0x34008	0x34	0x0	Conflict Miss
b[6][1]	0x38008	0x38	0x0	Conflict Miss
b[7][1]	0x3C008	0x3C	0x0	Conflict Miss
b[8][1]	0x40008	0x40	0x0	Conflict Miss
b[9][1]	0x44008	0x44	0x0	Conflict Miss
b[10][1]	0x48008	0x48	0x0	Conflict Miss
b[11][1]	0x4C008	0x4C	0x0	Conflict Miss
b[12][1]	0x50008	0x50	0x0	Conflict Miss
b[13][1]	0x54008	0x54	0x0	Conflict Miss
b[14][1]	0x58008	0x58	0x0	Conflict Miss
b[15][1]	0x5C008	0x5C	0x0	Conflict Miss
b[16][1]	0x60008	0x60	0x0	Conflict Miss

Each set can store only 12 blocks! So we will start to kick out b[0][0-7], b[1][0-7] ...

Matrix Multiplications

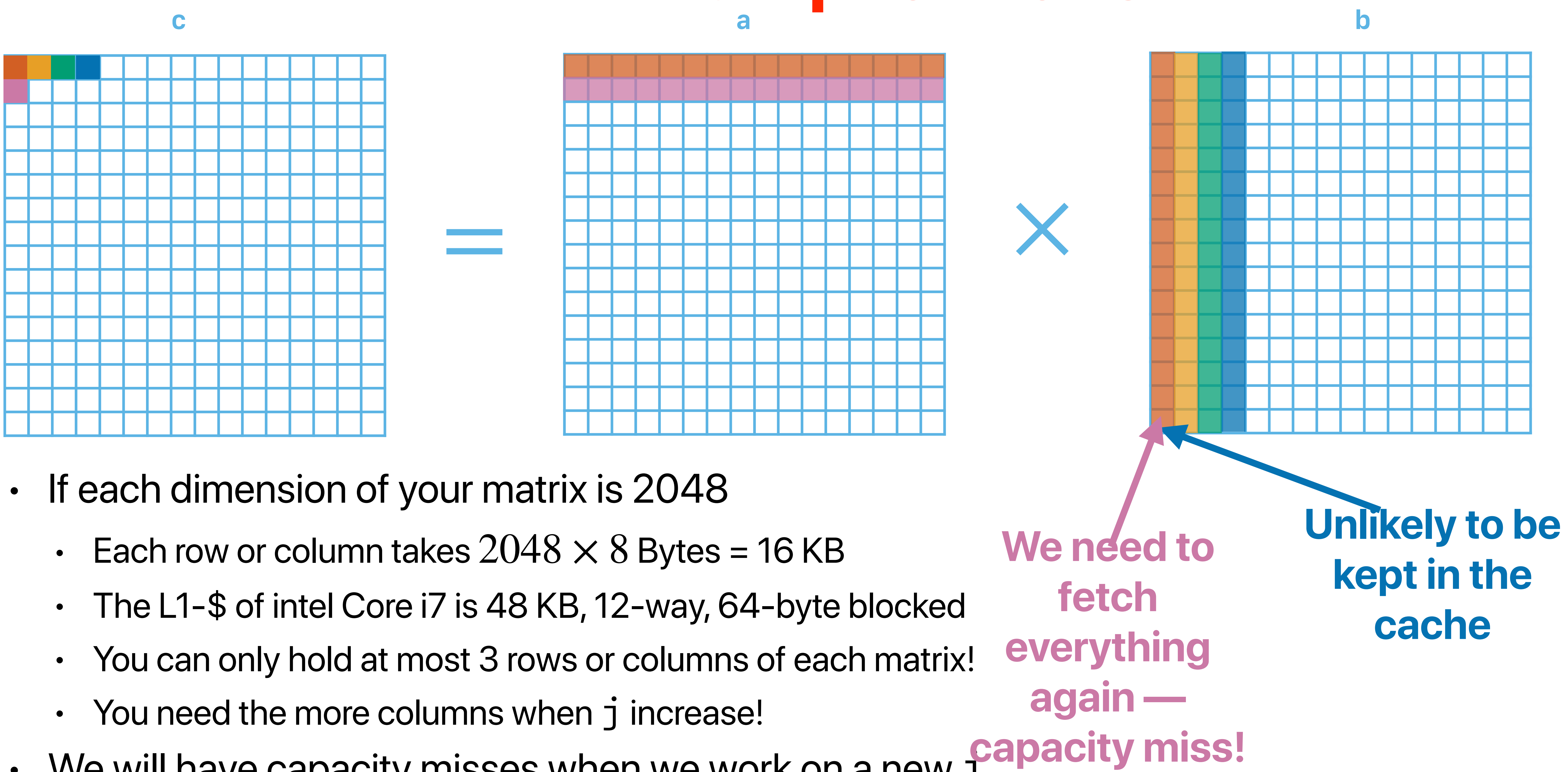



```
for(i = 0; i < M; i++) {
  for(j = 0; j < K; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

These are conflict misses as we have cached them before

These are conflict misses as we have cached them before

Matrix Multiplications



- If each dimension of your matrix is 2048
  - Each row or column takes 2048×8 Bytes = 16 KB
  - The L1-$ of intel Core i7 is 48 KB, 12-way, 64-byte blocked
  - You can only hold at most 3 rows or columns of each matrix!
  - You need the more columns when `j` increase!
- We will have capacity misses when we work on a new `i`

Ideas regarding reducing misses in matrix multiplications

- Reducing conflict misses — we need to reduce the length of a column that we visit within a period of time
- Reducing capacity misses — we need to reduce the length of a row that we visit within a period of time

Mathematical view of MM

$$c_{i,j} = \sum_{k=0}^{k=N-1} a_{i,k} \times b_{k,j} = \sum_{k=0}^{k=\frac{N}{2}-1} a_{i,k} \times b_{k,j} + \sum_{k=\frac{N}{2}}^{k=N-1} a_{i,k} \times b_{k,j}$$

$$= \sum_{k=0}^{k=\frac{N}{4}-1} a_{i,k} \times b_{k,j} + \sum_{k=\frac{N}{4}}^{k=\frac{N}{2}-1} a_{i,k} \times b_{k,j} + \sum_{k=\frac{N}{2}}^{k=\frac{3N}{4}-1} a_{i,k} \times b_{k,j} + \sum_{k=3N4-1}^{k=N-1} a_{i,k} \times b_{k,j}$$

$$\vdots$$

Let's break up the multiplications and accumulations into something fits in the cache well

Matrix Multiplication — let's consider "b"

```
for(i = 0; i < M; i++) {
  for(j = 0; j < K; j++) {
    for(k = 0; k < N; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

- If the row dimension of your matrix is 2048, each row element with the same column index is

$$2048 \times 8 = 16384 = 0x4000$$

away from each other

	Address	Tag	Index
b[0][0]	0x20000	0x20	0x0
b[1][0]	0x24000	0x24	0x0
b[2][0]	0x28000	0x28	0x0
b[3][0]	0x2C000	0x2C	0x0
b[4][0]	0x30000	0x30	0x0
b[5][0]	0x34000	0x34	0x0
b[6][0]	0x38000	0x38	0x0
b[7][0]	0x3C000	0x3C	0x0
b[8][0]	0x40000	0x40	0x0
b[9][0]	0x44000	0x44	0x0
b[10][0]	0x48000	0x48	0x0
b[11][0]	0x4C000	0x4C	0x0
b[12][0]	0x50000	0x50	0x0
b[13][0]	0x54000	0x54	0x0
b[14][0]	0x58000	0x58	0x0
b[15][0]	0x5C000	0x5C	0x0
b[16][0]	0x60000	0x60	0x0

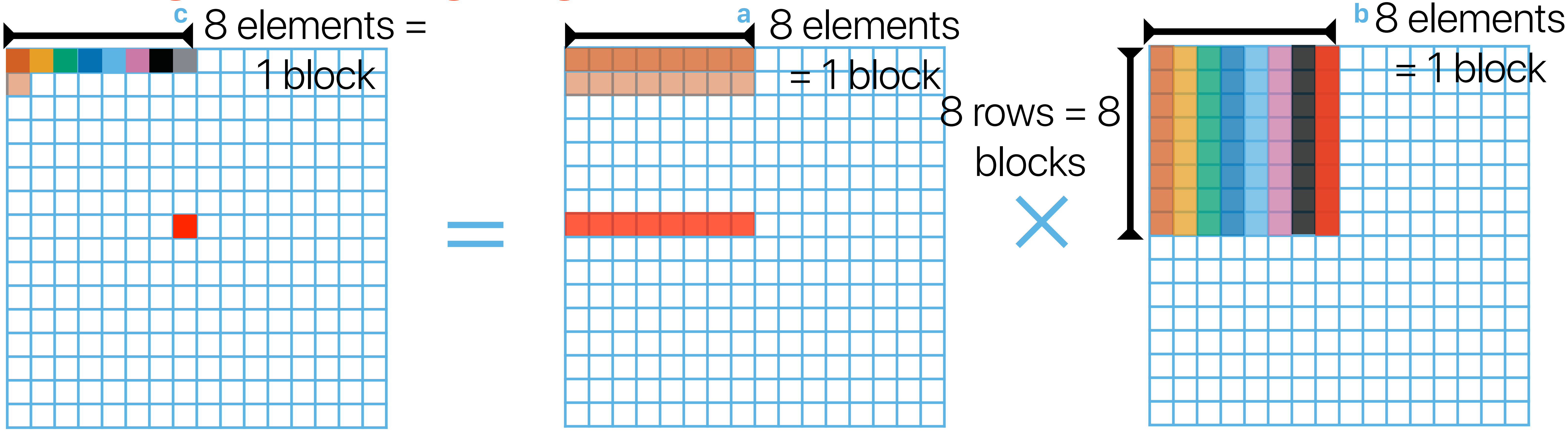
If we stop at somewhere before 12 blocks, we should be fine!

Since each block has 8 elements, let's break up in 8 for now

— 8 elements from a[i]

— 8 columns each covers 8 rows

Tiling/Blocking Algorithm for Matrix Multiplications



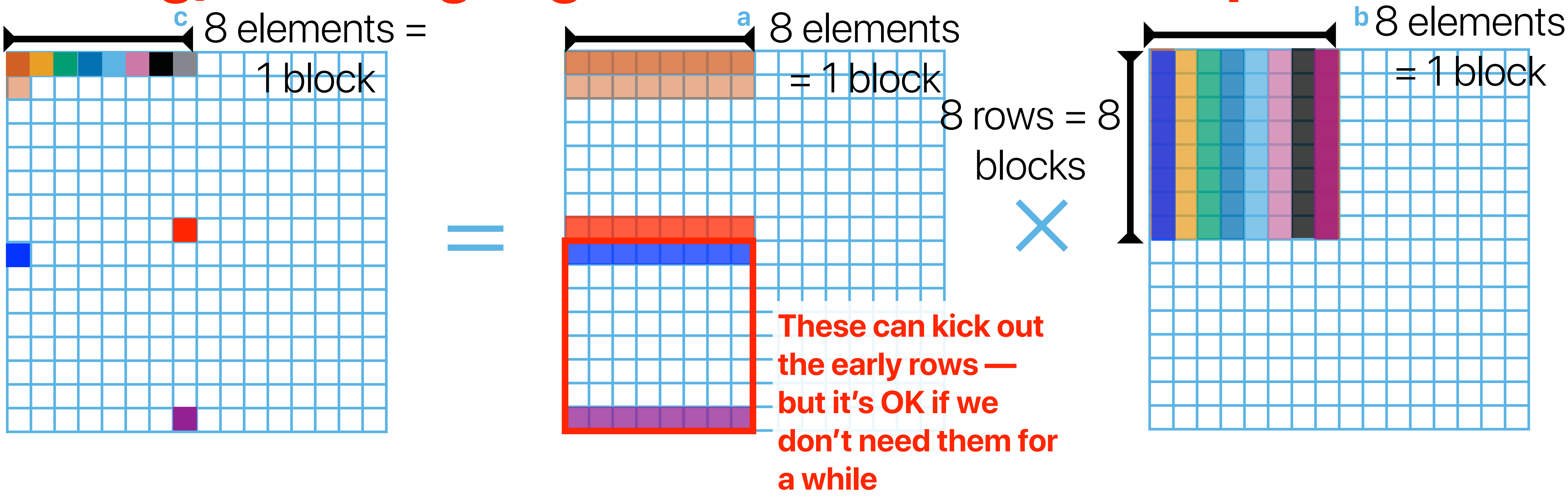
Only used 10 blocks for now

These are still around when we move to the next row in the "tile"

Only compulsory misses —

$$miss\_rate = \frac{total\ misses}{total\ accesses} = \frac{8+8}{3 \times 8 \times 8} = 0.083$$

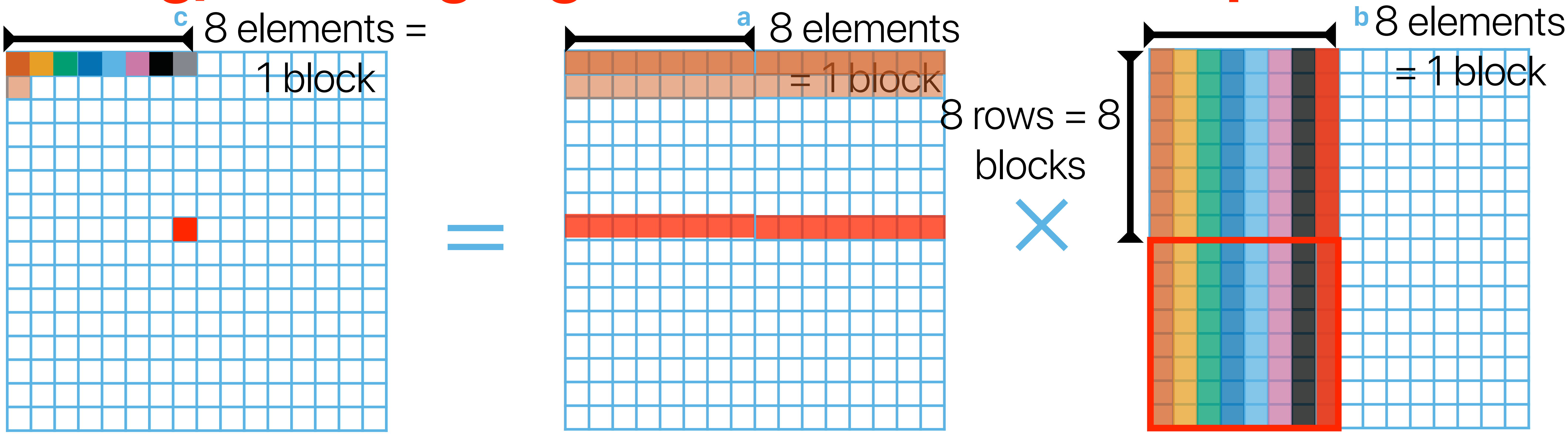
Tiling/Blocking Algorithm for Matrix Multiplications



Bringing miss rate even further lower now —

$$miss\_rate = \frac{total\ misses}{total\ accesses} = \frac{8+8+8}{3 \times 8 \times 8 + 3 \times 8 \times 8} = 0.042$$

Tiling/Blocking Algorithm for Matrix Multiplications




```
for(i = 0; i < M; i+=tile_size)
    for(j = 0; j < K; j+=tile_size)
        for(k = 0; k < N; k+=tile_size)
            for(ii = i; ii < i+tile_size; ii++)
                for(jj = j; jj < j+tile_size; jj++)
                    for(kk = k; kk < k+tile_size; kk++)
                        c[ii][jj] += a[ii][kk]*b[kk][jj];
```

These can kick out the upper portion of the columns — but it's OK if we don't need them for a while

Takeaways: Software Optimizations

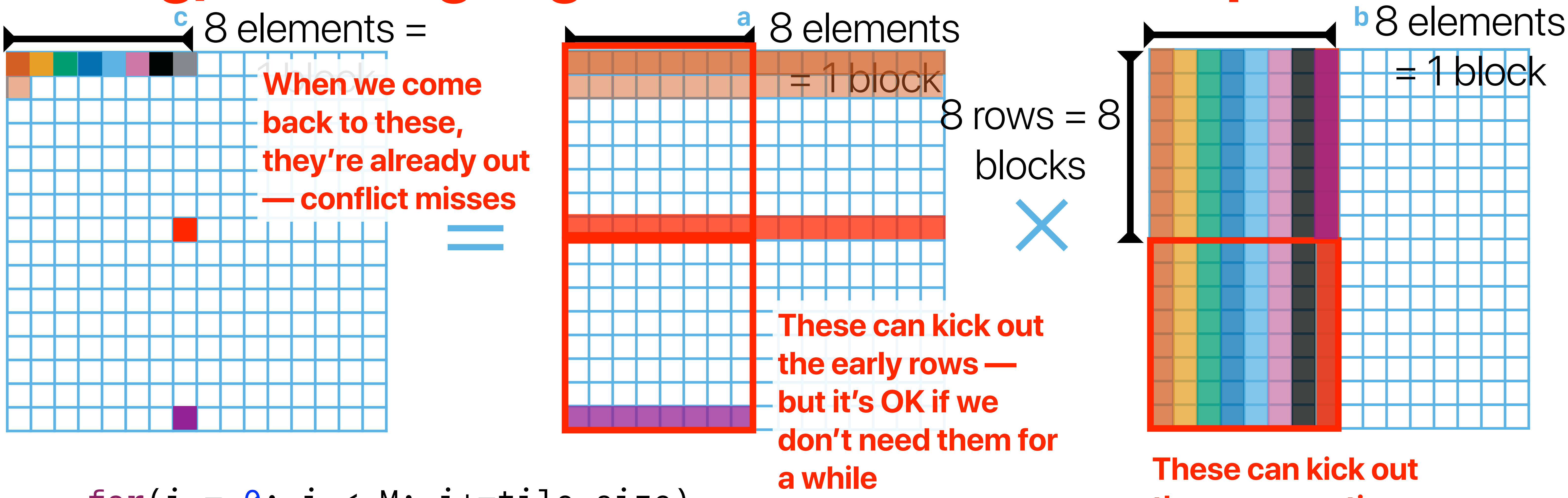
- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity
- Blocking/tiling — capacity miss, conflict miss

Matrix Transpose

```
for(i = 0; i < M; i+=tile_size) {
  for(j = 0; j < K; j+=tile_size) {
    for(k = 0; k < N; k+=tile_size) {
      for(ii = i; ii < i+tile_size; ii++)
        for(jj = j; jj < j+tile_size; jj++)
          for(kk = k; kk < k+tile_size; kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

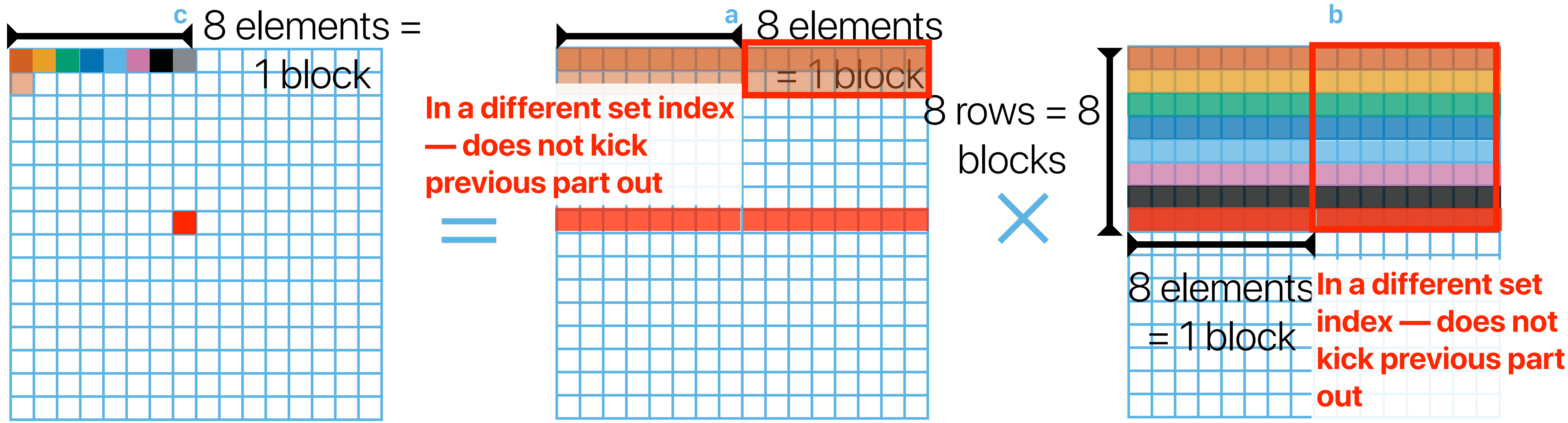
```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}

  for(i = 0; i < M; i+=tile_size) {
    for(j = 0; j < K; j+=tile_size) {
      for(k = 0; k < N; k+=tile_size) {
        for(ii = i; ii < i+tile_size; ii++)
          for(jj = j; jj < j+tile_size; jj++)
            for(kk = k; kk < k+tile_size; kk++)
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
      }
    }
  }
```

Tiling/Blocking Algorithm for Matrix Multiplications




```
for(i = 0; i < M; i+=tile_size)
    for(j = 0; j < K; j+=tile_size)
        for(k = 0; k < N; k+=tile_size)
            for(ii = i; ii < i+tile_size; ii++)
                for(jj = j; jj < j+tile_size; jj++)
                    for(kk = k; kk < k+tile_size; kk++)
                        c[ii][jj] += a[ii][kk]*b[kk][jj];
```

Tiling/Blocking Algorithm for Transposed Matrix Multiplications



We can make the "tile_size" larger without interfacing conflict misses

```
for(i = 0; i < M; i+=tile_size)
    for(j = 0; j < K; j+=tile_size)
        for(k = 0; k < N; k+=tile_size)
            for(ii = i; ii < i+tile_size; ii++)
                for(jj = j; jj < j+tile_size; jj++)
                    for(kk = k; kk < k+tile_size; kk++)
                        c[ii][jj] += a[ii][kk]*b_t[jj][kk];
```

Takeaways: Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Loop interchange — conflict/capacity miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity
- Blocking/tiling — capacity miss, conflict miss
- Matrix transpose (a technique changes layout) — conflict misses