Introduction to Parallel Computing (CMSC416 / CMSC616)

Parallel Algorithms

Abhinav Bhatele, Department of Computer Science

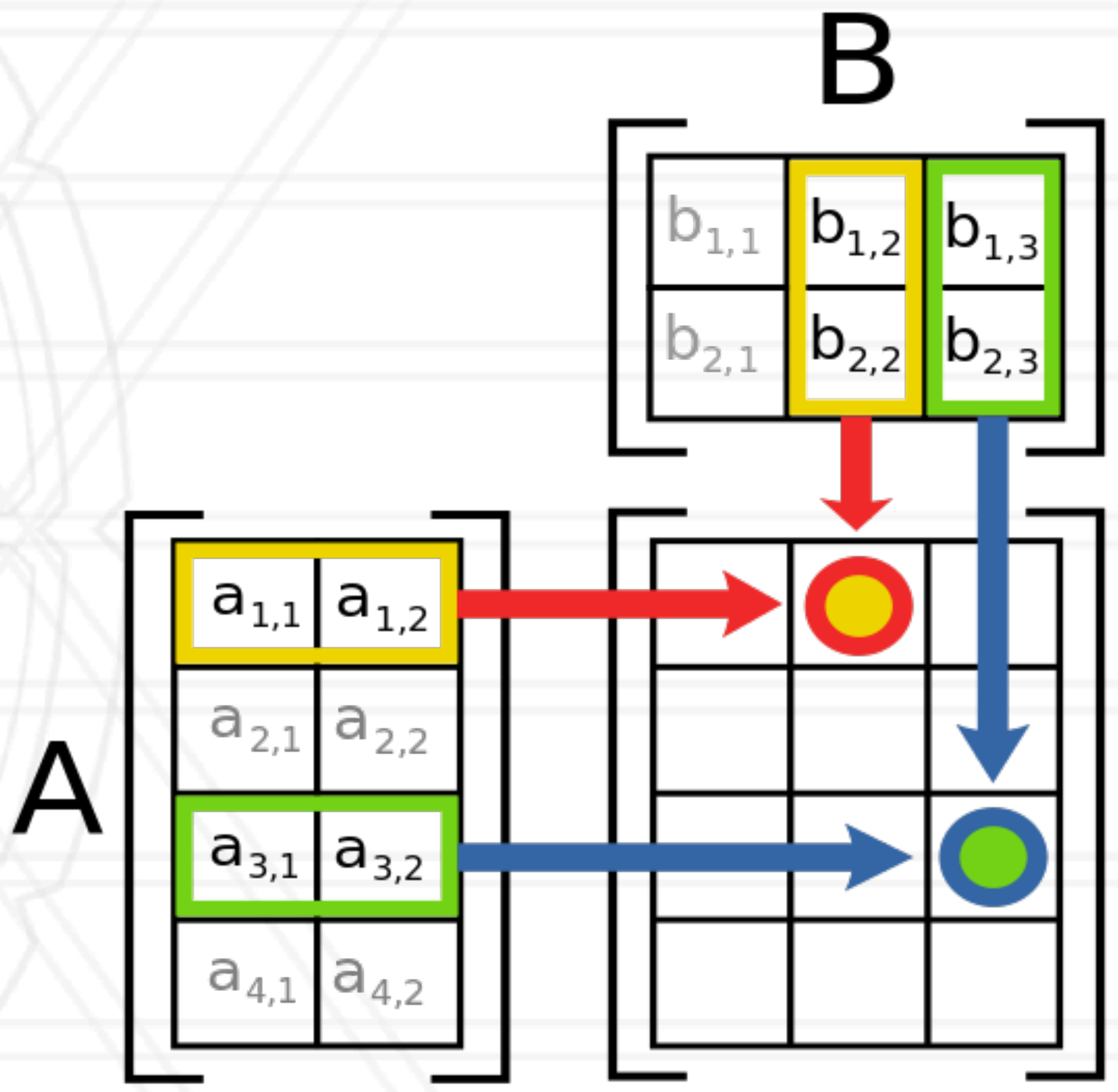
UNIVERSITY OF MARYLAND

Announcements

- Assignment 1 is due on Oct 4 11:59 pm
    - No late submissions allowed
- Assignment 2 has been posted and is due on Oct 11 11:59 pm

Matrix multiplication

```
for (i=0; i<M; i++)
  for (j=0; j<N; j++)
    for (k=0; k<L; k++)
      C[i][j] += A[i][k]*B[k][j];
```

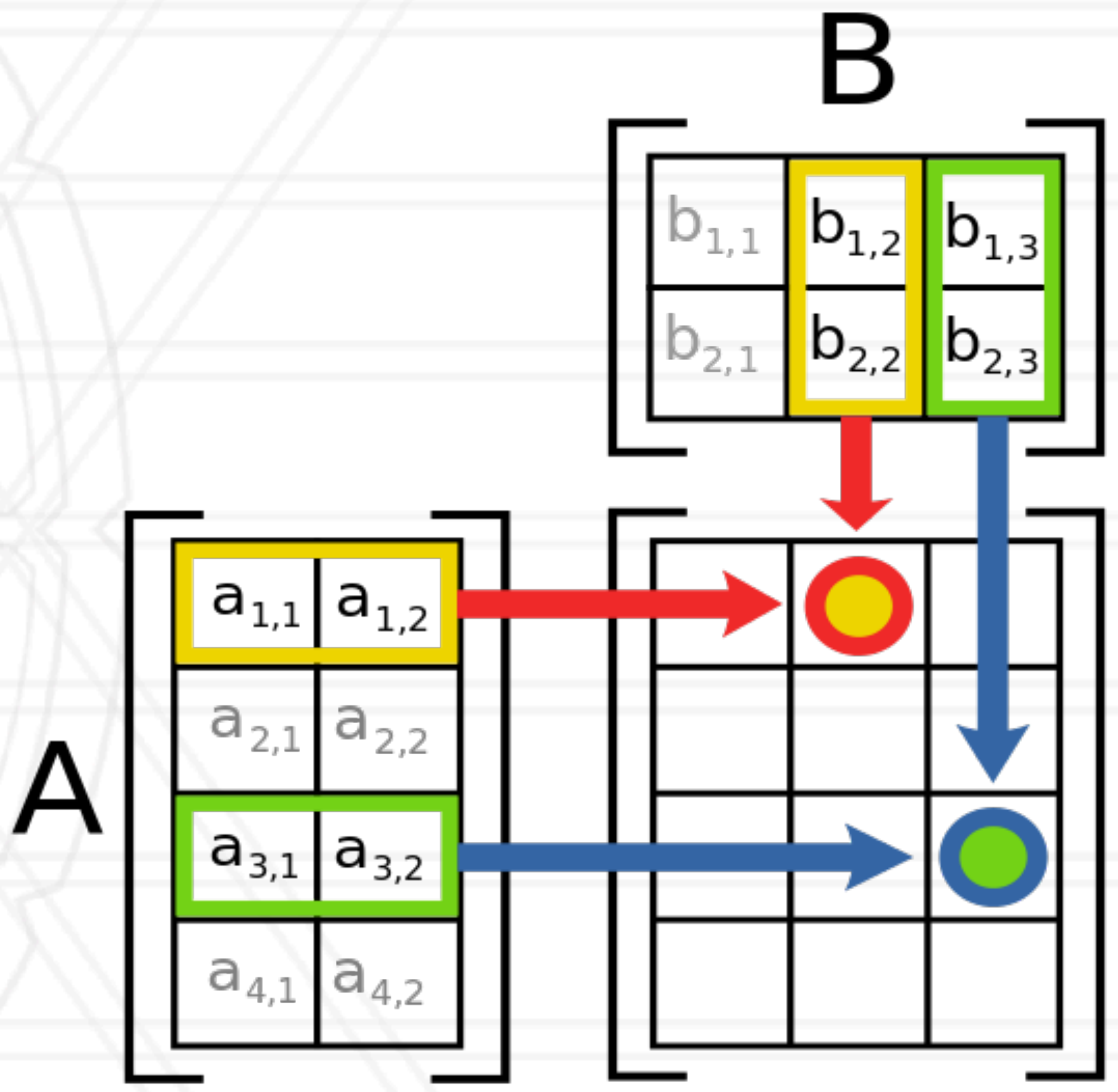


https://en.wikipedia.org/wiki/Matrix_multiplication

Matrix multiplication

```
for (i=0; i<M; i++)
  for (j=0; j<N; j++)
    for (k=0; k<L; k++)
      C[i][j] += A[i][k]*B[k][j];
```

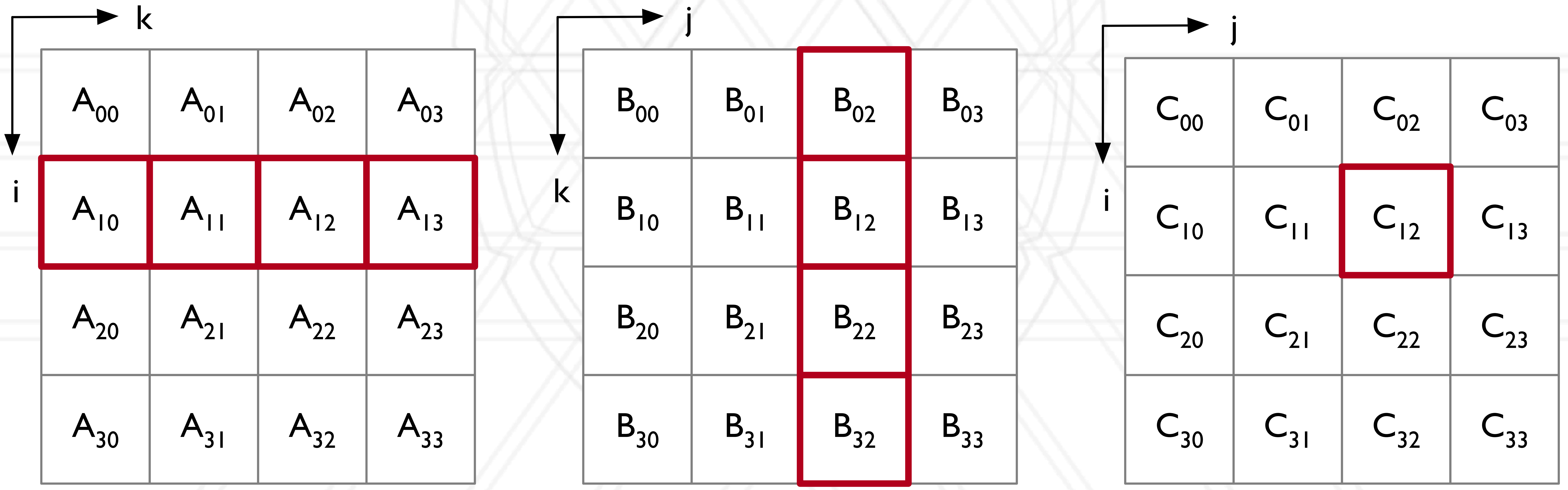
Any performance issues for large arrays?



https://en.wikipedia.org/wiki/Matrix_multiplication

Blocking to improve cache performance

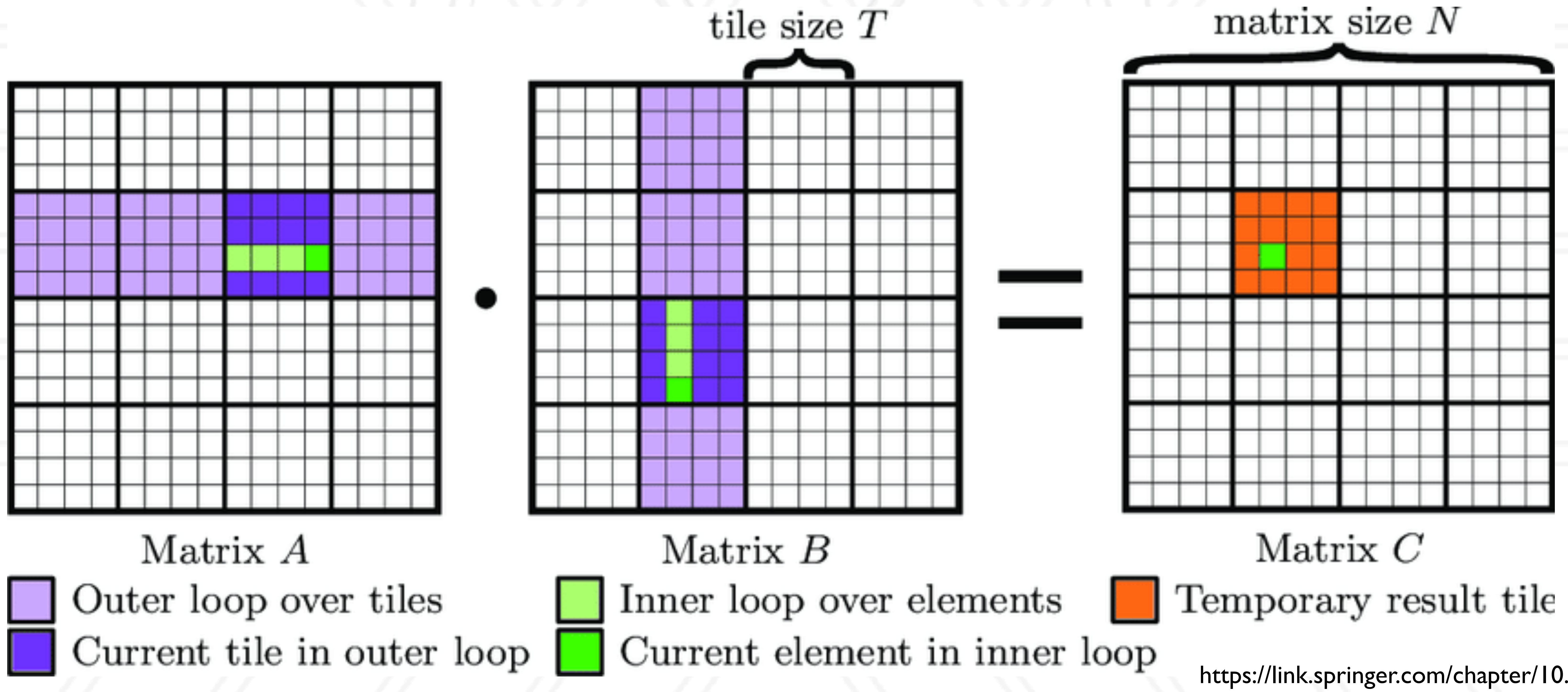
- Create smaller blocks that fit in cache: leads to cache reuse
- $C_{12} = A_{10} * B_{02} + A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32}$



https://link.springer.com/chapter/10.1007/978-3-319-67630-2_36

Blocking to improve cache performance

- Create smaller blocks that fit in cache: leads to cache reuse
- $C_{12} = A_{10} * B_{02} + A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32}$



https://link.springer.com/chapter/10.1007/978-3-319-67630-2_36

Blocked (tiled) matrix multiply

```
for (ii = 0; ii < n; ii+=B) {
  for (jj = 0; jj < n; jj+=B) {
    for (kk = 0; kk < n; kk+=B) {
      for (i = ii; i < ii+B; i++) {
        for (j = jj; j < jj+B; j++) {
          for (k = kk; k < kk+B; k++) {
            C[i][j] += A[i][k]*B[k][j];
          }
        }
      }
    }
  }
}
```

```
for (i=0; i<M; i++)
  for (j=0; j<N; j++)
    for (k=0; k<L; k++)
      C[i][j] += A[i][k]*B[k][j];
```

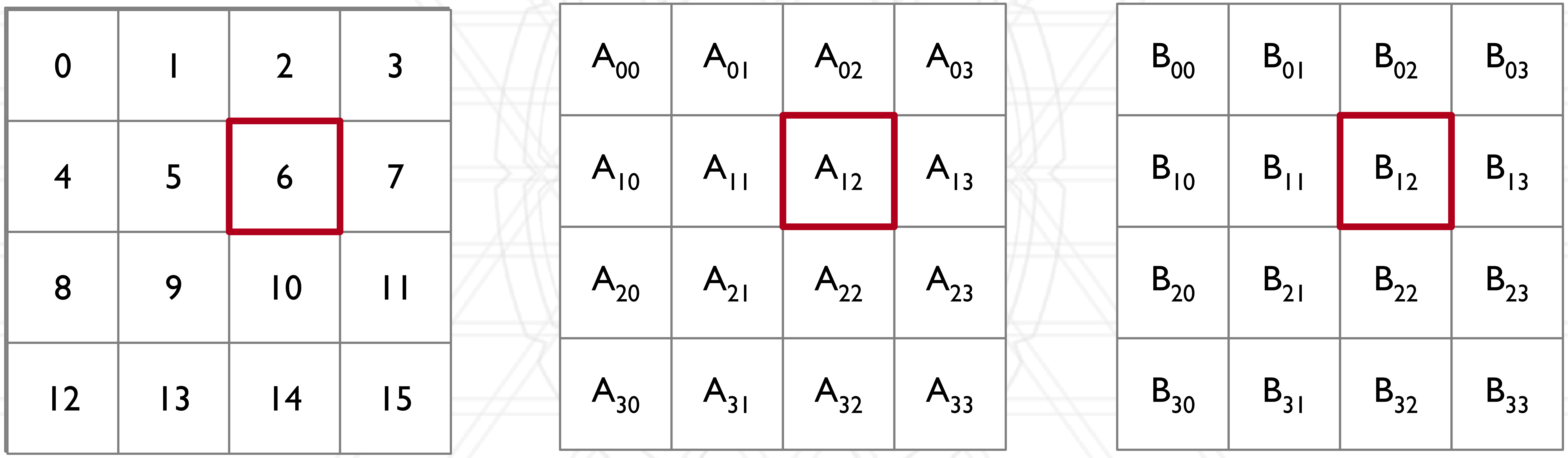
Parallel matrix multiply

- Store A and B in a distributed manner
- Communication between processes to get the right sub-matrices to each process
- Each process computes a portion of C

Cannon's 2D matrix multiply

- Arrange processes in a 2D virtual grid
- Each process computes a sub-block of C
- Requires other processes in its row and column to send A and B blocks

Cannon’s 2D matrix multiply



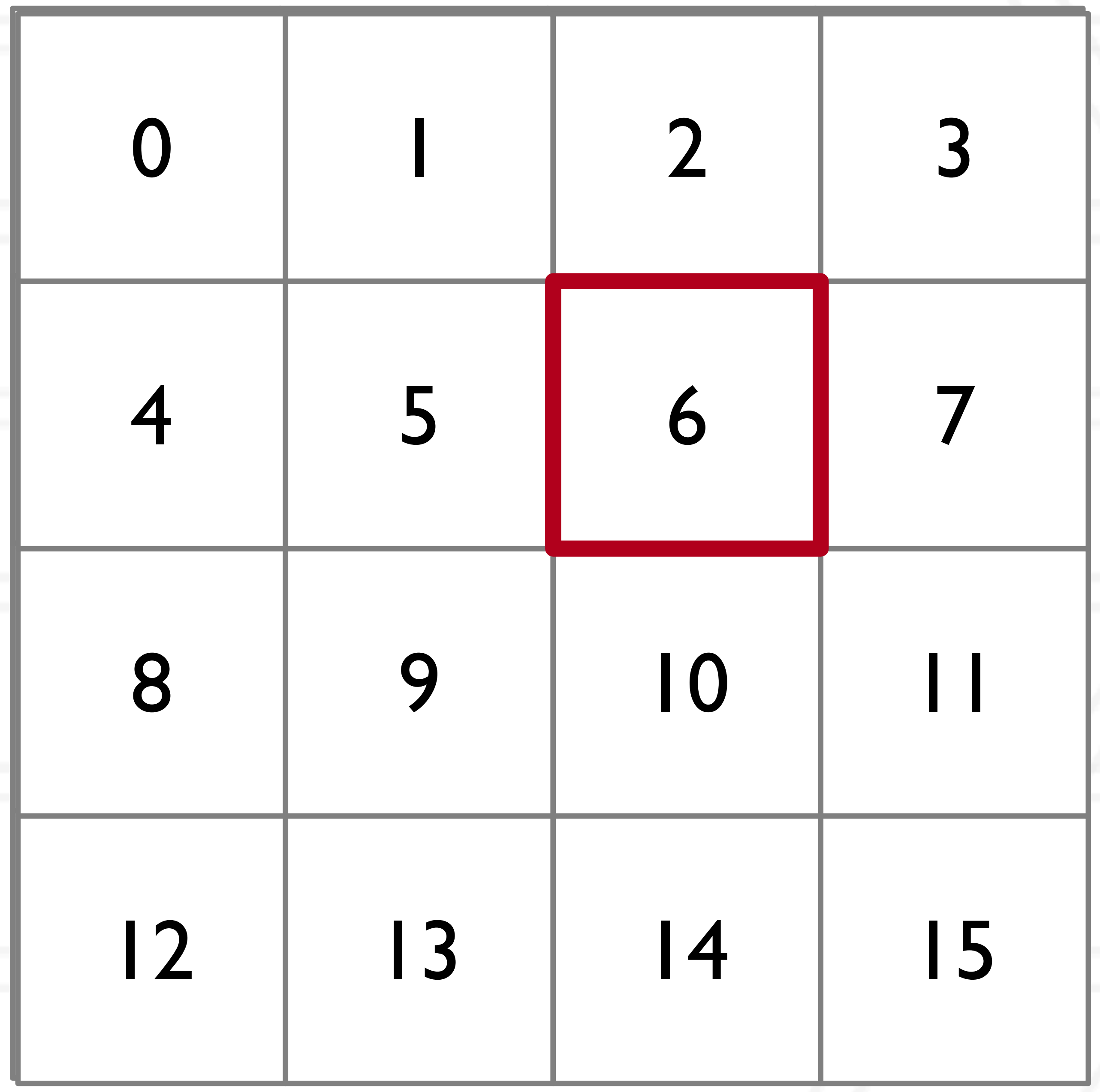
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

A_{00}	A_{01}	A_{02}	A_{03}
A_{10}	A_{11}	A_{12}	A_{13}
A_{20}	A_{21}	A_{22}	A_{23}
A_{30}	A_{31}	A_{32}	A_{33}

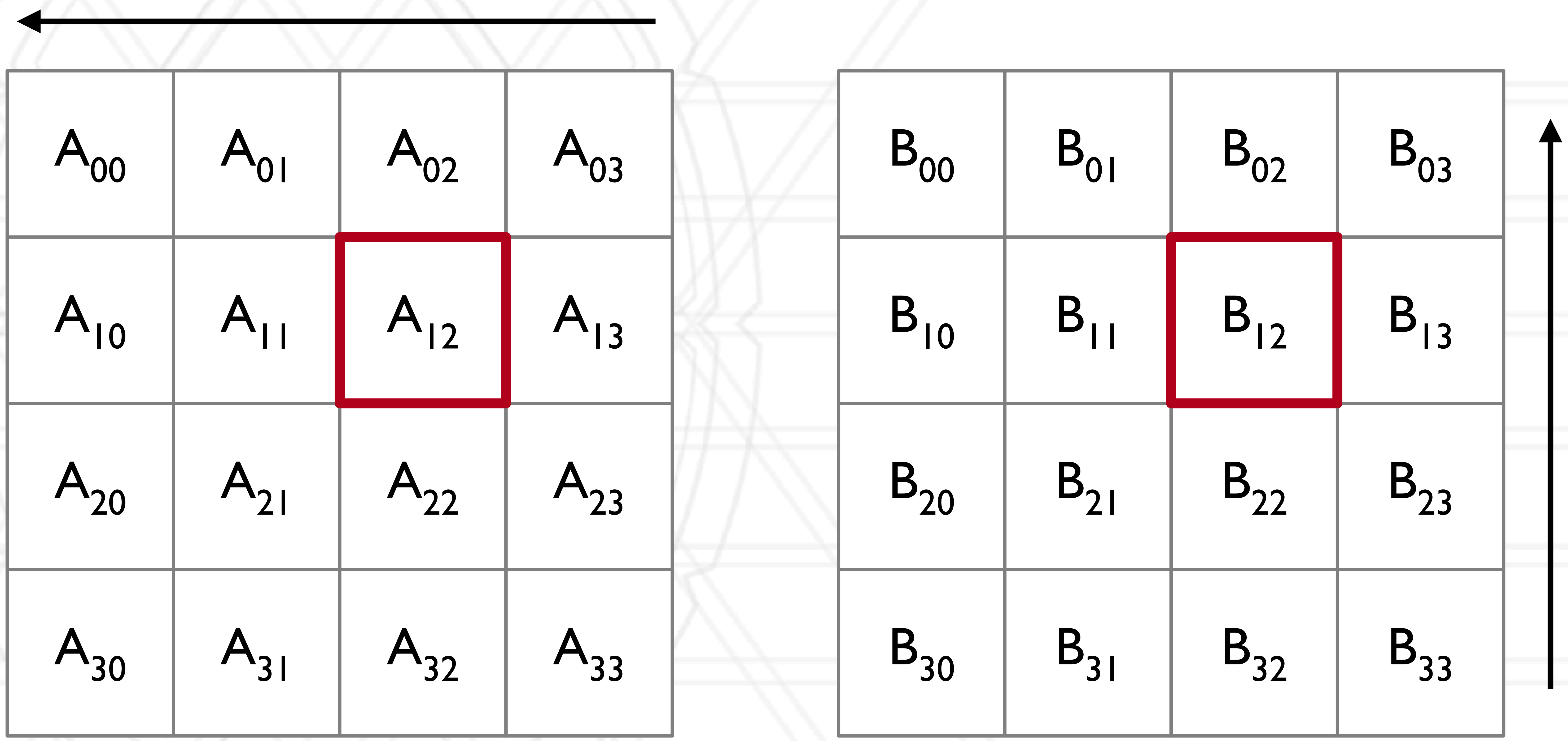
B_{00}	B_{01}	B_{02}	B_{03}
B_{10}	B_{11}	B_{12}	B_{13}
B_{20}	B_{21}	B_{22}	B_{23}
B_{30}	B_{31}	B_{32}	B_{33}

2D process grid

Cannon's 2D matrix multiply

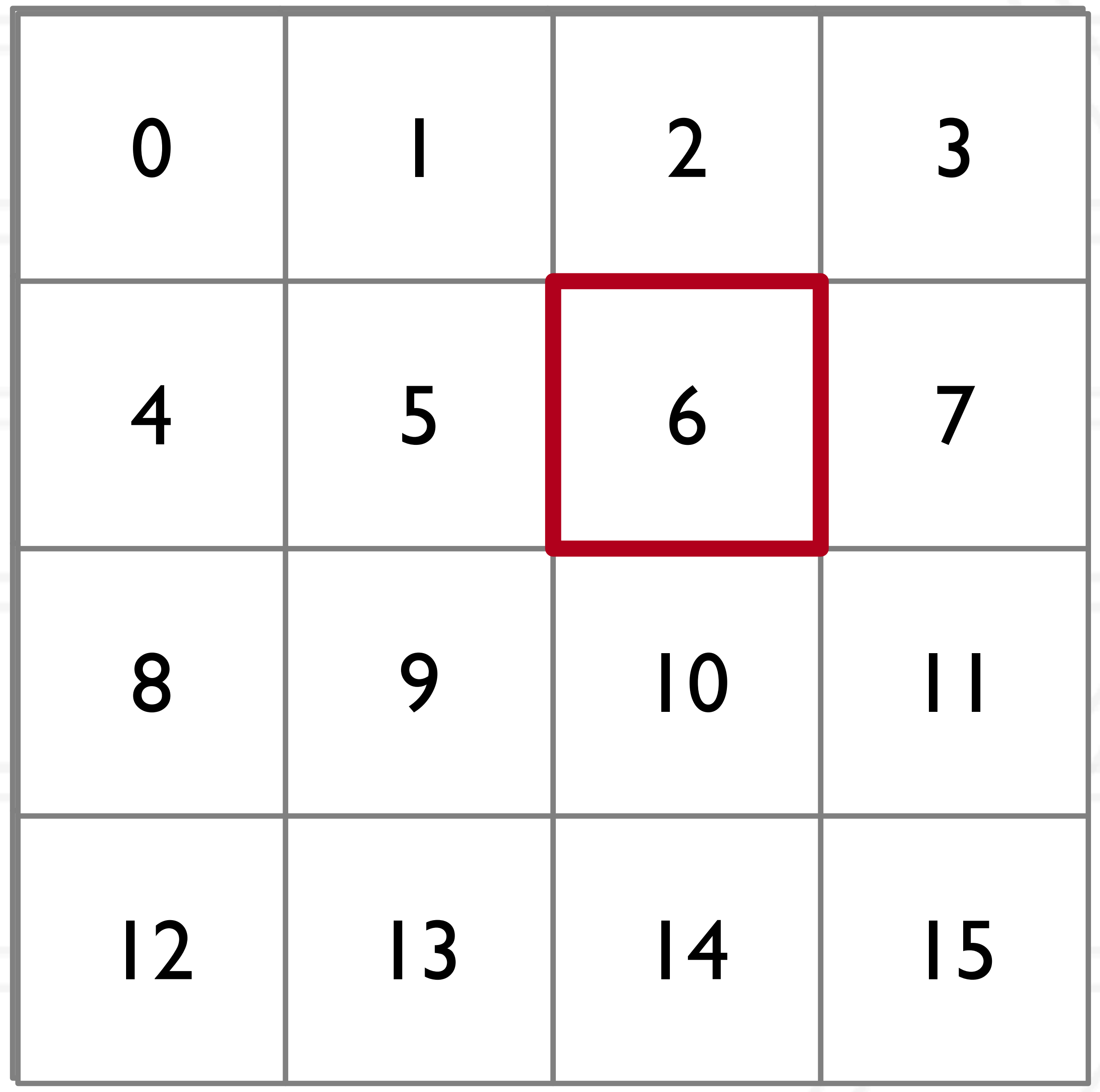


2D process grid

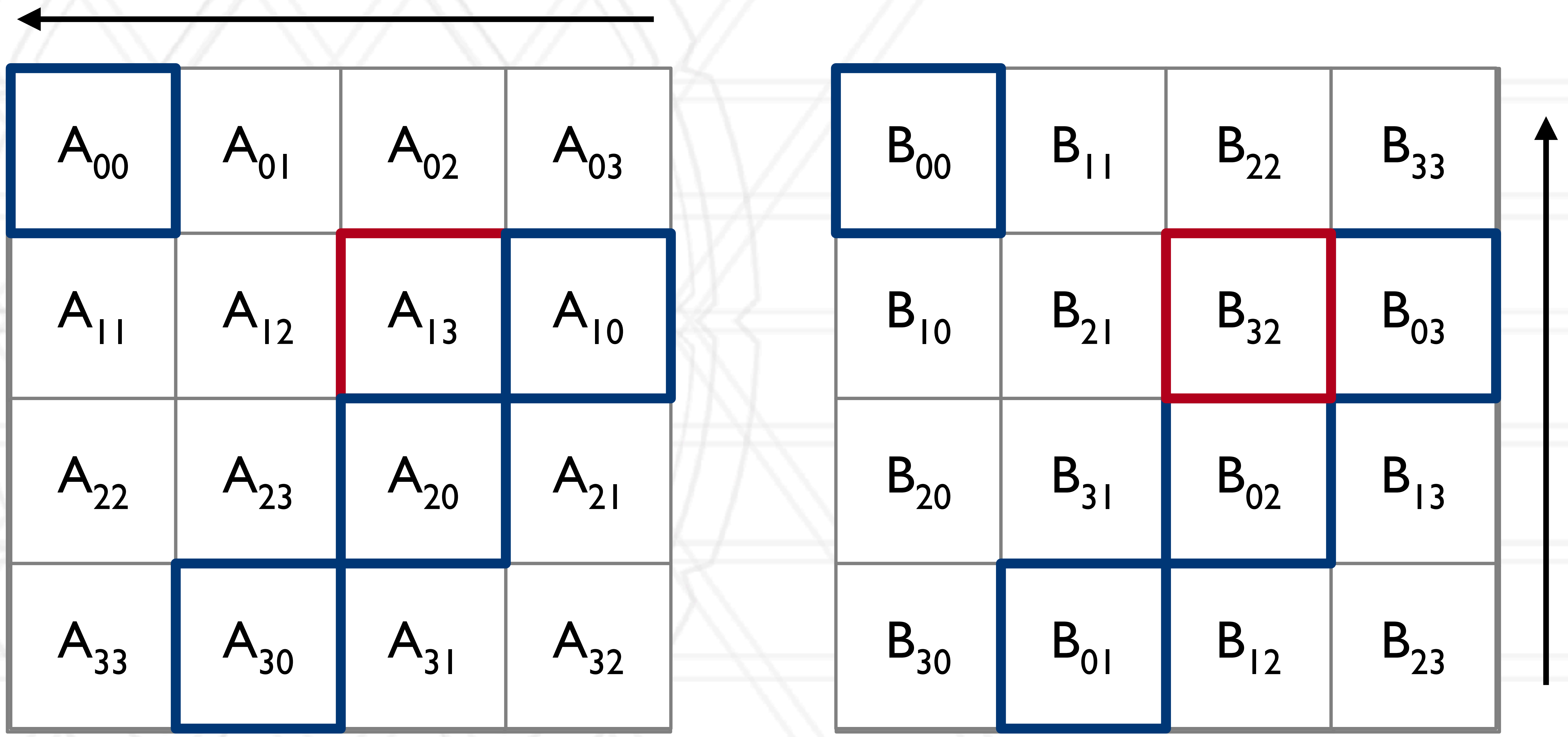


Initial skew

Cannon's 2D matrix multiply

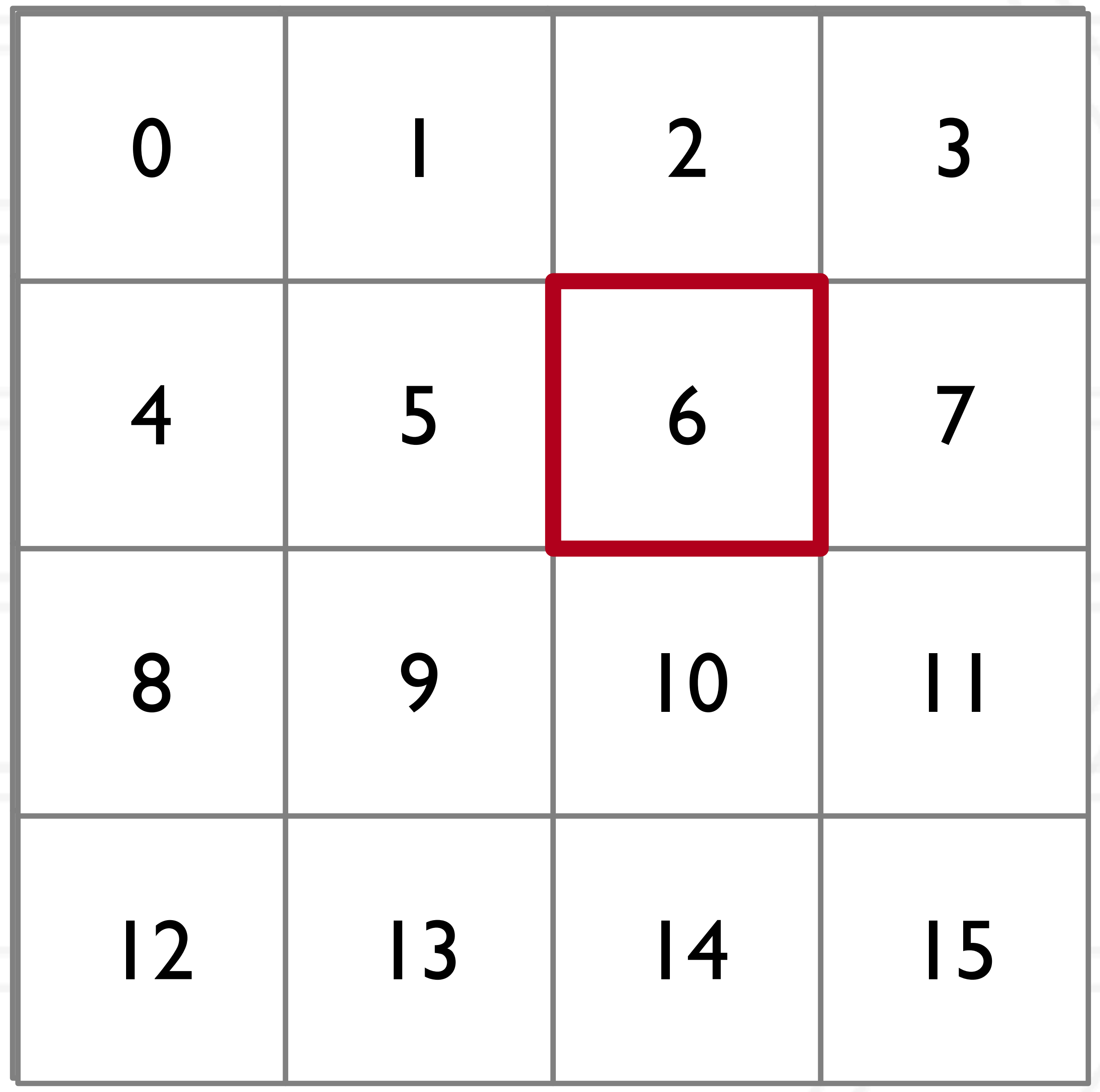


2D process grid

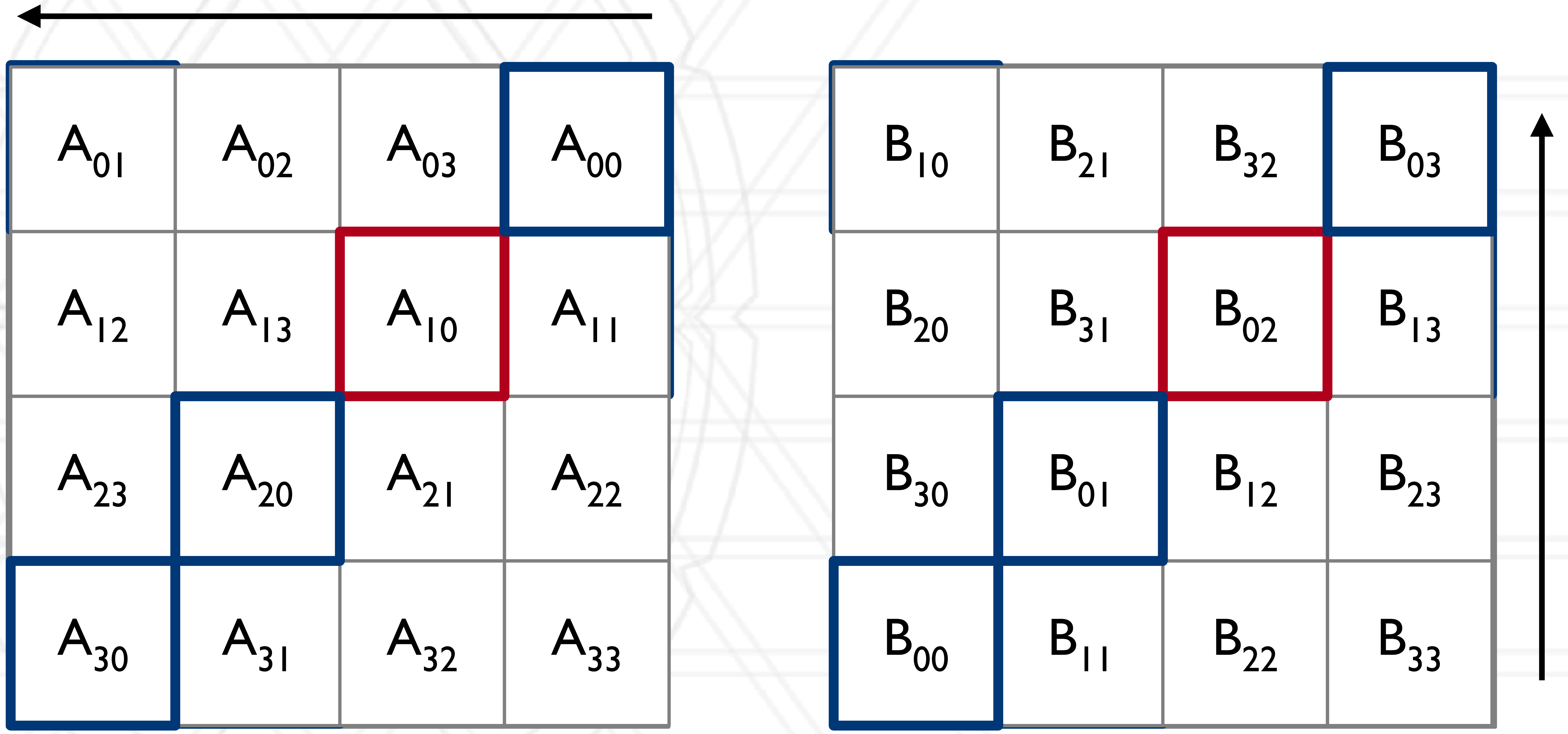

A_{00} A_{01} A_{02} A_{03}
A_{11} A_{12} A_{13} A_{10}
A_{22} A_{23} A_{20} A_{21}
A_{33} A_{30} A_{31} A_{32}
B_{00} B_{11} B_{22} B_{33}
B_{10} B_{21} B_{32} B_{03}
B_{20} B_{31} B_{02} B_{13}
B_{30} B_{01} B_{12} B_{23}

Initial skew

Cannon’s 2D matrix multiply



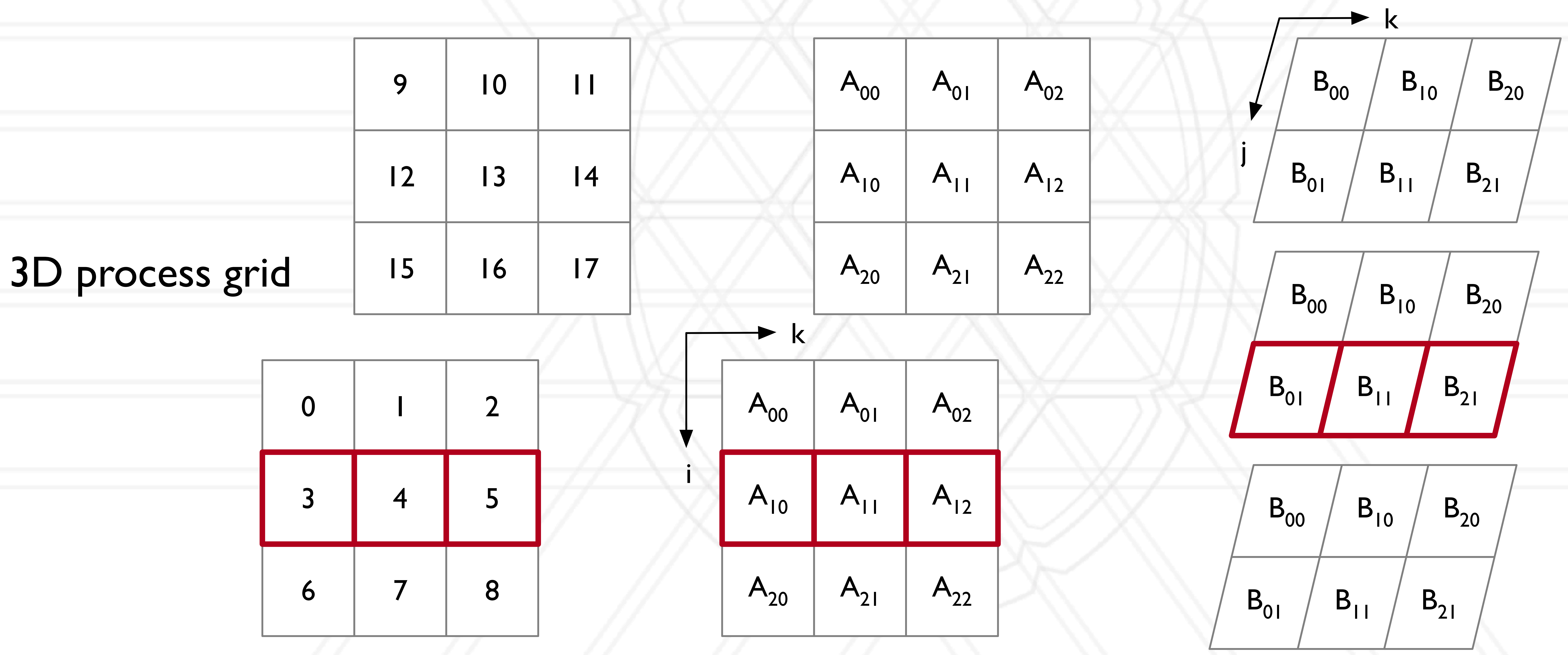
2D process grid



Shift-by-1

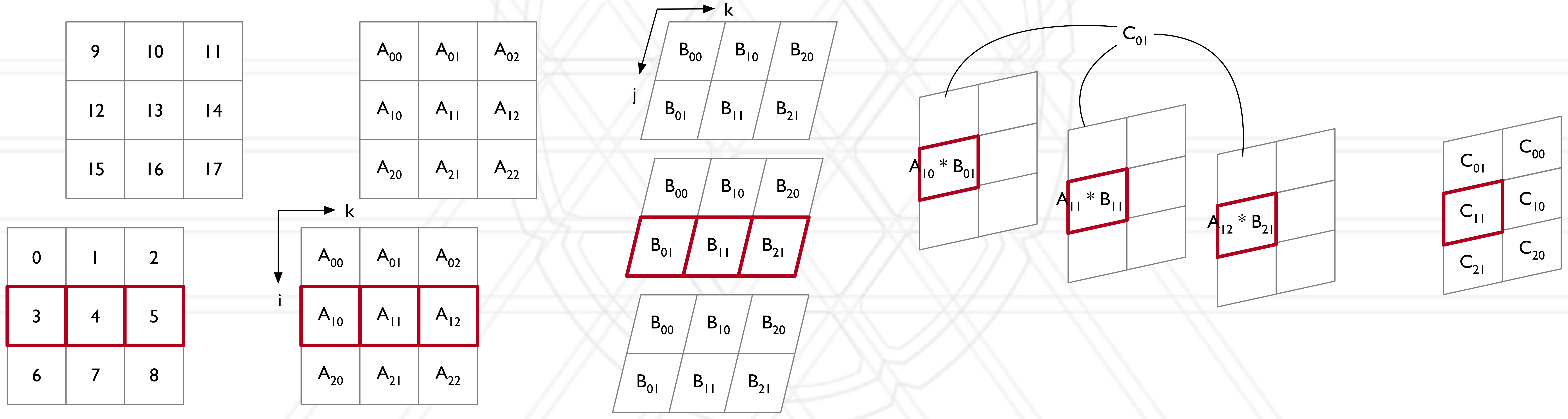
Agarwal's 3D matrix multiply

- Copy A to all i-k planes and B to all j-k planes


3D process grid
9
10
11
12
13
14
15
16
17
0
1
2
3
4
5
6
7
8
k
i
j
A_{00} A_{01} A_{02}
A_{10} A_{11} A_{12}
A_{20} A_{21} A_{22}
B_{00} B_{10} B_{20}
B_{01} B_{11} B_{21}

Agarwal’s 3D matrix multiply

- Perform a single matrix multiply to calculate partial C
- Allreduce along i-j planes to calculate final result



Communication algorithms

- Reduction
- All-to-all

Types of reduction

- Scalar reduction: every process contributes one number
  - Perform some commutative associate operation
- Vector reduction: every process contributes an array of numbers

Parallelizing reduction

MPI Reduction Algorithms: https://hcl.ucd.ie/system/files/TJS-Hasanov-2016.pdf

Parallelizing reduction

- Naive algorithm: every process sends to the root

MPI Reduction Algorithms: https://hcl.ucd.ie/system/files/TJS-Hasanov-2016.pdf

Parallelizing reduction

- Naive algorithm: every process sends to the root
- Spanning tree: organize processes in a k-ary tree

MPI Reduction Algorithms: https://hcl.ucd.ie/system/files/TJS-Hasanov-2016.pdf

Parallelizing reduction

- Naive algorithm: every process sends to the root
- Spanning tree: organize processes in a k-ary tree
- Start at leaves and send to parents
- Intermediate nodes wait to receive data from all their children

MPI Reduction Algorithms: https://hcl.ucd.ie/system/files/TJS-Hasanov-2016.pdf

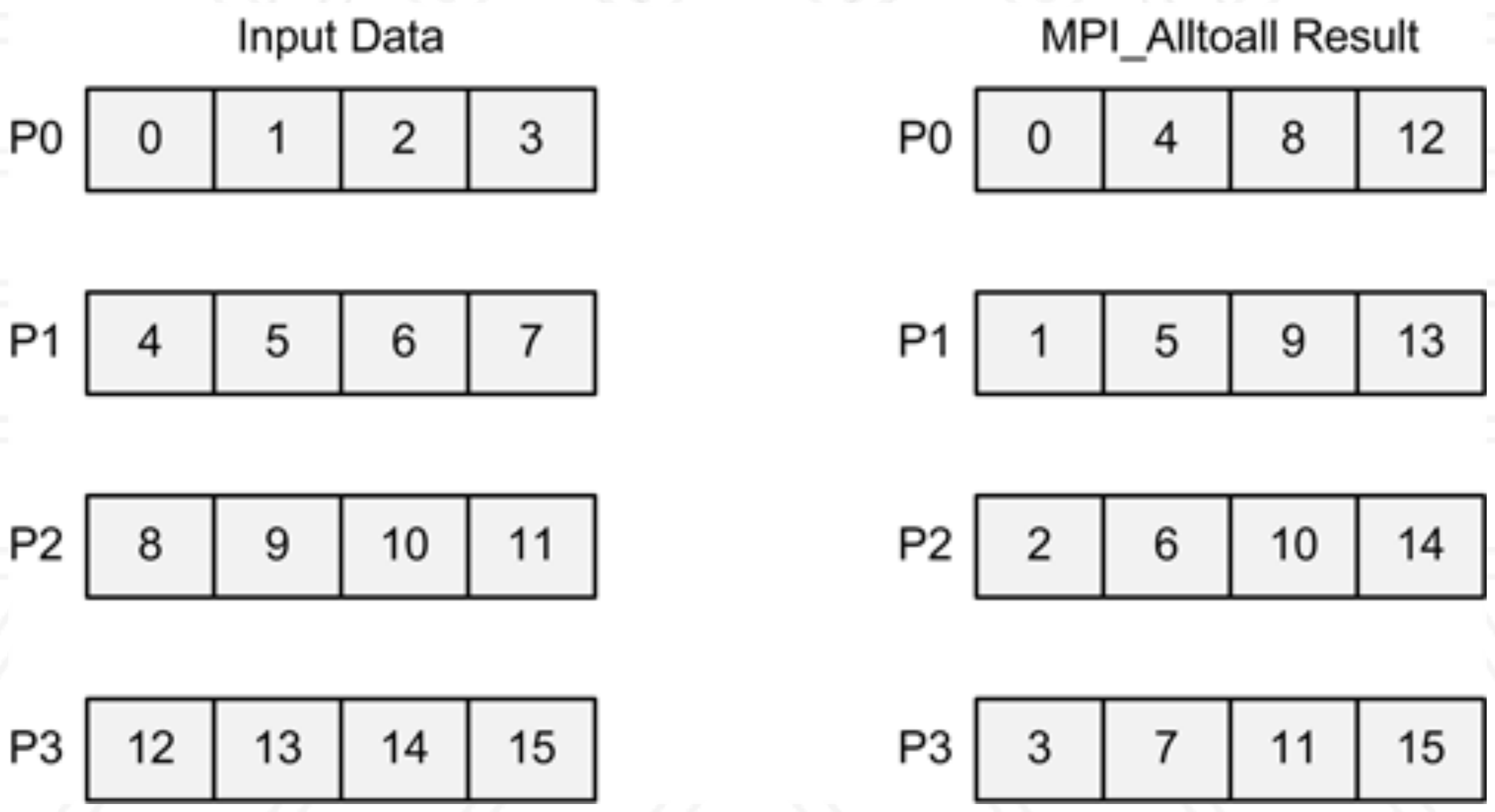
Parallelizing reduction

- Naive algorithm: every process sends to the root
- Spanning tree: organize processes in a k-ary tree
- Start at leaves and send to parents
- Intermediate nodes wait to receive data from all their children
- Number of phases: $\log_k p$

MPI Reduction Algorithms: https://hcl.ucd.ie/system/files/TJS-Hasanov-2016.pdf

All-to-all

- Each process sends a distinct message to every other process
- Naive algorithm: every process sends the data pair-wise to all other processes

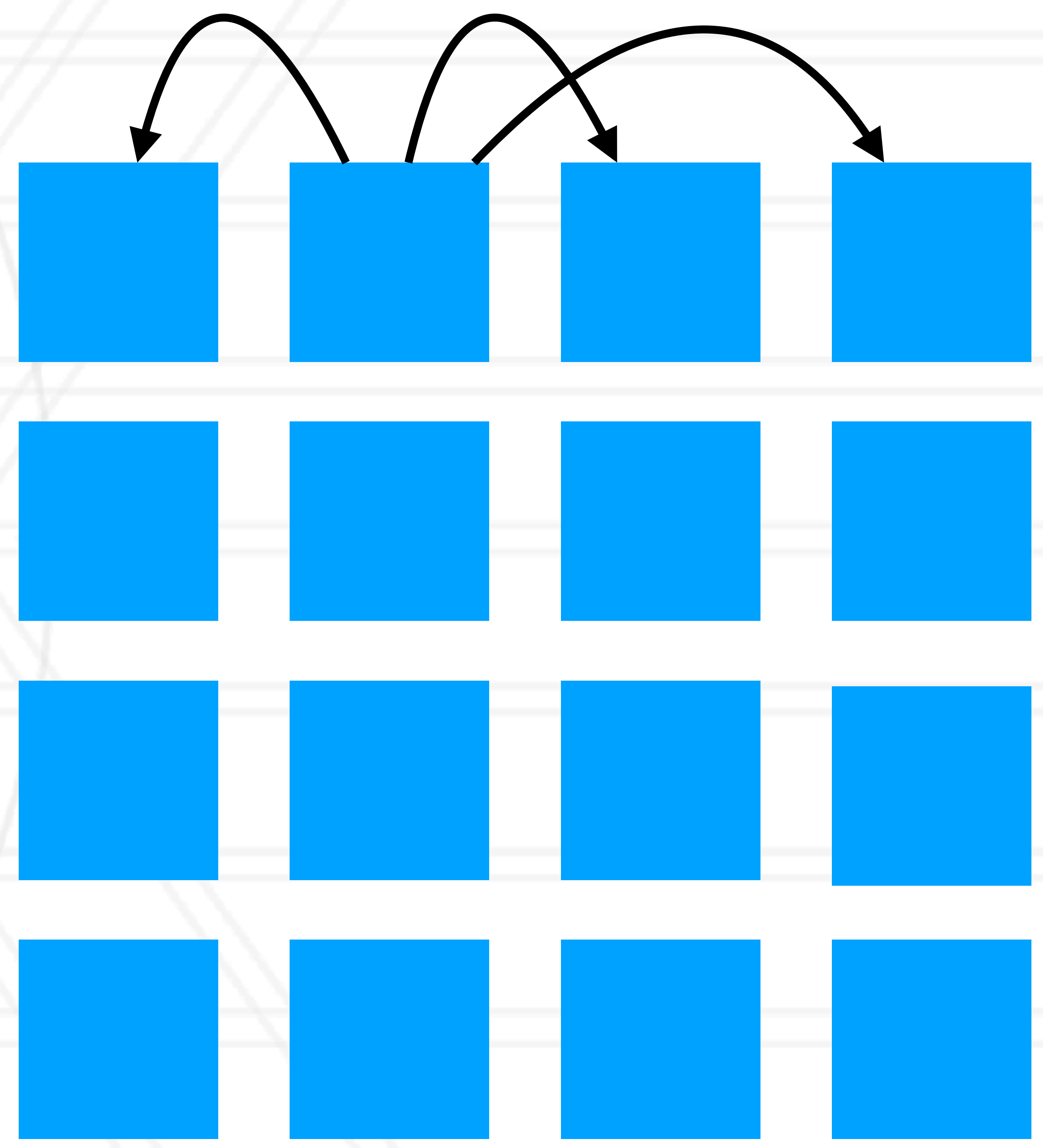


Input Data				
P0	0	1	2	3
P1	4	5	6	7
P2	8	9	10	11
P3	12	13	14	15

MPI_Alltoall Result				
P0	0	4	8	12
P1	1	5	9	13
P2	2	6	10	14
P3	3	7	11	15

https://www.codeproject.com/Articles/896437/A-Gentle-Introduction-to-the-Message-Passing-Inter

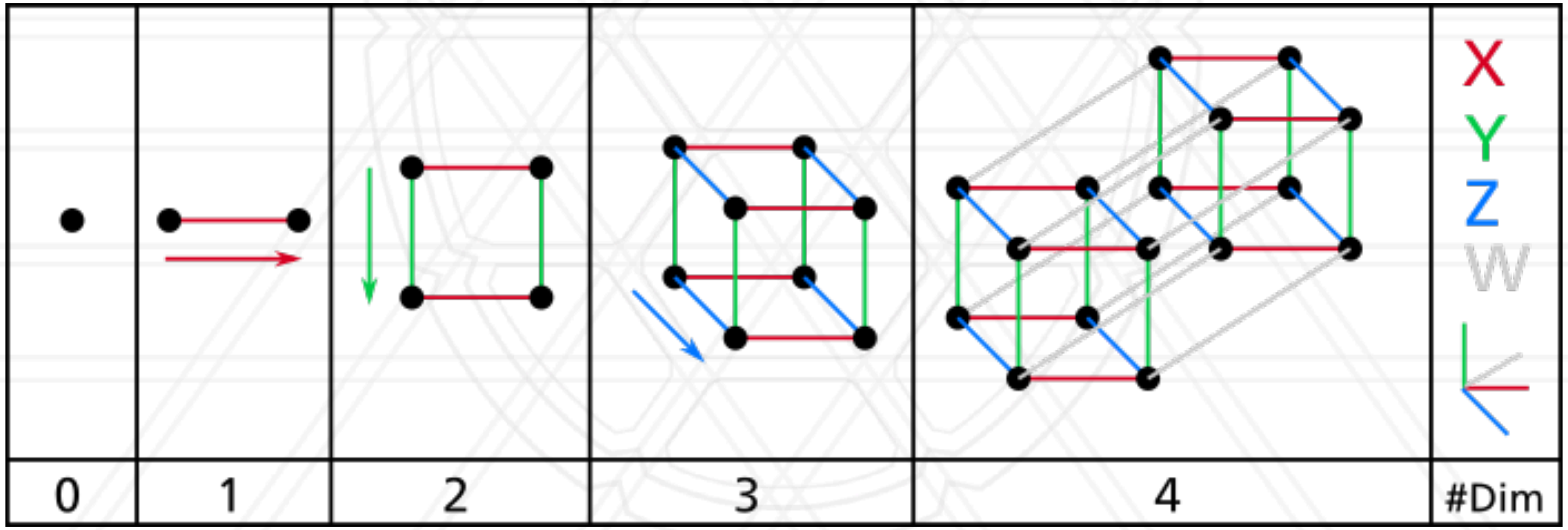
Virtual topology: 2D mesh



- Phase 1: every process sends to its row neighbors
- Phase 2: every process sends to column neighbors

Virtual topology: hypercube

- Hypercube is an n-dimensional analog of a square (n=2) and cube (n=3)
- Special case of k-ary d-dimensional mesh



https://en.wikipedia.org/wiki/Hypercube

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu