Accelerating BPMax for RNA-RNA Interactions: Using Polyhedral Compilation

HiCOMB Workshop 2021

Chiranjeb Mondal (Chiranjeb.mondal@colostate.edu)

Sanjay Rajopadhye (Sanjay.rajopadhye@colostate.edu)

Computer Science Department, Colorado State University

Colorado State University

Problem Statement

- **Motivation:** RNA RNA interactions (RRI) play important role in various biological process such as gene transcription
  - Known to play a critical role in diseases such as Cancer and Alzheimer's
    - Necessitating efficient computational tools

- **Problem:** We choose one of the simpler RRI - BPMax for optimization on CPU
  - BPMax - High complexity ($\Theta(N^3M^3)$ in time and $\Theta(N^2M^2)$ in space) makes it both essential and a challenge to parallelize
  - Long-term goal: Build efficient libraries for similar RRI algorithms.

- **Typical Approach:** RRI programs are developed and optimized by hand
  - Prone to human error, and costly to develop and maintain

- **Our approach:** Use a polyhedral compilation tool - ***ALPHAZ***, that takes user-specified mapping directives and automatically generate optimized code in C

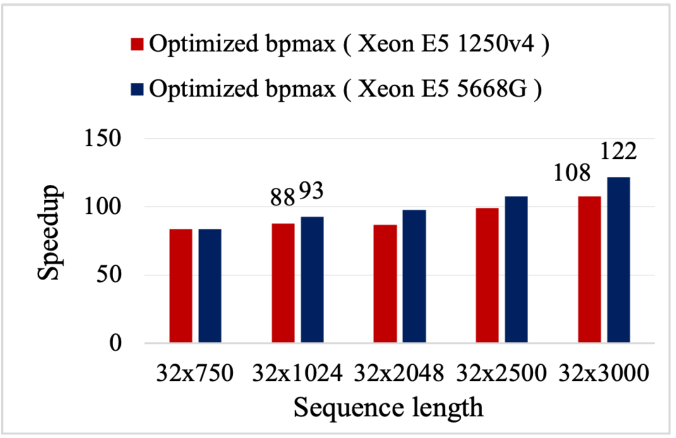
Contribution

- **Speedup:**
  - 100x over the original program
  - 170x for the most compute intensive part [$\Theta(N^3M^3)$]
    - 1.5x - 2x improvement over a similar kernel optimized previously

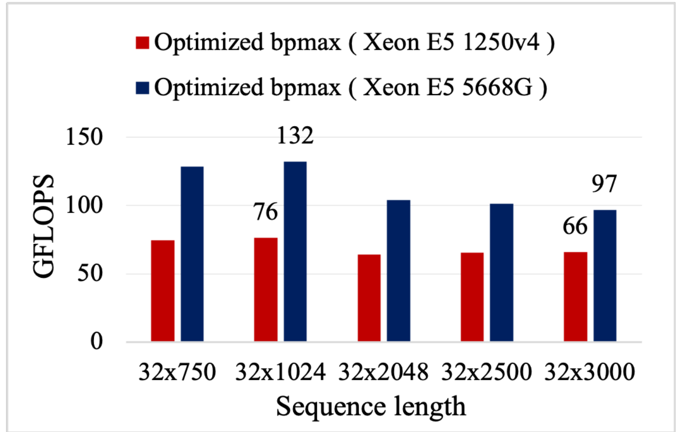
- **Performance on Xeon E5 1250v4:**
  - 76 GFLOPS single-precision (for entire program)
  - 117 GFLOPS (for most compute-intensive part)
    - One-third of the machine peak
  - Scales up with more compute power

- **Lines of Code Metric**
  - Original un-optimized hand-written version - 140
  - Final optimized version - 1400

Speedup over base program



Single-precision performance



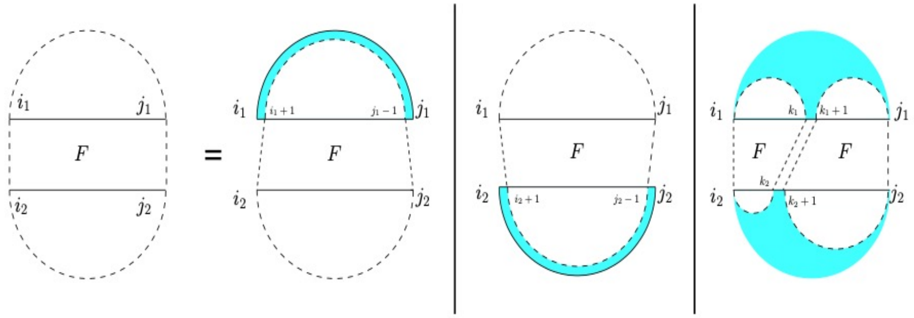
Implementation	*LOC*	a	b
BPMax base	140	140	NA
Double max-plus(coarse/fine)	150	None	3
BPMax coarse/fine/ hybrid	1200	None	30
BPMax hybrid with tiled	1400	<5	7

a - Hand written code
b - Macro replacement/Macro comment out

BPMax Overview

BPMax Overview

- BPMax computes a dynamic programming table to capture interactions between two RNAs
- It uses weighted base-pair counting for base-pair maximization
- Input
  - Two sequences of length M and N
- Output
  - A four-dimensional triangular table
    - A triangular collection of triangles



BPMax Cases

$$S_{i,j} = \begin{cases} 0 & j - i < 4 \\ \max\left(S_{i+1,j-1} + \text{score}(i,j),\ \max_{k=i}^{j-1} S_{i,k} + S_{k+1,j}\right) & \text{otherwise.} \end{cases}$$

$$F_{i_1,j_1,i_2,j_2} = \begin{cases} -\infty & j_1 < i_1 \text{ and } j_2 < i_2 \\ S^{(1)}_{i_1,j_1} & i_1 \leq j_1 \text{ and } j_2 < i_2 \\ S^{(2)}_{i_2,j_2} & j_1 < i_1 \text{ and } i_2 \leq j_2 \\ \text{iscore}(i_1, i_2) & i_1 = j_1 \text{ and } i_2 = j_2 \\ \max\,[\, F_{i_1+1,j_1-1,i_2,j_2} + \text{score}(i_1,j_1), & \\ \quad F_{i_1,j_1,i_2+1,j_2-1} + \text{score}(i_2,j_2), & \\ \quad H_{i_1,j_1,i_2,j_2}\,] & \text{otherwise,} \end{cases}$$

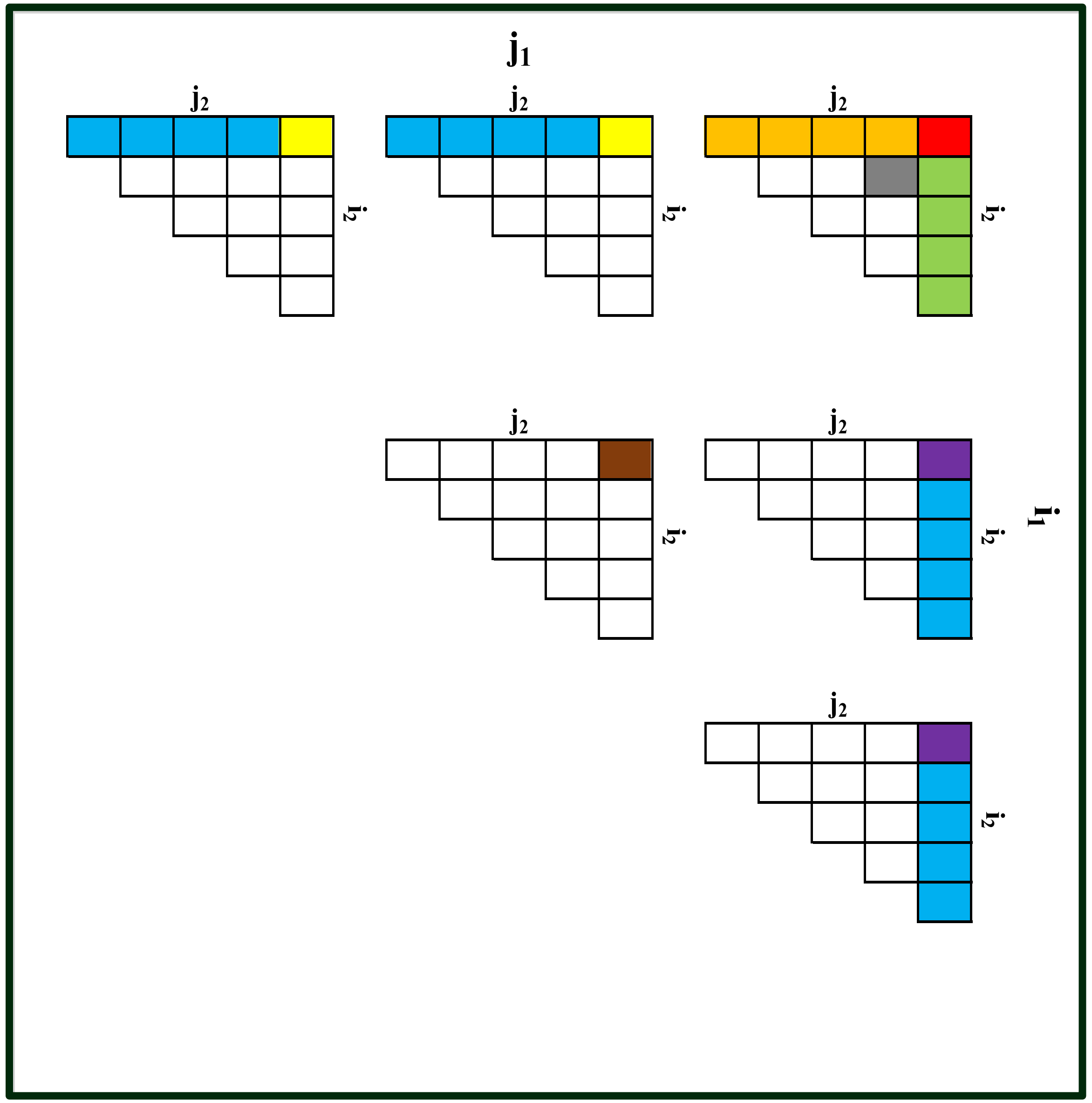
$$H_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1-1}^{j_1} \max_{k_2=i_2-1}^{j_2} (F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}).$$

Note that H is equivalent to

$$H_{i_1,j_1,i_2,j_2} = \max \begin{pmatrix} S^{(1)}(i_1,j_1) + S^{(2)}(i_2,j_2), \\ \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} S^{(2)}(i_2,k_2) + F_{i_1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} F_{i_1,j_1,i_2,k_2} + S^{(2)}(k_2+1,j_2), \\ \max_{k_1=i_1}^{j_1-1} S^{(1)}(i_1,k_1) + F_{k_1+1,j_1,i_2,j_2}, \\ \max_{k_1=i_1}^{j_1-1} F_{i_1,k_1,i_2,j_2} + S^{(1)}(k_1+1,j_1) \end{pmatrix}.$$

BPMax Recurrence

BPMax Dependency Overview



$$S_{i,j} = \begin{cases} 0 & j - i < 4 \\ \max\left(S_{i+1,j-1} + \text{score}(i,j),\ \max_{k=i}^{j-1} S_{i,k} + S_{k+1,j}\right) & \text{otherwise.} \end{cases}$$

$$F_{i_1,j_1,i_2,j_2} = \begin{cases} -\infty & j_1 < i_1 \text{ and } j_2 < i_2 \\ S^{(1)}_{i_1,j_1} & i_1 \le j_1 \text{ and } j_2 < i_2 \\ S^{(2)}_{i_2,j_2} & j_1 < i_1 \text{ and } i_2 \le j_2 \\ \text{iscore}(i_1,i_2) & i_1 = j_1 \text{ and } i_2 = j_2 \\ \max\left[\, F_{i_1+1,j_1-1,i_2,j_2} + \text{score}(i_1,j_1), \right. & \\ \quad F_{i_1,j_1,i_2+1,j_2-1} + \text{score}(i_2,j_2), & \\ \quad \left. H_{i_1,j_1,i_2,j_2} \,\right] & \text{otherwise,} \end{cases}$$

$$H_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1-1}^{j_1} \max_{k_2=i_2-1}^{j_2} (F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}).$$

Note that H is equivalent to

$$H_{i_1,j_1,i_2,j_2} = \max \begin{pmatrix} S^{(1)}(i_1,j_1) + S^{(2)}(i_2,j_2), \\ \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} S^{(2)}(i_2,k_2) + F_{i_1,j_1,k_2+1,j_2}, \\ \max_{k_2=i_2}^{j_2-1} F_{i_1,j_1,i_2,k_2} + S^{(2)}(k_2+1,j_2), \\ \max_{k_1=i_1}^{j_1-1} S^{(1)}(i_1,k_1) + F_{k_1+1,j_1,i_2,j_2}, \\ \max_{k_1=i_1}^{j_1-1} F_{i_1,k_1,i_2,j_2} + S^{(1)}(k_1+1,j_1) \end{pmatrix}.$$

R_0

R_1

R_2

R_3

R_4

Methodology Highlights

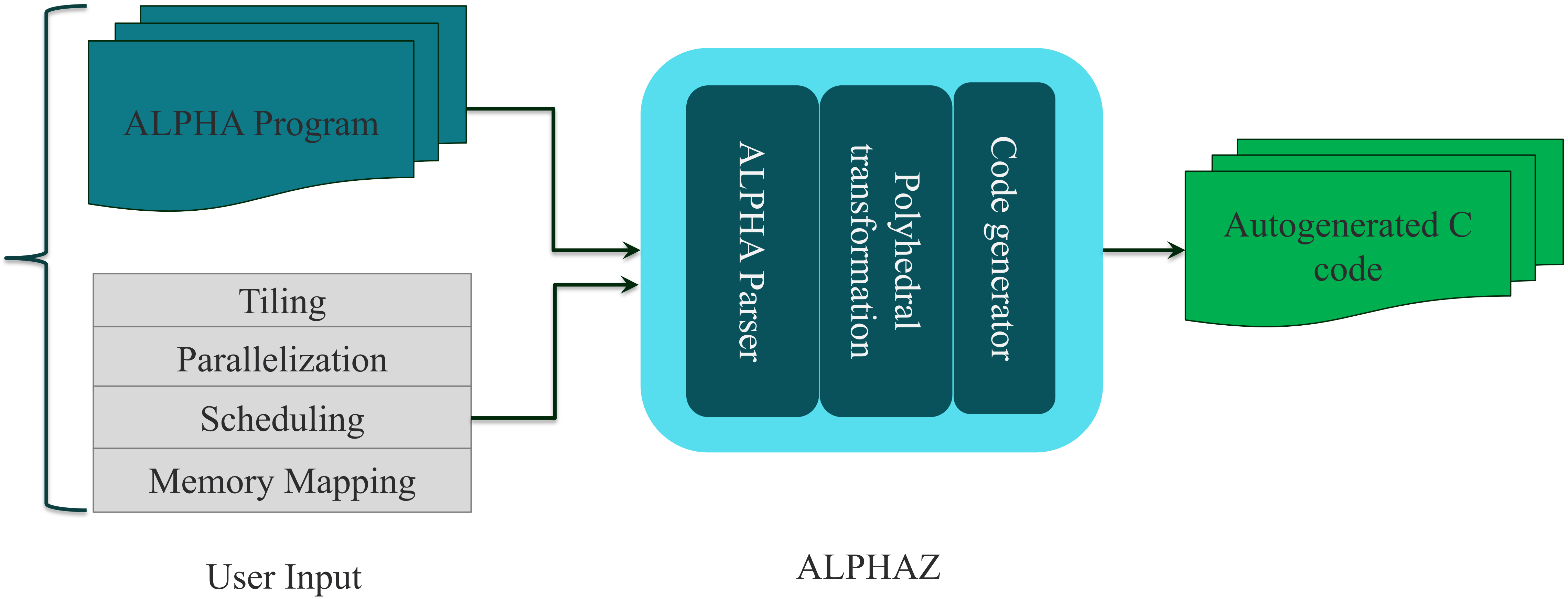
Polyhedral Model & ALPHAZ - Overview

- Polyhedral Model: Mathematical framework for automatic optimization and parallelization of affine programs

- ALPHA: Equational polyhedral programming language
  - Affine dependence
  - Polyhedral domains
  - Reductions with associative commutative operators

- ALPHAZ: Tool that allows the user to explore code generation of ALPHA programs using various-
  - Schedules
  - Memory-maps
  - Parallelization approaches
  - Tiling

Code Generation Methodology



- Typical Polyhedral code generator
  - No user intervention beside input programs(c sources)
  - Not always optimal
- ALPHAZ
  - User specifies polyhedral transformations to the tool
  - Allows larger exploration space

ALPHAZ Transformations Overview

- **Normalize:**
  - Normalizes the program
  - Provides better readability

- **NormalizeReduction**:
  - Transform specified reduction into Normal form
  - Reduce expression is a direct child of the equation

- Key target mapping transformations for schedule code generation
  - **setSpaceTimeMap**: Specifies the schedule and processor allocation
  - **setMemoryMap**: Affine mapping to memory location
  - **setParallel**: Loop parallelization, Parallel dimensions

Algorithm 1 Matrix Multiplication in Alphabets

```
affine MM {N, K, M | (M, N, K) > 0}
input
    float A {i, j | 0 ≤ i < M && 0 ≤ j < K} ;
    float B {i, j | 0 ≤ i < K && 0 ≤ j < N} ;
output
    float C {i, j | 0 ≤ i < M && 0 ≤ j < N};
local
    //local variables
output
    C[i, j] = reduce(+,  [k],  A[i, k] * B[k, j]);
```

Algorithm 2 Matrix Multiplication Command Script

```
// Step − 1 : Parse Alphabet
prog=ReadAlphabets("MM.ab");
system = "MM";
outDir="./src";

// Step − 2 : Perform polyhedral transformation
Normalize(prog);
setSpaceTimeMap(prog, system, "C",
                                   "(i, j, k ↦ i, k, j)",
                                   "(i, j ↦ i, −1, j)");
setParallel(prog, system, "", "0" );

// Step − 3 : Generate code
generateWriteC(prog, system, outDir);
generateScheduleC(prog, system, outDir);
```

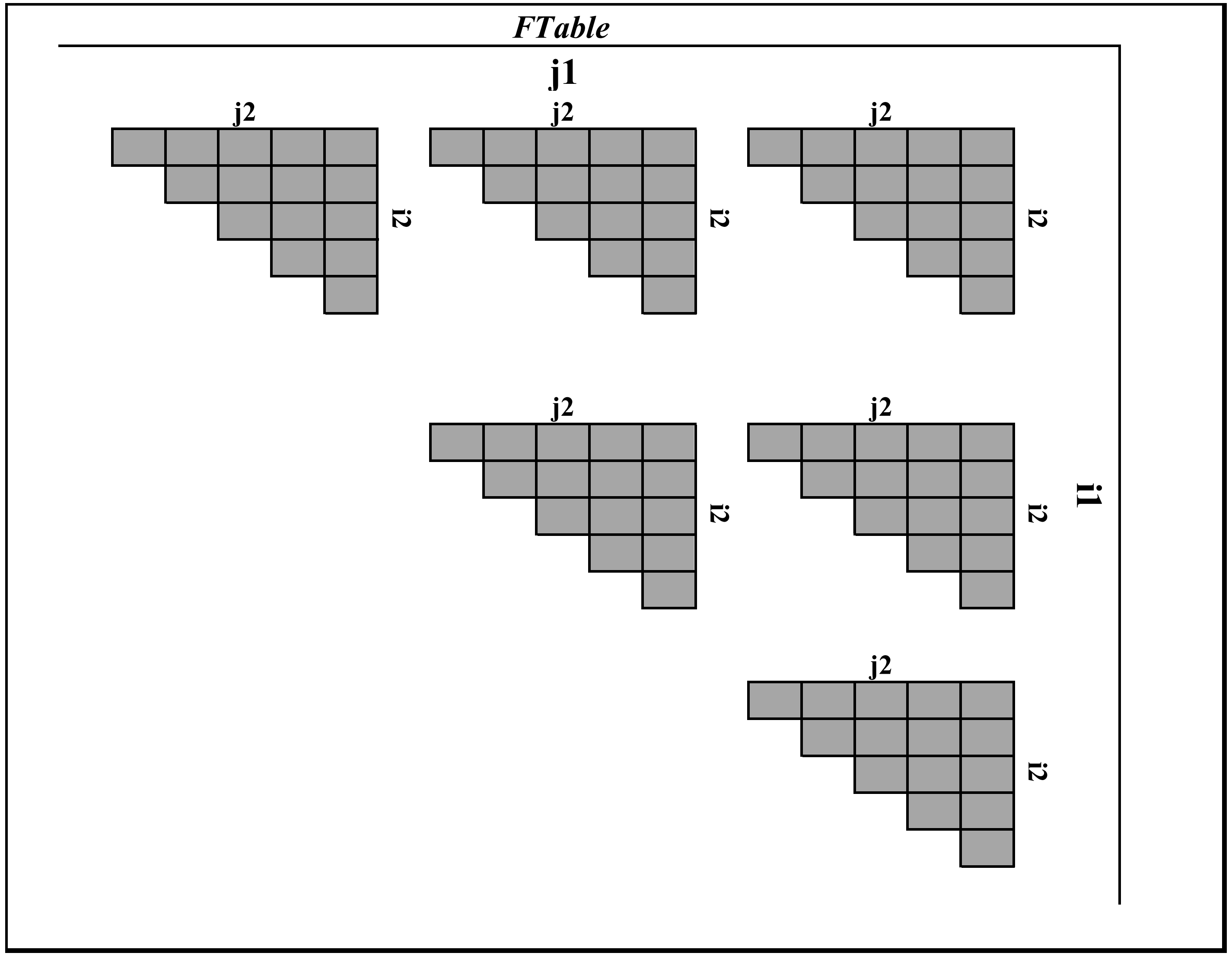
```c
#define S1(i,j,i2) C(i,i2) = 0.0
#define S0(i0,i1,i2) C(i0,i2) = (C(i0,i2))+((A(i0,i1))
 *(B(i1,i2)))
{
    int c1,c2,c3;
    #pragma omp parallel for private(c2,c3)
    for(c1=0;c1 <= M-1;c1+=1){
      for(c3=0;c3 <= N-1;c3+=1){
        S1((c1),(-1),(c3));
      }
        for(c2=0;c2 <= K-1;c2+=1){
            for(c3=0;c3 <= N-1;c3+=1){
                S0((c1),(c2),(c3));
            }
        }
    }
}
```

Optimization Highlights

Double Max-Plus Base Schedule

Double Max-Plus Base Schedule

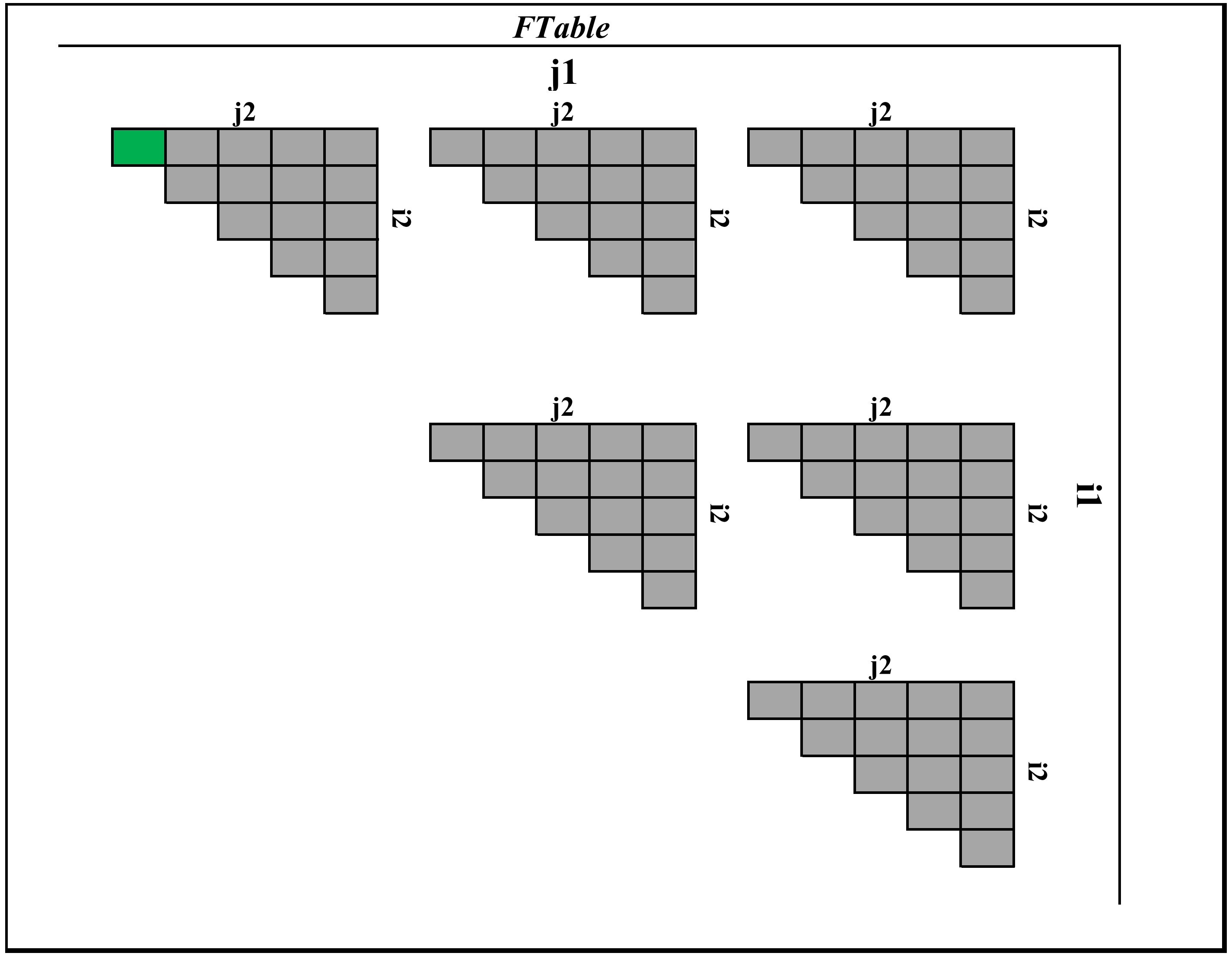
$$F_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}$$



Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Base Schedule

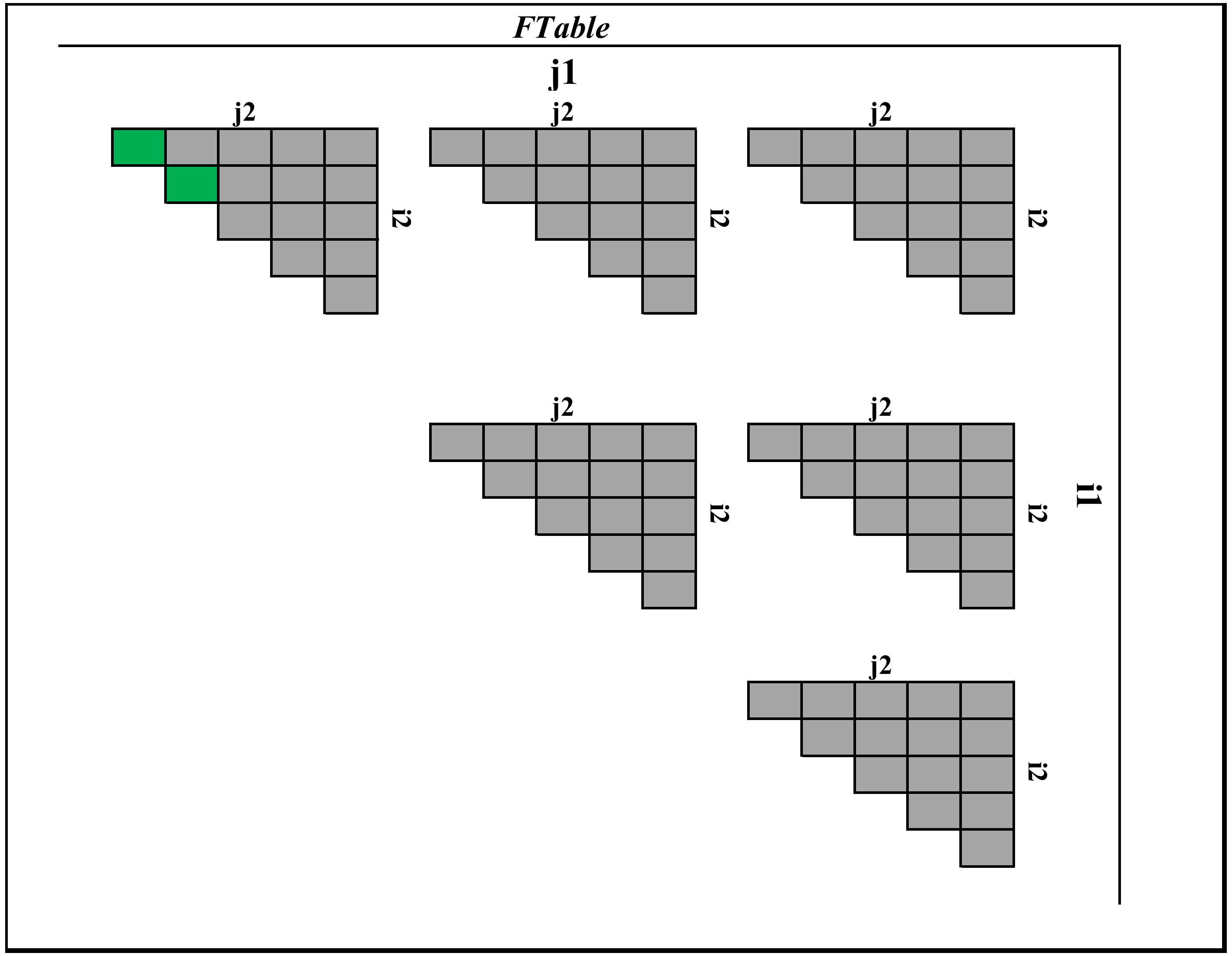
$$F_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}$$



Fill up triangles diagonally

Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Base Schedule

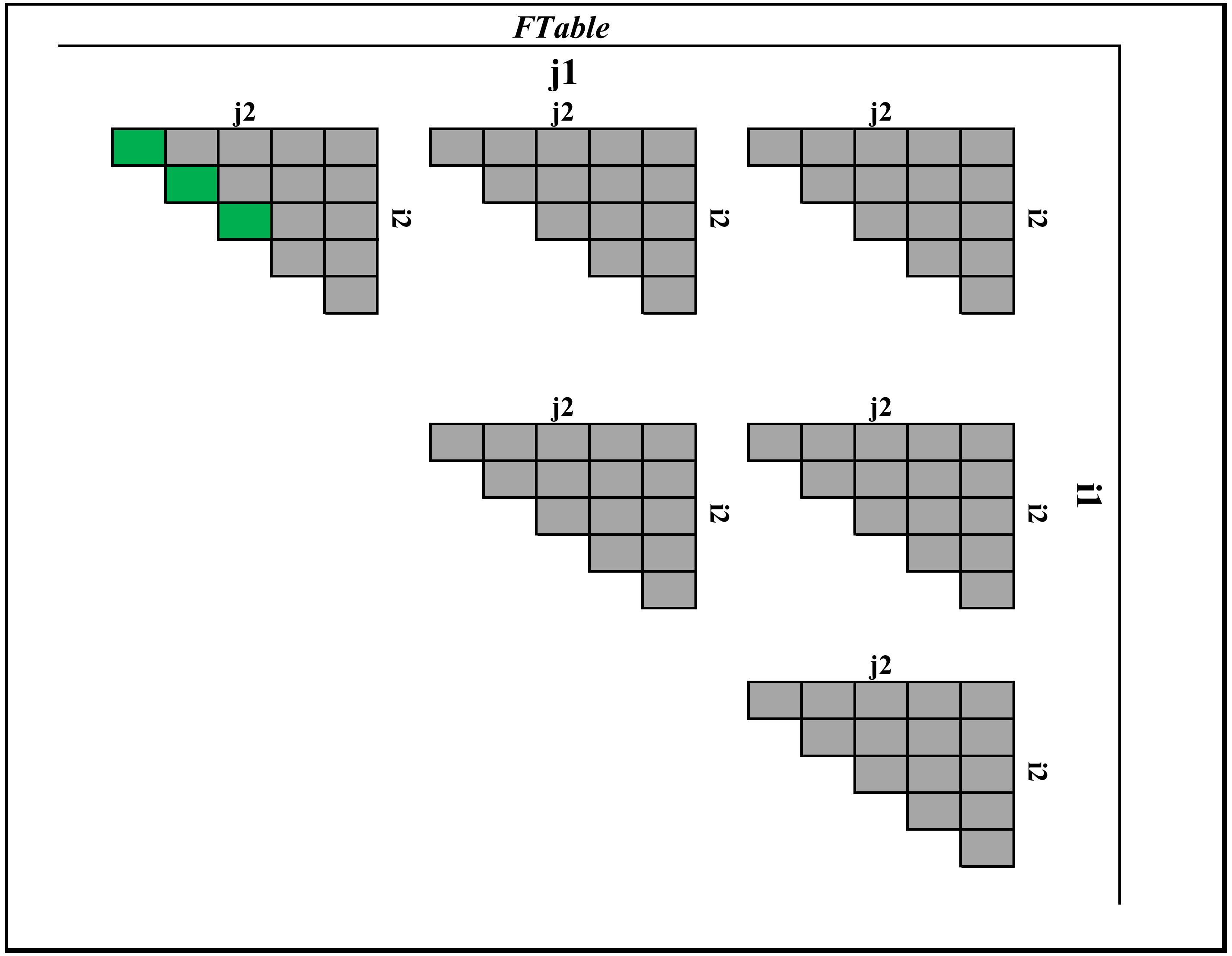


Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Fill up triangles diagonally

Double Max-Plus Computation – Base Schedule

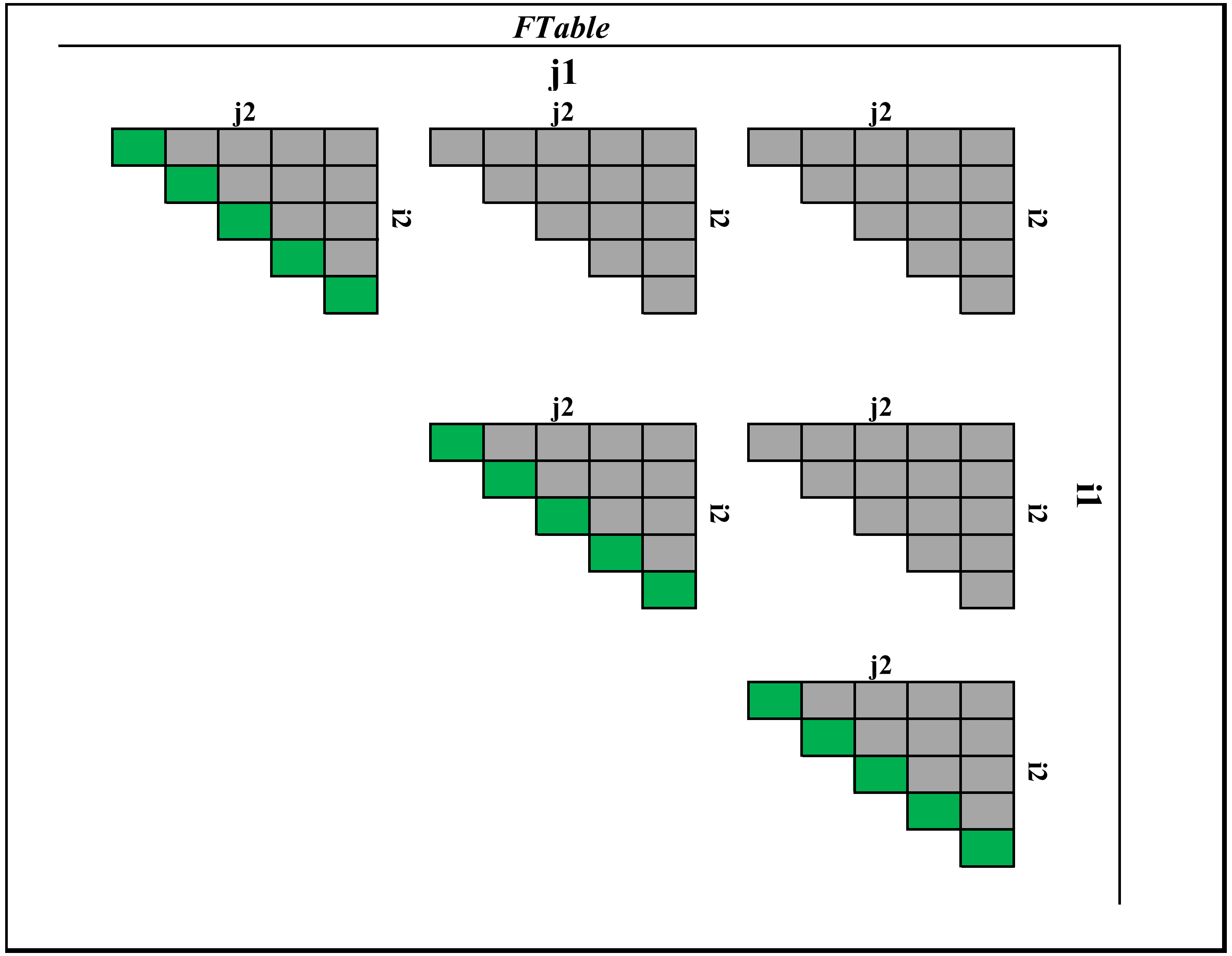
$$F_{i_1,j_1,i_2,j_2} = \max_{k_1=i_1}^{j_1-1} \max_{k_2=i_2}^{j_2-1} F_{i_1,k_1,i_2,k_2} + F_{k_1+1,j_1,k_2+1,j_2}$$



Fill up triangles diagonally

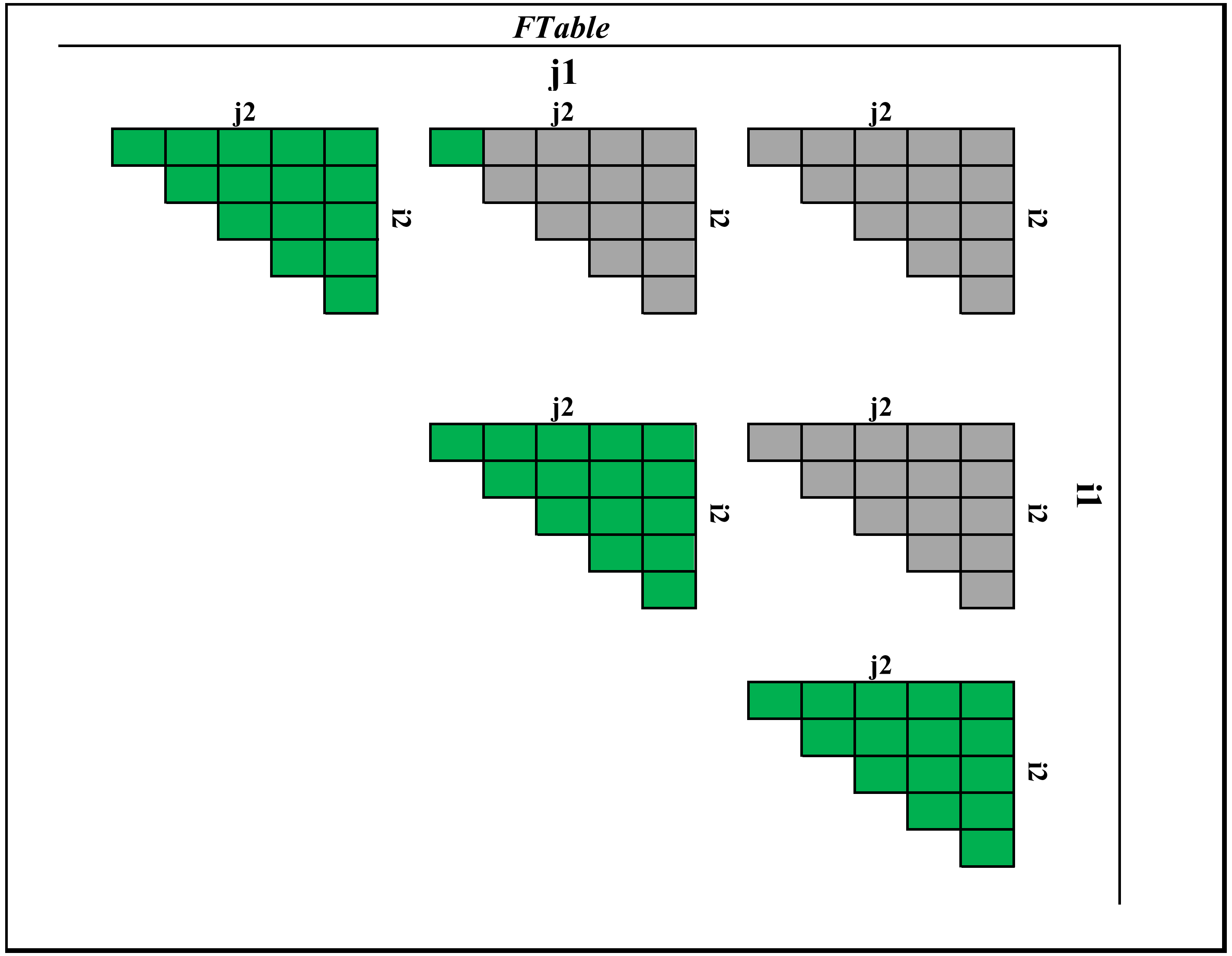
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule



Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

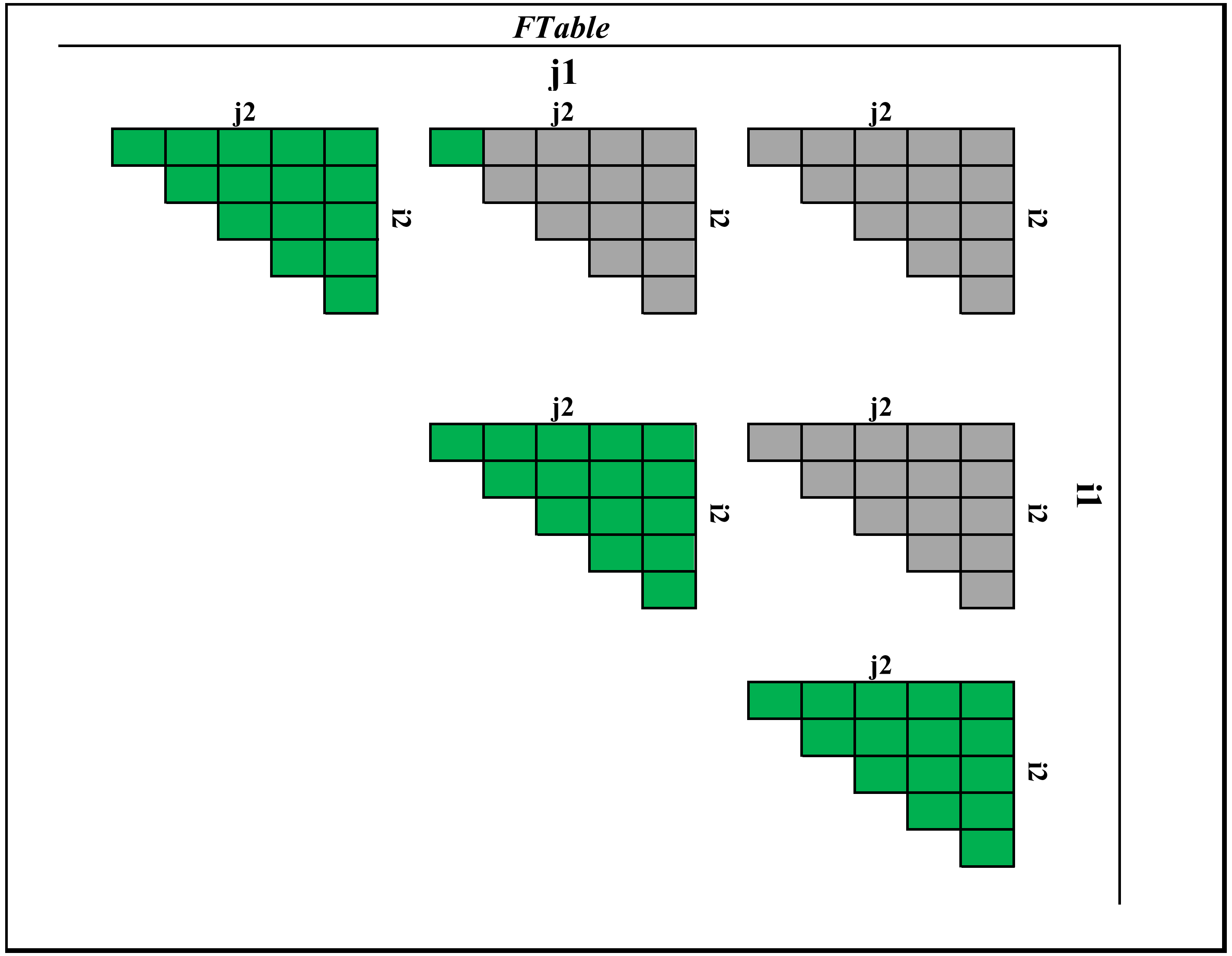
Double Max-Plus Computation – Base Schedule



Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

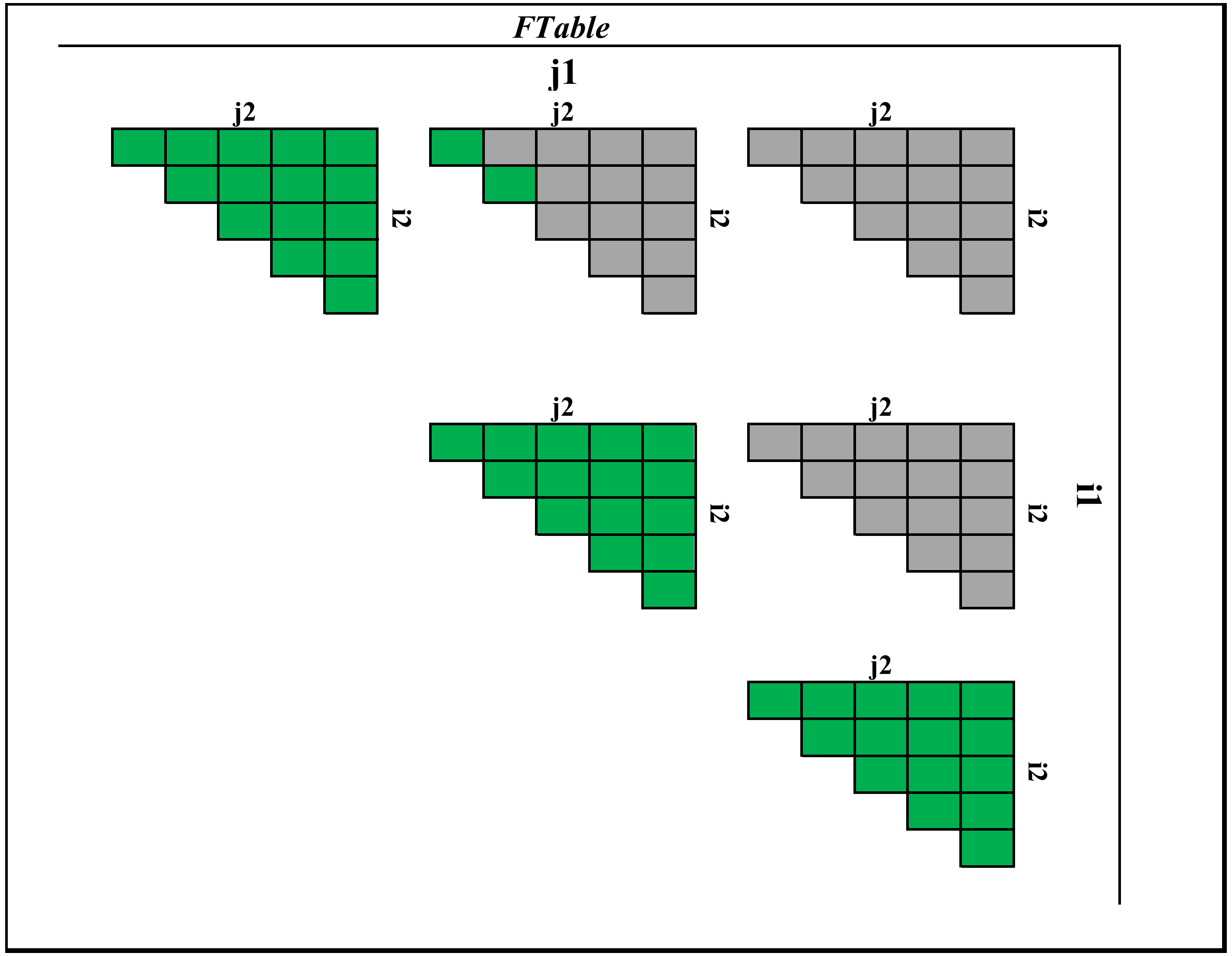
Each inner triangle is also filled diagonally

Double Max-Plus Computation – Base Schedule



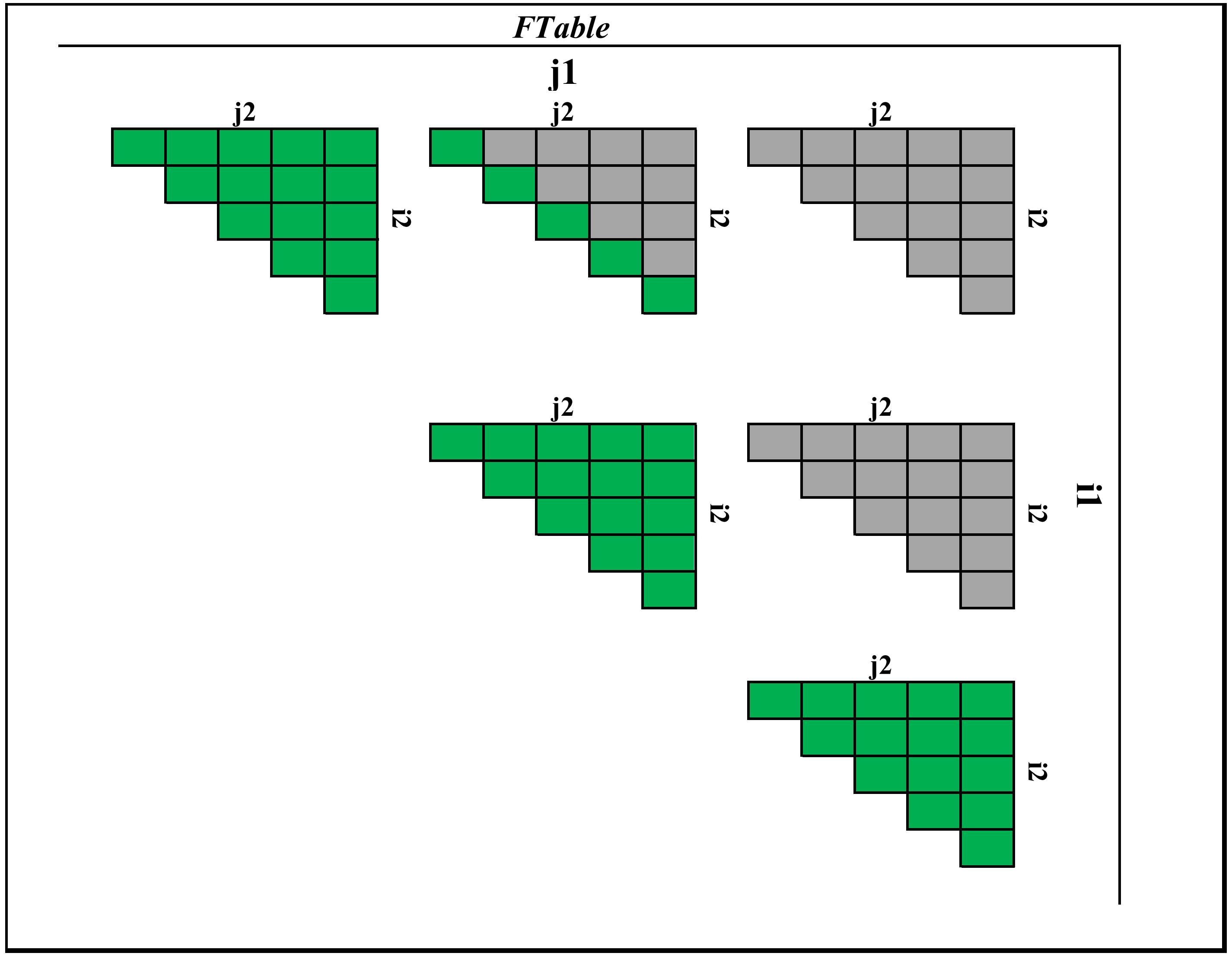
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule



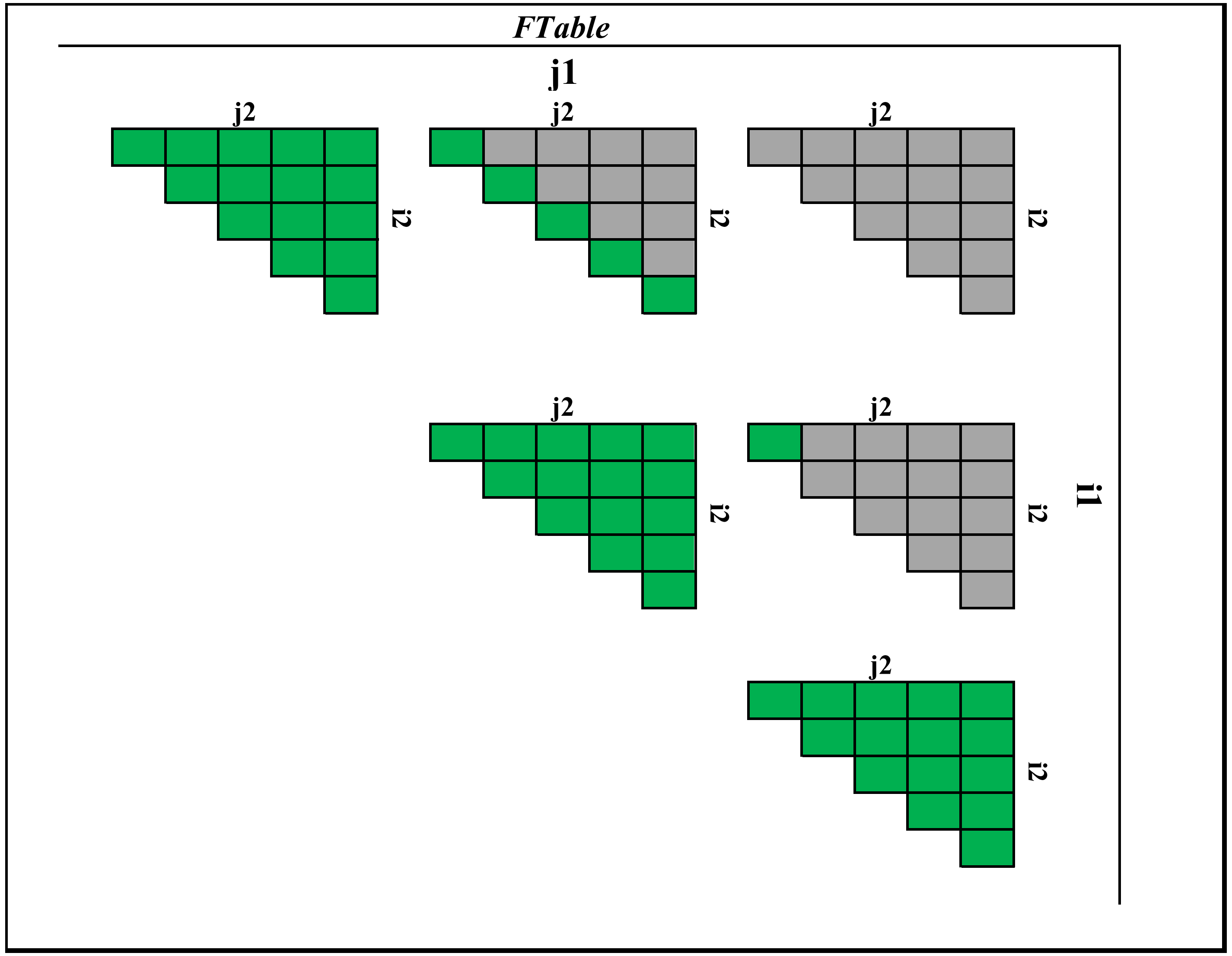
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule



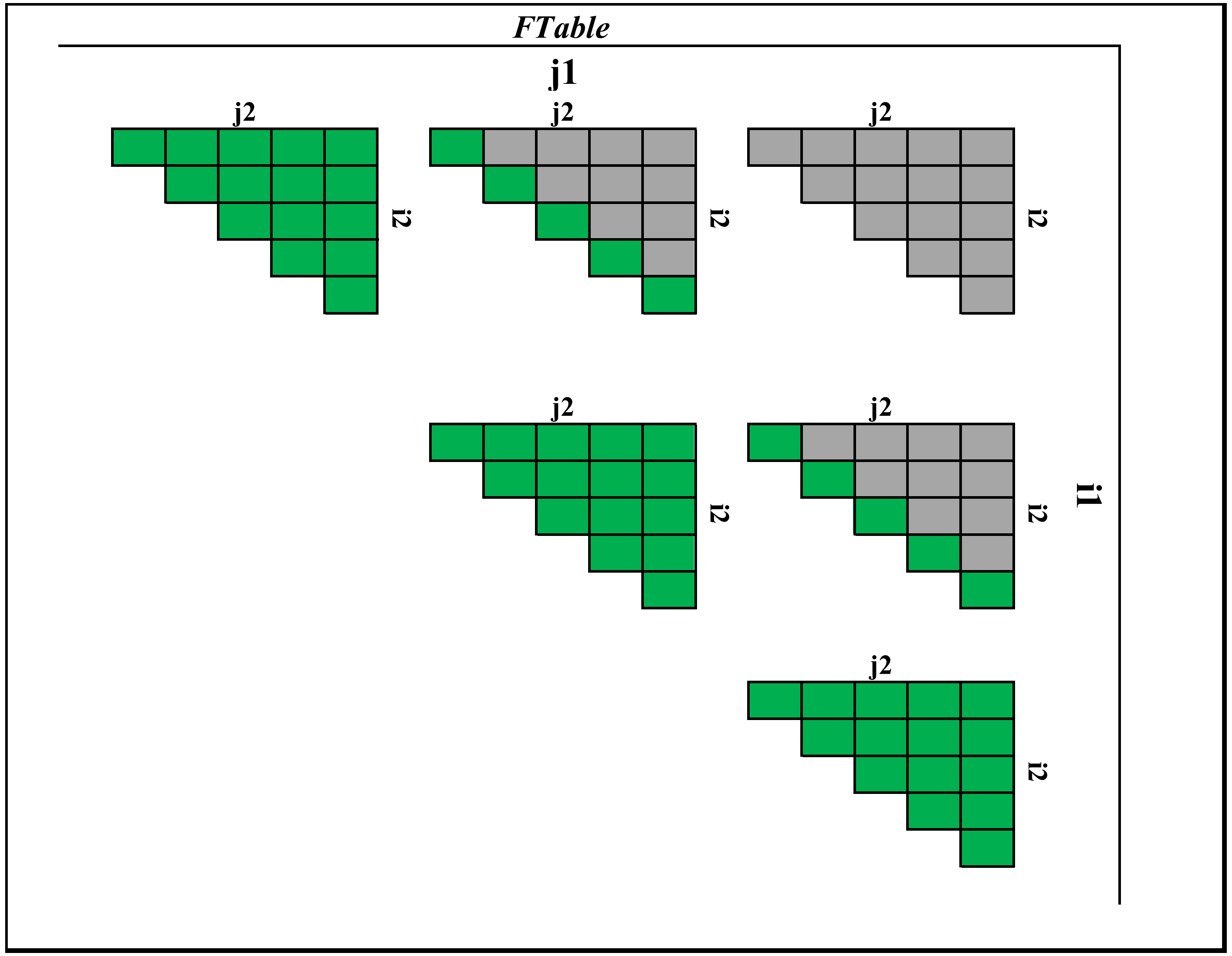
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule



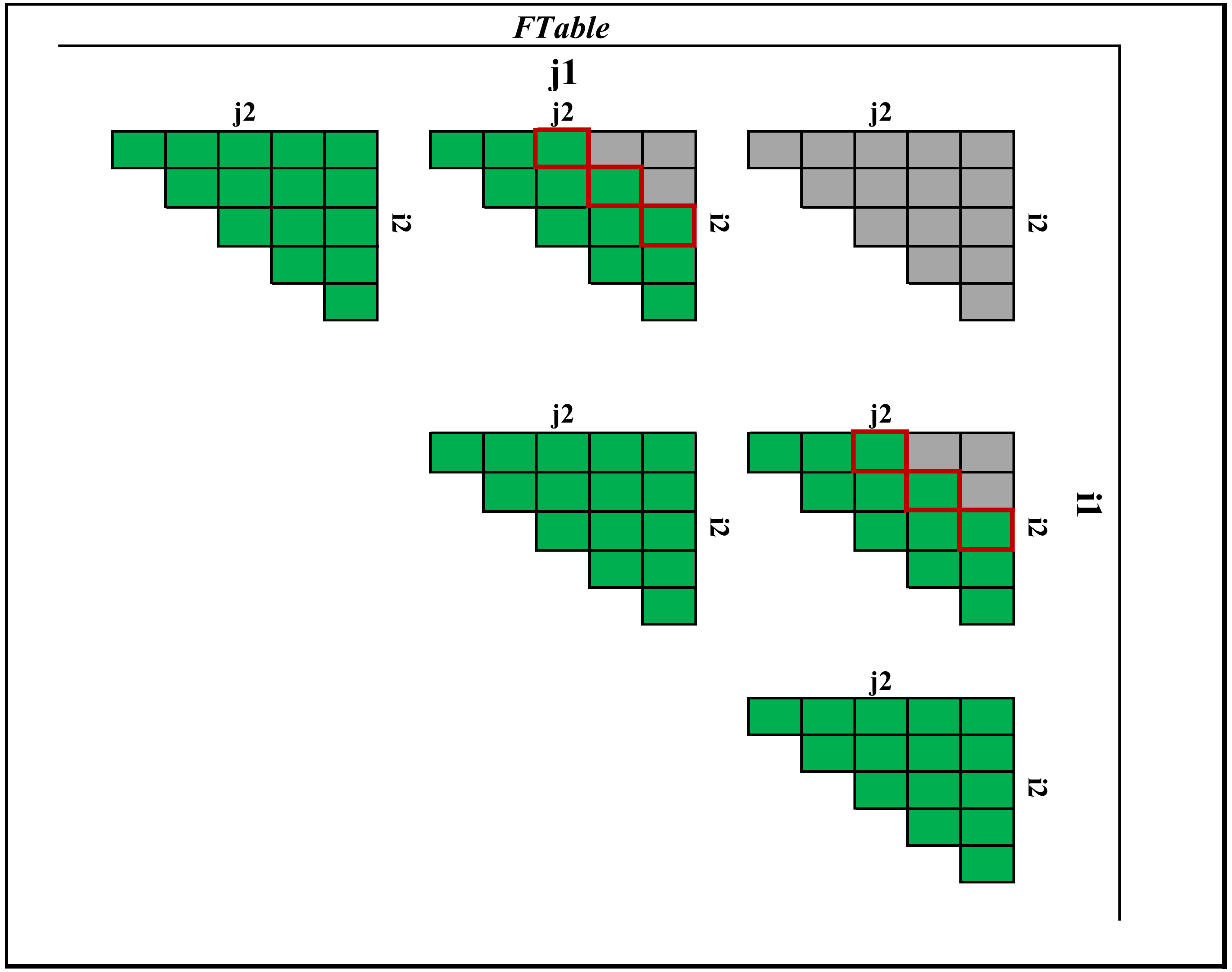
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule



Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

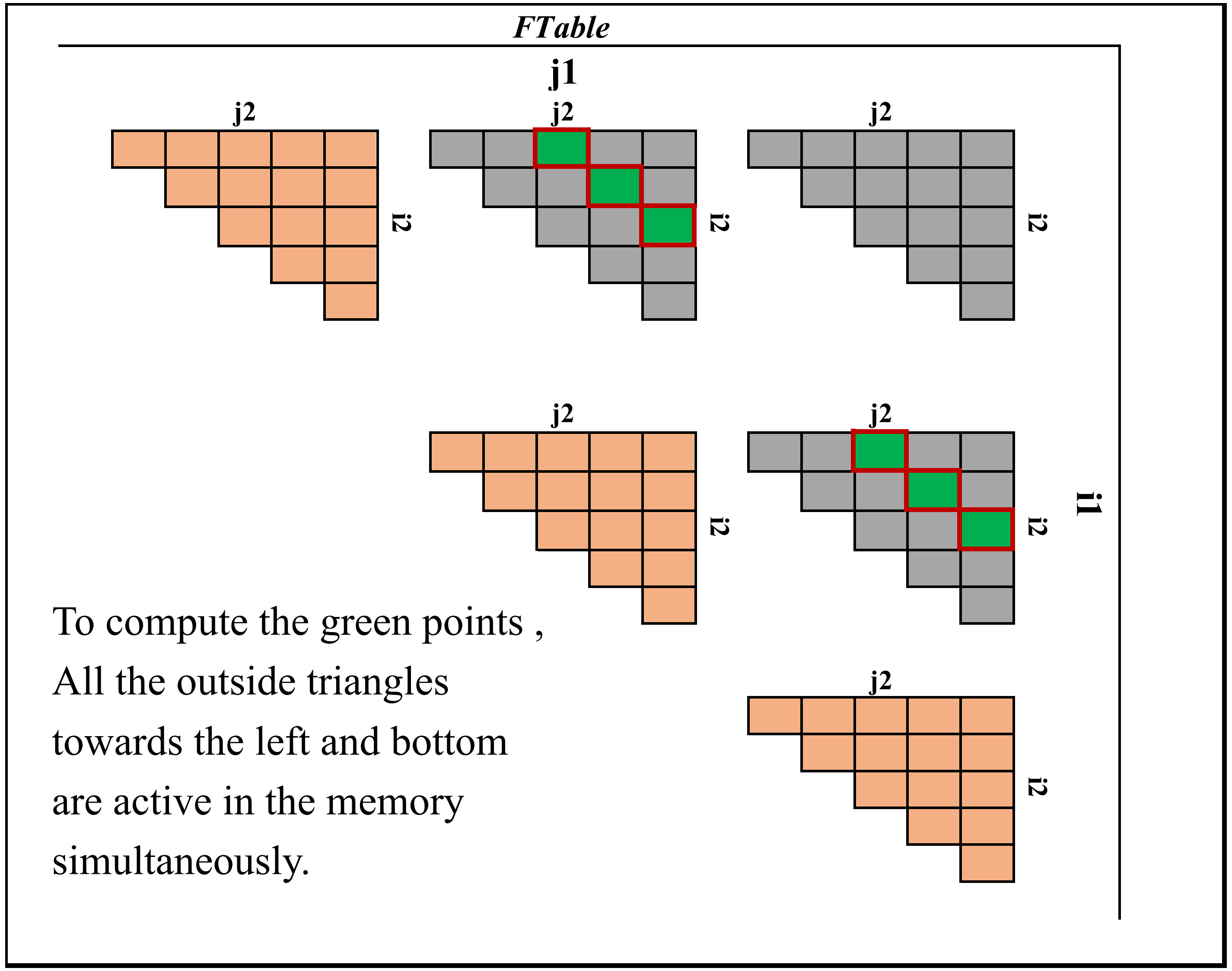
Double Max-Plus Computation – Base Schedule



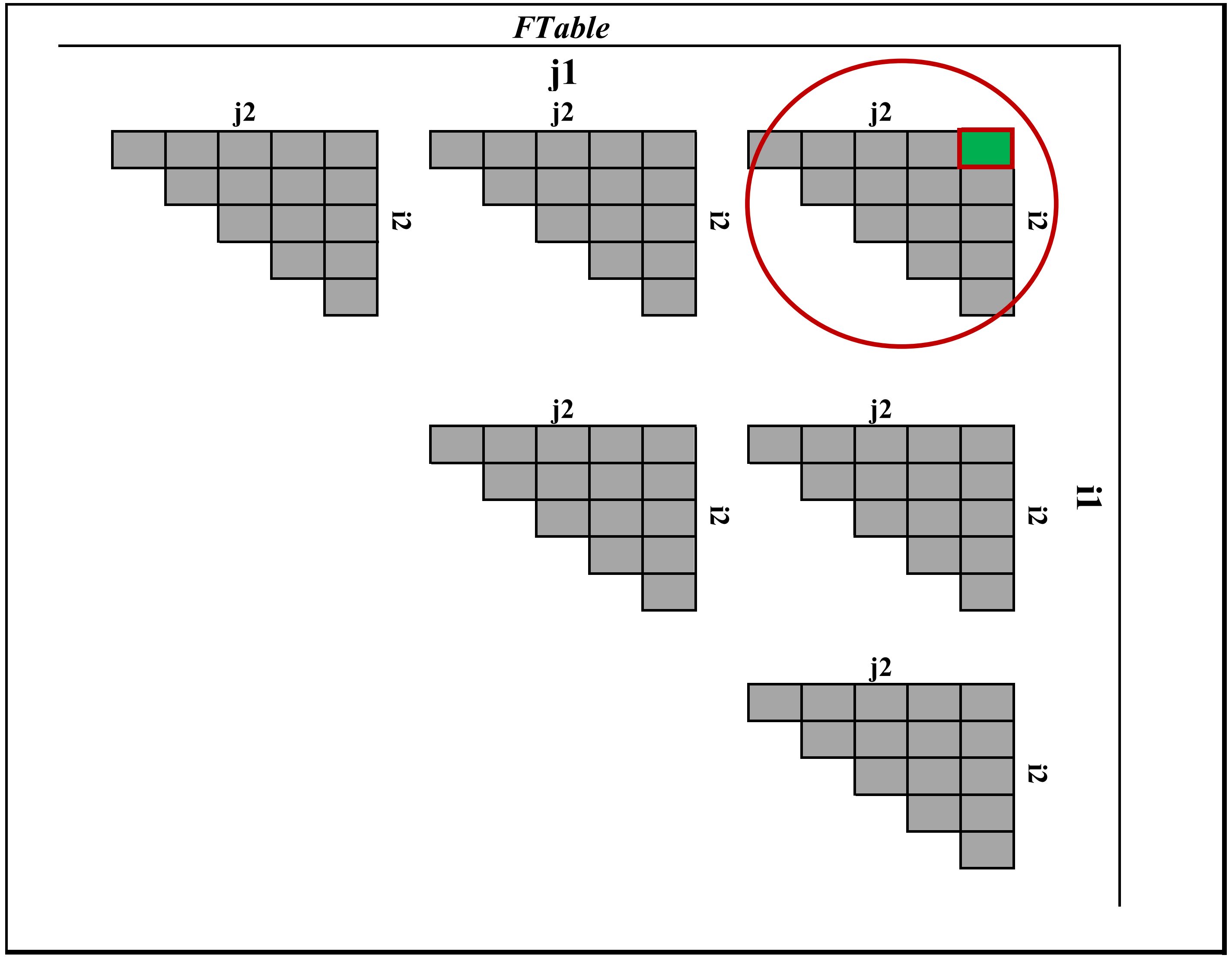
Base Program Schedule $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$

Double Max-Plus Computation – Base Schedule

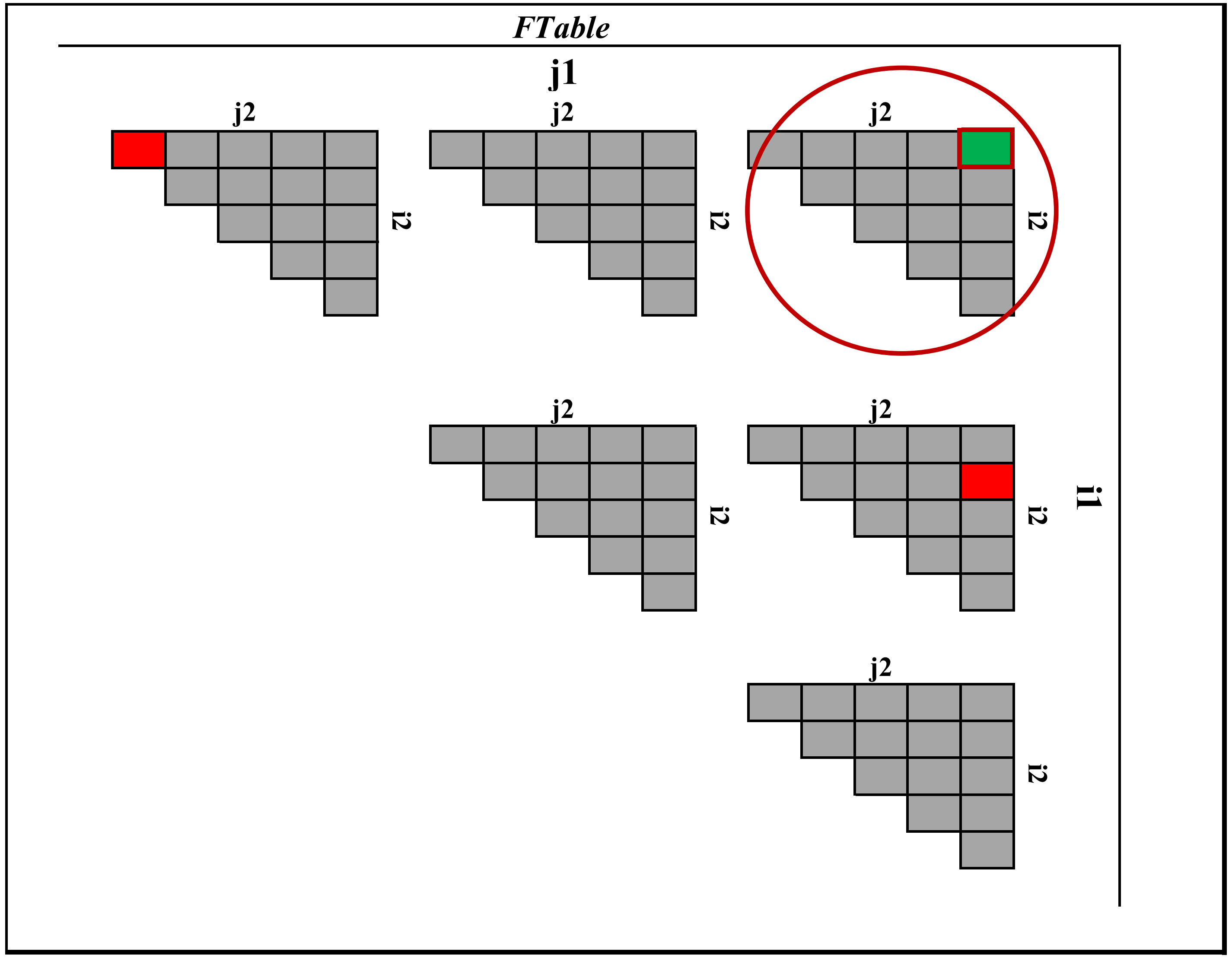
- **Base Double Max-plus Schedule:**
  - $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$
    - Allows maximum parallelization
    - Lot of data movements between different levels of caches
    - Loop carried dependency. No vectorization since k_1 and k_2 loops are inside



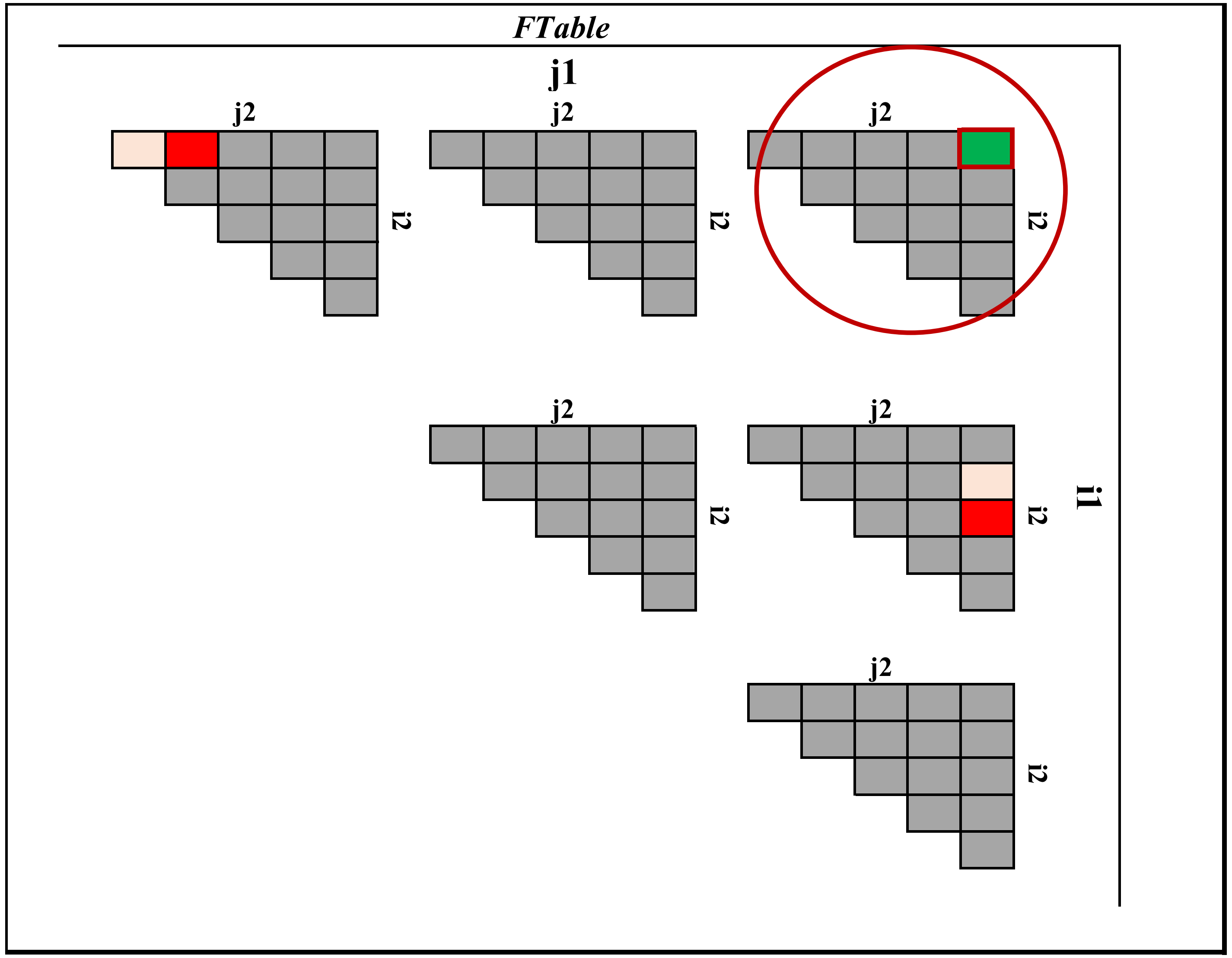
Double Max-Plus Computation



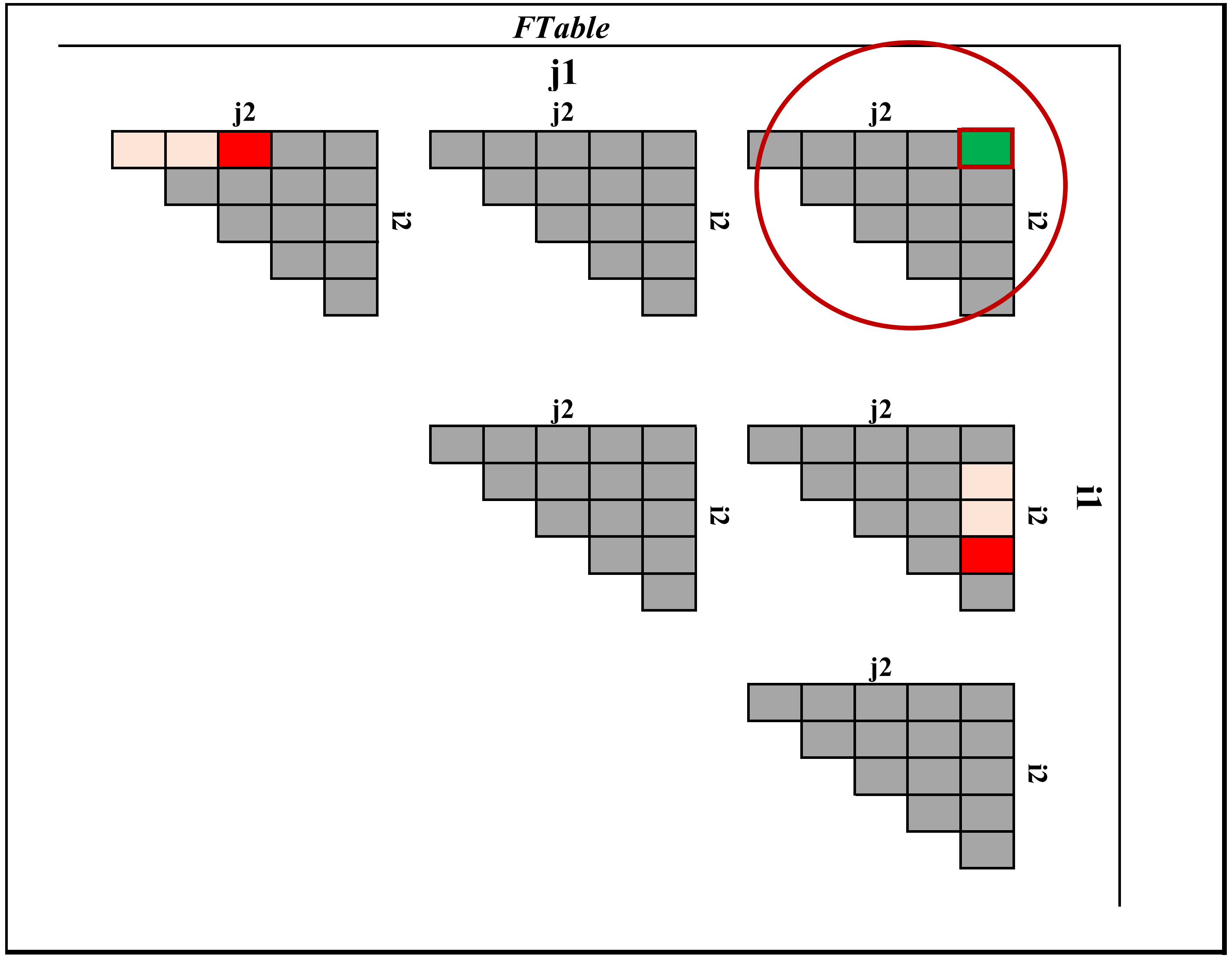
Double Max-Plus Computation



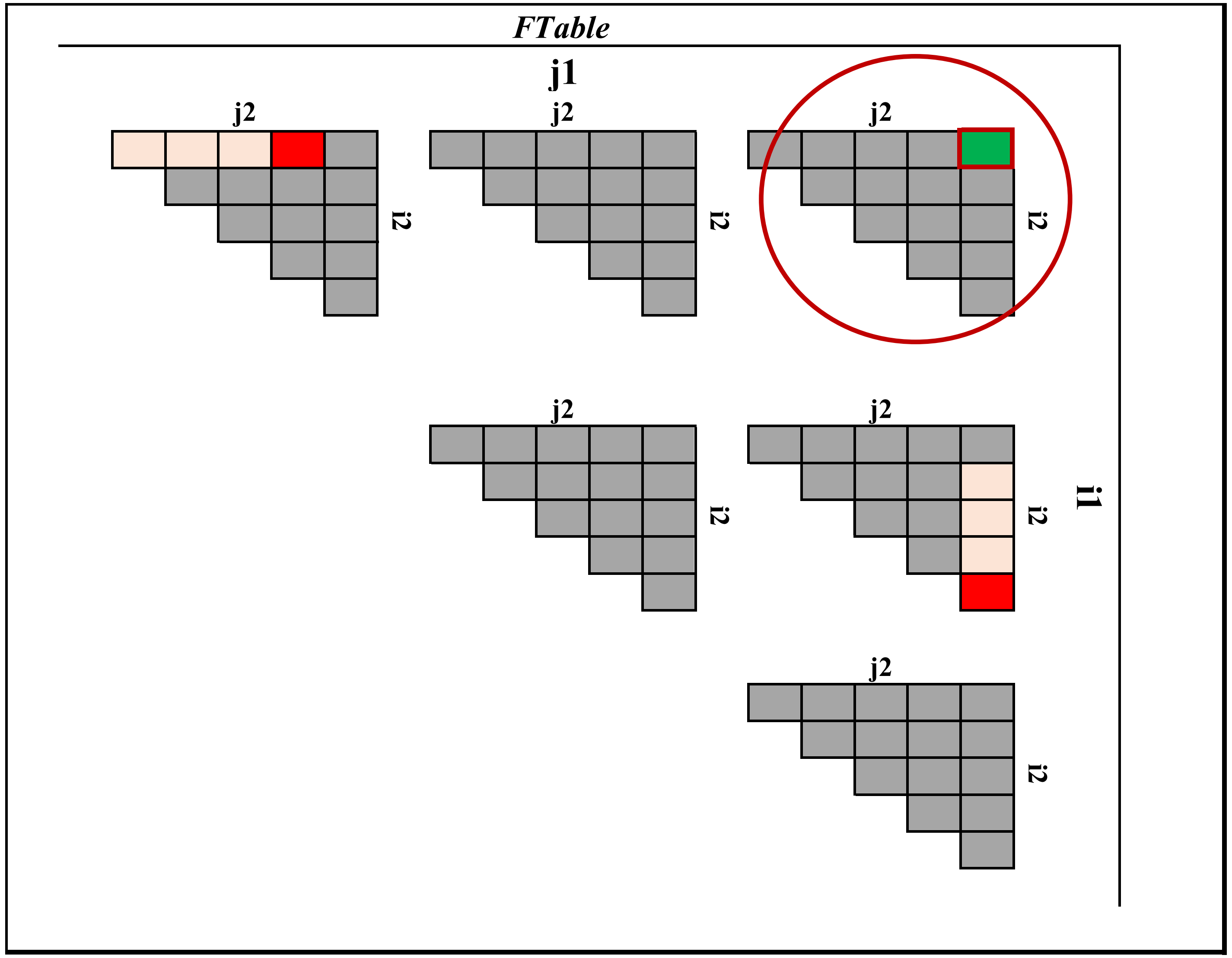
Double Max-Plus Computation



Double Max-Plus Computation

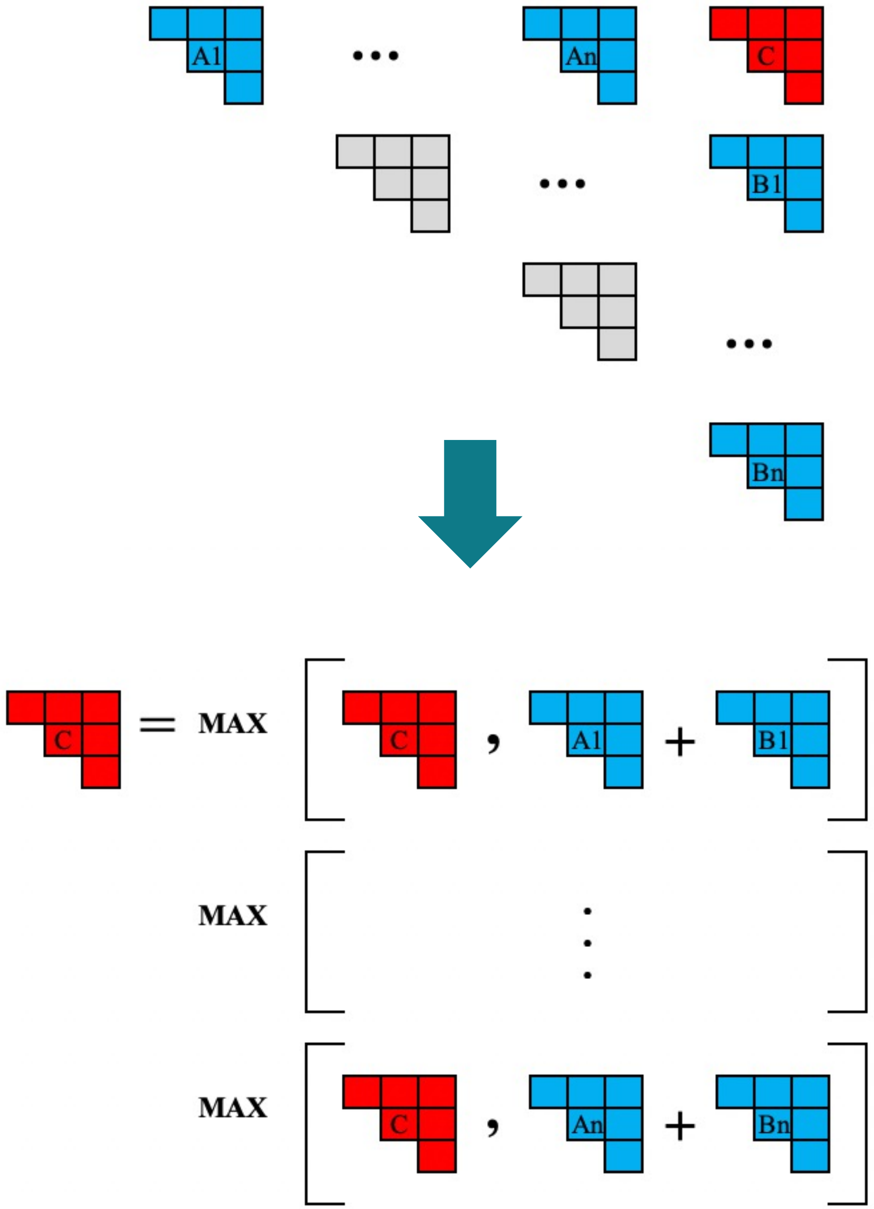


Double Max-Plus Computation



Double Max-plus Decomposition

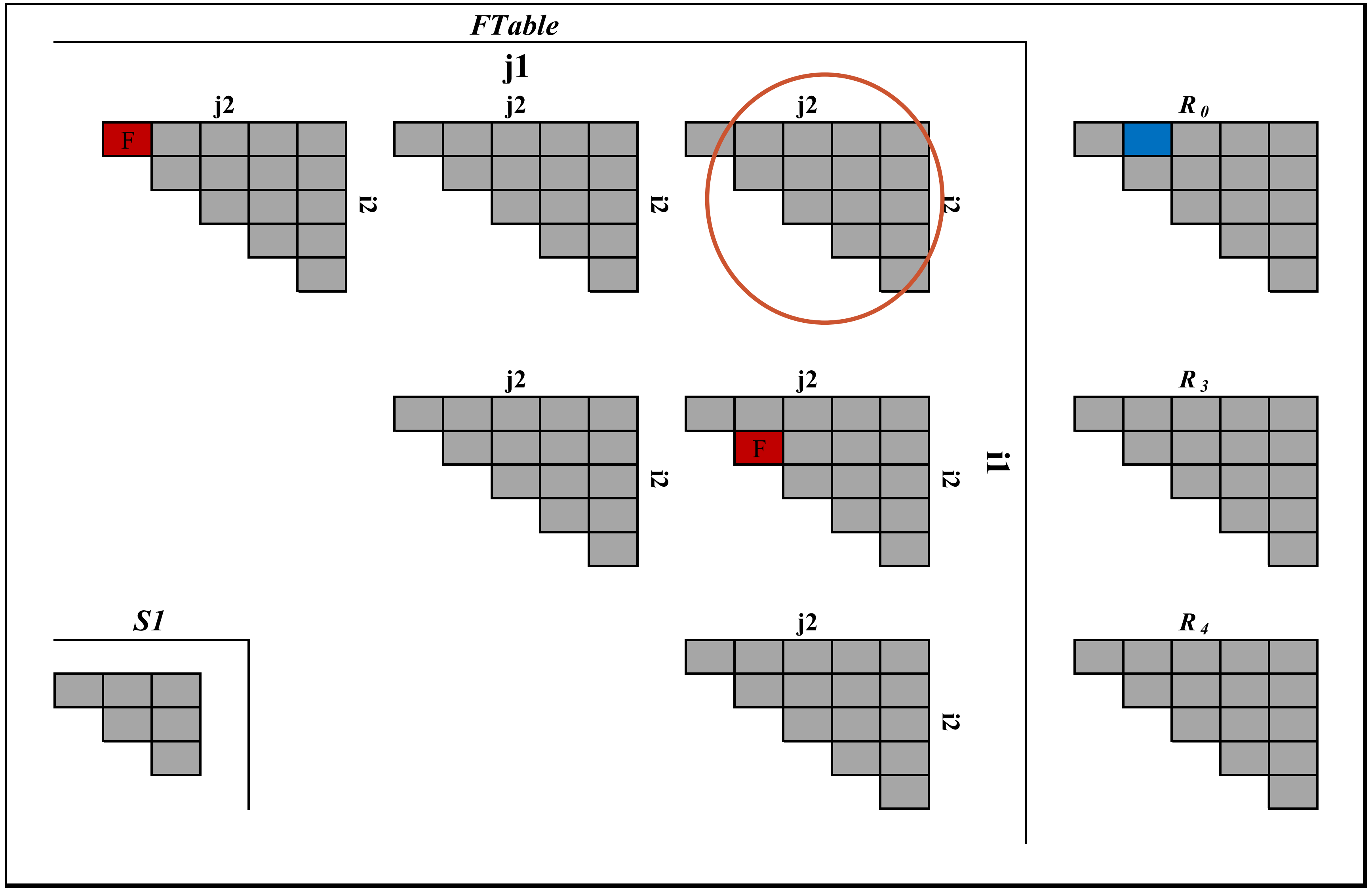
- Base schedule: $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, j_2\text{-}i_2, i_1, i_2, k_1, k_2]$
- Pulling k_1 loop outside of the inner three dimension decomposes the double max-plus operation to multiple matrix instance of max-plus operation
  - A series of matrices (i_1, k_1) to $(k_1 + 1, j_1)$
- Better schedule
  - Better schedule: $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, i_1, k_1, i_2, j_2, k_2]$
  - Schedule with auto-vectorization: $[i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, i_1, k_1, i_2, k_2, j_2]$
- Allows tiling of the inner three dimensions (i_2, k_2, j_2)
- Now, we can parallelize the i_1 or i_2 dimension



Scheduling

Double Max-Plus (R_0), R_3, and R_4

Scheduling Double Max-Plus (R_0), R_3, and R_4



Starting with a R_0 schedule which exploits auto-vectorization

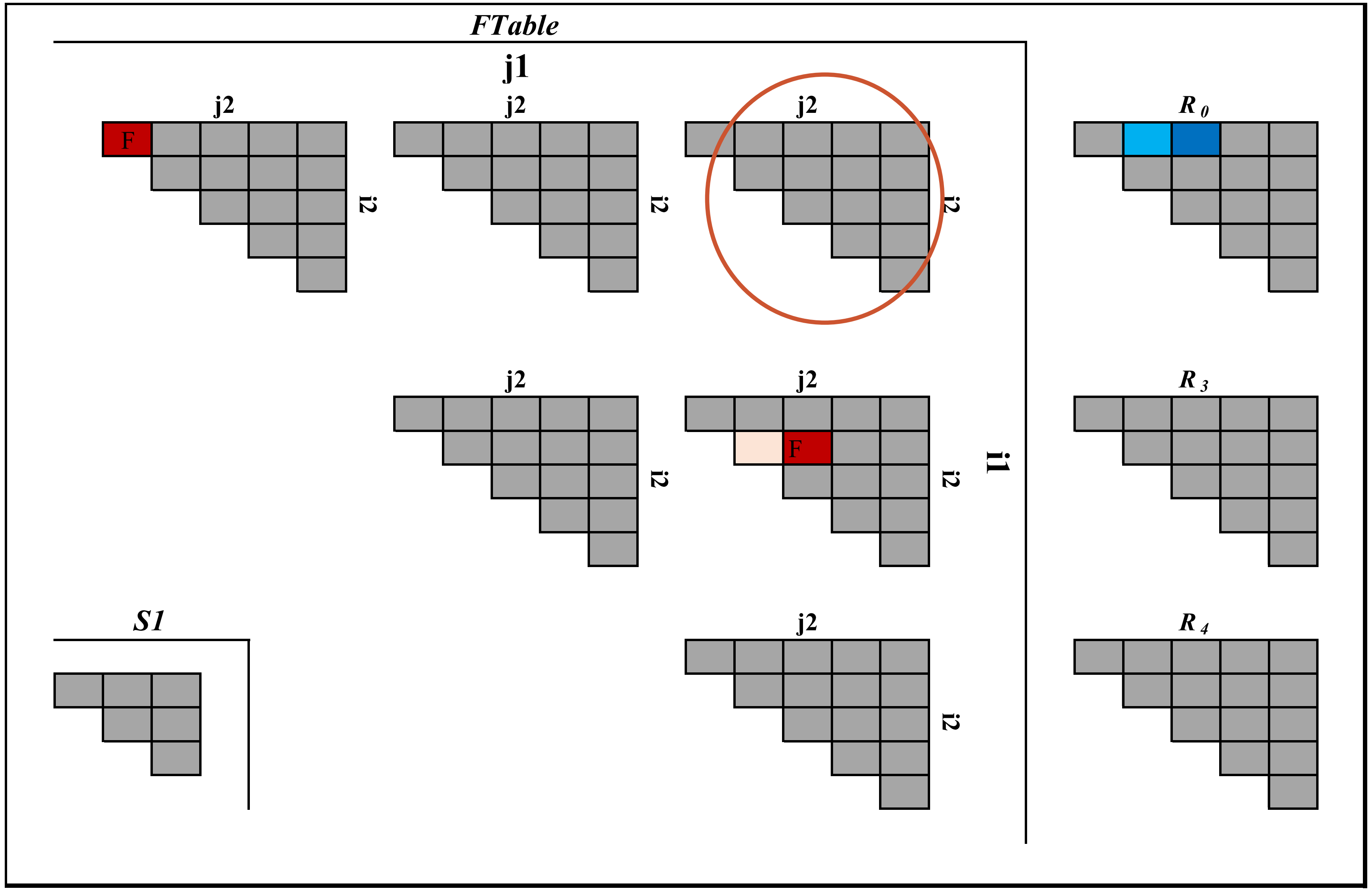
R_0 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2]$

R_3 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

R_4 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

New dimension added

Scheduling Double Max-Plus (R_0), R_3, and R_4



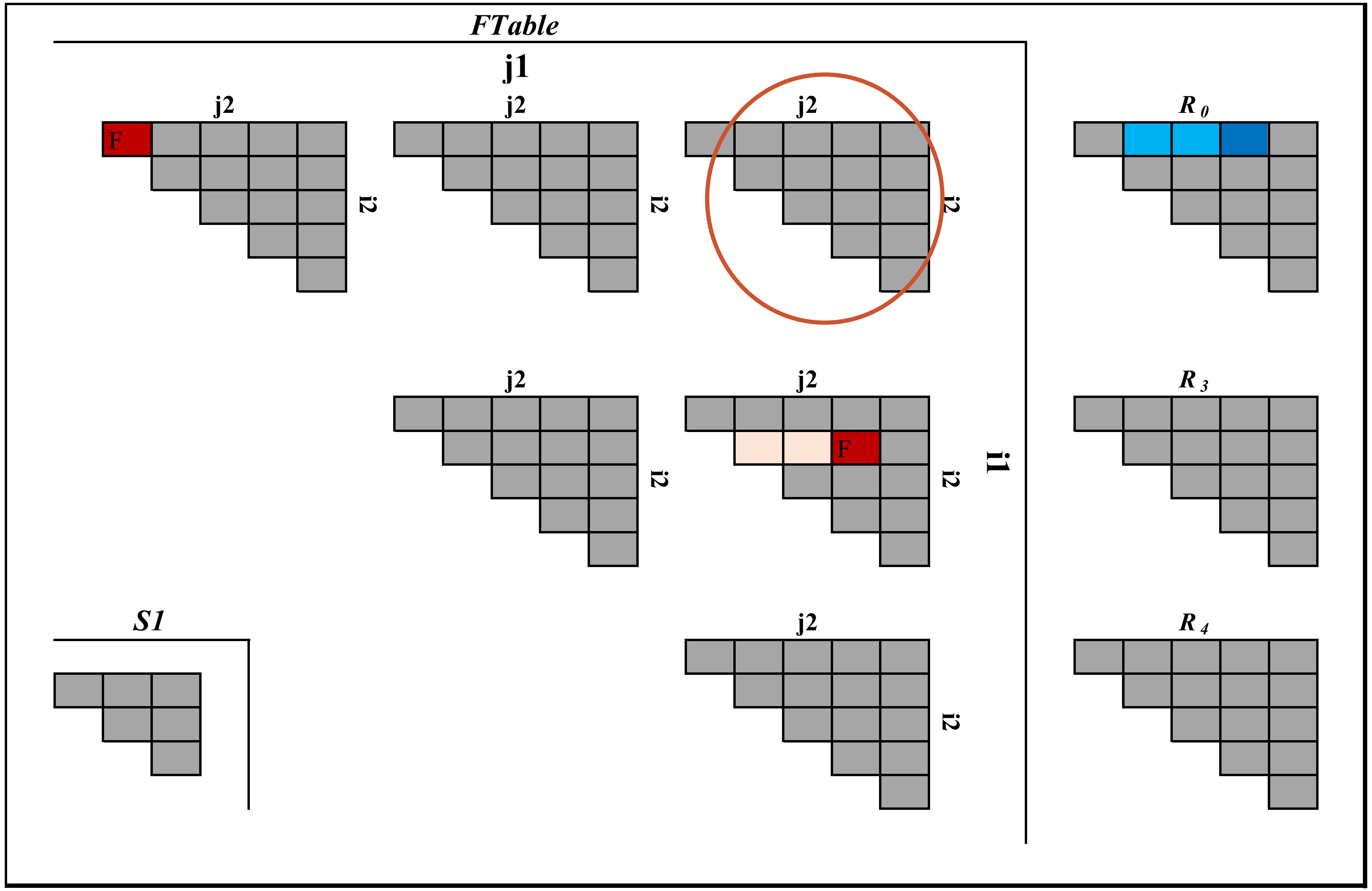
Single Memory element is used with many elements of triangles towards south

R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N{+}1,\ i_2,\ j_2$]

R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N{+}1,\ i_2,\ j_2$]

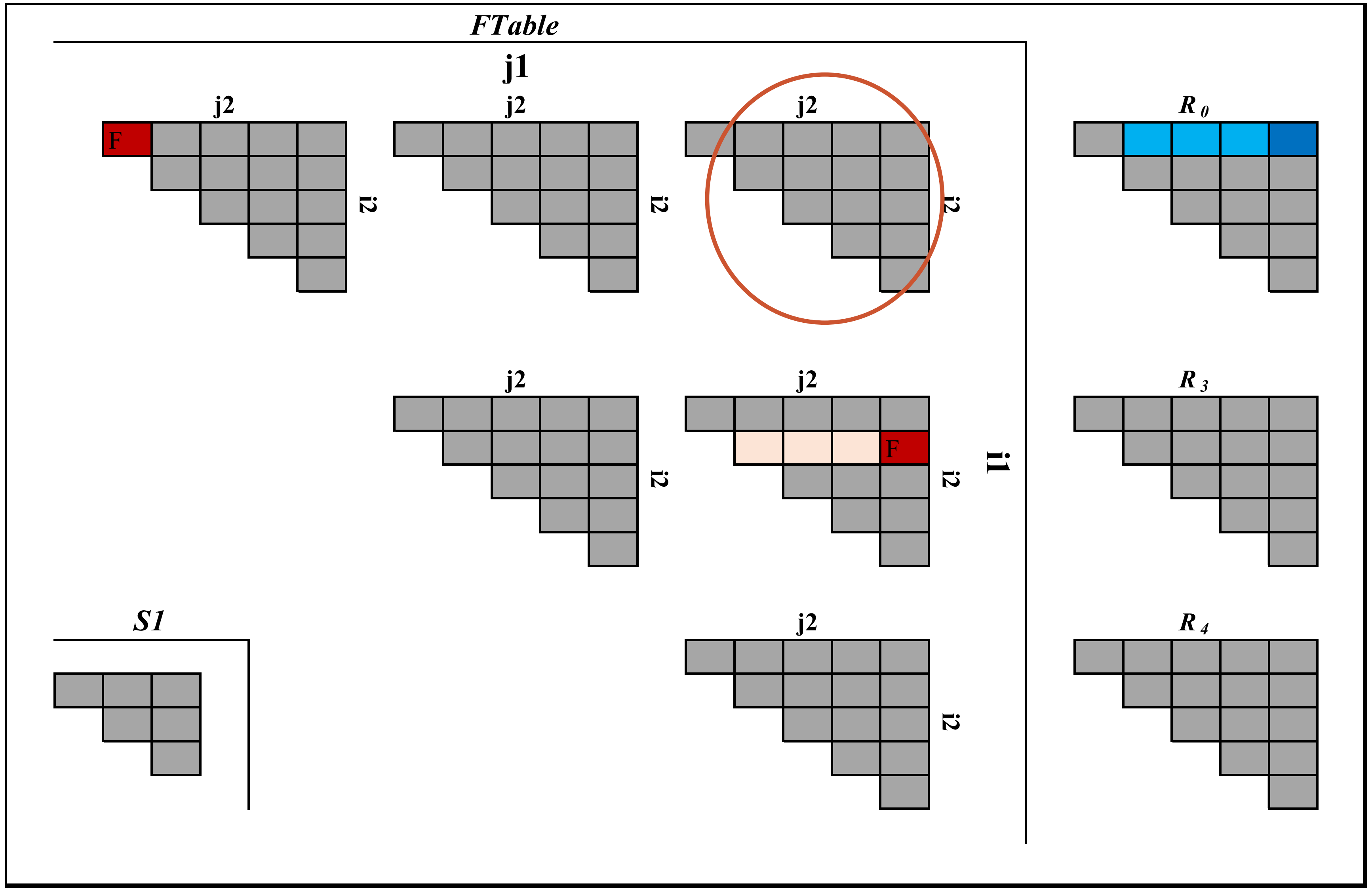
Scheduling Double Max-Plus (R_0), R_3, and R_4



Single memory element of left triangle is used with many elements of triangles towards south

R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, i_2, k_2, j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4

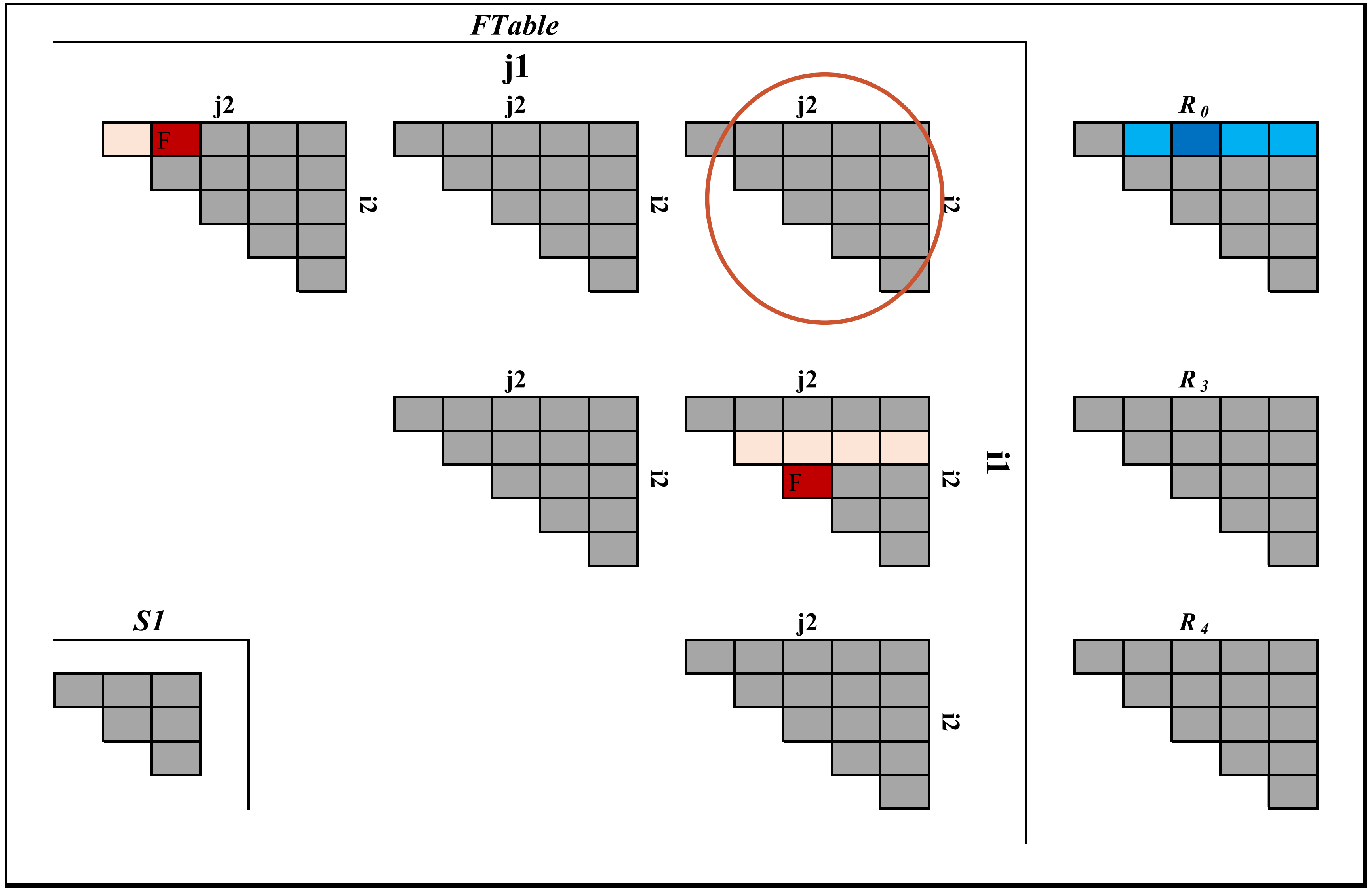


R_0 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2]$

R_3 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

R_4 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

Scheduling Double Max-Plus (R_0), R_3, and R_4

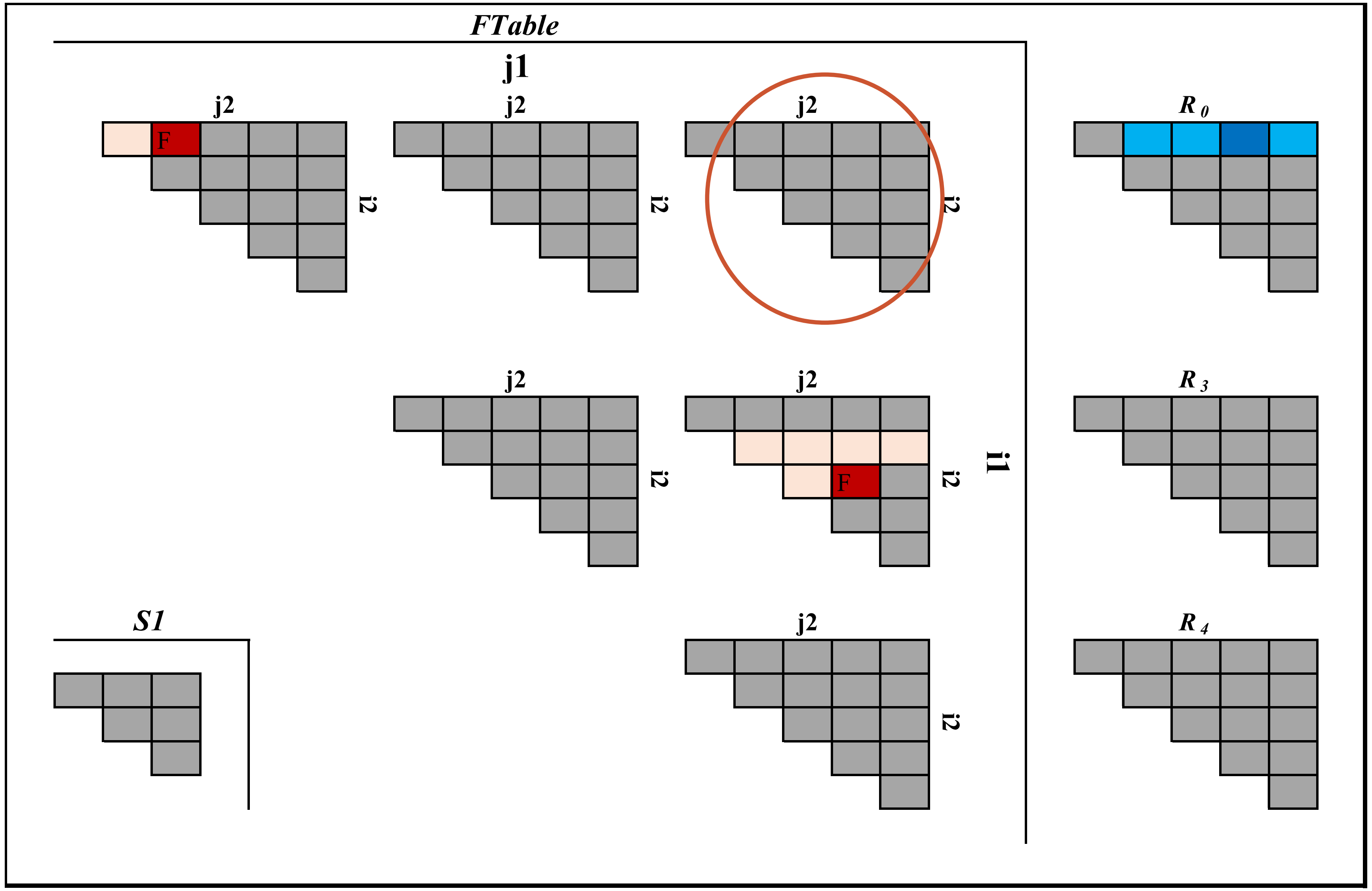


R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; i_2, \; k_2, \; j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]

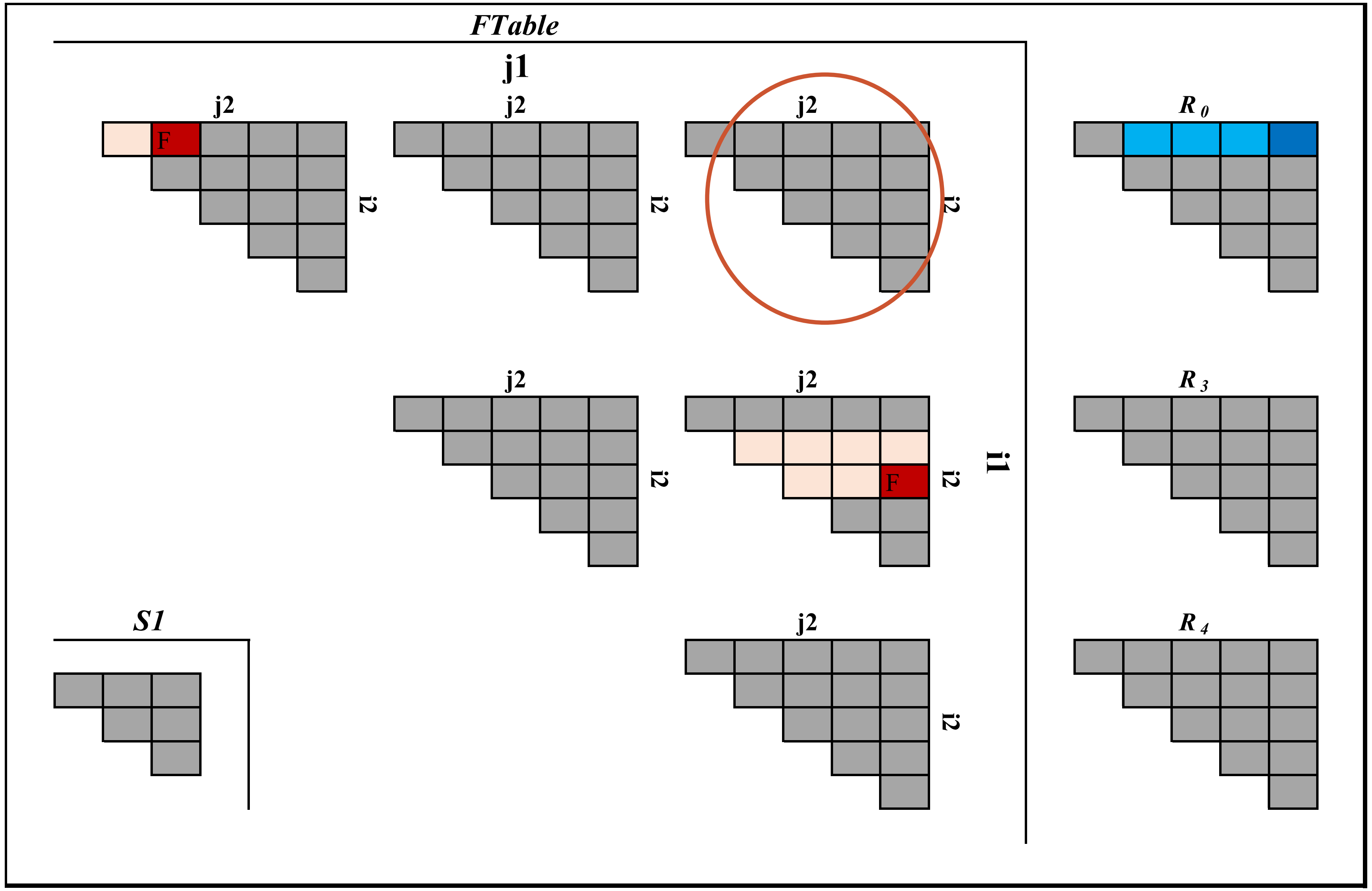
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4



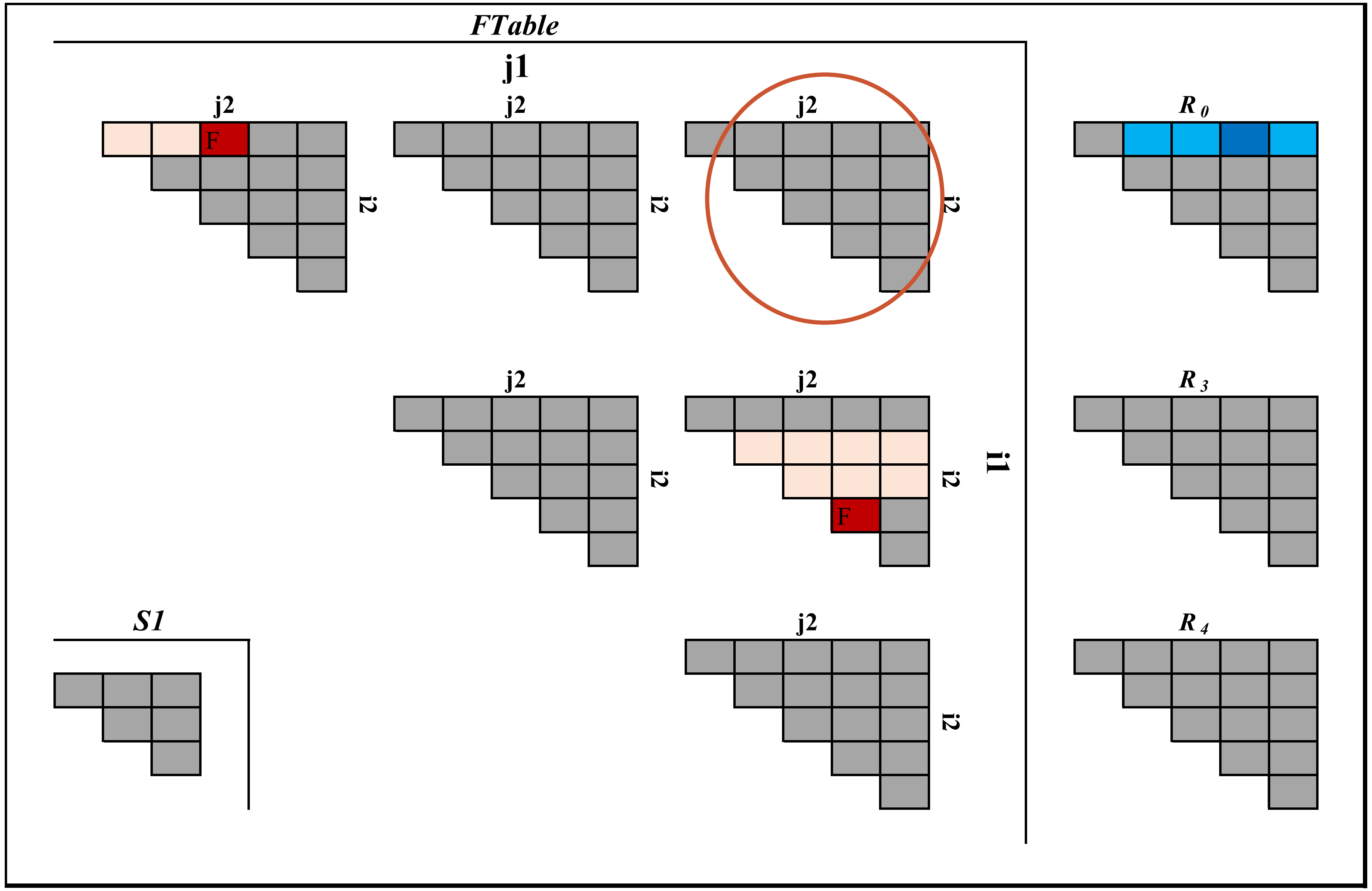
R_0 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, i_2, k_2, j_2]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, N+1, i_2, j_2]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, N+1, i_2, j_2]

Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, i_2, k_2, j_2]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, N+1, i_2, j_2]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow$ j_1-i_1, i_1, k_1, N+1, i_2, j_2]

Scheduling Double Max-Plus (R_0), R_3, and R_4

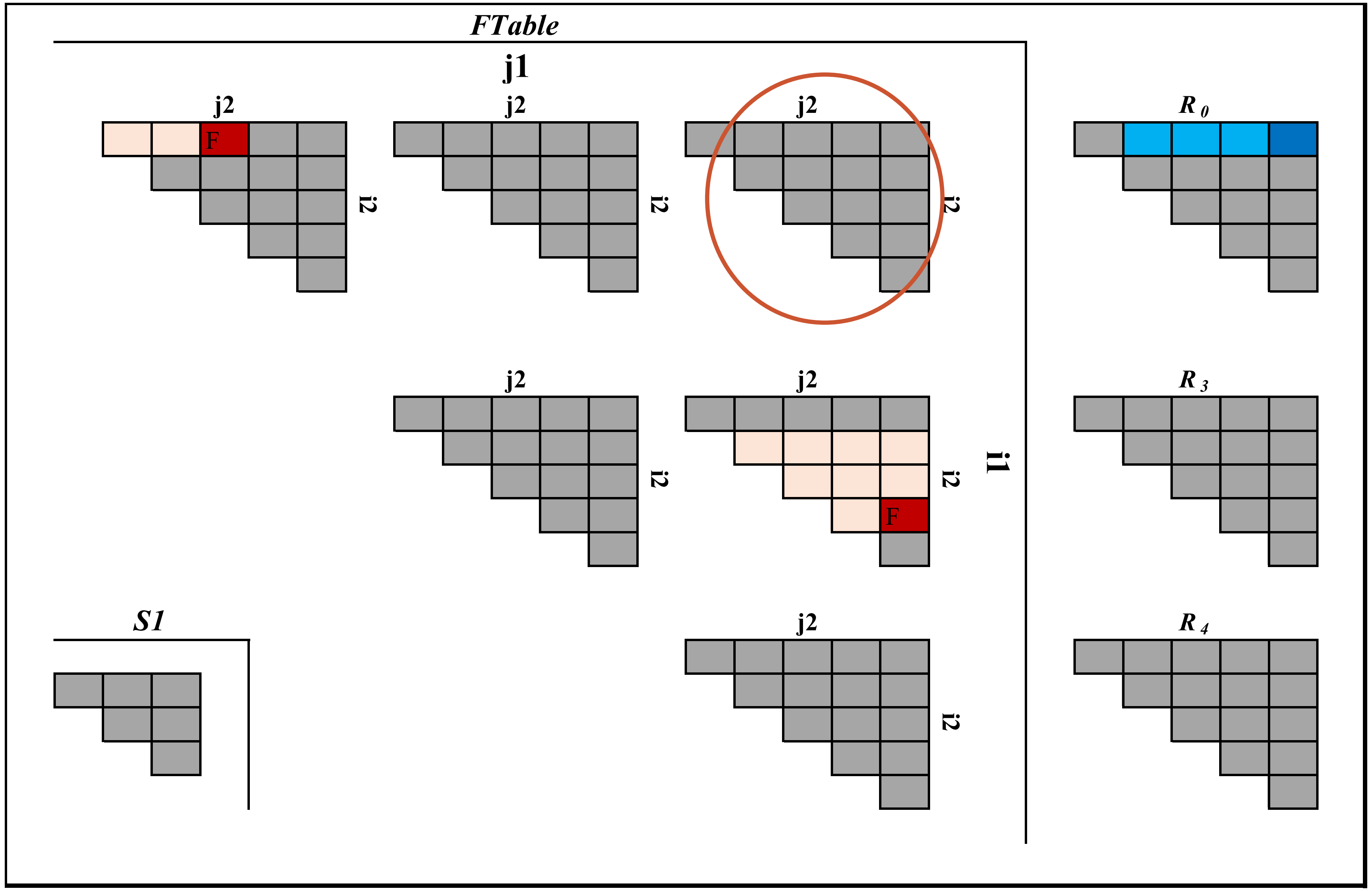


R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \ i_1, \ k_1, \ i_2, \ k_2, \ j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \ i_1, \ k_1, \ N+1, \ i_2, \ j_2$]

R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \ i_1, \ k_1, \ N+1, \ i_2, \ j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4

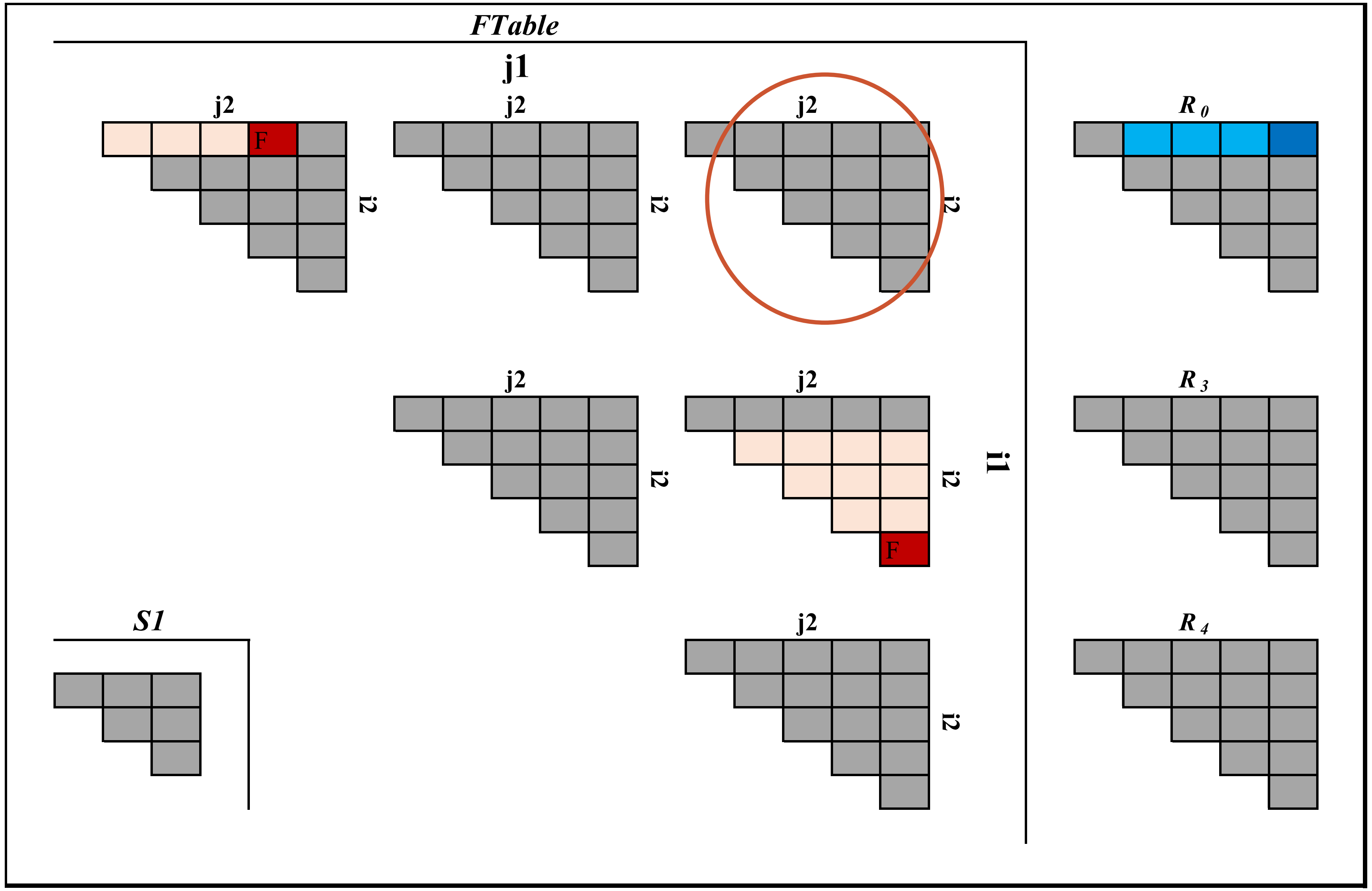


R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; i_2, \; k_2, \; j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N\text{+}1, \; i_2, \; j_2$]

R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N\text{+}1, \; i_2, \; j_2$]

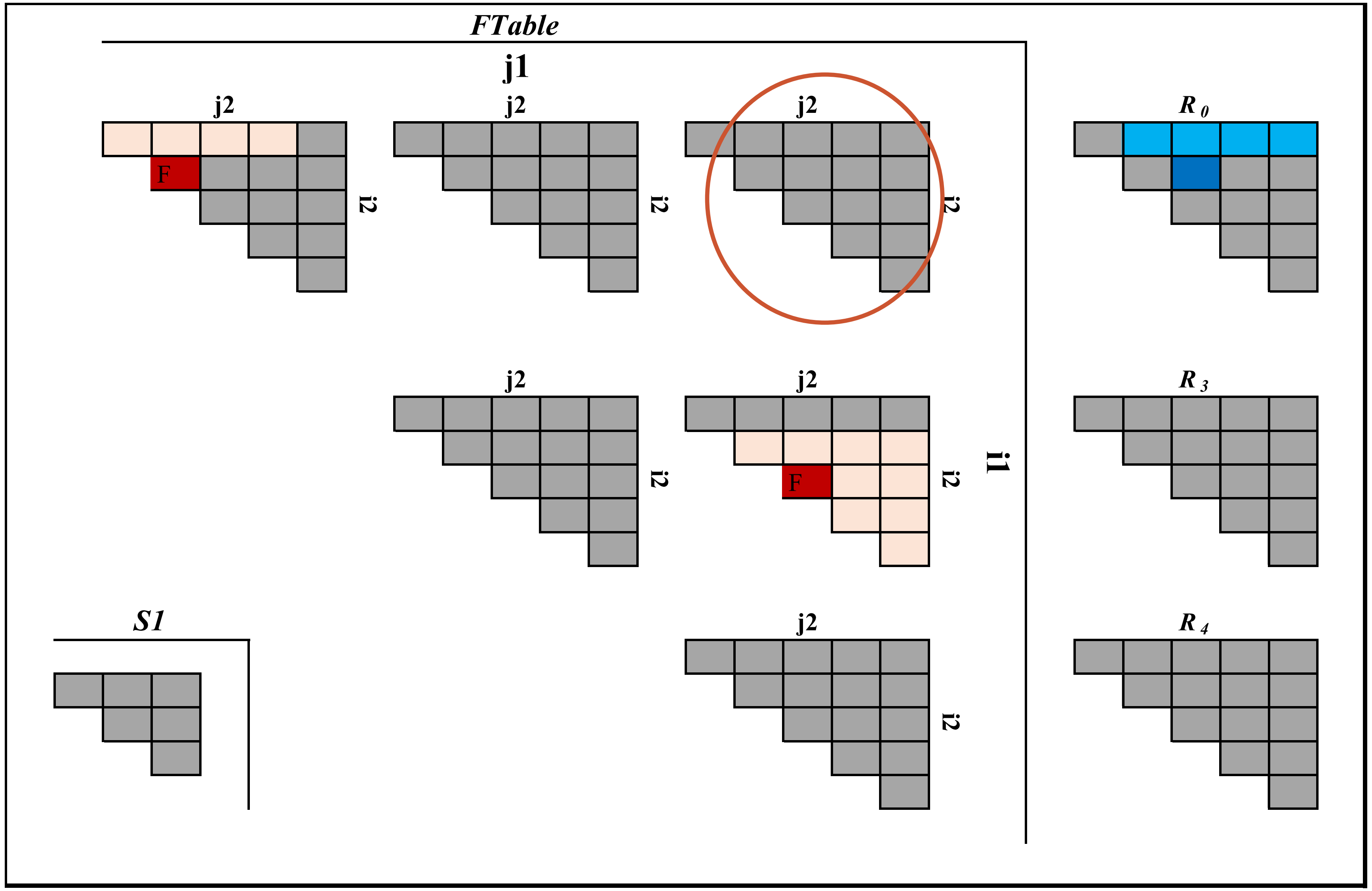
Scheduling Double Max-Plus (R_0), R_3, and R_4



Elements of triangle towards the south is also used multiple times

R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, i_1, k_1, i_2, k_2, j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, i_1, k_1, N+1, i_2, j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, i_1, k_1, N+1, i_2, j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4

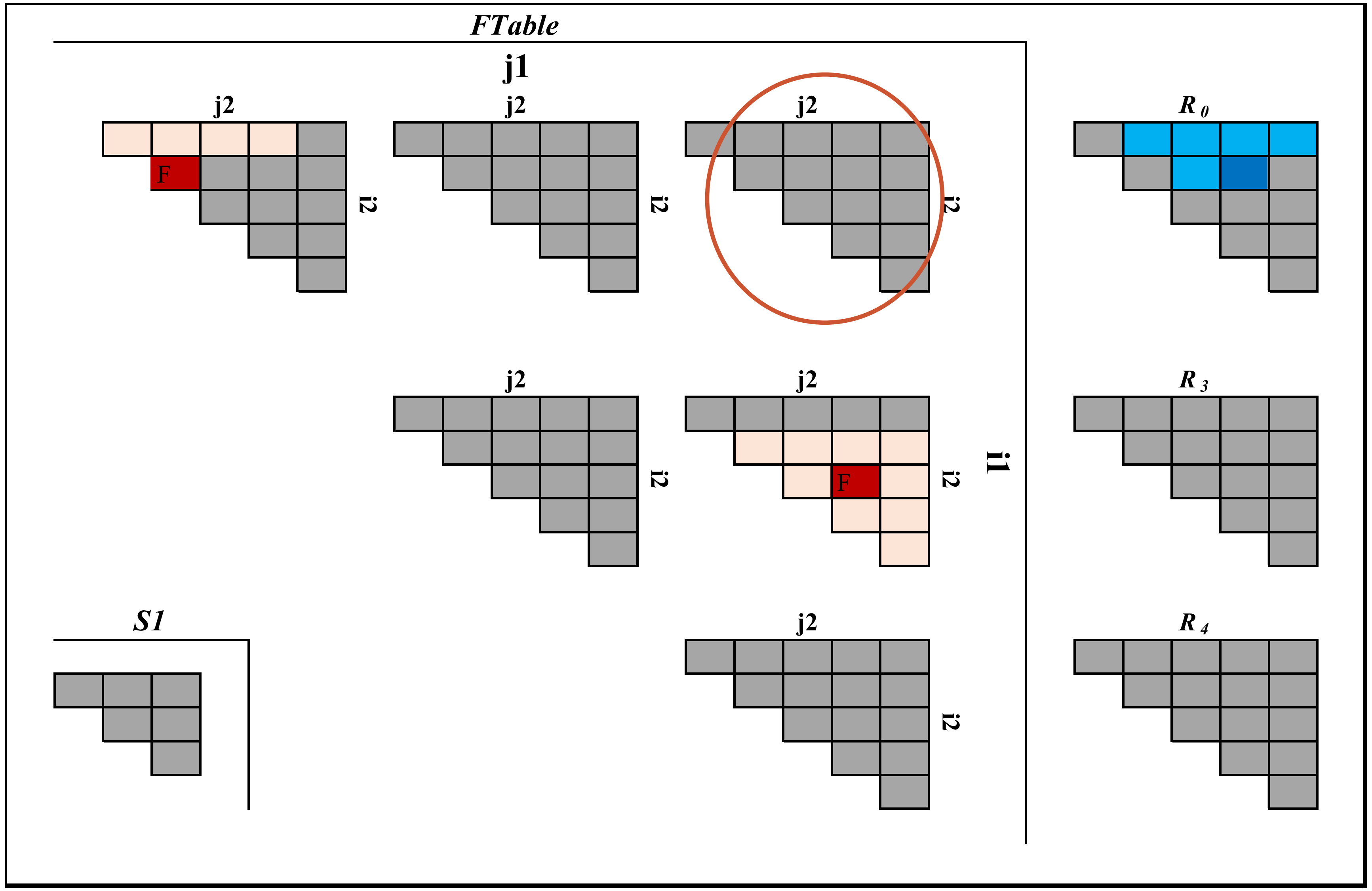


$R_0\ [\, i_1, j_1, i_2, j_2 \rightarrow j_1 - i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2]$

$R_3\ [\, i_1, j_1, i_2, j_2 \rightarrow j_1 - i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

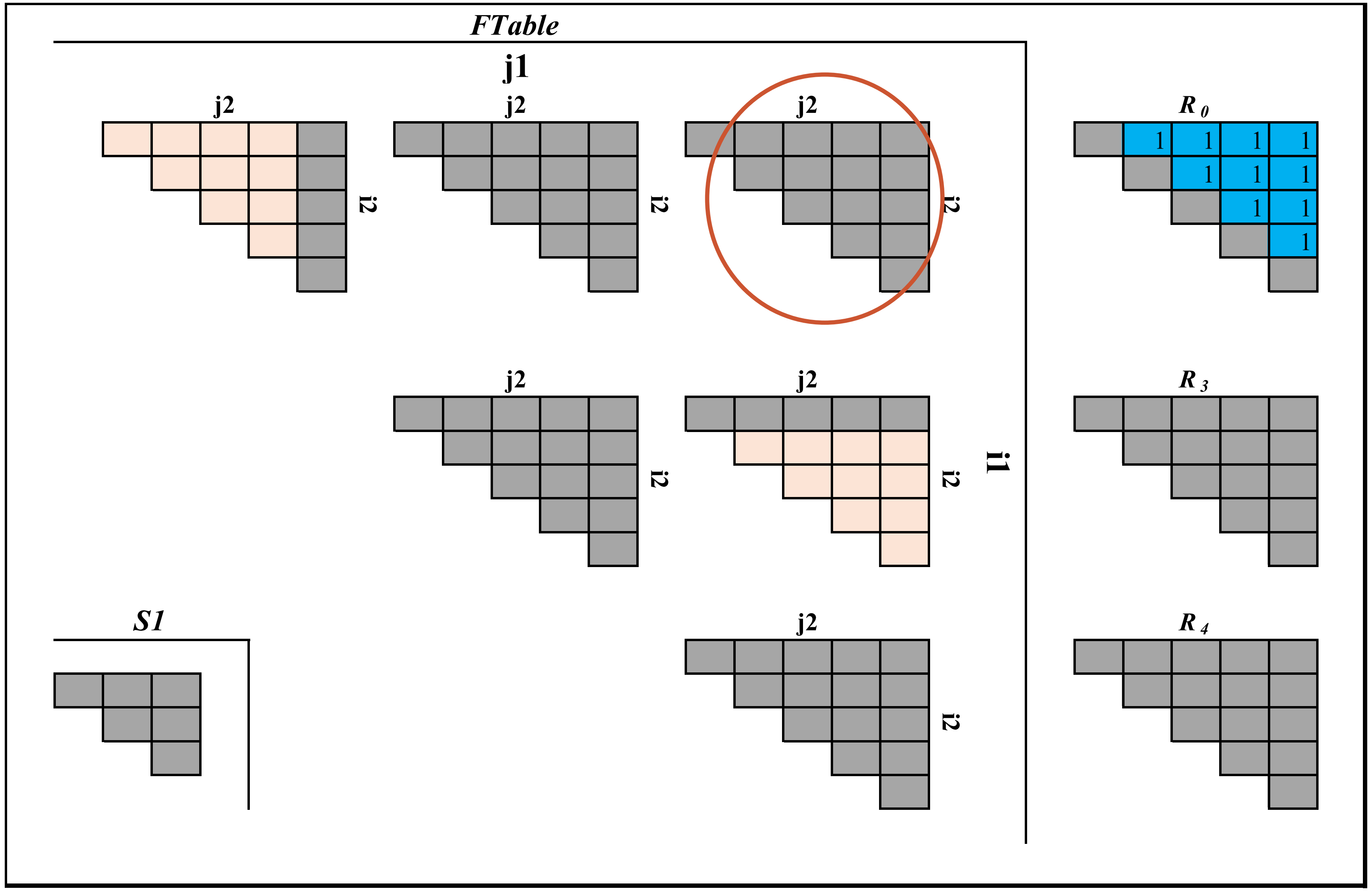
$R_4\ [\, i_1, j_1, i_2, j_2 \rightarrow j_1 - i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, i_2, k_2, j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]

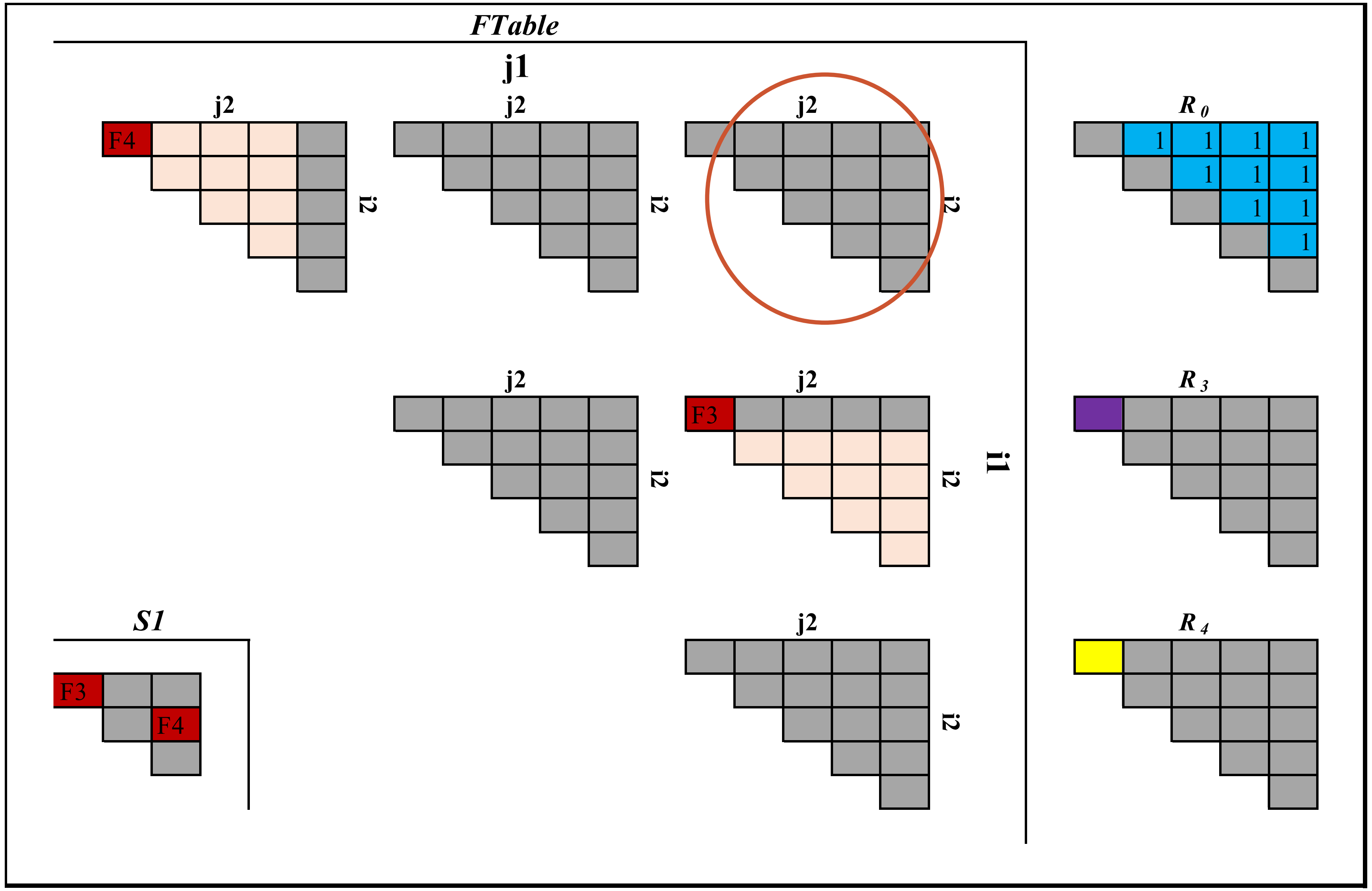
Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; i_2, \; k_2, \; j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]

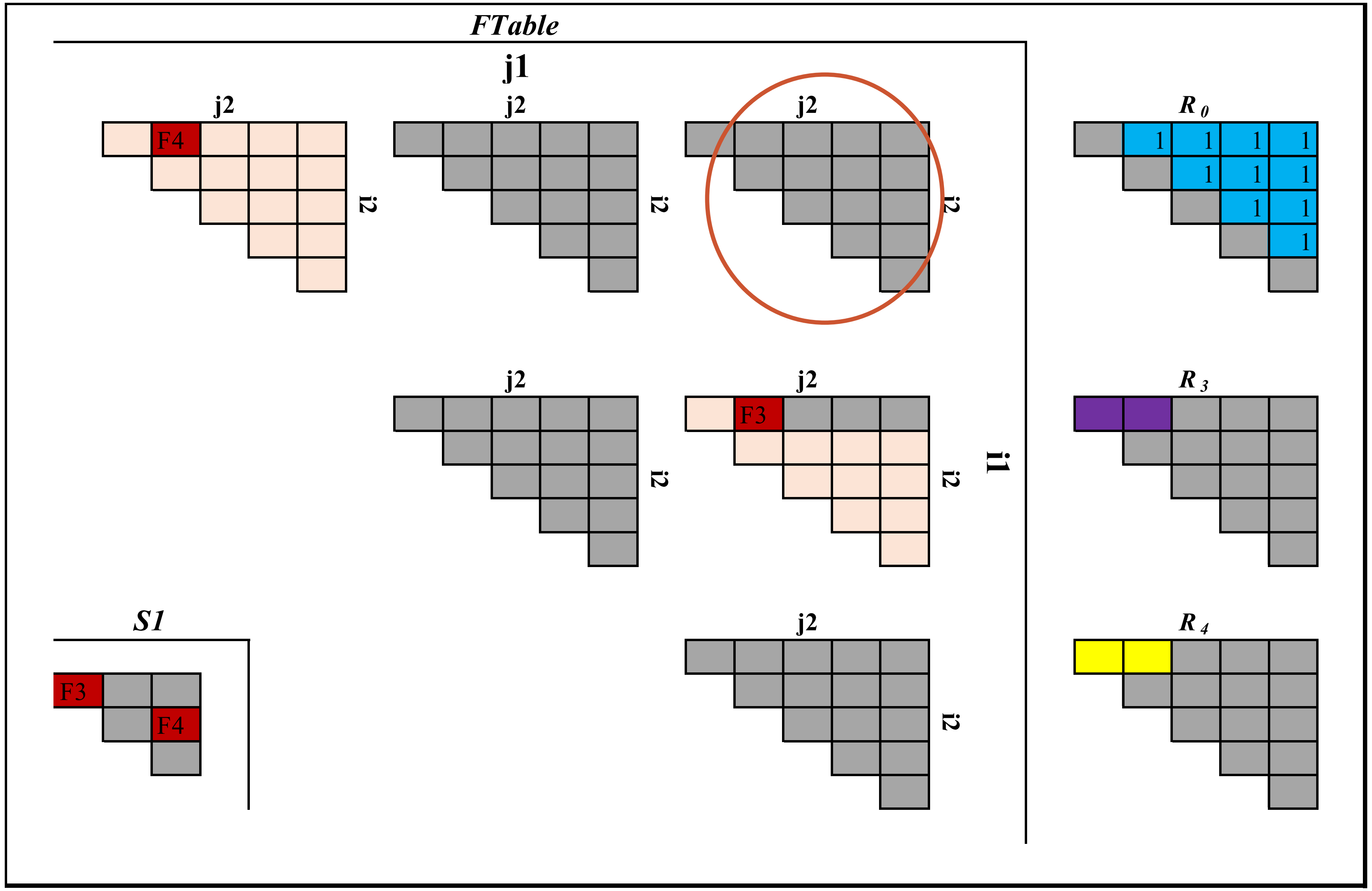
Elements of triangles from left and south can also be used to compute the corresponding R_3 and R_4

Scheduling Double Max-Plus (R_0), R_3, and R_4



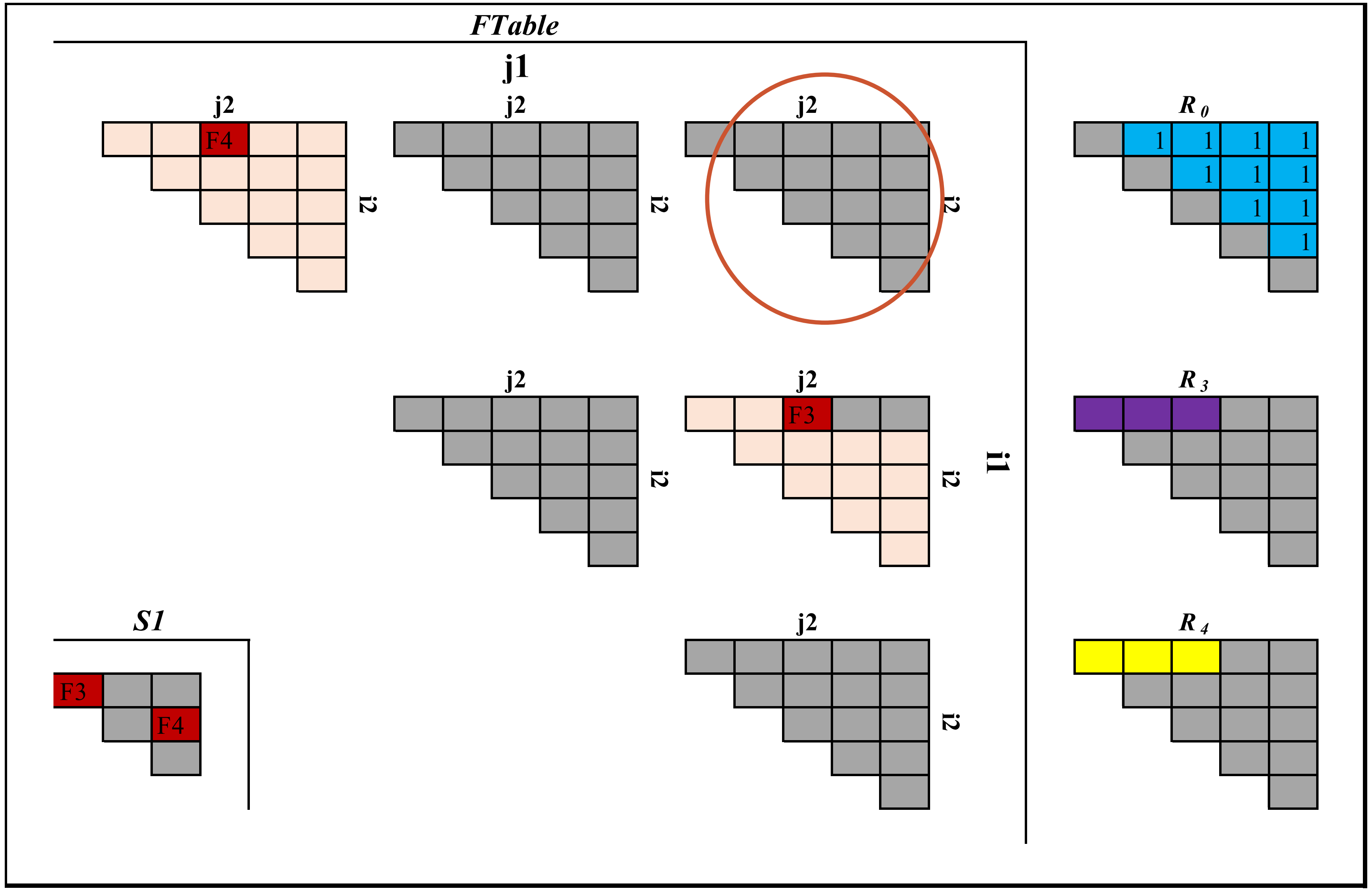
R_0 [i_1, j_1, i_2, j_2 → j_1-i_1, i_1, k_1, i_2, k_2, j_2]
R_3 [i_1, j_1, i_2, j_2 → j_1-i_1, i_1, k_1, N+1, i_2, j_2]
R_4 [i_1, j_1, i_2, j_2 → j_1-i_1, i_1, k_1, N+1, i_2, j_2]

Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; i_2, \; k_2, \; j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N{+}1, \; i_2, \; j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N{+}1, \; i_2, \; j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4

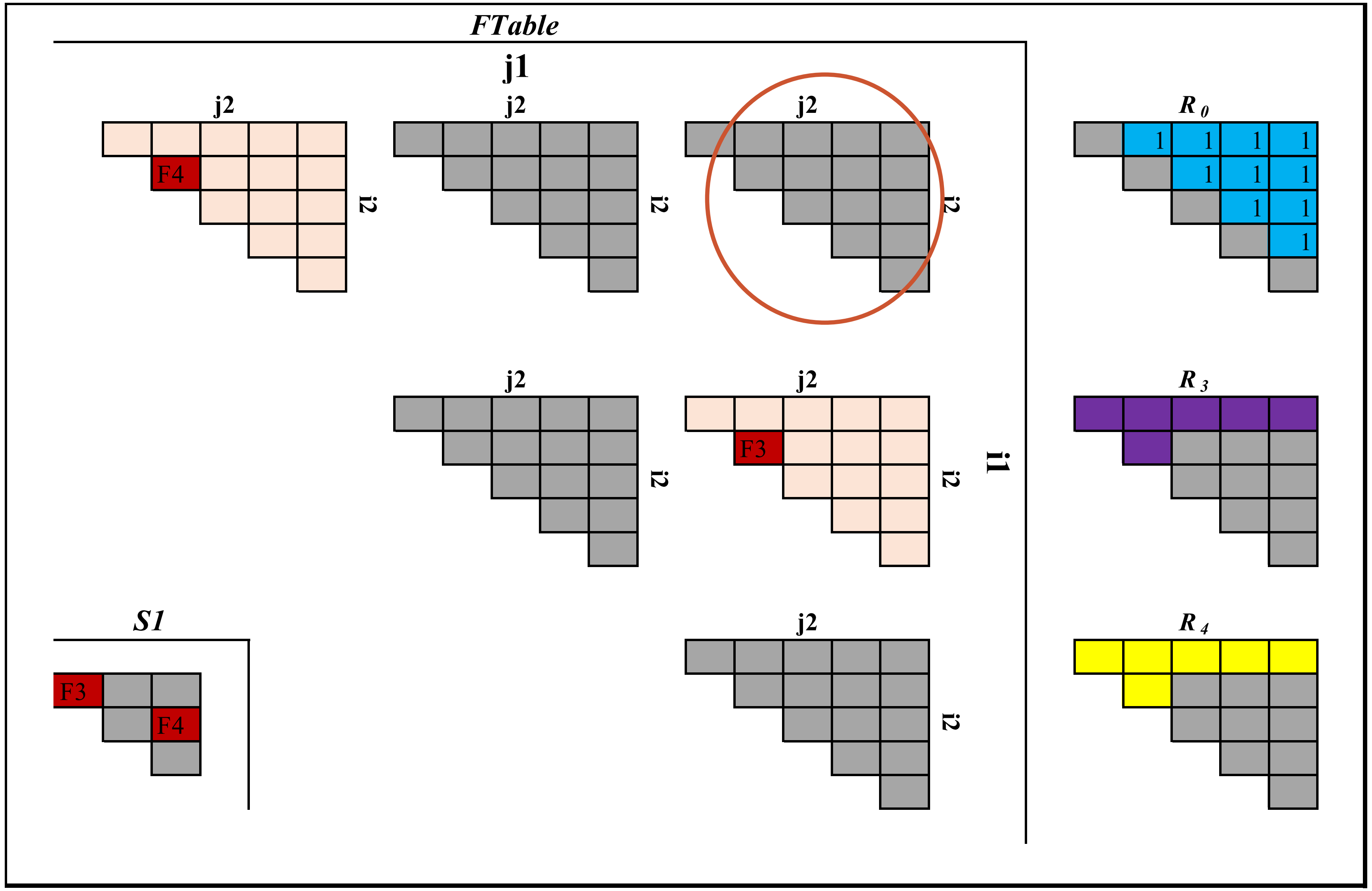


R_0 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2]$

R_3 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

R_4 $[\, i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2]$

Scheduling Double Max-Plus (R_0), R_3, and R_4

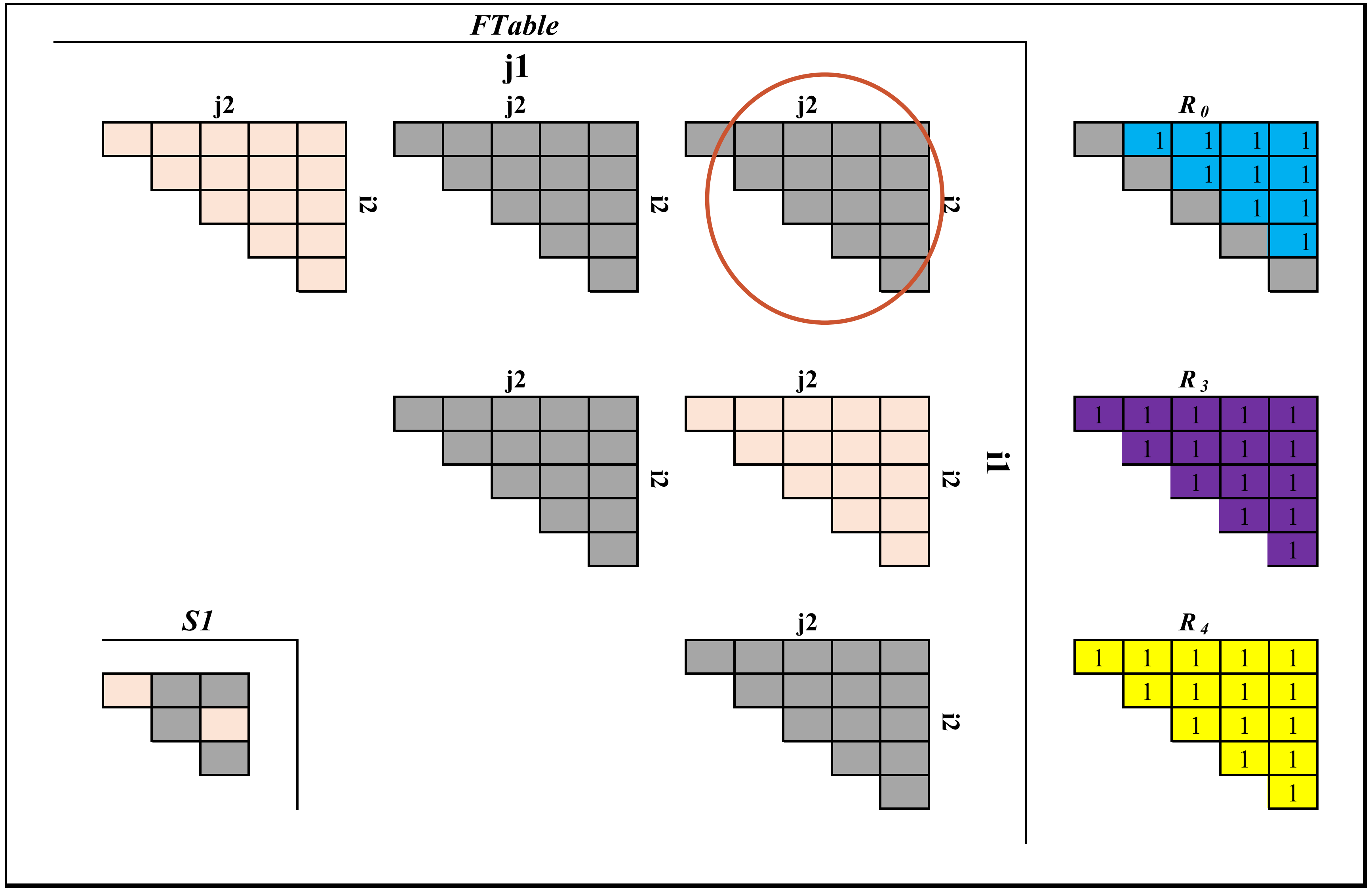


R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2$]

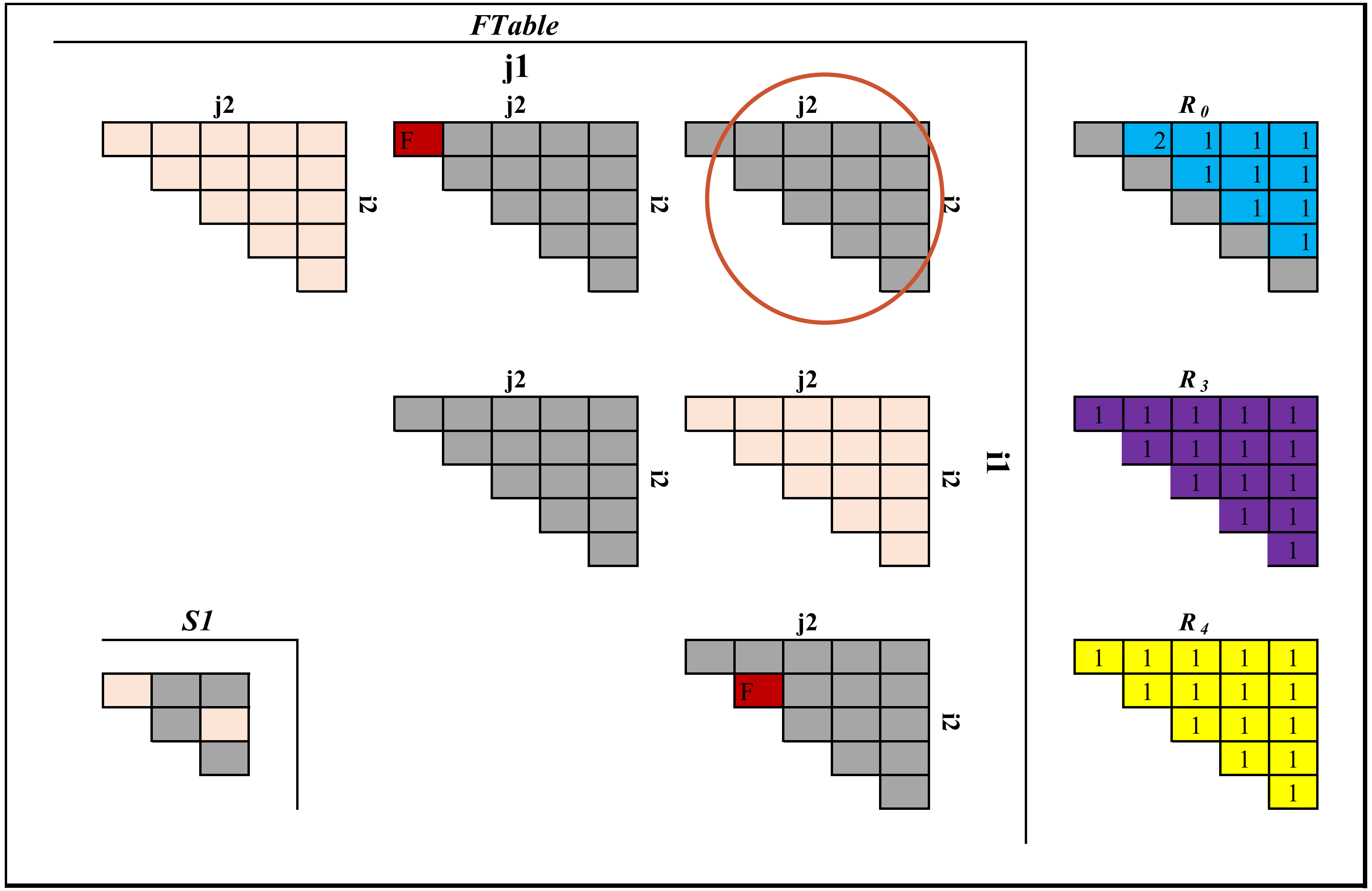
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, i_2, k_2, j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1, i_1, k_1, N+1, i_2, j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4

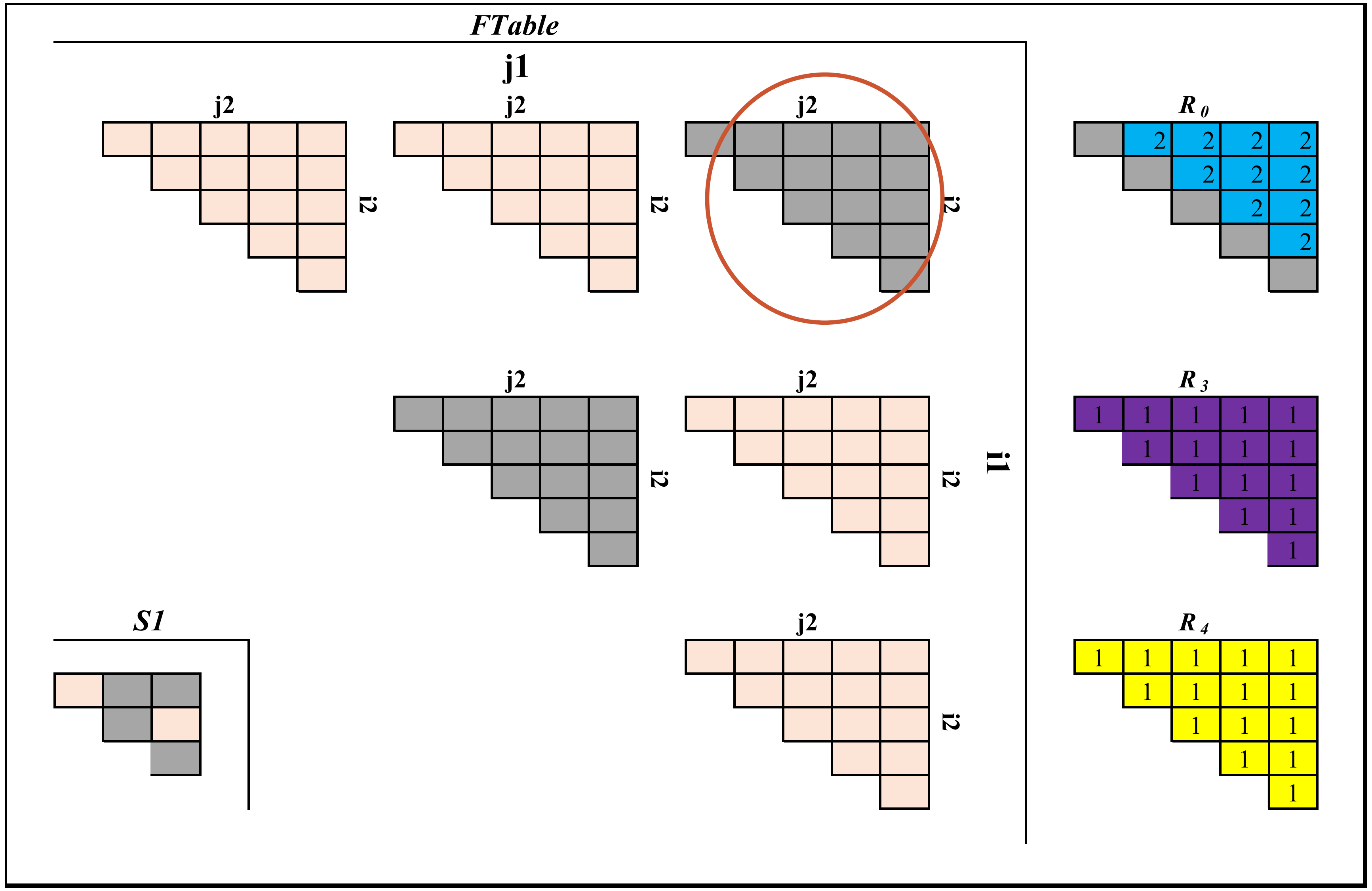


R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; i_2, \; k_2, \; j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]

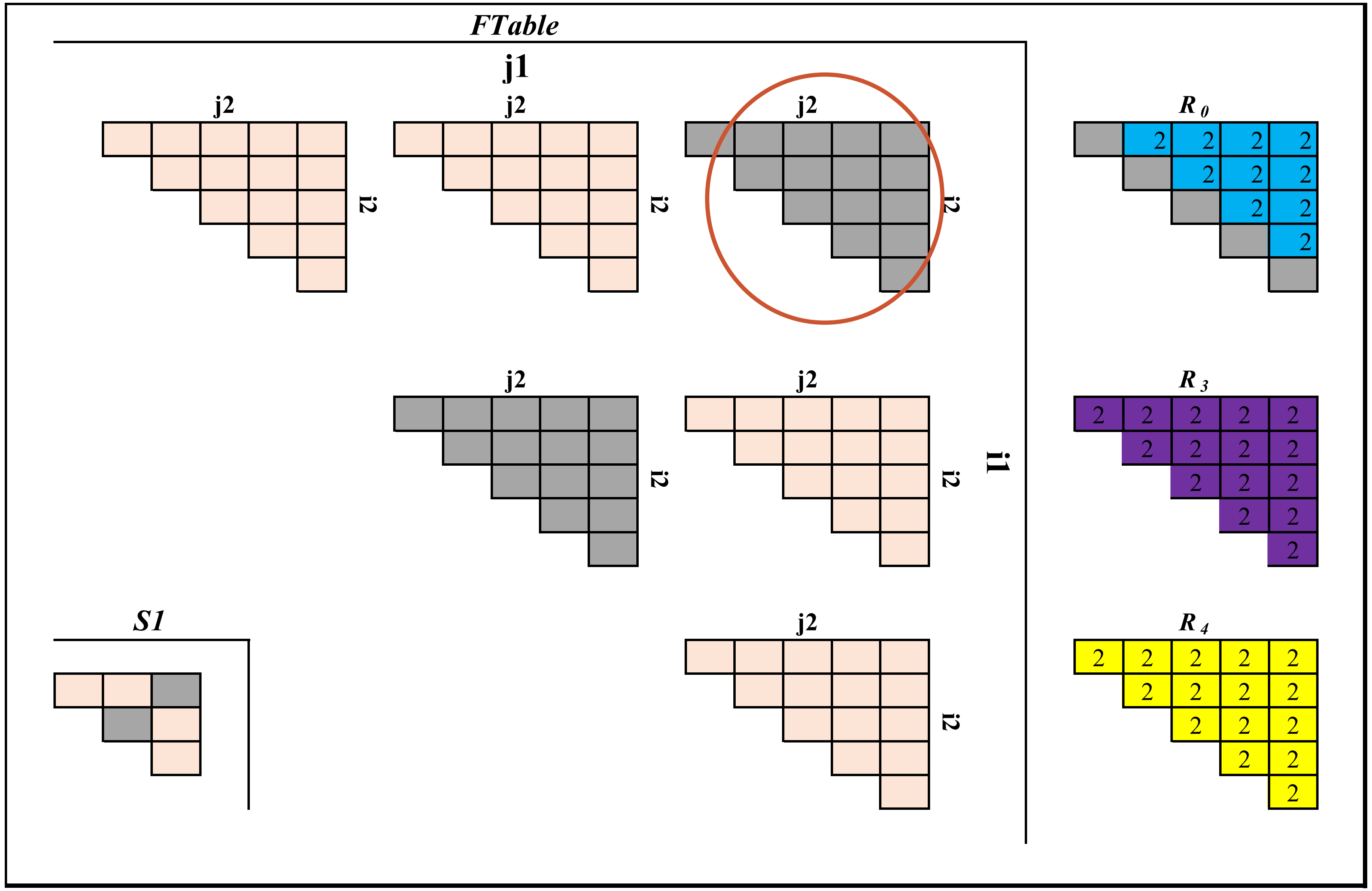
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1, \; i_1, \; k_1, \; N+1, \; i_2, \; j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4



R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2$]
R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N{+}1,\ i_2,\ j_2$]
R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1\text{-}i_1,\ i_1,\ k_1,\ N{+}1,\ i_2,\ j_2$]

Scheduling Double Max-Plus (R_0), R_3, and R_4



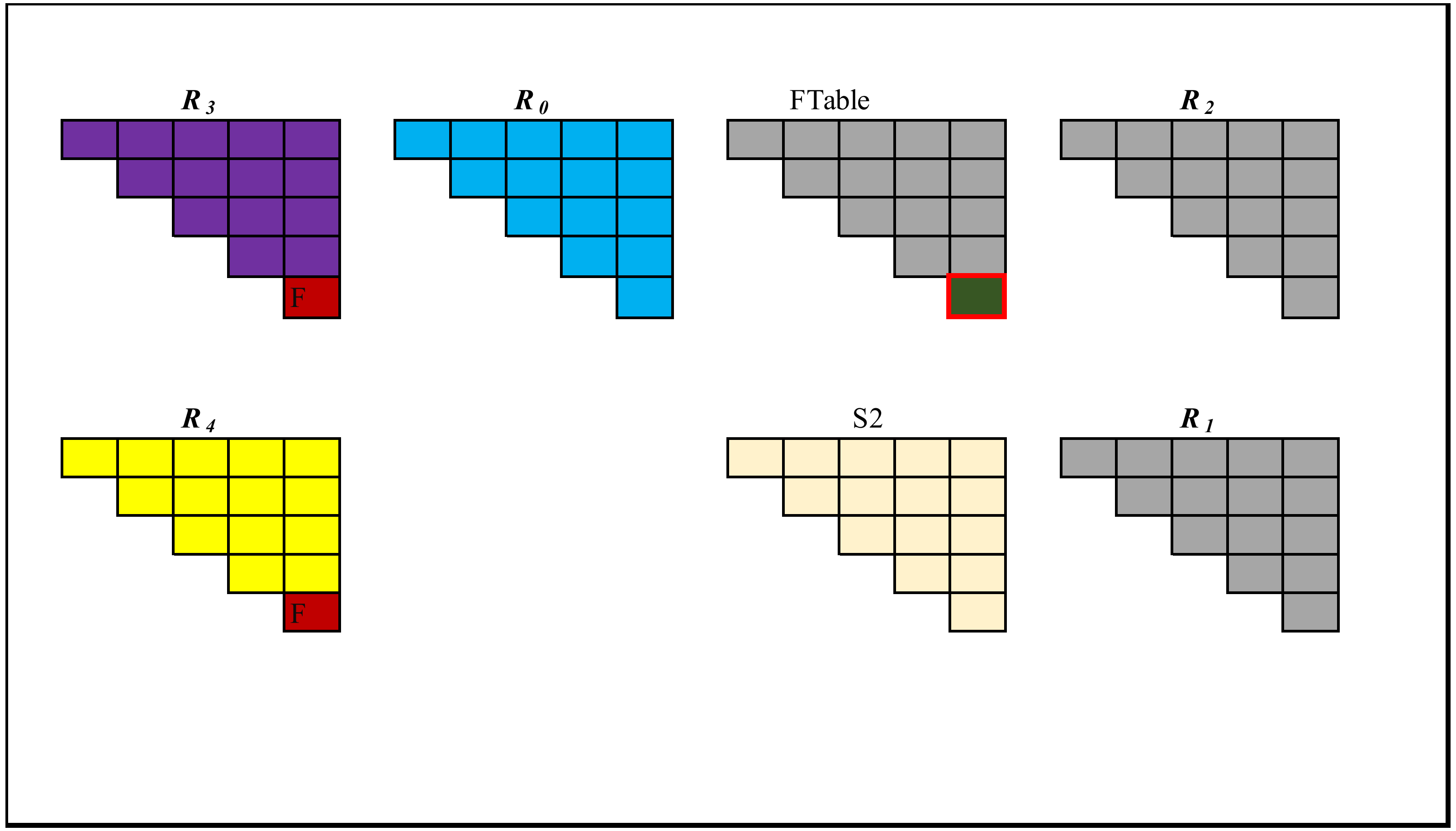
R_0 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ i_2,\ k_2,\ j_2$]

R_3 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2$]

R_4 [$i_1, j_1, i_2, j_2 \rightarrow j_1-i_1,\ i_1,\ k_1,\ N+1,\ i_2,\ j_2$]

Scheduling R_1, R_2, and F-Table

Scheduling R_1, R_2, and F-Table

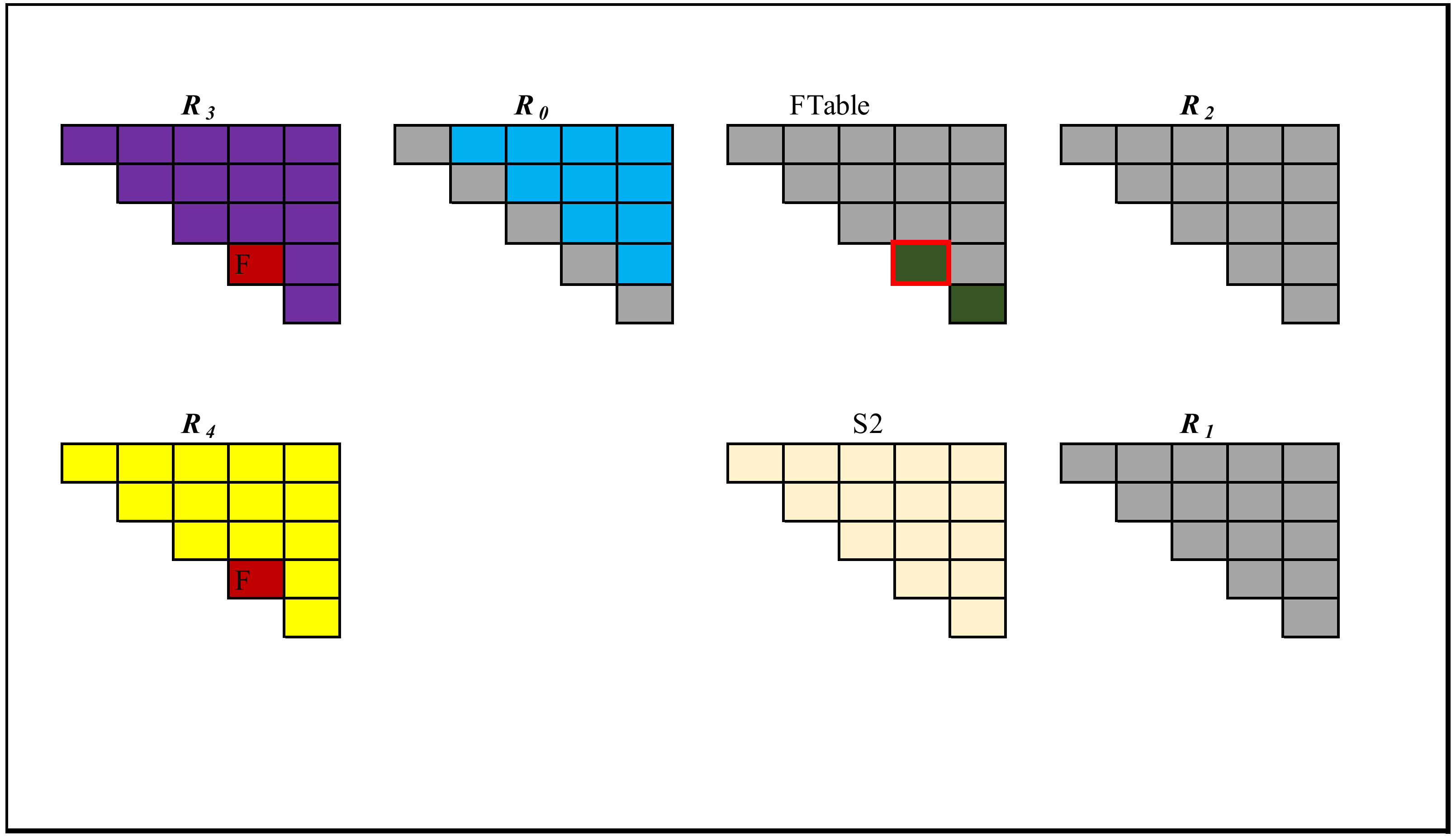


Each element of FTable is dependent on corresponding values from R_0, R_1, R_2, R_3, R_4

R_0, R_3, R_4 computation for the current triangle is already done

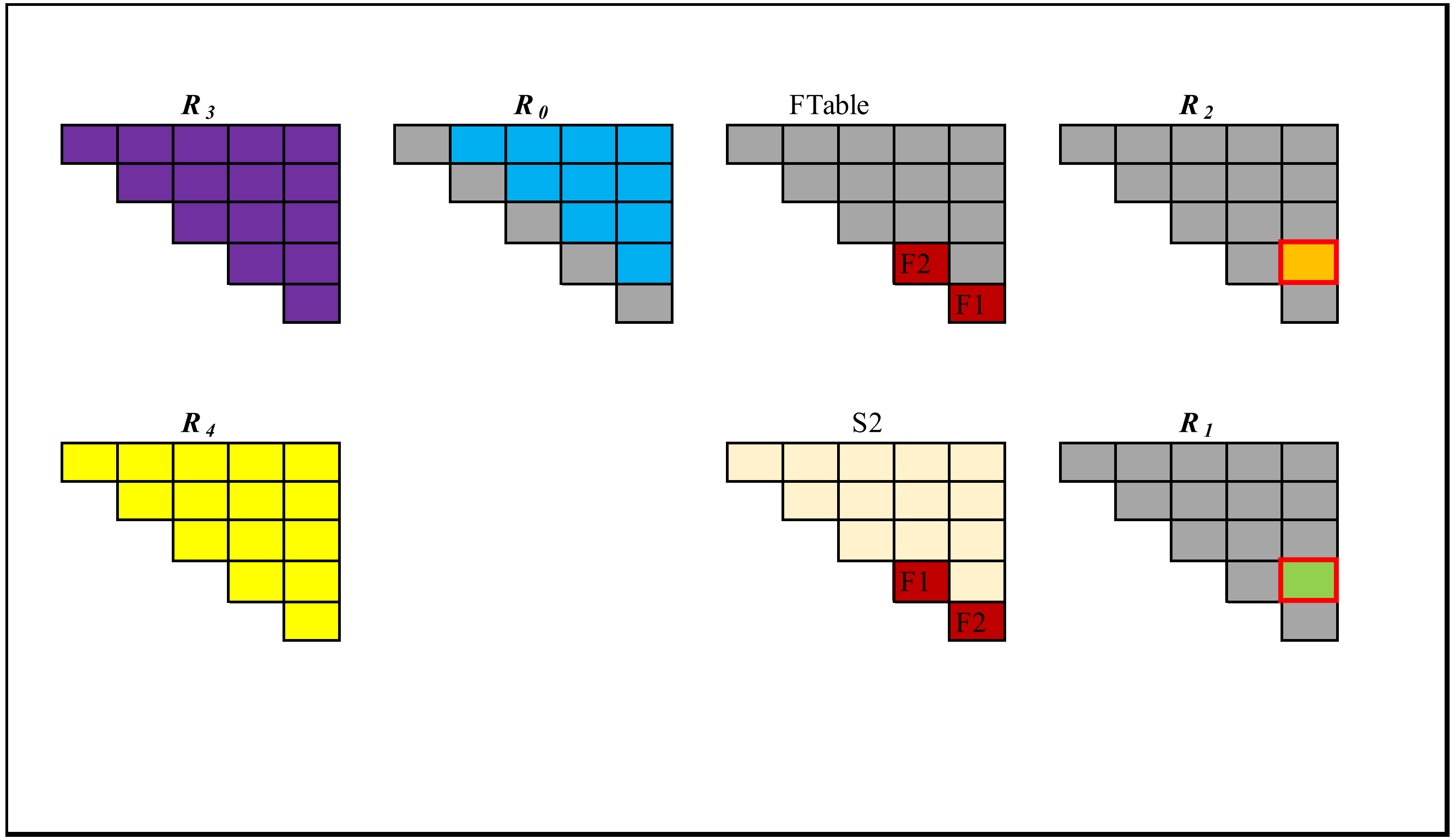
FTable	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, j_2, 0 \,]$
R_1	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2 \,]$
R_2	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2 \,]$

Scheduling R_1, R_2, and F-Table



FTable	**[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, j_2, 0$]**
R_1	**[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2$]**
R_2	**[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2$]**

Scheduling R_1, R_2, and F-Table

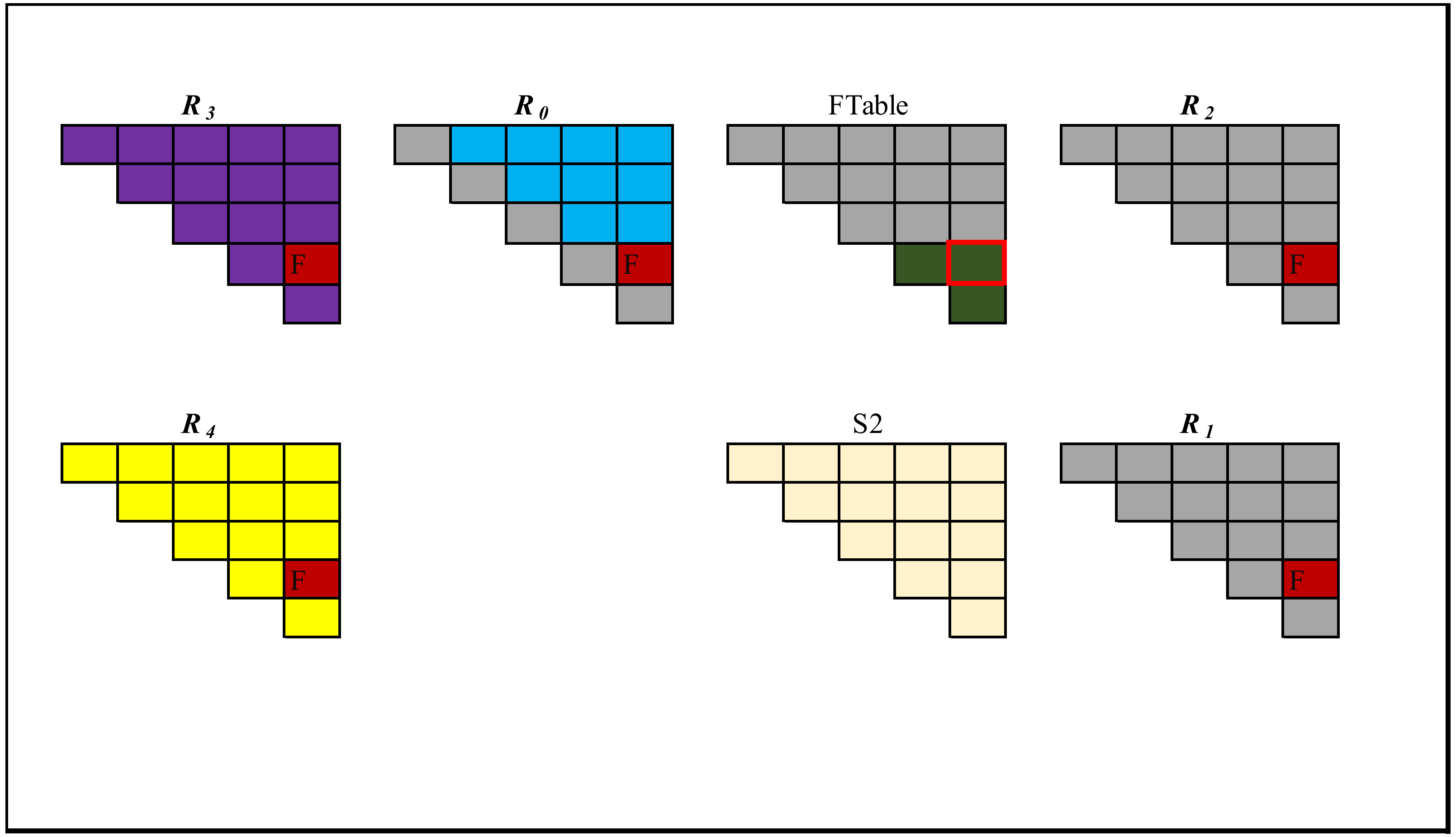


FTable $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ j_2,\ 0 \,]$

R_1 $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$

R_2 $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$

Scheduling R_1, R_2, and F-Table

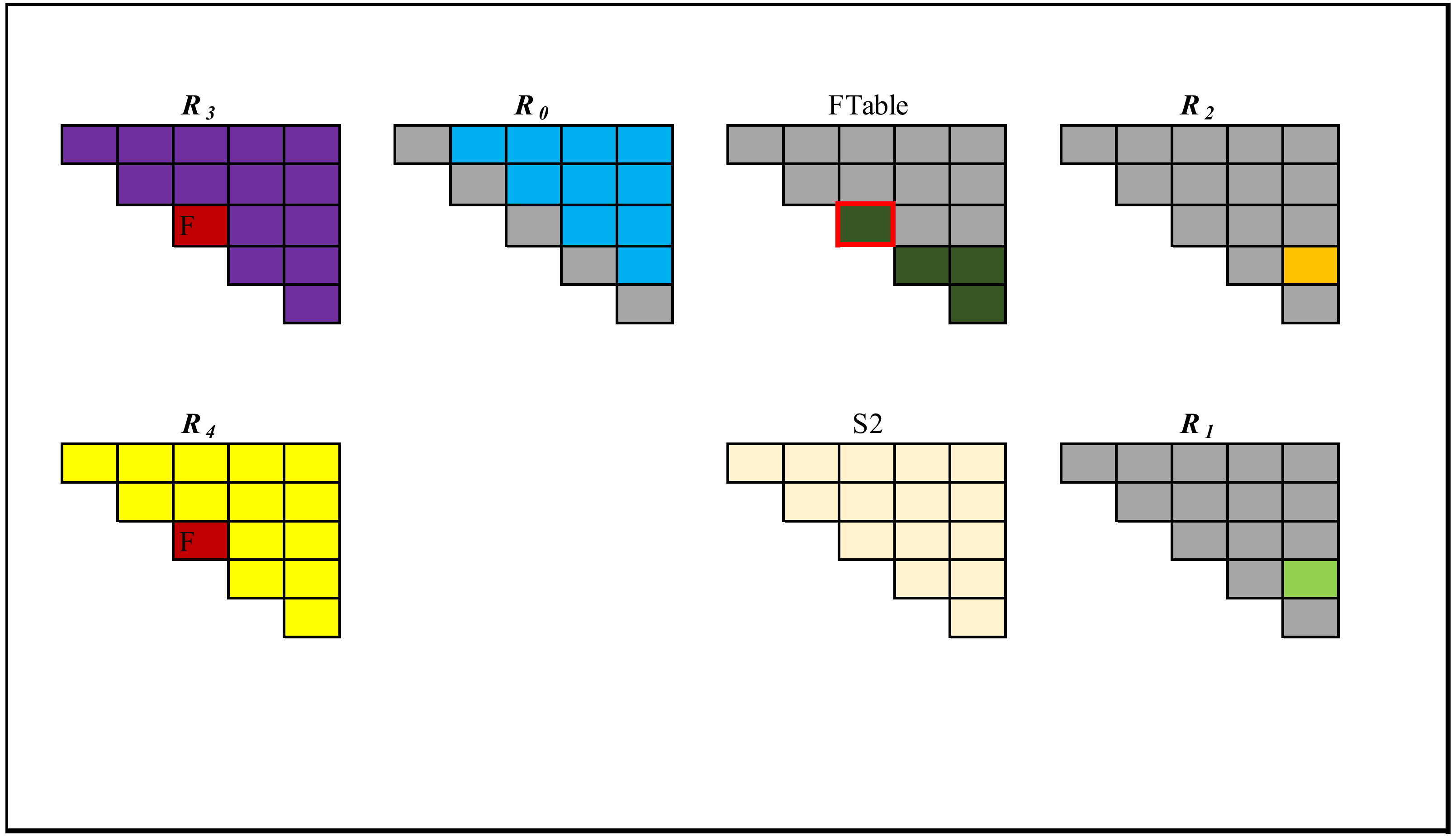


FTable $[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; j_2, \; 0 \,]$

R_1 $[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$

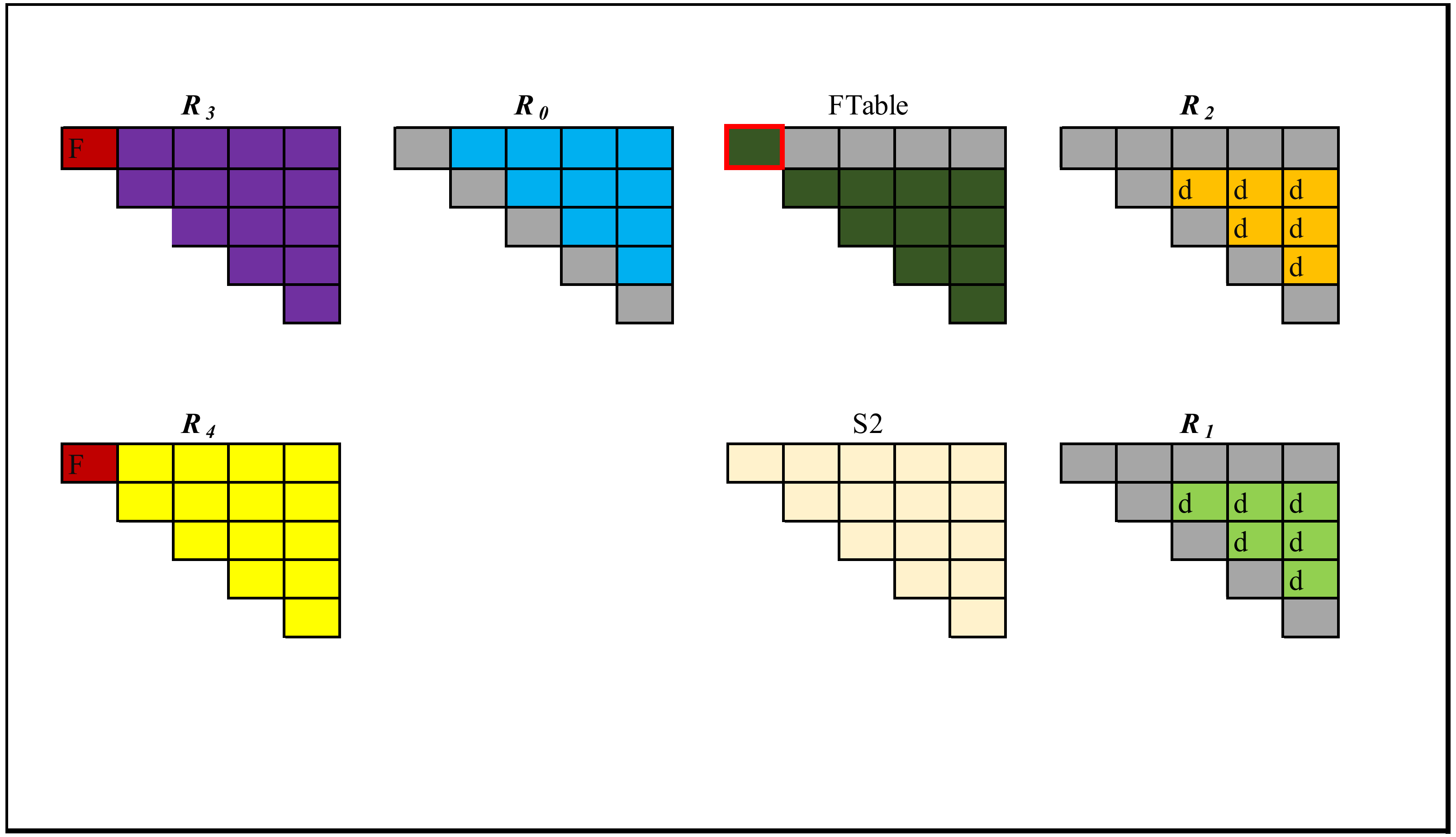
R_2 $[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$

Scheduling R_1, R_2, and F-Table



FTable	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; j_2, \; 0 \,]$
R_1	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$
R_2	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$

Scheduling R_1, R_2, and F-Table

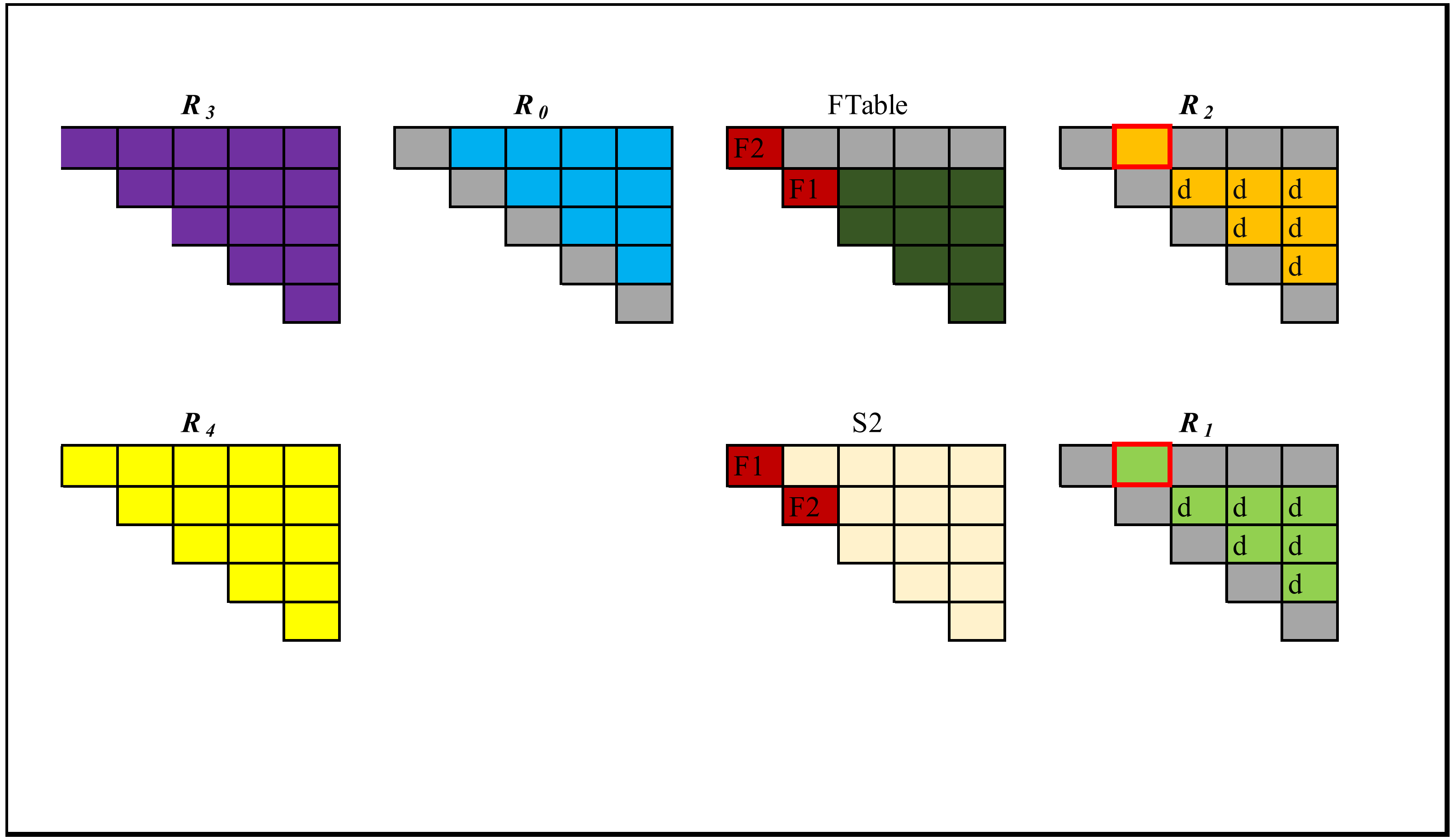


FTable $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ j_2,\ 0 \,]$

R_1 $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2 \,]$

R_2 $[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2 \,]$

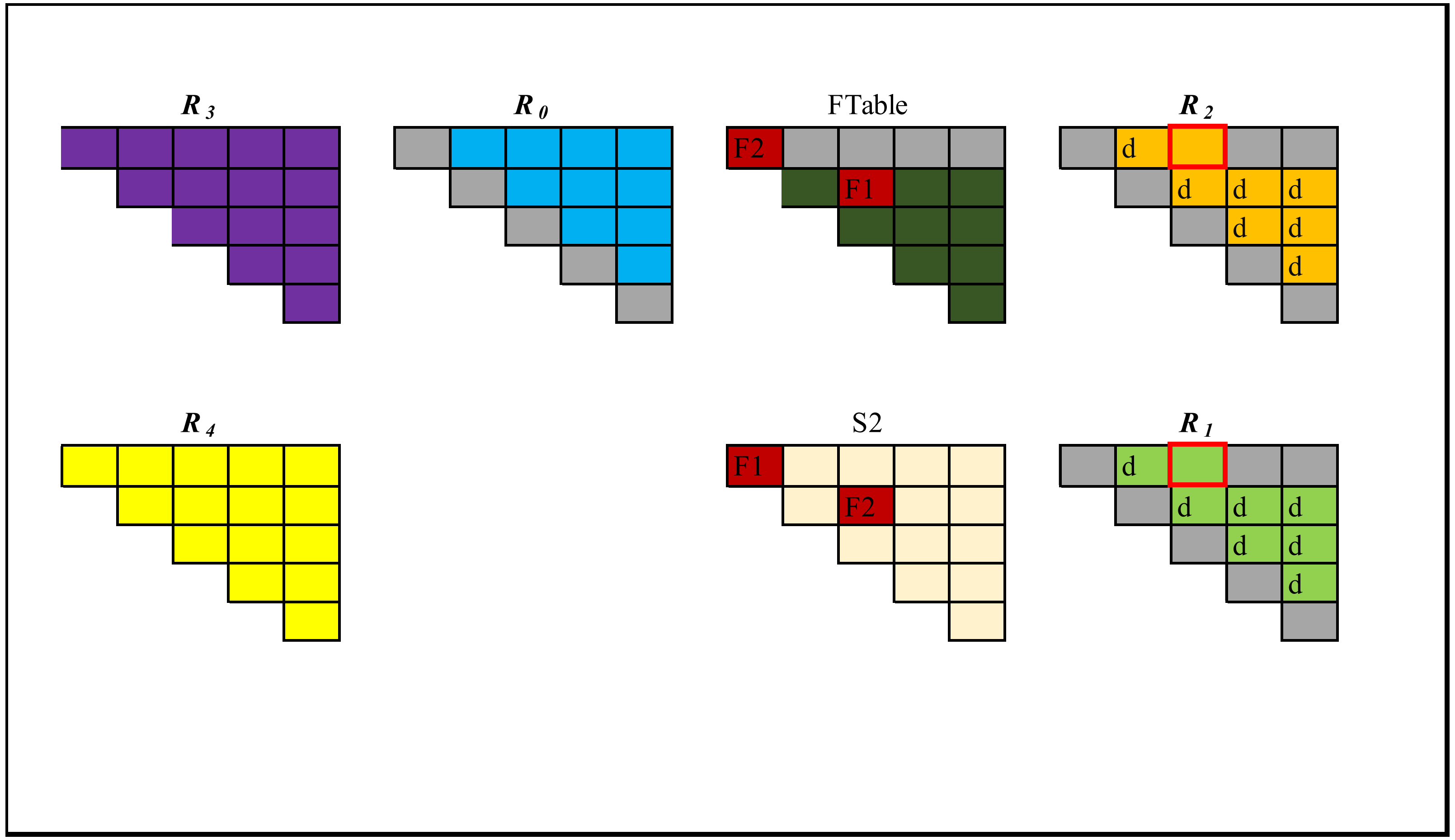
Scheduling R_1, R_2, and F-Table



R_1, R_2, computations are dependent on F-Table

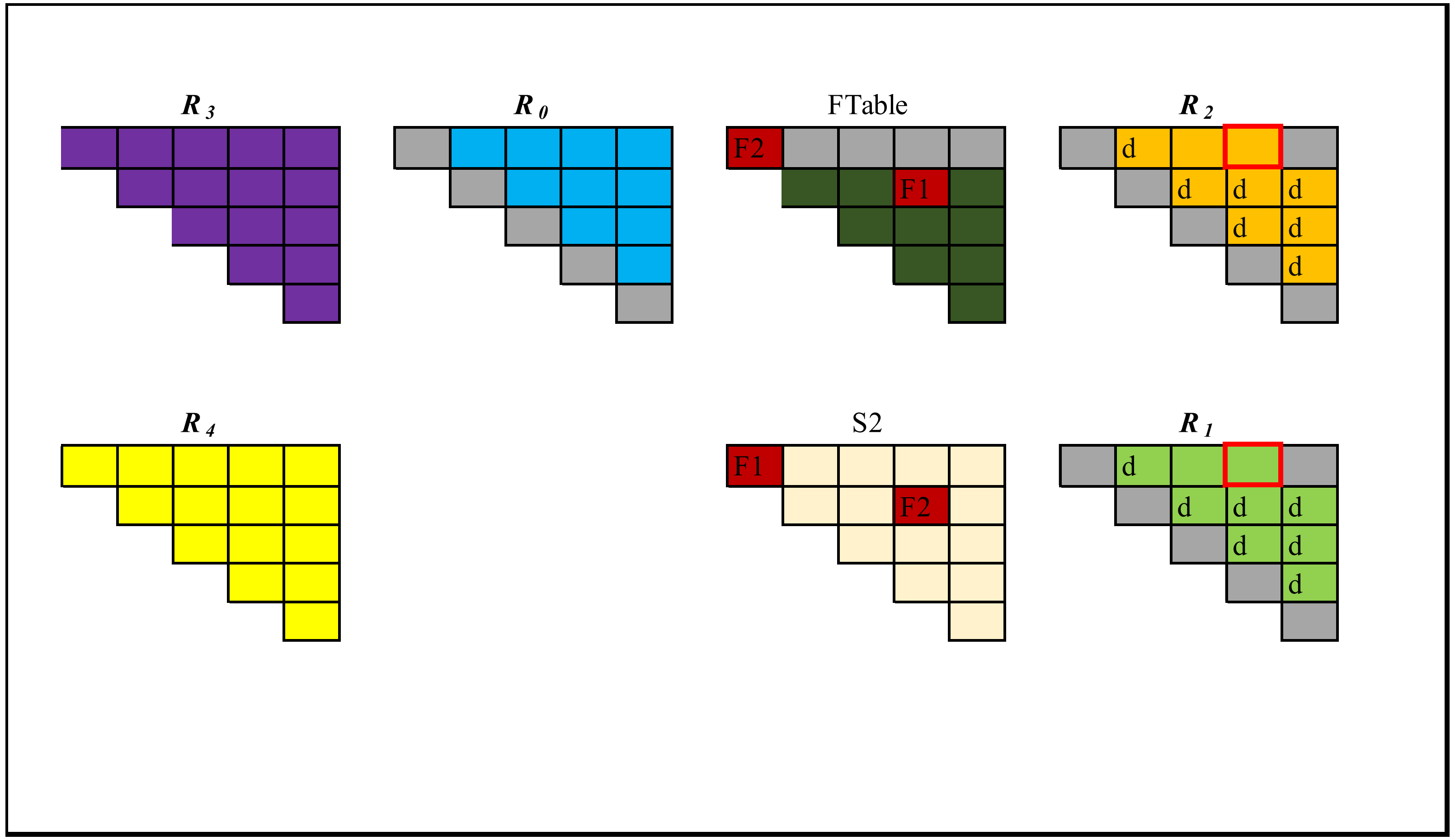
FTable	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, j_2, 0 \,]$
$\mathbf{R_1}$	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2]$
$\mathbf{R_2}$	$[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2]$

Scheduling R_1, R_2, and F-Table



FTable	$[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ j_2,\ 0 \,]$
$\mathbf{R_1}$	$[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$
$\mathbf{R_2}$	$[\, i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$

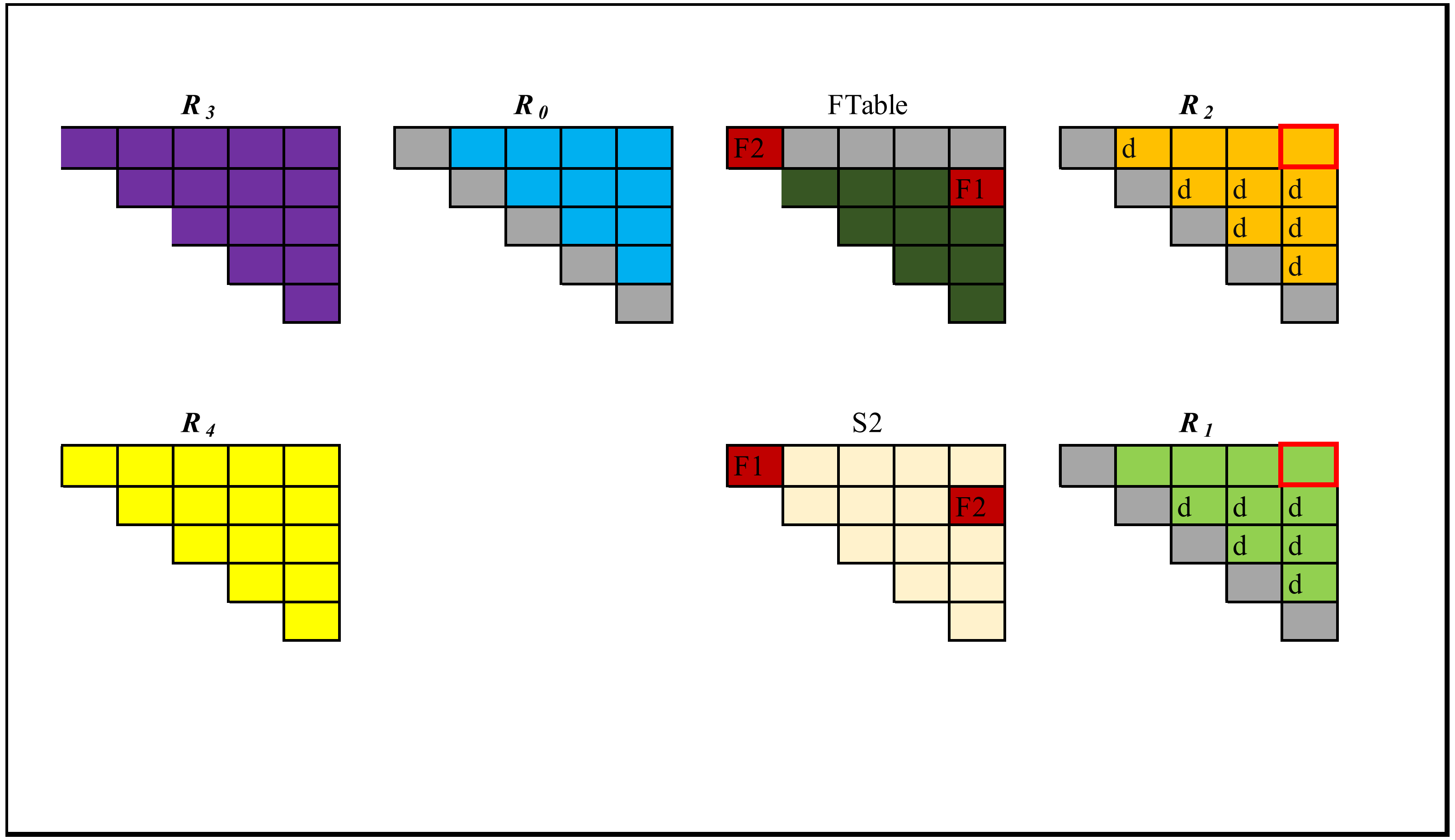
Scheduling R_1, R_2, and F-Table



R_1, R_2, also takes advantage of the auto vectorization.

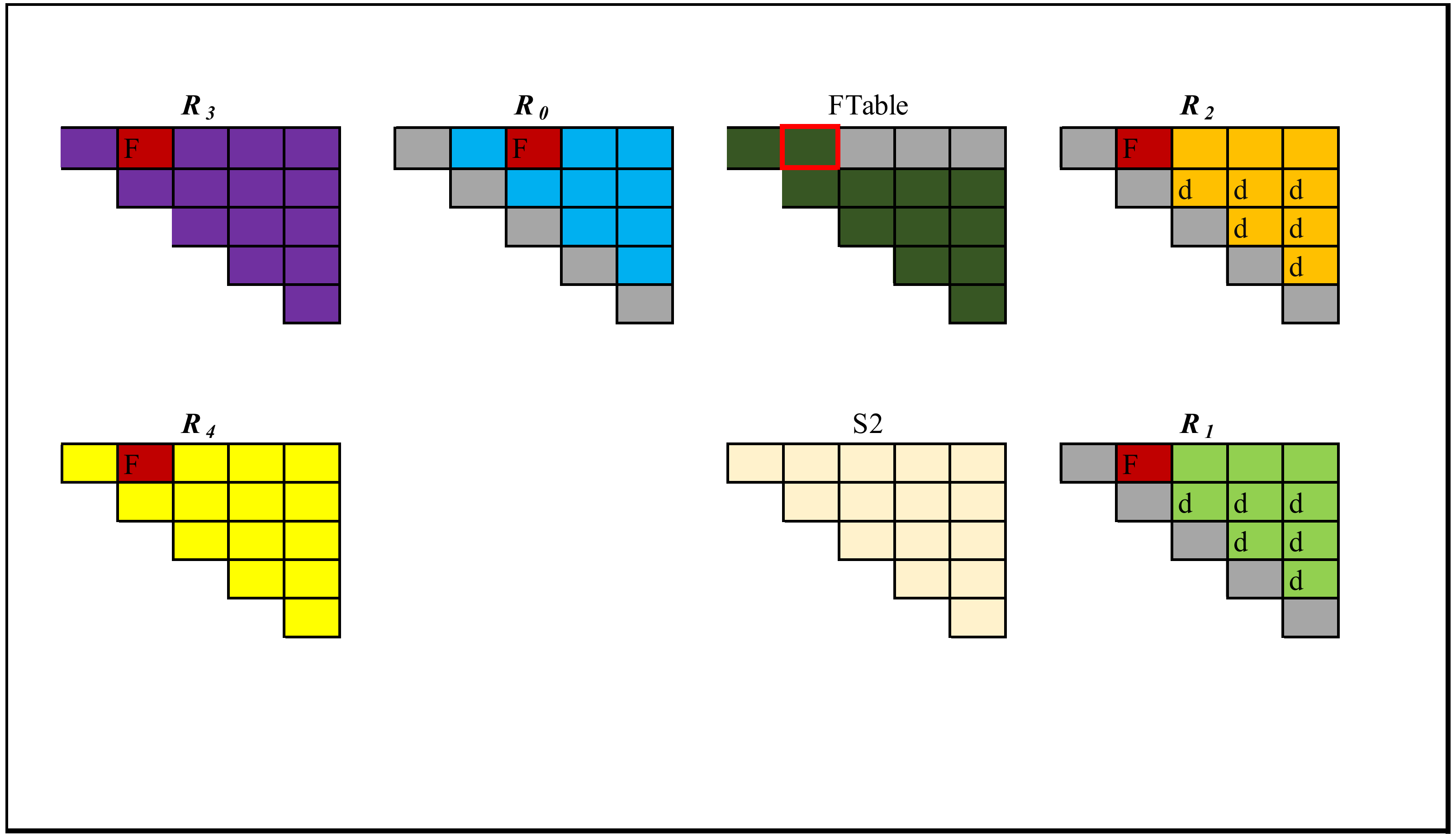
FTable	[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, j_2, 0$]
R_1	[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2$]
R_2	[$i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2$]

Scheduling R_1, R_2, and F-Table



FTable	[i_1, j_1, i_2, j_2 →	*X*,	*Y*,	*Z*,	$-i_2$,	j_2 ,	0]
$\mathbf{R_1}$	[i_1, j_1, i_2, j_2 →	*X*,	*Y*,	*Z*,	$-i_2$,	k_2,	j_2]
$\mathbf{R_2}$	[i_1, j_1, i_2, j_2 →	*X*,	*Y*,	*Z*,	$-i_2$,	k_2,	j_2]

Scheduling R_1, R_2, and F-Table

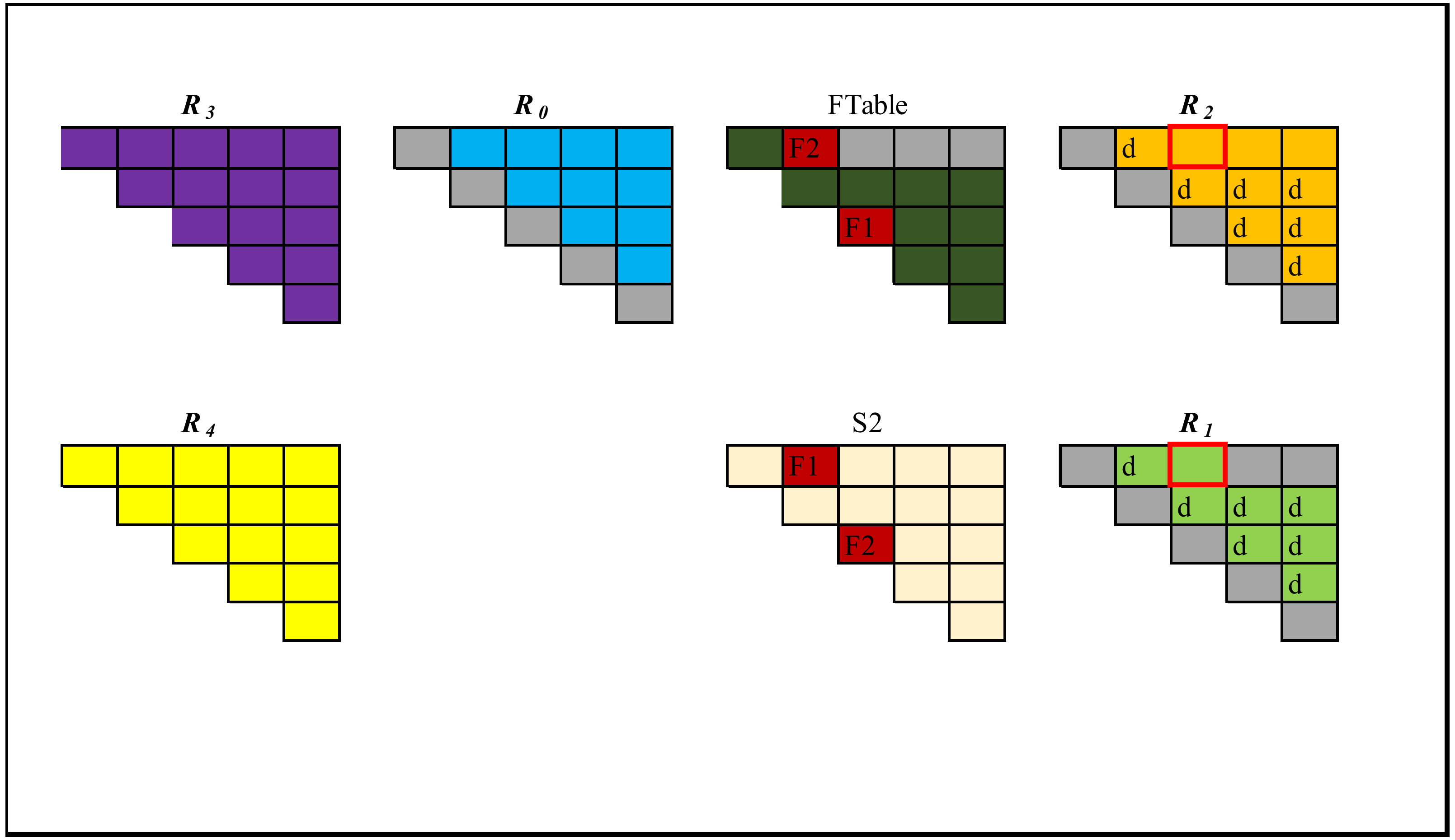


FTable $[\ i_1, j_1, i_2, j_2 \rightarrow X,\ \ Y,\ \ Z,\ \ -i_2,\ \ j_2,\ \ 0\ ]$

R_1 $[\ i_1, j_1, i_2, j_2 \rightarrow X,\ \ Y,\ \ Z,\ \ -i_2,\ \ k_2,\ \ j_2]$

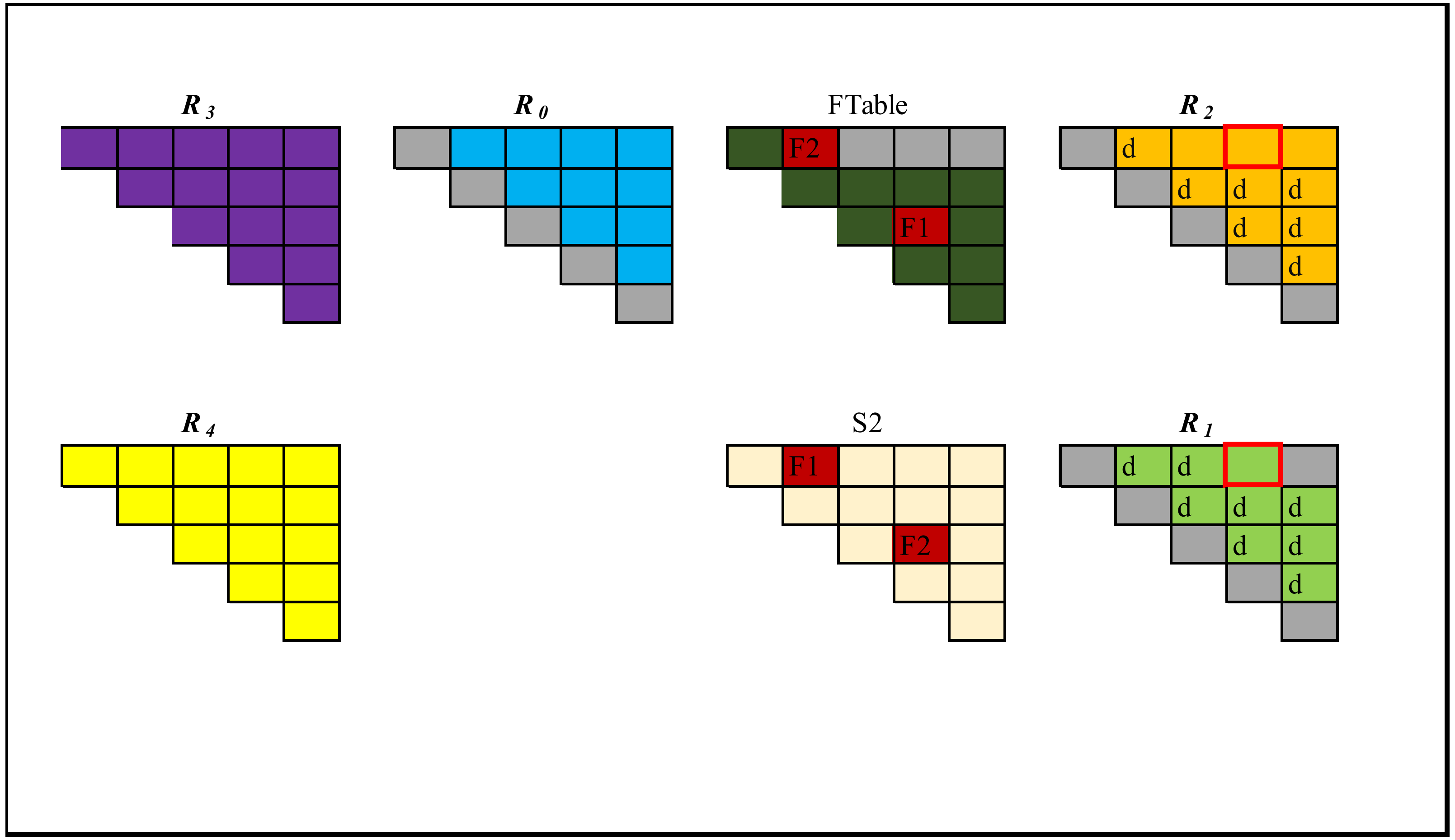
R_2 $[\ i_1, j_1, i_2, j_2 \rightarrow X,\ \ Y,\ \ Z,\ \ -i_2,\ \ k_2,\ \ j_2]$

Scheduling R_1, R_2, and F-Table



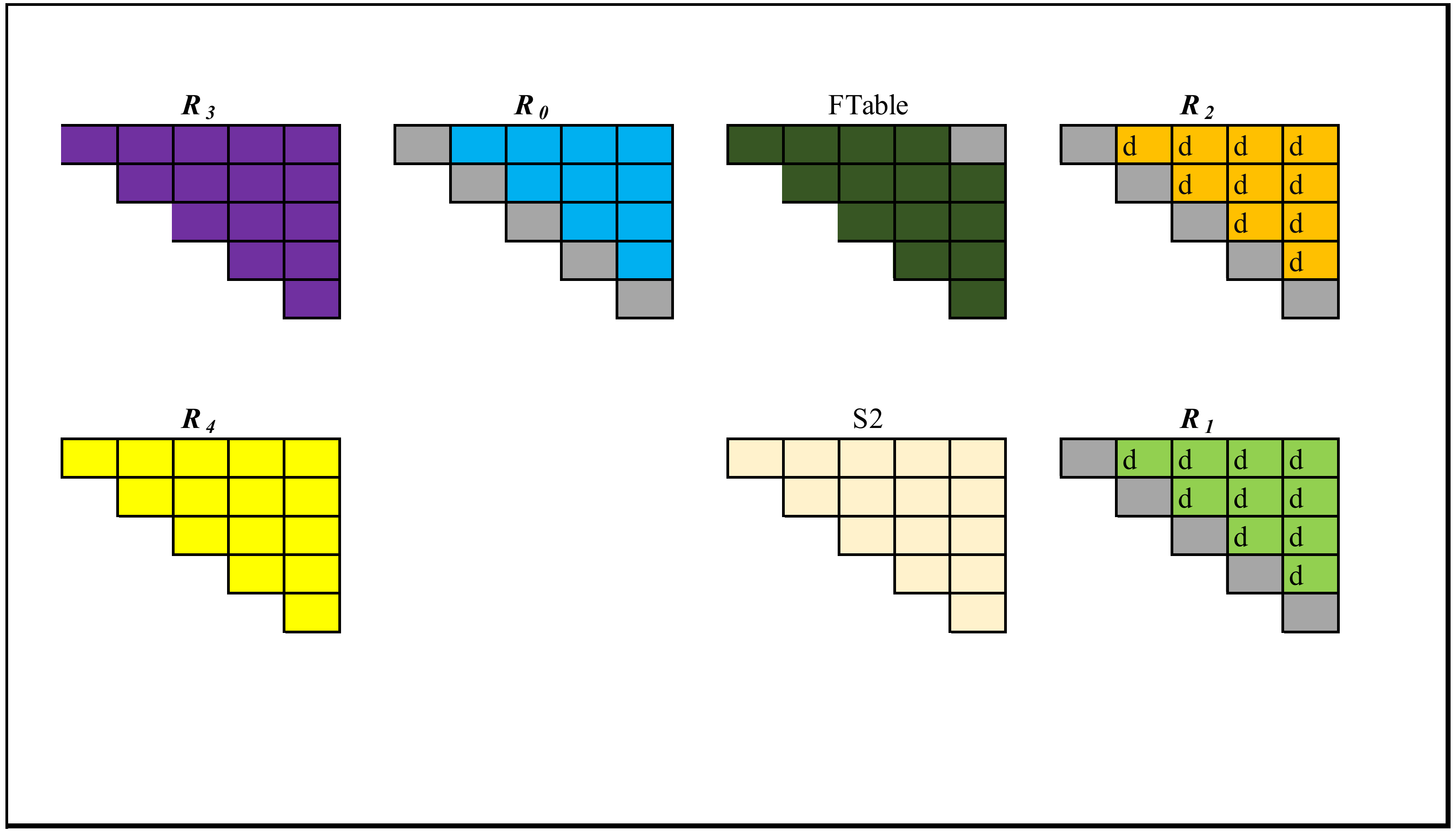
FTable	$[\ i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ j_2,\ 0\ ]$
$\mathbf{R_1}$	$[\ i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$
$\mathbf{R_2}$	$[\ i_1, j_1, i_2, j_2 \rightarrow X,\ Y,\ Z,\ -i_2,\ k_2,\ j_2]$

Scheduling R_1, R_2, and F-Table



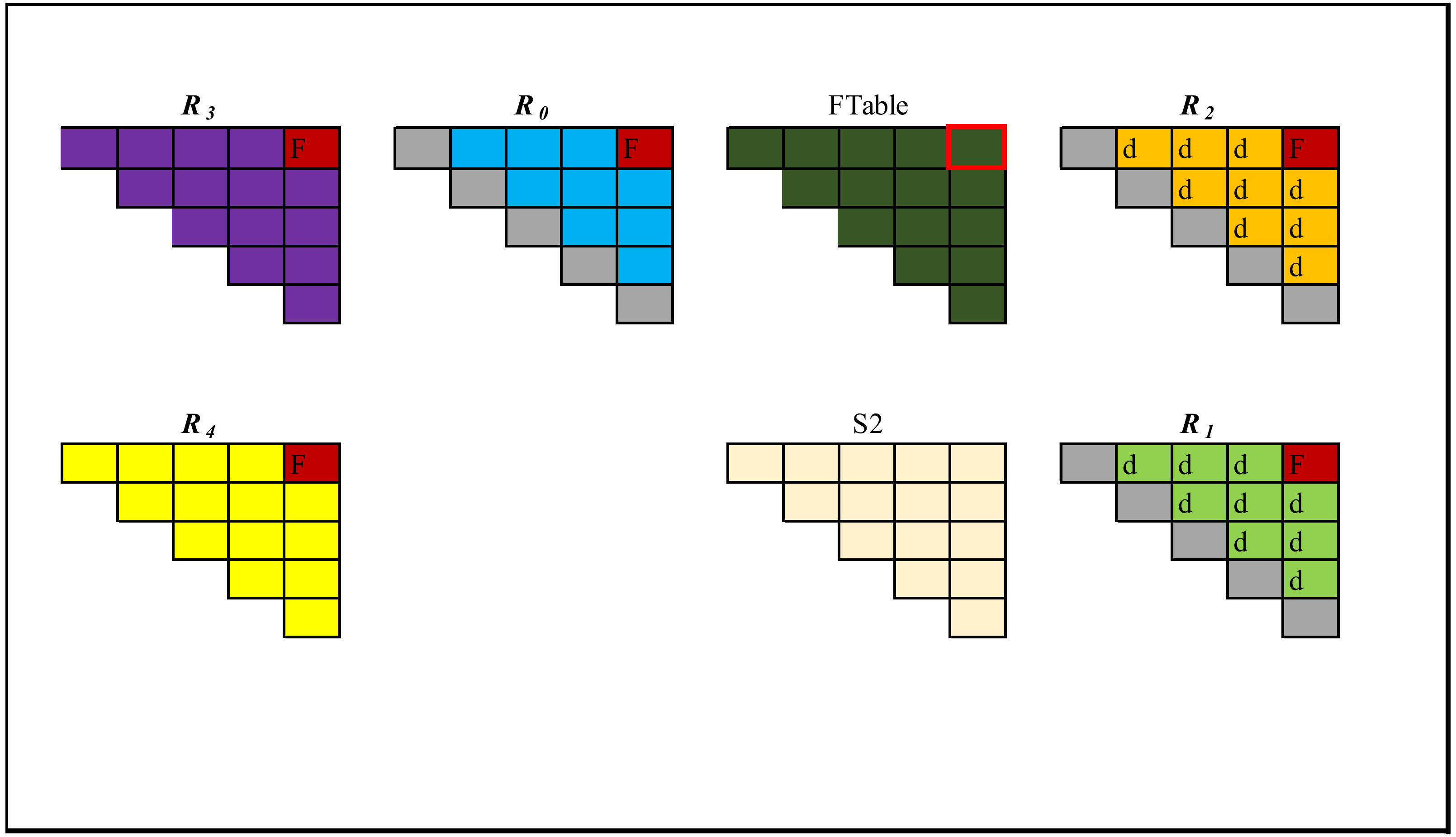
FTable	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; j_2, \; 0 \,]$
R_1	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$
R_2	$[\, i_1, j_1, i_2, j_2 \rightarrow X, \; Y, \; Z, \; -i_2, \; k_2, \; j_2]$

Scheduling R_1, R_2, and F-Table



FTable	**[i_1, j_1, i_2, j_2 →**	***X*,**	***Y*,**	***Z*,**	**$-i_2$,**	**j_2 ,**	**0]**
R_1	**[i_1, j_1, i_2, j_2 →**	***X*,**	***Y*,**	***Z*,**	**$-i_2$,**	**k_2,**	**j_2]**
R_2	**[i_1, j_1, i_2, j_2 →**	***X*,**	***Y*,**	***Z*,**	**$-i_2$,**	**k_2,**	**j_2]**

Scheduling R_1, R_2, and F-Table



Objective is to find such optimized schedules which increase resource utilization without changing program semantics

FTable $[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, j_2, 0 \,]$

R_1 $[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2 \,]$

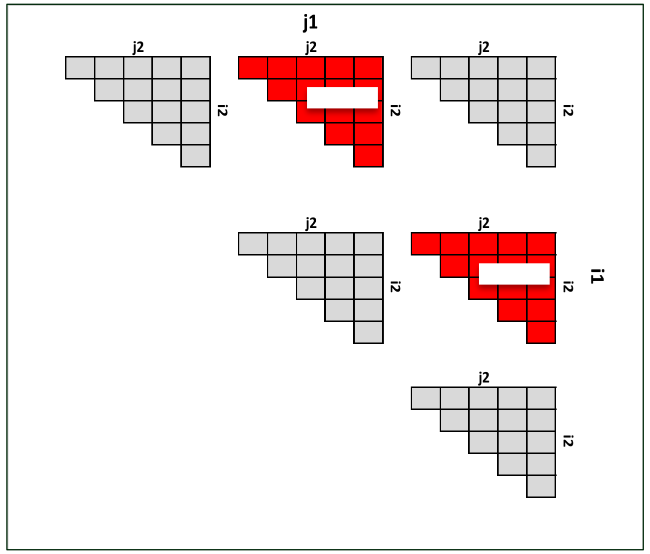
R_2 $[\, i_1, j_1, i_2, j_2 \rightarrow X, Y, Z, -i_2, k_2, j_2 \,]$

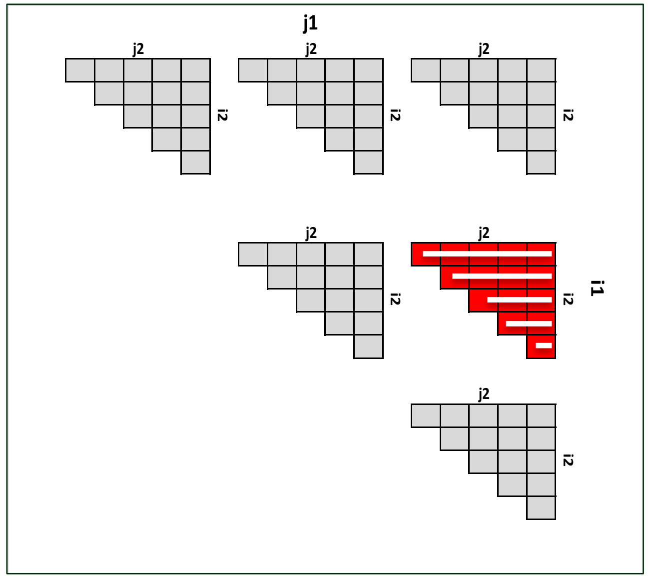
Parallelization Approach

- Coarse-grain
  - Multiple F-Table[i_1, j_1] elements are computed simultaneously
    - Poor memory reuse
    - Lot of cache misses for double max-plus computations

- Fine-grain
  - Multiple cores/threads computing one inner triangle - FTable[i_1, j_1]
    - Only R_0, R_3, and R_4 computations are parallelized
      - Low processor utilization

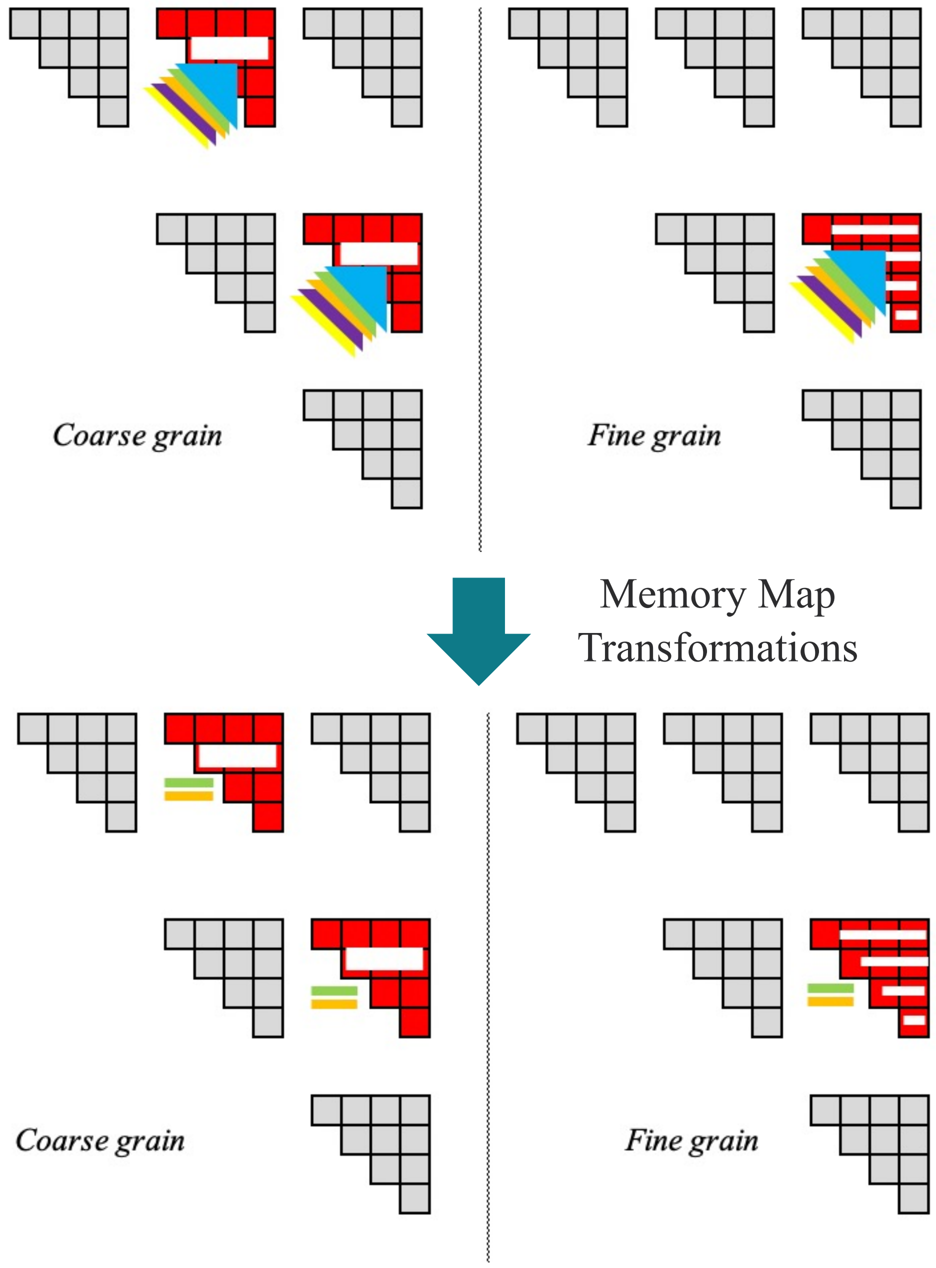
- Hybrid Schedule
  - We use the fine-grain parallelism for R_0, R_3, R_4 and the coarse-grain parallelism for F-Table, R_1, R_2





Memory Optimization

- Memory-overhead of ALPHAZ generated code is $M^2 \times N^2$
  - However, we only need one-fourth of that memory. Not too problematic
  - But reduction variables also take up memory space by default, which is wasteful
- Each inner triangle requires 5 2-D array for each reduction variables to be active in memory for each thread
- R_0, R_3 and R_4 are always computed before final F-table update
  - Can share the memory with F-Table
- Single row of an inner triangle is required for R_1 and R_2 to keep up with the F-Table update



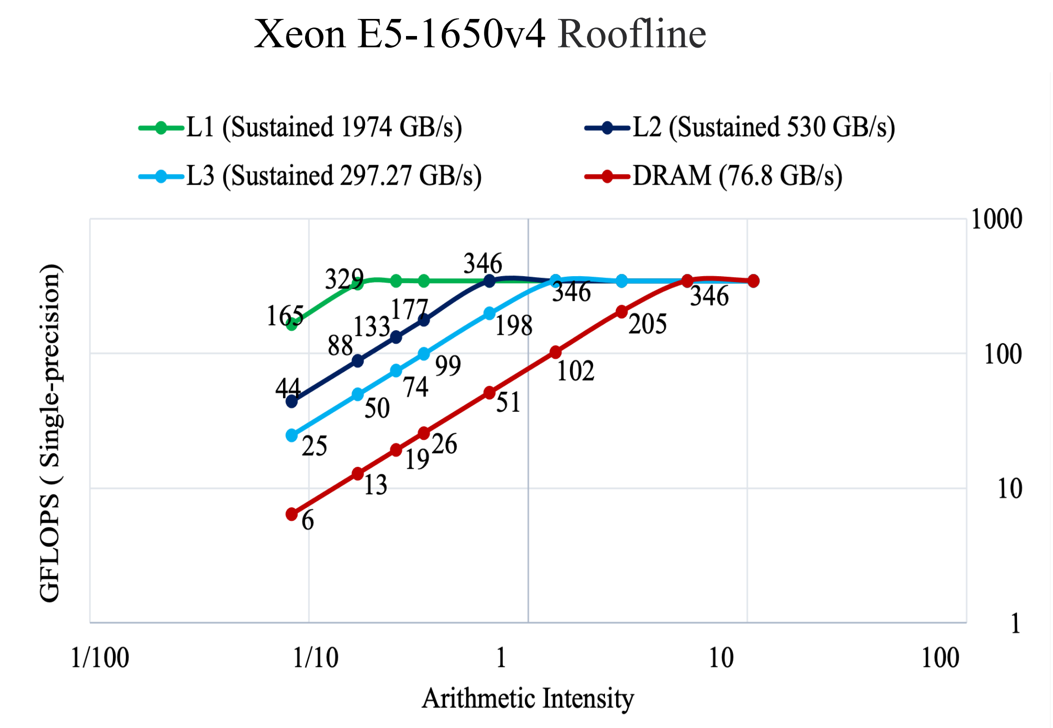
Results

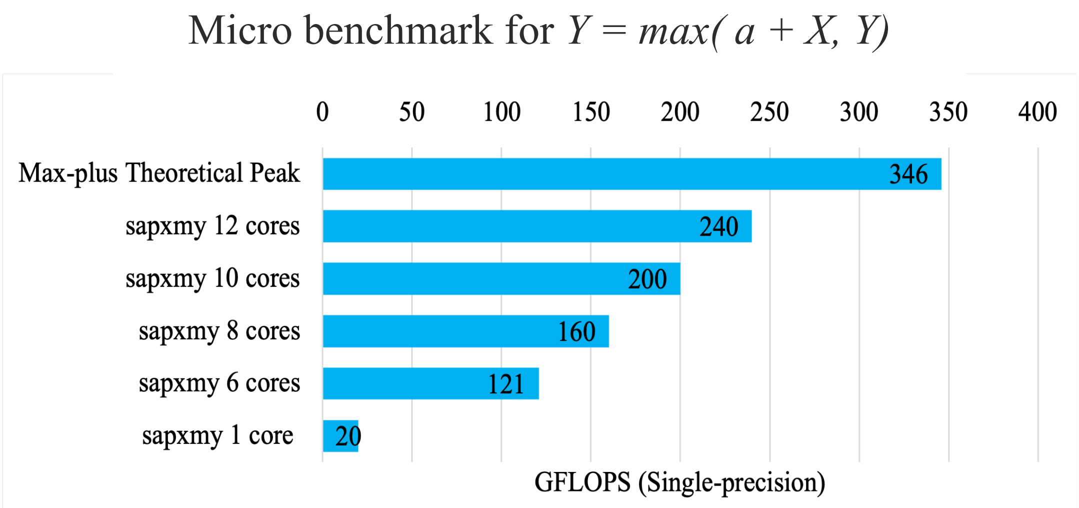
Performance Goal

- We use Xeon E5-1650v4 to present our optimization result
  - Theoretical max-plus machine peak is about 346 GFLOPS

CPU type	Frequency	Number of Cores	Level-1 (KB)	Level-2 (KB)	Level-3 (shared) (MB)	Theoretical Max Plus Peak (GFLOPs)
CPU - XeonE5-1650v4	6x3.6Ghz	6	6x32	6x256	15 MB	346

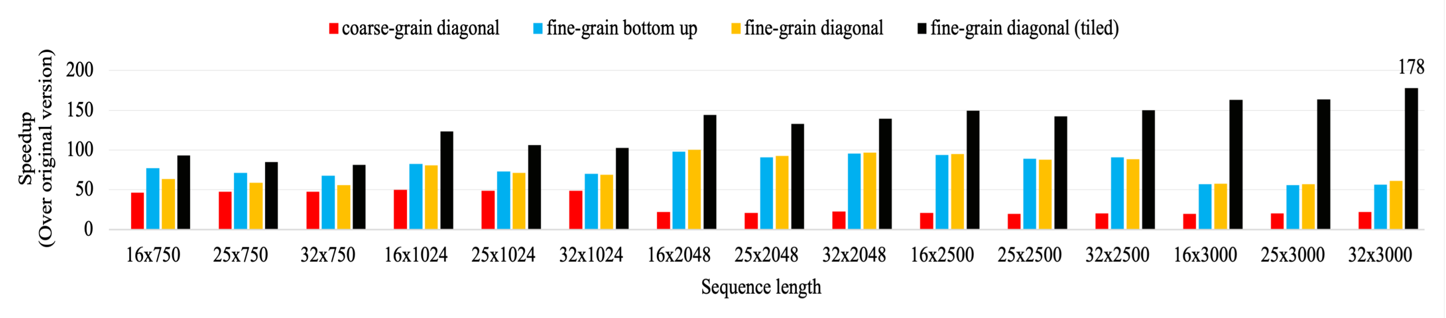
- Arithmetic intensity of BPMax 1/6
  - 2-arithmetic operations for 3-single-precision memory operations
  - Based on the roofline model, this translates to 329 GFLOPS for programs with similar arithmetic intensity
- Streaming Bandwidth
  - BPMax data access pattern - Y = max(a + X, Y)
  - Micro-benchmark estimation for the attainable L1 streaming bandwidth
    - 120 GFLOPS for 6 threads

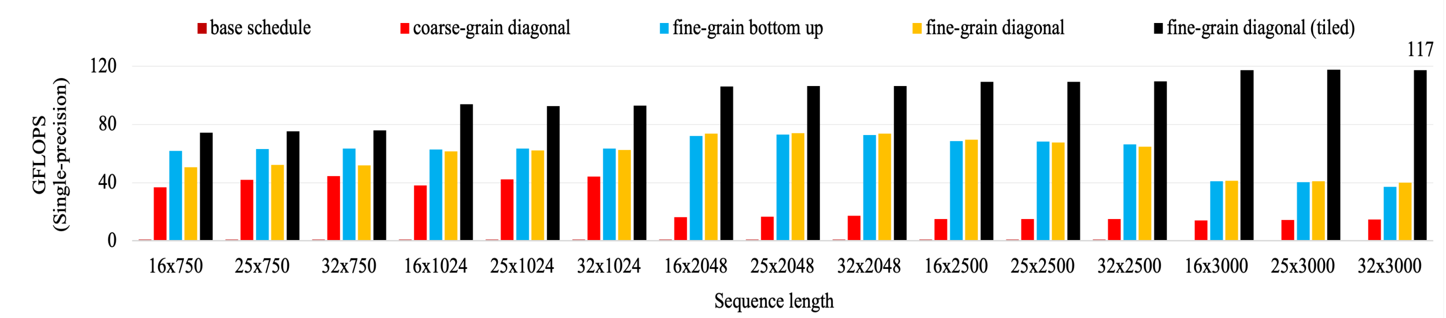




Double Max-plus Improvements

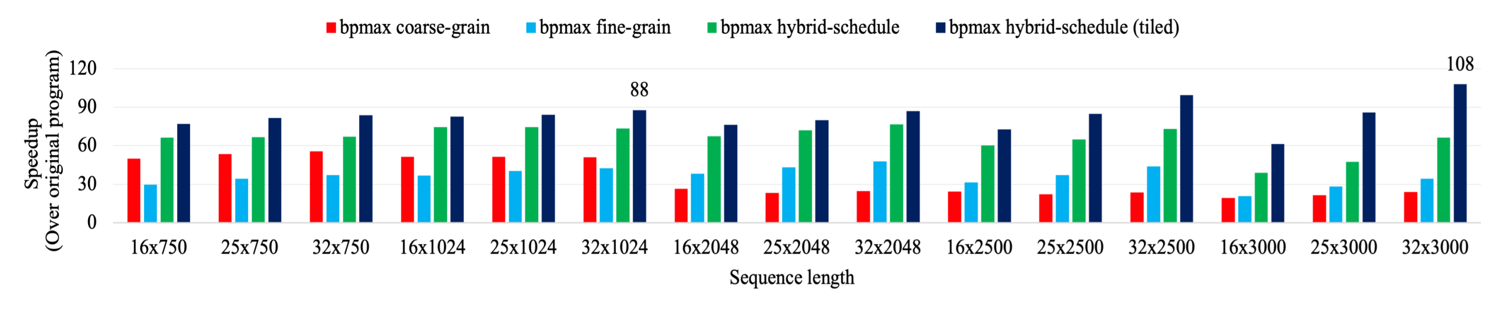
- Coarse-grain parallelization performs very poorly
  - Generates a lot of data movement between different levels of cache and makes the program slower
- Fine-grain parallelization performs better
  - There is a minor difference between computing the inner triangles of F-Table diagonally vs. bottom-up
  - In both cases, all the threads work on one inner triangle before moving to the next

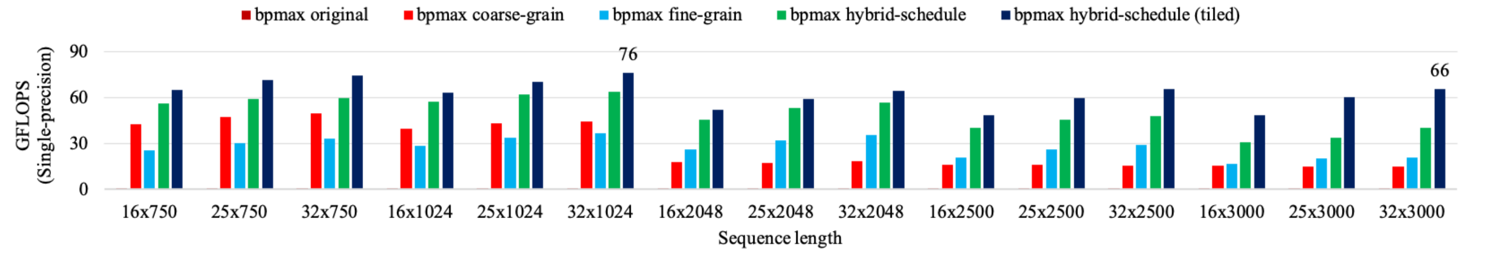




- Tiling approach improves locality, maintains automatic vectorization
  - Attains 117 GFLOPS with the tiling transformation. 97 % of our microbenchmark target
  - Tile dimensions of (32×4×N) and (64 × 16 × N) are used for presenting the performance and speedup comparison
    - (32 × 4 × N) is restricted for sequence length up to 2048

BPMax Improvements



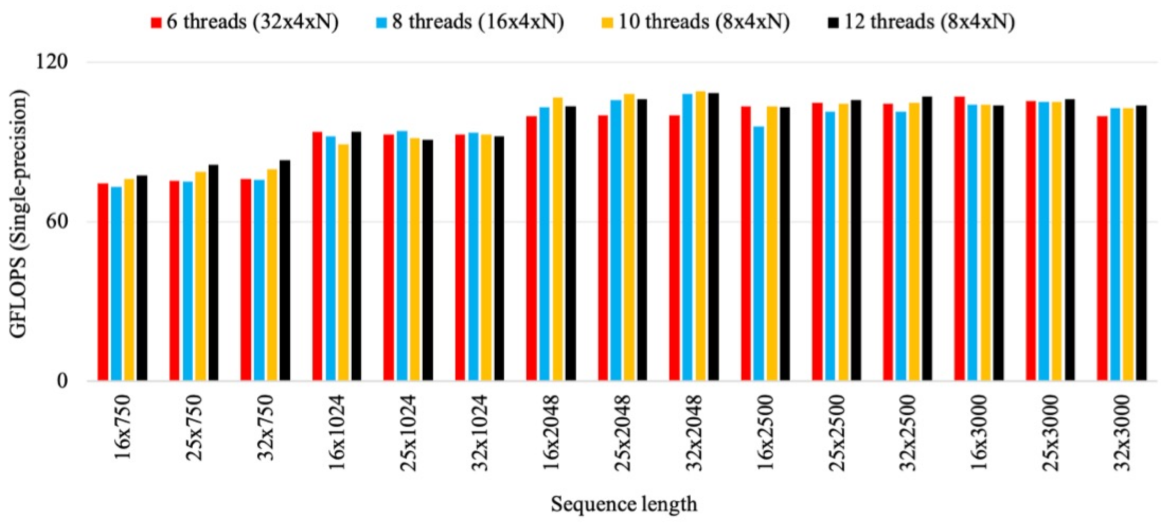


- Coarse-grain version performs worst
  - Severe impact on double max-plus computation
- Fine-grain version performs better
- Hybrid parallelization approach highlighted in green performs better than the coarse and fine-grain version
- Tiled version of the hybrid schedule highlighted in dark blue performs best
  - It achieves 100× speedup for longer sequence lengths with 6 threads
  - The improvement for the tiled version mainly comes from the optimization of R_0, R_3, R_4
  - The tiled version of the program reaches around 76 GFLOPS for moderate-size sequences
    - It is almost 60% lower than the best double max- plus version of the same sequence
    - Our analysis shows that R_3 and R_4 are almost free since those get computed along with the R_0
    - The other two $\Theta(M^2N^3)$ computations - R_1 and R_2 severely affect the overall performance.

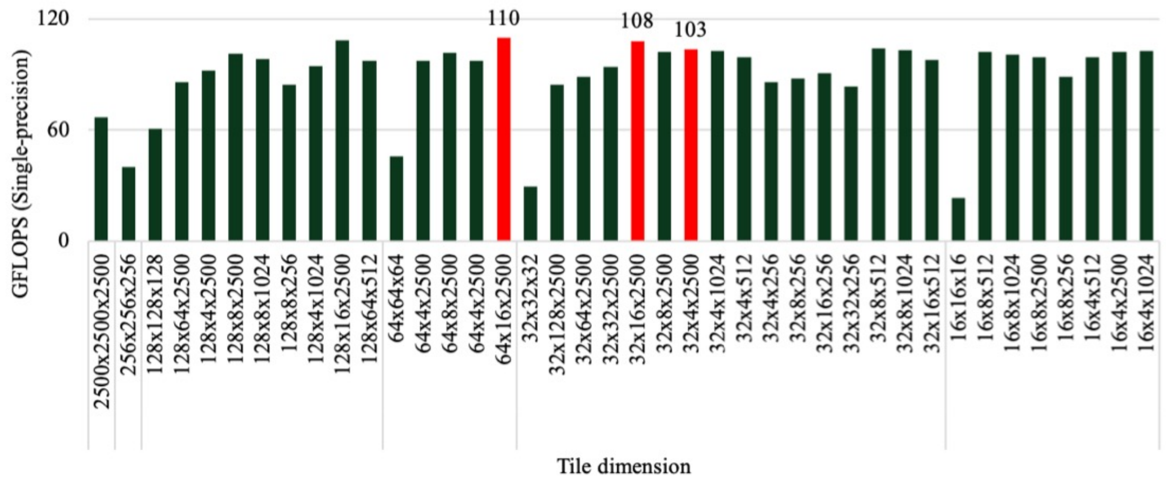
Effect of Hyper-Threading and Tile Size

- Hyper Threading Effect
  - Minimal (3−5%) improvement with hyper-threading over six threads

- Effect of Tiling ($i_2 \times k_2 \times j_2$) Parameters
  - Cubic tiles perform poorly
  - We observe the best result when j_2 is not tiled
    - Due to the streaming effect



Effect of hyper-threading on tiled double max-plus performance



Effect of tiling parameters ($i_2 \times k_2 \times j_2$) on double max-plus performance(sequence length – 16×2500)

Conclusions & Future Directions

- We demonstrate the optimization process of a complete RRI program using polyhedral transformations
  - Achieve significant performance improvements
- Tiling improves the performance of the most dominant part of the computation
- Inner reductions are still inefficient, which limit the overall performance improvement.
  - These computations are difficult to tile
- Double max-plus operation remains bandwidth-bound even after tiling transformation
  - Indicates that an additional level of tiling at the register level is required to make the program compute bound and improve performance
- Distribute the computation over a cluster using MPI

Thank You