Implementing an $O(n^3/\log(n))$ RNA Folding Algorithm

Sophia Chang, David Dang, David Hsiao

RNA Folding

- Nussinov Algorithm
  - 4 Rules
  - $O(n^3)$
- Reordering: Alternative Algorithm
  - $O(n^3)$
- Four-Russians Speedup
  - Pre-computing & reusing values
  - $O(n^3/\log(n))$

Nussinov Algorithm

Initialization:

S[i,i] = 0; i = {0 to n-1}

S[i, i-1] = 0; i = {1 to n-1}

Recursion:

S[i,j] = max {

S[i+1,j-1] + B(i,j) **Rule A**

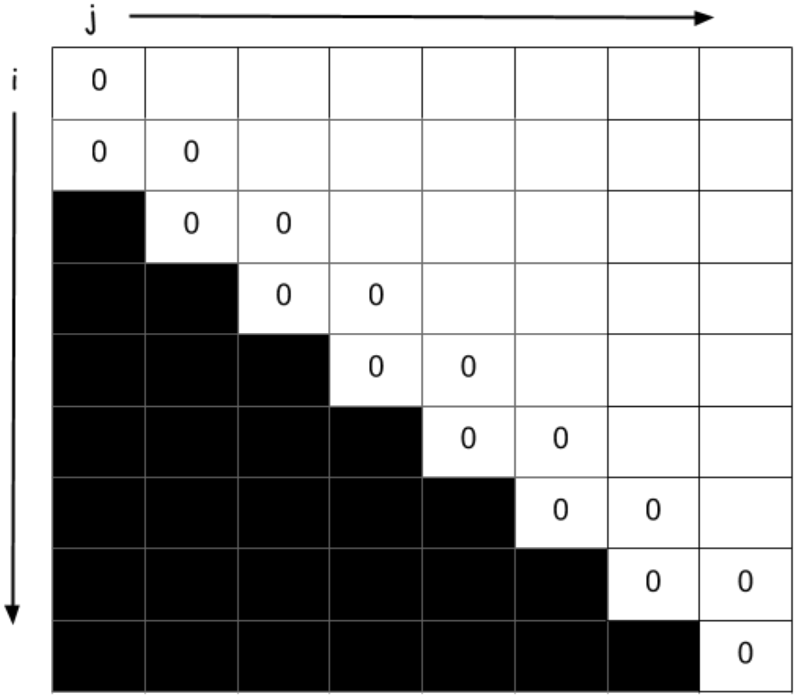
S[i+1,j] **Rule B**

S[i,j-1] **Rule C**

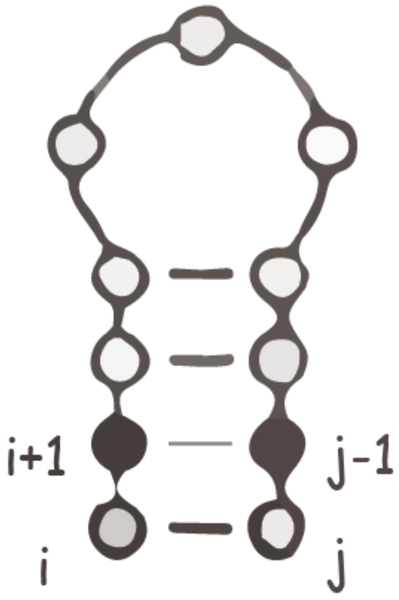
max (S[i,k] + S[k+1,j]) where i<k<j **Rule D**

}

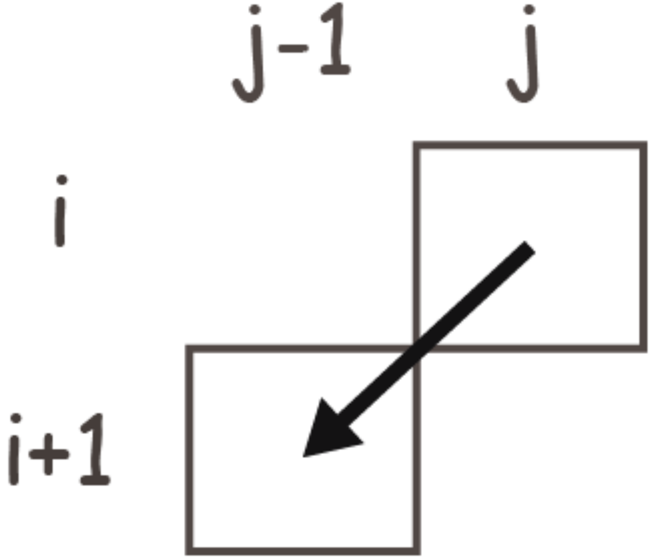
Cost: O(n^3)



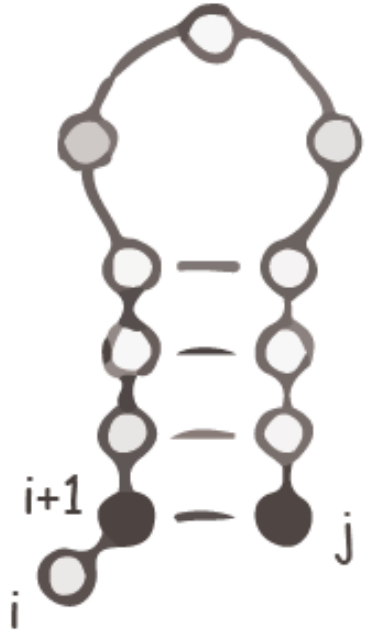
Nussinov: Rule A



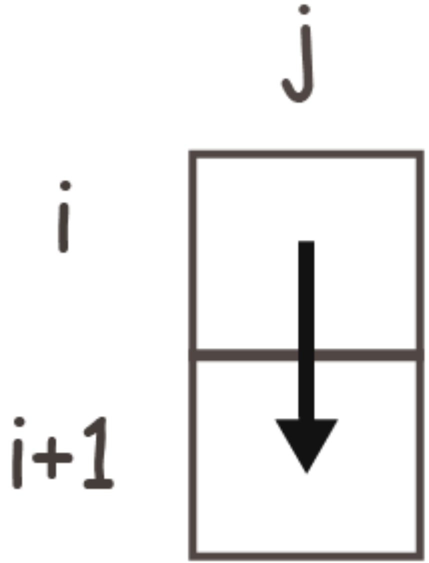
Rule A: i,j pair



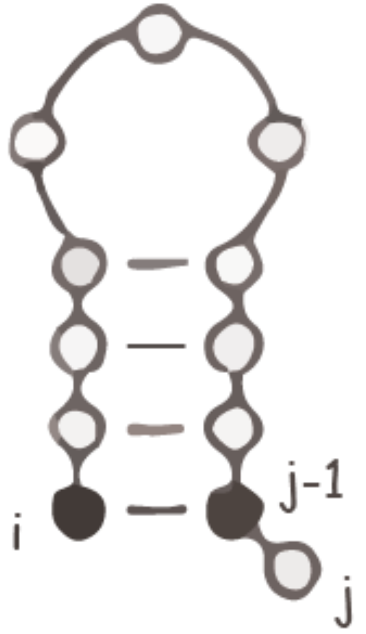
Nussinov: Rule B



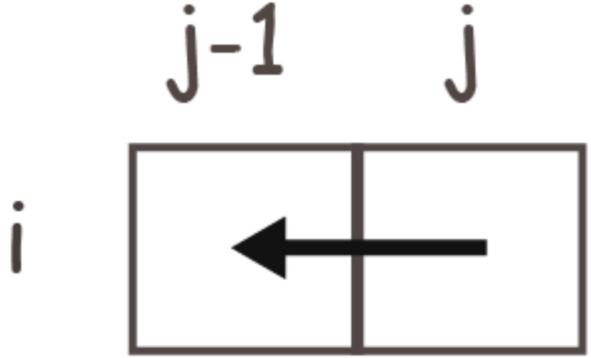
Rule B: i unpaired



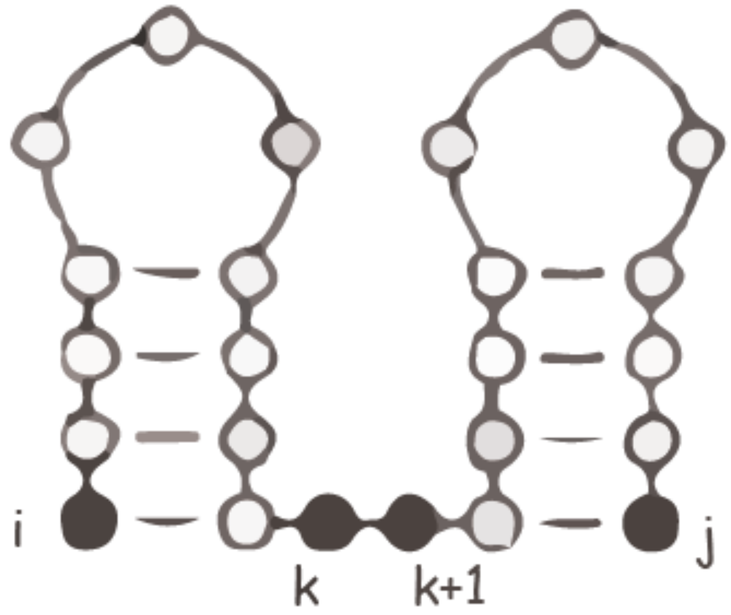
Nussinov: Rule C



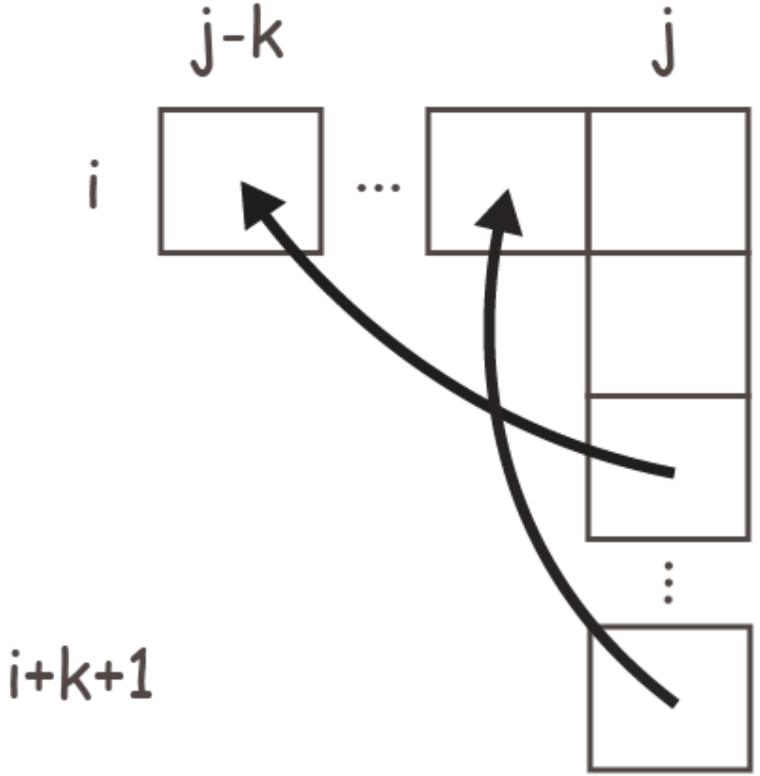
Rule C: j unpaired



Nussinov: Rule D



Rule D: bifurcation



Alternative Algorithm

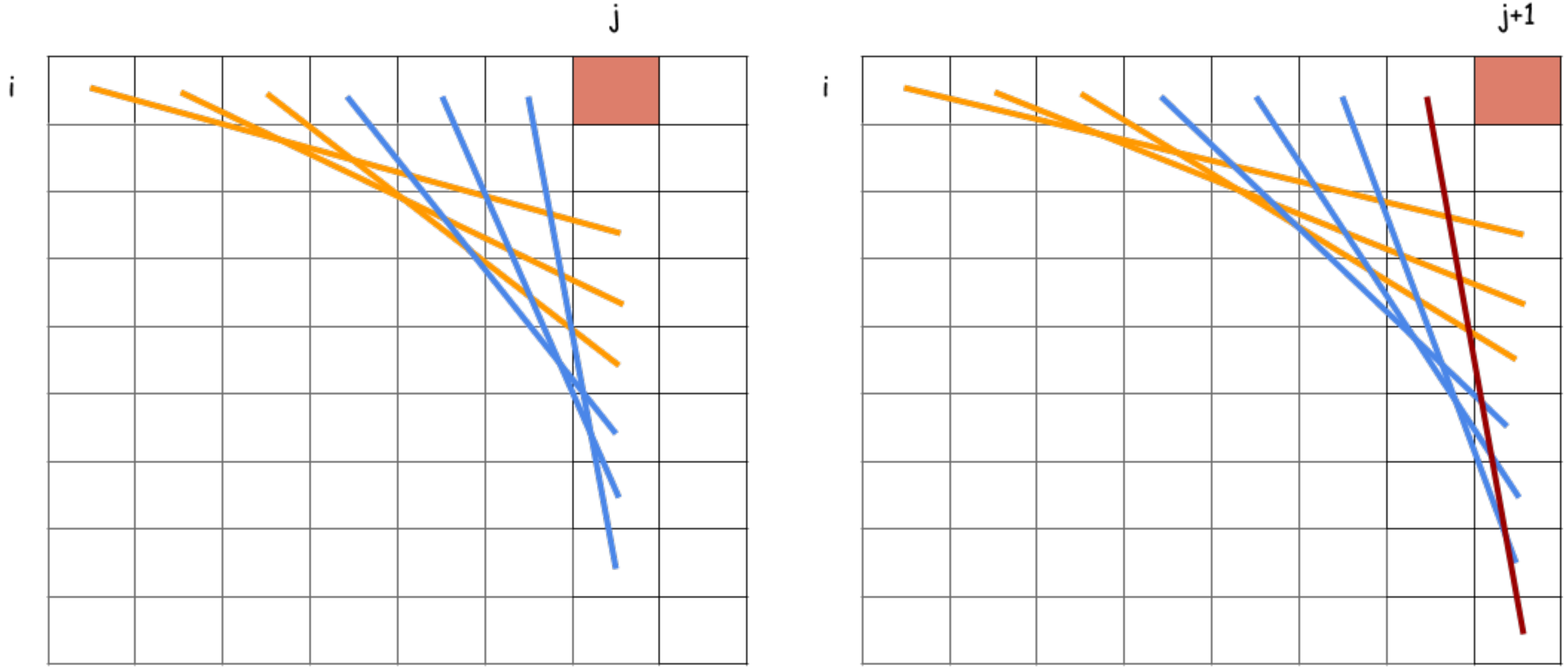
Recursion:

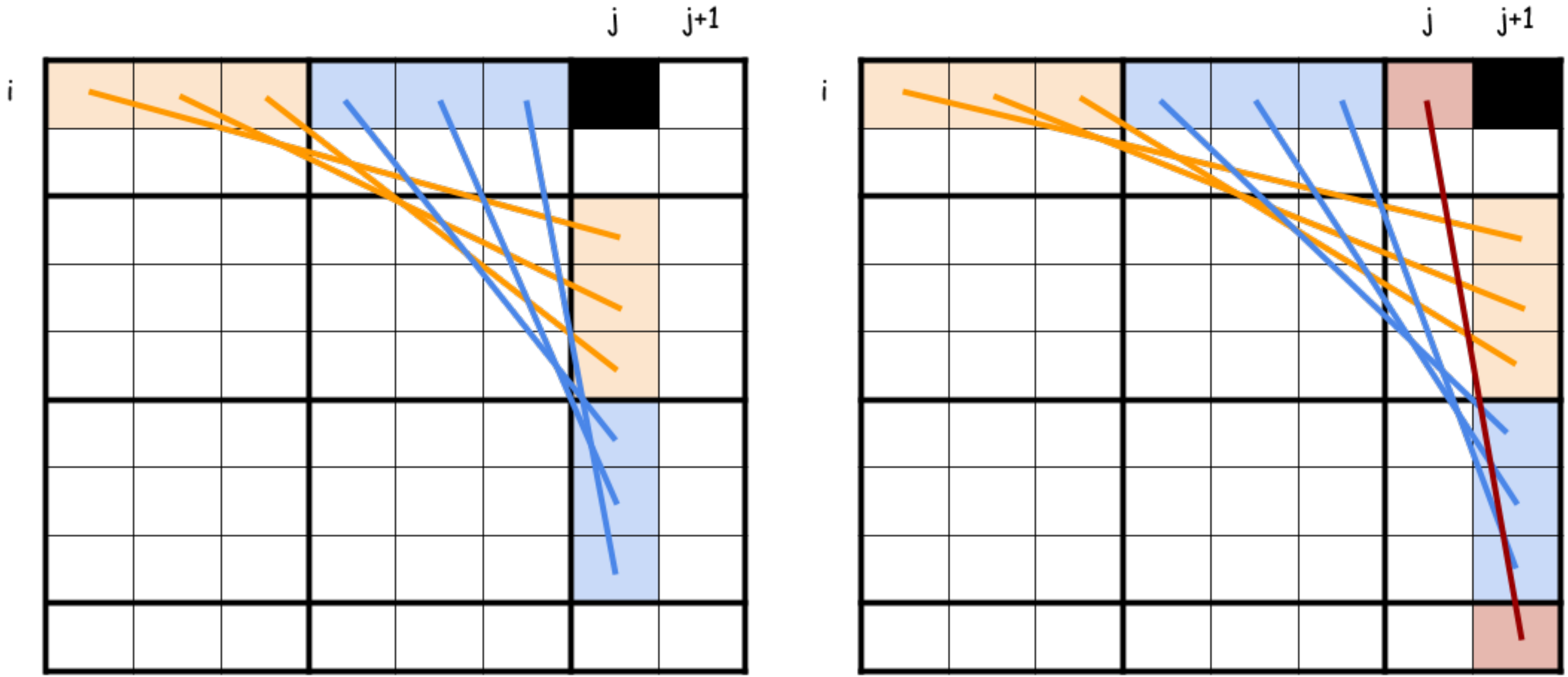
```
for j = 1 to n:
    for i = 0 to j:
        S[i,j] = max(S[i+1,j-1] +B[i,j], S[i,j-1]) Rule A and B
    for i =j-1 to 0:
        S[i,j] = max(S[i+1,j], S[i,j]) Rule C
        for k = j-1 to i:
            S[i,j] = max(S[i,j], S[i,k-1] + S[k,j]) Rule D
```

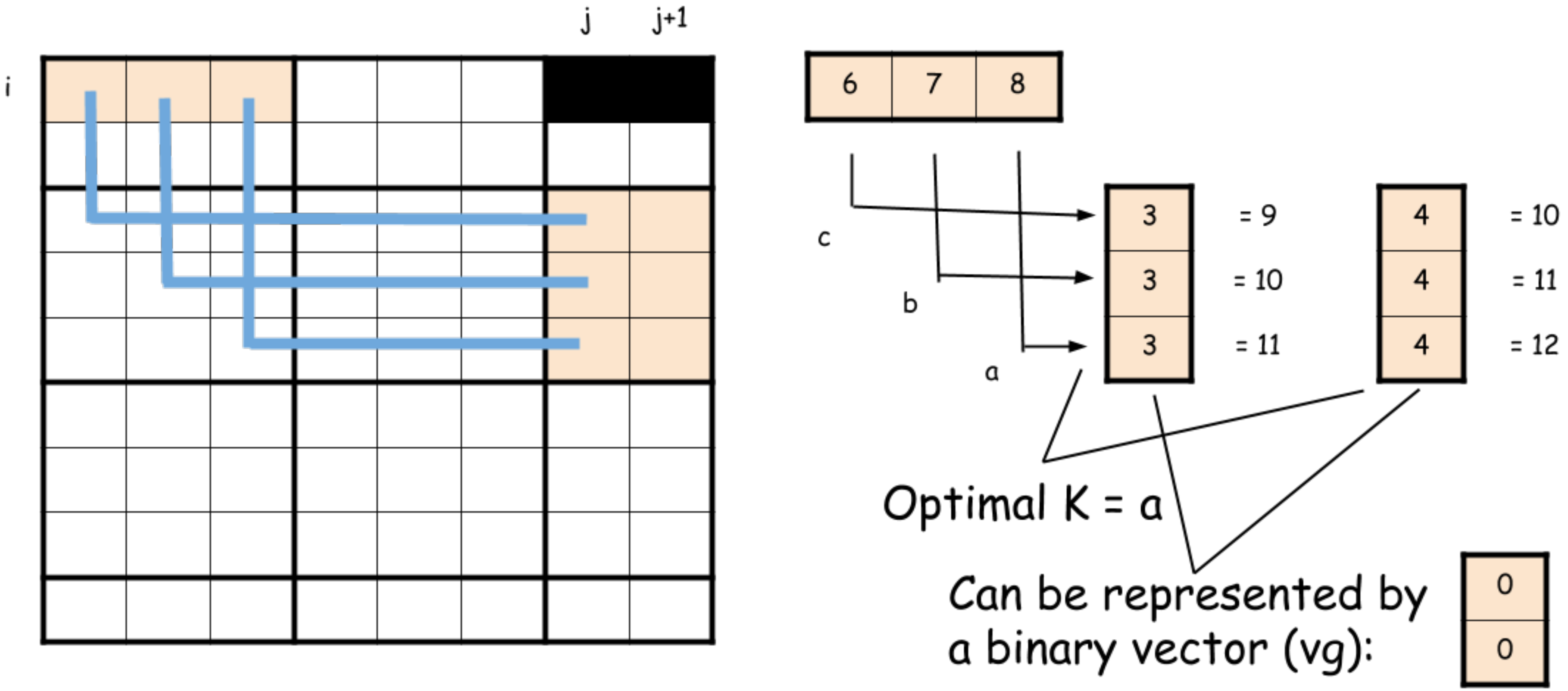
Cost: O(n^3)

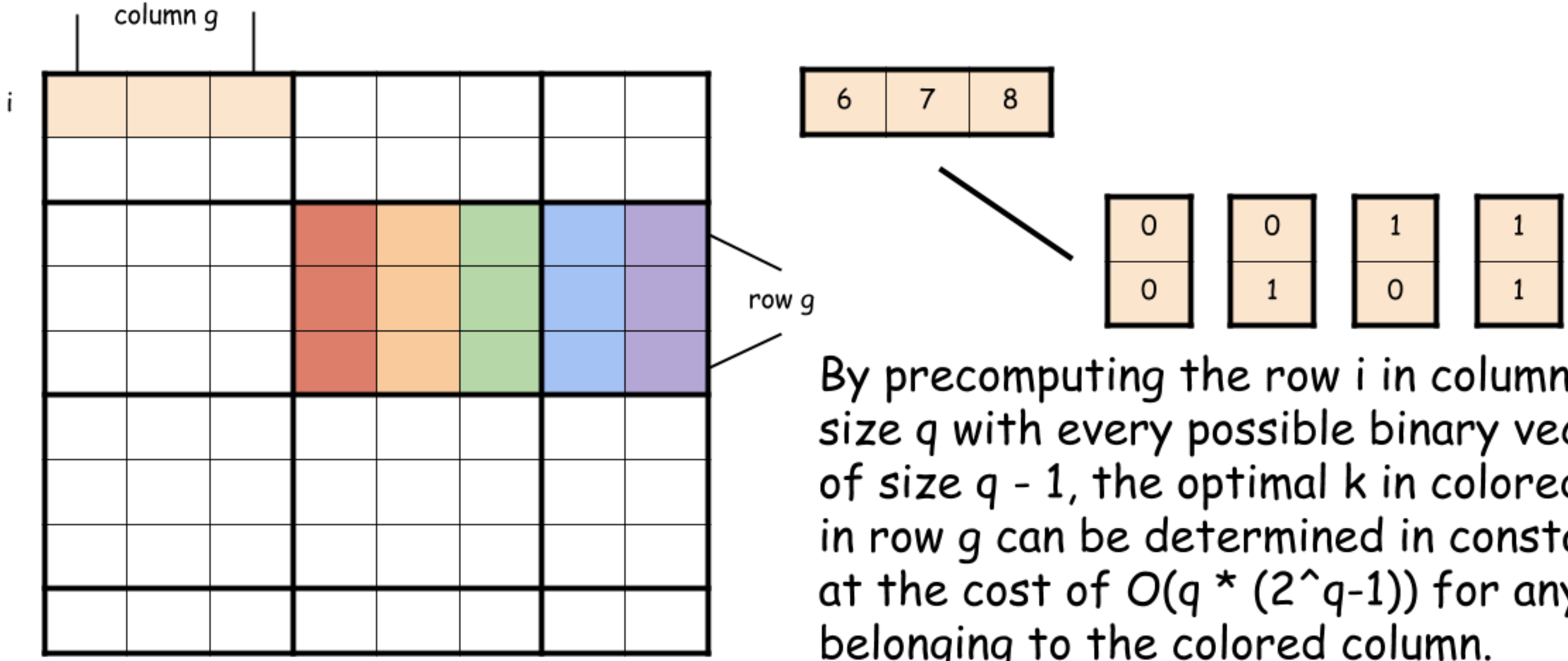
Speeding Up RNA Folding

- Speeding up Rule D will lower the algorithm's complexity
- Splitting the matrix into columns and rows of size q (C group and R group) to form blocks
- R table
- Binary vectors


j
i
j+1
i


j
j+1
i
j
j+1
i


j
j+1
i
6
7
8
c
b
a
3
3
3
= 9
= 10
= 11
4
4
4
= 10
= 11
= 12
Optimal K = a
Can be represented by
a binary vector (vg):
0
0



By precomputing the row i in column g of size q with every possible binary vector (vg) of size q - 1, the optimal k in colored column in row g can be determined in constant time at the cost of O(q * (2^q-1)) for any i belonging to the colored column.

Using the Four-Russians Technique

```
for j = 1 to n:
    for i = 0 to j:
        Do Rule A and B
    for i = j-1 to 0:
        for g = (j-1)/q to i/q:
            if  i >= g*q: // Is Row i a part of the R Group g
                Do Rule C and D
            else:
                Find optimal k for using the R Table and Binary Vector g
                Apply k to Rule D
        if (i mod q) == 0: // Is Row i the last row in a R Group
            Find the Binary Vector g and save it
    if (j+2 mod q) == 0: // Is Column j the last column in a C Group
        for all binary vectors possible of size q-1: // There are 2^(q-1) vectors
            for i = 0 to j-2
                Precompute the R Table for the current C Group by finding
                the optimal k given the binary vector
```

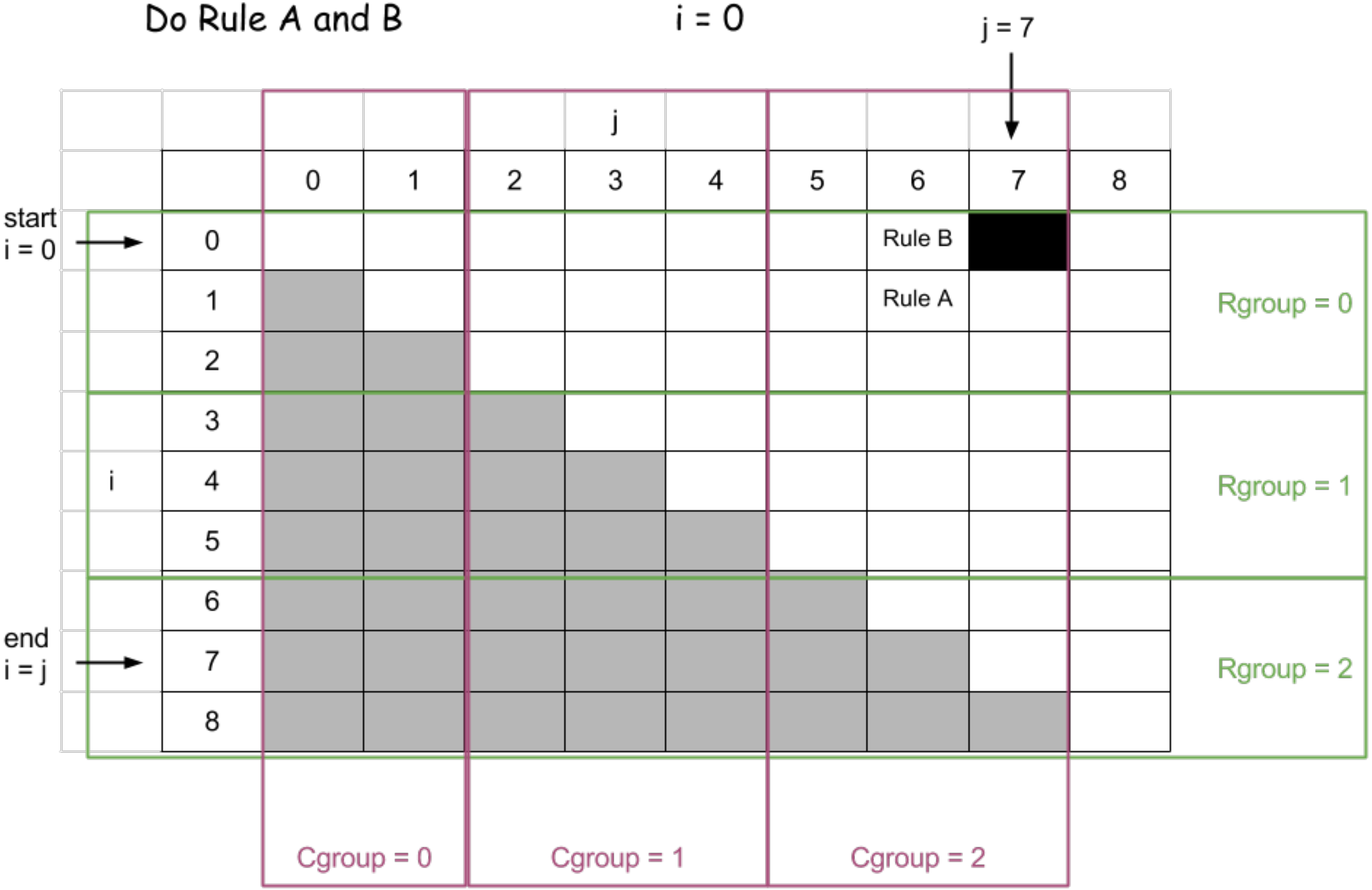
Four-Russians Algorithm

Example

- 9 X 9 table
- q = 3
- column j
- row i

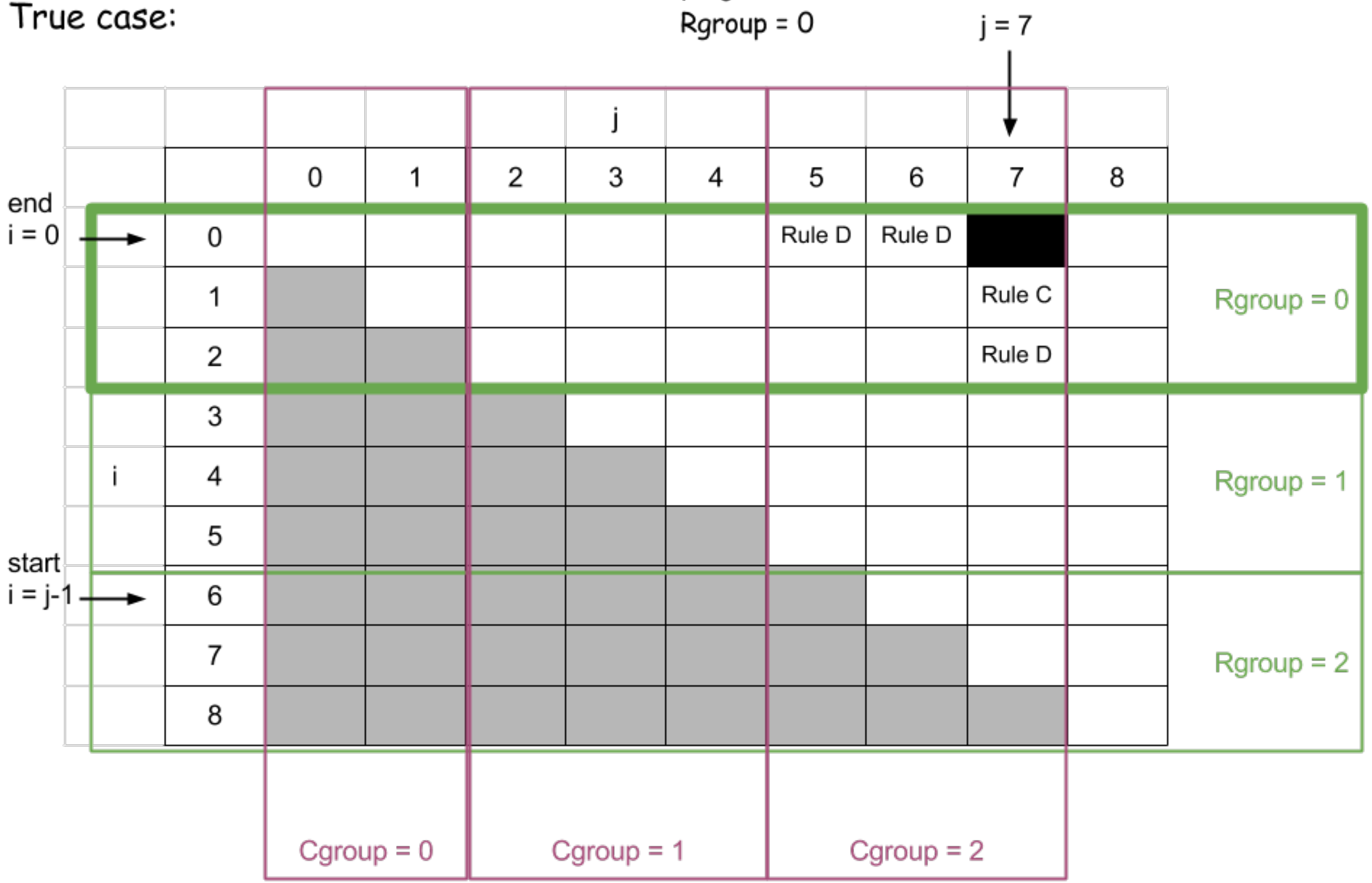
for j = 1 to n:
    for i = 0 to j:
        Do Rule A and B

Example: when
j = 7
i = 0



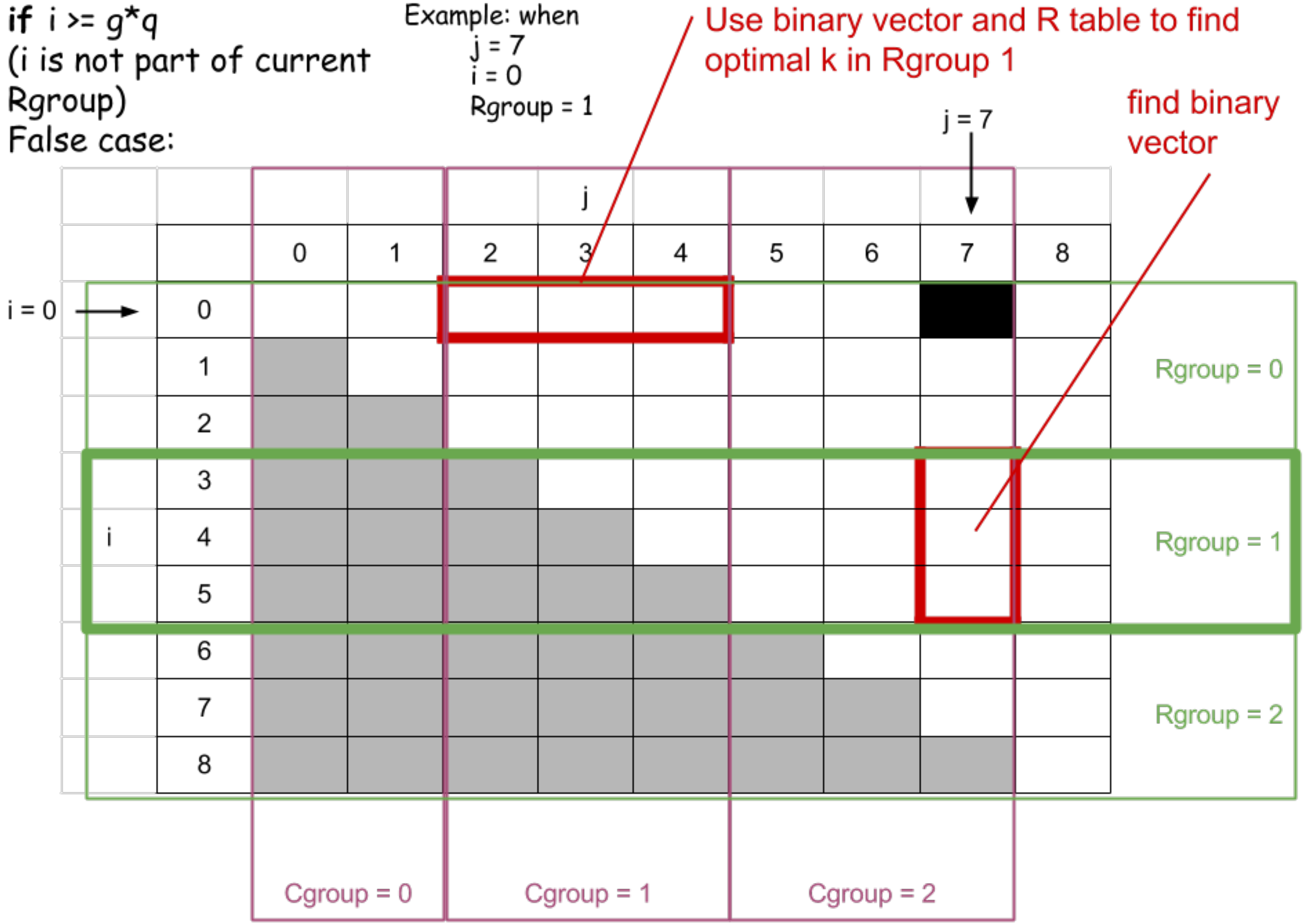
if i >= g*q
(i is part of current Rgroup)
True case:

Example: when
j = 7
i = 0
Rgroup = 0



if i >= g*q
(i is not part of current Rgroup)
False case:

Example: when
j = 7
i = 0
Rgroup = 1

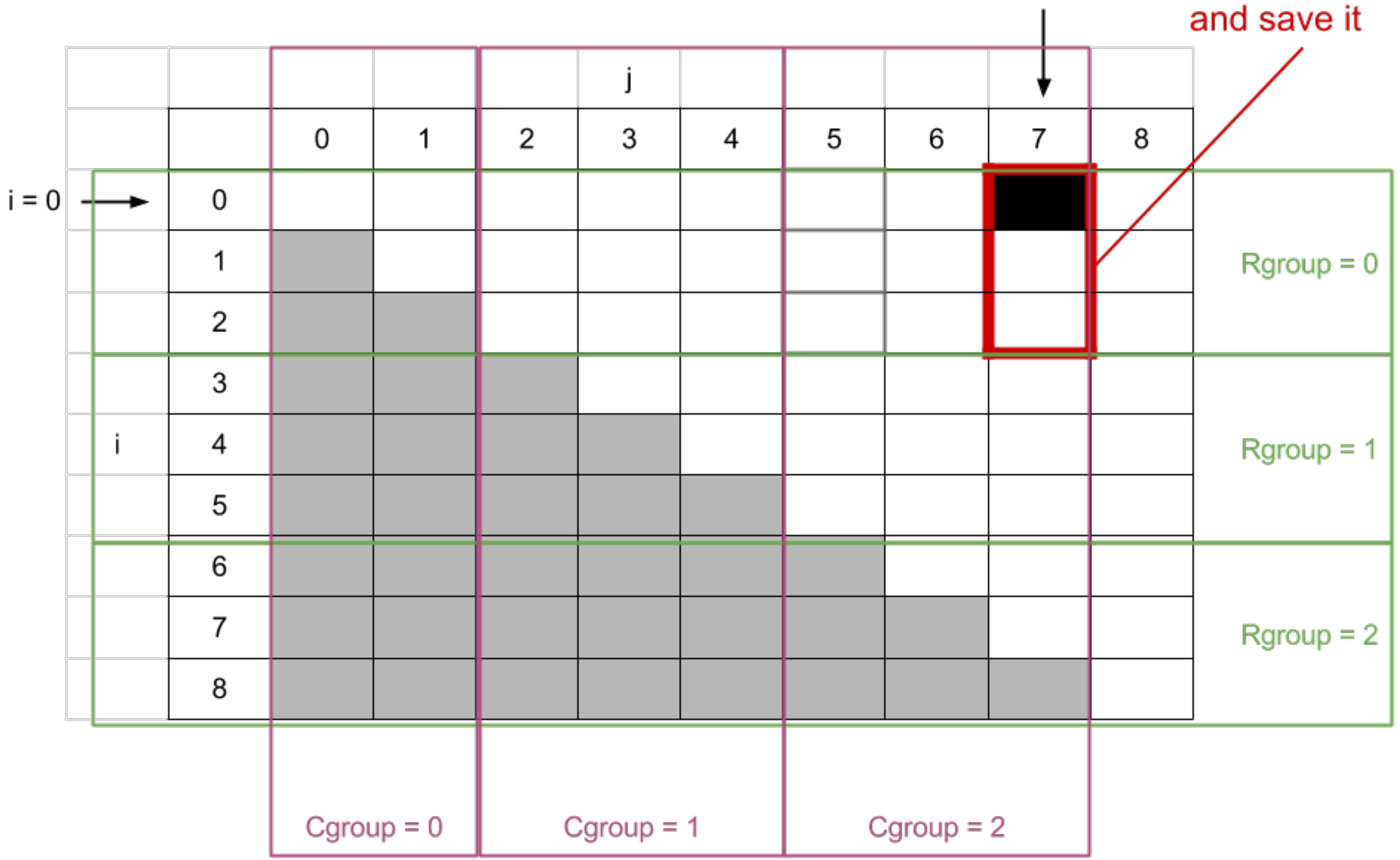


if (i mod q) == 0
(if i is the last (top) row in a Rgroup)

Example: when
j = 7
i = 0

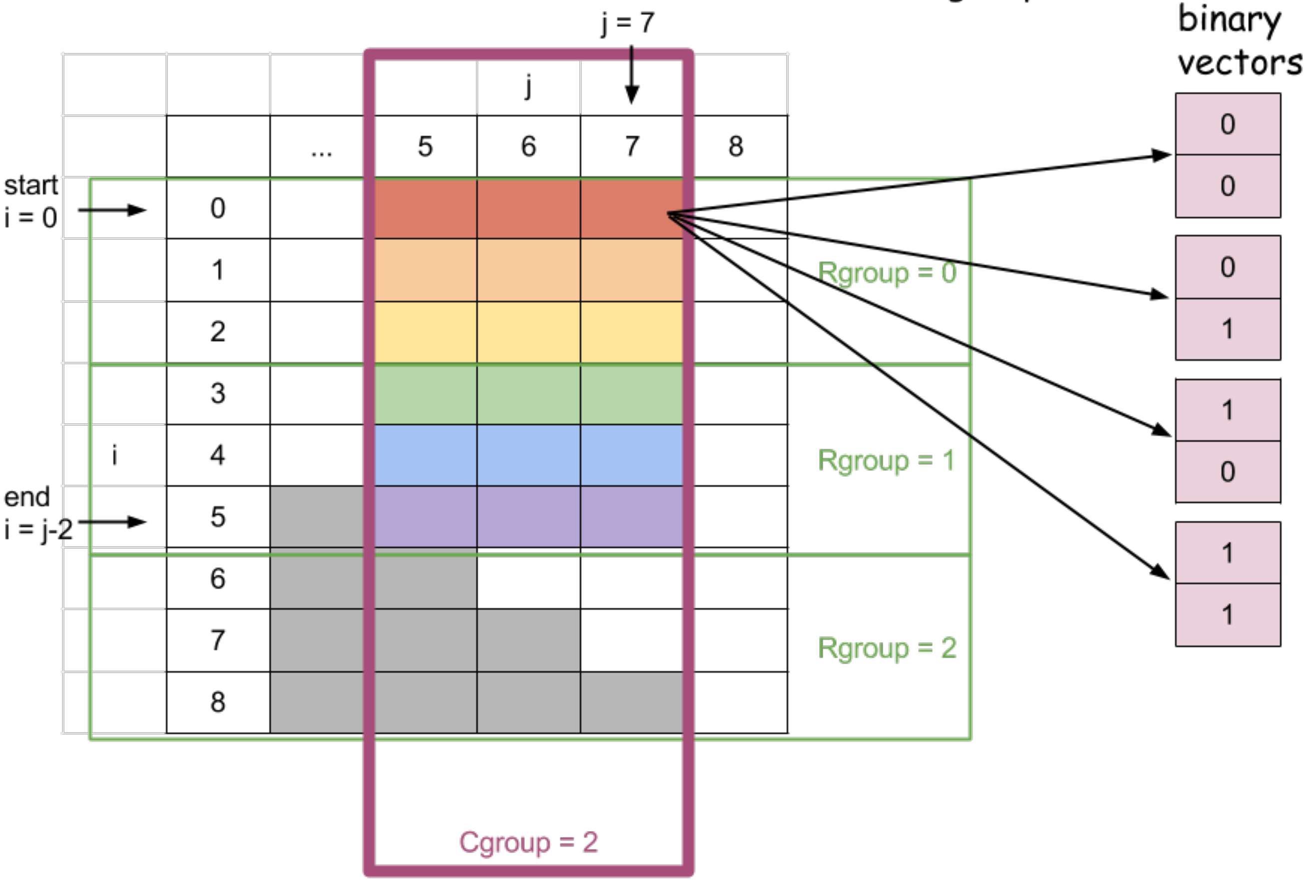
Find the binary vector and save it

j = 7

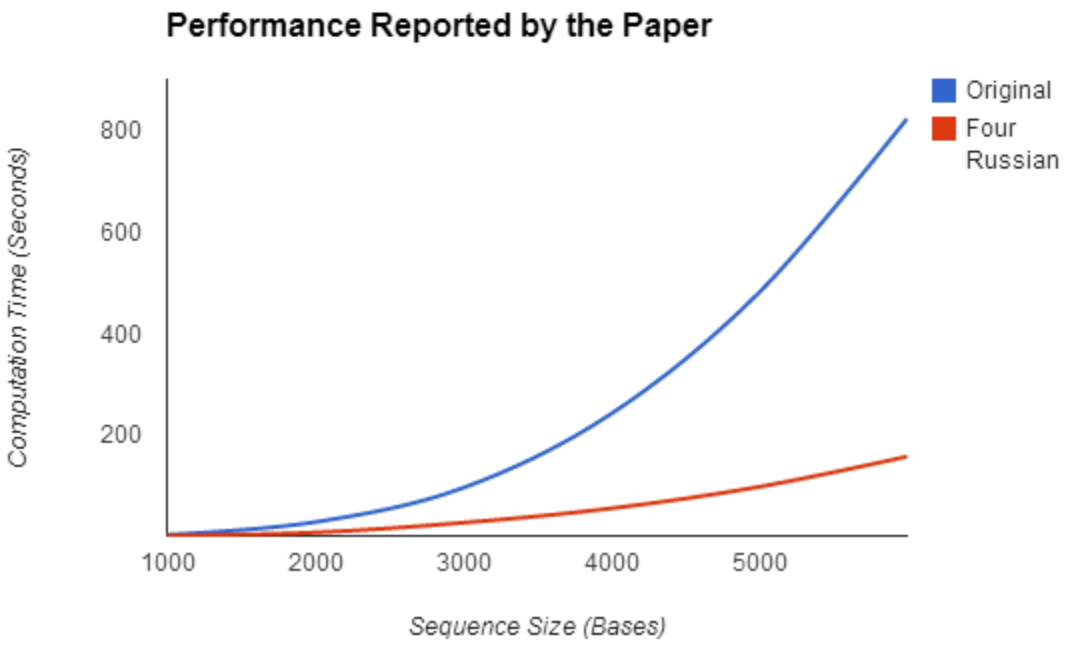


if (j + 2 mod q) == 0
(if j is the last (right most) column in a Cgroup)

Example: when
j = 7
Cgroup = 2

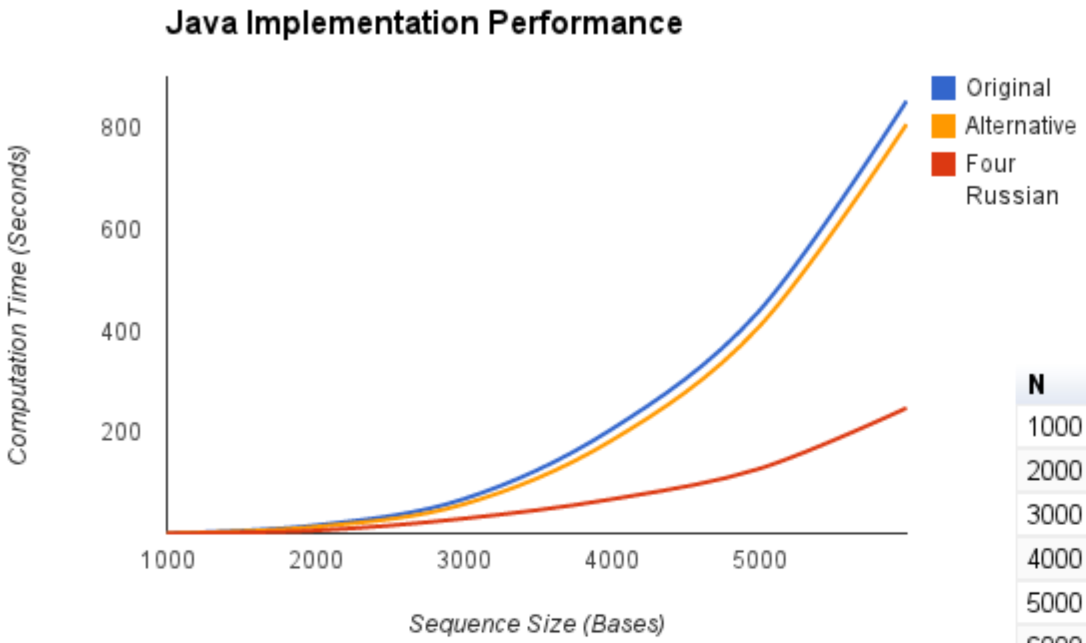


Performance



N	Original	Four Russian
1000	3	1
2000	28	8
3000	95	27
4000	241	55
5000	480	98
6000	823	157

Performance



N	Original	Alternative	Four Russian
1000	1	1	1
2000	17	14	7
3000	68	58	30
4000	205	183	68
5000	438	407	128
6000	853	807	248