Dancing Links and Exact Cover

Knuth, Donald E. "Dancing links." *arXiv preprint cs/0011047* (2000).

Speaker: Wei Li

Advisor: I-Chen Wu

Outline

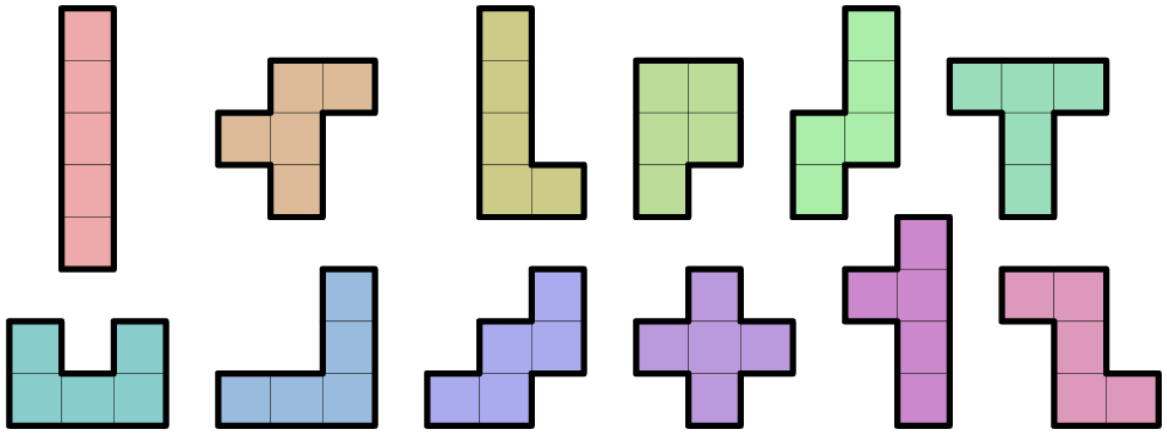
- Introductions
  - Exact Cover Problem
- Dancing Links and X algorithm
- Application and Comparison
  - Polyomino
  - Sudoku
  - N queens puzzle
- Conclusion

Outline

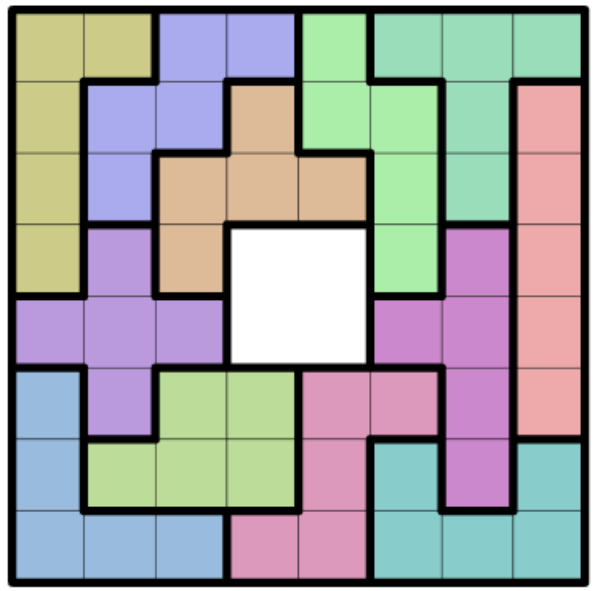
- Introductions
  - Exact Cover Problem
- Dancing Links and X algorithm
- Application and Comparison
  - Polyomino
  - Sudoku
  - N queens puzzle
- Conclusion

Exact Cover Problem

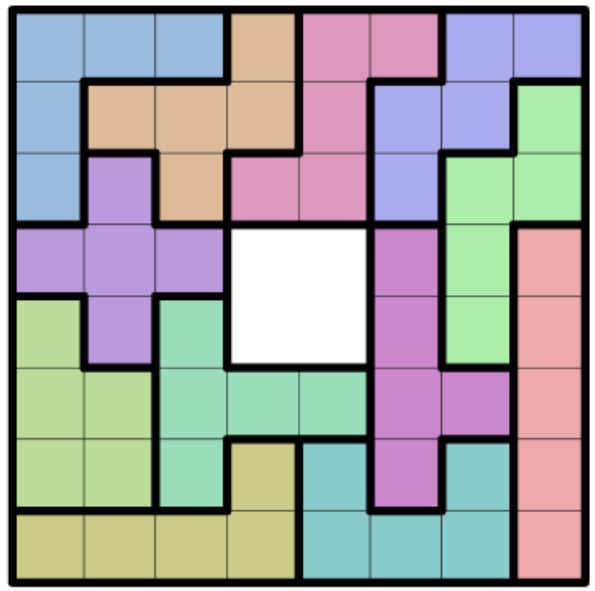
- Polyomino



12 *pentominoes to be placed* (60 *squares*)

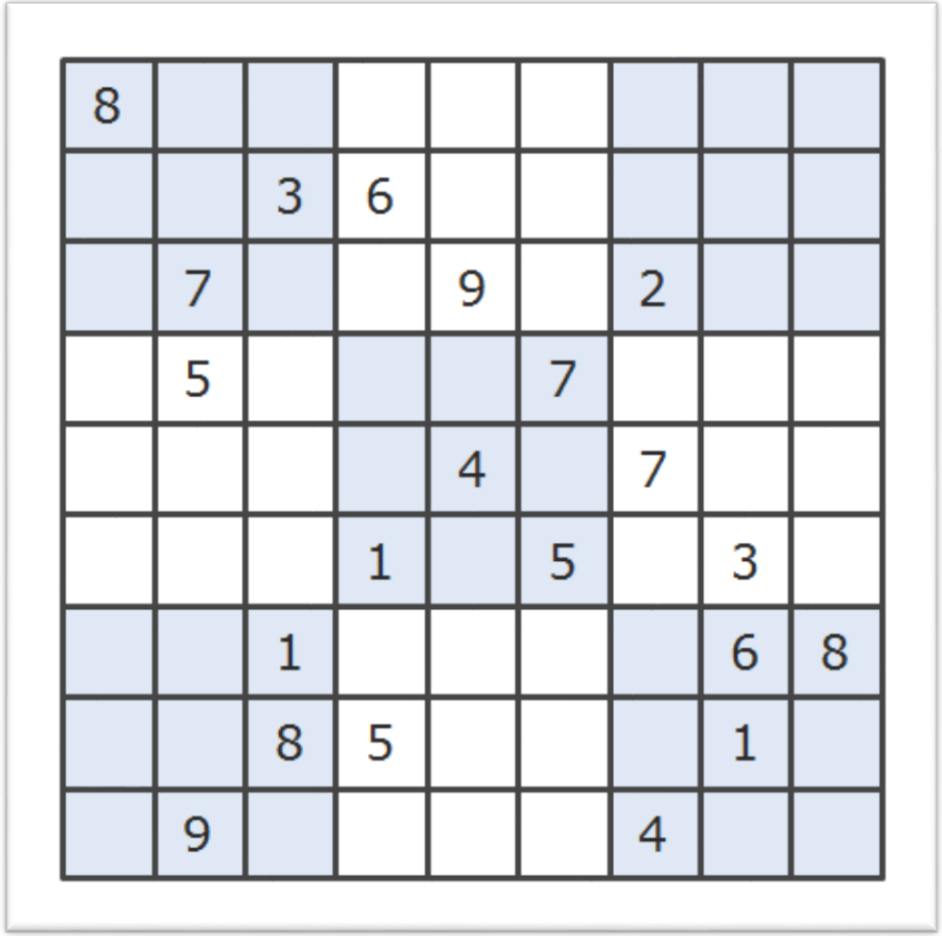


$8 * 8 - 4$
$= 60\ squares$

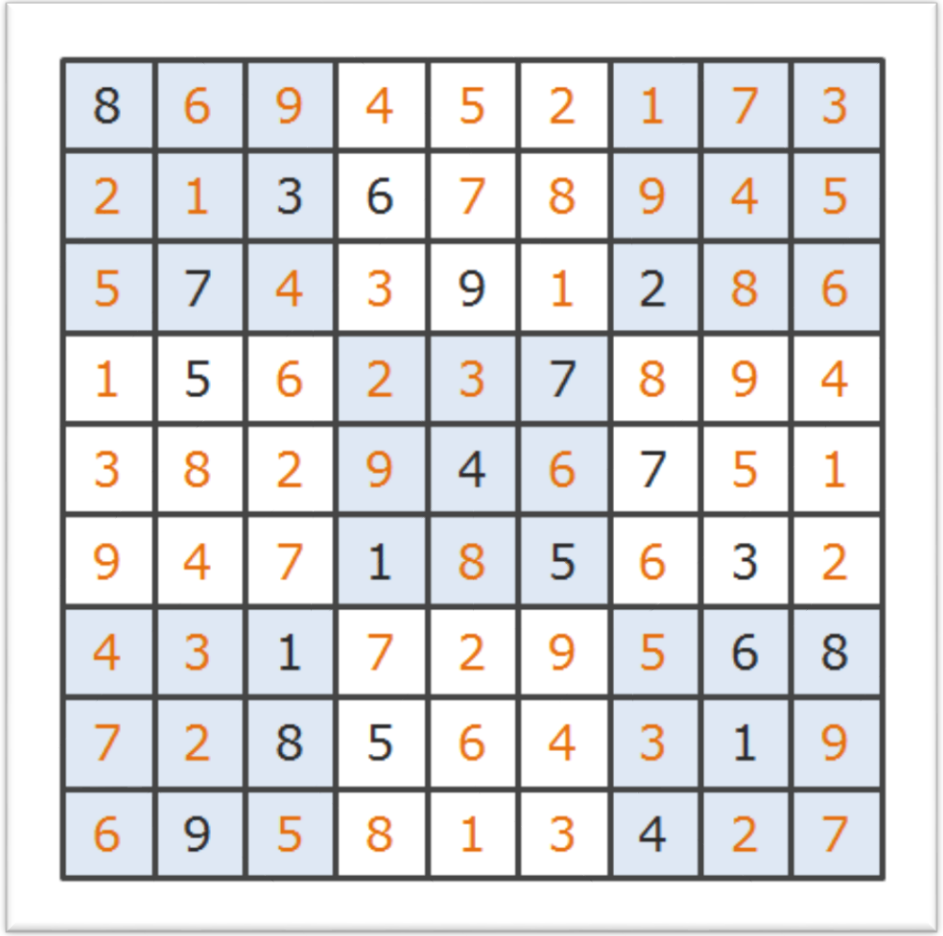


Exact Cover Problem (cont.)

- Sudoku



8								
		3	6					
	7			9		2		
	5				7			
				4		7		
			1		5		3	
		1					6	8
		8	5				1	
	9					4		



8	6	9	4	5	2	1	7	3
2	1	3	6	7	8	9	4	5
5	7	4	3	9	1	2	8	6
1	5	6	2	3	7	8	9	4
3	8	2	9	4	6	7	5	1
9	4	7	1	8	5	6	3	2
4	3	1	7	2	9	5	6	8
7	2	8	5	6	4	3	1	9
6	9	5	8	1	3	4	2	7

Exact Cover Problem (cont.)

- Given a matrix of 0s and 1s, does it have a set of rows containing exactly one 1 in each column? *a set (rows 1, 4, and 5).*

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 \\ & & & & & & \\ & & & & & & \\ \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & \mathbf{1} \\ & & & & & & \end{pmatrix}$$

- How to solve it?
  - Brute force search?
  - No, just dance!

Outline

- Introductions
  - Exact Cover Problem
- Dancing Links and X algorithm
- Application and Comparison
  - Polyomino
  - Sudoku
  - N queens puzzle
- Conclusion

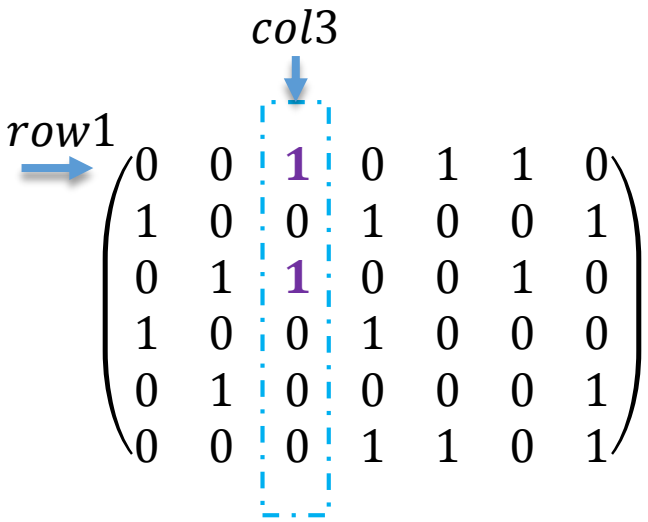
Dancing Links and X algorithm

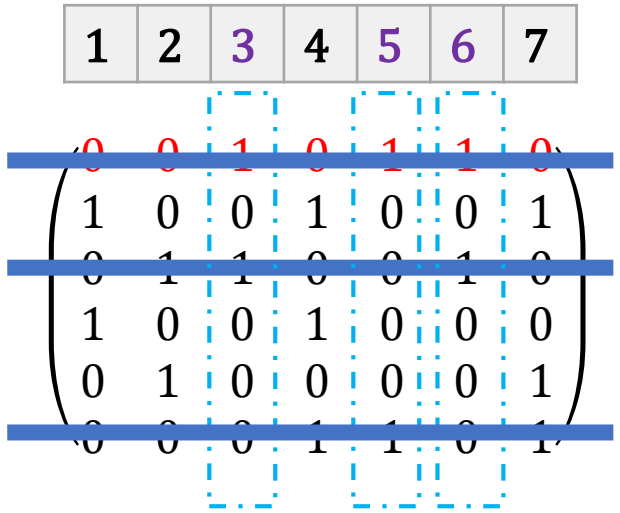
- Popularized by Knuth, “Dancing Links” (2000, arXiv)
  - Algorithm X = “traditional” backtracking (DFS)
  - Algorithm DLX = Dancing Links + Algorithm X
  - 26 pages, applications to packing pentominoes in a square

Knuth’s X algorithm

- If the set of unsatisfied constraints is empty, the problem is solved. Yield the solution and return.
- Otherwise, choose an unsatisfied constraint *c*.
- If there are no choices that satisfy this constraint, the problem cannot be solved from this position. Return.
- Otherwise, for each choice *i* that satisfies *c*:
  - For each constraint *j* satisfied by *i*:
    - Remove *j* from the set of unsatisfied constraints.
    - For each choice *k* that satisfies *j*, remove *k* from the set of choices.
  - Recursively run this algorithm on the reduced set of choices and constraints.
  - For each constraint *j* satisfied by *i*:
    - Restore *j* to the set of unsatisfied constraints.
    - For each choice *k* that satisfies *j*, restore *k* to the set of choices.

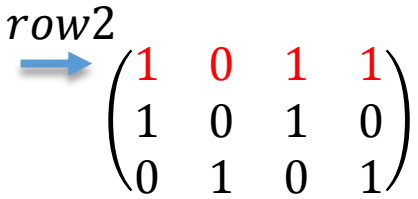
Knuth's X algorithm (cont.)

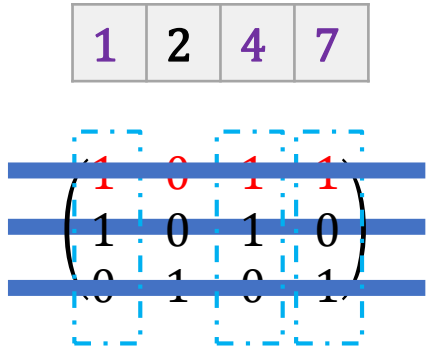




1	2	3	4	5	6	7
		✓		✓	✓	

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{matrix} row2 \\ row4 \\ row5 \end{matrix}$$

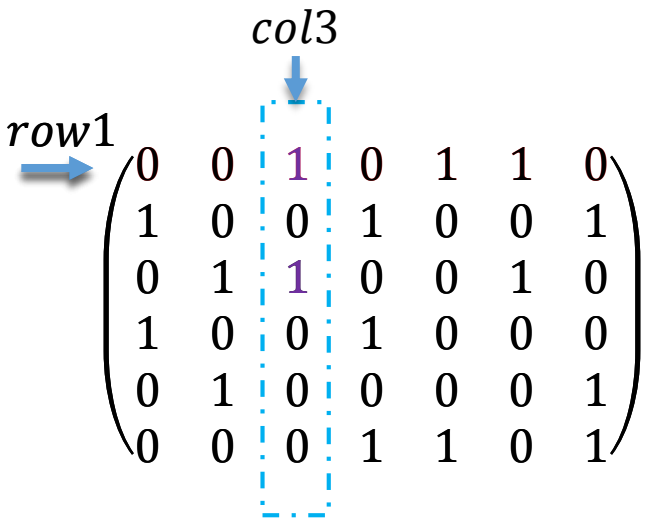


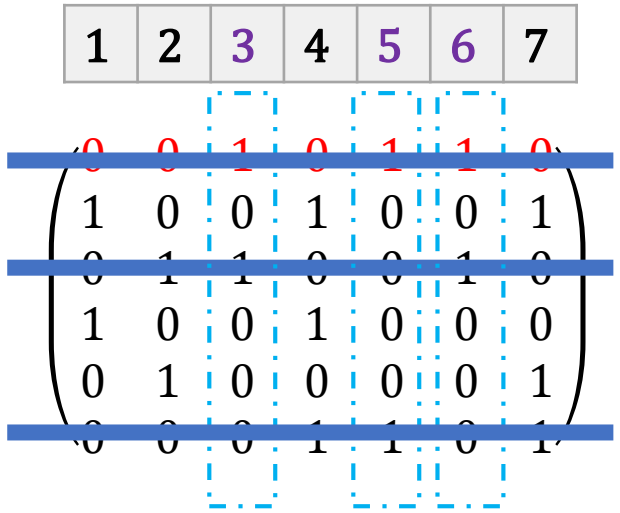


1	2	3	4	5	6	7
✓		✓	✓	✓	✓	✓

$$\begin{pmatrix} & & & \end{pmatrix} \; failed$$

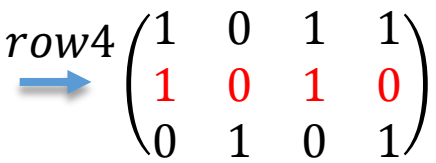
Knuth's X algorithm (cont.)

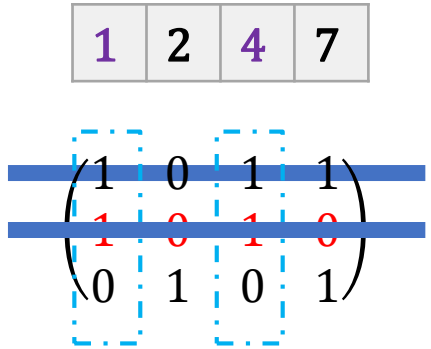




1	2	3	4	5	6	7
		✓		✓	✓	

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{matrix} row2 \\ row4 \\ row5 \end{matrix}$$

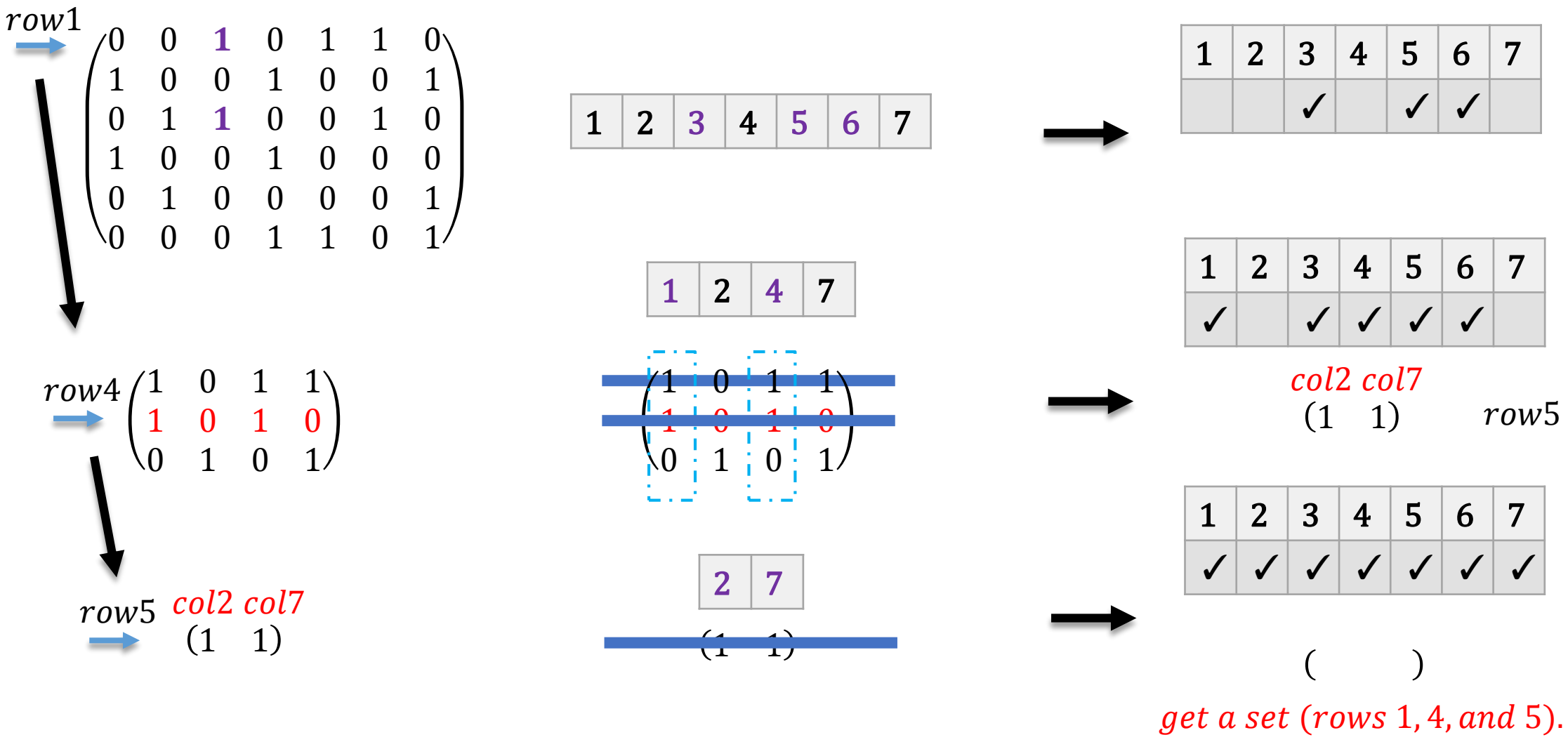




1	2	3	4	5	6	7
✓		✓	✓	✓	✓	

$$(1 \quad 1) \qquad row5$$

Knuth's X algorithm (cont.)



Knuth’s X algorithm (cont.)

- This is a trial-and-error approach
- We need lots of memory to store positions and matrix
- So, what should we do ?

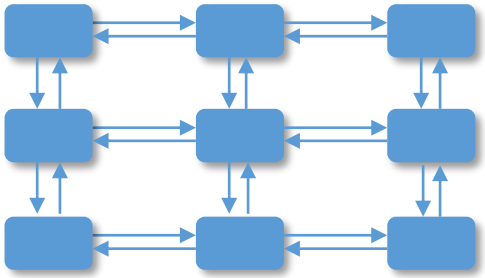
Dancing Links

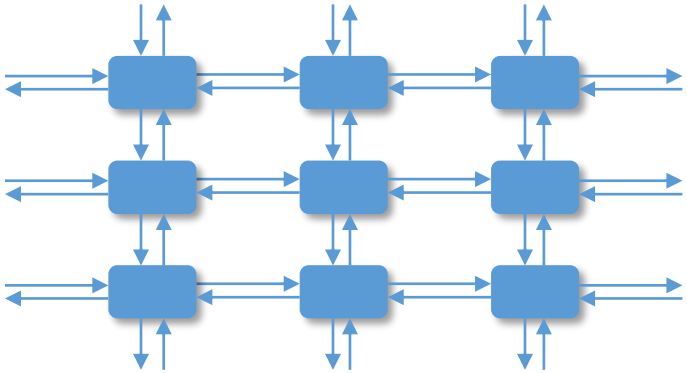
- A simple and beautiful data structure
  - Singly Linked List → Doubly Linked List

  - Orthogonal List → Doubly Orthogonal List

Dancing Links (cont.)

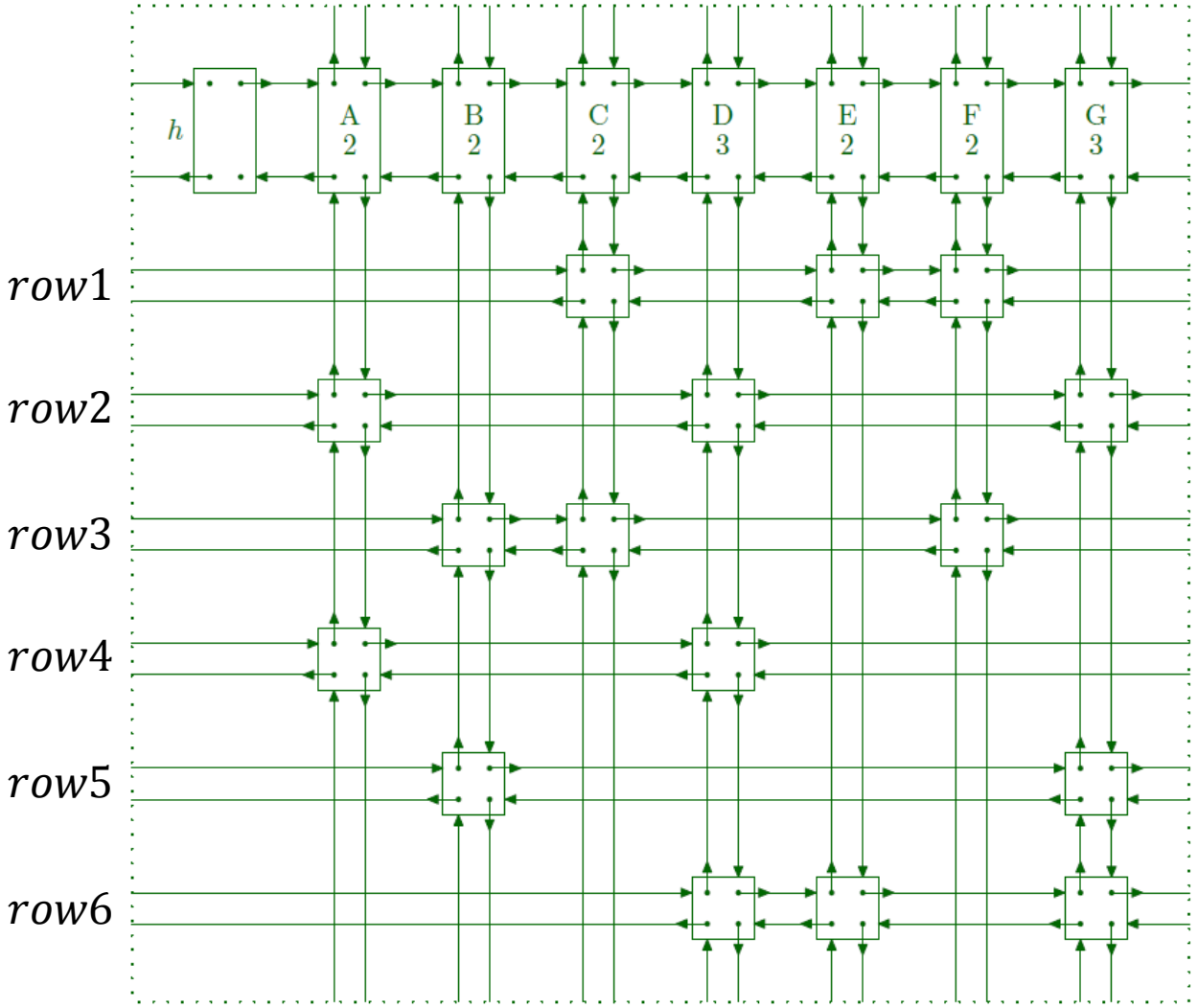
- A simple and beautiful data structure
  - Linked List → Doubly Linked List
  - Orthogonal List → Doubly Orthogonal List
  - Doubly Orthogonal List + Circular List → **Dancing Links**





Dancing Links (cont.)

$$
\begin{array}{c}
row1 \\ row2 \\ row3 \\ row4 \\ row5 \\ row6
\end{array}
\begin{pmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1
\end{pmatrix}
$$

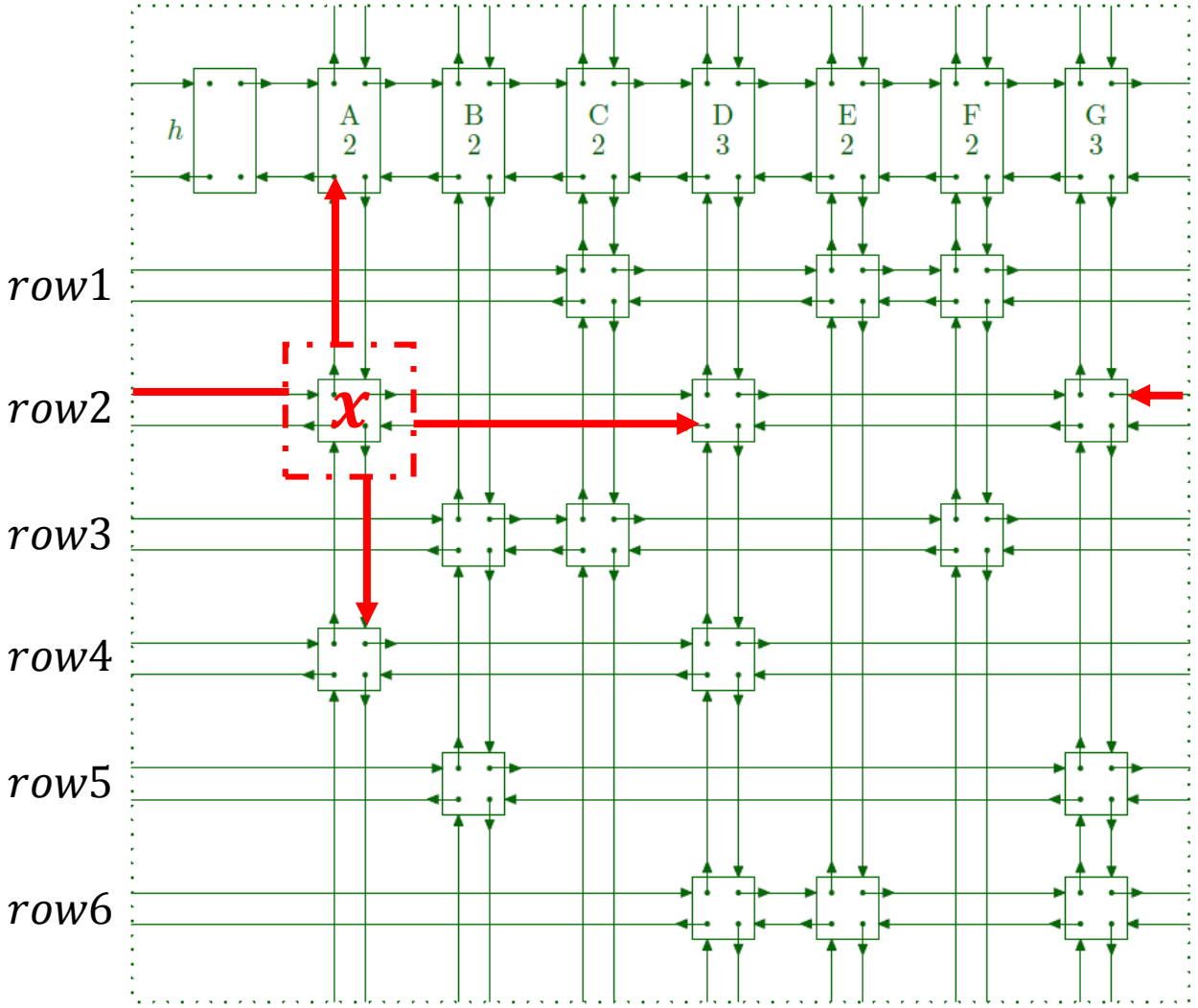


Dancing Links (cont.)

- Data object has five fields

$L[x]$、$R[x]$、$U[x]$、$D[x]$、$C[x]$

$$
\begin{array}{l}
row1 \\ row2 \\ row3 \\ row4 \\ row5 \\ row6
\end{array}
\begin{pmatrix}
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1
\end{pmatrix}
$$



Dancing Links (cont.)

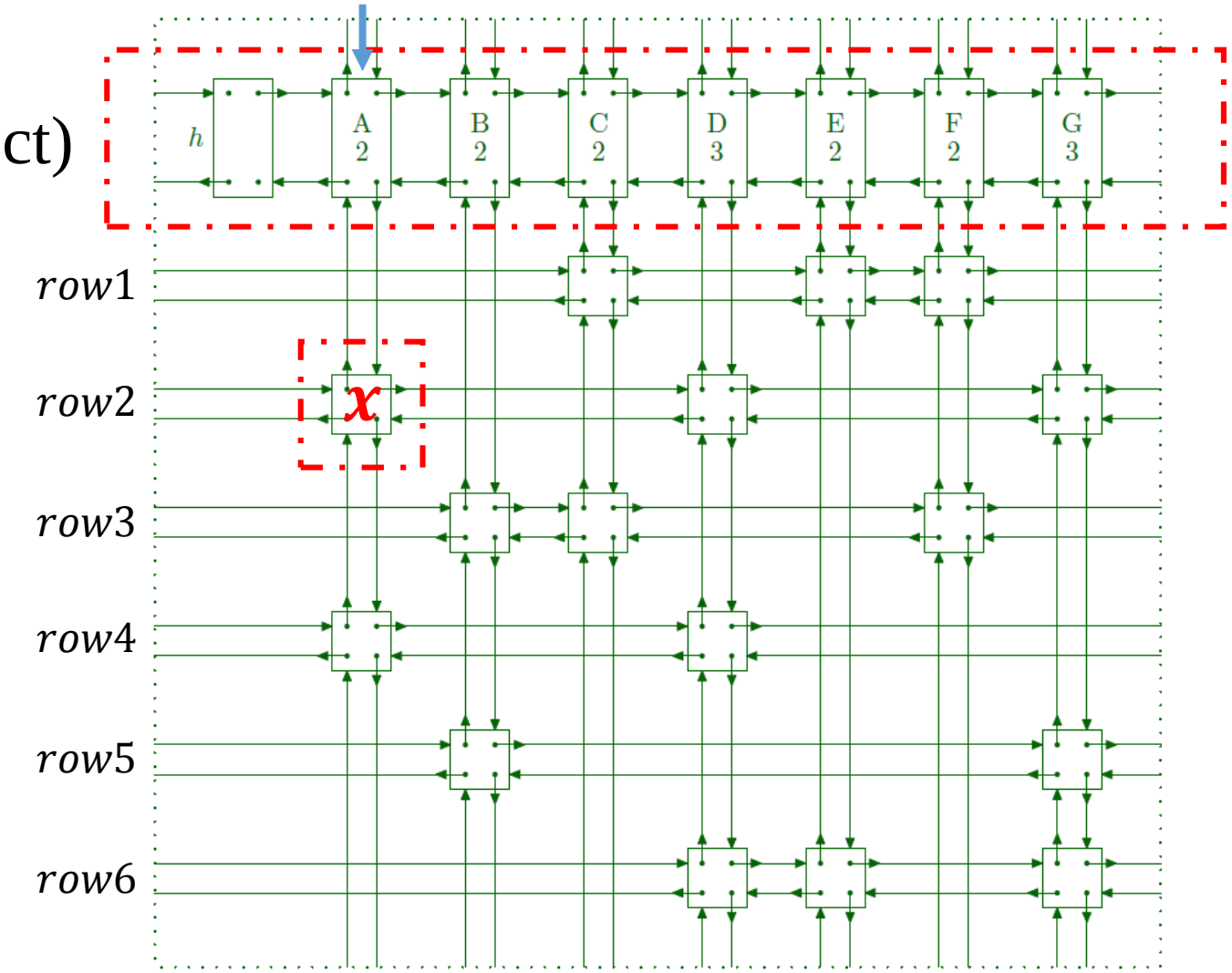
- List header(special data object)

$L[y]$、$R[y]$、$U[y]$、$D[y]$、$C[y]$

$\boldsymbol{S[y] - number\ of\ 1}$

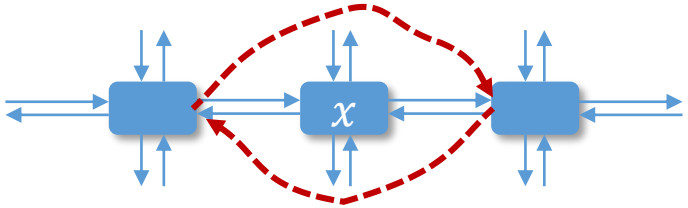
$N[y] - column's\ name$

$S[C[x]\ ] = 2, N[x] = "A"$

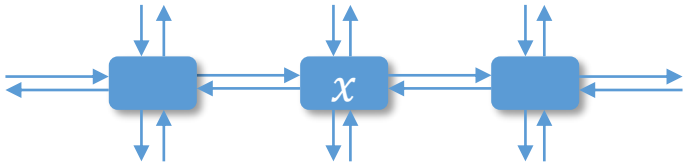


Dancing Links (cont.)

- Operation
  - Remove
    - $L[R[x]] = L[x], R[L[x]] = R[x]$
    - $D[U[x]] = D[x], U[D[x]] = U[x]$

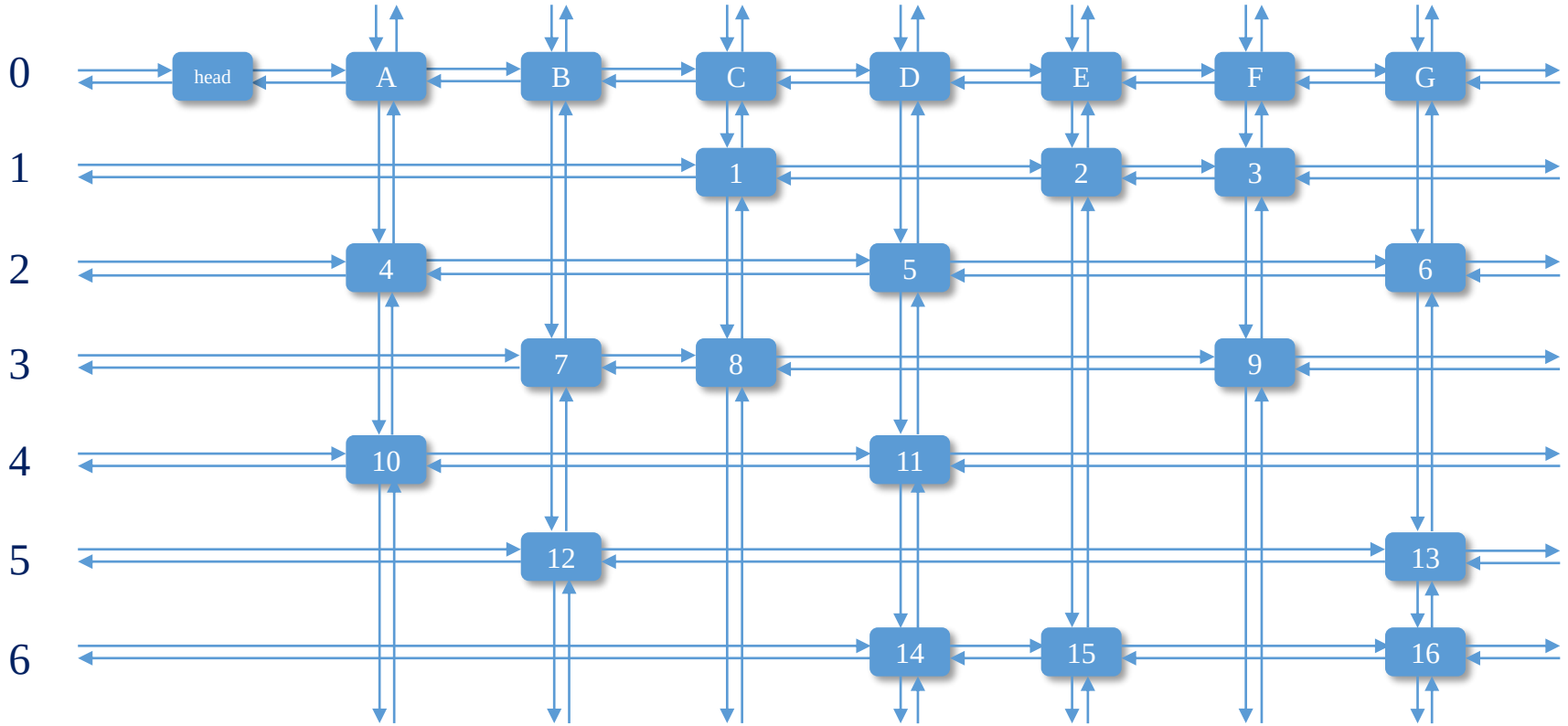


  - Resume
    - $\boldsymbol{L[R[x]] = x,\ R[L[x]] = x}$
    - $\boldsymbol{D[U[x]] = x,\ U[D[x]] = x}$



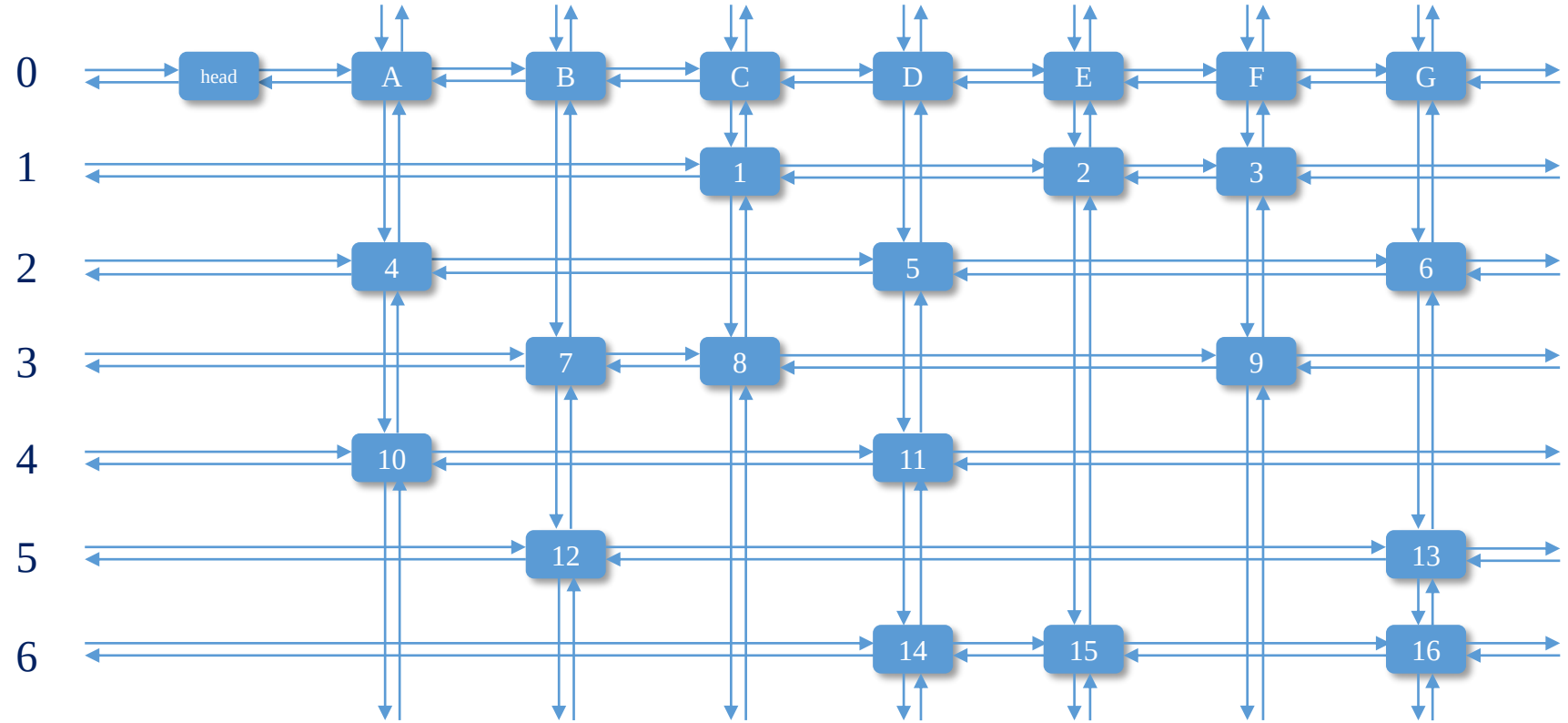
DLX Algorithm

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$



DLX Algorithm (cont.)

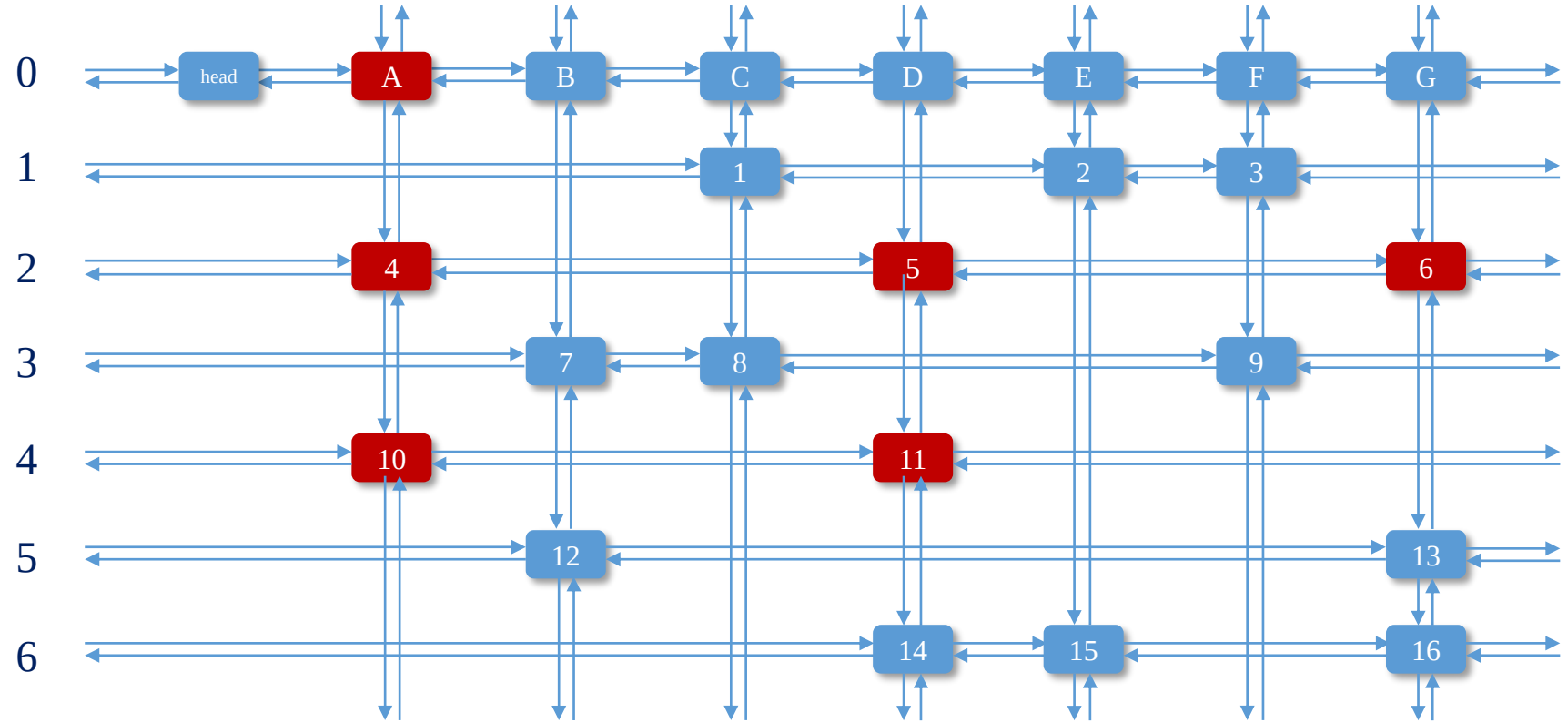
1. $R[Head] = A \neq Head$



DLX Algorithm (cont.)

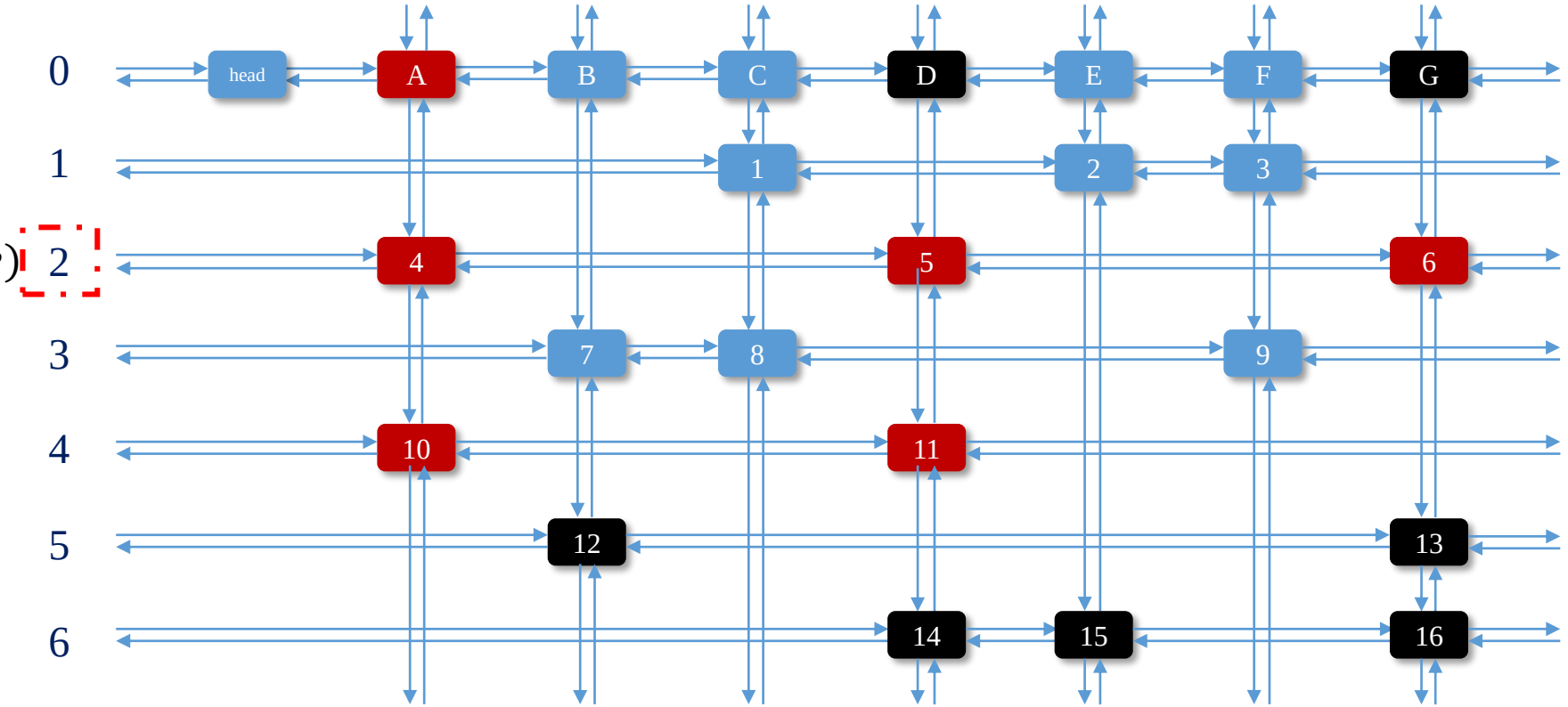
1. $R[Head] = A \neq Head$

2. $mark\ A, OP(remove)$



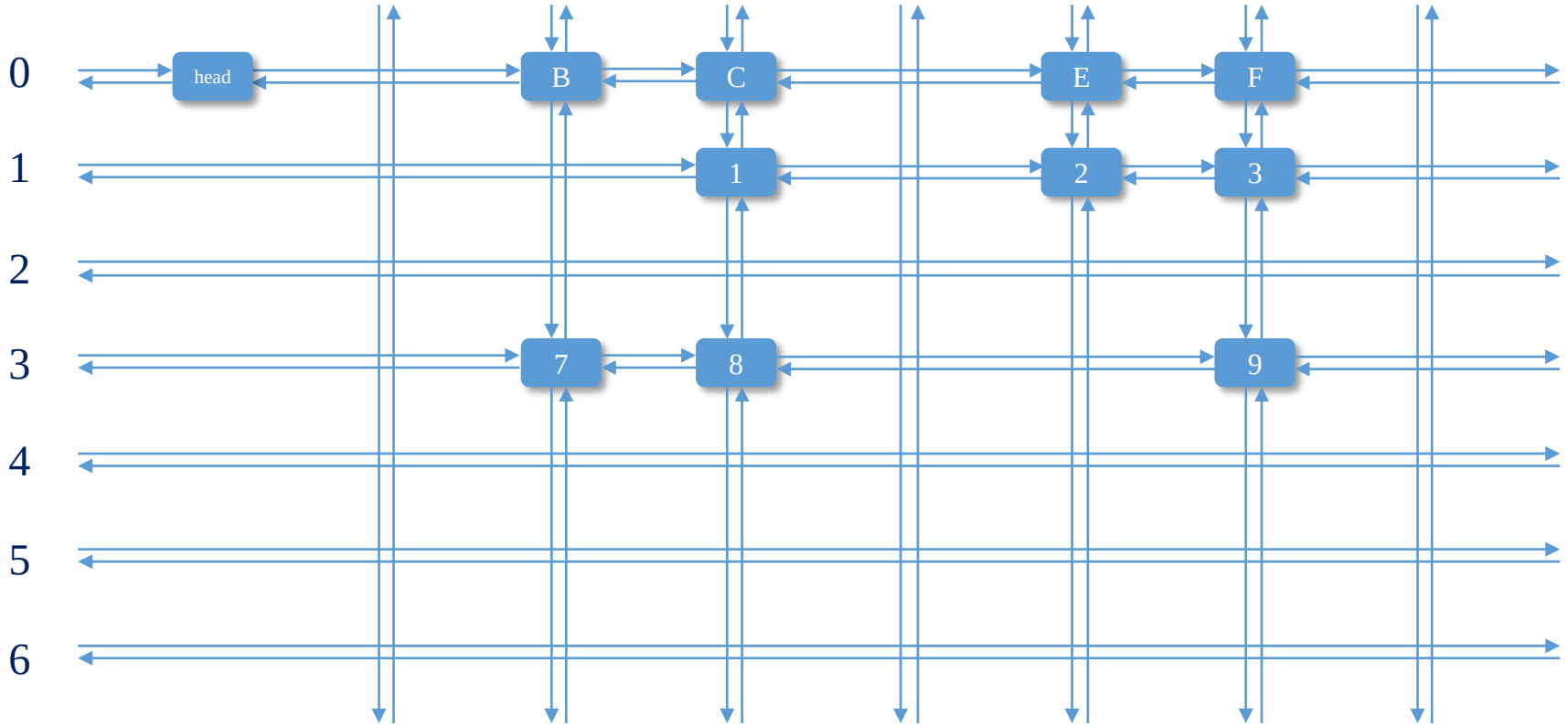
DLX Algorithm (cont.)

1. $R[Head] = A \neq Head$
2. $mark\ A, OP(remove)$
3. $choose\ row\ 2, OP(remove)$



DLX Algorithm (cont.)

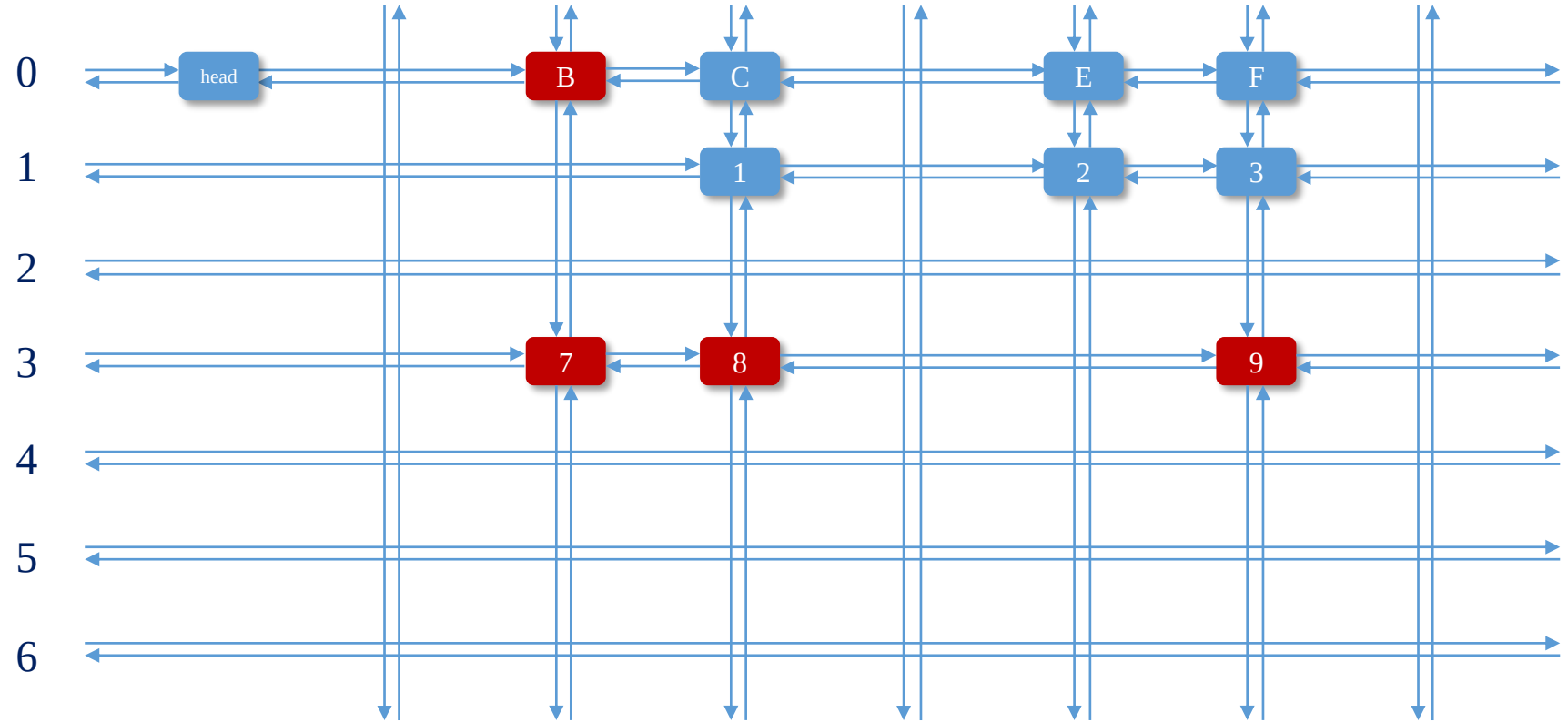
4. $R[Head] = B \neq Head$



DLX Algorithm (cont.)

4. $R[Head] = B \neq Head$

5. $mark\ B, OP(remove)$

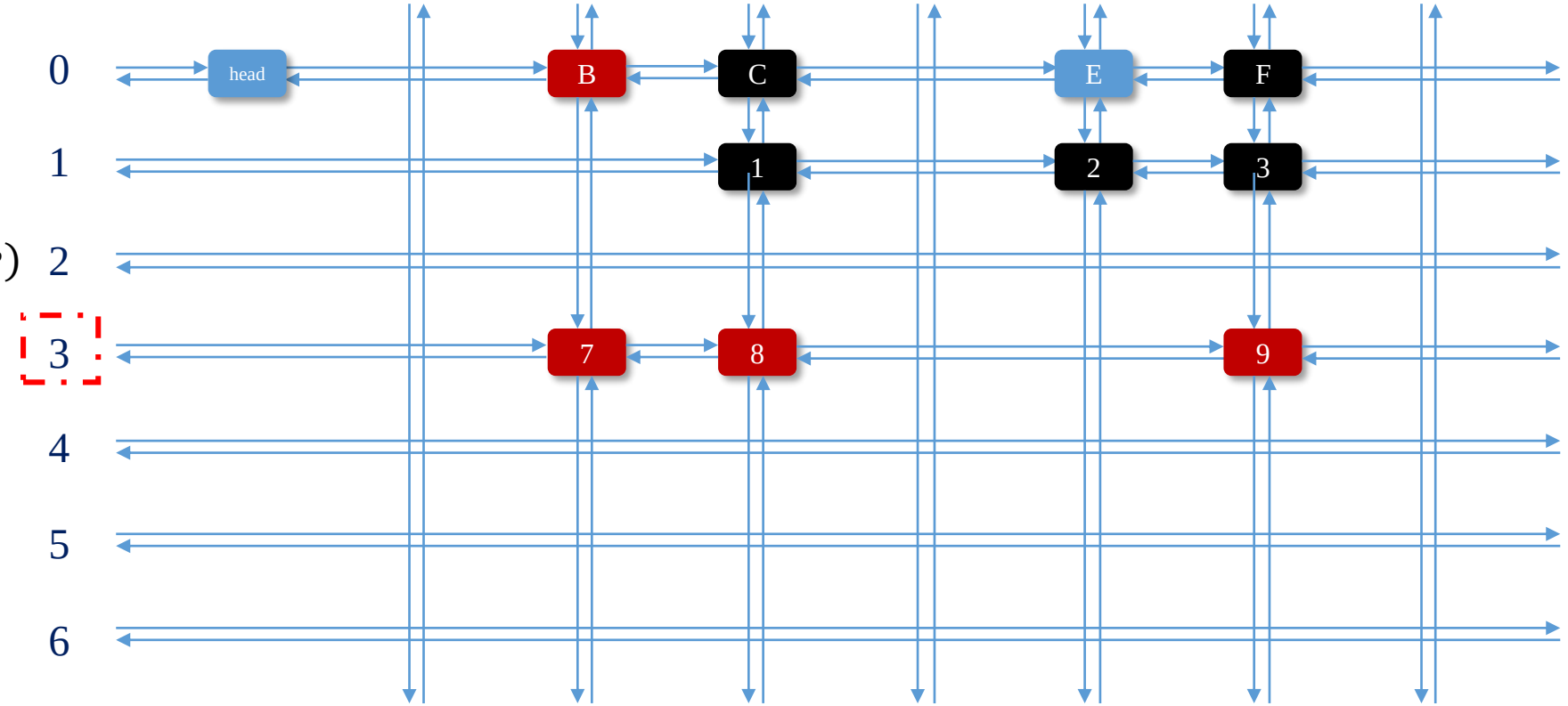


DLX Algorithm (cont.)

4. $R[Head] = B \neq Head$

5. $mark\ B, OP(remove)$

6. $choose\ row\ 3, OP(remove)$

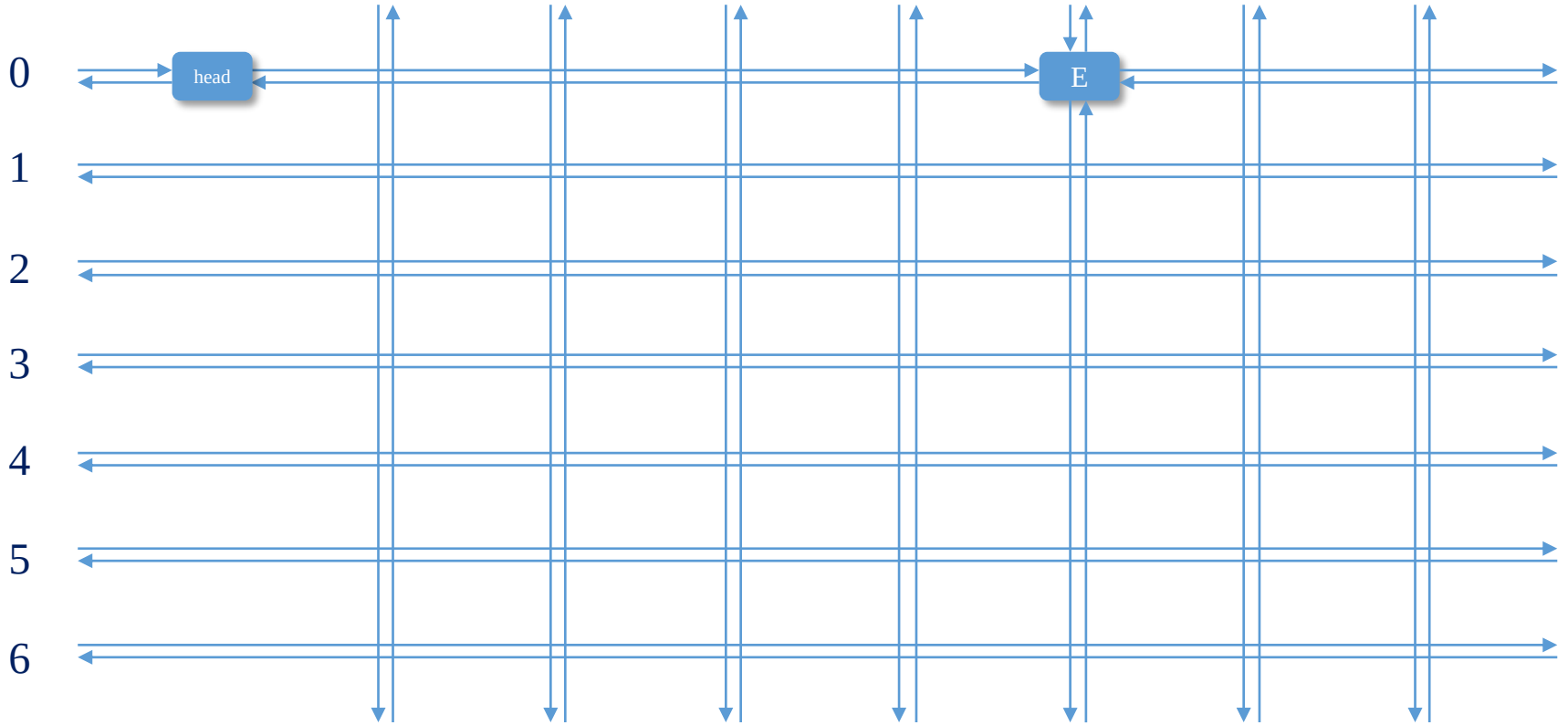


DLX Algorithm (cont.)

$7. R[Head] = E \neq Head$

no object cover E*,* $failed$

$OP(resume)$



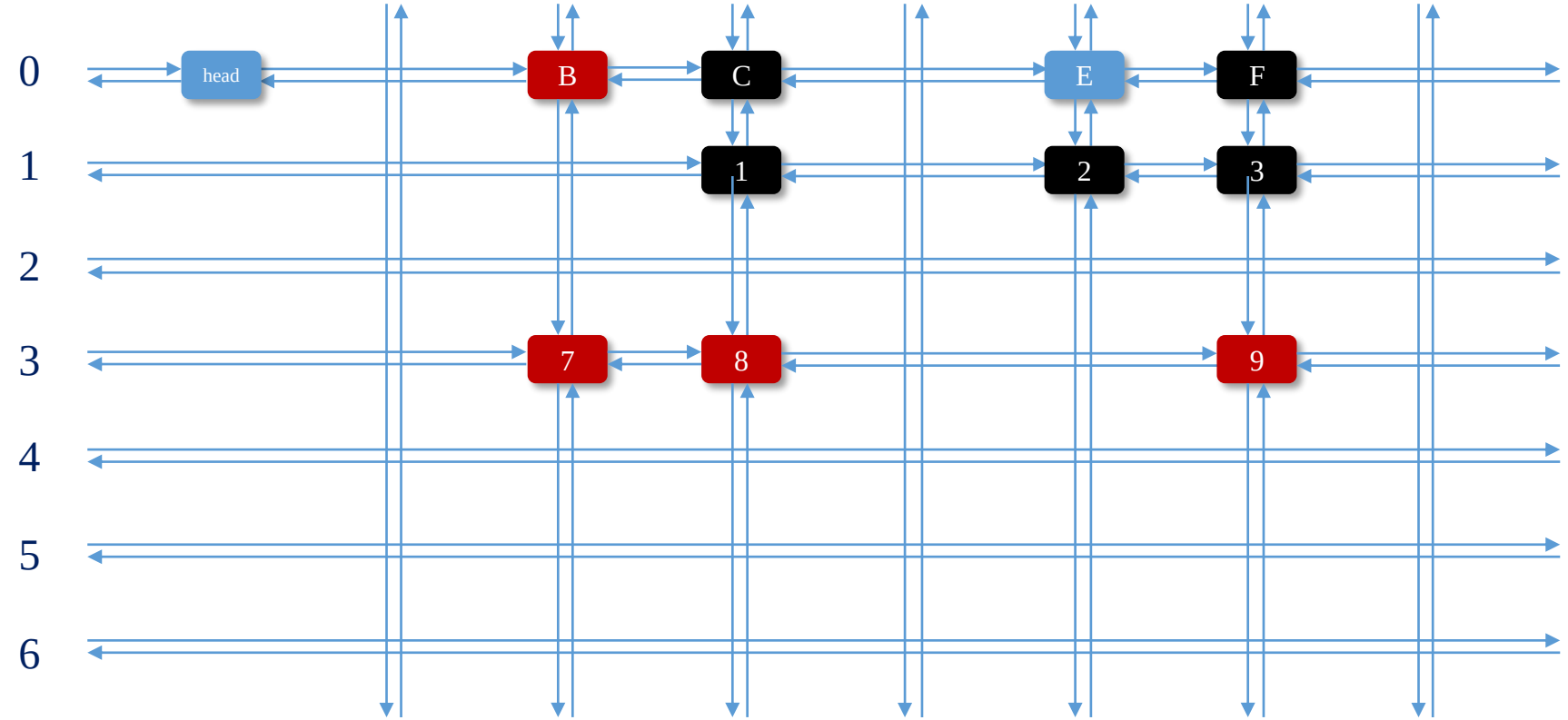
DLX Algorithm (cont.)

4. $R[Head] = B \neq Head$

5. $mark\ B, OP(remove)$

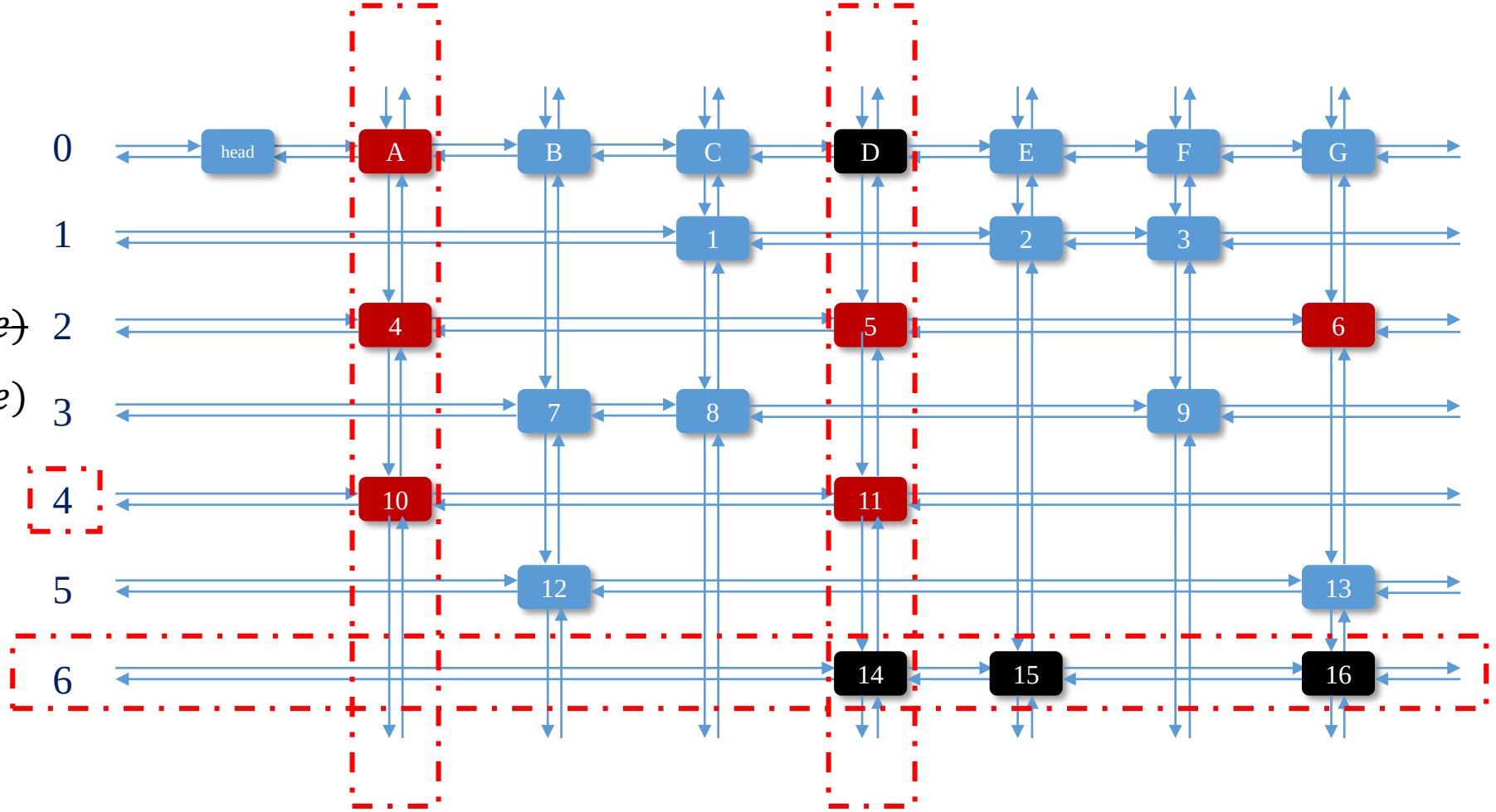
~~6. $choose\ row\ 3, OP(remove)$~~

$OP(resume)$



DLX Algorithm (cont.)

1. $R[Head] = A \neq Head$
2. $mark\ A, OP(remove)$
3. ~~$choose\ row\ 2, OP(remove)$~~
3. $choose\ row\ 4, OP(remove)$



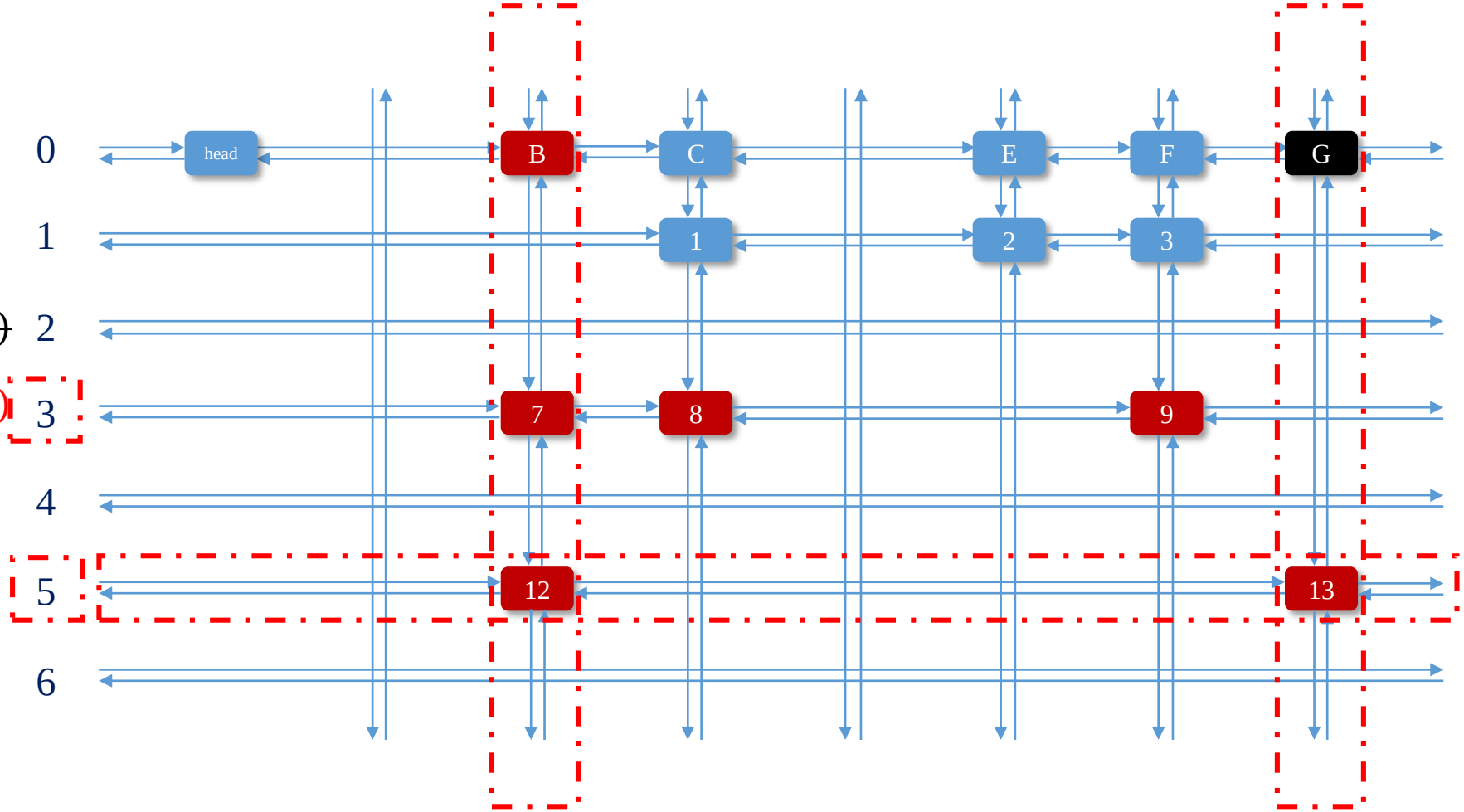
DLX Algorithm (cont.)

4. $R[Head] = B \neq Head$

5. $mark\ B, OP(remove)$

~~6. $choose\ row\ 3, OP(remove)$~~

6. $choose\ row\ 5, OP(remove)$



DLX Algorithm (cont.)

4. $R[Head] = B \neq Head$

5. $mark\ B, OP(remove)$

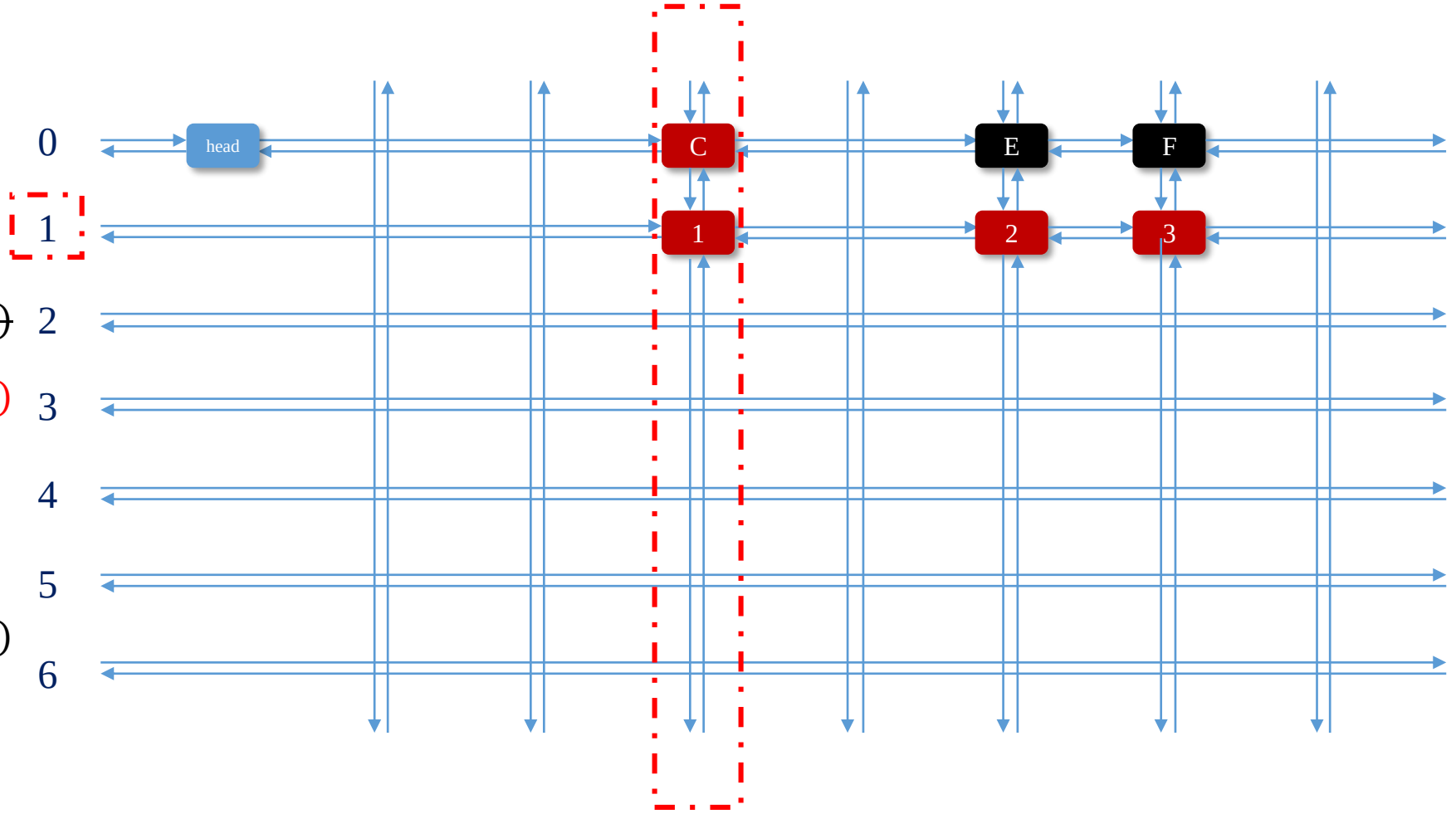
~~6. $choose\ row\ 3, OP(remove)$~~

6. $choose\ row\ 5, OP(remove)$

7. $R[Head] = C \neq Head$

8. $mark\ C, OP(remove)$

9. $choose\ row\ 1, OP(remove)$

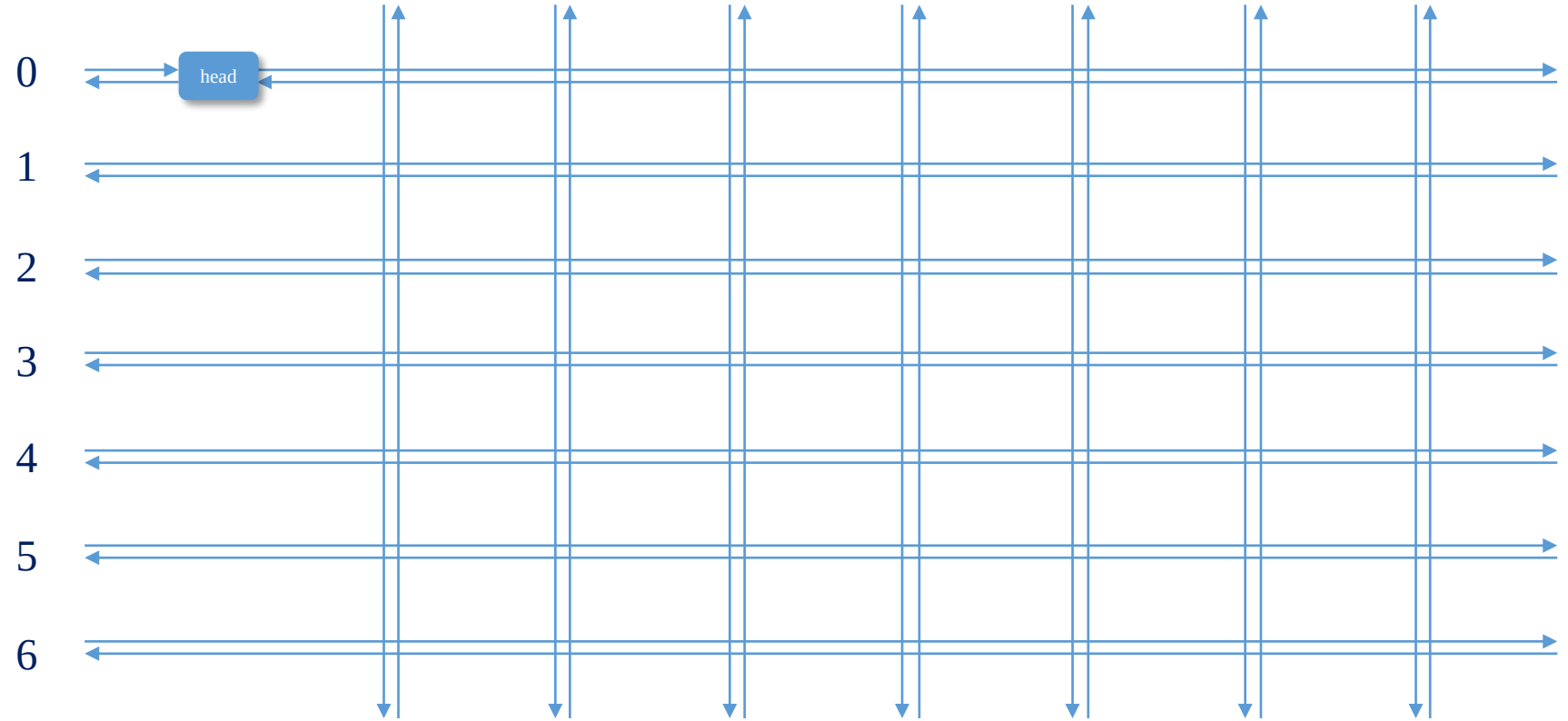


DLX Algorithm (cont.)

$R[Head] = Head$

All columns are covered

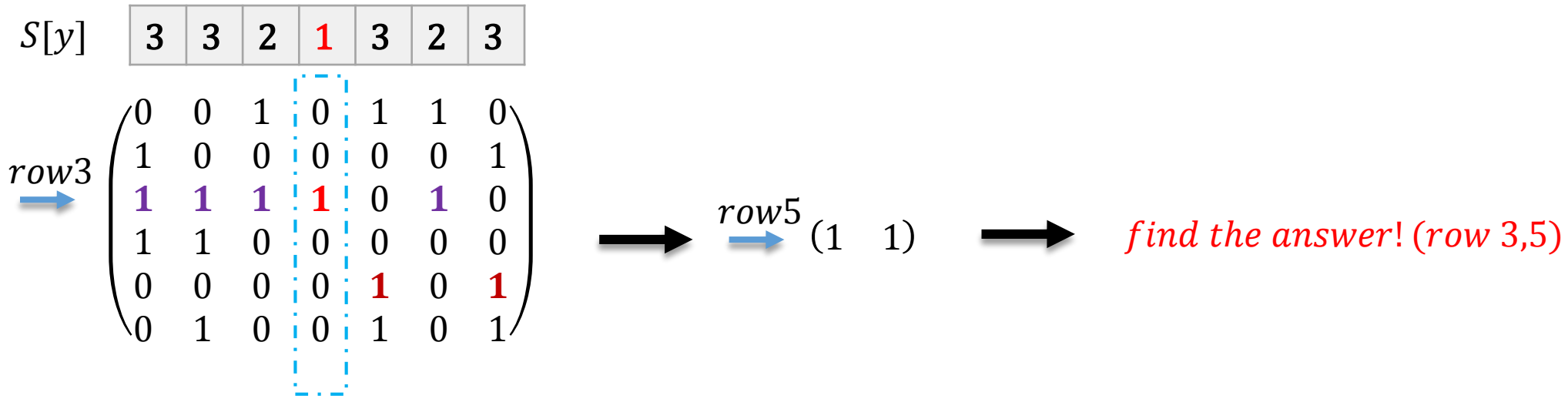
find the answer! (*row* 4,5,1)



DLX Algorithm (cont.)

- Heuristic $S[y]$
  - *select the smallest* $S[y]$ *of col*
  - $S[y]$ is the number of 1

```
int c = R[0];
for( int i = R[0]; i != 0; i = R[i] )
    if( S[i] < S[c] )
        c = i;
```



DLX Algorithm (cont.)

- Static Linked List

```
struct DLX{
    int n, m, cnt;
    int L[maxnode], R[maxnode], U[maxnode], D[maxnode], row[maxnode], col[maxnode];
    int S[MAXC], H[MAXR], o[MAXR];
    void remove( int c ){
        L[R[c]] = L[c]; R[L[c]] = R[c];
        for( int i = D[c]; i != c; i = D[i] ){
            for( int j = R[i]; j != i; j = R[j] ){
                U[D[j]] = U[j];
                D[U[j]] = D[j];
                --S[col[j]];
            }
        }
    }
    void resume( int c ){
        for( int i = U[c]; i != c; i = U[i] ){
            for( int j = L[i]; j != i; j = L[j] ){
                U[D[j]] = D[U[j]] = j;
                ++S[col[j]];
            }
        }
        L[R[c]] = R[L[c]] = c;
    }
```

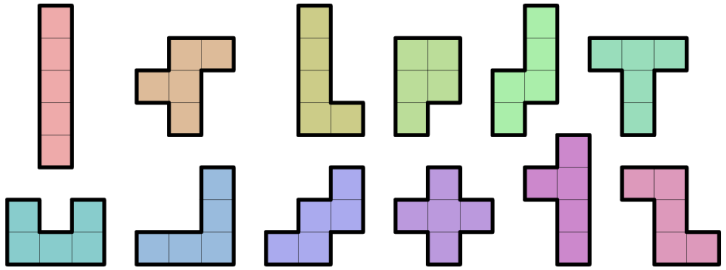
```
bool dancing( int d ){
    if( R[0] == 0 ) return true;
    int c = R[0];
    for( int i = R[0]; i != 0; i = R[i] )
        if( S[i] < S[c] )
            c = i;
    remove(c);
    for( int i = D[c]; i != c; i = D[i] ){
        o[d] = row[i];
        for( int j = R[i] ; j != i; j = R[j] ) remove( col[j] );
        if( dancing( d + 1 ) ) return true;
        for( int j = L[i] ; j != i; j = L[j] ) resume( col[j] );
    }
    resume(c);
    return false;
}
```

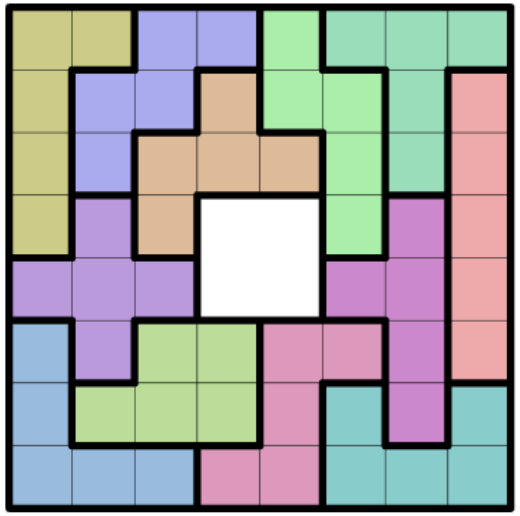
Outline

- Introductions
  - Exact Cover Problem
- Dancing Links and X algorithm
- Application and Comparison
  - Polyomino
  - Sudoku
  - N queens puzzle
- Conclusion

Polyomino and Exact Cover

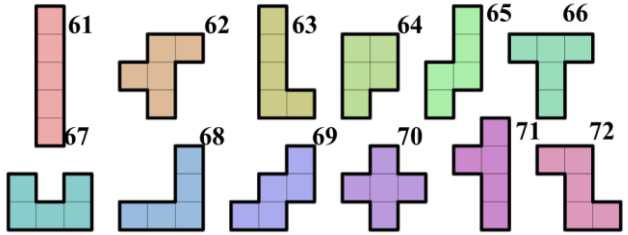
- $R(n, r, p)$
  - n – the n-th *pentominoes*
  - r – rotate (0,90,180,270)
  - p – position (x,y)





- E.g.
  - R(1,0,(2,8))
  - R(2,270,(3,3))
  - R(10,0,(5,1))
- Find a **Set of R** to cover the squares without conflict

Polyomino and Exact Cover (cont.)



12 *pentominoes*

60 *squares*

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27			28	29	30
31	32	33			34	35	36
37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60

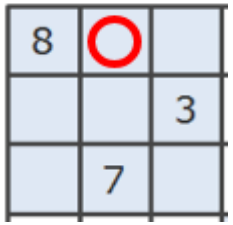
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27			28	29	30
31	32	33			34	35	36
37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52
53	54	55	56	57	58	59	60

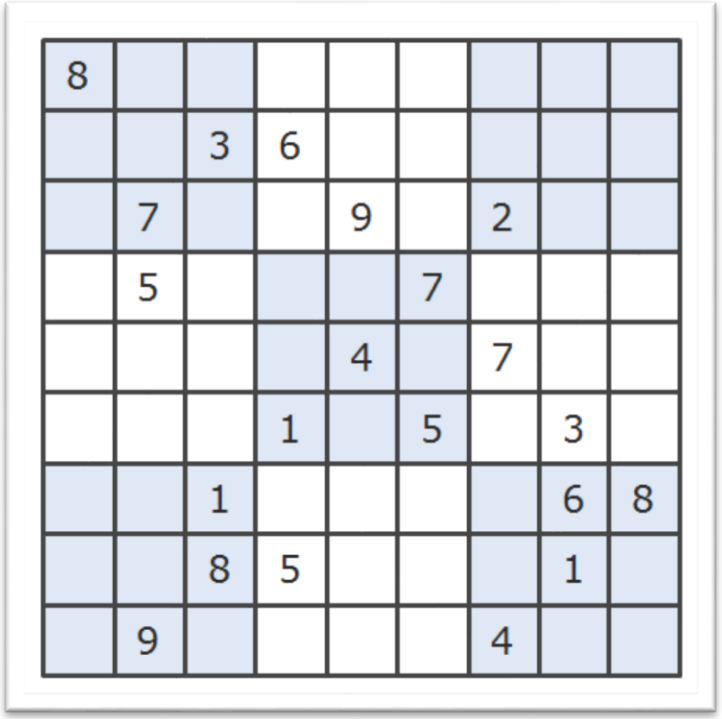
	1	2	3	⋯	9	10	11	12	⋯	16	17	18	19	⋯	60	61	⋯	70	71	72
$R(10,0,(2,1))$	0	1	0	0	1	1	1	0	⋯	0	0	1	0	⋯	0	0	⋯	1	0	0
$R(10,0,(2,2))$	0	0	1	0	0	1	1	1	⋯	0	0	0	1	⋯	0	0	⋯	1	0	0
	⋮																	⋱		⋮
																		⋯		

Sudoku and Exact Cover

- $R(n, p)$
  - n – the n-th *number*
  - p – position (x,y)

- E.g.
  - R(8,(1,1))
  - R(3,(2,3))
  - R(7,(3,2))
  - R(1,(1,2)), R(2,(1,2)), R(4,(1,2)),… R(9,(1,2))

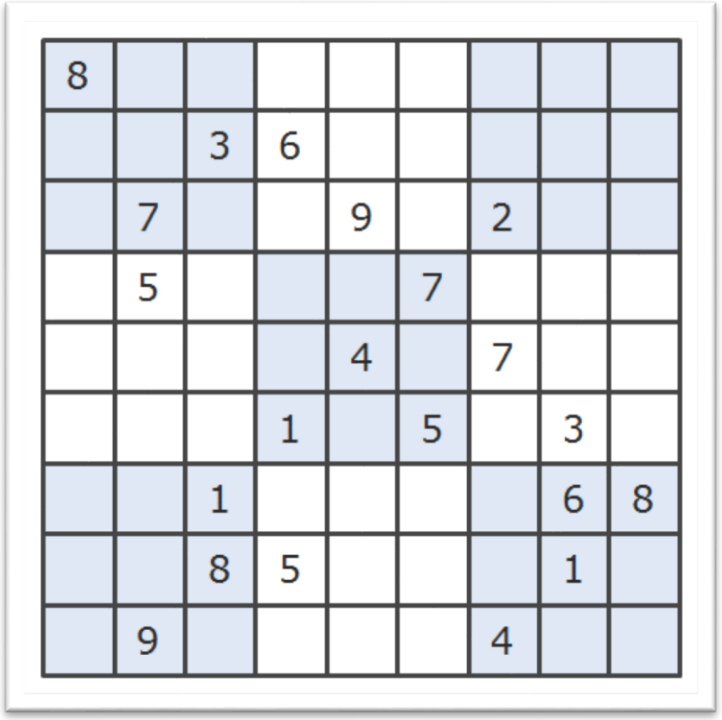




- Find a **Set of R** to cover the Sudoku without conflict

Sudoku and Exact Cover (cont.)

- Rows ($< N^3$):
  - Each number is placed in each square.
- Columns ($4 * N^2$):
  - Position Limit:
    - A square only one number.
  - Column Limit
    - Each number appears only once in a column.
  - Row Limit
    - Each number appears only once in a row.
  - Area Limit
    - Each number appears only once in a area.



R(8,(1,1))
R(3,(2,3))

Sudoku and Exact Cover (cont.)

- Position Limit (1 – 16)
- Column Limit
- Row Limit
- Area Limit

$4 * 4$ *sudoku*

2	4		
1		2	
			2
4			

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4	4-1	4-2	4-3	4-4

matrix col number

$r_i c_j$ *placed or not*

$$\begin{pmatrix} 1 & & & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & & & \\ & & & & 1 & & & & & & & & & & & \\ & & & & & & 1 & & & & & & & & & \\ & & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & & 1 & & & \end{pmatrix}$$

Sudoku and Exact Cover (cont.)

- Position Limit $(1 - 16)$
- Column Limit $(17 - 32)$
- Row Limit
- Area Limit

$4 * 4$ *sudoku*

2	4		
1		2	
			2
4			

17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4	4-1	4-2	4-3	4-4

matrix col number

col i placed j

$$\begin{pmatrix} & 1 & & & & & & & & & & & & & & \\ & & & & & & & 1 & & & & & & & & \\ 1 & & & & & & & & & & & & & & & \\ & & & & & & & & & 1 & & & & & & \\ & & & & & & & & & & & & & 1 & & \\ & & & 1 & & & & & & & & & & & & \end{pmatrix}$$

Sudoku and Exact Cover (cont.)

- Position Limit $(1 - 16)$
- Column Limit $(17 - 32)$
- Row Limit $(33 - 48)$
- Area Limit

$4 * 4$ *sudoku*

2	4		
1		2	
			2
4			

33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4	4-1	4-2	4-3	4-4

matrix col number

rol i placed j

$$\begin{pmatrix} & 1 & & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ 1 & & & & & & & & & & & & & & & \\ & 1 & & & & & & & & & & & & & & \\ & & & & & & & & & 1 & & & & & & \\ & & & & & & & & & & & & & & & 1 \end{pmatrix}$$

Sudoku and Exact Cover (cont.)

- Position Limit (1 – 16)
- Column Limit (17 – 32)
- Row Limit (33 – 48)
- Area Limit (49 – 64)

$4 * 4$ *sudoku*

2	4		
1		2	
			2
4			

box number

1	2
3	4

49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
1-1	1-2	1-3	1-4	2-1	2-2	2-3	2-4	3-1	3-2	3-3	3-4	4-1	4-2	4-3	4-4

matrix col number

box i placed j

$$\begin{pmatrix} & 1 & & & & & & & & & & & & & & \\ & & & 1 & & & & & & & & & & & & \\ 1 & & & & & & & & & & & & & & & \\ & & & & & 1 & & & & & & & & & & \\ & & & & & & & & & & & & & 1 & & \\ & & & & & & & & & & & 1 & & & & \end{pmatrix}$$

Sudoku and Exact Cover (cont.)

- Position Limit $(1 - 16)$
- Column Limit $(17 - 32)$
- Row Limit $(33 - 48)$
- Area Limit $(49 - 64)$
- Total columns $(4 * N^2 = 64)$

$4 * 4\ sudoku$

2	4		
1		2	
			2
4			

$box\ number$

1	2
3	4

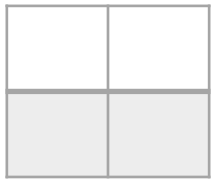
1	2	…	18	…	24	…	34	35	36	…	50	51	52	…	64

$matrix\ col\ number$

$$\mathrm{MAX}(rows) = N^3 \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ & 1 & & & & 1 & & & & 1 & & & & 1 & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & & \end{pmatrix}$$

Sudoku and Exact Cover (cont.)

$E.g. 2 * 2$ Sudoku



build matrix

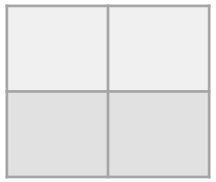
$row: 2 * 2 * 2 = 8$

$col: 4 * 2 * 2 = 16$

Choice	Constraint															
	Number in row				Number				Number				Number			
	1		2		1		2		1		2		1		2	
	and column				must in row				must in column				must in region			
	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
1 at (1,1)	1				1				1				1			
2 at (1,1)	1						1				1				1	
1 at (1,2)		1			1					1				1		
2 at (1,2)		1					1					1				1
1 at (2,1)			1			1			1				1			
2 at (2,1)			1					1			1				1	
1 at (2,2)				1		1				1				1		
2 at (2,2)				1				1				1				1

Sudoku and Exact Cover (cont.)

$E.g.\ 2 * 2$ Sudoku



build matrix

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}$$

DLX

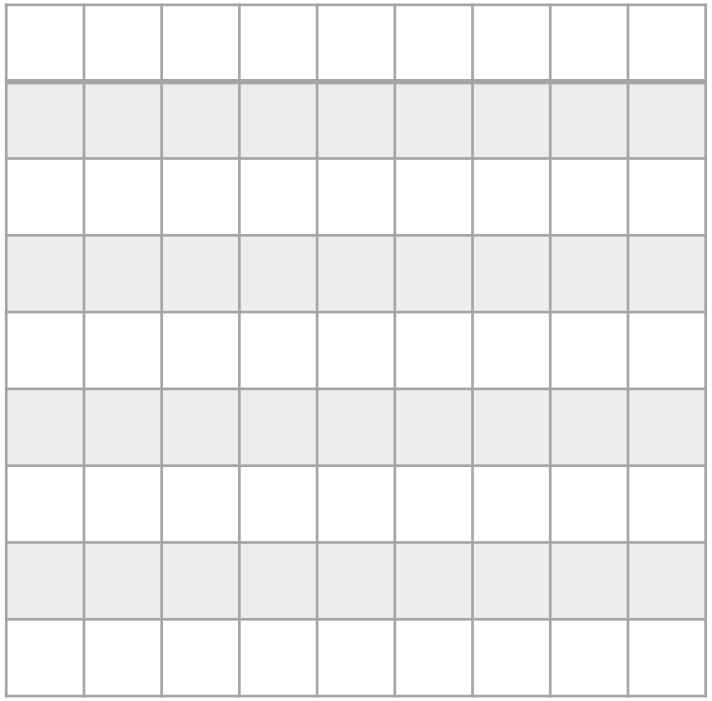
1 at (1,1)	1				1				1				1			
2 at (1,2)		1					1					1				1
2 at (2,1)			1					1			1				1	
1 at (2,2)				1		1				1				1		

get answer

1	**2**
2	**1**

Sudoku and Exact Cover (cont.)

$E.g. 9 * 9$ Sudoku



build matrix

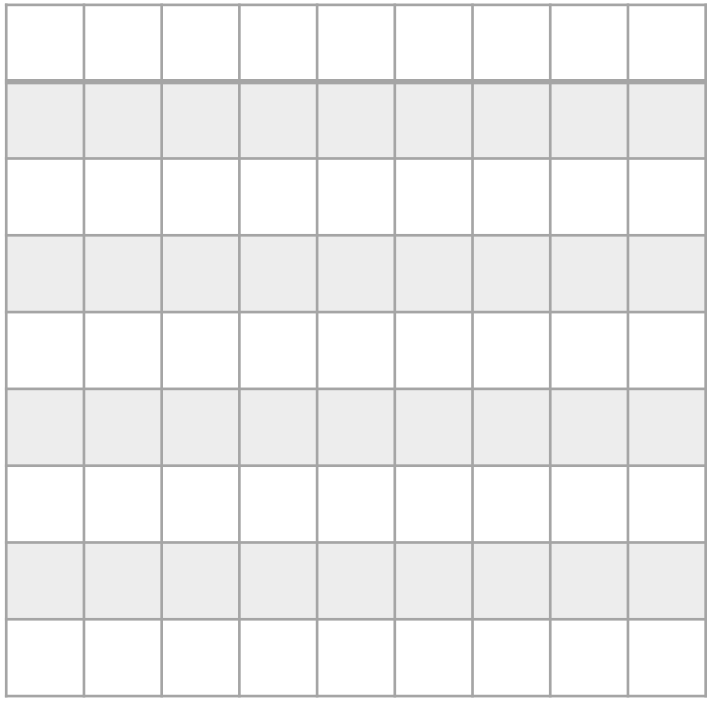
$$\begin{pmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{pmatrix}$$

$row: < 9 * 9 * 9 = 729$

$col: 4 * 9 * 9 = 324$

Sudoku and Exact Cover (cont.)

$E.g. 9 * 9$ Sudoku



build matrix

```
void place( int &r, int &c1, int &c2, int &c3, int &c4, int i, int j, int k ){
    r = (i*N+j)*N + k;
    c1 = i*N+j+1;
    c2 = N*N*1 + i*N + k;
    c3 = N*N*2 + j*N + k;
    c4 = N*N*3 + ((i/3)*3+(j/3))*N + k;
}

  dlx.init( N*N*N, N*N*4 );

  for( int i = 0; i < N; ++i )
      for( int j = 0;j < N; ++j )
          for( int k = 1; k <= N; ++k )
              if( g[i*N+j] == '.' || g[i*N+j] == '0' + k ){
                  place( r, c1, c2, c3, c4, i, j, k );
                  dlx.link( r, c1 );
                  dlx.link( r, c2 );
                  dlx.link( r, c3 );
                  dlx.link( r, c4 );
              }

  dlx.dancing(0);
```

c1: 1-81
c2 82-162
c3:163-243
c4:244-324

Sudoku and Exact Cover (cont.)

- I just use qqwing to generate Sudoku boards.
- For each difficulty setting (easy, intermediate, expert)
  - 200 boards were randomly generated.

- DLX is around **60-140** faster than DFS
  - This is just 9*9 Sudoku, 16*16 Sudoku will be more faster!

	Easy	*Intermediate*	*Expert*
Naive DFS Algorithm	2484ms	3095ms	2008ms
Dancing Links	**28ms**	**22ms**	**31ms**
Times	**88**	**140**	**64**

N-queens and Exact Cover

- Columns ($6N-2$):
  - Column Limit (N):
    - Each column can only place a queen.
  - Row Limit (N):
    - Each row can only place a queen.
  - Diagonal Limit ($2N-1$):
    - Each diagonal can only place a queen.
- Rows (N^2):
  - There are a total of N*N squares.

$|x-y|$

	1	2	3	4
1	0	1	2	3
2	1	0	1	2
3	2	1	0	1
4	3	2	1	0

$x+y$

	1	2	3	4
1	2	3	4	5
2	3	4	5	8
3	4	5	6	7
4	5	6	7	8

$row: N * N = 4 * 4 = 16$

$col: N + N + 2(2N-1) = 4 + 4 + 2 * 7 = 22$

N-queens and Exact Cover (cont.)

$E.g. 4-queens$

	1	2	3	4
1				
2				
3				
4				

build matrix →

$$\begin{pmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{pmatrix}$$

$row: 4 * 4 = 16$

$col: 6 * 4 - 2 = 22$

N-queens and Exact Cover (cont.)

$E.g. 4-queens$

	1	2	3	4
1				
2				
3				
4				

build matrix →

```
dlx.init( N*N , 6*N-2 );

for( int i = 1; i <= N; ++i ){
    for( int j = 1; j <= N; ++j ){
        int c1 = i;
        int c2 = N + j;
        int c3 = 2*N + N + i - j;
        int c4 = 4*N + i + j - 2;

        len++;
        dlx.link( len, c1 );
        dlx.link( len, c2 );
        dlx.link( len, c3 );
        dlx.link( len, c4 );
    }
}

dlx.dancing(0);
```

Outline

- Introductions
  - Exact Cover Problem
- Dancing Links and X algorithm
- Application and Comparison
  - Polyomino
  - Sudoku
  - N queens puzzle
- Conclusion

Conclusion

- DLX is a simple and beautiful algorithm.
- It can solve Exact Cover Problem efficiently.
  - It can also solve Overlapping Cover Problem.
- Thanks for Donald E. Knuth.

Let’s dance ! Thank you!

Reference

- Solving Sudoku with Dancing Links
- Knuth's Algorithm X and Dancing Links
- Exact cover I
- 跳跃的舞者，舞蹈链（Dancing Links）算法