


Quantum Gaussian Elimination


<https://youtu.be/nw0TkoGctqk>


장경배

Information Set Decoding (ISD)

- ISD 연산은 코드기반암호에 대한 **효율적인 Brute-force**
 - 공개키의 **Information set** () , 행렬 H_{n-k} 을 변경해가며 전수 조사
 - **Information set의 역행렬** \times **암호문** (C) 의 결과 벡터가 특정 (Gaussian Elimination 사용) low-weight (t)를 만족할 경우, 원본 메시지 복구 (e)에 성공

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

H_{n-k}

Brute-force


 Information set (inverse)⁻¹

\times

C

$=$

$\begin{bmatrix} e \\ \text{(Weight check } \rightarrow t?) \end{bmatrix}$

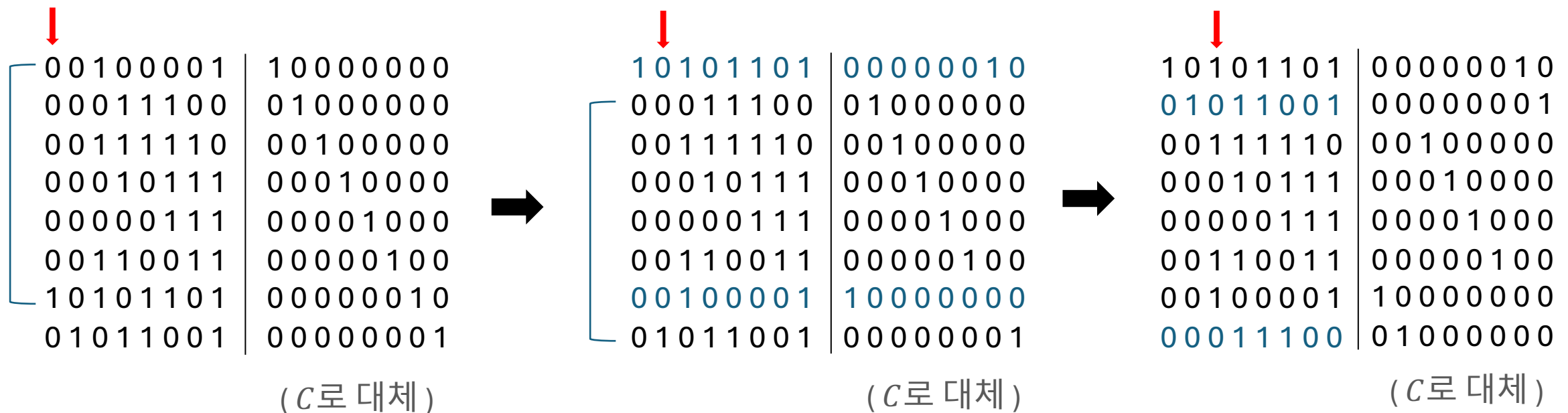
$$H = \left(\begin{array}{cccccc|cccc} & & & & & & \textcolor{blue}{0} & \textcolor{blue}{0} & \textcolor{blue}{H_{n-k}} & & & & & & & \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array} \right)$$

$$C = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

$$e = (0000001100000000), (n = 16, t = 2)$$

Secret

$$C = He = (00000011)$$



1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0
0	0	1	1	0	0	1	1	0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

(C로 대체)



1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0
0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0

(C로 대체)



1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0

(C로 대체)

1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0

(C로 대체)



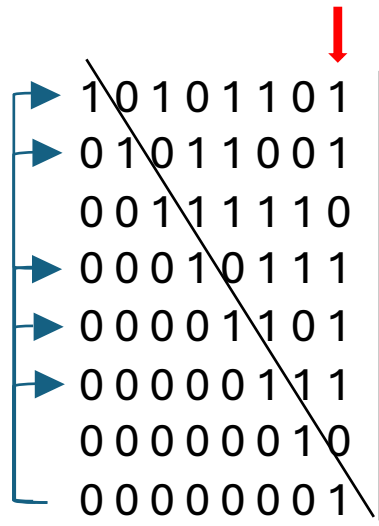
1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

(C로 대체)



1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	1	0	0	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

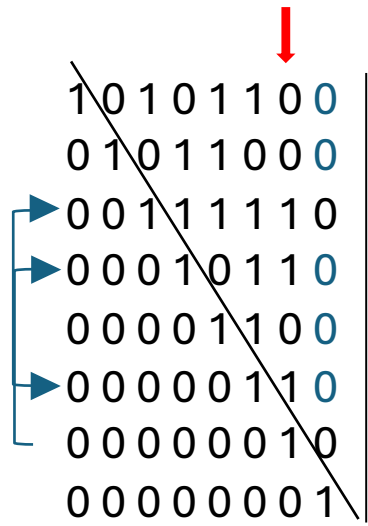
(C로 대체)



1	0	1	0	1	1	0	1
0	1	0	1	1	0	0	1
0	0	1	1	1	1	1	0
0	0	0	1	0	1	1	1
0	0	0	0	1	1	0	1
0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0
0	0	0	0	1	0	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	0	0	

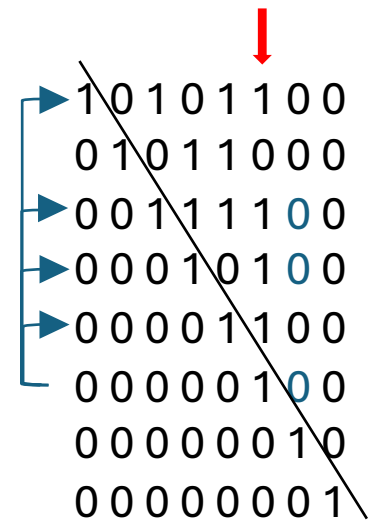
(C로 대체)

1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	0
0	0	1	1	1	1	1	0
0	0	0	1	0	1	1	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

0	1	1	1	1	1	1	0
0	1	1	1	1	1	0	1
0	0	1	0	0	0	0	0
0	1	1	0	1	1	0	0
0	1	0	1	1	0	0	0
0	1	1	1	0	1	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	0

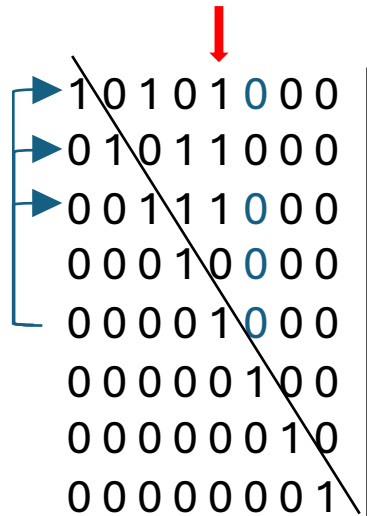
(C로 대체)

1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	0	1	0	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

0	1	1	1	1	1	1	0
0	1	1	1	1	1	0	1
1	0	1	1	1	1	0	0
1	1	1	1	0	0	0	0
0	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	0

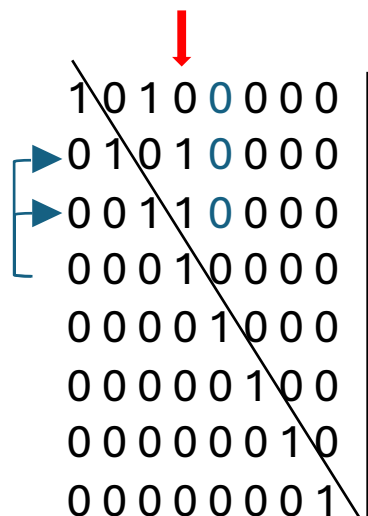
(C로 대체)



1	0	1	0	1	0	0	0
0	1	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

1	0	0	1	0	1	1	0
0	1	1	1	1	1	0	1
0	1	0	1	0	1	0	0
0	0	0	1	1	0	0	0
1	0	1	1	0	0	0	0
1	1	1	0	1	0	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	0

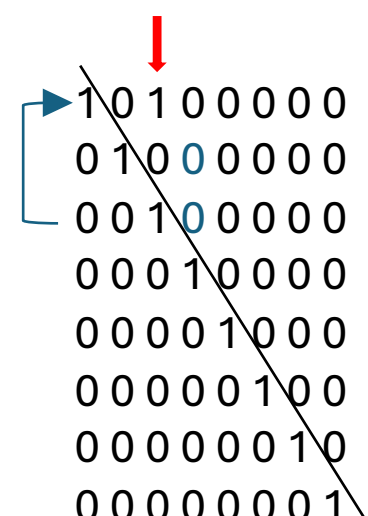
(C로 대체)

1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

0	0	1	0	0	1	1	0
1	1	0	0	1	1	0	1
1	1	1	0	0	1	0	0
0	0	0	1	1	0	0	0
1	0	1	1	0	0	0	0
1	1	1	0	1	0	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	0

(C로 대체)

1	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

0	0	1	0	0	1	1	0
1	1	0	1	0	1	0	1
1	1	1	1	1	1	1	0
0	0	0	1	1	0	0	0
1	0	1	1	0	0	0	0
1	1	1	0	1	0	0	0
1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	0

(C로 대체)

$$\begin{array}{c|c}
 \begin{array}{c} \text{blue arrow} \rightarrow \\ \text{red arrow} \downarrow \end{array}
 \begin{array}{l}
 10000000 \\
 01000000 \\
 00100000 \\
 00010000 \\
 00001000 \\
 00000100 \\
 00000010 \\
 00000001
 \end{array}
 &
 \begin{array}{l}
 11011010 \\
 11010101 \\
 11111100 \\
 00011000 \\
 10110000 \\
 11101000 \\
 10011100 \\
 01111100
 \end{array}
 \end{array}$$

(C로 대체)

$$(H_{n-k})^{-1} =$$

$$\begin{array}{l}
 11011010 \\
 11010101 \\
 11111100 \\
 00011000 \\
 10110000 \\
 11101000 \\
 10011100 \\
 01111100
 \end{array}$$

Invertible하지 않다?

만들 수 없음 →

$$\begin{array}{c}
 1 \\
 01 \\
 001 \\
 0001 \\
 00001 \\
 000001 \\
 0000001 \\
 00000001
 \end{array}$$

- $(H_{n-k})^{-1} \times C^T$ 의 Hamming weight가 t 인 경우, 공격 성공

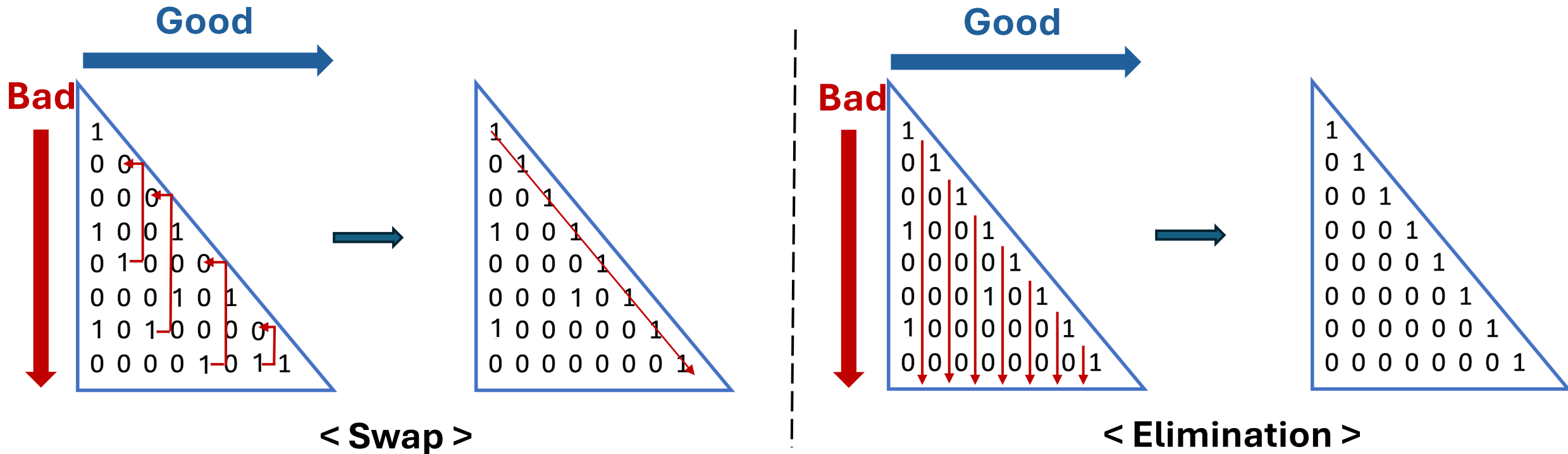
$$\begin{pmatrix}
 11011010 \\
 11010101 \\
 11111100 \\
 00011000 \\
 10110000 \\
 11101000 \\
 10011100 \\
 01111100
 \end{pmatrix}
 \times
 \begin{pmatrix}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 1 \\
 1
 \end{pmatrix}
 =
 \begin{pmatrix}
 1 \\
 1 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{pmatrix}
 \rightarrow \text{Weight} = 2(t)$$

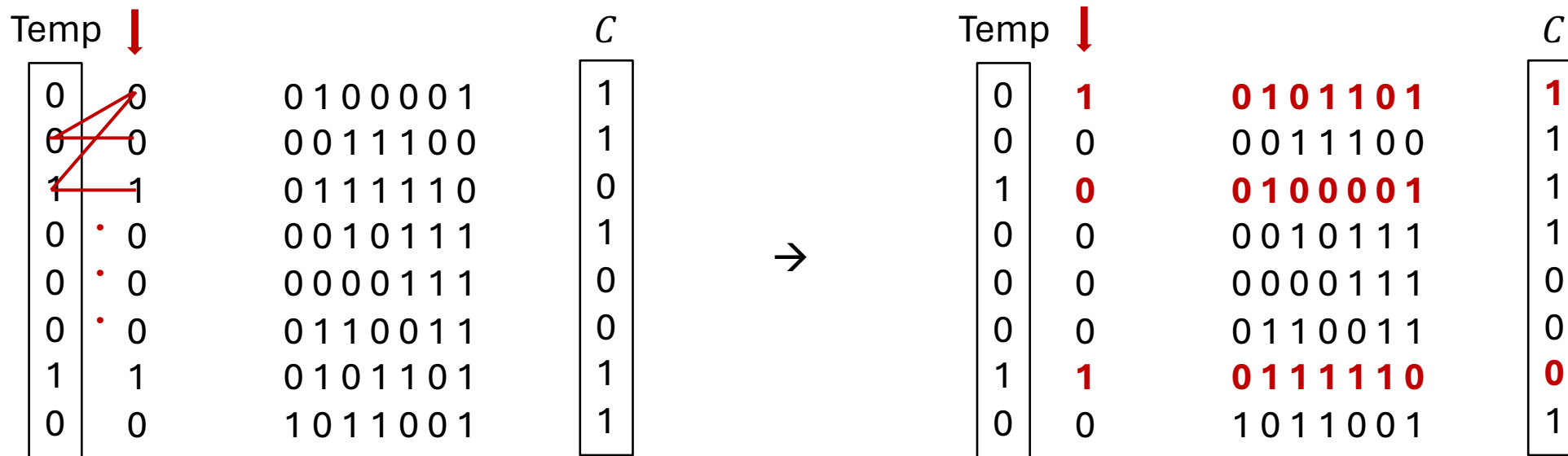
$$(H_{n-k})^{-1}$$

$$C^T$$

Gaussian Elimination: Parallelization

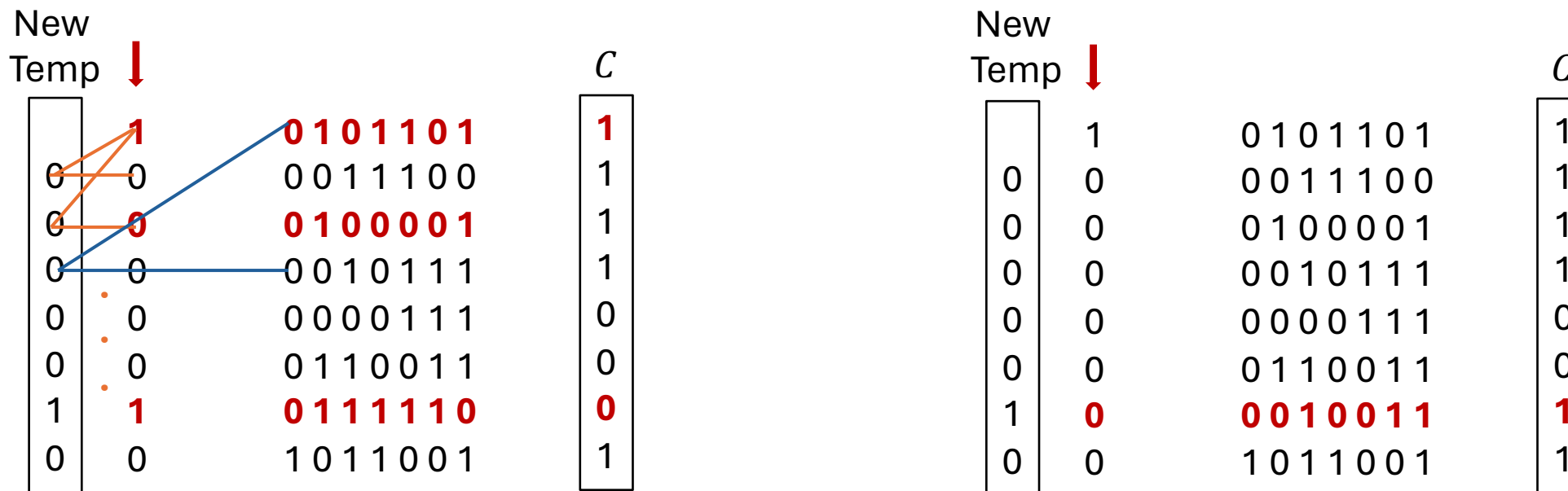
- Row 대상의 병렬화에 대한 **성능은 좋지 않았음**, 반면, Column 대상의 병렬화에 대한 **성능은 우수**
 - Column에 대한 병렬화만 수행 → Row 대상은 추가 큐비트 대비, Depth 감소가 낮음





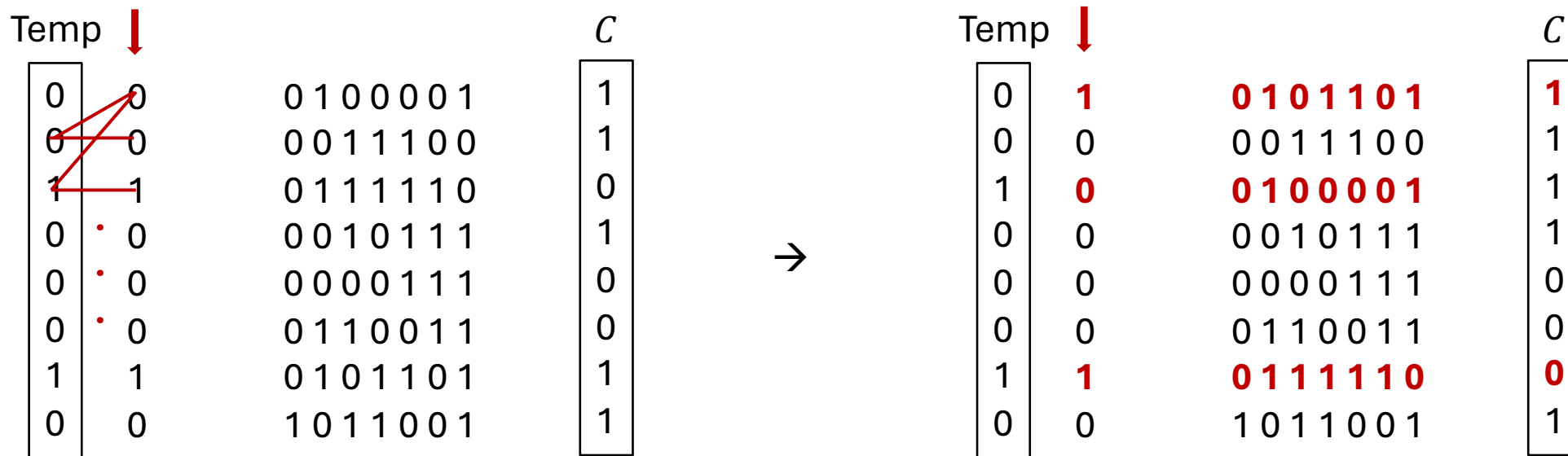
CSWAP

- Temp는 Garbage가 되고, 새로운 Temp에 현재 값 Copy

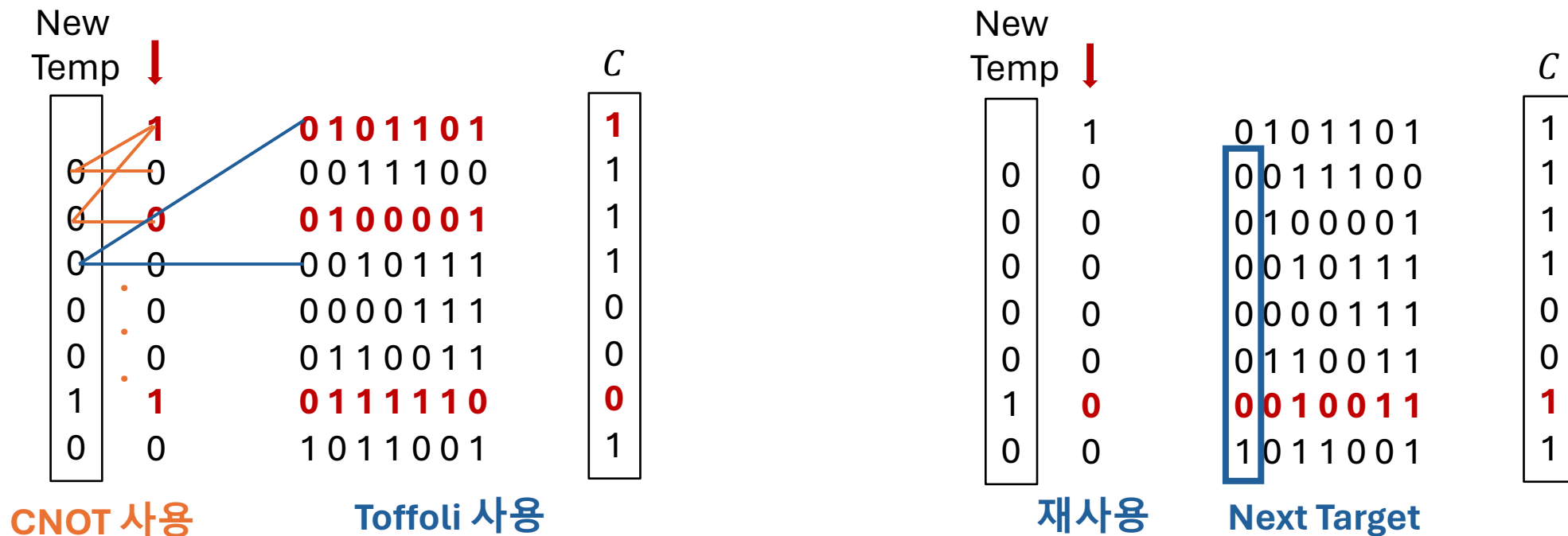


CNOT 사용

Toffoli 사용

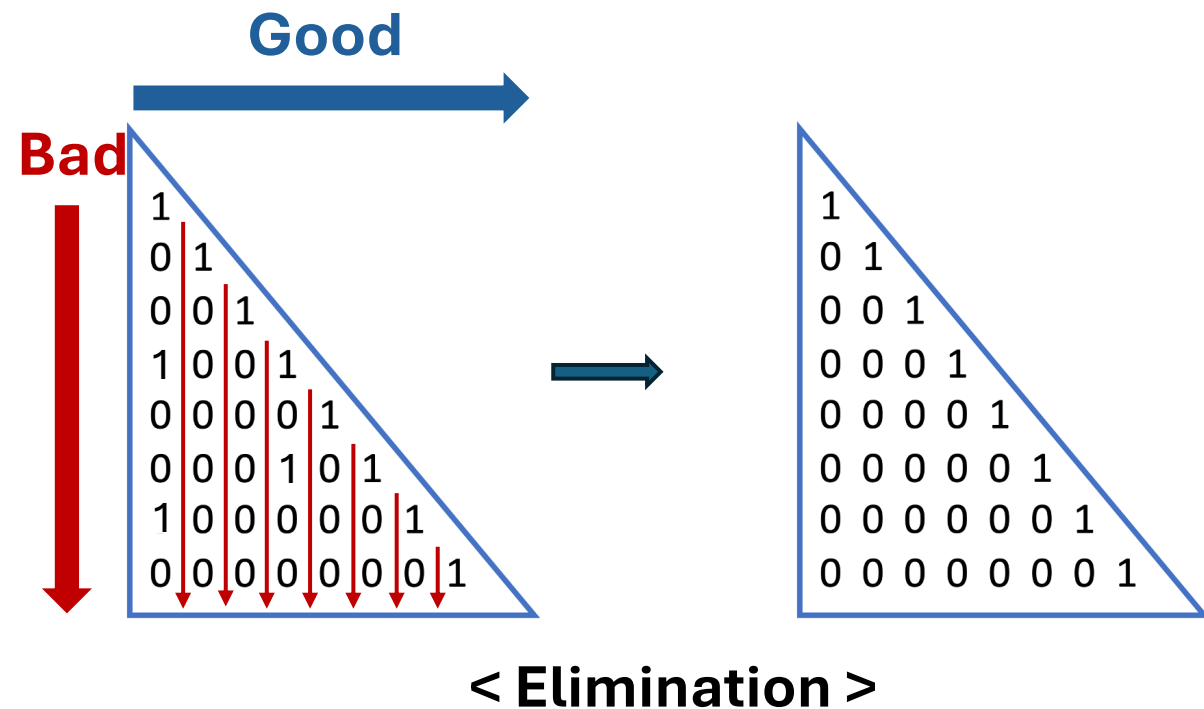
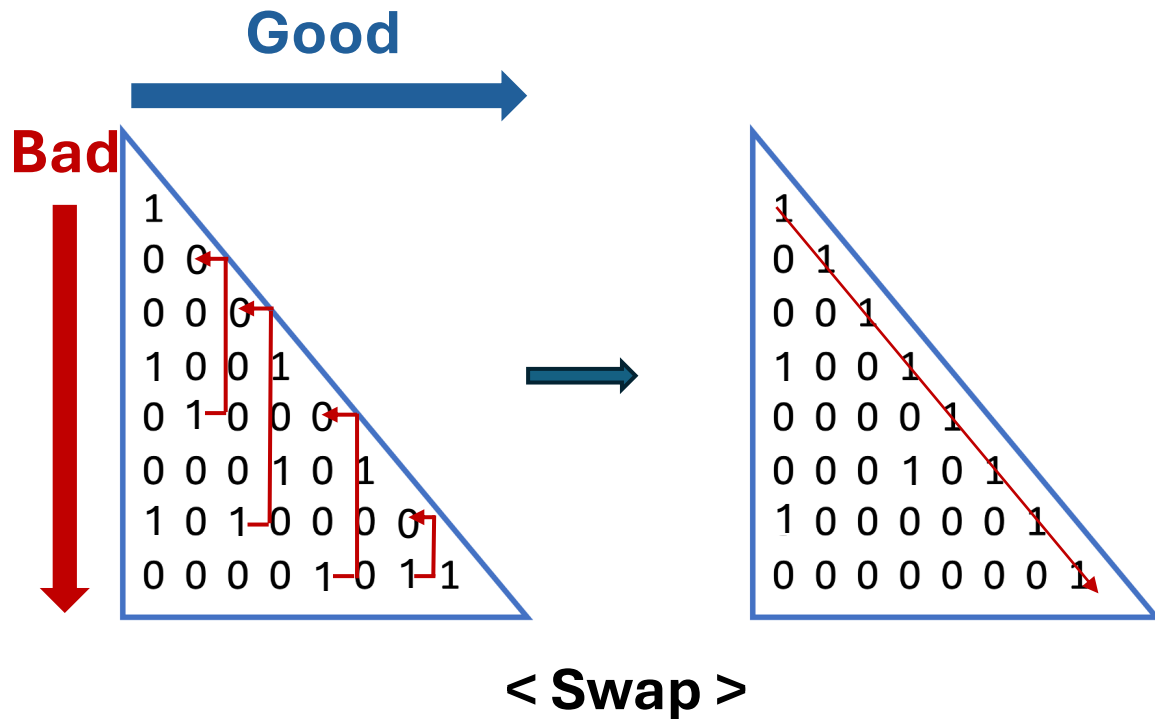


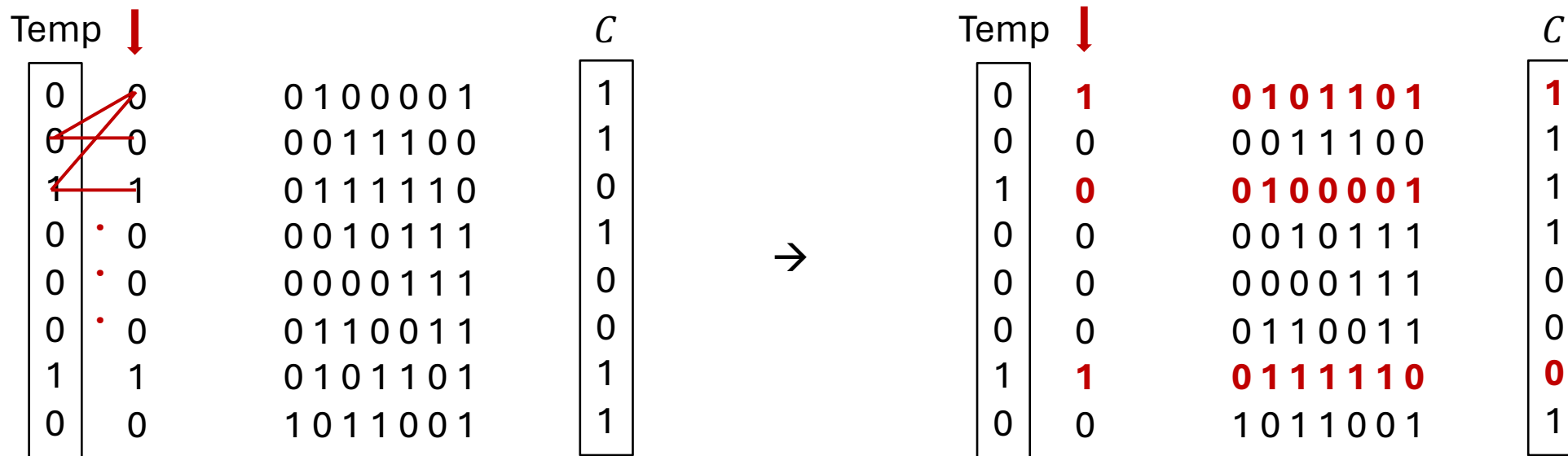
-
- Temp는 Garbage가 되고, 새로운 Temp에 현재 값 Copy



Gaussian Elimination: Parallelization

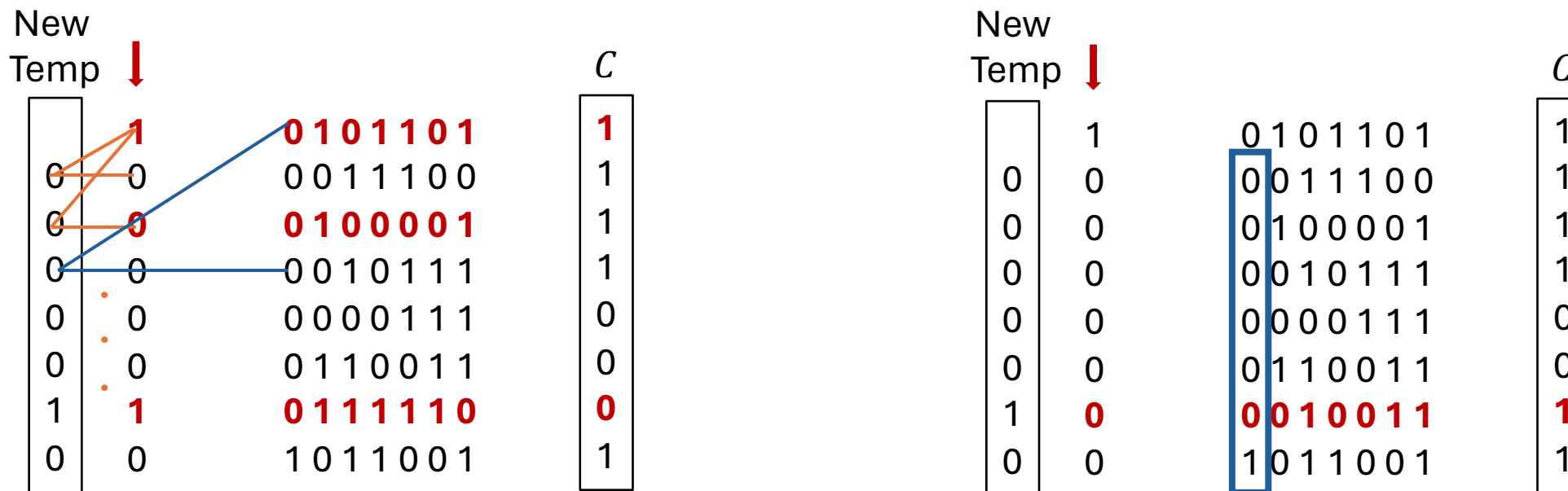
- Row 대상의 병렬화에 대한 **성능은 좋지 않았음**, 반면, Column 대상의 병렬화에 대한 **성능은 우수**
 - Column에 대한 병렬화만 수행 → Row 대상은 추가 큐비트 대비, Depth 감소가 낮음





CSWAP

- Temp는 Garbage가 되고, 새로운 Temp에 현재 값 Copy



CNOT 사용

Toffoli 사용

재사용

Next Target

다음 Swap과 중첩되기 때문

Gaussian Elimination: Parallelization

- Gauss-Jordan elimination 양자 회로 병렬화
- 기존 구현 [*]

Matrix size	Method	Qubits	X	CX (CNOT)	CCX (Toffoli)	CCCX	Multi-Controlled Swap	Depth
32 x 32	Gauss-Jordan Elimination	1,120	992	1,054	6,448	133,672	244,035	288,238

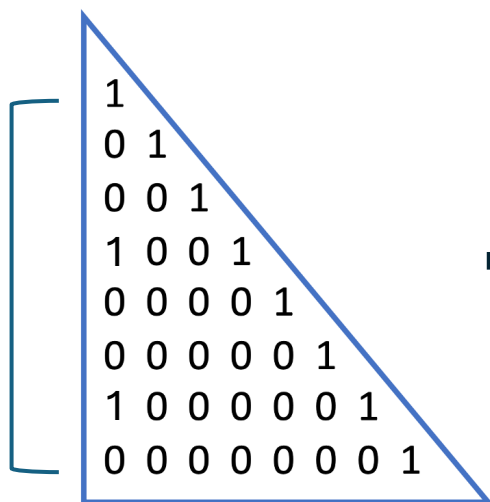
- 신규 구현: 회로 복잡도 및 Depth 대폭 감소
 - 병렬화를 통한 2차 개선

Matrix size	Method	Source	Qubits	Clifford	T	Full depth
32 x 32	Gauss-Jordan Elimination	신규	1,584	205,933	159,712	10,339
		신규 + 병렬화	2,110	249,550	159,712	6,144

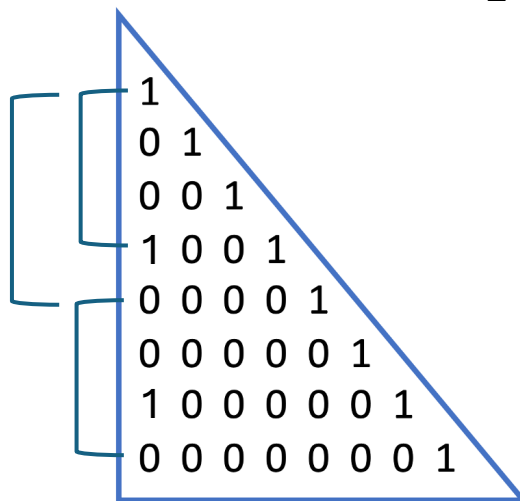
개선 방향?

- **Elimination 병렬화를 가져 가면서, Swap에 대한 부분 병렬화 (완전 병렬화 X)**
 - $O(n) \rightarrow O(n/2^{\text{parallel_level}})$

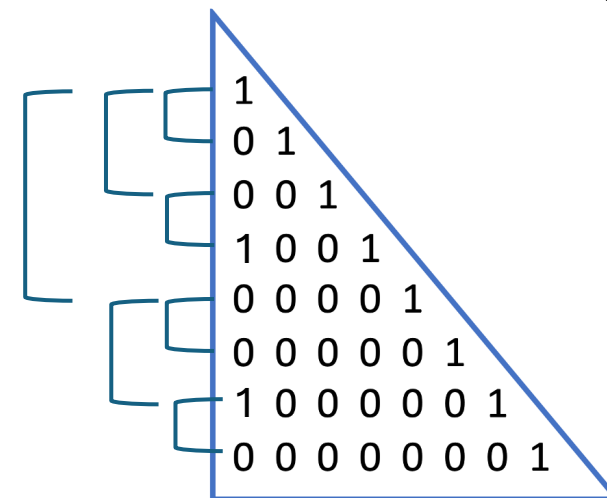
Complexity: $O(n)$



Complexity: $O(\frac{n}{2})$



Complexity: $O(\frac{n}{2^2})$





Thank you!