東京大学
量子ソフトウェア寄付講座

第9回 量子ソフトウェアハンズオン

2026年 2月 12日

本日の演習の概要

- 量子回路における誤り訂正符号について、Qiskitのシミュレータを用いて実装を行います。
- 演習を通じて、以下の目標を達成することを目指します。

本日の目標

1. 量子回路の誤り訂正符号の実装方法を学ぶ。
2. 誤り訂正符号が誤り訂正に成功する場合/失敗する場合を確認する。
3. 符号化された状態での論理ビット操作を試してみる。
4. 実際の誤り訂正において、求められる性質について考えてみる。

演習の内容 - 誤り訂正

- 前述の目標を達成するため、本日は４つの演習に取り組んでいただきます。
- 分からない点などがあれば、積極的に周囲の人と相談しながら取り組んでください。
  - 講師側でも適宜サポートに回ります。

#	演習テーマ	概要
1	Shorの符号の実装	• 1論理量子ビットの誤り訂正を行う方法について、Shorの符号の実装を行います。 • 誤り訂正が成功する場合／失敗する場合を実際に確認します。
2	Shorの符号の論理ビット操作	• 符号化された論理量子ビットの状態で、各種操作を実施する方法を体験します。 • 今回の演習では、ベルンシュタイン・ヴァジラニのアルゴリズムと呼ばれる簡単な回路を題材にします。
3	Steane符号の実装と論理ビット操作	• Steane符号と呼ばれる別の符号について、実装方法と論理ビット操作を体験します。 • 今回の演習では、2論理量子ビットを用いてベル状態を作成する回路を題材にします。
4	表面符号の実装と論理ビット操作	• 表面符号と呼ばれる符号について、実装方法を体験します。 • 今回の演習では、初歩的な操作演習に留めます。

演習環境の準備

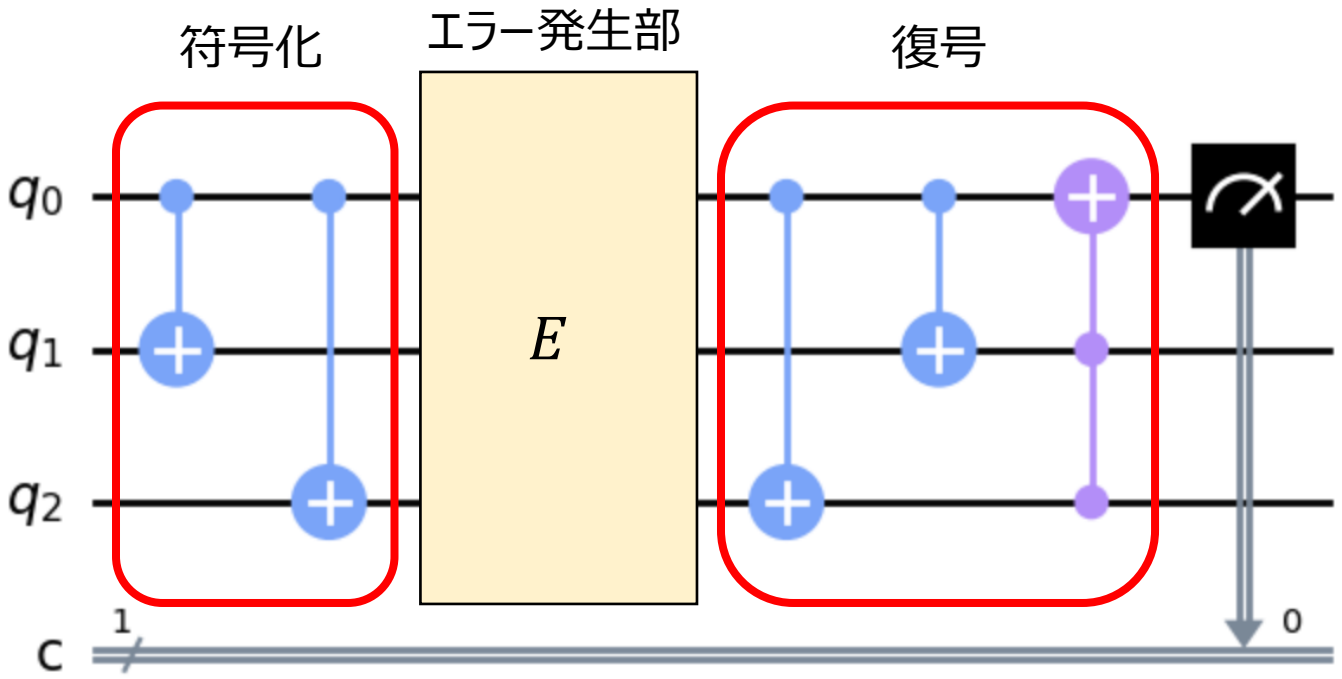
- 今回の演習はGoogle Colaboratoryを使って、演習・解説を行います
- 各自のPCのローカル環境上で行っていただいても構いませんが、ご自身の環境に依存するエラー対応については、自己責任でお願いいたします

今回の講義のgithubページに５つのノートブックを掲載してあります。

ご自身のドライブにコピーしていただき、Google Colab環境で「01_誤り訂正符号の実装.ipynb」を開いてください。

演習1. 誤り訂正符号の実装

- 最初に、1量子ビットのビット反転エラーを修正できる回路を作成します。
- 古典コンピュータにおけるシンプルな多数決による修正方法と基本的には同じです。



演習1．誤り訂正符号の実装

- 1量子ビットのビット反転エラーを修正できる回路では、2量子ビット以上がビット反転してしまうと、エラーを修正できません。
- しかしながら、2量子ビット以上でエラーが発生する確率が、1量子ビットでのエラー発生率より低ければ、エラーが発生する確率を下げる役割を果たせます。

α：1量子ビットあたりのエラー発生確率

エラーが発生する確率

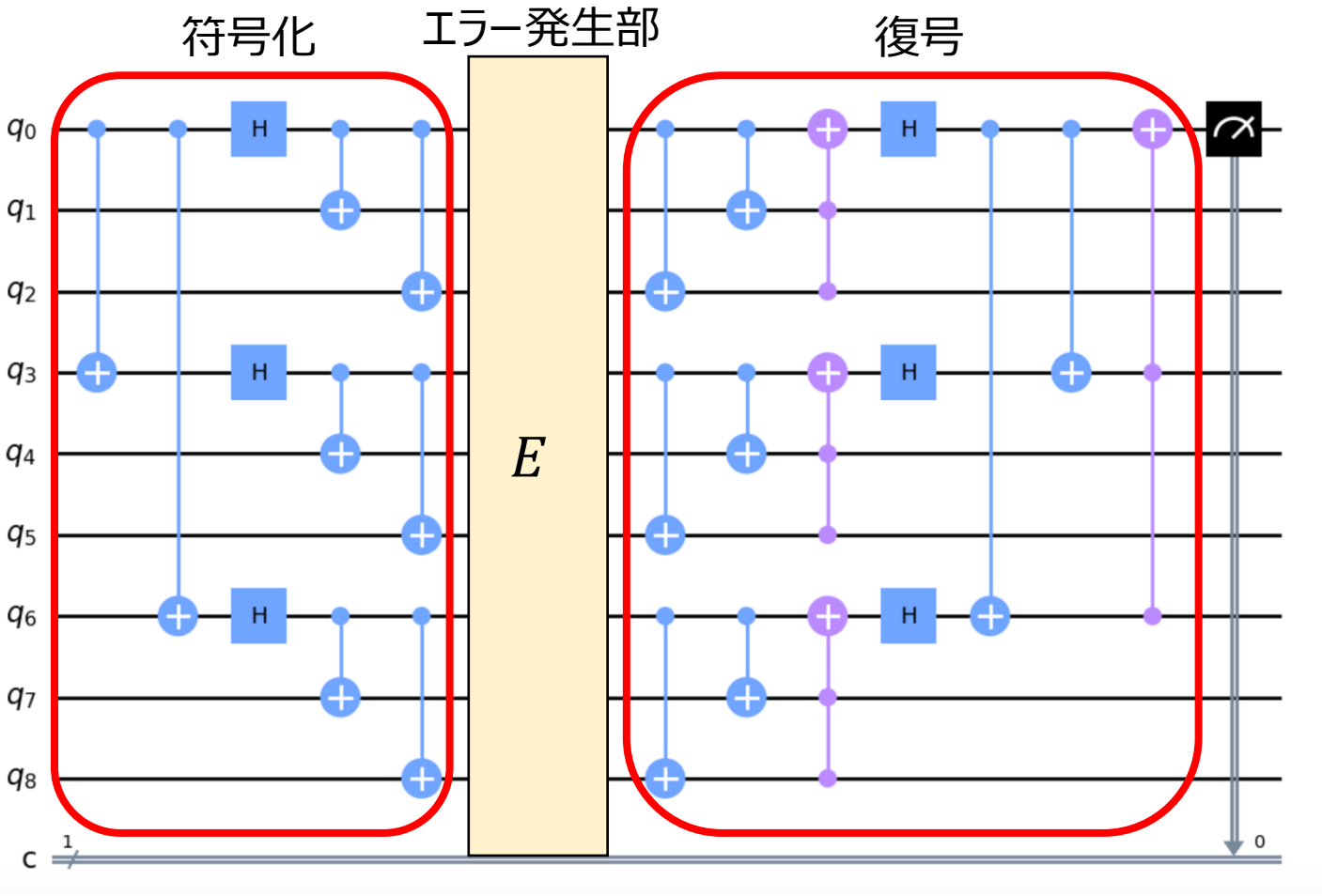
$$1-\underline{(1-\alpha)^3+3\alpha(1-\alpha)^2}$$

エラーが訂正できる確率
（エラー無しの確率）＋（1量子ビットのみエラーの確率）

$\alpha = 0.5$のとき、ちょうど0.5になり、
$\alpha < 0.5$では、αより低い値を取る
→ エラー発生確率が下がる

演習1. 誤り訂正符号の実装

- 1量子ビットのビット反転エラーと位相反転エラーに対応できるShorの符号について、実際に実装してみます。
- Shorの符号では9個の物理量子ビットを利用して、1個の論理量子ビットを作成します。



演習1．誤り訂正符号の実装

- Shorの符号では、9量子ビットのうち、2量子ビット以上でエラーが発生すると、エラーを訂正できないケースがあります。
- このため、9量子ビット中2量子ビット以上でエラーが発生する確率が、単一のビットのエラー率よりも高い場合、全体で誤り確率が上昇してしまいます。
- おおよその閾値は3%程度です。

α：1量子ビットあたりのエラー発生確率

エラーが発生する確率

$$1-\underbrace{\{(1-\alpha)^9+9(1-\alpha)^8\alpha\}}_{\text{エラーが訂正できる確率}}$$

$\alpha = 0.05$のとき、0.071
$\alpha = 0.03$のとき、0.028
$\alpha = 0.01$のとき、0.003

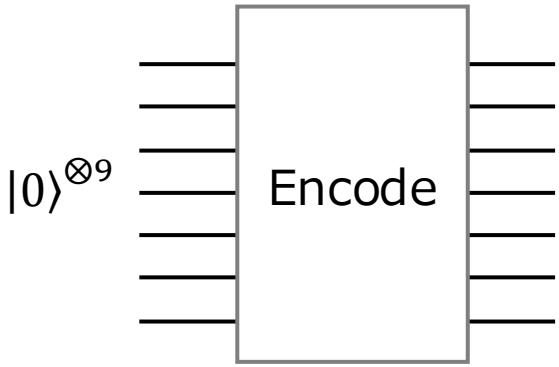
このため、符号化に多くの量子ビットを使うアルゴリズムでは、十分に誤り確率が低い必要があります。
誤り訂正を実際に活用していくためには、以下のようなことが重要になります。

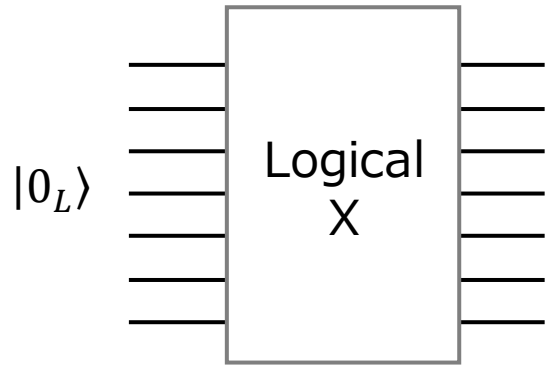
- エラーの発生率そのものを下げる
- 効率のよい符号化をする

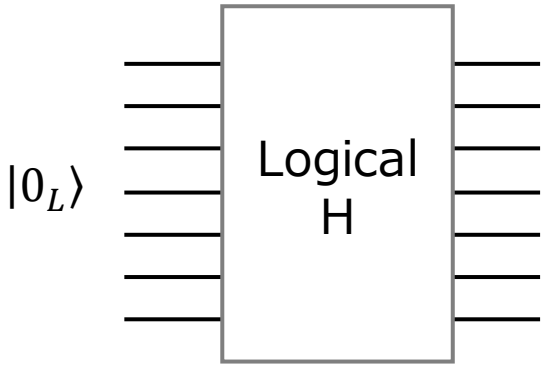
演習2. Shorの符号による論理ビット操作

- 現在の量子コンピュータにおいては、各種ゲート操作も大きなノイズの一因です。
- したがって、符号化された論理量子ビットに対して、各種ゲート操作に相当する操作を行い、このエラーを修正することが望ましいです。
- まずは、Shorの符号を用いて、実装を確認します。

＜論理ビット操作のイメージ図＞



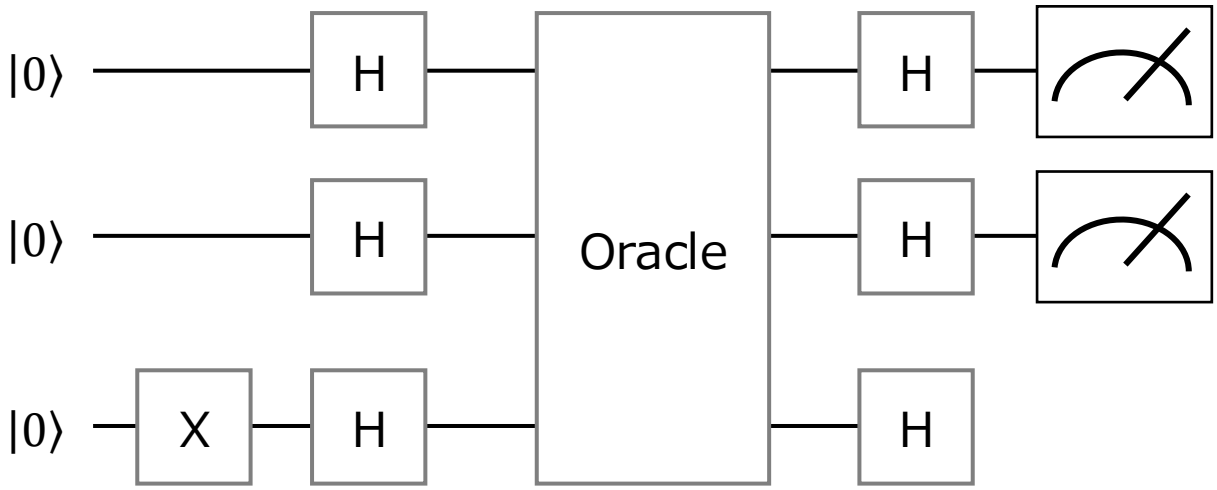




演習2. Shorの符号による論理ビット操作

- ここでは、ベルンシュタイン・ヴァジラニのアルゴリズムを題材として使います。
- ベルンシュタイン・ヴァジラニのアルゴリズムは、ある関数$f(x) = a_0 \cdot x_0 \oplus a_1 \cdot x_1$が与えられたときに、関数$f(x)$を一度だけ呼び出して、$a_0, a_1$の値を決定できるアルゴリズムです。
- ここでは、$a_0 = 1, a_1 = 0$のケースを扱います。

＜回路のイメージ図＞

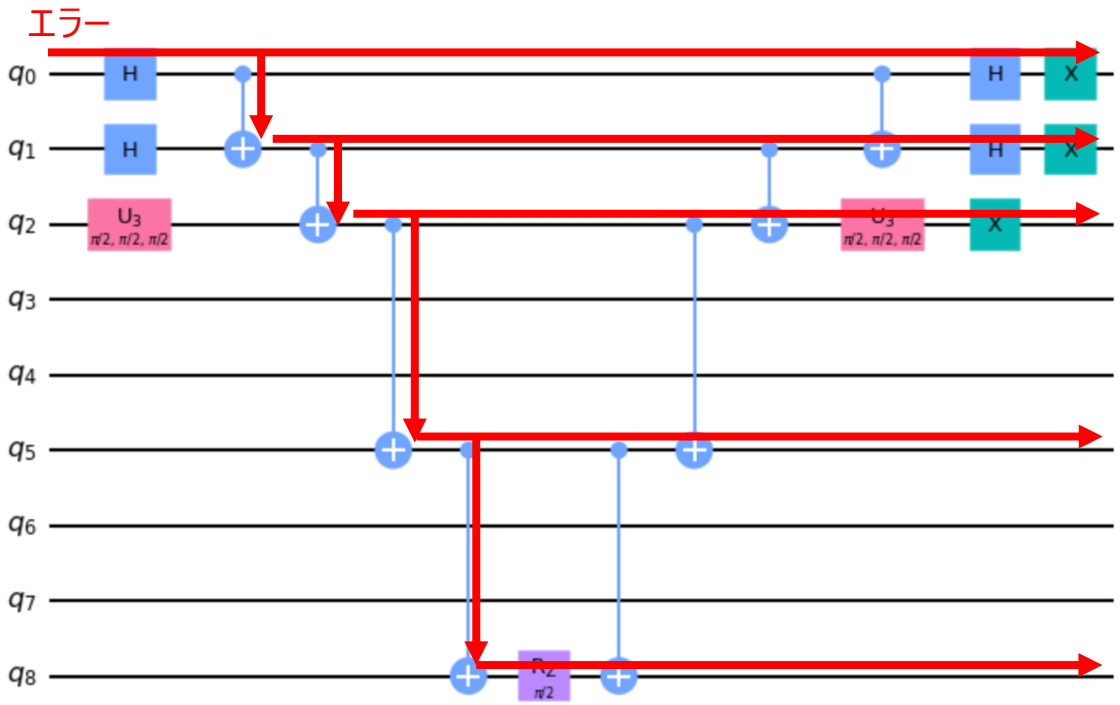


Oracle：関数fを回路に表現したもの

演習2. Shorの符号による論理ビット操作

- Shorの符号における論理ゲート操作の実装には課題が存在します。
- 論理Hゲートの実装では内部に2量子ビットゲートが存在するため、エラーが発生していると、他の量子ビットに伝播してしまう恐れがあります。
- このため、せっかく冗長化した意味が薄れてしまいます。

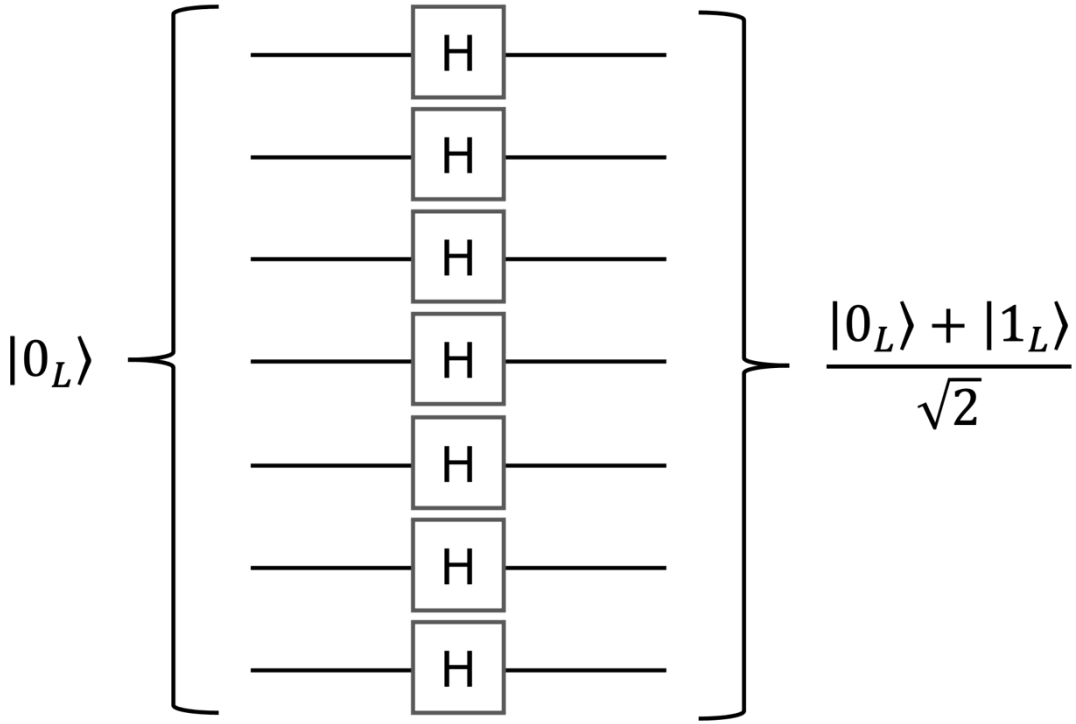
＜論理Hゲートでエラーが伝播していくイメージ＞



演習3. Steane符号による論理ビット操作

- ここで、Steane符号という別の誤り訂正符号を導入します。
- この符号では、論理Hゲートや、論理Xゲートを実装する際に、各量子ビットに並列に操作を加えることで、論理ビット操作を実現できます。
  - このような性質をTransversalityと呼びます。

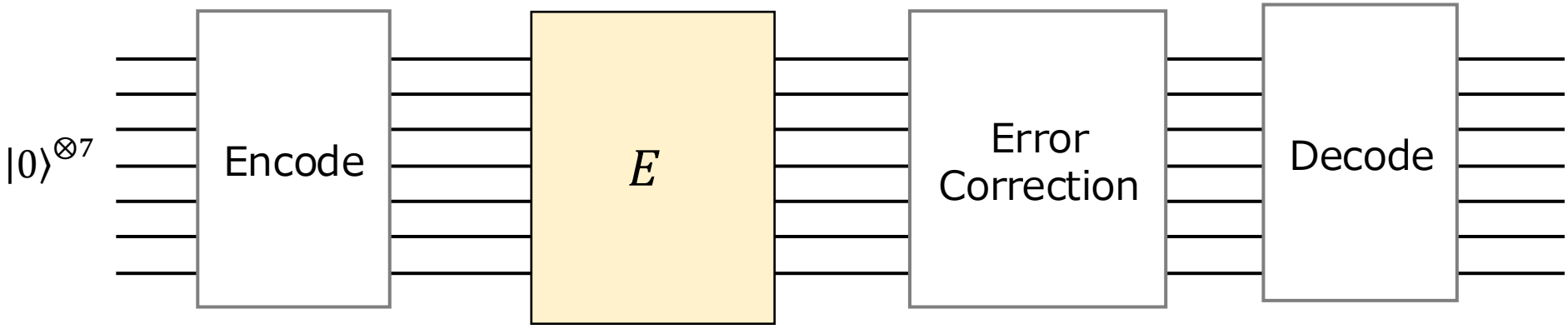
＜Steane符号における論理Hゲート＞



演習3. Steane符号による論理ビット操作

- まずはShorの符号の際と同様に、符号化とエラー訂正の流れだけ確認します。
- Shorの符号では、エラー訂正部と復号部が一体化していたものを紹介しましたが、ここでは分けて考えます。

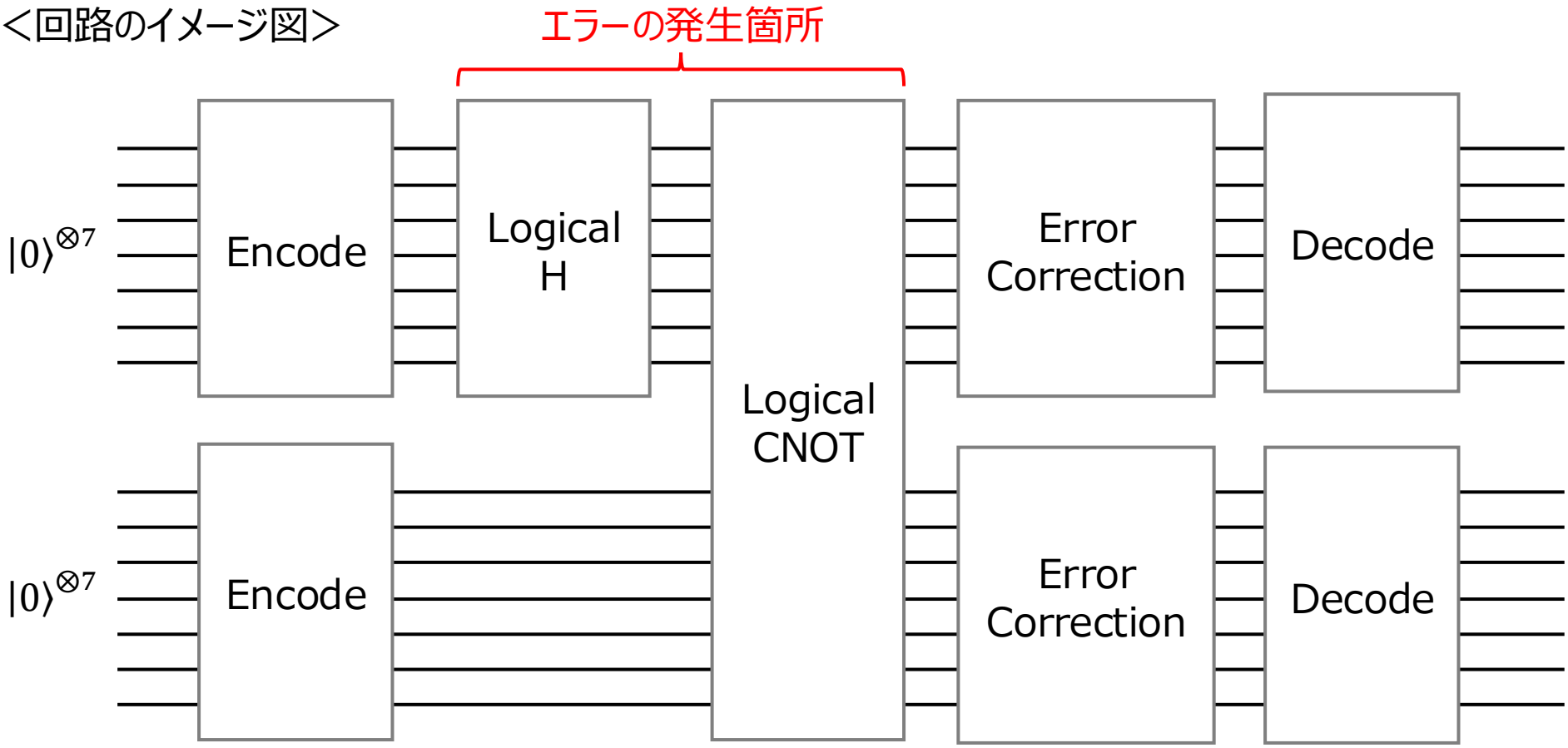
＜回路のイメージ図＞



演習3. Steane符号による論理ビット操作

- 今回の論理ビット操作の演習では、2論理量子ビットを用いて、ベル状態の作成を実施してみます。
- エラー訂正は、アダマールゲートとCNOTゲート適用後に1回だけ行います。
- 実用上は、長いアルゴリズムの場合は、何度も途中でエラー訂正を行いながら実行します。

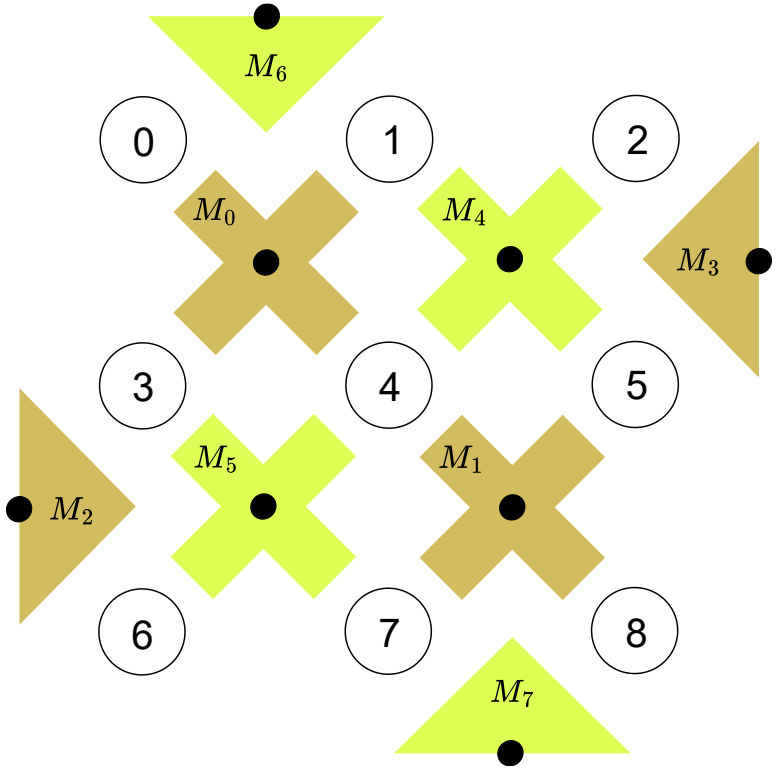
＜回路のイメージ図＞



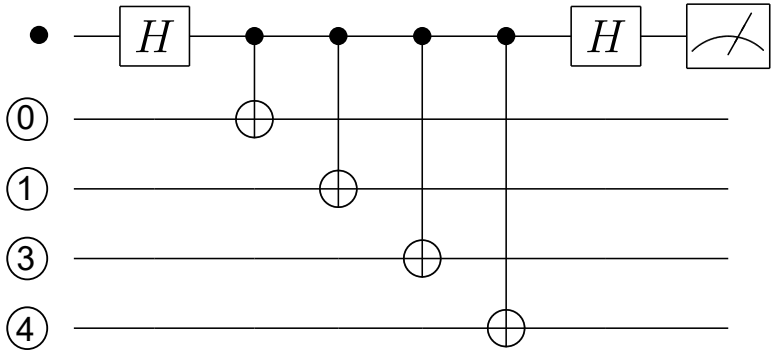
演習4. 表面符号の実装

- 続いて、表面符号と呼ばれる誤り訂正符号を扱います。
- ここでは平面上に 9 個の量子ビットが並んだ [9, 1, 3] 表面符号を実装し、論理ビット操作を行います。
  - M_0～M_3は X スタビライザー、M_4～M_7 は Z スタビライザー、黒丸はアンシラを表します。

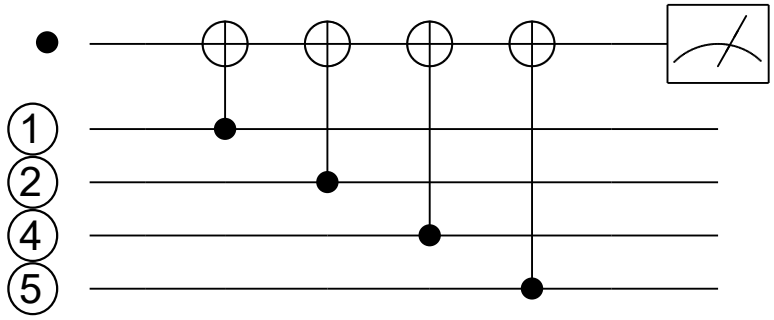
[9, 1, 3] 表面符号



エンコード（Xスタビライザー部）



エンコード（Zスタビライザー部）



（参考）[9, 1, 3] 表面符号の基底とエラーシンドローム

$$|0\rangle_L = \frac{1}{4}(|011101110\rangle + |000000011\rangle + |101011000\rangle + |000110101\rangle + |101101110\rangle + |000000000\rangle + |110000011\rangle + |101101101\rangle$$
$$+|011011000\rangle + |011101101\rangle + |101011011\rangle + |011011011\rangle + |000110110\rangle + |110110110\rangle + |110110101\rangle + |110000000\rangle)$$

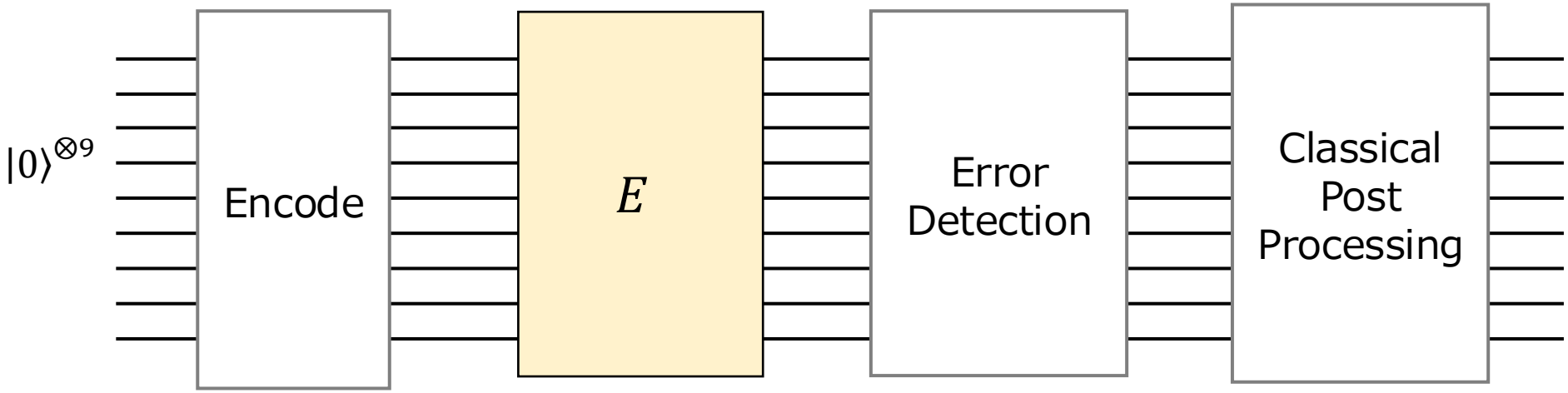
$$|1\rangle_L = \frac{1}{4}(|111001010\rangle + |001001010\rangle + |001001001\rangle + |111001001\rangle + |010100111\rangle + |100010001\rangle + |100100111\rangle + |010010010\rangle$$
$$+|100100100\rangle + |010010001\rangle + |100010010\rangle + |001111111\rangle + |111111100\rangle + |111111111\rangle + |001111100\rangle + |010100100\rangle)$$

ビット番号	0	1	2	3	4	5	6	7	8
M_0	X	X		X	X				
M_1					X	X		X	X
M_2				X			X		
M_3			X			X			
M_4		Z	Z		Z	Z			
M_5				Z	Z		Z	Z	
M_6	Z	Z							
M_7								Z	Z

演習4. 表面符号の実装

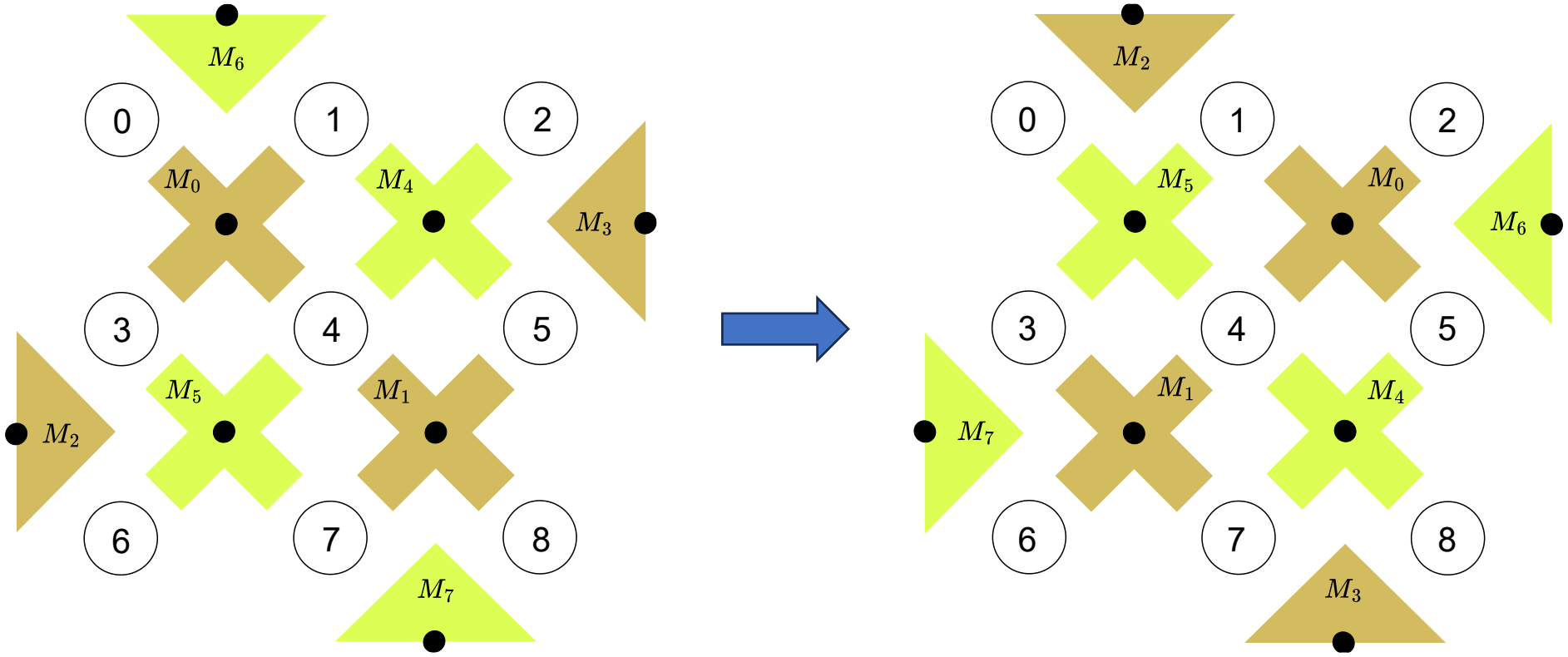
- Steane符号と同様に、符号化とエラー訂正の流れを確認していきます。
- [9, 1, 3] 表面符号では 1 個の量子ビットのエラーを検出・訂正できますが、ここでは検出までを行います。
- デコード部分は量子回路に実装せずに、測定結果を古典的に解析することで代替します。

＜回路のイメージ図＞



演習4. 表面符号による論理ビット操作

- 今回の演習では（誤りが発生しない状況で） 1 論理ビットの操作のみを行います。
- 論理 H ゲートは transversal な操作で実現できますが、図のように X と Z が交換されて、スタビライザーも変化します。



演習4. 表面符号による論理ビット操作

- スタビライザーが変われば符号も変わってしまいますが、図のように 90 度回転させることで元の符号に戻ります。
- しかし、データビットの順序が変わっているため、元に戻す必要があります。
- 演習では物理的にデータビットを動かすのではなく、ビット番号を変更することで実装します。