Qiskit Textbook Chap.5

Yuma Nakamura

Qiskit Advocate

Last updated: 2020-10-02

Quantum Tokyo

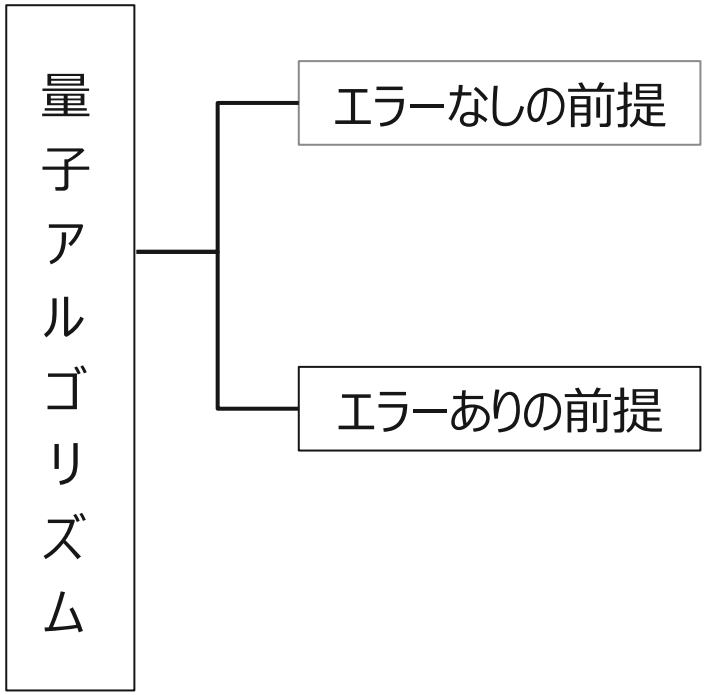
Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。

量子アルゴリズム

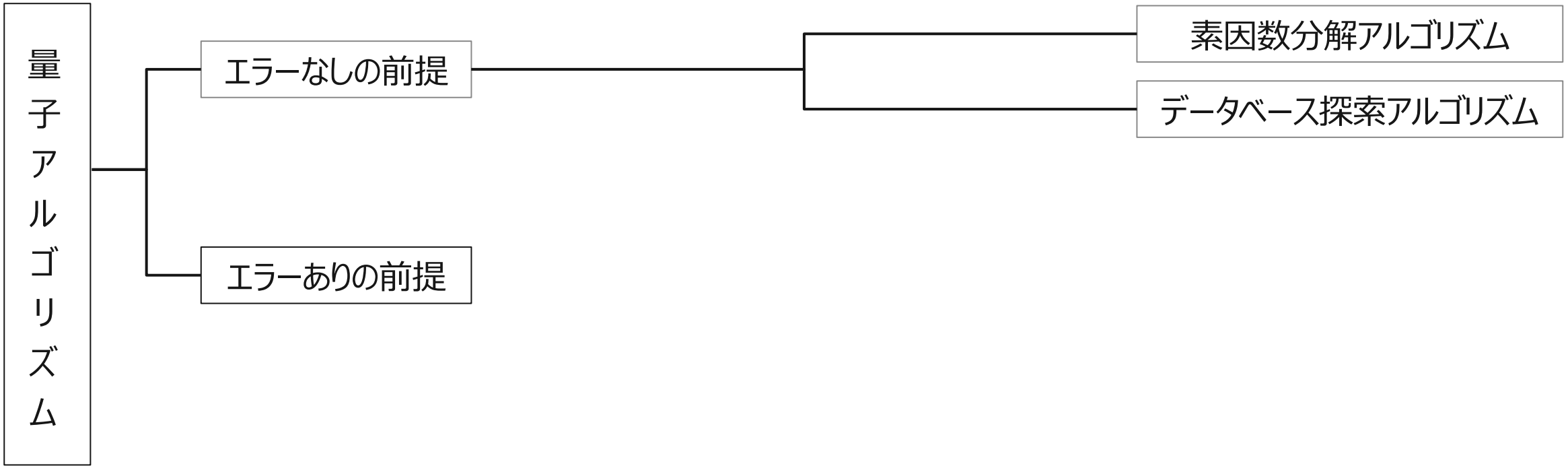
Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。



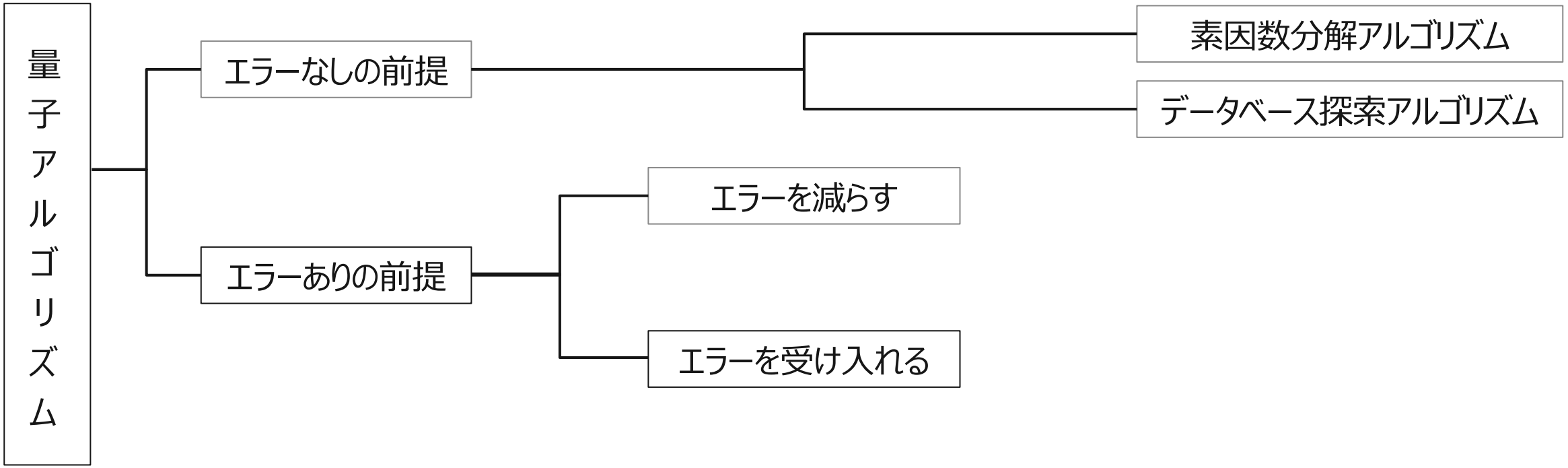
Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。



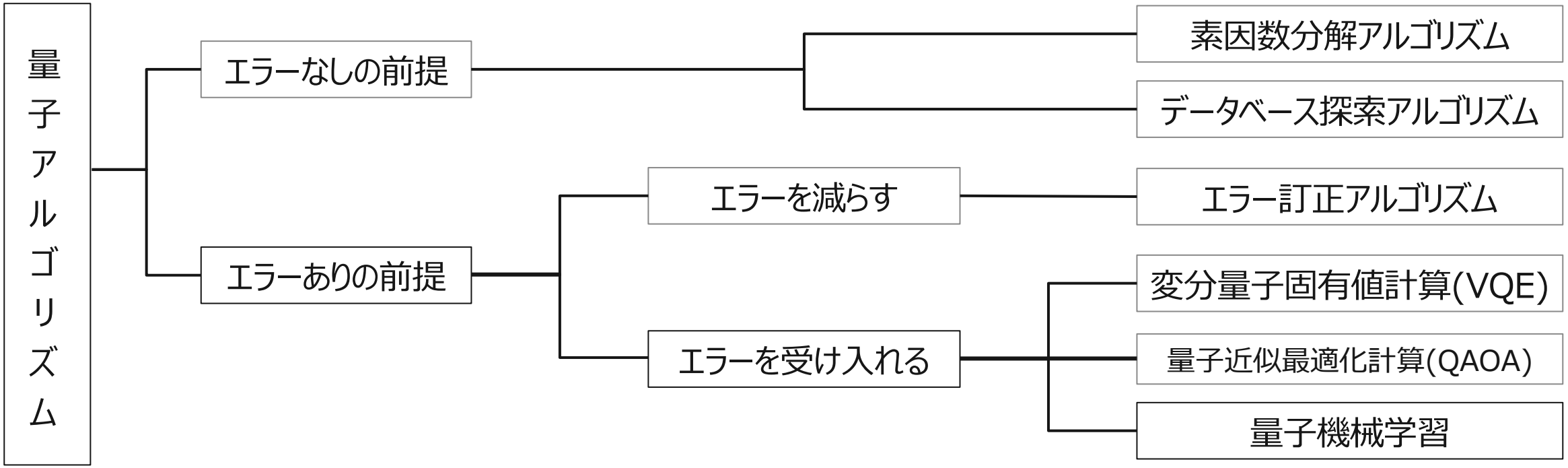
Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。



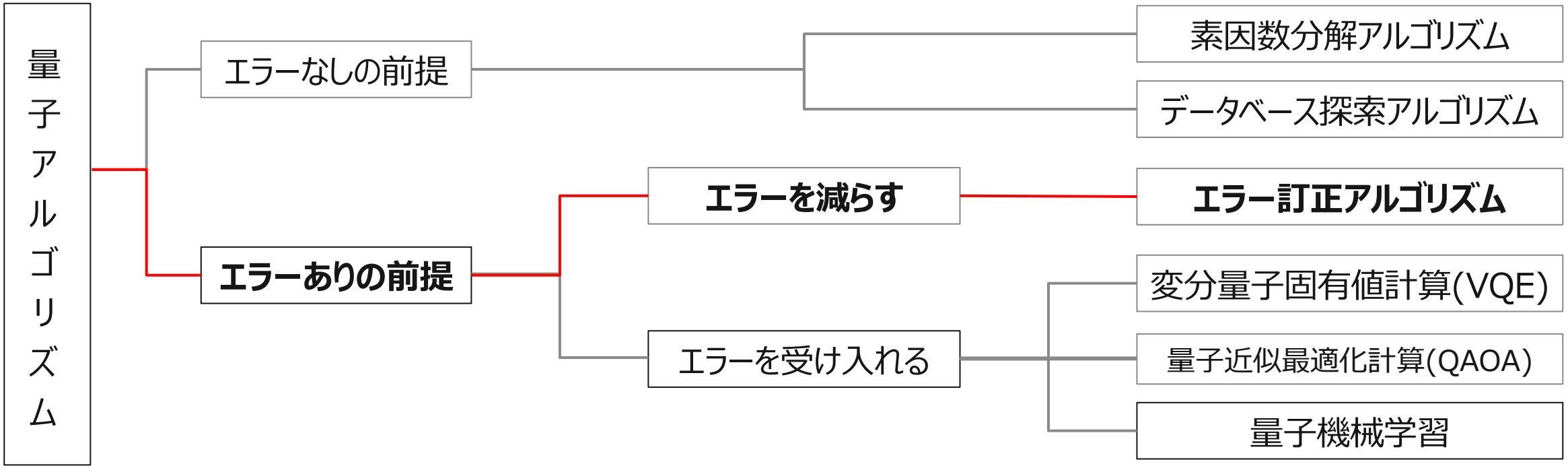
Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。



Qiskit Textbook 5章の立ち位置

ここまでのQiskit Textbookでのアルゴリズムはエラーなしの前提か、エラーを受け入れたNISQデバイスでの議論でした。
今回はエラーを減らすためのアルゴリズムの解説です。



アジェンダ

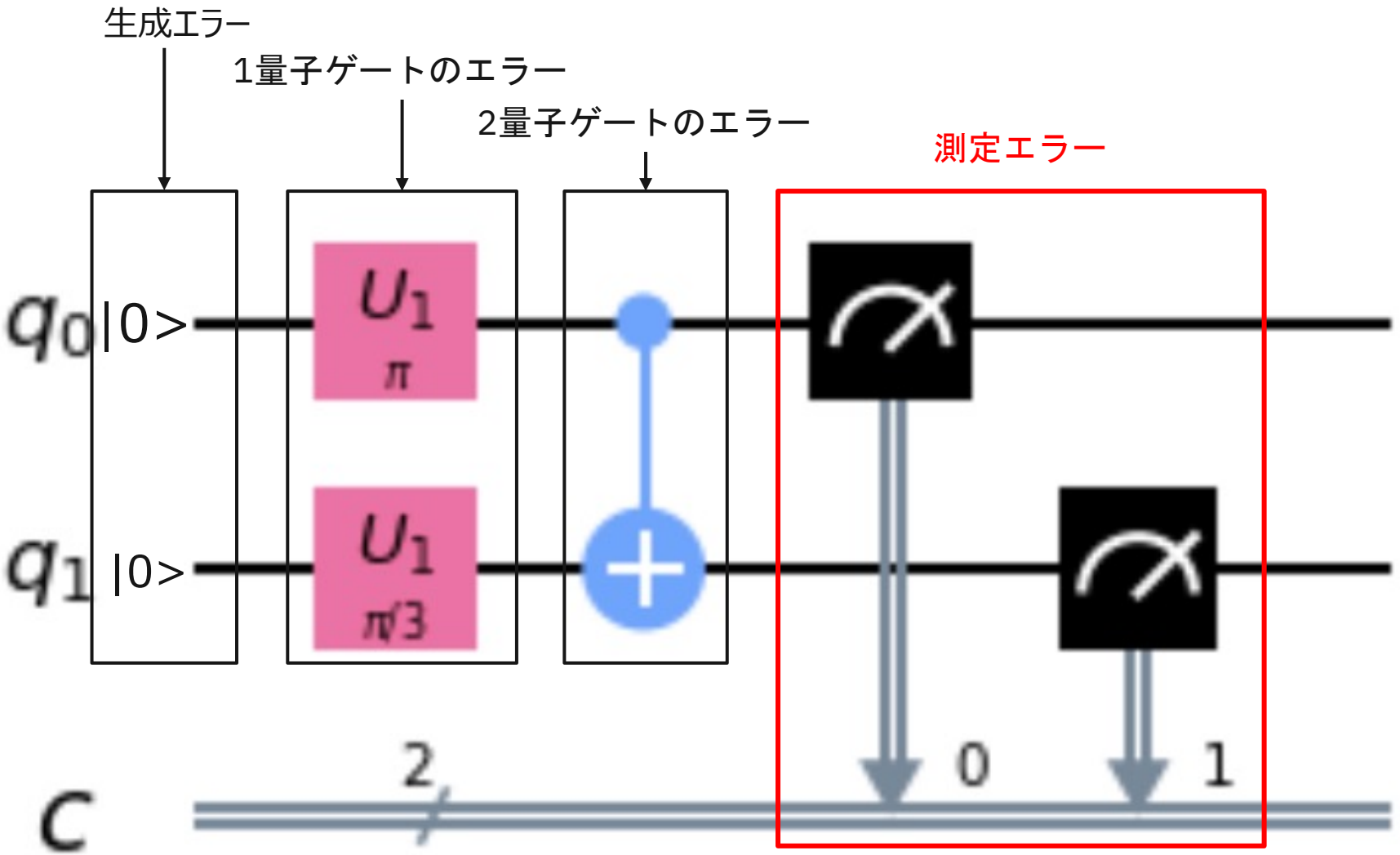
1. 測定エラー軽減

2. 反復符号によるエラー訂正

3. ランダム化ベンチマーキング

測定エラー軽減

量子回路のエラーの発生源は複数がありますが、今回は測定エラーの軽減に焦点をおきます

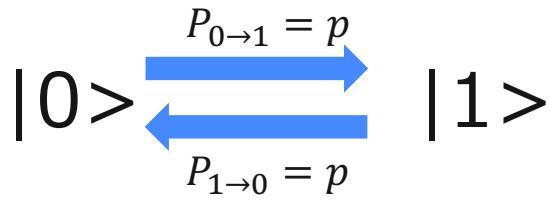


測定エラー軽減

確率$p=1\%$で各ビットがフリップするモデルから議論を始めます。

1量子ビットの場合:

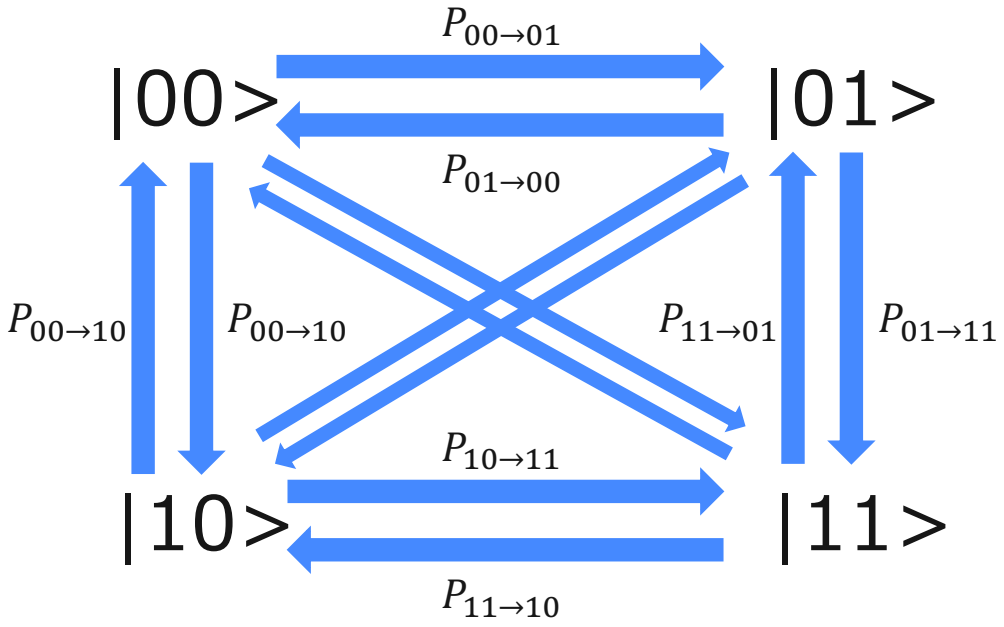
$$P_{x\to\bar{x}} = p$$



2量子ビットの場合:

$$P_{xy\to\bar{x}y} = P_{xy\to x\bar{y}} = p$$

$$P_{xy\to\overline{xy}} = p^2$$

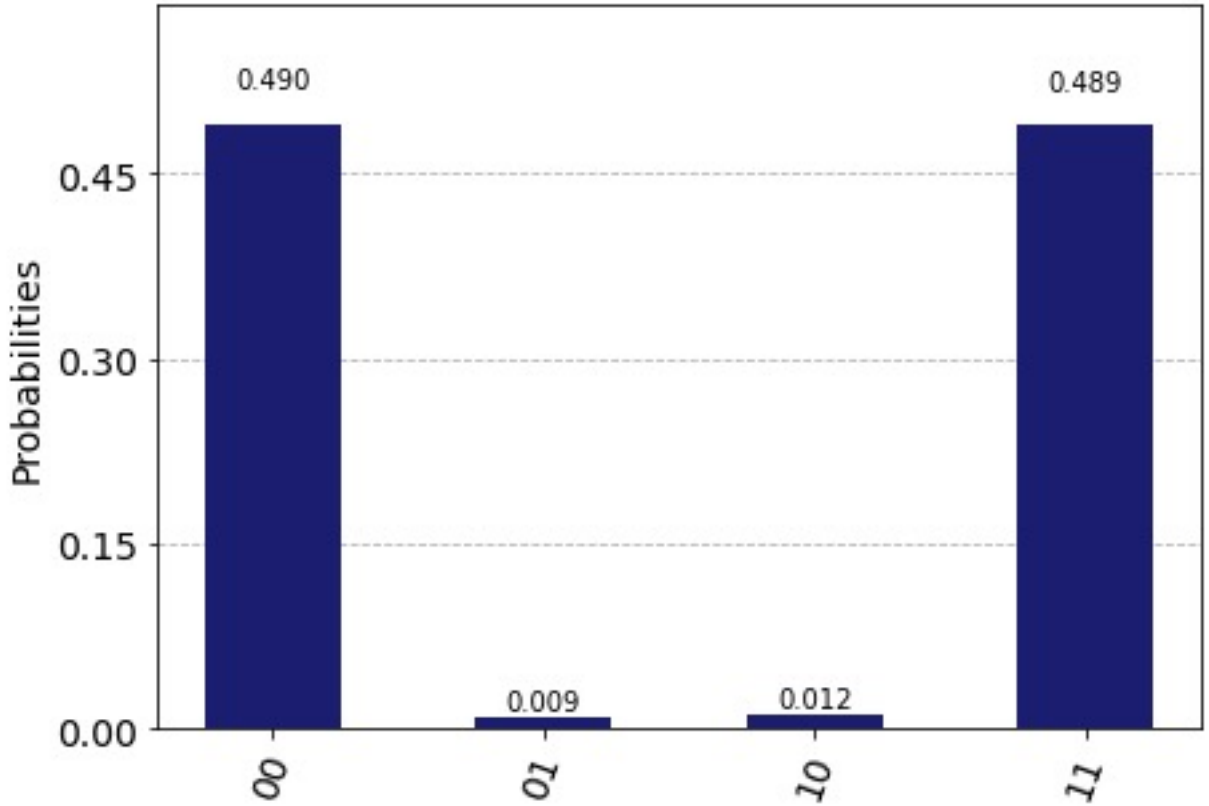


測定エラー軽減

次のような観測結果が得られたとき、観測前の状態は何だったでしょうか？

Shots=10,000

Counts = {'00': 4901, '01': 95, '10': 116, '11': 4888}



(1) |00>

(2) |11>

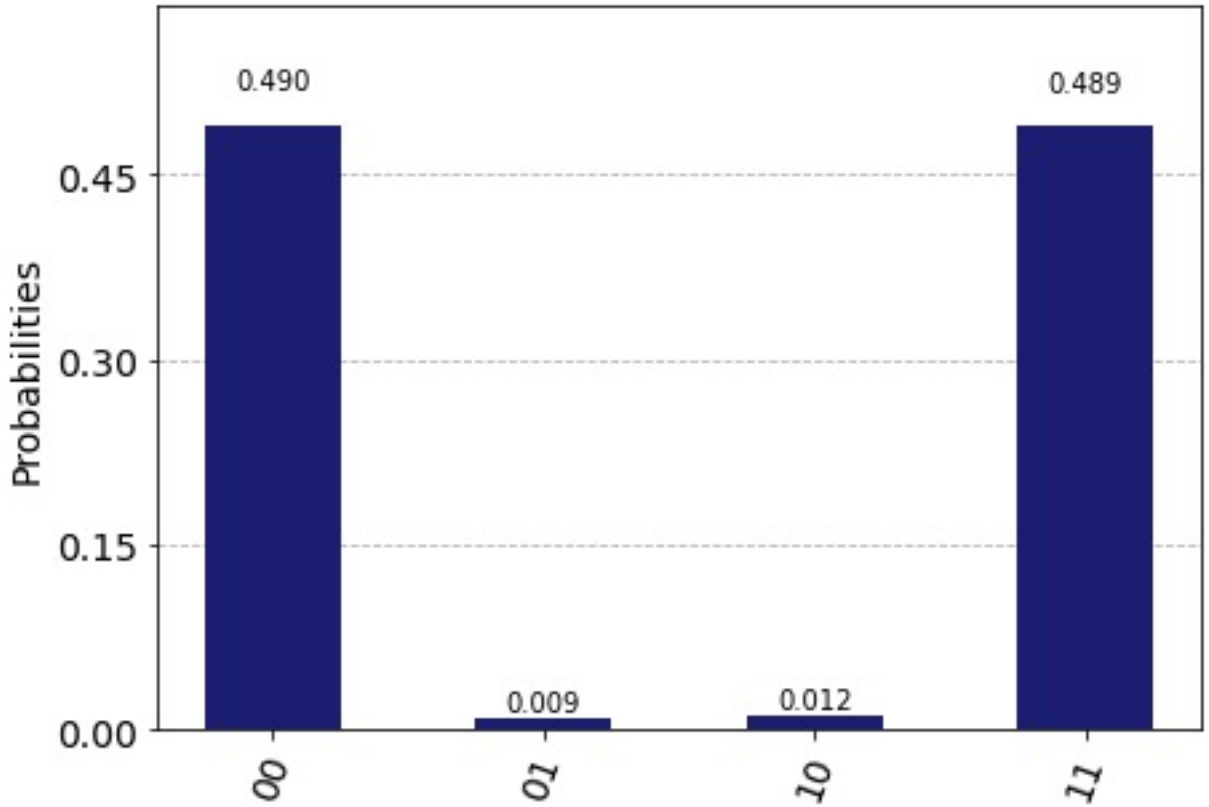
(3) $\frac{1}{\sqrt{2}}$(|00> + |11>)

測定エラー軽減

次のような観測結果が得られたとき、観測前の状態は何だったでしょうか？

Shots=10,000

Counts = {'00': 4901, '01': 95, '10': 116, '11': 4888}



(1) |00>

(2) |11>

(3) $\frac{1}{\sqrt{2}}$(|00> + |11>)

測定エラー軽減

では、定量的に先ほどの判断を行うにはどうしたらいいでしょうか？

Observed Counts

$\{'00': C_{00}^{obs}, '01': C_{01}^{obs}, '10': C_{10}^{obs}, '11': C_{11}^{obs}\}$

True Counts

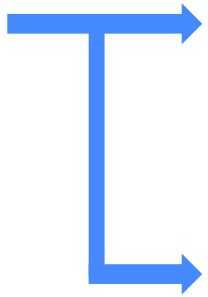
$\{'00': C_{00}^{idial}, '01': C_{01}^{idial}, '10': C_{10}^{idial}, '11': C_{11}^{idial}\}$

測定エラー軽減

では、定量的に先ほどの判断を行うにはどうしたらいいでしょうか？

Observed Counts

$\{'00': C_{00}^{obs}, '01': C_{01}^{obs}, '10': C_{10}^{obs}, '11': C_{11}^{obs}\}$



True Counts

$\{'00': C_{00}^{idial}, '01': C_{01}^{idial}, '10': C_{10}^{idial}, '11': C_{11}^{idial}\}$

Predicted Counts

$\{'00': C_{00}^{pred}, '01': C_{01}^{pred}, '10': C_{10}^{pred}, '11': C_{11}^{pred}\}$

観測エラー軽減ではこれを目標にします

測定エラー軽減

では、定量的に先ほどの判断を行うにはどうしたらいいでしょうか？

Observed Counts

$\{'00': C_{00}^{obs}, '01': C_{01}^{obs}, '10': C_{10}^{obs}, '11': C_{11}^{obs}\}$

True Counts

$\{'00': C_{00}^{idial}, '01': C_{01}^{idial}, '10': C_{10}^{idial}, '11': C_{11}^{idial}\}$

Predicted Counts

$\{'00': C_{00}^{pred}, '01': C_{01}^{pred}, '10': C_{10}^{pred}, '11': C_{11}^{pred}\}$

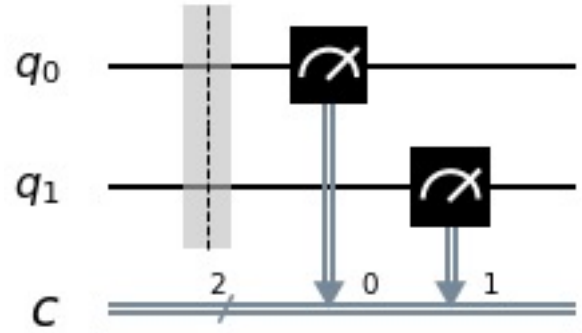
観測エラー軽減ではこれを目標にします

つまり $\begin{pmatrix} C_{00}^{obs} \\ C_{01}^{obs} \\ C_{10}^{obs} \\ C_{11}^{obs} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} C_{00}^{pred} \\ C_{01}^{pred} \\ C_{10}^{pred} \\ C_{11}^{pred} \end{pmatrix}$ とするfを決める

測定エラー軽減

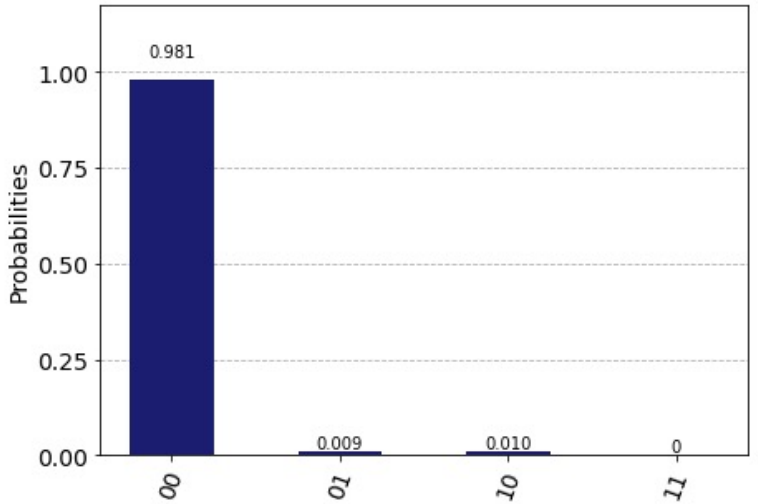
観測エラー軽減をするため、計算基底がどのように観測されるかを事前に調べます。

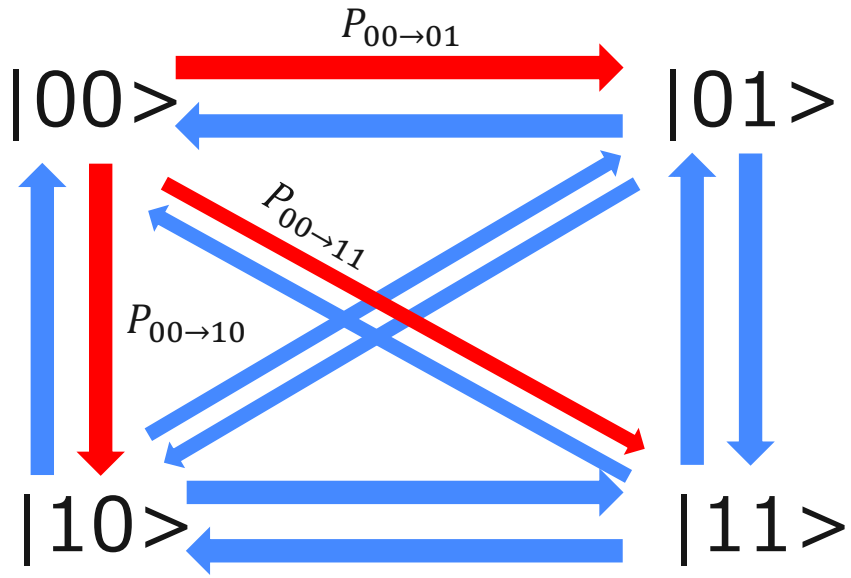
Step 1 :　|00>を観測



Shots=10,000

Counts = {'00': 9808, '01': 95, '10': 96, '11': 1}



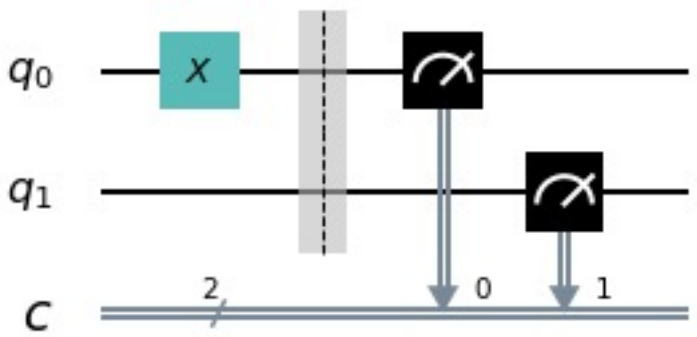


	|00>	|01>	|10>	|11>
00の出現率	**0.9808**			
01の出現率	**0.0095**			
10の出現率	**0.0096**			
11の出現率	**0.0001**			

測定エラー軽減

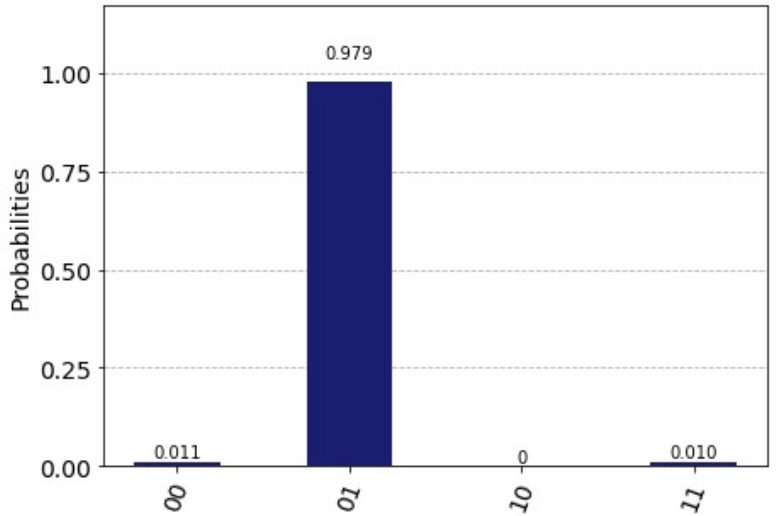
観測エラー軽減をするため、計算基底がどのように観測されるかを事前に調べます。

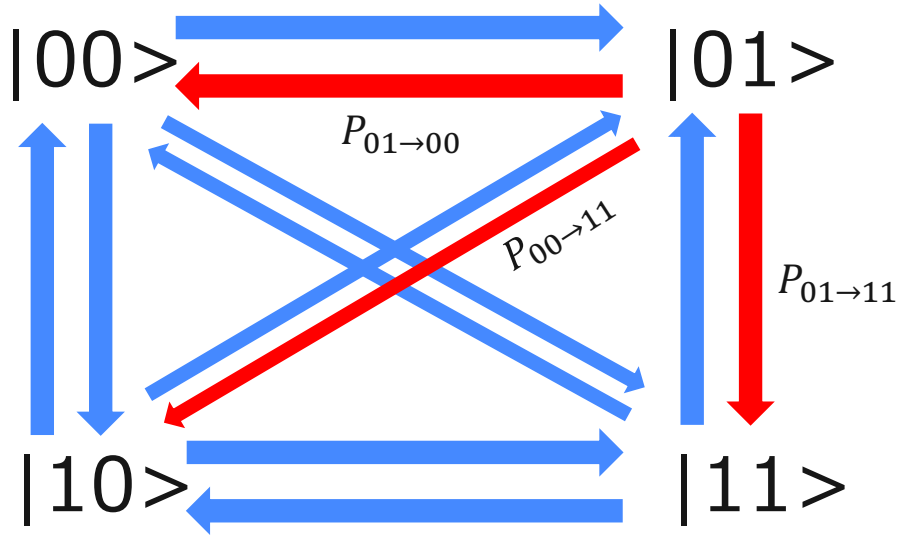
Step 2 :　|01>を観測



Shots=10,000

Counts = {'00': 107, '01': 9788, '10': 2, '11': 103}



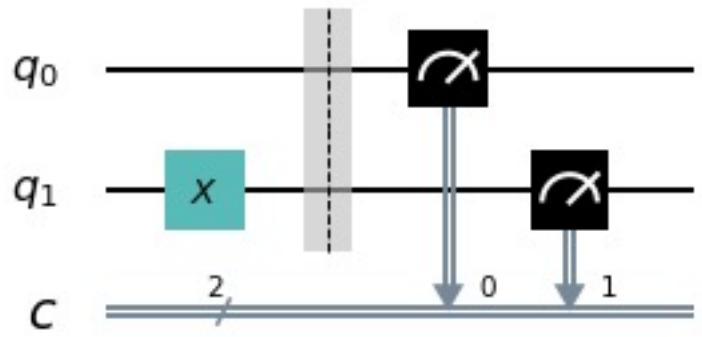


	|00>	|01>	|10>	|11>
00の出現率	0.9808	**0.0107**		
01の出現率	0.0095	**0.9788**		
10の出現率	0.0096	**0.0002**		
11の出現率	0.0001	**0.0103**		

測定エラー軽減

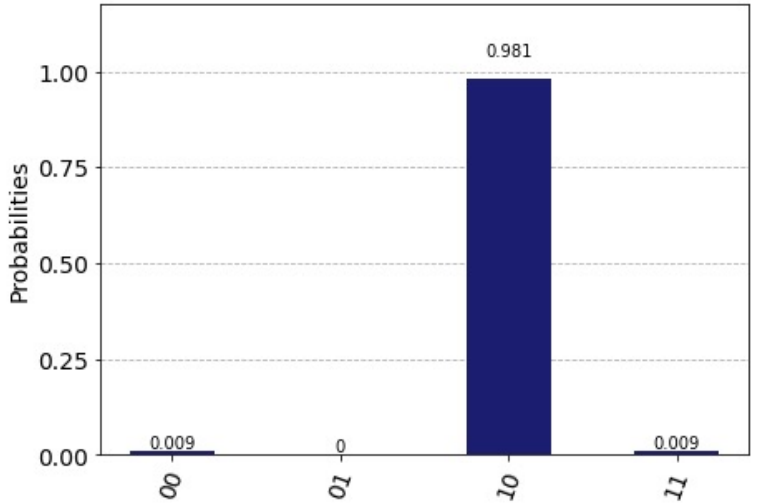
観測エラー軽減をするため、計算基底がどのように観測されるかを事前に調べます。

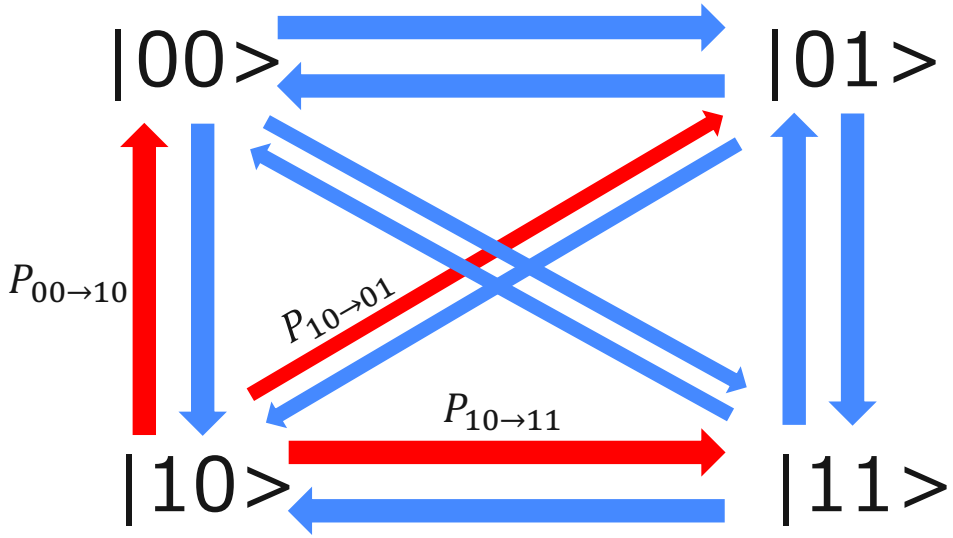
Step 3 :　|10>を観測



Shots=10,000

Counts = {'00': 95, '01': 1, '10': 9814, '11': 90}



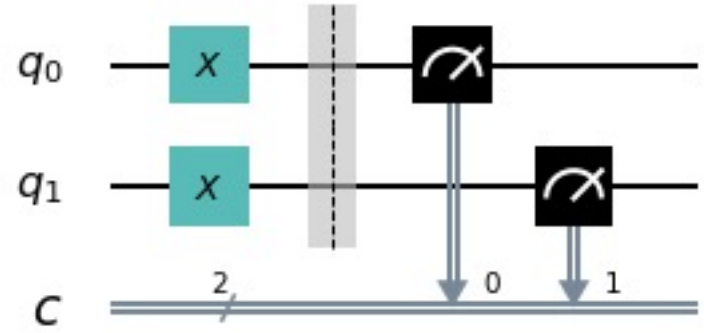


	|00>	|01>	|10>	|11>
00の出現率	0.9808	0.0107	**0.0095**	
01の出現率	0.0095	0.9788	**0.0001**	
10の出現率	0.0096	0.0002	**0.9814**	
11の出現率	0.0001	0.0103	**0.0090**	

測定エラー軽減

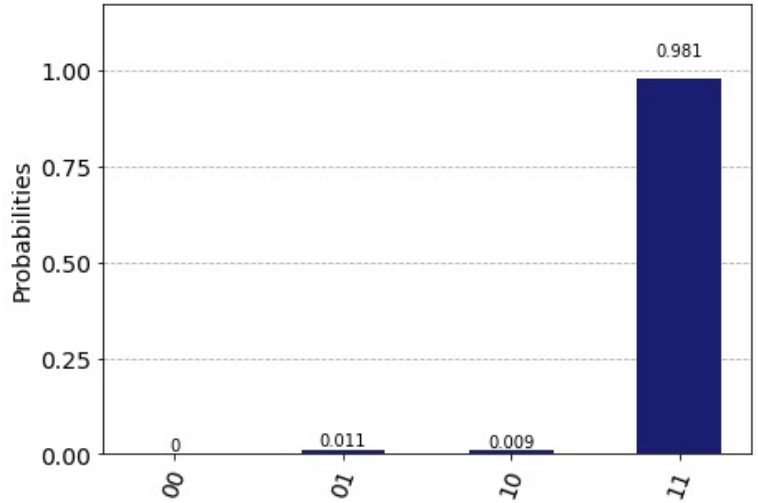
観測エラー軽減をするため、計算基底がどのように観測されるかを事前に調べます。

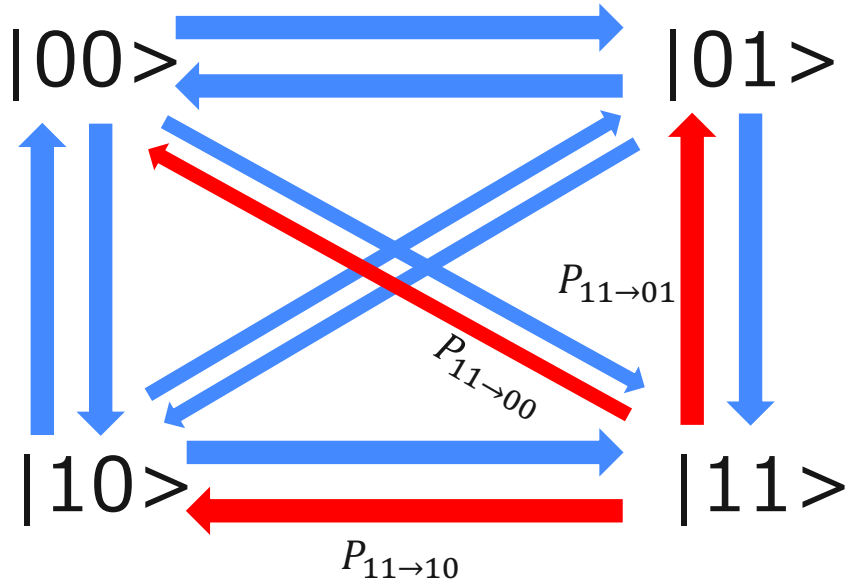
Step 4：　|11>を観測



Shots=10,000

Counts = {'00': 1, '01': 107, '10': 87, '11': 9805}





	|00>	|01>	|10>	|11>
00の出現率	0.9808	0.0107	0.0095	**0.0001**
01の出現率	0.0095	0.9788	0.0001	**0.0107**
10の出現率	0.0096	0.0002	0.9814	**0.0087**
11の出現率	0.0001	0.0103	0.0090	**0.9805**

測定エラー軽減

得られた表を行列Mとすると、ノイズレスの真の観測値から測定ノイズのある場合の観測値を再現できます。

	|00>	|01>	|10>	|11>
00の出現率	0.9808	0.0107	0.0095	0.0001
01の出現率	0.0095	0.9788	0.0001	0.0107
10の出現率	0.0096	0.0002	0.9814	0.0087
11の出現率	0.0001	0.0103	0.0090	0.9805

$$M = \begin{pmatrix} 0.9808 & 0.0107 & 0.0095 & 0.0001 \\ 0.0095 & 0.9788 & 0.0001 & 0.0107 \\ 0.0096 & 0.0002 & 0.9814 & 0.0087 \\ 0.0001 & 0.0103 & 0.0090 & 0.9805 \end{pmatrix}$$

このとき $M\ C_{ideal} = C_{noisy}$

このMを校正行列と呼びます

測定エラー軽減

実際にQasm_simulationで観測した結果と比べると、非常に近い結果となってます。

	|00>	|01>	|10>	|11>
00の出現率	0.9808	0.0107	0.0095	0.0001
01の出現率	0.0095	0.9788	0.0001	0.0107
10の出現率	0.0096	0.0002	0.9814	0.0087
11の出現率	0.0001	0.0103	0.0090	0.9805

$$M = \begin{pmatrix} 0.9808 & 0.0107 & 0.0095 & 0.0001 \\ 0.0095 & 0.9788 & 0.0001 & 0.0107 \\ 0.0096 & 0.0002 & 0.9814 & 0.0087 \\ 0.0001 & 0.0103 & 0.0090 & 0.9805 \end{pmatrix}$$

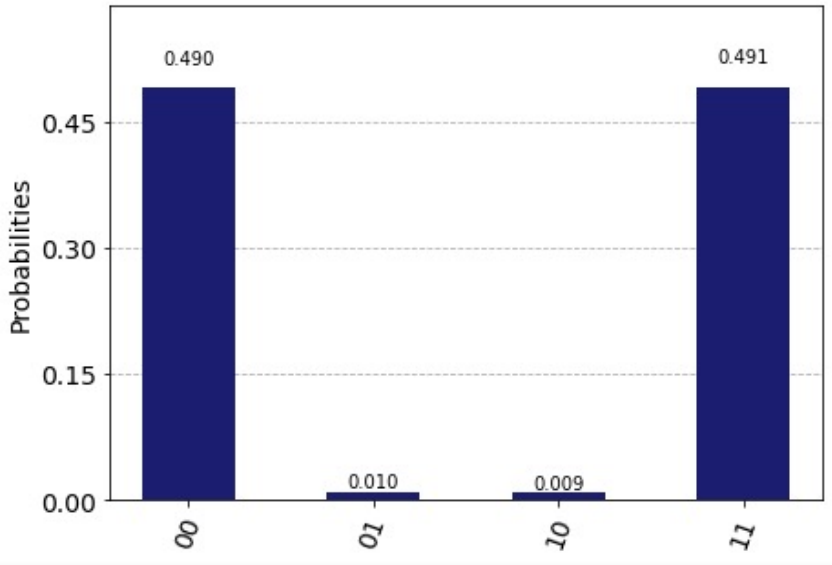
$\frac{1}{\sqrt{2}}$(|00> + |11>) の観測値C_{ideal}からC_{noisy}を計算してみます。

$$C_{ideal} = \begin{pmatrix} 5000 \\ 0 \\ 0 \\ 5000 \end{pmatrix} \quad より$$

$$C_{noisy} = MC_{ideal} = M \begin{pmatrix} 5000 \\ 0 \\ 0 \\ 5000 \end{pmatrix} = \begin{pmatrix} 4904.5 \\ 101 \\ 91.5 \\ 4903 \end{pmatrix}$$

ノイズモデルでのQasm_simulation

C_{noisy}^{obs}={'00': 4904, '01': 100, '10': 85, '11': 4911}



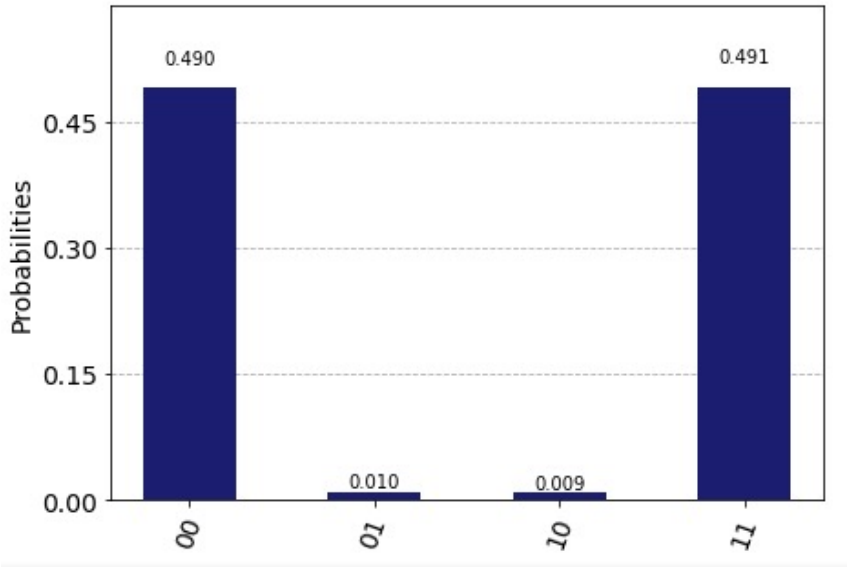
測定エラー軽減

較正行列Mの逆行列を使うと、観測ノイズありの観測結果から観測ノイズなしの結果をよく再現できます。

$$C_{ideal} = M^{-1} C_{noisy}$$

先ほどの観測結果に適用してみます。

{'00': 4904, '01': 100, '10': 85, '11': 4911}



$$C_{noisy}^{obs} = \begin{pmatrix} 4904 \\ 100 \\ 85 \\ 4911 \end{pmatrix}$$ より

$$C_{ideal} = M^{-1} \begin{pmatrix} 4904 \\ 100 \\ 85 \\ 4911 \end{pmatrix} = \begin{pmatrix} 4999.6 \\ -1.1 \\ -6.7 \\ 5008.2 \end{pmatrix} \approx \begin{pmatrix} 5000 \\ -1 \\ -7 \\ 5008 \end{pmatrix}$$

測定エラー軽減(ここからはPythonコードと合わせて説明します)

次にノイズを10倍(p=10%)にしてエラー軽減を試します。

まずビットフリップ型のノイズモデルを定義します

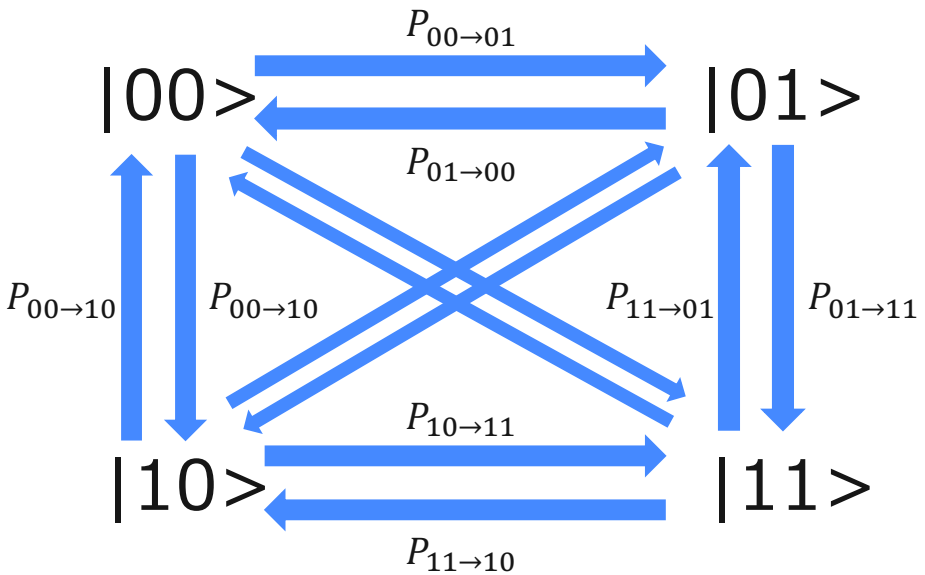
```python
from qiskit import *
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors import pauli_error, depolarizing_error

# 確率pでビットフリップするノイズモデルを作成
def get_noise(p):
    # Xゲートでビットフリップを定義
    error_meas = pauli_error([('X',p), ('I', 1 - p)])
    noise_model = NoiseModel()
    # 観測エラーを適用
    noise_model.add_all_qubit_quantum_error(error_meas, "measure")
    return noise_model
```

2量子ビットの場合:

$$P_{xy\to\bar{x}y} = P_{xy\to x\bar{y}} = p$$

$$P_{xy\to\overline{xy}} = p^2$$



測定エラー軽減(ここからはPythonコードと合わせて説明します)

次にノイズを10倍(p=10%)にしてエラー軽減を試します。

次に校正行列Mを作成します。

```
from qiskit.ignis.mitigation.measurement import (complete_meas_cal,CompleteMeasFitter)

#行列Mの計算用のため |00>、|01>、|10>、|11>を観測する回路を作成
meas_calibs, state_labels = complete_meas_cal(qr=qiskit.QuantumRegister(2))
# backendでqasm_simulatorを選択
backend = qiskit.Aer.get_backend('qasm_simulator')
# noise_modelを指定して、|00>、|01>、|10>、|11>の回路それぞれを実行
cal_results = qiskit.execute(meas_calibs, backend=backend,
                             shots=1000,noise_model=get_noise(0.1)).result()
# 実行結果から校正行列Mを取得
meas_fitter = CompleteMeasFitter(cal_results, state_labels)
calibulation_matrix = meas_fitter.cal_matrix)
```

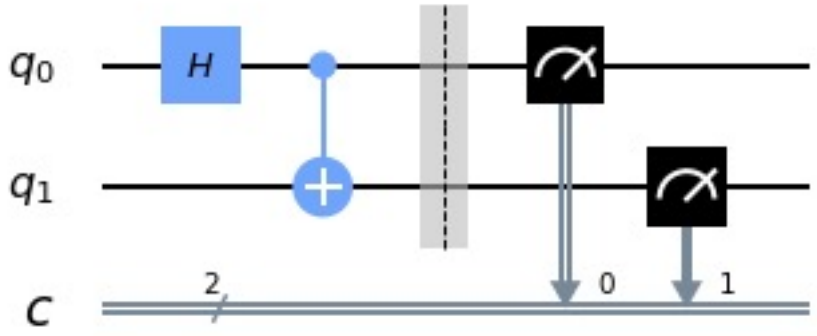
$$\mathrm{M} = \begin{bmatrix} 0.823 & 0.083 & 0.083 & 0.007 \\ 0.089 & 0.819 & 0.01 & 0.099 \\ 0.082 & 0.008 & 0.816 & 0.093 \\ 0.006 & 0.09 & 0.091 & 0.801 \end{bmatrix}$$

測定エラー軽減(ここからはPythonコードと合わせて説明します)

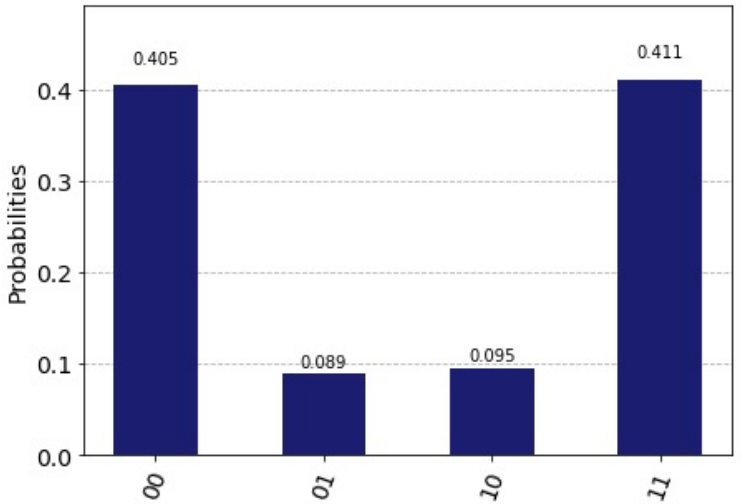
次にノイズを10倍(p=10%)にしてエラー軽減を試します。

$\frac{1}{\sqrt{2}}$(|00> + |11>)に対してC_{noisy}を計算します

```
from qiskit.visualization import *
qc = QuantumCircuit(2, 2)
qc.h(0)
qc.cx(0,1)
qc.barrier()
qc.measure([0, 1], [0, 1])
results = execute(qc,Aer.get_backend('qasm_simulator'),noise_model=get_noise(0.1),shots=10000).result()
noisy_counts = results.get_counts()
qc.draw(output="mpl")
print({x: noisy_counts[x] for x in sorted(noisy_counts)})
plot_histogram(noisy_counts, color='midnightblue',)
```



{'00': 4049, '01': 889, '10': 953, '11': 4109}



$$C_{noisy}^{obs} = \begin{pmatrix} 4049 \\ 889 \\ 953 \\ 4109 \end{pmatrix}$$

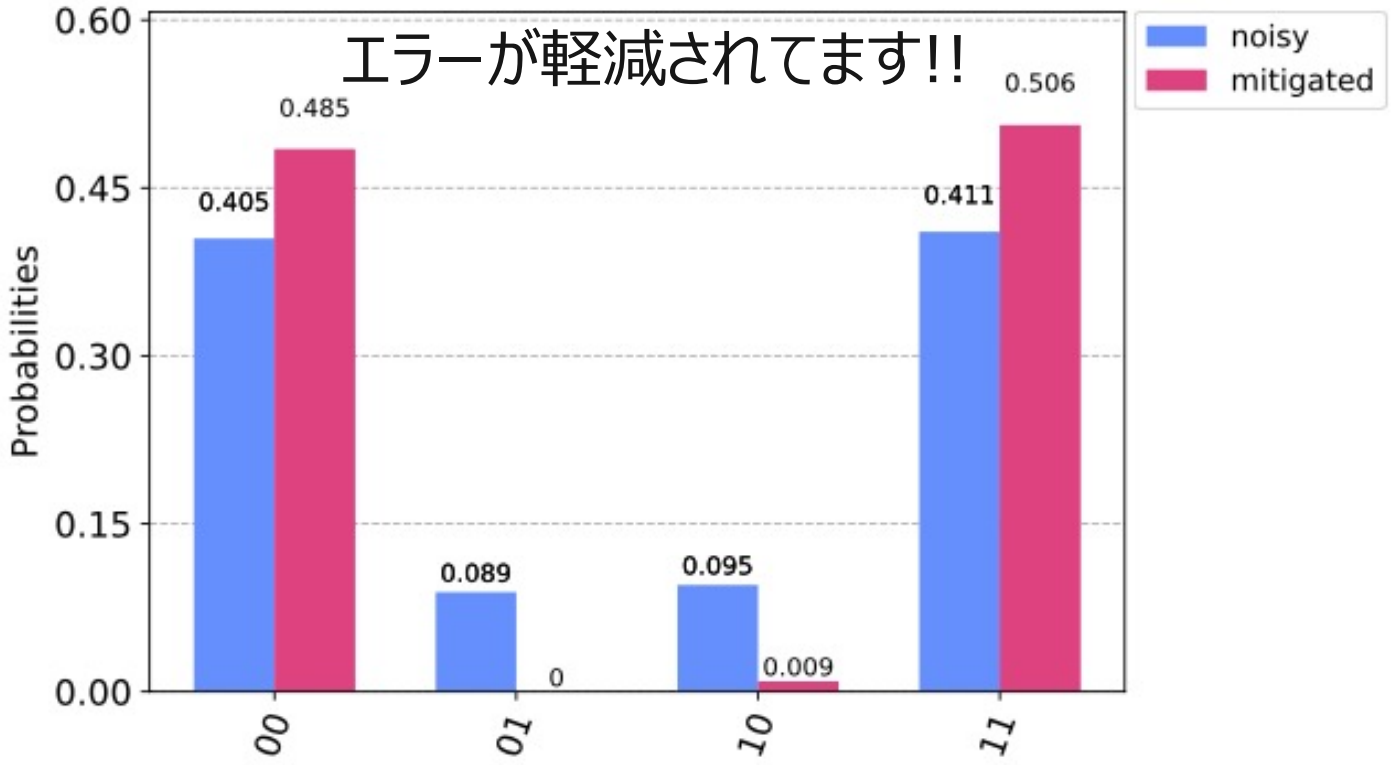
測定エラー軽減(ここからはPythonコードと合わせて説明します)

次にノイズを10倍(p=10%)にしてエラー軽減を試します。

C_{noisy}と校正行列MからC_{pred}計算します

```
# オブジェクトをフィルターする(詳しくは不明)
meas_filter = meas_fitter.filter
# 測定結果に校正行列を作用させる
mitigated_results = meas_filter.apply(results)
mitigated_counts = mitigated_results.get_counts(0)
print(mitigated_counts)
```

$$C_{pred} = M^{-1}C_{noisy}^{obs} = M^{-1}\begin{pmatrix}4049\\889\\953\\4109\end{pmatrix} = \begin{pmatrix}4873\\-60\\99\\5089\end{pmatrix}$$



1. 測定エラー軽減(まとめ)

・計算基底(|00>,…|11>)の観測結果から校正行列Mを作成
・Mの逆行列で観測エラーを軽減

2. 反復符号によるエラー訂正

3. ランダム化ベンチマーキング

1. 測定エラー軽減

2. 反復符号によるエラー訂正

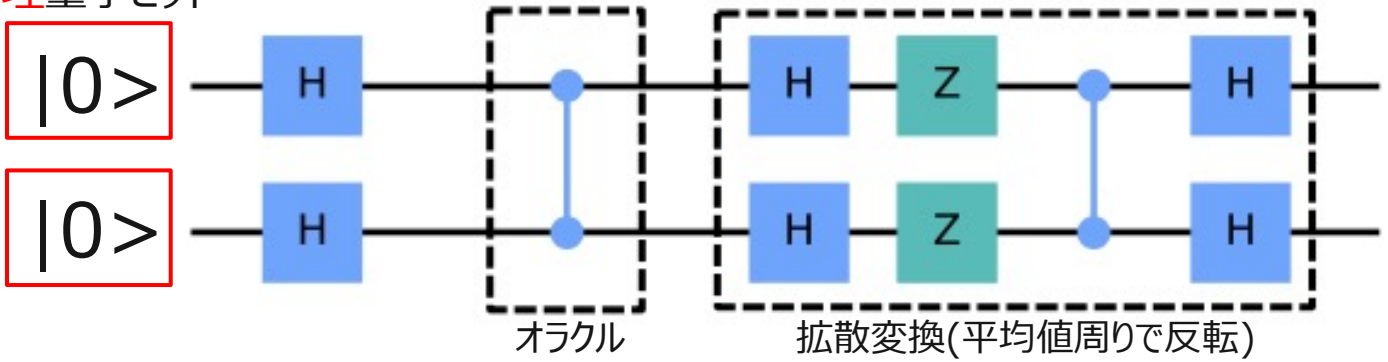
3. ランダム化ベンチマーキング

反復符号によるエラー訂正

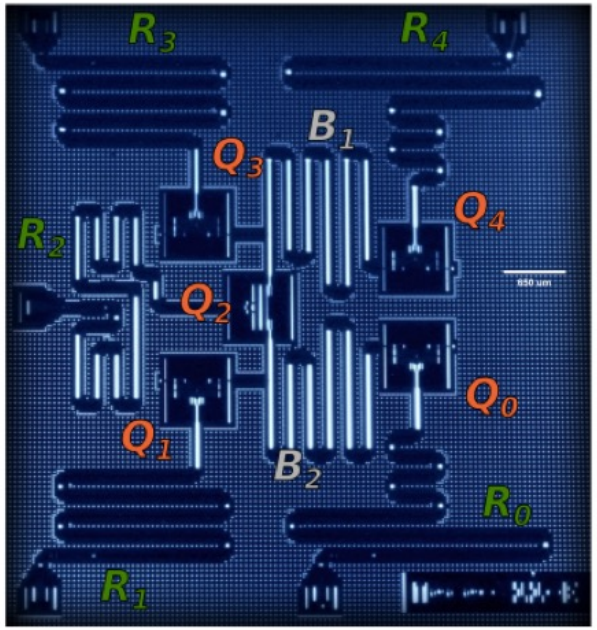
量子アルゴリズムを学ぶ時には論理量子ビットを前提に使っていましたが、
今回は論理ビットと物理ビットの関係を考えます。

論理量子ビット

Groverアルゴリズム



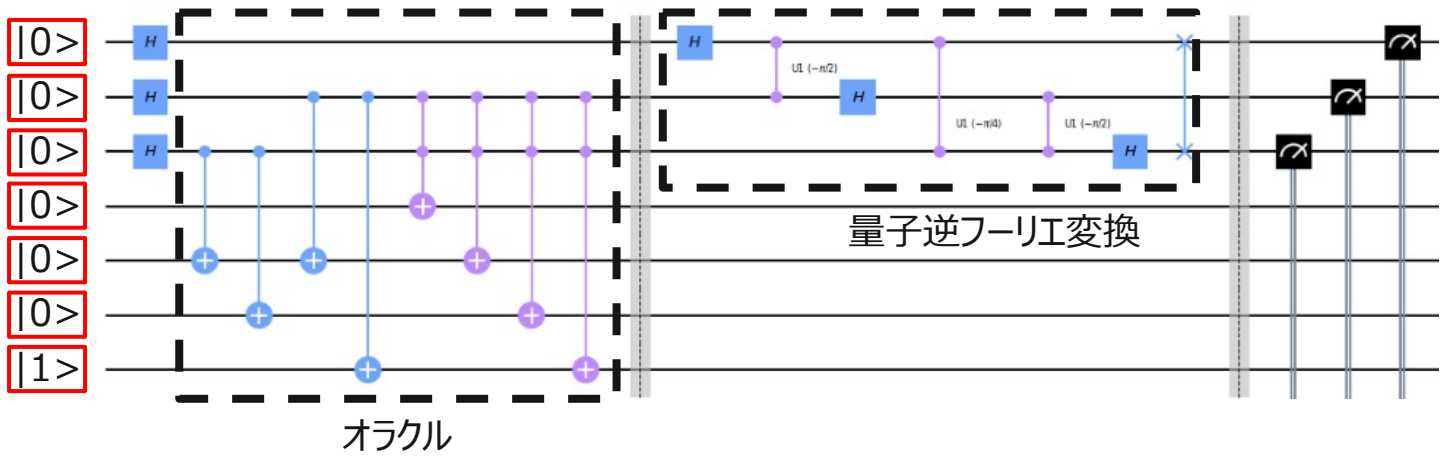
物理量子ビット



5量子ビットプロセッサー ibmqx4

論理量子ビット

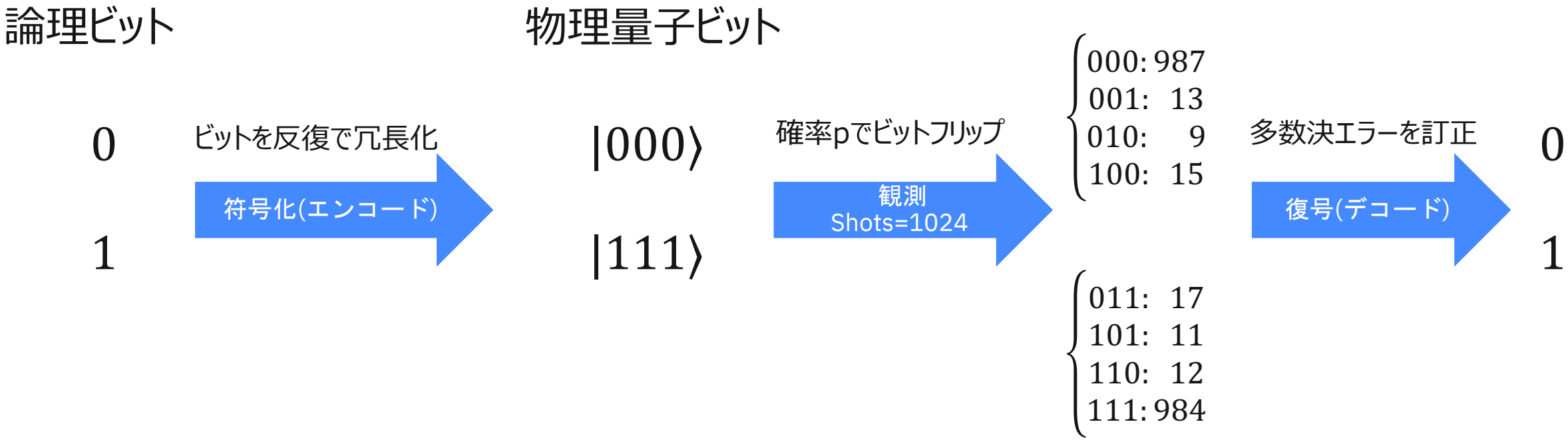
Shorアルゴリズム



反復符号によるエラー訂正

反復符号を使って、多数物理ビットが1つの論理ビットに対応させます。多数決でエラー訂正をします。

反復回数n=3で、エラーモデルは確率p=1%でビットフリップが起きることを仮定



反復符号によるエラー訂正

しかしエラーの発生確率が高いとエラー訂正が機能しなくなります。

反復回数n=3, 観測数shot=1024、ビットフリップエラー確率をp=1%, 10%, 50%で比較してみる

物理量子ビット	観測値(p=1%)	観測値(p=10%)	観測値(p=50%)
$\|000\rangle$	$\begin{cases} 000: 987 \\ 001: 13 \\ 010: 9 \\ 100: 15 \end{cases}$		
$\|111\rangle$	$\begin{cases} 011: 17 \\ 101: 11 \\ 110: 12 \\ 111: 984 \end{cases}$		

全て訂正可能

反復符号によるエラー訂正

しかしエラーの発生確率が高いとエラー訂正が機能しなくなります。

反復回数n=3, 観測数shot=1024、ビットフリップエラー確率をp=1%, 10%, 50%で比較してみる

物理量子ビット	観測値(p=1%)	観測値(p=10%)	観測値(p=50%)
$\|000\rangle$	000: 987 001: 13 010: 9 100: 15	000: 743 001: 80 010: 78 100: 96 011: 11 101: 11 110: 7 111: 2	
$\|111\rangle$	011: 17 101: 11 110: 12 111: 984	000: 1 001: 11 010: 9 100: 10 011: 87 101: 68 110: 85 111: 753	

エラーを正しく訂正できない
(p^2の確率で発生)

エラーを認識できない
(p^3の確率で発生)

エラーを正しく訂正できない

全て訂正可能

一部訂正不可(反復回数nを増やせば可能)

反復符号によるエラー訂正

しかしエラーの発生確率が高いとエラー訂正が機能しなくなります。

反復回数n=3, 観測数shot=1024、ビットフリップエラー確率をp=1%, 10%, 50%で比較してみる

物理量子ビット	観測値(p=1%)	観測値(p=10%)	観測値(p=50%)
$\|000\rangle$	000: 987 001: 13 010: 9 100: 15	000: 743 001: 80 010: 78 100: 96 011: 11 101: 11 110: 7 111: 2	000: 122 001: 139 010: 124 100: 113 011: 131 101: 118 110: 133 111: 144
$\|111\rangle$	011: 17 101: 11 110: 12 111: 984	000: 1 001: 11 010: 9 100: 10 011: 87 101: 68 110: 85 111: 753	000: 132 001: 155 010: 132 100: 112 011: 112 101: 139 110: 135 111: 107
	全て訂正可能	一部訂正不可(反復回数nを増やせば可能)	訂正不可

エラーを正しく訂正できない
(p^2の確率で発生)

エラーを認識できない
(p^3の確率で発生)

元の状態の原型なし

反復符号によるエラー訂正

量子状態を維持するためにもエラー訂正は必須で、観測し続けエラーを追跡する必要があります。

・物理ビットの量子状態作成から測定までに大幅な時間差がある場合が存在する(量子HDDなど)

・ゲート操作や測定以外でも、量子ビットがアイドル状態(待機状態)で環境ノイズ等の影響を受ける

・アイドル状態を避けるために情報を取得し続け、必要に応じて適宜エラー訂正する必要がある

・量子では情報取得とは測定なので、連続して測定する必要がある

反復符号によるエラー訂正

重ね合せ状態の直接の測定は破壊的で、前述の方法(多数決)が使えません。代わりにシンドローム測定をします。

状態 $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle) \rightarrow \frac{1}{\sqrt{2}}(|000\rangle+|111\rangle)$ の保存を考える

保存するには測定をし続ける必要があるが、実質的にはZ測定に対応する

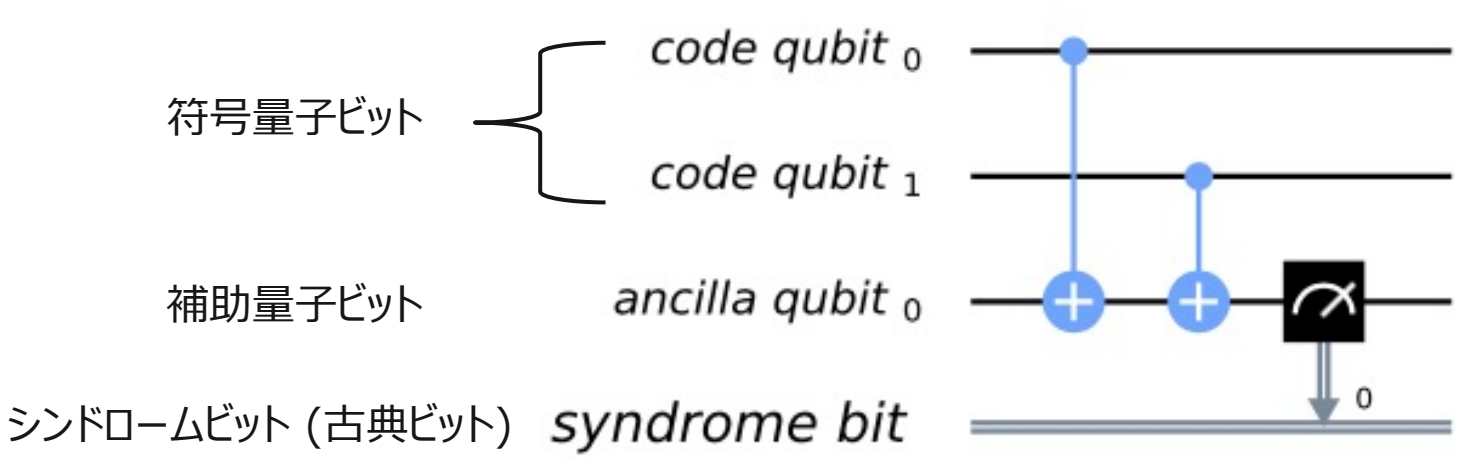
量子エラー訂正の難しさ
- 状態$|+\rangle$をZ測定すると状態が破壊されてしまう
- 量子状態はコピーできない

$|000\rangle$や$|111\rangle$を直接測定できない → シンドローム測定なるものを利用

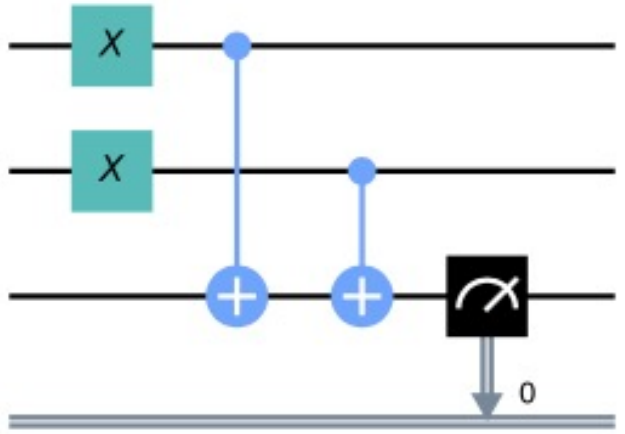
反復符号によるエラー訂正

シンドローム測定とは補助量子ビットを使って状態を壊さず間接的に観測する方法です。

論理ビット0を|00⟩で符号化する場合



論理ビット1を|11⟩で符号化する場合



シンドロームビットの性質:

符号量子ビットの状態は変わらない

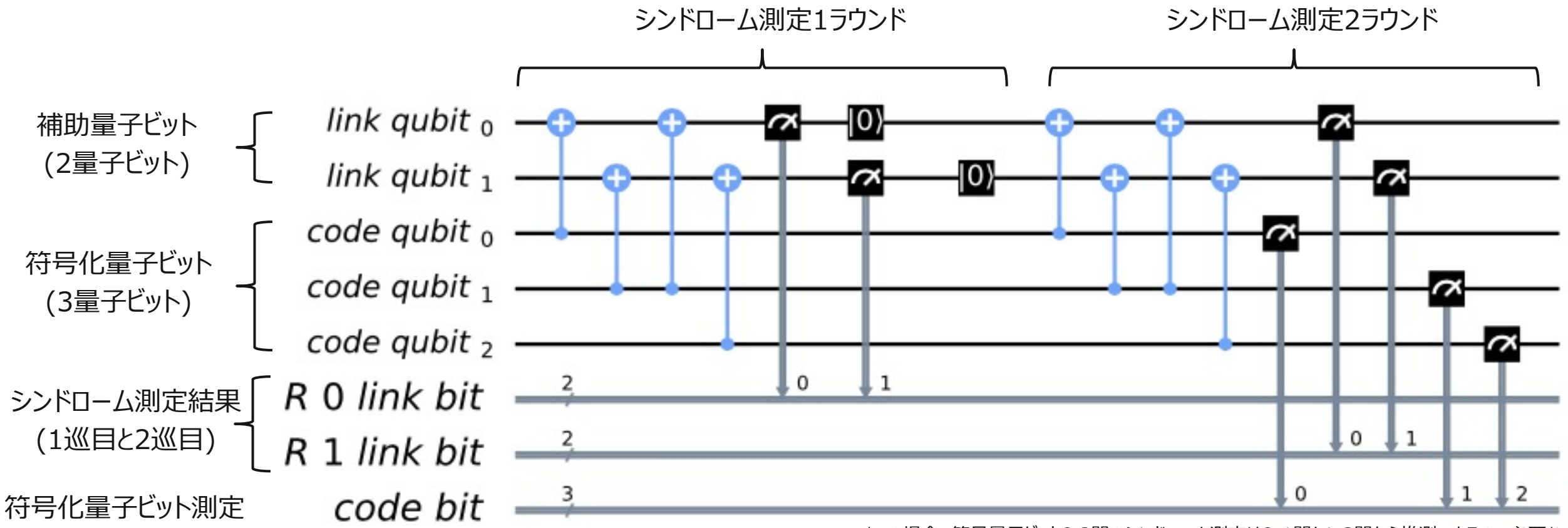
エラーがない場合*は 0 になる

エラーが存在する場合 1 になる(符号量子ビットのエラー or 補助量子ビットの観測エラー)

*繰り返し数n=2と同数のエラーは認識できない

反復符号によるエラー訂正

シンドローム測定では反復回数nに対しとは補助量子ビットn-1個*を使い、何巡(Round)測定するか選べます。



*この場合、符号量子ビット0,2間のシンドローム測定は0-1間と1-2間から推測できるので必要ない

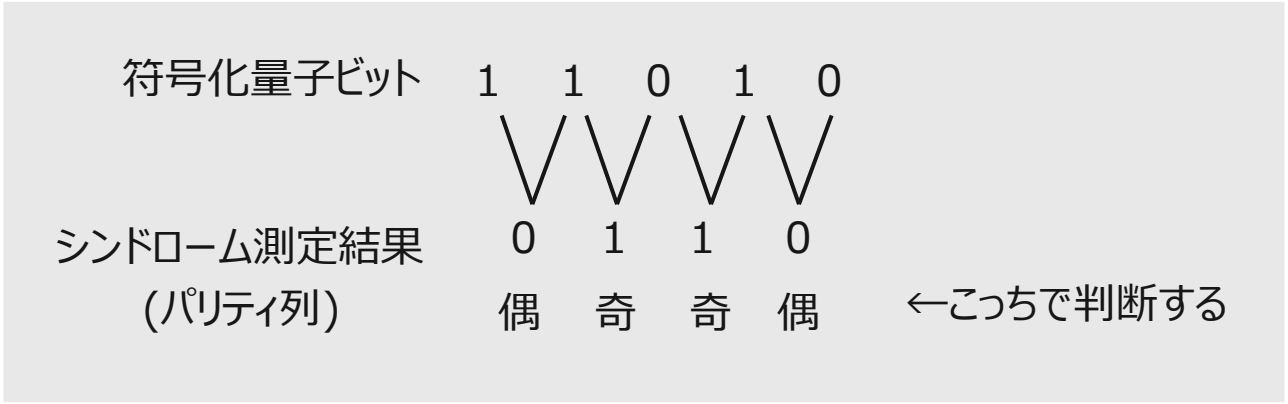
ノイズのないとき、次のような出力結果になる(エラーが無いのでシンドロームの値は常に0)。

Logical 0 : {000 00 00': 1024}　　Logical 1 : {'111 00 00': 1024}

符号化量子ビット測定値　シンドローム測定値(1巡目, 2巡目)

反復符号によるエラー訂正

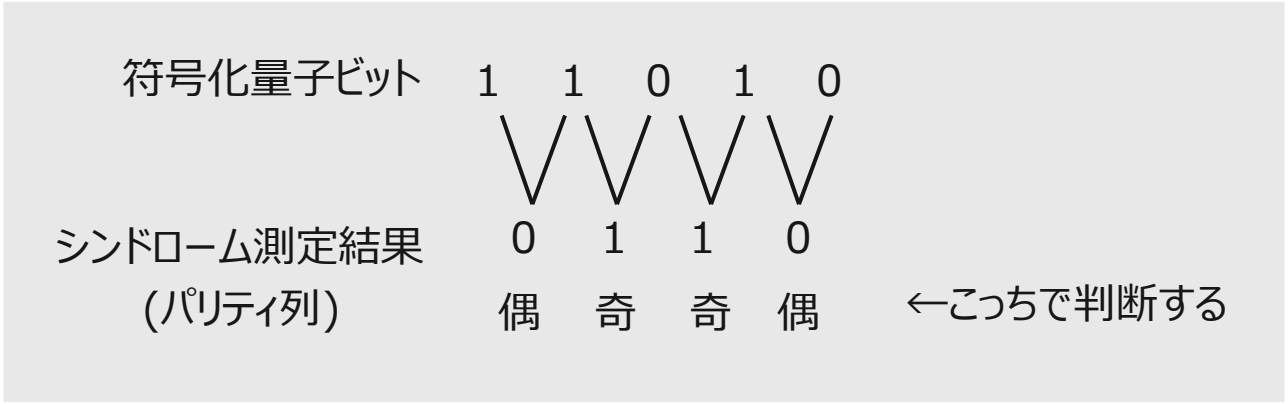
シンドローム測定結果はパリティの視点で理解することができます。



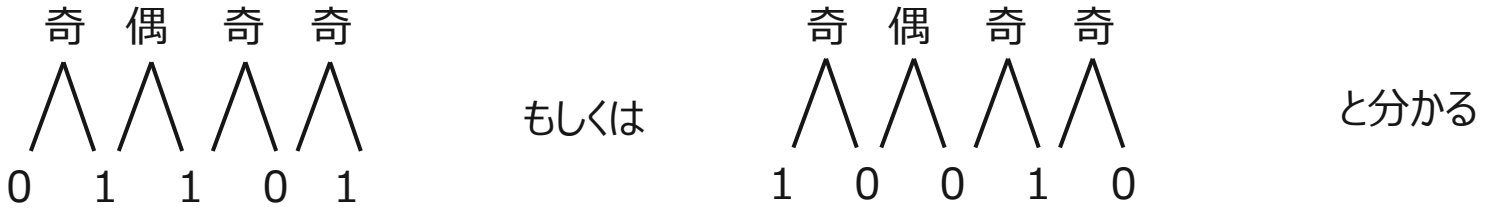
例えばシンドローム結果が　奇 偶 奇 奇　の場合(ビットだと1011)

反復符号によるエラー訂正

シンドローム測定結果はパリティの視点で理解することができます。

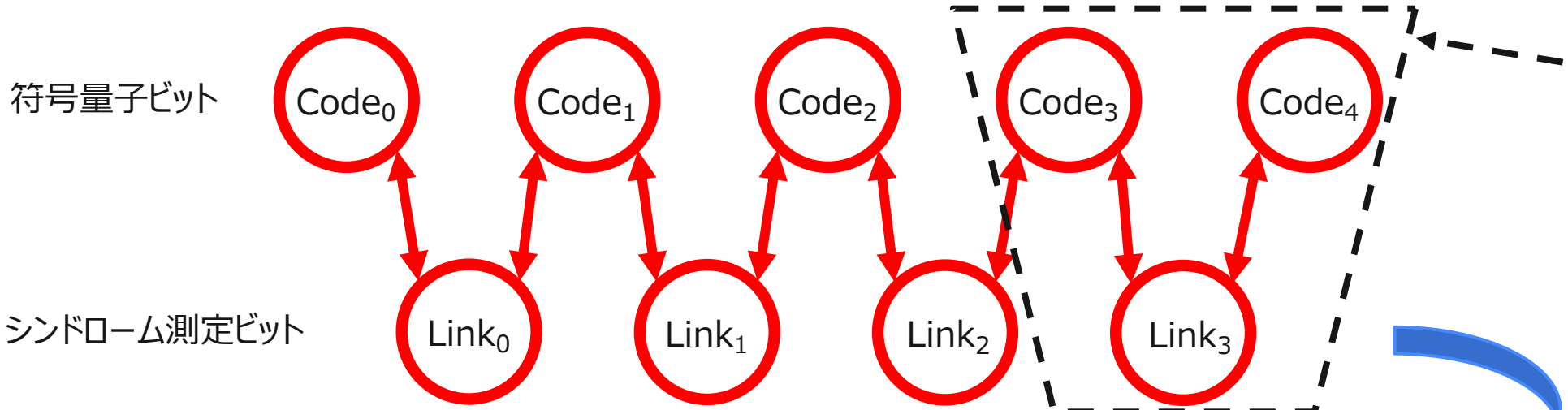


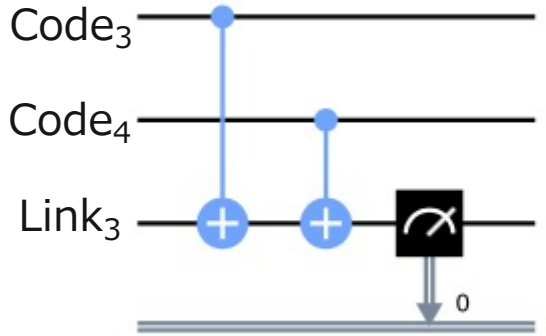
例えばシンドローム結果が　奇 偶 奇 奇　の場合(ビットだと1011)

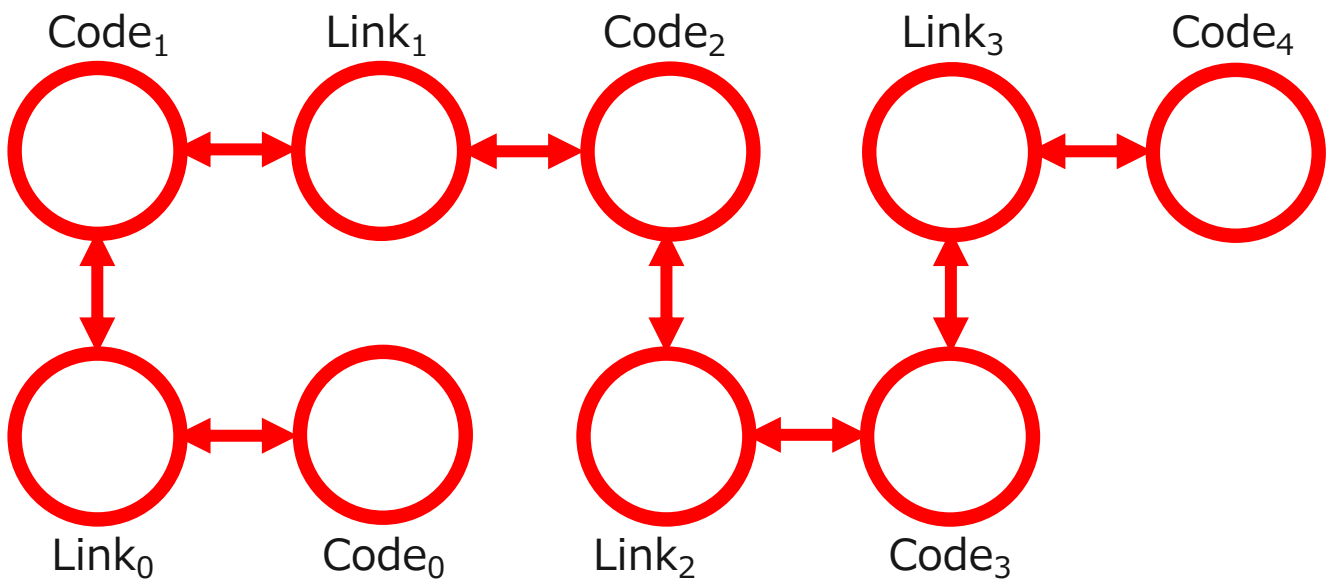


反復符号によるエラー訂正

物理ビットへの割り当てでは、符号量子ビットとシンドローム測定ビットが交互に並んでいます。

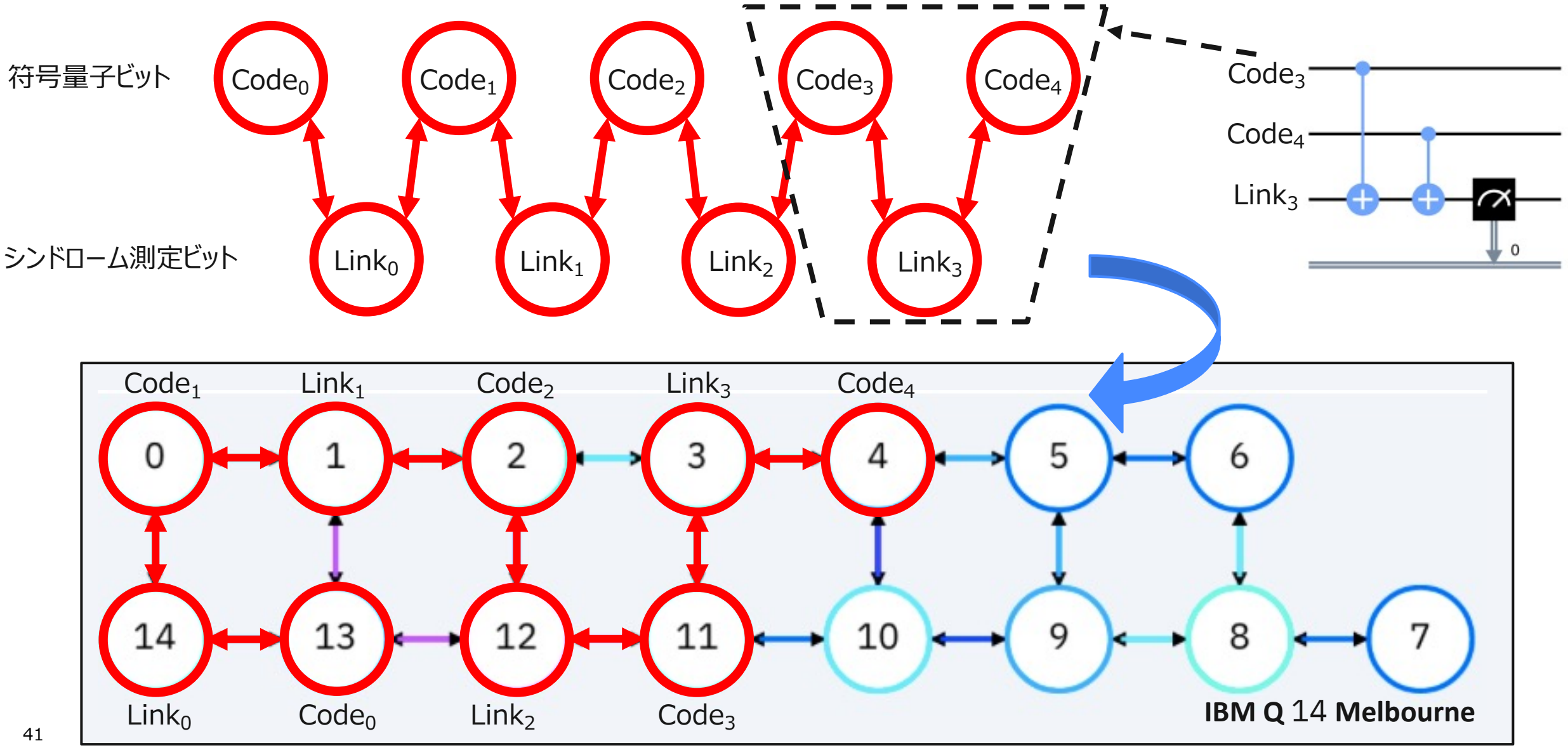






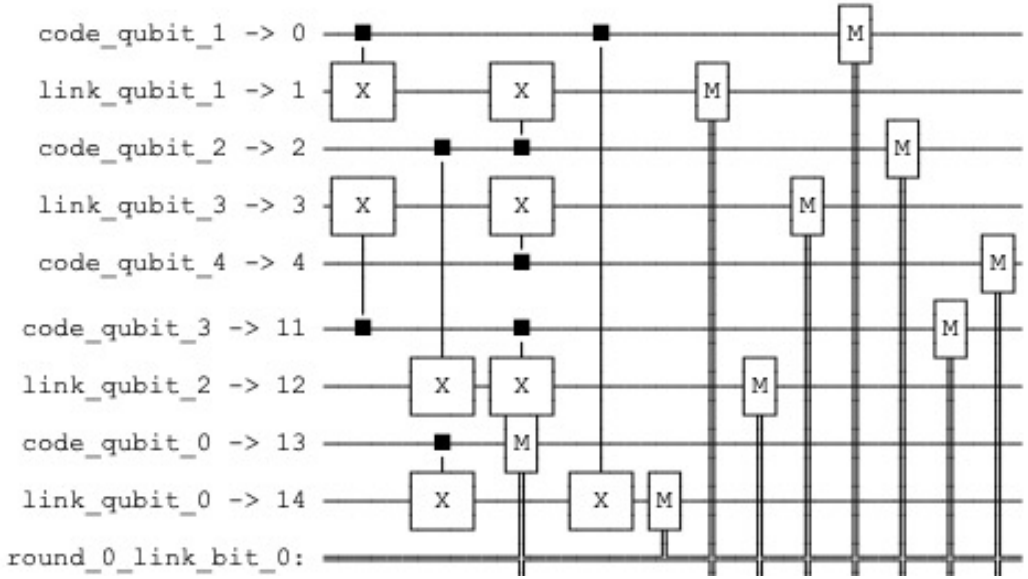
反復符号によるエラー訂正

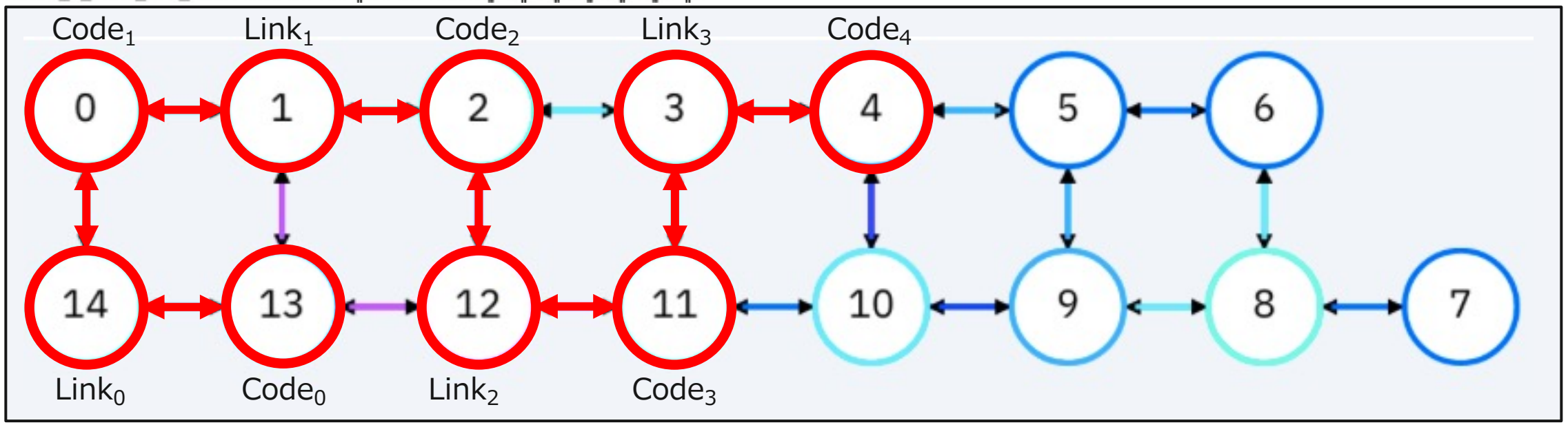
実際のデバイス(Melbourne)へは次のように配置されます。



反復符号によるエラー訂正

量子回路と物理ビットの対応は次のようになります。





反復符号によるエラー訂正

符号化量子ビットとシンドローム測定値を使って、発生したエラーを推測できます。

ノイズのあるとき、次のような出力結果になる。(反復回数n=3で2巡目まで観測; 主要な出力のみ抽出)

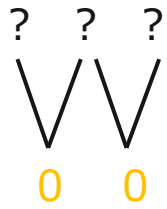
Logical 0: raw results {'000 01 00': 44, '000 10 00': 41, '010 00 00': 43, '001 00 00': 43, '100 00 00': 47, '000 00 00': 485}

例えばこの観測値を復号してみます

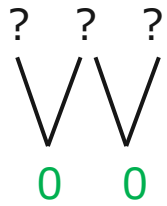
Logical 1: raw results {'111 00 01': 45, '111 10 00': 43, '111 01 00': 52, '111 00 00': 456, '111 00 10': 47}

001 00 00

1巡目シンドローム観測



2巡目シンドローム観測



符号量子ビット観測値

0 0 1

反復符号によるエラー訂正

符号化量子ビットとシンドローム測定値を使って、発生したエラーを推測できます。

ノイズのあるとき、次のような出力結果になる。(反復回数n=3で2巡目まで観測; 主要な出力のみ抽出)

Logical 0: raw results {'000 01 00': 44, '000 10 00': 41, '010 00 00': 43, '001 00 00': 43, '100 00 00': 47, '000 00 00': 485}

例えばこの観測値を復号してみます

Logical 1: raw results {'111 00 01': 45, '111 10 00': 43, '111 01 00': 52, '111 00 00': 456, '111 00 10': 47}

001 00 00

1巡目シンドローム観測　　2巡目シンドローム観測　　符号量子ビット観測値

反復符号によるエラー訂正

符号化量子ビットとシンドローム測定値を使って、発生したエラーを推測できます。

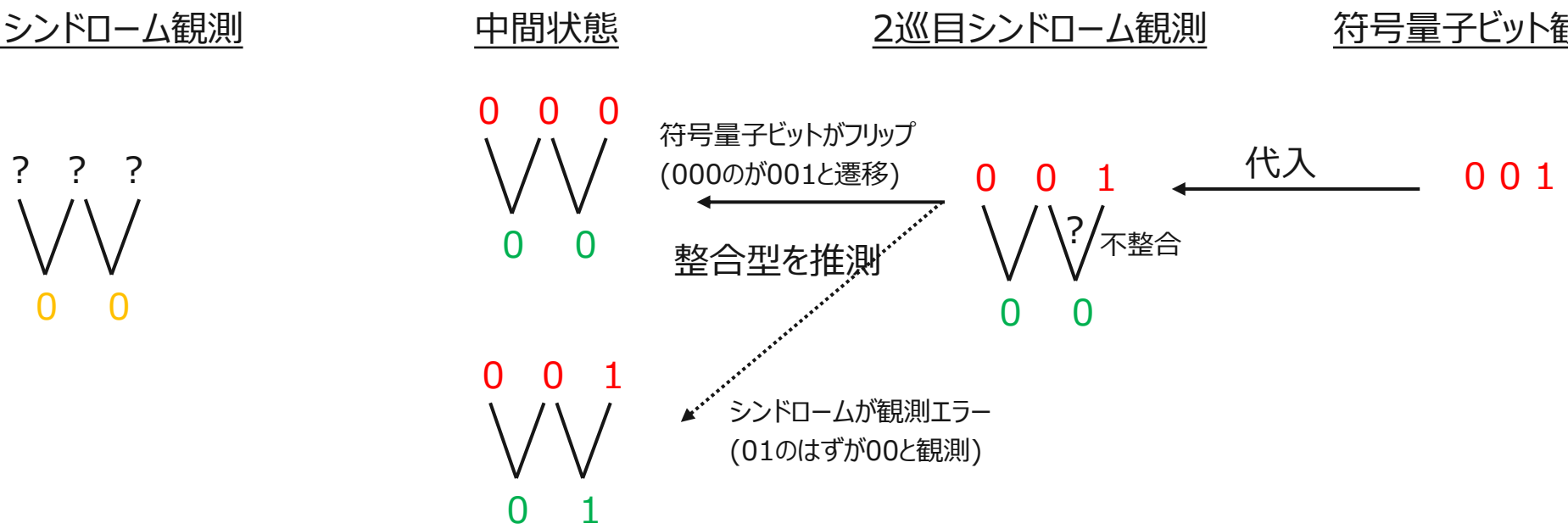
ノイズのあるとき、次のような出力結果になる。(反復回数n=3で2巡目まで観測; 主要な出力のみ抽出)

Logical 0: raw results {'000 01 00': 44, '000 10 00': 41, '010 00 00': 43, '001 00 00': 43, '100 00 00': 47, '000 00 00': 485}

例えばこの観測値を復号してみます

Logical 1: raw results {'111 00 01': 45, '111 10 00': 43, '111 01 00': 52, '111 00 00': 456, '111 00 10': 47}

001 00 00



反復符号によるエラー訂正

符号化量子ビットとシンドローム測定値を使って、発生したエラーを推測できます(ルックアップ・テーブル法)。

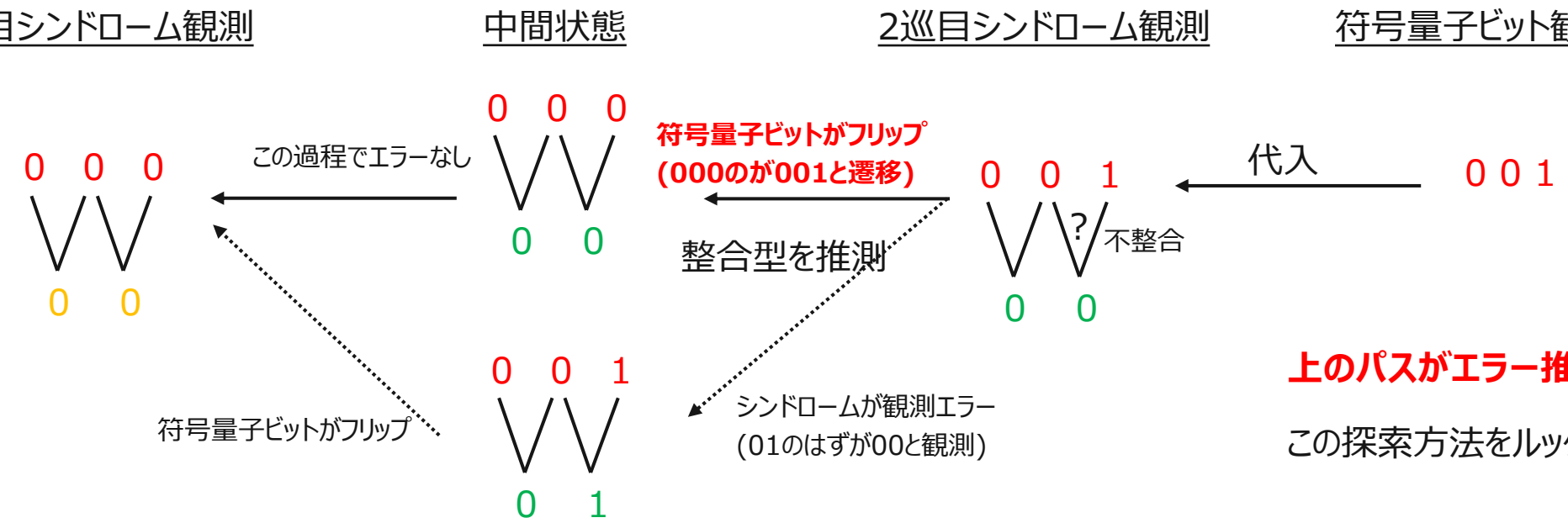
ノイズのあるとき、次のような出力結果になる。(反復回数n=3で2巡目まで観測; 主要な出力のみ抽出)

Logical 0: raw results {'000 01 00': 44, '000 10 00': 41, '010 00 00': 43, '001 00 00': 43, '100 00 00': 47, '000 00 00': 485}

例えばこの観測値を復号してみます

Logical 1: raw results {'111 00 01': 45, '111 10 00': 43, '111 01 00': 52, '111 00 00': 456, '111 00 10': 47}

001 00 00

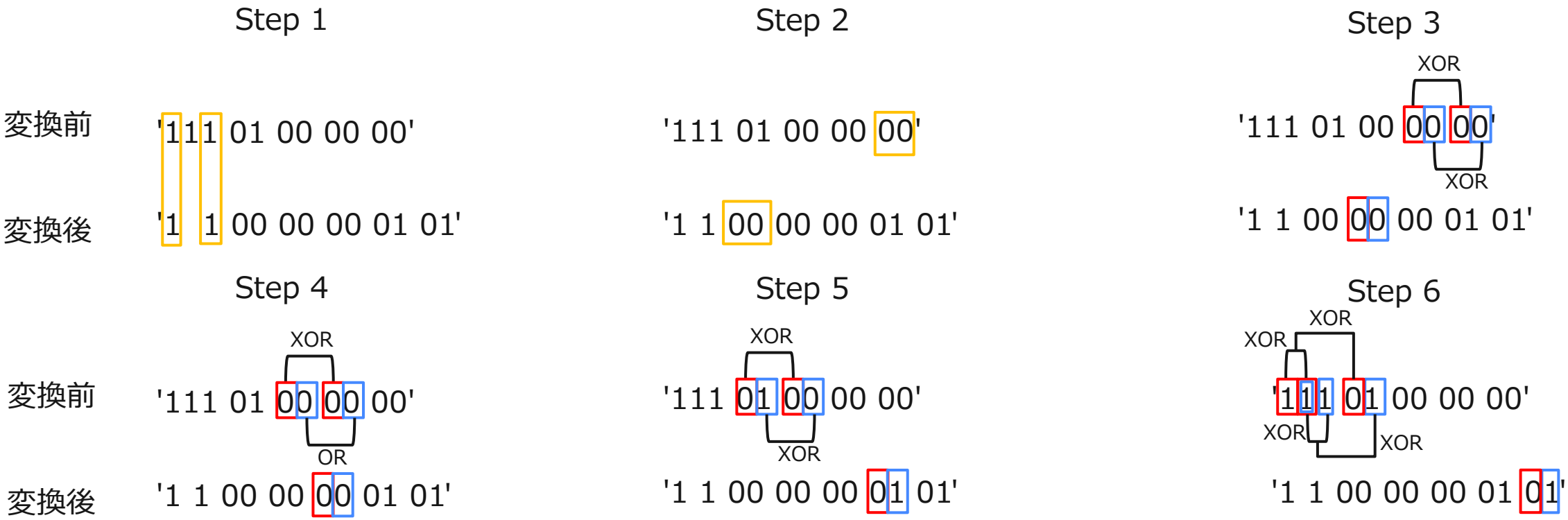


上のパスがエラー推測が可能性が高い

この探索方法をルックアップ・テーブル復号と呼ぶ

反復符号によるエラー訂正

ルックアップ・テーブル法は符号サイズに足して指数関数的に計算時間がかかります。
測定結果をトポロジカル符号に変換するとグラフ理論の問題として効率よく解けるらしい*です。



変換後はシンドロームビット列にエラーの発生数x2だけ“1”が出現する
→ エラー推測の重要な指標になる

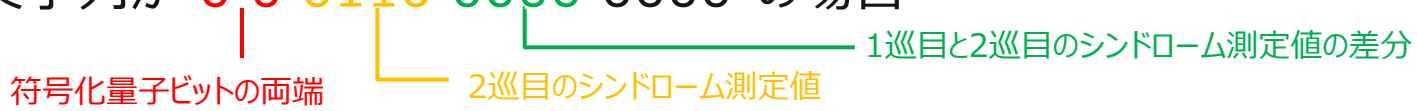
*参考
https://shota.io/public-posts/2017-12-24-surface-code.html
https://www.qec14.ethz.ch/slides/SergeyBravyi.pdf
https://arxiv.org/pdf/1708.02246.pdf
https://arxiv.org/pdf/1307.1740.pdf
https://www.nature.com/articles/ncomms7979

変換後のシンドロームの順序に関してQiskit Textの説明とPython codeの仕様が一致していないので注意

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

変換後の文字列が”0 0 0110 0000 0000”の場合



反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0			
シンドローム測定値		0 1 1 0		
シンドローム測定値(差分)			0 0 0 0	

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

変換後の文字列が”0 0 0110 0000 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0	0 0 1 0 0		
シンドローム測定値		0 1 1 0		
シンドローム測定値(差分)			0 0 0 0	

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

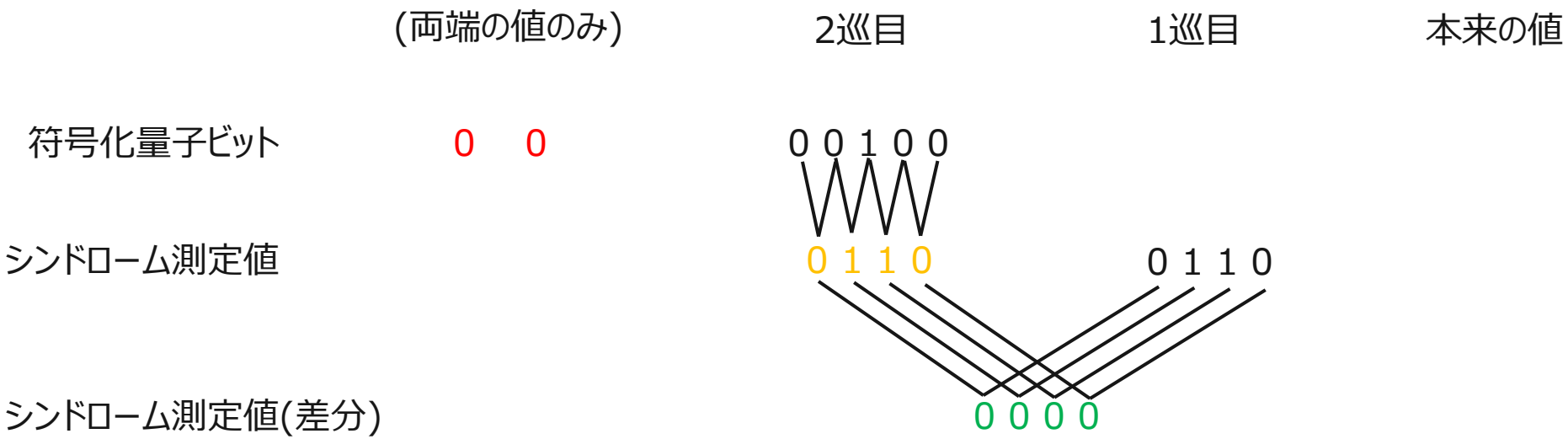
変換後の文字列が”0 0 0110 0000 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。



反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

変換後の文字列が”0 0 0110 0000 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0	0 0 1 0 0	0 0 1 0 0	
シンドローム測定値		0 1 1 0	0 1 1 0	
シンドローム測定値(差分)		0 0 0 0		

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

変換後の文字列が”0 0 0110 0000 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0	0 0 1 0 0	0 0 1 0 0	0 0 0 0 0
シンドローム測定値		0 1 1 0	0 1 1 0	
シンドローム測定値(差分)		0 0 0 0		

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(1)

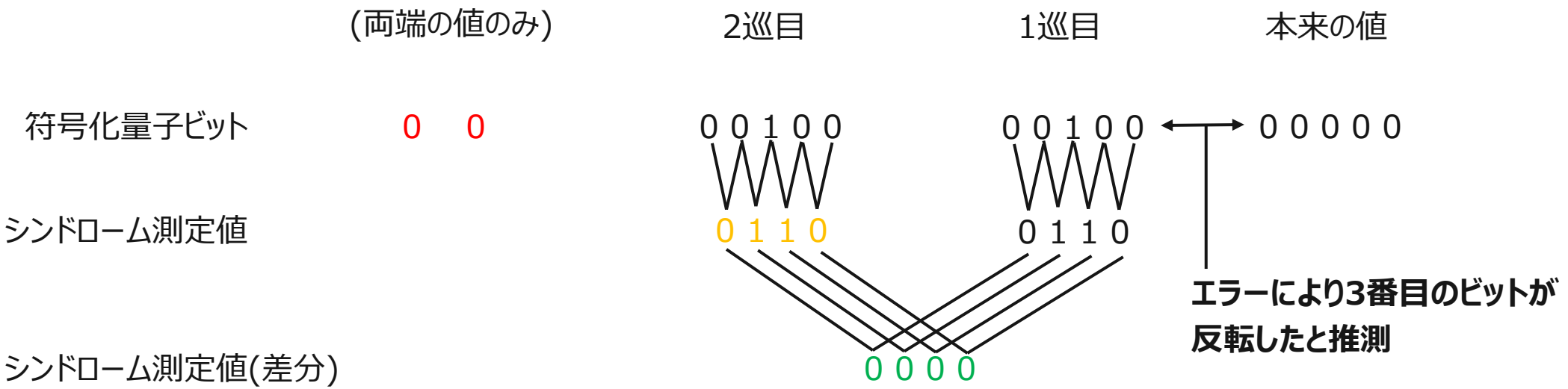
変換後の文字列が”0 0 0110 0000 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

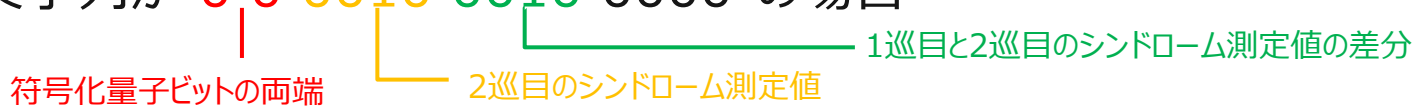
反復数n=5、ラウンド数T=2の反復符号を表します。



反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

変換後の文字列が”0 0 0010 0010 0000”の場合



反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0　0			
シンドローム測定値		0 0 1 0		
シンドローム測定値(差分)			0 0 1 0	

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

変換後の文字列が”0 0 0010 0010 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0	0 0 ? ? ?		
シンドローム測定値		0 0 1 0		
シンドローム測定値(差分)			0 0 1 0	

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

変換後の文字列が”0 0 0010 0010 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。

	(両端の値のみ)	2巡目	1巡目	本来の値
符号化量子ビット	0 0	0 0 ? ? ?		
シンドローム測定値		0 0 1 0	0 0 0 0	
シンドローム測定値(差分)		0 0 1 0		

反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

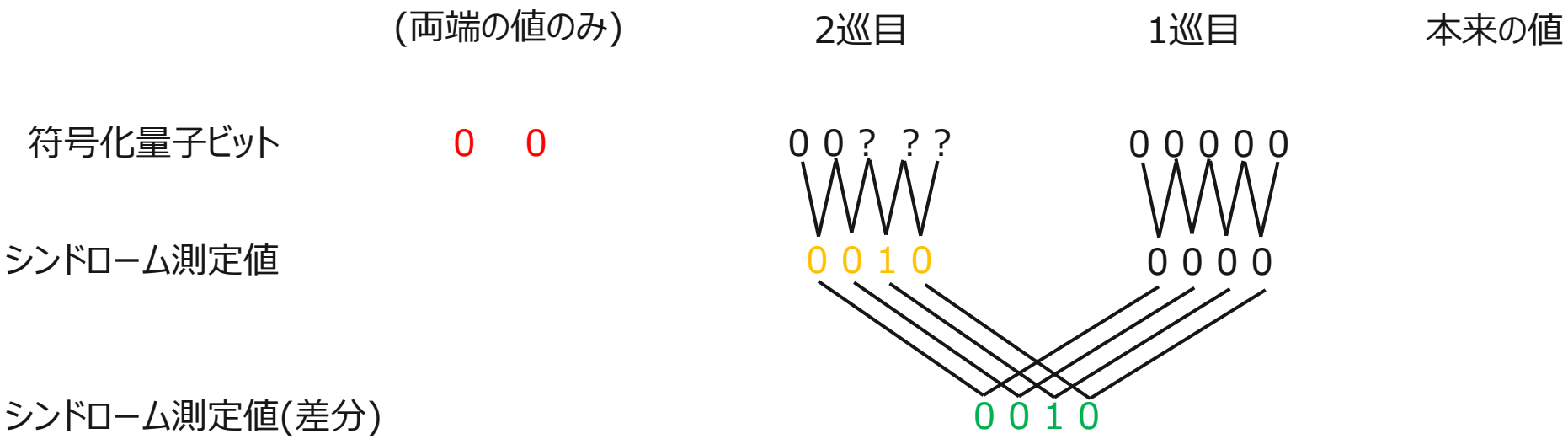
変換後の文字列が”0 0 0010 0010 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。



反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

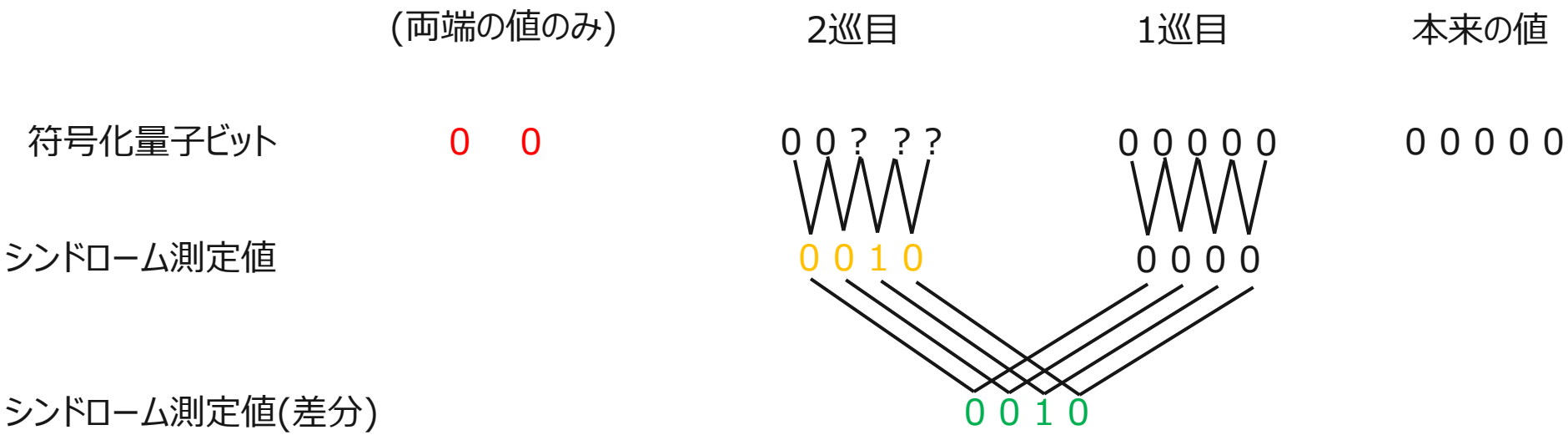
変換後の文字列が”0 0 0010 0010 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。



反復符号によるエラー訂正

Qiskit Textbookの例で変換後の出力文字列の解釈をしてみます(2)

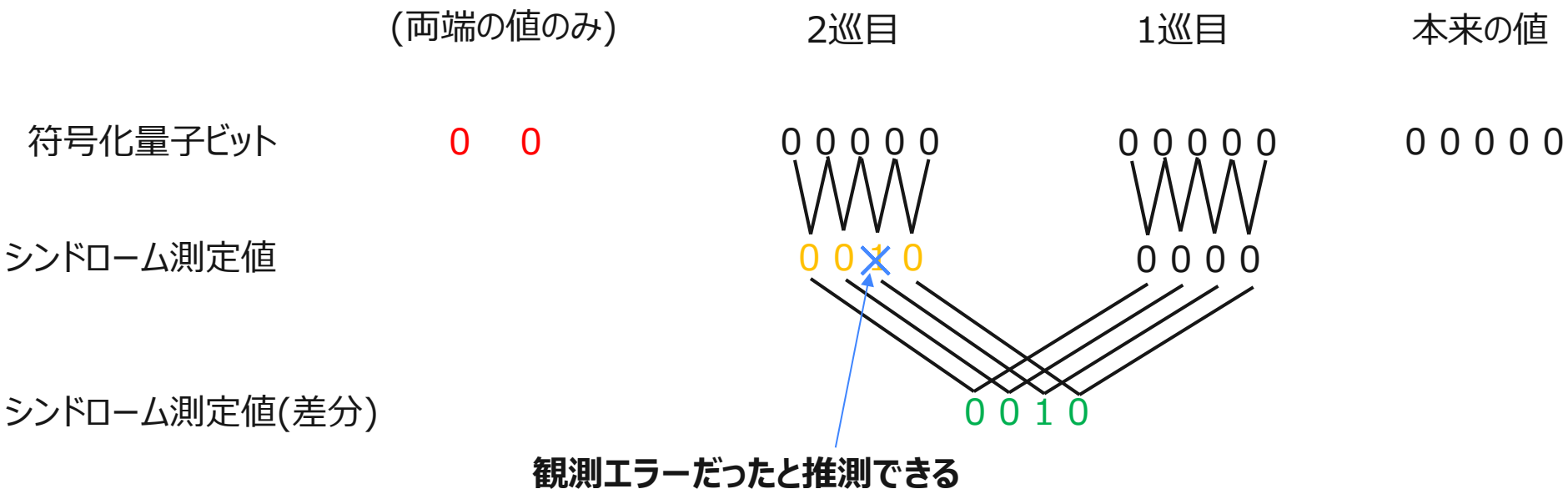
変換後の文字列が”0 0 0010 0010 0000”の場合

符号化量子ビットの両端

2巡目のシンドローム測定値

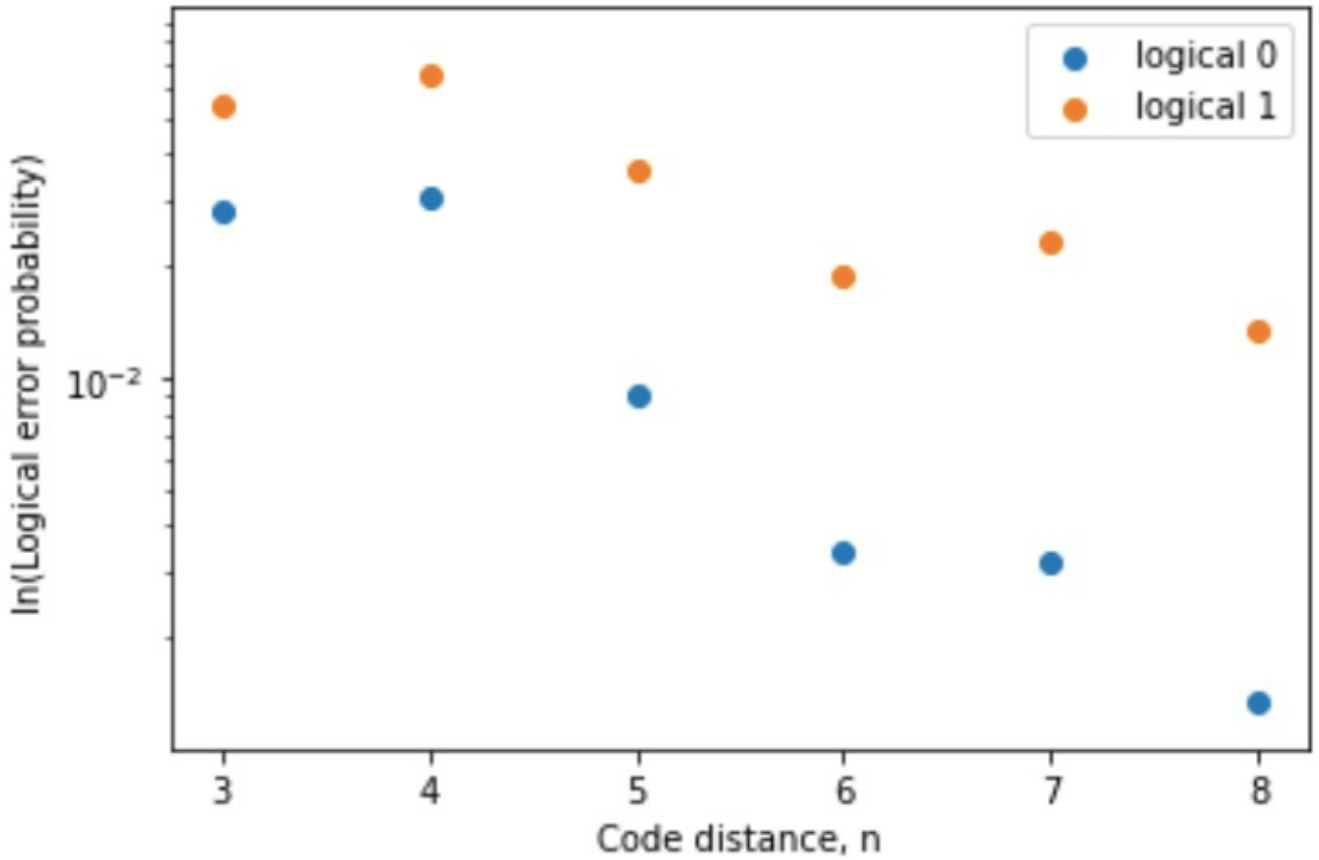
1巡目と2巡目のシンドローム測定値の差分

反復数n=5、ラウンド数T=2の反復符号を表します。



反復符号によるエラー訂正

最後に変換後のビット列をQiskit IgnisのGraphDecoderを使って復号し、エラー率計算を行います。
反復回数(Code Distance)を増やすほどエラー率が減少していることが分かります。



1. 測定エラー軽減

2. 反復符号によるエラー訂正(まとめ)
- 1つの論理ビットを複数の物理ビットで表現
- シンドローム測定で状態を壊さず観測を何度も行える
- シンドローム測定結果と符号量子ビットから復号が可能

3. ランダム化ベンチマーキング

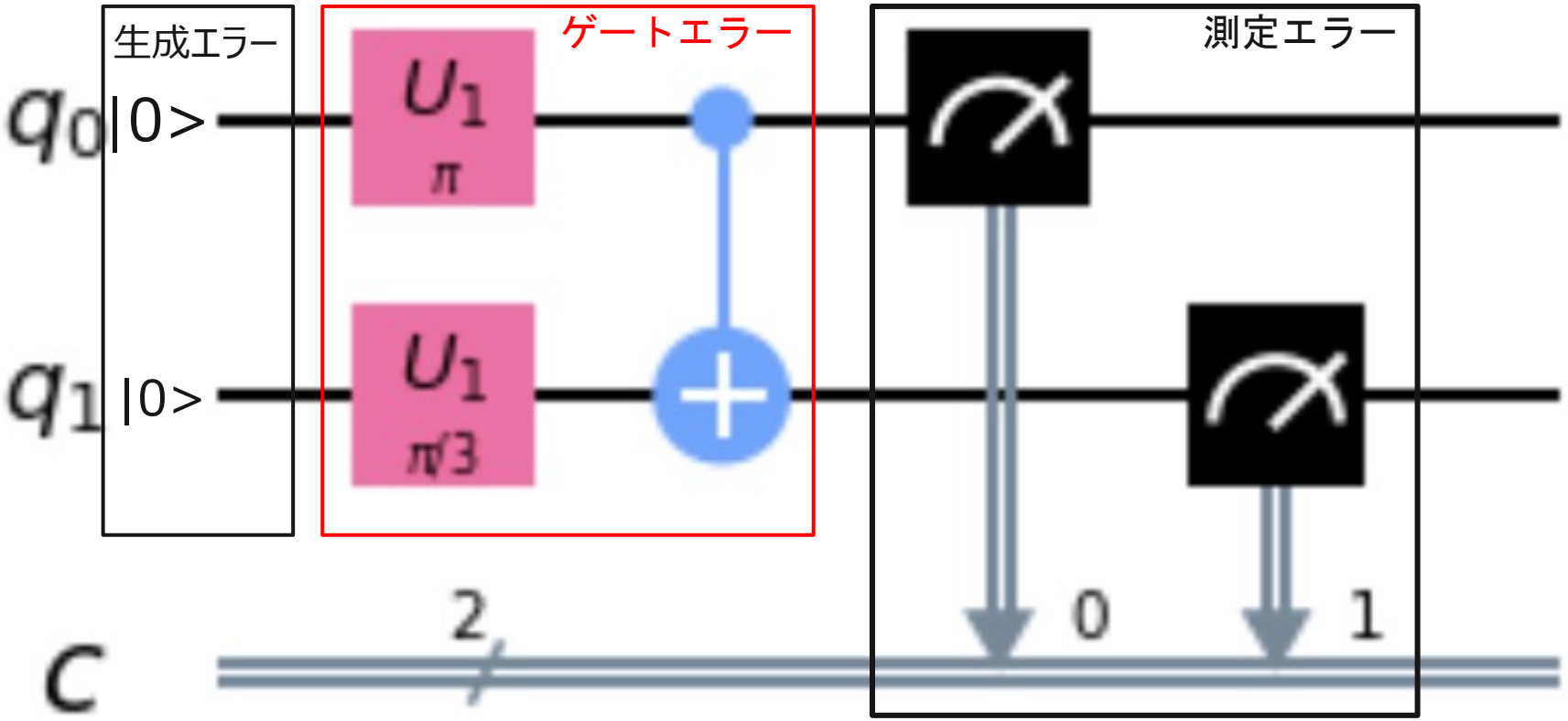
アジェンダ

1. 測定エラー軽減

2. 反復符号によるエラー訂正

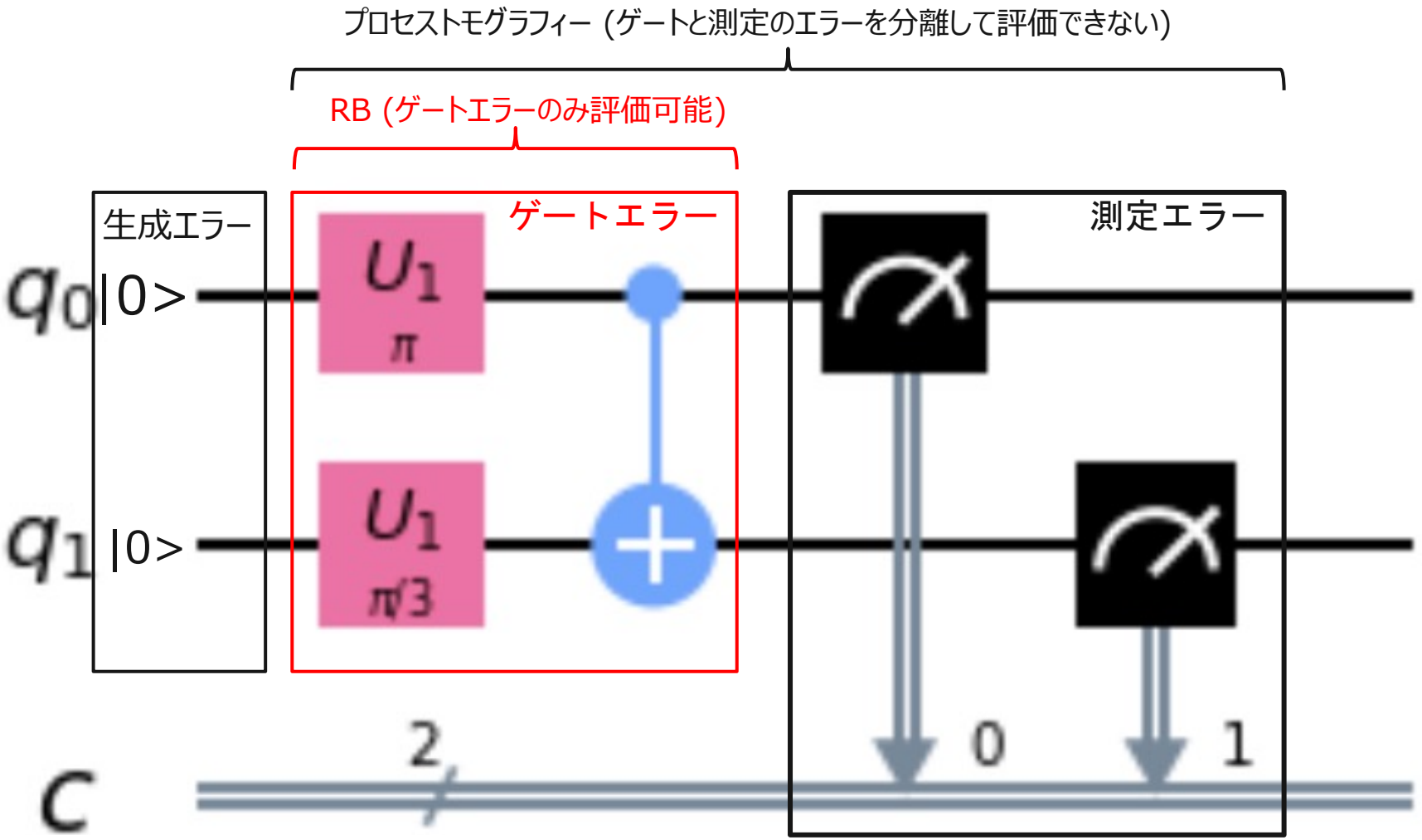
3. ランダム化ベンチマーキング

ランダム化ベンチマーキング(RB)ではゲートエラーの評価を行います



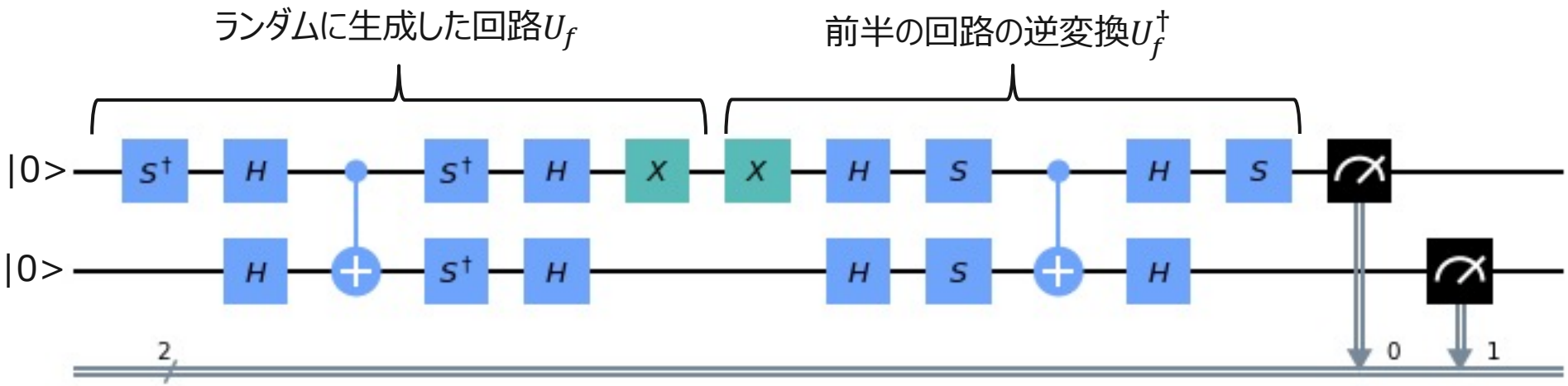
ランダム化ベンチマーキング

従来はプロセストモグラフィーで回路全体のエラー評価をしていましたが、
RBでゲートエラー部分のみのエラー評価が可能になりました。



ランダム化ベンチマーキング

RBではエラーがなければ恒等変換となる深い回路を作り、状態が変わるかを調べる手法です。



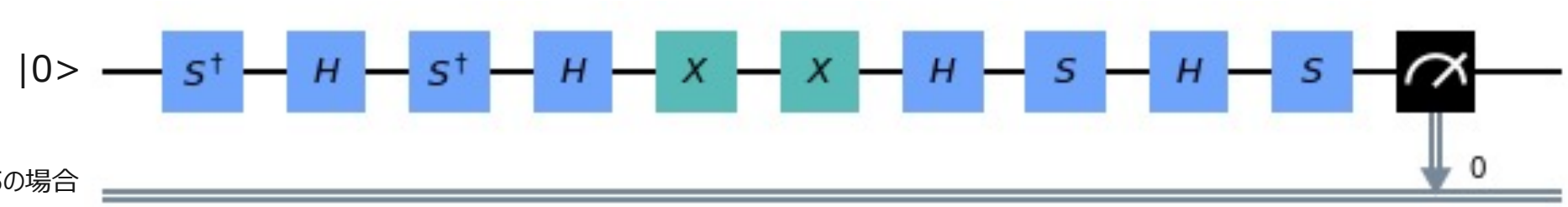
- $U_f^\dagger\ U_f = I$より観測値は初期状態|00>と等しいはず
- エラーがあると|00>以外の状態が観測される
- 回路U_fの深さmとエラー率の関係からゲートエラーを評価

ランダム化ベンチマーキング

生成エラー、観測エラー、ゲートエラー全てを考慮に入れた枠組みで、
初期状態と観測値が一致する確率(生存確率)を計算していきます。

生存確率: $Tr\left[E_{\psi}S_{i_m}(\rho_{\psi})\right]$

- ρ_{ψ}: 生成エラー
- S_{i_m}: ゲートエラー
- E_{ψ}: 測定時エラー



簡単のため1量子ビットで説明
前半のランダム回路の深さm=5の場合

ランダムに生成した回路にノイズが混じるモデルを定義します。

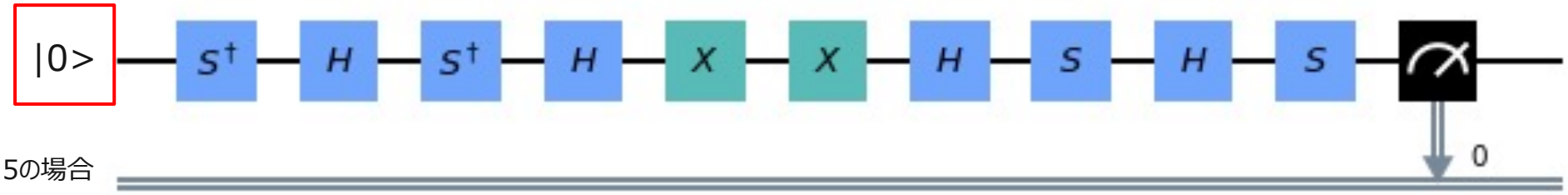
初期状態を準備する際にエラーが発生する可能性を考慮

・初期状態|0>を作ろうとして確率p_{gen}で|1>が作られるエラーモデルは密度演算子ρ_0で記述される

$$\rho_0 = (1 - p_{gen})|0\rangle\langle 0| + p_{gen}|1\rangle\langle 1|$$

・少し一般化すると、初期状態$|\psi\rangle$を作ろうとして確率pでビットフリップが発生するエラーモデルは密度演算子ρ_ψで記述される

$$\rho_\psi = (1 - p_{gen})|\psi\rangle\langle\psi| + p_{gen}X|\psi\rangle\langle\psi|X$$



簡単のため1量子ビットで説明
前半のランダム回路の深さm=5の場合

ランダムに生成した回路にノイズが混じるモデルを定義します。

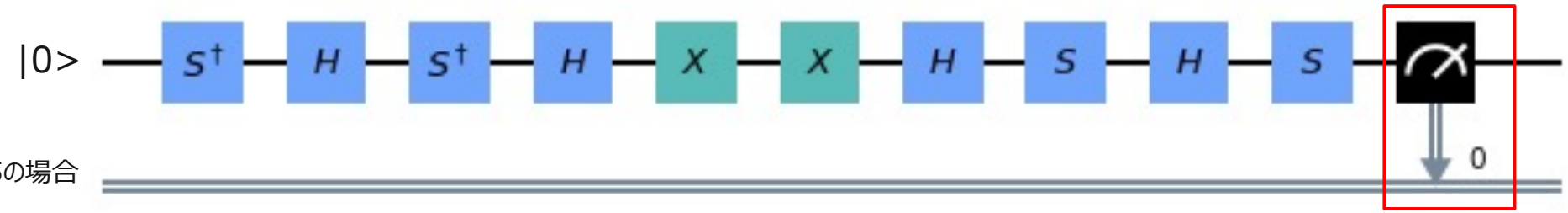
測定時にエラーが発生するモデルを考慮

・状態|0>を観測しようとして確率pで|1>が得られるエラーモデルは測定演算子E_0で記述される

$$E_0 = (1 - p_{obs})|0\rangle\langle 0| + p_{obs}|1\rangle\langle 1|$$

・少し一般化すると、終状態$|\psi\rangle$を観測しようとして確率pでビットフリップが発生するエラーモデルは測定演算子E_ψで記述される

$$E_\psi = (1 - p_{obs})|\psi\rangle\langle\psi| + p_{obs}X|\psi\rangle\langle\psi|X$$



簡単のため1量子ビットで説明
前半のランダム回路の深さm=5の場合

ランダムに生成した回路にノイズが混じるモデルを定義します。

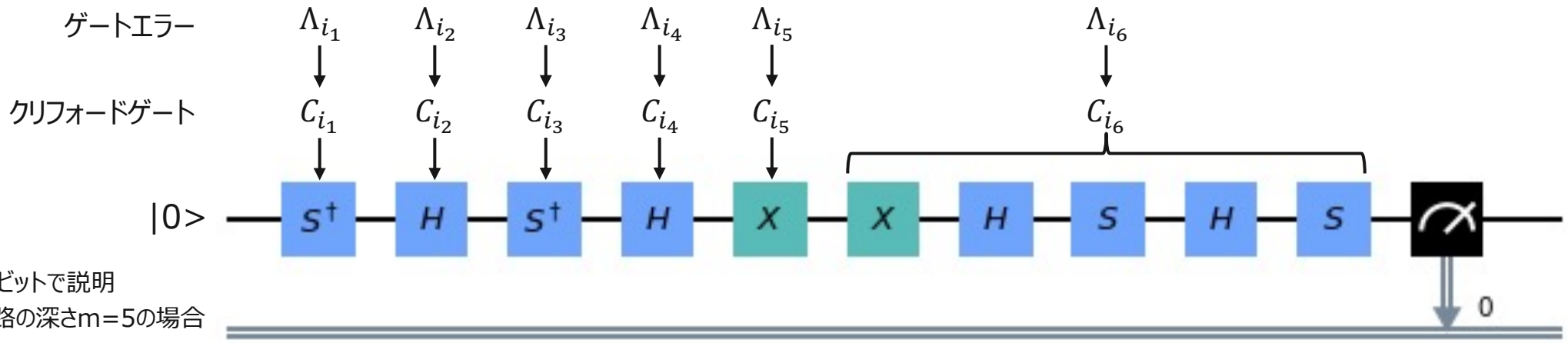
各ゲートエラーを次のモデルで定式化

・クリフォード群*からランダムに作られたゲートを$\{C_{i_1},\cdots, C_{i_{m+1}}\}$とします

・ 各ゲートのエラーを$\{\Lambda_{i_1},\cdots, \Lambda_{i_{m+1}}\}$とします

・この時の回路の量子状態を次のように表します。

$$S_{i_m}(\rho_\psi) = \Lambda_{i_{m+1}} C_{i_{m+1}} \cdots \Lambda_{i_1} C_{i_1}\ \rho_\psi\ \Lambda_{i_1}^\dagger C_{i_1}^\dagger \cdots \Lambda_{i_{m+1}}^\dagger C_{i_{m+1}}^\dagger = \left[\odot_{j=1}^{m+1} (\Lambda_{i_j} \circ C_i j)\right](\rho_\psi)$$



簡単のため1量子ビットで説明
前半のランダム回路の深さm=5の場合

＊S, H, CZから作ることのできるゲートセット(X, Y, ZもS, Hから作成可能)

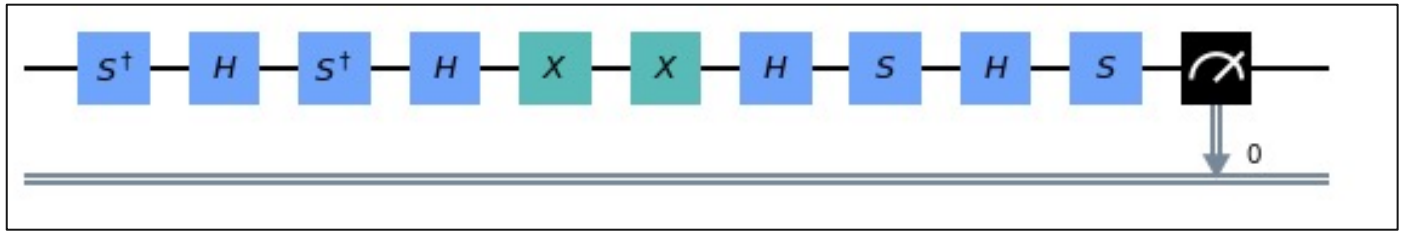
ランダム化ベンチマーキング

ランダム回路を多数のパターンで用意して、生存確率の平均(フィデリティー平均)を求めます。

フィデリティー平均　$F_{seq}(\mathrm{m}, |\psi\rangle) = \mathrm{Tr}\left[E_\psi S_{K_m}(\rho_\psi)\right] = \frac{1}{K_m}\sum_{\boldsymbol{i_m}} \mathrm{Tr}\left[E_\psi S_{\boldsymbol{i}_m}(\rho_\psi)\right]$

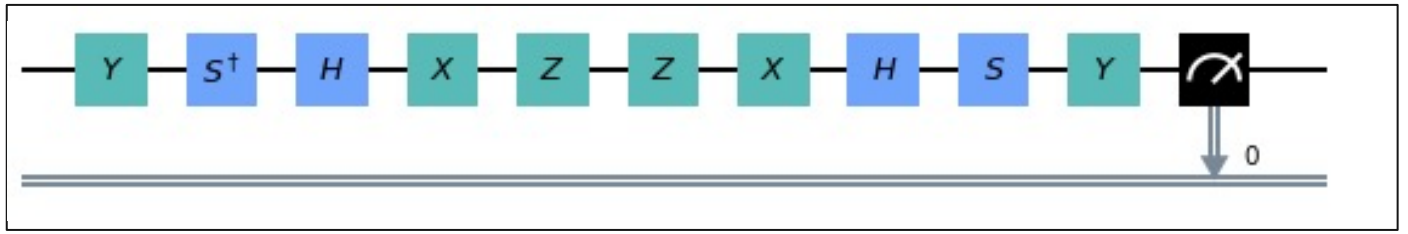
$K_m = 3$パターン
のランダム回路

$\boldsymbol{i_m} = \{i_1, \ldots, i_m\}$



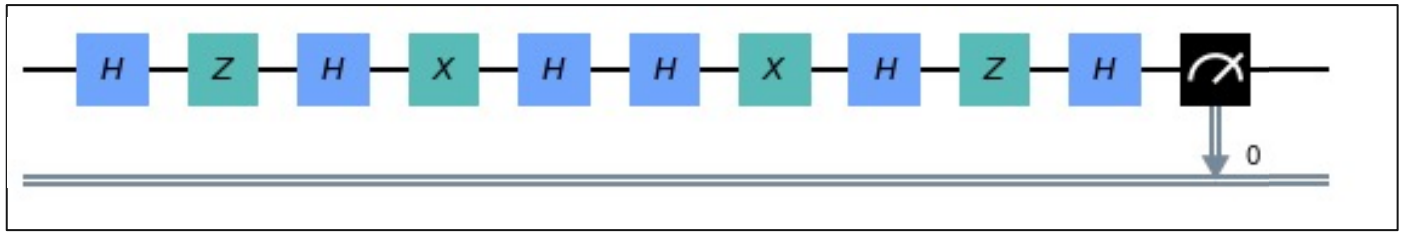
$\mathrm{Tr}\left[E_\psi S_{\boldsymbol{i}_m}(\rho_\psi)\right]$

$\boldsymbol{j_m} = \{\mathrm{j}_1, \ldots, j_m\}$



$\mathrm{Tr}\left[E_\psi S_{\boldsymbol{j}_m}(\rho_\psi)\right]$

$\boldsymbol{k_m} = \{k_1, \ldots, k_m\}$



$\mathrm{Tr}\left[E_\psi S_{\boldsymbol{k}_m}(\rho_\psi)\right]$

ランダム化ベンチマーキング

フィデリティー平均をフィッティングして平均エラー率(クリフォード単位エラー;EPC)が求まります。

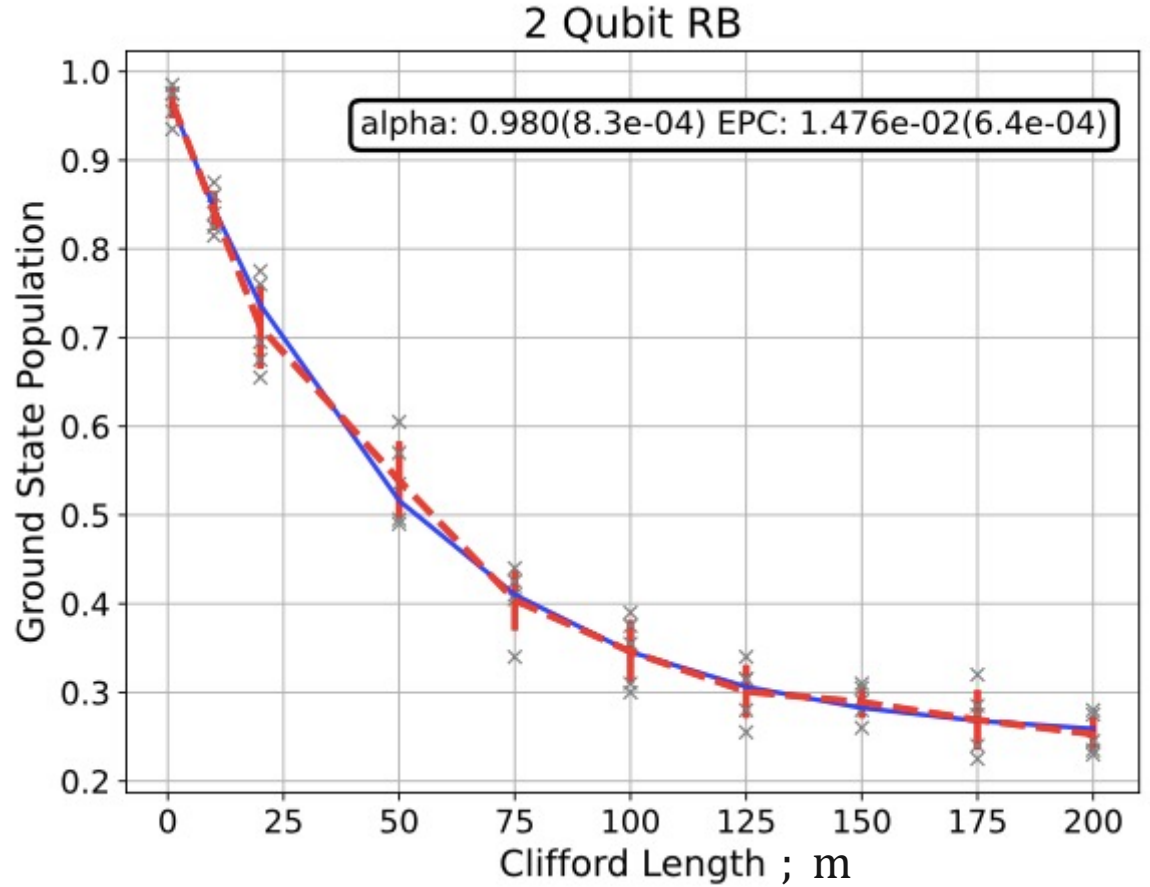
フィデリティー平均
(ゲート依存性がない場合)

$$F_{seq}^{(0)}(\mathrm{m}, |\psi\rangle) = A_0\alpha^m + B_0$$

・A_0, B_0に生成エラー、測定エラーの寄与が含まれる。
・mでフィッティングするとパラメータαが計算可能

平均エラー率
(クリフォード単位エラー;EPC)

$$r = \frac{2^{\mathrm{n}} - 1}{2^{\mathrm{n}}}(1 - \alpha)$$



1. 測定エラー軽減

2. 反復符号によるエラー訂正

3. ランダム化ベンチマーキング(まとめ)

・ゲートエラーを評価する手法
・エラーがなければ恒等変換となる回路を作り、初期状態との差分を評価
・初期状態と観測結果の差と、ゲート数の関係からゲート単位のエラーを算出

1. 測定エラー軽減

・計算基底(|00>,…|11>)の観測結果から校正行列Mを作成
・Mの逆行列で観測エラーを軽減

2. 反復符号によるエラー訂正

・1つの論理ビットを複数の物理ビットで表現
・シンドローム測定で状態を壊さず観測を何度も行える
・シンドローム測定結果と符号量子ビットから復号が可能

3. ランダム化ベンチマーキング

・ゲートエラーを評価する手法
・エラーがなければ恒等変換となる回路を作り、初期状態との差分を評価
・初期状態と観測結果の差と、ゲート数の関係からゲート単位のエラーを算出