誤り耐性量子コンピュータ・アーキテクチャ

理化学研究所量子コンピュータ研究センター
超伝導量子計算システム研究ユニット
基礎科学特別研究員
上野 洋典

上野 洋典（うえの ようすけ）

- 所属：理研RQC 超伝導量子計算システム研究ユニット
  基礎科学特別研究員（ポスドク）
- 経歴：
  - 1994年 福岡県北九州市門司区生まれ
  - 2013.4~2017.3: 東大工学部計数工学科
  - 2017.4~2022.3: 東大情報理工 システム情報学専攻
    - 指導教員：中村宏先生、近藤正章先生（慶應）
    - 博論: 超伝導回路を用いたオンライン量子誤り訂正
  - 2022.5~2023.2: ミュンヘン工科大学 訪問研究員（Prof. Martin Schulz）
  - 2023.4~現在:**（本務）理研RQC 基礎科学特別研究員**
  - 2025.5~現在:**（兼務）東大 中村・高瀬研 特任助教（非常勤）**
- 研究対象、興味：
  - 計算機アーキテクチャ、超伝導ディジタル（SFQ）回路、誤り耐性量子計算
- 物理の方々に囲まれつつ計算機アーキテクチャやってます!

ドイツでの
最初と最後のビール

今日の内容

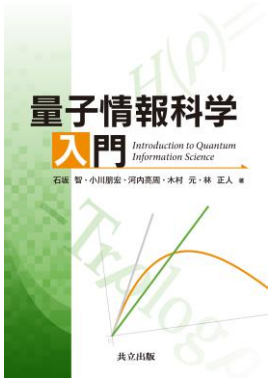
- 量子計算機アーキテクチャ研究の概要
- 量子計算の基礎
- 量子誤り訂正の基礎
- 表面符号を用いた量子誤り訂正
  - 上野の博論（DAC2021, HPCA2022）
- 表面符号＋格子手術を用いた誤り耐性量子計算（FTQC）
- ロードストア型FTQCアーキテクチャ
  - HPCA2025
- 高効率な格子手術FTQCのための2.5次元アーキテクチャ
  - HPCA2026投稿中論文

今回話さないこと

- 量子情報理論の基礎
- 量子誤り訂正、誤り耐性量子計算の理論的な側面
- 表面符号以外の量子誤り訂正符号
- 格子手術以外の論理演算手法
- 量子計算機のアプリケーション
- 量子計算周りの計算量理論
  - Aaronson, “Quantum Computing Since Democritus”
  - 「とても強い計算量クラスのコンピュータとその実現方法」 https://qiita.com/iKodack/items/d606a09f0a40b95bf2b6

参考になる書籍・講義動画

- 嶋田, “量子コンピューティング”
  - 網羅的に多くのトピックが触れられている
  - 格子手術に触れている現状唯一の日本語文献？
- 石坂ほか, “量子情報科学入門”
  - この発表で省略した理論的な詳細が書かれている
  - 表面符号については触れていない
    - 追記：第2版では触れているかも（未確認）
- 阪大藤井先生のYouTube
  - 第10回以降が量子誤り訂正
  - 歴史的経緯、理論的側面、
    誤り耐性量子計算手法など様々なトピック



発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

計算機アーキテクチャ

What is Computer architecture?

- “*Computer Architecture* is the science and art of selecting and interconnecting hardware components to create computers that meet **functional**, **performance** and **cost** goals.”

  - WWW Computer Architecture Page

建築（アーキテクチャ）とのアナロジー

計算機アーキテクチャ分野の研究対象

高
抽象度
低

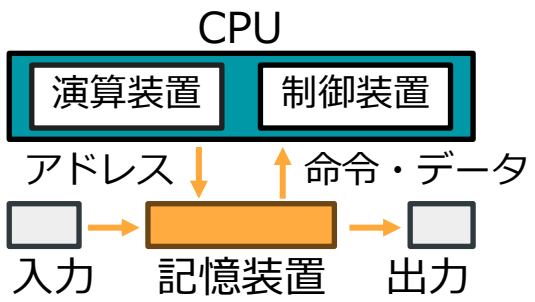
計算機全体の構成
計算機を取り巻く環境

システムアーキテクチャ
ノイマン型アーキテクチャ,
キャッシュ, 分散システム

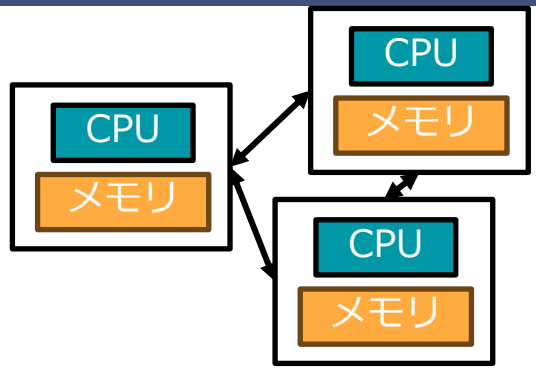
命令セットアーキテクチャ（ISA）
RISC, CISC, VLIW

計算機の中身

マイクロアーキテクチャ
CPU, パイプライン,
投機的実行

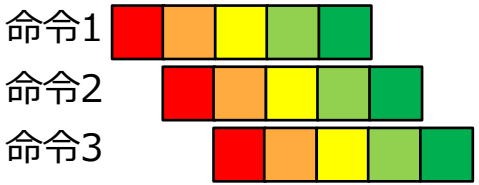


ノイマン型アーキテクチャ

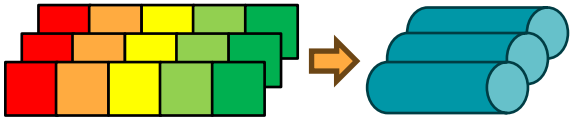


分散システム

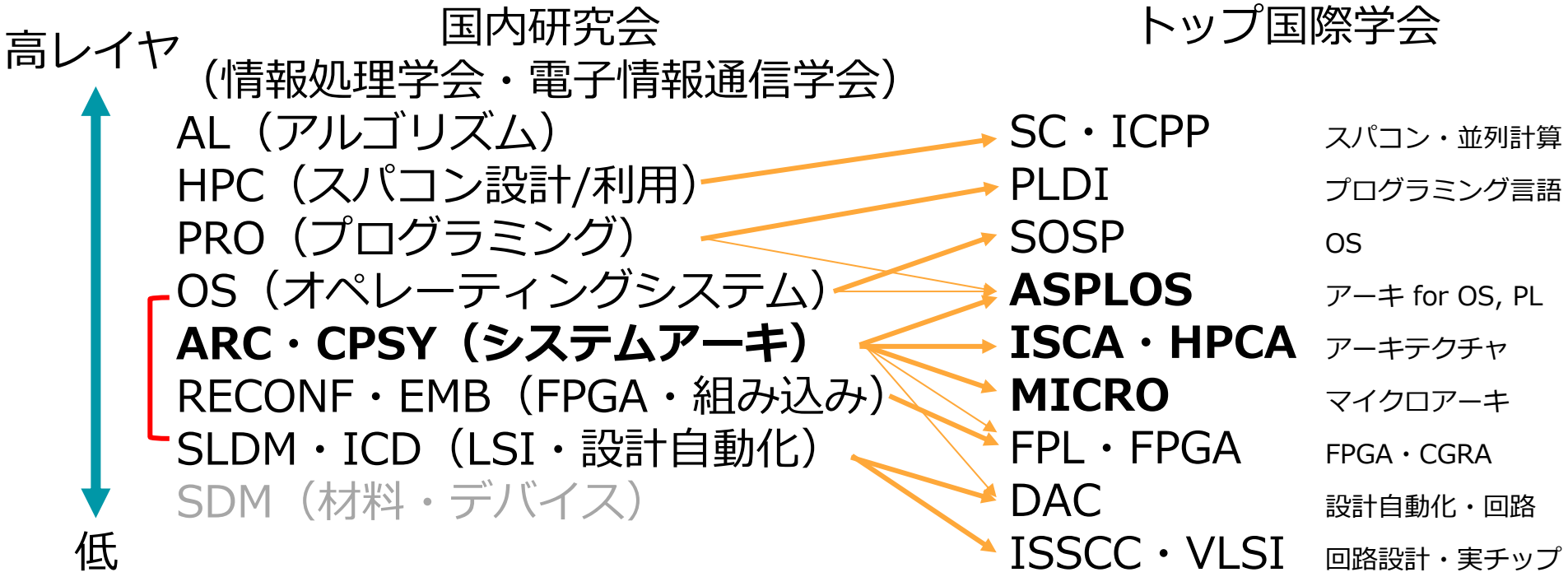
x86 (CISC) vs. RISC-V (RISC)



パイプライン


ベクトル演算器

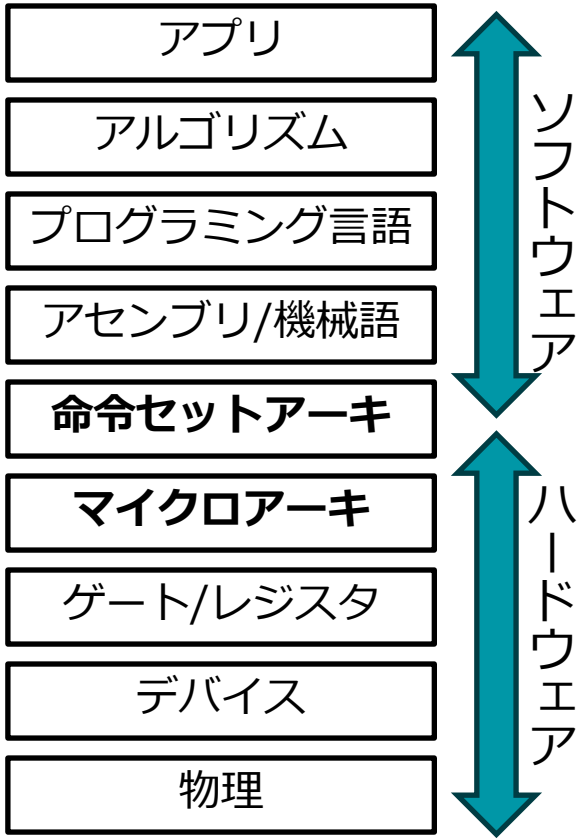
計算機アーキテクチャの隣接分野



上は計算機と不可分なソフトウェアまで
下はビットまで

計算機アーキテクチャの役割

- 抽象化のレベルを適切に定める
  - プログラマが何を意識して何を意識しないか
- 上位、下位レイヤの変化に応じて
  コストを最小化しつつ計算機の設計を
  更新
- 各コンポーネントを統合して
  計算機全体としての性能見積もり、
  フィードバック
  - どこに何を押し付けるか決める
- 計算能力の継続的な向上の展望を示す



発表内容

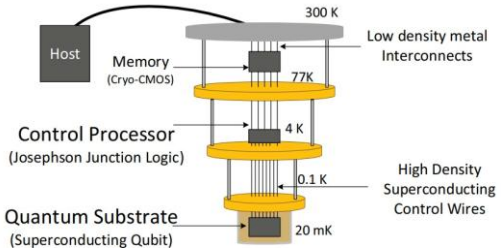
- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

量子計算機アーキテクチャ分野

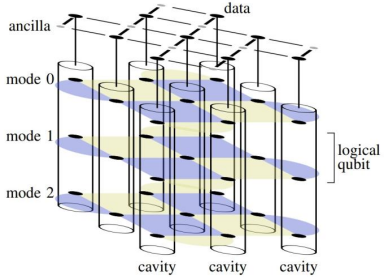
システムアーキテクチャ

命令セットアーキテクチャ（ISA）

マイクロアーキテクチャ



Cryogenic QEC architecture
MICRO2017

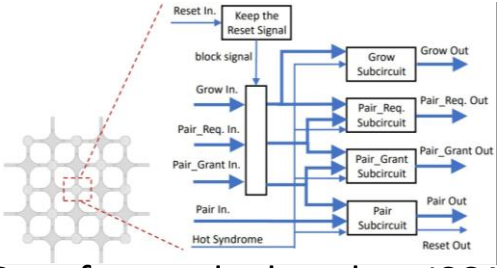


Virtualized Logical Qubit
MICRO2020

Table I. Overview of eQASM instructions. The operator `::` concatenates the two bit strings.

Type	Syntax	Description
Control	`CMP Rs, Rt`	(**CoMP**are GPR `Rs` and `Rt` and store the result into the comparison flags.
	`BR <Comp. Flag>, Offset`	(**BR**anch) Jump to `PC + Offset` if the specified comparison flag is '1'.
Data Transfer	`FBR <Comp. Flag>, Rd`	(**Fetch Branch Register**) Fetch the specified comparison flag into GPR Rd.
	`LDI Rd, Imm`	(**LoaD Immediate**) Rd = sign_ext(Imm[19..0], 32).
	`LDUI Rd, Imm, Rs`	(**LoaD Unsigned Immediate**) Rd = Imm[14..0]::Rs[16..0].
	`LD Rd, Rt(Imm)`	(**LoaD** from memory) Load data from memory address `Rt + Imm` into GPR Rd.
	`ST Rs, Rt(Imm)`	(**STore** to memory) Store the value of GPR `Rs` in memory address `Rt + Imm`.
	`FMR Rd, Qi`	(**F**etch **M**easurement **R**esult) Fetch the result of the last measurement instruction on qubit `i` into GPR Rd.
Logical	`AND/OR/XOR Rd, Rs, Rt` `NOT Rd, Rt`	Logical and, or, exclusive or, not.
Arithmetic	`ADD/SUB Rd, Rs, Rt`	Addition and subtraction.
Waiting	`QWAIT Imm` `QWAITR Rs`	(**Q**uantum **WAIT** **I**mmediate/**R**egister) Specify a timing point by waiting for the number of cycles indicated by the immediate value `Imm` or the value of GPR `Rs`.
Target Specify	`SMIS Sd, <Qubit List>` `SMIT Td, <Qubit Pair List>`	(**S**et **M**ask **I**mmediate for **S**ingle-/**T**wo-qubit operations) Update the single- (two-)qubit operation target register Sd (Td).
Q. Bundle	`[PI,] Q_Op [| Q_Op]*`	Applying operations on qubits after waiting for a small number of cycles indicated by `PI`.

eQASM, HPCA2019

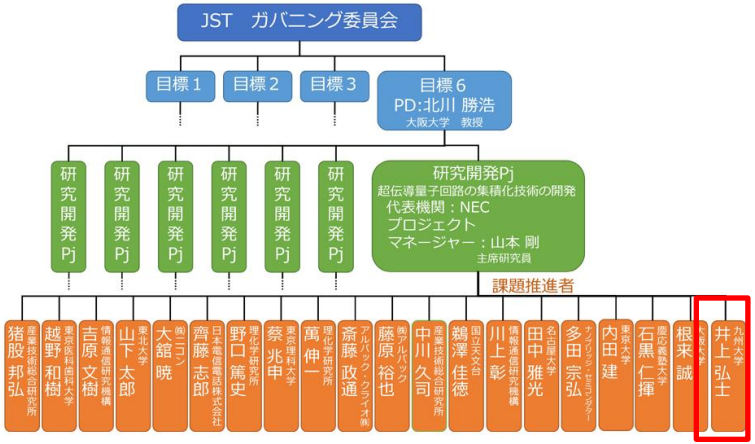


SFQ surface code decoder , ISCA2020

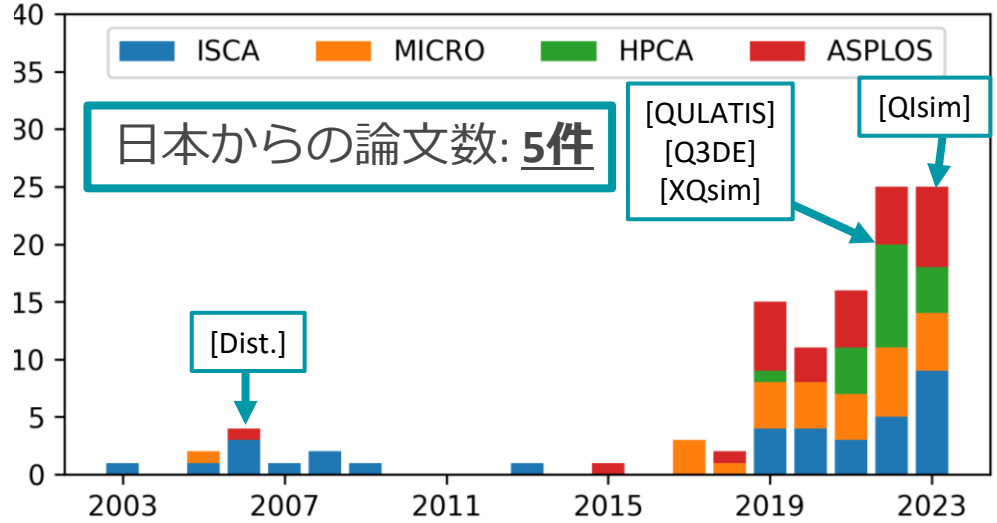
K. Bertels et al., “eQASM: An Executable Quantum Instruction Set Architecture,” in HPCA2019.
C. Duckering et al., “Virtualized Logical Qubits: A 2.5 D Architecture for Error-Corrected Quantum Computing,” in MICRO2020.
S. Tannu et al., “Taming the Instruction Bandwidth of Quantum Computers via Hardware-Managed Error Correction,” in MICRO2017.
A. Holmes et al., “NISQ+: boosting quantum computing power by approximating quantum error correction,” in ISCA2020.

量子計算機開発におけるアーキテクチャの知見

- 理論では考えられていないコンポーネントを詰める
  - 例: 誤り訂正符号のエラー推定が多項式時間の古典処理でできる
  - 誰がどこで解くのか、計算機全体の構成はどうなるのか
- 各要素技術を統合して考えて、性能見積もり、開発へのフィードバック
  - 例: 超伝導ムーンショットの井上先生チーム
- 経済的な制約
  - 例: $p = 10^{-5}$の量子ビット＋軽量誤り推定
    vs. $p = 10^{-3}$の量子ビット＋高精度誤り推定



国内外の量子計算機アーキテクチャ研究動向



アーキ系のトップ国際会議（ISCA, MICRO, HPCA, ASPLOS）
における量子計算機関連論文数（過去20年分）

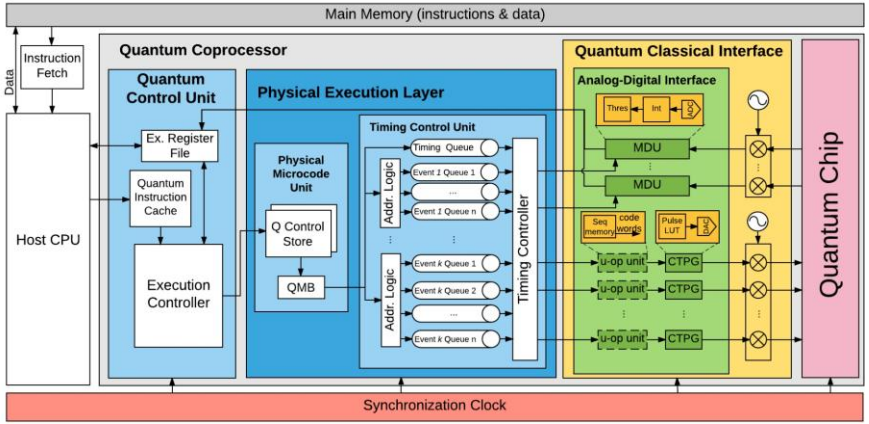
年	量子関連論文割合
2003~2018	0~1%程度
2019	5.4% (15/276)
2020	3.6% (11/308)
2021	5.0% (16/323)
2022	7.7% (25/325)
2023	6.1% (25/408)

トップ国際会議における
量子関連論文数割合

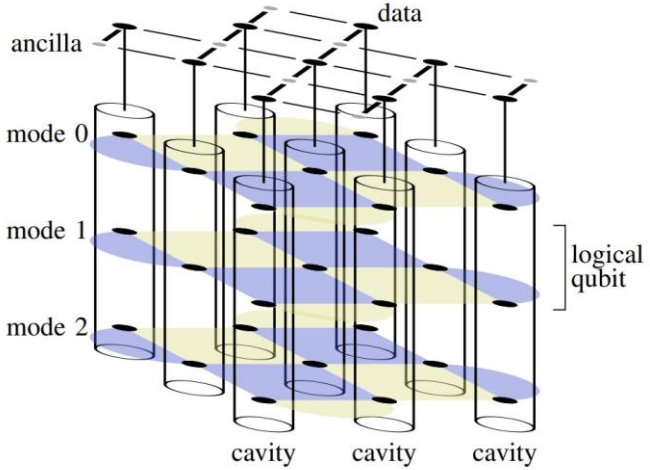
元データ: https://yuteno.github.io/ または上野のresearchmapで公開中

[Dist.] R. Van Meter, W. Munro, K. Nemoto, K. Itoh, “Distributed Arithmetic on a Quantum Multicomputer”, **ISCA2006**.
[QULATIS] **Y. Ueno**, M. Kondo, M. Tanaka, Y. Suzuki, Y. Tabuchi, “QULATIS: A Quantum Error Correction Methodology toward Lattice Surgery”, **HPCA2022**.
[Q3DE] Y. Suzuki, …, K. Inoue, T. Tanimoto, “Q3DE: A fault-tolerant quantum computer architecture for multi-bit burst errors by cosmic rays”, **MICRO2022**.
[XQsim] I. Byun, …, T. Tanimoto, M. Tanaka, K. Inoue, J. Kim, “XQsim: modeling cross-technology control processors for 10+K qubit quantum computers”, **ISCA2022**.
[QIsim] D. Min, …, M. Tanaka, K. Inoue, J. Kim, “QIsim: Architecting 10+K Qubit QC Interfaces Toward Quantum Supremacy”, **ISCA2023**.

量子計算機アーキテクチャ研究のインパクト



Quantum MicroArchitecture
MICRO'17 Best paper



Virtualized Logical Qubits
MICRO'20 Best paper runner-up

- 計算機アーキテクチャ分野でも量子計算機研究は評価される
- 物理（量子情報）+アーキテクチャの知見でインパクト大

X. Fu et al., "An Experimental Microarchitecture for a Superconducting Quantum Processor", MICRO 2017.
C. Duckering et al., "Virtualized Logical Qubits: A 2.5D Architecture for Error-Corrected Quantum Computing", MICRO 2020.

国内外の量子計算機アーキテクチャ研究動向

主要な研究グループ	論文数（割合）	First quantum paper in top conferences
University of Chicago (Fred Chong (@UCSB until 2015))	33本 (30.0%)	ISCA2003
Georgia Tech. (Moinuddin Qureshi, Swamit Tannu (UW-Madison))	14+4 (16.4%)	MICRO2017
Princeton University (Margaret Martonosi)	14 (12.7%) (内Chicagoと共同6)	ISCA2007
UC Santa Barbara (Yuan Xie, Yufei Ding)	7 (6.3%)	ASPLOS2019

- 65%の論文が上位4グループから出ている

各グループの初期の量子関連論文

Chicago University
- [初期] M. Oskin, F. Chong, I. Chuang, J. Kubiatowicz,
Building Quantum Wires: The Long and the Short of it, ISCA2003. (arXiv2001.06598)
- [最近] A. Litteken, L. Seifert, J. Chadwick, N. Nottingham, F. Chong, J. Baker,
Qompress: Efficient Compilation for Ququarts Exploiting Partial and Mixed Radix Operations for Communication Reduction, ASPLOS2023.

Georgia Tech.
- [初期] P. Das, C. Pattison, S. Manne, D. Carmean, K. Svore, M. Qureshi, N. Delfosse,
AFS: Accurate, Fast, and Scalable Error-Decoding for Fault-Tolerant Quantum Computers, HPCA2022. (arXiv2001.06598)
- [最近] S. Vittal, P. Das, M. Qureshi, Astrea: Accurate Quantum Error-Decoding via Practical Minimum-Weight Perfect-Matching, ISCA2023.

最初は「**アーキわかる（興味ある）物理研究者**」と併走
その過程で「**物理わかるアーキ研究者**」を育成

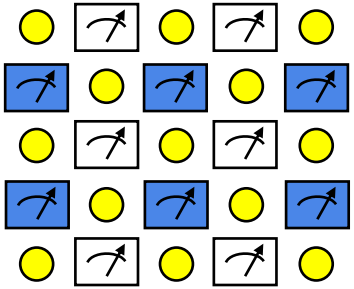
量子計算機アーキテクチャ研究動向まとめ

- 計算機アーキテクチャ
  ＝計算機の中身、全体の構成、取り巻く環境
  - 主な役割：各要素技術の統合、計算機としての展望を示す
- 計算機アーキテクチャ分野でも量子計算機はインパクト大
  - 物理（量子情報）系＋計算機系研究者の共同研究が重要
- 海外では量子計算機アーキテクチャの研究が盛ん、
  国内はまだこれから

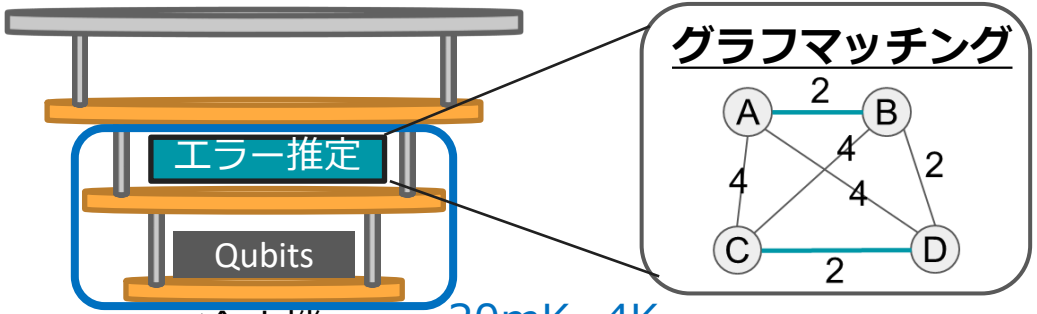
発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

1コマ目の概要



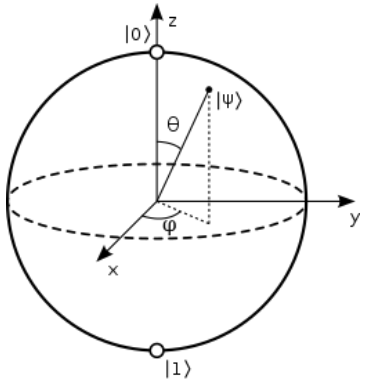
表面符号



冷凍機　20mK~4K

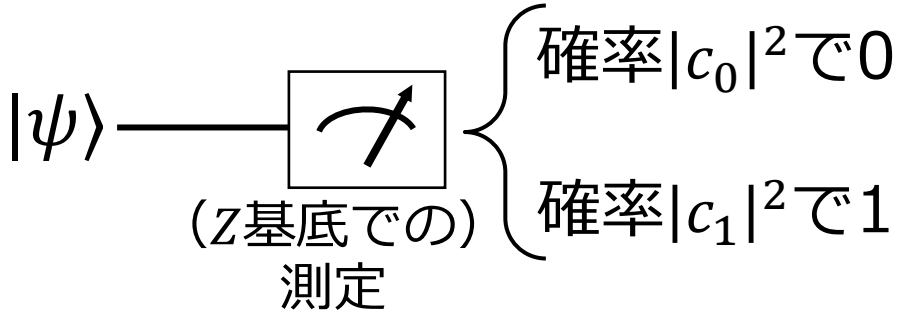
- 量子ビットはエラーまみれなのでエラーを訂正しながら計算をする必要あり
- エラーを直接観測はできないがエラーのパリティチェックはできる
  - 量子ビットには2種類のエラーが生じる
  - 量子誤り訂正符号（**表面符号**）
    ＝符号化用量子ビット ● ＋2種類のパリティチェック用量子ビット
- 量子誤り訂正＝パリティ情報から符号化用量子ビットの
  エラーを推定＝**グラフマッチング**問題に帰着できる
- 上野の博論：冷凍機内で低消費電力・高速にグラフマッチング

量子計算の基礎



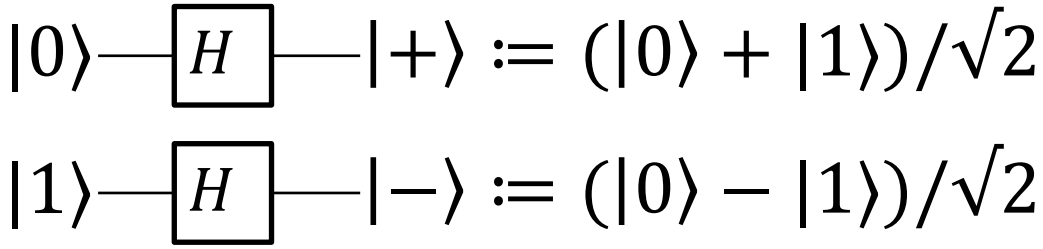
Bloch球

- 量子ビット：$|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ （c_0, c_1は複素数）

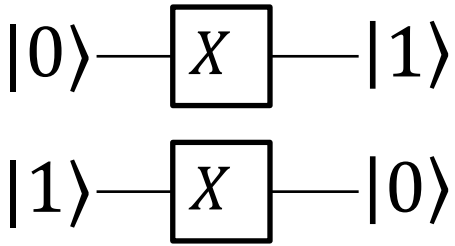


量子計算の基礎

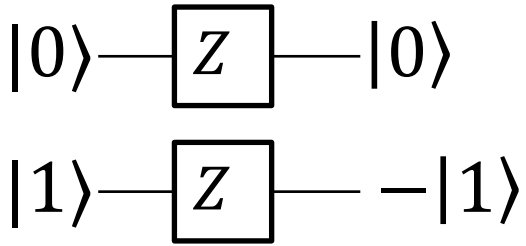
Hadamardゲート (H)



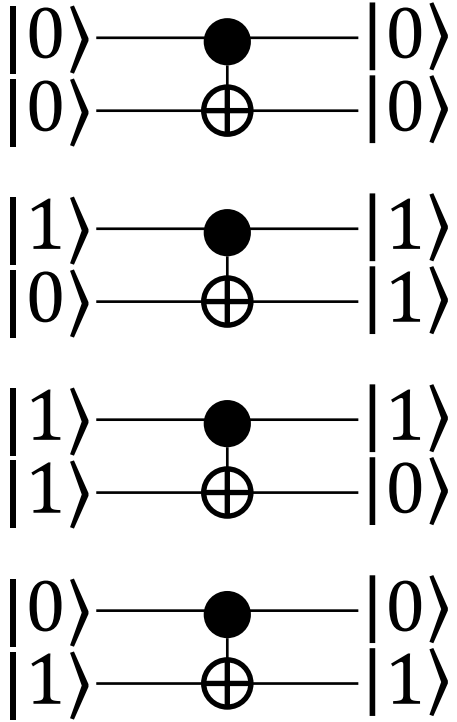
Xゲート



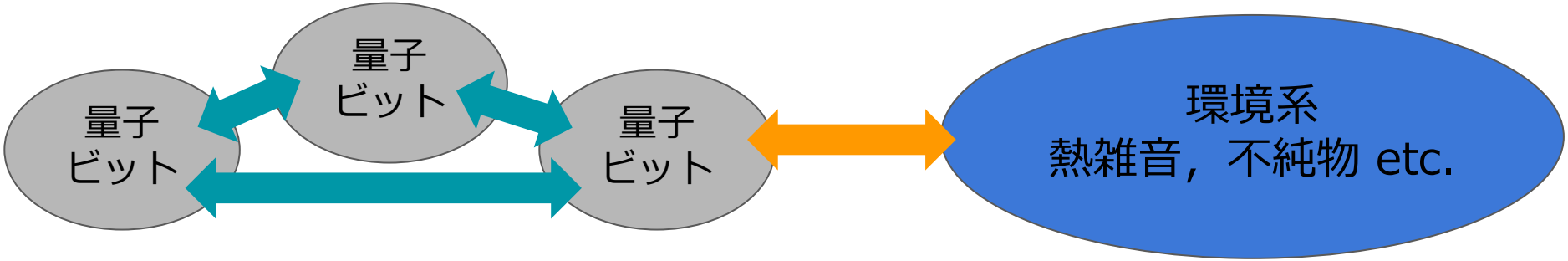
Zゲート



$CNOT$ゲート



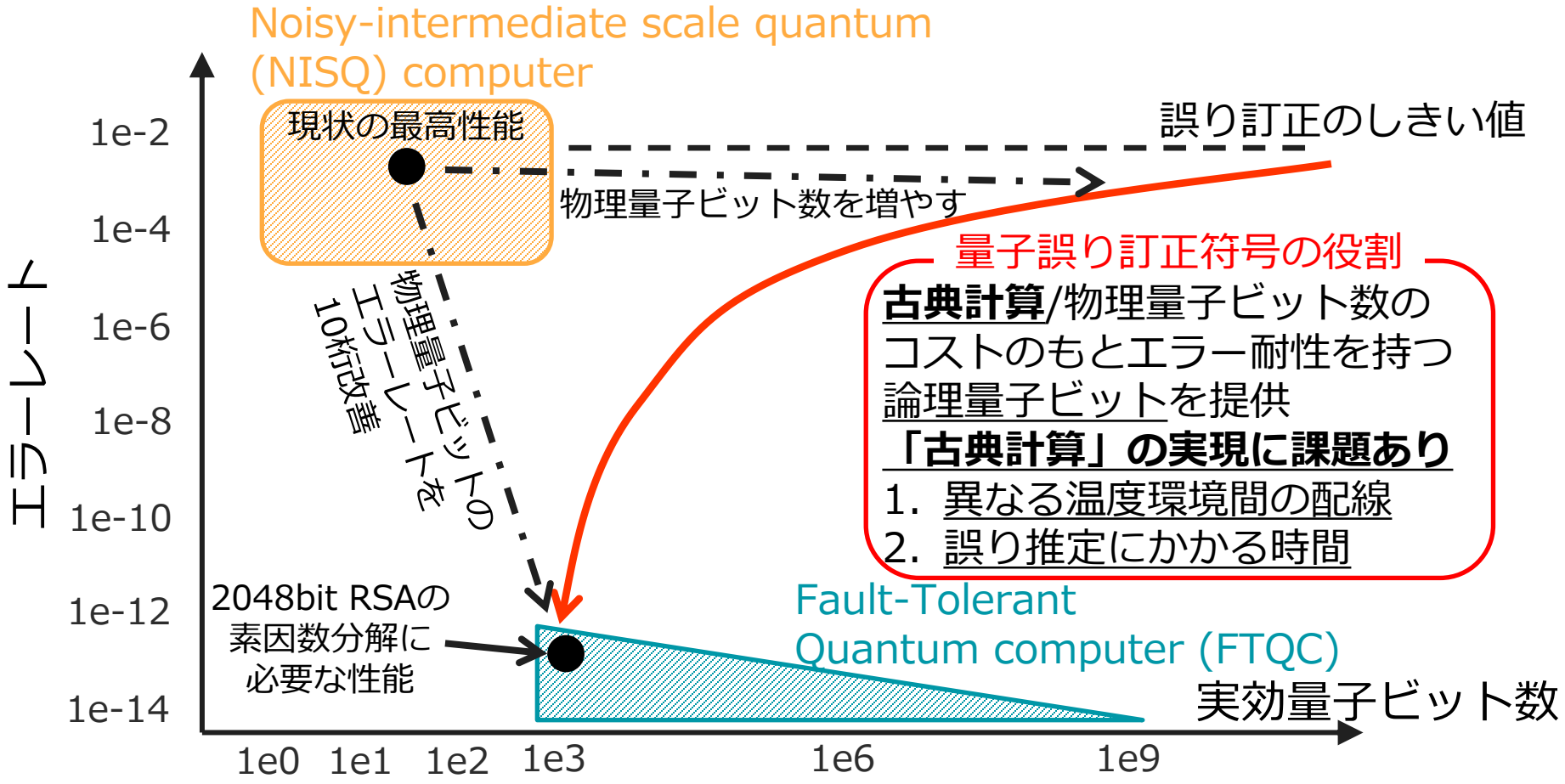
実現における障壁: エラー耐性



エラー発生原因: 量子ビットと環境系と意図しないエンタングルメント

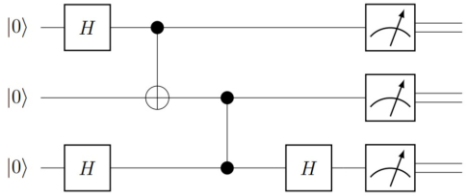
- 量子コンピュータはエラー発生確率が非常に高い
  - 1回の操作で0.1～1%程度
  - 例: 0.1%でも1000ステップだと計算失敗確率は$1-0.999^{1000}=0.63$
- Xエラー（ビット反転）とZエラー（位相）に分解できる
- **量子誤り訂正符号**という枠組みが提案されている

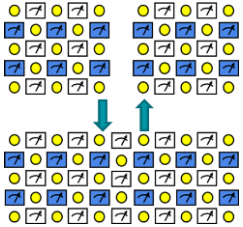
量子コンピュータの発展と量子誤り訂正の役割

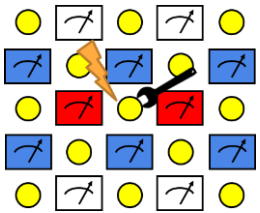


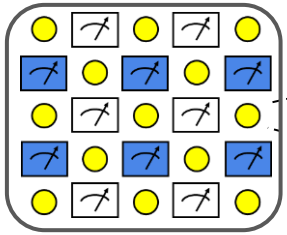
誤り耐性量子計算（FTQC）の主な構成要素

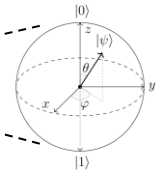
デジタル量子計算 ⇄ 論理命令 / 論理測定 ⇄ 誤り訂正プロセス ⇄ 物理命令 / 物理測定 ⇄ 超伝導量子プロセッサ







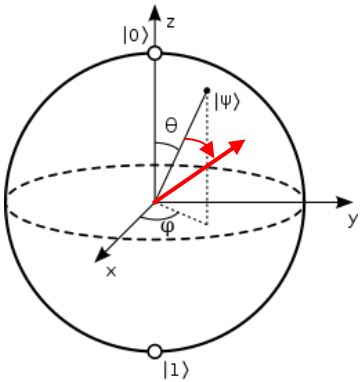


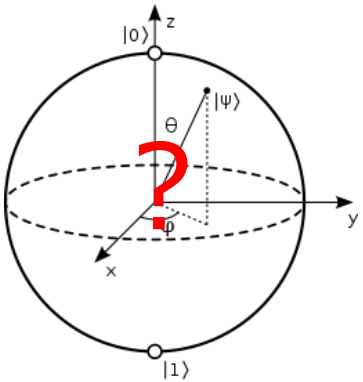


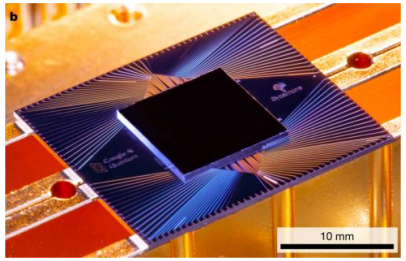
	量子回路	格子手術による論理演算	**表面符号の復号**	表面符号による符号化	物理量子ビット
要素技術	素因数分解 量子化学計算	魔法状態蒸留回路	Blossom alg. $(O(n^3))$ 機械学習手法	高効率な符号化 2.5D アーキテクチャ	100+ qubitsマシン

- 「いかにして効率的な誤り訂正プロセスを実現するか」が誤り耐性量子計算の実現性の決め手

量子誤り訂正符号に求められる条件







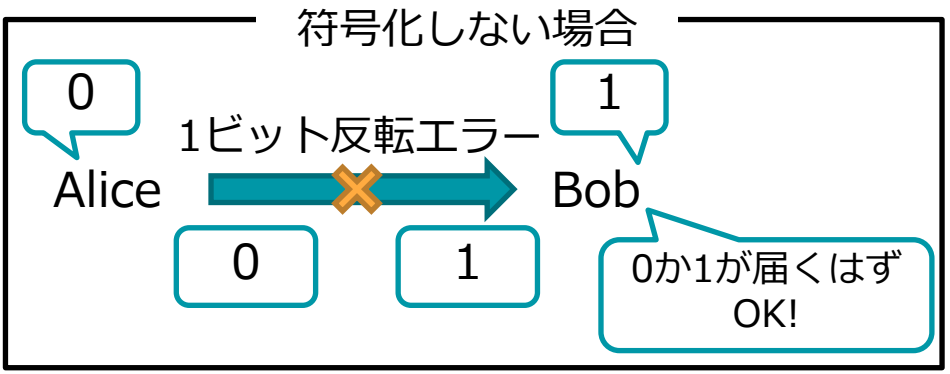
[3] より

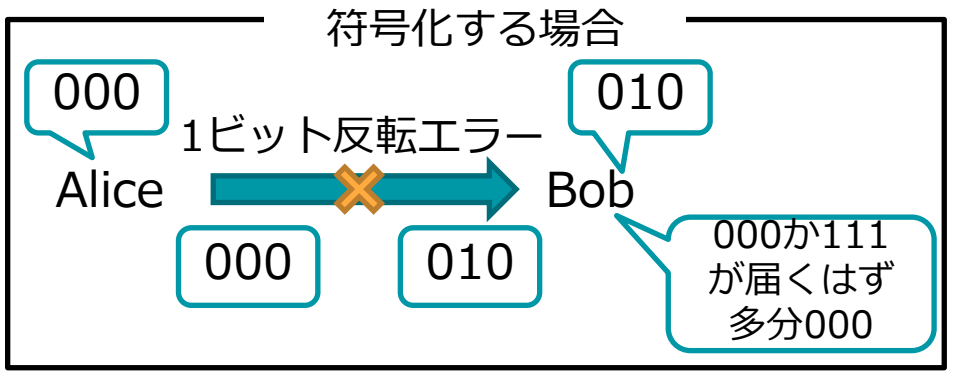
- 量子ビット数: 100~1000
- エラーレート: 0.1~1%

- 連続的な情報に生じたエラーを訂正できる
- 直接状態を観測することなく訂正できる
- 実験的に達成できる誤り率を超えるしきい値を持つ

[3] Frank Arute, Kunal Arya, Ryan Babbush, et al. Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510 (2019).

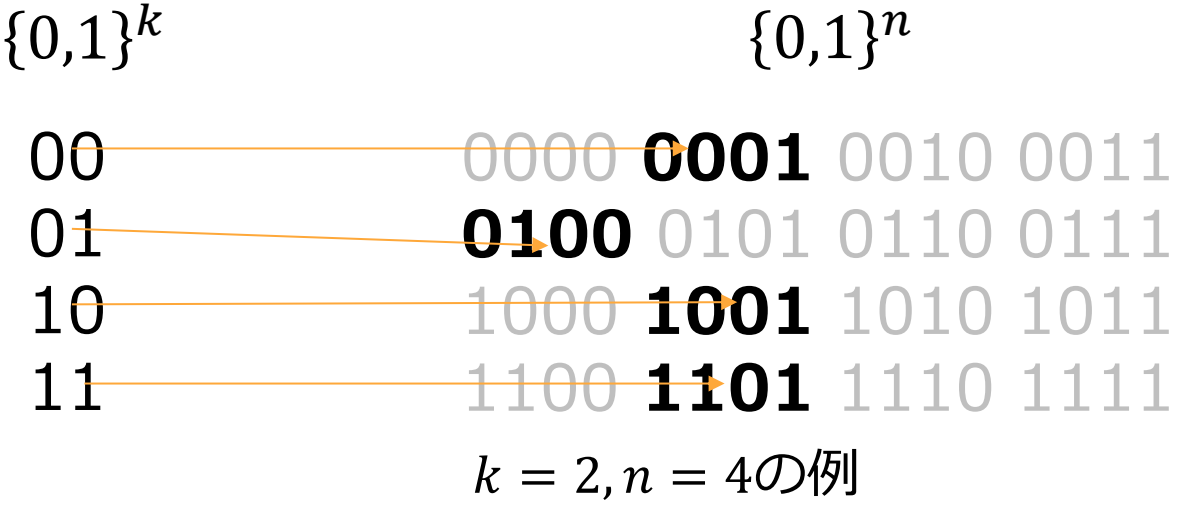
誤り訂正符号の役割





- 誤り訂正符号: 元データを冗長に表現して
  誤りの検出・訂正を可能にする符号
- 符号パラメータ$[n, k, d]$
  - n: 符号長（どの程度冗長に表現するか）
  - k: 情報数（元データのビット数）
  - d: 符号距離（論理操作を行う際に触る必要のある最低ビット数）
    - $\lfloor (d-1)/2 \rfloor$ビットまでは訂正可能、$d-1$個までは検出可能

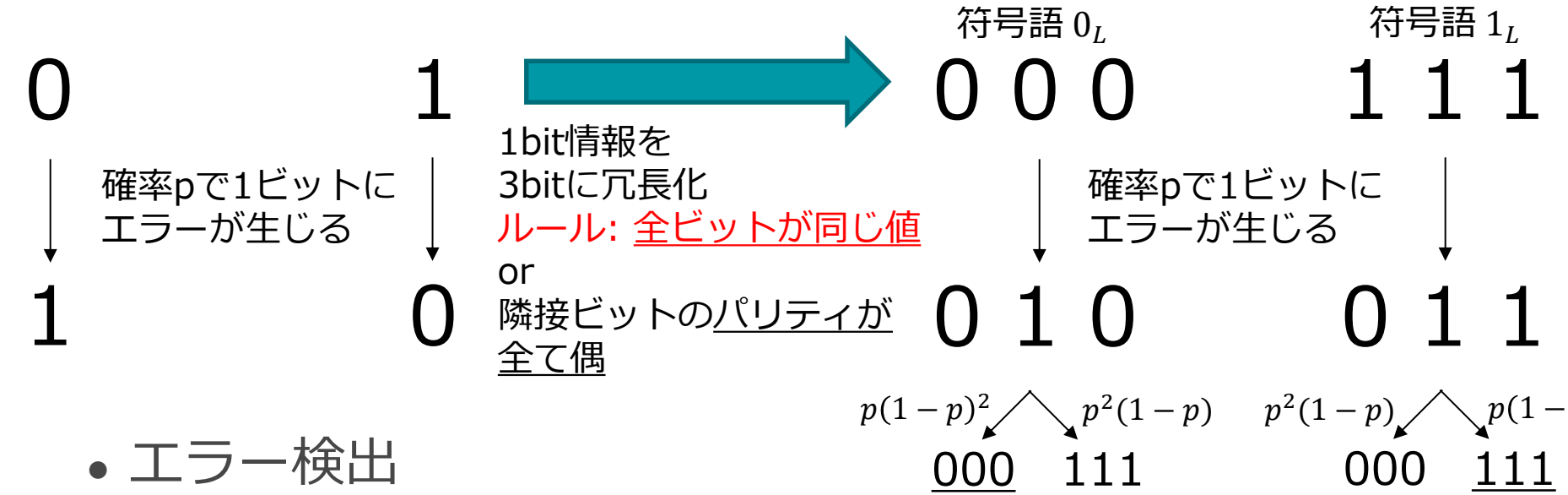
符号の作り方



適当に符号語に
割り当てると対応表が
膨大に

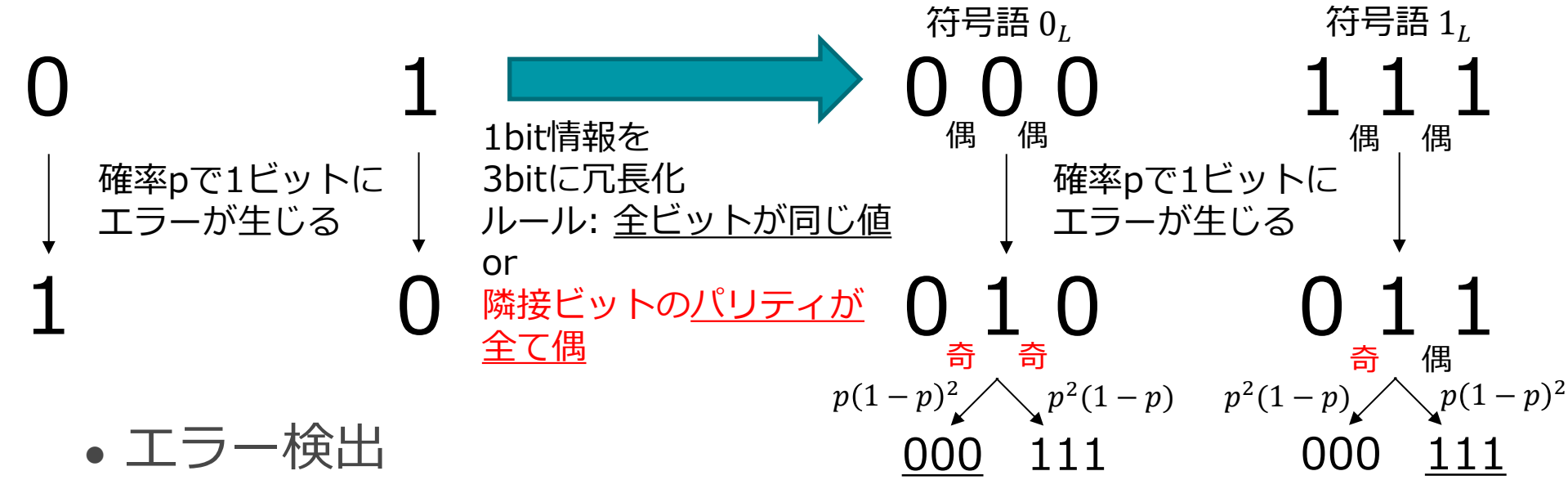
- k次元空間の各点をn次元空間の一部（**符号空間**）に割当
- 割当の際は何かしらのルールが必要
  - 特に量子版は重ね合わせの連続値を対応させる必要があるので、
    適当な割当は不可能

誤り訂正符号 古典の場合



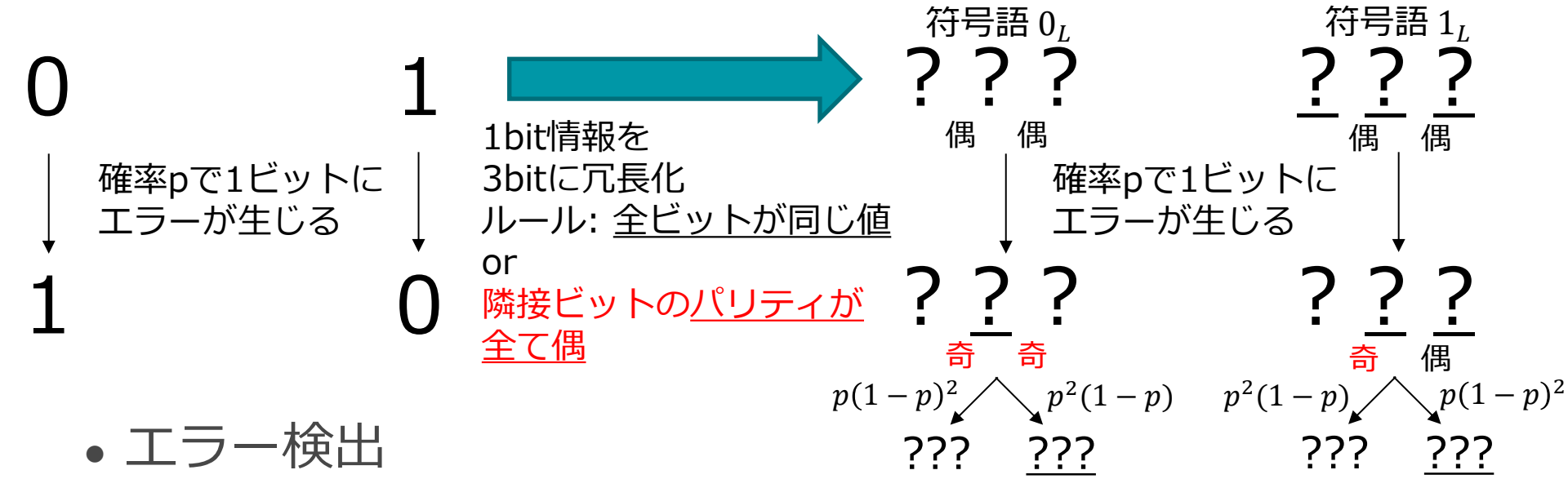
- エラー検出
  - 符号を構成する各ビットをチェックしてすべて一致しているかどうか
  - 隣接ビットのパリティがすべて偶であるか <- 量子でも可能なアプローチ
- エラー訂正（復号）
  - パリティ検査の結果から最も近い（確率の高い）符号語を特定

誤り訂正符号 古典の場合



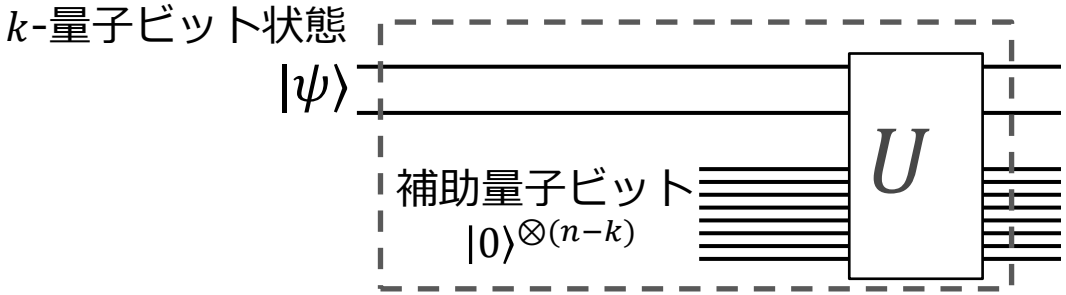
- エラー検出
  - 符号を構成する各ビットをチェックしてすべて一致しているかどうか
  - 隣接ビットのパリティがすべて偶であるか <- 量子でも可能なアプローチ
- エラー訂正（復号）
  - パリティ検査の結果から最も近い（確率の高い）符号語を特定

誤り訂正符号 古典の場合



- エラー検出
  - 符号を構成する各ビットをチェックしてすべて一致しているかどうか
  - 隣接ビットのパリティがすべて偶であるか <- 量子でも可能なアプローチ
- エラー訂正（復号）
  - パリティ検査の結果から最も近い（確率の高い）符号語を特定

量子誤り訂正符号の作り方



符号化操作 enc
補助量子ビットをつけて
ユニタリ行列で符号化

n-量子ビット状態

$$\mathrm{enc}(|\psi\rangle) := U|\psi\rangle|0\rangle^{\otimes(n-k)}$$

元のk-qubit空間

$$|\psi\rangle = \sum_{x=0}^{2^k-1} \alpha_x |x\rangle$$

符号空間

$$|\psi_L\rangle = \sum_{x=0}^{2^k-1} \alpha_x \mathrm{enc}(|x\rangle)$$

- ユニタリ行列Uを選ぶと符号の性質が決まる
  - シンドローム測定（=パリティチェック）が簡単
  - 符号空間中の論理操作が簡単
  - 望まない論理操作（=論理エラー）が起きにくい

} 満たしてほしい性質

ユニタリ行列Uの選び方

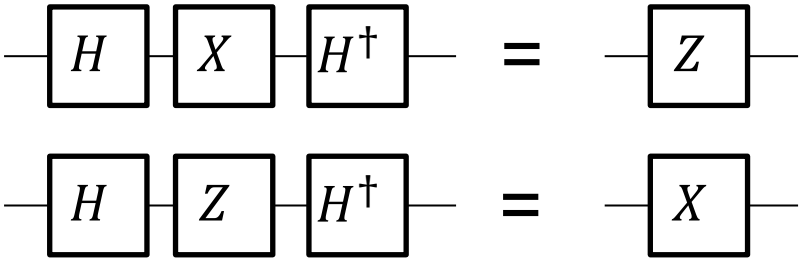
- Uを適切なClifford行列Cとするのが良いとされている
  - スタビライザ符号と呼ばれる
  - Surface code, Toric code, Color codeもスタビライザ符号の一種

Clifford行列C: 任意のPauli行列Pについて、
$CPC^\dagger$がPauli行列となるユニタリ演算

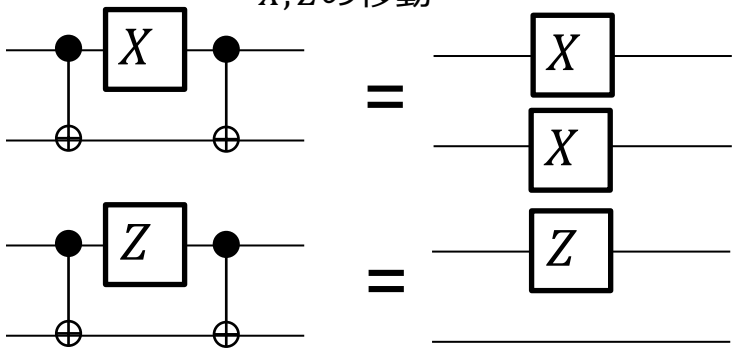
例: Hadamard, S, CNOTなど

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \quad S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix} \quad \mathrm{CNOT} = \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}$$

X, Zの入れ替え

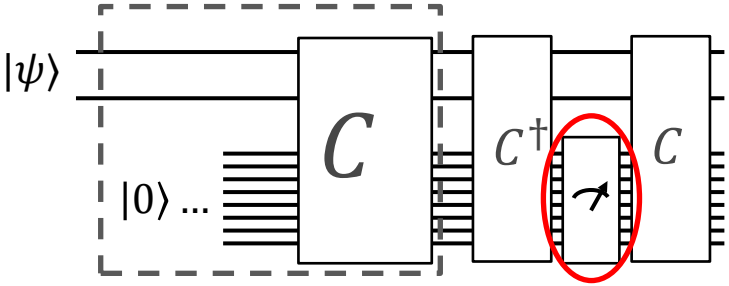


X, Zの移動



スタビライザ符号の性質

シンドローム測定



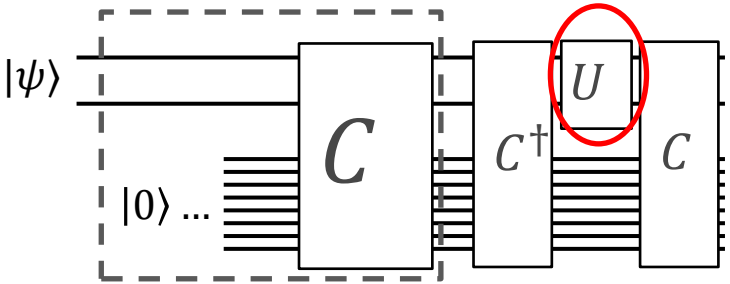
符号空間かチェック

Z測定が$CZC^\dagger$（Pauli測定）に変化

比較的容易

<u>測定値からエラー検出・訂正が可能</u>

論理演算



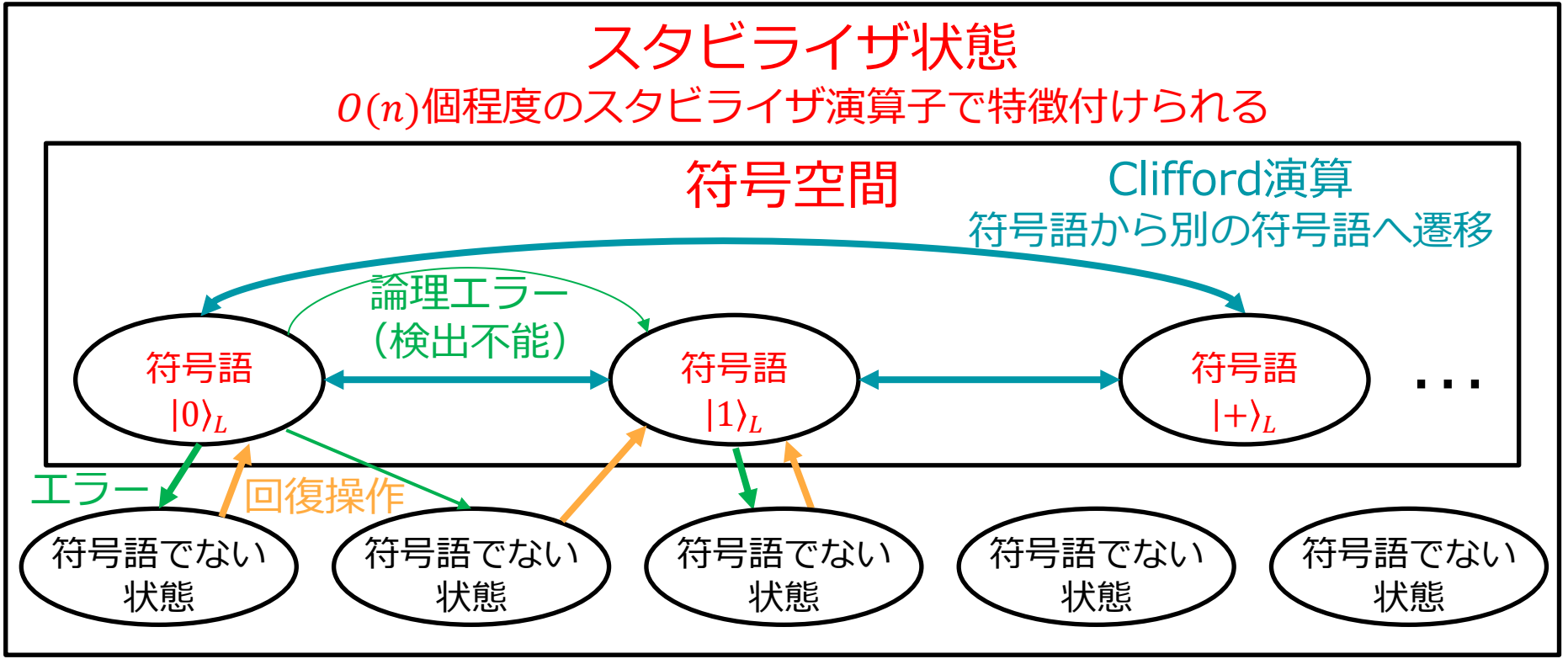
符号空間上での操作

Uが$C^\dagger UC$に変化

UがPauliかCliffordなら容易

誤り耐性量子計算に使う空間

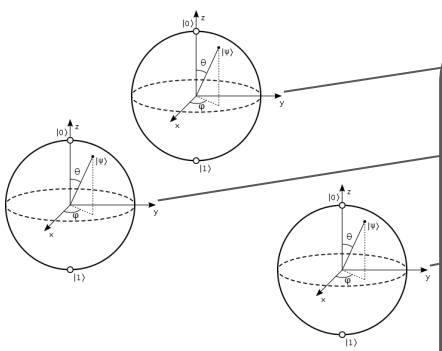
純粋状態 $|\psi\rangle = \sum_i^{2^n} \alpha_i |i\rangle$



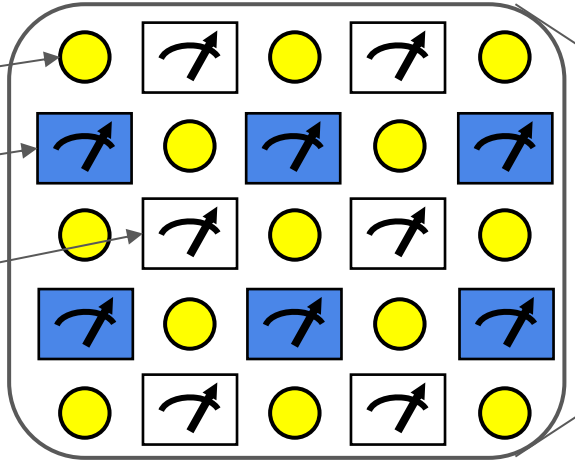
発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

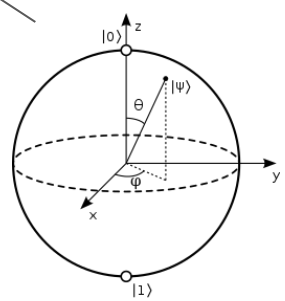
量子誤り訂正符号: 表面符号



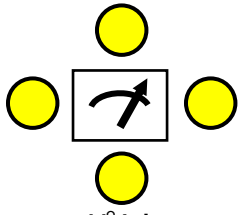
物理量子ビット
（超伝導、イオンなど）



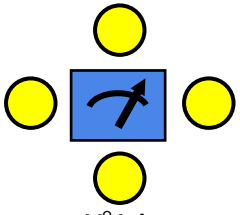
表面符号（**符号化距離** $d = 3$）



論理量子ビット



Zパリティ
チェック



Xパリティ
チェック

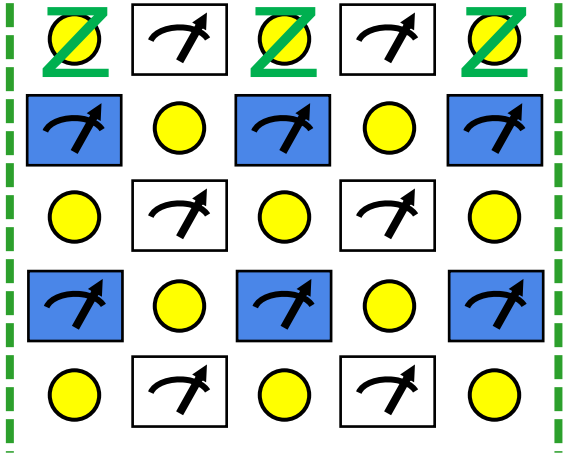
データ量子ビット

補助量子ビット
（Xスタビライザ）

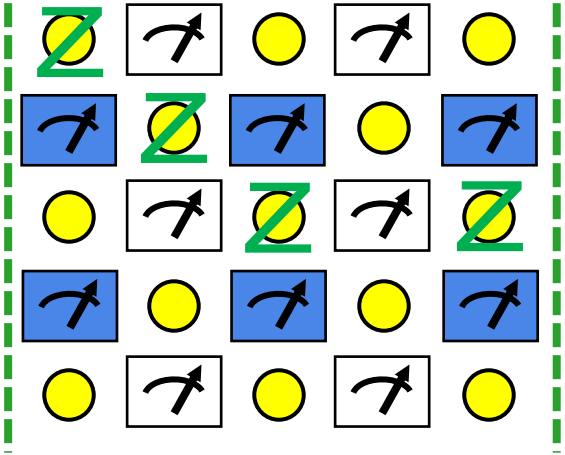
補助量子ビット
（Zスタビライザ）

- 規則的に並んだ複数の量子ビットで論理量子ビットを表現
  - 論理ビットの状態を表す**データ量子ビット**と観測用の**補助量子ビット**
- 補助量子ビットの観測値: XとZの2種類のエラーのうち、対応するもののパリティチェック

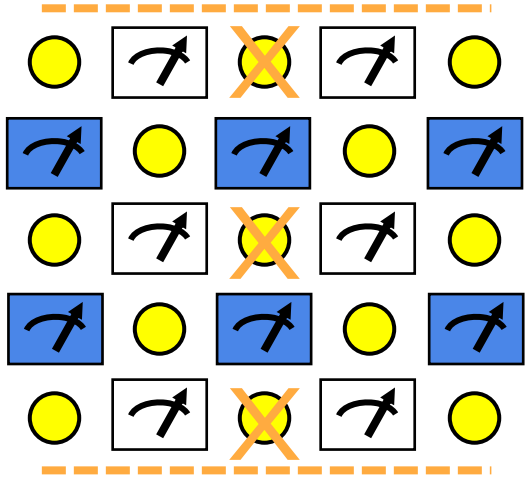
表面符号における論理Pauli演算



論理Z演算の一例



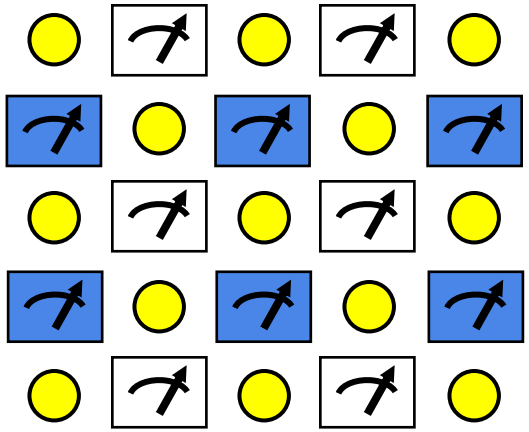
論理Z演算の別の例



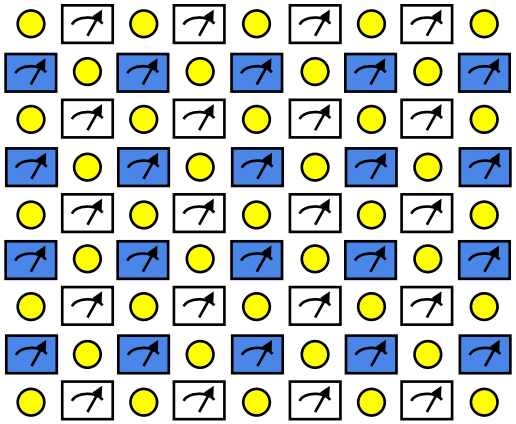
論理X演算の一例

- 論理Z演算：左右の境界をつなぐようにデータ量子ビットにZ演算を作用
- 論理X演算：上下の境界をつなぐようにデータ量子ビットにX演算を作用
- 表面符号の左右、上下の境界は意味が異なる

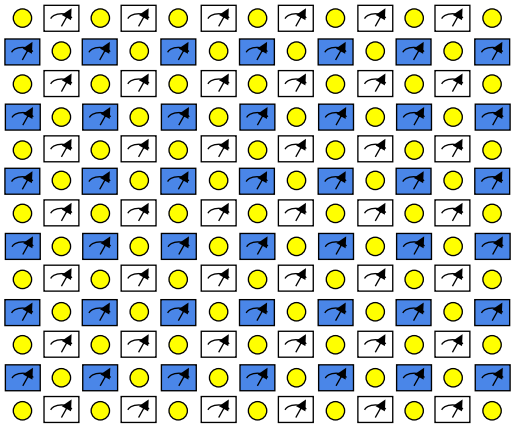
表面符号の符号距離



符号距離$d = 3$
データビット：$3^2 + 2^2 = 13$
補助ビット：$2 \times 3 \times (3 - 1) = 12$



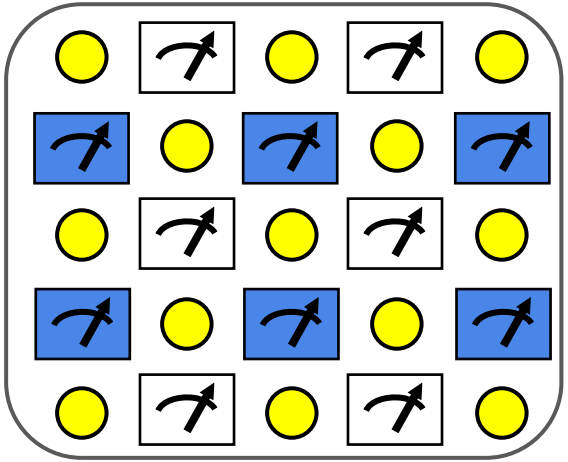
符号距離$d = 5$
データビット：41
補助ビット：40

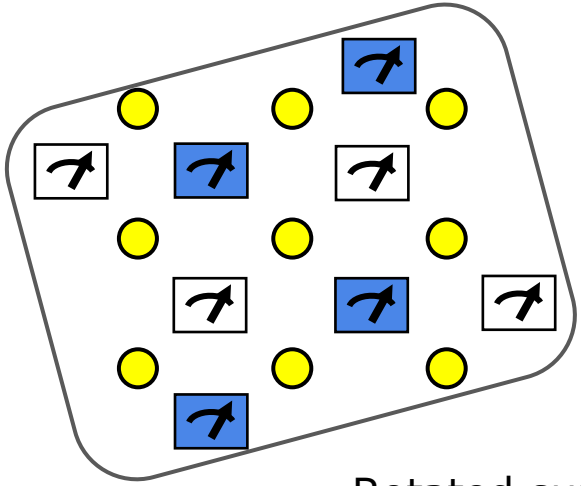


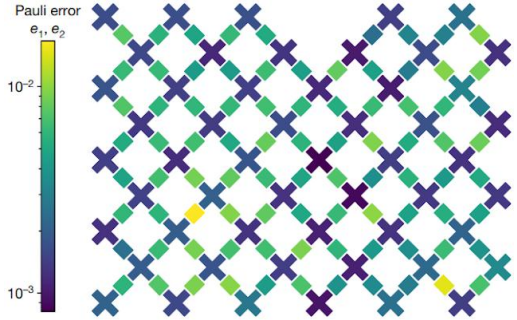
符号距離$d = 7$
データビット：85
補助ビット：84

- 符号距離d：論理操作をするのに触る必要があるデータ量子ビットの数
- 表面符号の構成に必要な量子ビット数は$O(d^2)$
- 1ビットエラー率p場合、望まない論理操作が起きる確率は$O(p^d)$
- 多項式的なオーバーヘッドで指数的な論理エラー率の低減

補足: Rotated surface code







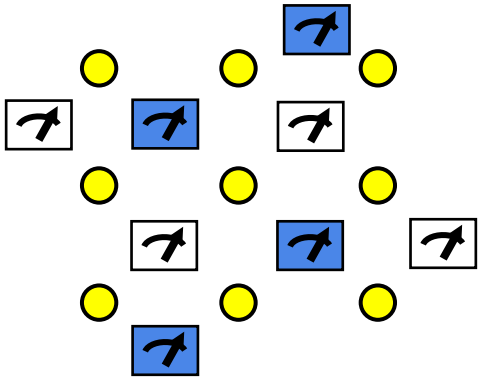
Cf. Google Sycamoreの配置

通常の表面符号（$d = 3$）
データ量子ビット: $d^2 + (d-1)^2$
補助量子ビット: $d(d-1) \times 2$
符号パラメータ: $[d^2 + (d-1)^2, 1, d]$

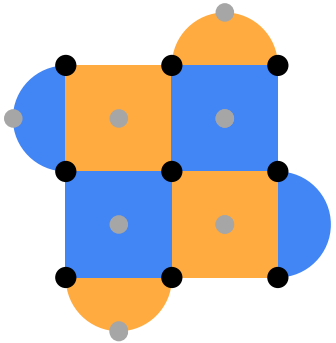
Rotated surface code（$d = 3$）
データ量子ビット: d^2
補助量子ビット: $(\frac{(d-1)^2}{2} + d - 1) \times 2$
符号パラメータ $[d^2, 1, d]$

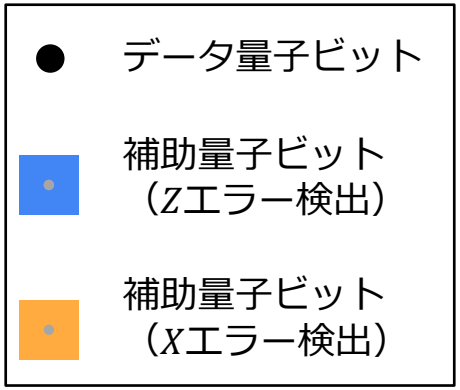
- 表面符号を45度回転させると符号距離を保ったまま物理量子ビット数を削減できる

補足: Rotated surface codeの表記

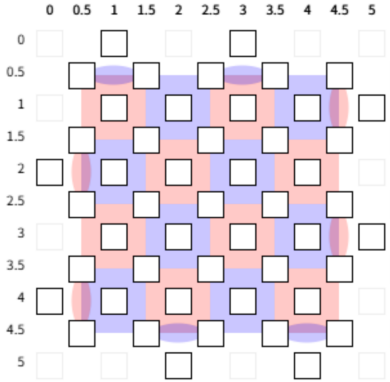


本資料前半での表記

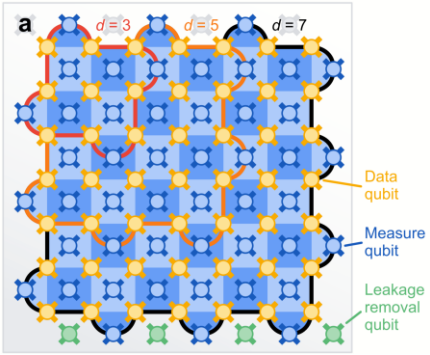




本資料後半での表記

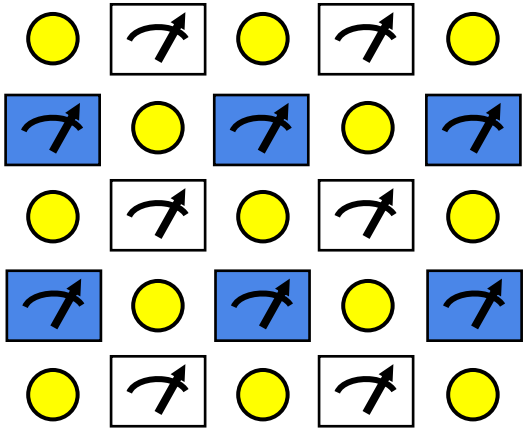


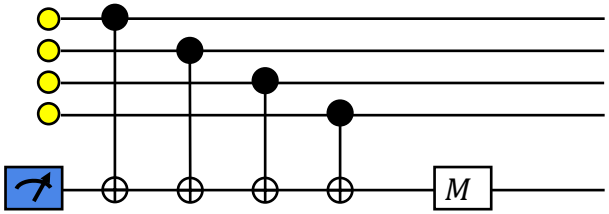
Crumble（後述）での表記 ($d = 5$)



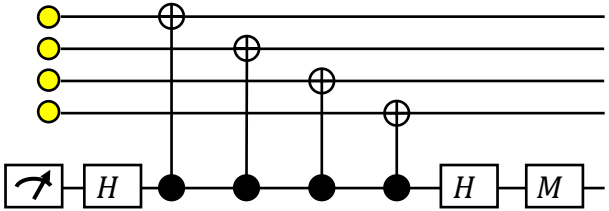
Google論文での表記
(From arXiv2408.13687)

表面符号におけるパリティチェック



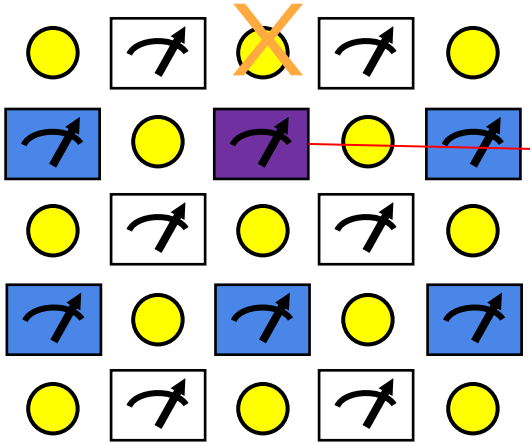


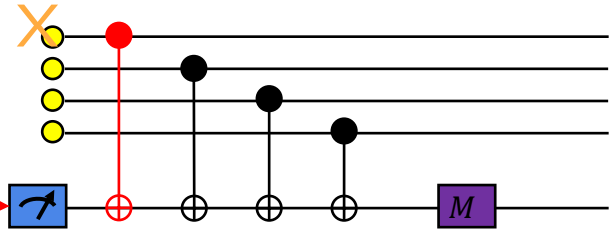
Zスタビライザ測定
（Xエラーパリティチェック）



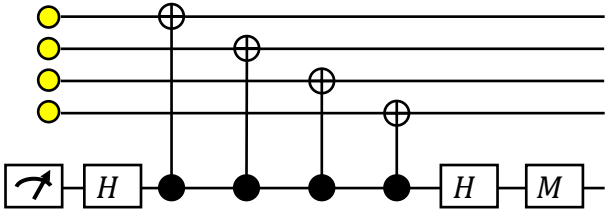
Xスタビライザ測定
（Zエラーパリティチェック）

表面符号におけるパリティチェック



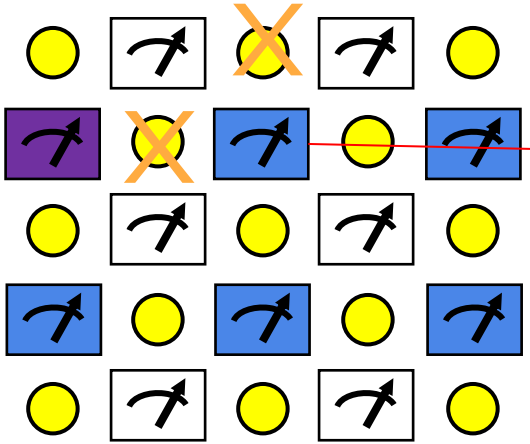


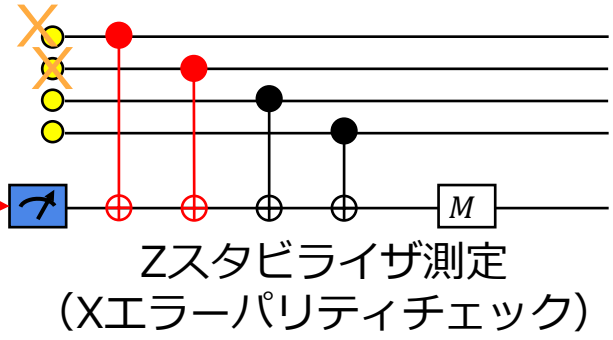
Zスタビライザ測定
（Xエラーパリティチェック）



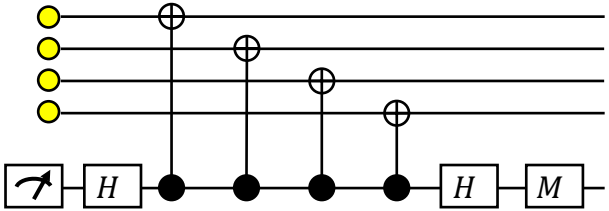
Xスタビライザ測定
（Zエラーパリティチェック）

表面符号におけるパリティチェック



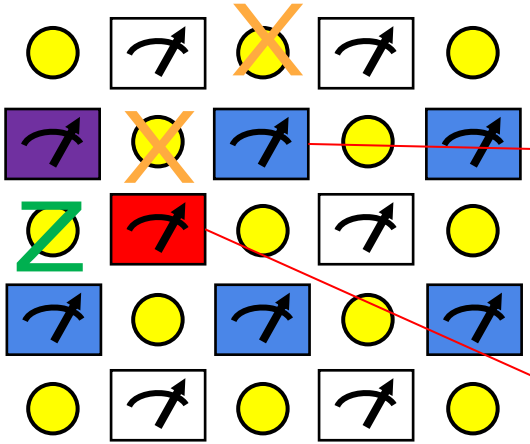


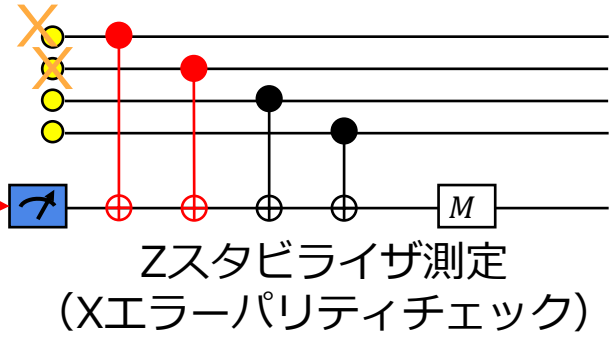
Zスタビライザ測定
（Xエラーパリティチェック）



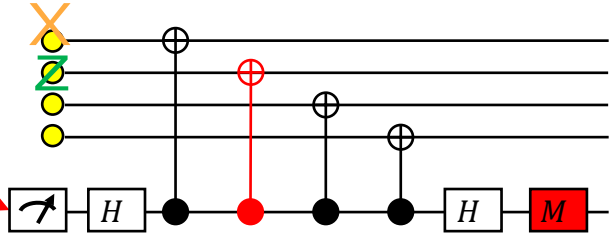
Xスタビライザ測定
（Zエラーパリティチェック）

表面符号におけるパリティチェック



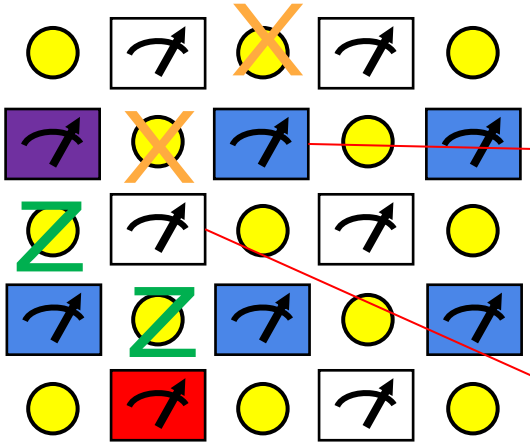


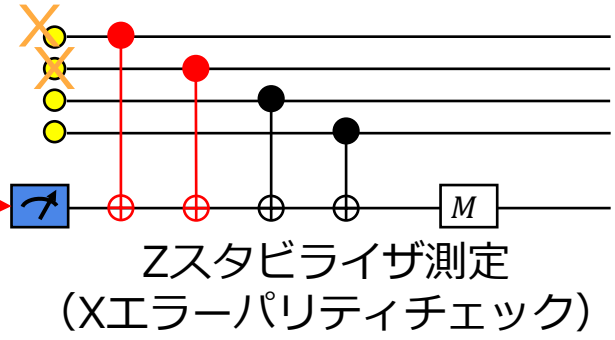
Zスタビライザ測定
（Xエラーパリティチェック）



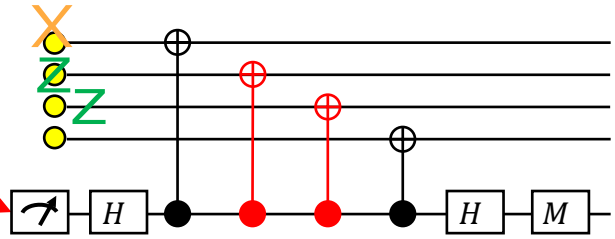
Xスタビライザ測定
（Zエラーパリティチェック）

表面符号におけるパリティチェック



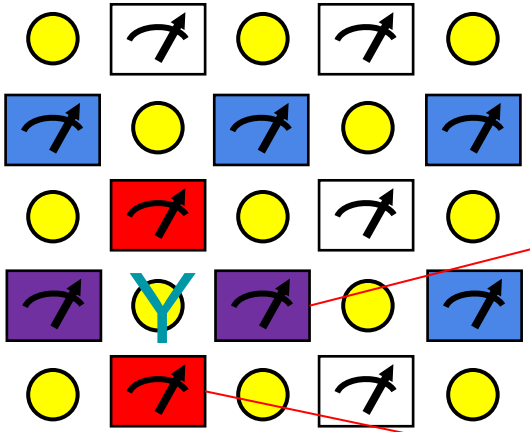


Zスタビライザ測定
（Xエラーパリティチェック）

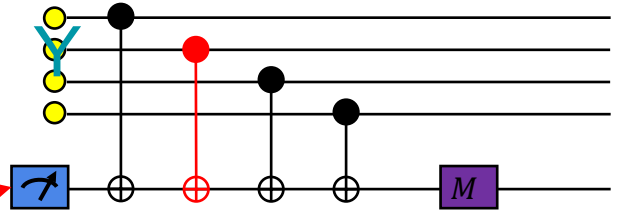


Xスタビライザ測定
（Zエラーパリティチェック）

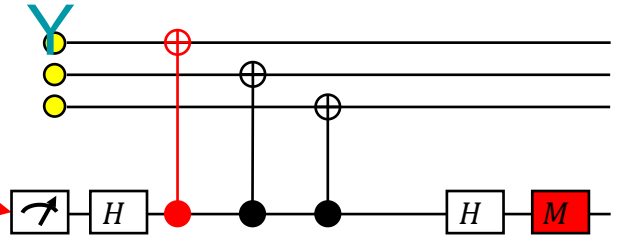
表面符号におけるパリティチェック



$Y = iXZ$なので
XとZが同時に生じているとみなせる

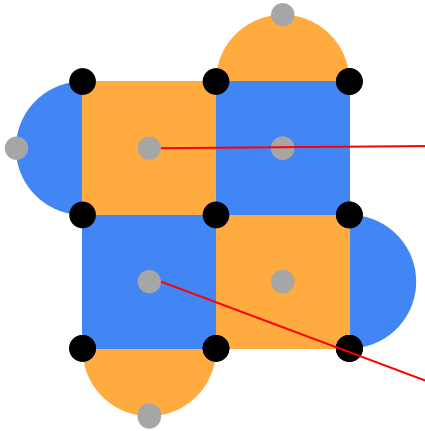


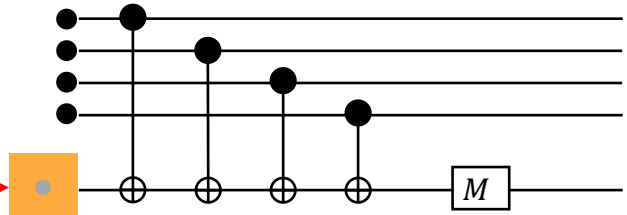
Zスタビライザ測定
（Xエラーパリティチェック）



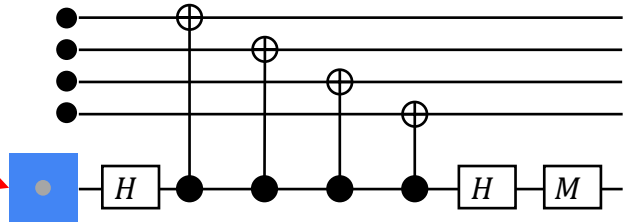
Xスタビライザ測定
（Zエラーパリティチェック）

補足：回転表面符号におけるパリティチェック





Zスタビライザ測定
（Xエラーパリティチェック）



Xスタビライザ測定
（Zエラーパリティチェック）

古典・量子誤り訂正符号の対比

古典

0 0 0
偶 偶

符号語 0_L

0 0 1
偶 奇

符号語でない

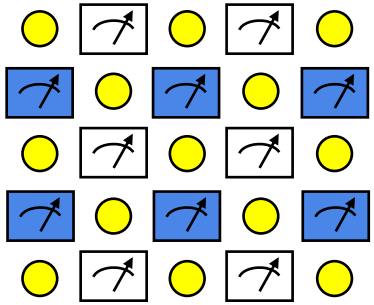
0 1 1
奇 偶

符号語でない

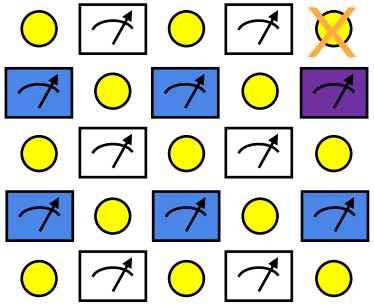
1 1 1
偶 偶

符号語 1_L

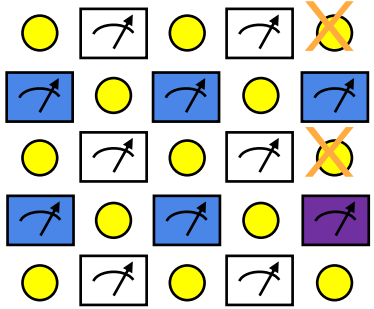
量子 (表面符号)



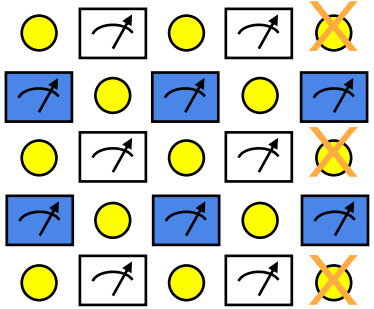
符号語 $|0\rangle_L$



符号語でない

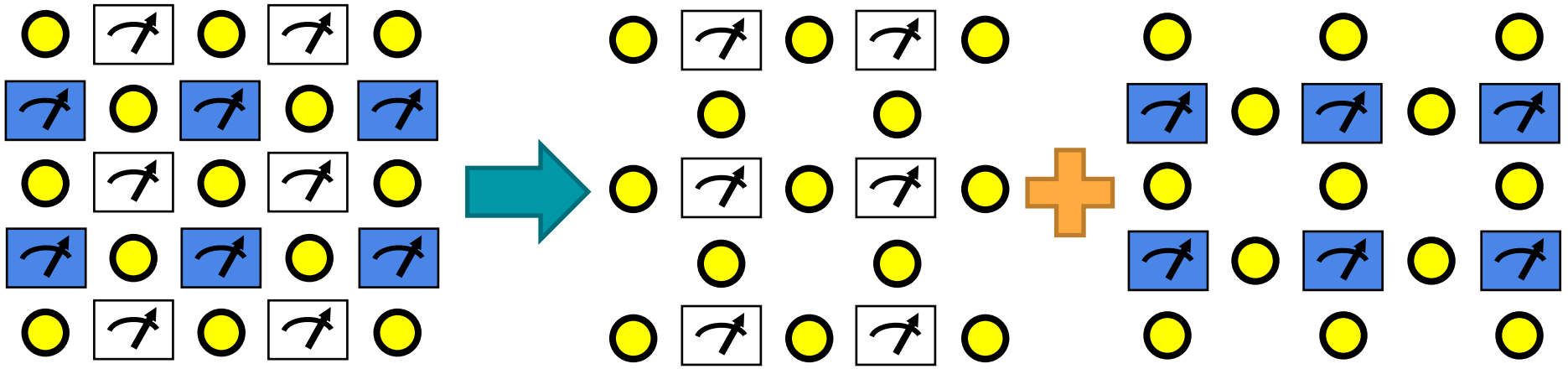


符号語でない



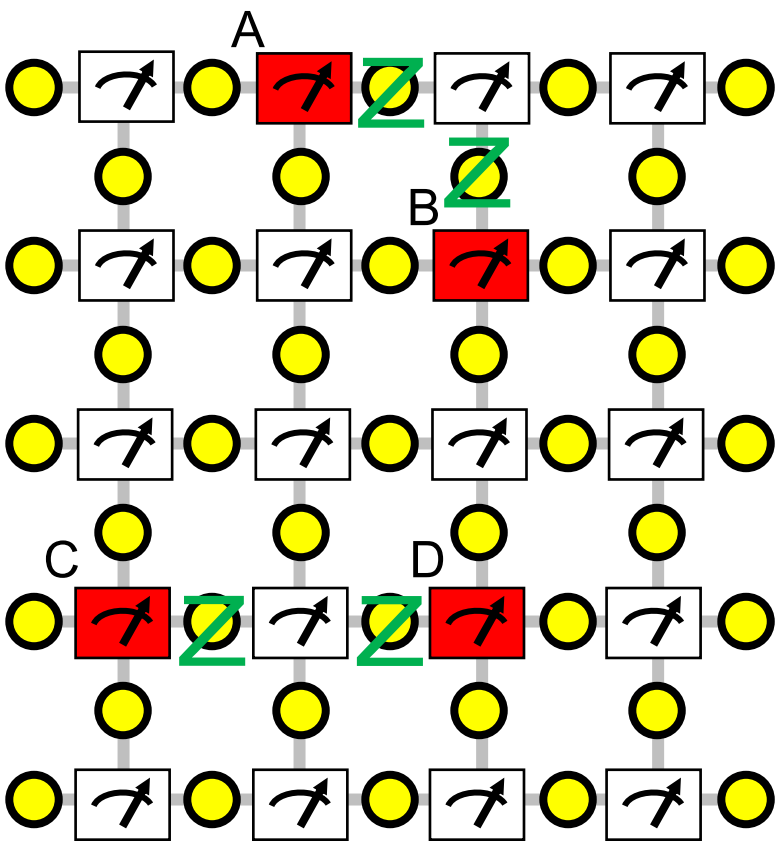
符号語 $|1\rangle_L$

表面符号におけるエラー推定（復号）



- XとZのそれぞれのエラーを独立に推定できる
  - マッチング問題に帰着する場合はXとZの格子を独立に復号
- XとZに相関がある場合、その情報を使えない

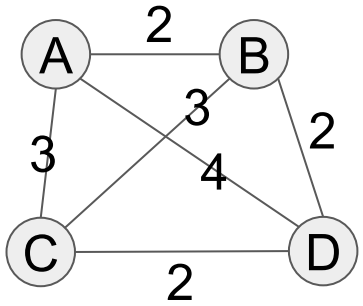
表面符号におけるエラー推定（復号）



仮定

- XとZのエラーは独立に推定できる
- なるべく短いエラー鎖が生じる

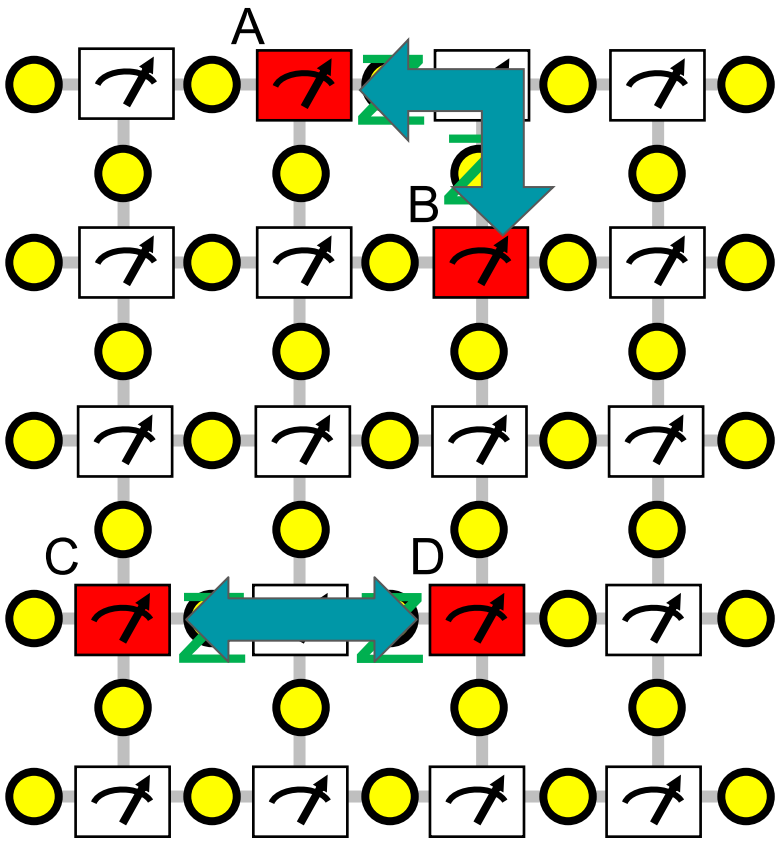
Minimum Weight Perfect Matching (MWPM)



V: Hot syndromes
W_e: Manhattan distance

Exact solution: **Blossom algorithm ($O(n^3)$)**

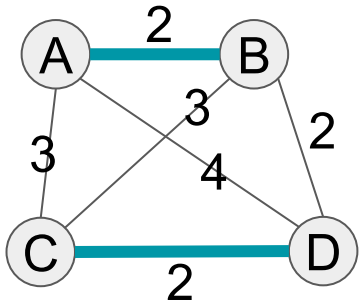
表面符号におけるエラー推定（復号）



仮定

- XとZのエラーは独立に推定できる
- なるべく短いエラー鎖が生じる

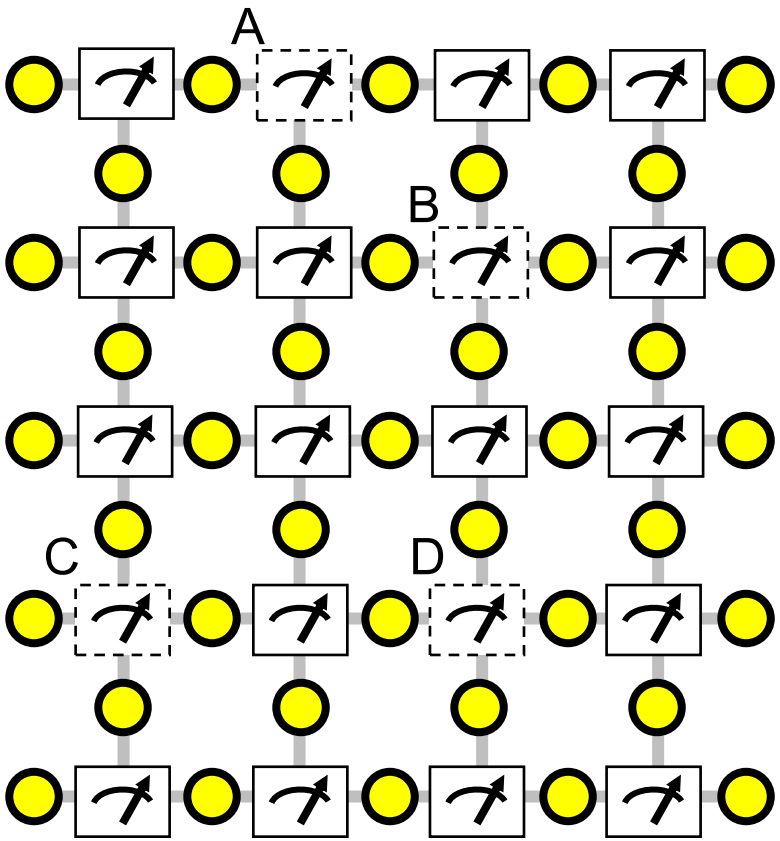
Minimum Weight Perfect Matching (MWPM)



V: Hot syndromes
W_e: Manhattan distance

Exact solution: **Blossom algorithm ($O(n^3)$)**

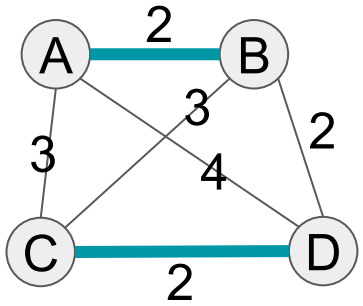
表面符号におけるエラー推定（復号）



仮定

- XとZのエラーは独立に推定できる
- なるべく短いエラー鎖が生じる

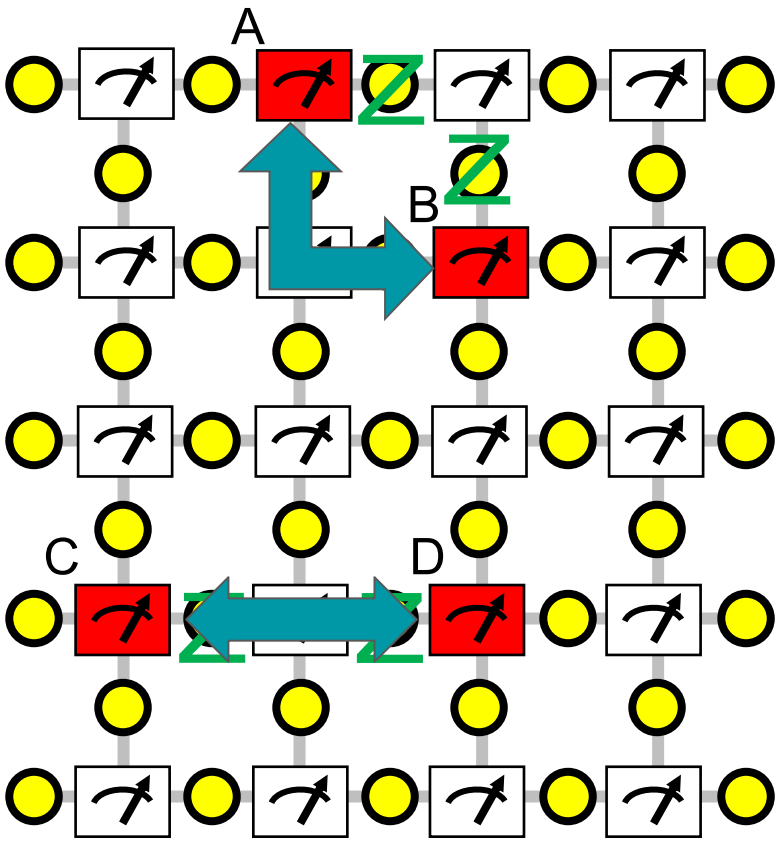
Minimum Weight Perfect Matching (MWPM)



V: Hot syndromes
W_e: Manhattan distance

Exact solution: **Blossom algorithm ($O(n^3)$)**

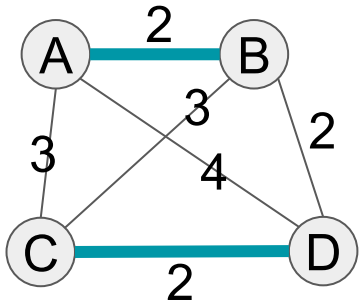
異なるつなぎ方をした場合



仮定

- XとZのエラーは独立に推定できる
- なるべく短いエラー鎖が生じる

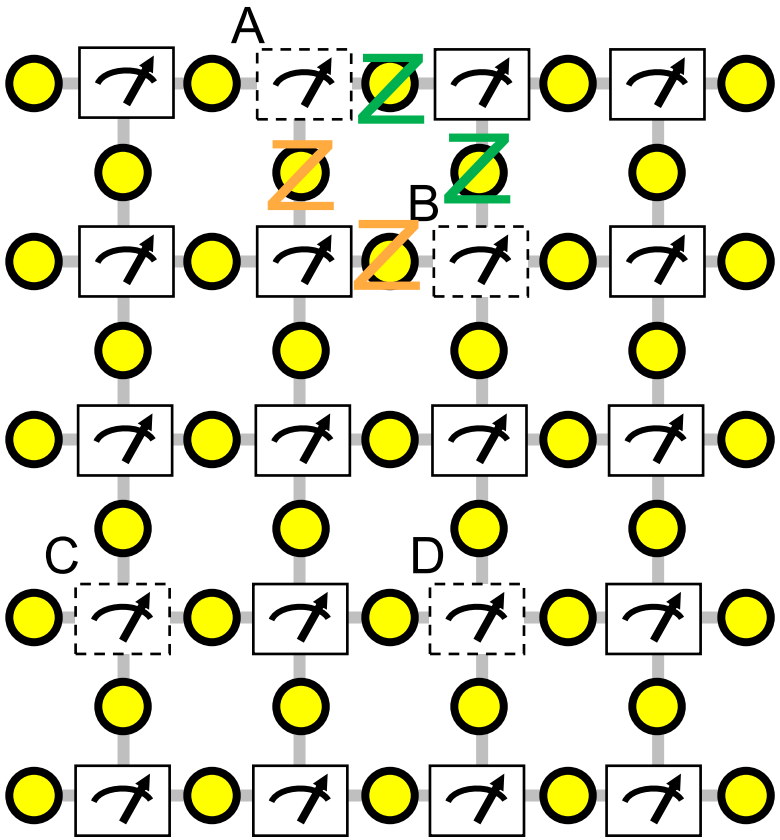
Minimum Weight Perfect Matching (MWPM)



V: Hot syndromes
W_e: Manhattan distance

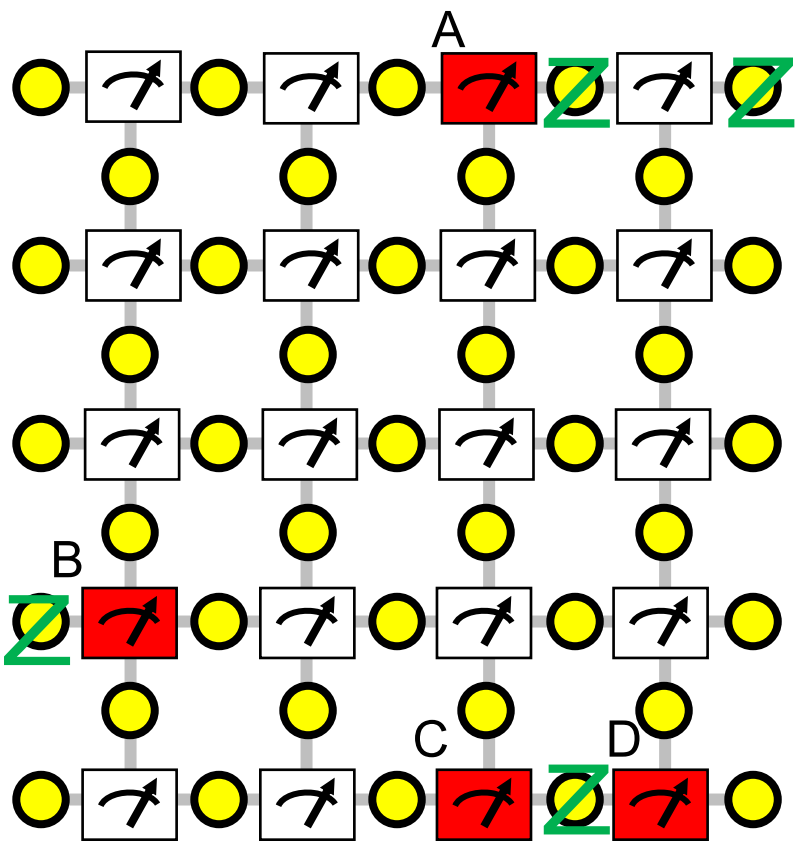
Exact solution: **Blossom algorithm ($O(n^3)$)**

異なるつなぎ方をした場合



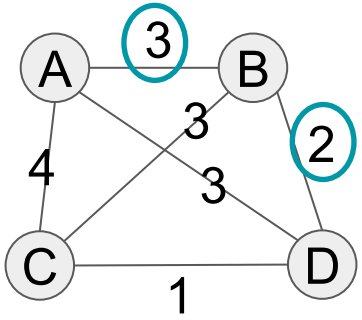
- エラーと訂正の合計が
  closed loopを作る場合も復号成功
  - （Topologically）trivial chainと呼ばれる
  - 論理*I*操作になる
    = 論理状態に影響を与えない
    = 論理エラーにならない
- 訂正の成否は奇パリティの
  ペア決めのみに依存しつなぎ方に
  依らない

境界につながるエラー



- グラフのエッジ重みを工夫することで境界につながるエラーもマッチング問題により推定できる
  - 「ノード間のマンハッタン距離」と「それぞれのノードの最寄りの境界からの距離の和」のうち小さい方を重みとする

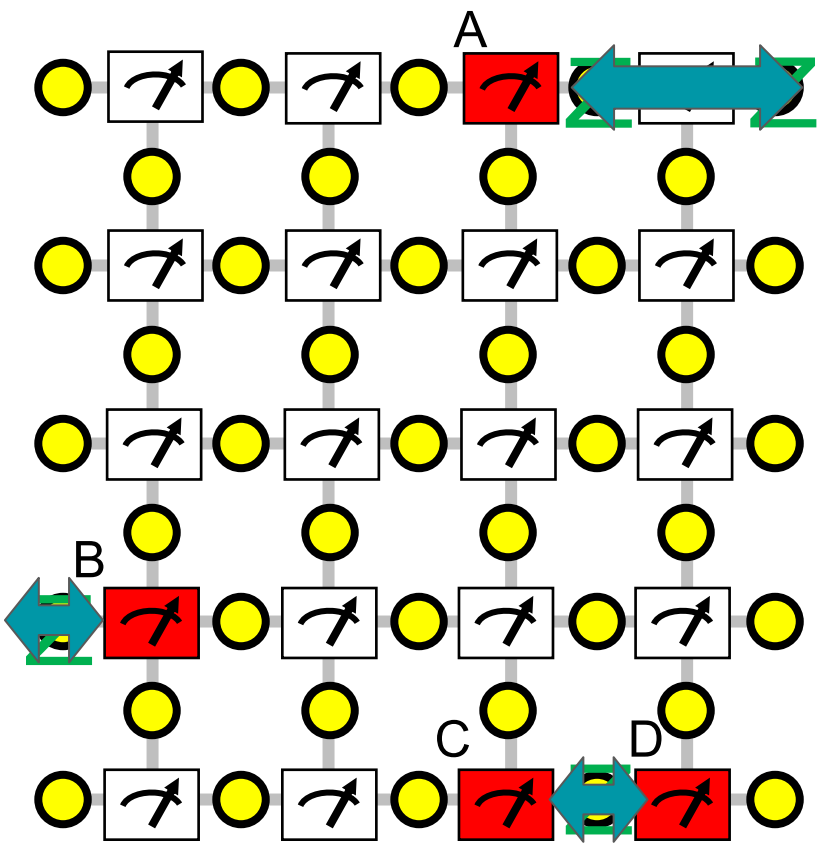
マンハッタン距離：5
境界との距離の和：1＋2＝**3**



マンハッタン距離：4
境界との距離の和：1＋1＝**2**

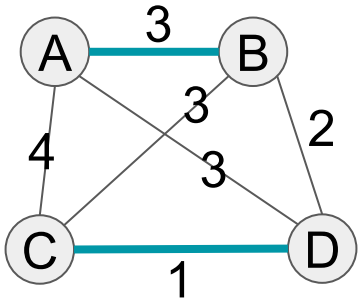
マンハッタン距離：**1**
境界との距離の和：1＋2＝3

境界につながるエラー



- グラフのエッジ重みを工夫することで境界につながるエラーもマッチング問題により推定できる
  - 「ノード間のマンハッタン距離」と「それぞれのノードの最寄りの境界からの距離の和」のうち小さい方を重みとする

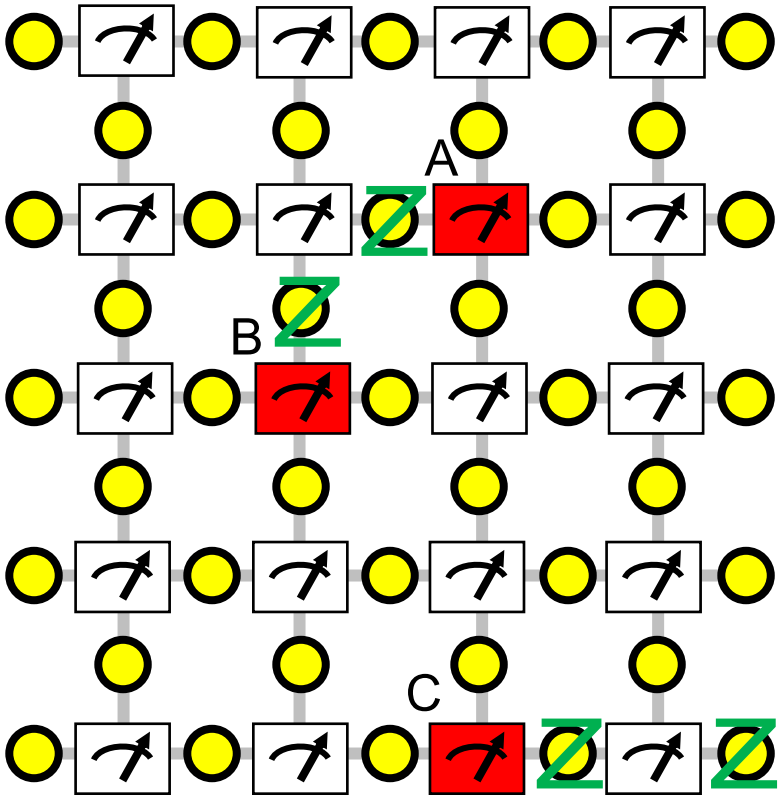
マンハッタン距離：5
境界との距離の和：1＋2＝**3**



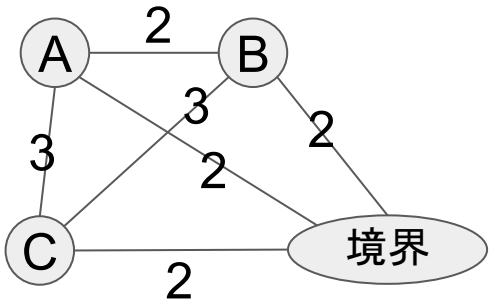
マンハッタン距離：4
境界との距離の和：1＋1＝**2**

マンハッタン距離：**1**
境界との距離の和：1＋2＝3

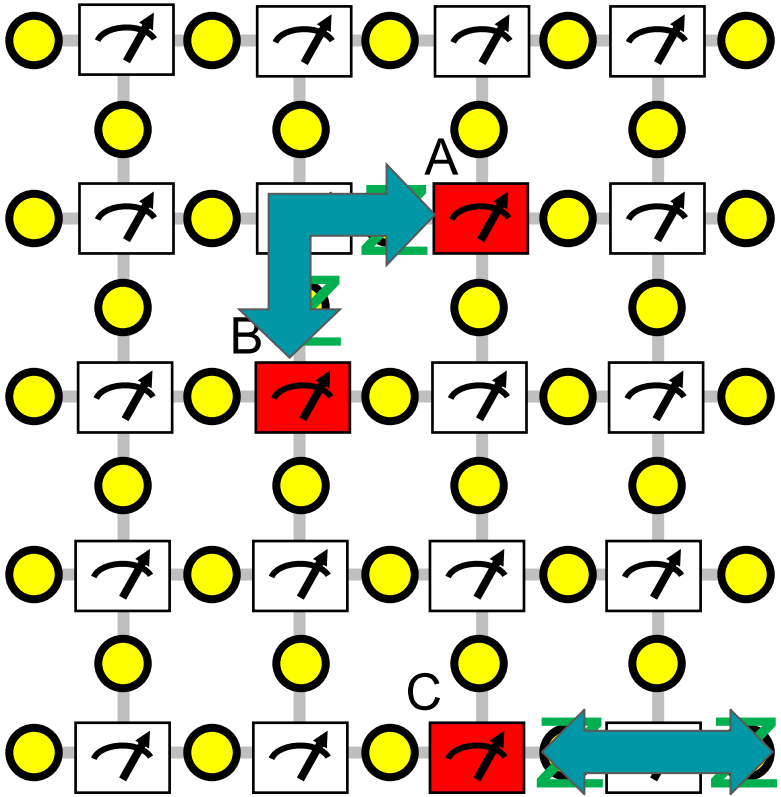
頂点が奇数の場合



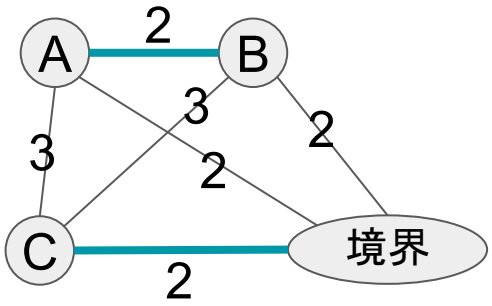
- 奇数ノードの完全グラフには完全マッチングが存在しない
- 「境界を表す仮想的なノード」を追加して偶数ノードの完全グラフをつくる
  - 境界ノードとのエッジの重みは最寄りの境界への距離



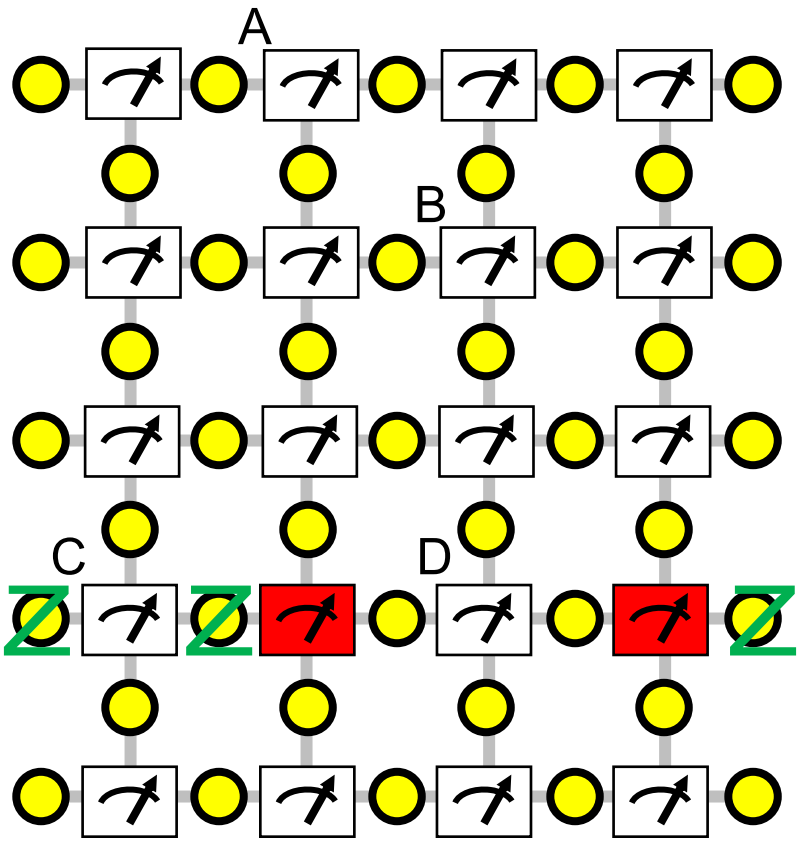
頂点が奇数の場合



- 奇数ノードの完全グラフには完全マッチングが存在しない
- 「境界を表す仮想的なノード」を追加して偶数ノードの完全グラフをつくる
  - 境界ノードとのエッジの重みは最寄りの境界への距離

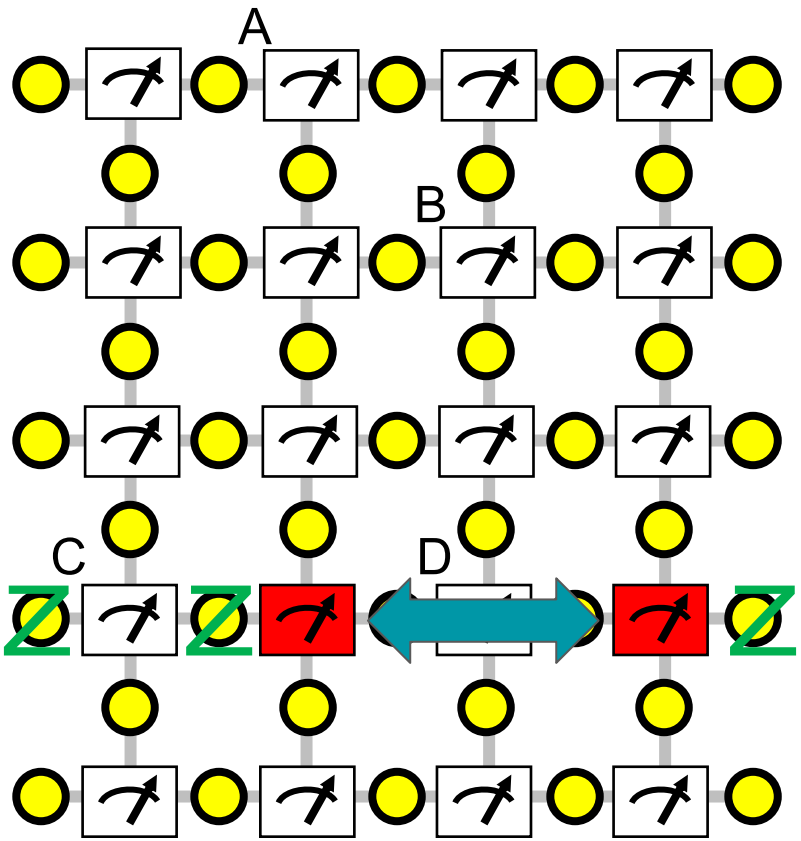


論理エラーが生じる場合



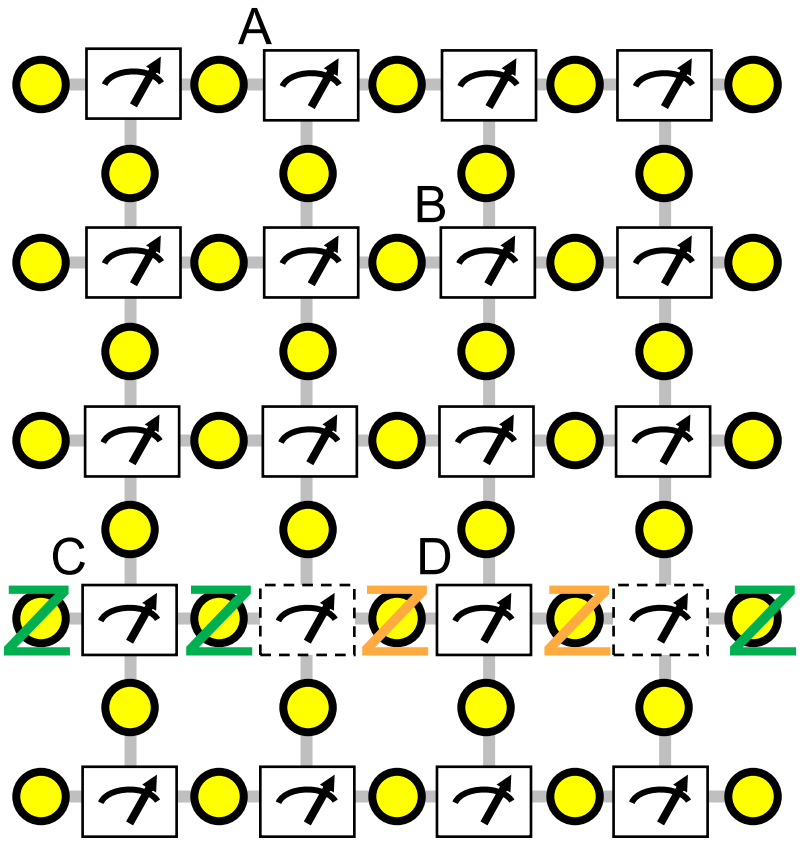
- エラーと訂正の合計がある境界から別の境界へとつながるchainを作る
  - Non-trivial chain
  - 論理Z操作になる
  - = 望まない論理ビットの状態変化
  - = 論理エラー
- 最低で符号距離dの半分以上のエラーで復号に失敗する

論理エラーが生じる場合



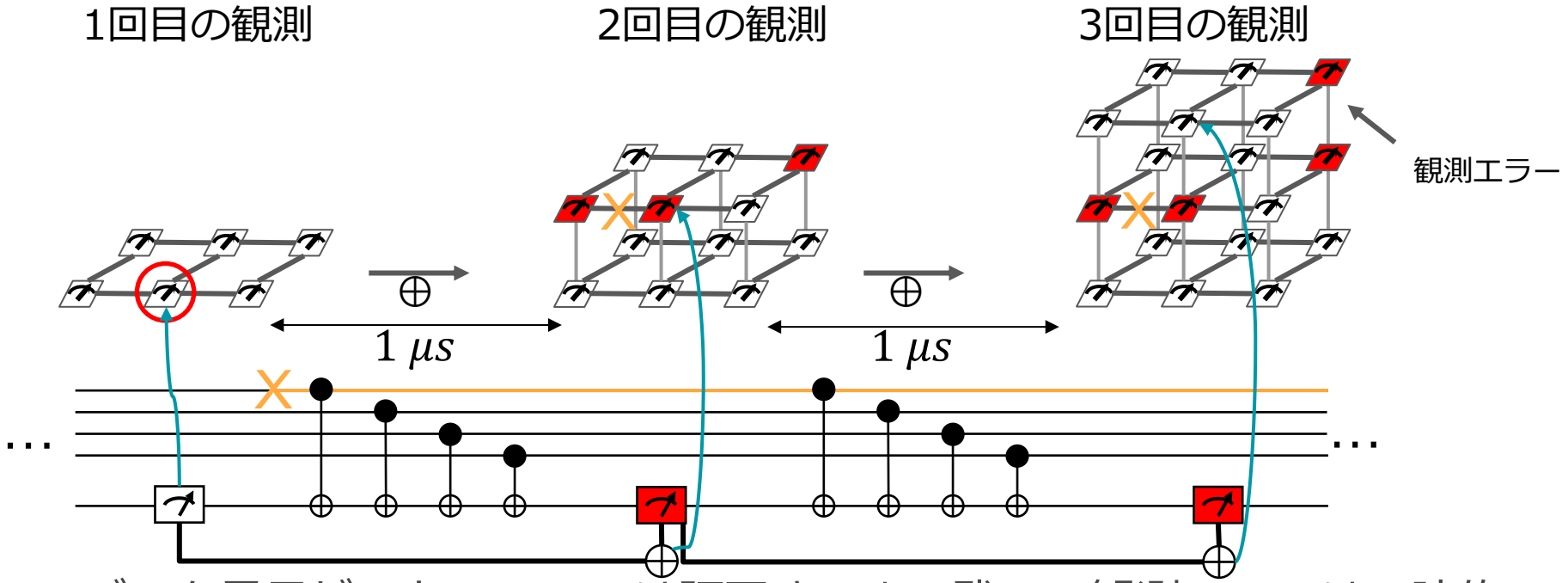
- エラーと訂正の合計がある境界から別の境界へとつながるchainを作る
  - Non-trivial chain
  - 論理Z操作になる
  - ＝ 望まない論理ビットの状態変化
  - ＝ 論理エラー
- 最低で符号距離dの半分以上のエラーで復号に失敗する

論理エラーが生じる場合



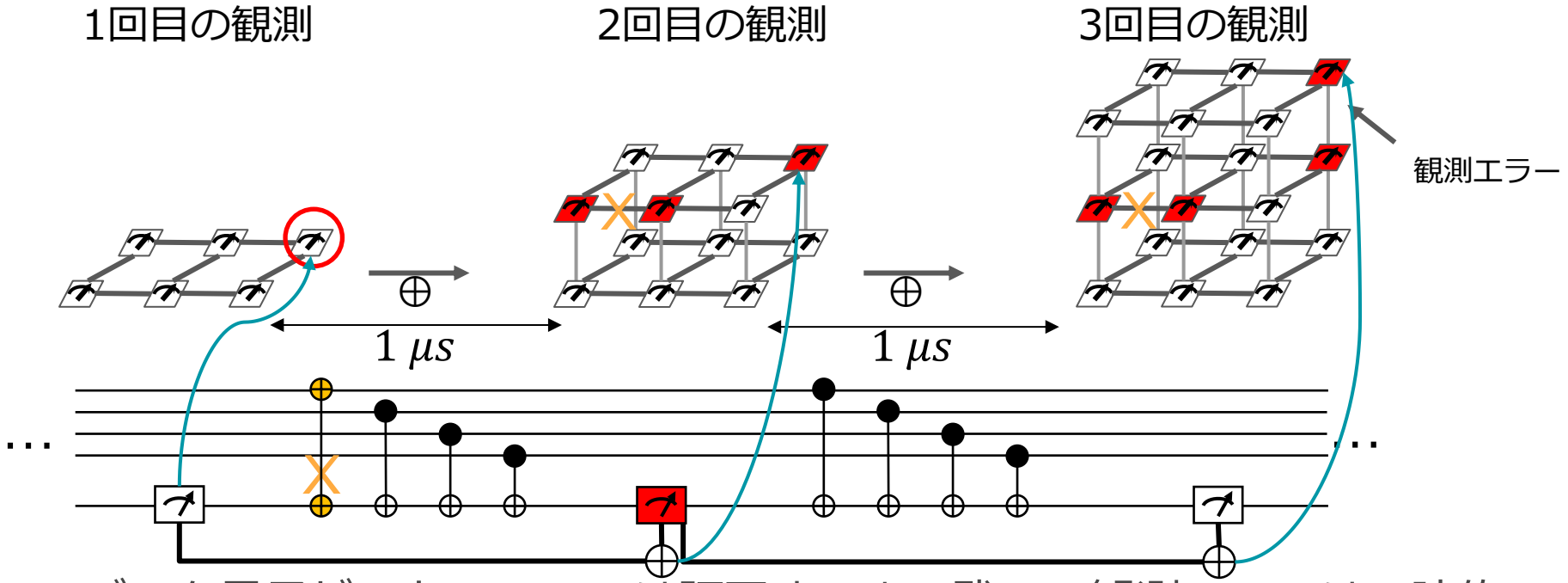
- エラーと訂正の合計がある境界から別の境界へとつながるchainを作る
  - Non-trivial chain
  - 論理Z操作になる
  - ＝ 望まない論理ビットの状態変化
  - ＝ 論理エラー
- 最低で符号距離dの半分以上のエラーで復号に失敗する

補助量子ビットの観測エラーへの対応



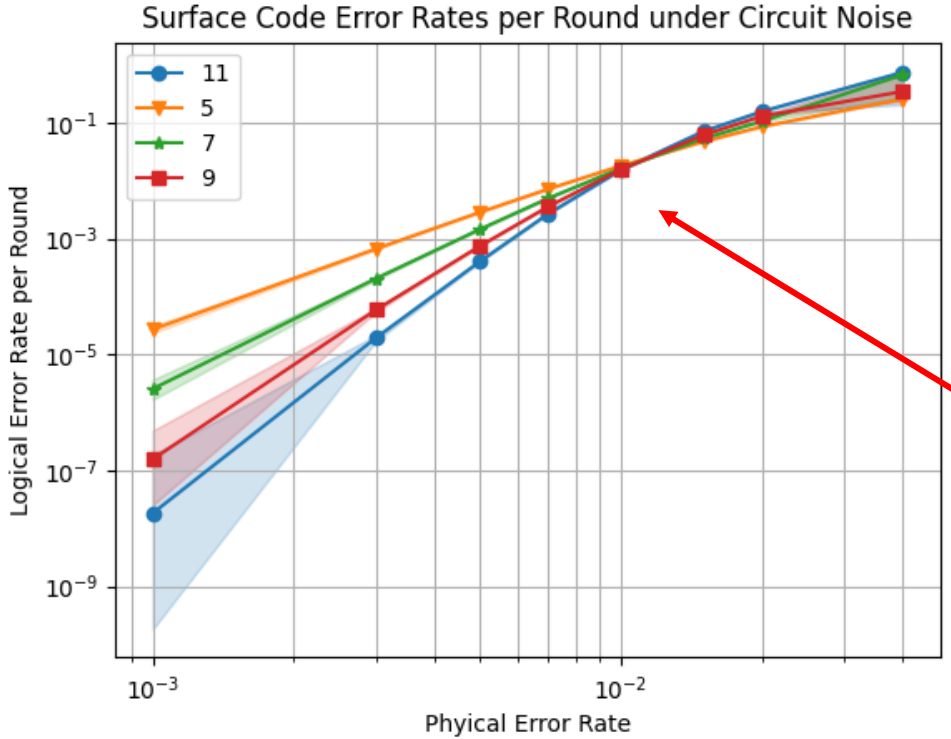
- データ量子ビットのエラーは訂正するまで残る、観測エラーは一時的
- 複数回観測をおこない観測値を垂直にスタックする
  - 前時刻の観測値とXORをとることで、観測値の変化した時点を表す
- エラー推定＝3次元格子上でグラフマッチング問題を解く

補助量子ビットの観測エラーへの対応



- データ量子ビットのエラーは訂正するまで残る、観測エラーは一時的
- 複数回観測をおこない観測値を垂直にスタックする
  - 前時刻の観測値とXORをとることで、観測値の変化した時点を表す
- エラー推定＝3次元格子上でグラフマッチング問題を解く

表面符号の性能



しきい値
1%程度

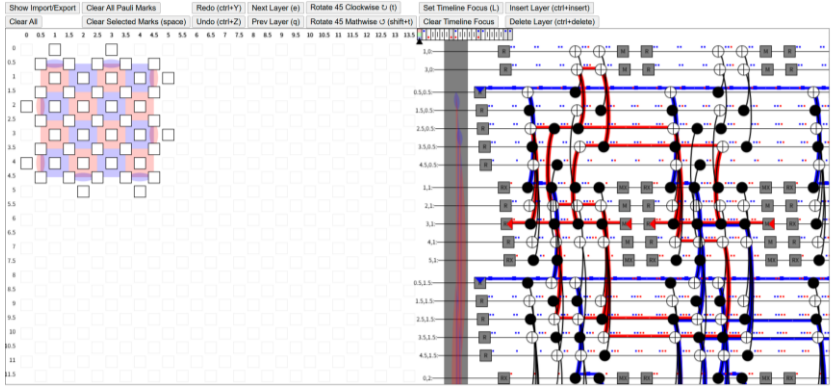
- しきい値: 物理エラー率がこの値以下であれば、
  符号距離dを増やせば指数的に論理エラー率を下げられる

発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

Stim+Crumbleを用いたシミュレーション

- Stim: Python経由で使えるClifford回路シミュレータ
  - 回路レベルノイズのシミュレーションを高速に行える
  - PyMatchingによる誤り推定、Sinterによる論理エラー率算出が容易
  - QEC符号の誤り訂正性能の評価ツールのデファクトスタンダード
- Crumble: スタビライザ回路とエラーイベントの可視化
  - Web上でインタラクティブにstim回路およびエラーイベントを可視化
  - Stim回路 ↔ Crumble双方のImport/Exportが可能



Stim+Crumbleを用いたシミュレーション

Stimコード例（$d = 5$のRotated surface code）

量子ビットの宣言

```
QUBIT_COORDS(0, 2) 0
QUBIT_COORDS(0, 4) 1
QUBIT_COORDS(0.5, 0.5) 2
QUBIT_COORDS(0.5, 1.5) 3
QUBIT_COORDS(0.5, 2.5) 4
QUBIT_COORDS(0.5, 3.5) 5
QUBIT_COORDS(0.5, 4.5) 6
QUBIT_COORDS(1, 0) 7
QUBIT_COORDS(1, 1) 8
QUBIT_COORDS(1, 2) 9
QUBIT_COORDS(1, 3) 10
…
QUBIT_COORDS(1, 4) 11
QUBIT_COORDS(5, 1) 47
QUBIT_COORDS(5, 3) 48
```

スタビライザ測定

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```

Detectorの定義

```
DETECTOR(1, 2, 0) rec[-24]
DETECTOR(1, 4, 0) rec[-23]
DETECTOR(2, 1, 0) rec[-22]
DETECTOR(2, 3, 0) rec[-21]
DETECTOR(2, 5, 0) rec[-20]
DETECTOR(3, 2, 0) rec[-19]
DETECTOR(3, 4, 0) rec[-18]
DETECTOR(4, 1, 0) rec[-17]
DETECTOR(4, 3, 0) rec[-16]
DETECTOR(4, 5, 0) rec[-15]
DETECTOR(3, 0, 0) rec[-14]
DETECTOR(1, 0, 0) rec[-13]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
...
```

Stim+Crumbleを用いたシミュレーション

Stimコード例（$d = 5$のRotated surface code）

…

スタビライザ測定+Detectorの定義

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```

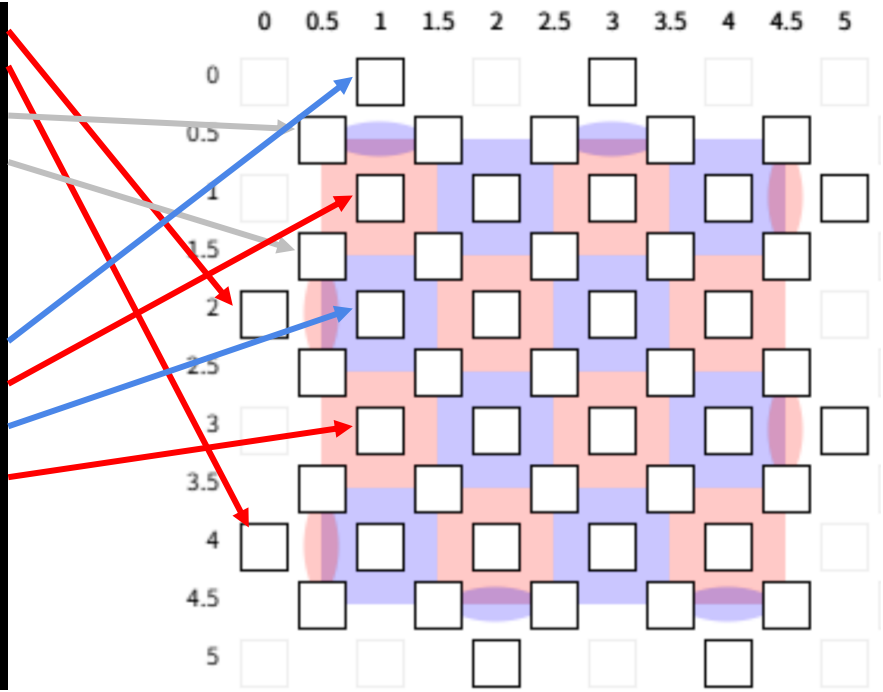
$\times O(d)$回

論理測定

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 2) rec[-24] rec[-48]
DETECTOR(1, 4, 2) rec[-23] rec[-47]
…
DETECTOR(3, 0, 2) rec[-14] rec[-38]
DETECTOR(1, 0, 2) rec[-13] rec[-37]
DETECTOR(1, 1, 2) rec[-12] rec[-36]
DETECTOR(1, 3, 2) rec[-11] rec[-35]
…
DETECTOR(0, 4, 2) rec[-1] rec[-25]
TICK
M 2 3 4 5 6 12 13 14 15 16 22 23 ...
```

量子ビットの宣言

```
QUBIT_COORDS(0, 2) 0
QUBIT_COORDS(0, 4) 1
QUBIT_COORDS(0.5, 0.5) 2
QUBIT_COORDS(0.5, 1.5) 3
QUBIT_COORDS(0.5, 2.5) 4
QUBIT_COORDS(0.5, 3.5) 5
QUBIT_COORDS(0.5, 4.5) 6
QUBIT_COORDS(1, 0) 7
QUBIT_COORDS(1, 1) 8
QUBIT_COORDS(1, 2) 9
QUBIT_COORDS(1, 3) 10
…
QUBIT_COORDS(1, 4) 11
QUBIT_COORDS(5, 1) 47
QUBIT_COORDS(5, 3) 48
```



データ量子ビット

[2, 3, 4, 5, 6,
12, 13, 14, 15, 16,
22, 23, 24, 25, 26,
32, 33, 34, 35, 36,
42, 43, 44, 45, 46]

補助量子ビット（Zエラー）

[0, 1, 8, 10, 18, 20,
28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー）

[7, 9, 11, 17, 19, 21,
27, 29, 31, 37, 39, 41]

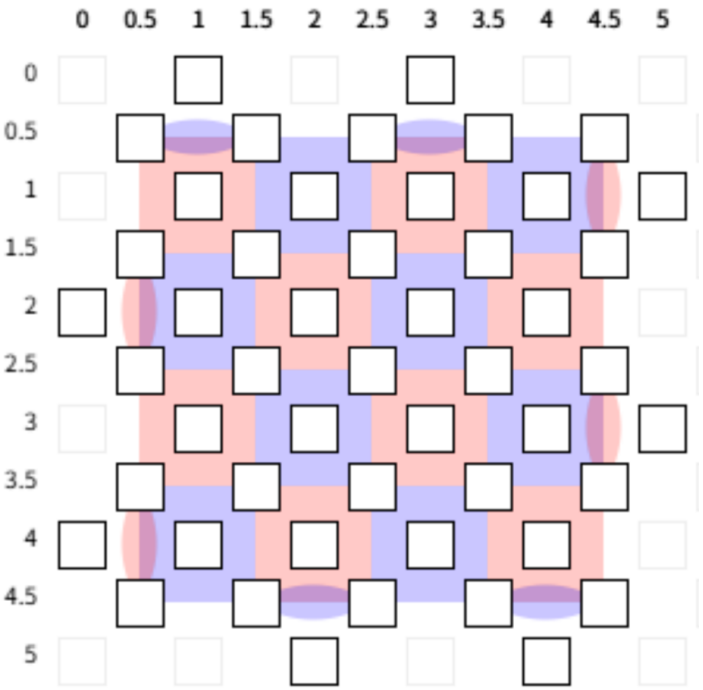
スタビライザ測定

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

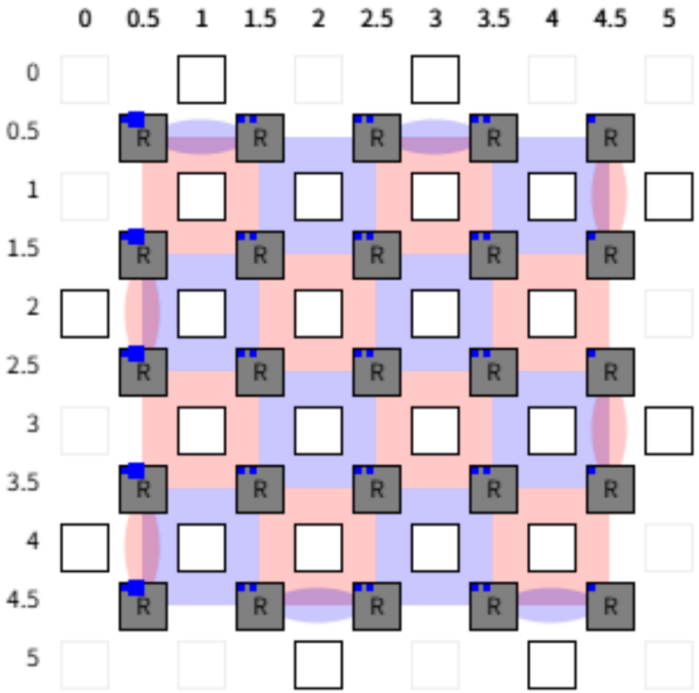
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットを
|0⟩にリセット
-> 論理 |0⟩状態の
メモリ操作

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

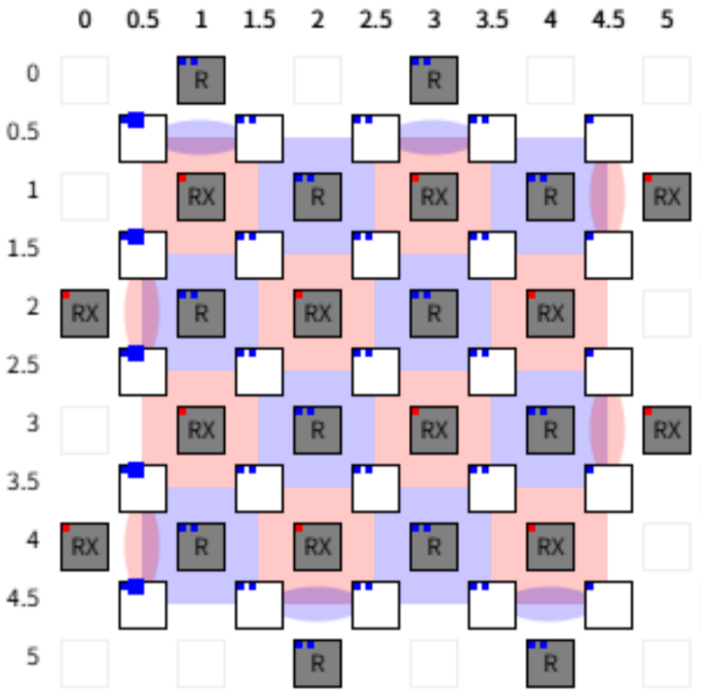
Xエラー検出用補助量子ビットを

$|0\rangle$にリセット

Zエラー検出用補助量子ビットを

$|+\rangle$にリセット

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

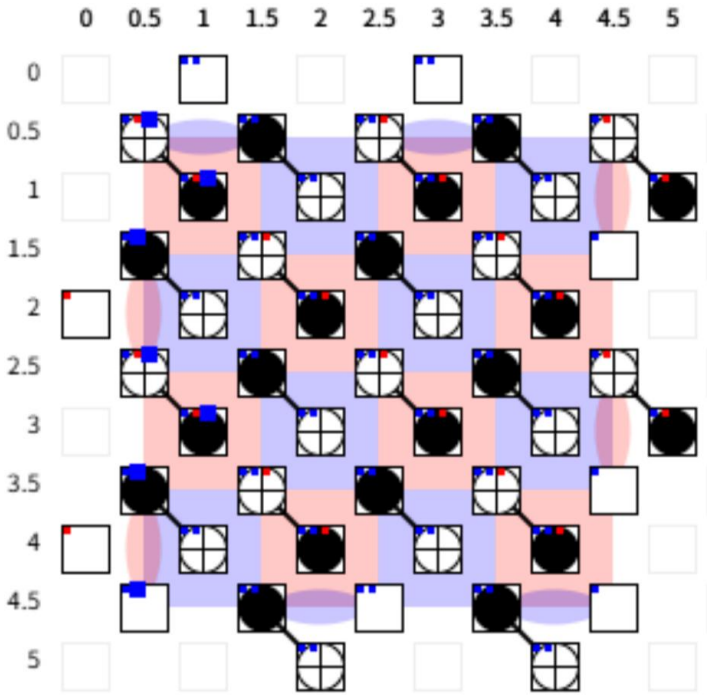
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT（第1層）

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

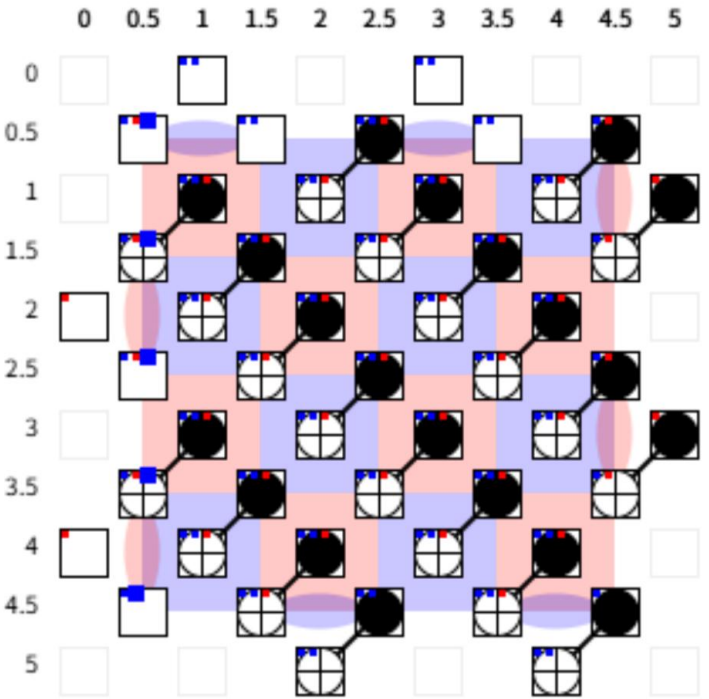
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT（第2層）

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

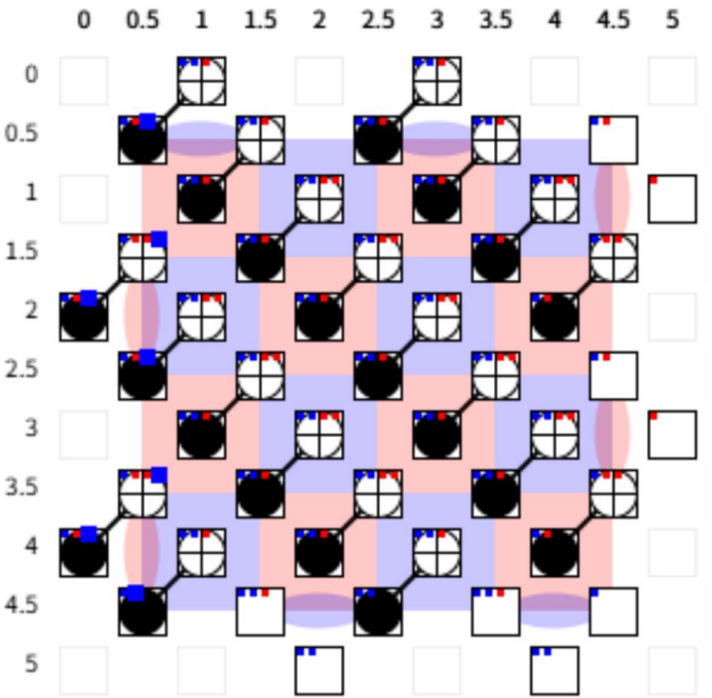
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT（第3層）

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

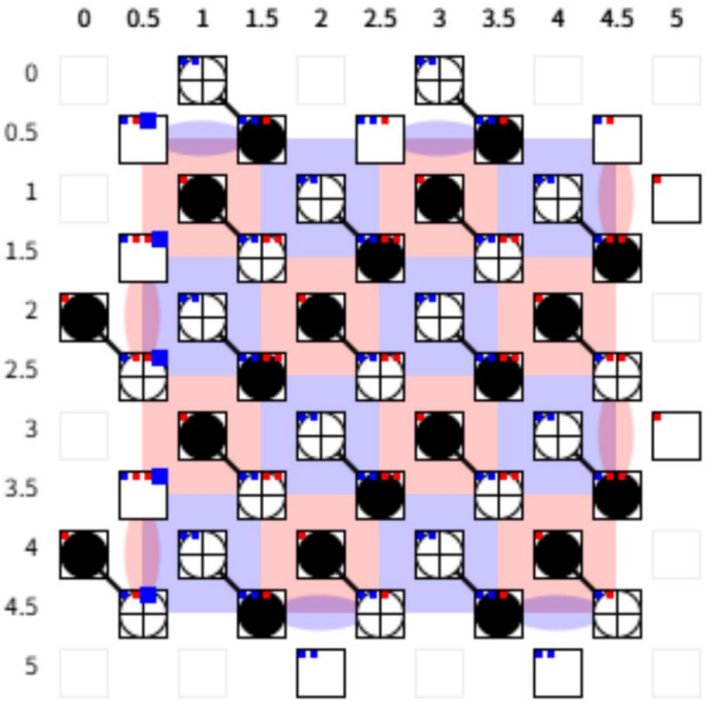
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT（第4層）

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

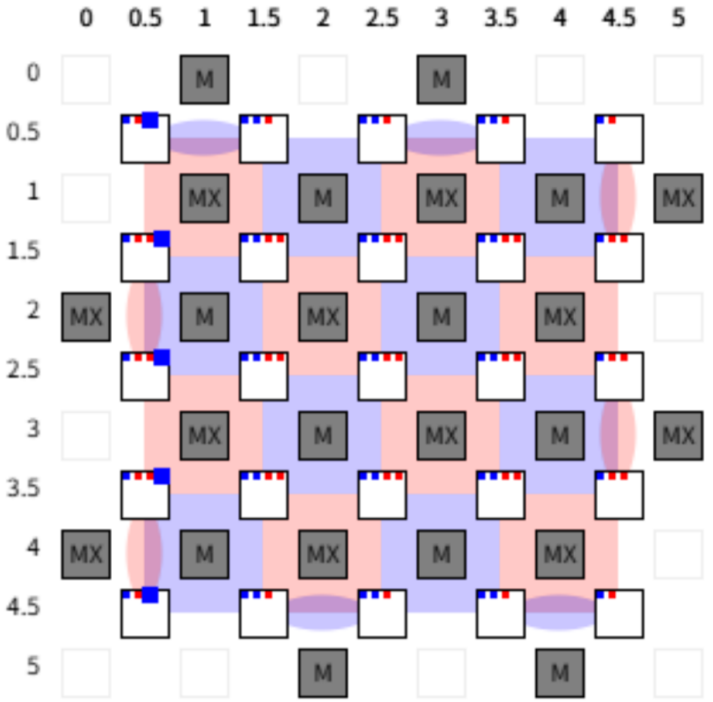
Xエラー検出用補助量子ビットを

Z基底で測定

Zエラー検出用補助量子ビットを

X基底で測定

```
TICK
R 2 3 4 5 6 12 13 14 15 16 22 23 24 ...
TICK
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



Detectorの定義（初回）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 0) rec[-24]
DETECTOR(1, 4, 0) rec[-23]
DETECTOR(2, 1, 0) rec[-22]
DETECTOR(2, 3, 0) rec[-21]
DETECTOR(2, 5, 0) rec[-20]
DETECTOR(3, 2, 0) rec[-19]
DETECTOR(3, 4, 0) rec[-18]
DETECTOR(4, 1, 0) rec[-17]
DETECTOR(4, 3, 0) rec[-16]
DETECTOR(4, 5, 0) rec[-15]
DETECTOR(3, 0, 0) rec[-14]
DETECTOR(1, 0, 0) rec[-13]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

recの中身:

[

(9の初回の測定値), <- rec[-24]

(11の初回の測定値), <- rec[-23]

…,

(7の初回の測定値), <- rec[-13]

(8の初回の測定値), <- rec[-12]

…,

(1の初回の測定値), <- rec[-1]

]

Detectorの定義（初回）

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 0) rec[-24]
DETECTOR(1, 4, 0) rec[-23]
DETECTOR(2, 1, 0) rec[-22]
DETECTOR(2, 3, 0) rec[-21]
DETECTOR(2, 5, 0) rec[-20]
DETECTOR(3, 2, 0) rec[-19]
DETECTOR(3, 4, 0) rec[-18]
DETECTOR(4, 1, 0) rec[-17]
DETECTOR(4, 3, 0) rec[-16]
DETECTOR(4, 5, 0) rec[-15]
DETECTOR(3, 0, 0) rec[-14]
DETECTOR(1, 0, 0) rec[-13]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

マッチング問題を解く
グラフのノードを定義

Xエラー検出用の
補助量子ビット：
観測値が-1であれば
ホットシンドローム

recの中身:

```
[
(9の初回の測定値),      <- rec[-24]
(11の初回の測定値),     <- rec[-23]
…,
(7の初回の測定値),      <- rec[-13]
(8の初回の測定値),      <- rec[-12]
…,
(1の初回の測定値),      <- rec[-1]
]
```

Zエラー検出用の
補助量子ビット：
初回の測定値は完全にランダムな値
になるが論理Zエラーは論理$|0\rangle$に影
響を与えないので初回は無視でOK

Detectorの定義（初回）

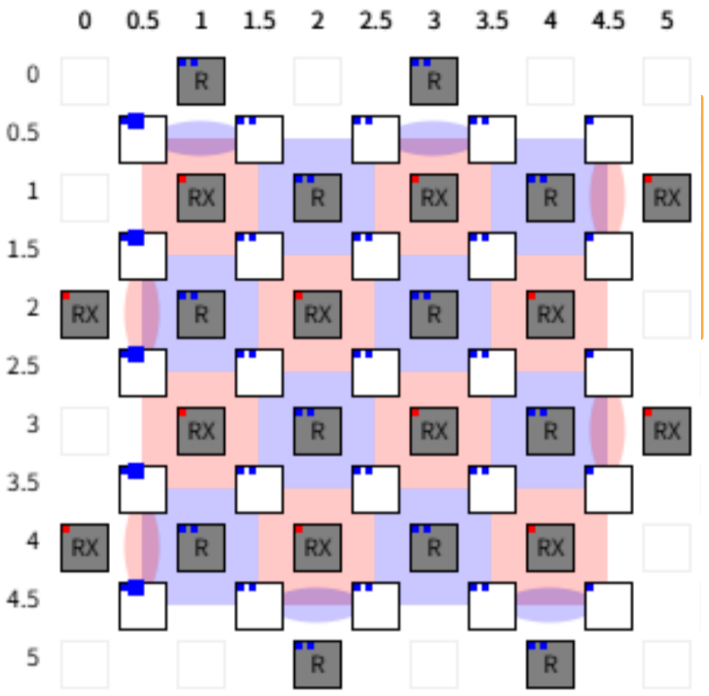
データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 0) rec[-24]
DETECTOR(1, 4, 0) rec[-23]
DETECTOR(2, 1, 0) rec[-22]
DETECTOR(2, 3, 0) rec[-21]
DETECTOR(2, 5, 0) rec[-20]
DETECTOR(3, 2, 0) rec[-19]
DETECTOR(3, 4, 0) rec[-18]
DETECTOR(4, 1, 0) rec[-17]
DETECTOR(4, 3, 0) rec[-16]
DETECTOR(4, 5, 0) rec[-15]
DETECTOR(3, 0, 0) rec[-14]
DETECTOR(1, 0, 0) rec[-13]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

補助量子ビットをそれぞれ $|0\rangle, |+\rangle$ にリセット

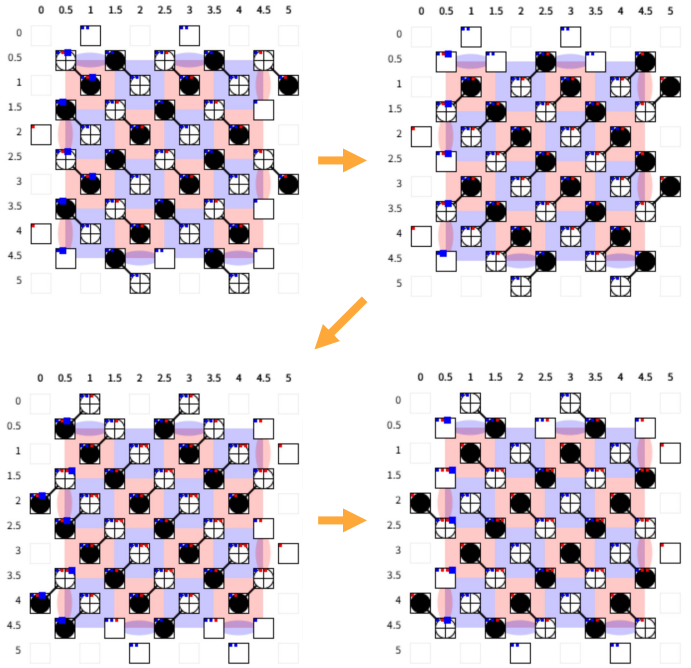


スタビライザ測定（2回目）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT×4

```
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定（2回目）

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

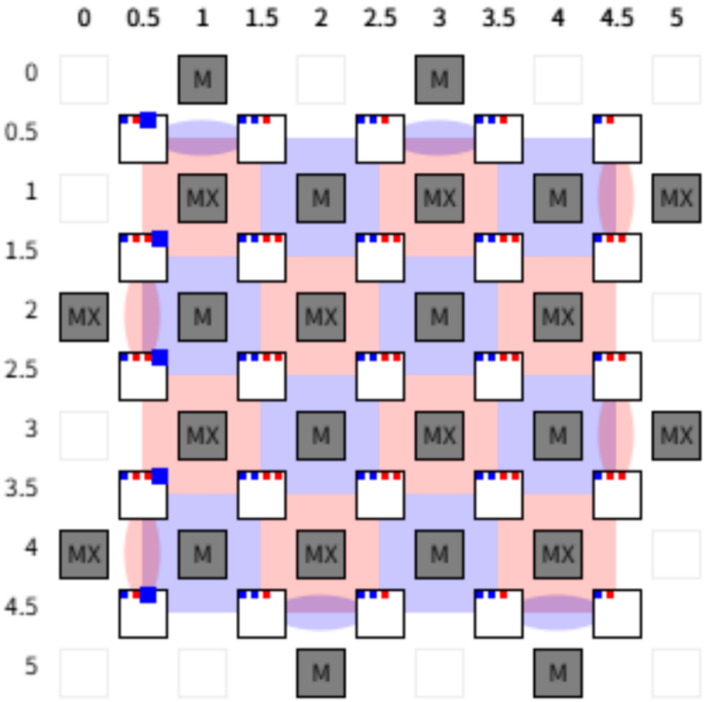
補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

*X*エラー検出用補助量子ビットを*Z*基底で測定

*Z*エラー検出用補助量子ビットを*X*基底で測定

```
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



Detectorの定義（2回目）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 1) rec[-24] rec[-48]
DETECTOR(1, 4, 1) rec[-23] rec[-47]
…
DETECTOR(3, 0, 1) rec[-14] rec[-38]
DETECTOR(1, 0, 1) rec[-13] rec[-37]
DETECTOR(1, 1, 1) rec[-12] rec[-36]
DETECTOR(1, 3, 1) rec[-11] rec[-35]
…
DETECTOR(0, 4, 1) rec[-1] rec[-25]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

recの中身:
[
(9の初回の測定値), <- rec[-48]
…,
(7の初回の測定値), <- rec[-37]
(8の初回の測定値), <- rec[-36]
…,
(1の初回の測定値), <- rec[-25]
(9の2回目の測定値), <- rec[-24]
…
(1の2回目の測定値), <- rec[-1]
]

Detectorの定義（2回目）

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 1) rec[-24] rec[-48]
DETECTOR(1, 4, 1) rec[-23] rec[-47]
…
DETECTOR(3, 0, 1) rec[-14] rec[-38]
DETECTOR(1, 0, 1) rec[-13] rec[-37]
DETECTOR(1, 1, 1) rec[-12] rec[-36]
DETECTOR(1, 3, 1) rec[-11] rec[-35]
…
DETECTOR(0, 4, 1) rec[-1] rec[-25]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

マッチング問題を解く
グラフのノードを定義

X/Zエラー検出用の
補助量子ビット：
前回の観測値との差分を
取り、異なっていれば
ホットシンドローム
-> 前回の測定から新た
に生じたエラーを検出

recの中身:
[
(9の初回の測定値), <- rec[-48]
…,
(7の初回の測定値), <- rec[-37]
(8の初回の測定値), <- rec[-36]
…,
(1の初回の測定値), <- rec[-25]
(9の2回目の測定値), <- rec[-24]
…
(1の2回目の測定値), <- rec[-1]
]

Detectorの定義（2回目）

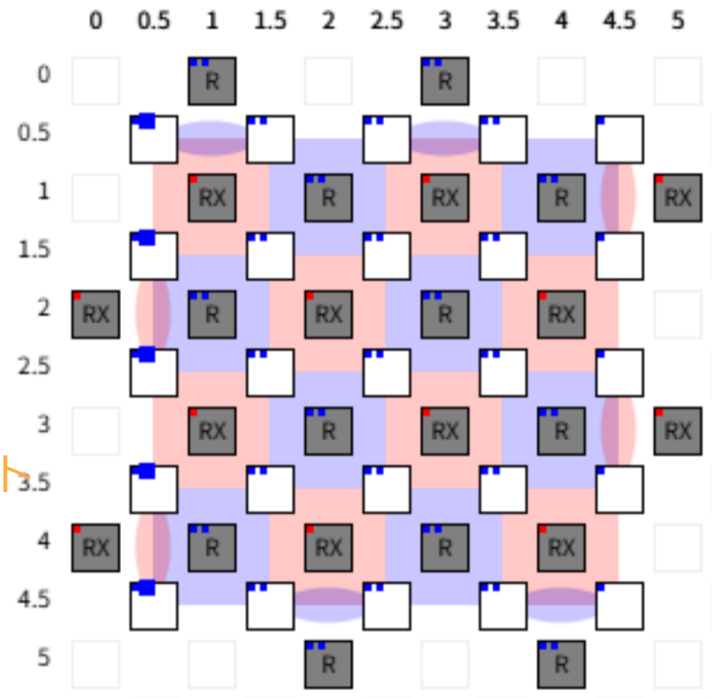
データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 1) rec[-24] rec[-48]
DETECTOR(1, 4, 1) rec[-23] rec[-47]
…
DETECTOR(3, 0, 1) rec[-14] rec[-38]
DETECTOR(1, 0, 1) rec[-13] rec[-37]
DETECTOR(1, 1, 1) rec[-12] rec[-36]
DETECTOR(1, 3, 1) rec[-11] rec[-35]
…
DETECTOR(0, 4, 1) rec[-1] rec[-25]
TICK
R 9 11 17 19 21 29 31 37 39 41 7 27
RX 8 10 18 20 28 30 38 40 47 48 1 0
```

補助量子ビットを
それぞれ$|0\rangle, |+\rangle$にリセット

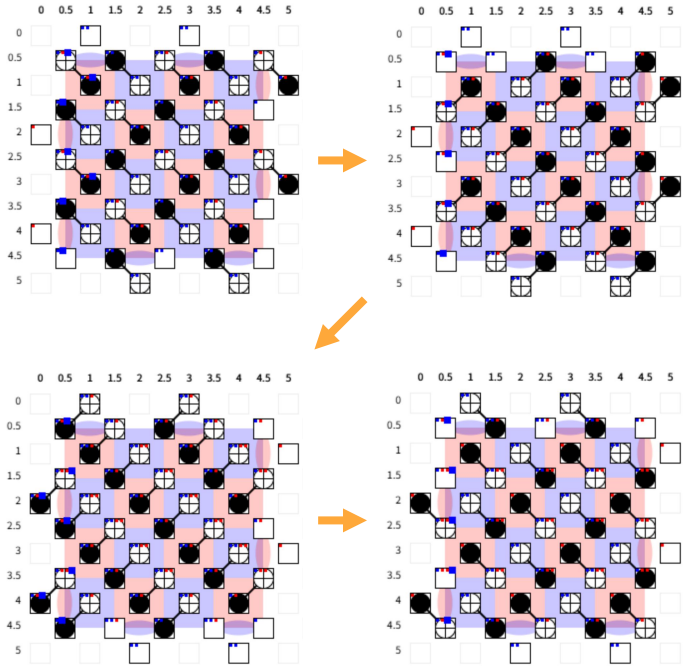


スタビライザ測定（3回目）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

データ量子ビットと補助量子ビットの間でCNOT×4

```
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



スタビライザ測定（3回目）

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

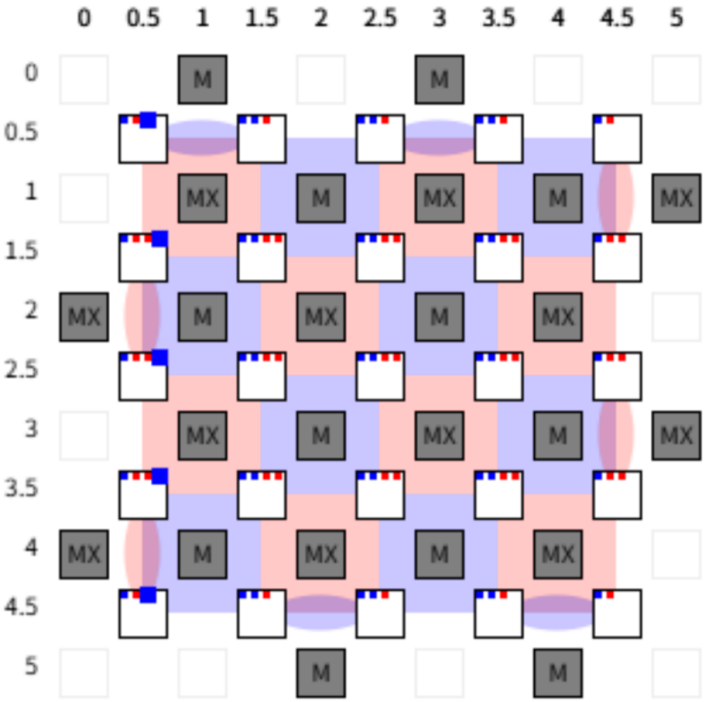
補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

*X*エラー検出用補助量子ビットを*Z*基底で測定

*Z*エラー検出用補助量子ビットを*X*基底で測定

```
R 9 11 17 19 21 29 31 37 39 41 27 7
RX 8 10 18 20 28 30 38 40 47 48 0 1
TICK
CX 8 2 3 9 10 4 5 11 12 17 18 13 14 ...
TICK
CX 13 9 15 11 22 17 24 19 26 21 33 ...
TICK
CX 4 9 6 11 13 17 15 19 24 29 26 31 ...
TICK
CX 8 13 14 9 10 15 16 11 23 17 18 ...
TICK
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
```



Detectorの定義（3回目）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 2) rec[-24] rec[-48]
DETECTOR(1, 4, 2) rec[-23] rec[-47]
…
DETECTOR(3, 0, 2) rec[-14] rec[-38]
DETECTOR(1, 0, 2) rec[-13] rec[-37]
DETECTOR(1, 1, 2) rec[-12] rec[-36]
DETECTOR(1, 3, 2) rec[-11] rec[-35]
…
DETECTOR(0, 4, 2) rec[-1] rec[-25]
TICK
M 2 3 4 5 6 12 13 14 15 16 22 23 ...
```

recの中身:
[
(9の初回の測定値), <- rec[-72]
…,
(1の初回の測定値), <- rec[-49]
(9の2回目の測定値), <- rec[-48]
…
(1の2回目の測定値), <- rec[-25]
(9の3回目の測定値), <- rec[-24]
…
(1の3回目の測定値), <- rec[-1]
]

Detectorの定義（3回目）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]
補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]
補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 2) rec[-24] rec[-48]
DETECTOR(1, 4, 2) rec[-23] rec[-47]
…
DETECTOR(3, 0, 2) rec[-14] rec[-38]
DETECTOR(1, 0, 2) rec[-13] rec[-37]
DETECTOR(1, 1, 2) rec[-12] rec[-36]
DETECTOR(1, 3, 2) rec[-11] rec[-35]
…
DETECTOR(0, 4, 2) rec[-1] rec[-25]
TICK
M 2 3 4 5 6 12 13 14 15 16 22 23 ...
```

マッチング問題を解く
グラフのノードを定義

X/Zエラー検出用の
補助量子ビット：
前回の観測値との差分を
取り、異なっていれば
ホットシンドローム
-> 前回の測定から新た
に生じたエラーを検出

recの中身:
[
(9の初回の測定値), <- rec[-72]
…,
(1の初回の測定値), <- rec[-49]
(9の2回目の測定値), <- rec[-48]
…
(1の2回目の測定値), <- rec[-25]
(9の3回目の測定値), <- rec[-24]
…
(1の3回目の測定値), <- rec[-1]
]

論理測定

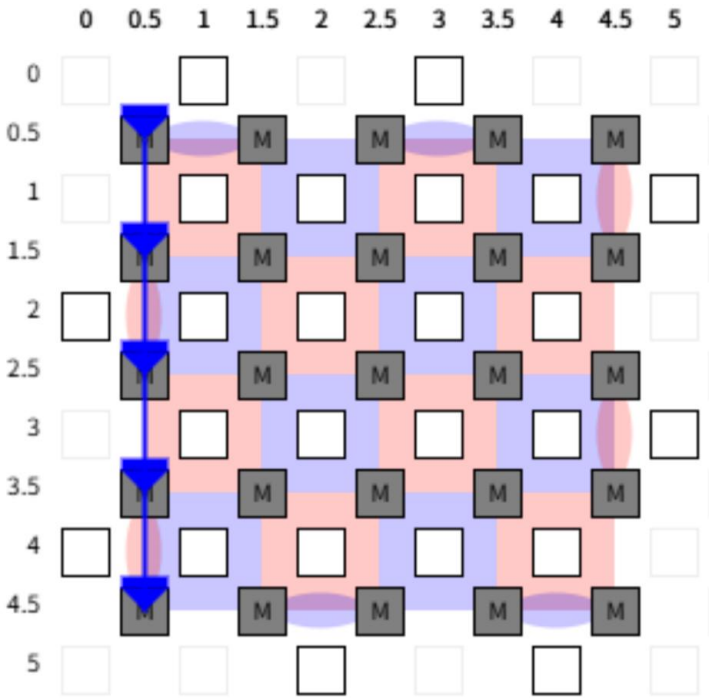
データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 9 11 17 19 21 29 31 37 39 41 27 7
MX 8 10 18 20 28 30 38 40 47 48 0 1
DETECTOR(1, 2, 2) rec[-24] rec[-48]
DETECTOR(1, 4, 2) rec[-23] rec[-47]
…
DETECTOR(3, 0, 2) rec[-14] rec[-38]
DETECTOR(1, 0, 2) rec[-13] rec[-37]
DETECTOR(1, 1, 2) rec[-12] rec[-36]
DETECTOR(1, 3, 2) rec[-11] rec[-35]
…
DETECTOR(0, 4, 2) rec[-1] rec[-25]
TICK
M 2 3 4 5 6 12 13 14 15 16 22 23 ..
```

データ量子ビットを
Z基底で測定



Detectorの定義（最後）

データ量子ビット : [2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用） : [0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用） : [7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 2 3 4 5 6 12 13 14 15 16 22 23 24 25 26 32 33 34 35 36 42 43 44 45 46
DETECTOR(1, 0, 3) rec[-20] rec[-25] rec[-39]
DETECTOR(1, 2, 3) rec[-18] rec[-19] rec[-23] rec[-24] rec[-49]
DETECTOR(1, 4, 3) rec[-16] rec[-17] rec[-21] rec[-22] rec[-48]
DETECTOR(2, 1, 3) rec[-14] rec[-15] rec[-19] rec[-20] rec[-47]
DETECTOR(2, 3, 3) rec[-12] rec[-13] rec[-17] rec[-18] rec[-46]
DETECTOR(2, 5, 3) rec[-11] rec[-16] rec[-45]
DETECTOR(3, 0, 3) rec[-10] rec[-15] rec[-38]
DETECTOR(3, 2, 3) rec[-8] rec[-9] rec[-13] rec[-14] rec[-44]
DETECTOR(3, 4, 3) rec[-6] rec[-7] rec[-11] rec[-12] rec[-43]
DETECTOR(4, 1, 3) rec[-4] rec[-5] rec[-9] rec[-10] rec[-42]
DETECTOR(4, 3, 3) rec[-2] rec[-3] rec[-7] rec[-8] rec[-41]
DETECTOR(4, 5, 3) rec[-1] rec[-6] rec[-40]
OBSERVABLE_INCLUDE(0) rec[-21] rec[-22] rec[-23] rec[-24] rec[-25]
```

*X*エラー検出用の補助量子ビットの前回の観測値とデータ量子ビットの観測値を比較してDetectorを定義
-> データ量子ビットに生じるエラーと測定エラーを区別なく扱うため

データ量子ビットを*Z*基底で測定しているので、
論理*Z*エラーの影響はない

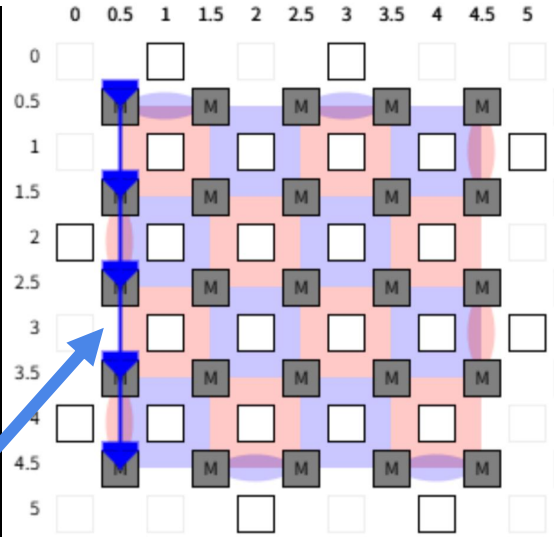
Observableの定義

データ量子ビット：[2, 3, 4, 5, 6, 12, 13, 14, 15, 16, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 42, 43, 44, 45, 46]

補助量子ビット（Zエラー検出用）：[0, 1, 8, 10, 18, 20, 28, 30, 38, 40, 47, 48]

補助量子ビット（Xエラー検出用）：[7, 9, 11, 17, 19, 21, 27, 29, 31, 37, 39, 41]

```
M 2 3 4 5 6 12 13 14 15 16 22 23 24 25 26 32 33 34 35 36 42 43 44 45 46
DETECTOR(1, 0, 3) rec[-20] rec[-25] rec[-39]
DETECTOR(1, 2, 3) rec[-18] rec[-19] rec[-23] rec[-24] rec[-49]
DETECTOR(1, 4, 3) rec[-16] rec[-17] rec[-21] rec[-22] rec[-48]
DETECTOR(2, 1, 3) rec[-14] rec[-15] rec[-19] rec[-20] rec[-47]
DETECTOR(2, 3, 3) rec[-12] rec[-13] rec[-17] rec[-18] rec[-46]
DETECTOR(2, 5, 3) rec[-11] rec[-16] rec[-45]
DETECTOR(3, 0, 3) rec[-10] rec[-15] rec[-38]
DETECTOR(3, 2, 3) rec[-8] rec[-9] rec[-13] rec[-14] rec[-44]
DETECTOR(3, 4, 3) rec[-6] rec[-7] rec[-11] rec[-12] rec[-43]
DETECTOR(4, 1, 3) rec[-4] rec[-5] rec[-9] rec[-10] rec[-42]
DETECTOR(4, 3, 3) rec[-2] rec[-3] rec[-7] rec[-8] rec[-41]
DETECTOR(4, 5, 3) rec[-1] rec[-6] rec[-40]
OBSERVABLE_INCLUDE(0) rec[-21] rec[-22] rec[-23] rec[-24] rec[-25]
```

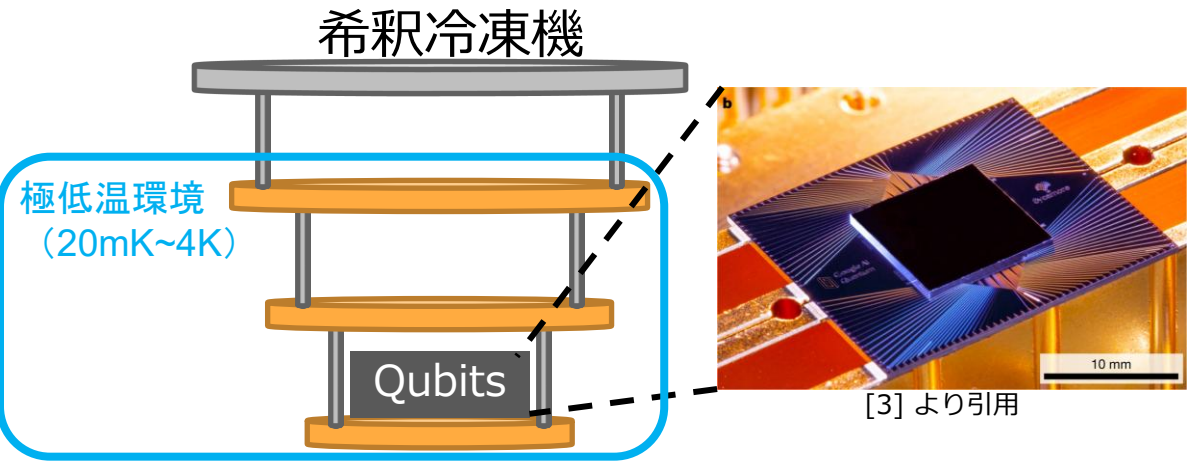


論理Z演算子（の1つ）をobservableとして指定

発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

超伝導量子コンピュータ



[3] より引用

現状の最高性能の超伝導量子コンピュータ[3]
- 量子ビット数: 50~100
- エラーレート: 0.1~1%
- Noisy Intermediate Scale Quantum (**NISQ**) device

- 超伝導量子ビット: 量子ビットの有望な実現方法の1つ
- 20 mK程度の極低温環境でのみ動作
  - 希釈冷凍機の中で動作

[3] Frank Arute, Kunal Arya, Ryan Babbush, et al. Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510 (2019).

超伝導量子ビット+極低温環境で動作する復号器

従来アーキテクチャ

復号器
（古典計算機）

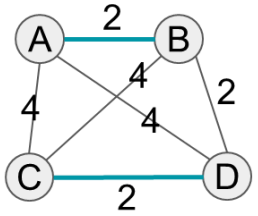
冷凍機

Error corr.

Qubit meas.

Qubits

グラフマッチング

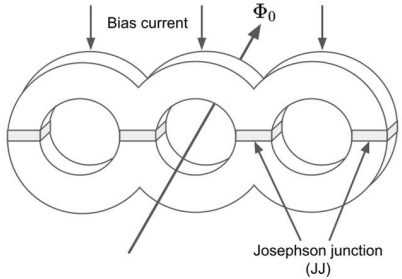


提案アーキテクチャ

復号器

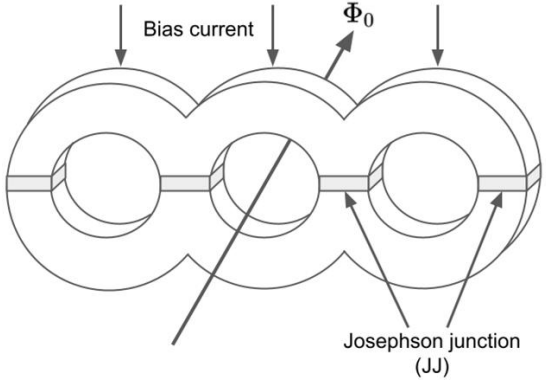
Qubits

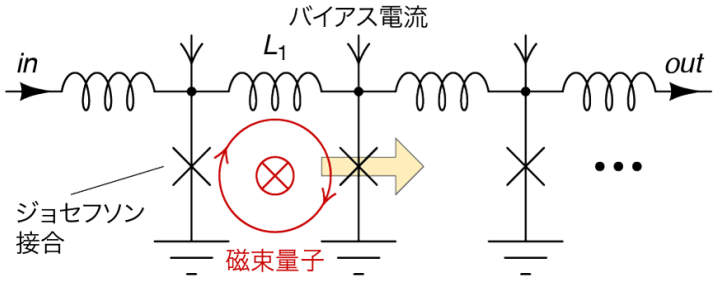
SFQ circuits



- 高速・低消費電力
- パルス論理
- RAMは高コスト

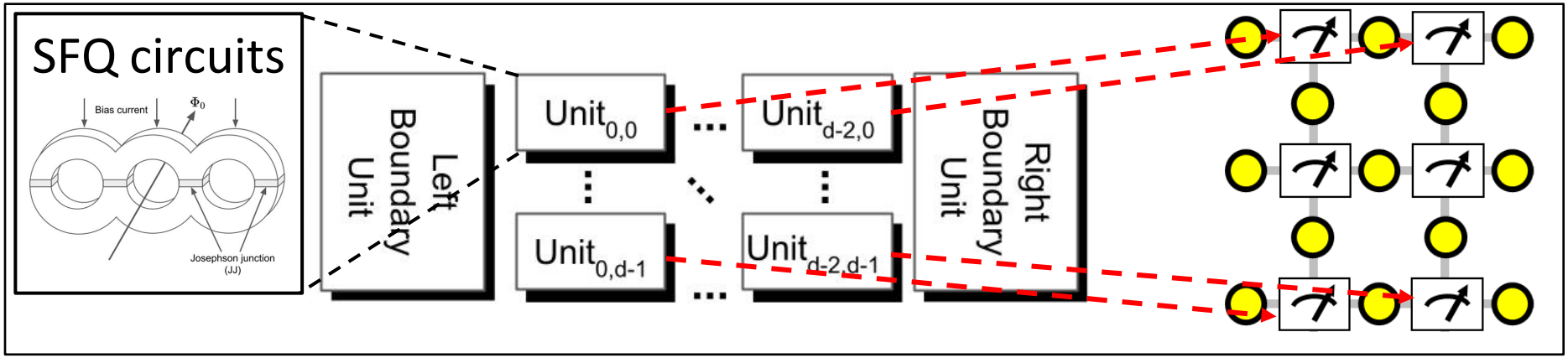
本研究のアプローチ: 超伝導回路による復号





- 単一磁束量子 (**SFQ**: Single Flux Quantum) 回路
- 超伝導リング内の磁束量子の有無で0 or 1を表現
- 4K程度の極低温環境でのみ動作
- CMOSに比べて高速・低消費電力
- 大規模なメモリの構築は難しい
  - 実行するアルゴリズムに工夫が必要 -> **分散処理方式のマッチングアルゴリズム**

提案手法: QECOOL

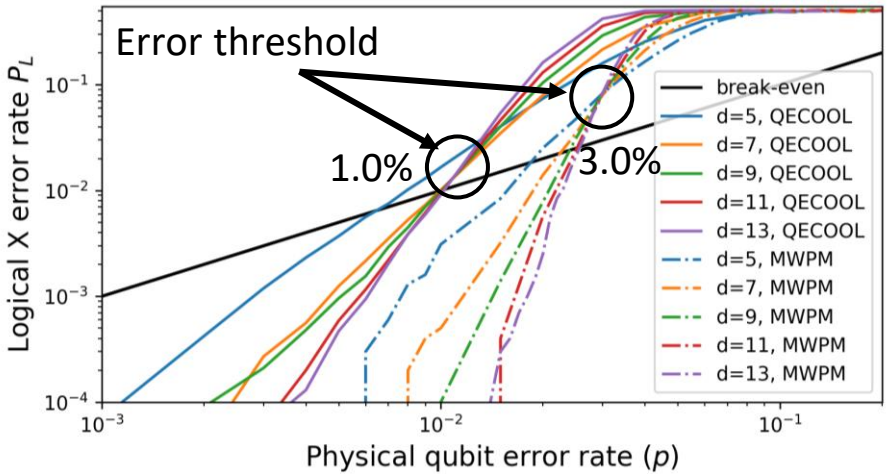


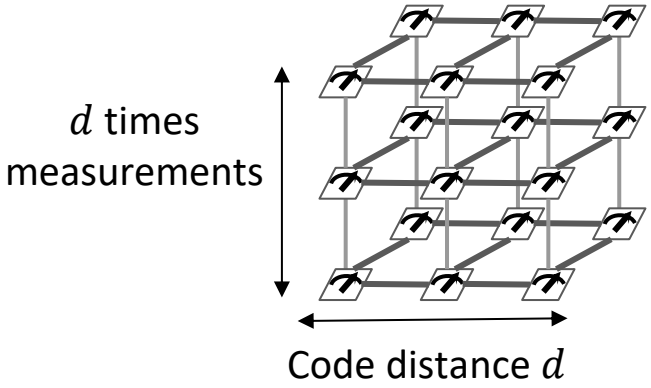
QECOOLのアーキテクチャ

- Quantum Error COrrection by On-Line decoding algorithm
- 大規模なRAMを必要としない**分散型のアーキテクチャ**
  - 補助量子ビットに1対1に対応する**Unit**を導入
  - Unit同士の3種類の信号伝播によりマッチング問題を解く

Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, Y. Tabuchi, “QECOOL: On-Line Quantum Error Correction with a Superconducting Decoder for Surface Code”, 58th IEEE/ACM Design Automation Conference. (DAC 2021)

QECOOLの誤り推定性能





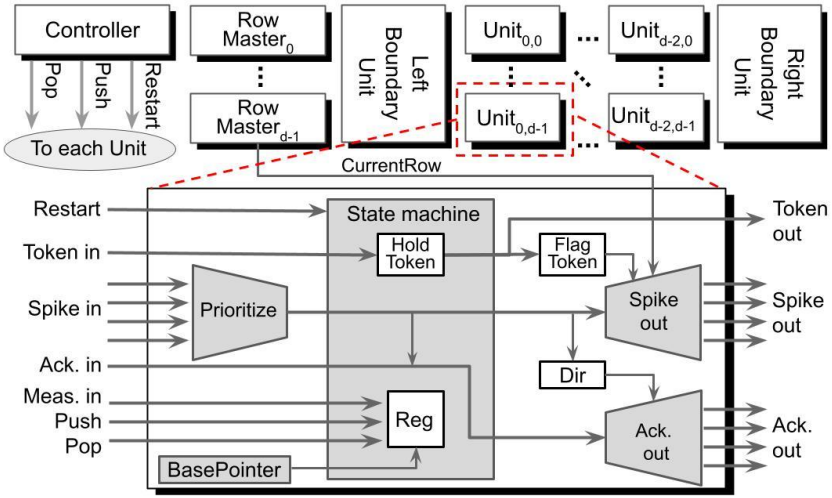
Target lattice shape

Experimental condition

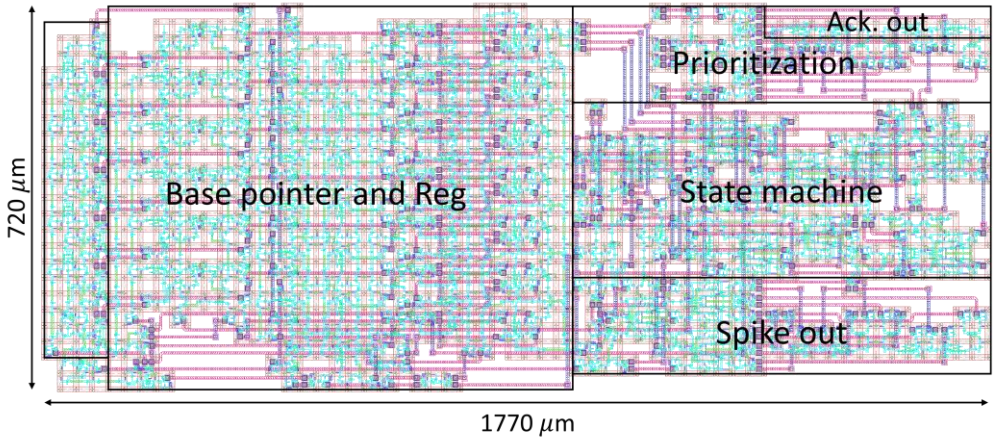
- Measurement process is performed **once every 1 μs**
- Each QECOOL Unit has a **7-bit** buffer to store syndrome values
- If buffer entry size is greater than $K = 3$, QECOOL is performed; otherwise, each Unit waits for measurement process
- MWPM operates with batch-QEC manner

- しきい値: QECOOL $p = 0.01$, MWPM $p = 0.03$

QECOOL誤り推定器のSFQ回路による実装



Architecture overview of QECOOL



SFQ design layout of QECOOL Unit

JJs: 3177	Area: 1.274 mm^2	Latency: 215 ps	Power cons.: **2.78 μW**

Decoder power consumption per one logical qubits

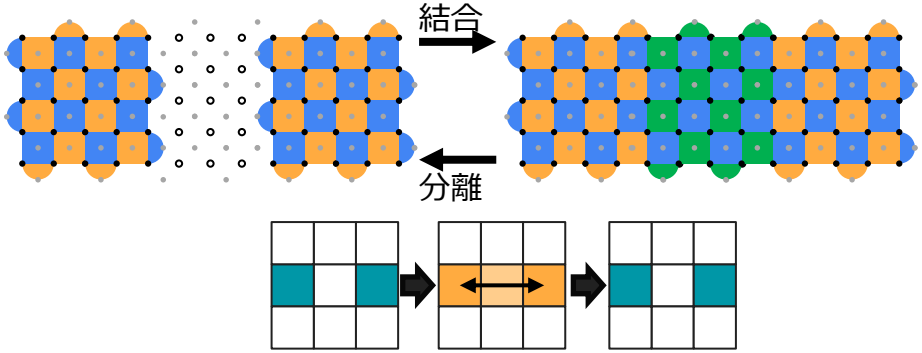
Suppose $d = 9$,

$$9 \times 8 \times 2 \times 2.78_{[\mu W]} \approx \boxed{400\ \mu\mathrm{W}}$$

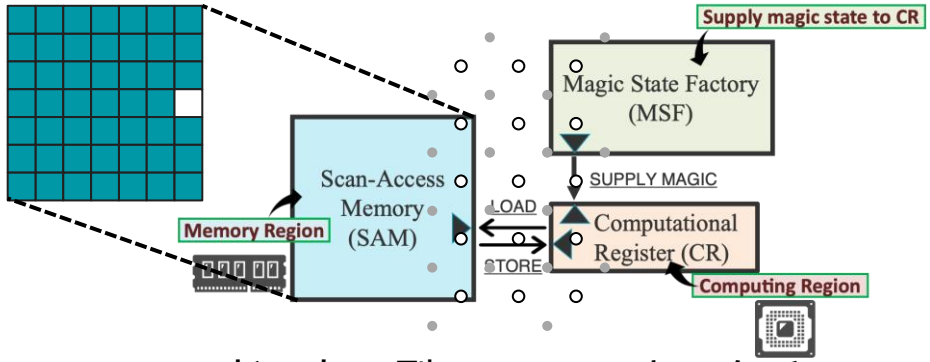
発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

2コマ目の概要



格子手術操作の例

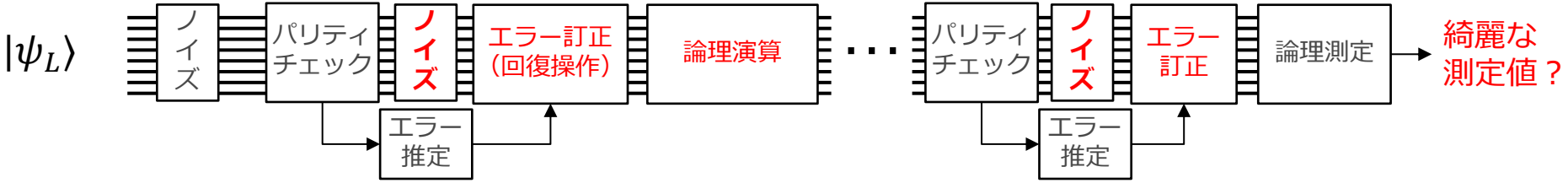


ロードストア型FTQCアーキテクチャ

- 表面符号で誤りを訂正しながら計算したい
  - 可能な限りエラー訂正は後ろ倒しにする（Pauli frame）
    - どうしても無理な場合は状態準備にコストを押し付ける
  - パリティチェックの枠組みを計算に活用（符号の変形）
- 格子手術に基づくFTQCは表面符号の拡大・縮小、結合・分離、移動
- 全ての論理量子ビットにいつでもアクセスできる必要はない
  → ロードストア型FTQCアーキテクチャ（HPCA2025）

表面符号上の論理演算

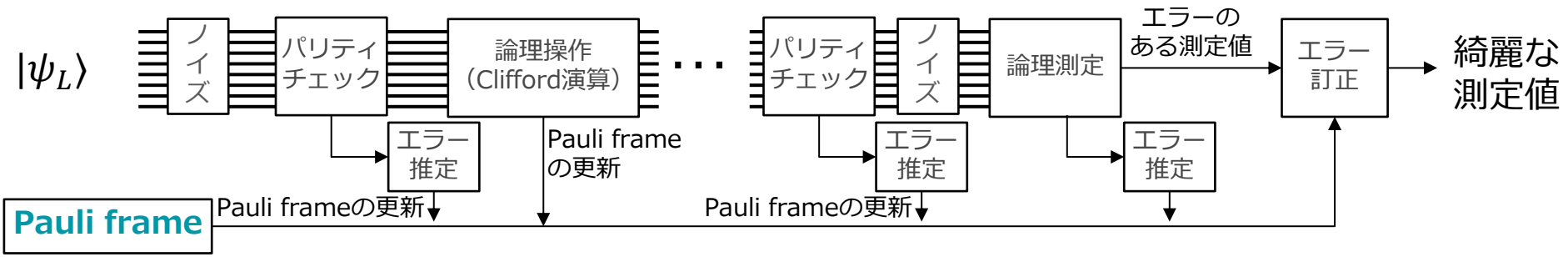
論理演算を行う場合のフローのイメージ（実際はうまくいかない）



- ナイーブな論理演算手順はうまくいかない
  - エラー推定中に新たなノイズが生じる可能性あり
  - エラー訂正操作自体がエラーを発生させる
  - パリティチェックの間隔を一定に保つ必要あり
    - 論理演算にかかる時間が符号の規模に応じて長くなるとまずい
- これを回避するためにエラーがいくつかの論理操作と可換であることを利用

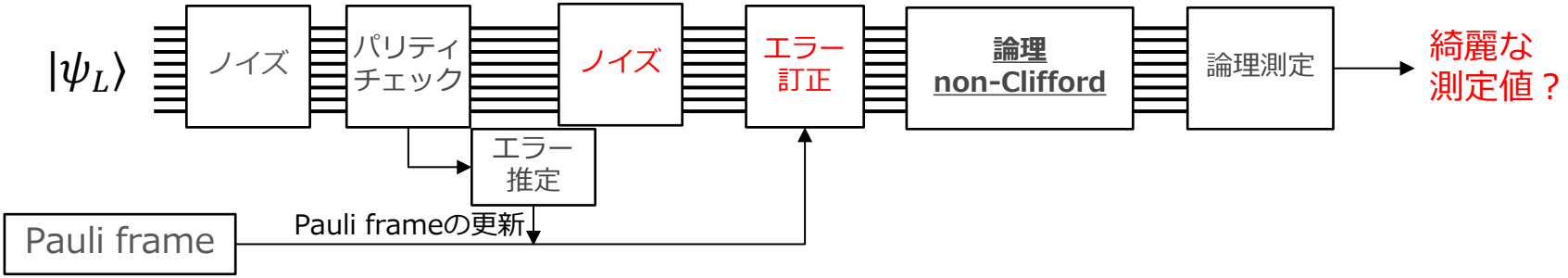
Pauli frameを用いたエラー訂正の後ろ倒し

実際の論理演算のフロー



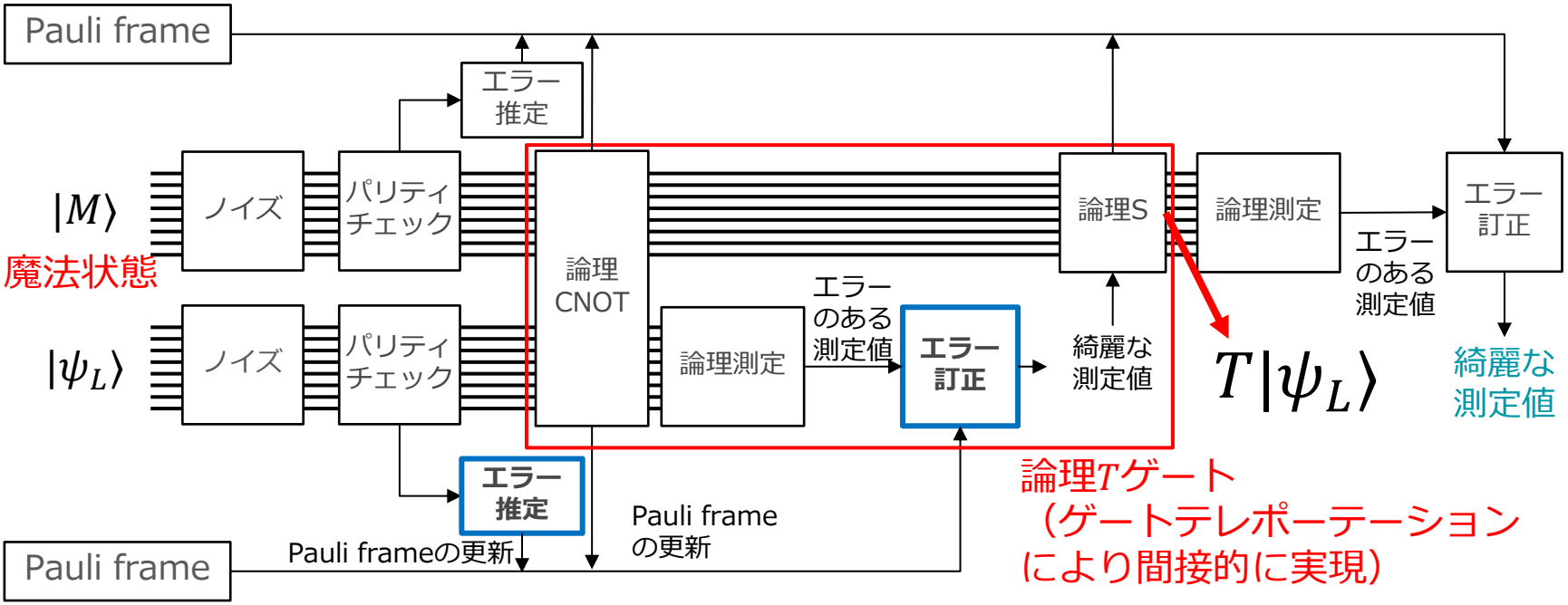
- Clifford演算はエラー訂正と可換であることを利用してエラー訂正操作を後回しにする
- 後で訂正する予定のPauli操作をPauli frameとして保持
- 論理Pauli演算はPauli frameの更新のみで実現

論理non-Clifford演算の難しさ



- 直接論理non-Clifford演算を行おうとするとPauli frameの書き戻しが必須となる
  - Non-Clifford演算は訂正操作と非可換
  - 復号している間に新たなノイズが乗ってしまい完全にエラーを解消できない
- Cliffordな論理操作で間接的にnon-Clifford演算を実現するゲートテレポーテーションが必須

ゲートテレポーテーションによる実現



- エラー率pの魔法状態$|M\rangle$は作れる（魔法状態注入）
- エラー率pの$|M\rangle$を15個集めてエラー率$35p^3$の$|M\rangle$を1つ作れる（**<u>魔法状態蒸留</u>**）

表面符号における論理演算

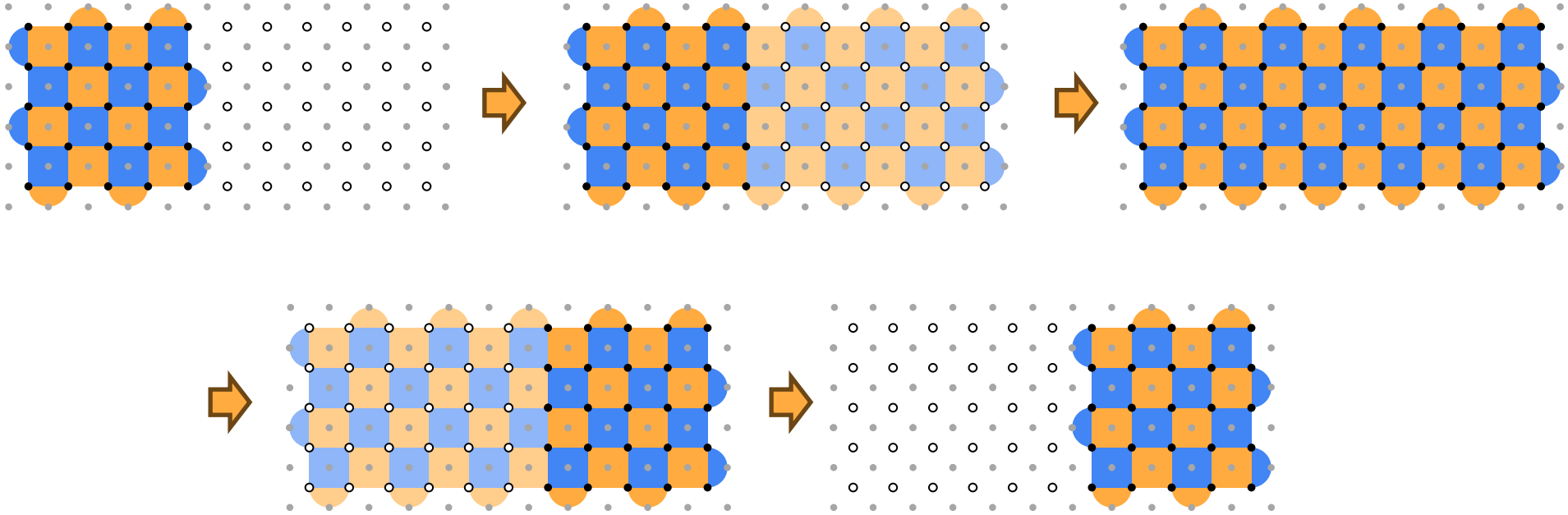
- 表面符号上のユニバーサルゲートセット$\{H, S, CNOT, T\}$
  - H:トランスバーサルな操作で実現
    （＋必要に応じて符号の変形）
  - S: 符号の変形により実現
  - $CNOT$: **格子手術**（符号変形の一種）による
    多体測定の組み合わせで実現

  Clifford演算

  - T: 魔法状態（特殊な乱数みたいなもの）の準備
    ＋Clifford操作で実現

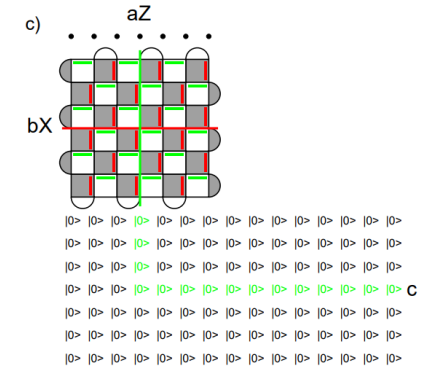
  Non-Clifford演算
- 特定の状態（$|0\rangle, |+\rangle, |M\rangle$）への初期化
- Pauli基底での論理測定

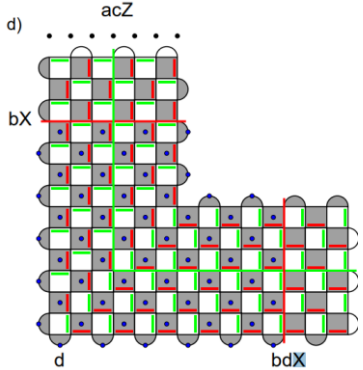
符号の変形

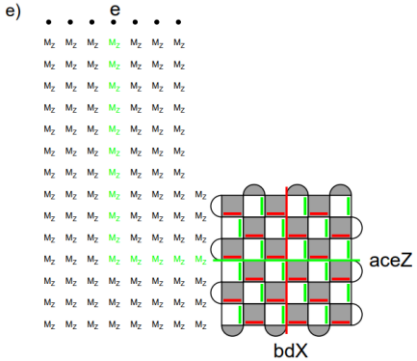


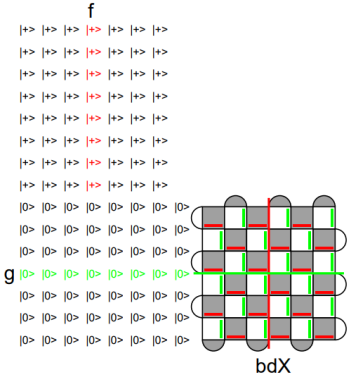
- パリティチェックの組を変えることで符号を変形
  - 論理状態を保ったまま変形 -> 拡大、縮小、**移動**、回転
  - 論理状態を変えながら変形 -> 論理S, CNOTゲート

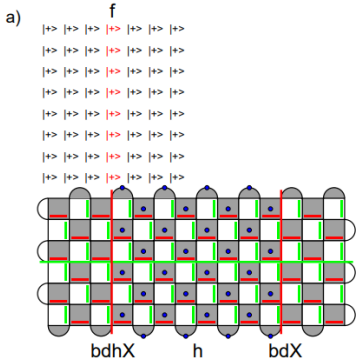
符号の変形を利用した格子の回転

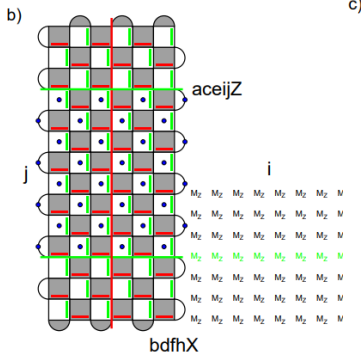


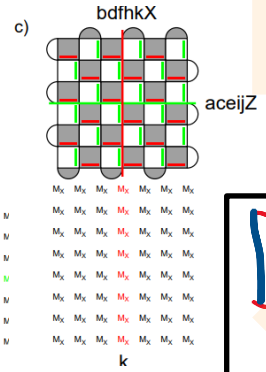


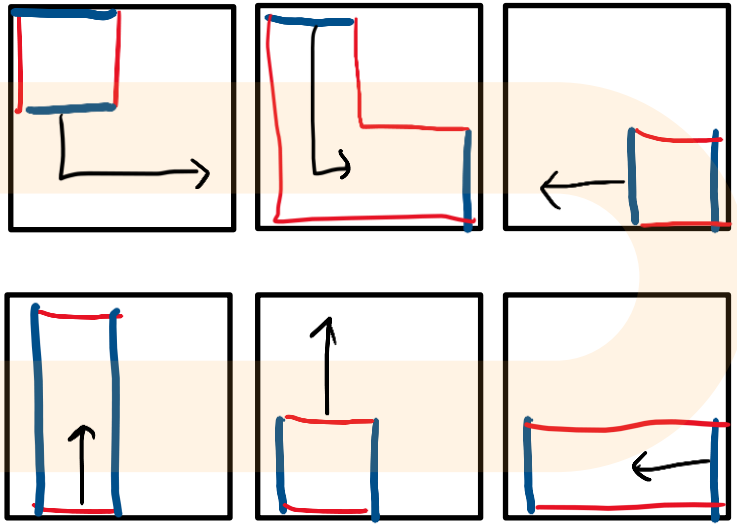




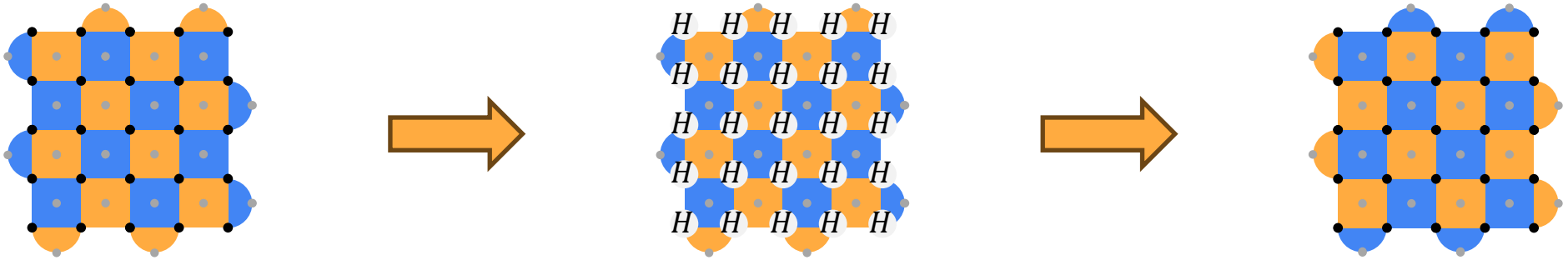








論理Hゲート



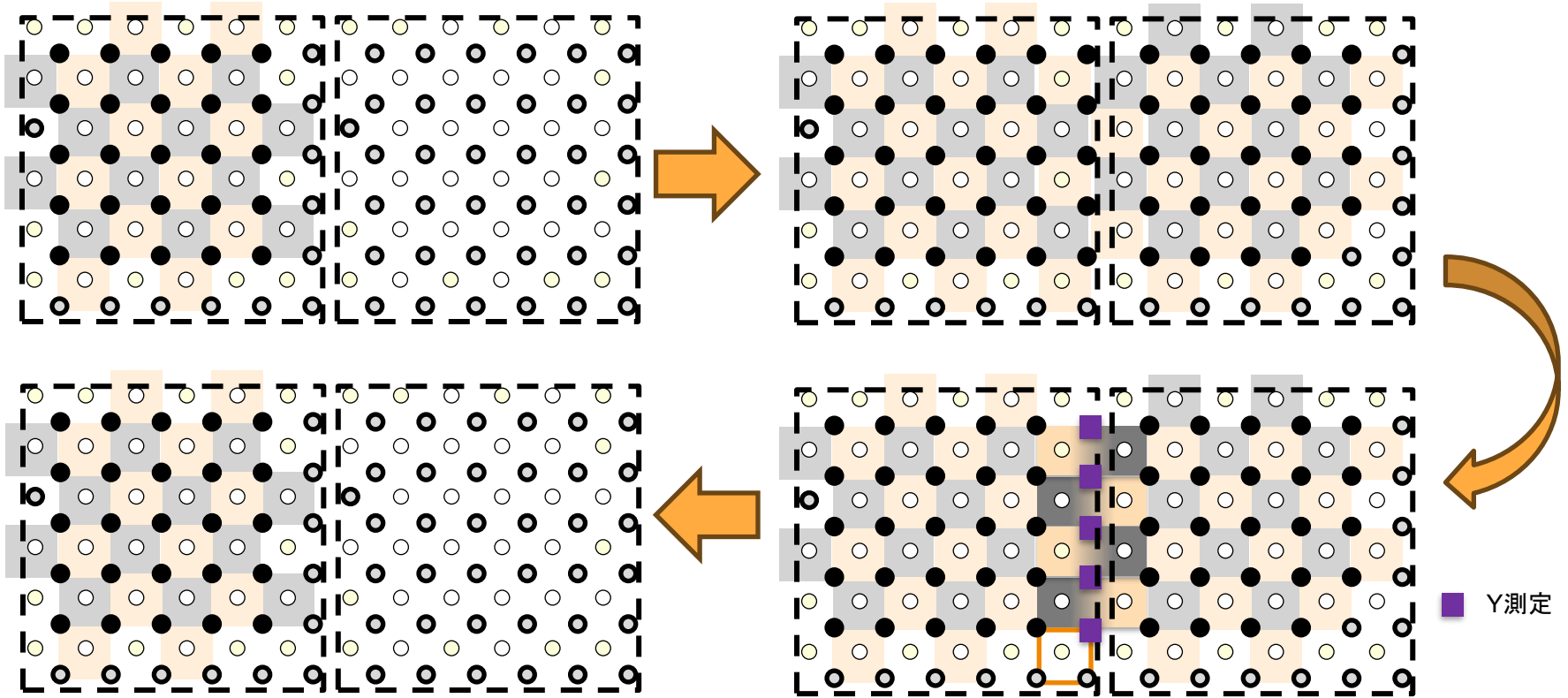
- Hはtransversalityがある→全データ量子ビットにHをかける
  - 符号距離によらず論理演算が定数時間で完了
- XとZのスタビライザが入れ替わるので、左右と上下の境界の意味が交換
  →必要に応じて回転操作で境界の変化を補正

論理Sゲート

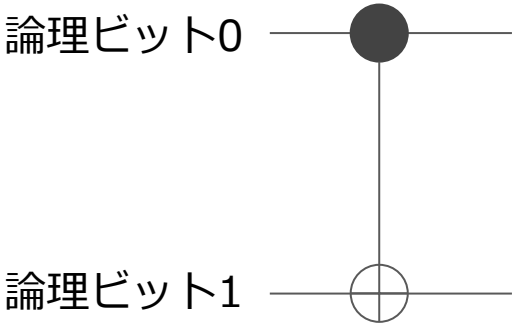
論理Sゲート (Twist)

B.J. Brown et al., PRX 7.2, 021029 (2017)

手続き　　符号を拡大しパリティ検査のパターンを何度か切り替えて元に戻す



多体測定の組み合わせによる$CNOT$ゲート

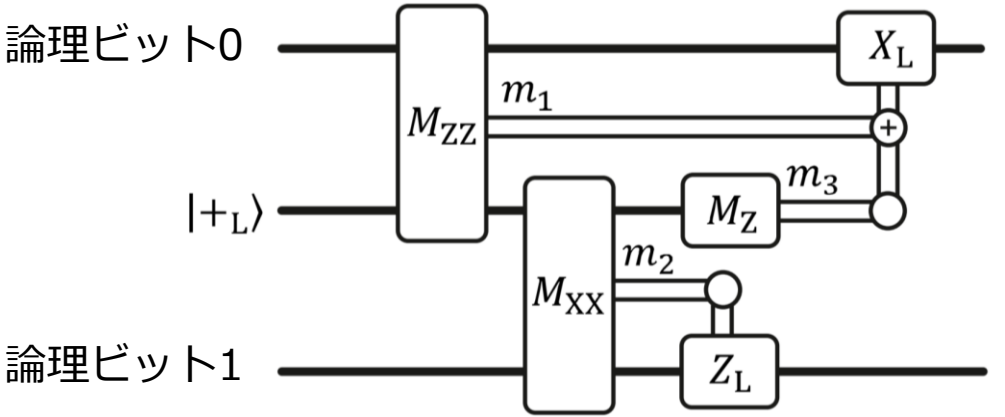


論理$CNOT$演算

等価な操作

横並びの
格子手術

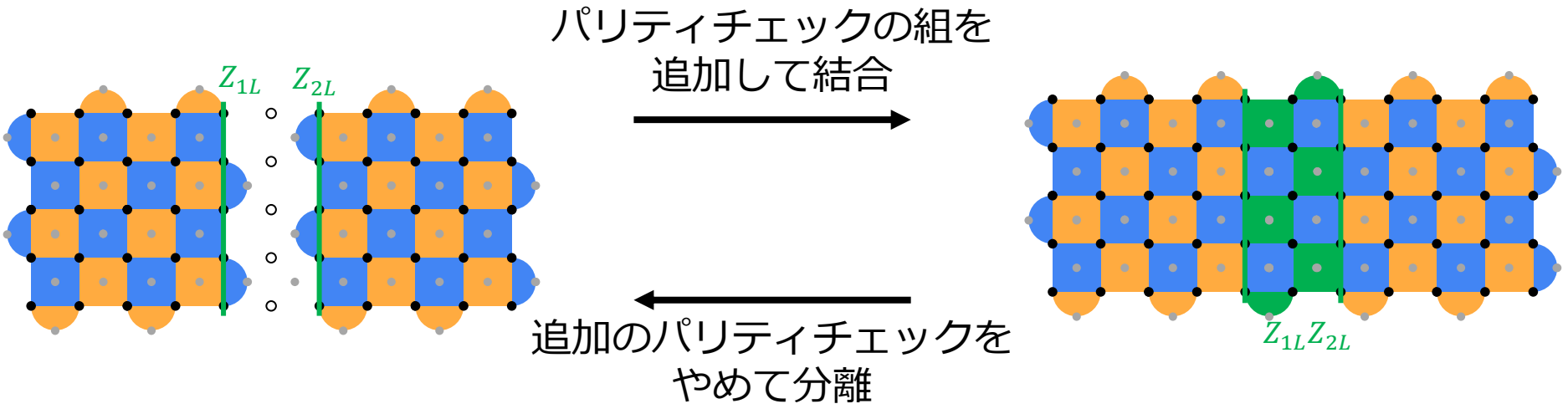
測定結果に応じた
論理X演算



縦並びの
格子手術

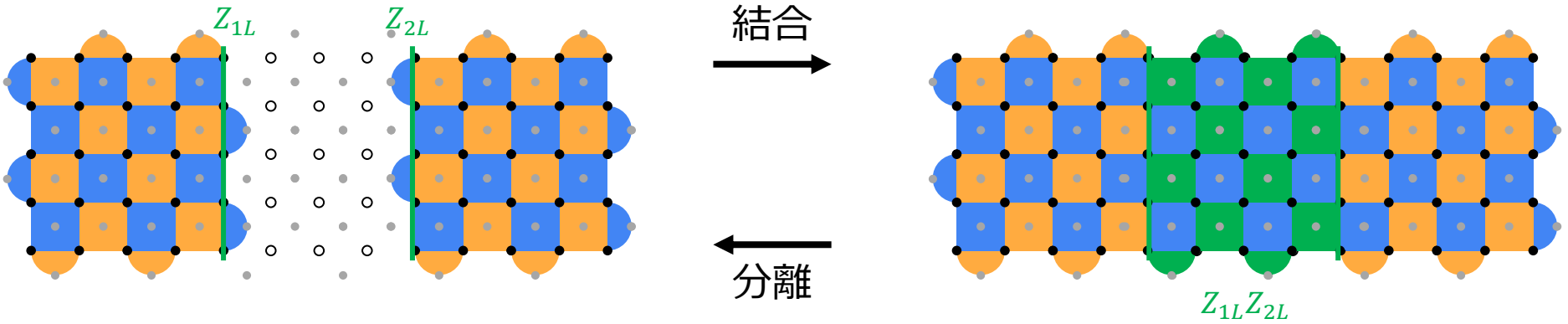
- 格子手術による多体Pauli測定ができれば
  論理$CNOT$を実現できる

格子手術による多体Pauli測定



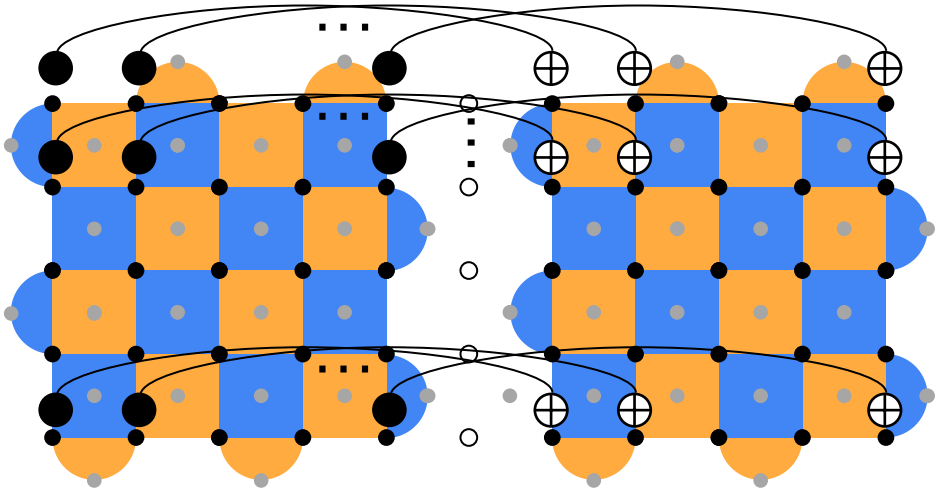
- 論理ビットの間のパリティチェックの結果が2論理ビットのZZ測定になる

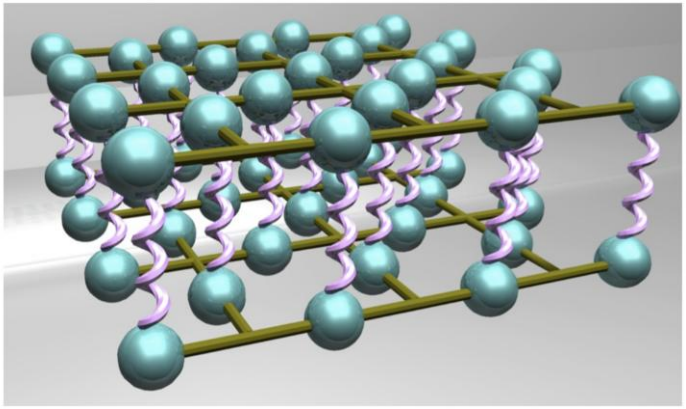
格子手術の経路の任意性



- 適切な境界同士の結合であれば経路によらず同じ操作になる
- 経路長が長くなると誤り訂正すべき箇所が増える->誤り推定は大変になる

参考：transversal CNOT



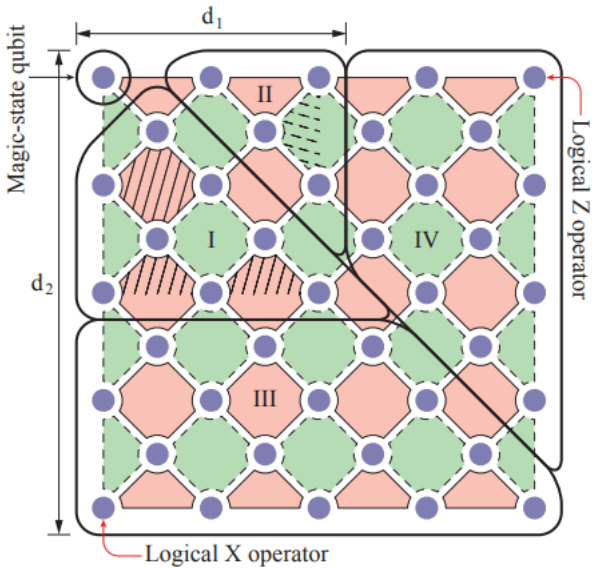


[1]より

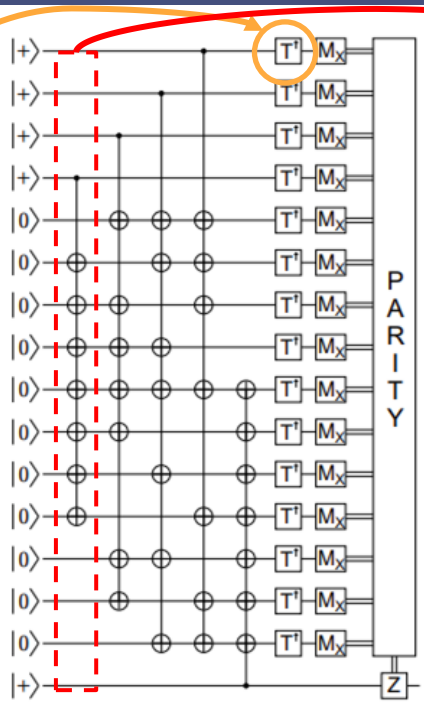
- $CNOT$はTransversalityを持つ：対応するデータ量子ビット間で$CNOT$をすると論理$CNOT$になる
- 隣接しない量子ビット間の相互作用が必要なためハードウェア要求が大きい
  - 中性原子量子コンピュータなど、量子ビット自体を動かせる場合には有望

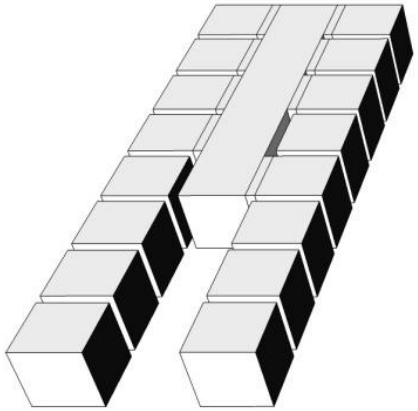
[1] Horsman, Dominic, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. "Surface code quantum computing by lattice surgery." *New Journal of Physics* 14, no. 12 (2012): 123011.

論理Tゲートのための魔法状態の準備



Magic state injection





格子手術による
Multi-target CNOT

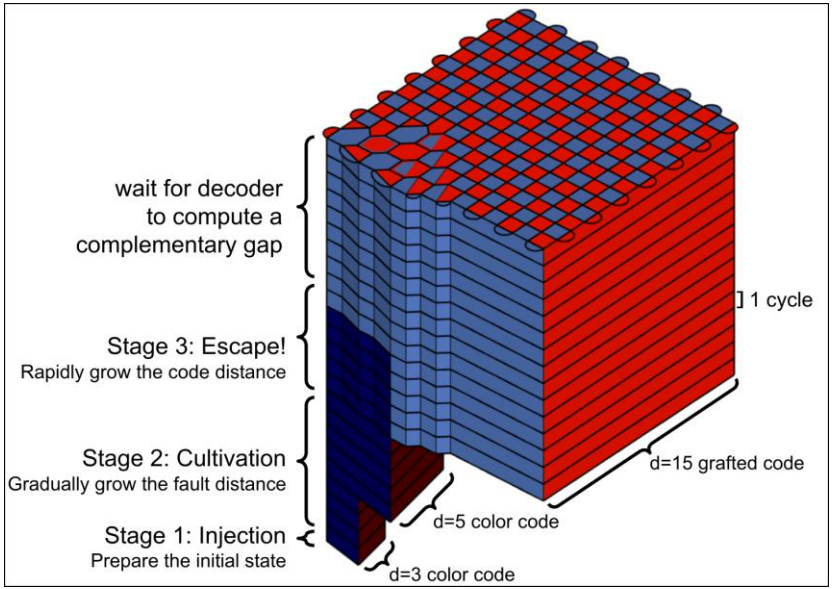
確率的に$|M\rangle$を埋め込む方法で、
成功するまで繰り返す
→上手く作れた状態だけを使う
（とは言ってもそこそこエラー率高い）

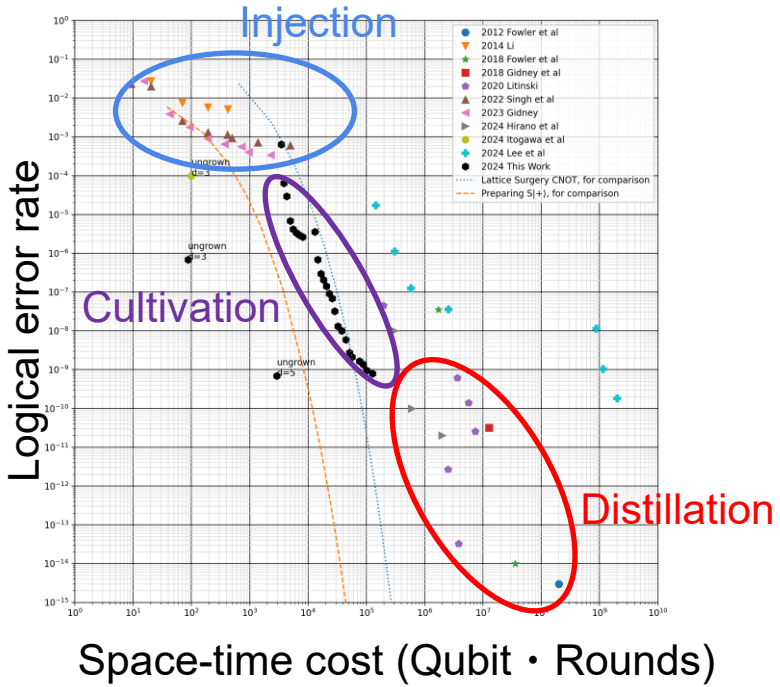
複数の汚い$|M\rangle$を入力にしたバカでかい格子手術を
行うと少数の綺麗な$|M\rangle$が得られる

余談: Tゲートは高価

- Non-Clifford演算を実行するのは大変…
- ひとまずClifford演算のみをサポートするFTQCを実現すればいいのでは？
  - No! Clifford演算のみの量子計算は古典計算機で多項式時間シミュレート可能=計算能力が同等
  - Gottesman-Knillの定理
- 量子計算にかかるコスト≒Tゲート数
  - 量子コンパイラはTゲートを可能な限り減らすのが大きな仕事の1つ

最近の進展：Magic state cultivation



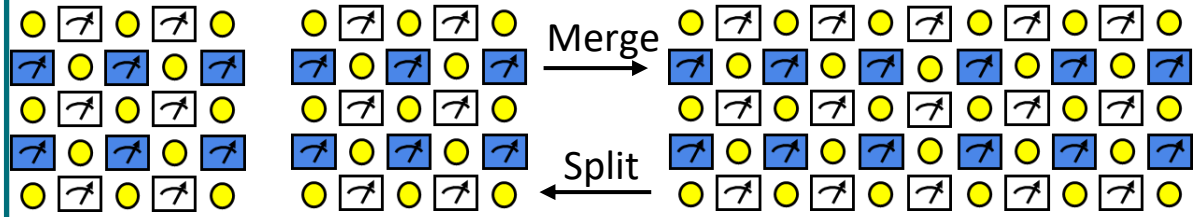


- 物理エラーレート$p = 10^{-3}$、論理エラーレート$p_L = 10^{-8}$くらいの魔法状態生成のコストを大幅に改善

Craig Gidney, Noah Shutty, and Cody Jones, Magic state cultivation: growing T states as cheap as CNOT gates, arXiv:2409.17595

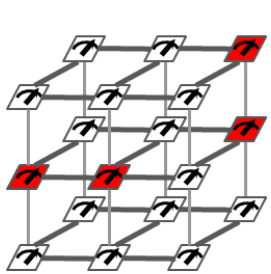
参考：QECOOLの格子手術向け拡張

Lattice surgery

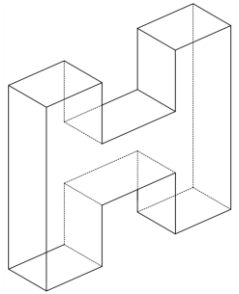


Framework to perform logical operations with SC-based QEC

Target lattice shape



Single logical qubit (QECOOL)

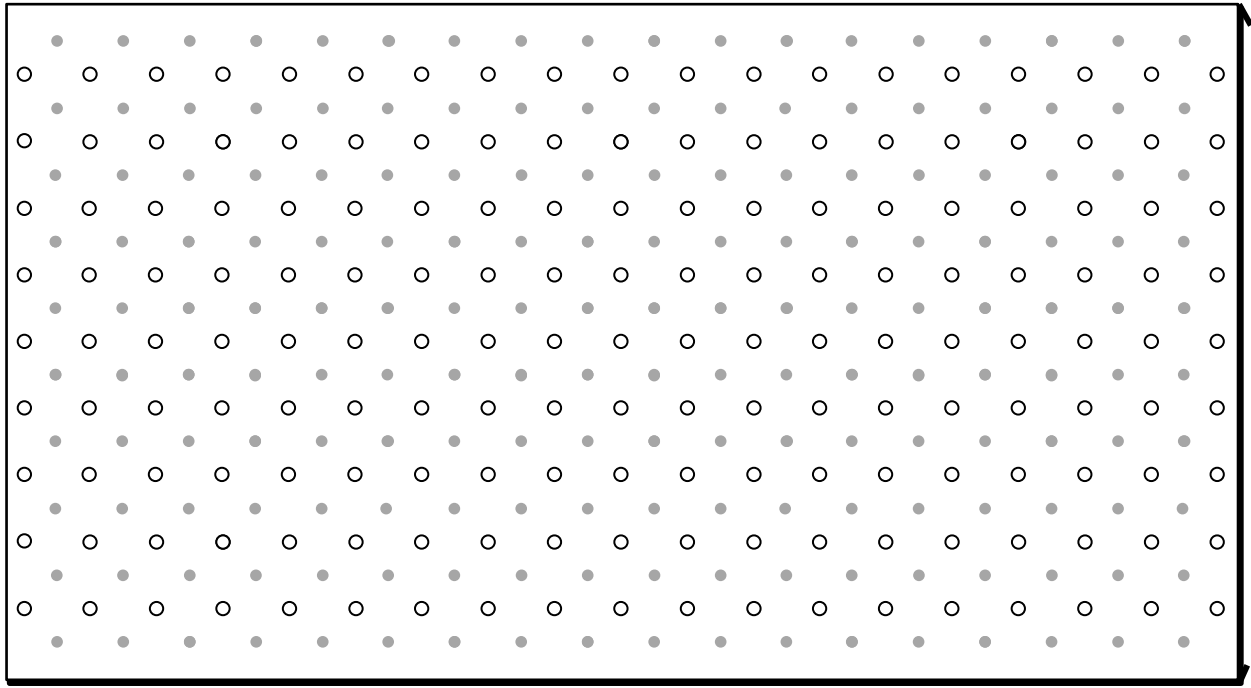


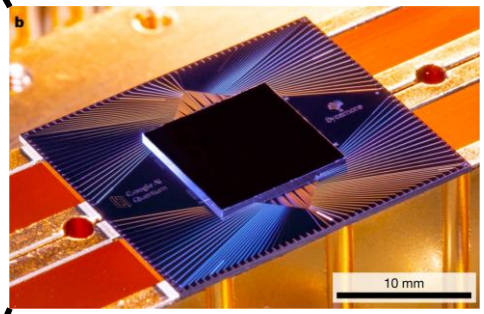
Lattice surgery (QULATIS)

- Extension of QECOOL for decoding of lattice surgery
  - Supporting logical operations of the universal quantum gate set $\{H, \mathrm{CNOT}, T\}$
- SFQ circuit design of QULATIS decoder is suitable for **online decoding in a cryogenic environment**

Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, Y. Tabuchi, “QULATIS: A Quantum Error Correction Methodology toward Lattice Surgery”, 28th IEEE International Symposium on High-Performance Computer Architecture. (HPCA 2022)

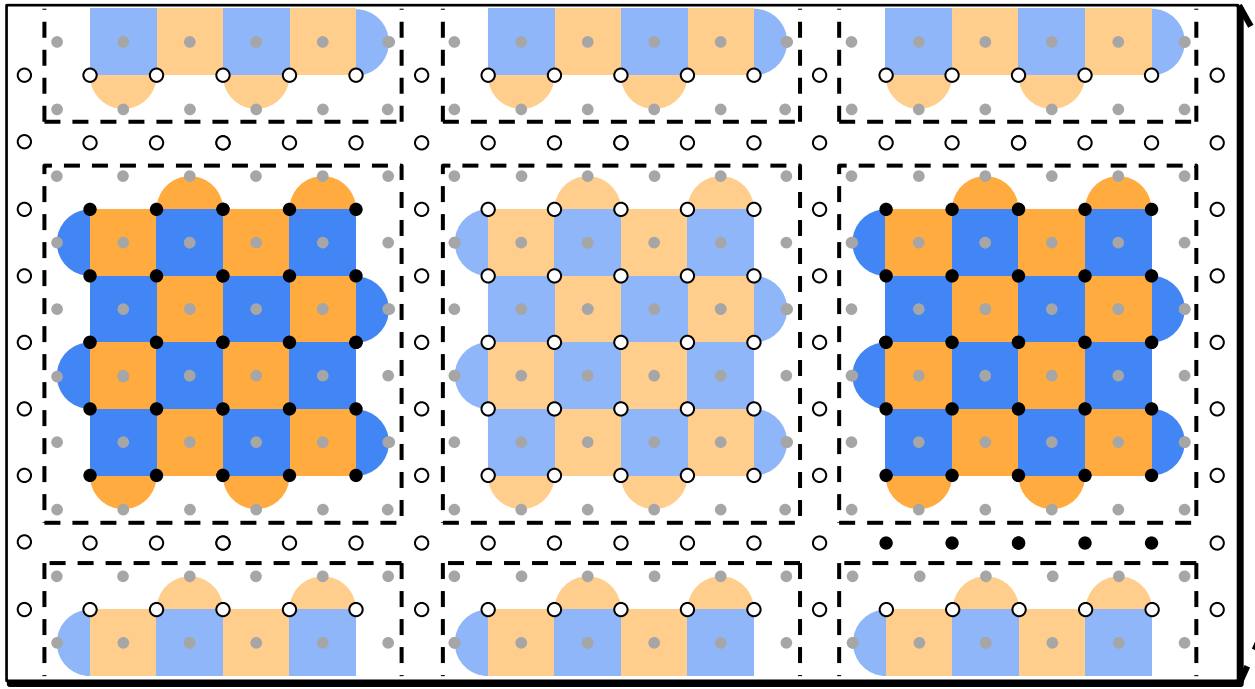
実際の量子ビットでの実装

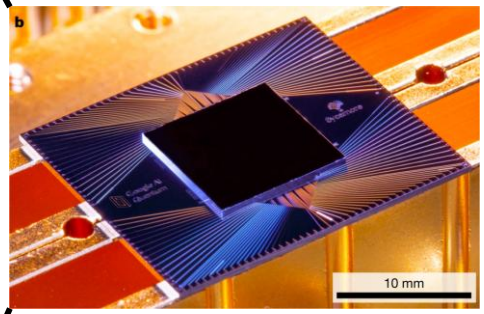




- 表面符号で計算を実行できるように区画を分ける
- 一部に論理状態を確保して一部は計算用に空けておく

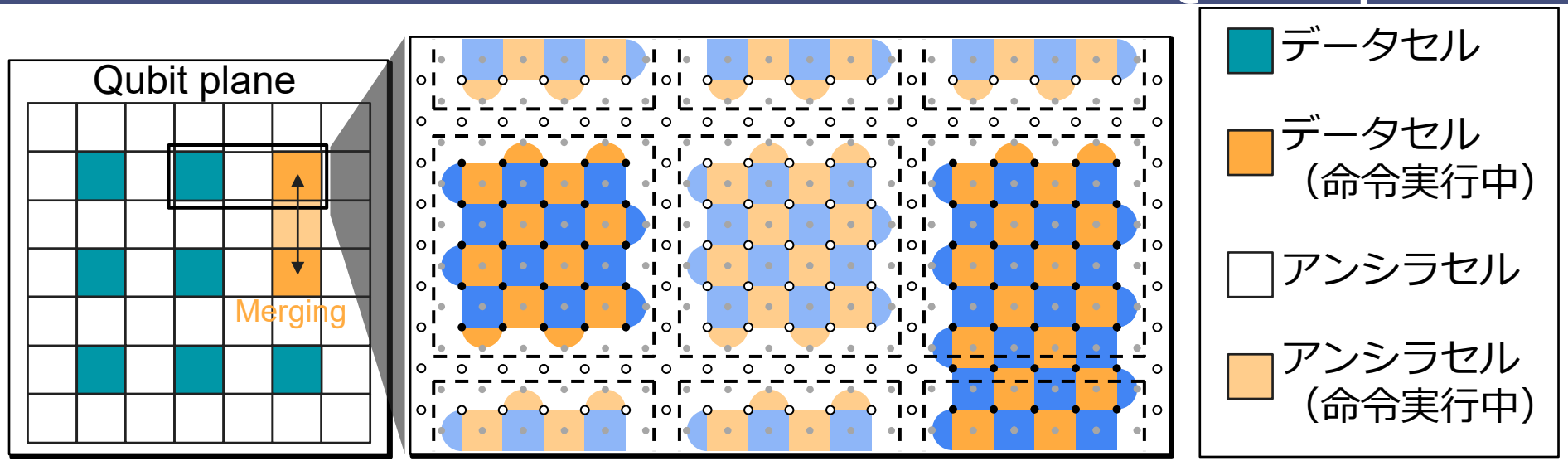
実際の量子ビットでの実装





- 表面符号で計算を実行できるように区画を分ける
- 一部に論理状態を確保して一部は計算用に空けておく

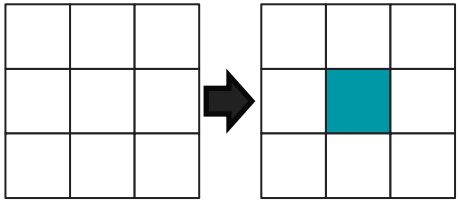
表面符号による論理演算を実行するQubit plane



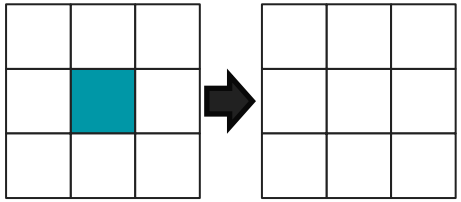
- 1つ1つのセルが符号距離dの表面符号を構成できる物理量子ビットを持つ
- **データセル**に計算用の論理状態を確保
- 空きスペース（**アンシラセル**）を使って
  計算（＝データセルの**拡大・縮小**、**結合・分離**）を行う

格子手術命令セットの一例

INIT_Z, INIT_X (0 code beat)

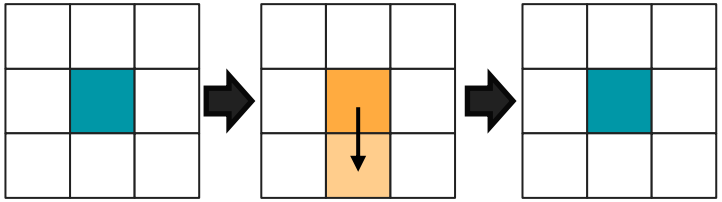


MEAS_X, MEAS_Z (0 code beat)



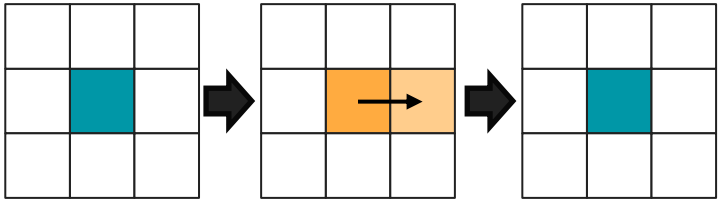
+Classical output

OP_H (3 code beats)



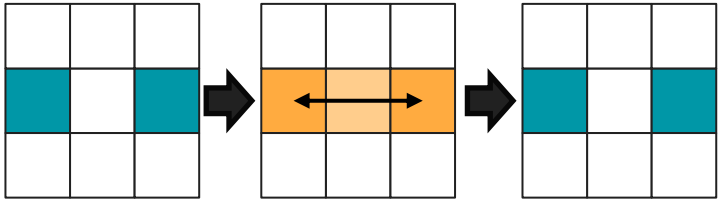
Any direction

OP_S (2 code beats)



Horizontal direction

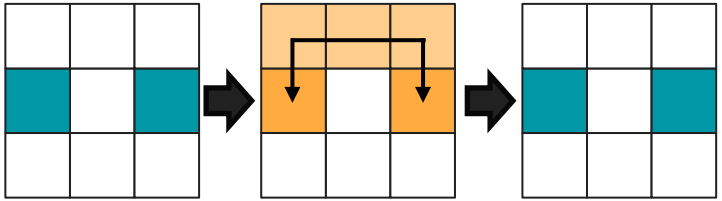
MEAS_ZZ (1 code beat)



Any path

+Classical output

MEAS_XX (1 code beat)

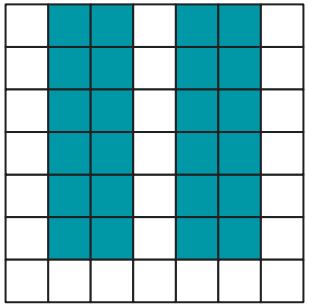


Any path

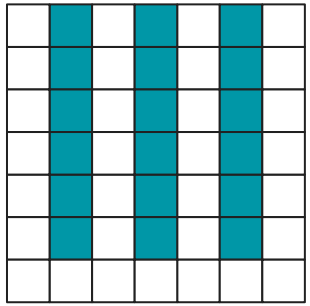
+Classical output

論理量子ビットのレイアウト（フロアプラン）

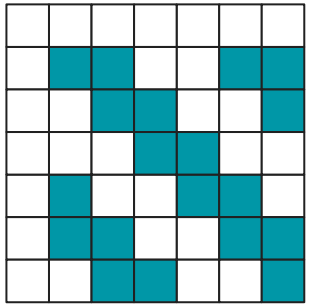
- 現状ランダムアクセスメモリがない
- 演算のために任意のデータセル間がアンシラセルを介して連結である必要がある



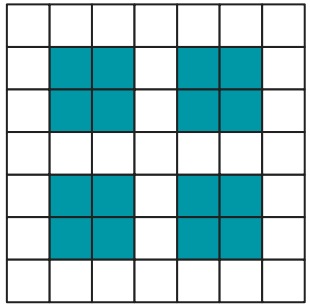
データセル密度
$R_{data} = 66\%$



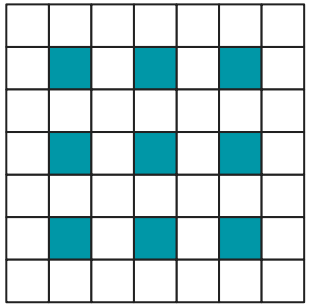
$R_{data} = 50\%$



$R_{data} = 50\%$



$R_{data} = 44\%$



$R_{data} = 25\%$

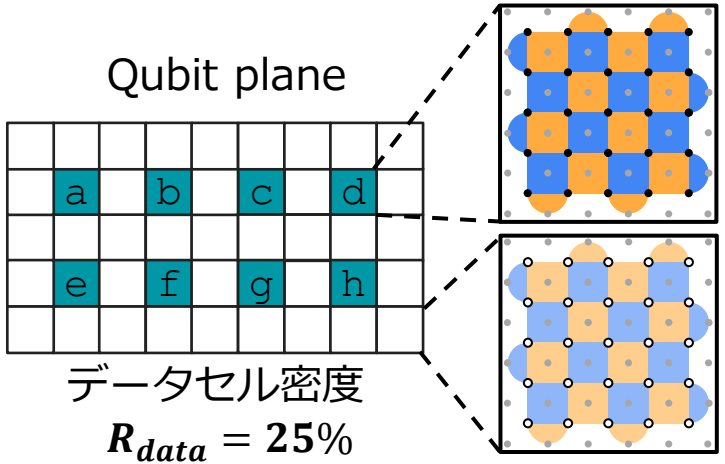
大 ← メモリ用スペース → 小

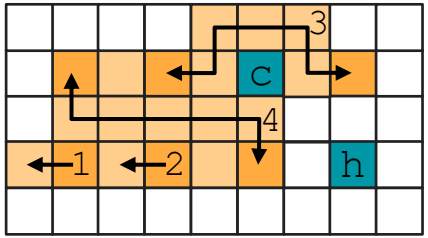
小 ← 演算並列度 → 大

格子手術の実行例

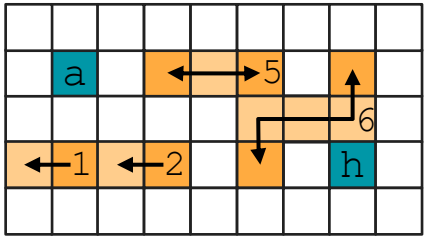
入力プログラム

```
1. OP_H e
2. OP_S f
3. MEAS_ZZ b, d
4. MEAS_XX a, g
5. MEAS_ZZ b, c
6. MEAS_XX d, g
7. MEAS_XX d, f
8. MEAS_XX g, h
9. MEAS_ZZ a, b
```

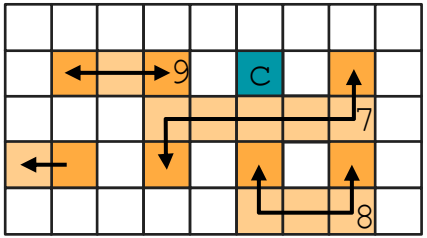




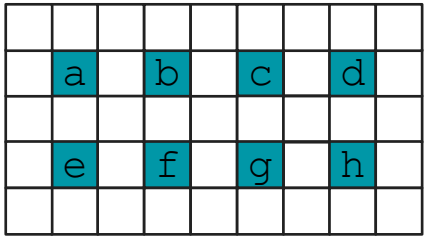
Code beat: 0
#Used cell: 8 + 14
Inst: 1, 2, 3, 4



Code beat: 1
#Used cell: 8 + 6
Inst: 1, 2, 5, 6



Code beat: 2
#Used cell: 8 + 10
Inst: 7, 8, 9

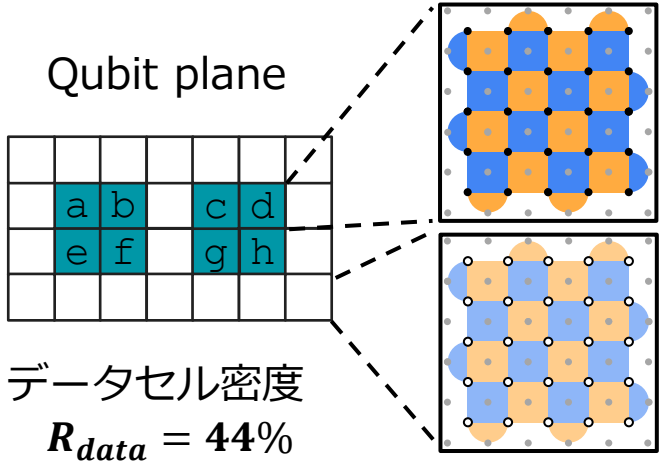


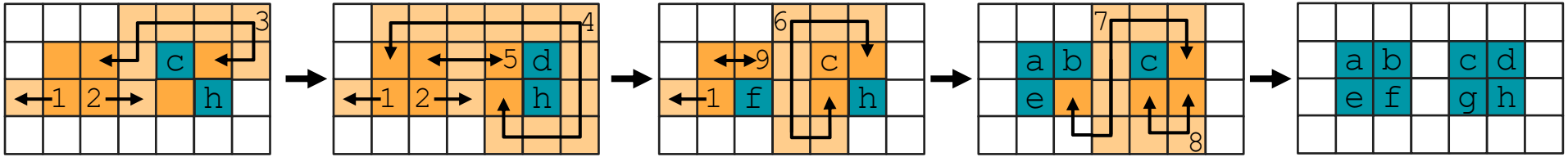
Code beat: 3
#Used cell: 8

格子手術の実行例：より高密度なデータセル配置

入力プログラム

```
1. OP_H e
2. OP_S f
3. MEAS_ZZ b, d
4. MEAS_XX a, g
5. MEAS_ZZ b, c
6. MEAS_XX d, g
7. MEAS_XX d, f
8. MEAS_XX g, h
9. MEAS_ZZ a, b
```





Code beat: 0
#Used cell: 8 + 8
Inst: 1, 2, 3

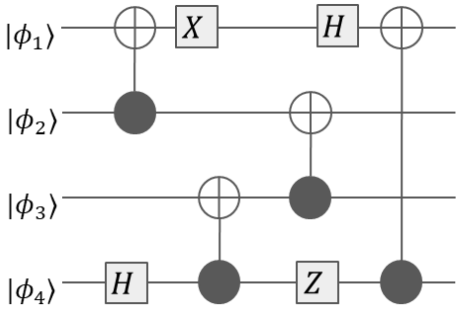
Code beat: 1
#Used cell: 8 + 14
Inst: 1, 2, 4, 5

Code beat: 2
#Used cell: 8 + 8
Inst: 1, 6, 9

Code beat: 3
#Used cell: 8 + 8
Inst: 7, 8

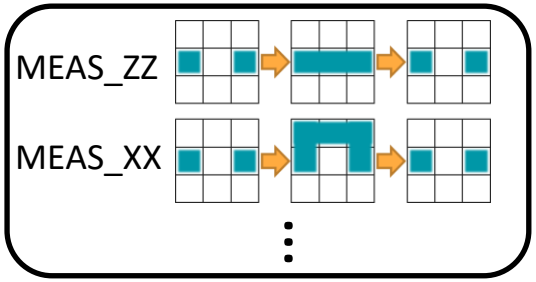
Code beat: 4
#Used cell: 8

従来の格子手術によるFTQCのコンパイルフロー



入力: 量子プログラム

格子手術命令列への変換

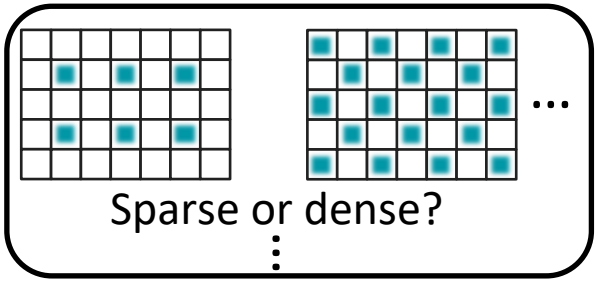


格子手術命令セット

```
MEAS_XX 2, 5
OP_H      6
MEAS_ZZ 3, 4
⋮
```

格子手術命令列

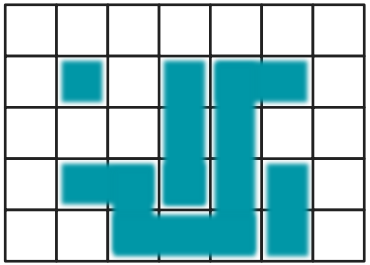
最適経路探索



2次元グリッド状に
配置された論理ビット

論理量子ビット割当

配置最適化

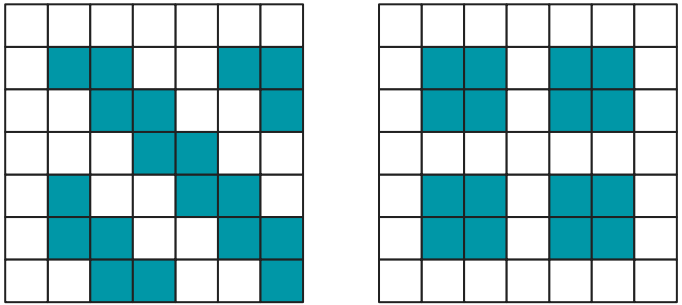


出力: 論理ビットの**割当**
および結合（**演算**）手順

発表内容

- 量子計算機アーキテクチャの研究動向
  - 計算機アーキテクチャの研究対象・隣接分野
  - 計算機アーキテクチャ分野における量子関連の研究動向
- 計算機系のための表面符号入門（1コマ目）
  - 量子誤り訂正の基礎
  - 表面符号の誤り推定処理のグラフ問題への帰着
  - Stim+Crumbleを用いた表面符号シミュレーション
  - 極低温環境での表面符号の誤り推定処理（上野の博論, DAC'21, HPCA'22）
- 表面符号＋格子手術に基づく誤り耐性量子計算（2コマ目）
  - ロードストア型FTQCアーキテクチャ（HPCA2025）
- HPCA2026投稿中論文について（3コマ目）
- まとめ

ロードストア型誤り耐性量子計算機アーキテクチャ



$R_{data} = 50\%$　　$R_{data} = 44\%$

従来のフロアプラン

- なるべく少ない量子ビット数でFTQCを実現したい
  → 全体のセルの数は少ないほど嬉しい
- 従来のフロアプラン：データセル位置固定、いつでもアクセスできる
  → データセルの比率（メモリ密度）は高々50%
- 着眼点：「任意のデータセルにいつでもアクセスできる」ことは必須なのか？→ NO!!

ロードストア型誤り耐性量子計算機アーキテクチャ

2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)

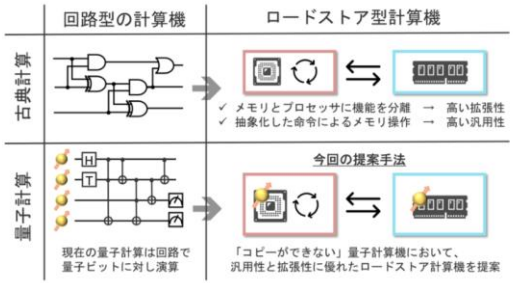
LSQCA: Resource-Efficient Load/Store Architecture for Limited-Scale Fault-Tolerant Quantum Computing

Takumi Kobori†, Yasunari Suzuki‡, Yosuke Ueno*, Teruo Tanimoto**, Synge Todo†, and Yuuki Tokunaga‡
The University of Tokyo†, NTT Corporation‡, RIKEN*, Kyushu University**
takumi.kobori@phys.s.u-tokyo.ac.jp, yasunari.suzuki@ntt.com, yosuke.ueno@riken.jp,
tteruo@kyudai.jp, wistaria@phys.s.u-tokyo.ac.jp, yuuki.tokunaga@ntt.com

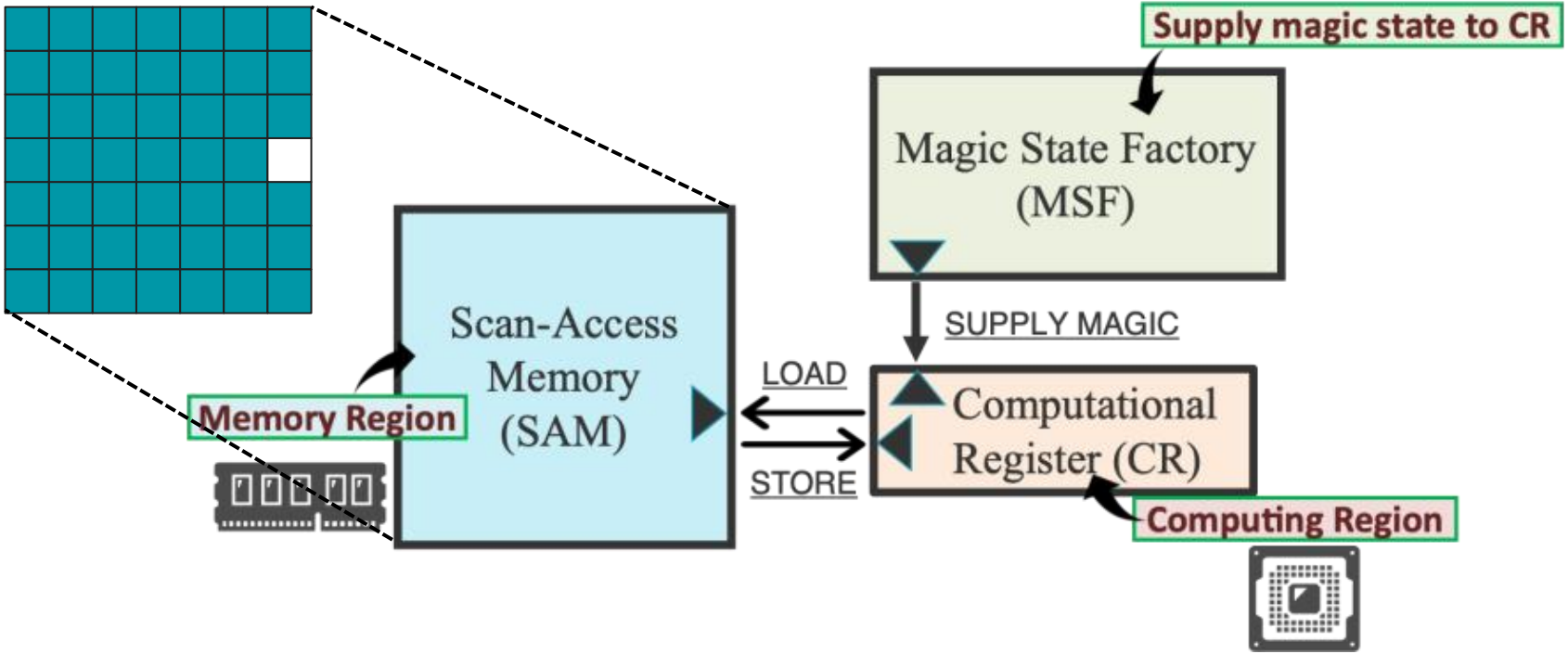
メモリとプロセッサを分離した新たな量子コンピュータのアーキテクチャを提案

研究成果　2025/03/05

東京大学大学院理学系研究科小堀拓生大学院生（当時日本電信電話インターン生）と藤堂眞治教授、日本電信電話株式会社の鈴木泰成研究員と徳永裕己研究員、理化学研究所量子コンピューター研究センターの上野洋典基礎科学特別研究員、そして九州大学大学院システム情報科学研究院の谷本輝夫准教授らによる研究グループは、従来の計算機の基本設計であるロードストア型計算機の考え方を量子計算機に適用した、新たな誤り耐性量子計算のアーキテクチャを提案しました。本技術は、プログラムの高い移植性（他の環境への移行のしやすさ）と高効率な量子ハードウェアの活用を可能とするものであり、有用な量子計算の早期実現を加速することが期待されます。



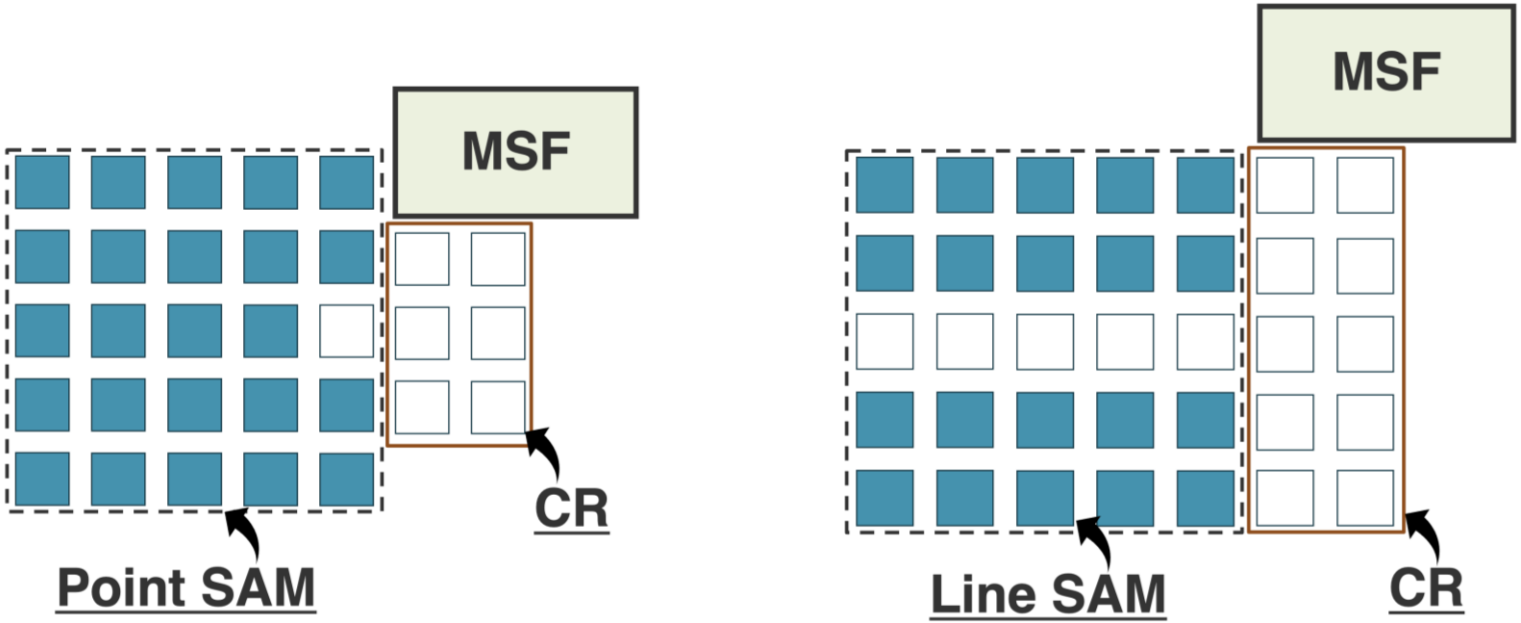
ロードストア型誤り耐性量子計算機アーキテクチャ



- 高密度にデータセルを配置するメモリ領域を構成
- データセルの移動を活用して計算領域に読み出し

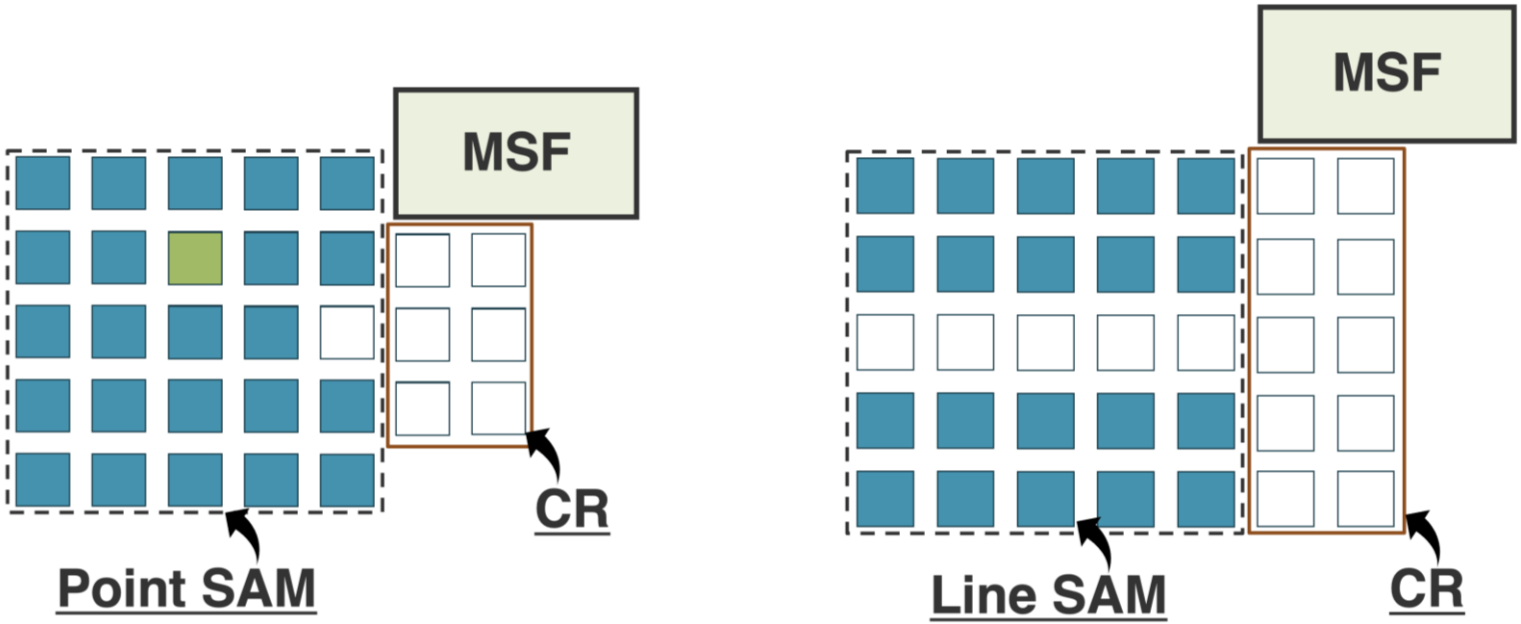
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



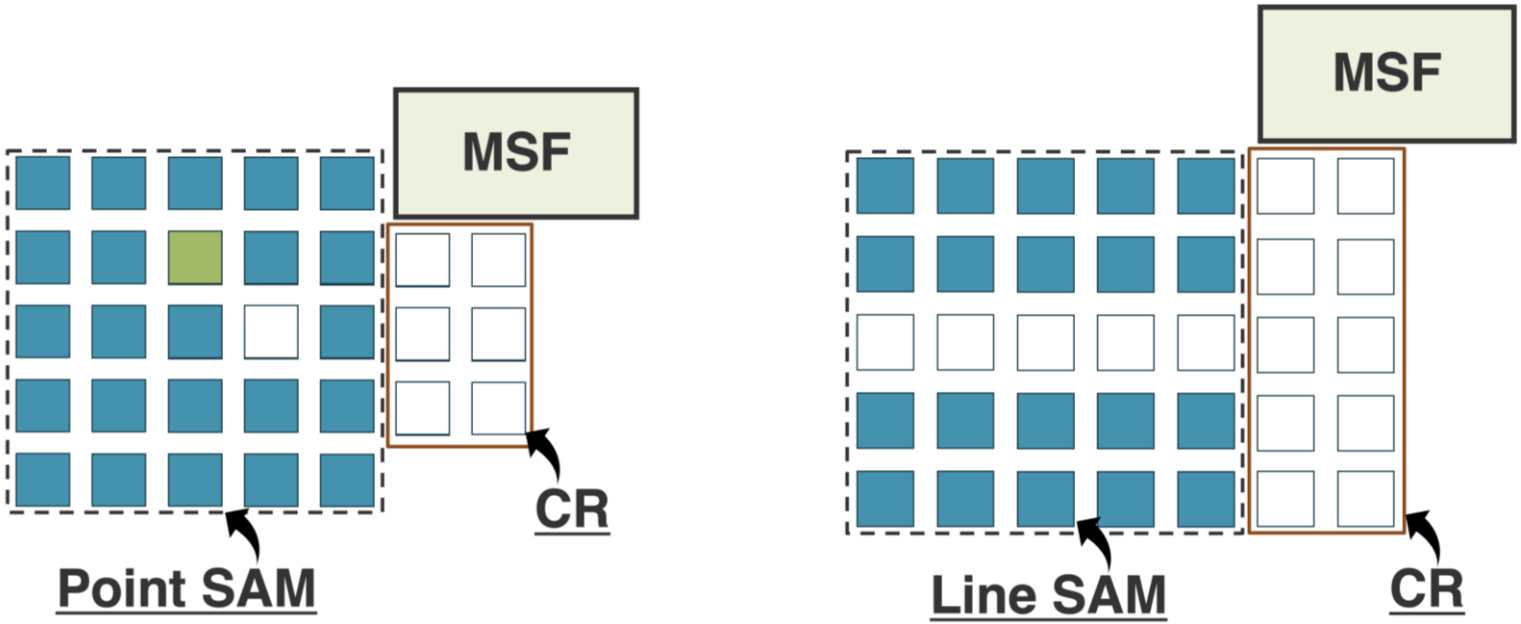
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



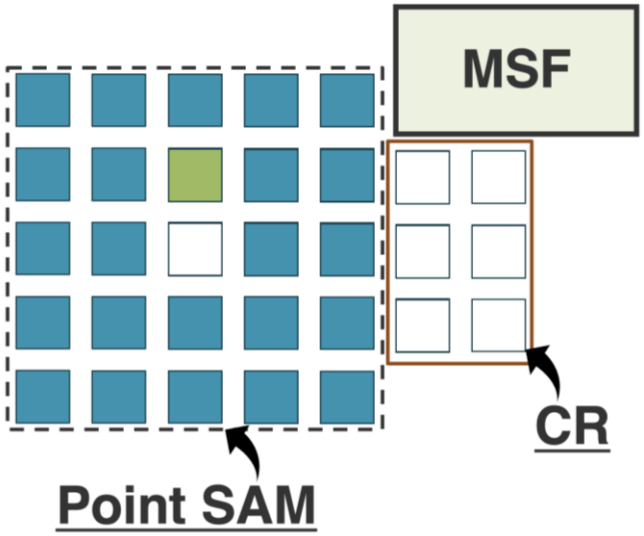
ロードストア型誤り耐性量子計算機アーキテクチャ

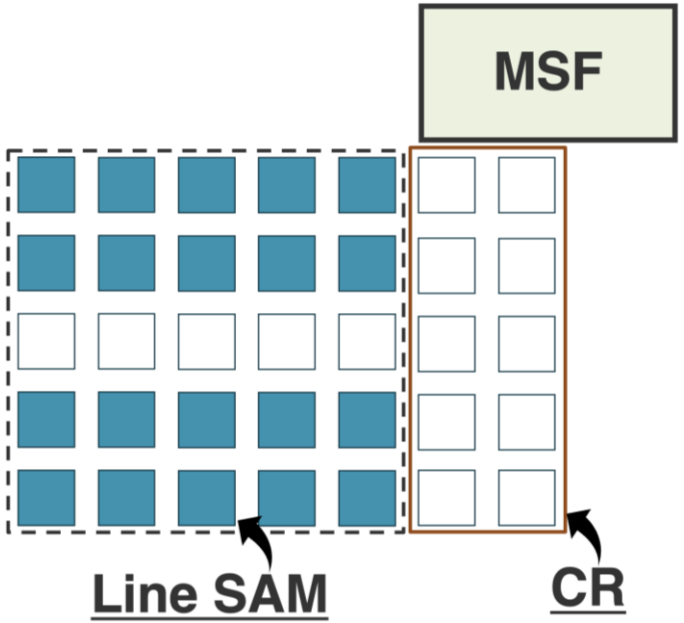
- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

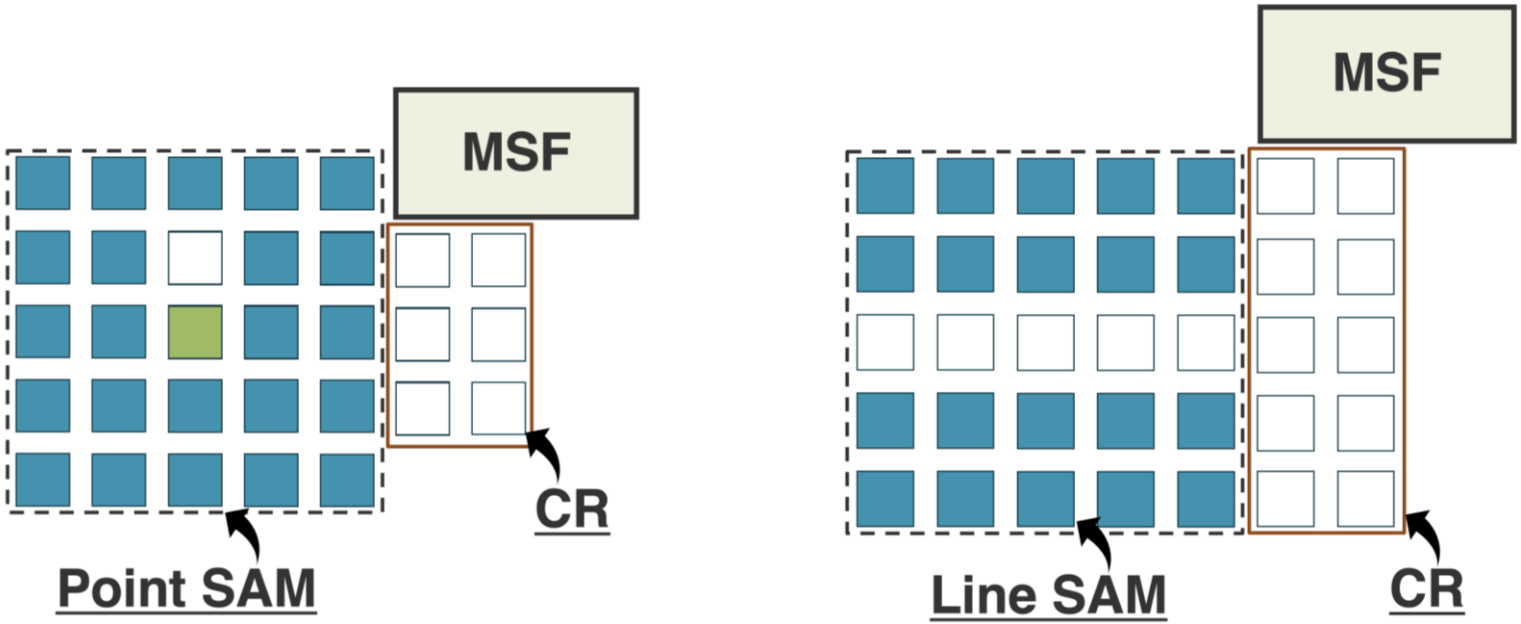
- Scan-Access Memory





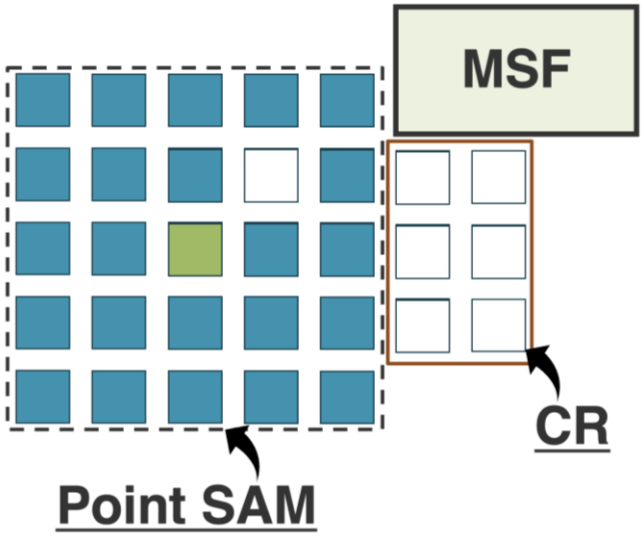
ロードストア型誤り耐性量子計算機アーキテクチャ

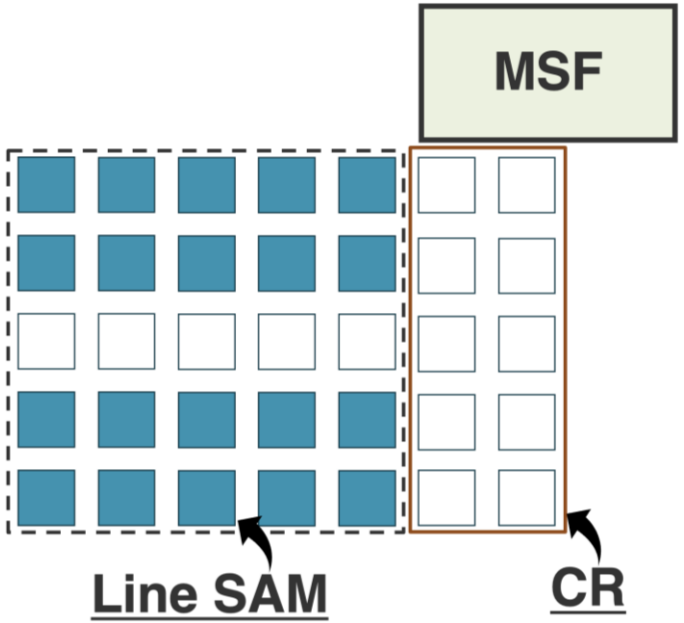
- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

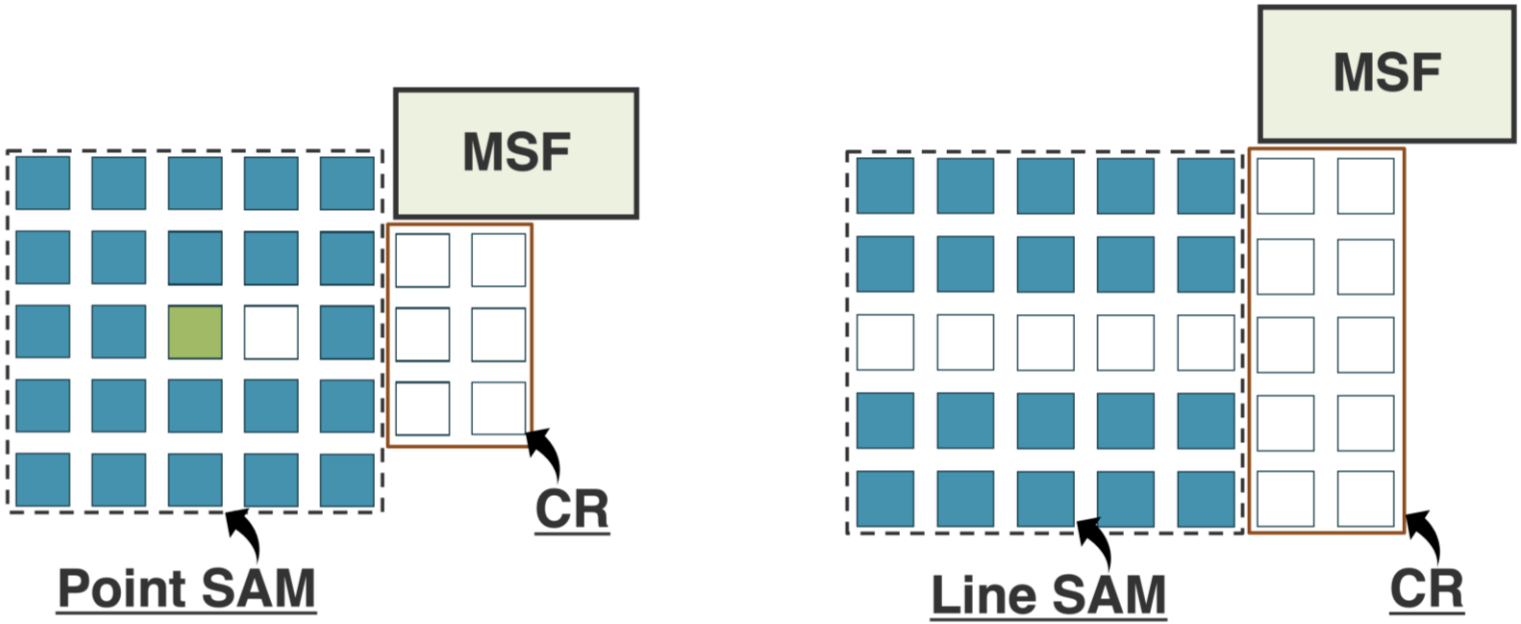
- Scan-Access Memory





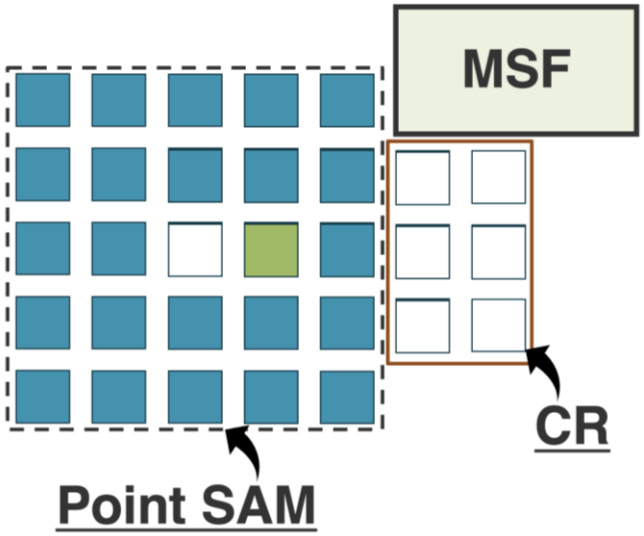
ロードストア型誤り耐性量子計算機アーキテクチャ

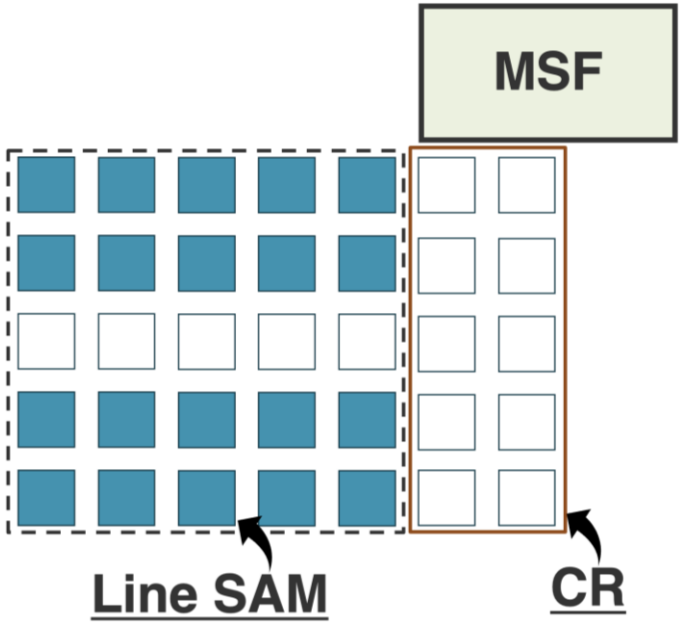
- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

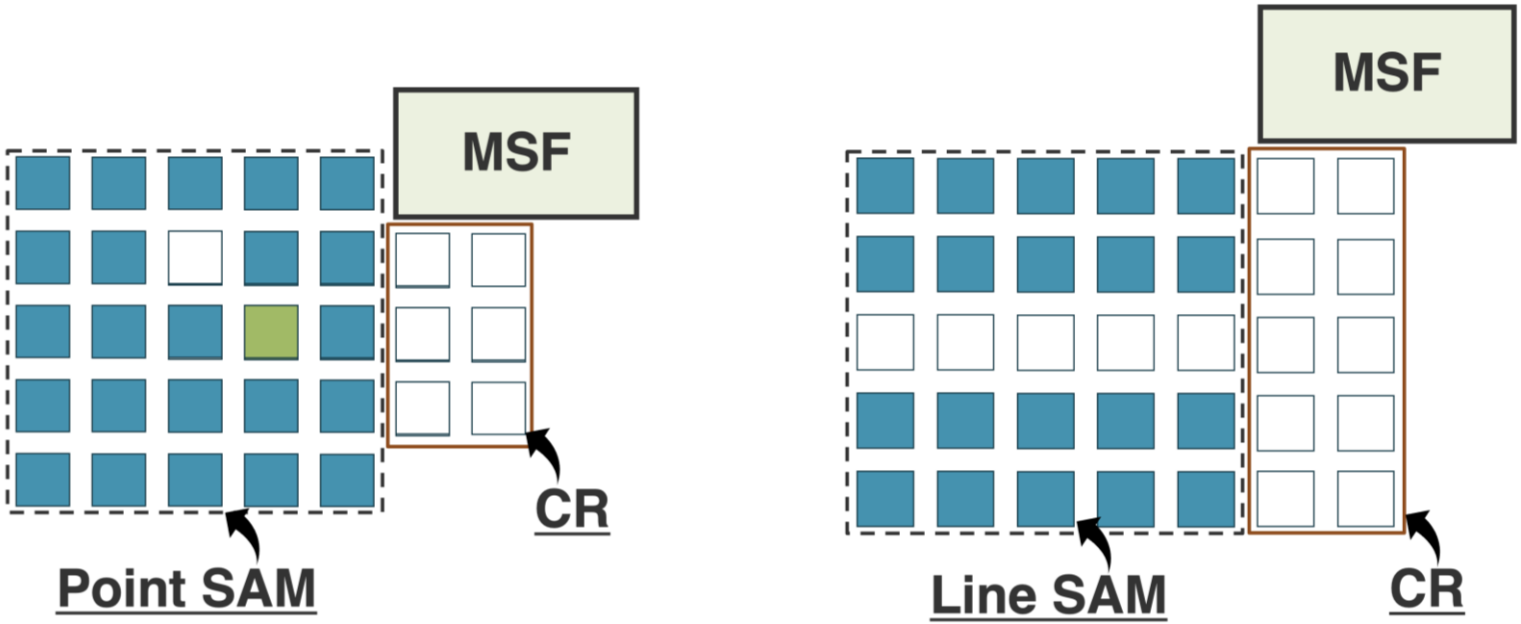
- Scan-Access Memory





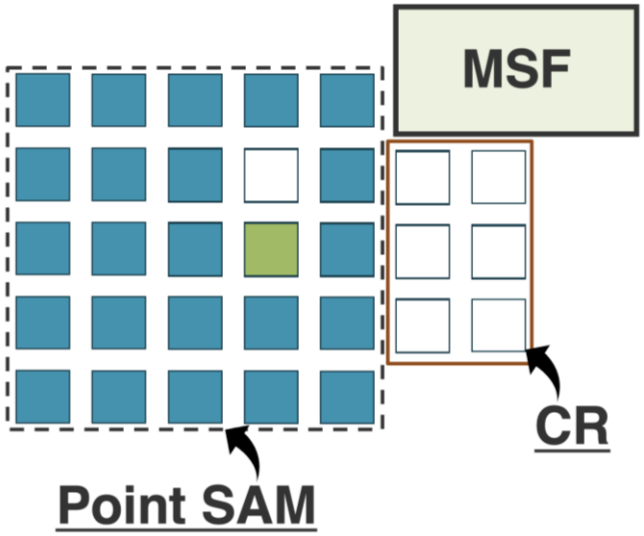
ロードストア型誤り耐性量子計算機アーキテクチャ

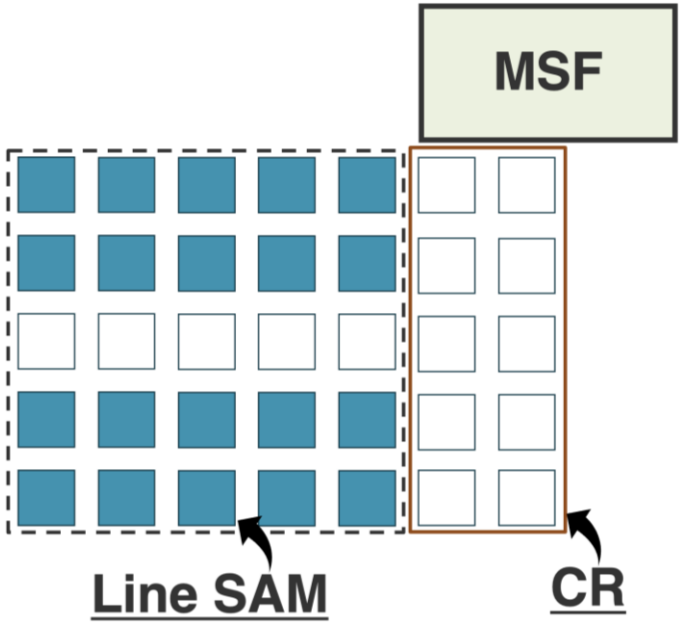
- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

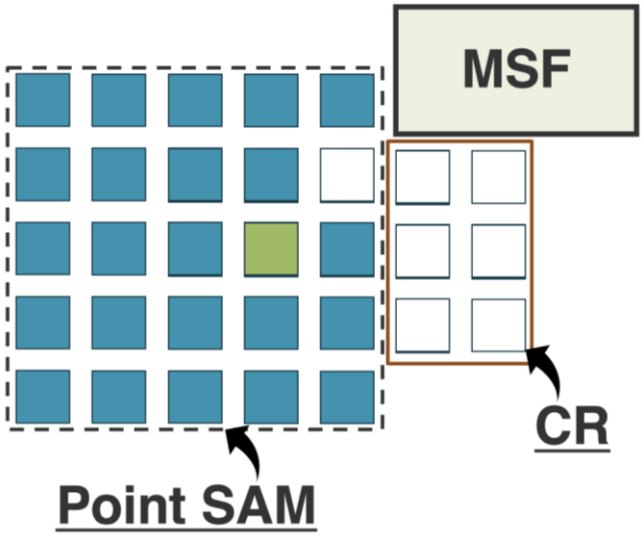
- Scan-Access Memory

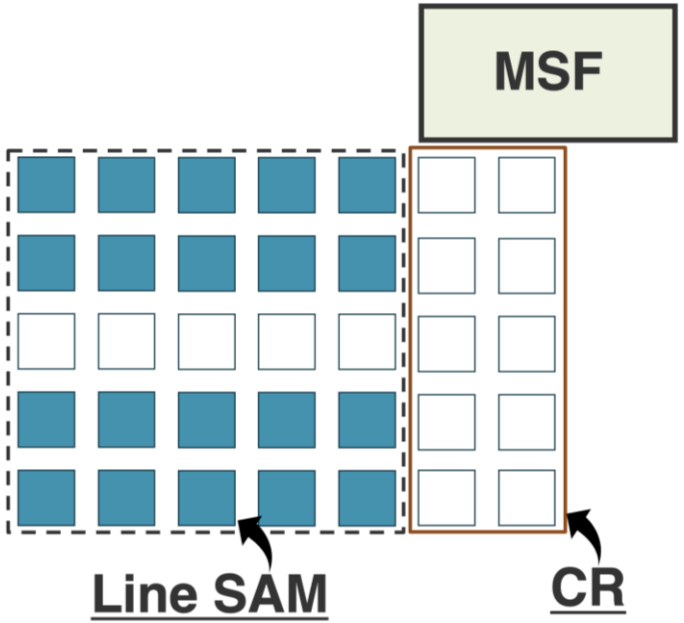




ロードストア型誤り耐性量子計算機アーキテクチャ

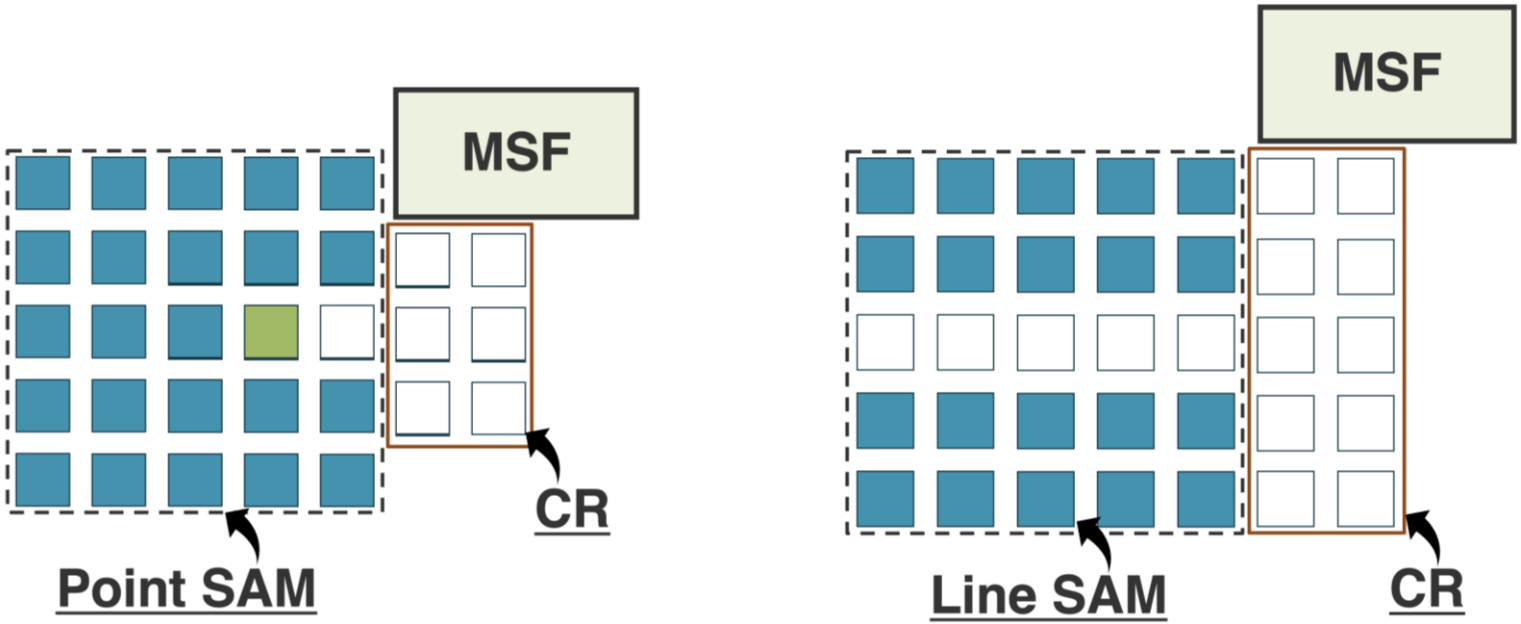
- Scan-Access Memory





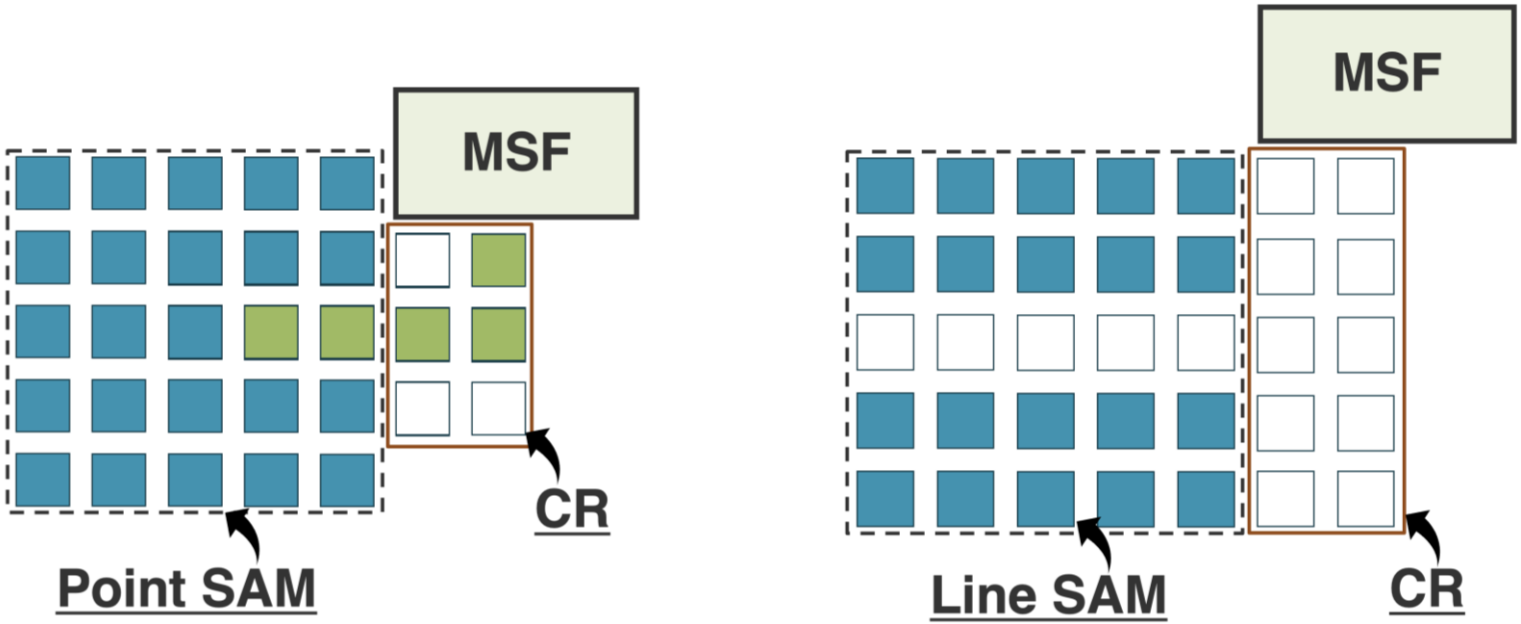
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



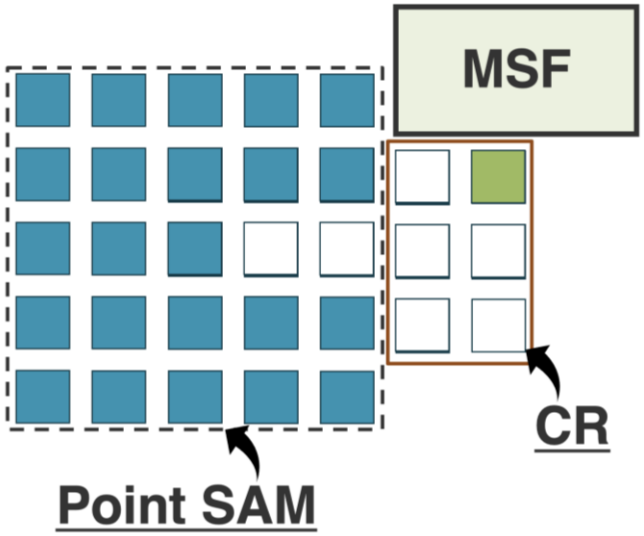
ロードストア型誤り耐性量子計算機アーキテクチャ

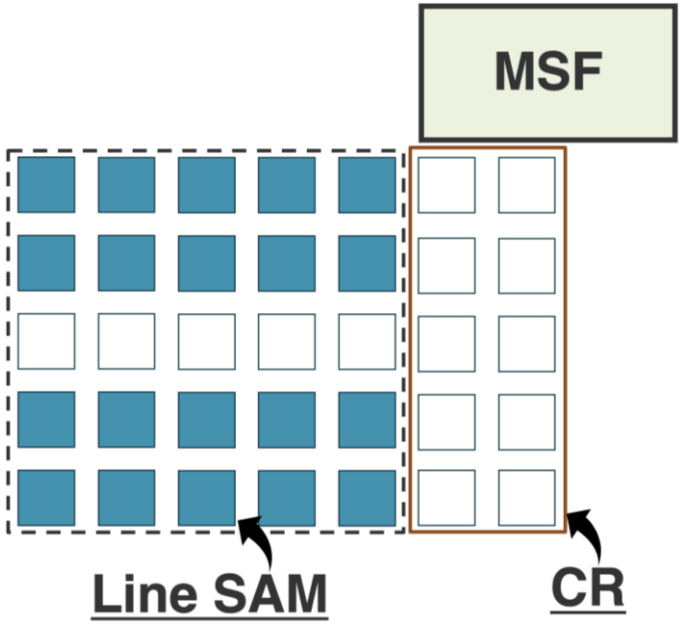
- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

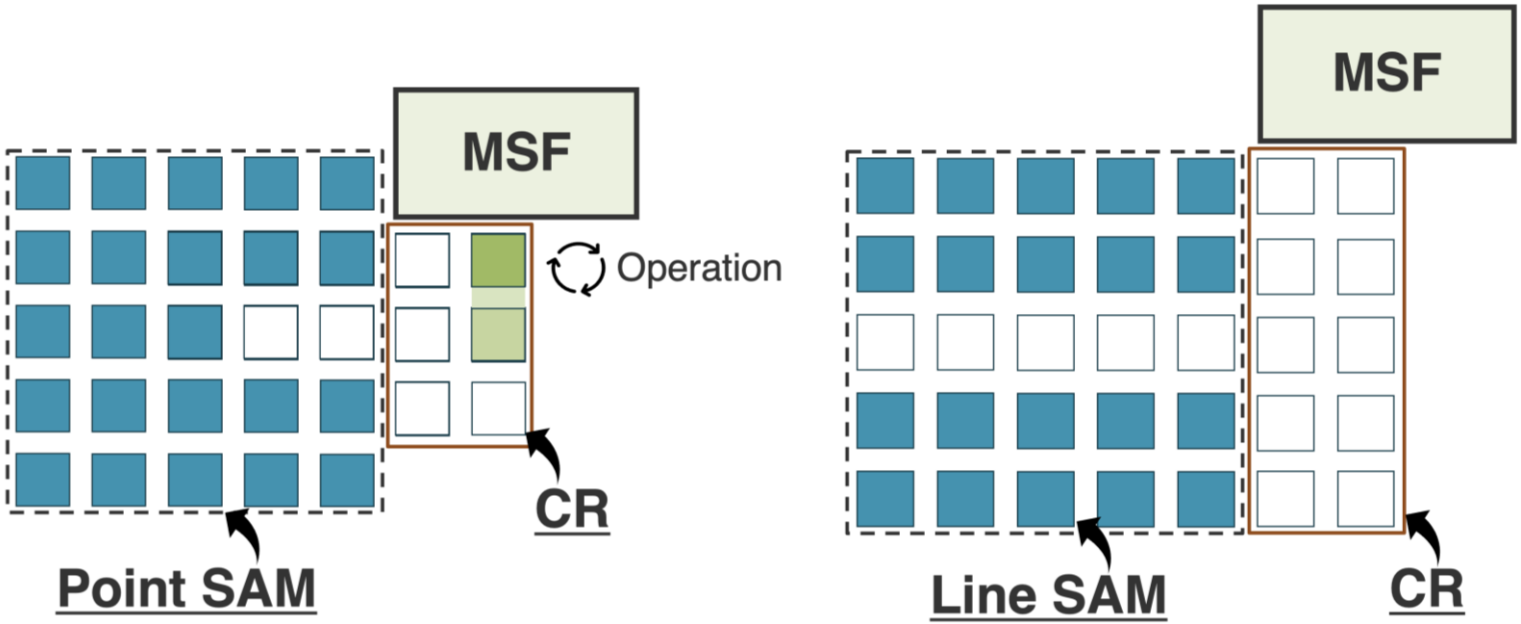
- Scan-Access Memory





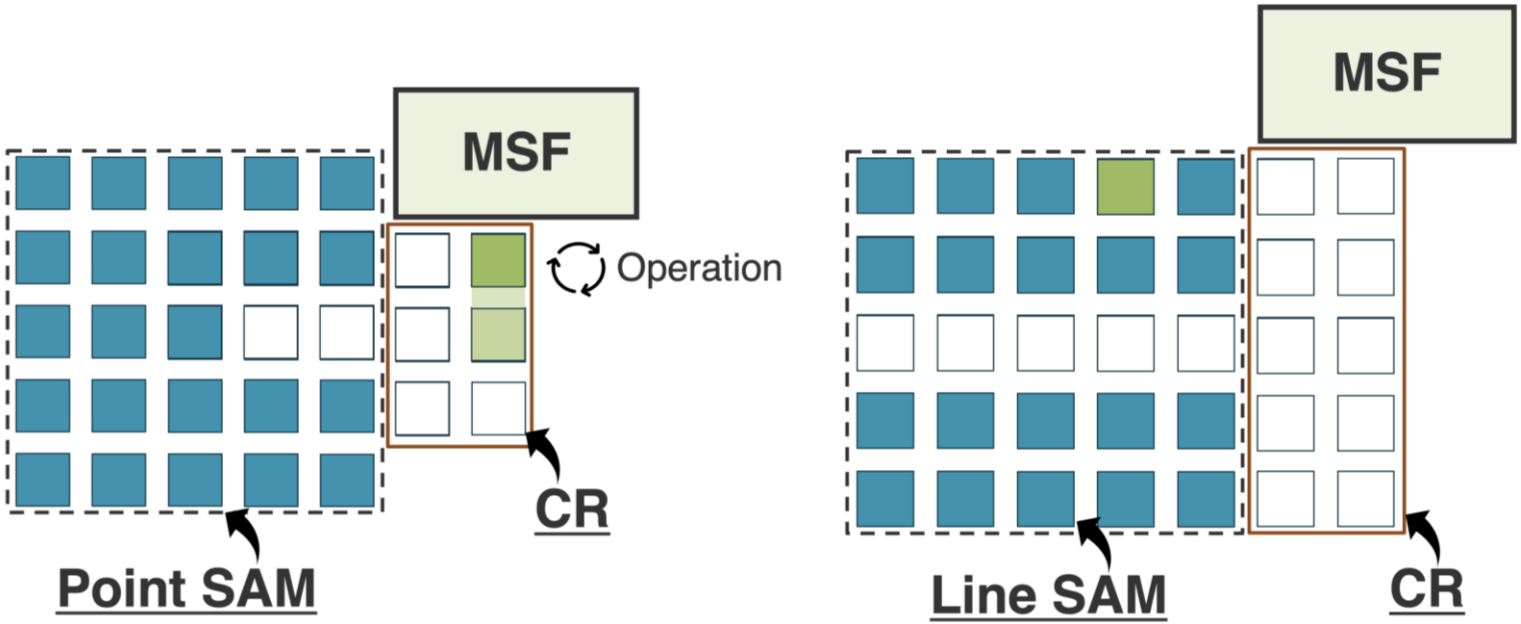
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



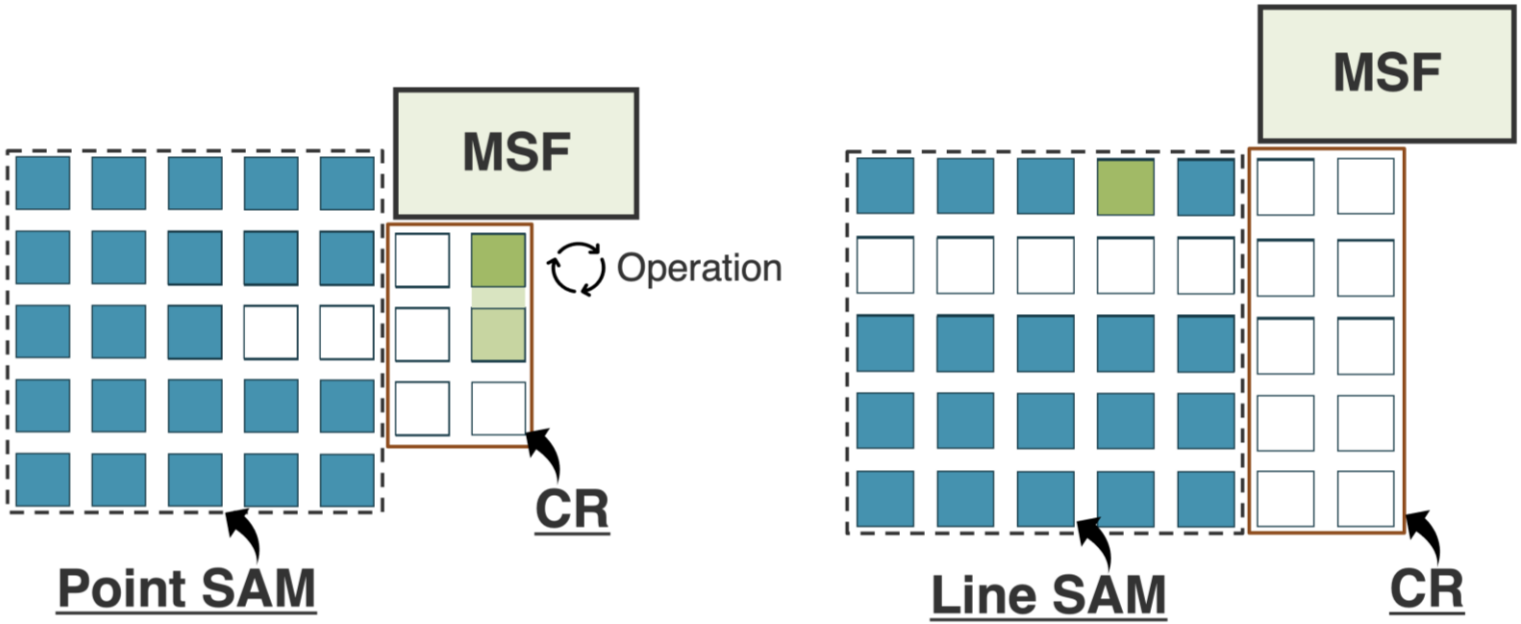
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



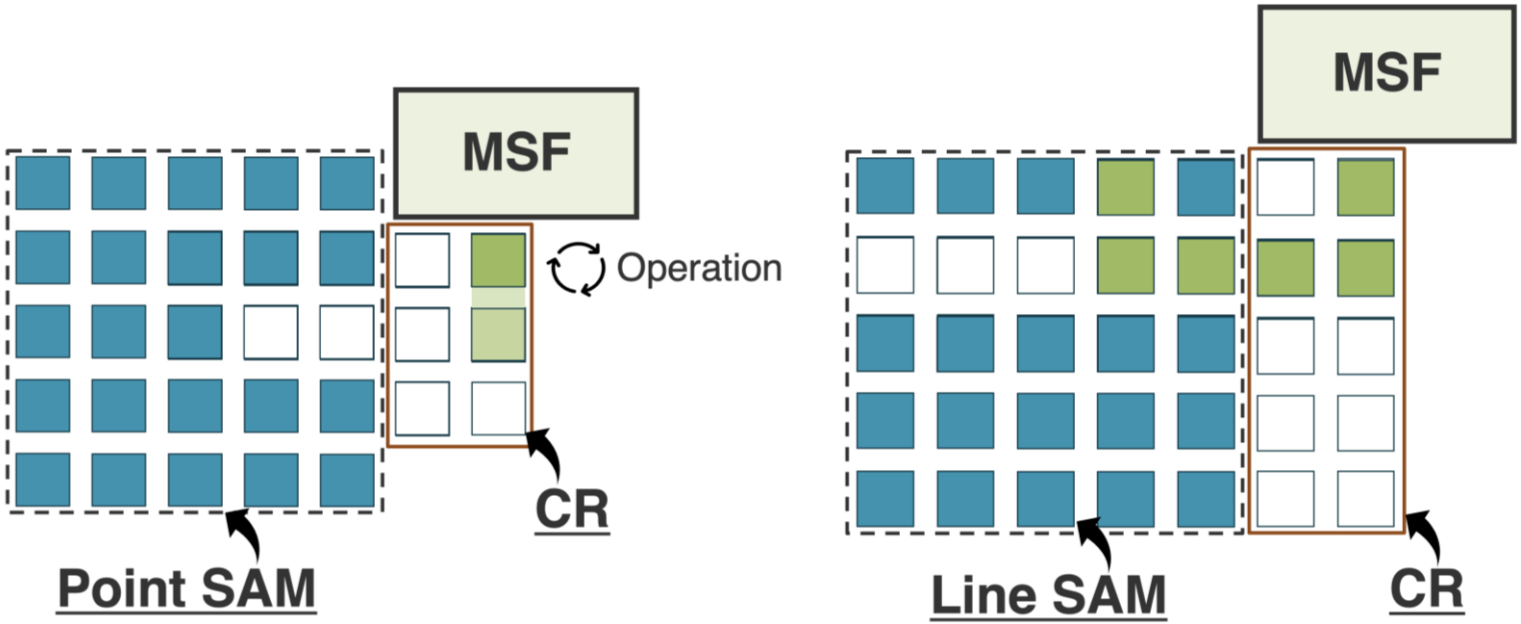
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



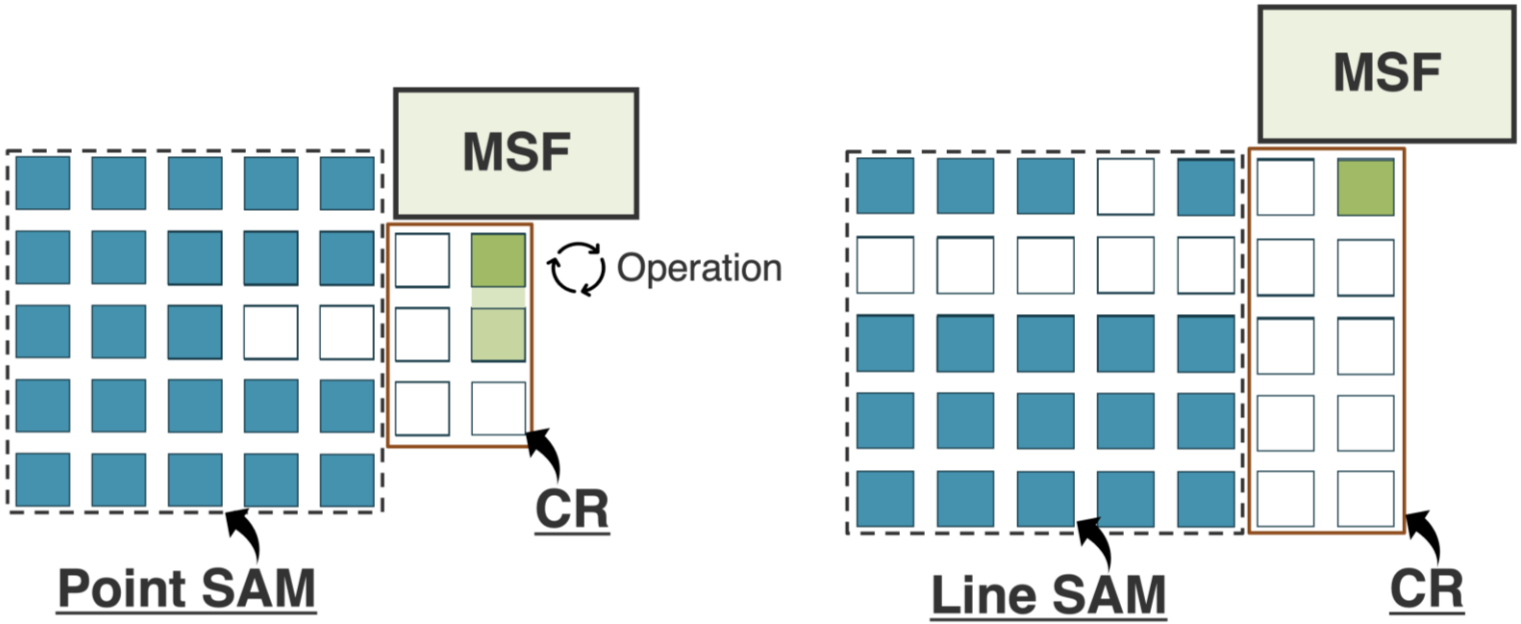
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



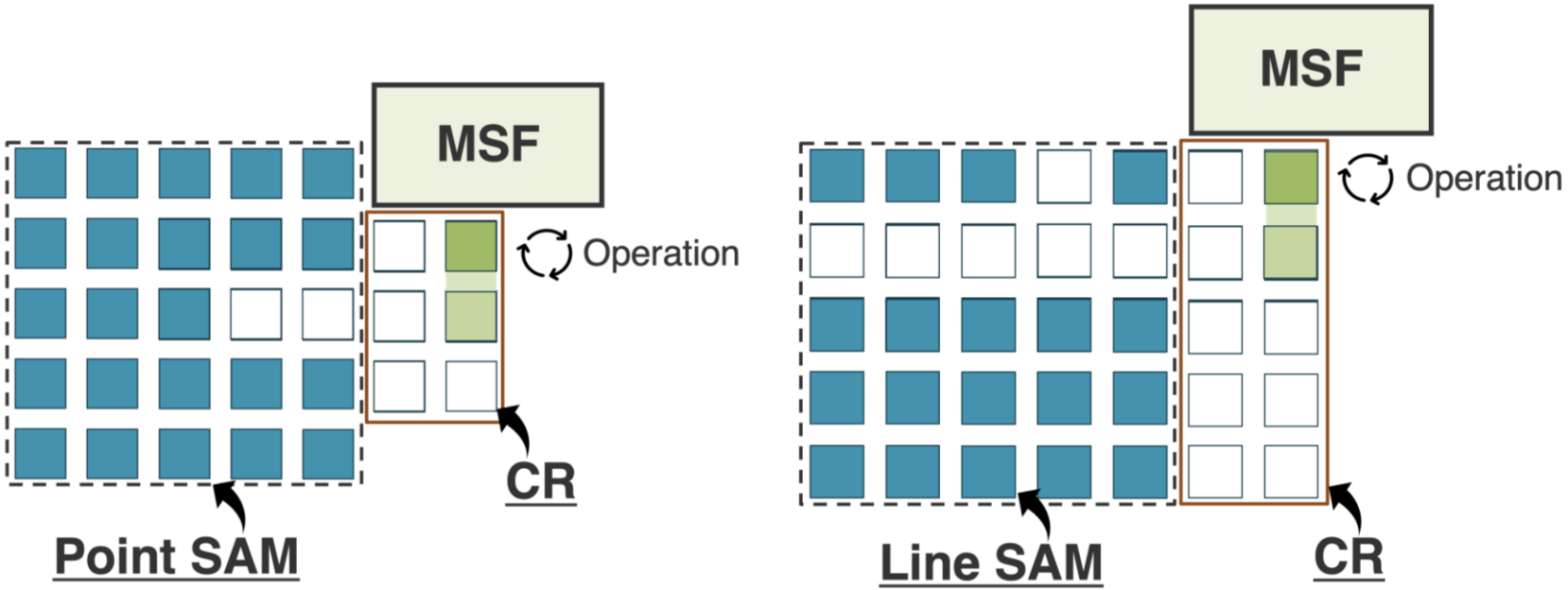
ロードストア型誤り耐性量子計算機アーキテクチャ

- Scan-Access Memory



ロードストア型誤り耐性量子計算機アーキテクチャ

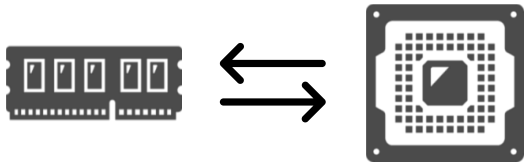
- Scan-Access Memory



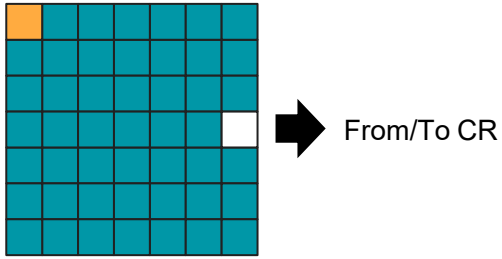
ロードストア型を採用するコスト

ロードストア型の欠点はメモリ-レジスタ間通信による計算の低速化

プログラムによっては通信の遅さがボトルネックに



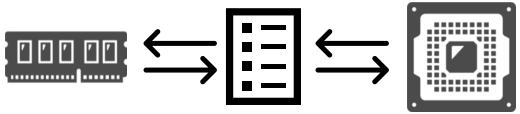
最悪ケースでは最もアクセスの遅いセルが
繰り返し呼び出される形になる



通常の計算機では典型的プログラムが持つ偏りを活用し様々なテクニックでこれらの問題を解決

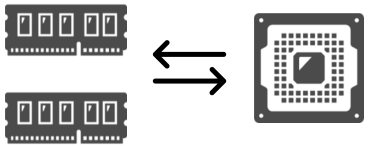
キャッシュ / 専用メモリ

頻繁にアクセスするものや
アクセスが予測されるものを
中規模で高速なメモリに保存する



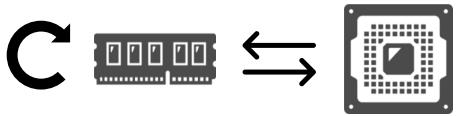
マルチチャンネルメモリ

複数のメモリバンクに分割し、
並列して読み出しを行うことで
実効的な読み出し速度を上げる



インメモリ計算

メモリ内部に軽量な計算機構を入れ
一部の計算をメモリ内部で行う



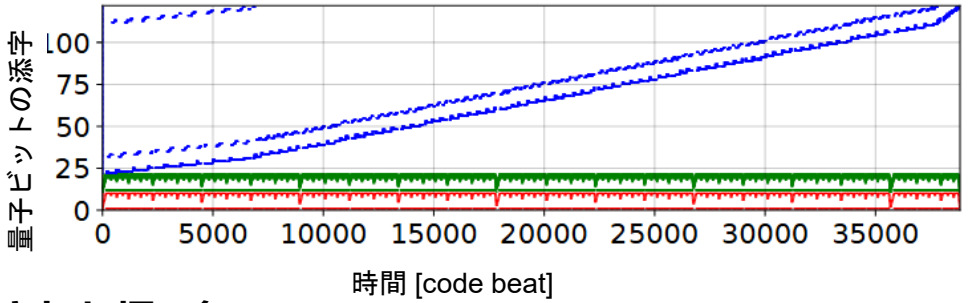
キャッシュなどの技術はメモリアクセスが典型的な特性を持っていることを仮定している　量子も同様の性質を持つか？

典型的なプログラムが持つ構造の解析

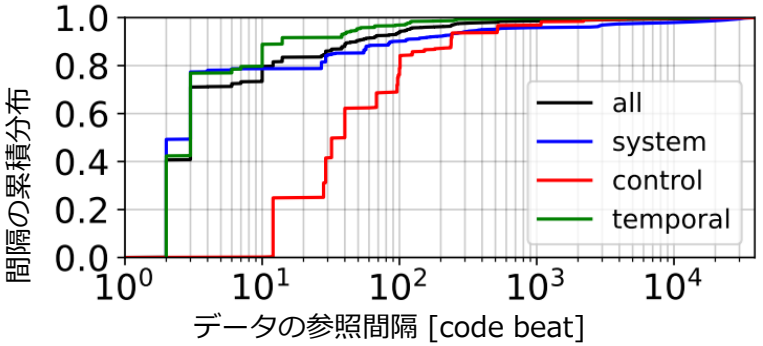
主要な量子アルゴリズムにおいてメモリのアクセスは最適化可能か？

Qubitization-basedな位相推定のボトルネックとなる操作で評価 (論文では他のものも評価)

メモリアクセスのタイムトレース



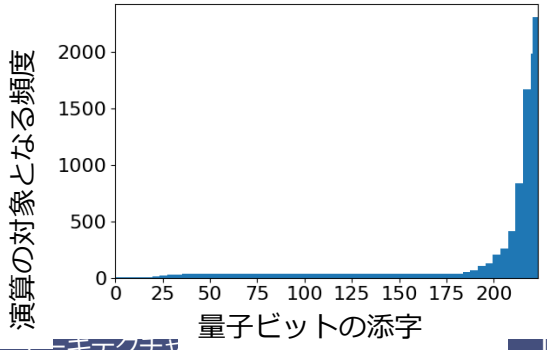
レジスタの再参照間隔



観察された振る舞い

時間的局所性	一度参照されたデータは、 高い確率で数命令以内に再参照される
空間的局所性	あるデータが参照された後に、 隣接するデータは参照されやすい
アクセスの偏り	頻繁に演算対象となる少数の量子ビットと 殆ど記憶されている多数の量子ビットに二極化する
要求並列度	典型的なプログラムでは処理並列度は低い 一斉に多くのデータが必要になるケースは稀

データの参照回数のバイアス



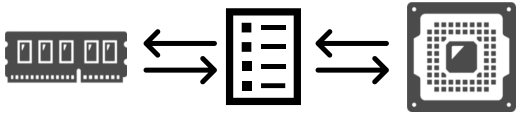
→ **メモリ最適化に有利な特性　これを利用して最適化が可能**

ロードストアの影響を軽減する最適化

最適化機構の提案

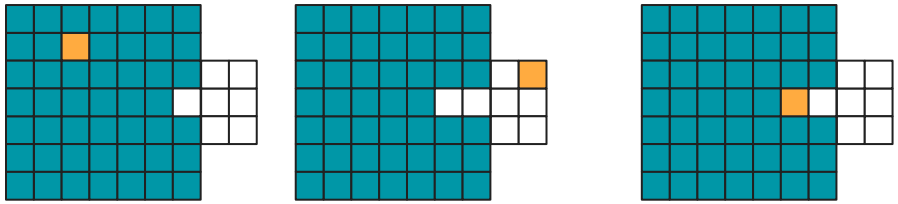
指針：キャッシュ/専用メモリ

頻繁にアクセスするものや
アクセスが予測されるものを
中規模で高速なメモリに保存する

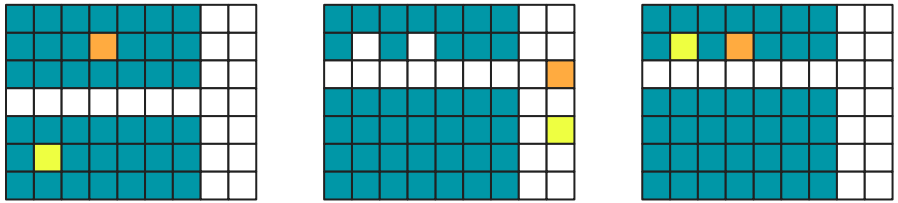


最適化１：Locality-aware store

利用したデータは元の場所ではなくアクセスの早い場所に戻す
これにより、プログラムの知識を用いずに再参照を早くする

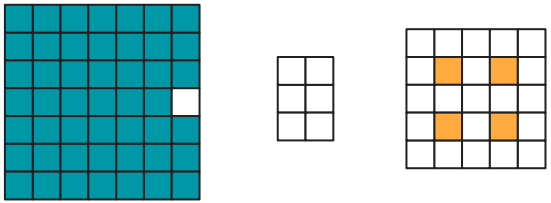


同時に利用したデータは同時にアクセスしやすい場所に戻す
これにより、空間的な局所性を活用し典型的ロードを早くする



最適化２：Hybrid Floorplan

レジスタのサイズを拡張し、静的解析に基づいて
頻繁にアクセスされる少数のデータを専用領域に配置する

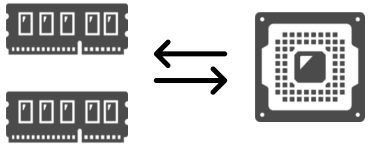


ロードストアの影響を軽減する最適化

最適化機構の提案

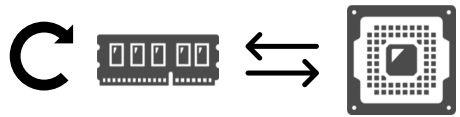
指針２：マルチチャンネルメモリ

複数のメモリバンクに分割し、
並列して読み出しを行うことで
実効的な読み出し速度を上げる



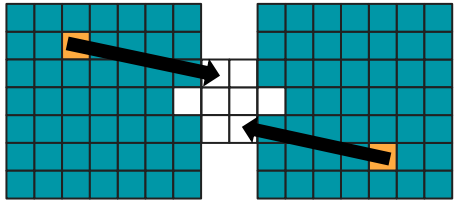
指針３：インメモリ計算

メモリ内部に軽量な計算機構を入れ
一部の計算をメモリ内部で行う



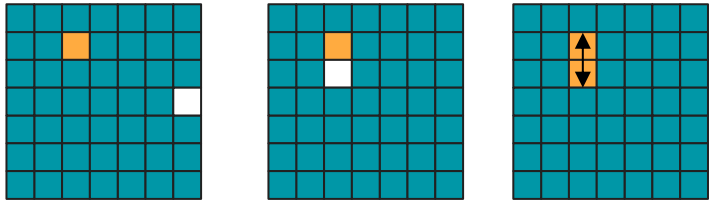
最適化３：Multi-bank SAM

データを複数のSAMに分割して配置し、
2並列でのLOAD/STOREを可能にする



最適化４：In-memory operations

移動用の追加セルで完結できる論理操作は、
CRに読みださずにメモリ内部で完結させる

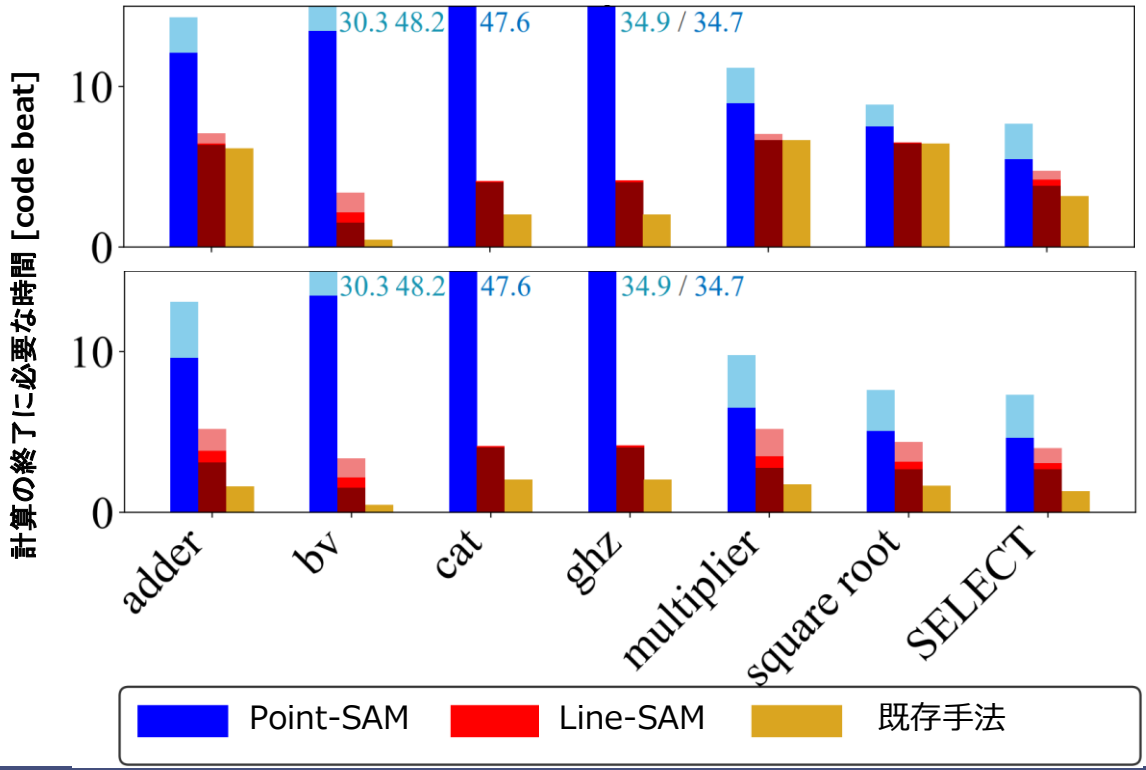


論理HゲートやSゲートをその場で実行

ベンチマーク：プログラム間比較

より詳細なアプリケーションごとの実行時間のオーバーヘッド

レジスタのサイズを2に固定しても、主要な操作ではオーバーヘッドは2倍～4倍に収まる



魔法状態の供給量

1 state per 15 lattice surgery

（魔法状態蒸留1基に相当）

魔法状態の供給量

4 state per 15 lattice surgery

（魔法状態蒸留4基に相当）

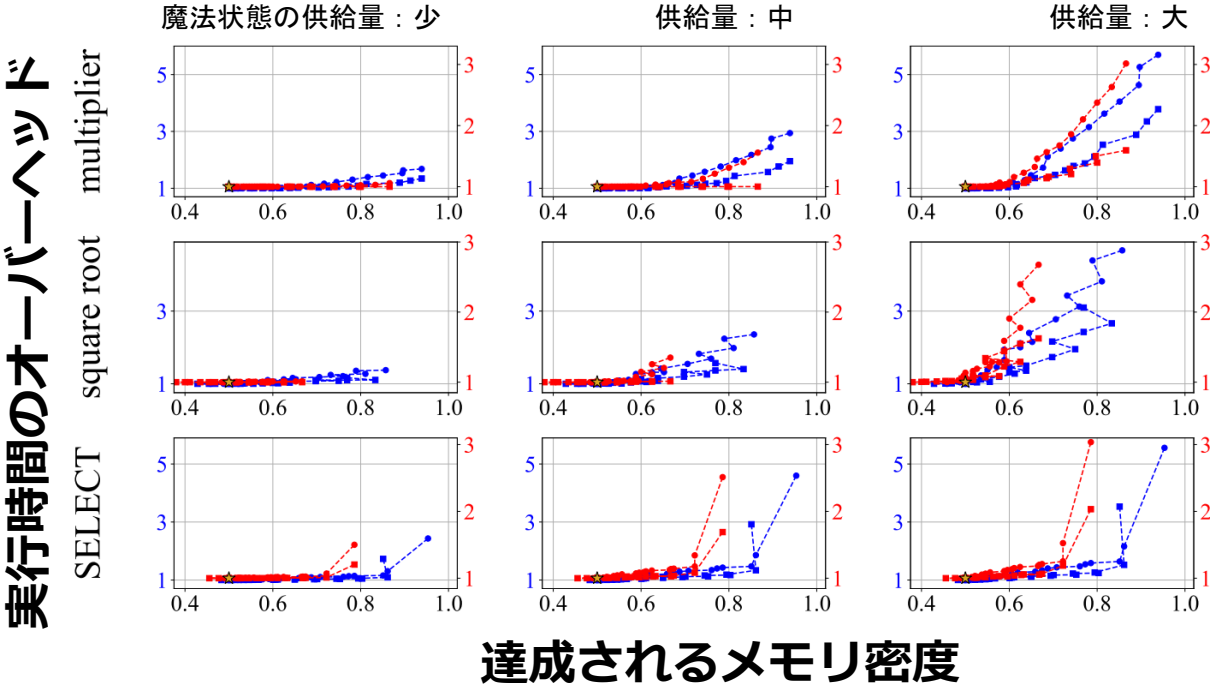
※ベースラインを保守的にするため
既存手法は論理操作のための
追加リソースは不要と仮定している
これは本提案に不利な過程
以降の数値計算でも同様

ベンチマーク：レジスタサイズの掃引

レジスタのサイズを掃引した際のパフォーマンス

レジスタのサイズを掃引することで、時間オーバーヘッドとメモリ密度を交換可能

FTQCで主要な操作はアクセスが局在しており、レジスタが少数でも性能はあまり劣化しない



青：Point-SAM

赤：Line-SAM

●：単一チャンネルメモリ

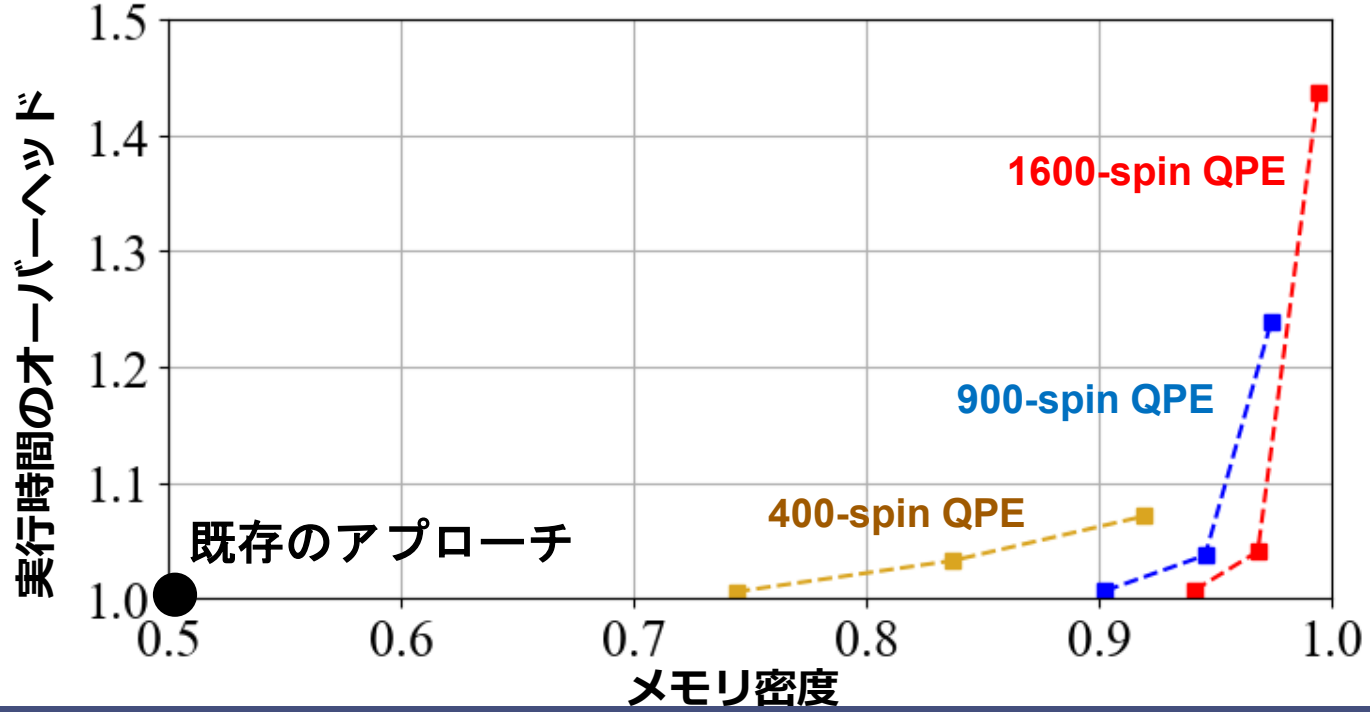
■：多チャンネルメモリ

ベンチマーク：問題サイズの掃引

SELECT回路(位相推定の一部)を様々なインスタンスサイズで評価

問題サイズに合わせてレジスタのサイズを最適化

問題サイズが大きいと実行時間オーバーヘッドも大きいがより高いメモリ効率が達成可能



まとめと展望：今回実施したこと

1．ロードストア命令セットの定義

様々なデバイスや符号と整合性のある命令セットアーキテクチャを定義

上記を処理するためのコンパイル手法を提案しシミュレータを構築

実機側の動作を実行時に整合させることで、共通の実行ファイルで複数の実機を低オーバーヘッドに駆動可能

2．表面符号を用いた高メモリ密度計算モデル（スキャン式メモリ）を定義

LDPC符号を用いる場合は格子手術があるため自明に定義可能

表面符号を用いる場合でも最大で倍程度にメモリ密度が改善可能なことを提案

3．実効的なメモリ帯域を改善し遅延を隠ぺいするデザインの提案

位相推定や素因数分解のプログラムが持つアクセス局所性や演算/メモリ命令比を活用し設計を最適化

1．マルチチャンネルメモリ	メモリを多重化し一度にいっぱい転送する
2．キャッシュ	また使いそうなデータはすぐ読める場所にしまう
3．インメモリ計算	メモリ内でも出来る演算はメモリ内で行う
4．追加の高速記憶領域	頻繁に参照するデータは早いメモリに入れる

ベンチマークを通し、実用ケースでx1.8の量子ビット削減を+5%の計算時間で実現

160ページ以降（3コマ目）は査読中の内容が含まれるため
公開版では削除、後日更新予定