Slides available in https://yuteno.github.io/

Load/Store Architecture for Fault-Tolerant Quantum Computing

Yosuke Ueno[1, 2]

In collaboration with

Takumi Kobori[2], Yasunari Suzuki[1], Teruo Tanimoto[3],

Synge Todo[1], Yuuki Tokunaga[4]

Physicists and Computer architects

[1]RIKEN, [2]The University of Tokyo, [3]Kyushu University, [4]NTT

arXiv 2412.20486

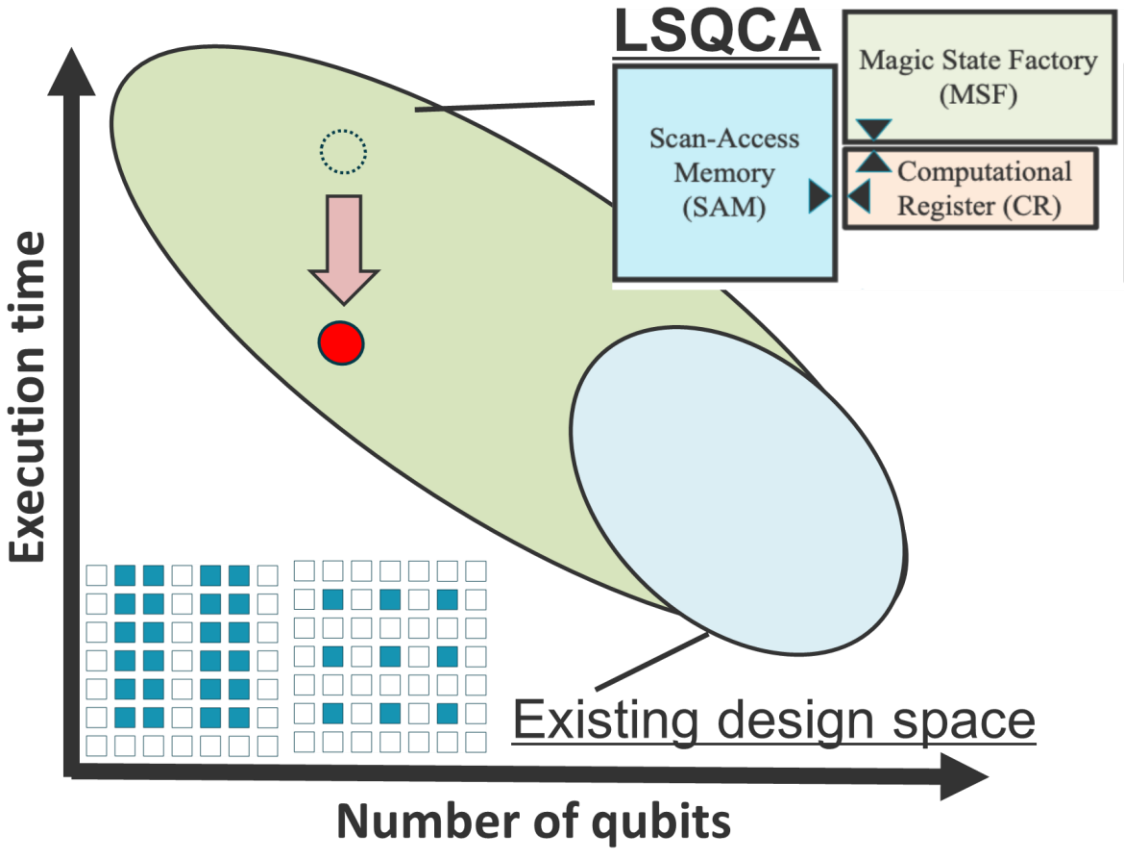
Takumi Kobori, Yasunari Suzuki, **Yosuke Ueno**, Teruo Tanimoto, Synge Todo, Yuuki Tokunaga, LSQCA: Resource-Efficient Load/Store Architecture for Limited-Scale Fault-Tolerant Quantum Computing, IEEE International Symposium on High-Performance Computer Architecture (HPCA) 2025, pp. 304-320.

Yosuke Ueno

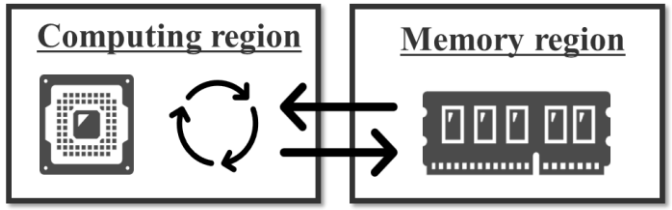
- Career
  - 2022.3: Ph.D. in Information Science and Technology @ The University of Tokyo
    - Supervisors: Prof. Masaaki kondo and Prof. Hiroshi Nakamura
    - Thesis: Online Quantum Error Correction Using a Superconducting Circuit
  - 2022.5 to 2023.2: **Guest researcher @ Technical University of Munich**
  - 2023.4 to Present: Postdoc @ RIKEN Center for Quantum Computing
  - 2025.5 to Present: Project Research associate @ The University of Tokyo
- Research Subjects and Interests
  - Computer architecture
  - Fault-tolerant quantum computing
  - Cryogenic computing

My first and last beer in Germany, 2022-2023.

Overview of our work and achievement



- ✓ Load/Store architecture on FTQC

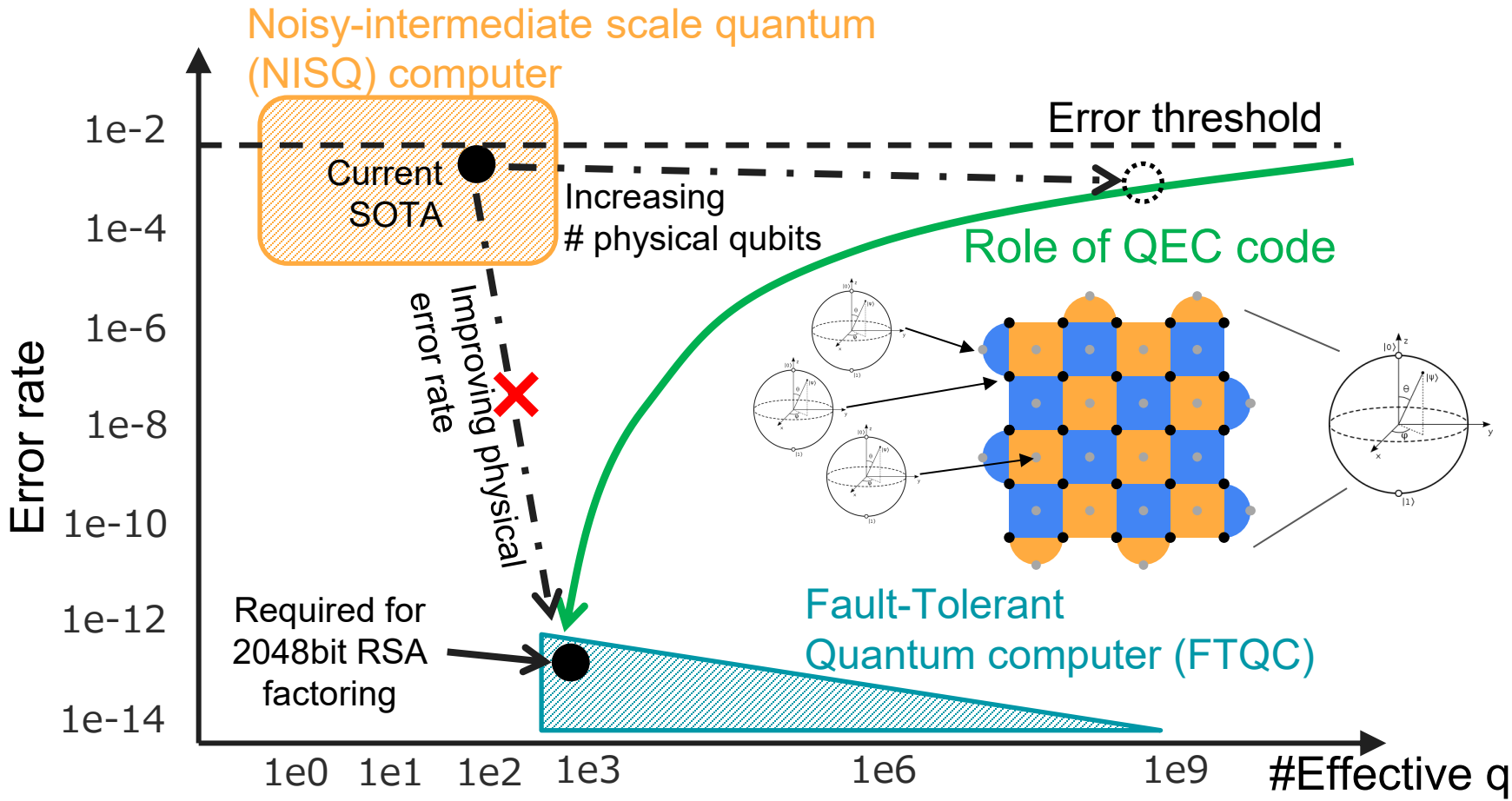


- ✓ Higher memory efficiency than existing design
- ✓ Utilize access localities in quantum programs
  - ✓ access localities in quantum programs

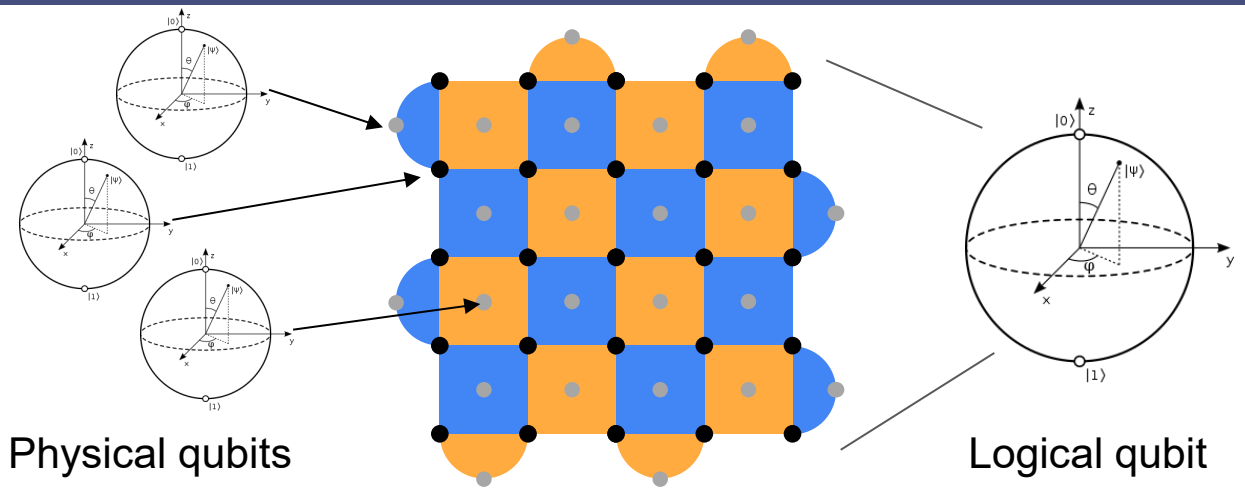
→ Expand design space and Offer novel and usable options

Contents

Role of Quantum Error Correction (QEC)



Surface code



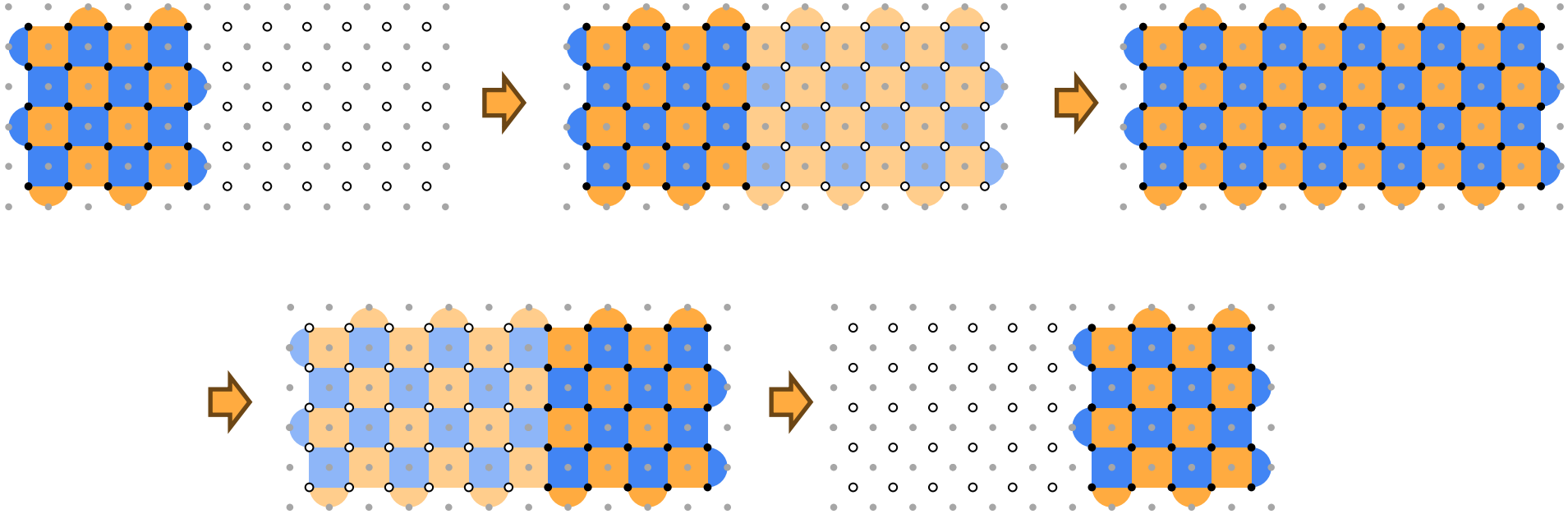
Data qubit

Ancillary qubit
(X error parity check)

Ancillary qubit
(Z error parity check)

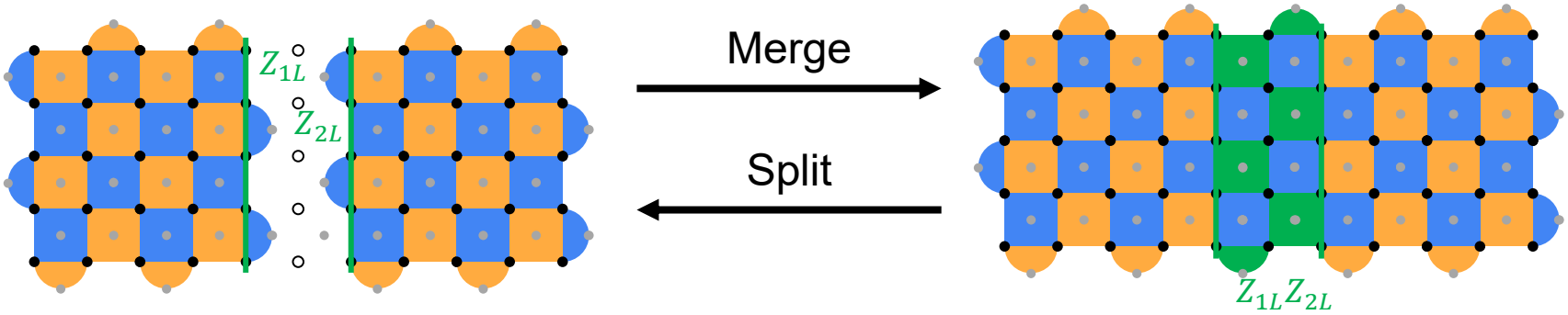
- Represent one logical qubit using $O(d^2)$ physical qubits
  - **Data qubits** for logical state + **Ancillary qubits** for error parity check
  - **Code distance d**: A parameter that quantifies. Logical error rate decreases exponentially to d for physical error rate p below threshold.
- Error decoding process can be reduced to graph matching problem

Code deformation



- Changing the set of parity checks deforms the code
  - Deformation while preserving logical state -> Expand, shrink, **move**, rotate
  - Deformation while changing logical state -> Logical S, CNOT gates

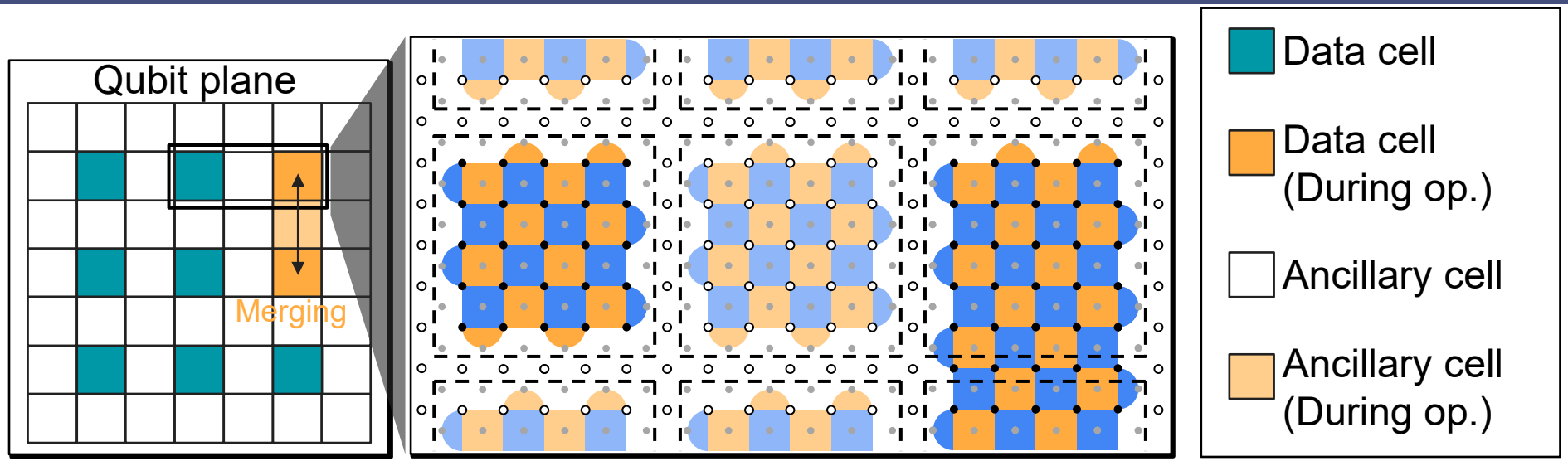
Lattice surgery (LS)



ZZ measurement via lattice surgery

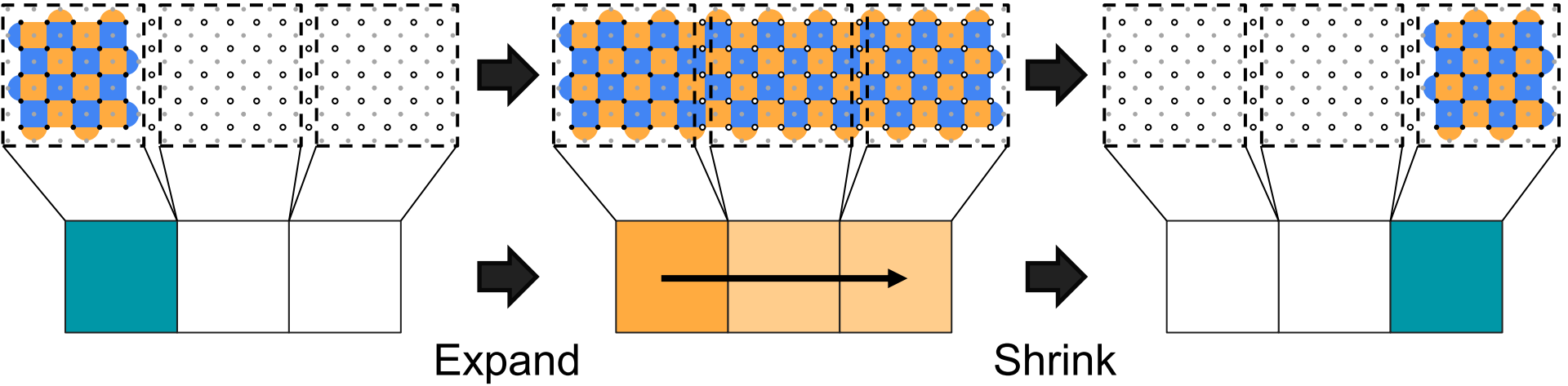
- Using LS technique, we can perform logical operation on SC-based logical qubits with nearest-neighbor qubit interactions.
- Universal gate sate $\{H, S, CNOT, T\}$
  - H, S: Performed via code deformation of single logical qubit
  - $CNOT$: Performed with multi-body Pauli measurements via lattice surgery
  - T: Realized gate teleportation technique and magic states

Qubit plane



- Each cell has sufficient physical qubits to construct distance-d SC
- Data cells store logical states for computation
- Ancillary cells serve as functional spaces for logical operations on data cells

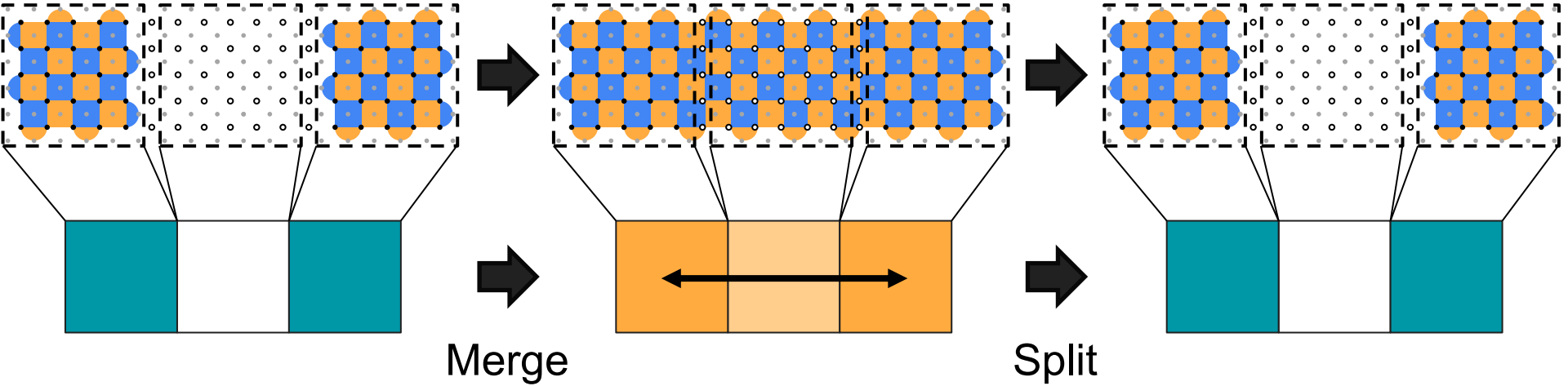
Example: logical qubit movement



Data cell storing logical state

Ancillary cell serving functional space for logical operation

Example: lattice surgery of two logical qubits

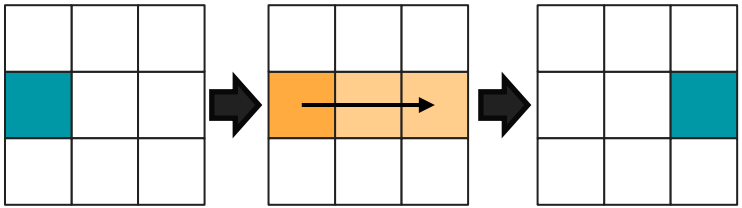


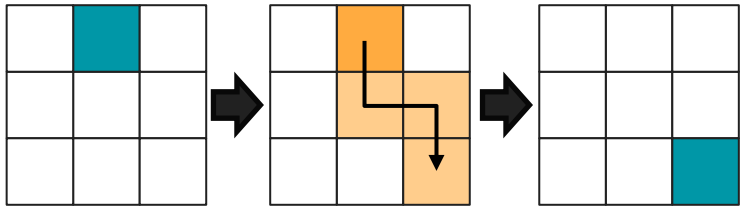
Data cell storing logical state

Ancillary cell serving functional space for logical operation

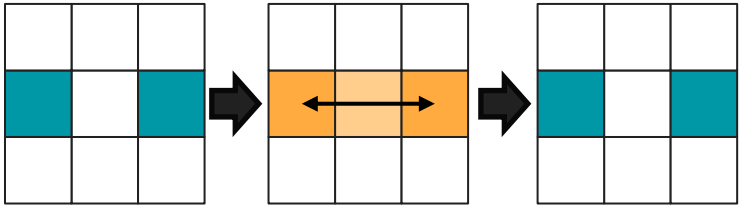
Path for move and lattice surgery operations

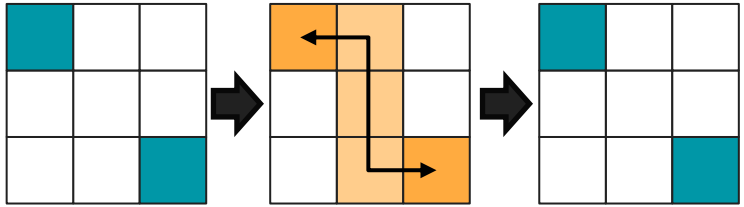
Move operations with various paths





Lattice surgery operations with various paths

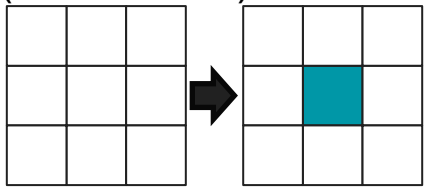




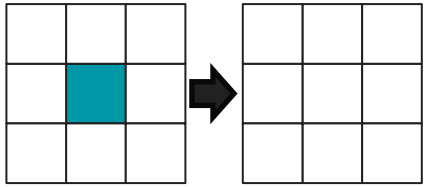
- Any path for move and lattice surgery operations is allowed.
- Execution time is 1 unit time (**code beat**) and independent of the path selection.

Lattice surgery instruction set

`INIT_Z,INIT_X`
(0 code beat)

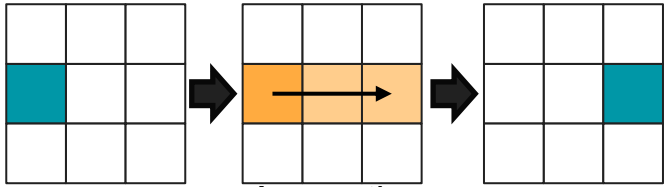


`MEAS_X,MEAS_Z`
(0 code beat)



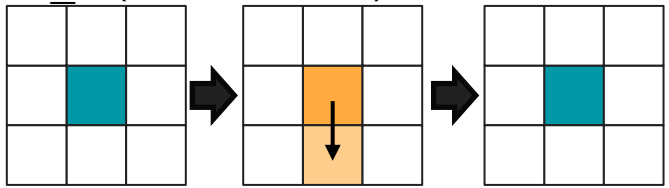
+Classical output

`MOVE`
(1 code beat)



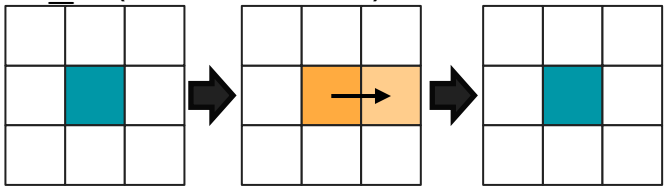
Any path

`OP_H` (3 code beats)



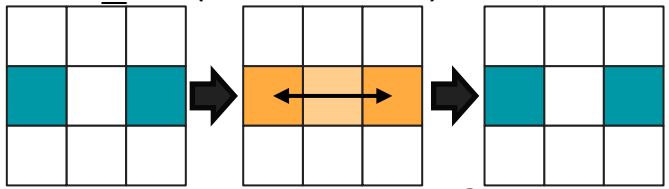
Any direction

`OP_S` (2 code beats)



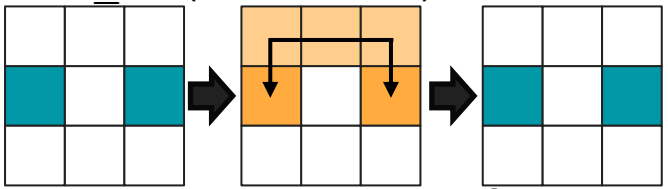
Horizontal direction

`MEAS_ZZ` (1 code beat)



Any path +Classical output

`MEAS_XX` (1 code beat)



Any path +Classical output

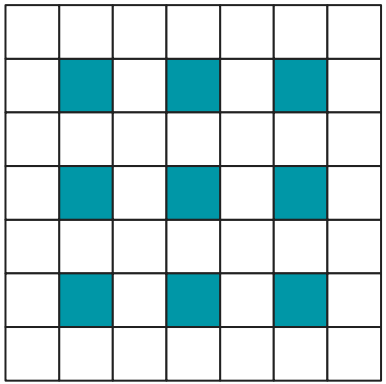
Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

Floorplans affect performance of FTQC

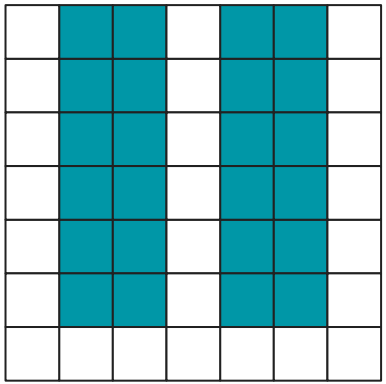
- Floorplan: how to place data and ancillary cells
- Tradeoff between **memory density** and execution time

The ratio of logical qubit cells (data cells) to total cells



Low: 1/4

Memory density



High: 2/3

(LEFT) can execute many operations in parallel

➔ Short execution time

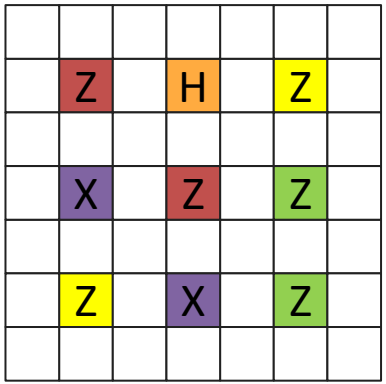
(RIGHT) some operations remain unoperated

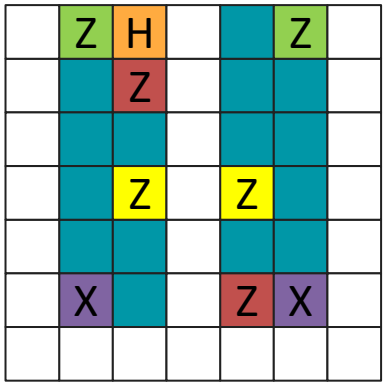
➔ Long execution time

Floorplans affect performance of FTQC

- Floorplan: how to place data and ancillary cells
- Tradeoff between **memory density** and execution time

The ratio of logical qubit cells (data cells) to total cells





Low: 1/4 — Memory density — High: 2/3

(LEFT) can execute many operations in parallel

➔ Short execution time

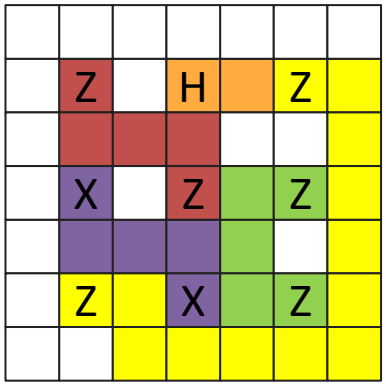
(RIGHT) some operations remain unoperated

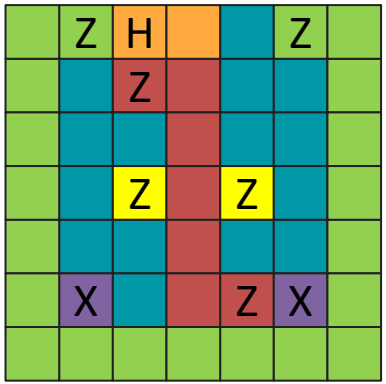
➔ Long execution time

Floorplans affect performance of FTQC

- Floorplan: how to place data and ancillary cells
- Tradeoff between **memory density** and execution time

The ratio of logical qubit cells (data cells) to total cells





Low: 1/4 — Memory density — High: 2/3

Short — Execution time — Long

(LEFT) can execute many operations in parallel

➔ Short execution time

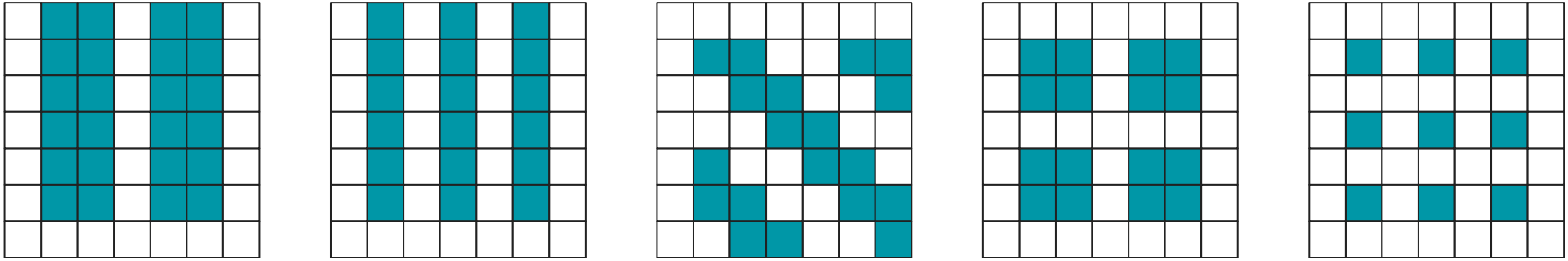
(RIGHT) some operations remain unoperated

➔ Long execution time

Previous works for floorplan

Existing floorplans

[1] J. Lee et al., PRX Quantum 2, 030305 (2021).
[2] M. Beverland et al., arXiv:2211.07629 (2022).
[3] Y. Ueno et al., arXiv:2411.17519 (2024).
[4] C. Chamberland and E. T. Campbell, PRX Quantum 3, 010331 (2022).
[5] M. Beverland et al., PRX Quantum 3, 020342 (2022).



2/3 [1] **1/2 [2]** **1/2 [3]** **4/9 [4]** **1/4 [5]**

Memory density

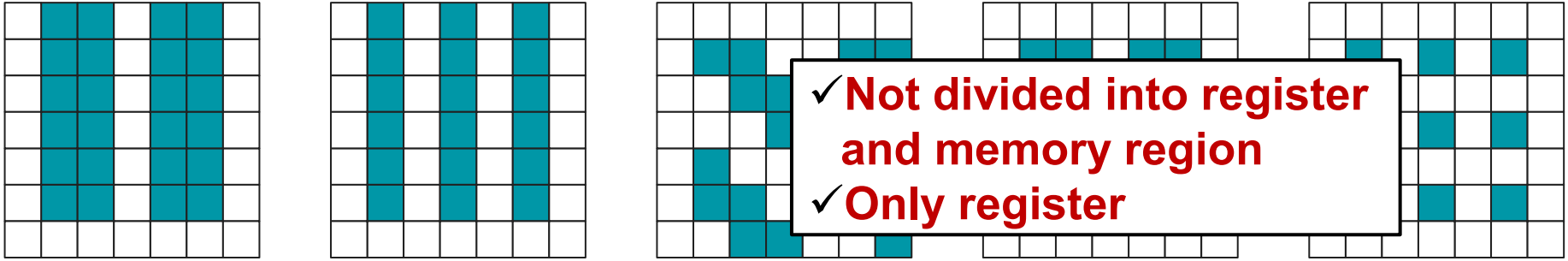
(−) < 1 memory density ➔ Requiring ancillary cells proportional to logical qubits

(+) Fast random access ➔ Any logical operations start in constant time

Previous works for floorplan

Existing floorplans

[1] J. Lee et al., PRX Quantum 2, 030305 (2021).
[2] M. Beverland et al., arXiv:2211.07629 (2022).
[3] Y. Ueno et al., arXiv:2411.17519 (2024).
[4] C. Chamberland and E. T. Campbell, PRX Quantum 3, 010331 (2022).
[5] M. Beverland et al., PRX Quantum 3, 020342 (2022).



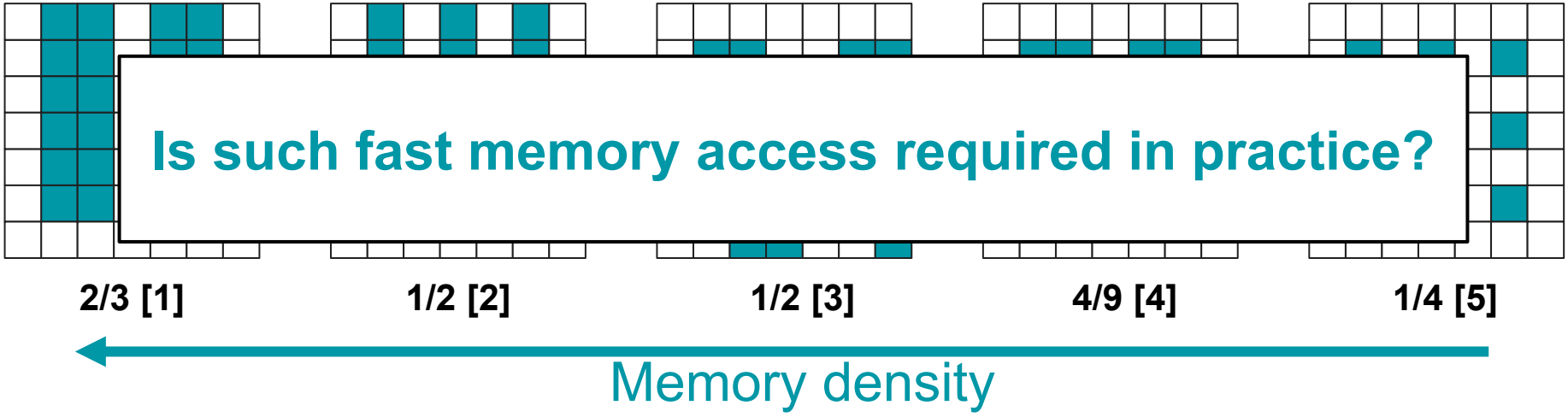
(−) < 1 memory density ➔ Requiring ancillary cells proportional to logical qubits

(+) Fast random access ➔ Any logical operations start in constant time

Previous works for floorplan

Existing floorplans

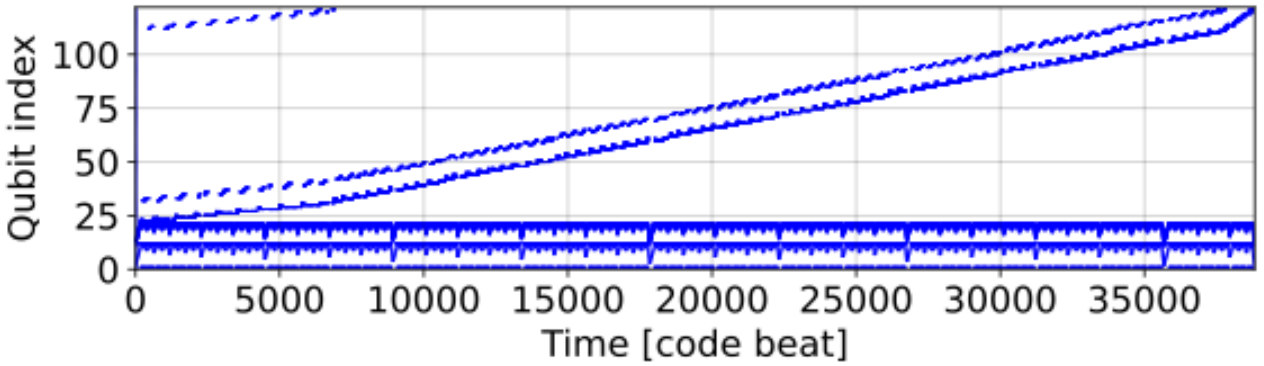
[1] J. Lee et al., PRX Quantum 2, 030305 (2021).
[2] M. Beverland et al., arXiv:2211.07629 (2022).
[3] Y. Ueno et al., arXiv:2411.17519 (2024).
[4] C. Chamberland and E. T. Campbell, PRX Quantum 3, 010331 (2022).
[5] M. Beverland et al., PRX Quantum 3, 020342 (2022).



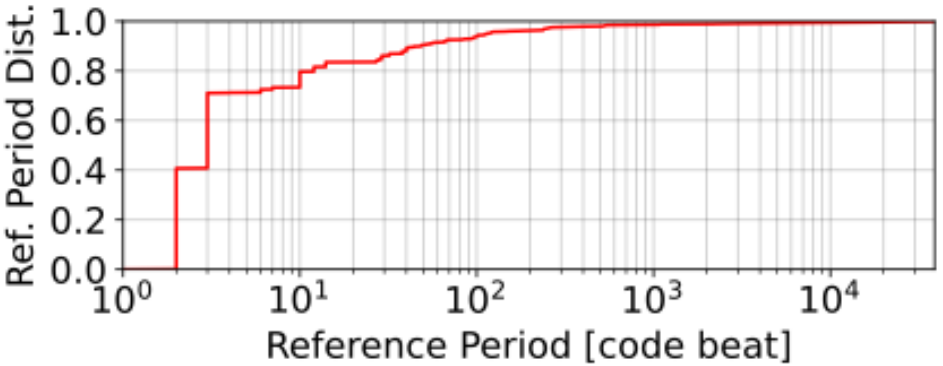
(−) < 1 memory density ➔ Requiring ancillary cells proportional to logical qubits
(+) Fast random access ➔ Any logical operations start in constant time

Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation

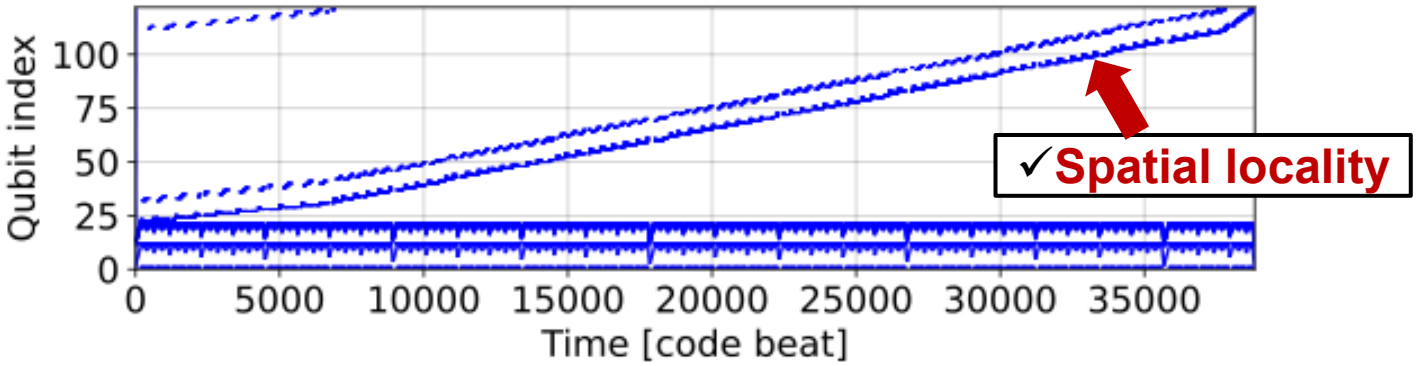


Cumulative distribution of average reference period for SELECT circuit

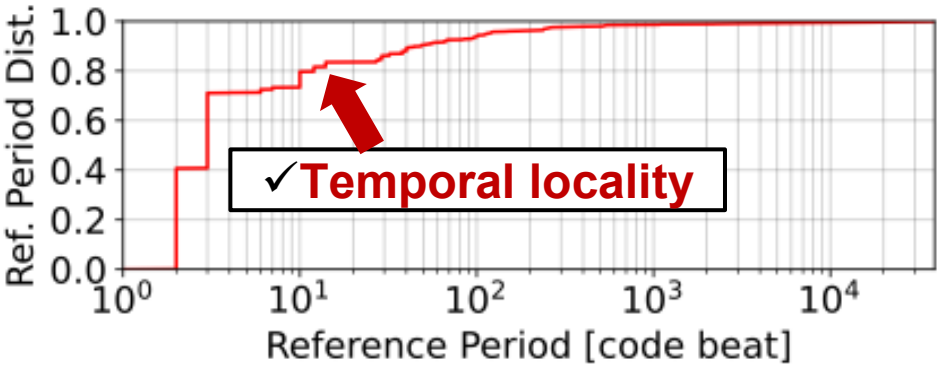


Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation

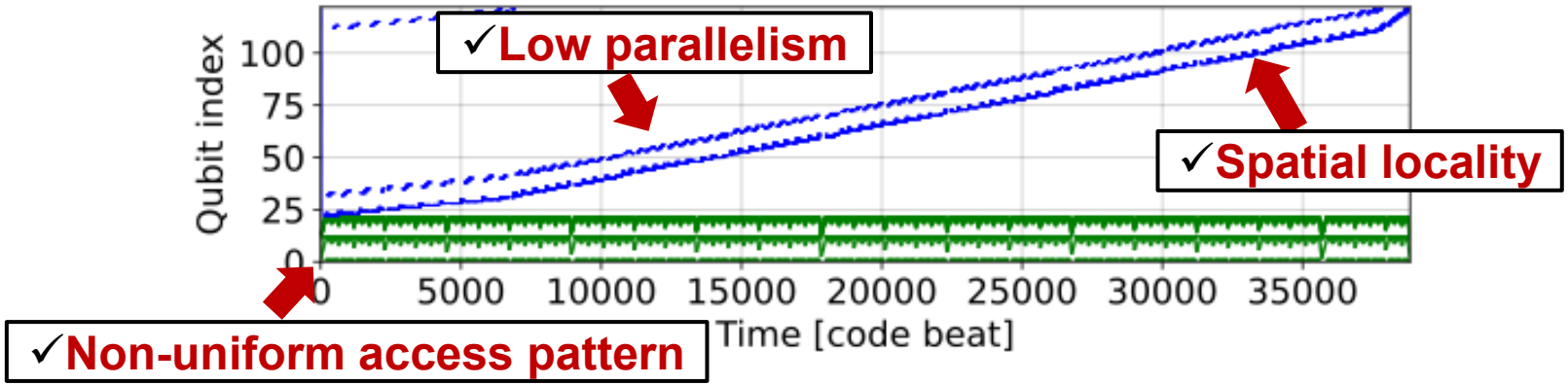


Cumulative distribution of average reference period for SELECT circuit

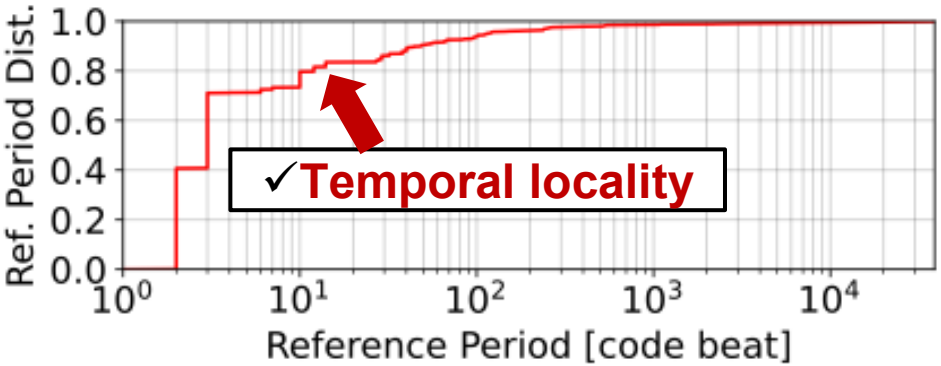


Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation

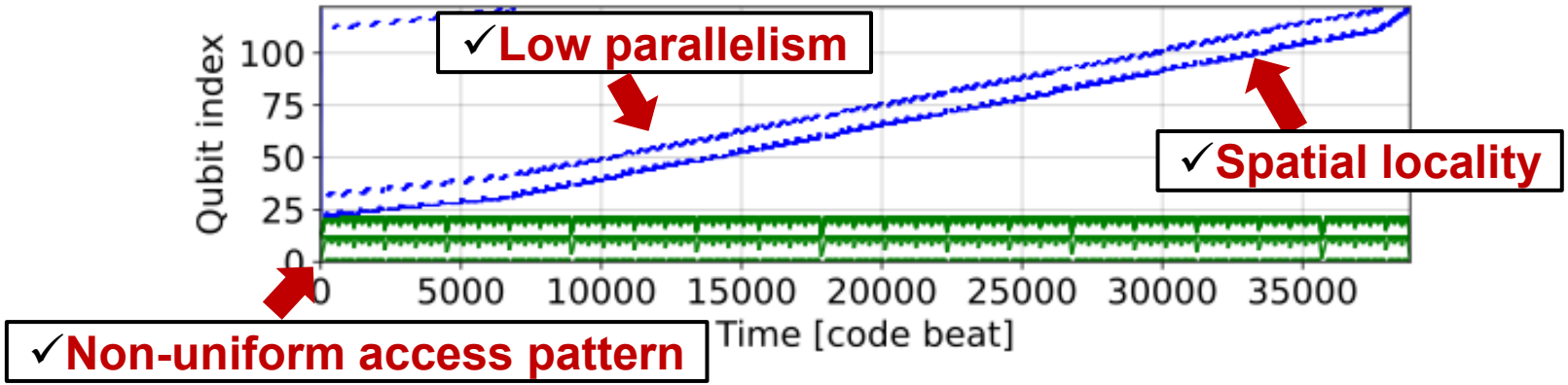


Cumulative distribution of average reference period for SELECT circuit

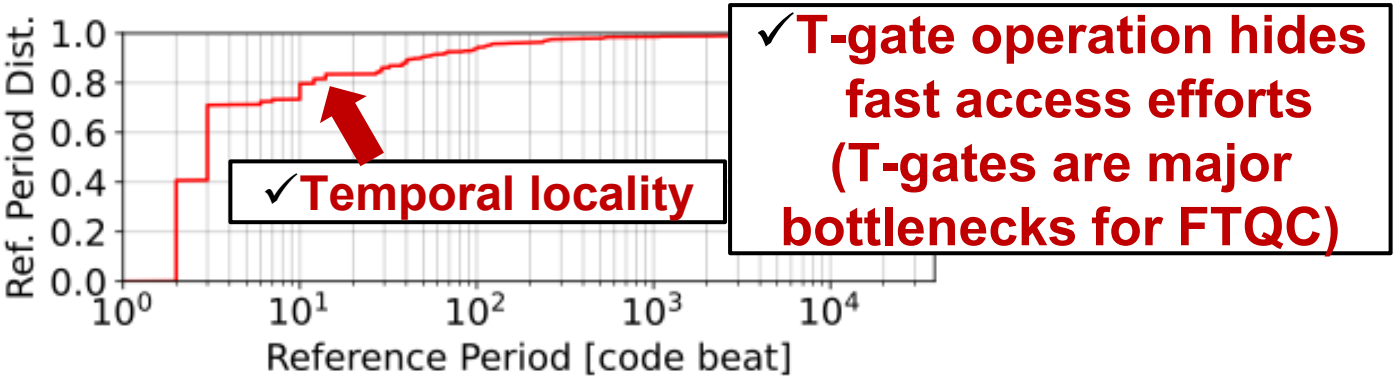


Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation

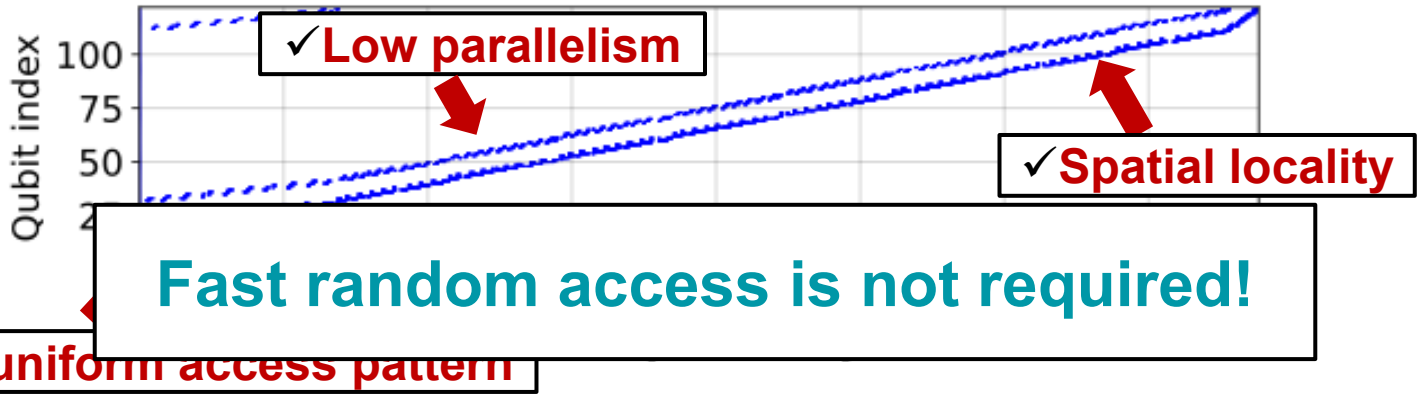


Cumulative distribution of average reference period for SELECT circuit

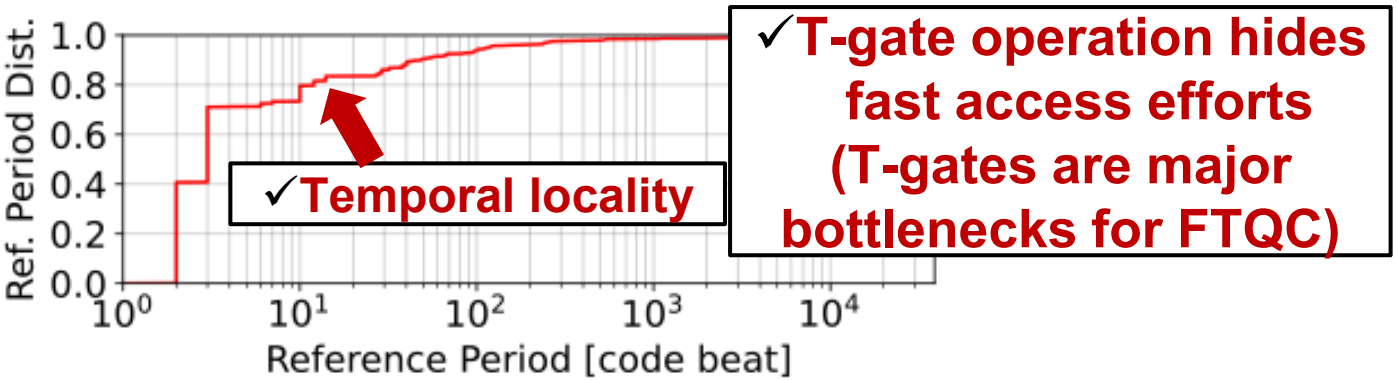


Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation

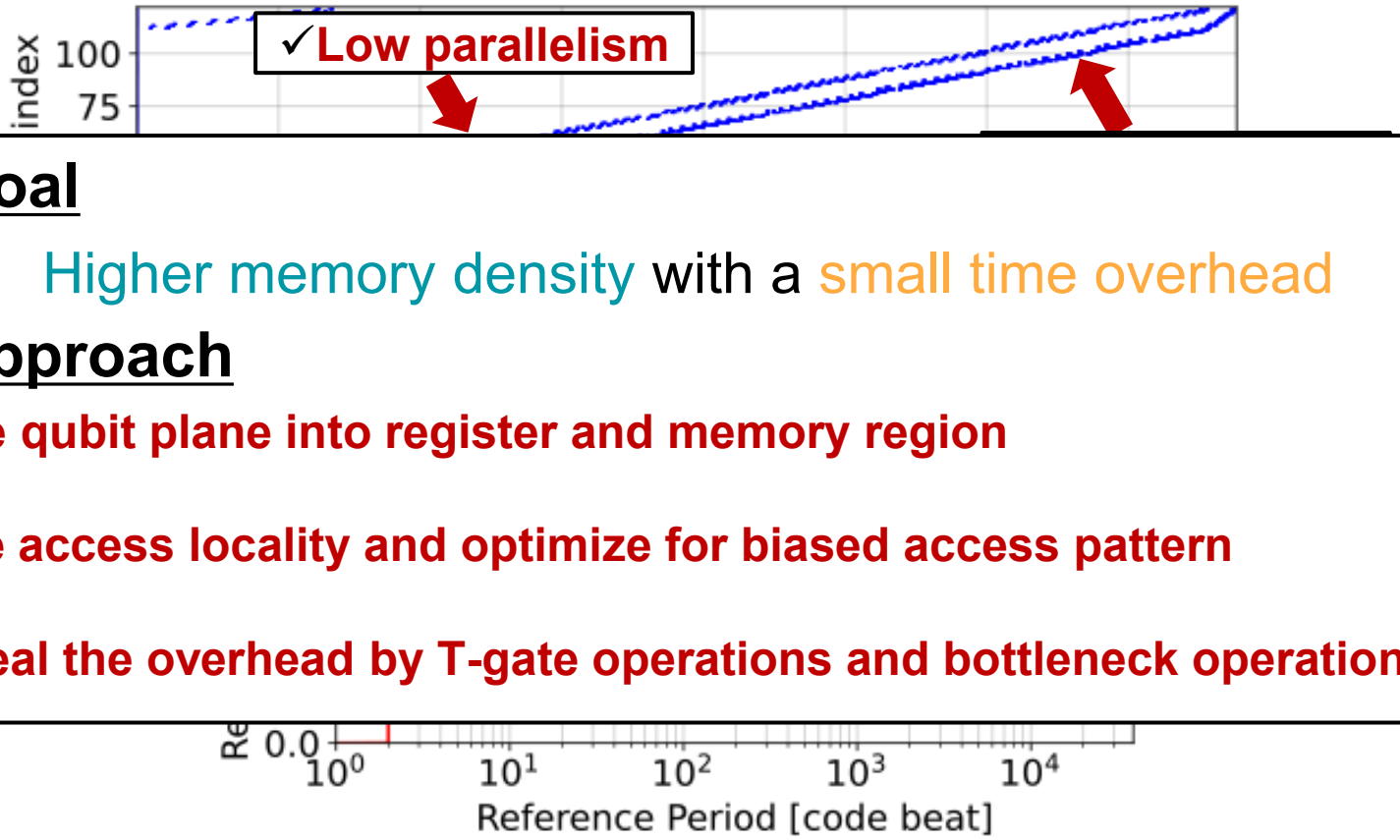


Cumulative distribution of average reference period for SELECT circuit



Motivation: Long Memory-Access Latency Tolerance

Reference time stamp for SELECT subroutine of quantum phase estimation



Our goal

Higher memory density with a small time overhead

Our approach

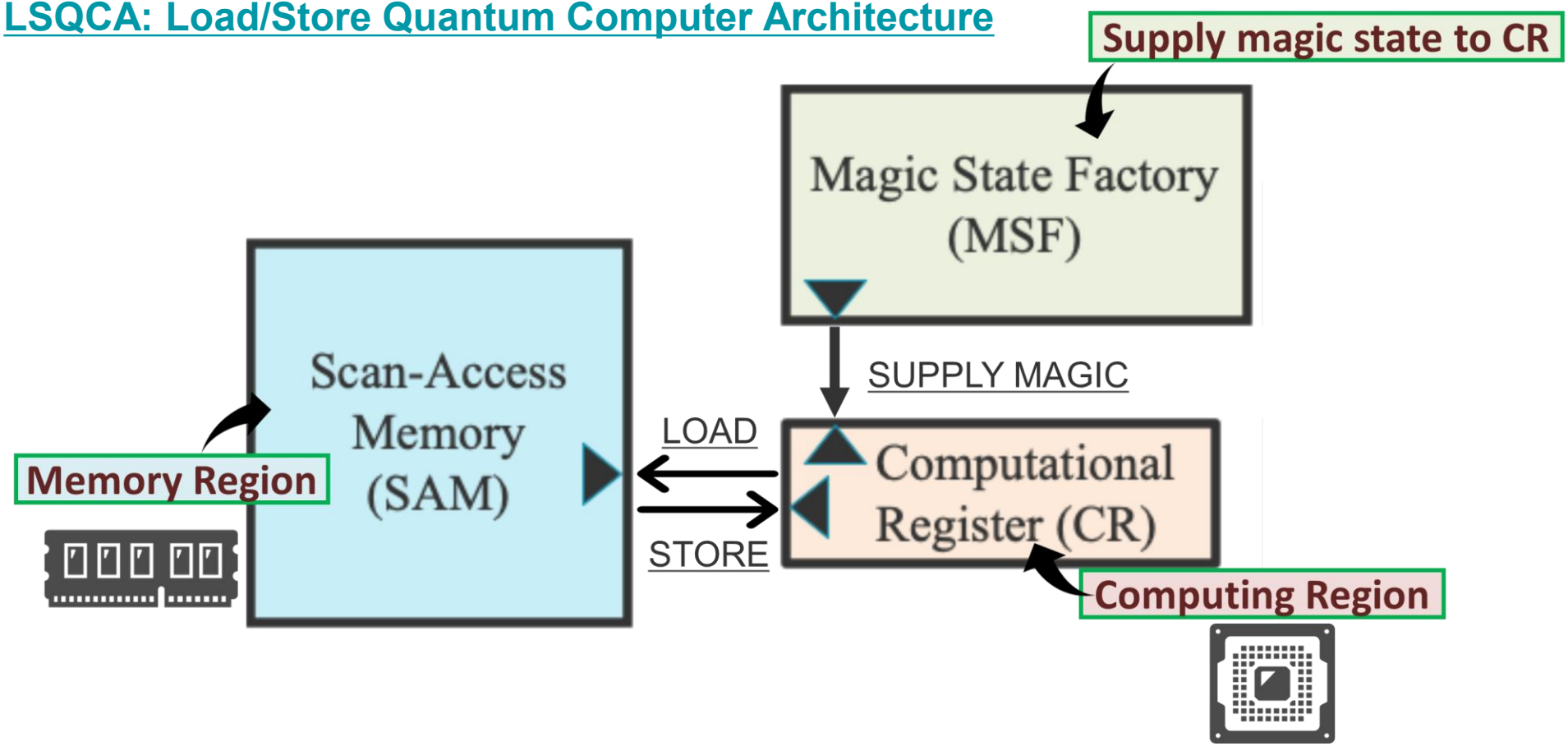
- ✓ **Divide qubit plane into register and memory region**
- ✓ **Utilize access locality and optimize for biased access pattern**
- ✓ **Conceal the overhead by T-gate operations and bottleneck operation**

Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

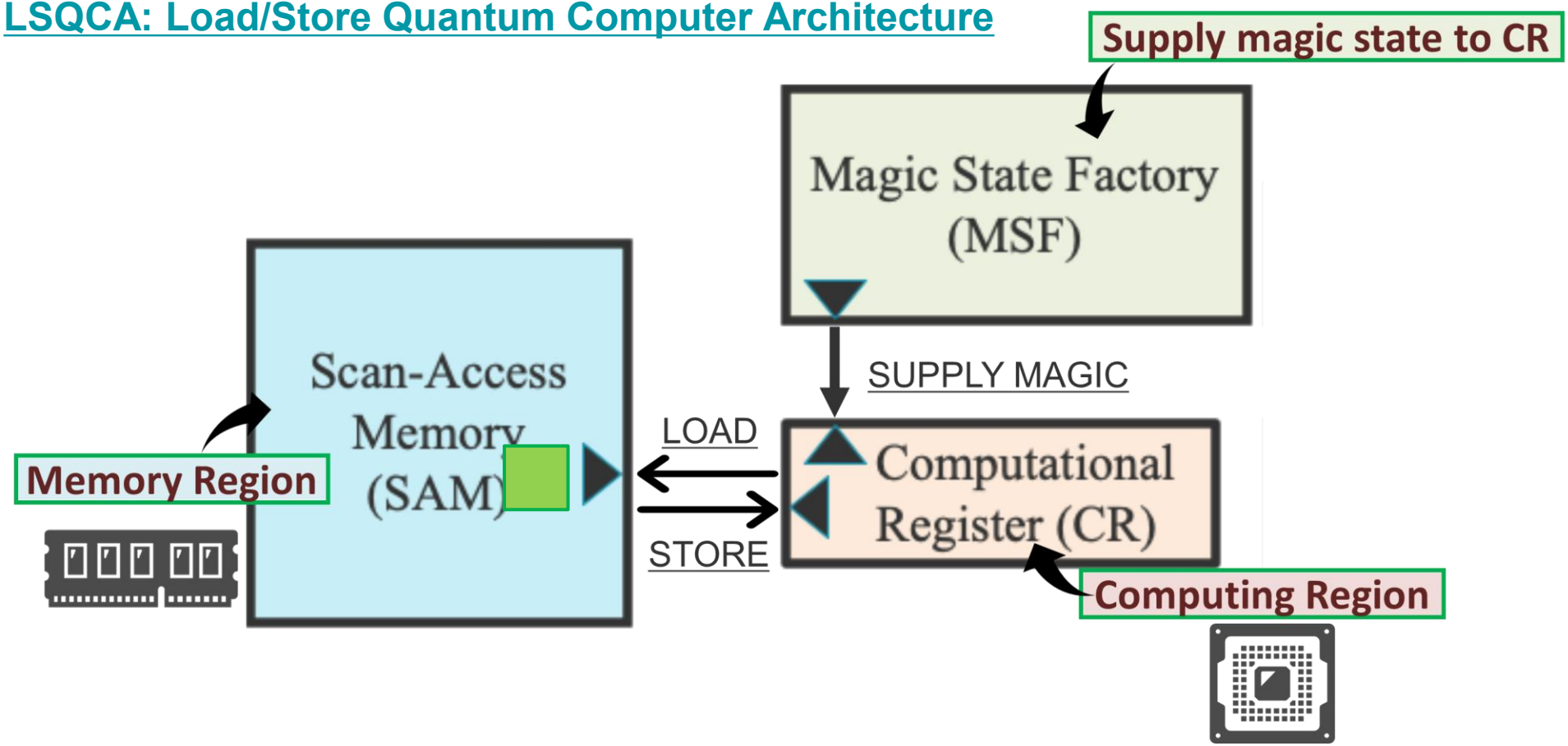
Proposal: Load/Store Architecture on FTQC

LSQCA: Load/Store Quantum Computer Architecture



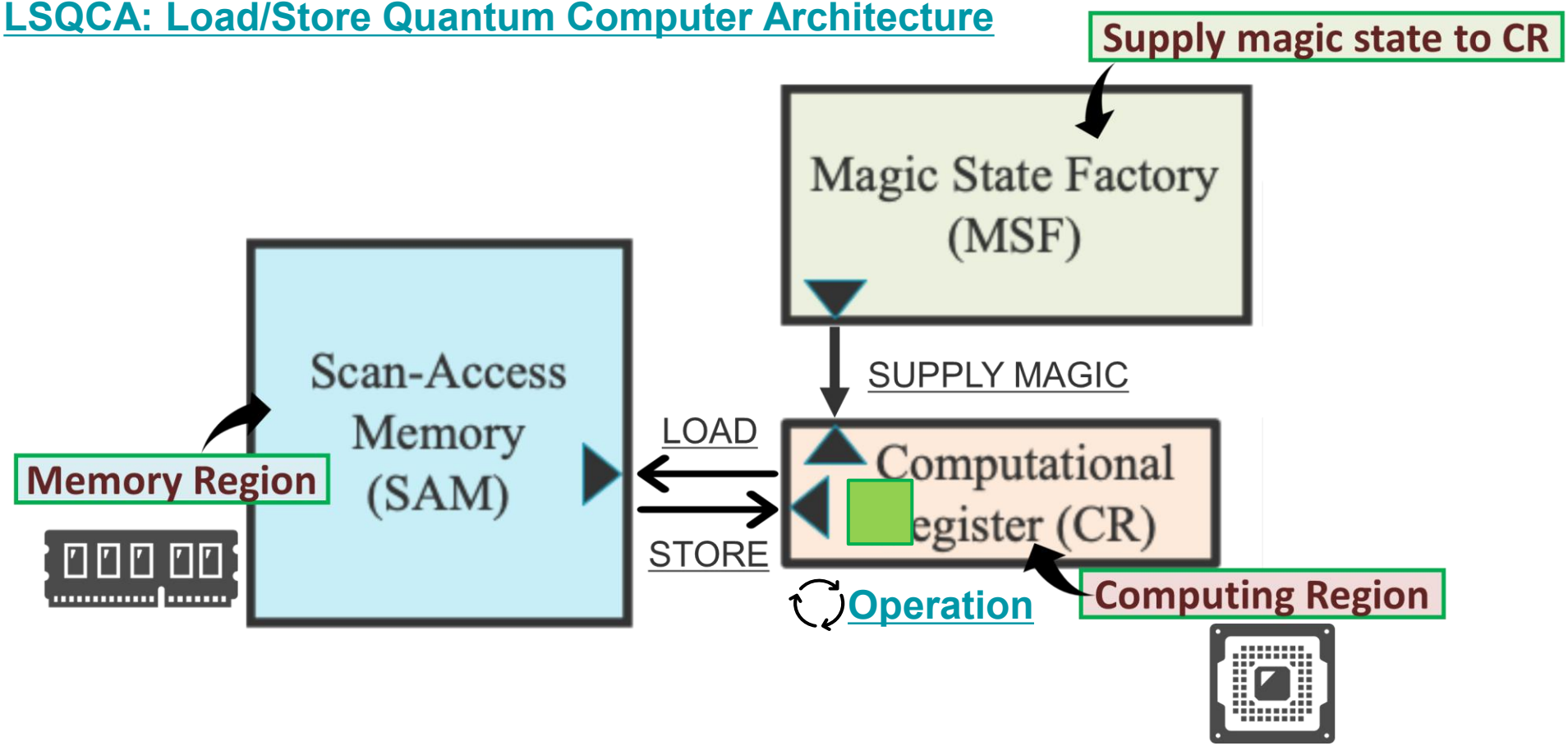
Proposal: Load/Store Architecture on FTQC

LSQCA: Load/Store Quantum Computer Architecture



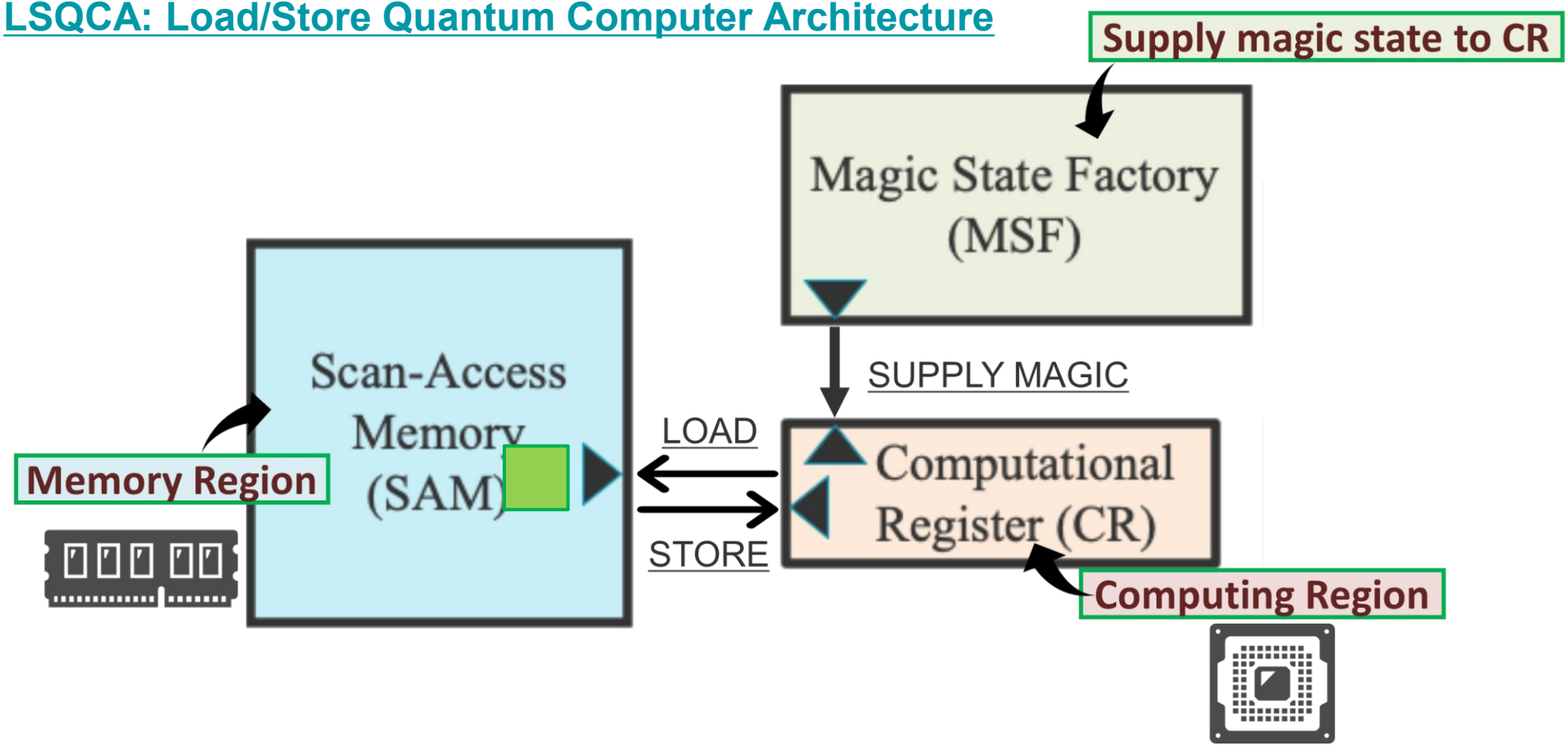
Proposal: Load/Store Architecture on FTQC

LSQCA: Load/Store Quantum Computer Architecture



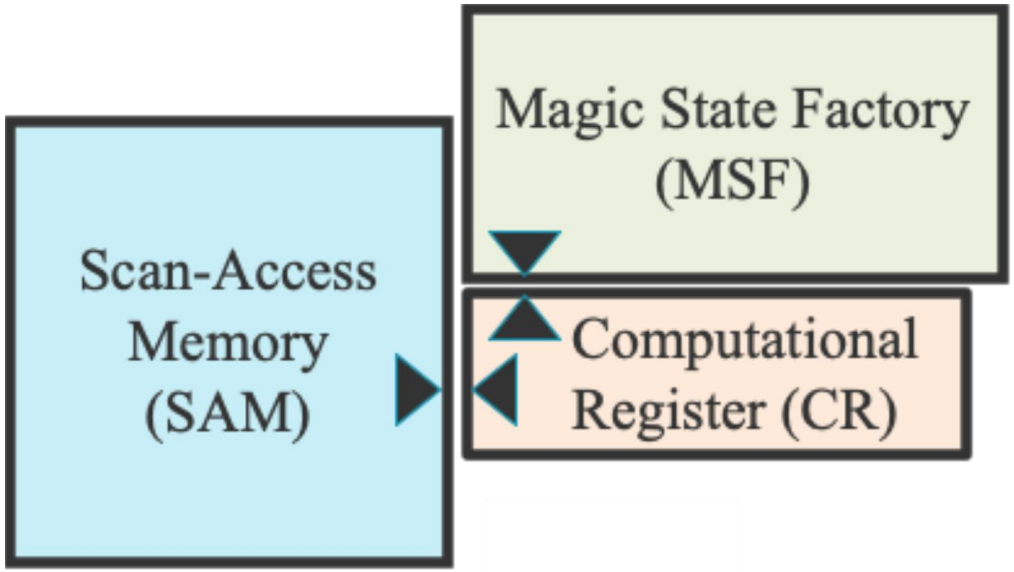
Proposal: Load/Store Architecture on FTQC

LSQCA: Load/Store Quantum Computer Architecture



Proposal: Load/Store Architecture on FTQC

LSQCA: Load/Store Quantum Computer Architecture



(+) High memory density

(-) Longer and variable latency of Load/Store instructions

(+) Our ISA used for different configurations

- Not limited to 2d plain or surface code
- e.g., surface code for register and LDPC code for memory region

ISA of our load/store architecture for FTQC

Load/ Store

Type	Syntax	Latency	Description
Memory	`LD M C`	variable	(Load) Load logical qubit from SAM to CR
	`ST C M`	variable	(Store) Store logical qubit from CR to SAM
Preparation	`PZ.C C`	0 beat	(Zero-Init) Initialize a logical qubit to $\|0\rangle$ state
	`PP.C C`	0 beat	(Plus-Init) Initialize a logical qubit to $\|+\rangle$ state
	`PM C`	variable	(Magic-init) Move magic state from MSF to CR
Unitary	`HD.C C`	3 beat	(Hadamard) Hadamard gate on a logical qubit
	`PH.C C`	2 beat	(Phase) Phase gate on a logical qubit
Measurement	`MX.C C V`	0 beat	(Pauli-X Meas) Pauli-X measurement on a logical qubit and store outcome
	`MZ.C C V`	0 beat	(Pauli-Z Meas) Pauli-Z measurement on a logical qubit and store outcome
	`MXX.C C1 C2 V`	1 beat	(Pauli-XX Meas) Pauli-XX measurement on logical qubits and store outcome
	`MZZ.C C1 C2 V`	1 beat	(Pauli-ZZ Meas) Pauli-ZZ measurement on logical qubits and store outcome
Control	`SK V`	variable	(Skip) Skip next instruction if a provided value is zero
In-Memory Preparation	`PZ.M M`	0 beat	(Zero-Init) Initialize a logical qubit to $\|0\rangle$ state
	`PP.M M`	0 beat	(Plus-Init) Initialize a logical qubit to $\|+\rangle$ state
In-Memory Unitary	`HD.M M`	variable	(Hadamard) Hadamard gate on a logical qubit
	`PH.M M`	variable	(Phase) Phase gate on a logical qubit
In-Memory Measurement	`MX.M M V`	0 beat	(Pauli-X Meas) Pauli-X measurement on a logical qubit and store outcome
	`MZ.M M V`	0 beat	(Pauli-Z Meas) Pauli-Z measurement on a logical qubit and store outcome
	`MXX.M C M V`	variable	(Pauli-XX Meas) Pauli-XX measurement on logical qubits and store outcome
	`MZZ.M C M V`	variable	(Pauli-ZZ Meas) Pauli-ZZ measurement on logical qubits and store outcome
Optimized Unitary	`CX M1 M2`	variable	(CNOT) CNOT gate on logical qubits

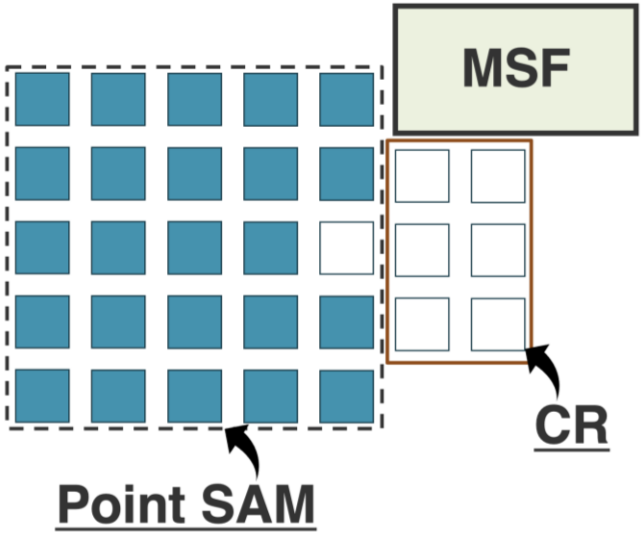
- ✓ Load/Store instructions are independent of memory region implementation (e.g., QEC code, allocation)
- ✓ Enable of abstraction, easily programable, and high program portability

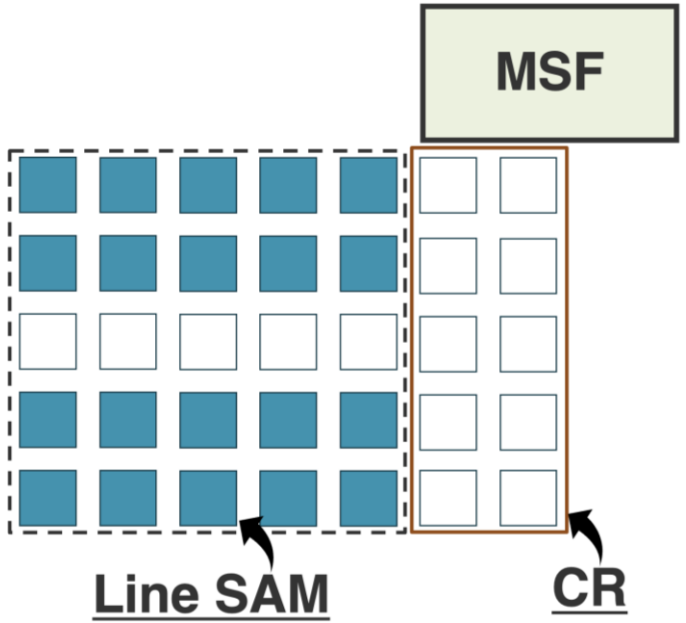
Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

Implementation of LSQCA with LS-based FTQC

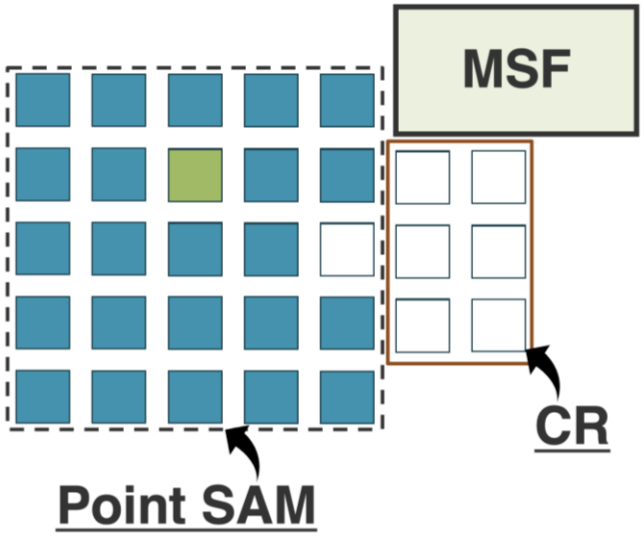
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

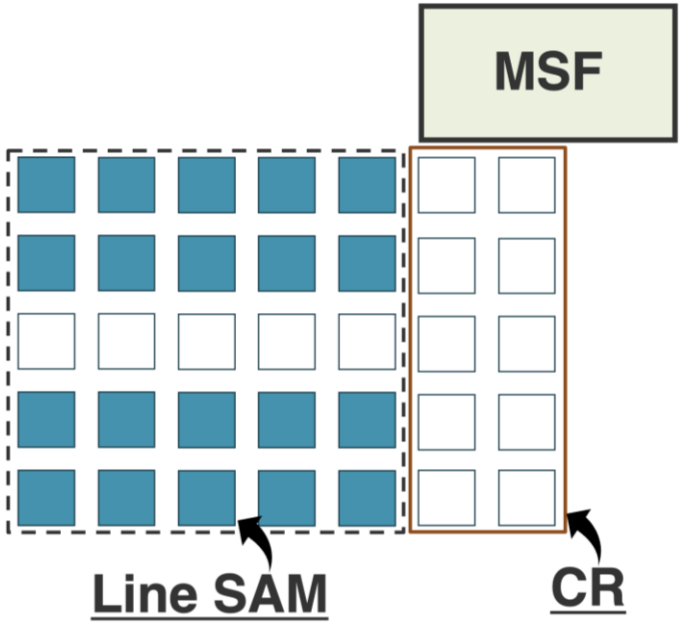




Implementation of LSQCA with LS-based FTQC

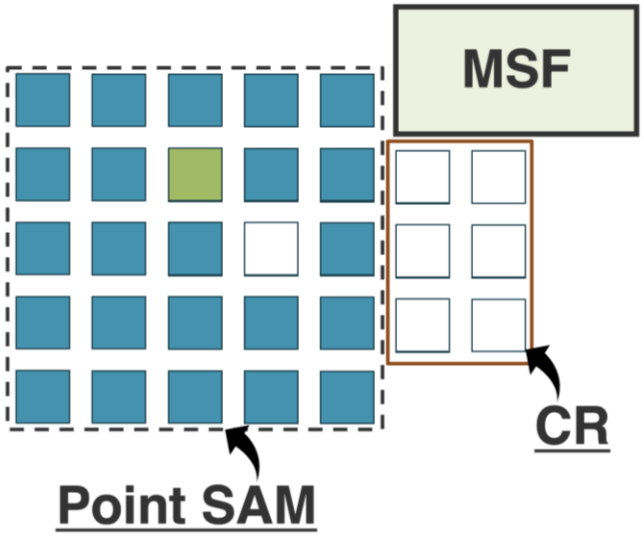
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

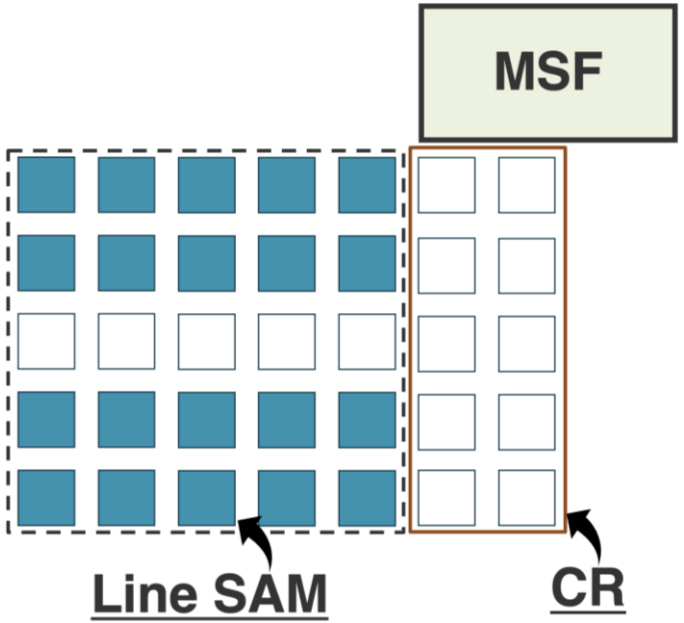




Implementation of LSQCA with LS-based FTQC

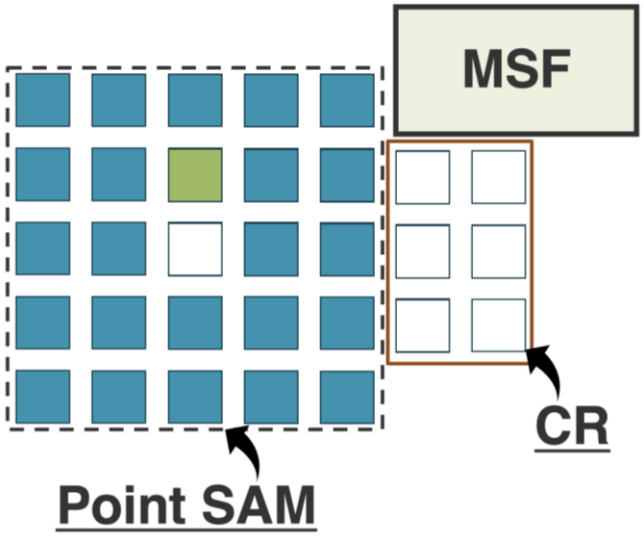
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

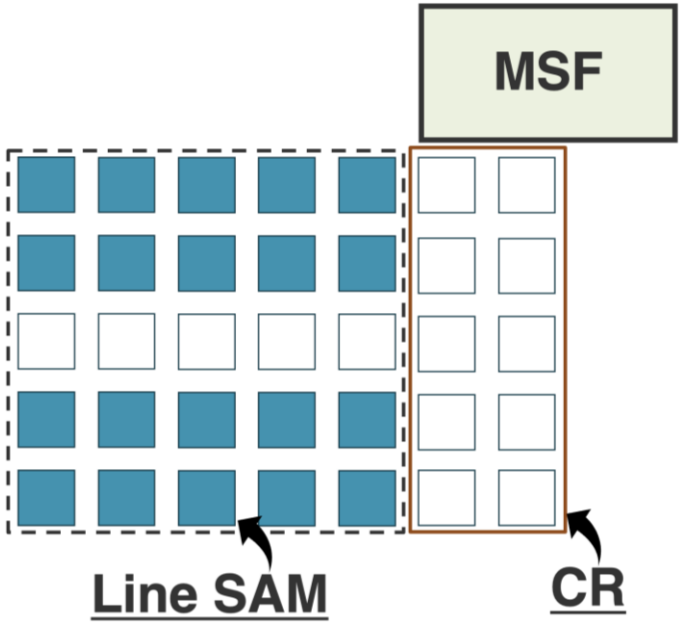




Implementation of LSQCA with LS-based FTQC

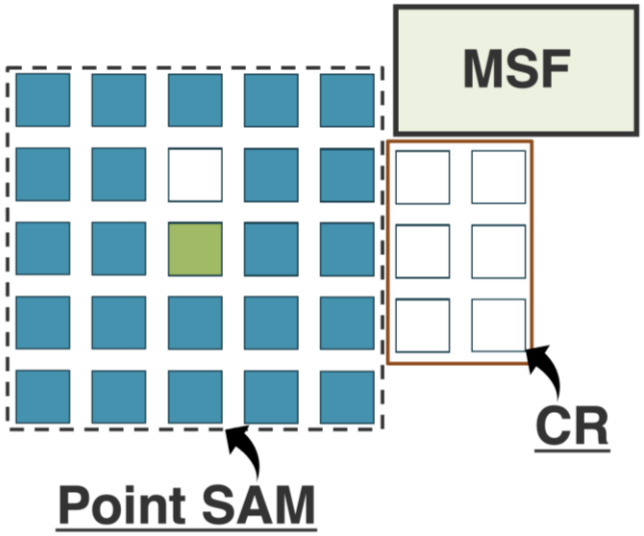
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

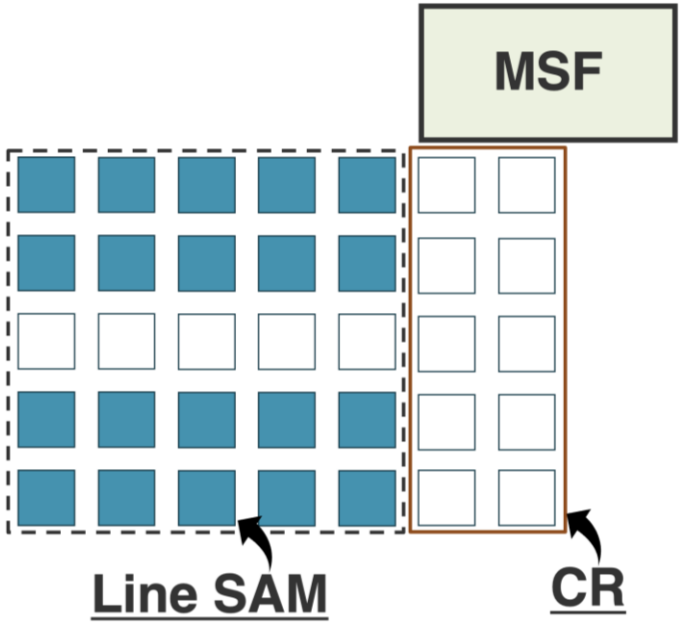




Implementation of LSQCA with LS-based FTQC

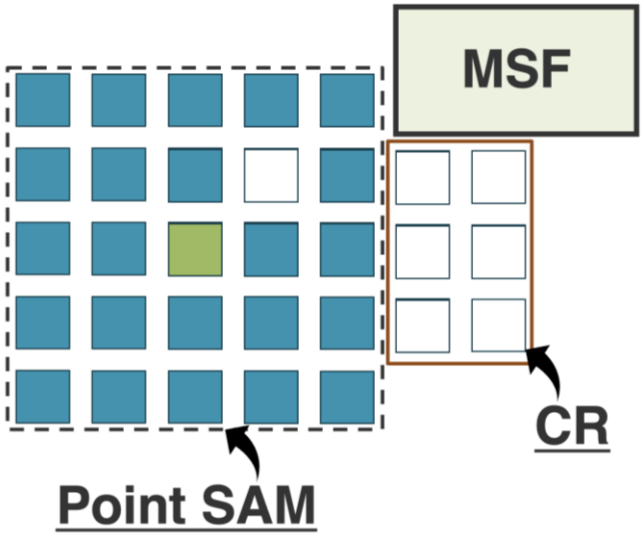
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

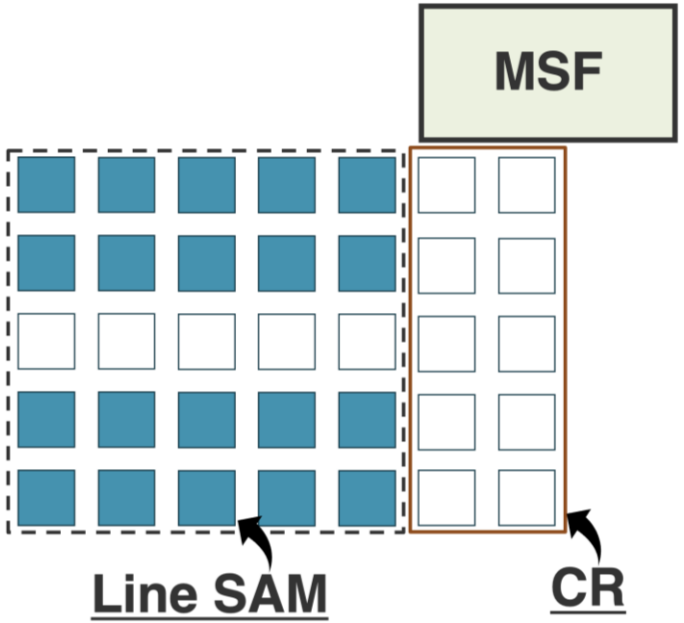




Implementation of LSQCA with LS-based FTQC

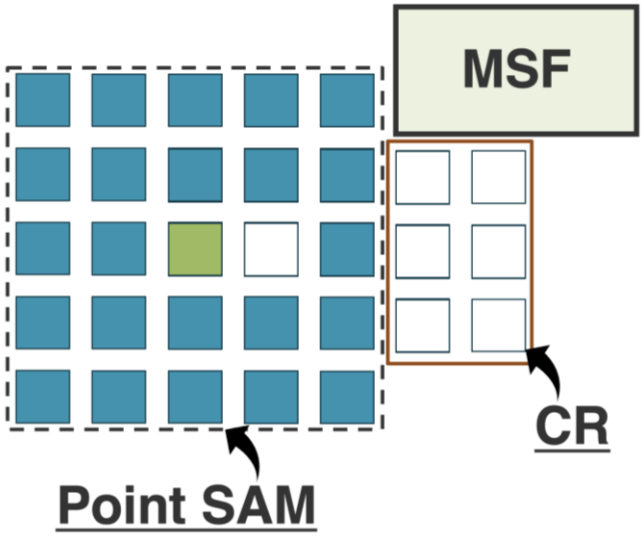
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

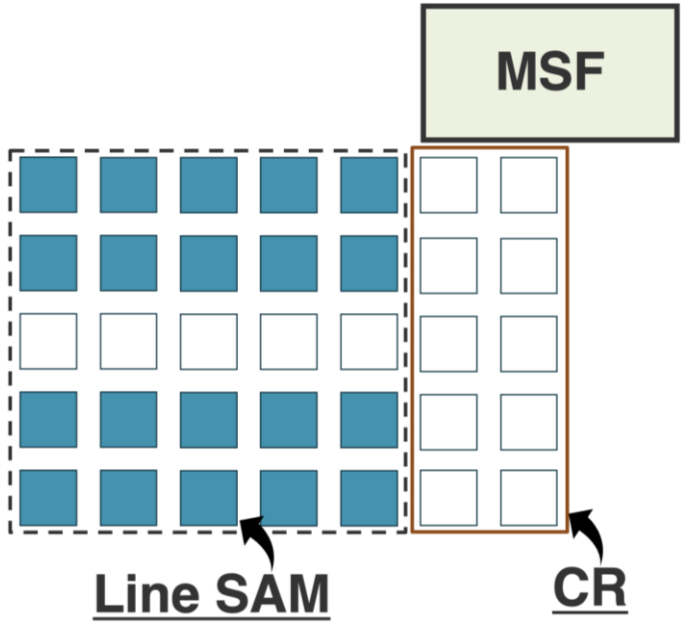




Implementation of LSQCA with LS-based FTQC

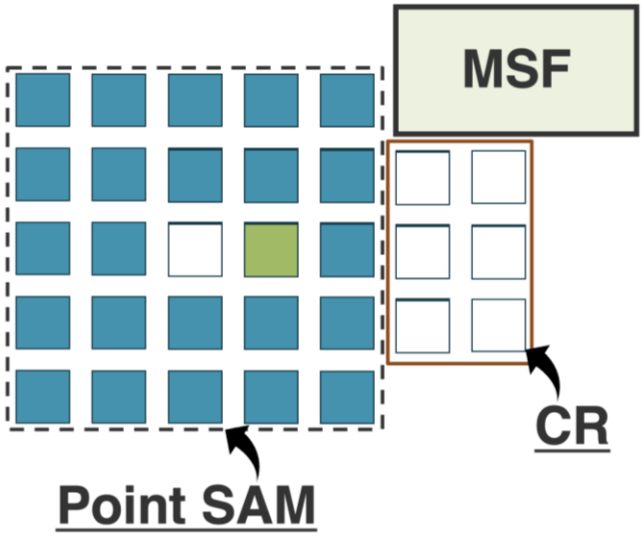
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

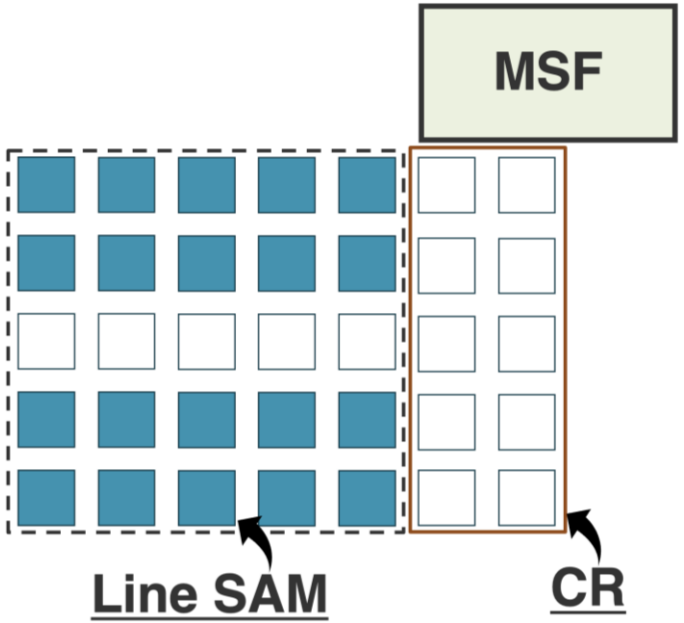




Implementation of LSQCA with LS-based FTQC

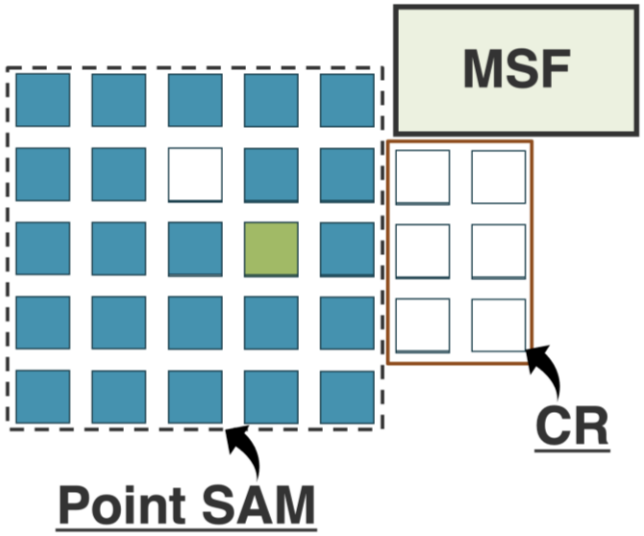
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

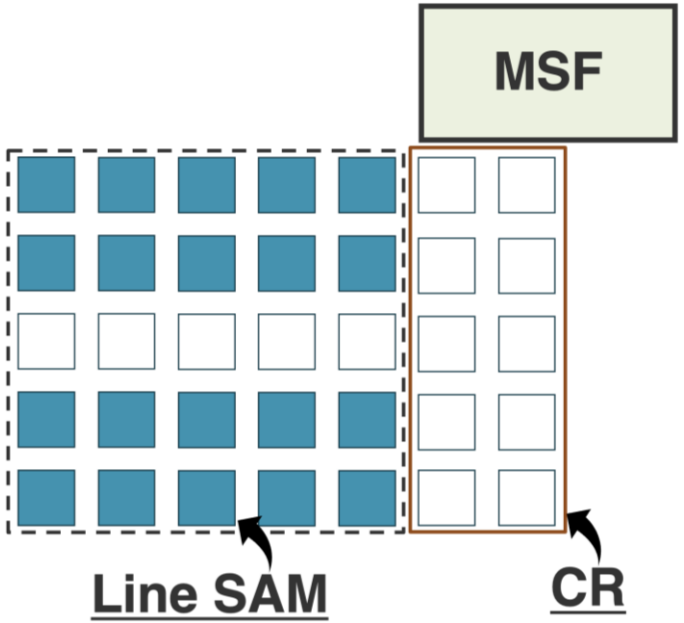




Implementation of LSQCA with LS-based FTQC

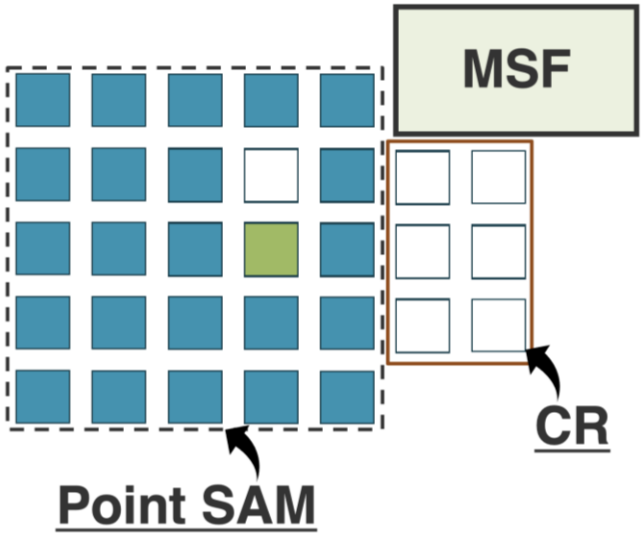
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

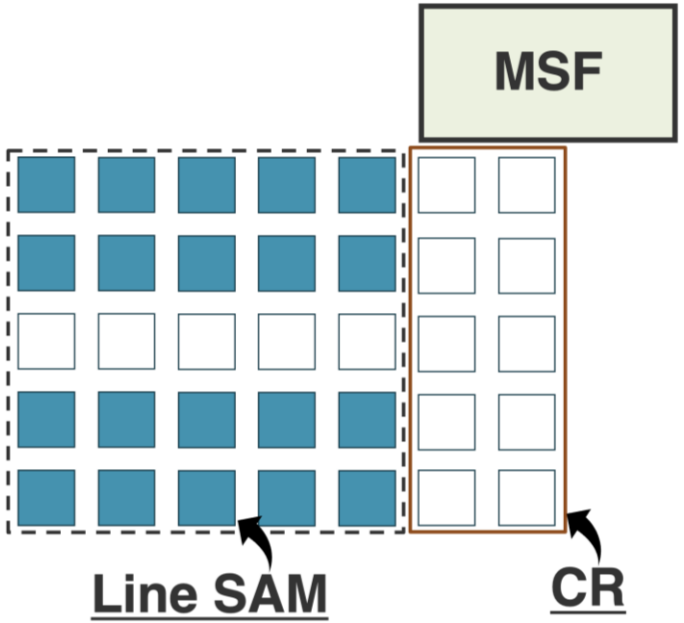




Implementation of LSQCA with LS-based FTQC

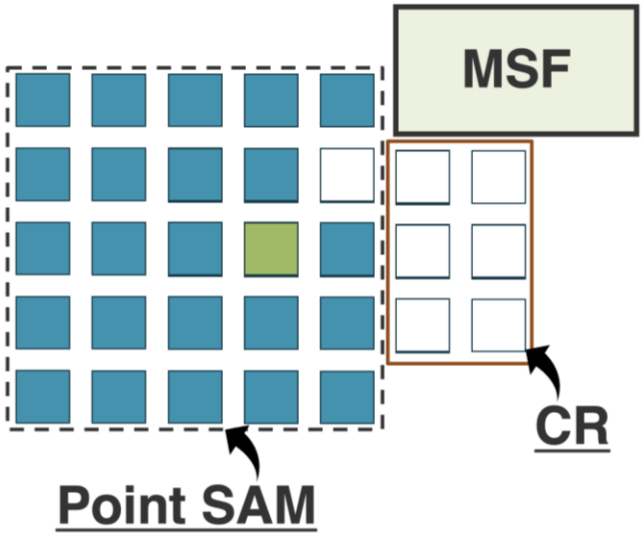
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

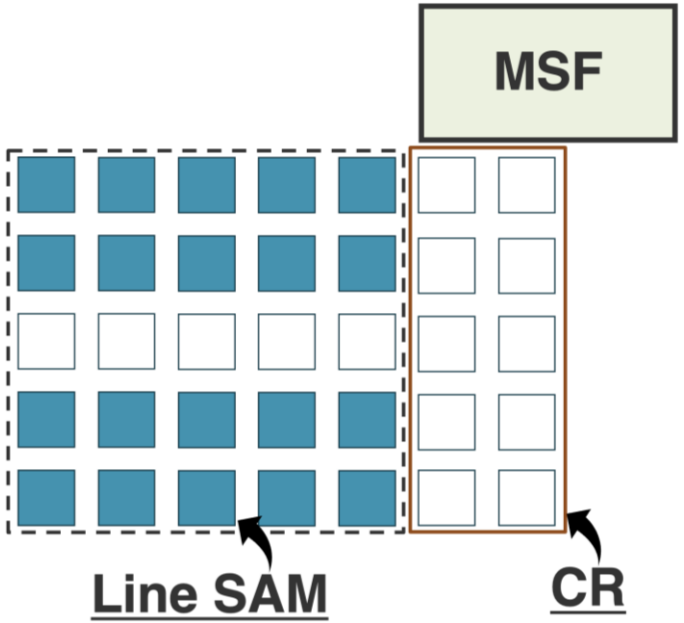




Implementation of LSQCA with LS-based FTQC

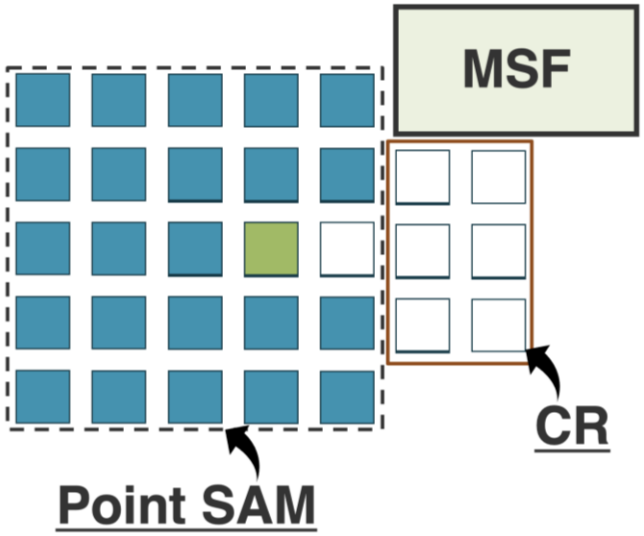
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

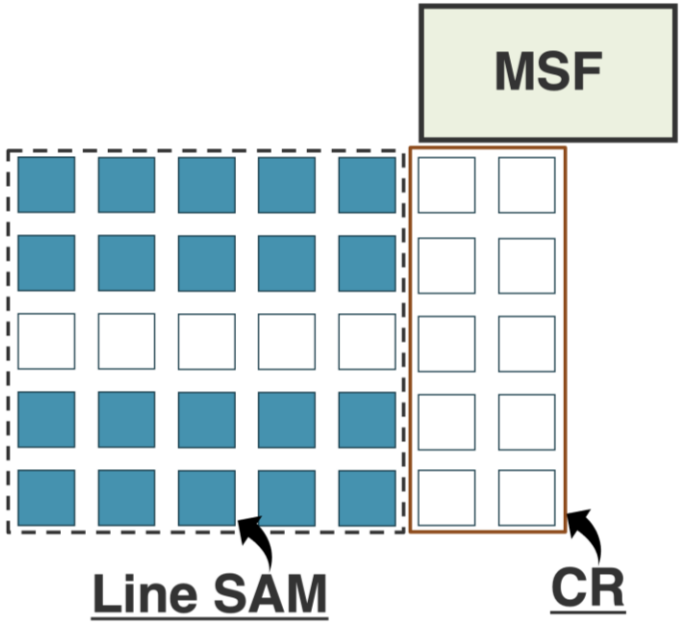




Implementation of LSQCA with LS-based FTQC

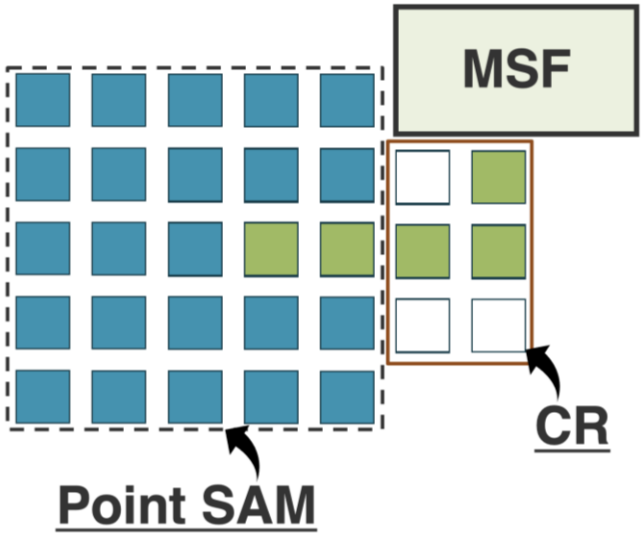
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

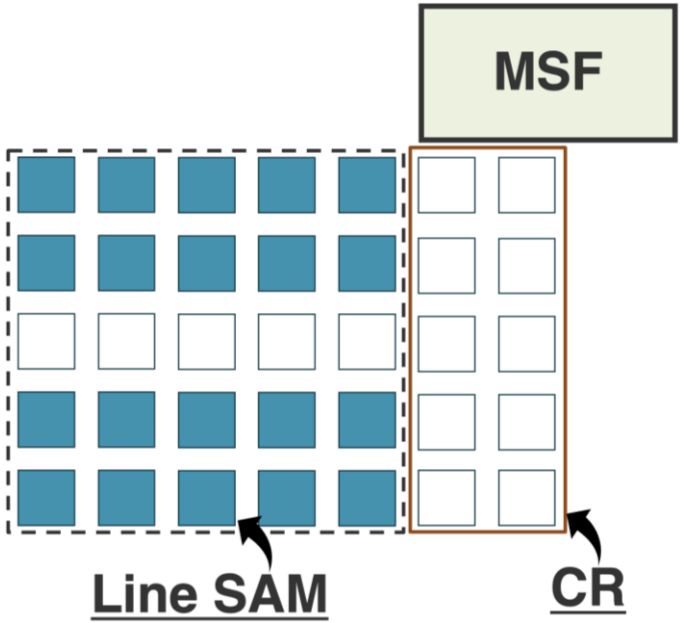




Implementation of LSQCA with LS-based FTQC

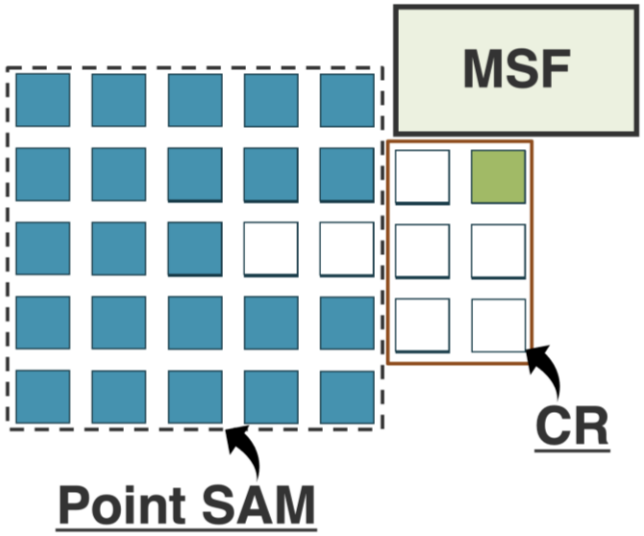
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency

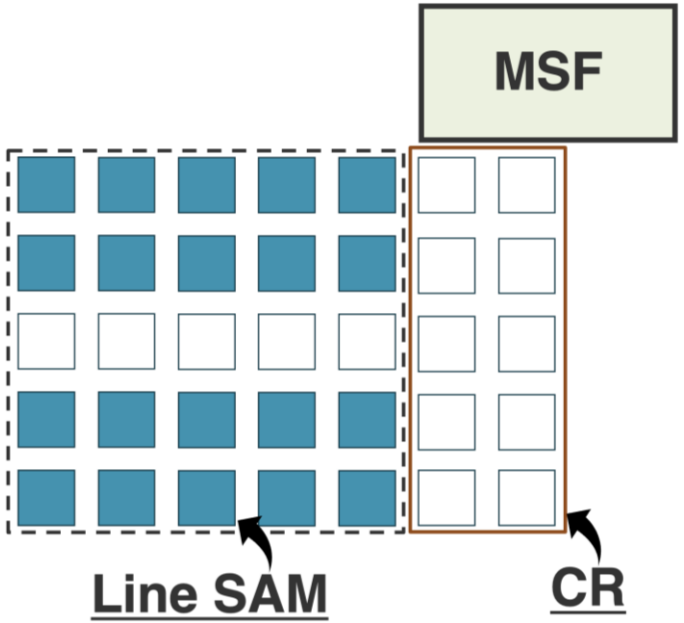




Implementation of LSQCA with LS-based FTQC

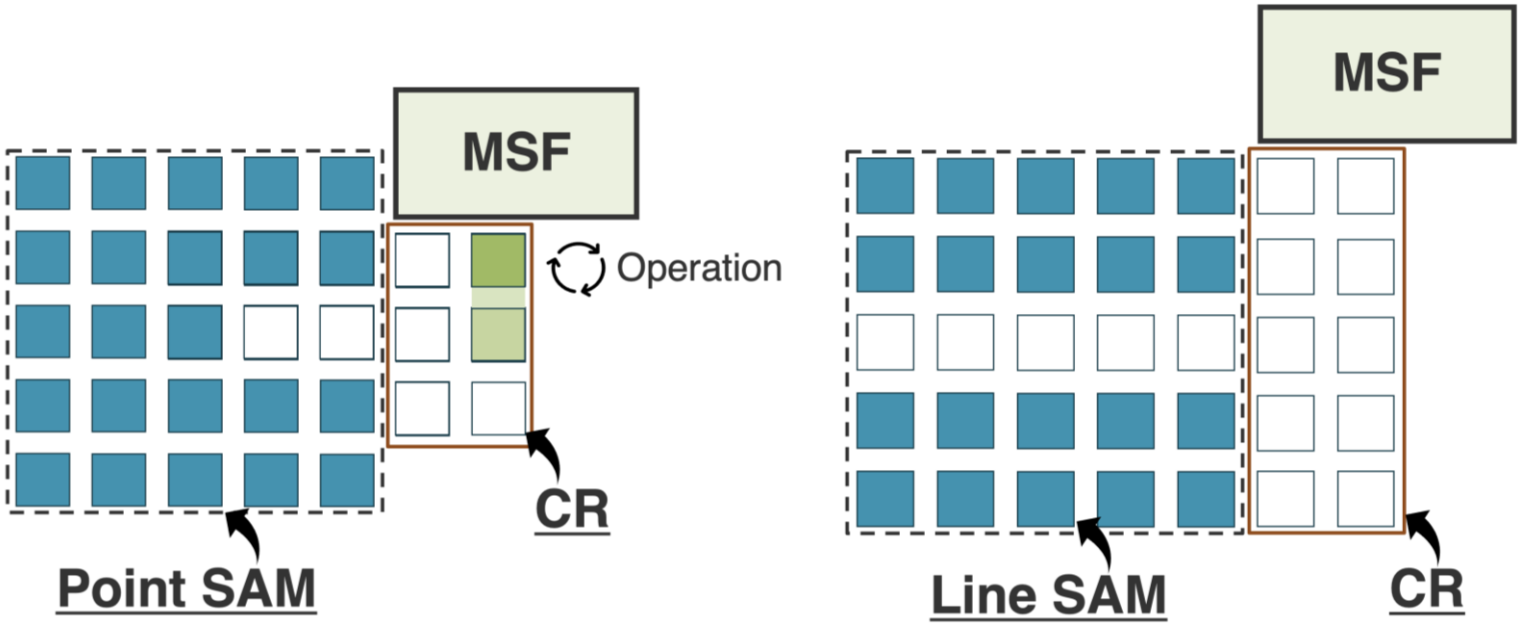
- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency





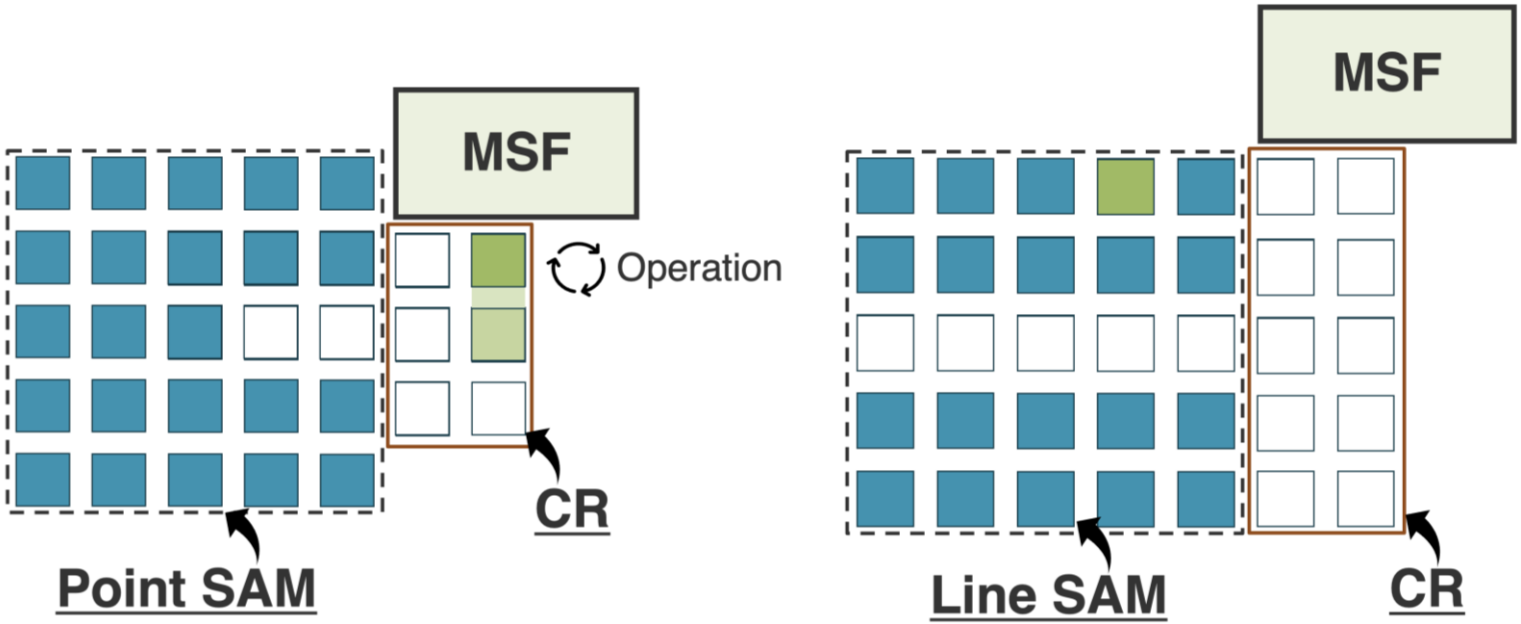
Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



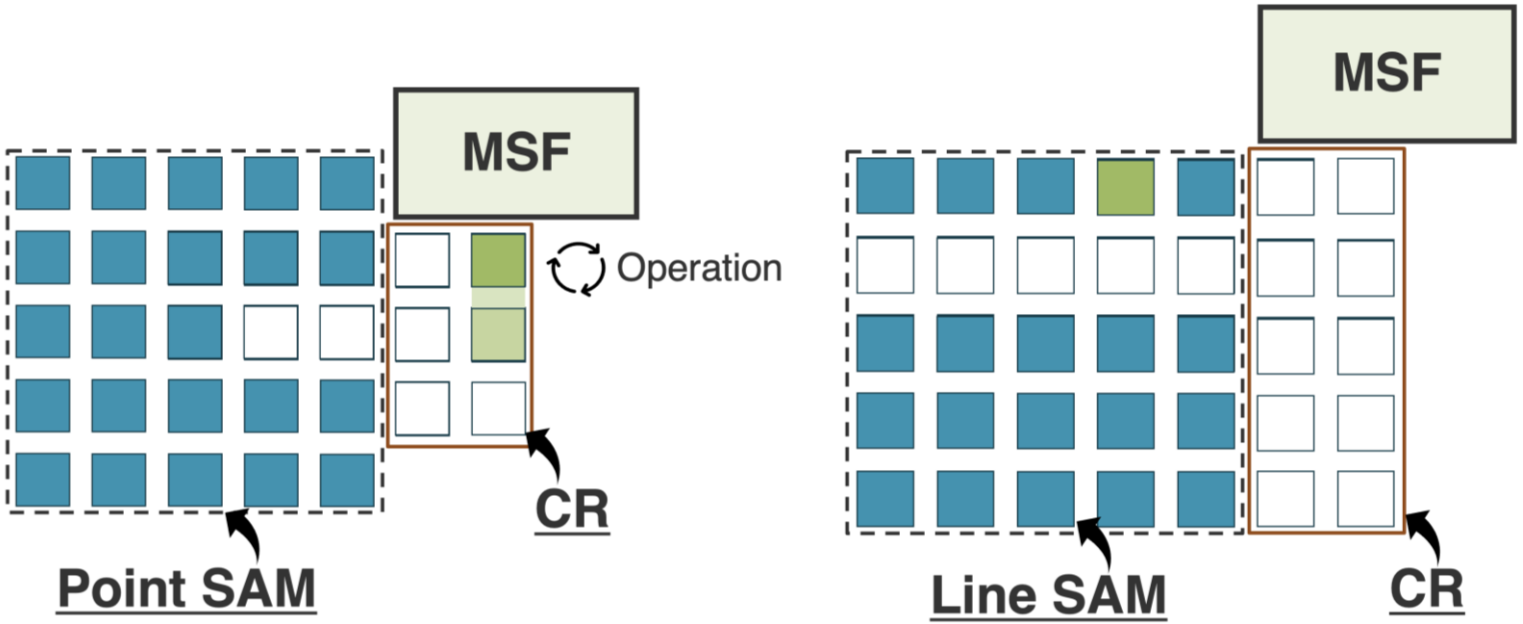
Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



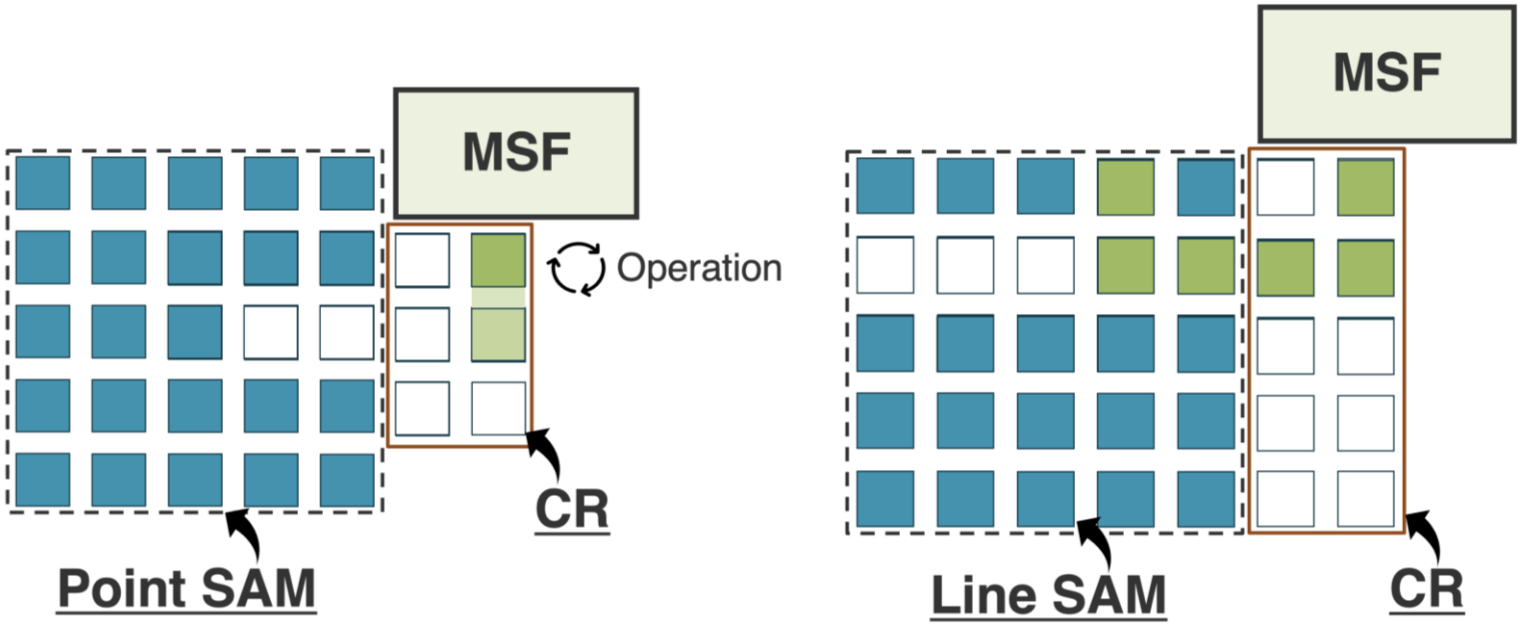
Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



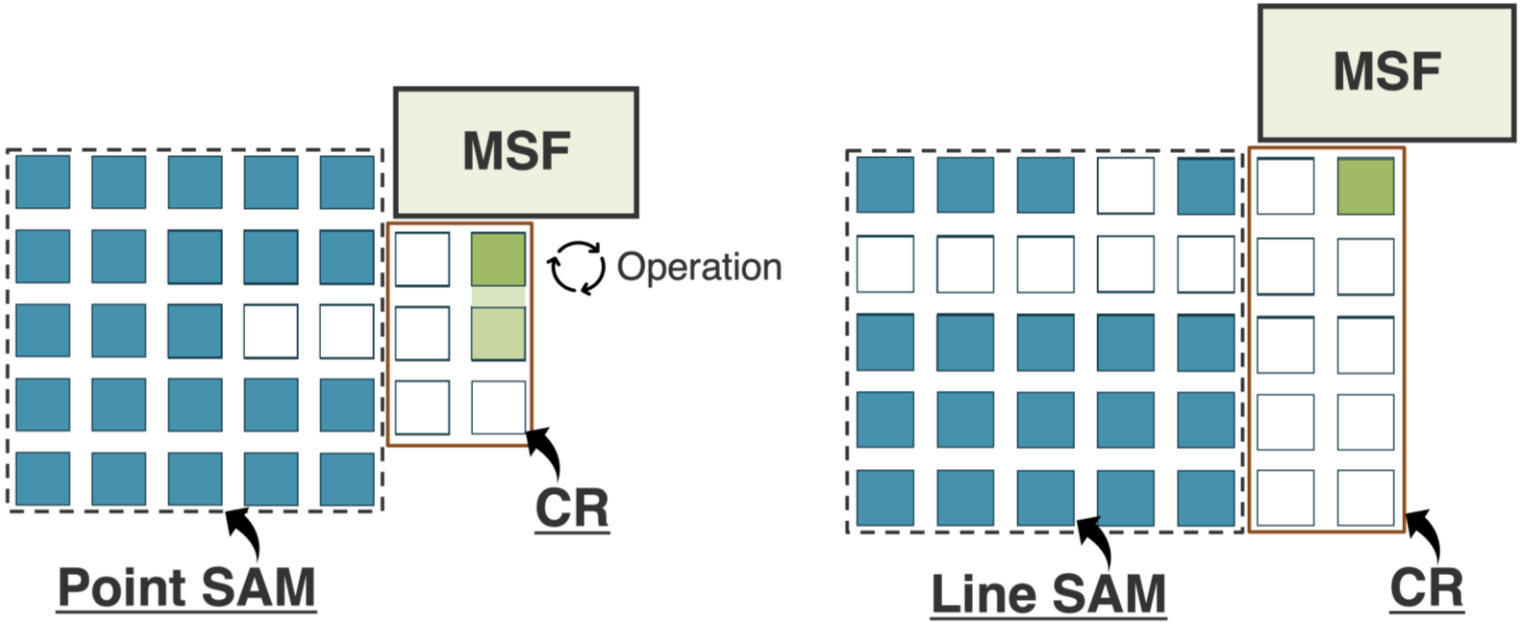
Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



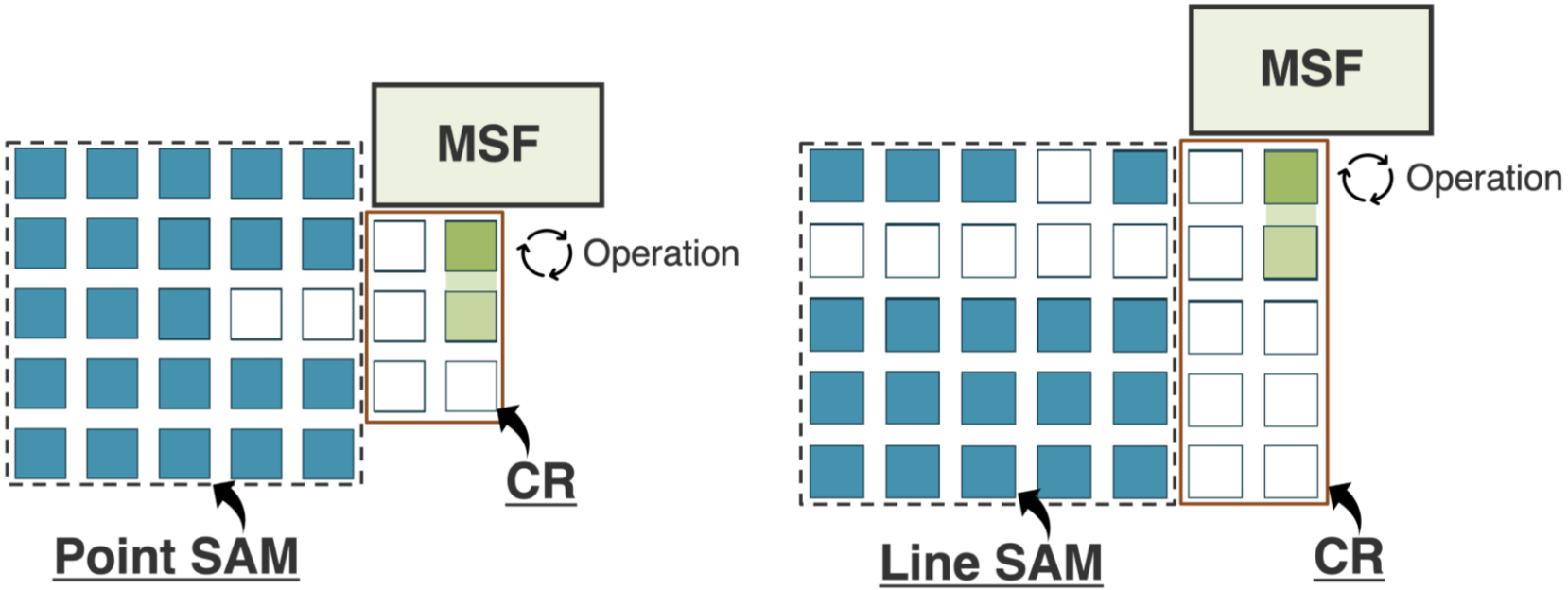
Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



Implementation of LSQCA with LS-based FTQC

- Two implementations of SAM for LS: Point and Line SAM
  - (+) Asymptotically 100% memory density
  - (-) Long access latency



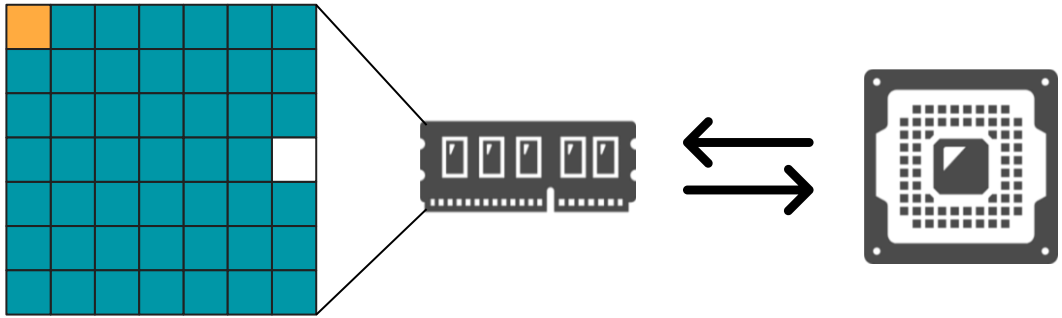
Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

Disadvantage of Load/Store Architecture

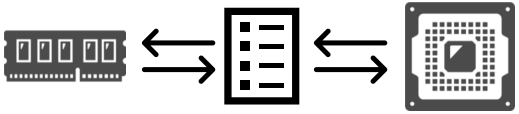
(-) Decreasing performance due to long memory access latency

Example of worst case

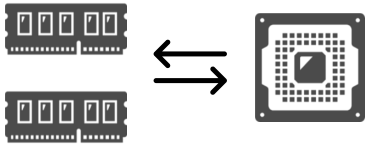


✓ **Classical computer uses many techniques for improving memory access latency**

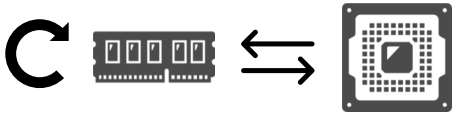
Cache / Scratchpad memory



Multi-channel memory



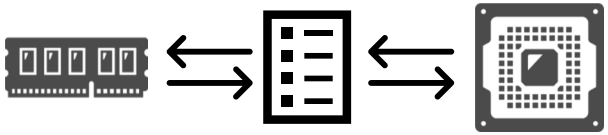
In-memory computing



Optimizing SAM to improve memory access latency

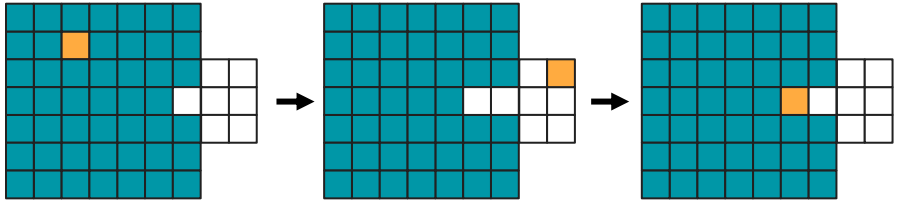
Idea1: Cache/Scratchpad memory

Store frequently accessed or predicted-to-be-accessed data in fast-access location.

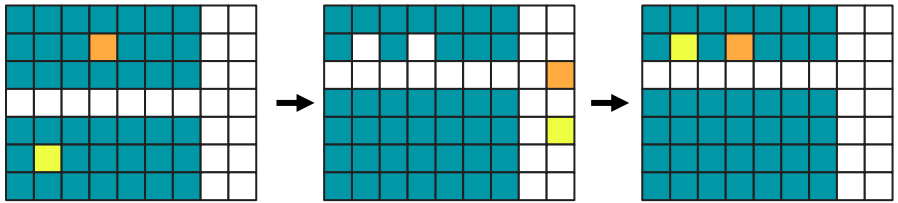


Optimization1: Locality-aware store

Store used data to a fast-access location, not its original place to leverage temporal access locality.

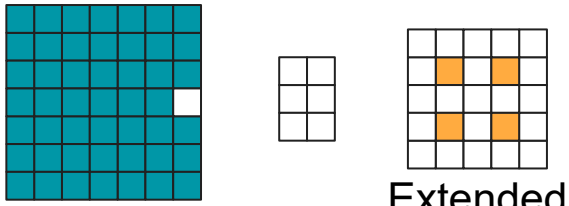


Store simultaneously used data to a simultaneously accessible location to leverage spatial access locality.



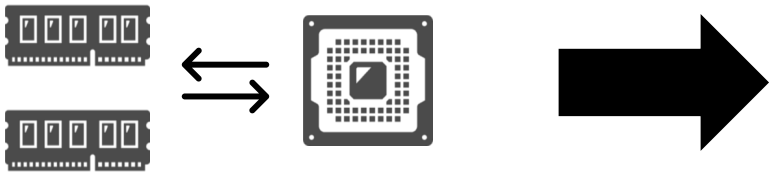
Optimization2: Hybrid Floorplan

Expand register and place a small amount of frequently accessed data in a dedicated area based on static analysis



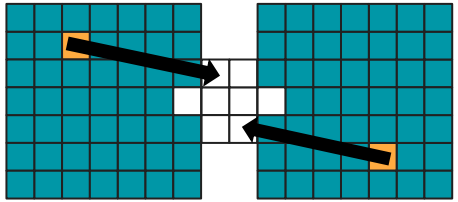
Optimizing SAM to improve memory access latency

Idea2: Multi-channel memory

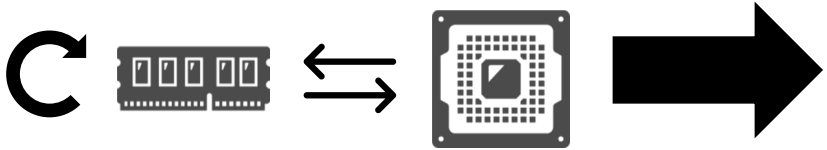


Optimization3: Multi-bank SAM

Divide and place data into multiple SAMs, enabling parallel LOAD/STORE

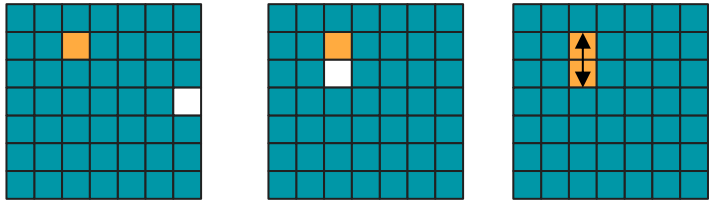


Idea3: In-memory computing



Optimization4: In-memory operations

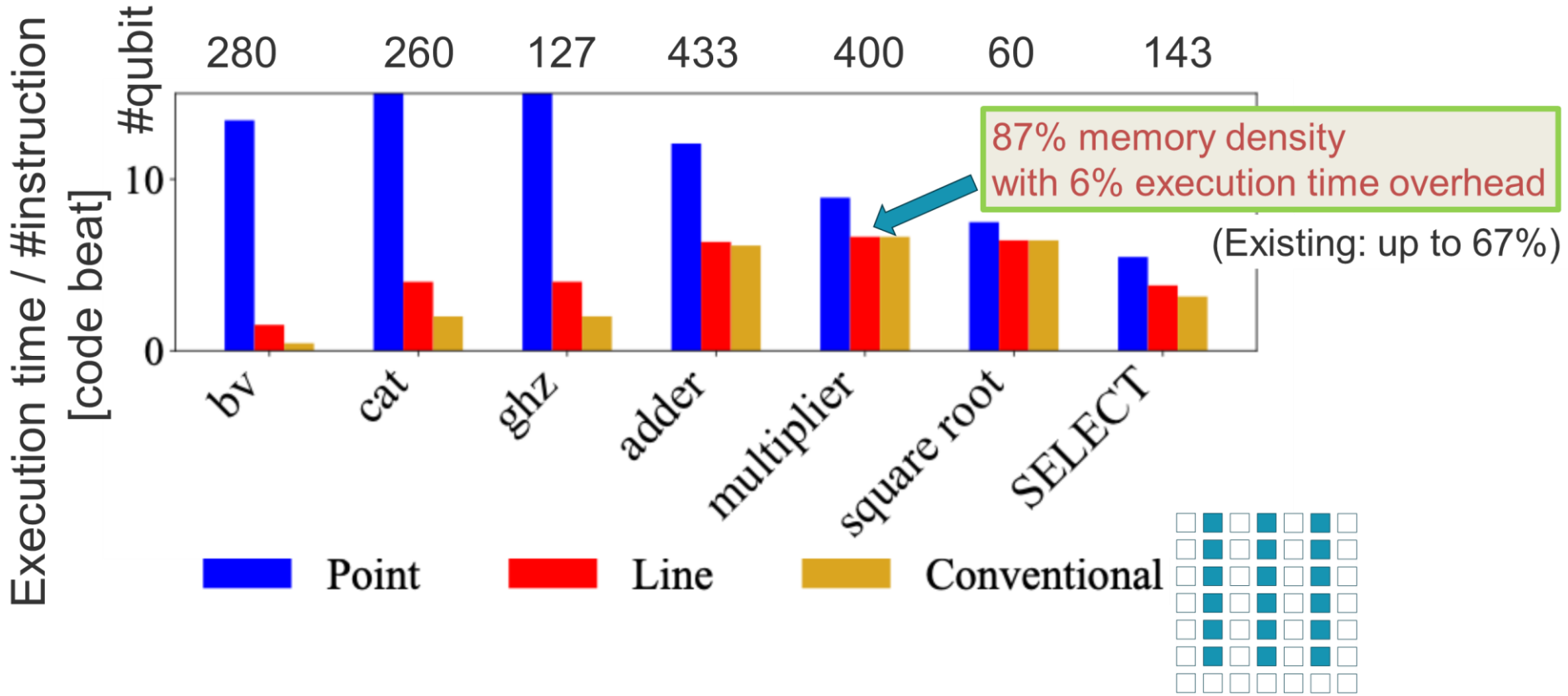
Performing logical H or S gates in memory region without load



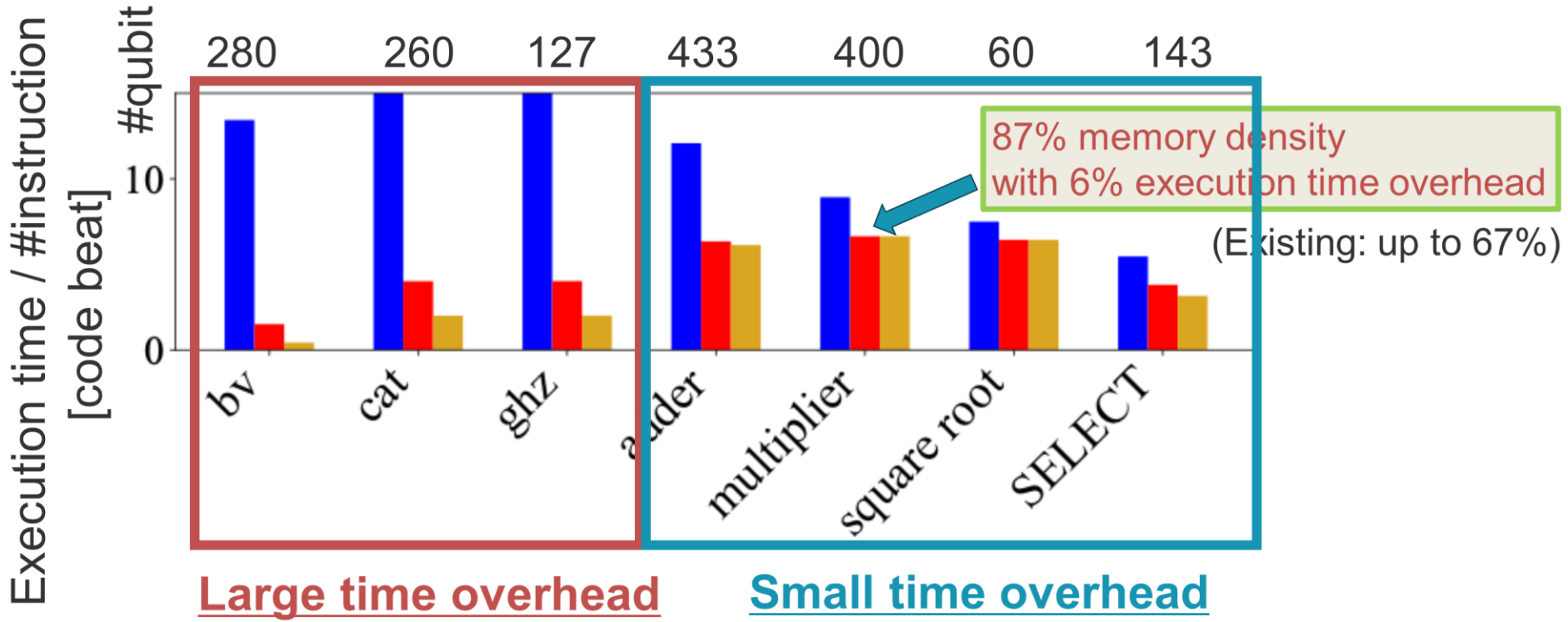
Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

Basic evaluation for benchmark program



Basic evaluation for benchmark program



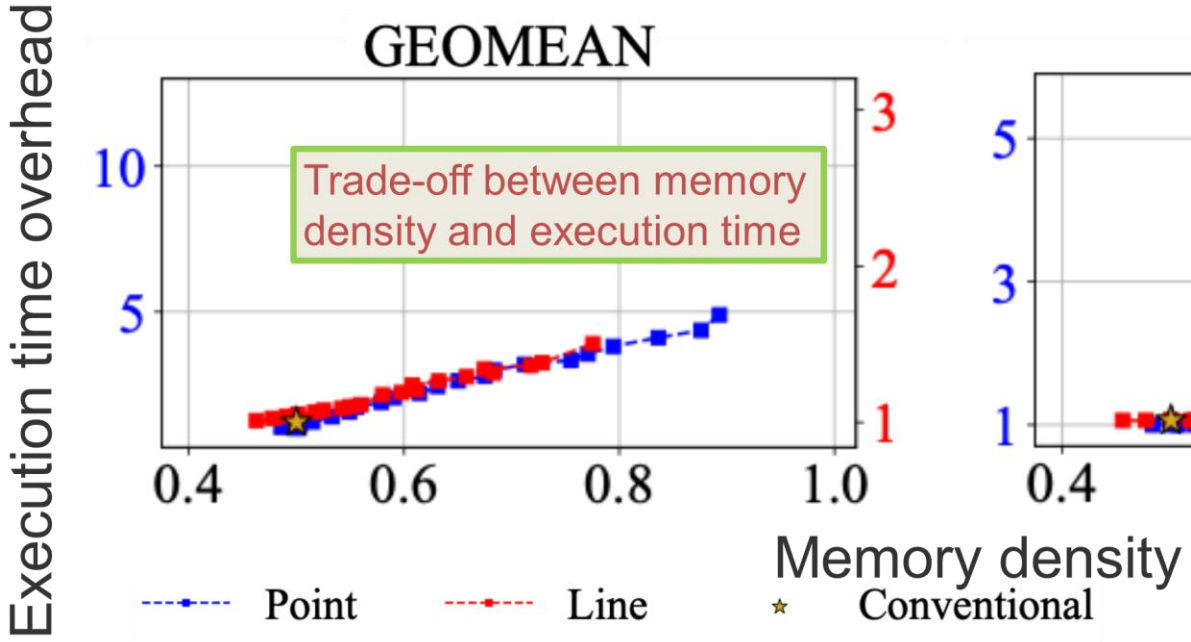
Large time overhead

- Less T-gates
- Not bottleneck in practice

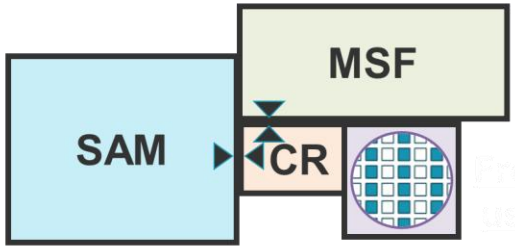
Small time overhead

- Low parallelism and many T-gates
- Bottleneck for many practical applications

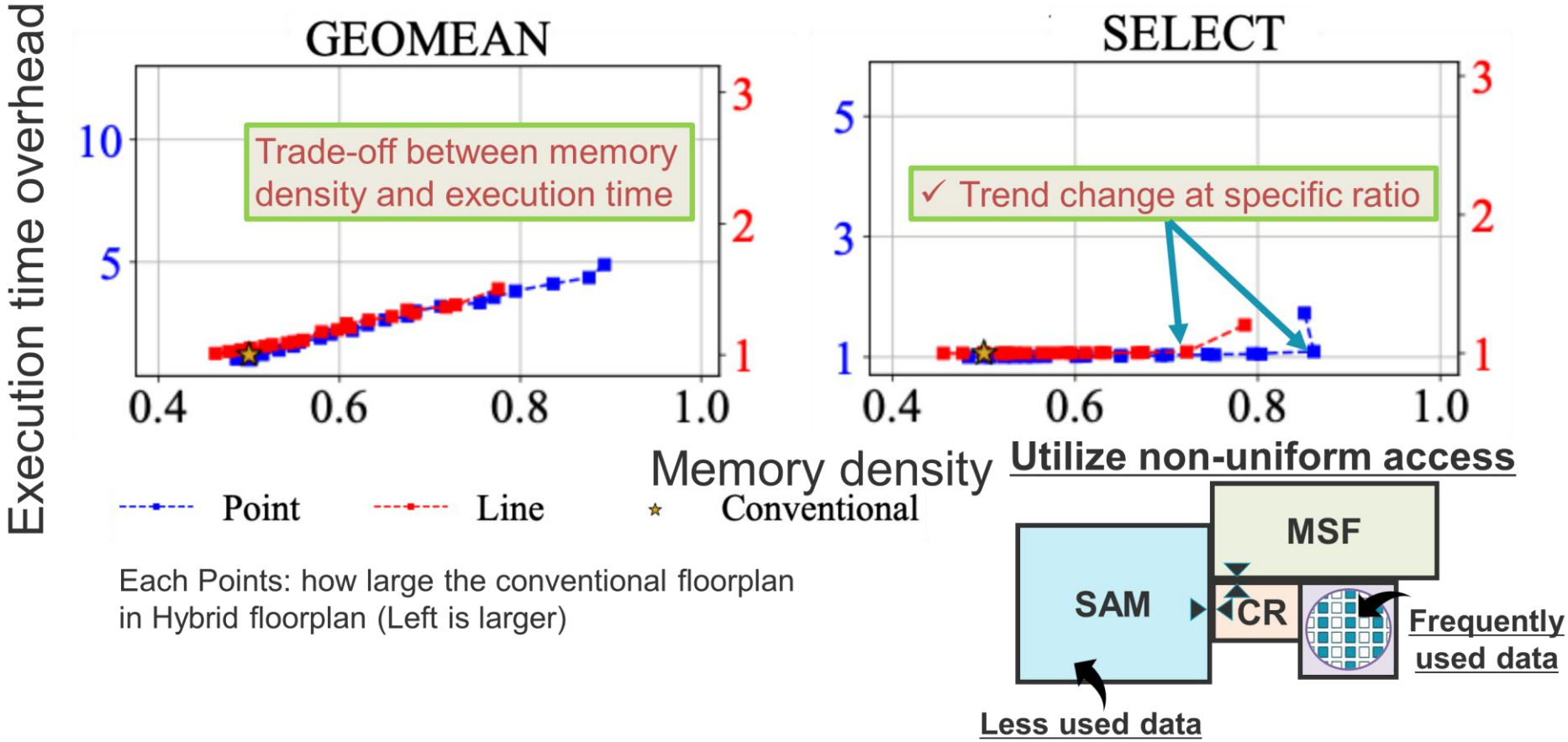
Hybrid floorplan for benchmark program



Each Points: how large the conventional floorplan in Hybrid floorplan (Left is larger)

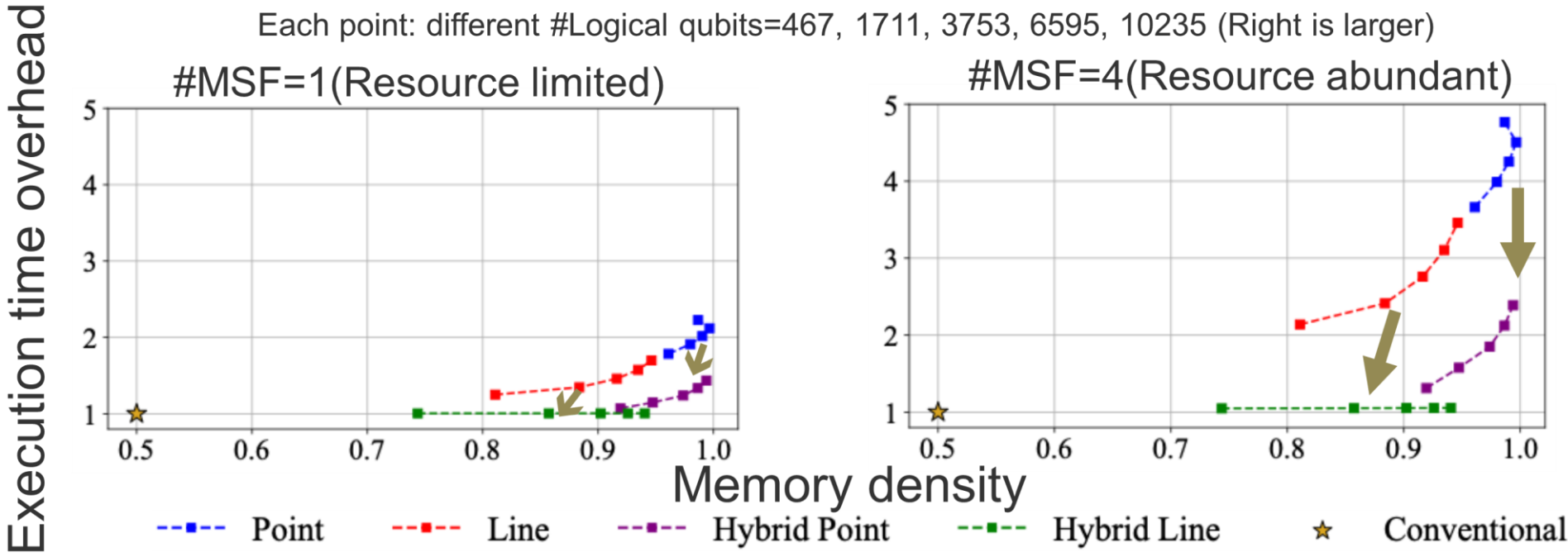


Hybrid floorplan for benchmark program



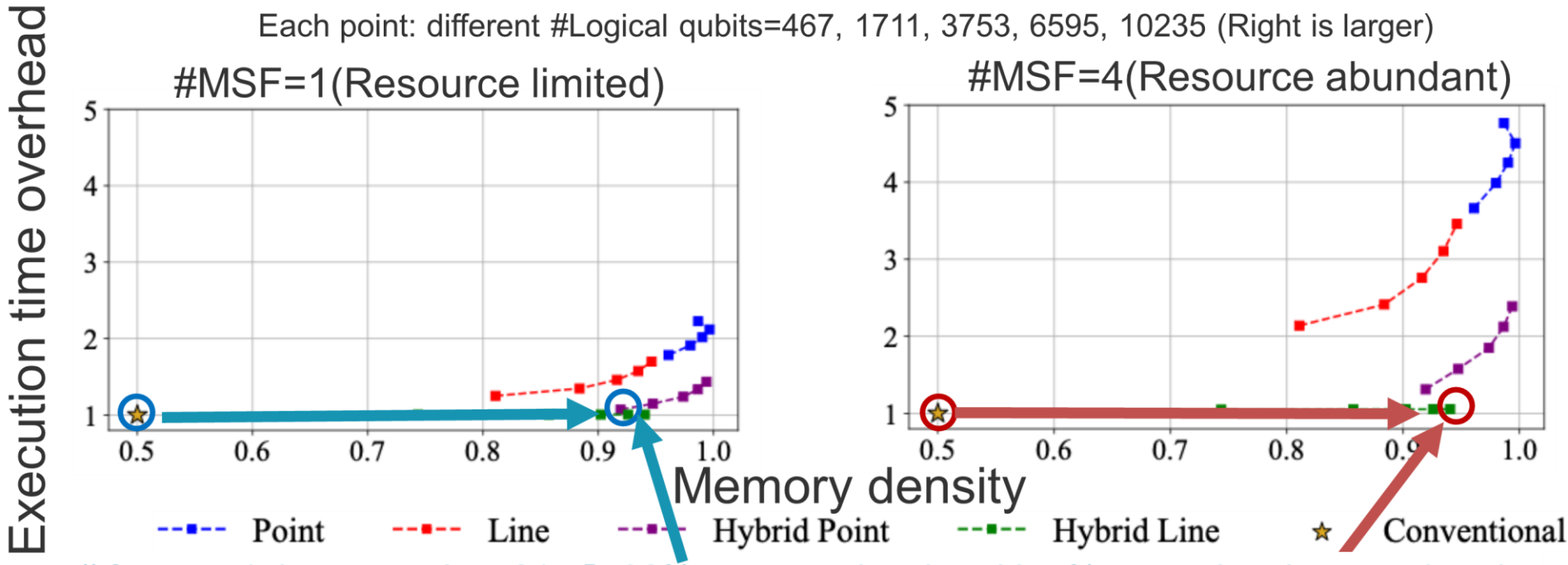
Each Points: how large the conventional floorplan in Hybrid floorplan (Left is larger)

Optimized hybrid floorplan evaluation for SELECT circuit



✓ Hybrid floorplan can reduce execution time by slightly decreasing memory density

Optimized hybrid floorplan evaluation for SELECT circuit



factory=1, instance size=21 ➔ 92% memory density with 7% execution time overhead by Hybrid point-SAM architecture

factory=4, instance size=101 ➔ 94% memory density with 6% execution time overhead by Hybrid line-SAM architecture

Contents

- Background: surface code and lattice surgery-based fault-tolerant quantum computation (FTQC)
- Motivation
  - Space-time tradeoff for lattice surgery-based FTQC
  - Long memory access latency tolerance
- Proposal: Load/Store architecture for FTQC
  - Concept
  - Memory region implementations: Point and Line scan-access memory
  - Optimization techniques inspired by classical computing
- Numerical simulation
- Summary

Summary

- We proposed Load/Store Quantum Computer Architecture (LSQCA)
  - ✓ Achieve almost 100% memory density and break existing density limit
  - ✓ Load/Store instruction leverage access locality
  - ✓ Its additional overhead hidden by magic state bottleneck
- Numerical simulations show the effectiveness of LSQCA
  - ✓ In the resource-limited scenario, about 90% memory density with a few additional time overhead for practical applications
  - ✓ In the resource-abundant scenario, utilizing hybrid floorplan LSQCA can also reduce memory overhead with a little time overhead