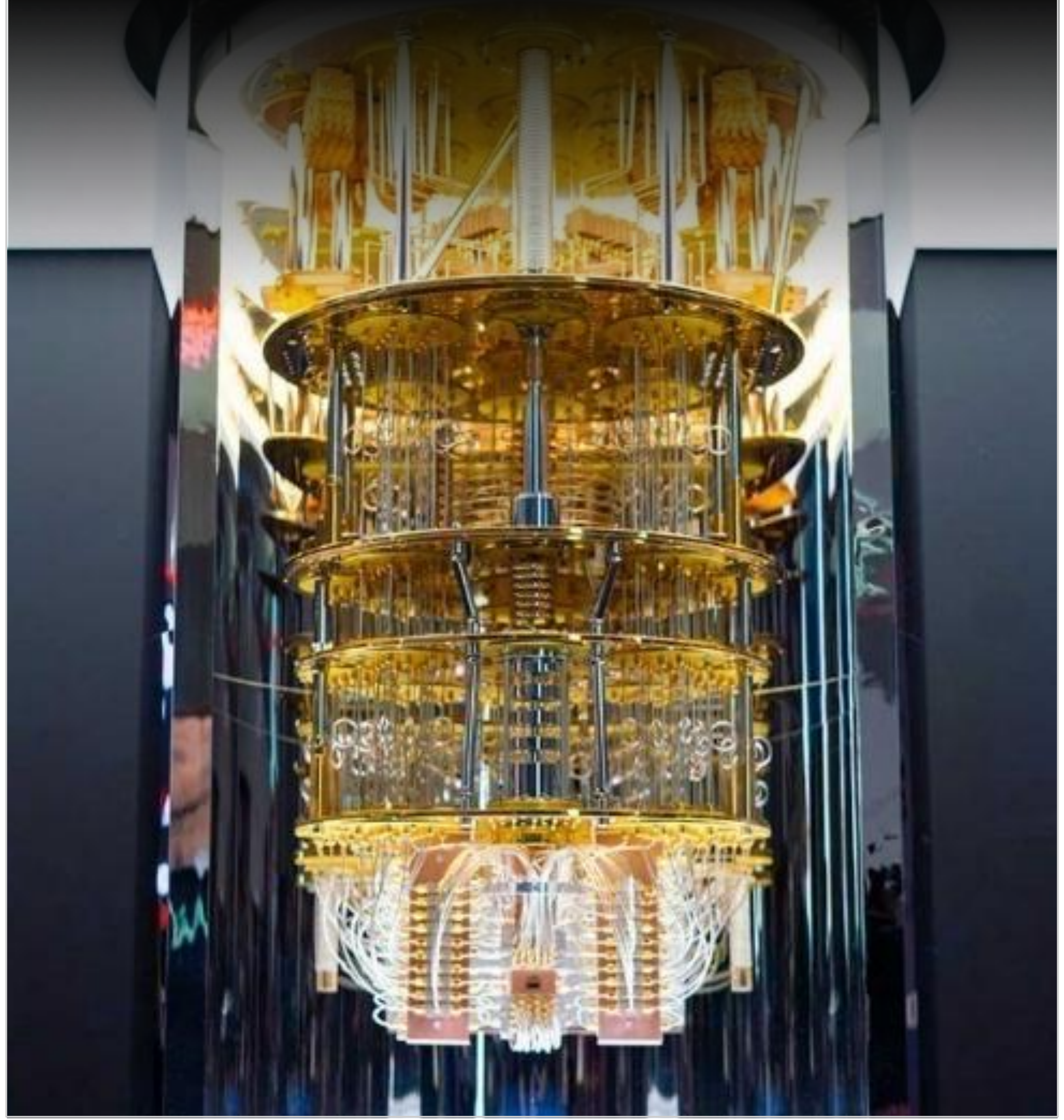
cuQuantum SDK: A High-Performance Library for Accelerating Quantum Circuit Simulation Part I – cuTensorNet

Azzam Haidar Senior Math & Quantum Libraries Engineer & cuQuantum SDK Team

QC-DCEP'23, Oct 24-25, 2023

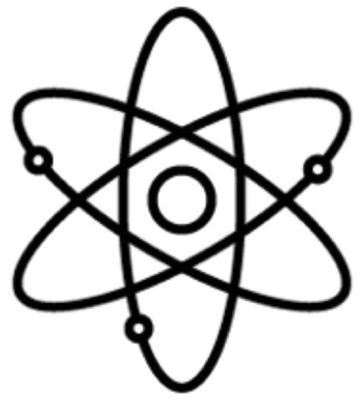
A NEW COMPUTING MODEL — QUANTUM

NEW COMPUTING MODEL



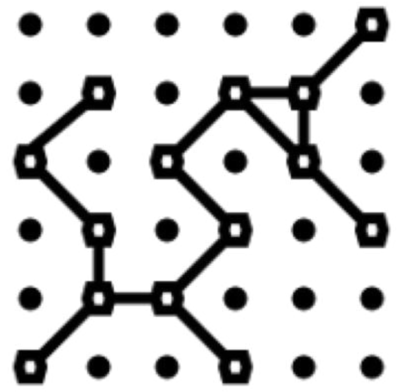
POTENTIAL USE CASES

Computational Finance



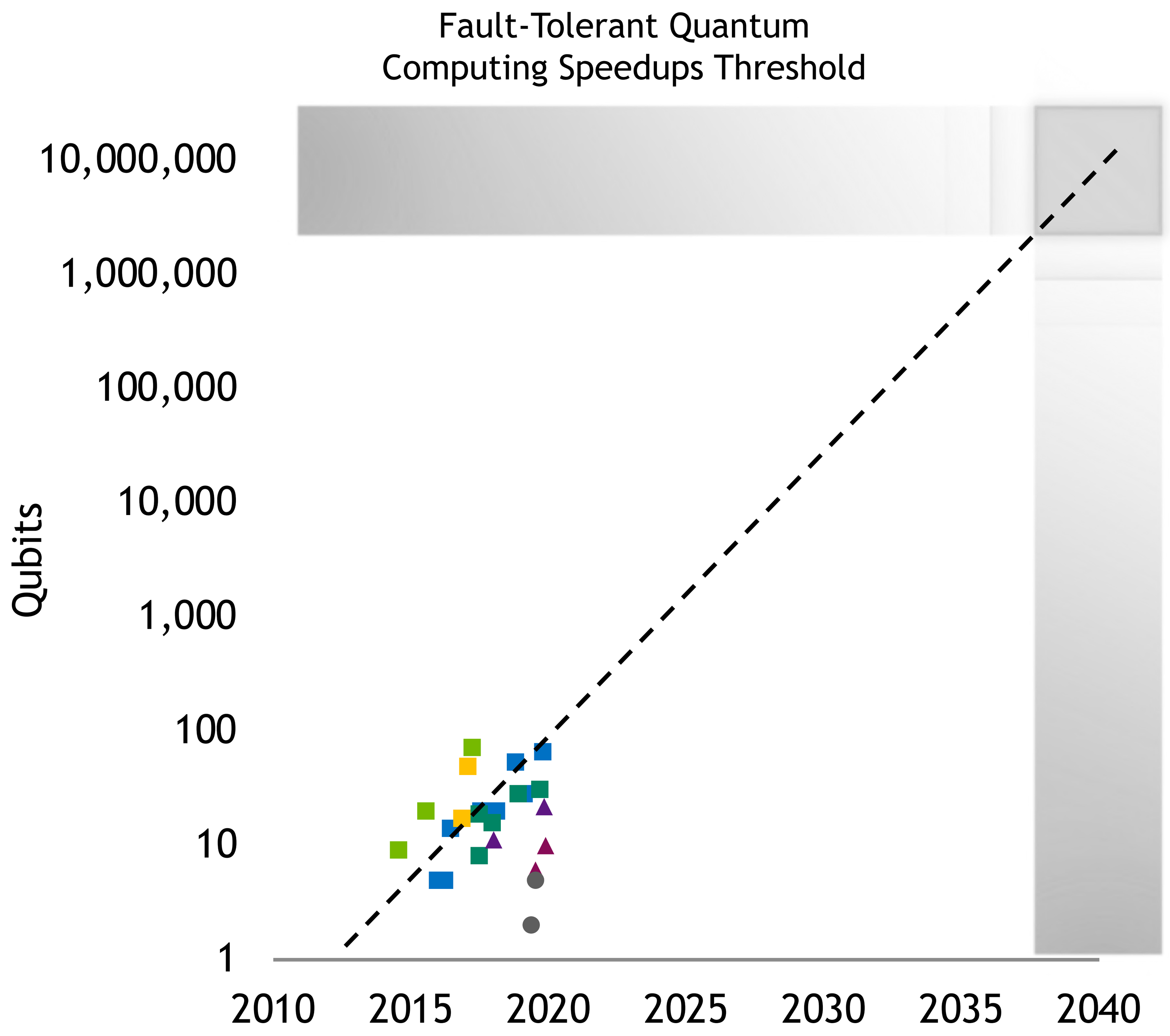
Quantum Chemistry and Physics

Cryptography



Combinatorial Optimization

REQUIRES QUBITS SCALE TO DOUBLE EVERY YEAR

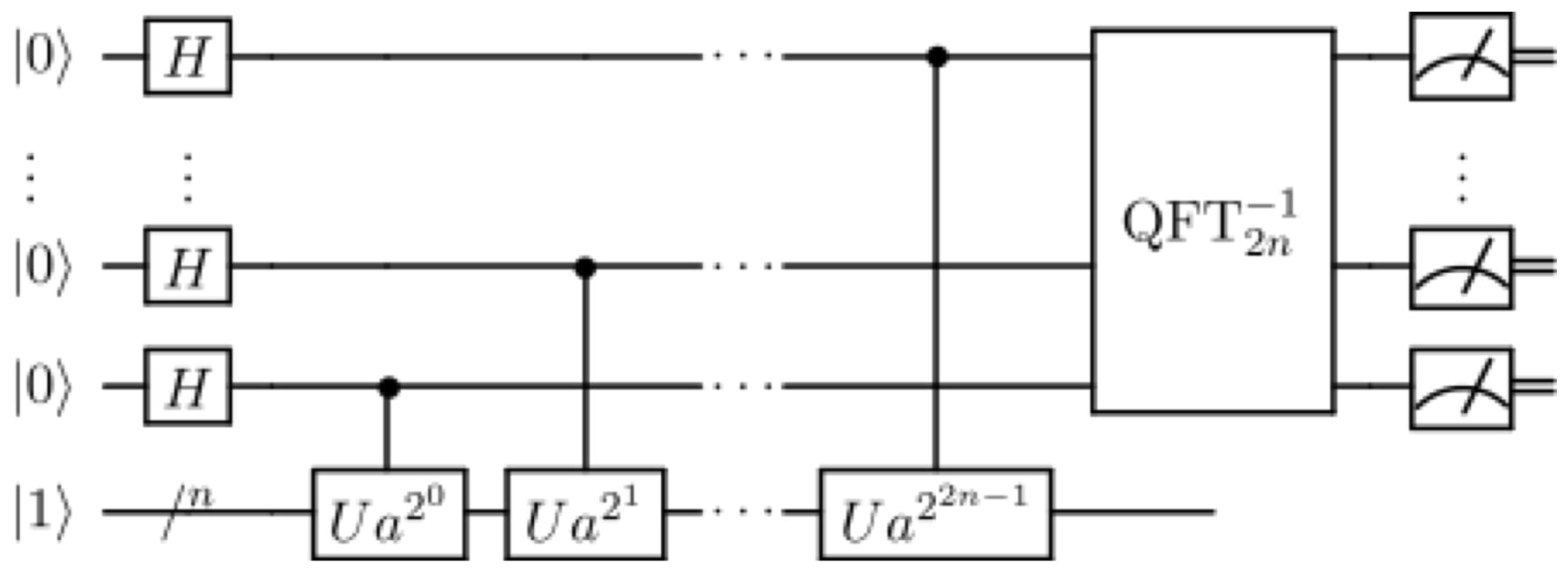


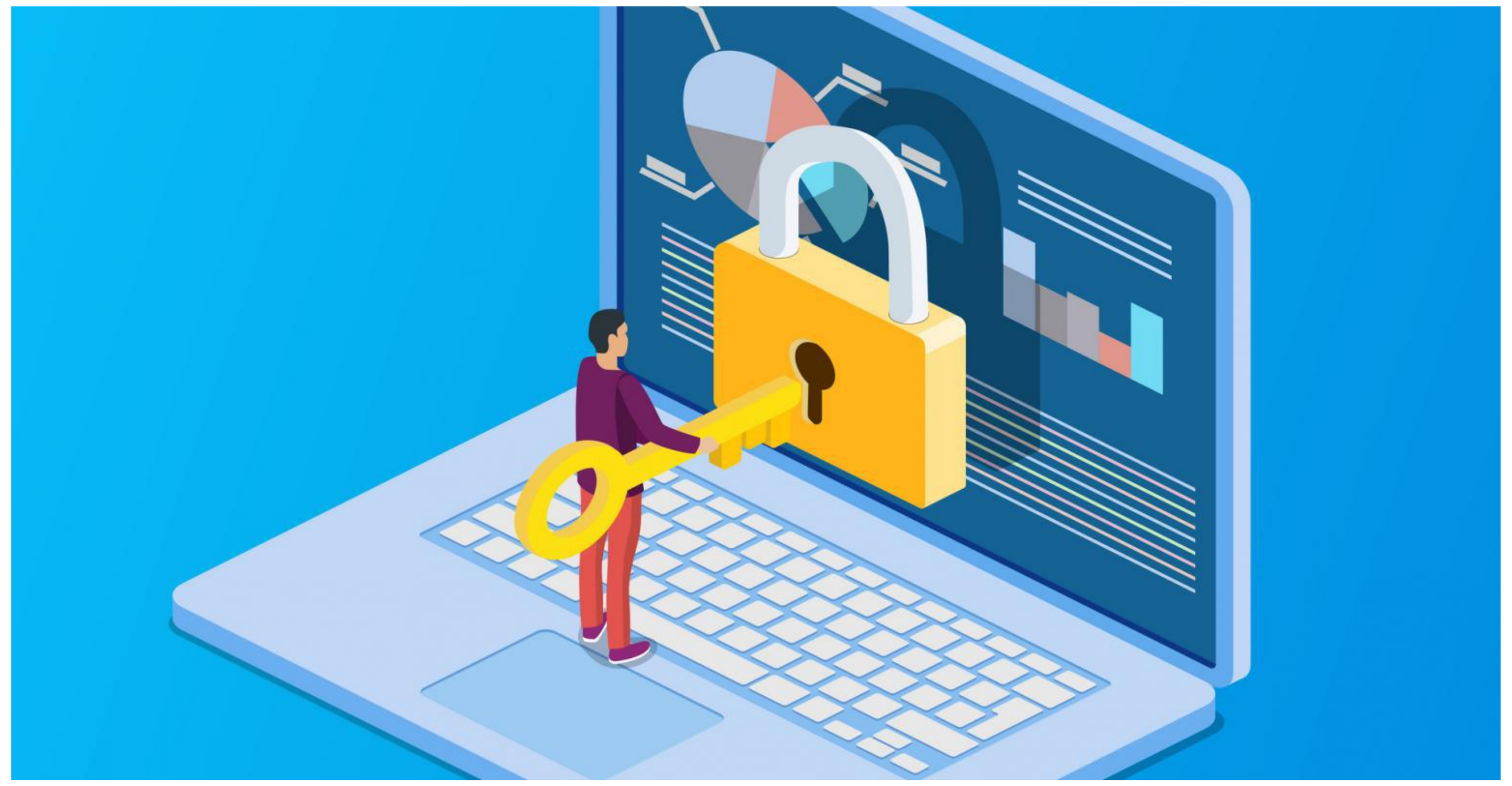
QUANTUM COMPUTATIONAL ADVANTAGE

Rigorous proofs of advantage, many “perfect” qubits required

SHOR’S ALGORITHM

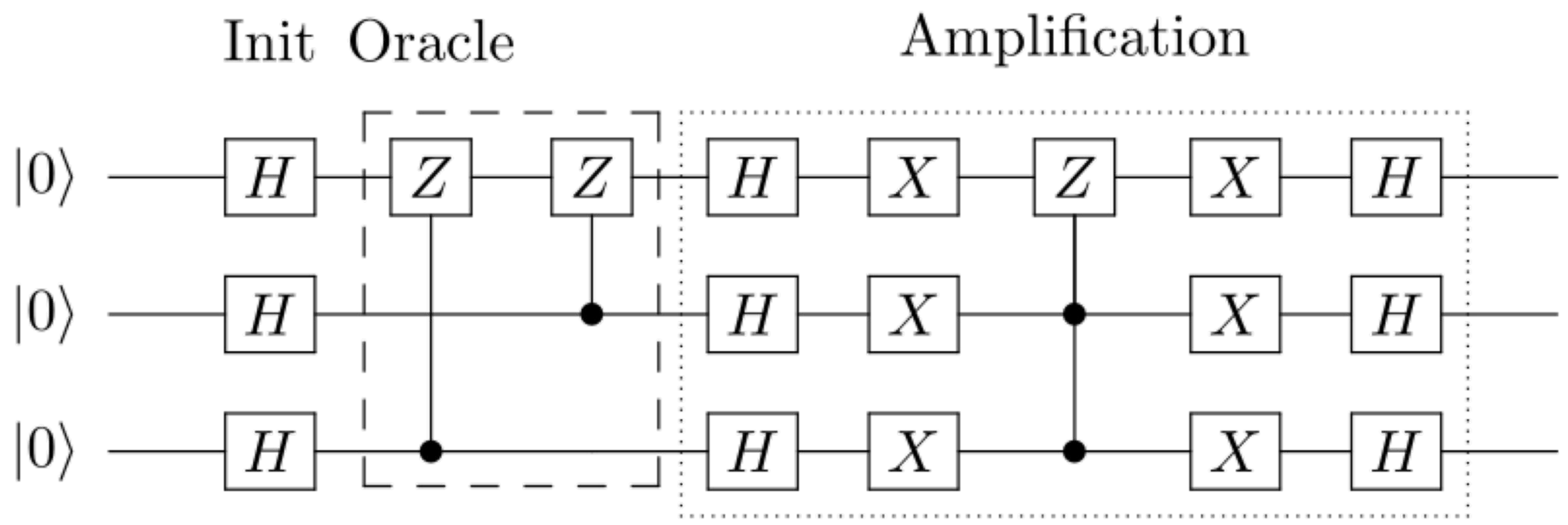
- Prime factorization of numbers - encryption
- Exponential speed-up

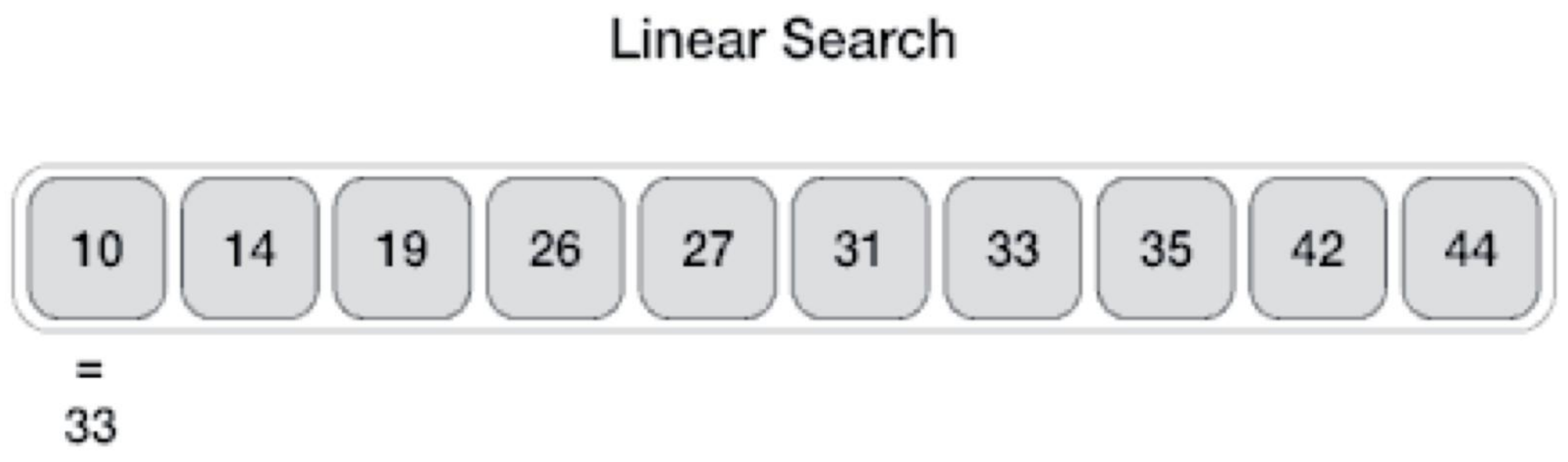




GROVER’S ALGORITHM

- Unstructured search
- Quadratic speed-up





However, the main QIS research efforts are focused on achieving quantum advantage for practical problems on near-term quantum devices without theoretical guarantees:

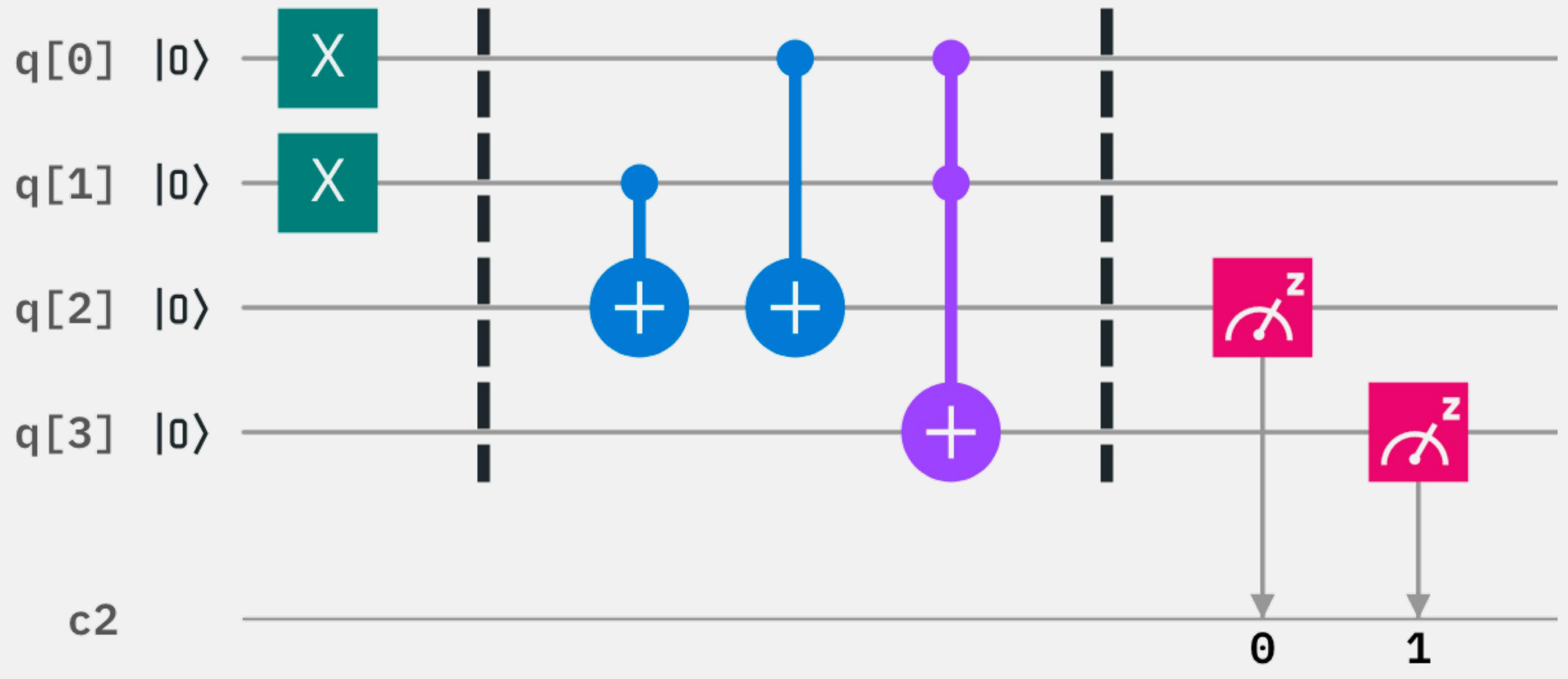
Scalable efficient classical simulators of quantum devices are vital here!

GPU SUPERCOMPUTING IN THE QUANTUM COMPUTING ECOSYSTEM

Researching the Quantum Computers of Tomorrow with the Supercomputers of Today

QUANTUM CIRCUIT SIMULATION

Critical tool for answering today's most pressing questions in Quantum Information Science (QIS):



- What quantum algorithms are most promising for near-term or long-term quantum advantage?
- What are the requirements (number of qubits and error rates) to realize quantum advantage?
- What quantum processor architectures are best suited to realize valuable quantum applications?

HYBRID CLASSICAL/QUANTUM APPLICATIONS

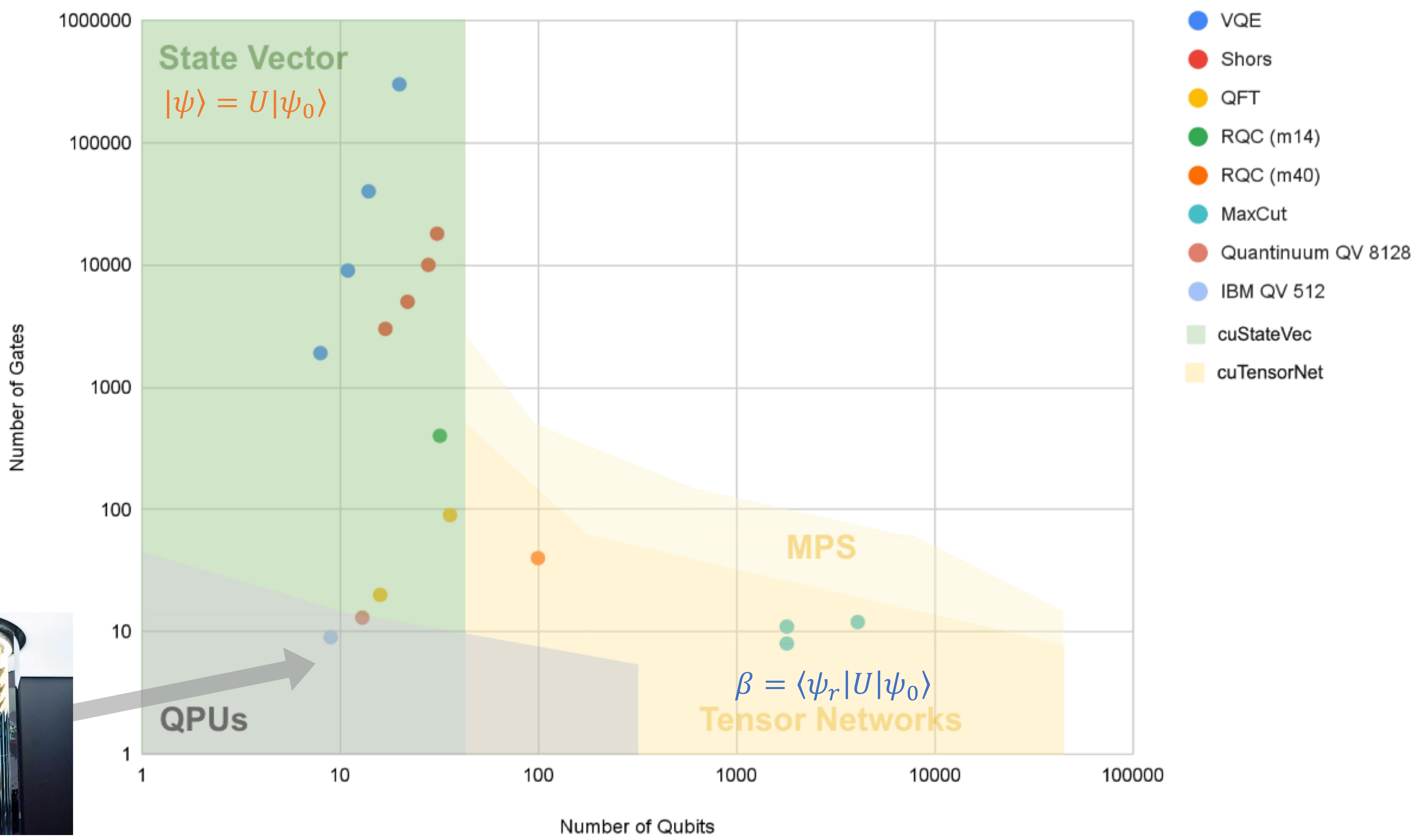
Impactful QC applications (e.g., simulating quantum materials and systems) will require classical supercomputers with quantum co-processors

+

- How can we integrate and take advantage of classical HPC to accelerate hybrid classical/quantum workloads?
- How can we allow domain scientists to easily test co-programming of QPUs with classical HPC systems?
- Can we take advantage of GPU acceleration for circuit synthesis, classical optimization, and error correction decoding?

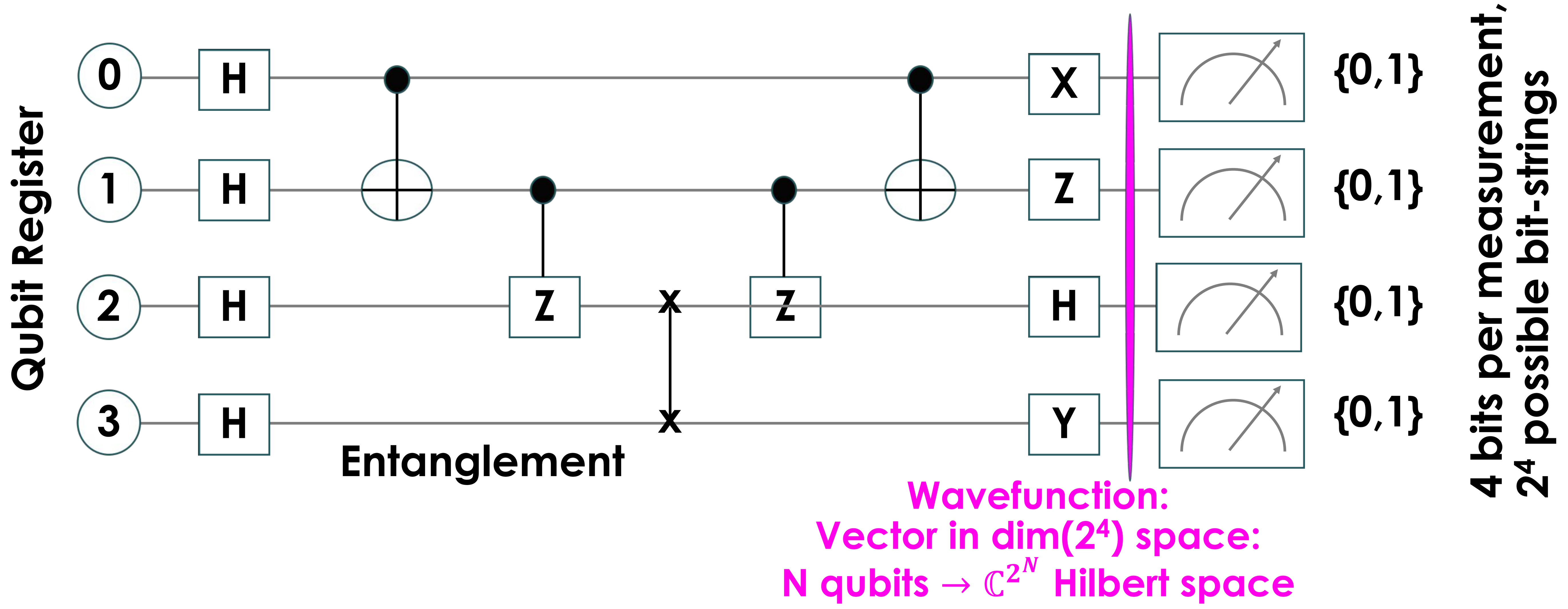
ENABLING LARGE-SCALE QUANTUM CIRCUIT SIMULATIONS VIA CUQUANTUM

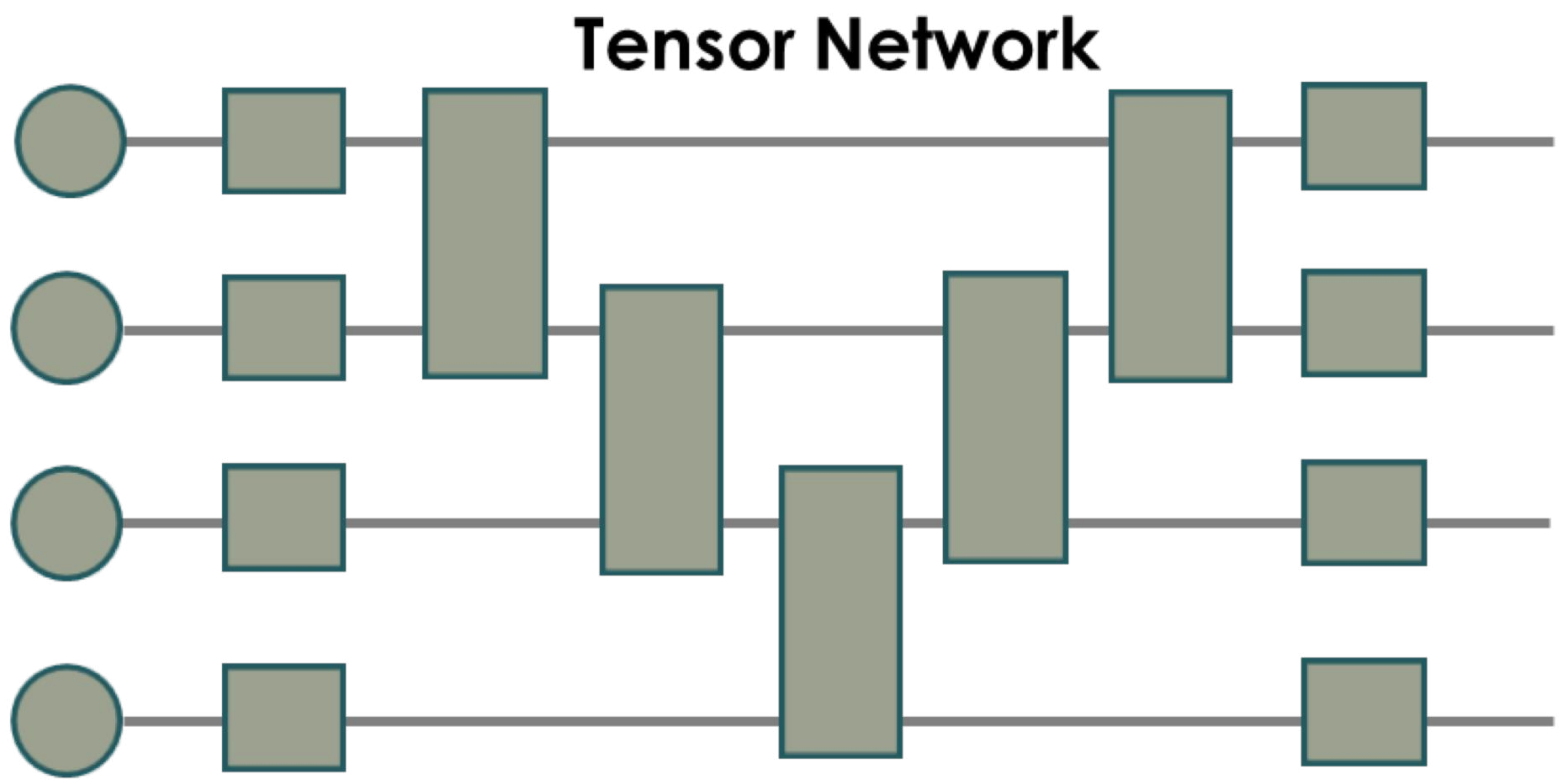
Researching & Developing the Quantum Computers of Tomorrow Requires Powerful Simulations Today



TWO LEADING QUANTUM CIRCUIT SIMULATION APPROACHES

GPUs are a great fit for either approach





State vector simulation

"Gate-based emulation of the full quantum state"

- Maintain full 2^n qubit vector state in memory
- Update the full quantum state every timestep, probabilistically sample or perform direct evaluation of expectation values
- Memory capacity and time grow exponentially with the number of qubits - practical limit around 50 qubits on a supercomputer
- Can model either ideal or noisy qubits

Tensor network simulation

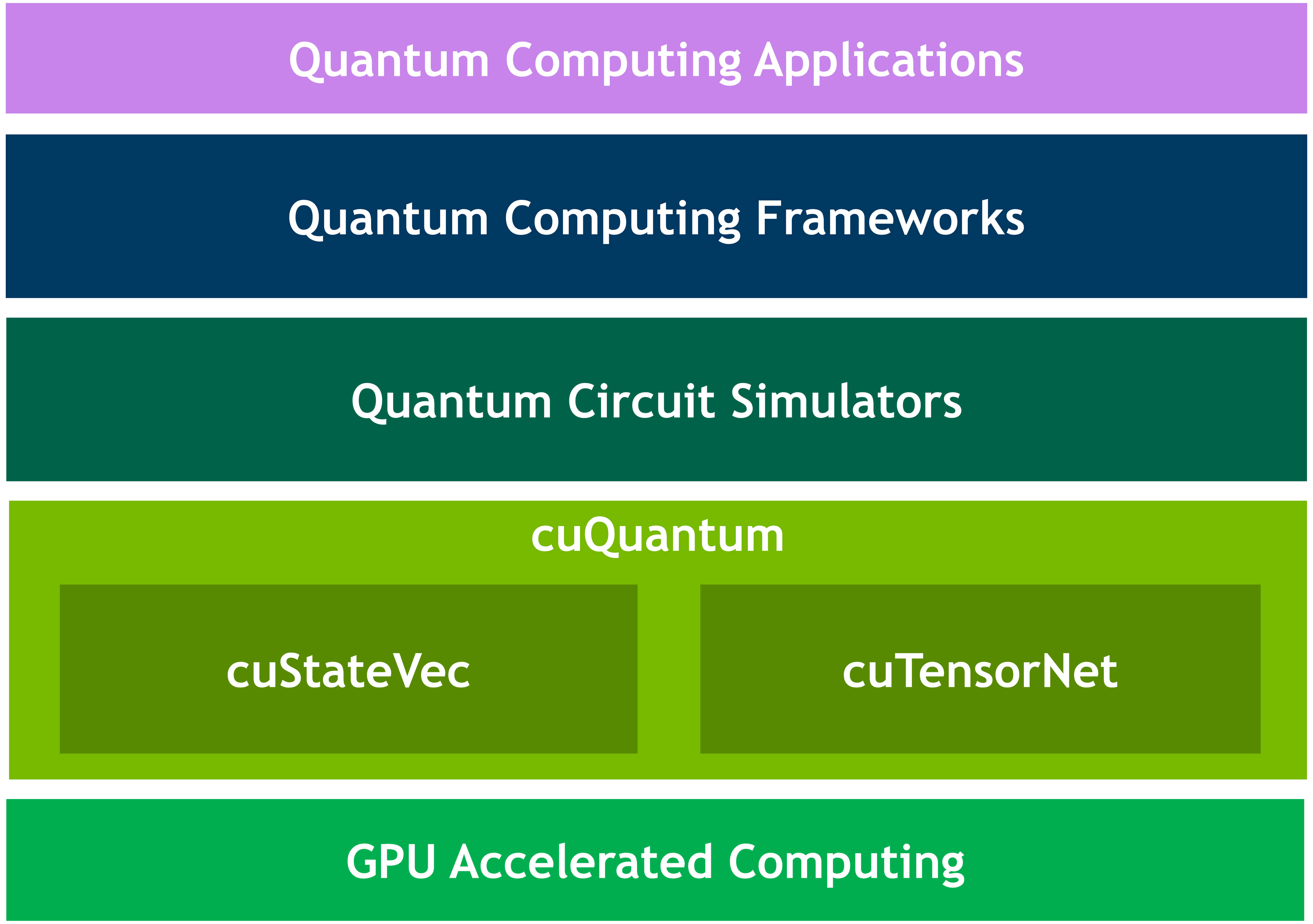
"Factorized tensor representations of observables"

- Performs a time-optimal sequence of tensor network contractions to dramatically reduce memory demands in simulating quantum circuits
- Can simulate 100s or 1000s of qubits for practically interesting quantum circuits
- Enables approximate compression of the original tensor network by simpler tensor networks resulting in a drastically reduced computational cost

cuQuantum SDK: Accelerating Quantum Simulation Ecosystem

High-performance GPU libraries for quantum circuit simulations

- Collection of **high-performance GPU libraries** for quantum computing researchers and engineers to accelerate and scale up their quantum simulators using NVIDIA GPU platforms
- Current focus is on offering **building blocks** for developing efficient and scalable **quantum circuit simulators** needed by the QIS community
  - Low-level primitives with high degree of control for advanced simulator developers interested in maximal performance
  - High-level primitives with high degree of automation and ease of use
- **cuStateVec**:
  - Targeting state-vector based simulator developers
- **cuTensorNet**:
  - Targeting tensor-network-based simulator developers (exact or approximate)
- **cuQuantum Appliance**:
  - NGC docker container for easy deployment
  - Offering optimized multi-GPU cuStateVec backend for Cirq/Qsim simulator and multi-GPU/multi-node cuStateVec backend for IBM's Qiskit/Aer simulator
- **cuQuantum Python**:
  - Allows easy integration with Python applications and frameworks
  - Provides low-level Python bindings for both cuStateVec and cuTensorNet C API with flexible calling conventions
  - Provides high-level Pythonic API
  - Interoperable with NumPy/CuPy/PyTorch CPU & GPU tensors
  - Open-sourced on GitHub (NVIDIA/cuQuantum) & pip-/conda- installable



https://developer.nvidia.com/cuquantum-sdk

cuStateVec Module of cuQuantum

A library for building efficient state-vector based quantum circuit simulators, not a simulator per se

- **cuStateVec**: Library for accelerating and scaling **state-vector** based quantum circuit simulators:
  - Most computations are “in-place” to **reduce memory demands**
  - Provides **building blocks** to cover common use cases:
    1) Apply gate matrix
       Dense/diagonal/generalized permutation matrices
    2) Apply exponential of a Pauli matrix product
    3) Expectation value
       Matrix or Pauli operator as an observable
    4) Measurement
       Batched Z-basis measurements, Z-product basis measurement
    5) Sampling the state-vector
    6) Support of batched state-vectors (multiple state-vectors)
    7) State vector segment insertion/extraction
    8) In-place qubit reordering for multi-GPU and multi-node simulators
  - **Easy integration & adoption** for a wide variety of frameworks and programming languages
  - Also available in the cuQuantum Ap

C API

```
custatevecStatus_t custatevecApplyMatrix(
    custatevecHandle_t handle,
    void *sv,
    cudaDataType_t svDataType,
    const uint32_t nIndexBits,
    const void *matrix,
    cudaDataType_t matrixDataType,
    custatevecMatrixLayout_t layout,
    const int32_t adjoint,
    const int32_t *targets,
    const uint32_t nTargets,
    const int32_t *controls,
    const int32_t *controlBitValues,
    const uint32_t nControls,
    custatevecComputeType_t computeType,
    void *extraWorkspace,
    size_t extraWorkspaceSizeInBytes)
```

Python API

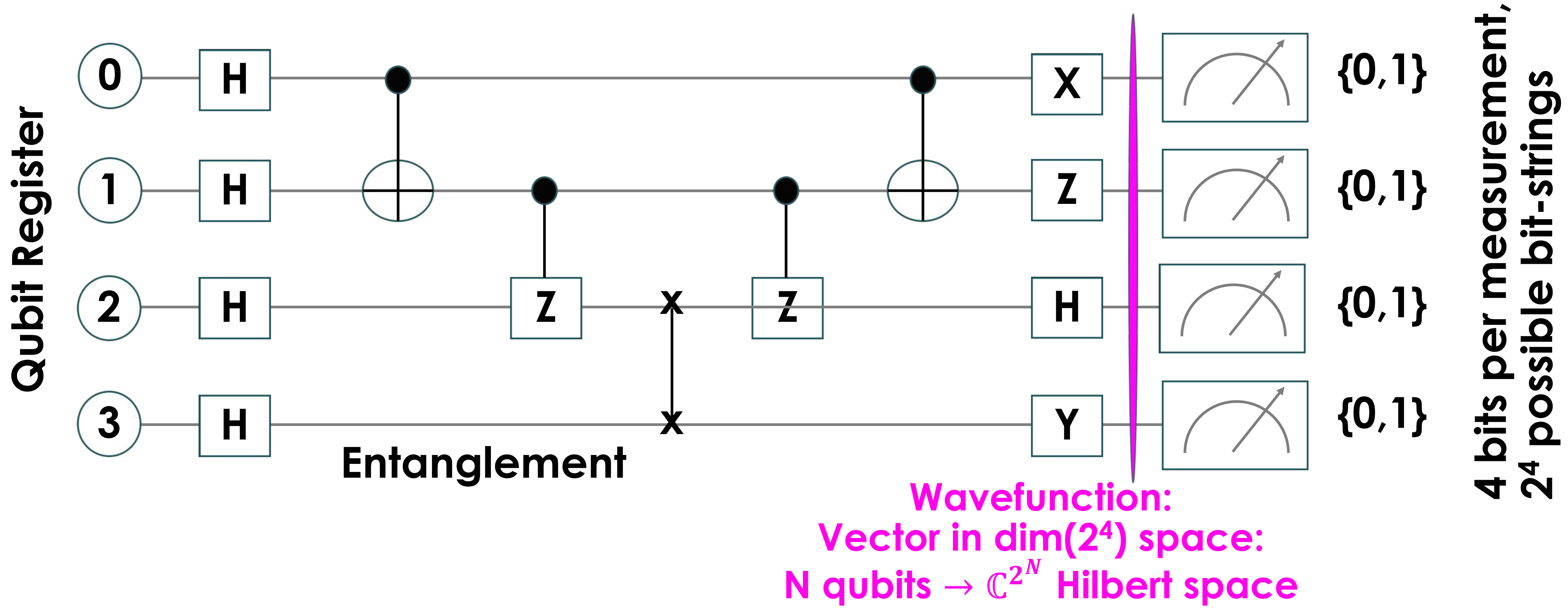
```
cuquantum.custatevec.apply_matrix(
    handle,
    sv,
    sv_data_type,
    n_index_bits,
    matrix,
    matrix_data_type,
    layout,
    adjoint,
    targets,
    n_targets,
    controls,
    control_bit_values,
    n_controls,
    compute_type,
    workspace,
    workspace_size)
```

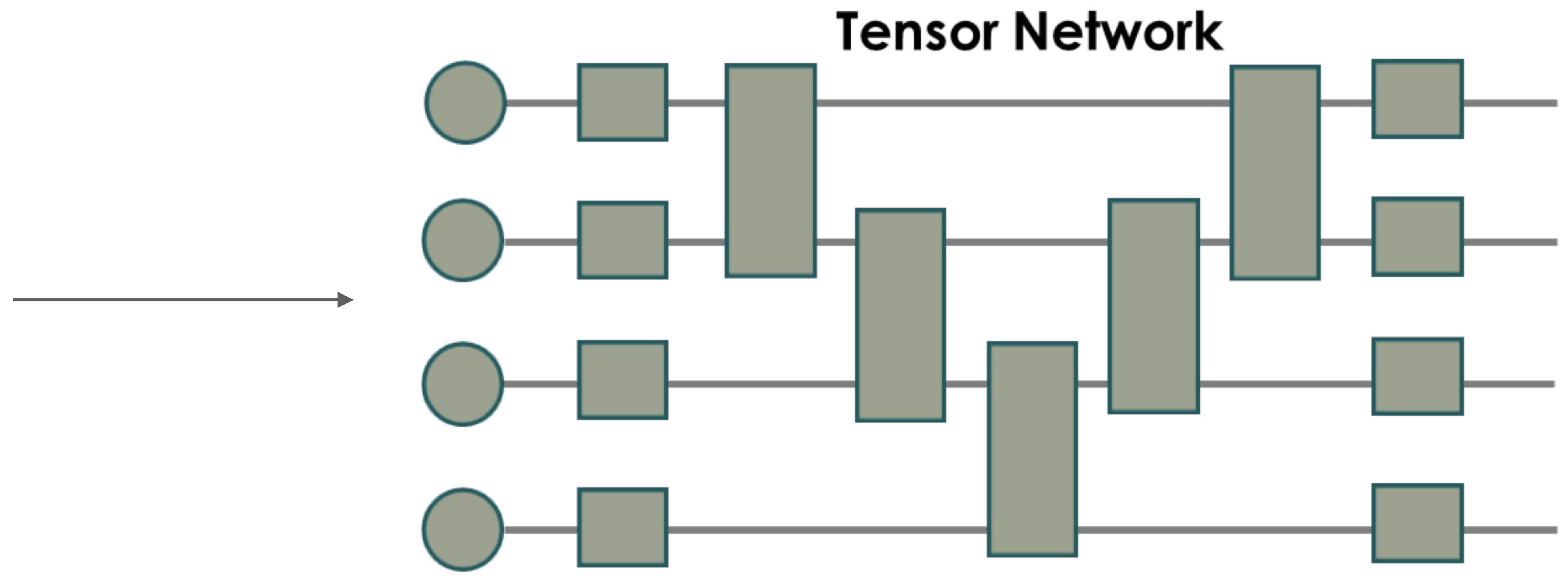
2:05 PM Dr. Shinya Morino, Nvidia Tokyo
NVIDIA cuQuantum SDK: Accelerating Quantum Circuit simulation II - cuStateVec

TENSOR REPRESENTATION OF QUANTUM CIRCUITS

Tensor-algebraic techniques form powerful numerical machinery for quantum many-body methods

Quantum Circuit as a Tensor Network:

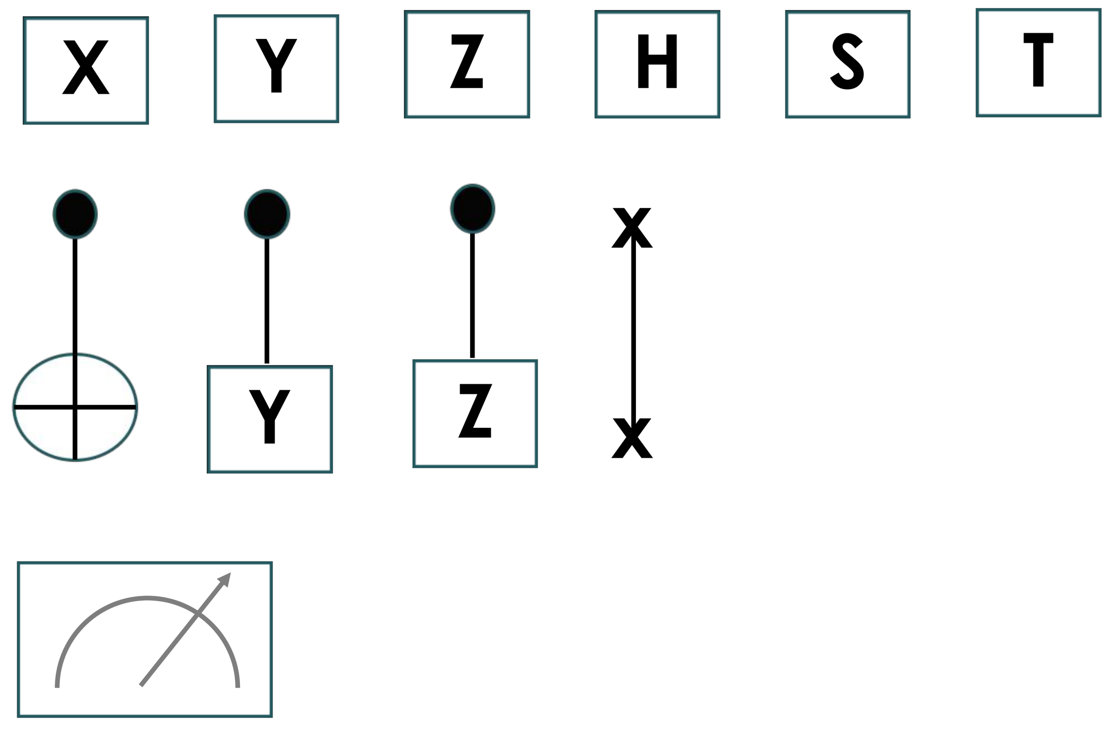




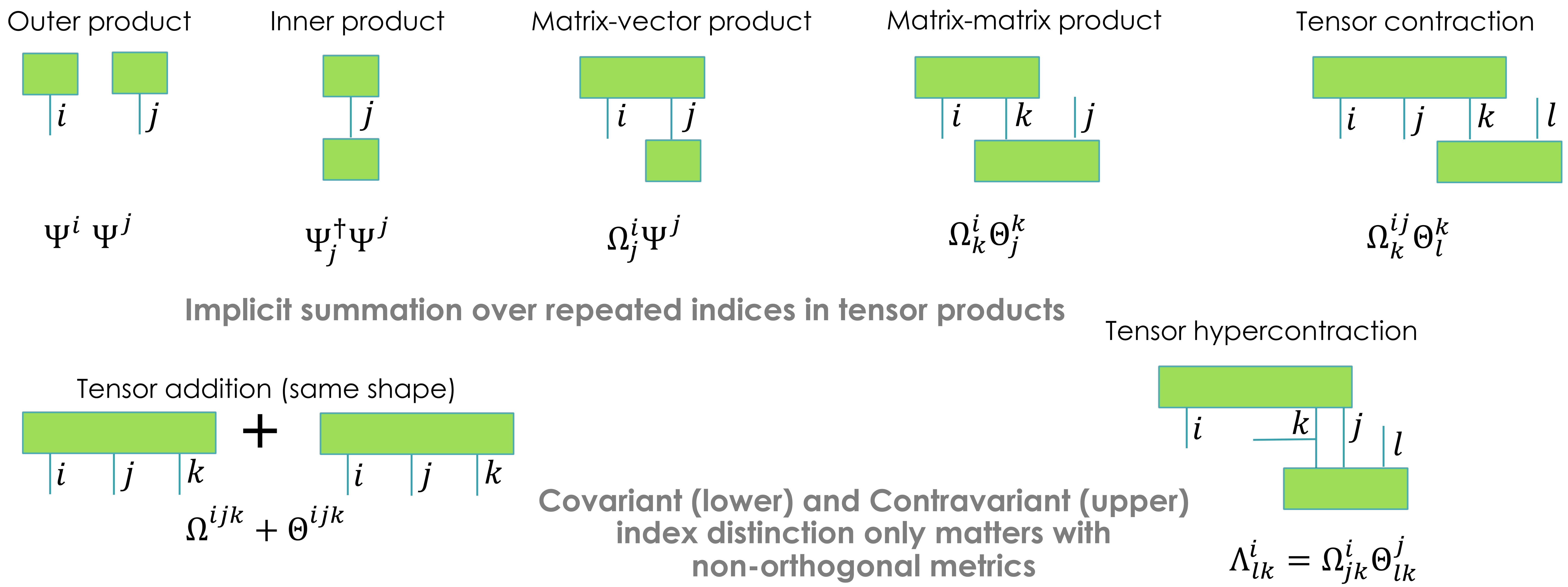
1-qubit gates:

2-qubit gates:

Measurement:



Graphical Tensor Network Algebra Notation



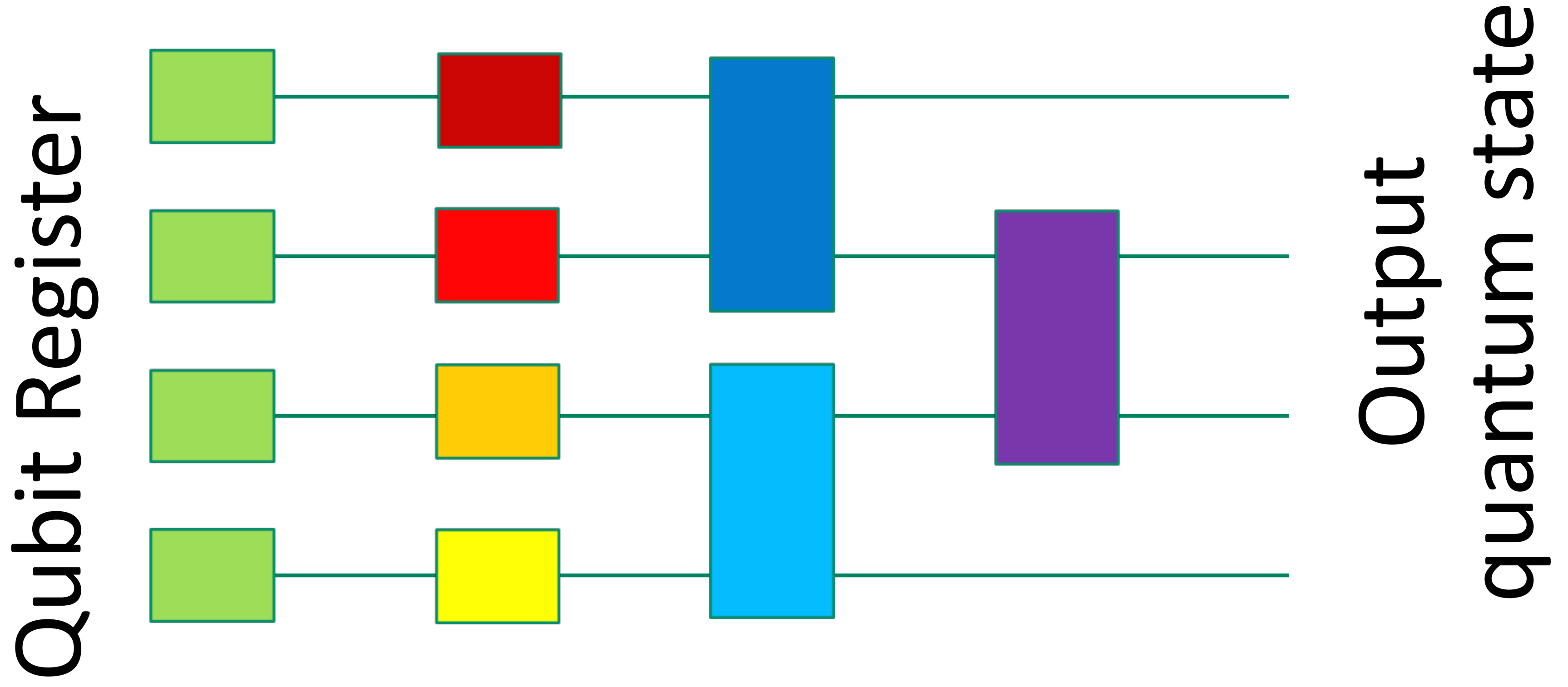
TENSOR REPRESENTATION OF QUANTUM CIRCUITS

Problem of the Optimal Contraction Path:
Obtain a complete list of pairwise tensor contractions with the minimal time to solution (NP-hard optimization problem)

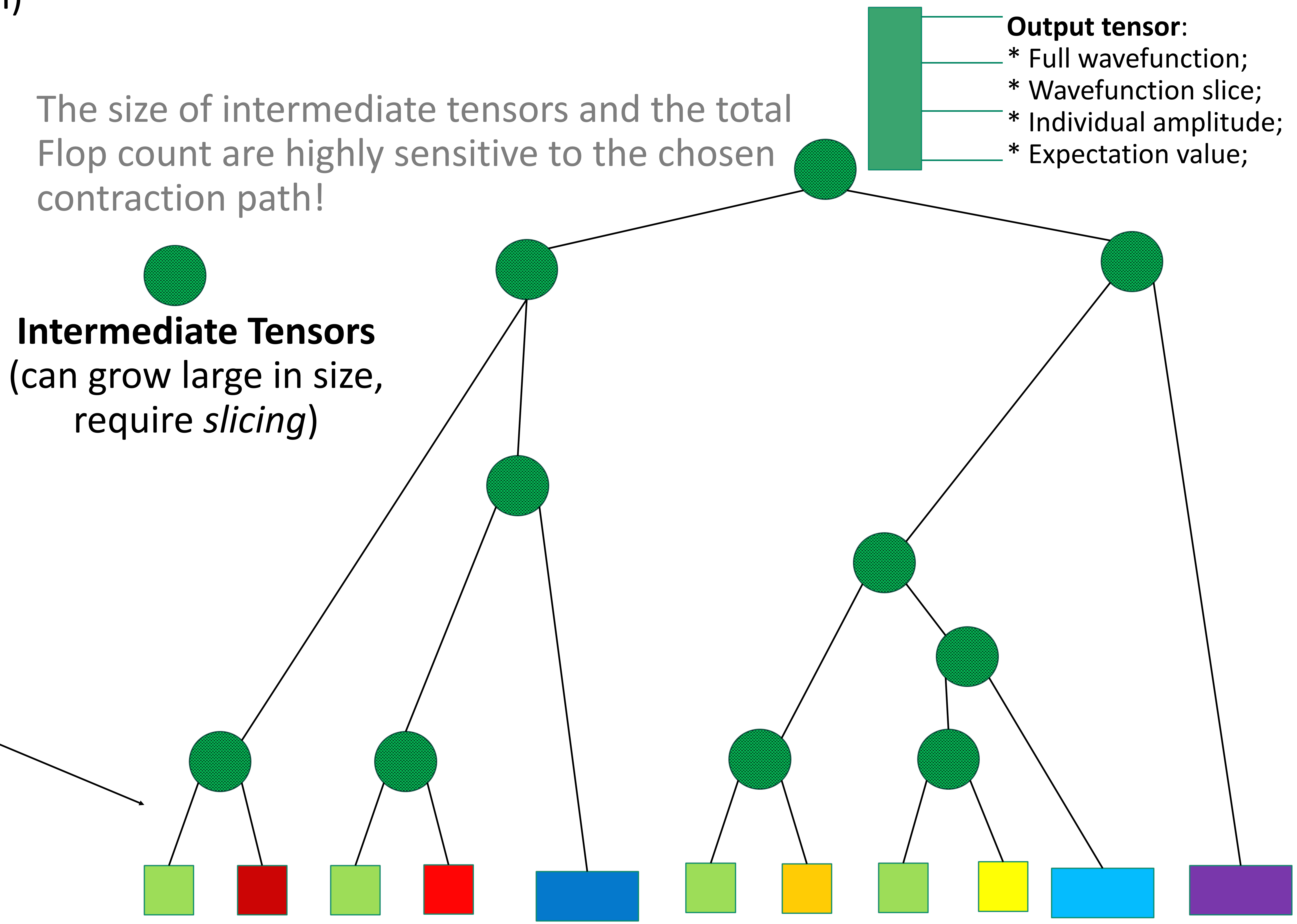
Any slice of the wavefunction tensor is accessible by tensor contraction (Markov, Shi, *SIAM J. Comput.*, https://doi.org/10.1137/050644756)

The size of intermediate tensors and the total Flop count are highly sensitive to the chosen contraction path!

Quantum Gates
(state evolution)



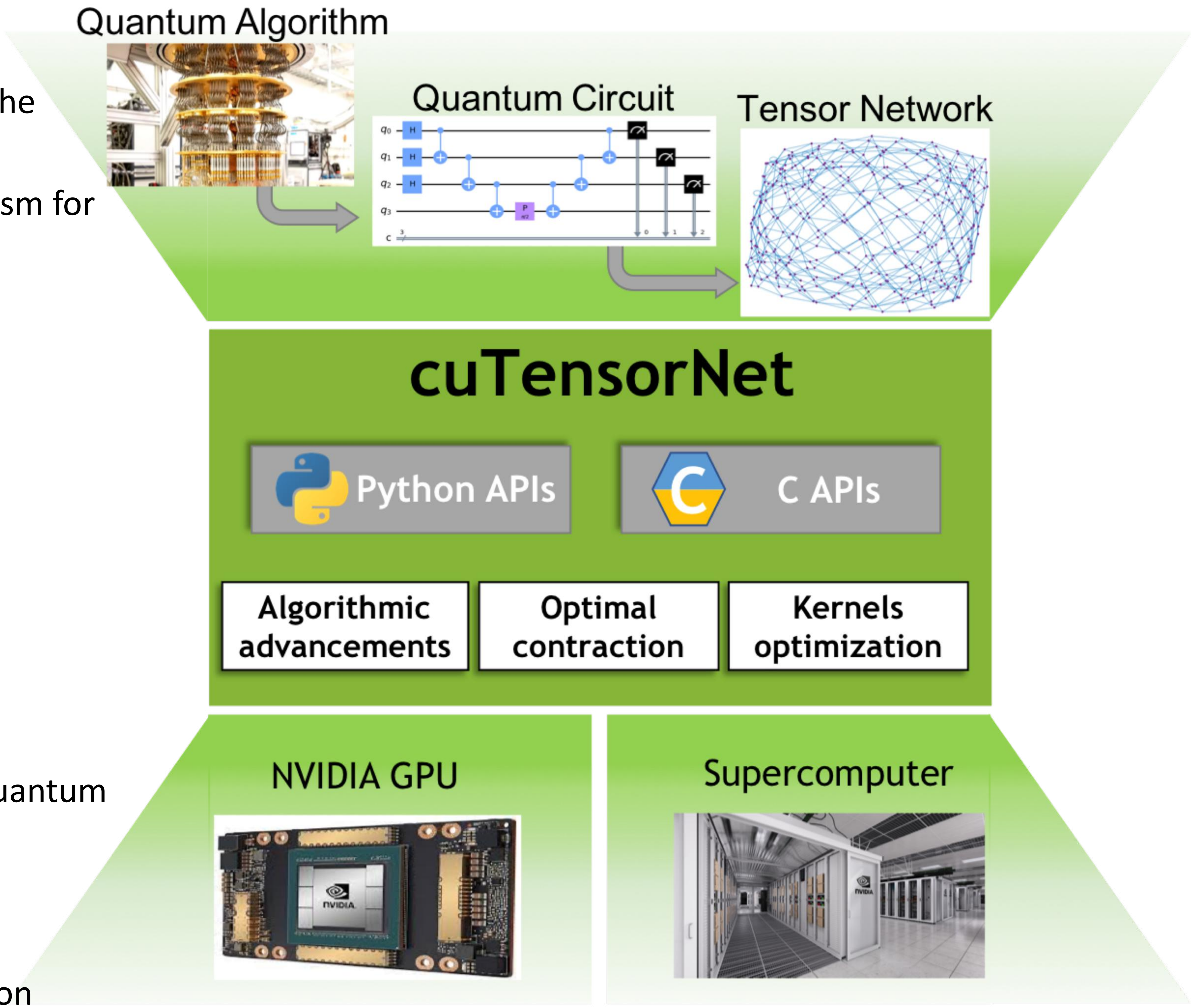
Quantum Circuit as a Tensor Network



cuTensorNet Module of cuQuantum

A library for building efficient tensor-network based quantum circuit simulators, not a simulator per se

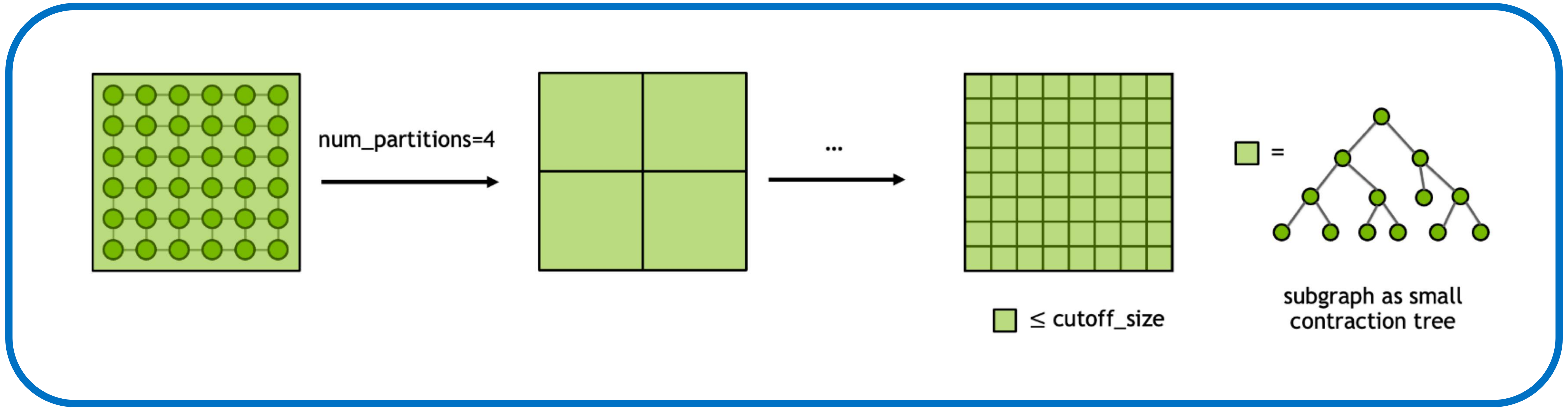
- **cuTensorNet**: Library for accelerating and scaling **tensor-network** based quantum circuit simulators:
- Provides primitives to cover common use cases:
  - Calculate a **cost-optimal path** for tensor network contraction:
    - Recursive hyper-graph partitioning and hyper-optimization are used to find a contraction path with the **lowest total cost** (Flop count or time-to-solution estimate)
    - **Optimized slicing** is introduced to reduce the maximum intermediate tensor size and create parallelism for distributed execution on multi-GPU/multi-node platforms
  - Define the **optimal execution plan** and execute tensor network contraction:
    - Leverages **cuTENSOR** heuristics for selecting the best pairwise tensor contraction kernels
  - **Automatic parallelization** of the contraction path optimization and execution across multiple GPUs and multiple/many nodes
  - Automatic **intermediate tensor caching** and reuse to reduce time taken by repeated tensor network contractions: Computing many amplitudes, direct bit-string sampling, etc.
  - **High-level API** for defining tensor network states and computing their properties:
    - Quantum **gate application**
    - Computing **reduced density matrices** for a tensor network state
    - Direct **bit-string sampling** of a tensor network state (e.g., sampling output bit-strings from a given quantum circuit)
  - Compute **gradients of a tensor network** with respect to selected input tensors (training) via the backpropagation algorithm
  - **Contract/decompose tensor API** for implementing approximate tensor-network-based simulators based on MPS, PEPS, and other tensor network factorizations.
  - **Interoperable** with other libraries/frameworks (e.g., NumPy, cuPy, pyTorch)



cuTensorNet Module of cuQuantum

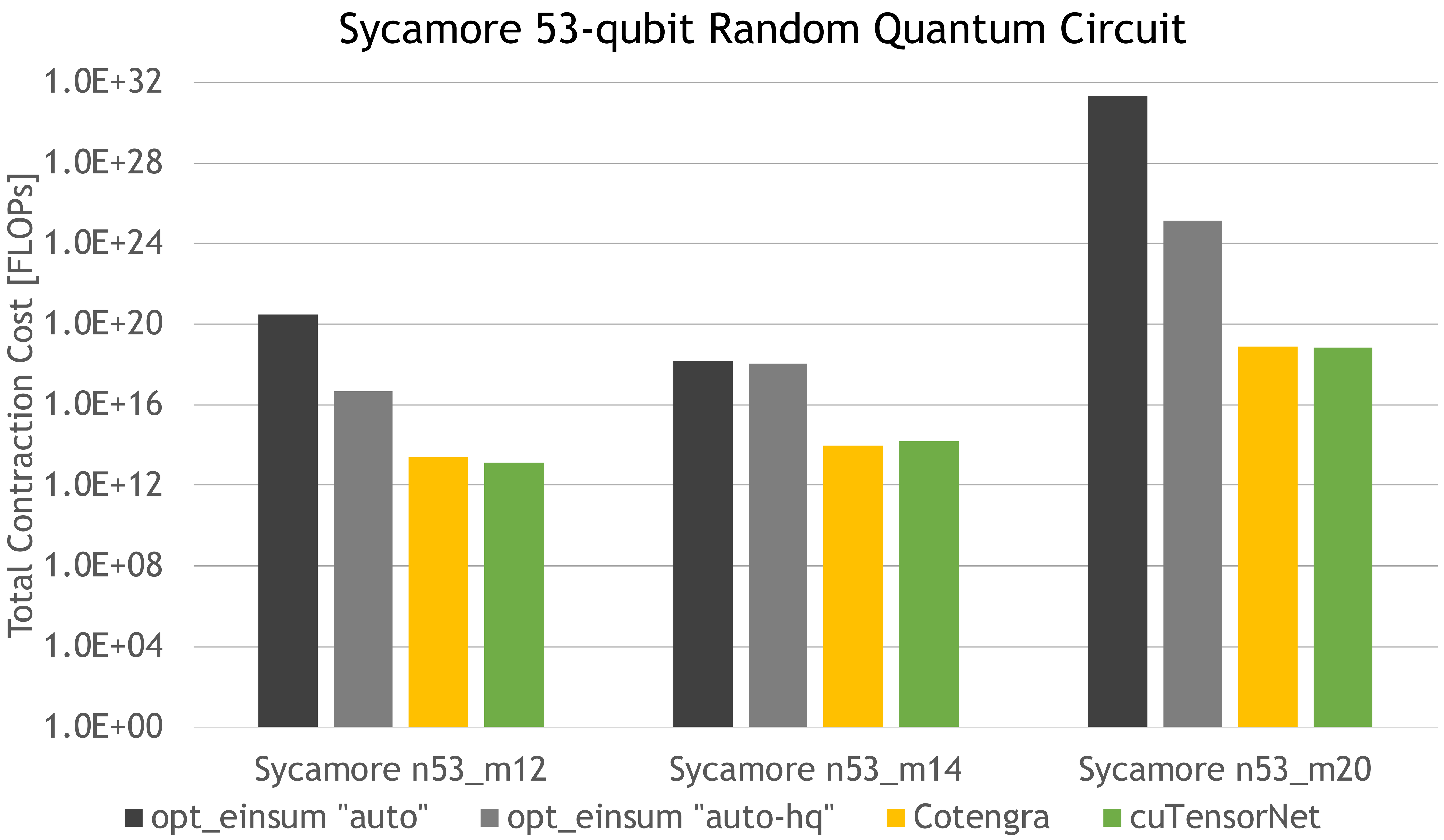
Tensor contraction path optimizer

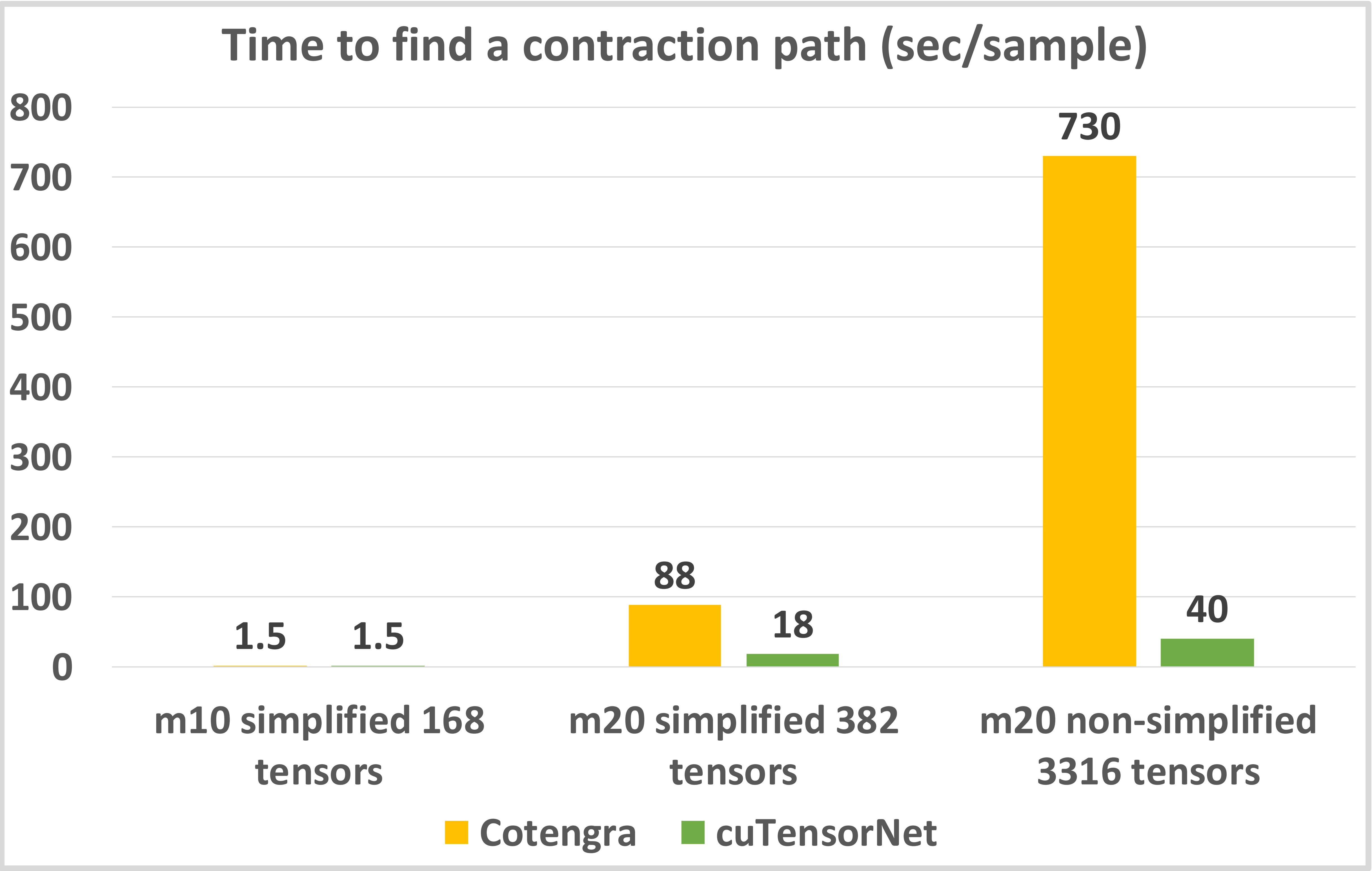
- **Tensor network contraction path optimizer**:
  - Tensor network simplification
  - Hyper-optimizer (global search):
    - Recursive k-way graph partitioner
    - Tensor mode slicing
    - Subtree reconfiguration



cuTensorNet Module of cuQuantum

Tensor contraction path optimizer: Performance



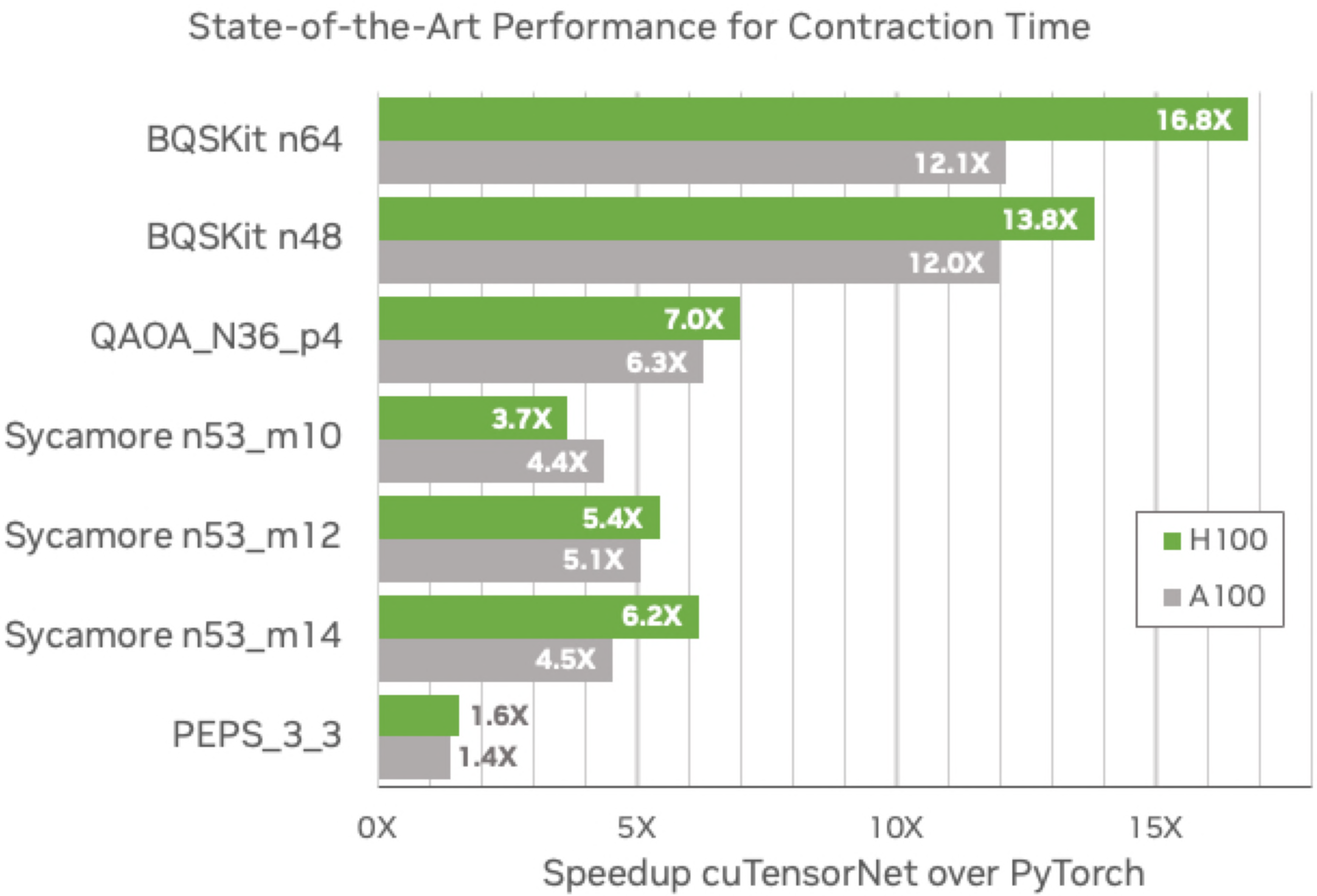


Tensor network contraction path finder available in **cuTensorNet** produces state-of-the-art contraction paths

[1] Gray & Kourtis, Hyper-optimized tensor network contraction, 2021 https://quantum-journal.org/papers/q-2021-03-15-410/pdf/
[2] opt-einsum https://pypi.org/project/opt-einsum/

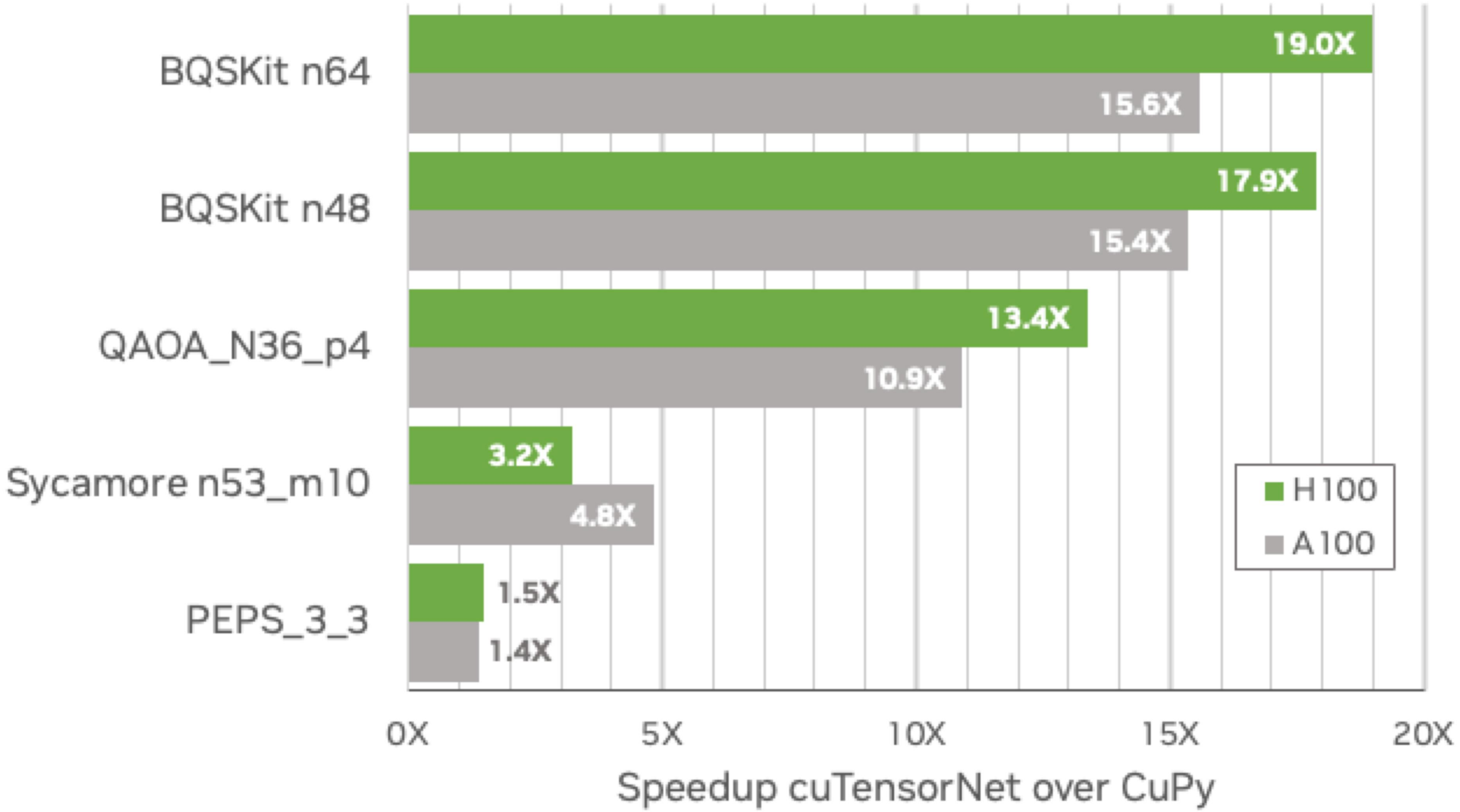
cuTensorNet Module of cuQuantum

Tensor contraction performance



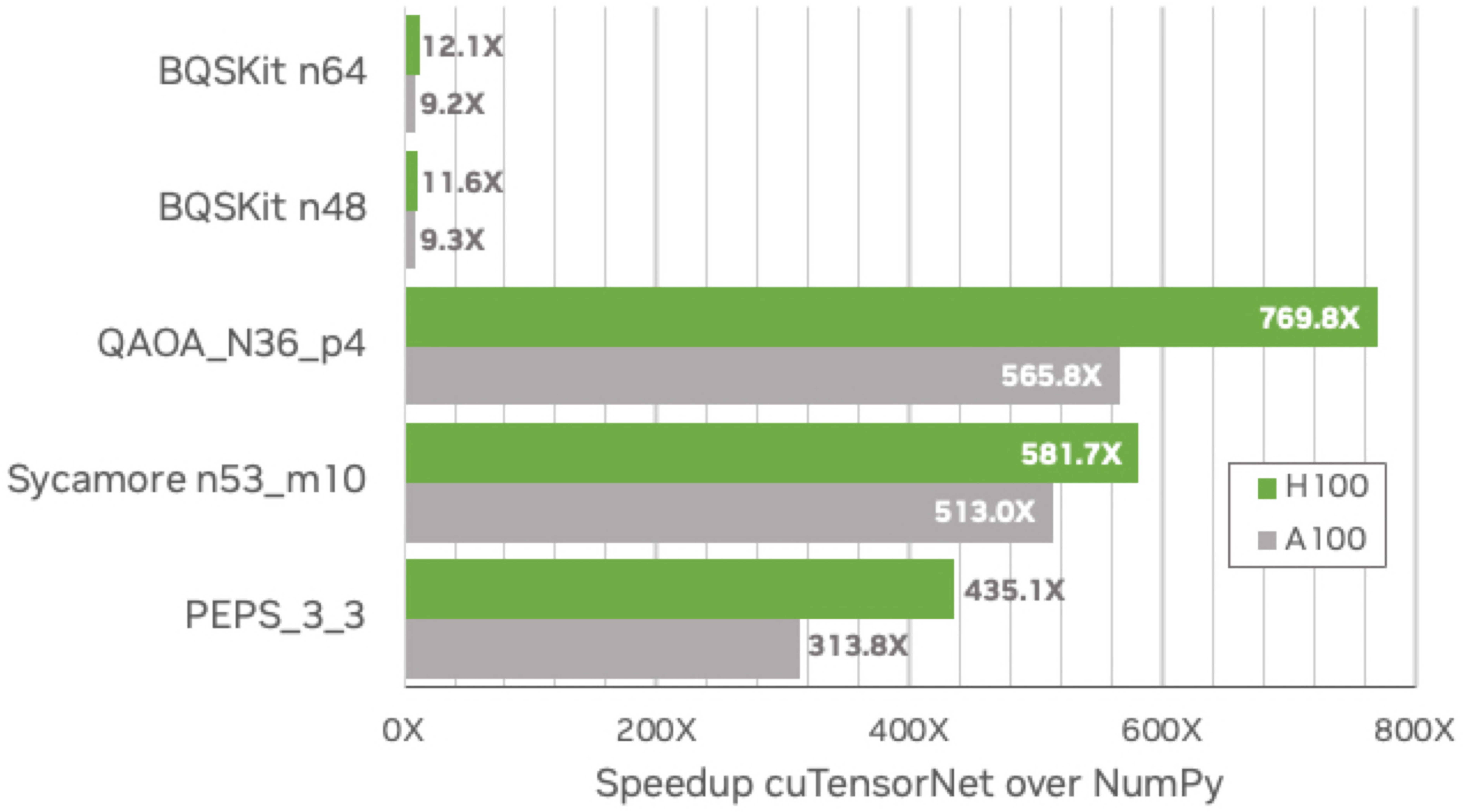
cuTensorNet Module of cuQuantum

Tensor contraction performance



cuTensorNet Module of cuQuantum

Tensor contraction performance

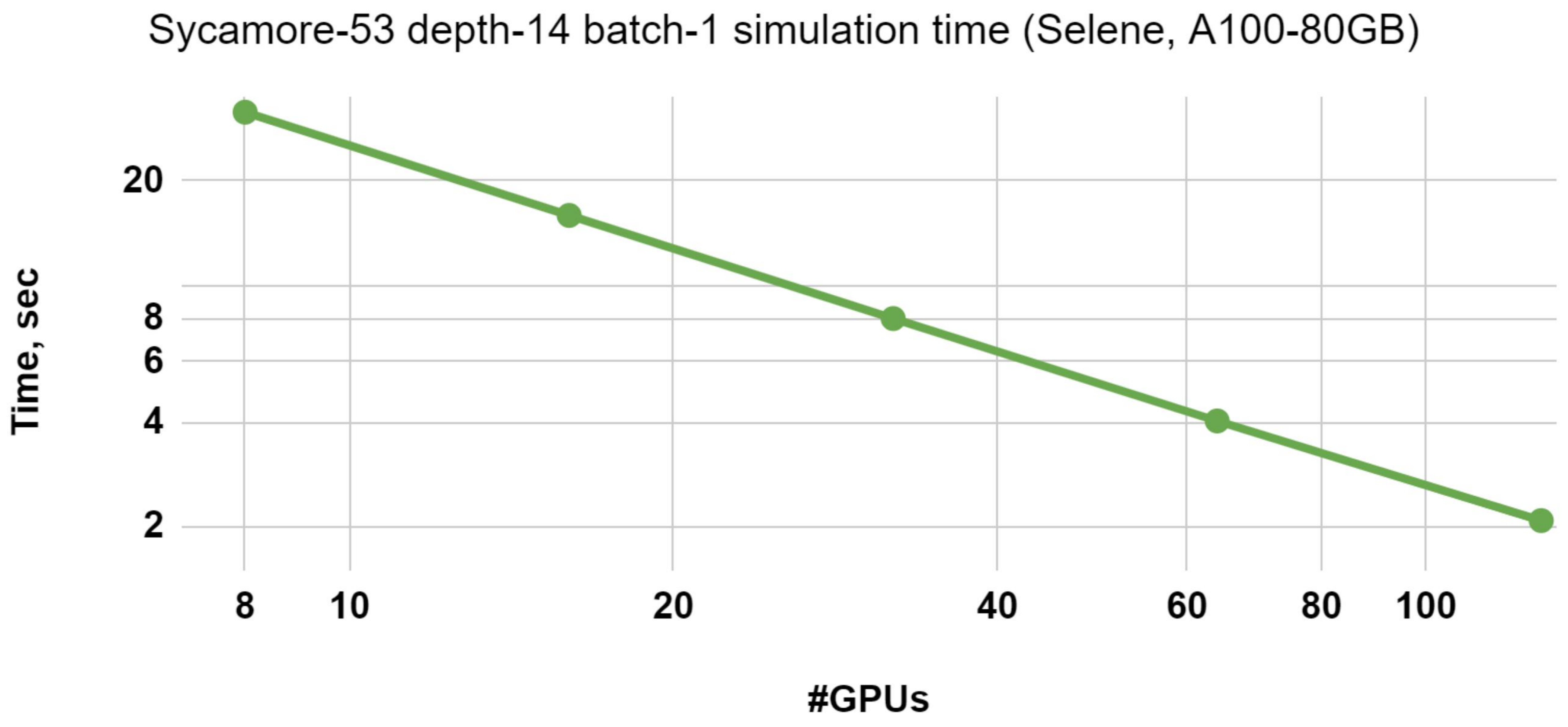


cuTensorNet Module of cuQuantum: Scalability

Distributed multi-GPU/multi-node tensor network contraction via cuTensorNet

cuTensorNet Module of cuQuantum: Scalability

Distributed multi-GPU/multi-node tensor network contraction via cuTensorNet



Simulation of a single amplitude of the 53-qubit random quantum circuit with 14 layers (Google's Sycamore)

cuTensorNet Module of cuQuantum: Generic Polymorphic High-Level API

High-level building blocks for defining and computing tensor network states

Tensor Network State:

- initialize
- applyTensor
- applyNetworkOperator
- updateTensor
- finalize
- configure
- prepare
- compute

Quantum Circuit Simulators

Quantum Chemistry Simulators

Condensed Matter Simulators

cuTensorNet Module of cuQuantum: Generic Polymorphic High-Level API

High-level building blocks for defining and computing tensor network states

MPS Pure Tensor Network State

Pure Tensor Network State

Mixed Tensor Network State

Tensor Network State:

- initializeXXX (e.g., XXX=MPS)
- applyTensor
- applyNetworkOperator
- updateTensor
- finalizeXXX (e.g., XXX=MPS)
- configure
- prepare
- compute

Quantum Circuit Simulators

Quantum Chemistry Simulators

Condensed Matter Simulators

cuTensorNet Module of cuQuantum: Generic Polymorphic High-Level API

High-level building blocks for defining and computing tensor network states

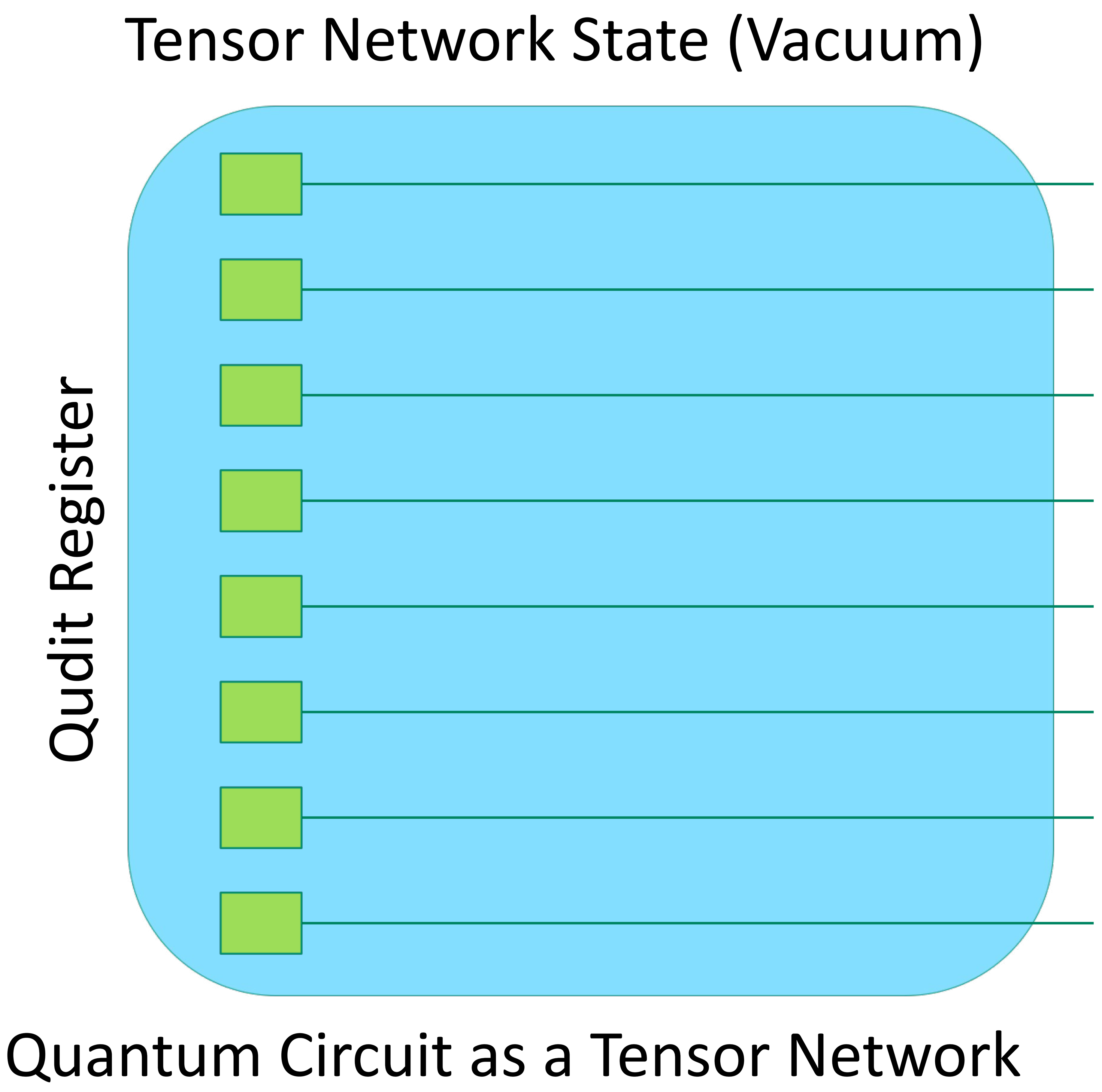
cuTensorNet Module of cuQuantum: Generic Polymorphic High-Level API

High-level building blocks for defining and computing tensor network states

- Reduced Density Matrices and State Sampling are available in cuQuantum-23.06
- State Accessor, Expectation Value and MPS states are expected in the upcoming release

Tensor Network Computing: Tensor Network State

cutensornetCreateState: Creates an empty tensor network state (vacuum)

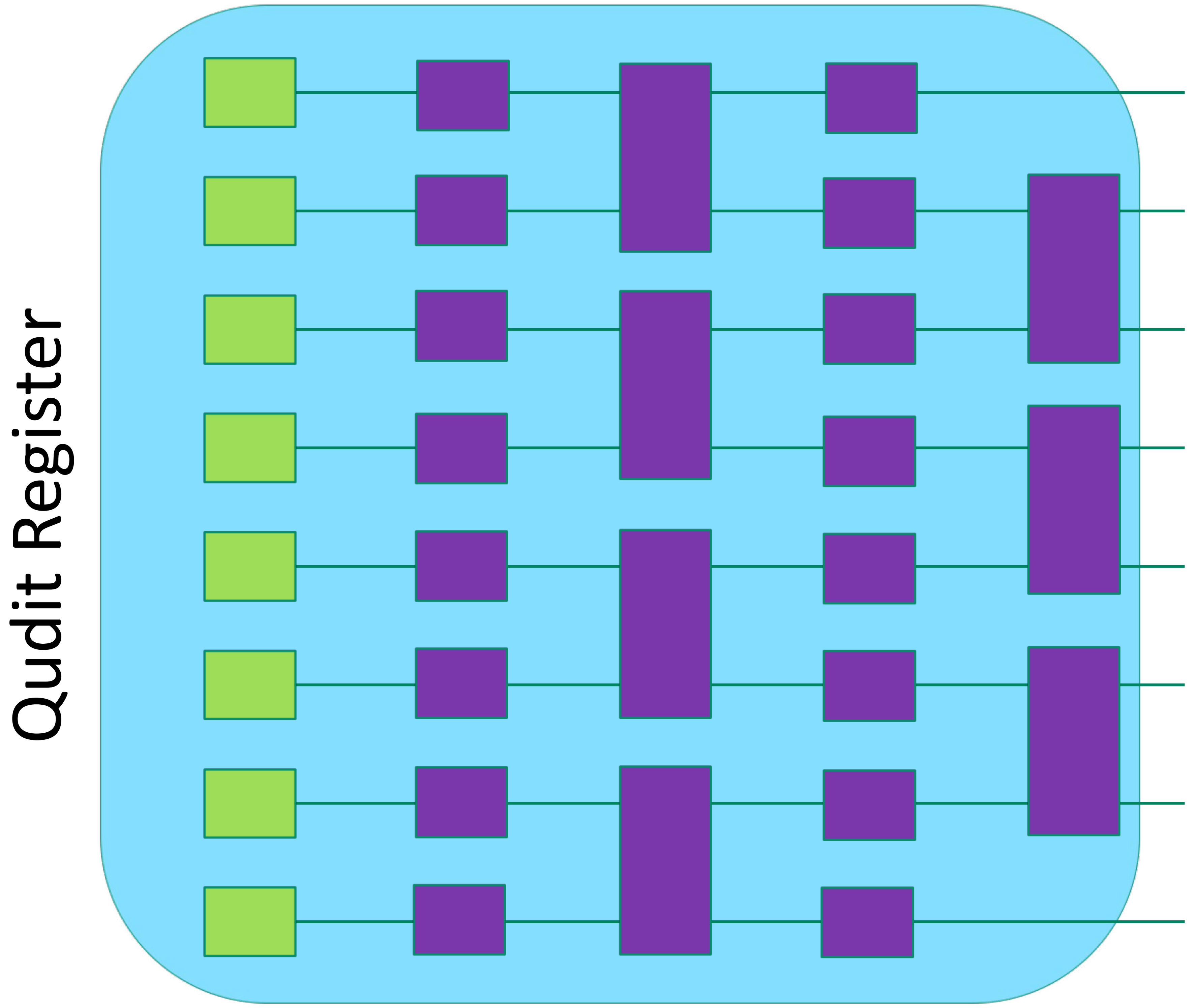


- Specify mode (qudit) dimensions
- Specify purity
- Specify data type

Tensor Network Computing: Tensor Network State

cutensornetStateApplyTensor: Applies tensor operators

Quantum Gates
(state evolution)

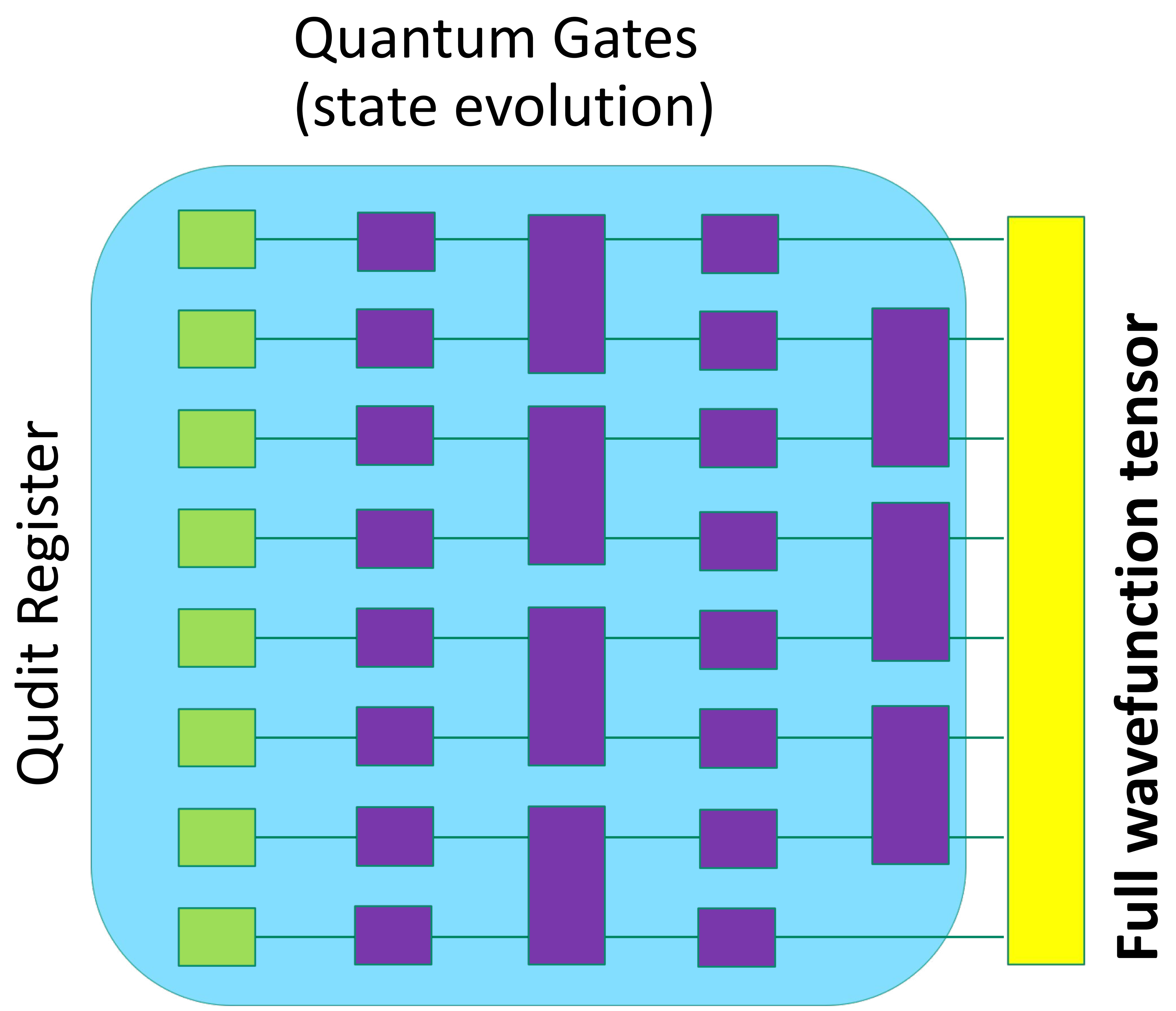


Quantum Circuit as a Tensor Network

- Apply arbitrary qudit gates

Tensor Network Computing: Amplitudes Accessor

cutensornetStateAccessor: Configure, Prepare, Compute



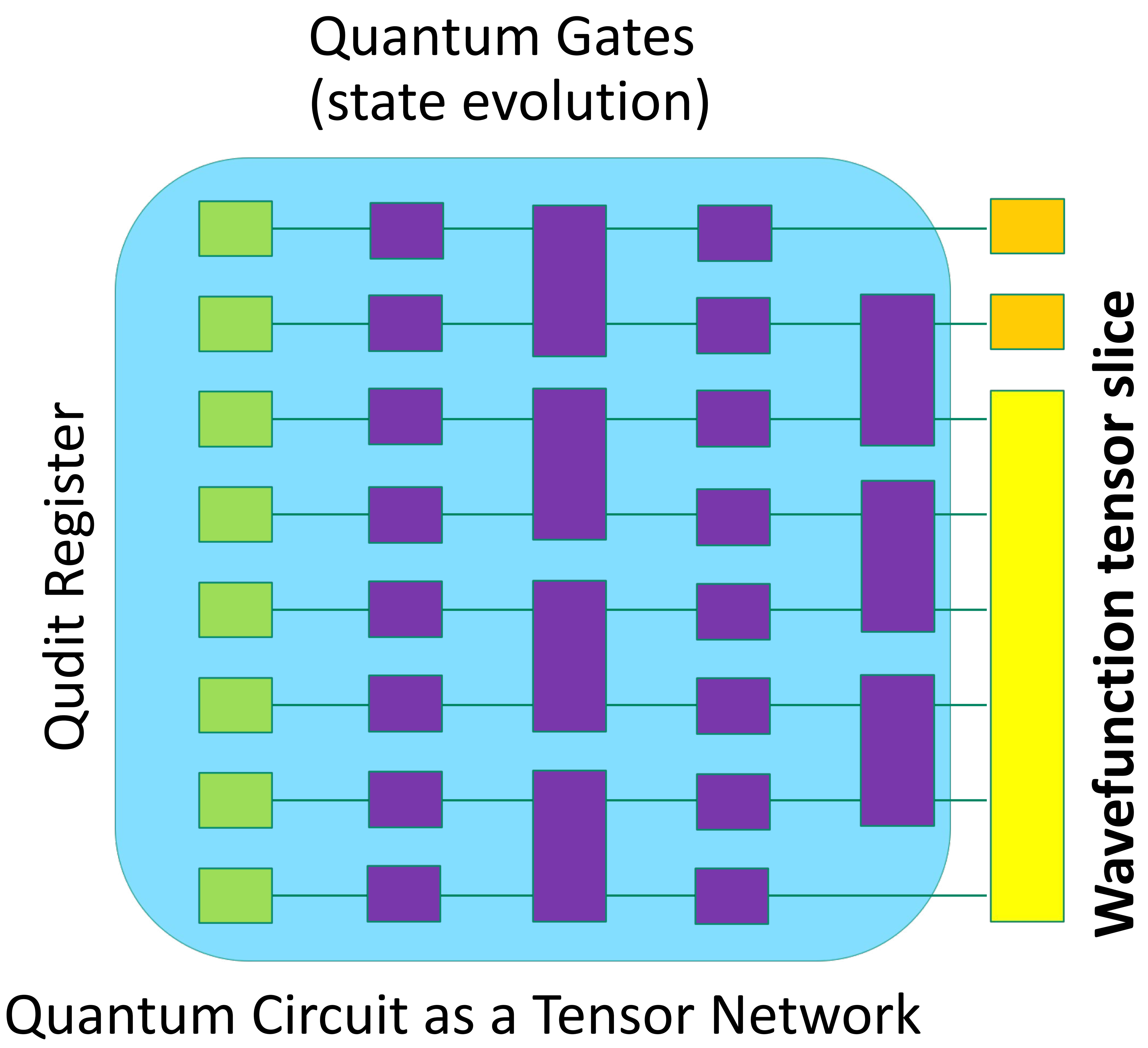
Quantum Circuit as a Tensor Network

- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

* Upcoming feature

Tensor Network Computing: Amplitudes Accessor

cutensornetStateAccessor: Configure, Prepare, Compute

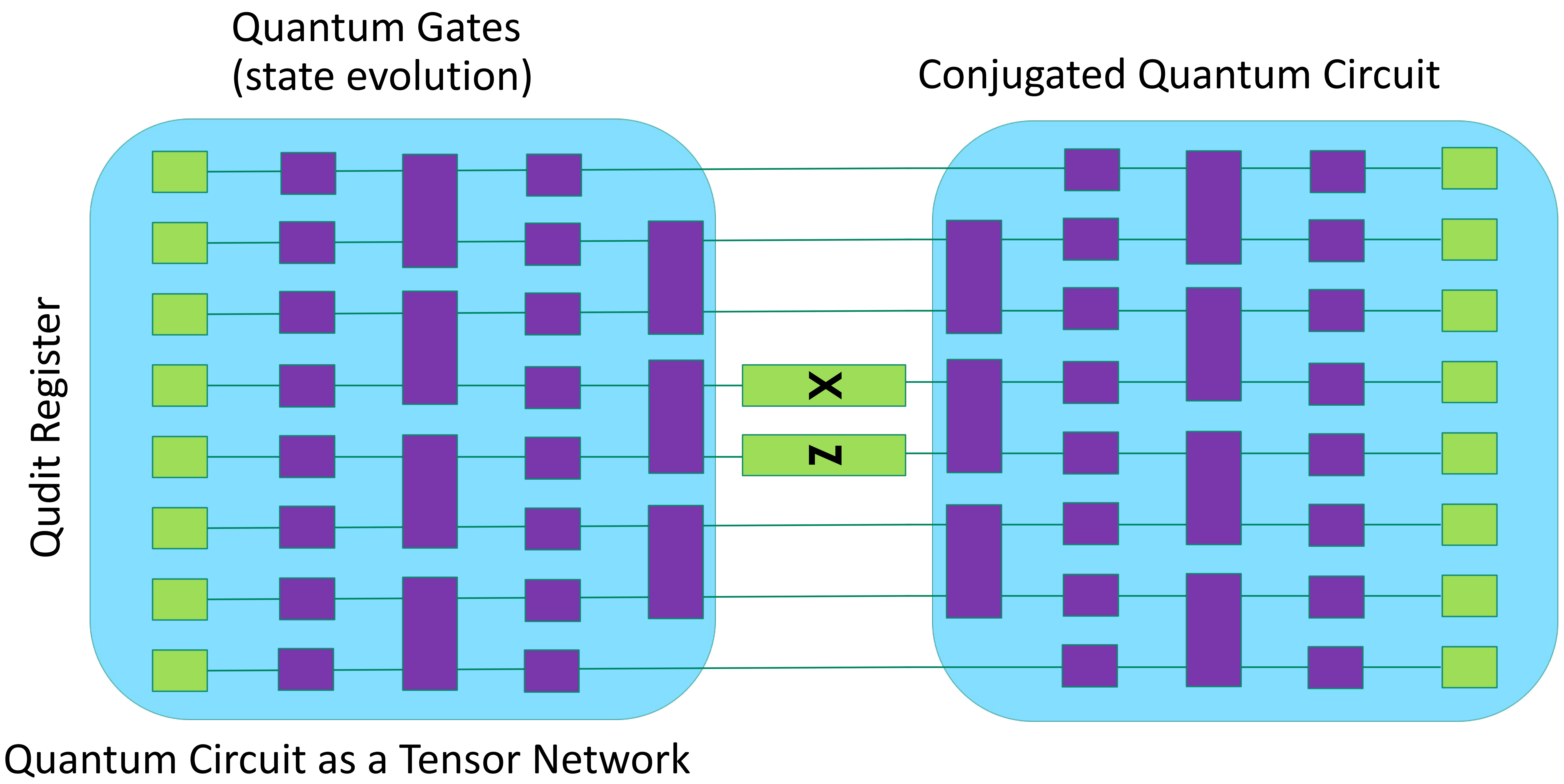


- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

* Upcoming feature

Tensor Network Computing: Expectation Values

cutensornetStateExpectation: Configure, Prepare, Compute

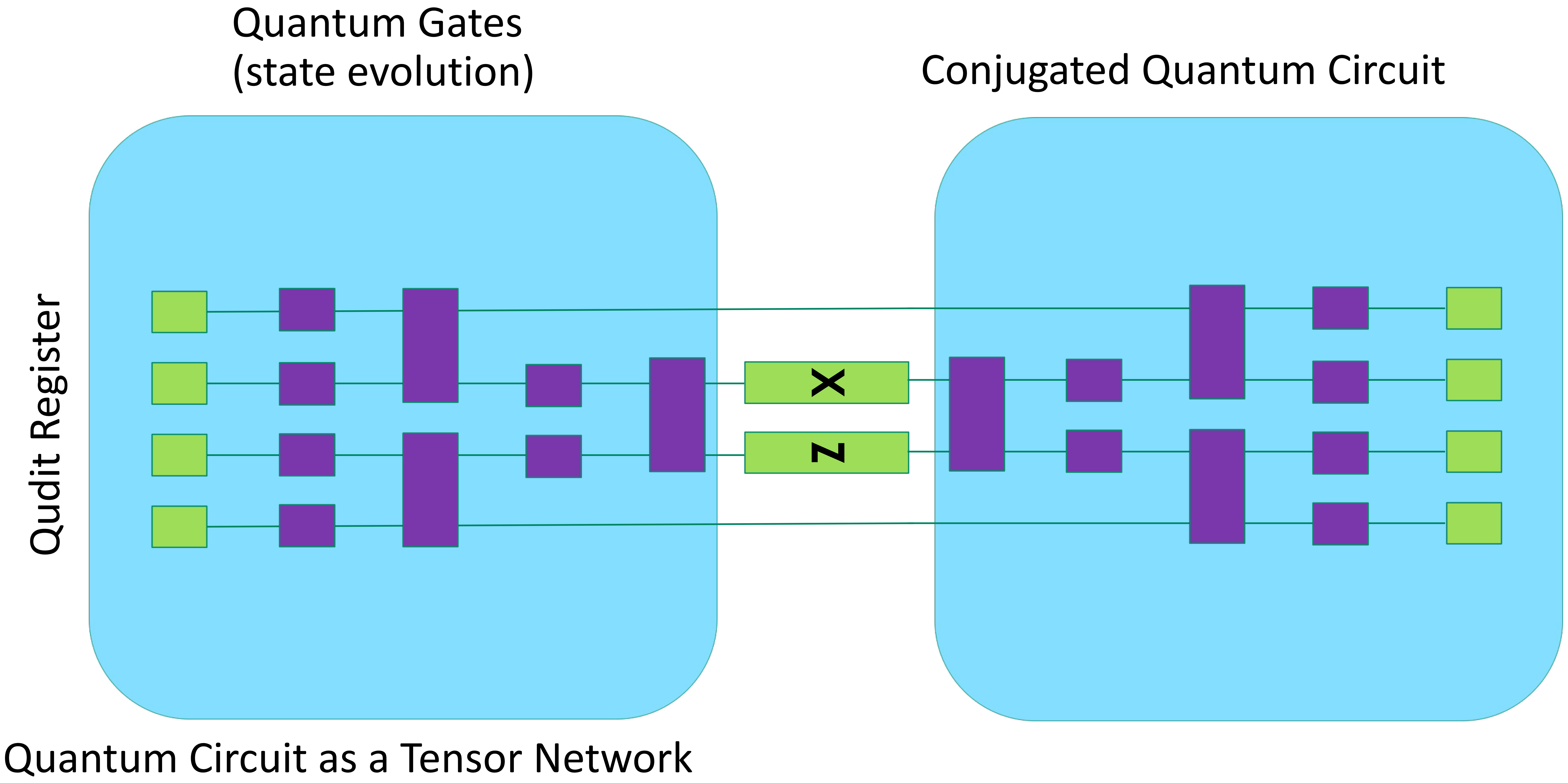


- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

* Upcoming feature

Tensor Network Computing: Expectation Values

cutensornetStateExpectation: Configure, Prepare, Compute

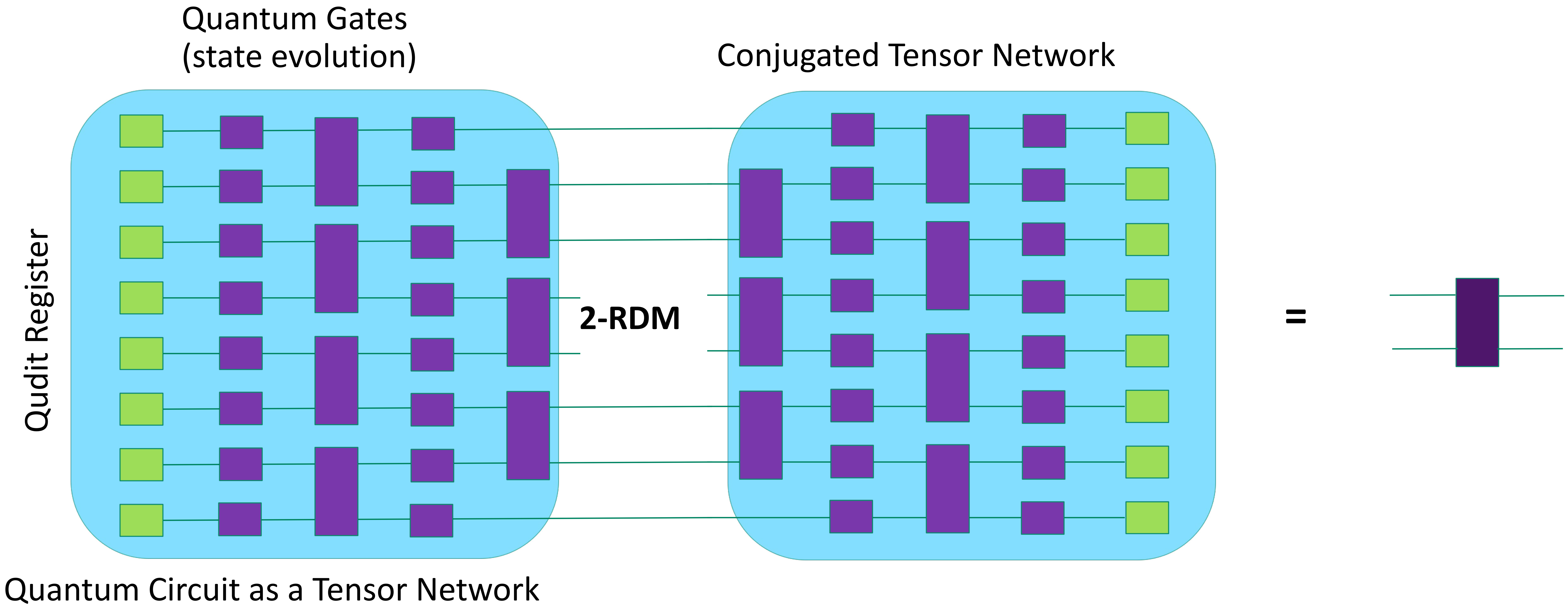


- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

* Upcoming feature

Tensor Network Computing: Reduced Density Matrices

cutensornetStateMarginal: Configure, Prepare, Compute



- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

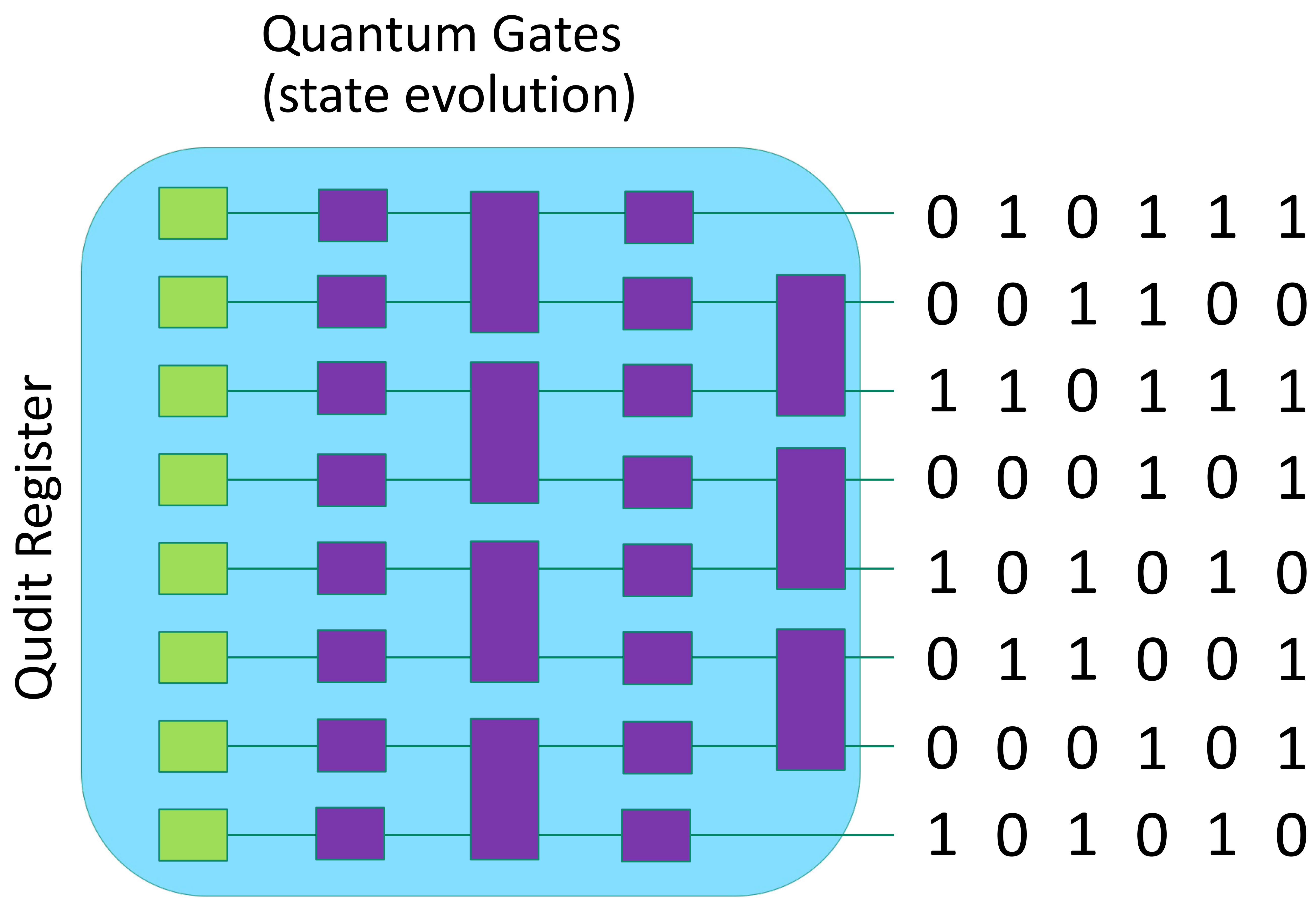
Tensor Network Computing: Reduced Density Matrices

cutensornetStateMarginal: Configure, Prepare, Compute

- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the result
- **Update**: Update tensors, recompute

Tensor Network Computing: Output Tensor Sampler

cutensornetStateSampler: Configure, Prepare, Compute

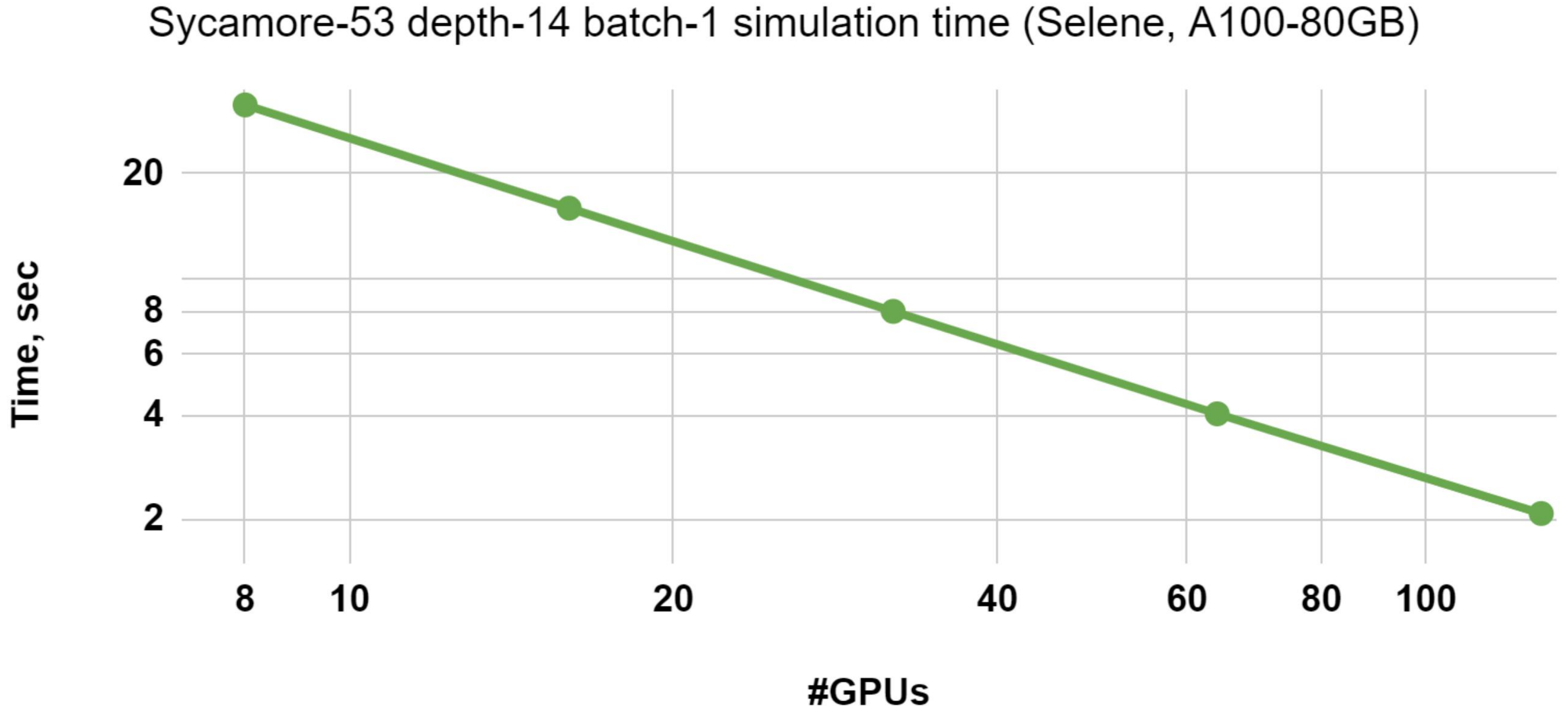


Quantum Circuit as a Tensor Network

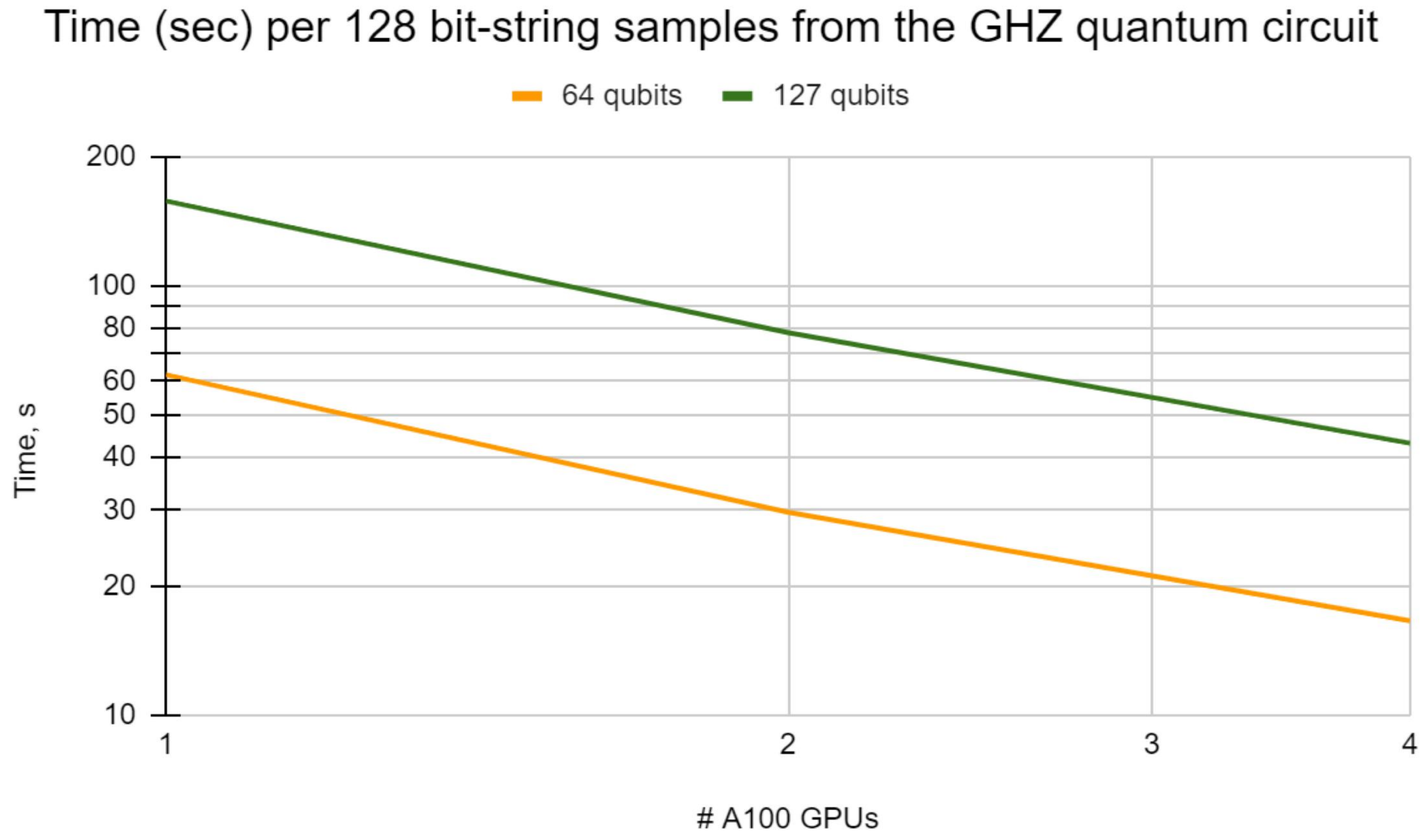
- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute any number of samples
- **Update**: Update tensors, recompute

cuTensorNet Module of cuQuantum: Scalability

Distributed multi-GPU/multi-node tensor network contraction via cuTensorNet



Simulation of a single amplitude of the 53-qubit random quantum circuit with 14 layers (Google's Sycamore)

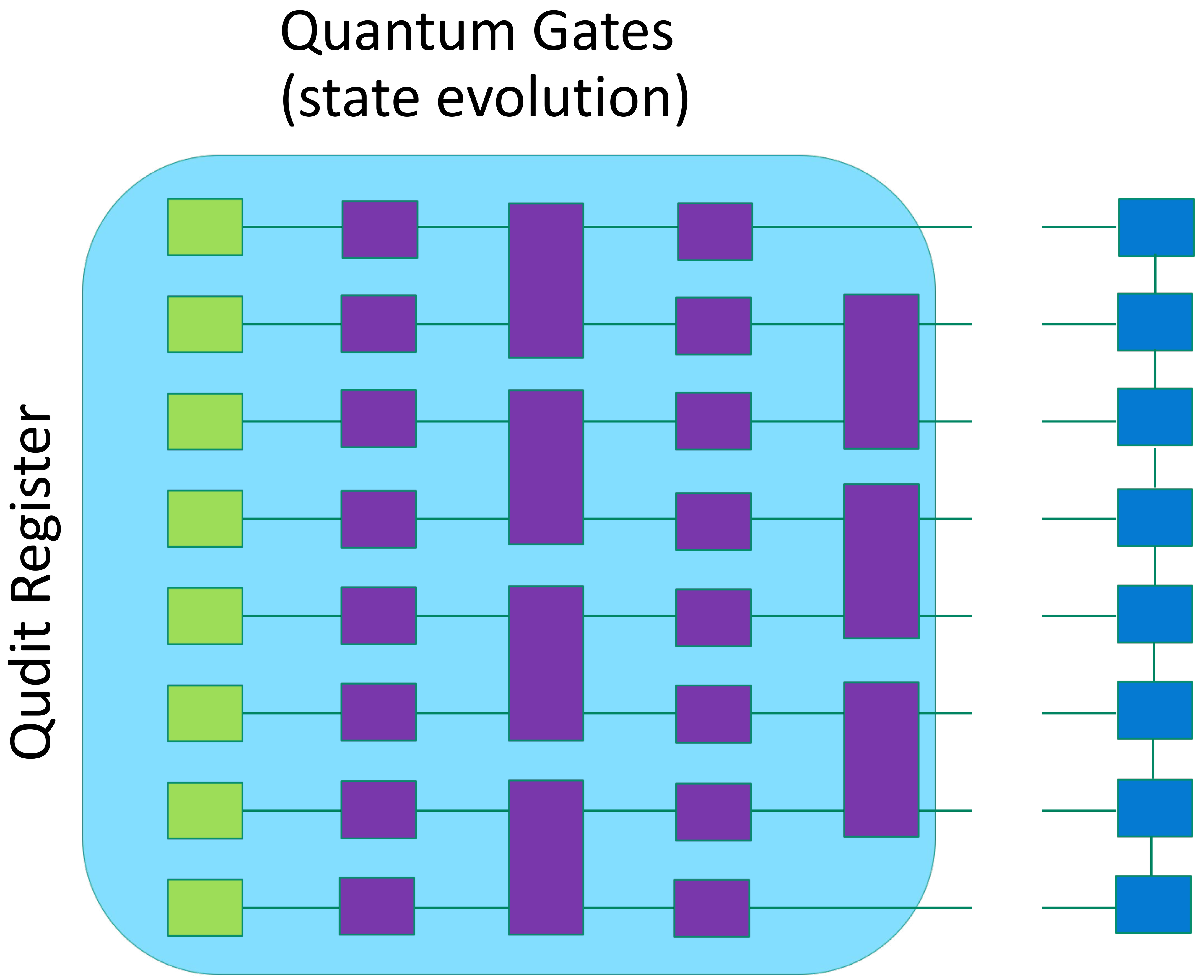


Direct bit-string sampling of the GHZ quantum circuits with 64 and 127 qubits

* Scheduled for an upcoming release

Tensor Network Computing: MPS Factorization of the State

cutensornetStateFinalizeMPS: Configure, Prepare, Compute

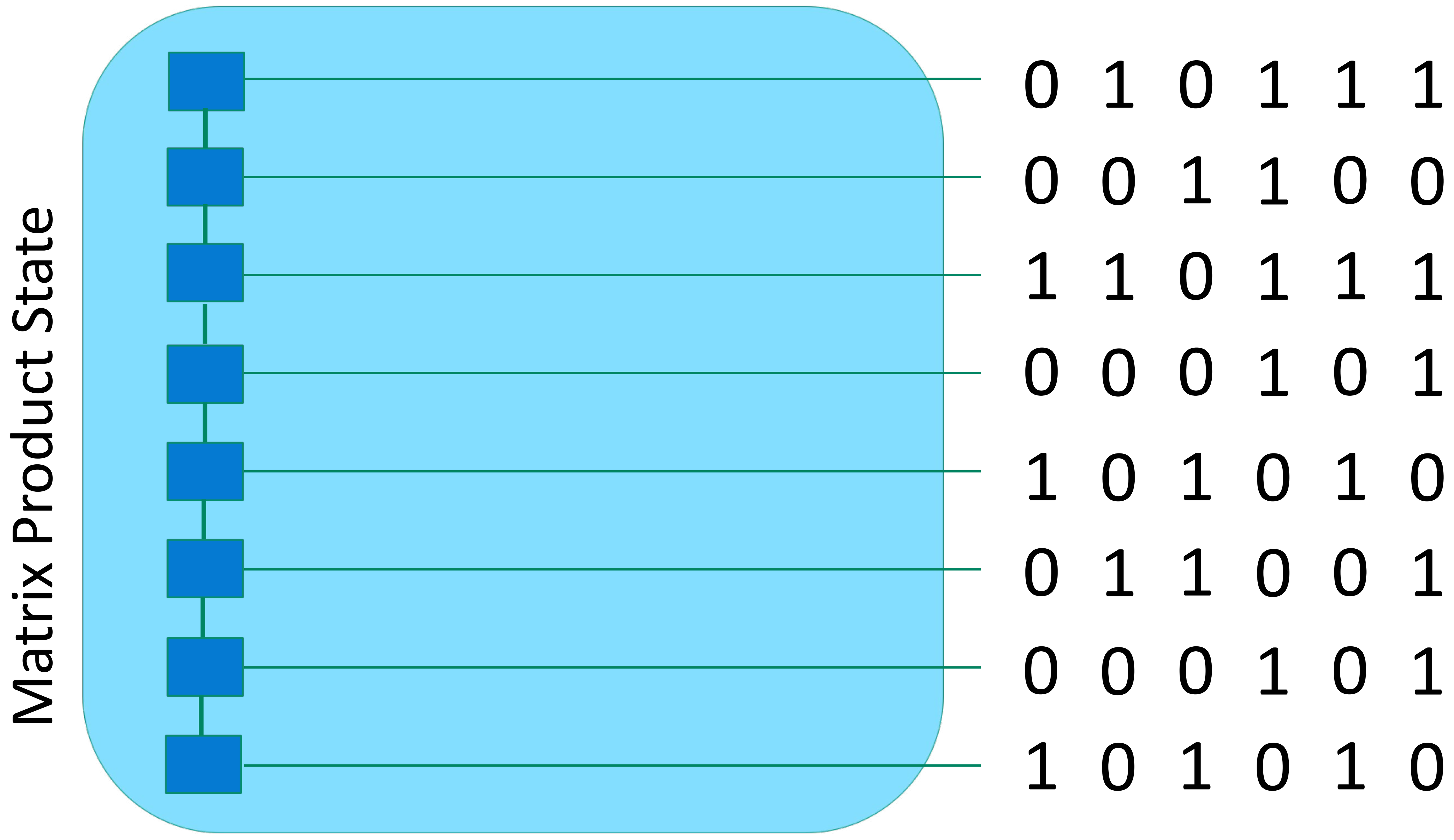


Quantum Circuit as a Tensor Network

- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute the MPS factorized state
- **Update**: Update tensors, recompute

Tensor Network Computing: Output Tensor Sampler

cutensornetStateSampler: Configure, Prepare, Compute



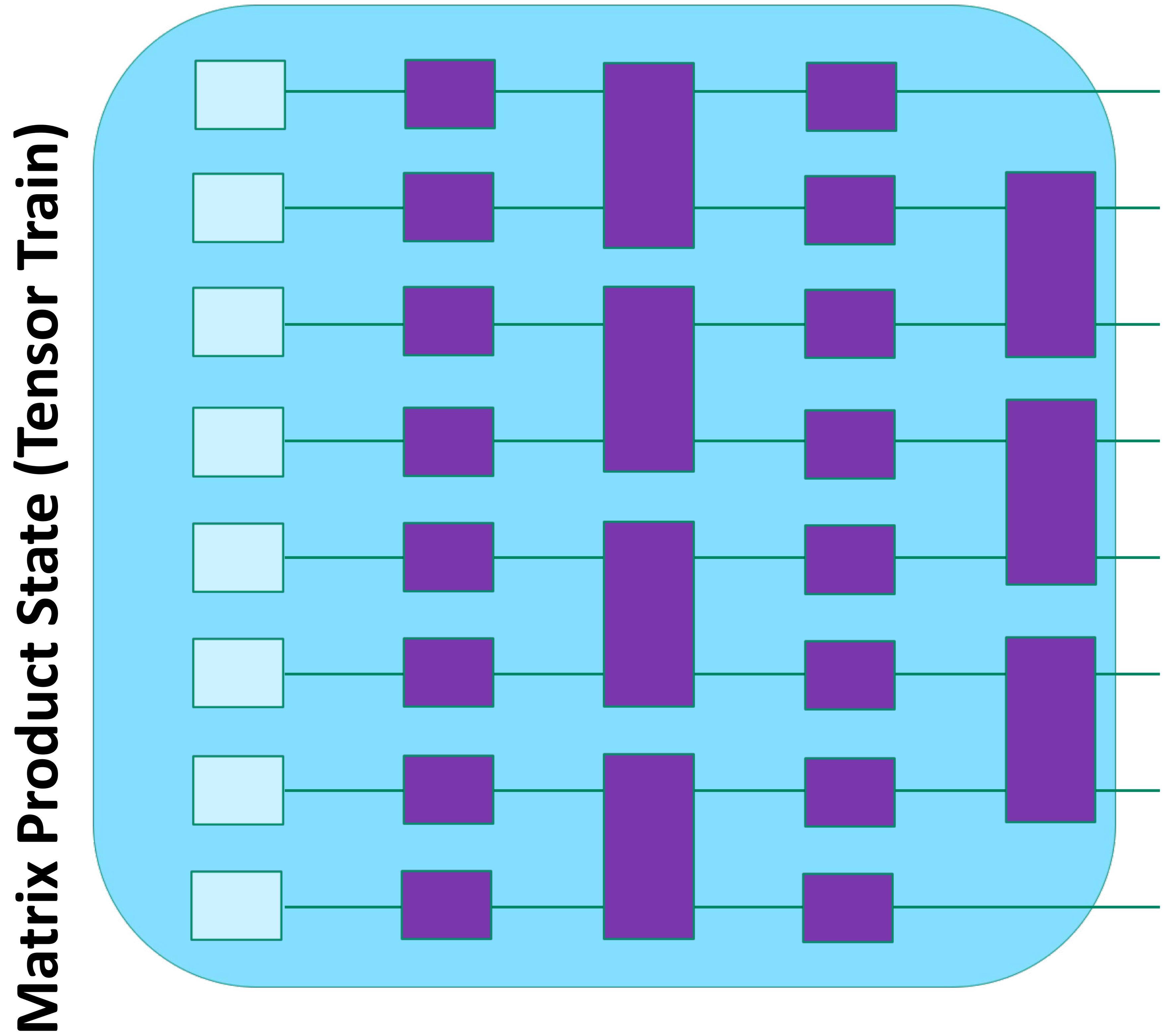
Quantum Circuit as a Tensor Network

- **Configure**: Set computation parameters
- **Prepare**: Prepare computation
- **Compute**: Compute any number of samples
- **Update**: Update tensors, recompute

Amplitudes Accessor, Expectation Value, Reduced Density Matrix, and Sampler will work with any tensor network state

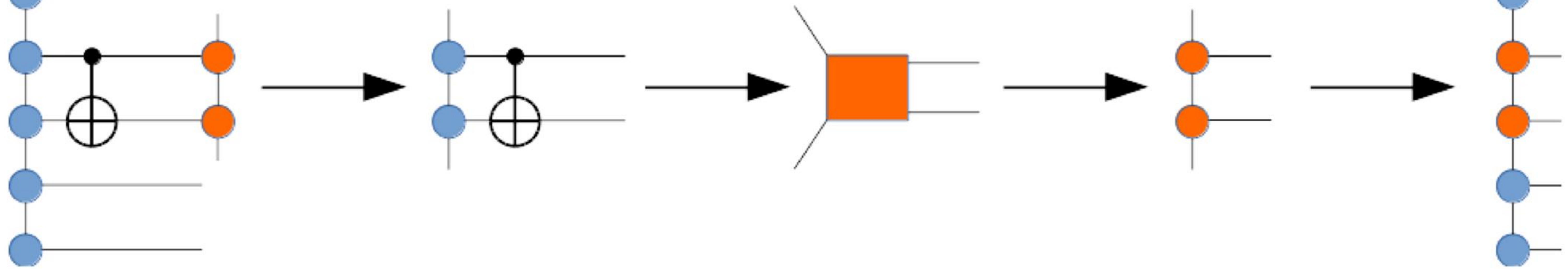
Tensor Network Computing: MPS Factorization

MPS state evolution via low-level cuTensorNet primitives



Initial state

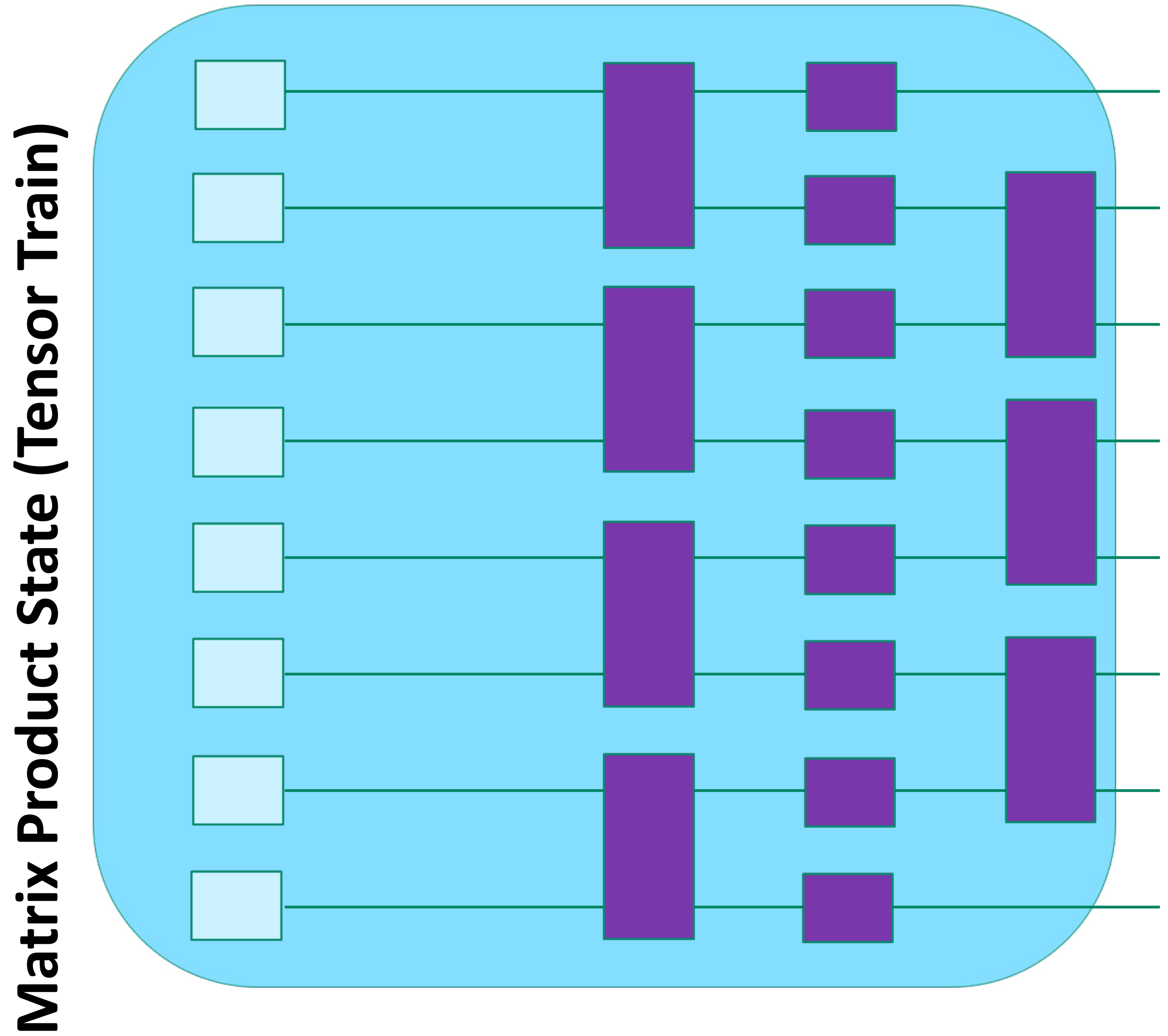
- The necessary **contract-decompose** low-level primitives are provided by **cuTensorNet**:
  - **Decompose** a tensor into two tensors (SVD, QR)
  - **Apply-Split**: Applies a tensor operator to two MPS tensors, followed by decomposition back into two updated MPS tensors



2-body quantum gate application

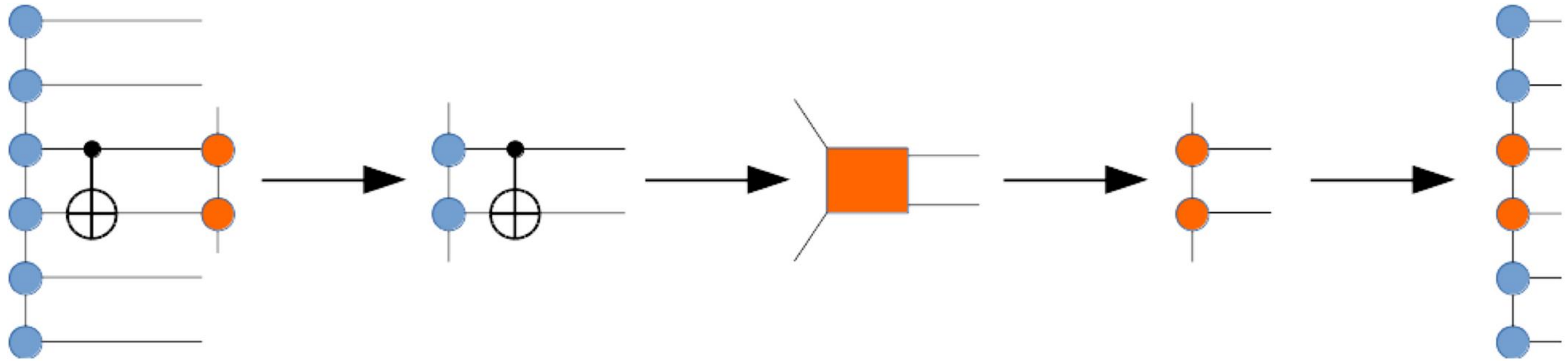
Tensor Network Computing: MPS Factorization

MPS state evolution via low-level cuTensorNet primitives



Absorbed a layer of 1-body quantum gates (no entanglement yet)

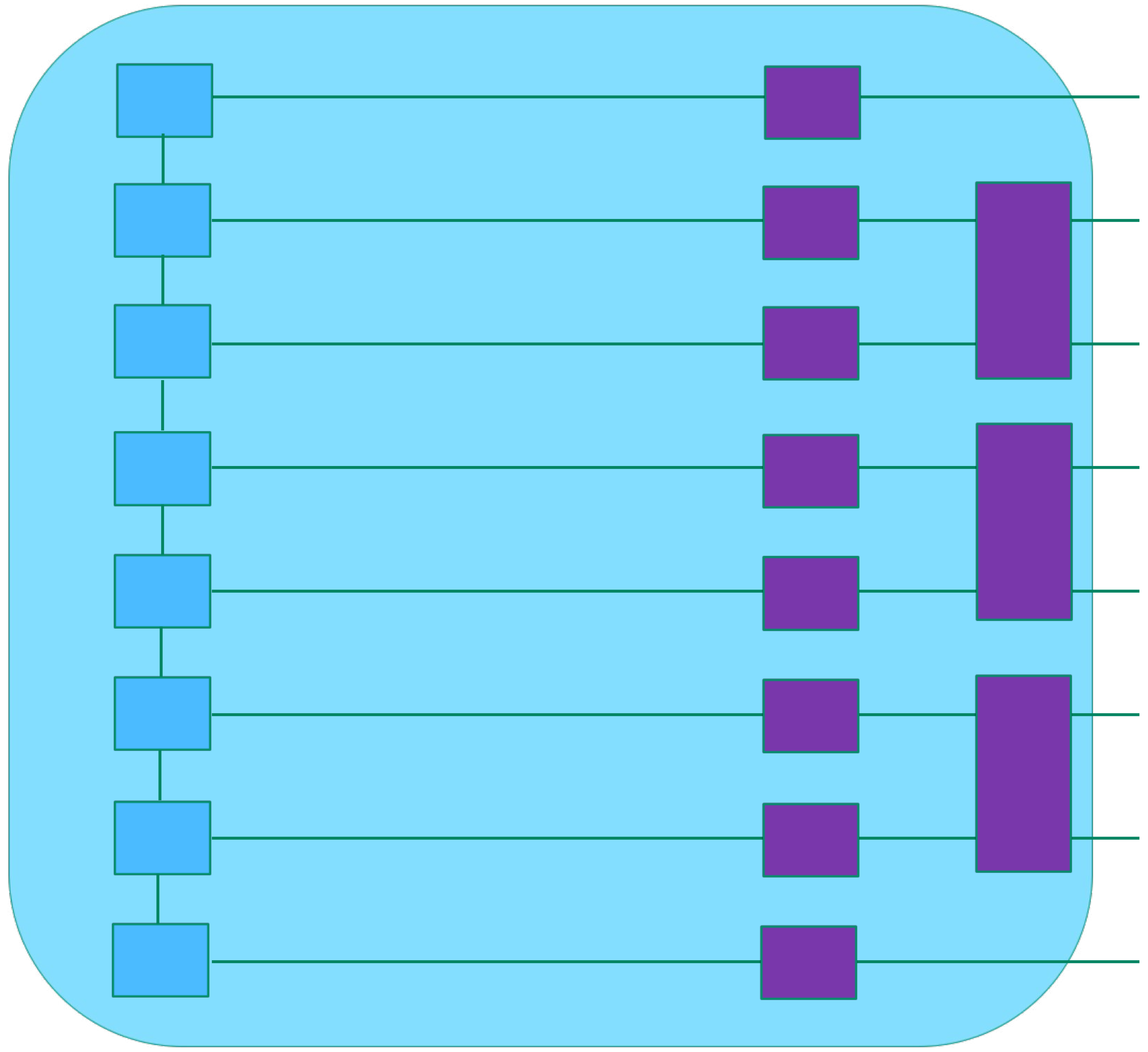
- The necessary **contract-decompose** low-level primitives are provided by **cuTensorNet**:
  - **Decompose** a tensor into two tensors (SVD, QR)
  - **Apply-Split**: Applies a tensor operator to two MPS tensors, followed by decomposition back into two updated MPS tensors



2-body quantum gate application

Tensor Network Computing: MPS Factorization

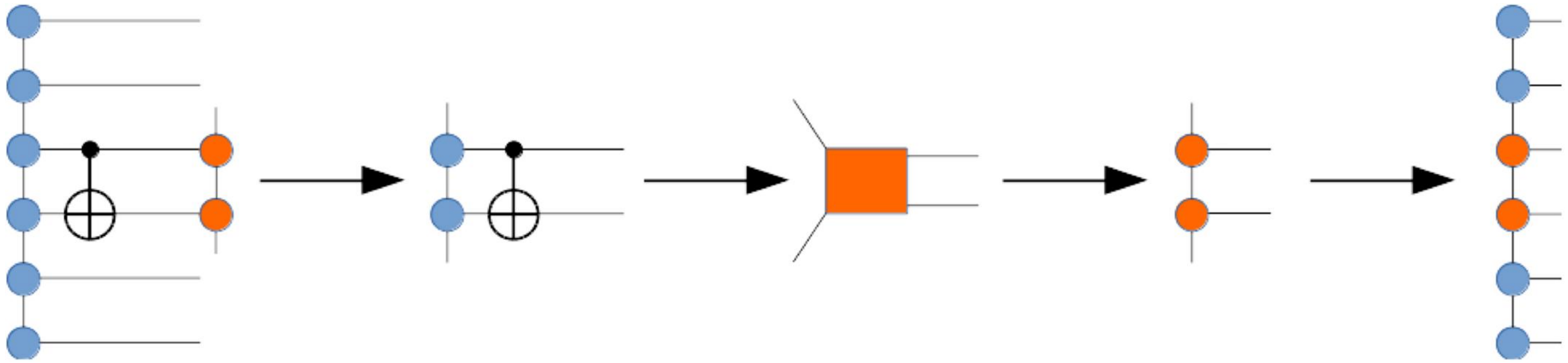
MPS state evolution via low-level cuTensorNet primitives



Matrix Product State (Tensor Train)

Absorbed a layer of 2-body quantum gates:
(entanglement created)

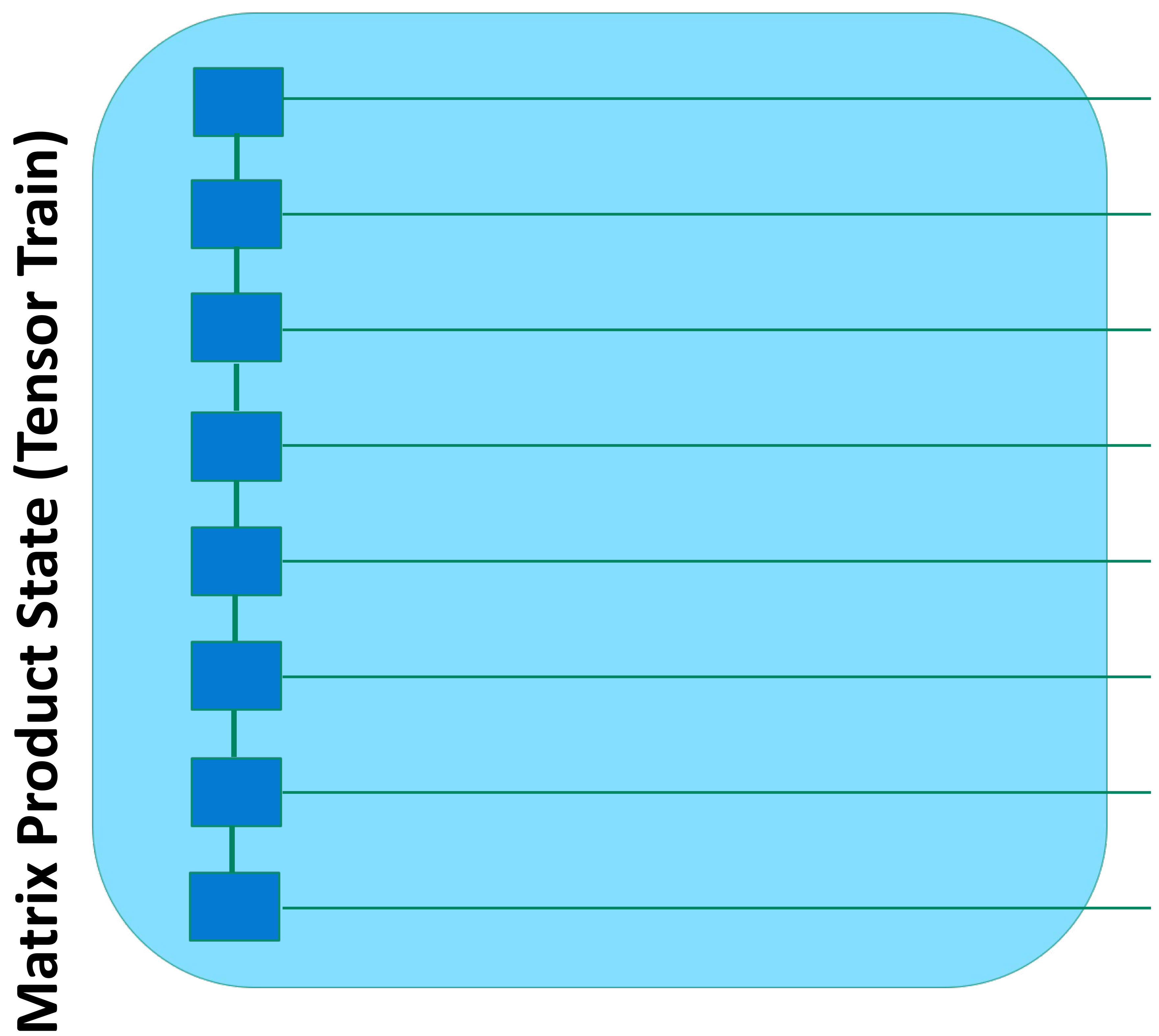
- The necessary **contract-decompose** low-level primitives are provided by **cuTensorNet**:
  - **Decompose** a tensor into two tensors (SVD, QR)
  - **Apply-Split**: Applies a tensor operator to two MPS tensors, followed by decomposition back into two updated MPS tensors



2-body quantum gate application

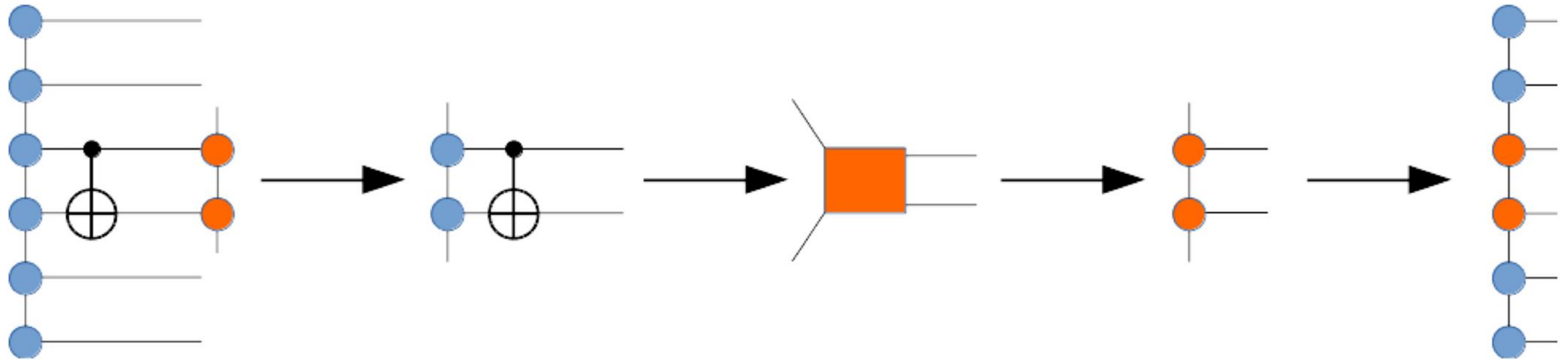
Tensor Network Computing: MPS Factorization

MPS state evolution via low-level cuTensorNet primitives



Absorbed another layer of 2-body quantum gates:
Entanglement grows (if MPS bond dimension allows)

- The necessary **contract-decompose** low-level primitives are provided by **cuTensorNet**:
  - **Decompose** a tensor into two tensors (SVD, QR)
  - **Apply-Split**: Applies a tensor operator to two MPS tensors, followed by decomposition back into two updated MPS tensors

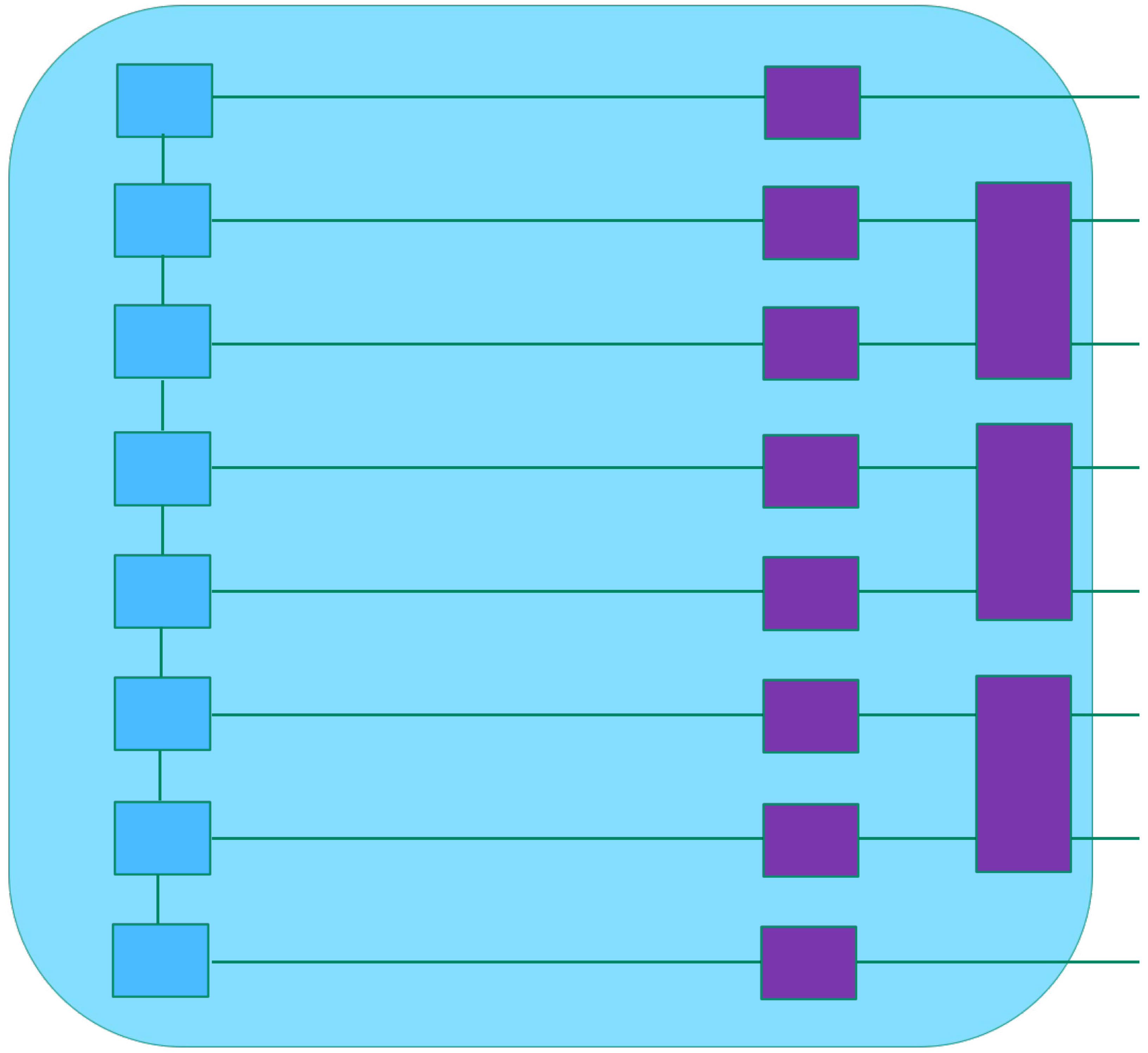


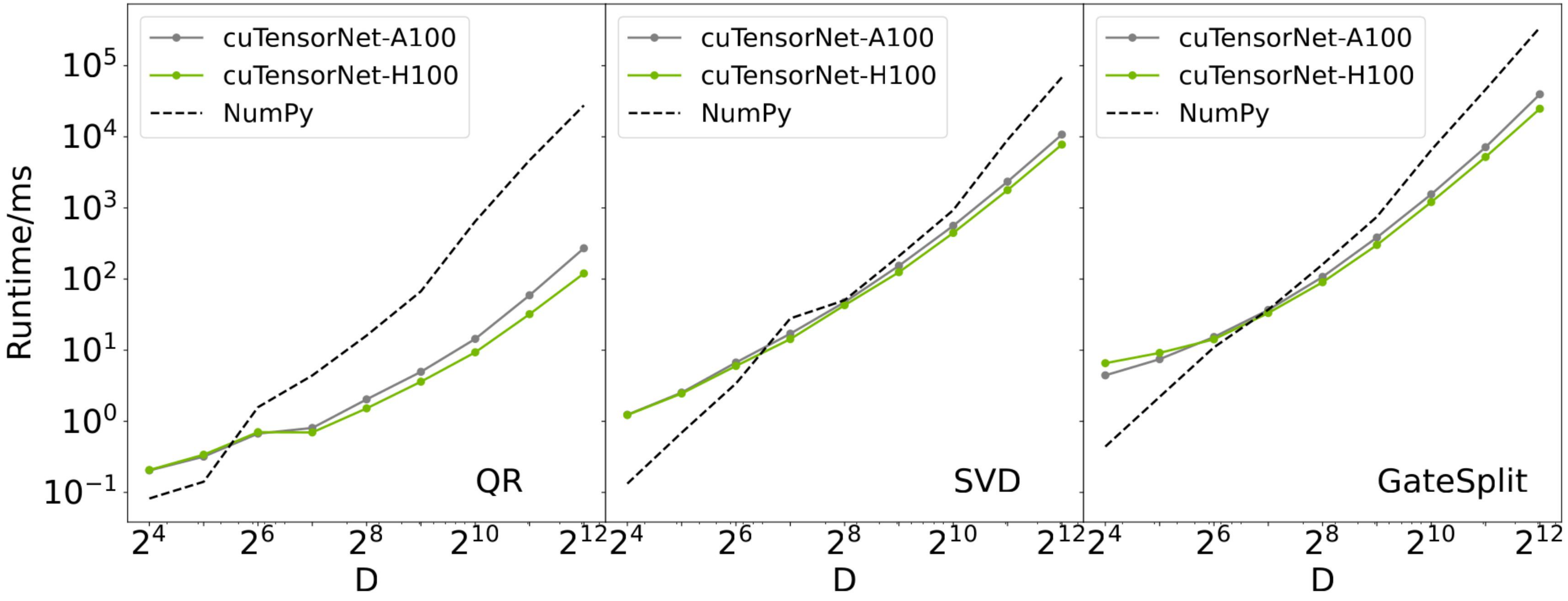
2-body quantum gate application

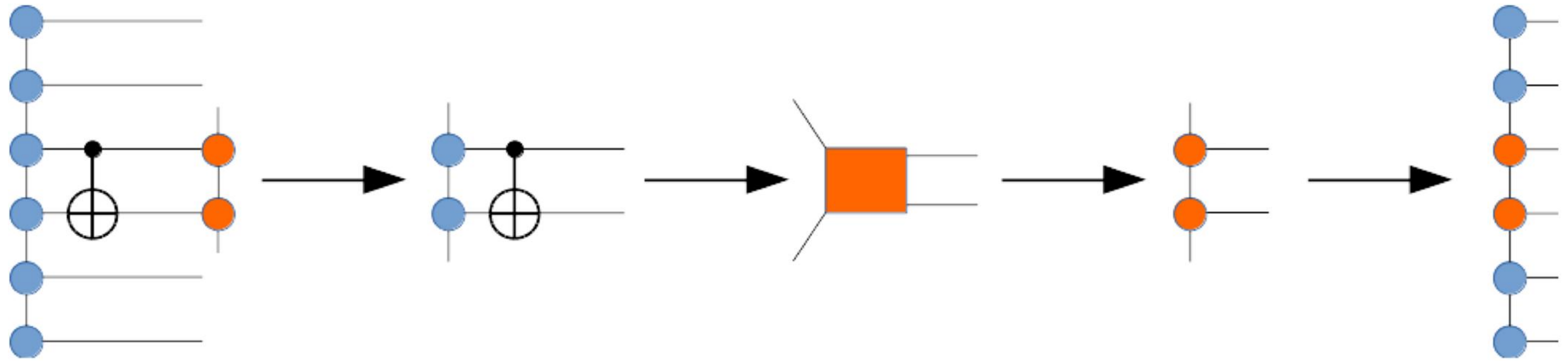
cuTensorNet Module of cuQuantum: MPS primitives

Contract-decompose primitives for implementing approximate tensor network simulators

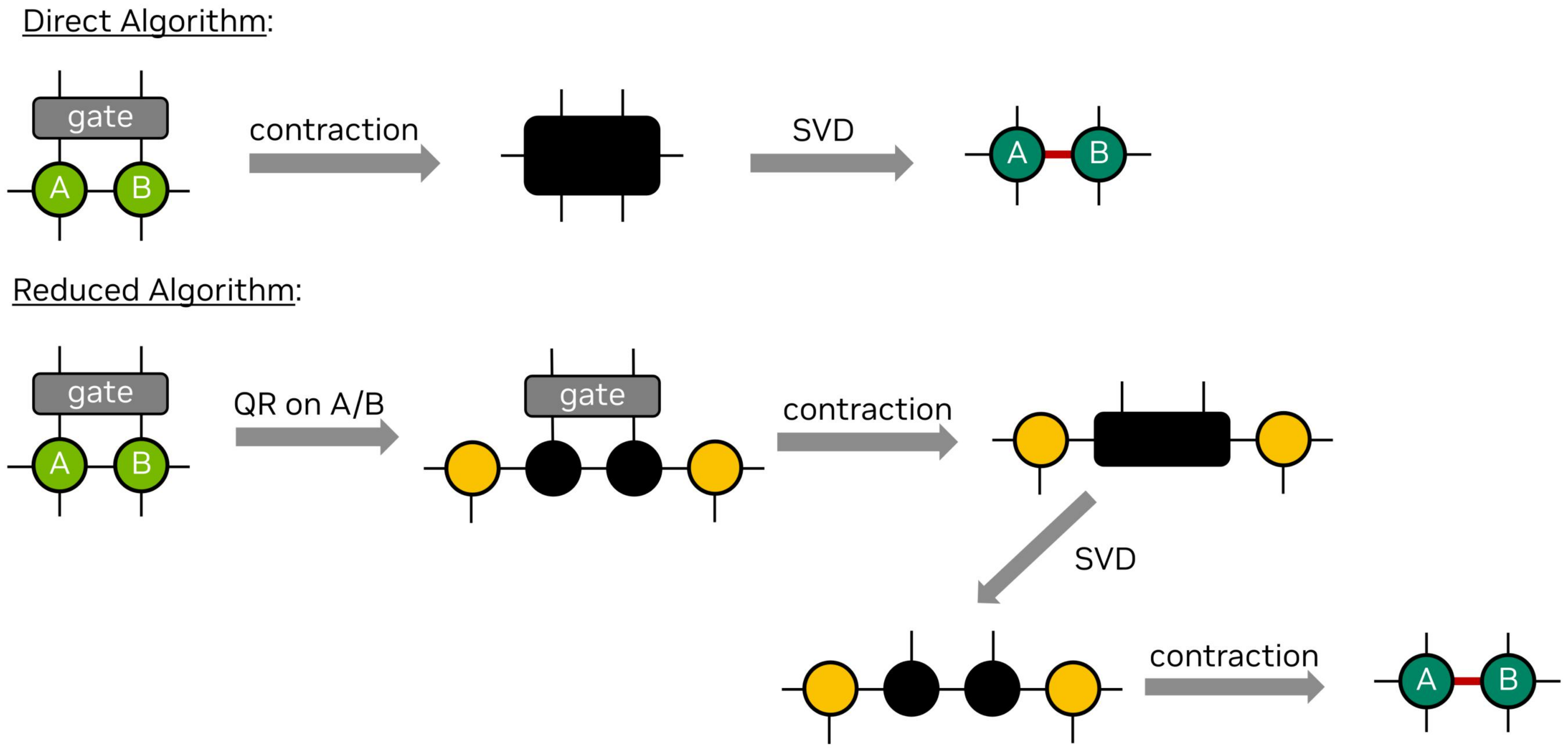
Matrix Product State (Tensor Train)







2-body quantum gate application



Exploring Reduced/Mixed Precision Arithmetic in cuTensorNet

Composite BF16/9 tensor core arithmetic fully reproduces the FP32 precision

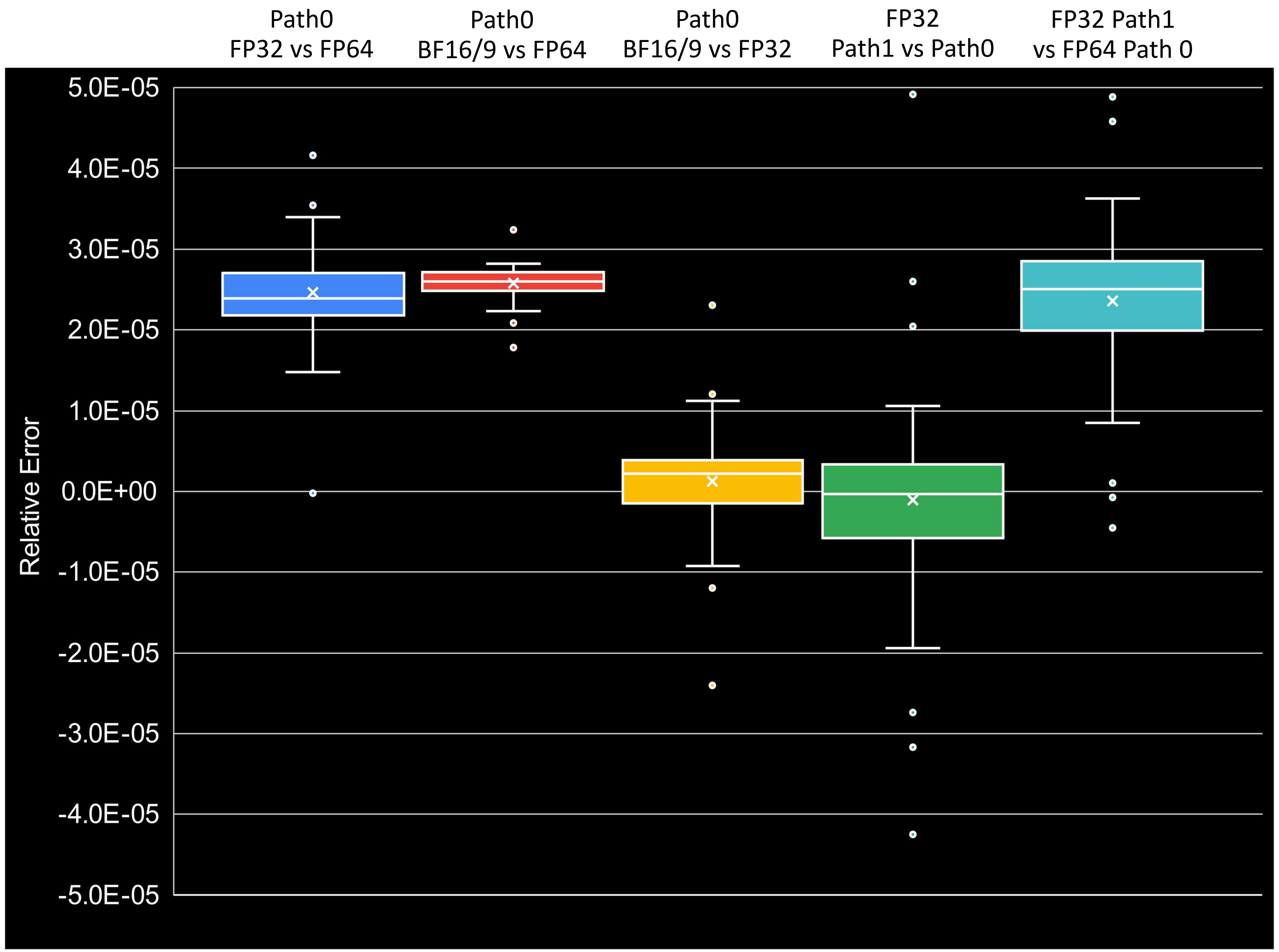
- The FP32 inputs are decomposed into 3 scaled BF16 components

$$a = a0 + 2^{-8}.a1 + 2^{-16}.a2$$
$$b = b0 + 2^{-8}.b1 + 2^{-16}.b2$$

- The multiply-add operation is computed as a sum of 9 scaled partial products

$$\begin{aligned} a * b + c = \; & a0.b0 + 2^{-8}.a0.b1 + 2^{-16}.a0.b2 \\ & + 2^{-8}.a1.b0 + 2^{-16}.a1.b1 + 2^{-24}.a1.b2 \\ & + 2^{-16}.a2.b0 + 2^{-24}.a2.b1 + 2^{-32}.a2.b2 + c \end{aligned}$$

- 53-qubit Sycamore random quantum circuit with 12 layers of random gates and computed probability amplitudes for 64 bit-strings
- 649216 total pairwise tensor contractions
- 1.7% of the tensor contractions account for 95% of the total 0.83 PFLOPs (k-dim >= 16)
  - We offload these to **BF16/9 tensor cores**
- The relative error of the computed probability amplitudes with BF16/9 is slightly less than FP32 when compared to the FP64 baseline
- Variation of amplitude values due to the use of different tensor network contraction paths for FP32 compute introduces larger differences than BF16/9



Summary

- **GPU computing** can drastically accelerate emulation of **quantum processors** and execution of classical pre- and post-processing steps (error correction, error mitigation, device calibration, etc.)
- **CUDA Quantum** extends the CUDA programming model to quantum processors
- **CUDA Quantum** enables **tight integration** of CPU, GPU, and QPU accelerators
- **cuQuantum** is a library of efficient GPU-accelerated **computational primitives** for quantum circuit simulator developers: Provides **state-vector** and **tensor-network** simulator building blocks
- **cuQuantum** delivers **significant speed-ups** in state-vector and tensor-network based simulators
- **cuQuantum** enables **multi-GPU/multi-node parallelization** in both kinds of simulators for clouds and HPC
- The new **high-level cuTensorNet** API provides easy-to-use high-level building blocks for TN simulators automatically enhanced with parallelization, intermediate reuse, and other performance optimizations
- **cuTensorNet** supports tensor network gradients via back-propagation, enabling easy integration with machine learning frameworks (the new ML-native Pythonic API are upcoming)
- **cuQuantum** is developed to ease the life of simulator developers, let us know what else you need

https://developer.nvidia.com/cuquantum-sdk

https://github.com/nvidia/cuda-quantum | https://catalog.ngc.nvidia.com/orgs/nvidia/containers/cuda-quantum

NVIDIA