



# CUDA Quantum & cuQuantum Bootcamp

Yun Yuan Wang (Pika Wang), Solutions Architect | NVAITC, NVIDIA Taiwan



# NVIDIA Quantum Platform Portfolio

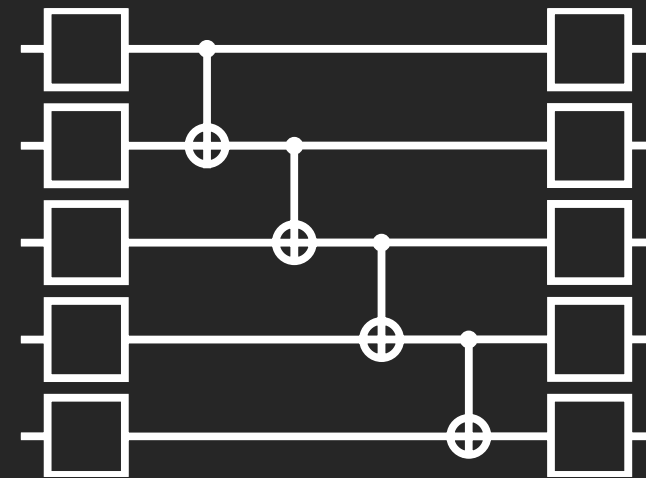
Research the computers of tomorrow with the most powerful computers of today

**CUDA Quantum** is the open-source quantum analog of CUDA. It enables domain scientists to easily and performantly coprogram CPU, GPU, and simulated or real QPUs

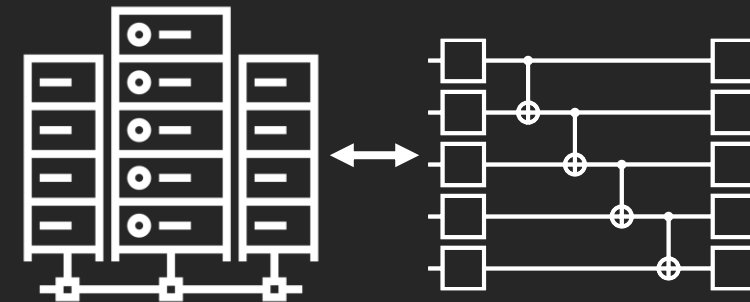
**cuQuantum** integrates with all quantum frameworks enabling GPU-accelerated, supercomputer scale quantum simulations

**DGX Quantum** is an integrated system with a low-latency connection from Grace Hopper to a quantum control system, enabling tight physical GPU integration with any QPU

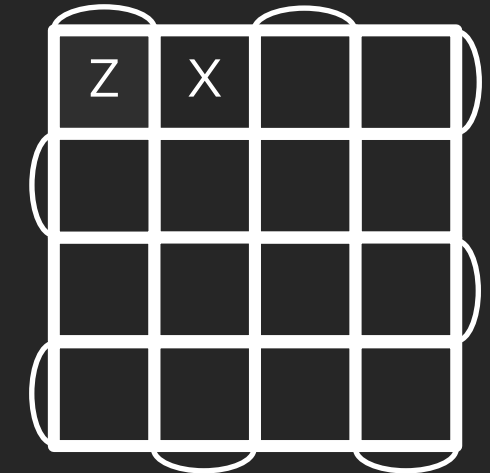
Quantum Algorithms Research



Quantum-Integrated Applications



Error Correction, Calibration, Control



**cuQuantum**

Accelerated Quantum Simulation

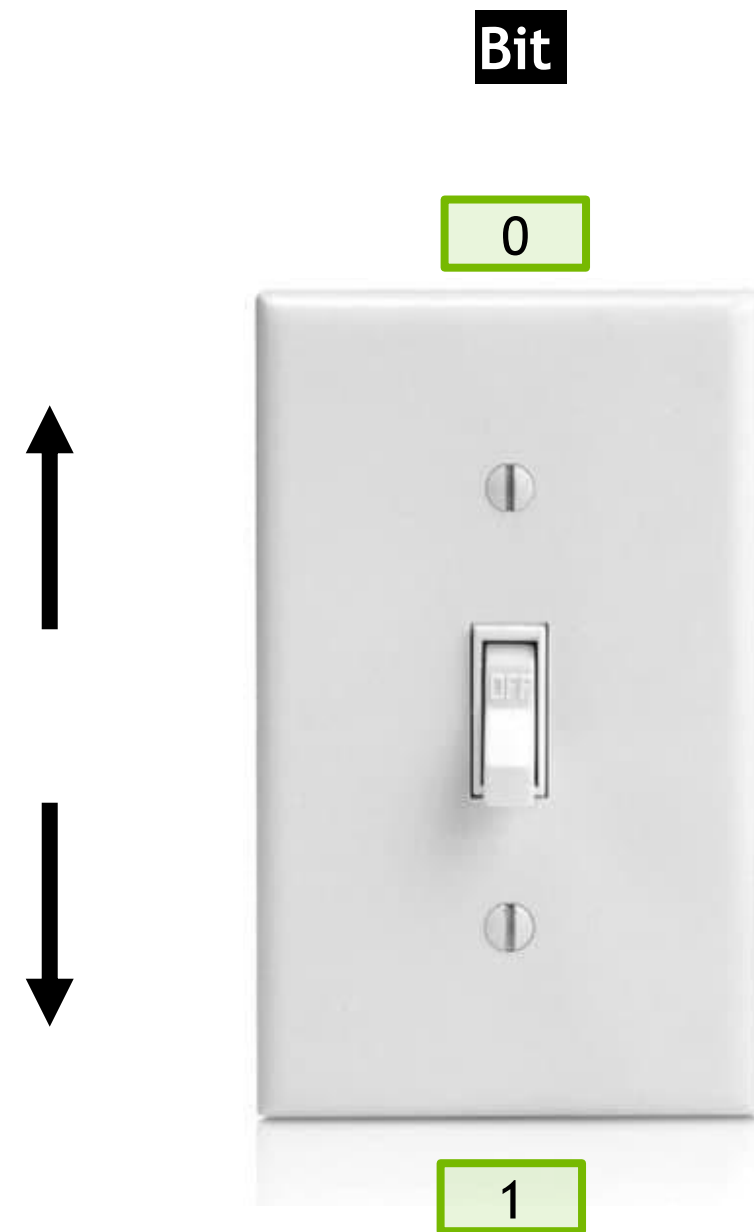
**CUDA Quantum**

Quantum-Classical Developer Platform

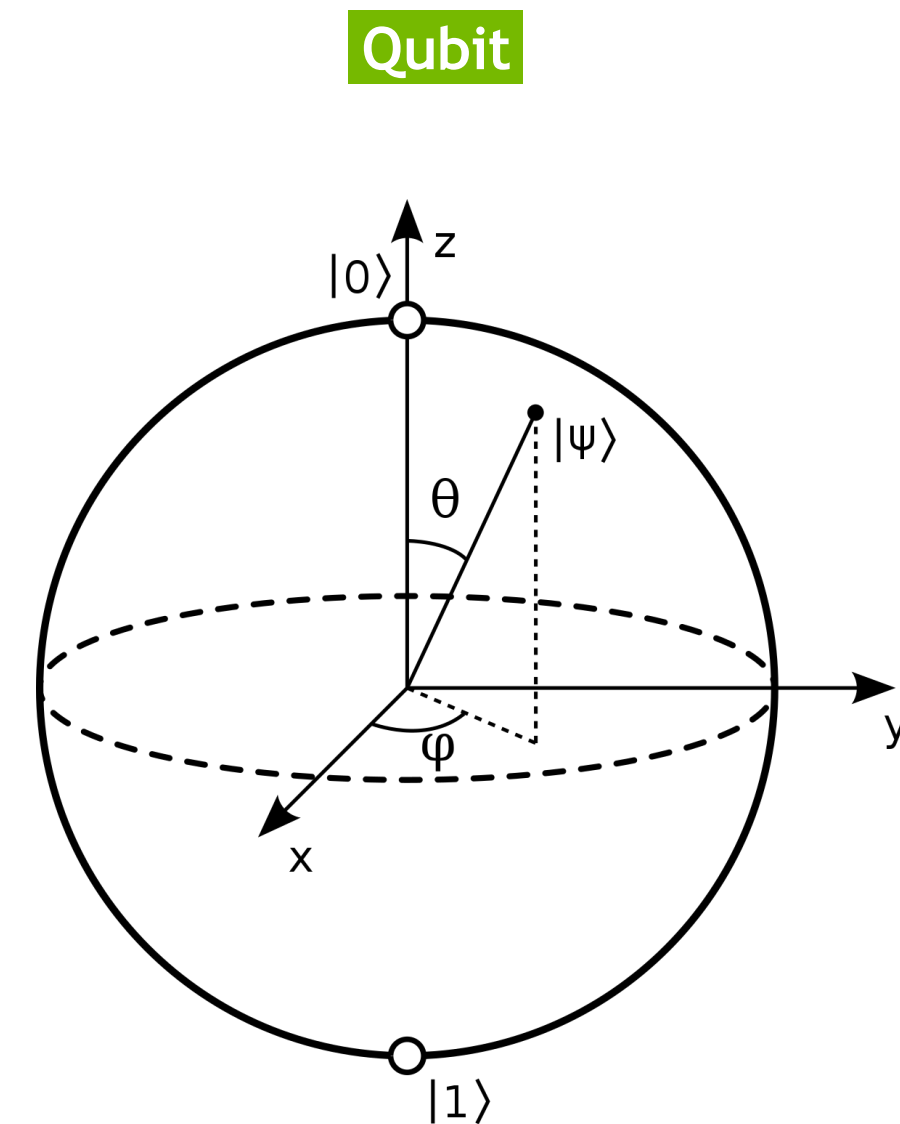
Quantum Integrated GPU Supercomputing  
DGX | HGX | **DGX Quantum**

# Qubits and Superposition

Single qubit is more informative than a classical bit



Either 0 or 1



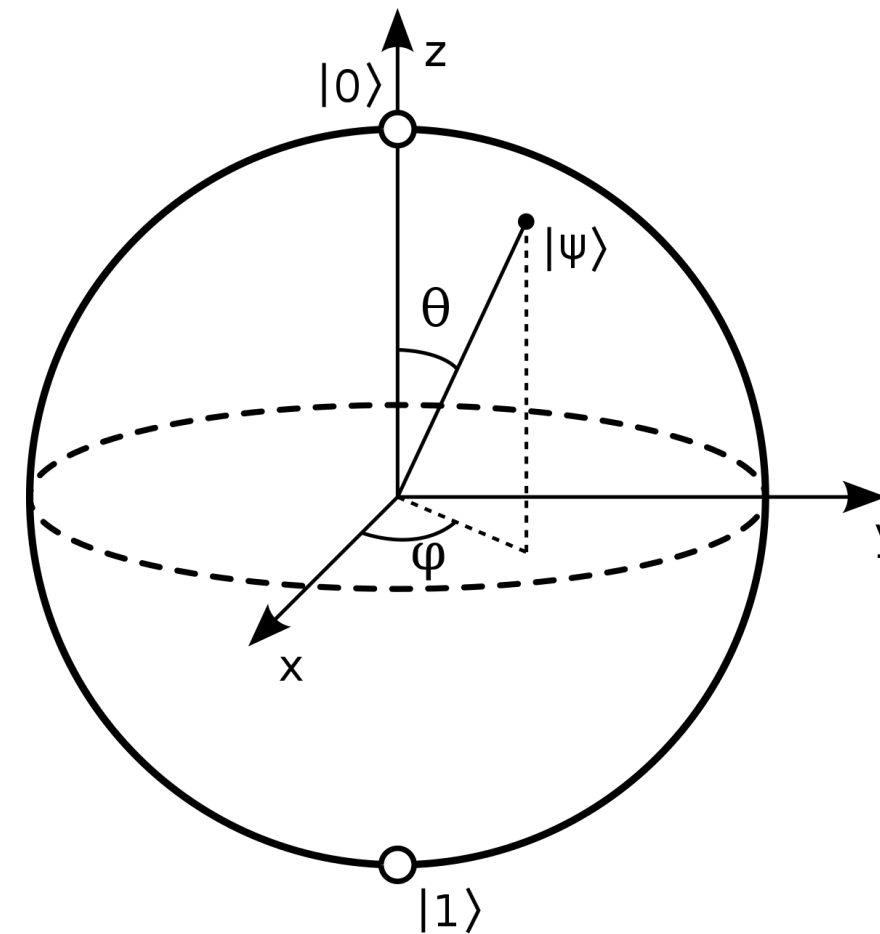
Bloch sphere

$$|\Psi\rangle = \cos\theta |0\rangle + e^{i\varphi} \sin\theta |1\rangle$$

Superposition of both 0 and 1

# Qubits and Superposition

Multiple qubits hold remarkably rich information



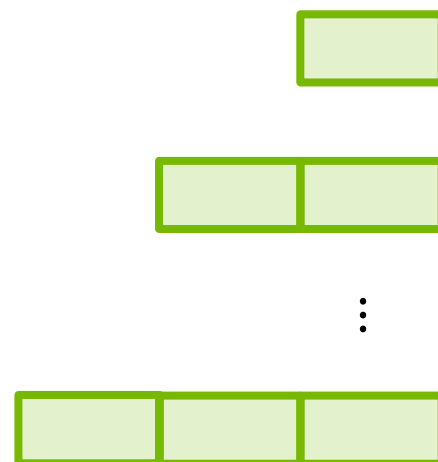
$$c_0|0\rangle + c_1|1\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$$

$$c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle = \begin{bmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{bmatrix}$$

$$c_{000}|000\rangle + c_{001}|001\rangle + c_{010}|010\rangle + c_{011}|011\rangle + c_{100}|100\rangle + c_{101}|101\rangle + c_{110}|110\rangle + c_{111}|111\rangle$$

$$= \begin{bmatrix} c_{000} \\ c_{001} \\ c_{010} \\ c_{011} \\ c_{100} \\ c_{101} \\ c_{110} \\ c_{111} \end{bmatrix}$$

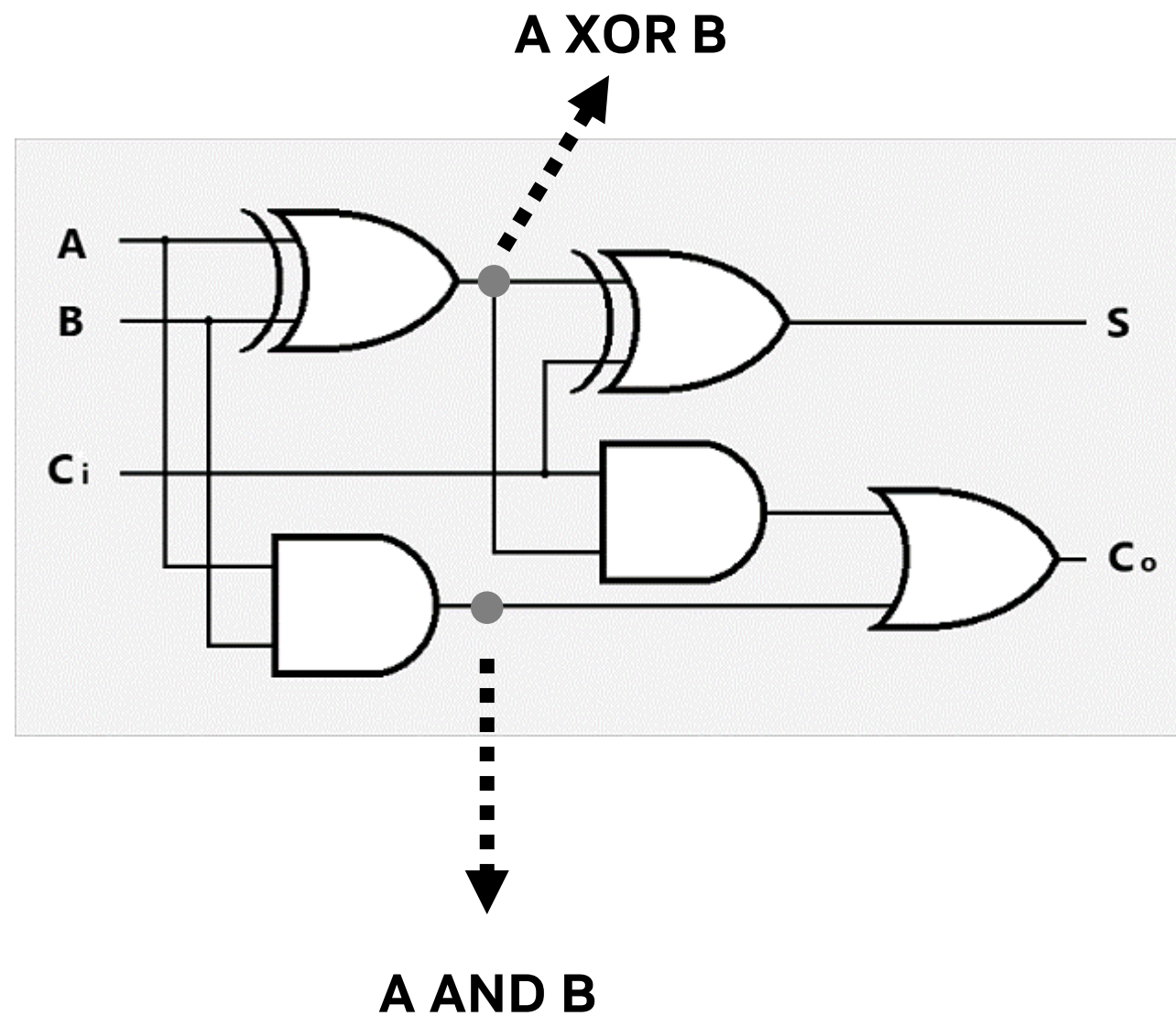
N qubits can represent  $2^N$  states simultaneously



# Quantum Parallelism and Entanglement

## Classical Circuit

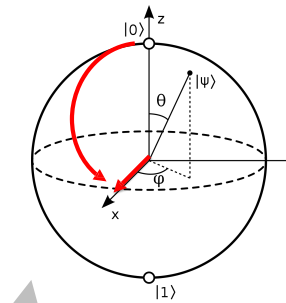
Classical logic gates (**AND**, **OR**, **XOR**...) can only perform bit-wise operation on certain states



## Quantum Circuit

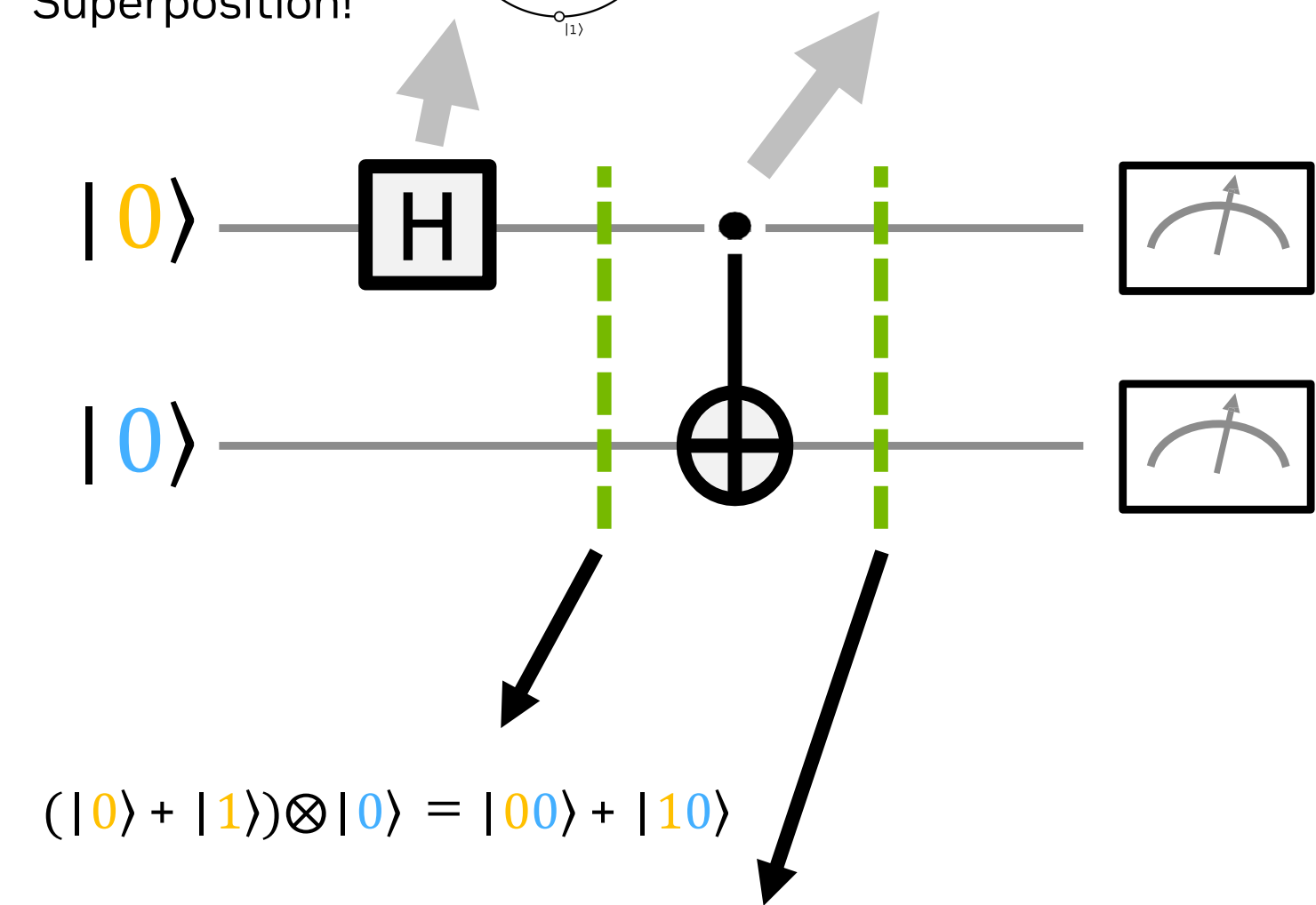
### Hadamard Gate

$H|0\rangle \rightarrow |0\rangle + |1\rangle$   
Superposition!



### CNOT Gate:

Flip **2<sup>nd</sup>** qubit if **1<sup>st</sup>** qubit is  $|1\rangle$



$$(|0\rangle + |1\rangle) \otimes |0\rangle = |00\rangle + |10\rangle$$

$$\text{CNOT}(|00\rangle + |10\rangle) = \text{CNOT}|00\rangle + \text{CNOT}|10\rangle = |00\rangle + |11\rangle$$

**Quantum Parallelism:**  
Quantum gate acts on  
superposition states

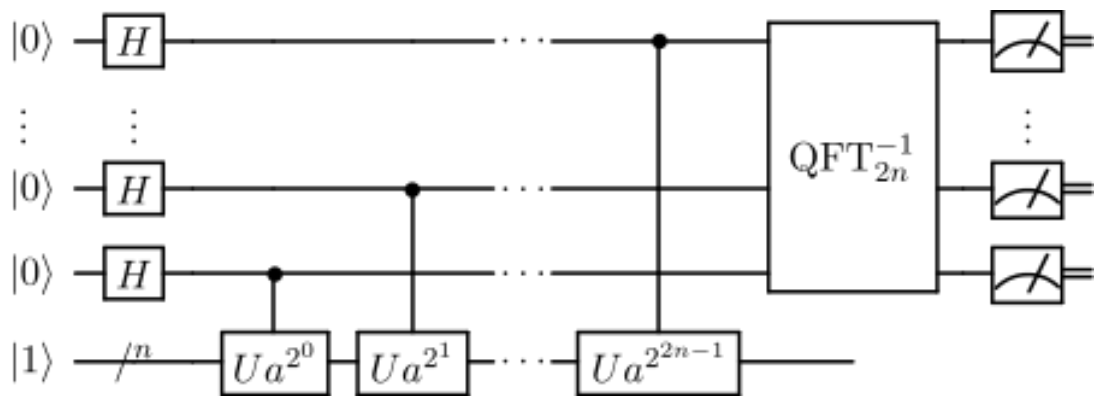
**Entanglement**

# Far-term Applications

Rigorous proofs of advantage, many “perfect” qubits required

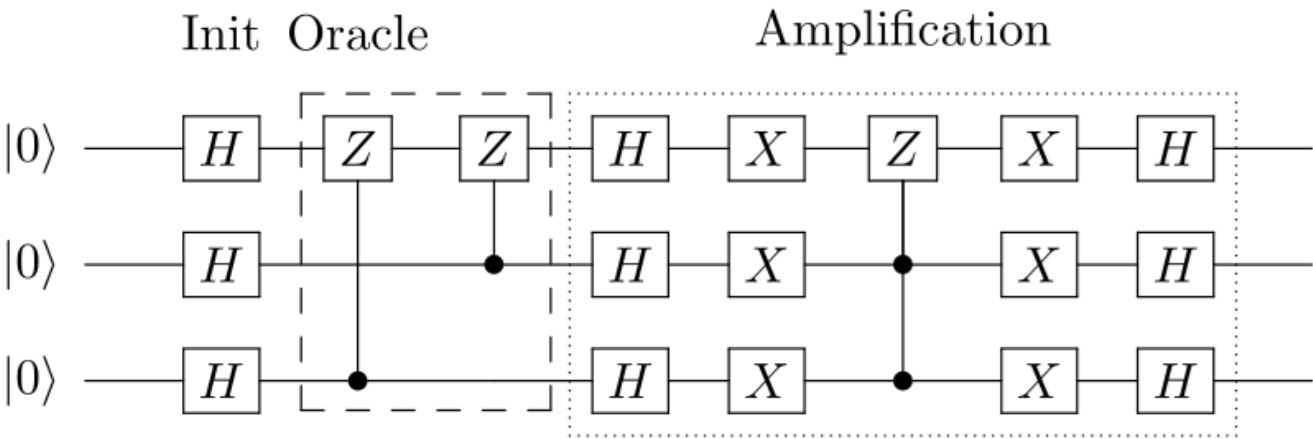
## Shor’s Algorithm

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



## Grover’s Algorithm

- Unstructured search
- Quadratic speed-up



### Linear Search



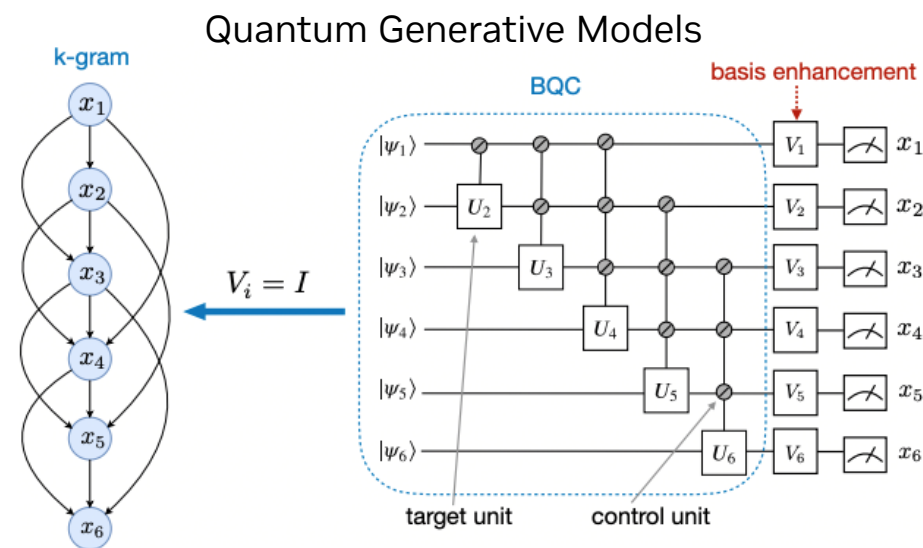
**Classical computation** requires **millions of years** to crack 2048-bit RSA encryption  
**A fault-tolerant quantum computer** can take only **a few hours**



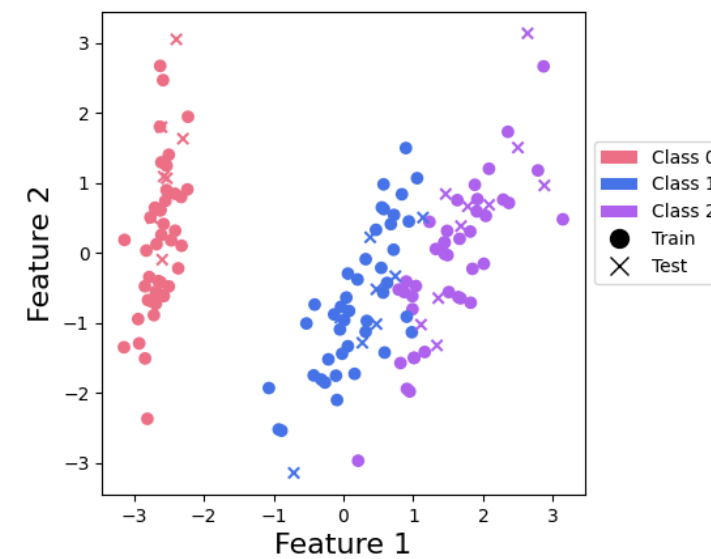
# Potential Near-term Quantum Use-cases

Applications with near-term potential, but quantum advantage is an open question

## Quantum Machine Learning



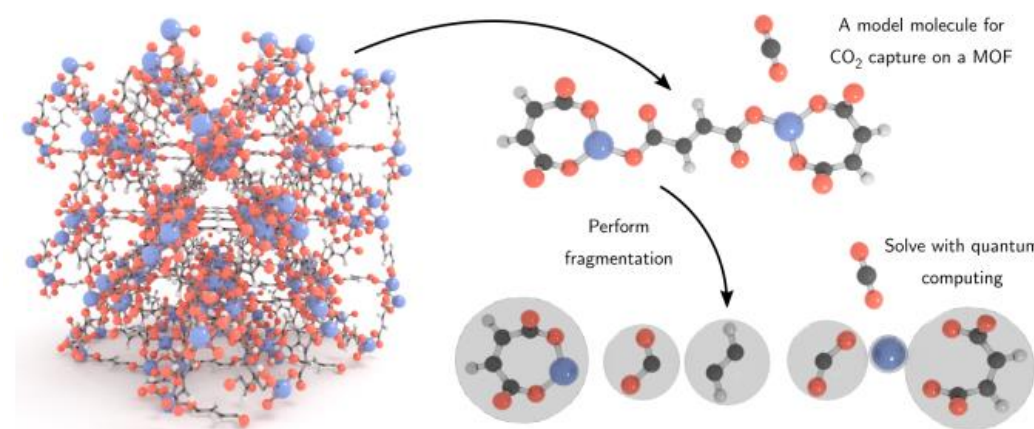
## Quantum Support Vector Machine



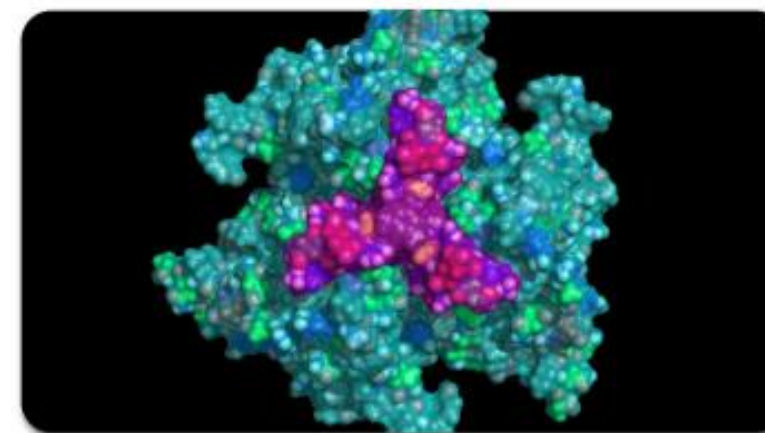
Gao, et al, Phys. Rev. X **12**, 021037  
Pennylane.ai

## Quantum Chemistry

### Variational Quantum Eigensolver for carbon capture



### Protein folding



Greene-Diniz, et al, arXiv:2203.15546,  
Menten.ai

## Combinatorial Optimization

### QAOA for resource allocation



### Logistics optimization

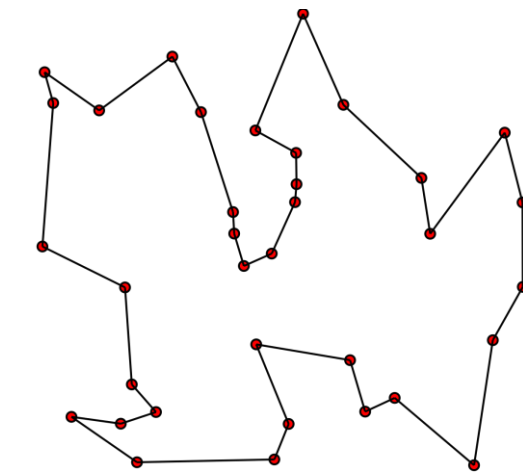


Image from ibm.com  
Wikipedia.com

# Power of Quantum Property

In quantum machine learning, native superposition can efficiently represent information

**Superposition** allows exponential information encoding:

$$\mathbf{x} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \end{bmatrix} \quad \longrightarrow \quad |\mathbf{x}\rangle = \frac{1}{2} \underline{|00\rangle} + \frac{1}{2} \underline{|01\rangle} - \frac{1}{2} \underline{|10\rangle} - \frac{1}{2} \underline{|11\rangle}$$

Vector of length 4 requires only 2 qubits

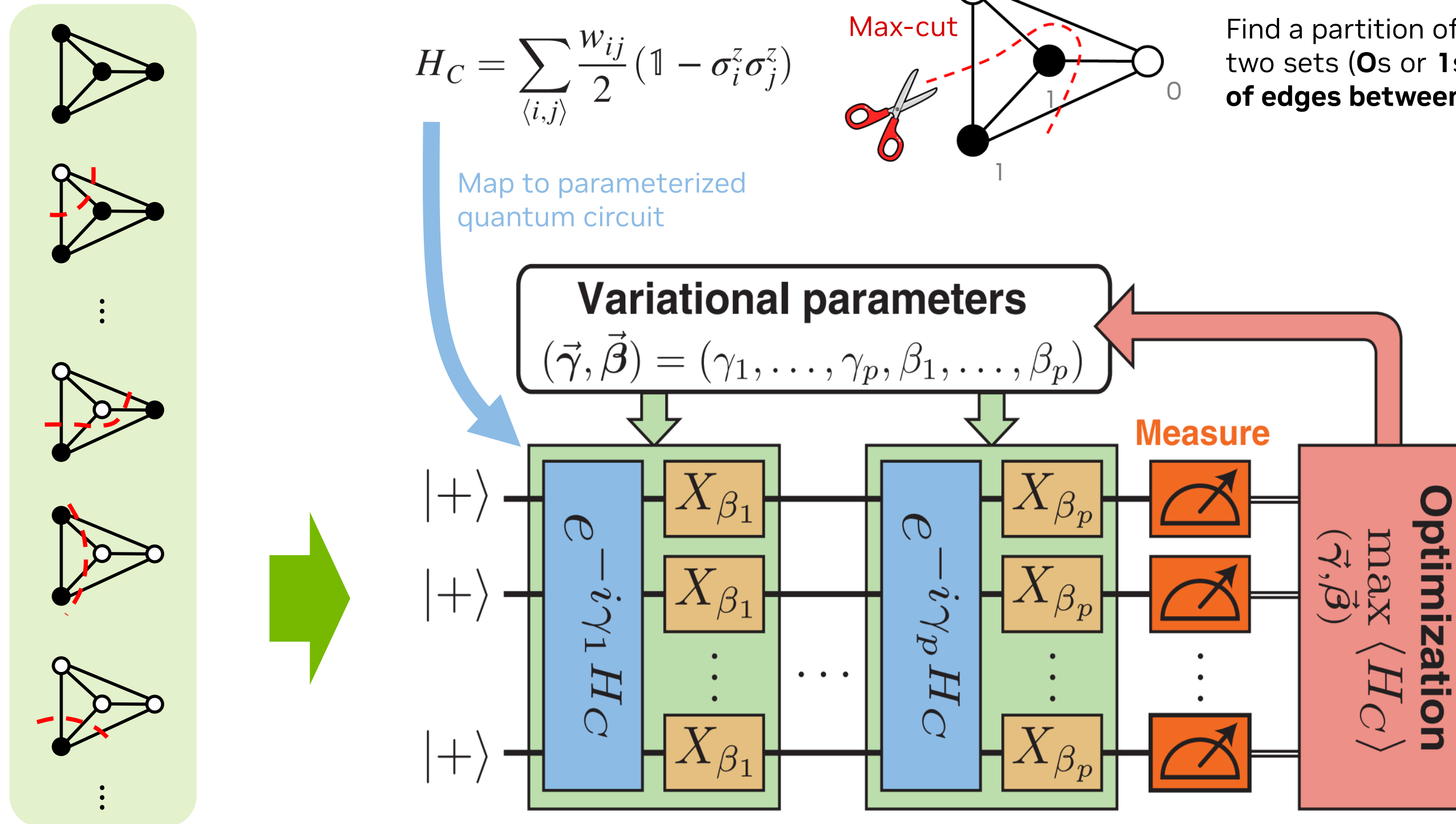
Amplitude encoding maps an **N-dimensional vector** into amplitudes of an **n-qubit quantum state** with  $n=\lceil \log_2(N) \rceil$ , which is suitable for data embedding in quantum machine learning





# Power of Quantum Property

In optimization problem, solutions can be computed in parallel via superposition

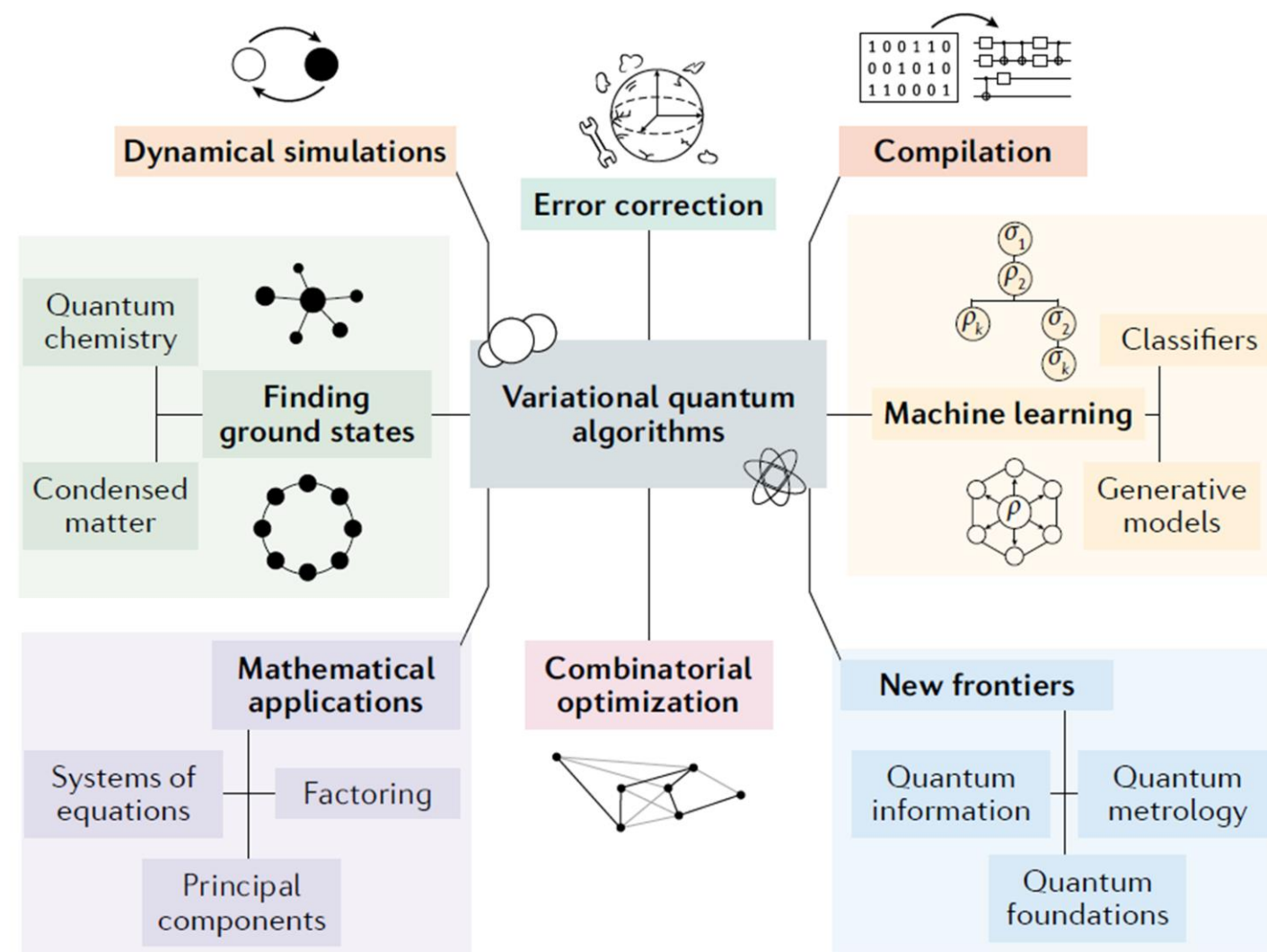


Zhou, et al. Phys. Rev. X 10.2: 021067.

# Potential Near-term Quantum Use-cases

Near-term potential applications with potential quantum advantage

- Quantum-classical computing
- Variational quantum algorithms



Cerezo, Marco, et al., Nat. Rev. Phys., 3.9, 625-644.

- Can be implemented in NISQ era and validated by simulations
- Need **workload management** across QPU/GPU/CPU

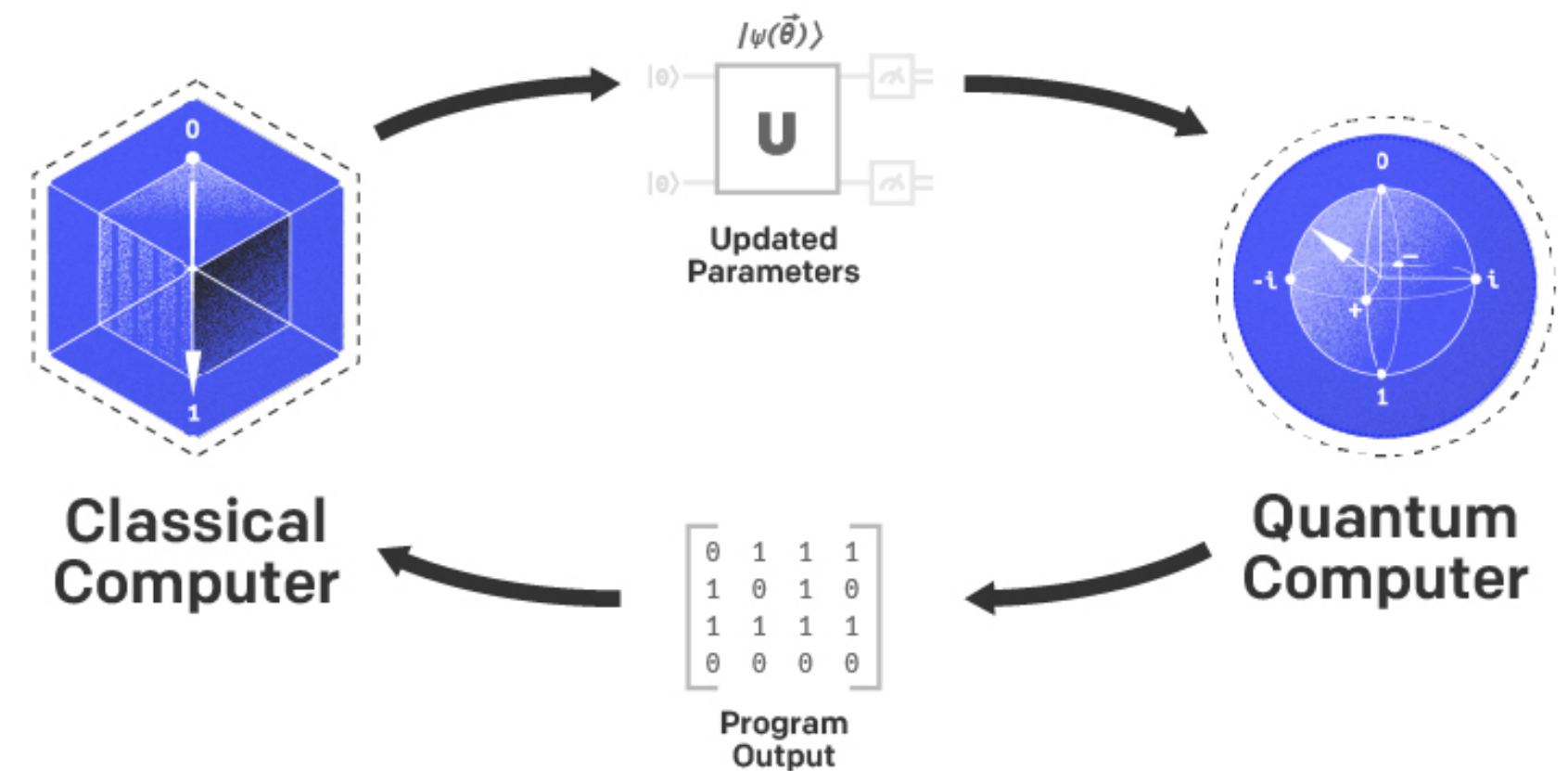


Image from ionq.com

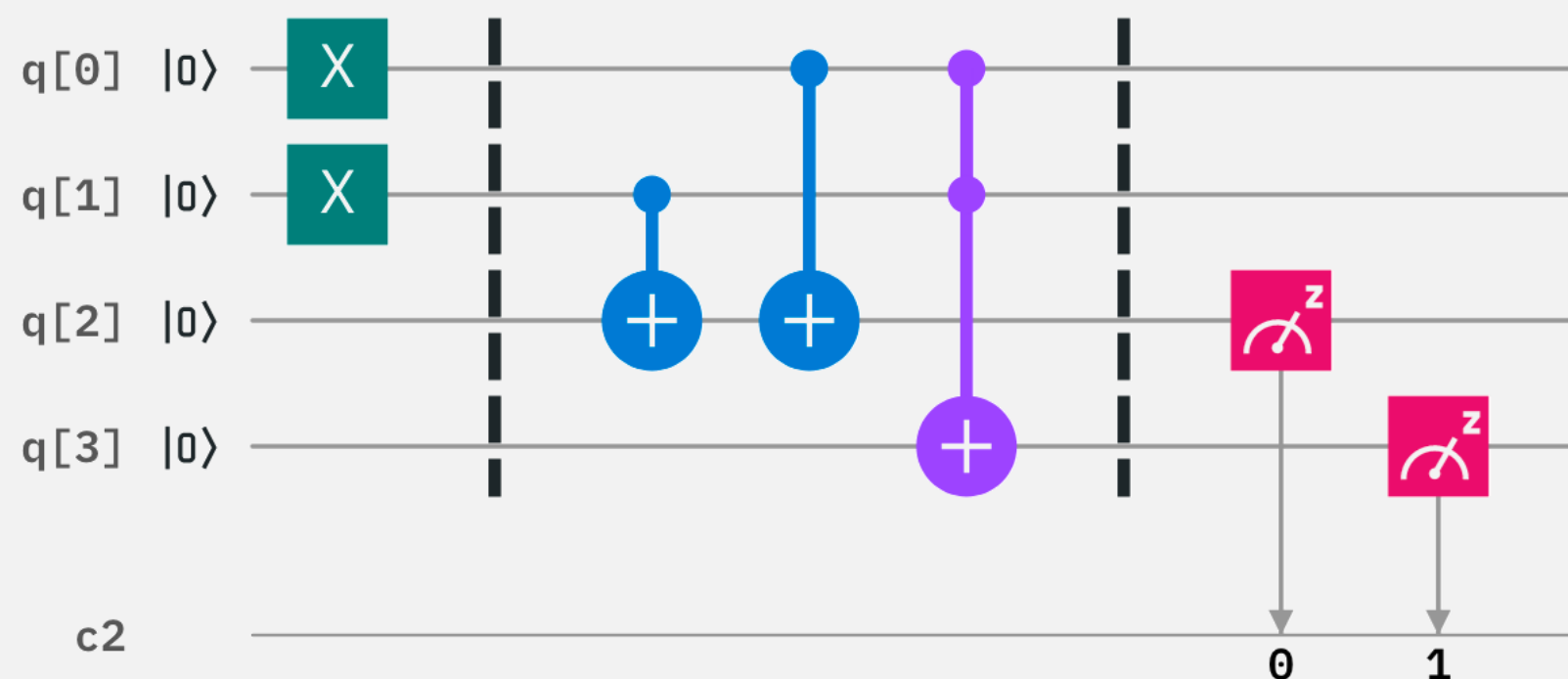
# GPU-based Supercomputing in the Quantum Computing Ecosystem

Researching the quantum computers of tomorrow with the supercomputers of today

## cuQuantum

### LOW-LEVEL LIBRARY FOR 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?

## CUDA Quantum

### PROGRAMMING MODEL FOR HYBRID CLASSICAL/QUANTUM APPLICATIONS

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



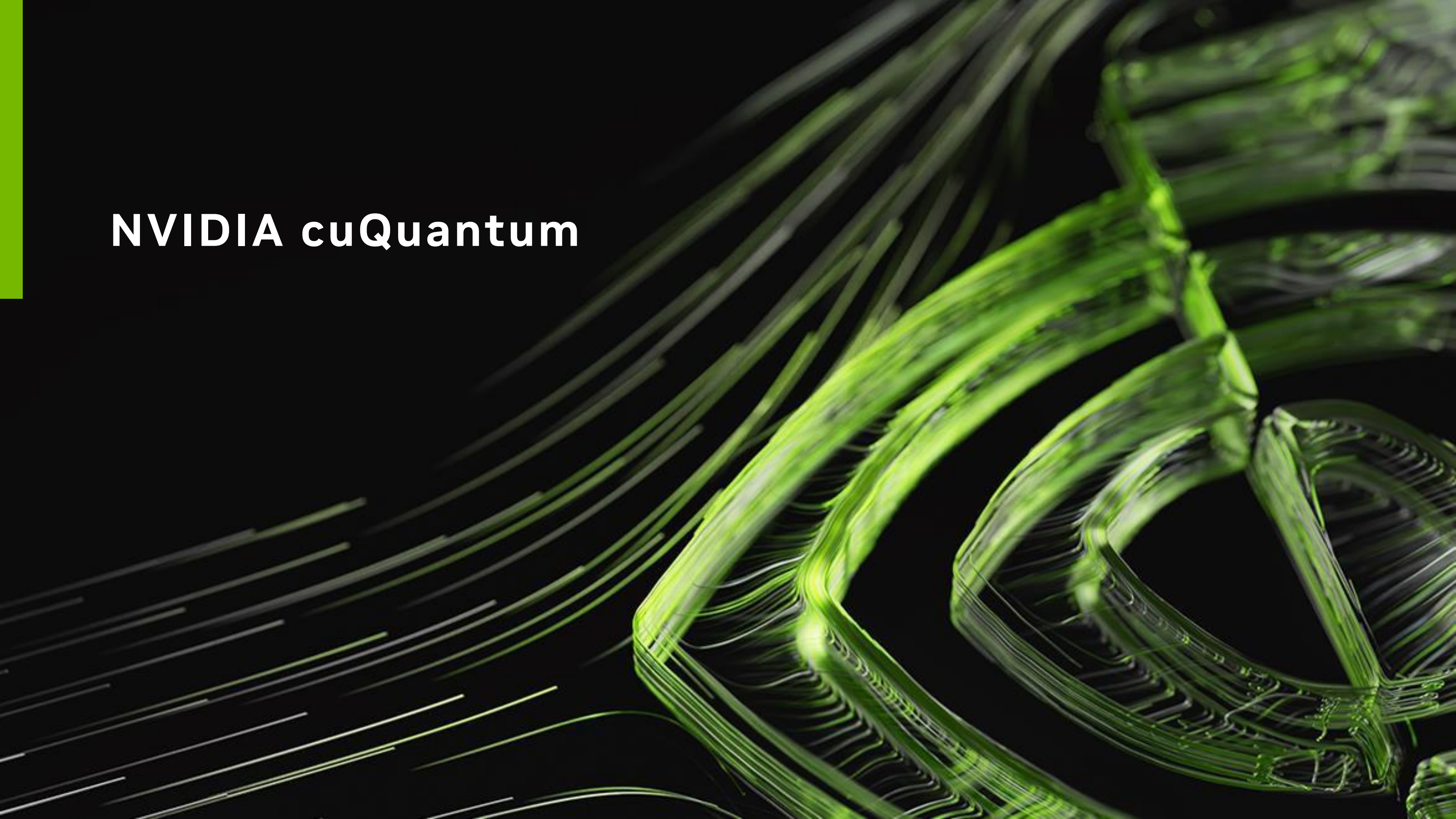
+



- 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 coprogramming of QPUs with classical HPC systems?
- Can we take advantage of GPU acceleration for circuit synthesis, classical optimization, and error correction decoding?



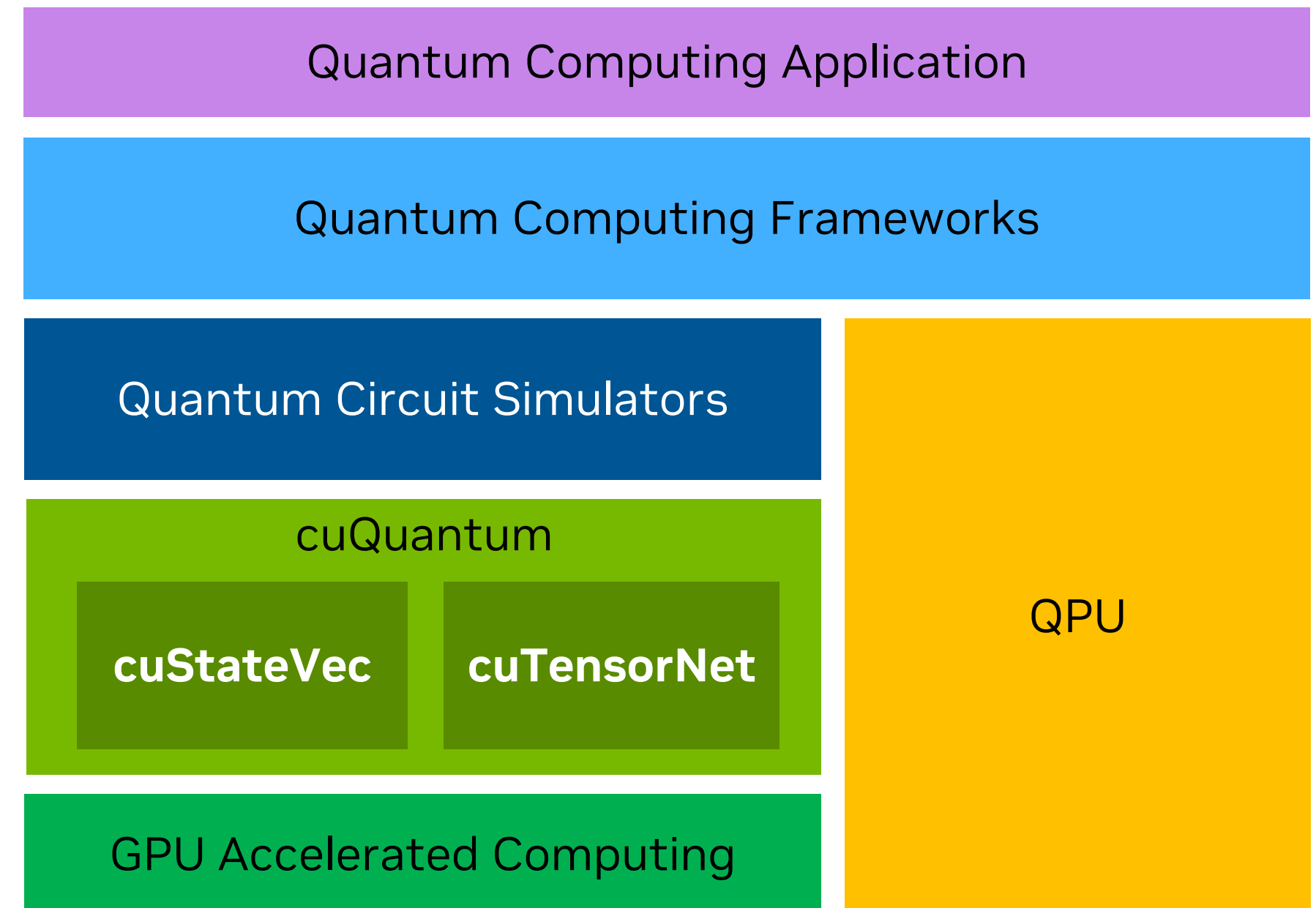
**NVIDIA cuQuantum**





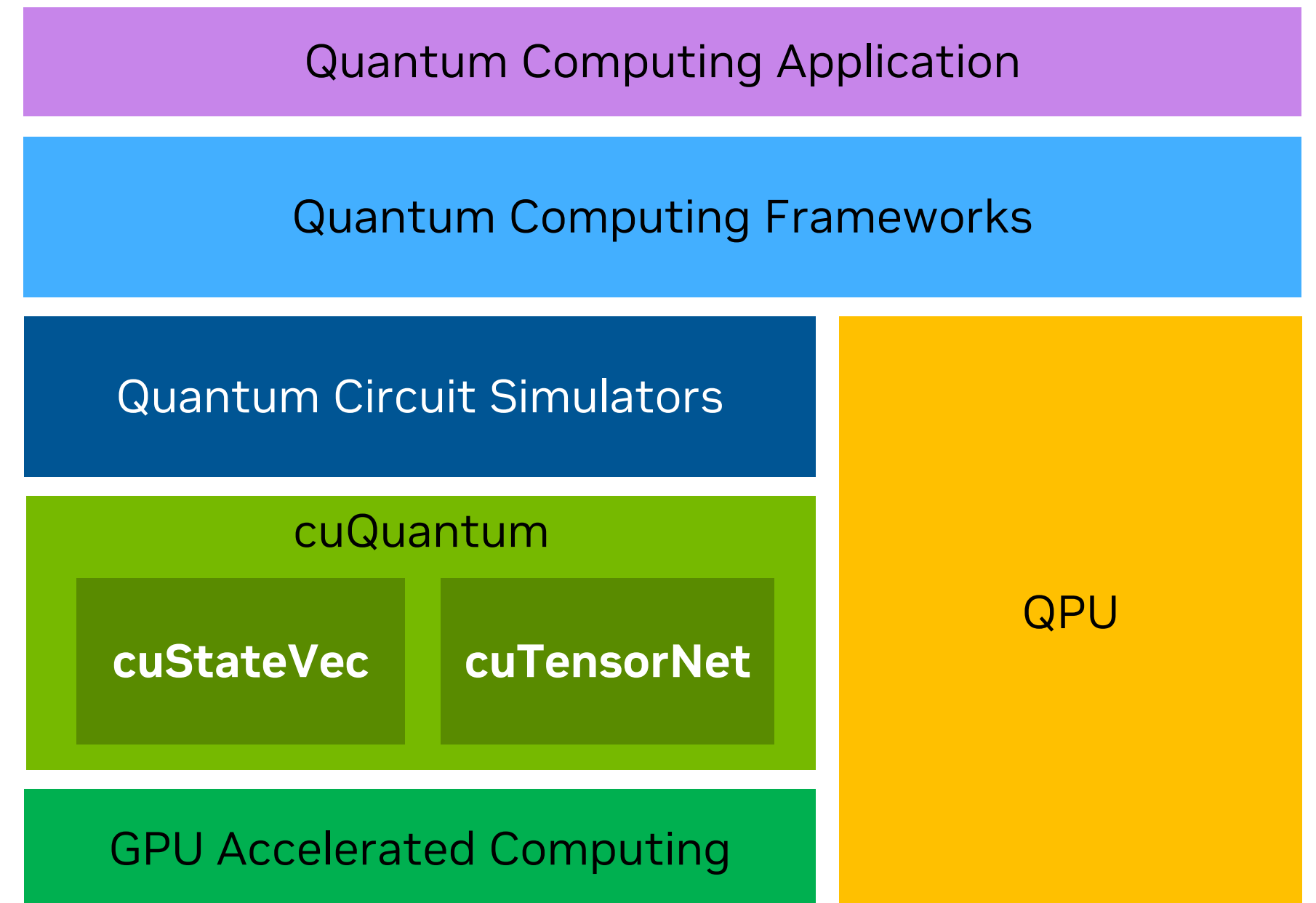
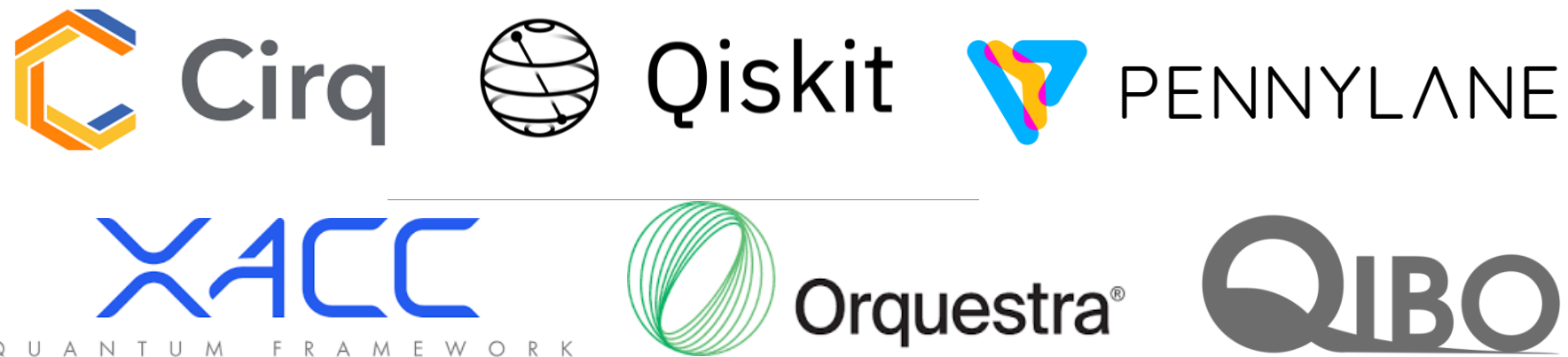
# cuQuantum Overview

- **cuQuantum is an SDK of optimized libraries and tools** for accelerating quantum computing workflows
- **cuQuantum is not a:**
  - Quantum Computer
  - Quantum Computing Framework
  - Quantum Circuit Simulator
- **cuQuantum is a collection of software libraries supporting simulator in existing frameworks to speedup quantum simulation**



# cuQuantum Overview

- **Platform for quantum computing research**
  - Accelerate Quantum Circuit Simulators on GPUs
  - Simulate ideal or noisy qubits
  - Enable algorithms research with scale and performance not possible on quantum hardware, or on simulators today
- **cuQuantum General Access available now**
  - Integrated into leading quantum computing frameworks Cirq, Qiskit, and PennyLane
  - C and Python APIs
  - Available today at [developer.nvidia.com/cuquantum](https://developer.nvidia.com/cuquantum)






# cuQuantum Overview

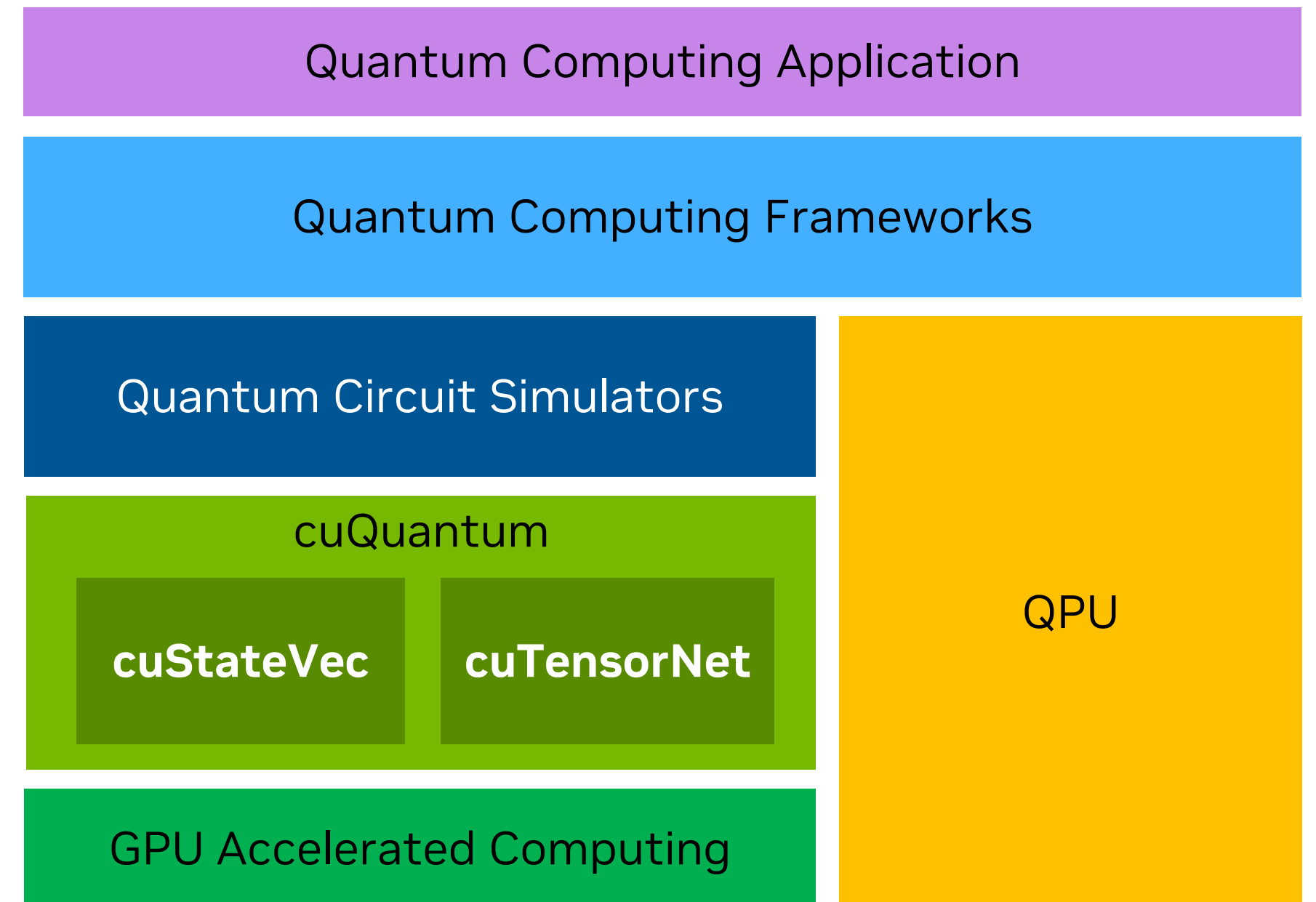
```
from qiskit import QuantumCircuit, transpile
from qiskit import Aer
from mpi4py import MPI

def run(n_qubits, use_cusvaer):
    simulator = Aer.get_backend('aer simulator statevector')
    simulator.set_option('cusvaer_enable', use_cusvaer)
    circuit = customized_circuit(n_qubits)
    circuit.measure_all()
    circuit = transpile(circuit, simulator)
    job = simulator.run(circuit)
    result = job.result()

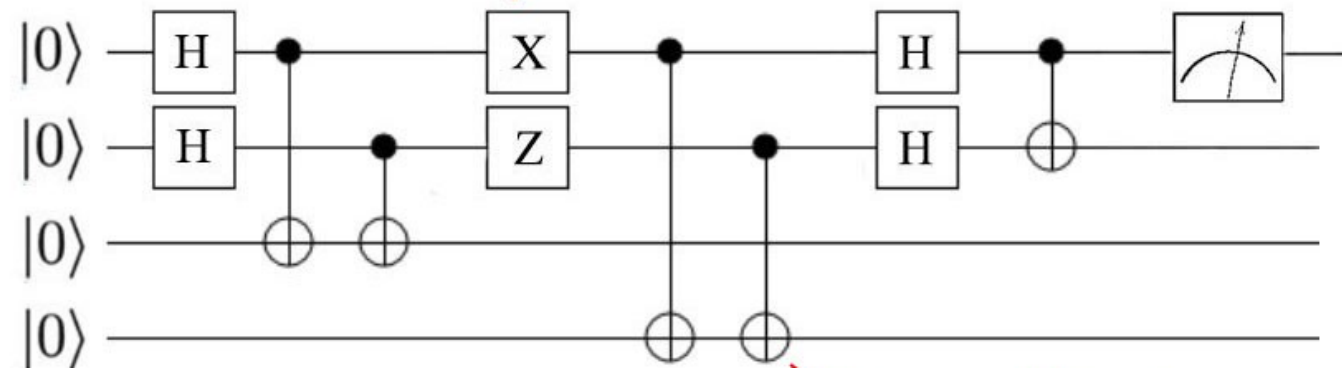
    if MPI.COMM_WORLD.Get_rank() == 0:
        print(result.get_counts())
```



- **Simple setup** → Enable cuQuantum acceleration in popular frameworks/simulators
- **MPI supported** → Allow scalable multi-GPU simulation



# Two Leading Quantum Circuit Simulation Approaches



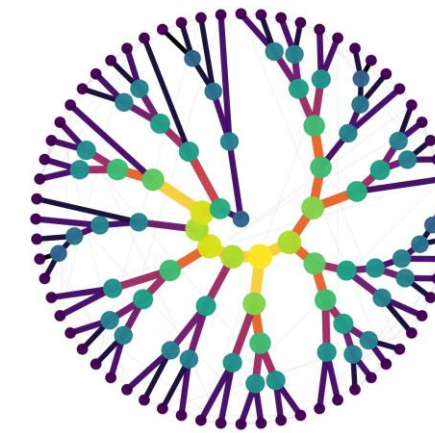
## State vector simulation

“Gate-based emulation of a quantum computer”

- Maintain full  $2^n$  qubit vector state in memory
- Update all states every timestep, probabilistically sample  $n$  of the states for measurement

Memory capacity & time grow exponentially w/ # of qubits - practical limit around 50 qubits on a supercomputer

Can model either ideal or noisy qubits



## Tensor networks

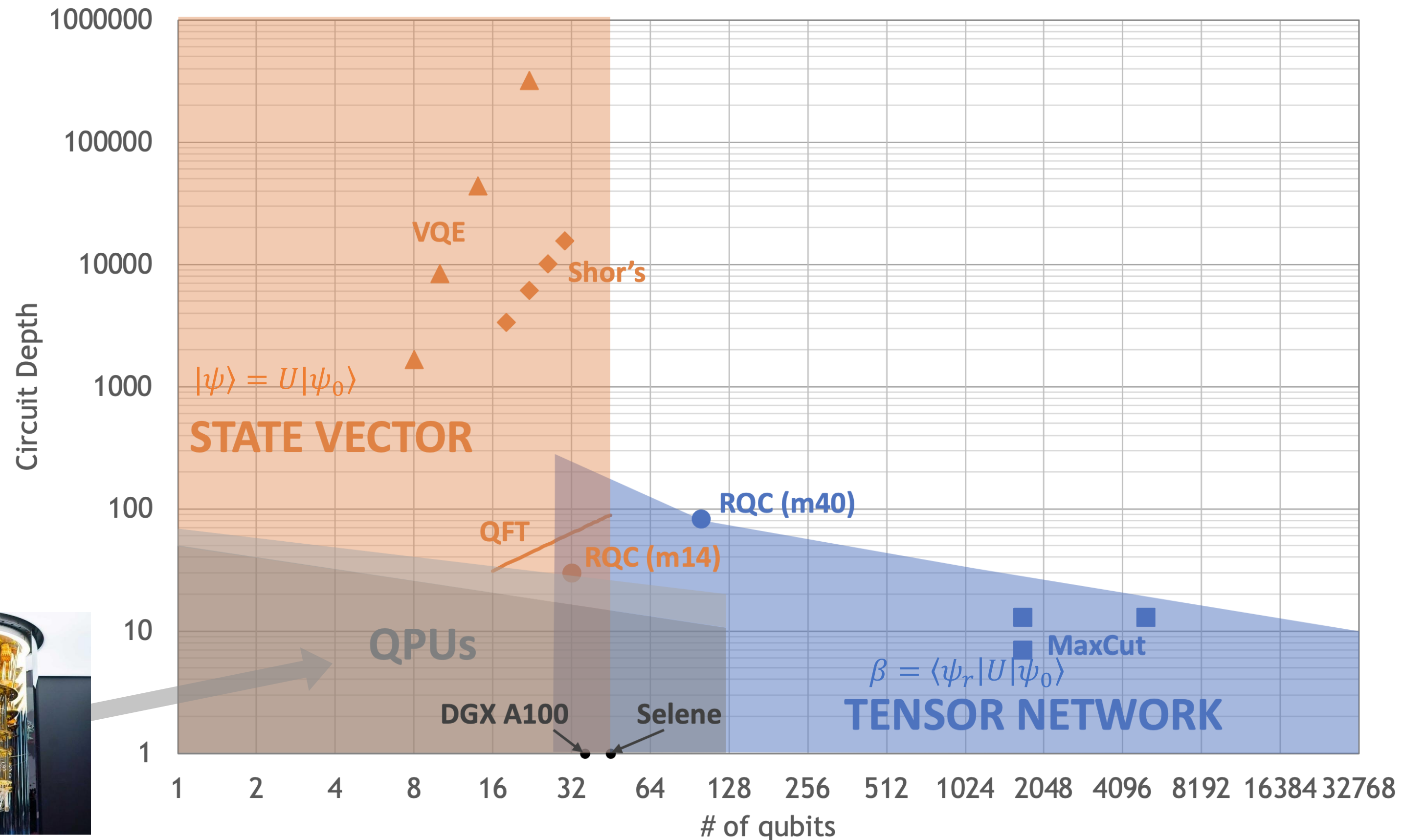
“Only simulate the states you need”

- Uses tensor network contractions to dramatically reduce memory for simulating circuits
- Can simulate 100s or 1000s of qubits for many practical quantum circuits

*GPUs are a great fit for either approach*

# Powerful Simulations with cuQuantum

Researching & Developing the Computers of Tomorrow Requires Powerful Simulations Today

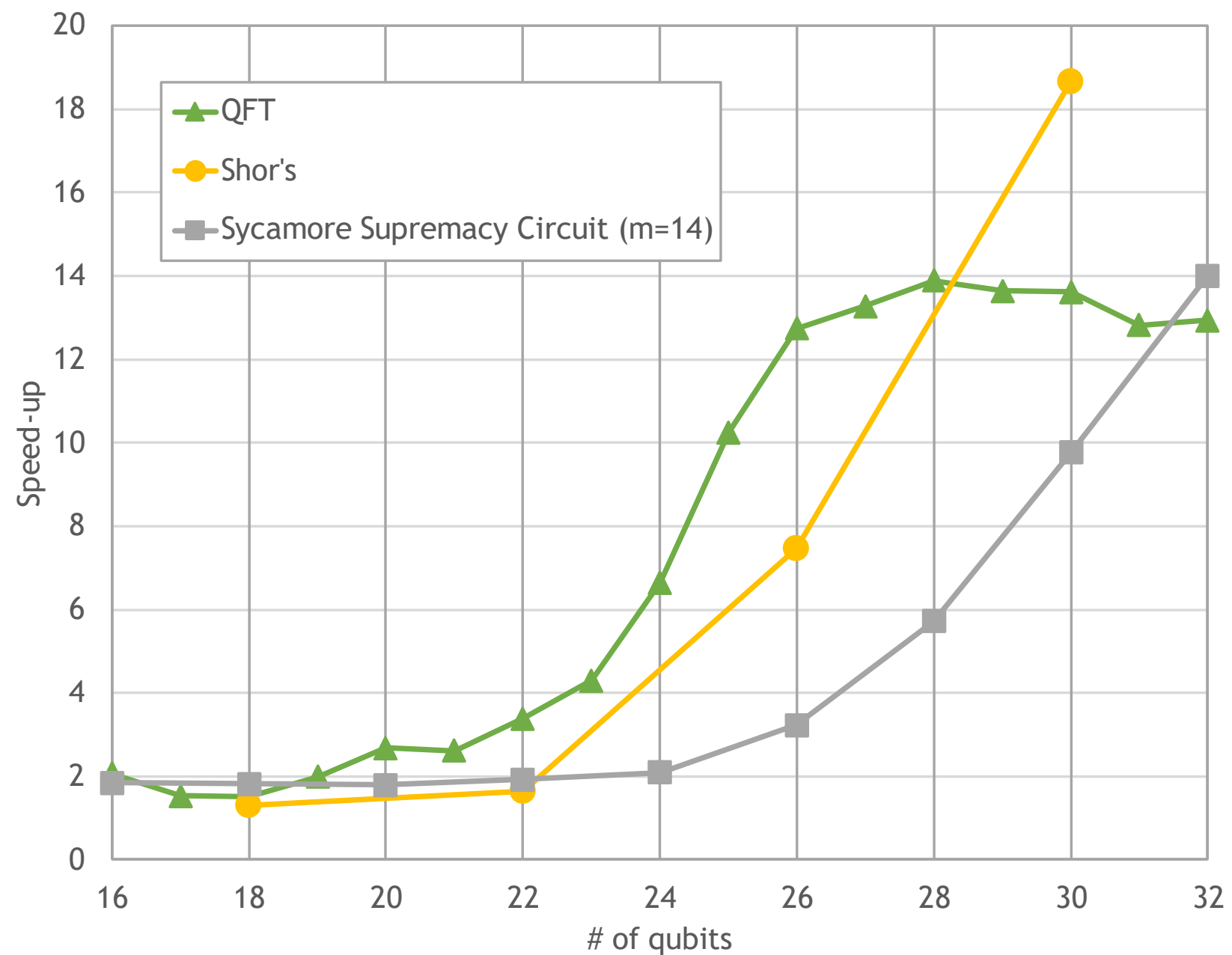




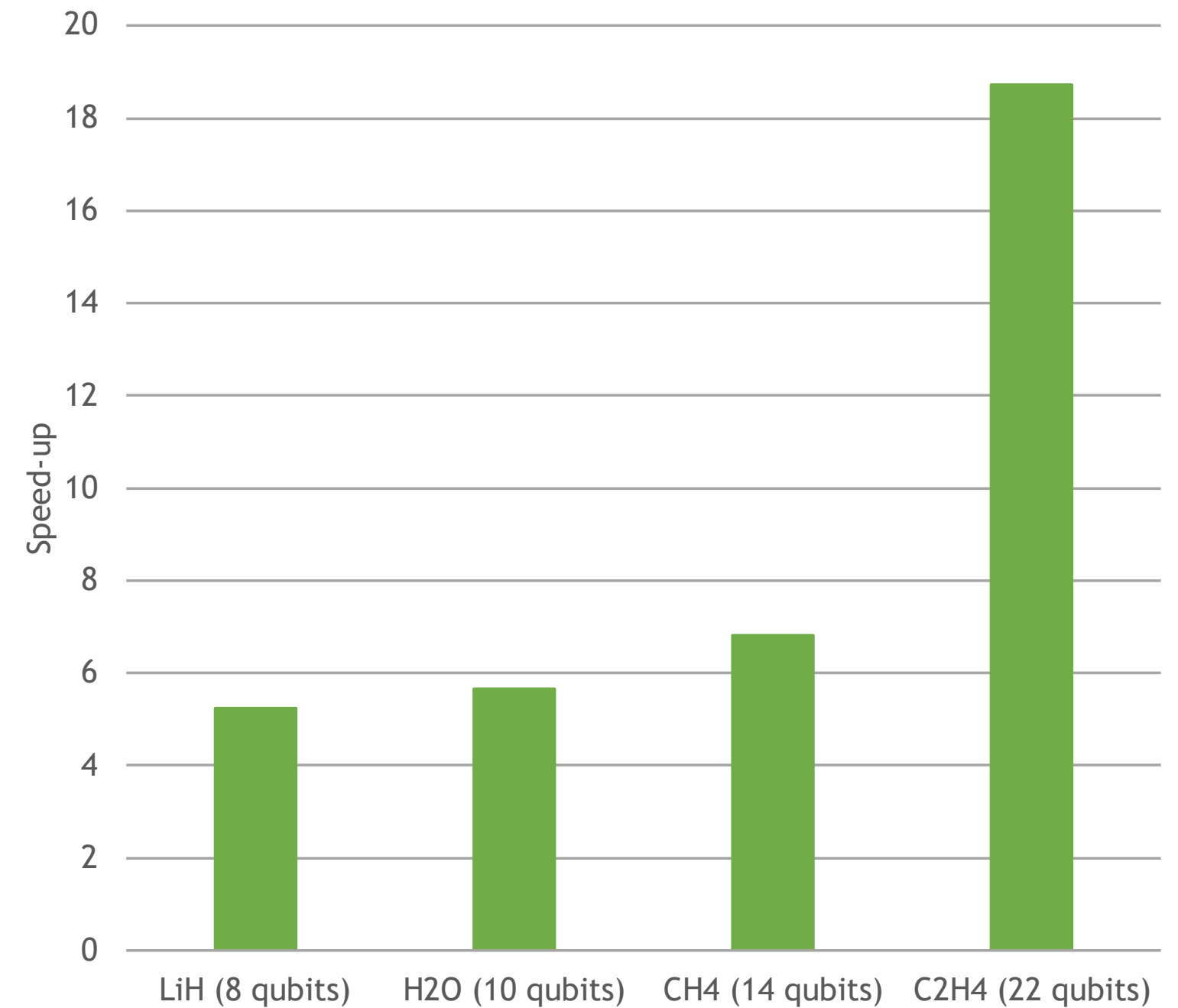
# cuStateVec : Single-GPU Performance

Preliminary performance of cirq/qsim + cuStateVec on A100

A100 80G vs 64 core CPU



VQE speed-up relative to single CPU



Benchmarks run using cirq/qsim with modifications to integrate cuStateVec  
CPUs used were AMD EPYC 7742 with 64 cores  
QFT circuit with 32 qubits and depth 63  
Shor's circuit with 30 qubit and depth 15560 (integer factorized: 65)  
Sycamore supremacy circuit m=14 with 7480 gates

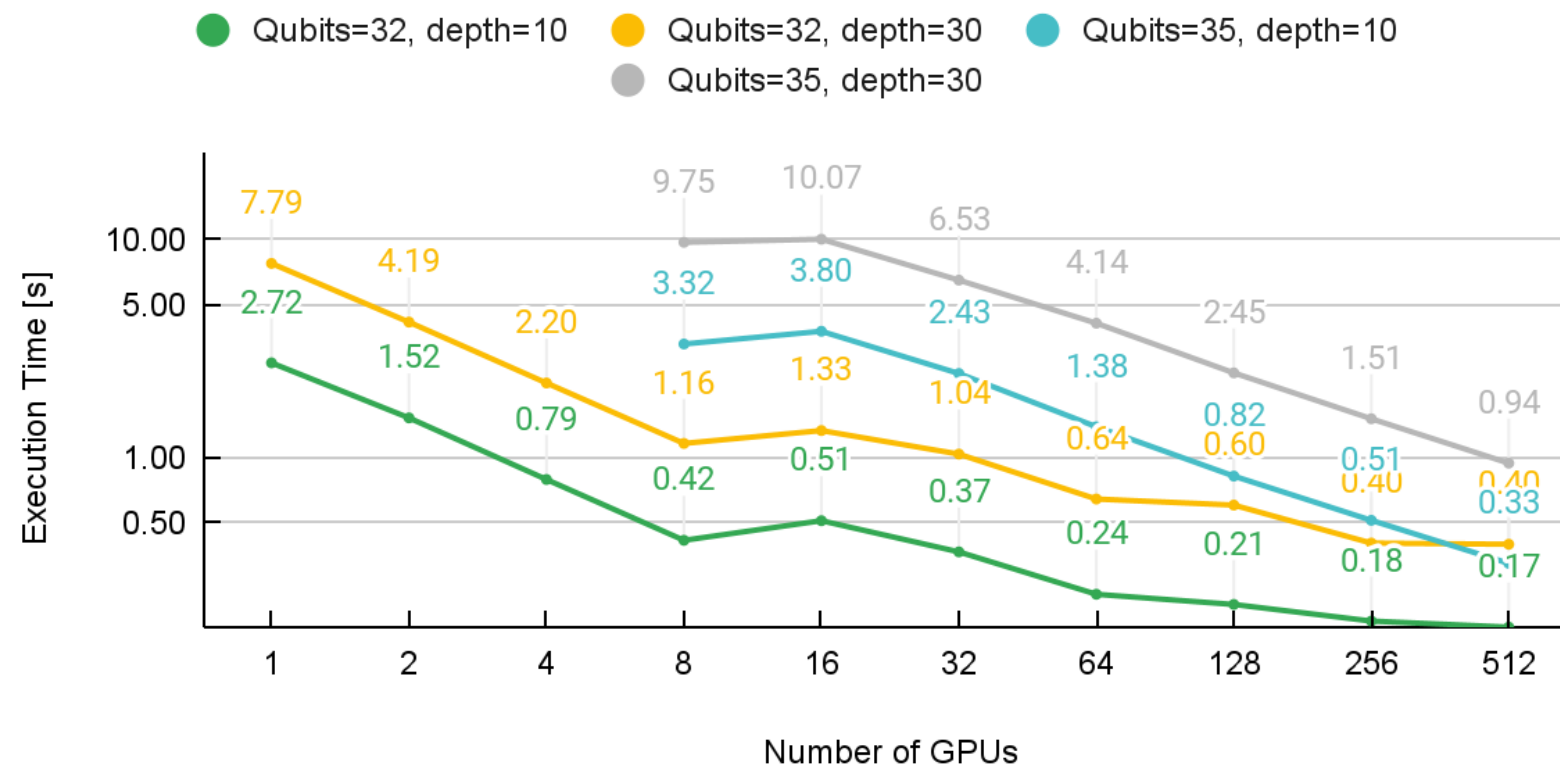
VQE benchmarks have all orbitals and results were measured for the energy function evaluation

# cuStateVec : Multi-GPU Performance

Best-in-class performance on ABCI supercomputer

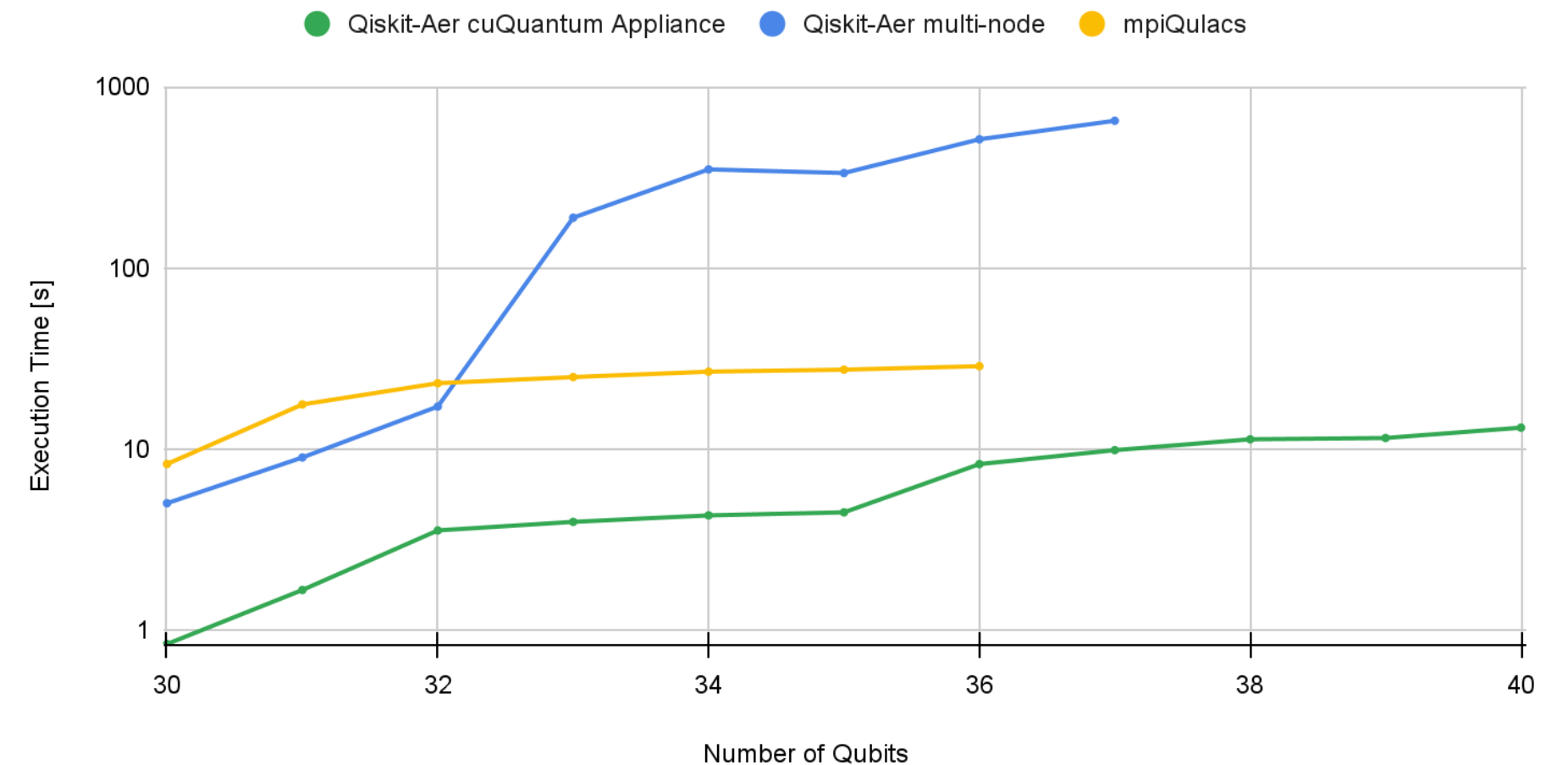
## cuQuantum Appliance Strong Scaling for Quantum Volume

Complex 64 Numbers Used



## Weak Scaling Comparison of Multi-node Simulators

Quantum Volume, Depth=10



<https://developer.nvidia.com/blog/best-in-class-quantum-circuit-simulation-at-scale-with-nvidia-cuquantum-appliance/>

- cuQuantum-integrated simulator in the popular framework (Qiskit) can further speedup
- Benchmark against the competitor (mpiQulacs)

The simulation was held to 32 and 35 qubits and distributed across 512 NVIDIA A100 40GB GPUs on the ABCI Compute Node (A)

mpiQulacs is a fast, multi-node, full-state vector quantum circuit simulator developed to run on the Fujitsu A64FX CPU architecture

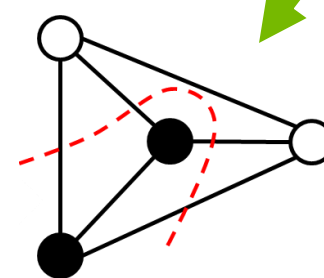
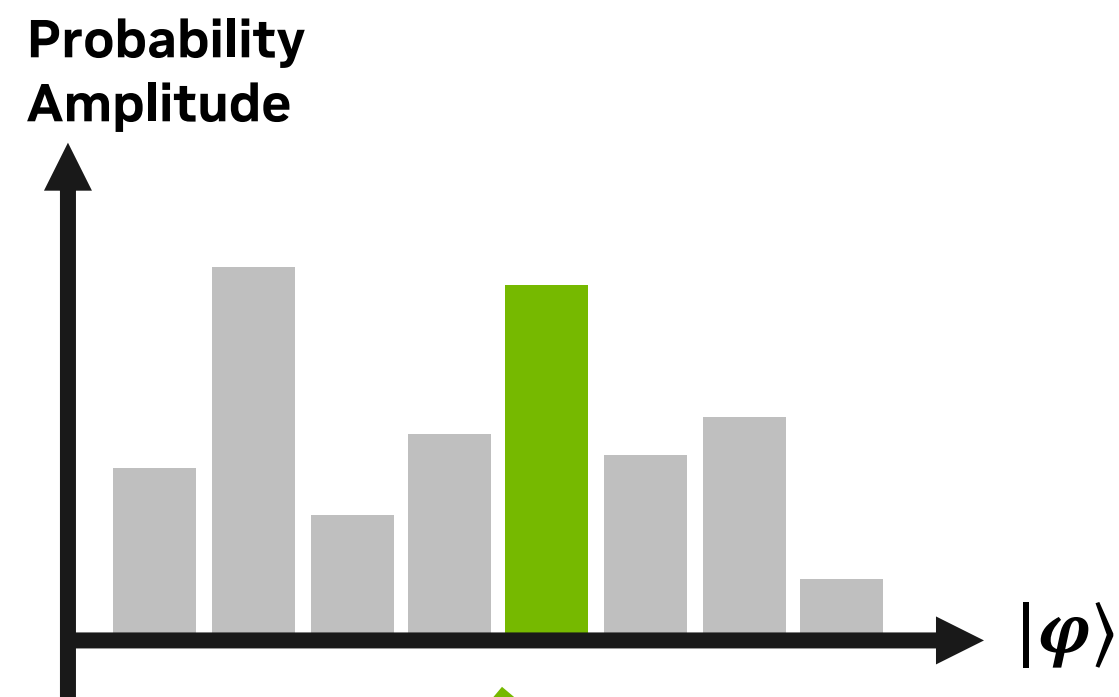
# cuTensorNet

A library to accelerate tensor-network-based quantum circuit simulation

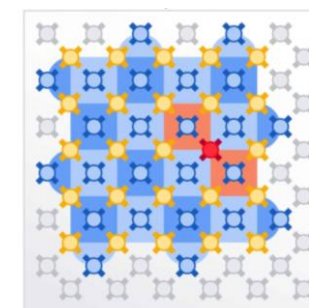
- For many practical quantum circuits, tensor networks enable scaling of simulation to 100s or 1000s of qubits
- cuTensorNet provides APIs to:
  1. Convert a circuit written in Cirq or Qiskit to a tensor network
  2. Calculate an optimal path for the contraction
    - Hyper-optimization is used to find contraction path with lowest total cost (eg, FLOPS or time estimate)
    - Slicing is introduced to create parallelism or reduce maximum intermediate tensor sizes
  3. Calculate an execution plan and execute the TN contraction
    - Leverages cuTENSOR heuristics

<https://developer.nvidia.com/blog/scaling-quantum-circuit-simulation-with-cutensornet/>

- **Only simulating the desired states.** Example scenarios:
  1. Approximation or validation of optimization algorithms
  2. Error syndrome simulation for quantum error correction



How well does the algorithm approach a certain result?



What is the probability of certain types of error?

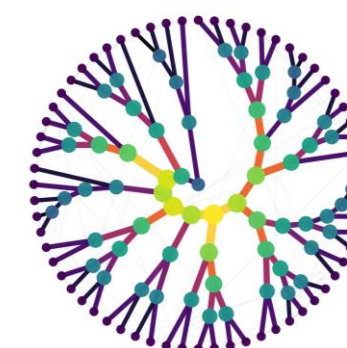
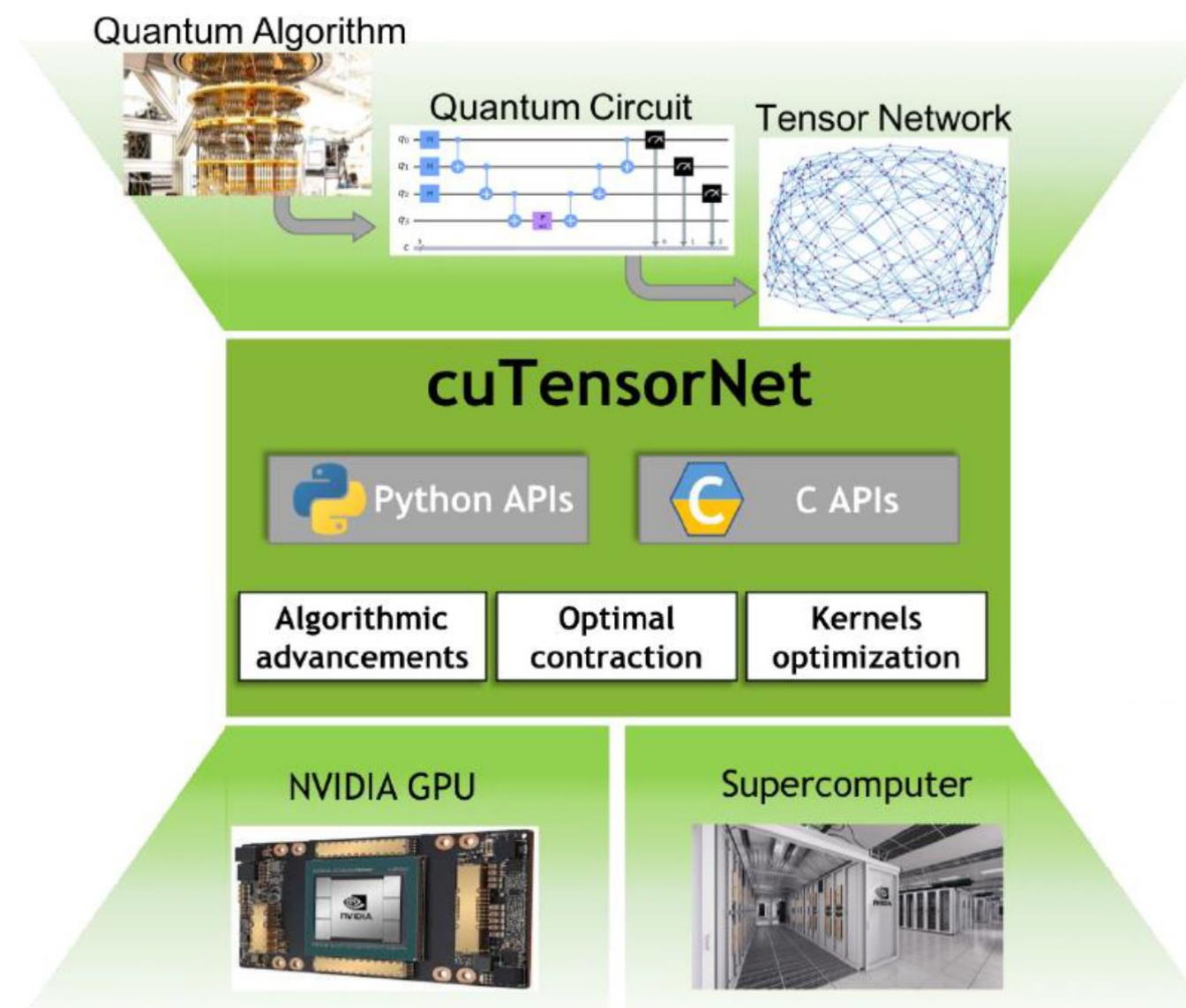
Modified from physicsworld.com



# cuTensorNet

A library to accelerate tensor-network-based quantum circuit simulation

- For many practical quantum circuits, tensor networks enable scaling of simulation to 100s or 1000s of qubits
- cuTensorNet provides APIs to:
  1. Convert a circuit written in Cirq or Qiskit to a tensor network
  2. Calculate an optimal path for the contraction
    - Hyper-optimization is used to find contraction path with lowest total cost (eg, FLOPS or time estimate)
    - Slicing is introduced to create parallelism or reduce maximum intermediate tensor sizes
  1. Calculate an execution plan and execute the TN contraction
    - Leverages cuTENSOR heuristics



<https://developer.nvidia.com/blog/scaling-quantum-circuit-simulation-with-cutensornet/>

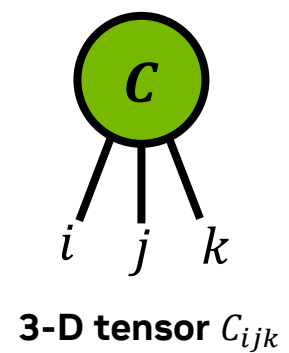
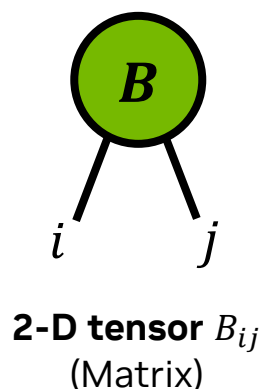
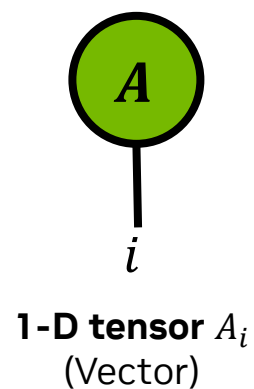
# cuTensorNet

## Going from quantum to tensor

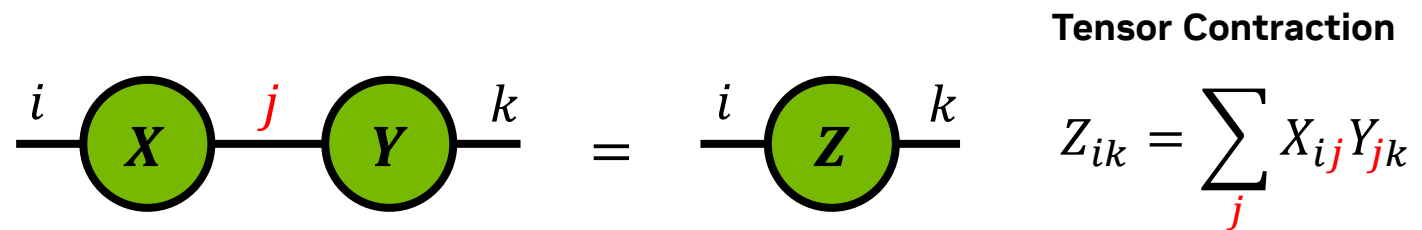
- Tensor can be interpreted as a generalization of scalars, vectors, and matrices

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & \cdots & B_{1n} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mn} \end{bmatrix} \quad C = \begin{bmatrix} \begin{bmatrix} C_{111} & \cdots & C_{1n1} \\ \vdots & \ddots & \vdots \\ C_{m11} & \cdots & C_{mn1} \end{bmatrix}^1 \\ \vdots \\ \begin{bmatrix} C_{111} & \cdots & C_{1n1} \\ \vdots & \ddots & \vdots \\ C_{m11} & \cdots & C_{mn1} \end{bmatrix}^l \end{bmatrix}^3$$

Rank-3, dimension = (m, n, 3)

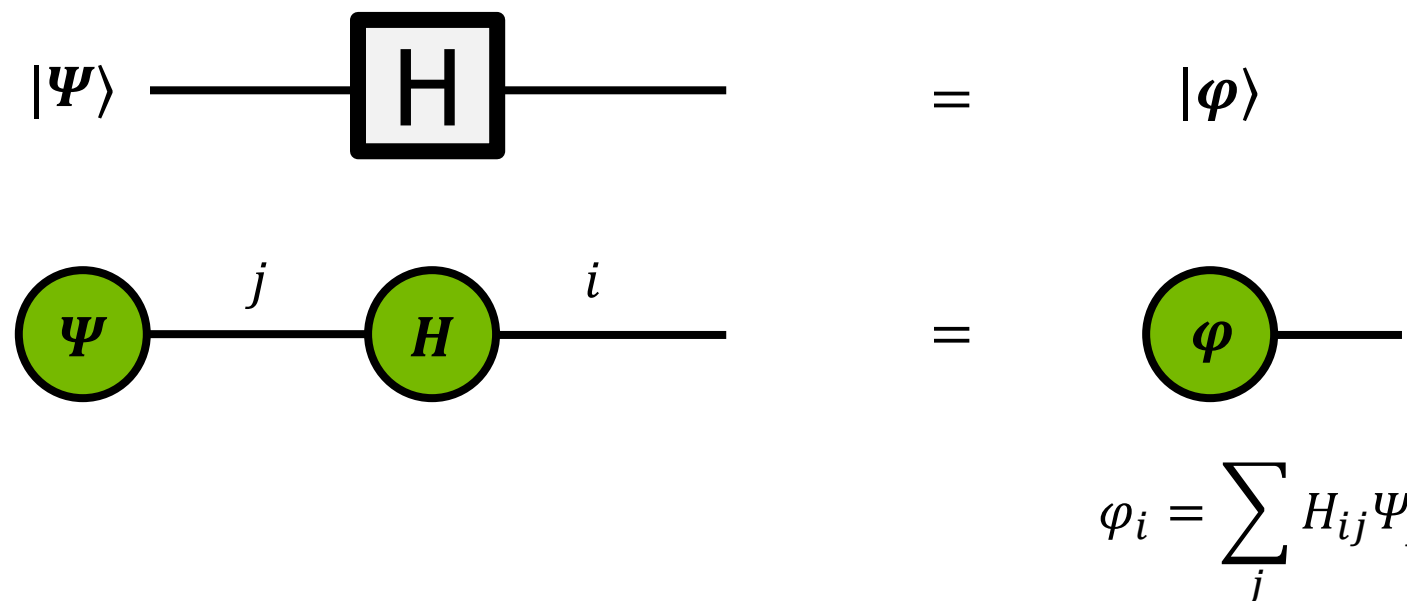


- Tensor can **contract** together



$$\mathbf{Z} = \begin{bmatrix} Z_{00} & Z_{01} \\ Z_{10} & Z_{11} \end{bmatrix} = \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} \begin{bmatrix} Y_{00} & Y_{01} \\ Y_{10} & Y_{11} \end{bmatrix} = \begin{bmatrix} X_{0\mathbf{0}}Y_{\mathbf{00}} + X_{0\mathbf{1}}Y_{\mathbf{10}} & X_{0\mathbf{0}}Y_{\mathbf{01}} + X_{0\mathbf{1}}Y_{\mathbf{11}} \\ X_{\mathbf{10}}Y_{\mathbf{00}} + X_{\mathbf{11}}Y_{\mathbf{10}} & X_{\mathbf{10}}Y_{\mathbf{01}} + X_{\mathbf{11}}Y_{\mathbf{11}} \end{bmatrix}$$

- Tensor representation of a quantum circuit



- Only the **subspaces of the tensors** need to be contracted for calculating a desired quantum state

$$|\varphi\rangle = \begin{bmatrix} \varphi_0 \\ \varphi_1 \end{bmatrix} = \begin{bmatrix} H_{00} & H_{01} \\ H_{10} & H_{11} \end{bmatrix} \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix}$$

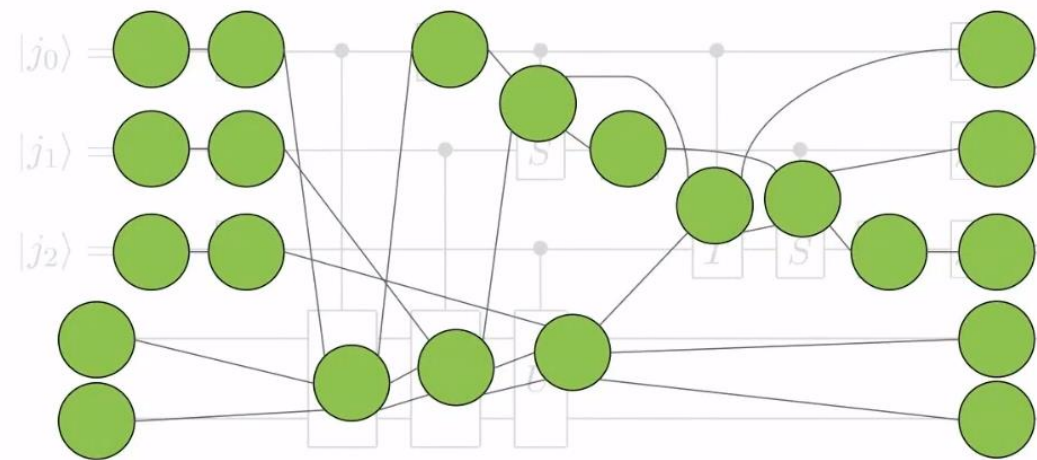
$$\varphi_{\mathbf{1}} = \sum_j \underline{H_{\mathbf{1}j}} \Psi_j$$

### Subspaces of $H$ with $i=1$

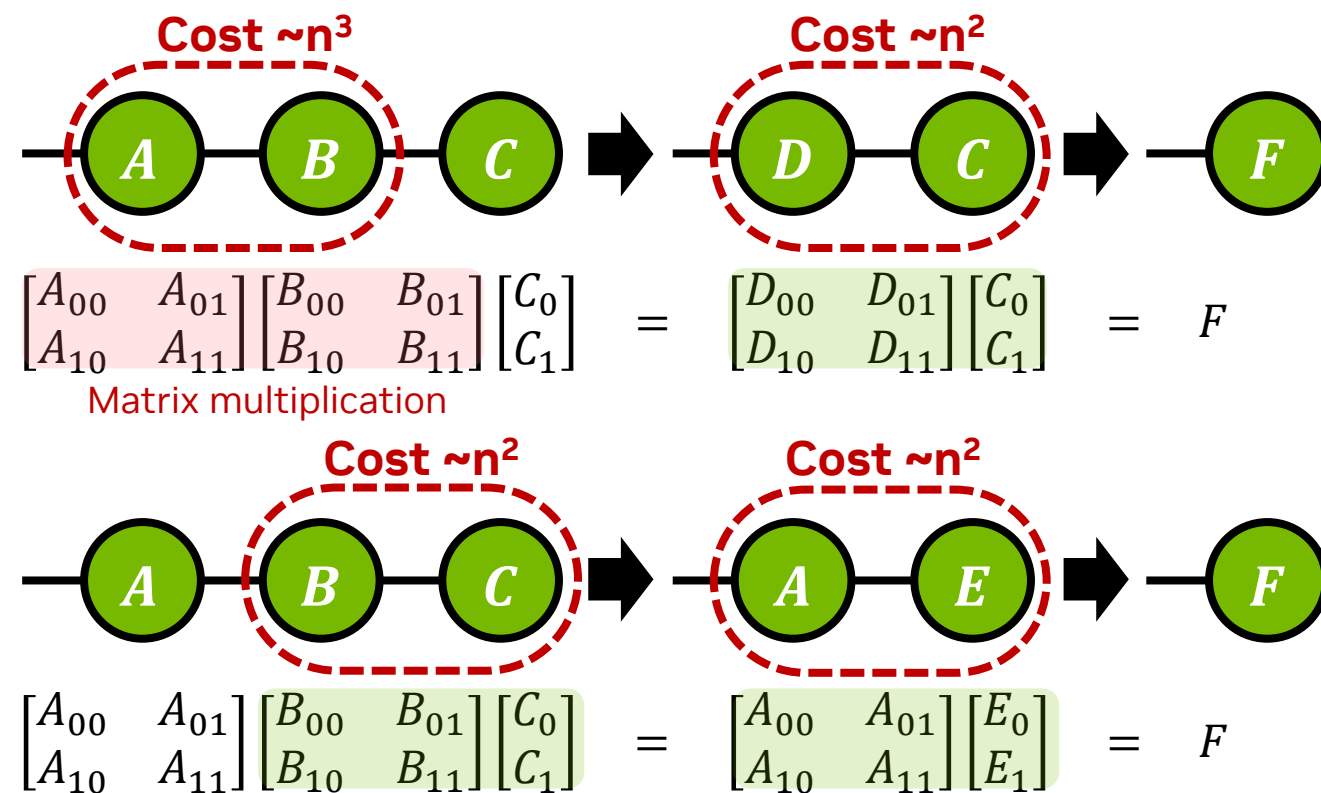
# cuTensorNet

Going from tensor to optimal contraction

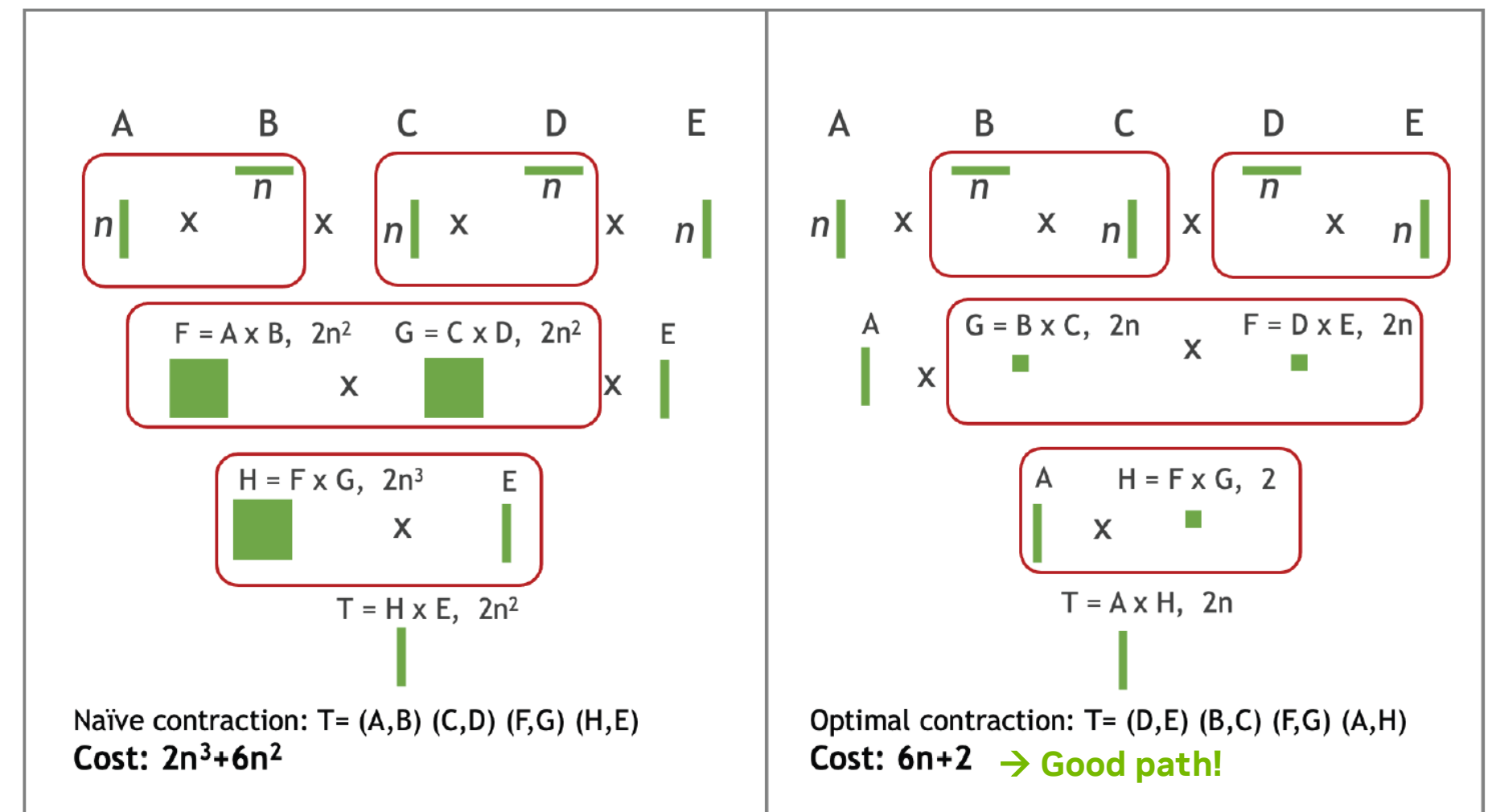
- Complex quantum circuit forms a tensor Network



- The path of the pairwise contractions does matter



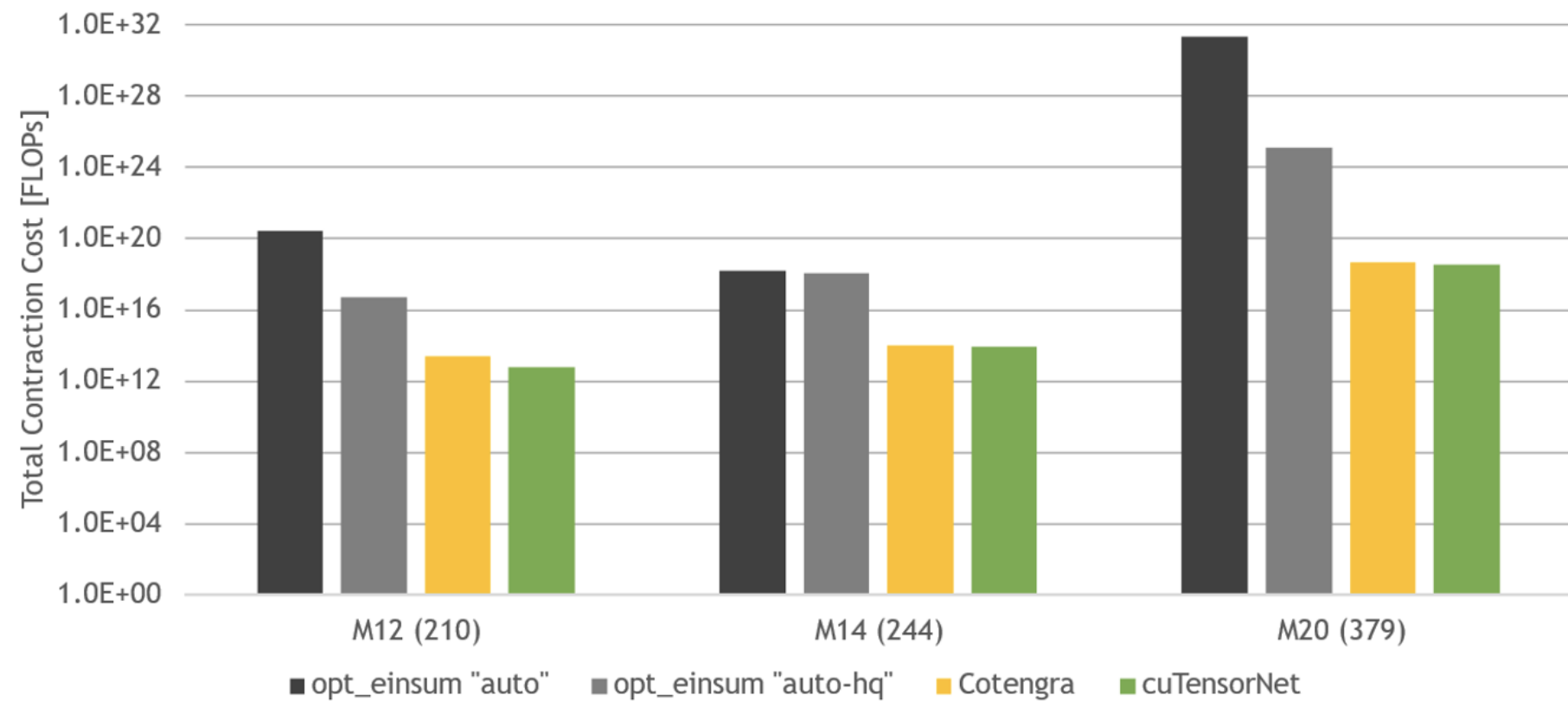
- cuTensorNet gives optimal contraction path for the tensor network subject to memory constraints and parallelization needs



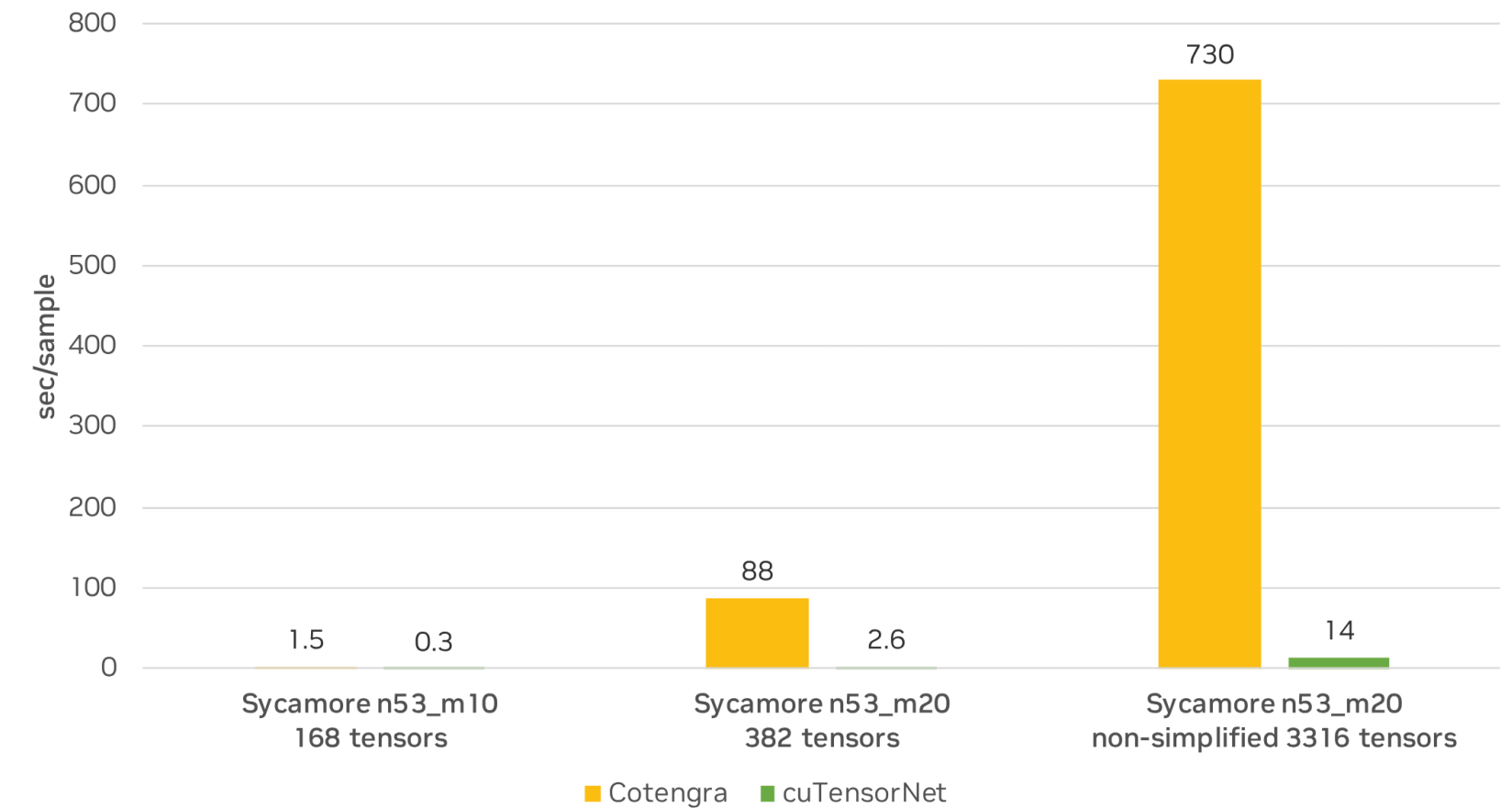


# cuTensorNet

## Performance Benchmark



*cuTensorNet pathfinding performance compared to similar packages, measured in FLOPs for the resulting contraction*



*Time to solution for cuTensorNet pathfinding, compared to Cotengra, for the Sycamore quantum circuits problems*

cuTensorNet provides state-of-the-art performance for both the pathfinding and contraction stages of tensor network simulation

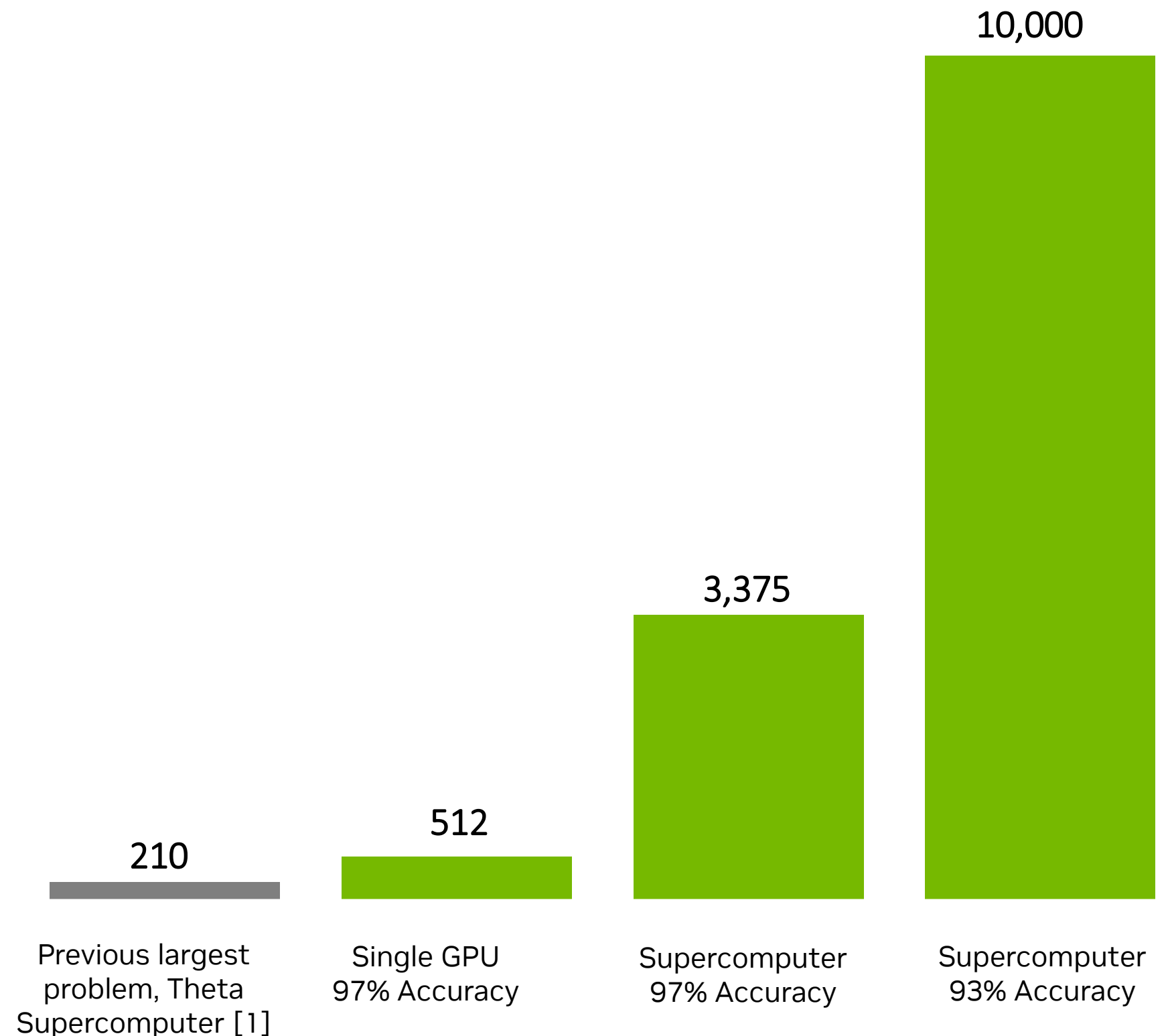
# Scaling to a Supercomputer

Vertex Count



NVIDIA's Selene DGX SuperPOD based supercomputer

- Using NVIDIA's Selene supercomputer
- Solved a 3,375 vertex problem (1,688 qubits) with 97% accuracy
- Solved a 10,000 vertex problem (5,000 qubits) with 93% accuracy



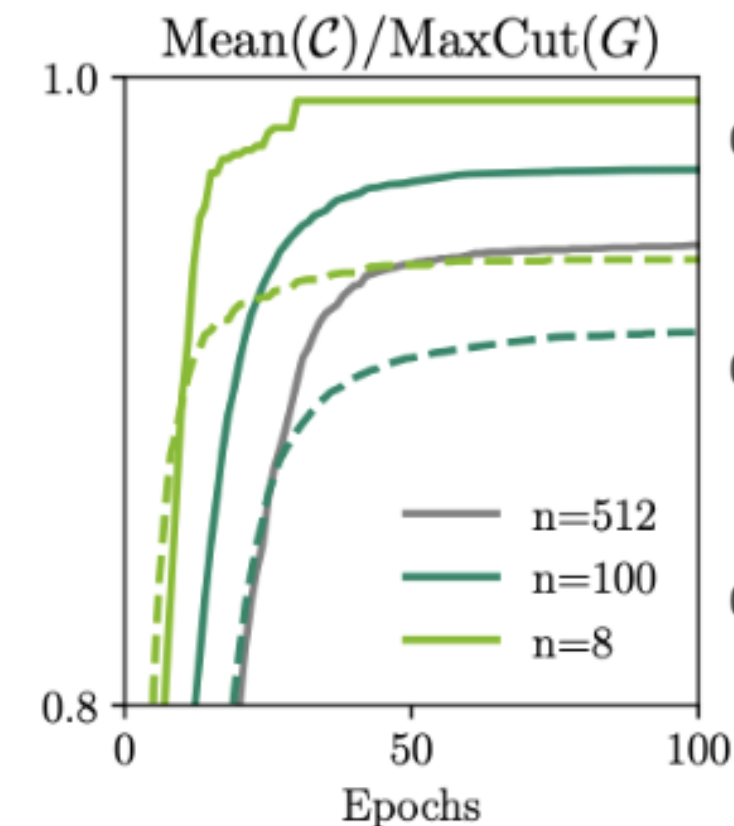
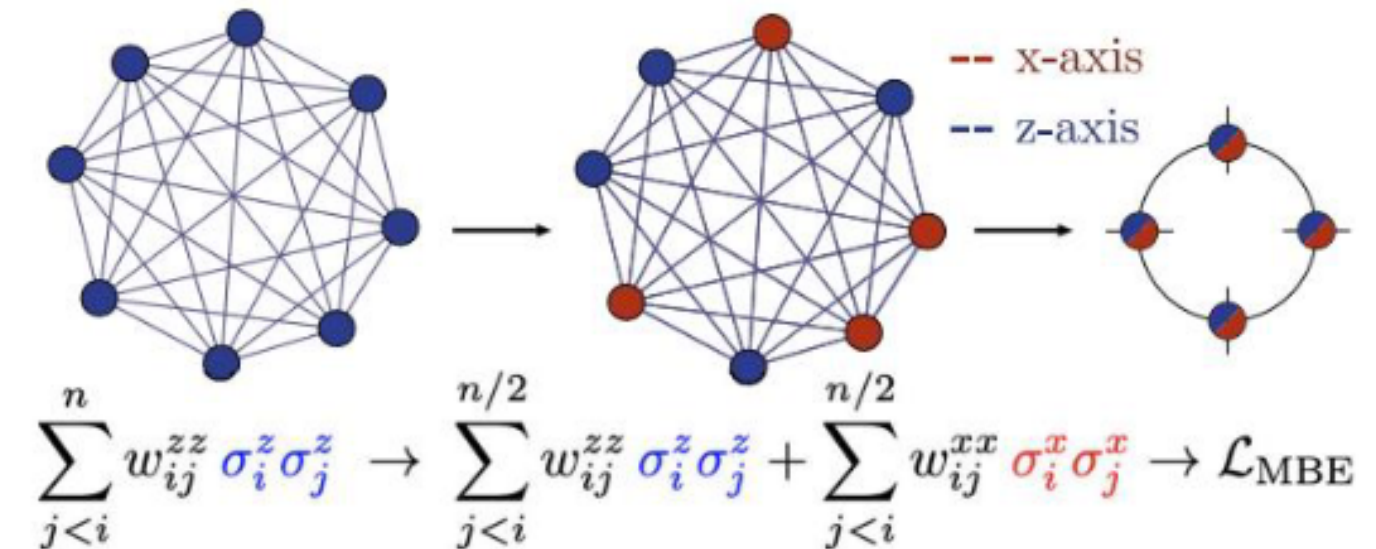
[1] Danylo Lykov et al, Tensor Network Quantum Simulator With Step-Dependent Parallelization, 2020  
<https://arxiv.org/pdf/2012.02430.pdf>

# Simulating MaxCut using Tensor Networks

- Tensor Networks are a natural fit for MaxCut
  - Fried et. al. (2017) [arxiv.org/abs/1709.03636](https://arxiv.org/abs/1709.03636)
  - Huang et. al (2019) [arxiv.org/abs/1909.02559](https://arxiv.org/abs/1909.02559)
  - Lykov et. al. (2020) [arxiv.org/abs/2012.02430](https://arxiv.org/abs/2012.02430)
- Patti et. al.(2021): NVIDIA Research proposes a novel variational quantum algorithm

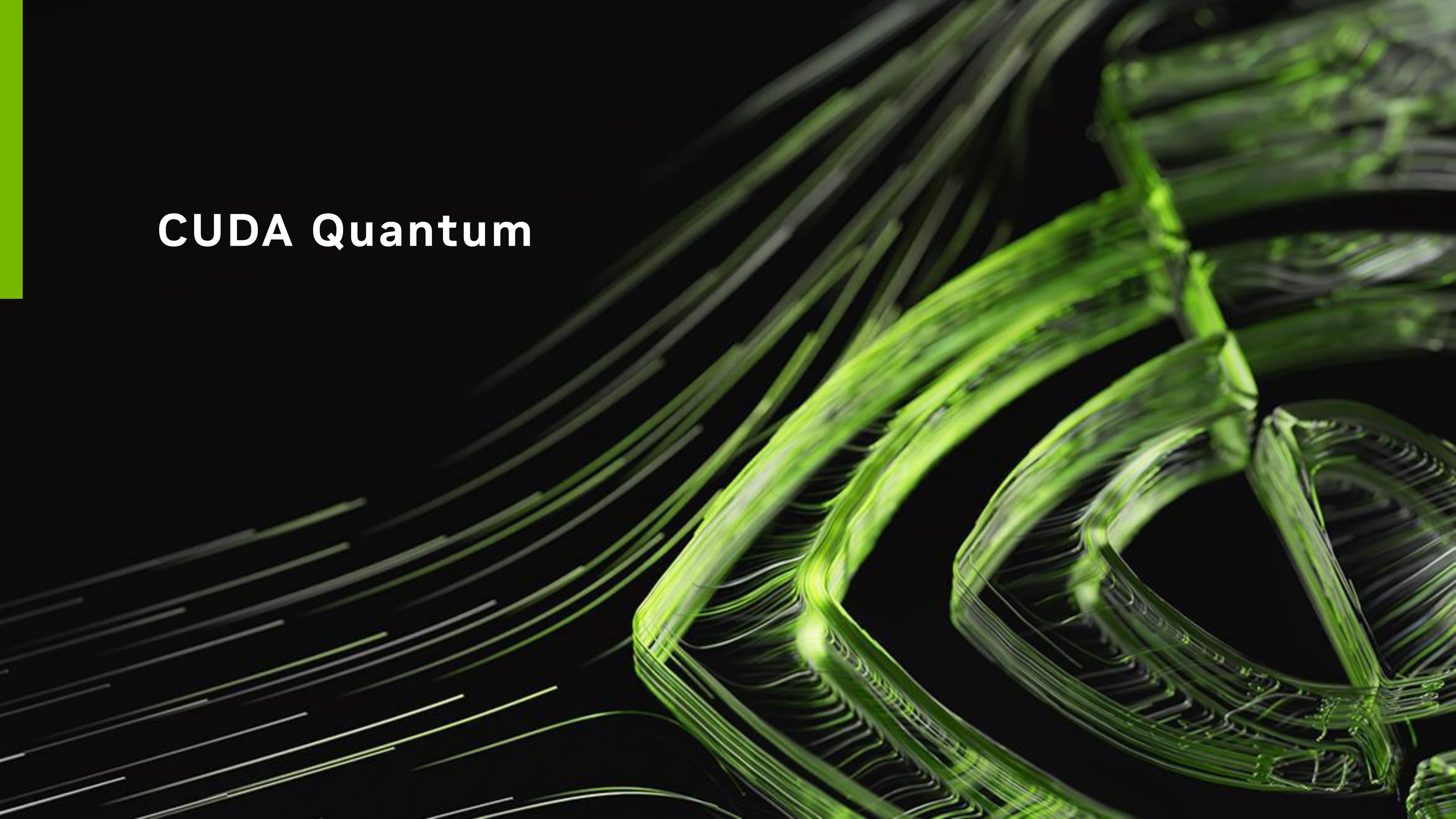
- Based on 1D tensor ring representation
- Multibasis encoding
- Able to find accurate solution for 512 vertices (256 qubits) on a single GPU

- Paper: [arxiv.org/abs/2106.13304](https://arxiv.org/abs/2106.13304)
- Code: [github.com/tensorly/quantum](https://github.com/tensorly/quantum)





# CUDA Quantum

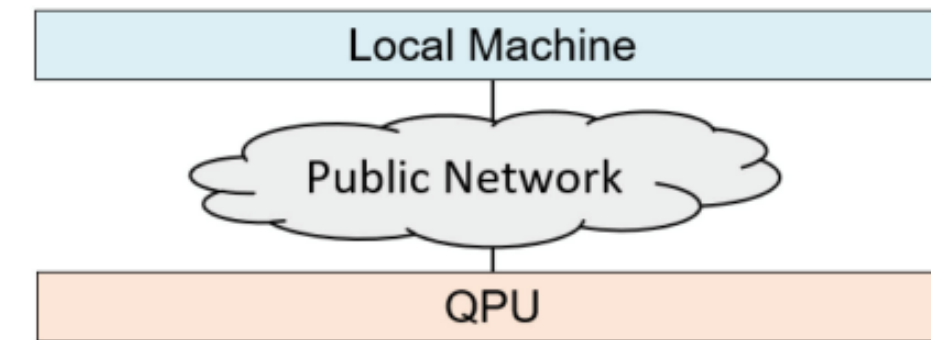




# Motivation behind CUDA Quantum

Integrate quantum computers seamlessly with the modern scientific computing ecosystem

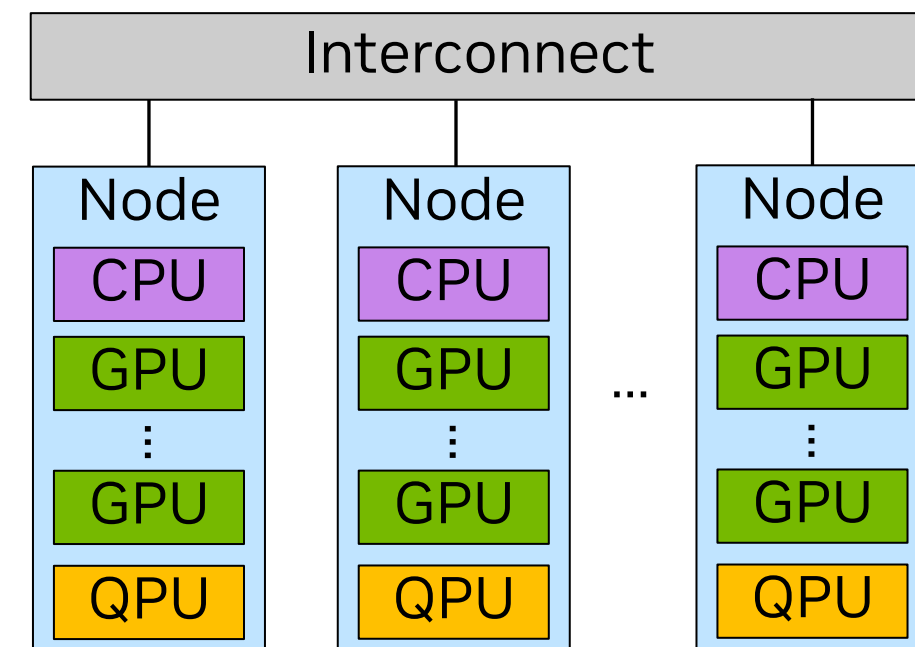
- HPC centers and many other groups worldwide are focused on the integration of quantum computers with classical supercomputers
- We expect quantum computers will accelerate some of today's most important computational problems and HPC workloads
  - Quantum chemistry, Materials simulation, AI
- We also expect CPUs and GPUs to be able to enhance the performance of QPUs
  - Classical preprocessing (circuit optimization) and postprocessing (error correction)
  - Optimal control and QPU calibration
- Want to enable researchers to seamlessly integrate CPUs, GPUs, and QPUs
  - Develop new hybrid applications and accelerate existing ones
  - Leverage classical GPU computing for control, calibration, error mitigation, and error correction



## Quantum Programming Today

Great for early experimentation.

vs. ...



## Where we need to get...

Programming a coherent quantum-classical node

Hybrid quantum-classical applications at scale.

Figure adapted from:  
Quantum Computers for High-Performance Computing.  
Humble, McCaskey, Lyakh, Gowrishankar, Frisch, Monz.  
IEEE Micro Sept 2021. 10.1109/MM.2021.3099140

# Motivation behind CUDA Quantum

Integrate quantum computers seamlessly with the modern scientific computing ecosystem

**We believe quantum programming should be:**

## Easy to Learn:

- No domain-specific expertise or new language required
- Integrable with today's scientific computing and AI workflows

## Fast

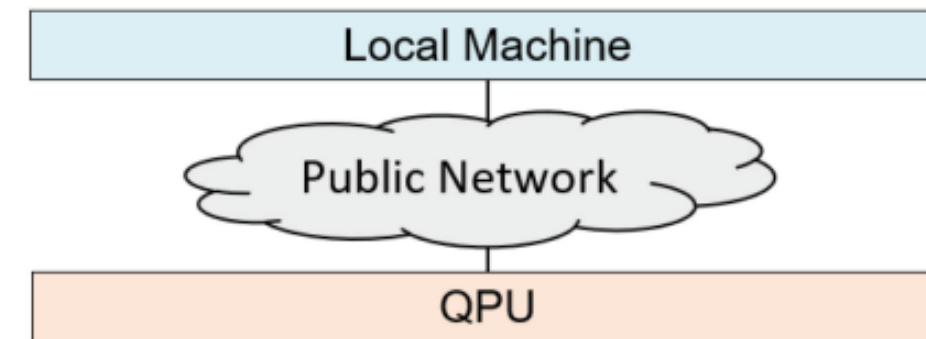
- Performant compilation with no quantum-specific bottlenecks
- Integrated state-of-the-art simulation methods
- Straightforward interoperability with classical accelerated computing

## Flexible

- Easy porting between classical computers, simulated QPUs, and real QPUs

## Scalable

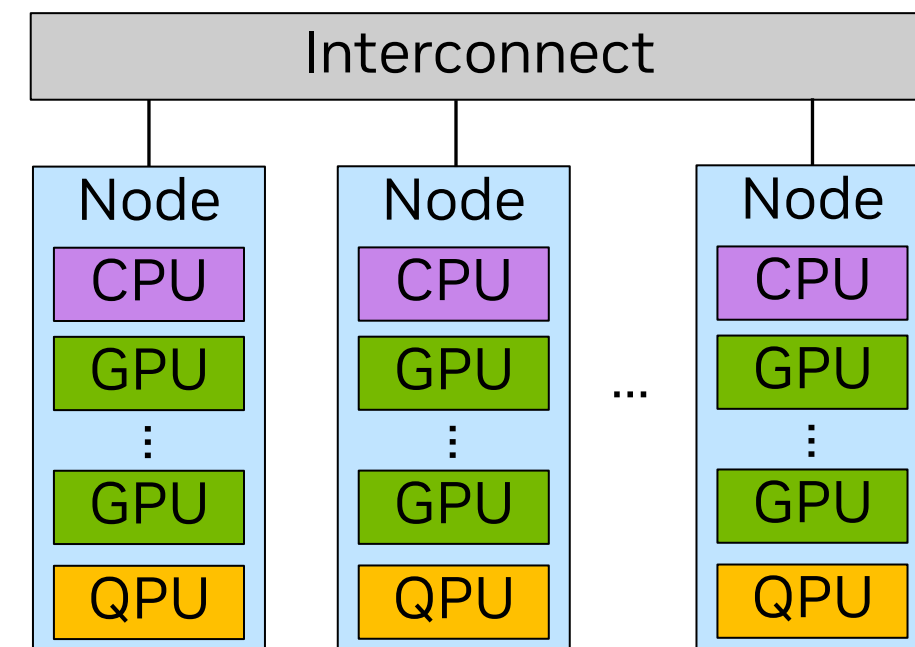
- Bring supercomputers to bear to advance quantum research



## Quantum Programming Today

Great for early experimentation.

VS. ...



## Where we need to get...

Programming a coherent quantum-classical node

Hybrid quantum-classical applications at scale.

Figure adapted from:  
Quantum Computers for High-Performance Computing.  
Humble, McCaskey, Lyakh, Gowrishankar, Frisch, Monz.  
IEEE Micro Sept 2021. 10.1109/MM.2021.3099140



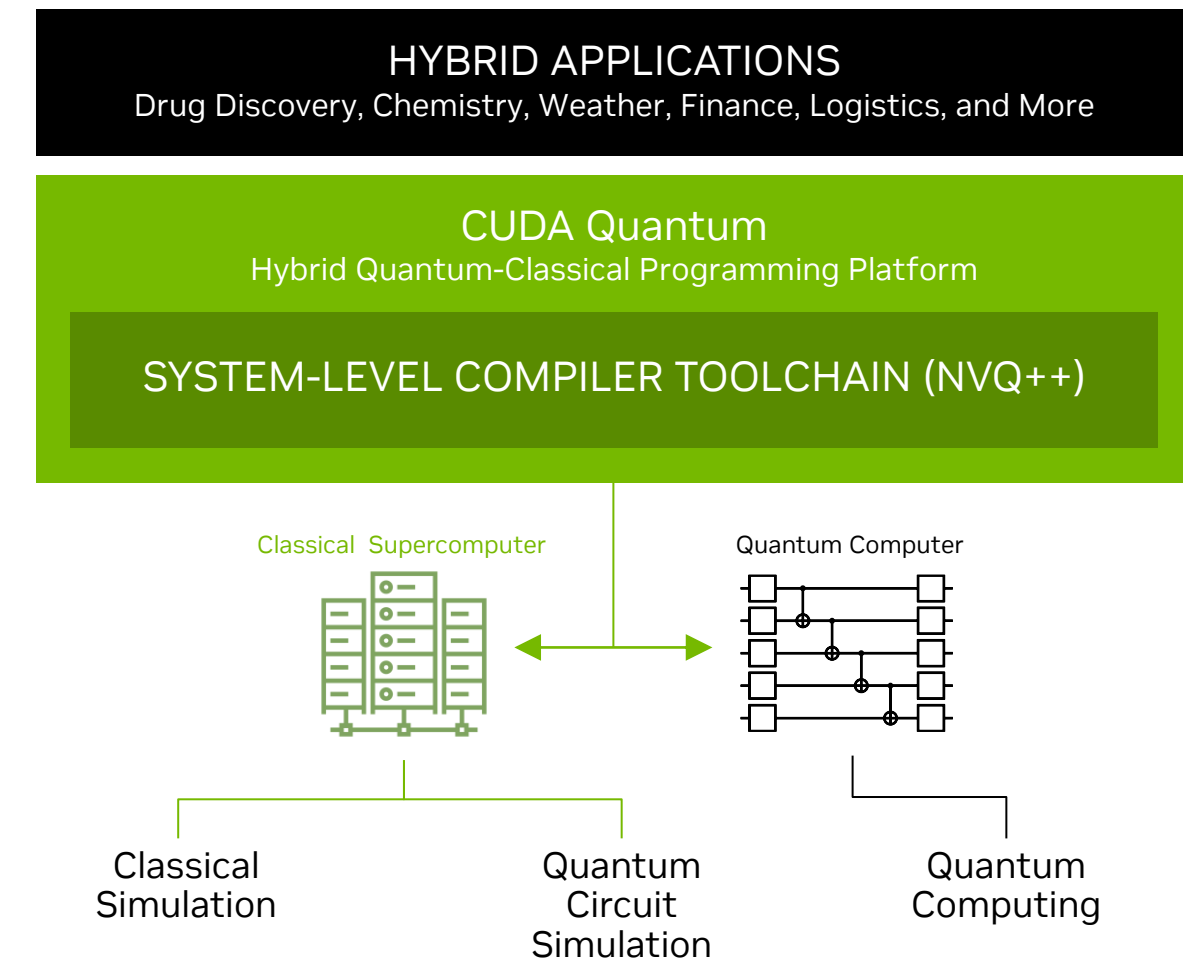
# Introducing CUDA Quantum

Platform for unified quantum-classical accelerated computing

- Programming model extending C++ and Python with quantum kernels
- Open programming model, open-source compiler
  - <https://github.com/NVIDIA/cuda-quantum>
- QPU Agnostic – Partnering broadly including superconducting, trapped ion, neutral atom, photonic, and nitrogen-vacancy center QPUs
- Interoperable with the modern scientific computing ecosystem
- Seamless transition from simulation to physical QPU

```
auto ansatz = [](std::vector<double> thetas) __qpu__ {
    cudaq::qreg<3> q;
    x(q[0]);
    ry(thetas[0], q[1]);
    ry(thetas[1], q[2]);
    x<cudaq::ctrl>(q[2], q[0]);
    x<cudaq::ctrl>(q[0], q[1]);
    ry(-thetas[0], q[1]);
    x<cudaq::ctrl>(q[0], q[1]);
    x<cudaq::ctrl>(q[1], q[0]);
};

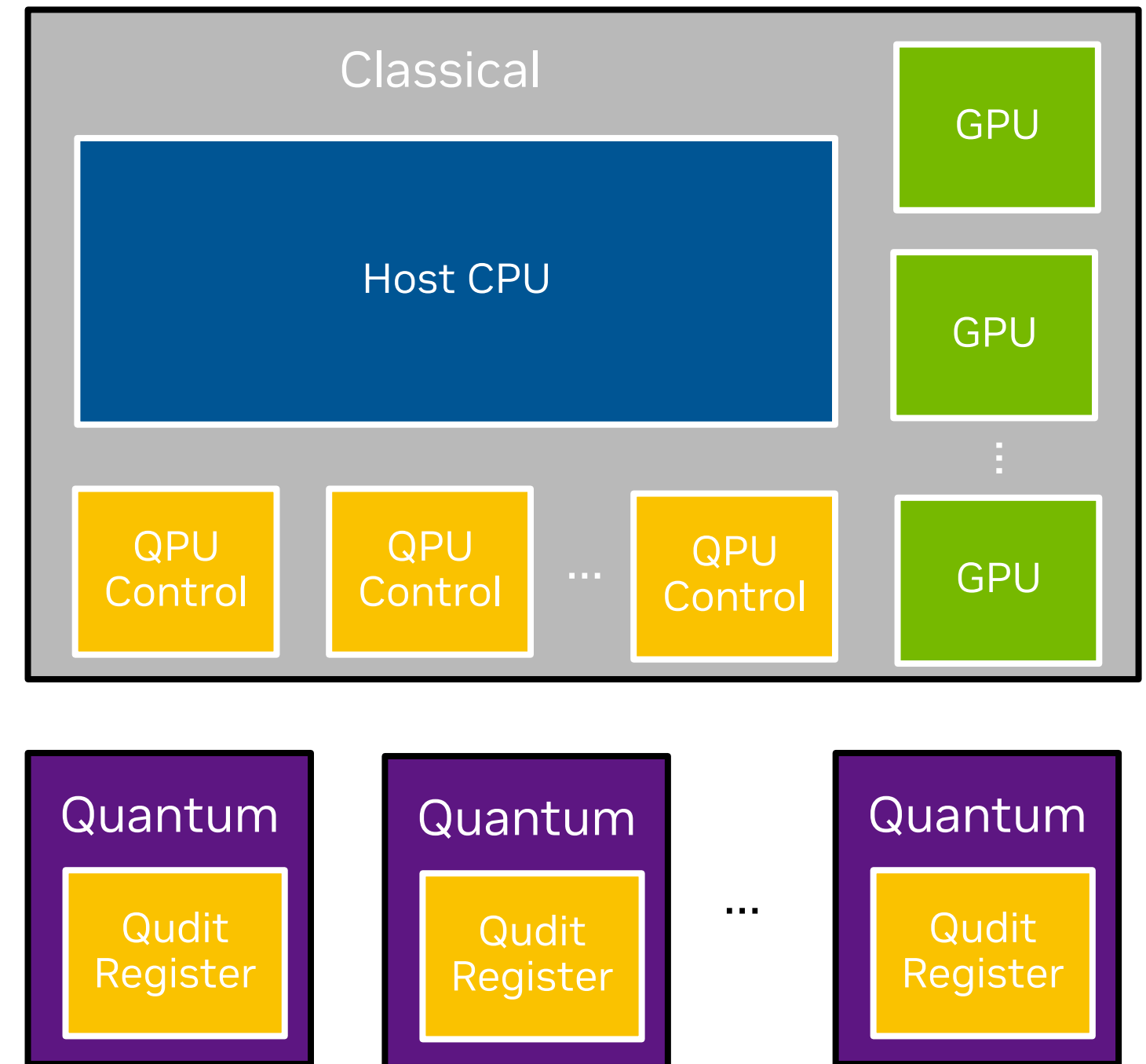
cudaq::spin_op H = ...;
double energy = cudaq::observe(ansatz, H, {M_PI, M_PI_2});
```



# CUDA Quantum Machine Model

Devices available and discrete memory spaces.

- The system architecture assumes multiple discrete memory spaces that are either classical or quantum
- Classical
  - Host CPU, Device GPU(s), QPU(s) Control
- Quantum
  - A general **qudit** register
- Implications
  - Some classical compute available for hybrid quantum-classical code (arithmetic operations, conditional statements on qubit measurements, etc.)
  - GPU compute available for pre- / post-processing, hybrid application workflows
  - QISA on general qudits (e.g. qubits or bosonic modes)
    - **NOTE CUDAQ development focus thus far has been on qudit<2>**
  - Agnostic to the control system architecture (just assume there is some control ISA and our compiler lowers C++ to that representation)



# CUDA Quantum Python Bindings

Core C++ compiler platform exposed via builder API  
JIT compile Quake representation at runtime

- The C++ Quake builder API is what CUDAQ exposes to Python

```
import cudaq

# Set the Simulator to cuQuantum
cudaq.set_target('nvidia')

# Create a Bell State Kernel
bell = cudaq.make_kernel()
qr = bell.qalloc(2)
bell.h(qr[0])
bell.cx(qr[0], qr[1]);
bell.mz(qr)

# Print the Quake Code
print(bell)

# JIT Compile and Execute
counts = cudaq.sample(bell)
print(counts)
```

Sample

```
import cudaq

# Set the backend
cudaq.set_target('quantinuum')

# Create the kernel function signature
# here void(float)
ansatz, theta = cudaq.make_kernel(float)
q = ansatz.qalloc(2)
ansatz.x(q[0])
ansatz.ry(theta, q[1]);
ansatz.cx(q[0], q[1]);

h = cudaq.SpinOperator(...)

# API mirrors the C++
result = cudaq.observe(ansatz, h, .59)
print('<H> = ', result.expectation_z())
```

Observe

```
import cudaq

# Set the backend
cudaq.set_target('density-matrix-cpu')

# Create a depolarization channel on
# X operations on qubit 0
depol = cudaq.DepolarizationChannel(.1)
noise = cudaq.NoiseModel()
noise.add_channel('x', [0], depol)

# Create your kernel code
bell = cudaq.make_kernel()
...

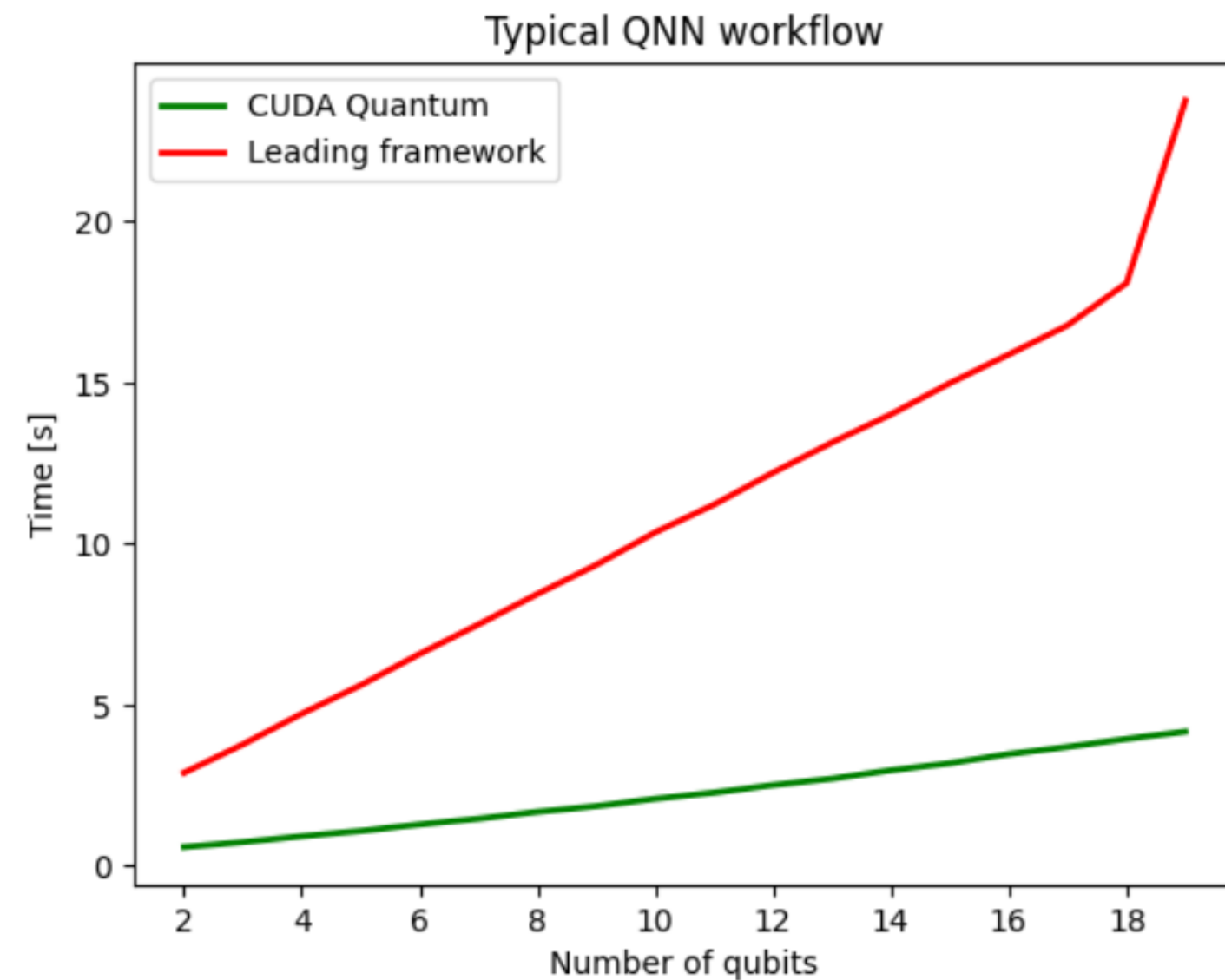
# Sample in the presence of noise
noisyCounts = cudaq.sample(bell,
                           noise_model=noise)
print(noisyCounts)
```

Noise Modeling

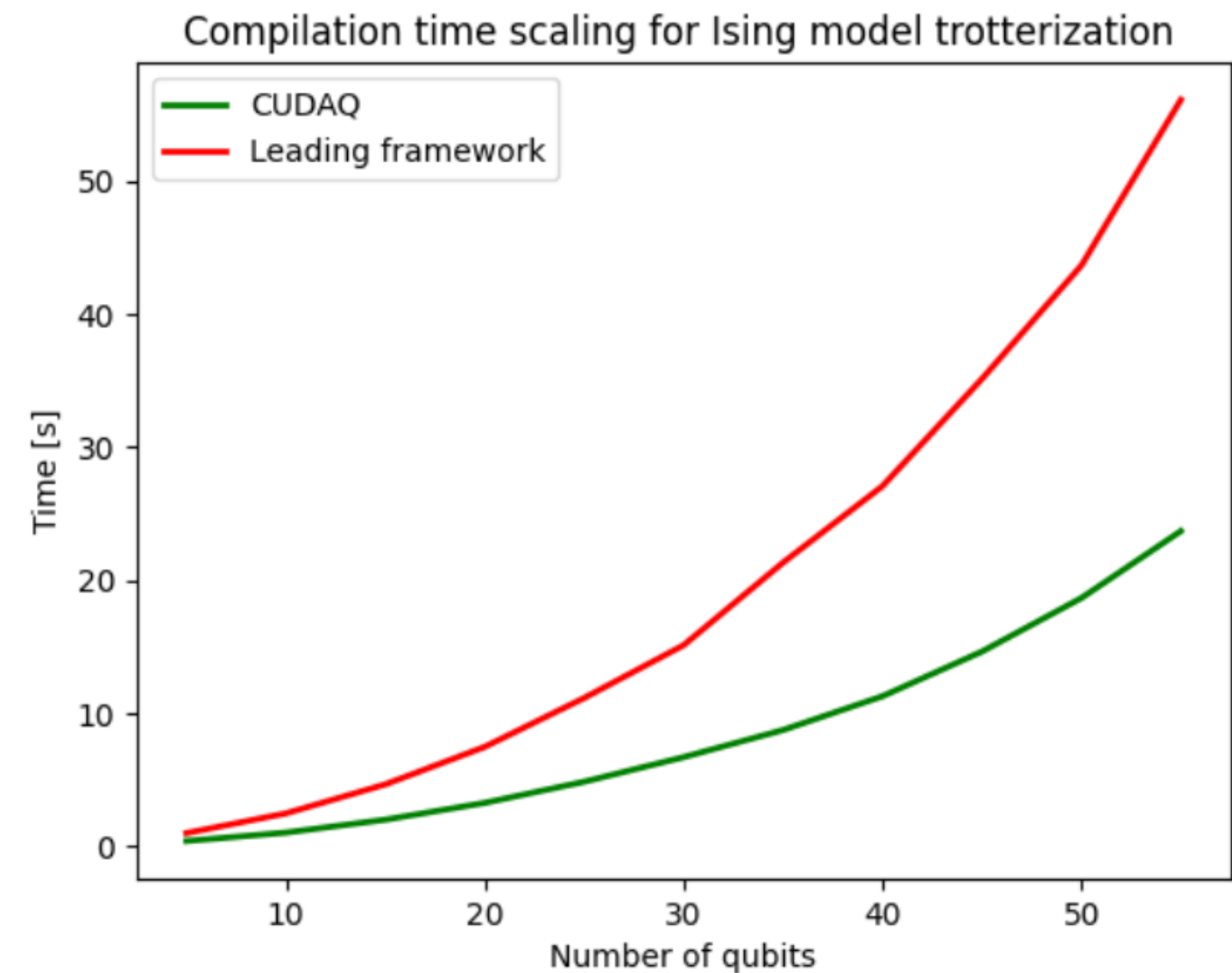


# CUDA Quantum Compile Time Speedup

Faster at synthesizing quantum circuits via optimized compiler toolchain



- **5x speedup:** Comparison between CUDA Quantum with a leading quantum computing SDK, both leveraging the NVIDIA cuQuantum backend to optimally offload circuit simulation onto NVIDIA GPUs.

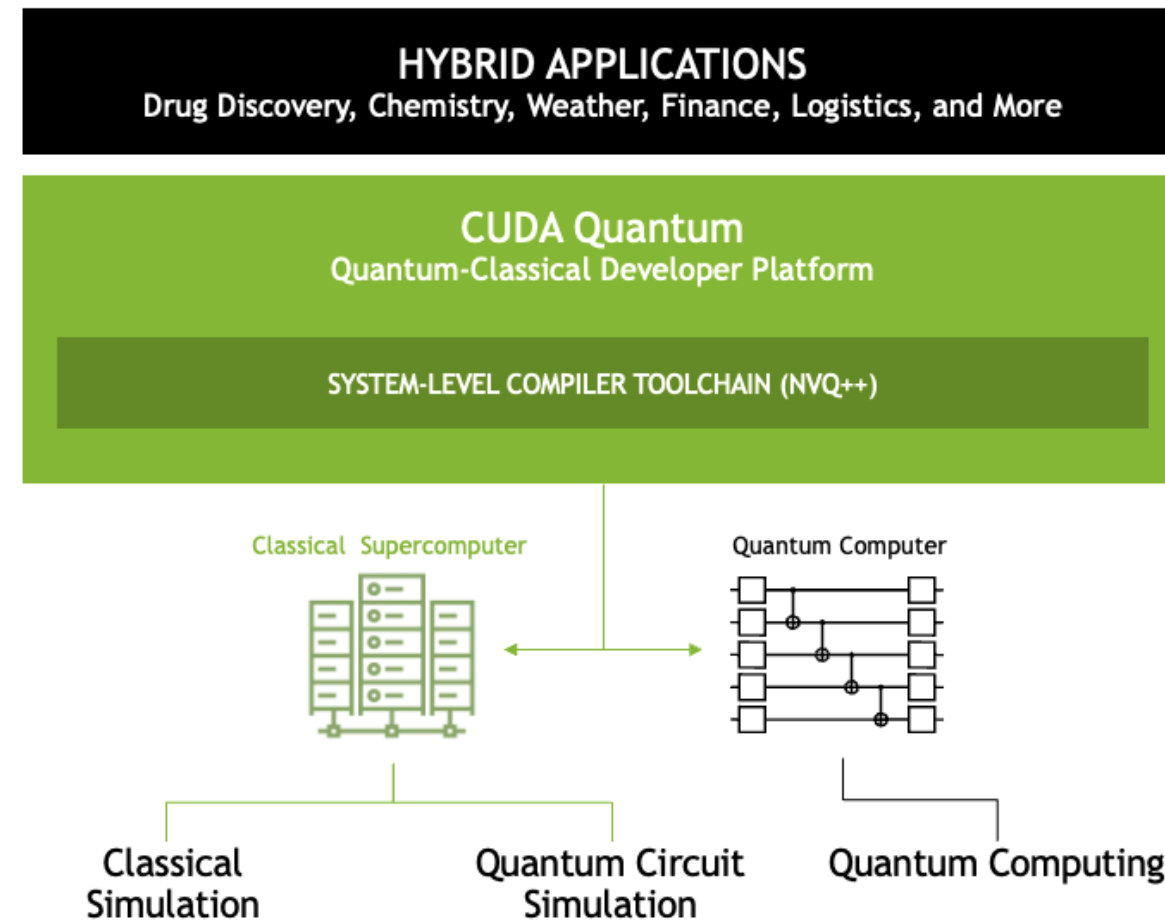


- **2.4x Faster:** The nvq++ compiler used by CUDA Quantum is on average faster compared to its competition. Compilation involves circuit optimization, decomposing into the native gate sets supported by the hardware and qubit routing.

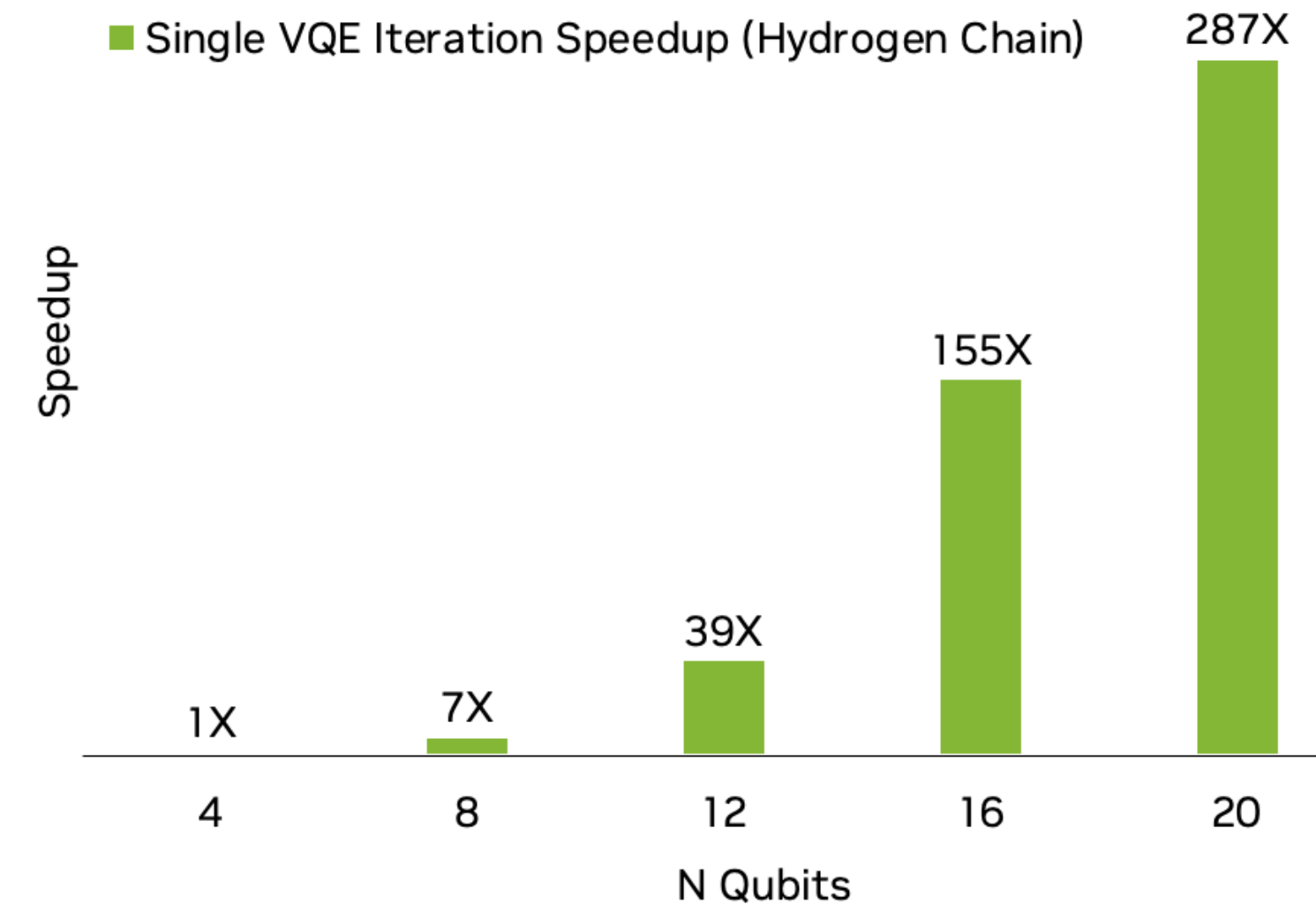
# CUDA Quantum: Open-Source Software for Quantum Accelerated Supercomputing

Develop applications for integrated quantum-classical computing

## CUDA QUANTUM PLATFORM



## CUDA QUANTUM FEATURES



[github.com/nvidia/cuda-quantum](https://github.com/nvidia/cuda-quantum)

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/cuda-quantum>

# CUDA Quantum Platform and Asynchronous Execution

Expose the underlying system architecture to the programmer

- The system architecture model considers access to multiple quantum accelerators
- CUDA Quantum provides programmatic access to this configuration via the `quantum_platform`
- CUDA Quantum and cuQuantum expose a native platform that models a virtual QPU for every CUDA device.
- Each CUDA device gets a cuQuantum based simulator
- Enable experimentation with distributed quantum computing

```
// Programmer can query info about the platform
auto& platform = cudaq::get_platform();

// Get number of QPUs available
auto numQpus = platform.num_qpus();

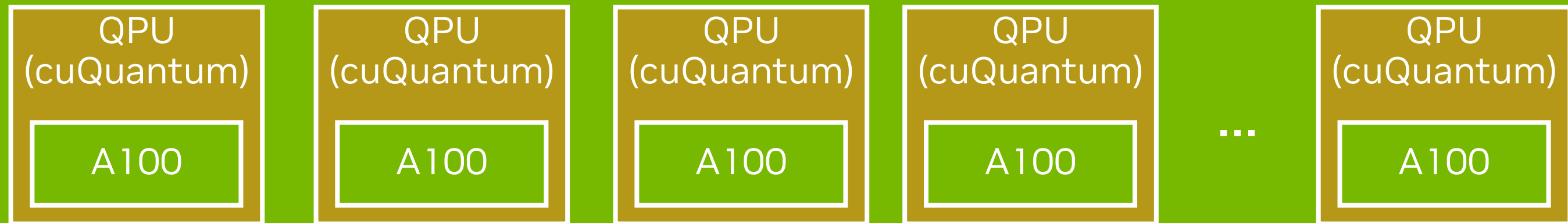
// Get the number of qubits on QPU 1
auto nQ1 = platform.get_num_qubits(1);

// Get QPU 0 connectivity.
auto connectivity = platform.get_connectivity(0);

// Async task execution on available QPUs
std::vector<std::future<double>> subs;
for (auto qpuIdx : cudaq::range(numQpus))
    subs.emplace_back(cudaq::my_async_task(qpuIdx, ...));

auto sum = std::reduce(std::execution::par,
                      cudaq::when_all(subs), 0.0);
```

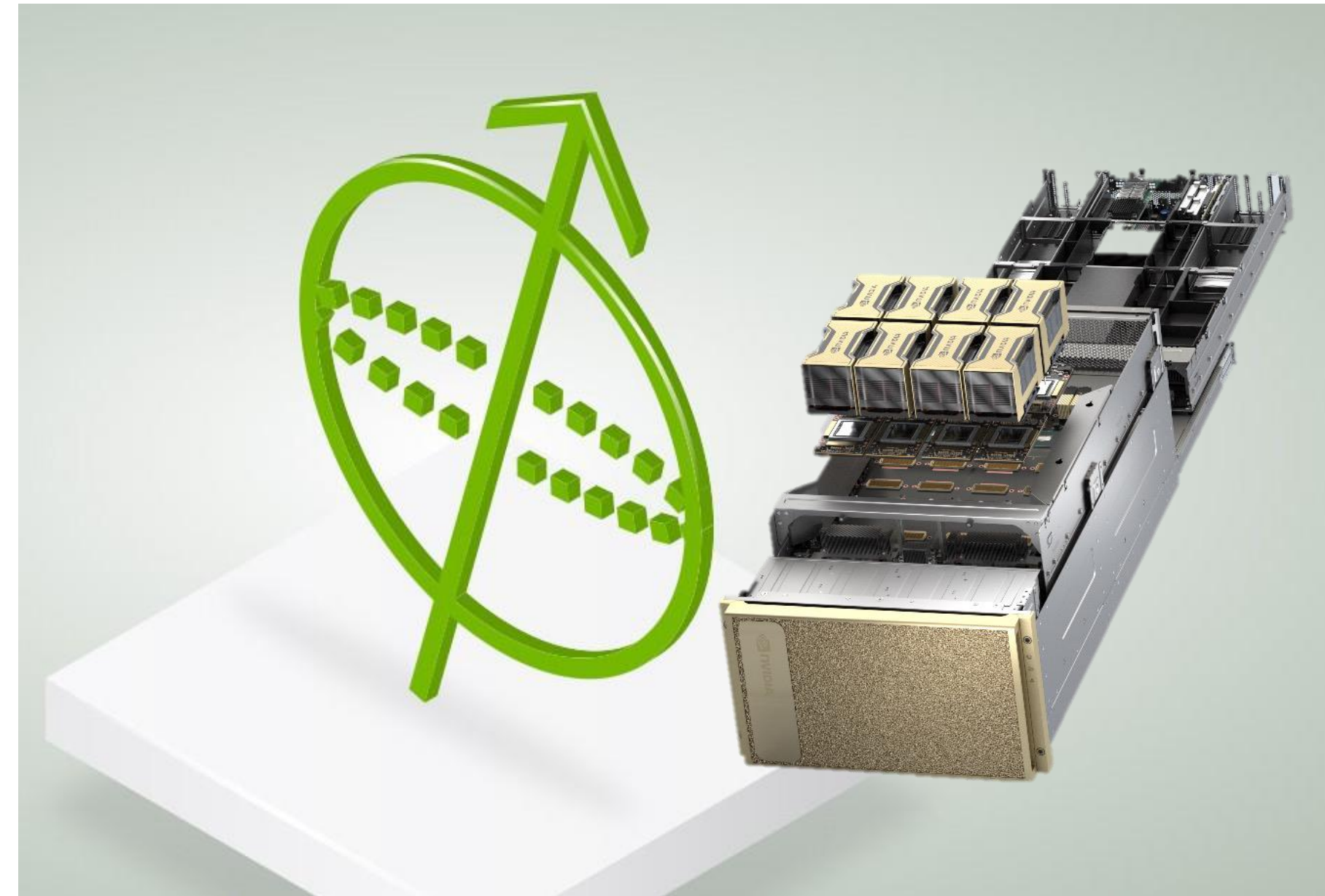
e.g. `cudaq::mqpu_platform`





# SUMMARY

- **cuQuantum:** An **SDK designed to accelerate and scale quantum circuit simulations**, enabling researchers to simulate larger quantum systems and explore new capacities in quantum computing.
- **CUDA Quantum:** An **open-source programming model** and compiler toolchain for integrating and programming across quantum processing units (QPUs), GPUs, and CPUs
- **DGX Quantum:** The first **unified system for integrated quantum-classical computing**, developed by NVIDIA and Quantum Machines. It combines NVIDIA's Grace Hopper Superchip with OPX for ultra-low latency quantum control, offering an optimized platform for quantum research







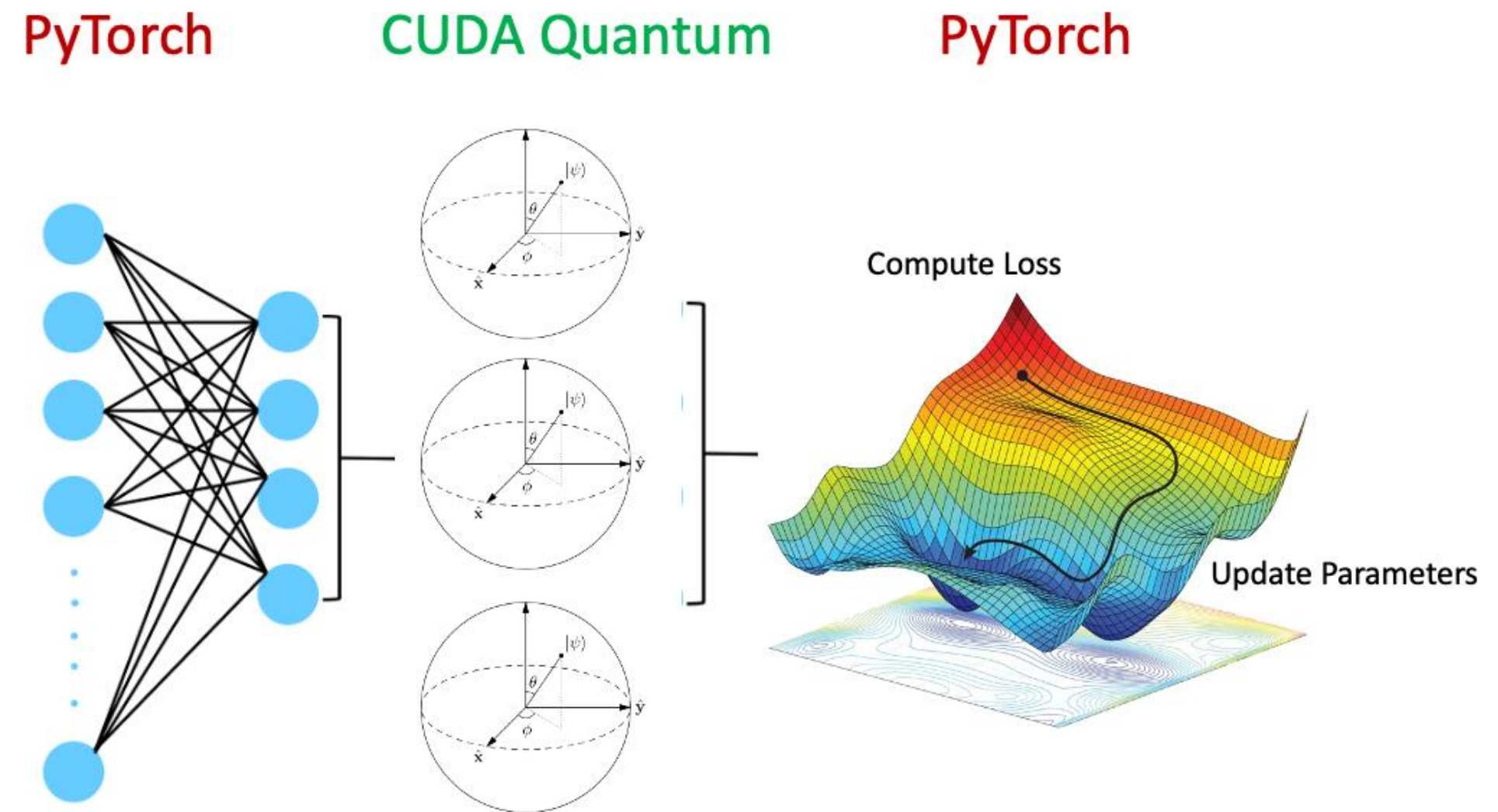
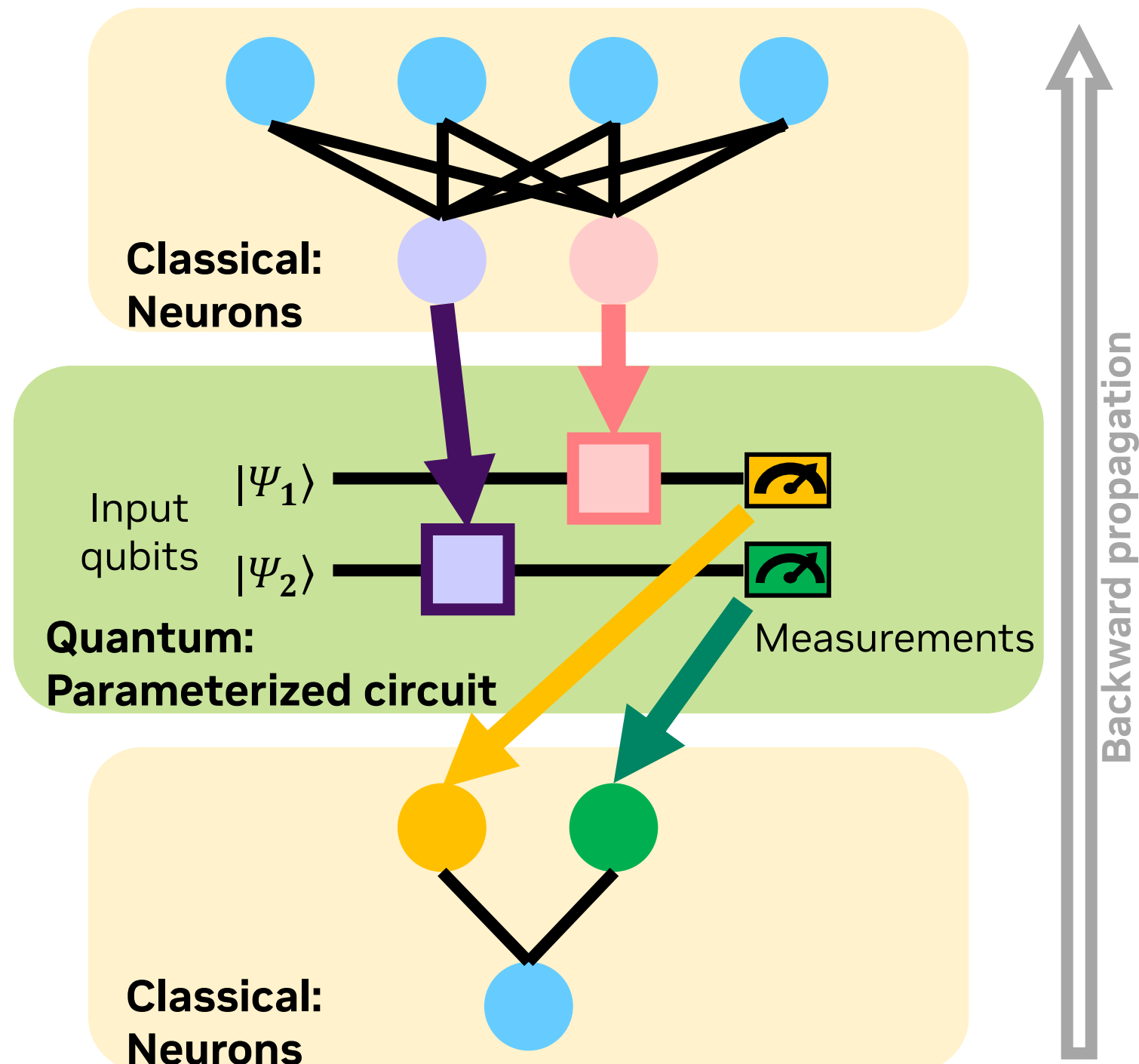
# Hands-on Lab



# CUDA Quantum + Machine Learning Framework

Hybrid quantum neural network architecture accelerated by GPUs made possible by CUDA Quantum

- Explore hybrid quantum-classical algorithms
- Combine quantum circuit and neural network
- CUDA Quantum also seamlessly integrates with existing machine learning frameworks

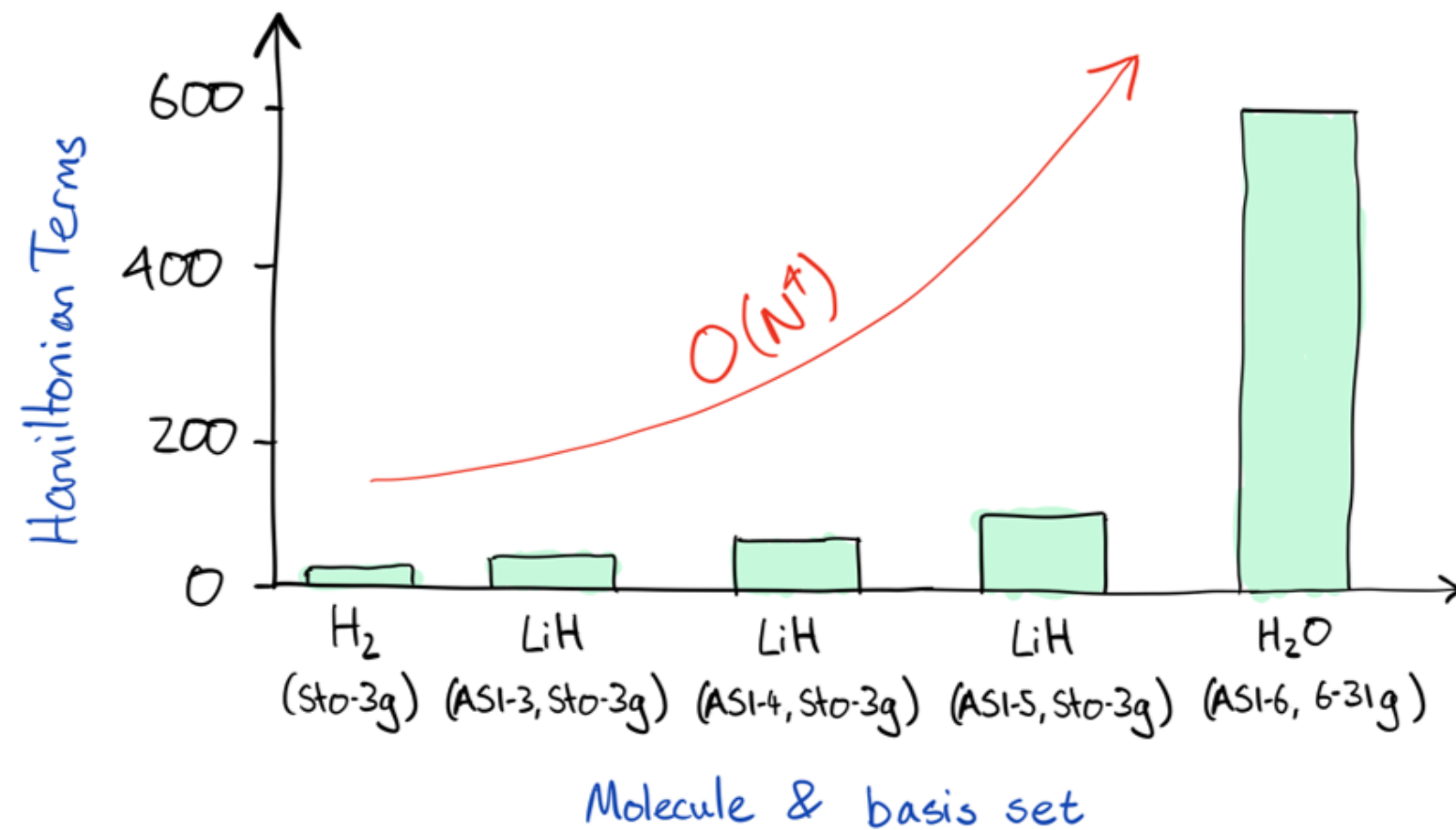




# Potential Limitations of VQE

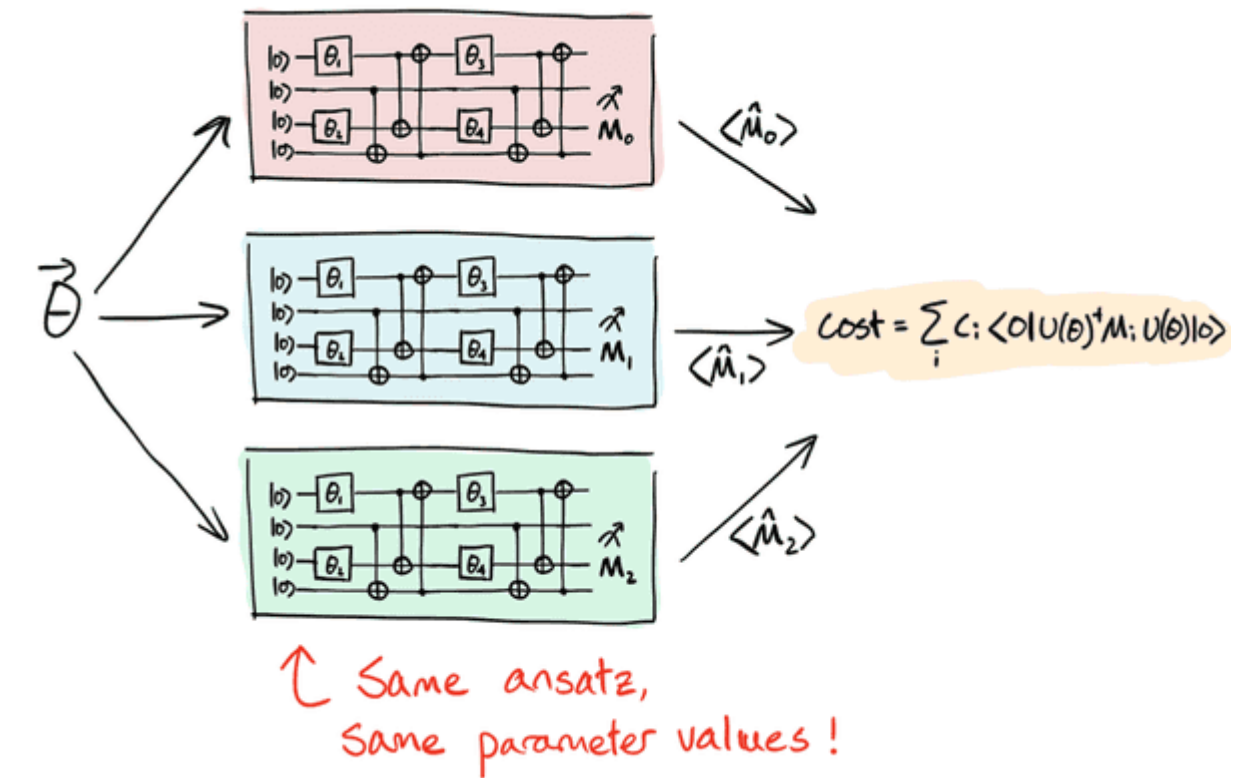
Deep quantum circuit and numerous measurements are required for large molecules

Number of Hamiltonian terms for measurement:

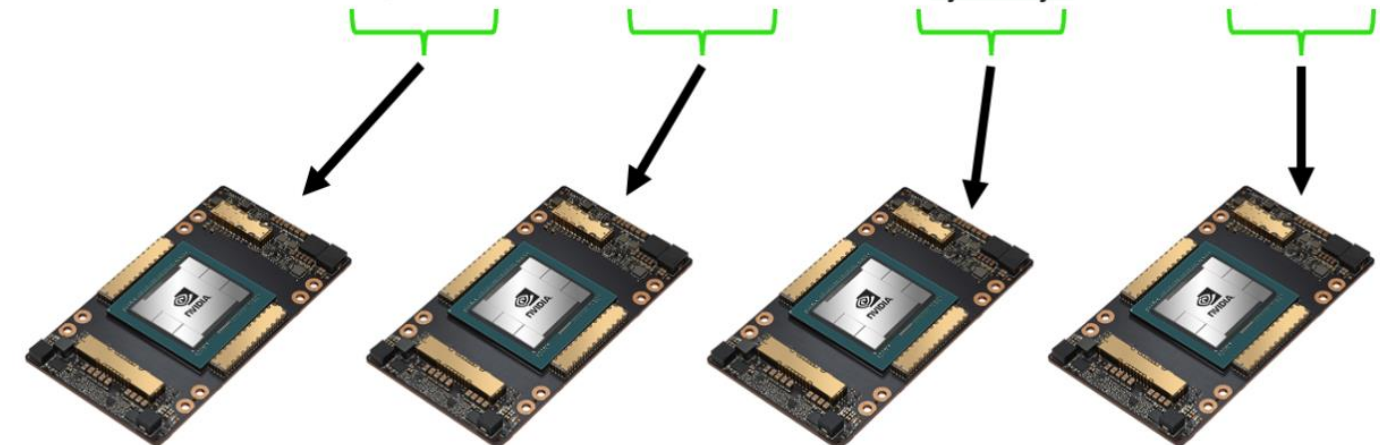


Large molecules  
→ Hamiltonian terms grows polynomially

Asynchronous computing for multi-term hamiltonians:



$$\text{Hamiltonian} = P_0 + P_1 + \dots + P_i + P_{i+1} + \dots + P_j + P_{j+1} + \dots + P_{n-1} + P_n$$



## Useful Info

- **More Examples:** [https://github.com/Squirtle007/CUDA\\_Quantum](https://github.com/Squirtle007/CUDA_Quantum)

- **Run Locally:**

- Install CUDA Quantum with pip or Docker
  - Docker: <https://nvidia.github.io/cuda-quantum/latest/install.html>
  - pip: <https://pypi.org/project/cuda-quantum/>



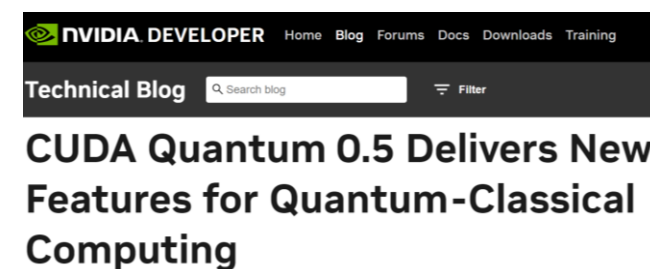
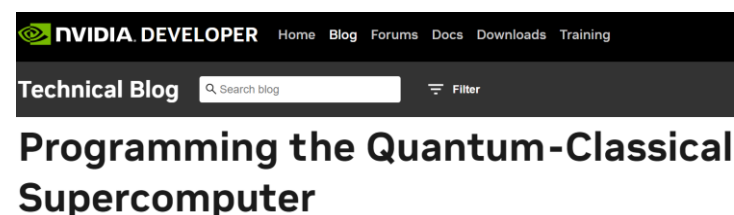
```
pip install cuda-quantum
```



- **Documentation:**

- <https://nvidia.github.io/cuda-quantum/latest/>

- **Blogs:**



- **Pika Wang:**







# Thank you

Yun Yuan Wang (Pika Wang), Solutions Architect | NVAITC, NVIDIA Taiwan

[yunyuanw@nvidia.com](mailto:yunyuanw@nvidia.com)



Pikachu generated by AI