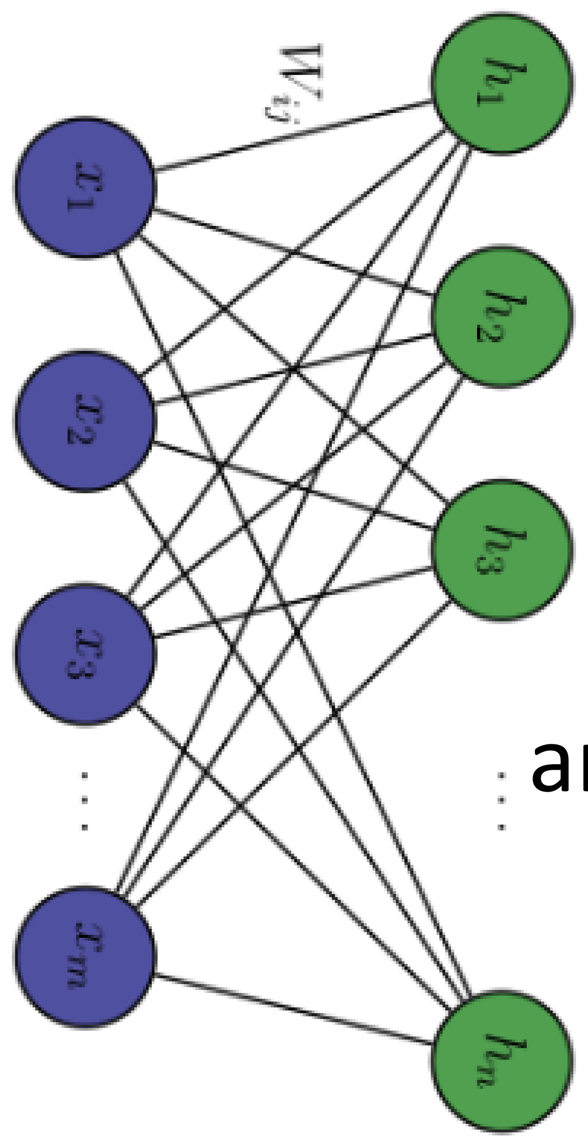
Quantum machine learning: scaling up and accelerating the quantum circuit simulation with Nvidia CUDA-Q

Marwa Farag, Quantum Algorithm Engineer | NSF workshop on post Quantum AI/
April 1st, 2024.

Agenda

- Introduction: quantum machine learning
- CUDA-Q platform
- Accelerating and scaling up the Q-RBM circuit
- Conclusion

Quantum machine learning: Quantum Restricted Boltzmann Machine (Q-RBM)

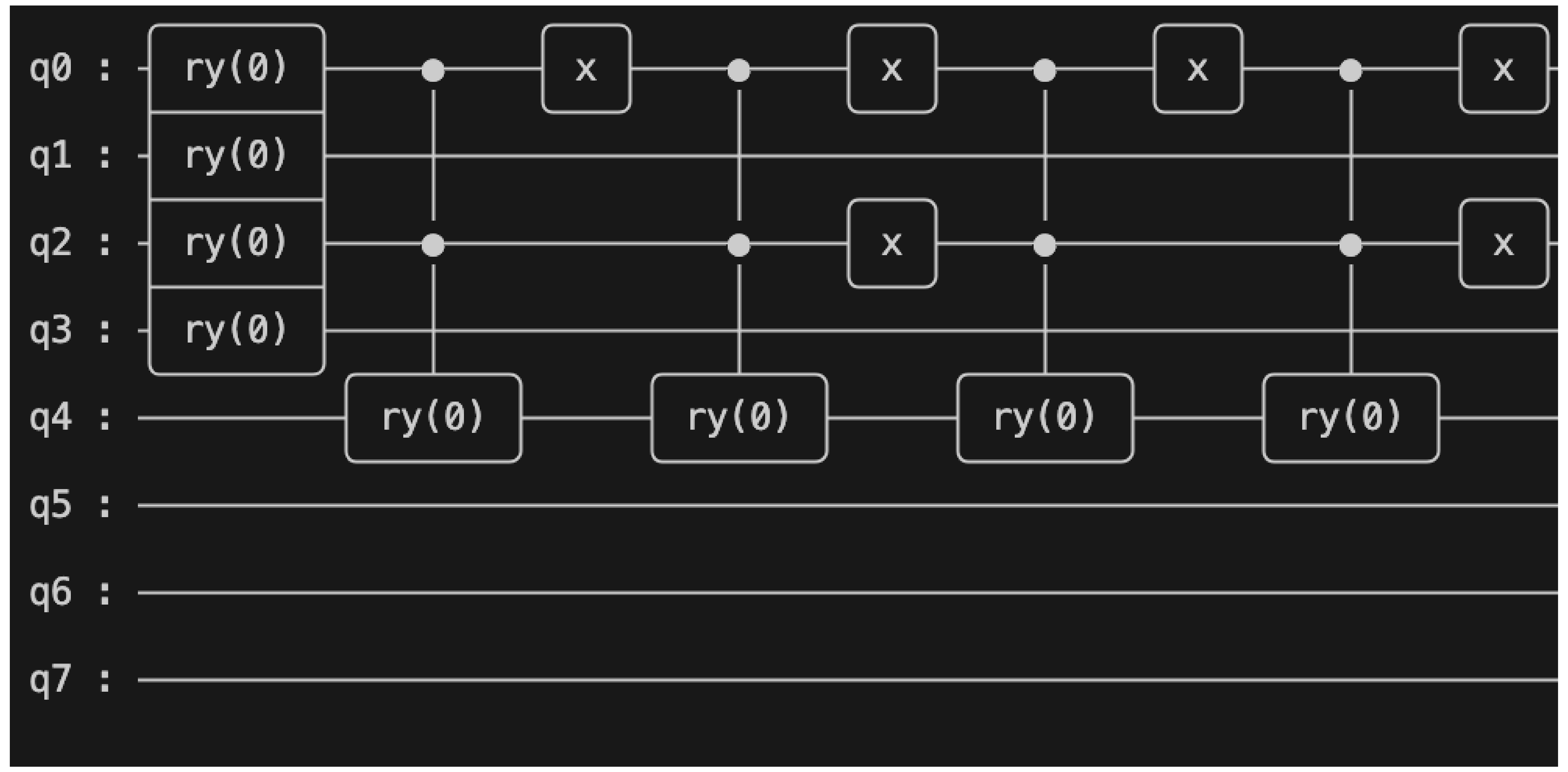


QPU

x

h

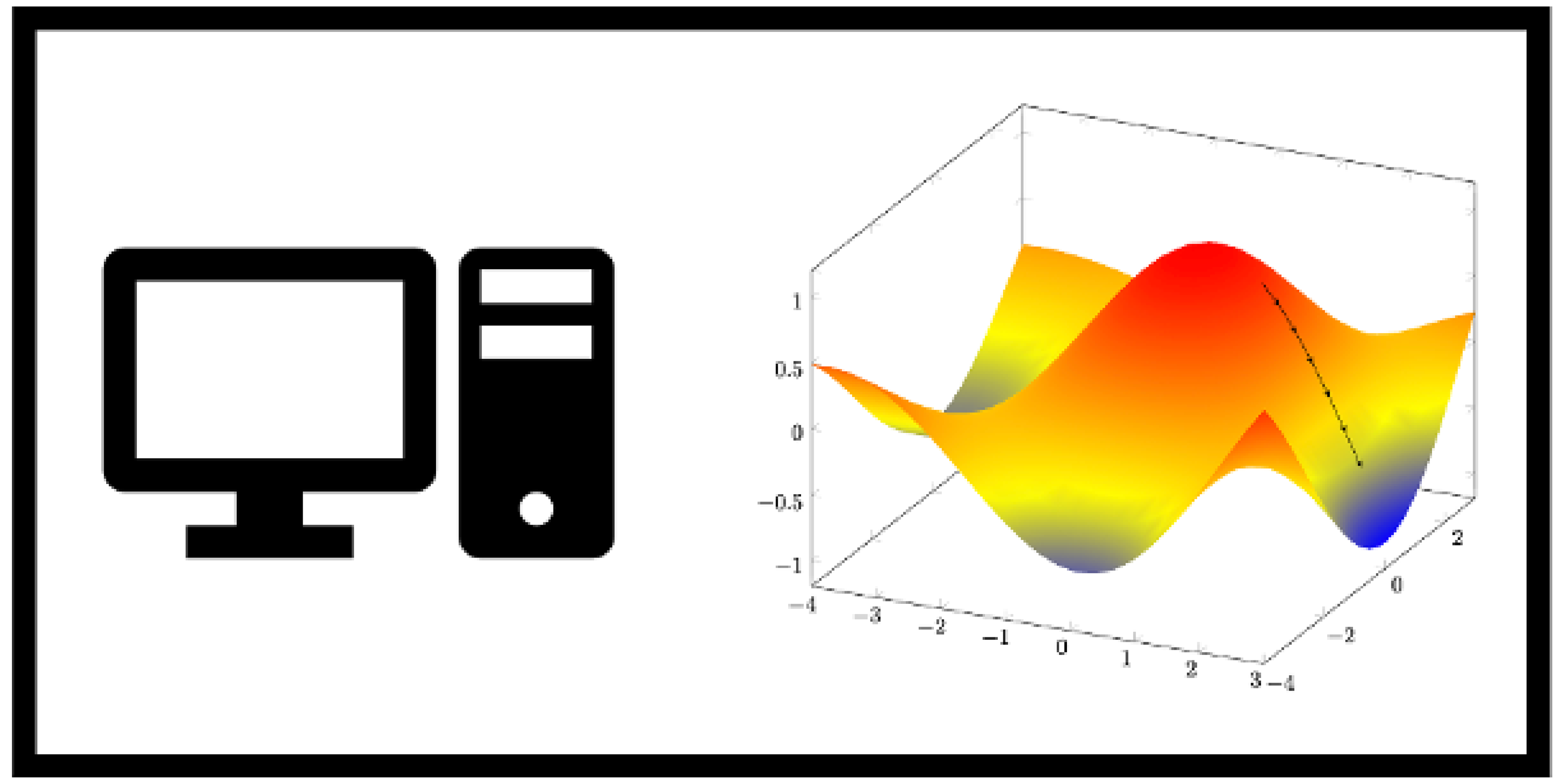
ancilla



Probability

Update parameters

Classical optimizer



Nature Comm., (2018) 9; 4195

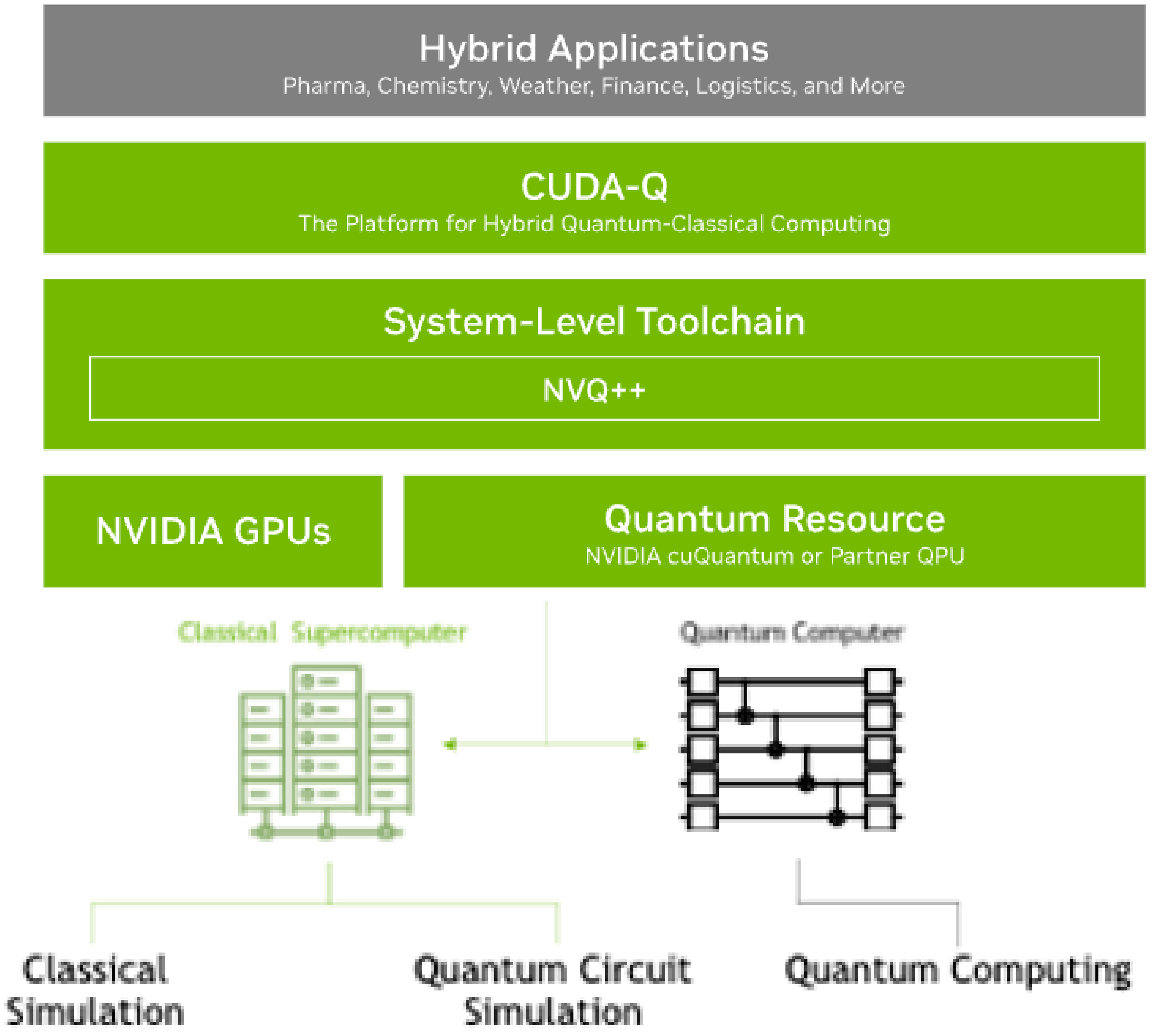
$$P(\mathbf{x}) = \frac{\sum_{\{h\}} e^{\sum_i a_i \sigma_i^z + \sum_j b_j h_j + \sum_{ij} w_{ij} \sigma_i^z h_j}}{\sum_{x'} \sum_{\{h\}} e^{\sum_i a_i \sigma_i^z + \sum_j b_j h_j + \sum_{ij} w_{ij} \sigma_i^z h_j}}$$

$$|\varphi\rangle = \sum_x \sqrt{P(x)} \ |x\rangle$$

$$E = \langle \varphi | \hat{H} | \varphi \rangle$$

Various industrial and technological applications.

Nvidia CUDA-Q: a platform for quantum-classical computing



https://developer.nvidia.com/cuda-q

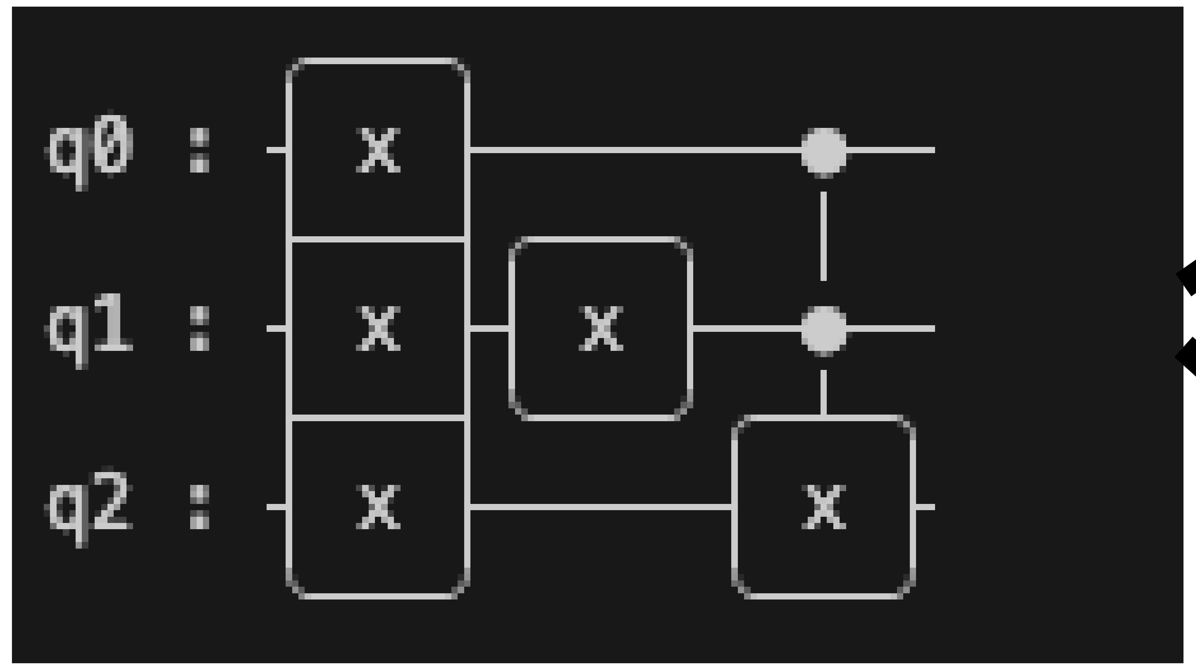
Nvidia CUDA-Q Features

- ✓ Single resource Python and C++ programming model
- ✓ High performance compiler for hybrid GPU/CPU/QPU systems
- ✓ QPU agnostic - works with any type of QPU, emulated or physical
- ✓ Supports both state-vector and tensor network backend
- ✓ Interoperable with leading scientific computing and AI tools

Quantum Computing Partners

ALICE & BOB, atom computing, IONQ, IQM, QuEra Computing Inc., ORCA Computing, OQC, PASQAL, qBraid, QCWARE, QUANTINUUM, QUANTUM BRILLIANCE, QUANTUM MACHINES, rigetti, seeqc, Terra Quantum, XANADU, ZAPATA

Building hybrid applications with CUDA-Q on HPC



Python

```python
import cudaq

@cudaq.kernel
def kernel():
    qvector = cudaq.qvector(3)
    x(qvector)
    x(qvector[1])
    x.ctrl([qvector[0], qvector[1]], qvector[2])
    mz(qvector)
```

Add a decorator to the function.

C++

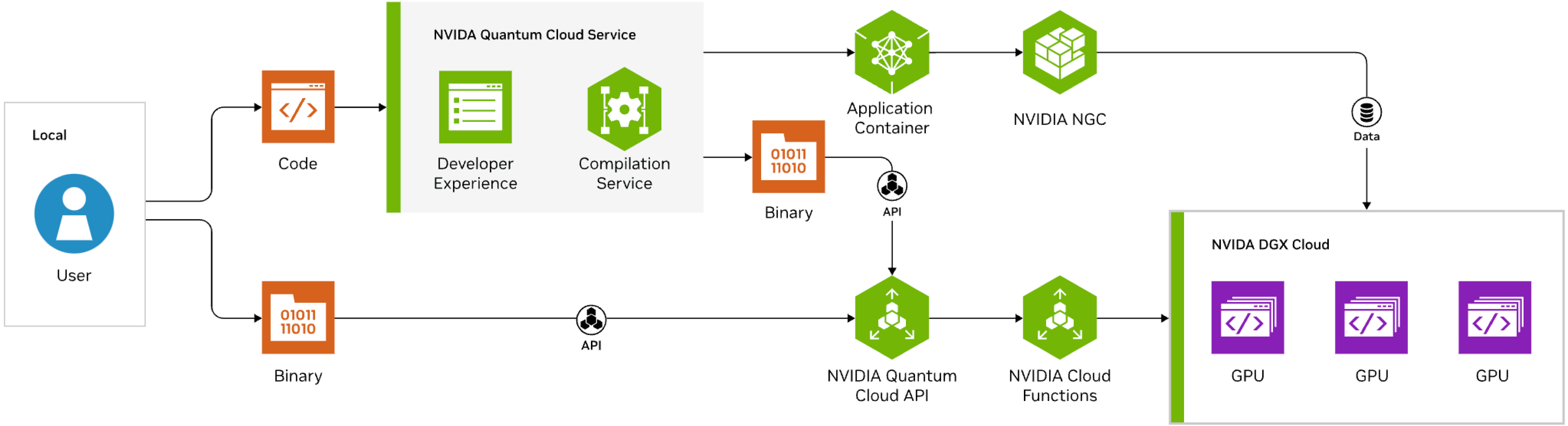
```cpp
struct ApplyX {
  void operator()(cudaq::qubit &q) __qpu__ { x(q); }
};

struct ccnot_test {
  // constrain the signature of the incoming kernel
  void operator()(cudaq::takes_qubit auto &&apply_x) __qpu__ {
    cudaq::qvector qs(3);

    x(qs);
    x(qs[1]);

    // Control U (apply_x) on the first two qubits of
    // the allocated register.
    cudaq::control(apply_x, qs.front(2), qs[2]);

    mz(qs);
  }
};
```

CUDA quantum requires the **__qpu__** function attribute for quantum kernel declaration.

To learn more about the quantum kernel in CUDA-Q visit:

https://nvidia.github.io/cuda-quantum/0.7.0/index.html

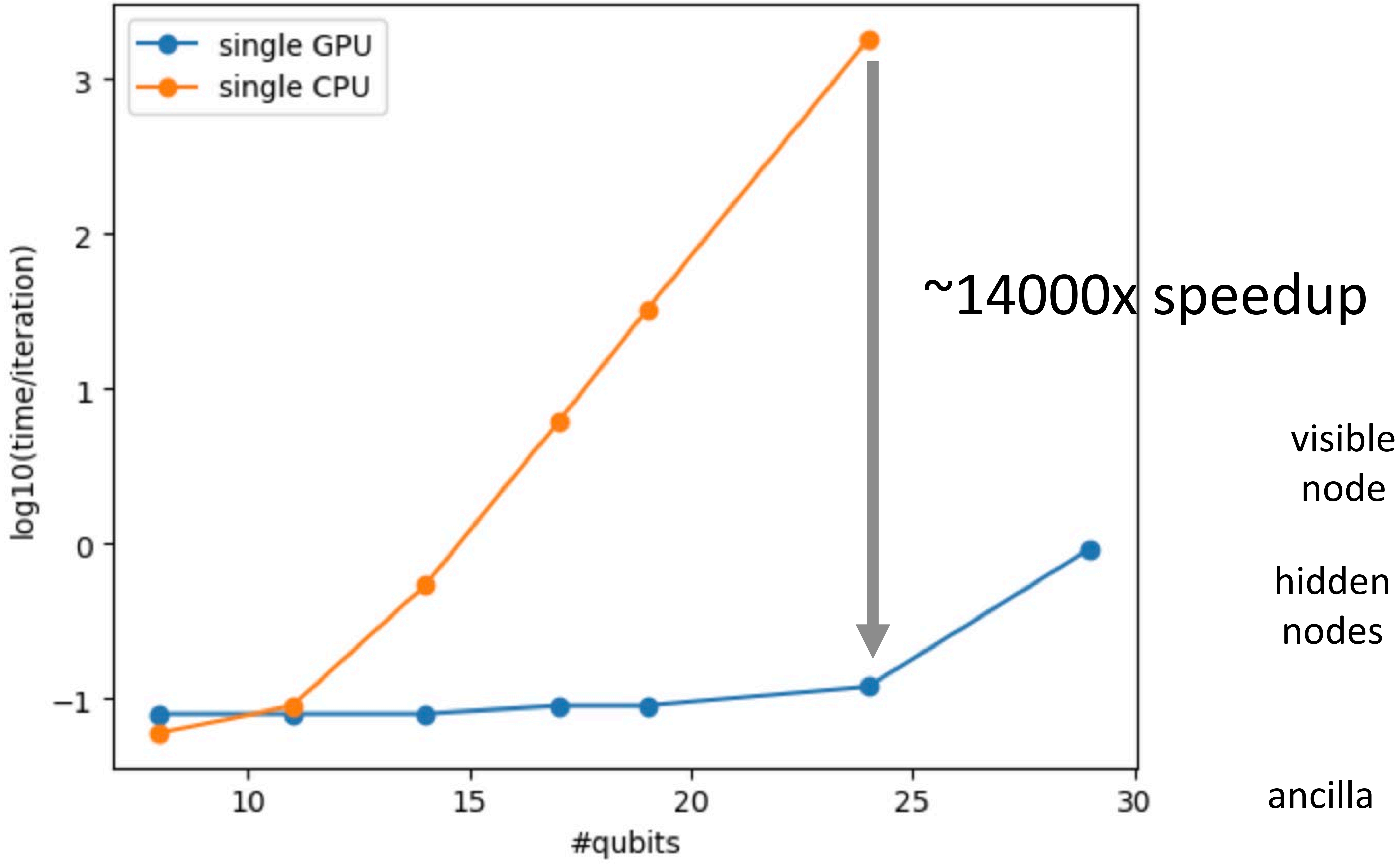
Nvidia quantum cloud

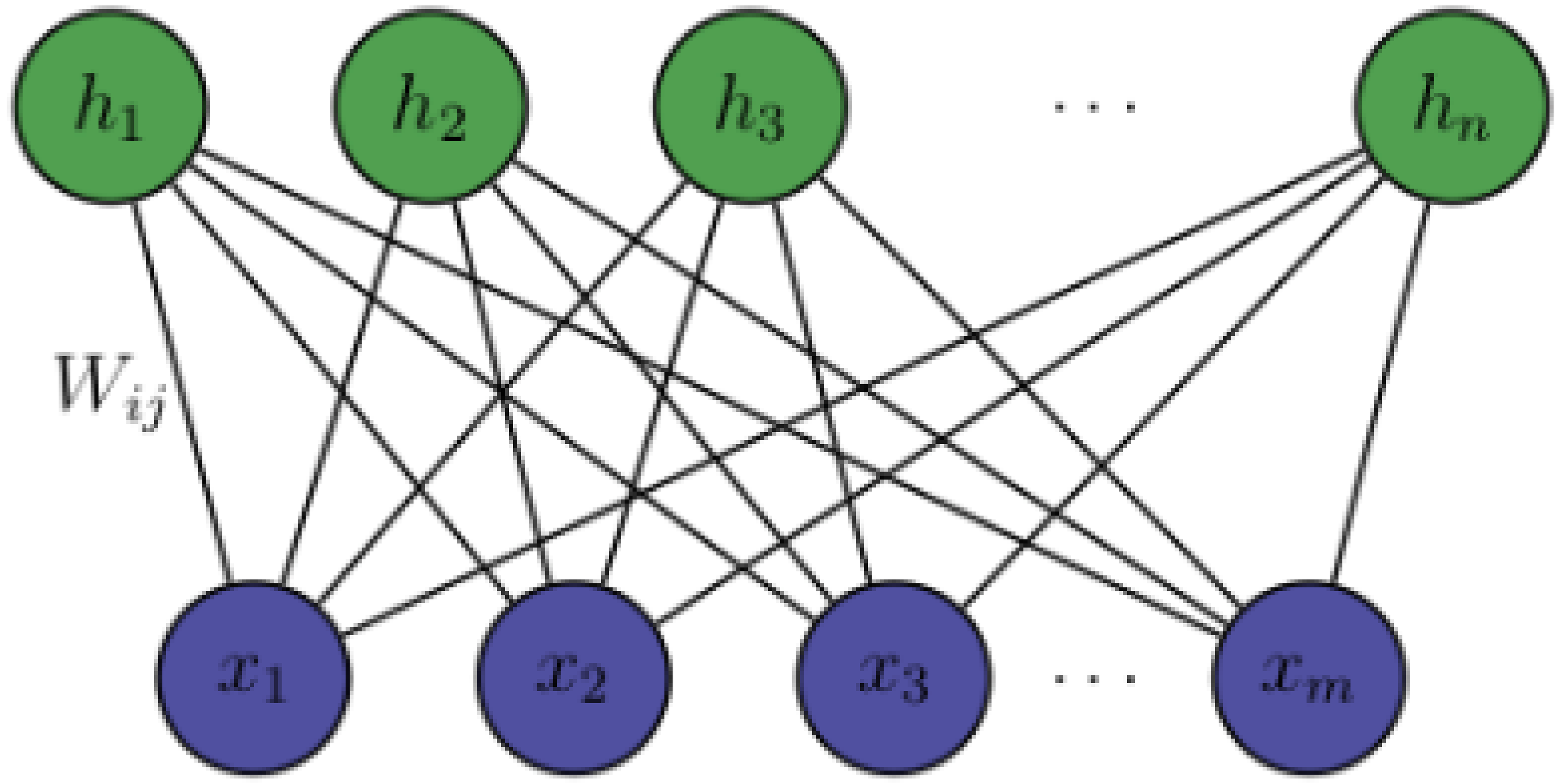


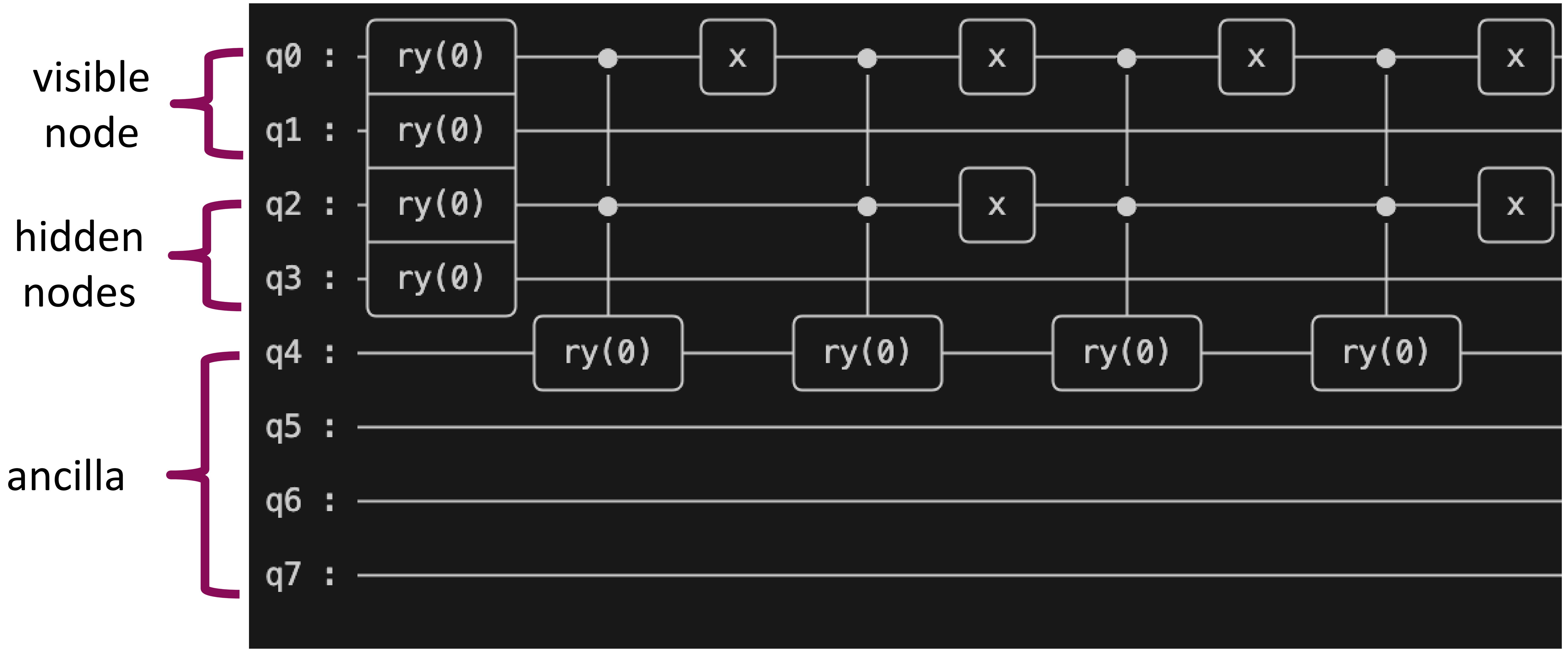
- ✓ Access to the most powerful quantum resource
- ✓ Develop locally, run any CUDA-Q app seamlessly in the cloud
- ✓ Run workloads on GPU supercomputers
- ✓ Apply for access at https://www.nvidia.com/en-eu/solutions/quantum-computing/cloud/

Accelerating the Q-RBM circuit simulations with CUDA-Q

Sampling the Gibbs distribution using the Q-RBM (#shots=10000)







GPU: Nvidia A100

CPU: AMD EPYC 7742 64-Core Processor

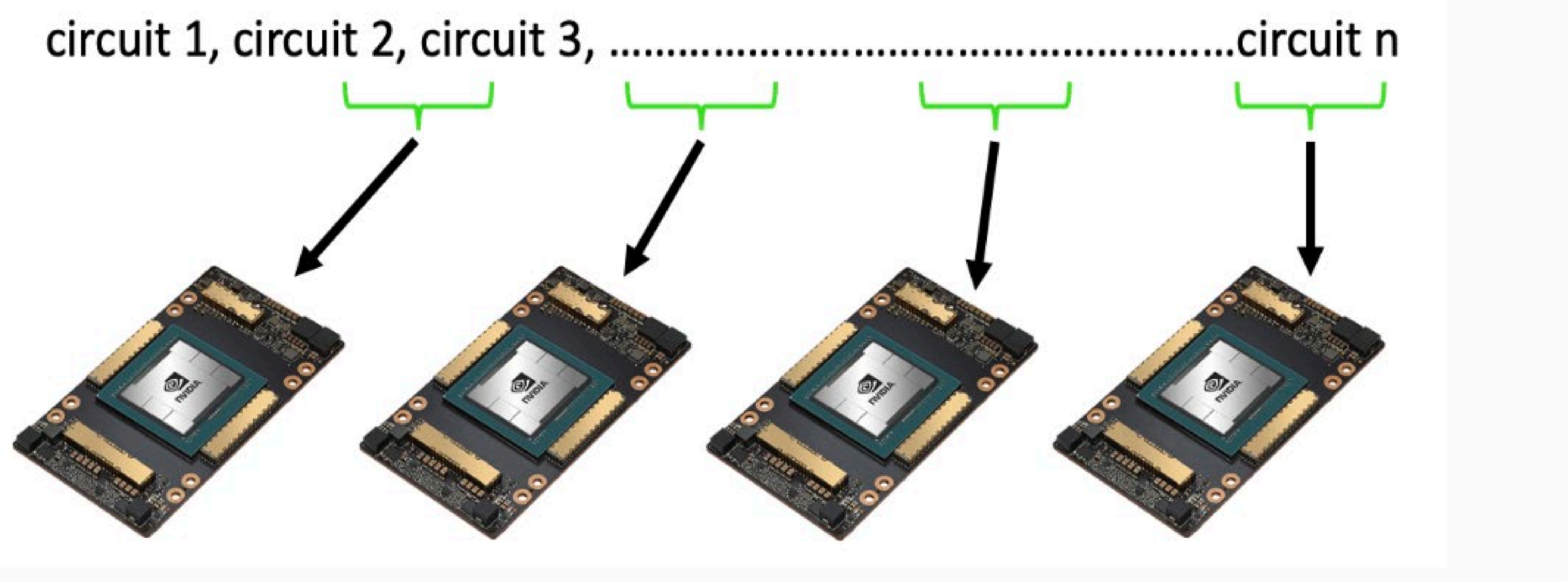
Accelerating Q-RBM circuit simulations with CUDA-Q using multi-GPUs

Single GPU:

```
cudaq.set_target("nvidia")
```

```
start_time = timeit.default_timer()
result = cudaq.sample(main_kernel,theta,shots_count=1000000)
end_time = timeit.default_timer()
```

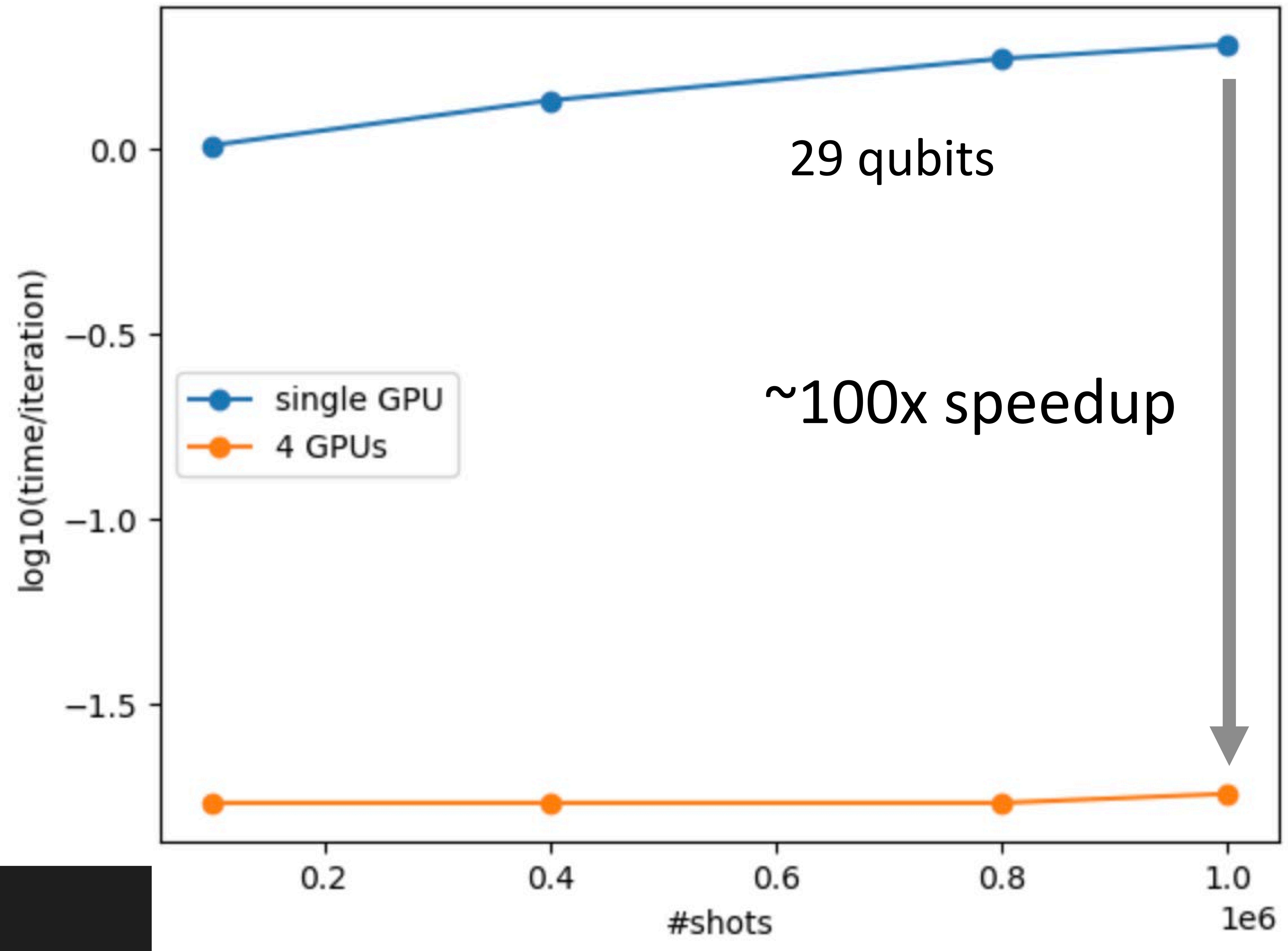
Multi-GPUs:




```
cudaq.set_target("nvidia-mqpu")
target = cudaq.get_target()
qpu_count = target.num_qpus()
```

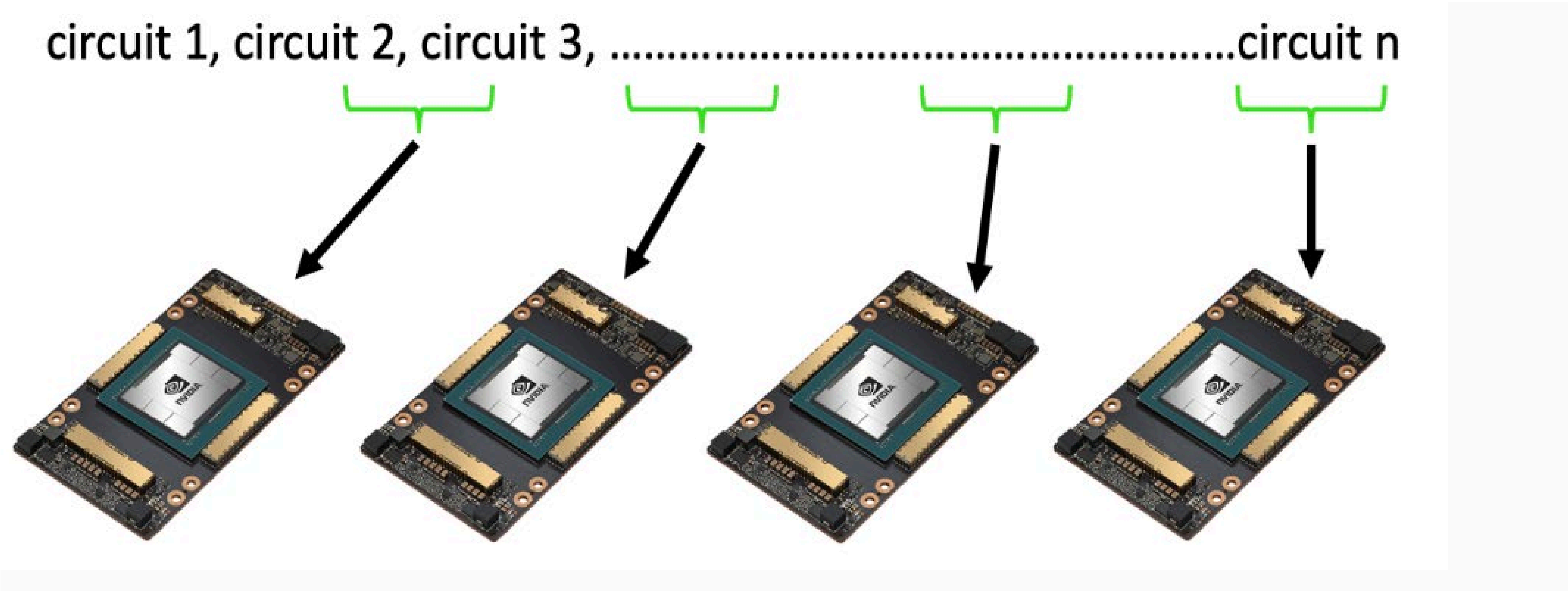
```
start_time = timeit.default_timer()
count_futures = []
for qpu in range(qpu_count):
    count_futures.append(cudaq.sample_async(main_kernel,theta,shots_count=1000000, qpu_id=qpu))
end_time = timeit.default_timer()
```

Sampling the Gibbs distribution: single GPU vs multi-GPUs

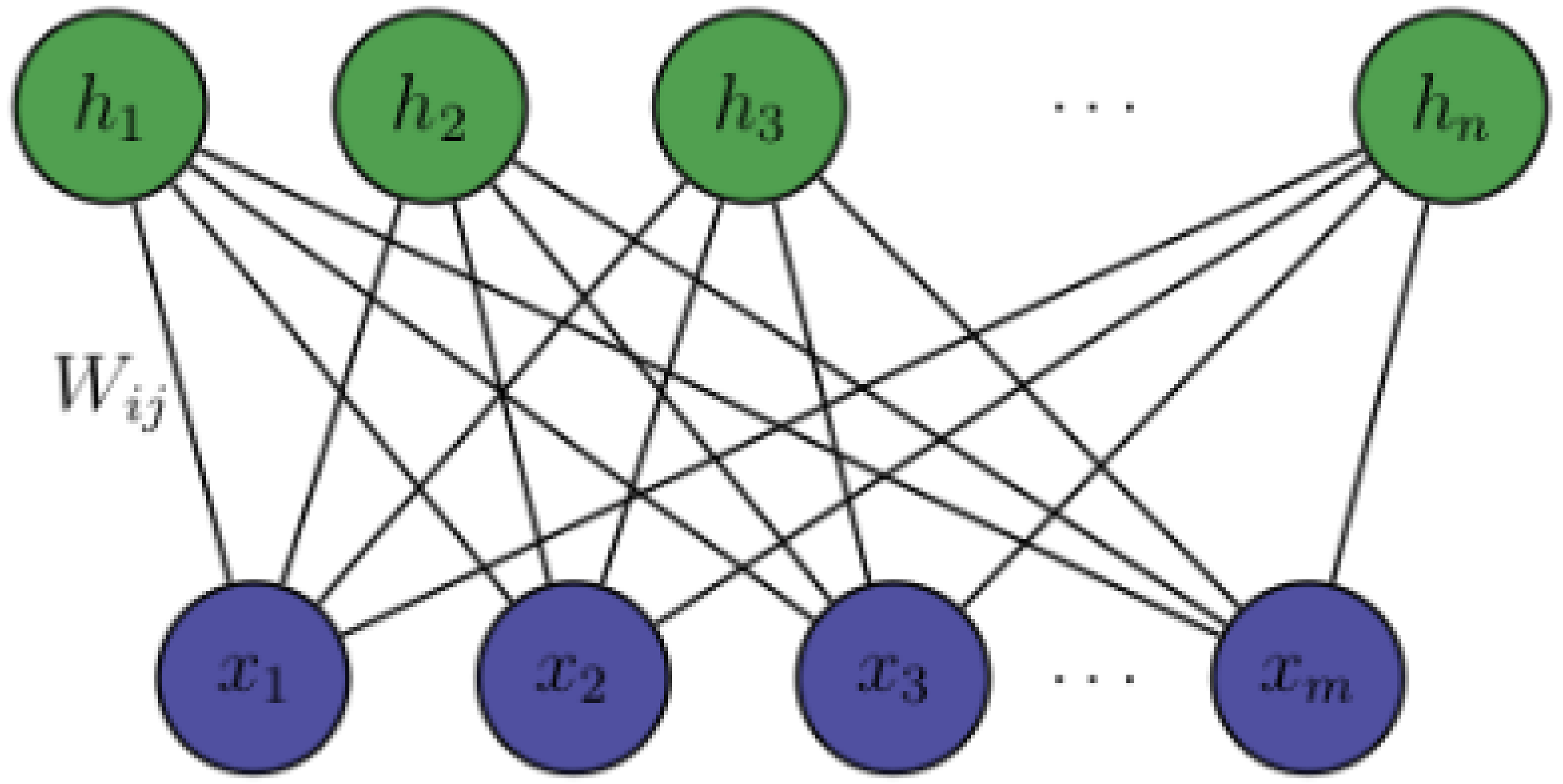


GPU: Nvidia A100

Scaling up Q-RBM circuit simulations with CUDA-Q using multi-GPUs




```
cudaq.set_target("nvidia-mgpu")
```



- n qubits has 2^n complex amplitudes
- Each requires 8 bytes of memory to store.

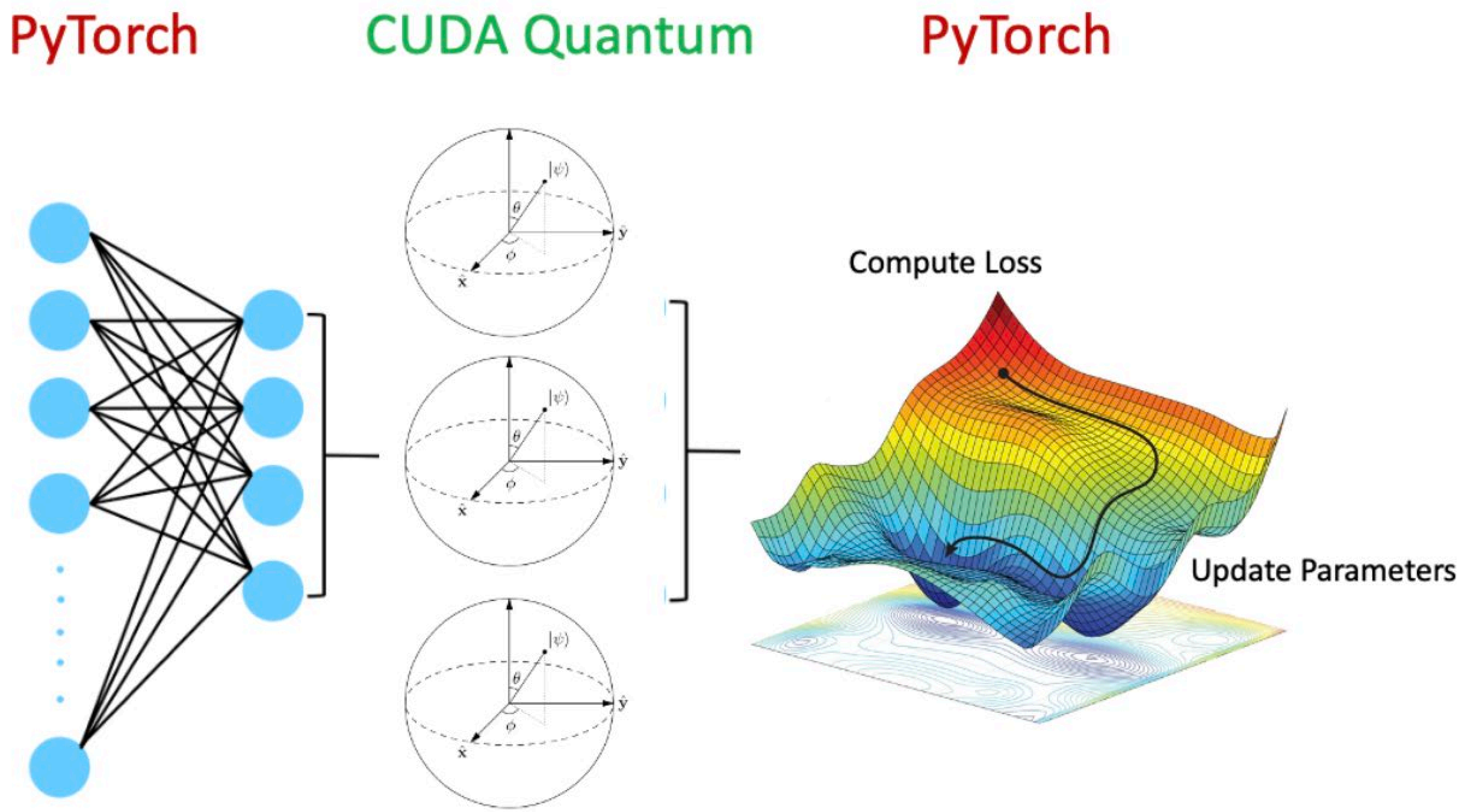
- For 30 qubits: 8 bytes x 2^{30} = 8.6 GB

- For 34 qubits ~137 GB

- Number of visible nodes:10
- Number of hidden nodes: 23
- Number of ancilla qubits (reuse ancilla): 1
- #qubits: 34 qubits

Elapsed time for sampling each circuit with **34 qubits: 4.73 sec. (#shots: 10000)**

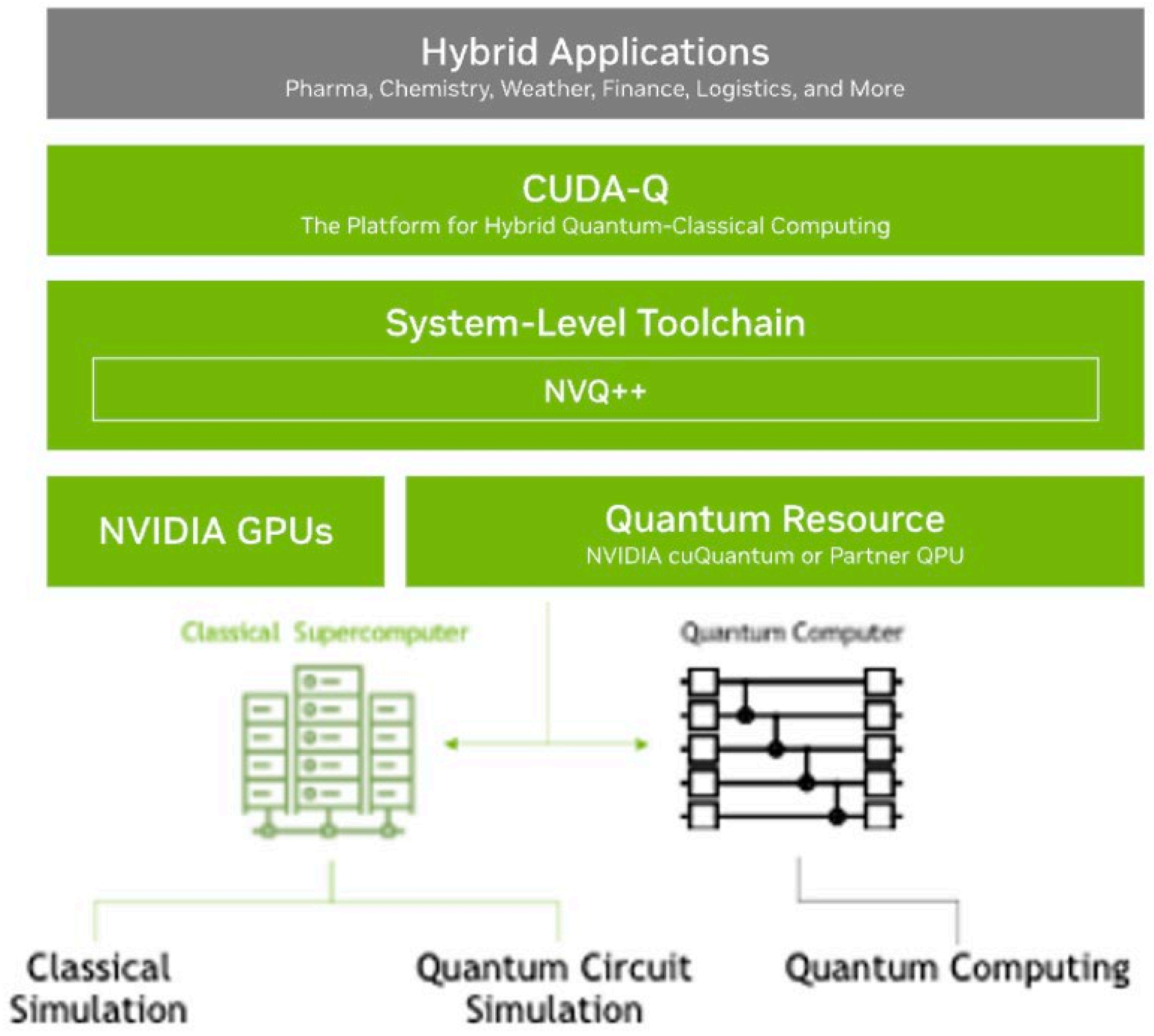
Other applications: using PyTorch with CUDA-Q

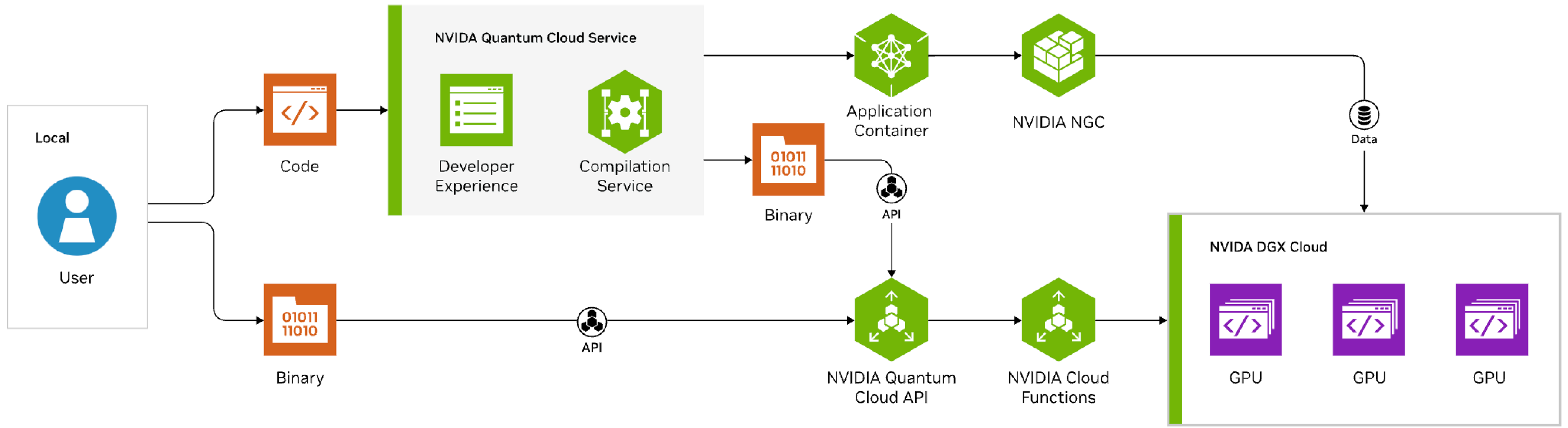


Zohim Chandani

To learn more about this application visit:
https://nvidia.github.io/cuda-quantum/0.7.0/examples/python/tutorials/hybrid_qnns.html

Conclusion

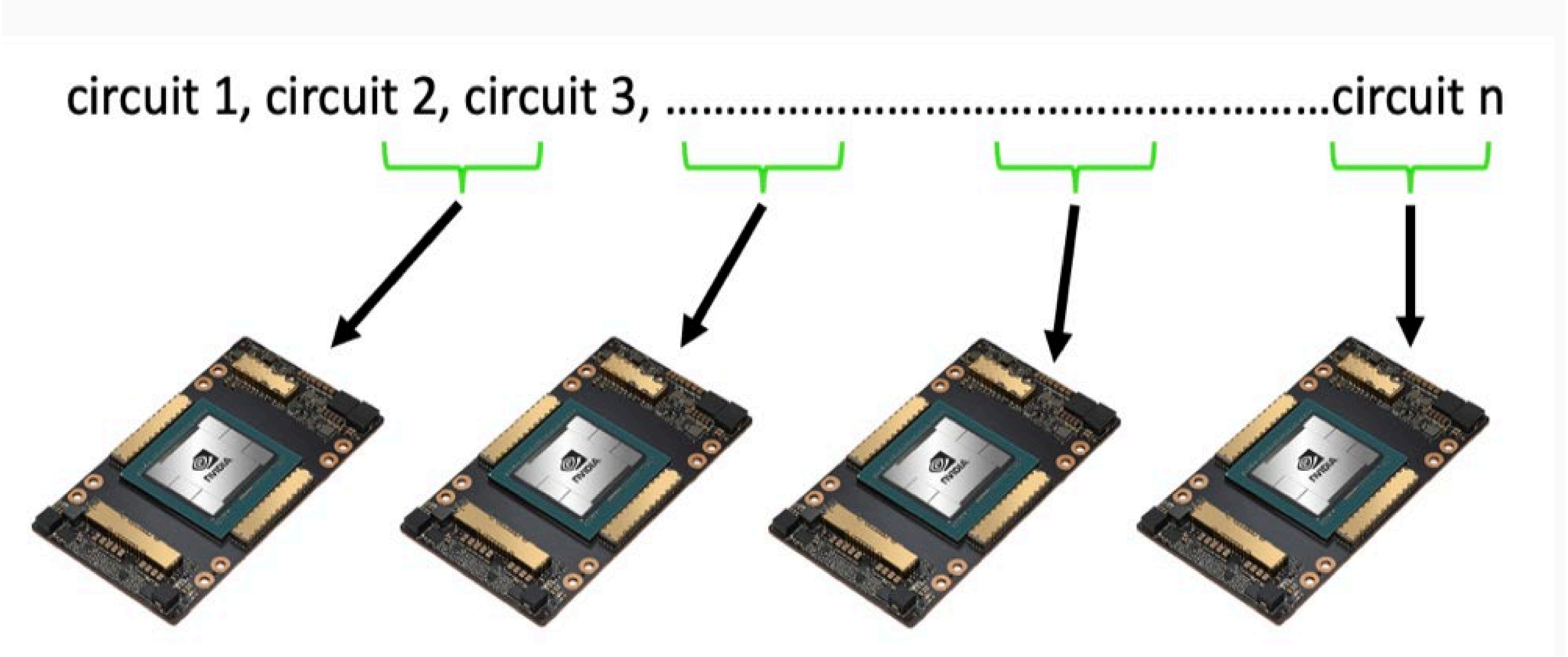




Nvidia Quantum Cloud

CUDA-Q:

- ❑ A platform for quantum classical computing.
- ❑ Supports both state-vector and tensor network.
- ❑ GPUs help accelerating simulation compared to CPUs.
- ❑ Mutli-GPUs will help scaling up the problem.
- ❑ QPU agnostic: it works on any QPUs emulated or physical.



NVIDIA