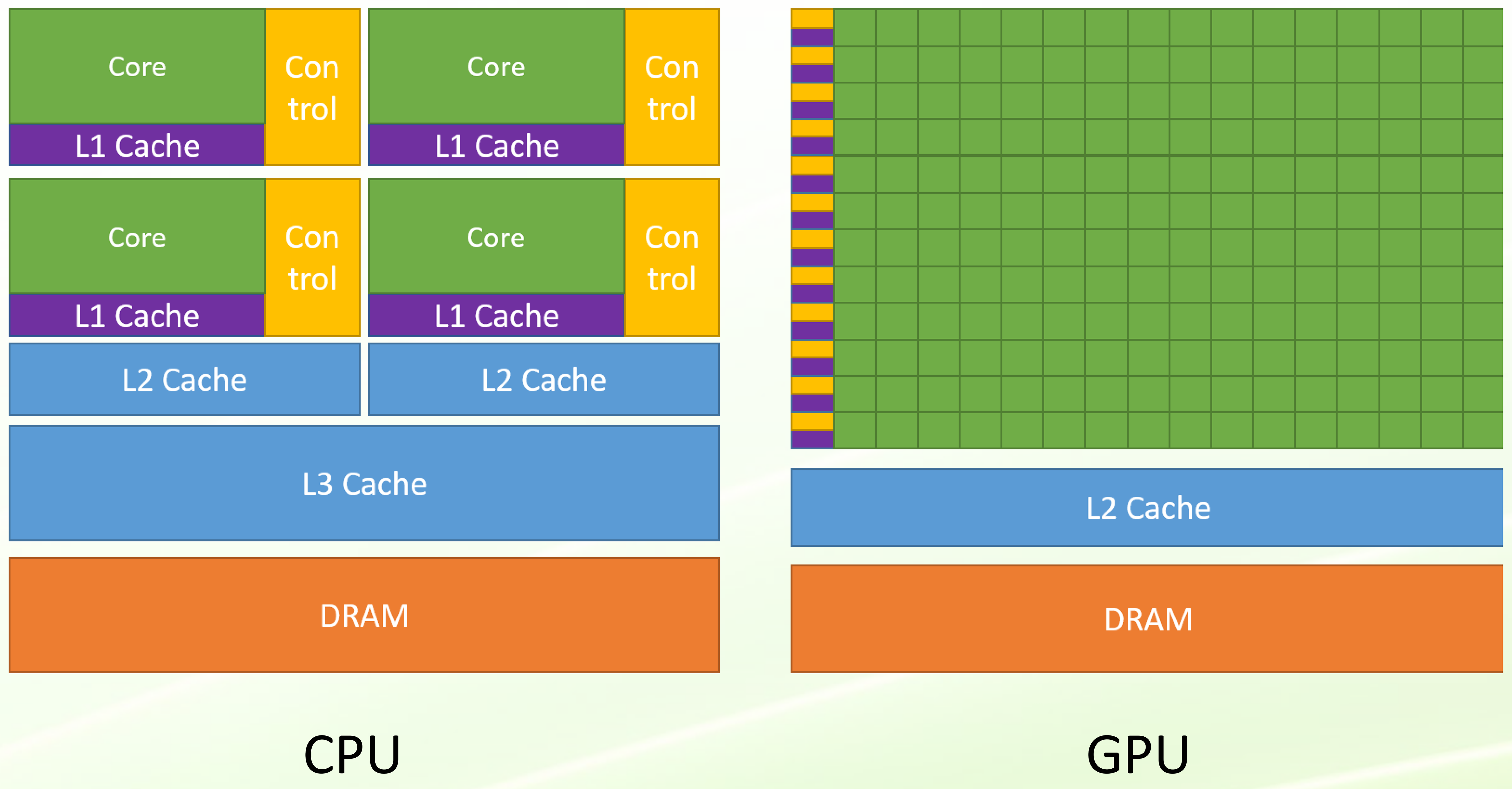
CUDA-Q and Quantum Accelerated Supercomputing

Monica VanDieren, Sr Technical Marketing Engineer | August 2024

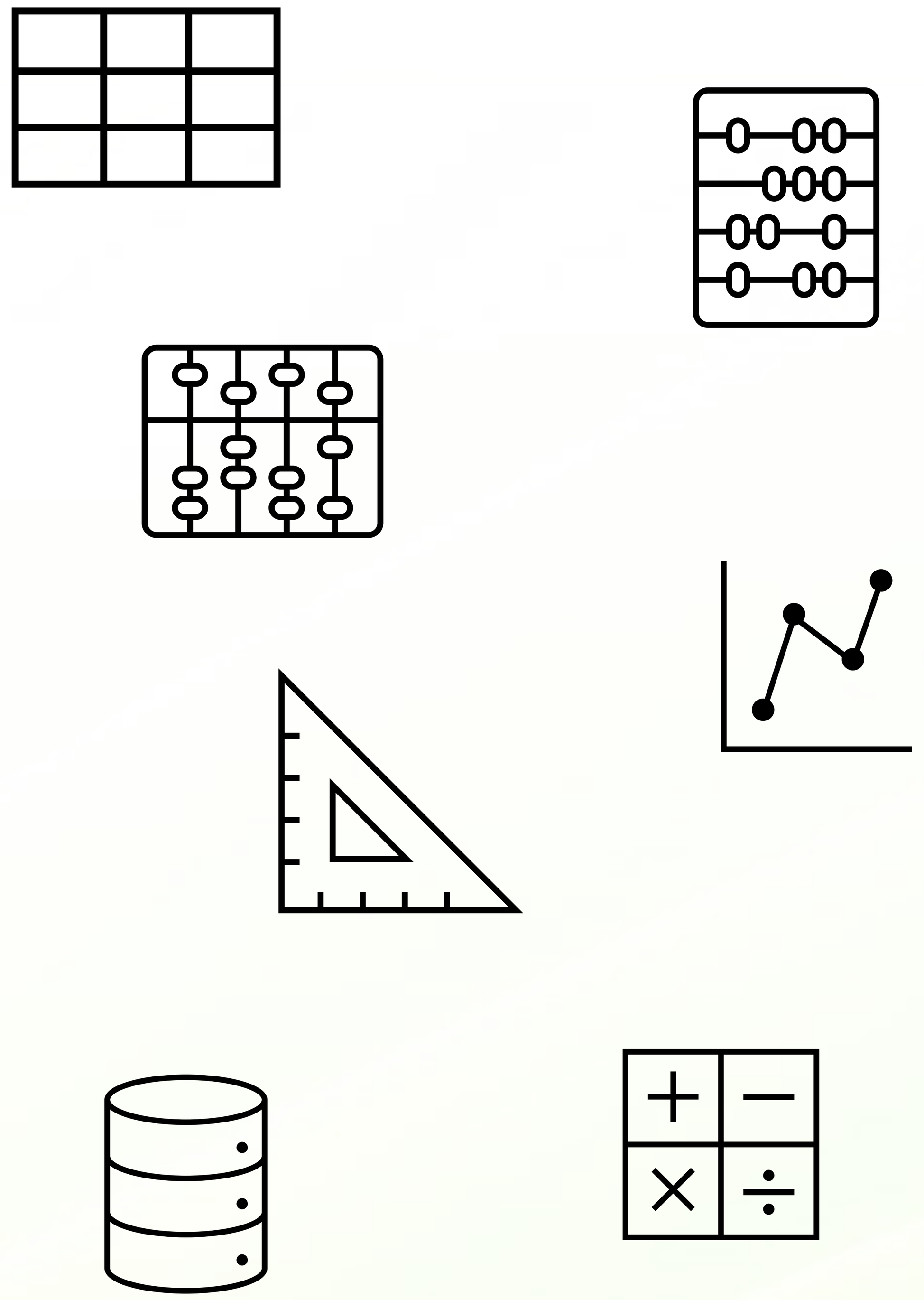
Agenda

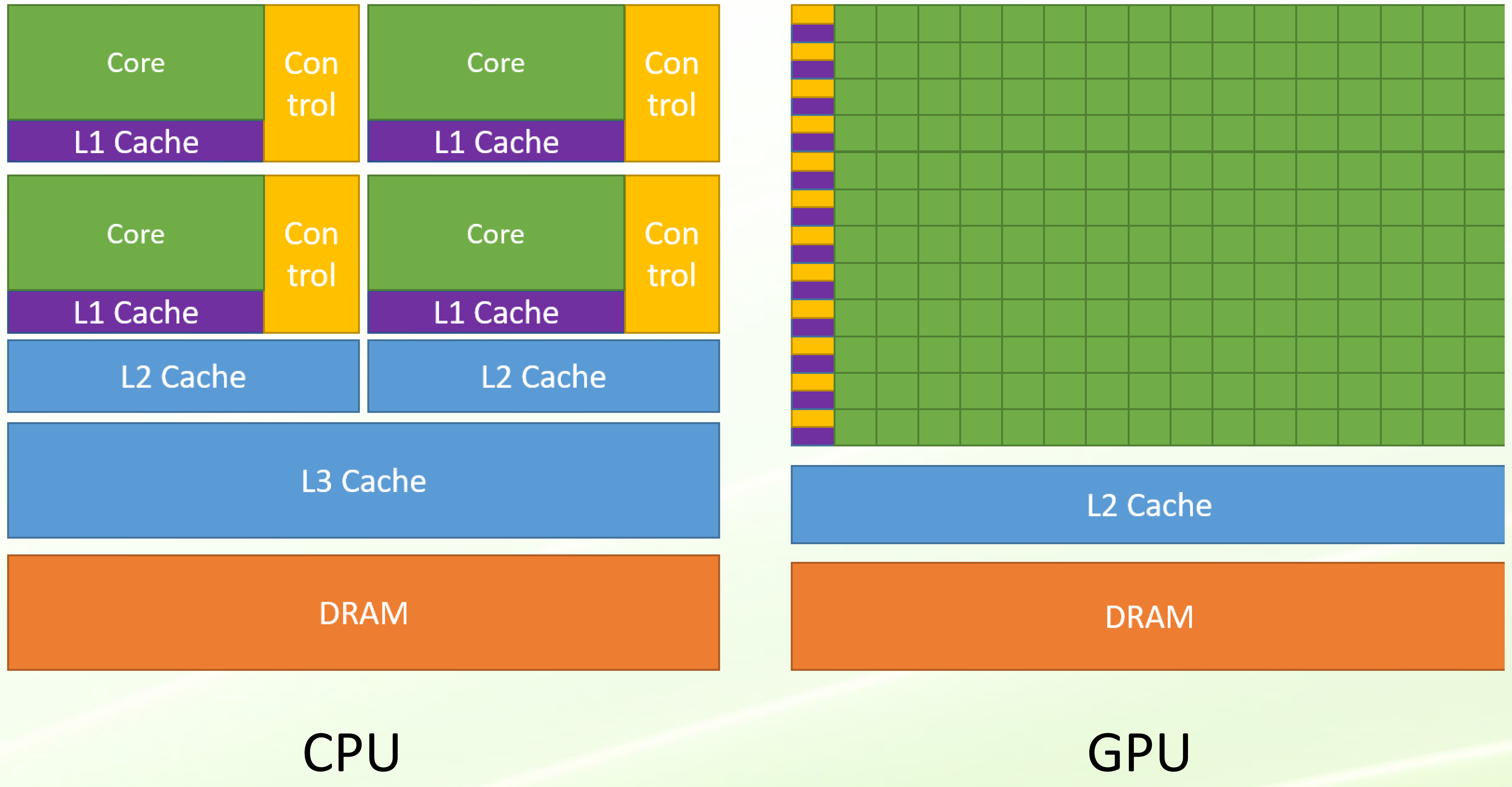
- What is ~~Quantum~~ Accelerated Supercomputing
- Useful Quantum Simulation
- How-to Guide to CUDA-Q
- Distributed Quantum Computing
- Conclusion

Accelerated Supercomputing

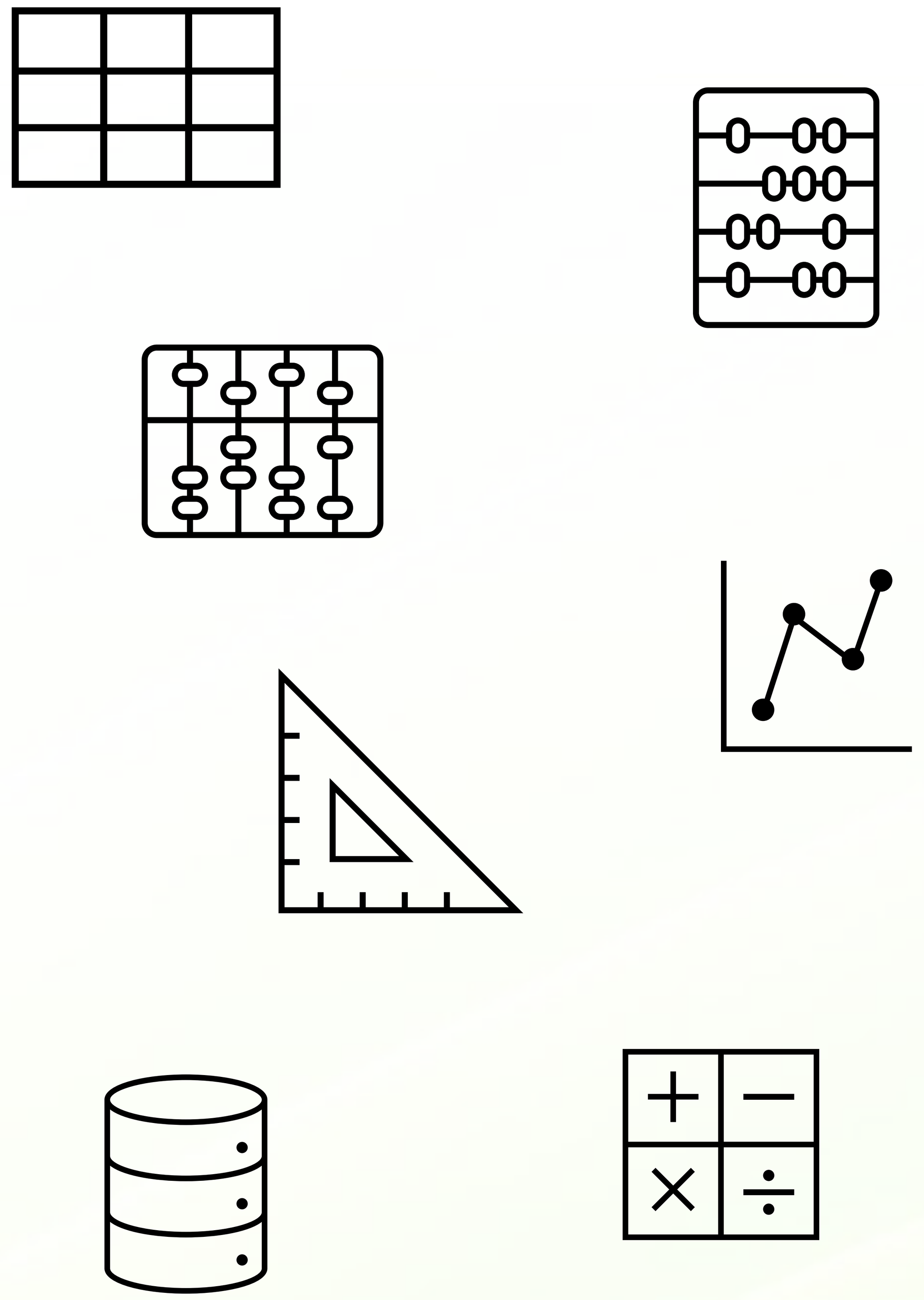


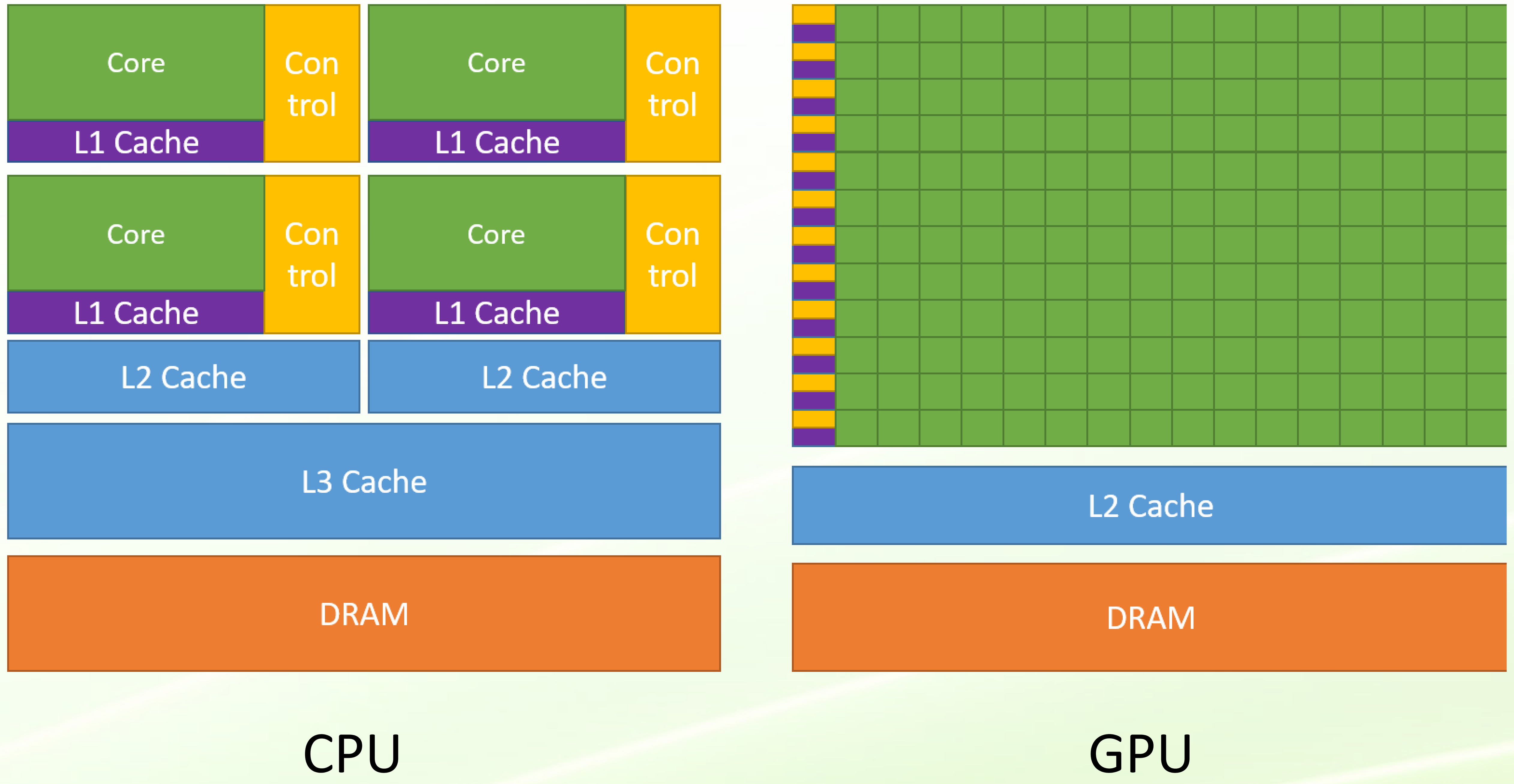
Accelerated Supercomputing

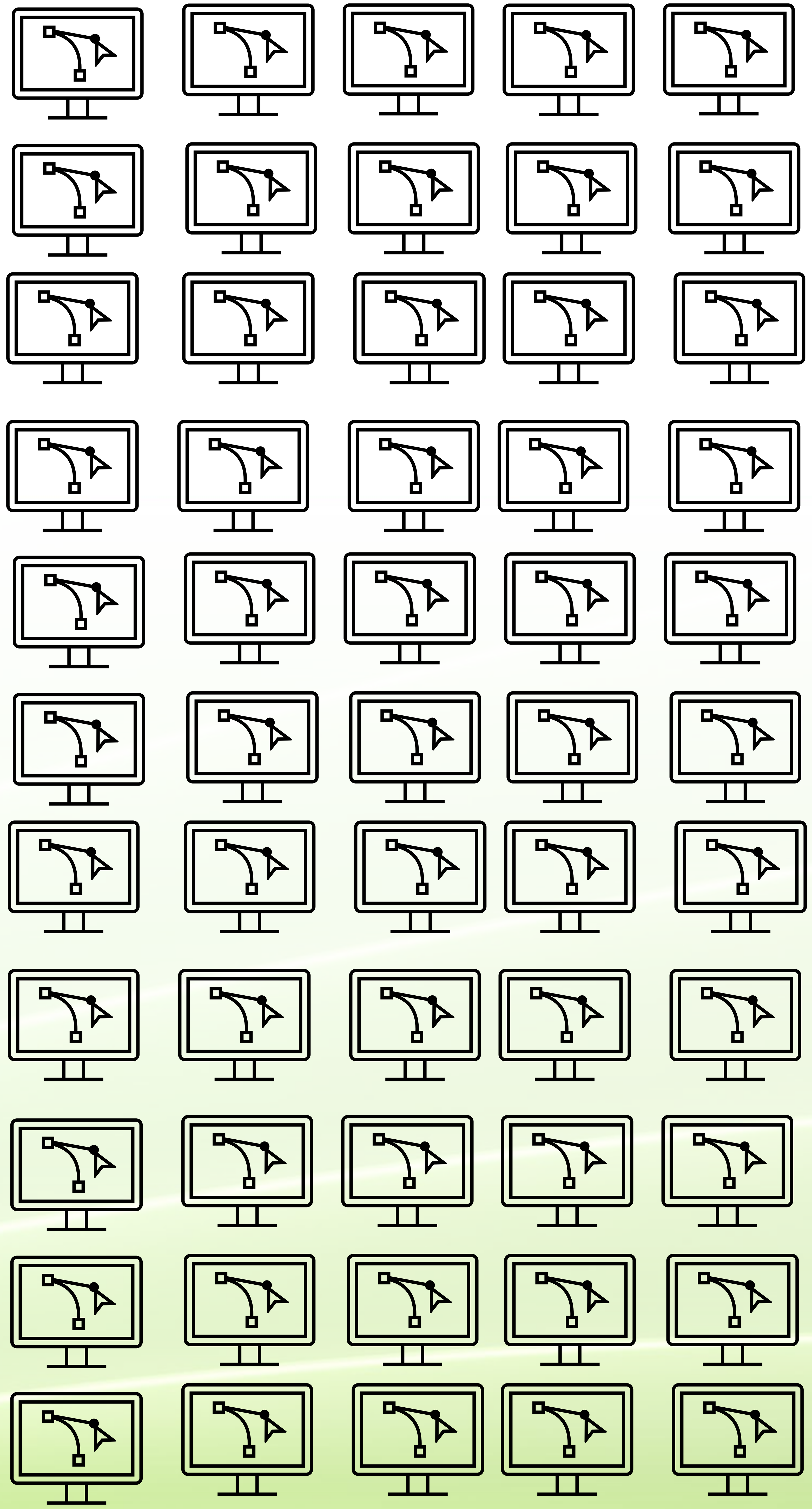




Accelerated Supercomputing

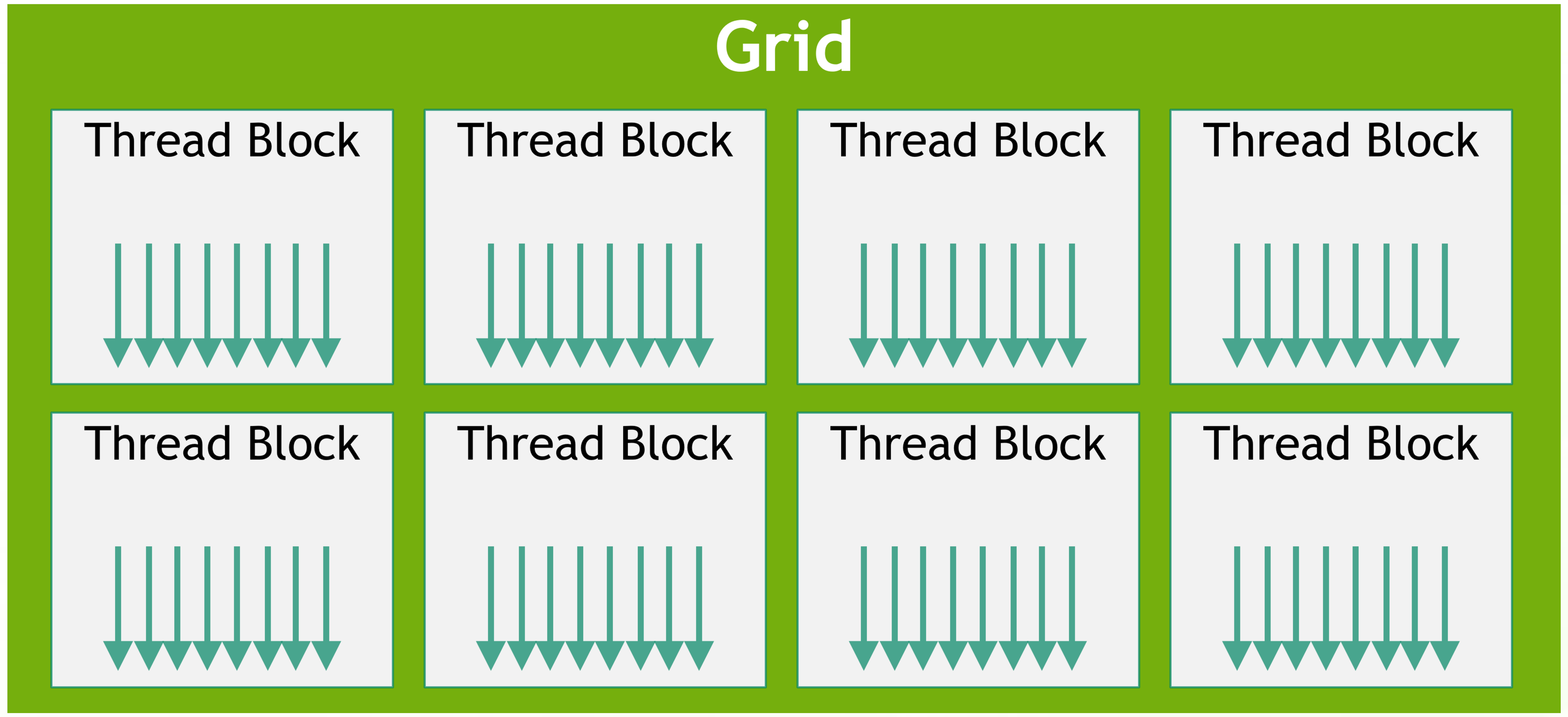


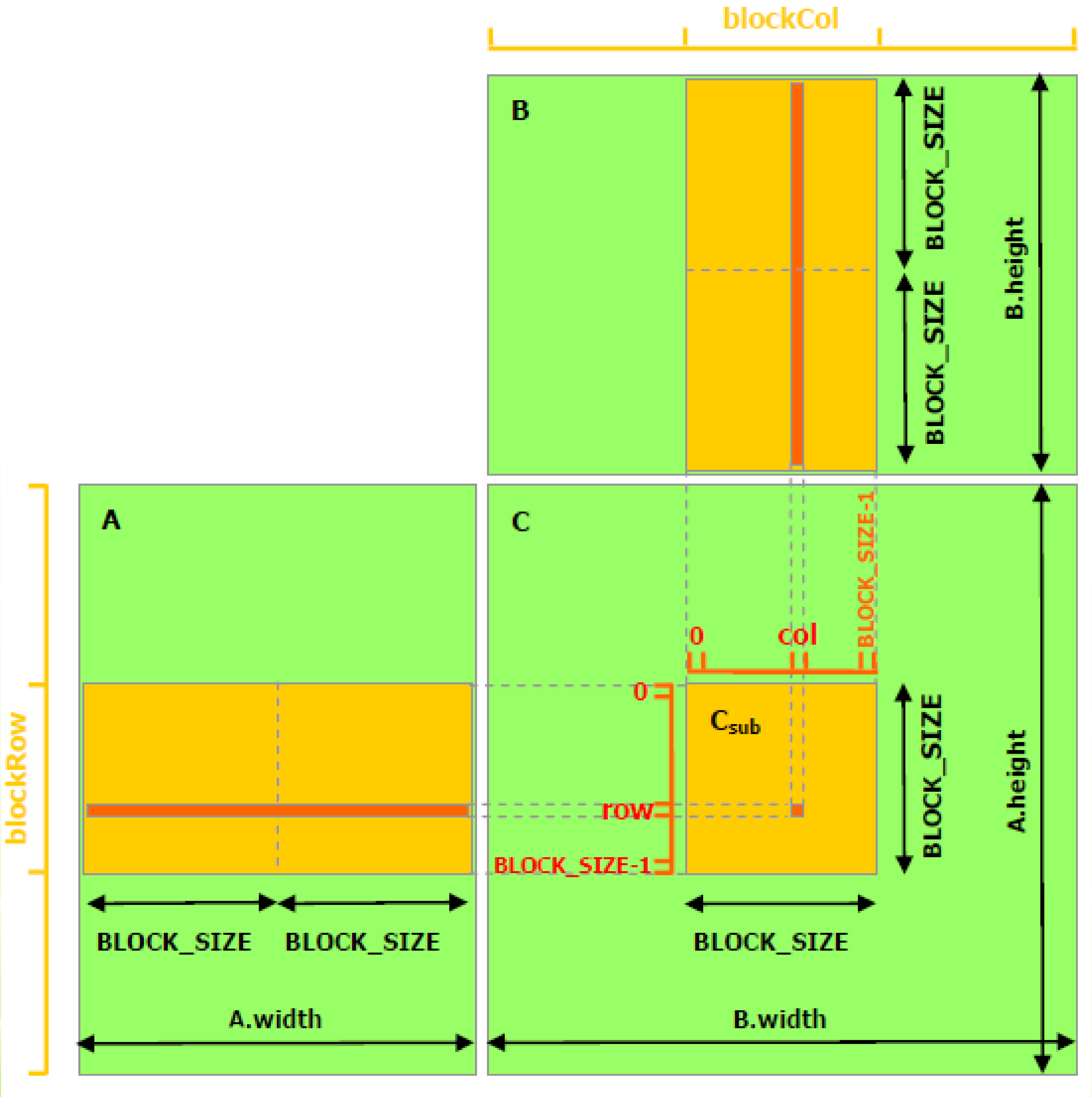




Matrix Multiplication in Parallel on a GPU

A x B = C





Matrix Multiplication in Parallel on a GPU

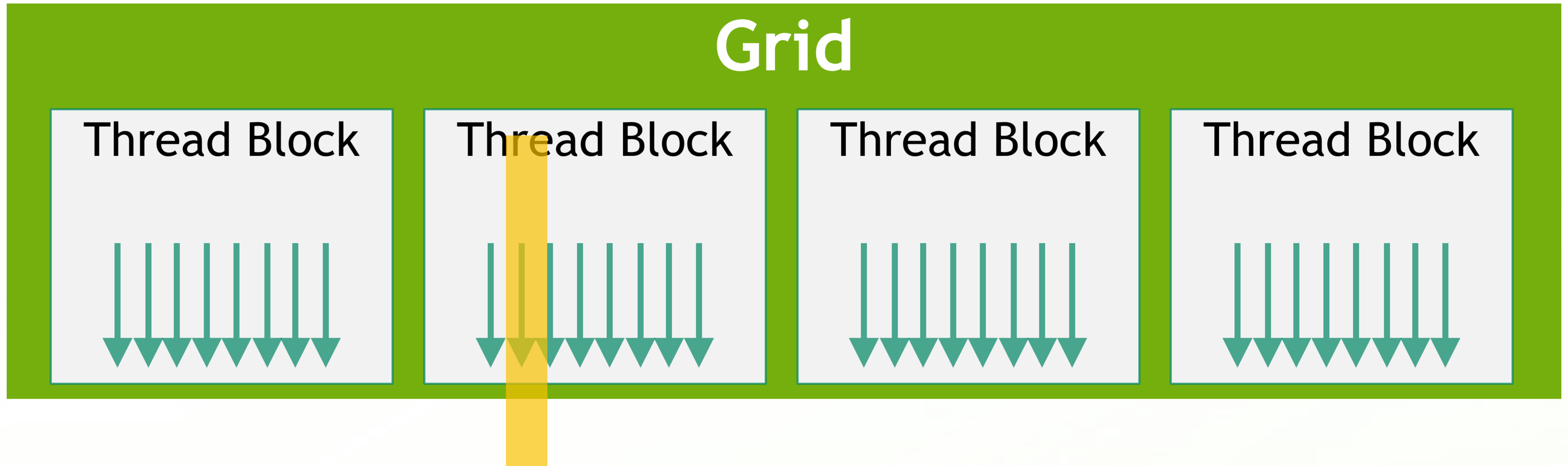
A x B = C

$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 \\ 12 & 11 & 10 & 9 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}
\begin{pmatrix} \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \end{pmatrix}$$

Matrix Multiplication in Parallel on a GPU

A x B = C



$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 \\ 12 & 11 & 10 & 9 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

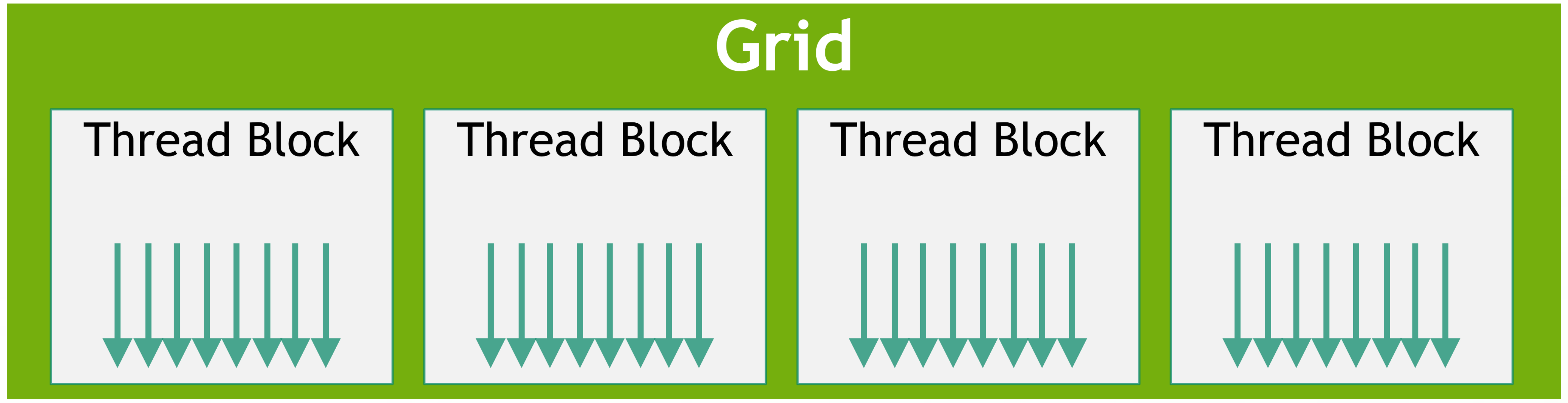
$$\begin{pmatrix} \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \end{pmatrix}$$

Kernel = instruction for each thread to follow

Kernel for matrix multiplication:
compute the dot product of
an assigned row in A with an assigned column of B

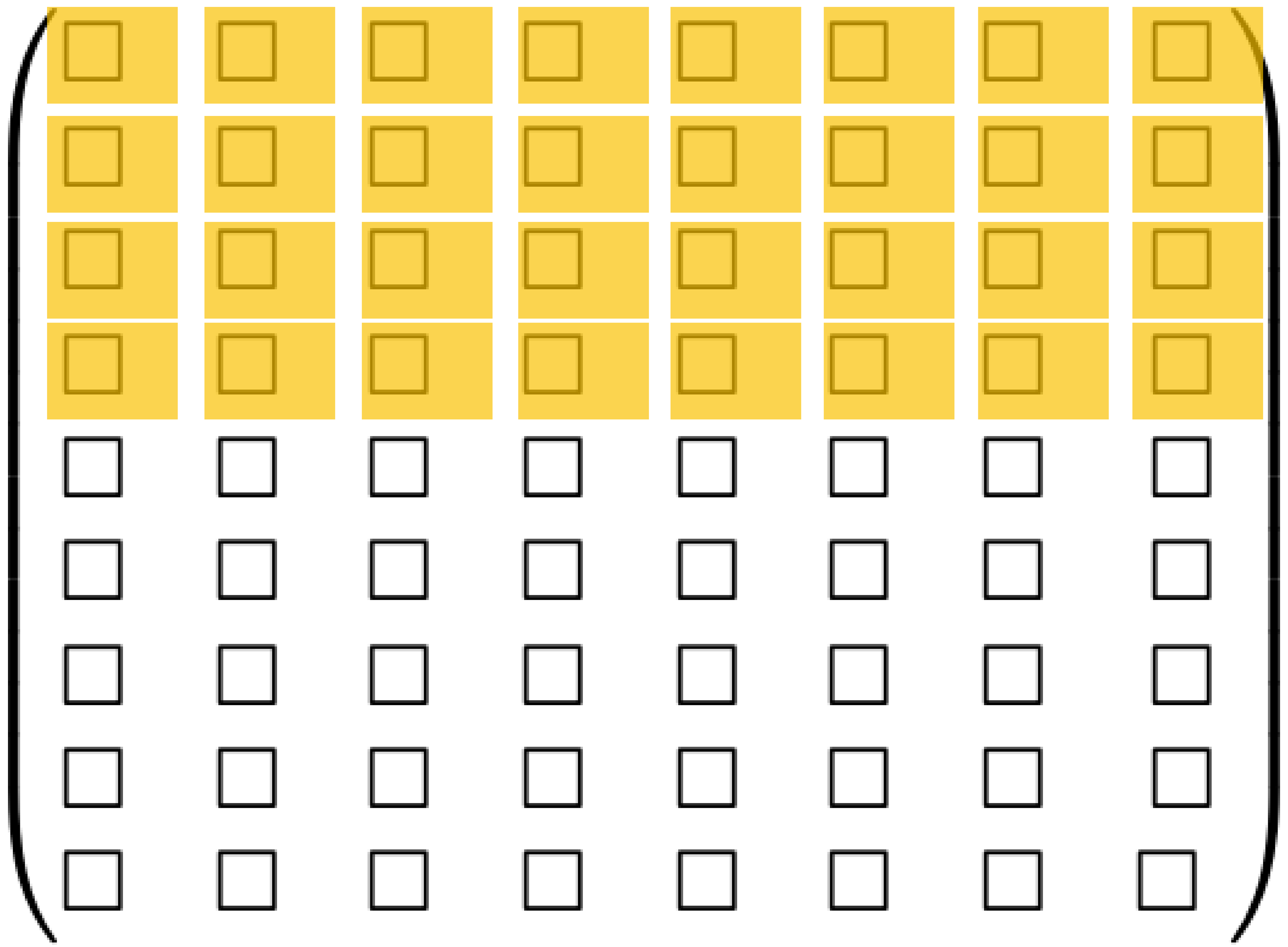
Matrix Multiplication in Parallel on a GPU

A x B = C



$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix}$$

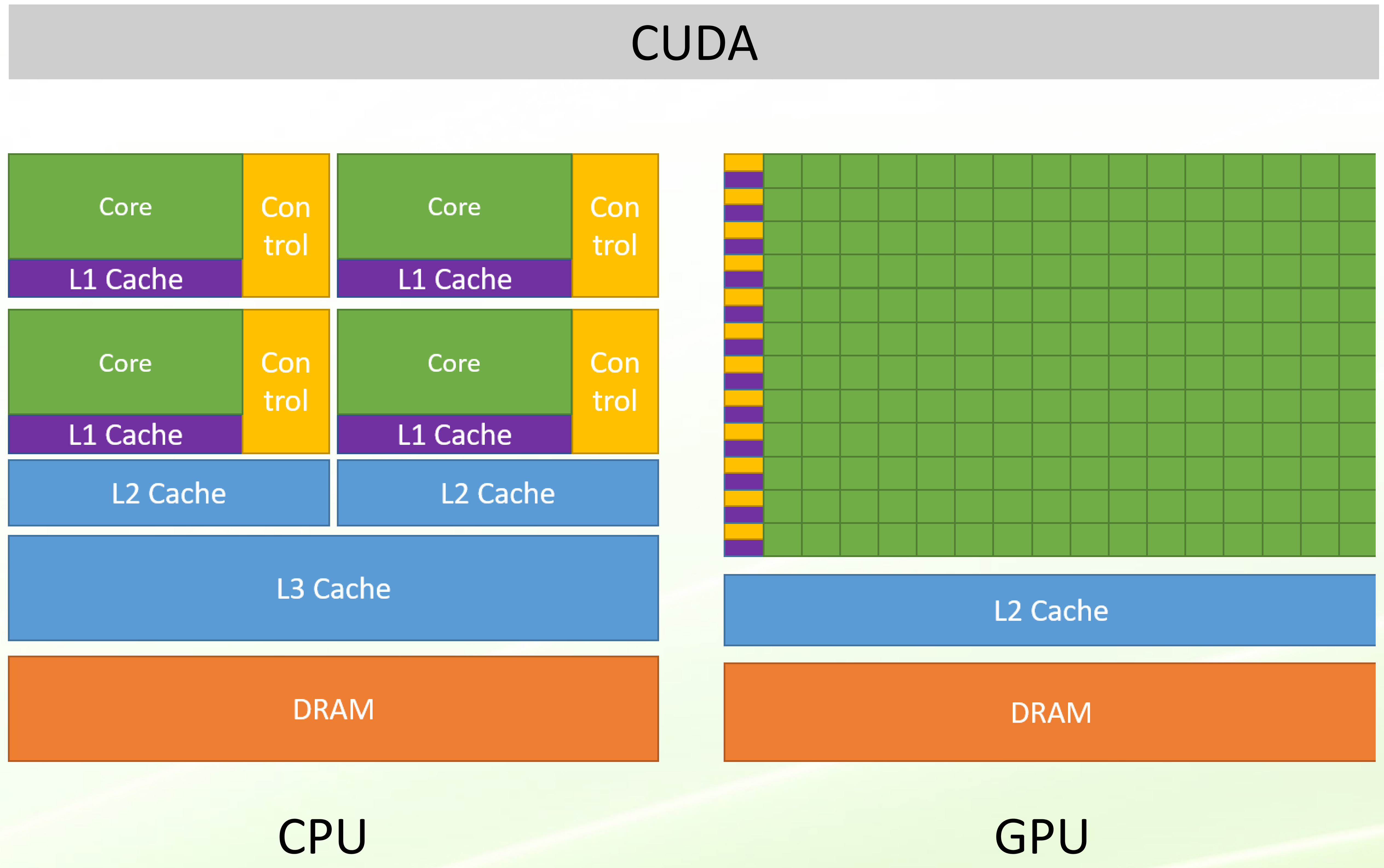
$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 \\ 12 & 11 & 10 & 9 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$



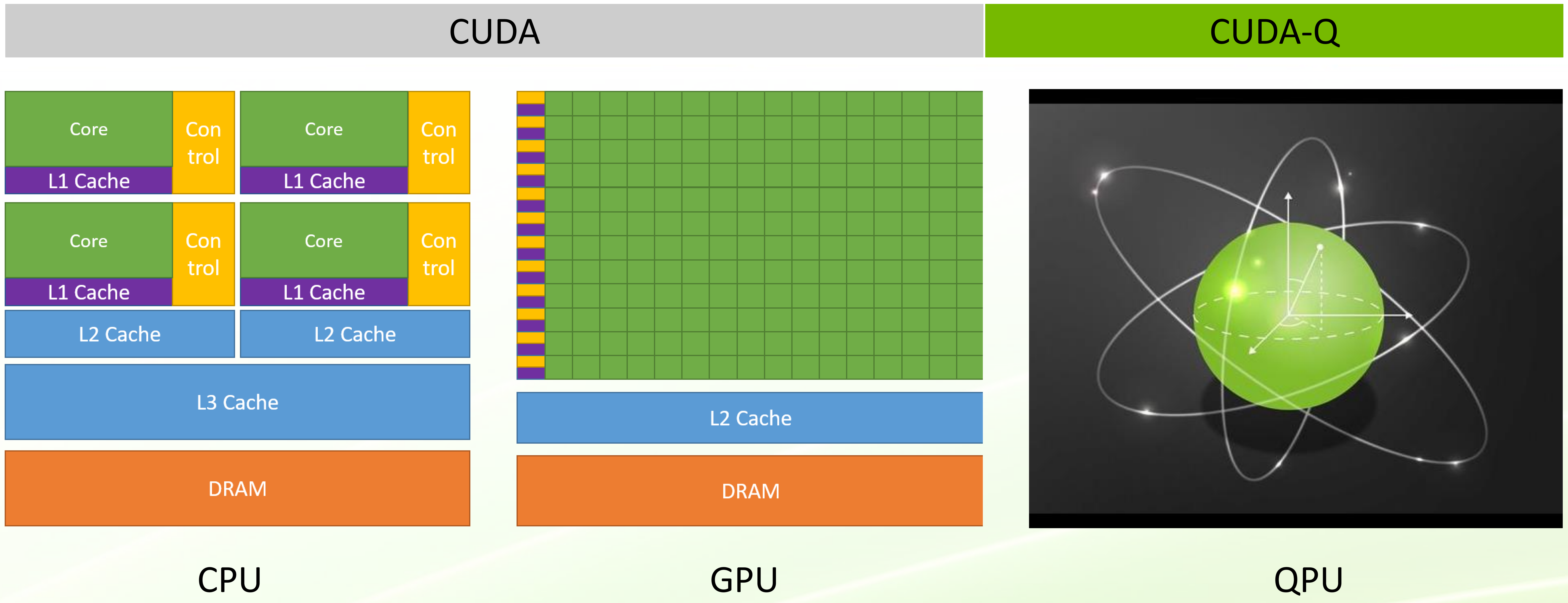
Kernel = instruction for each thread to follow

Kernel for matrix multiplication:
compute the dot product of
an assigned row in A with an assigned column of B

Accelerated Supercomputing



Quantum Accelerated Supercomputing

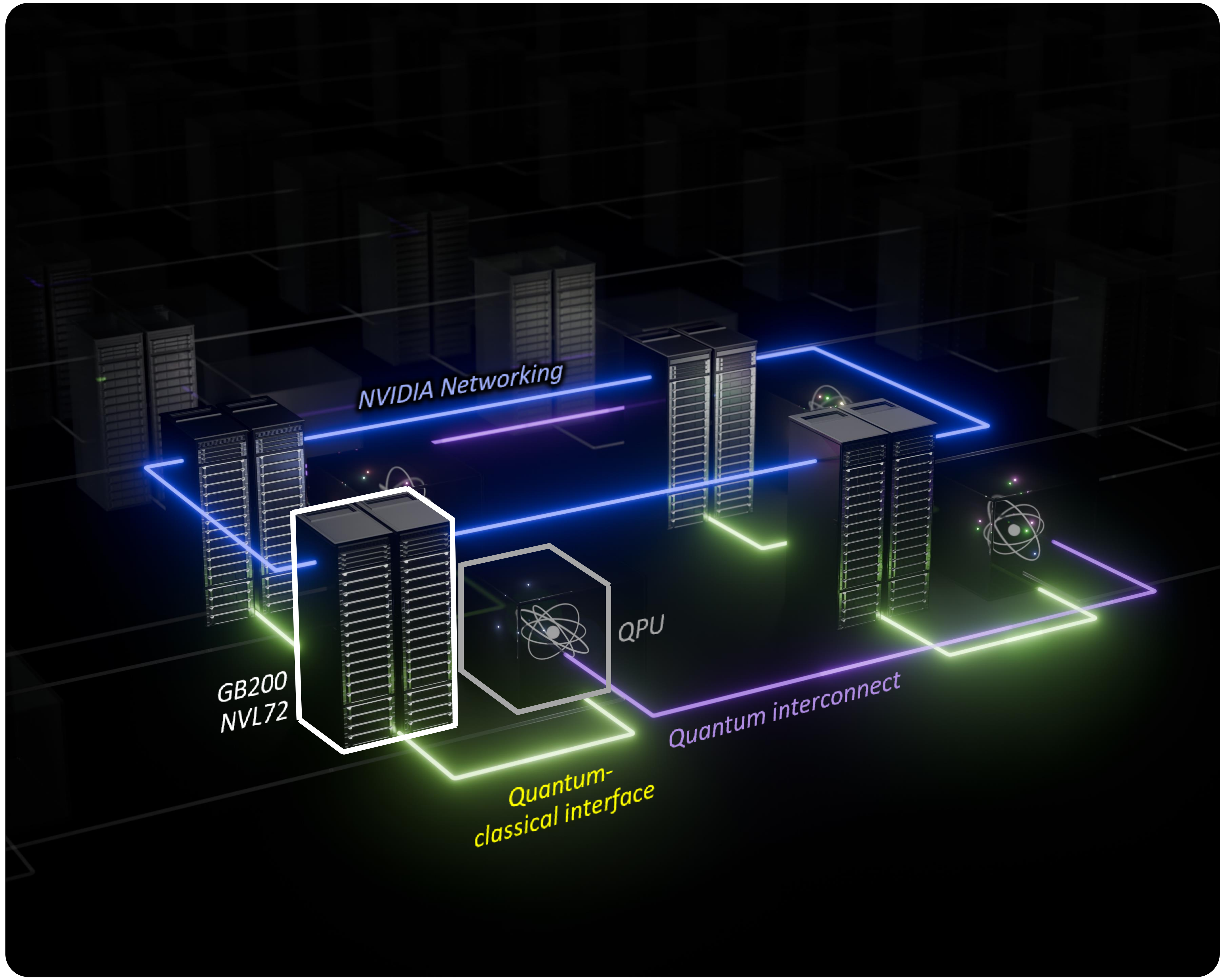


Tomorrow's Accelerated Quantum Supercomputers are GPU Supercomputers

Accelerated Quantum Supercomputer

*A **hybrid quantum-classical** device that uses **GPU-supercomputing** to turn qubit technology into a computer able to run useful applications*

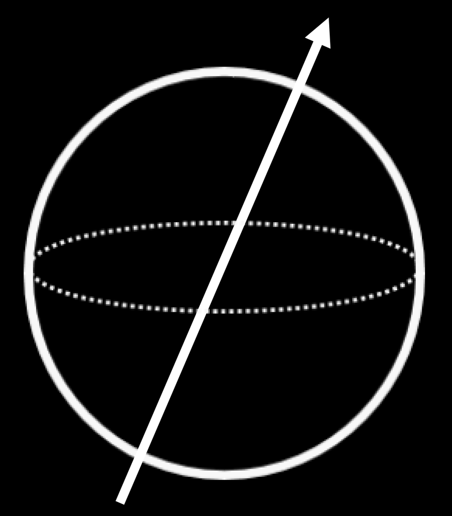
- Useful quantum computers are mostly an **AI supercomputer**
- NV supercomputers are QPU-agnostic
- **Hybrid applications** use CPUS, GPUs and QPUS
- AI supercomputing needed to **control and operate** QPU hardware



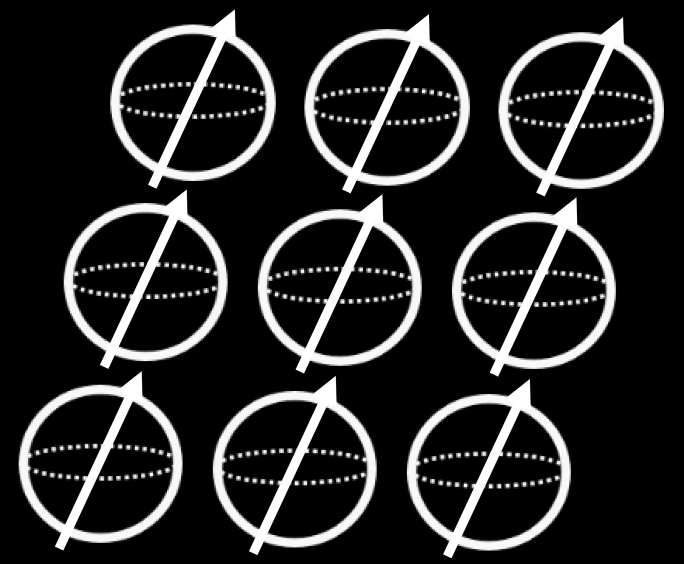
Accelerated Computing

Quantum Challenges

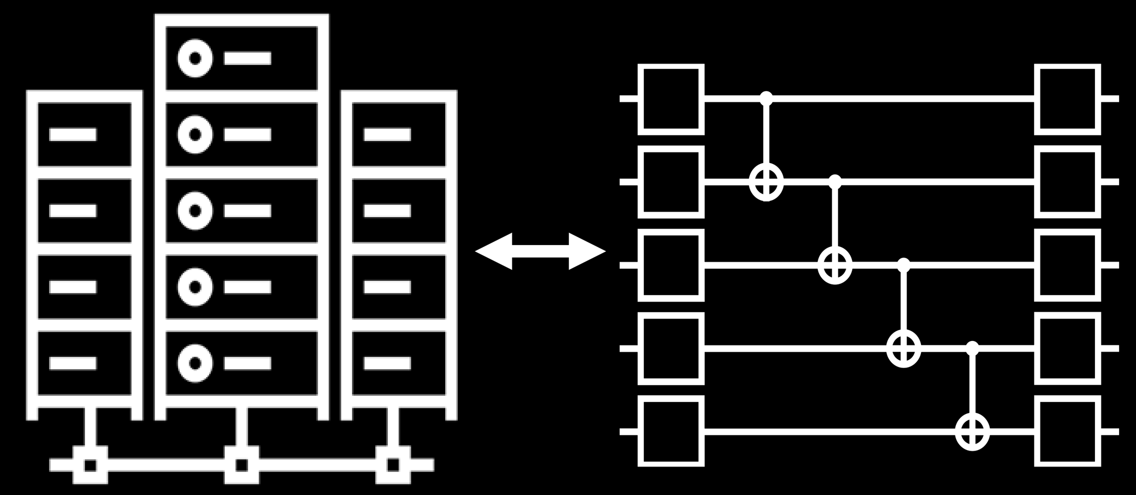
What's Standing Between Today and Useful Quantum Computing?



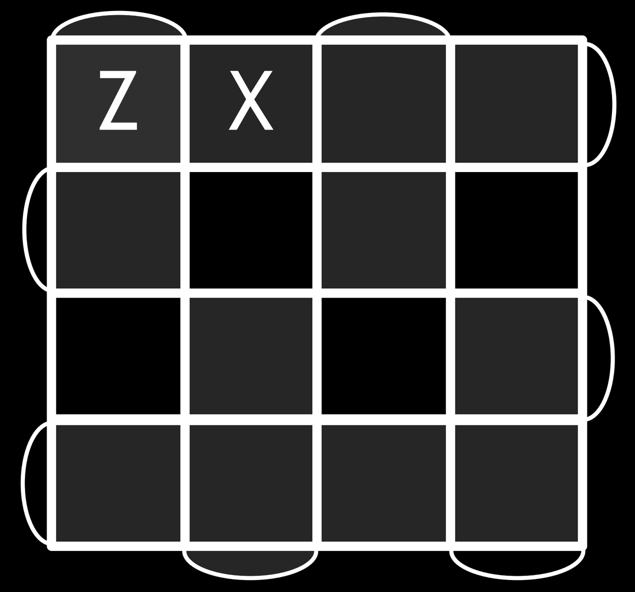
Qubit Fidelity
99.99% 2-Qubit Gate Fidelity



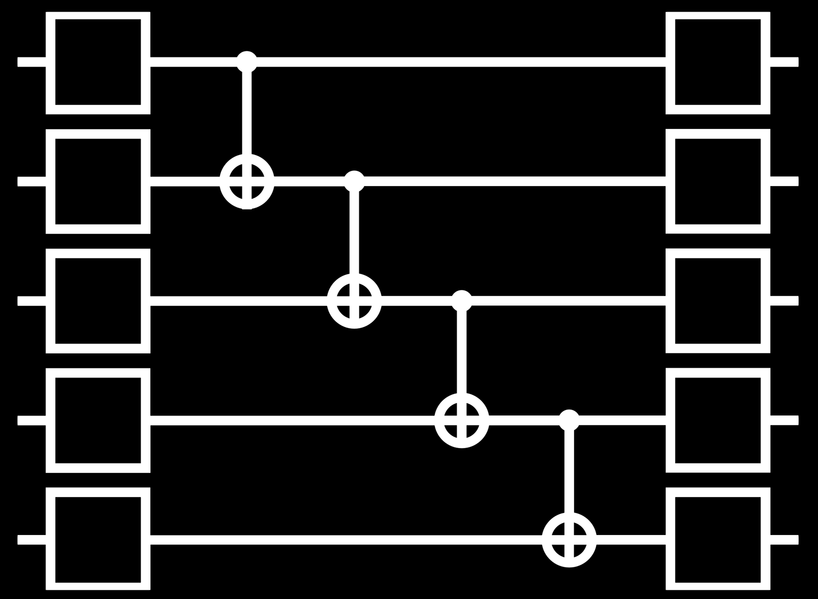
Qubit Scale
100k-1M+ Qubits for FTQC



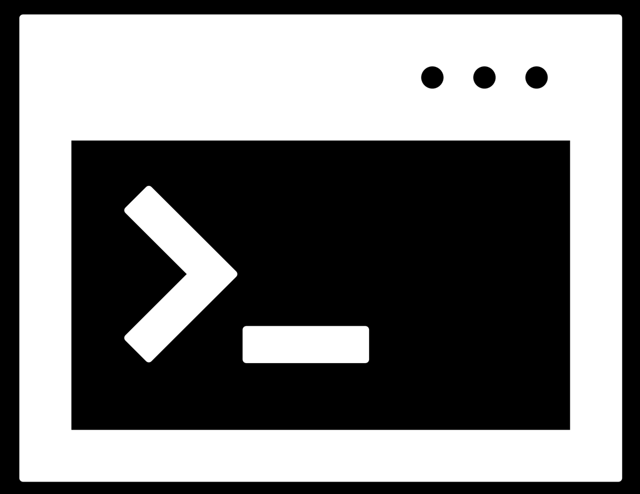
HPC Integration
Sub-Microsecond HPC-QC Latency



Error Correction
Methods that Scale to Large Quantum Systems



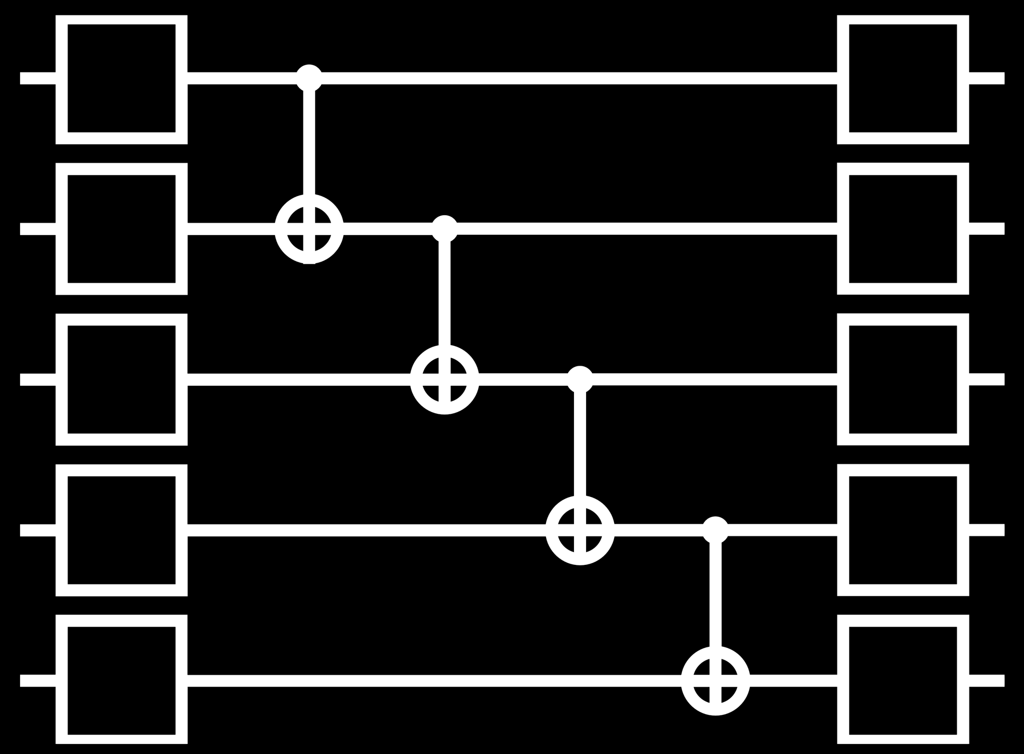
Algorithms
Algorithms with Exponential Speed-up



Developer Tools
Integrate with Scientific Computing
Familiar to non-Quantum Physicists

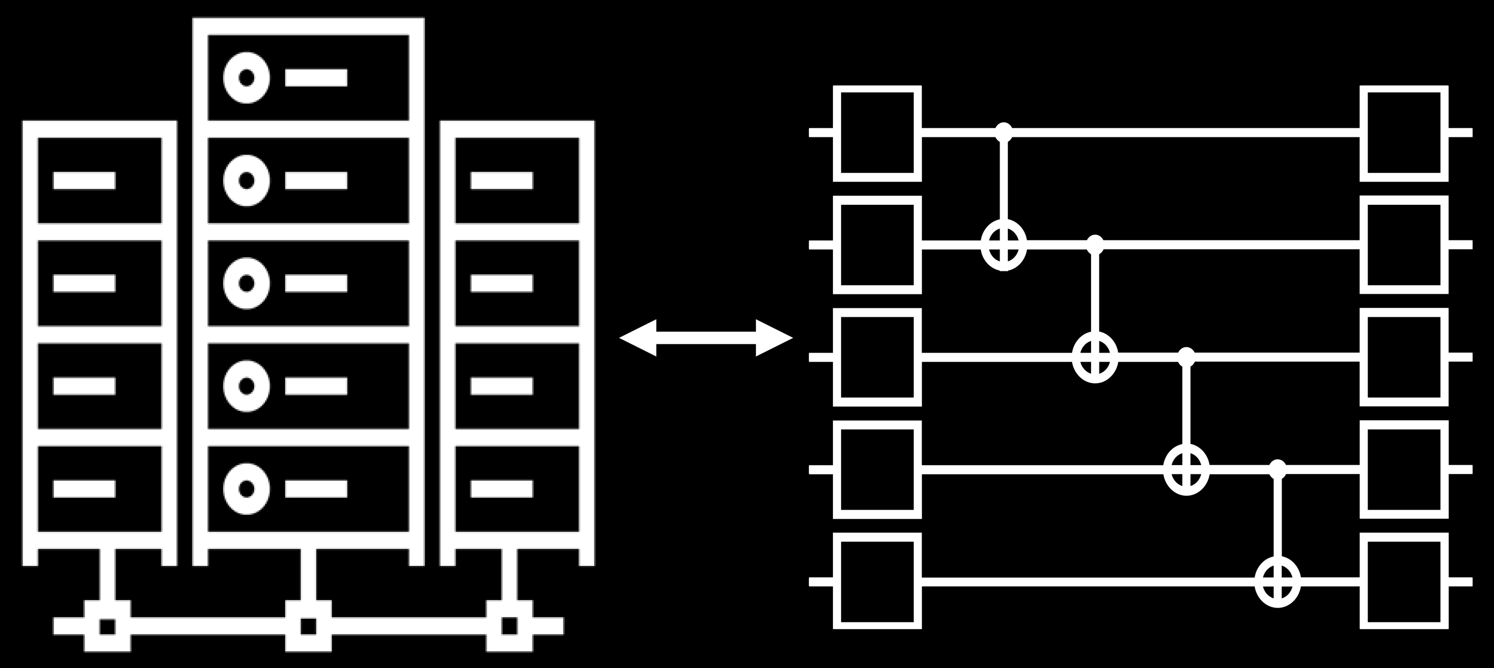
NVIDIA Quantum

Powering the Global Quantum Computing Community



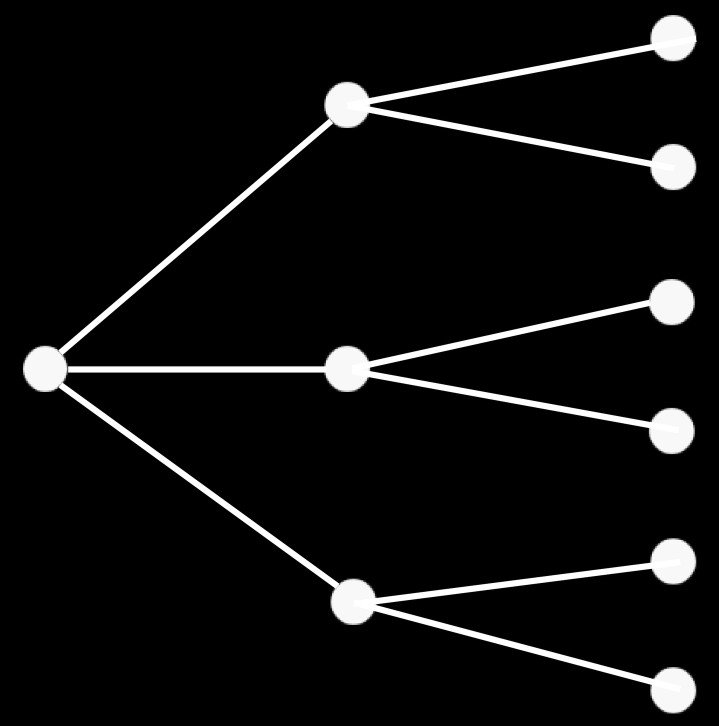
Simulation

Algorithm Design, Resource Estimation, QPU Design



HPC Quantum Integration

Integrated Applications, QEC, Sub-Microsecond Latency



AI for Quantum

QEC, Calibration, Algorithms

CUDA-Q

Libraries | Programming Model | Tools | Infrastructure

cuQuantum

Quantum Simulation

DGX-Q

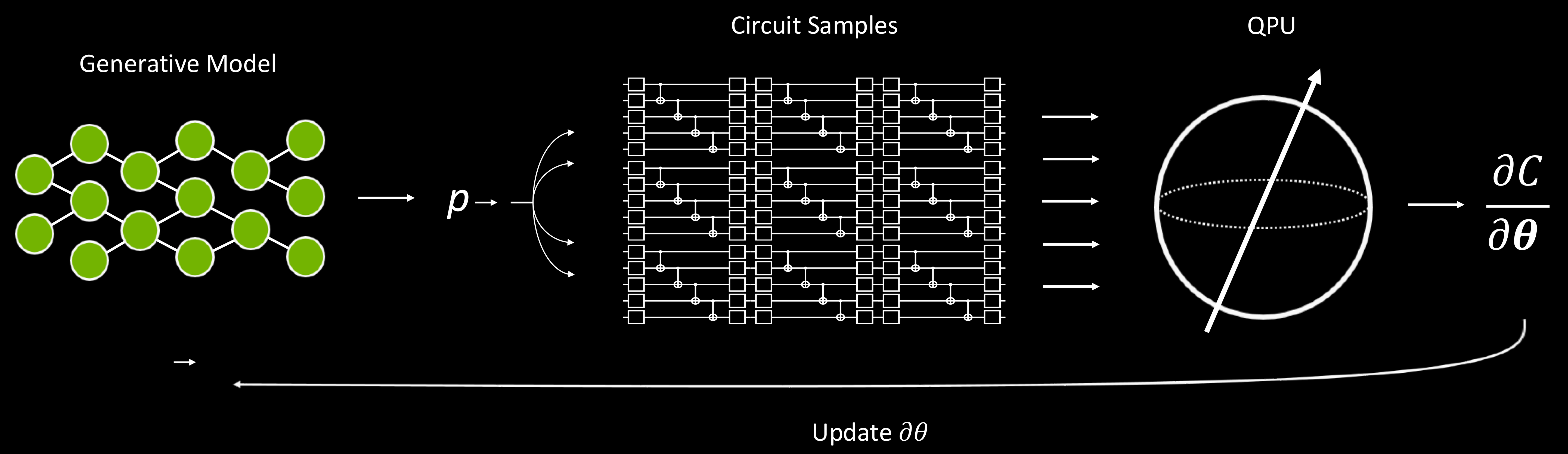
Quantum Integrated Computing

GPU Supercomputing

Generative AI + Quantum Algorithms

University of Toronto, St Jude's, and NVIDIA partner to invent GPT-QE

- Generative Pre-Trained Transformer-based (GPT) method for computing the ground state energies
- First GPT-generated quantum circuit
- Run via CUDA-Q on NERSC Perlmutter



UNIVERSITY OF TORONTO

NVIDIA

St. Jude Children's Research Hospital

Agenda

- What is Quantum Accelerated Supercomputing
- Useful Quantum Simulation
- How-to Guide to CUDA-Q
- Distributed Quantum Computing
- Conclusion

The Case for Quantum Computing Simulation

Quantum research is limited by access

Availability

~500k
Quantum Developers

~50
Publicly available QPUs

Uptime

10-20%
Typical uptime for deployed QPU

Scale

0
Fault-Tolerant Qubits Available

O(100)-O(1000)
Needed for useful applications

Accuracy

19
Circuit Depth, best case

3-4
Circuit Depth, typical

Iteration Time

40Q sim

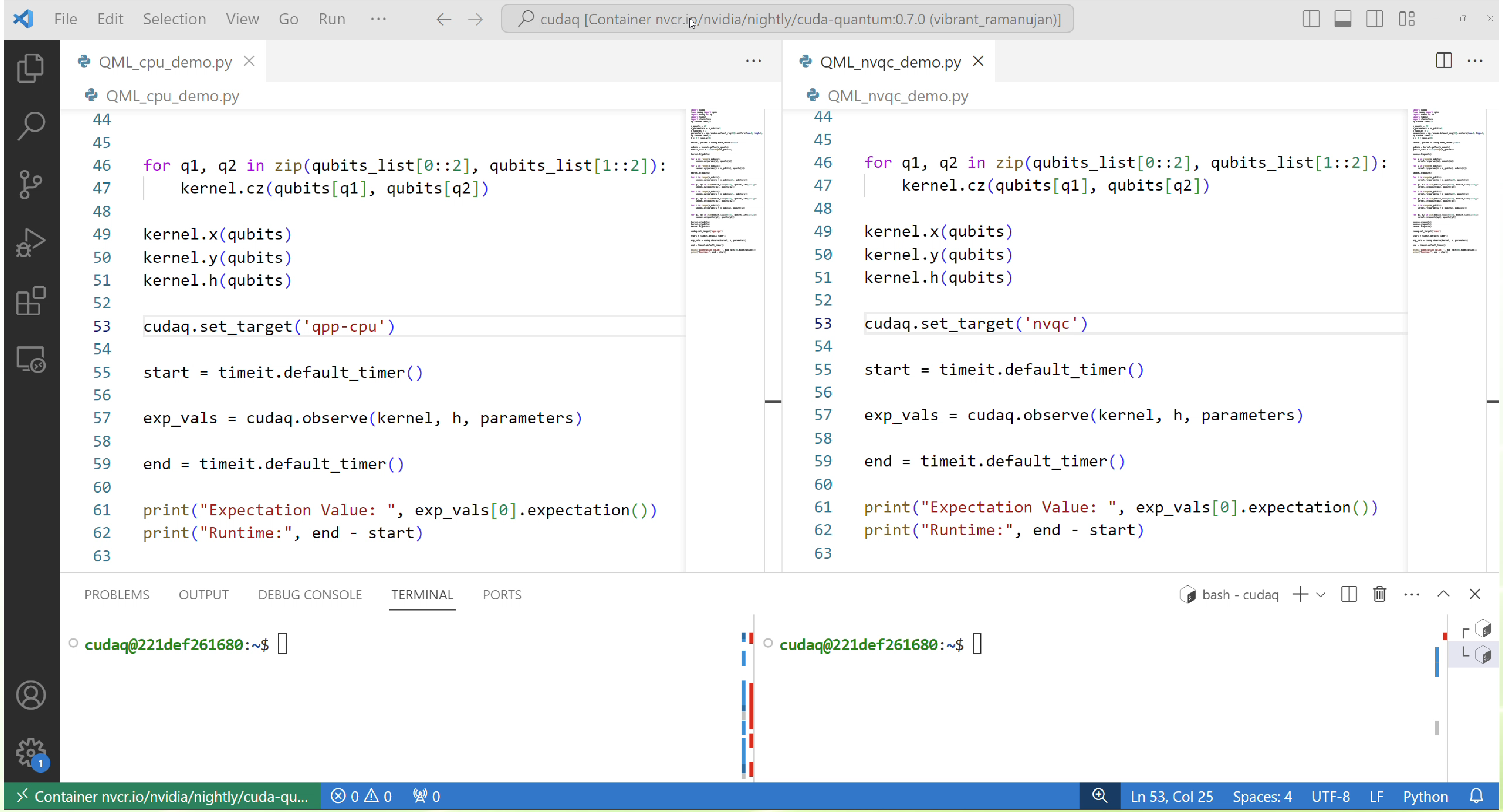
1 hour
GPU Cluster

7.5 years
CPU

Integration

4 million
CUDA Developers worldwide

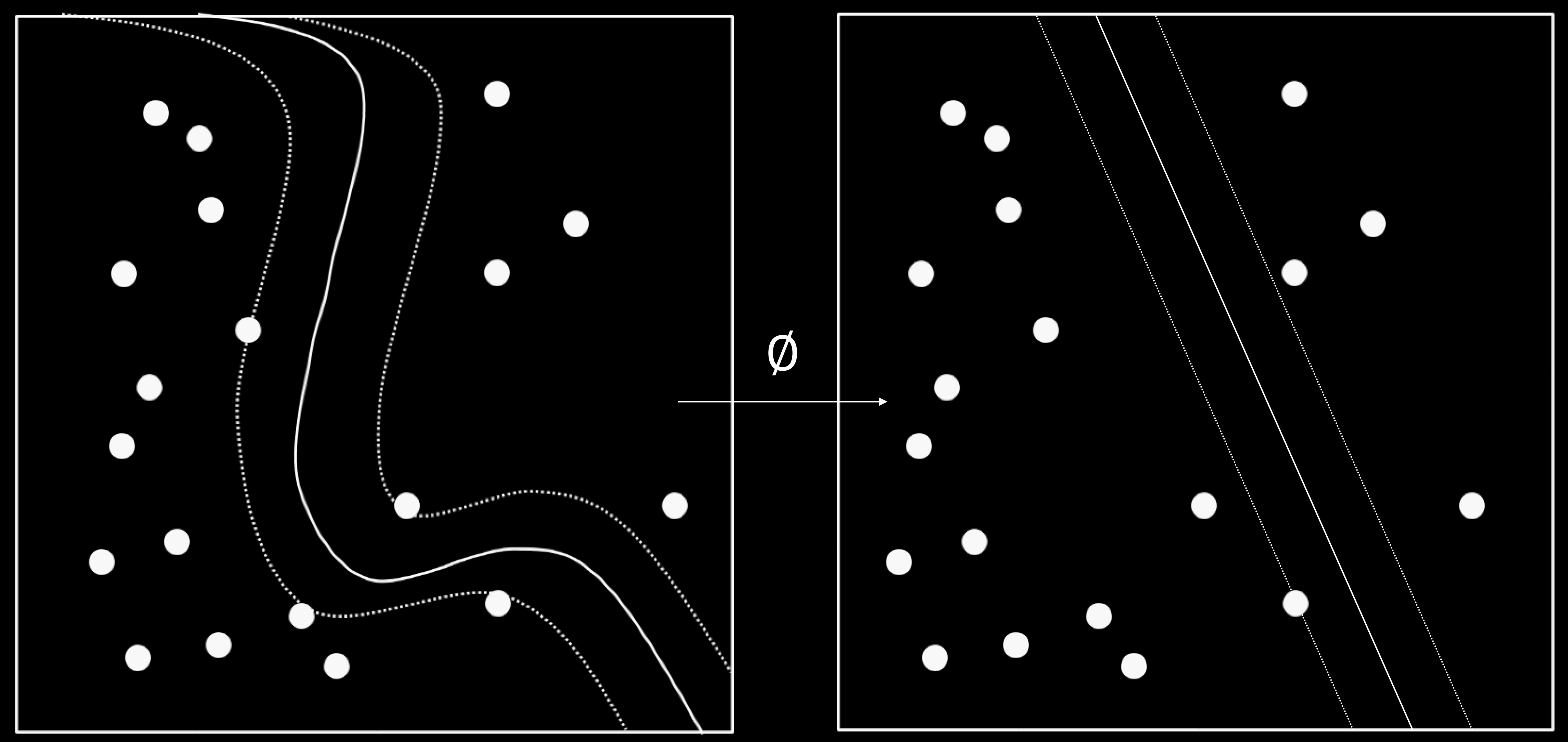
Quantum Simulation on a GPU



Fraud Detection

HSBC Leverages CUDA-Q to Develop Improved Fraud Detection

- Fraudulent transactions: loss of $1.9BN per year for UK alone
- Quantum-inspired methods may improve fraud detection
- Reduced false positives by 4%, improved true positives by 2%
- Run as 165 qubit classification problem with CUDA-Q



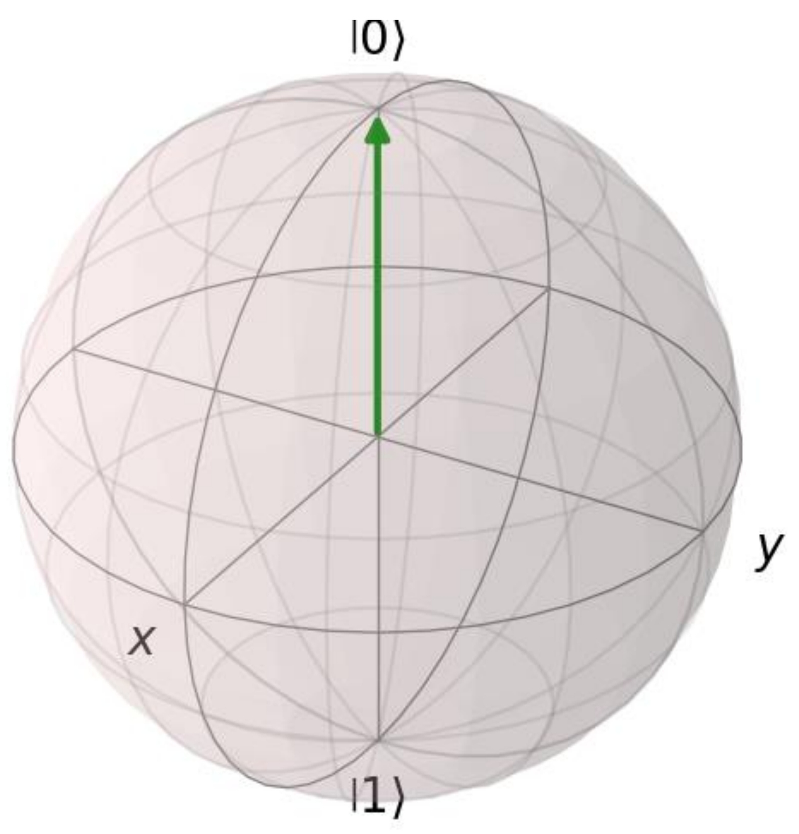
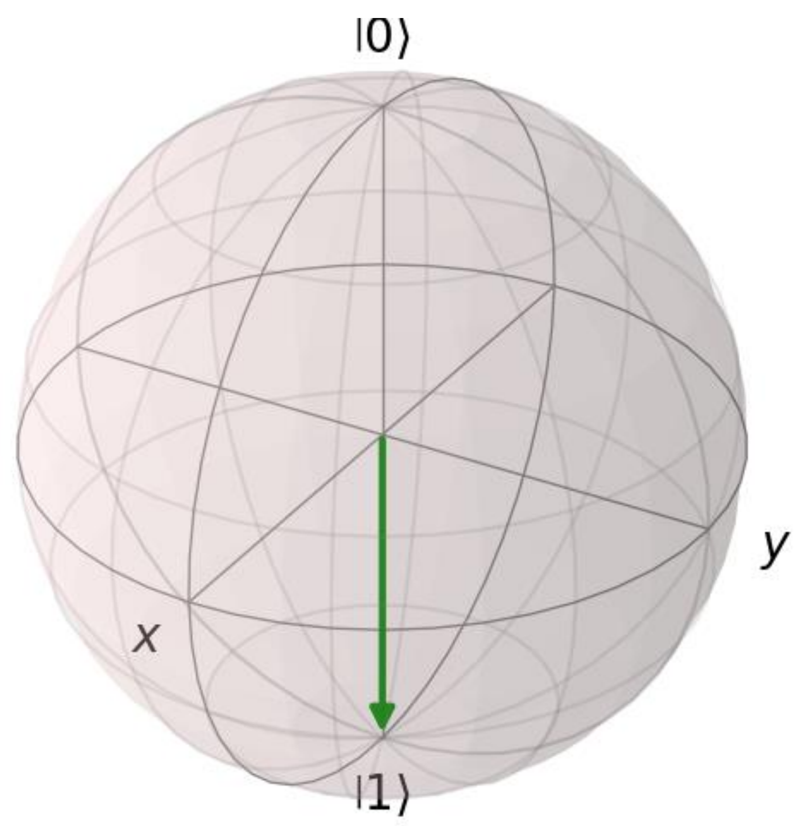
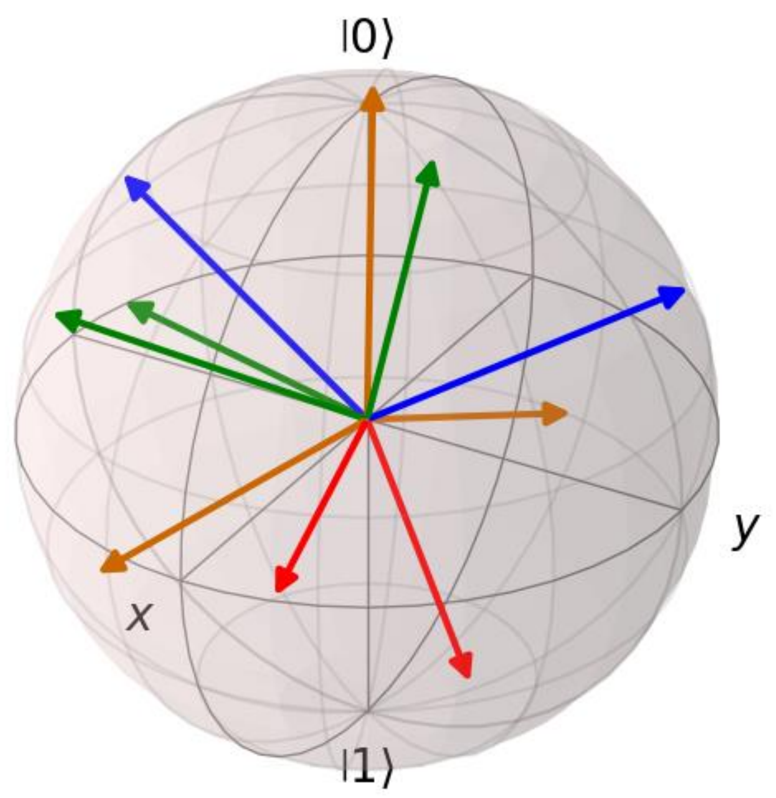
HSBC

Agenda

- What is Quantum Accelerated Supercomputing
- Useful Quantum Simulation
- How-to Guide to CUDA-Q
- Distributed Quantum Computing
- Conclusion

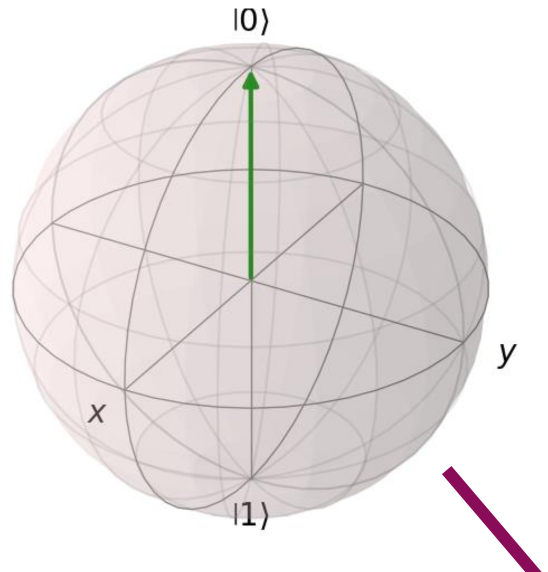
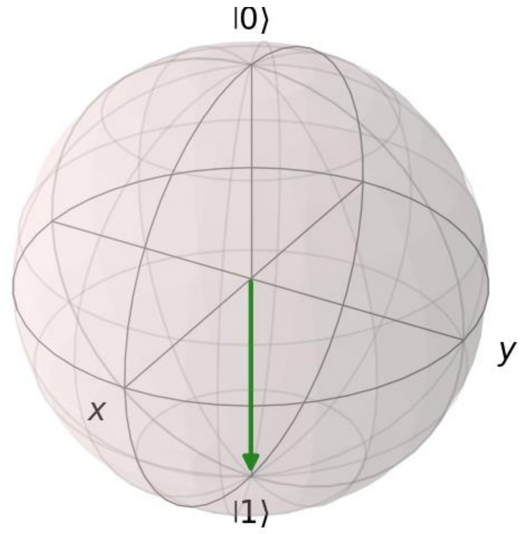
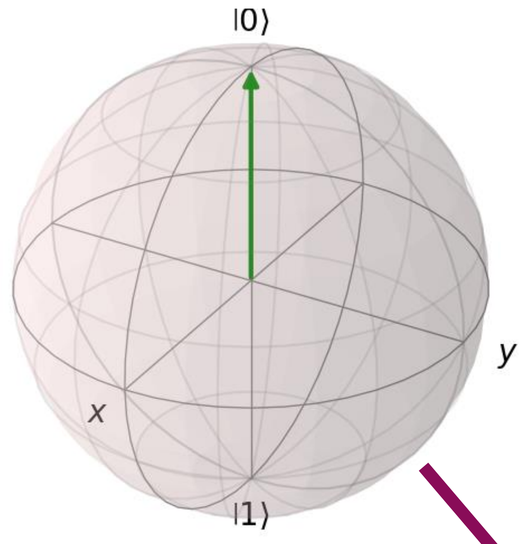
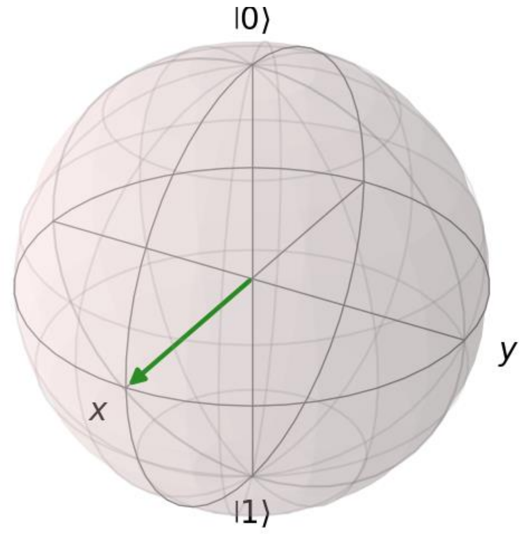
Quantum States

Single-qubit states

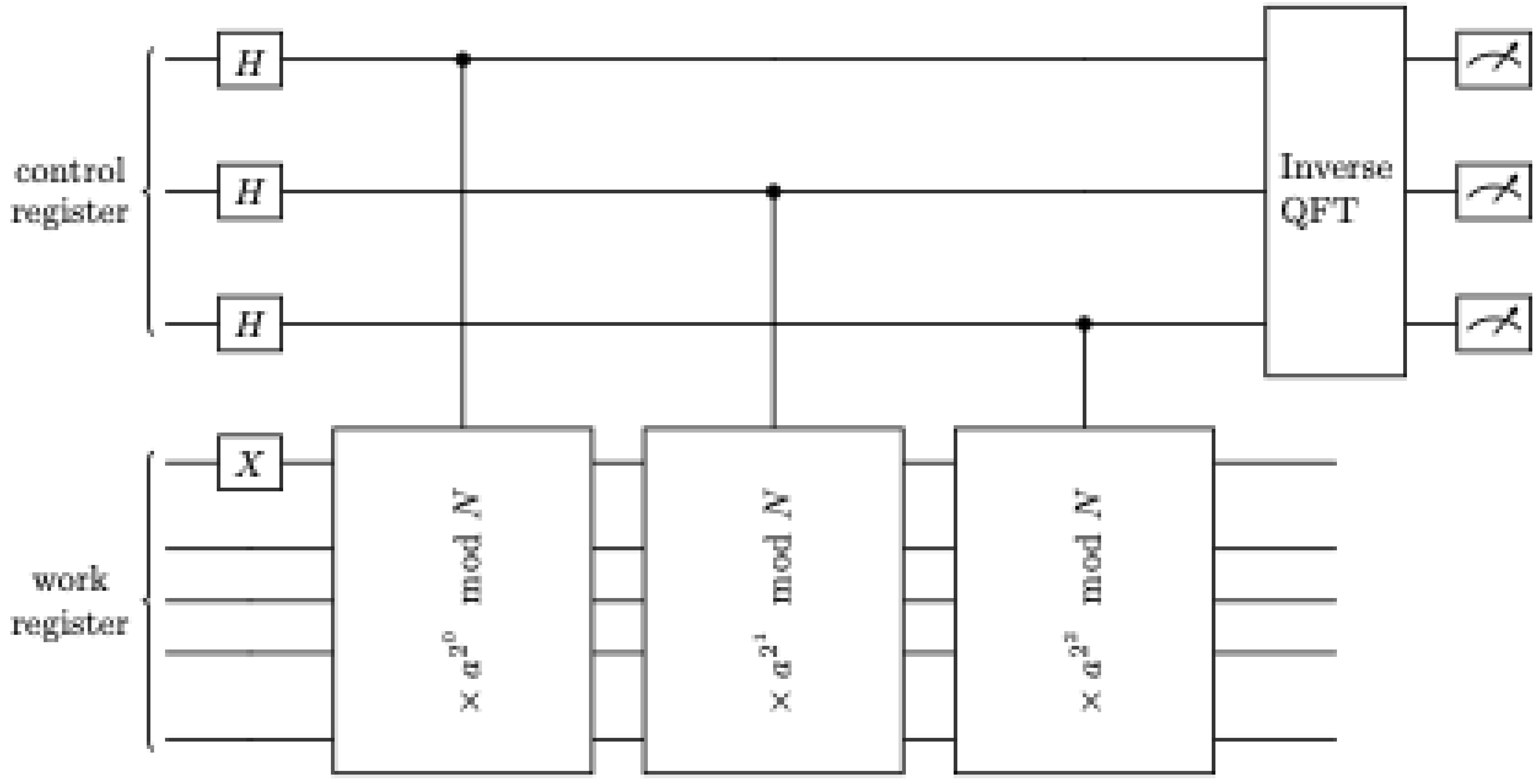
<table>
<tr><th></th><th>The zero state</th><th>The one state</th><th>Quantum states</th></tr>
<tr><td>Ket notation</td><td>$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$</td><td>$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$</td><td>$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,$$ where α and β are complex numbers satisfying the equation $|\alpha^2| + |\beta^2| = 1$. The coefficients α and β are referred to as *probability amplitudes*, or *amplitudes* for short.</td></tr>
<tr><td>Bloch Sphere</td><td>
</td><td>
</td><td>
</td></tr>
</table>

Quantum Gates or Operations

Some examples

	The Bit Flip Gate	The Hadamard Gate	CNOT Gate
Ket notation	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ $X\vert 0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \vert 1\rangle$	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ $H\vert 0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \vert +\rangle$	$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$
Bloch Sphere Representation	 Rotate 180 degrees about the X axis 	 Rotate 180 degrees about the X+Z axis 	$CNOT(\frac{1}{\sqrt{2}}\vert 00\rangle + \frac{1}{\sqrt{2}}\vert 10\rangle)$ $= \frac{1}{\sqrt{2}}CNOT\vert 00\rangle + \frac{1}{\sqrt{2}}CNOT\vert 10\rangle$ $= \frac{1}{\sqrt{2}}\vert 00\rangle + \frac{1}{\sqrt{2}}\vert 11\rangle$

Quantum Kernels or Circuits

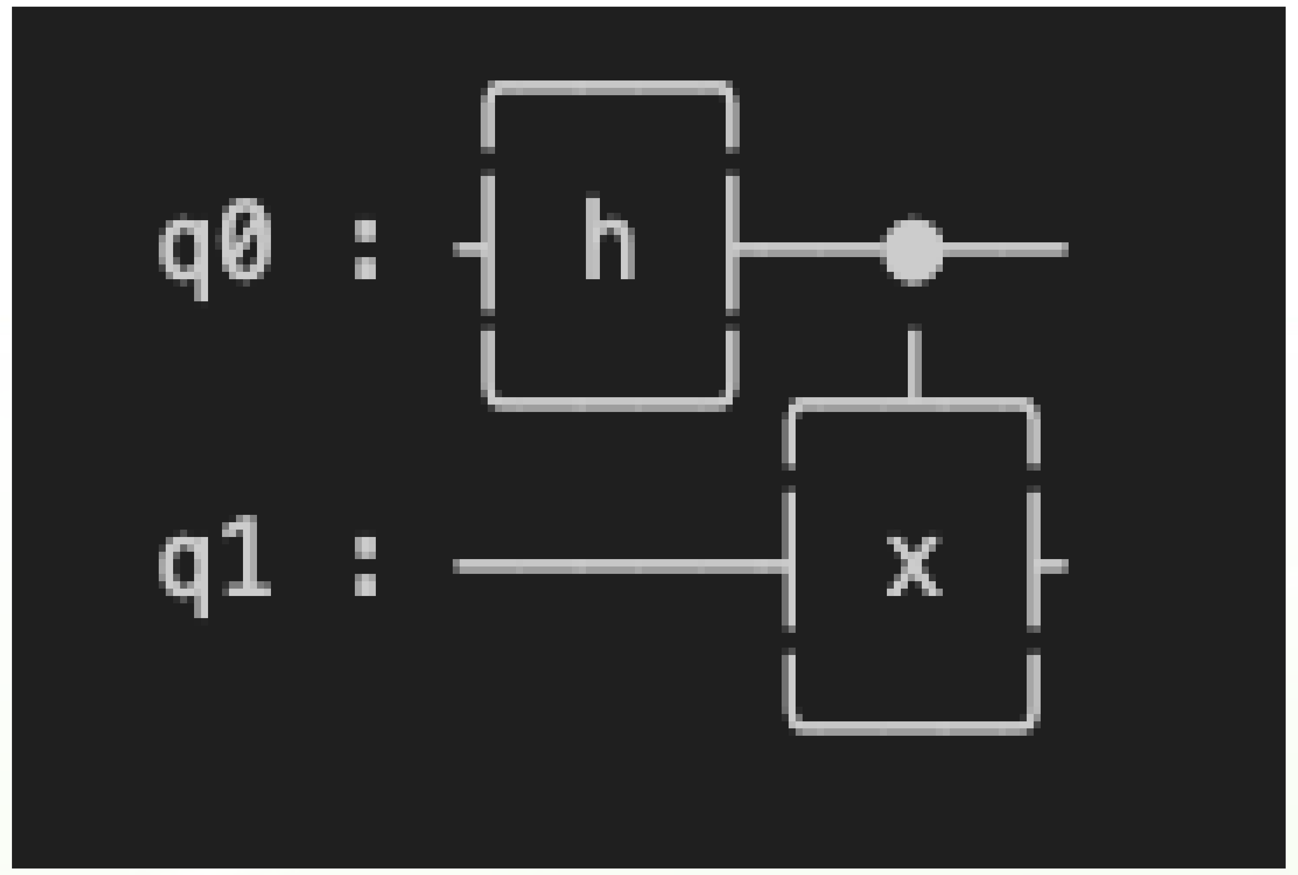


Template for a Quantum Program

- Initialize/allocate the qubits
- Manipulate the quantum with gates
- Extract information from the quantum state by taking measurement(s)

Building your First CUDA-Q Kernel

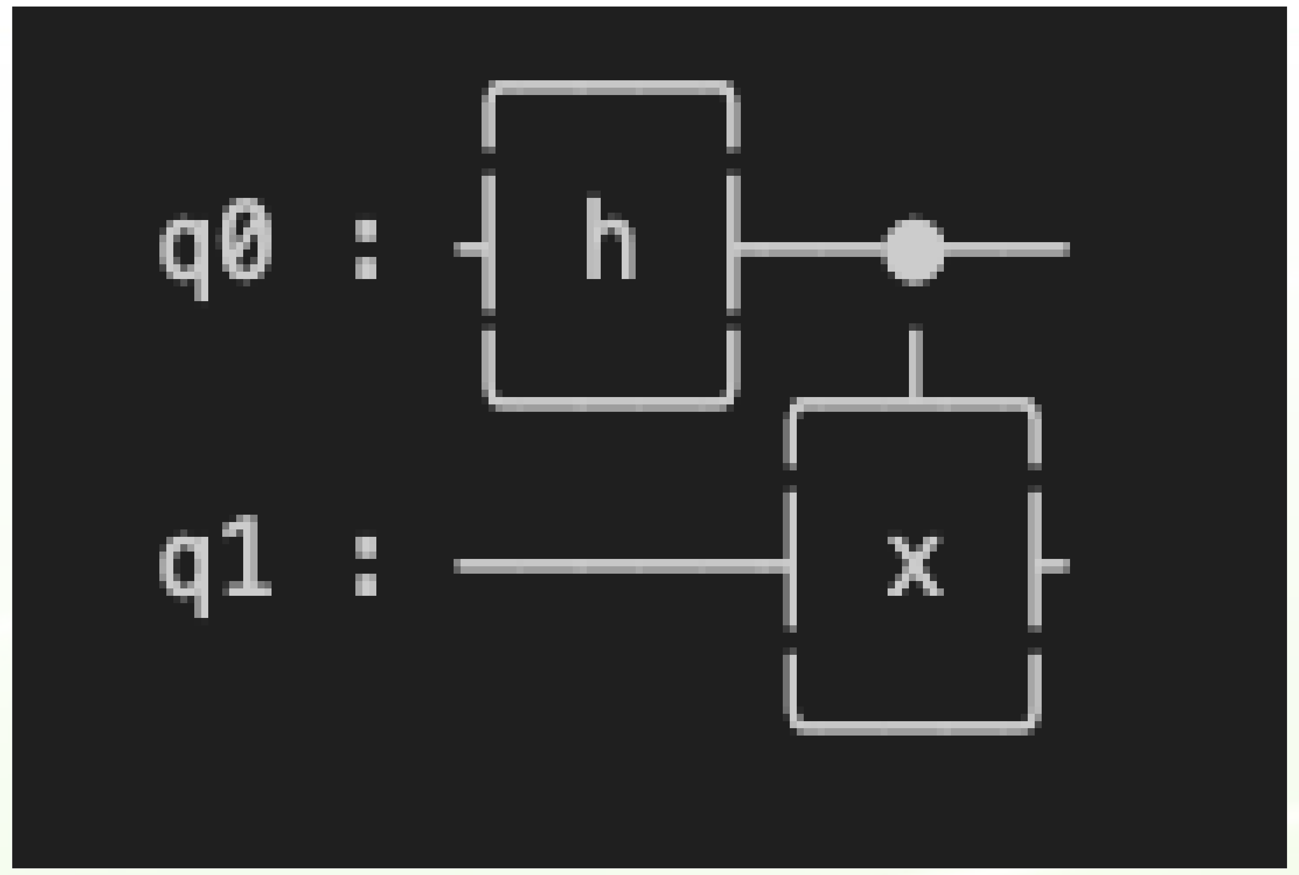
```
import cudaq

qubit_count = 2

# Define the kernel
@cudaq.kernel
def my_kernel(qubit_count: int):
  # Allocate our `qubit_count` to the kernel.
  qubits = cudaq.qvector(qubit_count)

  # Apply a Hadamard gate to the qubit indexed by 0.
  h(qubits[0])

  # Apply a Controlled-X gate between qubit 0 (acting
  # as the control) and each of the remaining qubits.
  for i in range(qubit_count - 1):
    x.ctrl(qubits[i], qubits[i + 1])

  # Measure the qubits
  # If we don't specify measurements, all qubits are measured in the Z-basis by default.
  mz(qubits)

print(cudaq.draw(my_kernel, qubit_count))
```



- Initialize/allocate the qubits
- Manipulate the quantum with gates
- Extract information from the quantum state by taking measurement(s)

Sampling your First CUDA-Q Kernel

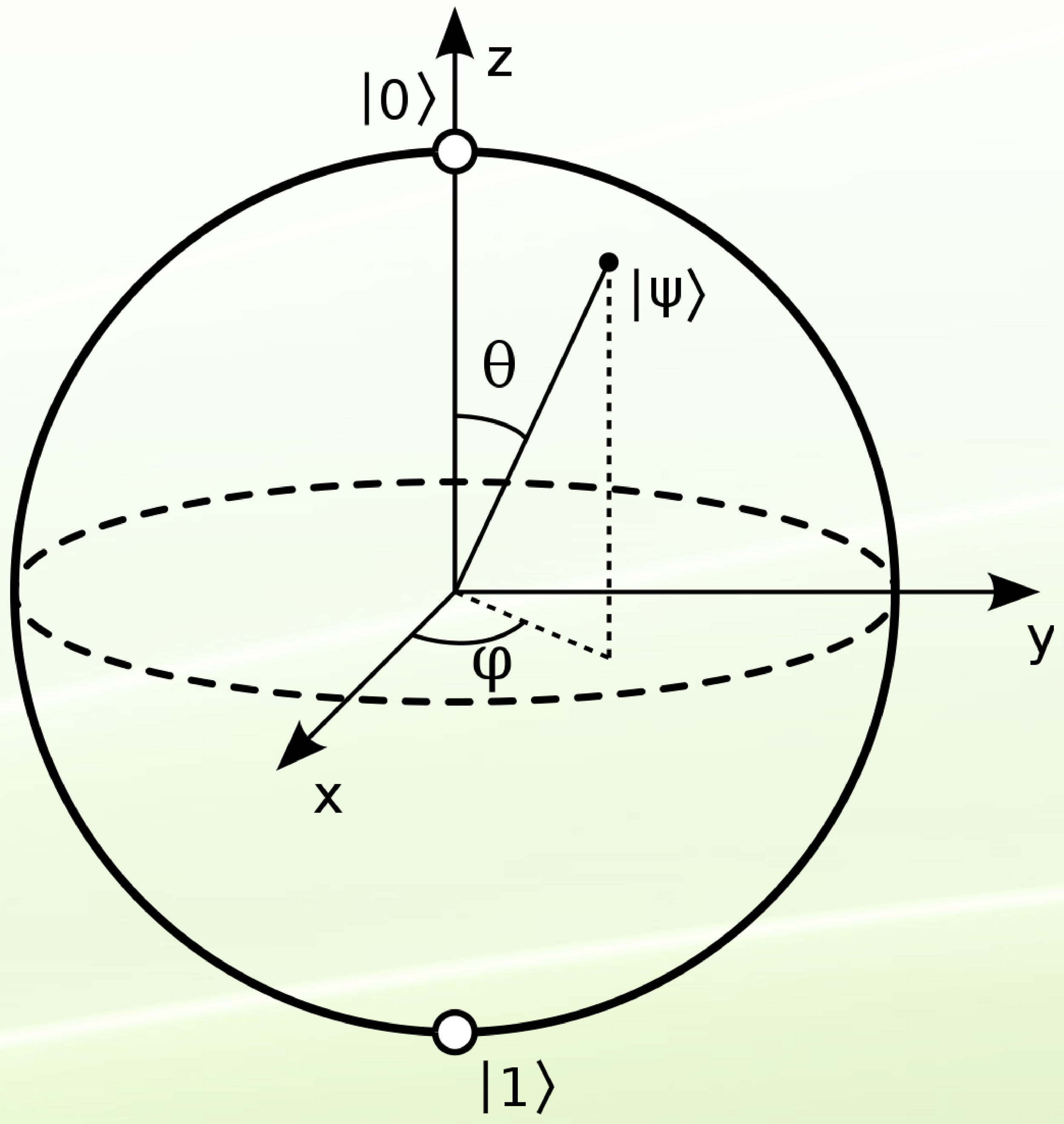
```
# First set the backend for kernel execution
cudaq.set_target('qpp-cpu') # selects a CPU backend

if cudaq.num_available_gpus() > 0:
    cudaq.set_target(`nvidia') # selects a GPU backend

# cudaq.set_target('nvqc') # selects the NVIDIA Quantum Cloud
# cudaq.set_target('ionq')  # select an available QPU backend

qubit_count = 2

results =  cudaq.sample(my_kernel, qubit_count, shots_count = 10000)

print(results) # Example: {00:5005, 11: 4995}

print(results.most_probable()) # prints: `00`

print(results.probability(results.most_probable())) # prints: `0.5005`
```



Computing Expectation Values with CUDA-Q

$$\langle + | Z | + \rangle$$

```python
import cudaq
from cudaq import spin

# First set the backend for kernel execution
cudaq.set_target('qpp-cpu') # selects a CPU backend

if cudaq.num_available_gpus() > 0:
    cudaq.set_target('nvidia') # selects a GPU backend

# Define your Hamiltonian Operator
operator = spin.z(0)
print(operator) # prints: [1+0j] Z

# Define your kernel to generate the plus state
@cudaq.kernel
def plus_state():
    qubit = cudaq.qubit()
    h(qubit)

result =  cudaq.observe(plus_state, operator, shots_count = 10000)

print(result.expectation()) # prints the approximate expectation value computed from 10000 shots
```

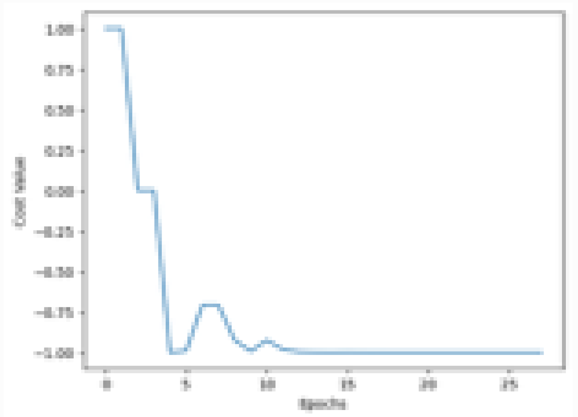


More Sample Code on our Website

https://nvidia.github.io/cuda-quantum

CUDA-Q Tutorials

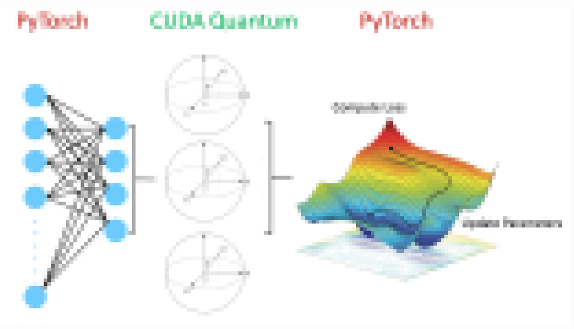
Tutorials that give an in depth view of CUDA-Q and its applications in Python.



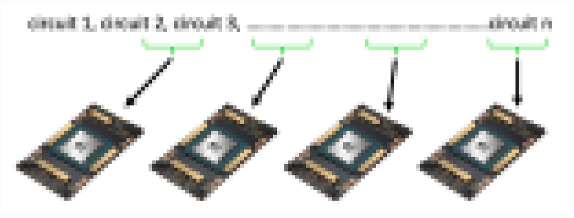
Cost Minimization

Deutschs' Algorithm

Executing Quantum Circuits



Hybrid Quantum Neural Networks



Multi-GPU Workflows

Multiple Qubits

Noisy Simulation

Quantum Approximate Optimization Algorithm for Max Cut

Quantum Bits

Quantum Fourier Transform

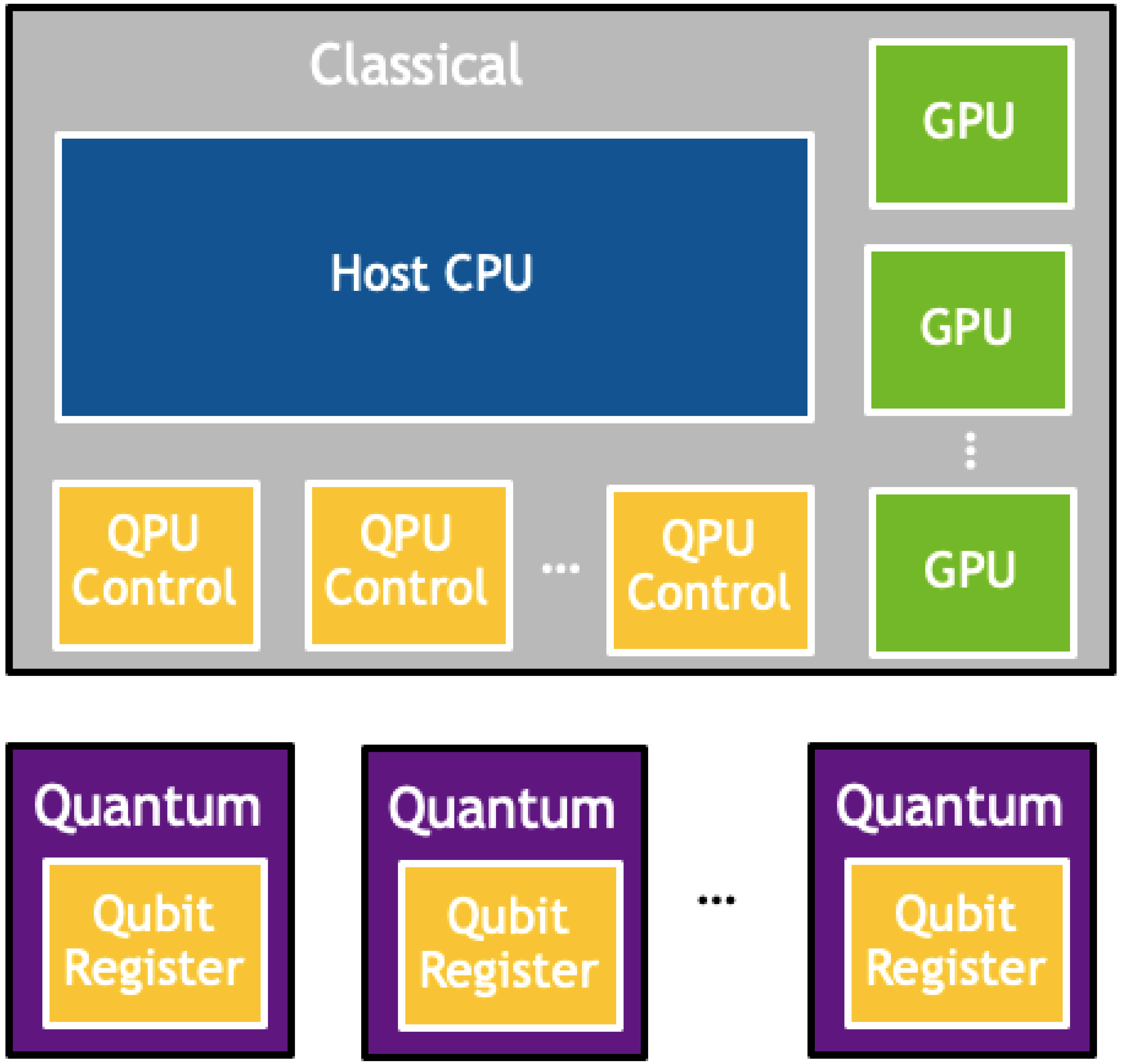
Variational Quantum Eigensolver

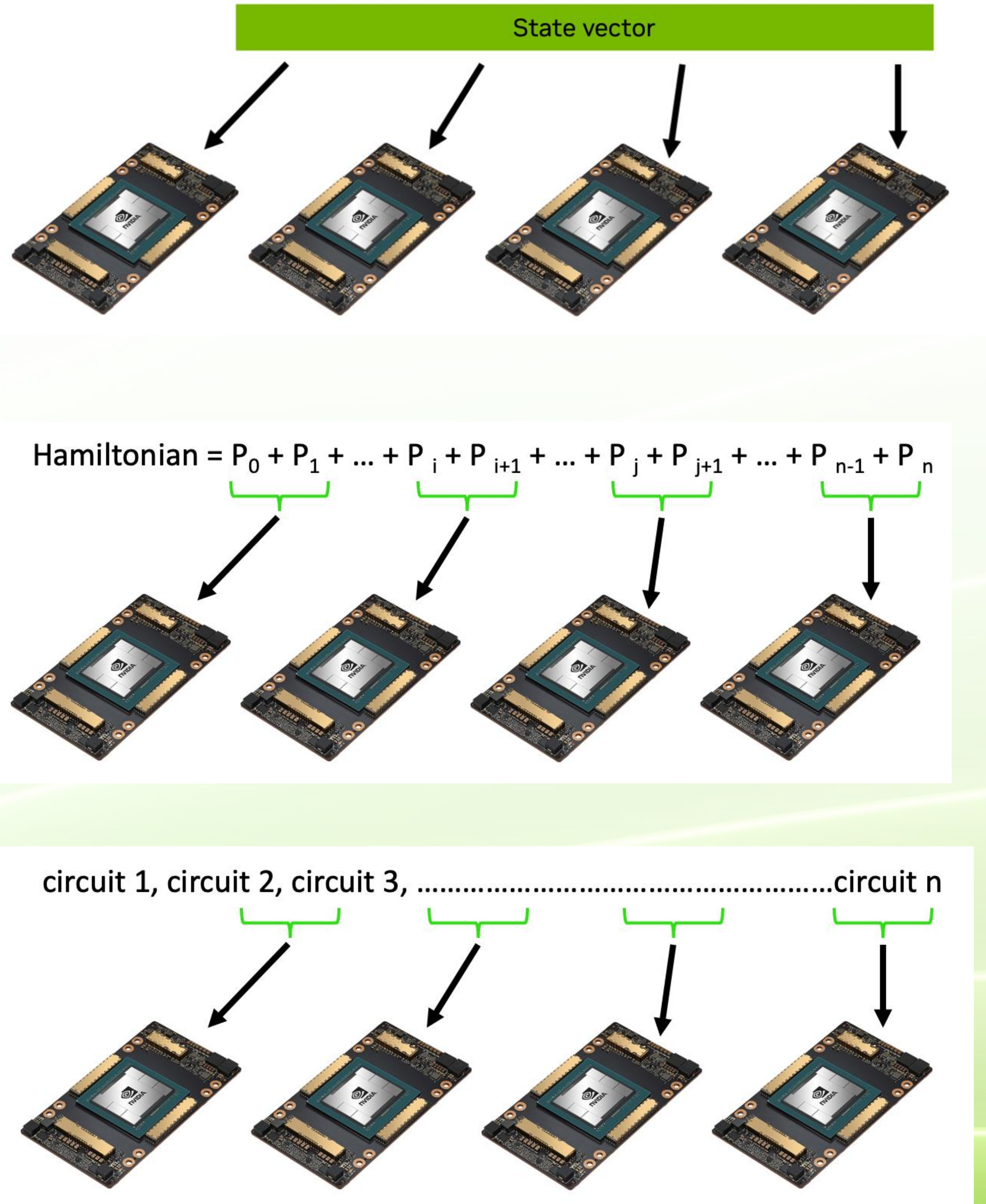
Agenda

- What is Quantum Accelerated Supercomputing
- Useful Quantum Simulation
- How-to Guide to CUDA-Q
- Distributed Quantum Computing
- Conclusion

GPU-Accelerated Quantum Computing

Some High-Level Strategies for Parallelization





Agenda

- What is Quantum Accelerated Supercomputing
- Useful Quantum Simulation
- How-to Guide to CUDA-Q
- Distributed Quantum Computing
- Conclusion

Quantum: Not Just for Physicists

Overcoming these challenges requires broad spectrum of expertise

Qubit Fidelity
99.99% 2-Qubit Gate Fidelity

Qubit Scale
100k-1M+ Qubits for FTQC

HPC Integration
Sub-Microsecond HPC-QC Latency

Error Correction
Methods that Scale to Large Quantum Systems

Algorithms
Algorithms with Exponential Speed-up

Developer Tools
Integrate with Scientific Computing
Familiar to non-Quantum Physicists

- Physicists
- Engineers
- Computer Scientists
- Developers
- Mathematicians
- Chemists
- Biologists
- Subject Matter Experts
- Students
- ...

NVIDIA