Moving Beyond QPU as an Accelerator

Embracing Non-Von Neumann and Physics-Based Approaches in Quantum Programming Models

Stefano Markidis

KTH Royal Institute of Technology, Stockholm, Sweden

Outline of this Talk

01

Quantum machine and programming models

- Discuss offloading models

02

More physics-accurate machine model

- Discuss pulse-level model

03

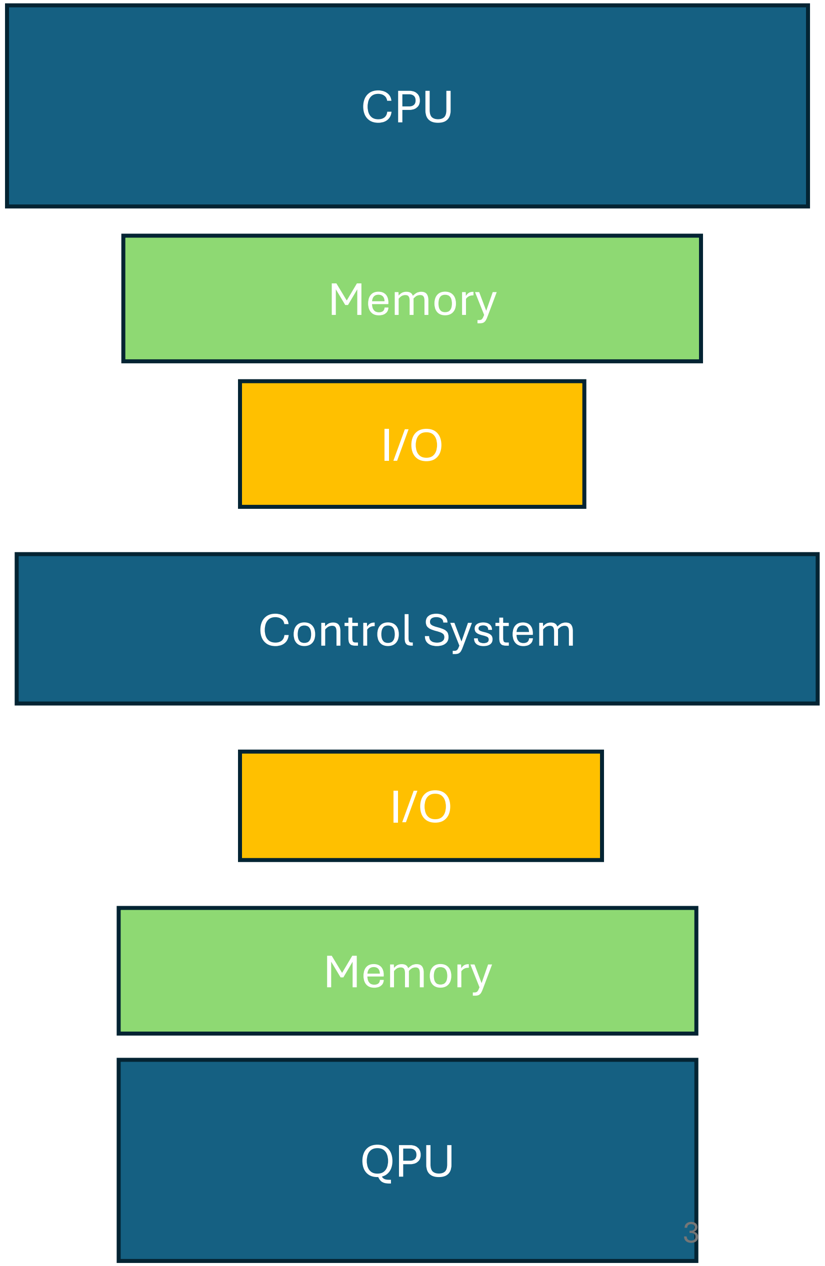
QC similarity with neuro-computing

- Opportunities for codesign and progress

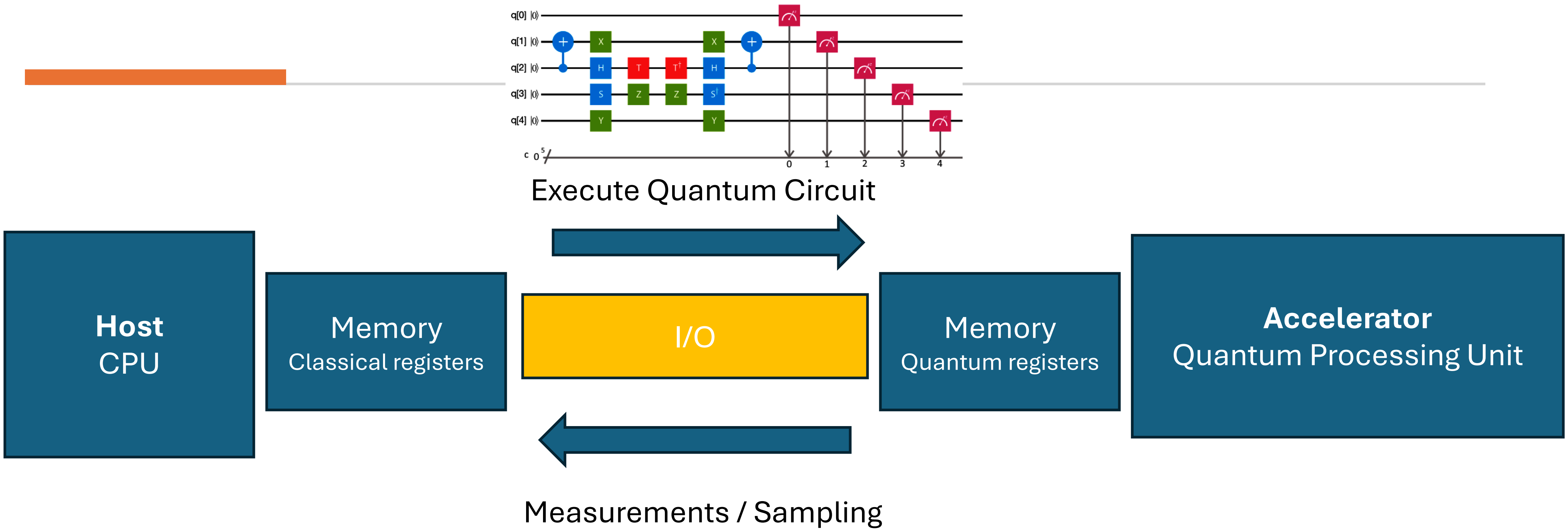
Understanding Machine Models

- Machine models describe the underlying hardware architecture and its operational principles.
  - This includes the organization and behavior of processors, memory hierarchies, I/O systems, and interconnects.
- Understanding the underlying machine model is crucial for
  1. Optimizing workflows
     - Compilation / Execution / Feedback Loop Highly Coupled Systems
  2. Understand system performance bottlenecks, such as memory and I/O Bottlenecks, and Optimize code written in a high-level programming model.
     - Knowing the architecture of quantum computers (machine models) in quantum programming models helps write efficient code.
  3. Allow for developing performance models.

Example of Quantum Computer Machine Model



A Quantum Computing Machine Model: Host Accelerator Machine Model



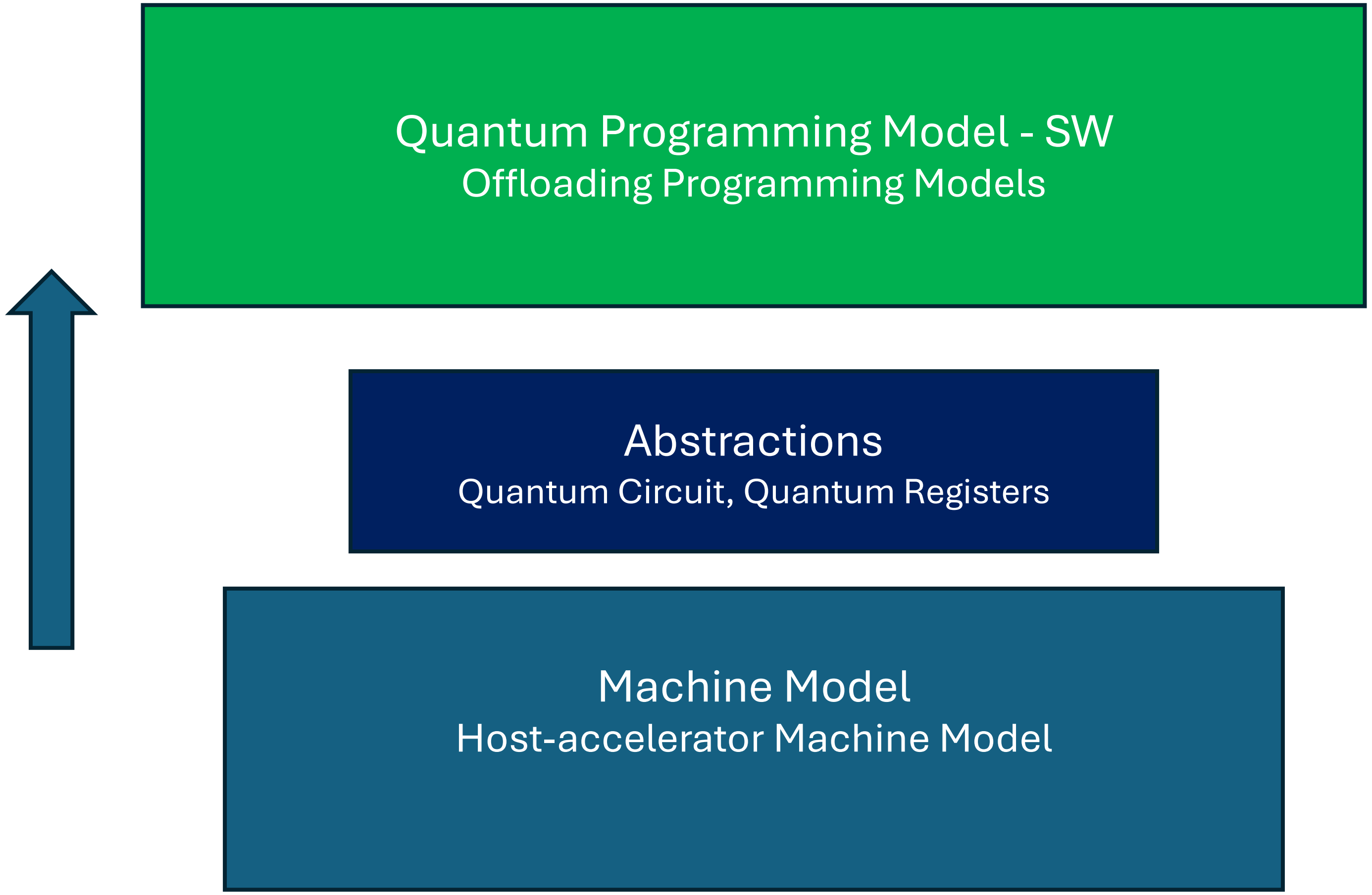
One of the most common machine models is the QPUs as an accelerator

- tackling specific tasks that benefit from quantum computation
- the classical CPU manages the overall process and performs non-quantum tasks.

Bridging Hardware and Software with Programming Models

Programming models abstract and simplify the interaction with machine models, allowing developers to write complex software efficiently.

- A programming model provides the foundational abstractions that developers use to build **software.**



An Example Offloading Programming Model - Qiskit

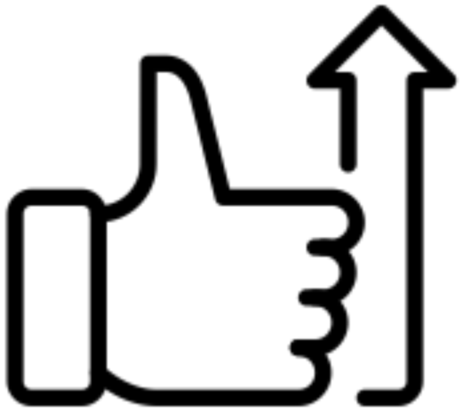
```
q = QuantumRegister(2,'q')
c = ClassicalRegister(2,'c')

def firstBellState():
    circuit = QuantumCircuit(q,c)

    circuit.h(q[0]) # Hadamard gate
    circuit.cx(q[0],q[1]) # CNOT gate
    circuit.measure(q,c) # Qubit Measurment

    print(circuit)

    job = execute(circuit, backend, shots=8192)

    job_monitor(job)
    counts = job.result().get_counts()

    print(counts)
```

Define variables on the host and accelerator

Launch the Kernel

Offloading Programming Model Abstractions

- **Quantum Circuit / Network**
  - Description of the quantum algorithms - similar to the definition of kernels in classical term
    - I/O included in the quantum circuit definition
  - It allows for general-purpose computation
  - It allows for complexity analysis of quantum algorithms
  - It allows us to reason in terms of quantum parallelism
- **Quantum Registers**
  - Useful to determine data movement and interaction between classical and quantum systems

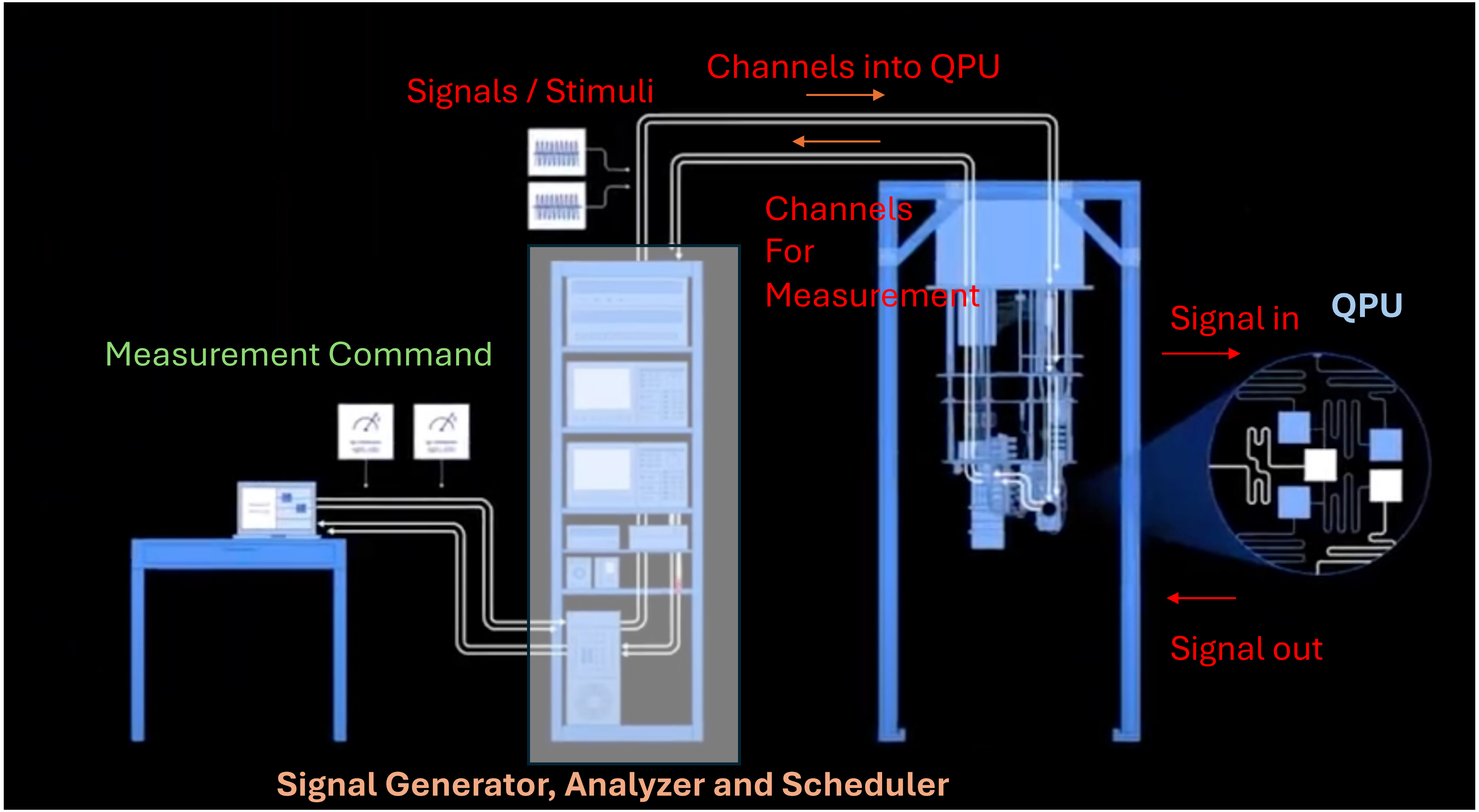


Markidis S. What is Quantum Parallelism, Anyhow?. InISC High Performance 2024 Research Paper Proceedings (39th International Conference) 2024 May 12 (pp. 1-12). Prometeus GmbH.

Some of Issues of the Offloading Programming Model Abstractions

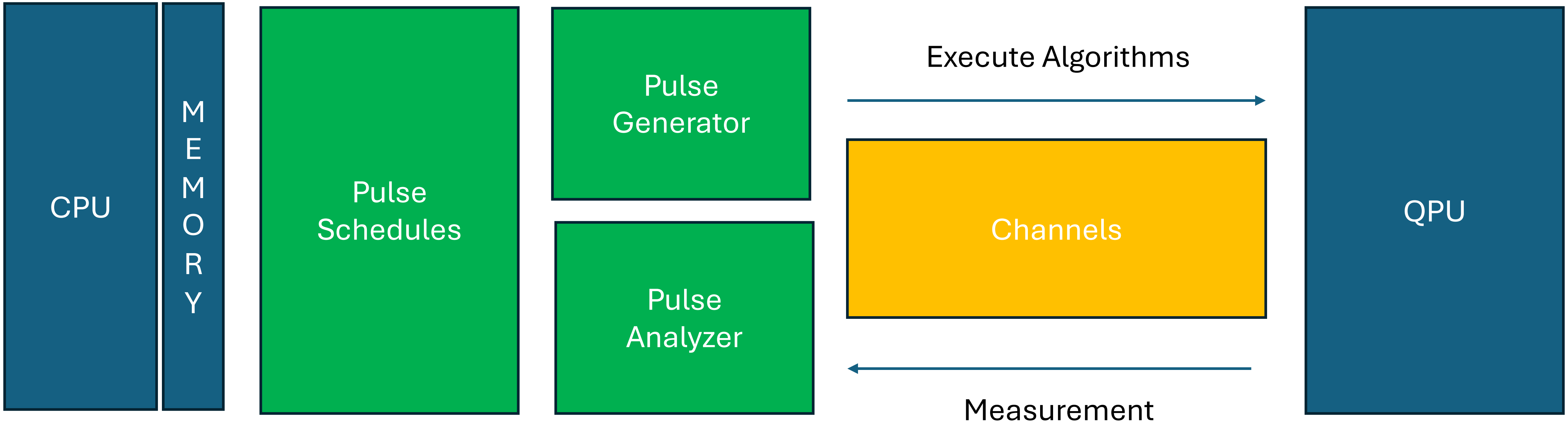
- **Quantum Circuit / Network**
  - It obfuscates the physical functioning of quantum systems - computation is a physical process, after all!
  - It limits the expressiveness of language in terms of a limited set of gates.
  - It is challenging to express highly-integrated
    - It does not have the concept of time
      - Needed for scheduling
- **Quantum Registers**
  - It implicitly introduces a von Neumann machine architecture and associated “artificial” problems such as the I/O and memory walls.

A More Detailed Hardware Perspective



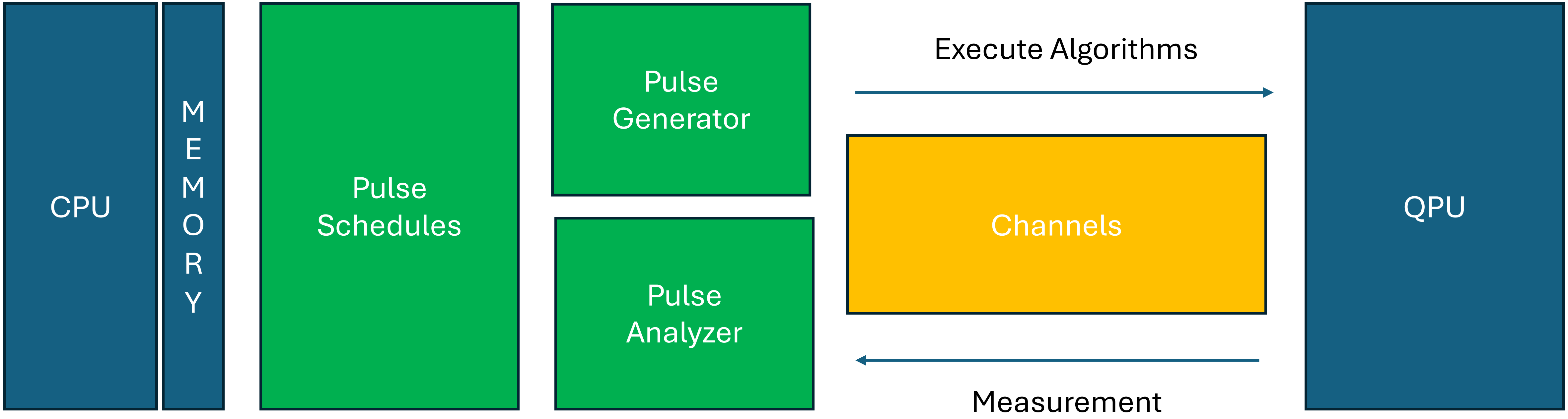
Adapted from Source: IBM / Superconducting Qubit – Similar diagrams could apply to other technologies

A More Refined Quantum Machine Model – Pulse-Level Machine Model



A pulse-level quantum machine model offers a detailed more integrated approach to quantum computation, focusing on the **physical control mechanisms** that implement quantum operations.

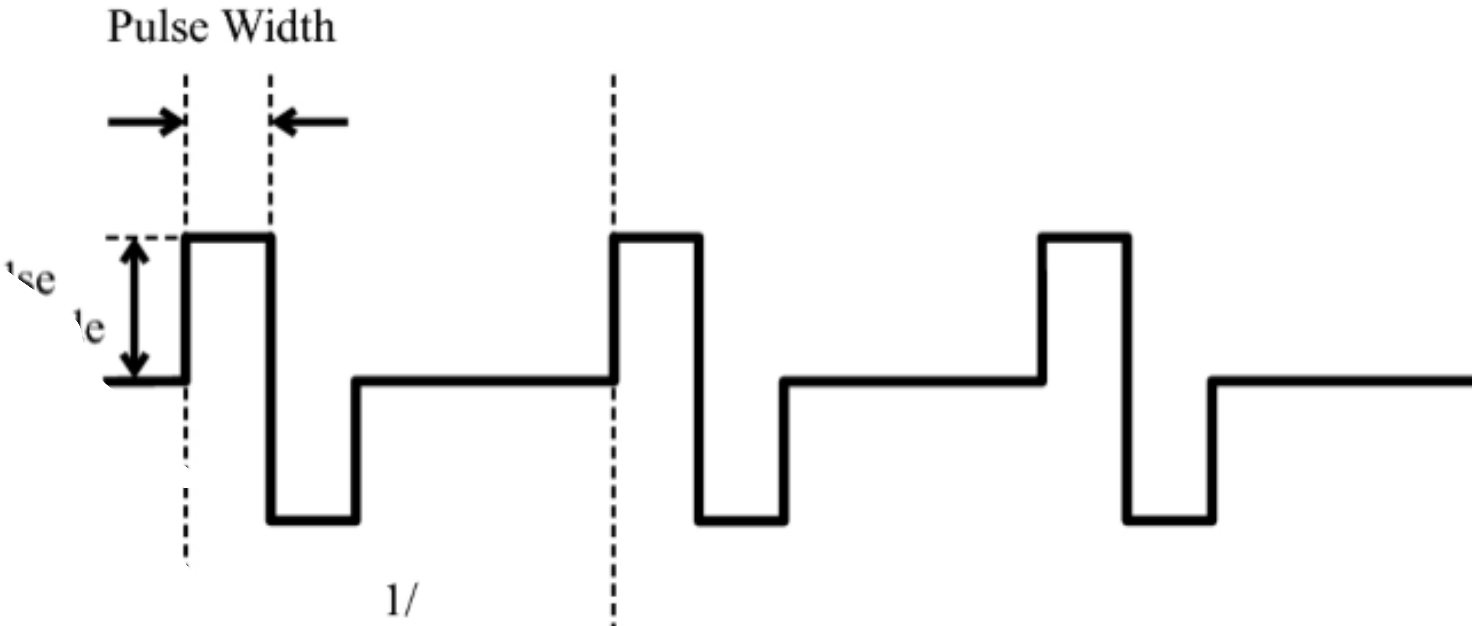
Some Comments on this Machine Model



- Non-Von-Neuman Architecture
  - No explicit memory in the machine model (memory is the channel)→ memory / I/O walls are a real problem or the results of abstractions.
- Intermediate measurements are allowed by this machine model without going back to the CPU.

Pulse-Level Quantum Computing Abstractions

- **Pulse / Stimuli:** A waveform with specific characteristics (amplitude, phase, frequency, and duration) used to manipulate the state of a qubit.
  - Pulse Shapes: Various shapes like Gaussian, square, and DRAG (Derivative Removal by Adiabatic Gate) pulses are optimized for different operations.
- **Schedules:** Sequences of pulses applied to qubits over time, defining when and how pulses are delivered.
  - Timelines: Representation of pulse schedules on a timeline, indicating the precise timing of each pulse.
- **Channels:** Interfaces between the control electronics and the qubits through which pulses are delivered.



Example of Pulse-Level Programming Model: Qiskit Pulse

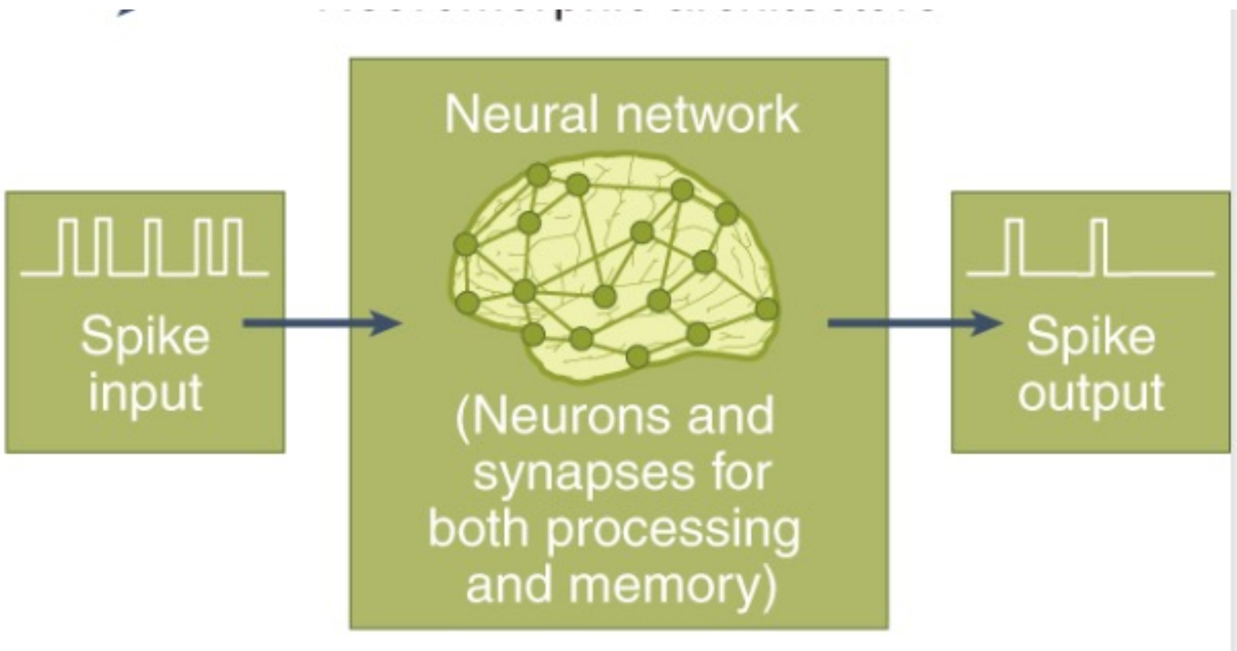
```python
from qiskit import pulse
from qiskit.pulse import Play, Schedule, DriveChannel, Gaussian

# Define a Gaussian pulse
pulse_duration = 128
sigma = 16
amplitude = 0.1
gaussian_pulse = Gaussian(duration=pulse_duration, amp=amplitude, sigma=sigma)

# Create a schedule to play the pulse
drive_channel = DriveChannel(0)
schedule = Schedule(name='Gaussian Pulse Schedule')
schedule += Play(gaussian_pulse, drive_channel)

print(schedule)
```

Waveform: Represents the shape of the pulse, such as Gaussian or drag, specified by parameters like amplitude and duration.

DriveChannel: A specific channel used to send pulses to a qubit.

Pulse Schedule: a schedule defines a sequence of pulse instructions.

Have we seen this before?



- **Spiking neural networks and neuromorphic computing**:
  - Physics-based Programming Model: Stimuli and Response
  - Collocated computing and memory
  - Asynchronous Event-driven execution model

Comparison Quantum Computing

Concept	Pulse-Level Quantum Computing	Neuromorphic Computing (Spiking Neural Networks)
Pulse	**Definition**: Waveform used to manipulate qubits via specific characteristics (amplitude, frequency, phase, duration).	**Definition**: Electrical spikes (or action potentials) that represent discrete events in time, used to transmit information.
	Types: Gaussian, square, DRAG pulses, etc.	**Types**: Fixed or variable spike shapes, typically resembling biological neuron spikes.
Schedule	**Definition**: Sequences of pulses applied to qubits over a timeline, specifying when and how pulses are delivered.	**Definition**: Timing of spikes generated by neurons, representing the sequence and temporal dynamics of neuronal firing.
	Example: A schedule for a CNOT gate might include a series of precisely timed microwave pulses.	**Example**: Spike-timing-dependent plasticity (STDP) where the timing of spikes influences synaptic strength.
Channel	**Definition**: Physical or logical pathways through which pulses are delivered to qubits (e.g., DriveChannel, ControlChannel).	**Definition**: Pathways through which spikes travel between neurons, typically represented by synapses and axons.
	Function: Interfaces between control electronics and qubits for precise pulse delivery.	**Function**: Connects neurons, enabling the propagation of spikes and communication between different parts of the neural network.
	Implementation: Microwave transmission lines, optical fibers, or other control mechanisms.	**Implementation**: Synaptic connections in artificial neurons, often implemented using analog or digital circuits.

Connection between pulse-level QC and Neuromorphic Programming Models

- Physics-based programming models – as envisioned by all the fathers of QV
  - Avoiding implicit problems with I/O and Memory Wall
  - More expressiveness
  - Seamless progress towards qudit systems
- Cross-fertilization between QC and neuromorphic
  - Especially in the area of quantum machine learning
  - More biologically inspired neural networks are designed
- A more physics-based approach that is more natural for Quantum simulations and quantum machine learning approaches.
  - Quantum circuit-based approaches are integrated at higher levels to generate more general-purpose quantum computing.

Disadvantages

- Difficult to identify explicit sources of quantum advantages
  - We have already seen this in determining quantum machine learning.
  - Pulses and channels are too low-level for easy theoretical analysis.
  - It is challenging to identify superposition and quantum parallelism.
- Synchronization and scheduling are major challenges.
- Portability might be a challenge, but standards start developing.

Conclusions

- To think about quantum machine models and associated programming models is critical for the progress of quantum computing.
  - Offloading the quantum circuit model might hinder progress on understanding the advantages and limitations of quantum software and algorithms.
    - I/O bottleneck is real?
- Pulse-level quantum machine models have several approaches similar to those of neuromorphic systems programming.
  - It allows for co-design with neuromorphic systems and applications.