

# Quantum Computer Programming, Compilation, and Execution with XACC

Alex McCaskey

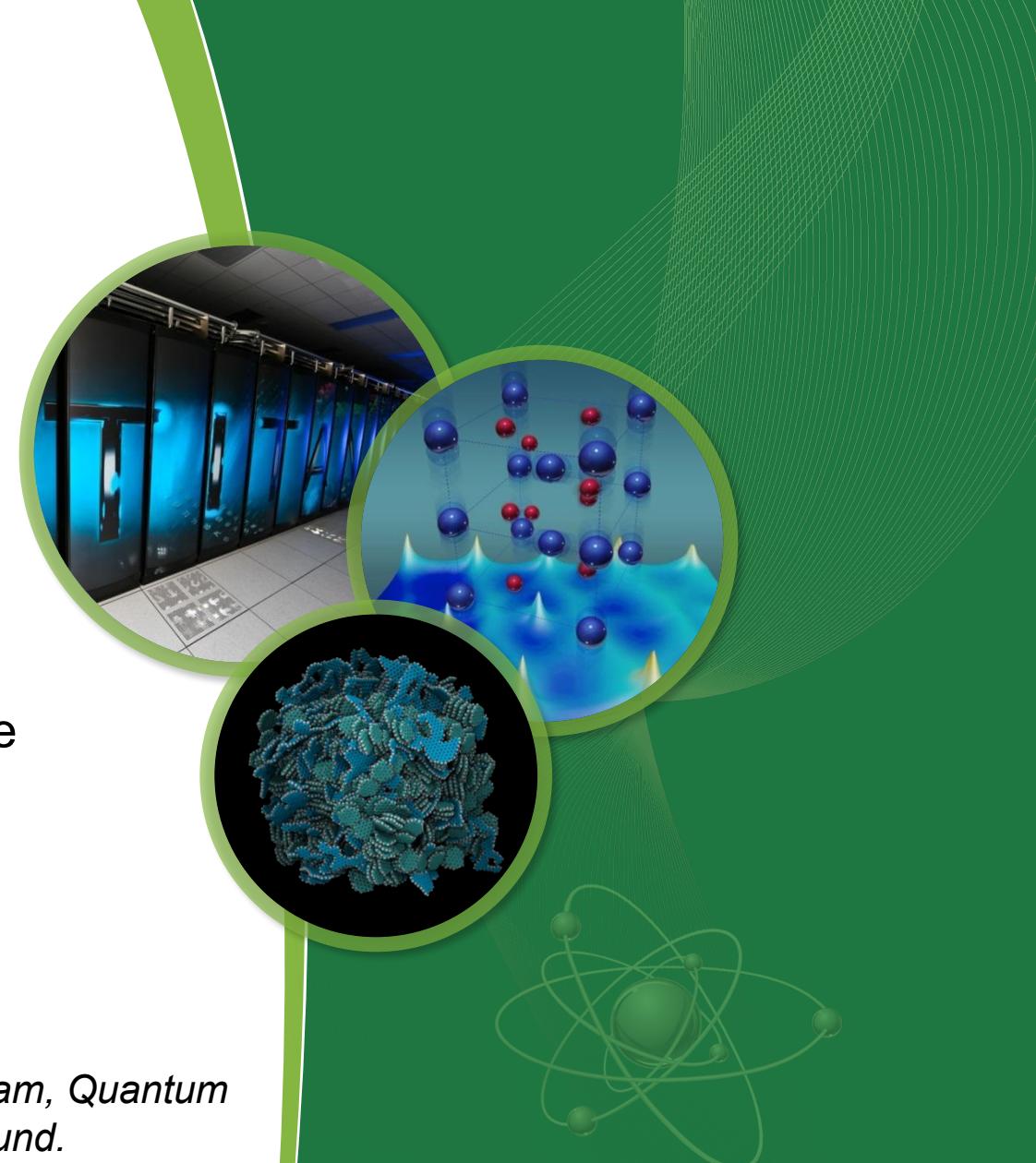
System and Software Lead, Quantum Computing Institute  
Oak Ridge National Laboratory

[mccaskeyaj@ornl.gov](mailto:mccaskeyaj@ornl.gov)

with Eugene Dumitrescu, Dmitry Liakh, and Travis Humble

*ORNL Software Expo Tutorial, May 2018*

*This work is supported by the DOE ASCR Early Career Research Program, Quantum Algorithms Teams, Quantum Testbed Pathfinder and the ORNL LDRD fund.*



# Outline of Today's Tutorial...

- Quantum Computer Programming
- XACC - Motivation and Architecture
- Programming Interesting Hybrid Algorithms
  - Variational Quantum Eigensolver (VQE)
- Hands-on Demonstration
  - Compute ground state energy of H<sub>2</sub>
  - Compute binding energy of deuteron (time permitting)



# Logistics for the Demo

- Docker image with XACC, Openfermion, Psi4 and Jupyter Lab

```
$ git clone https://github.com/eclipse/xacc
$ cd xacc/docker/expo_tutorial
$ docker-compose pull
$ docker-compose up -d
```

Navigate to localhost:9000 on your browser!

```
$ (after tutorial) docker-compose down
```



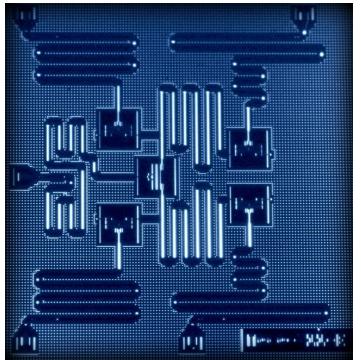
<https://quantumexperience.ng.bluemix.net>



<https://www.rigetti.com/forest>

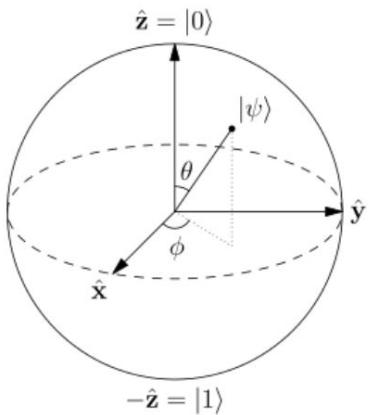


# Quantum Computer Programming

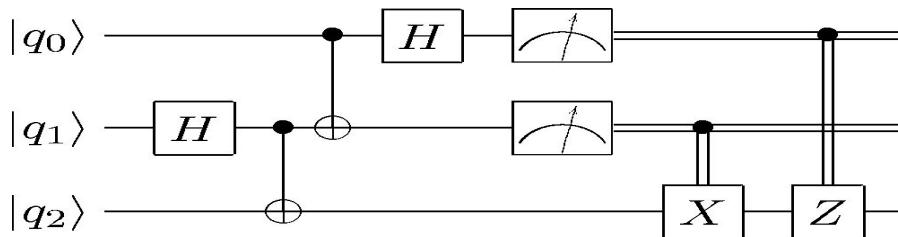


## Gate Model

- Gates operate on qubits, modify qubit state
- Common gates are Rotations, entangling gates, and superposition gates
- Measure!

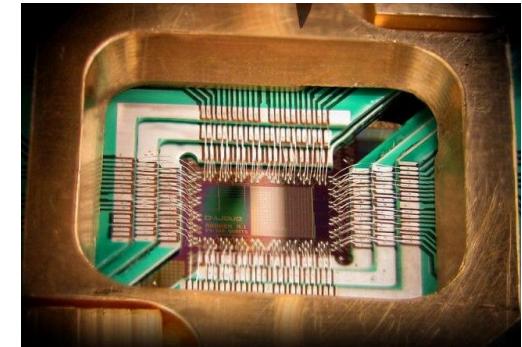


```
X(qreg[0]);  
H(qreg[1]);  
CNOT(qreg[1],qreg[2]);  
CNOT(qreg[0],qreg[1]);  
H(qreg[0]);  
cbit c1 = MeasZ(qreg[0]);  
cbit c2 = MeasZ(qreg[1]);
```



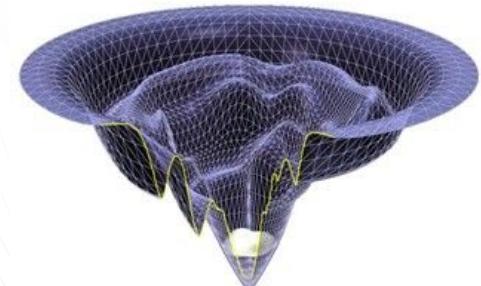
## Quantum Annealing Model

- Encode problem into energy landscape -  $h_i$  and  $J_{ij}$
- Suitable for (Ising type) satisfiability problems



0 0 20;  
1 1 50;  
2 2 60;  
4 4 50;  
5 5 60;  
6 6 -160;  
1 4 -1000;  
2 5 -1000;  
0 4 -14;  
0 5 -12;  
0 6 32;  
1 5 68;  
1 6 -128;  
2 6 -128;

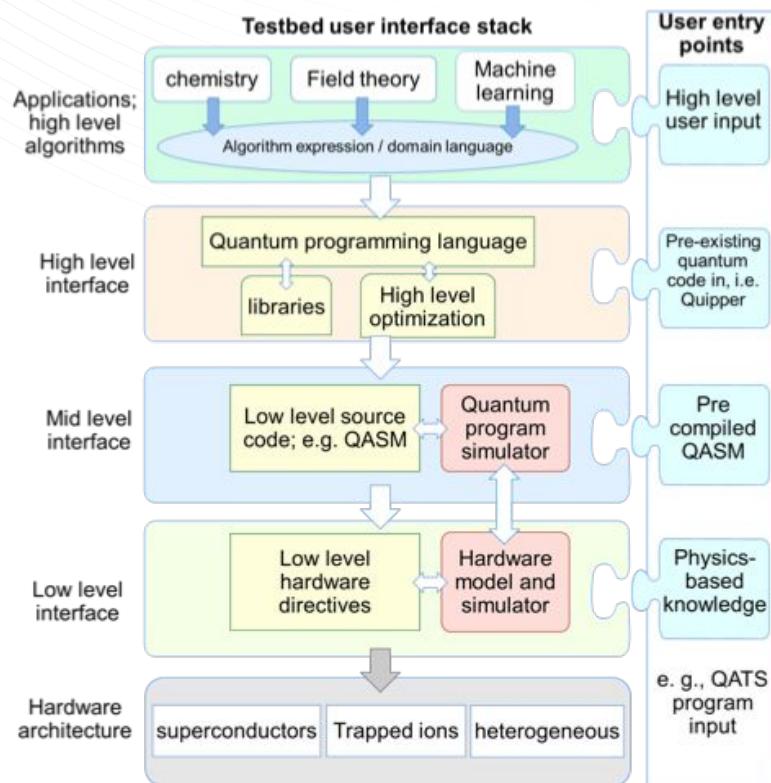
$$E(x_1, \dots, x_N) = \sum_{i=1}^N h_i x_i + \sum_{i < j=1}^N J_{ij} x_i x_j$$



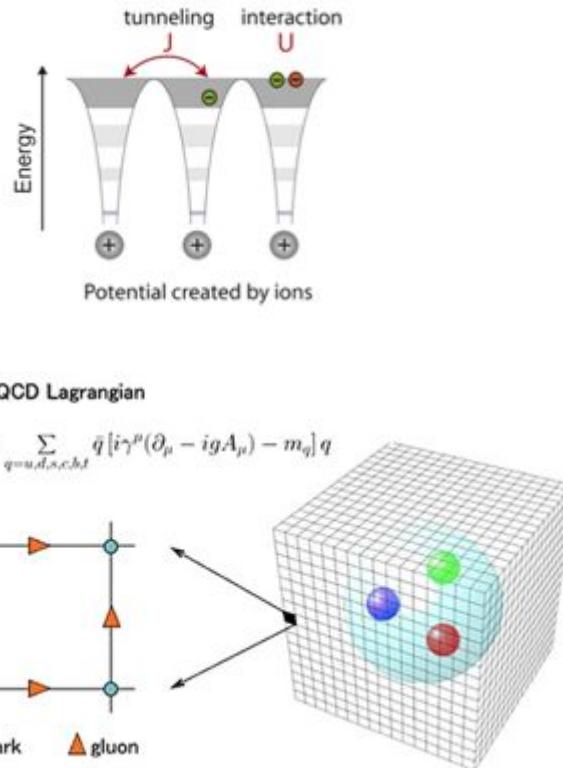
# Quantum Software Efforts at ORNL

- What is driving ORNL quantum programming efforts?

## DOE Testbed Project - Materials and Interfaces for Quantum Acceleration of Science Applications (MIQASA)



## DOE QAT - Heterogeneous Digital-Analog Quantum Dynamics Simulation (HDAQDS)



## ORNL Programming Requirements

- Anticipation of ORNL Post-Exascale Computing
  - treat QPUs as accelerators
- MIQASA benchmarking across QPU hardware types
  - enable hardware-agnostic programming
- HDAQDS execution across QPU types and high-level program expression
  - extensibility in compilation techniques

# Quantum Programming with XACC

Users define Quantum Kernels (Kernel in the GPU sense, a C-like Function)

```
__qpu__ quantum_kernel_foo(AcceleratorBuffer  
qubit_register, Param p1, ..., Param pN);
```



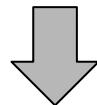
## XACC Compiler

Scaffold

Quil

OpenQasm

**Compiler Extension Point**  
(Map high-level kernels to IR)



Readout Error

Qubit Connectivity

Logical-to-Physical

**IR Transformations Extension Point**



Rigetti Forest

IBM QE

TNQVM

**Accelerator Extension Point**

## XACC:

- Open-source at <https://github.com/eclipse/xacc>
- Familiar API and model
  - OpenCL-like, hardware agnostic
- Key Abstractions
  - Kernels
  - Compilers
  - Intermediate Representation
  - Accelerators
- OSGI C++ Plugin Architecture (Python API available)

```
// Get reference to the QPU, and allocate a buffer of qubits
auto qpu = xacc::getAccelerator("ibm");
auto buffer = qpu->createBuffer("qreg", 2);

// Create and compile the Program from the kernel
// source code. Get executable Kernel source code
auto kernelSourceCode = "__qpu__ foo(double theta) {....}";
xacc::Program program(qpu, kernelSourceCode);
program.build();
auto kernel = program.getKernel<double>("foo");

// Execute over theta range
for (auto theta : thetas) kernel.execute(buffer, theta);
```

# XACC Kernels

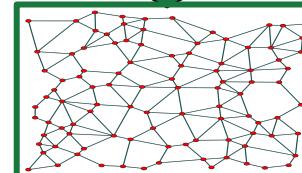
## XACC - Heterogeneous CPU-QPU Programming Model

```
__qpu__ quantum_kernel_foo(AcceleratorBuffer qubit_register,  
                           Param p1, ..., Param pN);
```

Circuit

$$U_{\text{kernel}}|\psi_{\text{register}}\rangle$$

QUBO



### Example Gate Model Kernel written in Scaffold

```
__qpu__ quantum_kernel_foo(AcceleratorBuffer qubit_register,  
                           Param p1, ..., Param pN);
```

#### XACC Kernel Requirement

#### Description

Annotation

All kernels must be annotated with the `__qpu__` function attribute to enable static, ahead-of-time compilation

Kernel Name

All kernels must be given a unique name

Accelerator Buffer Argument

All kernels must take as a first function argument the Accelerator Buffer this kernel acts on.

Runtime Parameters

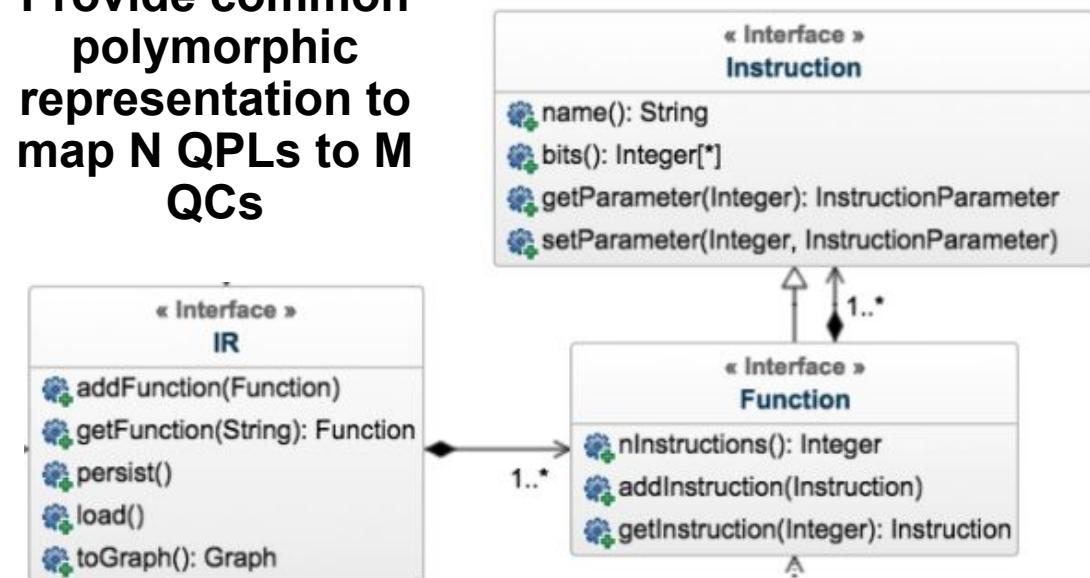
All kernels can take any number of runtime arguments.

```
__qpu__ teleport (qbit& qreg) {  
    X(qreg[0]);  
    H(qreg[1]);  
    CNOT(qreg[1],qreg[2]);  
    CNOT(qreg[0],qreg[1]);  
    H(qreg[0]);  
    cbit c1 = MeasZ(qreg[0]);  
    cbit c2 = MeasZ(qreg[1]);  
    if(c1 == 1) Z(qreg[2]);  
    if(c2 == 1) X(qreg[2]);  
}
```

# XACC Intermediate Representation

- Instructions
  - operate on qubits, unique names, parameterized
- Functions
  - Are Instructions but also contain further Instructions - Composite Pattern
  - Compiled representation of Kernel functions
- IR
  - Container of Functions
  - IR Transformations and IR Preprocessors

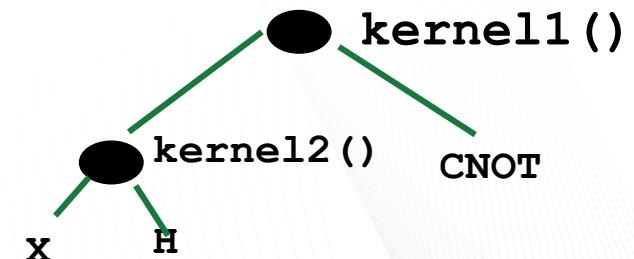
Provide common polymorphic representation to map N QPLs to M QCs



```
qpu kernel2(qbit qreg) {
    X(qreg[0]);
    H(qreg[1]);
}

qpu kernel1(qbit qreg) {
    kernel2(qreg);
    CNOT(qreg[0], qreg[1]);
}
```

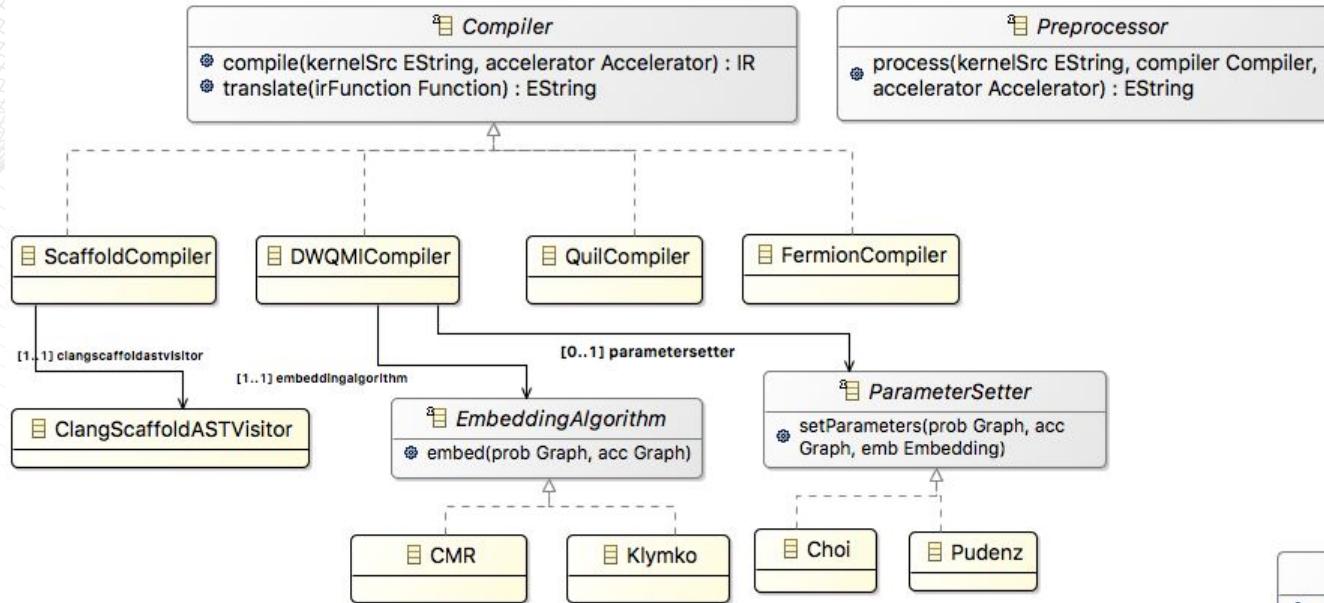
Simple kernels written in Scaffold



XACC IR models an *n*-ary tree, and we walk (pre-order) that tree to perform translations, optimizations, and executions



# Compilers and Accelerators



**Compilers take source code and Accelerator, produce XACC IR**

**Compilers provide source-to-source translation capabilities**

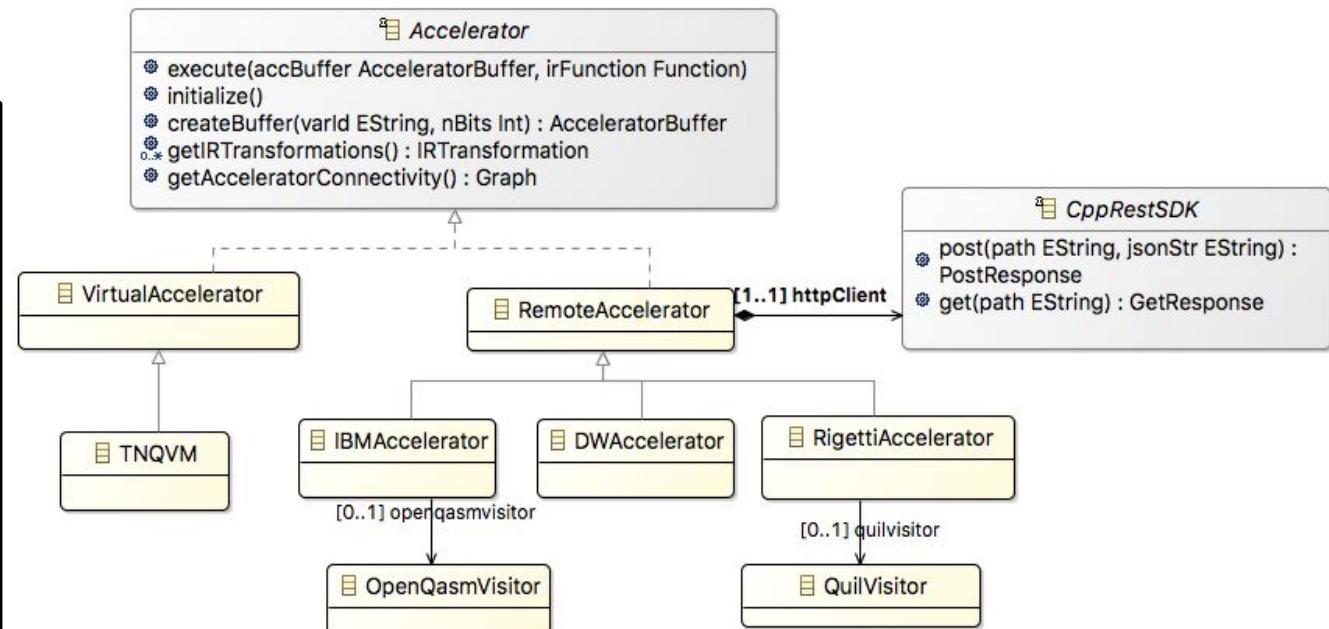
**Extensible Preprocessor mechanism**

**Accelerators execute XACC IR**

**Can be simulators, physical QCs, remote or local**

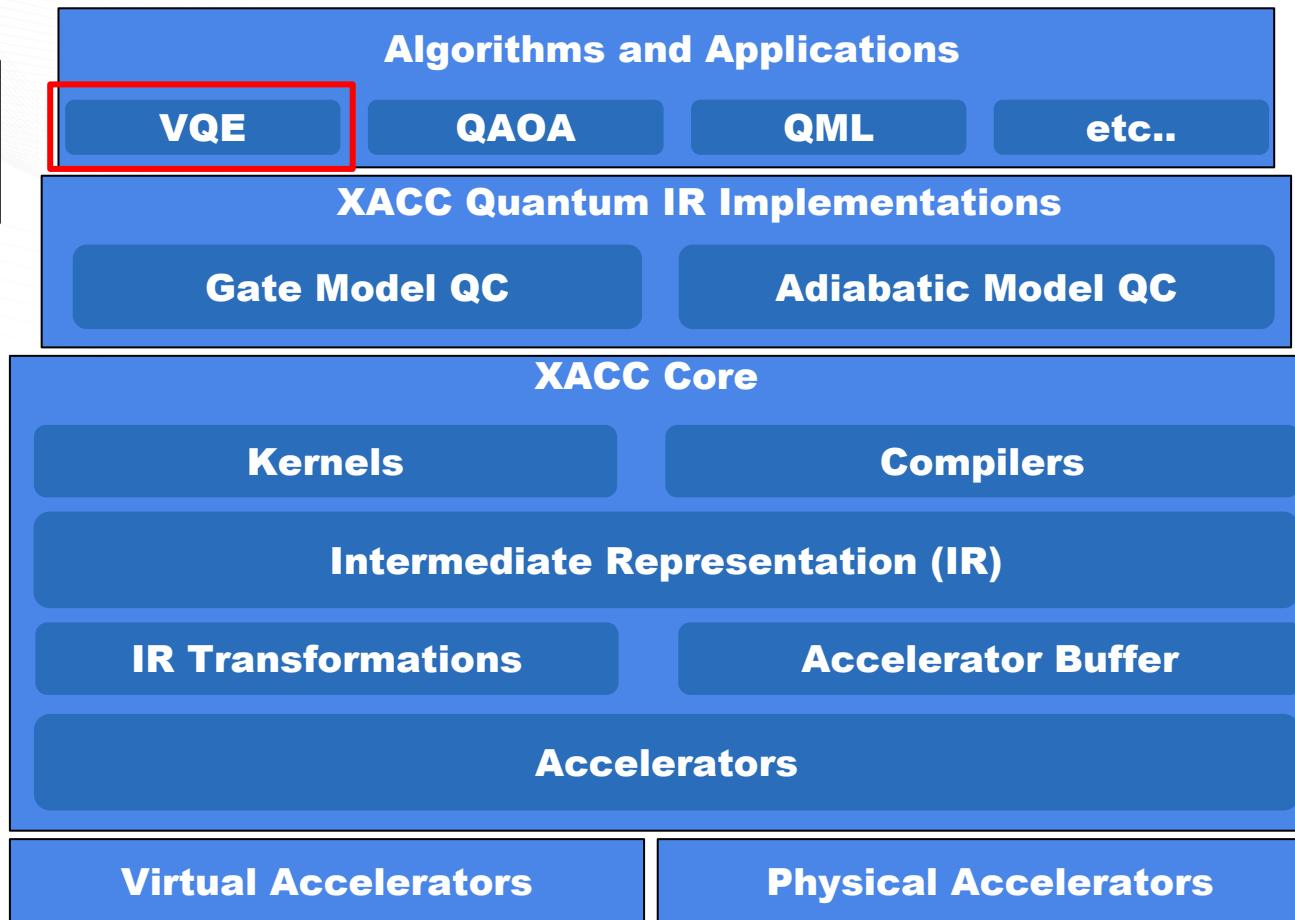
**Remote Accelerators operate with HTTP Rest Client**

**Visitors walk IR, produce native code**

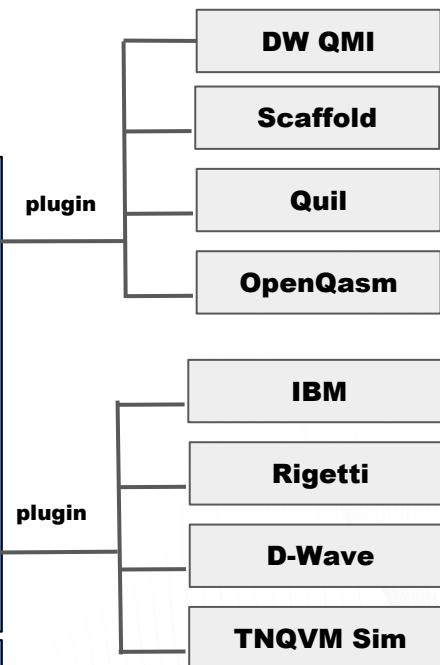


# Current XACC Application Stack

We'll be using  
this App for our  
Demo!



Key architectural design:  
Plugin infrastructure



XACC is the **ONLY** quantum programming model and framework that enables hardware-agnostic quantum programming, compilation, and execution. ORNL owns the only integration framework for quantum computing.

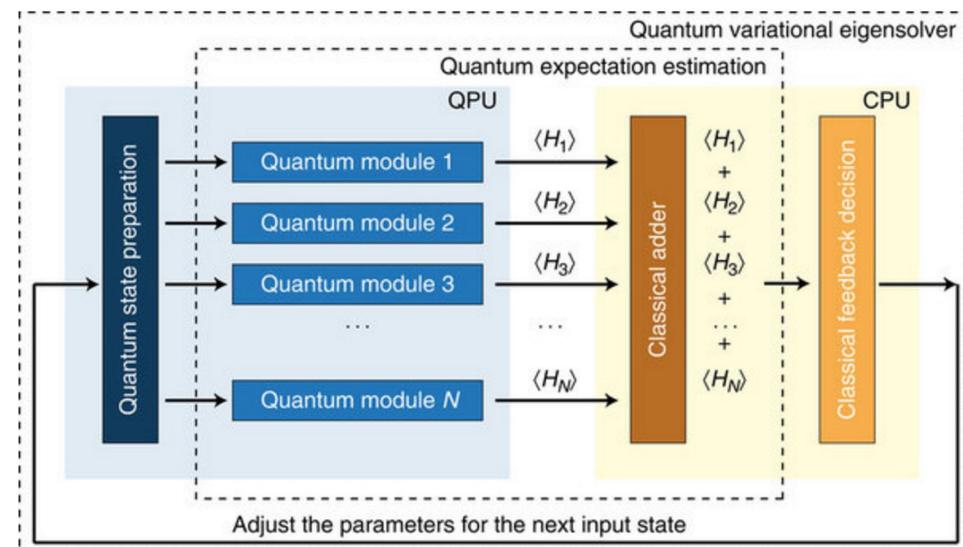
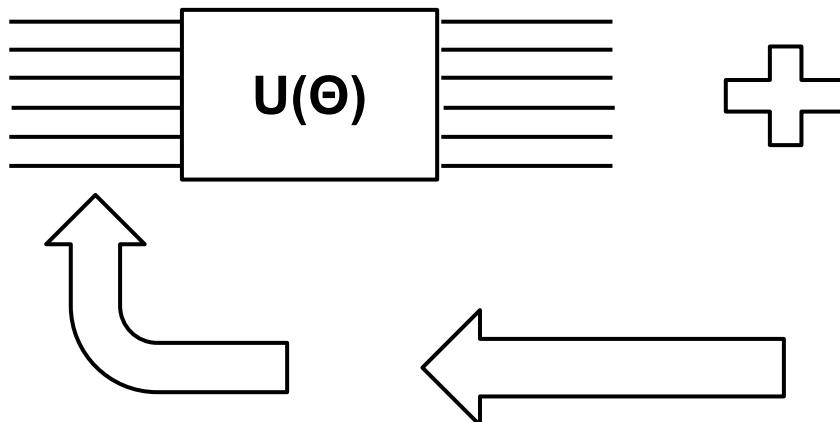
# Variational Quantum Eigensolver (VQE)

McClean 2014

- Primary workhorse algorithm for research at ORNL
  - Compute eigenvalues of  $\mathbf{H}$
  - resistant to system level noise
  - **Hybrid** - leverages classical and quantum resources
  - Classical nonlinear optimization (search for ground state by optimizing classical parameters)

$$H = \sum_{p,q} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{p,q,r,s} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s$$

Parameterized Circuit - Explores  $\mathbf{H}$  Vector Space via controllable  $\Theta$



# VQE Programming Steps

- Define your Hamiltonian
  - for  $H_2$  / general quantum chemistry, we will use Openfermion and Psi4 quantum chemistry package
- Define your  $U(\Theta)$ 
  - $H_2$  - UCCSD auto generated by XACC
- Define classical optimization routine
  - We will use Nelder-Mead
- XACC handles compilation and execution on the Accelerator of your choice
  - TNQVM, IBM, Rigetti

$$H = \sum_{p,q} h_{pq} a_p^\dagger a_q + \frac{1}{2} \sum_{p,q,r,s} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s$$

Psi4 provides an easy way to generate hpq,  
hpqrs



OpenFermion provides an easy to use FermionOperator data structure, which XACC can compile



# XACC-VQE

- Application built on XACC for VQE
  - <https://github.com/ornl-qci/xacc-vqe>
  - xacc-vqe binary executable
- Task-based architecture
  - compute-energy, vqe, diagonalize, etc.
- 2 API functions in Python
  - compile(), execute()
  - Both take as input a representation of the problem Hamiltonian
  - execute kwargs
    - ansatz, n-electrons, task, accelerator, etc...

```
ham = PauliOperator(5.906709445) +
      PauliOperator({0:'X',1:'X'}, -2.1433) +
      PauliOperator({0:'Y',1:'Y'}, -2.1433) +
      PauliOperator({0:'Z'}, .21829) +
      PauliOperator({1:'Z'}, -6.125)

src = """__qpu__ ansatz(AcceleratorBuffer, double t0) {
X 0
RY(t0) 1
CNOT 1 0
}"""
ansatz = xacc.Program(qpu,src).getKernel('f').getIRFunction()

minEnergy = vqe.execute(ham, **{'ansatz':ansatz, 'accelerator':'ibm', \
                                'task':'vqe', 'n-qubits':12, \
                                'error-mitigation':['correct-readout-errors'], \
                                'qubit-map':[10,11]}).energy
```

```
ibmqx5Result = vqe.execute(H_f, **{
    'task':'compute-energy',
    'vqe-params':str(tnqvmResult.angles[0]), 'ansatz':statePrep,
    'accelerator':accelerator, 'qubit-map':[10,9],
    'error-mitigation':['correct-readout-errors']
})
print('E = ', ibmqx5Result.energy)
```

A screenshot of the JupyterLab interface. On the left, a file browser shows a list of files and folders: 'deuteron' (modified a minute ago), 'docs' (modified a minute ago), 'h2' (modified seconds ago), 'Y: docker-compose.yml' (modified an hour ago), 'Dockerfile' (modified 12 minutes ago), and 'README.md' (modified 3 days ago). Below the file browser are two sections: 'Launcher' containing 'Notebook' (Python 3) and 'Console' (Python 3), and 'Other' containing 'Terminal' and 'Text Editor'. In the center, the word 'Demonstration' is displayed in large, bold, black font. To the right of the title, several terminal commands are listed:

```
$ git clone https://github.com/eclipse/xacc
$ cd xacc/docker/expo_tutorial
$ docker-compose pull
$ docker-compose up -d
```

Below the commands, the text 'Navigate to localhost:9000 on your browser!' is displayed. Further down, the text 'Start a new notebook or open the already worked examples in the h2/ and deuteron/ folders' is shown. At the bottom, the command '\$ (after tutorial) docker-compose down' is listed.

# Accelerator Configuration

If you would like to run on Rigetti Forest or IBM Quantum Experience, go to their respective sites and request free access.

<https://quantumexperience.ng.bluemix.net/>  
<https://www.rigetti.com/forest>

Open a Terminal in the JupyterLab instance and run the commands to left to register your credentials with XACC

Otherwise, default Accelerator for this Demo will be the TNQVM tensor network simulator!

```
H2.ipynb    x  $ root@03a1314e x
root@03a1314ef7dc:/# python -m pyxacc --help
usage: pyxacc.py [-h] [-c SET_CREDENTIALS] [-k API_KEY] [-u USER_ID] [--url URL] [-L] [--python-include-dir] [-b BRANCH]
XACC Framework Utility.

optional arguments:
  -h, --help            show this help message and exit
  -c SET_CREDENTIALS, --set-credentials SET_CREDENTIALS
                        Set your credentials for any of the remote Accelerators. (default: None)
  -k API_KEY, --api-key API_KEY
                        The API key for the remote Accelerators. (default: None)
  -u USER_ID, --user-id USER_ID
                        The User Id for the remote Accelerators. (default: None)
  --url URL            The URL for the remote Accelerators. (default: None)
  -L, --location        Print the path to the XACC install location. (default: False)
  --python-include-dir Print the path to the Python.h. (default: False)
  -b BRANCH, --branch BRANCH
                        Print the path to the XACC install location. (default: master)
root@03a1314ef7dc:/# python -m pyxacc -c ibm -k YOUR_API_KEY
```

## IBM Credentials

```
H2.ipynb    x  $ root@03a1314e x
root@03a1314ef7dc:/# python -m pyxacc --help
usage: pyxacc.py [-h] [-c SET_CREDENTIALS] [-k API_KEY] [-u USER_ID] [--url URL] [-L] [--python-include-dir] [-b BRANCH]
XACC Framework Utility.

optional arguments:
  -h, --help            show this help message and exit
  -c SET_CREDENTIALS, --set-credentials SET_CREDENTIALS
                        Set your credentials for any of the remote Accelerators. (default: None)
  -k API_KEY, --api-key API_KEY
                        The API key for the remote Accelerators. (default: None)
  -u USER_ID, --user-id USER_ID
                        The User Id for the remote Accelerators. (default: None)
  --url URL            The URL for the remote Accelerators. (default: None)
  -L, --location        Print the path to the XACC install location. (default: False)
  --python-include-dir Print the path to the Python.h. (default: False)
  -b BRANCH, --branch BRANCH
                        Print the path to the XACC install location. (default: master)
root@03a1314ef7dc:/# python -m pyxacc -c rigetti -k YOUR_API_KEY -u YOUR_USER_ID
```

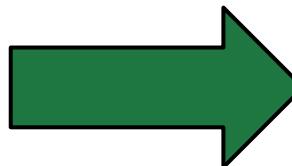
## Rigetti Credentials

# The Binding Energy of Deuteron

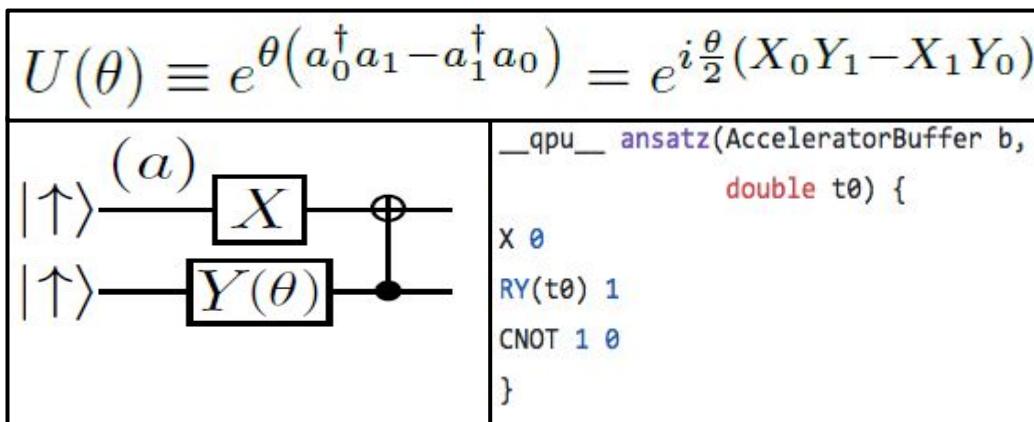
- Pionless EFT, discrete variable representation, HO basis, consider N=2,3, Jordan-Wigner
- We use variational quantum eigensolver (VQE) (over the cloud) method to identify the ground eigenstate (**arxiv:1801.03897, accepted PRL'18**)

## Deuteron N=2 Hamiltonian

$$H_2 = 5.906709I + 0.218291Z_0 - 6.125Z_1 \\ - 2.143304(X_0X_1 + Y_0Y_1)$$



## State Preparation Ansatz



```
#pragma vqe-coefficient 5.906709445
__qpu__ idTerm(AcceleratorBuffer b) {
}
#pragma vqe-coefficient -2.1433
__qpu__ x0x1(AcceleratorBuffer b) {
H 0
H 1
MEASURE 0 [0]
MEASURE 1 [1]
}
#pragma vqe-coefficient -2.1433
__qpu__ y0y1(AcceleratorBuffer b) {
RX(1.57078) 0
RX(1.57078) 1
MEASURE 0 [0]
MEASURE 1 [1]
}
#pragma vqe-coefficient .21829
__qpu__ z0(AcceleratorBuffer b) {
MEASURE 0 [0]
}
#pragma vqe-coefficient -6.125
__qpu__ z1(AcceleratorBuffer b) {
MEASURE 1 [0]
}
```

# Deuteron - Higher-level programming and error mitigation

## Deuteron binding energy via standard XACC Python API

```
ham = PauliOperator(5.906709445) +
    PauliOperator({0:'X',1:'X'}, -2.1433) +
    PauliOperator({0:'Y',1:'Y'}, -2.1433) +
    PauliOperator({0:'Z'}, .21829) +
    PauliOperator({1:'Z'}, -6.125)

src = """__qpu__ ansatz(AcceleratorBuffer, double t0) {
    X 0
    RY(t0) 1
    CNOT 1 0
}"""
ansatz = xacc.Program(qpu,src).getKernel('f').getIRFunction()

minEnergy = vqe.execute(ham, **{'ansatz':ansatz, 'accelerator':'ibm', \
    'task':'vqe', 'n-qubits':12, \
    'error-mitigation':['correct-readout-errors'], \
    'qubit-map':[10,11]}).energy
```

### Of note with this code sample

- Unified VQE API
- Problem programming at high-level, i.e. define Hamiltonian
  - can define fermionic second quantized hamiltonians, XACC compiles to qubit representation
- Custom ansatz generation using Quill, Scaffold, OpenQasm, etc.
- Specify target Accelerator - IBM, Rigetti, TNQVM
- Built-in error mitigation (readout, arxiv:1612.02058, extrap. arxiv:1611.09301)
- Mapping to physical qubits (map to qubits with lowest two-qubit gate error rates)

## ...or how about program using OpenFermion and target IBM???

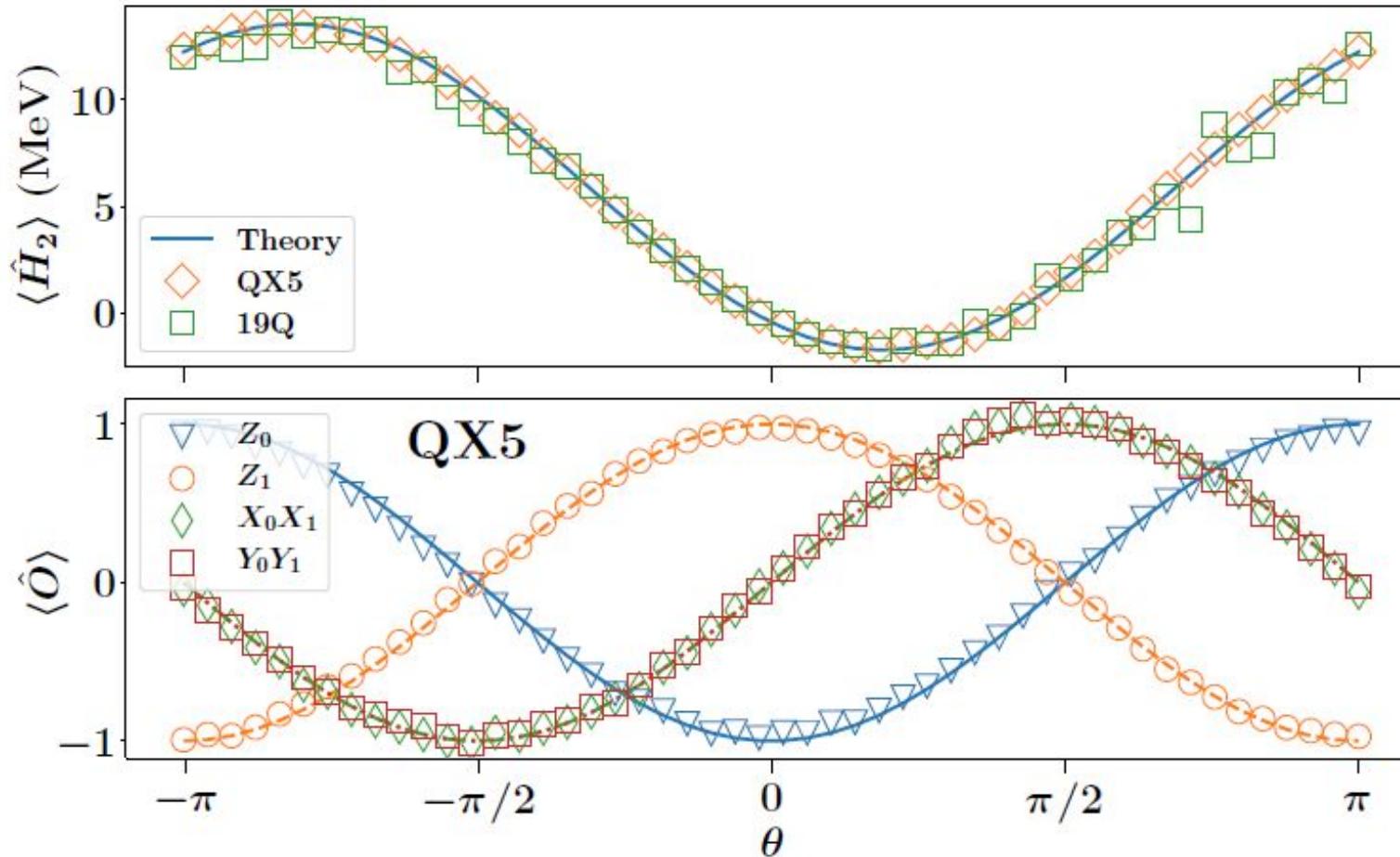
```
# Matrix elements
mat2=np.array([[ -0.43658111, -4.28660705],
               [ -4.28660705, 12.25      ]])

# Create the Hamiltonian with OpenFermion
H_f = FermionOperator()
for i,j in [(0,0),(0,1),(1,0),(1,1)]:
    H_f += FermionOperator('{0}^ {1}'.format(i,j), mat2[i,j])
```

```
ibmqx5Result = vqe.execute(H_f, **{
    'task':'compute-energy',
    'vqe-params':str(tnqvmResult.angles[0]), 'ansatz':statePrep,
    'accelerator':accelerator, 'qubit-map':[10,9],
    'error-mitigation':['correct-readout-errors']
})
print('E = ', ibmqx5Result.energy)
```

# The Binding Energy of Deuteron - Results

Cloud Quantum Computing of an Atomic Nucleus -  
<https://arxiv.org/abs/1801.03897>



## Top Panel

- This is the energy estimate of the theta-parameterized state
- Data taken on two processors, IBM and Rigetti

## Bottom Panel

- The weighted sum of these terms gives the energy estimate
- These results yield a binding energy of 2.18 MeV based on fits of Luscher's formula

**Thanks! Check out the project at:**

**<https://github.com/eclipse/xacc>**

**<https://xacc.readthedocs.io>**

**<https://arxiv.org/abs/1710.01794>**

**<https://hub.docker.com/r/xacc>**

**Email: [xacc-dev@eclipse.org](mailto:xacc-dev@eclipse.org)**

**Funded by ORNL LDRD, DOE Testbed Pathfinder, DOE  
Quantum Algorithms Teams, DOE Early Career Research  
Program**