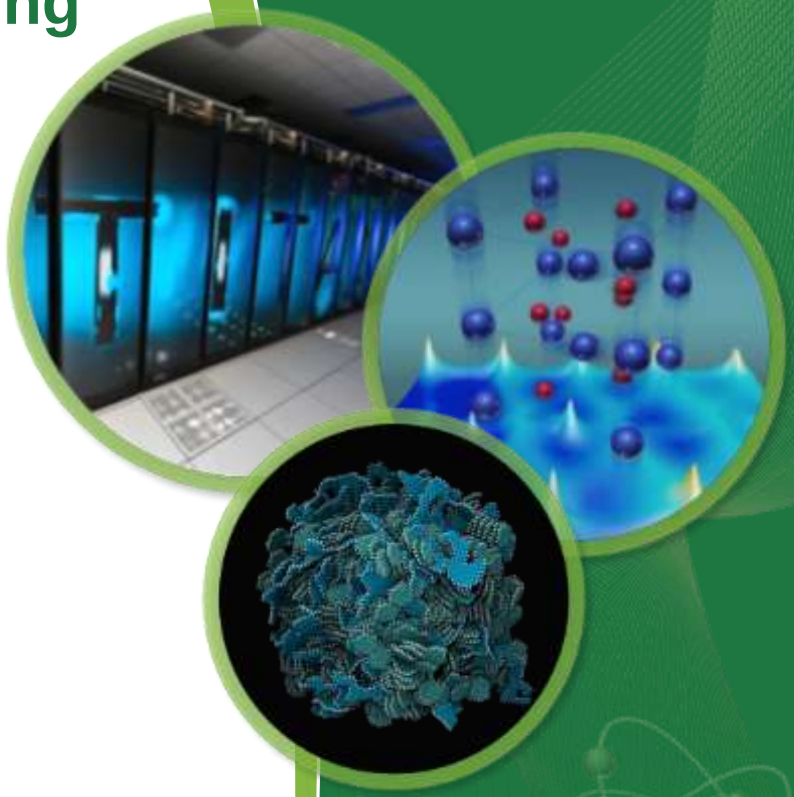
Presented to North Carolina State University

Systems and Software for Quantum Computing



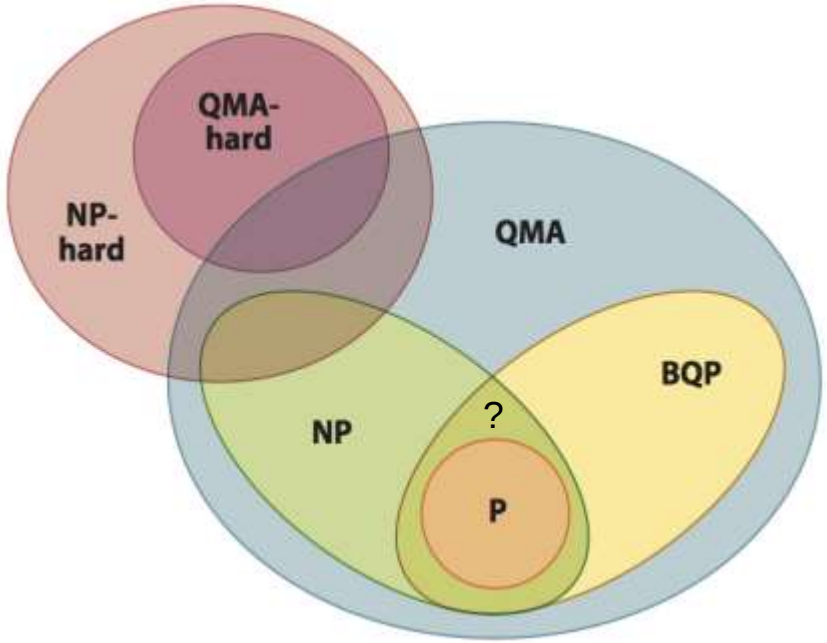
Travis Humble
Quantum Computing Institute
Oak Ridge National Laboratory

This work is supported by the DOE ASCR Early Career Research Program and the ORNL LDRD fund.

Presented 27 Feb 2018, Google Hangouts

Scientific Applications of Quantum Computing

- **Algorithms within the quantum computing model are found to take fewer steps to solve key problems**
  - Factoring
  - Unstructured Search
  - Eigensystems
  - Linear Systems
  - Quantum Simulation
  - Partition Functions
  - Discrete Optimization
  - Machine Learning
- **Several physical domains motivate quantum computing as a paradigm for scientific computing**
  - High-energy Physics
  - Materials Science
  - Chemistry
  - Biological Systems
  - Artificial Intelligence
  - Data Analytics
  - Planning and Routing
  - Verification and Validation



A complexity hierarchy hypothesis

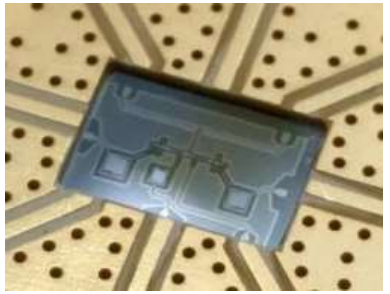
The relationship of BQP to other relevant classes is still largely uncertain.

- $P \subseteq BPP \subseteq BQP$
- Is BPP = BQP?
- Does BQP intersect NP?
- Does BQP intersect NP-Hard?

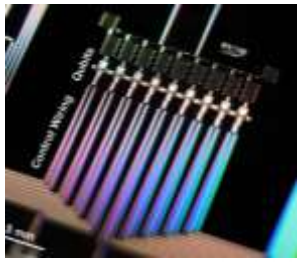
Current Quantum Processing Units

- **QPU's are devices that implement the principles of digital quantum computing**
  - Many different technologies demonstrated
  - Small-scale registers (1-50)
  - Very high 1-qubit gate fidelities (0.999+)
  - Moderately high 2-qubit gate fidelities (0.99+)
  - Limited connectivity, addressability
  - Sequences of operations demonstrated
  - Small-scale applications
- **Early stage vendors are offering QPU access**
  - D-Wave, IBM, IonQ, Google, Rigetti, etc.
  - Client-server interaction model
  - Very loose integration with modern computing

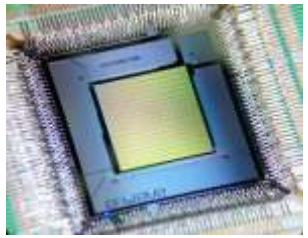
Superconducting chip from IBM



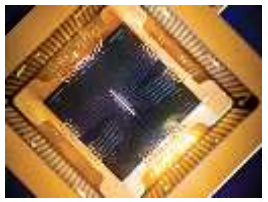
Superconducting chip from Rigetti



Superconducting chip from Google



Superconducting chip from D-Wave Systems



Ion trap chip from Sandia

Linear optical chip from Univ. Bristol/QET Labs

Modern Scientific Computing

- **State-of-the-art scientific computing is dominated by massively parallel processing**
  - Support large-scale linear algebra and pde problems for complex, multi-scale models
  - Application codes are parallelized and capable of utilizing distributed resources
  - Constrained by programming complexity, memory latency, and power consumption

*Top 5 HPC systems ranked by LINPACK benchmark (*Nov 2017*)*

System	R_{max}(PF)	Memory (TiB)	Power (MW)
TaihuLight	93.0	1,310	15.4
Tianhe-2	33.8	1,375	17.8
Piz Daint	19.6	340	2.2
Gyoukou	19.1	575	7.9
Titan	17.6	710	8.2

Titan HPC system composed from 18,688 nodes with a peak performance of 18 petaflops

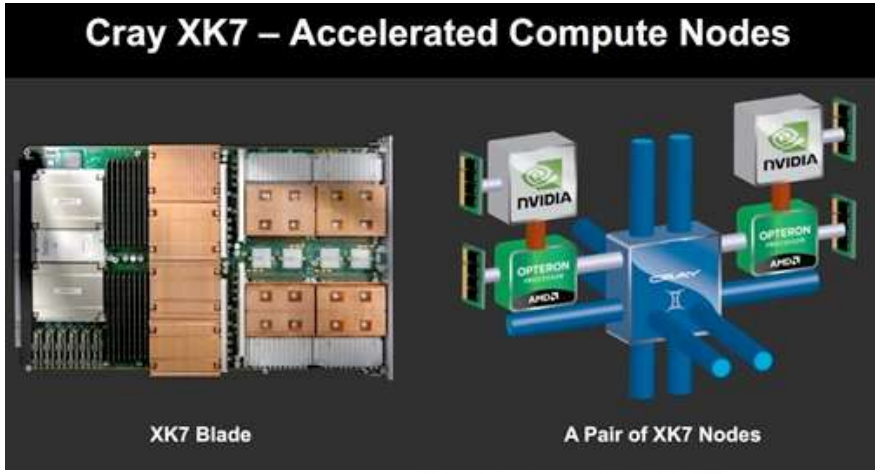
Summit HPC system is designed to have a peak performance of 200 petaflops

OAK RIDGE National Laboratory

Quantum-accelerated High-Performance Computing

- **We are addressing development of quantum computing co-design**
  - We are testing and tuning new programming and execution models
  - We are benchmarking against physical, computational, and scientific metrics
- **Are QPUs compatible with modern computing?**
  - When do QPU's accelerate applications relative to state of the art HPC?
  - What are the behavioral and functional requirements placed on the processor?

Computer node with interconnect for Titan architecture



Integration of existing QPUs faces engineering barriers

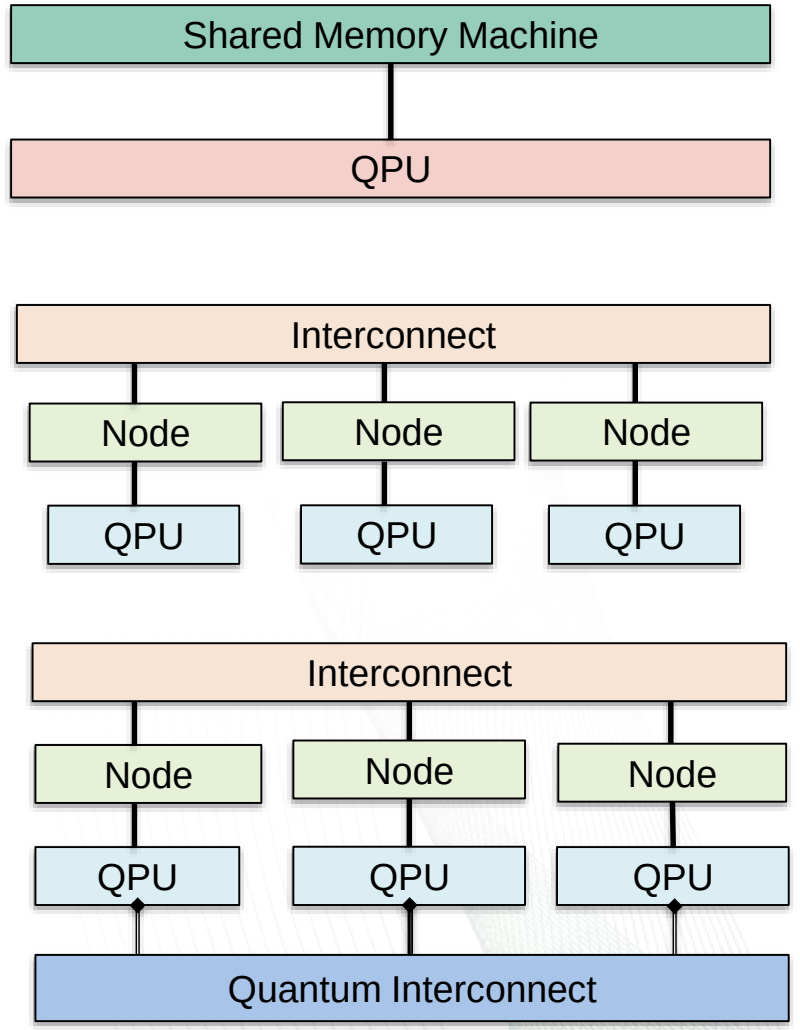
What is the architecture of these systems?

- **There are several possible architectures for an HPC system with QPUs**
  - Abstract machine models explore design alternatives, basis for performance expectations
  - The models are differentiated by how the quantum processing is partitioned
  - The architecture impacts the programming model, domain decomposition driven by algorithms
- **Architecture choices impact computational power**
  - Hilbert space for n nodes with q-qubit registers

$$n2^q \ vs. \ 2^{nq}$$

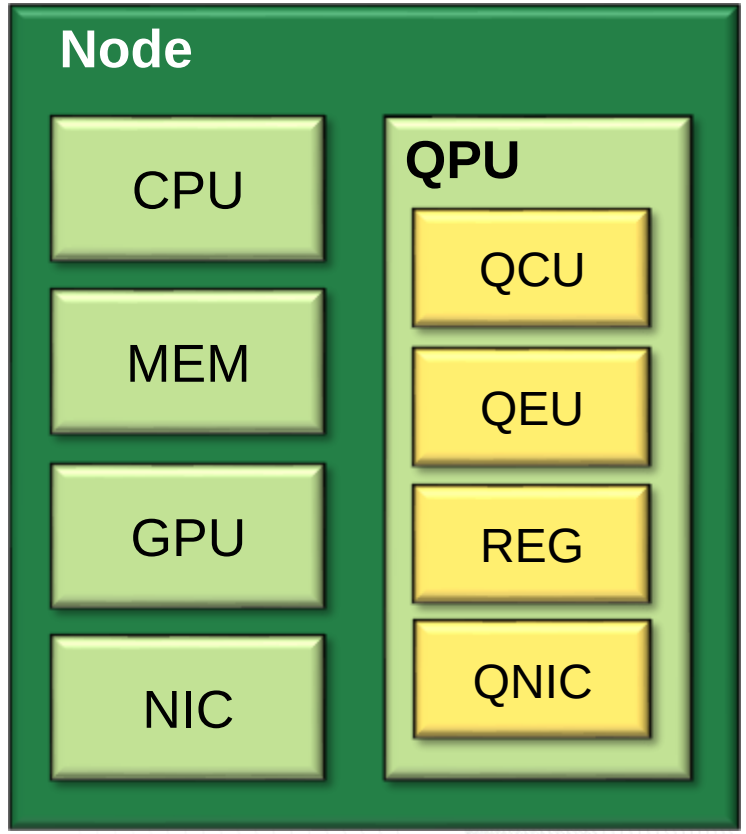
  - Communication costs for size m_q messages

$$m_q n \ vs. \ m_q n^2$$



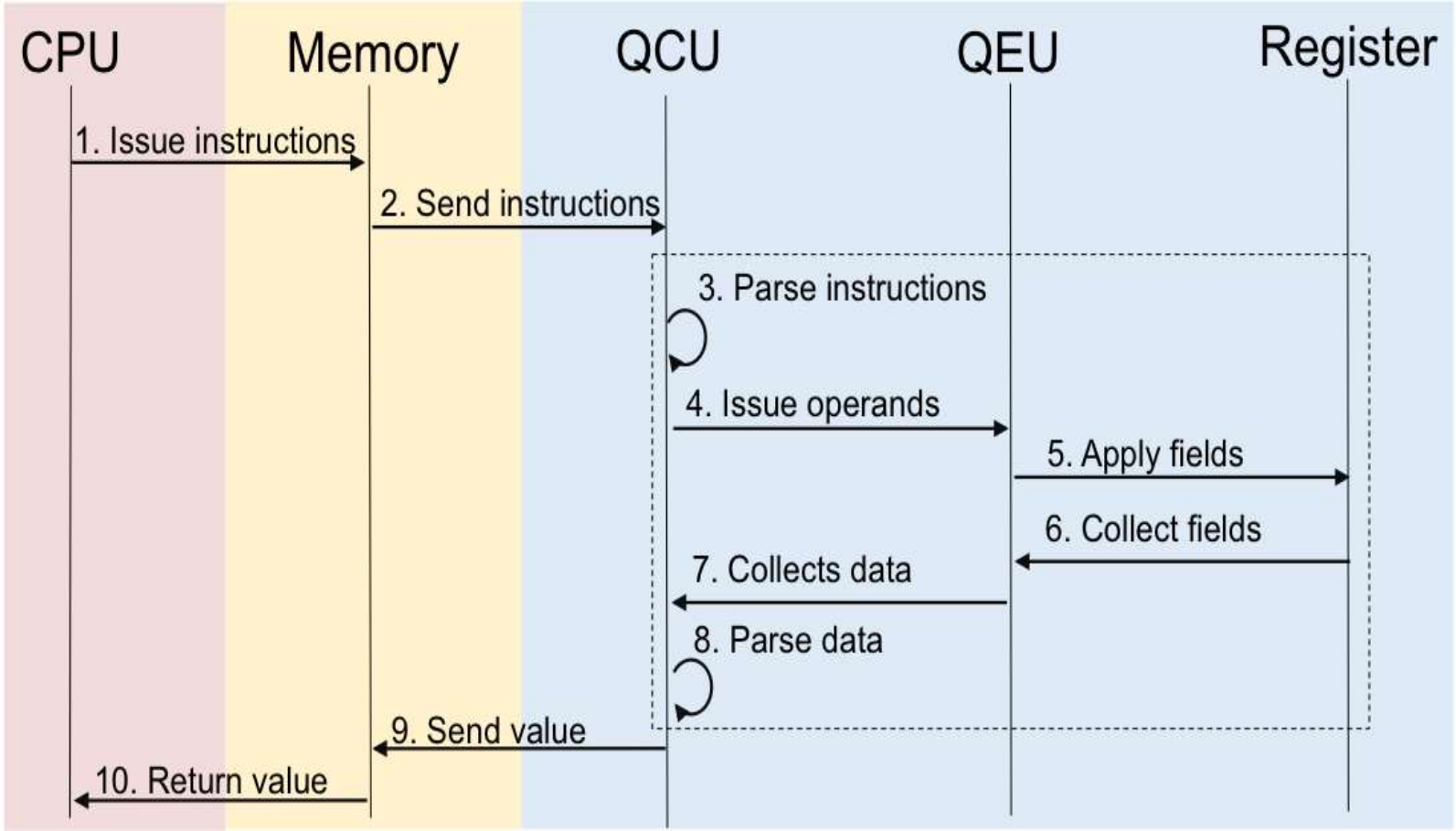
Quantum Accelerator Node Model

- A **node** may be composed from CPUs GPUs, and memory hierarchies as well as QPUs.
- The **quantum processing unit** (QPU) encompasses methods for parsing and executing quantum programs.
- The **quantum control unit** (QCU) parses instruction sent by the CPU.
- A **quantum execution unit** (QEU) applies fields to initiate gates. There may be multiple QEU's.
- Applied fields drive changes in the **quantum register**. The register state stores the value of the computation.
- **I/O** is based on fields to prepare and measure the register in computational basis states.
- **Network interfaces** for the conventional (NIC) and quantum (QNIC) interconnects support communication

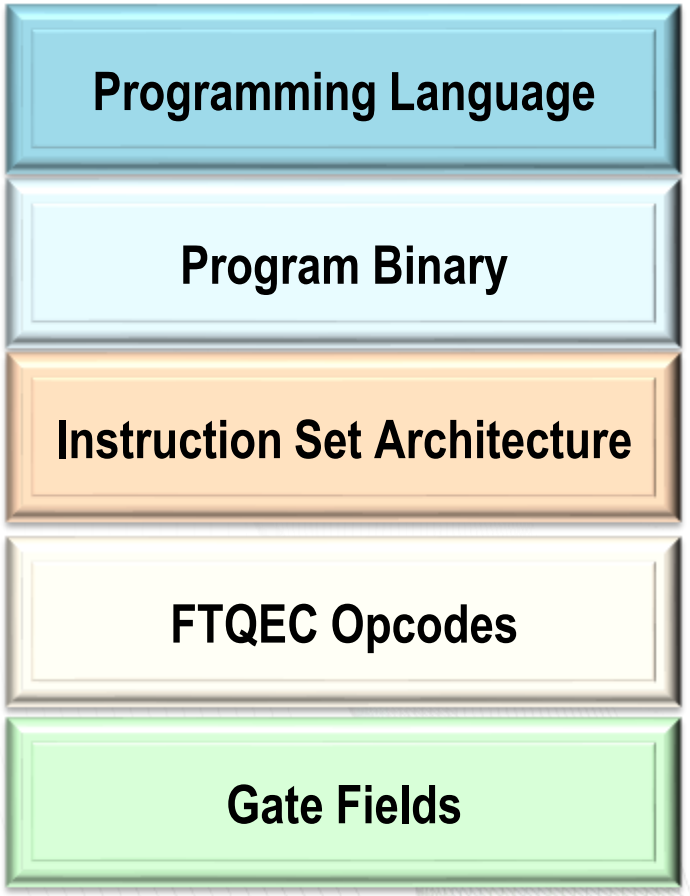


QPU Execution Model

- A typical interaction sequence between node components illustrates the language hierarchy for program execution



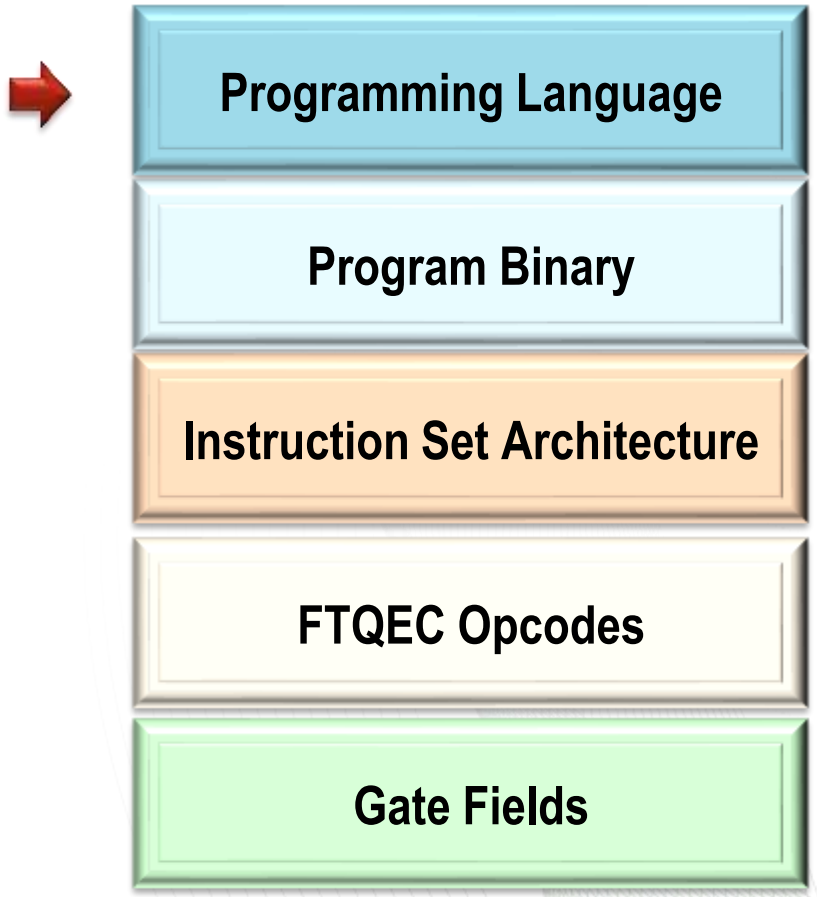
Language Hierarchy



Domain Specific Languages for QPUs

- **QPU's require unique language considerations**
  - Rigorous constraints on logical primitives
    - No cloning prohibits *memcpy*, = sign
    - Pure functions avoid entanglement side effects
  - Non-local communication primitives
    - Teleportation uses pre-allocated resources
  - Syntax varies with QPU operational models
- **Many existing quantum programming language (QPLs) largely address these concerns**
  - Embedded domain specific languages (DSL); Quipper (Haskell), Scaffold (C), LiQui|> (F#), ProjectQ (python)
  - All require expert knowledge of quantum computing and they do not integrate with existing workflows

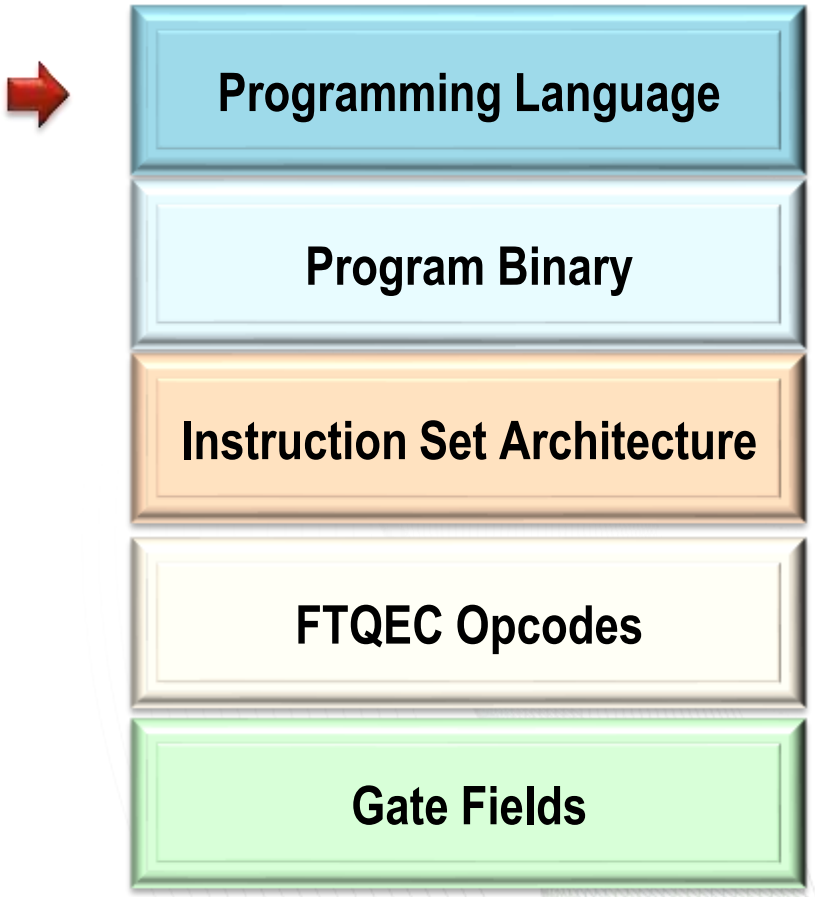
Language Hierarchy



Accelerator Programming Framework

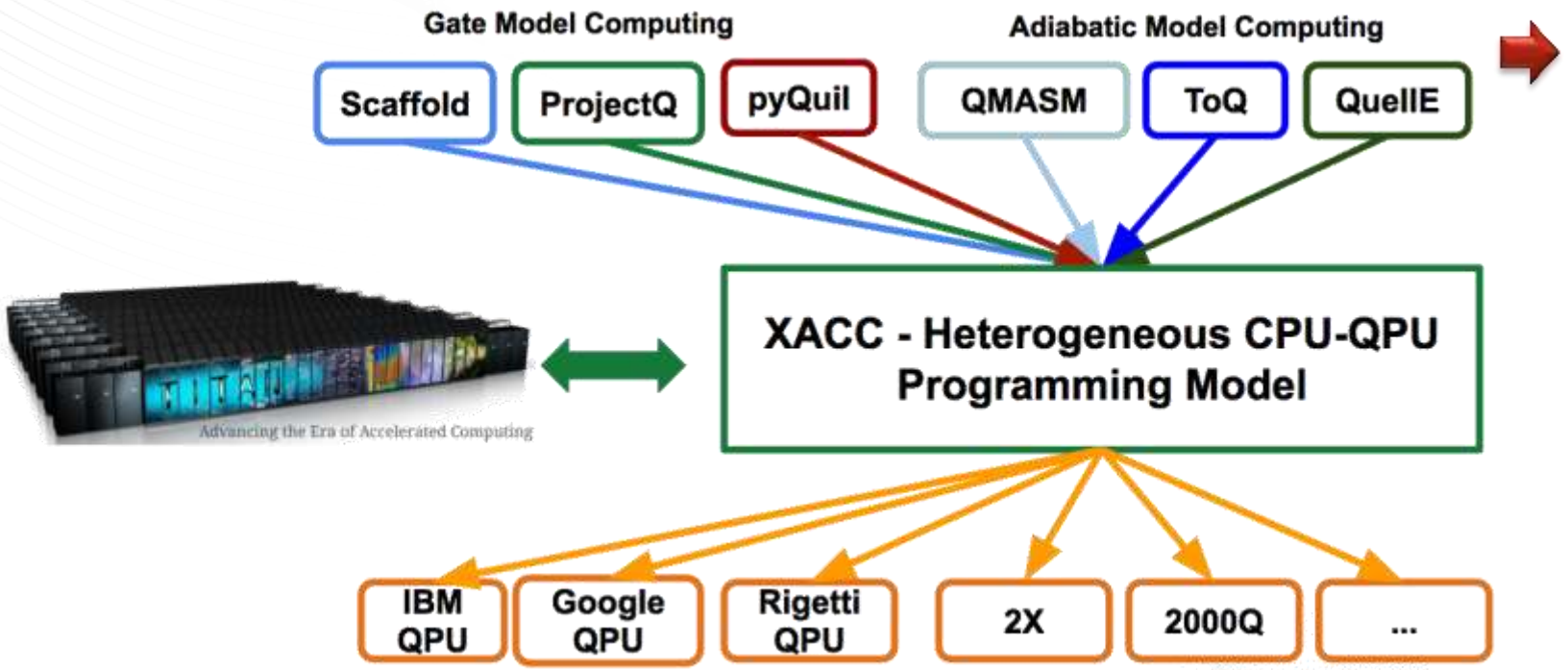
- **We are developing an OpenCL-like approach to QPU programming called XACC**
  - User picks the host language and defines a ‘kernel’ within a DSL tailored to the available QPU
  - Example: Host C/C++ program using Scaffold kernel to run on Rigetti QPU
- **XACC links and manages QPU resources**
  - Programming directives to manage QPU usage
  - Compilation mechanisms to support device-specific concerns, based off llvm
- **There are several key benefits to the user**
  - Maintain existing application codes
  - Employ host language and tools, e.g., C, Fortran
  - Easily switch between accelerator languages, SDKs

Language Hierarchy

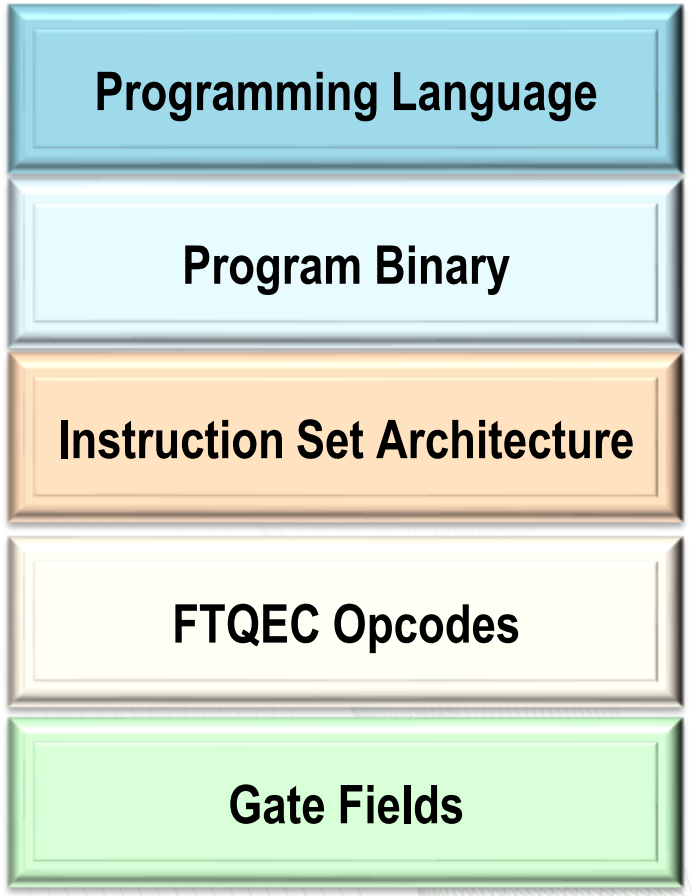


Programming Quantum Accelerators

- XACC: https://github.com/ornl-qci/xacc



Language Hierarchy



VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

Host application, C/C++ code

```
15  int main(int argc, char** argv) {
16
17          // All our important stuff is in the xacc::vqe namespace
18          using namespace xacc::vqe;
19
20          // Set the default Accelerator to TNQVM, and
21          // default number of electrons to 2
22          xacc::setAccelerator("tnqvm");
23          xacc::setOption("n-electrons", "2");
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

Host application, C/C++ code

```
15  int main(int argc, char** argv) {
16
17          // All our important stuff is in the xacc::vqe namespace
18          using namespace xacc::vqe;
19
20
21
22
23
```

```
96                  std::ifstream moleculeKernelHpp(xacc::getOption("vqe-kernel-file"));
97                  VQEProblem problem(moleculeKernelHpp);
98
99                  auto params = problem.initializeParameters();
100                 cppoptlib::NelderMeadSolver<VQEProblem> solver;
101                 solver.setStopCriteria(VQEProblem::getConvergenceCriteria());
102                 solver.minimize(problem, params);
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

Host application, C/C++ code

```
15  int main(int argc, char** argv) {
16
17          // All our important stuff is in the xacc::vqe namespace
18          using namespace xacc::vqe;
19
20
21
22
23
```

```
96                  std::ifstream moleculeKernelHpp(xacc::getOption("vqe-kernel-file"));
97                  VQEProblem problem(moleculeKernelHpp);
98
99                  auto params = problem.initializeParameters();
100                 cppoptlib::NelderMeadSolver<VQEProblem> solver;
101                 solver.setStopCriteria(VQEProblem::getConvergenceCriteria());
102                 solver.minimize(problem, params);
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

VQEProblem Class, C/C++ code

```
94          VQEProblem(std::istream& moleculeKernel) : nParameters(0), currentEnergy(0.0) {
95                  // This class only takes kernels
96                  // represented as Fermion Kernels.
97                  xacc::setCompiler("fermion");
98
99                  // Create the Accelerator. This will be TNQVM
100                 // if --accelerator not passed to this executable.
101                 qpu = xacc::getAccelerator();
102
103                 // Create the Program
104                 Program program(qpu, moleculeKernel);
105
106                 // Start compilation
107                 program.build();
108
109                 // Create a buffer of qubits
110                 nQubits = std::stoi(xacc::getOption("n-qubits"));
111
112                 // Get the Kernels that were created
113                 kernels = program.getRuntimeKernels();
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

VQEProblem Class, C/C++ code

```
94          VQEProblem(std::istream& moleculeKernel) : nParameters(0), currentEnergy(0.0) {
95                  // This class only takes kernels
96                  // represented as Fermion Kernels.
97                  xacc::setCompiler("fermion");
98
99                  // Create the Accelerator. This will be TNQVM
100                 // if --accelerator not passed to this executable.
101                 qpu = xacc::getAccelerator();
102
103                 // Create the Program
104                 Program program(qpu, moleculeKernel);
105
106                 // Start compilation
107                 program.build();
108
109                 // Create a buffer of qubits
110                 nQubits = std::stoi(xacc::getOption("n-qubits"));
111
112                 // Get the Kernels that were created
113                 kernels = program.getRuntimeKernels();
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

VQEProblem Class, C/C++ code

```
94          VQEProblem(std::istream& moleculeKernel) : nParameters(0), currentEnergy(0.0) {
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
17  113
```

```
160     #pragma omp parallel for reduction (+:sum)
161                     for (int i = 0; i < kernels.size(); i++) {
162
163                             // Get the ith Kernel
164                             auto kernel = kernels[i];
165
166                             // Insert the state preparation circuit IR
167                             // at location 0 in this Kernels IR instructions.
168                             kernel.getIRFunction()->insertInstruction(0, evaluatedStatePrep);
169
170                             // Create a temporary buffer of qubits
171                             auto buff = qpu->createBuffer("qreg", nQubits);
172
173                             // Execute the kernel!
174                             kernel(buff);
```

VQE example code using XACC

https://github.com/ORNL-QCI/xacc-vqe

VQEProblem Class, C/C++ code

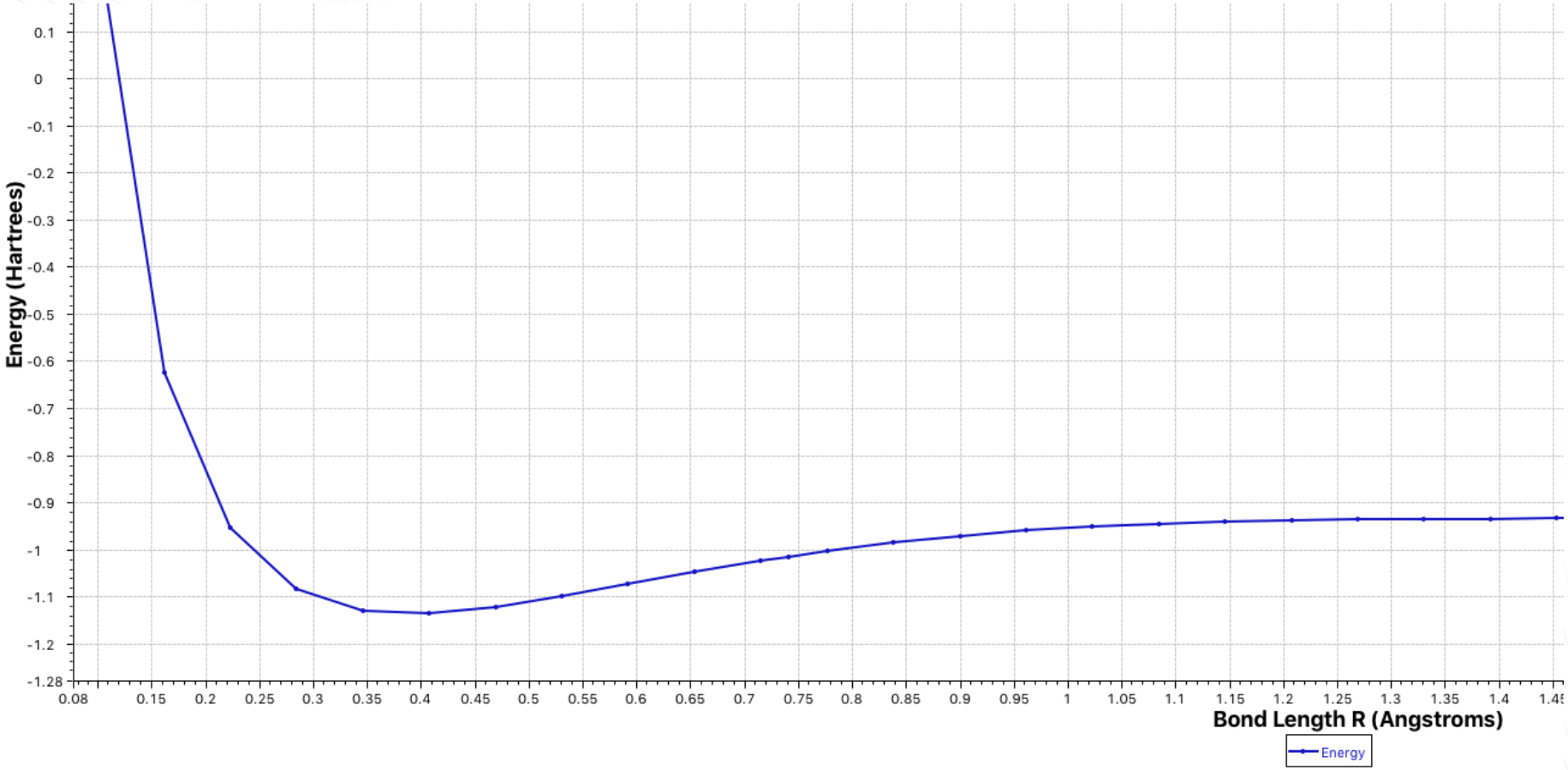
```
94          VQEProblem(std::istream& moleculeKernel) : nParameters(0), currentEnergy(0.0) {
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
```

```
160     #pragma omp parallel for reduction (+:sum)
161                     for (int i = 0; i < kernels.size(); i++) {
162
163                             // Get the ith Kernel
164                             auto kernel = kernels[i];
165
166                             // Insert the state preparation circuit IR
167                             // at location 0 in this Kernels IR instructions.
168                             kernel.getIRFunction()->insertInstruction(0, evaluatedStatePrep);
169
170                             // Create a temporary buffer of qubits
171                             auto buff = qpu->createBuffer("qreg", nQubits);
172
173                             // Execute the kernel!
174                             kernel(buff);
```

VQE example code using XACC

https://github.com/ORNL-QCI/tnqvm

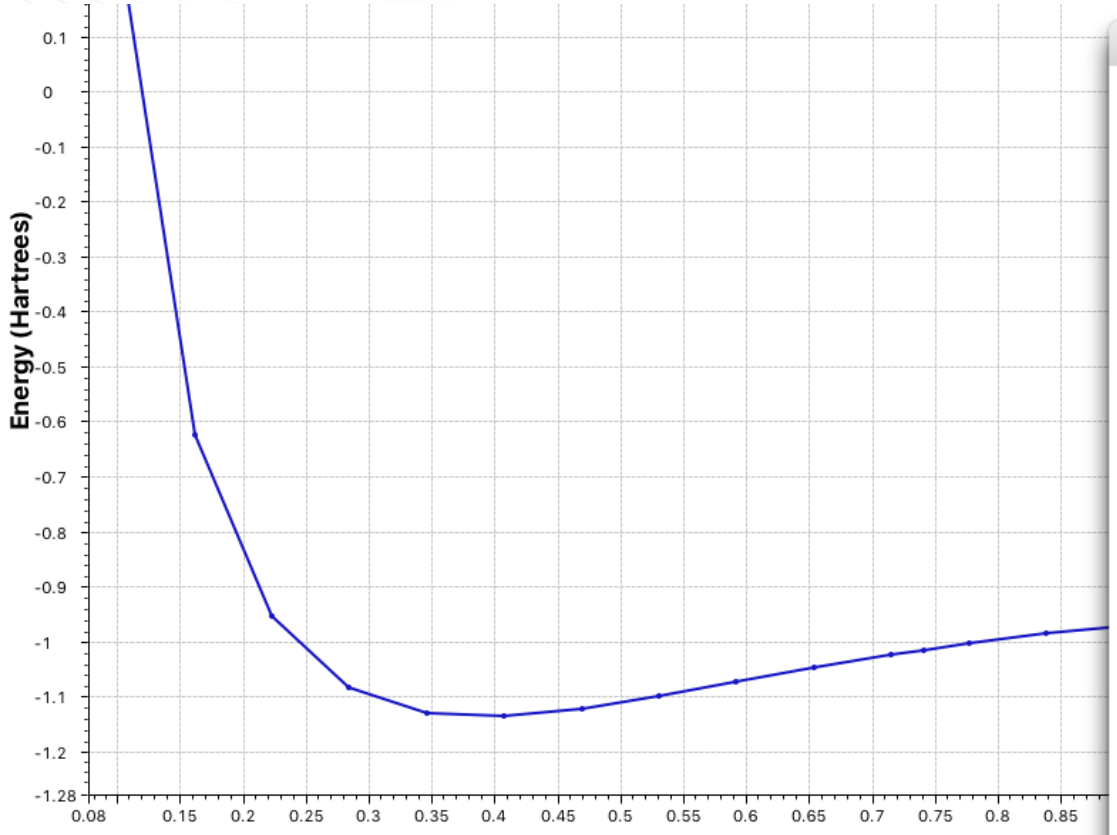
TNQVM accelerator, Tensor network (MPS) numerical simulator

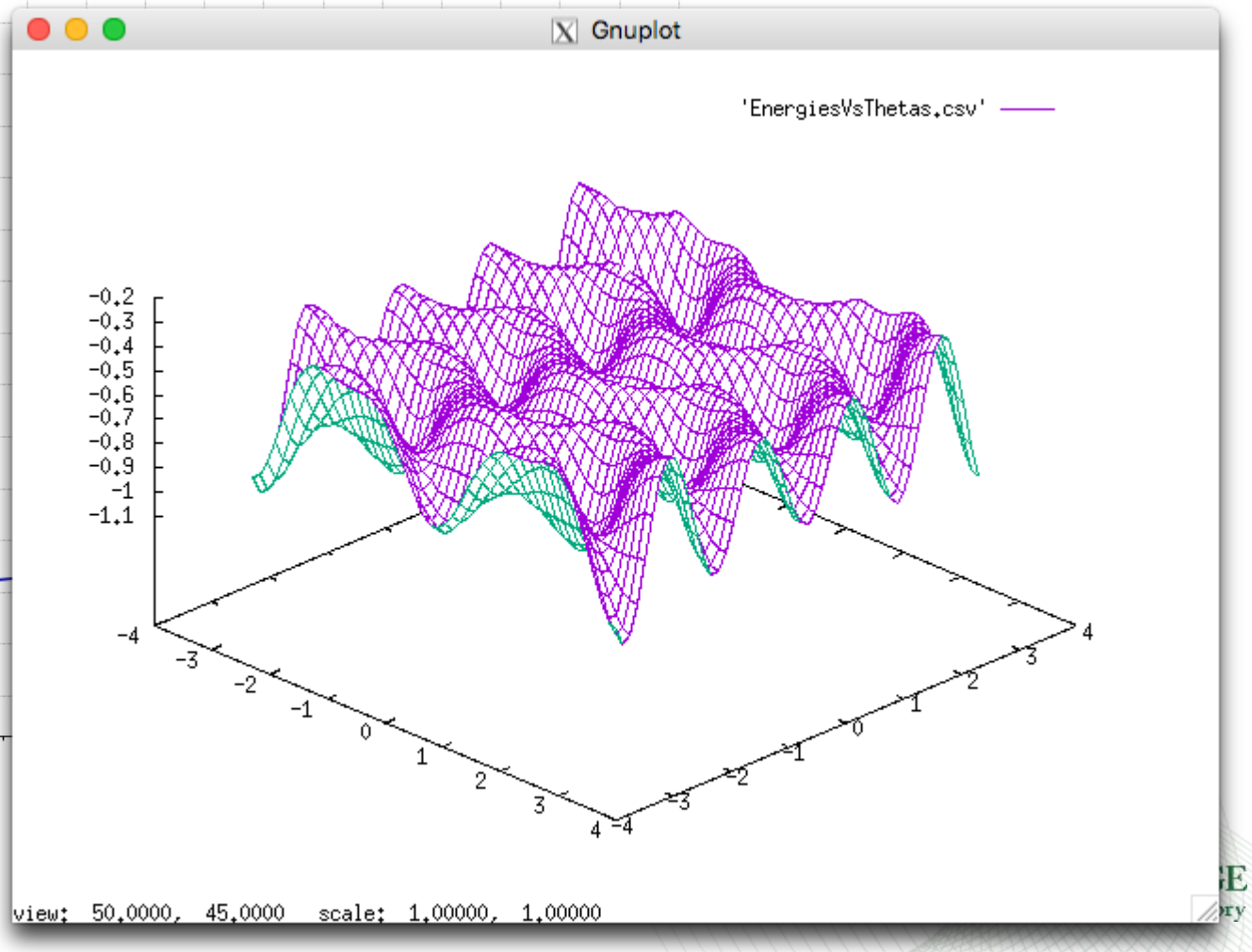


VQE example code using XACC

https://github.com/ORNL-QCI/tnqvm

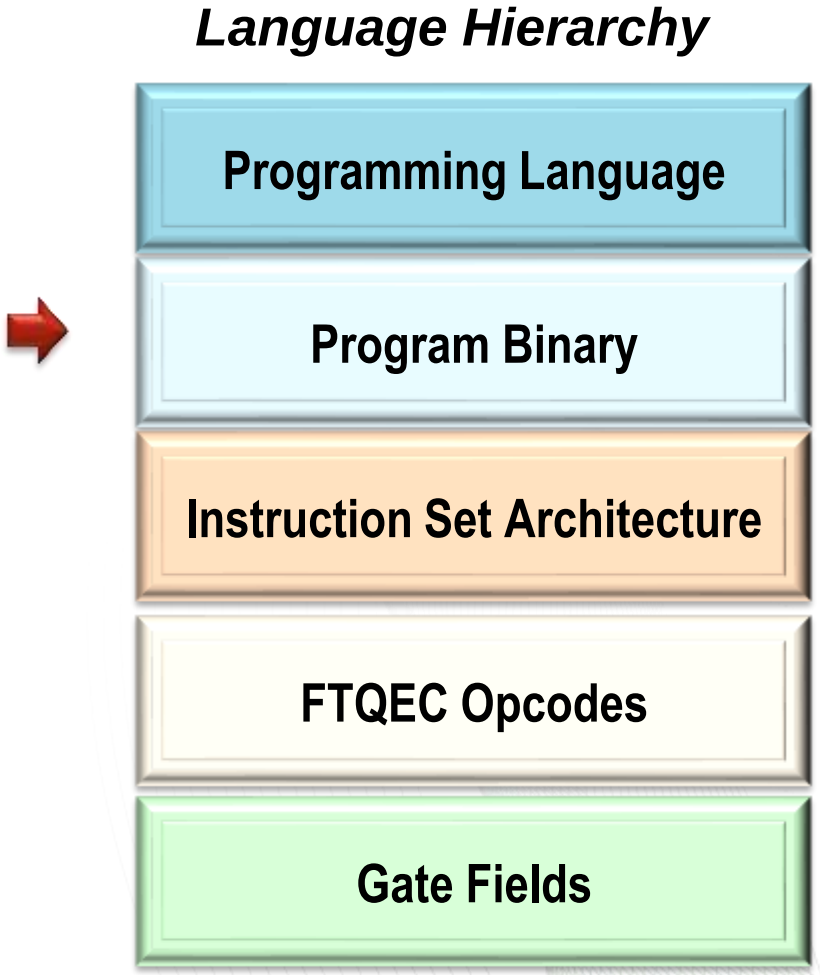
TNQVM accelerator, Tensor network (MPS) numerical simulator





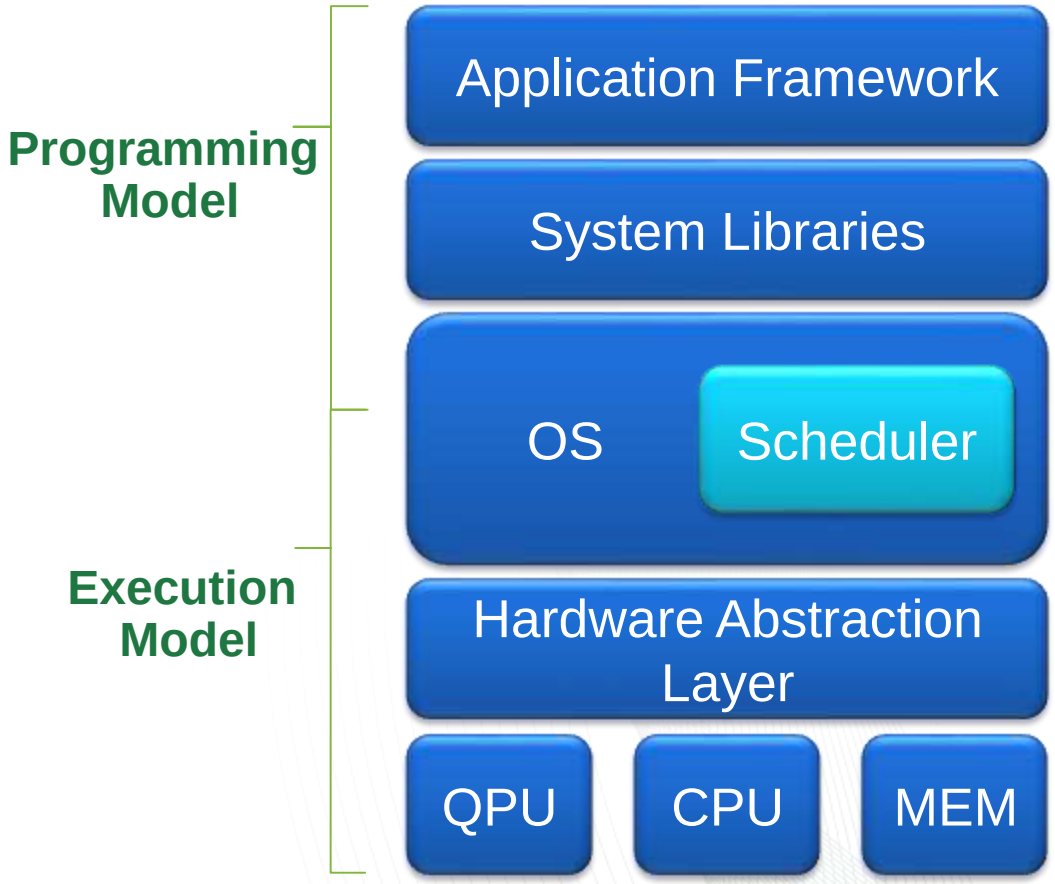
Executing the Compiled Program

- **In principle, programming models translate DSLs into executable instructions**
  - (All?) Existing QPL's create interpreted representations
  - Actual QPU scheduling based on interpreters
- **We are developing virtual machine representations for interpreters and numerical simulators**
  - Virtual machine paradigm uses hardware abstraction layer to manage different QPU devices
    - Current API's for IBM, Rigetti, and D-Wave
  - VM also offers interaction with numerical simulator
    - Currently using quantum state simulation



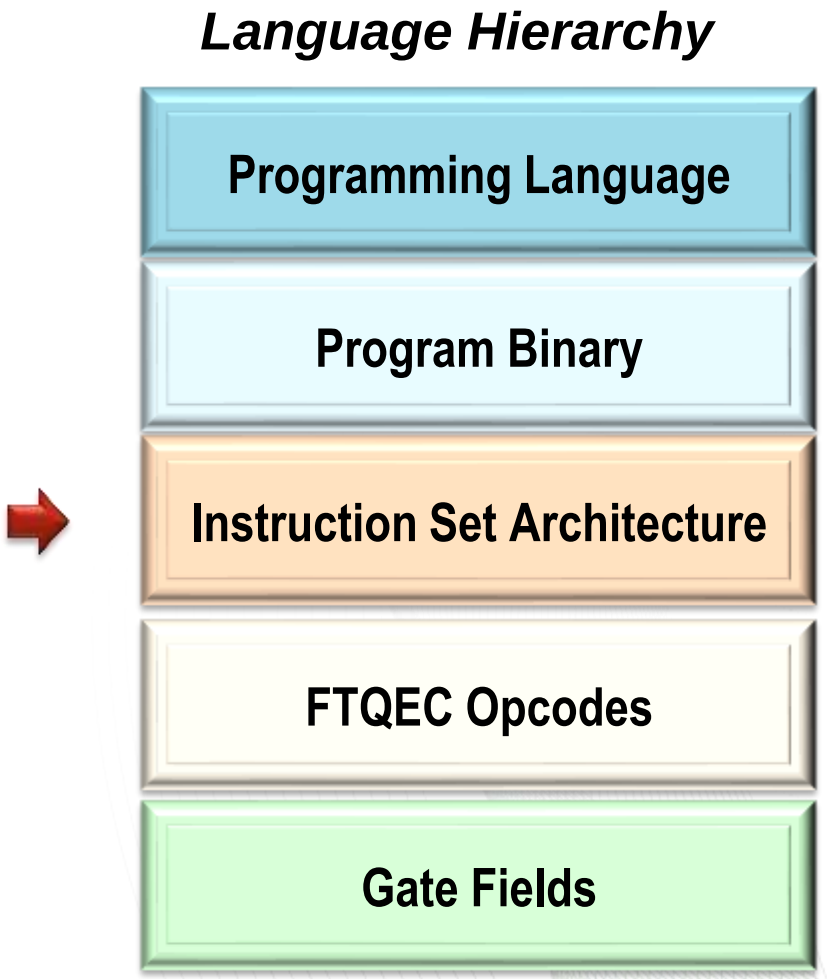
How does a host OS manage a QPU?

- **Current QPUs are very loosely integrated with the host system, e.g., client-server interactions**
  - This is driven by infrastructure constraints
- **The host run-time system must accommodate the QPU device and the programming model**
  - The run-time system is responsible for managing resources, errors, permissions
  - Instructions must be issued to the QCU via memory managed by the OS
  - Some program control statements require measurement feedback for evaluation
  - This evaluation may be caught closer to QEU given additional synchronization



The ISA provides an interface for the QPU

- **The logic supported by QPUs is under negotiation**
  - QASM is a popular pseudo-code, but it has lacked a complete definition for 20 years
  - Recent specifications try to fill this gap for gate model
- **RISC vs CISC ISA designs impact performance**
  - Example: how should we initialize the register?
  - Britt and Humble, “Instruction Set Architectures for Quantum Processing Units,” arXiv:1707.06202
- **We are developing software to analyze instruction and evaluate tradeoffs**
  - Parser, lexer, and listener for walking source files
  - We are adding technology constraints, e.g., register size, connectivity limitations



QPU Programming depends on device ISA

- IBM has released a written spec for their variant of QASM
  - https://github.com/IBM/qiskit-openqasm

Statement	Description
`OpenQASM 2.0;`	Denotes a file in Open QASM format[a]
`qreg name[size];`	Declare a named register of qubits
`creg name[size];`	Declare a named register of bits
`include "filename";`	Open and parse another source file
`gate name(params) qargs { body }`	Declare a unitary gate
`opaque name(params) qargs;`	Declare an opaque gate
`// comment text`	Comment a line of text
`U(theta,phi,lambda) qubit|qreg;`	Apply built-in single qubit gate(s)[b]
`CX qubit|qreg,qubit|qreg;`	Apply built-in CNOT gate(s)
`measure qubit|qreg -> bit|creg;`	Make measurement(s) in Z basis
`reset qubit|qreg;`	Prepare qubit(s) in $\|0\rangle$
`gatename(params) qargs;`	Apply a user-defined unitary gate
`if(creg==int) qop;`	Conditionally apply quantum operation
`barrier qargs;`	Prevent transformations across this source line

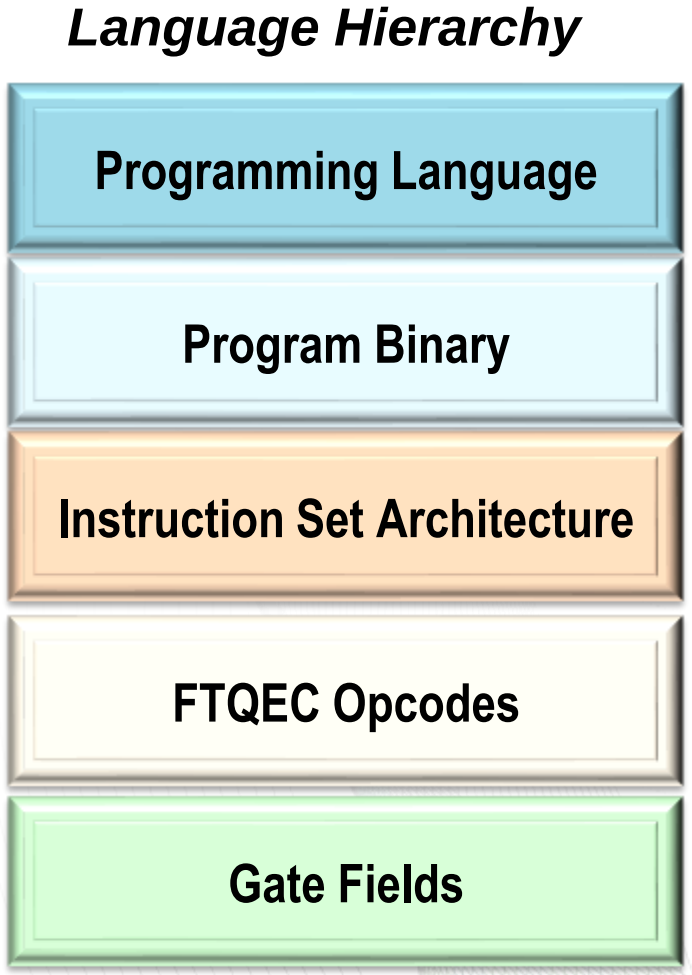
  - Not a complete language spec (embedded)
- Rigetti has a complete language specification
  - A Practical Quantum Instruction Set Architecture, arxiv:1608.03355

ANTRL4 grammar specification for Open QASM

```
/**
* This is the Open QASM Grammar Specification for ANTRL4
* We abbreviate it as OQASM
* Created by Travis Humble at Oak Ridge National Lab based on IBM Open QASM Specification
**/
grammar OQASM;

/**
* A program may conists of zero or more lines before end of file
**/
prog
                :       (line? EOL) ;

/**
* A line in the program may be several different things
**/
line
                : comment
                | instruction
                | assemblerinstruction
                | lbl
    ;

/**
* An instruction may have a label, but does have an opcode which may have an argumentlist
**/
instruction
                : label? opcode argumentlist? comment?
                ;
```

Instructions trigger machine opcodes

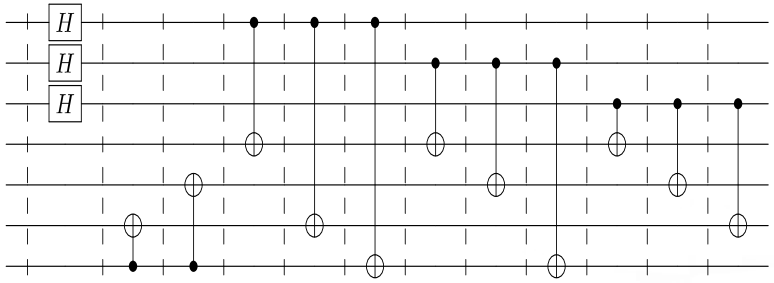
- **Opcodes trigger the execution units to apply fields**
  - These are dependent on microarchitecture, QEC specifications, and device parameters
  - The implementation is tied to how we use quantum execution units
- **Ensuring fault-tolerant operation requires additional gates and registers**
  - Quantum error correction codes redundantly encode state information
  - Syndrome measurements query if the state lies outside the codespace
  - Correction operations return state to the correct codespace
  - QEU scope differs these concerns to device maker



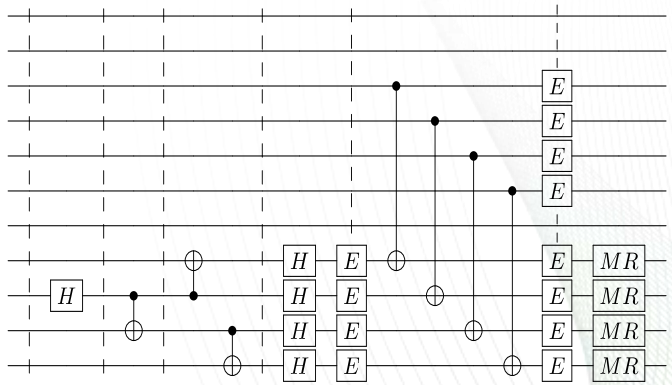
Implementing FTQEC operations

- **Opcode scheduling becomes dependent on both time and space**
  - FTQEC opcodes may account for real-time feedback or track evolving error state
  - Tradeoff in QEC codes and physical noise models
- **We use numerical simulation to certify specifications of opcodes for block and surface codes**
  - QASM-based noisy circuit modeling with stabilizer-based numerical simulations
  - Pseudo-threshold calculations for FTQEC opcodes
  - Bennink et al., “Unbiased Simulation of Near-Clifford Quantum Circuits,” Phys. Rev. A **95**, 062337 (2017)
  - Path integral methods with O(n^3) memory requirment

Steane [7,1,3] encoding circuit

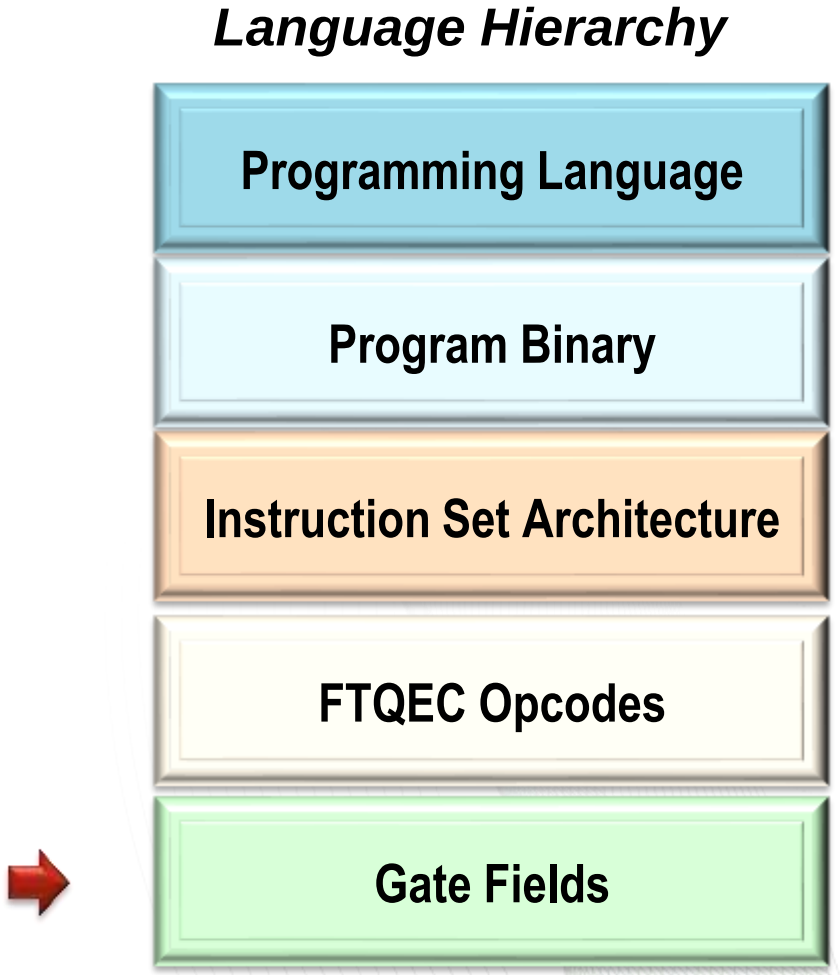


One syndrome measurement circuit



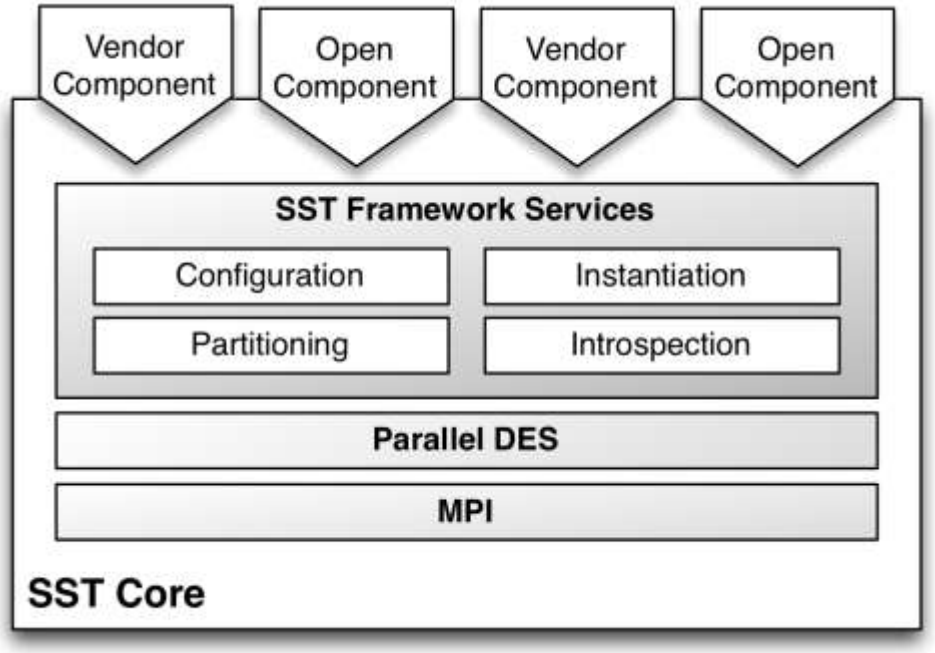
Gate fields define how opcodes are implemented

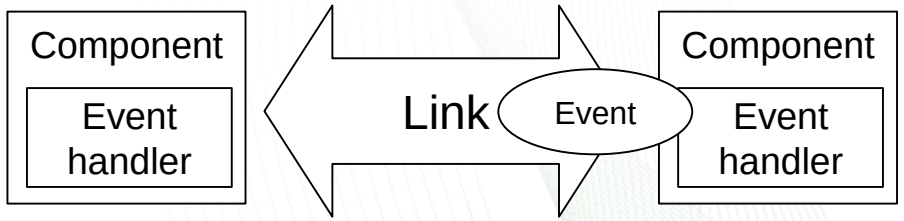
- **This is were the physics lives!**
  - Field specs are strongly dependent on technology and device design: solid state, atomic, photonic, etc.
  - Designed to address time-sensitive data registers
    - Interplay with decoherence, control, and QEC
    - Sets register lifetime and effective circuit depth
- **Gates are modeled as externally controlled Hamiltonians driving the register state**
  - Gate designs define expectations for behavior but gate operations must be validated
  - Actual behavior is characterized by experiment with support from simulation and heuristics
  - Humble et al.," A Computational Workflow for Designing Silicon Donor Qubits," Nanotechnology 27, 424002 (2016) (2016).



Modeling and Simulation of the Accelerator System

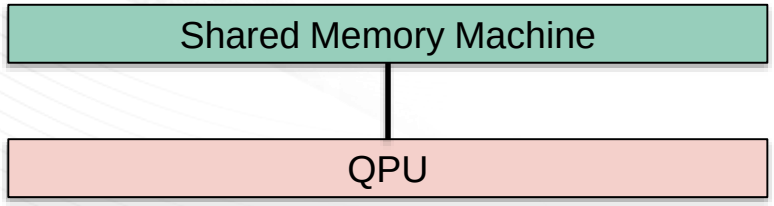
- **We model interactions between hardware components using these language interfaces**
  - We construct an executable model for the architecture and the component devices
  - The model is the input to a simulator that estimates system behaviors and metrics
- **We use the Structural Simulation Toolkit (SST) to model nodes, memory, network**
  - SST is a discrete-event simulations used to model conventional computing systems
  - It has existing models that account for data movement, latency, and power consumption
  - We use it to profile applications against architecture and device parameters



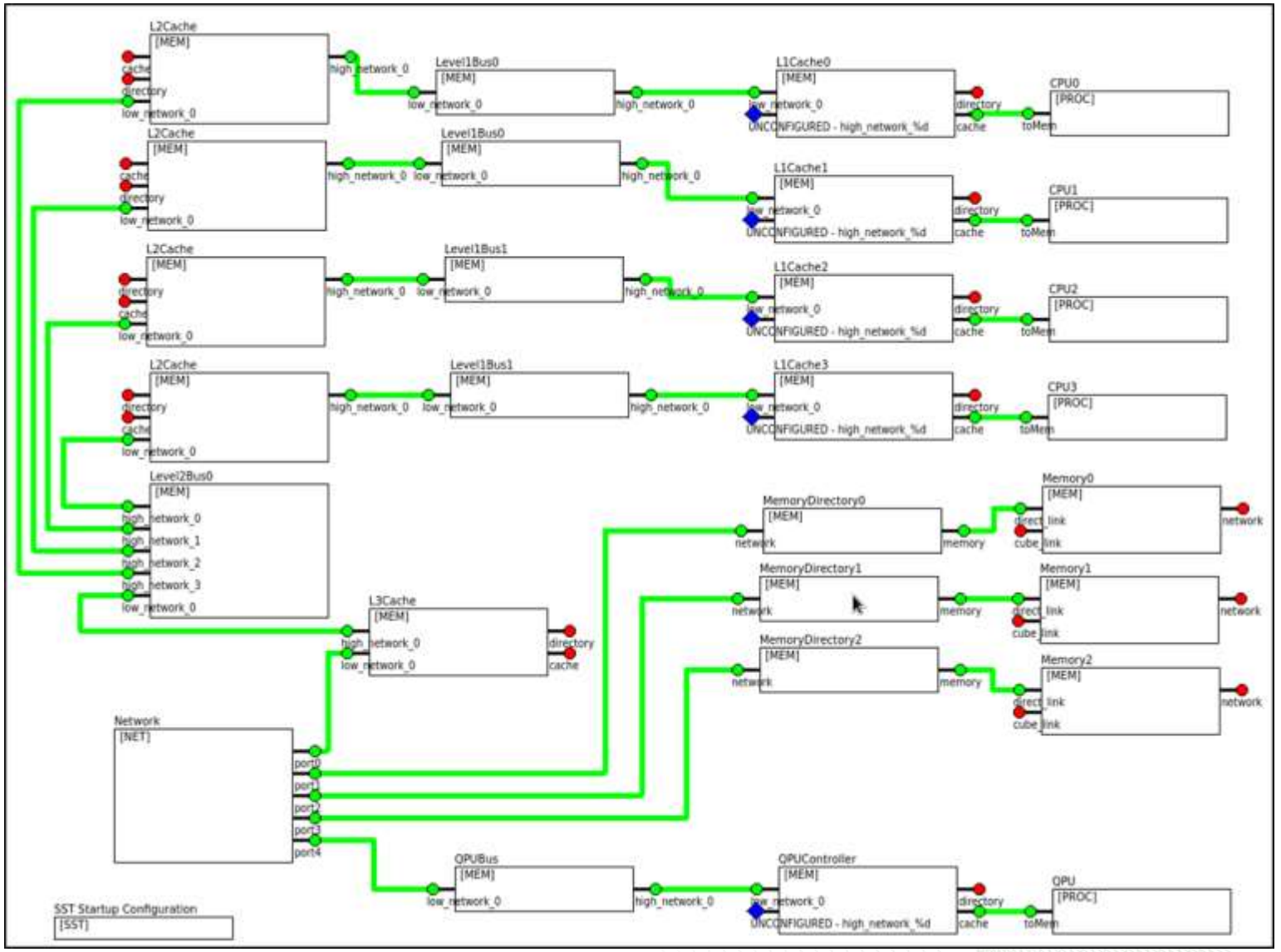


http://sst-simulator.org/

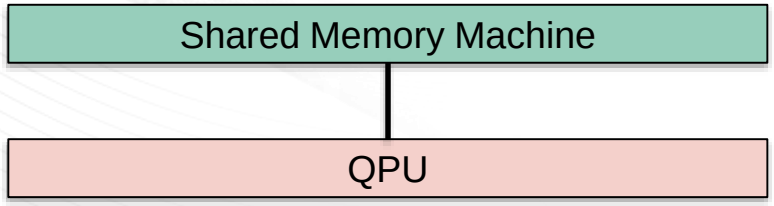
Capturing CPU-MEM-QPU Interactions



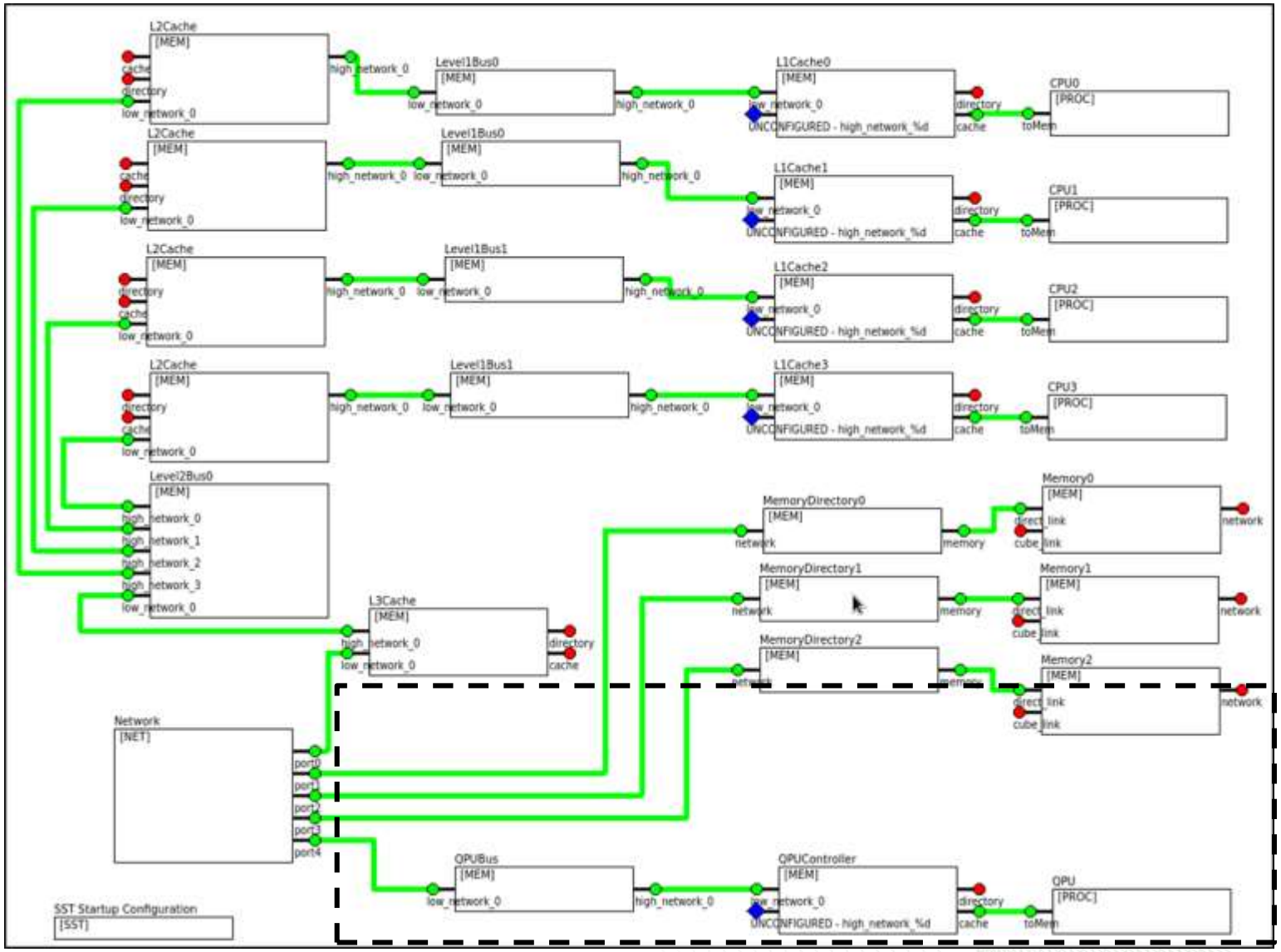
- A 4-core CPU connected to a QPU via a memory hierarchy



Capturing CPU-MEM-QPU Interactions

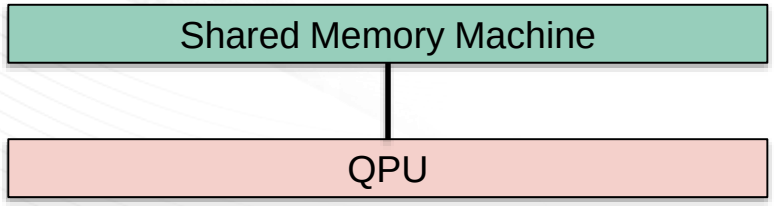


- A 4-core CPU connected to a QPU via a memory hierarchy

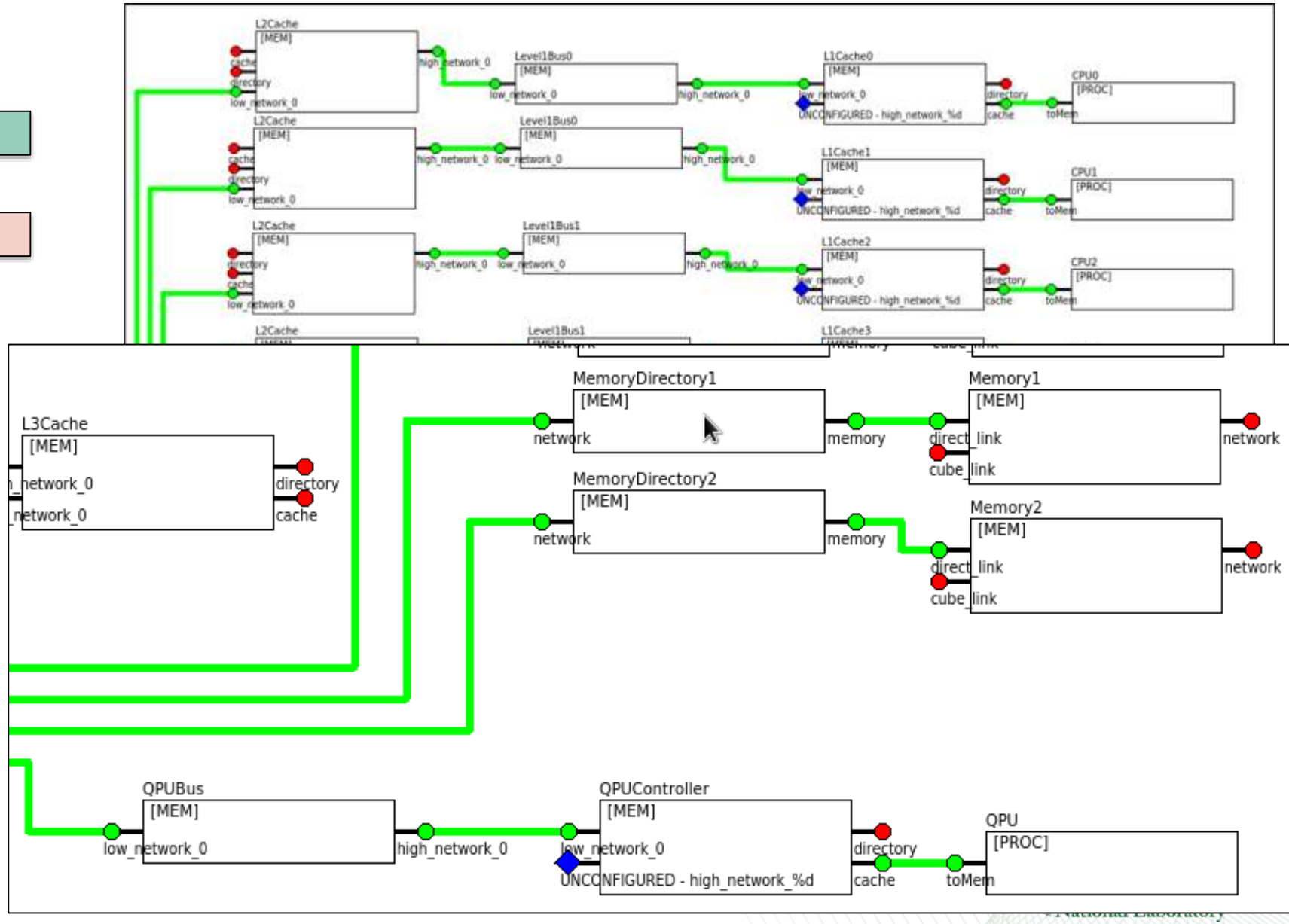


OAK RIDGE National Laboratory

Capturing CPU-MEM-QPU Interactions



- A 4-core CPU connected to a QPU via a memory hierarchy



Capturing CPU-MEM-QPU Interactions

- **CPU Model snippet**

```
//for(int s=0; s < msg->payload.size(); s++) {
for(int s=0; s < payload_size; s++) {
  // FIXME This should store elements from instruction.queue
  msg->payload.push_back(s);
};

// Calculate upcoming size for instruction queue.
int upcoming_size = current_size - payload_size;

// "If the current size is greater than count, the container is reduced to its fi
// FIXME Right now I don't care about which elements are removed, but later I wil
instruction_queue.resize(upcoming_size);

// Send message
remote_component->send(msg);

message_counter_sent++;

if(output_message_info) {
    std::cout << " CPU sent message: " << message_counter_sent << std::endl;
    std::cout << "     payload size: " << payload_size << std::endl;
    std::cout << " CPU queue size: " << instruction_queue.size() << std::endl;
}
```

- **QPU Model snippet**

```
// work starts at 0
int work_done = 0;

// completed instruction counter starts at 0
int instruction_counter = 0;

// Step through queue, accumulating work time until max is reached
//for(int s=0; s < instruction_queue.size(); s++) {
while((work_done < max_work_per_tick) && (instruction_queue.size() != 0)) {

  // Grab the first available instruction code at position s
  int an_instruction = instruction_queue[0];
  instruction_queue.pop_front();

  // Grab time for this instruction from lookup table
  // FIXME We don't check if an_instruction is valid, probably should do that
  time_per_instruction = instruction_times[an_instruction];

  // FIXME Is work the same as time? No, other Barry, it isn't...
  int work_for_instruction = time_per_instruction;

  // Add to total work time
  work_done += work_for_instruction;
```

Our current model captures CPU-QPU interactions

- **Python script executes model within SST framework**

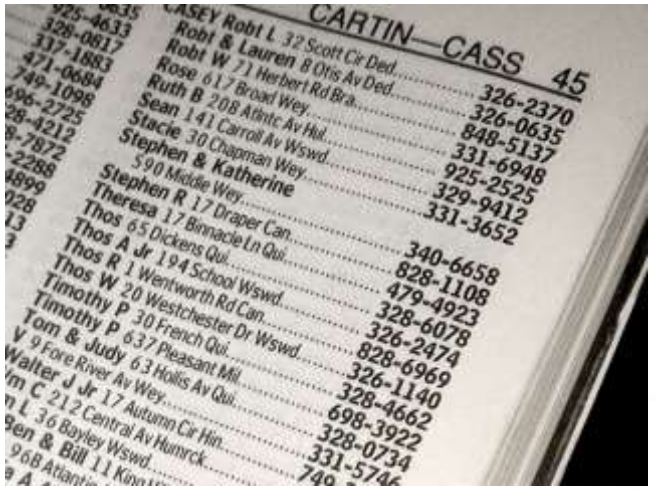
```
# QPU-CPU messaging demo
import sst

# Define SST core options
sst.setProgramOption("timebase", "1 ps")
sst.setProgramOption("stopAtCycle", "1 ms")

# Set output for statistics
sst.setStatisticOutput("sst.statOutputConsole")

# Set load level for desired statistics
# Load level defined in elementInfoStatistics entry
# Any statistics at this level or lower is collected
sst.setStatisticLoadLevel(7)

###################################
# Define the simulation components
###################################

# Define QPU component
comp_QPU = sst.Component("QPU", "quantum.qpuMessageGeneratorComponent")

# Set parameters for QPU component
comp_QPU.addParams({
# outputinfo = 1 means print recevied event message
"outputinfo" : """1""",
"sendcount" : """4""",
# clock controls the rate at which messages are issued
"clock" : """1MHz"""
})
```

- **Test-based output shows instruction work flows**

```
$ sst test_QpuSimpleMessageGeneratorComponent.py
Clock is configured for: 1MHz
Clock is configured for: 1MHz
CPU initial queue size = 53
****************
[QPU cycle: 1] (time=1us)
QPU queue size = 0
QPU instructions processed = 0
QPU queue size = 0
****************
[CPU cycle: 1] (time=1us)
CPU queue size = 53
CPU sent message: 1
payload size: 10
CPU queue size: 43
****************
[QPU event: 1] (time=1us)
payload size: 10
QPU queue size: 10
****************
[QPU cycle: 2] (time=2us)
QPU queue size = 10
(instr, work, sum) = (0,1,1)
(instr, work, sum) = (1,1,2)
(instr, work, sum) = (2,1,3)
(instr, work, sum) = (3,1,4)
```

Test Case: Energy Requirements for Unstructured Search

- **Problem: Find a specific item in an unstructured database**
  - The optimal classical algorithm to find a marked item requires *N/2* queries for an *N*-item database
  - Parallelizable across *K* system nodes with *K*-fold gather as the last step
- **Quantum search is a method for finding an item in an unstructured database**
  - First proposed by Grover (1996)
  - Sqrt(*N*) queries to find a single marked item
  - Sqrt(*N*/*M*) queries to find one of *M* marked items
  - Partial search: decompose *N* into *K* subsets, find the subset containing the marked item
- **What are the expectations for energy requirements?**

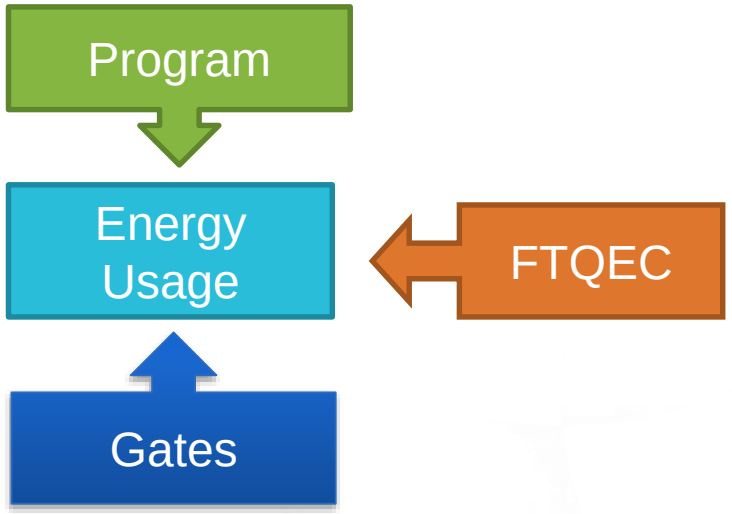


Example: Function inversion
$y = h(x) \Rightarrow h^{-1}(y) = x$

Bitcoin mining
Hash(x) = SHA256(SHA256(x))
x => 4 bytes = nonce

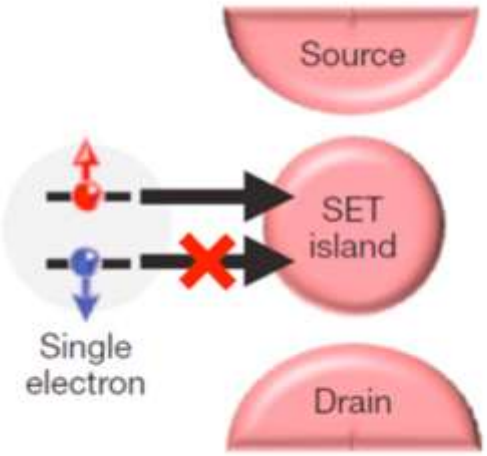
Defining the Energy Usage Metric

- **We add up the number of gates required to implement quantum search in silicon qubit technology.**
  - We use energy per gate based state of the art methods
  - We include simplified query implementation
  - We assume all transversal gates for FTQEC
  - We assume complete connectivity of qubits, no congestion
- **We test for a range of input sizes, large sizes**
  - n = 32, N = 4.9 x 10^9, possible Bitcoin nonces
  - n = 64, N = 1.8 x 10^{19}, modern CPU address space
  - n = 128, N = 3.4 x 10^{38}, number of IPv6 addresses



Estimating Energy Costs for a Silicon Quantum Computer

- **Flip-flop architecture for qubits in silicon**
  - Two-qubit gates induced through long-range resonators
  - Single-electron transistor (SET) is used to readout and initialize spin state
  - Tosi et al., arXiv:1509.08538

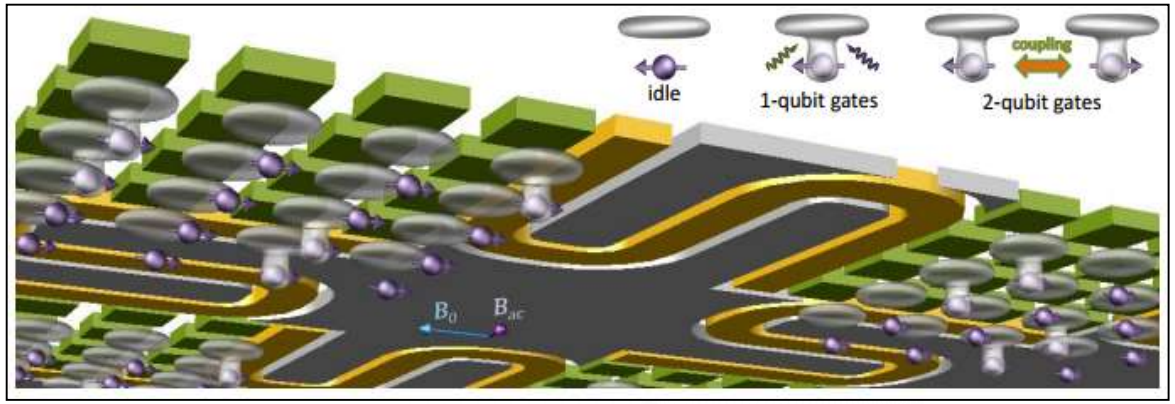


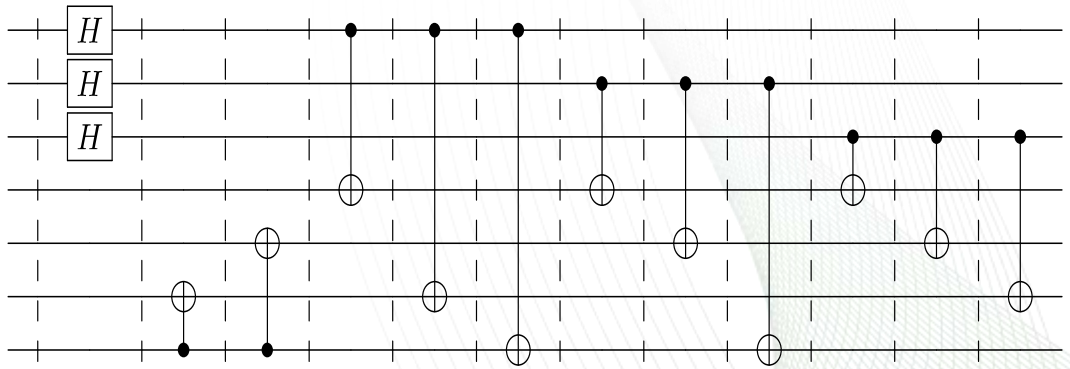
V_{SD} = 100 uV
I_{SET} = 1 nA

T_{read} = 100 us
E_{read} = 5 aJ

T_{init} = 300 us
E_{init} = 5 aJ

Qubit	Time	Power	Energy
Flip-flop	40 ns	0.1 pW	4 zJ



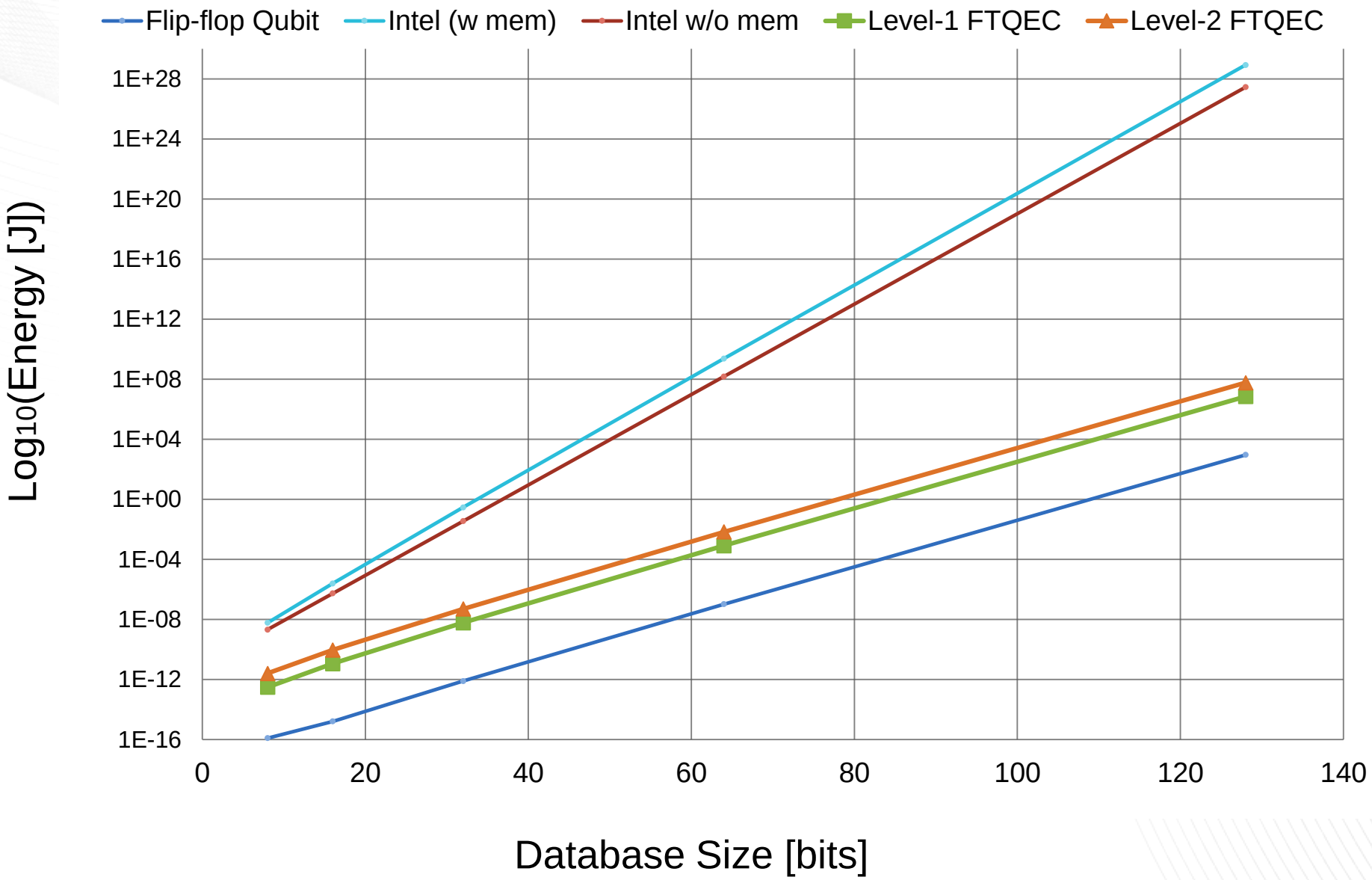


Conventional Computing Baseline

- **A serial search through a list**
  - We use brute force search program to compile into assembly instructions
  - 6 Instructions per iteration
  - *N*/2 iterations, average-case
- **We estimate energy per instruction**
  - Intel Core i7-6700K
  - 2.73 pJ Energy Per Instruction (EPI)
    - 1.35V, 1.5 pF, 91W
  - 3.7pJ/bit for DRAM read (best)
    - Deng et al., ASPLOS 2011
  - log(*N*) bits per read

```
1   start:   cmp bx,[si]
2            je  found
3            add si,2
4            inc cx
5            dec dx
6            jnz start
```

CPU vs QPU Energy Estimates (Steane FTQEC)



Energy Dividends

- **Energy cost scales exponentially with input for all methods due to growth in queries**
  - Quantum appears feasible for all input sizes
  - Energy for conventional CPU is split across memory movement and comparator
  - Energy for QPU is dominated by logical Query and Diffusion stages
  - FTQEC, syndromes are main contributor
- **Working on system power usage**
  - Need gate parallelization, scheduling methods
  - ROM: Level-2 FTQEC, 64-bits, 20 days, *~3 nW*
  - *Need to include field generators, decoding costs, instruction movement, thermodynamics*

Level-2 FTQEC Energy Usage

n	Usage [J]	Dividend [J]
32	10^{-8}	**10^{-1}**
64	10^{-3}	**10^{9}**
128	10^{7}	**10^{28}**

In terms of gravitational energy:

10^{28} J

10^{7} J

Energy Dividends

- **Energy cost scales exponentially with input for all methods due to growth in queries**
  - Quantum appears feasible for all input sizes
  - Energy for conventional CPU is split across memory movement and comparator
  - Energy for QPU is dominated by logical Query and Diffusion stages
  - FTQEC, syndromes are main contributor
- **Working on system power usage**
  - Need gate parallelization, scheduling methods
  - ROM: Level-2 FTQEC, 64-bits, 20 days, *~3 nW*
  - *Need to include field generators, decoding costs, instruction movement, thermodynamics*

Level-2 FTQEC Energy Usage

n	Usage [J]	Dividend [J]
32	10^{-8}	**10^{-1}**
64	10^{-3}	**10^{9}**
128	10^{7}	**10^{28}**

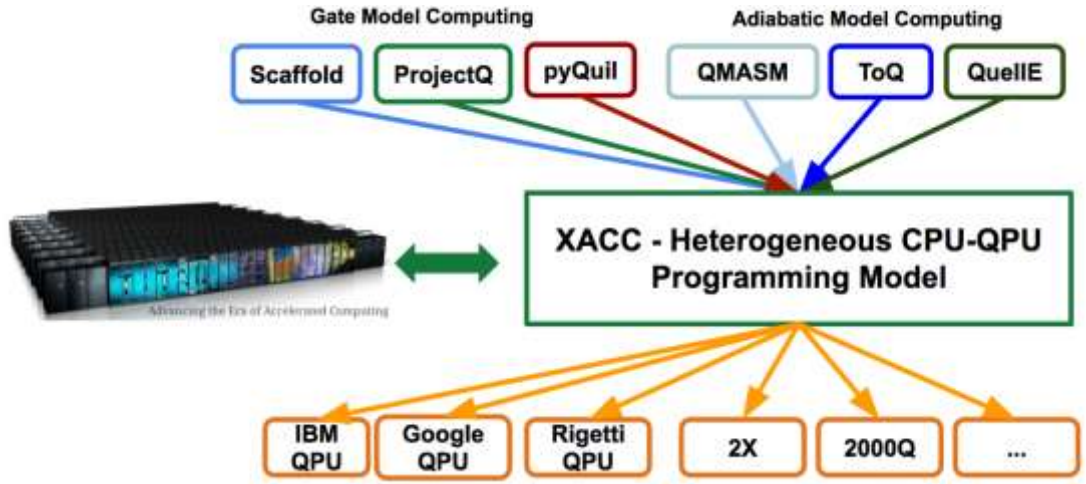
In terms of gravitational energy:

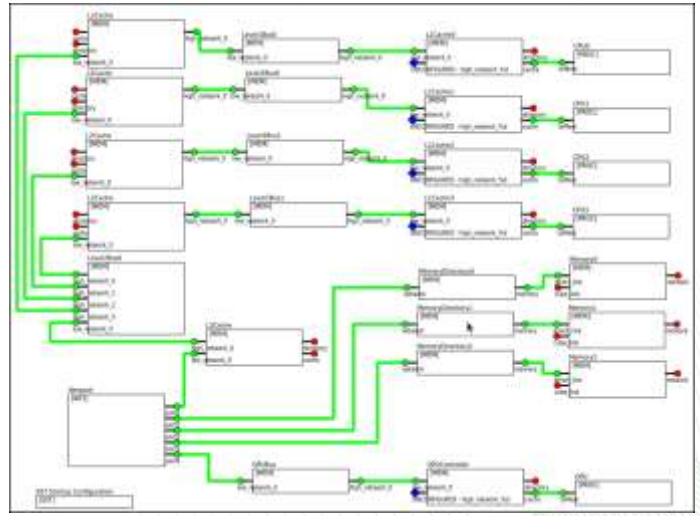
10^{28} J

10^{7} J

Summary

- We are developing system software to integrate QPU's with modern scientific workflows for HPC
- We are evaluating when QPU's can accelerate this work and when usage warrants integration
- The rise of commercial QPU's is likely to lead to many new ideas and applications
- Verifying the benefits of quantum computing is become increasingly necessary for science





The End