Exponential speedup in progress

Tomas Babej
ProteinQure Inc.

FOSDEM 19
Brussels, Belgium
Feb 02, 2019

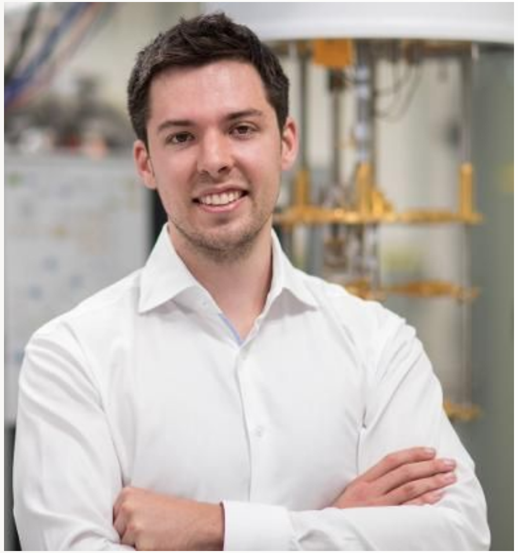
@tomasbabej

/tbabej

Quantum computing at FOSDEM

Tomas Babej

Mark Fingerhuth



Will Zeng

Why *quantum computing* and *open source*?

Quantum Open Source Foundation

-

QOSF

ProteinQure

So what awaits us at FOSDEM?

Saturday: Quantum computing devroom

Forest Beta

QuTiP

Quantum Toolbox in Python

STRAWBERRY FIELDS

D-Wave

The Quantum Computing Company™

PENNYLANE

Cirq

Sunday: Quantum computing workshop

- Overview of 11 open source quantum computing projects

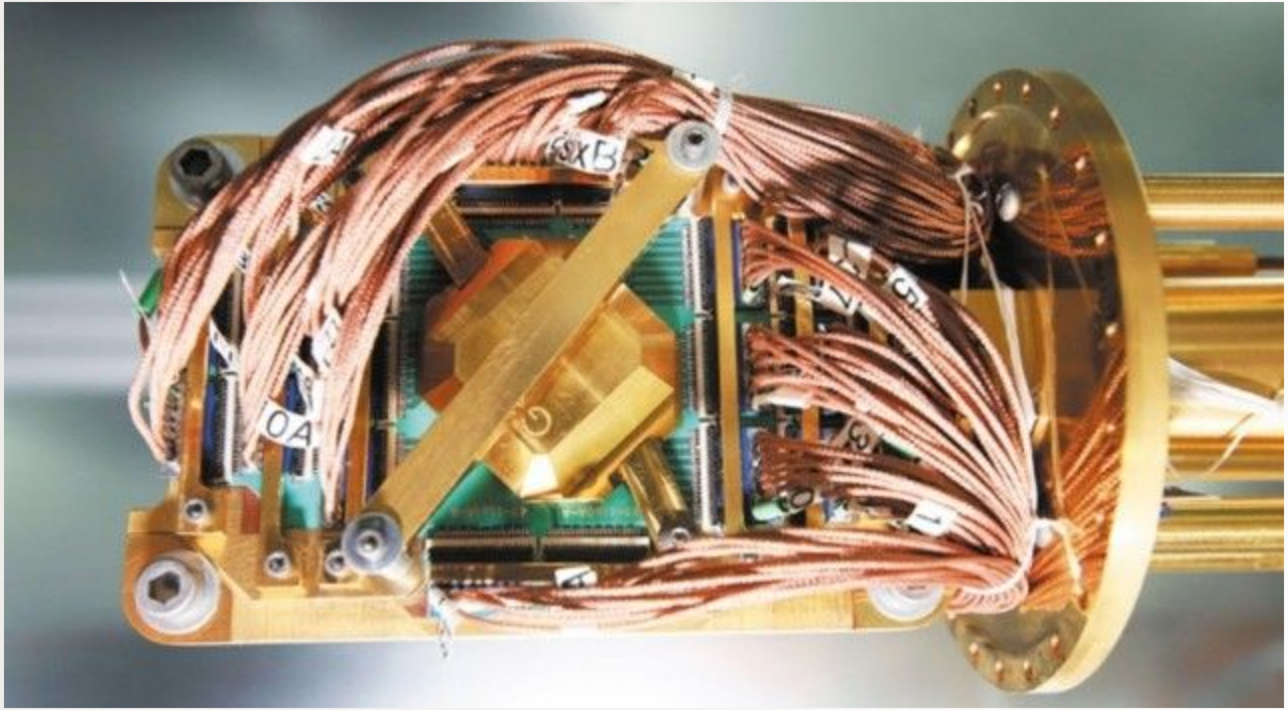
  NISQAI, Bayesforge, QCL, Curry, PyZX, RevKit, Q-bug, SimulaQron, QuantumInformation.jl and more!

- Dedicated hackathon sprint - learn, understand and contribute
- Coached by authors and developers

Sign up to reserve a spot: https://qosf.org/fosdem19-qc-workshop/

Why *invest* in learning QC **now**?

Quantum chips are here to explore



Investment in QC is rising

- No longer a purely academic domain
- Regional race happening
  - China: $3 billion, EU $1.1 billion, USA $1.275 billion
  - Also very active: Canada, Israel, Australia
- Private VC funding of more than $700 million
  - D-Wave, Rigetti, Silicon Quantum Computing, Cambridge Quantum Computing, 1Qbit, IonQ and others

Dedicated startup incubator

Quantum Machine Learning Program

Mission

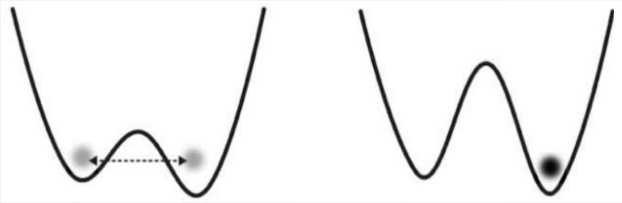
By 2022 the QML Program will have produced more well- capitalized, revenue generating quantum machine learning software companies than the rest of the world combined. The majority of these will be based in Canada.

CREATIVE DESTRUCTION LAB

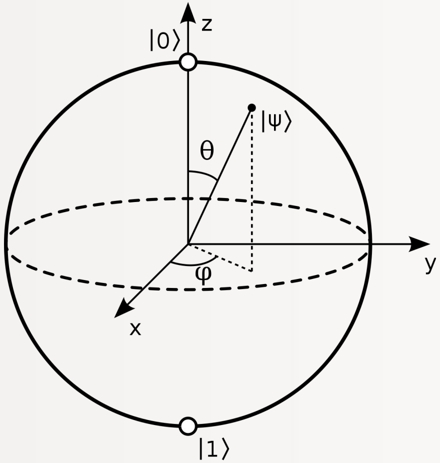
Applications are open now!

What *is* **quantum** computing?

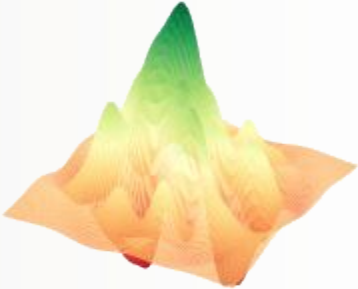
Paradigms of Quantum Computing



Quantum annealing

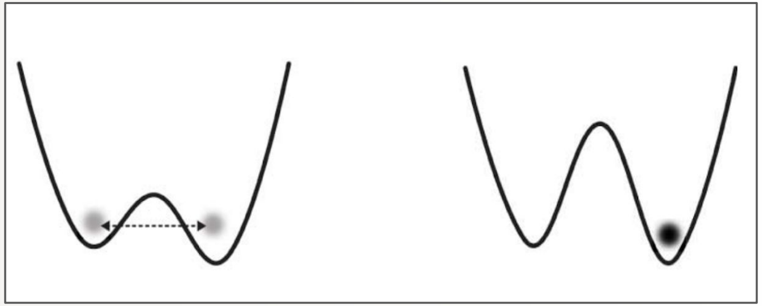


Discrete gate-based



Continuous gate-based

Paradigm: Quantum annealing



Quantum Annealing, D-Wave (2012)
Tunneling helps find optimal solutions

Big players in quantum annealing

D:WAVE

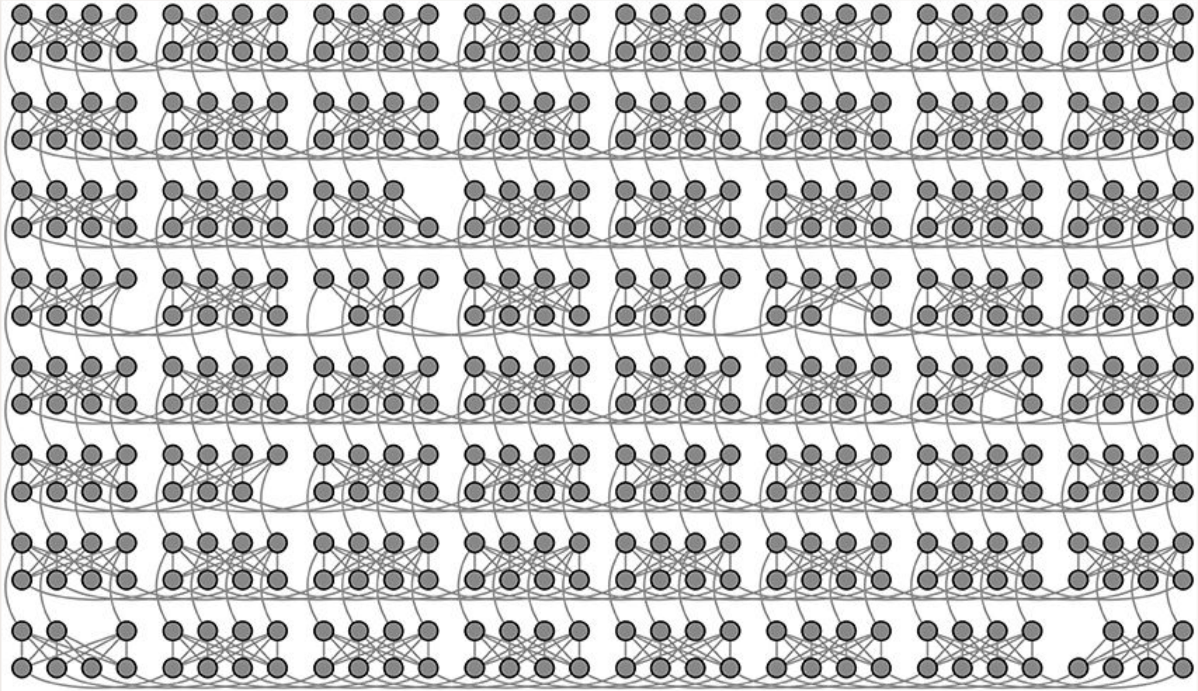
The Quantum Computing Company™

Crafting the energy landscape

$$H_0 = -\sum_{i,j} J_{i,j}\sigma_i^z\sigma_j^z - \sum_i h_i\sigma_i^z$$

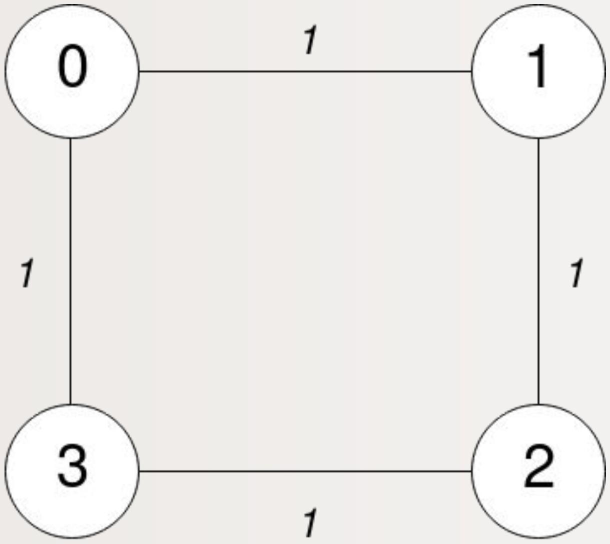
$$H(t) = (1-t)H_{\text{initial}} + tH_{\text{final}}$$

Paradigm: Quantum annealing



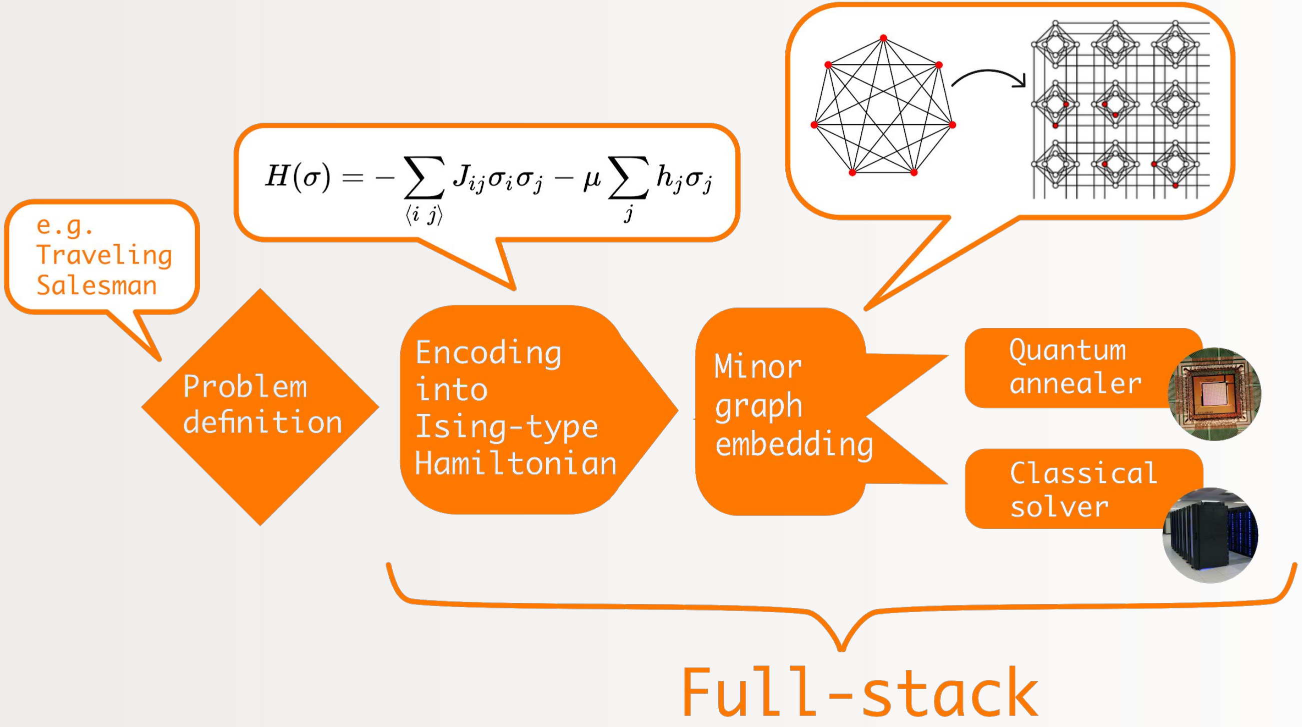
- Binary variables
- Finite resolution couplings and biases
- Classical readout

Example problem: Checkerboard

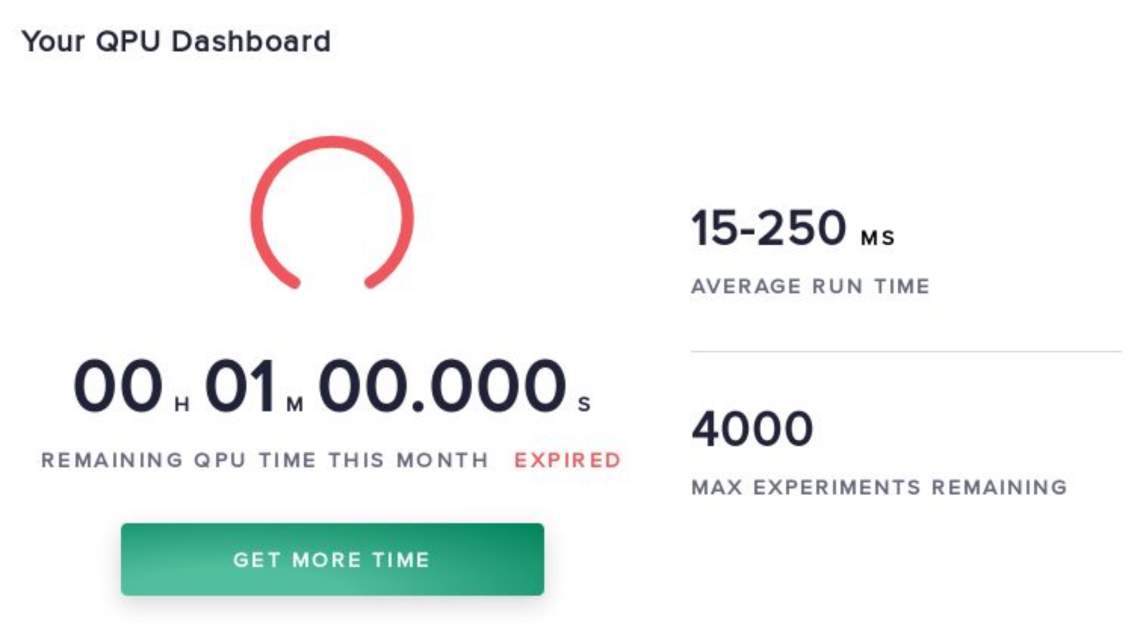



```
import dimod
from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite

h = {}
J = {(0, 1): 1, (1, 2): 1, (2, 3): 1, (3, 0): 1}
bqm = dimod.BinaryQuadraticModel.from_ising(h, J)
sampler = EmbeddingComposite(DWaveSampler())
result = sampler.sample(bqm)

print(result.first)

>>> Sample(sample={0: -1, 1: 1, 2: -1, 3: 1}, ...)
```

Quantum annealing workflow



D-Wave Leap

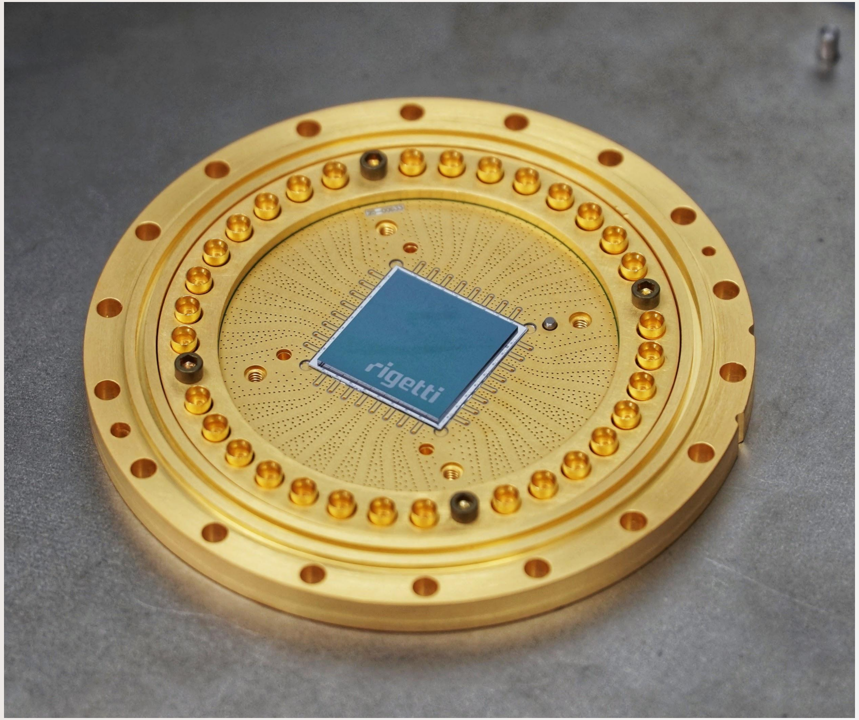


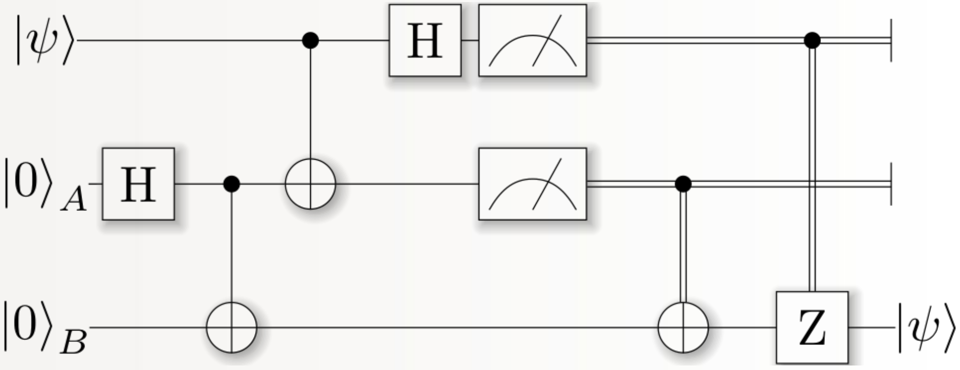
Your QPU Dashboard
00 H 01 M 00.000 S
REMAINING QPU TIME THIS MONTH EXPIRED
GET MORE TIME
15-250 MS
AVERAGE RUN TIME
4000
MAX EXPERIMENTS REMAINING

D-Wave further resources

- Leap portal: https://cloud.dwavesys.com/leap/
- Getting started with Ocean SDK: https://docs.ocean.dwavesys.com/en/latest/overview/install.html
- Quantum annealing Youtube series: https://www.youtube.com/watch?v=zvfkXjzzYOo

Paradigm: Universal gate-based QC





Quantum circuit for quantum teleportation

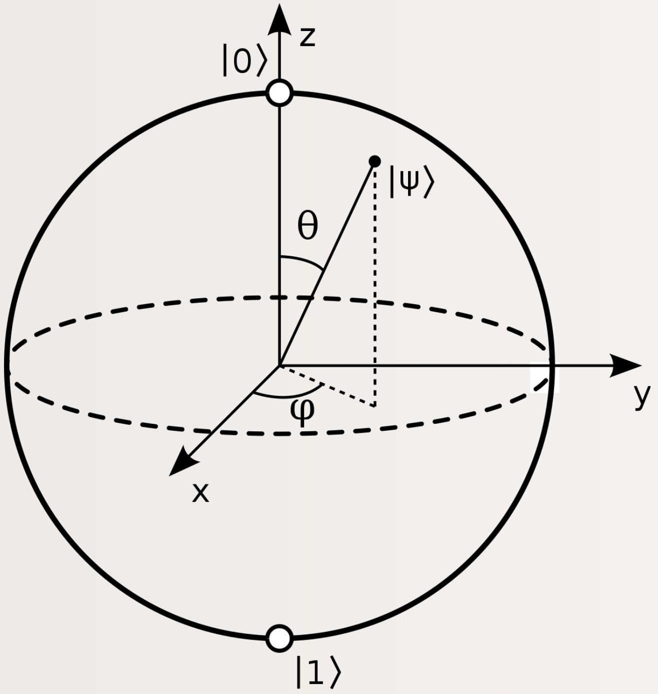
Big players in universal QC

Qiskit

Cirq

Forest

Paradigm: Universal gate-based QC



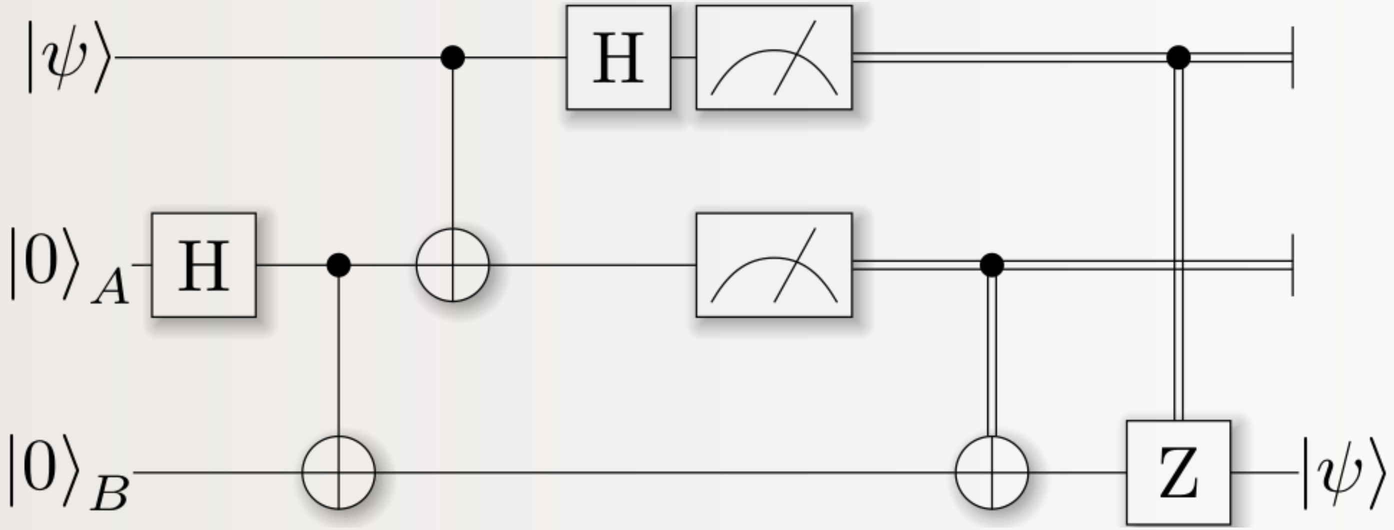
$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

$$|\psi\rangle = \cos(\tfrac{\theta}{2})\,|0\rangle + e^{i\phi}\sin(\tfrac{\theta}{2})\,|1\rangle$$

where $0 \leq \theta \leq \pi$ and $0 \leq \phi < 2\pi$

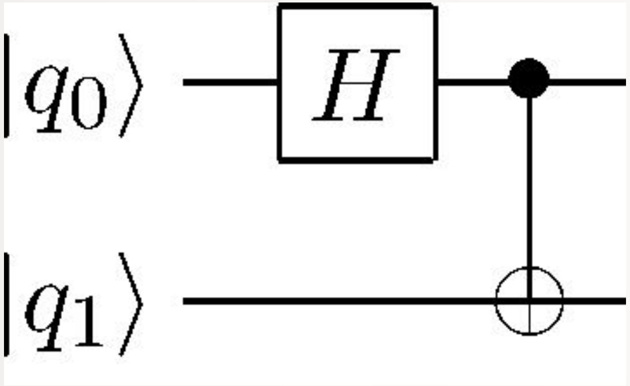
$$x = \sin(\theta)\cos(\phi)$$
$$y = \sin(\theta)\sin(\phi)$$
$$z = \cos(\theta)$$

Paradigm: Universal gate-based QC



- "Quantum" variables
- Gate operations
- Classical readout after measurement

Example: A pair of entangled qubits



Paradigm: Universal gate-based QC

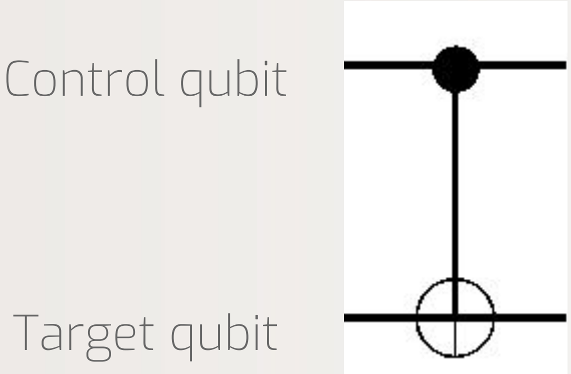
Hadamard gate (H)

$|0\rangle$ to $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$

```
>>> from pyquil.gates import H
```

Paradigm: Universal gate-based QC

Controlled-NOT (CNOT) gate

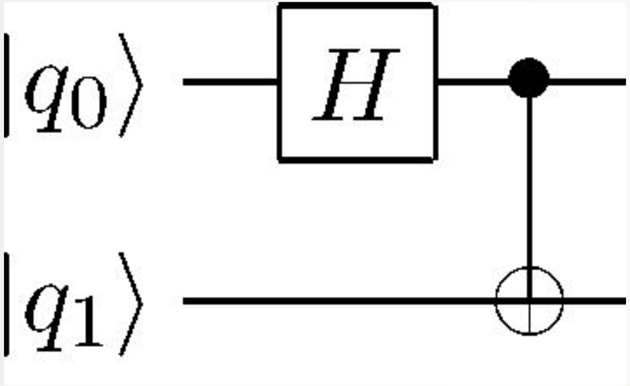



```
>>> from pyquil.gates import CNOT
```

Target qubit	Control qubit	Output
0	0	00
0	**1**	**11**
1	0	10
1	**1**	**01**

Essential for **entangling** two qubits!

Paradigm: Universal gate-based QC



$$\mathrm{CNOT}(0,1)(\mathbb{I}\otimes H)\,|00\rangle = \mathrm{CNOT}(0,1)\left[\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle\right]$$
$$= \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

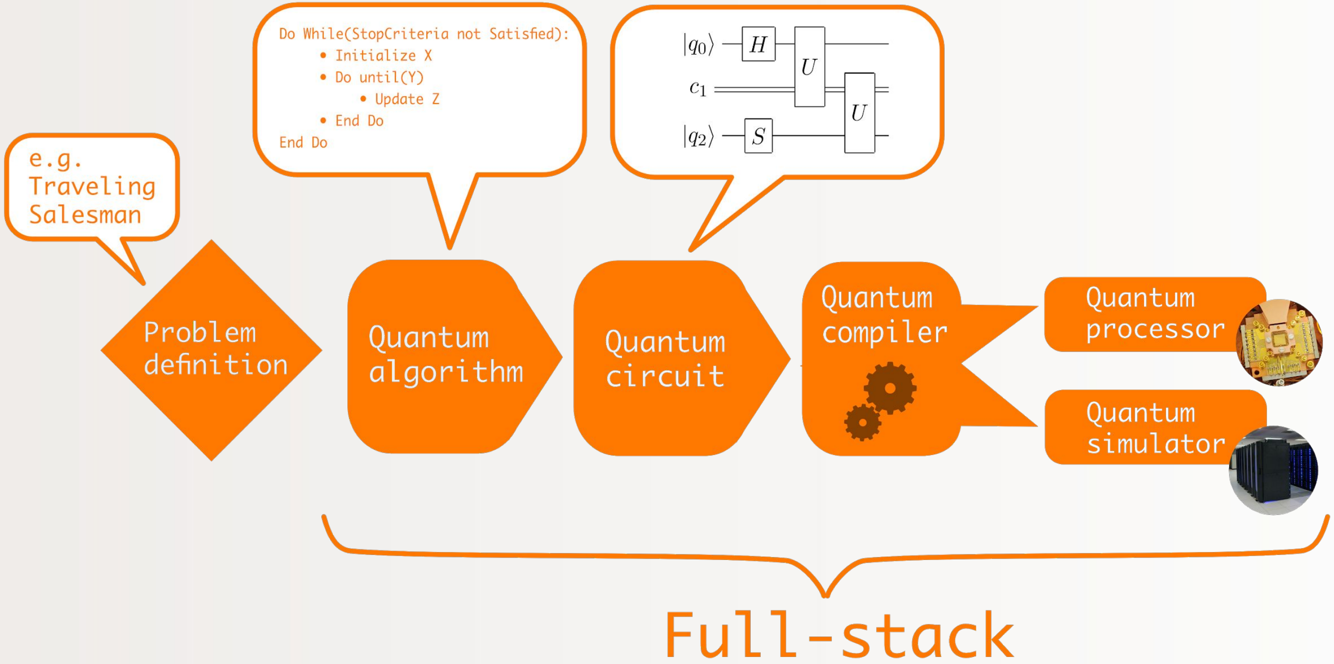
Applying the CNOT and H results in a maximally entangled state.
It's often called the **first Bell state.**

Paradigm: Universal gate-based QC

```python
from pyquil import Program
from pyquil.gates import H, CNOT, MEASURE
from pyquil import get_qc

program = Program(H(0), CNOT(0, 1))
classical_register = program.declare('ro', 'BIT', 2)
program += MEASURE(0, classical_register[0])
program += MEASURE(1, classical_register[1])
print(program)

qc = get_qc('2q-qvm')  # We need 2 qubits
executable = qc.compile(program)
result = qc.run(executable)
print(result)
```

```
>>>
H 0
CNOT 0 1
DECLARE ro BIT[2]
MEASURE 0 ro[0]
MEASURE 1 ro[1]

[[1 1]]
```

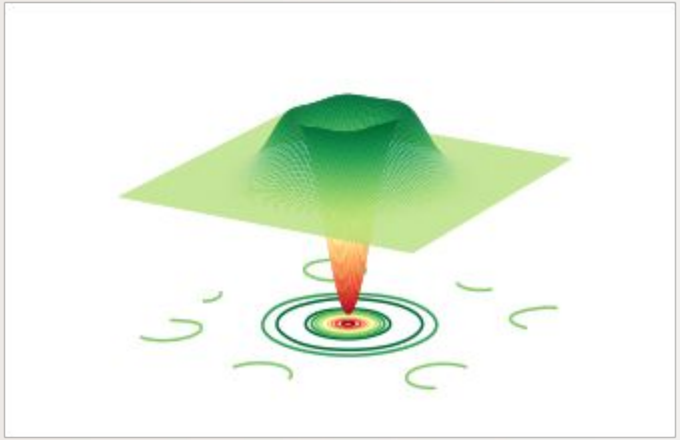
Gate model workflow



Further resources

- Forest: https://www.rigetti.com/forest
  Getting started:
  http://docs.rigetti.com/en/stable/start.html
- Cirq: https://github.com/quantumlib/Cirq
- Qiskit: https://qiskit.org/

Paradigm: Continuous-variable QC



Qubit	$\|\phi\rangle = \phi_0 \|0\rangle + \phi_1 \|1\rangle$
Qumode	$\|\psi\rangle = \int dx\, \psi(x) \|x\rangle$

Big players in continuous-variable QC

XANADU

Further resources

- Xanadu: https://www.xanadu.ai/
- Strawberry Fields: https://strawberryfields.readthedocs.io/en/latest/
  Installation instructions:
- https://strawberryfields.readthedocs.io/en/latest/installing.html

```
import strawberryfields as sf
from strawberryfields.ops import *

eng, q = sf.Engine(2)

with eng:
  # construct your quantum circuit
  # using blackbird code here

state = eng.run('fock', cutoff_dim=10)
```

Thank you for your attention!