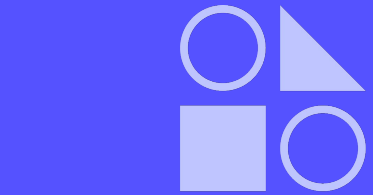
Covalent

Quantum Machine Learning using Covalent

A QAOA application



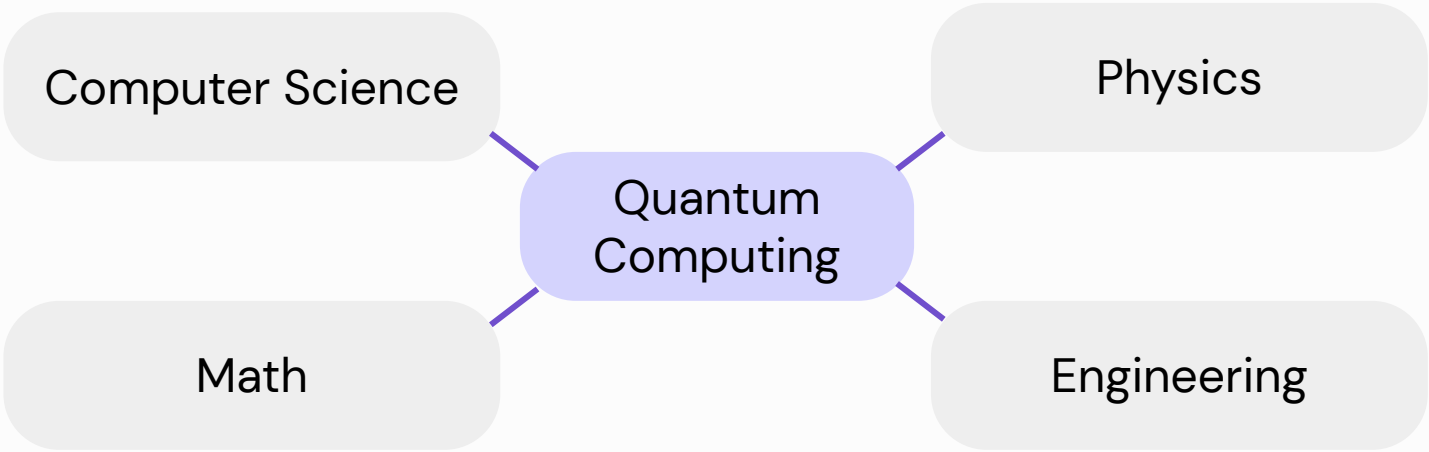
Covalent.

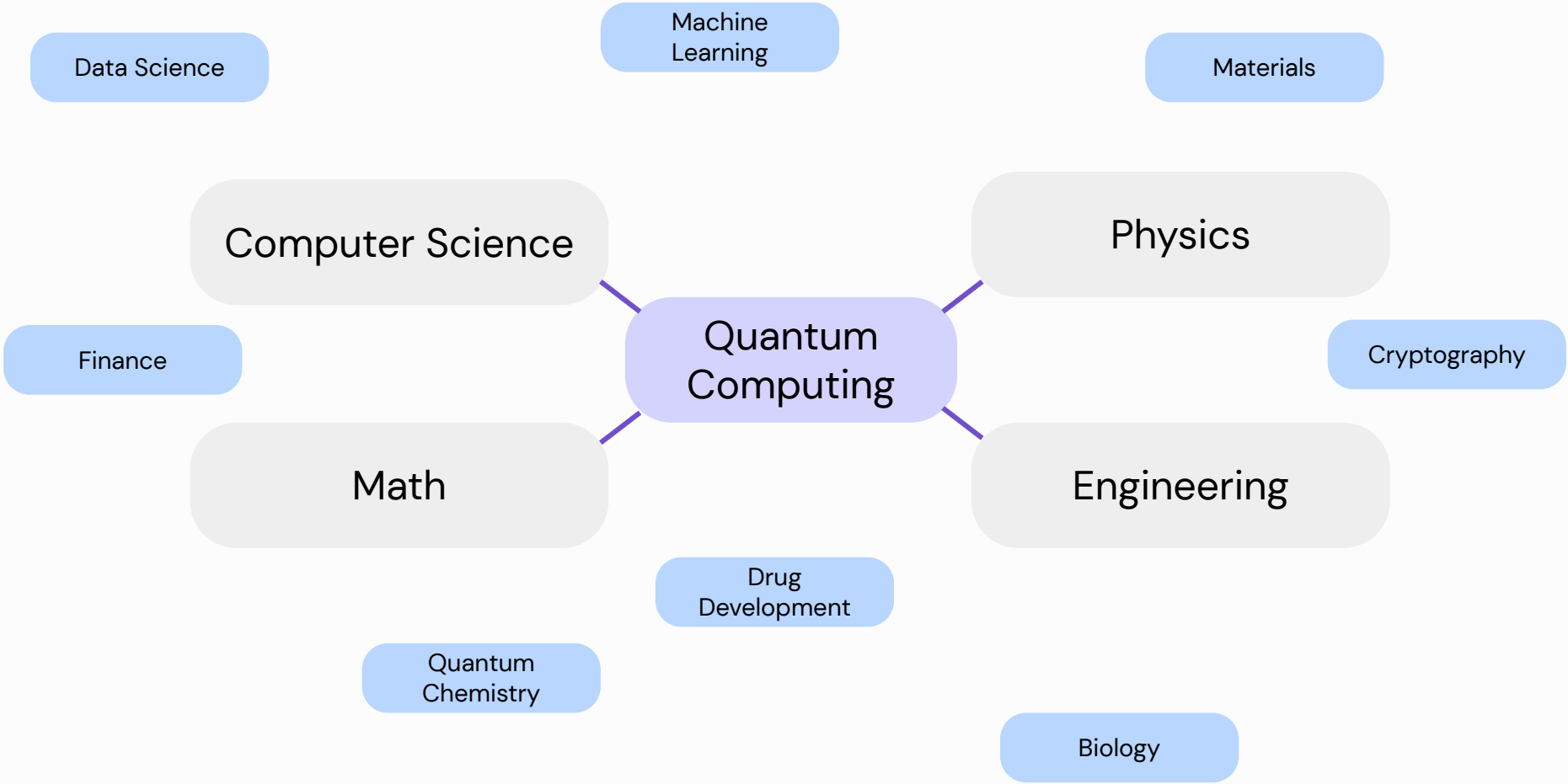
Anna Hughes, PhD
Quantum Software Engineer

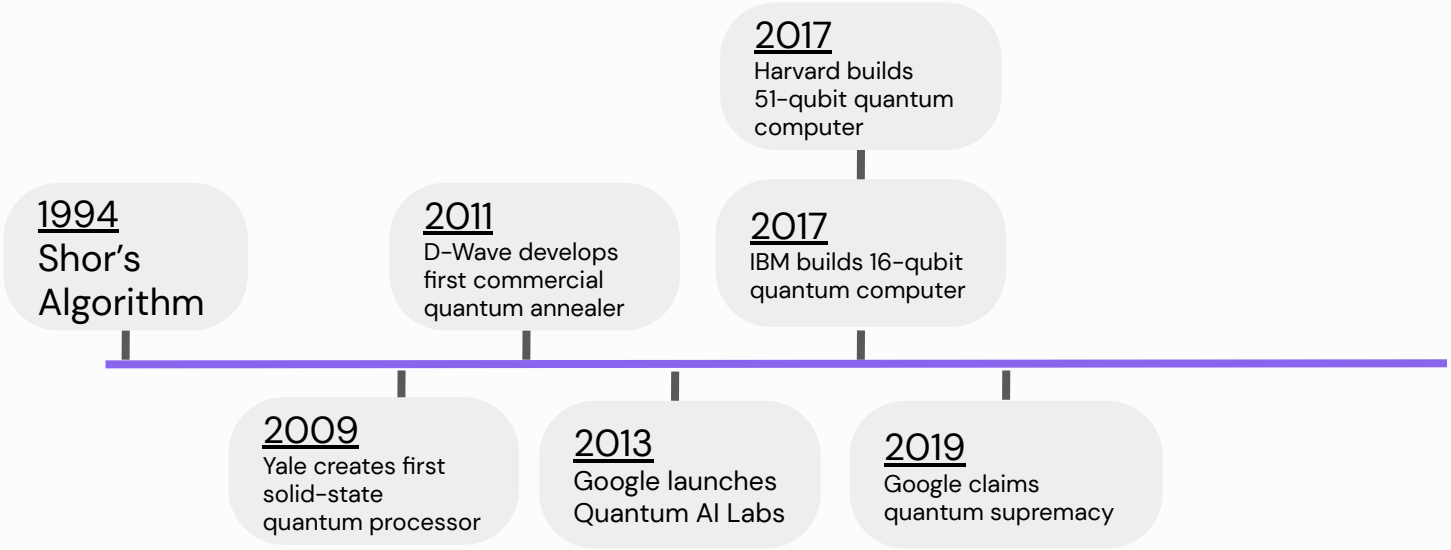
Some text to explain relevant role / background

Introduction to Quantum Computing

Subheading

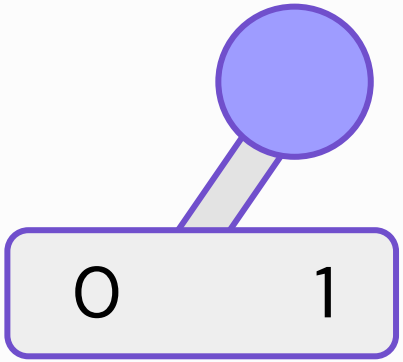

Computer Science
Physics
Quantum Computing
Math
Engineering


Data Science
Machine Learning
Materials
Computer Science
Physics
Finance
Quantum Computing
Cryptography
Math
Engineering
Drug Development
Quantum Chemistry
Biology


1994
Shor's Algorithm
2009
Yale creates first solid-state quantum processor
2011
D-Wave develops first commercial quantum annealer
2013
Google launches Quantum AI Labs
2017
IBM builds 16-qubit quantum computer
2017
Harvard builds 51-qubit quantum computer
2019
Google claims quantum supremacy

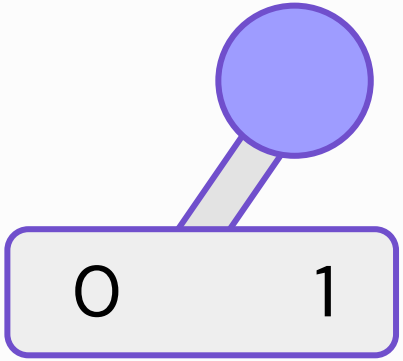
Classical Computers

Composed of **bits**, which can take on values of 0 or 1



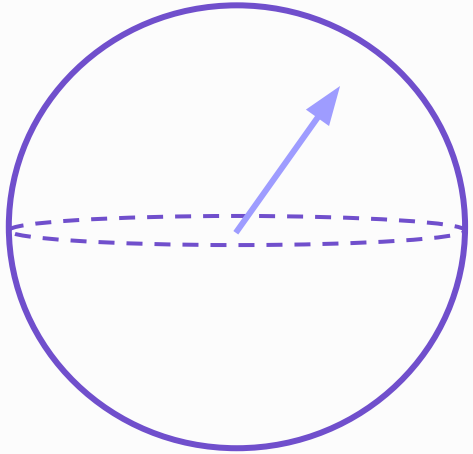
Classical Computers

Composed of **bits**, which can take on values of 0 or 1

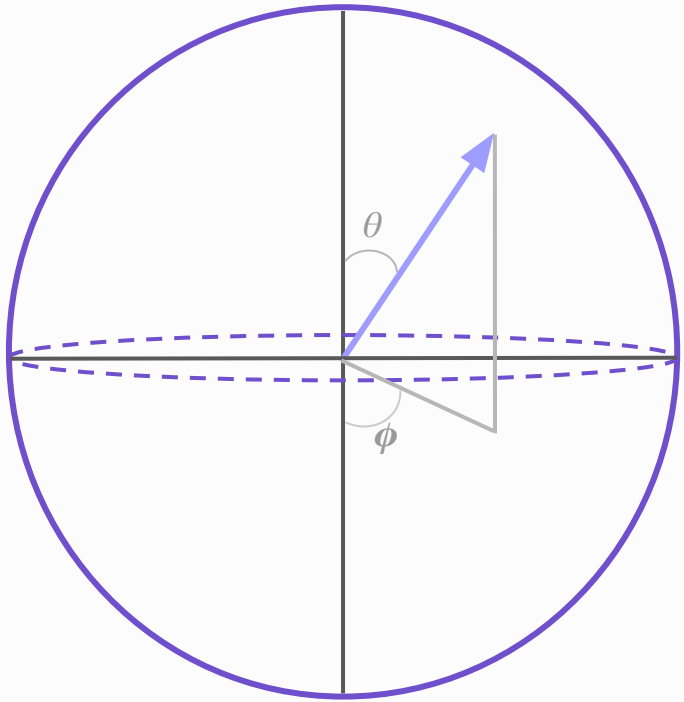


Quantum Computers

Composed of **qubits**, which can be in a superposition of 0 and 1

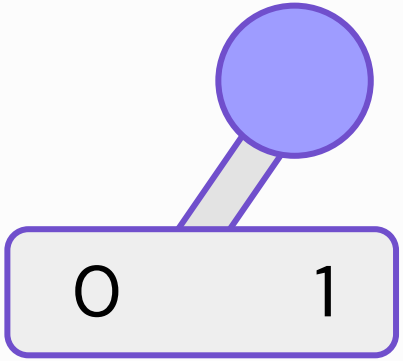


Bloch Sphere



Classical Computers

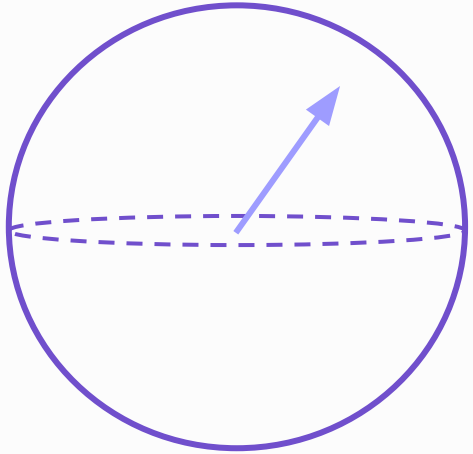
Composed of **bits**, which can take on values of 0 or 1



Deterministic measurements

Quantum Computers

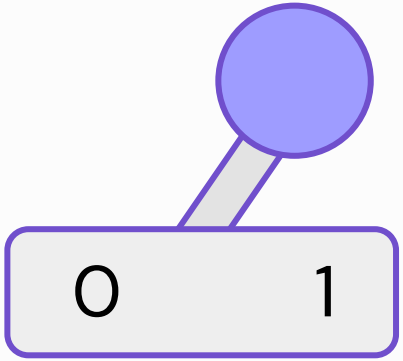
Composed of **qubits**, which can be in a superposition of 0 and 1



Probabilistic measurements

Classical Computers

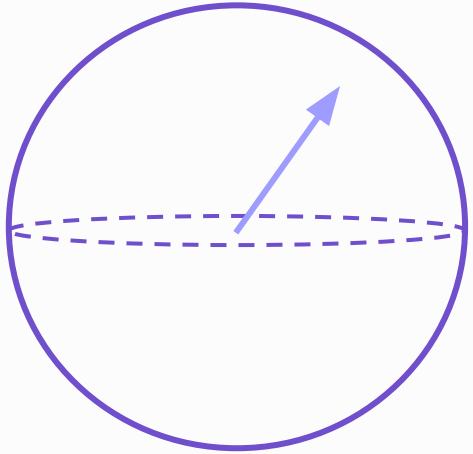
Composed of **bits**, which can take on values of 0 or 1



Deterministic measurements

Quantum Computers

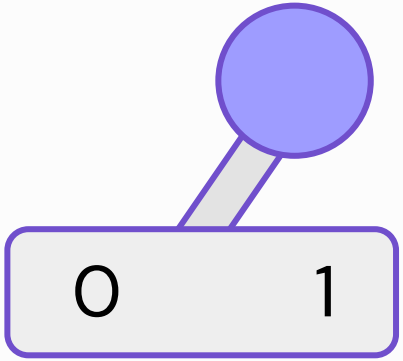
Composed of **qubits**, which can be in a superposition of 0 and 1



Probabilistic measurements

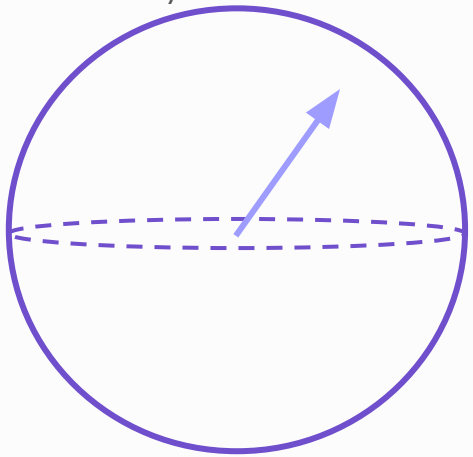
Classical Computers

If you have N bits, you have 2^N states that you can only execute 1 at a time (or in parallel)



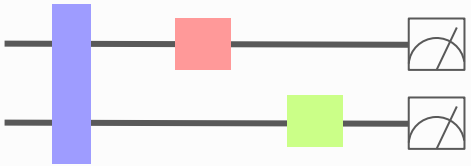
Quantum Computers

If you have N qubits, you can encode all 2N components into one state simultaneously



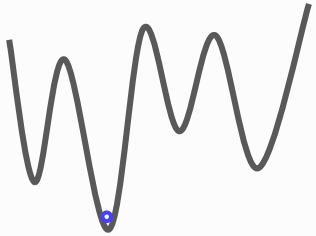
Probabilistic measurements

Types of Quantum Computers.



Gate-Based.

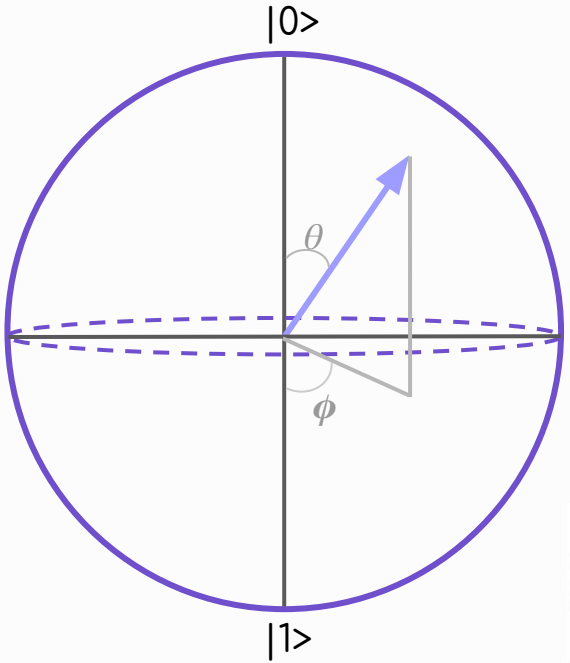
- Broad applications
- Apply gates, or circuit operations, to quantum state
-



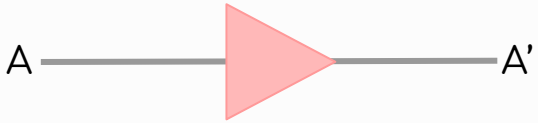
Quantum Annealers.

- Can solve optimization problems: search an energy landscape for the lowest-energy solution
- Problem encoded as a Hamiltonian
-

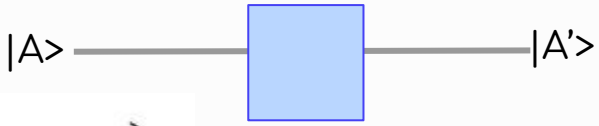
Quantum Gates



- Classical NOT Gate

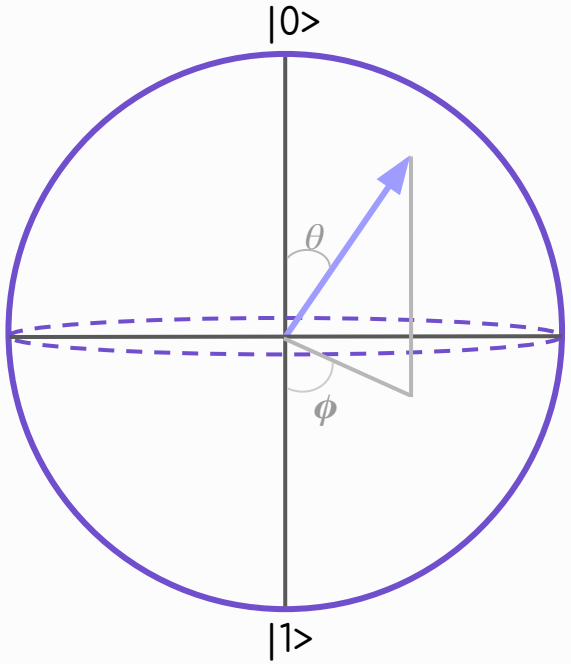


- Pauli X Gate



$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Quantum Gates



- Pauli X Gate

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

- Pauli Y Gate

$$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

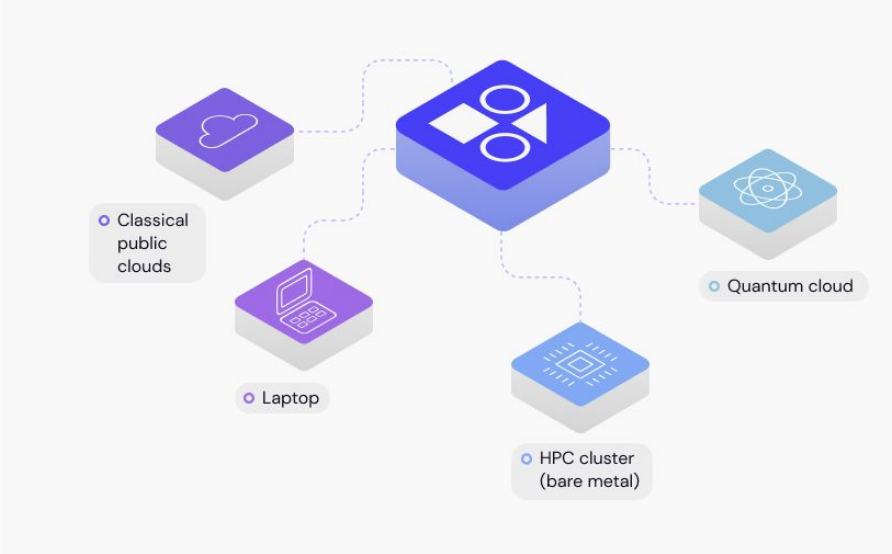
- Pauli Z Gate

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Covalent is an open source workflow orchestration platform for quantum and high performance computing

Covalent is designed to make your experiments:

- Modular
- Scalable
- Reproducible



Why does Covalent exist?

Computational research

Challenges.

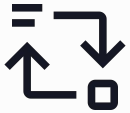
Experimental-Organization

Manageability.

- Organize 1000s of experiments
- Experimental versioning of multiple runs
- Input/parameters logging for each run
- Checkpointing costly computations
- Job failure management
- Real time monitoring

Reproducibility.

- Environment saving/caching
- Hardware metadata caching
- Inputs/parameters logging
- Experiment dependant device setup

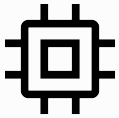


Shareability.

- Experiment organization
- Clear and intuitive code structure
- Reproducible experiment parameters and setup

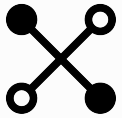
Challenges.

Computational



Hardware-potpourri.

Hybrid research / experiments. Single experiment now contains CPU+GPU+QPU+TPU



Interactive HPC.

High performance computing in the age of rapid prototyping and experimentation.

Distributed computation.

Era of high-throughput calculations. Running massive parallel jobs at scale.

Limited resource.

Long queue time, quantum queues and Server-less HPC!

How Covalent can help.

Solution 1.

Hybrid research / experiments. Single experiment now contains CPU+GPU+QPU+TPU

Solution 2.

High performance computing in the age of rapid prototyping and experimentation.

Solution 3.

Era of high-throughput calculations. Running massive parallel jobs at scale.

Solution 4.

Long queue time, quantum queues and Server-less HPC!

3. Real-time monitoring

Visual overview

Visualize an share your workflow to transfer knowledge as fast as possible

Status/Error updates

Get real time Updates on errors and completion

Checkpoints

Stores anything and everything automatically without a single line of code

Meta-data

Maintains details from environment to hardwares used

Parameter

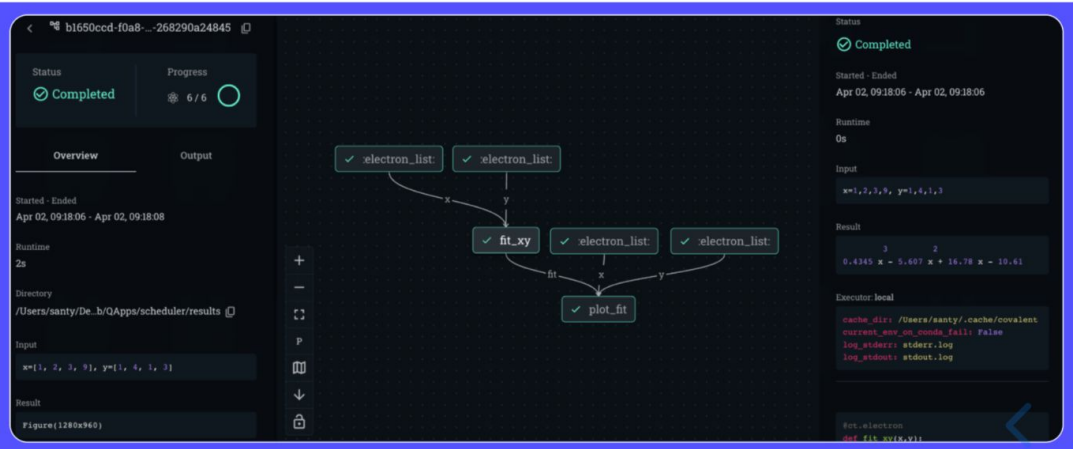
Never forget the hyper-parameters that worked

Interactive

Start/stop/manage your HPC jobs right from the UI

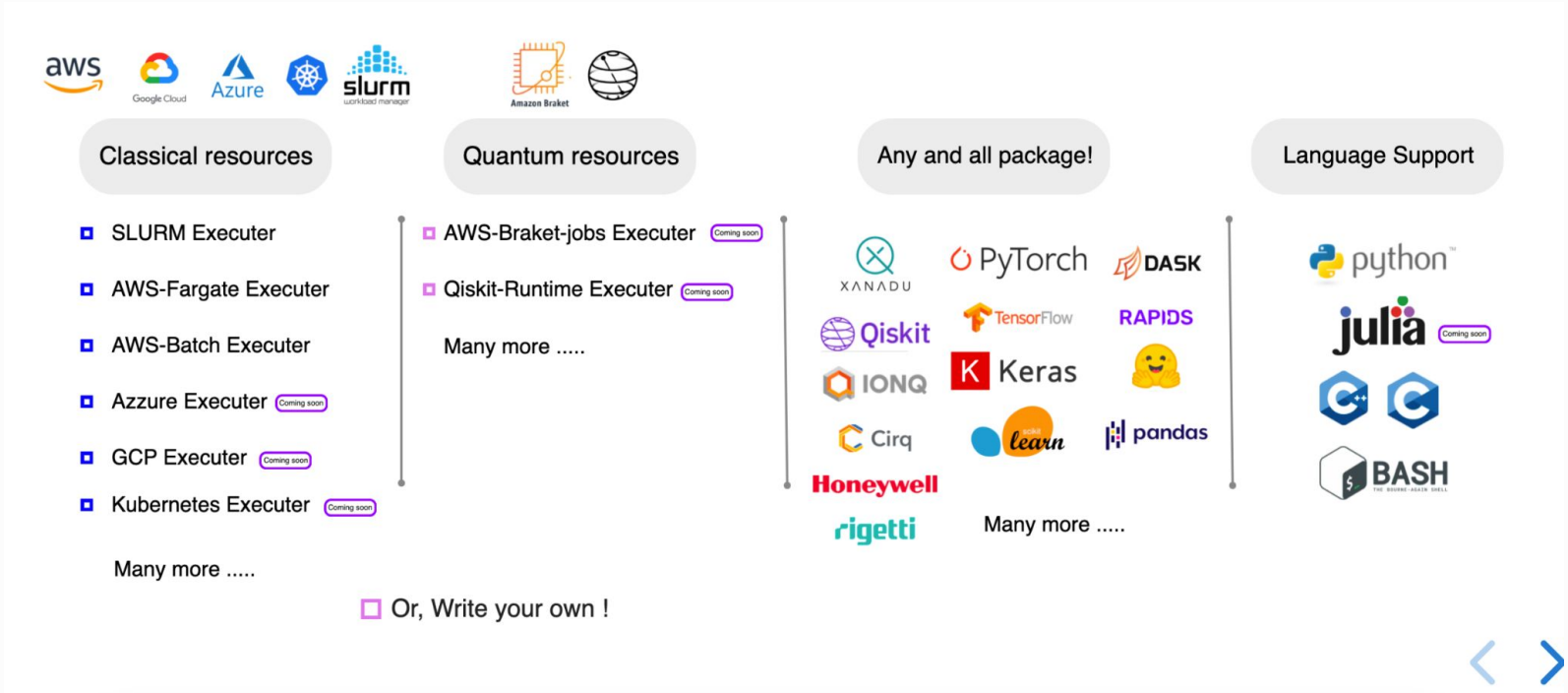
Beautiful and interactive UI

bring your workflows to life!



Where Covalent fits in the stack.

Ecosystem


aws
Google Cloud
Azure
slurm
workload manager
Amazon Braket
Classical resources
Quantum resources
Any and all package!
Language Support
SLURM Executer
AWS-Fargate Executer
AWS-Batch Executer
Azzure Executer
Coming soon
GCP Executer
Coming soon
Kubernetes Executer
Coming soon
Many more .....
AWS-Braket-jobs Executer
Coming soon
Qiskit-Runtime Executer
Coming soon
Many more .....
XANADU
PyTorch
DASK
Qiskit
TensorFlow
RAPIDS
IONQ
Keras
Cirq
learn
pandas
Honeywell
rigetti
Many more .....
python
julia
Coming soon
BASH
Or, Write your own !

There is more.

Pythonic workflows	Automatic checkpointing	Multiple language support	Little-to-no overhead	Customizable
Reproducibility	Code locally, un anywhere		Intuitive User-interface	Natively hybrid workflows
Native parallelization				Variety of executers
Code isolation	Parameter caching	Cloud Agnostic	Interactive jobs	Start locally and scale

Comparison table.

Languages.

- Python
- C / C++
- Julia*
- Bash

Executors.

- Local executors
- Slurm
- AWS*
- GCP*
- Azure*

*Roadmap item

Code.

```
# Transaction in Python
session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

Code.

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
for x in order.line_items:
 db.inventory.update(
   { _id  : x.item } ,
   { $inc : { number : -1 * x.quantity } },
   session=session

session.start_transaction()
```


```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
for x in order.line_items:
 db.inventory.update(
   { _id  : x.item } ,
   { $inc : { number : -1 * x.quantity } },
   session=session

session.start_transaction()
```

Code.

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```


```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

Code.

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

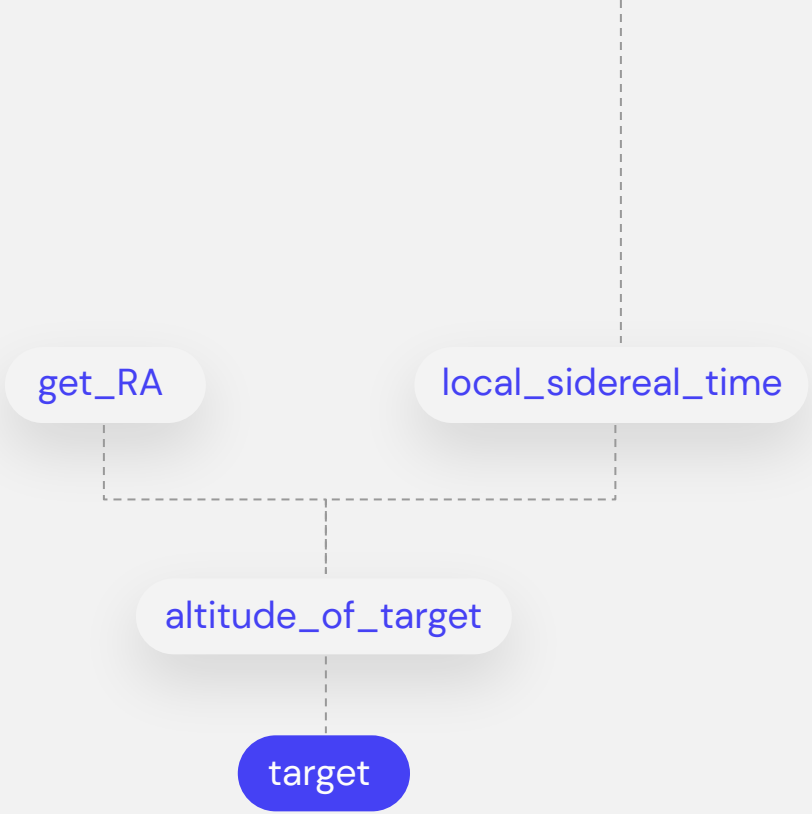

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

```
# Transaction in Python

session.start_transaction()
order = { line_items : [ { ite10m : 5, quantity: 6 } ] }
db.orders.insertOne( order, session=session );
```

Components.

Diagram



Optimizing financial portfolios on superconducting and trapped ion quantum computers.

A combinatorial optimization problem arises whenever we are presented with a number of choices and we task ourselves with selecting the best choice. One example surfaces from a problem often posed by delivery companies: “given a set of possible vehicle routes, which one permits the driver to deliver all parcels the fastest?” Another is asked in telecommunications: “given a set number of approved locations for broadcast antennas, which sites should be used to maximize the reach of mobile phone signal?”

Lastly, and directly related to this post, a financial investor asks: “Given a list of stocks and a budget for how many different stocks can be bought, which assignment maximizes the amount of cash accumulated over time with the minimal amount of risk?”.

Appendix / Page Break

Subheading

Covalent

Thank You

covalent.xyz

agnostiqHQ/covalent

@covalentxyz

Covalent

Learn more

agnostiq.ai

agnostiqHQ

@agnosticHQ

Manage, deploy & scale workloads across the worlds most advanced computing hardware.

Workflows are composed of python decorators that create what we call lattices and electrons. Electrons are workflow components and lattices are groups of electrons.