# SW stack

## for noisy intermediate-scale quantum devices

WACQT

NordIQuEst

OpenSuperQPlus

Miroslav Dobsicek, October 26, 2023

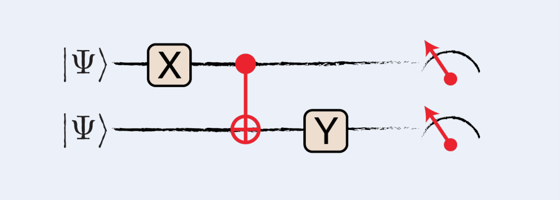
# Presentation overview

- SW stack overview
- User-space quantum stack
- Circuit level assembly
- Hardware level encoding

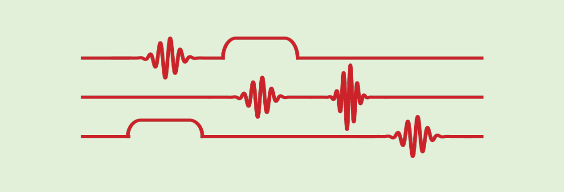
# SW stack

```
let shorCorrector (qs:Qubits) =
    let out = xflipSyndrome qs.[0 .. 2]
    if (out > 0) then
        X [qs.[out - 1]]
```

Circuit design



Compiler



Pulse schedules

Instrument orchestration

**Computer science domain**
Output for idealized quantum computer

Co-design for NISQ devices

**Experimentalist domain**
Single-user environment, lab work

WACQT

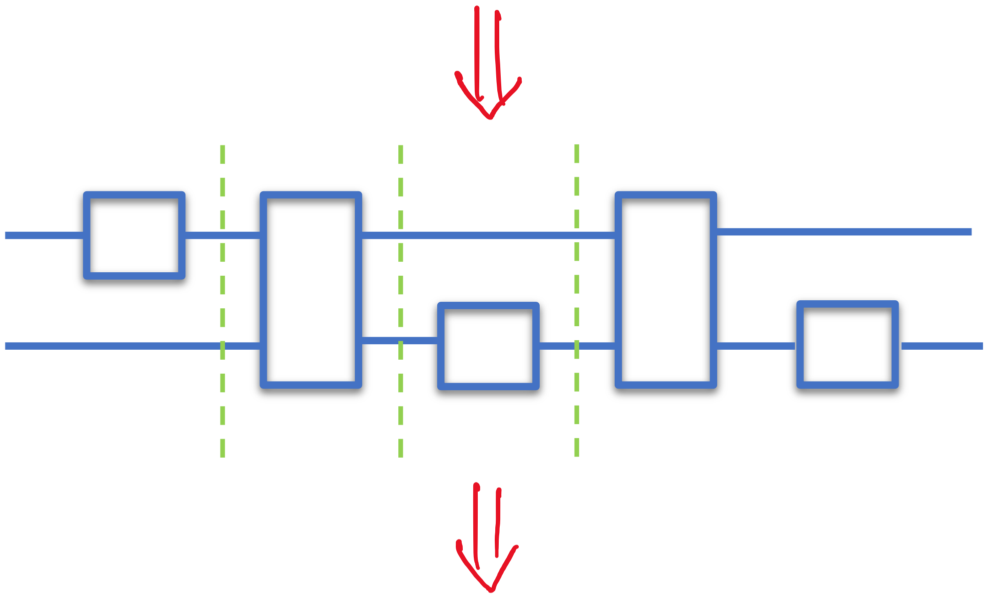
NordIQuEst

OpenSuperQPlus

# SW stack is built around the circuit model

What is above?

How do we get a circuit?



What is below?

How do we run it?

WACQT

NordIQuEst

OpenSuperQPlus

# High level parts of a SW stack

How do we generate quantum circuits?

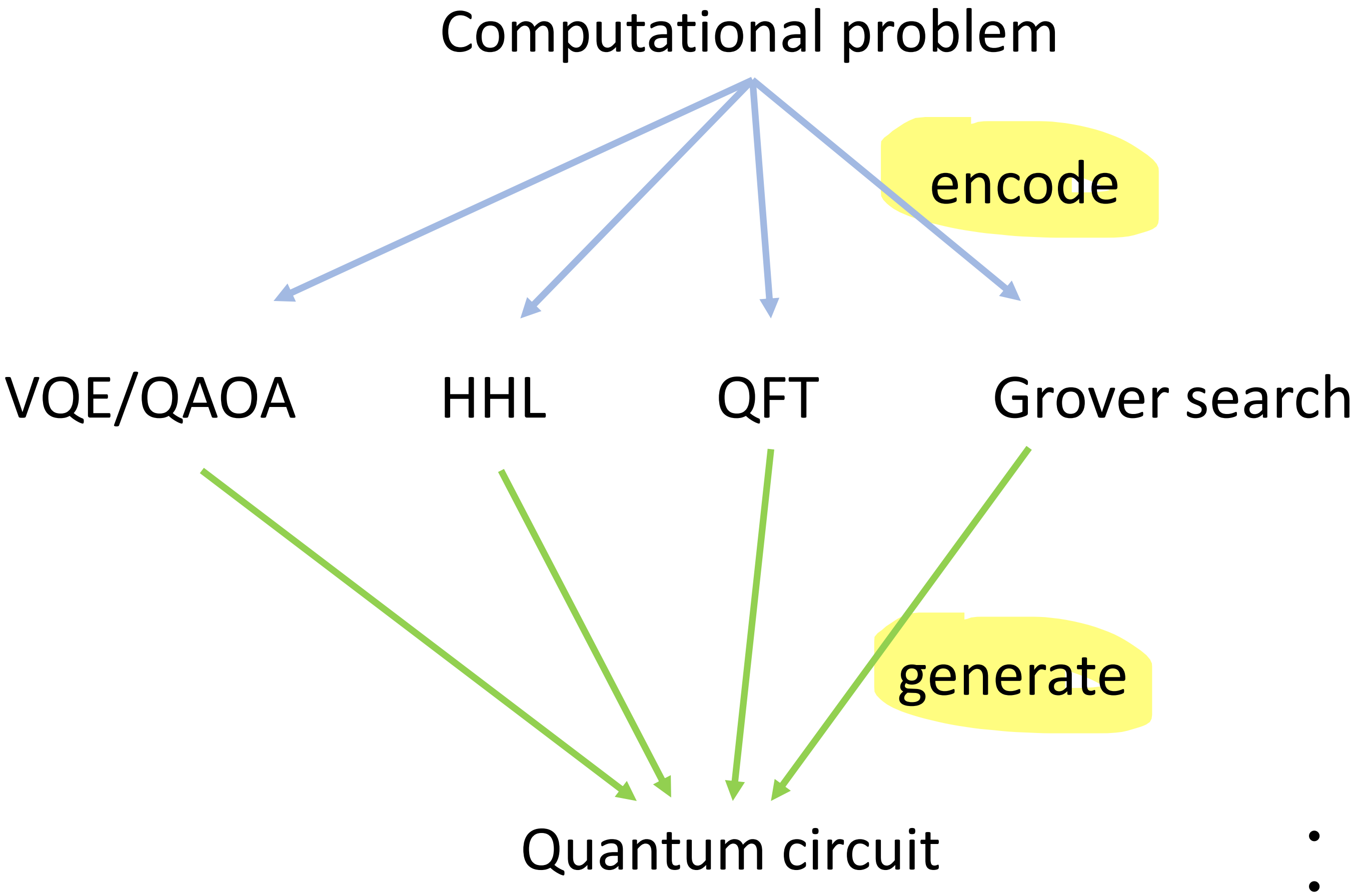
Generic methods

- **Encode** your problem into known quantum algorithms
- **Embed** a classical circuit into a quantum one through reversible logic
- **Automatically decompose** large transformations into sequences of smaller ones

Attacking directly the problem

- **Design** your own quantum algorithm

# 1. Problem enconding into an existing quantum algorithm



Computational problem

encode

VQE/QAOA HHL QFT Grover search

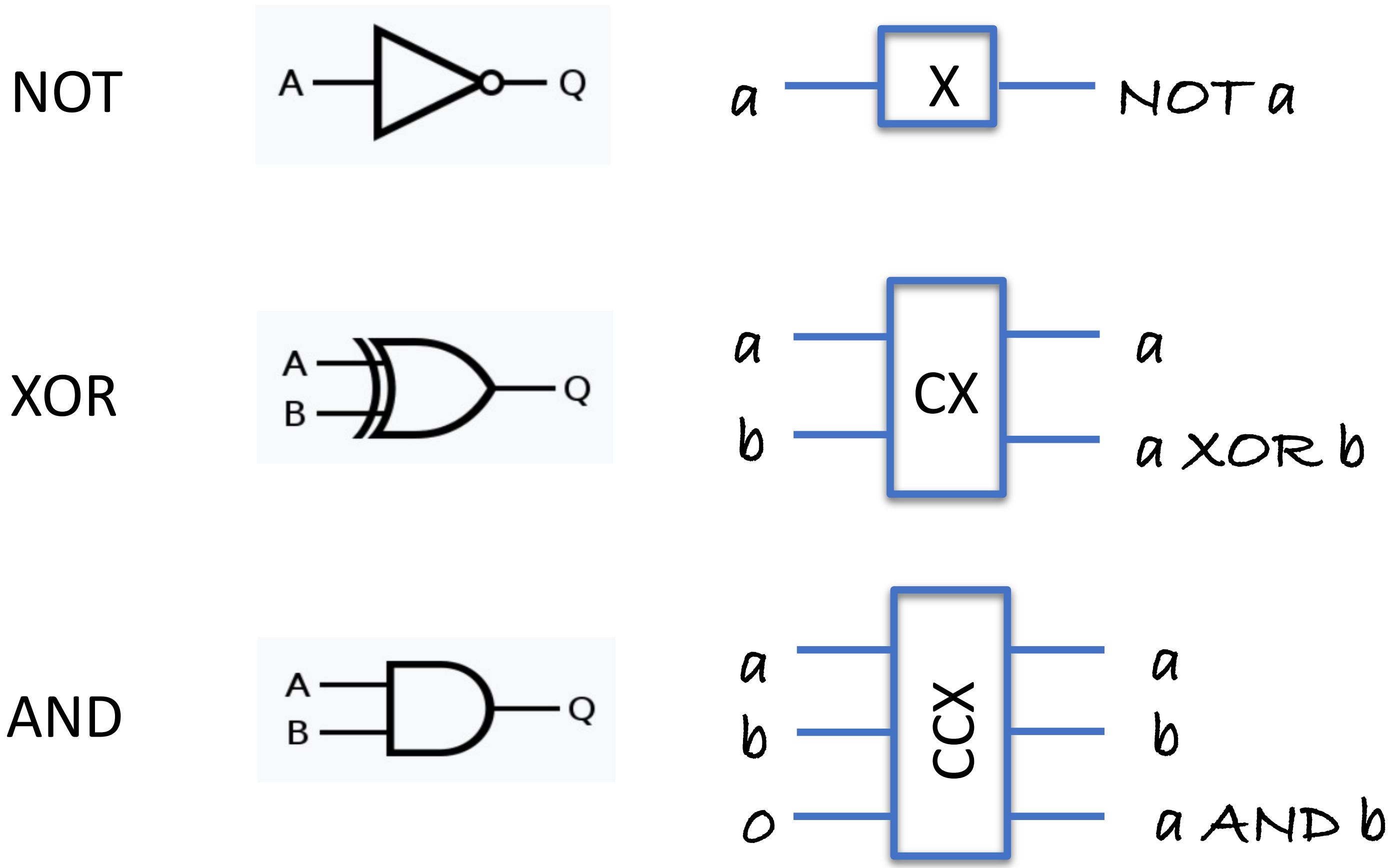
generate

Quantum circuit

This is currently the most feasible way how to do a computation on a quantum computer.

- VQE – quantum chemistry problems
- QAOA – combinatorial opt. problems
- HHL – systems of linear equations (ML)
- QFT – detect group-like properties
- Grover search – generic square root speed-up

# 2. Embedding of classical circuits via reversible logic



Classical logical gates mostly map to quantum gates in 1:1 fashion.

A quantum circuit generated in this way will have the same **overall** complexity as the classical circuit. Not better or worse. But! it will be capable of working with superposition of states.

**The cost are extra qubits guaranteeing reversibility.**

Do you know that the QFT circuit and the circuit for a classical FFT are structurally the same?

# 3. Automatic decomposition

Desired transformation:

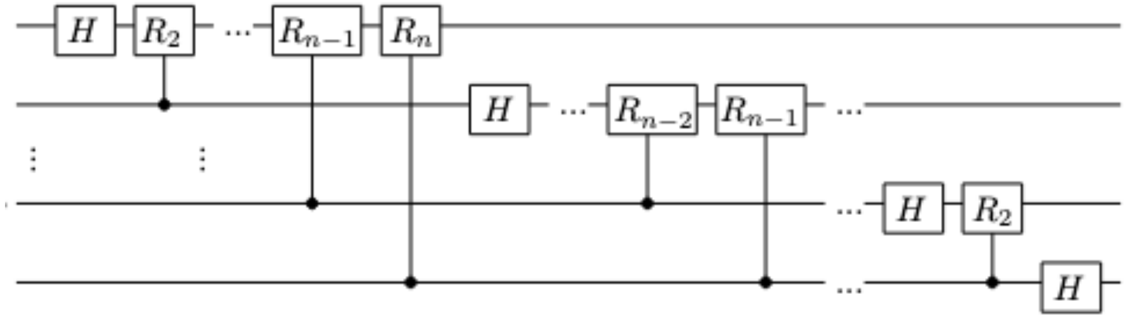
$$\mathrm{QFT} : |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{xk} |k\rangle.$$

Matrix form:

$$F_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

You start with a mathematical description of the desired unitary transformation and write it down in a matrix form. Then apply unitary decomposition algorithm(s). This process is usually based on Singular Value Decomposition (SVD).

This approach is unlikely to lead to efficent circuits! The number of generated gates is generally exponential in the number of qubits.



Efficient circuit for QFT (if you get lucky)

# 4. Novel design

- Not an easy task
- Much of our reasoning is still tied to circuits and complex Hilbert spaces
- We are "chasing vectors around" in an analogy to "chasing bits around"
- The most active fields in quantum algorithm theory are:
  - Quantum error correction codes
  - Quantum complexity classes
    - MIP* = RE, Certifiable randomness, Classically verifiable quantum advantage
  - Finding new classical algorithms by ”dequantization”

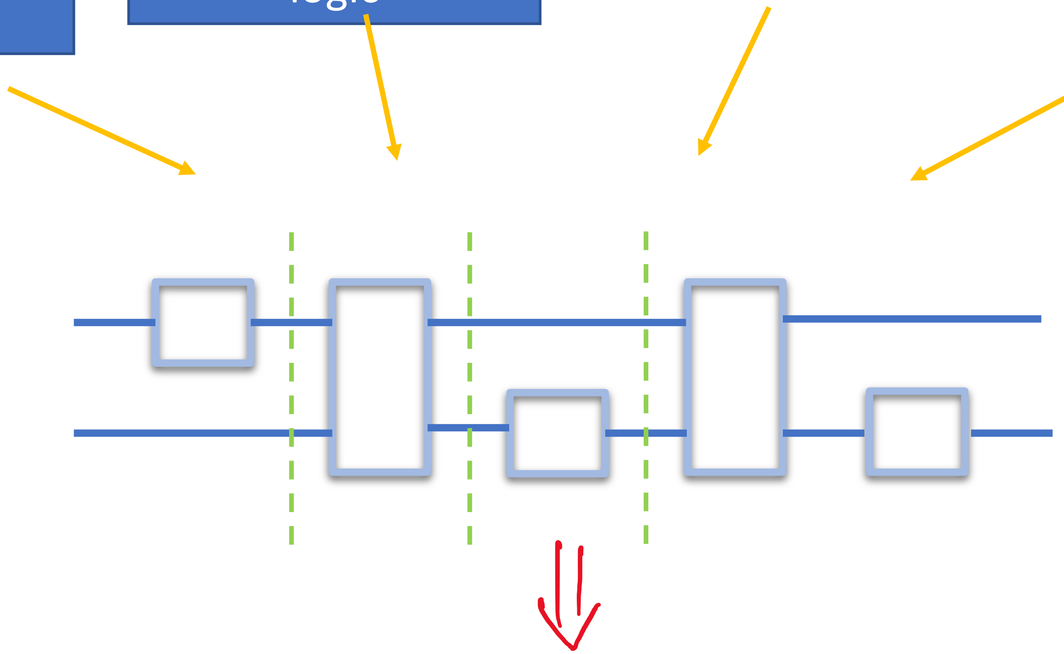
# Gate-based quantum computing model

Problem encoding into known generic algorithms

Embedding through reversible logic

Automatic decomposition

Novel development



What is below?

How do we run it?

generate ✓

SW stack

Control engineering

Qubit technology

WACQT

NordIQuEst

OpenSuperQPlus

# A number of circuit optimizations

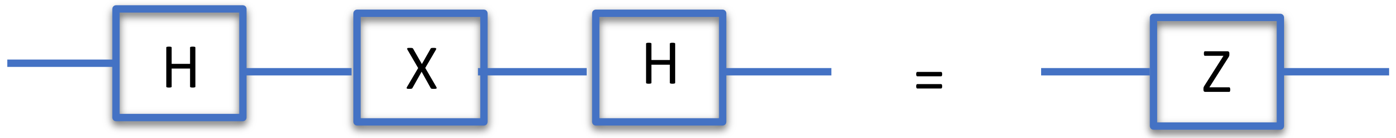
- **Circuit compression** – minimize the number of gates used (coupling gates in particular)
- **Unroll/decompose** to the native gate set supported by the quantum HW
- **Optimal routing** – map the logical circuit to the physical chip while respecting its connectivity map. Insert SWAP gates where needed.
- (Insert **error mitigation** gates).

These optimizations techniques are partwise orthogonal, quantum HW dependent, and may be applied iteratively/recursively in order to achieve the best results.

# Circuit compression

- The most common technique is to exploit logical circuit identities

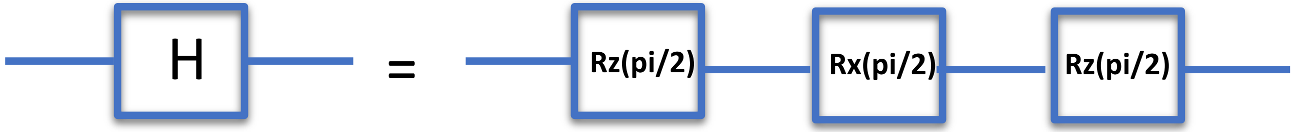
Eg:



- One of the newer approaches is called **ZX-calculus**.
  - It relaxes the unitarity condition: operates in a less restrictive linear regime instead
  - But, it's not always possible to revert back to a unitary circuit

# Unrolling/decomposition

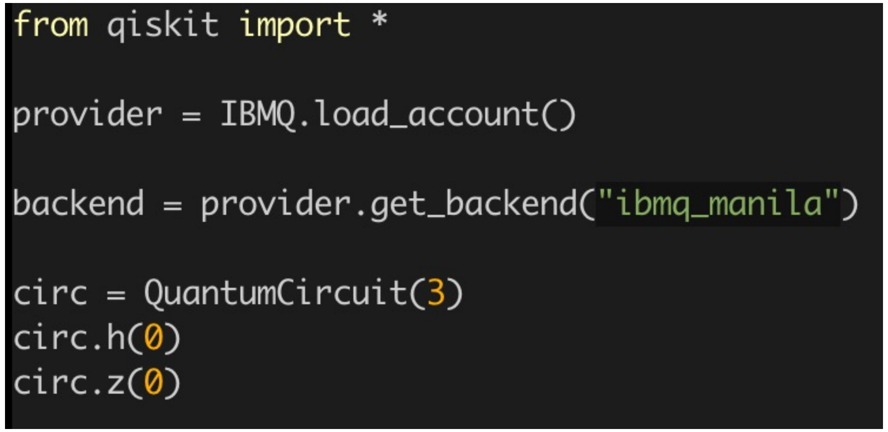
- There are many universal gate sets for quantum computing.
- For superconducting qubits common entangling gates are: **CX**, **CZ**, or **iSWAP** accopanied with **Rx(..)** and **Rz(..)** single qubit rotations. We call it a **native** gate set.
- SW stack typically contains a **library** of definitions of other **commons gates** in terms of the native universal gate set. Thus, for example, the Hadamard gate H can be '*unrolled*' in terms of Rx(..) and Rz(..).



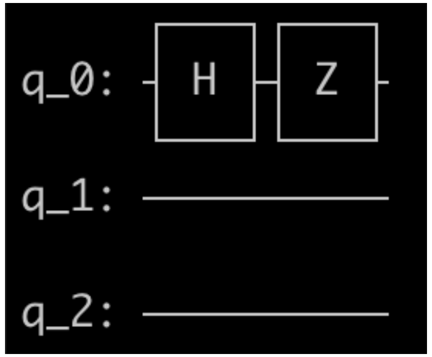
- Uncommon gates needs to be (brute-force) decomposed (eg. by SVD).

# Example: Qiskit’s built-in circuit optimizations

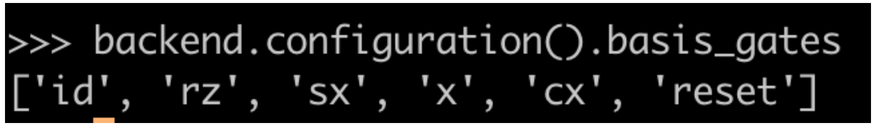
Original circuit



```
from qiskit import *

provider = IBMQ.load_account()

backend = provider.get_backend("ibmq_manila")

circ = QuantumCircuit(3)
circ.h(0)
circ.z(0)
```



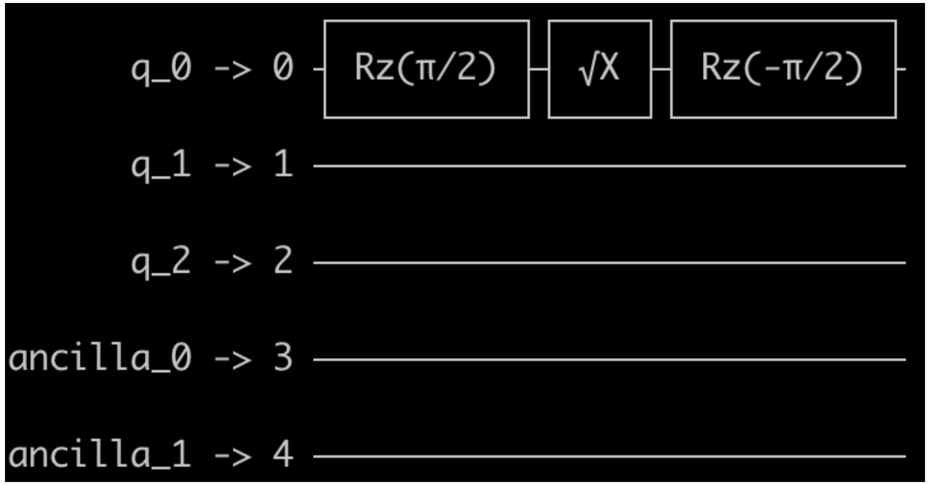
Check the native gate set



```
>>> backend.configuration().basis_gates
['id', 'rz', 'sx', 'x', 'cx', 'reset']
```

Transpiled circuit

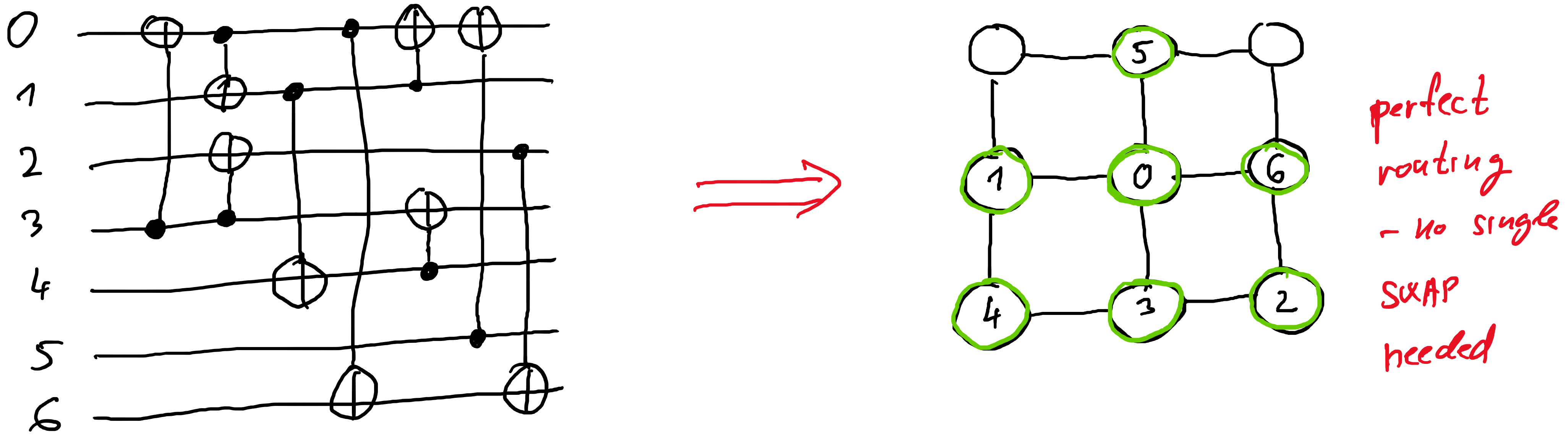
```
trc = transpile(circ, backend)
```



Unrolling and compression has been applied.

# Optimal routing

- A quantum chip typically supports only interactions between nearest-neigbour qubits. We talk about a connectivity map.
- More distant interactions are achieved via inserting (multiple) SWAP gates. We want to minimize the number of burdersome SWAPs.

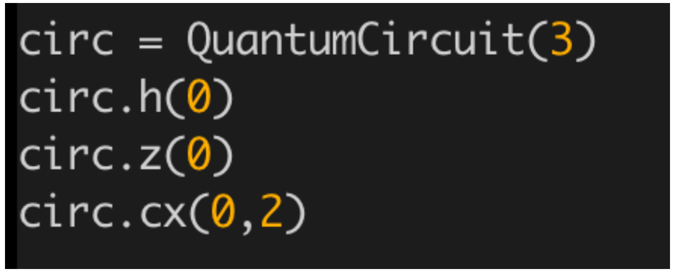


perfect routing – no single SWAP needed

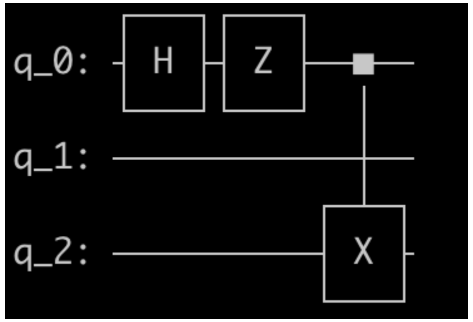
This problem is quite similar to a CPU register allocation.

# Example: Qiskit’s built-in circuit optimizations

Original circuit

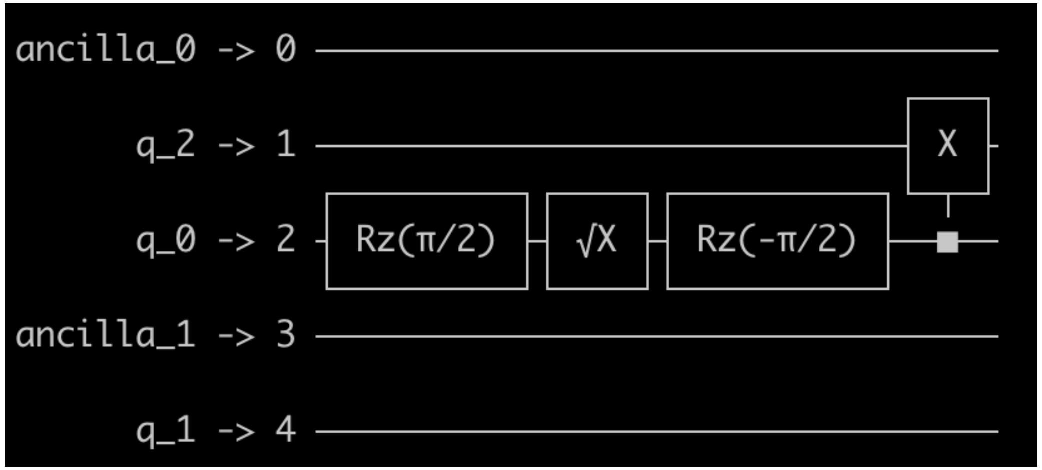


```
circ = QuantumCircuit(3)
circ.h(0)
circ.z(0)
circ.cx(0,2)
```



Transpiled circuit

```
trc = transpile(circ, backend)
```



Manila’s coupling map

Unrolling, compression
**and routing** has been applied.

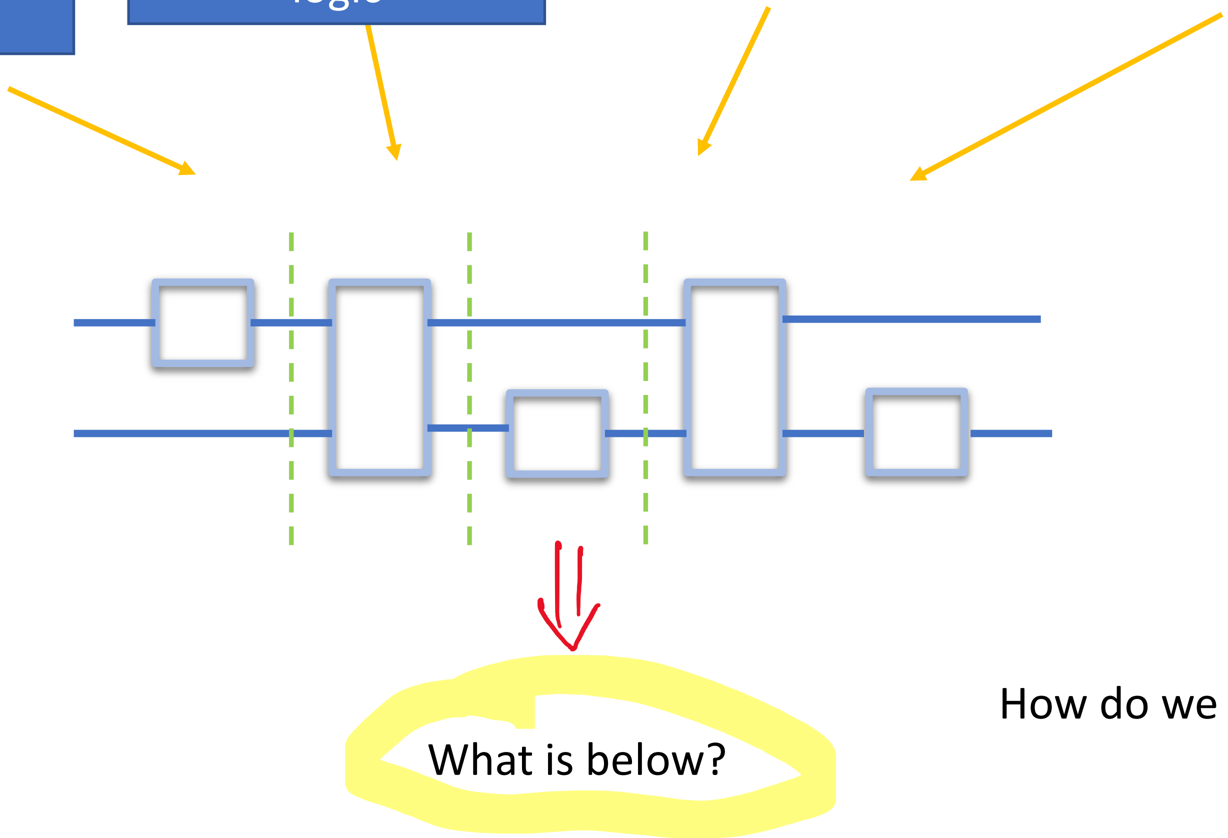
# Gate-based quantum computing model

Problem encoding into known generic algorithms

Embedding through reversible logic

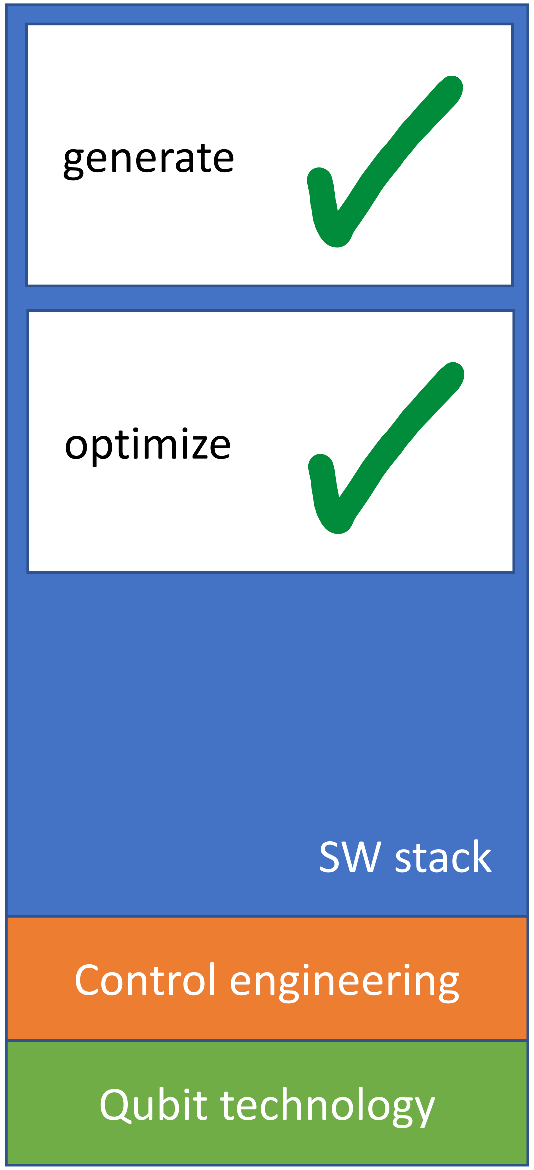
Automatic decomposition

Novel development



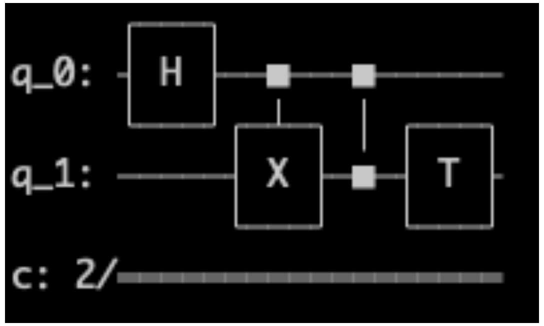
What is below?

How do we run it?

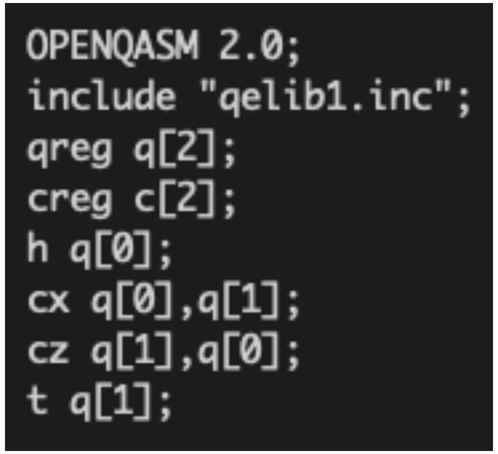


# Quantum circuit execution

- The generated & optimized circuit needs to be converted from an internal high-level representation (say a Python object) to a flattened textual or binary representation suitable for network transfer and execution.



assemble

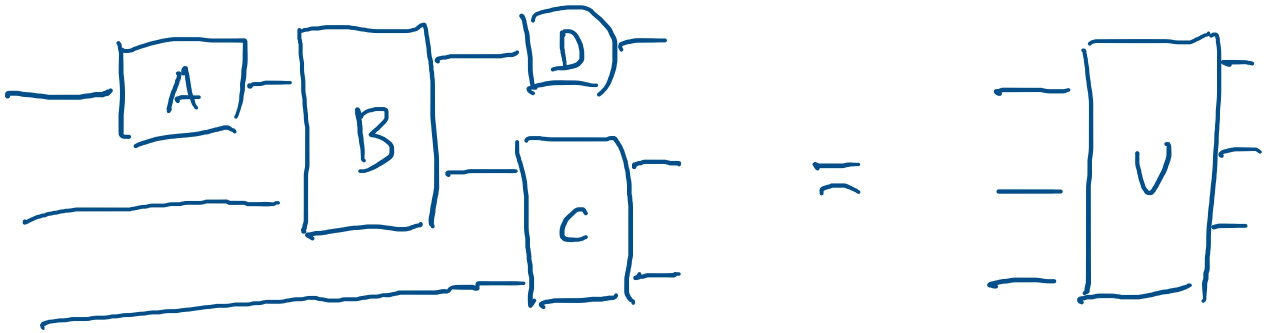


```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
cz q[1],q[0];
t q[1];
```

- *OpenQASM V2* from IBM has emerged as a practical standard due to its simplicity and permissive licensing.
- *OpenQASM V2* is also often used as inter-operability language between different circuit toolkits.

# Execution target: simulator

- Gates are expanded into their matrix form representation
- Matrices and the input vector are multiplied to produce the output vector



where

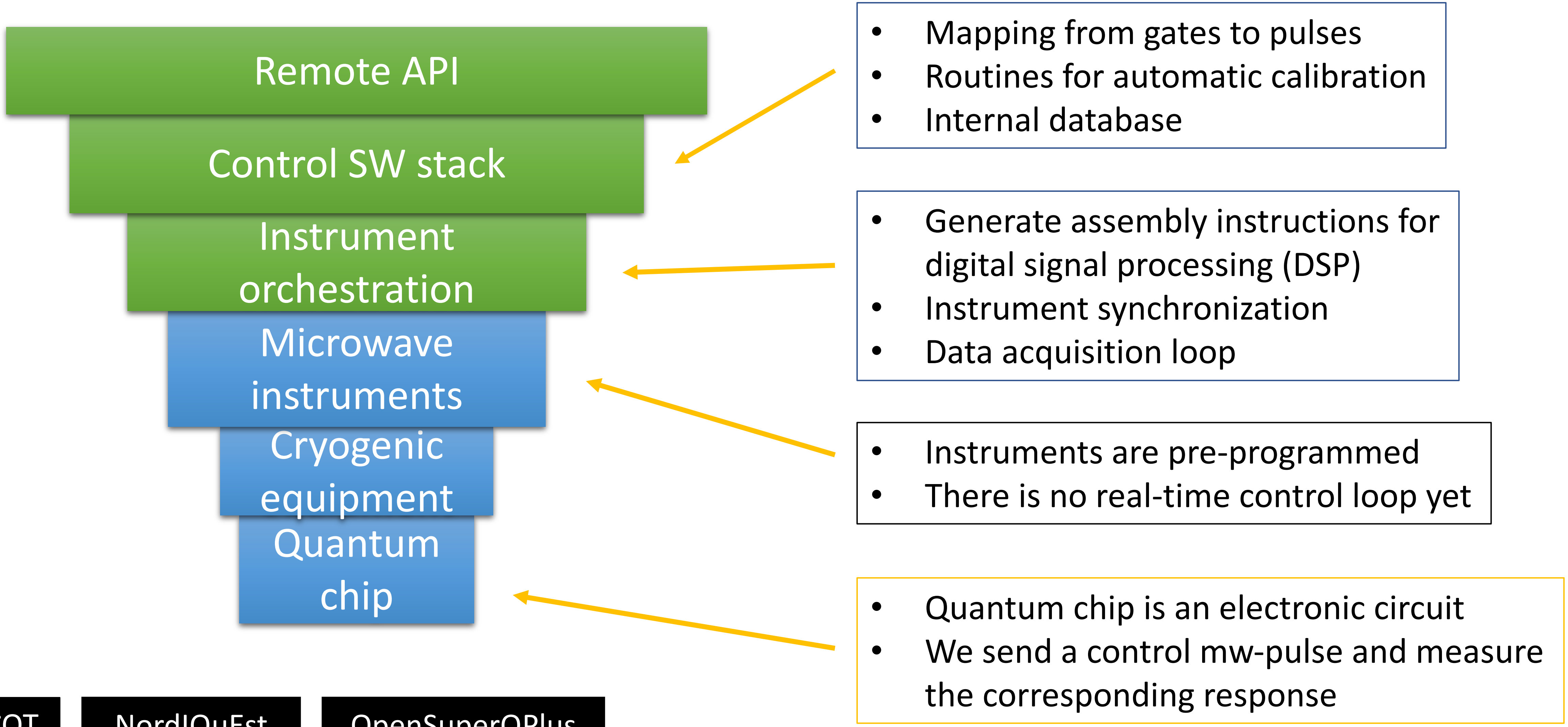
Matrix multiplication

$$U = (D \otimes C) \cdot (B \otimes I) \cdot (A \otimes I \otimes I)$$

Tensor product

- |output> = U . |input>
- A big advantage is that one gets the whole output vector!
- Simulators are slow and memory consuming!!!

# Execution target: NISQ device



Remote API

Control SW stack

Instrument orchestration

Microwave instruments

Cryogenic equipment

Quantum chip

- Mapping from gates to pulses
- Routines for automatic calibration
- Internal database

- Generate assembly instructions for digital signal processing (DSP)
- Instrument synchronization
- Data acquisition loop

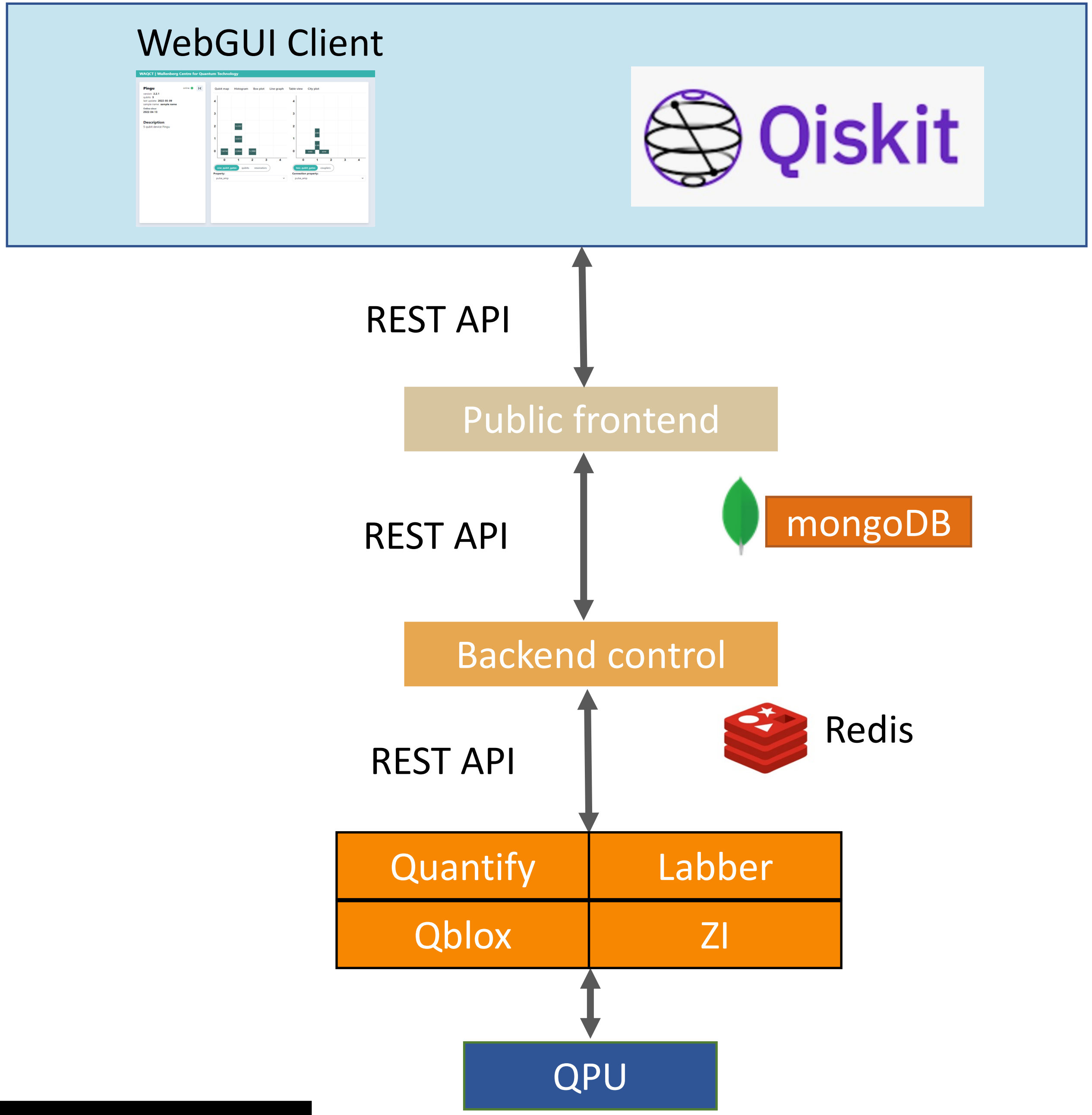
- Instruments are pre-programmed
- There is no real-time control loop yet

- Quantum chip is an electronic circuit
- We send a control mw-pulse and measure the corresponding response

WACQT

NordIQuEst

OpenSuperQPlus



WebGUI Client

Qiskit

REST API

Public frontend

REST API

mongoDB

Backend control

REST API

Redis

Quantify

Labber

Qblox

ZI

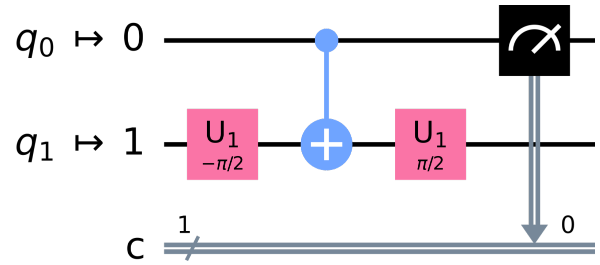
QPU

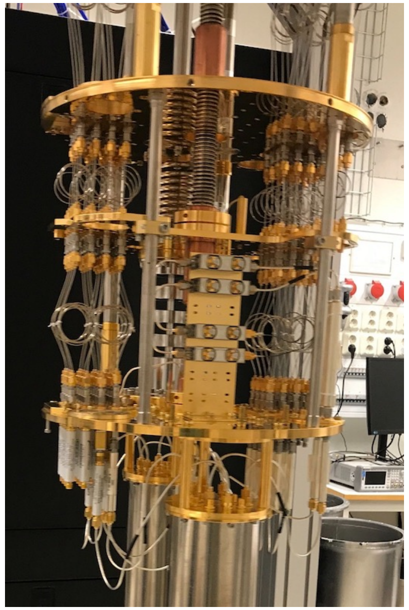
WACQT

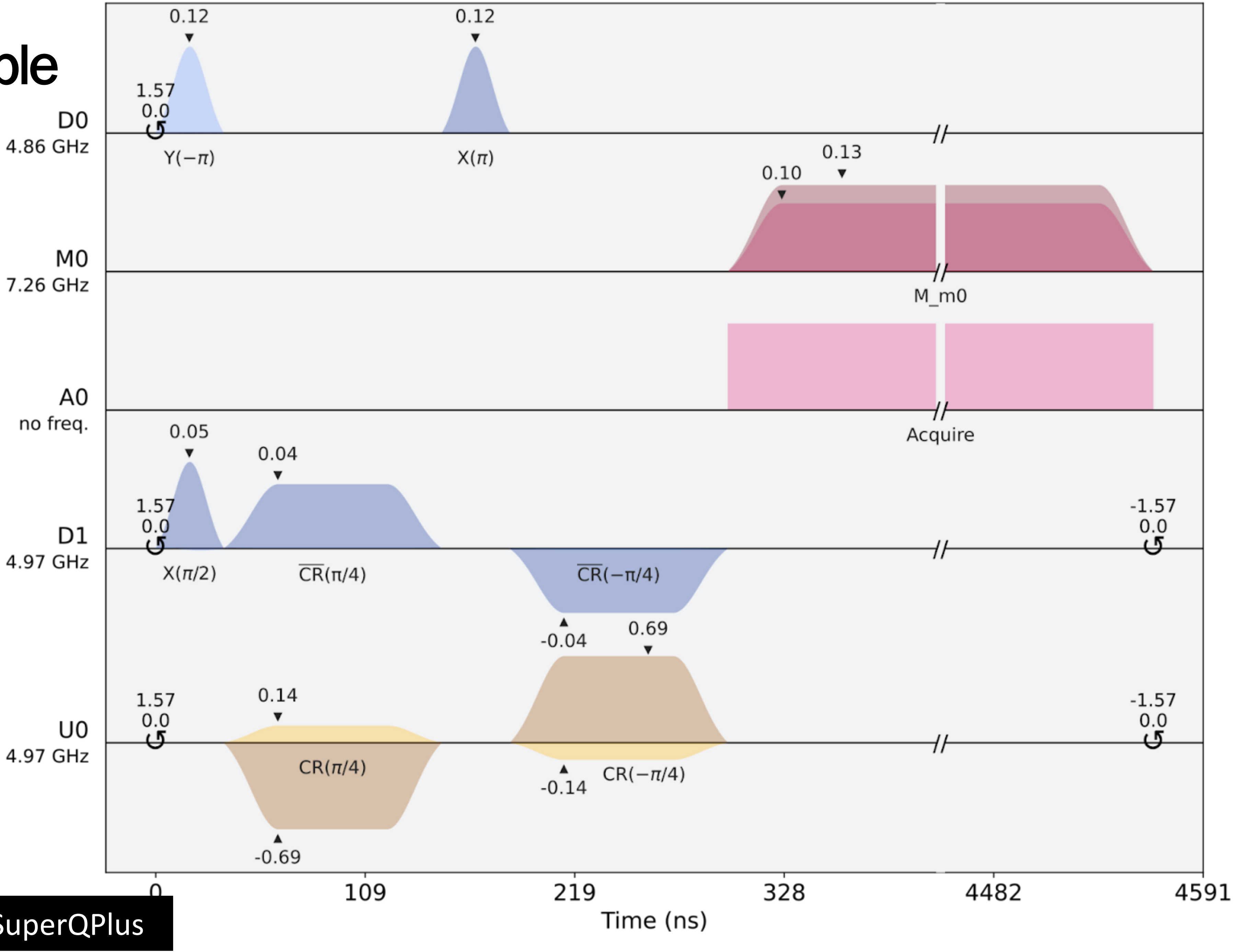
NordIQuEst

OpenSuperQPlus

# Pulse schedule example

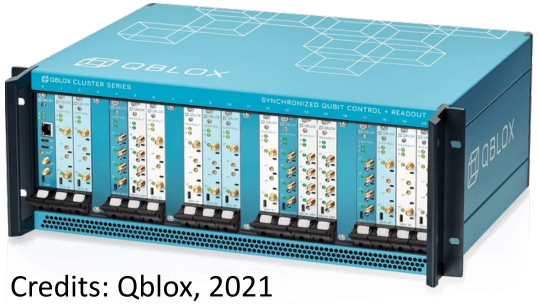






**WACQT** **NordIQuEst** **OpenSuperQPlus**

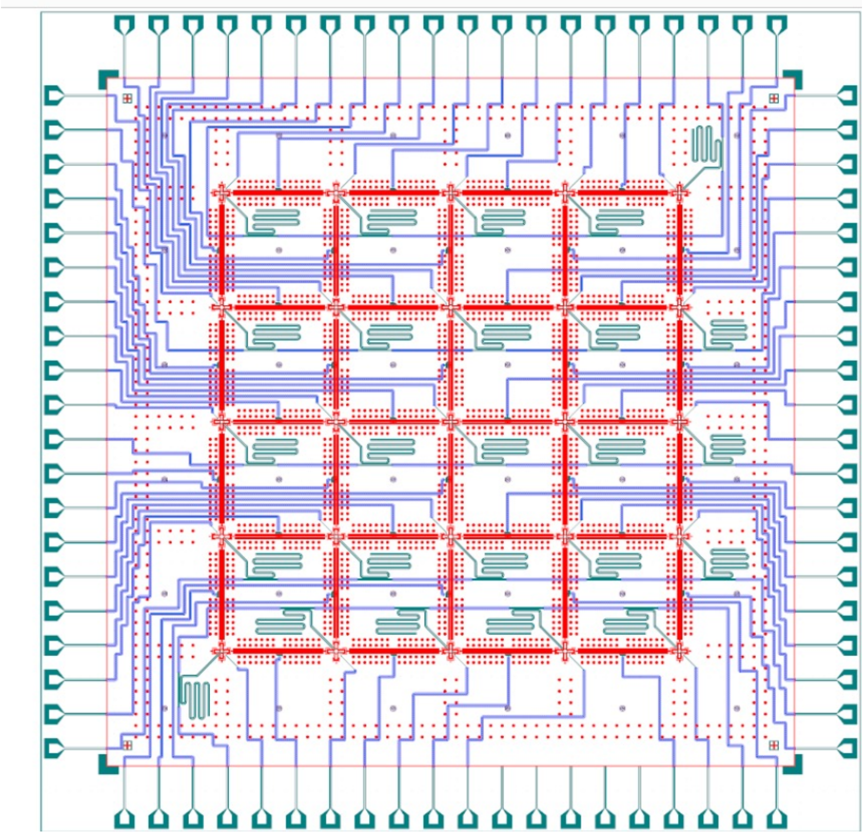
# Example: Qblox instruments assembly



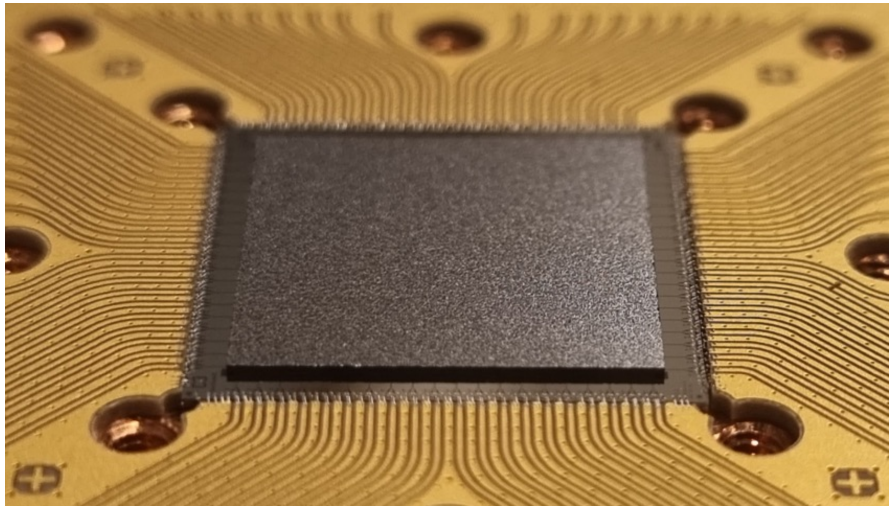
Credits: Qblox, 2021

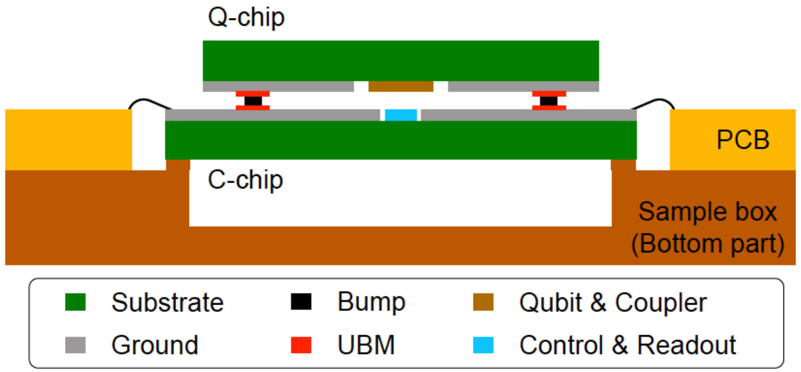
```
Q1ASM program:

0:              wait_sync         4
1:              upd_param         4
2:              set_mrk           15          # set markers to 15
3:              wait              4           # Latency correction of 0 ns.
4:              move              2000,R0     # iterator for loop with label start
5:        start:
6:              reset_ph
7:              upd_param         4
8:              wait              65532       # auto generated wait (300000 ns)
9:              wait              65532       # auto generated wait (300000 ns)
10:             wait              65532       # auto generated wait (300000 ns)
11:             wait              65532       # auto generated wait (300000 ns)
12:             wait              37872       # auto generated wait (300000 ns)
13:             set_awg_gain      851,0       # setting gain for gaussian-d1-0
14:             play              0,1,4       # play gaussian-d1-0 (100 ns)
15:             wait              96          # auto generated wait (96 ns)
16:             wait              4           # auto generated wait (4 ns)
17:             set_awg_gain      851,0       # setting gain for gaussian-d1-104
18:             play              0,1,4       # play gaussian-d1-104 (100 ns)
19:             wait              3596        # auto generated wait (3596 ns)
20:             loop              R0,@start
21:             set_mrk           0           # set markers to 0
22:             upd_param         4
23:             stop
```

# QPU chip



A layout of a 25 qubit processor developed at Chalmers.





generate ✓

optimize ✓

execution ✓

Control engineering

Qubit technology

WACQT

NordIQuEst

OpenSuperQPlus