Transpilation

IQM Winter Quantum School – Day 3, part 1

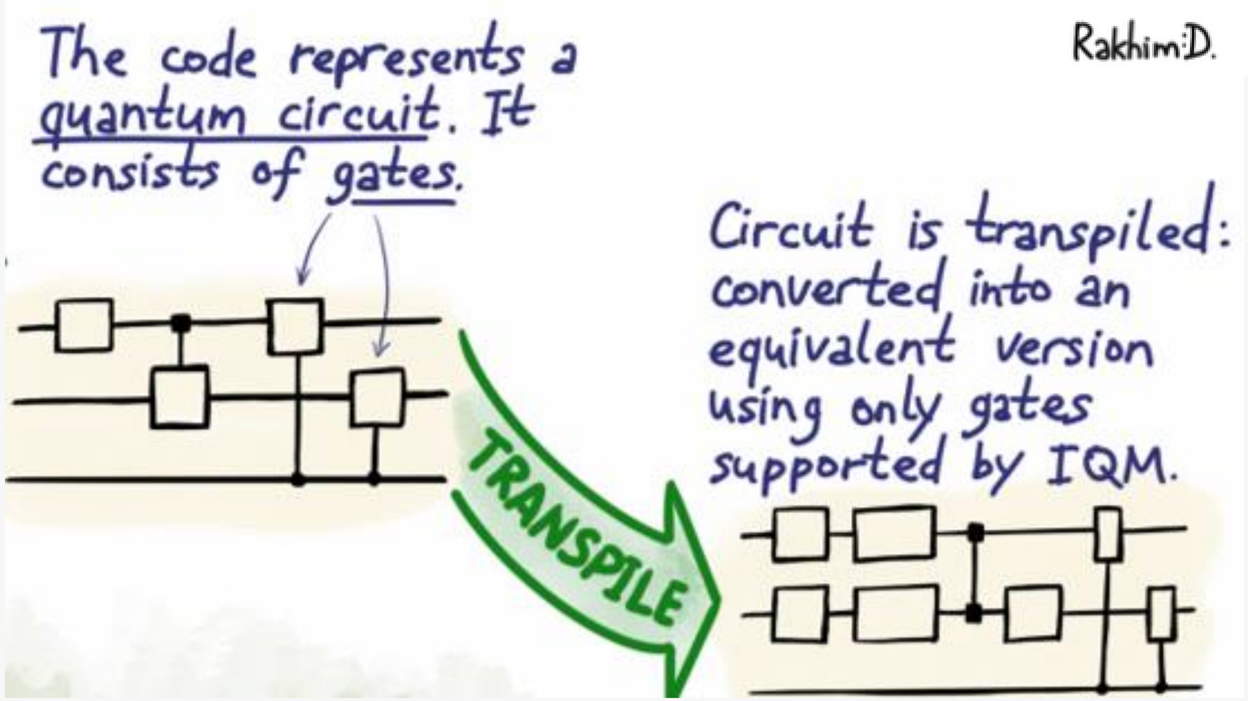
Authors: Daniel Bulmash, Stefan Seegerer, Nadia Milazzo

Last Updated 11/2025

www.meetiqm.com

What happens on a real quantum computer?

Transpilation

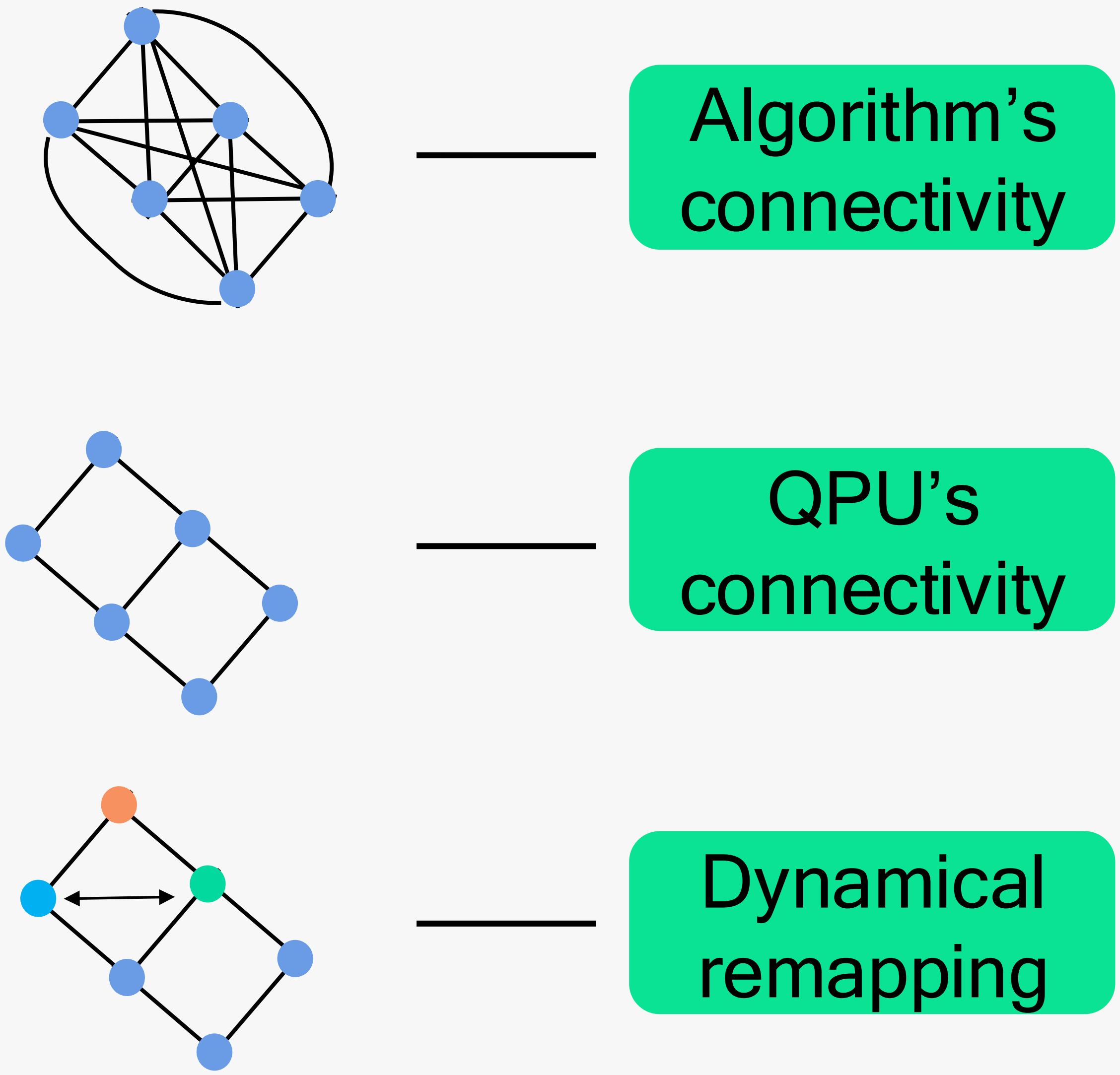

The code represents a quantum circuit. It consists of gates.
RakhimD.
TRANSPILE
Circuit is transpiled: converted into an equivalent version using only gates supported by IQM.

Transpiling describes the process of converting a quantum circuit into an equivalent quantum circuit that is compatible with a specific quantum hardware's constraints and gate set.

Circuit routing

Routing (= solving QPU connectivity constraints)

- Quantum algorithms assume arbitrary, even all-to-all connectivity
- Connectivities of QPU's are usually limited to nearest neighbor graphs
- Solution: dynamically remap **logical qubits** ("qubit 1" in the program") to the graph of **physical qubits** ("qubit 1" on the physical hardware) with SWAP gates



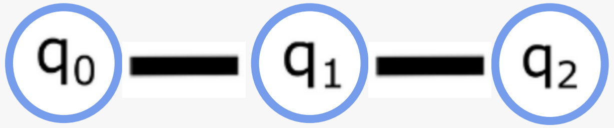
Algorithm's connectivity

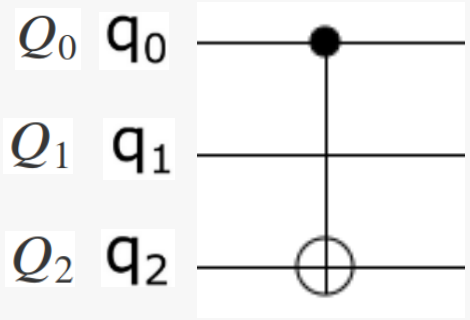
QPU's connectivity

Dynamical remapping

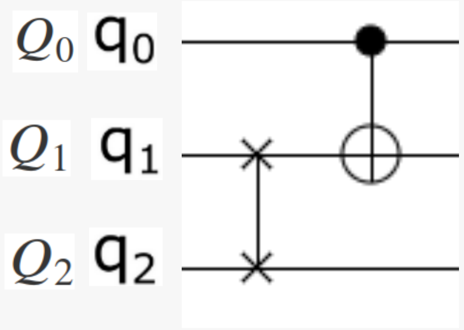
Routing example

Consider the following connectivity:

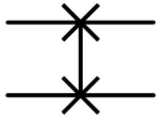




Routing incompatible operation

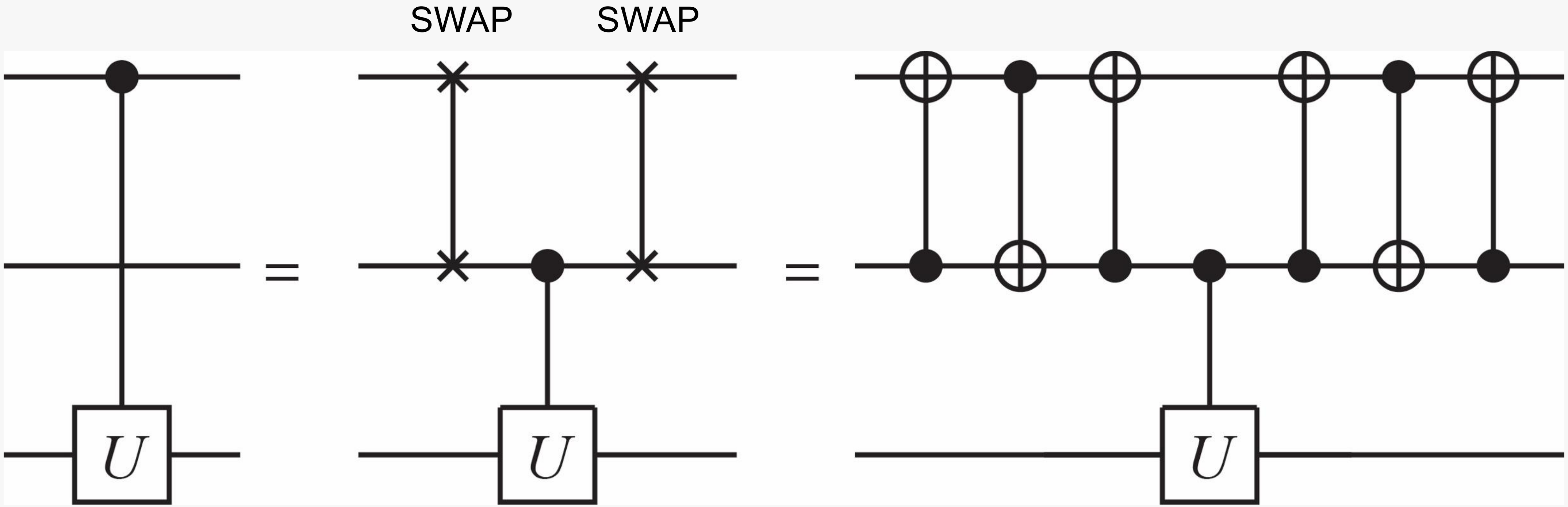


Routing compatible operation

$\mathrm{SWAP} =$  $= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

SWAP gate swaps the states of two qubits

SWAP is expensive!



IQM

Gate decomposition

Implementing gates on a QPU

- We need to be able to implement *any* unitary operation on our quantum computer. But not all operations are simple on all hardware.
- The set of simple gates used on a *specific* quantum device is called the native gate set of the device.
    - All operations must be rewritten as a sequence of native gates.

IQM systems support arbitrary X and Y rotations as the native single-qubit gates and CZ as the native two-qubit gate.

Exercise

- Imagine your QPU's only native gates are X gates on each qubit. Find a gate you can't implement on your QPU.

- Repeat the above if your native gate set also includes Y gates on each qubit.

- Repeat the above if your native gate set includes arbitrary rotations about the x and y axes on each qubit.

Universal gate set

- We need to be able to implement *any* unitary operation on our quantum computer. But we need enough native gates to do that.

- Our native gate set should be universal. What does that mean?

A set of gates S is said to be universal if **any** unitary operation can be approximated to any desired precision by composing only the gates in the set S.

Universal gate set

A universal set of gates for quantum computing can:

- Create superposition of states
- Create entangled states
- Contains gates with real AND complex entries
- Contains more than the "Clifford group"

Special quantum circuits which can be efficiently simulated by a classical computer. To read more: "Gottesman-Knill theorem"

Universal gate set

Is there only one universal set of gates? If so, we'd be forced to use hardware with a very specific set of native gates…

No, there exist different universal gate sets, such as:

$\boxed{\{CNOT, H, S, T\}}$

$S =$ *phase gate*
$T = \pi/8$ *gate*

$\boxed{\left\{CZ, R_x(\theta), R_y(\phi)\right\}}$

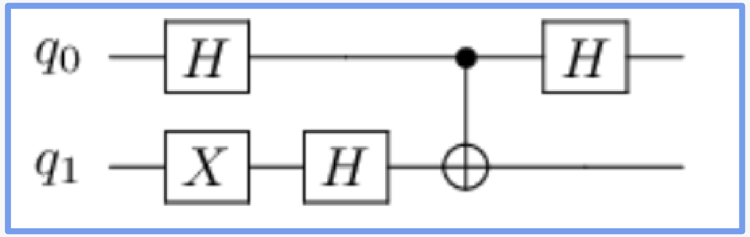
$\boxed{\{H, TOFF\}}$

$TOFF =$ *"Toffoli gate"*

Decomposing (= writing a gate a series of native gates)

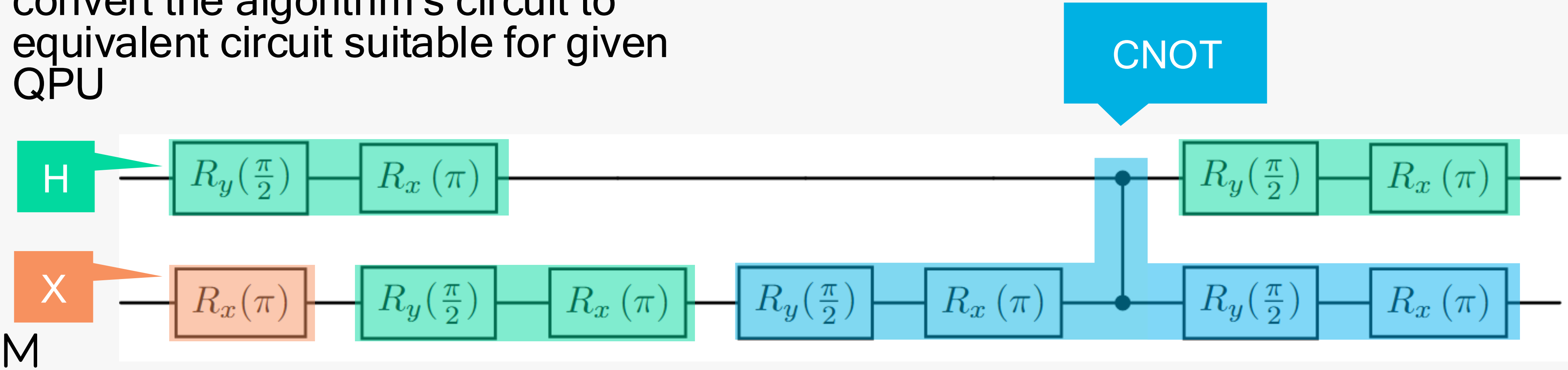
- Circuits of quantum algorithms usually use ’intuitive’ gates like CNOT, X, H, P, SWAP
- Real QPU’s use only few *native,* hardware friendly gates to process their qubits
- Solution: gate identities can convert the algorithm’s circuit to equivalent circuit suitable for given QPU

Circuit for Deutsch’s algorithm with CNOT as the oracle



Example QPU native gates

$\{R_x(\theta), R_y(\phi), CZ\}$



Redefining more precisely…

Transpiling = routing circuit to connectivity of a QPU and decomposing gates to its native gates

Day 3, part 1 lab

Find the Day 3, part 1 lab here

https://tinyurl.com/iqm-transpile

Please don’t worry about debugging everything during the lab session! It will be more useful to follow along and debug later on your own.

Day 3, part 2 – From Abstract Code to Physical Pulses