CPEN 400Q
Lecture 11: an overview of quantum compilation

Monday 12 February 2024

Announcements

- Quiz 4 today
- Literacy Assignment 1 due Tuesday at 23:59
- Assignment 2 due Thursday at 23:59
- Project details available on PrairieLearn
- Wednesday is hands-on class

Final project details

Implement the methods of a recent research paper and reproduce the results as closely as possible.

Four parts:

- (15%) Midterm checkpoint (write up + short meeting)
- (40%) Companion report
- (40%) Software implementation
- (5%) Weekly peer assessment surveys

Work in groups of 4.

Final project: important dates

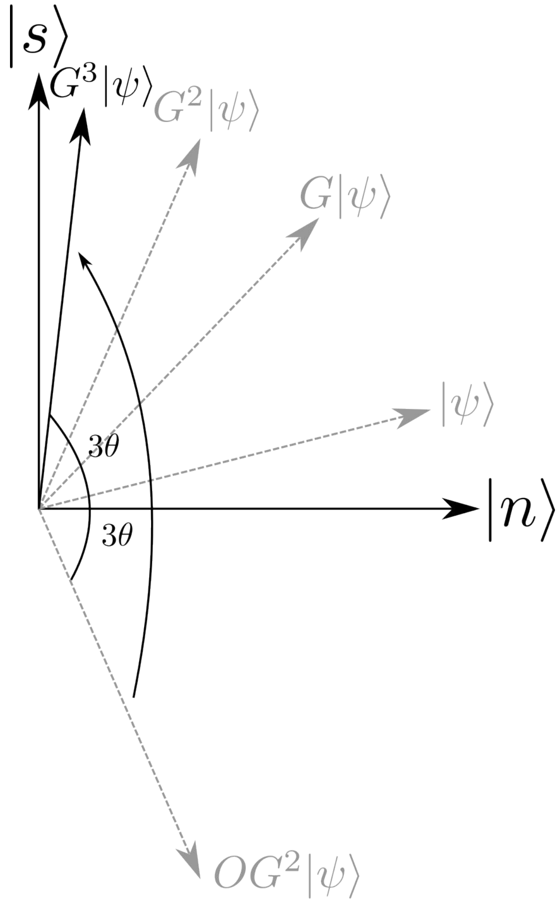
- **2024-02-16**: Group and topic selection due.
- **2024-03-01**: First weekly survey due
- **2024-03-06 – 2024-03-08**: Midterm checkpoint
- **2024-04-12**: Final report and software implementation due.

Last time

We discussed how Grover search of an unstructured space has a square-root speedup in **query complexity** over classical methods.

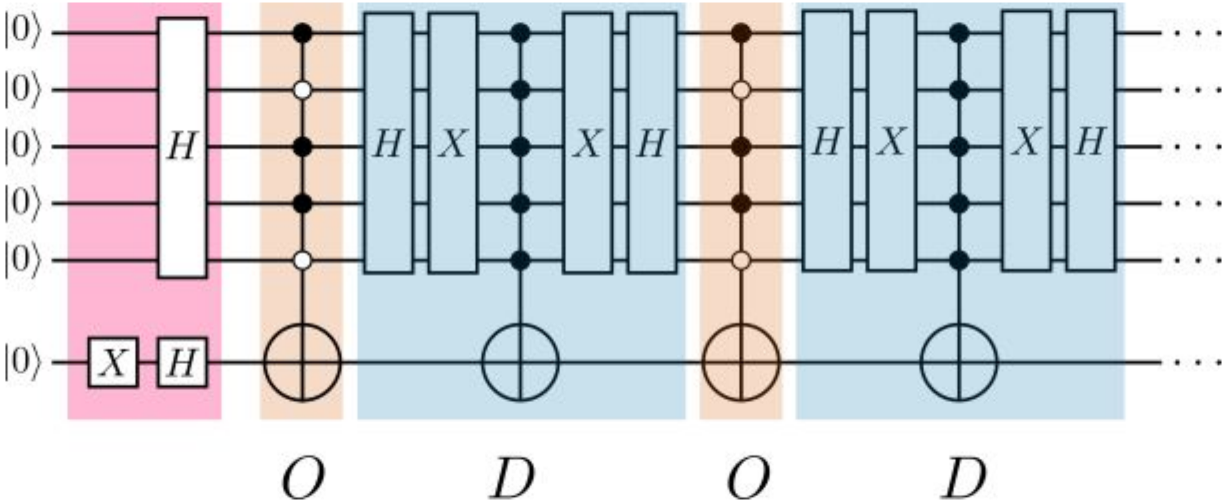
classical: $O(2^n)$

quantum: $O(\sqrt{2^n})$



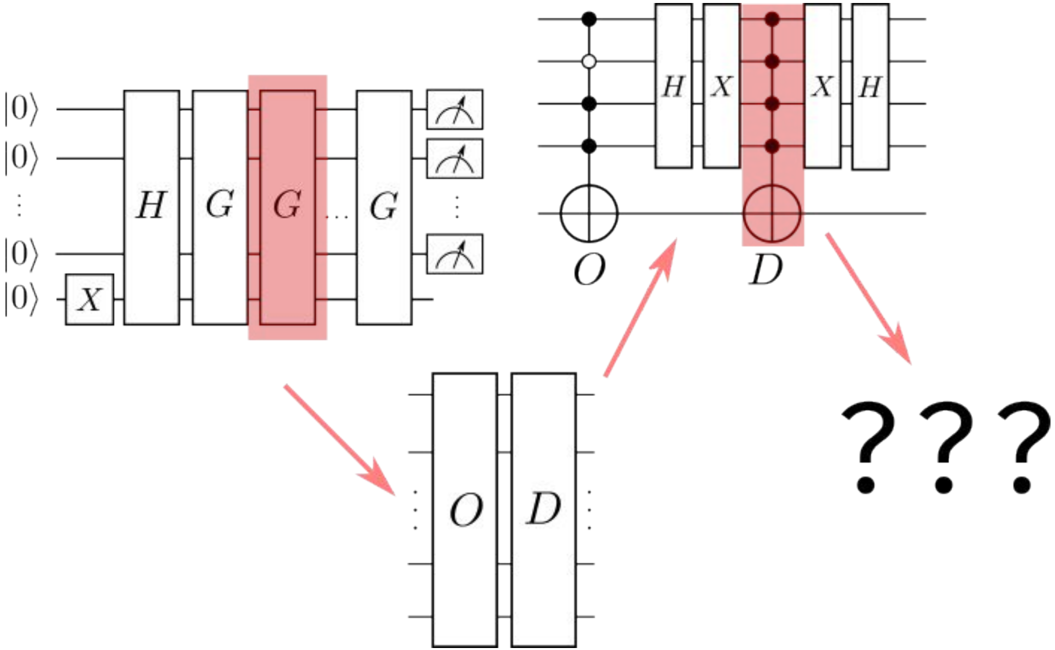
Last time

We implemented Grover’s algorithm in PennyLane.



Last time

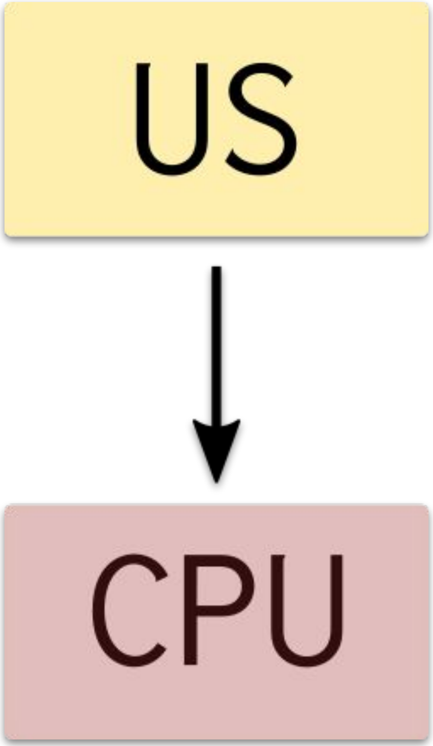
We finished off with this picture:



Today

Learning outcomes:

- explain the limitations of query complexity as a metric
- estimate the actual resources (width, depth, gate count) required to run quantum circuits
- define what it means for a gate set to be universal, and list some key sets
- draw and identify the different parts of the compilation stack


US
CPU

US

High-level algorithmic description

Software
(Human-readable code)

CPU

Compilers are **essential**

```
#include<stdio.h>

void main() {
    printf("Hello, world!");
}
```

Compilers are **essential**

```c
#include<stdio.h>

void main() {
    printf("Hello, world!");
}
```

```
        .file   "hello_world.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9
```

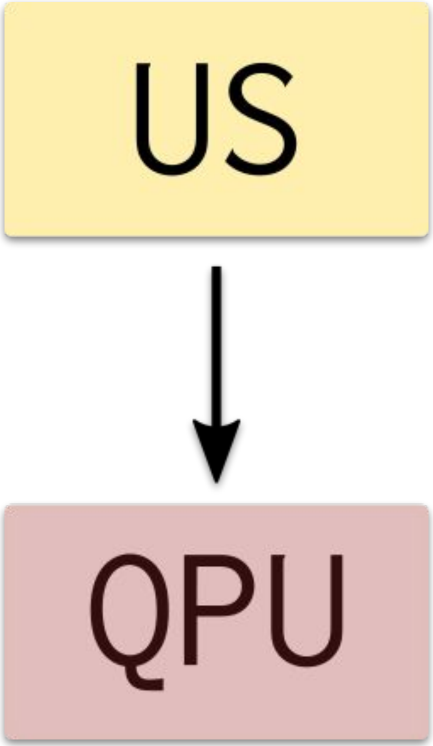
Compilers are essential

```c
#include<stdio.h>

void main() {
    printf("Hello, world!");
}
```

```
        .file   "hello_world.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.2.1-9ubuntu2) 9
```

```
 11a0 ff48c1fd 03741f31 db0f1f80 00000000  .H...t.1........
 11b0 4c89f24c 89ee4489 e741ff14 df4883c3  L..L..D..A...H..
 11c0 014839dd 75ea4883 c4085b5d 415c415d  .H9.u.H...[]A\A]
 11d0 415e415f c366662e 0f1f8400 00000000  A^A_.ff.........
 11e0 f30f1efa c3                          .....
Contents of section .fini:
 11e8 f30f1efa 4883ec08 4883c408 c3        ....H...H....
Contents of section .rodata:
 2000 01000200 48656c6c 6f2c2077 6f726c64  ....Hello, world
 2010 2100                                 !.
Contents of section .eh_frame_hdr:
 2014 011b033b 40000000 07000000 0cf0ffff  ...;@...........
 2024 74000000 2cf0ffff 9c000000 3cf0ffff  t...,.......<...
 2034 b4000000 4cf0ffff 5c000000 35f1ffff  ....L...\...5...
 2044 cc000000 5cf1ffff ec000000 ccf1ffff  ....\...........
 2054 34010000                             4...
Contents of section .eh_frame:
```


US
QPU

In an ideal world…

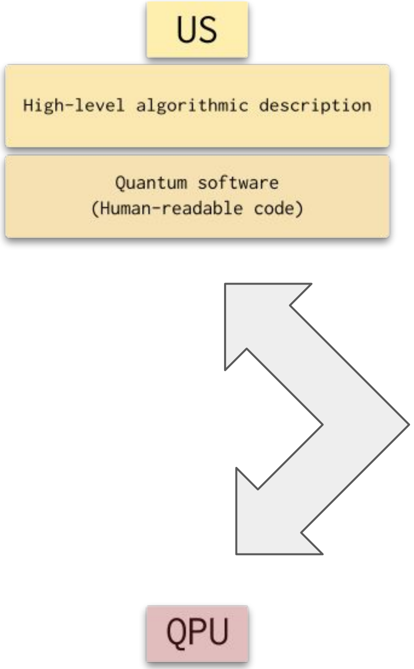
US

High-level algorithmic description

Quantum software
(Human-readable code)

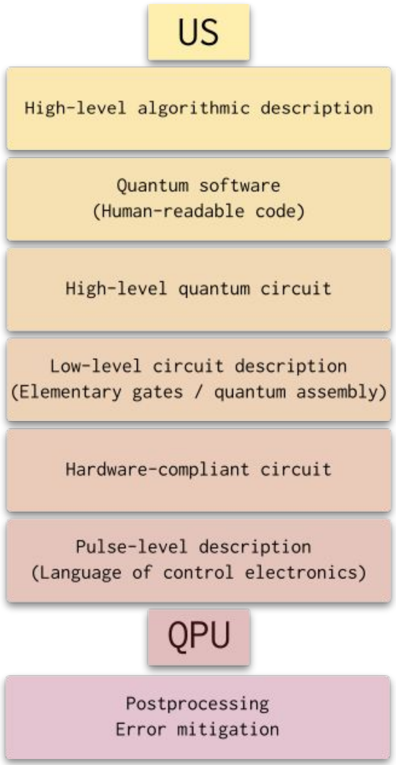
QPU

This lecture

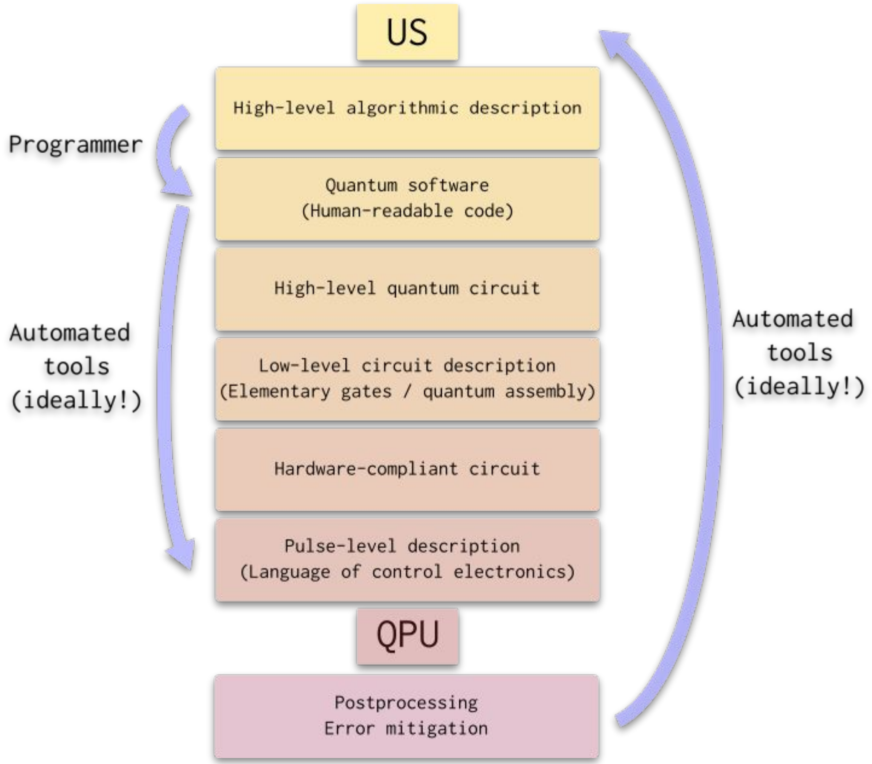


What happens in between?

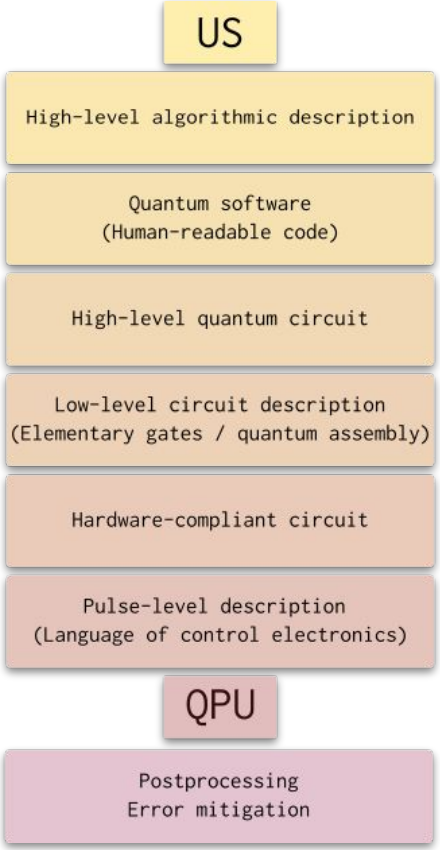
The quantum compilation stack

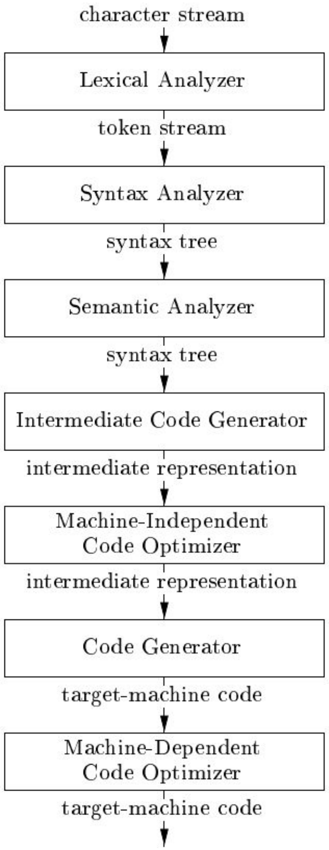


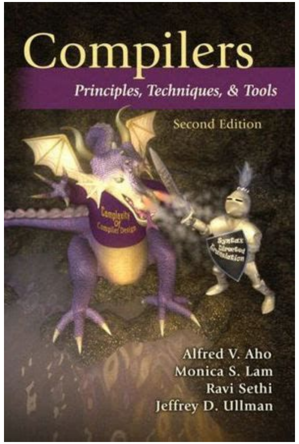
The quantum compilation stack



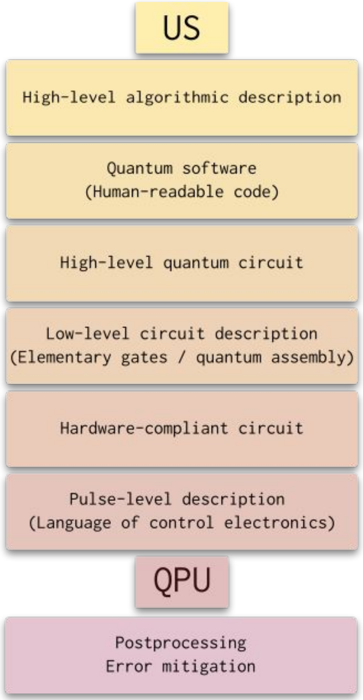
The quantum compilation stack







Compilation

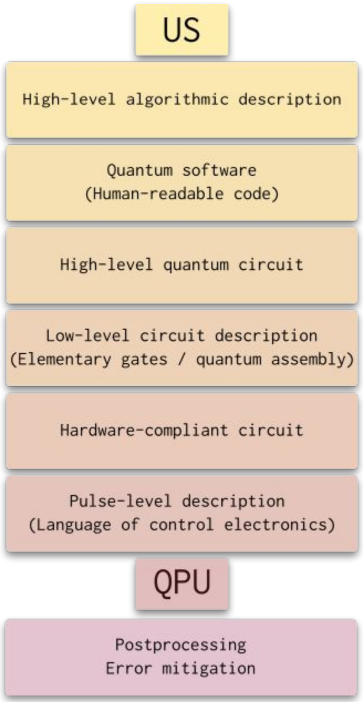


Implemented as a sequence of modular ***passes***, or ***transforms***, that modify a circuit in various ways.

Loosely divided into 3 stages:

- decomposition and synthesis
- hardware-independent optimization
- hardware-dependent optimization

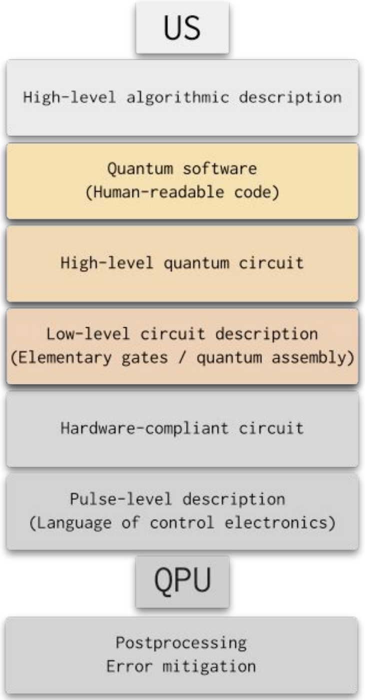
Compilation: key metrics



We want the circuits we implement to be **as efficient as possible** with respect to:

- Circuit width (number of qubits)
- Circuit depth (length of longest path)
- 1- and 2-qubit gate count

Circuit synthesis



Given:

- arbitrary unitary matrix U
- a finite set of gates $\{G_k\}$
- a target precision, ϵ

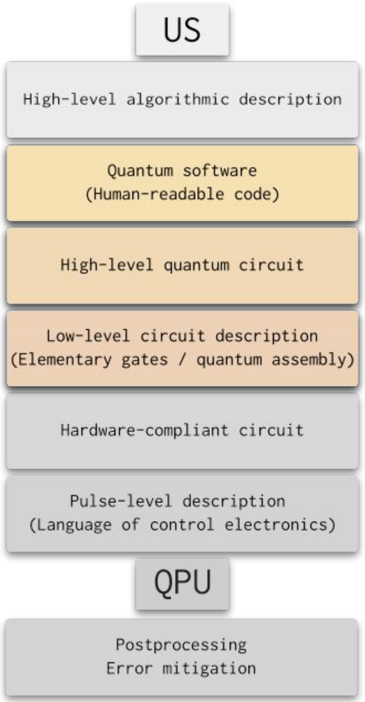
Find

$$U' = C_1 C_2 \cdots C_m$$

s.t.

$$||U - U'|| < \epsilon, C_i \in \{G_k\}$$

Circuit synthesis: universal gate sets



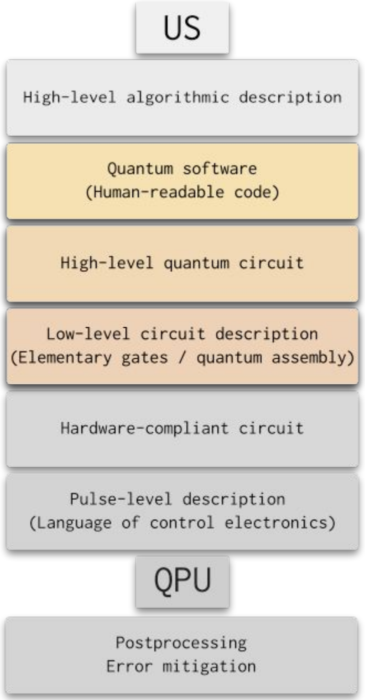
A $\{G_k\}$ for which this can be done is called a **universal gate set**.

For a single qubit:

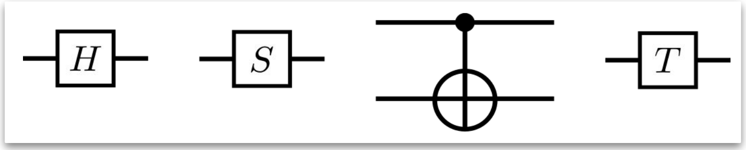
- Any two Pauli rotations (exact)
- Hadamard and T (approximate)

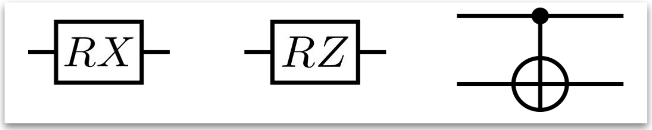
Theoretical bound: $O(\log^c(1/\epsilon))$ gates, c varies from 1-4 depending on assumptions.

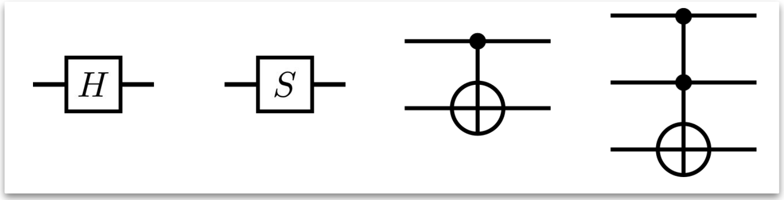
Circuit synthesis: universal gate sets



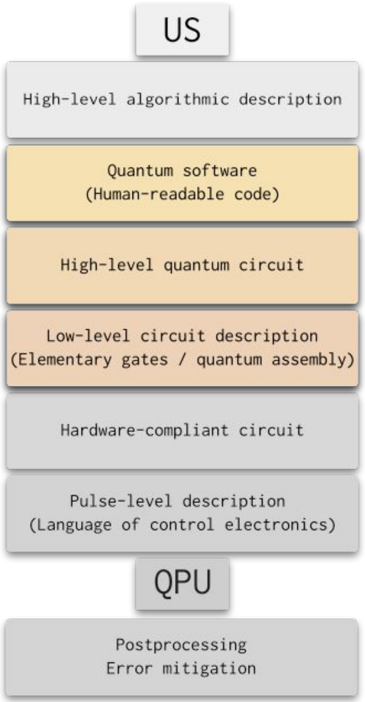
Multi-qubit gate sets:







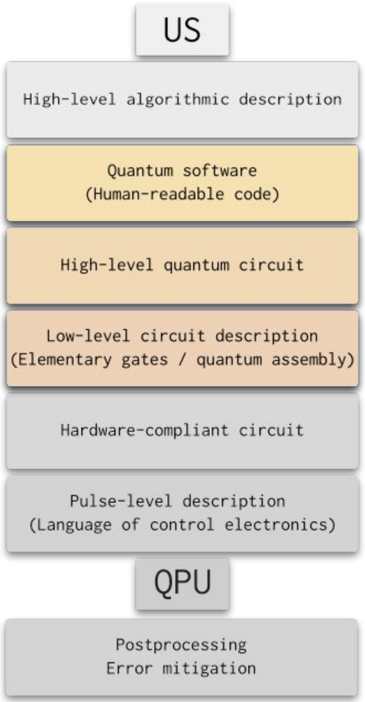
Circuit synthesis: universal gate sets



Working with a different gate set is like *speaking a different language*.

A	I would like to use a real quantum computer.	*I would like to use a real quantum computer.*
R	J’aimerais utiliser un vrai ordinateur quantique.	*I would like to use a real computer that is quantum.*
P	本当の量子コンピュータを作りたい。	*A real quantum computer, I would like to use.*

Circuit synthesis and decomposition



Exercise: Express Pauli *X* in terms of *H* and *T*.

Hint: start by expressing it using *H* and *Z*.

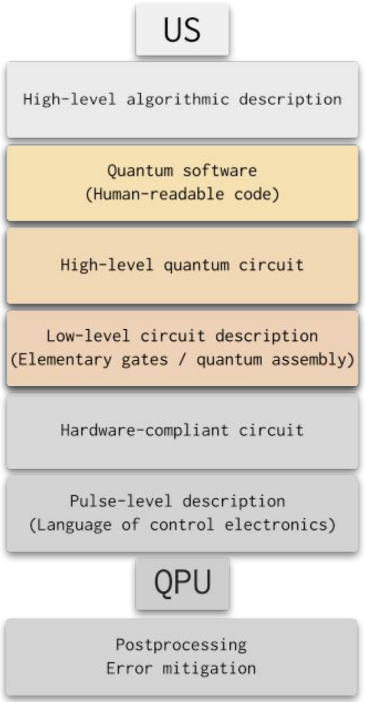
Step 1: $-\boxed{X}- \; = \; -\boxed{H}-\boxed{Z}-\boxed{H}-$

Step 2: $= \; -\boxed{H}-\boxed{T}-\boxed{T}-\boxed{T}-\boxed{T}-\boxed{H}-$

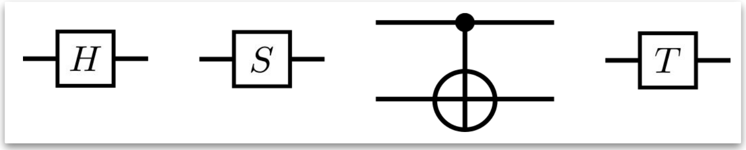
QASM

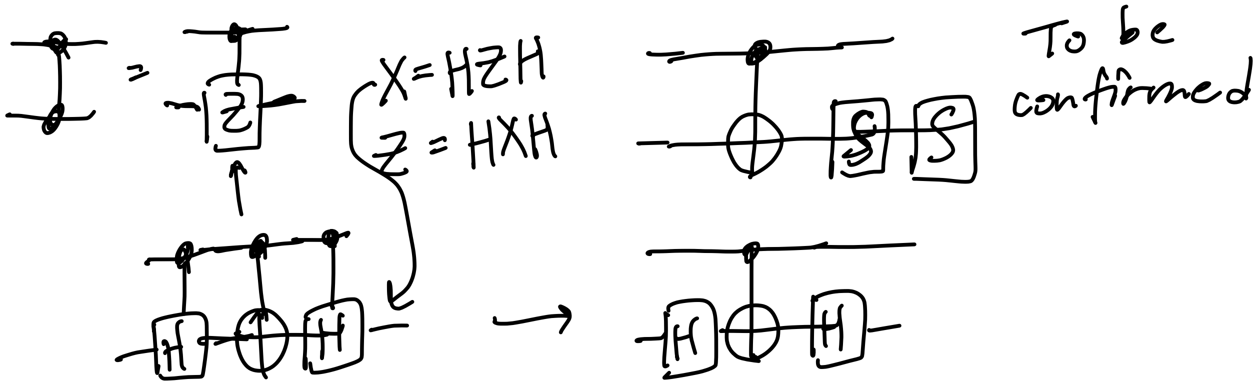
$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

Circuit synthesis and decomposition

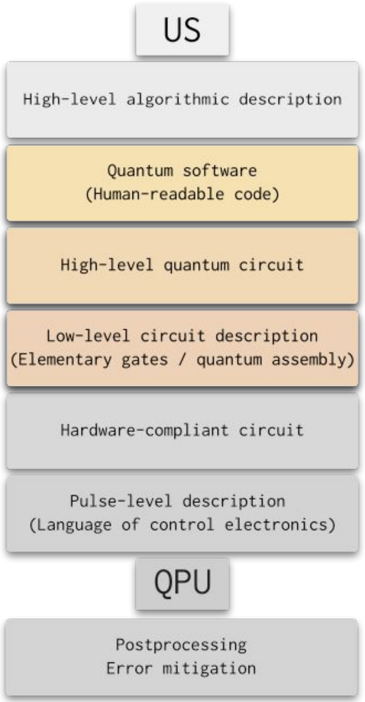


Exercise: Express the *CZ* gate in the set

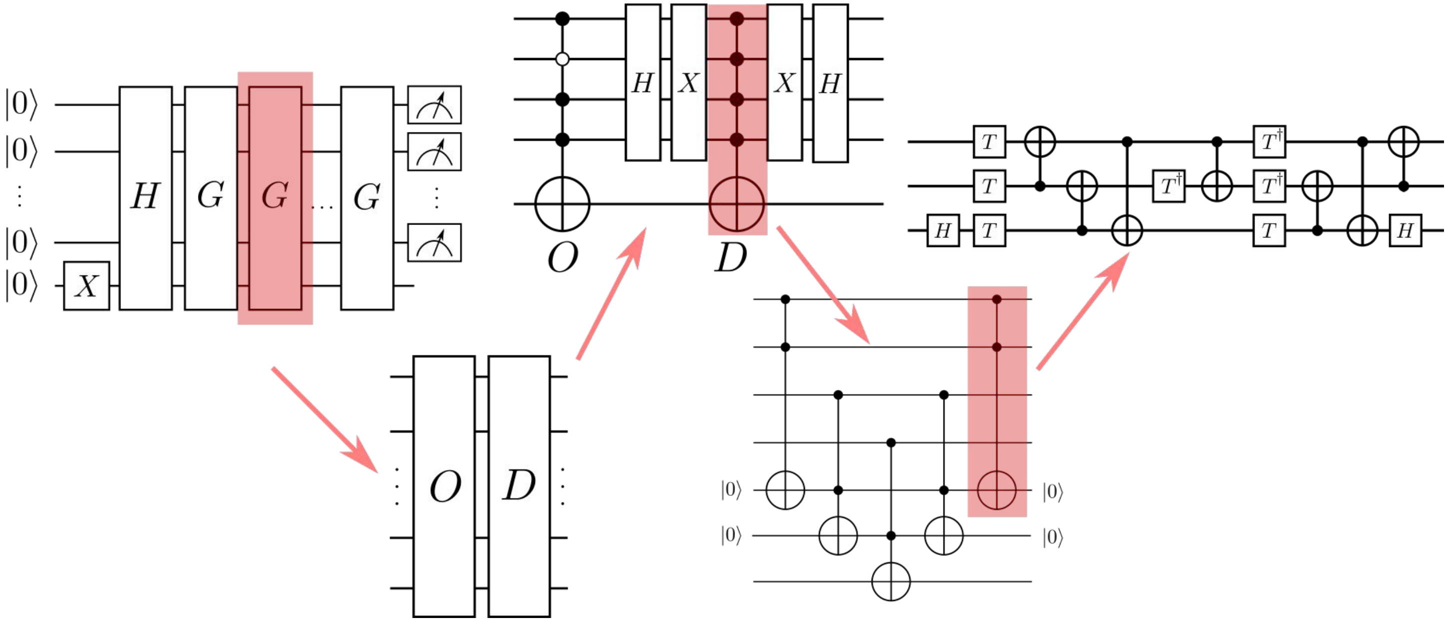




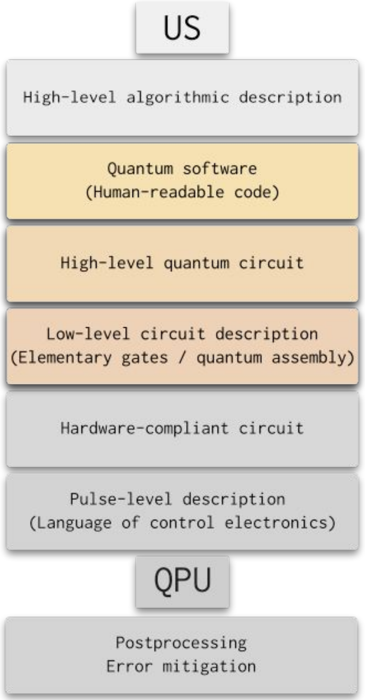
Circuit synthesis and decomposition

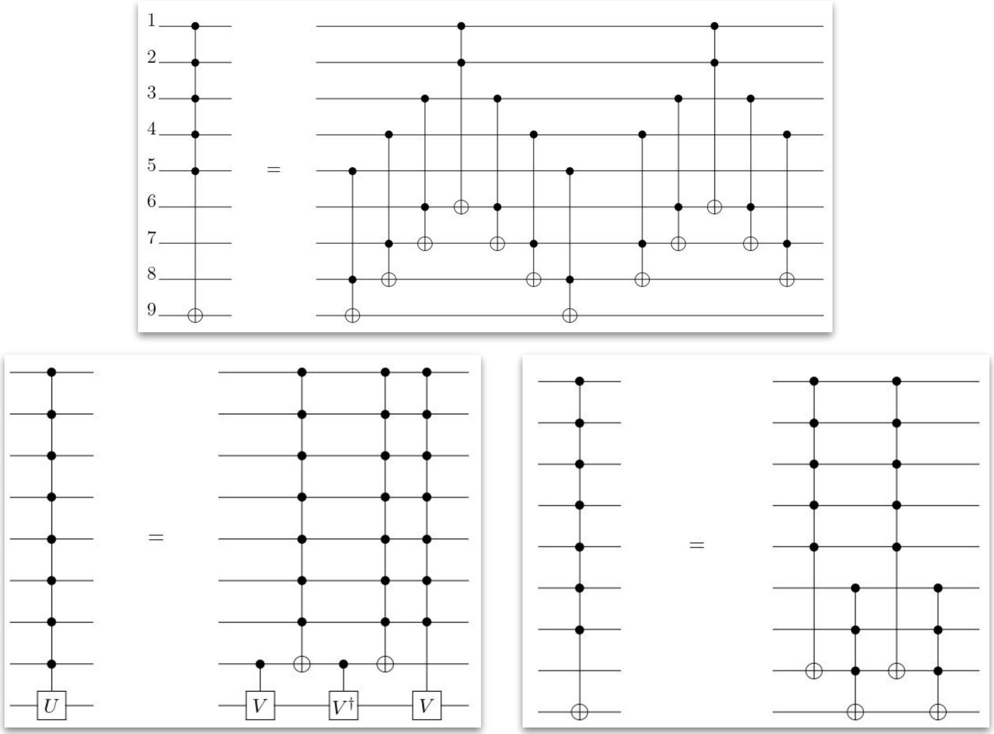


Decompositions of many common operations are **known**.



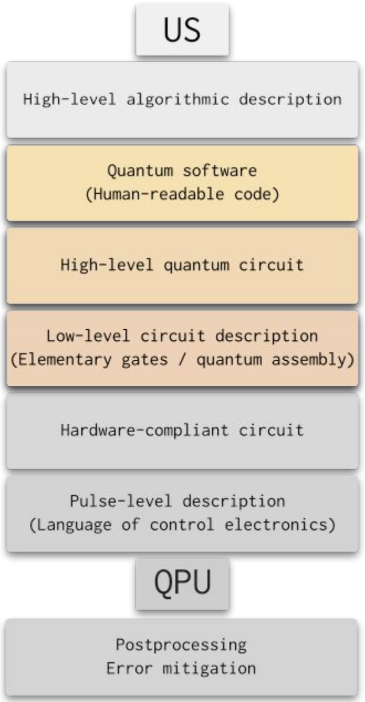
Circuit synthesis and decomposition





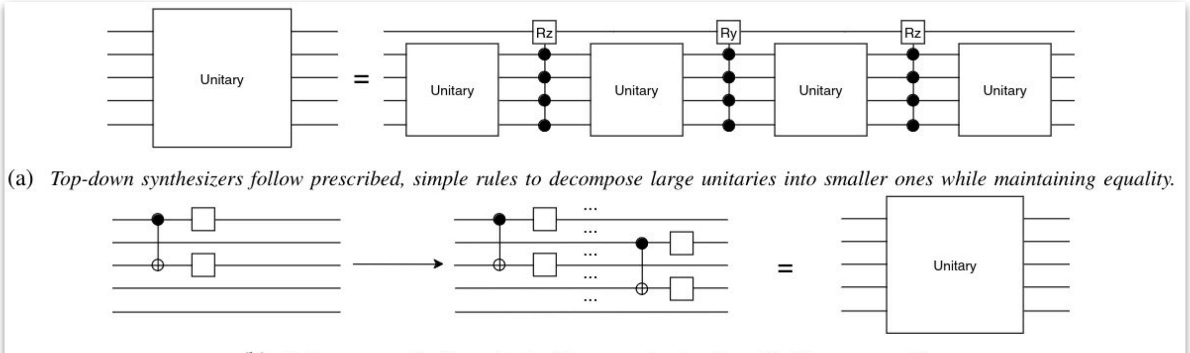
Images: Barenco et al., *Elementary gates for quantum computation* Phys.Rev. A 52 (1995) 3457

Circuit synthesis and decomposition



For arbitrary operations, or special gate sets, we may need to ***find*** a decomposition.

- search problem
- number-theoretic techniques



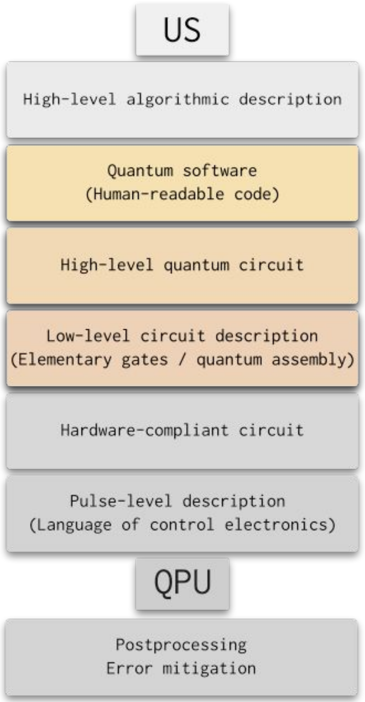
(a) *Top-down synthesizers follow prescribed, simple rules to decompose large unitaries into smaller ones while maintaining equality.*

(b) *Bottom-up synthesizers start with an empty circuit and build up to equality.*

Figure 1: *Quantum synthesizers are either top-down or bottom-up.*

Image: E. Younis, K. Sen, K. Yelick, and C. Iancu. *QFAST: Conflating Search and Numerical Optimization for Scalable Quantum Circuit Synthesis.* arXiv:2103.07093 [quant-ph]

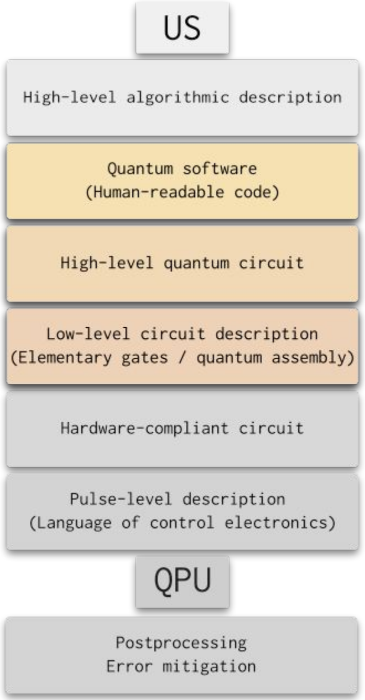
Circuit synthesis and decomposition



Many challenges:

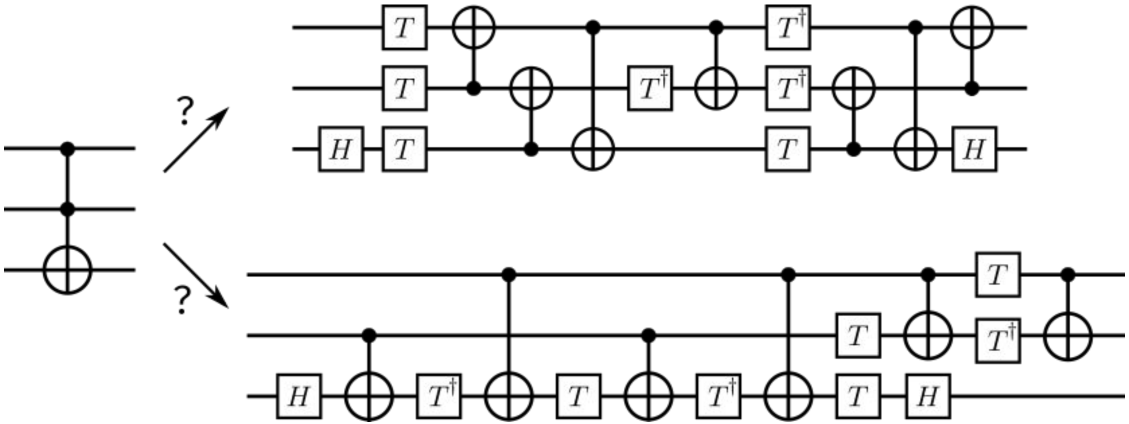
- algorithm complexity and computational resources

Circuit synthesis and decomposition

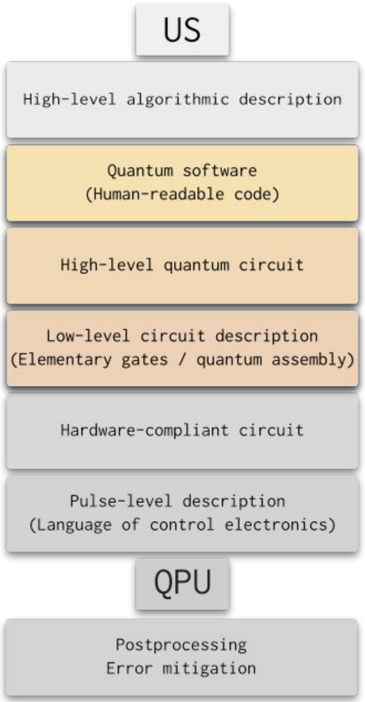


Many challenges:

- algorithm complexity and computational resources
- how to choose the best decomposition to *enable further optimization* down the line?



Circuit synthesis and decomposition



Many challenges:

- algorithm complexity and computational resources
- how to choose the best decomposition to *enable further optimization* down the line?
- manage tradeoffs between various resources

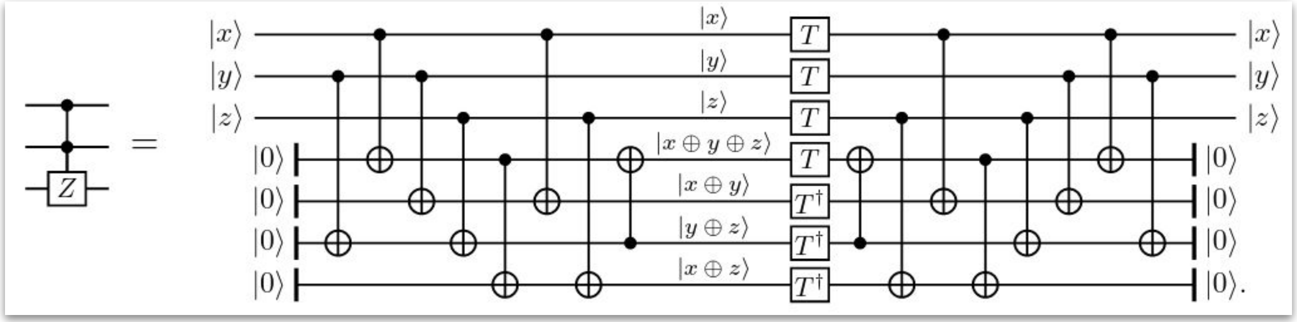
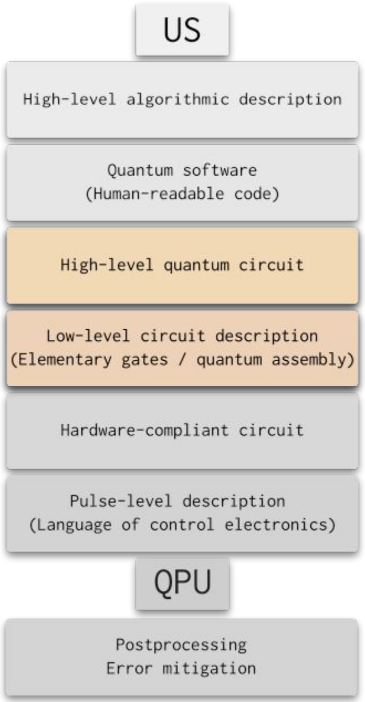


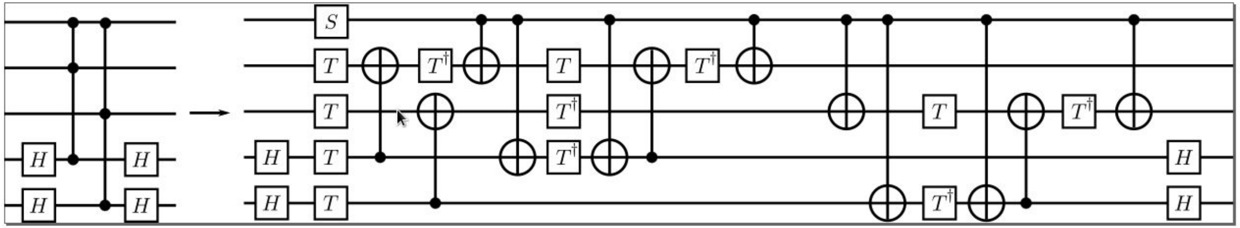
Image: P. Selinger (2013) *Quantum circuits of T -depth one*. Phys. Rev. A 87, 042302 (2013)

Circuit optimization

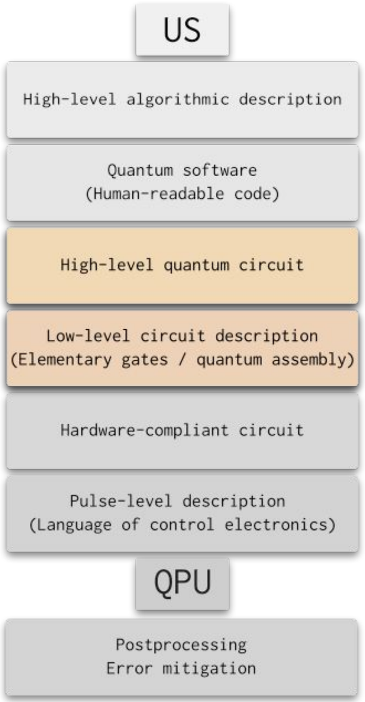


Machine-independent (or dependent) sequence of *passes* to reduce:

- gate count (overall, or for particular gate)
- circuit depth
- qubit count



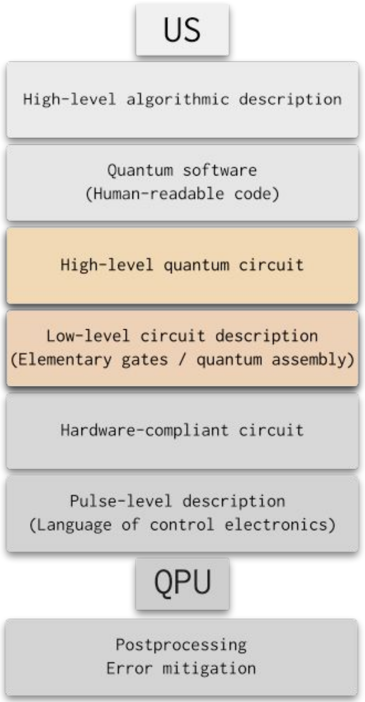
Circuit optimization



Passes have varying levels of complexity:

- inverse cancellation

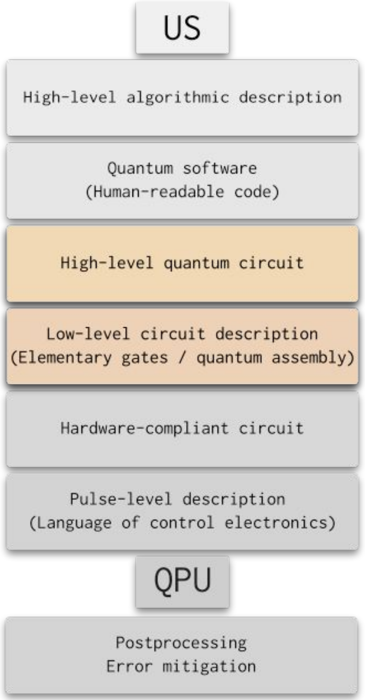
Circuit optimization



Passes have varying levels of complexity:

- inverse cancellation
- rotation merging

Circuit optimization



Passes have varying levels of complexity:

- inverse cancellation
- rotation merging
- template matching

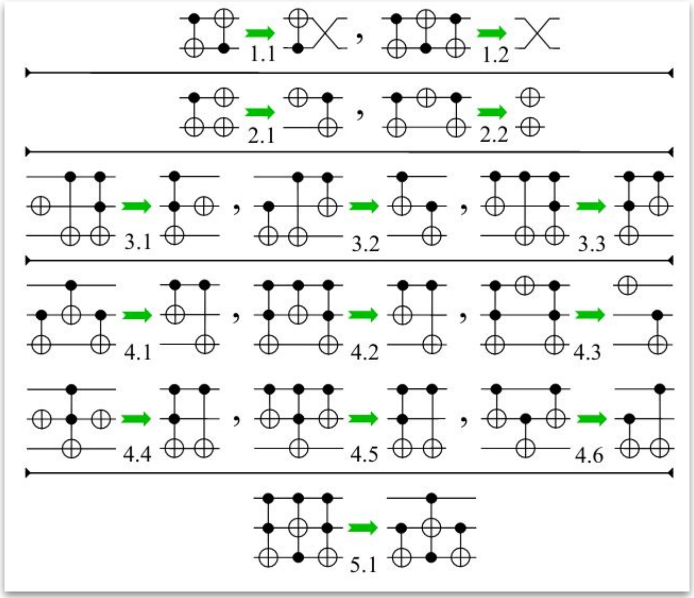
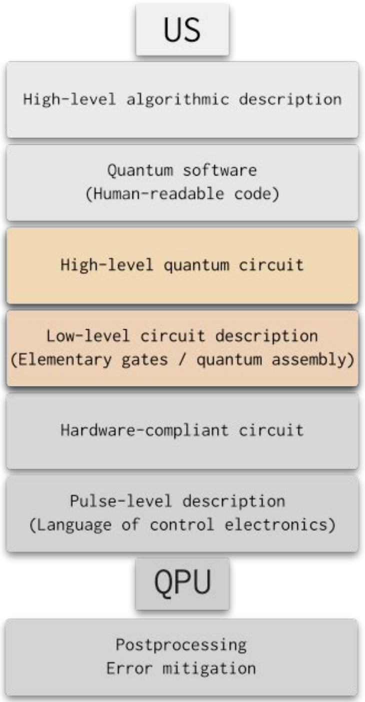


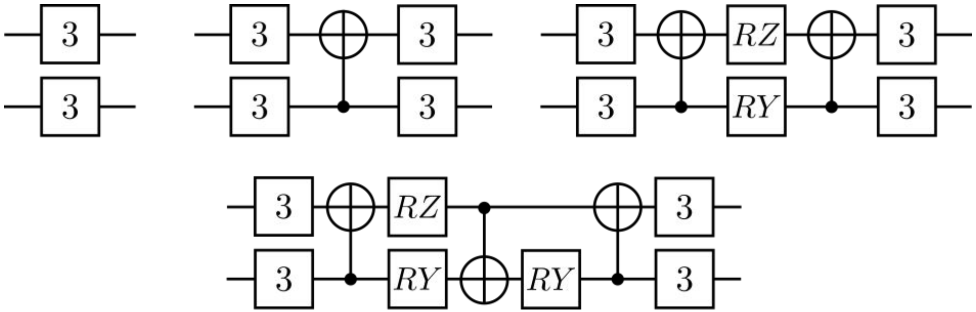
Image: D. M. Miller, D. Maslov, G. W. Dueck.
A Transformation Based Algorithm for Reversible Logic synthesis. DAC 2003.

Circuit optimization



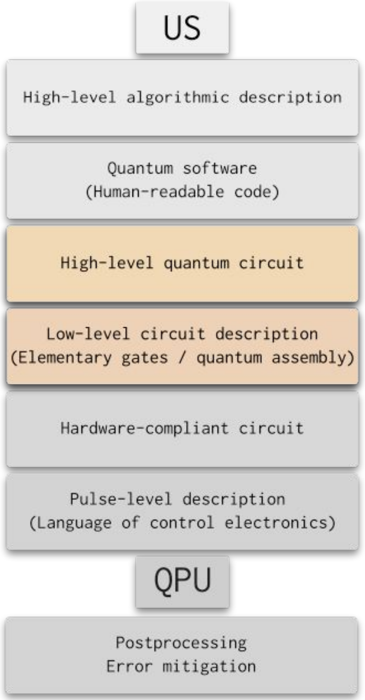
Passes have varying levels of complexity:

- inverse cancellation
- rotation merging
- template matching
- two-qubit block *resynthesis*



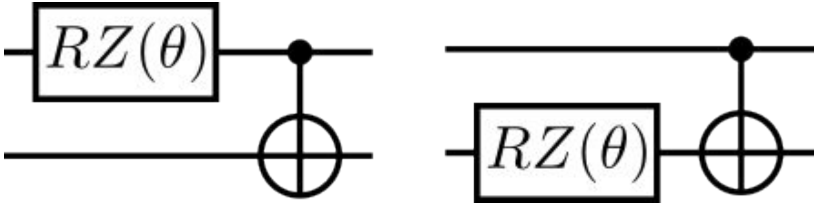
Reference: V. V. Shende, I. L. Markov, and S. S. Bullock (2004) *Minimal universal two-qubit controlled-NOT-based circuits*. Phys. Rev. A 69 062321.

Circuit optimization: commutation

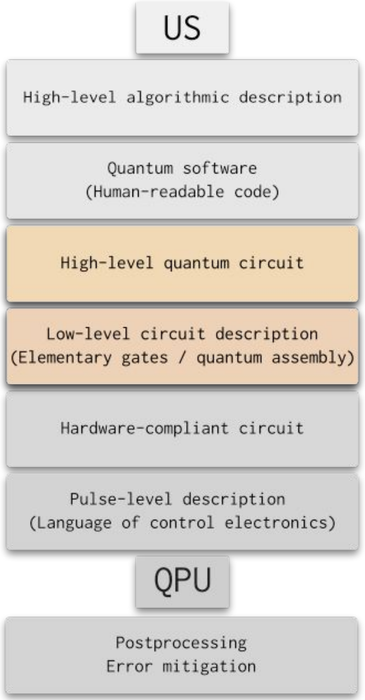


Commutation relations can help us move gates "through" each other to enable optimizations.

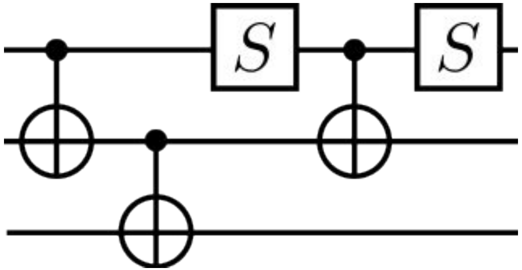
Exercise: do the following gates commute?



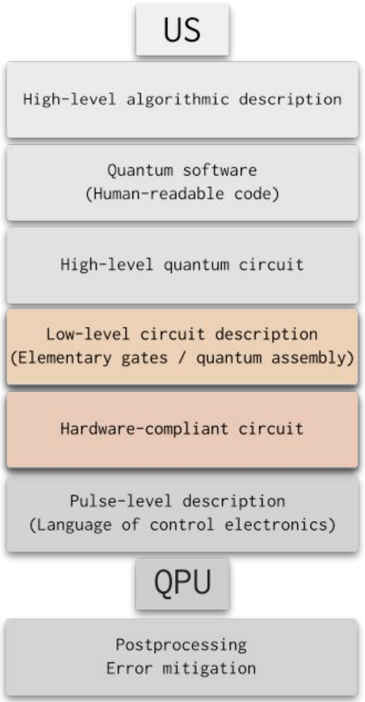
Circuit optimization: commutation



Exercise: optimize this circuit. What is the minimum depth?

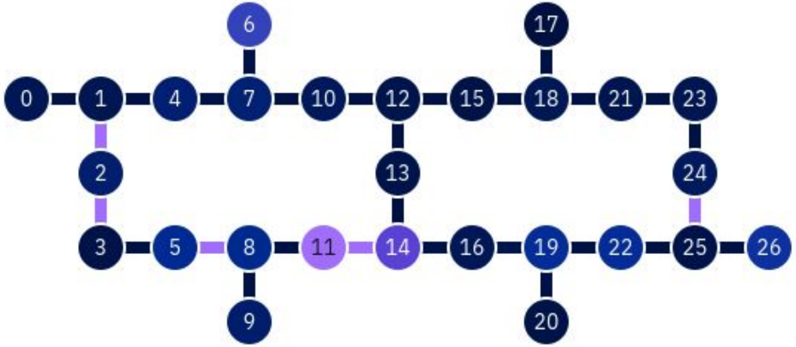


Qubit placement and routing



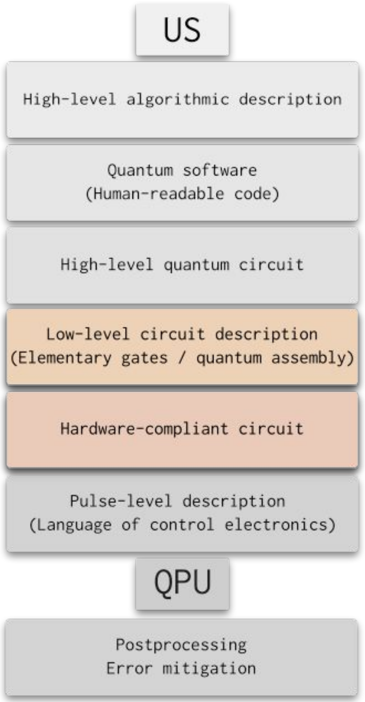
Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology



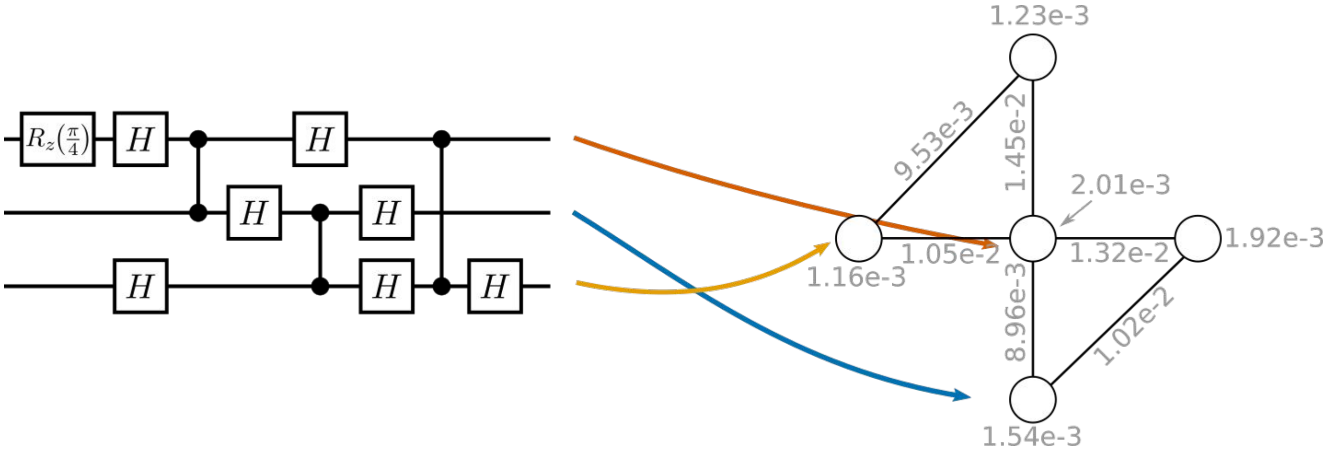
IBM Q Kolkata — screencap 2023-08-24

Qubit placement and routing

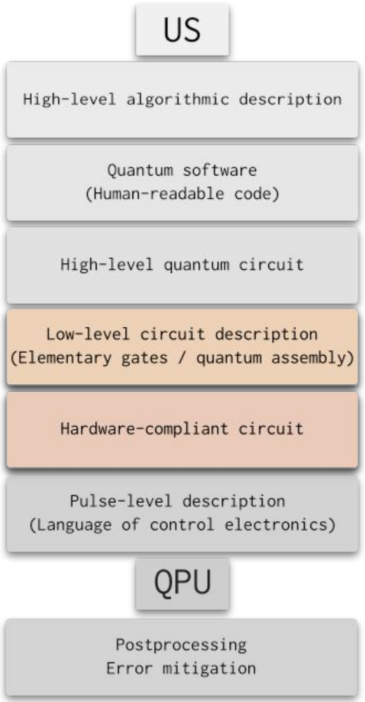


Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology

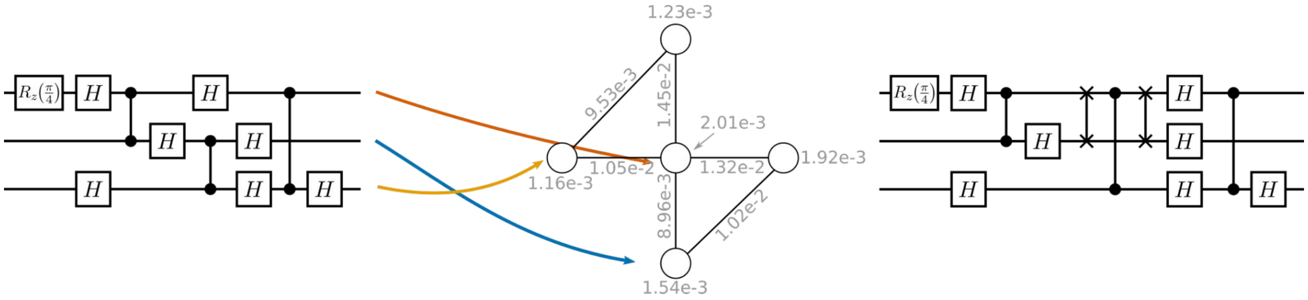


Qubit placement and routing



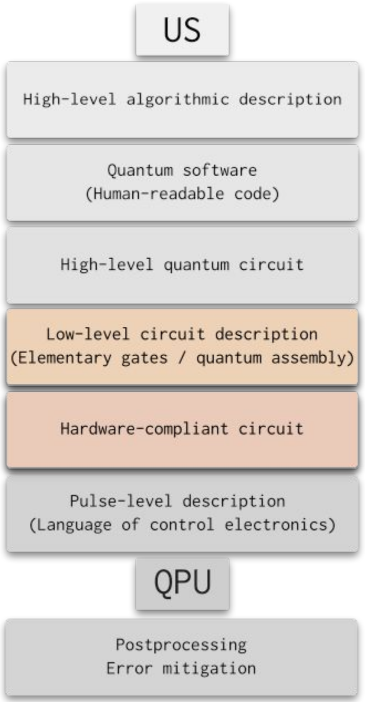
Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology



Placement is NP-complete; routing is NP hard.

Qubit placement and routing



Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology
- neutral atom machines can lose atoms

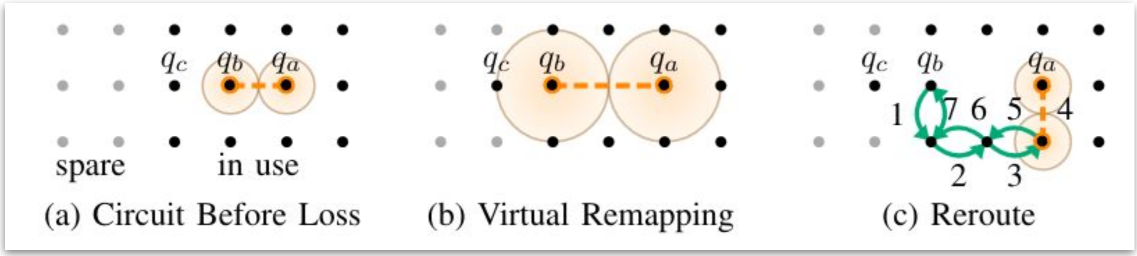
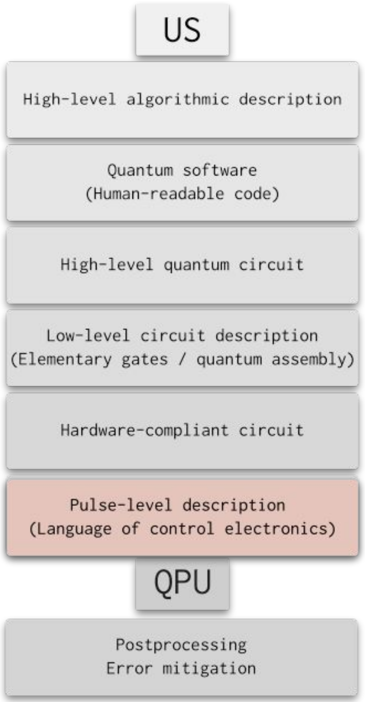


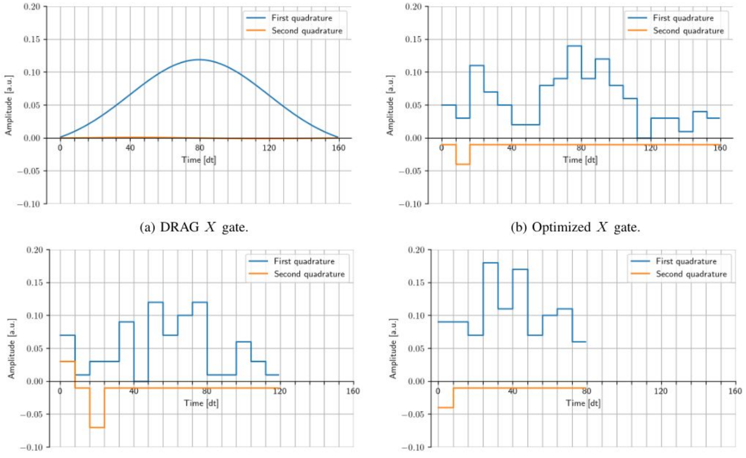
Image: Baker et al. (2021) *Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures*. ISCA '21.

Pulse design and optimization



Electromagnetic pulses are what *actually* implement the gates in a gate. Interesting problems here:

- optimizing duration/shape

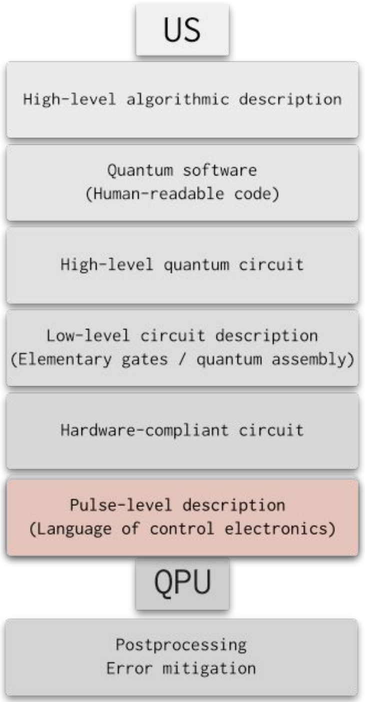


(a) DRAG X gate. (b) Optimized X gate.

(c) Optimized X gate $1.33\times$ faster. (d) Optimized X gate $2\times$ faster.

Image: E. Wright and R. de Sousa (2023) *Fast quantum gate design with deep reinforcement learning using real-time feedback on readout signals.* arXiv:2305.01169 [quant-ph]

Pulse design and optimization



Electromagnetic pulses are what *actually* implement the gates in a gate. Interesting problems here:

- optimizing duration/shape
- scheduling constraints

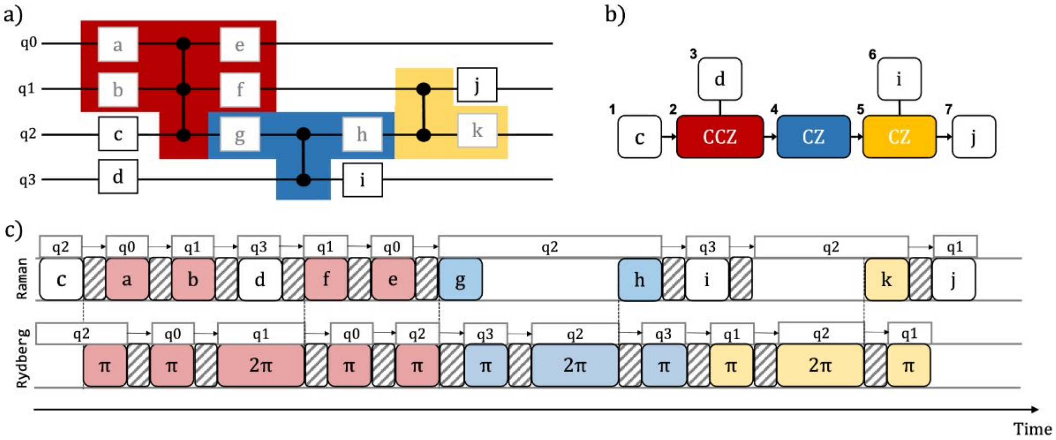
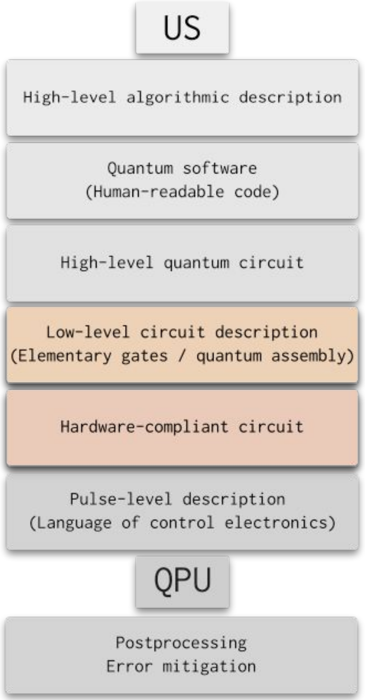
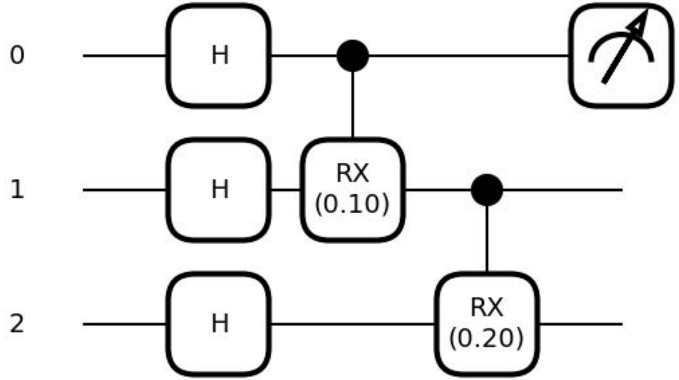


Image: R. B.-S. Tsai, H. Silvério, L. Henriet (2022) *Pulse-Level Scheduling of Quantum Circuits for Neutral-Atom Devices*. Phys. Rev. Applied 18 064035.

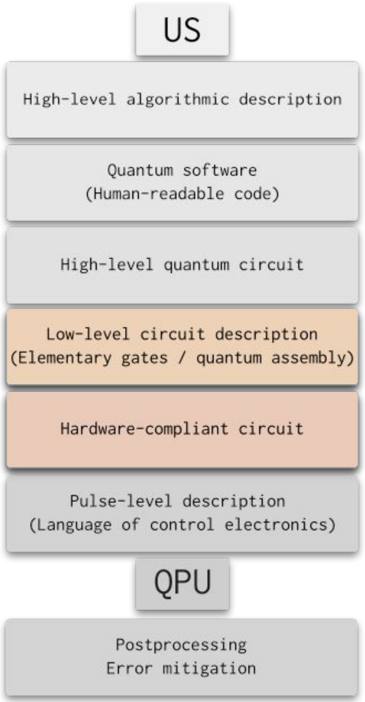
End-to-end example



Compile and optimize this circuit so it can run on a trapped-ion processor.



End-to-end example



The native gate set used by some trapped-ion processors (e.g., IonQ) is

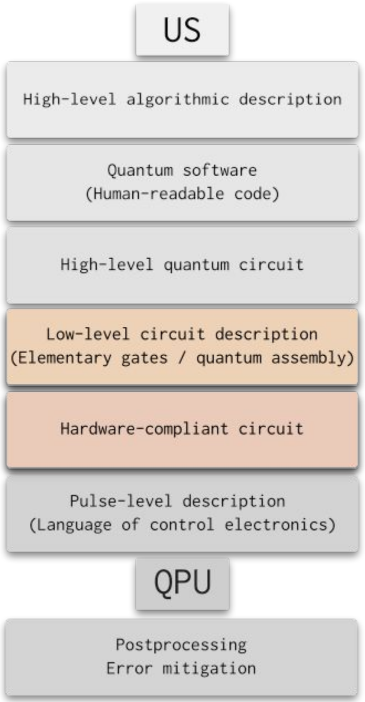
$$GPI(\phi) = \begin{pmatrix} 0 & e^{-i\phi} \\ e^{i\phi} & 0 \end{pmatrix}$$

$$GPI2(\phi) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -ie^{-i\phi} \\ -ie^{i\phi} & 1 \end{pmatrix}$$

$$MS = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & -i \\ 0 & 1 & -i & 0 \\ 0 & -i & 1 & 0 \\ -i & 0 & 0 & 1 \end{pmatrix}$$

*MS is actually a parametrized gate in general, but only this fixed-parameter version is implemented in hardware.

End-to-end example



RX/RY/RZ are **arbitrary-angle** rotations about **fixed axes.**

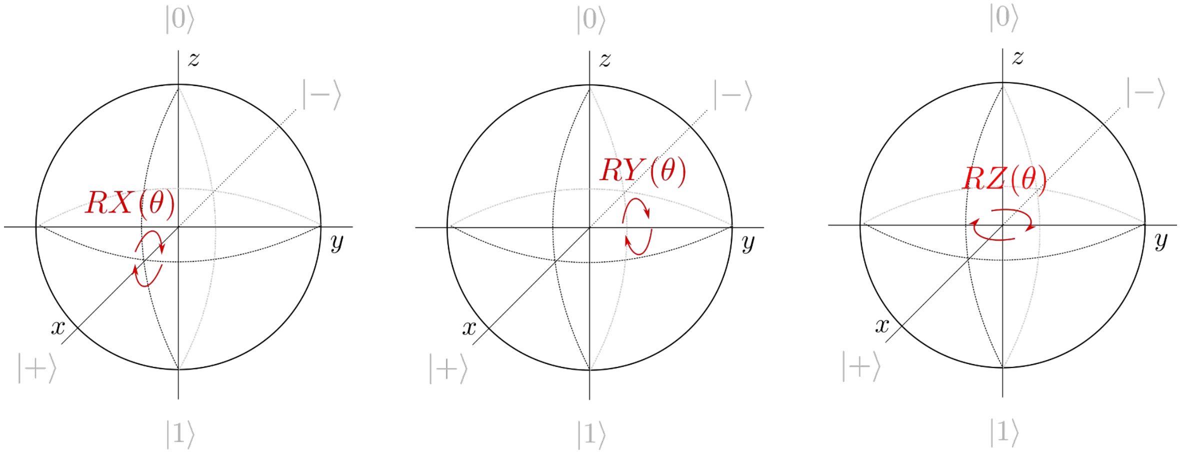
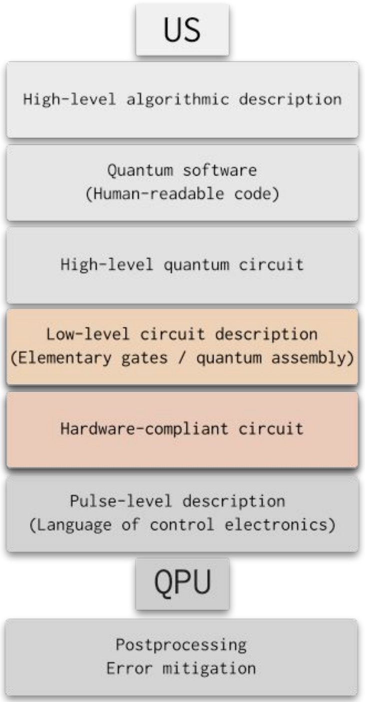


Image adapted from Xanadu Quantum Codebook Node I.6

End-to-end example

$$GPI(\phi) = \begin{pmatrix} 0 & e^{-i\phi} \\ e^{i\phi} & 0 \end{pmatrix}$$

$$GPI2(\phi) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -ie^{-i\phi} \\ -ie^{i\phi} & 1 \end{pmatrix}$$



GPI/*GPI2* are **fixed-angle** rotations about **arbitrary axes** in *xy*-plane.

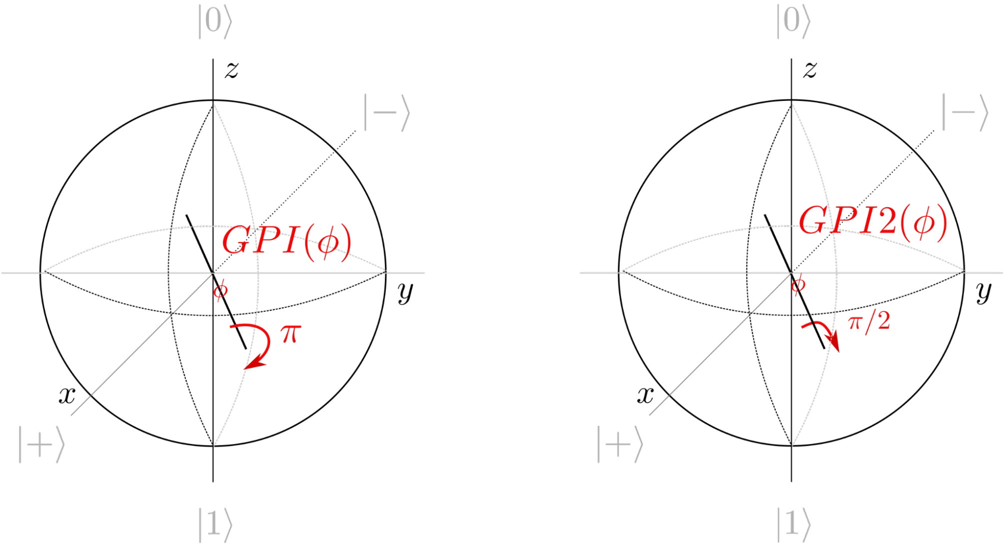
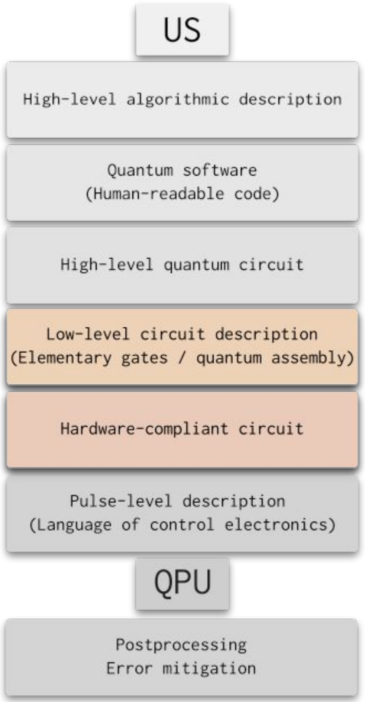
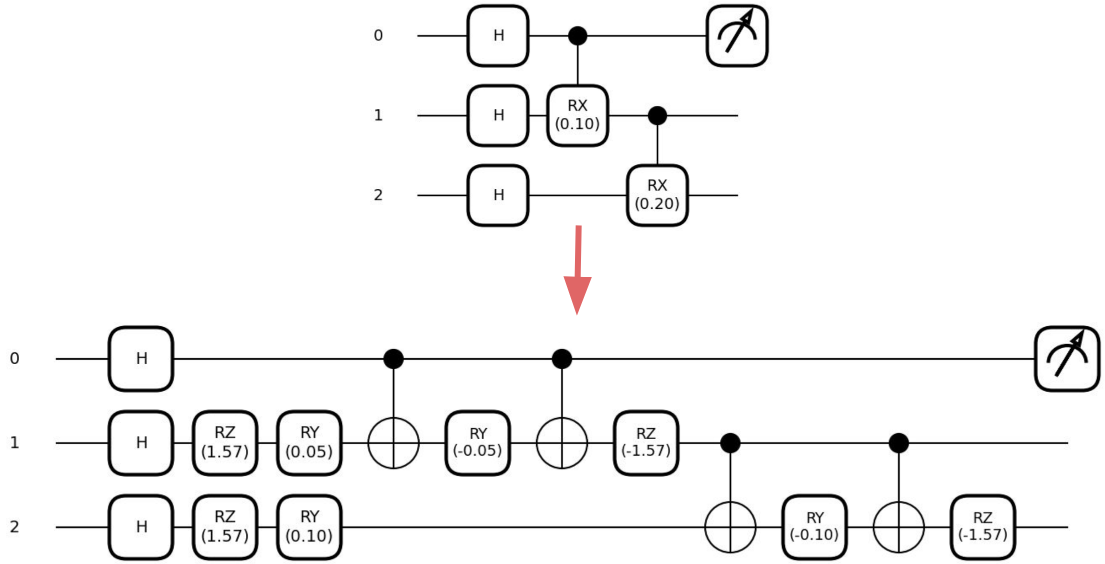


Image adapted from Xanadu Quantum Codebook Node I.6

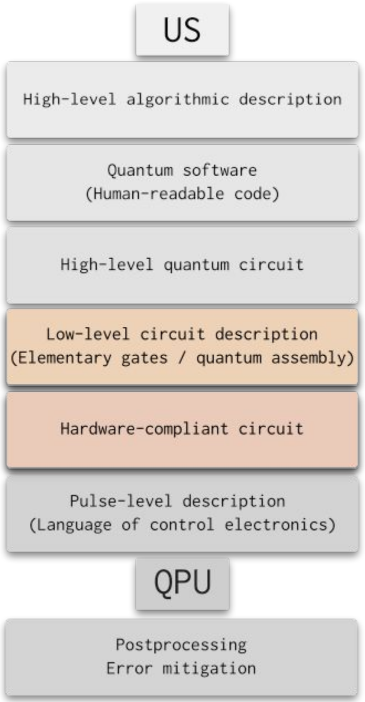
End-to-end example



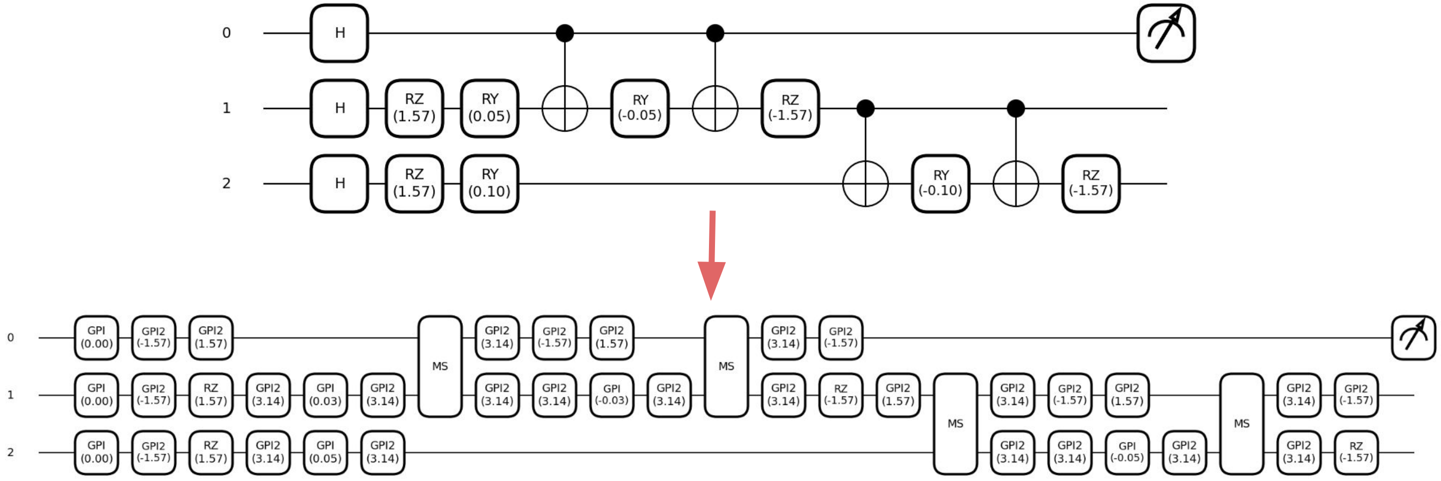
Unroll using known decompositions.



End-to-end example



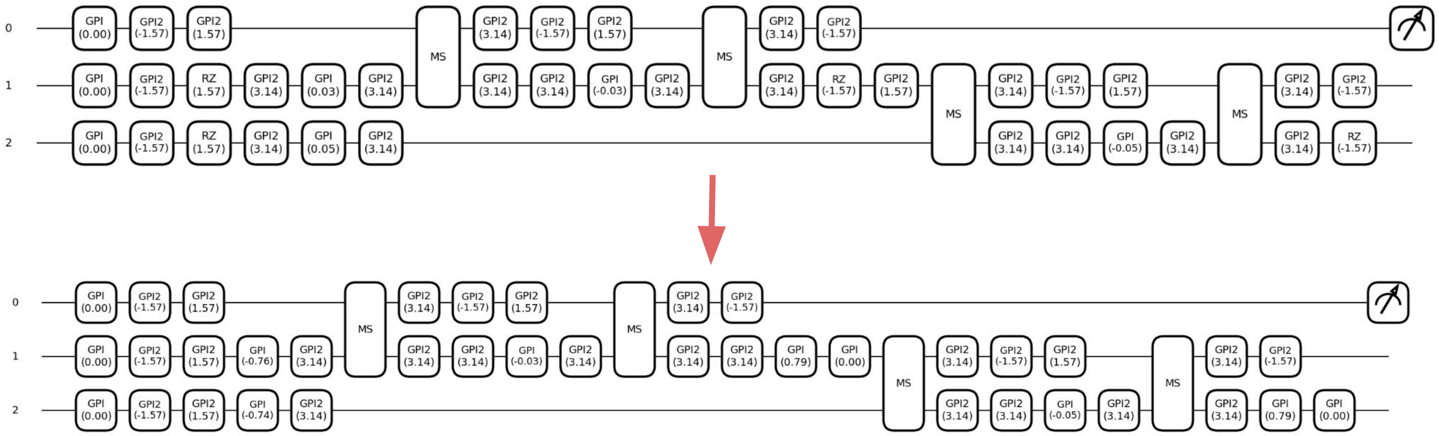
Map to native trapped-ion gates.



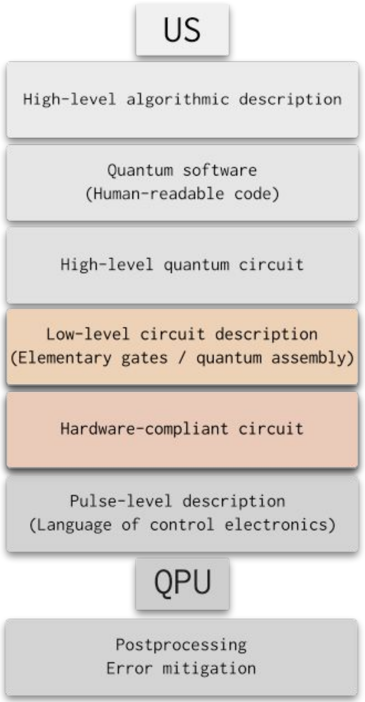
End-to-end example

- US
- High-level algorithmic description
- Quantum software (Human-readable code)
- High-level quantum circuit
- Low-level circuit description (Elementary gates / quantum assembly)
- Hardware-compliant circuit
- Pulse-level description (Language of control electronics)
- QPU
- Postprocessing Error mitigation

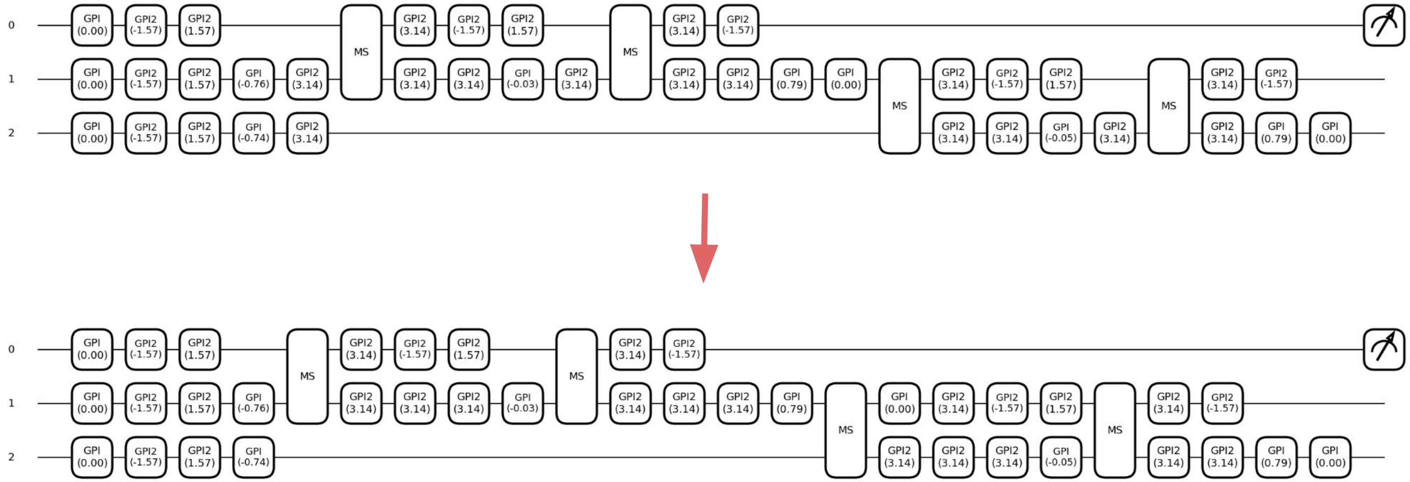
Hardware-specific optimization: virtually apply RZs



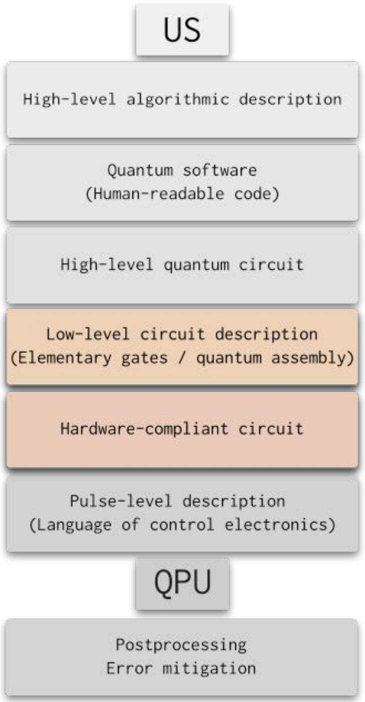
End-to-end example



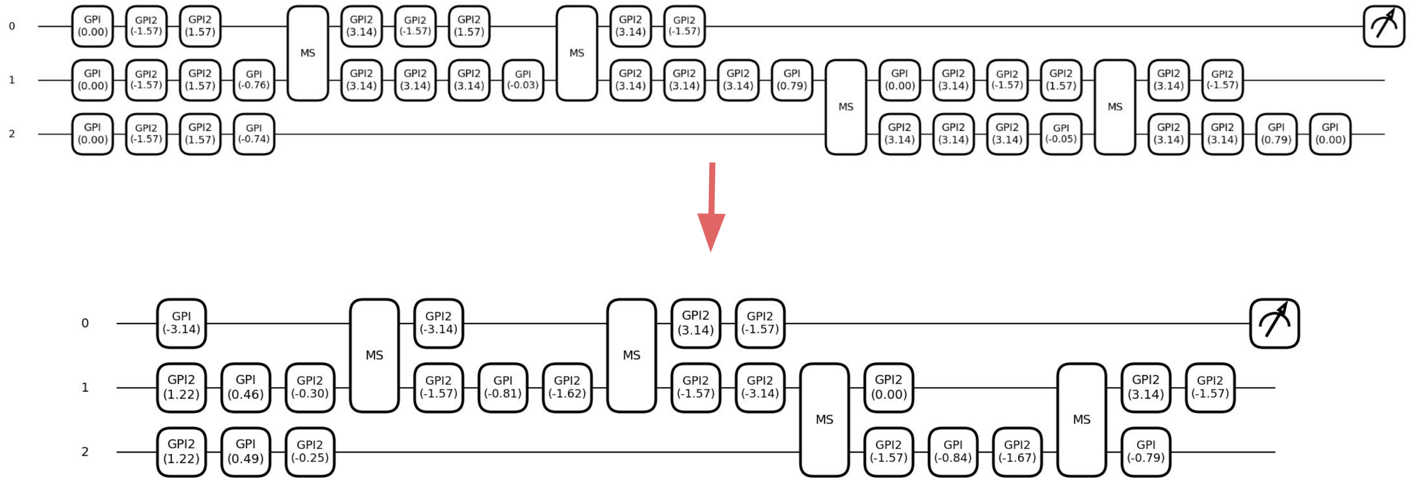
Optimization: exchange order of commuting gates



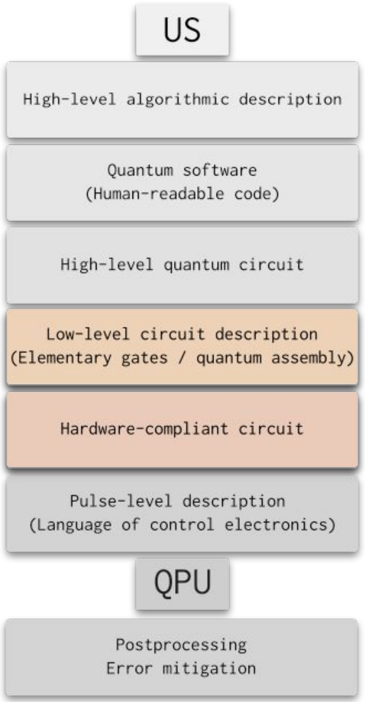
End-to-end example



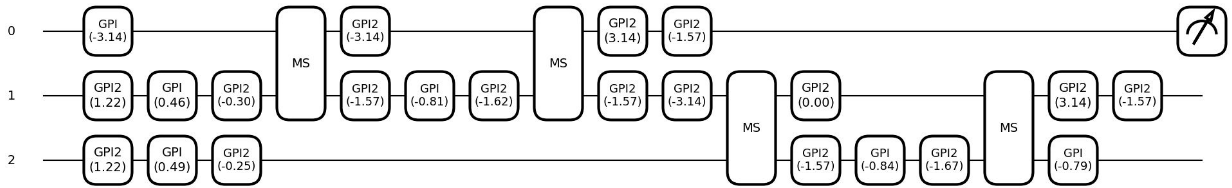
Optimization: fuse sequences of >3 single-qubit gates and apply circuit identities.

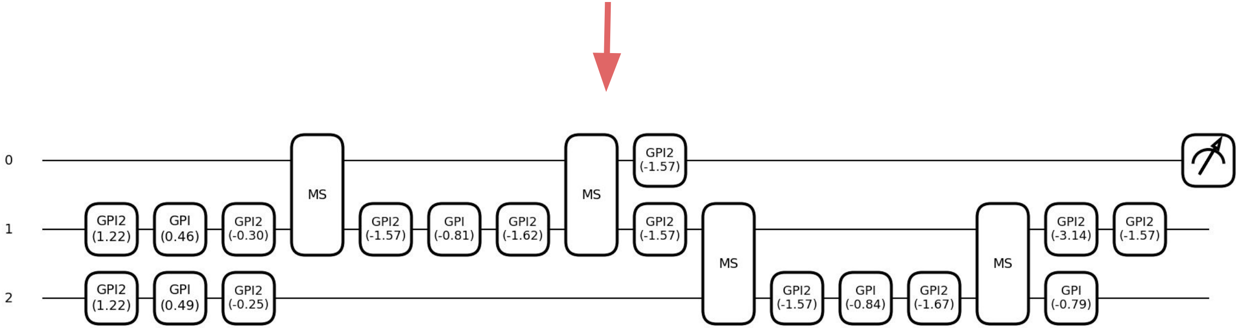


End-to-end example

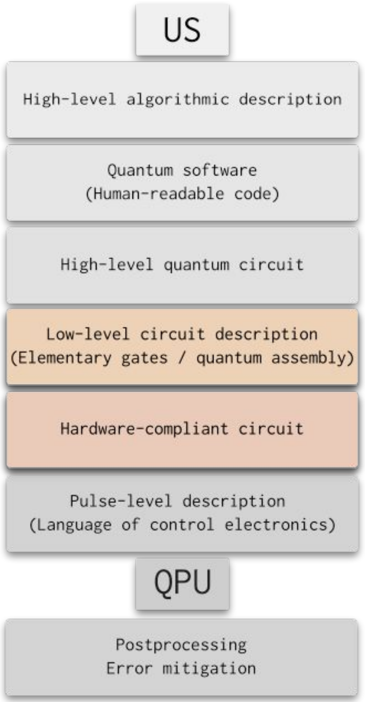


Repeat pushing/fusing/identity application.

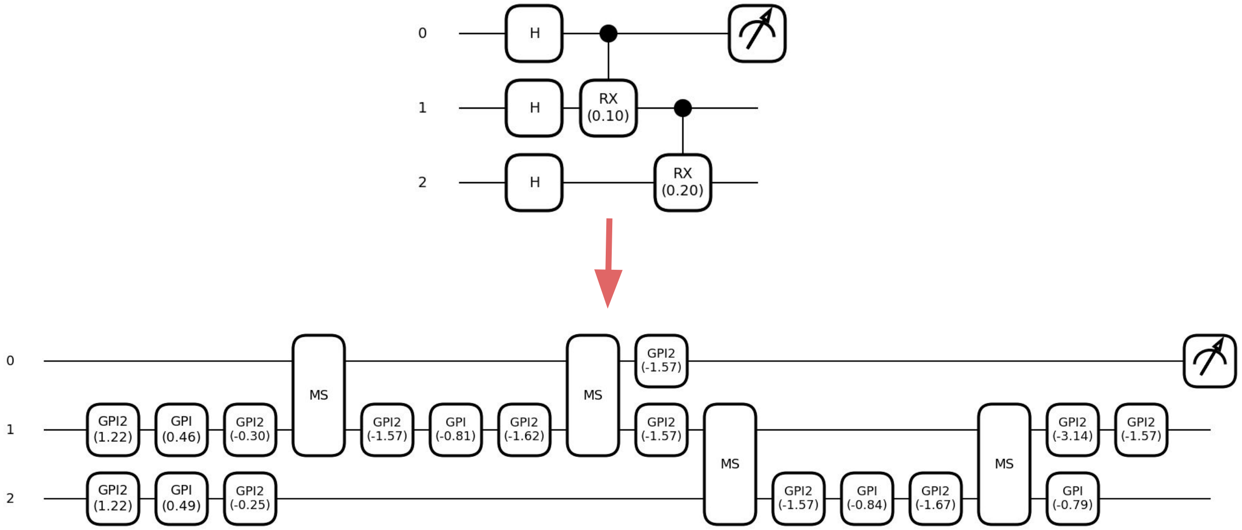




End-to-end example



Bask in the glory of compilation.



Software usage

Many general-purpose packages contain preset passes, and the ability to create your own.

```python
qiskit_circuit = QuantumCircuit(...)

transpiled_circuit = transpile(
    qiskit_circuit,
    optimization_level=3,
    coupling_map=...,
    layout_method="sabre"
)
```

```python
@qml.qnode(dev)
@expand_rot_and_remove_zeros
@qml.compile(pipeline=[
    qml.transforms.cancel_inverses,
    qml.transforms.single_qubit_fusion(),
    qml.transforms.commute_controlled(direction="left"),
    qml.transforms.merge_rotations()
], num_passes=3)
def tapered_circuit_simplified(params):
    for wire in dev.wires:
        if hf_state_tapered[wire] == 1:
            qml.PauliX(wires=wire)
```

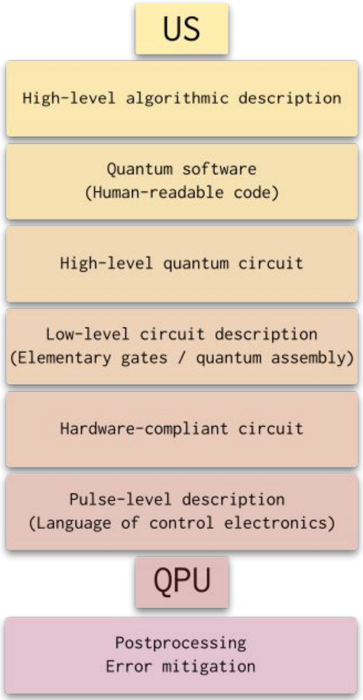
See also: Cirq, TKET, staq, XACC, and more.

Challenge
~~Problem:~~ many things are (computationally) hard

Many parts of the stack **still require expert input** and tailored design.

Need **automation** to scale up to (multi-QPU) devices w/1000s of qubits:

- circuit synthesis
- circuit optimization
- mapping, routing, and scheduling
- noise-aware techniques
- verification and debugging



Next time

Content:

- Hands-on with quantum compilation (write your own transforms)

Action items:

- A2 and Lit A1
- Choose paper and post project group details to Piazza

Recommended reading:

- Transforms blog post (*syntax is out-of-date, but idea holds*)
  https://pennylane.ai/blog/2021/08/how-to-write-quantum-function-transforms-in-pennylane/
- Explore the **qml.transforms** module
  https://pennylane.readthedocs.io/en/stable/code/qml_transforms.html