CPEN 400Q
Quantum compilation II

Wednesday 12 February 2025

Announcements

- Quiz 5 Monday 24 Feb (about this week)
- Literacy Assignment 1 due today at 23:59
  - LA2 tomorrow morning
- Assignment 2 due Thursday 27 Feb at 23:59
  - All questions available now
- Final project details available on PrairieLearn
  - Still working on list of potential papers

From tomorrow I’m off work 7-10 days - reach out to Ritu (email / Piazza) for questions.

Final project details

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

Five parts:

- (10%) Midterm checkpoint (write up + short meeting)
- (28%) Companion report
- (29%) Software implementation
- (28%) In-class presentation (last weeks of class)
- (5%) Weekly peer assessment surveys

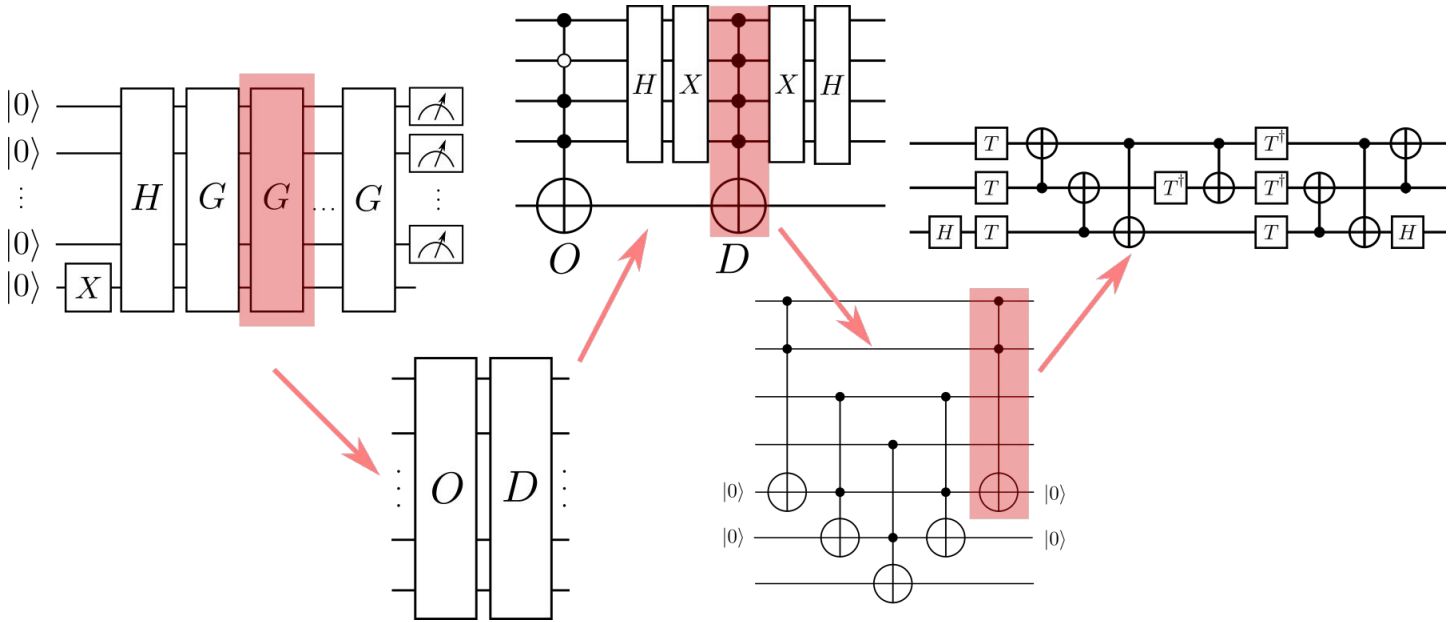
Work in groups of 4 (7 groups) or 3 (2 groups).

Final project: important dates

- **2024-02-24**: Group and topic selection due (Piazza)
- **2024-02-28**: First weekly survey due
- **2024-03-14; 2024-03-17-2024-03-19**: Midterm checkpoints
- **2024-04-10**: Final report and software implementation due.

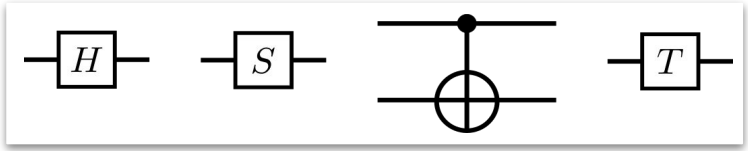
We opened the black box

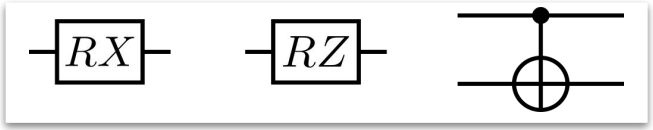
High-level gates in quantum circuits must be synthesized or decomposed into gates from a **universal gate set.**

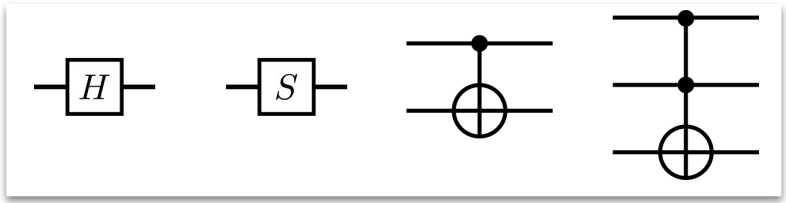


Universal gate sets

With just a few gates, we can implement any unitary either exactly, or approximately up to arbitrary precision.

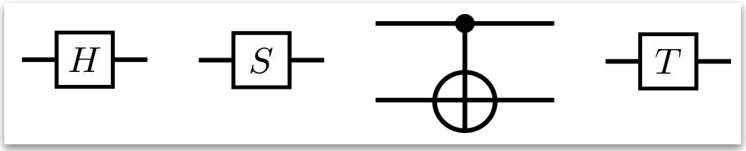






Take-home exercise

Express controlled Hadamard using gates from



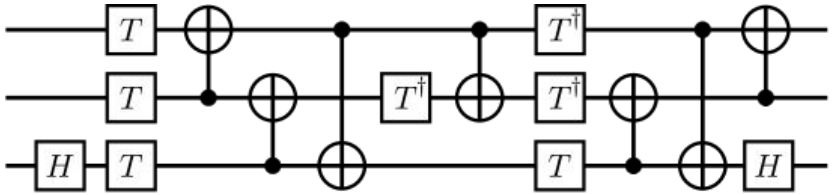
Last time

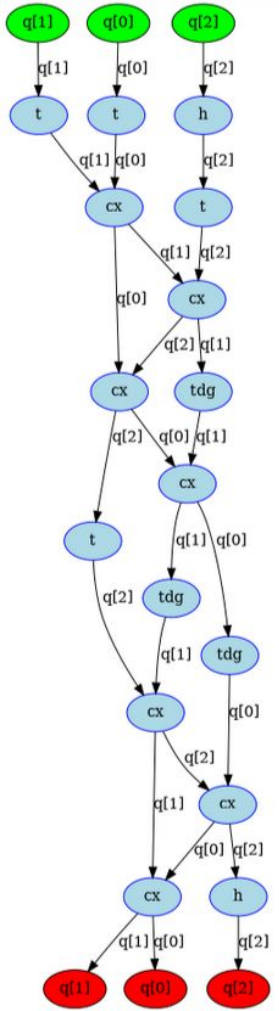
Circuit optimization

Metrics of interest:

- width (number of qubits)
- gate count (1 or 2-qubit gates, or specific gates)
- depth (of whole circuit, or specific gate)
  - longest path in directed, acyclic graph representation

Must consider **tradeoffs** involved



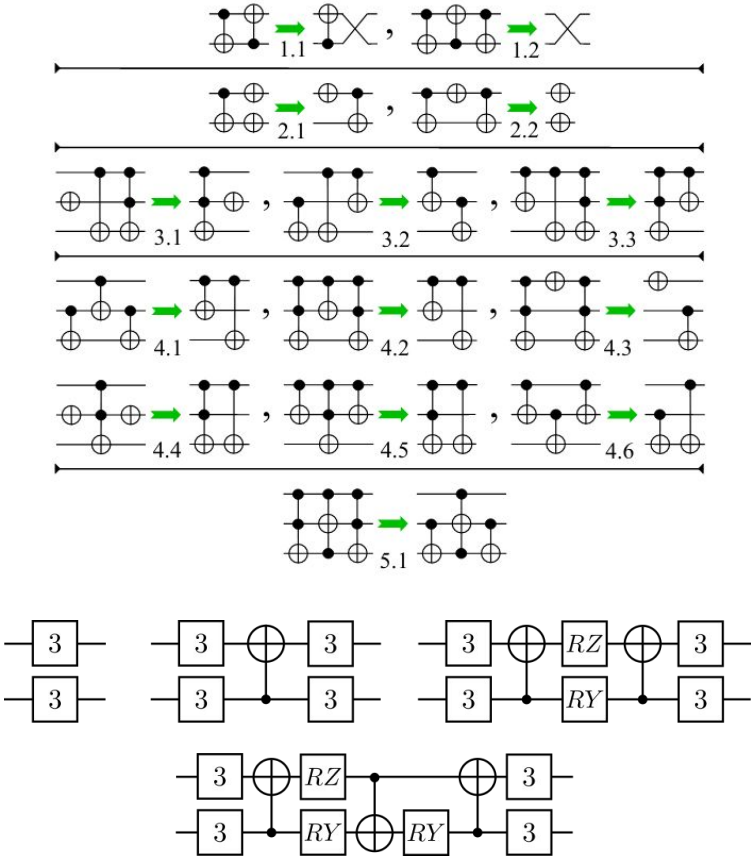


Last time

Optimization passes

Passes have varying complexity:

- inverse cancellation
- rotation merging
- template matching
- pushing commuting gates through each other to enable cancellations
- two-qubit block *resynthesis*

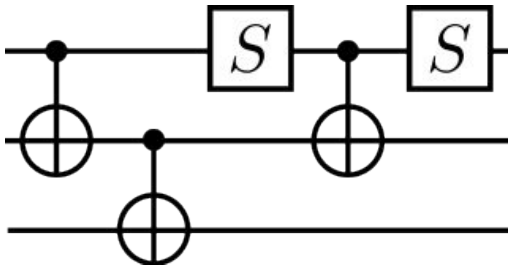


D. M. Miller, D. Maslov, G. W. Dueck. *A Transformation Based Algorithm for Reversible Logic synthesis.* DAC 2003.

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

Take-home exercise

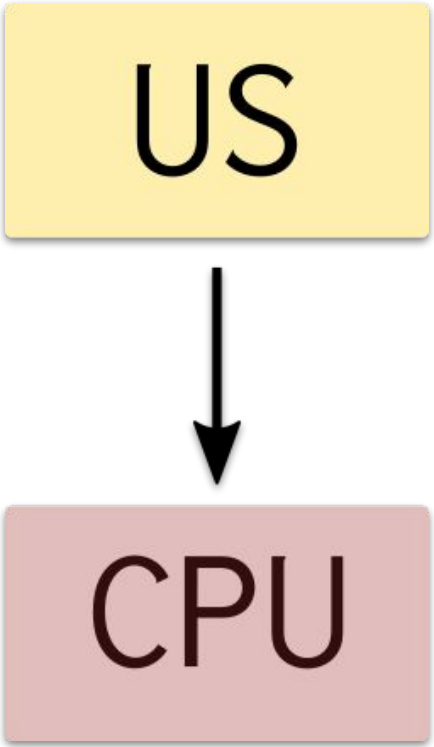
Determine this circuit’s resource counts, then optimize it.
What is the minimum depth?

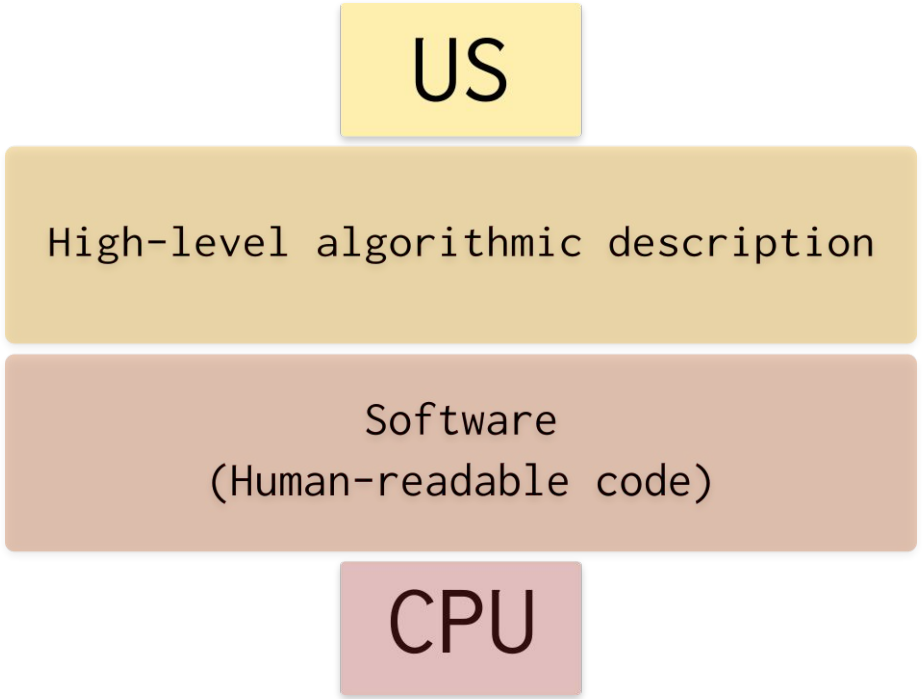


Today

Mostly “infotainment”:

- identify the different parts of the compilation stack
- describe the challenges involved in compiling quantum circuits to hardware
- define “hybrid” algorithms, and outline strategies for compiling combined classical + quantum workflows


US
CPU


US
High-level algorithmic description
Software
(Human-readable code)
CPU

Compilers are **essential**

```c
#include<stdio.h>

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

Compilers are **essential**

```c
#include<stdio.h>

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

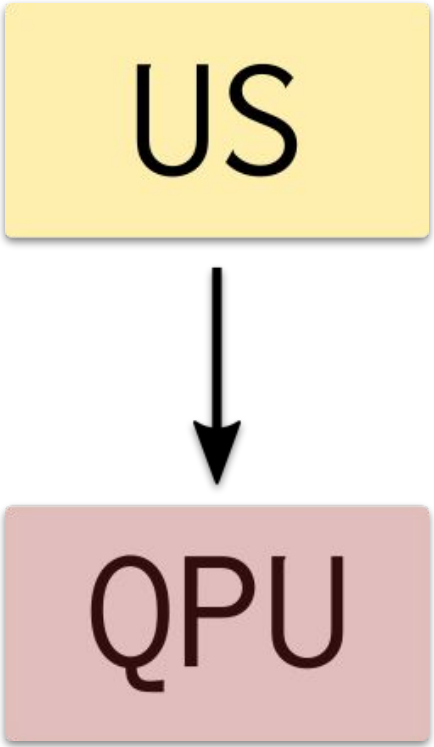
```
        .file   "hello_world.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 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**

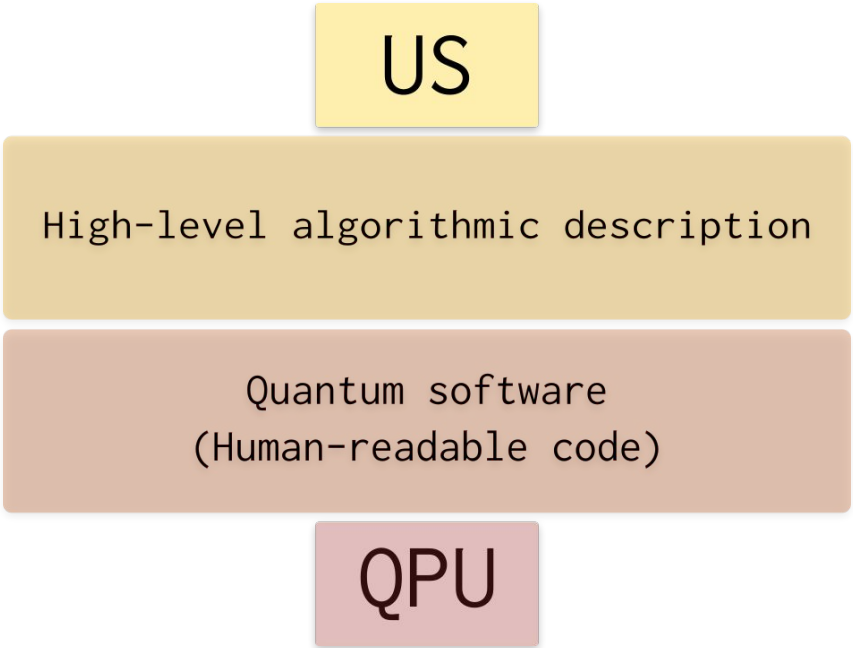
```c
#include<stdio.h>

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
```

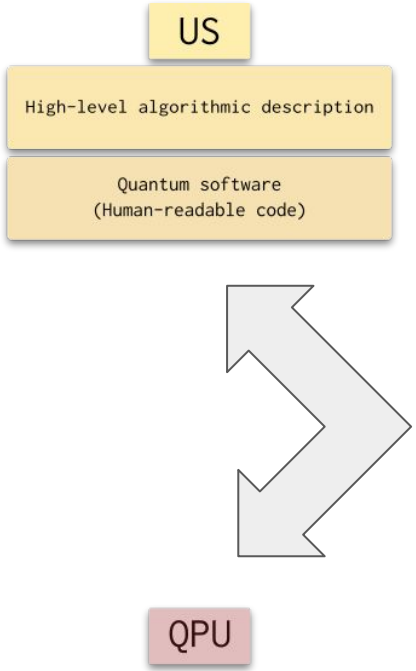
```
 11a0 ff48c1fd 03741f31 db0f1f80 00000000  .H...t.1........
 11b0 4c89f24c 89ee4489 e741ff14 df4883c3  L..L..D..A...H..
 11c0 014839dd 75ea4883 c4085b5d 415c415d  .H9.u.H...[]A\A]
 11d0 415e415f c366662e 0f1f8400 00000000  A^A_.ff.........
 11e0 f30f1efa c3                          .....
Contents 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:
 2058 14000000 00000000 017a5200 01781001  .........zR..x..
```


US
QPU

In an ideal world...

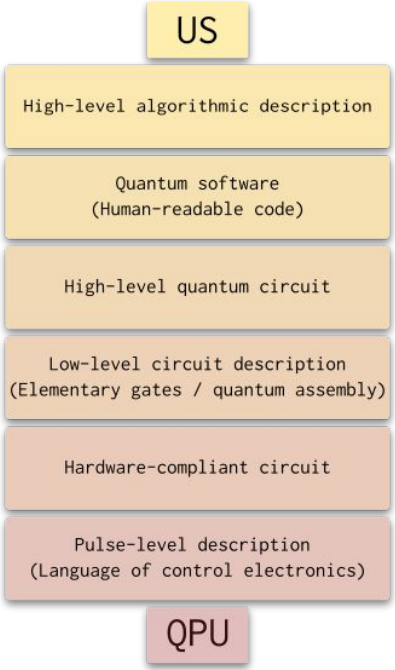


This lecture

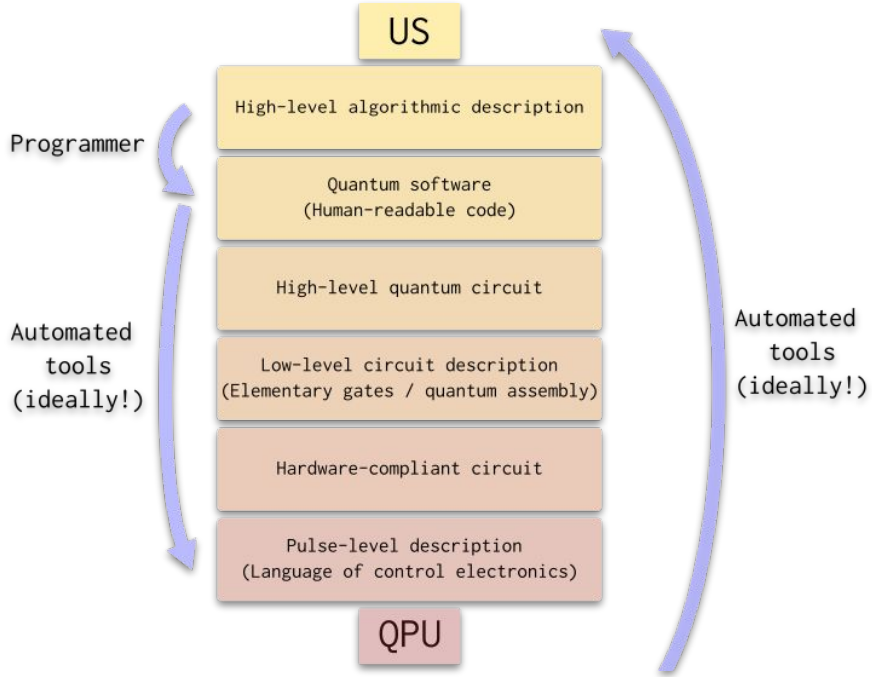


What happens in between?

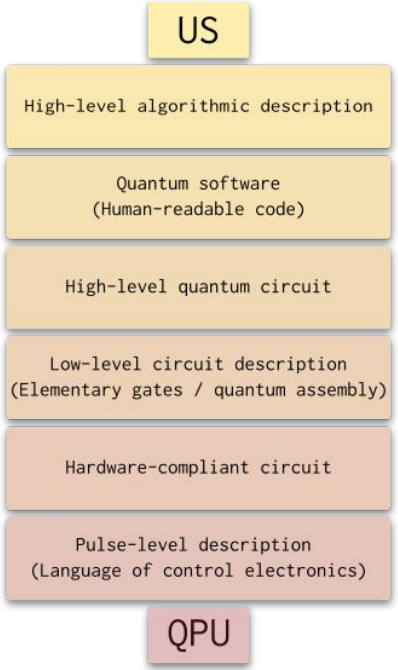
The quantum compilation stack

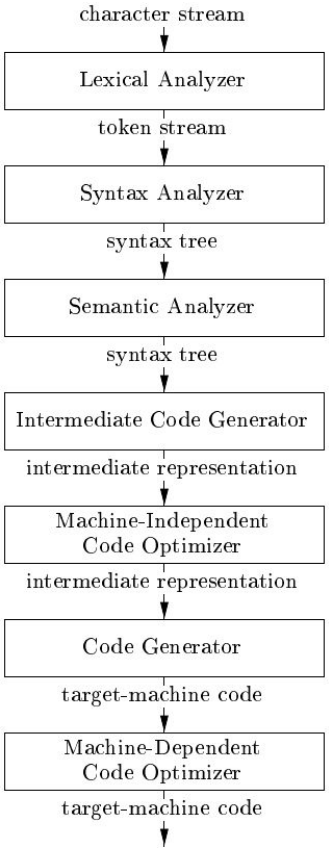


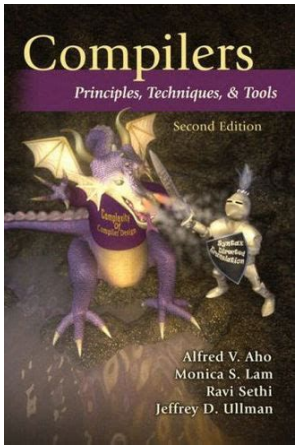
The quantum compilation stack



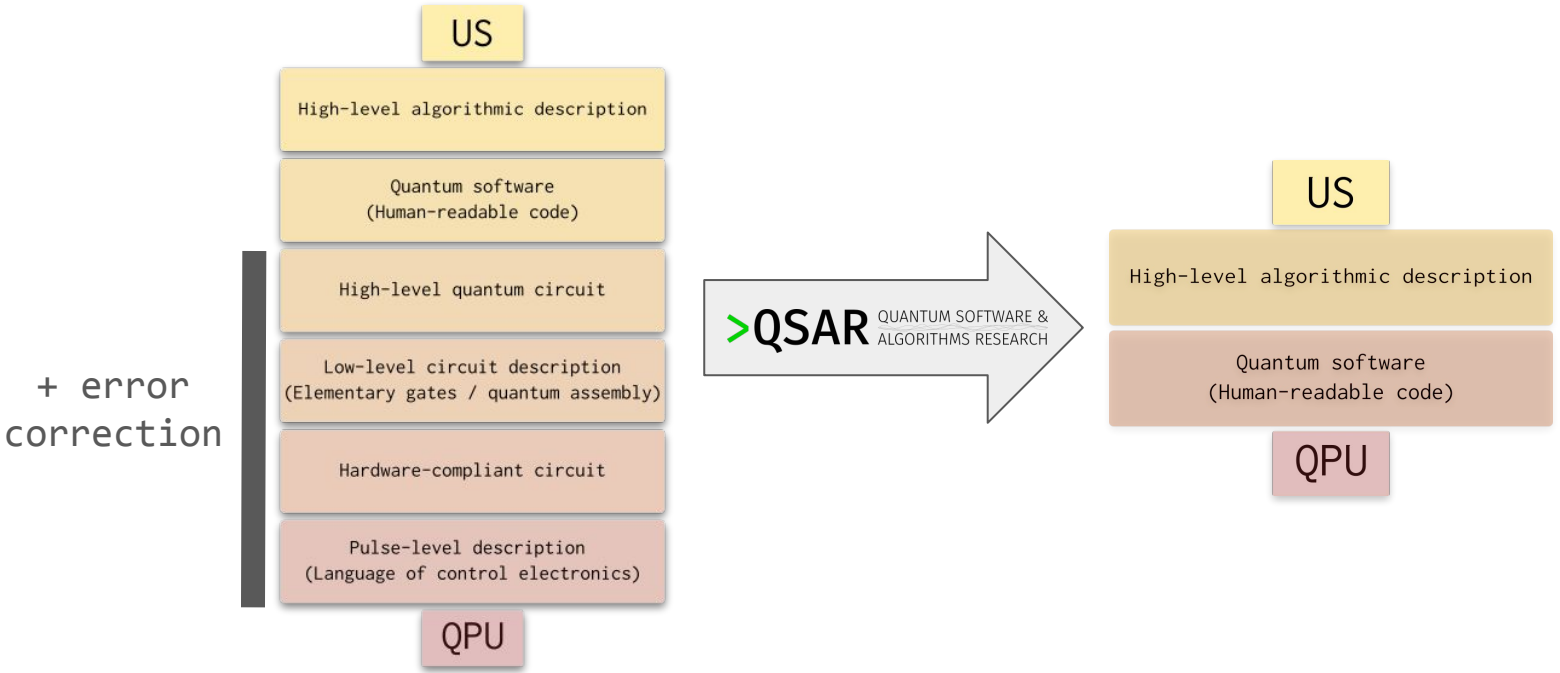
The quantum compilation stack



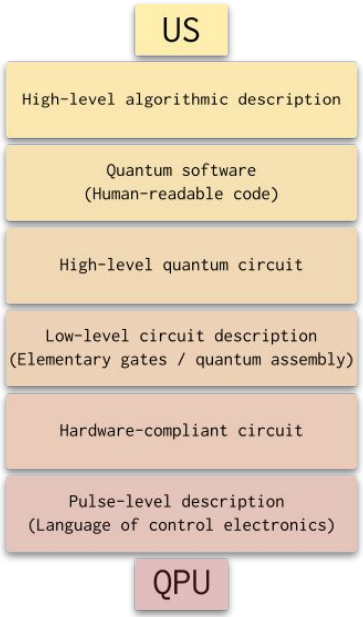




Goal of my research group



Compilation



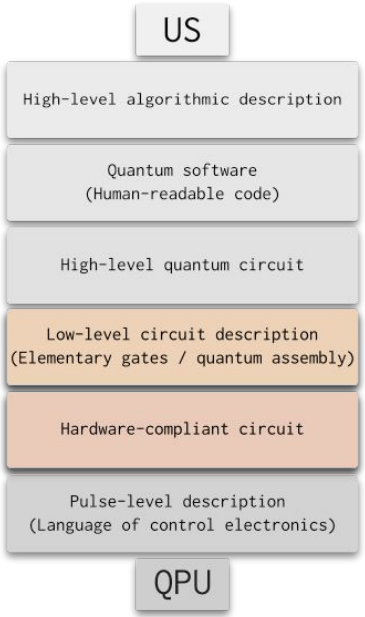
Implemented as a sequence of modular ***passes***, or ***transforms***, that modify a circuit in various ways.

Loosely divided into 3 stages:

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

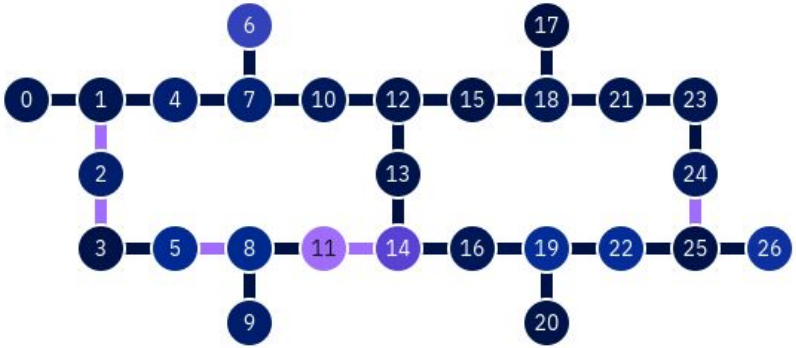
*talked about these Monday

Qubit placement and routing



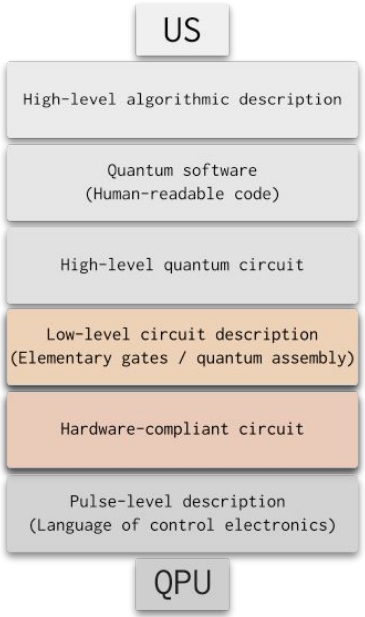
Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology



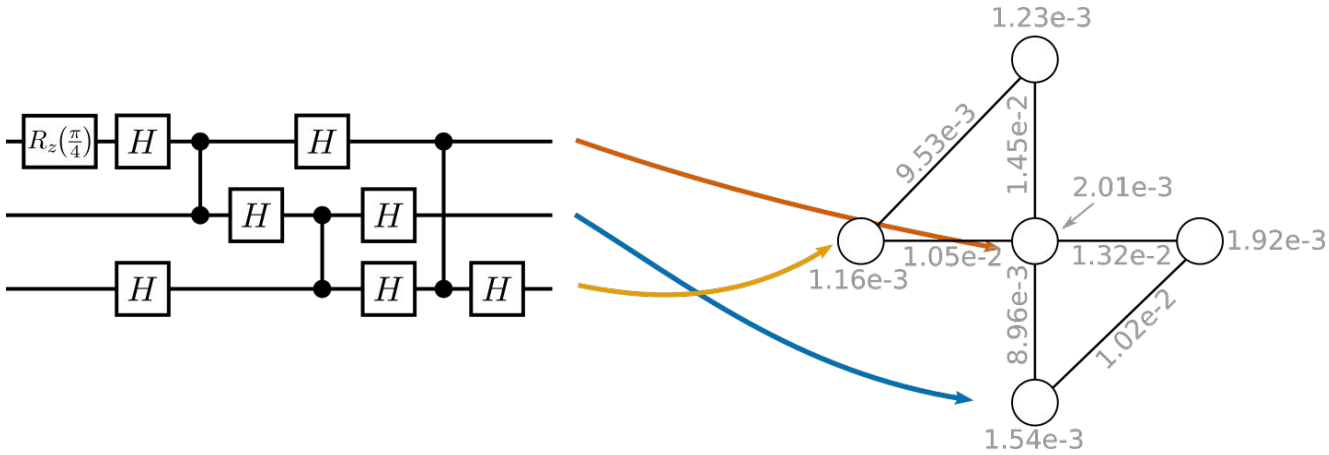
IBM Q Kolkata – screencap 2023-08-24

Qubit placement and routing

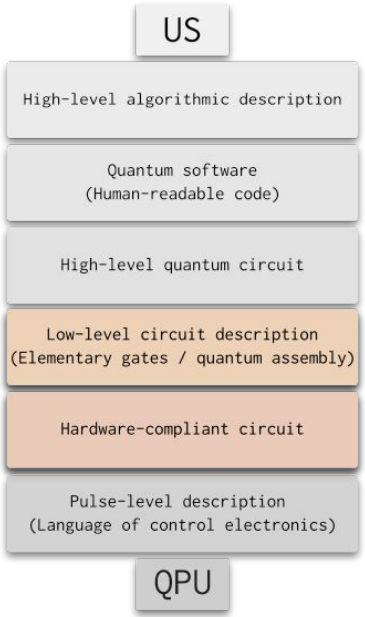


Each type of machine has its own compilation challenges:

- superconducting machines have restricted topology

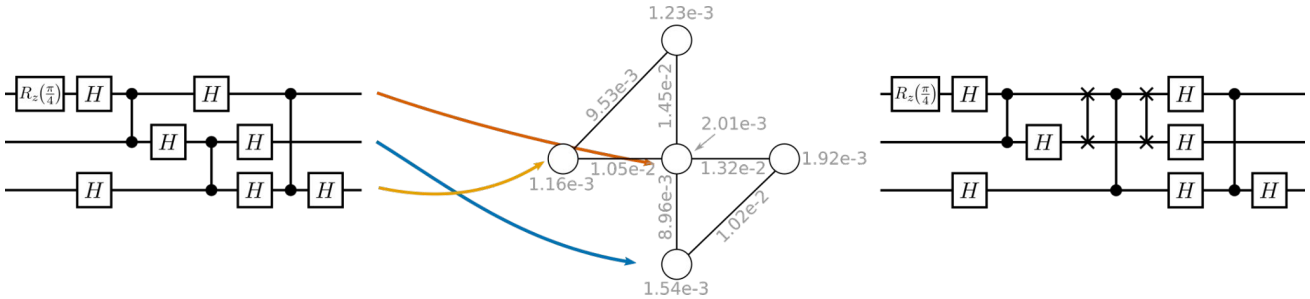


Qubit placement and routing



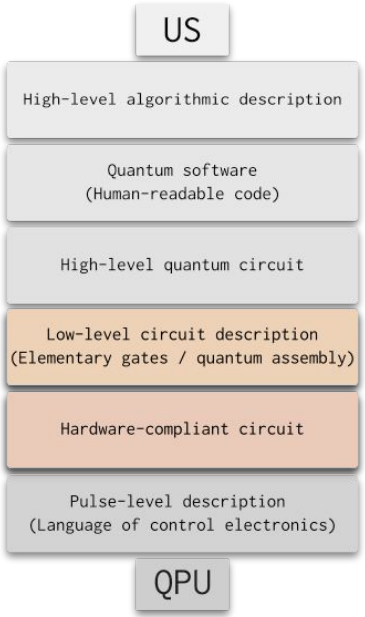
Each type of machine has its own compilation challenges:

- 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

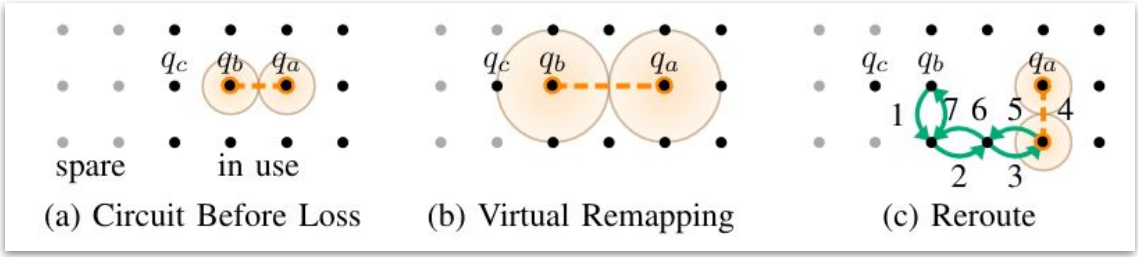
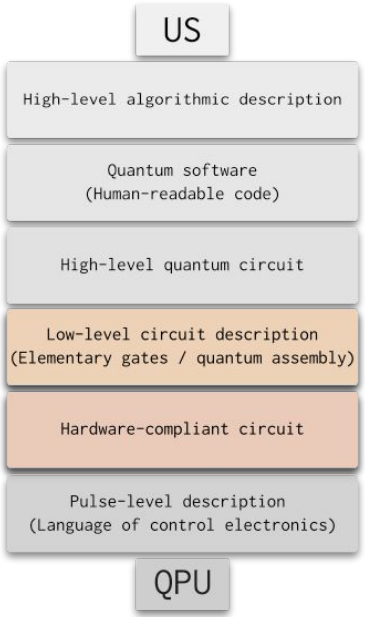


Image: Baker et al. (2021) *Exploiting Long-Distance Interactions and Tolerating Atom Loss in Neutral Atom Quantum Architectures*. ISCA '21.

Qubit placement and routing



Techniques can use optimal solvers, heuristics, or a combination of both.

Example: SABRE (“SWAP-based BidiREctional heuristic search)

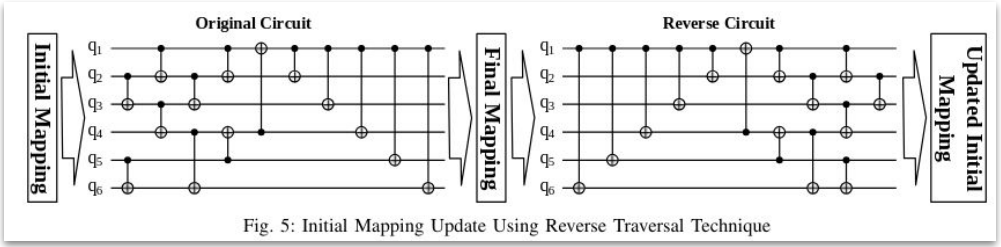


Fig. 5: Initial Mapping Update Using Reverse Traversal Technique

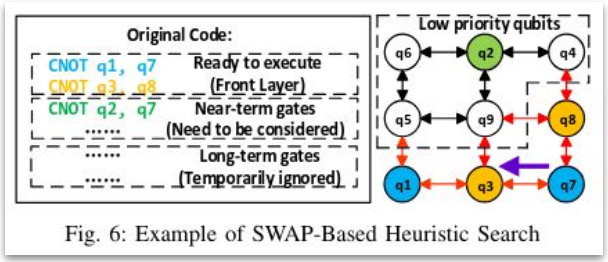
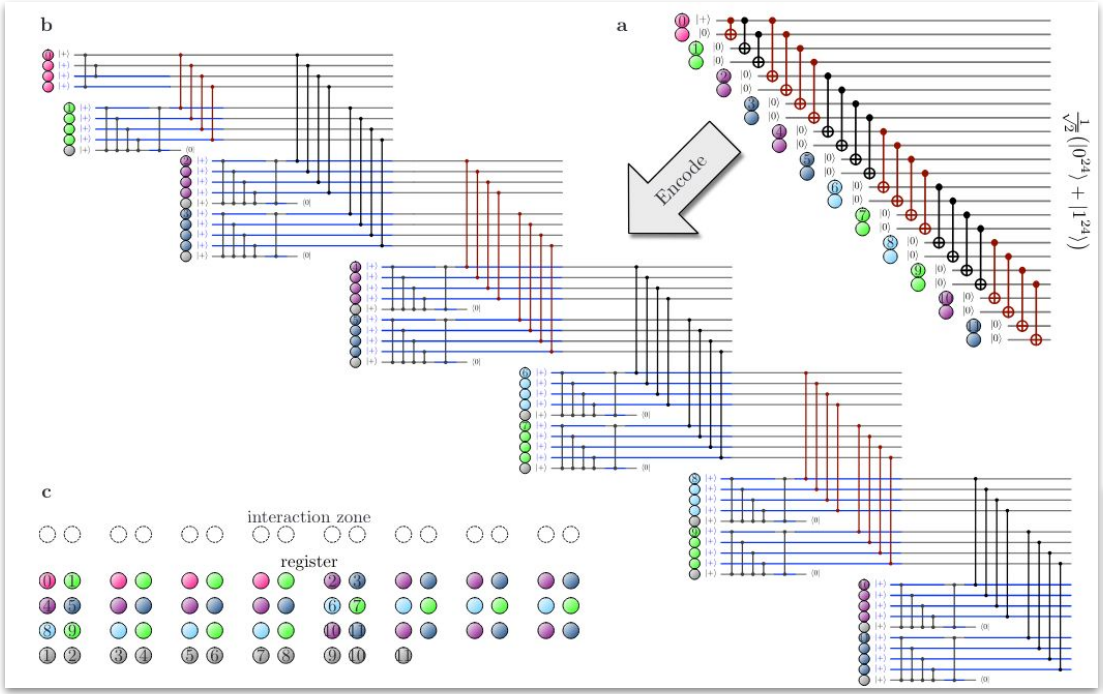


Fig. 6: Example of SWAP-Based Heuristic Search

Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). ACM, New York, NY, USA, 1001–1014.

Gate scheduling



Recent demonstration of logical qubits on neutral atom machine: *move qubits in and out of "interaction zone" for 2-qubit gates*

(Microsoft + Atom computing + collaborators)

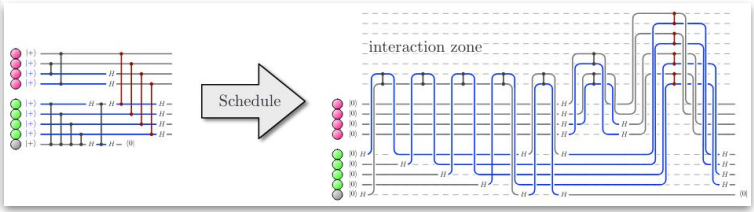
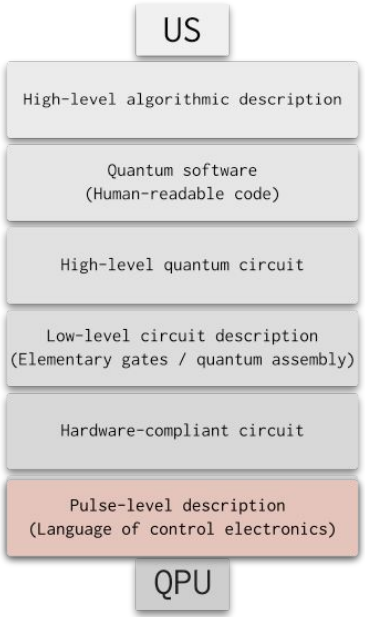


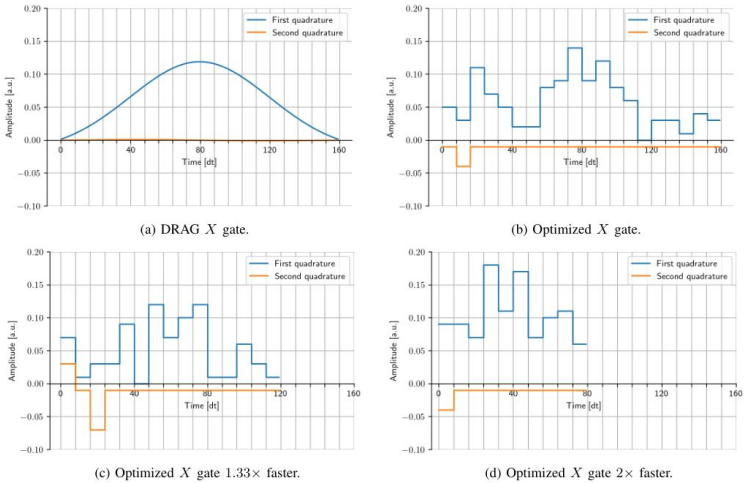
Image: Fig. 4 and 5, B. W. Reichardt et al., "Logical computation demonstrated with a neutral atom quantum processor". arXiv:2411.11822 [quant-ph].

Pulse design and optimization



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

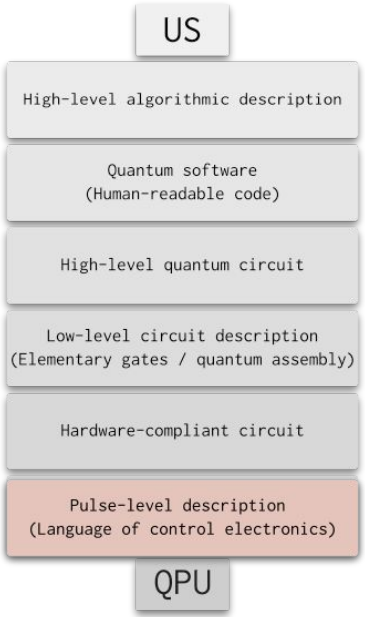
- optimizing duration/shape



(a) DRAG X gate. (b) Optimized X gate. (c) Optimized X gate 1.33× faster. (d) Optimized X gate 2× 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

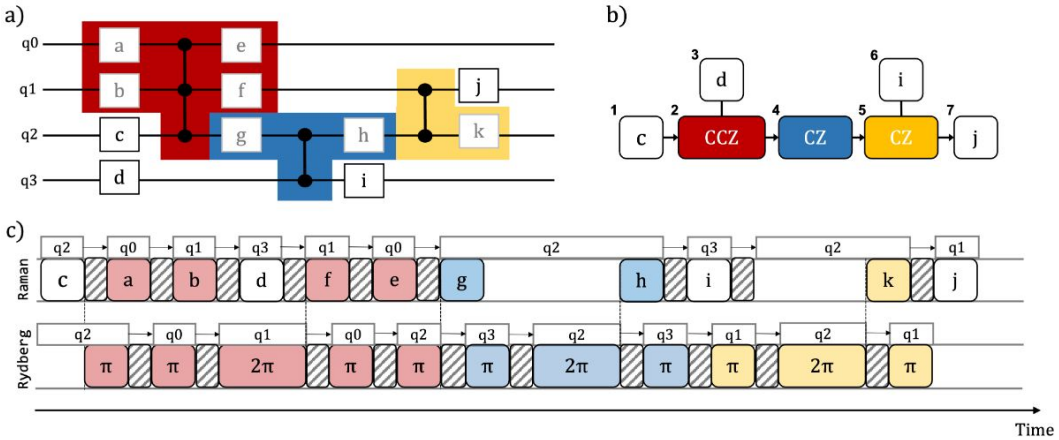
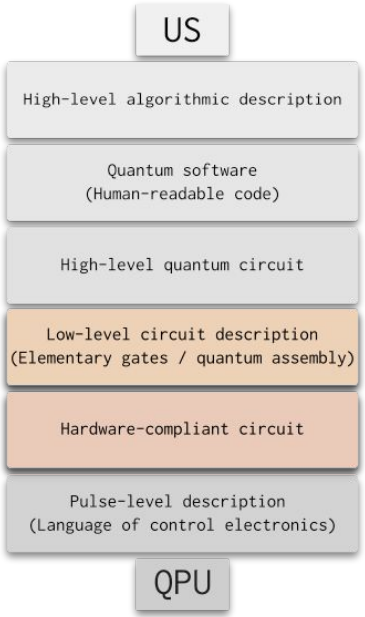
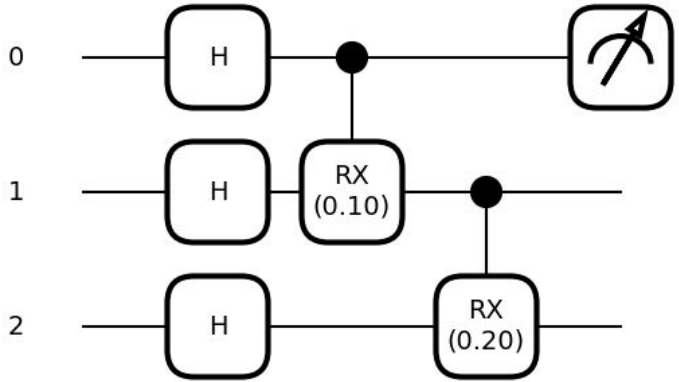


Image: R. B.-S. Tsai, H. Silvério, L. Henriet (2022) *Pulse-Level Scheduling of Quantum Circuits for Neutral-Atom Devices*. Phys. Rev. Applied 18 064035.

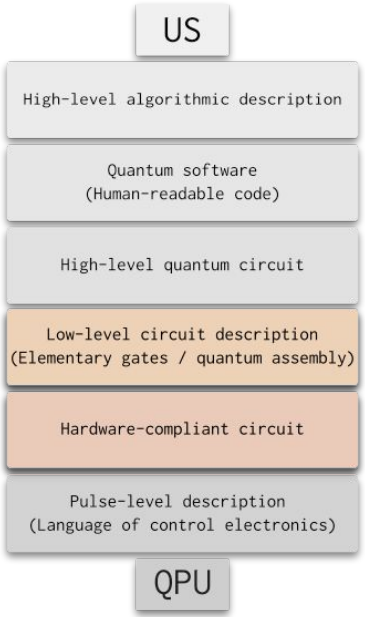
End-to-end example



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

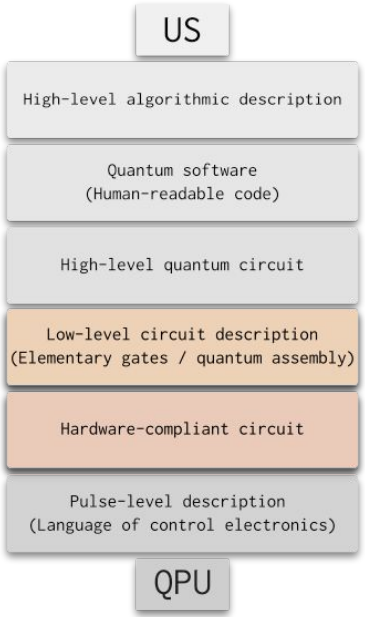
$$GPI(\phi) = \begin{pmatrix} 0 & e^{-i\phi} \\ e^{i\phi} & 0 \end{pmatrix}$$

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

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

*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*: **arbitrary-angle** rotations about **fixed axes.**

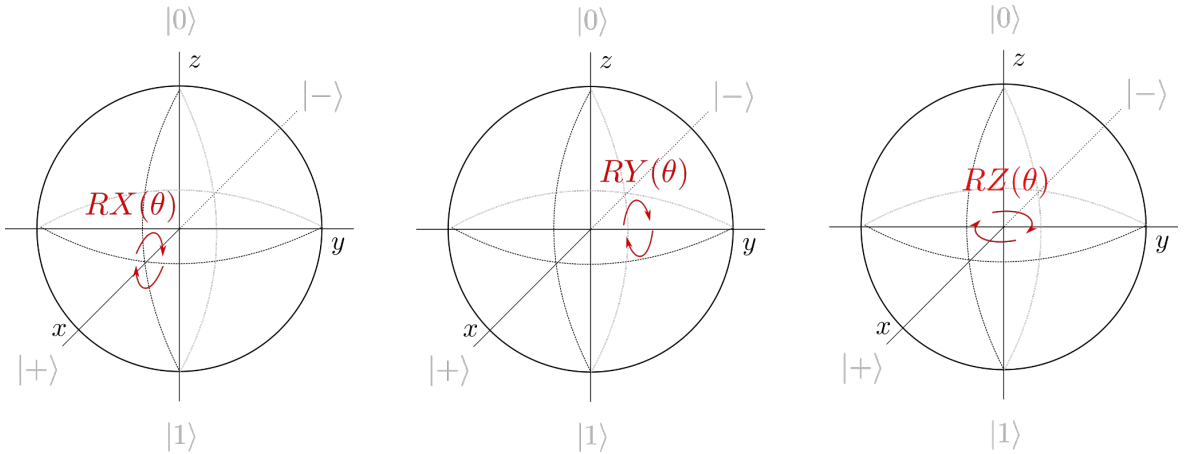
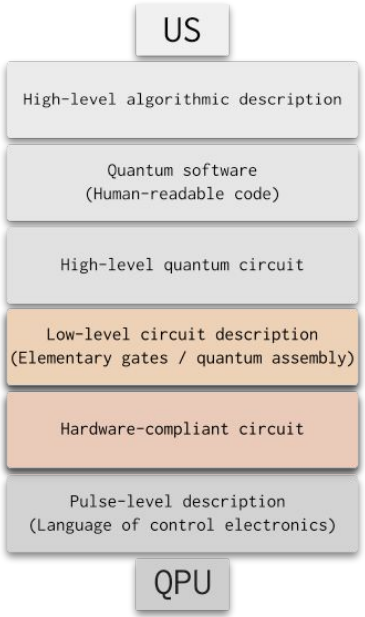


Image adapted from Xanadu Quantum Codebook Node I.6

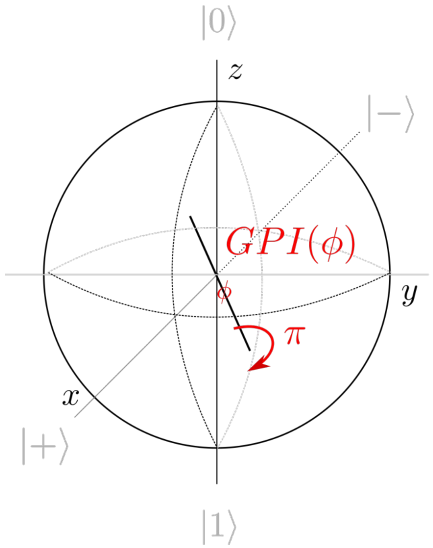
$$GPI(\phi) = \begin{pmatrix} 0 & e^{-i\phi} \\ e^{i\phi} & 0 \end{pmatrix}$$

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

End-to-end example



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



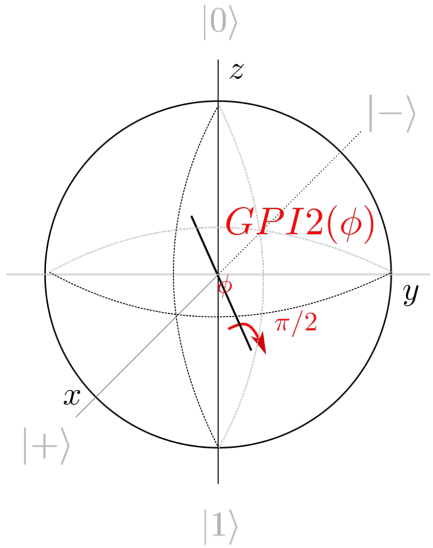
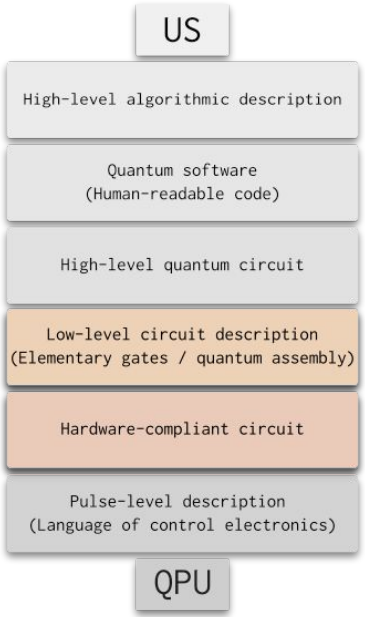
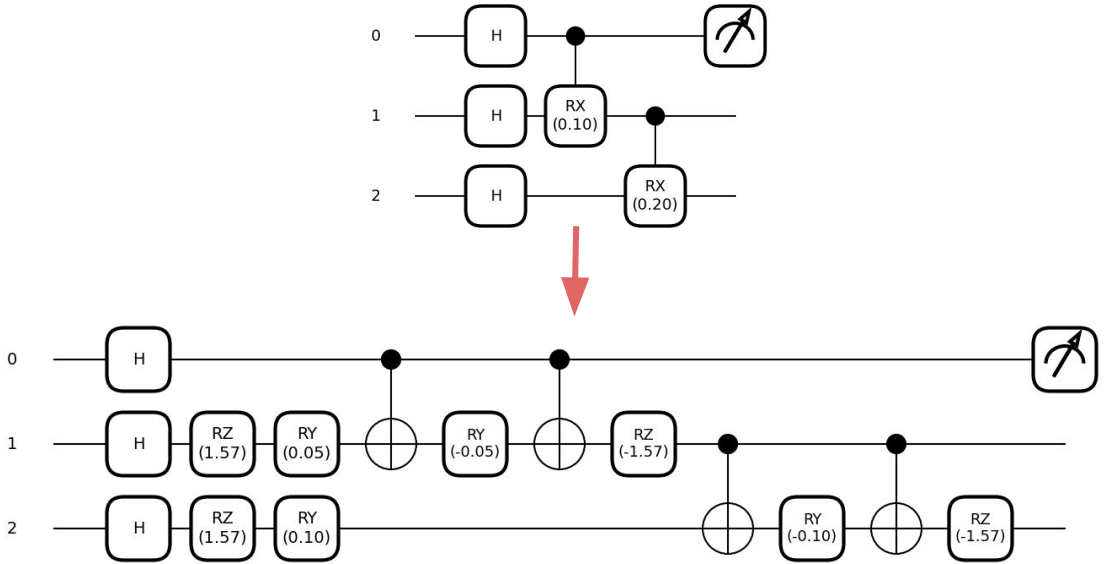


Image adapted from Xanadu Quantum Codebook Node I.6

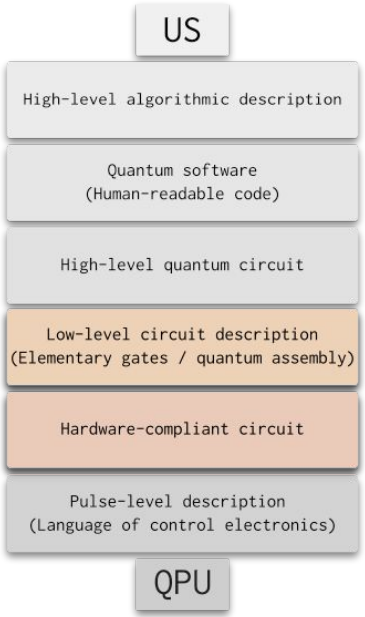
End-to-end example



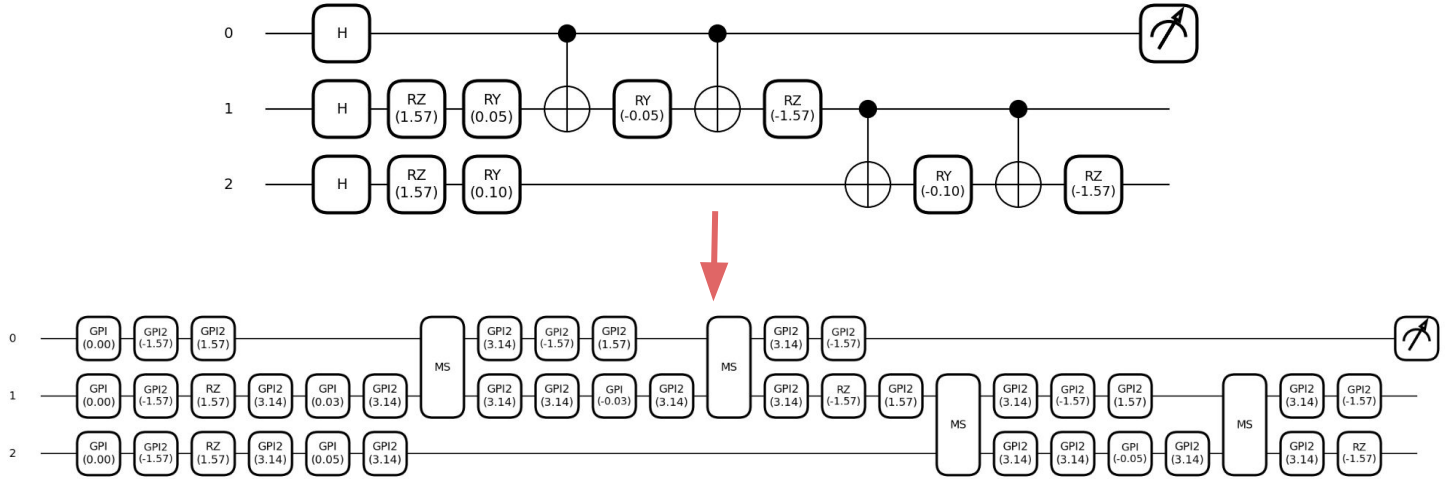
Unroll using known decompositions.



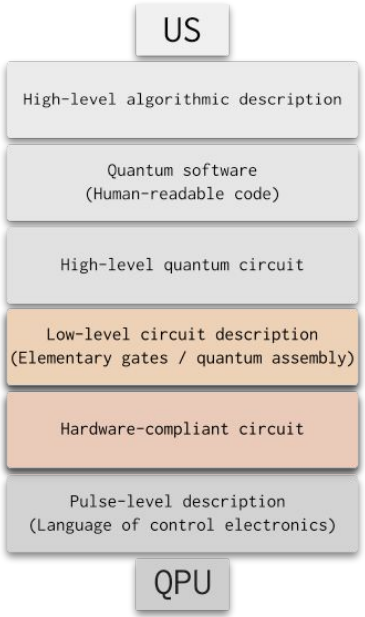
End-to-end example



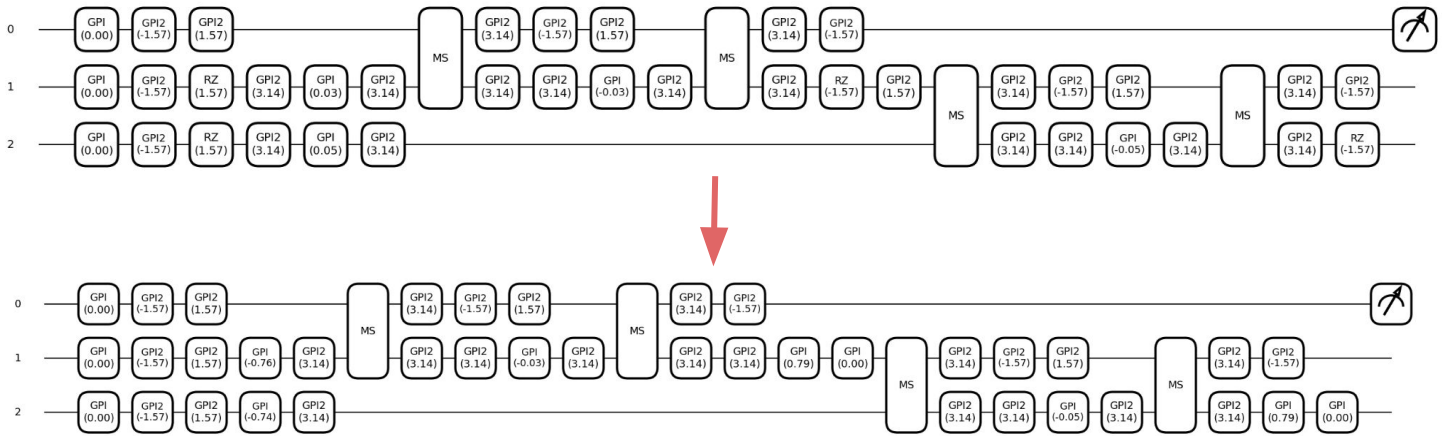
Map to native trapped-ion gates.



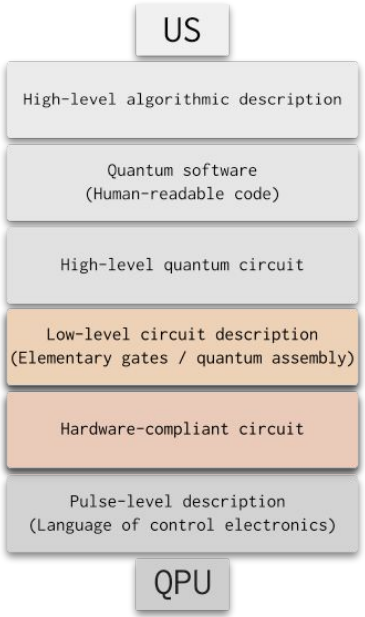
End-to-end example



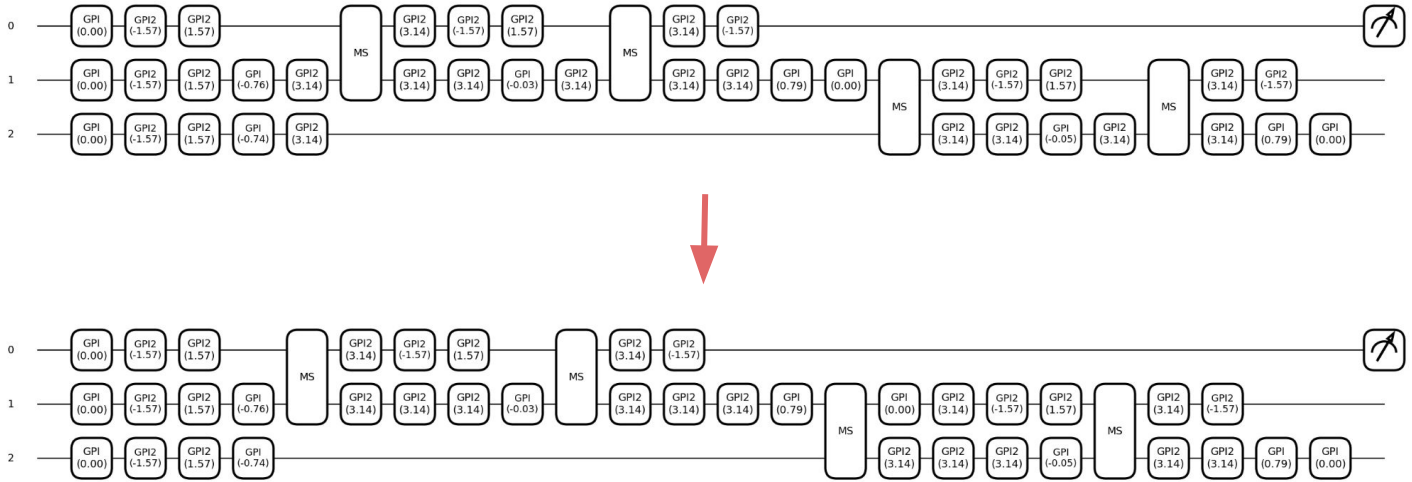
Hardware-specific optimization: virtually apply RZs



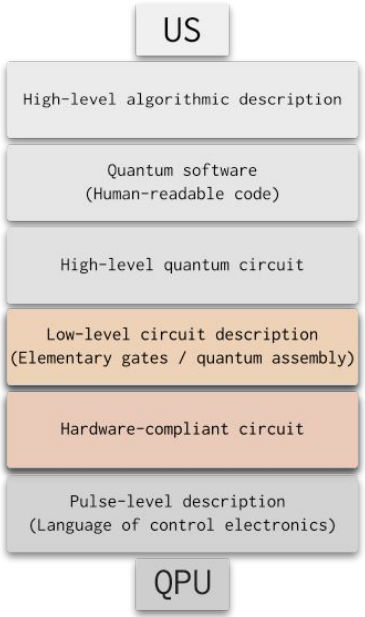
End-to-end example



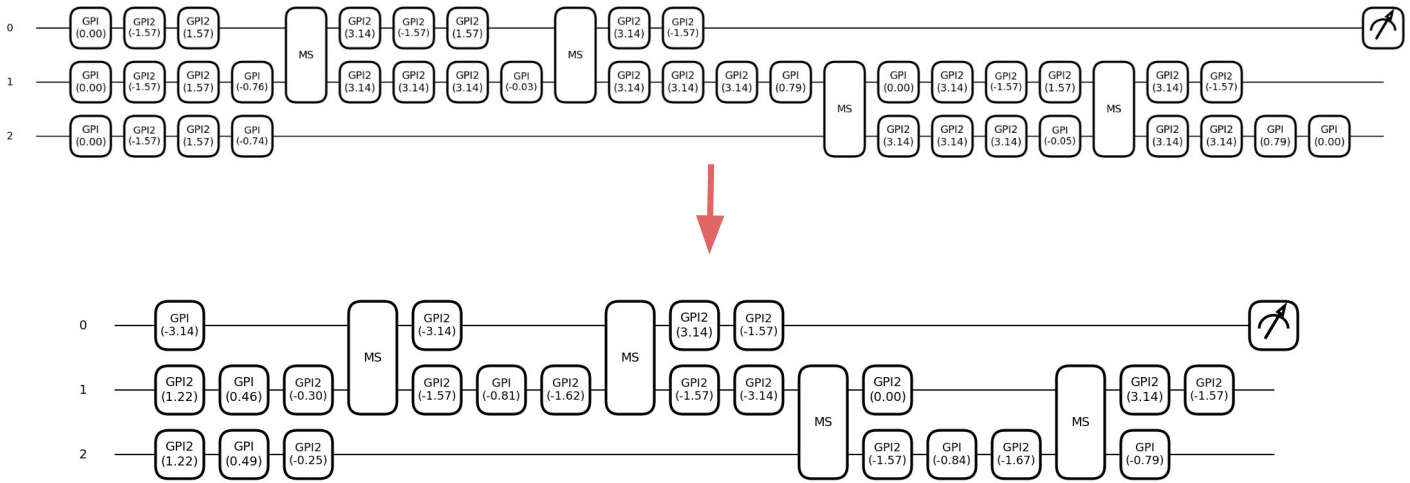
Optimization: exchange order of commuting gates



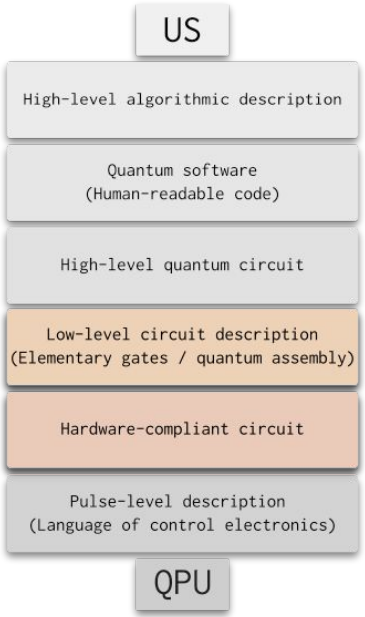
End-to-end example



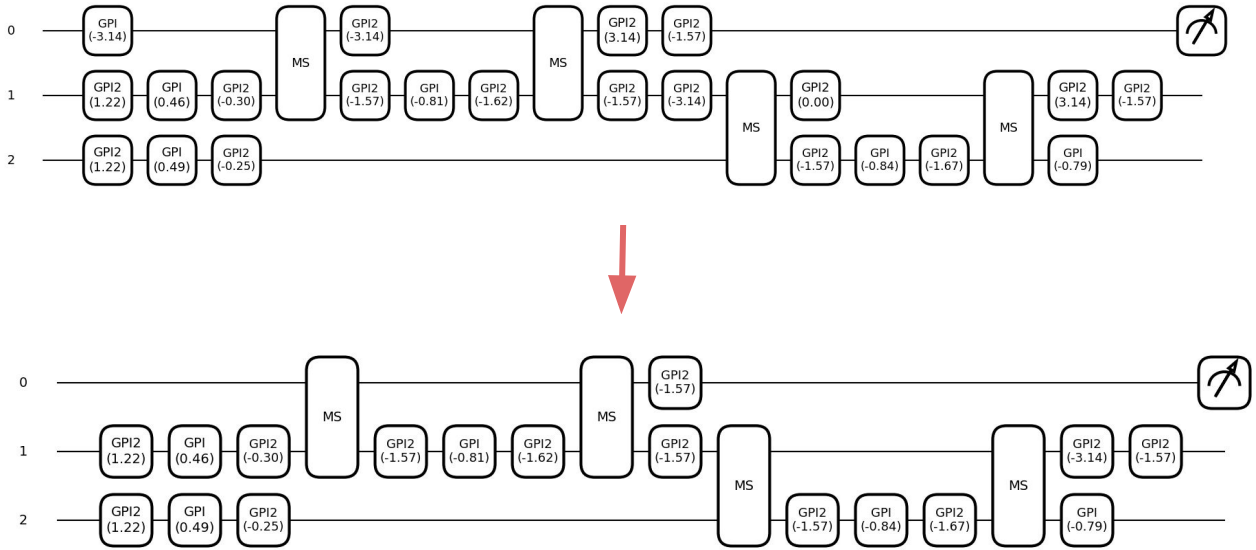
Optimization: fuse sequences of >3 single-qubit gates and apply circuit identities.



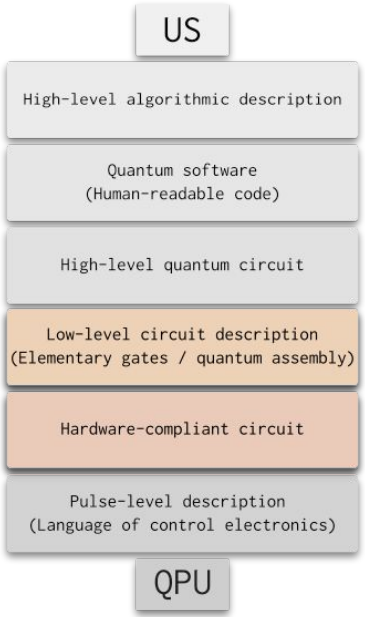
End-to-end example



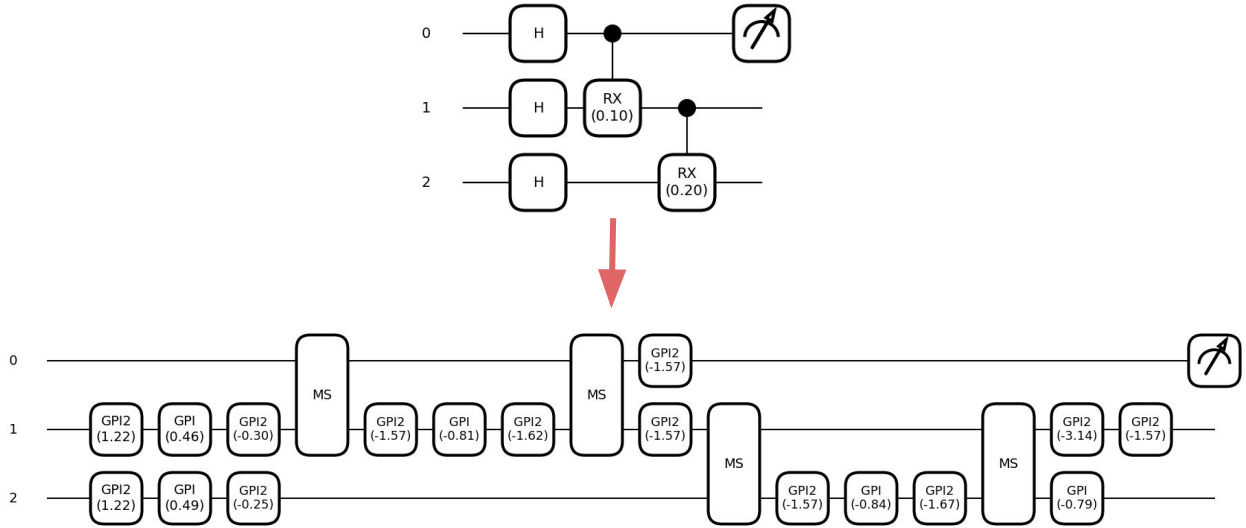
Repeat pushing/fusing/identity application.



End-to-end example



Final result



Q: How to choose the order of the passes?

Software tools

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

```
qiskit_circuit = QuantumCircuit(...)

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

```
@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.

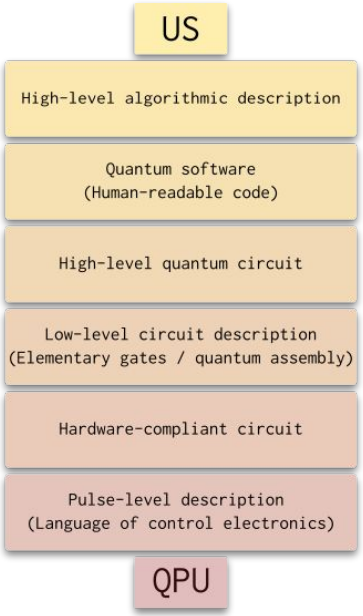
Challenge

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

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

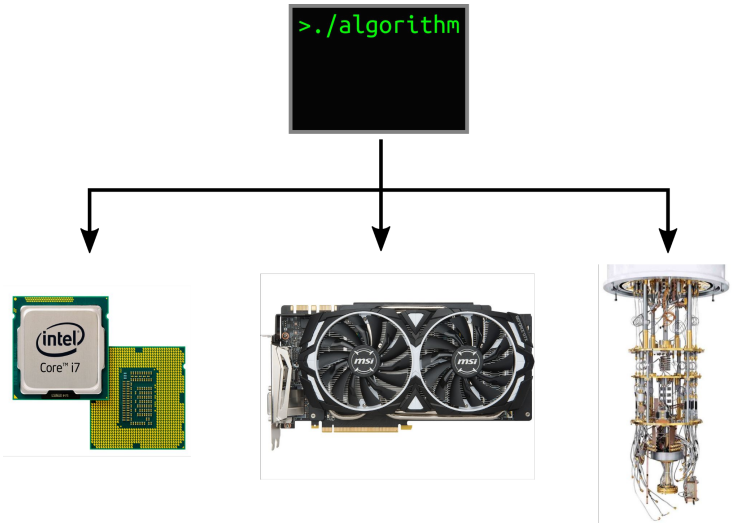
Need **automation** to scale up to (multi-QPU) devices w/1000s of qubits:

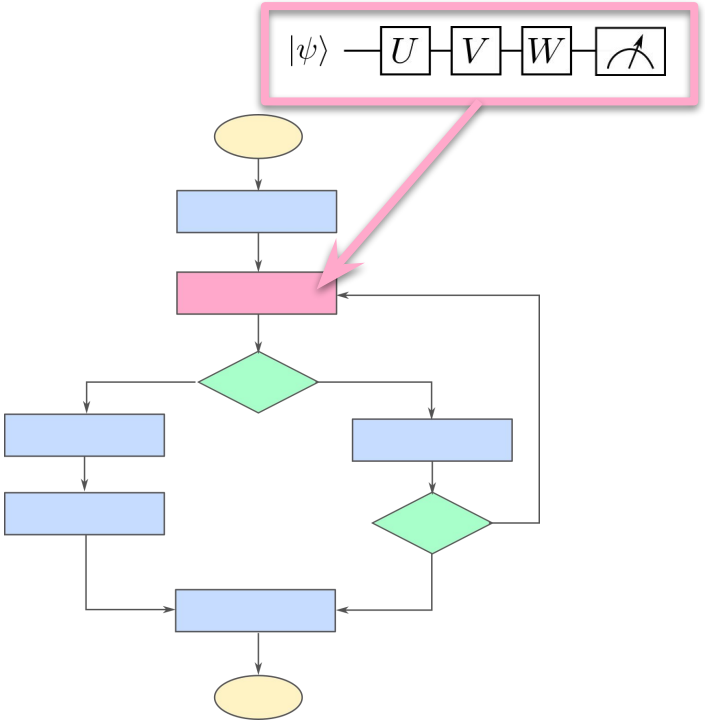
- circuit synthesis
- circuit optimization
- mapping, routing, and scheduling
- noise-aware techniques
- verification and debugging



Compiling classical and quantum code

Quantum algorithms don’t exist in a vacuum.

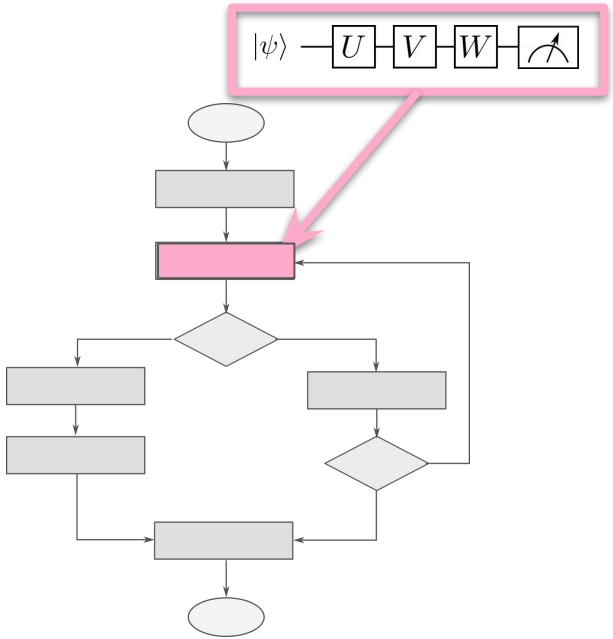




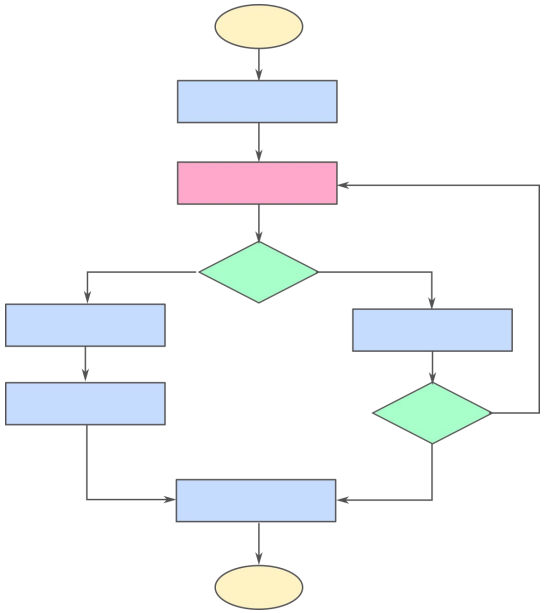
https://phys.org/news/2012-04-intel-batch-ivy-bridge-processors.html
https://www.techpowerup.com/img/17-03-28/65ee0f227ed0.jpg
(Photo/Justin Fantl, Rigetti Computing)

Compiling classical and quantum code

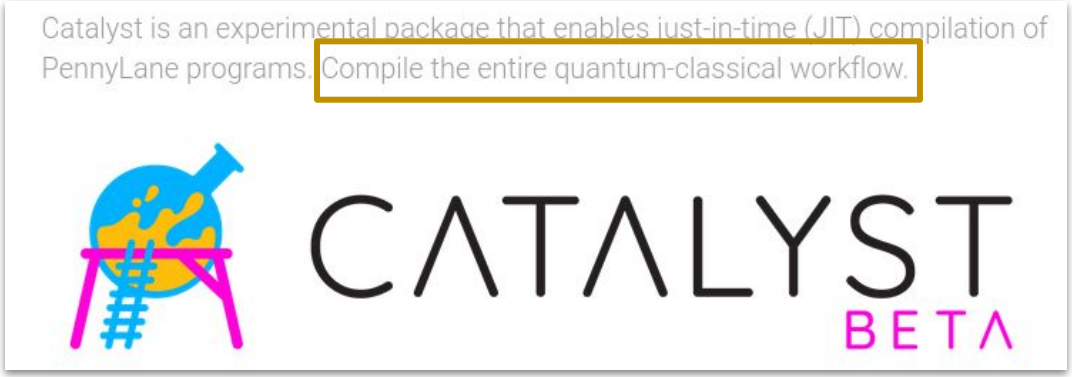
So far, we've addressed the problem of compiling this part:



Can we compile the whole thing?



Catalyst



https://docs.pennylane.ai/projects/catalyst/en/stable/
https://github.com/PennyLaneAI/catalyst

Just-in-time compilation

Ahead-of-time compilation

Compile program to executable **before** runtime

- done only once
- optimized; runs fast, but arch-specific
- executable code not human-readable

```
> g++ -o exec prog.cpp
> ./exec
```

Just-in-time compilation

Compile **at** runtime

- first execution takes longer (overhead of compilation)
- subsequent iterations reuse cached, compiled form of function

```
import jax

@jax.jit
def my_fast_function(a):
    z = a + 3
    y = z ** 2
    return z
```

Interpretation

Execute statements directly

- programs runs more slowly
- human-readable
- easier to debug

```
olivia➜~» python
>>> a = 3
>>> b = 4
>>> a + b
7
```

JIT compilation with tracing

```
import jax
import jax.numpy as jnp

def my_fast_function(a):
    print(a)
    z = jnp.sin(a) + 3
    y = z ** 2
    return z

print(jax.make_jaxpr(my_fast_function)(4.))
```

```
Traced<ShapedArray(float32[], weak_type=True)>with<DynamicJaxprTrace(level=1/0)>
{ lambda ; a:f32[]. let
    b:f32[] = sin a
    c:f32[] = add b 3.0
    _:f32[] = integer_pow[y=2] c
  in (c,) }
```

Example: teleportation

Next time

Content:

- Module 3

Action items:

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

Recommended reading for next class:

- Codebook QFT
- Nielsen and Chuang Ch. 5.1

```python
def shors_algorithm(N):
    p, q = 0, 0

    while p * q != N:
        a = np.random.choice(list(range(2, N - 1)))

        if np.gcd(a, N) != 1:
            p = np.gcd(a, N)
            q = N // p
            return p, q

        sample = get_sample(a, N)
        phase = fractional_binary_to_float(sample)
        candidate_order = phase_to_order(phase, N)

        if candidate_order % 2 == 0:
            square_root = (a ** (candidate_order // 2)) % N

            if square_root not in [1, N - 1]:
                p = np.gcd(square_root - 1, N)
                q = np.gcd(square_root + 1, N)

    return p, q
```