The Bitter Truth About Quantum Algorithms in the NISQ Era

(TU Wien, November 24th, 2021)

Prof. Dr. Dr. h.c. Frank Leymann

Kurt Gödel Visiting Professor TU Wien

Institut für Architektur von Anwendungssystemen (IAAS)
Universität Stuttgart

Leymann, Frank; Barzen, Johanna: The bitter truth about gate-based quantum algorithms in the NISQ era.
In: Quantum Science and Technology, 2020

Technological Problems

Decoherence : Qbits are not stable

⇒ State of a qbit decays over time (often, rather quick!)

→ Implementation of qbits disturb each other

⇒ Increasing number of qbits is quite difficult

Gate Infidelity : Each operation is (a bit) imprecise

⇒ Error of an algorithm increases with number of opertions

⇒ Only algorithms with "few" operations can be executed precisely

Readout Error: Measurement of a qbit is imprecise

⇒ Results are distorted

Qbit Connectivity : Not all qbits are physically connected

⇒ 2-qbit operations cannot be applied to arbitrary pairs of qbits

→ Reminder: 2-qbit operations are mandatory in a set of universal operations

⇒ Additional SWAP operations must be performed

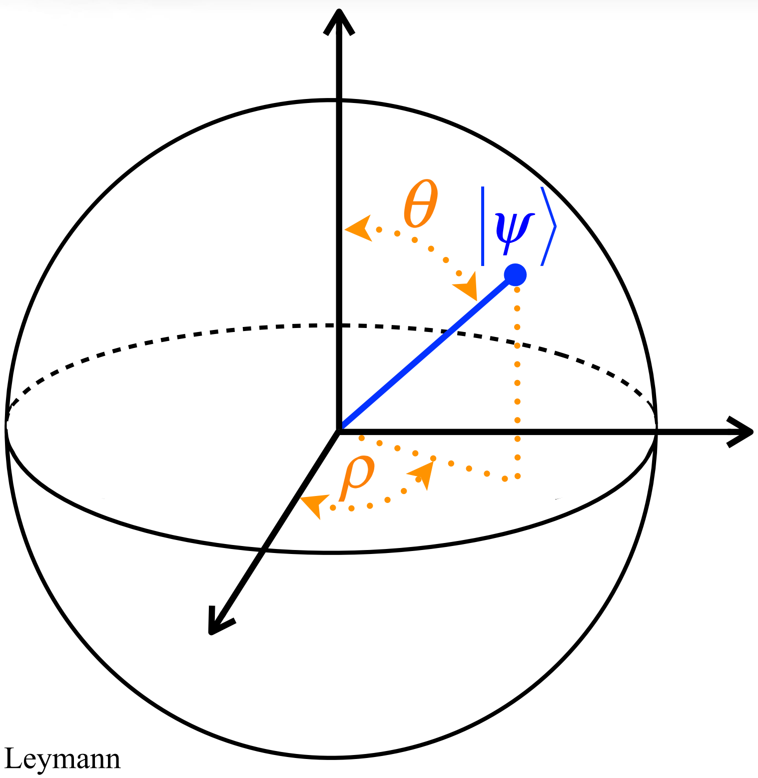
⇒ Number of operations of proper algorithms further limited

Decoherence

Bloch Sphere

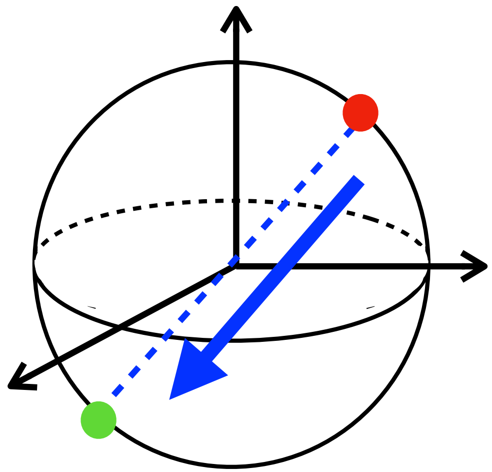
For $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ there is a $\theta \in [0, \pi]$ and a $\varrho \in [0, 2\pi]$, such that

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\rho}\sin\frac{\theta}{2}|1\rangle$$

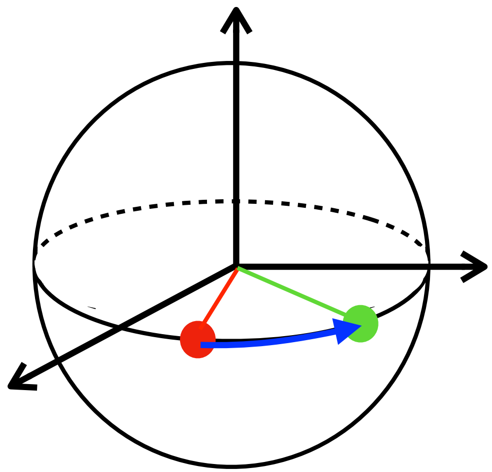


$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\rho}\sin\frac{\theta}{2}|1\rangle \mapsto (\theta, \rho)$$

Decoherence



T_1 (*relaxation time*) - collapse:
Transition into an orthogonal state



T_2 (*dephasing time*) - small disturbance:
Random change of phase

Non-Applicability of Classical Error Correction

Redundant codes (copies of qbits) cannot be created: **No-Cloning**!

A qbit will not change in a discrete manner (0 to 1, 1 to 0), but the amplitudes of superposition can be changed arbitrarily: **Continuous Errors**!

Reading means measurement, but this destroys the state, i.e. recovery of the original state is impossible: **Destructive Reads**!

Physical/Logical Qbits

Encoding ***1*** qbit by ***9*** qbits allows to detect and correct any (bit single) error!

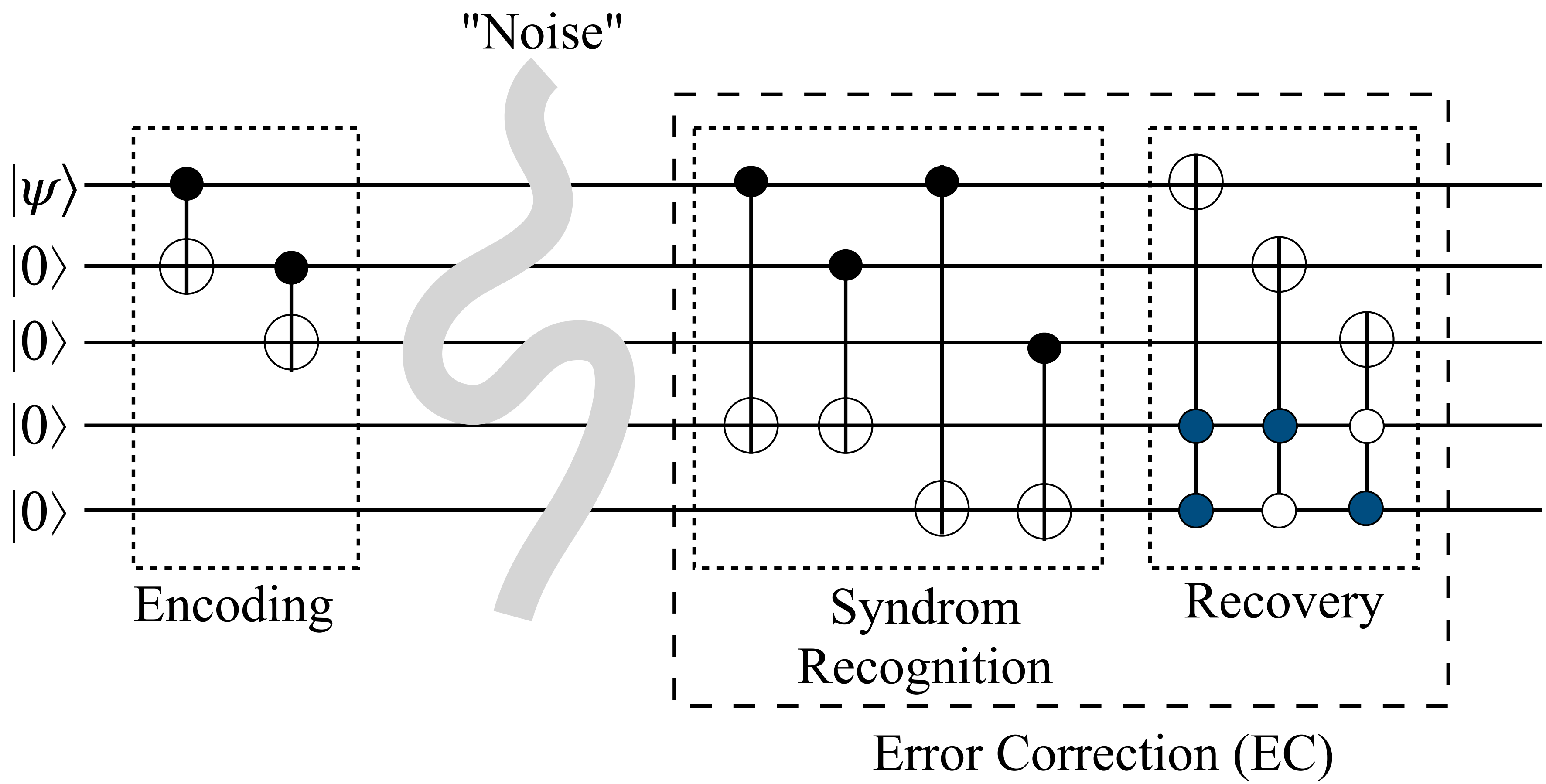
$$|0\rangle \mapsto \frac{\left(|000\rangle+|111\rangle\right)\cdot\left(|000\rangle+|111\rangle\right)\cdot\left(|000\rangle+|111\rangle\right)}{2\sqrt{2}}$$

$$|1\rangle \mapsto \frac{\left(|000\rangle-|111\rangle\right)\cdot\left(|000\rangle-|111\rangle\right)\cdot\left(|000\rangle-|111\rangle\right)}{2\sqrt{2}}$$

…and other encodings are possible. But:

Multiple noisy "physical" qbits needed to realize 1 stable "logical" qbit!

Error Correction of Qbits



Gate Fidelity

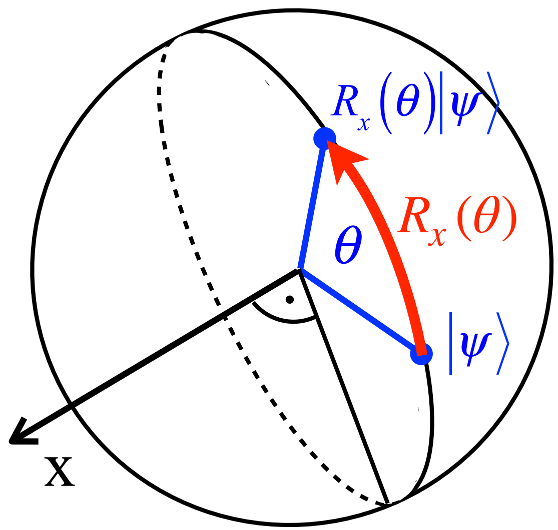
1-Qbit Operators: Decomposition

A set $\mathcal{U}$ of 1-qbit operators is called *universal* $:\Leftrightarrow$
Each 1-qbit operator is a finite combination of operators from $\mathcal{U}$

Let U be a 1-qbit operator. Then:

$$\exists\ \alpha,\beta,\gamma,\delta \in \mathbb{R}:\ U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

Gates are Inherent Imprecise



$R_x(\theta)$ is rotation by angle θ around x-axis

Exact rotation around an angle is in general impossible

⇒ Rotation is inherent imprecise

⇒ Each qbit operation $U = e^{i\alpha} R_z(\beta)\, R_y(\gamma)\, R_z(\delta)$ has a small error

Gate Errors

Applying the algorithm $U_T \circ \cdots \circ U_1$ to φ_0 results in φ_T:

$$\left|\varphi_T\right\rangle = U_T \circ \ldots \circ U_1 \left|\varphi_0\right\rangle$$

Each operation U_i is a bit imprecise, produces a small deviation from the exact result, i.e. instead of U_i an operation $\widetilde{U}_i$ is performed:

Thus, instead of $\left|\varphi_1\right\rangle = U_1 \left|\varphi_0\right\rangle$ the result $\tilde{U}_1 \left|\varphi_0\right\rangle = \left|\varphi_1\right\rangle + \left|E_1\right\rangle$ is produced (*Gate Error* or [lack of] *Gate Fidelity*)

I.e. the final computed result of the algorithm is:

$$\left|\tilde{\varphi}_T\right\rangle = \left|\varphi_T\right\rangle + \left|E_T\right\rangle + \tilde{U}_T \left|E_{T-1}\right\rangle + \tilde{U}_T \tilde{U}_{T-1} \left|E_{T-2}\right\rangle + \ldots + \tilde{U}_T \tilde{U}_{T-1} \ldots \tilde{U}_2 \left|E_1\right\rangle$$

Error Propagation

$$\left\| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \right\| \leq \left\| \, |E_T\rangle \, \right\| + \left\| \, |E_{T-1}\rangle \right\| + \left\| \, |E_{T-2}\rangle \right\| + \ldots + \left\| \, |E_1\rangle \right\|$$

Let ε be the maximum error of all gates : $\forall \; 1 \leq \mathrm{t} \leq \mathrm{T}: \left\| \left(\tilde{U}_t - U_t \right) \right\| \leq \varepsilon$

$$\Rightarrow \left\| \, |\tilde{\varphi}_T\rangle - |\varphi_T\rangle \, \right\| \leq T\varepsilon$$

The accumulated error grows linear with the length of the computation

Threshold Theorem

For any required precision of a computation C of a set of ideal gates, there is **an implementation C' based on fault tolerant** gates that computes the results of C within the required precision…

…if the fault tolerant gates fail less than a threshold η-times

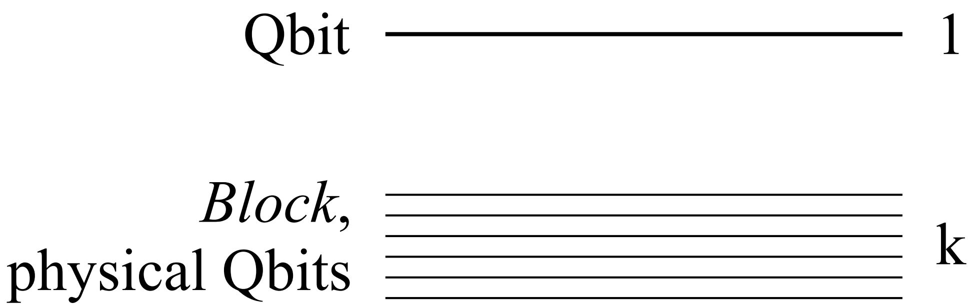
Today[*] (2019), $\eta \approx 10^{-2}$

⇒ Fault tolerance scales - in principle!

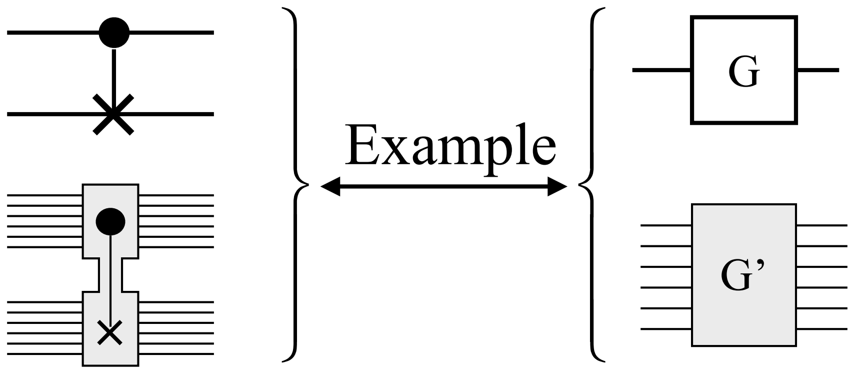
<u>N</u>oisy **<u>I</u>**ntermediate **<u>S</u>**cale **<u>Q</u>**uantum computing: NISQ

(*) https://en.wikipedia.org/wiki/Quantum_threshold_theorem

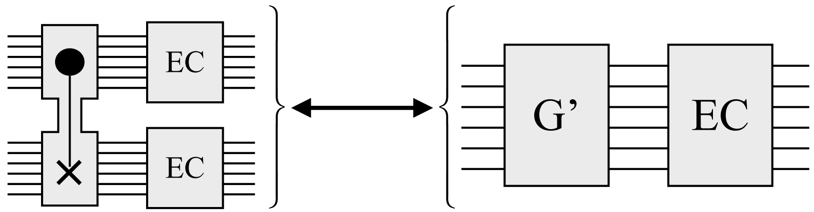
Fault-Tolerance: Principle



1 Qbit is encoded by
k error-correcting Qbits



Universal gate G is substituted by a *coded gate* G’
(coded gate G’ ist quantum subroutine implementing the functionality of G)



After executing a coded gate, error correction on affected blocks are run

Implication of Noise

N noisy "physical" qbits are needed to realize 1 stable "logical" qbit!

⇒ **More qbits needed** than estimated by theoretical algorithms

Single universal but noisy gate is realized by quantum subroutine!

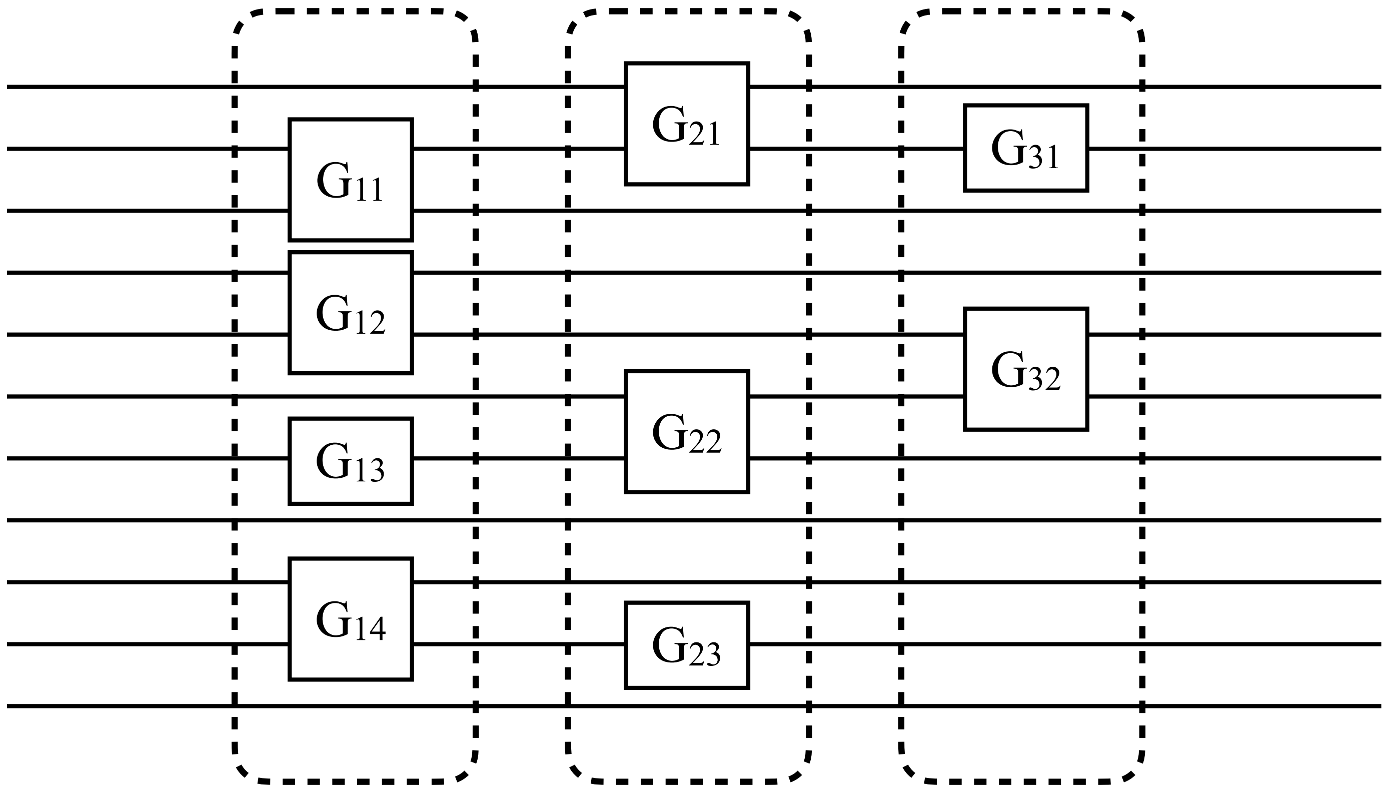
Error correction on noisy qbit is run periodically!

⇒ **More operations needed** than estimated by theoretical algorithms

Metrics of an Algorithm

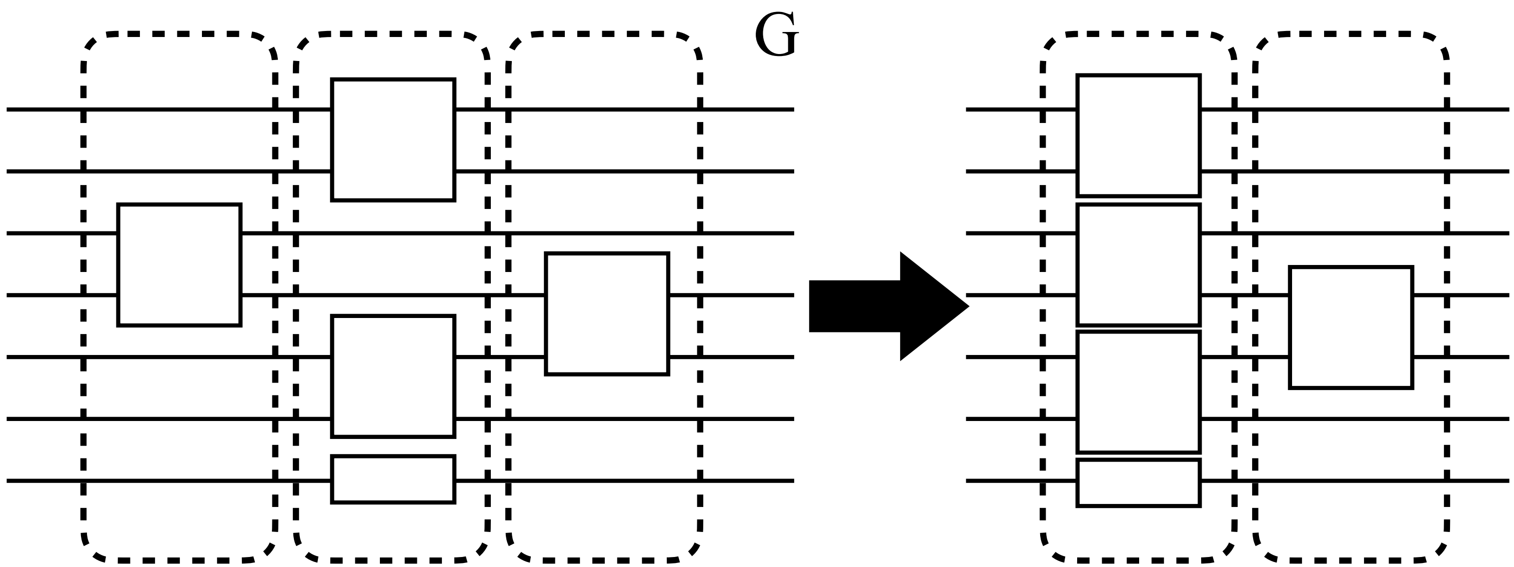
Depth and Width of an Algorithm

The *depth* of a quantum circuit is the number of layers of 1- or 2-qbit gates that operate in parallel on disjoint qbits.

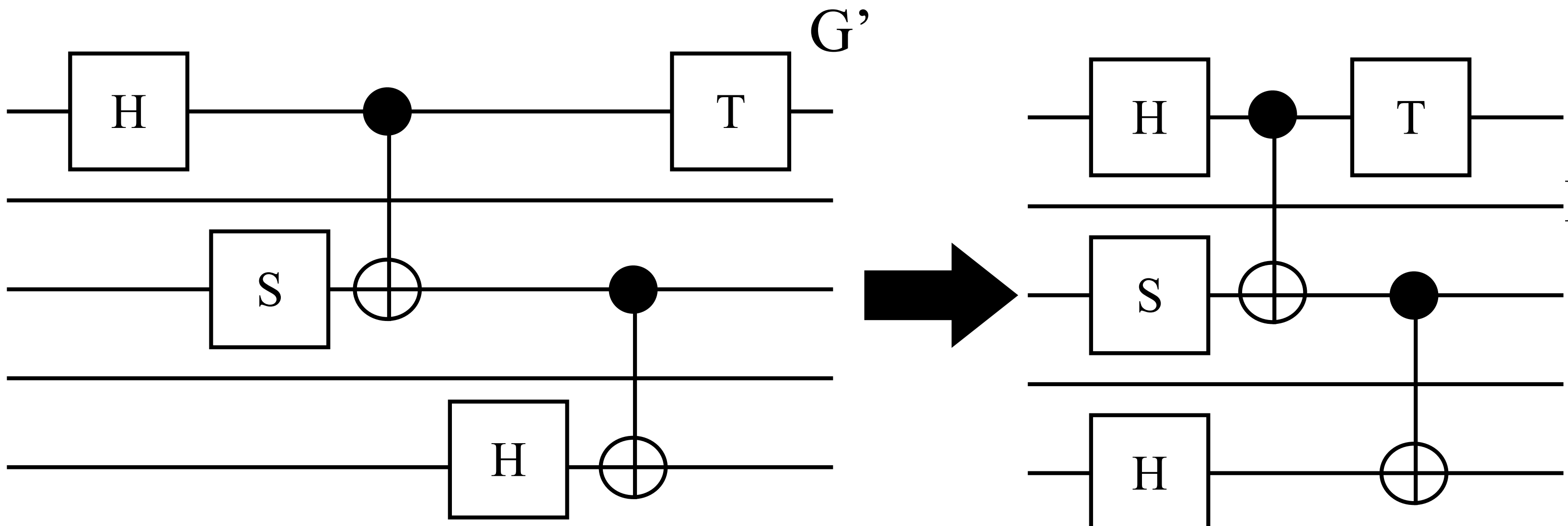


The *width* of a quantum circuit is the number of manipulated qbits.

Examples

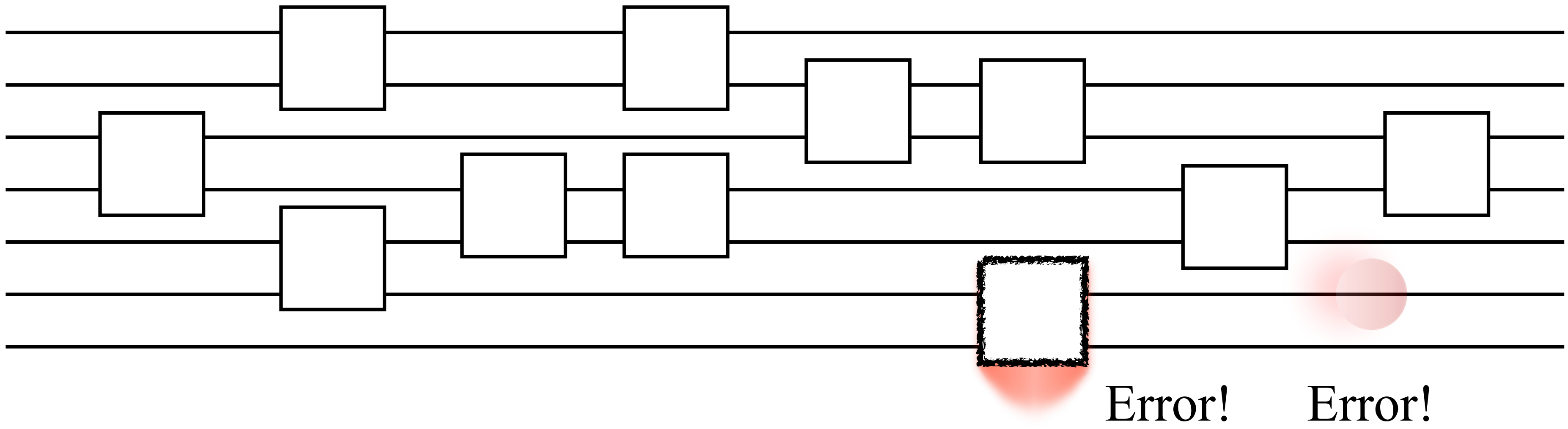


Depth(G) = 2
Width(G) = 7



Depth(G’) = 3
Width(G’) = 3

Noisy Algorithms



Rough estimation of the "size" of a quantum algorithm
that can be performed without errors:

$$wd \ll \frac{1}{\varepsilon}$$

w: width
d: depth
ε: error rate

Consequences

$$wd \ll \frac{1}{\varepsilon}$$

Deep quantum algorithms $\Rightarrow$ few qbits
$\Rightarrow$ efficient classical simulation possible

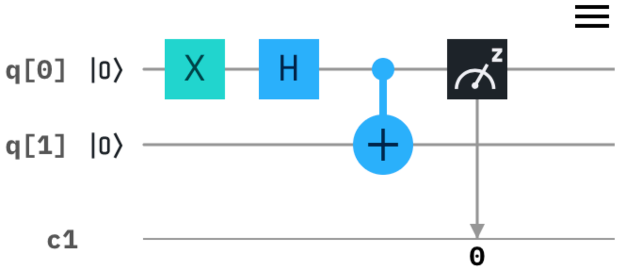
Shallow quantum algorithms $\Rightarrow$ many qbits
$\Rightarrow$ potential for *quantum advantage*

Transpilation

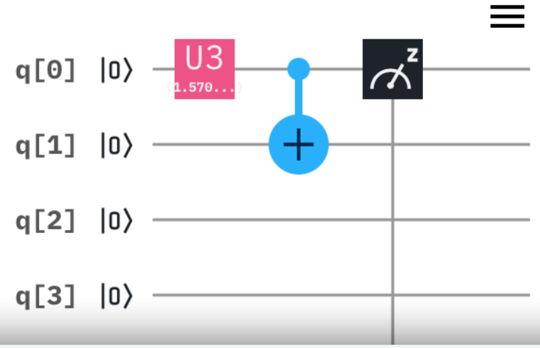
(a.k.a. Cross-Compilation)

Transpilation:
Mapping to Hardware Gates

Original circuit



Transpiled circuit



Original circuit

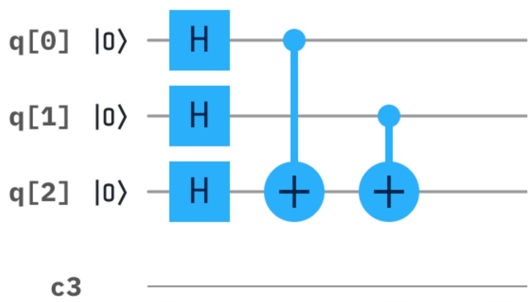
```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[1];

x q[0];
h q[0];
cx q[0],q[1];
measure q[0] -> c[0];
```

Transpiled circuit

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];
creg c[1];

u3(1.5707963267948968, 3.141592653589793, 3.141592653589793) q[0];
cx q[0], q[1];
measure q[0] -> c[0];

```

Transpilation: Increasing Depth

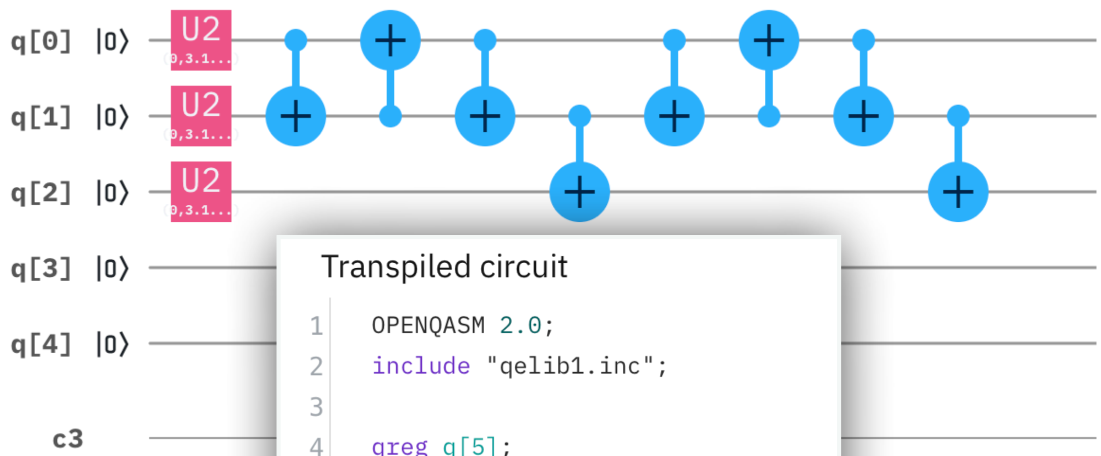
Original circuit



Original circuit

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[3];
creg c[3];

h q[0];
h q[1];
h q[2];
cx q[0],q[2];
cx q[1],q[2];
```

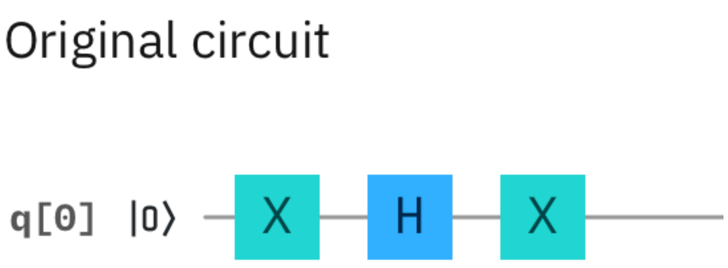
Transpiled circuit

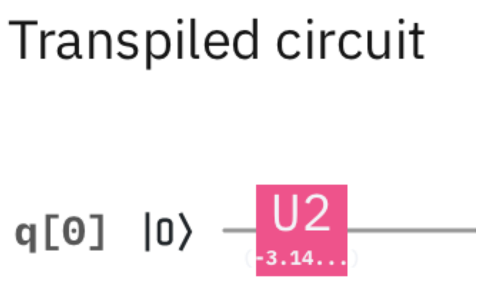


Transpiled circuit

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];
creg c[3];

u2(0, 3.141592653589793) q[0];
u2(0, 3.141592653589793) q[1];
cx q[0], q[1];
cx q[1], q[0];
cx q[0], q[1];
u2(0, 3.141592653589793) q[2];
cx q[1], q[2];
cx q[0], q[1];
cx q[1], q[0];
cx q[0], q[1];
cx q[1], q[2];
```

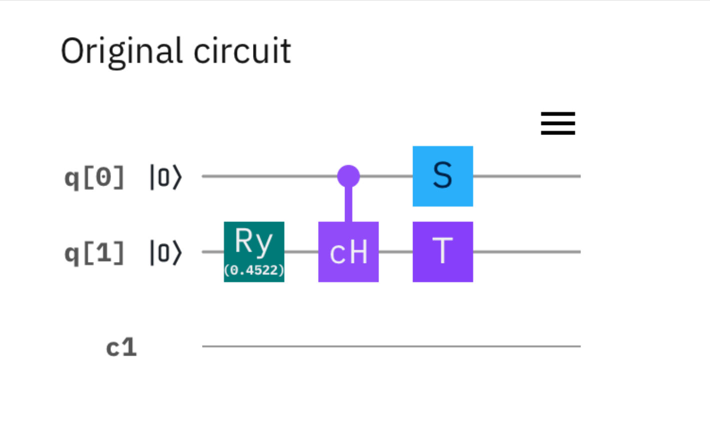
Transpilation: Decreasing Depth

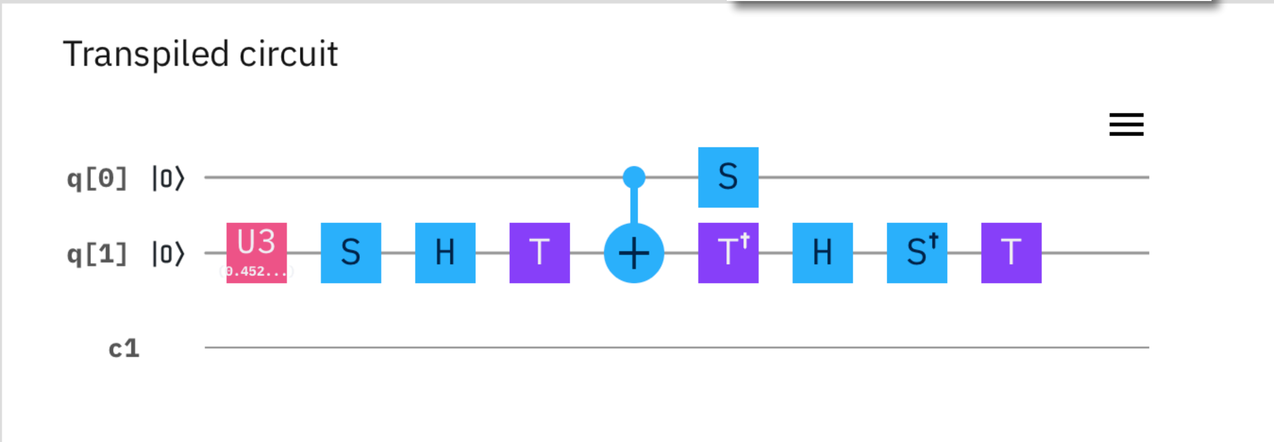




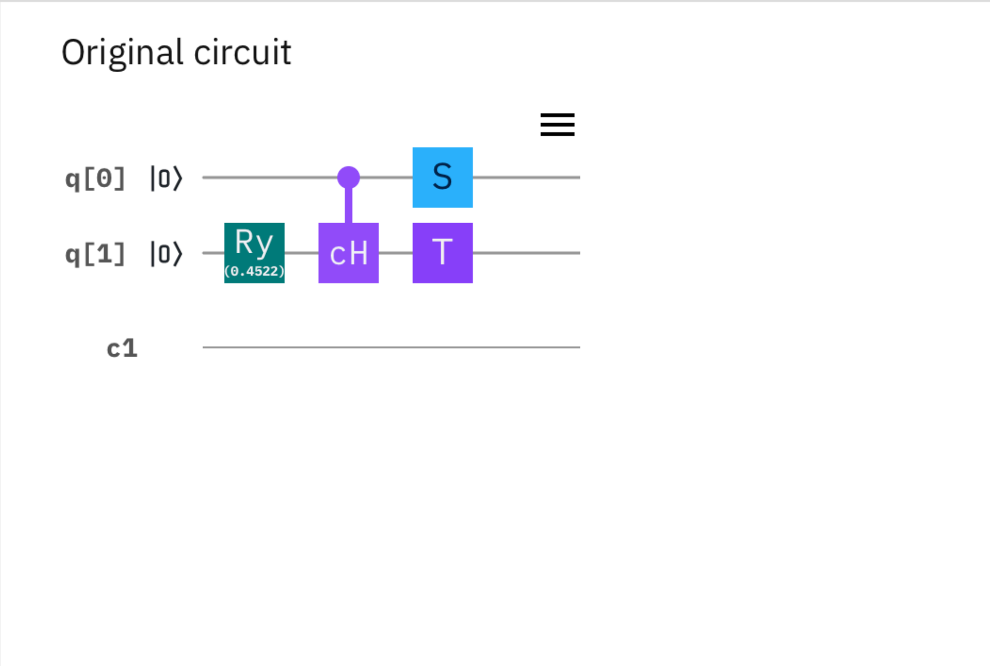
Transpilation

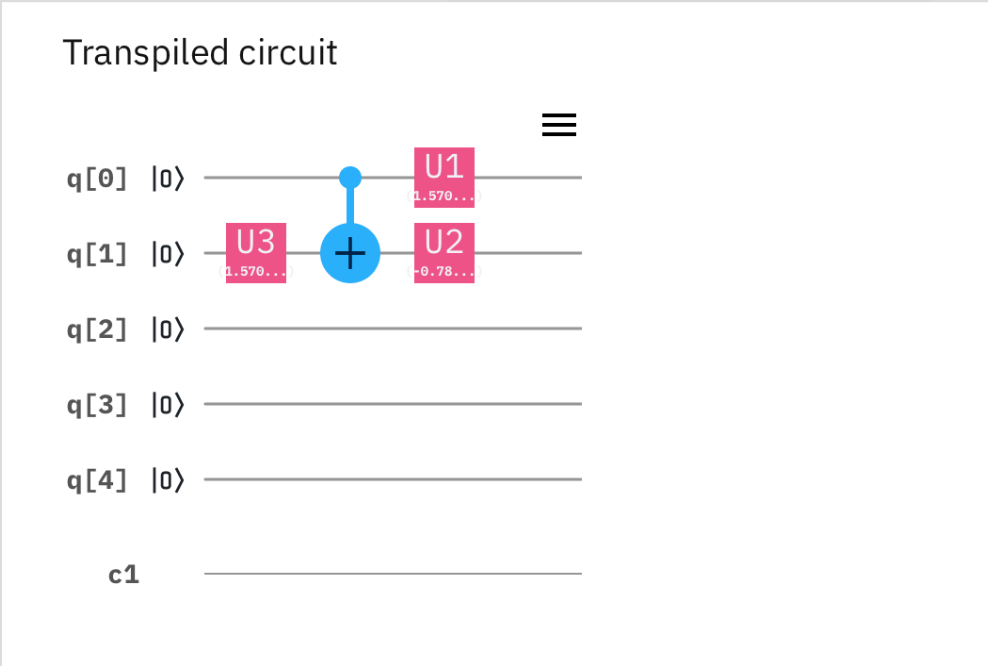
…on simulator





…on QPU





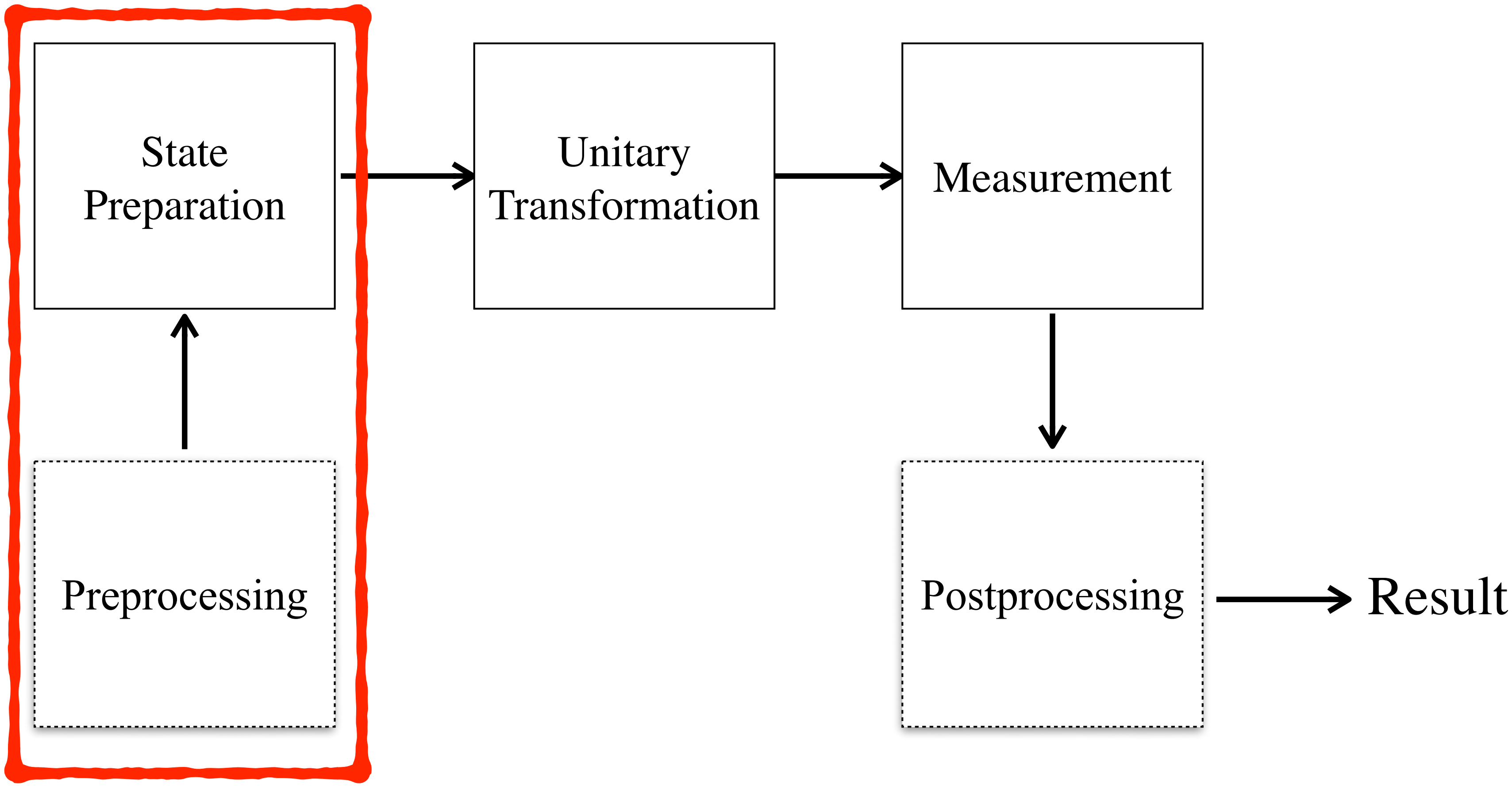
Circuit Rewrite: Implications

The depth of a circuit can often be reduced by "shifting gates to the left as far as possible", i.e. without sacrificing the data flow. This is mainly hardware independent.

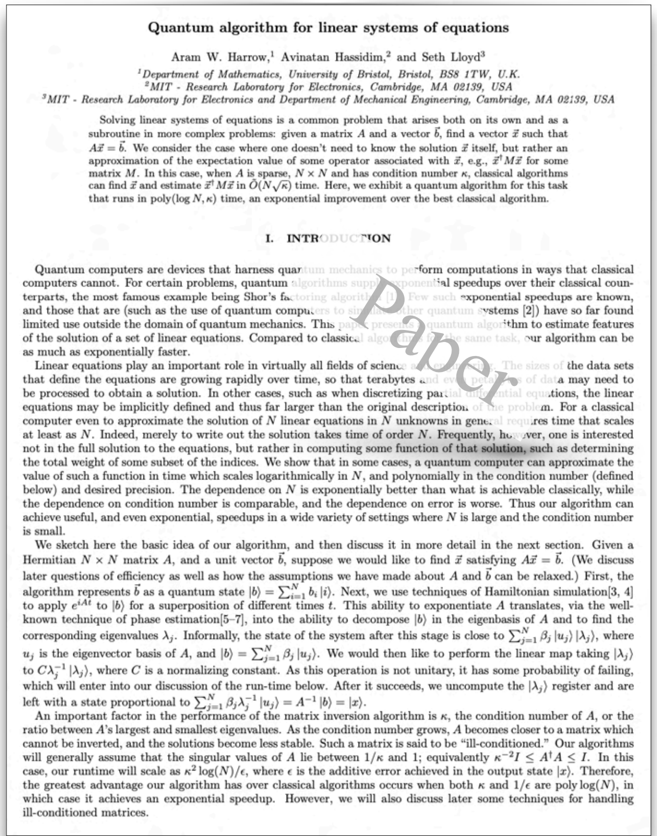
Hardware dependent rewrite is required, e.g. to map the gates of a hardware-independent circuit to the gates supported by the concrete hardware. This typically increases the depth of an algorithm (but may decrease it).
⇒ Inspection of transpiled circuit needed to assess executability.

Input Preparation

Reminder: Quantum Algorithm



Quantum Algorithm: Paper Version



Quantum algorithm for linear systems of equations

Aram W. Harrow,[1] Avinatan Hassidim,[2] and Seth Lloyd[3]

[1]*Department of Mathematics, University of Bristol, Bristol, BS8 1TW, U.K.*
[2]*MIT - Research Laboratory for Electronics, Cambridge, MA 02139, USA*
[3]*MIT - Research Laboratory for Electronics and Department of Mechanical Engineering, Cambridge, MA 02139, USA*

Solving linear systems of equations is a common problem that arises both on its own and as a subroutine in more complex problems: given a matrix A and a vector $\vec{b}$, find a vector $\vec{x}$ such that $A\vec{x} = \vec{b}$. We consider the case where one doesn't need to know the solution $\vec{x}$ itself, but rather an approximation of the expectation value of some operator associated with $\vec{x}$, e.g., $\vec{x}^\dagger M \vec{x}$ for some matrix M. In this case, when A is sparse, $N \times N$ and has condition number κ, classical algorithms can find $\vec{x}$ and estimate $\vec{x}^\dagger M \vec{x}$ in $\tilde{O}(N\sqrt{\kappa})$ time. Here, we exhibit a quantum algorithm for this task that runs in poly$(\log N, \kappa)$ time, an exponential improvement over the best classical algorithm.

I. INTRODUCTION

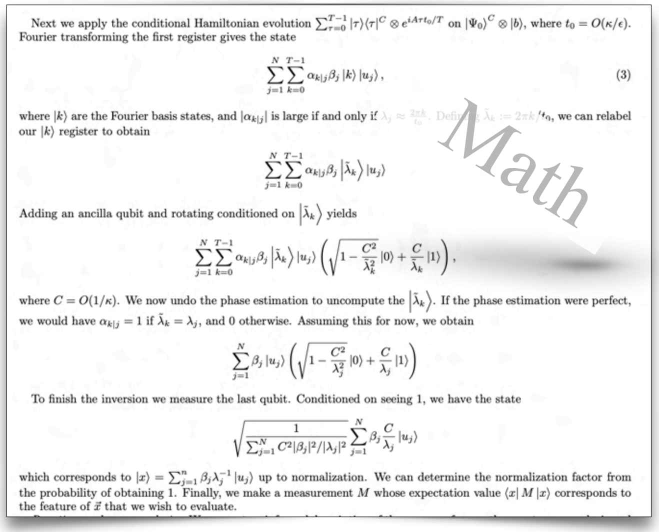
Quantum computers are devices that harness quantum mechanics to perform computations in ways that classical computers cannot. For certain problems, quantum algorithms supply exponential speedups over their classical counterparts, the most famous example being Shor's factoring algorithm [1]. Few such exponential speedups are known, and those that are (such as the use of quantum computers to simulate other quantum systems [2]) have so far found limited use outside the domain of quantum mechanics. This paper presents a quantum algorithm to estimate features of the solution of a set of linear equations. Compared to classical algorithms for the same task, our algorithm can be as much as exponentially faster.

Linear equations play an important role in virtually all fields of science and engineering. The sizes of the data sets that define the equations are growing rapidly over time, so that terabytes and even petabytes of data may need to be processed to obtain a solution. In other cases, such as when discretizing partial differential equations, the linear equations may be implicitly defined and thus far larger than the original description of the problem. For a classical computer even to approximate the solution of N linear equations in N unknowns in general requires time that scales at least as N. Indeed, merely to write out the solution takes time of order N. Frequently, however, one is interested not in the full solution to the equations, but rather in computing some function of that solution, such as determining the total weight of some subset of the indices. We show that in some cases, a quantum computer can approximate the value of such a function in time which scales logarithmically in N, and polynomially in the condition number (defined below) and desired precision. The dependence on N is exponentially better than what is achievable classically, while the dependence on condition number is comparable, and the dependence on error is worse. Thus our algorithm can achieve useful, and even exponential, speedups in a wide variety of settings where N is large and the condition number is small.

We sketch here the basic idea of our algorithm, and then discuss it in more detail in the next section. Given a Hermitian $N \times N$ matrix A, and a unit vector $\vec{b}$, suppose we would like to find $\vec{x}$ satisfying $A\vec{x} = \vec{b}$. (We discuss later questions of efficiency as well as how the assumptions we have made about A and $\vec{b}$ can be relaxed.) First, the algorithm represents $\vec{b}$ as a quantum state $|b\rangle = \sum_{i=1}^{N} b_i |i\rangle$. Next, we use techniques of Hamiltonian simulation[3, 4] to apply e^{iAt} to $|b\rangle$ for a superposition of different times t. This ability to exponentiate A translates, via the well-known technique of phase estimation[5–7], into the ability to decompose $|b\rangle$ in the eigenbasis of A and to find the corresponding eigenvalues λ_j. Informally, the state of the system after this stage is close to $\sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle$, where u_j is the eigenvector basis of A, and $|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$. We would then like to perform the linear map taking $|\lambda_j\rangle$ to $C\lambda_j^{-1} |\lambda_j\rangle$, where C is a normalizing constant. As this operation is not unitary, it has some probability of failing, which will enter into our discussion of the run-time below. After it succeeds, we uncompute the $|\lambda_j\rangle$ register and are left with a state proportional to $\sum_{j=1}^{N} \beta_j \lambda_j^{-1} |u_j\rangle = A^{-1}|b\rangle = |x\rangle$.

An important factor in the performance of the matrix inversion algorithm is κ, the condition number of A, or the ratio between A's largest and smallest eigenvalues. As the condition number grows, A becomes closer to a matrix which cannot be inverted, and the solutions become less stable. Such a matrix is said to be "ill-conditioned." Our algorithms will generally assume that the singular values of A lie between $1/\kappa$ and 1; equivalently $\kappa^{-2} I \le A^\dagger A \le I$. In this case, our runtime will scale as $\kappa^2 \log(N)/\epsilon$, where ϵ is the additive error achieved in the output state $|x\rangle$. Therefore, the greatest advantage our algorithm has over classical algorithms occurs when both κ and $1/\epsilon$ are poly $\log(N)$, in which case it achieves an exponential speedup. However, we will also discuss later some techniques for handling ill-conditioned matrices.

Quantum Algorithm



Next we apply the conditional Hamiltonian evolution $\sum_{\tau=0}^{T-1} |\tau\rangle\langle\tau|^C \otimes e^{iA\tau t_0/T}$ on $|\Psi_0\rangle^C \otimes |b\rangle$, where $t_0 = O(\kappa/\epsilon)$. Fourier transforming the first register gives the state

$$\sum_{j=1}^{N} \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j |k\rangle |u_j\rangle , \tag{3}$$

where $|k\rangle$ are the Fourier basis states, and $|\alpha_{k|j}|$ is large if and only if $\lambda_j \approx \frac{2\pi k}{t_0}$. Defining $\tilde{\lambda}_k := 2\pi k/t_0$, we can relabel our $|k\rangle$ register to obtain

$$\sum_{j=1}^{N} \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j \left|\tilde{\lambda}_k\right\rangle |u_j\rangle$$

Adding an ancilla qubit and rotating conditioned on $\left|\tilde{\lambda}_k\right\rangle$ yields

$$\sum_{j=1}^{N} \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j \left|\tilde{\lambda}_k\right\rangle |u_j\rangle \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda}_k^2}} |0\rangle + \frac{C}{\tilde{\lambda}_k} |1\rangle \right),$$

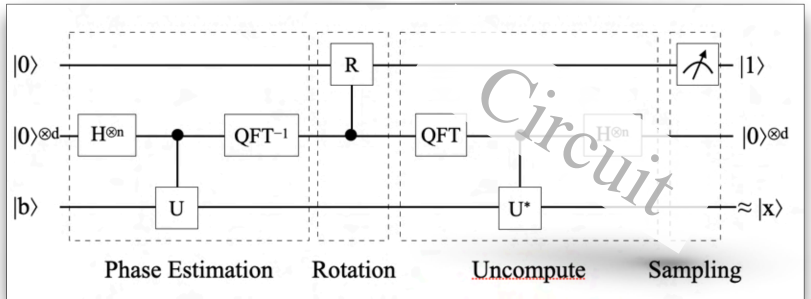
where $C = O(1/\kappa)$. We now undo the phase estimation to uncompute the $\left|\tilde{\lambda}_k\right\rangle$. If the phase estimation were perfect, we would have $\alpha_{k|j} = 1$ if $\tilde{\lambda}_k = \lambda_j$, and 0 otherwise. Assuming this for now, we obtain

$$\sum_{j=1}^{N} \beta_j |u_j\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)$$

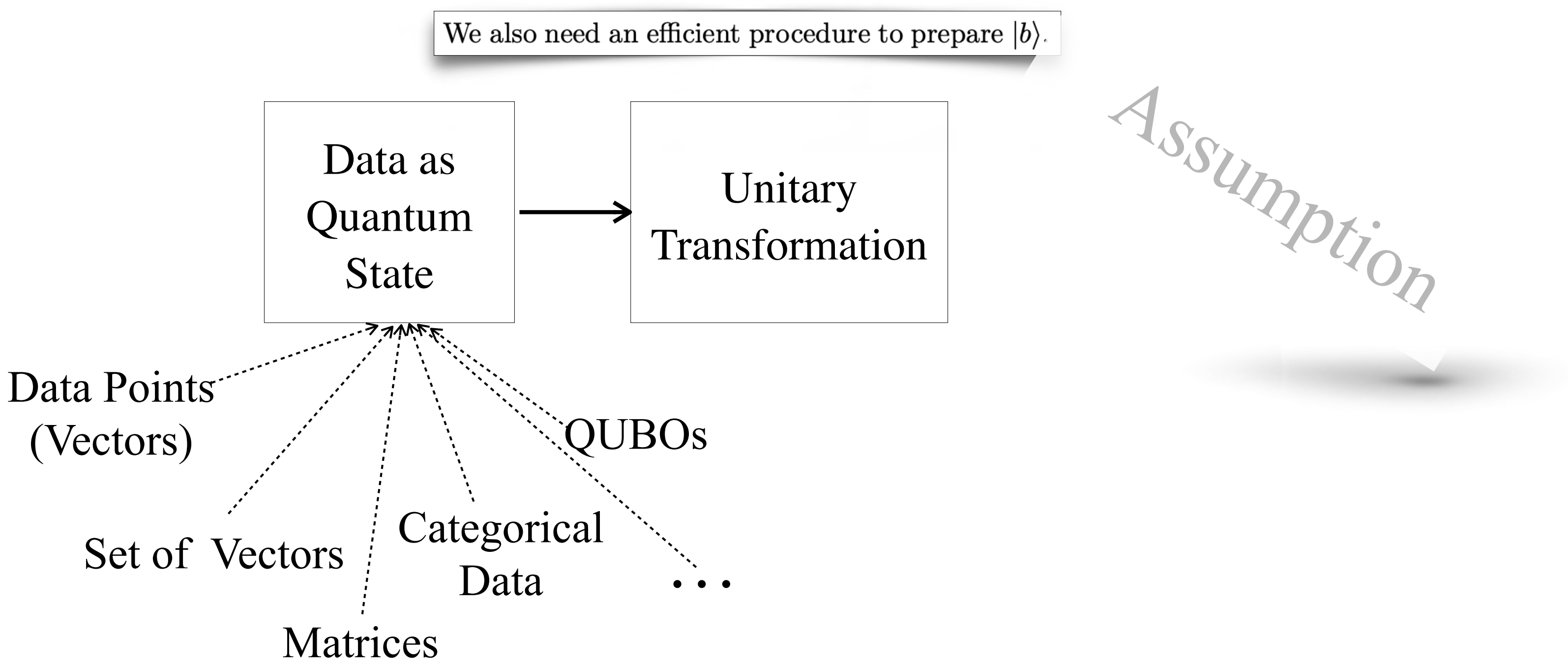
To finish the inversion we measure the last qubit. Conditioned on seeing 1, we have the state

$$\sqrt{\frac{1}{\sum_{j=1}^{N} C^2 |\beta_j|^2 / |\lambda_j|^2}} \sum_{j=1}^{N} \beta_j \frac{C}{\lambda_j} |u_j\rangle$$

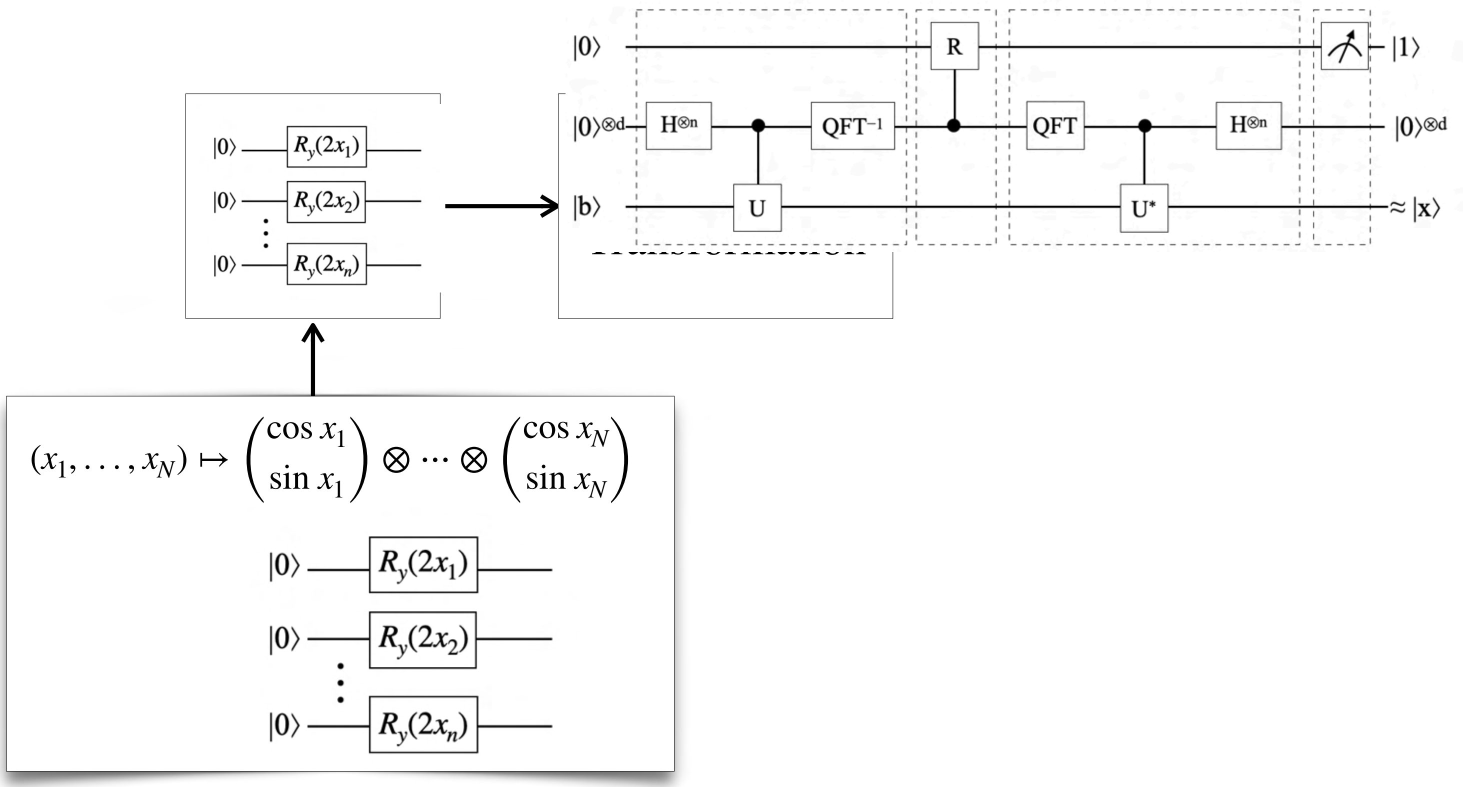
which corresponds to $|x\rangle = \sum_{j=1}^{n} \beta_j \lambda_j^{-1} |u_j\rangle$ up to normalization. We can determine the normalization factor from the probability of obtaining 1. Finally, we make a measurement M whose expectation value $\langle x| M |x\rangle$ corresponds to the feature of $\vec{x}$ that we wish to evaluate.



Data for the Algorithm



Data as Quantum State



State Preparation

Various possibilities (each with pros and cons), e.g.:

- Basis Encoding
- Amplitude Encoding
- Tensor product encoding
- Schmidt encoding

Corresponds to two categories

- Digital encoding
  - …for performing arithmetics
- Analog encoding
  - …for processing in high-dimensional feature spaces

Basis Encoding

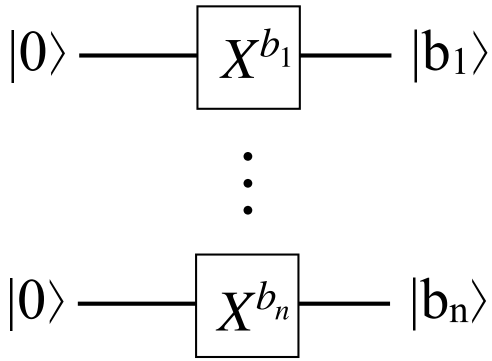
Let $x \in \mathbb{N}$

Then, x will be binary encoded, i.e. $(x_1,\ldots,x_n) \in \{0,1\}^n$ with $\Sigma x_k 2^k = x$

$x \mapsto |x_1,\ldots,x_n\rangle$ is called *Basis Encoding* of $x \in \mathbb{N}$

Base encoding is a representative of *digital encodings*

Basis Encoding: Circuit



Resources for encoding n bits:

- n qbits
- n gates
- depth 1
- 0 ancillae

Obviously, this circuit can be <u>generated</u> in a preprocessing step:

$$X^{b_1} \otimes \cdots \otimes X^{b_n} |0 \cdots 0\rangle$$

Basis Encoding: Real Numbers

A number $x \in \mathbb{R}$ is approximated in binary representation to k decimal places:

$$x \approx \sum_{i=0}^{n} b_i 2^i + \sum_{i=1}^{k} b_{-i} \cdot \frac{1}{2^i}$$

E.g. let x=1.7 and k=4, i.e. $x = 1 \cdot 2^0 + \sum_{i=1}^{4} b_{-i} \cdot \frac{1}{2^i}$

…next, compute decimal places:

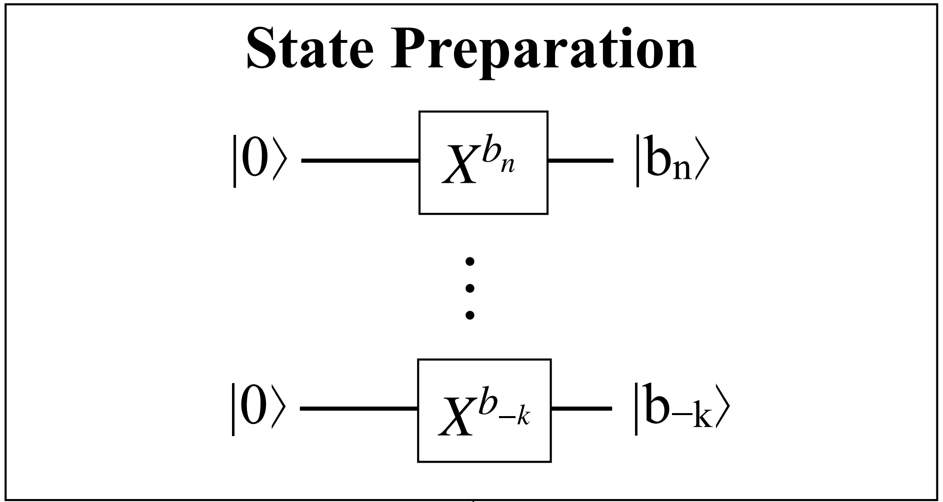
$$\begin{aligned}
0.7 \cdot 2 &= \mathbf{1}.4 \\
0.4 \cdot 2 &= \mathbf{0}.8 \\
0.8 \cdot 2 &= \mathbf{1}.6 \\
0.6 \cdot 2 &= \mathbf{1}.2 \\
0.2 \cdot 2 &= \mathbf{0}.4 \\
0.4 \cdot 2 &= \ldots
\end{aligned}$$

…periodic…

$$\longrightarrow \quad 0.7_{10} = 1\overline{0110}_2$$

…i.e. 1.7 approximated to 4 decimal places: **1 1011**

Input Preparation of Real Numbers: Base Encoding



Preprocessing

Compute binary representation of x

$$x \approx \sum_{i=0}^{n} b_i 2^i + \sum_{i=1}^{k} b_{-i} \cdot \frac{1}{2^i}$$

Generate corresponding circuit

$$X^{b_n} \otimes \cdots \otimes X^{b_{-k}} |0 \cdots 0\rangle$$

(Note: signs w.l.o.g. not considered)

Basis Encoding of Real Vectors

$$\text{Let} \quad x = \begin{pmatrix} -0.7 \\ 0.1 \\ 0.2 \end{pmatrix} \in \mathbb{R}^3$$

The sign of a number is represented by a leading 1 ("−") or 0 ("+")

I.e. (4 decimal places): $-0.7_{10} = 1\ 1011_2$ $\quad +0.1_{10} = 0\ 1001_2$ $\quad +0.2_{10} = 0\ 0011_2$

Thus,

$$x = \begin{pmatrix} -0.7 \\ 0.1 \\ 0.2 \end{pmatrix} \mapsto \begin{pmatrix} 11011 \\ 01011 \\ 00011 \end{pmatrix} \mapsto |11011\ \ 01001\ \ 00011\rangle = |x\rangle$$

(It's obvious how to generalize the preparation method for real numbers in base encoding to real vectors in base encoding)

Basis Encoding of Data Sets

Let $D = \{x_1,\ldots,x_m\}$ be a data set to be processed by a quantum algorithm

Representation of D as a quantum state: $|D\rangle = \frac{1}{\sqrt{m}} \sum_{i=1}^{m} |x_i\rangle$

<u>Example</u>: $x_1 = |101\rangle$ and $x_2 = |011\rangle$, then $|D\rangle = \frac{1}{\sqrt{2}} \left(|101\rangle + |011\rangle \right)$

…as a amplitude vector: $|D\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$

I.e. state vectors of binary data sets are typically sparse vectors

Amplitude Encoding

Let $x=(x_1,\ldots,x_N) \in \mathbb{R}^N$ be a unit-length vector, $N = 2^n$ $(\Rightarrow |x_i| \leq 1 \ \forall i)$

$x \mapsto \Sigma x_i|i\rangle$ is called *amplitude encoding* of $x \in \mathbb{R}^N$

Required qubits: $\lceil \log_2 N \rceil$

Required gates: 4^N

The amplitude encoding is an *analog encoding*

For $x \in \mathbb{R}^N$, $N \neq 2^n$, use a proper embedding (called *padding*):

$x \mapsto (x,0) \in \mathbb{R}^N \times \mathbb{R}^M$, $(N+M) = 2^n$, for the smallest possible n

Amplitude Encoding of Non-Unit-Length Vectors

For $x \in \mathbb{R}^N \setminus \{0\}$ the encoding is $x \mapsto \sum \frac{x_i}{\| x \|} |i\rangle$

Note: a matrix $A \in \mathbb{R}^{n \times m}$ can be represented as vector in $\mathbb{R}^{nm}$

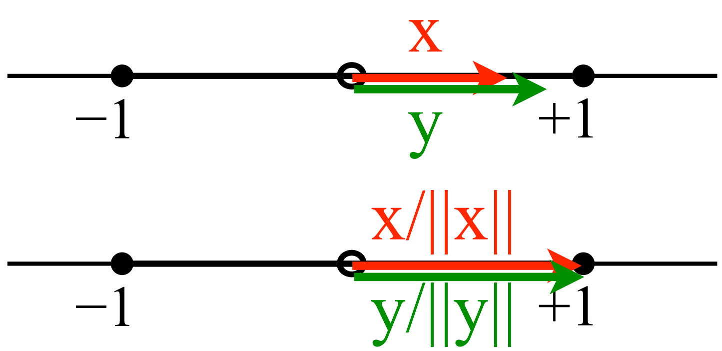
$$|A\rangle = \sum \frac{a_{ij}}{\| A \|} |i\rangle |j\rangle$$

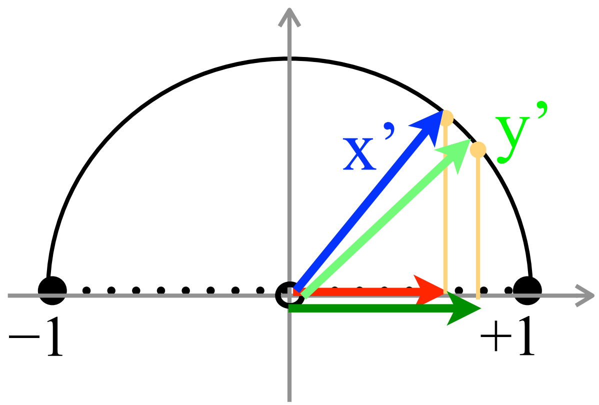
The $\| \, . \, \|$ may be computed classically as preprocessing step

Normalization and "Neighborhood"

Normalizing the members set $D \subseteq \mathbb{R}^N$ changes the relation between the members

- …which must be considered in certain algorithms (e.g. clustering)





(Tensor) Product Encoding

Let $x=(x_1,\dots,x_N) \in \mathbb{R}^N$ be a unit-length vector ($\Rightarrow |x_i| \leq 1 \ \forall i$)

Each x_i is represented by a separate qbit:

$$x_i \mapsto \cos x_i \cdot|0\rangle + \sin x_i \cdot|1\rangle$$

Then, $$x \mapsto \begin{pmatrix} \cos x_1 \\ \sin x_1 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} \cos x_N \\ \sin x_N \end{pmatrix}$$

Required qubits: N
Required gates: N

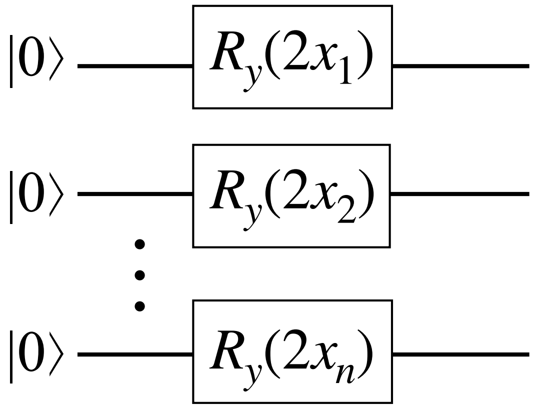
is called *(tensor) product encoding* of x (a.k.a. *angle encoding*)

Product encoding is a representative of an *analog encoding*

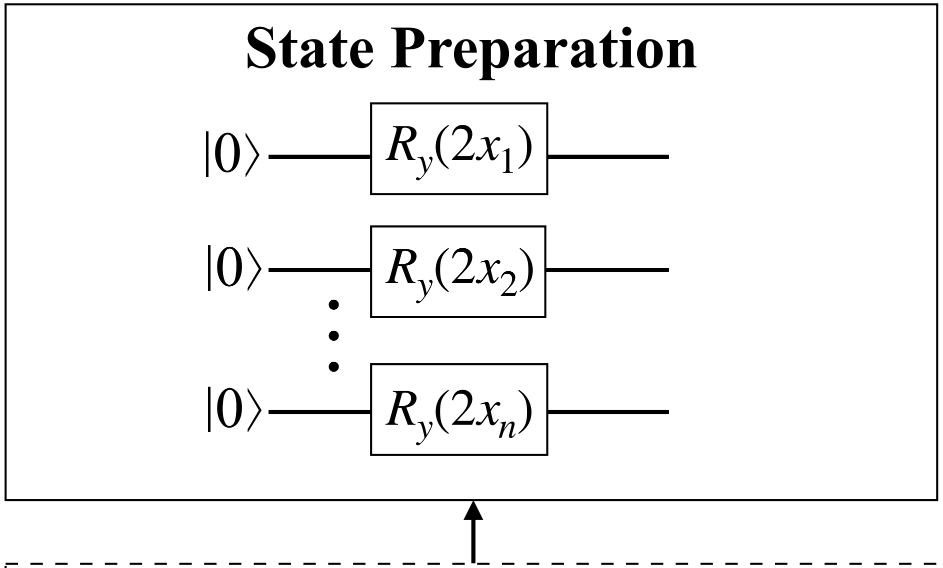
Circuit for Product Encoding

$$R_y(2x) = \begin{pmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{pmatrix} \quad \Rightarrow \quad R_y(2x)\,|0\rangle = \cos x \cdot |0\rangle + \sin x \cdot |1\rangle$$

Thus: $$\left(\bigotimes_{i=1}^{n} R_y(2x_i) \right) |0 \cdots 0\rangle = \begin{pmatrix} \cos x_1 \\ \sin x_1 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} \cos x_n \\ \sin x_n \end{pmatrix}$$



Input Preparation of Real Vectors: Product Encoding



Preprocessing

Generate the circuit

$$\left(\bigotimes_{i=1}^{n} R_y(2x_i)\right)|0\cdots0\rangle$$

Schmidt Decomposition

Let x ∈ V ⊗ W. There exist ONB $\{u_j\} \subseteq V$ and $\{v_j\} \subseteq W$ such that:

$$x = \sum_{i=1}^{K} \lambda_i \cdot u_i \otimes v_i$$

mit $\lambda_i>0$ and $\sum \lambda_i = 1$.

λ_i are called *Schmidt Coefficients* of v, K is called *Schmidt Number* of v (a.k.a.: *Schmidt Rank*)

Schmidt Decomposition via Singular Value Decomposition

Split the quantum register R into two parts: $R = V \otimes W$

Choose ONB $\{e_i\}$ and $\{f_j\}$ for V and W

Represent x as $x = \sum_{i,j} \beta_{ij} \cdot e_i \otimes f_j$

Compute the *singular value decomposition* of $M = (\beta_{ij})$: $M = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} A \\ 0 \end{pmatrix} V^*$

Choose the column vectors of $U_1 \rightarrow \{u_1,\ldots,u_K\}$

Choose the column vectors of $V \rightarrow \{v_1,\ldots,v_K\}$

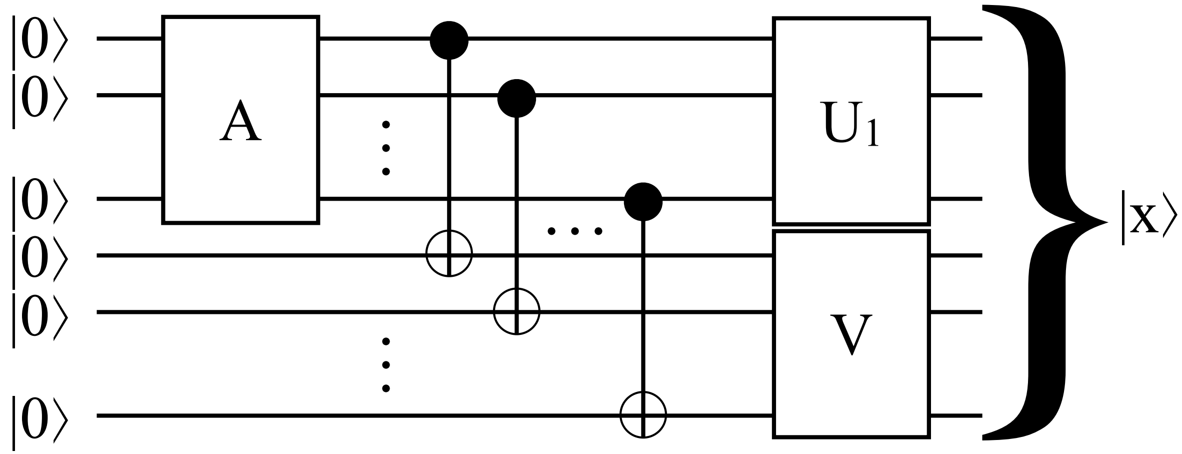
$A = \mathrm{diag}(\lambda_1,\ldots\lambda_K)$

A, U_1, V

$$\Rightarrow \quad x = \sum_{i=1}^{K} \lambda_i \cdot u_i \otimes v_i$$

State Preparation Based On Schmidt Decomposition

$$x = \sum_{i,j} \beta_{ij} \cdot e_i \otimes f_i \text{ , then SVD: } (\beta_{ij}) = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} A \\ 0 \end{pmatrix} V^*$$



U_1, V, A have to be composed of 1-qbit & 2-qbit operations:

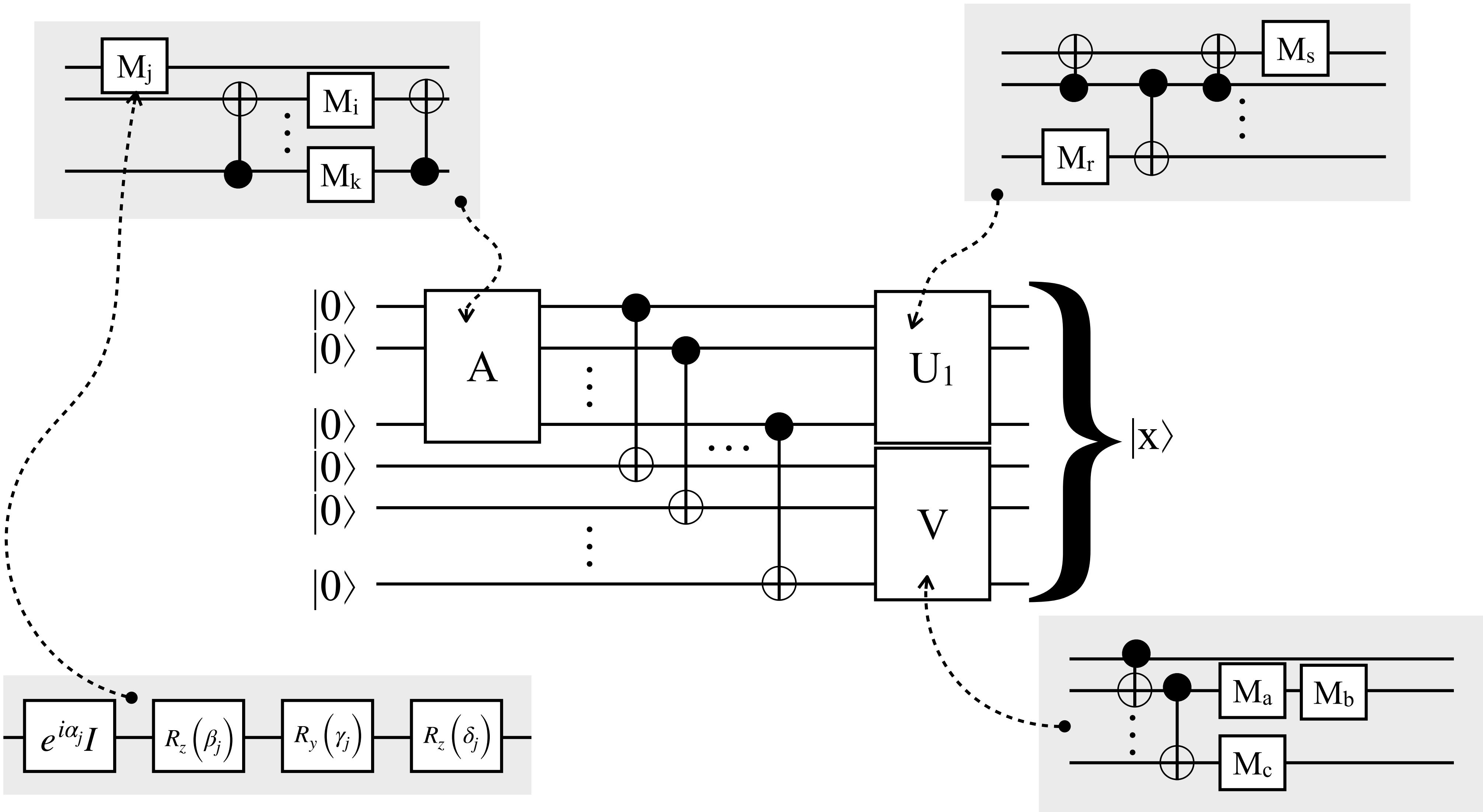
→ $U_1 = M_1 \otimes \ldots \otimes M_r$, $V = M_{r+1} \otimes \ldots \otimes M_{r+s}$, $A = M_{r+s+1} \otimes \ldots \otimes M_{r+s+t}$

where each M_i is a 1-qbit gate or a CNOT

…and each of the 1-qbit gates is represented as rotations:

$$M_j = e^{i\alpha_j} R_z\left(\beta_j\right) R_y\left(\gamma_j\right) R_z\left(\delta_j\right)$$

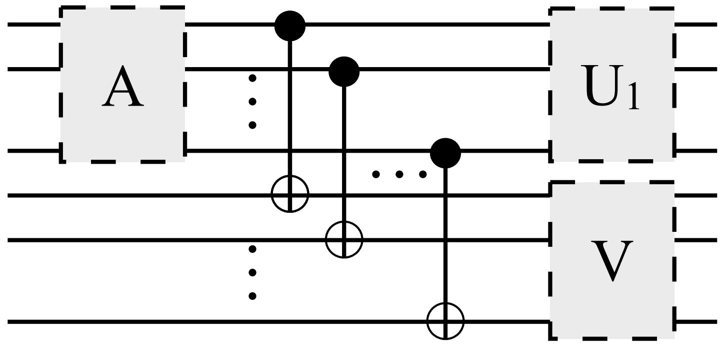
Refinement



M_j
M_i
M_k
M_s
M_r
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
A
U_1
V
$|x\rangle$
M_a
M_b
M_c
$e^{i\alpha_j}I$
$R_z(\beta_j)$
$R_y(\gamma_j)$
$R_z(\delta_j)$

Input Preparation of Real Vectors: Schmidt Decomposition

State Preparation



Preprocessing

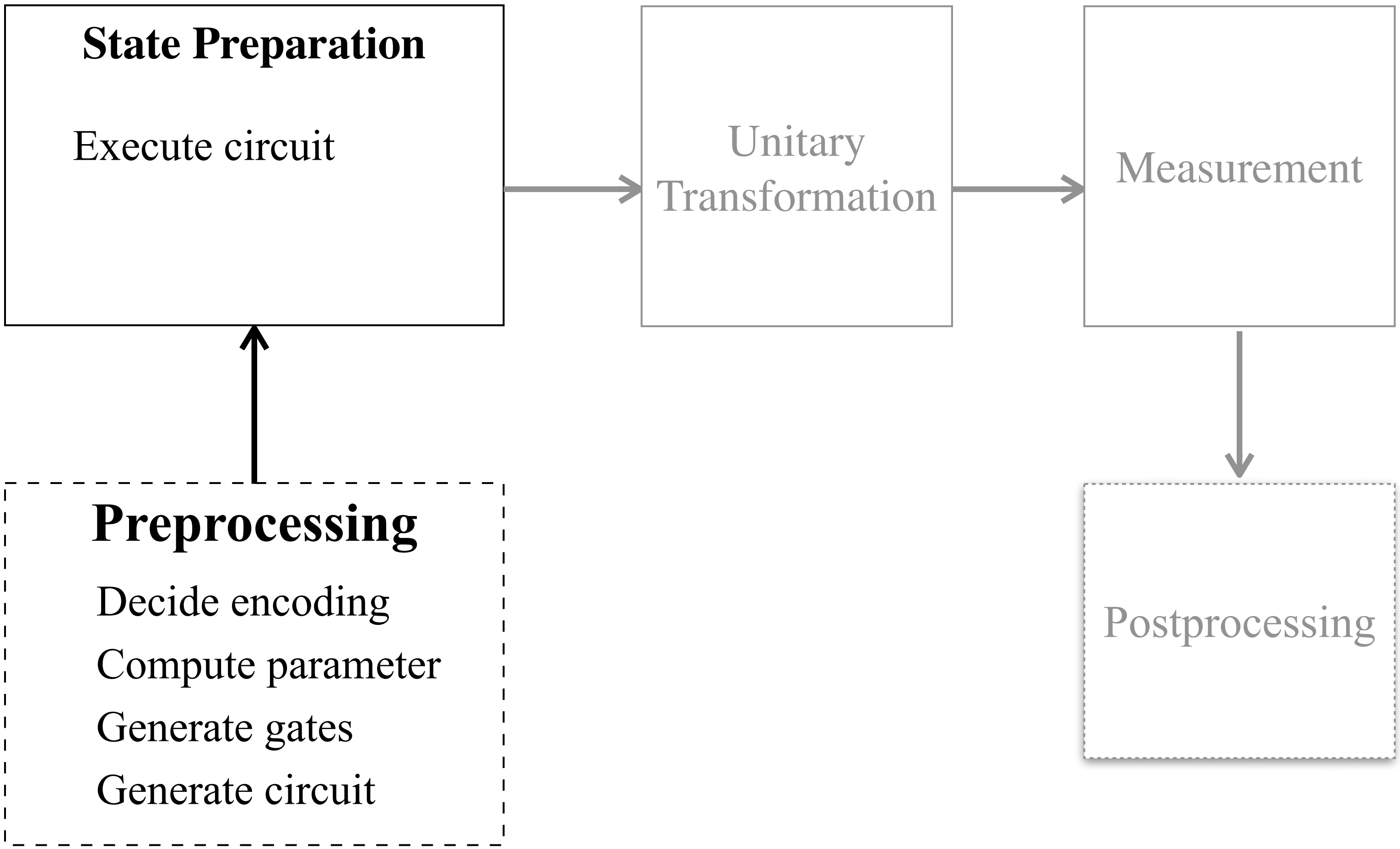
Compute A, U_1, V

Compute $A = \bigotimes_i M_i,\ U_1 \bigotimes_j M_j,\ V = \bigotimes_k M_k$

Compute $M_l = e^{i\alpha_l} R_z(\beta_l)\, R_y(\gamma_l)\, R_z(\delta_l)$

Generate corresponding circuits

General Proceeding: Input Preparation

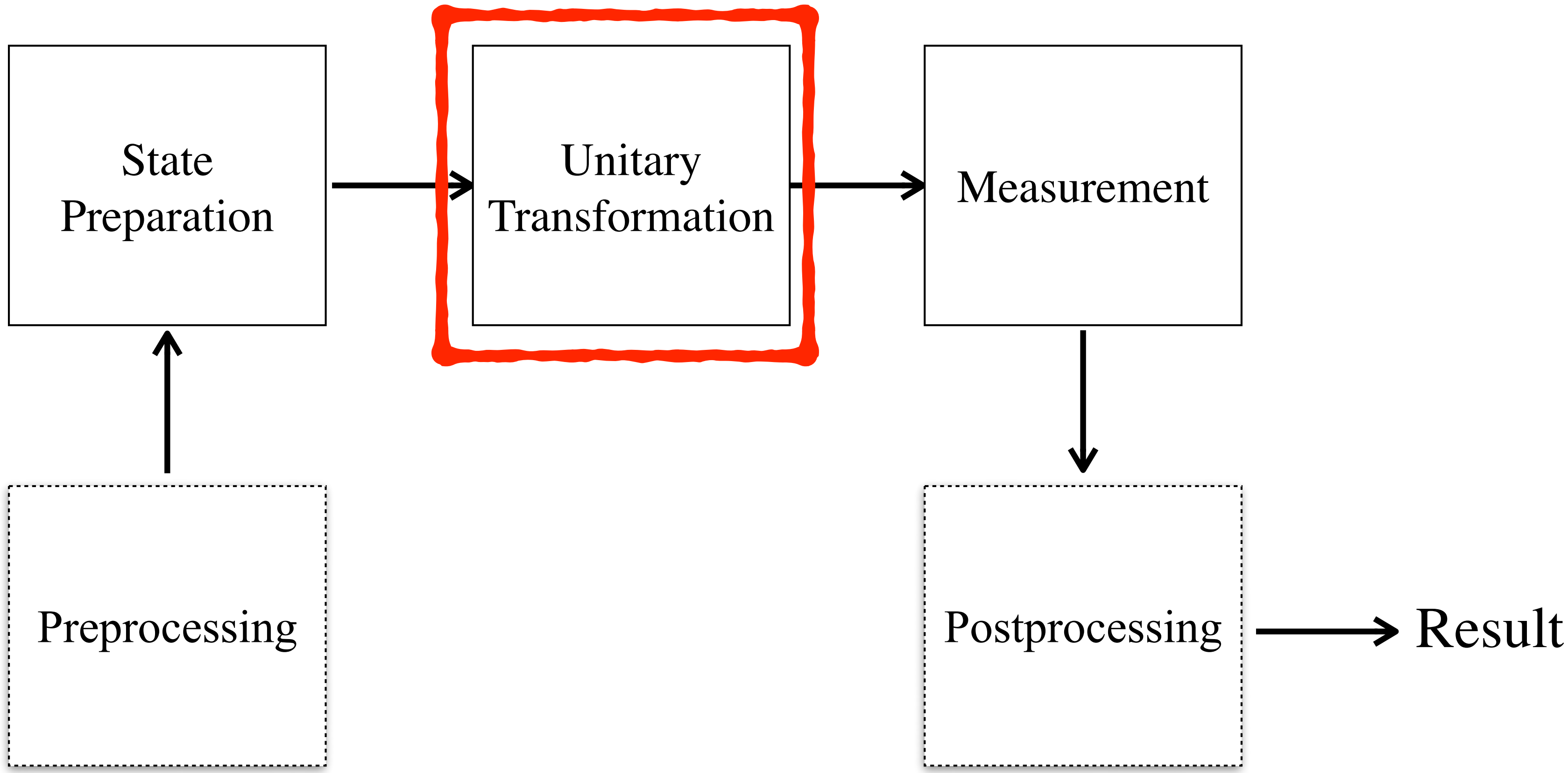


Implication of State Preparation

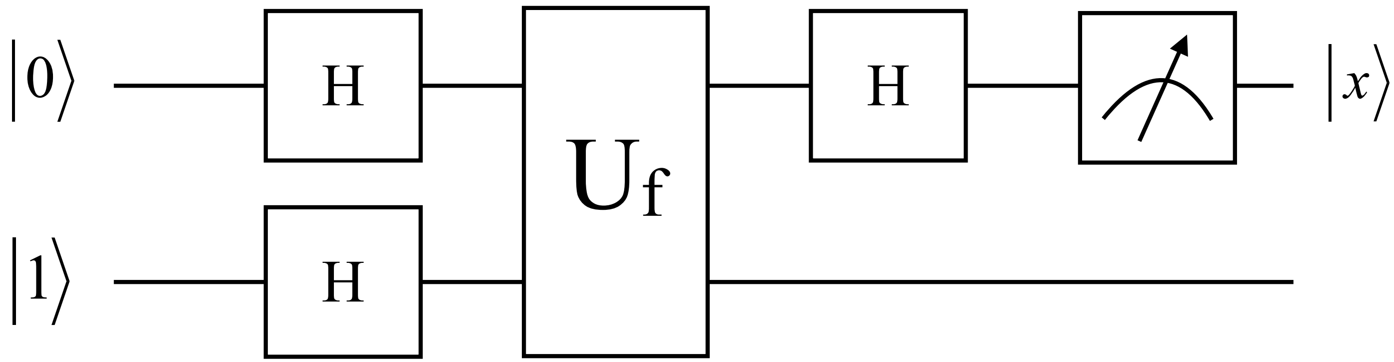
State preparation requires additional operations and additional qbits
as well as classical preprocessing
compared to the "ideal" algorithm.

Oracle Expansion

Reminder: Quantum Algorithm



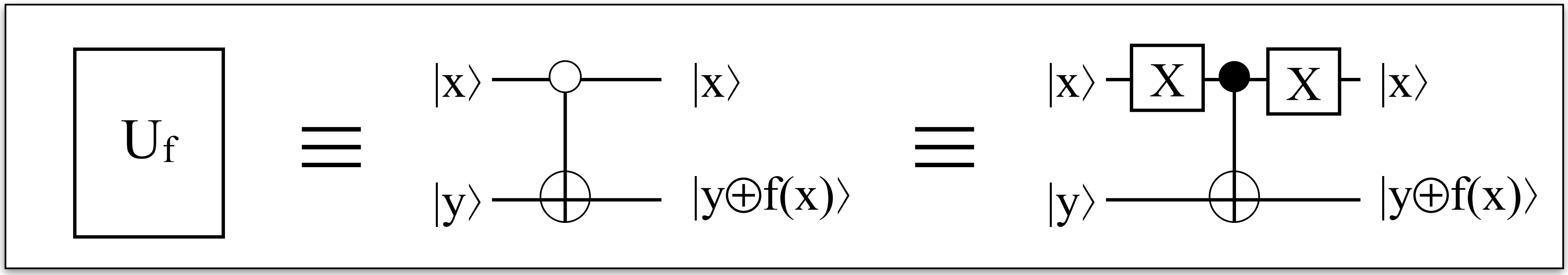
Algorithm of Deutsch



Sample Oracle

Let f : {0,1} → {0,1} be the function $0 \mapsto 1,\ 1 \mapsto 0$

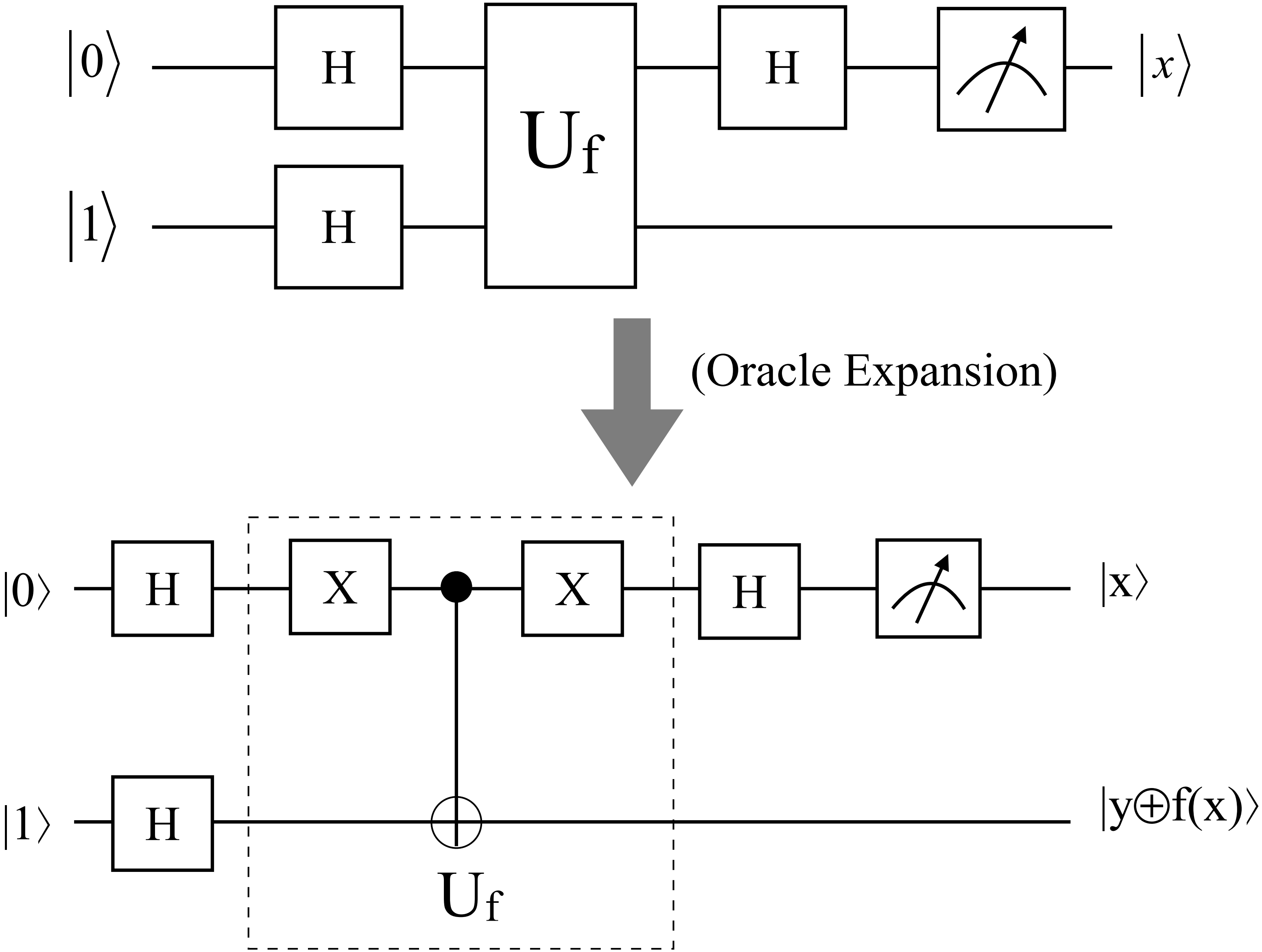
For $U_f|x,y\rangle = |x, y\oplus f(x)\rangle$ an oracle is:



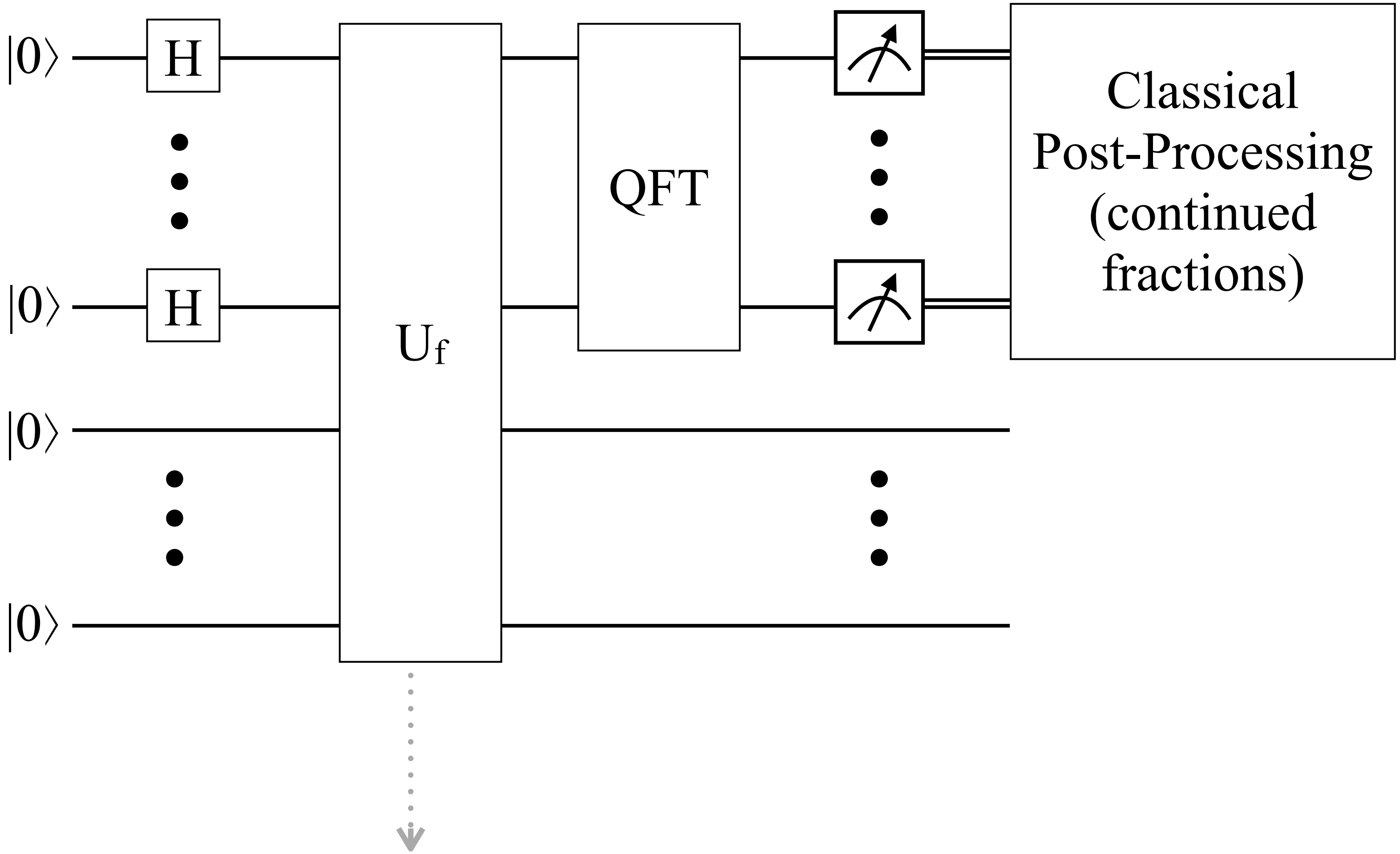
$$\begin{matrix}0\\0\end{matrix} \rightarrow \begin{matrix}0\\1\end{matrix} \qquad \begin{matrix}0\\1\end{matrix} \rightarrow \begin{matrix}0\\0\end{matrix} \qquad \begin{matrix}1\\0\end{matrix} \rightarrow \begin{matrix}1\\0\end{matrix} \qquad \begin{matrix}1\\1\end{matrix} \rightarrow \begin{matrix}1\\1\end{matrix}$$

(Note: different f require different U_f!)

Resulting Circuit

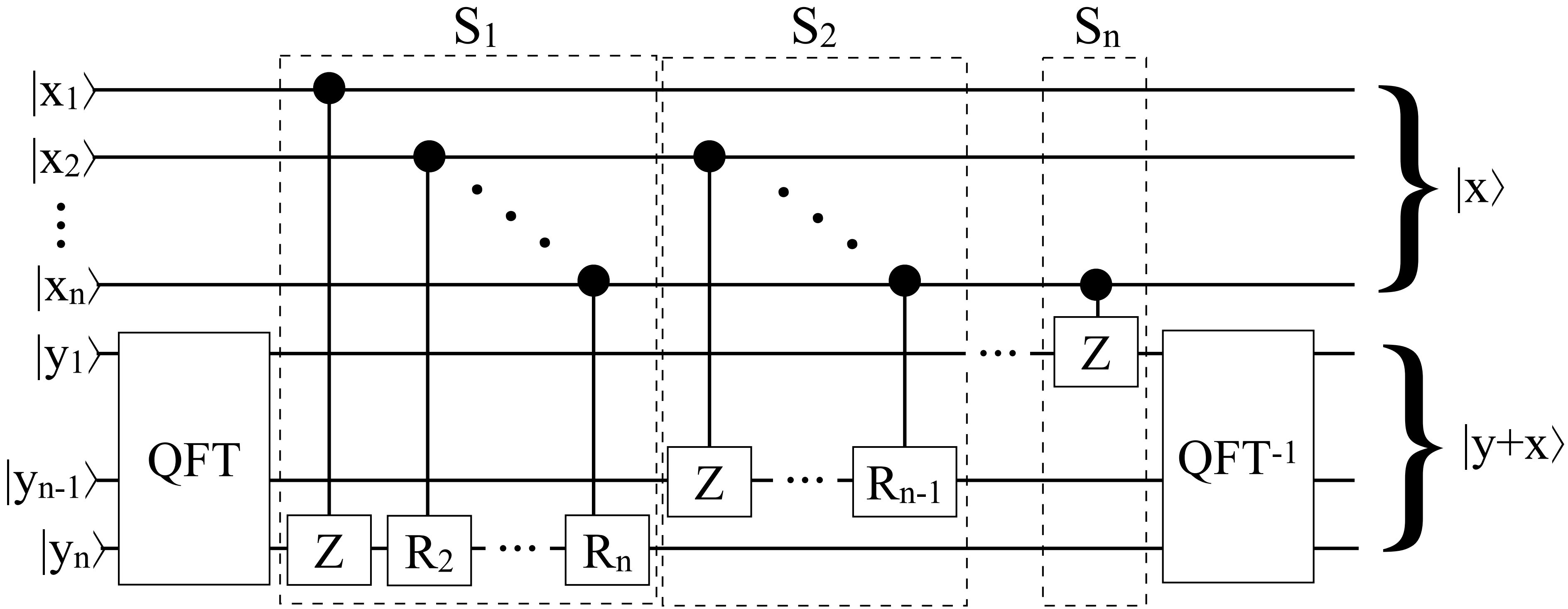


Algorithm of Shor



…computing f(x) = a^x mod n (→ multiplication, addition,…)

Addition



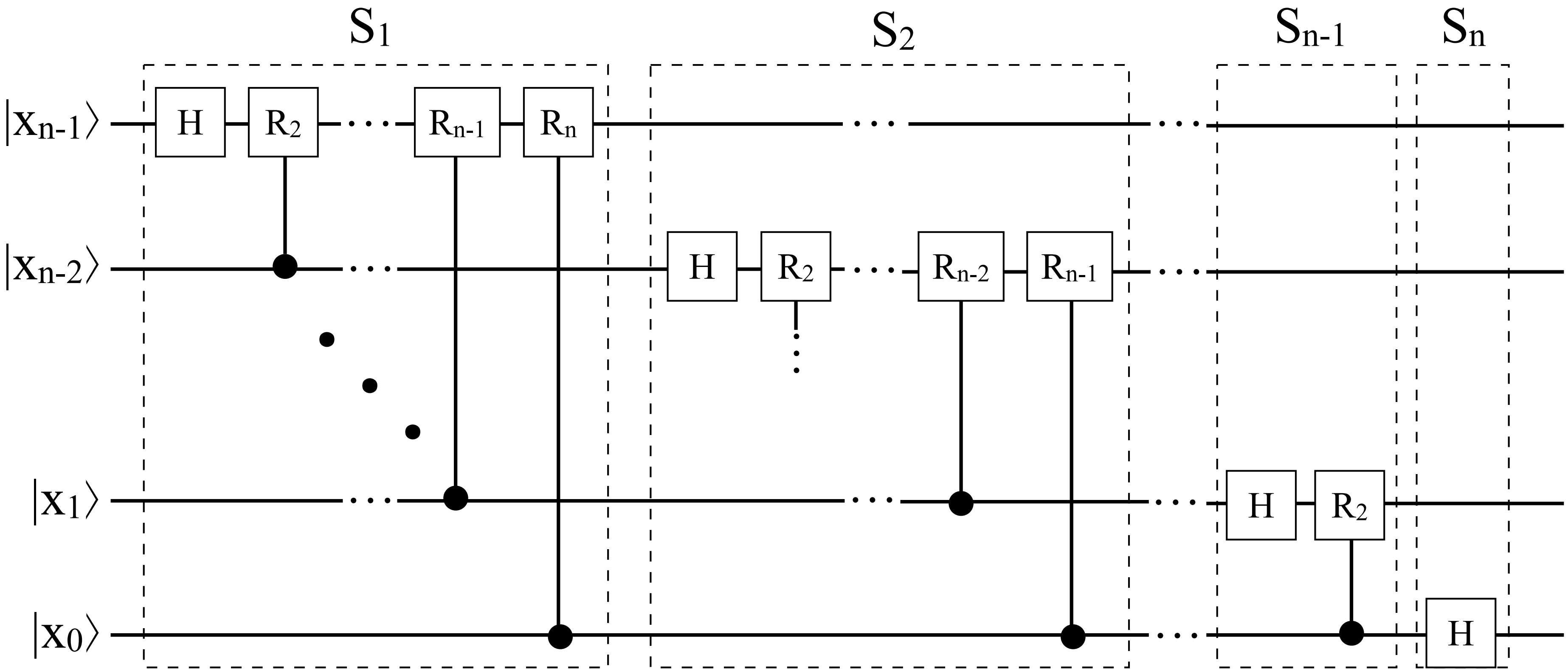
$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}$$ (and $R_0 = Z$)

Time complexity: $O(n^2)$

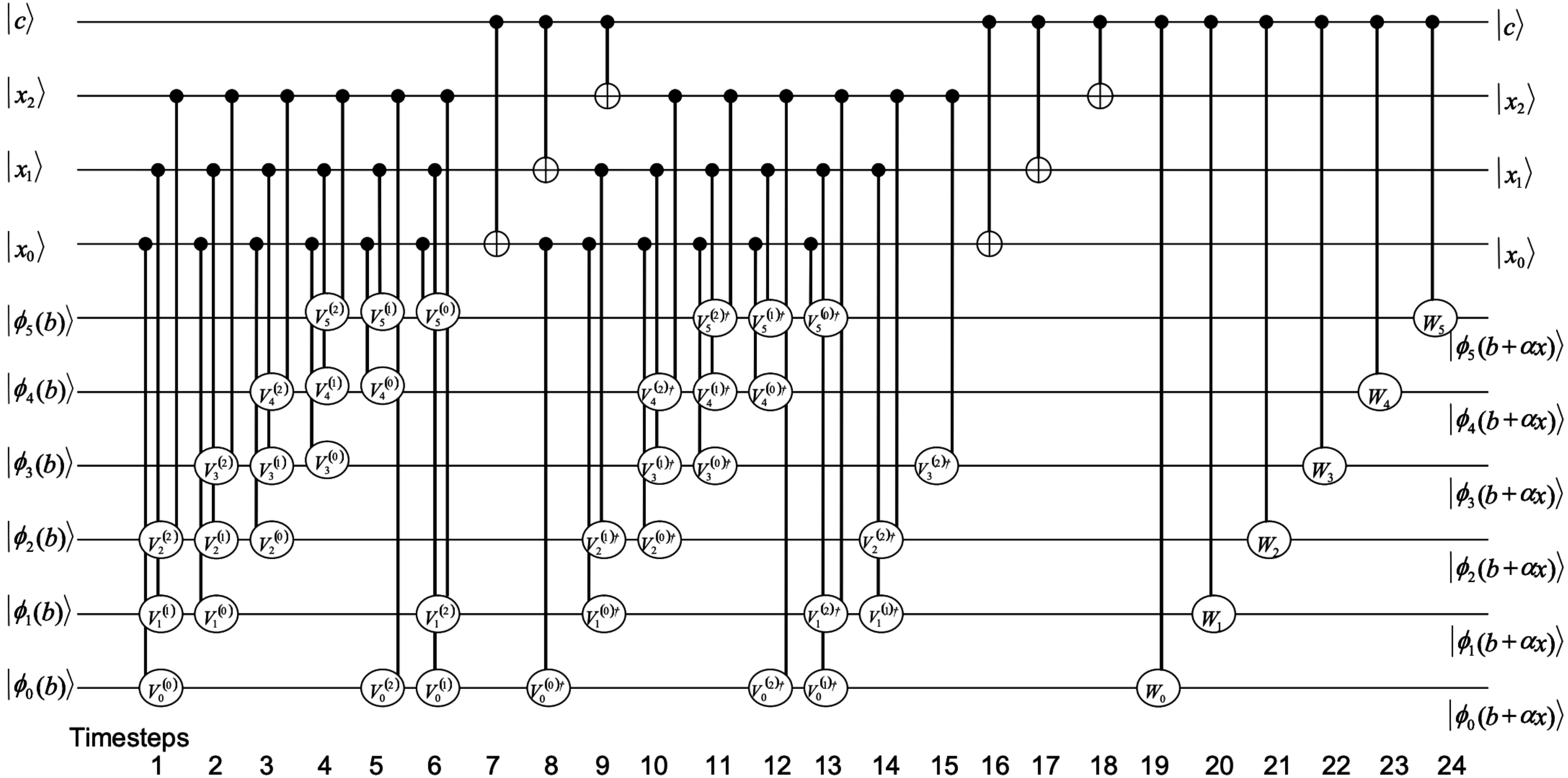
(Depth of the circuit can be significantly reduced, e.g. Z^C of S_2 can run in parallel to Z^C of S_1 etc…)

https://arxiv.org/abs/quant-ph/0008033v1

QFT



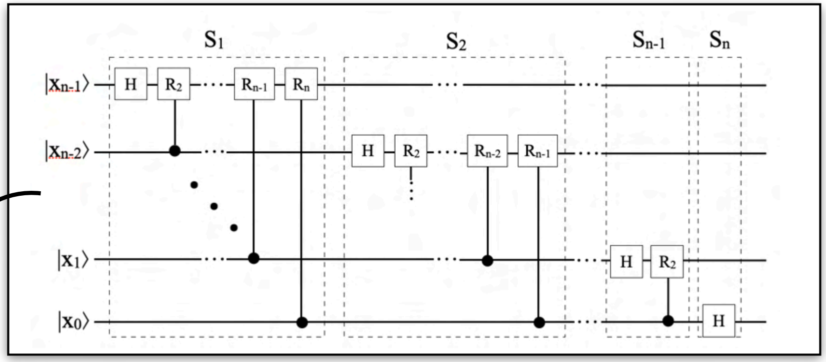
Multiplication: Sample

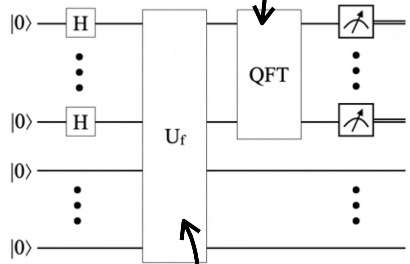


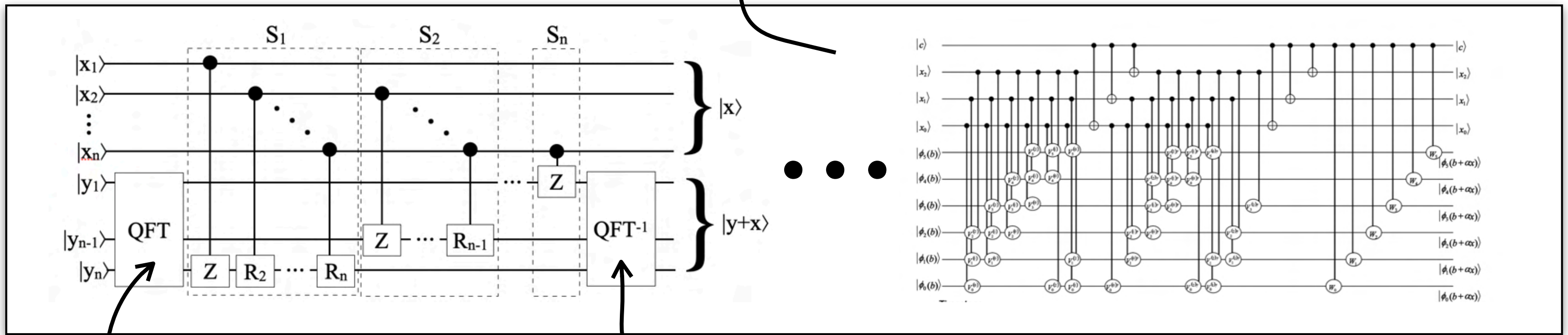
Computation of *b+ax*, *a:* 3-bit constant, *x:* 3 qbit, *b:* 6 qbit

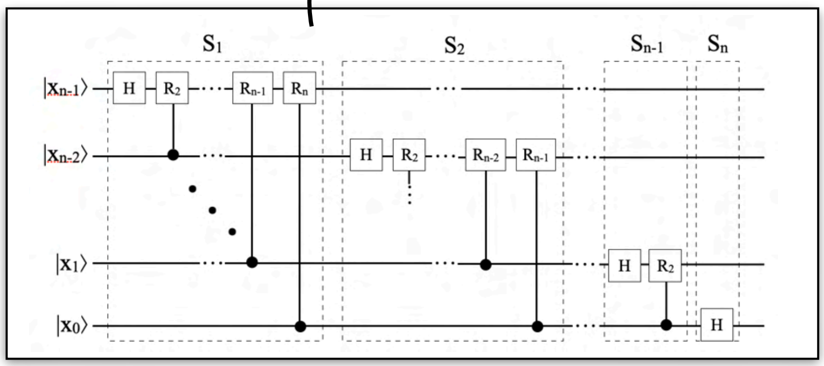
http://arxiv.org/abs/1207.0511v5

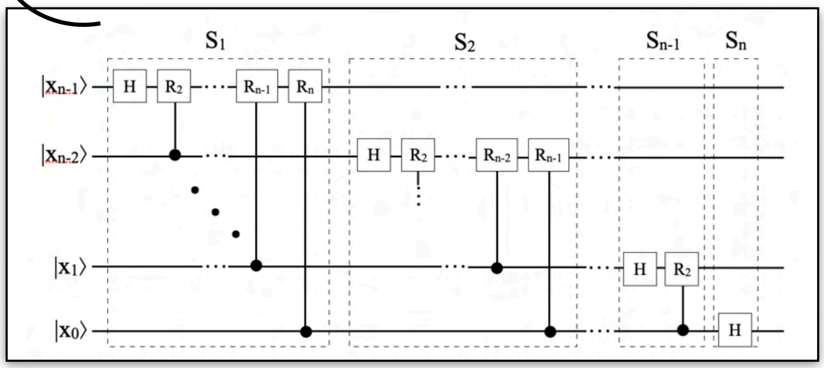
Shor Circuit: Summary









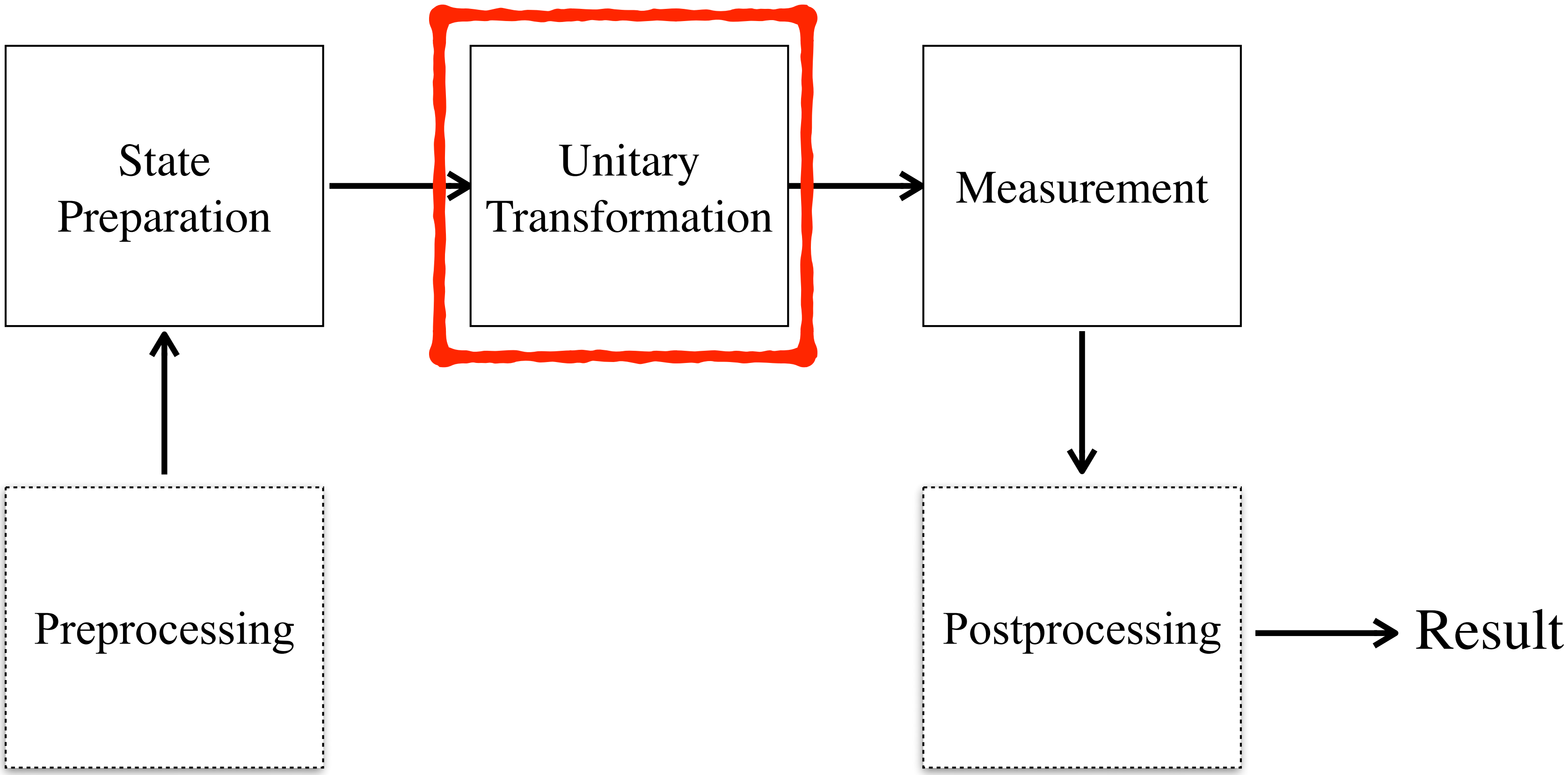


Implication of Oracle Expansion

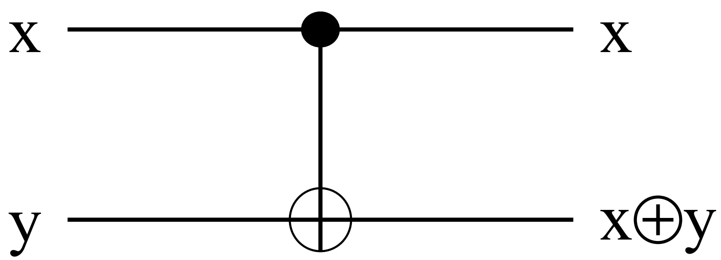
Oracle expansion requires additional operations (and additional qbits) compared to the "ideal" algorithm.

Connectivity

Reminder: Quantum Algorithm



CNOT (Controlled Not)



CNOT Gate

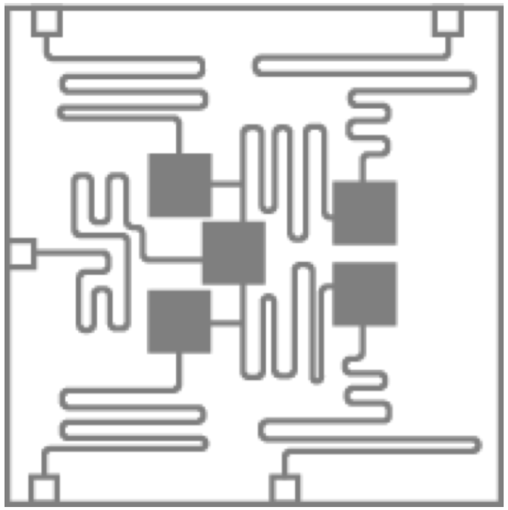
$\oplus$	0	1
0	0	1
1	1	0

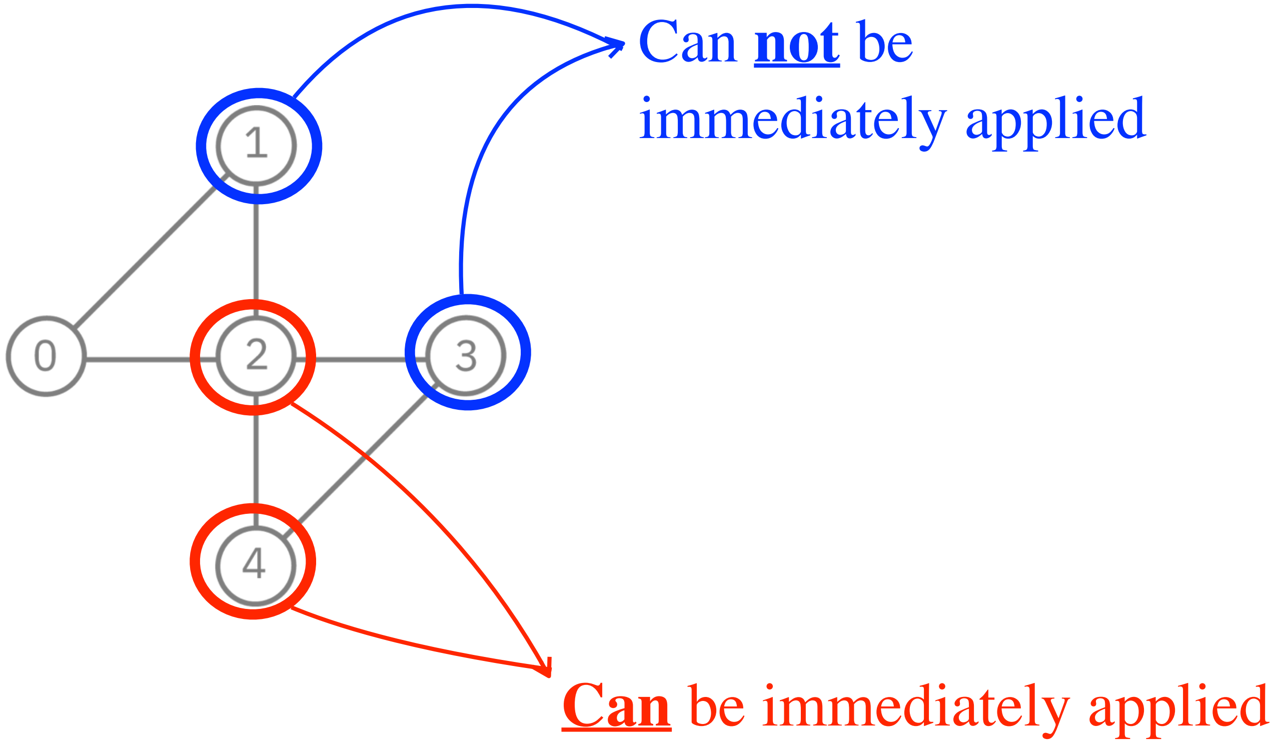
If x=1 then y will be negated; otherwise, y is not changed at all
(x is called *control*-qbit, y is called *target*-qbit)

The set of 1-qbit Operators and CNOT is universal.

Hardware Restrictions

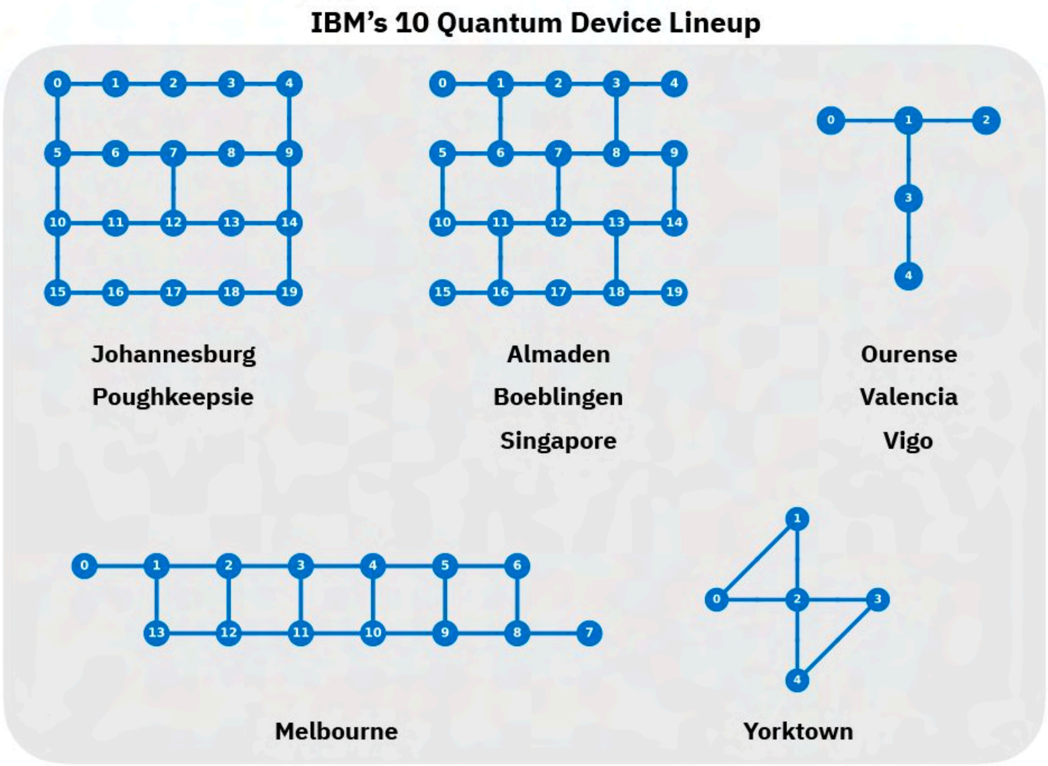
2-qbit operator on two qbits requires connection between them



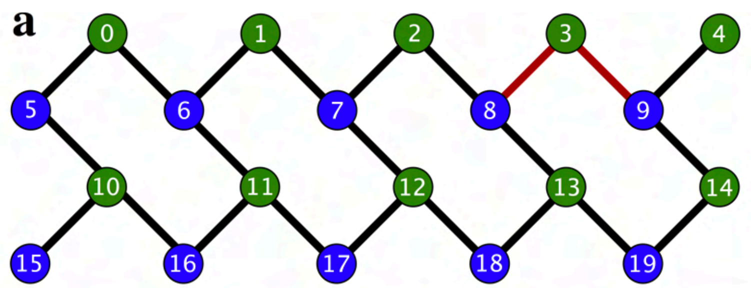


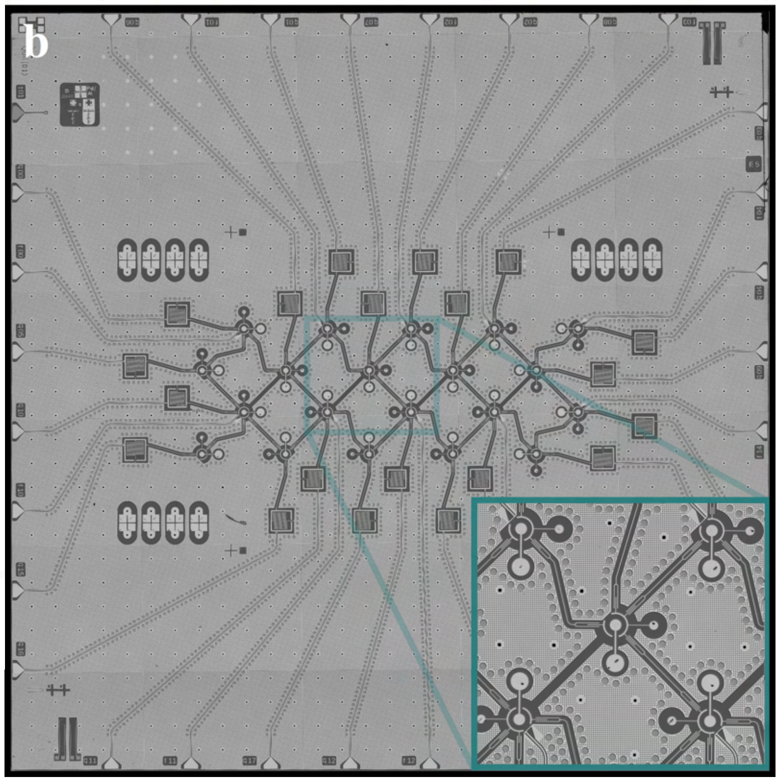
⇒ Connectivity of a quantum chip is important

Hardware: Connectivity



https://www.ibm.com/blogs/research/2019/09/quantum-computation-center/





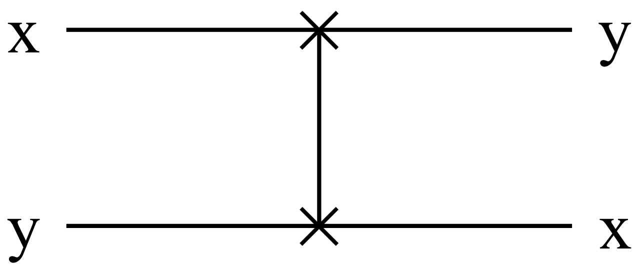
http://docs.rigetti.com/en/1.9/_images/acorn.png

Swap Operator

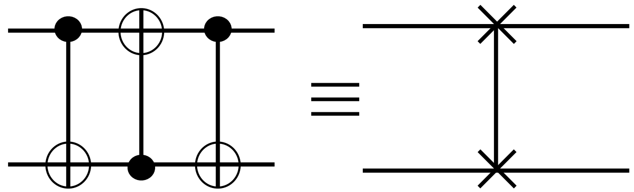
$$\mathrm{SWAP} : \mathbb{H} \otimes \mathbb{H} \to \mathbb{H} \otimes \mathbb{H}$$

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |10\rangle \\ |10\rangle &\mapsto |01\rangle \\ |11\rangle &\mapsto |11\rangle \end{aligned}$$

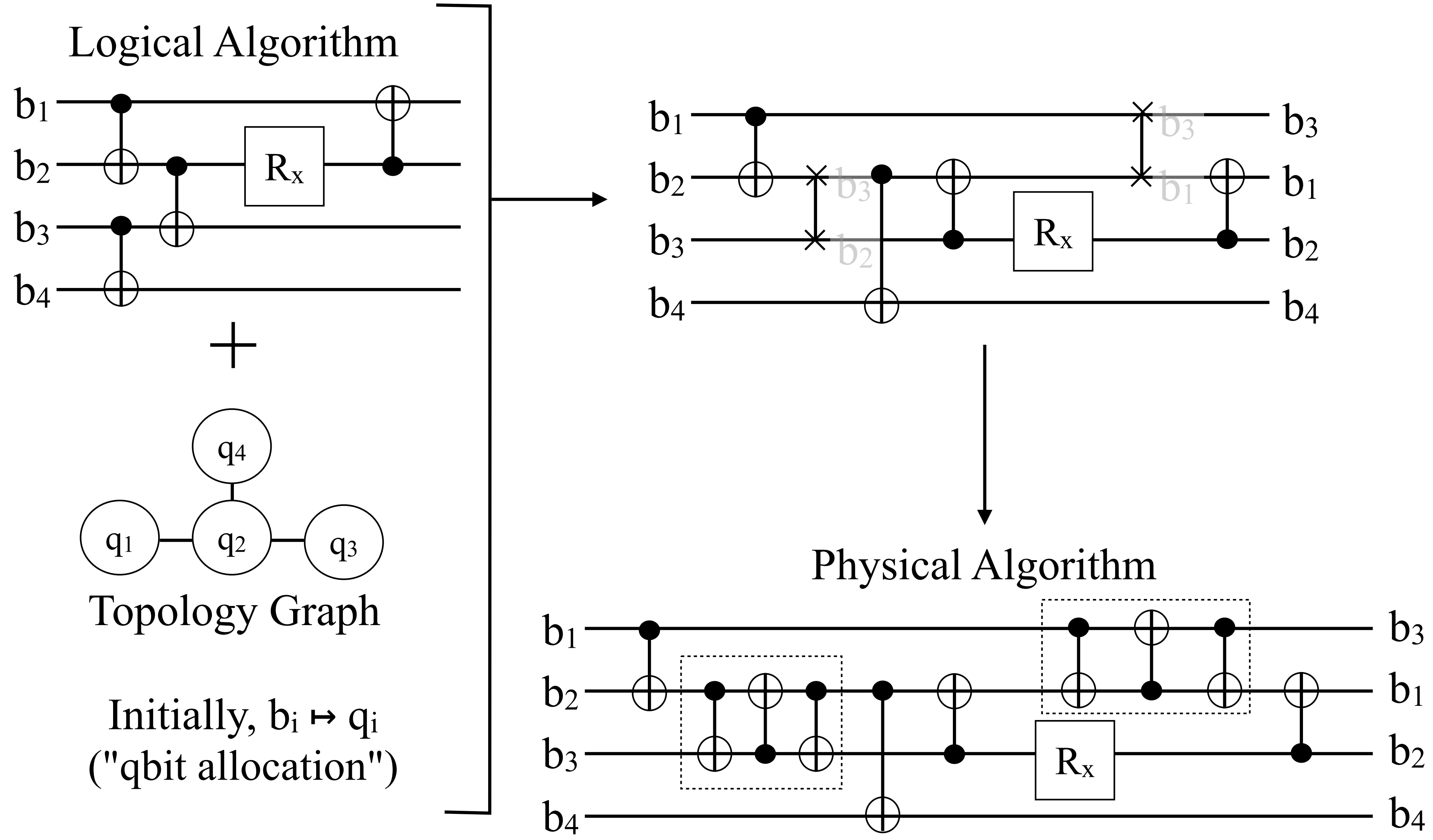
I.e. both input qbits are exchanged



SWAP Gate



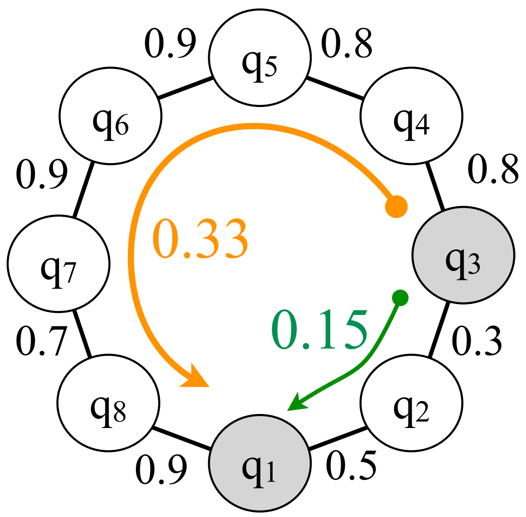
Example: Considering Topology



Example: Variation-Aware Qbit Movement

Typically, 2-qbit operations along different connections have different success rate

Annotation s_{ij} on the edge $\{q_i,q_j\}$ denotes the success rate of a 2-qbit operation involving qbit q_i and q_j



Topology Graph

Scenario: a 2-qbit operation Ω is to be performed on q_1, q_3

Swapping $q_3 \rightarrow q_2$, followed by $\Omega(q_1,q_2)$ has success rate $0.3 \times 0.5 = 0.15$

Swapping $q_3 \rightarrow q_4 \rightarrow q_5 \rightarrow q_6 \rightarrow q_7 \rightarrow q_8$ followed by $\Omega(q_1,q_8)$ has success rate $0.8 \times 0.8 \times 0.9 \times 0.9 \times 0.7 \times 0.9 = 0.33$

⇒ Using a single SWAP followed by Ω has a lower success rate than using 5 SWAPs followed by Ω

⇒ Success rate of qbit connections influences the number of SWAPs performed as well as error rates of 2-qbit operations

Even worse, the success rate changes over time!

Example: Variation-Aware Qbit Allocation

The qbits of the quantum circuit must be assigned to physical qbits of the QPU

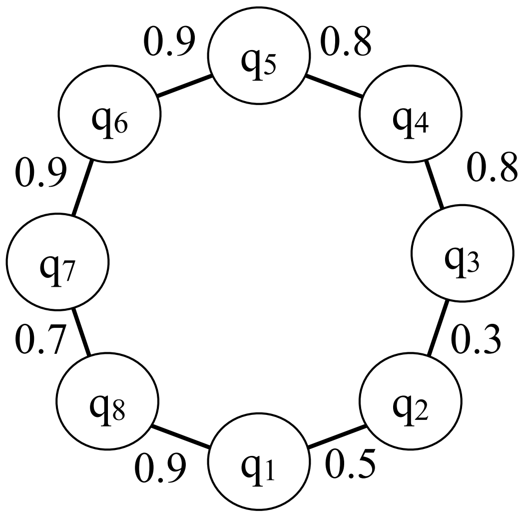
- This is an initial allocation that changes during the execution
- The goal is to improve reliability of the computation

Naive allocation selects any subgraph to minimizes SWAPs

Considering success rate of connections determines connected subgraph with maximum weights

- In the example: Q0, Q1, Q2 ↦ q_5, q_6, q_7

```
qreg Q[3];
creg C[3];

x Q[0];
cx Q[0],Q[1];
cx Q[2],Q[1];
measure Q[1] -> C[1];
```



Mapping	Weight
q_1, q_2, q_3	0.15
q_2, q_3, q_4	0.24
q_3, q_4, q_5	0.64
...	
q_5, q_6, q_7	0.81
...	
q_7, q_8, q_1	0.63

Implication of Connectivity

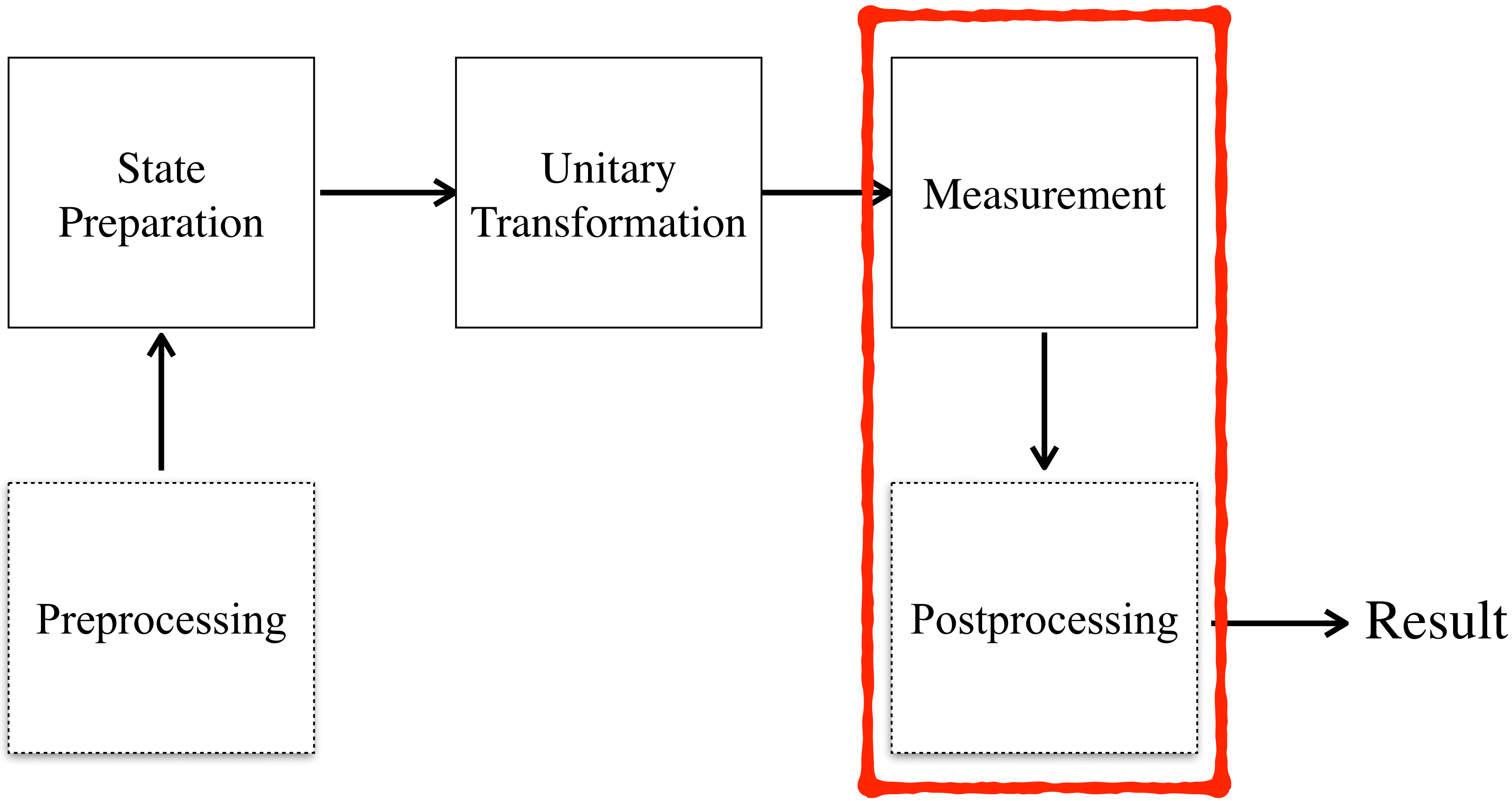
The connectivity of a QPU implies
the injection of additional (SWAP) operations
into the "ideal" algorithm.

The success rate of qbit connections influence the number of SWAPs
as well as the error rate of 2-qbit operations

Considering the success rate of qbit connections
as well as error rate of 1-qbit operations
during qbit allocation of a quantum circuit
influences the reliability of its execution

Readout Errors

Reminder: Quantum Algorithm



Readout Errors

Duration of a measurement is significantly larger than decoherence time

$\Rightarrow$ a qbit under measurement may relax during this time
(e.g. flip from $|1\rangle$ to $|0\rangle$ in between)

Thus, *readout errors* correspond to
disturbed probability distributions of measured results

Principle: Correcting Readout Errors

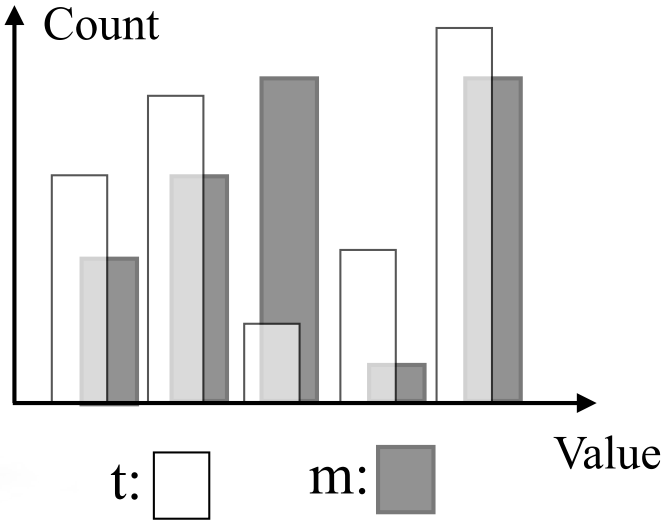
Unfolding: Reconstruction of a true "undisturbed" distribution out of a measured "disturbed" distribution

(several unfolding methods exist, the following is a straightforward one)

Let t be the true distribution, m be the measured distribution — $t, m \in \mathbb{N}^k$ — where k is the number of values, and $t_i, m_i \in \mathbb{N}$ is the count of the i-th value

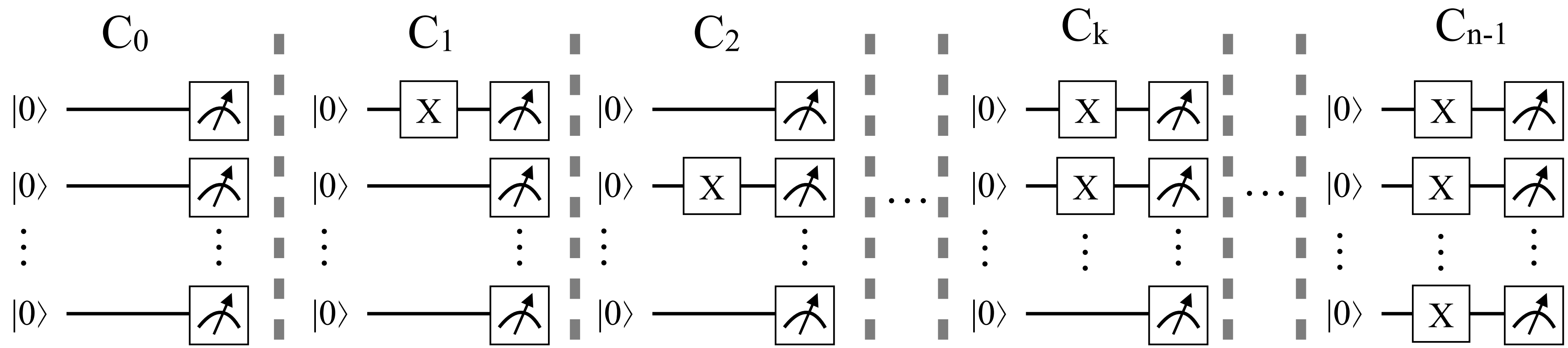
t, m are related by a *calibration matrix*(*) C: $t = C \cdot m$
with C_{ij} = Prob(measured value = j | true = i)

Correcting readout errors means determining the calibration matrix C (*unfolding method*)



(*) a.k.a. *response matrix*

Constructing the Calibration Matrix



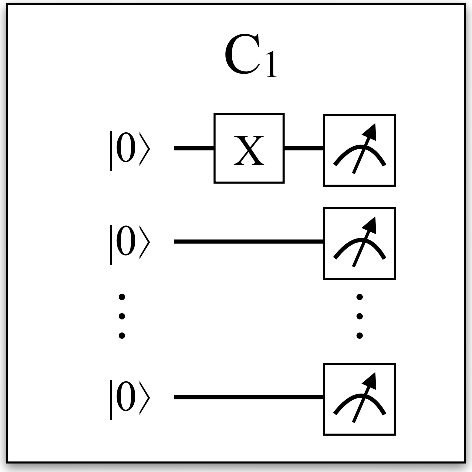
Construct and measure each element of the computational basis $|i\rangle \in \{0,1\}^n$

- I.e. use the above so-called *calibration circuits* C_i, $0 \leq i \leq n-1$

Applying circuit C_i should result in [i], but result [j] is readout error

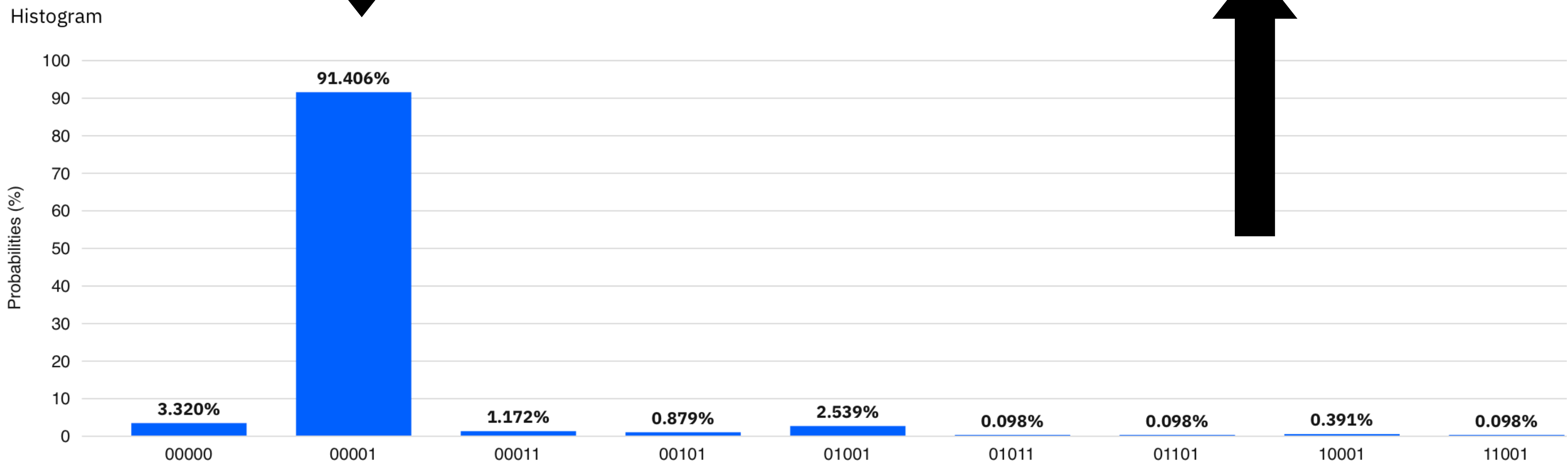
- C_i is performed M times
- If [j] results K times, then $C_{ij} = K \backslash M$
  $\Rightarrow C_{ij} = \text{Prob(measured value} = j \mid \text{true} = i)$

[i] : integer value of $|i\rangle$



$$
\begin{aligned}
C_{1,0} &= 0.03320\\
C_{1,1} &= 0.91406\\
C_{1,3} &= 0.01172\\
C_{1,5} &= 0.00879\\
C_{1,9} &= 0.02539\\
C_{1,11} &= 0.00098\\
C_{1,13} &= 0.00098\\
C_{1,17} &= 0.00391\\
C_{1,25} &= 0.00098
\end{aligned}
$$

$\ldots$all other $C_{1,i} = 0$

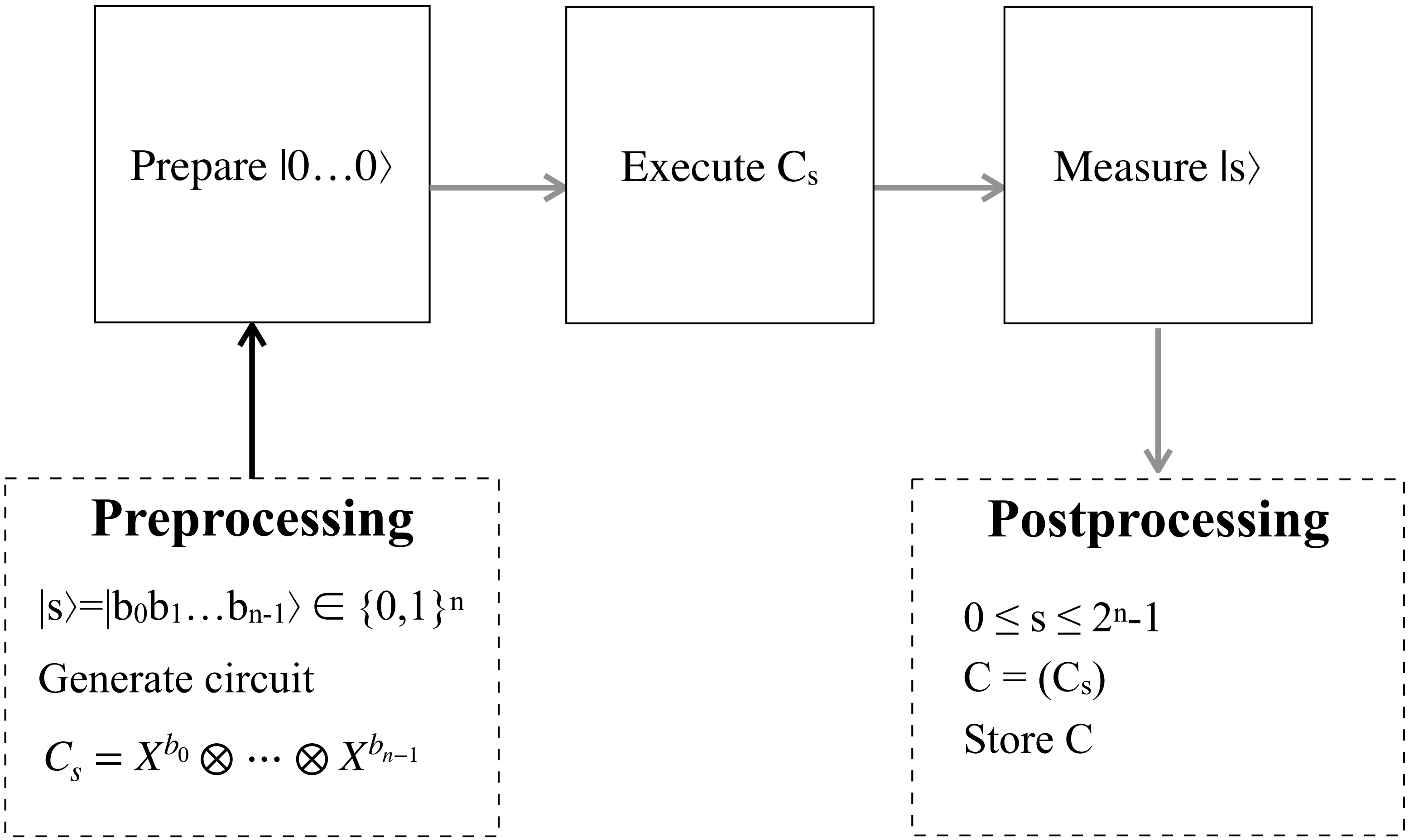


Implication of Readout Errors

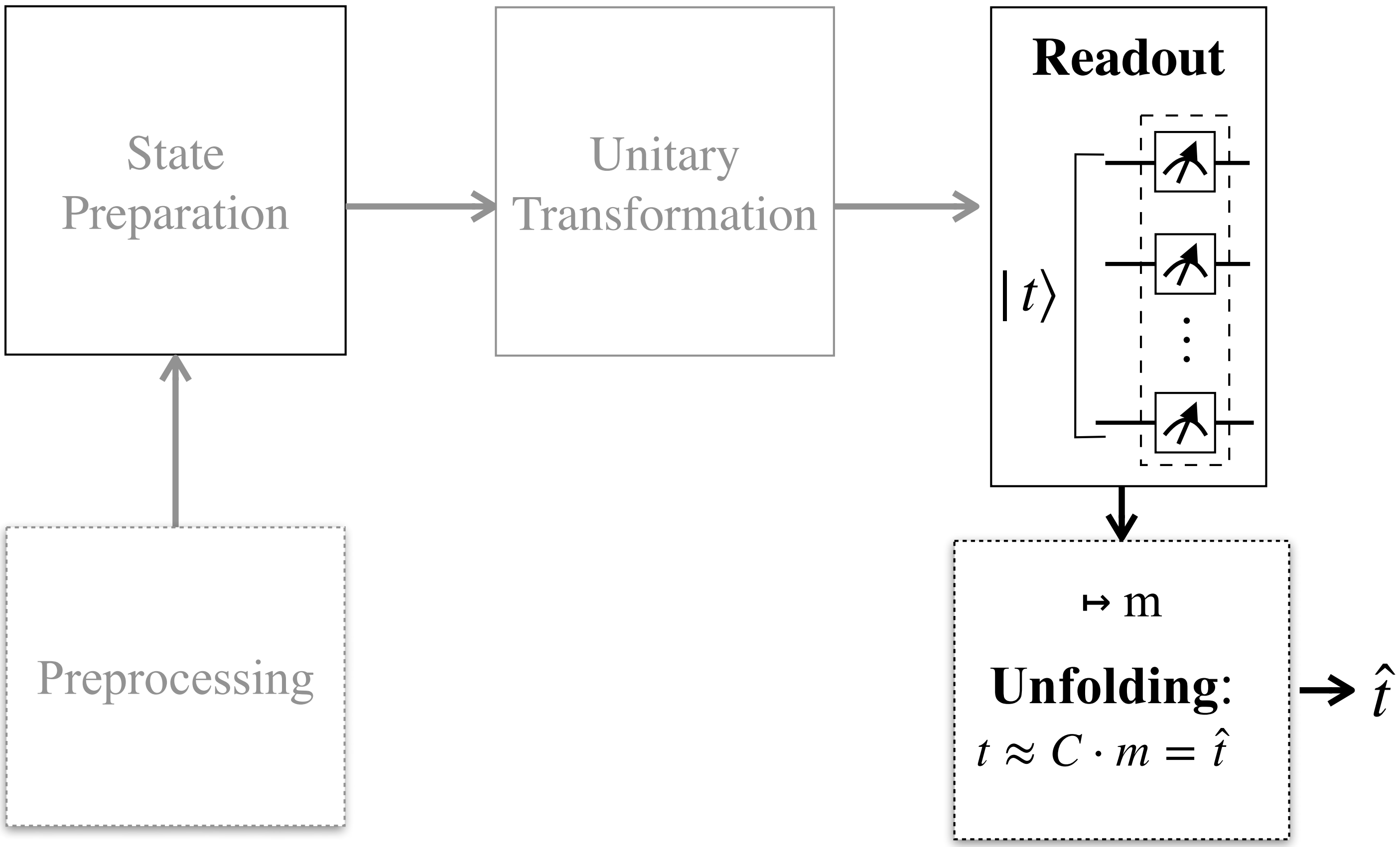
Correcting readout errors requires additional operations
(namely the calibration circuits)
to determine the calibration matrix regularly
(fortunately not for every execution of the "ideal" algorithm)

Correcting readout errors requires classical post-processing,
i.e. applying the calibration matrix to the measured results

Readout Errors: Periodic processing

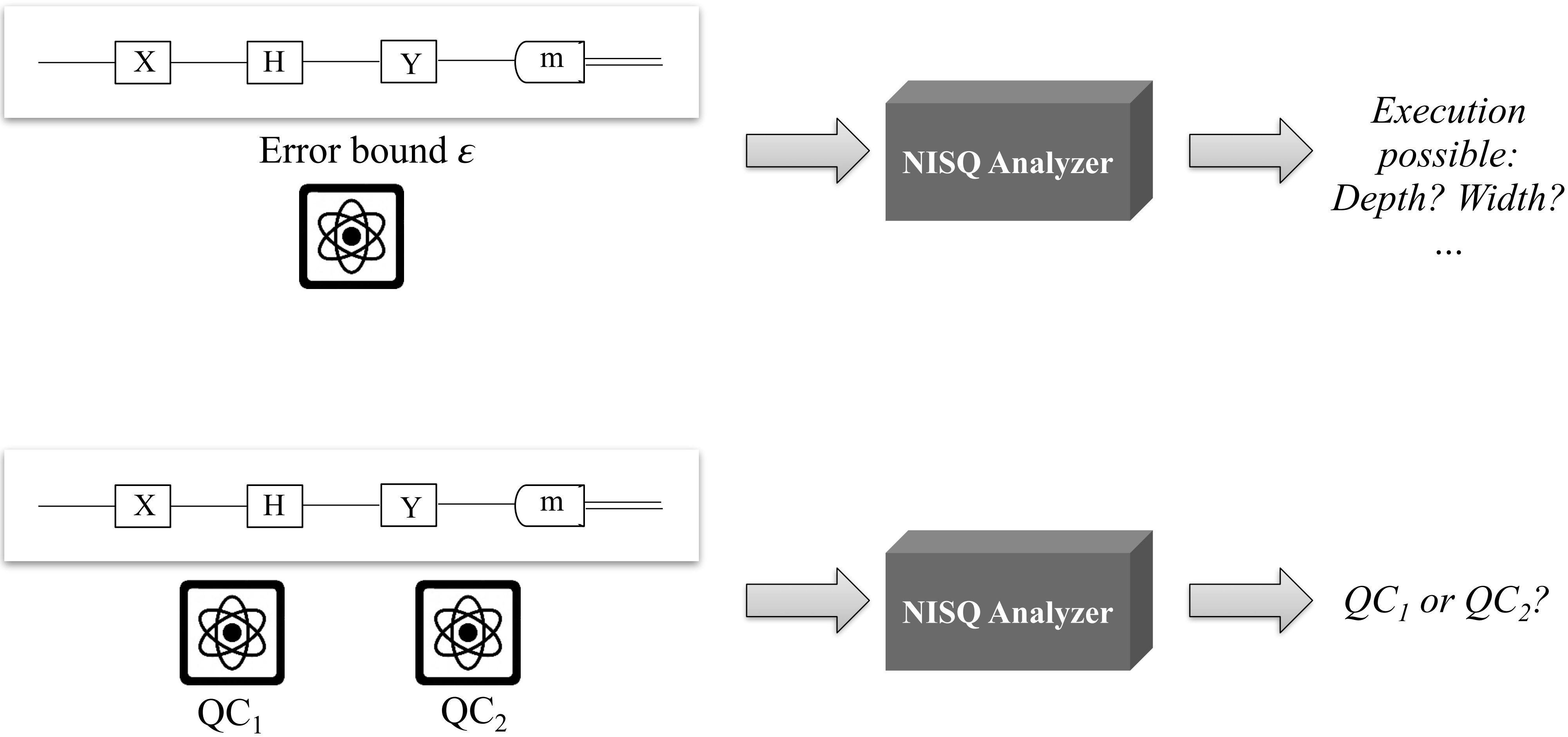


Readout Errors: Postprocessing



NISQ Analyzer

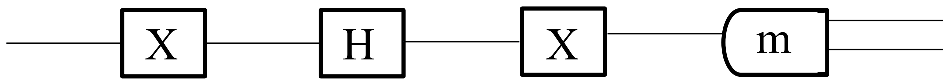
NISQ Assessment



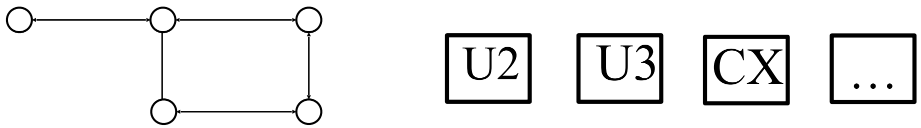
Provenance: Definition

- Definition
  - Information describing a process, computation, or data
  - Goals: reproducibility, understandability, quality
- Importance for QC
  - Noisy machines (decoherence, gate infidelity,…)
  - Very different hardware implementations (superconducting, trapped ion, optical, …)

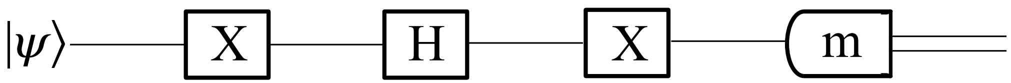
Provenance: Categories



Used gates, depth,…

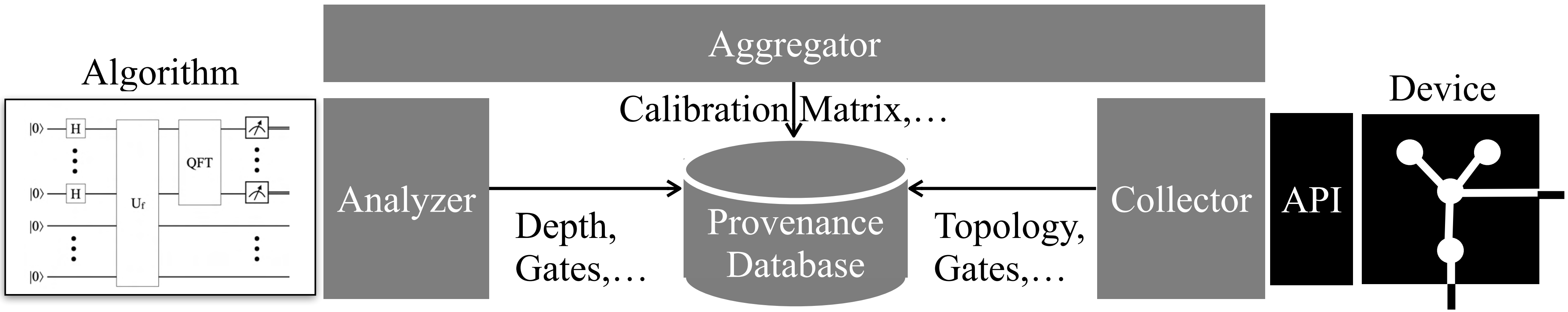


Available gates, topology,…



SPAM Errors

Provenance Usage



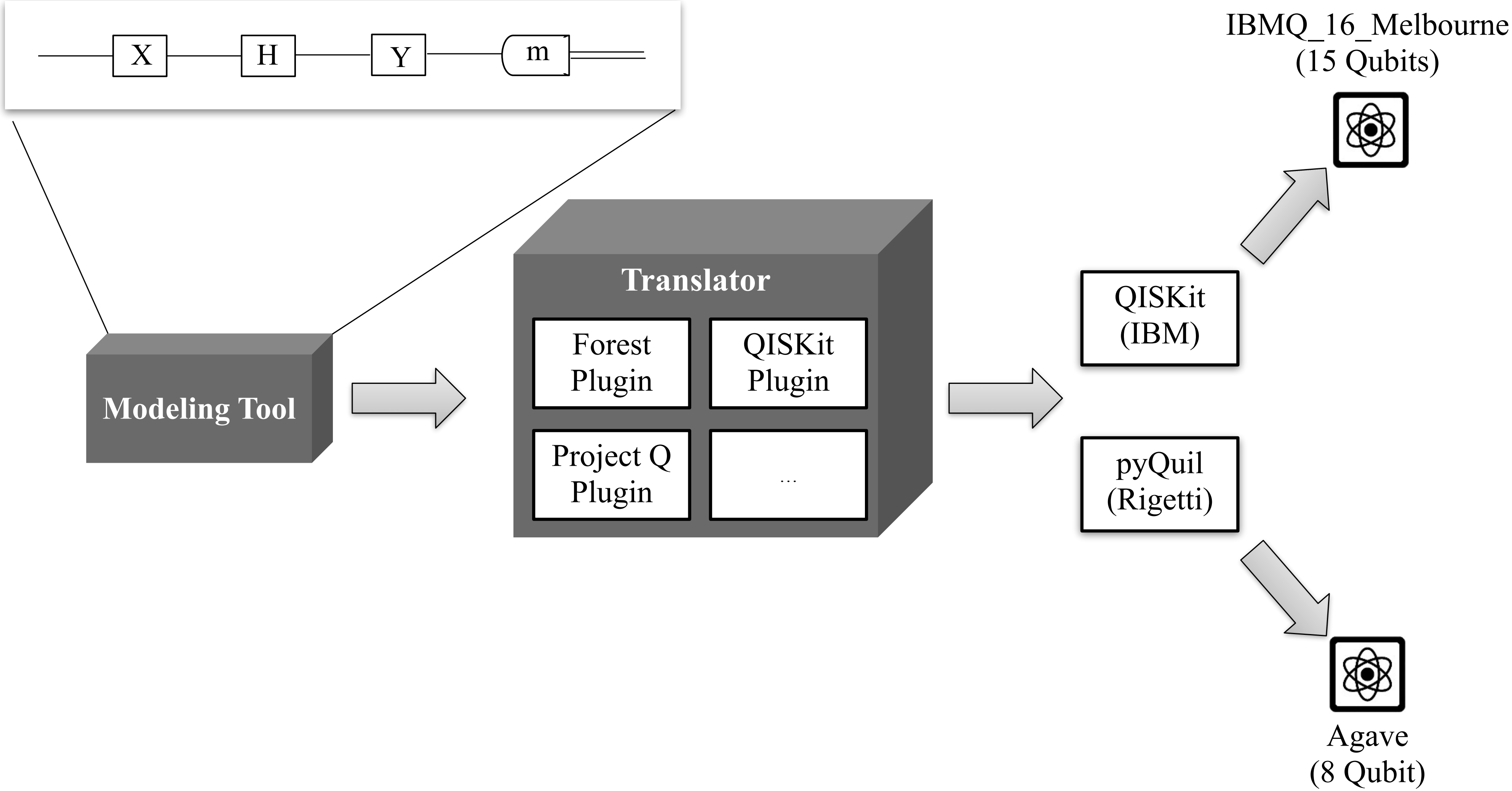
Hardware Dependent Operations

Set of basic operators is hardware *implementation* dependent

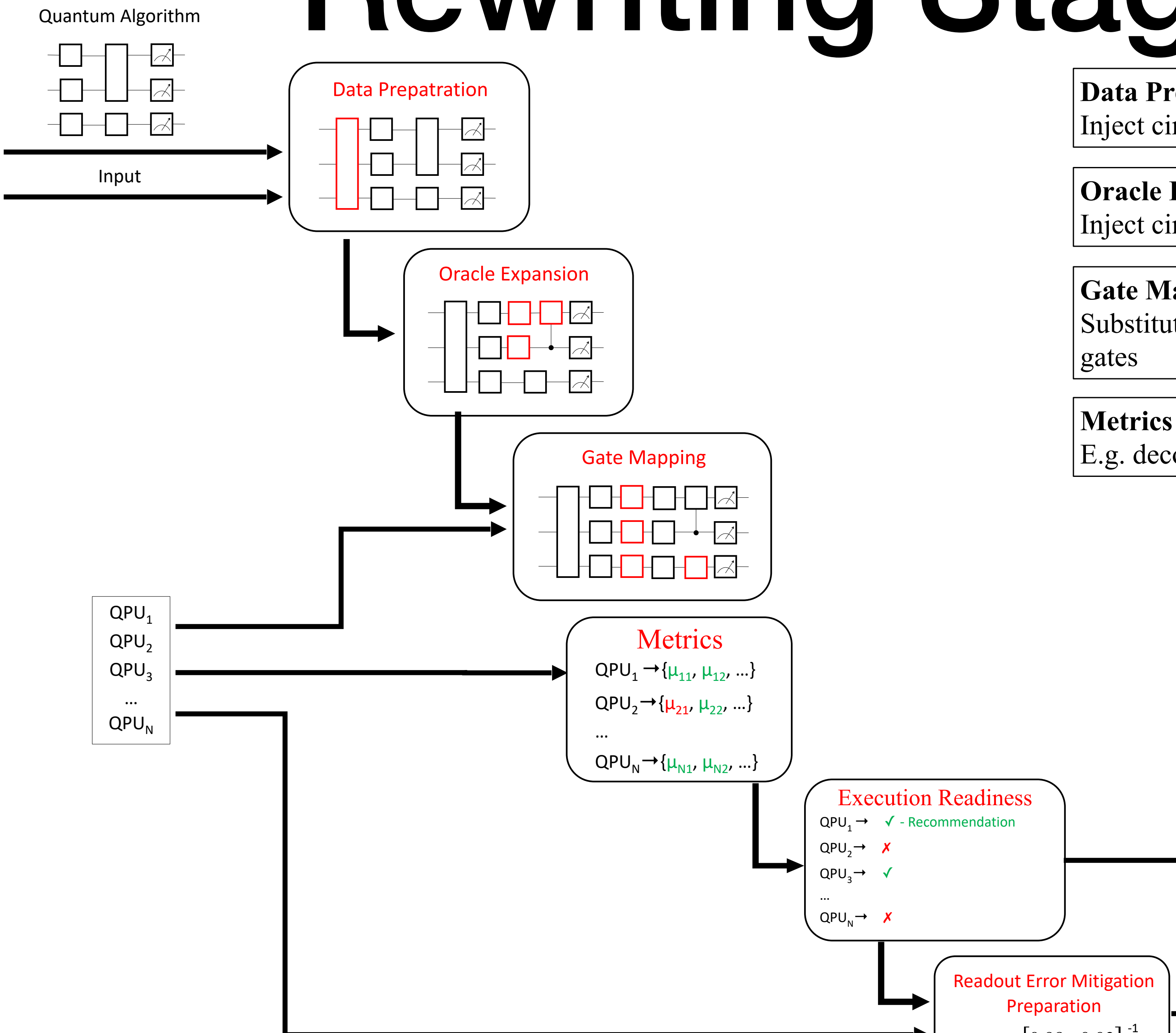
- E.g. continuous-variable (CV) operations in optical quantum computers
  - Squeezing, FockState,... in PennyLane
- E.g. different sets of basic operators implemented by vendors of same category of hardware implementation
  - E.g. U1, U2, U3,… on IBM Q; or Rx(π/2), CZ,… on Rigetti; …

Thus, NISQ compiler must even be aware of implementation of hardware

NISQ Rewriting



Rewriting Stages



Recommentation:
Deploy on QPU_1, QPU_3

Calibration Matrix

$$\begin{bmatrix} 1.02105263 & -0.03157895 \\ -0.02105263 & 1.03157895 \end{bmatrix}$$

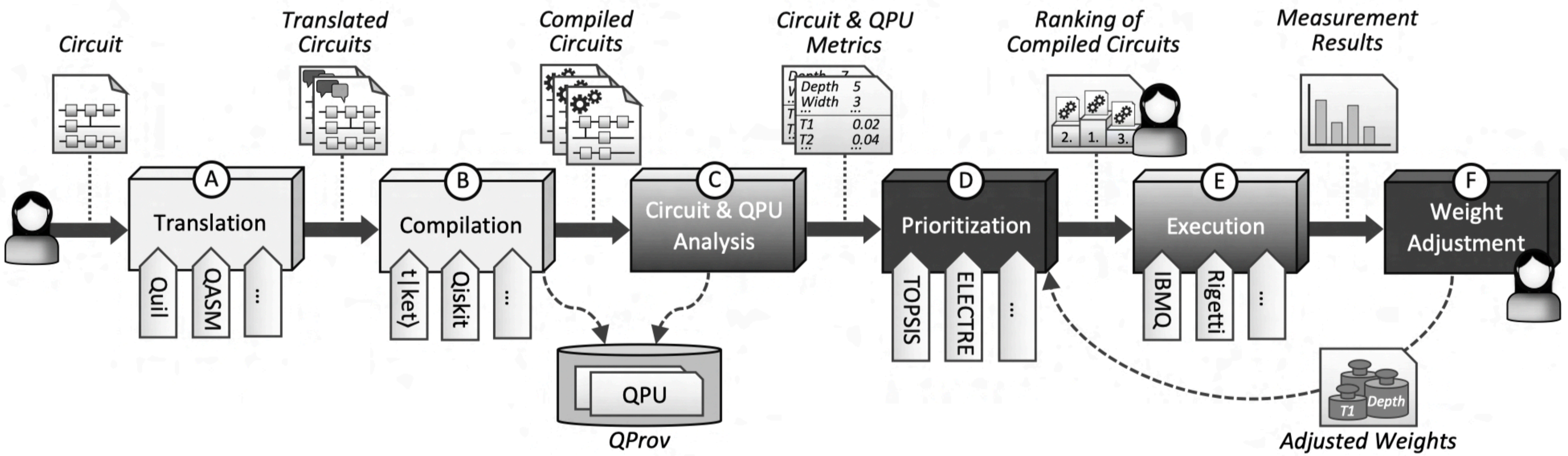
Data Preparation
Inject circuit that initializes input state

Oracle Expansion
Inject circuit of oracle logic

Gate Mapping
Substitute gates by circuit of vendor supported gates

Metrics
E.g. decoherence time (T1, T2), gate fidelity,…

NISQ Recommender



Final Remarks

Summary

- NISQ is determined by
  - Decoherence
  - Gate infidelity
  - Readout errors
  - Connectivity
- Data preparation is another problem
- Measurement yet another one
- All these problems can be addressed…
  - …but require additional gates and qbits
- Thus, resources available for proper algorithm is further reduced
- NISQ Analyzer (Rewriter, Recommender,…) will be a tool that helps to determine best QPU to be used for solving a problem based on a given algorithm and given data under constraints like cost, precision,…

The End