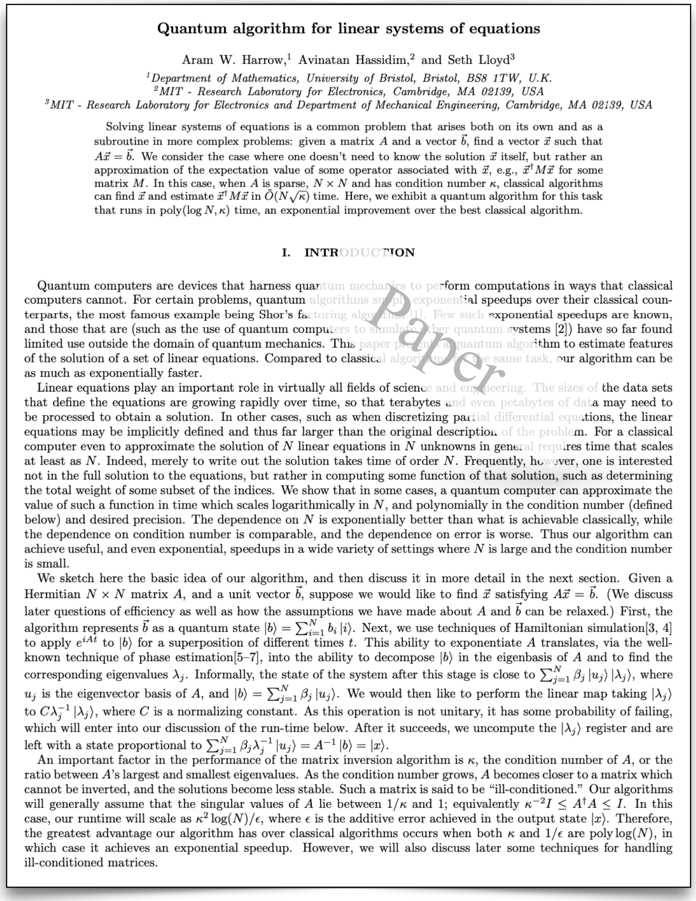
Quantum Applications are Hybrid

(SummerSoC 2022; Crete; July 3 - 9, 2022)

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

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, German

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

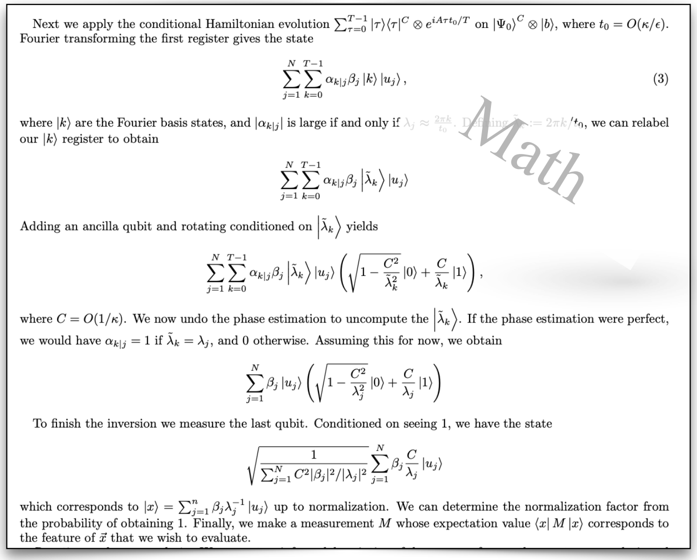
Quantum computers are devices that harness quantum mechanics to perform computations in ways that classical computers cannot. For certain problems, quantum algorithms supply exponential speedups over their classical counterparts, the most famous example being Shor's factoring algorithm [1]. Few such 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 $\kappa^{-2} I \leq A^\dagger A \leq I$. 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

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

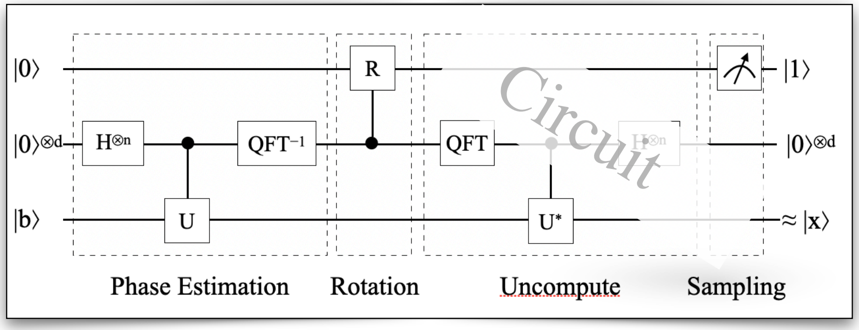
where $C = O(1/\kappa)$. We now undo the phase estimation to uncompute the $\left|\tilde{\lambda}_k\right\rangle$. If the phase estimation were perfect, we would have $\alpha_{k|j} = 1$ if $\tilde{\lambda}_k = \lambda_j$, and 0 otherwise. Assuming this for now, we obtain

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

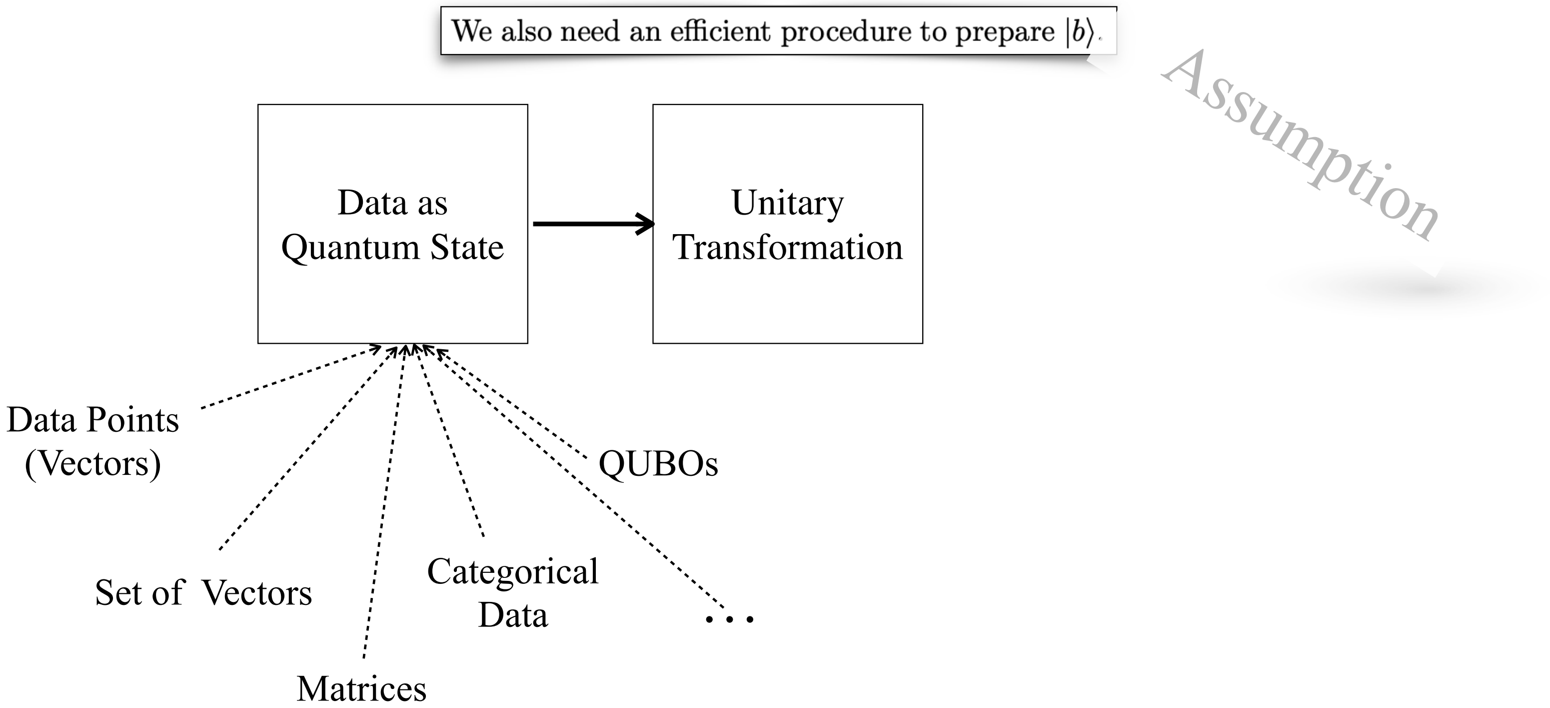
To finish the inversion we measure the last qubit. Conditioned on seeing 1, we have the state

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

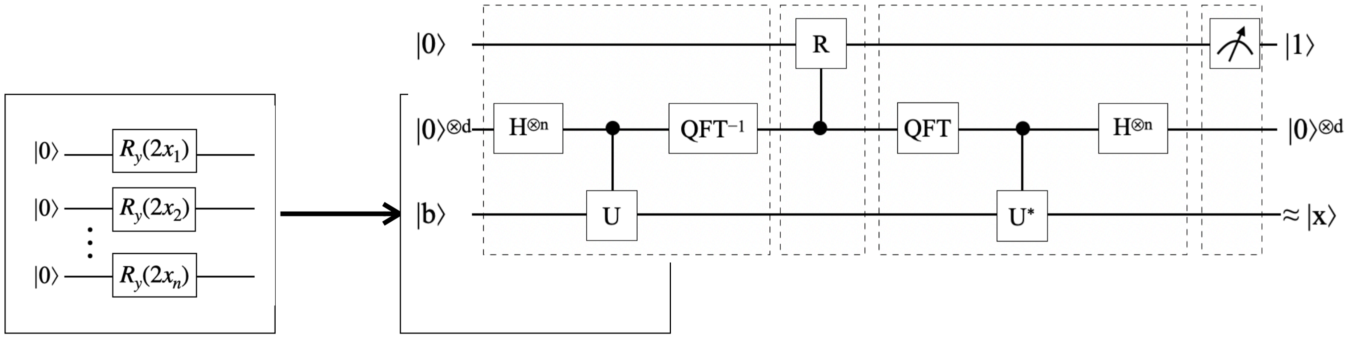
which corresponds to $|x\rangle = \sum_{j=1}^{n} \beta_j \lambda_j^{-1} |u_j\rangle$ up to normalization. We can determine the normalization factor from the probability of obtaining 1. Finally, we make a measurement M whose expectation value $\langle x| M |x\rangle$ corresponds to the feature of $\vec{x}$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



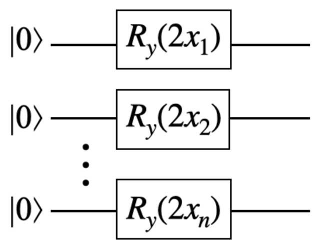
Data for the Algorithm



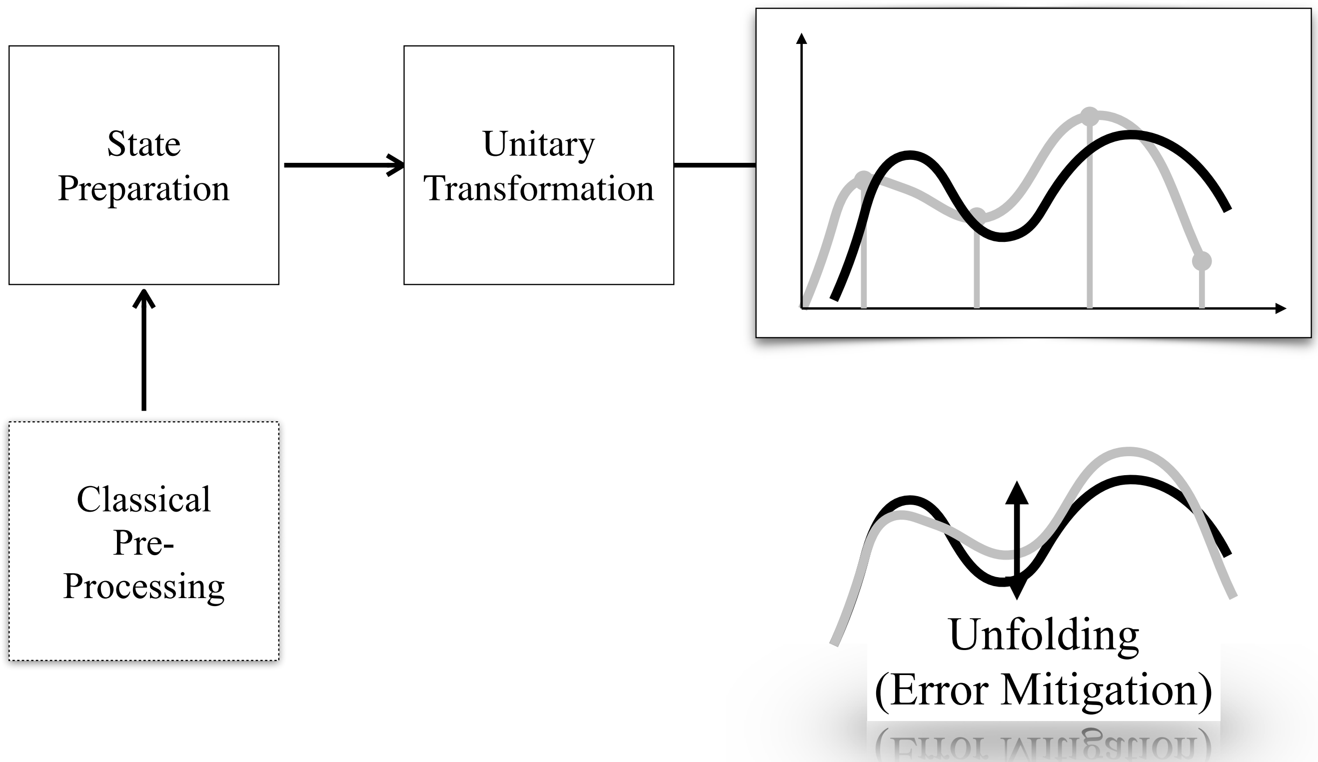
Data as Quantum State



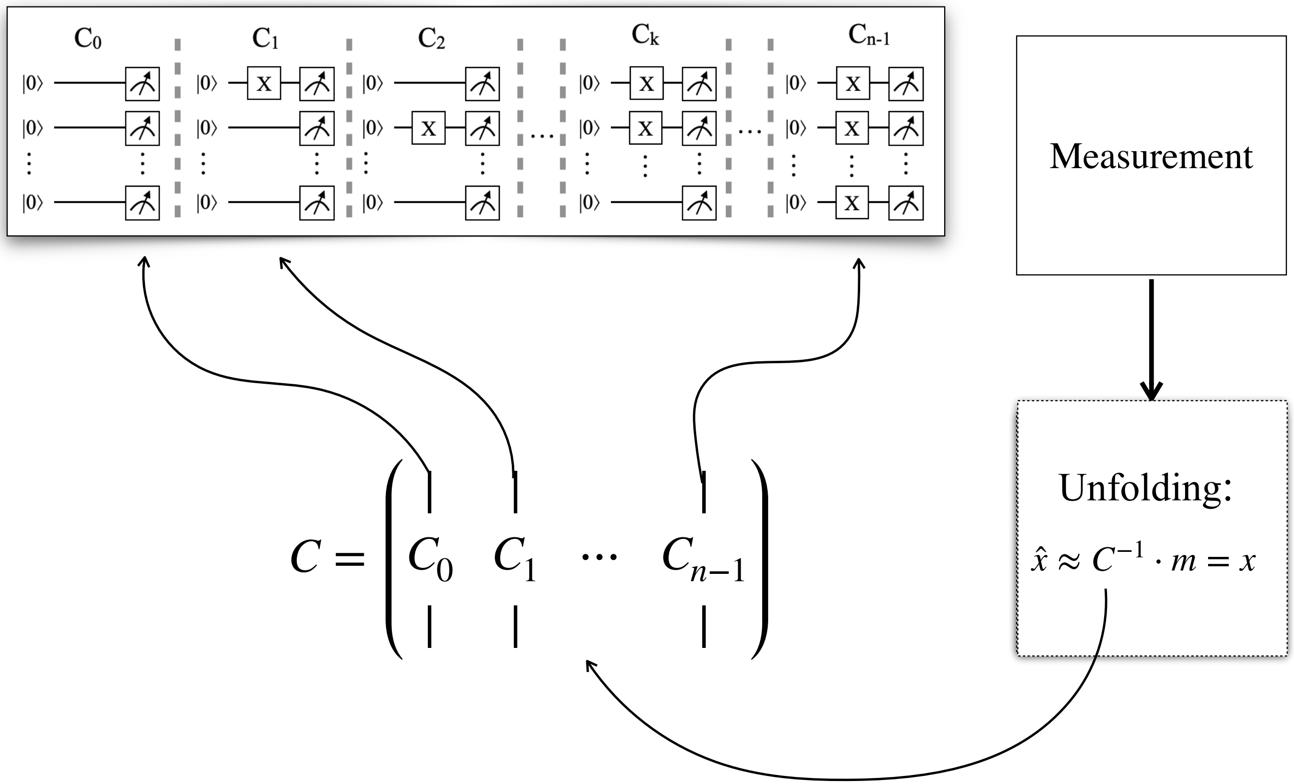
$$(x_1, \ldots, x_N) \mapsto \begin{pmatrix} \cos x_1 \\ \sin x_1 \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} \cos x_N \\ \sin x_N \end{pmatrix}$$



Readout Errors

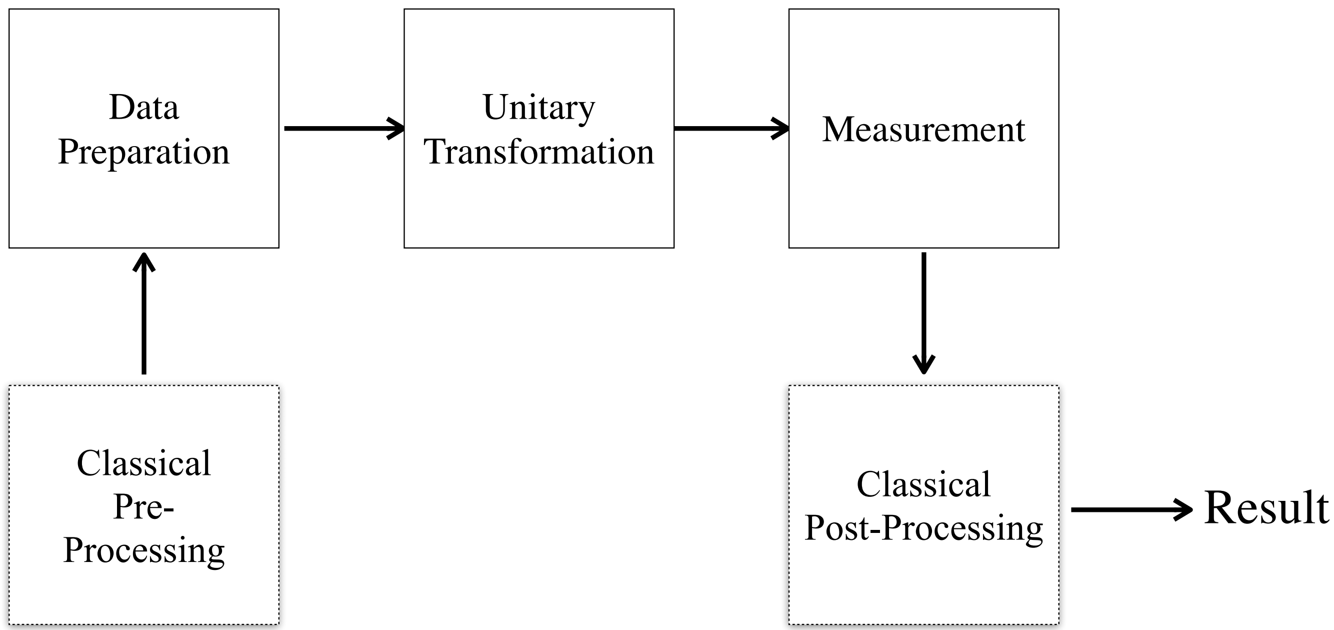


Unfolding



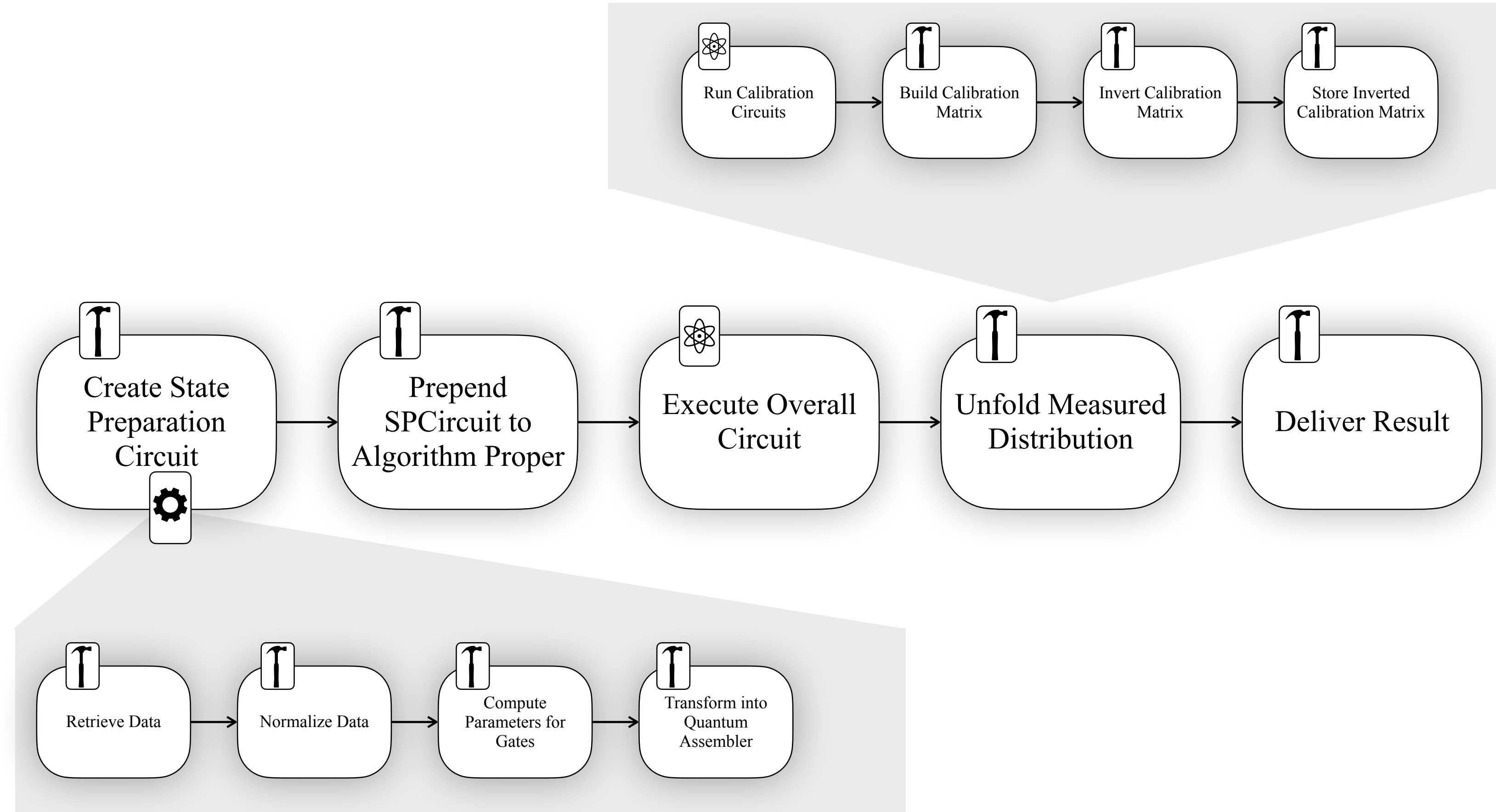
C0
C1
C2
Ck
Cn-1
|0⟩
X
…
Measurement
Unfolding:
$\hat{x} \approx C^{-1} \cdot m = x$
$C = \begin{pmatrix} | & | & & | \\ C_0 & C_1 & \cdots & C_{n-1} \\ | & | & & | \end{pmatrix}$

Structure of a Quantum Algorithm

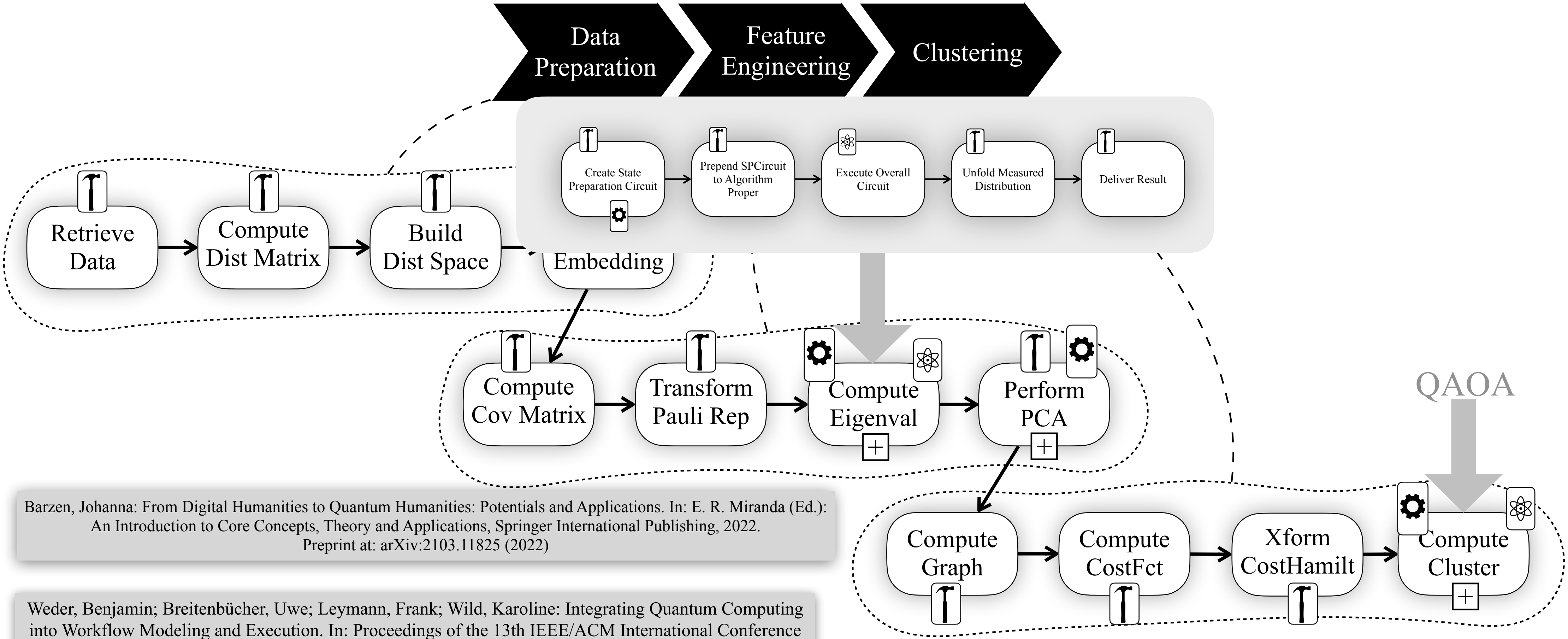


Leymann, Frank; Barzen, Johanna: The bitter truth about gate-based quantum algorithms in the NISQ era. In: Quantum Science and Technology, 2020

Hybrid Quantum-Classical Workflows



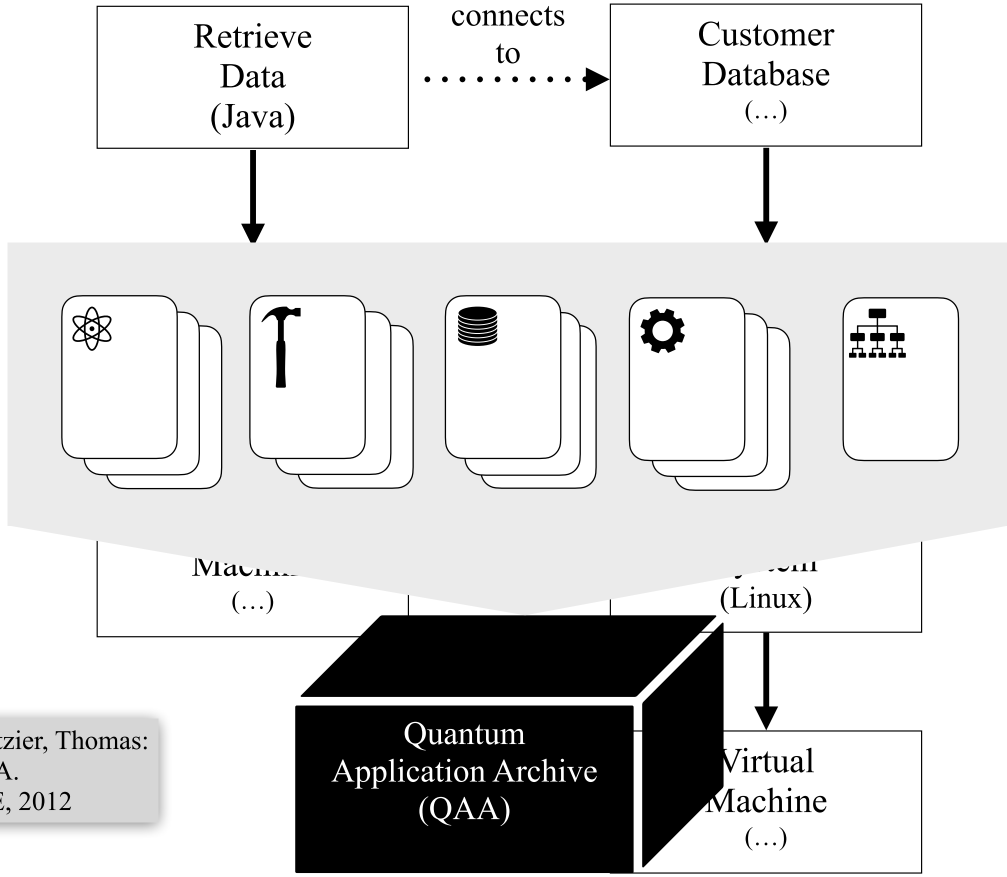
Real Use Case (Sketch)



Barzen, Johanna: From Digital Humanities to Quantum Humanities: Potentials and Applications. In: E. R. Miranda (Ed.): An Introduction to Core Concepts, Theory and Applications, Springer International Publishing, 2022. Preprint at: arXiv:2103.11825 (2022)

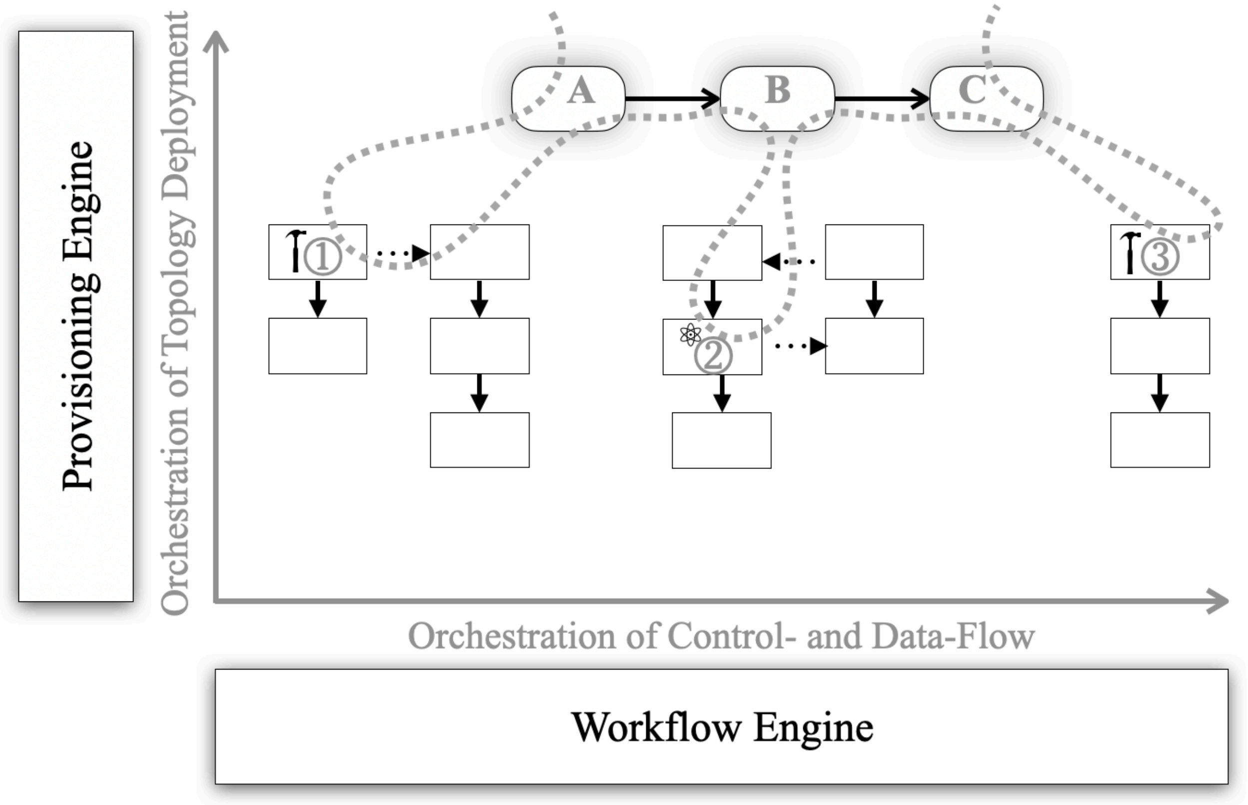
Weder, Benjamin; Breitenbücher, Uwe; Leymann, Frank; Wild, Karoline: Integrating Quantum Computing into Workflow Modeling and Execution. In: Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020), IEEE Computer Society, 2020.

Packaging and Deployment



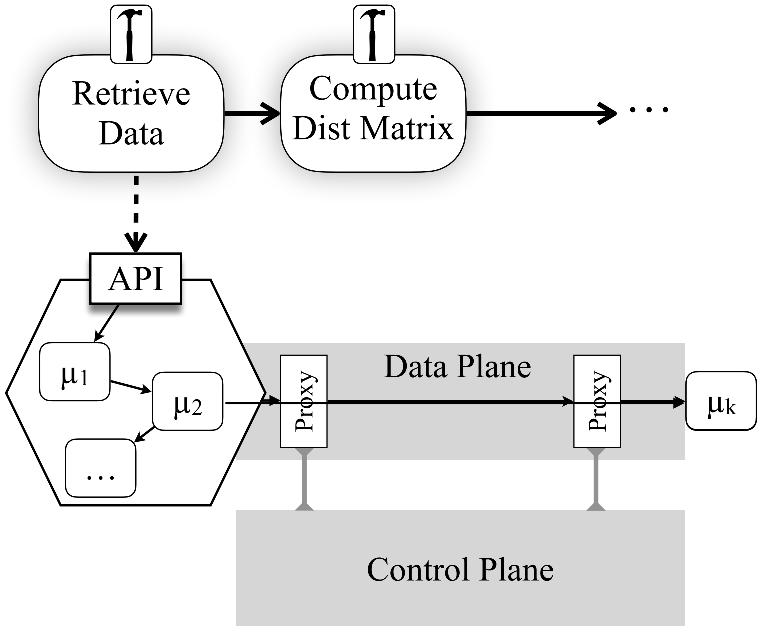
Binz, Tobias; Breiter, Gerd; Leymann, Frank; Spatzier, Thomas:
Portable Cloud Services Using TOSCA.
In: IEEE Internet Computing 16 (03), IEEE, 2012

Two Orchestrations in "Superposition"

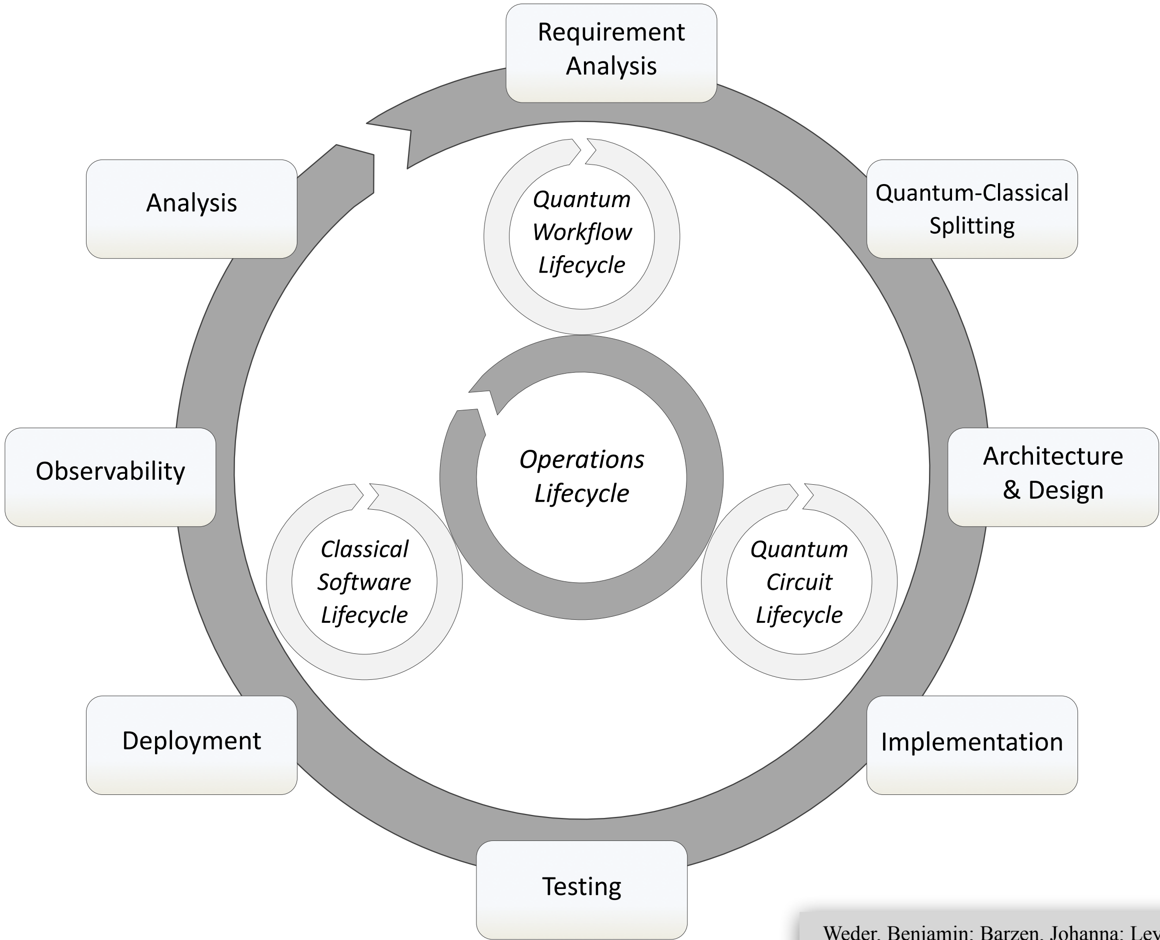


Leymann, Frank; Barzen, Johanna: Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. arXiv:2103.04320, 2021.

The Role of APIs, (Micro-)Services, Service Mesh,…



Quantum Software Lifecycle

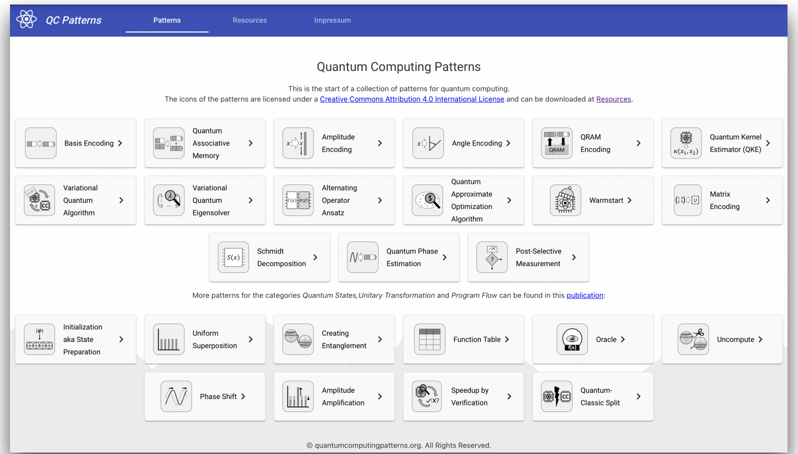


Weder, Benjamin; Barzen, Johanna; Leymann, Frank; Vietz, Daniel: Quantum Software Development Lifecycle. In: Quantum Software Engineering, Springer International Publishing, 2022.

HAna
Quantum Humanities
Analysis Tool

When to use quantum and when to use classical?

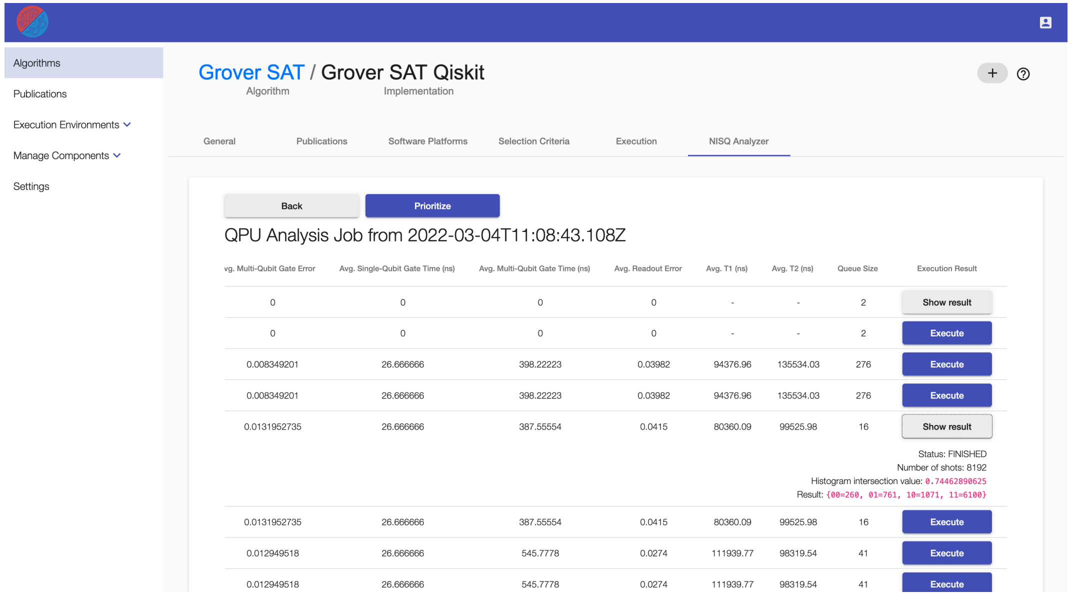
What are best practices to build quantum applications?



Barzen, Johanna: From Digital Humanities to Quantum Humanities: Potentials and Applications. In: E. R. Miranda (Ed.): An Introduction to Core Concepts, Theory and Applications, Springer International Publishing, 2022.
Preprint at: arXiv:2103.11825 (2022)

Leymann, Frank: Towards a Pattern Language for Quantum Algorithms. In: QTOP 2019 Proceedings, Springer, 2019

On which available QPU may my quantum program succeed?



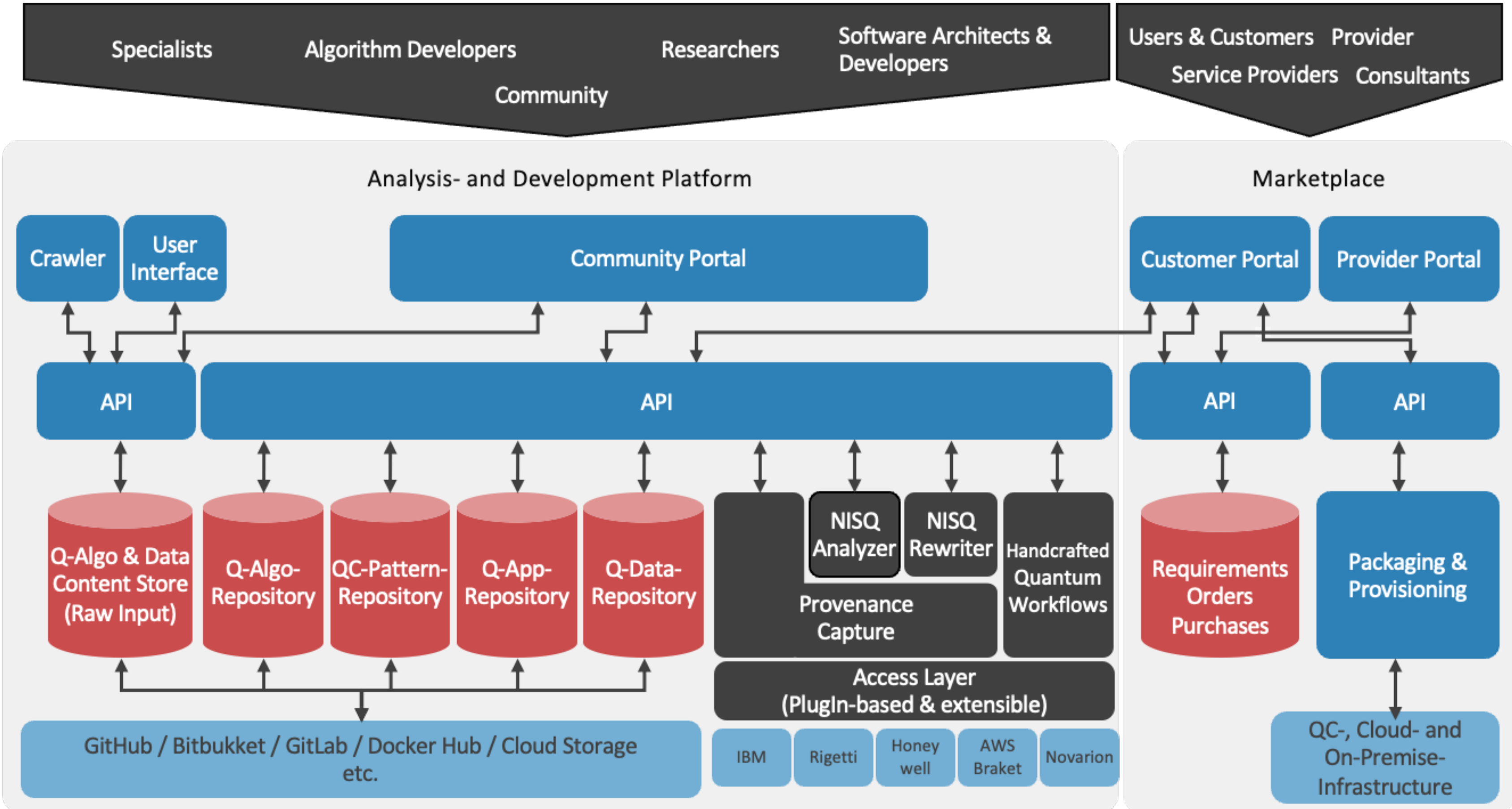
Salm, Marie; Barzen, Johanna; Breitenbücher, Uwe; Leymann, Frank; Weder, Benjamin; Wild, Karoline:
The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms. In: Proc. SummerSOC 2020

Supported by:

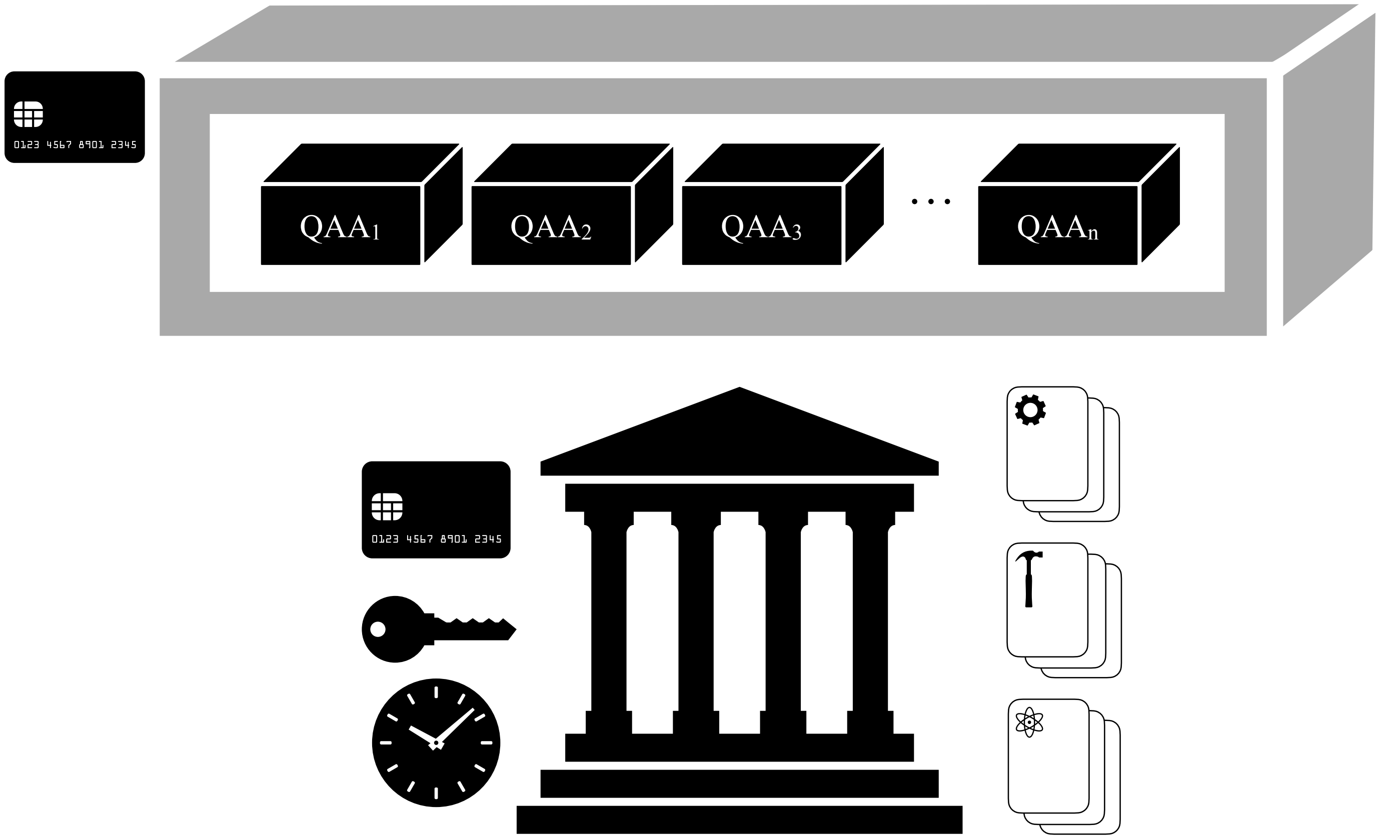
Federal Ministry
for Economic Affairs
and Climate Action

on the basis of a decision
by the German Bundestag

PlanQK



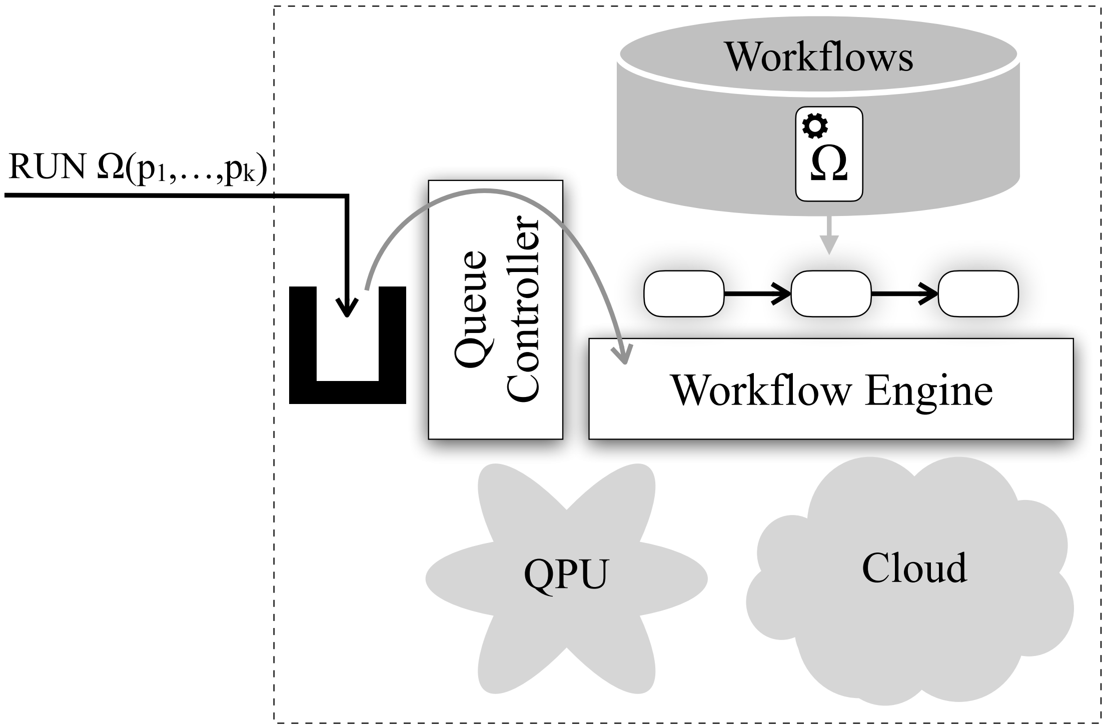
Monetarization: AppStore & API Mgmt



Required Middleware

API Manager	Topology Modeler	Workflow Modeler
Service Mesh	Provisioning Engine	Workflow Engine

Executing a Hybrid Quantum Application



Weder, Benjamin; Breitenbücher, Uwe; Leymann, Frank; Wild, Karoline: Integrating Quantum Computing into Workflow Modeling and Execution. In: Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020), IEEE Computer Society, 2020

Prototype: IBM

Camunda (BPMN Engine)

OpenTOSCA (Provisioning)

QisKit Runtime

Algos

IBM Cloud

Prototype: WSO2 & IBM

Low-Code

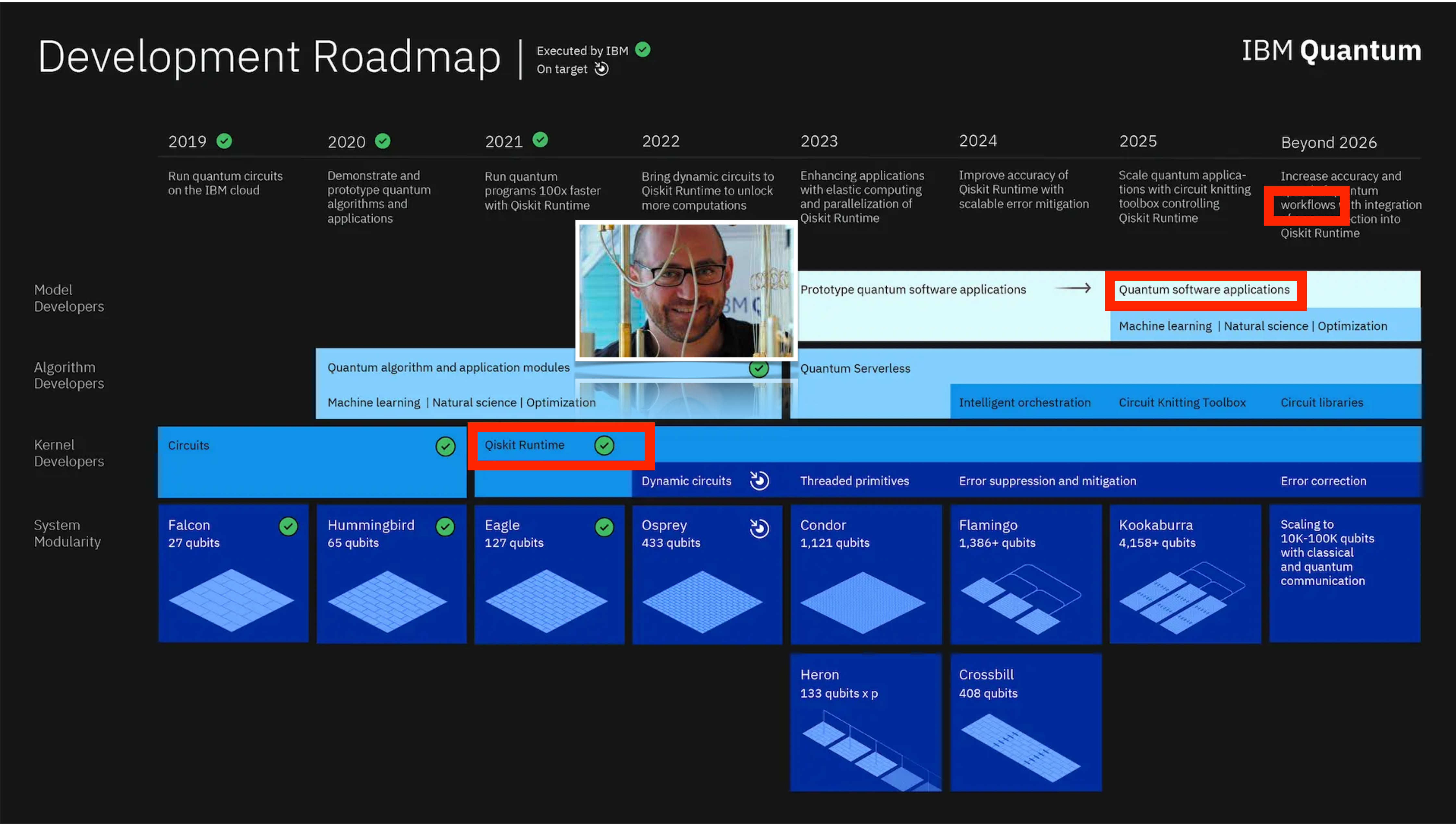
Kubernetes
((hidden))

Circuits

QisKit
Runtime

Algos

Choreo by WSO2



Development Roadmap
Executed by IBM
On target
IBM Quantum
2019
2020
2021
2022
2023
2024
2025
Beyond 2026
Run quantum circuits on the IBM cloud
Demonstrate and prototype quantum algorithms and applications
Run quantum programs 100x faster with Qiskit Runtime
Bring dynamic circuits to Qiskit Runtime to unlock more computations
Enhancing applications with elastic computing and parallelization of Qiskit Runtime
Improve accuracy of Qiskit Runtime with scalable error mitigation
Scale quantum applications with circuit knitting toolbox controlling Qiskit Runtime
Increase accuracy and
workflows
Qiskit Runtime
Model Developers
Prototype quantum software applications
Quantum software applications
Machine learning | Natural science | Optimization
Algorithm Developers
Quantum algorithm and application modules
Machine learning | Natural science | Optimization
Quantum Serverless
Intelligent orchestration
Circuit Knitting Toolbox
Circuit libraries
Kernel Developers
Circuits
Qiskit Runtime
Dynamic circuits
Threaded primitives
Error suppression and mitigation
Error correction
System Modularity
Falcon 27 qubits
Hummingbird 65 qubits
Eagle 127 qubits
Osprey 433 qubits
Condor 1,121 qubits
Flamingo 1,386+ qubits
Kookaburra 4,158+ qubits
Scaling to 10K-100K qubits with classical and quantum communication
Heron 133 qubits x p
Crossbill 408 qubits

Conclusion

- Real-world quantum applications are hybrid
- Such hybrid quantum-classical applications require orchestrations
  - Workflows: Orchestration of control- and dataflow of steps of the application
  - Provisioning: Orchestration of deployment of the infrastructure and code of the application
- As a result, quantum applications become packages
  - …which can be used as tradable artifacts
- Quantum applications can be deployed and executed on premise or in a cloud environment or mixed
- Building Quantum Application is a integration problem

The End