HQSNoiseApp: Working with Noisy Quantum Computers

Workshop

Munich, Germany – 11.05.2023

Name: Dr. Konstantina Alexopoulou
Business Development & Grant Manager

Bundesministerium
für Bildung
und Forschung

Q-EXA

HQS
QUANTUM
SIMULATIONS

Backgroung & Objectives

Background

- Leibniz Supercomputing Center (LRZ) purchased HQSNoiseApp 2022 in the context of the Q-Exa Grant.
- LRZ asked HQS to set up a training for their users on HQSNoiseApp.

Today´s Objectives

- Provide participants with knowledge of starting working with HQSNoiseApp Software.

HQS Team

LRZ Workshop

Dr. Konstantina Alexopoulou

Title: Business Development and Grant Manager

Role in the workshop: Introduction & Overview of the Workshop

Dr. Pascal Stadler

Title: Expert and software developer in the field of quantum simulation

Role in the workshop: Hands-on Training

Dr. Giorgio Silvi

Title: Expert in the field of quantum algorithms and quantum computing

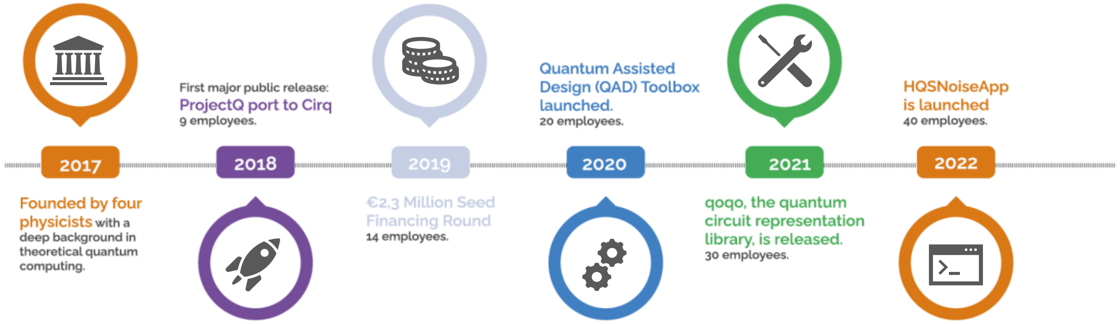
Role in the workshop: Hands-on Training

Agenda

- **09:30** Introduction: Supercomputing and HQS
  Dr. Konstantina Alexopoulou
- **10:00** Connection to the QLM system at LRZ (15 minutes)
  LRZ
- **10:15** Introduction to quantum computing
  Dr. Pascal Stadler
- **10:30** Break
- **10:45** Qoqo / Roqo: Introduction & Examples
  Dr. Giorgio Silvi
- **11:30** Struqture: Introduction & Examples
  Dr. Pascal Stadler
- **12:15** Lunch Break
- **13:15** Time Evolution: Introduction & Examples
  Dr. Giorgio Silvi
- **14:15** Noisy Algorithm Model (Basic) Introduction & Examples
  Dr. Pascal Stadler
- **15:00** Break
- **15:15** Noisy Algorithm Model (Basic) Introduction & Examples
  Dr. Pascal Stadler
- **16:30** Open Discussion / Feedback

HQS at a glance

HQS has grown strongly every year since its inception



Collaborations with BASF, Bosch, Merck, Total, Covestro, AstraZeneca and others.

BASF We create chemistry

BOSCH

MERCK

TotalEnergies

covestro

AstraZeneca

8 different countries

That employees come from

25%

of our employees are female

75%

of our employees have a PhD

HQS at a glance

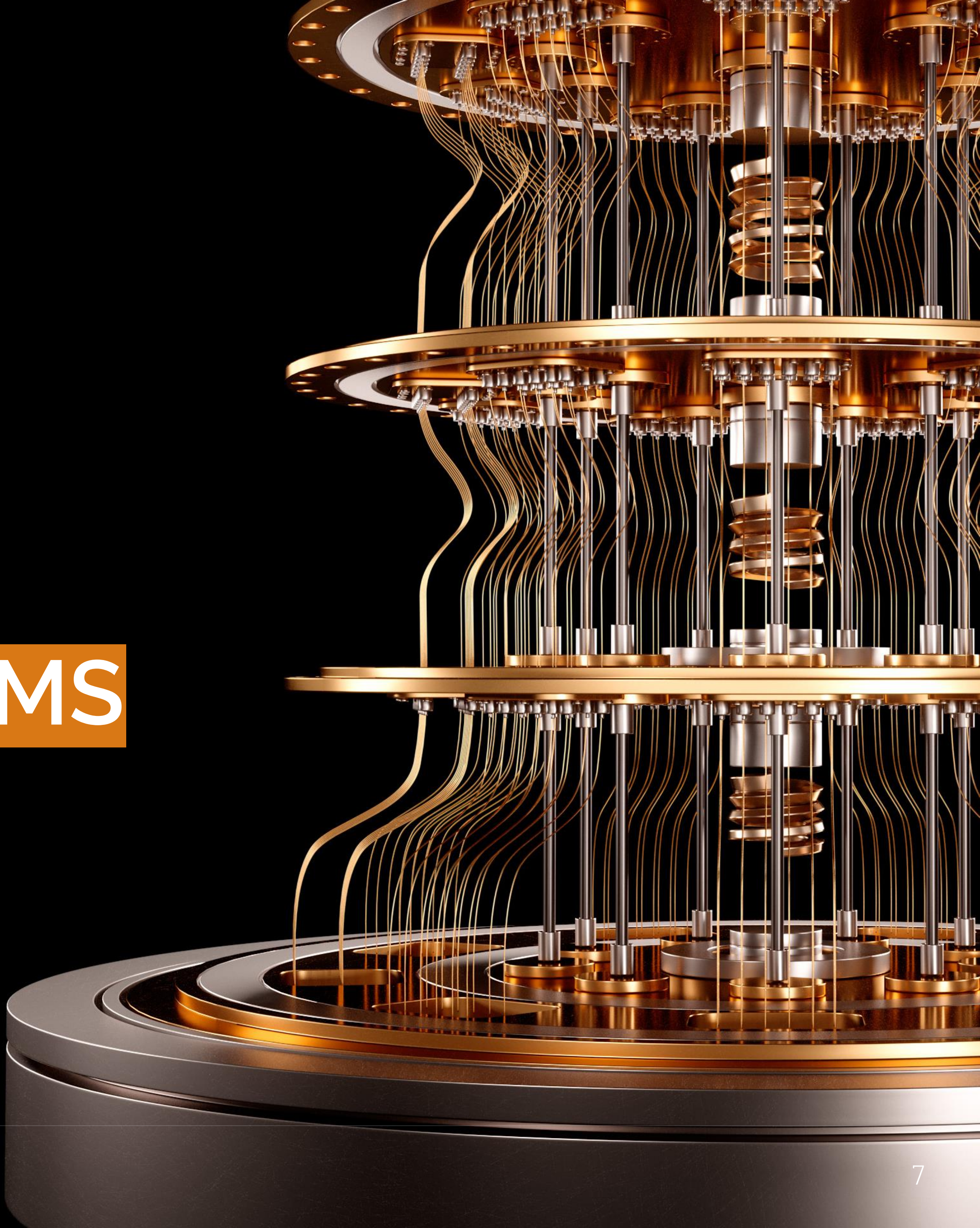
Research Grants supporting our work

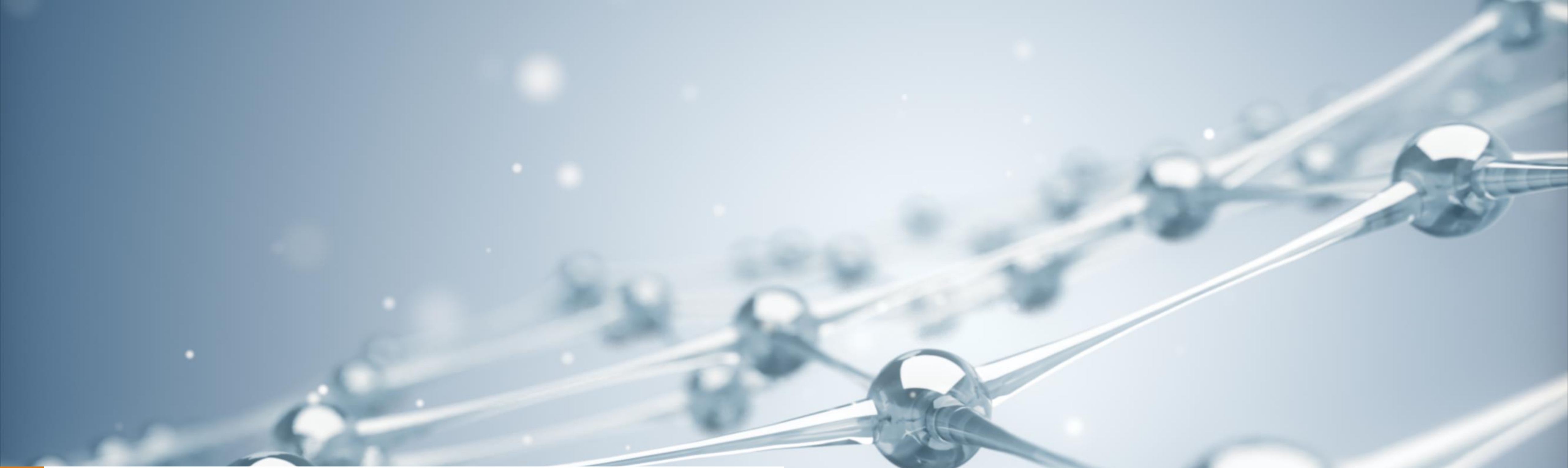
PlanQK, BMWK	QSolid, BMBF
AQUAS, BMWK	Q-Exa, BMBF
QUASAR, BMBF	NEASQC, EU
MANIQU, BMBF	Avaqus, EU
PhoQuant, BMBF	BRISQ, EU

Image Credit: Andy Sproles, ORNL, image darkened

HQS works on using quantum computers to solve

OPEN QUANTUM SYSTEMS





HPC Centers and Quantum Computing

The Way Forward: Bringing HPC and Quantum Computing Together

High-performance computing (HPC) is a key tool to address the most challenging problems faced by our society

Quantum meets HPC

27%

HPC centers are already experimenting with quantum computing.

76 %

HPC centers worldwide plan to use quantum computing by 2023.

71%

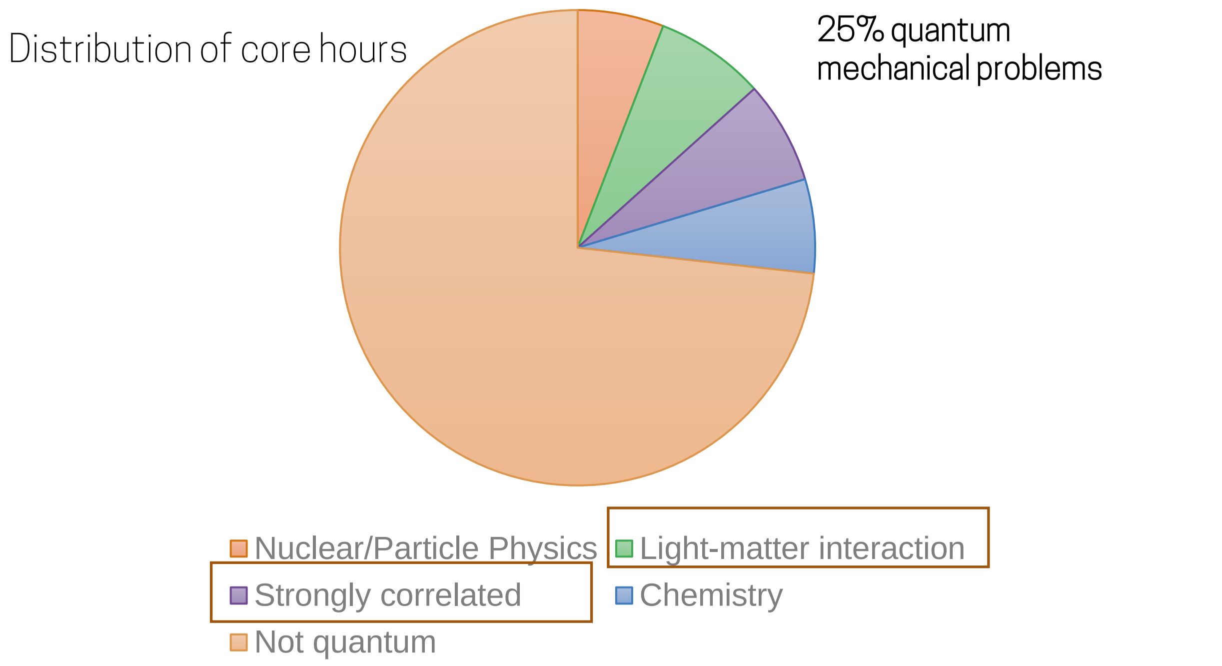
Plan to move to on-premises quantum computing by 2026.

https://ml-eu.globenewswire.com/Resource/Download/40710644-657f-43db-be00-fca17f0b77be

The Way Forward: Bringing HPC and Quantum Computing Together

High-performance computing (HPC) is a key tool to address the most challenging problems faced by our society

Quantum meets HPC



INCITE leadership program (2021)

The Way Forward: Bringing HPC and Quantum Computing Together

Two main topics are to be mapped to the quantum computer / Examples of projects

Light-Matter Interaction

e.g. Large-scale simulations of light-matter interaction

This project carries out … quantum simulations … so as to tackle … the design of sustainable materials that efficiently capture and convert solar energy.

Machine: Cray XC40

Node-hours: 1.200.000

Strongly Correlated Systems

e.g. Towards predictive simulations of functional and quantum materials

Goal of this project: prediction and understanding of quantum-mechanical properties of materials that display novel properties including novel quantum phases.

Machine: Cray XC 40

Node-hours: 1.200.000

Machine: IBM AC922

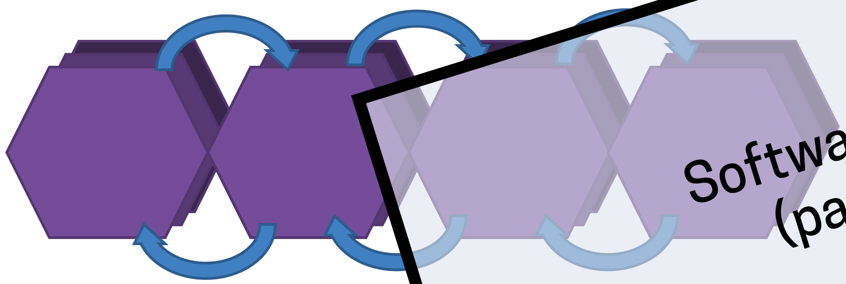
Node-hours: 500.000

The Way Forward: Bringing HPC and Quantum Computing Together

LRZ: Working with HQSNoiseApp: Technological Pioneer in these fields

Light-Matter Interaction

e.g. Photosynthesis, organic solar cells, excitonic interactions.

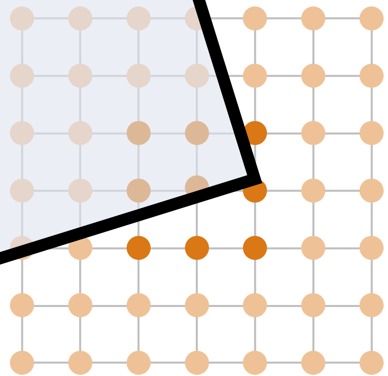


Resonance energy transfer

HQSNoiseApp Software available starting 01.07.2022

Strongly Correlated Systems

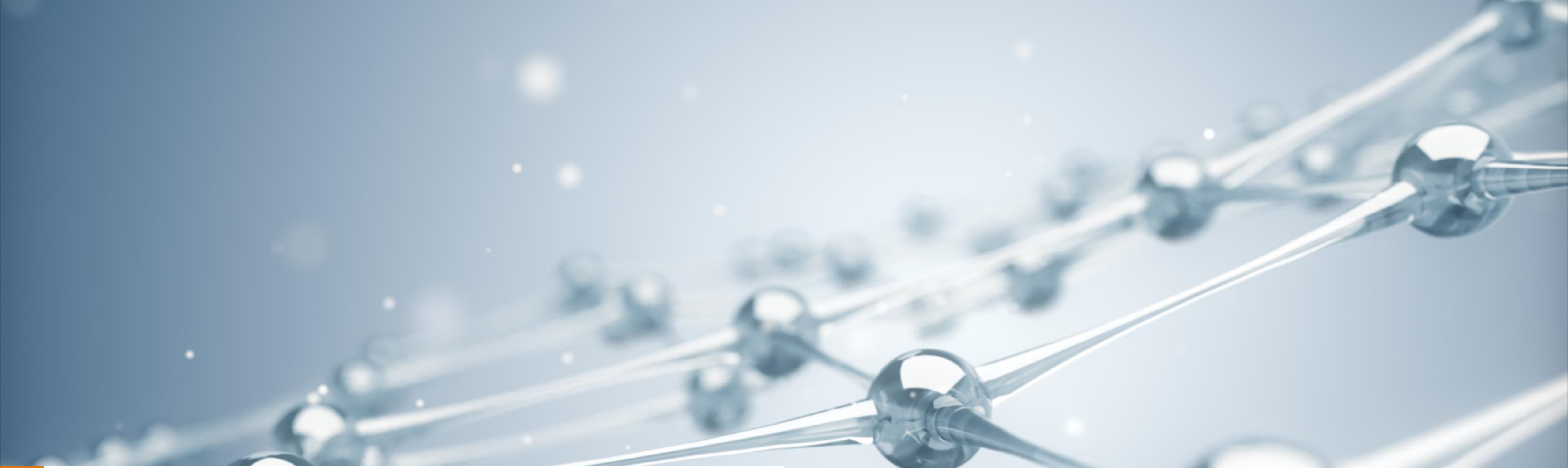
e.g. superconductivity, quantum phase transitions, magnetism



Lattice Models

HQSNoiseApp Software available starting 31.01.2023

Software available for LRZ users
(part of the Q-Exa project)



HQS QUANTUM SIMULATIONS

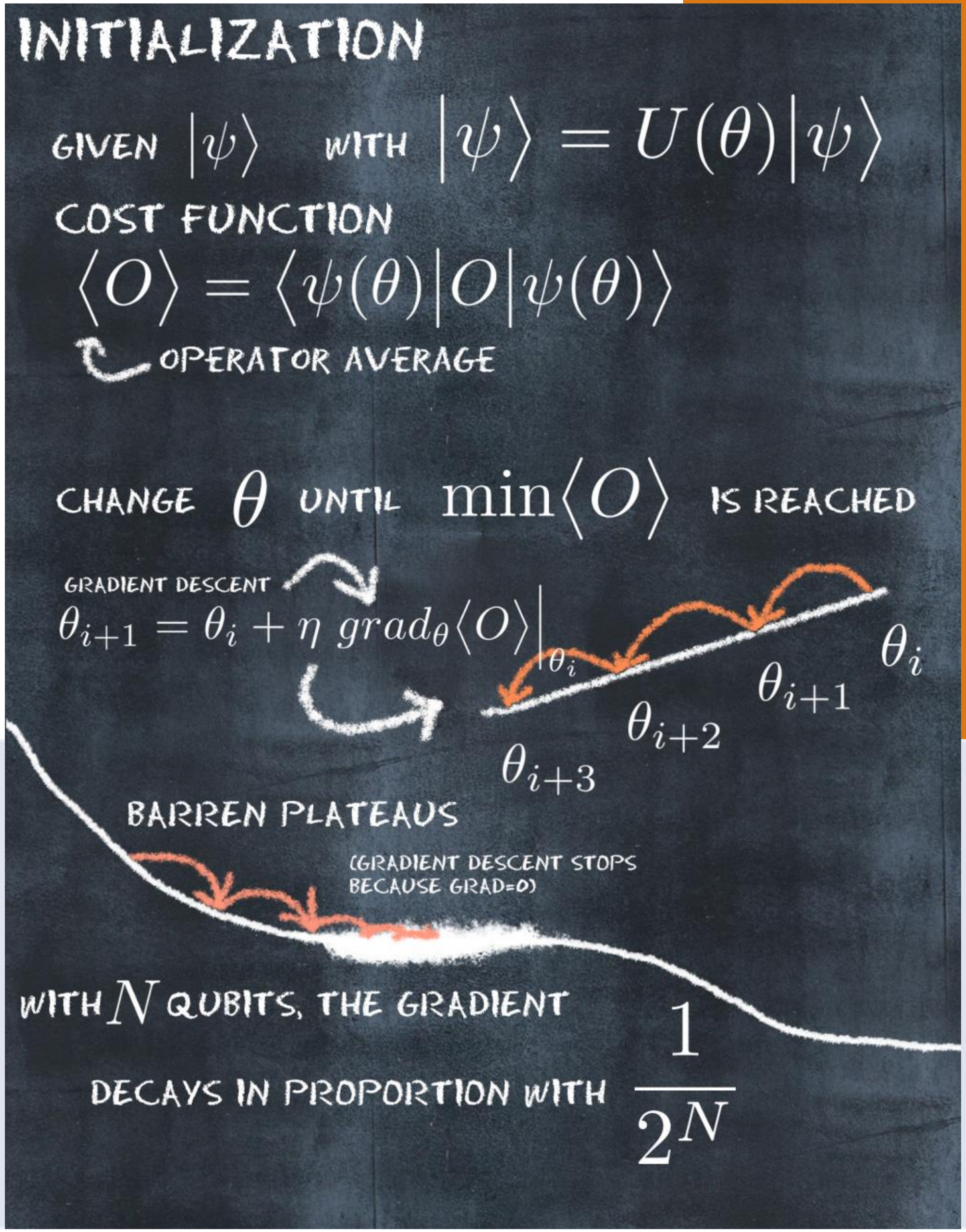
Variational algorithms and their limits

Will variational algorithms work?

The current standard path to applications for NISQ computers is variational algorithms.

However, variational algorithms have two fundamental weaknesses:

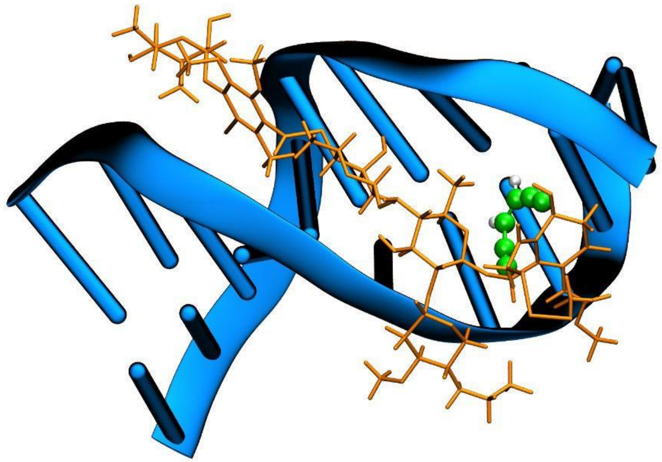
- The number of operations is too large for NISQ computers.
- The time to optimize the variational parameters often scales exponentially with the number of qubits.



HQS Alternative approach

Simulate Quantum Systems on Quantum Computers

Time Evolution: Proven exponential advantage on quantum computers



Time Evolution

For a given physical Hamiltonian, the quantum computer can efficiently create the time evolution $U(t) = e^{iH_s t}$

We need to get all the relevant properties we want from time evolution.

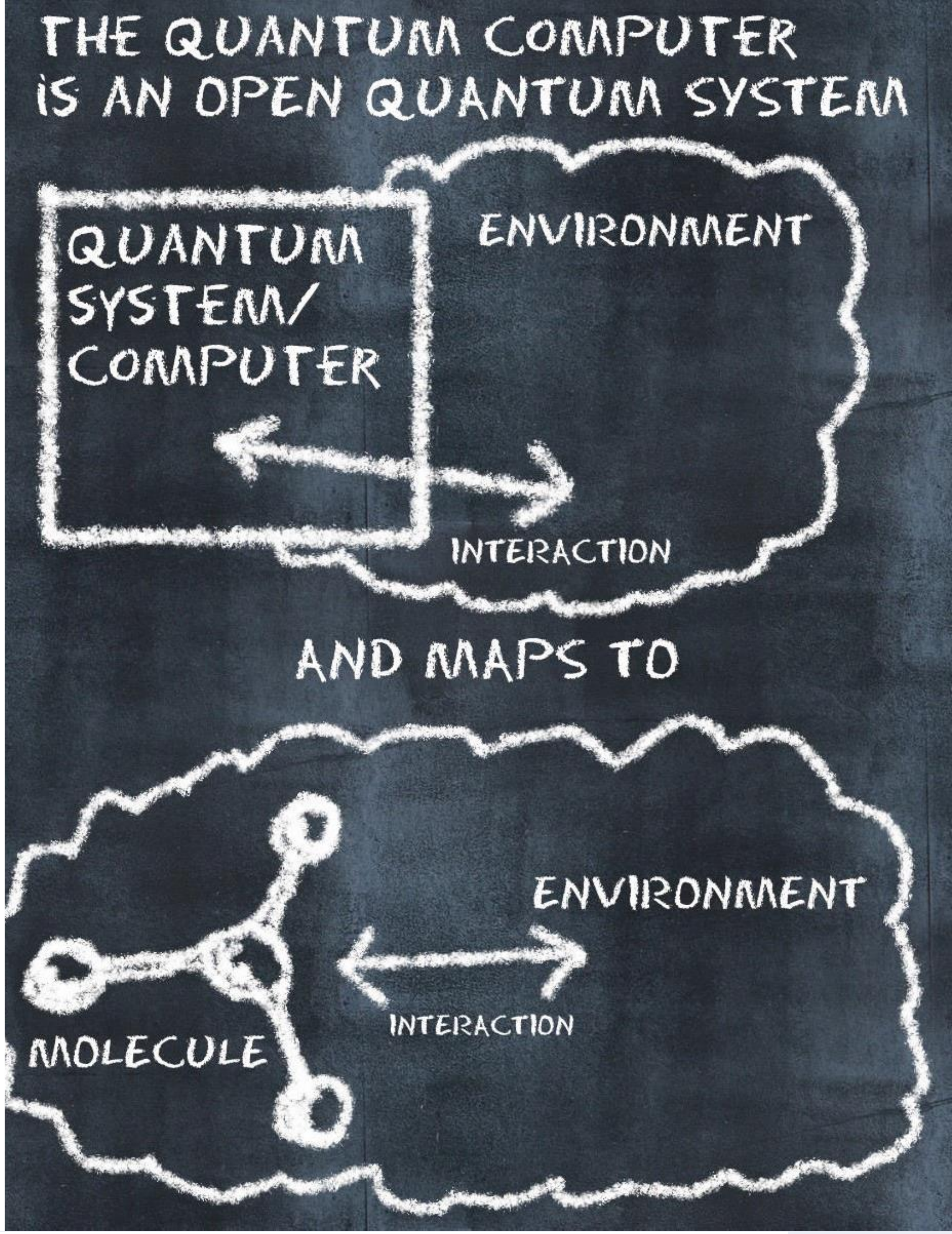
HQS Software: Integrating the error into the algorithm

We need to simulate the systems as they are in nature:

Time-evolving open quantum systems

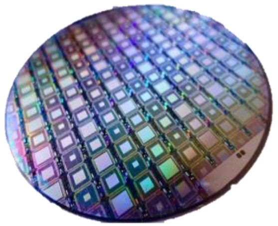
HQS solves this problem by making the error part of the algorithm.

All quantum systems are embedded in an environment that can be mapped to quantum computers and turn a source of errors into computing power.



Our Approach

HQSNoiseApp Workflow

Define problem	Device specification	HQSNoiseApp	Gates and instructions	Simulator or actual device
H_0	qoqo BY HQS	$\mathcal{L}_{eff}$	qoqo BY HQS	

TWO PAPERS EXPLAIN OUR APPROACH IN DETAIL

1. A quantum algorithm for solving open quantum system dynamics on quantum computers using noise.
   Read more: https://arxiv.org/abs/2210.12138
2. Describing Trotterized Time Evolutions on Noisy Quantum Computers via Static Effective Lindbladian.
   Read more: https://arxiv.org/abs/2210.11371

Our Approach

Struqrure: Definition of spin Halmitonian

Define problem

$$H_0$$

Device specification

>

HQSNoiseApp

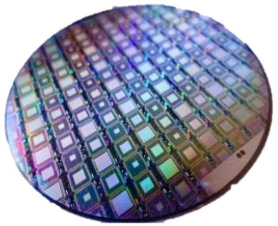
$$\mathcal{L}_{eff}$$

>

Gates and instructions

>

Simulator or actual device



Specification via Struqture as a spin model

Our Approach

Qoqo: HQS toolkit to represent quantum circuits

Define problem

H_0

Device specification

qoqo BY HQS

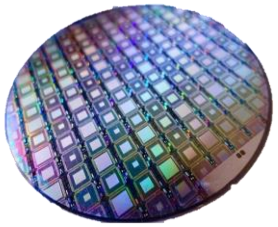
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

qoqo BY HQS

Simulator or actual device



Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

Our Approach

Define problem

H_0

Device specification

qoqo BY HQS

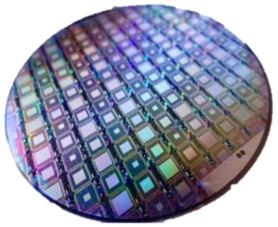
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

qoqo BY HQS

Simulator or actual device



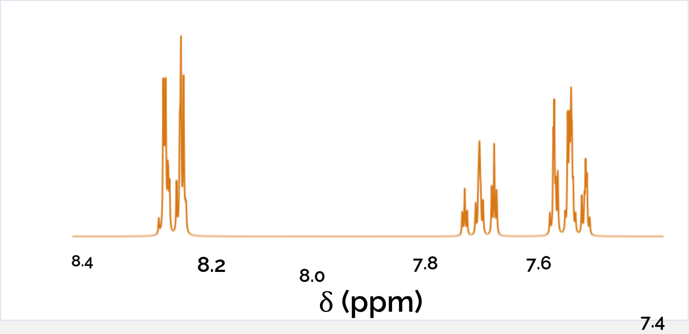
Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

Internal simulator in qoqo or QLM

Simulate Quantum Systems on Quantum Computers

HQSNoiseApp in practice / Use Case: Nuclear Magnetic Resonance

Challenge: NMR spin problem



- Spectroscopic technique that is used in chemistry, and pharma to study the structure and properties of molecules
- Measures the interactions of nuclear spins when placed in a powerful magnetic field
- Simulation using NISQ devices: high levels of noise → limitation on their ability to perform more advanced NMR simulations

Approach: HQSNoiseApp

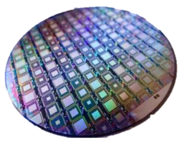
Define Problem — H_0

Device Specification — qoqo BY HQS

HQSNoiseApp — $\mathcal{L}_{eff}$

Gates and instructions — qoqo BY HQS

Simulator or actual device



Benefits: Integrates the noise into the algorithm

- More accurate simulations
- Integration of noise
- More efficient resource utilization

Agenda

09:30 Introduction: Supercomputing and HQS
Dr. Konstantina Alexopoulou

10:00 Connection to the QLM system at LRZ (15 minutes)
LRZ

10:15 Introduction to quantum computing
Dr. Pascal Stadler

10:30 Break

10:45 Qoqo / Roqo: Introduction & Examples
Dr. Giorgio Silvi

11:30 Struqture: Introduction & Examples
Dr. Pascal Stadler

12:15 Lunch Break

13:15 Time Evolution: Introduction & Examples
Dr. Giorgio Silvi

14:15 Noisy Algorithm Model (Basic) Introduction & Examples
Dr. Pascal Stadler

15:00 Break

15:15 Noisy Algorithm Model (Basic) Introduction & Examples
Dr. Pascal Stadler

16:30 Open Discussion / Feedback

HQS QUANTUM SIMULATIONS

Thank you for your attention!

info@quantumsimulations.de
www.quantumsimulations.de

Introduction to Quantum Computing

HQS QUANTUM SIMULATIONS

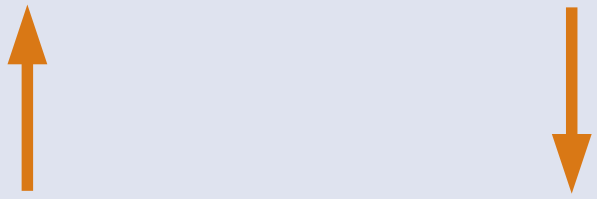
Introduction to Quantum Computing

- Qubit, operations and circuits
- Decoherence theory
- Physical realisations

Qubits, operations, circuits

Bit

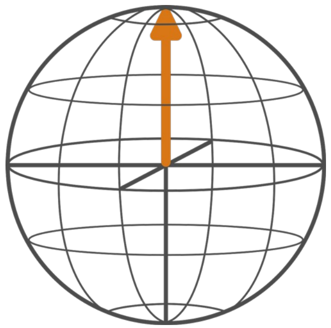
Switch with two positions: state either 0 or 1



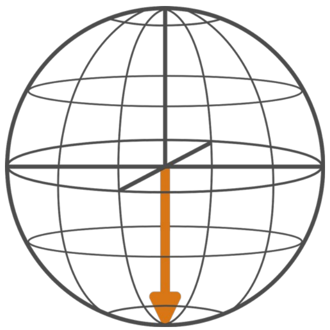
0 $\qquad$ 1

Qubit

Quantum mechanical two-state system



$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$



$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Superposition: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$

Geometrical representation: Bloch sphere

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

Measurement

Conventional computer: We can examine a bit to determine if it is in state 0 or 1.

Quantum computer: We can not determine the quantum state.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

$$|\alpha|^2 + |\beta|^2 = 1$$

Result 0 with probability $|\alpha|^2$

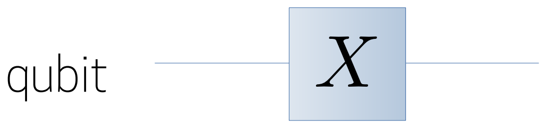
Result 1 with probability $|\beta|^2$

Quantum operations

Quantum operators by unitary gates

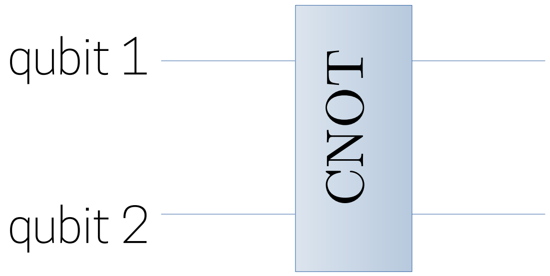
Single-qubit gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$



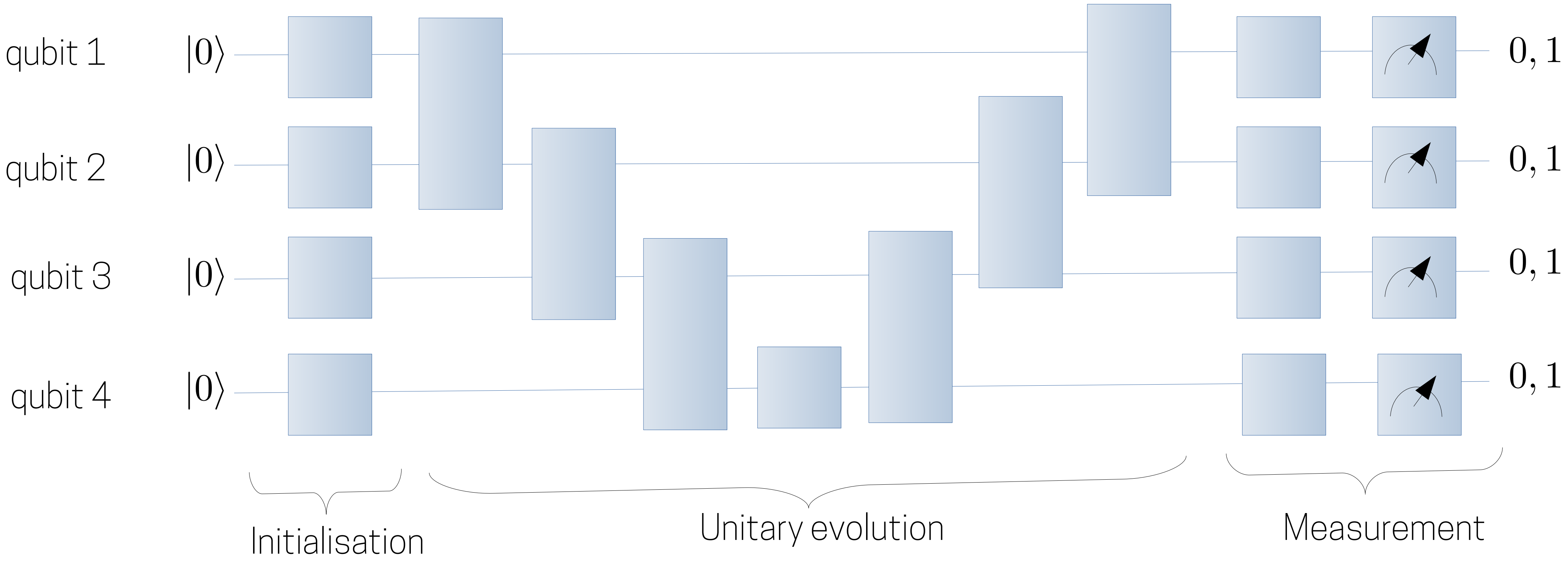
Two-qubit operators

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \mathrm{CNOT} \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{11} \\ \alpha_{10} \end{pmatrix}$$



Universal set: Hadamard, phase, pi/8 gate, CNOT

Circuits



qubit 1
qubit 2
qubit 3
qubit 4
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$0, 1$
$0, 1$
$0, 1$
$0, 1$
Initialisation
Unitary evolution
Measurement

Decoherence theory

Decoherence theory

Close system: $\boxed{H = H_S}$

- Schrödinger equation $|\psi(t)\rangle = e^{-iH_S t}|\psi(0)\rangle$

system

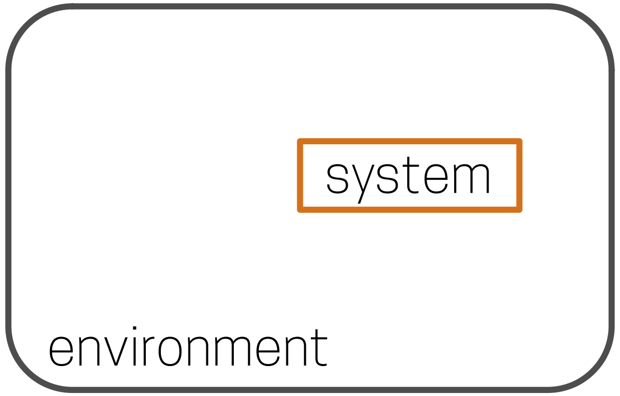
Decoherence theory

Close system: $\boxed{H = H_S}$

- Schrödinger equation $|\psi(t)\rangle = e^{-iH_S t}|\psi(0)\rangle$

Open quantum system: $\boxed{H = H_S + H_C + H_E}$

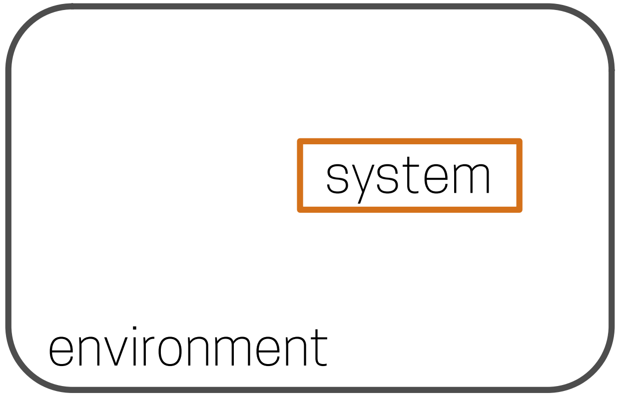
- von Neumann equation $\dot{\rho}_{\text{tot}}(t) = -i\left[H, \rho_{\text{tot}}\right]$



Decoherence theory

Close system: $\boxed{H = H_S}$

- Schrödinger equation $|\psi(t)\rangle = e^{-iH_S t}|\psi(0)\rangle$



Open quantum system: $\boxed{H = H_S + H_C + H_E}$

- von Neumann equation $\dot{\rho}_{\mathrm{tot}}(t) = -i\,[H, \rho_{\mathrm{tot}}]$
- Lindblad-master equation

rate matrix

left and right Lindblad operators

$$\dot{\rho} = \mathcal{L}(\rho) = -i[H_S, \rho] + \sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)$$

Examples: Damping and dephasing

$$\boxed{\dot{\rho} = \mathcal{L}(\rho) = -i[\hat{H}, \rho] + \sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)}$$

Lindblad operators dephasing

$$A_j = A_k = \sigma_z$$

Lindblad equation

$$\dot{\rho} = -i[\hat{H}, \rho] + \gamma_{\text{dephasing}} \left(\sigma_z \rho \sigma_z - \rho\right)$$

Rate matrix

$$M_{\sigma_z, \sigma_z} = \gamma_{\text{dephasing}}$$

Examples: Damping and dephasing

$$\dot{\rho} = \mathcal{L}(\rho) = -i[\hat{H}, \rho] + \sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)$$

Lindblad operators dephasing

$$A_j = A_k = \sigma_z$$

Lindblad equation

$$\dot{\rho} = -i[\hat{H}, \rho] + \gamma_{\text{dephasing}} \left(\sigma_z \rho \sigma_z - \rho\right)$$

Rate matrix

$$M_{\sigma_z, \sigma_z} = \gamma_{\text{dephasing}}$$

Lindblad operators for damping

$$A_j = A_k = \sigma^+ = \frac{1}{2}\left(\sigma^x + \mathrm{i}\sigma^y\right)$$

Lindblad equation

$$\begin{aligned}\dot{\rho} &= -i[\hat{H}, \rho] \\ &+ \gamma_{\text{damping}} \left(\sigma^+ \rho \sigma^- - \frac{1}{2}\sigma^- \sigma^+ \rho - \frac{1}{2}\rho \sigma^- \sigma^+ \right)\end{aligned}$$

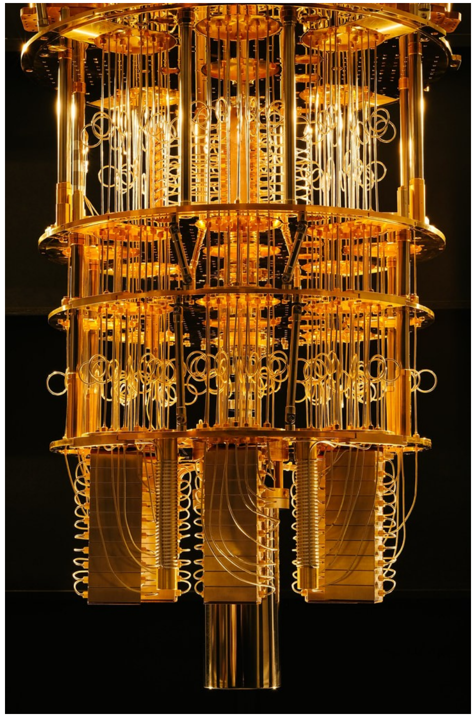
Rate matrix

$$M_{\sigma_x, \sigma_x} = M_{\sigma_x, \mathrm{i}\sigma_y} = M_{\mathrm{i}\sigma_y, \sigma_x} = M_{\mathrm{i}\sigma_y, \mathrm{i}\sigma_y} = \gamma_{\text{damping}}/4$$

Physical realizations

Superconducting qubits

- Superconducting qubits consists of a loop a superconductivting wire interrupted by small insulating layers.
- Different types of superconducting qubits: transmon qubit and flux qubit. Most common type is a transmon
- Connectivity: depends on device (linear, square, ..)
- Two-qubit gates
  - CZ, SWAP, iSWAP,

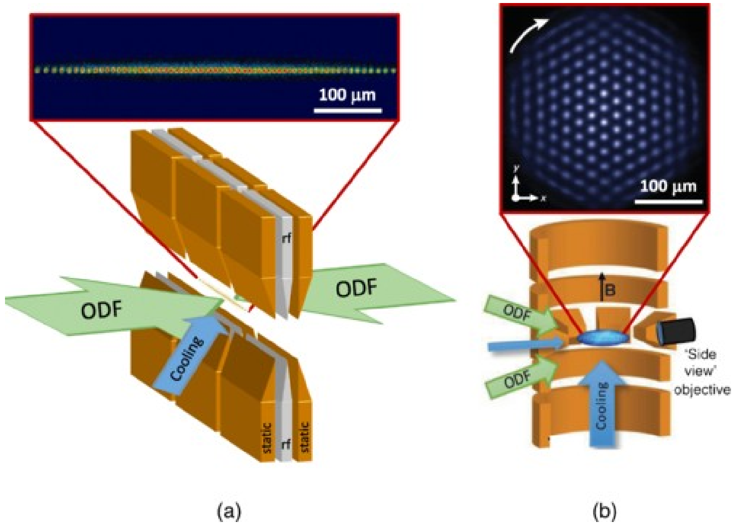


Linear connectivity

Ion traps

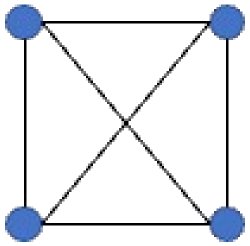
- Ion trapped quantum computers use charged atoms to store and manipulate quantum information
- Ions are confined using electromagnetic fields and manipulated with laser pulses to perform quantum operators
- Connectivity: all ions can interact (all-to-all)
- Two-qubit gate: Mølmer-Sørensen gate

$$\mathrm{VariableMSXX}(\theta) = \begin{pmatrix} \cos(\theta/2) & 0 & 0 & -\mathrm{i}\sin(\theta/2) \\ 0 & \cos(\theta/2) & -\mathrm{i}\sin(\theta/2) & 0 \\ 0 & -\mathrm{i}\sin(\theta/2) & \cos(\theta/2) & 0 \\ -\mathrm{i}\sin(\theta/2) & 0 & 0 & \cos(\theta/2) \end{pmatrix}$$



Rev. Mod. Phys. 93, 025001

All-to-all connectivity



HQS
QUANTUM
SIMULATIONS

HQS`s Qoqo / RoQoqo

HQS QUANTUM SIMULATIONS

Our Approach

Define problem

H_0

Device specification

qoqo by HQS

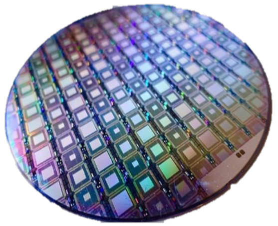
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

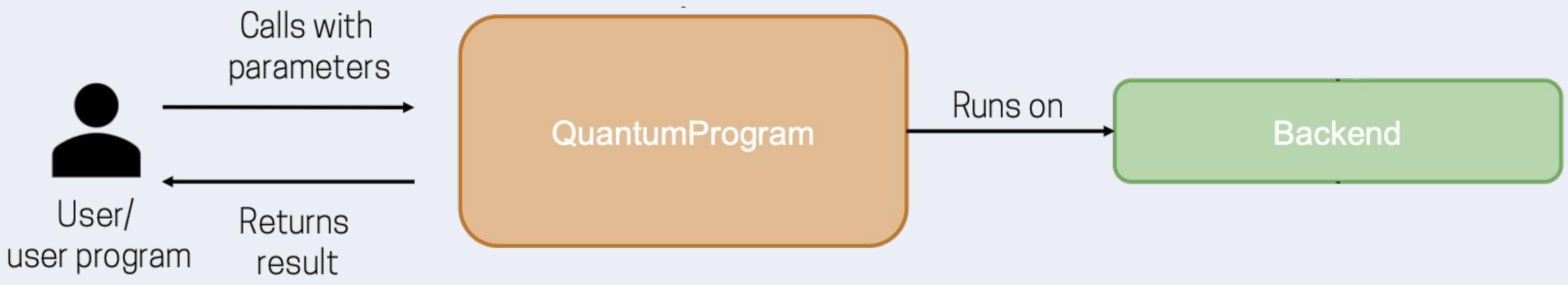
qoqo by HQS

Simulator or actual device



Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

Why Qoqo?



Ease of Use

(optional) Python interface with minimal dependencies

Portability

Linux, macOS, Windows, x86, ARM

Speed

Fast symbolic variable evaluation and Rust core (roqoqo)

What is Qoqo?

qoqo

BY HQS

What Qoqo is:

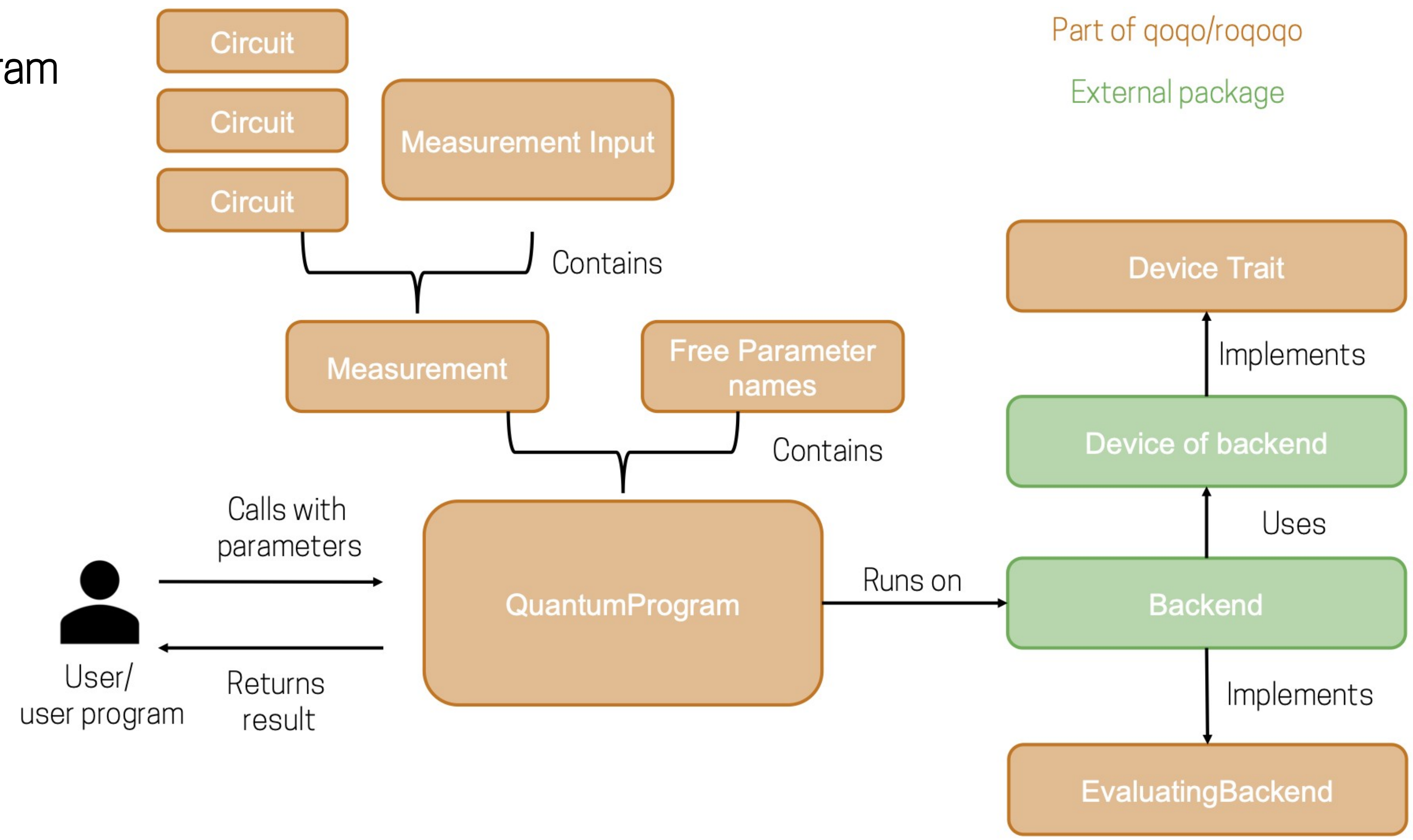
- A toolkit to represent quantum programs including circuits and measurement information
- A thin runtime to run quantum measurements
- A way to serialize quantum program
- A set of optional interfaces to devices and simulators

What Qoqo is not:

- A decomposer translating circuits to a specific set of gates
- A quantum circuit optimizer
- A collection of quantum algorithms

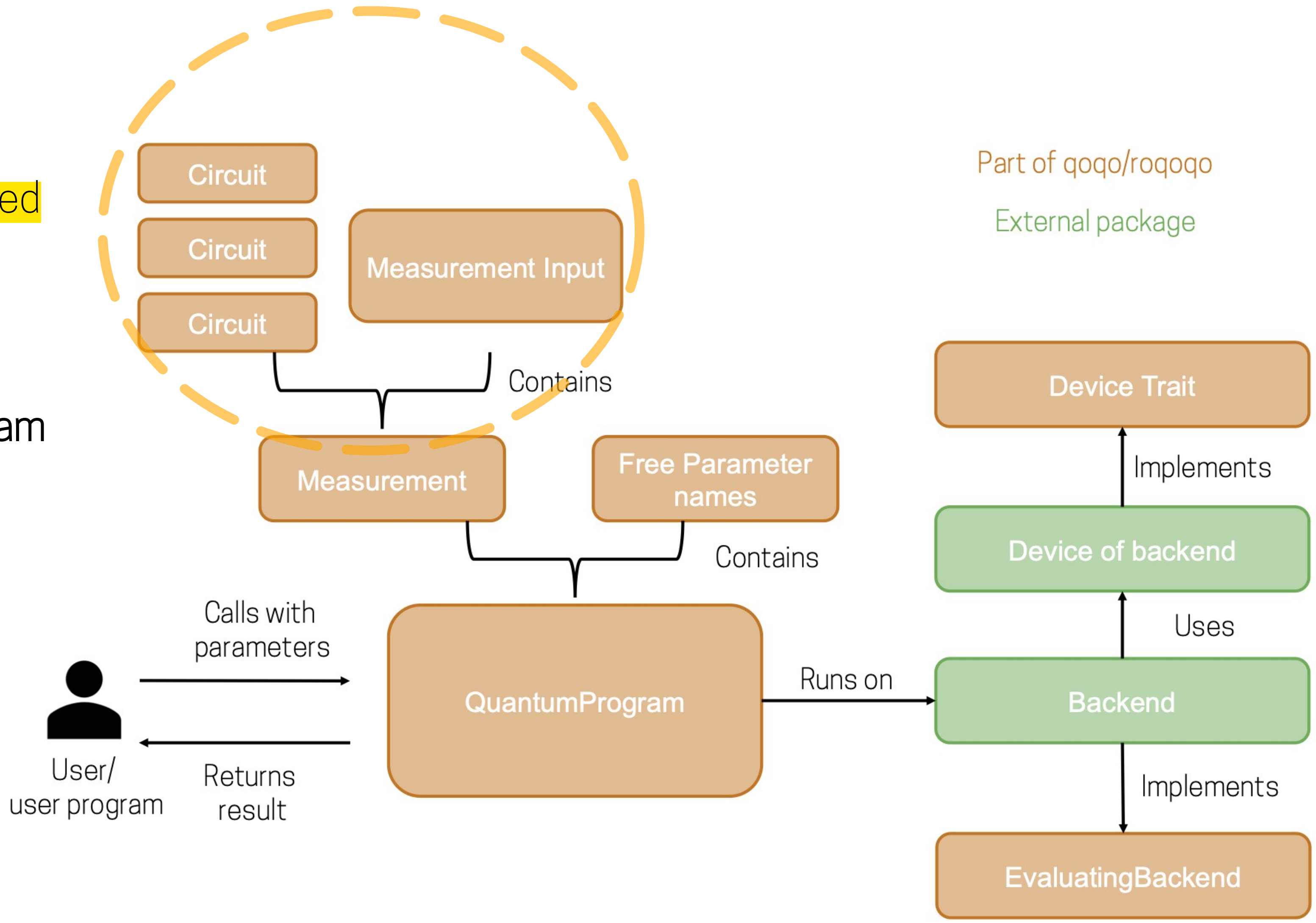
Architecture

- Operations and Circuit
- Measurements and QuantumProgram
- Backends
- Devices



Architecture

- Operations and Circuit
  - Operations and Circuit can be used to represent single quantum circuits that run on quantum hardware.
- Measurements and QuantumProgram
- Backends
- Devices



Operations and Circuit

Three main types:

- **Definitions**: Operations that declare *classical* register values
- **Gate**: Unitary operations that can be executed on *every* unitary quantum computer
- **Pragma**: operations that help model the noise on simulators (as well as compilation directives, e.g. loop)

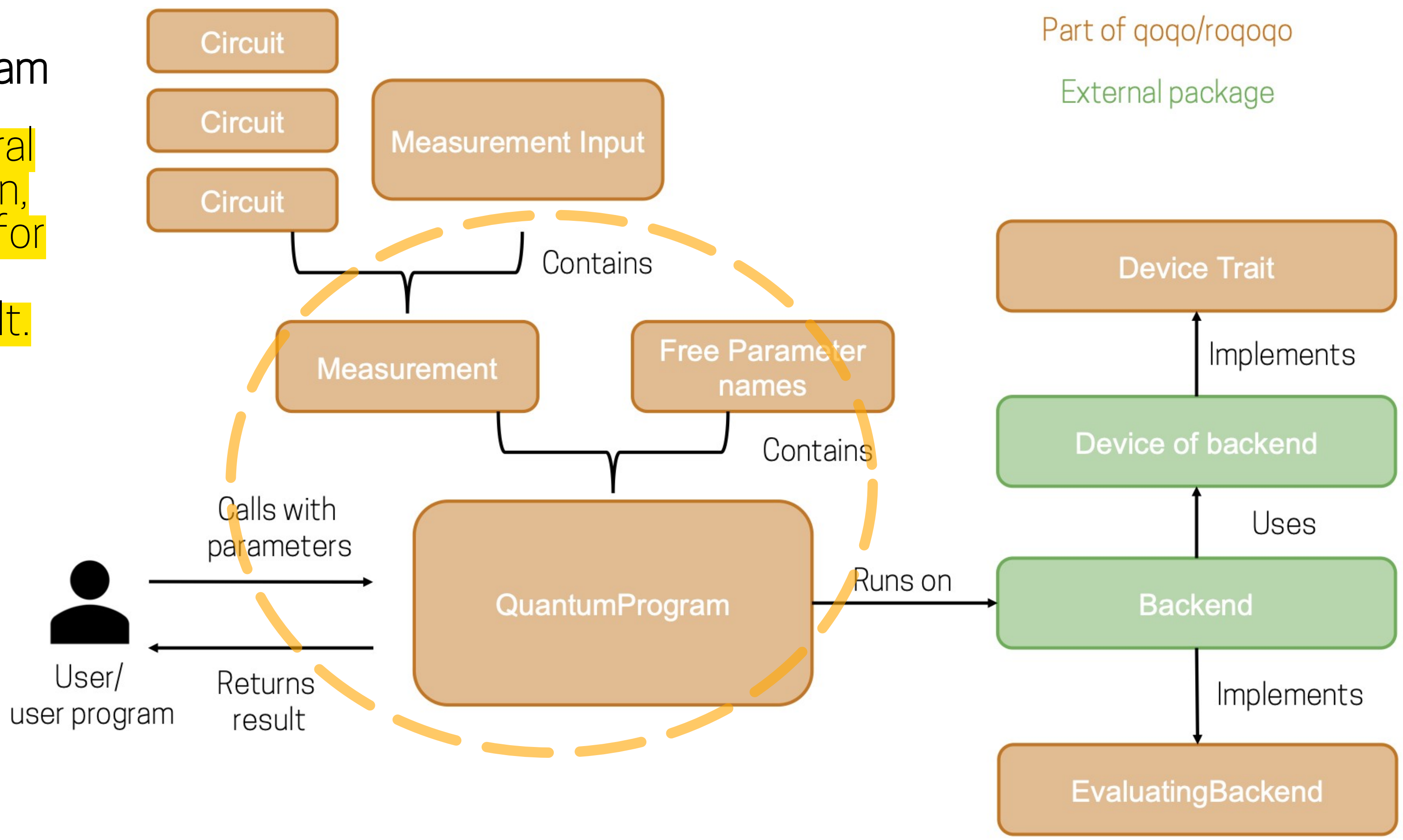
In order to create a useful result, a Circuit in qoqo must contain:

- A definition of a classical register for readout
- Operations to change the state of the quantum computer, for example RotateZ or CNOT gate operations.
- A measurement to return classical information based on the state of the quantum computer.

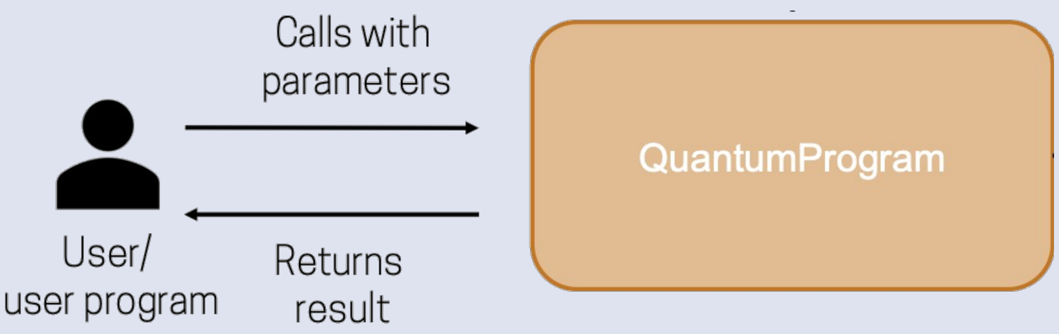
```
from qoqo import Circuit
from qoqo import operations as ops
# create a new circuit
circuit = Circuit()
# Define the readout for two qubits
circuit += ops.DefinitionBit(name="ro", length=2, is_output=True)
# Rotation around Z axis by pi/2 on qubit 0
circuit += ops.RotateZ(qubit=0, theta=1.57)
# Entangling qubits 0 and 1 with CNOT gate
circuit += ops.CNOT(control=0, target=1)
# Measuring the qubits
circuit += ops.MeasureQubit(qubit=0, readout="ro", readout_index=0)
circuit += ops.MeasureQubit(qubit=1, readout="ro", readout_index=1)
```

Architecture

- Operations and Circuit
- Measurements and QuantumProgram
  - QuantumProgram combine several circuits with classical information, to provide a high-level interface for running quantum programs that yield an immediately usable result.
- Backends
- Devices



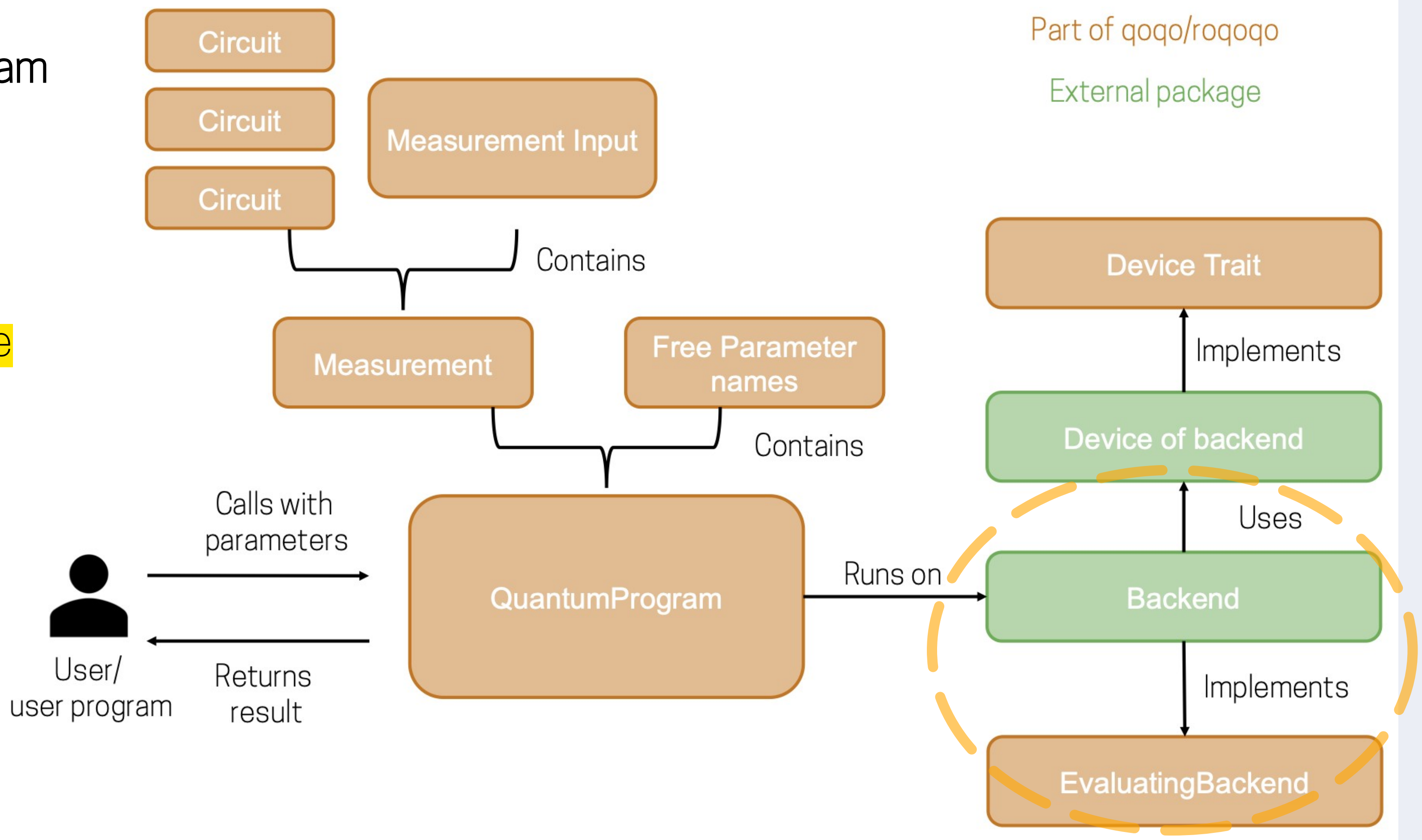
Quantum Program



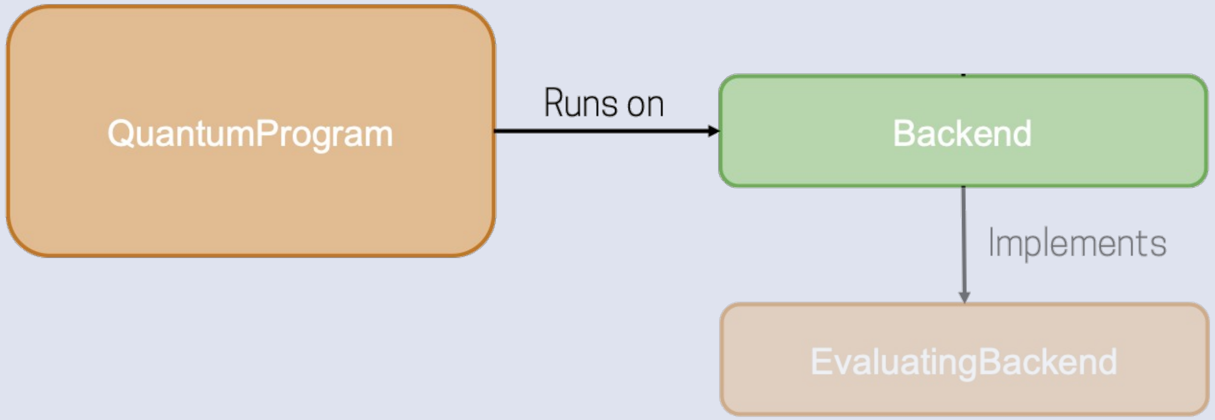
- Qoqo wraps quantum circuits and measurement information into a **QuantumProgram**
- It can contain free parameters and can be serialized.
- Ideal for non-trivial quantum algorithm
- It exchange small amount of data:
  - ✔ Input is minimal (e.g. atom types).
  - ✔ Output is JSON serializable and yields processed results (not bitstrings)
  - ✔ Ideal for WebAPI
  - ✔ Little communication overhead
  - ✔ Can be called many times for different calculations

Architecture

- Operations and Circuit
- Measurements and QuantumProgram
- Backends
  - To execute quantum circuits or quantum programs, a backend connecting to quantum hardware or a simulator is required.
- Devices

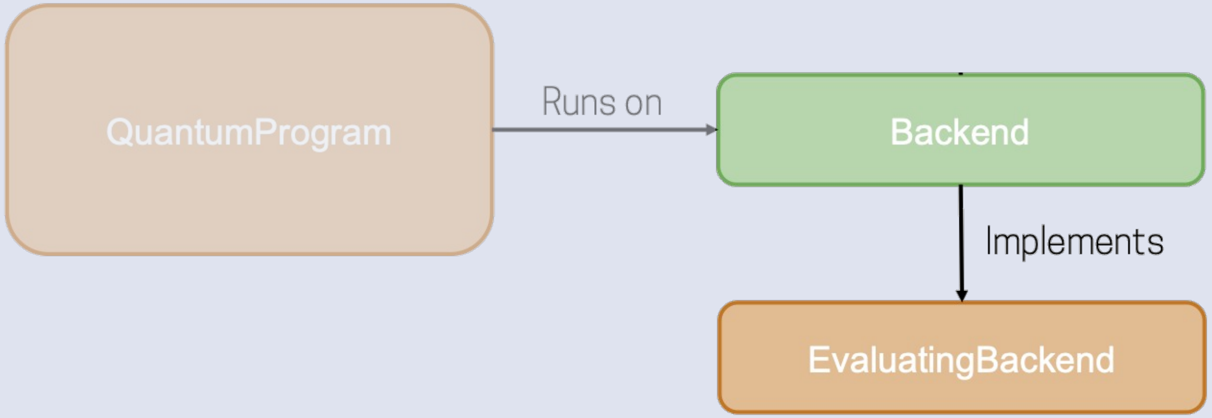


Backends



- Backends are used for:
  - Running quantum programs and obtaining results from them
  - Translating qoqo objects to other frameworks
- To minimize dependencies, Backends are implemented in separate packages, e.g.:
  - qoqo-quest QuEST
  - qoqo-myqlm myQLM
  - ...

Backends



- EvaluatingBackend provides the functionality to run:
  - A *single* circuit. The backend will execute just the circuit and return the measurement results of all registers in a tuple (bit-registers, float-registers, complex-registers).

```
# a) Run a single circuit
(bit_registers, float_registers, complex_registers) = backend.run_circuit(circuit)
```

  - A measurement. *All* circuits collected in the measurement are executed and the post-processed expectation values are returned.

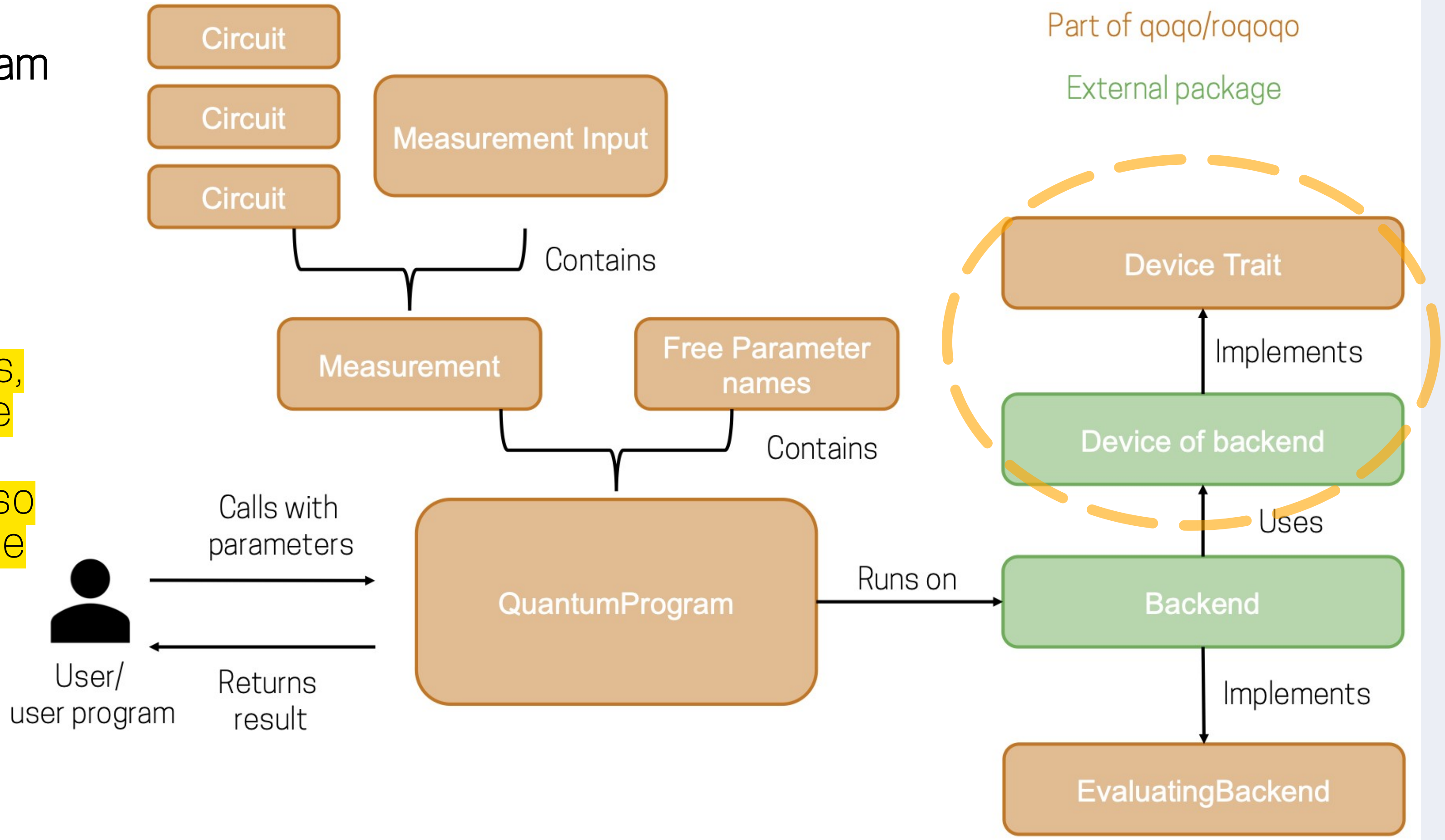
```
# b) To run a measurement we need to replace the free parameter by hand
executable_measurement = measurement.substitute_parameters({"angle": 0.2})
expectation_values = backend.run_measurement(executable_measurement)
```

  - A quantum program.

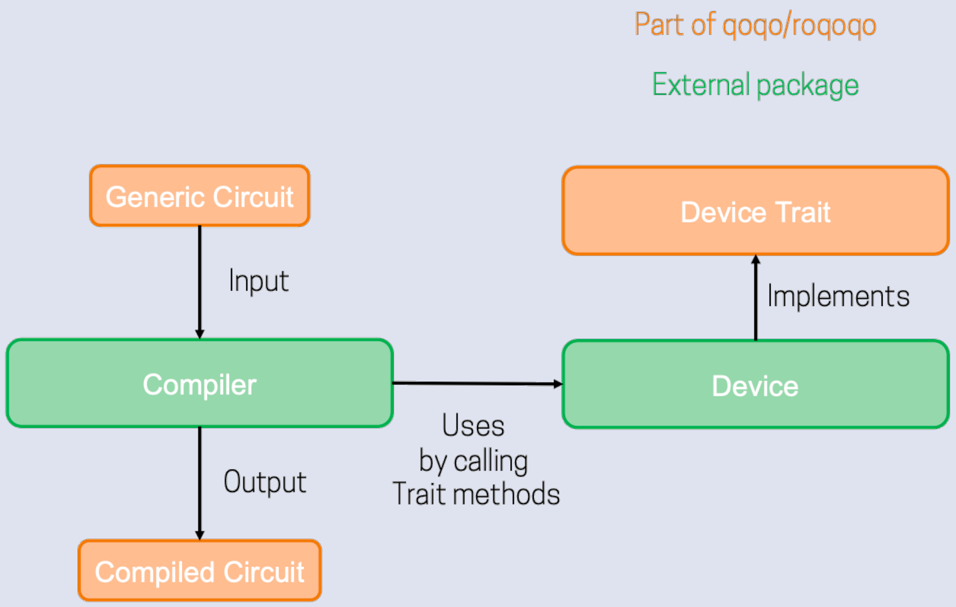
```
# c) Run a quantum program. It has one free parameter that must be set when executing it.
program = QuantumProgram(measurement = measurement,
                         input_parameter_names = ["angle"])
# Run the program with 0.1 substituting `angle`
expectation_values = program.run(backend, [0.1])
```

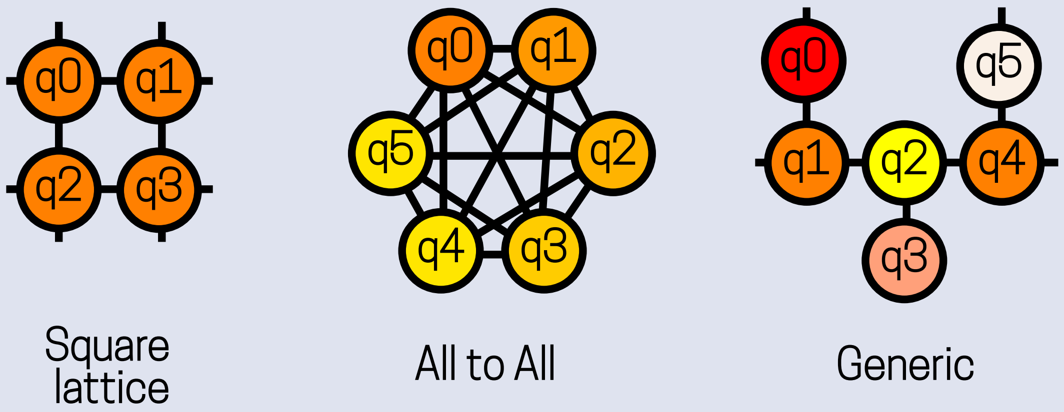
Architecture

- Operations and Circuit
- Measurements and QuantumProgram
- Backends
- Devices
  - When compiling quantum circuits, it is often necessary to know the topology of a target quantum device. Device properties can also be used by backends, for example to accurately simulate a given quantum device



Devices





Devices can be:

- **Abstract devices** contain abstract information about the device topology, the available gates and the noise model.
- **Backend devices** are devices that are implemented by a qoqo backend. They can specify additional information for accessing the device on the backend.

Device properties include:

- Topology
- Number of qubits
- Single-, two- and multi-qubit gates times
- Decoherence rates (for each qubit)
- Device's native gates

Hands on session!

HANDS ON: Qoqo`s devices

- Set Square-Lattice Device
- Set All-to-All Device (Exercise 1)
- Add damping, dephasing and depolarising noise to Device (Exercise 2)

Each exercise has:

```
problem.show_hint() #show a hint
problem.show_solution() #show the solution to the exercise
problem.check(user_solution) #check the variable containing the solution by the user
```

HQS QUANTUM SIMULATIONS

https://cloud.quantumsimulations.de

https://github.com/HQSquantumsimulations/qoqo

info@quantumsimulations.de

Log in on QLM machine

1) ssh -J <username>@qclogin.srv.lrz.de <username>@qlm.for.lrz.de

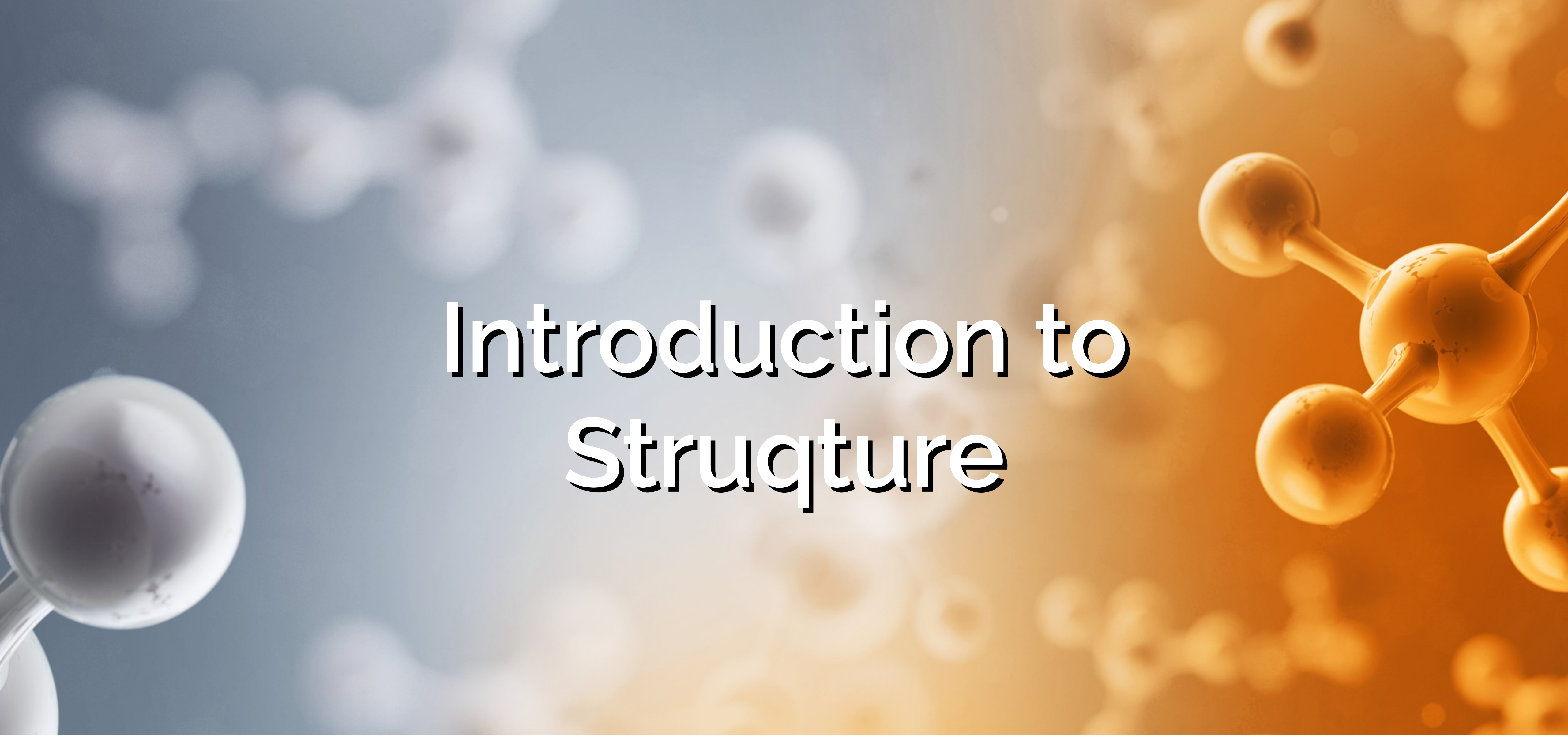
2)./launch_qlm_notebooks

    a)In the output take note of the port in: http://127.0.0.1:####

3) (from another terminal) ssh -J <username>@qclogin.srv.lrz.de -L <PortInOutput2>:localhost:<PortInOutput2> <username>@qlm.for.lrz.de

4) Ctrl+click on the link https://127… from step 2

5)Exercises are in home/LRZ_workshop_exercises/examples



Introduction to Struqture

HQS QUANTUM SIMULATIONS

Our Approach

Define problem

H_0

Device specification

qoqo

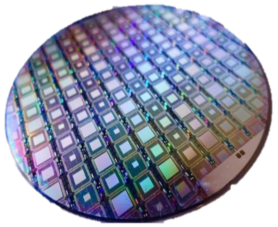
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

qoqo

Simulator or actual device



Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

Struqture: Overview

Struqture

Struqture is a library to represent

- quantum-mechanical operators
- Hamiltonians
- open-quantum systems

Open-source on the GitHub (https://github.com/HQSquantumsimulations)

- Rust (struqture)
- Python (struqture_py)

```
pip install struqture_py
```

The library supports building spin, fermion, bosonic and mixed objects.

Struqture

Struqture is a library to represent

- quantum-mechanical operators
- Hamiltonians
- open-quantum systems

Open-source on the GitHub (https://github.com/HQSquantumsimulations)

- Rust (struqture)
- Python (struqture_py)

```
pip install struqture_py
```

The library supports building spin, fermion, bosonic and mixed objects.

How we use struqture in the workshop:

- We define the Hamiltonian in struqture
- Nuclear magnetic resonance (NMR):

$$\hat{H} = -\sum_i \gamma_k B_0 \hat{\sigma}_i^z + \sum_{i<j} J_{ij}\, \hat{\boldsymbol{\sigma}}_i \cdot \hat{\boldsymbol{\sigma}}_j$$

- Spin-boson Hamiltonian (mixed system):

$$\hat{H} = \epsilon \hat{\sigma}_z + \sum_i \hat{b}_i^\dagger \hat{b}_i + \frac{1}{2}\hat{\sigma}_x \sum_i \left(\hat{b}_i^\dagger + \hat{b}_i\right)$$

Our Approach

Define problem

$$H_0$$

Device specification

qoqo

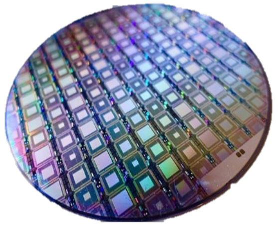
HQSNoiseApp

$$\mathcal{L}_{eff}$$

Gates and instructions

qoqo

Simulator or actual device



Classical solver

Struqture-QuTiP

Diagonalization

...

Spin, bosonic, mixed systems

Spin-systems

Spin-Hamiltonian (*SpinHamiltonianSystem*)

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \underbrace{\prod_k \sigma_j^k}_{\text{PauliProduct}}$$

α_j real coefficient

$\sigma^k \in X, Y, Z, I$ Pauli matrices and identity

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Creating a spin-Hamiltonian in struqture:

$$\hat{H} = 0.5\underbrace{\sigma_0^z \sigma_1^x}_{\text{PauliProduct}}$$

Spin-systems

Spin-Hamiltonian (*SpinHamiltonianSystem*)

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \underbrace{\prod_k \sigma_j^k}_{\text{PauliProduct}}$$

α_j real coefficient

$\sigma^k \in X, Y, Z, I$ Pauli matrices and identity

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Creating a spin-Hamiltonian in struqture:

$$\hat{H} = 0.5 \underbrace{\sigma_0^z \sigma_1^x}_{\text{PauliProduct}}$$

```
# Hamiltonian with 2 spins.
ham = SpinHamiltonianSystem(2)

# Coefficient
alpha0 = 0.5

# Pauli-product
pauli_product = PauliProduct().z(0).x(1)

# Add Pauli-product to Hamiltonian.
ham.add_operator_product(pauli_product, alpha0)
```

Tutorial 2.1 on spin-Hamiltonians

Boson-systems

Bosonic-Hamiltonian (*BosonHamiltonianSystem*)

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \prod_{k=0}^{N-1} f(j,k) \prod_{l=0}^{N-1} g(j,l)$$

α_j complex coefficient

$f(j,k) \in c_k^{j\,\dagger}, 1$ creation operator

$g(j,k) \in c_k^{j}, 1$ annihilation operator

Commutation relations $[c_i, c_j^\dagger] = \delta_{ij}$

$[c_i^\dagger, c_j^\dagger] = [c_i, c_j] = 0$

Normal ordering

Boson-systems

Bosonic-Hamiltonian (*BosonHamiltonianSystem*)

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \prod_{k=0}^{N-1} f(j,k) \prod_{l=0}^{N-1} g(j,l)$$

α_j complex coefficient

$f(j,k) \in c_k^{i^\dagger}, 1$ creation operator

$g(j,k) \in c_k^i, 1$ annihilation operator

Commutation relations

$$[c_i, c_j^\dagger] = \delta_{ij}$$
$$[c_i^\dagger, c_j^\dagger] = [c_i, c_j] = 0$$

Normal ordering

$$H = 2.0\, c_0^\dagger c_1 c_2 + 2.0\, c_2^\dagger c_1^\dagger c_0$$

In struqture the hermitian conjugate term is stored internally.

Choice which of the terms is stored: Smallest index of creation operators smaller or equal to the smallest index in the annihilation operator.

```
# Boson Hamiltonian
ham = BosonHamiltonianSystem(3)

# Add Pauli-product to Hamiltonian.
hbp = HermitianBosonProduct([0], [1,2])

# Add operator product
ham.add_operator_product(hbp, 2.0)
```

Mixed-Hamiltonians

Any system with more than one type is called a *MixedHamiltonianSystem* in struqture. Mixed-system also include having several subsystems of one kind. For example to two spin-subsystems.

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \underbrace{\prod_{k} \sigma_j^k}_{\text{spins}} \underbrace{\overbrace{\prod_{k=0}^{N-1} f(j,k)}^{\text{creation}} \overbrace{\prod_{l=0}^{N-1} g(j,l)}^{\text{annihilation}}}_{\text{bosons}}$$

- Spin-boson Hamiltonian
- System with several subsystems of same type

Two examples

Spin-boson Hamiltonian with 1 spin and 4 bosons.

$$H = \epsilon\sigma_z + \sum_{i=0}^{3} b_i^\dagger b_i + \frac{1}{2}\sigma_x \sum_{i=0}^{3} \left(b_i^\dagger + b_i\right)$$

```
# Mixed Hamiltonian with 1 spin and 4 bosons.
spin_boson_hamiltonian = MixedHamiltonianSystem([1], [4], [ ])

# Adding terms to spin_boson_hamiltonian...
```

Two examples

Spin-boson Hamiltonian with 1 spin and 4 bosons.

$$H = \epsilon\sigma_z + \sum_{i=0}^{3} b_i^\dagger b_i + \frac{1}{2}\sigma_x \sum_{i=0}^{3}\left(b_i^\dagger + b_i\right)$$

```
# Mixed Hamiltonian with 1 spin and 4 bosons.
spin_boson_hamiltonian = MixedHamiltonianSystem([1], [4], [ ])

# Adding terms to spin_boson_hamiltonian...
```

Spin-spin Hamiltonian with two spin-subsystem (s and b)

$$H = \epsilon\sigma_z^s + \sum_{i=0}^{2} \sigma_z^{b_i} + \frac{1}{2}\sigma_x^s \sum_{i=0}^{2} \sigma_x^{b_i}$$

```
# Mixed Hamiltonian with 2 spin-subsystems.
spin_spin_hamiltonian = MixedHamiltonianSystem([1, 3], [ ], [ ])

# Adding terms to spin_spin_hamiltonian...
```

Open-quantum systems

Open-quantum systems

The Lindblad equation is a master equation determining the time-evolution of the density matrix.

rate matrix

left and right Lindblad operators

$$\dot{\rho} = \mathcal{L}(\rho) = \underbrace{\underbrace{-i[\hat{H}, \rho]}_{\textit{SpinHamiltonianSystem}} + \underbrace{\sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)}_{\textit{SpinLindbladNoiseSystem}}}_{\textit{SpinLindbladOpenSystem}}$$

Open-quantum systems

The Lindblad equation is a master equation determining the time-evolution of the density matrix.

rate matrix

left and right Lindblad operators

$$\underbrace{\underbrace{\dot{\rho} = \mathcal{L}(\rho) = -i[\hat{H}, \rho]}_{\textit{SpinHamiltonianSystem}} + \underbrace{\sum_{j,k} M_{j,k} \left(A_j \rho A_k^{\dagger} - \frac{1}{2}\{A_k^{\dagger} A_j, \rho\} \right)}_{\textit{SpinLindbladNoiseSystem}}}_{\textit{SpinLindbladOpenSystem}}$$

```
# Hamiltonian with 1 spin.
noise_system = SpinLindbladNoiseSystem(1)

# Create decoherence product (Lindblad operators) for dephasing.
dp = DecoherenceProduct().z(0)

# Add to noise system.
noise_system.add_operator_product((dp, dp), 0.1)

# Initialise Hamiltonian ...

# Group Hamiltonian and noise-system.
open_system = SpinLindbladOpenSystem(1)
open_system = open_system.group(spin_hamiltonian, noise_system)
```

Open-quantum systems

The Lindblad equation is a master equation determining the time-evolution of the density matrix.

rate matrix

left and right Lindblad operators

$$\dot{\rho} = \mathcal{L}(\rho) = \underbrace{\underbrace{-i[\hat{H}, \rho]}_{\textit{SpinHamiltonianSystem}} + \underbrace{\sum_{j,k} M_{j,k} \left(A_j \rho A_k^{\dagger} - \frac{1}{2}\{A_k^{\dagger} A_j, \rho\} \right)}_{\textit{SpinLindbladNoiseSystem}}}_{\textit{SpinLindbladOpenSystem}}$$

```
# Hamiltonian with 1 spin.
noise_system = SpinLindbladNoiseSystem(1)

# Create decoherence product (Lindblad operators) for dephasing.
dp = DecoherenceProduct().z(0)

# Add to noise system.
noise_system.add_operator_product((dp, dp), 0.1)

# Initialise Hamiltonian ...

# Group Hamiltonian and noise-system.
open_system = SpinLindbladOpenSystem(1)
open_system = open_system.group(spin_hamiltonian, noise_system)
```

Open-quantum systems

The Lindblad equation is a master equation determining the time-evolution of the density matrix.

rate matrix

left and right Lindblad operators

$$\dot{\rho} = \mathcal{L}(\rho) = \underbrace{\underbrace{-i[\hat{H}, \rho]}_{\textit{SpinHamiltonianSystem}} + \underbrace{\sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)}_{\textit{SpinLindbladNoiseSystem}}}_{\textit{SpinLindbladOpenSystem}}$$

```
# Hamiltonian with 1 spin.
noise_system = SpinLindbladNoiseSystem(1)

# Create decoherence product (Lindblad operators) for dephasing.
dp = DecoherenceProduct().z(0)

# Add to noise system.
noise_system.add_operator_product((dp, dp), 0.1)

# Initialise Hamiltonian ...

# Group Hamiltonian and noise-system.
open_system = SpinLindbladOpenSystem(1)
open_system = open_system.group(spin_hamiltonian, noise_system)
```

Tutorial 2.2 on noise systems

Design of struqture

Systems

- *SpinHamiltonianSystem*
- *BosonHamiltonianSystem*

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \prod_k \sigma_j^k \qquad \alpha_j \text{ real}$$

$$\hat{H} = \sum_{j=0}^{N-1} \alpha_j \left(\prod_{k=0}^{N-1} f(j,k) \right) \left(\prod_{l=0}^{N-1} g(j,l) \right)$$

always hermitian operator

$$\hat{H} = \sigma_0^x + \cancel{i\sigma_0^z}$$

$$\hat{H} = \frac{1}{2} c_0 a_0 + h.c. + \cancel{\frac{i}{4} a_1}$$

Operators

- *SpinSystem*
- *BosonSystem*

Spins $\hat{O} = \sum_j \alpha_j \prod_k \sigma_j^k \quad \alpha_j \text{ complex}$

Bosons $\hat{O} = \sum_j \alpha_j \left(\prod_k f(j,k) \right) \left(\prod_l g(j,l) \right)$

Example $\hat{O} = \sigma_0^x + i\sigma_0^z$

$$\hat{O} = \frac{1}{2} c_0 a_0 + \frac{i}{4} a_1$$

Physical types

Struqture is designed to construct objects across all physical type.

- Spins
- Bosons
- Fermions
- Mixed-systems

Container types

Container types are common to all physical types (stored as dictionaries).

- Indices
  - *PauliProduct*
  - *HermitianBosonProduct*
  - *DecoherenceProduct*
- Operators
  - *SpinSystem*
  - *BosonSystem*
- Hamiltonian systems
  - *SpinHamiltonianSystem*
  - *BosonHamiltonianSystem*
  - *MixedHamiltonianSystem*
- Noise systems
- Open systems

Hands-on!

Overview of notebooks

5 notebooks with examples in struqture

- 2_1_struqture_spins.ipynb
- 2_2_struqture_spins_noise.ipynb
- 2_3_struqture_bosons.ipynb
- 2_4_struqture_bosons_noise.ipynb
- 2_5_struqture_mixed_systems.ipynb

Hands-on

Additional material: No hands-ons. You can go through if you have time.

The goals of the tutorials are that the user is comfortable with setting up Hamiltonians and open-system model for spins.

HQS
QUANTUM SIMULATIONS

Login on the QLM machine

1) ssh -J <username>@qclogin.srv.lrz.de <username>@qlm.for.lrz.de
2) ./launch_qlm_notebooks
   a) In the output take note of the port in: http://127.0.0.1:####
3) (from another terminal) ssh -J <username>@qclogin.srv.lrz.de -L <PortInOutput2>:localhost:<PortInOutput2> <username>@qlm.for.lrz.de
4) Ctrl+click on the link https://172... from step 2

Trotter evolution

QUANTUM SIMULATIONS

Our Approach

Define problem

H_0

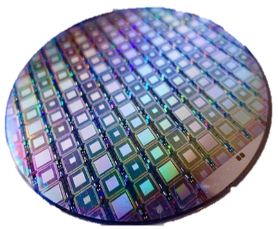
Device specification

HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

Simulator or actual device



Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

The HqsNoiseApp has dependencies on 3 HQS libraries:

- Qoqo: The HQS quantum computing toolkit.
  - Device and Backend: to run the algorithm
- Struqture: An HQS library to represent quantum Operator, Hamiltonian and Lindblad open systems. Supports spin, Fermionic, Bosonic and Mixed systems.
- Qoqo calculator: A simple symbolic value library for qoqo and struqture that allows for mathematical operations.

Our Approach

Define problem

H_0

Device specification

qoqo BY HQS

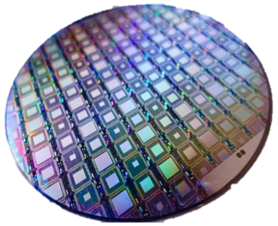
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

qoqo BY HQS

Simulator or actual device

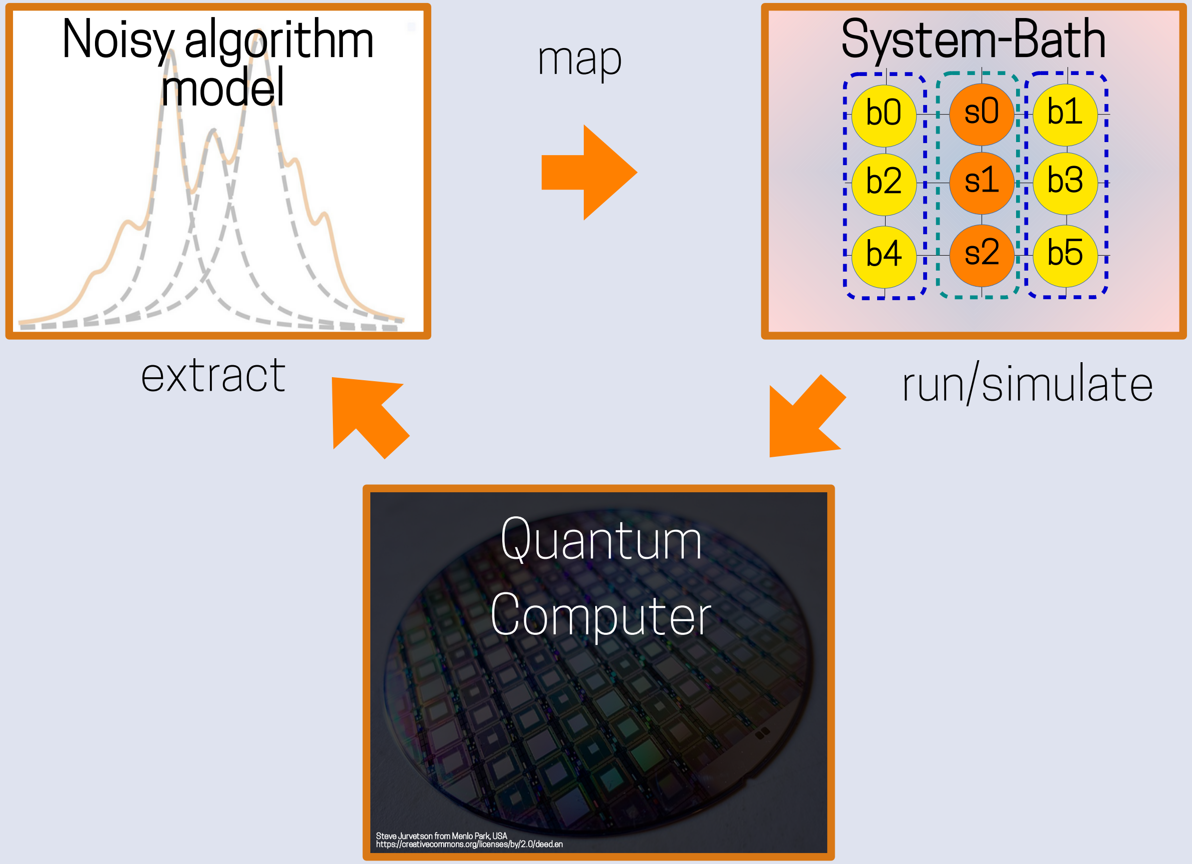


Steve Jurvetson from Menlo Park, USA https://creativecommons.org/licenses/by/2.0/deed.en

Device specification including noise and output including the **noisy algorithm model.**

HQS Noise App

HqsNoiseApp



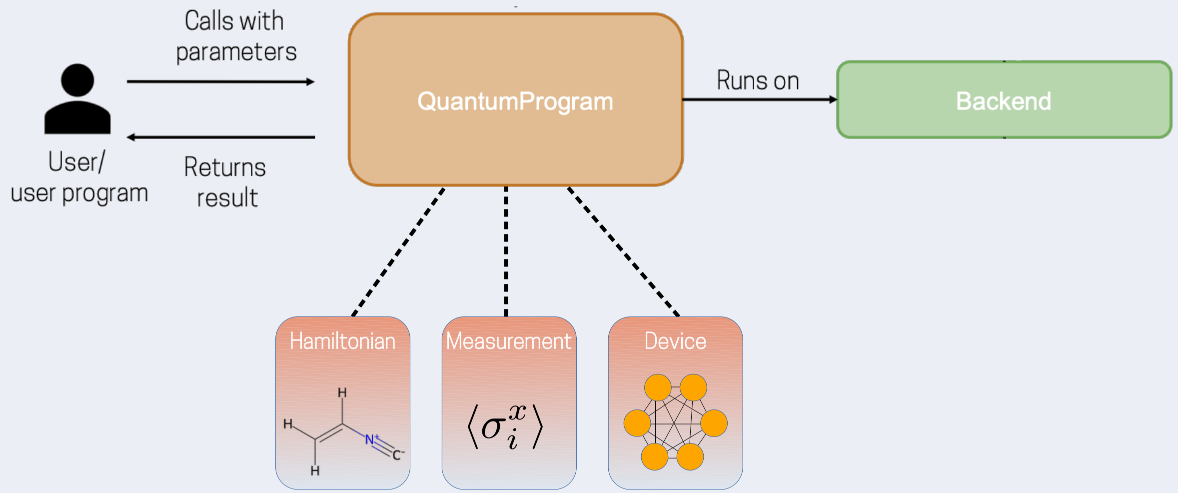
The HqsNoiseApp has *two* main purposes:

- To extract the effective **noisy algorithm model** for a coherent time propagation run on a noisy quantum computer
- To use the noisy algorithm model to create optimized **system-bath quantum circuits** to simulate a open quantum system of interest with the help of the physical noise on a quantum computer.

HqsNoiseApp

This session will focus on:

Creating a QuantumProgram for time propagating a state under a spin Hamiltonian and measuring a collection of Spin operators



Time evolution on a quantum device

Time evolution

Unitary time-evolution

Quantum simulations aim to predict the behavior of quantum systems over time.

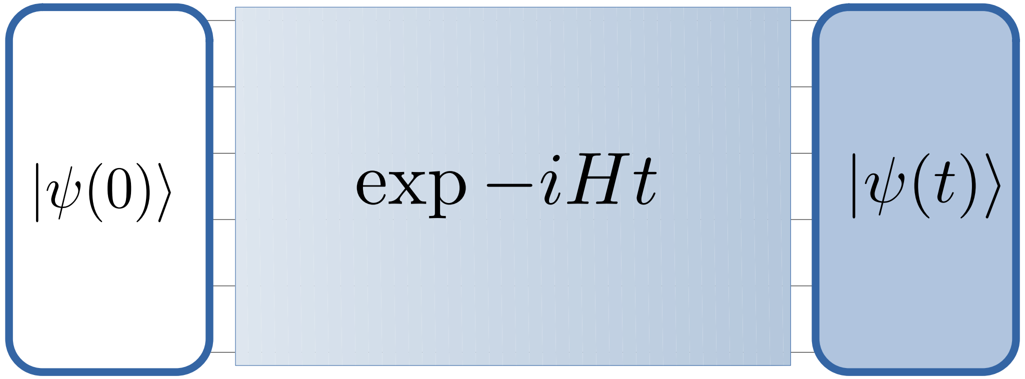
The Schrödinger equation dictates how quantum states evolve:

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$$

Directly simulating an entire Hamiltonian with quantum gates is challenging because Hamiltonians typically involve non-commuting terms!

Nonetheless, in many quantum systems, the Hamiltonian can be expressed as a sum of local interactions:

$$H = \sum_j H_j$$



Trotter evolution

SOLUTION: Trotter-Suzuki expansion

It allows us to rewrite exponential operators as:

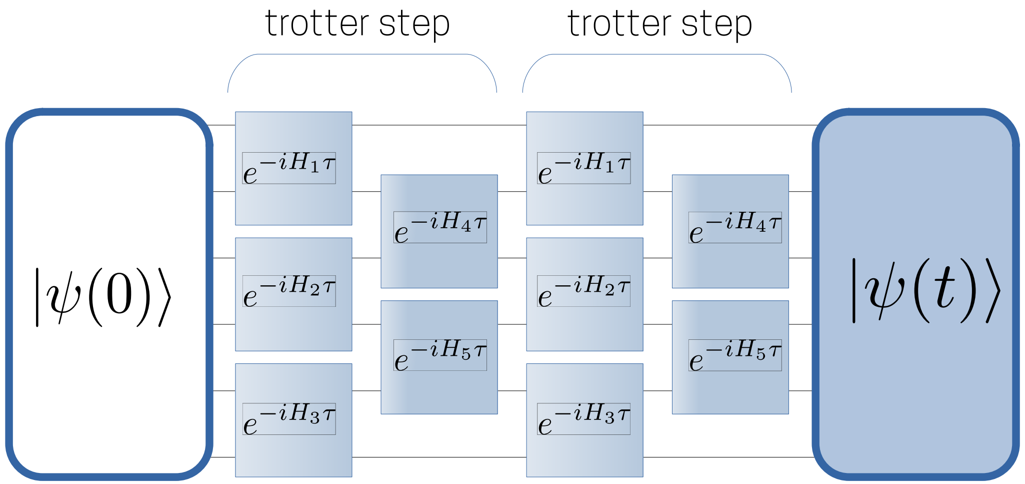
$$e^{(A+B)} = \lim_{n\to\infty} (e^{\frac{A}{n}} e^{\frac{B}{n}})^n$$

which applied to our Hamiltonian becomes:

$$\exp^{-iHt} = [\exp^{-iHt/m}]^m \equiv [\exp^{-iH\tau}]^m$$
$$\approx [\prod_j \exp^{-iH_j\tau}]^m$$

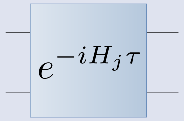
$$\tau = t/m$$

An approximation (error) occurs when individual unitaries do not commute. But is under control for small increment of virtual time

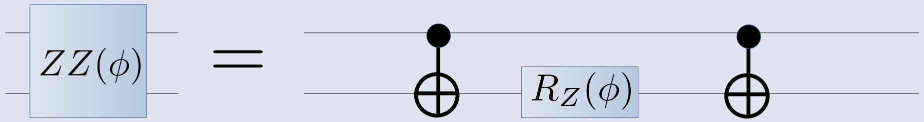


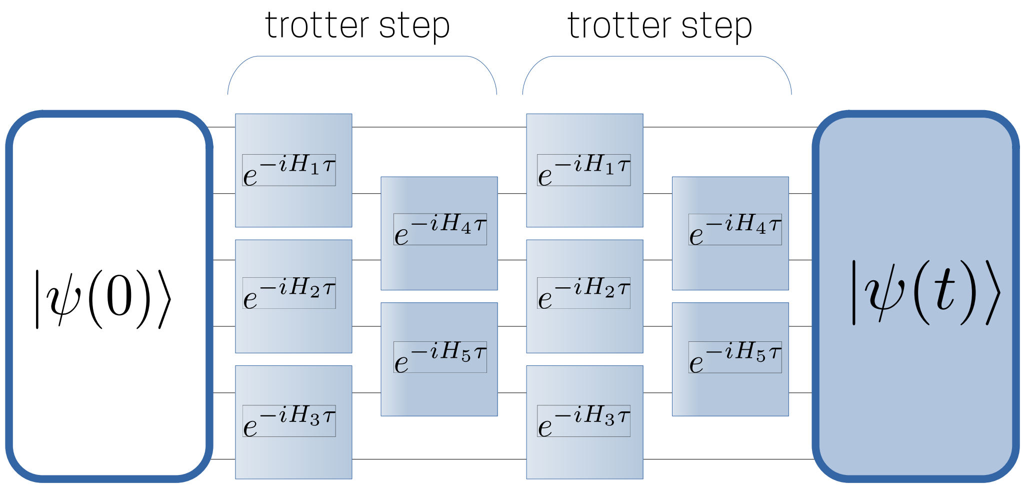
Trotter evolution

The goal of the division is to have a sequence of “small-angle” unitary transformations which can be implemented efficiently on hardware (e.g with Pauli gates).



In case the 2 qubits gate is not natively present on the quantum device, this can be decomposed in a sequence of 3 basic gates, e.g.:

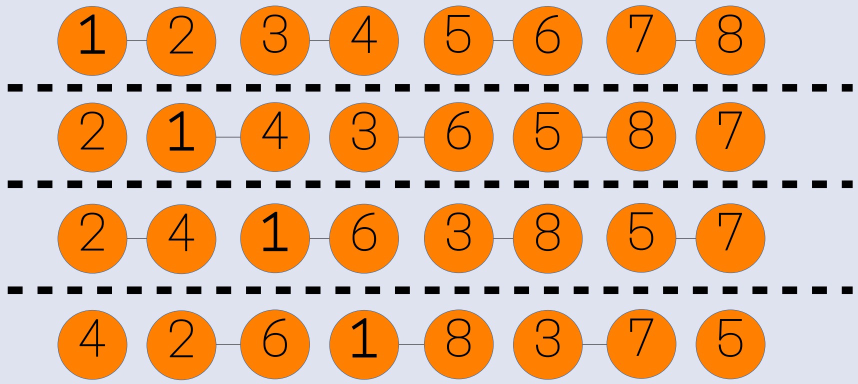


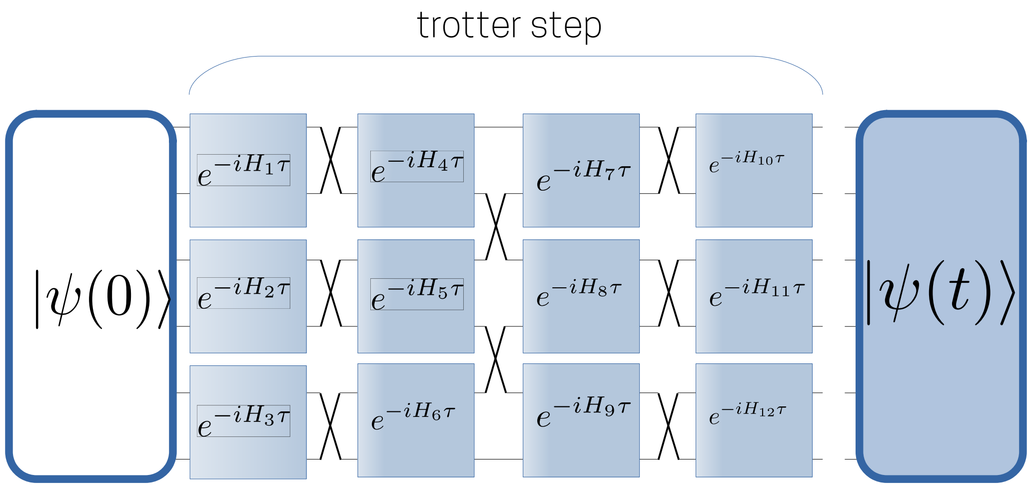


Trotter evolution

Sometimes the Hamiltonian might have a non-trivial nearest neighboor interaction, for which spins “far apart” are coupled and needs to interact.

This can be efficiently handled by the **SWAP algorithm**.





Trotter evolution with the Noise App

Trotter evolution with Noise App

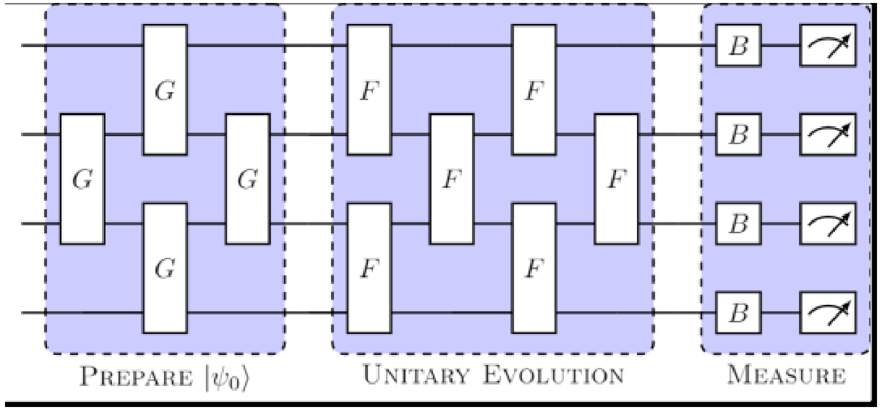
The trotterized time-evolution consist then of the following steps:

1) Preparation of the initial state

2) Trotterized time-evolution

   a) run the 1-step trotterized circuit N time

3) Measurement



Trotter evolution with Noise App

In the HQS Noise App we approximate an entire trotter evolution using a **Quantum Program** object, which contains:

Initialisation
Hamiltonian
Measured Operators
(Quantum) Device

setup

```
quantum_program = noise_app.quantum_program(
    system_hamiltonian,
    trotter_timestep=trotter_timestep,
    initialisation=initialisation,
    measured_operators=operators,
    operator_names=operator_names,
    device=device,
)
```

execution

```
for i in range(number_trottersteps):
    simulation = noise_app.simulate_quantum_program(
        quantum_program, i, device, number_spins
    )
```

Hands on session!

HANDS ON: Time-evolution on perfect device

- Import Hamiltonian (C2H3NC)
- Set Device
- Set Initialisation
- Set Measurement (Exercise 1)
- Set HQSNoiseApp
- Create Quantum Program
- Plot circuit
- Execute the quantum program with Quest (Exercise 2)
- Fourier Transform the results
- Execute the quantum program with QLM
- Fourier Transform the results

Noisy Algorithm Model: Intro

Our Approach

Define problem

H_0

Device specification

qoqo BY HQS

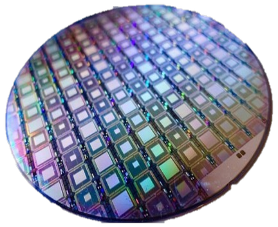
HQSNoiseApp

$\mathcal{L}_{eff}$

Gates and instructions

qoqo BY HQS

Simulator or actual device



Steve Jurvetson from Menlo Park, USA
https://creativecommons.org/licenses/by/2.0/deed.en

The HqsNoiseApp has dependencies on 3 HQS libraries:

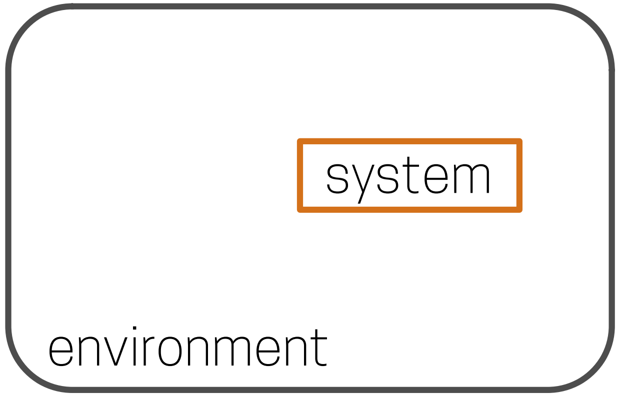
- Qoqo: The HQS quantum computing toolkit.
  - Device and Backend: to run the algorithm
- Struqture: An HQS library to represent quantum Operator, Hamiltonian and Lindblad open systems. Supports spin, Fermionic, Bosonic and Mixed systems.
- Qoqo calculator: A simple symbolic value library for qoqo and struqture that allows for mathematical operations.

Decoherence theory

Decoherence theory

Close system: $\boxed{H = H_S}$

- Schrödinger equation $|\psi(t)\rangle = e^{-iH_S t}|\psi(0)\rangle$



Open quantum system: $\boxed{H = H_S + H_C + H_E}$

- Lindblad-master equation

rate matrix

left and right Lindblad operators

$$\dot{\rho} = \mathcal{L}(\rho) = -i[\hat{H}, \rho] + \sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)$$

- Properties

$$\rho = \rho^\dagger, \ \mathrm{tr}(\rho) = 1, \text{ and } \rho > 0$$

Damping and dephasing

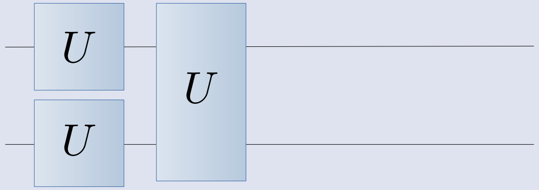
$$\dot{\rho} = \mathcal{L}(\rho) = -i[\hat{H}, \rho] + \sum_{j,k} M_{j,k} \left(A_j \rho A_k^\dagger - \frac{1}{2}\{A_k^\dagger A_j, \rho\} \right)$$

- Dephasing
  - Lindblad operators $A_j = A_k = \sigma_z$
  - Elements of rate matrix $M_{\sigma_z,\sigma_z} = \gamma_{\text{dephasing}}$
- Damping
  - Lindblad operators $A_j = A_k = \sigma^+ = (\sigma^x + \mathrm{i}\sigma^y)/2$
  - Elements of rate matrix $M_{\sigma_x,\sigma_x} = M_{\sigma_x,\mathrm{i}\sigma_y} = M_{\mathrm{i}\sigma_y,\sigma_x} = M_{\mathrm{i}\sigma_y,\mathrm{i}\sigma_y} = \gamma_{\text{damping}}/4$

Noisy-algorithm model: Intro

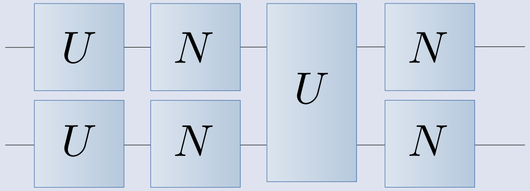
The noisy algorithm model

- In a single Trotter step, the state evolves with



$$\boxed{\rho(t) = e^{\mathcal{L}t}\rho_0}$$

- The time-evolution in a noisy quantum circuits deviates from the noisy-free time-evolution.

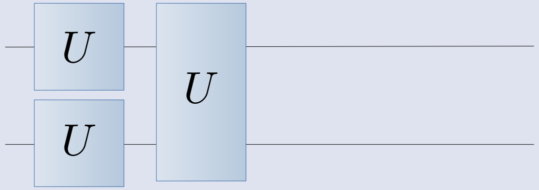


$$\boxed{\rho(t) = e^{\mathcal{L}_N t_{\text{gate}}} e^{\mathcal{L}t}\rho_0 \rightarrow e^{\mathcal{L}_{\text{eff}} t}\rho_0}$$

Physical noise

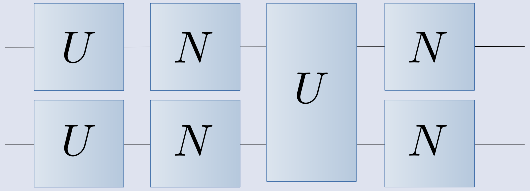
The noisy algorithm model

- In a single Trotter step, the state evolves with



$$\rho(t) = e^{\mathcal{L}t}\rho_0$$

- The time-evolution in a noisy quantum circuits deviates from the noisy-free time-evolution.



$$\rho(t) = e^{\mathcal{L}_N t_{\text{gate}}} e^{\mathcal{L}t}\rho_0 \to e^{\mathcal{L}_{\text{eff}} t}\rho_0$$

Physical noise

Noisy-algorithm model describes time-propagation of state under Lindblad equation

Effective Lindblad equation

$$\mathcal{L}_{\text{eff}}(\rho) = -i[H, \rho] + \sum_k \mathcal{L}_{\text{eff}}^k$$

$$\rho(t) = e^{\mathcal{L}_{\text{eff}} t}\rho_0$$

Hands-on!
Noisy-algorithm model: basic example

Noisy-algorithm model from HQSNoiseApp

```
# Setup Hamiltonian
number_spins = 2
hamiltonian = spinHamiltonianSystem(number_spins)
...

# Setup device
noisy_device = devices.AllToAllDevice(..).add_damping_all(0.001)

# Initialise the HQSNoiseApp.
noise_app = HqsNoiseApp(noise_mode).algorithm(algorithm)

# Get noisy-algorithm model
noisy_algorithm_model = noise_app.noisy_algorithm_model(hamiltonian, trotter_timestep, noisy_device)
```

Noisy-algorithm model: Basic example

The goal of the tutorial is to compare the time-evolution of the noisy-algorithm model with the execution of the circuit.

1) Simulate circuit with QuEST
2) Time-evolve state with noisy-algorithm model (QuTiP)

$$\rho(t) = e^{\mathcal{L}_{\text{eff}} t}\rho_0 \qquad \mathcal{L}_{\text{eff}}(\rho) = -i[H,\rho] + \sum_k \mathcal{L}^k_{\text{eff}}\,.$$

3) What are the dominant noise-terms in the noisy-algorithm model

Left and right Lindblad operators

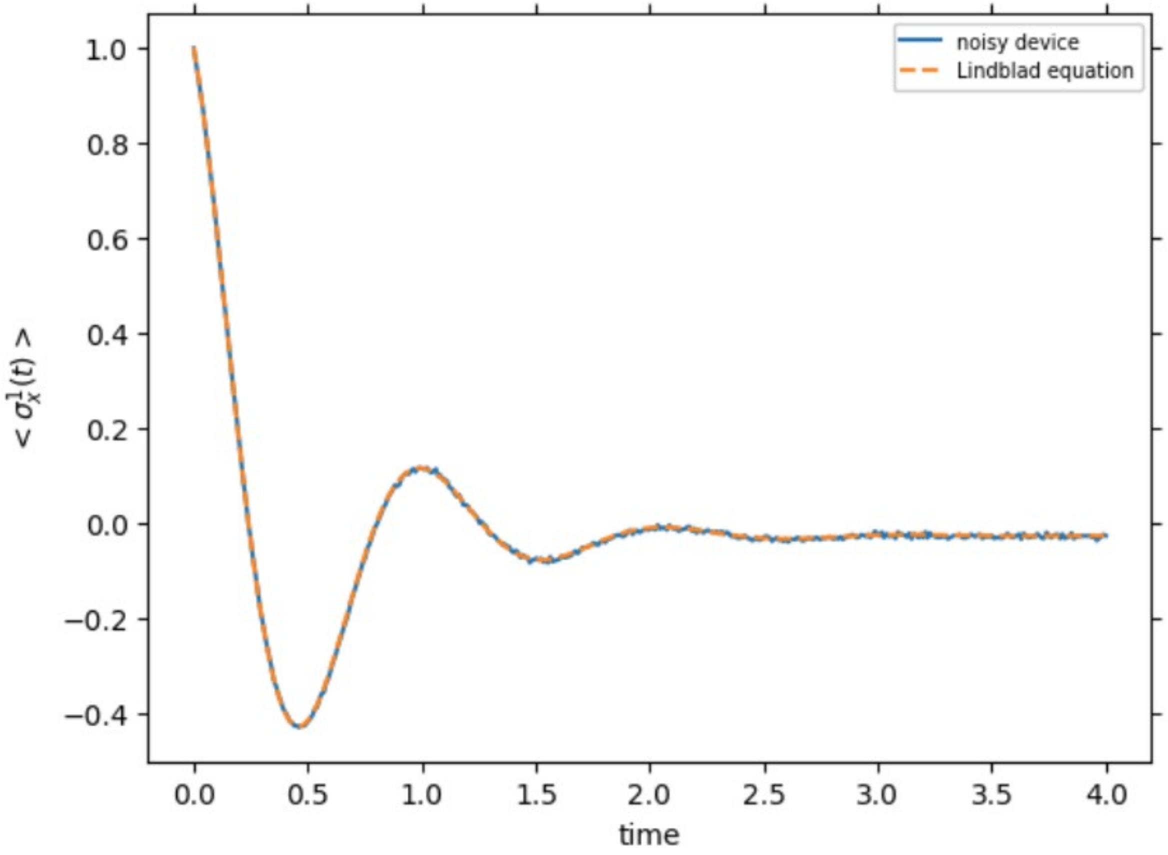
```
SpinLindbladNoiseSystem(3){
(2iY, 2X): (7.999999994434254e-1 + i * 0e0),
(2X, 2X): (7.999999992579003e-1 + i * 0e0),
(1X, 1X): (7.499999970117005e-1 + i * 0e0),
(0X, 0X): (6.999999981729255e-1 + i * 0e0),
(2iY, 2iY): (8.999999994434255e-1 + i * 0e0),
```

Change basis from $(\sigma_x, \sigma_y, \sigma_z)$ to $(\sigma_+, \sigma_-, \sigma_z)$

Summary

Summary

Good agreement between execution of circuit and noisy-algorithm model.



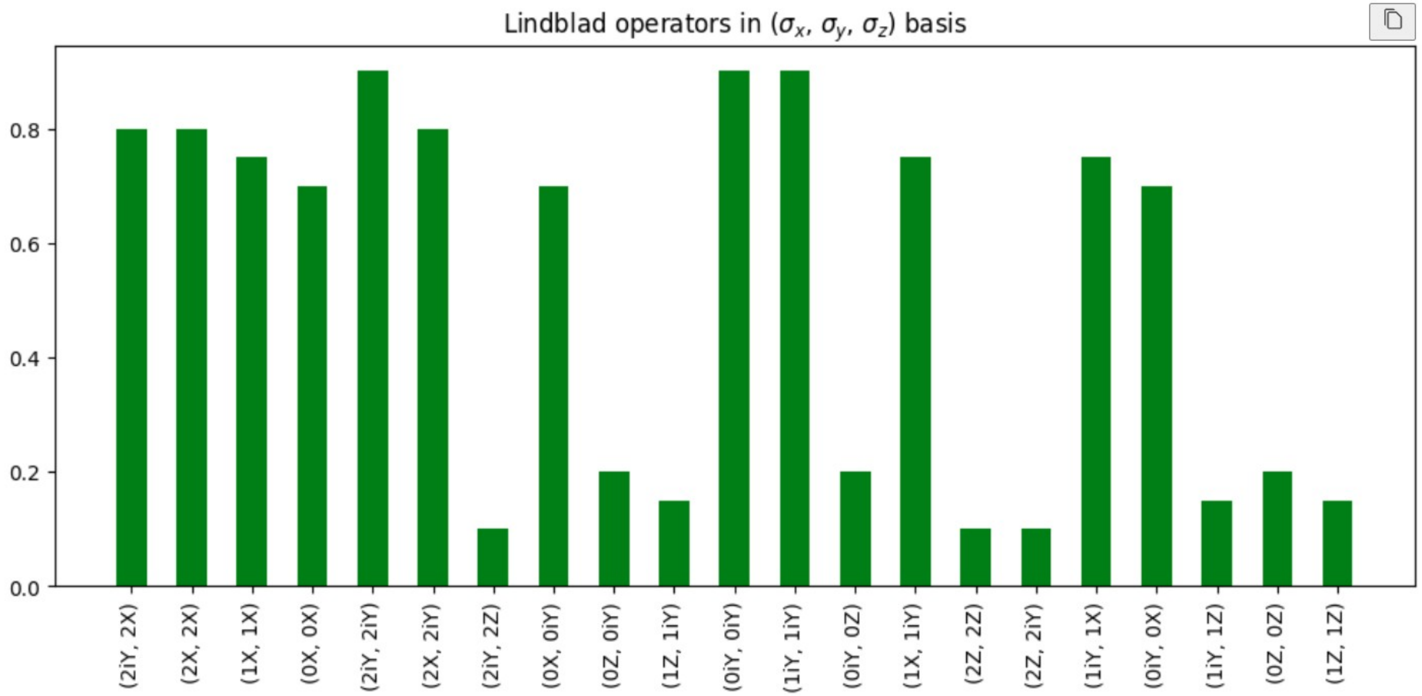
The noisy-algorithm describes very exact how noise appears while running a circuit.

- We set only damping in the device.

- Damping is described by

$$A_j = A_k = \sigma^- = \frac{1}{2}\left(\sigma^x + \mathrm{i}\sigma^y\right)$$

- Rotate basis to $(\sigma_+, \sigma_-, \sigma_z)$



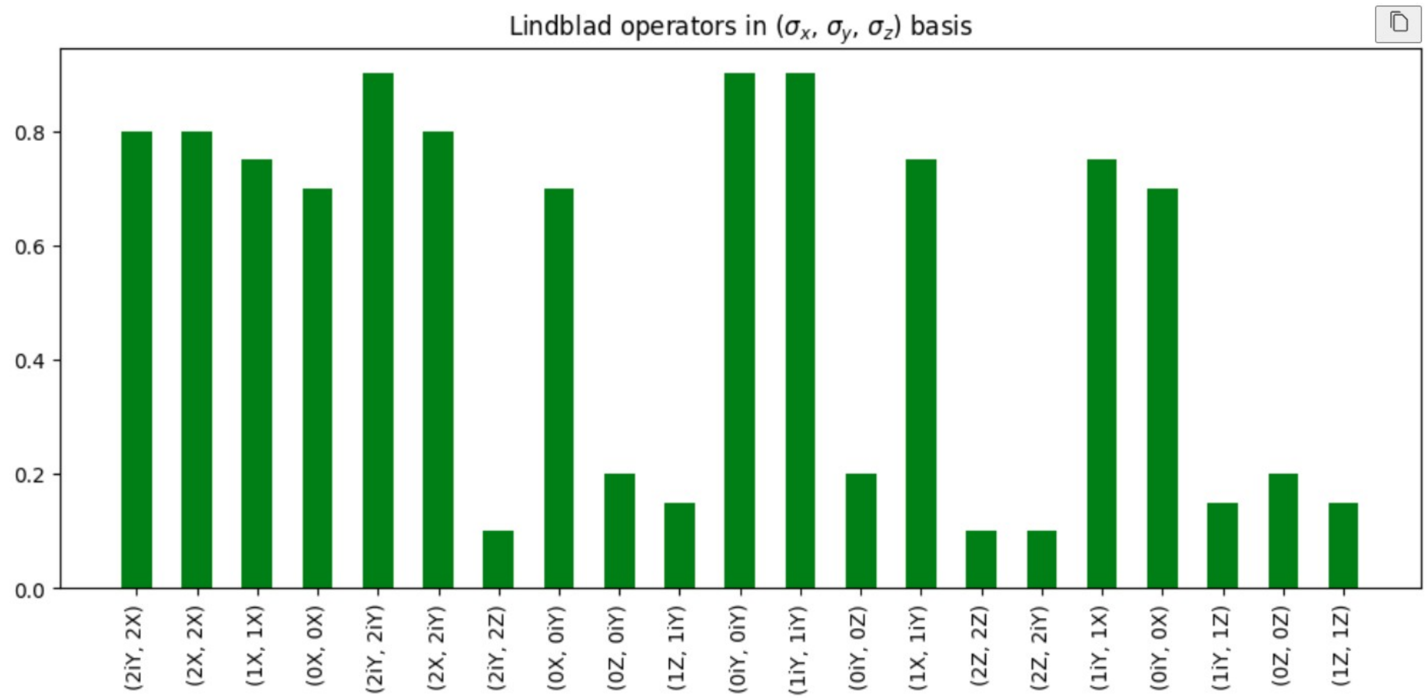
- We set only damping in the device.
- Damping is described by

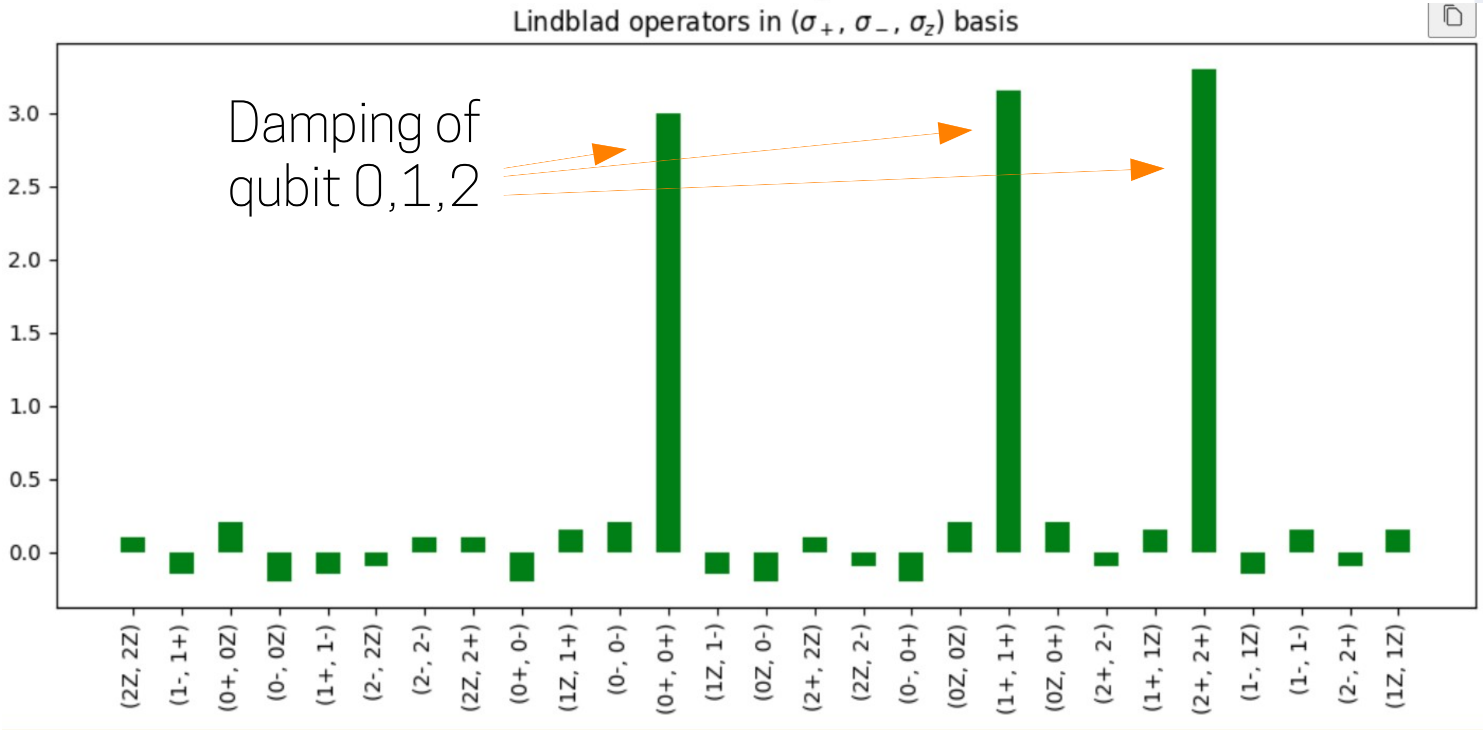
$$A_j = A_k = \sigma^+ = \frac{1}{2}\left(\sigma^x + \mathrm{i}\sigma^y\right)$$

- Rotate basis to $(\sigma_+, \sigma_-, \sigma_z)$

- Main noise is still damping

- But: there are others noise terms during the execution of the circuit





HQS QUANTUM SIMULATIONS

Thank you for your attention!

info@quantumsimulations.de
www.quantumsimulations.de

Noisy Algorithm Model: Advanced

Our Approach

Define problem

$$H_0$$

Device specification

qoqo BY HQS

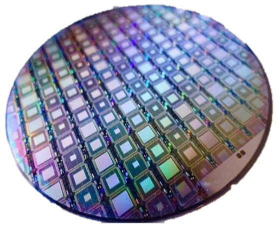
HQSNoiseApp

$$\mathcal{L}_{eff}$$

Gates and instructions

qoqo BY HQS

Simulator or actual device

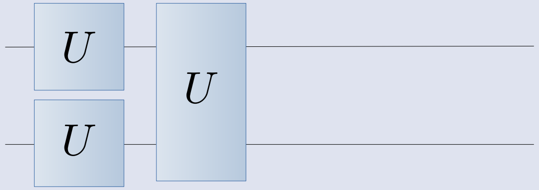


Device specification including noise and output including the **noisy algorithm model.**

Lets now analyze the noisy algorithm model a little more.

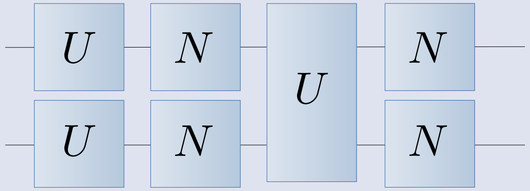
The noisy algorithm model

- In a single Trotter step, the state evolves with



$$\rho(t) = e^{\mathcal{L}t}\rho_0$$

- The time-evolution in a noisy quantum circuits deviates from the noisy-free time-evolution.



$$\rho(t) = e^{\mathcal{L}_N t_{\text{gate}}} e^{\mathcal{L}t}\rho_0 \rightarrow e^{\mathcal{L}_{\text{eff}}t}\rho_0$$

Physical noise

Noisy-algorithm model describes time-propagation of state under Lindblad equation

Effective Lindblad equation

$$\mathcal{L}_{\text{eff}}(\rho) = -i[H,\rho] + \sum_k \mathcal{L}_{\text{eff}}^k$$

$$\rho(t) = e^{\mathcal{L}_{\text{eff}}t}\rho_0$$

- Distinguish between
  - Native gates
  - Non-native gates

Describing Trotterized Time Evolution on Noisy Quantum Computers via Static Effective Lindbladians, arXiv:2210.11371

Circuits with native gates

- Trotterized time-evolution

$$\exp(-iHt) = \prod_{j=1}^{M} \exp(-iH\tau) \qquad \tau = t/M$$
$$= \prod_{j=1}^{M} \exp(-i\sum_k H_k\tau)$$
$$\approx \prod_{j=1}^{M}\prod_{k=1}^{N} \exp(-iH_k\tau)$$

- Add noise to partial Hamiltonian

$$\exp(-iH_k\tau) \rightarrow \exp(\mathcal{L}_N^k t_{\text{gate}})\exp(-iH_k\tau)$$
$$= \exp(\mathcal{L}_{\text{eff}}^k \tau)\exp(-iH_k\tau)$$

- Rescale noise rates $\quad M_{i,j}^{\text{eff}} = M_{i,j}\dfrac{t_{\text{gate}}}{\tau}$

Native gates:

- No fundamental change in noise behaviour
- Contribution to effective noise depends on ratio of gate time to simulate Trotter timestep.

Non-native gates

- Noisy algorithm model for a partial Hamiltonian

$$\exp(-iH_k\tau) \to \exp(\mathcal{L}_{\text{eff}}^k\tau)\exp(-iH_k\tau)$$

- Commuting noise

$$\begin{aligned} e^{-iH_k\tau} &= U_{0,k}U_{1,k}\dots \\ &\to \exp(\mathcal{L}_N^{k,0}t_{\text{gate}})U_{0,k}\exp(\mathcal{L}_N^{k,1}t_{\text{gate}})U_{1,k} \end{aligned}$$

$$U_{0,k}\exp(\mathcal{L}_N^{k,1}t_{\text{gate}}) = \exp(\mathcal{L}_{\text{eff}}^{1,k}\tau)U_{0,k}$$

- Modification of noise operators

$$A_{\text{eff}}^{1,k} = U_{0,k}AU_{0,k}^\dagger \qquad M_{i,j}^{\text{eff}} = M_{i,j}\frac{t_{\text{gate}}}{\tau}$$

Large angle gates:

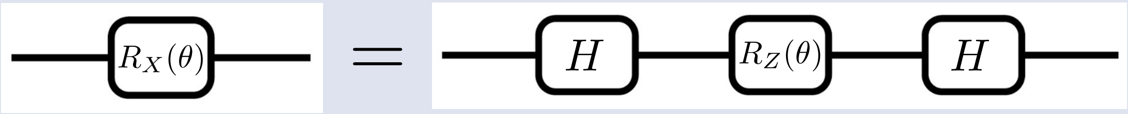
- Noise terms are transformed by unitary gates
- Noise rates are rescaled
- Qualitatively different behaviour

Example: Commuting Noise

- Partial Hamiltonian and damping

$$\mathrm{H}_k = \sigma_x \qquad\qquad A = \sigma^-$$

- Decomposition



- Add noise operator after every gate

$$U_0 U_1 U_2 \rightarrow e^{\mathcal{L}_N^{k,0} t_{\text{gate}}} U_{k,0} e^{\mathcal{L}_N^{k,1} t_{\text{gate}}} U_{k,1} e^{\mathcal{L}_N^{k,2} t_{\text{gate}}} U_{k,2}$$

$$A_{\text{eff}}^{1,k} = U_{0,k} A U_{0,k}^{\dagger} = \frac{1}{2}\sigma_z - \frac{i}{2}\sigma_y$$

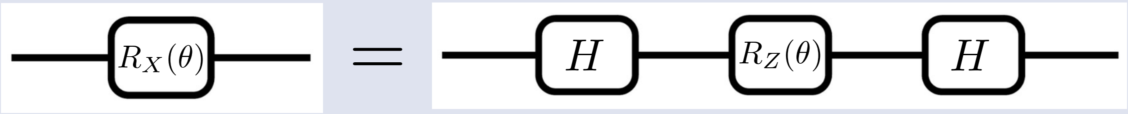
- Change from pure damping to dephasing, damping and excitation

Example: Commuting Noise

- Partial Hamiltonian and damping

$$\mathrm{H}_k = \sigma_x \qquad\qquad A = \sigma^-$$

- Decomposition



- Add noise operator after every gate

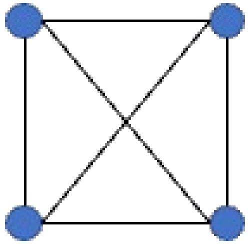
$$U_0 U_1 U_2 \rightarrow e^{\mathcal{L}_N^{k,0} t_{\text{gate}}} U_{k,0} e^{\mathcal{L}_N^{k,1} t_{\text{gate}}} U_{k,1} e^{\mathcal{L}_N^{k,2} t_{\text{gate}}} U_{k,2}$$

$$A_{\text{eff}}^{1,k} = U_{0,k} A U_{0,k}^{\dagger} = \frac{1}{2}\sigma_z - \frac{i}{2}\sigma_y$$

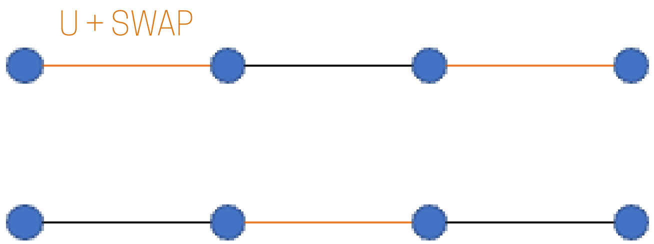
- Change from pure damping to dephasing, damping and excitation

- Transformation depends on
  - Choice of decomposition
  - Available gates of hardware

All-to-All



QSWAP Algorithm



Tutorial: noisy analysis for different algorithms

Hands-on
Noisy-algorithm model: Advanced

Correlated noise

The goal of the tutorial are:
1) Study how large angle two-qubit gates modify the noisy-algorithm model

- Device with linear connectivity: SWAP gate (large angle two-qubit gate)
- The large angle two-qubit gate will introduce correlated noise terms

correlated noise terms

```
SpinLindbladNoiseSystem(3){
(0iY1X, 0iY1X): (8.749999977185001e-2 + i * 0e0),
(0X1iY, 0X1X): (6.2498050000001984e-2 + i * 0e0),
(0Z1X, 1iY): (6.249999999999999e-2 + i * 0e0),
(2iY, 0Z2X): (6.249999999999999e-2 + i * 0e0),
(1X2iY, 1iY2iY): (2.499731250000083e-2 + i * 0e0),
(0X, 0iY): (6.499999995432315e-1 + i * 0e0),
```

Correlated noise

The goal of the tutorial are:
1) Study how large angle two-qubit gates modify the noisy-algorithm model

- Device with linear connectivity: SWAP gate (large angle two-qubit gate)
- The large angle two-qubit gate will introduce correlated noise terms

correlated noise terms

```
SpinLindbladNoiseSystem(3){
(0iY1X, 0iY1X): (8.749999977185001e-2 + i * 0e0),
(0X1iY, 0X1X): (6.249805000001984e-2 + i * 0e0),
(0Z1X, 1iY): (6.249999999999999e-2 + i * 0e0),
(2iY, 0Z2X): (6.249999999999999e-2 + i * 0e0),
(1X2iY, 1iY2iY): (2.499731250000083e-2 + i * 0e0),
(0X, 0iY): (6.499999995432315e-1 + i * 0e0),
```

2) Develop an alternative noise model and compare with the noisy-algorithm model

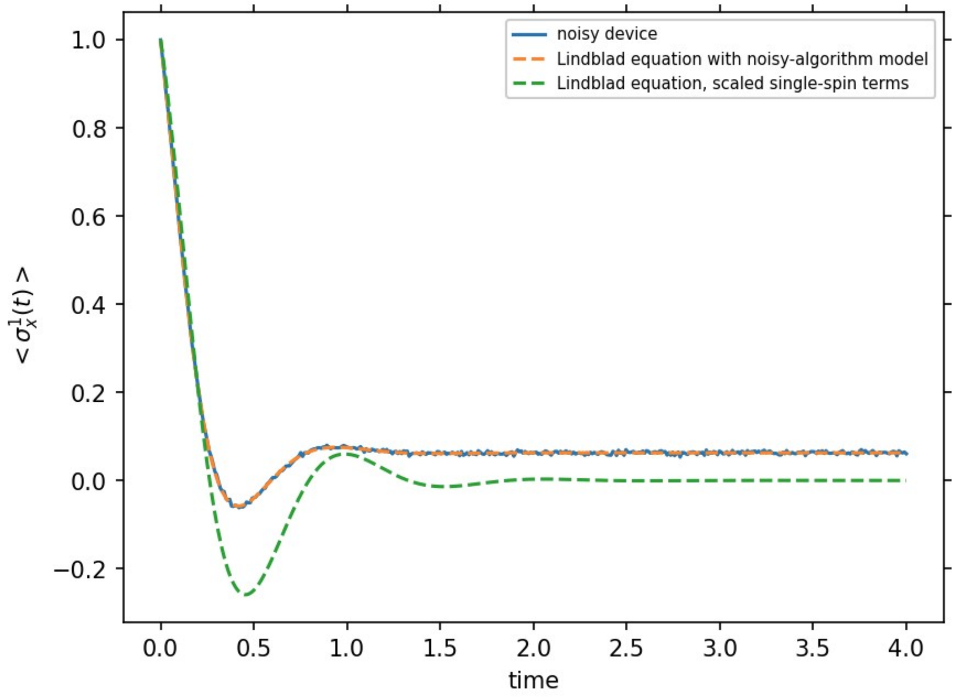
- Noisy-algorithm model: All Lindblad terms
- Alternative noise model: Discard all two-qubit Lindblad terms and scale the single-qubit terms while keeping the noise strength the same.

Noise strength: $M_{\text{tot}} = \sum_{ij} M_{ij}$ rate matrix

Summary

Summary

Correlated noise terms are important. The alternative noise model with scaled single-spin Lindblad terms does not describe execution of circuit.



HQS QUANTUM SIMULATIONS

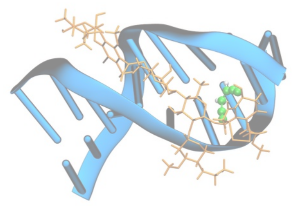
Thank you for your attention!

info@quantumsimulations.de
www.quantumsimulations.de

System-bath approach

Mapping open system model to noisy algorithm model

actual problem



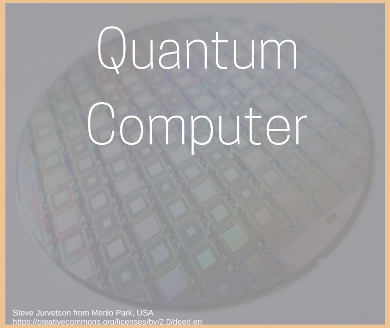
open system model

H_0

Mapping

Noisy algorithm model

$\mathcal{L}_{\text{eff}}$



$$H_0 = H_S + H_C + H_B$$

$$= \underbrace{\frac{\epsilon}{2}\sigma_z}_{H_S} + \underbrace{\sum_k v_k \sigma_x \left(b_k^\dagger + b_k\right)}_{H_C} + \underbrace{\sum_k \omega_k b_k^\dagger b_k}_{H_B}$$

Bring to the same form

$$\dot{\rho} = \mathcal{L}_{\text{eff}}\rho$$

$$= i[\rho, H_S + H_{\text{aux}}] + L\rho$$

Agenda

1) Time-evolution
2) Open-quantum systems
3) Fitting spectral function to bosonic modes
4) Hands-on!

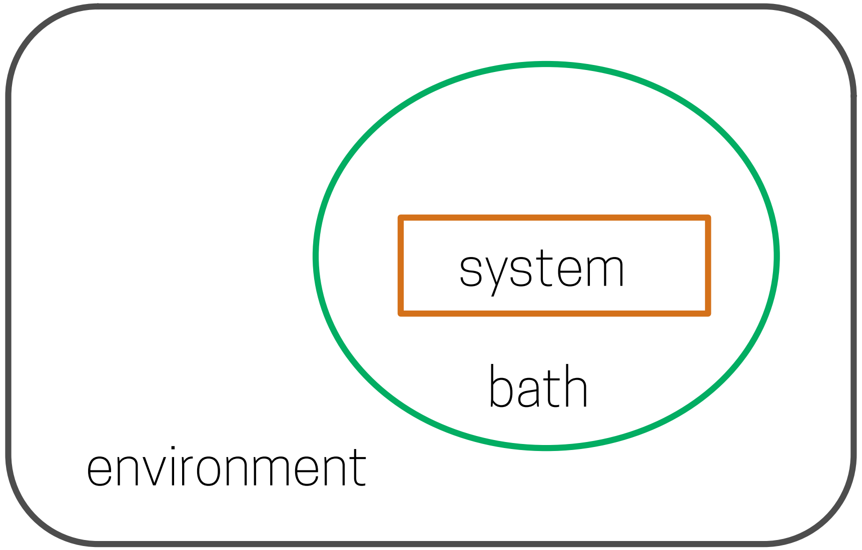
Open-quantum systems

Terminology

System: Spin model, bosons, lattices

Bath: free bosons with arbitrary spectrum

Environment: free bosons with flat spectrum

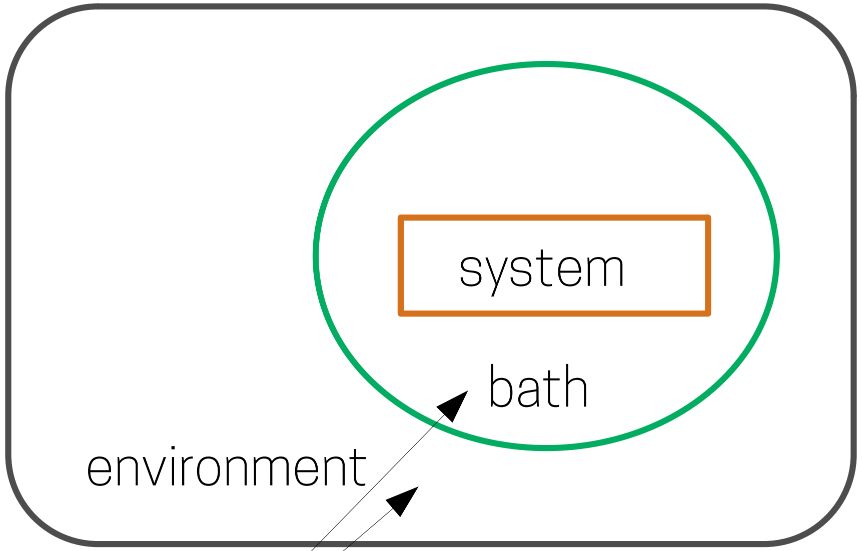


Terminology

System: Spin model, bosons, lattices

Bath: free bosons with arbitrary spectrum

Environment: free bosons with flat spectrum



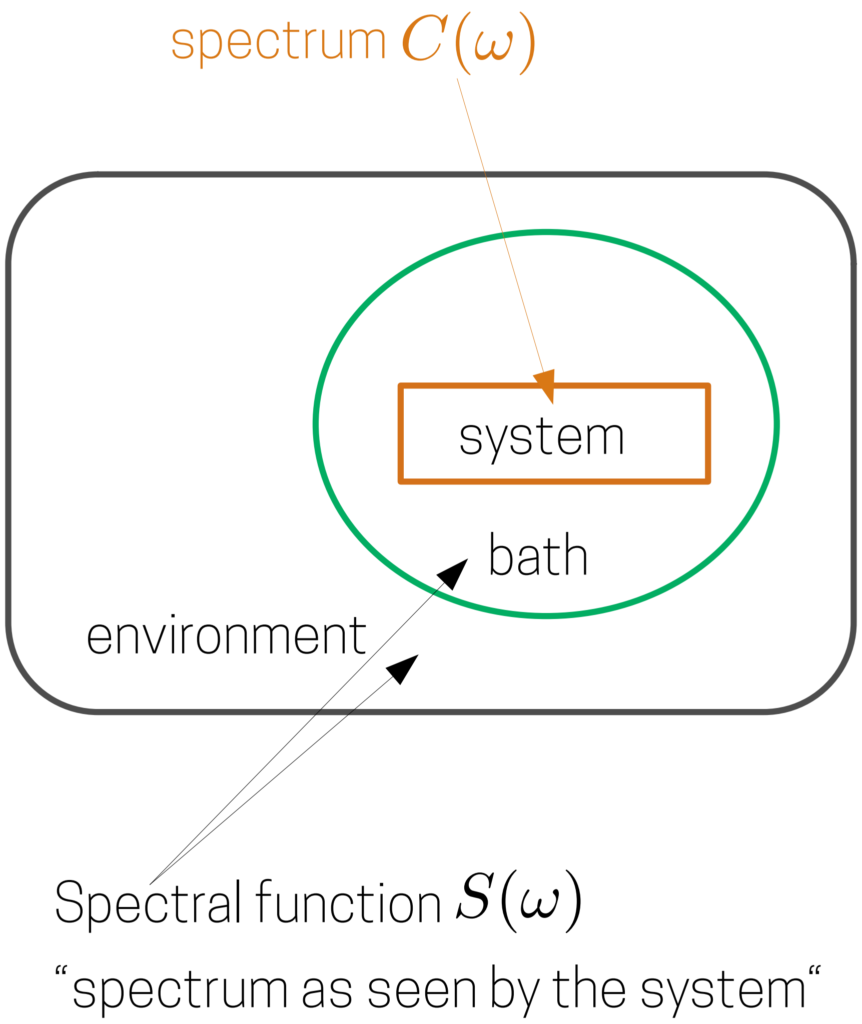
Spectral function $S(\omega)$
"spectrum as seen by the system"

Terminology

System: Spin model, bosons, lattices

Bath: free bosons with arbitrary spectrum

Environment: free bosons with flat spectrum



Hamiltonian for system-environment and system-bath

$$H = H_S + H_C + H_{B,E}$$

Linear coupling $$H_C = O \sum_k v_n (b_k^\dagger + b_k)$$

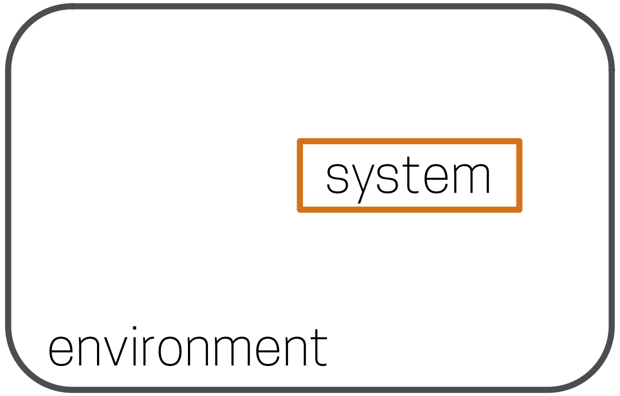
Bosonic bath $$H_{B,E} = \sum_k \omega_k b_k^\dagger b_k$$

Spectral density $$J(\omega) = \sum_k v_k^2 \delta(\omega - \omega_k) \quad (\omega > 0)$$

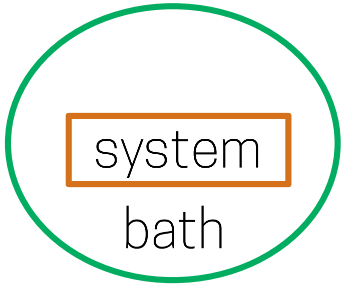
Spectral function $$S(\omega) = \frac{\sum_{k=1}^{\infty} v_k^2 \delta(\omega - \omega_k)}{1 - \exp\left(-\frac{\omega}{k_\mathrm{B} T}\right)} \mathrm{sign}(\omega)$$

All coupling combinations are possible

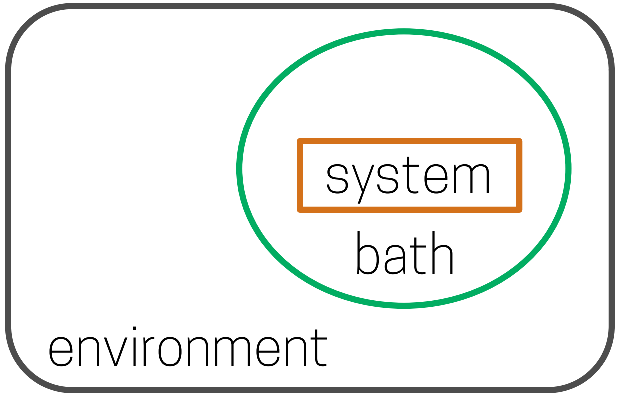
System-environment



System-bath



System-bath-environment

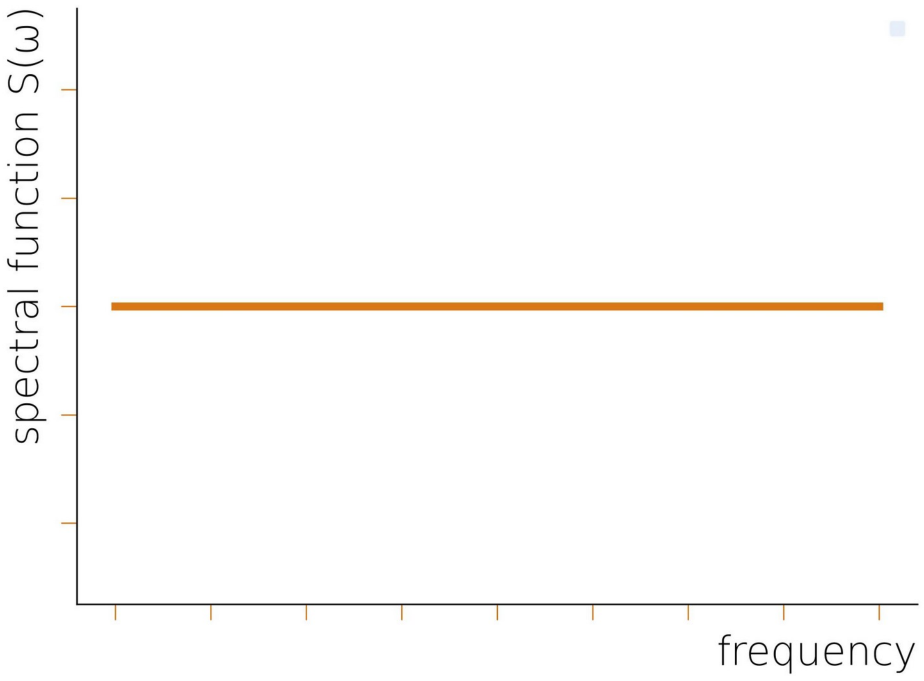


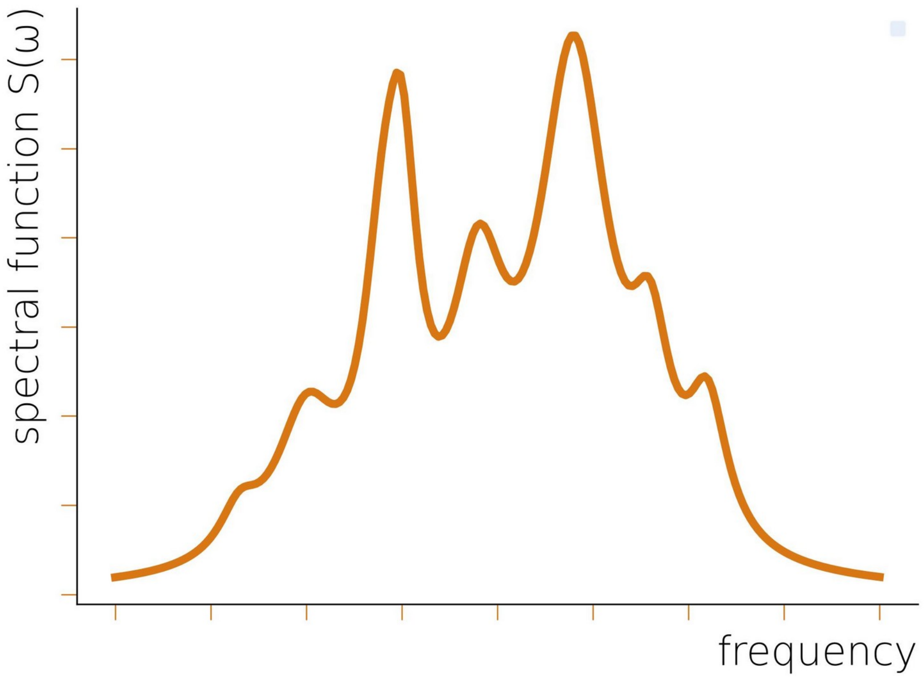
Fitting of spectral function

Flat and structured spectral function

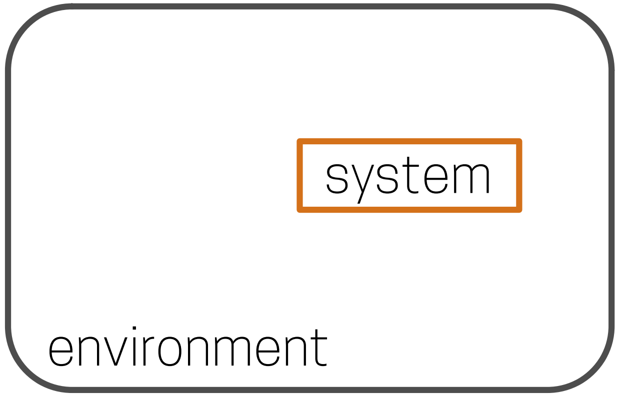
Flat spectral function ➡ environment

Structured spectral function ➡ bath





What does the difference between bath and environment matter?



For the environment we can do perturbation theory

$$H = H_S + H_C + H_E$$

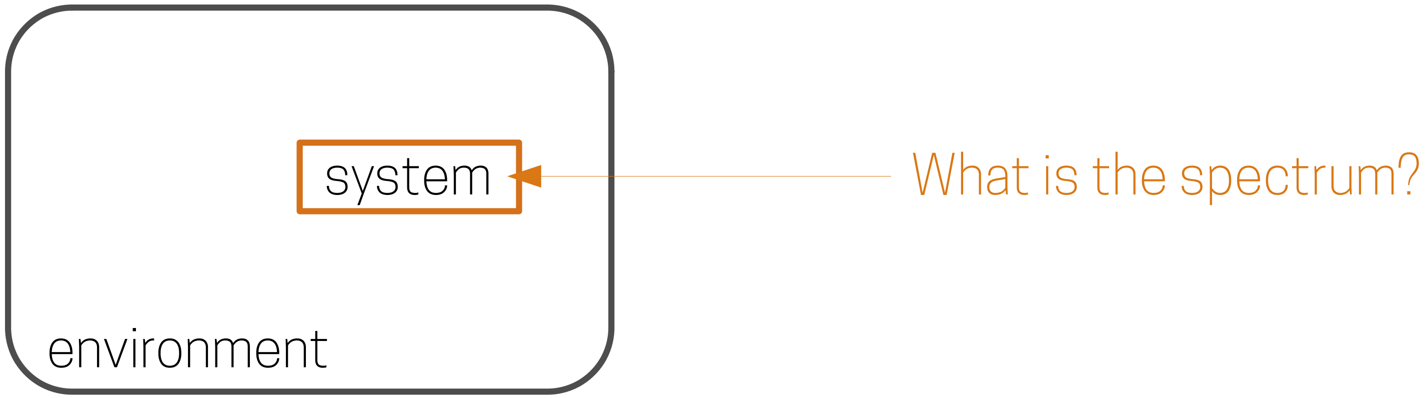
System Hamiltonian $\quad H_S = \omega_0 a^\dagger a$

Equation of motion for density matrix for flat spectral density

$$\dot{\rho} = \underbrace{-i\omega_0[a^\dagger a, \rho]}_{\text{coherent evolution}} + \underbrace{\frac{\kappa}{2}(\bar{n}+1)(2a\rho a^\dagger - a^\dagger a\rho - \rho a^\dagger a) + \frac{\kappa}{2}\bar{n}(2a^\dagger \rho a - aa^\dagger \rho - \rho a a^\dagger)}_{\text{damping}}$$

$\bar{n}$: thermal number of phonons

Spectral function of the system

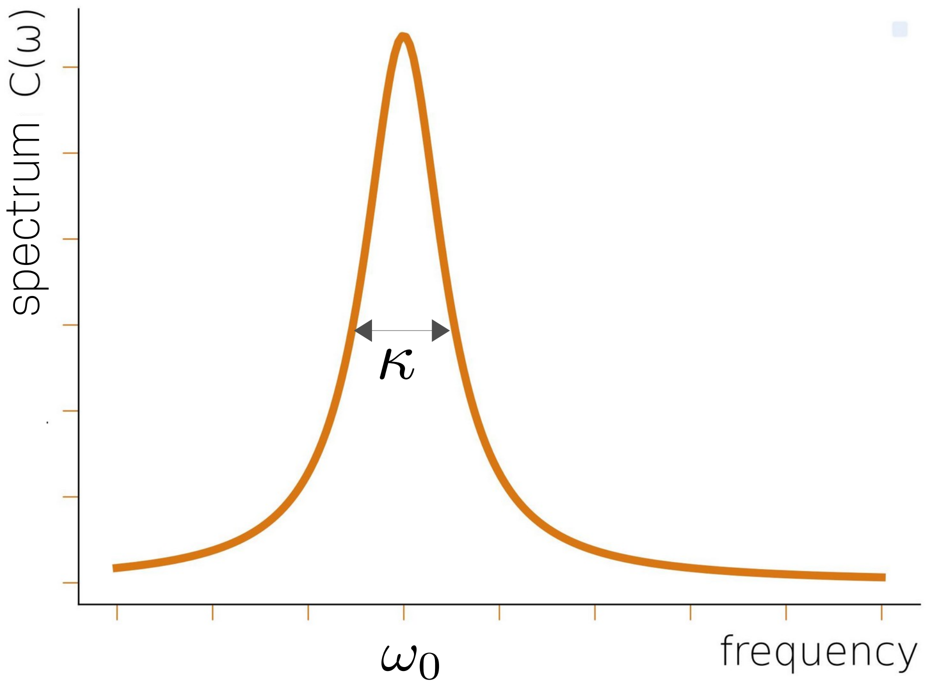


Example 1: Single oscillator

$$H_S = \omega_0 a^\dagger a$$

$$C(\omega) = \int dt\, e^{i\omega t} \langle a^\dagger(t) a(0) \rangle$$

$$= \frac{\kappa}{(\kappa/2)^2 + (\omega - \omega_0)^2}$$



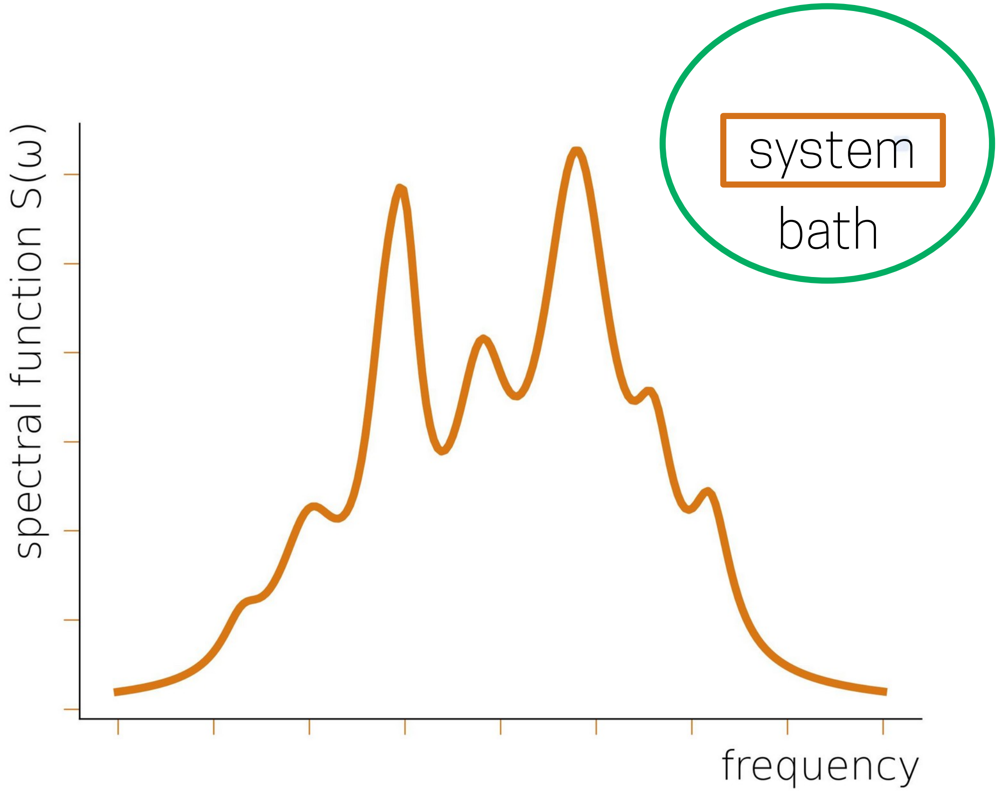
System coupled to bath

$$H = H_S + H_C + H_B$$

$$H_S = \varepsilon\sigma_z/2$$

$$H_C = \sigma_x \sum_n v_k (b_k^\dagger + b_k)$$

$$H_B = \sum_k \omega_k b_k^\dagger b_k$$

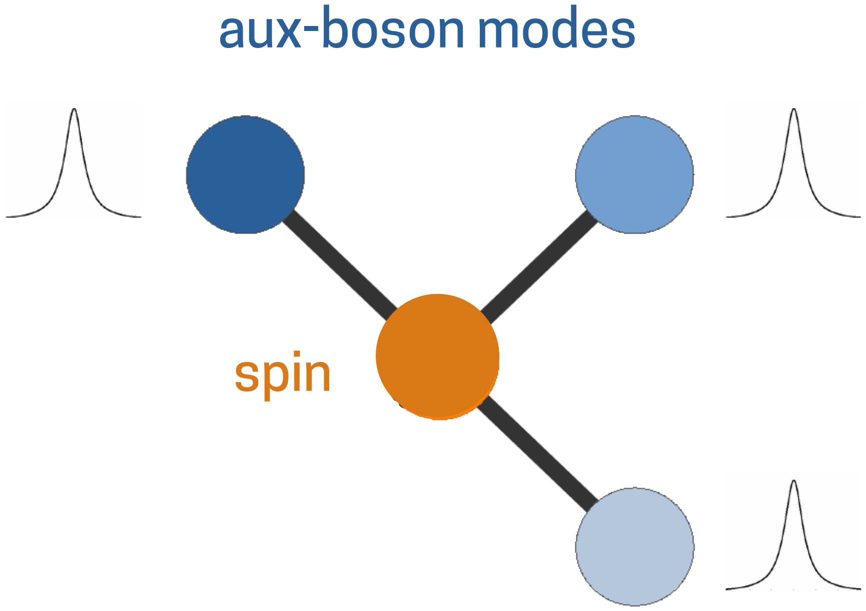


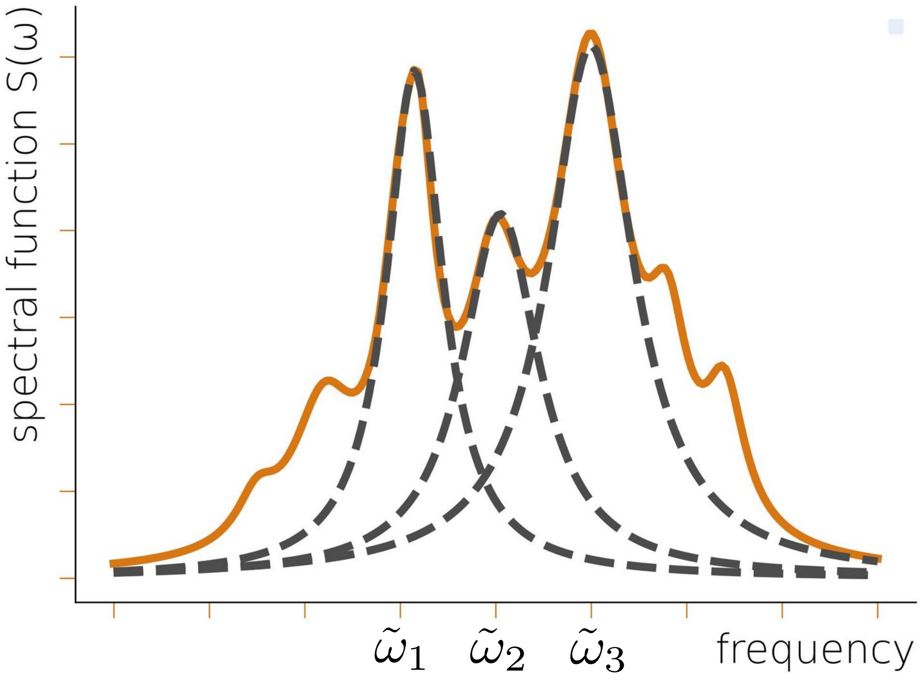
System coupled to bath – fit to spectral function

$$H = H_S + H_C + H_B = H_S + H_{\mathrm{aux}} + \tilde{H}_C + \tilde{H}_E$$

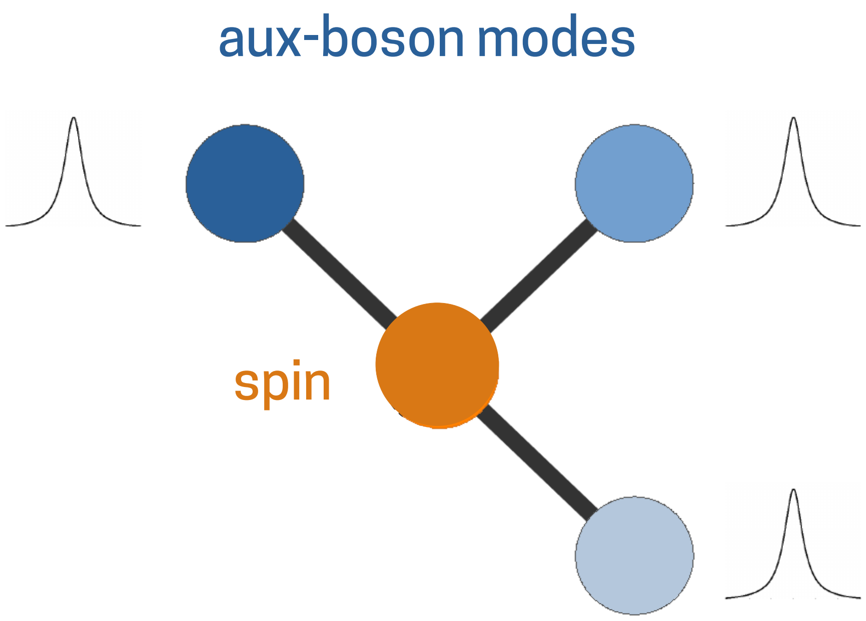
$$H_S = \varepsilon\sigma_z/2$$

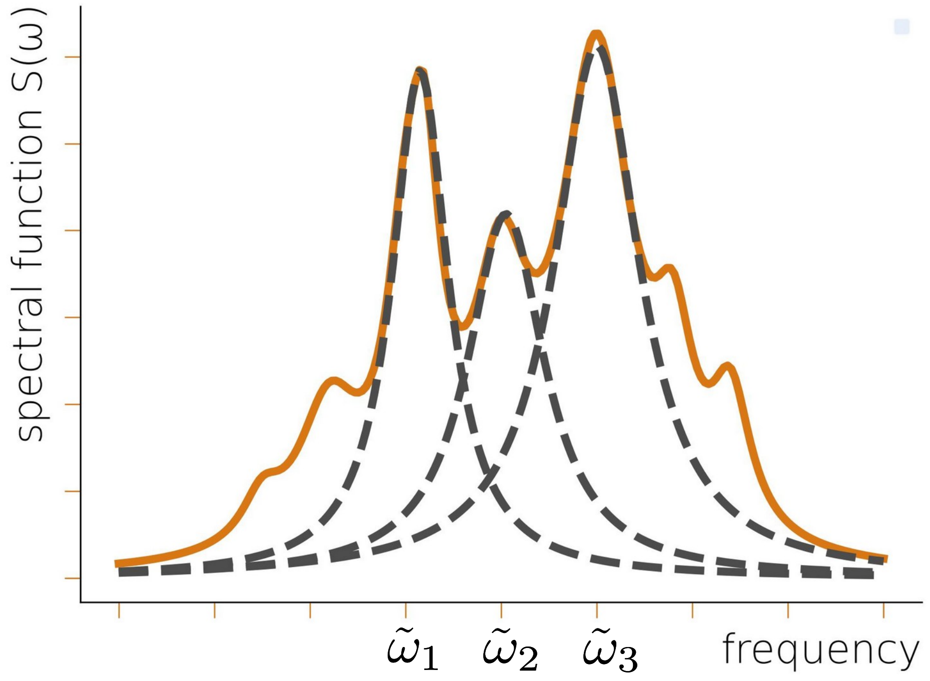
$$H_{\mathrm{aux}} = \sigma_x \sum_{i=1}^{3} v_i(a_i^\dagger + a_i) + \sum_{i=1}^{3} \tilde{\omega}_i a_i^\dagger a_i$$

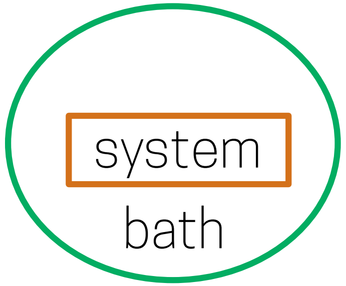


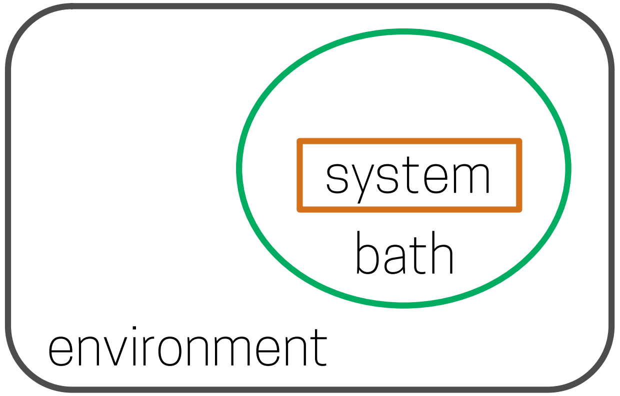


System coupled to bath – fit to spectral function









Hand-on!

System-spin coupled to ohmic bath

- System-Hamiltonian (single spin)
- Device
- Ohmic spectral function

$$S(\omega) = \frac{4\pi\hbar^2\alpha\omega}{1-\exp\left(-\frac{\hbar\omega}{k_\mathrm{B}T}\right)} \, ,$$

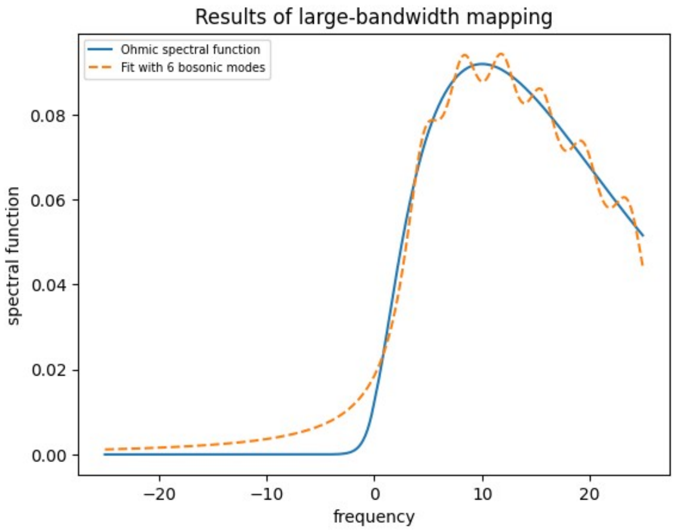
- Fitting using the *BathFitter*
  - Fit ohmic spectral function to boson modes
  - Represent spins by bosons
- System-bath QuantumProgram
- Time-evolution with QuEST

BathFitter is a python class for fitting open-quantum systems. It is initialized with

- Number of bosonic modes
- Number of spins to represent bosons
- Broadening constrain

We use the functions:

- *fit_boson_bath_to_spectral_function*
- *fit_spin_bath_to_spectral_function*



- Click to add Text