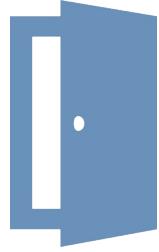




Leibniz Supercomputing Centre  
of the Bavarian Academy of Sciences and Humanities



## Emergency Exit

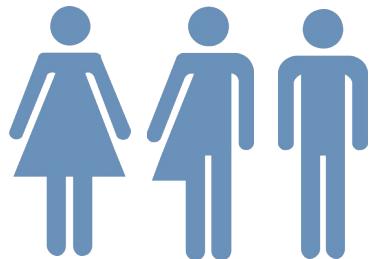
Please exit using glass back door immediately outside this room. Convene at the assembly point on service road between TUM and LRZ in the direction of the compute cube (end of road).



## WiFi

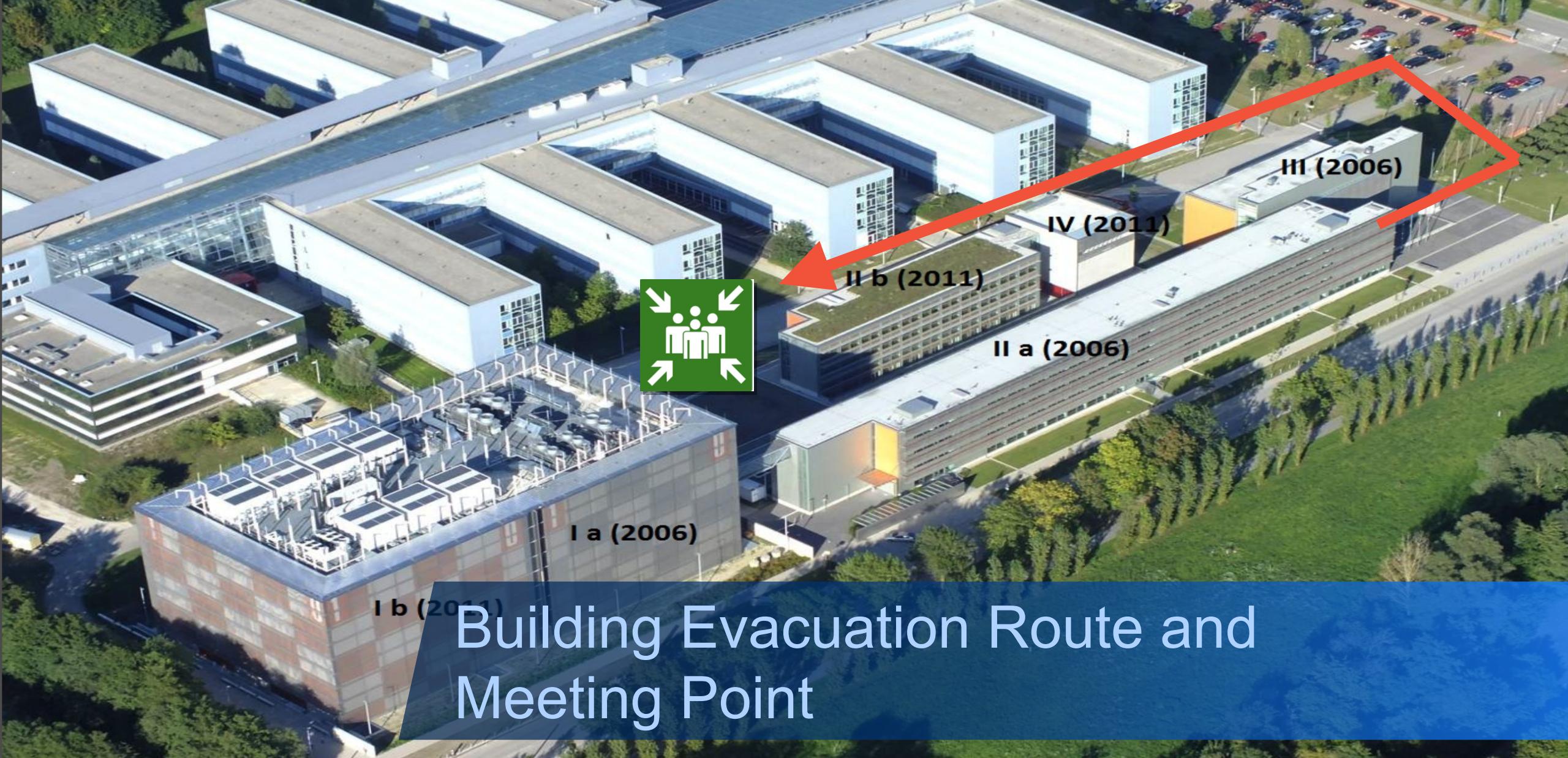
Eduroam

BayernWLAN



## Restrooms

Located one floor down (stairs by elevator)





Leibniz Supercomputing Centre  
of the Bavarian Academy of Sciences and Humanities

# Working with IQM Quantum Computer

LRZ's Ecosystem | 2024.07.30

- ❖ Introduction to LRZ full stack
  
- ❖ Our Access Paths
  - Munich Quantum Portal (MQP) Access
    - ◆ Dashboard
    - ◆ MQP Provider
      - Accessing the MQP - Preparation
  
  - HPCQC Access
    - ◆ Login to BEAST / Wolpertinger
    - ◆ Setup / Jobscrip
    - ◆ Compile and run using QPI and Qiskit

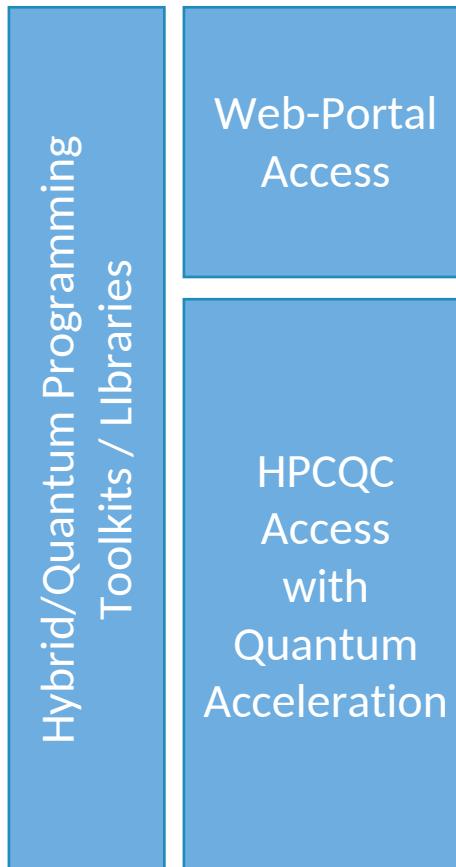
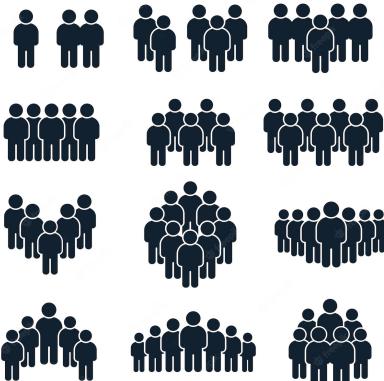


# Introduction

LRZ's full stack



### Wide User Communities



### Quantum Compiler

Optimizers  
Schedulers  
Transpilers  
Mappers



Quantum Server



Quantum Server



Quantum Server

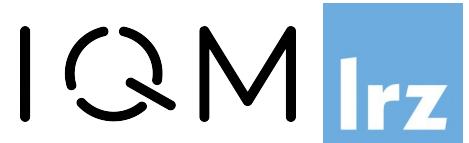


Enabling Domain User Communities to Compute on Quantum Devices

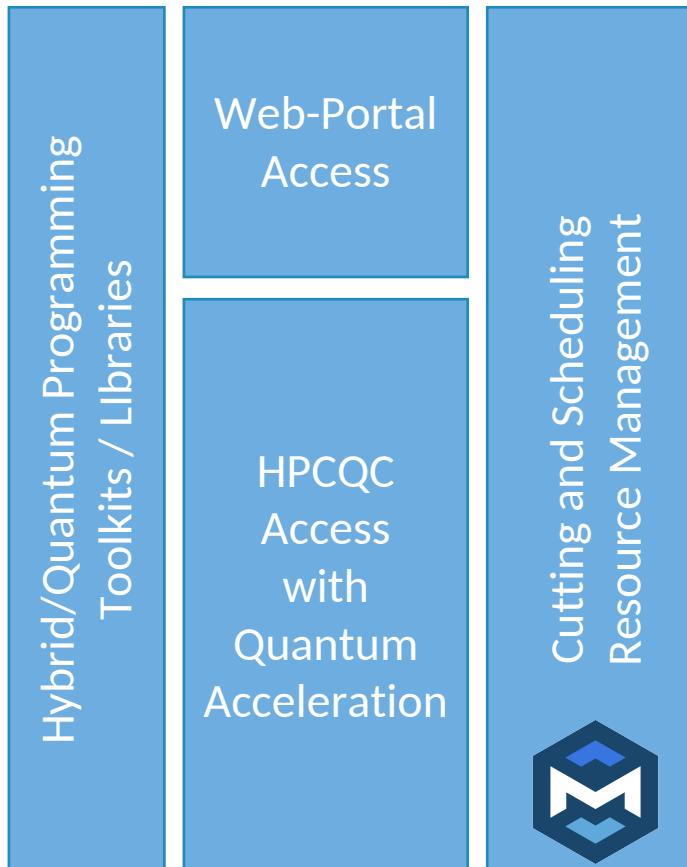
Many/Different Quantum Devices

# Introduction

## Munich Quantum Software Stack: Separation of Concerns



### Wide User Communities



## Introduction

### Munich Quantum Portal Access: Dashboard



Website exposing access to account and token management for our end users

### Web-Portal Access

The end users can track:

- Status of submitted jobs
- Available resources

A screenshot of the Munich Quantum Portal (MQP) dashboard. The interface has a dark header with the MQP logo and a light-colored sidebar on the left containing links for Status, Tokens, Jobs, Budgets, Resources (which is highlighted), and Log Out. The main content area is titled "Available Resources" and displays four quantum devices: QExa20, QLM, Q20, and WMI3. Each device card provides details such as name, vendor, status (Online), number of qubits, and quantum technology. At the bottom of the dashboard, there are links for Data Privacy, Imprint, Accessibility, and a copyright notice: © 2024 Munich Quantum Portal.

Device	Vendor	Status	Qubits	Quantum Technology
QExa20	IQM	Online	20	Superconducting
QLM	Eviden	Online	37	Simulator
Q20	IQM	Online	20	Superconducting
WMI3	WMI	Online	3	Superconducting
Q5	IQM	Online	5	Superconducting

Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

## Introduction

### Munich Quantum Portal Access: Qiskit Provider

Wide User Communities



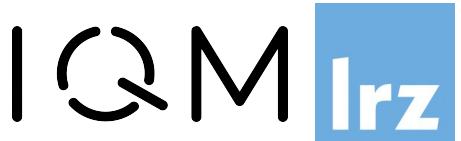
Allows access programmatically directly from Qiskit to the quantum resources



The access is granted through the authentication with the LRZ/Quantum credentials via MQP tokens

### Web-Portal Access

Enabling Domain User Communities to Compute on Quantum Devices



```
from qiskit import QuantumCircuit
from bayesian_quantum_portal_provider import BOPProvider
token = 'wCLYUWUR2WI94...excspQLni3vEbgMHM3DdNkqgLWN0VOIcnaTMDvxZ53H5mMFix'
provider = BOPProvider(token=token)

# get a specific backend by its name
backend = provider.get_backend("Q20")

# create a circuit
qcirc = QuantumCircuit(2, 2)
qcirc.h(0)
qcirc.cx(0,1)
qcirc.measure_all()

print(qcirc)

job = backend.run(qcirc, shots=128)

# get the result of the job
result = job.result()

# get the job's resulting counts
counts = result.get_counts()

print("")
for state in counts:
    print(f"|{state}>: {counts[state]}")
```

A programming interface implemented as a library in C to express quantum circuits

HPCQC Access with Quantum Acceleration

Tailored for HPC environments: QPI parses the quantum circuit into LLVM and offloads it to the middle-end



Enabling Domain User Communities to Compute on Quantum Devices

```
#include "qpi.h"

void bell(int numshots, uint32_t *output)
{
    Qcircuit circuit;

    qCircuitBegin(&circuit);
    qH(0);
    qCX(0,1);
    qMeasureAll();
    qCircuitEnd();

    Qstatus status;
    qExecute(circuit, numshots, &status);
    printf("\n [QPI]...Circuit submitted");

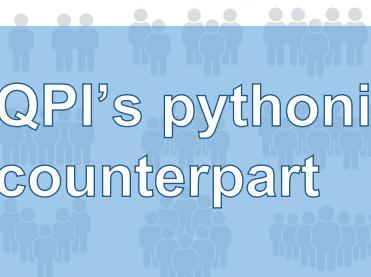
    qWait(status);
    printf("\n [QPI]...Execution is complete");

    printf("\n [QPI]...Received results:\n\n");
    int *Results = qRead(circuit), i;
    for (i=0; i<(long)1 << qCircuitWidth(circuit); i++)
        printf("\t%d: %d\n", i, Results[i]);

    qCircuitFree(circuit);
}
```

Many/Different Quantum Devices

### Wide User Communities



QPI's pythonic counterpart

More familiar interface for many researchers

Hybrid Quantum Programming

Web-Portal Access

HPCQC Access with Quantum Acceleration

Figures of Merit and Constraints

```
from qiskit import QuantumCircuit
from hpc_offload_provider import HPCOffloadProvider

provider = HPCOffloadProvider()
backend = provider.get_backend()

circuit = QuantumCircuit(2,2)
circuit.h(0)
circuit.cx(0,1)
circuit.measure_all()

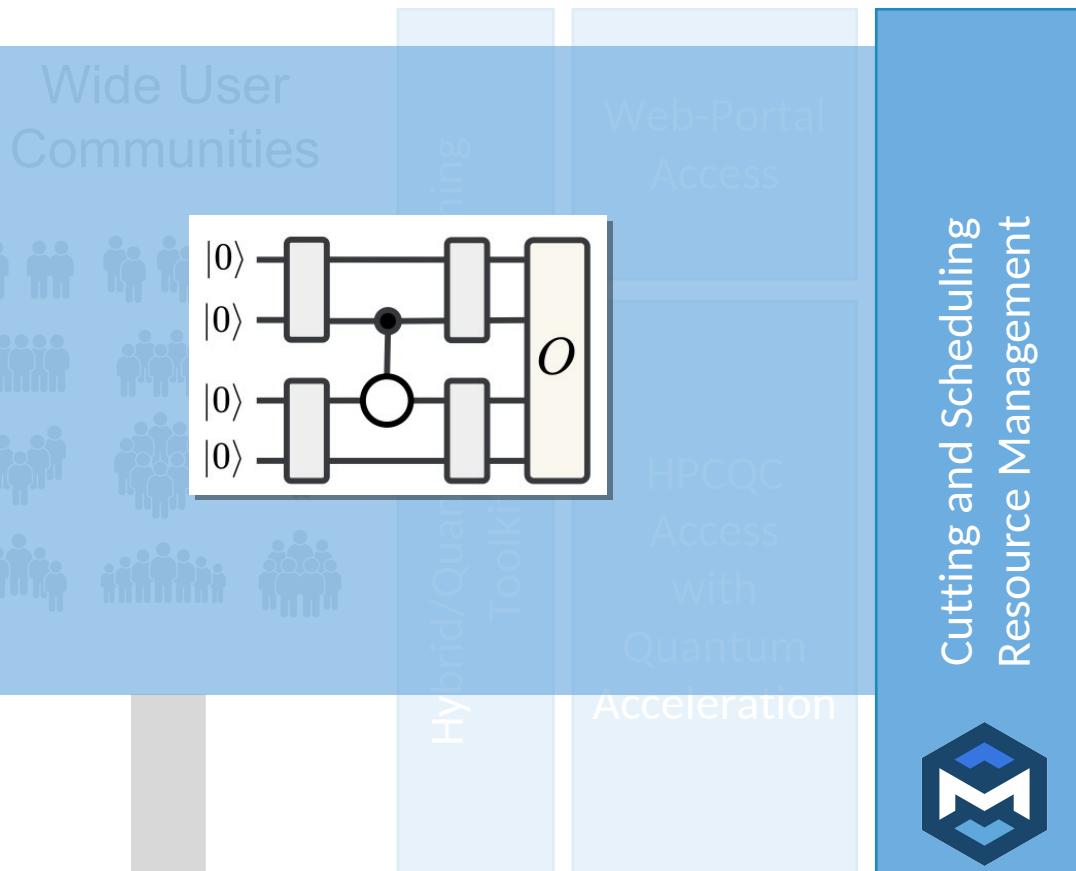
job = backend.run(circuit, shots=100)
print("[Qiskit]...Circuit submitted")

counts = job.result().get_counts()
print("[Qiskit]...Execution is complete")

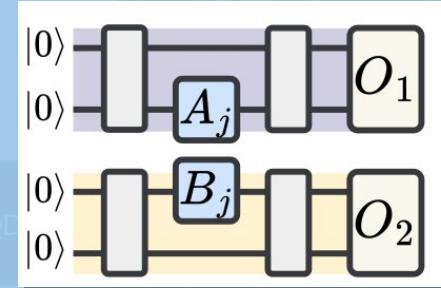
print("[Qiskit]...Received results:\n")
for binary_key in counts:
    decimal_key = int(binary_key[::-1], 2)
    print(f"\t{decimal_key % 10}: {counts[binary_key]}")
```

Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices



Quantum divide-and-conquer technique to efficiently run quantum circuits that exceed the size of a target device



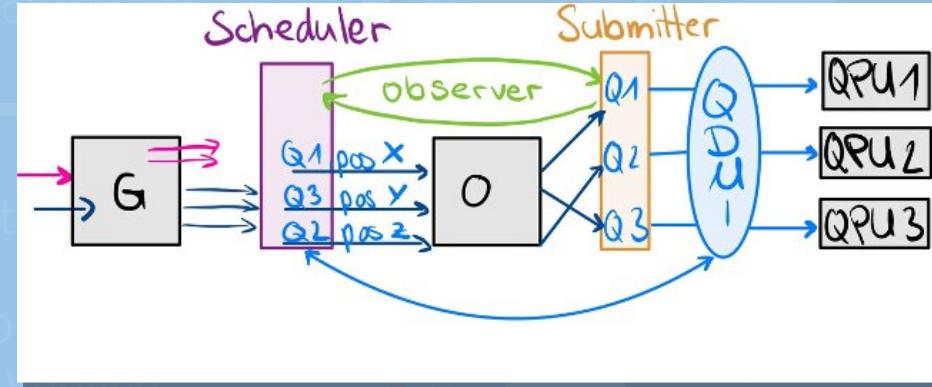
Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Optimize the utilization of quantum resources while accommodating individual device preferences and constraints

E.g.: number of qubits, fidelities, estimated execution time

### Cutting and Scheduling Resource Management



Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

### Wide User Communities



Programming  
Languages

Web-Portal  
Access

Cutting and Scheduling  
Resource Management

Figures of Merit and  
Constraints

Quantum Device Management Interface

QDMI Backends



A Pythonic solution that  
utilizes Qiskit's framework  
for transpiling to the  
supported gate set and  
mapping to the topology of  
the target device



Quantum Compiler

Optimizers  
Verifiers  
Transpilers  
Mappers



QDMI Backends

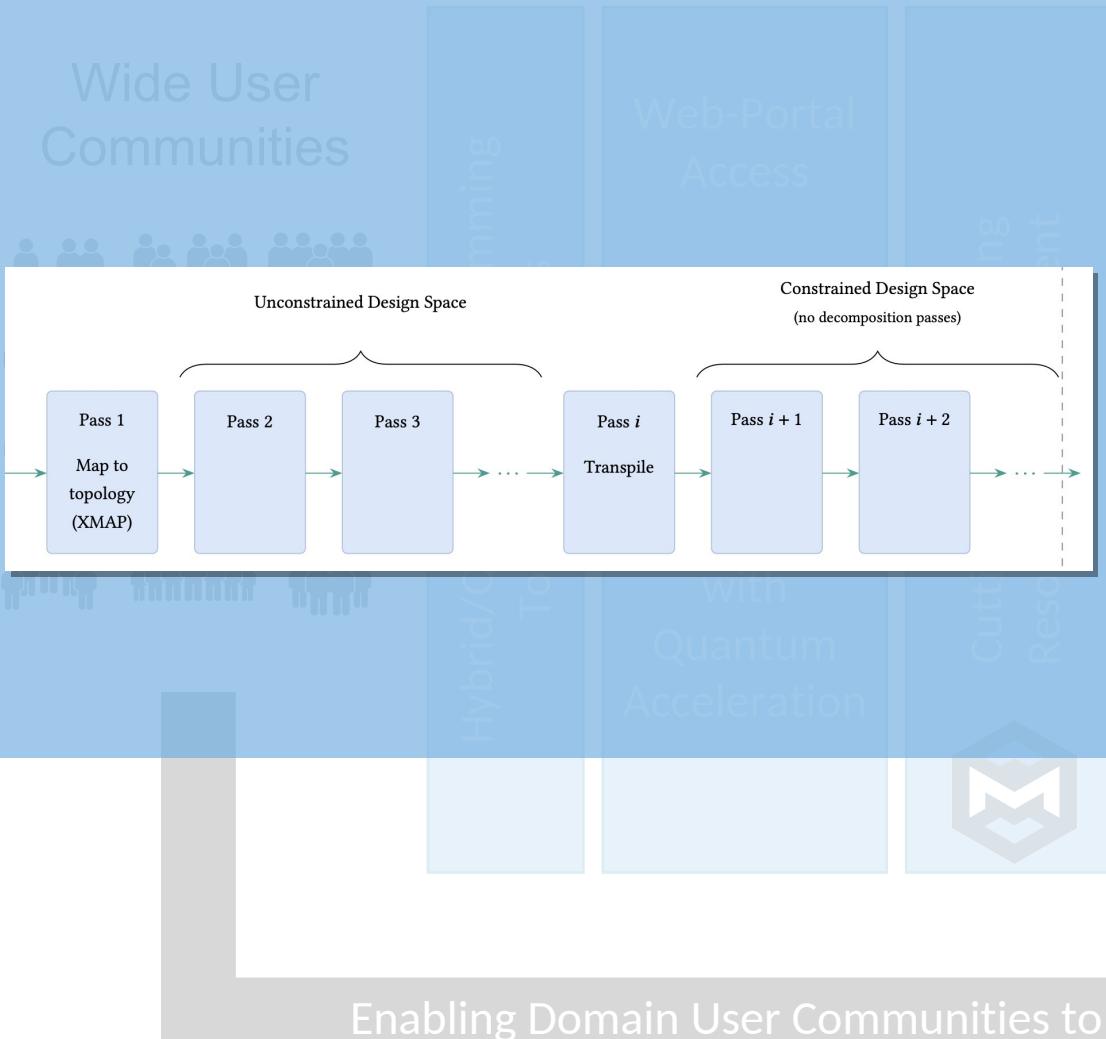


QDMI Backends



Enabling Domain User Communities to Compute on Quantum Devices

Many/Different  
Quantum Devices



A more performant, better tailored solution for HPC ecosystems implemented in C++ that adopts an LLVM-compliant IR



All transformations to the submitted quantum circuit occur in the form of custom-made LLVM passes

Its primary objective is to create an abstraction layer across various quantum technologies

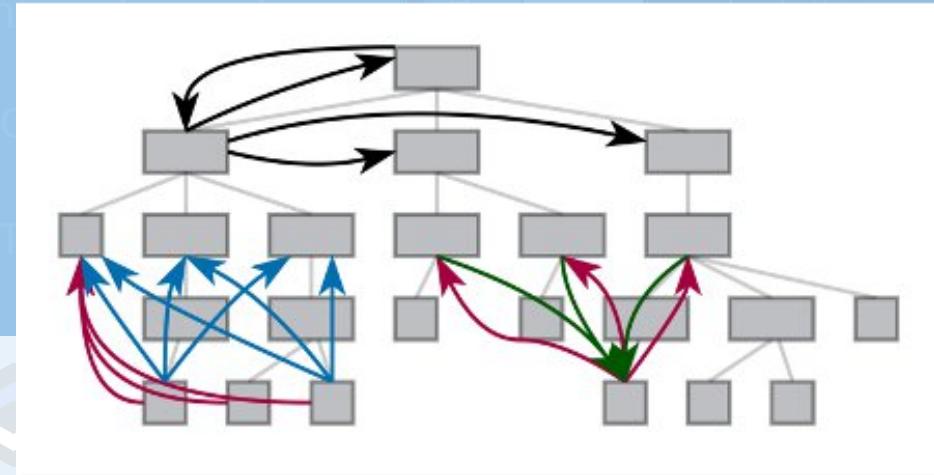
It retrieves information from the devices using QDMI, then computes and caches more complex queries

Hybrid

Acceleration



## Figures of Merit and Constraints

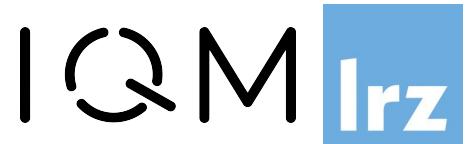


Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

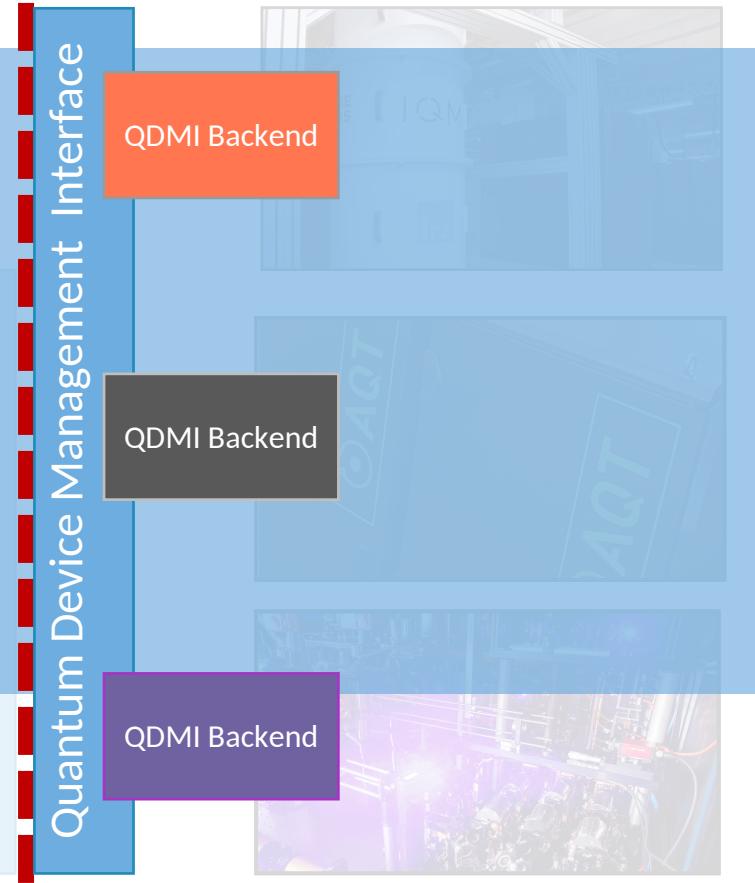
# Introduction

## Quantum Device Management Interface (QDMI)



```
void compile_and_run(Module *qirmmod, QDMI_Device dev, int numshots)
{
    QDMI_Fragment frag;
    QDMI_Job job;
    QDMI_Status status;
    QIR_pass_t *pass = NULL;
    QRM_agnostic_selector(selector_a, pass);
    pass = NULL;
    while(pass = next_pass(pass))
        pass->run(qirmmod);
    QRM_specific_selector(selector_s, pass);
    pass = NULL;
    while(pass = next_pass(pass))
        pass->run(dev, qirmmod);
    QDMI_control_pack_qir(qirmmod, &frag);
    QDMI_control_submit(dev, frag, numshots, &job);
    QDMI_control_wait(dev, job, &status);
    //...
}
```

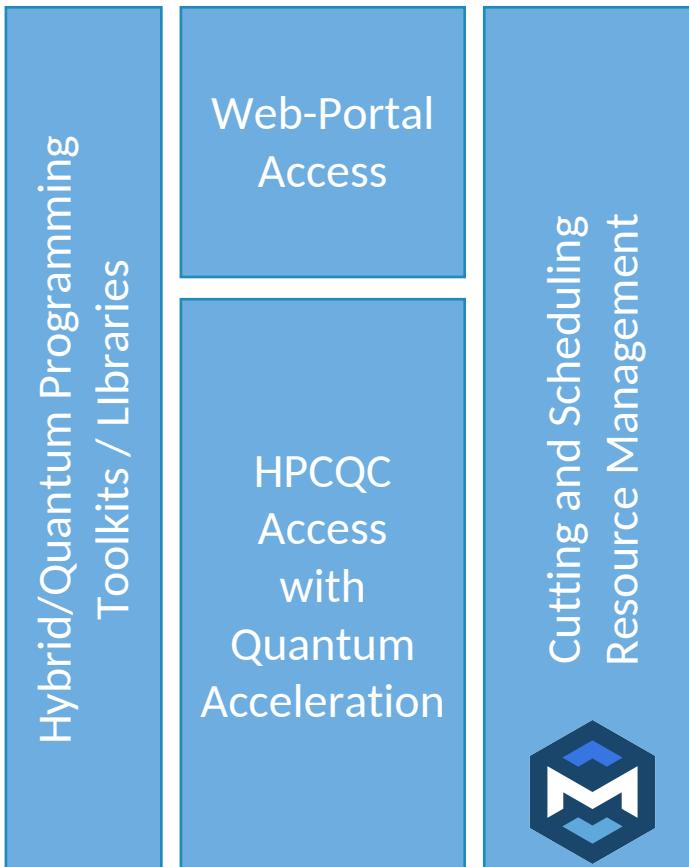
The QDMI backends enable the rest of the software stack to automatically retrieve and adapt to changing physical characteristics and constraints of each platform



Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

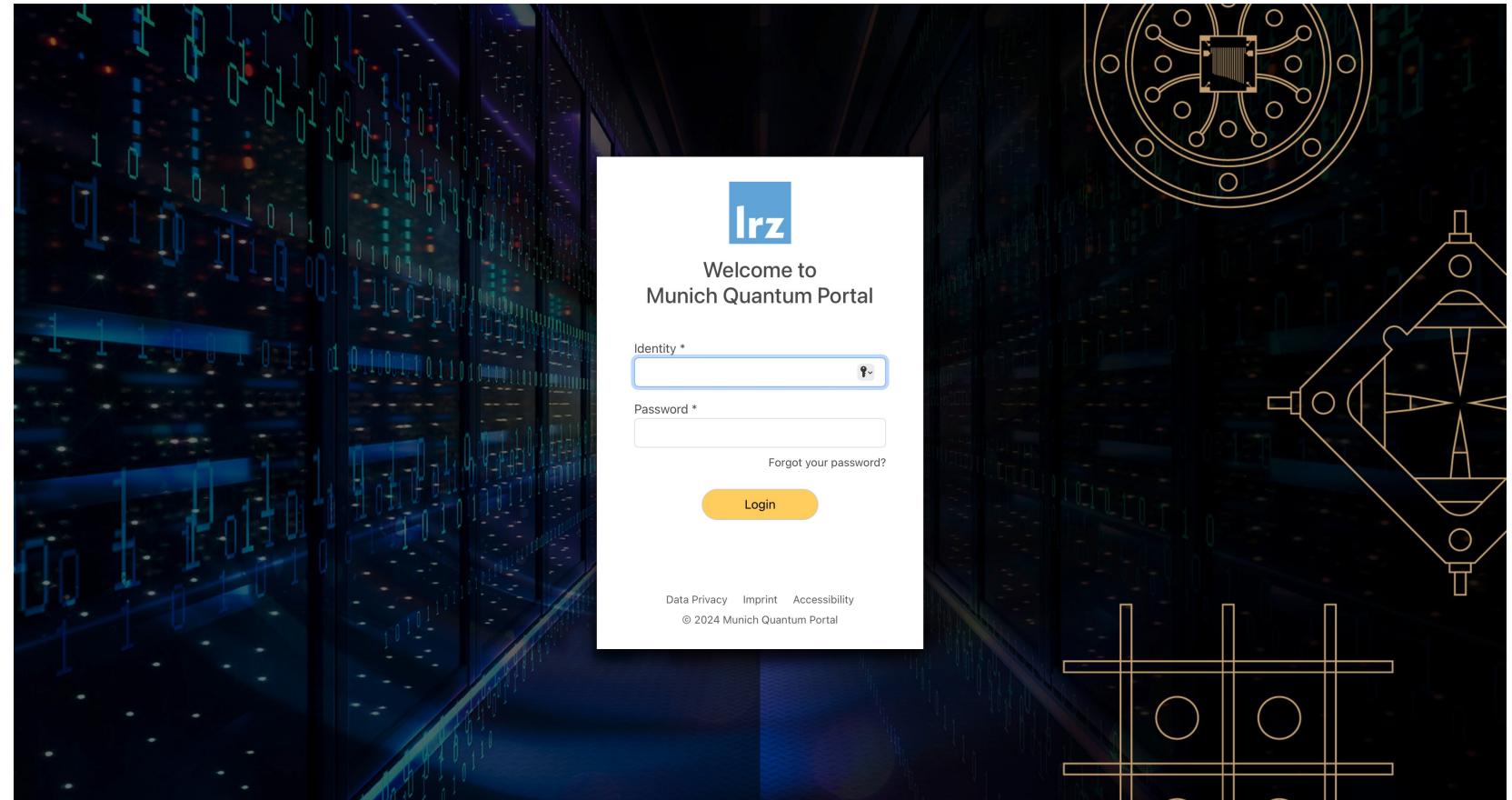
### Wide User Communities



# Access Points

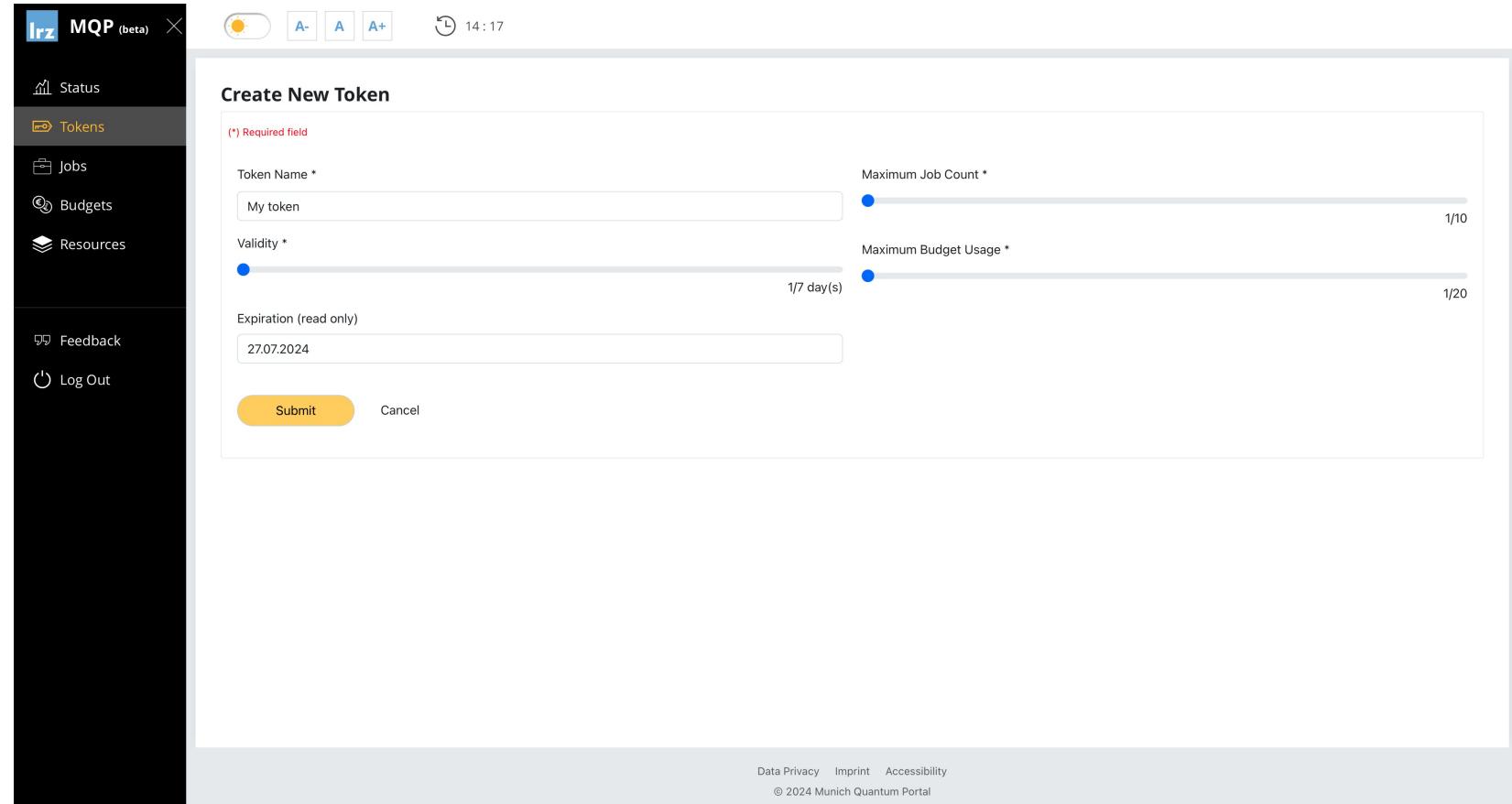
Munich Quantum Portal (MQP) Access: Dashboard

**Visit** <https://portal.quantum.lrz.de>



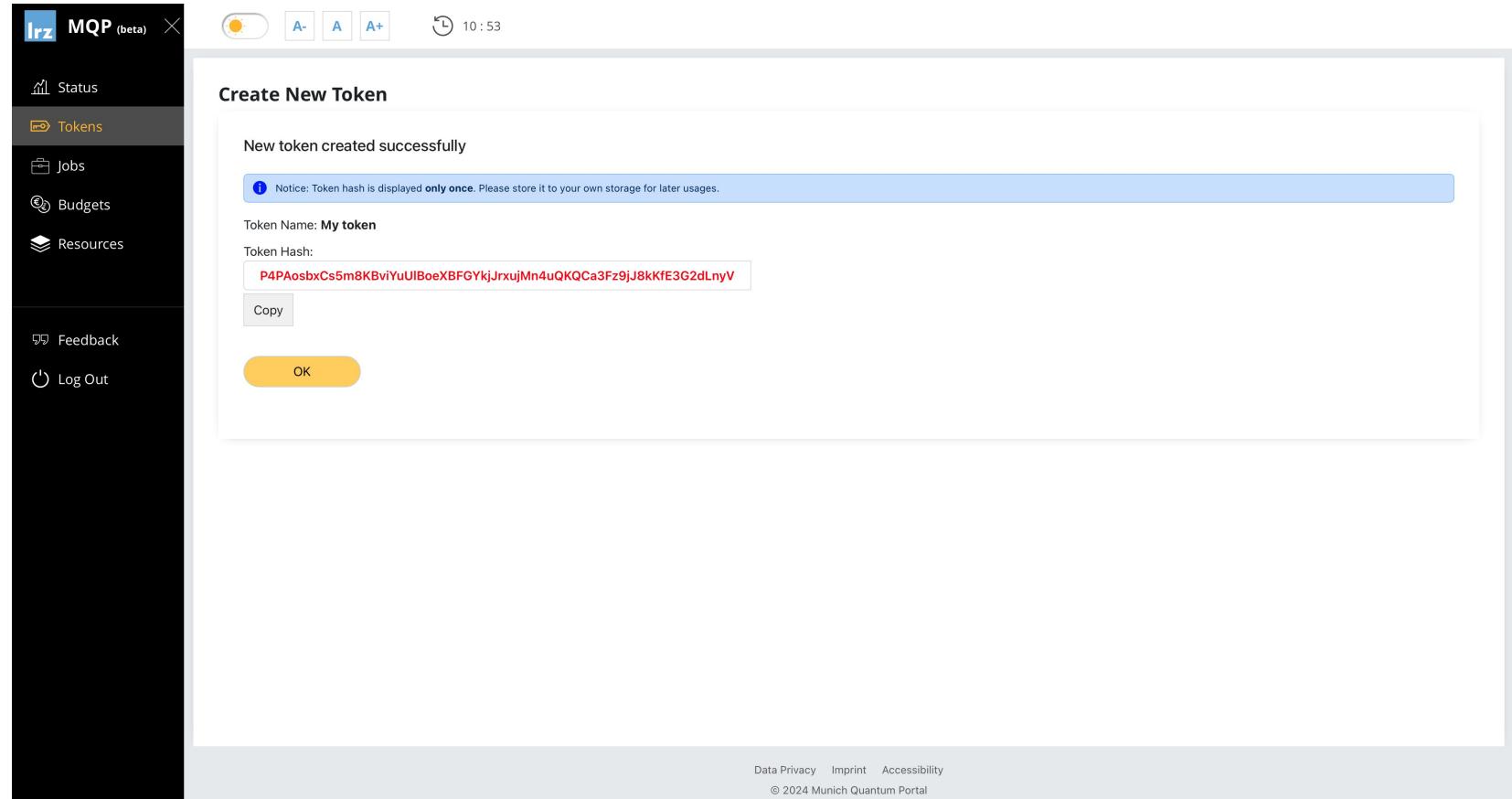
Authentication to the web frontend is compatible with the existing LRZ Active Directory / LDAP infrastructure, enabling users to use their preexisting university-affiliated accounts to log in

### Create a new token



The screenshot shows the 'Create New Token' page of the MQP (beta) web interface. The left sidebar includes links for Status, Tokens (which is the active tab), Jobs, Budgets, Resources, Feedback, and Log Out. The main content area has a title 'Create New Token' and a note '(\*) Required field'. It contains fields for 'Token Name \*' (set to 'My token'), 'Validity \*' (set to '1/7 day(s)'), 'Expiration (read only)' (set to '27.07.2024'), 'Maximum Job Count \*' (set to '1/10'), and 'Maximum Budget Usage \*' (set to '1/20'). At the bottom are 'Submit' and 'Cancel' buttons.

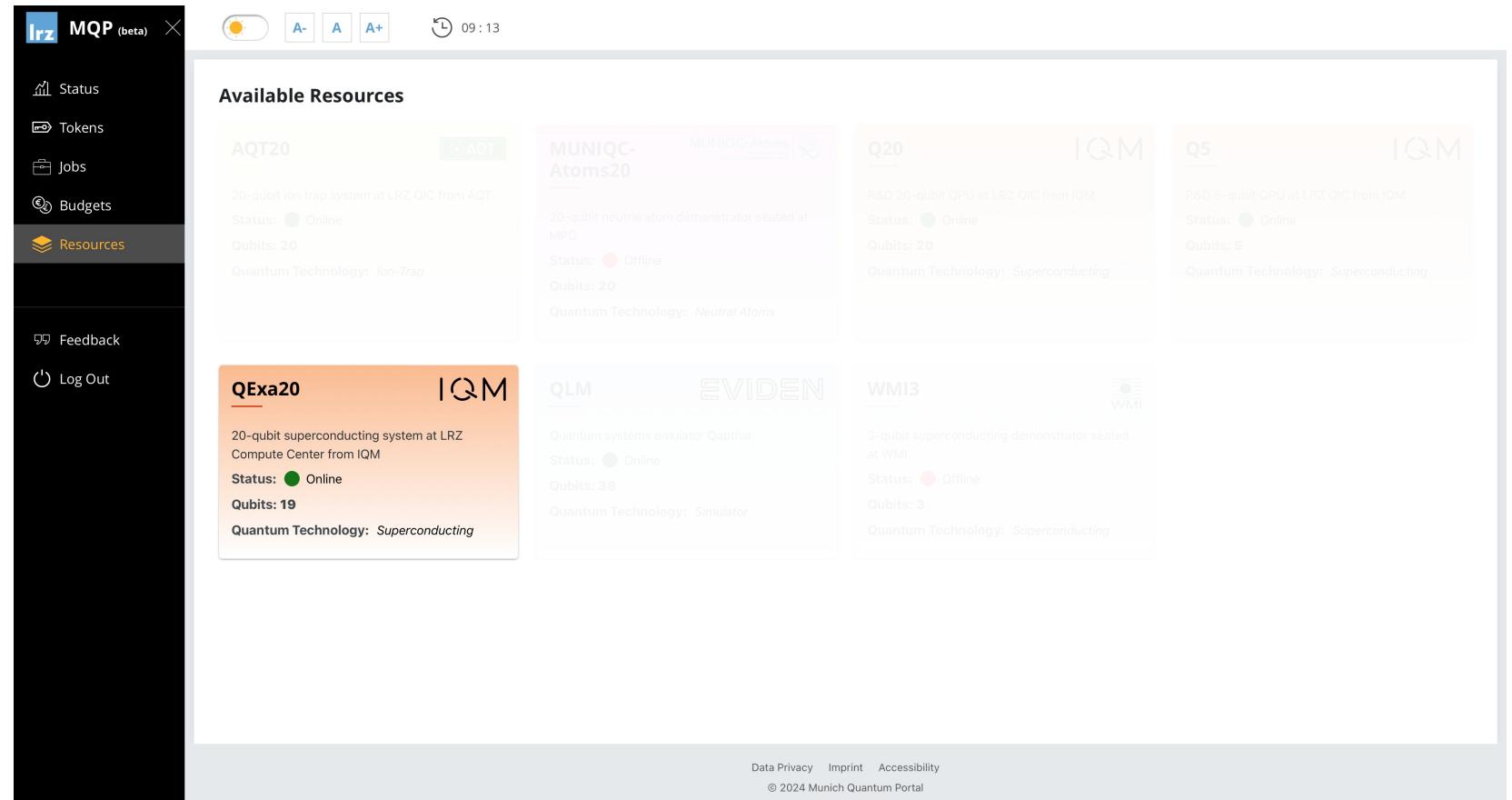
### Create a new token



The screenshot shows the MQP (beta) dashboard. On the left, a sidebar menu includes Status, Tokens (which is the active tab), Jobs, Budgets, Resources, Feedback, and Log Out. The main content area is titled "Create New Token" and displays a success message: "New token created successfully". It shows a token name "My token" and a long token hash "P4PAosbxCs5m8KBviYuUlBoeXBFGYkjJrxujMn4uQKQCa3Fz9jJ8kKfE3G2dLnyV". A "Copy" button is available to copy the token hash. At the bottom right of the modal is an "OK" button. The footer of the page includes links for Data Privacy, Imprint, Accessibility, and the copyright notice "© 2024 Munich Quantum Portal".

Remember to keep it somewhere safe!

**Check the status of  
your target accelerator**

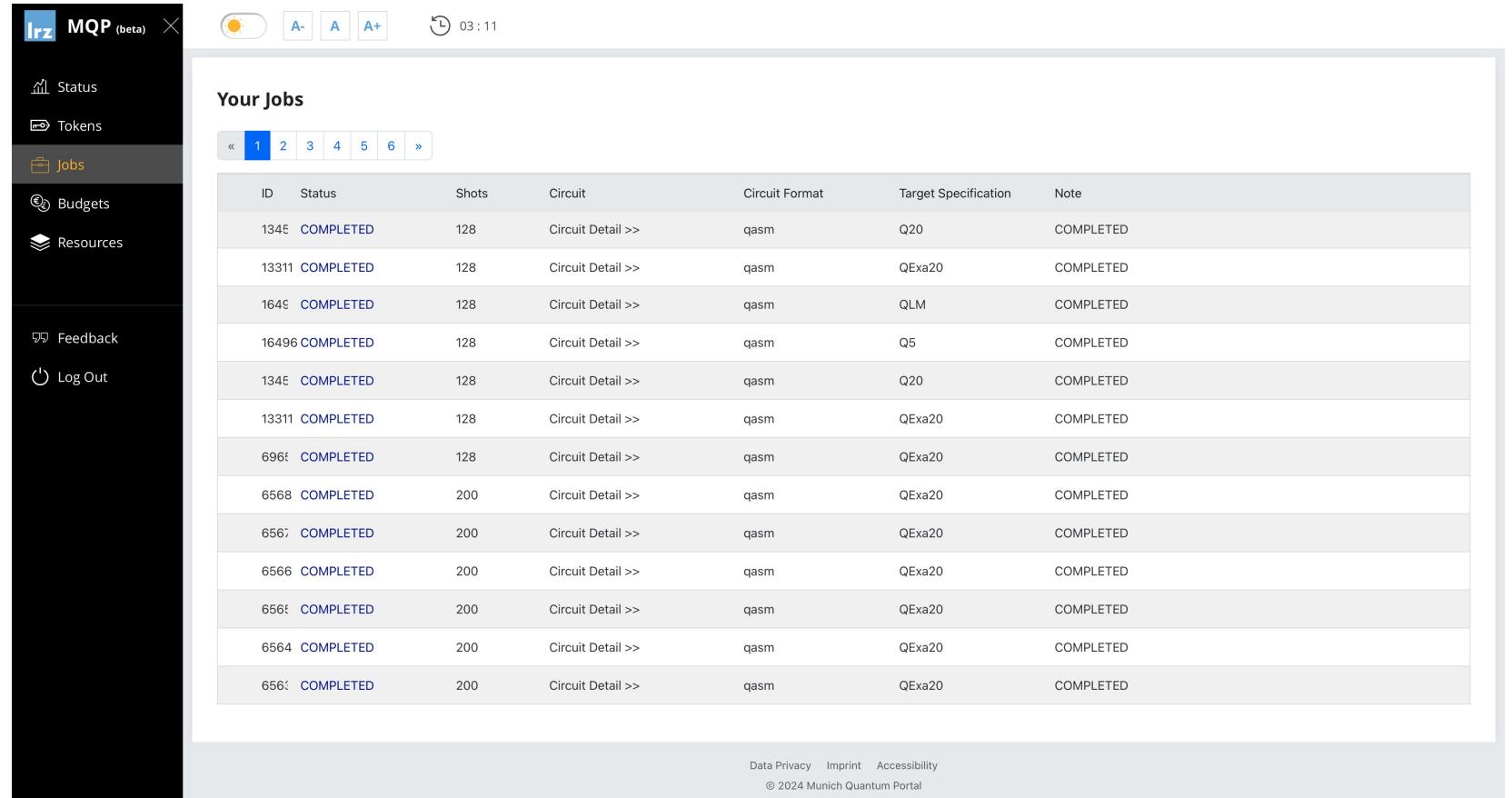


The screenshot shows the MQP (beta) dashboard with the following interface elements:

- Top Bar:** Includes the LRZ logo, MQP (beta) title, system status (yellow sun icon), and zoom controls (A-, A+, A+).
- Header:** Shows the current time as 09:13.
- Sidebar:** Contains links for Status, Tokens, Jobs, Budgets, Resources (highlighted in orange), Feedback, and Log Out.
- Available Resources:** A grid of resource cards:
  - AQT20:** 20-qubit ion trap system at LRZ QIC from AQT. Status: Online. Qubits: 20. Quantum Technology: Ion-Trap.
  - MUNIQC-Atoms20:** 20-qubit neutral atom demonstrator seated at MPO. Status: Offline. Qubits: 20. Quantum Technology: Neutral Atoms.
  - Q20:** R&D 20-qubit QPU at LRZ QIC from IQM. Status: Online. Qubits: 20. Quantum Technology: Superconducting.
  - IQM Q5:** R&D 5-qubit QPU at LRZ QIC from IQM. Status: Online. Qubits: 5. Quantum Technology: Superconducting.
  - QExa20 (highlighted in orange):** 20-qubit superconducting system at LRZ Compute Center from IQM. Status: Online. Qubits: 19. Quantum Technology: Superconducting.
  - QLM:** Quantum systems emulator Captiva. Status: Online. Qubits: 38. Quantum Technology: Simulator.
  - EVIDEN:** Quantum technology demonstrator. Status: Offline. Qubits: 3. Quantum Technology: Superconducting.
  - WM13:** 3-qubit superconducting demonstrator seated at WMI. Status: Offline. Qubits: 3. Quantum Technology: Superconducting.
- Footer:** Includes links for Data Privacy, Imprint, Accessibility, and the copyright notice: © 2024 Munich Quantum Portal.

\*cough\* QExa20 \*cough\*

Once you have submitted  
your job you can come back  
to check its status



The screenshot shows the MQP (beta) dashboard with a sidebar on the left and a main content area on the right.

**Sidebar:**

- Status
- Tokens
- Jobs** (selected)
- Budgets
- Resources
- Feedback
- Log Out

**Main Content Area:**

At the top, there are icons for brightness, font size (A-, A, A+), and a refresh button with the time 03:11.

### Your Jobs

A table listing 12 completed jobs:

ID	Status	Shots	Circuit	Circuit Format	Target Specification	Note
1345	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	Q20	COMPLETED
13311	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
16495	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QLM	COMPLETED
16496	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	Q5	COMPLETED
1345	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	Q20	COMPLETED
13311	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6965	COMPLETED	128	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6568	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6567	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6566	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6565	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6564	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED
6563	COMPLETED	200	<a href="#">Circuit Detail &gt;&gt;</a>	qasm	QExa20	COMPLETED

At the bottom, there are links for Data Privacy, Imprint, Accessibility, and the copyright notice © 2024 Munich Quantum Portal.

# Access Points

Munich Quantum Portal (MQP) Access: MQP Provider

First setup a new environment (preferably Python3.11):

```
$ python3 -m venv QExaTraining  
$ source QExaTraining/bin/activate      OR      $ conda create -n Qexa python=3.11
```

Then install the following packages:

```
$ pip install qiskit-iqm  
$ pip install networkx  
$ pip install matplotlib  
$ pip install pylatexenc  
$ pip install mthree==2.5.1  
$ pip install stim  
$ pip install mqp-qiskit-provider
```



# Access Points

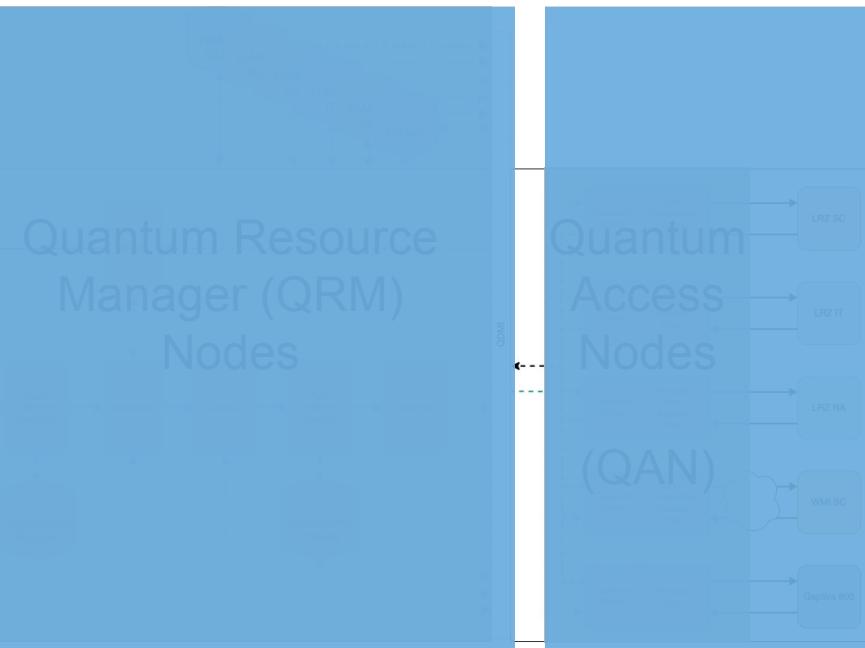
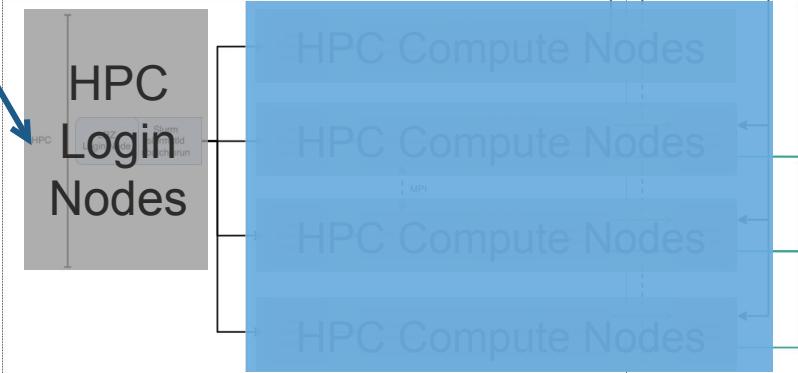
HPCQC Access

HPCQC  
Users



ssh <to-HPC-  
login-node>

- 1) Access HPC login nodes
- 2) Your application
- 3) Prepare SLURM-job description file
  - Specify computation resources, e.g., CPU, GPU, QPU
  - Specify the command to run your application
- 4) Submit Job
- 5) Wait for resource allocation & Get back results



## Log into an HPC cluster enabling quantum job submission

- Access Linux Cluster at LRZ

**ssh -Y lxlogin2.lrz.de -l <lrz-account>**

*(lrz-account@lxlogin2.lrz.de) Password:*

*(lrz-account@lxlogin2.lrz.de) !!! 2FA !!!*

*All LRZ cluster login nodes require now a second factor for authentication!*

*Please refer to <https://doku.lrz.de/x/EgDxBw>*

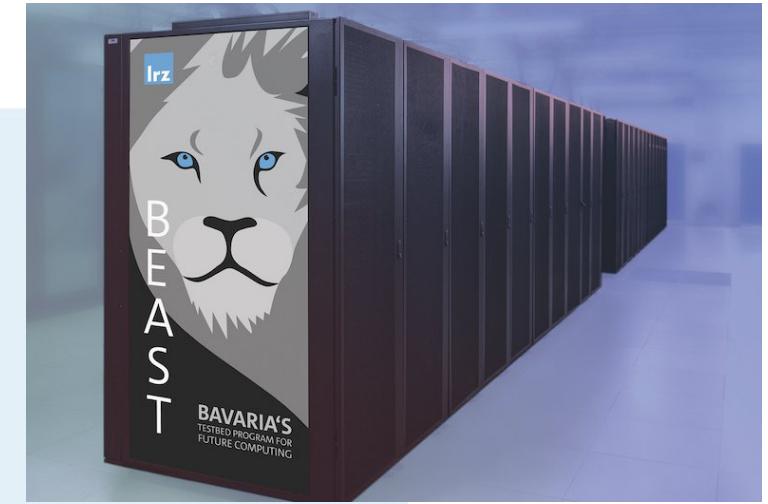
*Note for using push tokens:*

*Without PIN directly press enter upon the 2FA prompt to receive the push token!*

*Token\_Response:*

- Access BEAST system from Linux Cluster and Wolpertinger Node1 to submit jobs

**ssh -A -J lrz-account@gw.beast.lrz.de lrz-account@wolpy01**



### BEAST Testbed System at LRZ

- BEAST Gateway (Login)
- HPCQC Wolpertinger Nodes

## HPCQC programming models

1. In Python: qiskit with HPCQC Provider
  
2. In C: Quantum Programming Interface (QPI)

## 1. Using qiskit and HPCQC Provider | Example: **ghz\_circuit.py**

```
from qiskit import ClassicalRegister, QuantumRegister
from qiskit import QuantumCircuit
from hpc_offload_provider import HPCOffloadProvider
if __name__ == "__main__":
    ...
provider = HPCOffloadProvider()
backend = provider.get_backend("QExa20")
n = 20 # num_qubits
# define the circuit
q = QuantumRegister(n)
c = ClassicalRegister(n)
ghz_circ = QuantumCircuit(q, c)
ghz_circ.h(0)
for i in range(num_qubits - 1):
    ghz_circ.cx(i, i+1)
    ...
}
```

} import HPC-QC Offload Provider package at LRZ

} specify provider and backend

} your circuit

## 1. Using qiskit and HPCQC Provider | SLURM job description file

```
#!/bin/sh
#SBATCH -J ghz_qexa20
#SBATCH -p wolpy
#SBATCH -o ./%x_%j.out
#SBATCH -e ./%x_%j.err
#SBATCH --ntasks=1

## -----
module use -p /home/sw/qis/wolpy/hpcqc-software/modules/linux-rocky9-
icelake/
module load python/3.11.7-gcc-11.4.1-o75q74w
module load py-qiskit
module load py-hpc-offload-provider
## -----


## -----
/bin/sh /home/sw/qis/wolpy/mqss/scripts/load-mqss.sh
## -----


python ghz_circuit.py
## -----


/bin/sh /home/sw/qis/wolpy/mqss/scripts/unload-mqss.sh
## -----
```

• job name  
• partition  
• output file  
• error log file  
• ...

• use „*module*“ to load related „*installed*“ packages  
• do not need to load them if you use your local env  
• **SHOULD** load „*py-hpc-offload-provider*“ to use the real quantum-device backends at LRZ

load MQSS to make connection with quantum devices

your application/code/executable file

unload MQSS

## 1. Using qiskit and HPCQC Provider | Wait for resource allocation & Get back results

```
[di35hef@wolpy01 ghz]$ sbatch submit_ghz_qexa20.cmd
Submitted batch job 37701
[di35hef@wolpy01 ghz]$ squeue
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
      37701    wolpy  ghz_qexa  di35hef R      0:00      1 wolpy01
[di35hef@wolpy01 ghz]$ ls
ghz_circuit.py          ghz_qexa20_37025.out  ghz_qexa20_37701.out  submit_ghz_q5.cmd
ghz_qexa20_37025.err    ghz_qexa20_37701.err  submit_ghz_q20.cmd   submit_ghz_qexa20.cmd
[di35hef@wolpy01 ghz]$ █
```

ghz\_qexa20\_37701.err  
ghz\_qexa20\_37701.out

-----  
*Measured results:*  
`{'00000': 428, '11111': 393, '11110': 14, '00111': 19, '10000': 14, '01000': 2, '01111': 15, '11000': 22, '11101': 14, '00011': 19, '00001': 13, '10111': 15, '11100': 23, '00101': 7, '00100': 7, '01011': 3, '00110': 2, '11011': 7, '10011': 3, '00010': 2, '01100': 1, '11010': 1}`  
-----

## 2. Using Quantum Programming Interface (QPI) | Example: `qpi_bellstates.c`

```
#include "qpi.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// -----
// Define bellstate function
// -----

int bell_zero(){
    QCircuit circuit;
    QClassicalRegisters MyClassicalRegisters;
    Qubit qubitZero = 0;
    Qubit qubitOne = 1;

    int numberOfShoots = 1024;
    qCircuitBegin(&circuit);
    qInitClassicalRegisters(&MyClassicalRegisters, 2);
    qH(qubitZero);
    qCX(qubitZero, qubitOne);
    qMeasure(qubitZero, MyClassicalRegisters, 0);
    qMeasure(qubitOne, MyClassicalRegisters, 1);
    qCircuitEnd();
```

Define the circuit, gates with QPI

```
int isErr = qExecute(circuit, numberOfShoots);
```

Execute the circuit

```
if(!isErr) {
    int *Results = qRead(circuit);
    for (int i = 0; i < 1 << qCircuitWidth(circuit); i++){
        printf("%d %d\n", i, Results[i]);
    }
}

qCircuitFree(circuit);
return 42;

// -----
// Main function
// -----
int main() {
    int err = 0;
    err = bell_zero();
    return err;
}
```

More information about QPI, please contact  
Ercüment Kaya, ercument.kaya@lrz.de

## 2. Using Quantum Programming Interface (QPI) | SLURM job description file

```
#!/bin/sh
#SBATCH -J qpi_bellstate
#SBATCH -p wolpy
#SBATCH -o ./%x_%j.out
#SBATCH -e ./%x_%j.err
#SBATCH --ntasks=1
```

```
## -----
/bin/sh /home/sw/qis/wolpy/mqss/scripts/load-mqss.sh
##
module use -p /home/sw/qis/wolpy/hpcqc-
software/modules/linux-rocky9-icelake/
module load quantum-programming-interface
##
./build/qpi_bellstates
##
##
/bin/sh /home/sw/qis/wolpy/mqss/scripts/unload-mqss.sh
##
```

SLURM job  
description file

```
[di35hef@wolpy01 qpi_bellstate]$ sbatch submit_qexa20_qpi_bellstate.cmd
Submitted batch job 37702
[di35hef@wolpy01 qpi_bellstate]$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
37702 wolpy qpi_bell di35hef R 0:02 1 wolpy01
[di35hef@wolpy01 qpi_bellstate]$
```

Wait for resource allocation & Get back the results