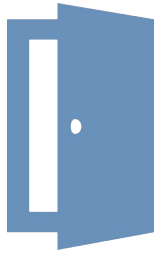
Leibniz Supercomputing Centre

of the Bavarian Academy of Sciences and Humanities

General LRZ House Information



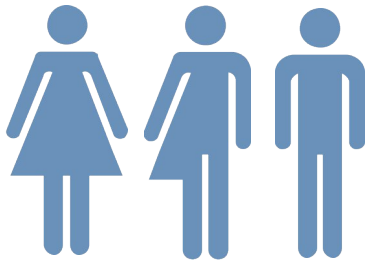
Emergency Exit

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)

Building Evacuation Route and Meeting Point

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 / Jobscript
    - Compile and run using QPI and Qiskit

Introduction

LRZ’s full stack

Applications

Programming API(s)

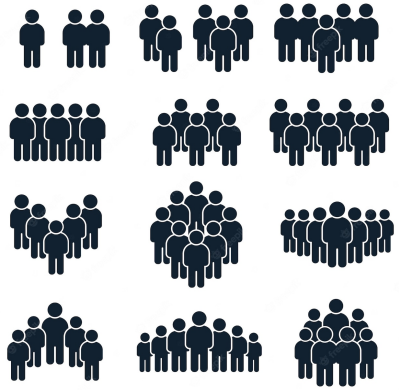
Munich Quantum Software Stack

Feedback / FoMaC

Backend API

Many/Different
Quantum Devices

Wide User Communities



Hybrid/Quantum Programming Toolkits / LIbraries

Web-Portal Access

HPCQC Access with Quantum Acceleration

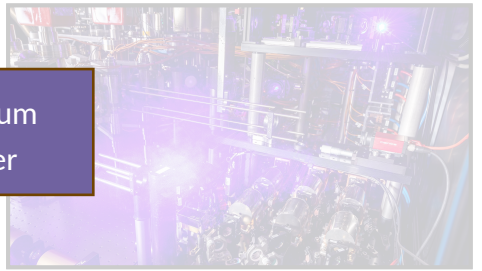
Quantum Compiler

Optimizers
Schedulers
Transpilers
Mappers

Quantum Server

Quantum Server

Quantum Server



Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Munich Quantum Software Stack: Separation of Concerns

Wide User Communities

Hybrid/Quantum Programming Toolkits / Libraries

Web-Portal Access

HPCQC Access with Quantum Acceleration

Cutting and Scheduling Resource Management

Figures of Merit and Constraints

Quantum Compiler

Optimizers
Verifiers
Transpilers
Mappers

Quantum Device Management Interface

QDMI Backend

QDMI Backend

QDMI Backend

Enabling Domain User Communities to Compute on Quantum Devices

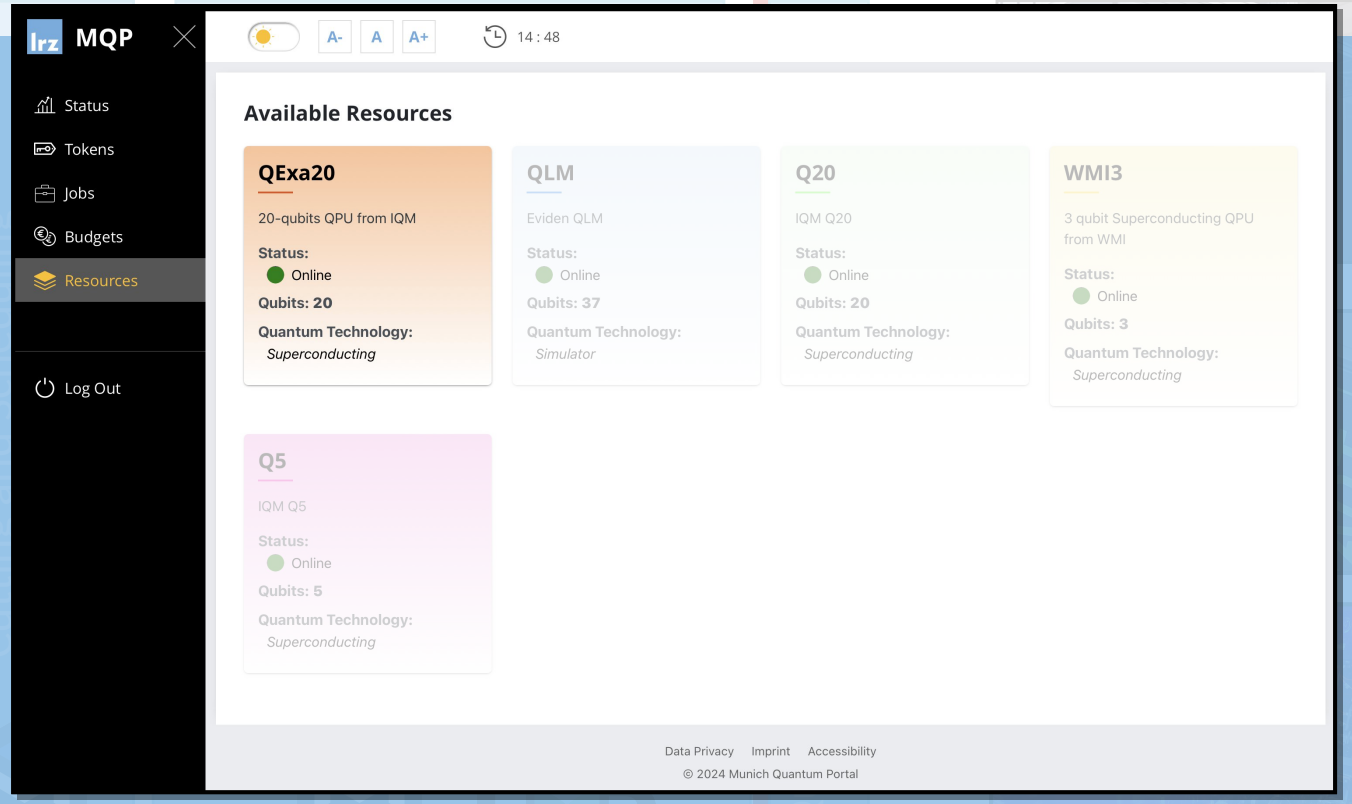
Many/Different Quantum Devices

Web-Portal Access

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

The end users can track:

- Status of submitted jobs
- Available resources



Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Web-Portal Access

Allows access programatically directly from Qiskit to the quantum resources

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

```python
from qiskit import QuantumCircuit
from bavarian_quantum_portal_provider import BQPProvider

token = 'wClYUWUR2WI9...xcspQLni3vEbqMHM3DdNkqgLWN0VOIcnaTMDvxZ53H5mMFix'
provider = BQPProvider(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]}")
```

More performant than pythonic programming interfaces

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

```c
#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);
}
```

Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Introduction
HPCQC Access: HPC Offload Provider

Wide User Communities

Web-Portal Access

Figures of Merit an Constraints

QPI's pythonic counterpart

More familiar interface for many researchers

HPCQC Access with Quantum Acceleration

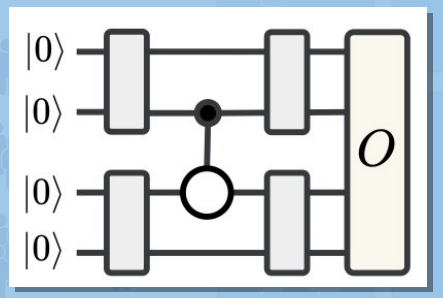
Tailored for HPC environments offloading quantum circuits to the middle-end

```python
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

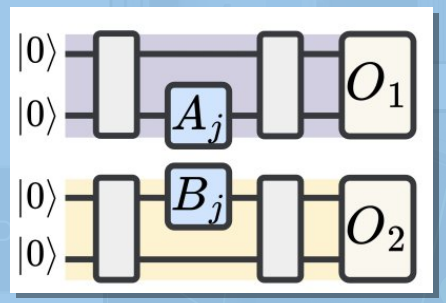
Many/Different Quantum Devices

Wide User Communities



Cutting and Scheduling Resource Management

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



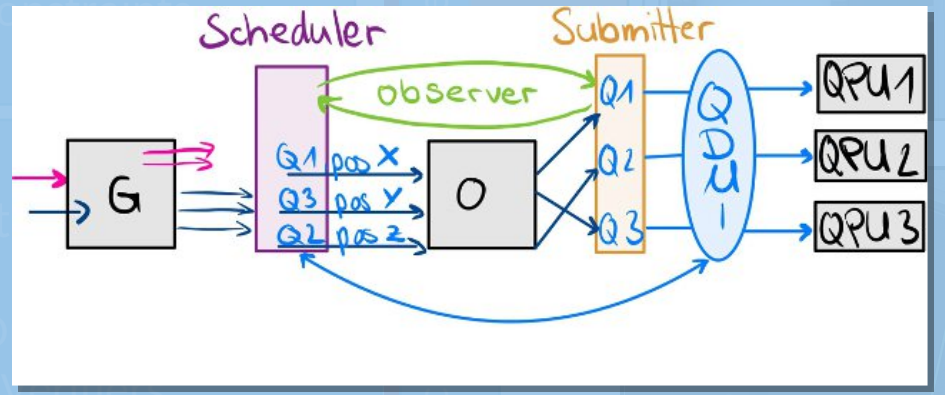
Enabling Domain User Communities to Compute on Quantum Devices

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

Web-Portal Access

Figures of Merit and Constraints

Quantum Device Management Interface

QDMI Back

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

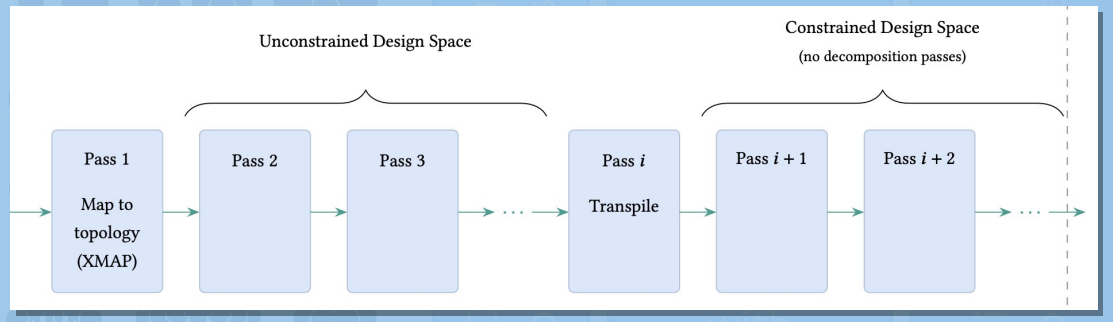
Quantum Compiler

Optimizers
Verifiers
Transpilers
Mappers

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



Quantum Compiler

Optimizers
Verifiers
Transpilers
Mappers

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

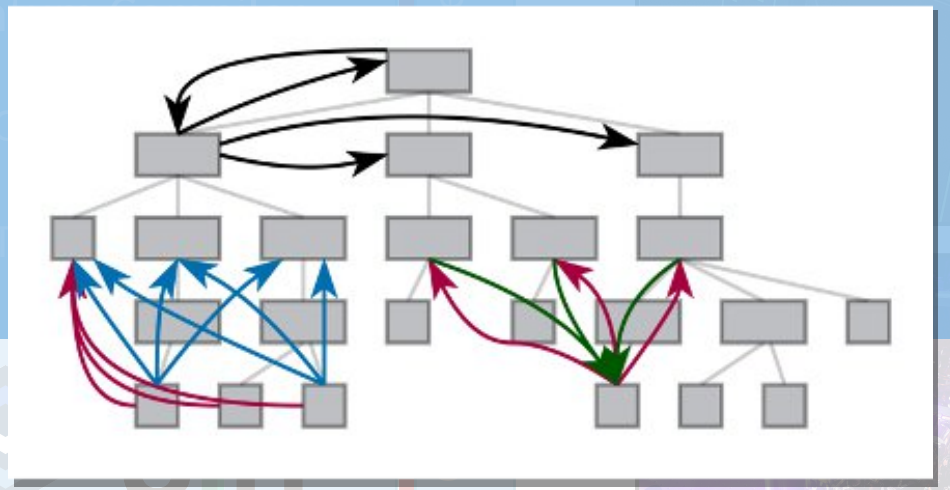
Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Figures of Merit and Constraints

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



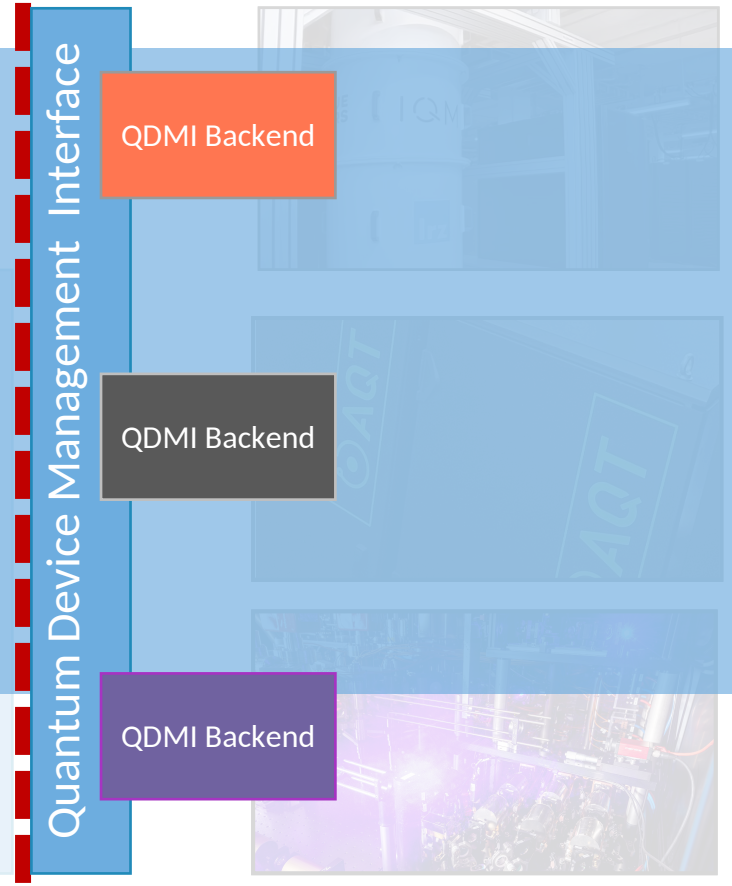
Enabling Domain User Communities to Compute on Quantum Devices

Many/Different Quantum Devices

Introduction
Quantum Device Management Interface (QDMI)

```c
void compile_and_run(Module *qirmod, 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(qirmod);
    QRM_specific_selector(selector_s, pass);
    pass = NULL;
    while(pass = next_pass(pass))
        pass->run(dev, qirmod);
    QDMI_control_pack_qir(qirmod, &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

Hybrid/Quantum Programming Toolkits / Libraries

Web-Portal Access

HPCQC Access with Quantum Acceleration

Cutting and Scheduling Resource Management

Figures of Merit and Constraints

Quantum Compiler

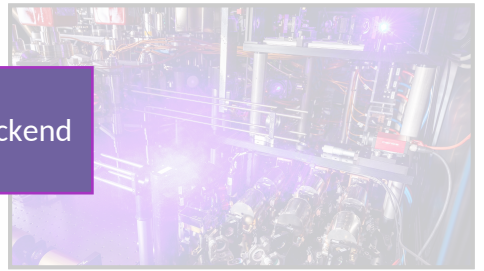
Optimizers
Verifiers
Transpilers
Mappers

Quantum Device Management Interface

QDMI Backend

QDMI Backend

QDMI Backend



Enabling Domain User Communities to Compute on Quantum Devices

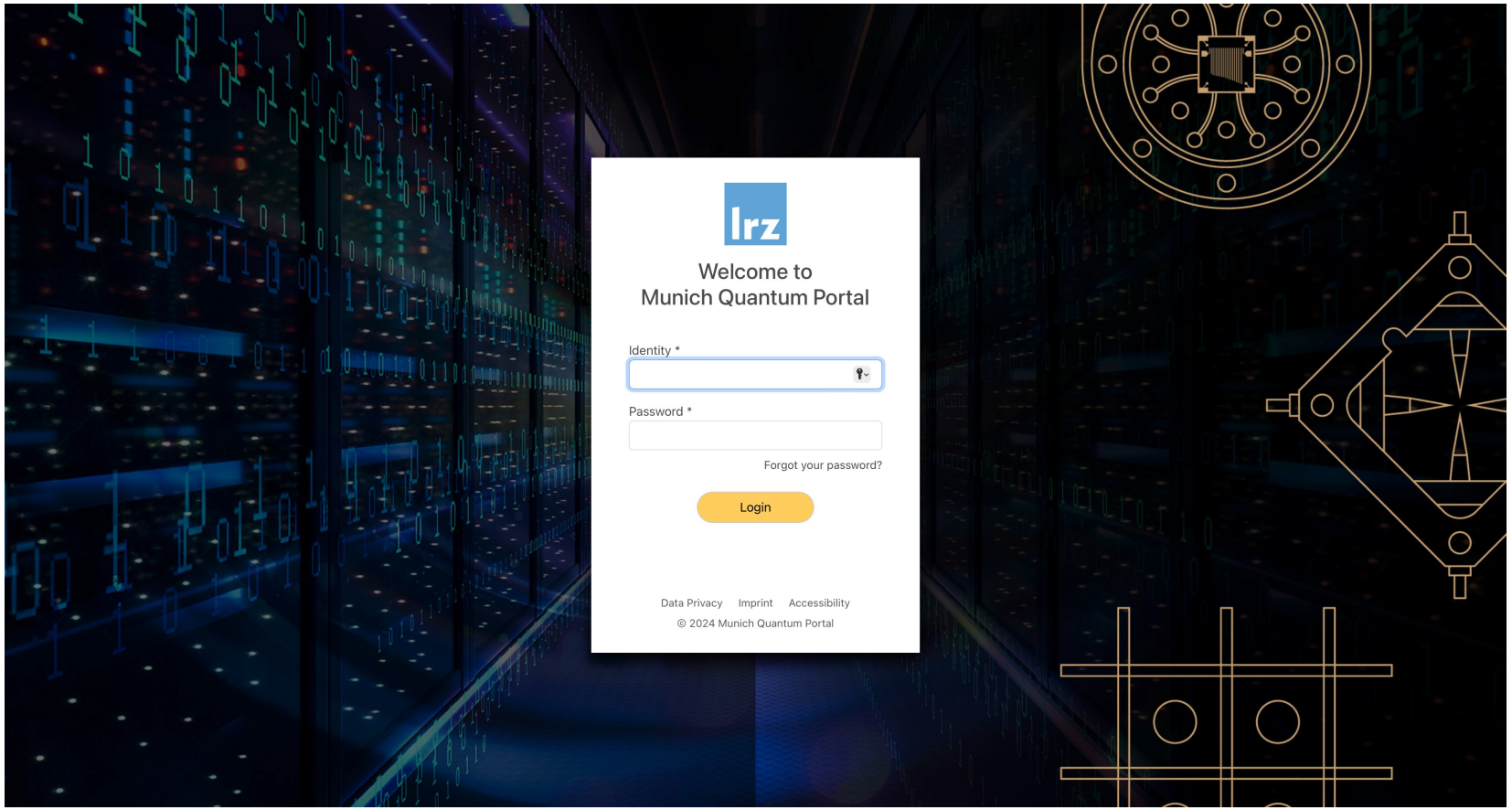
Many/Different Quantum Devices

Access Points

Munich Quantum Portal (MQP) Access: Dashboard

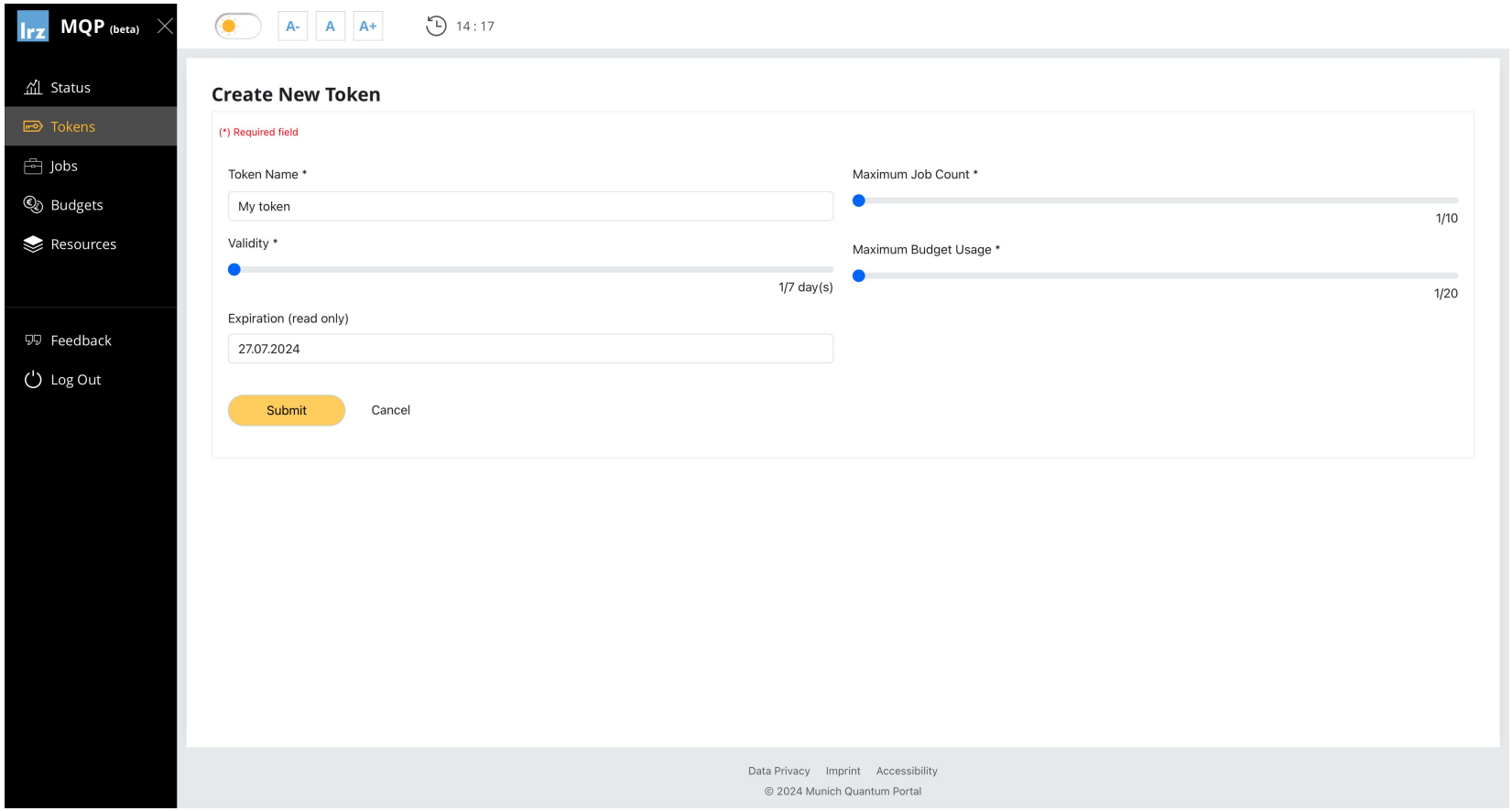
Access Points
Munich Quantum Portal (MQP) Access: Dashboard

Visit `https://portal.quantum.lrz.de`

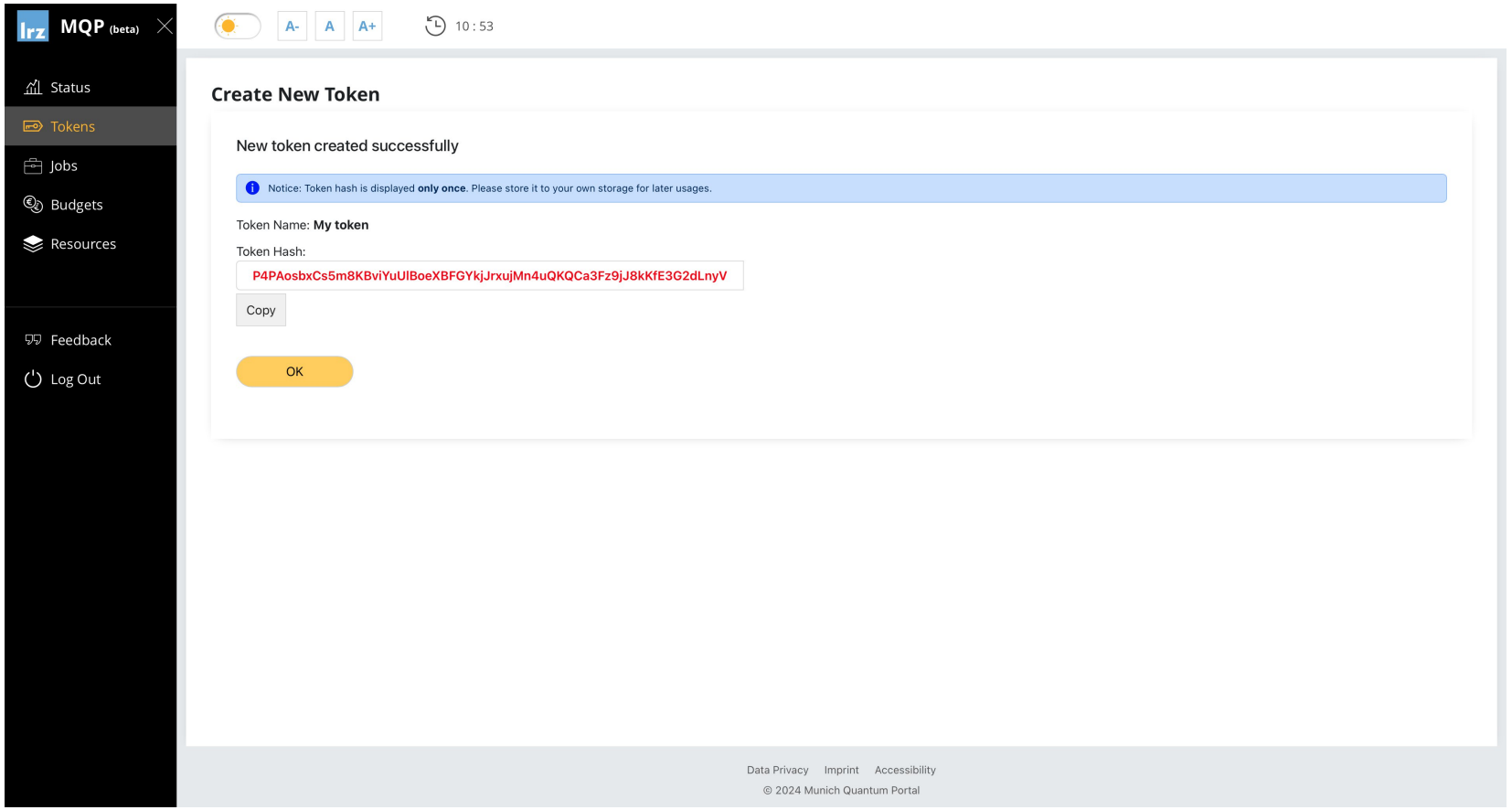


Authentication to the web frontend is compatible with the existing LRZ Active Directory / LDAP infrastructure, enabling users to use their preexisting university-affiliated accounts to log in

Create a new token



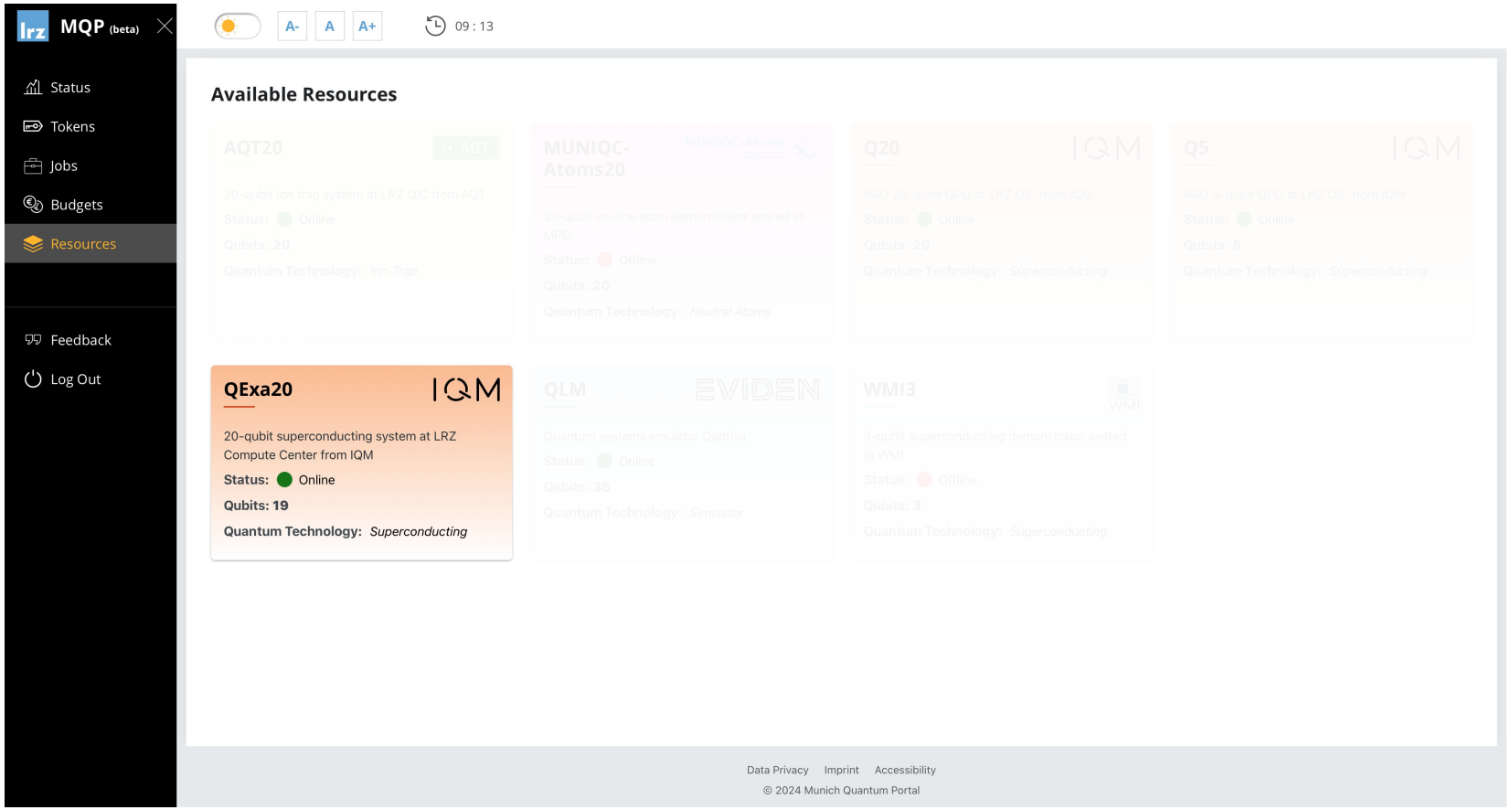
Create a new token



Remember to keep it somewhere safe!

Access Points
Munich Quantum Portal (MQP) Access: Dashboard

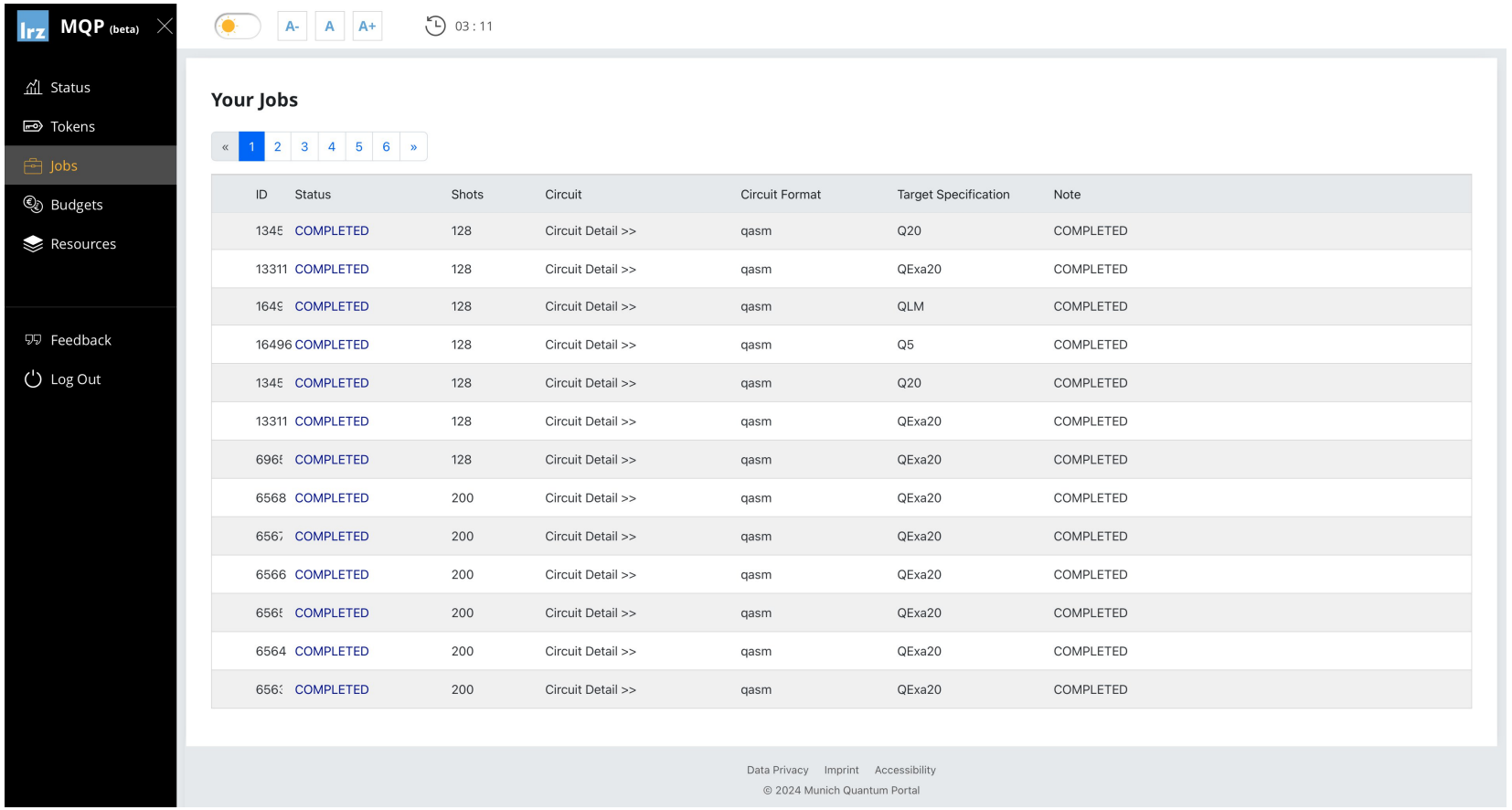
Check the status of your target accelerator



cough QExa20 *cough*

Access Points
Munich Quantum Portal (MQP) Access: Dashboard

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



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

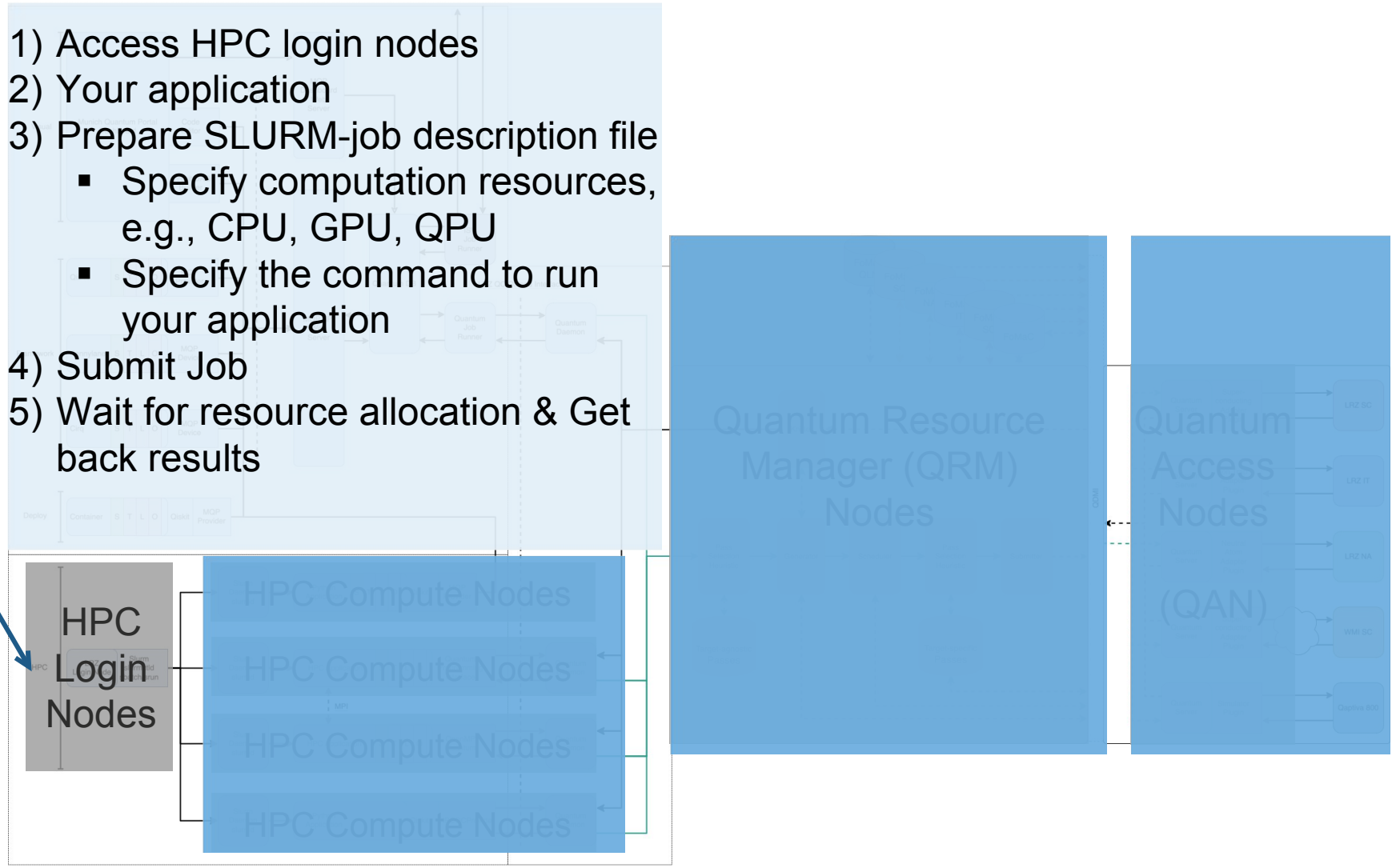
HPCQC Access

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 !!! 2FA !!! 2FA !!! 2FA !!! 2FA !!! 2FA !!! 2FA !!! 2FA !!! 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 desciption 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**

```c
#include "qpi.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
// ---------------------------------
// Define bellstate function
// ---------------------------------
int bell_zero(){
  QCircuit circuit;
  QClassicalRegisters MyClassicalRegisters;
  Qqubit qubitZero = 0;
  Qqubit qubitOne = 1;

  int numberOfShoots = 1024;

  qCircuitBegin(&circuit);
  qInitClassicalRegisters(&MyClassicalRegisters, 2);
  qH(qubitZero);
  qCX(qubitZero, qubitOne);
  qMeasure(qubitZero, MyClassicalRegisters, 0);
  qMeasure(qubitOne, MyClassicalRegisters, 1);
  qCircuitEnd();

  int isErr = qExecute(circuit, numberOfShoots);

  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;
}
```

Define the circuit, gates with QPI

Execute the circuit

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

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

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
## ----------------------------------------------
```

```
[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