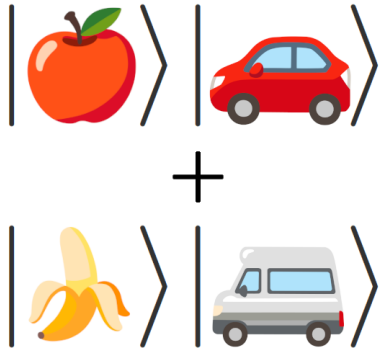
Fraunhofer FOKUS

Fraunhofer-Institut für Offene Kommunikationssysteme

Jasp – Real-time computations with ECLIPSE qrisp

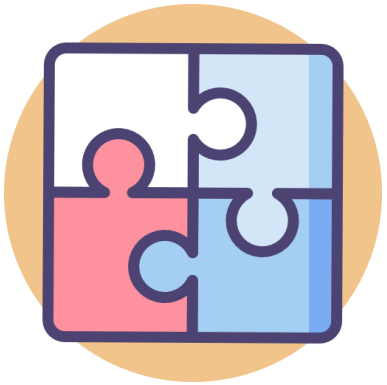
The Qrisp Team

Features in Qrisp

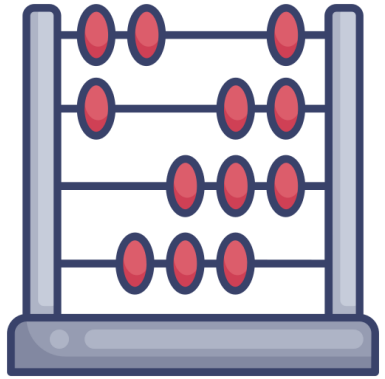


QuantumVariables

Automatic Uncomputation



Algorithms



Arithmetic

Multiplying two QuantumFloats…

Qiskit

```
from qiskit import (QuantumCircuit, QuantumRegister,
ClassicalRegister, transpile)
from qiskit_aer import Aer
from qiskit.circuit.library import RGQFTMultiplier
n = 6
a = QuantumRegister(n)
b = QuantumRegister(n)
res = QuantumRegister(2*n)
cl_res = ClassicalRegister(2*n)
qc = QuantumCircuit(a, b, res, cl_res)
for i in range(len(a)):
        if 3 & 1<<i: qc.x(a[i])
for i in range(len(b)):
        if 4 & 1<<i: qc.x(b[i])
qc.append(RGQFTMultiplier(n, 2*n),
list(a) + list(b) + list(res))
qc.measure(res, cl_res)
backend = Aer.get_backend('qasm_simulator')
qc = transpile(qc, backend)
counts_dic = backend.run(qc).result().get_counts()
print({int(k, 2) : v for k, v in counts_dic.items()})
#Yields: {12: 1024}
```

qrisp

```
from qrisp import QuantumFloat
n = 6
a = QuantumFloat(n)
b = QuantumFloat(n)
a[:] = 3
b[:] = 4
res = a*b
print(res)
#Yields: {12: 1.0}
```


Fraunhofer FOKUS

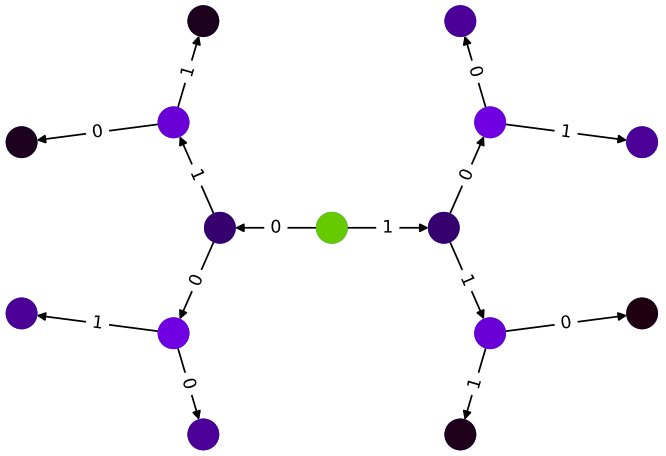
Shor‘s Algorithm in 11 lines of code

Quantum Subroutine **find_order**

```
def find_order(a, N):
    qg = QuantumModulus(N)
    qg[:] = 1
    qpe_res = QuantumFloat(2 * qg.size + 1, exponent=-(2 * qg.size + 1))
    h(qpe_res)
    for i in range(len(qpe_res)):
        with control(qpe_res[i]):
            qg *= a
            a = (a * a) % N
    QFT(qpe_res, inv=True)
    return qpe_res.get_measurement()
```

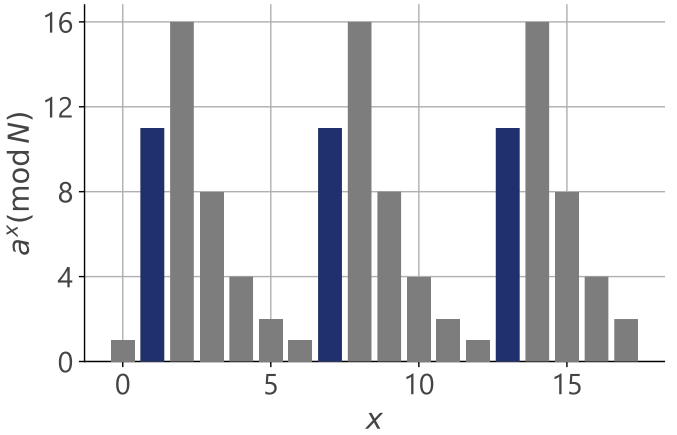
The rest is classical processing!

Algorithms in Qrisp



Quantum Backtracking

Hamiltonian Simulation



Shor`s Algorithm

VQE, QAOA, LCU, HHL, RTC, …

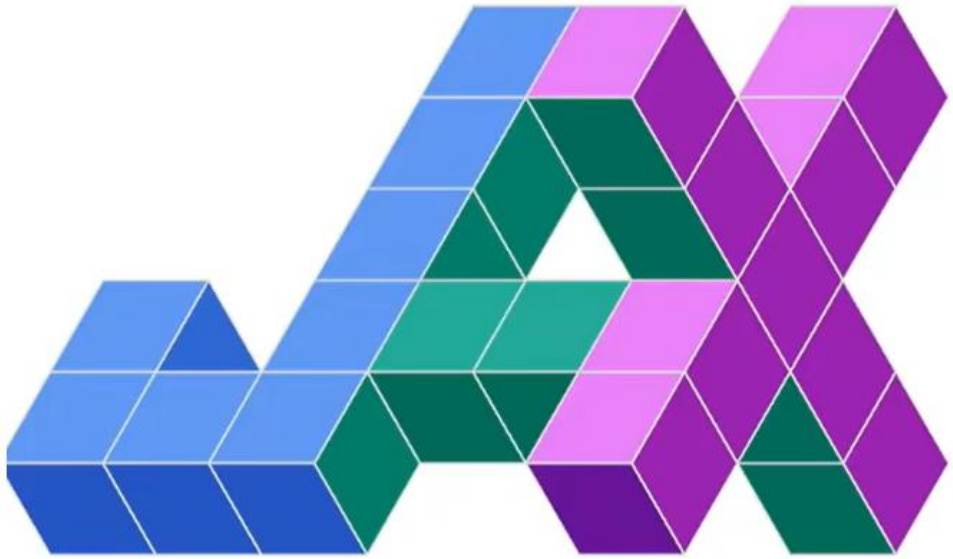
Fraunhofer FOKUS

JASP

- Qrisp uses Python, which unfortunately comes with compilation issues

- **Solution**: Interpreter Tracing with *Google's JAX* framework

- We send symbolic values („Tracers“) through the code, that behave scale-invariant



Compliation Pipeline

- We use the intermediate representation **Jaxpr**, which is compilable to binary
- We can compile to QIR with Catalyst!

- Writing a JASP program is similar to writing a JAX program
- Key insight is the distinction between static and dynamic values!

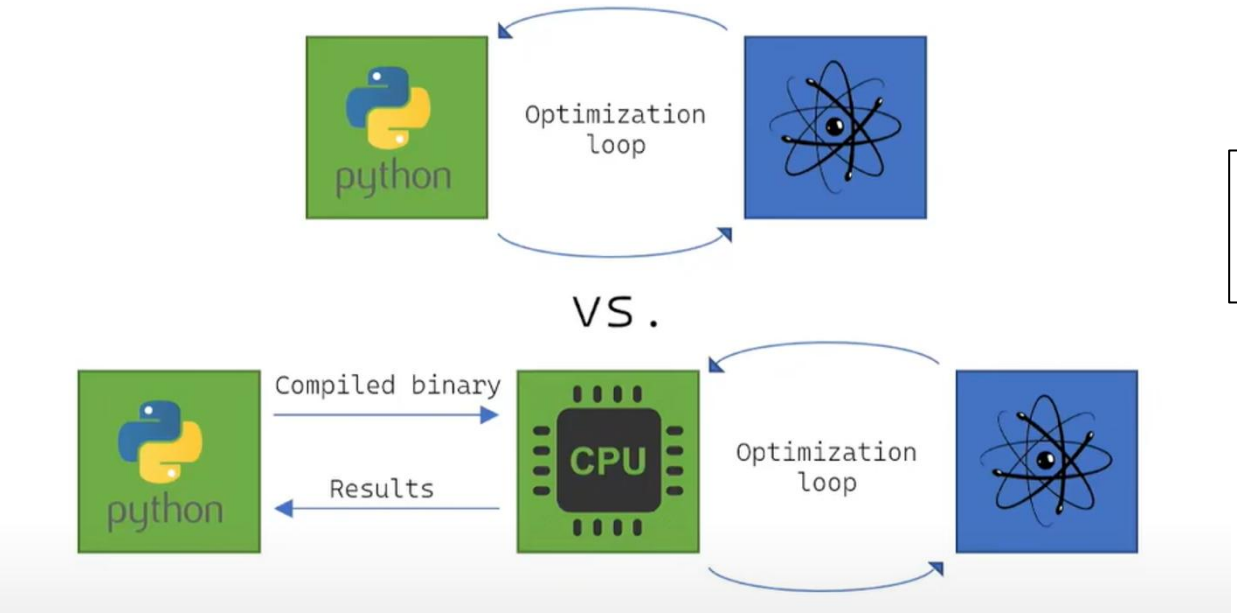
```
from qrisp import *

def main(i):
    qf = QuantumFloat(i)
    h(qf[0])
    cx(qf[0], qf[1])
    meas_float = measure(qf)
    return meas_float

jaspr = make_jaspr(main)(5)
print(jaspr)
```

```
{ lambda ; a:QuantumCircuit b:i64[]. let
    c:QuantumCircuit d:QubitArray = jasp.create_qubits a b
    e:Qubit = jasp.get_qubit d 0
    f:QuantumCircuit = jasp.h c e
    g:Qubit = jasp.get_qubit d 1
    h:QuantumCircuit = jasp.cx f e g
    i:QuantumCircuit j:i64[] = jasp.measure h d
    k:QuantumCircuit = jasp.reset i d
    l:QuantumCircuit = jasp.delete_qubits k d
  in (l, j) }
```

- Improved hybrid-algorithm optimization loop
- Real-time computation within the JAX infrastructure
- And much more!



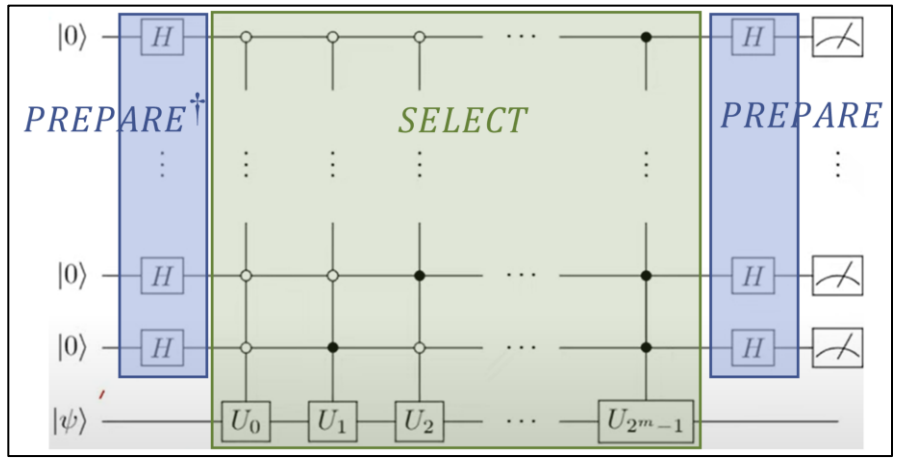

```
with control(some_bool):
```

```
@qache
def my_func():
```

```
@RUS
def my_func():
```

```
@terminal_sampling
def my_func():
```

LCU with Jasp and RUS




```python
def inner_LCU(operand_prep, state_prep, unitaries, num_unitaries):

    operand = operand_prep()
    num_unitaries = len(unitaries)

    # Specify the QuantumVariable that indicates which case to execute
    n = np.log2(num_unitaries)
    case_indicator = QuantumFloat(n)

    # LCU protocol with conjugate preparation
    def LCU_state_prep(case_indicator, operand):
        with conjugate(state_prep)(case_indicator):
            qswitch(operand, case_indicator, unitaries)
```

```python
@RUS
def LCU(operand_prep, state_prep, unitaries, num_unitaries):

    case_indicator, qv = inner_LCU(
        operand_prep, state_prep, unitaries, num_unitaries)

    # Success condition
    success_bool = measure(case_indicator) == 0
    return success_bool, qv
```


Fraunhofer FOKUS

pip install qrisp

you suddenly feel the urge to pip install qrisp

Qrisp is Open-Source

https://github.com/eclipse-qrisp/Qrisp

Contact

Prof. Dr. Nikolay Tcholtchev
Business Unit SQC
Tel. +49 (0)30 3463 7175
nikolay.tcholtchev@fokus.fraunhofer.de

Sebastian Bock
Business Unit SQC
Tel. +49 (0)30 3463 7360
sebastian.bock@fokus.fraunhofer.de

Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin
www.fokus.fraunhofer.de