LibKet: Cross-Platform Library for Running Quantum Algorithms on NISQ processors

LibKet – The Basics

IEEE Quantum Week 2022

September 18-23, 2022

Matthias Möller[1] and Carmen G. Almudever[2]

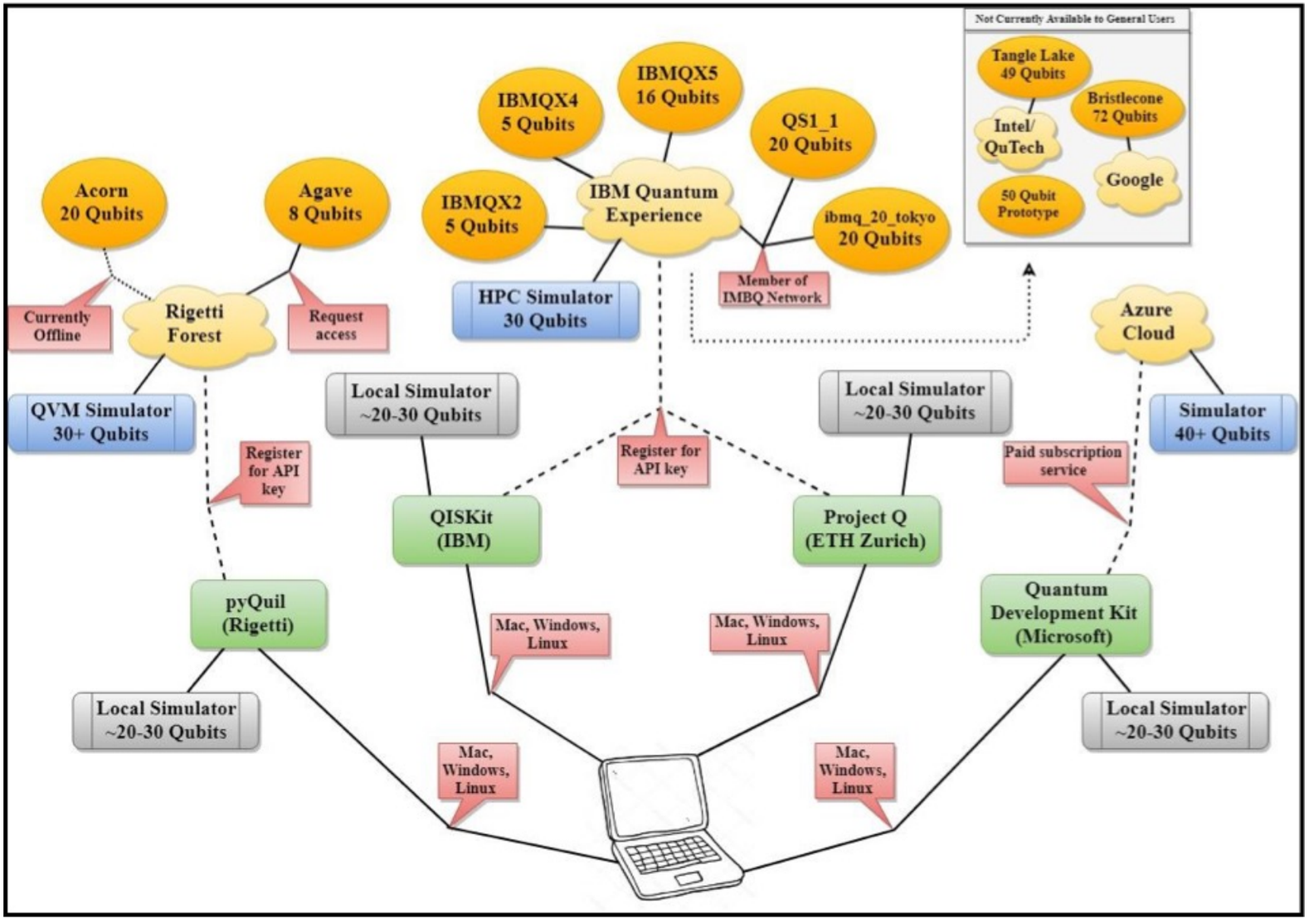
[1]Delft University of Technical (m.moller@tudelft.nl)

[2]Technical University of Valencia (cargara2@disca.upv.es)

Lessons learned from classical HPC

- Standardized programming languages and libraries
  - C/C++/Fortran
  - MPI/OpenMP
  - CUDA/OpenACC/HiP
- Flexibility in coding
  - Write-once-run-anywhere (e.g., C++ standard library)
  - Easy integration of hand-crafted routines (e.g., C++ intrinsics)
- Application-inspired benchmarking
  - HPL (High-performance LINPACK) -> TOP500 supercomputers
  - SPEC ACCEL, SPEChpc, SPEC MPI, SPEC OMP, …

Quantum computing today



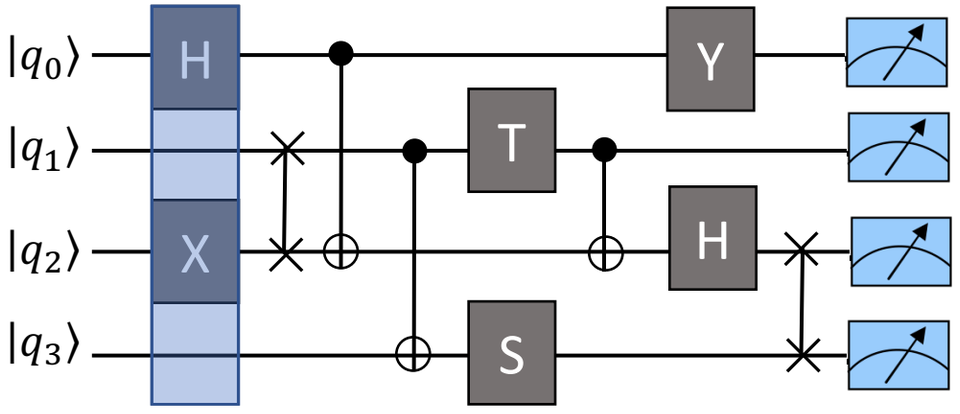
LaRose: Overview and Comparison of Gate Level Quantum Software Platforms, arXiv:1807.02500, 2018

LibKet – more than yet another quantum programming library

- Unified quantum programming interface in C++17 that
  - … allows to write code once and run it anywhere
  - … specializes to vendor-specific toolchains under the hood
  - … enables injection of vendor-specific code
  - … adopts concepts and terminology from CUDA

- LibKet is not alone
  - XACC – eXtreme-scale Accelerator programming framework
  - QODA – NVIDIA quantum optimized device architecture
  - Qunity, Silq, … – formal language definitions not yet SDKs
  - …

LibKet's two coding philosophies



Quantum program

```
H q0 | X q2
SWAP q1, q2
CNOT q0, q2
CNOT q1, q3
T q1 | S q3
```

Quantum expression

```
CNOT(q1, q3,
  CNOT(q0, q2,
    SWAP(q1, q2,
      H(q0, …) | X(q2, …)
        )
      )
    )
```

LibKet's execution philosophy

1. Create hardware-agnostic quantum program/expression

   QProgram prog / auto expr = …

2. Create quantum device and load program/expression

   QDevice<backend, #qubits> device(prog/expr)

3. Execute quantum kernel on the quantum device

   result = device.eval(#shots, …, stream)

   job = device.execute(#shots, …, stream)

   job = device.execute_async(#shots, …, stream)

   job.[wait(), query(), get(), run()]

https://tinyurl.com/3vw4zdc8

Tutorial at IEEE QCE22, September 18-23, 2022

LibKet: A Cross-Platform Library for Running Quantum Algorithms on NISQ Processors

Organizers: Carmen G. Almudever, Matthias Möller

Session 1: Sunday, September 18, 10:00 AM – 11:30 AM MDT (UTC-6)

Time	Content	Lecturer	Slides	Binder
10:00-11:00 am	Hands-on Introduction to Quantum Computing	Carmen	slides	tutorial 01
11:00-11:30 am	Libket - The Basics	Matthias	slides	tutorial 02

Session 2: Sunday, September 18, 12:00 AM – 1:30 PM MDT (UTC-6)

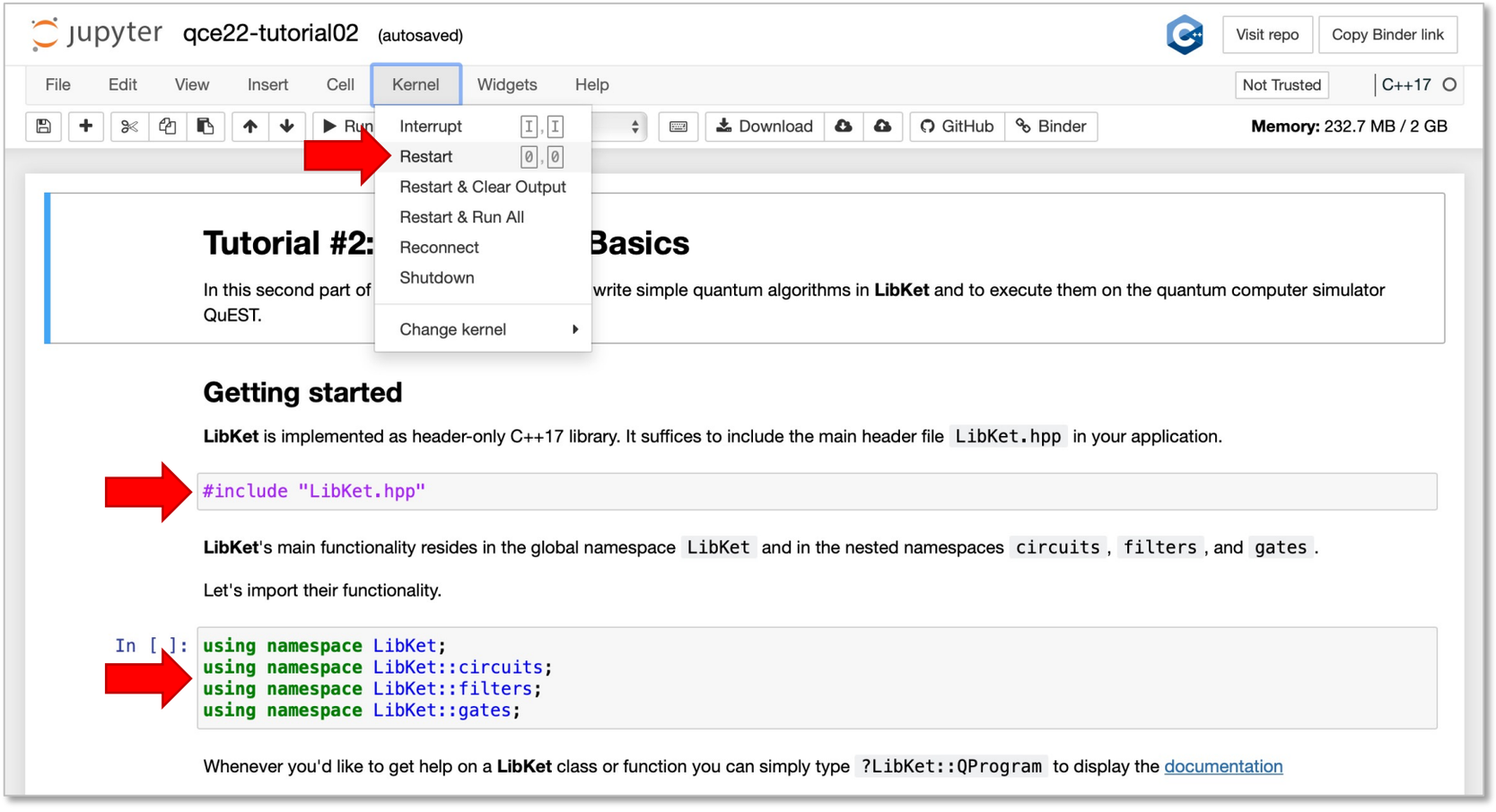
Time	Content	Lecturer	Slides	Binder
1:00-1:45 pm	LibKet - Advanced Features	Matthias	slides	tutorial 03
1:45-2:30 pm	Variational Quantum Algorithms	Carmen/Matthias	slides	tutorial 04

Disclaimer

- The following tutorials use xeus-cling to *interpret* C++ code in a jupyter notebook environment.
- This is meant for educational purposes only.
- Execution of the code is slower than normal.
- Asynchronous execution is not supported.
- Some functionality of LibKet does not work.

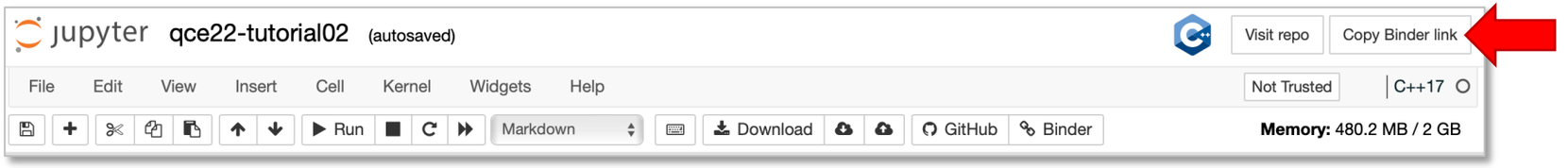
If things go wrong

- **Restart the kernel** and execute the code blocks you need but don’t forget to include the library and namespaces



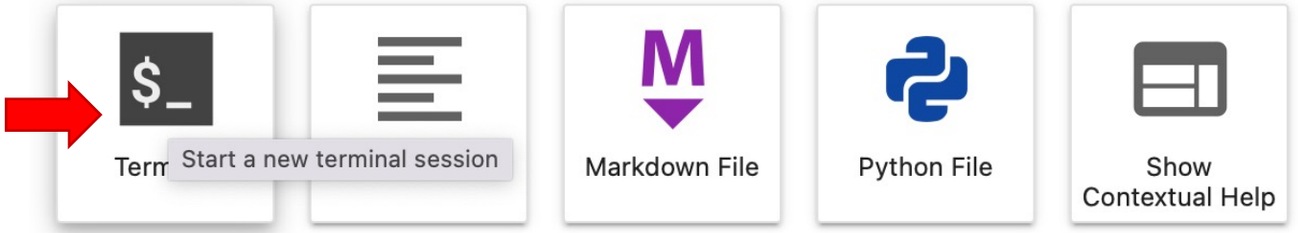
If things go really wrong

1. Copy the binder link



2. Insert the binder link into a new browser tab

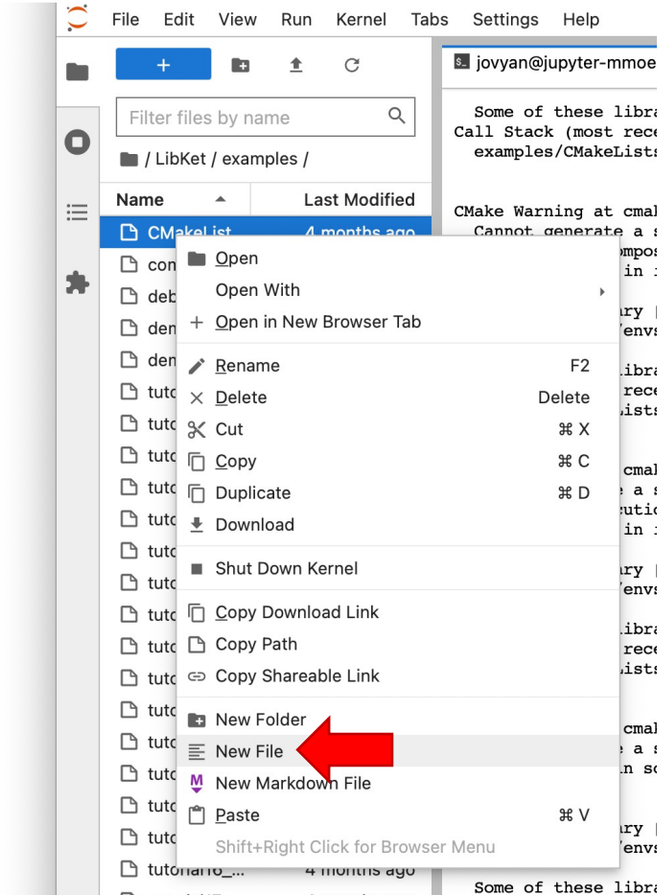
3. Start a new Terminal session



If things go really wrong

3. Run
   cd LibKet/build
   cmake . -DLIBKET_BUILD_EXAMPLES=ON

4. Create a new C++ files in *LibKet/examples*



5. Open the file and add
   #include <LibKet.hpp>
   #include <iostream>

   using namespace LibKet;
   using namespace LibKet::circuits;
   using namespace LibKet::filters;
   using namespace LibKet::gates;

   int main() { your code goes here }

If things go really wrong

5. Compile the source file
   cmake .
   make demo01
6. Run the compiled executable
   ./examples/demo01