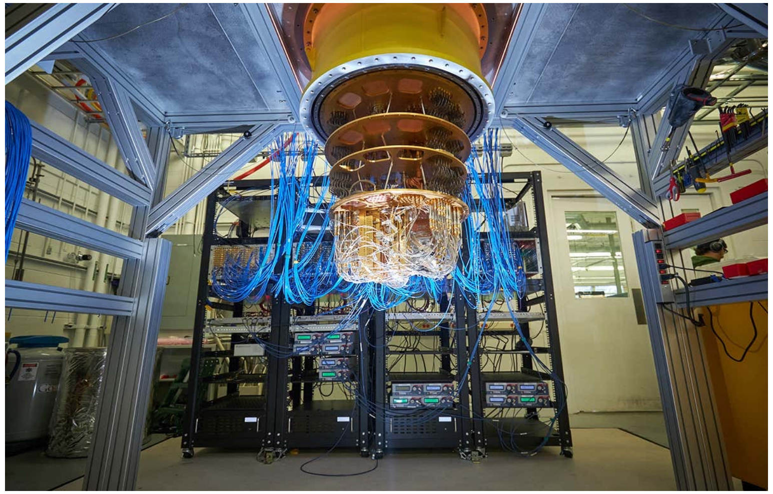
YaoCompiler: Unleash Julia on quantum devices

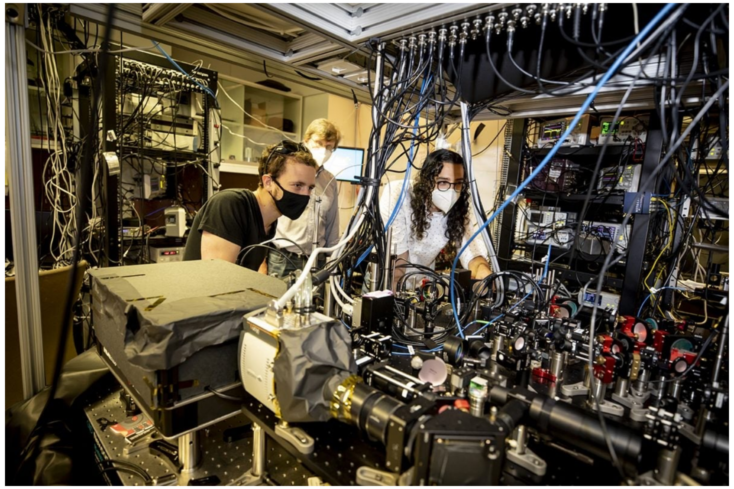
Xiu-zhe (Roger) Luo, Chen Zhao, Valentin Churavy, Roger Melko

QUANTUM INTELLIGENCE LAB University of Waterloo 中国科学院数学与系统科学研究院 Academy of Mathematics and Systems Science, CAS Massachusetts Institute of Technology Unitary Fund

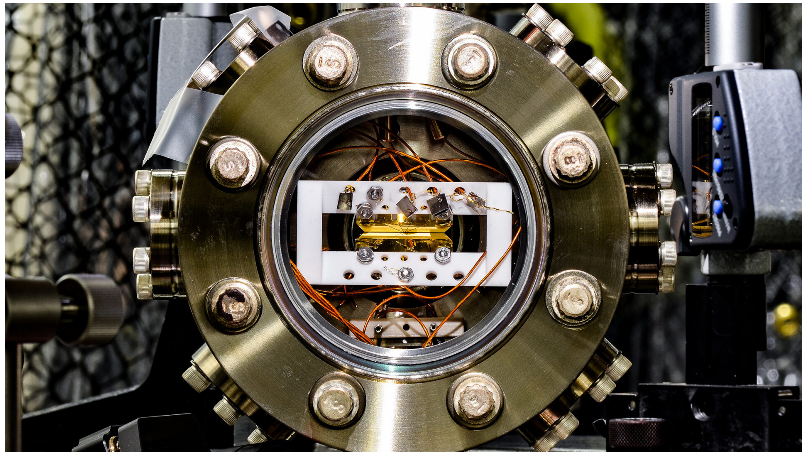
Quantum Information on Real Hardware



Google's superconducting device



cold atom device, Harvard



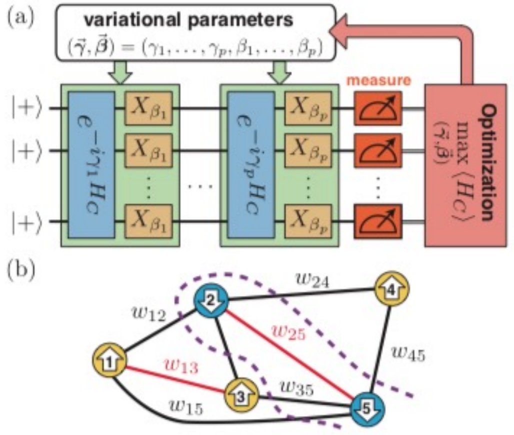
Ion trap device, Harvard

Jiuzhang optical device, USTC

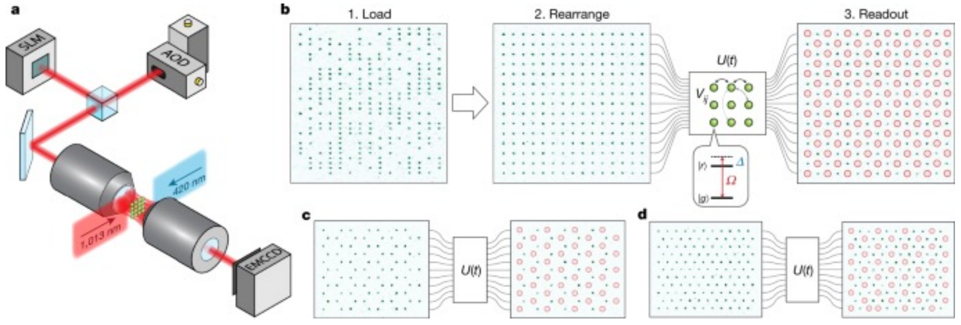
Superconducting device, IBM

And more…

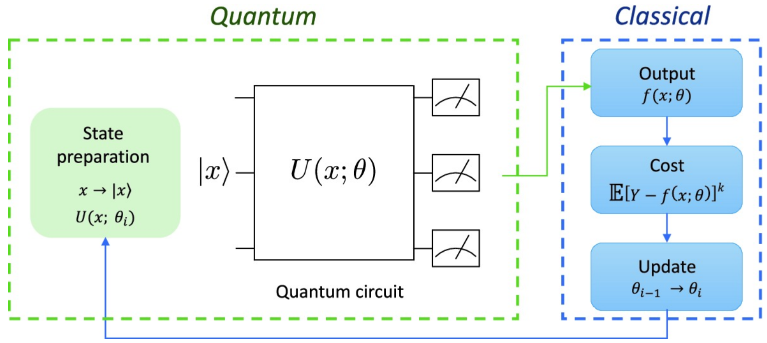
Applications



optimization



simulate quantum system



Quantum

Classical

State preparation

$x \rightarrow |x\rangle$

$U(x;\ \theta_i)$

$|x\rangle$

$U(x;\theta)$

Quantum circuit

Output
$f(x;\theta)$

Cost
$\mathbb{E}[Y - f(x;\theta)]^k$

Update
$\theta_{i-1} \rightarrow \theta_i$

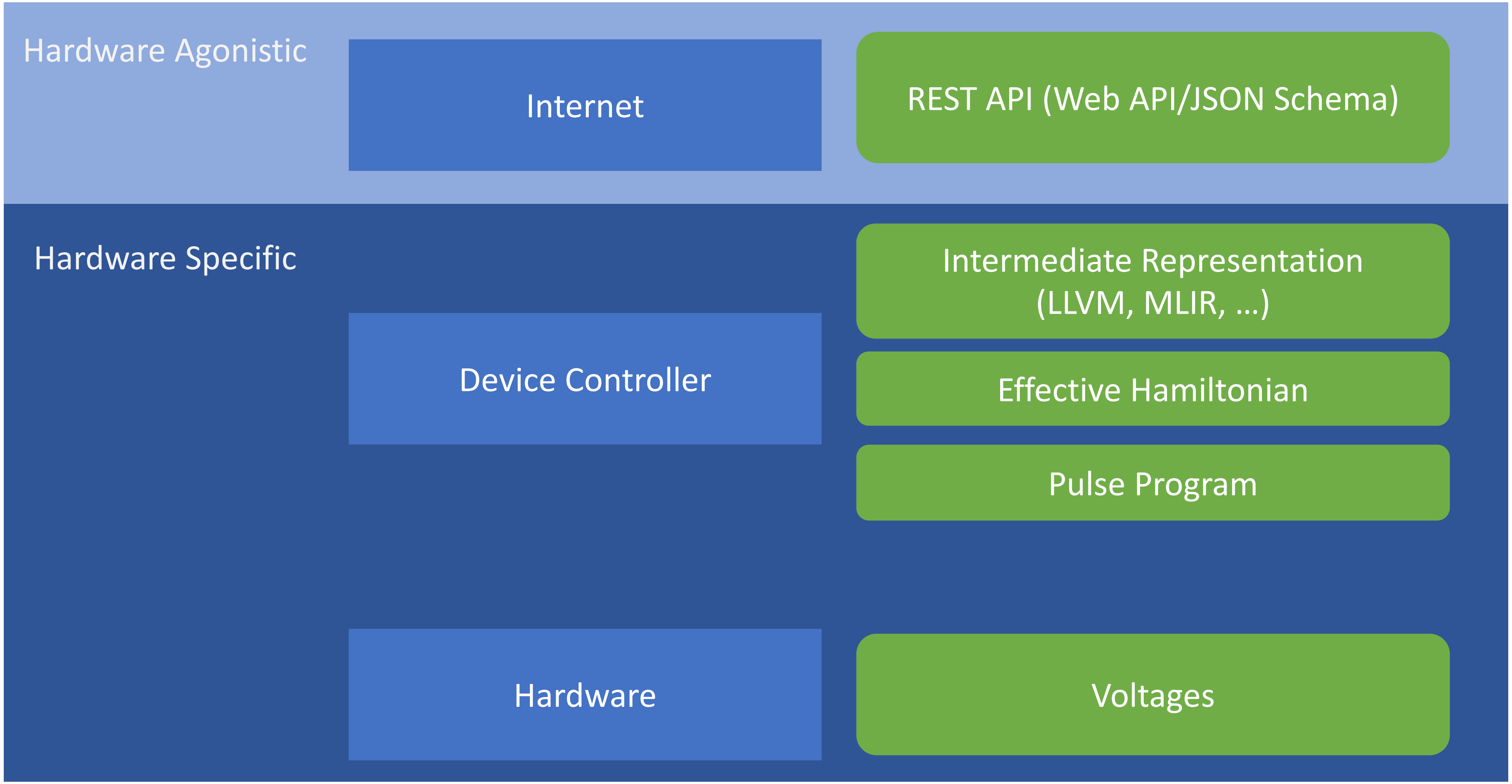
variational eigensolver/machine learning

Ebadi, S., Wang, T.T., Levine, H. *et al.* Quantum phases of matter on a 256-atom programmable quantum simulator. *Nature* **595,** 227–232 (2021).

Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm[J]. arXiv preprint arXiv:1411.4028, 2014.

Peruzzo A, McClean J, Shadbolt P, et al. A variational eigenvalue solver on a photonic quantum processor[J]. Nature communications, 2014, 5(1): 1-7

Programability of Quantum Devices

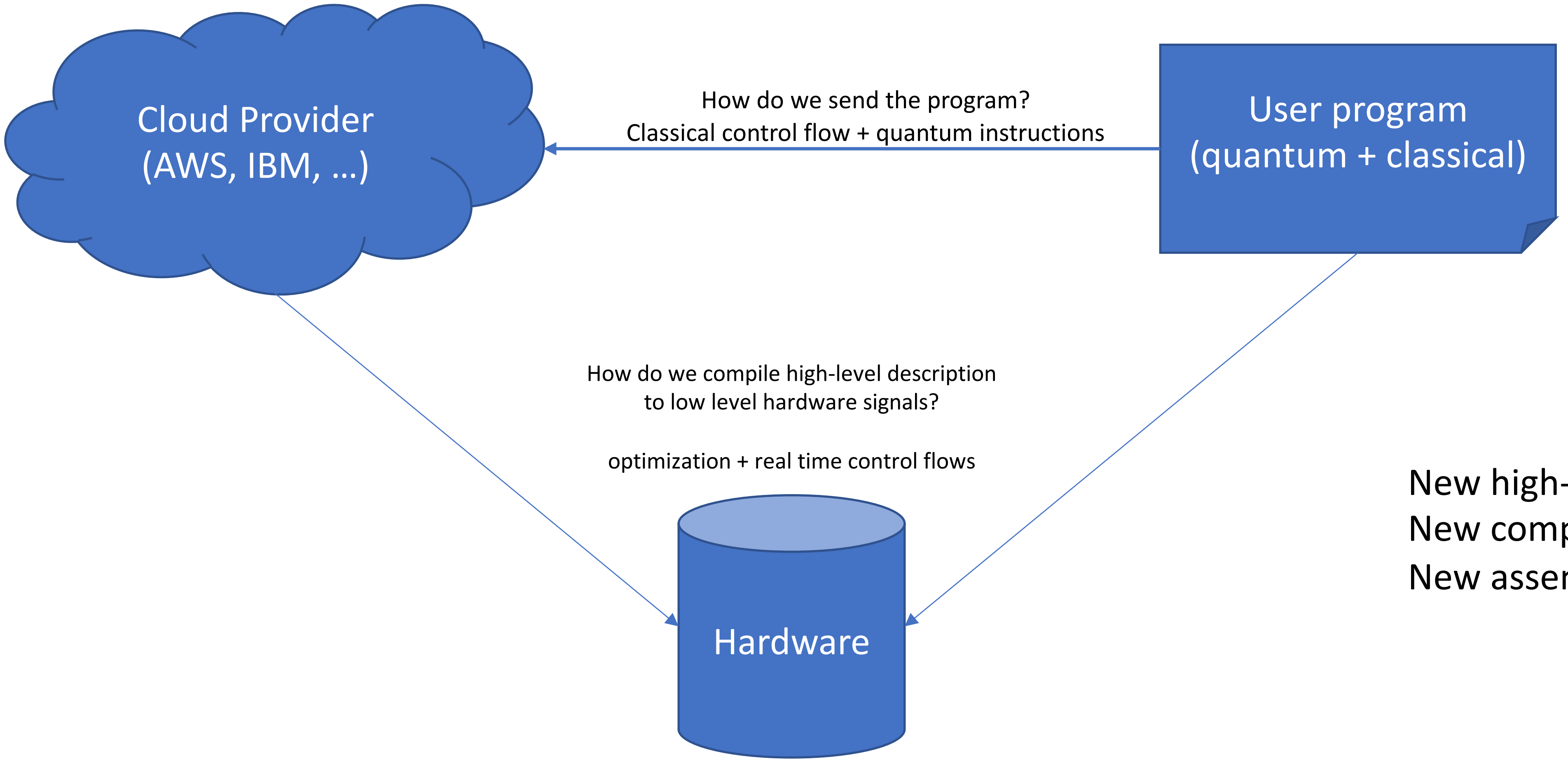


circut-based algorithms

Hybrid algorithms

quantum simulation

How do we program these devices?



Cloud Provider
(AWS, IBM, …)

How do we send the program?
Classical control flow + quantum instructions

User program
(quantum + classical)

How do we compile high-level description
to low level hardware signals?

optimization + real time control flows

Hardware

New high-level language?
New compiler?
New assembly?

Julia: we want one language rather than two!

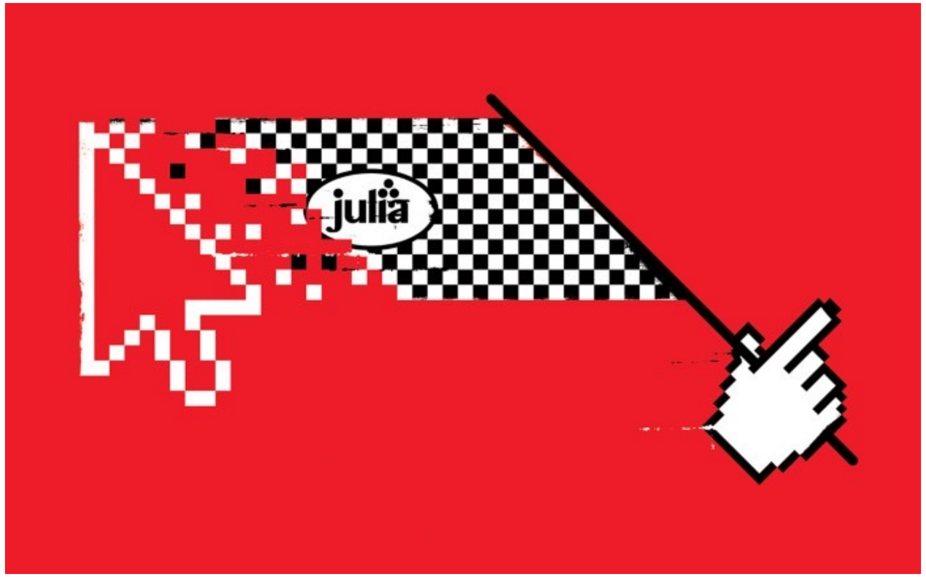
Static Compiled Language

good compiler analysis! But not interactive

Interpreter Language:

Interactive but not much compiler analysis!

The language designed for JIT!



Julia: come for the syntax, stay for the speed, nature, 30 July 2019

Can we have just one language for quantum device?

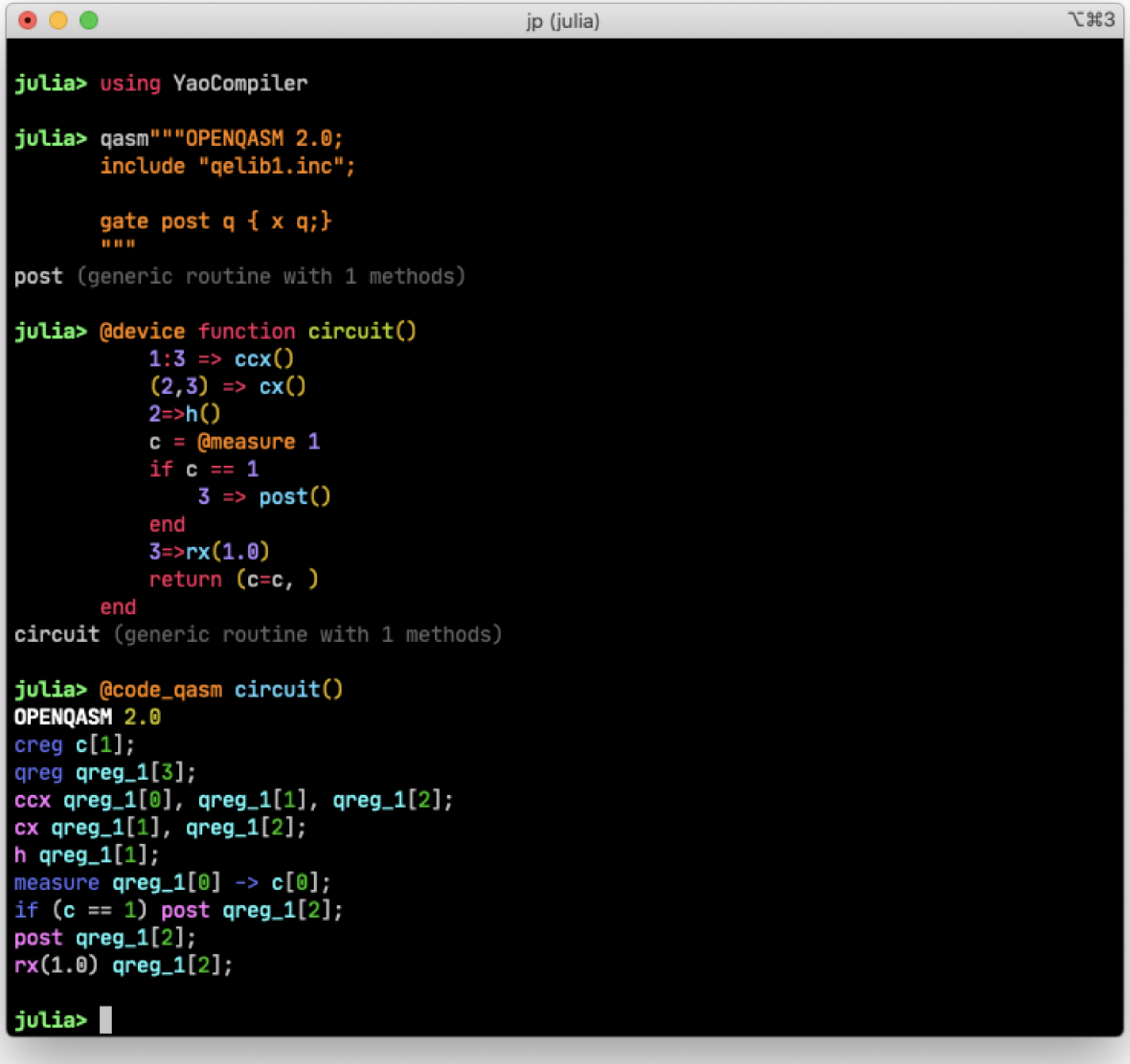
YaoCompiler

Goal of Yao Ecosystem

- Power quantum information research in Julia

Goal of YaoCompiler

- Foundation of quantum compilation in Julia



```
julia> using YaoCompiler

julia> qasm"""OPENQASM 2.0;
       include "qelib1.inc";

       gate post q { x q;}
       """
post (generic routine with 1 methods)

julia> @device function circuit()
           1:3 => ccx()
           (2,3) => cx()
           2=>h()
           c = @measure 1
           if c == 1
               3 => post()
           end
           3=>rx(1.0)
           return (c=c, )
       end
circuit (generic routine with 1 methods)

julia> @code_qasm circuit()
OPENQASM 2.0
creg c[1];
qreg qreg_1[3];
ccx qreg_1[0], qreg_1[1], qreg_1[2];
cx qreg_1[1], qreg_1[2];
h qreg_1[1];
measure qreg_1[0] -> c[0];
if (c == 1) post qreg_1[2];
post qreg_1[2];
rx(1.0) qreg_1[2];

julia> 
```

Powered by the GPUCompiler infrastructure !

Live demo

Compile a quantum program to openQASM

Julia in heterogonous computing era

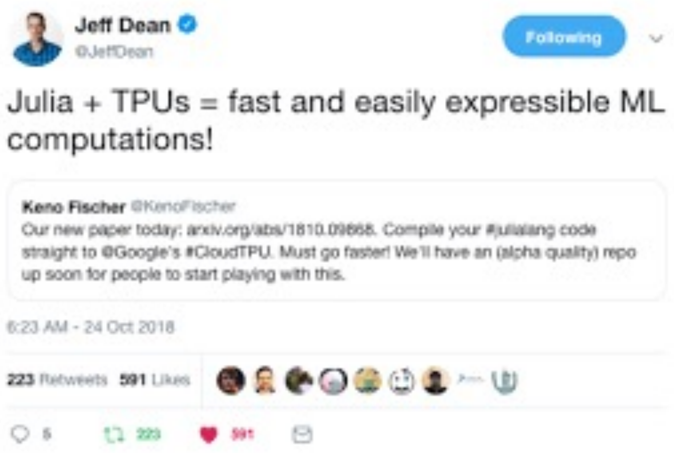
DSL for heterogonous computing
(new language/API with new compiler)

NVIDIA.
CUDA.

OpenCL™

Julia with Compiler Plugins
(new intrinsic with the same language)

JuliaGPU (CUDA, AMDGPU, Metal)



Jeff Dean @JeffDean

Julia + TPUs = fast and easily expressible ML computations!

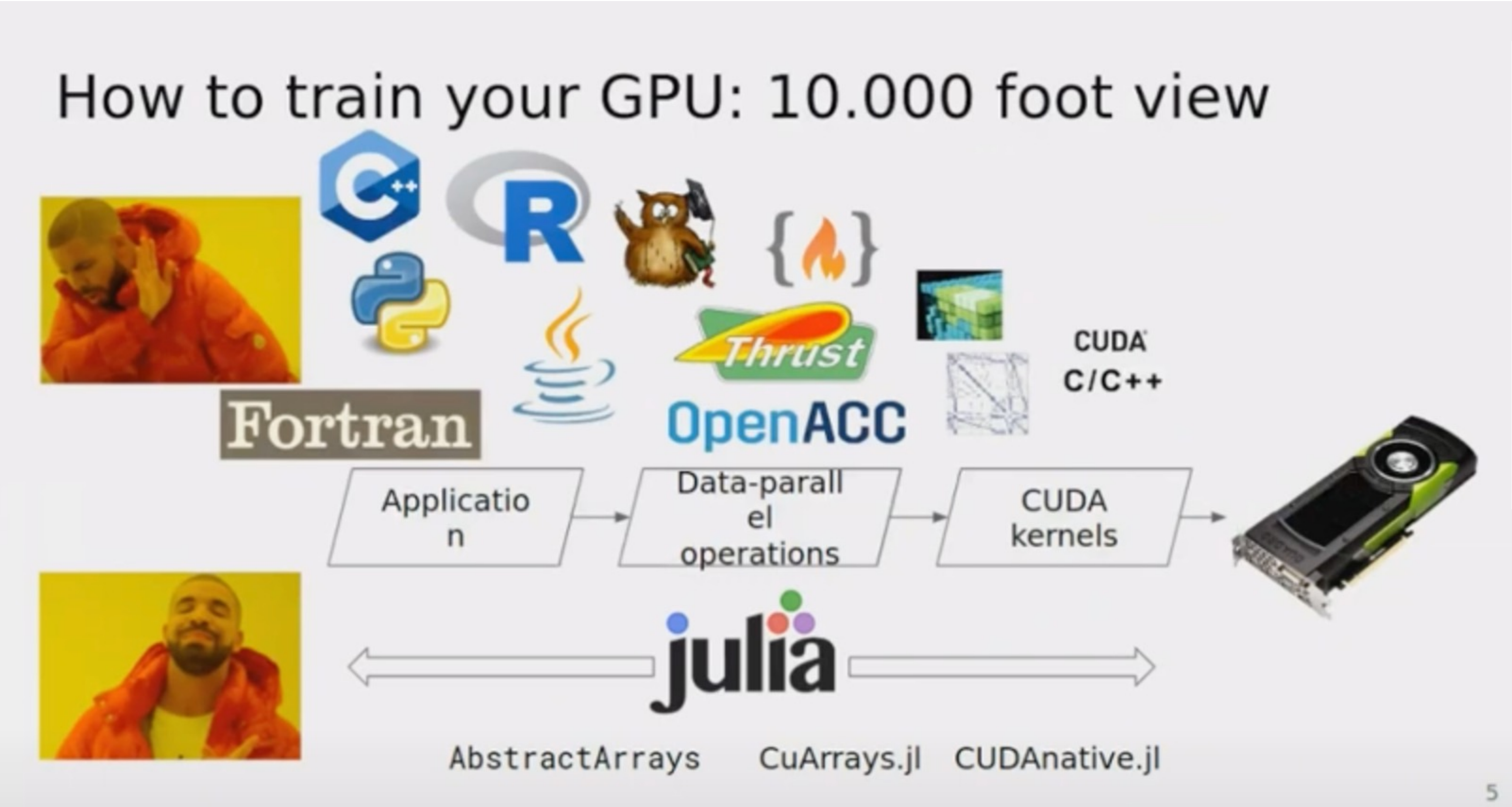
Keno Fischer @KenoFischer
Our new paper today: arxiv.org/abs/1810.09868. Compile your #julialang code straight to @Google's #CloudTPU. Must go faster! We'll have an (alpha quality) repo up soon for people to start playing with this.

6:23 AM - 24 Oct 2018

223 Retweets 591 Likes

5 223 591

And TPU!



How to train your GPU: 10.000 foot view

Application → Data-parallel operations → CUDA kernels →

AbstractArrays CuArrays.jl CUDAnative.jl

5

Tim Besard

Valentin Churavy

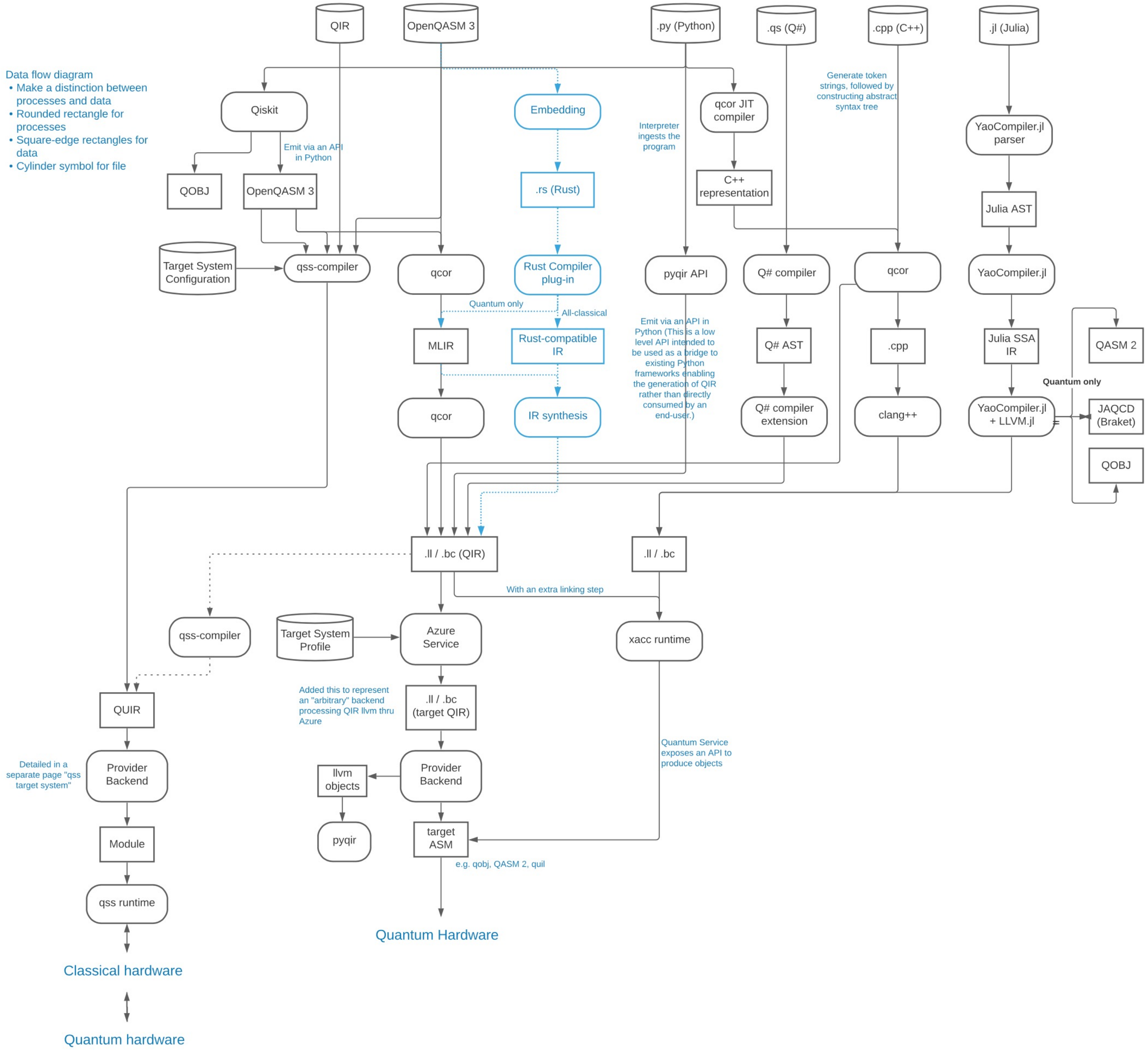
What about…?

	YaoCompiler	QCOR	Q#	Quingo	qiskit
extends	Julia	C++ & Python	.Net	N/A	Python
level	High-level	System	System	System	High-level
Hybrid program	Yes	Yes	Yes	Yes	N/A
Kernel Language	Julia	C++/Python	Q#	Quingo	Python (class)
Host Language	Julia	C++/Python	C#	Python	Python (class)

Advantage of writing quantum program with YaoCompiler in Julia

- Interactive, nice REPL, interactive code generation for debugging
- Only one language needed for compiler, kernel and host
- Ecosystem for algebraic transform, LLVM.jl, Metatheory.jl, Symbolics.jl, Optimization, Graphs
- Composability, compiler understands the whole ecosystem!

- Mintz T M, Mccaskey A J, Dumitrescu E F, et al. Qcor: A language extension specification for the heterogeneous quantum-classical model of computation[J]. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2020, 16(2): 1-17.
- McCaskey A, Nguyen T. A MLIR Dialect for Quantum Assembly Languages[J]. arXiv preprint arXiv:2101.11365, 2021.
- Cross A. The IBM Q experience and QISKit open-source quantum computing software[C]//APS March Meeting Abstracts. 2018, 2018: L58. 003.
- Team T Q D. Quingo: A Programming Framework for Heterogeneous Quantum-Classical Computing with NISQ Features[J]. arXiv preprint arXiv:2009.01686, 2020.


Data flow diagram
• Make a distinction between processes and data
• Rounded rectangle for processes
• Square-edge rectangles for data
• Cylinder symbol for file
QIR
OpenQASM 3
.py (Python)
.qs (Q#)
.cpp (C++)
.jl (Julia)
Qiskit
Emit via an API in Python
QOBJ
OpenQASM 3
Embedding
.rs (Rust)
Rust Compiler plug-in
qcor JIT compiler
C++ representation
Interpreter ingests the program
Generate token strings, followed by constructing abstract syntax tree
YaoCompiler.jl parser
Julia AST
YaoCompiler.jl
Target System Configuration
qss-compiler
qcor
pyqir API
Q# compiler
qcor
Quantum only
All-classical
MLIR
Rust-compatible IR
Emit via an API in Python (This is a low level API intended to be used as a bridge to existing Python frameworks enabling the generation of QIR rather than directly consumed by an end-user.)
Q# AST
.cpp
Julia SSA IR
QASM 2
Quantum only
qcor
IR synthesis
Q# compiler extension
clang++
YaoCompiler.jl + LLVM.jl
JAQCD (Braket)
QOBJ
.ll / .bc (QIR)
.ll / .bc
With an extra linking step
qss-compiler
Target System Profile
Azure Service
xacc runtime
QUIR
Added this to represent an "arbitrary" backend processing QIR llvm thru Azure
.ll / .bc (target QIR)
Quantum Service exposes an API to produce objects
Detailed in a separate page "qss target system"
Provider Backend
llvm objects
Provider Backend
Module
pyqir
target ASM
e.g. qobj, QASM 2, quil
qss runtime
Quantum Hardware
Classical hardware
Quantum hardware

What has been done or has prototype?

Code generation

- ✓ LLVM IR code generation
- ✓ OpenQASM frontend/codegen

Hardware Targets

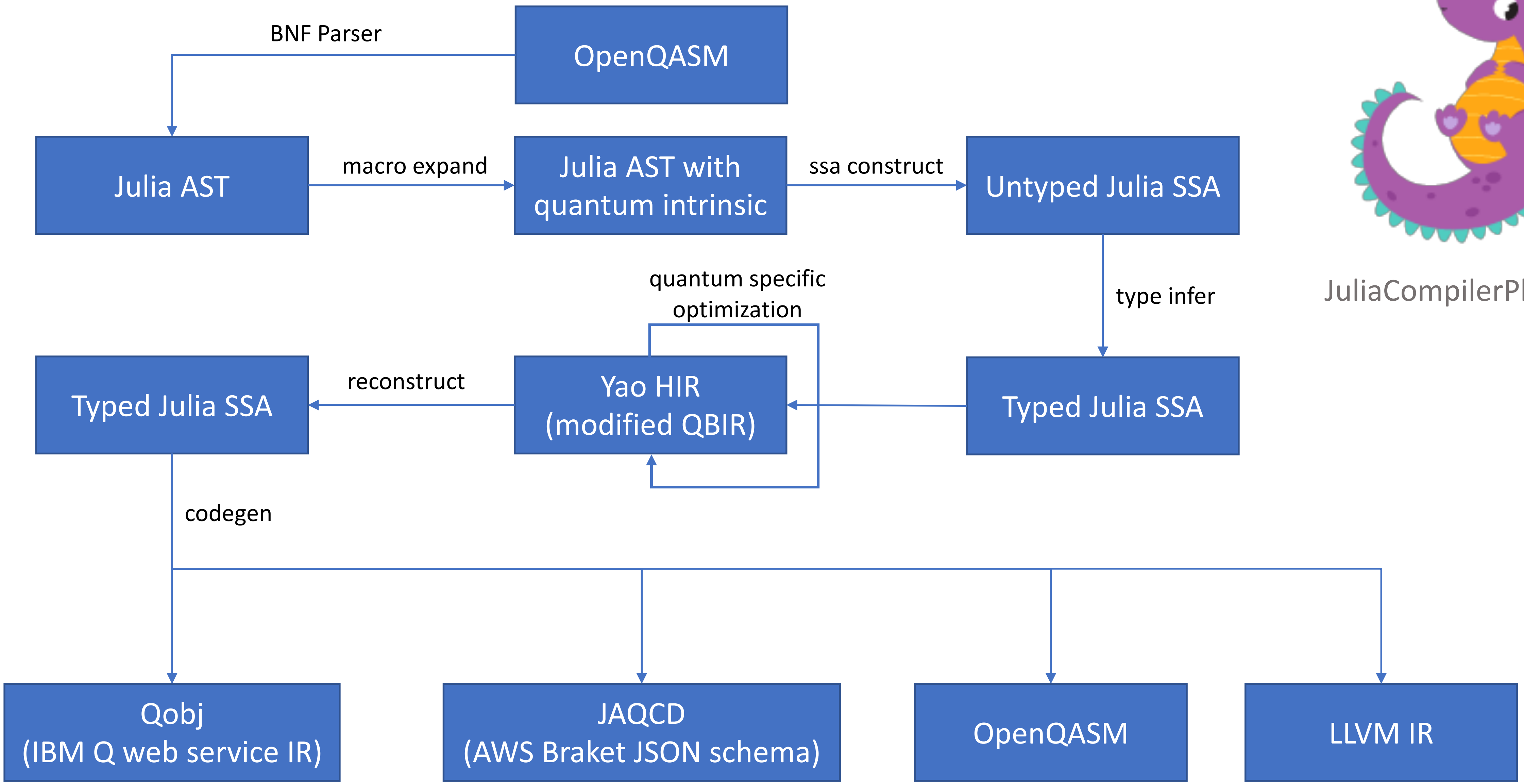
- ✓ IBM Q service as backend
- ○ AWS Braket as backend (WIP)

Compiler Optimization

- ✓ ZX calculus-based circuit simplification
- ✓ Hybrid program analysis (via Julia compiler)
- ○ Brute force pattern match via egg (WIP)

Landscape of compiler toolchain (2021)
QEDC Compiler workshop

YaoCompiler pipeline



OpenQASM

BNF Parser

Julia AST

macro expand

Julia AST with quantum intrinsic

ssa construct

Untyped Julia SSA

type infer

Typed Julia SSA

Yao HIR (modified QBIR)

quantum specific optimization

reconstruct

Typed Julia SSA

codegen

Qobj (IBM Q web service IR)

JAQCD (AWS Braket JSON schema)

OpenQASM

LLVM IR

JuliaCompilerPlugin org

Challenge for modern compilers

How to:

- Compile a high-level language all the way to down to the hardware? (architecture, this work)
- simplify a quantum circuit to reduce gate count? (ZX calculus, pattern matching)
- decompose arbitrary gate into given gate set? (circuit synthesis)
- decompose long entangled gates into more local gates? (circuit synthesis, mapping)
- decompose a gate operator into pulse sequence? (pulse design)
- mitigate noise effects? (error mitigation)
- find the effective Hamiltonian of a Hamiltonian in question?
- Assert the error due to ill-defined quantum operation? (quantum programming theory)