- What is Open Source hardware?
  - electronic designs/schematics (e.g. mixing boards)
  - firmware for FPGA boards
    - ideally, FPGA boards would be more geared towards Quantum computing, only having the needed components and thus reducing cost (needs sufficient market power)
  - all designs modifiable by the community
  - includes software to drive the hardware

- Mid-circuit measurement?
  - dynamic circuits and feedforward: make a measurement in the middle and respond based on the results
  - requires low-latency between measurement and drive hardware
  - software support for programming
  - different types of readout for mid-circuit and final
  - stringent requirements on hardware for scaling
  - application level / algorithms should drive requirements

  - circuit cutting/knitting algorithms
  - iterative phase estimation reusing qubits
  - teleportation, reusing qubits to generate new keys

  - adding classical, non-unitary operations to a quantum program

- Entanglement across different sites?
  - initial goals is across Berkeley, final across the State
  - mix different technologies? no reason for this to be better

- Run-time systems?
  - Amazon bra|ket integrating with differents
  - splitting circuits if they don't fit

- How much to expose to the user?
  - users typically don't know the right questions to ask and if given, what to do with the information
  - need experts who understand both sides and can communicate, just like HPC consultants

- Faster turnaround?
  - for most users, is an issue of queues
  - sometimes matter for results (drift, new calibration), often not
  - trade-offs possible at the hardware (e.g. heralding)

- Abstraction levels?
  - when/how do we reach the same levels as classical computing?
  - quantum computing had it's computational model imposed on it, and the abstraction level as relatively low (gates)
  - compilers may be a help for mapping to native gates
  - gate abstraction level has been very useful to popularize initial interest in quantum computing

  - domain scientists do not care about the device, only about the results and can afford this with classical computing
  - assumption that quantum computing is "almost there" is simply not true yet, it's not just an engineering problem
  - investing people-power in abstraction level may very well be too early until some fundamental problems are solved
  - the above points to the importance of workforce development

- Classical computing changed the programming paradigm
  - after scaling down components ended, parallel computing took over and exponential scaling continued
  - there are still physics problems to be solved to accomplish the same in quantum computing (e.g. communication between QPUs, or different QPU technologies for lower noise)
  - abstraction levels design today are likely to break in the future until the fundamental problems are solved and keeping hardware in mind in the abstraction levels allows a more gradual change over time

- Workforce development?
  - graduate programs that combine phyiscs and computer science, with the latter focused on quantum computing
  - provide programs for computer scientists to learn quantum computing and vv. for physicists

---

- Simulation
  - reduced graphs are typically 50% smaller

- Noise simulation
  - what models are necessary to get accurate results?
    - time and state dependencies?
    - who is the end-user?
    - how do you combine different models? what's the physics?
  - noise scaling? small model mismatch may remain small at

scale, or it may get much worse
  - algorithms have different noise sensitivy so simulating individual noise source may allow you to determine where to put mitigating efforts

  - hardware characterization, how and what?

- Distributed quantum computers
  - with quantum communication not available, can classical communication be used?

- Backend?
  - use LLVM? classical optimization has many well known optimizations, so an IR that supports these makes sense
  - for machine learning, MLIR, the common optimizations are in the backend b/c of heterogeneity
  - what does the above mean for quantum hardware? the common optimizations are not yet defined
  - frontend should always be reusable
  - need to capture commonality between hardware vendors that could be part of an MLIR for specific optimizations

- Topology?
  - coupling graph is arbitrary, qubits are not assumed to be homogeneous

- Error correction?
  - use run-time detection and adapt
  - equivalent of check pointing?
  - errors such as from cosmic rays, which affect classical systems, don't matter yet but might in the future

- Compiler design
  - collaborative work helps to improve the abstraction levels
  - what are the backends are the best to use?
  - dearth of students with a compiler backend
  - heterogeneous computing and quantum computin put more importance of compiles, which may renew interest

---

- How far can you push error mitgation in post-processing?

  - main goal should remain engineering better gates
  - coherent and cross-talk errors should be reduced as much as possible before noise tailoring with Pauli errors
  - current model captures some crosstalk
  - effect of dynamic programming on post-processing may go either way (reducing or adding noise)

- Noise model
  - multiple mistakes can cancel each other out
  - drift is a concern
  - additional methods for correcting two-level system, which may make noise models more stable
  - higher weight Pauli terms
  - not clear whether a better noise model will provide better result than the current
  - better gates exponentially reduce the requirements of the noise model
  - could a machine learning model work?
    - QML training bakes in the noise through training< which suggests their might be something to capture by a model
    - Pauli twirling removes a lot, but not all structure, eg. noise b/c of TLS for qubits with low T1

- Programmability
  - introduce primitives (e.g. "estimator") that add noise mitigation in post-processing based on "resilience levels"
  - generate circuits from desired resilience level

- Security
  - has been a discussion topic in the past DOE workshops
  - what about small circuits run in parallel on larger chips, which could figure out what others are running based on crosstalk behavior (but would be a very hard problem)

- Codesign challenges

  - Flexible control hardware platforms that are both cost-effective and scalable such as AQT's Qubic which integrates an FPGA (field-programmable gate array) RF (radio frequency) system, which modulates the signals at room temperature to manipulate and measure the superconducting qubits cooled down to cryogenic temperatures.

- The cost, size, and complexity of control and measurement hardware increase with a growing number of qubits. This presents a significant roadblock for startups and junior academic research groups worldwide.

What is different between hardware and software?
Hardware can be defined as schematics for electronics, open source hardware
There is lot of components on FPGA board such as busses 5g wave and readout is multiplex and you don't need that for quantum
lean efficient cost effective and powerful,
University of oxford went to adaption of bringing cooperation to partner with them.
Ideally software and hardware would be codesigned

What you mean by circuit measurement?
Quantum circuit used to be statics and fixed, dynamic circuits are when you make a decision at middle of a circuit dynamically and measure one qubit to make a decision. Latency needs to be fast, communication and decision making needs to be fast.
It has to be done on control system.
Qiskit and openwasm3 support this.

Quantum processor on readout: readout needs to be fast enough adding stacks of qubit hardware together can be get complicated
Decision making needs to sit on top of 100 or 1000 control system and the topology of how to make fast decision making is important

It is important to codesign how many operation you need between dynamic system and in terms of control system you need to specify that
What is the length of entanglement wire:
It is at start up but ideally it is going to be between state at beginning

The control system:
Do you think it is going to be developed into something similar to HPC ?
Amazon already tried to do such a thing to unify the access to braket in one instance

One benefit of dynamic circuit is to have mid circuit decision
Classical version of dynamic circuit is non unitary circuits similar to if and else
When you have a 2^10 possibilities
How much exposure for control system should be done?
Does the developer needs to know everything to get max performance?

If you explain the lower hardware to people they get disinterested due to the difficulty and normally because they don't know what technologies they are using they usually don't ask the right question

Compare to HPC is quantum there yet?
How different each technology is ?
In terms of executing fast operation to get an answer or not there are lot of technologies to do that.
There is so many trade off
There is different type of users first type is the ones who just submit a circuit and software stack just works for them
The other type is they want transparency and want to know what's happing in each step
These are difficult but needed but the question is how do you balance this to give more access and keep everything standard

The level of abstraction stayed for classical but do you think this can happen for quantum?
With quantum computing people build abstraction level first instead of
To make parallel progress you need some abstraction
The abstraction model build for quantum was based on gate model and it takes a long time to calibrate for that model
The level of abstraction has not happen for separation of problems and build on top of some abstraction
It would be useful for openqasm to incorporate what unitary gate is being use or if we can change some operation in level of compiler level

Qiskit primitive and other tools shows a new way of using quantum computers compare to before and show a new way of expecting non perfect state from results

Allocation of resources for progressing in quantum computing hardware is one of the most important problem. Quantum computing is at the beginning of path and all the investors just investing just because of hype

Explore application and multiple level of stack is also important
In classical computing device level people have looked to build a large system,
Sun attempting to build 60 connected and it was assumed that was the end but today we scale with other ways

1950, 1960 transistor bell labs is the analogy to quantum computer
Instead of big hype maybe a slow steady push would be more appropriate

To detect smaller graph attention based gnn and pulling and it is based on physics principle
The accuracy is about 50%

The noise model: do you think it is possible to tune at hardware level and codesign way
The information we can get from noise model show the controllable factor , this is software and this is hardware how to manage noise ?
There is gap between hardware and noise to software can we get some build some accurate model ?
Real hardware noise model what is look like?

The short answer: it is possible to get as much as info possible
The problem is how much information you need
One philosophy is to get much information as possible and the other group says as less info just useful ones.
The sweet spot is to research and get ideal spot
In terms of qiskit you query backend by date
What are the limiting source of error ?
Tunable coupler
Noise characterization evolves noise tailoring, gate calibration model instead of t1 t2 .
You can tailor gate error to coherent error or something expected
Cross talk and other things to characterize
Should software stack includes more information about error ?

Which users you need to specify

How to combine all sort and type of noise together?
How to include some valuable information about noise?

It is current research direction to provide some uncertainty about error and physical system and their on going system for qubits

Power grid, they don't consider noise in simulation
It is hard to validate the noise and verification

There is no right level of noise and the question is what level of accuracy you want for your application
Small scale qubit noise is vary from large scale 127 1000 qubit
Maybe a multi model noise stack
Type of noise for correction first
List of noise which one you want to correct

Is it better to study noise in pulse level ?
It is very hard to do, at this point the pulse is not accurate and you don't have perfect model, that's the most convenient but not the most accurate to follow

Another problem is the shots problem if have pauses between runs on quantum hardware you get more calibration happening and that throughs the calculation away

Using simulator for hardware, d-wave doesn't actually met the requirement,

Scale up quantum computing:
VQE the idea is you have to run it multiple times, the lots of devices have different properties,

Backend llvm use: llvm can be use in infrastructure but it is not useful for compiler based that they are doing
Looking at Microsoft, IBM and etc. finding the right model and standard model would be next step.
One solution is provide specification and vendor takes into that for building devices
QIR was originally only worked with Q# but now they have more support for open QASM for simulator perspective you can look at like basic gates operation
One perspective is to look at it as the whole knowledge and the rest would be backend optimization
Qubit mapping: program with lower cost can be achieved by selecting different sets of mapping
How do you take care of temporal issue? How do you resolve that?

That is not have been studied yet, but I think that is an important issue, we know that device is noisy and detects temporal effects.
How many times error occurs and how effective is that on the entire circuit?
HPC resilience is checkpoints and restore it back
MIT led bricks to protect against cosmetic rays

The limit to get precise data ? how far can we push?
Any useful quantum advantage?

We have to keep improving the hardware, IBM path error correction has been updated for the last few month with growth code. It still needs good gates,
Dynamic circuit: are there near term algorithm that can leverage the dynamic circuit, mid circuit measurement and PEC ?
Is there any improvement to current noise model?

The tricky part is you can have multiple coefficient canceling each other, the drift and how to modify and verify is next step to take
To make noise model more stable: the topology, and other factors can be considered
The continues path to quantum advantage is IBM direction.

Do you think that is possible to use ML to make decision for compiler ?
Maybe, my personal take on ML: if you can do it in systemic way to learn how to do it, but you have to consider drift of noise model
Noise can be drift at any point to any random value so ML might not be able to create a accurate model.
Error propagation:
For software system, how do you think we should engage error propagation?
Sampler, estimator you can expectation value and mitigate readout error,
In terms of security tow concept:
Can you steal the information or destroy information
Destroy is easy and it could be a problem for cause an issue for integrity