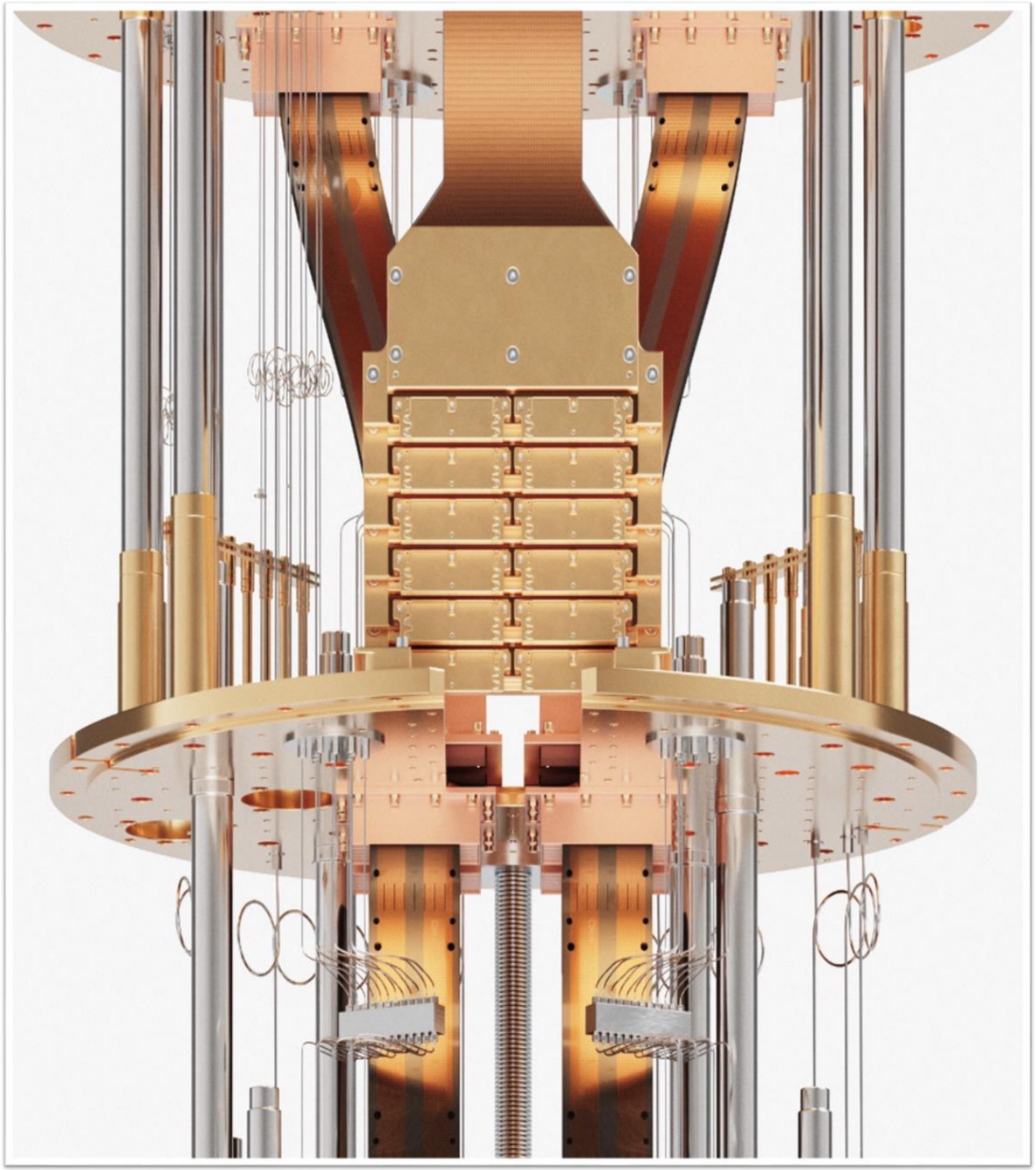
Error Mitigation

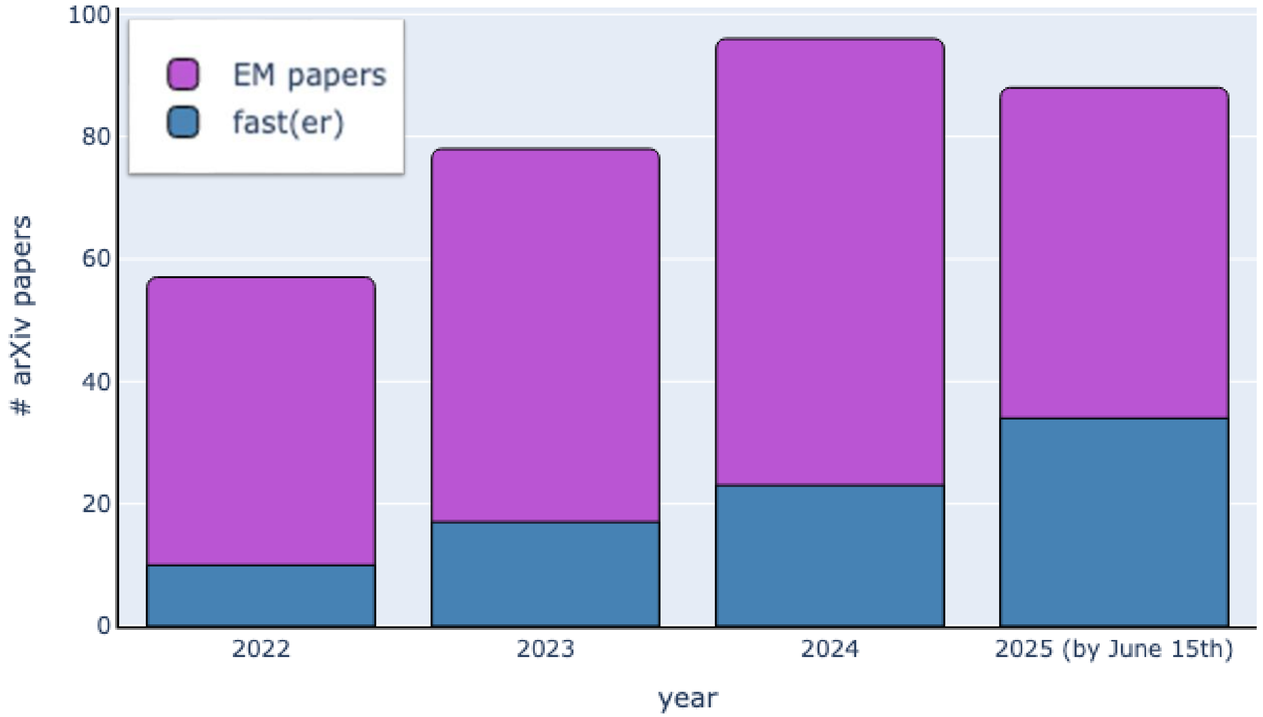
From theory to performant code

Sam Ferracin
Backend Software Engineer
IBM Quantum

WERQSHOP
New York, USA
July 17, 2025



Error Mitigation: A field of growing interest



Efficiently improving the performance of noisy quantum computers

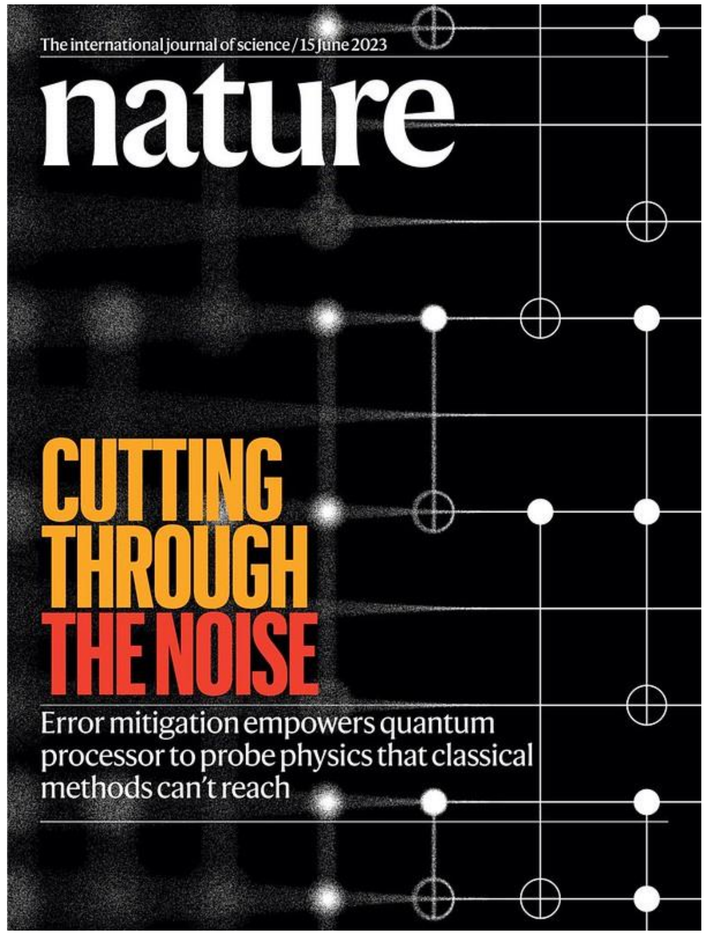
Samuele Ferracin[1,2], Akel Hashim[3,4], Jean-Loup Ville[3], Ravi Naik[3,4], Arnaud Carignan-Dugas[1], Hammam Qassim[1], Alexis Morvan[3,4], David I. Santiago[3,4], Irfan Siddiqi[3,4,5], and Joel J. Wallman[1,2]

On arXiv in June 2022, many improvements since:

- More efficient noise learning techniques.
- More efficient PEC sampling, e.g. light cones.
- RC on FPGA.

None of this matters
if our protocols do not become
performant software.

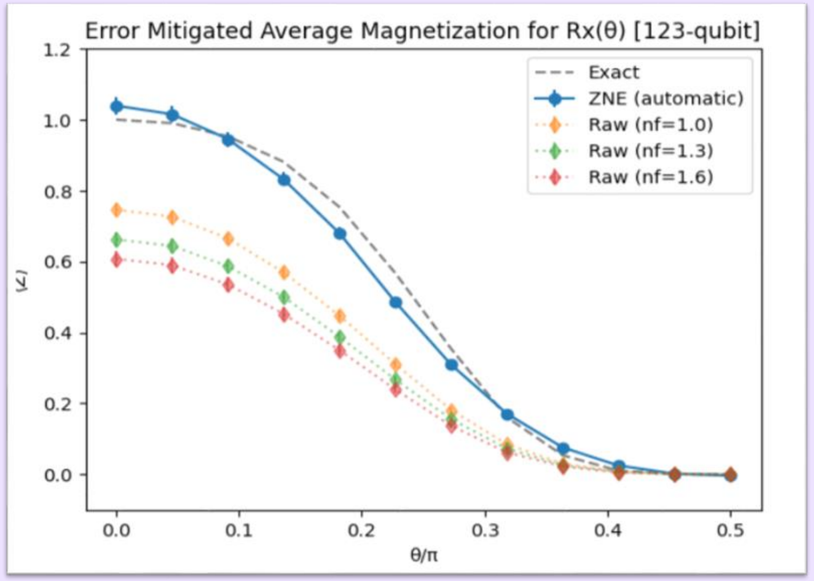
Efficient “on paper” vs performant software



[1] Kim, Y. and others, *Evidence for the utility of quantum computing before fault tolerance*, Nature 618 (2023).

In summary.

- Introduces “Pauli Error Amplification”, or PEA.
- Shows PEA-mitigated results for a circuit with **2’880** two-qubit gate.



- Runtime: ~~~112hrs~~ **~2.2hrs**

From “on paper” efficient to performant code

What does it take to turn a good error-mitigation paper into performant software?

In this talk.

- **Section I.** Software performance.
  - Achieving a 50x speedup.
- **Section II.** Software capabilities.
  - The evolution of primitives.

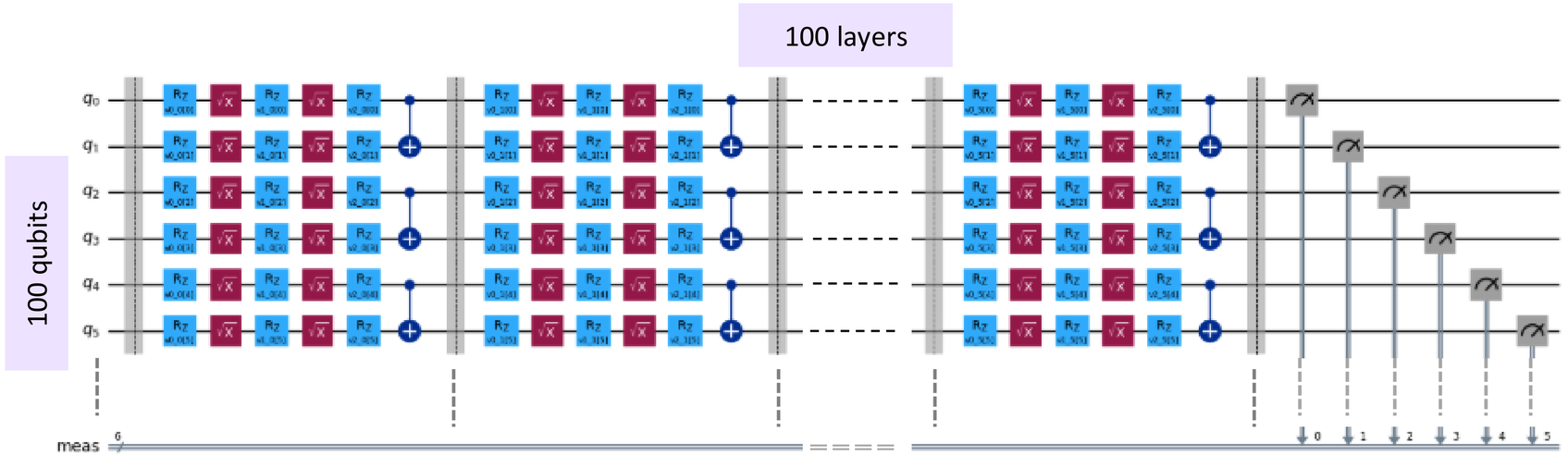
Patterns for performance.

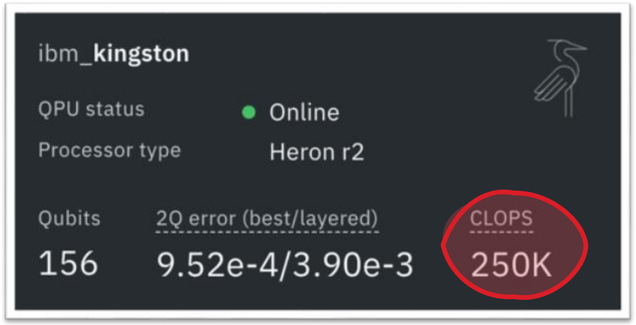
Achieving a 50x speedup.

The CLOPS benchmark

CLOPS, or “Circuit Layer Operations Per Second”, see Ref. [1].

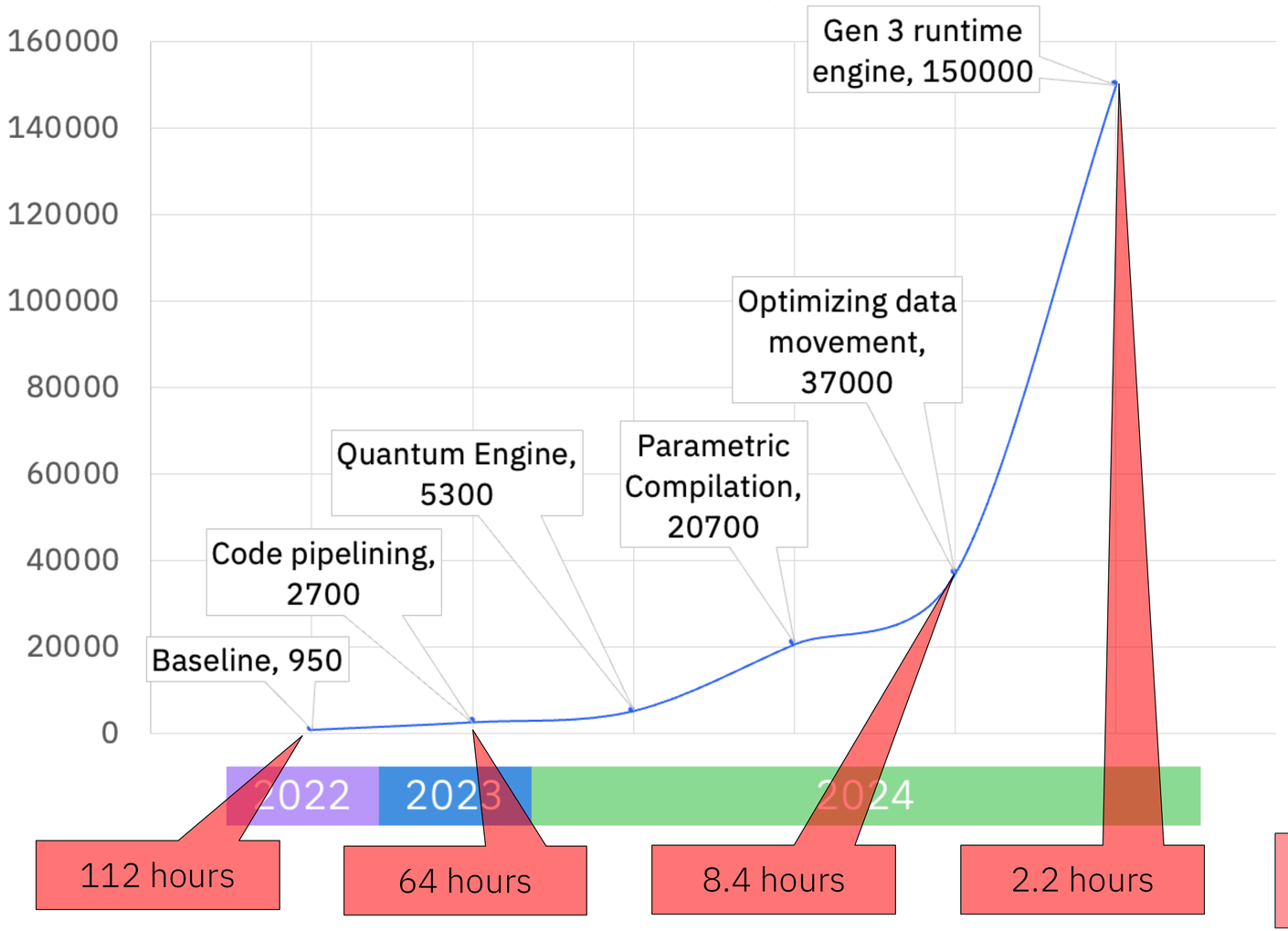
- Measures the steady throughput of parametrized, HW efficient, utility-scale circuits (100 qubits x 100 layers).
- Measured regularly and reported on the cards.





[1] Wack, A. and others, *Quality, Speed, and Scale: three key attributes to measure the performance of near-term quantum computers*, arXiv:2110.14108.
[2] Check it out at https://github.com/qiskit-community/qiskit-device-benchmarking/tree/main/qiskit_device_benchmarking/clops

CLOPS progress



ibm_torino			
QPU status	Online		
Processor type	Heron r1		
Qubits	2Q error (best/layered)	CLOPS	
133	1.53e-3/7.76e-3	210K	

ibm_kingston			
QPU status	Online		
Processor type	Heron r2		
Qubits	2Q error (best/layered)	CLOPS	
156	9.52e-4/3.90e-3	250K	

ibm_aachen			
QPU status	Online		
Processor type	Heron r2		
Qubits	2Q error (best/layered)	CLOPS	
156	9.45e-4/4.37e-3	250K	

Time to run the utility experiment

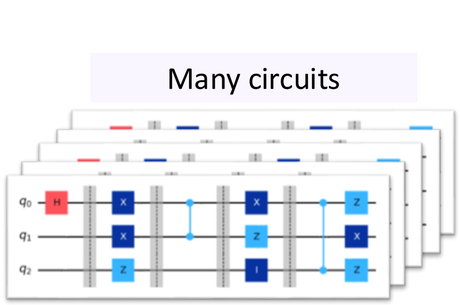
How to achieve a 50x speedup

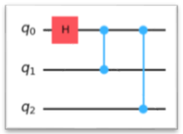
Patterns for performance:

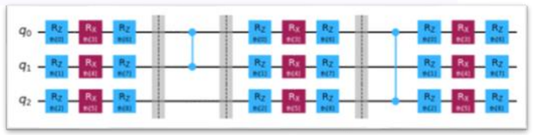
1. **Parallelization.**
   Increase classical work and hide that behind quantum work.

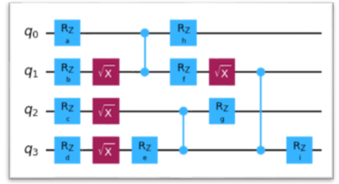
   - **Compiler.** [utility experiment → 64hrs].
     - Sampler and Estimator leverage this parallelism for you.
     - Many circuits in one job better than one circuit in many jobs.

   - **Primitives.**
     - Run more jobs from the queue, potentially from different clients.



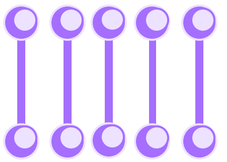






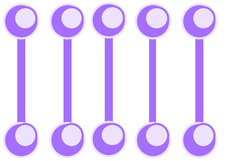
Orchestration code

User space.
- Provides circuit.
- Specifies desired twirling and mitigation strategy.



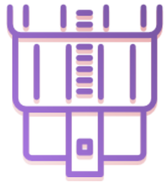
Orchestration code

Primitives.
- Prepares a set of circuits.
- Dispatches to compiler.



Orchestration code

Compiler.
- Maps circuits to instructions streams for control electronics.



Control electronics.
- Executes instructions.
- All circuits loaded run at full speed, no restart/reload required.

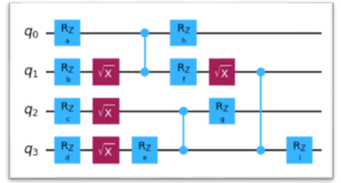
How to achieve a 50x speedup

Patterns for performance:

1. **Parallelization.**
   Increase classical work and hide that behind quantum work.

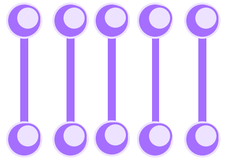
2. **Parametrization.**
   Multiple circuits can be compiled at the cost of one.

   - **Compiler.** [utility experiment → 8.4hrs].
     - Compiler knows how to compile parametrized circuits.
     - Stitches the parameters only before sending to control electronics.



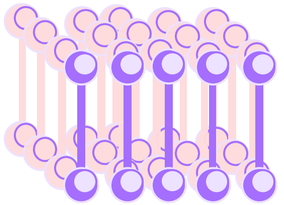
Orchestration code

User space.
- Provide circuit.
- Specify desired twirling and mitigation strategy.



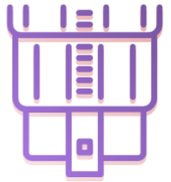
Orchestration code

Primitives.
- Prepare a set of circuits.
- Dispatch to compiler.



Orchestration code

Compiler.
- Map circuits to instructions streams for control electronics.



Control electronics.
- Executes instructions.
- All circuits loaded run at full speed, no restart/reload required.

How to achieve a 50x speedup

Patterns for performance:

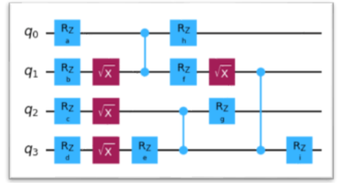
1. **Parallelization.**
   Increase classical work and hide that behind quantum work.

2. **Parametrization.**
   Multiple circuits can be compiled at the cost of one.

3. **Optimization.**
   Clean code runs faster.

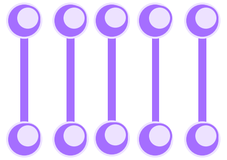
   - Started with rewriting parts of the lower level of the stack in Rust.
   - Continued with rewriting the compiler gen3 entirely in Rust.

   More details in Andrew Wack's QDC talk:
   https://www.youtube.com/watch?v=uLuDyrJIvO4



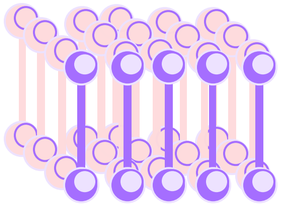
Orchestration code

User space.
- Provide circuit.
- Specify desired twirling and mitigation strategy.



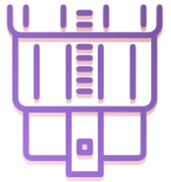
Orchestration code

Primitives.
- Prepare a set of circuits.
- Dispatch to compiler.



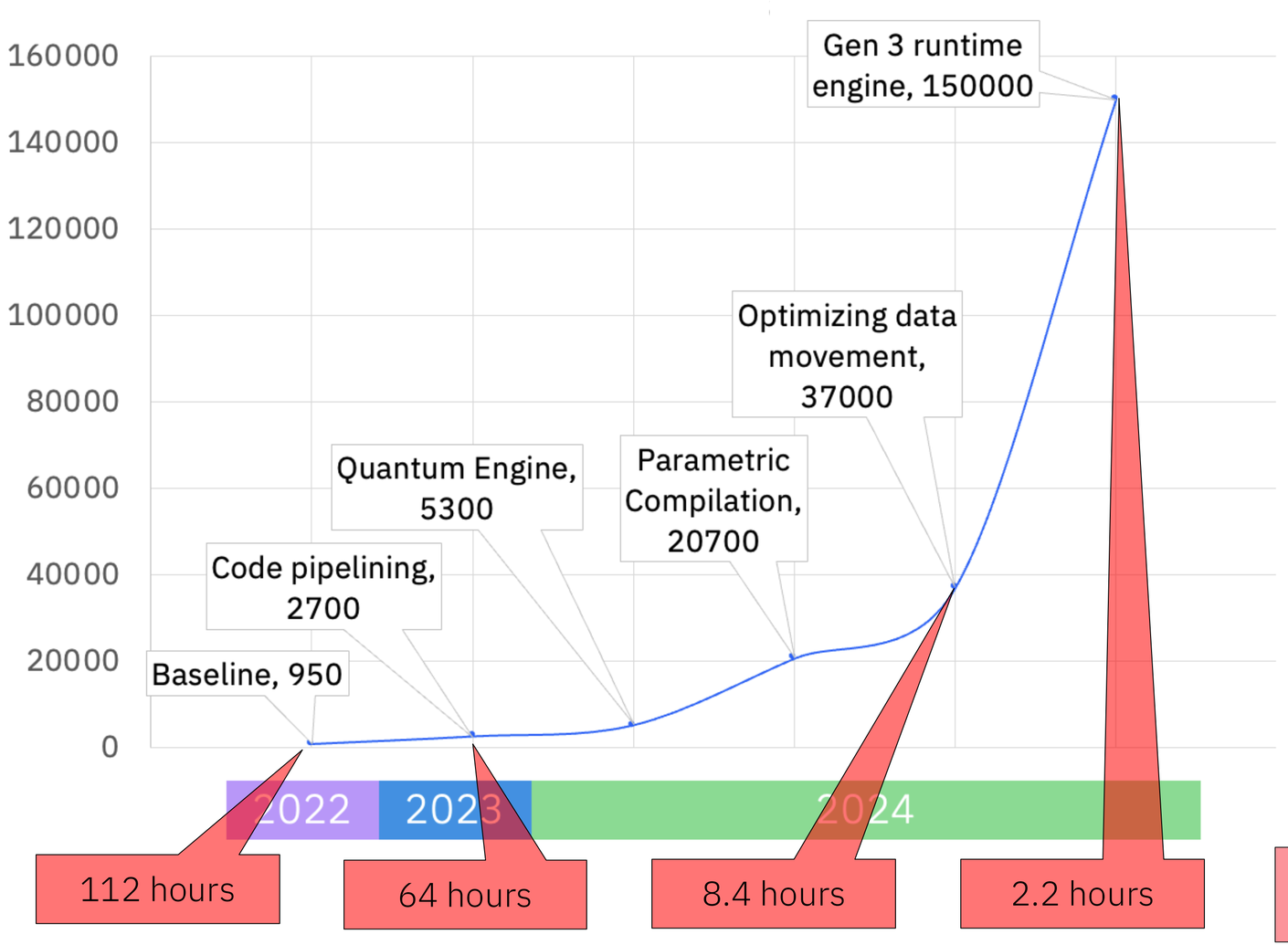
Orchestration code

Compiler.
- Map circuits to instructions streams for control electronics.



Control electronics.
- Executes instructions.
- All circuits loaded run at full speed, no restart/reload required.

CLOPS progress



ibm_torino		
QPU status	Online	
Processor type	Heron r1	
Qubits	2Q error (best/layered)	CLOPS
133	1.53e-3/7.76e-3	210K

ibm_kingston		
QPU status	Online	
Processor type	Heron r2	
Qubits	2Q error (best/layered)	CLOPS
156	9.52e-4/3.90e-3	250K

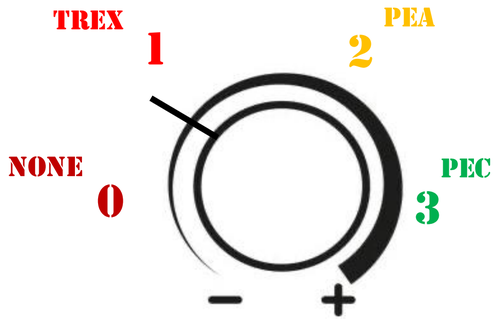
ibm_aachen		
QPU status	Online	
Processor type	Heron r2	
Qubits	2Q error (best/layered)	CLOPS
156	9.45e-4/4.37e-3	250K

Time to run the utility experiment

Software capabilities.

The evolution of primitives.

The evolution of the primitives.



Phase 1. Fully-automated mitigation.
Where we mitigate your circuits for you.

- Users selects resilience levels.
- Server does all the heavy lifting.

```
************************
* ZNE options:
*  mitigation: PEA
*  factors: [1, 1.2, 1.4]
*  extrapolator: linear
************************
```

Phase 2. Guided control.
Where you can tweak some parameters.

- Resilience levels still supported.
- Additionally, users can define custom options to meet their needs.

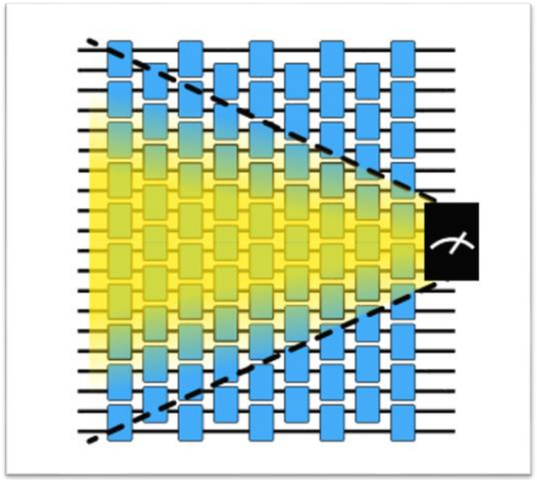
Adding a new feature.

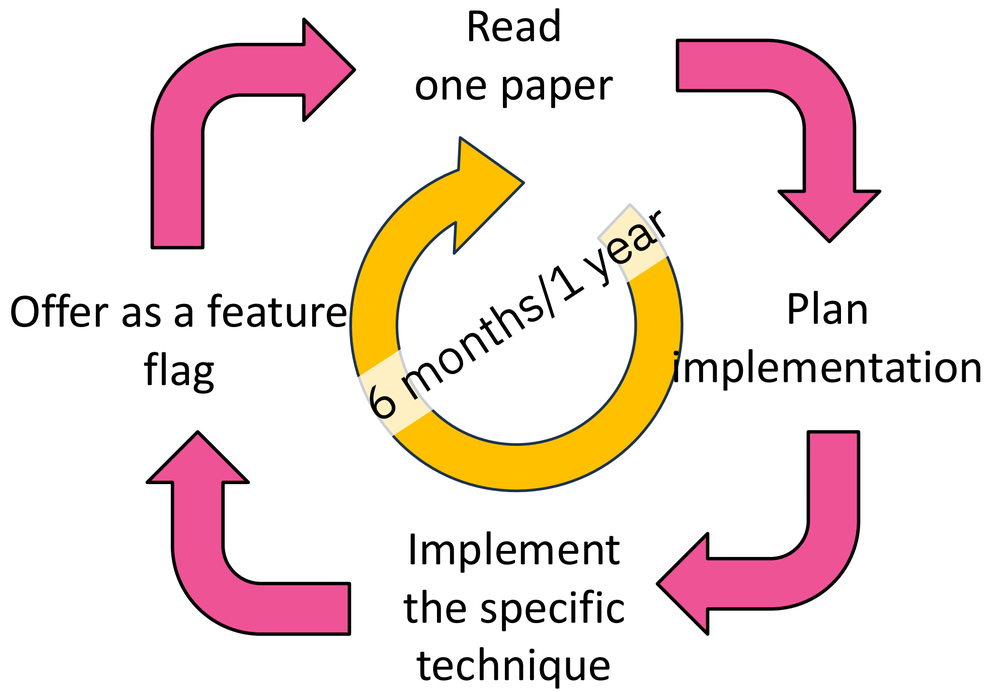
New feature request.

➢ PEC with shaded lightcones [1].

Reactions.

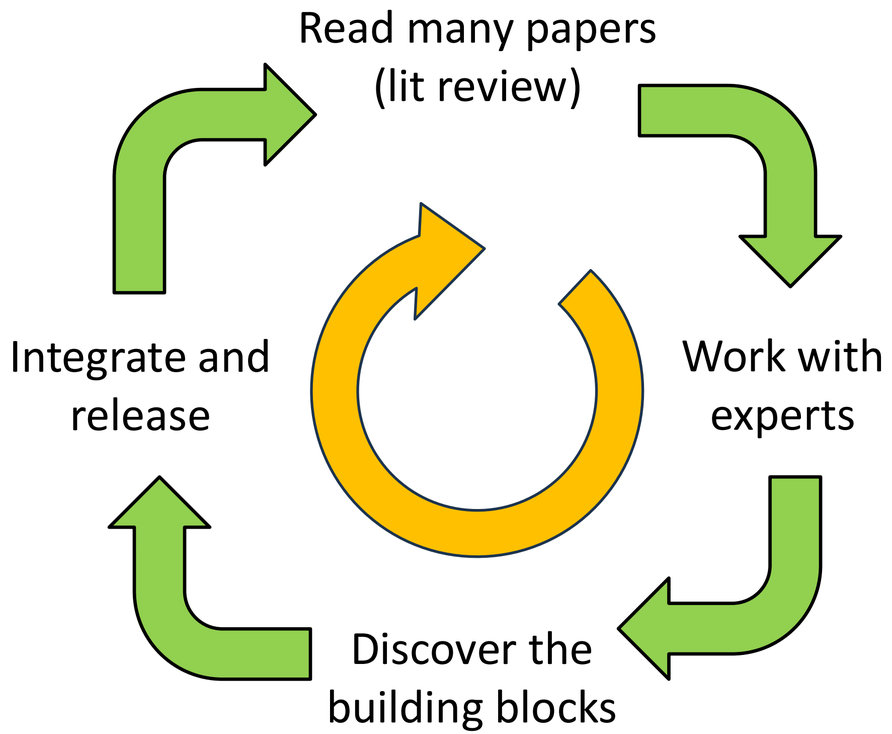
- What are we talking about?
- Should this be added on top of PEC?
- Should we just push back?





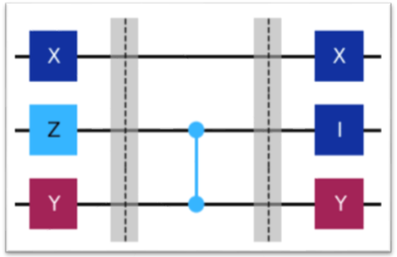
[1] Eddins, A. and others, *Lightcone shading for classically accelerated quantum error mitigation*, arXiv:240904401.

A healthy development cycle.

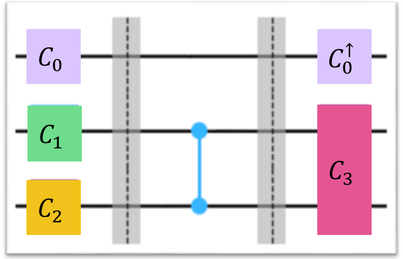


From features to building blocks

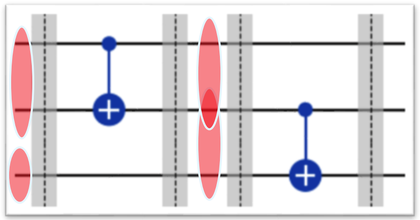
Pauli twirling.



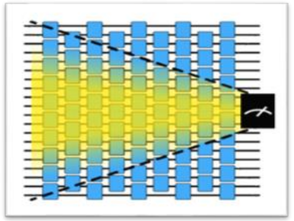
Advanced twirling.



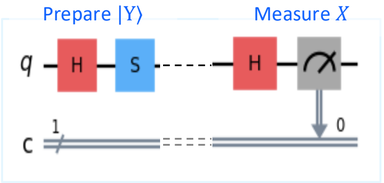
Noise injection.



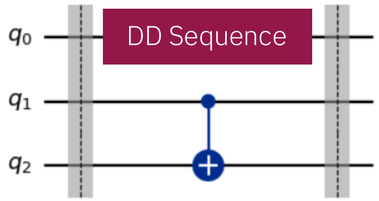
Opacity filters.



Changing basis.



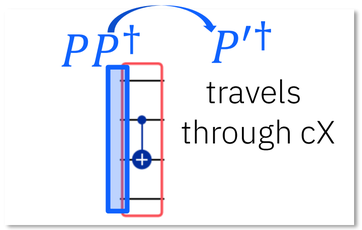
Dynamical decoupling.



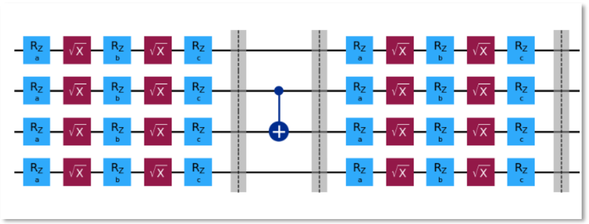
Conceptually, all these capabilities are described by **virtual gates** that:

- Are:
  - **generated** somewhere.
  - **propagated** through gates.
  - **collected** by some element of a template circuit.
- Can be of different types, e.g., Pauli, U2, one-qubit Clifford, …

Twirling on paper.



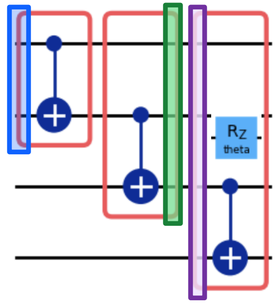
Twirling in practice.



Working towards a declarative execution

A new intermediate representation (IR)

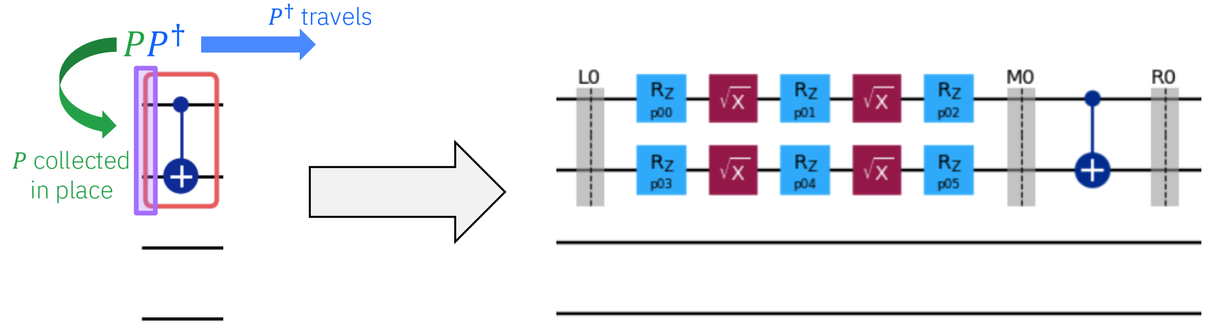
- Boxes introduced in Qiskit 2.0.
- Annotations introduced in Qiskit 2.1.
- Main ideas:
  - Use boxes to isolate subcircuits.
  - Use annotions to add virtual gates to boxes.
  - More details in Qiskit/RFC [1].




```
circuit = QuantumCircuit(4)

with circuit.box(annotations=[PauliTwirl], dressing="left"):
    circuit.cx(0, 1)

with circuit.box(annotations=[PauliTwirl], dressing="right"):
    circuit.noop(0)
    circuit.cx(1, 2)

with circuit.box(annotations=[PauliTwirl, InjectNoise(...)], dressing="left"):
    circuit.noop(0)
    circuit.rz(Parameter("theta"), 1)
    circuit.cx(2, 3)

circuit.measure_all()
```

PSEUDOCODE

Example. Pauli-twirled box.

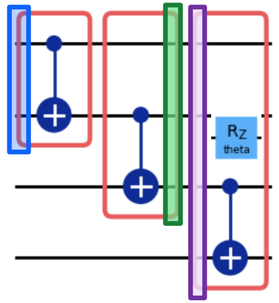


[1] https://github.com/Qiskit/RFCs/blob/master/0022-circuit-block.md

Working towards a declarative execution

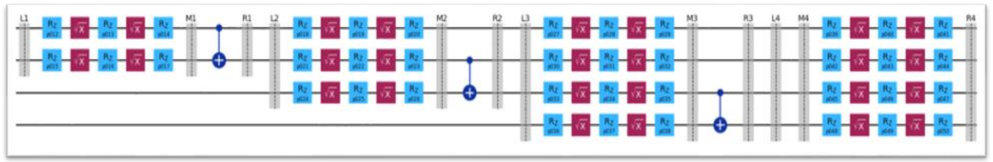
A new intermediate representation (IR)

- Boxes introduced in Qiskit 2.0.
- Annotations introduced in Qiskit 2.1.
- Main ideas:
  - Use boxes to isolate subcircuits.
  - Use annotions to add virtual gates to boxes.
  - More details in Qiskit/RFC [1].



An interpreter to convert from new IR to existing IR

- Returns:
  - A **template circuit** with parametrized gates.



  - A **“dag”** object to sample random parameters.

```
template, dag = build(circuit)
parameter_values = dag.sample(num_randomizations=100)

sampler_pubs = [(template, parameter_values)]

sampler = Sampler(backend)
job = sampler.run(sampler_pubs)
```

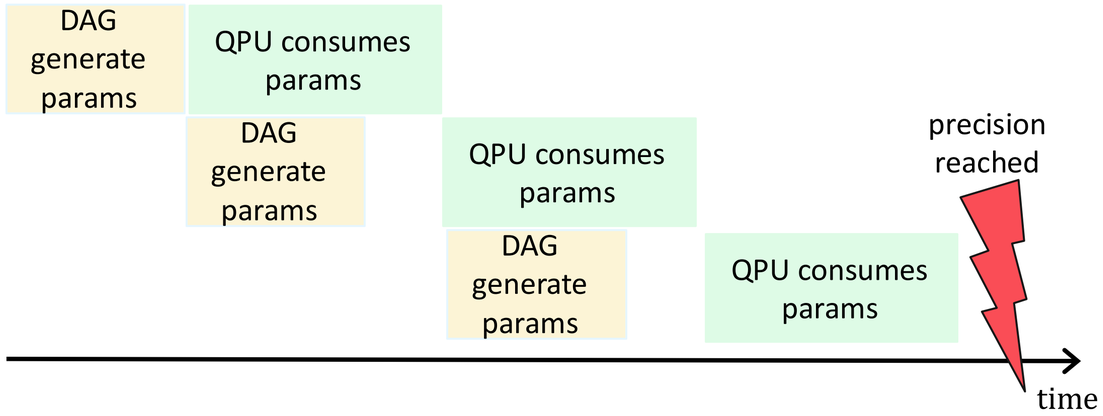
PSEUDOCODE

[1] https://github.com/Qiskit/RFCs/blob/master/0022-circuit-block.md

Using DAGs to improve performance

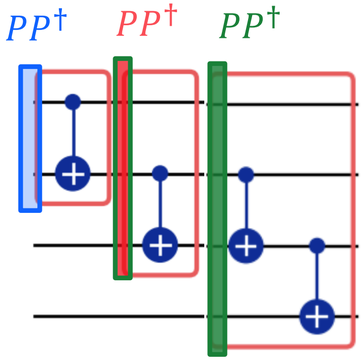
DAG unlocks new patterns for performance:

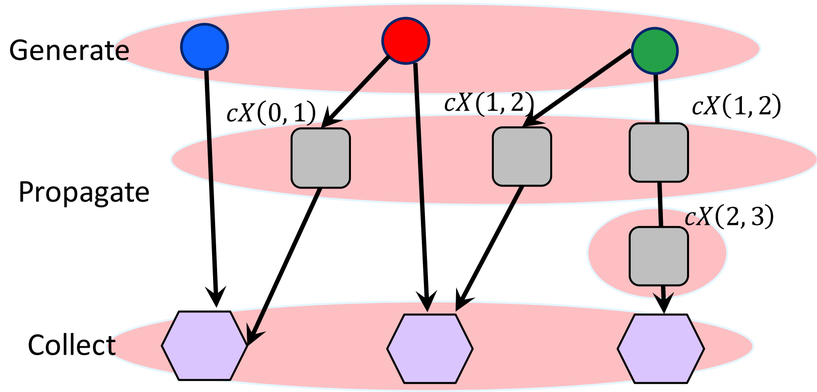
- Generating params on-the-fly to keep HW busy.
- Execute until target precision is reached.



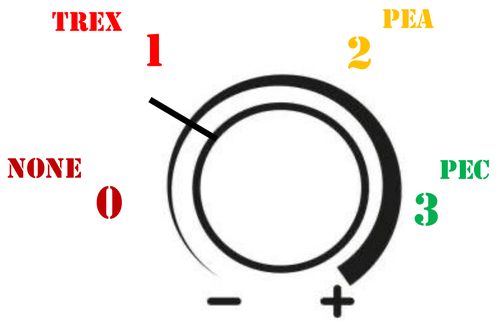
Dag needs to be faster than HW.

- Internal implementation as directed acyclic graph.





The evolution of the primitives.



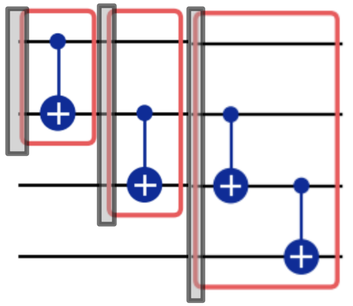
Phase 1. Fully-automated mitigation.
Where we mitigate your circuits for you.

- Users selects resilience levels.
- Server does all the heavy lifting.

```
**************************
* ZNE options:
*  mitigation: PEA
*  factors: [1, 1.2, 1.4]
*  extrapolator: linear
**************************
```

Phase 2. Guided control.
Where you can tweak some parameters.

- Resilience levels still supported.
- Additionally, users can define custom options to meet their needs.



Phase 3. Declarative execution.
Where users get near-complete control.

- IR to reason about virtual gates.
- The primitives provide building blocks to write new functionalities.
- New paths to performance to explore and benefit from.

Conclusions.

What does it take to turn a good error-mitigation paper into performant software?