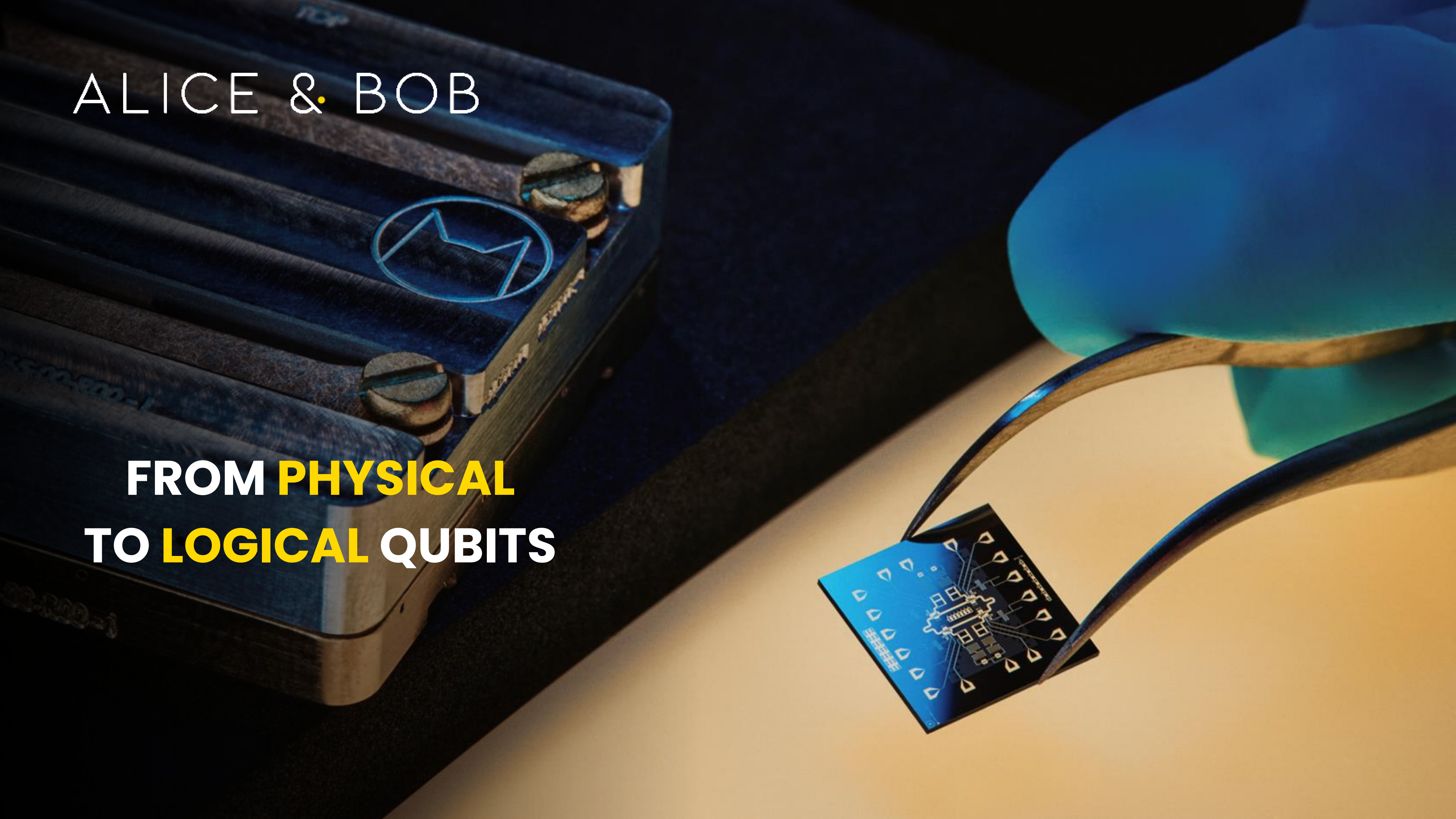
ALICE & BOB

FROM PHYSICAL TO LOGICAL QUBITS



Hi, I’m Laurent 👋

I’m a Product Manager at Alice & Bob

- Background in applied mathematics
- Product Manager for 13 years
- Setting up a cloud access to Alice & Bob’s quantum computers

Alice & Bob: building a universal fault-tolerant quantum computer

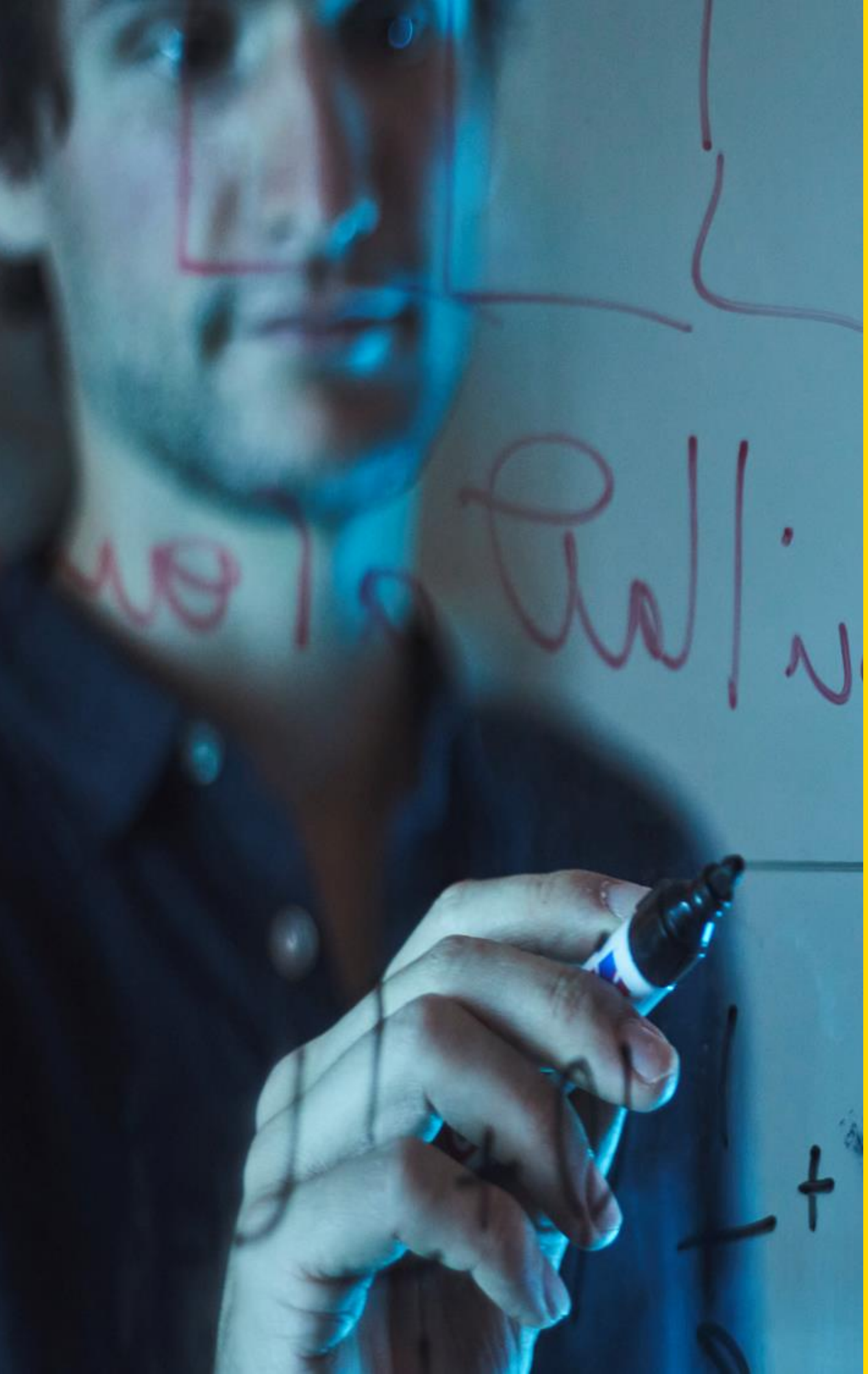
THÉAU PERONNIN
Co-founder & CEO

PhD in Quantum Physics at ENS
Graduated from École Polytechnique
Expert in modular quantum architecture

RAPHAËL LESCANNE
Co-founder & CTO

PhD in Quantum Physics at ENS
Graduated from ENS Ulm
Co-inventer of the cat qubit technology

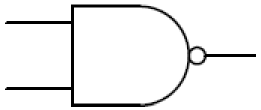
Created in 2020	90 employees today (incl. 50 R&D)	30M€ raised in VC capital	>15 patents filed	>15 academic partnerships



01

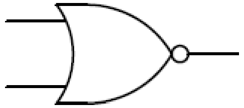
The need for error correction

Classical computing



NAND

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

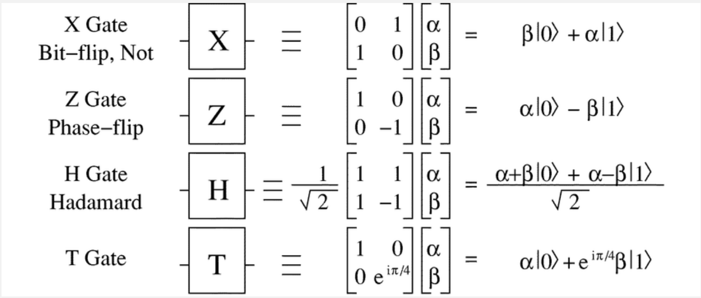


NOR

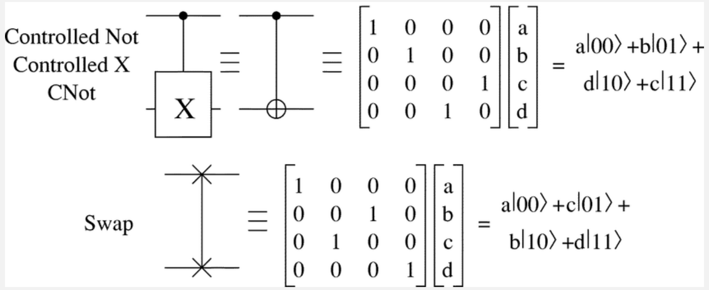
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

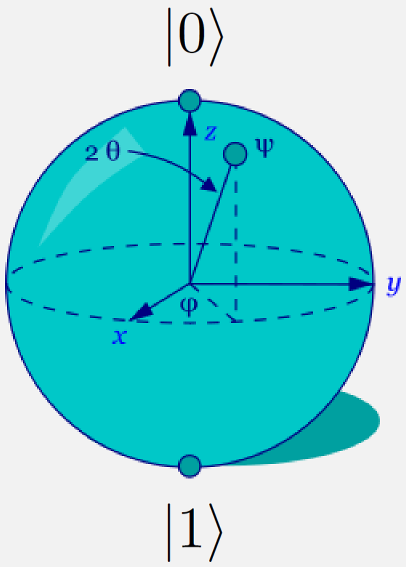
Quantum computing

1-qubit gates



2-qubit gates





Quantum computers are SLOW!

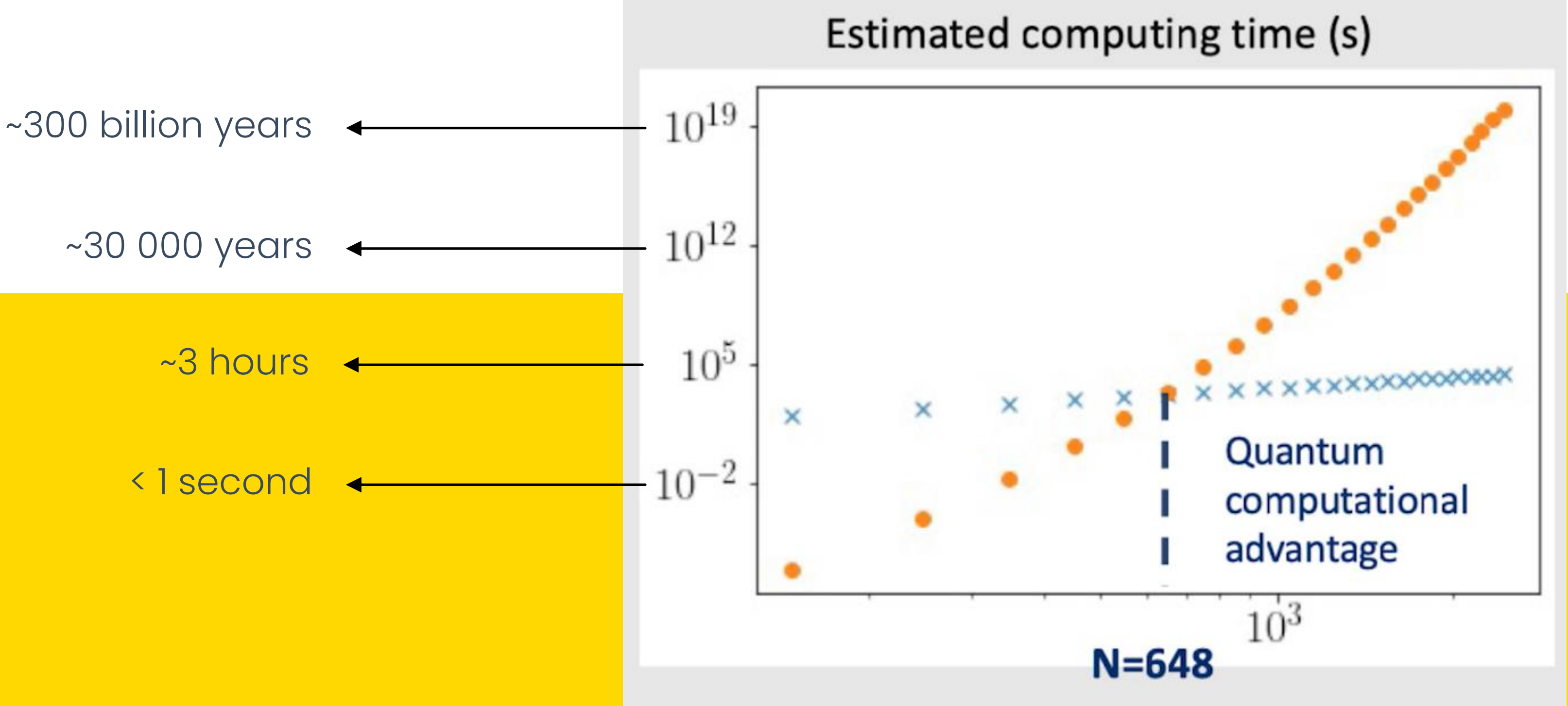
We compare the peak performance of a single classical chip that can be manufactured today (like an NVIDIA A100 GPU, or an ASIC with a similar number of transistors) with a future quantum computer with 10,000 error-corrected logical qubits, 10μs gate time for logical operations and all-to-all connectivity. We consider an estimate of the I/O bandwidth (namely the number of operations per second) and three types of operations: logical binary operations, 16-bit floating point, 32-bit integer or fixed-point arithmetic multiply add operations.

	GPU	ASIC	Future Quantum
I/O Bandwidth	10,000 Gbit/s	10,000 G/s	1 Gbit/s
Operation throughput			
16-bit floating point	195 Top/s	550 Top/s	10.5 kop/s
32-bit integer	9.75 Top/s	215 Top/s	0.83 kop/s
binary (Boolean logical)	4,992 Top/s	77,000 Top/s	235 kop/s

https://cacm.acm.org/research/disentangling-hype-from-practicality-on-realistically-achieving-quantum-advantage/

Quantum advantage depends on problem size

Example : time required to find the prime factors of a N-bit integer



Source: Alexia Auffèves (CNRS) - The energy and environmental challenges of quantum computing (June 2022)

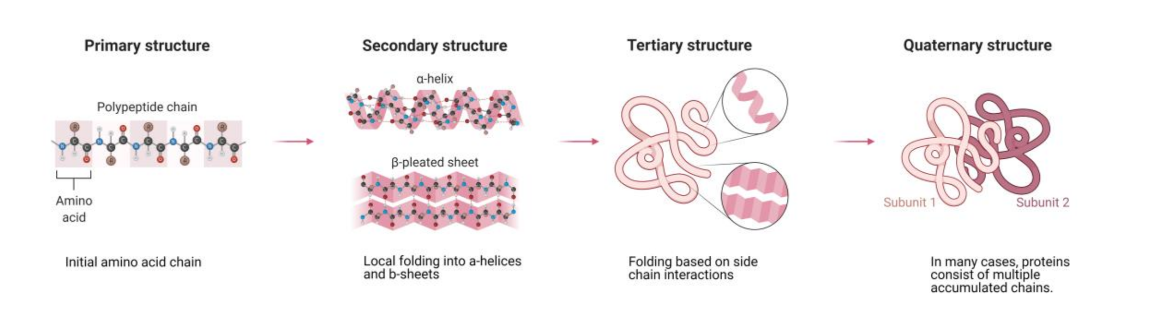
We require large speedups

For example, the polynomial speedup of a quantum Monte Carlo is eaten away by differences in execution velocity.

SUPERPOLYNOMIAL SPEEDUP	POLYNOMIAL SPEEDUP
Quantum simulation for chemistry	Exponential congruence
ODE and PDE	Subset sum
Shor	Grover (search)
Discrete-log	Constraint Satisfaction

Modeling antibody loops on a quantum computer
Quantum Monte Carlo
Roche & Tencent
https://arxiv.org/pdf/2105.09690v1.pdf

	n_{steps}	t_{step}	Total time	Logical qubits
Classical [65]	4×10^9	2.3×10^{-4}s	10.6 days	-
Quantum (parallel)	8.9×10^4	16.4 mins	2.8 years	$\sim 4 \times 10^4$
Quantum (serial)	8.9×10^4	25.2 hours	256 years	$\sim 10^4$



Exponential acceleration requires deep circuits

Running Shor's algorithm on an N bit key requires:

- O(N) qubits
- $O(N^3)$ circuit depth

These are $O(N^4)$ opportunities to get an error

If:

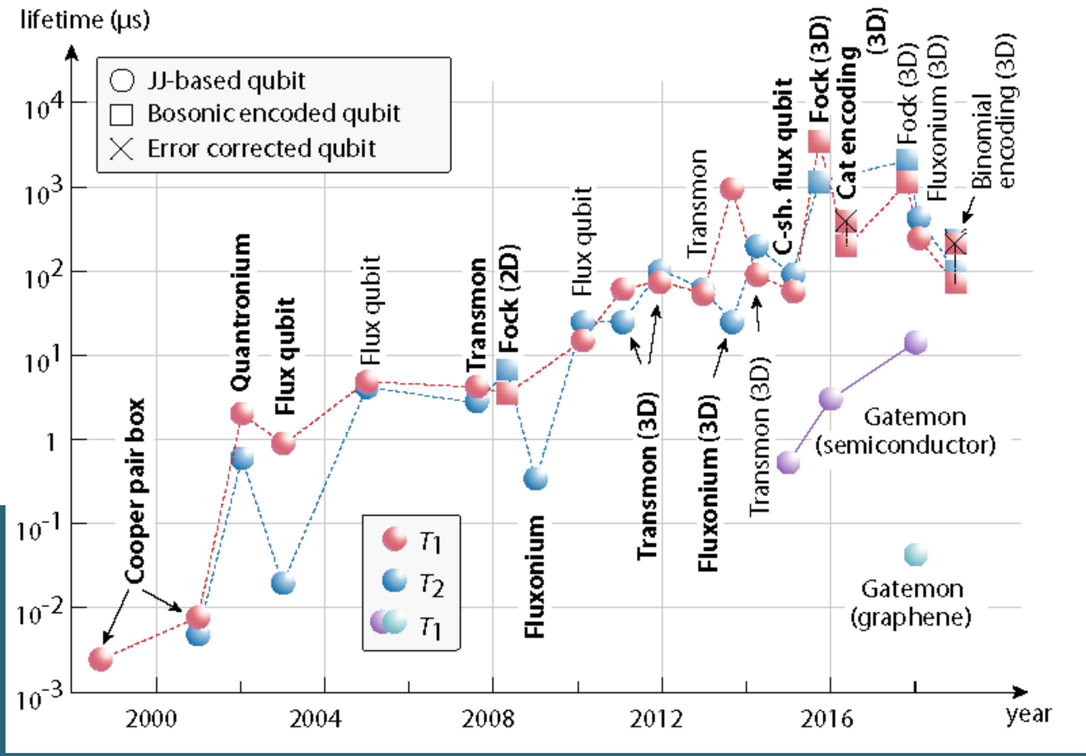
- A step fails with probability p
- We want a 10% probability to get the right result

Then we want $\mathbf{(1-p)^{N^4} > 10^{-1}}$.

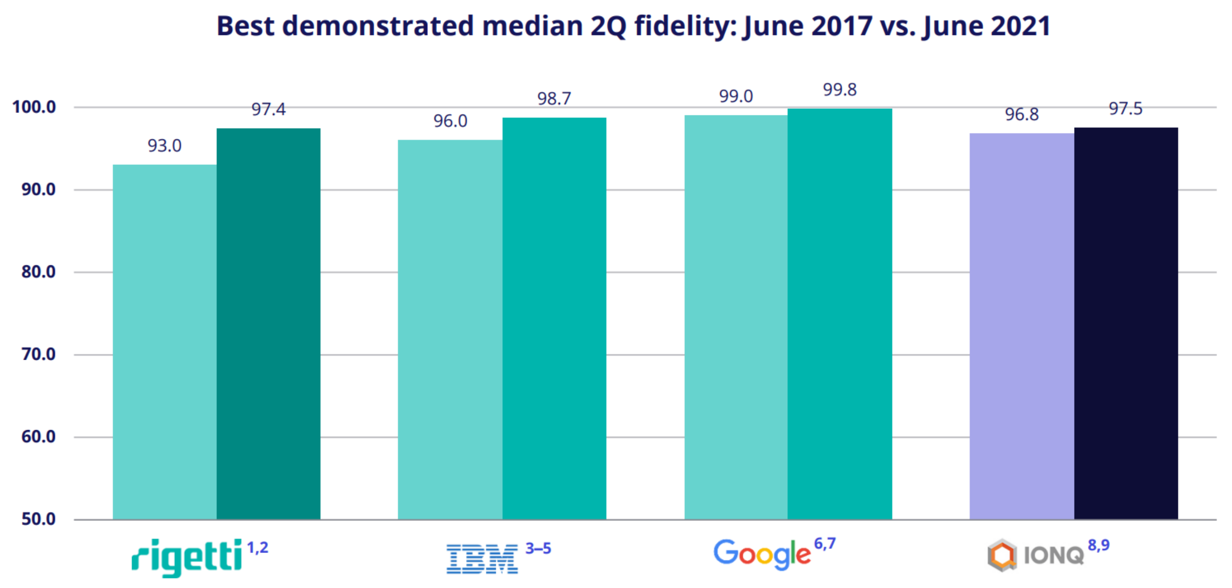
RSA2048 $p < 10^{-12}$

Deep circuits require low error rates

Qubits are very far from that fidelity



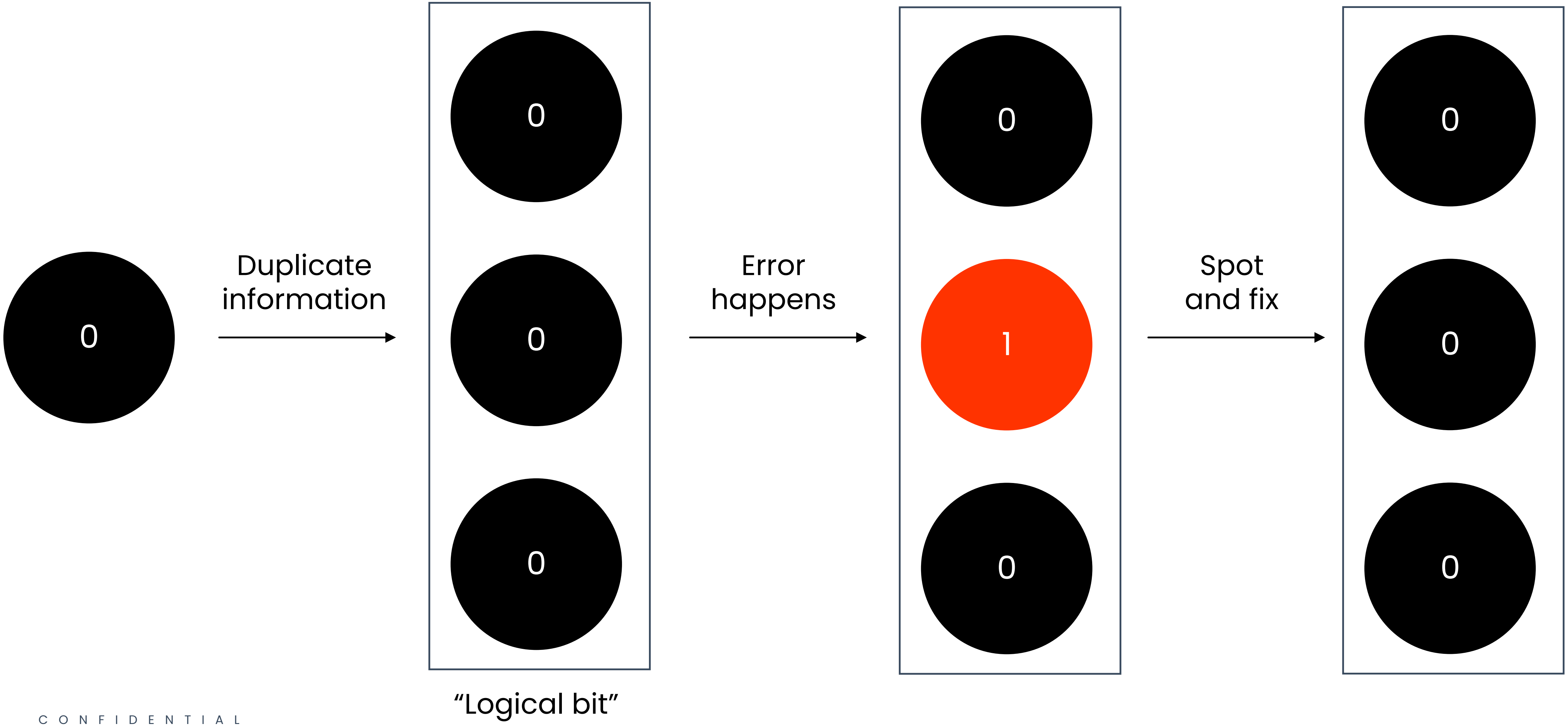
M. Kjaergaard, W.D. Oliver et al., Annual Review of Condensed Matter Physics, 2019.



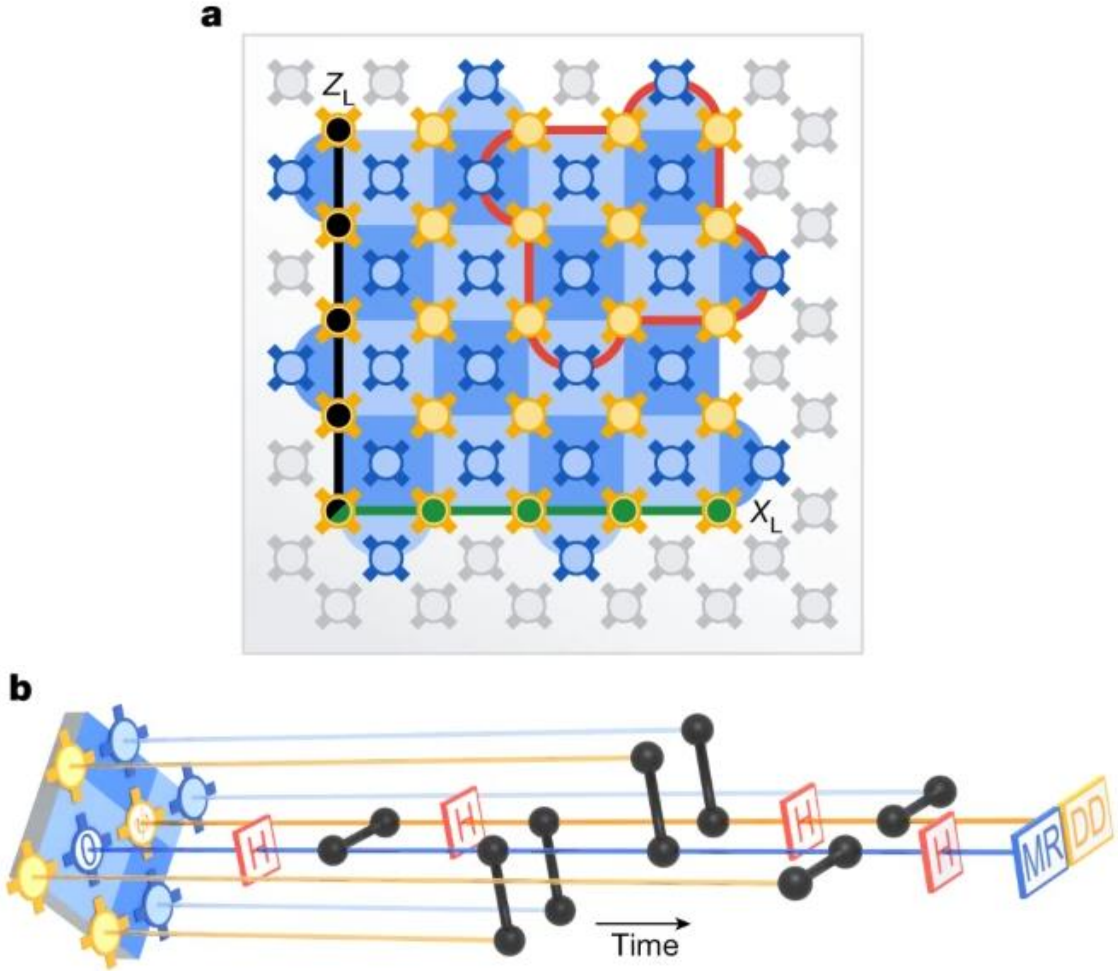
Rigetti investor presentation
October 2021

So, we need error correction

Or “fault-tolerance”



And it's even harder in quantum computing



- You can't access the state of a qubit during computation
- You can't duplicate quantum information
- There are two types of errors to correct
- Error correction operations have a high probability of adding errors

The road to a logical fault-tolerant quantum computer

1. Physical qubits
- Create and stabilize qubits
- Perform physical gates

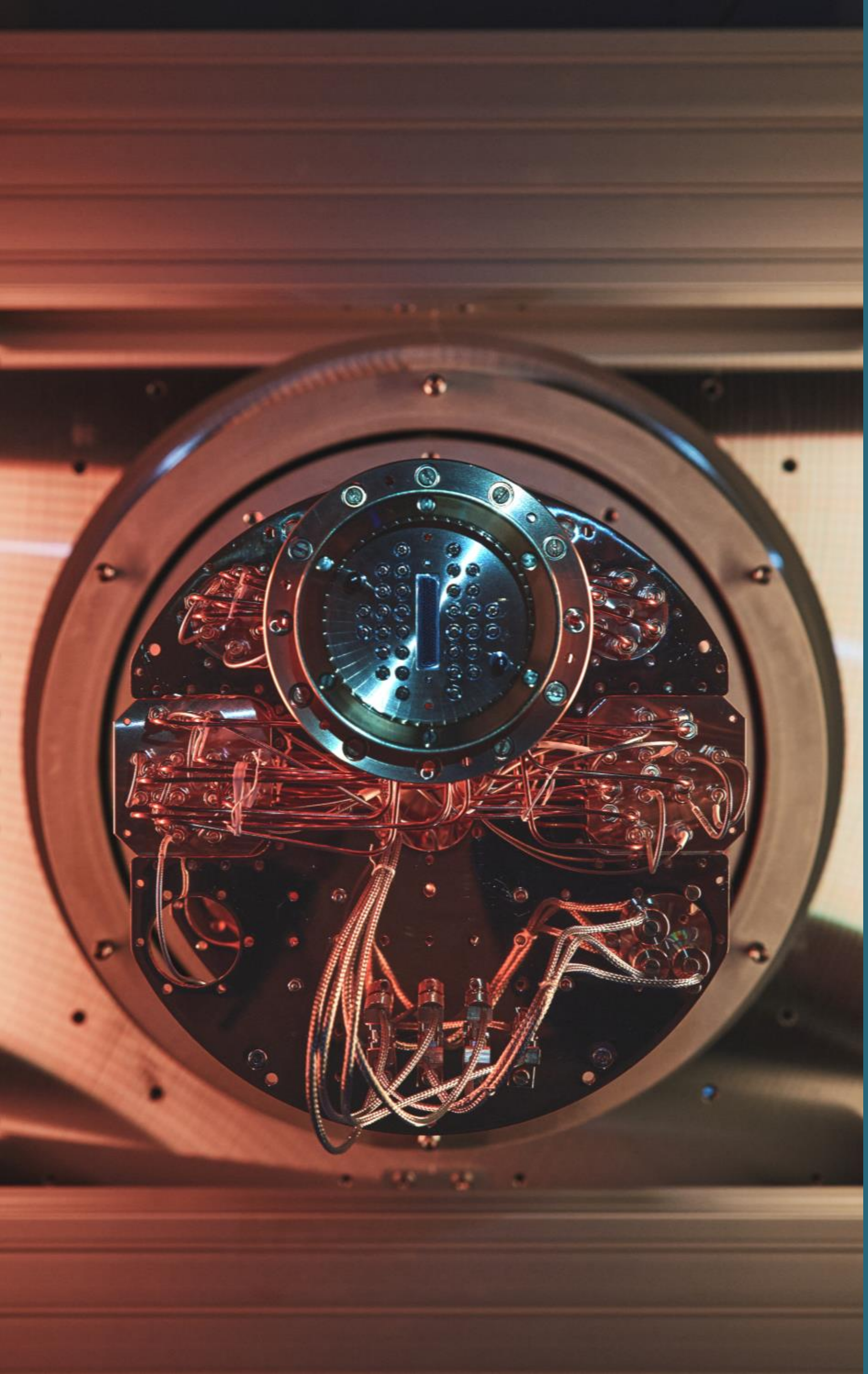
← Everybody is here

2. Logical qubits (= fault-tolerance)
- Implement quantum error correction
- Improve hardware fidelity as required

← **The new big race!**

3. Logical operations (= universality)
- Transversal gates
- Non-transerval gates (magic states)

← Several years in the future



02

Build physical qubits

The cat qubit: Alice & Bob's choice

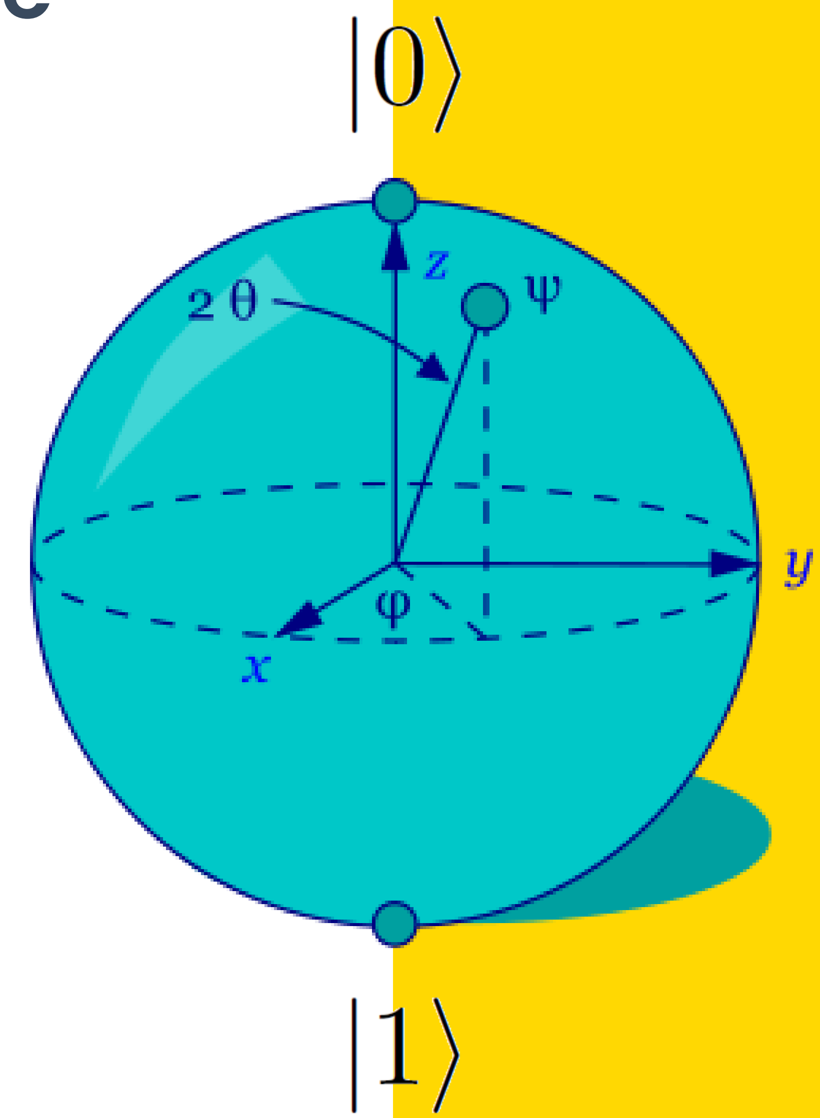
Two types of errors:

1. **Bit-flip** (error on Θ)
2. **Phase-flip** (error on φ)

Cat qubits exhibit:

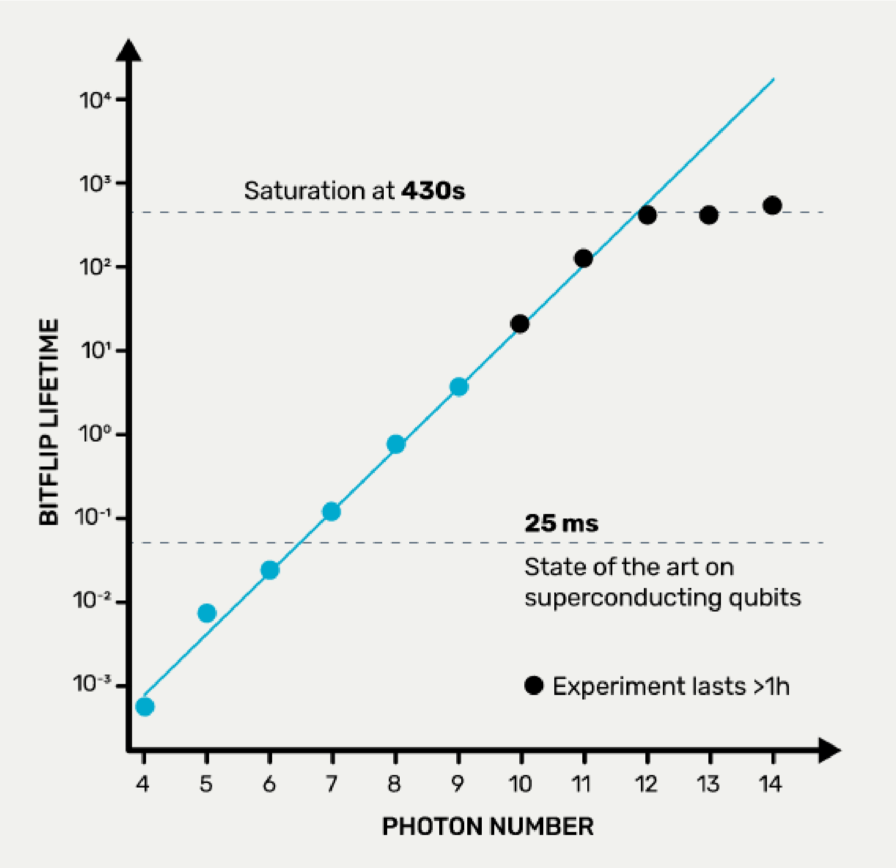
1. A lot **fewer bit-flips** (~10^6)*
2. A little **more phase-flips** (~10^2)*

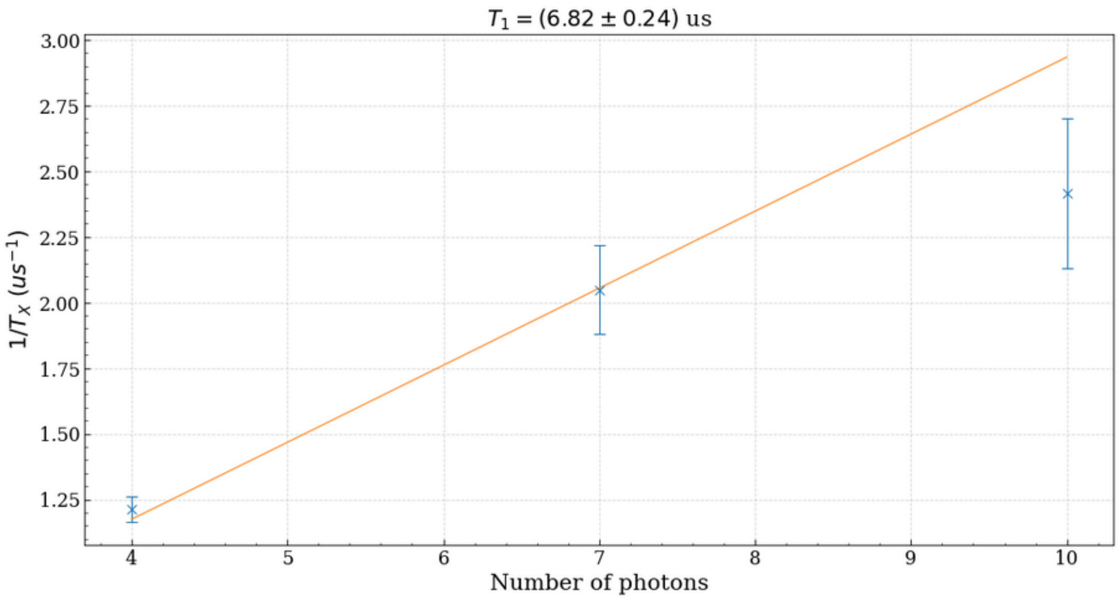
* : Boson 4 vs. a transmon with lifetime ~10^{-4} s



More details: Lescanne et al., Nature Physics 2020
https://doi.org/10.1038/s41567-020-0824-x

The noise bias of cat qubits





Bit-flip lifetime improves exponentially

Phase-flip lifetime decreases linearly

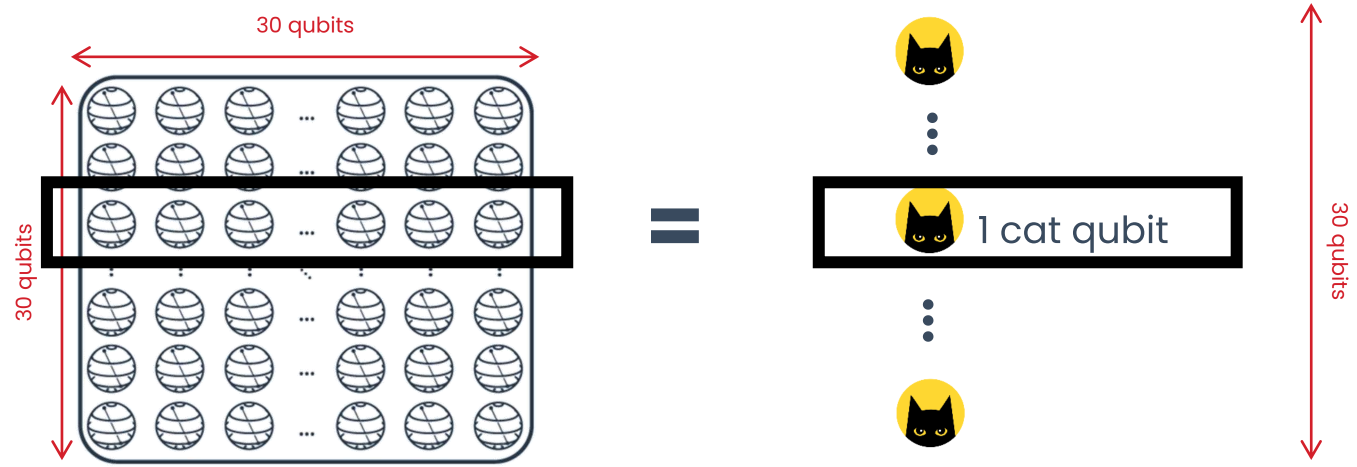
Figures can be reproduced using https://console.cloud.google.com/marketplace/product/cloud-prod-0/felis-cloud

Cat qubits

The perfect basis for logical qubits

QUANTITATIVE APPROACH
STANDARD QUBITS + SURFACE CODE

QUALITATIVE APPROACH
CAT QUBITS + REPETITION CODE



Shor to break RSA

20M
physical qubits

C. Gidney et al. 2019

vs

350k
cat qubits

E. Gouzien et al. 2022

or

100k
cat qubits using LDPC codes

D. Ruiz et al. 2024

vs Google

1 ≈ 49

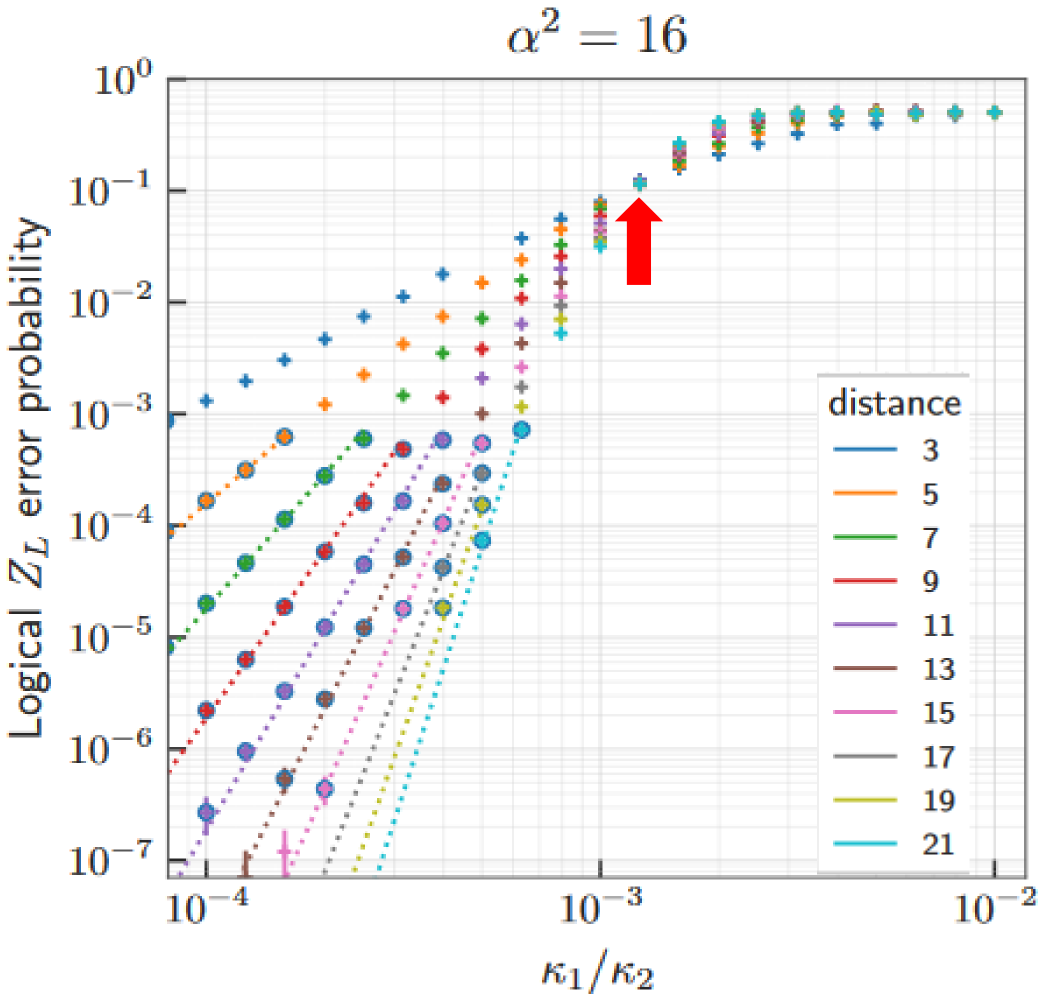
A&B cat qubit ≈ Google physical qubits

vs amazon

100% of the scientific articles cited are from A&B

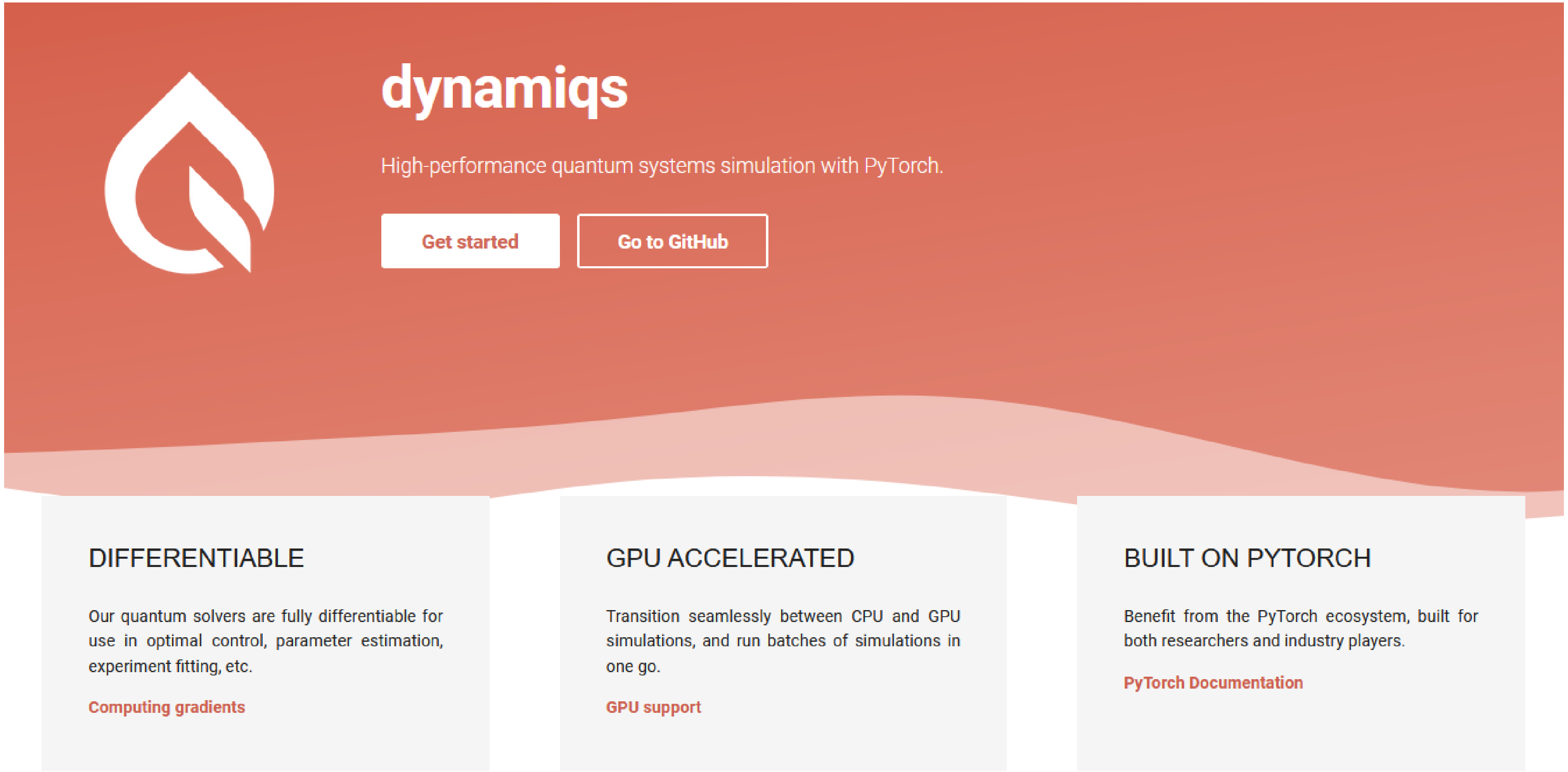
151x mentions of A&B technology

Next goal: take qubits below the threshold



- Logical operations fidelity is mainly influenced by the underlying physical qubit quality
- There exists a magic point, called the threshold, below which the error rate of the logical operations decreases drastically (exponentially)
- We want to reach that point!

Tool #1: physical simulation



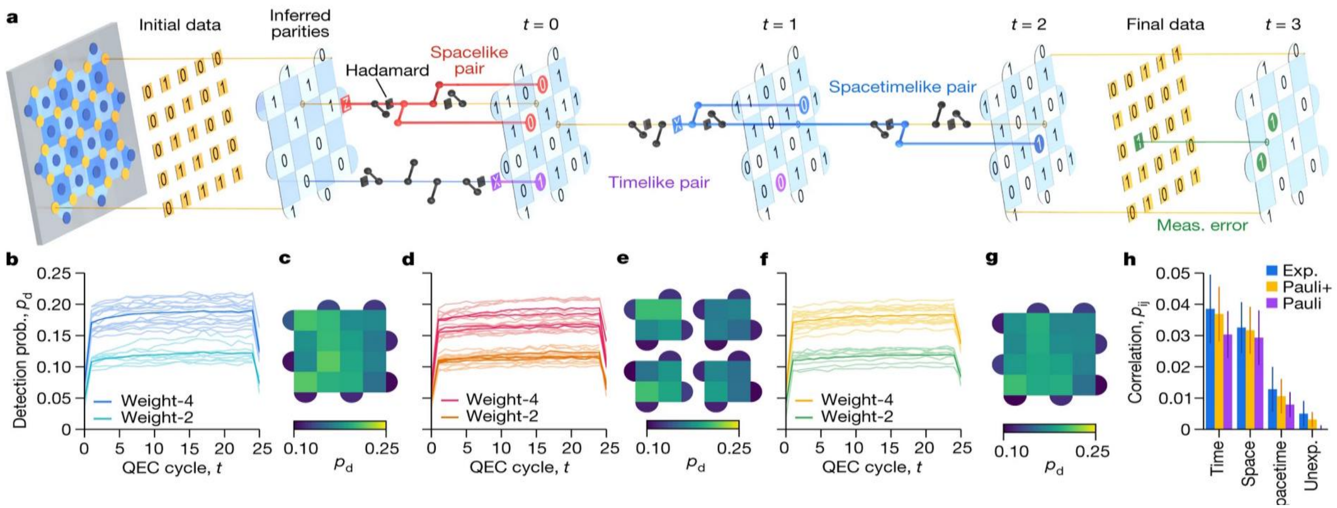
https://www.dynamiqs.org

03

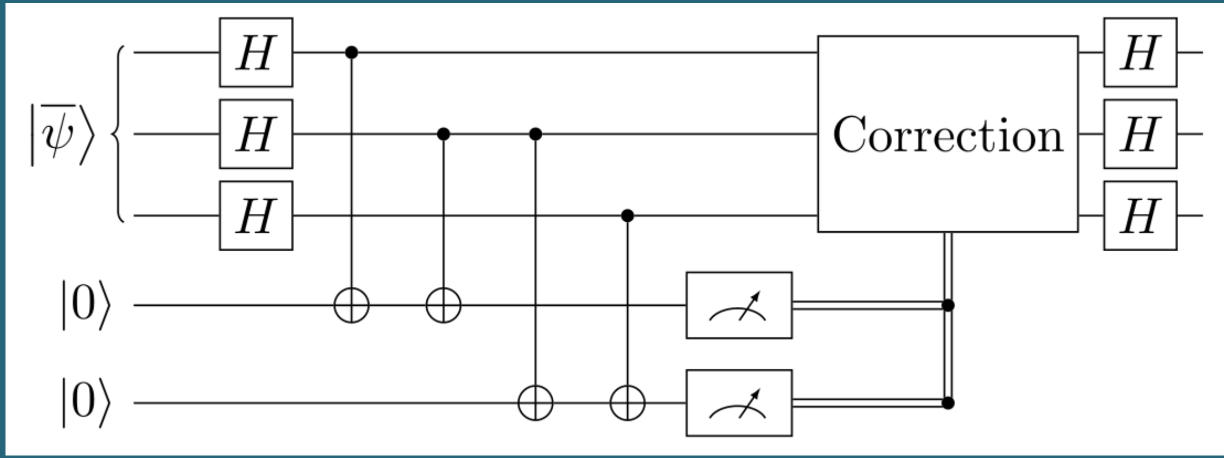
Build logical qubits

Error correction creates logical qubits

Biased vs. unbiased noise

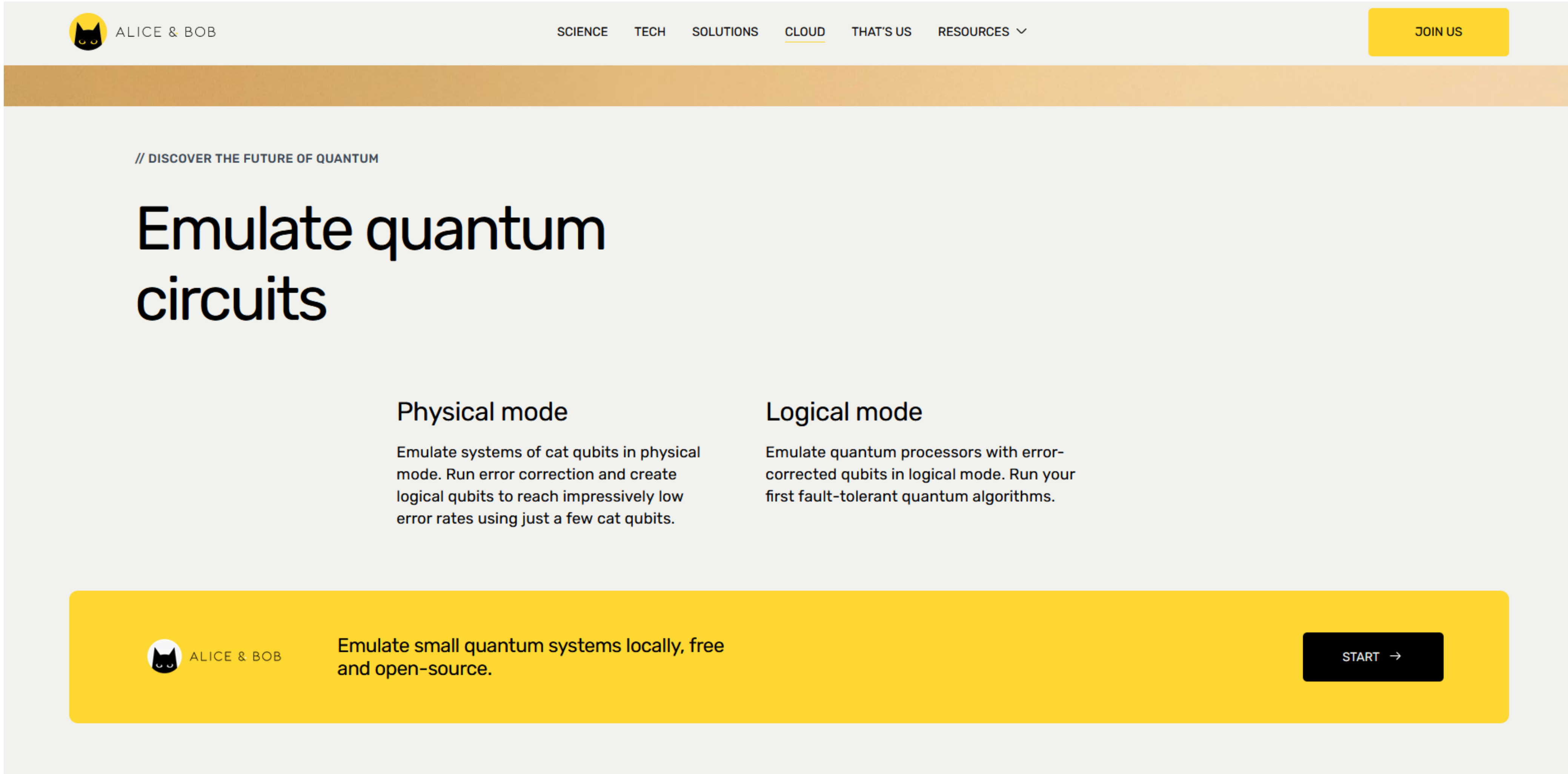


Unbiased noise:
Requires a complex 3D
surface code



Strongly biased noise:
Requires only a 2D
repetition code

Tool #2: gate-level emulation



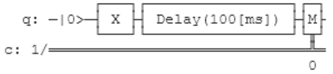
https://github.com/Alice-Bob-SW/qiskit-alice-bob-provider

Felis key features

Exponential suppression of bit flips

```
[26]: circ = QuantumCircuit(1, 1)
      circ.reset(0)
      circ.x(0)
      circ.delay(100, 0, unit='ms')
      circ.measure(0, 0)
      circ.draw()
```

[26]:



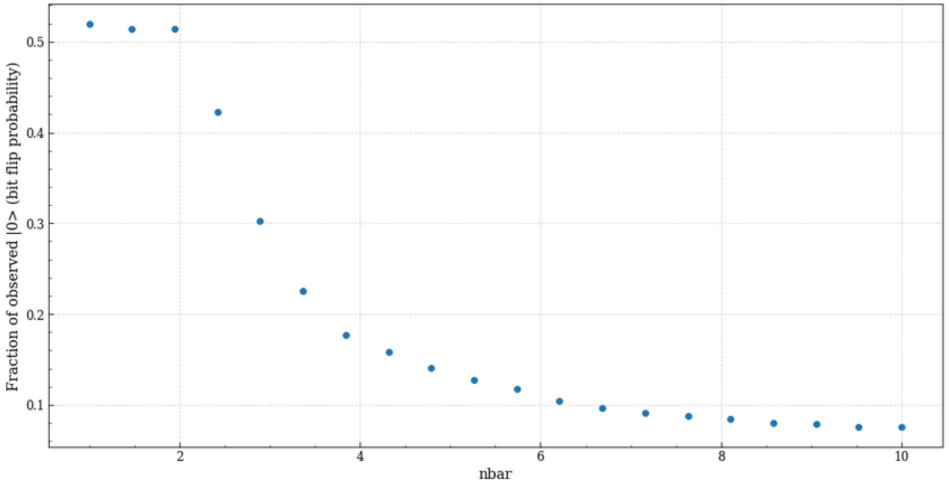

```
[13]: nbars = np.linspace(1, 10, 20)
      shots = 100000
```

```
[14]: jobs = []
      for nbar in nbars:
          jobs.append(execute(circ, backend, shots=shots, average_nb_photons=nbar))
```

```
[15]: results = [job.result() for job in tqdm(jobs)]
```

```
100%|██████████| 20/20 [00:17<00:00,  1.16it/s]
```

```
[16]: plt.figure()
      plt.scatter(nbars, [r.get_counts()['0'] / shots for r in results])
      plt.xlabel('nbar')
      plt.ylabel('Fraction of observed |0> (bit flip probability)')
      plt.show()
```



Access to real hardware

- Benchmark record-breaking cat qubits
- Reproduce Alice & Bob's experiments

Emulation capabilities

- Study error correction w/ physical backends
- Run algorithms w/ logical backends

Supported by Qiskit

- Quantum computing's most popular framework

Tool #3: Clifford simulation

Preview Code Blame 128 lines (100 loc) · 6.65 KB Raw

Stim

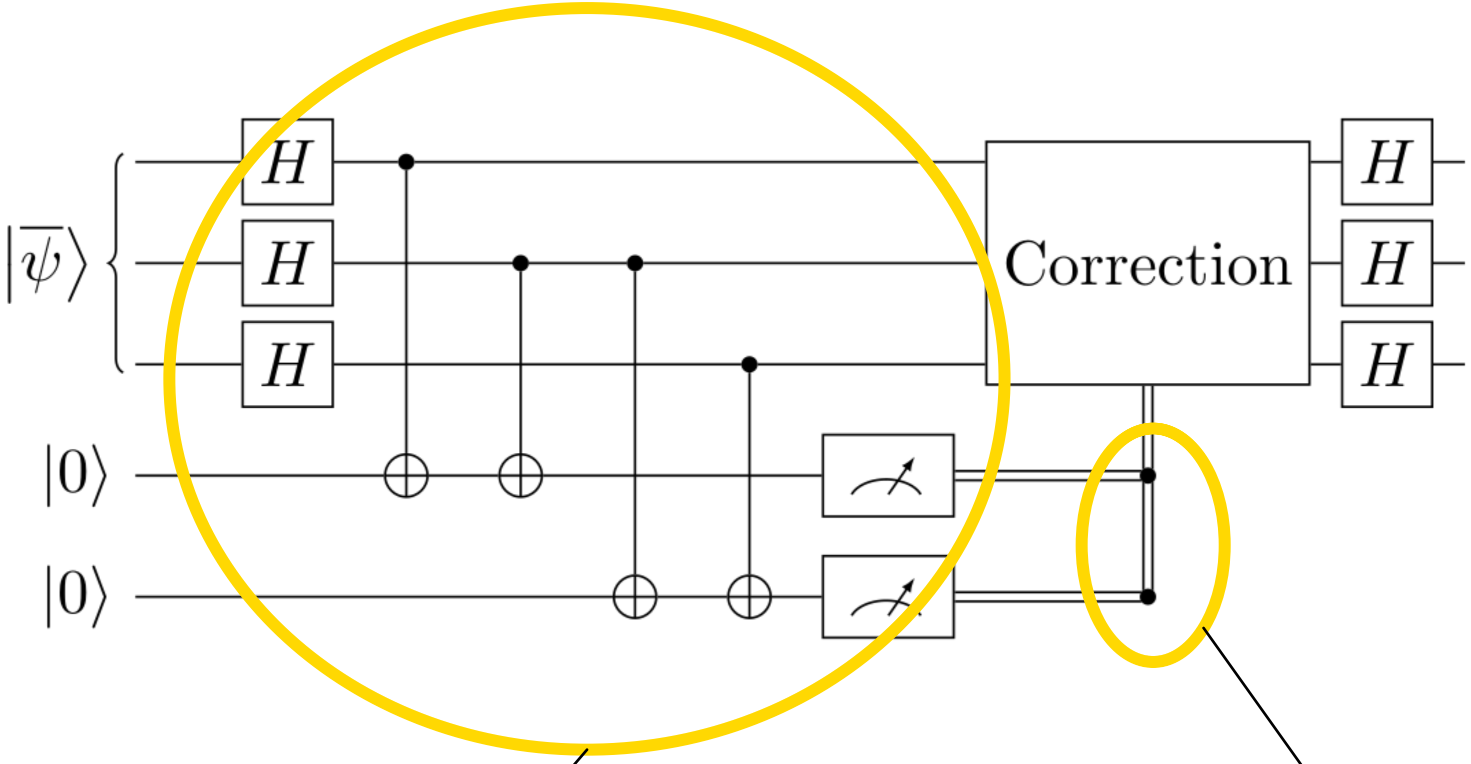
- What is Stim?
- How do I use Stim?
- How does Stim work?
- How do I cite Stim?
- subproject: Sinter decoding sampler
- subproject: Crumble interactive editor

What is Stim?

Stim is a tool for high performance simulation and analysis of quantum stabilizer circuits, especially quantum error correction (QEC) circuits. Typically Stim is used as a python package (`pip install stim`), though stim can also be used as a command line tool or a C++ library.

https://github.com/quantumlib/Stim/blob/main/README.md#how-cite-stim

Measurements require decoding



These operations are noisy

What decision rule here?

Tool #4: QEC decoding

README | Apache-2.0 license

PyMatching 2

ci passing | codecov 100% | docs passing | pypi package 2.2.0 | SUPPORTED BY UNITARY FUND

PyMatching is a fast Python/C++ library for decoding quantum error correcting (QEC) codes using the Minimum Weight Perfect Matching (MWPM) decoder. Given the syndrome measurements from a quantum error correction circuit, the MWPM decoder finds the most probable set of errors, given the assumption that error mechanisms are *independent*, as well as *graphlike* (each error causes either one or two detection events). The MWPM decoder is the most popular decoder for decoding surface codes, and can also be used to decode various other code families, including subsystem codes, honeycomb codes and 2D hyperbolic codes.

Version 2 includes a new implementation of the blossom algorithm which is **100-1000x faster** than previous versions of PyMatching. PyMatching can be configured using arbitrary weighted graphs, with or without a boundary, and can be combined with Craig Gidney's Stim library to simulate and decode error correction circuits in the presence of circuit-level noise. The sinter package combines Stim and PyMatching to perform fast, parallelised monte-carlo sampling of quantum error correction circuits.

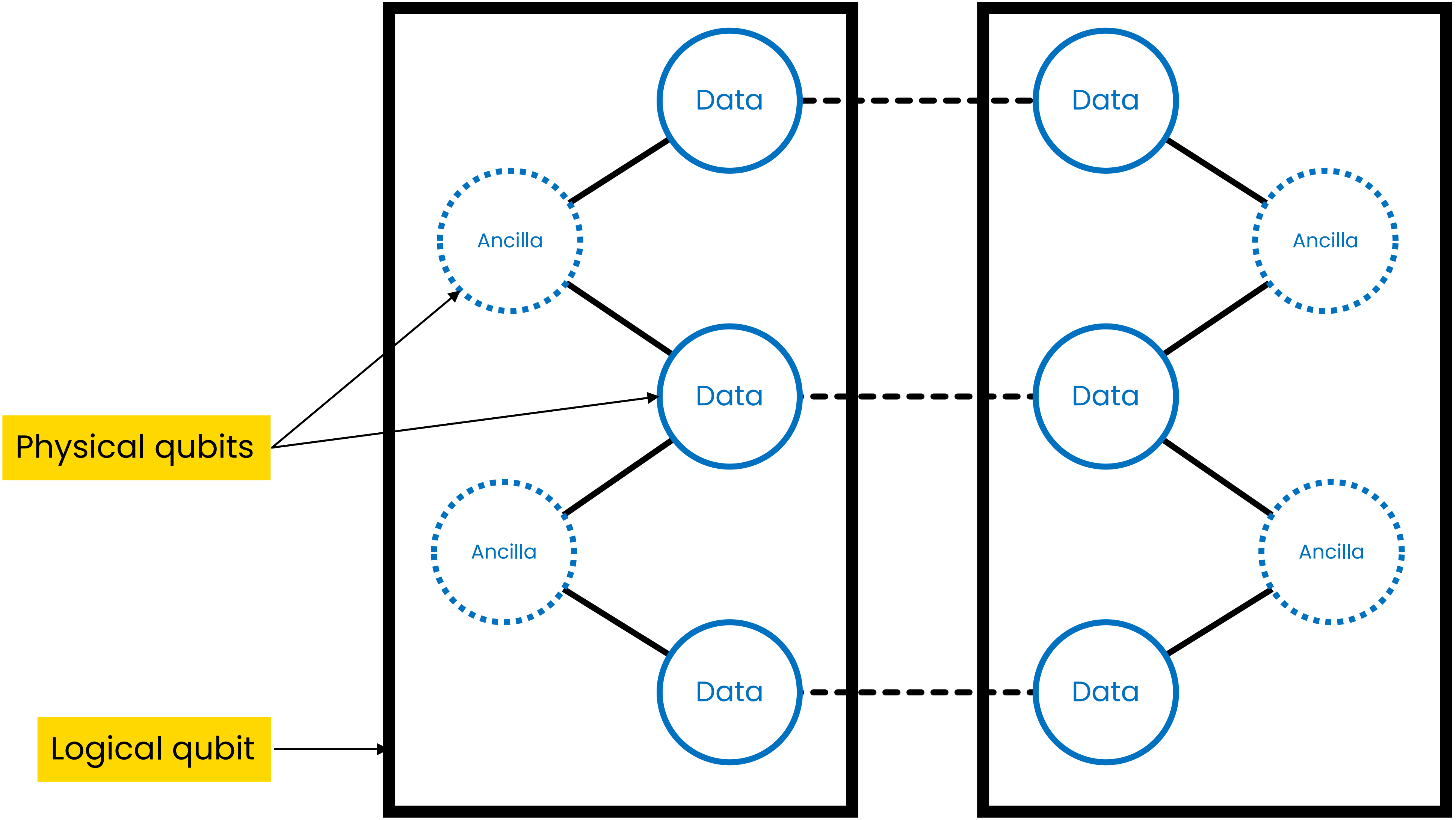
Documentation for PyMatching can be found at: pymatching.readthedocs.io

https://github.com/oscarhiggott/PyMatching

04

Build logical gates

A transversal gate



But transversal gates aren't enough…

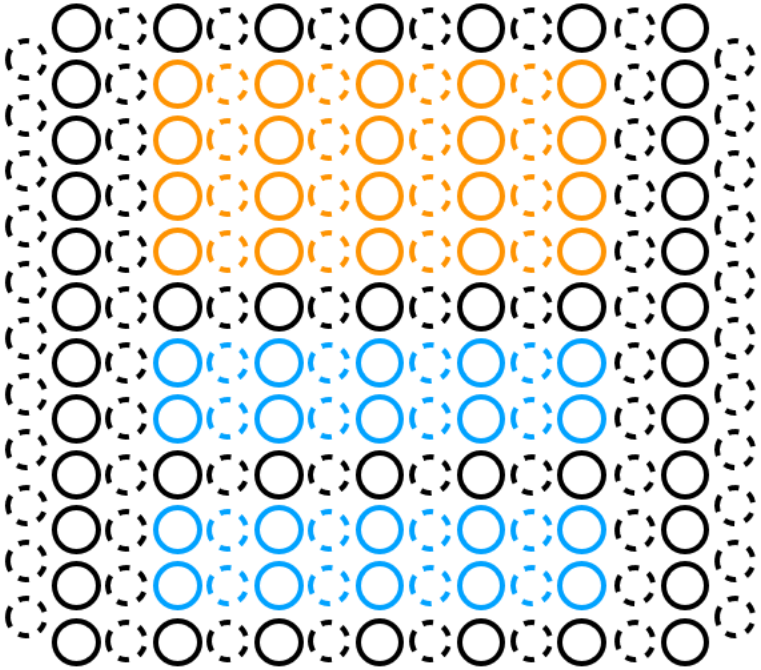
Compiling on "logical" hardware

↓

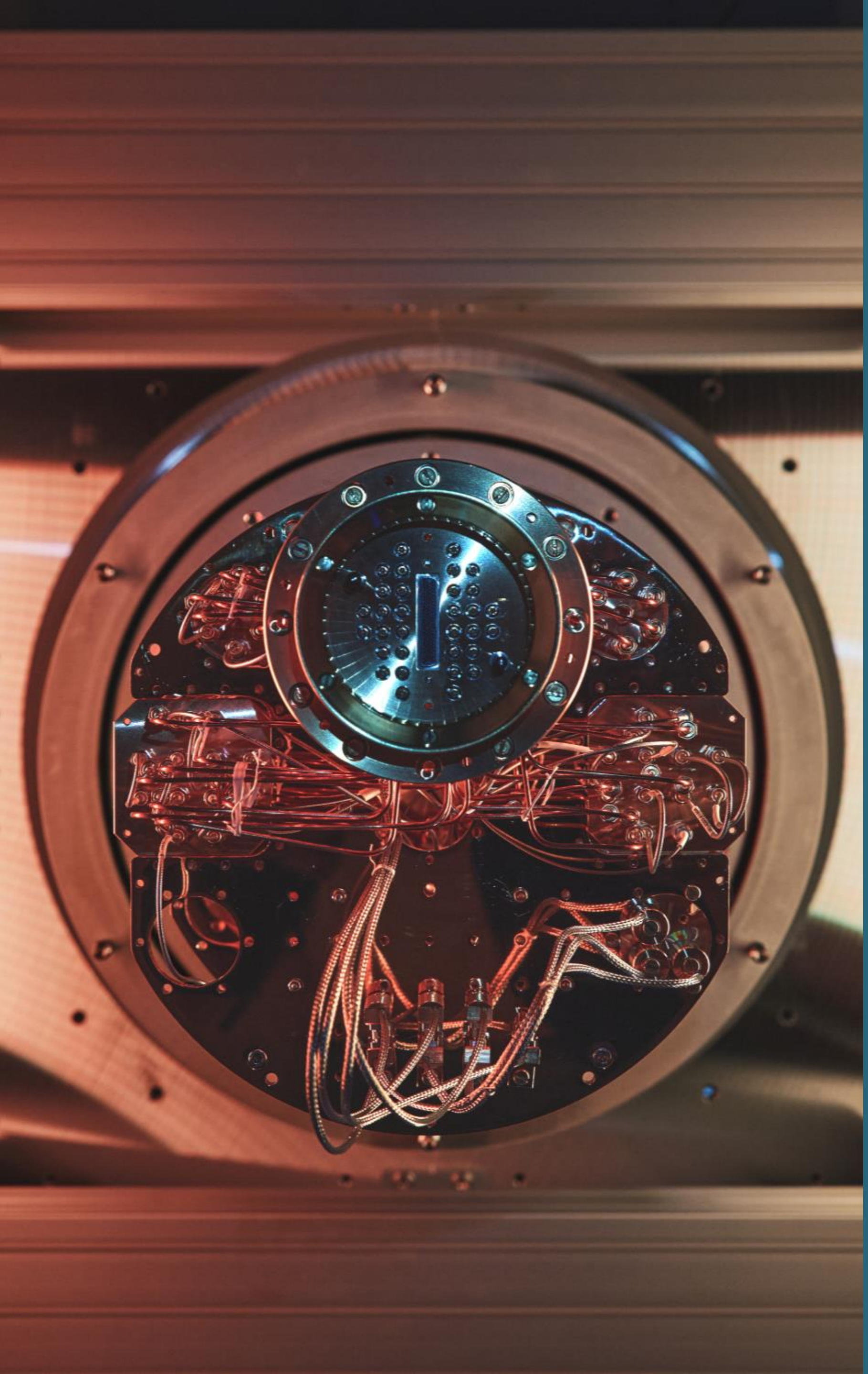
Compiling on "physical" hardware

A 4-logical, 178-physical qubit processor

- data qubit
- ancilla qubit
- Toffoli magic state factory qubit
- Computational qubit
- Routing qubit



Logical qubit (distance-5)



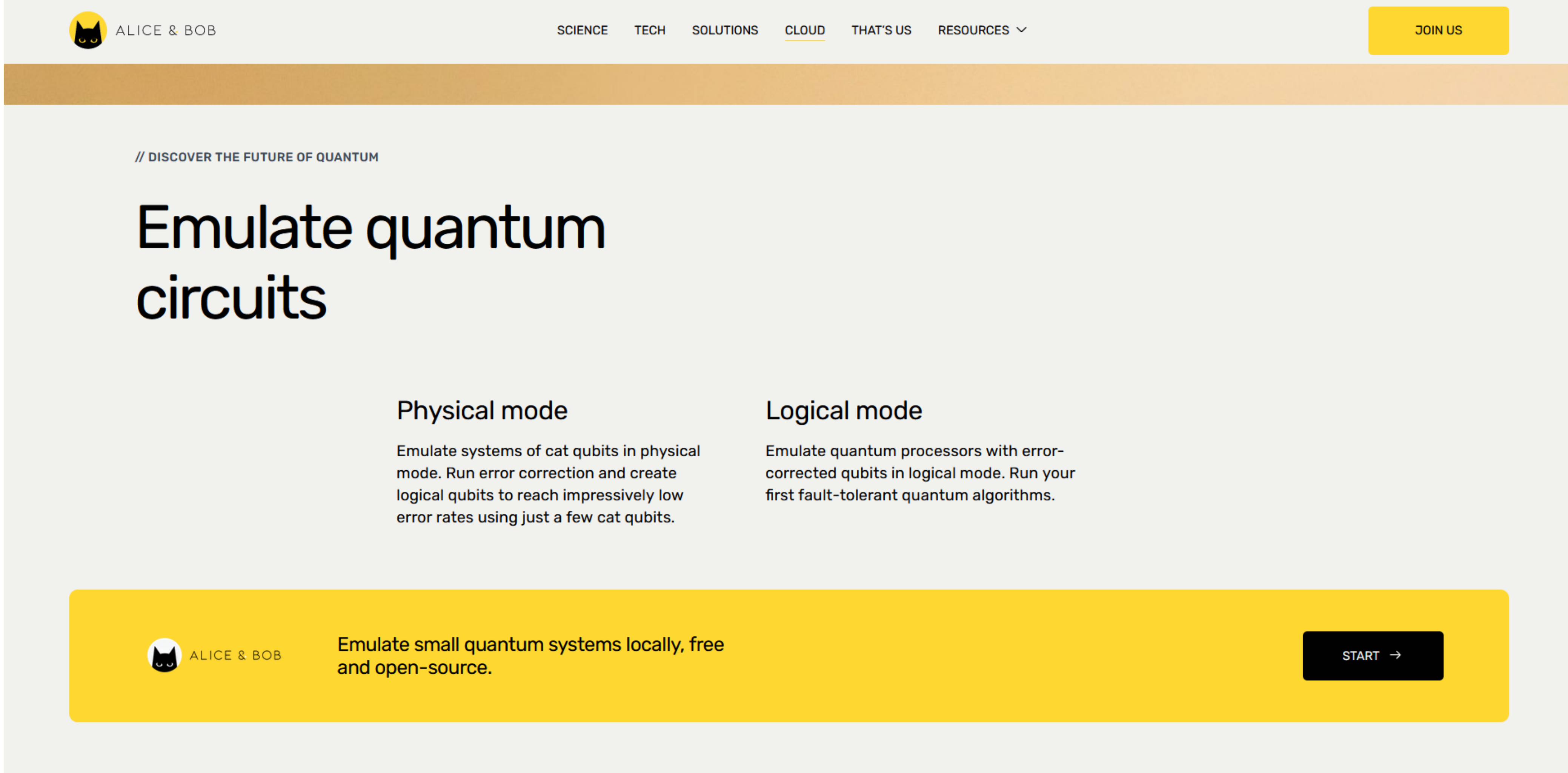
05

Use logical qubits

New paradigm, new metrics

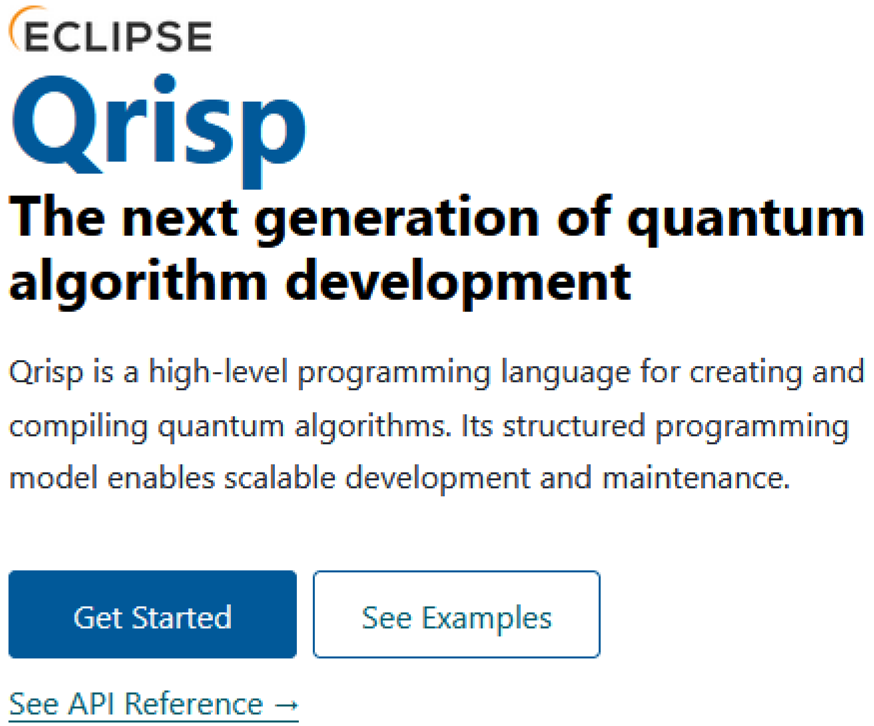
Metric	Physical qubits	Logical qubits
Error rate	Limited (probably > 10^{-4} or 10^{-5})	Virtually arbitrarily good
Logical:physical ratio	1:1	15:1 to 3000:1
# of usable qubits	Limited by noise	Limited by # of physical qubits
Connectivity	Limited	All-to-all
Gate set	Continuous (incl. native parametrized rotations)	Discrete (Clifford + T)
Gate time	Fast (< 1 μs)	Slower (~20 μs)

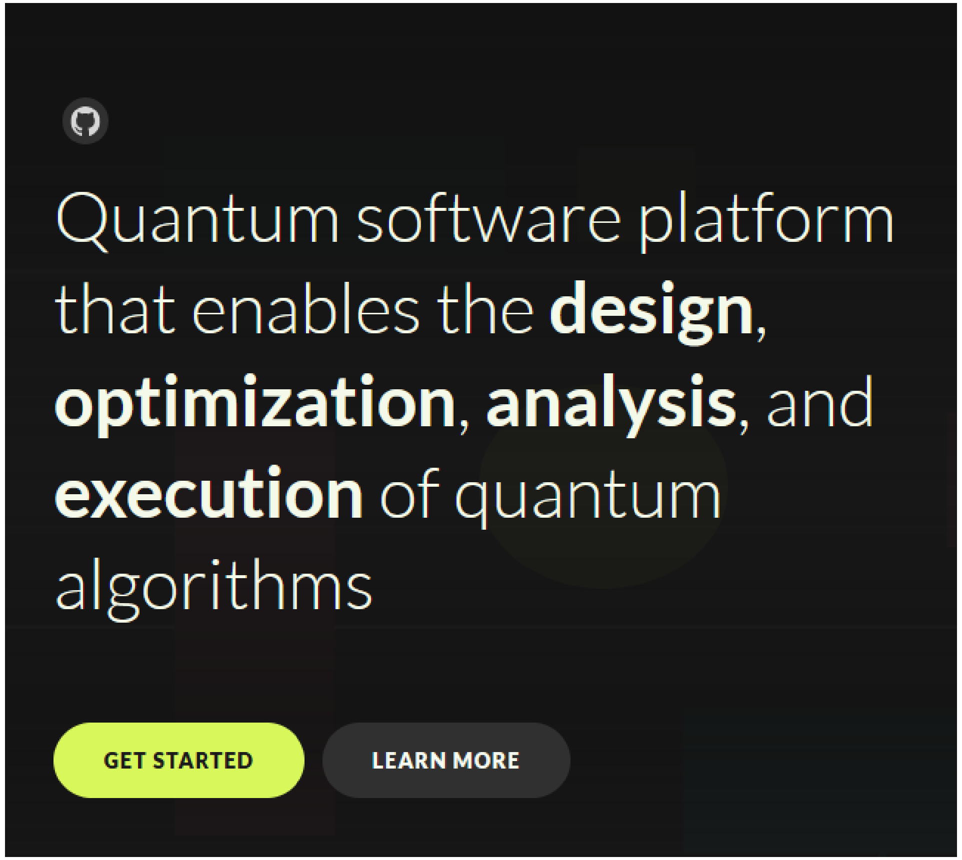
Tool #2 (again!): gate-level emulation



https://github.com/Alice-Bob-SW/qiskit-alice-bob-provider

Tool #5: quantum software development





https://qrisp.eu/

https://www.classiq.io/

Tool #6: resource estimation

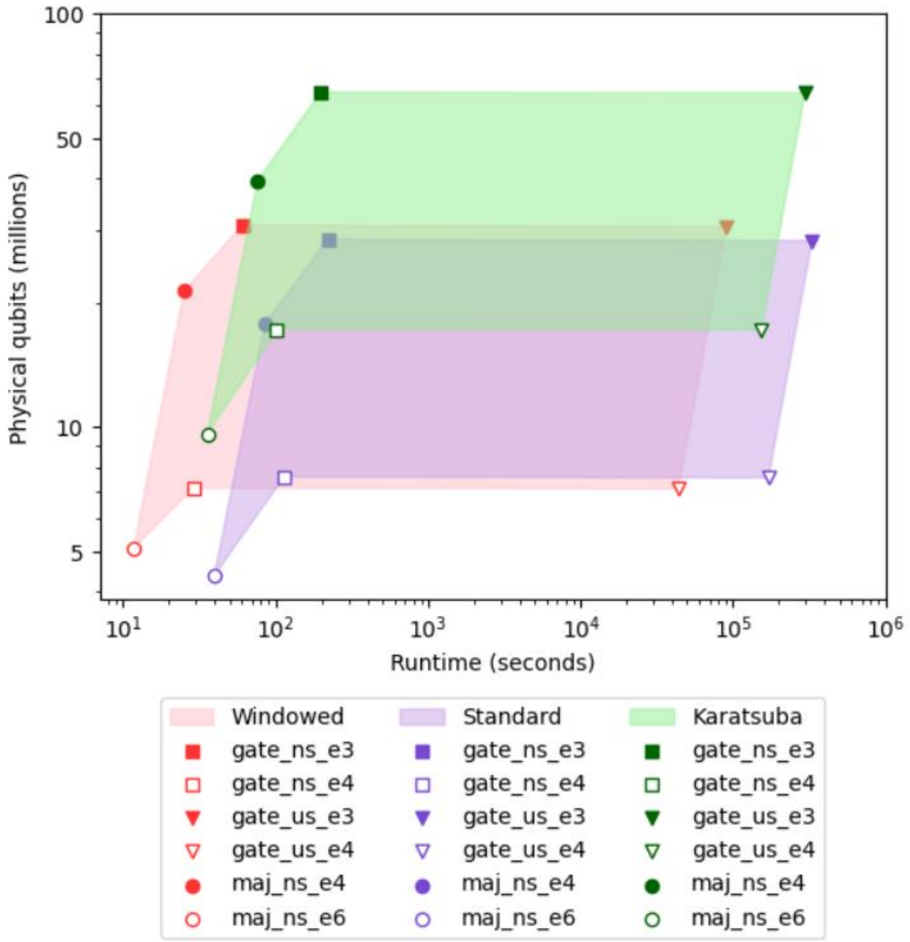
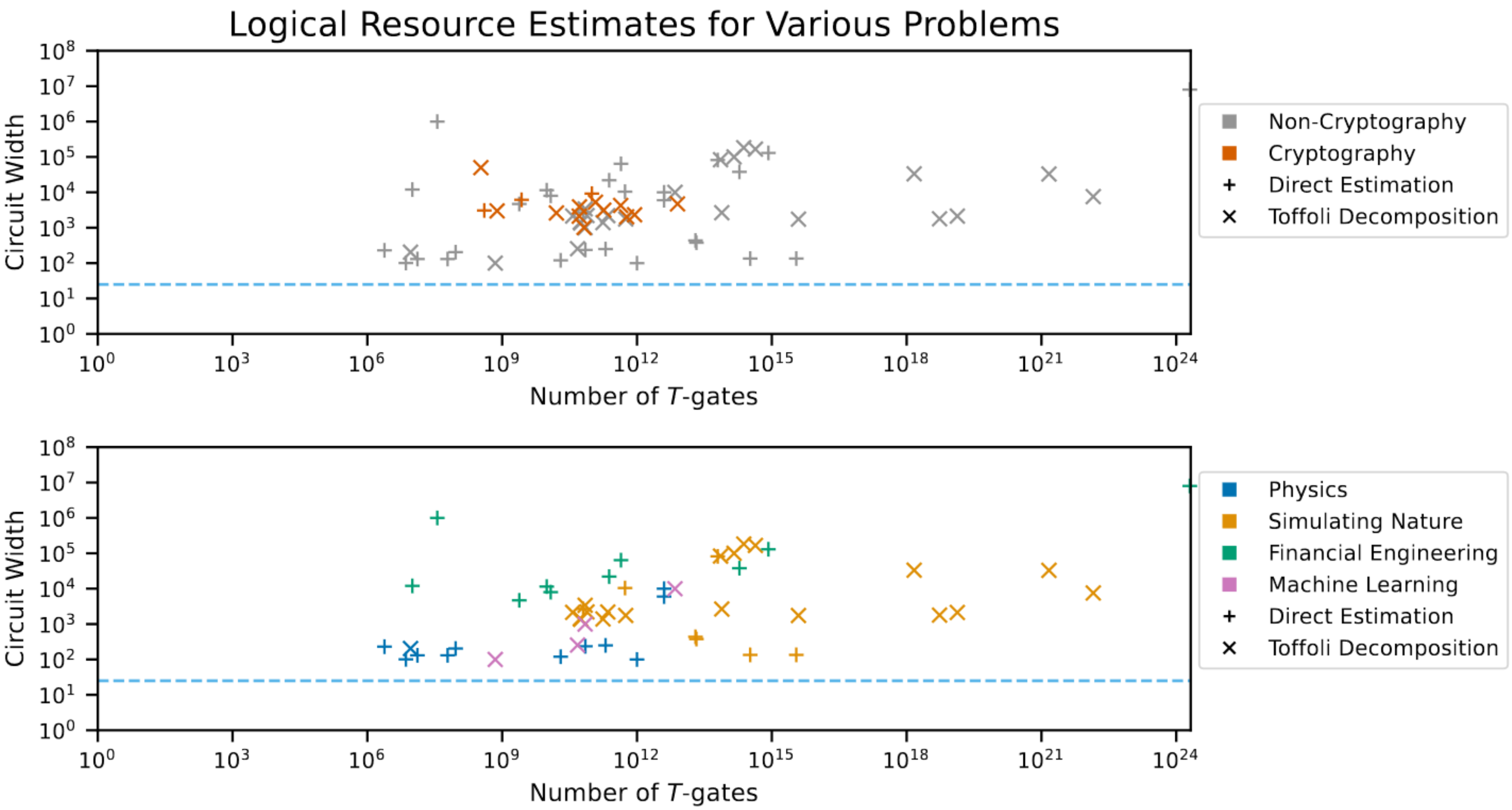


Fig. 4. The number of physical qubits and algorithm runtime across six different hardware profiles for the three multiplication algorithms, estimated for 2048-bit integers. The hardware profiles are described in Section III.C of [1]. The results are produced for total error budget 0.0001. The estimates for gate-based hardware profiles used the surface code QEC scheme with default parameters, and Majorana hardware profiles used floquet code QEC scheme with default parameters.

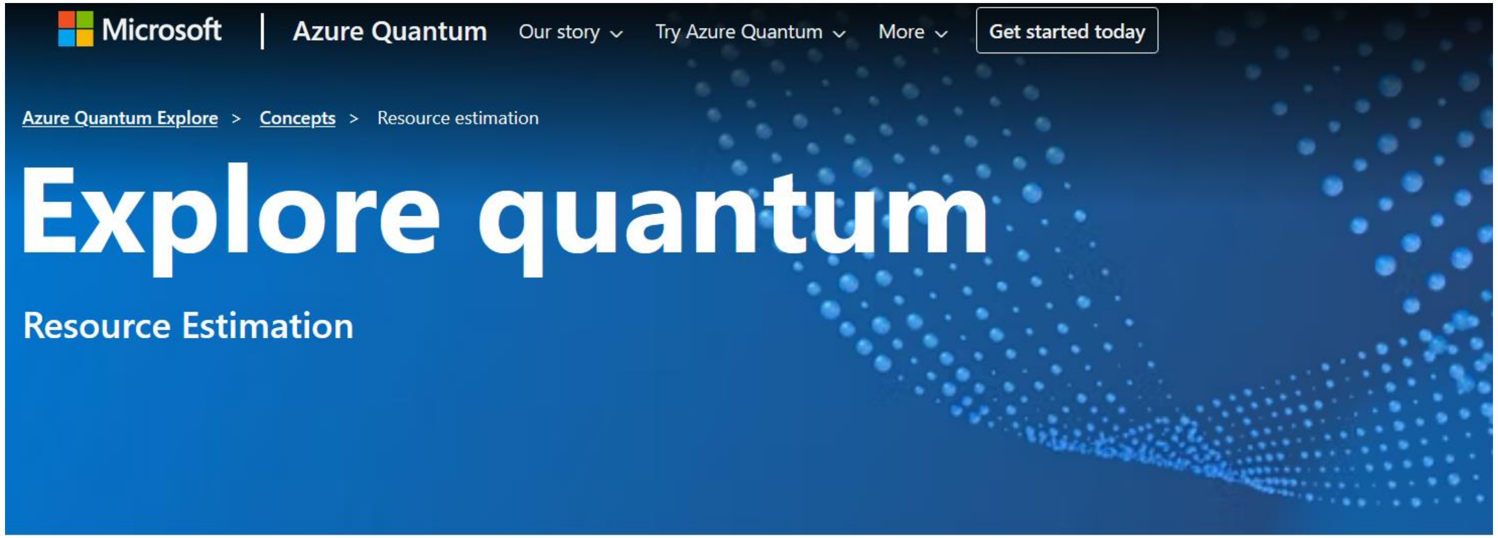
https://arxiv.org/pdf/2311.05801

Tool #6: resource estimation



https://arxiv.org/pdf/2401.16317

Tool #6: resource estimation



Although quantum computers promise to solve some of the most intractable problems our society faces, such as climate change and food security, commercially viable solutions will require large-scale, fault-tolerant quantum computers. Solving these problems will also require algorithms and quantum applications capable of executing solutions at a scale, and in a timeframe, that is practical. But how do you know how long a given quantum application will take to run, or how many physical qubits it will require? In quantum computing, *Resource Estimation* is the process used to answer these questions. Resource Estimation can help you determine the number of qubits, quantum gates, processing time, and other resources needed to run a quantum program assuming specific hardware characteristics.

https://quantum.microsoft.com/en-us/explore/concepts/resource-estimation

Tool #6: resource estimation

README

Q# resource estimator for Alice & Bob's architecture

This project contains the code for using Microsoft Q# resource estimator (presented in this paper) for Alice & Bob's architecture, using cat qubits and repetition code (LDPC codes might be added in the future).

Shor's algorithm for solving the elliptic curve discrete logarithm problem is used as an example, as in the paper Phys. Rev. Lett. 131, 040602 (arXiv: 2302.06639). Results from the resource estimator can be compared with the one of the code coming with the paper.

Big thanks to Mathias Soeken for having written the initial version of this repository, and rebuilt Microsoft Q# resource estimator to allow our architecture to be handled.

https://github.com/Alice-Bob-SW/qsharp-alice-bob-resource-estimator


Thank you!