Efficient Benchmarking with Provable Guarantees and more

Sami Abdul Sater

M. Garnier*, C. Gustiani, E. Kashefi, D. Leichtle, L. Music, T. Martinez*, H. Ollivier*, A. Saha*

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS | cnrs | Inria

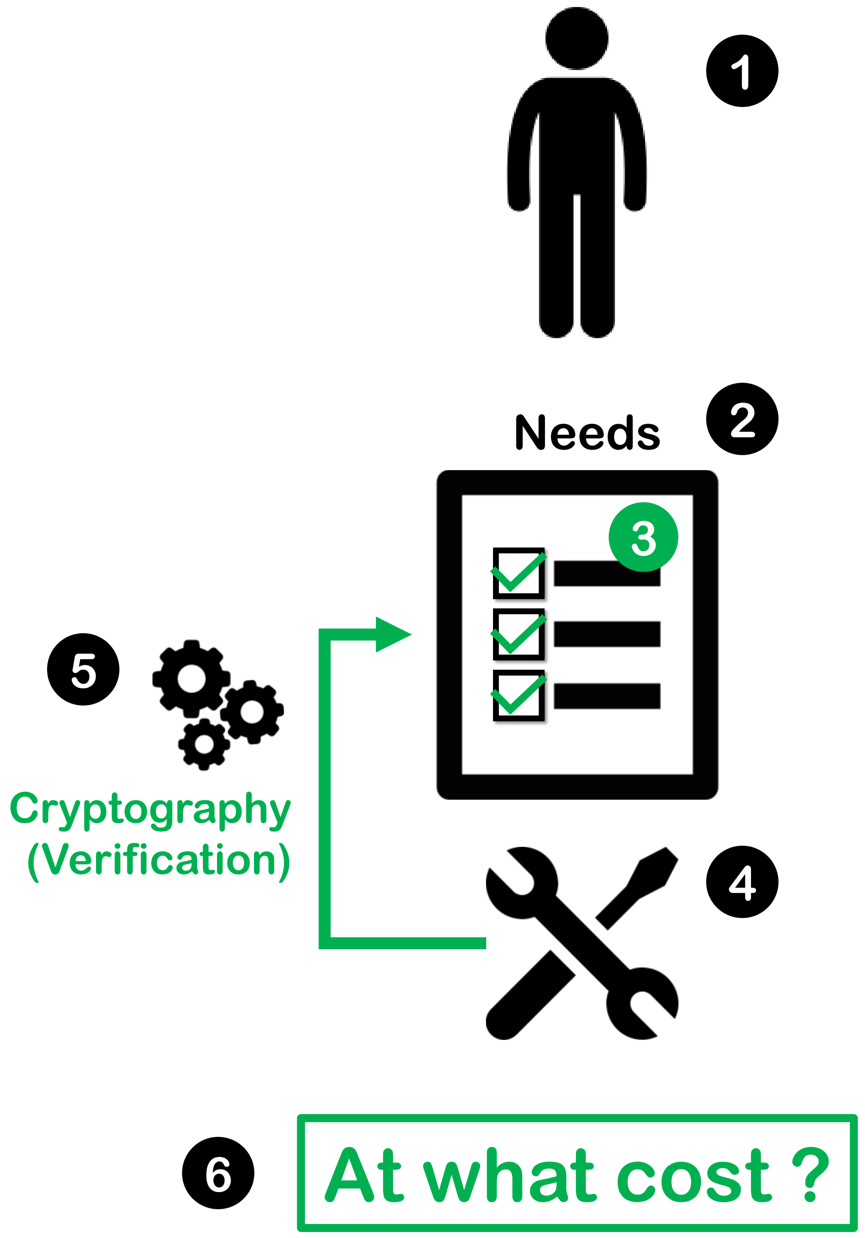
x Quandela, LIP6, University of Edinburgh

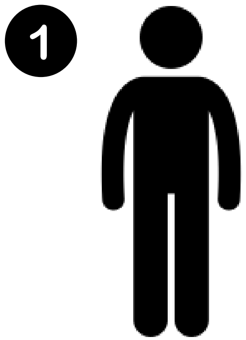
Outline

- Motivation
  1. Benchmark… for who ?
  2. Addressing which needs ?

- Contributions
  3. What guarantees are reached ?
  4. Benchmarking protocol
  5. Origins: Verification
  6. Consequence on assumptions

- Conclusion

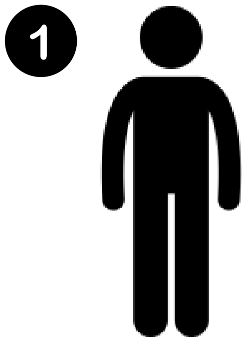




Motivation: benchmark… for whom ?

Benchmark: An approach that enables an entity to compare options based on a metric that is relevant to its usage.

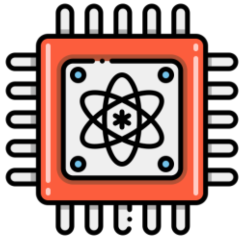
- Enlighten hardware constructors…

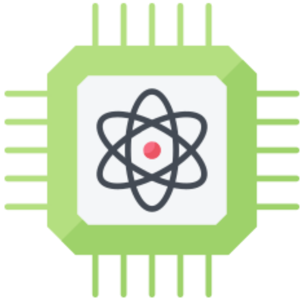


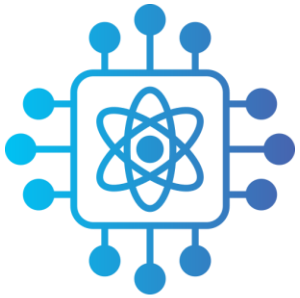
Motivation: benchmark… for whom ?

Benchmark: An approach that enables an entity to compare options based on a metric that is relevant to its usage.

- Enlighten hardware constructors…







THIS WORK:

- **Enlighten hardware buyers.**

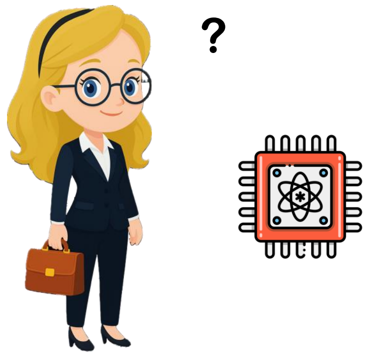
  Very generic usage!

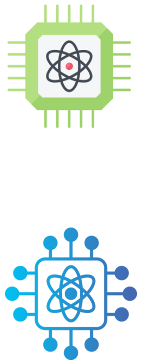
Addressing HW buyers needs

HW = hardware

Wish-list*

- ❑ Reliable <u>QC capability</u>
- ❑ Abstraction on the HW





Output of a benchmark : Meaningful* guarantee

δ —soundness
Worst—case guarantee

QC Capability

<u>Max</u> [Pr(incorrect | Benchmark OK)] < δ

Benchmark properties

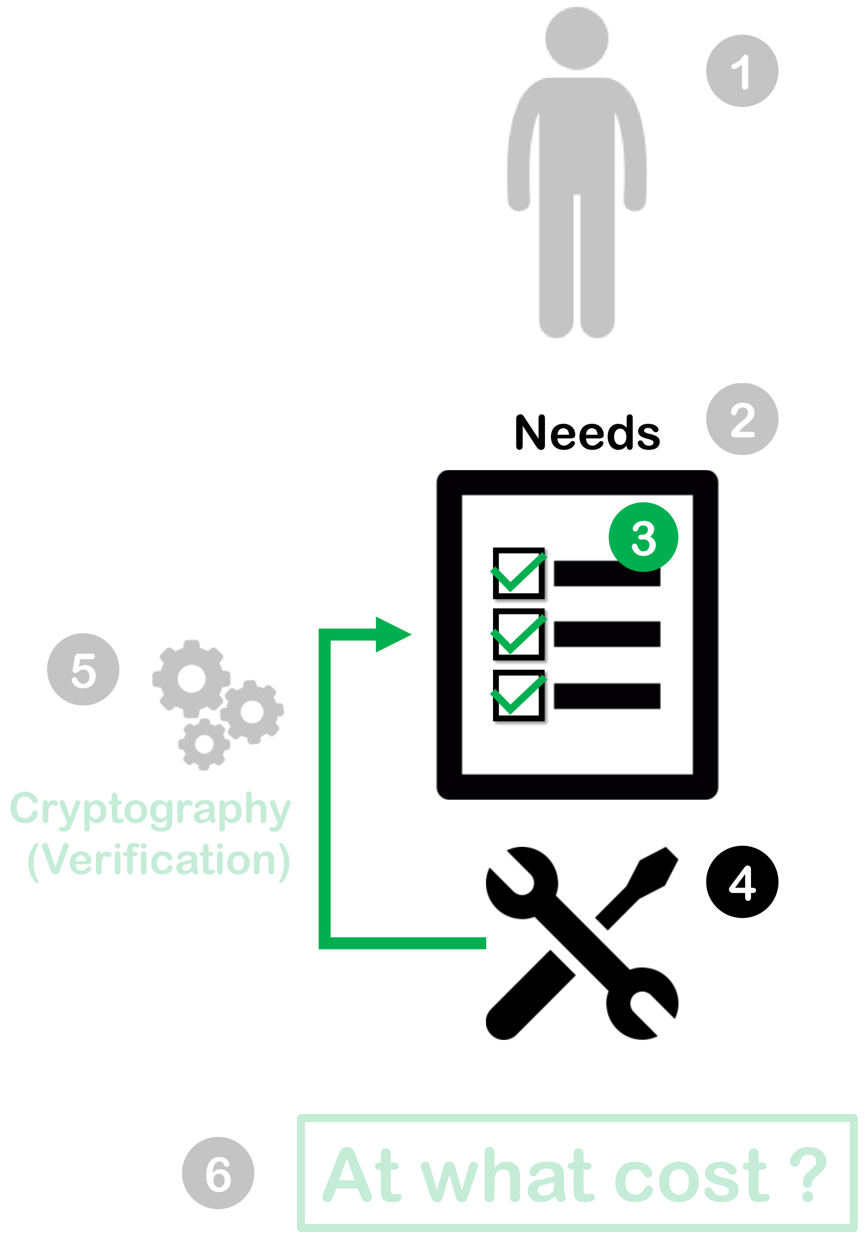
- ❑ Scalability:
  what happens when the target computation size gets bigger ?
- ❑ Efficiency:
  what happens when we want a better guarantee (smaller δ) ?
- ❑ Minimal assumptions
  (technology-independent, …)

Outline

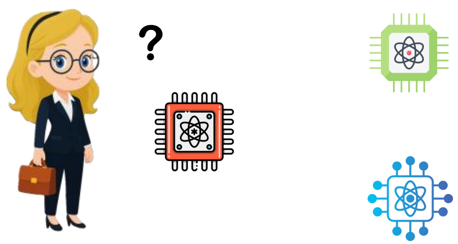
- Motivation
  1. Benchmark… for who ?
  2. Addressing which needs ?

- Contributions
  3. What guarantees are reached ?
  4. Benchmarking protocol
  5. Origins: Verification
  6. Consequence on assumptions

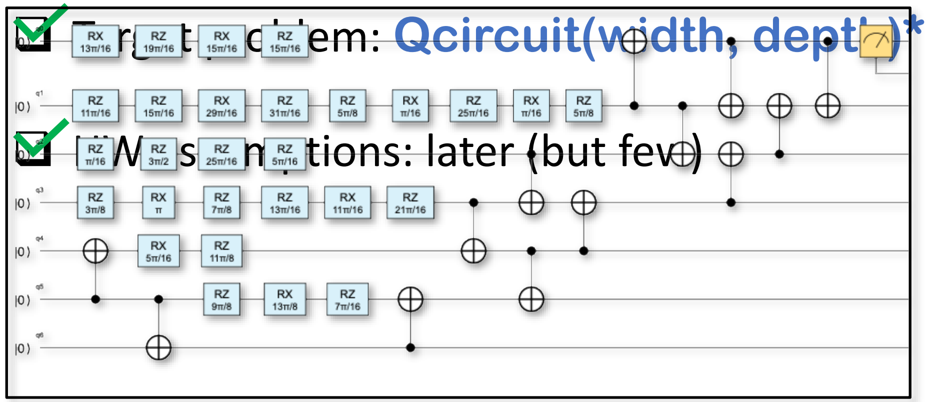
- Conclusion



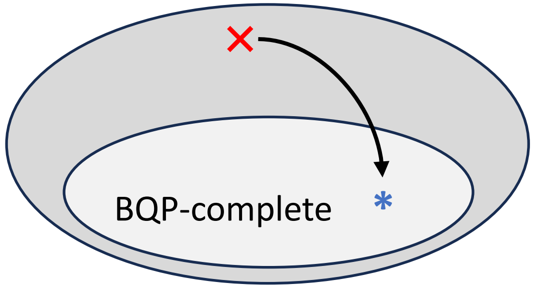
Technical statements



Evaluating a Quantum Computer (on useful problems)



- ☑ Target problem: Qcircuit(width, depth)*
- ☑ HW assumptions: later (but few)



Benchmark properties

- ☑ Scalability
- ☑ Efficiency

efficient

"test" rounds: $\sim \log(1/\delta)$

Size of "test" : size of the comp.

scalable

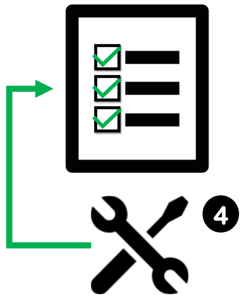
Output of the benchmark:

- **Soundness:** Pr(incorrect | Benchmark OK) < δ
- **Proved** by cryptography
- **Worst-case guarantee:** for all instances*, **in particular the worst one**

→ Meaningful and relevant guarantee for HW buyers

The (simple) benchmarking protocol.

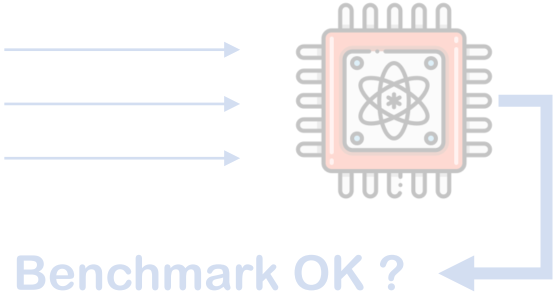
Needs



1. Define guarantee δ ⟶ ⟶ Number (#) of **test rounds**
   + size (depth, width) of comp.

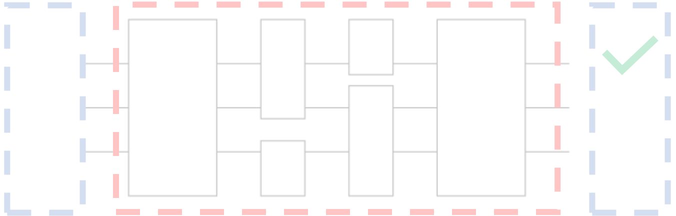
$$\# \sim \log(1/\delta)$$
$$\delta \sim \exp(-\#)$$

2. Test phase. Run **tests**, analyze outcomes, decide if **Benchmark OK** or not



Target comp.

Test.



Check!

(different inputs, same comp, deterministic outcomes)

3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | Benchmark OK) < δ

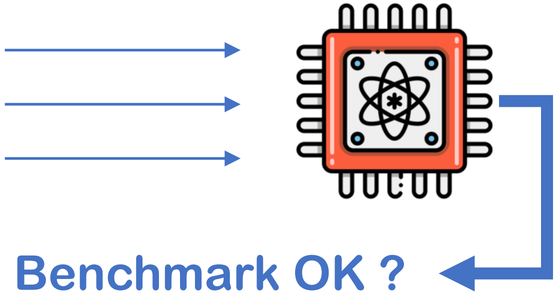
The (simple) benchmarking protocol.

1. Define guarantee δ ⟶ Number (#) of **test rounds**
   + size (depth, width) of comp.

$\# \sim \log(1/\delta)$

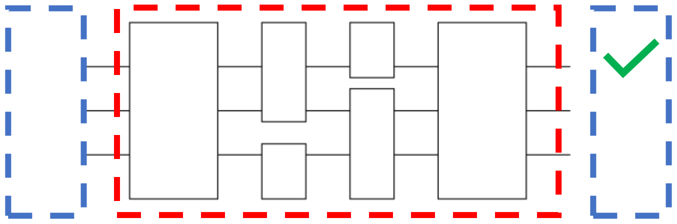
$\delta \sim \exp(-\#)$

2. Test phase. Run **tests**, analyze outcomes, decide if **Benchmark OK** or not



Target comp.

Test.



Check!

(different inputs, same comp, deterministic outcomes)

3. Guarantee. For any instance of QCircuit of the same width and depth,

$$\Pr(\text{Failed computation} \mid \text{Benchmark OK}) < \delta$$

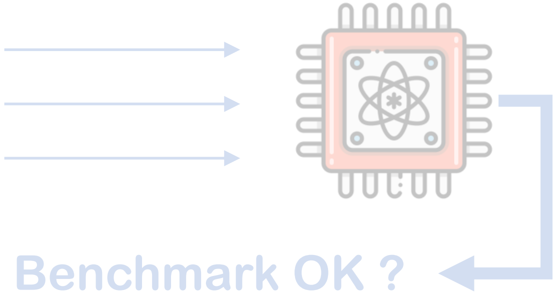
The (simple) benchmarking protocol.

1. Define guarantee δ + size (depth, width) of comp. → Number (#) of **test rounds**

$\# \sim \log(1/\delta)$

$\delta \sim \exp(-\#)$

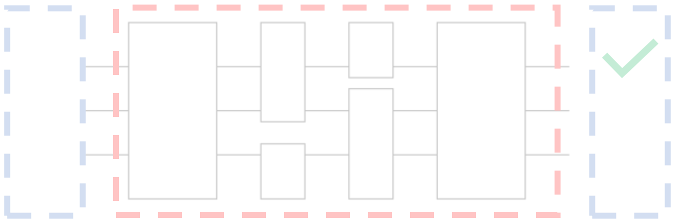
2. Test phase. Run **tests**, analyze outcomes, decide if **Benchmark OK** or not



Benchmark OK ?

Target comp.

Test.



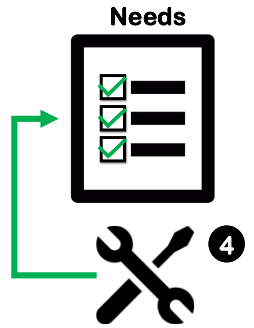
Check!

(different inputs, same comp, deterministic outcomes)

3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | Benchmark OK) < δ

The (simple) benchmarking protocol.

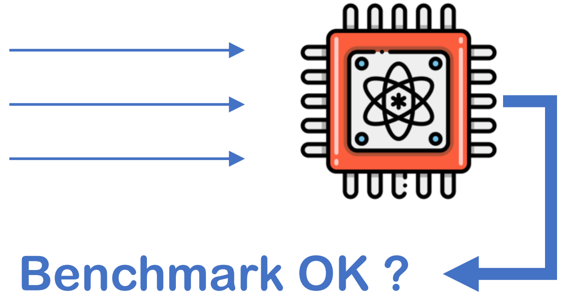


1. Define guarantee δ → Number (#) of **test rounds**
   + size (depth, width) of comp.

$\# \sim \log(1/\delta)$

$\delta \sim \exp(-\#)$

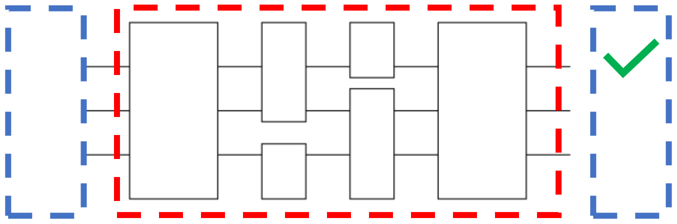
2. Test phase. Run **tests**, analyze outcomes, decide if **Benchmark OK** or not



Benchmark OK ?

Target comp.

Test.



Check!

(different inputs, same comp, deterministic outcomes)

3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | Benchmark OK) < δ

The (simple) benchmarking protocol.

BENCHMARK PROPERTIES

Meaningful guarantee

- Bound on the failure probability
- For any computation of the class
- Proved bound

Efficiency: more precision?
More tests, but logarithmic

Scalability: bigger computation?
Only impacts tests' size (!)

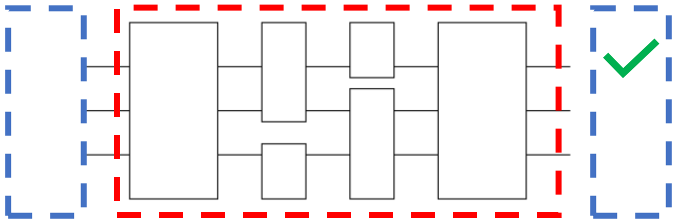
Number (**#**) of **test rounds**

$$\# \sim \log(1/\delta)$$

$$\delta \sim \exp(-\#)$$

Target comp.

Test.



Check!

(different inputs, same comp, deterministic outcomes)

For any instance of QCircuit of the same width and depth,

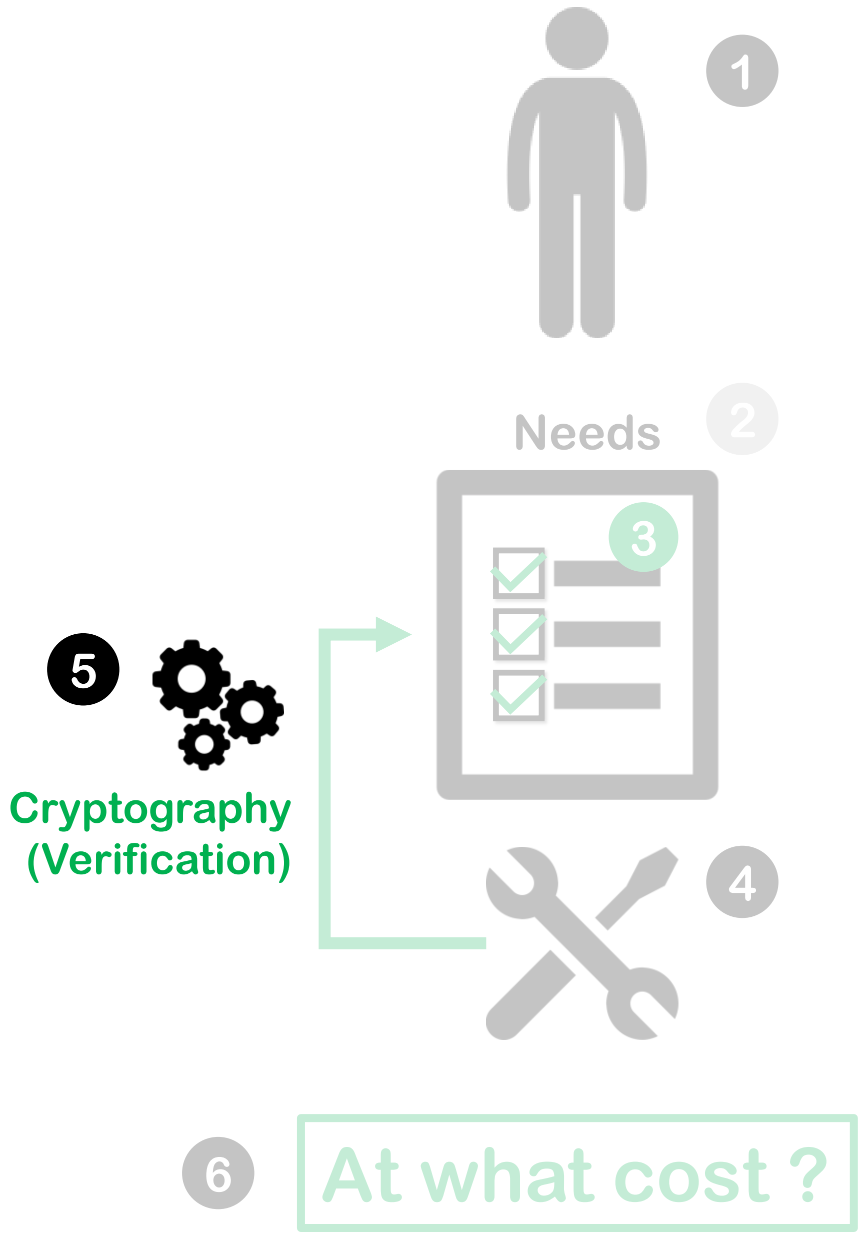
Pr(Failed computation | Benchmark OK) < δ

Outline

- Motivation
  1. Benchmark… for who ?
  2. Addressing which needs ?

- Contributions
  3. What guarantees are reached ?
  4. Benchmarking protocol
  5. Origins: Verification
  6. Consequence on assumptions

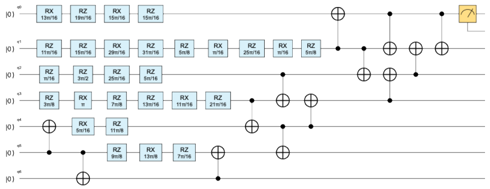
- Conclusion



The (original) verification protocol.

5 Cryptography (Verification)

1. Define



 # test rounds, computation rounds, threshold

2. Interleave test and computation rounds.

Protocol

If n. failed tests < threshold,
Accept computation

TESTS

- Efficiently checkable instances: Clifford versions ?
- 2 types of tests are enough (Measurement-Based QC)
- Same comp.
- Different states.

3. Guarantee.

For any instance of QCircuit of the same width and depth,

Pr(Failed computation | n. failed tests < threshold) < δ

The (original) verification protocol.

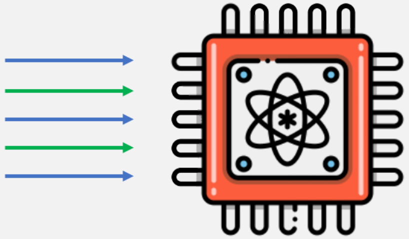
5 Cryptography (Verification)

1. Define

test rounds, computation rounds, threshold

2. Interleave test and computation rounds.

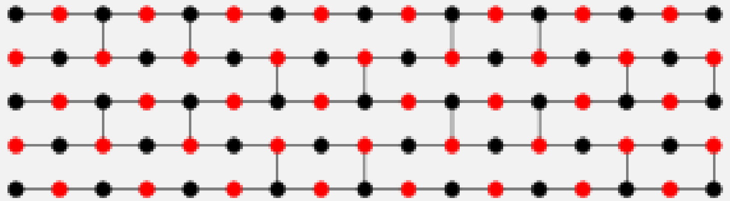
Protocol



If n. failed tests < threshold, Accept computation

TESTS

- Efficiently checkable instances: Clifford versions ?
- 2 types of tests are enough (Measurement-Based QC)
- Same comp.
- Different states.



3. Guarantee. For any instance of QCircuit of the same width and depth,

$$\Pr(\text{Failed computation} \mid \text{n. failed tests} < \text{threshold}) < \delta$$

The (original) verification protocol.

5 Cryptography (Verification)

1. Define

test rounds, computation rounds, threshold

2. Interleave test and computation rounds.

Protocol

If n. failed tests < threshold,
Accept computation

TESTS

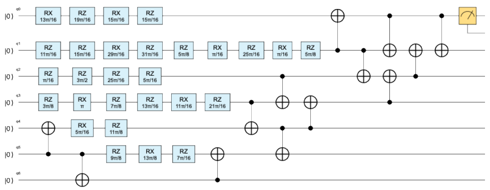
- Efficiently checkable instances: Clifford versions ?
- 2 types of tests are enough (Measurement-Based QC)
- Same comp.
- Different states.

3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | n. failed tests < threshold) < δ

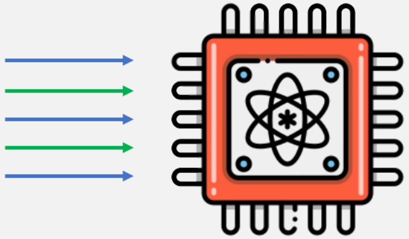
The (original) verification protocol.

Cryptography (Verification)

1. Define  # test rounds, computation rounds, threshold

2. Interleave test and computation rounds.

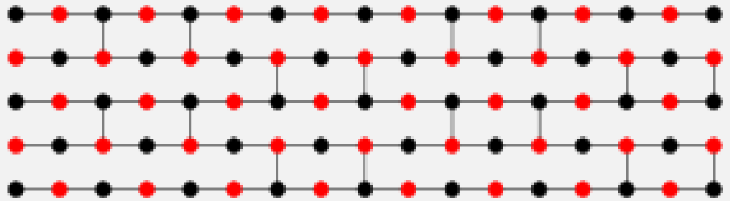
Protocol



If n. failed tests < threshold,
Accept computation

TESTS

- Efficiently checkable instances: Clifford versions ?
- 2 types of tests are enough (Measurement-Based QC)
- Same comp.
- Different states.

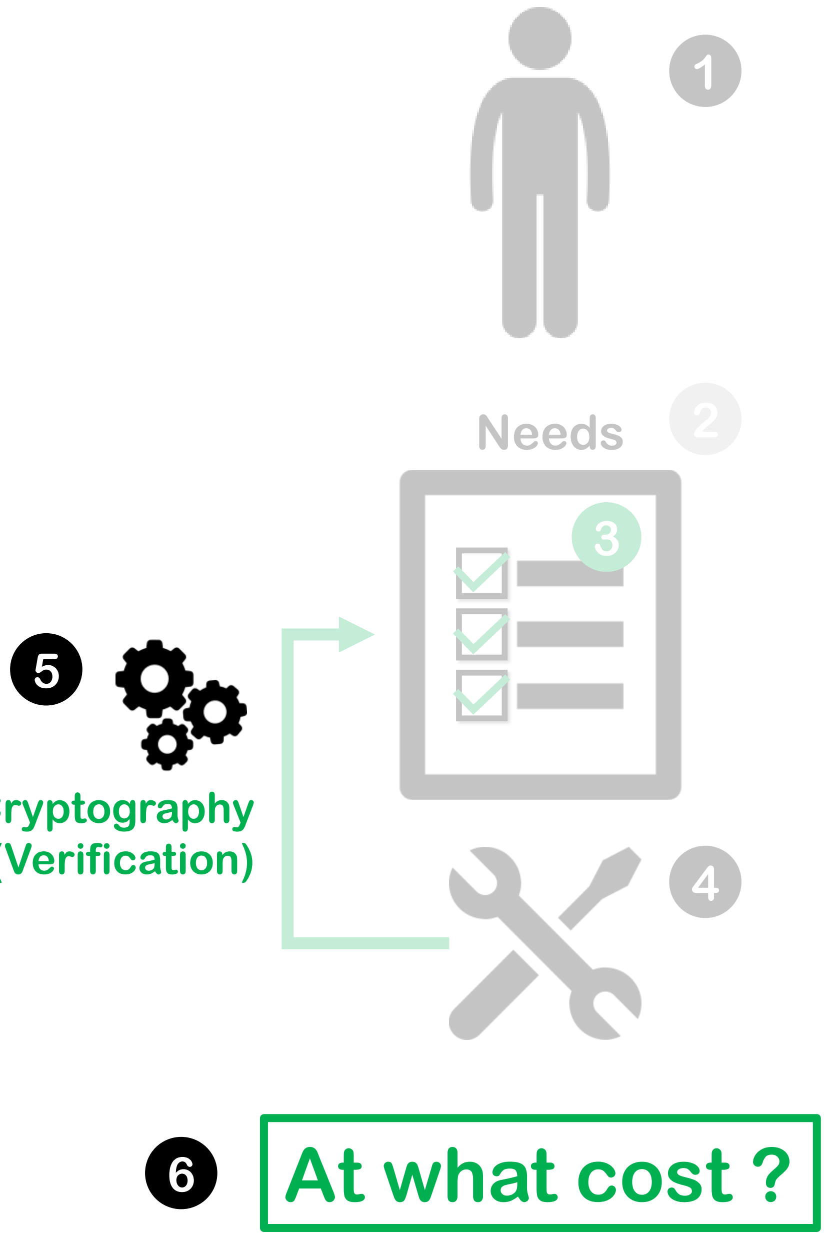


3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | n. failed tests < threshold) < δ $\delta \in \mathrm{negl}(\#, \#)$

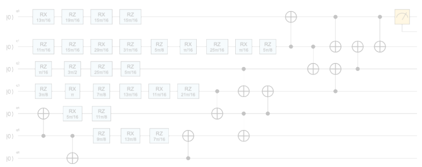
Outline

- Motivation
  1. Benchmark… for who ?
  2. Addressing which needs ?
- Contributions
  3. What guarantees are reached ?
  4. Benchmarking protocol
  5. Origins: Verification
  6. Consequence on assumptions
- Conclusion



Benchmark via Verification: at what cost ?

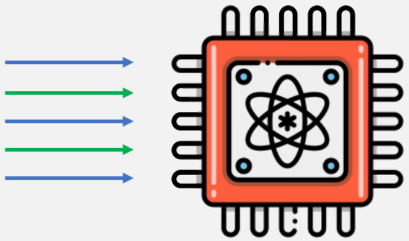
1. Define



test rounds, computation rounds, threshold

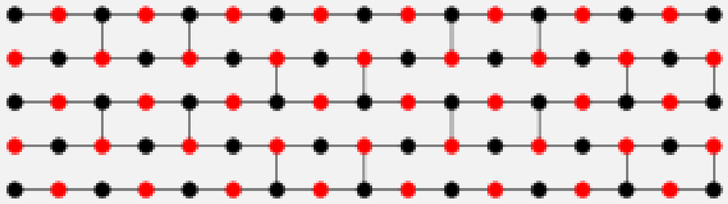
2. Interleave test and computation rounds.

Verification Protocol



If n. failed tests < threshold,
Accept computation

Assumptions



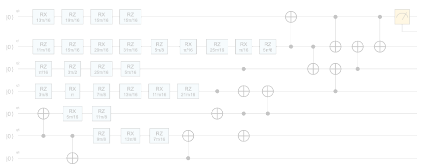
States preparation
Noise must be state-independent!

3. Guarantee. **For any instance of QCircuit of the same width and depth,**

Pr(Failed computation | n. failed tests < threshold) < δ $\delta \in \mathrm{negl}(\#, \#)$

Benchmark via Verification: at what cost ?

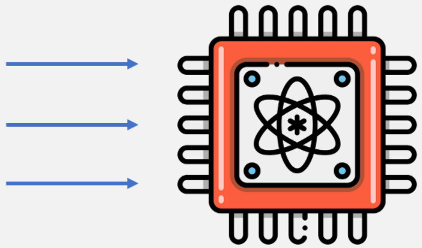
1. Define



test rounds, computation rounds, threshold

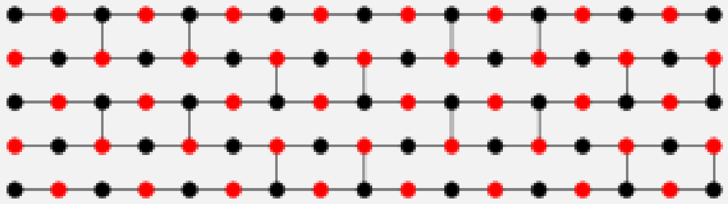
2. Interleave test and computation rounds.

~~Verification~~ Benchmarking Protocol



If n. failed tests < threshold,
The computation is likely to be good

Assumptions



<u>States preparation</u>
Noise must be state-independent!

<u>Repeatability</u>
HW behaves the same through rounds

3. Guarantee.

For any instance of QCircuit of the same width and depth, when we do the computation,

Pr(Failed computation | n. failed tests < threshold) < δ

(n. failed tests < threshold: **(Benchmark OK)**)

$\boldsymbol{\delta} \in \mathrm{negl}(\#, \#)$

Conclusion: what does it tell us ?

1. A protocol evaluating a QC's performance through a metric characterizing its reliability on a set of useful computations, relevant for HW Buyers

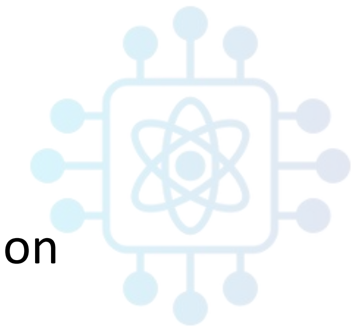
= Benchmark

2. Efficient: repetition is the only overhead

3. Scalable: the computation size only affects the tests size

4. The guarantee is valid under assumptions.
   - **Repeatability**
   - **State-independent noise**
   - Target for HW constructors !

5. And more!
   - Simpler, circuit-model protocol
   - With assumption on the noise model, possibility to do noise characterization



Thank you for your attention!

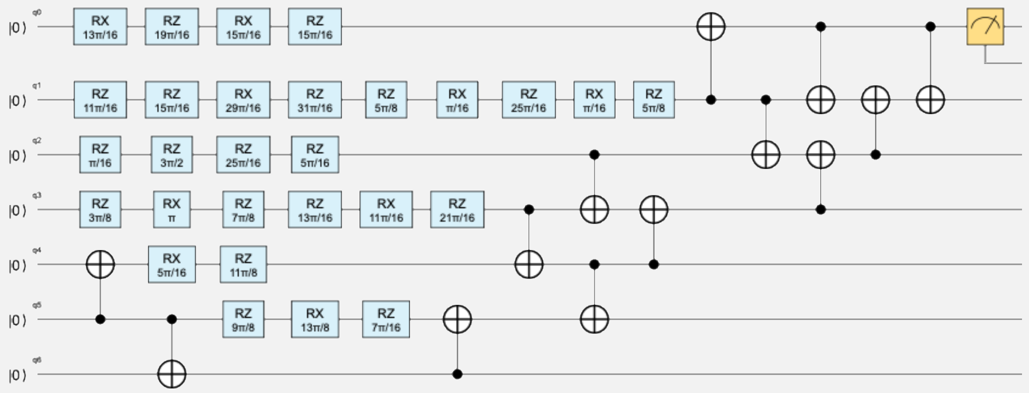
sami.abdul-sater@inria.fr

Backslides

Description of tests

TESTS

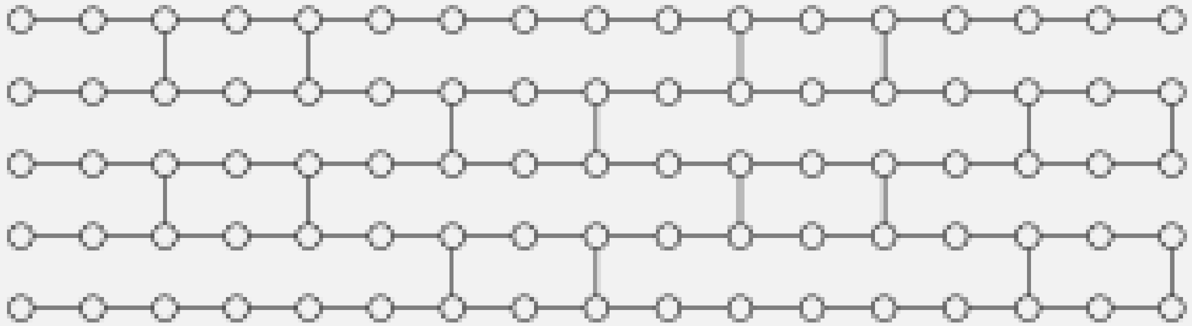
Original computation (QCircuit instance)



Description of tests

TESTS (Measurement-Based QC)

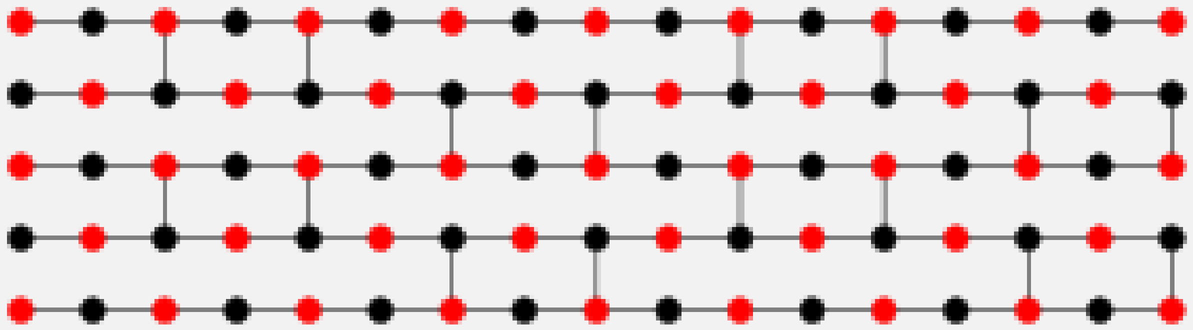
Original computation → MBQC Pattern on brickwork state



Description of tests

TESTS (Measurement-Based QC) : two types

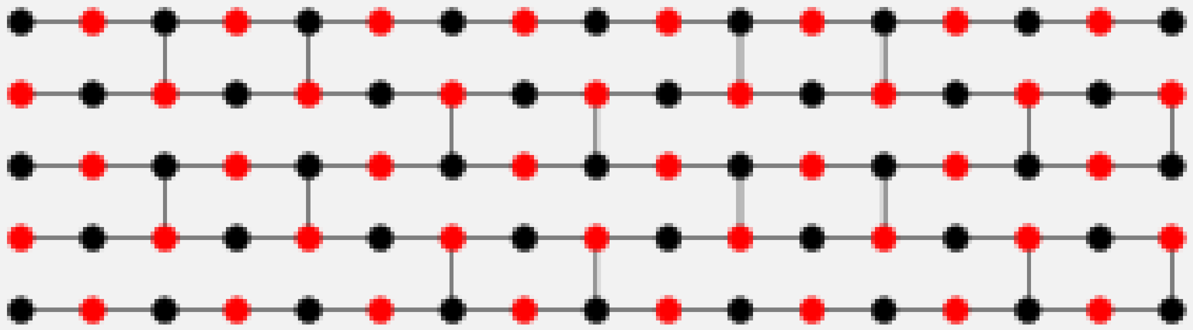
Prepare qubits: $|0\rangle$ **or** $|+\rangle$



Description of tests

TESTS (Measurement-Based QC) : two types

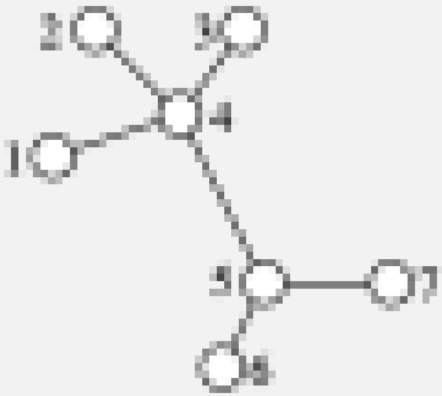
Prepare qubits: $|0\rangle$ **or** $|+\rangle$

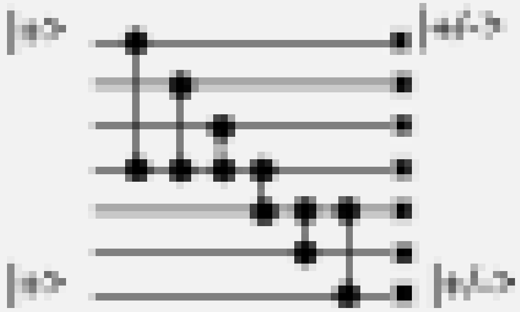


Description of tests

TESTS (Measurement-Based QC) : two types

Delegate **blindly.**





Description of tests

TESTS (Measurement-Based QC) : two types

Delegate **blindly, using Universal Blind QC**.