QRAND

A multiprotocol and multiplatform quantum random number generation framework

Pedro Rivero – November 23, 2021

Unitary Fund

Table of contents

The need for *random numbers**

Applications

- Information/communications security (e.g. crypto-systems, IoT)
- Computation (e.g. algorithms, physical simulations, Monte Carlo)
- Financial systems (e.g. transactions, credit card chips)
- Statistical sampling (e.g. scientific experiments, social studies)
- Testing (e.g. randomized benchmarking)
- Legal and trust sensitive processes (e.g. jury, lotteries, *Kleroterion*)

**Randomly generated numbers*

What we mean by random

A slippery concept

- The attribute of being random applies more correctly to a sequence of numbers: It is strictly related to **lack/impossibility of predictability**.
- The impossibility of **compression** can be considered as the defining trait of randomness (*Chaitin & Kolmogorov*).
- **Shannon Entropy** (H) → Measure of disinformation/uncertainty

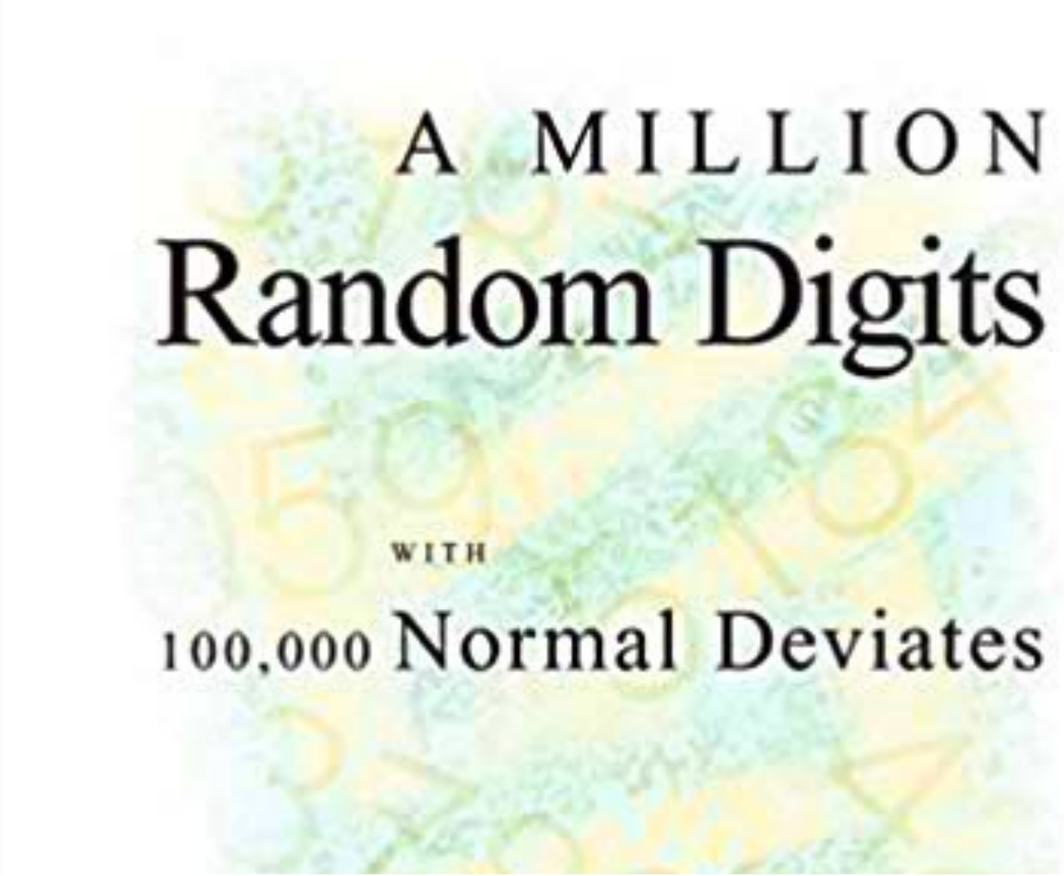
> *The minimum number of yes/no questions that need to be answered about something to be able to guess it correctly with 100% confidence.*

- Bit strings which are not random have limited entropy and can therefore be described with **fewer bits than the string itself**.

Randomness quality

Output tests

- Numbers are not inherently random *per se:* R(NG) ≠ (RN)G
- It is nearly impossible to establish whether a source is truly random.
- We can analyze the general properties of output strings (e.g. weight)
- A passing output will always pass (the tests): an adversary might *predict* or *fabricate* the output.



TDB

★☆☆☆☆ **Not really random**

Reviewed in the United States on September 26, 2012

I bought two copies of this book. I find that the first copy perfectly predicts what the numbers will be in the second copy. I feel cheated.

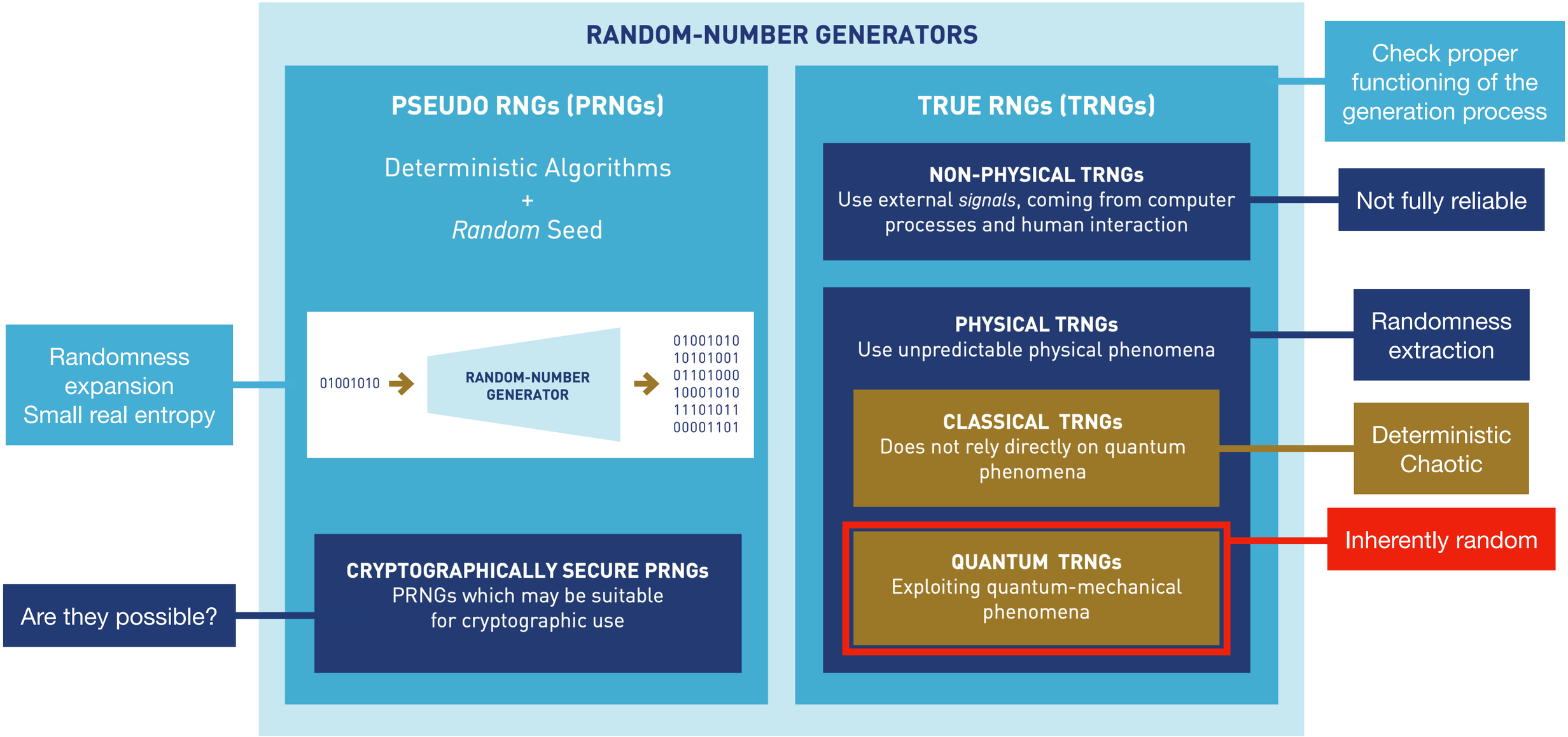
141 people found this helpful

Random number generation

Properties

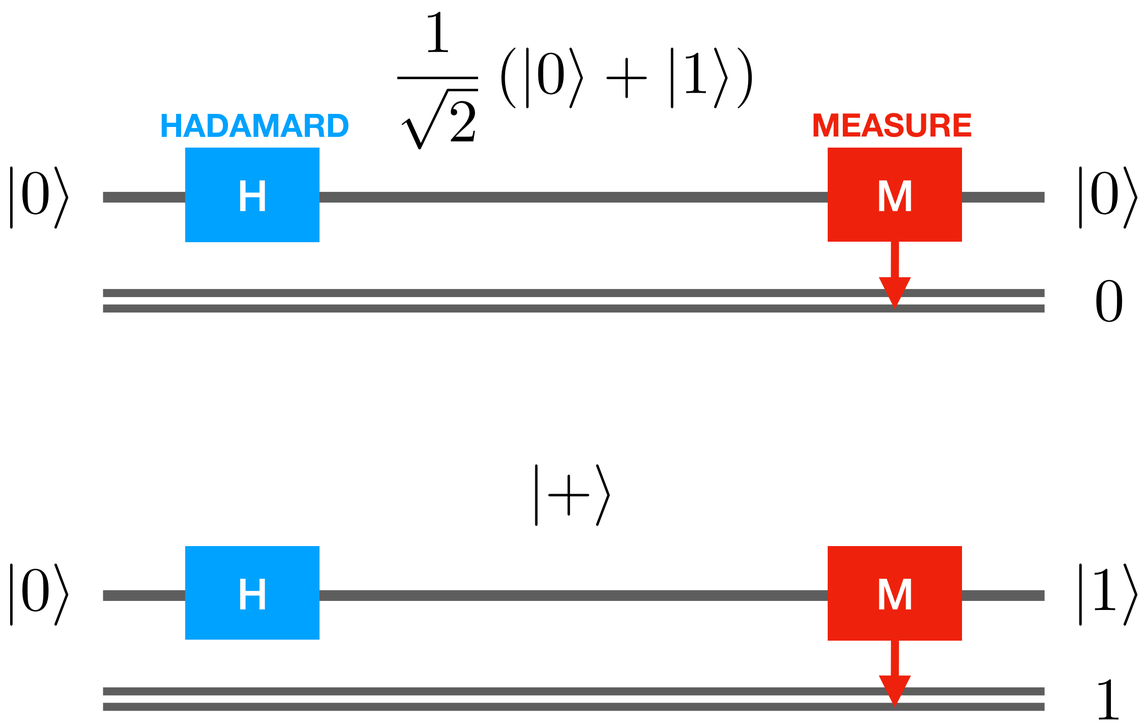
- **Uniformity** → all sequences have the same probability of occurring
- **Scalability** → any test applicable to a sequence must must apply to subseq.
- **Consistency** →the behavior must be consistent across all outputs
- **Forward unpredictability** → the next output should not be predictable
- **Backward unpredictability** → previous outputs should not be predictable

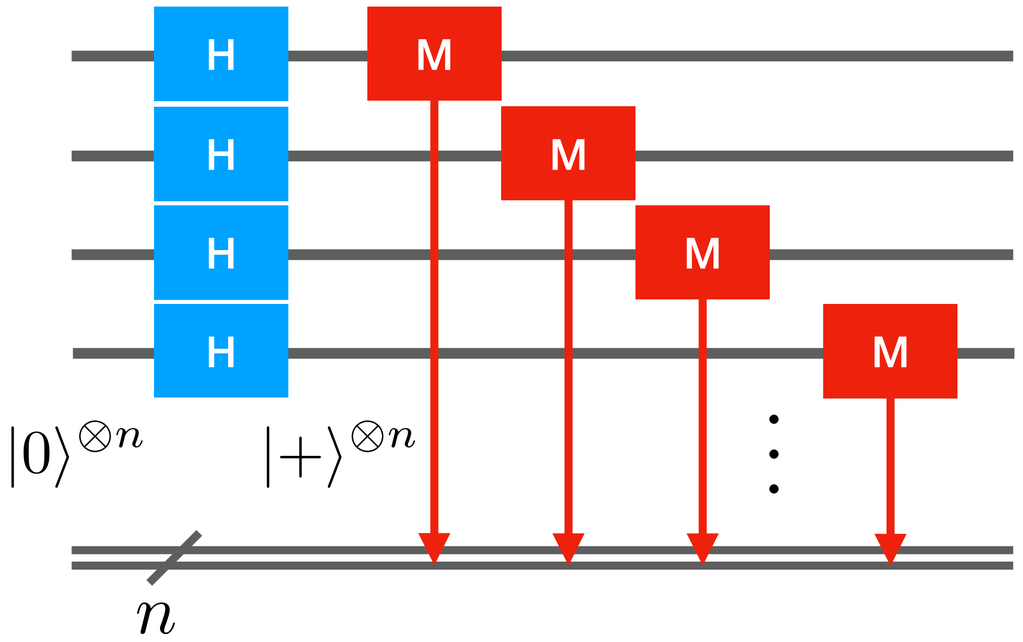
Random number generation



Hadamard Protocol

The naive approach using quantum circuits





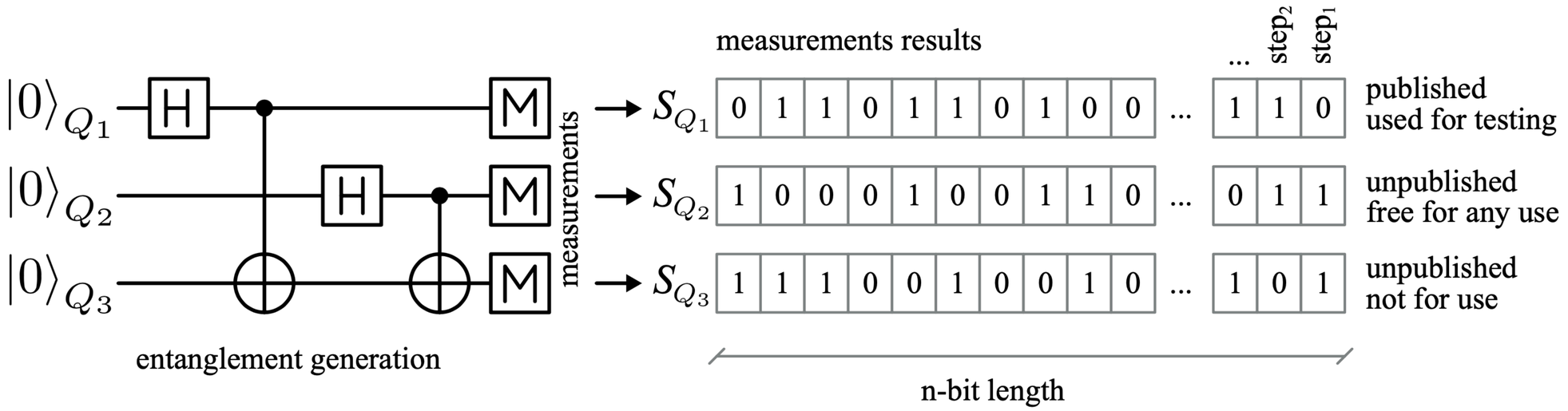
Entanglement Protocol

Validation outsourcing

- Subsequence validation → Exponential overhead

$$|\psi_{Q_1Q_2Q_3}\rangle = \frac{1}{2}\left(|000\rangle + |011\rangle + |101\rangle + |110\rangle\right)$$

TWO BITS OF ENTROPY

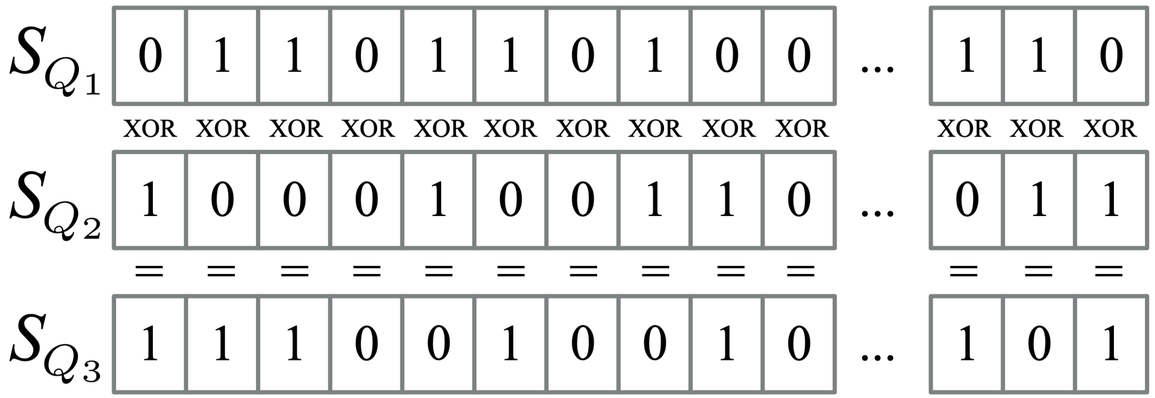


PARITY

Entanglement Protocol

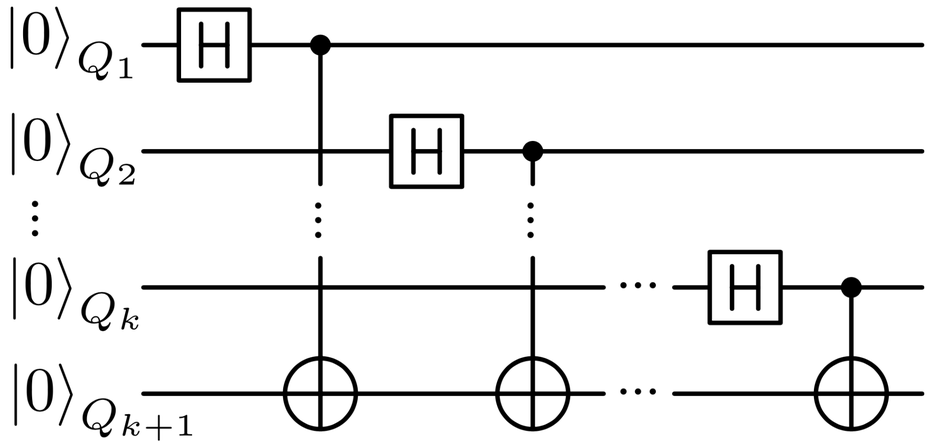
Error correction and scaling

- Parity bit



PURIFY ENTROPY

1 SEQUENCE TO VALIDATE

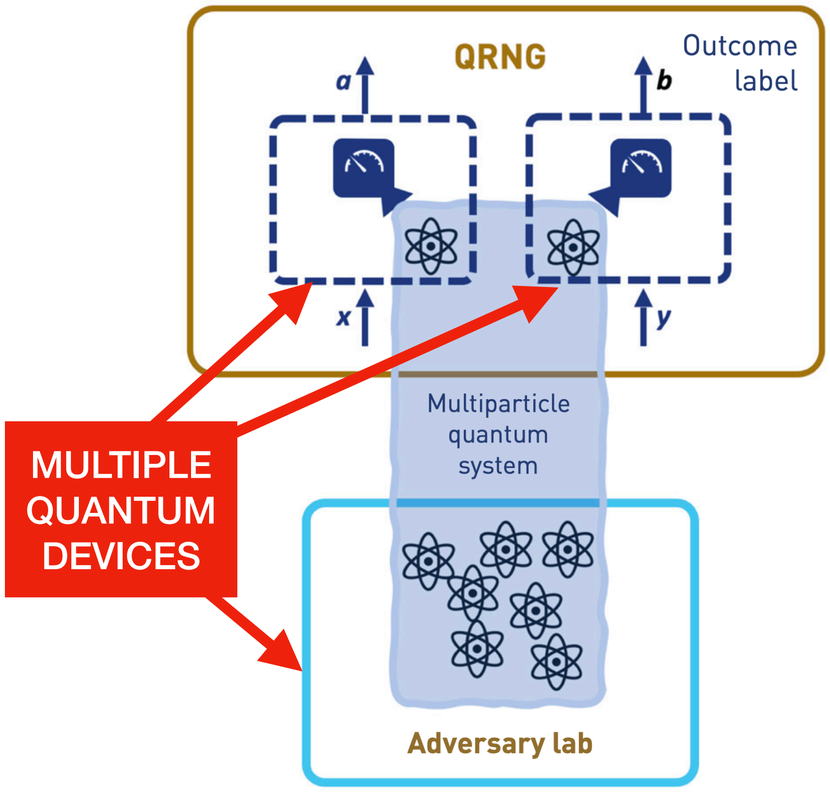


K-1 USABLE SQUENCES

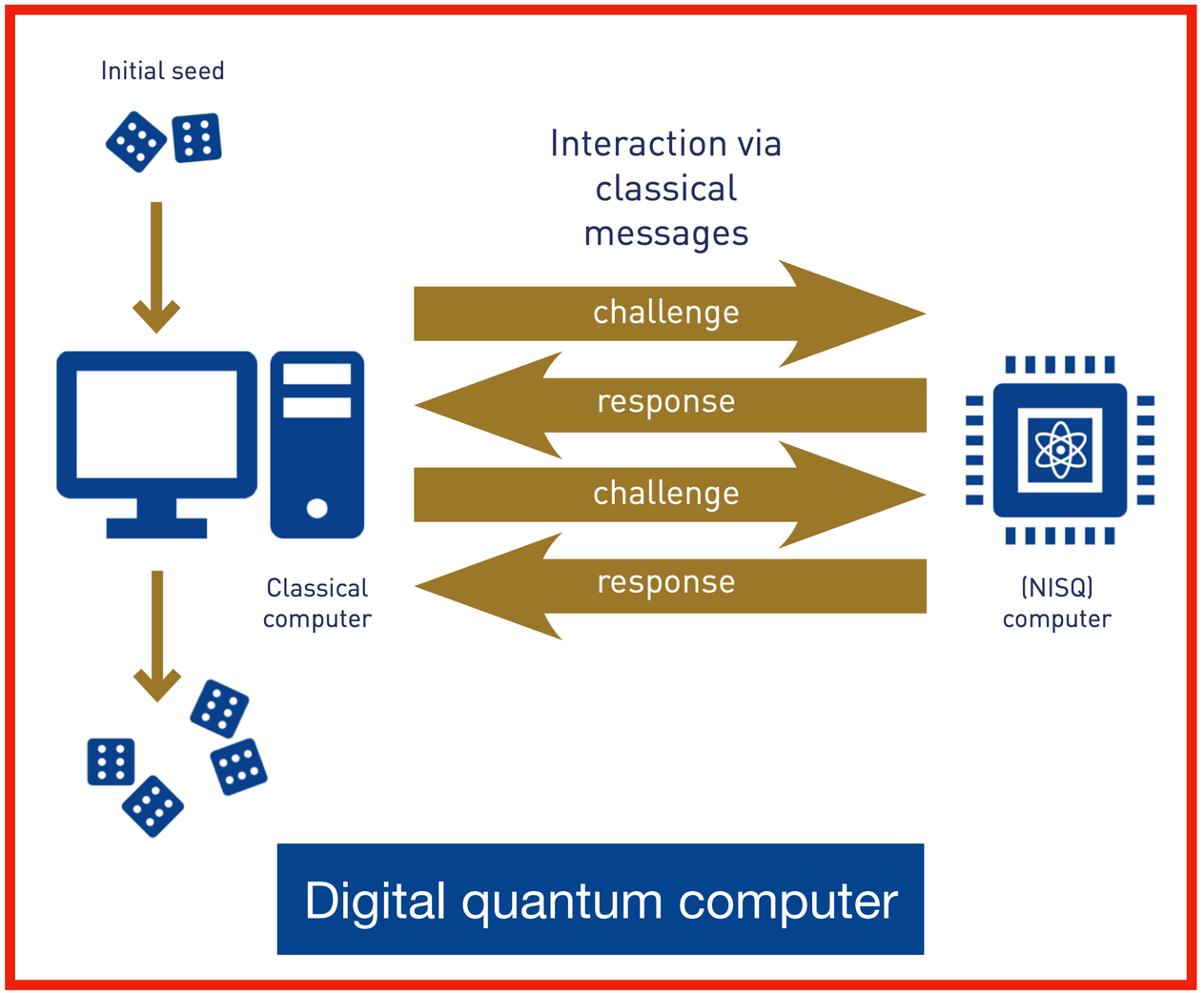
PARITY

Verification strategies

Device independency



Locality proofs

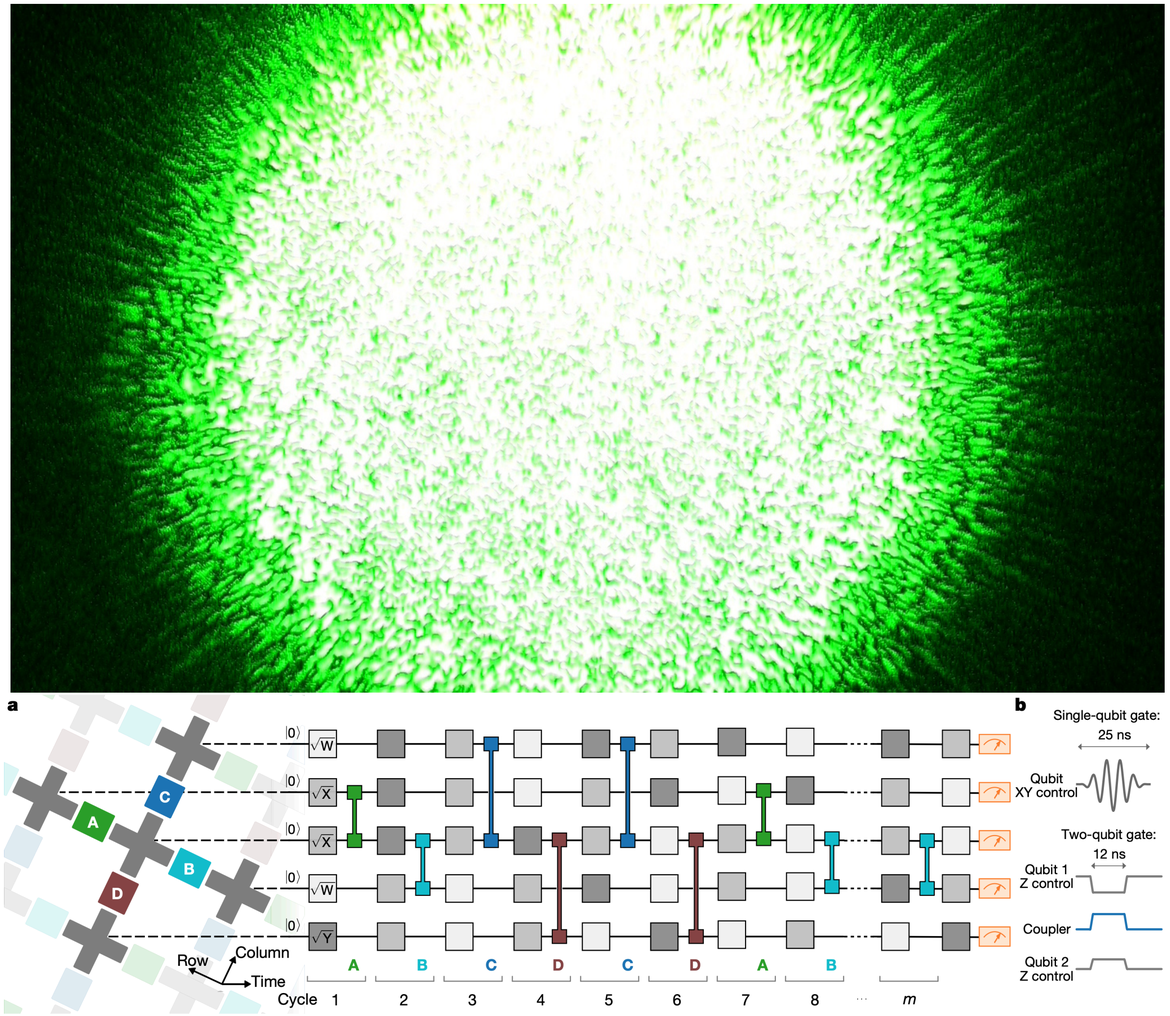


Digital quantum computer

Sycamore Protocol

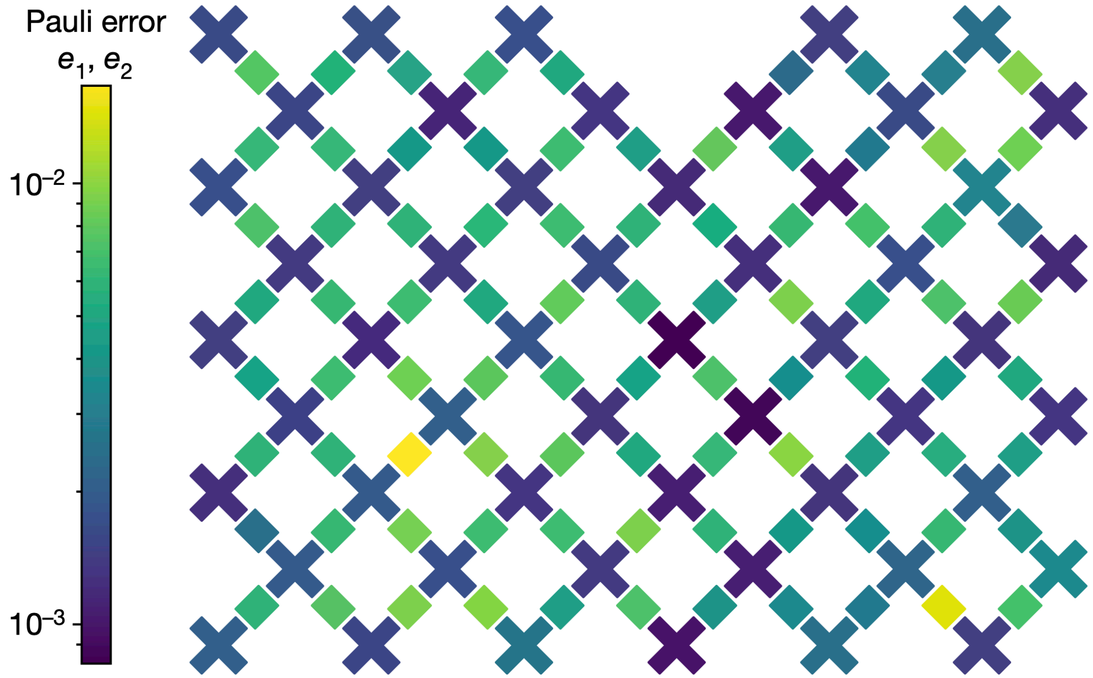
Scott Aaronson

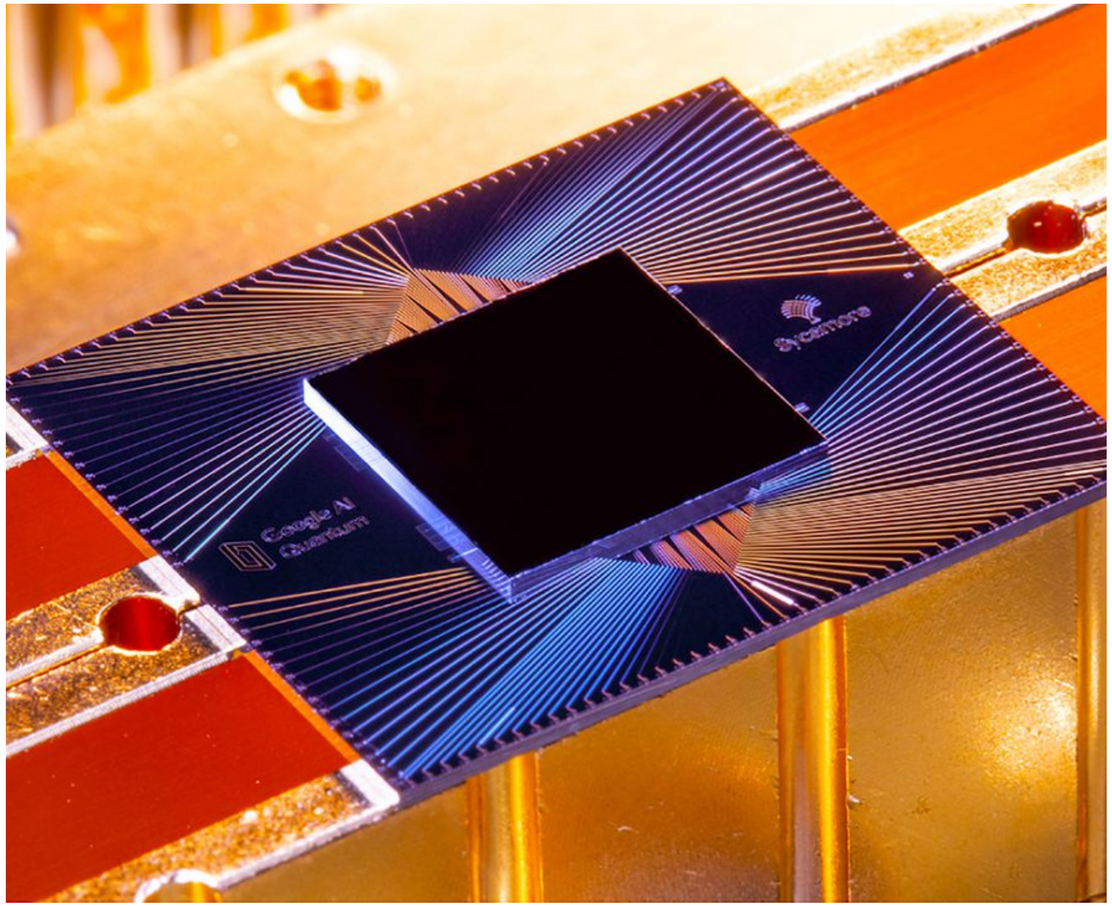
- Process:
  1. *Run a randomly chosen (known) circuit.*
  2. *Make a "small" number of measurements.*
  3. *Simulate results classically and see if they agree.*
- Computational hardness assumption.



Sycamore Protocol

Google's quantum supremacy claim

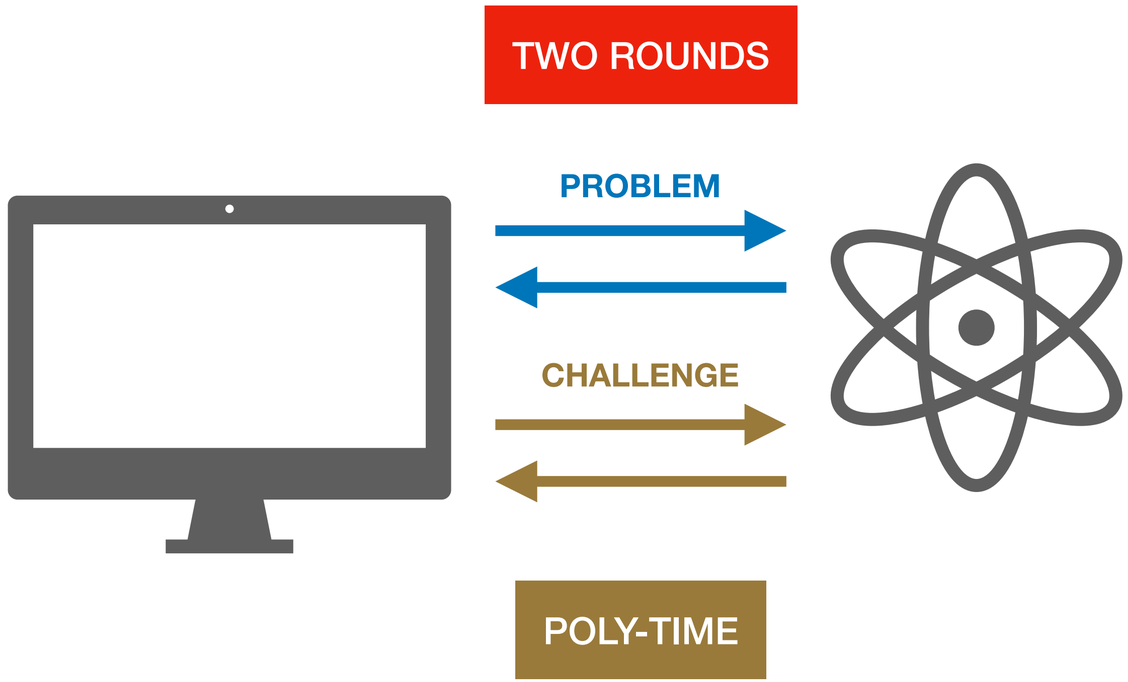




BCMVV Protocol

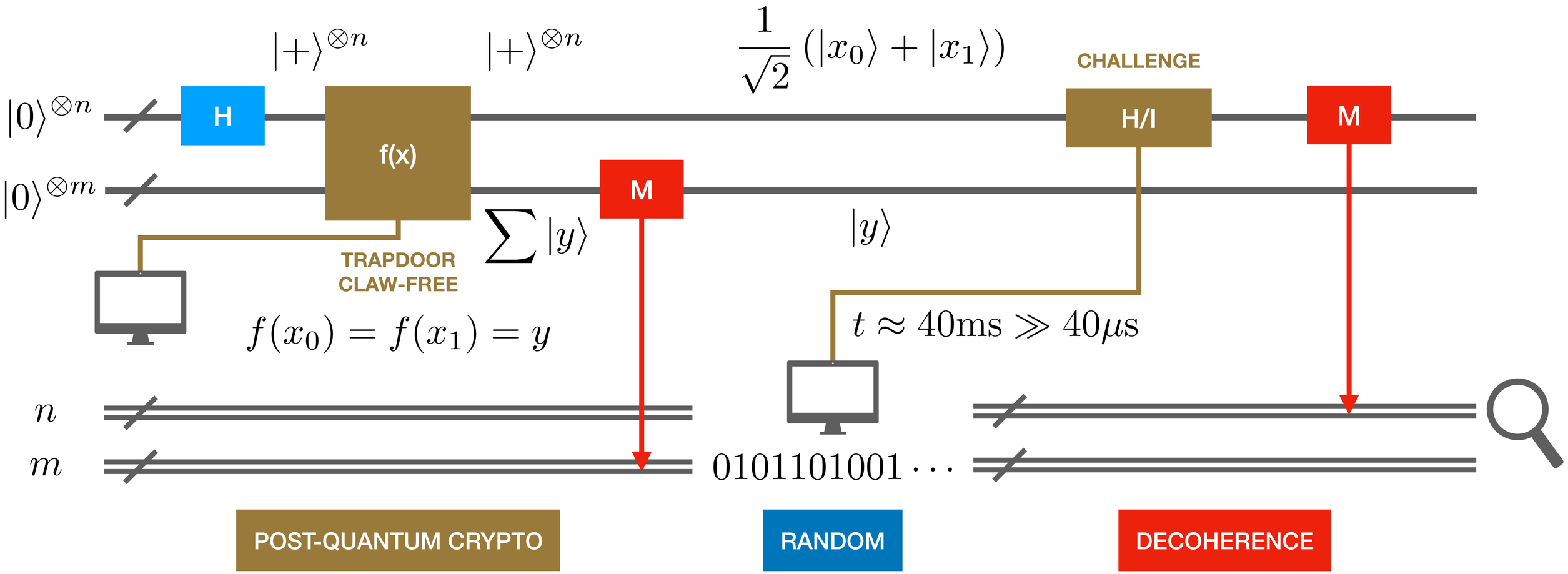
Post-quantum cryptography approach

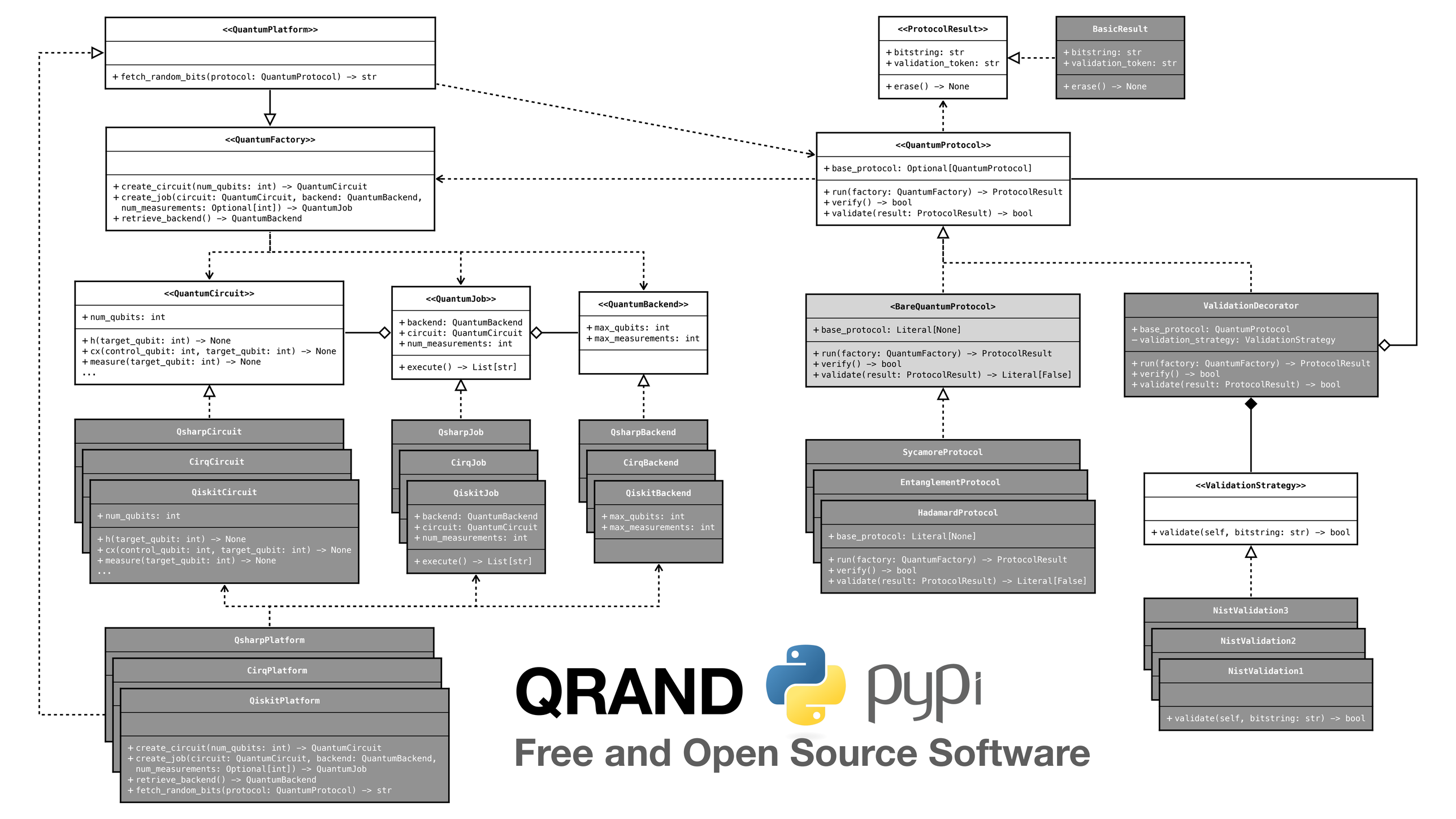
- Previous approach relied on the classical exponential overhead → *NISQ devices*
- **Post-quantum crypto** (not to be confused with quantum crypto)
- Two rounds:
  - *Problem*
  - *Challenge*



BCMVV Protocol

Post-quantum cryptography approach




<<QuantumPlatform>>
+ fetch_random_bits(protocol: QuantumProtocol) -> str
<<QuantumFactory>>
+ create_circuit(num_qubits: int) -> QuantumCircuit
+ create_job(circuit: QuantumCircuit, backend: QuantumBackend, num_measurements: Optional[int]) -> QuantumJob
+ retrieve_backend() -> QuantumBackend
<<ProtocolResult>>
+ bitstring: str
+ validation_token: str
+ erase() -> None
BasicResult
+ bitstring: str
+ validation_token: str
+ erase() -> None
<<QuantumProtocol>>
+ base_protocol: Optional[QuantumProtocol]
+ run(factory: QuantumFactory) -> ProtocolResult
+ verify() -> bool
+ validate(result: ProtocolResult) -> bool
<<QuantumCircuit>>
+ num_qubits: int
+ h(target_qubit: int) -> None
+ cx(control_qubit: int, target_qubit: int) -> None
+ measure(target_qubit: int) -> None
...
<<QuantumJob>>
+ backend: QuantumBackend
+ circuit: QuantumCircuit
+ num_measurements: int
+ execute() -> List[str]
<<QuantumBackend>>
+ max_qubits: int
+ max_measurements: int
<BareQuantumProtocol>
+ base_protocol: Literal[None]
+ run(factory: QuantumFactory) -> ProtocolResult
+ verify() -> bool
+ validate(result: ProtocolResult) -> Literal[False]
ValidationDecorator
+ base_protocol: QuantumProtocol
- validation_strategy: ValidationStrategy
+ run(factory: QuantumFactory) -> ProtocolResult
+ verify() -> bool
+ validate(result: ProtocolResult) -> bool
QsharpCircuit
CirqCircuit
QiskitCircuit
+ num_qubits: int
+ h(target_qubit: int) -> None
+ cx(control_qubit: int, target_qubit: int) -> None
+ measure(target_qubit: int) -> None
...
QsharpJob
CirqJob
QiskitJob
+ backend: QuantumBackend
+ circuit: QuantumCircuit
+ num_measurements: int
+ execute() -> List[str]
QsharpBackend
CirqBackend
QiskitBackend
+ max_qubits: int
+ max_measurements: int
SycamoreProtocol
EntanglementProtocol
HadamardProtocol
+ base_protocol: Literal[None]
+ run(factory: QuantumFactory) -> ProtocolResult
+ verify() -> bool
+ validate(result: ProtocolResult) -> Literal[False]
<<ValidationStrategy>>
+ validate(self, bitstring: str) -> bool
NistValidation3
NistValidation2
NistValidation1
+ validate(self, bitstring: str) -> bool
QsharpPlatform
CirqPlatform
QiskitPlatform
+ create_circuit(num_qubits: int) -> QuantumCircuit
+ create_job(circuit: QuantumCircuit, backend: QuantumBackend, num_measurements: Optional[int]) -> QuantumJob
+ retrieve_backend() -> QuantumBackend
+ fetch_random_bits(protocol: QuantumProtocol) -> str
QRAND
pypi
Free and Open Source Software

QRAND

Free and Open Source Software

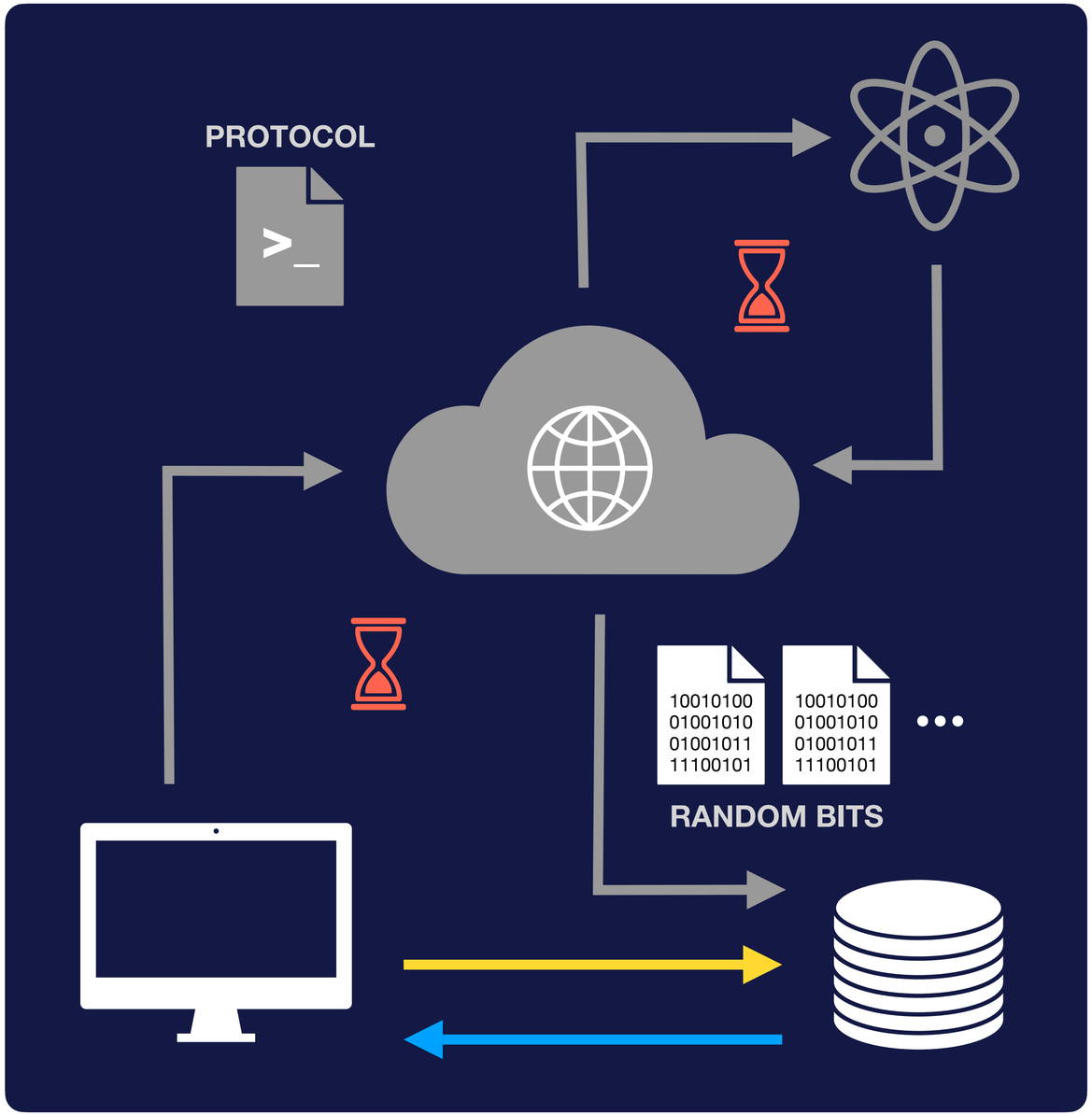
- Multiprotocol → *Hadam, Entang, ...*
- Multiplatform → ***qiskit**, Q#, cirq*
- Output formats → *bitstring, int, float, complex, hex, base64...*
- Entropy validation suite
- NumPy interface → non-uniform probability distributions
- Efficiency sampling → *Backend Select. + Multithreading + Caching*

```python
from qrand import Qrng
from qrand.platforms import QiskitPlatform
from qrand.protocols import HadamardProtocol
from qiskit import IBMQ

provider = IBMQ.load_account()
platform = QiskitPlatform(provider)
protocol = HadamardProtocol()
qrng = Qrng(platform, protocol)

s = qrng.get_random_bitstring(2)
i = qrng.get_random_int(4)
z = qrng.get_random_complex_polar(1., 3.14)
b = qrng.get_random_base32(1000)

from numpy.random import Generator
gen = Generator(qrng)

n = gen.normal()
l = gen.logistic()
p = gen.poisson()
v = gen.vonmises(0, 4.0)
```

QRAND pypi

Free and Open Source Software

- Multiprotocol → *Hadam, Entang, ...*
- Multiplatform → ***qiskit**, Q#, cirq*
- Output formats → *bitstring, int, float, complex, hex, base64...*
- Entropy validation suite
- NumPy interface → non-uniform probability distributions
- Efficiency sampling → *Backend Select. + Multithreading + Caching*



References

(1) Piani et al, *Quantum Random-Number Generators: Practical Considerations and Use Cases.* **EvolutionQ**.

(2) Abellán, *Randomness and quantum entropy w/ Carlos Abellán from Quside.* **Quantum Barcelona**.

(3) Jacak et al, *Quantum random number generators with entanglement for public randomness testing.* **Nature Research**.

(4) Arute et al, *Quantum supremacy using a programmable superconducting processor*. **Nature**.

(5) Brakerski et al, *A Cryptographic Test of Quantumness and Certifiable Randomness from a Single Quantum Device.* **arXiv:1804.00640v3**.

Thanks

Pedro Rivero

ILLINOIS TECH

CHICAGO QUANTUM EXCHANGE

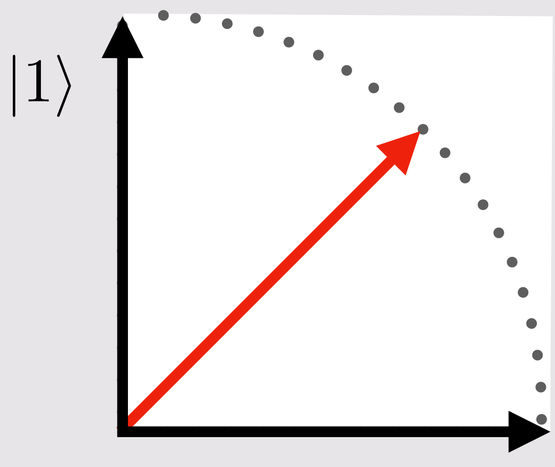
Argonne NATIONAL LABORATORY

Quantum random number generators

Quantum mechanics primer

$$\text{Bits: } \{0,1\} \quad \rightarrow \quad \text{Qubits: } \{|0\rangle, |1\rangle\}$$

- **Qubits** → quantum system with two possible (distinguishable) states
- **Superposition** → a quantum system can be in a state which is a mix of classically distinguishable states.
- **Inherently random measurements**
- **Entanglement** → classically inexplicable correlations between subsystems



$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Quantum random number generators

Trade-offs of quantum RNGs

ENTROPY SOURCE

Fundamentally random

ENTROPY QUALITY

High from the start

CERTIFICATION

Can validate the underlying process

SECURITY

Built-in with device independent techniques

SPEED

High for its quality
Slower on digital quantum computers

SIZE

Variable, from very small to room size

VERSATILITY

Easily scalable, specially with universal quantum computers

SUSTAINABILITY

Better for universal quantum computers

Migration to quantum safe solutions

Building blocks of a crypto-system

- These can all be performed quantumly:

Migration to quantum safe solutions

Why it is important to act now

COMPLIANCE WITH REGULATIONS

NIST standards

LONG MIGRATION TIME

Crypto-agile methodologies

PRODUCTS WITH LONG LIFETIMES

Car industry, self-serving

STORE NOW DECRYPT LATER

Information can be sensitive for extended periods of time

DEMO