Title : Randomisation Benchmarking Protocol

In [1]:

```
import numpy as np
from qiskit import QuantumCircuit, transpile, Aer, execute

# Generate a random quantum circuit
def generate_random_circuit(num_qubits, depth):
    circuit = QuantumCircuit(num_qubits, num_qubits)
    for _ in range(depth):
        for qubit in range(num_qubits):
            circuit.rx(np.random.uniform(0, 2 * np.pi), qubit)
            circuit.ry(np.random.uniform(0, 2 * np.pi), qubit)
            circuit.rz(np.random.uniform(0, 2 * np.pi), qubit)
        for qubit in range(num_qubits - 1):
            circuit.cz(qubit, qubit + 1)
    return circuit

# Perform randomized benchmarking
def randomized_benchmarking(num_qubits, depths, num_sequences, shots):
    backend = Aer.get_backend('statevector_simulator')
    results = []
    for depth in depths:
        success_counts = 0
        for _ in range(num_sequences):
            # Generate a random circuit and the corresponding inverse circuit
            circuit = generate_random_circuit(num_qubits, depth)
            inverse_circuit = circuit.inverse()

            # Apply the circuit and obtain the final statevector
            circuit_result = execute(circuit, backend=backend).result()
            final_statevector = circuit_result.get_statevector()

            # Apply the inverse circuit and obtain the final statevector
            inverse_result = execute(inverse_circuit, backend=backend).result()
            inverse_statevector = inverse_result.get_statevector()

            # Calculate the success rate based on state fidelity
            fidelity = np.abs(np.dot(final_statevector, inverse_statevector.cor
            success_counts += shots * (1 - fidelity)

        success_rate = success_counts / (num_sequences * shots)
        results.append(success_rate)
    return results

# Example usage
num_qubits = 2
depths = [1, 2, 3, 4]
num_sequences = 100
shots = 1024

results = randomized_benchmarking(num_qubits, depths, num_sequences, shots)
print(results)
```

```
/tmp/ipykernel_1339/4273929913.py:36: DeprecationWarning: The return type of
saved statevectors has been changed from a `numpy.ndarray` to a `qiskit.quant
um_info.Statevector` as of qiskit-aer 0.10. Accessing numpy array attributes
is deprecated and will result in an error in a future release. To continue us
ing saved result objects as arrays you can explicitly cast them using  `np.as
array(object)`.
  fidelity = np.abs(np.dot(final_statevector, inverse_statevector.conj())) **
2

[0.6083172764220851, 0.6932170742787327, 0.7039259927778176, 0.71867855893384
06]
```

In [ ]: