# Shor’s Algorithm Implementation

Scott McHaffie, Jai Anand Iyer, Venkatesh Elayaraja

Department of Applied Physics

Kungliga Tekniska Högskolan (KTH)

# Chapters

# Project Scope

- **Problem Addressed:**
  - Shor's Algorithm
- **Project Direction:**
  - Limitations of Full-Adder based Modulo Function
  - Classical implementation of Modular Exponentiation
- **Study Setup:**
  - Unit tests: N = [15, ..., 100], base = [2, 3]
- **Software:**
  - Language: Python
  - Packages: Qiskit, NumPy, Matplotlib

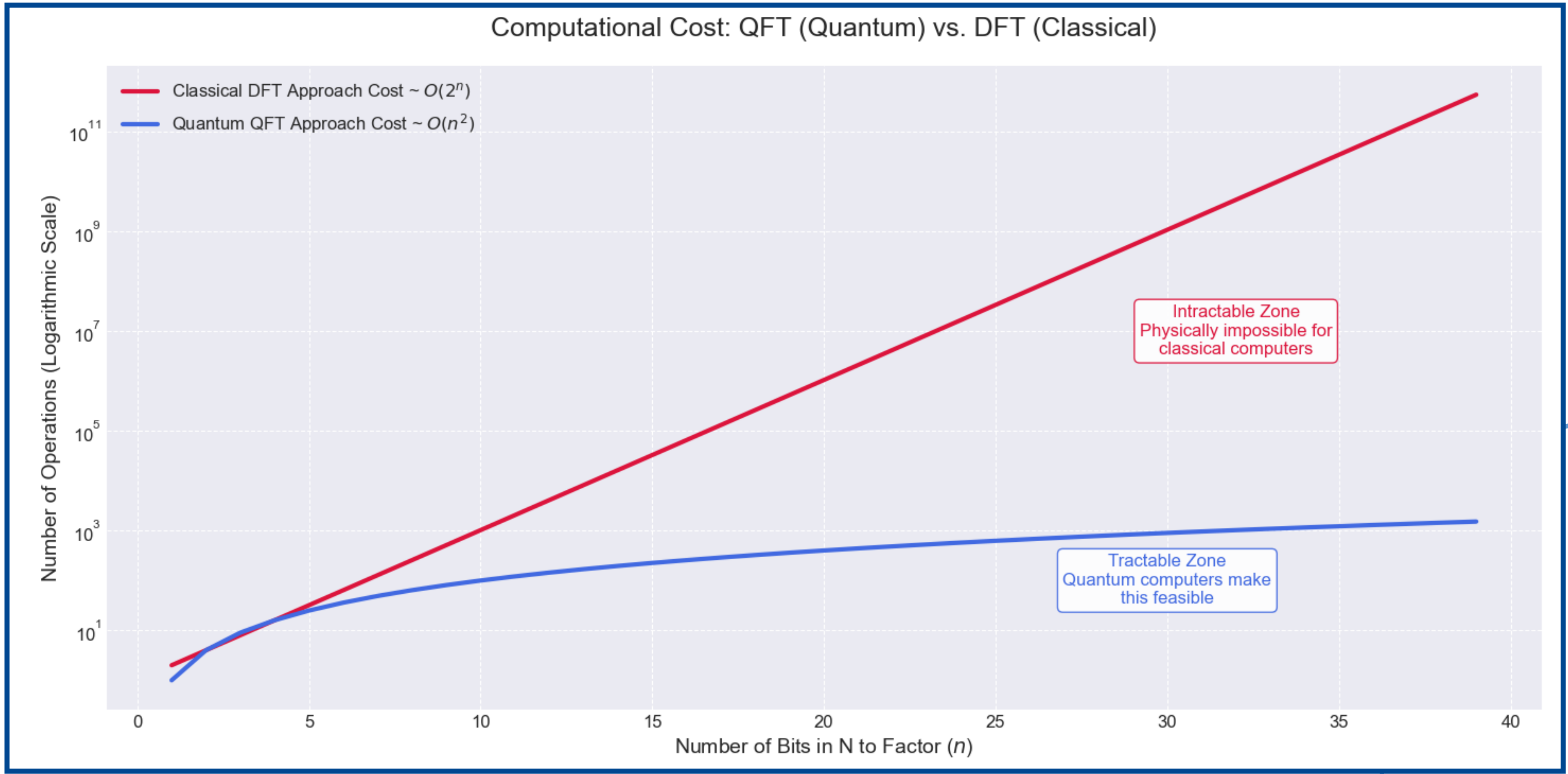
# Background

- Classical Prime Factorization runtime
  - **Generative Number Field Sieve** algorithm (GNFS): $\mathcal{O}(e^{(lnN)^{1/3}(lnlnN)^{2/3}})$ → Sub-exponential, not efficient for large N
  - Hard problem for classical computing, basis for RSA 256, 2048, etc. Public-Key Encryption systems

- Shor's Algorithm for Prime Factorization
  - Achieves polynomial runtime $\mathcal{O}(\ln N^3)$ with high probability
  - Core ideas:
    - Equivalence of prime factorization problem and **period finding** for $a^x mod\ N$
    - Period finding using **Quantum Fourier Transform** → Speedup over classical FFT
    - **Parallelization** of modular exponential computation

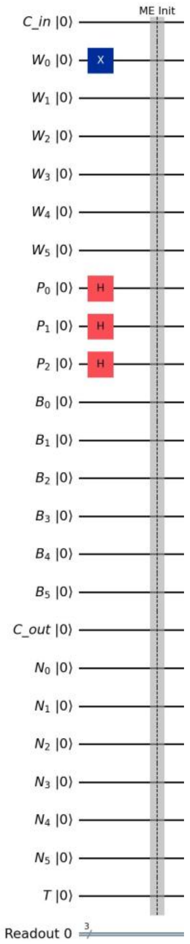
# Background

## DFT vs QFT

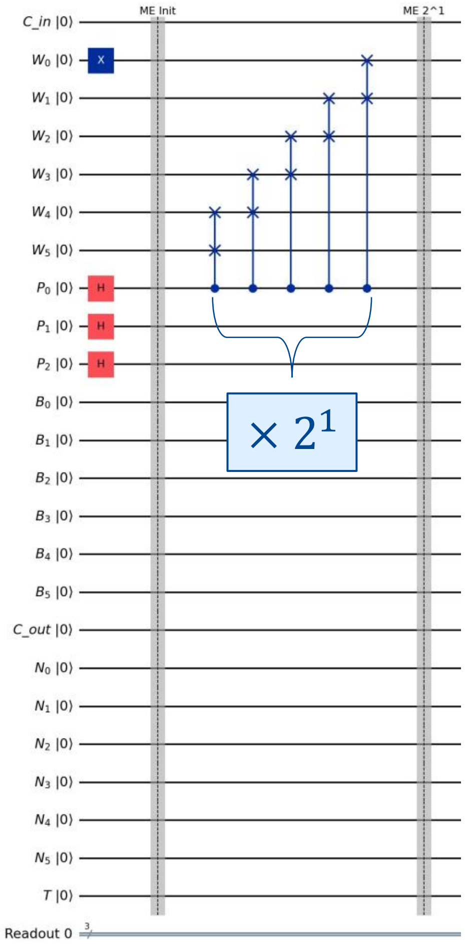



- Classical DFT: Exponential Time - $O(2^n)$. Hits an "impossibility wall".
- Quantum QFT: Polynomial Time - $O(n^2)$. Stays tractable and efficient.

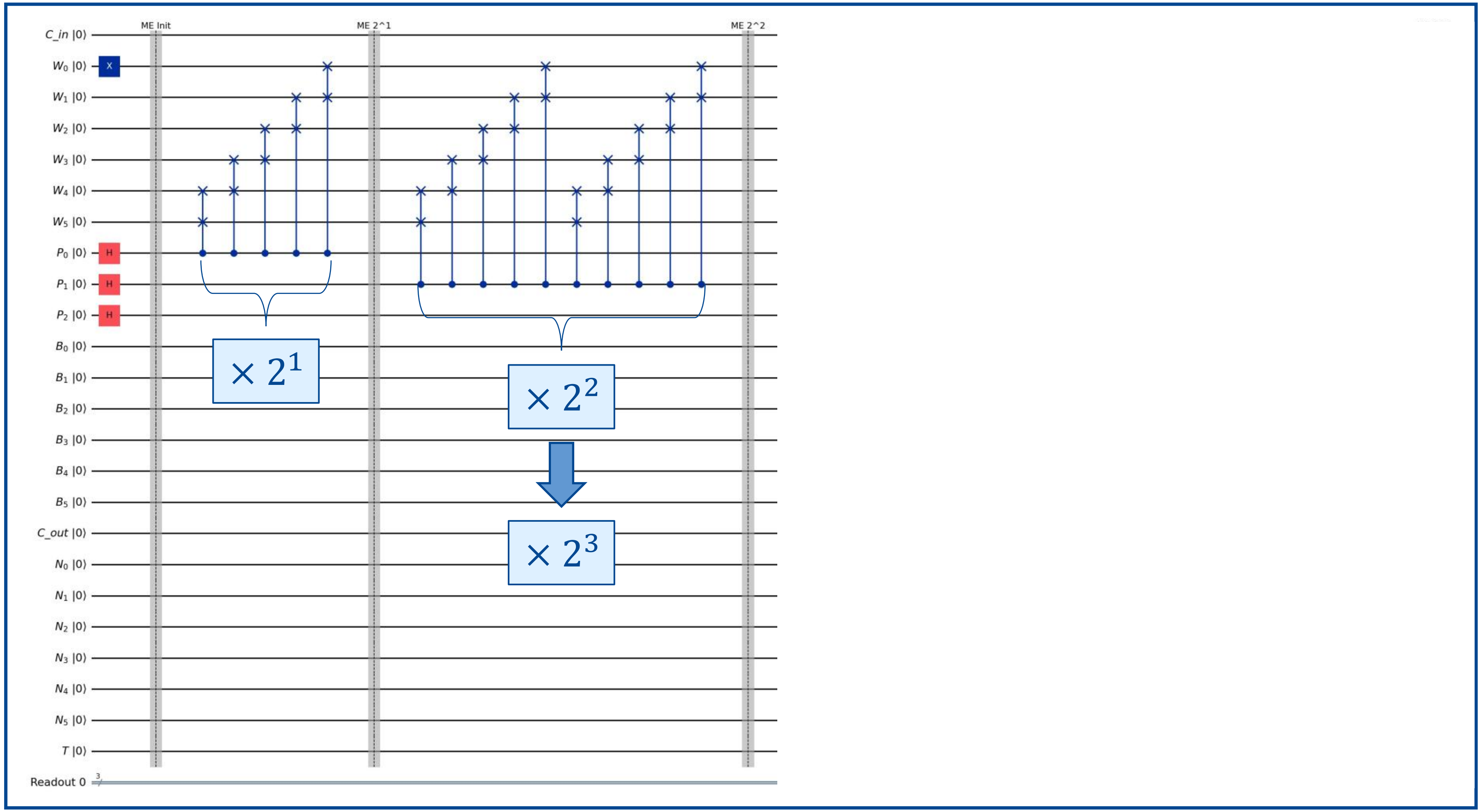
# Methodology
## Quantum Exponentiation



# Methodology
## Quantum Exponentiation

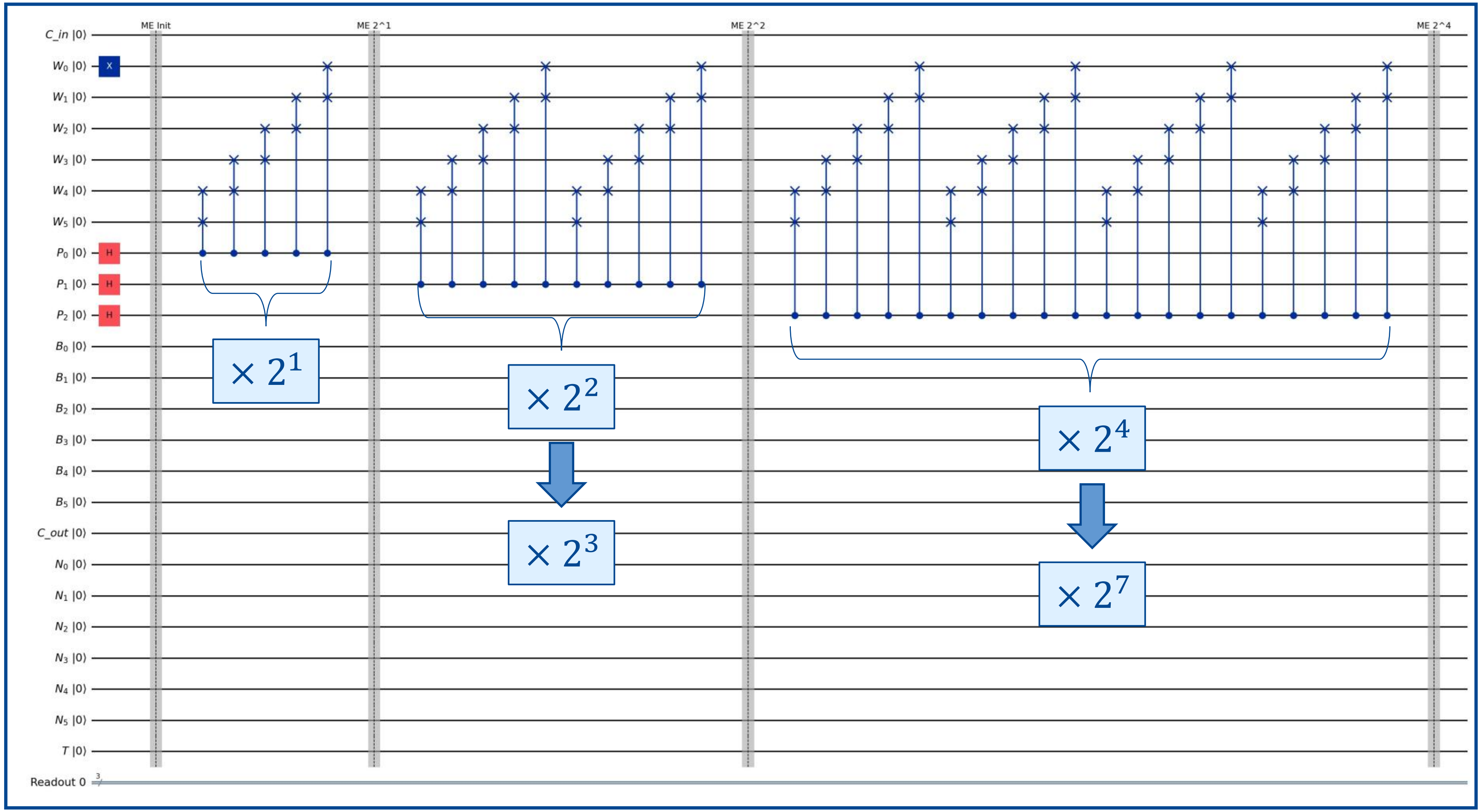


# Methodology
## Quantum Exponentiation



# Methodology

## Quantum Exponentiation



# Methodology
## Quantum Adder Mod N

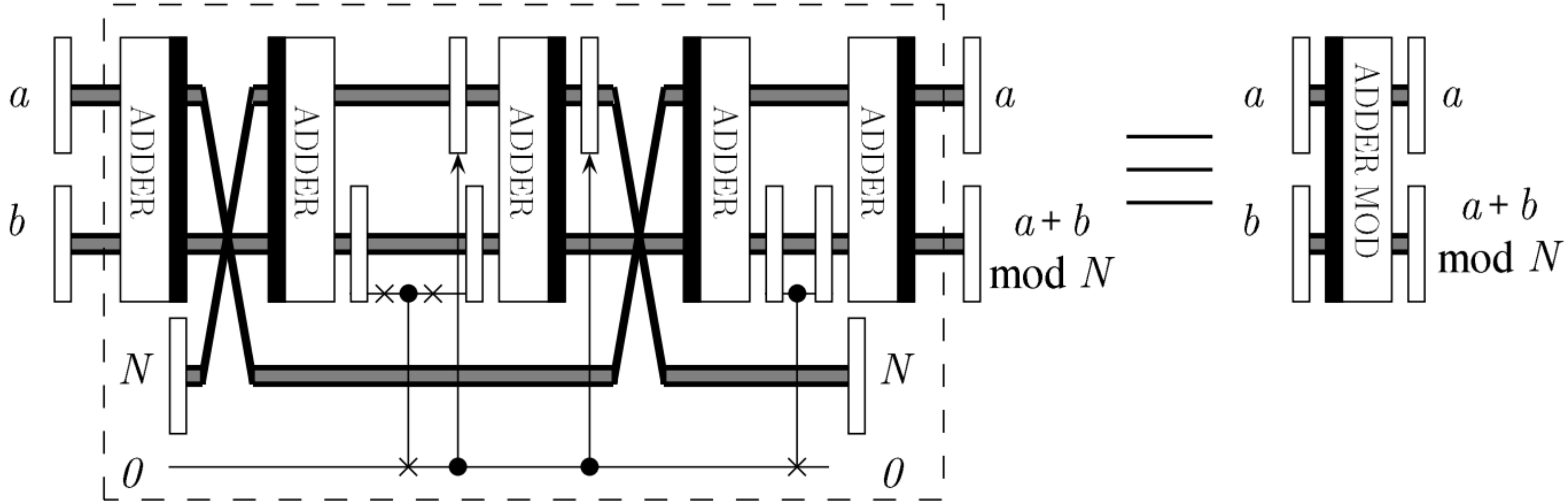
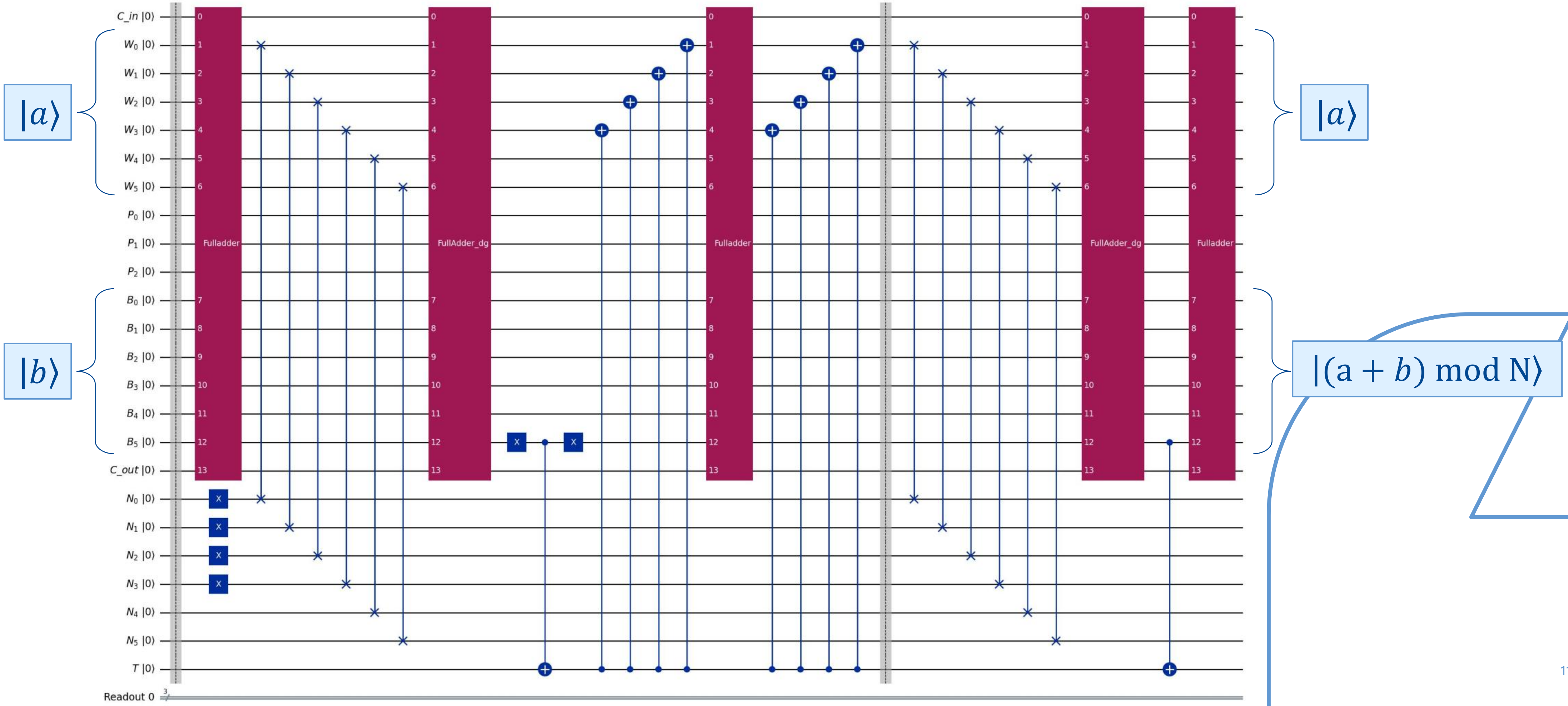



Figure is taken from [2]

# Methodology
## Quantum Adder Mod N



# Methodology

## Classical Modular Exponentiation

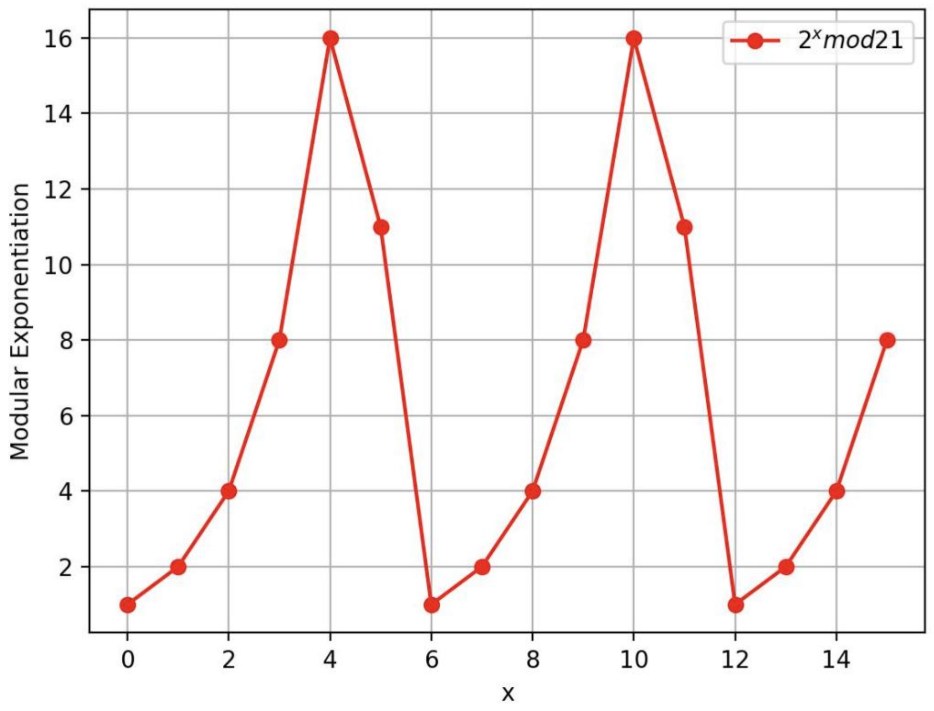
- Step 1: Evaluate $a^x mod\ N$ classically and store in a list as binary values
  - `signal_binary: [‘000001’,…]`
  - `signal_size =  (2 ** precision_bits )`

- Step 2:  Initialize working register to $|a^x mod\ N\rangle$ conditioned on precision register state $|x\rangle$

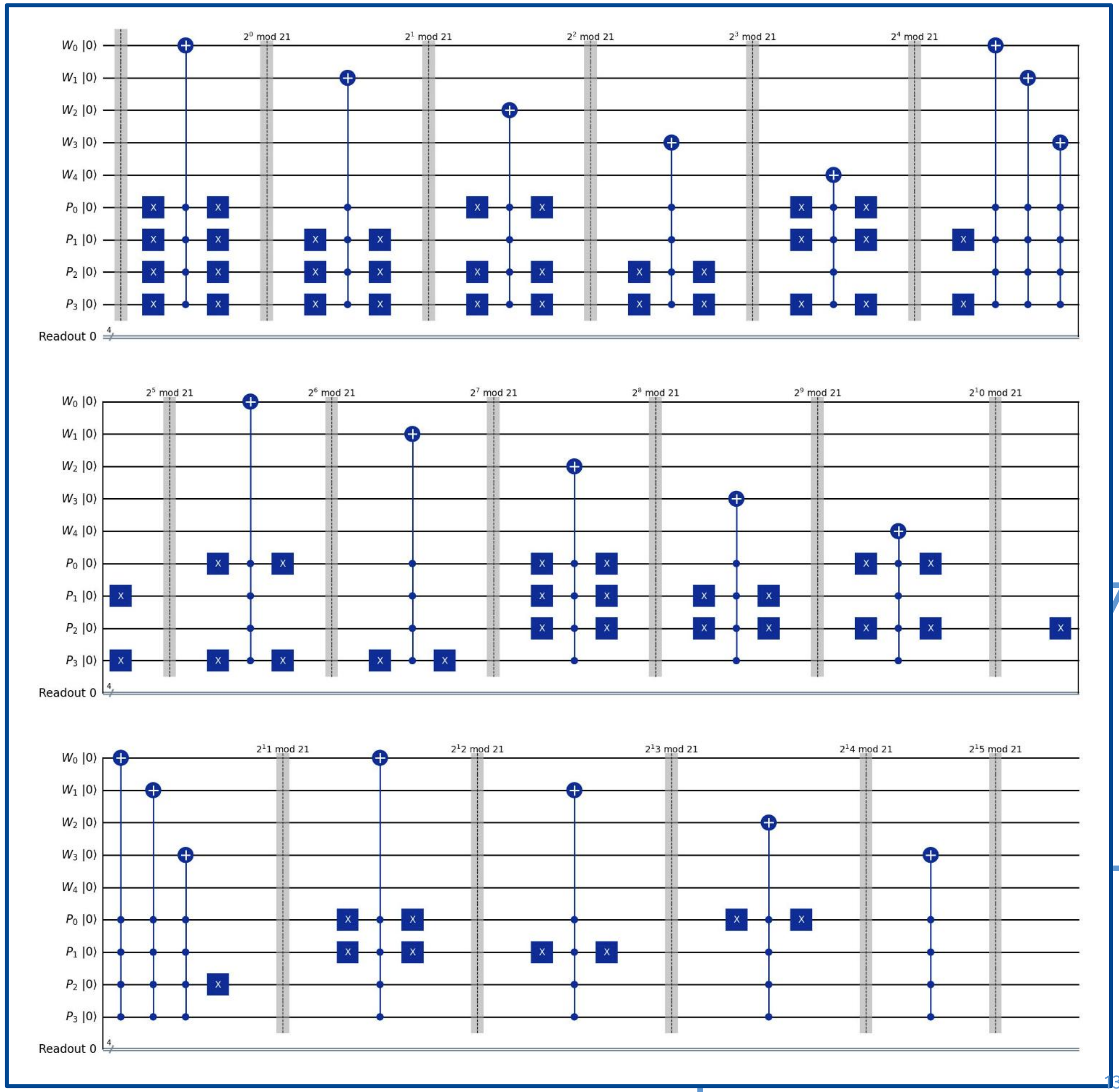
# Methodology

## Classical Modular Exponentiation

Signal for 2^x mod 21



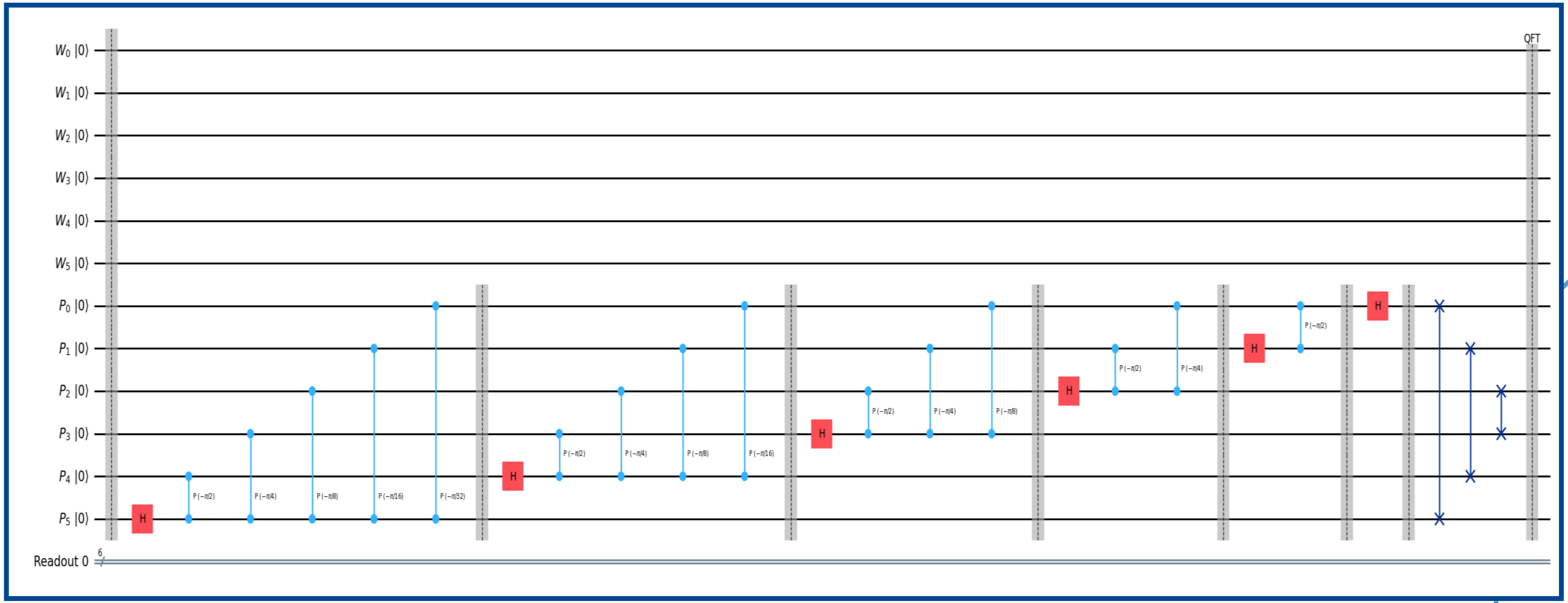

Conditional Initialization



# Methodology

## Quantum Fourier Transformation

$$\mathrm{QFT}|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} |k\rangle$$



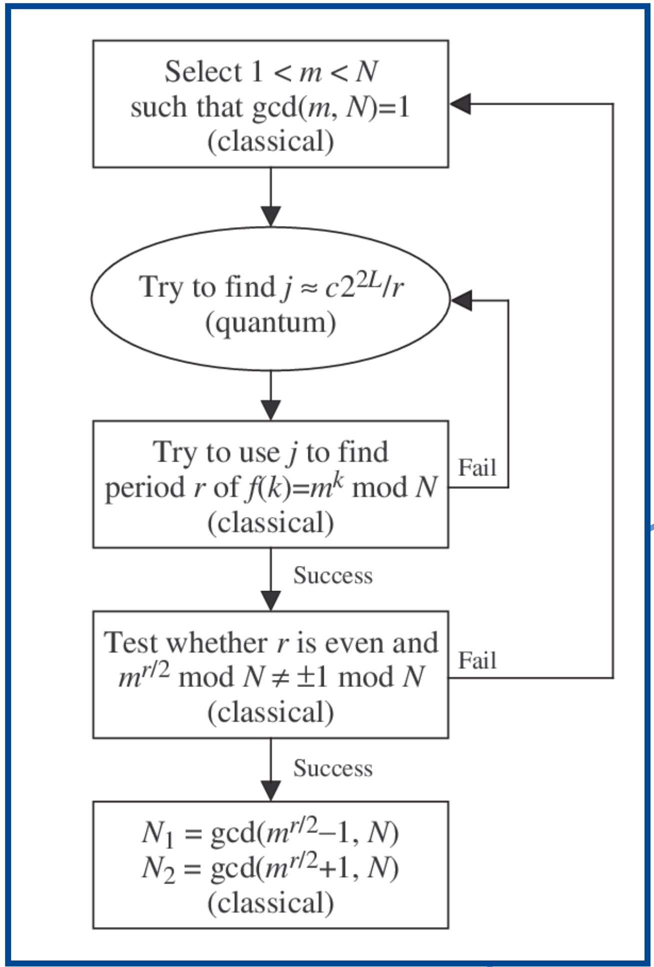
# Methodology

## Classical Components

This part assumes a quantum computer has successfully found the period r of the function $a^x (\mathrm{mod}\ N)$.

- Validate the period r:
  - If r is odd, the attempt fails. **Restart** with a new random a.
  - If $a^{r/2} (\mathrm{mod}\ N) \equiv 1$ the attempt also fails. **Restart**.
- Calculate the factors: If the checks pass, the factors of N are found by computing:

$$\gcd\left(a^{r/2} - 1, N\right) \quad \text{and} \quad \gcd\left(a^{r/2} + 1, N\right)$$

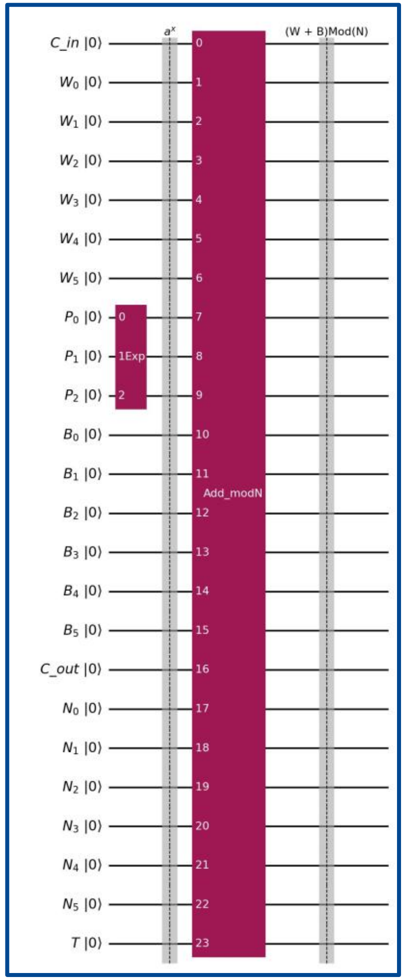


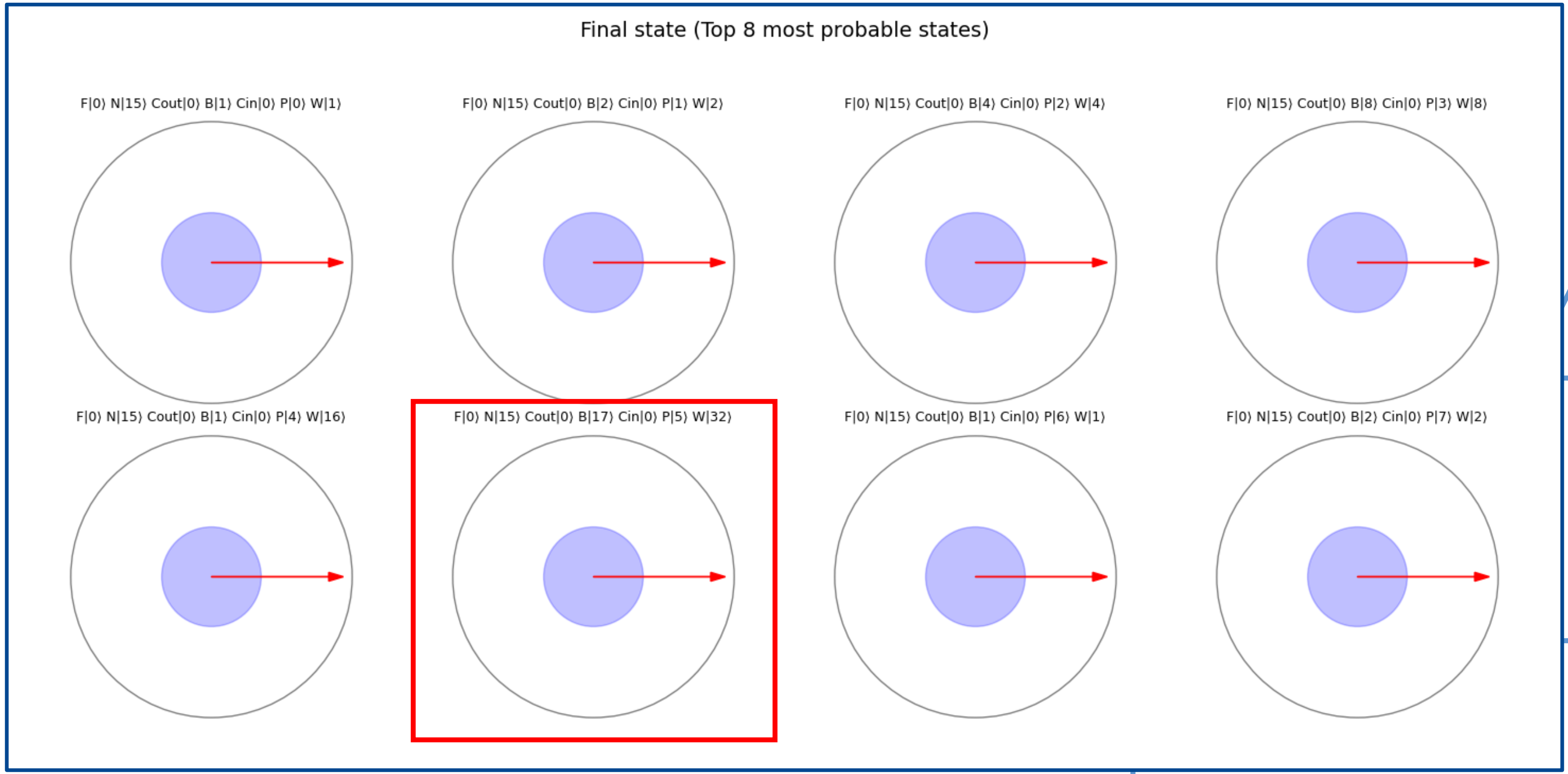
# Implementation

- Implemented on the Aer simulator
  - Ran into syntax trouble trying to call backends
  - Running 2048 shots
  - Using up to 7 working bits (allows up to 128) and 6 precision bits (to allow for 2^63 mod (N) in the precision register) to ensure spike finding algorithm works
- From results, run the classical spike finding and factoring algorithms on ALL measured states, returning once any prime factors are found

# Experiments & Results – Mod 15

## Quantum Modulo Exponentiation: $a = 2, N = 15$



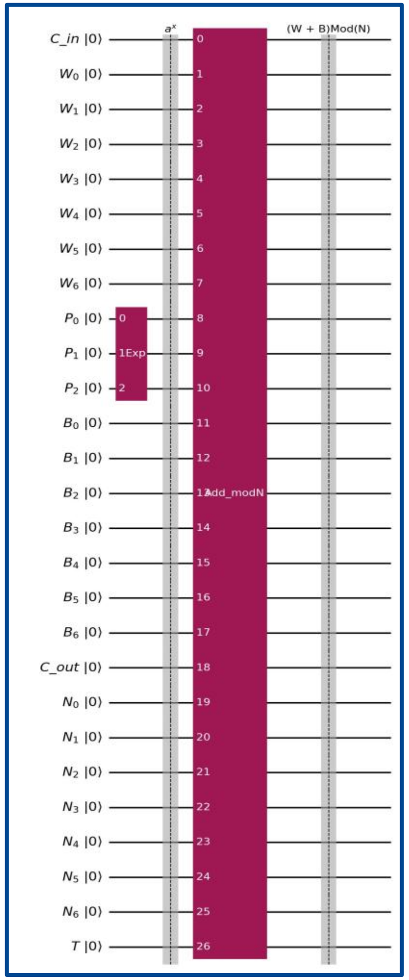



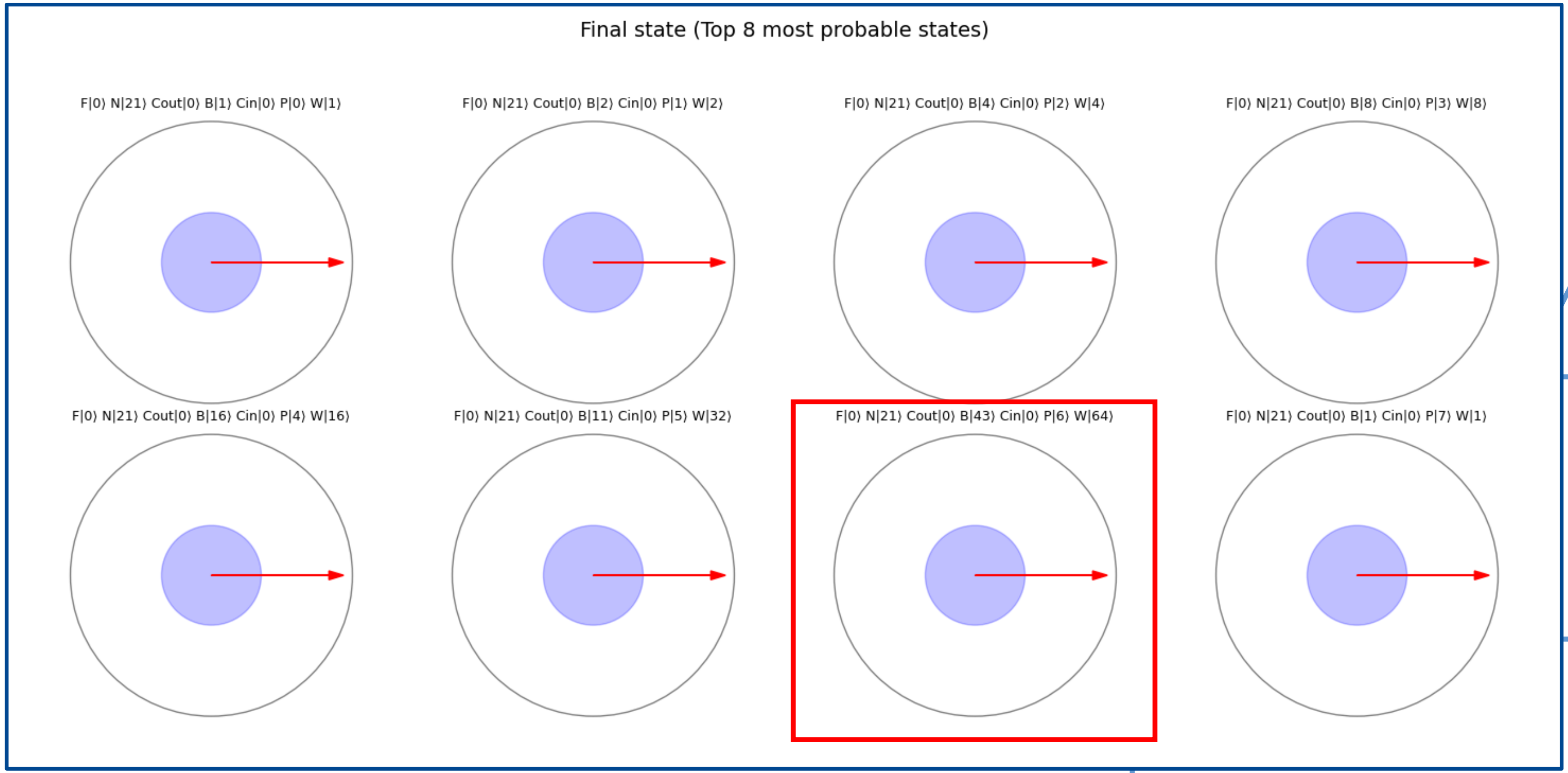

$32 \ \% \ 15 \neq 17$

$32 \ \% \ 15 = 2$

# Experiments & Results – Mod 21

## Quantum Modulo Exponentiation: $a = 2, N = 21$



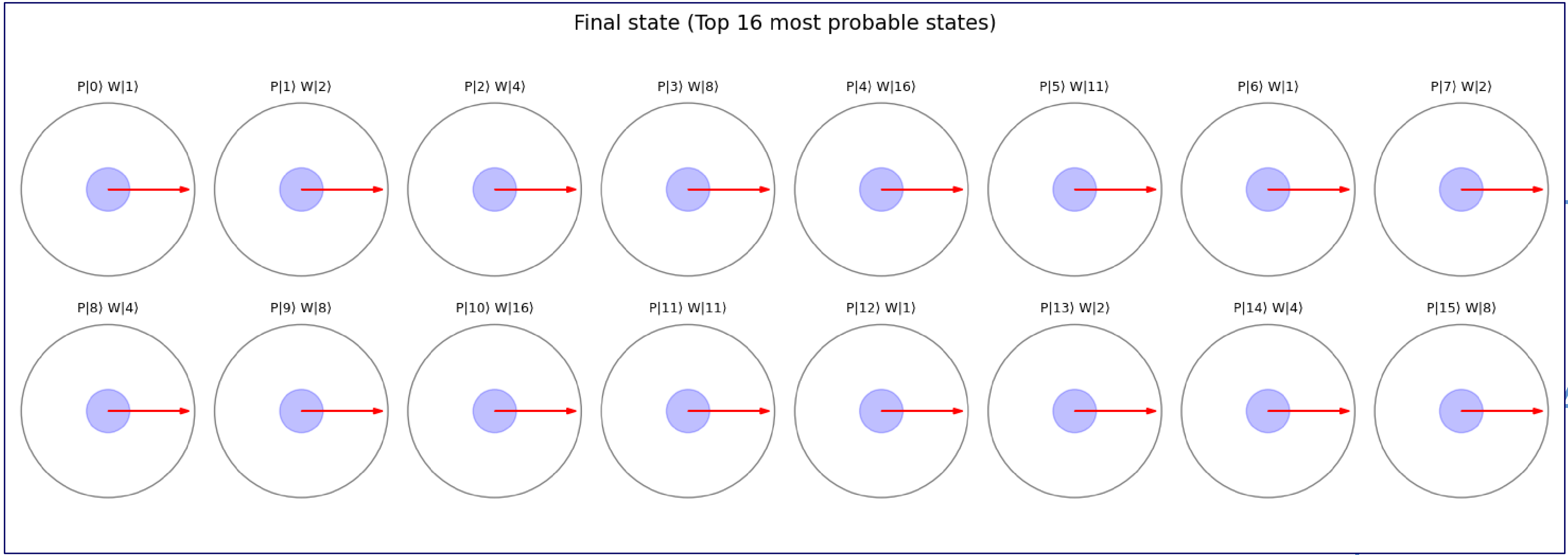




$$64 \% 21 \neq 43$$
$$64 \% 21 = 1$$

# Experiments & Results

## Classical Modulo Exponentiation - Conditional Initialization

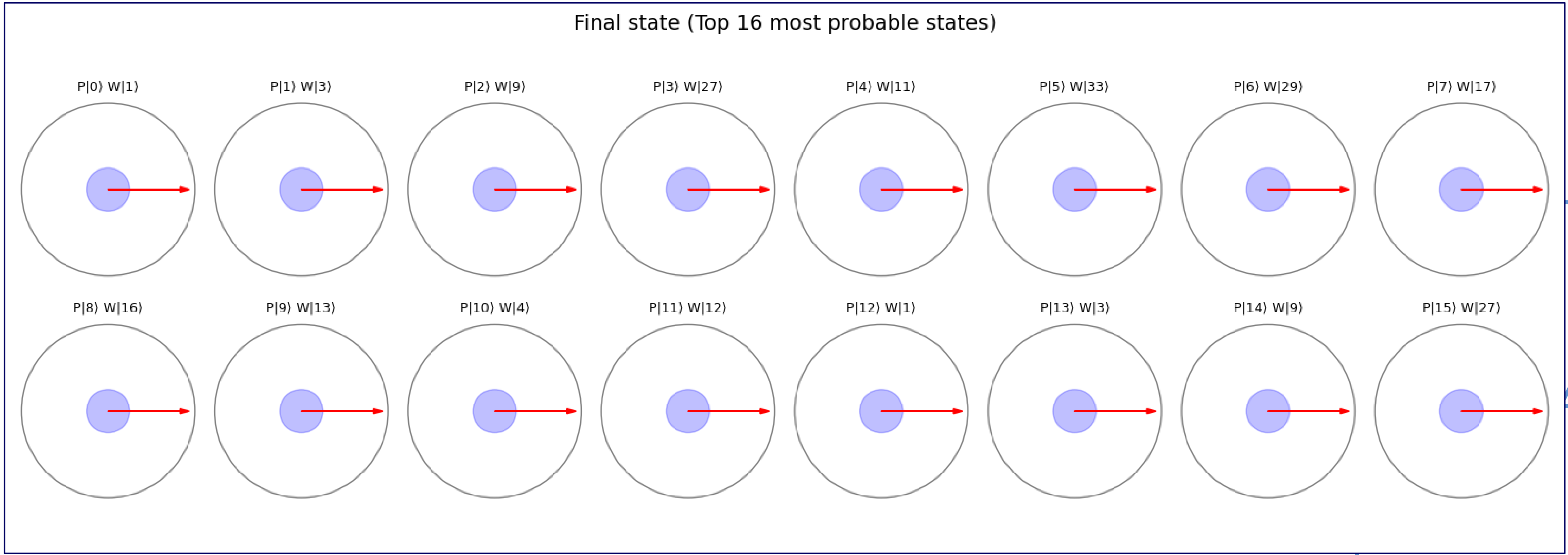
a = 2, N = 21



# Experiments & Results

## Classical Modulo Exponentiation - Conditional Initialization

a =3, N = 35



# Experiments & Results

## Classical Peak finding from QFT Probabilities


```
N = 15, a = 2
True order r = 4
Detected spike positions: [16, 32, 48]
Estimated order (from continued fractions) = 4
Recovered nontrivial factors: 3 and 5
```

# Experiments & Results

## Classical Modulo Exponentiation – Results from Prime Factorization

```
Base =  2
Prime factors of  15 :  (3, 5)
Prime factors of  16 :  (2, 8)
Prime factors of  21 :  (7, 3)
Prime factors of  24 :  (2, 12)
Prime factors of  32 :  (2, 16)
Prime factors of  33 :  (11, 3)
Prime factors of  35 :  (7, 5)
Prime factors of  39 :  (3, 13)
Prime factors of  40 :  (10, 4)
Prime factors of  45 :  (9, 5)
Prime factors of  48 :  (8, 6)
Prime factors of  51 :  (3, 17)
Prime factors of  55 :  (11, 5)
Prime factors of  56 :  (4, 14)
Prime factors of  57 :  (19, 3)
Prime factors of  63 :  (7, 9)
Prime factors of  65 :  (5, 13)
Prime factors of  69 :  (23, 3)
Prime factors of  72 :  (2, 36)
Prime factors of  75 :  (3, 25)
Prime factors of  77 :  (7, 11)
Prime factors of  80 :  (40, 2)
Prime factors of  85 :  (5, 17)
Prime factors of  87 :  (3, 29)
Prime factors of  88 :  (44, 2)
Prime factors of  91 :  (7, 13)
Prime factors of  93 :  (31, 3)
Prime factors of  95 :  (19, 5)
Prime factors of  96 :  (2, 48)
```

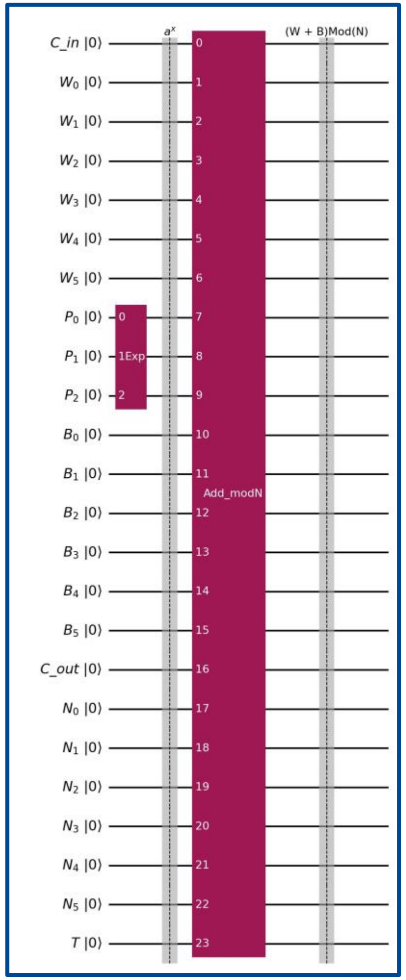
```
Base =  3
Prime factors of  15 :  (5, 3)
Prime factors of  21 :  (7, 3)
Prime factors of  24 :  (2, 12)
Prime factors of  25 :  (5, 5)
Prime factors of  32 :  (8, 4)
Prime factors of  33 :  (3, 11)
Prime factors of  35 :  (7, 5)
Prime factors of  39 :  (13, 3)
Prime factors of  45 :  (5, 9)
Prime factors of  48 :  (6, 8)
Prime factors of  49 :  (7, 7)
Prime factors of  55 :  (11, 5)
Prime factors of  56 :  (2, 28)
Prime factors of  57 :  (19, 3)
Prime factors of  63 :  (7, 9)
Prime factors of  64 :  (8, 8)
Prime factors of  65 :  (13, 5)
Prime factors of  69 :  (3, 23)
Prime factors of  72 :  (2, 36)
Prime factors of  77 :  (11, 7)
Prime factors of  81 :  (9, 9)
Prime factors of  88 :  (22, 4)
Prime factors of  91 :  (13, 7)
Prime factors of  93 :  (3, 31)
Prime factors of  95 :  (19, 5)
Prime factors of  96 :  (8, 12)
Prime factors of  99 :  (11, 9)
```

# Discussion – Prime Factorization

- We implemented Shor's Algorithm to search for factors of all integers from 15 to 100, using the base value = {2, 3}
- Using base 2, we found prime factors of all integers of the form N = p * q where {p, q} are prime numbers
- Using base 3, we factorized less numbers → need to use bigger precision register, increases computation time
- Using base 5 and above is difficult due to very large numbers from exponentiation → can try repeated modulo

# Discussion – Mod 33 and Above

## Quantum Modulo Exponentiation




- To compute $2^x \ \% \ 33$ to the point of failure we need 8 working bits to avoid overflow
- This drastically increases run time and each run took too long to compute state vectors
- Expect it to fail at $128 \ \% \ 33$
  - Will return $128 - 33 = 95$ instead of:
    $128 \ \% \ 33 = 29$

# Discussion – The Failure of the Modulo Adder

## Quantum Modulo Exponentiation

- Ultimately the "modulo adder" implements $a + b - N$

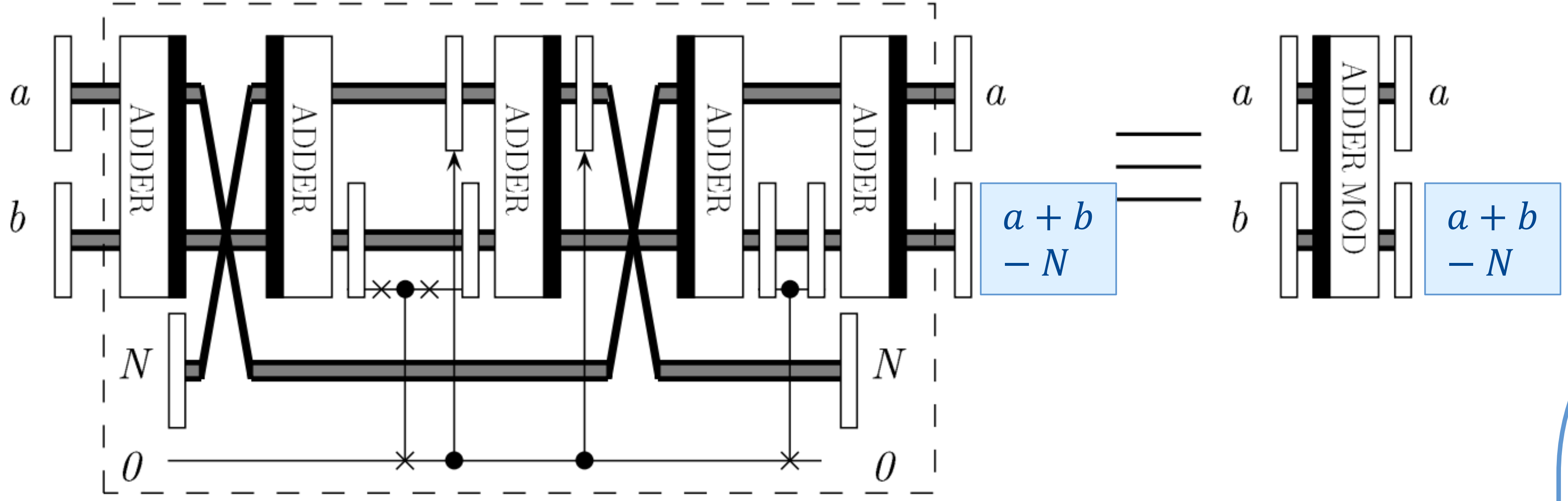



Figure is adapted from [2]

- Therefore, the adder is only valid for $a + b < 2N$

# Work Division & Expected Grade

- We expect a final grade of A. We implemented Shor's Algorithm and:
  - Compared quantum to classical modular exponentiation
  - Completed unit tests from N = [15, …, 100] with base = [2, 3]
- This report meets the A grade

## Individual Contributions

- Scott McHaffie:
  - Developed code for Quantum Addition Modulo N, running the quantum circuit, the digital logic to process the results and return prime factors
- Jai Anand Iyer:
  - Developed code for Classical Modular Exponential Initialization, Quantum Exponentiation, Quantum Addition Modulo N
- Venkatesh Elayaraja:
  - Developed code for the Quantum Fourier Transform (QFT) and its inverse
  - Developed unit tests for the Shor's circuits, and classical code, Optimized visualizations for probability distribution of state vectors

# References


1. Markidis, S., 2025. *Lecture slides: DD2367 Quantum Computing for Computer Scientists.* KTH Royal Institute of Technology.
2. Vedral, V., Barenco, A. and Ekert, A., 1996. *Quantum networks for elementary arithmetic operations. Physical Review A, 54(1), pp.147–153.* doi:10.1103/PhysRevA.54.147.
3. Fowler, Austin., 2005. Towards Large-Scale Quantum Computation. https://arxiv.org/abs/quant-ph/0506126