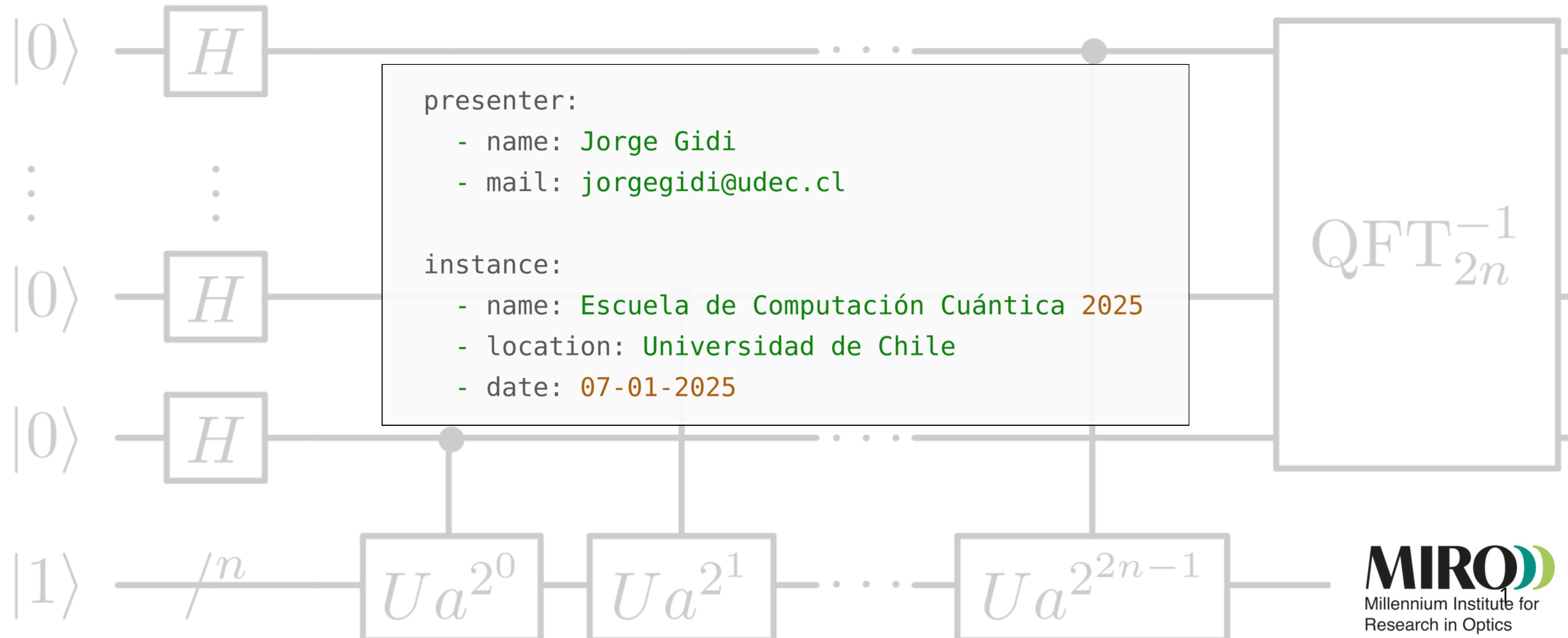


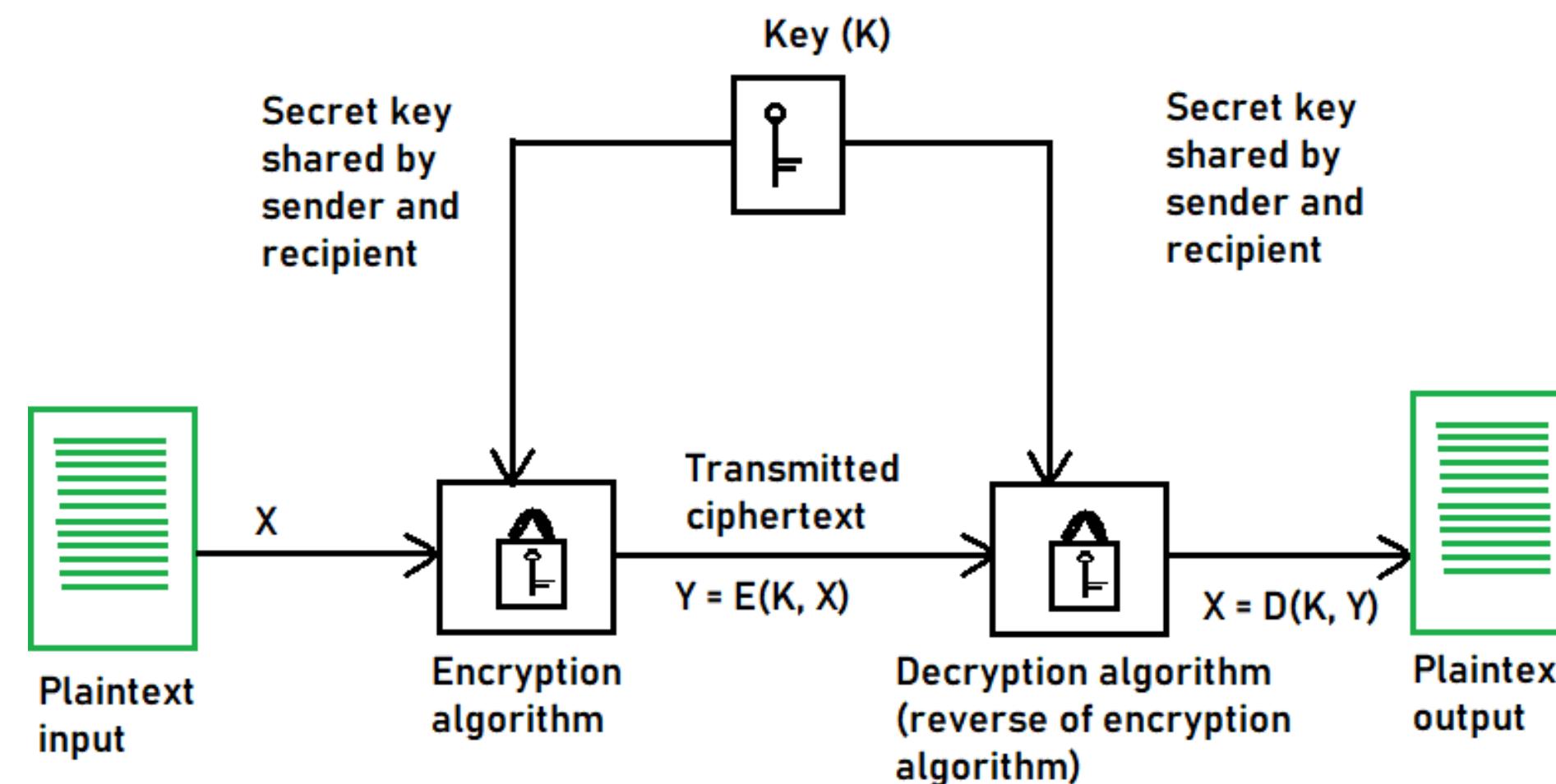
RSA y algoritmo de Shor



Criptografía simétrica

Criptografía: Protección de la información

Históricamente la criptografía era **simétrica**, es decir, se usaba una misma clave compartida para cifrar y descifrar un mensaje.

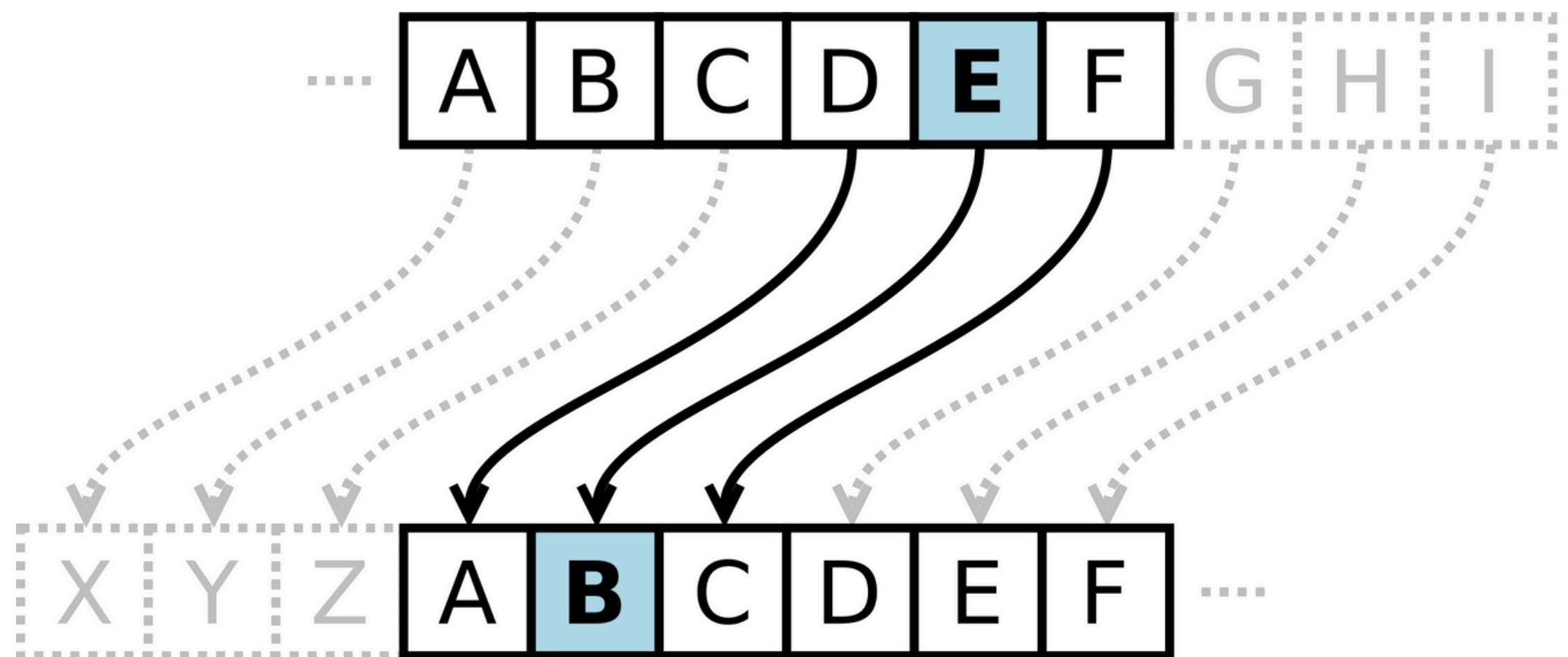


Criptografía simétrica

Ejemplo: Cifrado Cesar (50 AC)

Se desplaza cada letra del alfabeto en 3 posiciones

HOLA
↓
+3 a cada letra
KRÑD
↓
-3 a cada letra
HOLA



Criptografía simétrica segura

Ejemplo: One-time Pad

- Se genera clave aleatoria del mismo largo que el mensaje
- Se traducen el mensaje y la clave en binario.
- Para encriptar, se suma en binario (módulo 2) cada dígito de la clave y el mensaje
- Para desencriptar, se suma nuevamente la clave con el mensaje encriptado

ENCRYPT		
\oplus	00110101	Plaintext
<hr/>		
\oplus	11100011	Secret Key
<hr/>		
$=$	11010110	Ciphertext
DECRYPT		
\oplus	11010110	Ciphertext
<hr/>		
\oplus	11100011	Secret Key
<hr/>		
$=$	00110101	Plaintext

Imagen de Pelosi et al (2018). Advances in Science, Technology and Engineering Systems Journal. 3. 271-282

Es perfectamente seguro **siempre y cuando** la contraseña sea usada sólo una vez, sea aleatoria, y nadie mas que emisor y receptor la conozcan (problema de distribución)

Seguridad de clave pública

En 1976 Whitfield Diffie and Martin Hellman introducen un concepto revolucionario: criptografía de clave pública

Dos extremos pueden establecer una clave de forma segura, en un canal público, sin haber compartido una contraseña. Es teóricamente posible reconstruir el secreto que comparten, pero en la práctica requiere resolver problemas que no cuentan con solución eficiente.

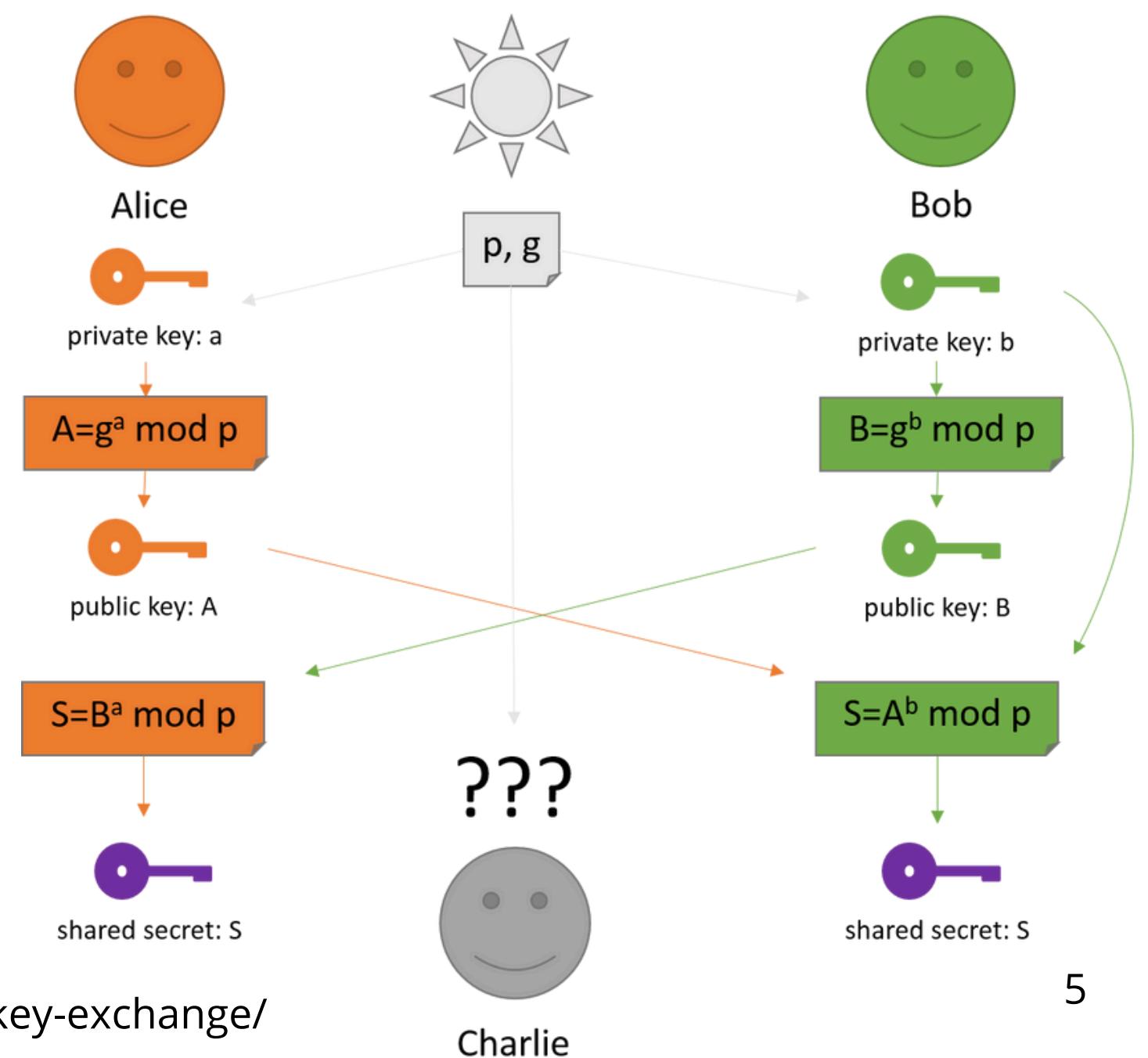


Diagrama de <https://www.noser.com/techblog/asymmetric-cryptography-diffie-hellman-key-exchange/>

Diffie, W., and M. Hellman. (1976). IEEE Transactions on Information Theory 22(6):644–54

Rivest-Shamir-Adleman (RSA)



Ron Rivest, Adi Shamir y Leonard Adleman se inspiran en la idea de Diffie-Hellman, y diseñan el algoritmo RSA.

RSA fue el primer sistema práctico que permite el intercambio de claves, crifrado directo y firmas digitales.

Mientras DH se basa en el problema de logaritmo discreto, RSA se basa en la factorización de números grandes.

Complejidad

Nosotros hablaremos de complejidad en tiempo.

Notación:

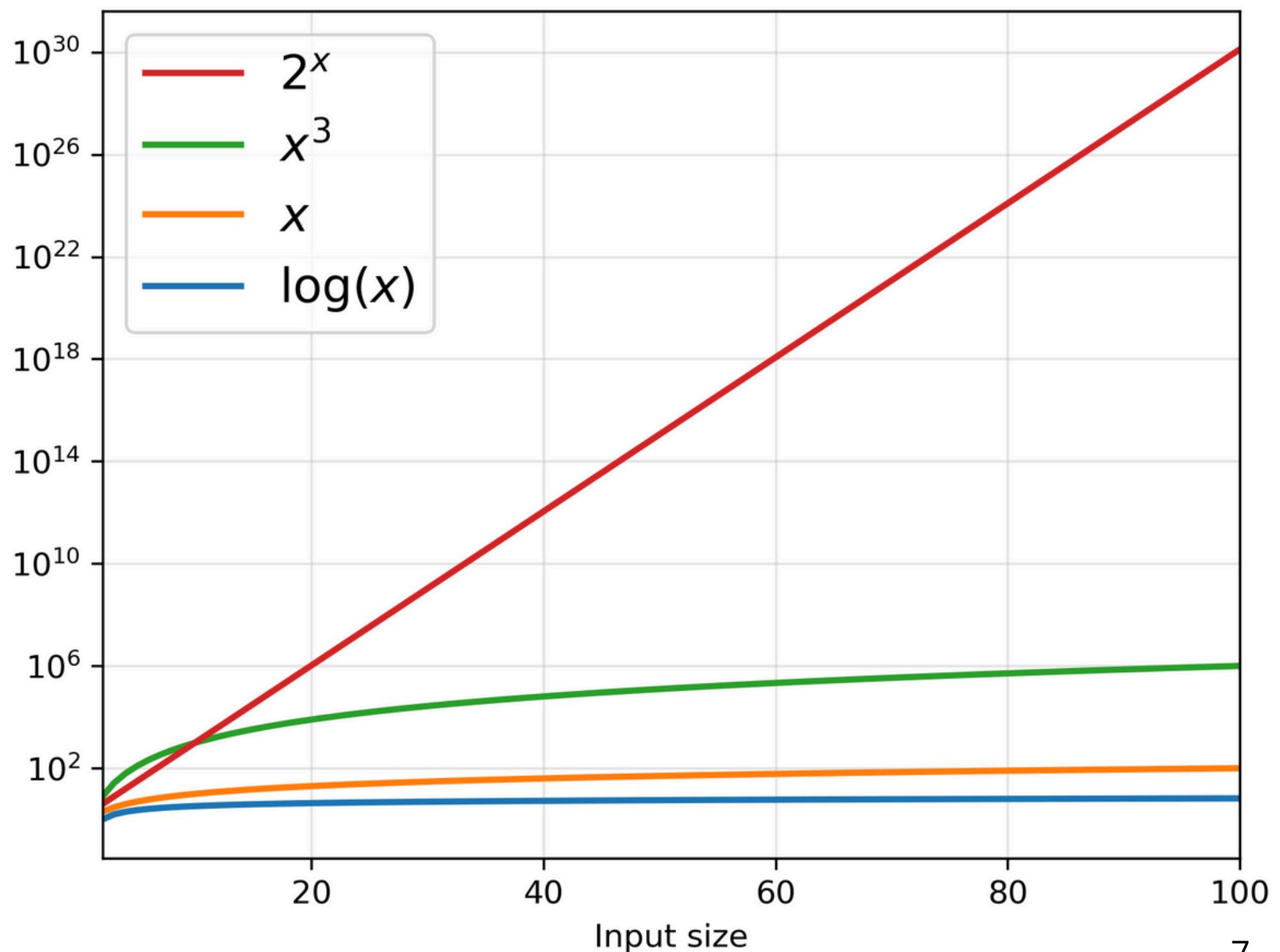
“La complejidad es $O(f(x))$ ”

Se lee:

“La complejidad es del orden de $f(x)$ ”.

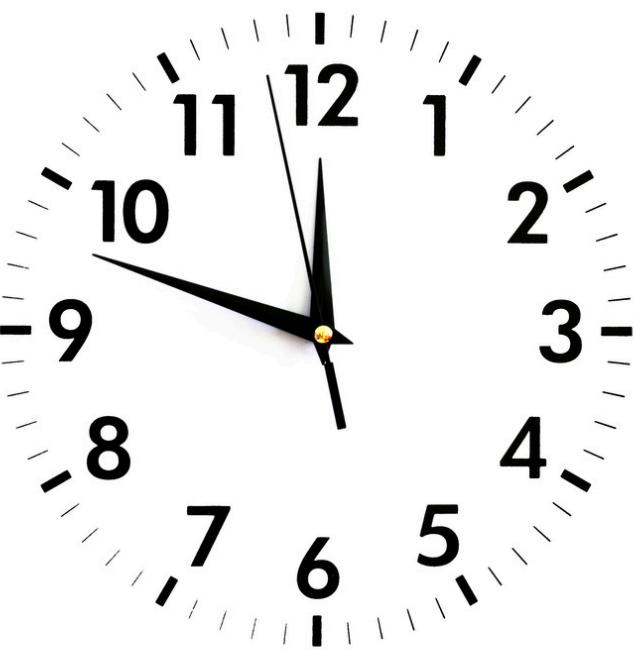
Significado:

La cantidad de operaciones que debe realizar un algoritmo es proporcional a $f(x)$ para valores de x asintóticamente grandes.



Aritmética modular

$$a \bmod N = b \Leftrightarrow a = b + kN, \text{ para algún } k \in \mathbb{Z}$$



En general, se calcula

$$k = \lfloor a/N \rfloor$$

$$a \bmod N = a - kN$$

$$17 \bmod 12 :$$

$$k = \lfloor 17/12 \rfloor = 1$$

$$17 \bmod 12 = 17 - 1 \cdot 12 = 5$$

El cálculo en ordenador tiene complejidad en tiempo $O(\log(N)\log(a))$

Máximo común divisor (mcd o gcd en inglés)

El mayor número entero que divide a otros dos de forma exacta. Se puede calcular conociendo la descomposición de ambos números, o con el método de Euclides (300 AC).

Iteración	a	b	
1	48	18	$48 \% 18 = 12$
2	18	12	$18 \% 12 = 6$
3	12	6	$12 \% 6 = 0$

```
def mcd(a, b):  
    # % es la operación módulo  
    while b > 0:  
        a, b = b, a % b  
    return a
```

Cantidad de iteraciones:
 $O(\log(\min(a, b)))$

Función totiente $\varphi(N)$

- Cuenta la cantidad de números menores que N, que son coprimos con N

$$\varphi(N) = |\{x < N : \text{mcd}(x, N) = 1\}|$$

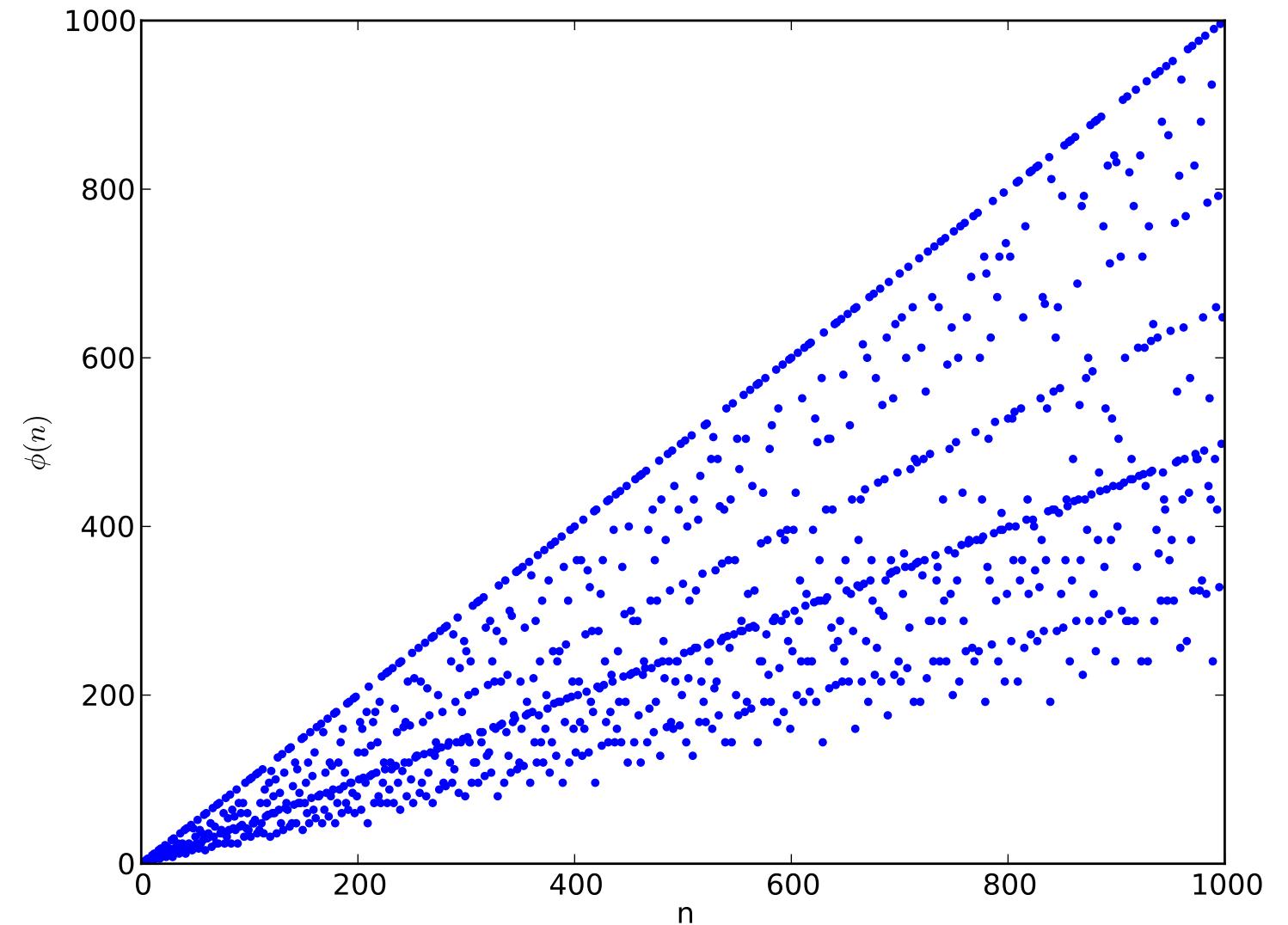
- Propiedades importantes:

$$\varphi(p) = p - 1, \quad \text{si } p \text{ es primo.}$$

$$\varphi(pq) = \varphi(p)\varphi(q)$$

- Teorema de Euler

“Si $\text{mcd}(M, N) = 1$, entonces $M^{\varphi(N)} \text{ mod}(N) = 1$.”

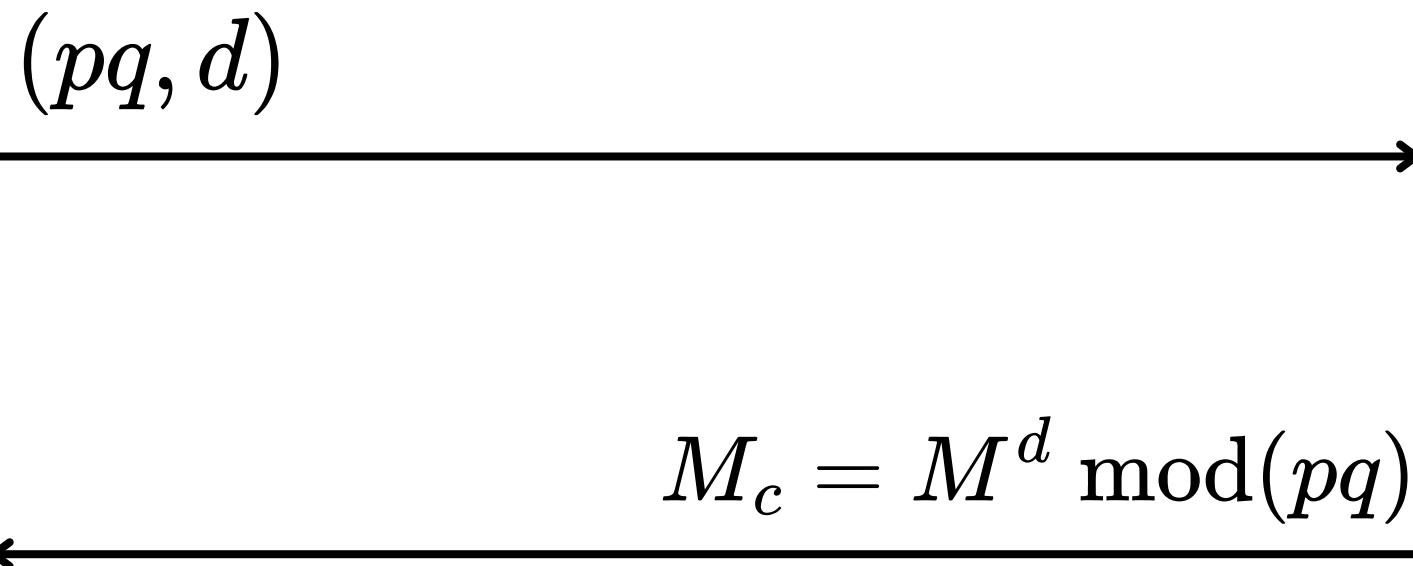


“La función totiente se calcula fácilmente sólo cuando se conocen los factores primos de su argumento”

Encriptación mensaje con RSA

Genera p, q, d, e

Publica (pq, d)

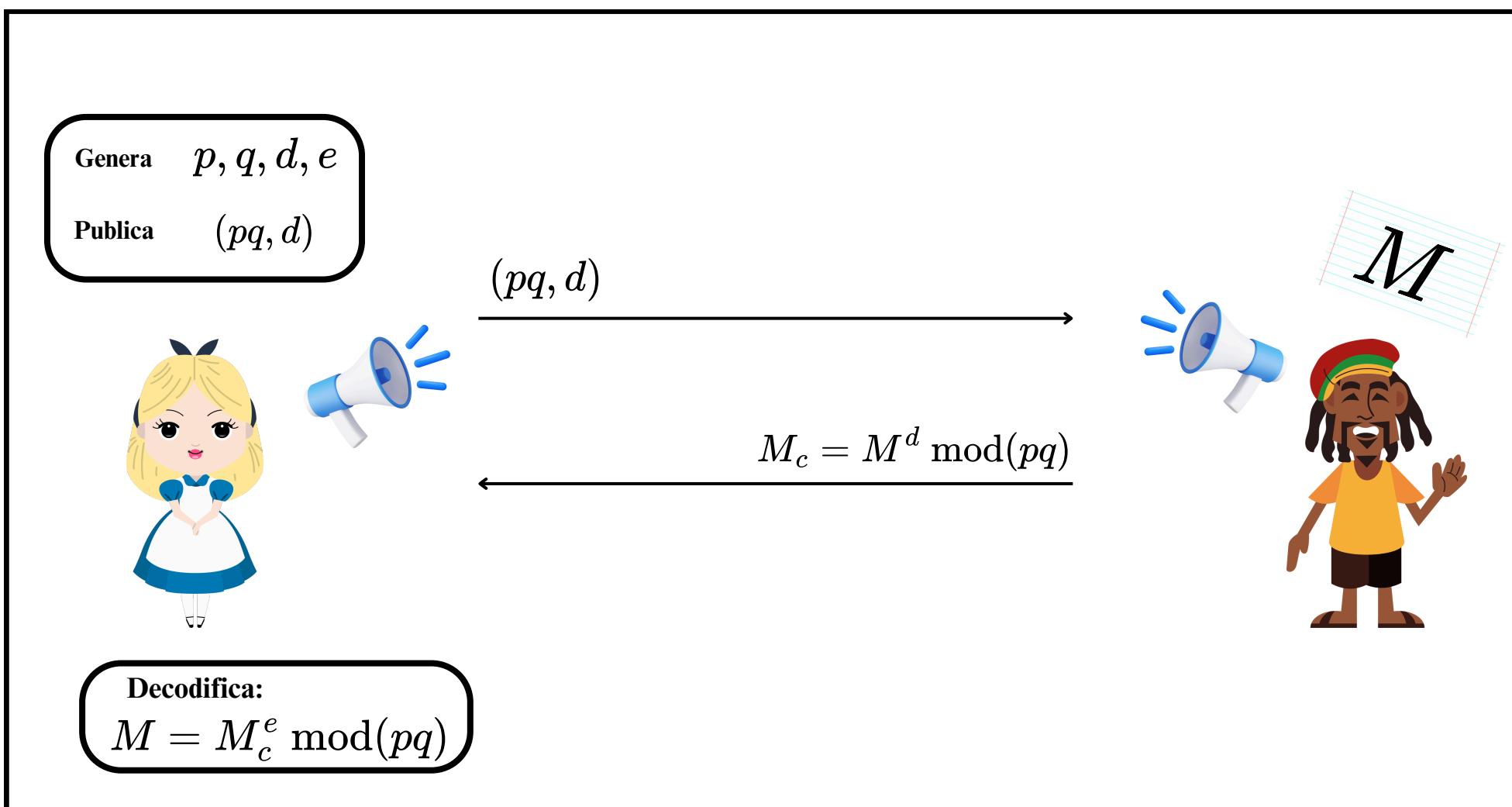


Decodifica:

$M' = M_c^e \text{ mod}(pq)$

Condiciones para p, q, d e

Buscamos $M' = M^{de} \bmod(pq) = M$



Condiciones para p, q, d e

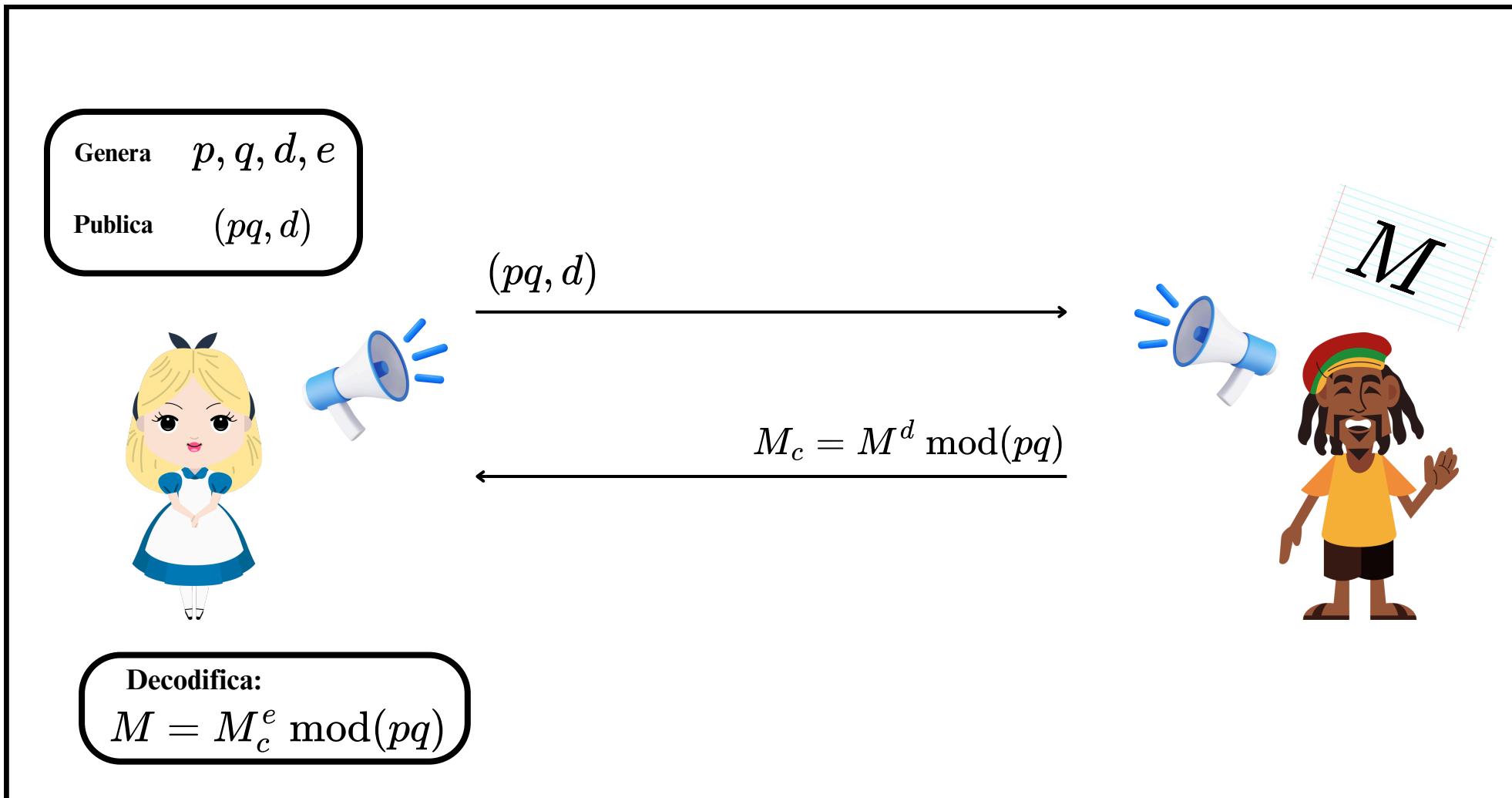
Buscamos $M' = M^{de} \bmod(pq) = M$

$$M^{\varphi(N)} \bmod(N) = 1$$

Teorema de Euler

Si $de = k\varphi(pq) + 1$, entonces $M^{de} \bmod pq = M(M^{\varphi(pq)})^k \bmod pq = M \bmod pq$

Notar que e existe cuando $\text{mcd}(d, \varphi(pq)) = 1$



Condiciones para p, q, d e

Buscamos $M' = M^{de} \text{ mod}(pq) = M$

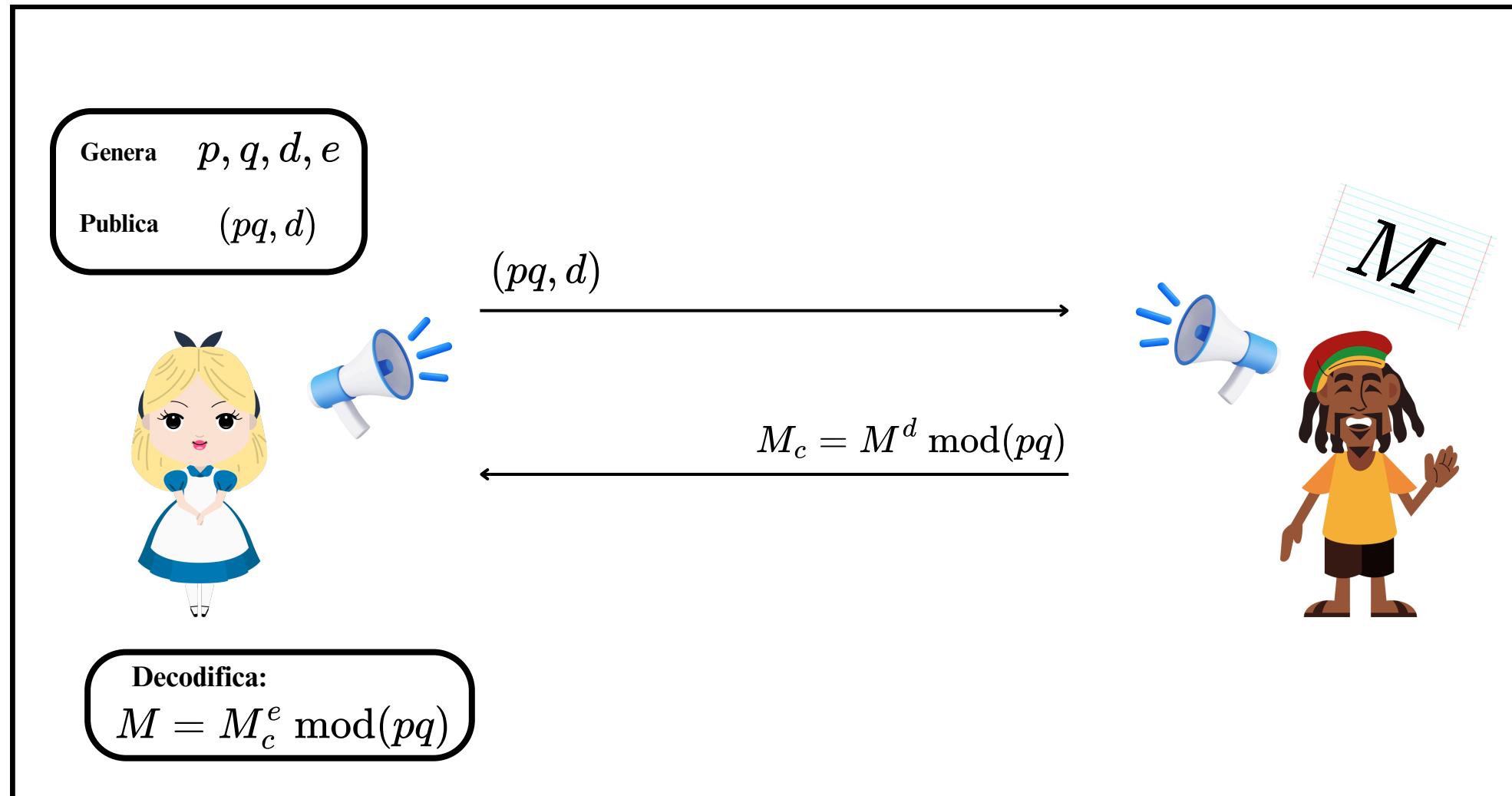
Si $de = k\varphi(pq) + 1$, entonces $M^{de} \text{ mod } pq = M(M^{\varphi(pq)})^k \text{ mod } pq = M \text{ mod } pq$

Notar que e existe cuando $\text{mcd}(d, \varphi(pq)) = 1$

$$M^{\varphi(N)} \text{ mod}(N) = 1$$

Teorema de Euler

El calculo de $\varphi(N)$ es eficiente cuando se conoce la descomposicion en numeros primos de N .



Cuando p y q son primos,
 $\varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1)$

Condiciones para p, q, d e

Buscamos $M' = M^{de} \text{ mod}(pq) = M$

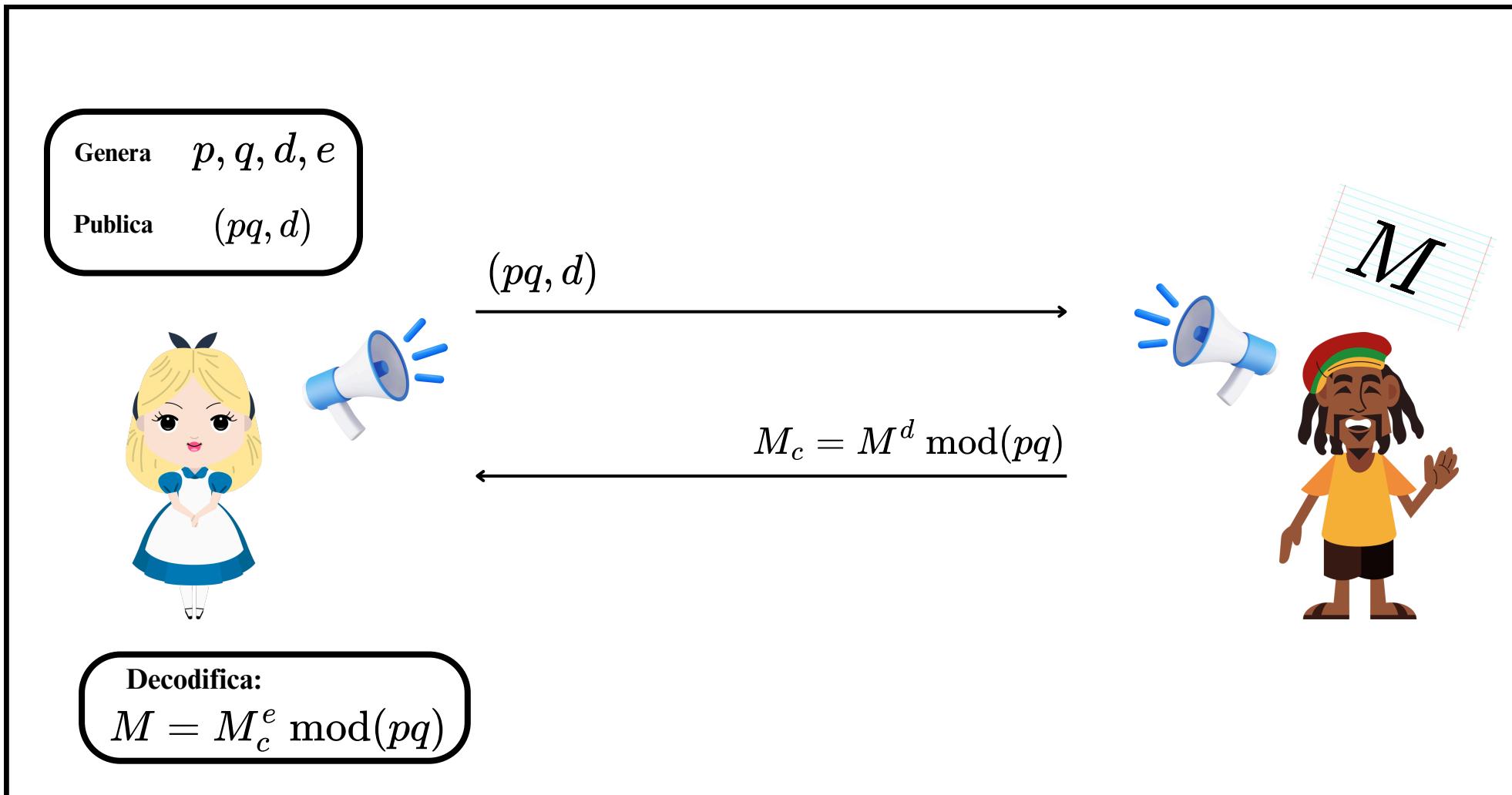
Si $de = k\varphi(pq) + 1$, entonces $M^{de} \text{ mod } pq = M(M^{\varphi(pq)})^k \text{ mod } pq = M \text{ mod } pq$

Notar que e existe cuando $\text{mcd}(d, \varphi(pq)) = 1$

$$M^{\varphi(N)} \text{ mod}(N) = 1$$

Teorema de Euler

El calculo de $\varphi(N)$ es eficiente cuando se conoce la descomposicion en numeros primos de N .



Cuando p y q son primos,
 $\varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1)$

$M < pq$
 $\text{mcd}(M, pq) = 1$
 p, q primos
 $\text{mcd}(d, (p - 1)(q - 1)) = 1$
 $ed \text{ mod}((p - 1)(q - 1)) = 1$

Encriptación mensaje con RSA

Genera p, q, d, e

Publica (pq, d)



(pq, d)



$M_c = M^d \text{ mod}(pq)$



M

Decodifica:

$M' = M_c^e \text{ mod}(pq)$

p, q primos

$\text{mcd}(d, (p - 1)(q - 1)) = 1$

$ed \text{ mod}((p - 1)(q - 1)) = 1$

Creación de llaves RSA

(Criba de Erastótenes, $O(N)$), (test de primalidad $O(\text{poly}(\log(N)))$)

- 1) Se fija la cantidad de bits para $N = pq$. Se eligen p y q tales que $\text{nbits}(p) + \text{nbits}(q) = \text{nbits}(N)$
(Es mas seguro cuando p y q son aleatorios, grandes y tienen una diferencia grande entre ellos)
- 2) Se calcula $\varphi(pq) = (p - 1)(q - 1)$
- 3) Se escoge aleatoriamente $1 < e < \varphi(pq)$, $\text{mcd}(e, \varphi(pq)) = 1$ (Algoritmo de Euclides, $O(\log(N)^3)$)
- 4) Se calcula $d: ed \bmod(\varphi(pq)) = 1$ (Algoritmo extendido de Euclides, $O(\log(N)^3)$)
- 5) Se publica la llave (N, e)
- 6) Se guarda como llave privada (p, q, d)

Firma mensaje con RSA

Genera p, q, d, e

Publica (pq, d)



(F, M)

$H = \text{HASH}(M)$

$H = \text{HASH}(M)$

$F = H^e \bmod(pq)$



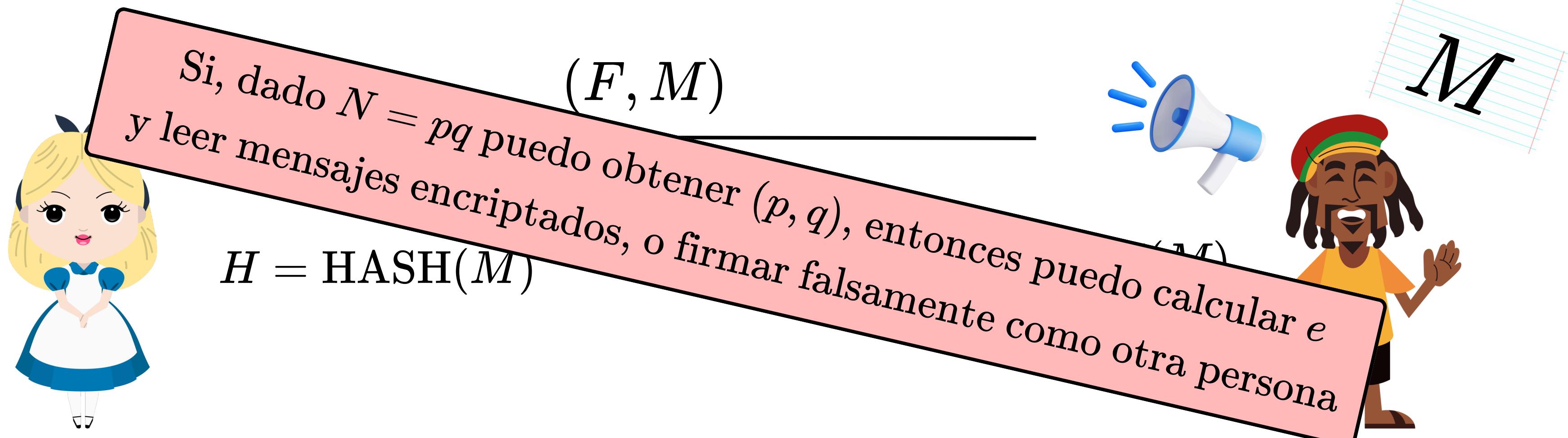
M

“Si $H = F^d \bmod(pq)$,
sé que este mensaje es de Bob”

Firma mensaje con RSA

Genera p, q, d, e

Publica (pq, d)



“Si $H = F^d \bmod(pq)$,
sé que este mensaje es de Bob”



Checkpoint



Puntos clave

- La criptografía simétrica requiere distribuir una clave de forma segura
- La criptografía asimétrica se basa en problemas sin solución eficiente
- RSA permite encriptar mensajes para que sólo una persona pueda leerlos
- RSA permite también firmar un mensaje para verificar su emisor
- El procedimiento para llevar a cabo RSA es eficiente, $O(\log(N)^3)$
- Si podemos separar un número en sus factores primos, rompemos RSA

General Number Field Sieve

$$\text{Complejidad } O \left(\exp \left[1.923 (\log N)^{1/3} (\log \log N)^{2/3} \right] \right)$$

El algoritmo conocido mas eficiente para factorizar números grandes clásicamente

Si $N \sim 2^{2048}$, la complejidad es del orden de 10^{35} operaciones

Un supercomputador puede alcanzar las 10^{18} operaciones por segundo

Tiempo aproximado: 10^9 años (casi del orden de la edad del universo)

Algoritmo de Shor

Complejidad $O(\log N)$ en el espacio, $O((\log N)^3)$ en tiempo.

Mejor escalamiento en tiempo: $O((\log N)^2 \log \log N)$ [1]

2048 bits corresponde a $N \sim 2^{2048} \Rightarrow (\log N)^2 \log \log N = 1.46 \cdot 10^7$

Computadores superconductores ~ 100 [ns/operación]

$$\text{Tiempo de ejecución} \sim \frac{(\log N)^2 \log \log N [\text{operaciones}]}{10^7 [\text{operaciones/s}]} = 1.46 [s]$$

[1] Harvey, D. and van der Hoeven, J. (2021), Annals of Mathematics. 193 (2)

Factorización á la Shor (1994)

Sea N el producto de 2 primos y $1 < a < N$
un número aleatorio tal que $\text{mcd}(a, N) = 1$

Sea también la función periódica

$f(x) = a^x \text{mod}(N)$, con periodo r

Luego, $a^r \text{mod}(N) = 1 \Leftrightarrow (a^r - 1)\text{mod}(N) = 0$
 $\Leftrightarrow (a^{r/2} + 1)(a^{r/2} - 1) = kN, \quad k \in \mathbb{N}$

Podemos aislar los factores de $N = pq$

$$p = \text{mcd}(a^{r/2} + 1, N)$$

$$q = \text{mcd}(a^{r/2} - 1, N)$$

```
def shor(N: int) -> list[int]:
    a = random.randint(2, N-2)

    # Ensure a and N are coprimes
    f1 = math.gcd(a, N)
    if f1 > 1:
        return [f1, N//f1]

    # Here is where the magic happens!
    r = find_period(a, N) (QPE)

    if r%2 == 1: # r is odd
        return shor(N)
    else:          # r is even
        ar2 = pow(a, r//2)
        p, q = ar2 + 1, ar2 - 1

        if p%N == 0: # This case would return [1, N]
            return shor(N)

    return [math.gcd(p, N), math.gcd(q, N)]
```

Requerimientos para búsqueda de periodo

- Transformada cuántica de Fourier (QFT)
- Estimación cuántica de fase (QPE)

Quantum Fourier Transform (o cómo calcular frecuencias/periodos)

Sea $\mathcal{B} = \{|x\rangle\}_{x=0}^{d-1}$ una base

ortonormal de dimensión d

$$|\psi\rangle = \sum_{x=0}^{d-1} c_x |x\rangle$$

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle$$

Quantum Fourier Transform (o cómo calcular frecuencias/periodos)

Sea $\mathcal{B} = \{|x\rangle\}_{x=0}^{d-1}$ una base
ortonormal de dimensión d

$$|\psi\rangle = \sum_{x=0}^{d-1} c_x |x\rangle$$

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle$$

$$\text{QFT}^{-1}|x\rangle = \text{QFT}^\dagger|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{-\frac{2\pi i}{d} xy} |y\rangle$$

Quantum Fourier Transform (o cómo calcular frecuencias/periodos)

Sea $\mathcal{B} = \{|x\rangle\}_{x=0}^{d-1}$ una base ortonormal de dimensión d

$$|\psi\rangle = \sum_{x=0}^{d-1} c_x |x\rangle$$

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle$$

$$\begin{aligned}\text{QFT}|\psi\rangle &= \sum_{x=0}^{d-1} c_x \text{QFT}|x\rangle \\ &= \sum_{x=0}^{d-1} c_x \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle \\ &= \sum_{y=0}^{d-1} \left[\frac{1}{\sqrt{d}} \sum_{x=0}^{d-1} e^{\frac{2\pi i}{d} xy} c_x \right] |y\rangle\end{aligned}$$

$$\text{QFT}^{-1}|x\rangle = \text{QFT}^\dagger|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{-\frac{2\pi i}{d} xy} |y\rangle$$

Quantum Fourier Transform

(o cómo calcular frecuencias/periodos)

Sea $\mathcal{B} = \{|x\rangle\}_{x=0}^{d-1}$ una base ortonormal de dimensión d

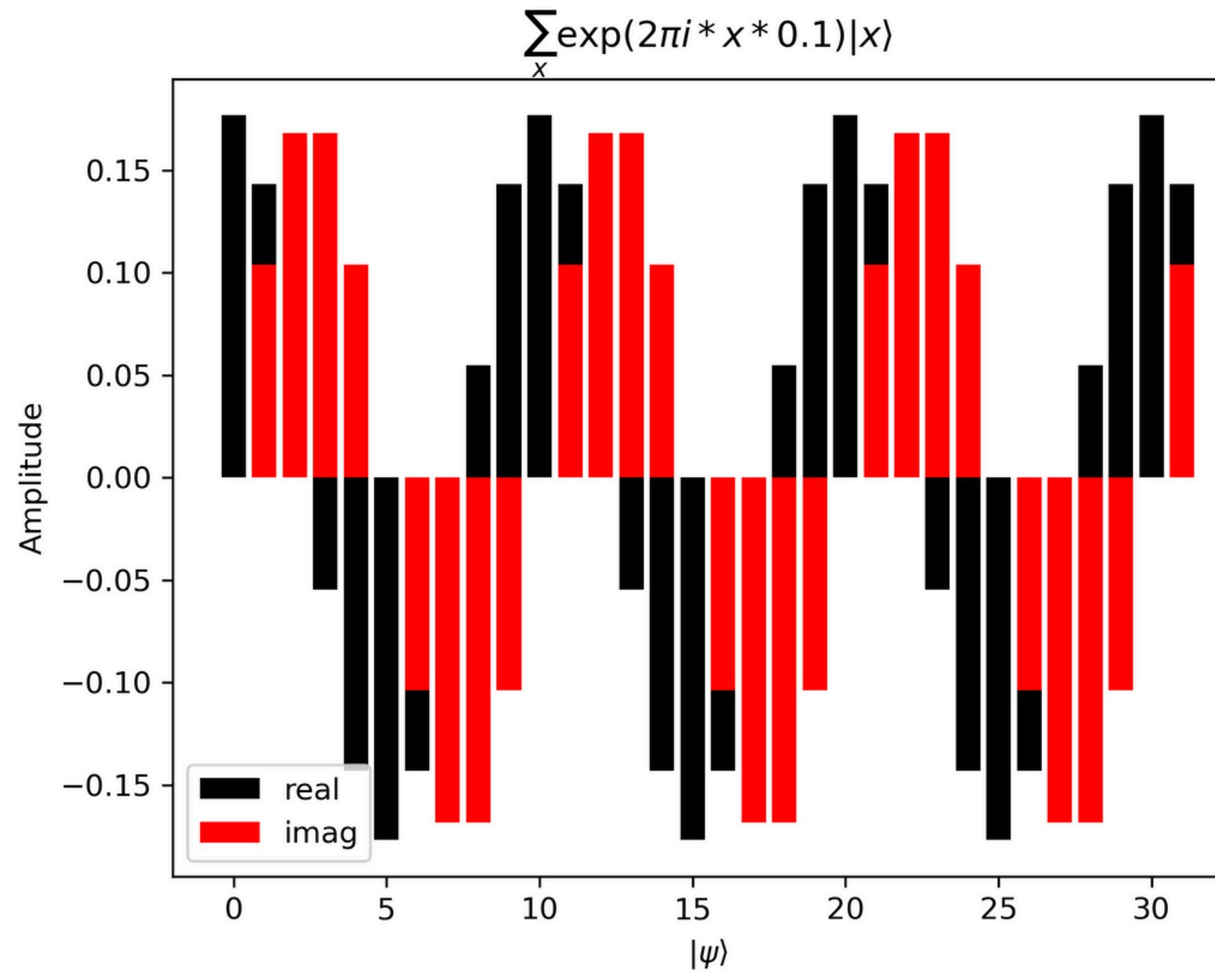
$$|\psi\rangle = \sum_{x=0}^{d-1} c_x |x\rangle$$

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle$$

$$\text{QFT}^{-1}|x\rangle = \text{QFT}^\dagger|x\rangle = \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{-\frac{2\pi i}{d} xy} |y\rangle$$

$$\begin{aligned} \text{QFT}|\psi\rangle &= \sum_{x=0}^{d-1} c_x \text{QFT}|x\rangle \\ &= \sum_{x=0}^{d-1} c_x \frac{1}{\sqrt{d}} \sum_{y=0}^{d-1} e^{\frac{2\pi i}{d} xy} |y\rangle \\ &= \sum_{y=0}^{d-1} \left[\frac{1}{\sqrt{d}} \sum_{x=0}^{d-1} e^{\frac{2\pi i}{d} xy} c_x \right] |y\rangle \end{aligned}$$

IDFT($\{c_x\}$)



```

def circuit_preparation():
    qc = QuantumCircuit(num_qubits + 1, num_qubits)

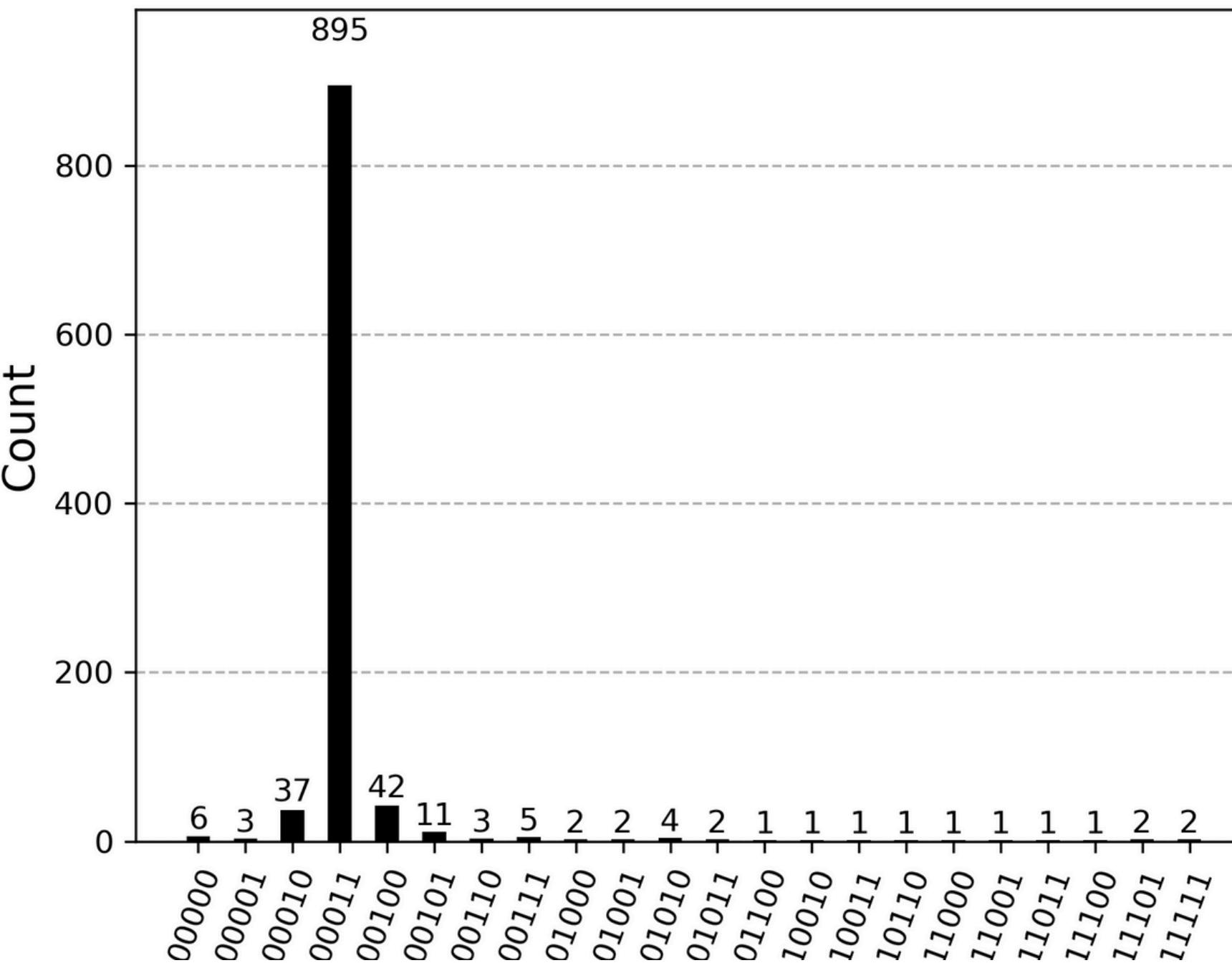
    # (...) THIS PART IS SECRET (...)

    return qc

# Simulate the statevector
state_simulator = Aer.get_backend("statevector_simulator")
compiled_circuit = transpile(circuit_preparation(), state_simulator)
state_result = state_simulator.run(compiled_circuit).result()
statevector = state_result.get_statevector()

# Print the statevector components
plt.bar(range(32), statevector.real[32:], color='black', label='real')
plt.bar(range(32), statevector.imag[32:], color='red', label = 'imag')
plt.xlabel(r'$ |\psi\rangle $')
plt.ylabel('Amplitude')
plt.title(r'$ \sum_x \exp(2\pi i * x * 0.1) |x\rangle $')
plt.legend(loc='best')
plt.savefig('amplitudes.png', dpi=300)
plt.show()

```



```
def circuit_iqft():
    qc = circuit_preparation() # Prepare state

    # Apply the final inverse QFT and measure
    qc.compose(QFT(num_qubits, inverse=True), range(num_qubits),
               inplace=True)
    qc.measure(range(num_qubits), range(num_qubits))

    return qc

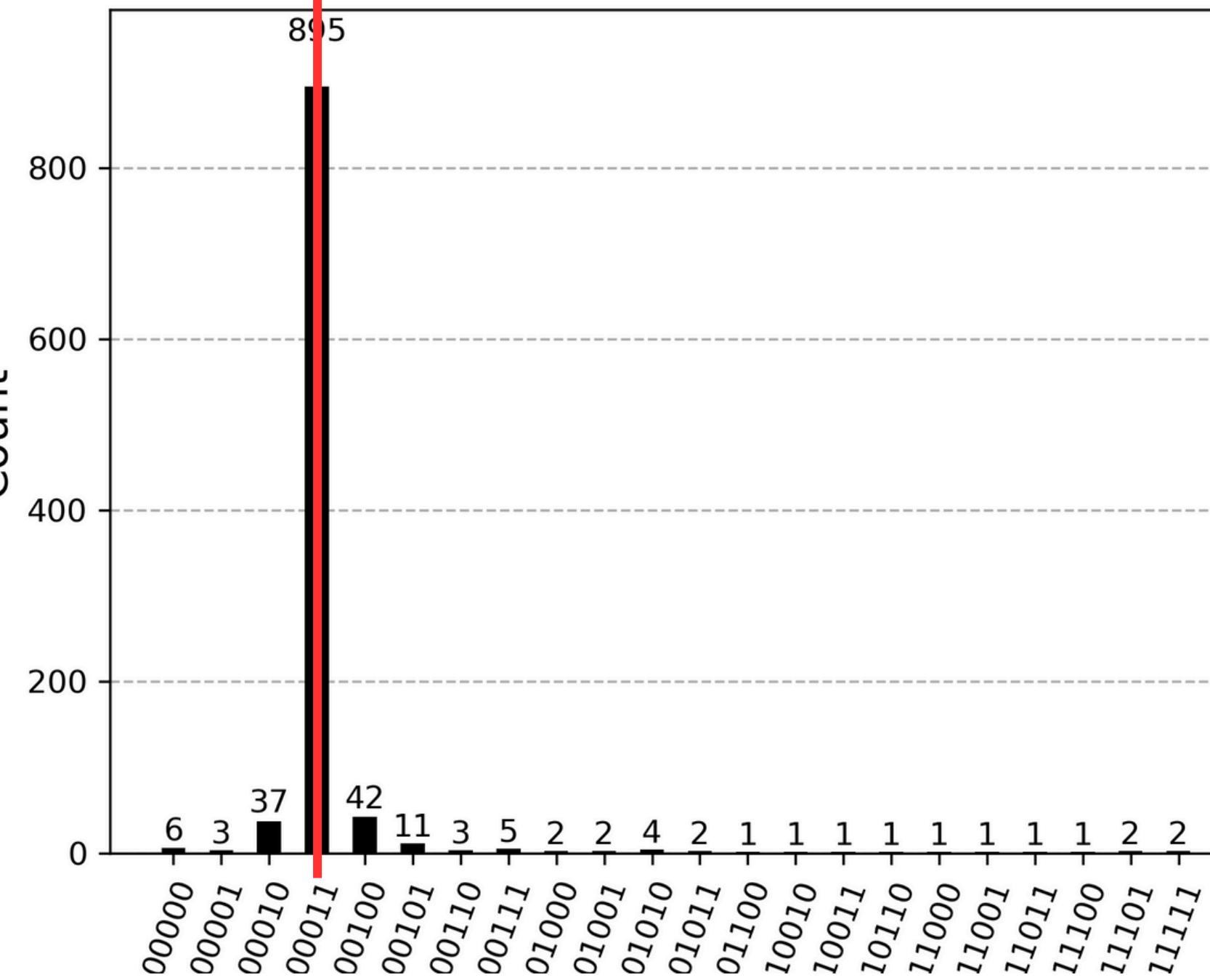
# Simulate measurements
simulator = Aer.get_backend("qasm_simulator")
compiled_circuit = transpile(circuit_iqft(), state_simulator)
result = simulator.run(compiled_circuit, shots=1024).result()
counts = result.get_counts()

# Plot
plot_histogram(counts, color='black')
plt.savefig('counts.png', dpi=300)
plt.show()
```

$$f = 3/2^5 = 0.09375$$

$$\Delta f = 1/2^5 = 0.03125$$

$$T = 1/f = 10.6$$



```

def circuit_iqft():
    qc = circuit_preparation() # Prepare state

    # Apply the final inverse QFT and measure
    qc.compose(QFT(num_qubits, inverse=True), range(num_qubits),
               inplace=True)
    qc.measure(range(num_qubits), range(num_qubits))

    return qc

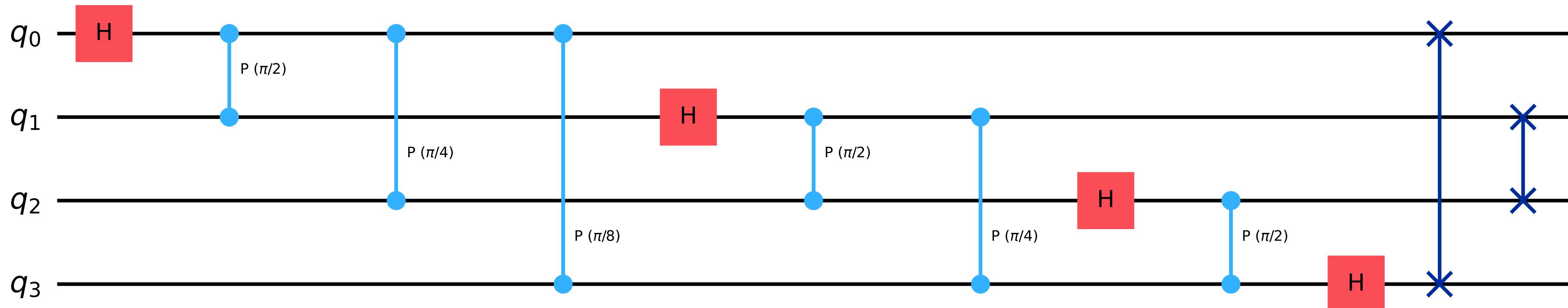
# Simulate measurements
simulator = Aer.get_backend("qasm_simulator")
compiled_circuit = transpile(circuit_iqft(), state_simulator)
result = simulator.run(compiled_circuit, shots=1024).result()
counts = result.get_counts()

# Plot
plot_histogram(counts, color='black')
plt.savefig('counts.png', dpi=300)
plt.show()

```

Implementación QFT

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{2^n}} \bigotimes_{k=1}^n \left(|0\rangle + \exp(2\pi i x / 2^k) |1\rangle \right)$$



Son n Hadamard, $n(n - 1)/2$ fases controladas y $n/2$ CNOT. En total: $O(n^2)$ 33



Checkpoint



Puntos clave

- Factorizar en un ordenador clásico es muy ineficiente
- Se puede reducir el problema de factorización a una búsqueda de periodo, que Shor resuelve mediante un computador cuántico
- La transformada de Fourier mapea amplitudes de funciones en frecuencias/periodos
- La transformada cuántica de Fourier se puede llevar a cabo en un circuito cuántico, de forma exponencialmente mas eficiente que la DFT clásica, $O(n^2)$ vs $O(n2^n)$
- La transformada cuántica de Fourier tiene resolución en frecuencias/fases de $1/2^n$

Quantum Phase Estimation

Dado un operador unitario U y un autovector $|\psi\rangle$ tales que $U|\psi\rangle = \exp(2\pi i\phi)|\psi\rangle$, encuentra el valor de ϕ .

2 etapas

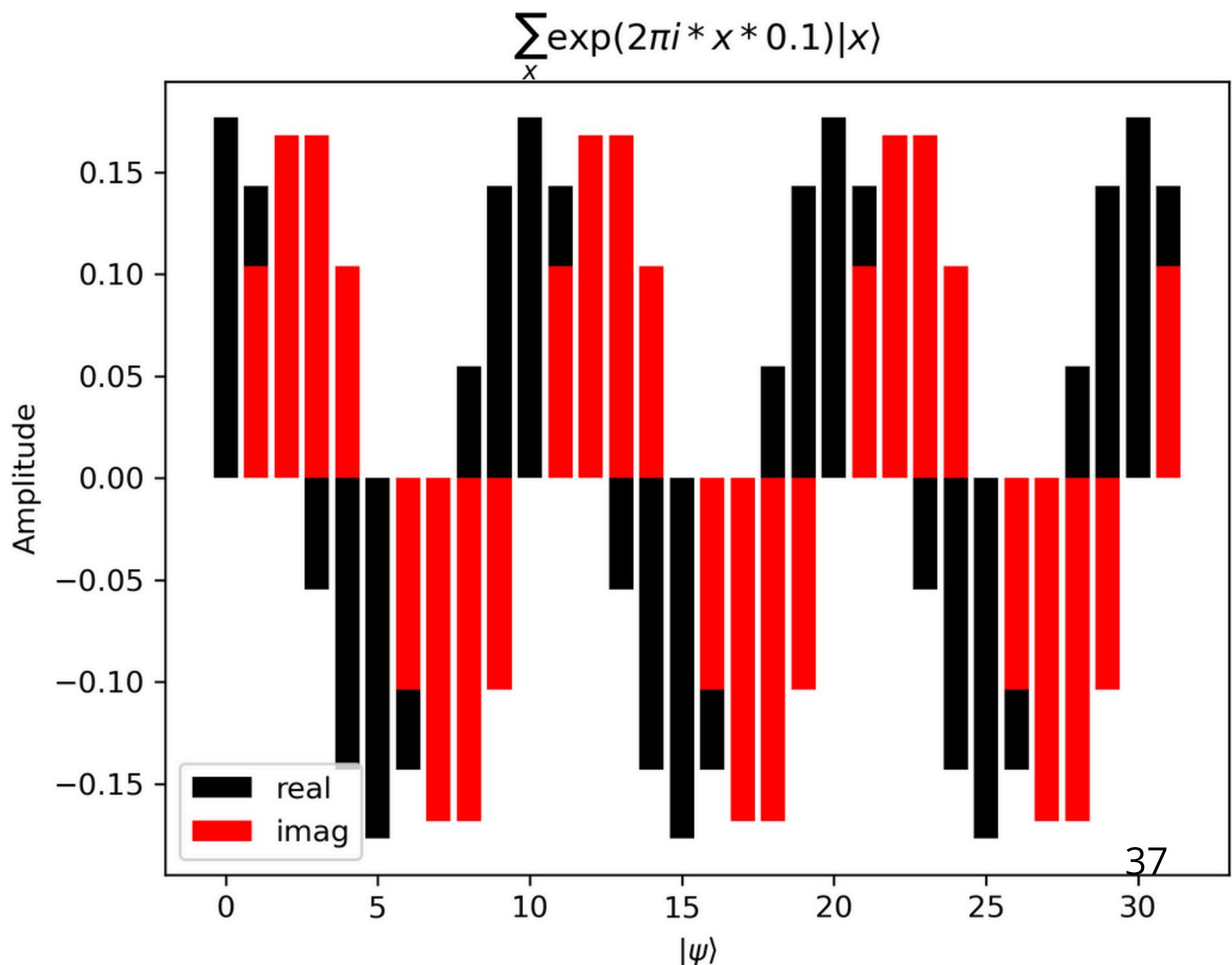
- Formar el estado $\sum_x^{2^n-1} \exp(2\pi i x \phi) |x\rangle$
- Extraer la fase usando QFT y midiendo

Quantum Phase Estimation

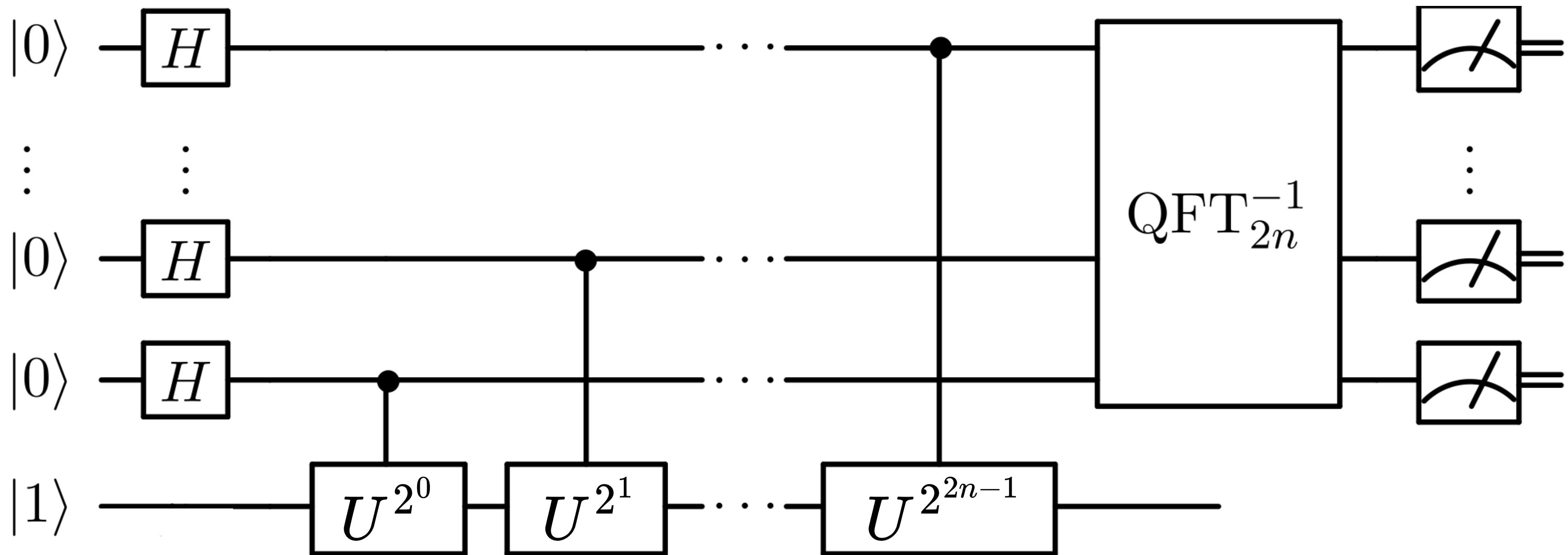
Dado un operador unitario U y un autovector $|\psi\rangle$ tales que $U|\psi\rangle = \exp(2\pi i\phi)|\psi\rangle$, encuentra el valor de ϕ .

2 etapas

- Formar el estado $\sum_x \exp(2\pi i x \phi) |x\rangle$
- Extraer la fase usando QFT y midiendo



Quantum Phase Estimation

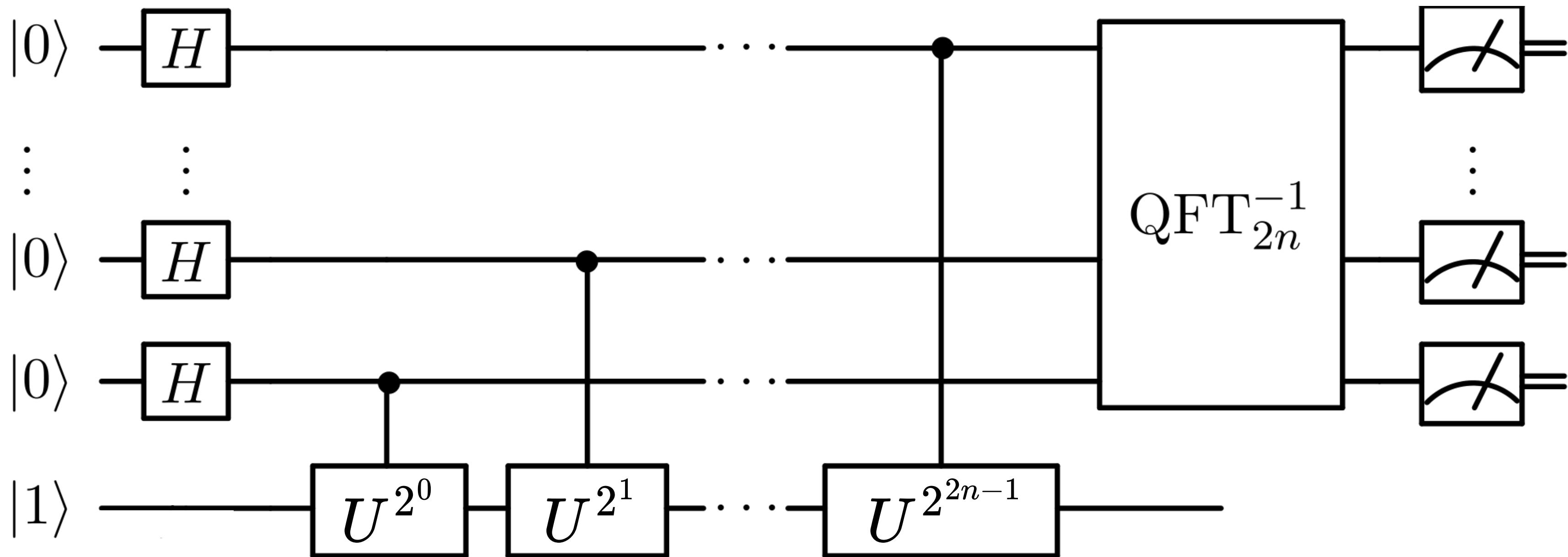


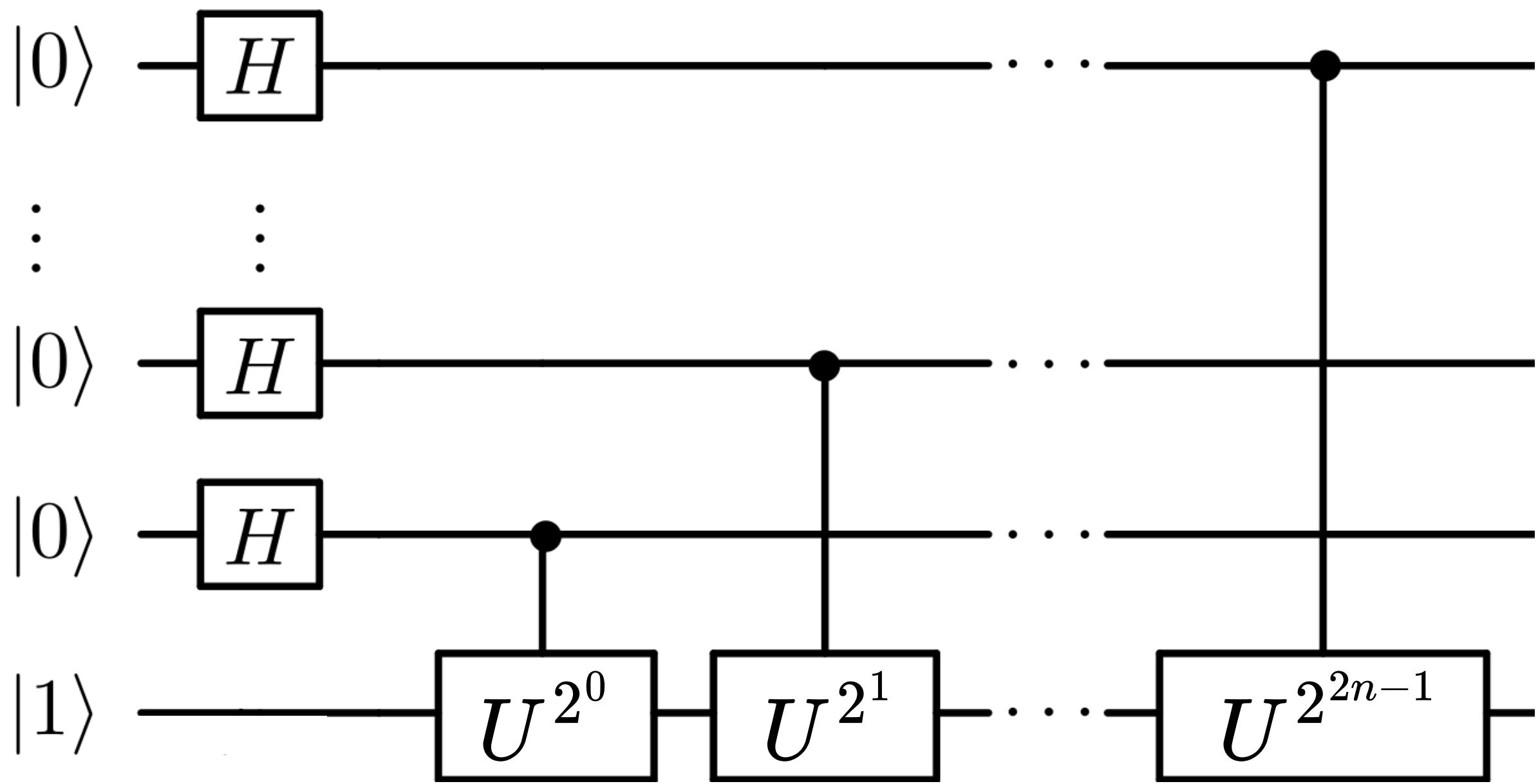
Quantum Phase Estimation

$$U = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i \phi} \end{pmatrix}$$

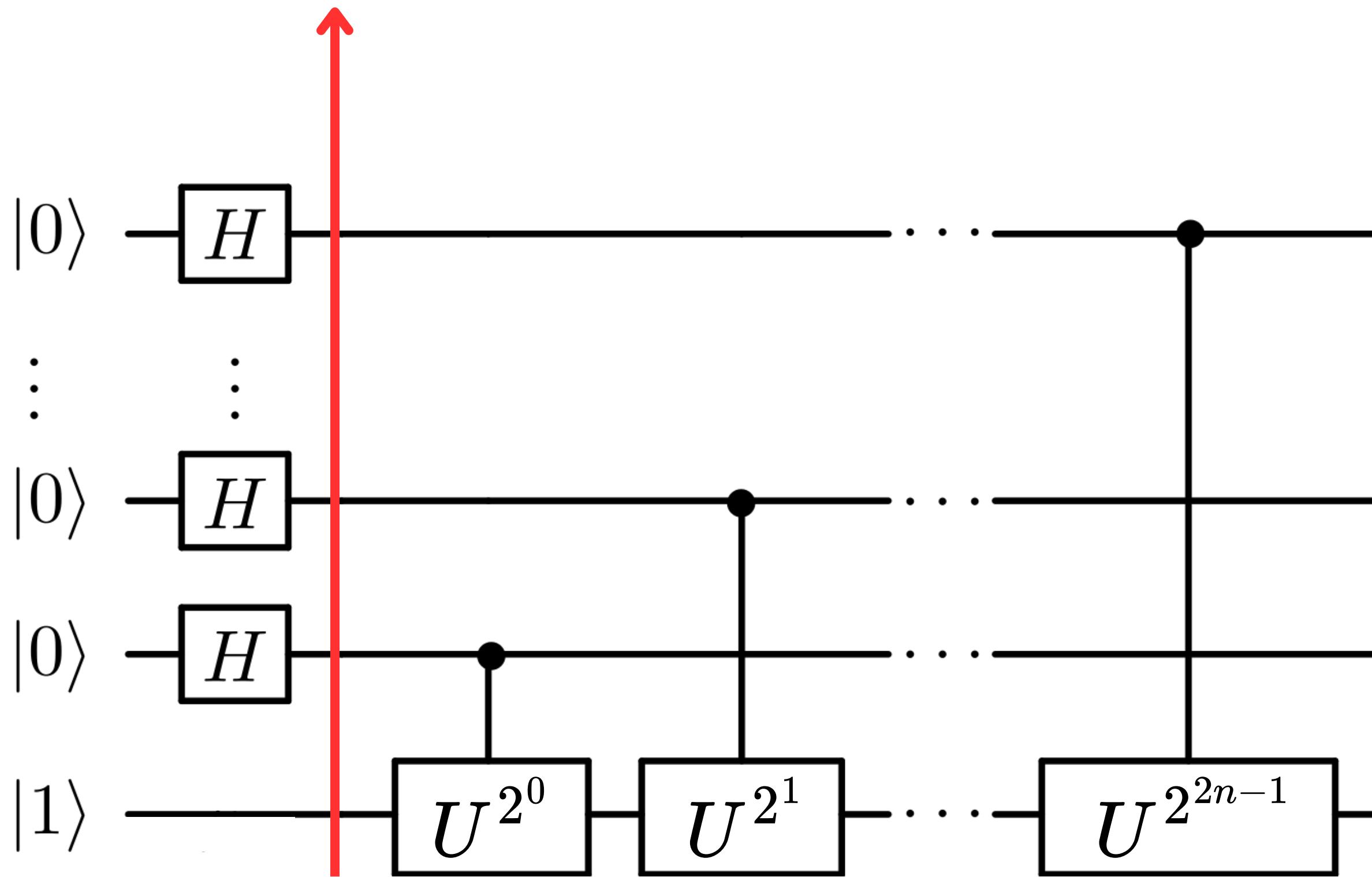
$$U|0\rangle = |0\rangle$$

$$U|1\rangle = \exp(2\pi i \phi)|1\rangle$$

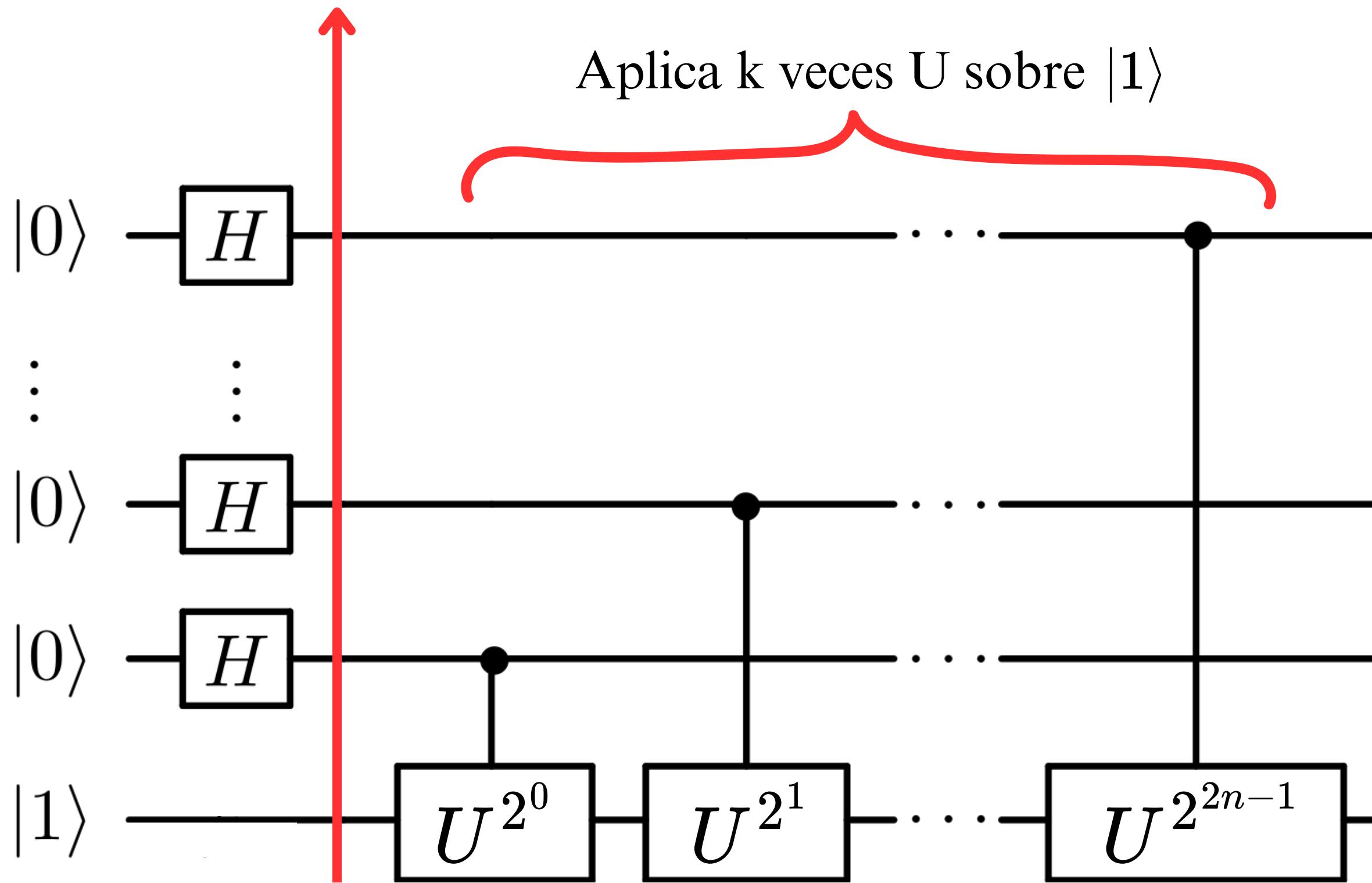




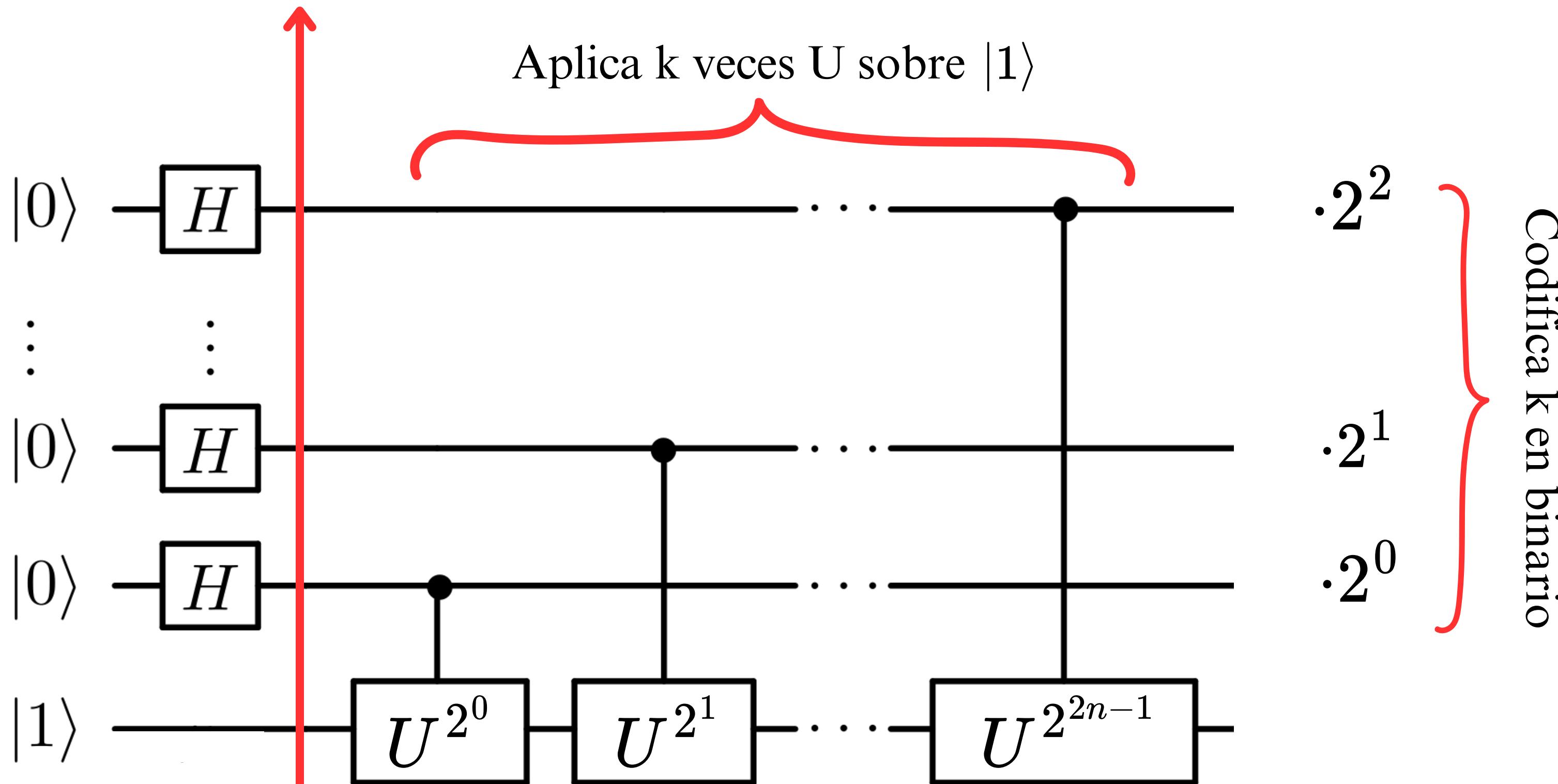
$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle|1\rangle$$



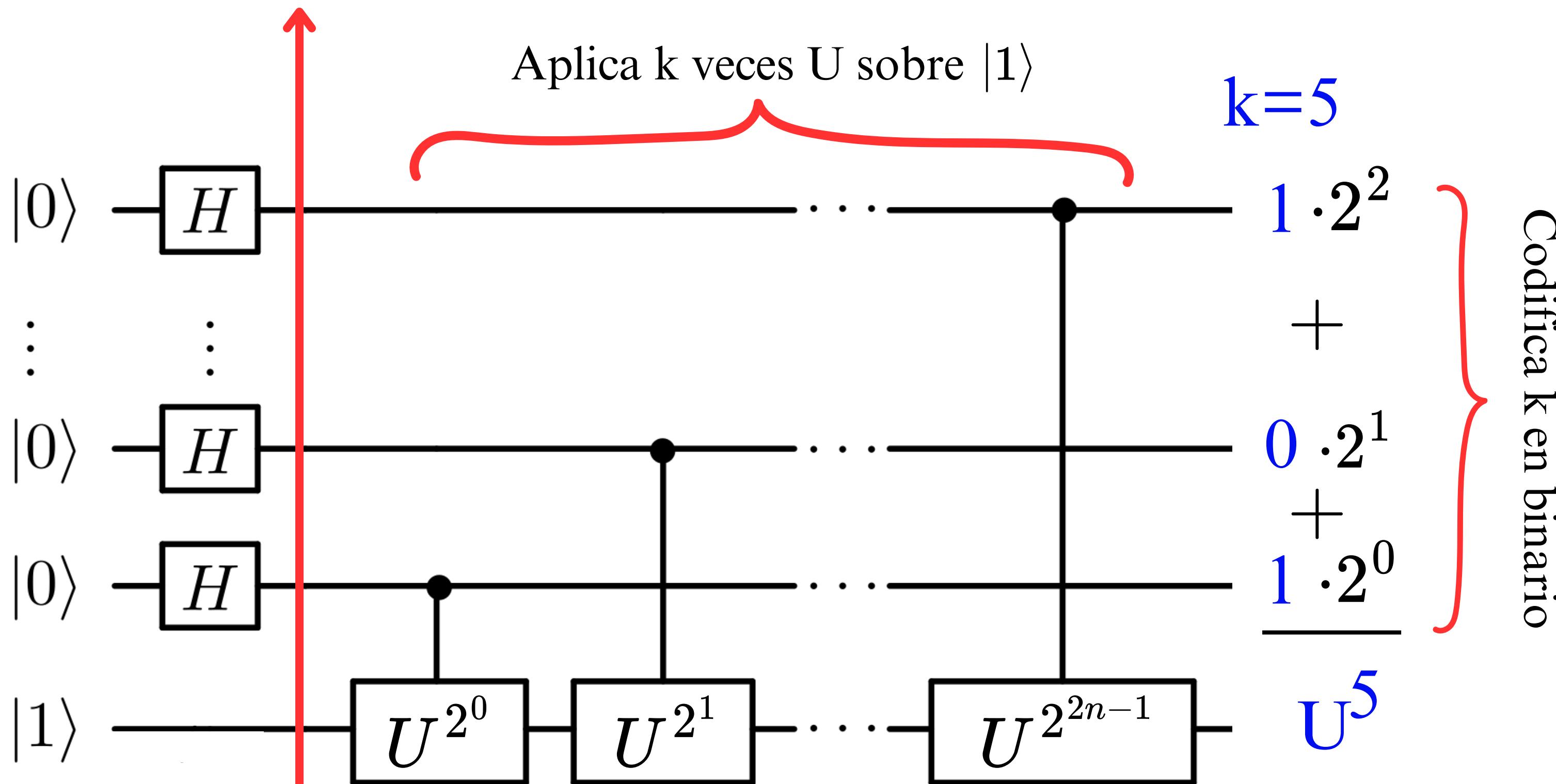
$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle|1\rangle$$



$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle|1\rangle$$

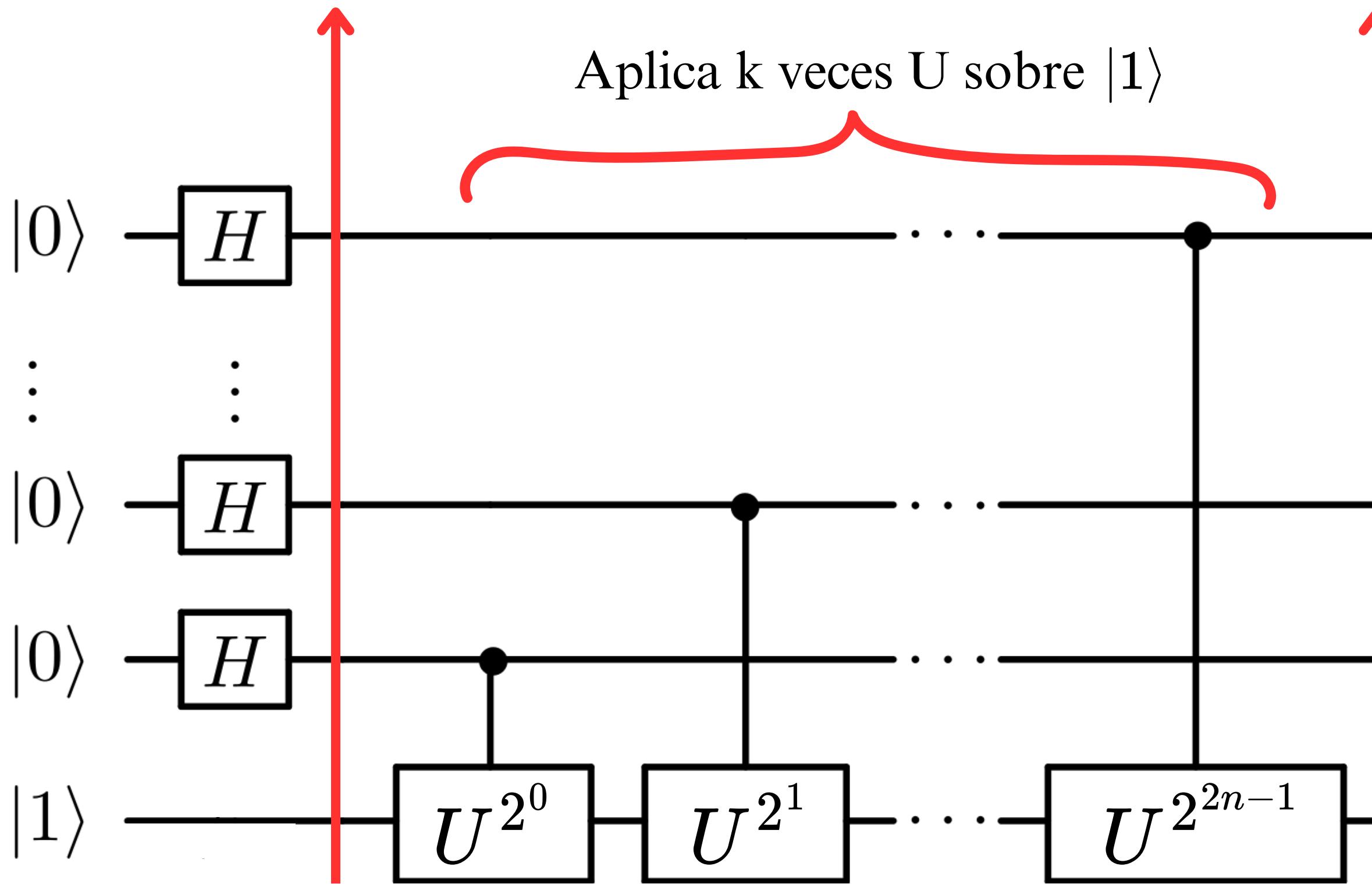


$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle|1\rangle$$

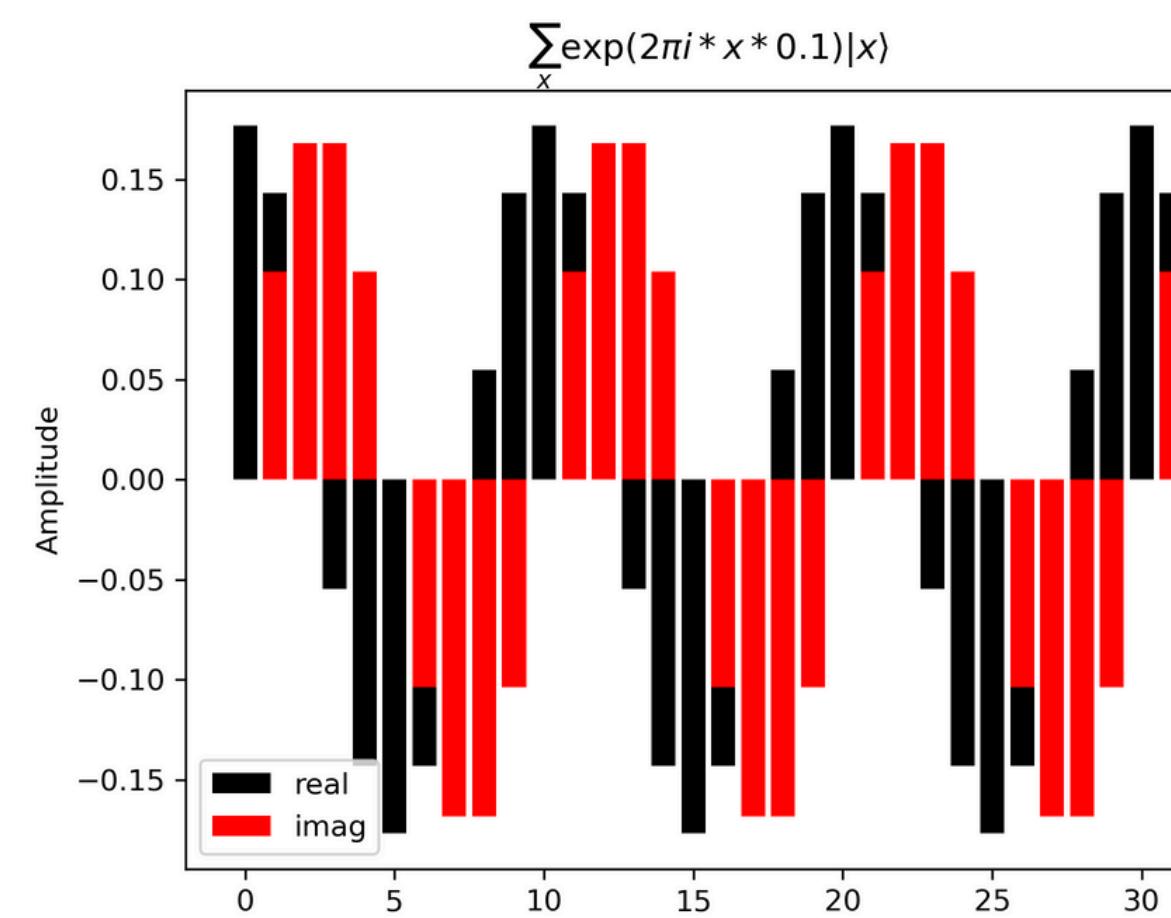
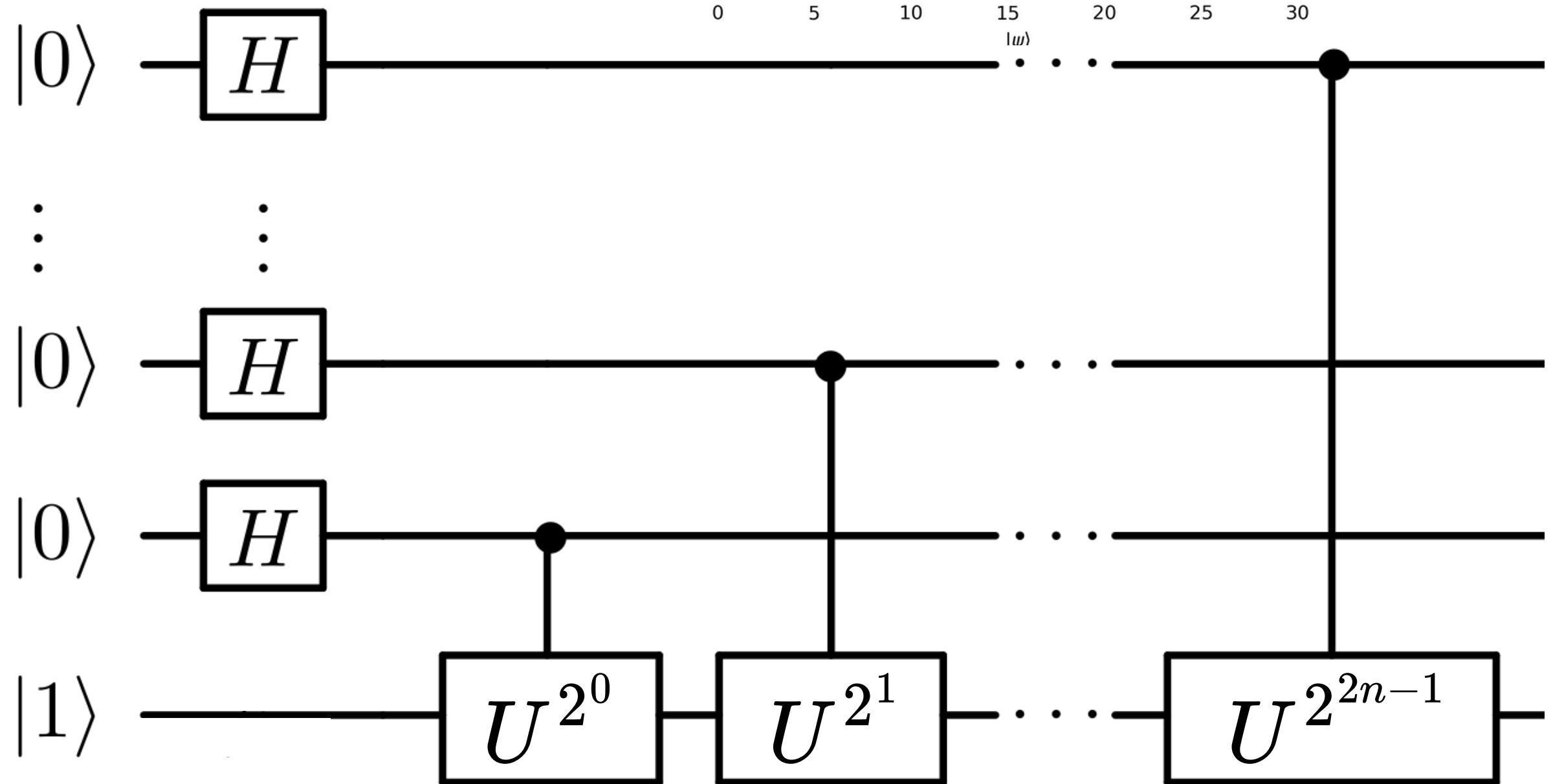


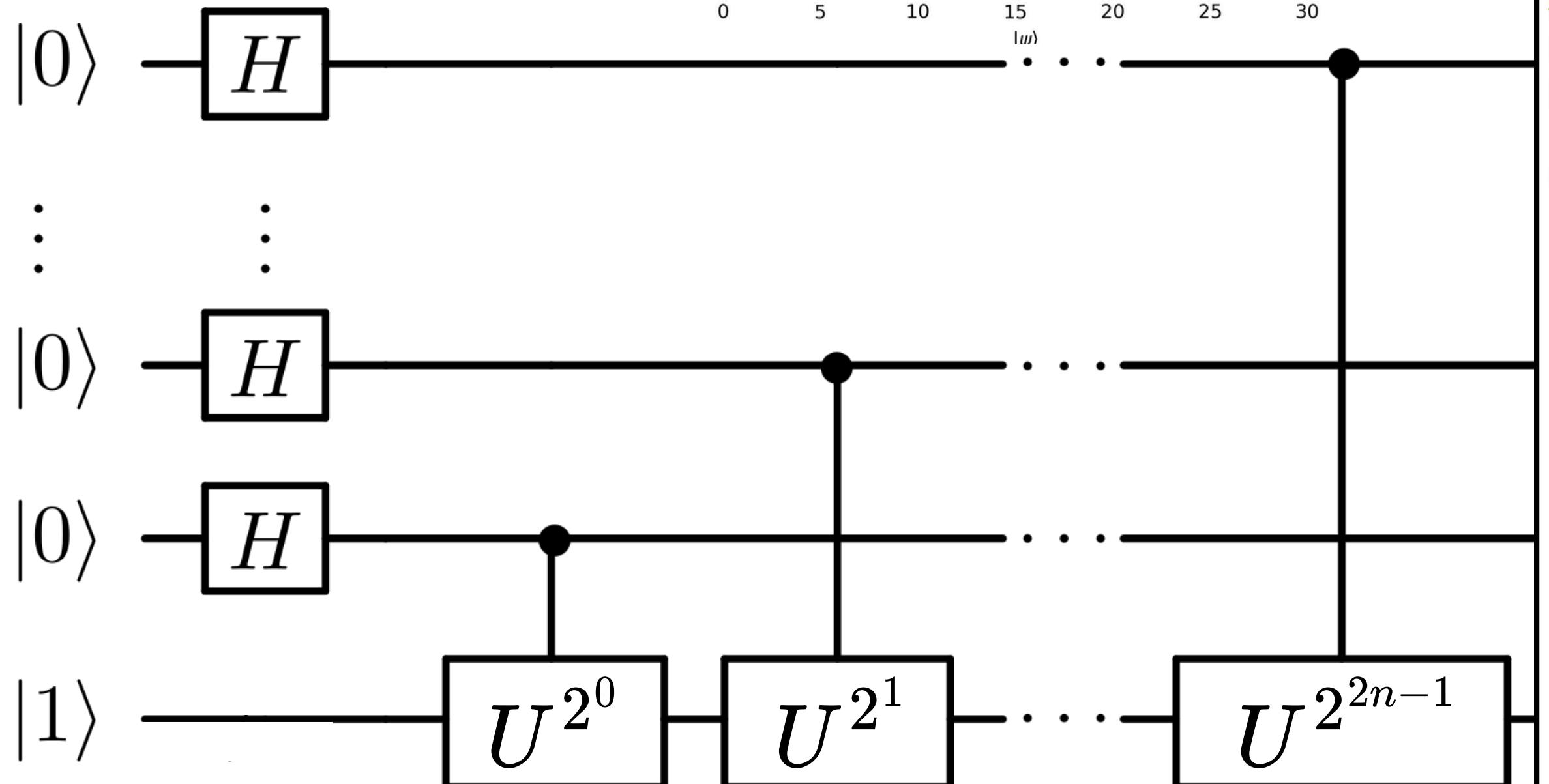
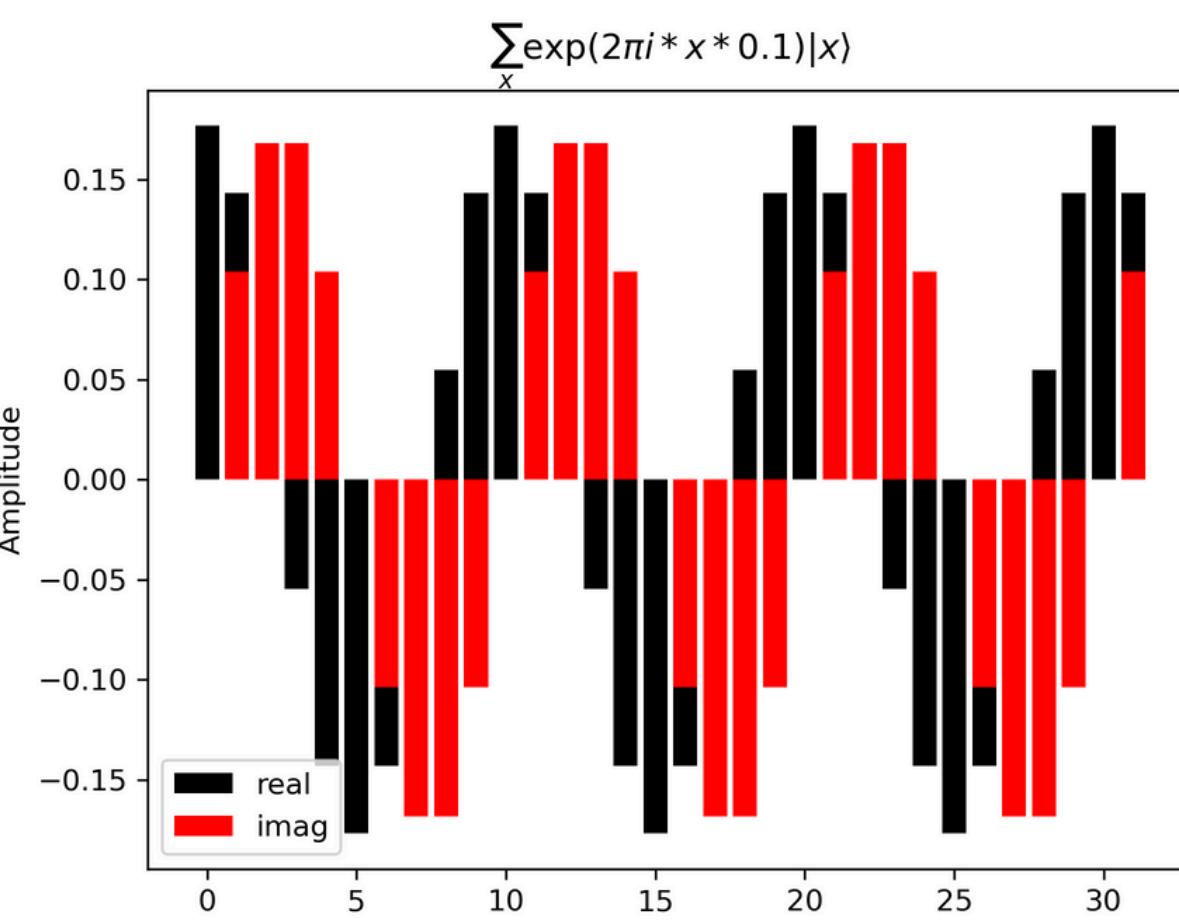
$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle|1\rangle$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle U^k|1\rangle \\ = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) |k\rangle|1\rangle$$

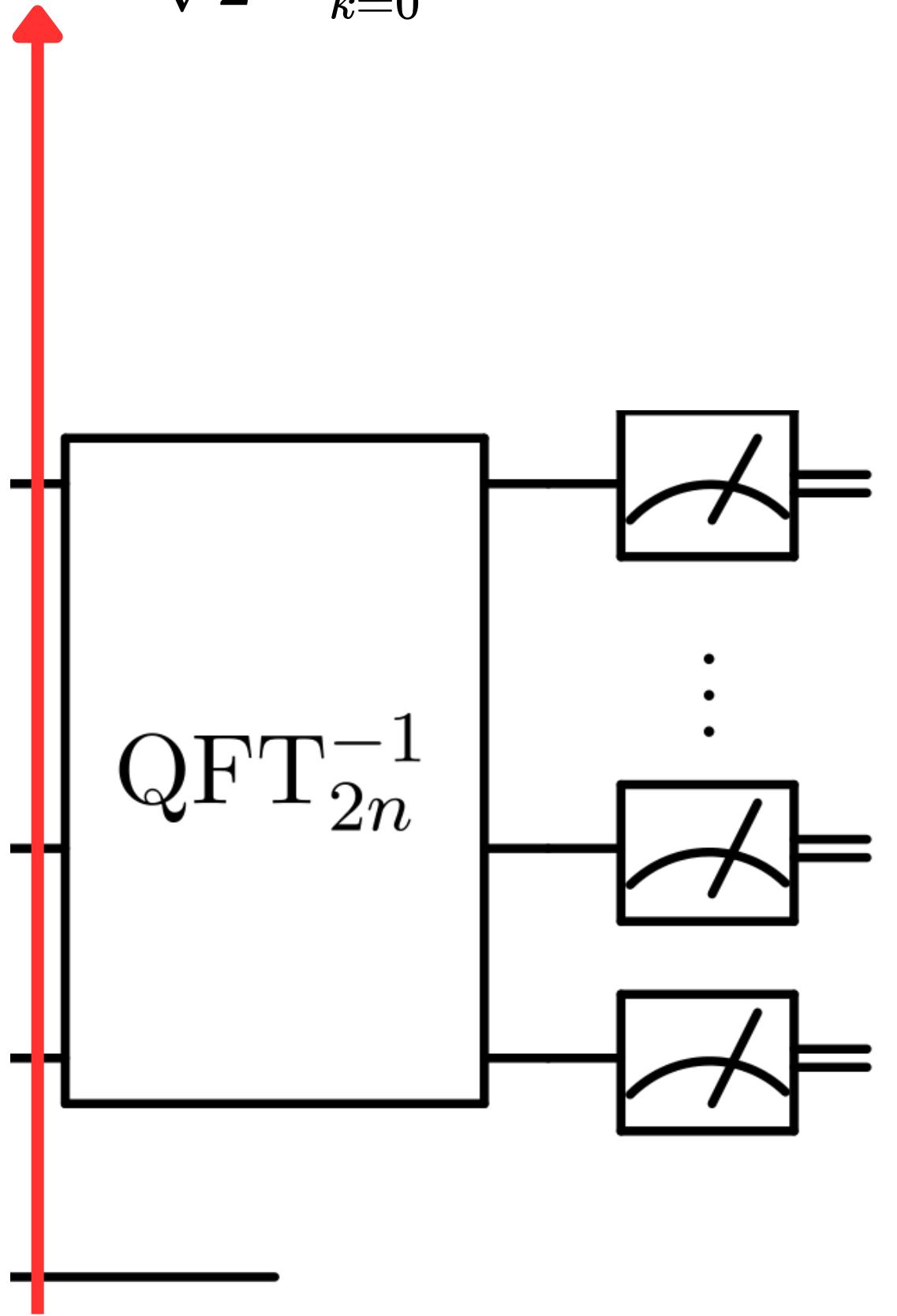


$$\begin{aligned}
|\psi_2\rangle &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle U^k |1\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) |k\rangle |1\rangle
\end{aligned}$$

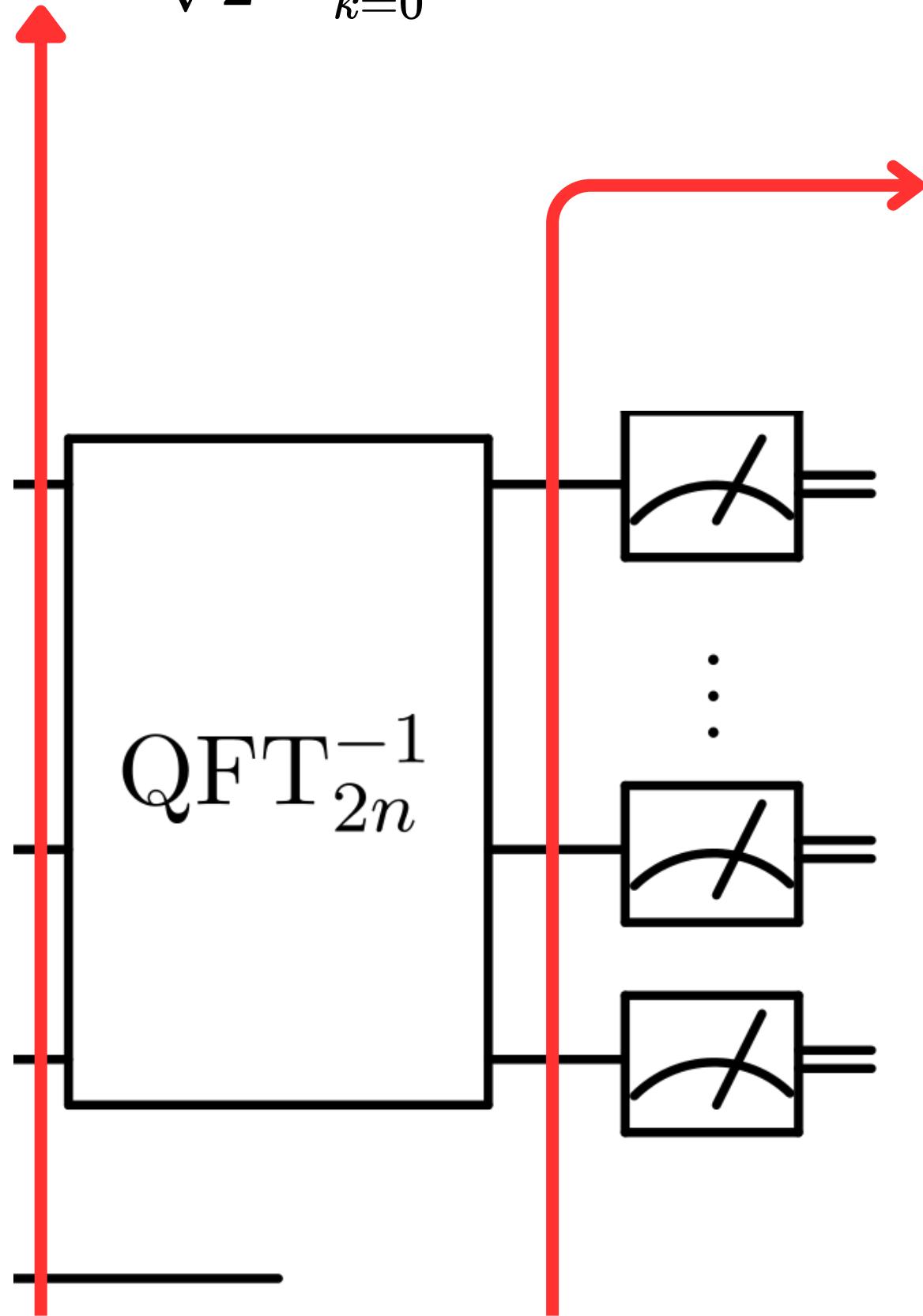




$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) |k\rangle |1\rangle$$



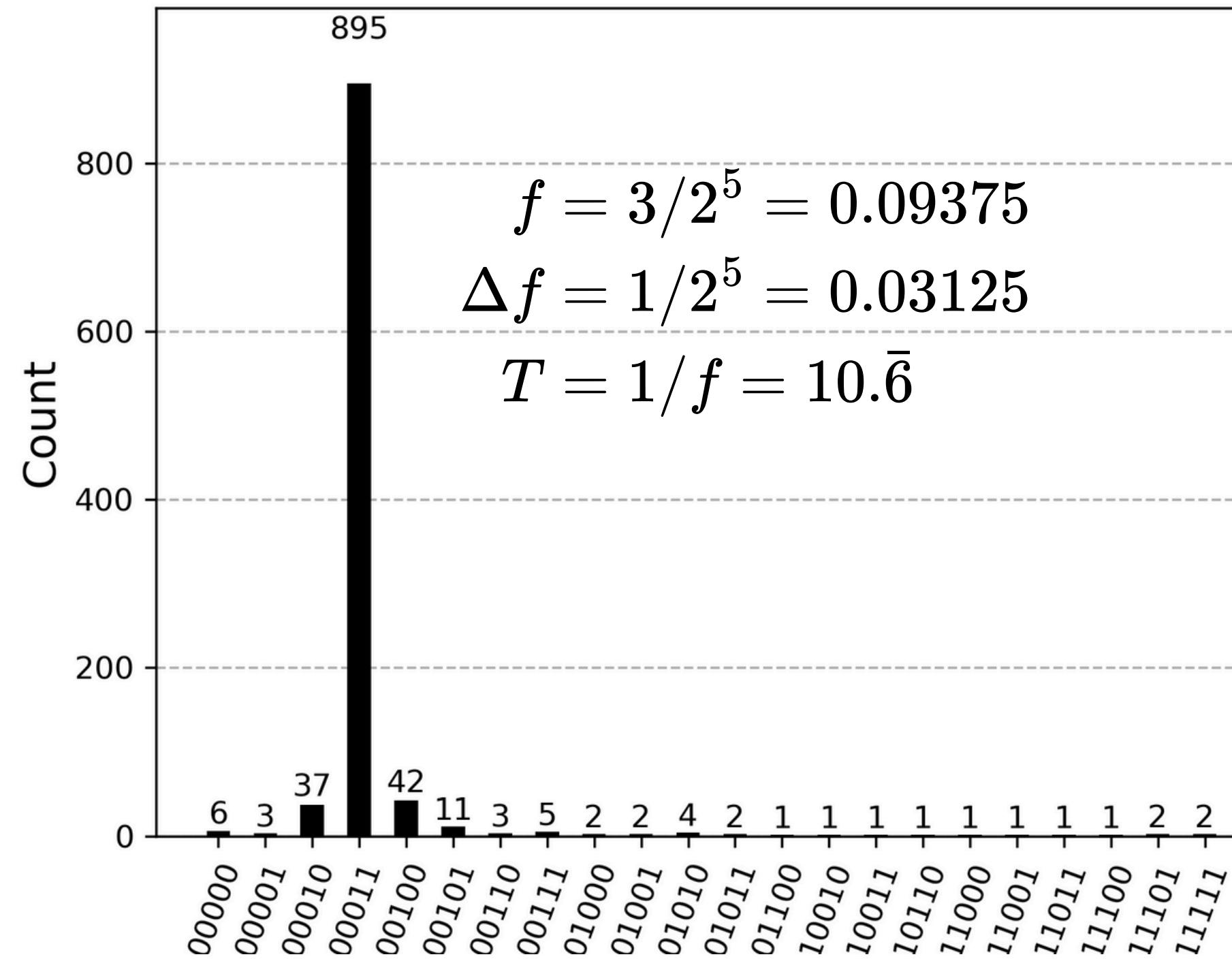
$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) |k\rangle |1\rangle$$



$$\begin{aligned}
 |\psi_3\rangle &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) \text{QFT}[k\rangle |1\rangle] \\
 &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \exp(2\pi i k \phi) \frac{1}{\sqrt{2^n}} \sum_{\ell=0}^{2^n-1} \exp(-2\pi i k \ell / 2^n) |\ell\rangle |1\rangle \\
 &= \frac{1}{2^n} \sum_{k,\ell=0}^{2^n-1} \exp\left(\frac{2\pi i k}{2^n}(2^n \phi - \ell)\right) |\ell\rangle |1\rangle \\
 &= \sum_{\ell=0}^{2^n-1} \delta_{2^n \phi, \ell} |\ell\rangle |1\rangle \quad 2^n \phi \in \mathbb{N} \\
 &= |2^n \phi\rangle |1\rangle
 \end{aligned}$$

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{k,\ell=0}^{2^n-1} \exp\left(\frac{2\pi i k}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle = |2^n\phi\rangle |1\rangle$$

i



?

Kernel de Dirichlet

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{2^n} \sum_{k,\ell=0}^{2^n-1} \exp\left(\frac{2\pi i k}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle \\ &= \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \frac{\sin[\pi(2^n\phi - \ell)]}{\sin[\pi(2^n\phi - \ell)/2^n]} \exp\left(i\pi \frac{2^n - 1}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle \end{aligned}$$

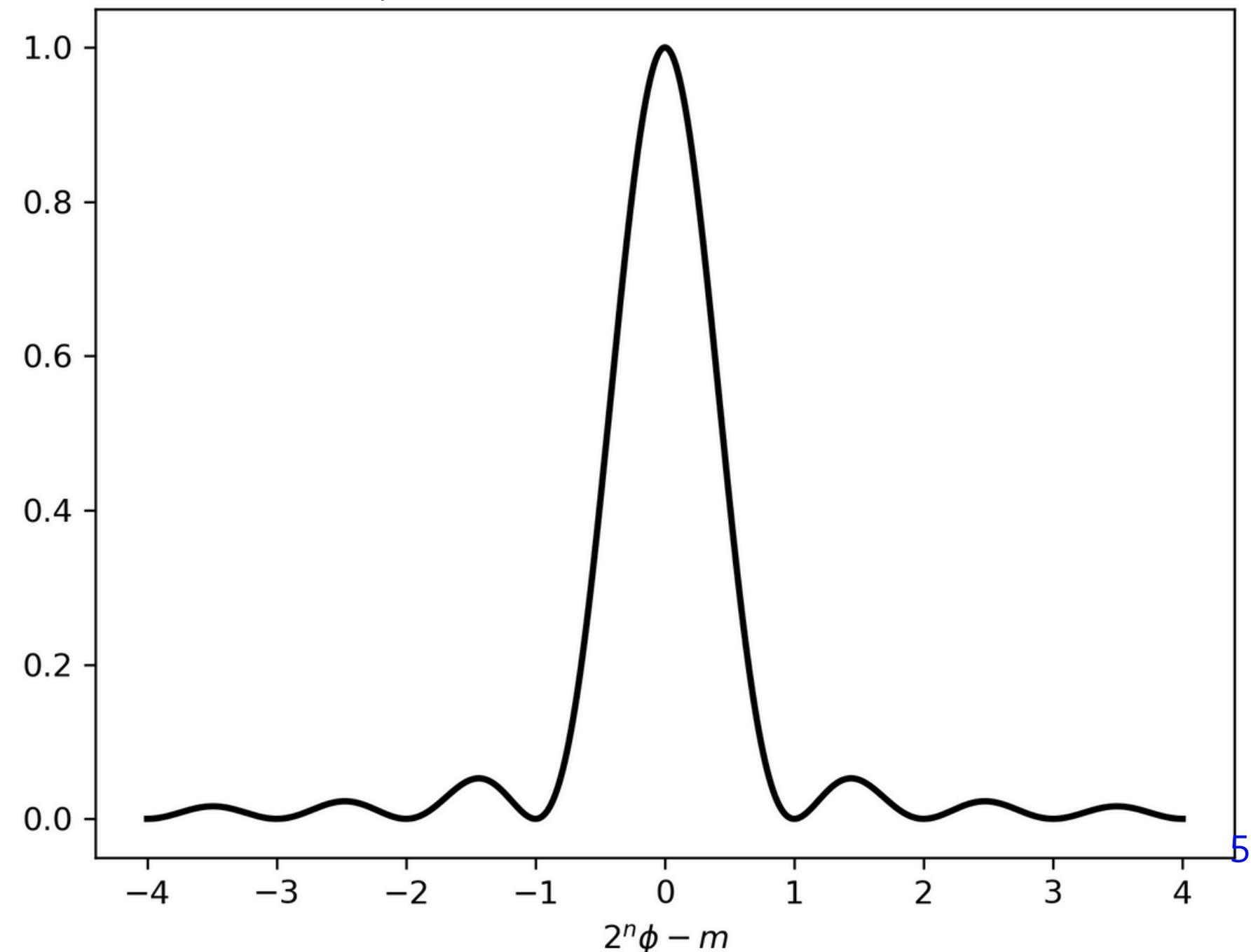
Kernel de Dirichlet

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{k,\ell=0}^{2^n-1} \exp\left(\frac{2\pi i k}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle$$

$$= \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \frac{\sin[\pi(2^n\phi - \ell)]}{\sin[\pi(2^n\phi - \ell)/2^n]} \exp\left(i\pi \frac{2^n - 1}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle$$

Probabilidad de medir estado m:

$$\begin{aligned} P(|m\rangle) &= |\langle m|\psi_3\rangle|^2 \\ &= \frac{1}{4^n} \frac{\sin^2[\pi(2^n\phi - m)]}{\sin^2[\pi(2^n\phi - m)/2^n]} \end{aligned}$$



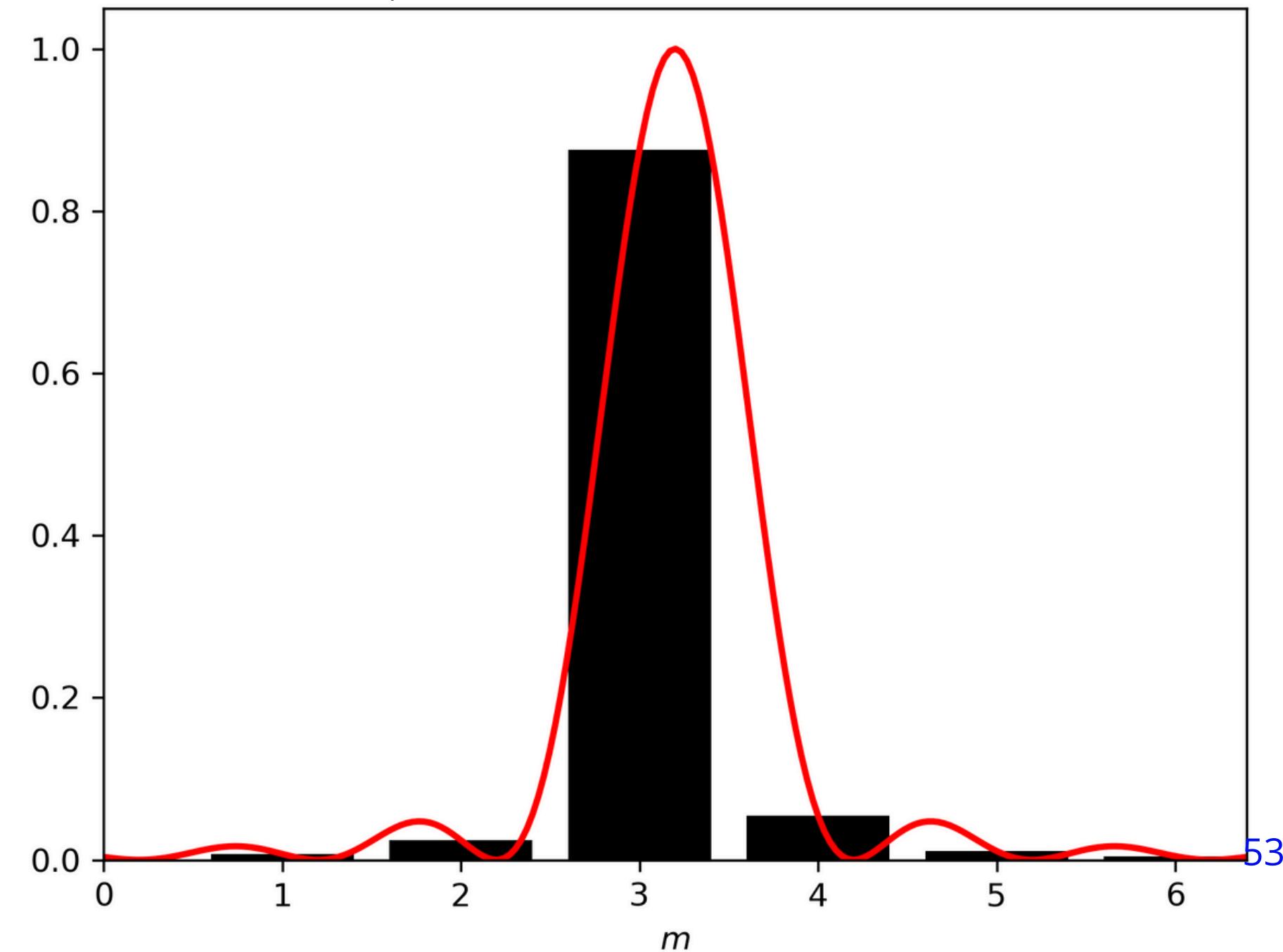
Kernel de Dirichlet

$$\begin{aligned} |\psi_3\rangle &= \frac{1}{2^n} \sum_{k,\ell=0}^{2^n-1} \exp\left(\frac{2\pi ik}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle \\ &= \frac{1}{2^n} \sum_{\ell=0}^{2^n-1} \frac{\sin[\pi(2^n\phi - \ell)]}{\sin[\pi(2^n\phi - \ell)/2^n]} \exp\left(i\pi \frac{2^n - 1}{2^n}(2^n\phi - \ell)\right) |\ell\rangle |1\rangle \end{aligned}$$

Probabilidad de medir estado m :

$$\begin{aligned} P(|m\rangle) &= |\langle m|\psi_3\rangle|^2 \\ &= \frac{1}{4^n} \frac{\sin^2[\pi(2^n\phi - m)]}{\sin^2[\pi(2^n\phi - m)/2^n]} \end{aligned}$$

$$n = 5, \quad \phi = 0.1$$



Fuente de error QPE: Desalineamiento entre fase y estado mas cercano

Para el entero ℓ mas cercano: $\Delta := |2^n\phi - \ell| \leq \frac{1}{2}$. Luego,

$$\begin{aligned} P(|\ell\rangle) &= \frac{1}{4^n} \frac{\sin^2[\pi\Delta]}{\sin^2[\pi\Delta/2^n]} \\ &\geq \frac{1}{4^n} \frac{4\Delta^2}{\sin^2[\pi\Delta/2^n]} \quad \left(|\sin(x)| > 2|x|/\pi, \quad |x| < \frac{\pi}{2} \right) \\ &\geq \frac{1}{4^n} \frac{4\Delta^2}{\pi^2\Delta^2/4^n} \quad (|\sin(x)| < |x|) \\ &= \frac{4}{\pi^2} \approx 40\% \end{aligned}$$

La probabilidad de error ϵ , de medir un estado a mayor distancia que e del correcto

$$P(|\ell - \lceil 2^n \phi \rceil| > e) = \left(\sum_{\ell=-2^t-1}^{-e-1} + \sum_{\ell=e+1}^{2^t-1} \right) \frac{1}{4^n} \frac{\sin^2[\pi(2^n \phi - \ell)]}{\sin^2[\pi(2^n \phi - \ell)/2^n]} \leq \frac{1}{2(e-1)}$$

Si queremos medir con m bits de precisión, $|\ell/2^n - \phi| < 2^{-m}$

$$|\ell - 2^n \phi| < 2^{n-m}$$

Eligiendo $e = 2^{n-m} - 1$ y reemplazando en la probabilidad de éxito $1 - \epsilon$,

$$n \geq m + \log_2 \left(2 + \frac{1}{2\epsilon} \right)$$



Checkpoint



Puntos clave

QPE

- Extrae la fase de unitarias usando la QFT como decodificador de fases y midiendo
- Requiere la aplicación controlada de potencias de la unitaria, $U^{2^i}, i = 0, \dots, n - 1$
- Tiene probabilidad de error que no depende de N
- La probabilidad de error disminuye exponencialmente añadiendo qubits

¿Cómo usamos QPE dentro del algoritmo de Shor?

Implementación de U

Buscamos implementar la función $f(x) = a^x \bmod N$

$$U_f|x\rangle|1\rangle = |x\rangle|a^x \bmod N\rangle$$

Definamos una unitaria cuya acción sea: $U|n\rangle = |an \bmod N\rangle$
y la controlamos por el primer registro como en QPE. Necesitamos ejecutar

$$U^{2^\ell}|1\rangle = |a^{2^\ell} \bmod N\rangle,$$

para esto debemos calcular las potencias $a^{2^\ell} \bmod N$, $\ell = 0 \dots n - 1$, y sumarlas en forma controlada por los dígitos de x en binario.

Implementación de U

El cálculo de las potencias de a se realiza como una multiplicación iterativa por si mismo

$$a^{2^0} \bmod N = a \bmod N$$

$$a^{2^1} \bmod N = (a^{2^0} \cdot a^{2^0}) \bmod N$$

$$a^{2^2} \bmod N = (a^{2^1} \cdot a^{2^1}) \bmod N$$

⋮

$$a^{2^{n-1}} \bmod N = (a^{2^{n-2}} \cdot a^{2^{n-2}}) \bmod N$$

lo que toma n pasos de complejidad $O(n^2)$, con $n = \log(N)$. Total: $O(\log(N)^3)$

Implementación de U

$$U^{2^\ell} |1\rangle = |a^{2^\ell} \bmod N\rangle$$

Implementamos sumas modulares CNOTs en registros auxiliares.
Luego con ellas implementamos multiplicación modular.

Es decir:

Calculamos las potencias con $O(\log(N))$ multiplicaciones, cada una con $O(\log(N))$ sumas de costo $O(\log(N))$, para una complejidad total $O(\log(N)^3)$. Este paso domina el costo del algoritmo de Shor. Existen técnicas de multiplicación rápida de menor complejidad computacional, bajando este paso a $O(\log(N)^2 \log\log(N))$ [1].

Autovalores de U

Sea U tal que $U|x\rangle = |ax \bmod N\rangle$ y sea $|u_k\rangle$ un conjunto de autovectores de U con autovalores λ_k . Luego,

$$U^r|u_k\rangle = \lambda_k^r|u_k\rangle = |u_k\rangle.$$

y los autovalores son raíces de la unidad con periodicidad r ,

$$\lambda_k = \exp(2\pi i k/r)$$

Autovectores de \mathbf{U}

El conjunto $\{|a^k \bmod N\rangle\}_{k=0}^{r-1}$ es invariante bajo la acción de U

$$\begin{aligned} U\{|1 \bmod N\rangle, |a \bmod N\rangle, \dots, |a^{r-2} \bmod N\rangle, |a^{r-1} \bmod N\rangle\} \\ = \{|a \bmod N\rangle, |a^2 \bmod N\rangle, \dots, |a^{r-1} \bmod N\rangle, |a^r \bmod N\rangle\} \\ = \{|1 \bmod N\rangle, |a \bmod N\rangle, \dots, |a^{r-2} \bmod N\rangle, |a^{r-1} \bmod N\rangle\} \end{aligned}$$

Y podemos construir los autovectores de \mathbf{U} como

$$|u_\ell\rangle = \sum_{k=0}^{r-1} \exp(-2\pi i k \ell / r) |a^k \bmod N\rangle$$

Autovectores de \mathbf{U}

Notamos que estos verifican su condición de autovector

$$\begin{aligned} U|u_\ell\rangle &= \sum_{k=0}^{r-1} \exp(-2\pi i k \ell / r) |a^{k+1} \bmod N\rangle \\ &= \exp(2\pi i \ell / r) \sum_{k=0}^{r-1} \exp(-2\pi i (k+1)\ell / r) |a^{k+1} \bmod N\rangle \\ &= \exp(2\pi i \ell / r) \sum_{k=0}^{r-1} \exp(-2\pi i k \ell / r) |a^k \bmod N\rangle \\ &= \lambda_\ell |u_\ell\rangle \end{aligned}$$

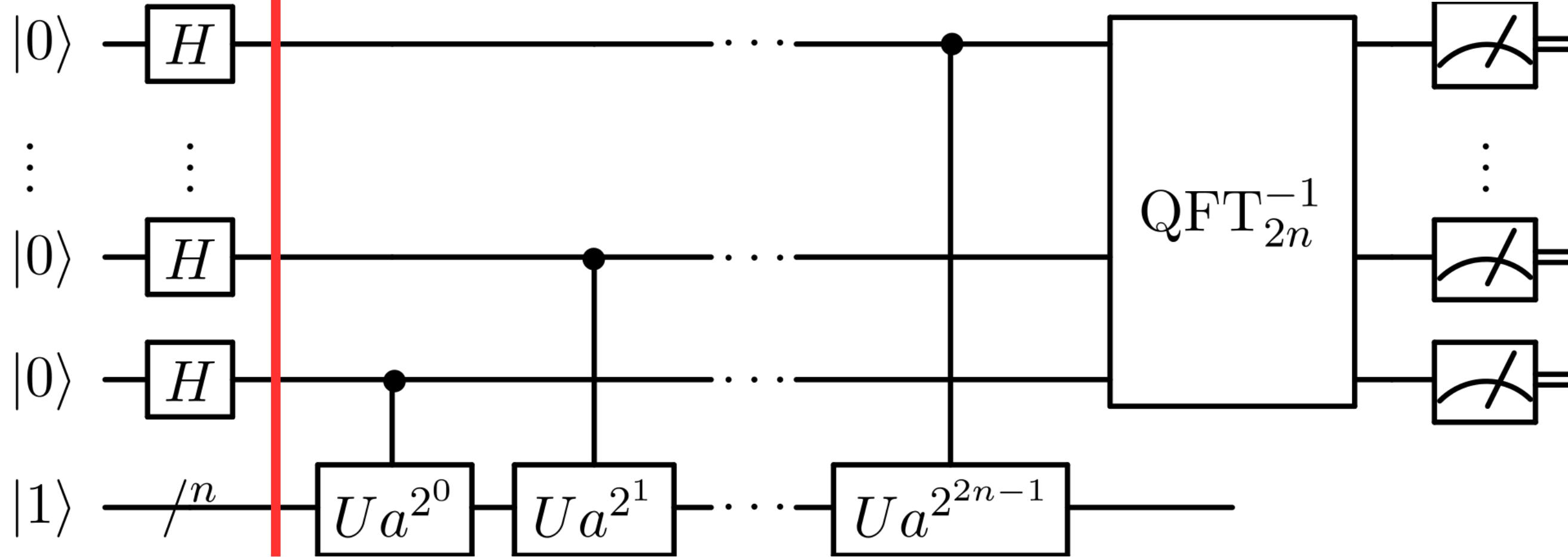
Autovectores de U

¿Cómo preparamos un autovector de U para el circuito de QPE?

No lo hacemos. Modificamos el esquema, y seguimos usando el estado 1, que pertenece al espacio generado por autovectores de U, y se puede escribir como una suma de los mismos

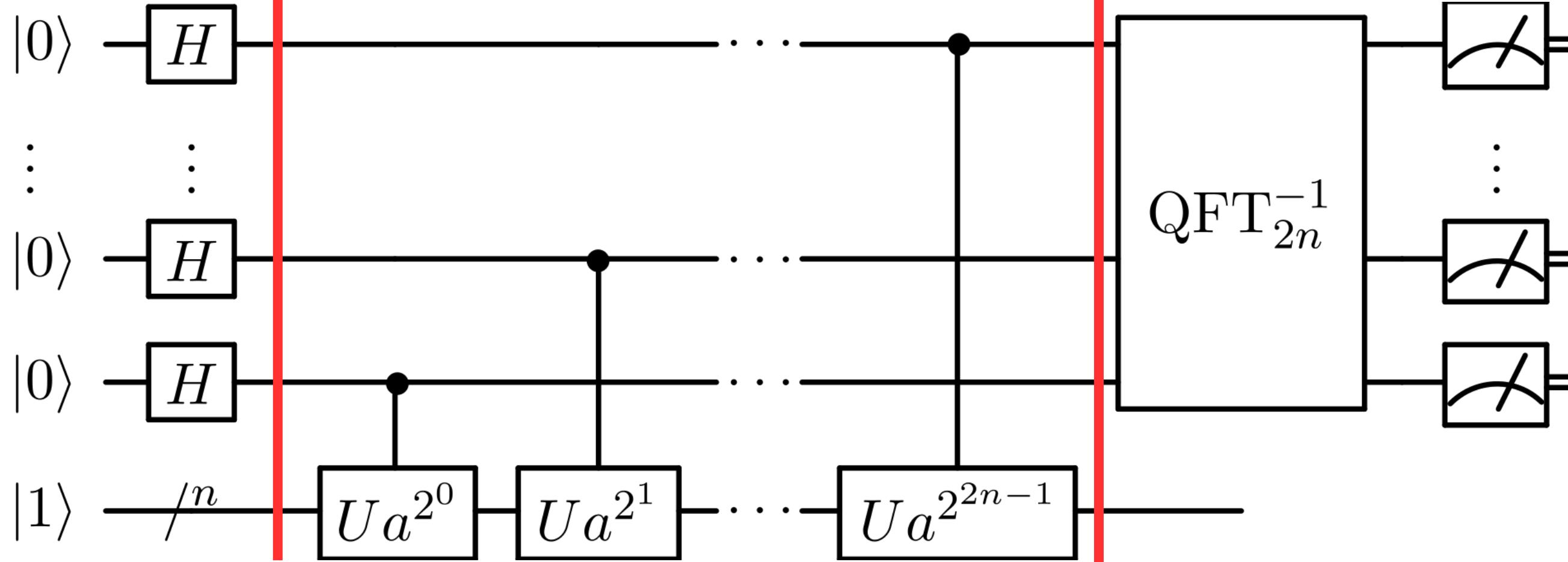
$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} c_\ell |u_\ell\rangle$$

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} |k\rangle \sum_{\ell=0}^{r-1} c_\ell |u_\ell\rangle$$

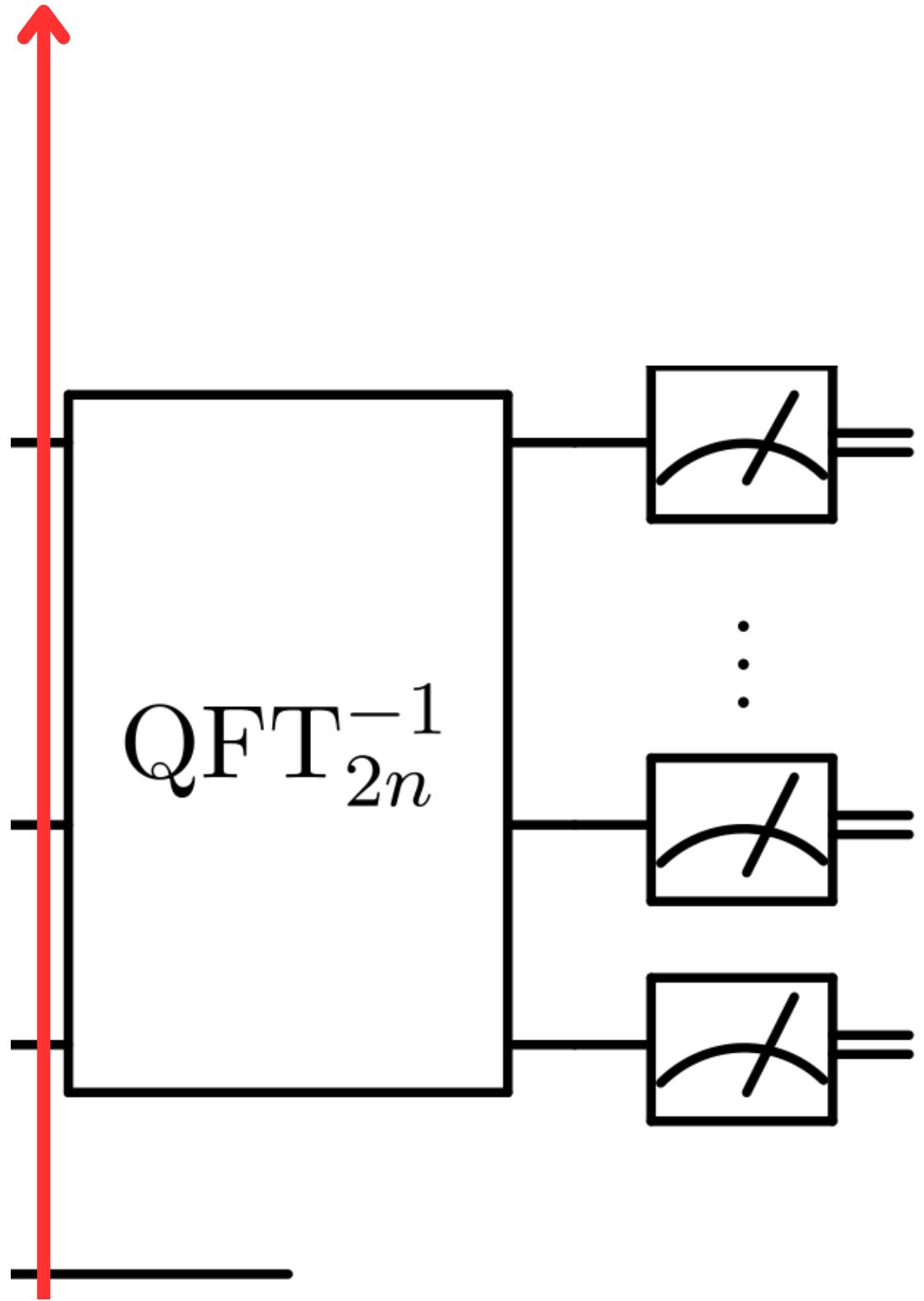


$$|\psi_1\rangle = \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} |k\rangle \sum_{\ell=0}^{r-1} c_\ell |u_\ell\rangle$$

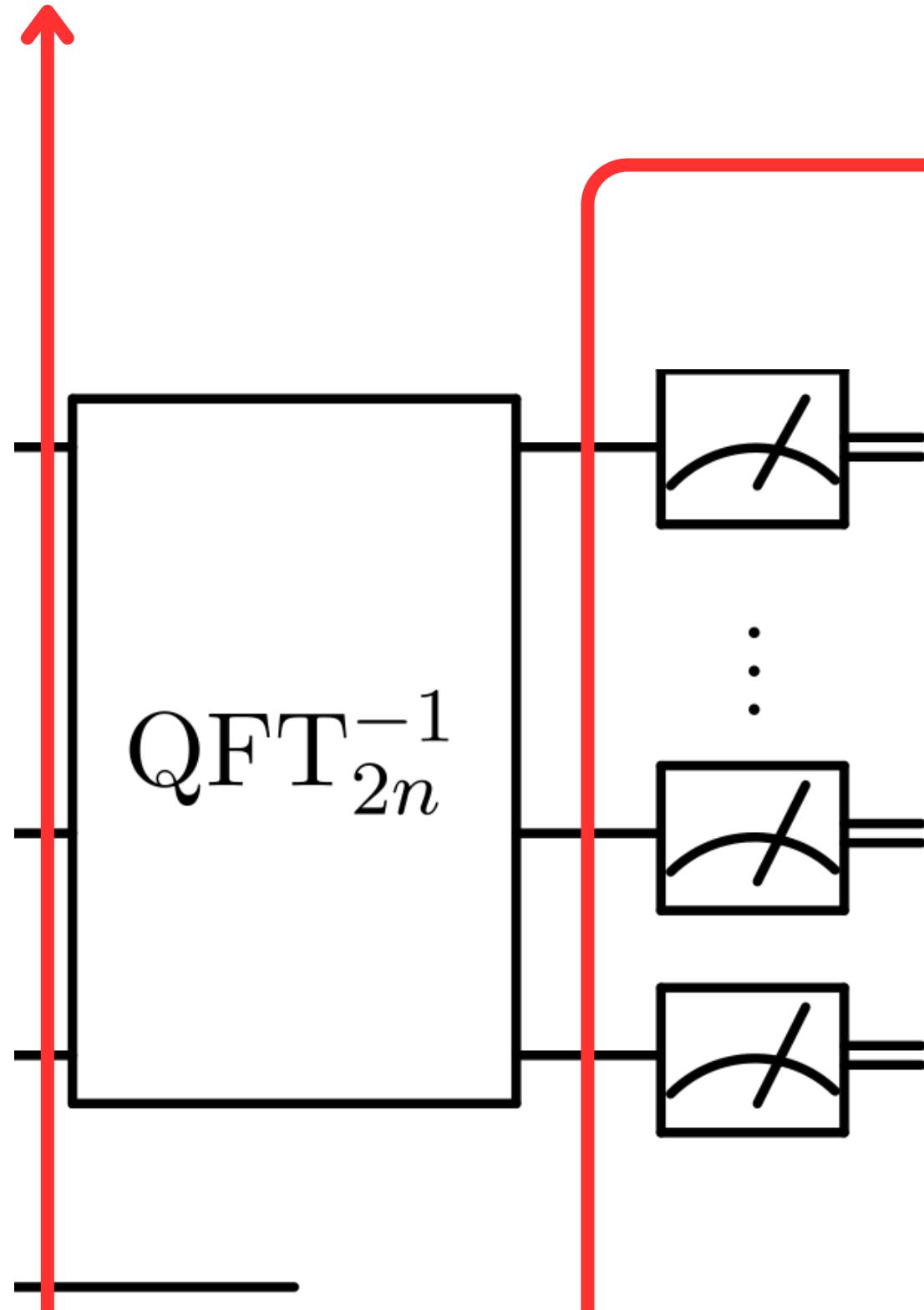
$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} \sum_{\ell=0}^{r-1} c_\ell |k\rangle U^k |u_\ell\rangle \\ &= \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} \sum_{\ell=0}^{r-1} c_\ell \exp(2\pi i k \ell / r) |k\rangle |u_\ell\rangle \end{aligned}$$



$$|\psi_2\rangle = \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} \sum_{\ell=0}^{r-1} c_\ell \exp(2\pi i k \ell / r) |k\rangle |u_\ell\rangle$$



$$|\psi_2\rangle = \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} \sum_{\ell=0}^{r-1} c_\ell \exp(2\pi i k \ell / r) |k\rangle |u_\ell\rangle$$



$$\begin{aligned}
 |\psi_2\rangle &= \frac{1}{\sqrt{2^n r}} \sum_{k=0}^{2^n-1} \sum_{\ell=0}^{r-1} c_\ell \exp(2\pi i k \ell / r) \text{QFT}^{-1} |k\rangle |u_\ell\rangle \\
 &= \frac{1}{2^n \sqrt{r}} \sum_{\ell=0}^{r-1} c_\ell \sum_{k,m=0}^{2^n-1} \exp(2\pi i k (2^n \ell / r - m) / 2^n) |m\rangle |u_\ell\rangle \\
 &= \frac{1}{\sqrt{r}} \sum_{\ell=0}^{r-1} c_\ell |2^n \ell / r\rangle |u_\ell\rangle \quad 2^n \ell / r \in \mathbb{N}
 \end{aligned}$$

Superposición de estados, donde todos codifican alguna fase que nos permite extraer r

Obtención de r

Tenemos un estimado de $2^n l/r$. Ahora extraemos r usando fracciones continuas
Una fracción continua es una suma anidada de enteros e inversos

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

a_0 es la parte entera de x

a_1 es la parte entera de $1/(x - a_0)$

a_i es la parte entera del inverso
del último residuo

Los convergentes p_i/q_i se obtienen

$$p_i = a_i p_{i-1} + p_{i-2}$$

$$q_i = a_i q_{i-1} + q_{i-2}$$

$$p_{-1} = 1, p_0 = a_0$$

$$q_{-1} = 0, q_0 = 1$$

Se requieren $O(\log(N))$ iteraciones, de costo $O(\log(N)^2)$ cada una. Costo total $O(\log(N)^3)^{70}$

Ejemplo: Fracciones continuas para 45.3247

$$a_0 = 45$$

$$a_1 = \lfloor 1/0.3247 \rfloor = \lfloor 3.0797\dots \rfloor$$

$$a_2 = \lfloor 1/0.0797\dots \rfloor = \lfloor 12.5366\dots \rfloor$$

$$a_3 = \lfloor 1/0.5366\dots \rfloor = \lfloor 1.8633\dots \rfloor$$

⋮

$$45.3247 = 45 + \cfrac{1}{3 + \cfrac{1}{12 + \cfrac{1}{1 + \dots}}}$$

iteración	a_i	p_i	q_i	p_i/q_i
-1	-	1	0	-
0	45	$a_0 = 45$	1	45
1	3	136	3	45.3333
2	12	1677	37	45.3514
3	1	1813	40	45.325

Obtención de r

Teorema: Si ℓ/r es un número racional y $\left| \frac{\ell}{r} - \phi \right| \leq \frac{1}{2r^2}$,

entonces ℓ/r es un convergente de la fracción continua de ϕ .

Si se tienen al menos $m = 2 \log(N) + 1$ qubits de precisión,

entonces $|\ell/r - \phi| \leq \frac{1}{2^m} \leq \frac{1}{2r^2}$, ya que $r \leq N \leq 2^m$.

Luego, se puede usar fracciones continuas para recuperar el valor del periodo r.

Es posible que se obtenga un valor de ℓ/r tal que se simplifiquen factores en común, pero la probabilidad de éxito es $P_{\text{success}} \geq 1/4$.

Obtención de r

Teorema: Si ℓ/r es un número racional y $\left| \frac{\ell}{r} - \phi \right| \leq \frac{1}{2r^2}$,

entonces ℓ/r es un convergente de la fracción continua de ϕ .

Si se tienen al menos $m = 2 \log(N) + 1$ qubits de precisión,

entonces $|\ell/r - \phi| \leq \frac{1}{2^m} \leq \frac{1}{2r^2}$, ya que $r \leq N \leq 2^m$.

Luego, se puede usar fracciones continuas para recuperar el valor del periodo r.

Es posible que se obtenga un valor de ℓ/r tal que se simplifiquen factores en común, pero la probabilidad de éxito es $P_{\text{success}} \geq 1/4$.

**¡Ya estudiamos todos los componentes
del algoritmo de Shor!**



Checkpoint



Puntos clave

- El algoritmo de Shor modifica QPE y almacena una superposicion de autovalores en el segundo registro, requiriendo $\log(N)$ qubits
- El costo del algoritmo de Shor está dominado por la etapa de implementación de U dentro de QPE, del $O(\log(N)^3)$
- Esta complejidad puede reducirse a $O(\log(N)^2 \log\log(N))$
- Las probabilidades de error del algoritmo de Shor no crecen con el número a factorizar, por lo que si falla puede repetirse sin aumentar la complejidad
 - Al medir obtenemos $2^n \ell / r$ para algún $\ell < r$
 - Finalmente se usan fracciones continuas para extraer el valor del periodo r .

Recapitulación algoritmo de Shor

1. Se elige $1 < a < N$ aleatoriamente. Si $\text{mcd}(a, N) \neq 1$, ya se encontró un factor de N . Sino, continúa.
2. Se utiliza un dispositivo cuántico para encontrar el periodo r de la función $f(x) = a^x \bmod N$.
3. Si r es impar no es útil, se comienza de nuevo.
Si es par, continua.
4. Como $(a^r - 1) \bmod N = 0$, entonces
$$(a^{r/2} + 1)(a^{r/2} - 1) = kN, \quad k \in \mathbb{N}$$

Si $a^{r/2} \neq -1 \bmod N$, entonces con alta probabilidad uno de los siguientes es un factor no trivial de N :

$$p' = \text{mcd}(a^{r/2} + 1, N)$$
$$q' = \text{mcd}(a^{r/2} - 1, N)$$

```
def shor(N: int) -> list[int]:  
    a = random.randint(2, N-2)  
  
    # Ensure a and N are coprimes  
    f1 = math.gcd(a, N)  
    if f1 > 1:  
        return [f1, N//f1]  
  
    # Here is where the magic happens!  
    r = find_period(a, N) (QPE)  
  
    if r%2 == 1: # r is odd  
        return shor(N)  
    else:         # r is even  
        ar2 = pow(a, r//2)  
        p, q = ar2 + 1, ar2 - 1  
  
        if p%N == 0: # This case would return [1, N]  
            return shor(N)  
  
    return [math.gcd(p, N), math.gcd(q, N)]
```

Recapitulación algoritmo de Shor

1. Se elige $1 < a < N$ aleatoriamente. Si $\text{mcd}(a, N) \neq 1$, ya se encontró un factor de N . Sino, continúa.
2. Se utiliza un dispositivo cuántico para encontrar el periodo r de la función $f(x) = a^x \bmod N$.

3. Si r es impar no es útil, se comienza de nuevo.
Si es par, continua.

4. Como $(a^r - 1) \bmod N = 0$, entonces
 $(a^{r/2} + 1)(a^{r/2} - 1) = kN, \quad k \in \mathbb{N}$

Si $a^{r/2} \neq -1 \bmod N$, entonces con alta probabilidad uno de los siguientes es un factor no trivial de N :

$$p' = \text{mcd}(a^{r/2} + 1, N)$$
$$q' = \text{mcd}(a^{r/2} - 1, N)$$

Teorema 5.3 Nielsen & Chuang

$P_{\text{success}} \wedge 1/2$

```
def shor(N: int) -> list[int]:  
    a = random.randint(2, N-2)  
  
    # Ensure a and N are coprimes  
    f1 = math.gcd(a, N)  
    if f1 > 1:  
        return [f1, N//f1]  
  
    # Here is where the magic happens!  
    r = find_period(a, N) (QPE)  
  
    if r%2 == 1: # r is odd  
        return shor(N)  
    else: # r is even  
        ar2 = pow(a, r//2)  
        p, q = ar2 + 1, ar2 - 1  
  
        if p%N == 0: # This case would return [1, N]  
            return shor(N)  
  
    return [math.gcd(p, N), math.gcd(q, N)]
```

Fuentes algorítmicas de fallo

- Resolución QFT
 - La resolución es suficiente para que fracciones continuas extraiga r si el número de qubits en el primer registro es $n \geq 2 \log(N) + 1$
 - Probabilidad de medir un estado que no esté en la vecindad del ideal
 - El error no depende de N y decae exponencialmente con n
 - Probabilidad fallo en fracciones continuas
 - Error no depende de N, y puede repetirse QPE-FC. $P_{\text{success}} \geq 1/4$
 - Probabilidad de encontrar un r impar
 - Probabilidad de que los factores que encontramos de N sean triviales
- $\left. \begin{array}{l} \\ \\ \end{array} \right\} P_{\text{success}} \geq 1/2$

¡Gracias!