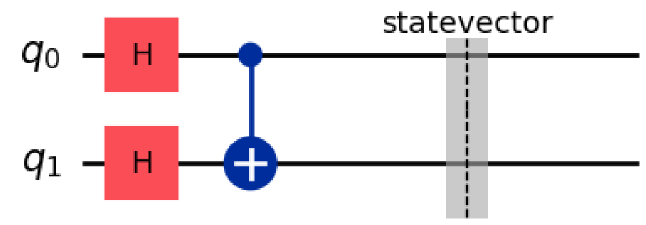
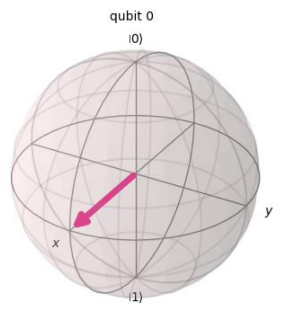
ECE 550/650 QC
Quantum Phase Estimation
Quantum Fourier Transform

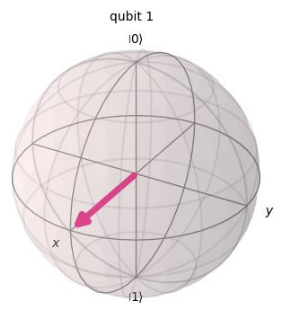
Robert Niffenegger

UMass Amherst | College of Engineering

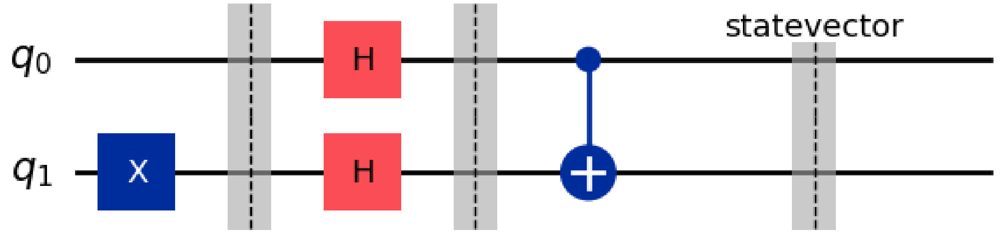
Review - Phase Kickback



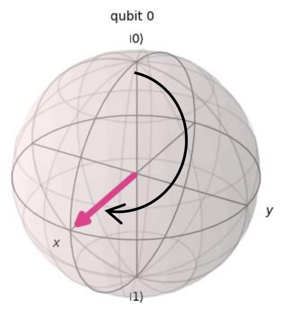


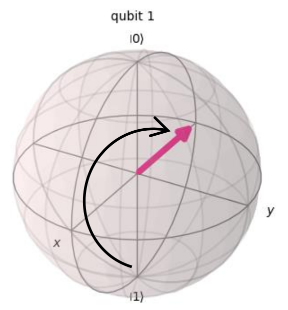


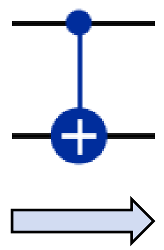
$\frac{1}{2}[\,|00\rangle+|01\rangle+|10\rangle+|11\rangle\,]$

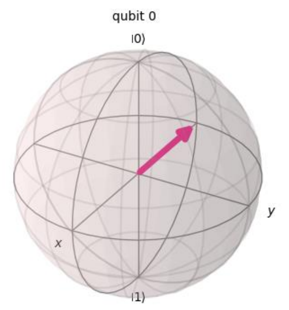


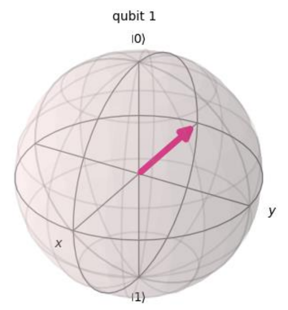
$|+\rangle$ $|-\rangle$ $|-\rangle$ $|-\rangle$











$\text{CNOT}|-+\rangle$

$= \text{CNOT}\ \frac{1}{2}[|00\rangle+|01\rangle-|10\rangle-|11\rangle]$

$= \frac{1}{2}[|00\rangle-|01\rangle-|10\rangle+|11\rangle]$

$= \frac{1}{\sqrt{2}}[|0\rangle-|1\rangle] \otimes \frac{1}{\sqrt{2}}[|0\rangle-|1\rangle]$

$= |-\rangle \otimes |-\rangle$

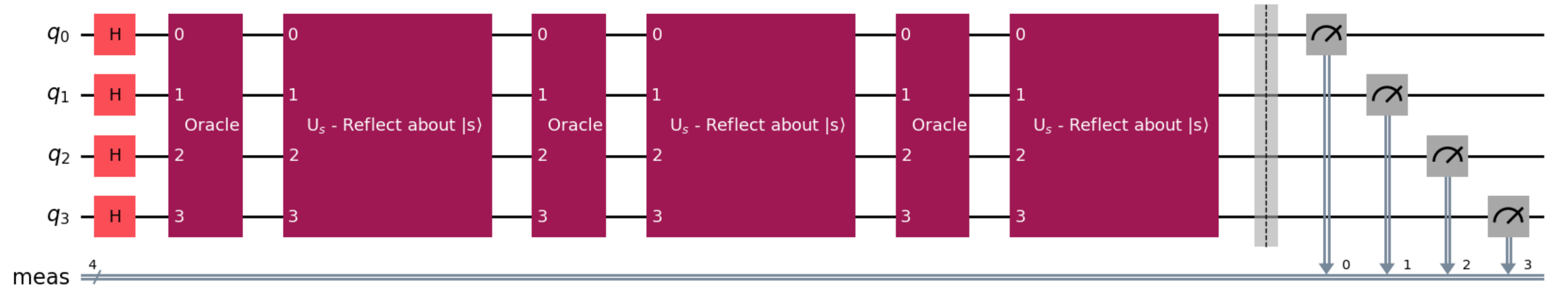
$= |--\rangle$

UMass Amherst | College of Engineering

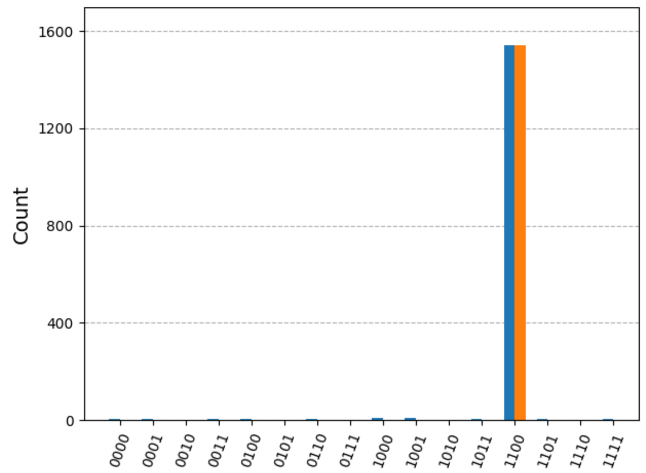
Review - Amplitude Amplification (Grover Search)

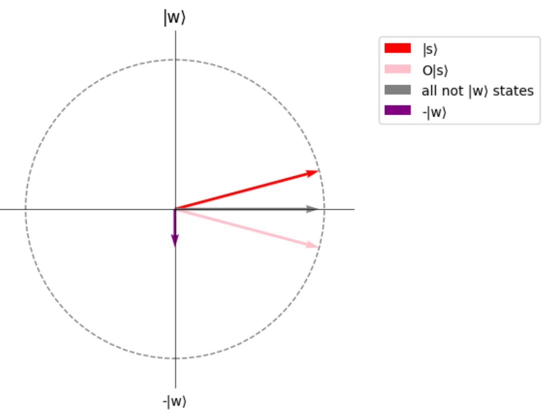
n bit register

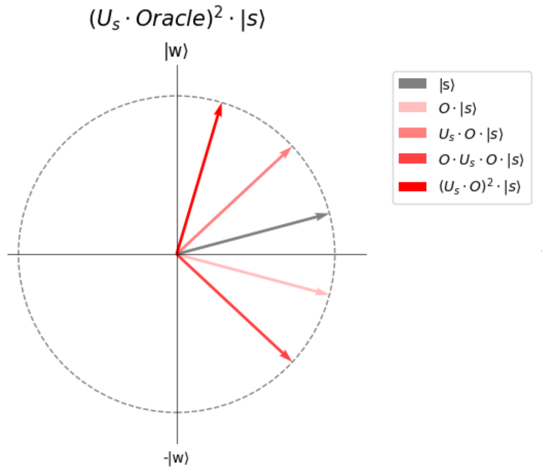
Repeat $\frac{\pi}{4}\sqrt{N}$ times



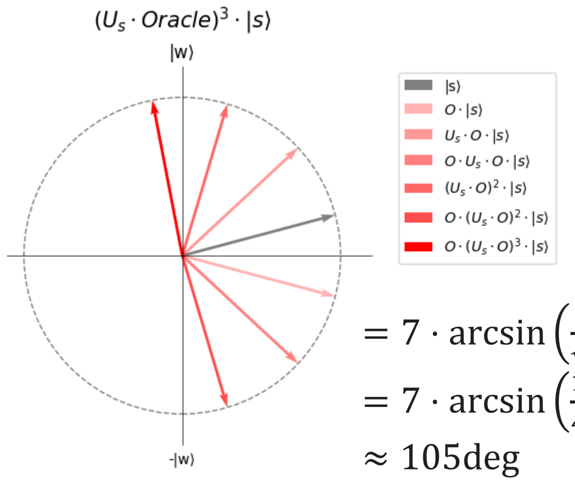
Repeat $\frac{\pi}{4}\sqrt{16}$ times (~3)







$(U_s \cdot Oracle)^2 \cdot |s\rangle$
|w⟩
-|w⟩
|s⟩
$O \cdot |s\rangle$
$U_s \cdot O \cdot |s\rangle$
$O \cdot U_s \cdot O \cdot |s\rangle$
$(U_s \cdot O)^2 \cdot |s\rangle$



$(U_s \cdot Oracle)^3 \cdot |s\rangle$
|w⟩
-|w⟩
|s⟩
$O \cdot |s\rangle$
$U_s \cdot O \cdot |s\rangle$
$O \cdot U_s \cdot O \cdot |s\rangle$
$(U_s \cdot O)^2 \cdot |s\rangle$
$O \cdot (U_s \cdot O)^2 \cdot |s\rangle$
$O \cdot (U_s \cdot O)^3 \cdot |s\rangle$

$$= 7 \cdot \arcsin\left(\frac{1}{\sqrt{N}}\right)$$
$$= 7 \cdot \arcsin\left(\frac{1}{4}\right)$$
$$\approx 105\text{deg}$$

Overrotated!!

Now we see that we've actually over rotated past the optimal answer and the further interations would just continue to rotated us farther from the answer.

The angle is now $\theta = 7 \cdot \arcsin(1/\sqrt{N}) = 7 \cdot \arcsin(1/4) \approx 105\,\text{deg}$

Only 3 iteractions were needed!

Turns out it can be shown that the optimal number of iterations is: $\frac{\pi}{4}\sqrt{N}$

This is because we always want a final angle of $\frac{\pi}{4}$ (90°) and the initial angle of the superposition state will always be : $\arcsin(1/\sqrt{N})$

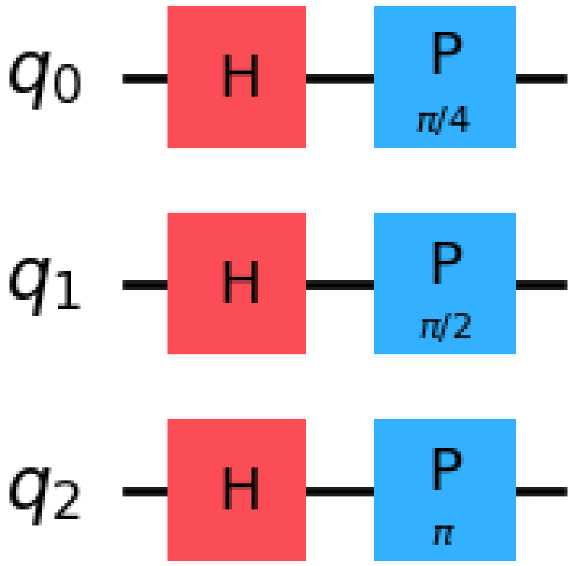
Thus dividing them gives the optimal number of iterations.

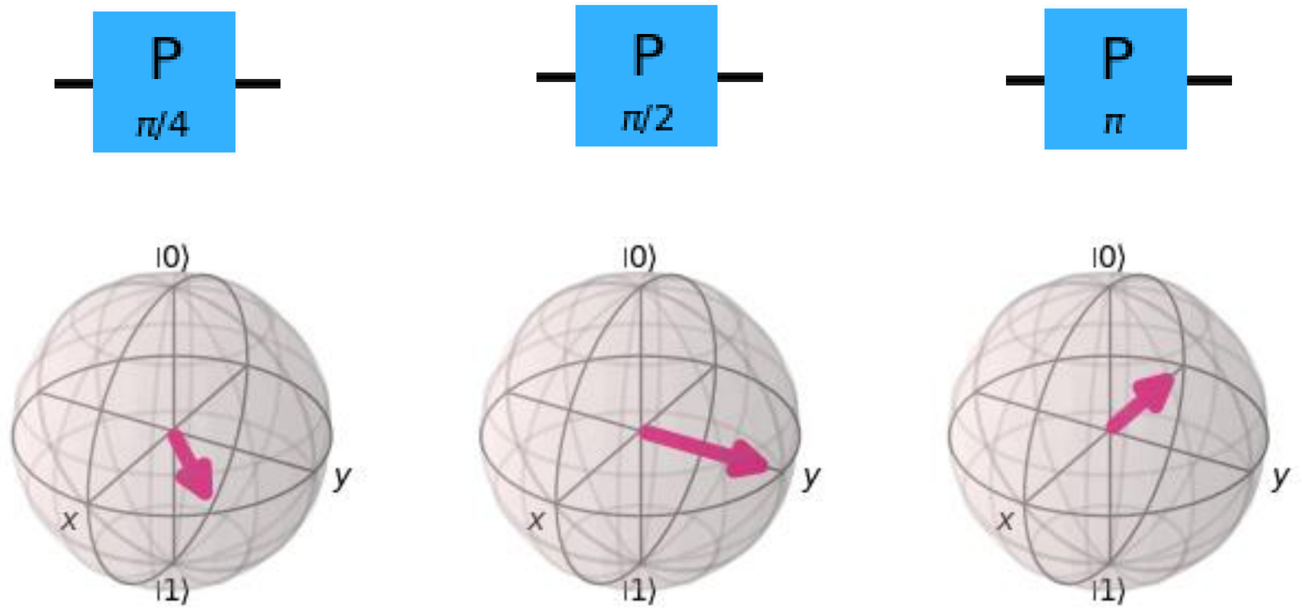
> See Thomas Wong's Textbook

Which for n=4 and N=16 means the optimal number of iterations is : $\frac{\pi}{4}\sqrt{16} = \pi \approx 3$

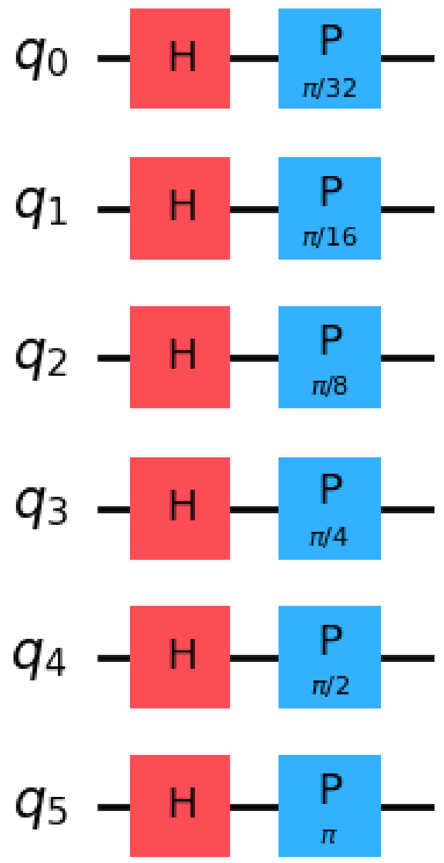
UMassAmherst | College of Engineering

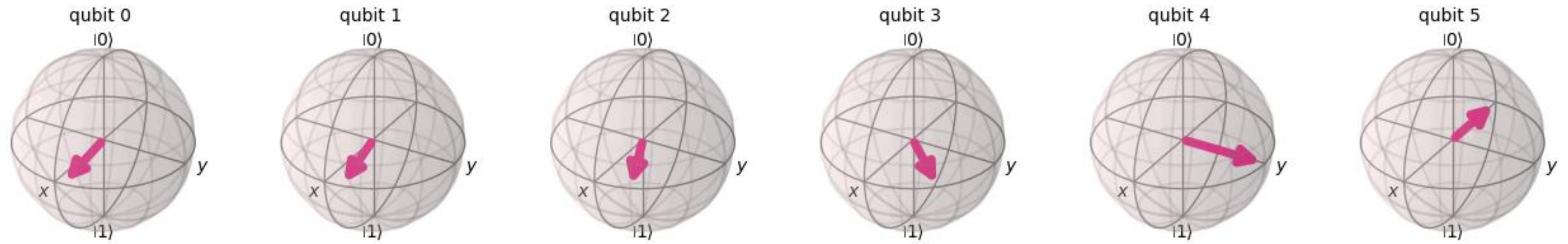
Quantum Phase



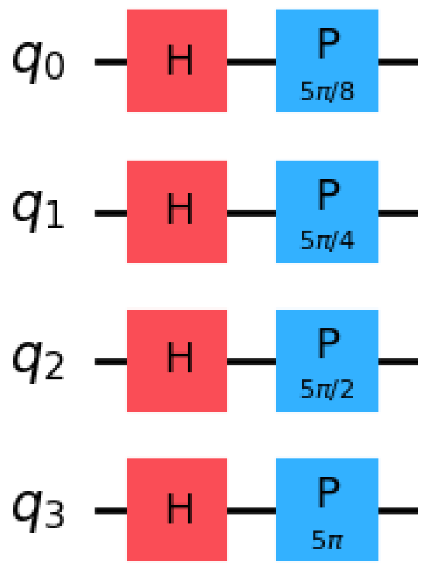


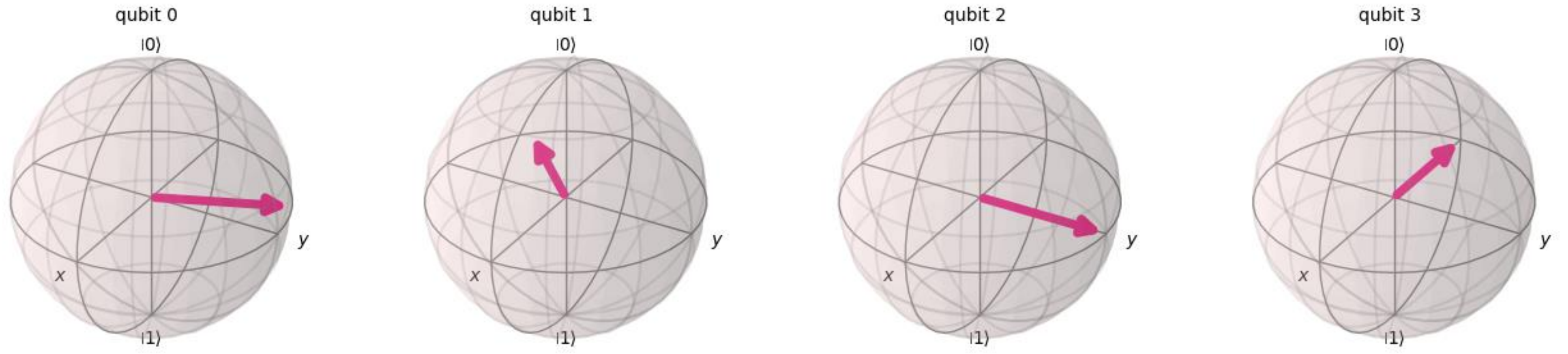
Precision


q_0
H
P
π/32
q_1
H
P
π/16
q_2
H
P
π/8
q_3
H
P
π/4
q_4
H
P
π/2
q_5
H
P
π


qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
|0⟩
|1⟩
x
y

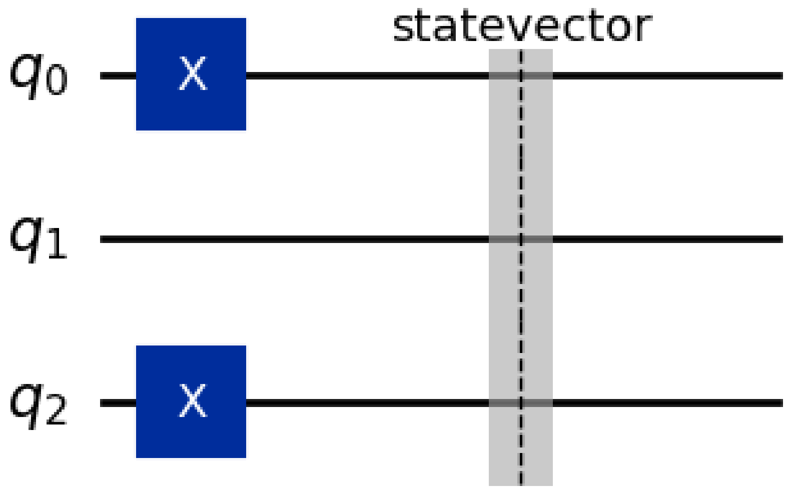
Non integer phase

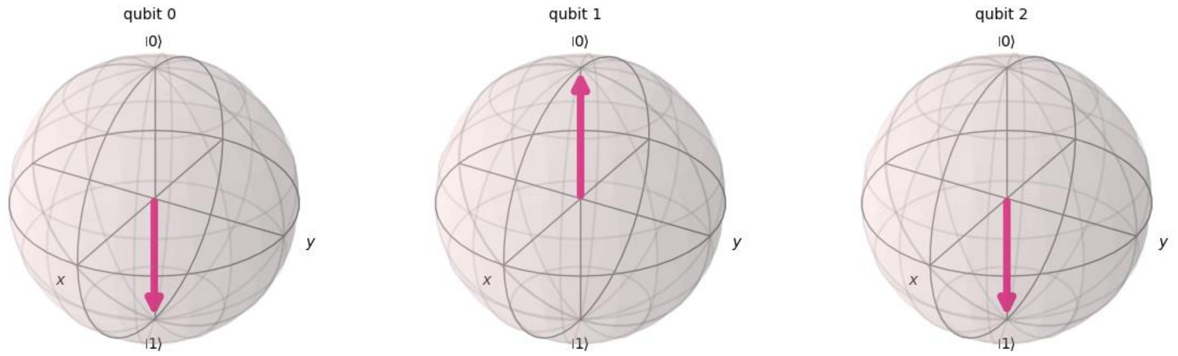




UMass Amherst | College of Engineering

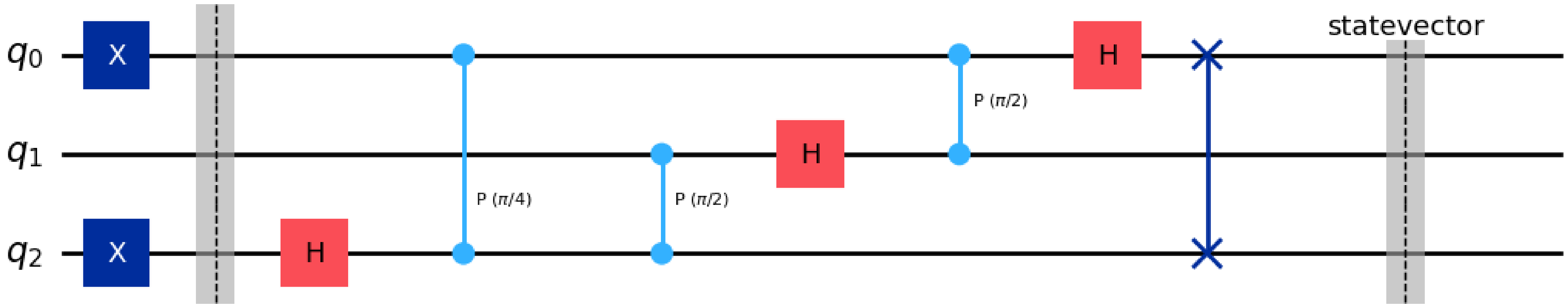
BIT space

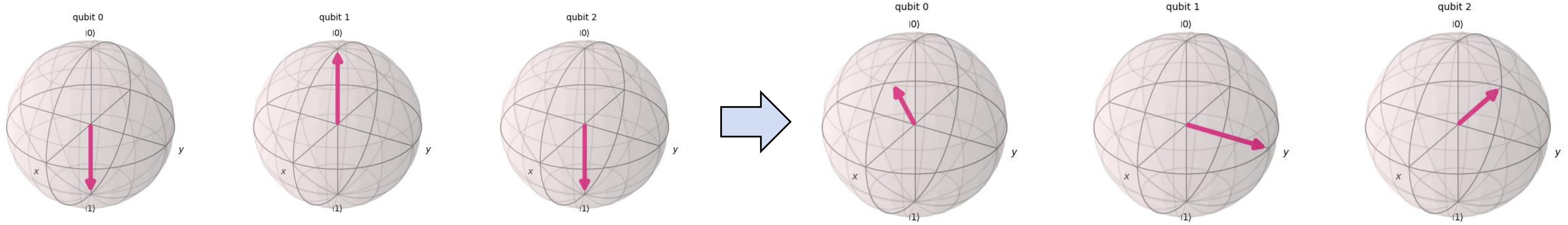




UMassAmherst | College of Engineering

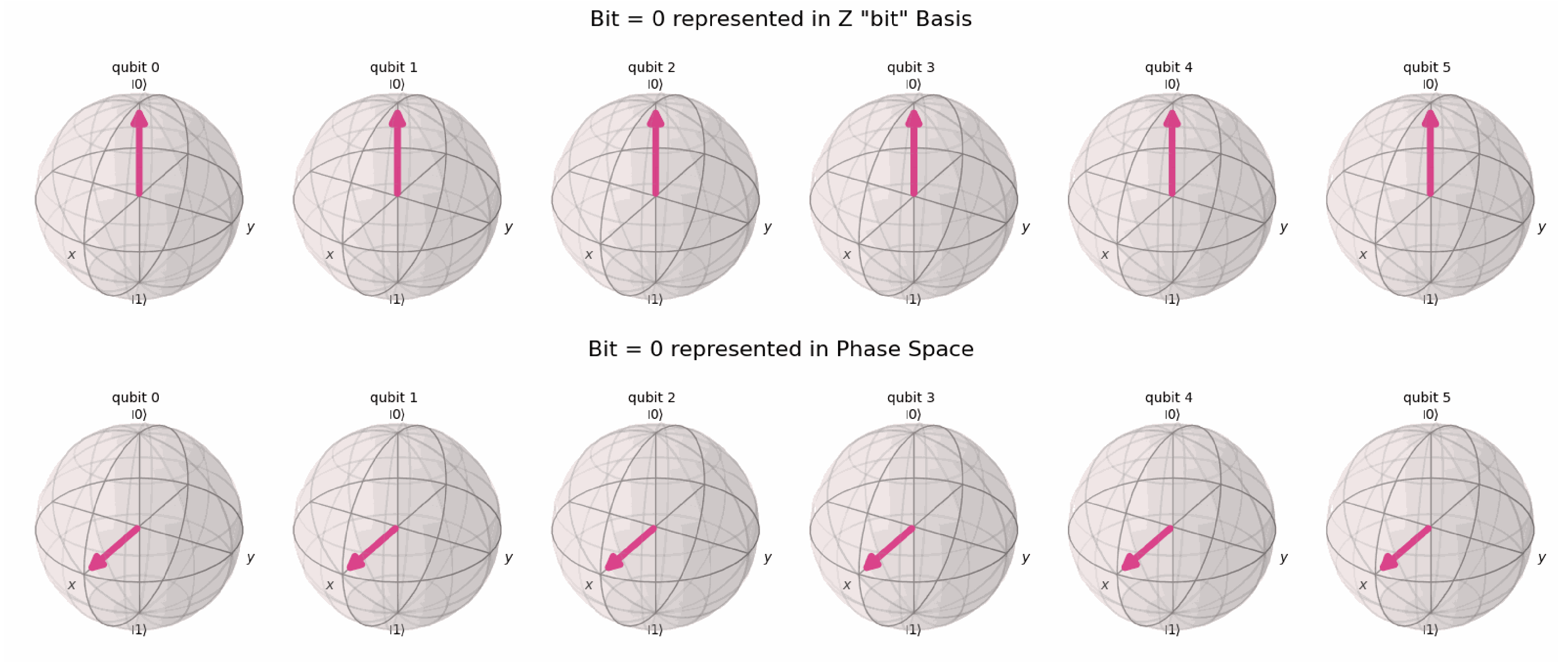
Quantum Fourier Transform = BIT → PHASE





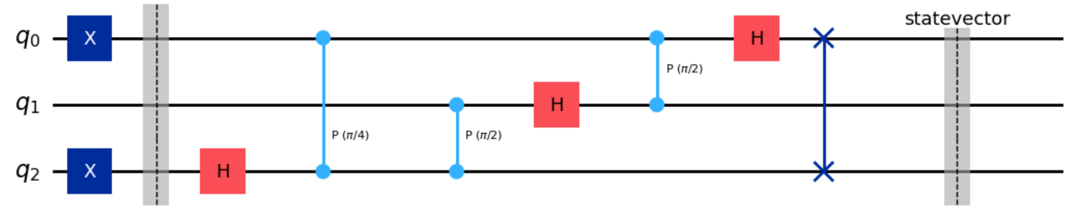
UMass Amherst | College of Engineering

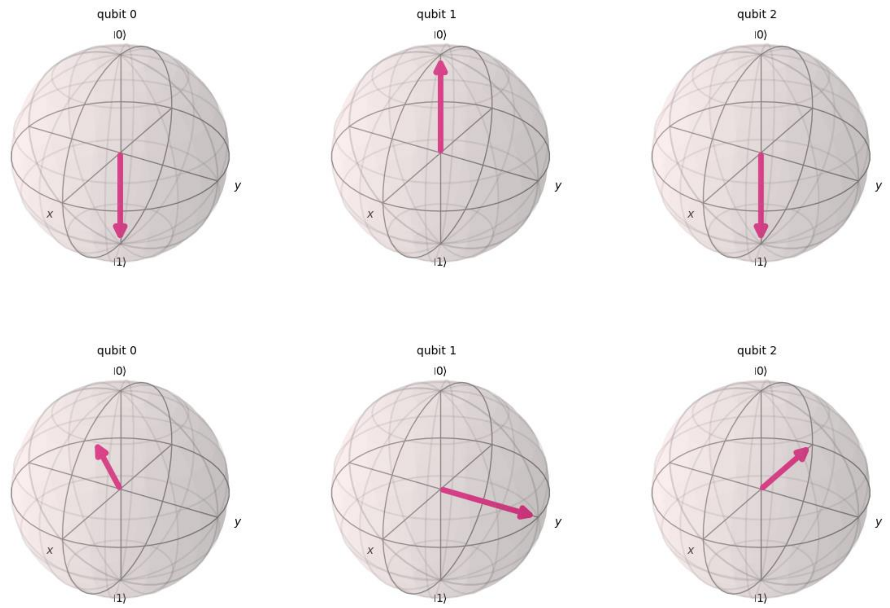
Quantum Fourier Transform = BIT → PHASE


Bit = 0 represented in Z "bit" Basis
qubit 0
qubit 1
qubit 2
qubit 3
qubit 4
qubit 5
|0⟩
|1⟩
x
y
Bit = 0 represented in Phase Space

UMassAmherst College of Engineering

QFT - breakdown





Qubit 2:

- **Starts in |0⟩ then the X gate flips it to |1⟩**
- **Hadamard transforms |1⟩ to |-⟩**
- **Control phase gate with q1 does nothing**
- **Control phase gate with q0 rotates by π/4**

Qubit 1:

- **Starts in |0⟩**
- **Hadamard transforms it from |0⟩ to |+⟩**
- **Control phase gate with q0 rotates π/2**

Qubit 0:

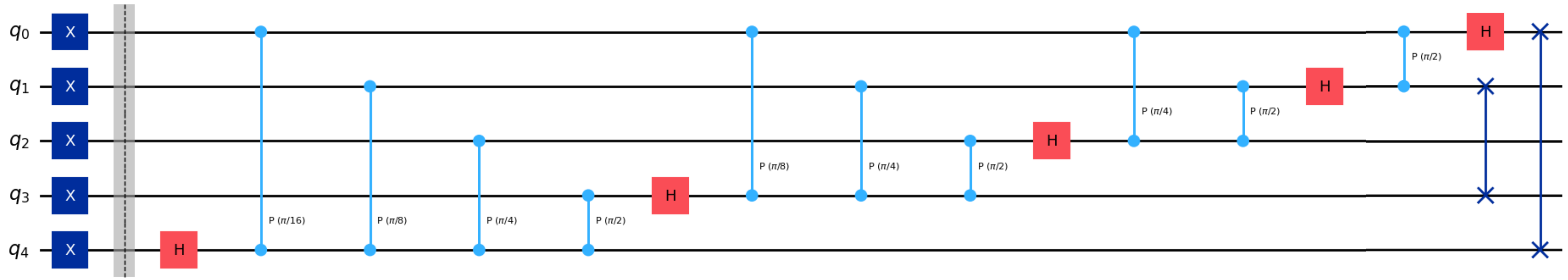
- **Starts in |0⟩ then the X gate flips it to |1⟩**
- **Hadamard transforms |1⟩ to |-⟩**

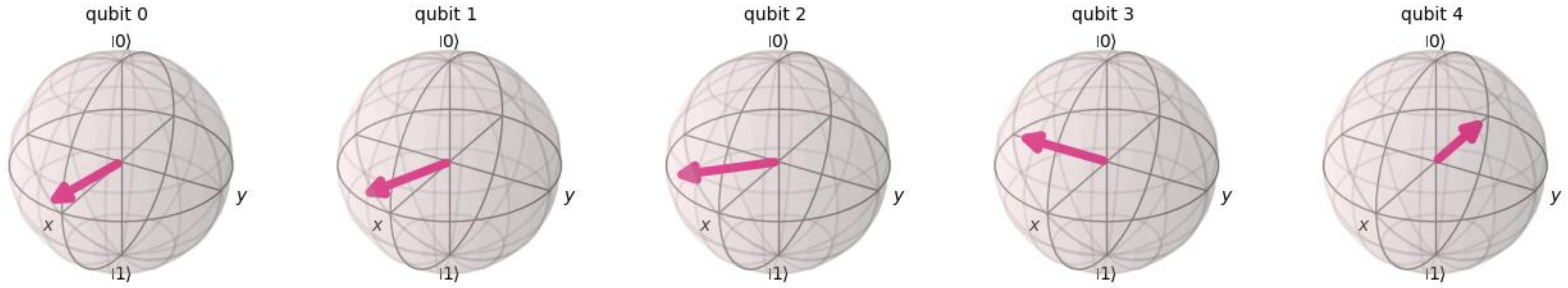
Swap:

- **Swap of qubit 0 and qubit 2**

UMassAmherst College of Engineering

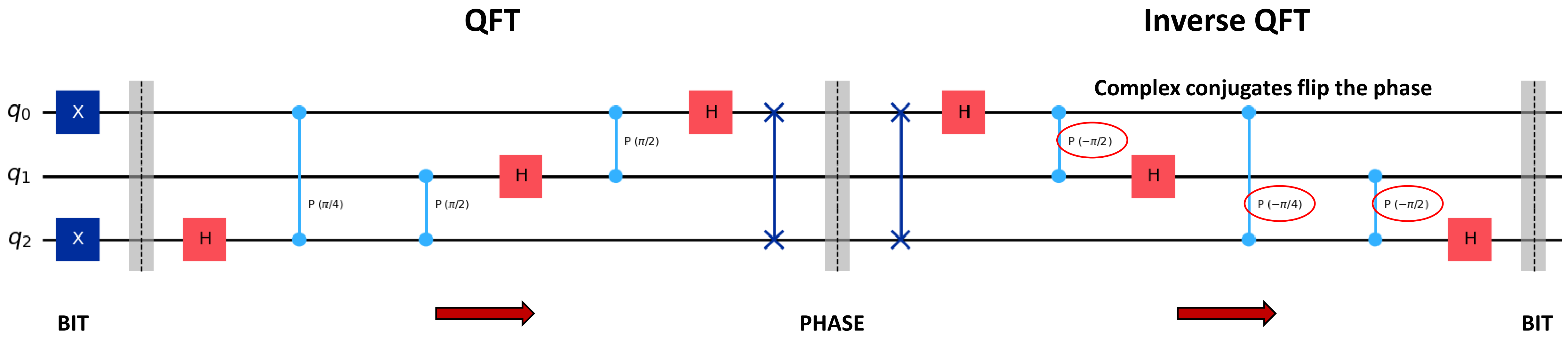
Larger register QFT



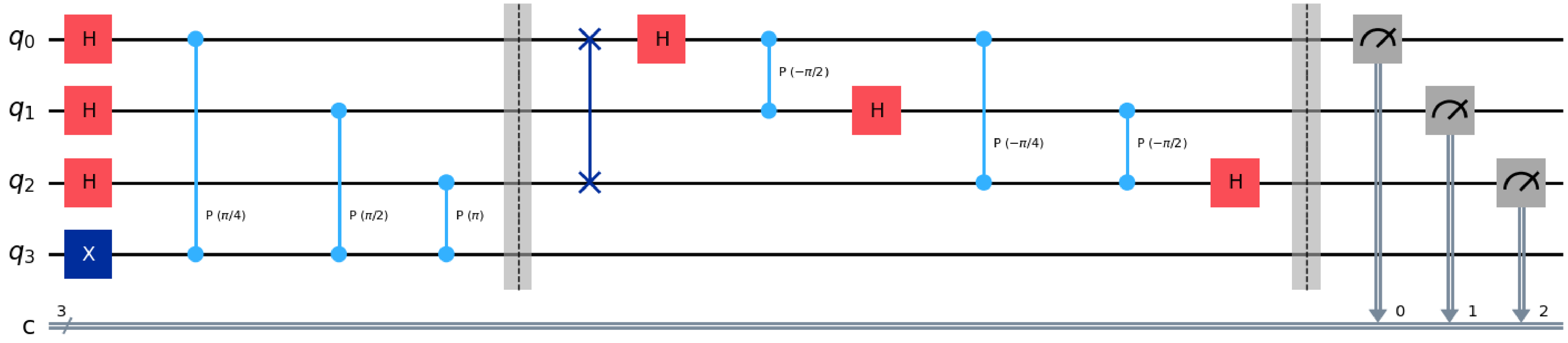


UMassAmherst | College of Engineering

Inverse QFT


QFT
Inverse QFT
Complex conjugates flip the phase
q_0
q_1
q_2
X
X
H
P (π/4)
P (π/2)
H
P (π/2)
H
H
P (−π/2)
H
P (−π/4)
P (−π/2)
H
BIT
PHASE
BIT

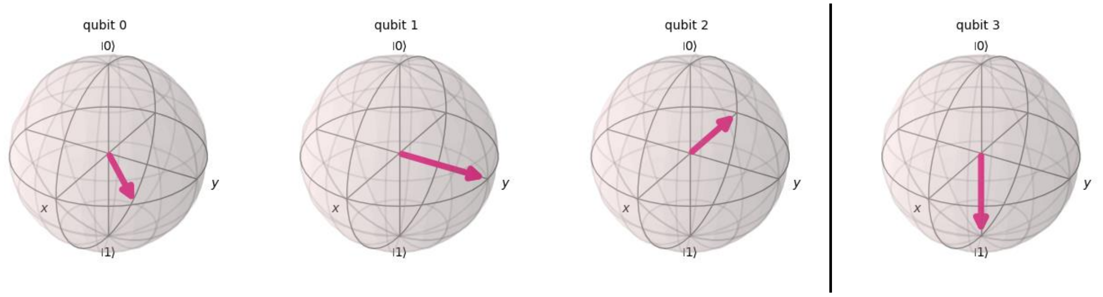
Quantum Phase Estimation - 1/8

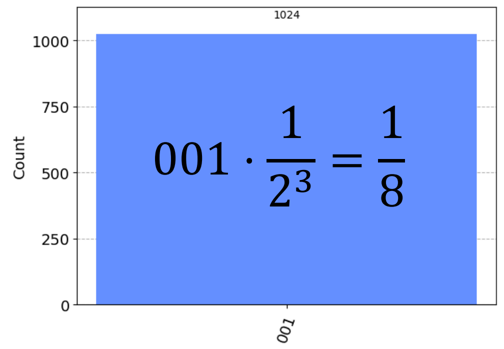


q_0 phase $= 2\pi \cdot 2^0 \cdot \frac{1}{8}$

q_1 phase $= 2\pi \cdot 2^1 \cdot \frac{1}{8}$

q_2 phase $= 2\pi \cdot 2^2 \cdot \frac{1}{8}$

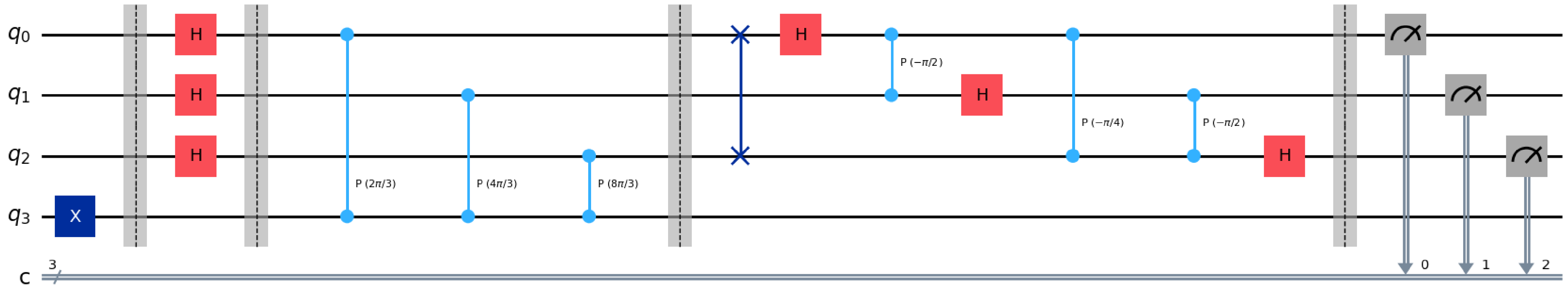




$$001 \cdot \frac{1}{2^3} = \frac{1}{8}$$

UMass Amherst | College of Engineering

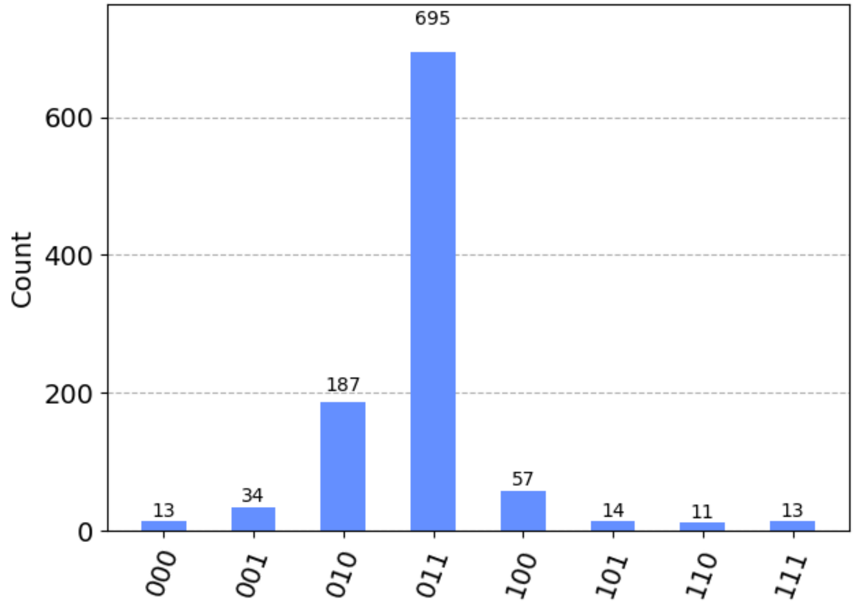
Quantum Phase Estimation – 1/3



$$q_0 \text{ phase } = 2\pi \cdot 2^0 \cdot \frac{1}{3}$$

$$q_1 \text{ phase } = 2\pi \cdot 2^1 \cdot \frac{1}{3}$$

$$q_2 \text{ phase } = 2\pi \cdot 2^2 \cdot \frac{1}{3}$$



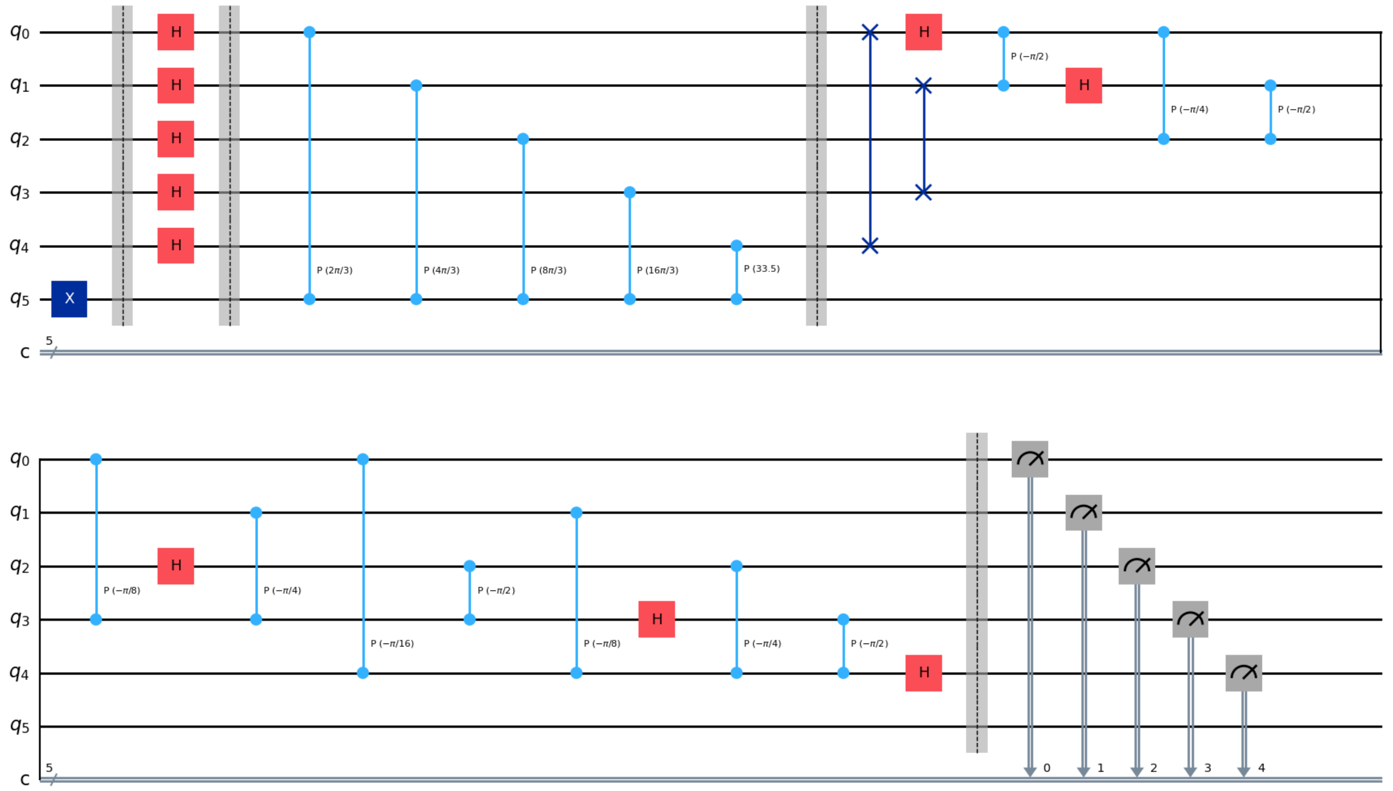
$$011 = 3\,,\ 3 \cdot \frac{1}{2^3} = \frac{3}{8} = 0.375$$

$$010 = 2\,,\ 2 \cdot \frac{1}{2^3} = \frac{2}{8} = 0.25$$

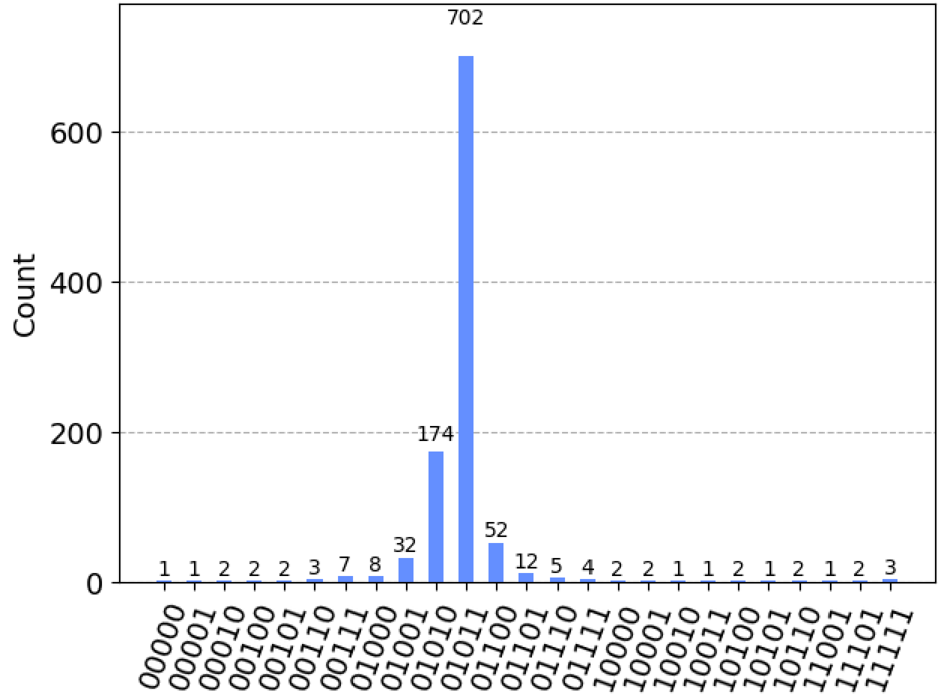
$$100 = 5\,,\ 2 \cdot \frac{1}{2^3} = \frac{5}{8} = 0.625$$

$$(0.187) \cdot \mathbf{0.25} + \\ (0.695) \cdot \mathbf{0.375} + \\ (0.057) \cdot \mathbf{0.625} = \mathbf{0.335}$$

QPE – larger register → more precision



$$01011 = 11\,,\ \ 11 \cdot \frac{1}{2^5} = \frac{11}{32} = 0.34375$$



UMassAmherst | College of Engineering

RSA Encryption (review)

RSA encryption is based on the difficulty of factoring large numbers that are the product of prime numbers.

The mathematical root is based on the observation that with integers e, d, and n:

- $(m^e)^d = m(mod\ n)$

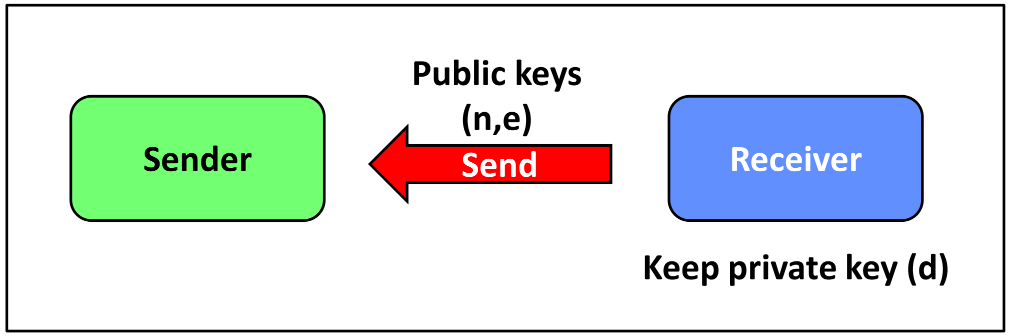
That is, modular exponentiation for all integers m to the power 'e' and 'd' is equal to m (mod n).

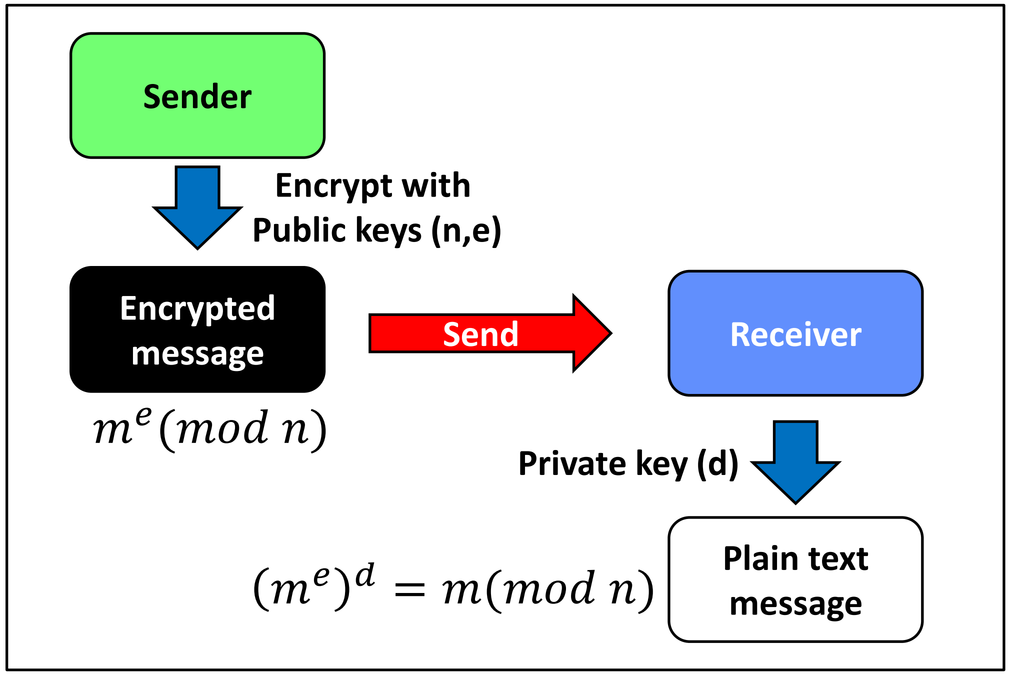
- https://en.wikipedia.org/wiki/Modular_exponentiation

Therefore, if we want to encrypt 'm' we exponentiate by a number 'e'.

- e can be public!
- Then upon exponentiation by 'd' we recover m! [(mod n) of course]

How is this possible? And how to get 'e' and 'd'?





UMassAmherst College of Engineering

RSA Encryption (review)

We require the ability to receive secret messages from a sender without any prior exchanging of secret information (only public keys).

Use primes (p & q) to generate, and then publicize, public keys (e , n):

$$n = p \cdot q$$

PUBLIC KEY = e

1. Less than **n** $(e < n)$
2. Coprime with:

$$\phi \text{ : Euler's totient function}$$

$$\phi = (p-1)(q-1)$$

(Which requires that: $\gcd(\phi, e) = 1$)

1. Start by choosing two 'secret' prime numbers:

$$p = 5 \ , q = 11$$

2. Public Key 'n': $n = p \cdot q \ = \ 55$
(using p and q to generate 'n' for the modulo)

$$(m^e)^d = m(mod\ n) = m(mod\ p \cdot q)$$

Which will be true if: $e \cdot d \equiv 1(mod\ p \cdot q)$

3. For e we first need Euler's totient function ϕ:

$$\phi = (p-1)(q-1) = 4 \cdot 10 = 40$$

4. Try different 'e' keys and check if they are coprime with ϕ :

$$e = 7$$

✓ (Factors of ϕ=40: 1, 2, 4, 5, 8, 10, 20, 40) → none with e=7 except 1

RSA Encryption (review)

1. Start by choosing two 'secret' prime numbers:

$$p = 5 \ , q = 11$$

2. Public Key 'n': $n = p \cdot q = 55$
(using p and q to generate 'n' for the modulo)

$$(m^e)^d = m(mod\ n) = m(mod\ p \cdot q)$$

Which will be true if: $e \cdot d \equiv 1(mod\ p \cdot q)$

3. For e we first need Euler's totient function ϕ:

$$\phi = (p-1)(q-1) = 4 \cdot 10 = 40$$

4. Try different 'e' keys and check if they are coprime with ϕ :

$$e = 7$$

✓ (Factors of ϕ=40: 1, 2, 4, 5, 8, 10, 20, 40) → none with e=7 except 1

To check if: $m^{ed}(mod\ p \cdot q) = m(mod\ p \cdot q)$

it is equivalent to check that they are congruent to mod p and mod q separately:

$m^{ed}(mod\ p) = m(mod\ p)$

$m^{ed} = m^{ed-1}m = m^{h(p-1)}m = (m^{p-1})^h m = ?$

Fermat's little theorem:

If p is prime, then $(a^p - a)$ is a multiple of p,

meaning: $a^p(mod\ p) = a(mod\ p)$

So, if we divide by 'a' then:

$$a^{p-1}(mod\ p) = 1$$

Therefore:

$$(m^{p-1})^h \cdot m(mod\ p) = (1)^h \cdot m(mod\ p)$$

And so:

$$m^{ed}(mod\ p) = m(mod\ p)$$

d , private key

We need ‘d’ such that: $e \cdot d \equiv 1 \, mod \, \phi$

We’ll use the Extended Euclidean algorithm to check $gcdEE \ (\phi, e)$, even though we know it is ‘1’ because it simultaneously calculates:

$$gcdEE(a, b) = a \cdot x + b \cdot y$$

- For: $gcdEE(e, \phi) = e \cdot x + \phi \cdot y$
- We already know is equal to 1, so:

$$e \cdot x + \phi \cdot y = 1$$

- Modulo of ϕ, removes the factor $\phi \cdot y$ (which mod ϕ is zero), so:

$$e \cdot x = 1 \ (mod \ \phi)$$

Therefore:

$$e \cdot d \equiv 1 \, mod \, \phi \ \rightarrow x = d$$

Conversely:

If we know ‘p’ and ‘q’ we can calculate ‘d’

Goal for cracking will be to factor n

$$n = p \cdot q$$

to get ‘p’ and ‘q’

to then calculate ‘d’

RSA decryption - time

Conversely:

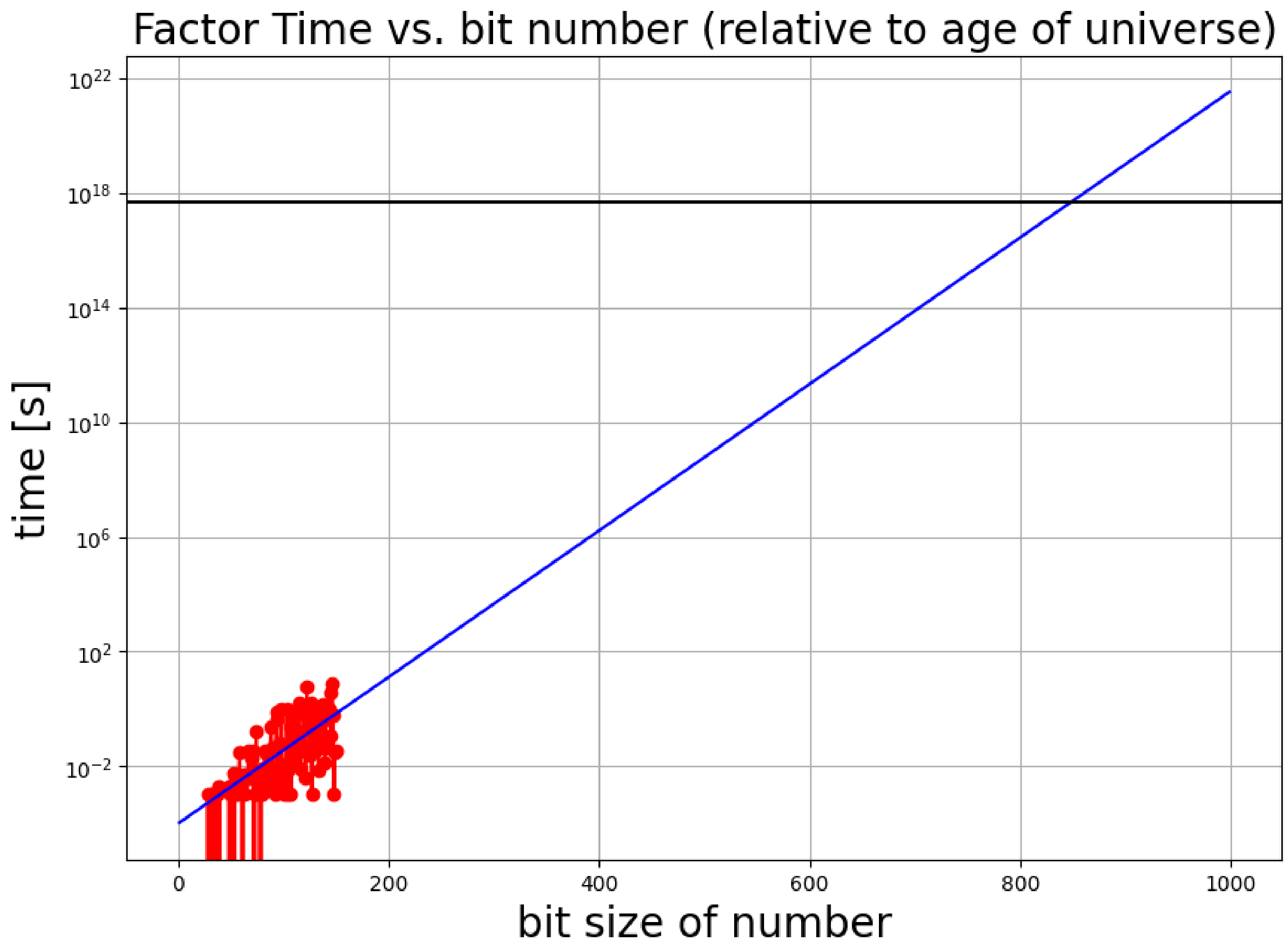
If we know ‘p’ and ‘q’ we can calculate ‘d’

Goal for cracking will be to factor n

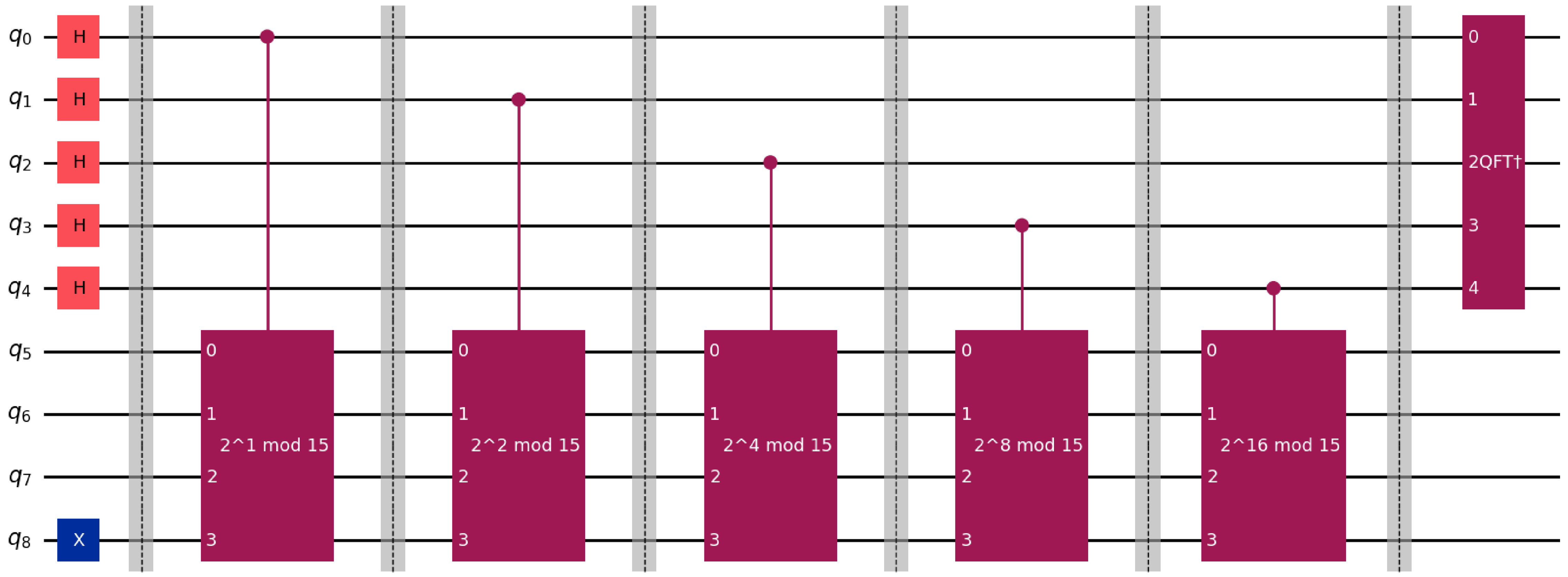
$$n = p \cdot q$$

to get ‘p’ and ‘q’

to then calculate ‘d’



Shor’s Algorithm



UMass Amherst | College of Engineering