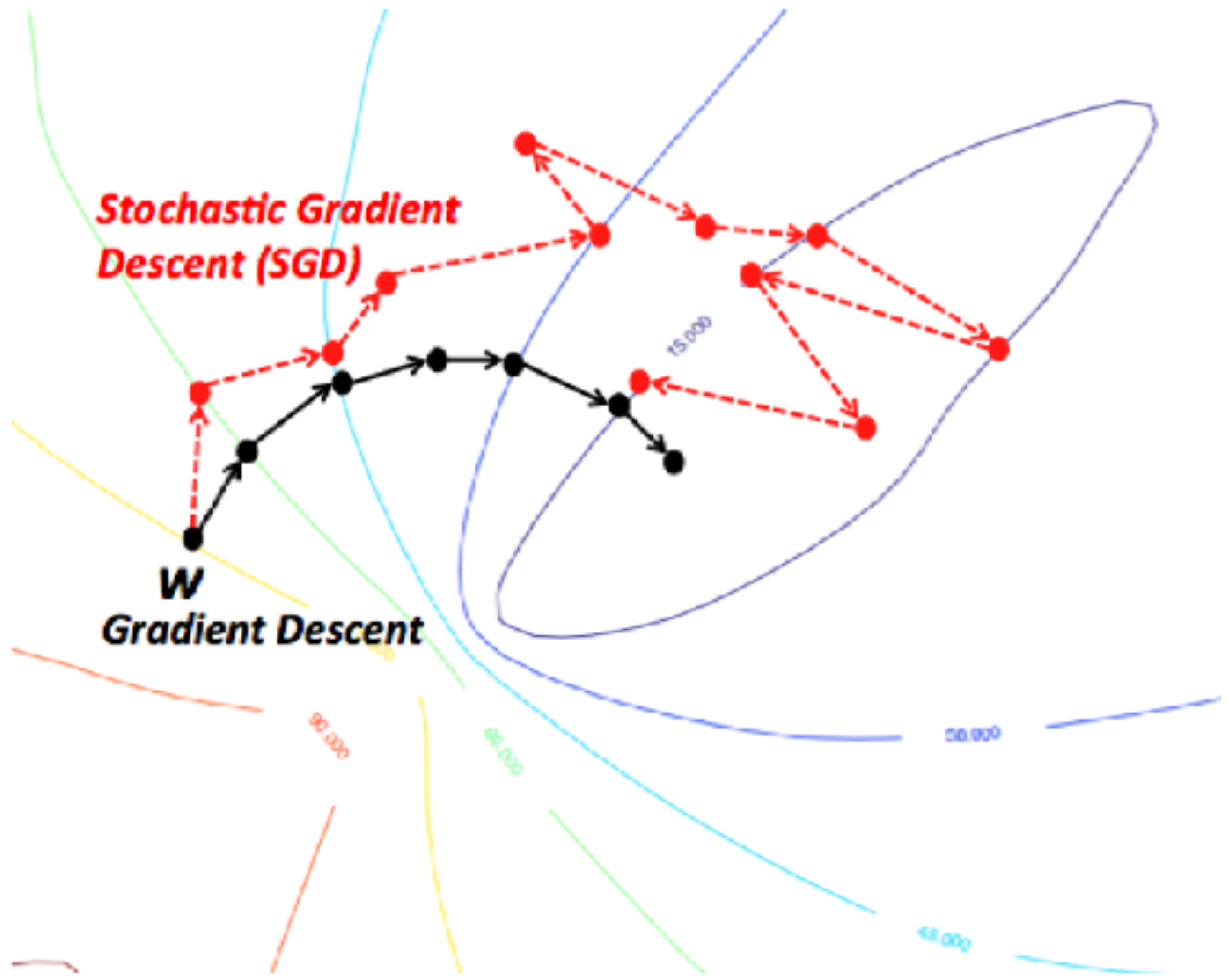
Estimating gradients using quantum computation

Rigetti quantum computing meetup
November 2, 2017

Keri A. McKiernan
kmckiern.github.io

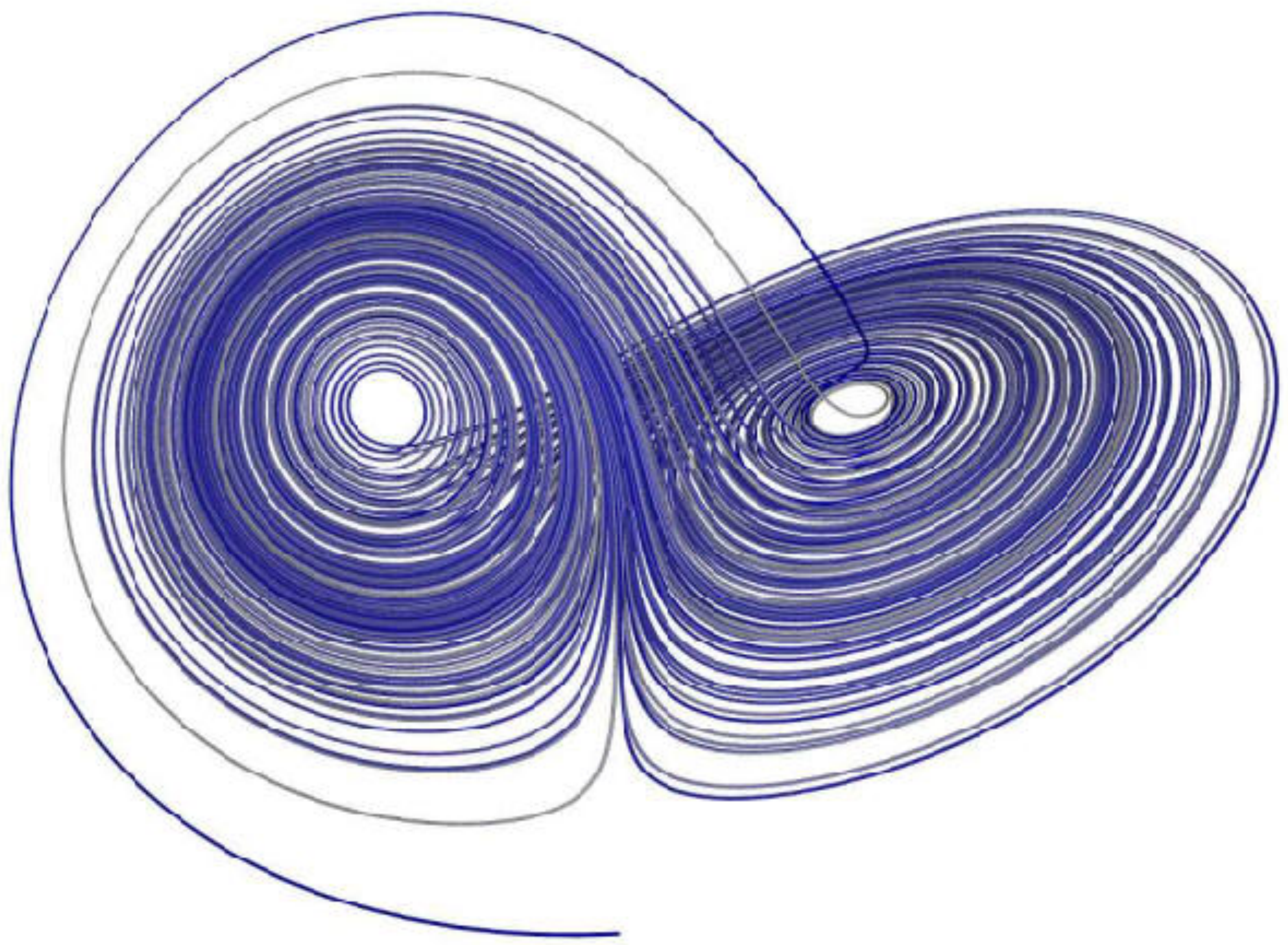
Gradients are fundamental to research and engineering applications

Optimization problems



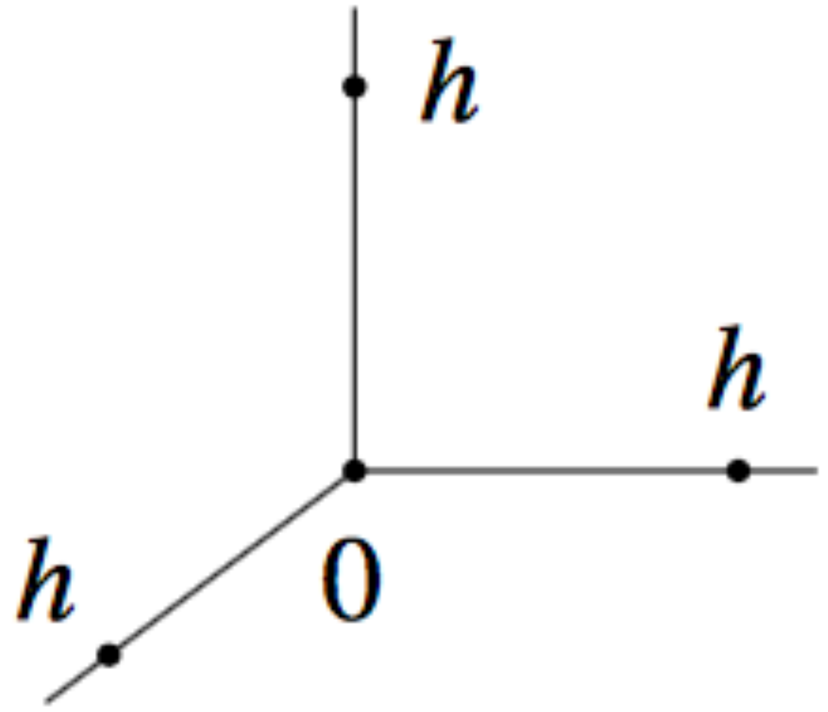
https://wikidocs.net/3413

Dynamical systems



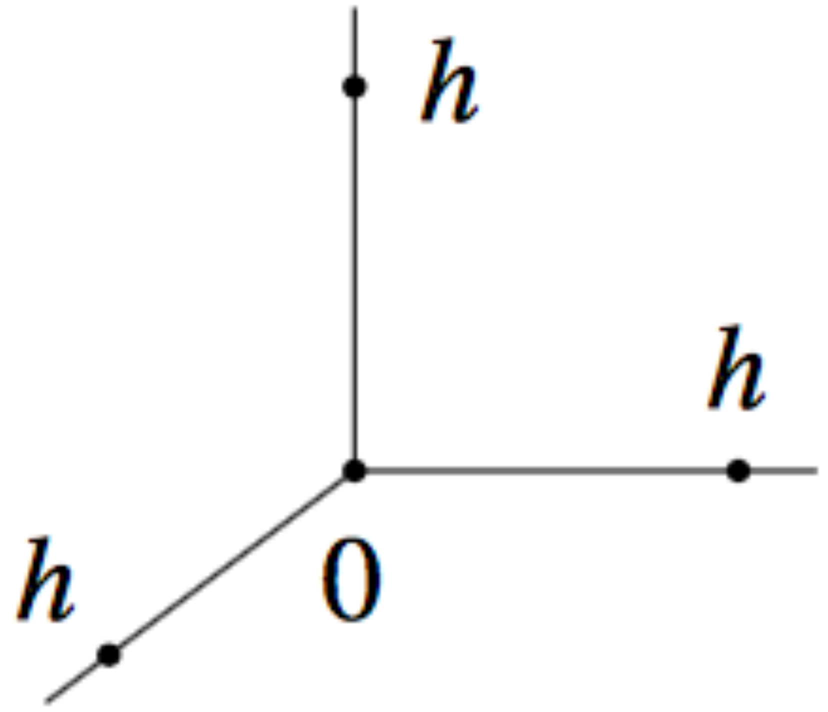
https://www.math.uci.edu/~asgor/dynsys/

Computing numerical gradient estimates



10.1063/1.3266959

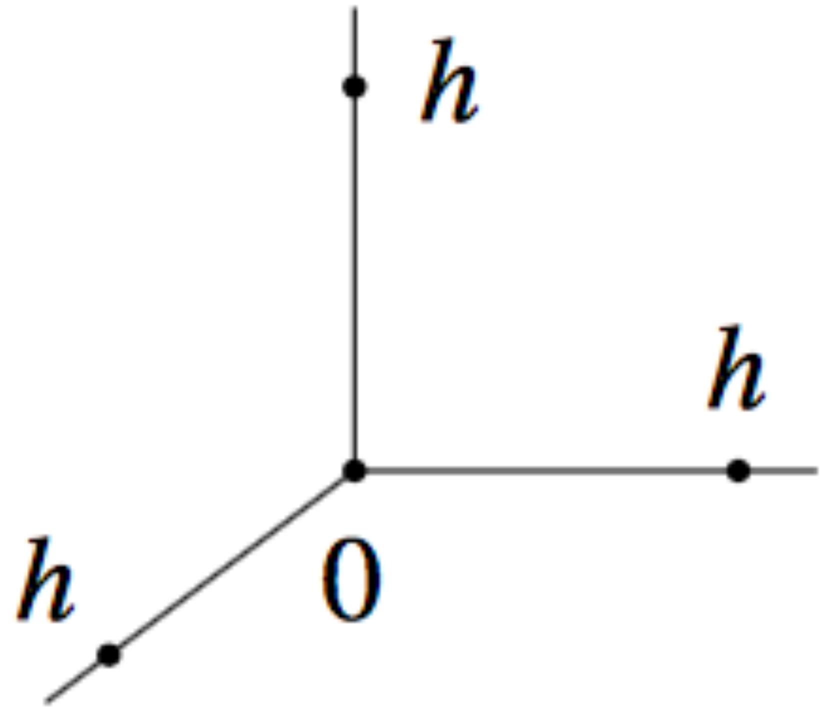
Computing numerical gradient estimates



$$E(h) \approx E(0) + h\nabla E(0)$$

10.1063/1.3266959

Computing numerical gradient estimates

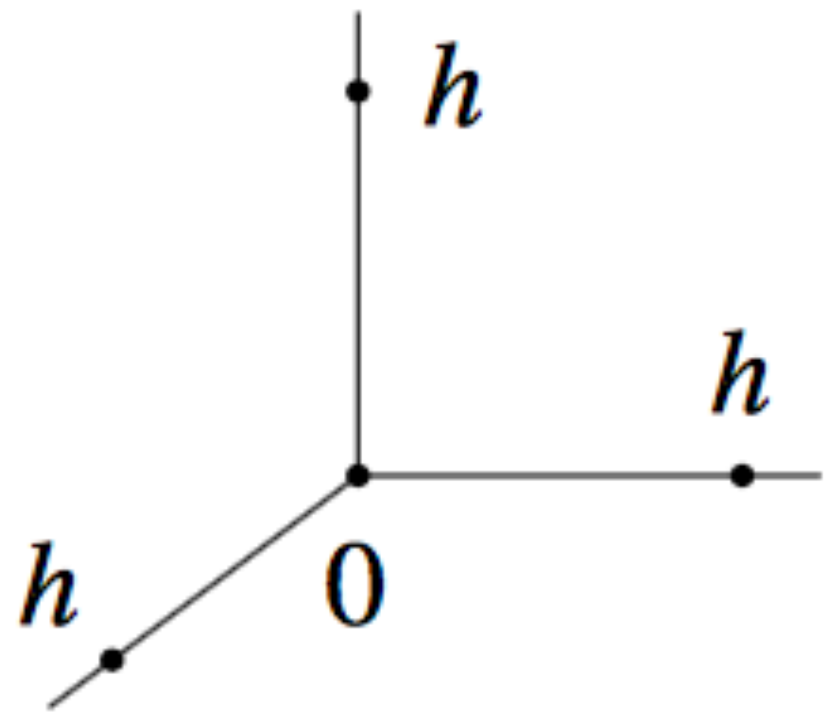


$$E(h) \approx E(0) + h\nabla E(0)$$

$$\nabla E(0) \approx \frac{E(0) - E(h)}{h}$$

10.1063/1.3266959

Numerical gradient estimation is fast on quantum architecture



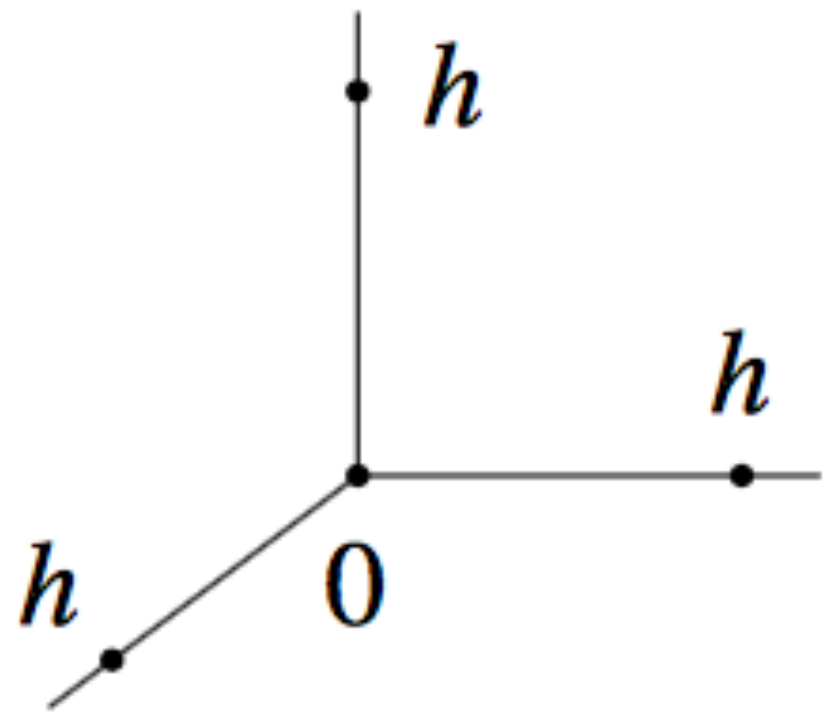
$$E(h) \approx E(0) + h\nabla E(0)$$

$$\nabla E(0) \approx \frac{E(0) - E(h)}{h}$$

	Classical		Quantum
Derivative	Numerical	Analytical	Numerical
$\frac{\mathrm{d}E}{\mathrm{d}\mu}$	$d+1$	$O(1)$	1
$\frac{\mathrm{d}^2E}{\mathrm{d}\mu^2}$	d^2+1	$O(d)$	2
$\frac{\mathrm{d}^3E}{\mathrm{d}\mu^3}$	d^3+1	$O(d)$	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{\mathrm{d}^nE}{\mathrm{d}\mu^n}$	d^n+1	$O\left(d^{\lfloor n/2 \rfloor}\right)$	2^{n-1}

10.1063/1.3266959

Numerical gradient estimation is fast on quantum architecture



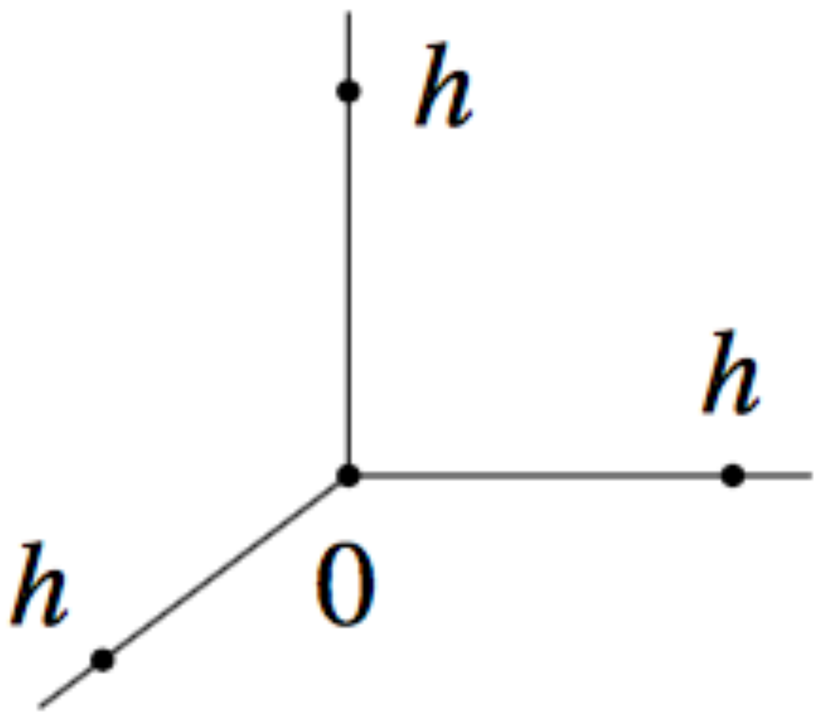
$$E(h) \approx E(0) + h\nabla E(0)$$

$$\nabla E(0) \approx \frac{E(0) - E(h)}{h}$$

	Classical		Quantum
Derivative	Numerical	Analytical	Numerical
$\frac{\mathrm{d}E}{\mathrm{d}\mu}$	$d+1$	$O(1)$	1
$\frac{\mathrm{d}^2E}{\mathrm{d}\mu^2}$	d^2+1	$O(d)$	2
$\frac{\mathrm{d}^3E}{\mathrm{d}\mu^3}$	d^3+1	$O(d)$	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{\mathrm{d}^nE}{\mathrm{d}\mu^n}$	d^n+1	$O\left(d^{\lfloor n/2 \rfloor}\right)$	2^{n-1}

10.1063/1.3266959

Numerical gradient estimation is fast on quantum architecture



$$E(h) \approx E(0) + h\nabla E(0)$$

$$\nabla E(0) \approx \frac{E(0) - E(h)}{h}$$

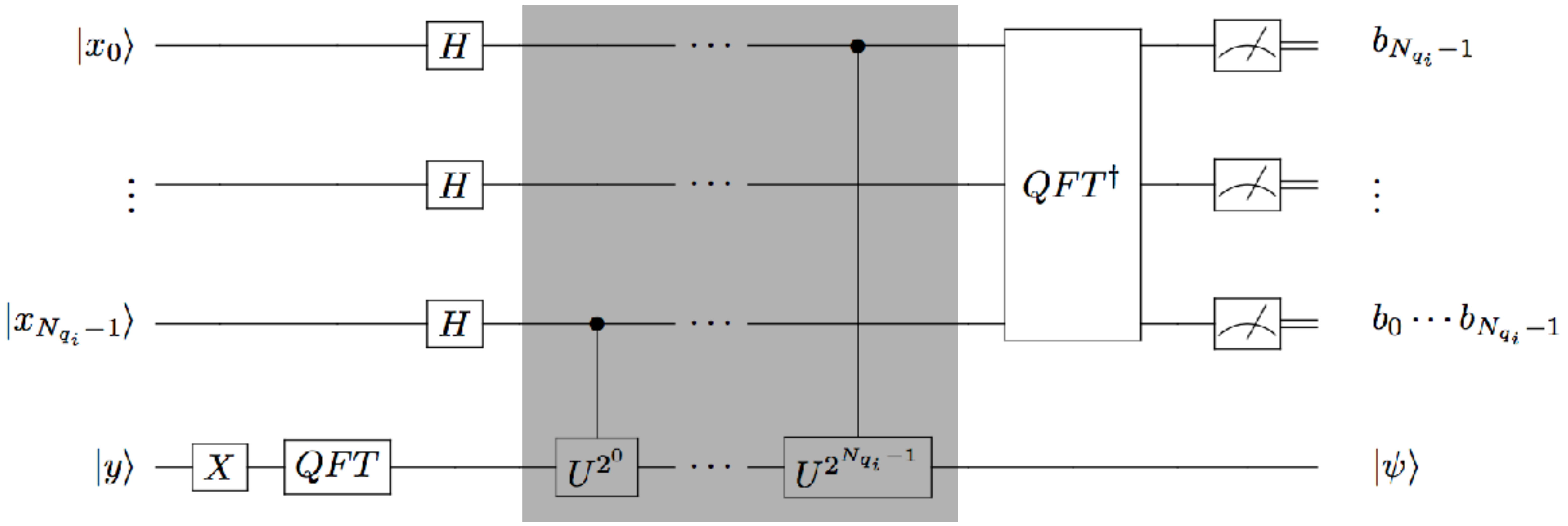
	Classical		Quantum
Derivative	Numerical	Analytical	Numerical
$\frac{\mathrm{d}E}{\mathrm{d}\mu}$	$d+1$	$O(1)$	1
$\frac{\mathrm{d}^2E}{\mathrm{d}\mu^2}$	d^2+1	$O(d)$	2
$\frac{\mathrm{d}^3E}{\mathrm{d}\mu^3}$	d^3+1	$O(d)$	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\frac{\mathrm{d}^nE}{\mathrm{d}\mu^n}$	d^n+1	$O\left(d^{\lfloor n/2 \rfloor}\right)$	2^{n-1}

10.1063/1.3266959

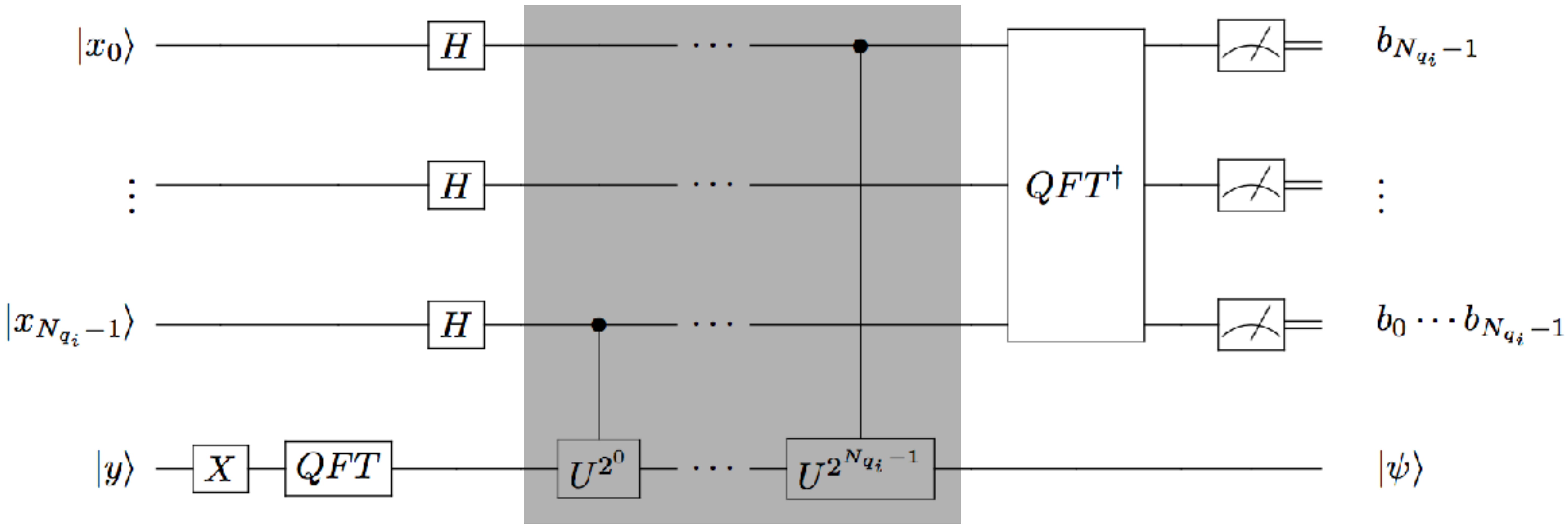
Jordan gradient estimation

$$f : \mathbb{R}^d \to \mathbb{R}$$

$$\vec{\nabla} f(\vec{x}) \approx 0.b_0 \cdots b_{n-1}$$

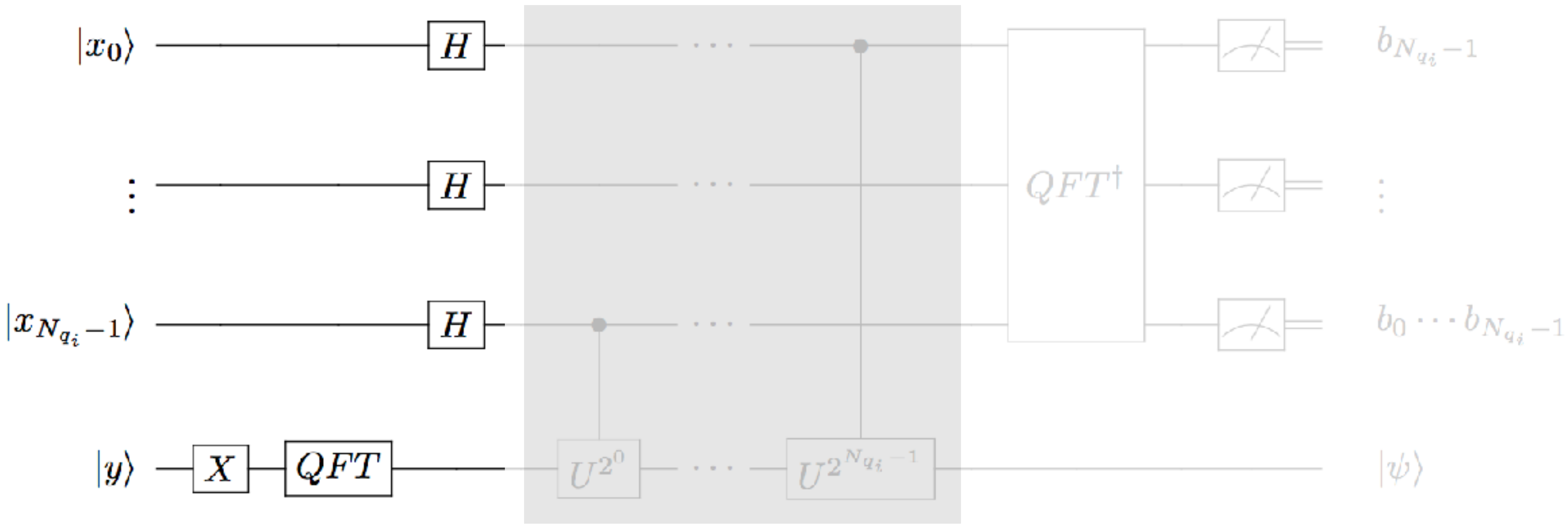


Jordan gradient estimation



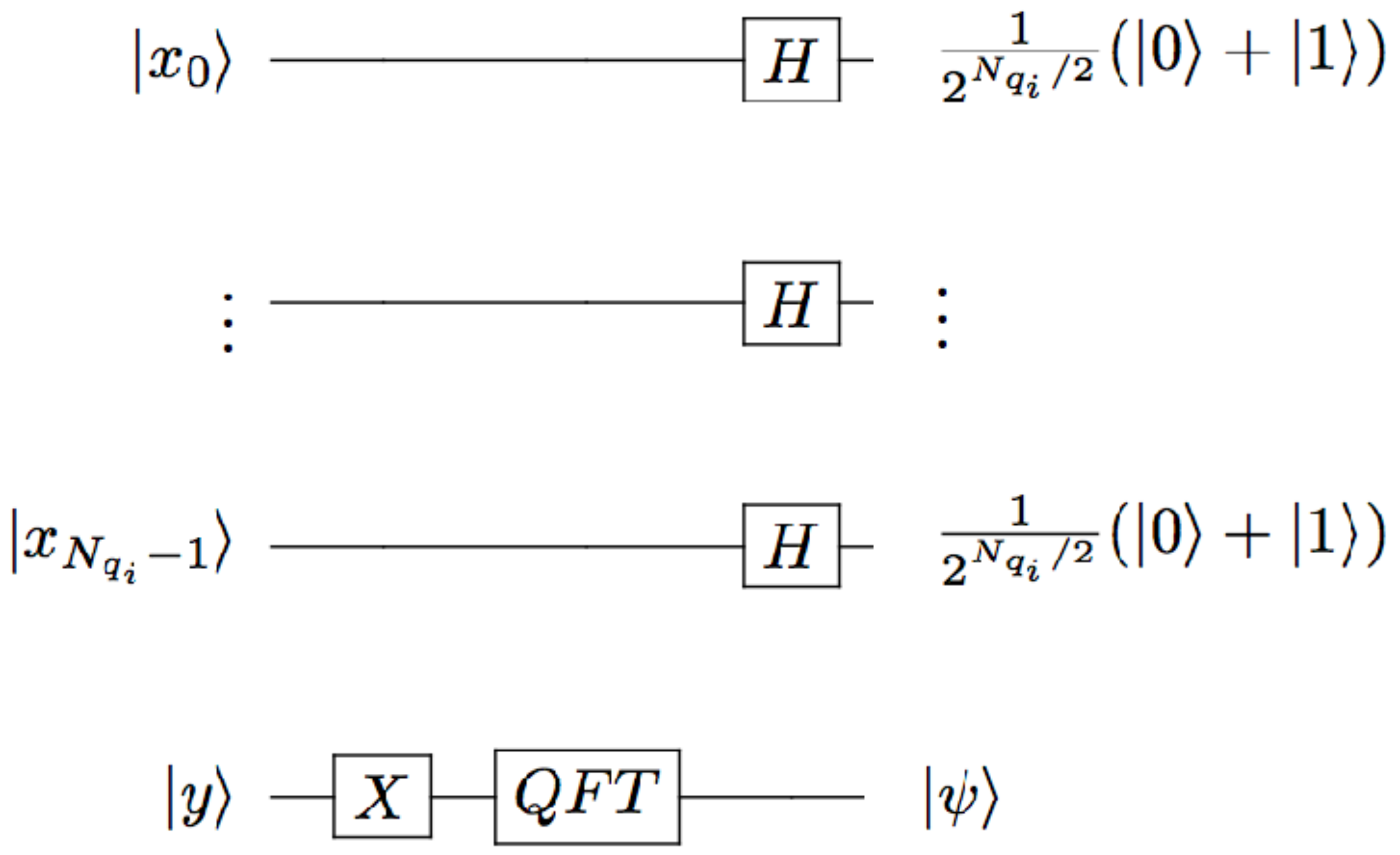
program	**ancilla qubits**	**input qubits**
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

Jordan gradient estimation



program	ancilla qubits	input qubits
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

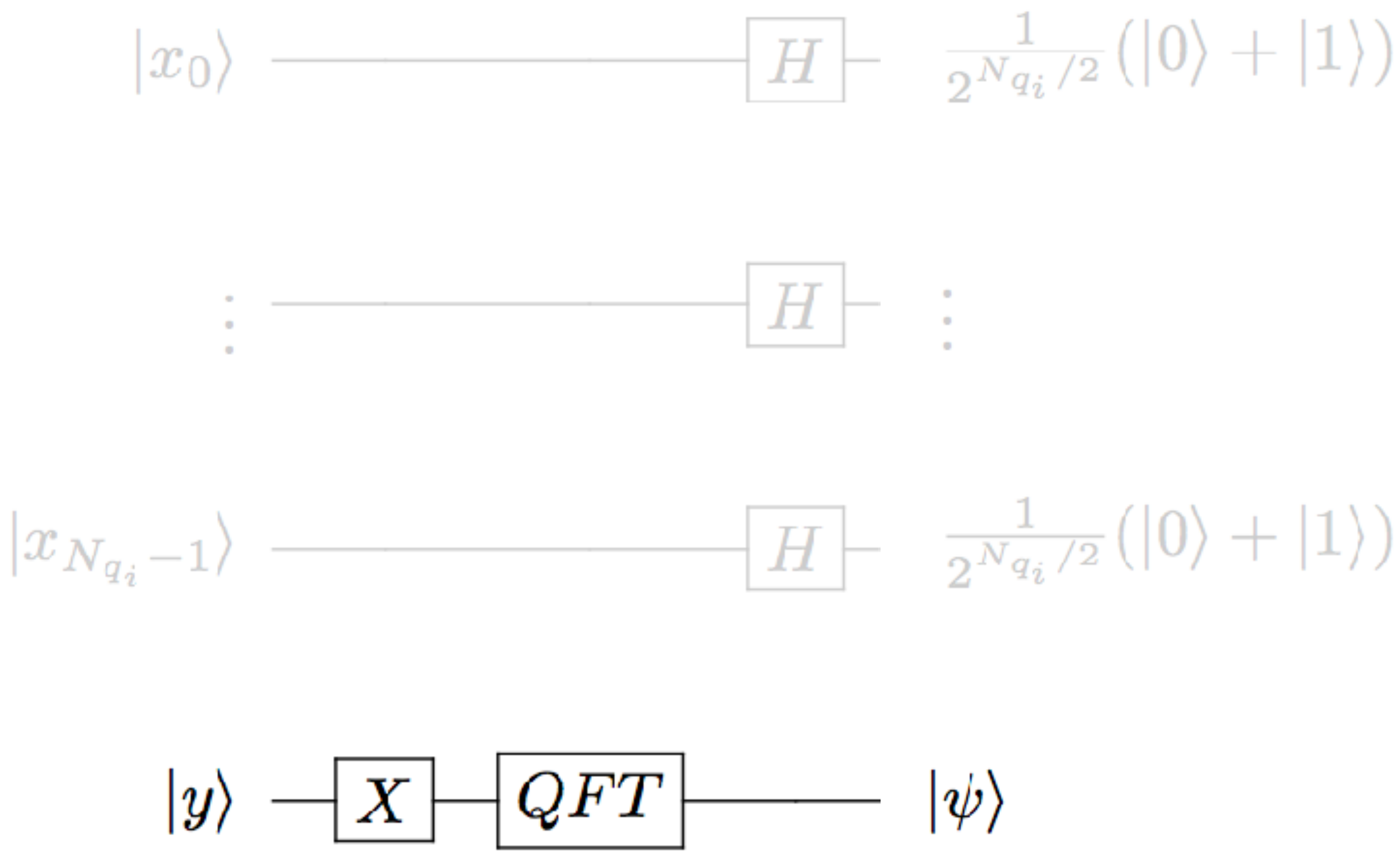
initialize system



Prepare the ancilla register to a plane wave state

$$|\psi\rangle = \frac{1}{\sqrt{2^{N_{q_o}}}} \sum_{k=0}^{2^{N_{q_o}}-1} e^{i2\pi k/2^{N_{q_o}}} |k\rangle \tag{1}$$

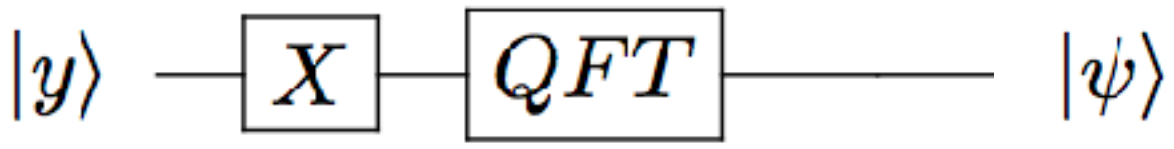
initialize system



Prepare the ancilla register to a plane wave state

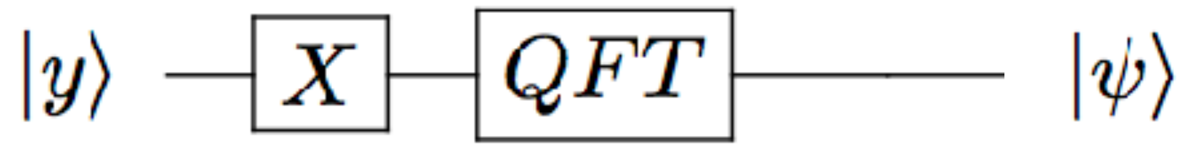
$$|\psi\rangle = \frac{1}{\sqrt{2^{N_{qo}}}} \sum_{k=0}^{2^{N_{qo}}-1} e^{i2\pi k/2^{N_{qo}}} |k\rangle \tag{1}$$

initialize system



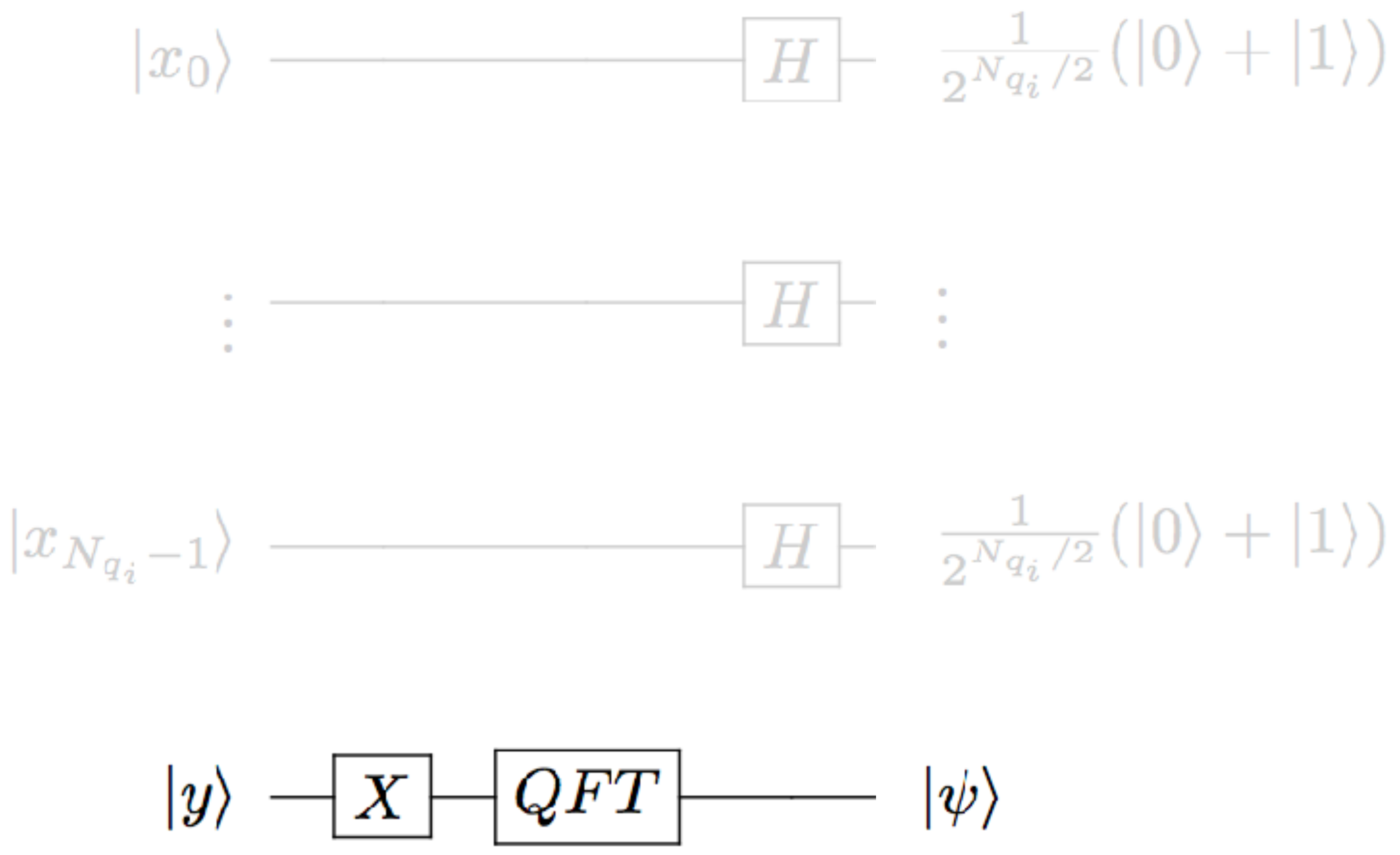

```python
from pyquil.gates import X
from grove.qft.fourier import qft
import pyquil.quil as pq

def initialize_system(input_qubits, ancilla_qubits):
    """ Prepare initial state

    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :return Program p_ic: Quil program to initialize this system.
    """

    # ancilla qubits to plane wave state
    ic_out = list(map(X, ancilla_qubits))
    ft_out = qft(ancilla_qubits)
    p_ic_out = pq.Program(ic_out) + ft_out
    # input qubits to equal superposition
    ic_in = list(map(H, input_qubits))
    p_ic_in = pq.Program(ic_in)
    # combine programs
    p_ic = p_ic_out + p_ic_in
    return p_ic
```

initialize system




```python
from pyquil.gates import X
from grove.qft.fourier import qft
import pyquil.quil as pq

def initialize_system(input_qubits, ancilla_qubits):
    """ Prepare initial state

    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :return Program p_ic: Quil program to initialize this system.
    """

    # ancilla qubits to plane wave state
    ic_out = list(map(X, ancilla_qubits))
    ft_out = qft(ancilla_qubits)
    p_ic_out = pq.Program(ic_out) + ft_out
    # input qubits to equal superposition
    ic_in = list(map(H, input_qubits))
    p_ic_in = pq.Program(ic_in)
    # combine programs
    p_ic = p_ic_out + p_ic_in
    return p_ic
```

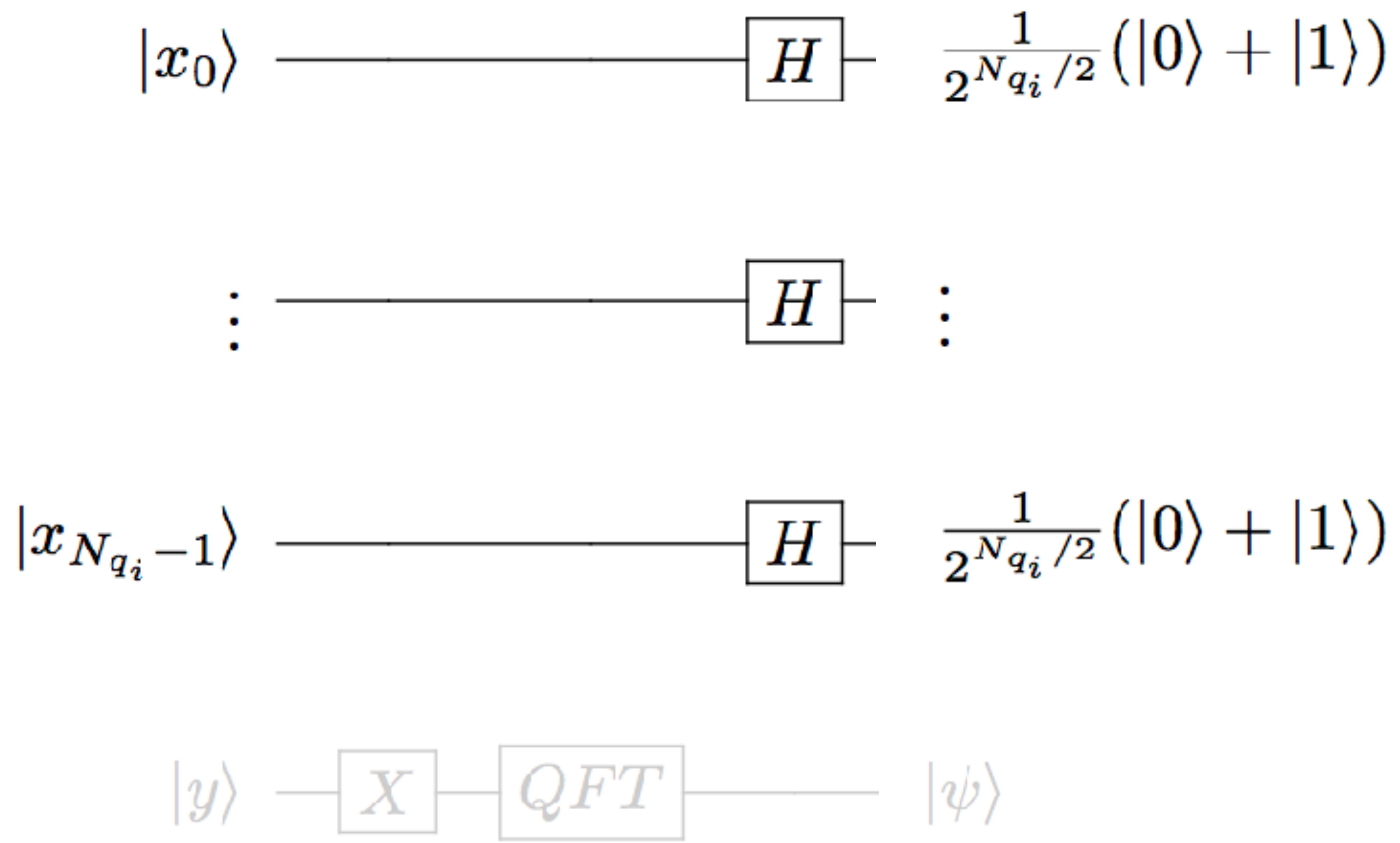
initialize system



Prepare the ancilla register to a plane wave state

$$|\psi\rangle = \frac{1}{\sqrt{2^{N_{q_o}}}} \sum_{k=0}^{2^{N_{q_o}}-1} e^{i2\pi k/2^{N_{q_o}}} |k\rangle \tag{1}$$

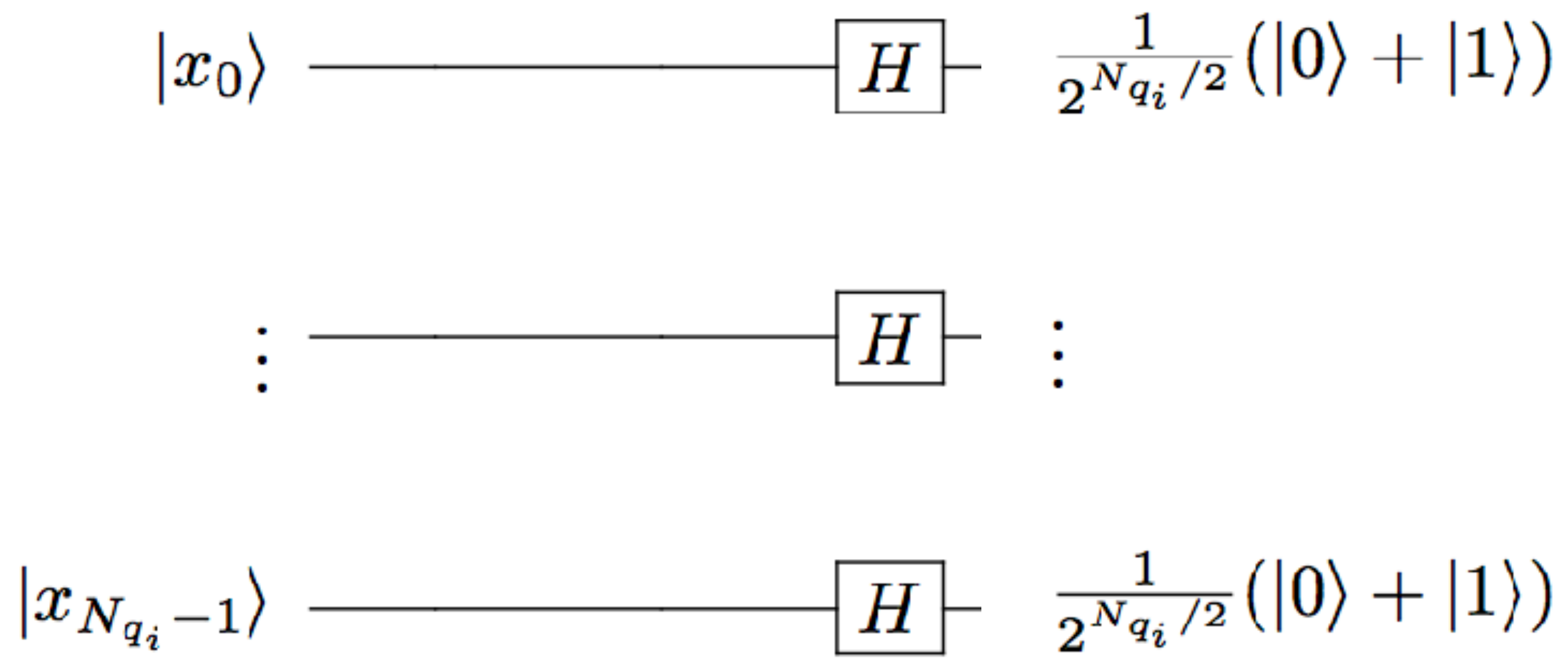
initialize system



Prepare the ancilla register to a plane wave state

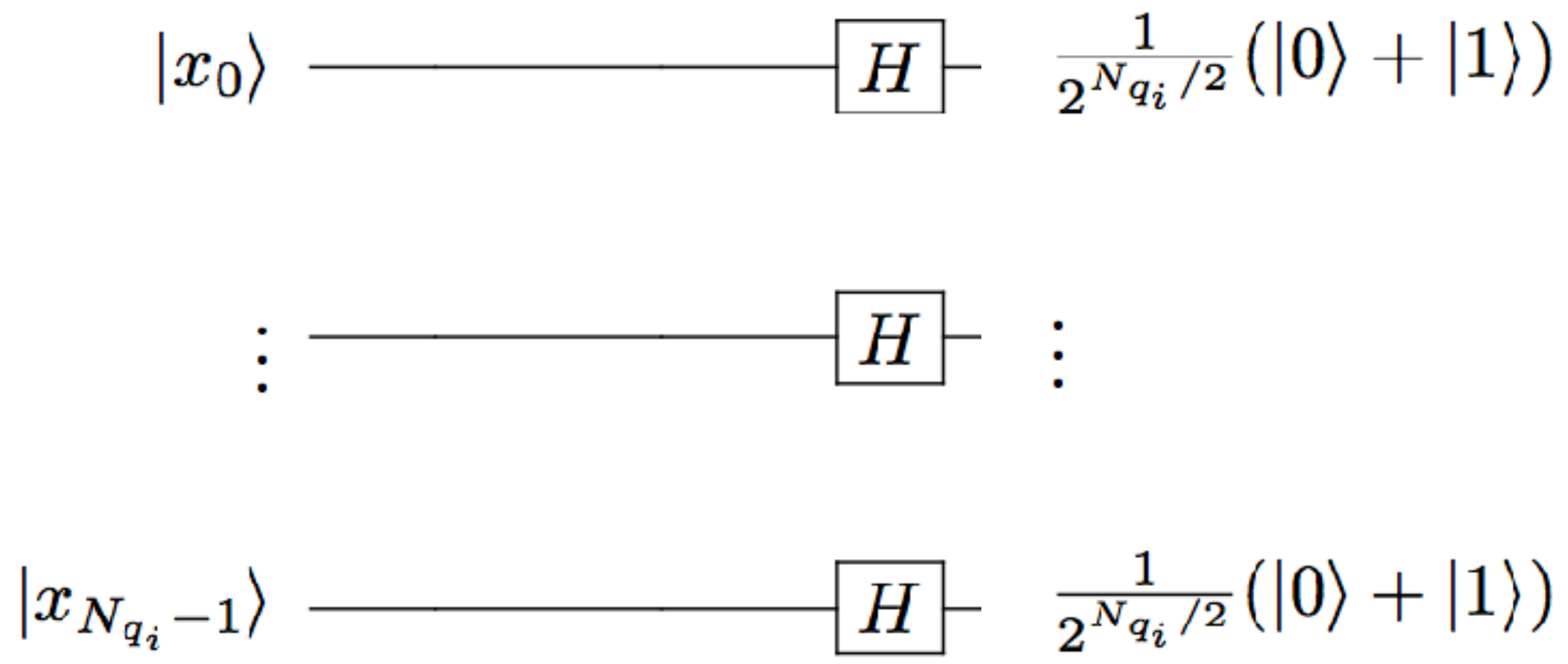
$$|\psi\rangle = \frac{1}{\sqrt{2^{N_{q_o}}}} \sum_{k=0}^{2^{N_{q_o}}-1} e^{i2\pi k/2^{N_{q_o}}} |k\rangle \tag{1}$$

initialize system




```python
from pyquil.gates import H

# ancilla qubits to plane wave state
ic_out = list(map(X, ancilla_qubits))
ft_out = qft(ancilla_qubits)
p_ic_out = pq.Program(ic_out) + ft_out
# input qubits to equal superposition
ic_in = list(map(H, input_qubits))
p_ic_in = pq.Program(ic_in)
# combine programs
p_ic = p_ic_out + p_ic_in
return p_ic
```

initialize system



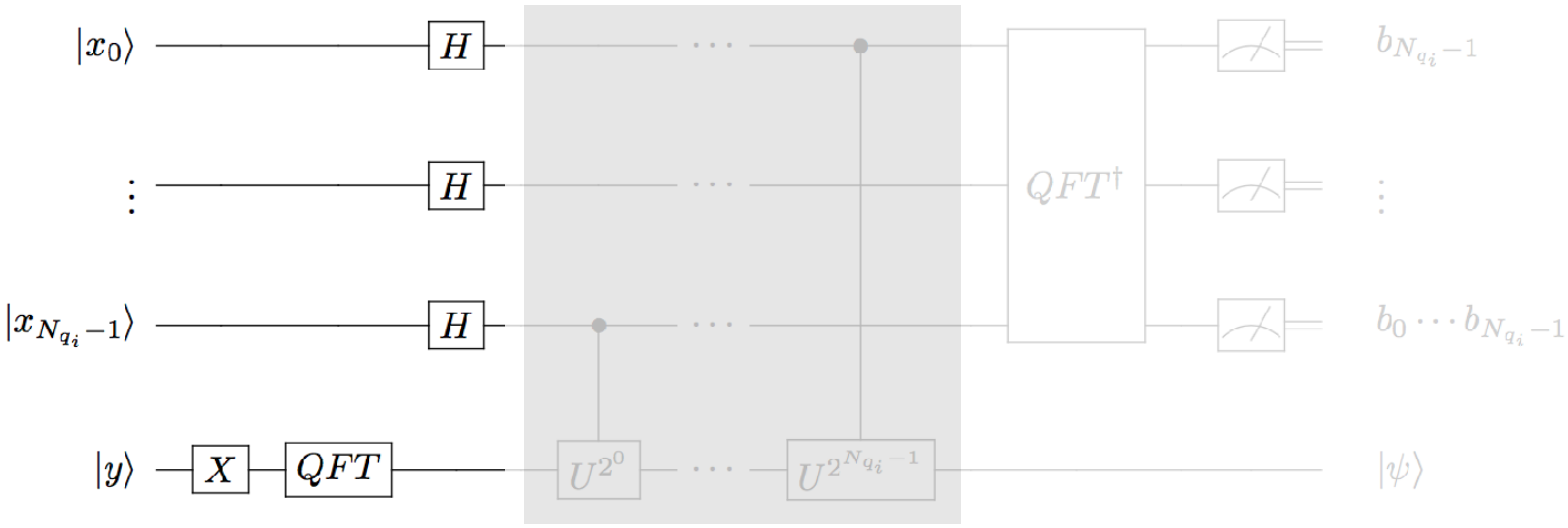

```
from pyquil.gates import H

# ancilla qubits to plane wave state
ic_out = list(map(X, ancilla_qubits))
ft_out = qft(ancilla_qubits)
p_ic_out = pq.Program(ic_out) + ft_out
# input qubits to equal superposition
ic_in = list(map(H, input_qubits))
p_ic_in = pq.Program(ic_in)
# combine programs
p_ic = p_ic_out + p_ic_in
return p_ic
```

initialize system

program	ancilla qubits	input qubits
initialize system	QFT, X	H

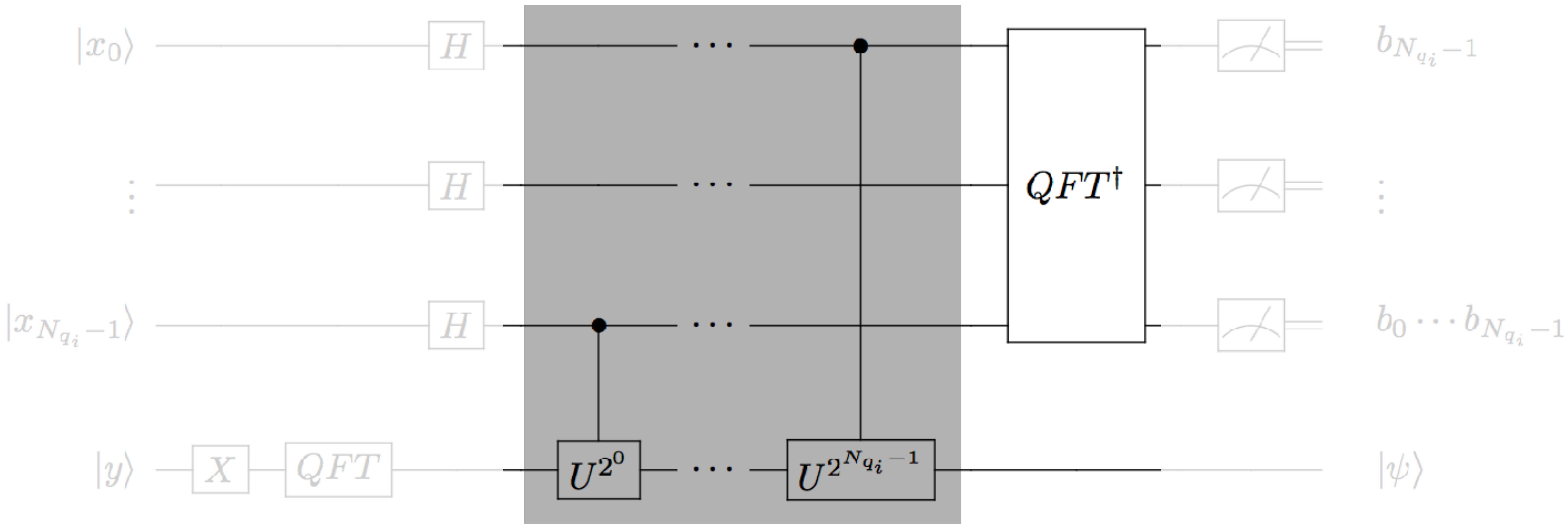
```python
def initialize_system(input_qubits, ancilla_qubits):
    """ Prepare initial state

    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :return Program p_ic: Quil program to initialize this system.
    """

    # ancilla qubits to plane wave state
    ic_out = list(map(X, ancilla_qubits))
    ft_out = qft(ancilla_qubits)
    p_ic_out = pq.Program(ic_out) + ft_out
    # input qubits to equal superposition
    ic_in = list(map(H, input_qubits))
    p_ic_in = pq.Program(ic_in)
    # combine programs
    p_ic = p_ic_out + p_ic_in
    return p_ic
```

Jordan gradient estimation



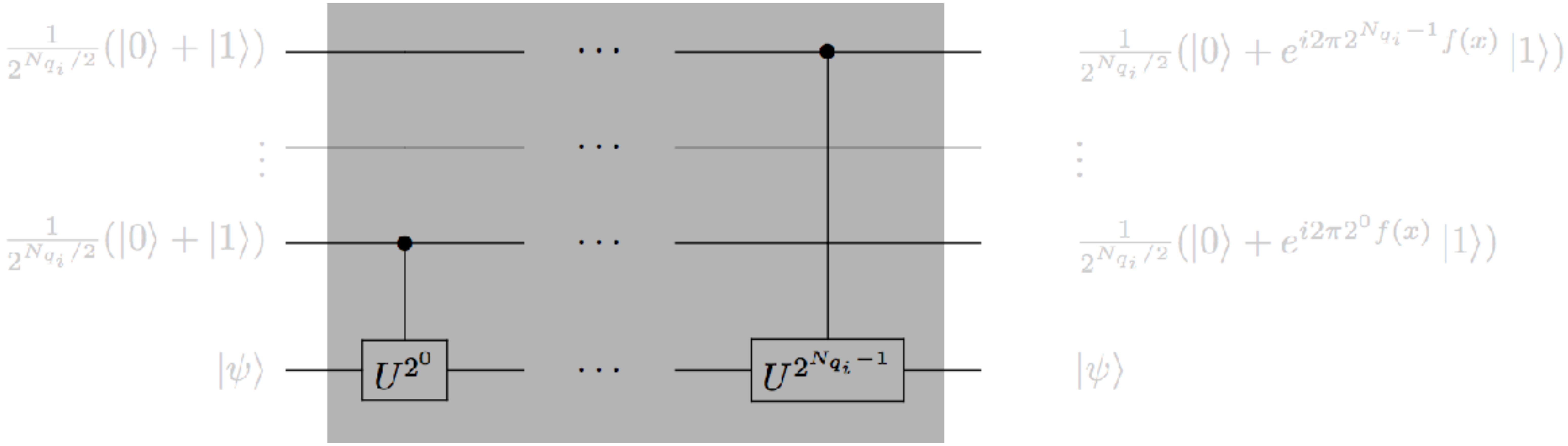
program	ancilla qubits	input qubits
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

Jordan gradient estimation



program	ancilla qubits	input qubits
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

phase kickback



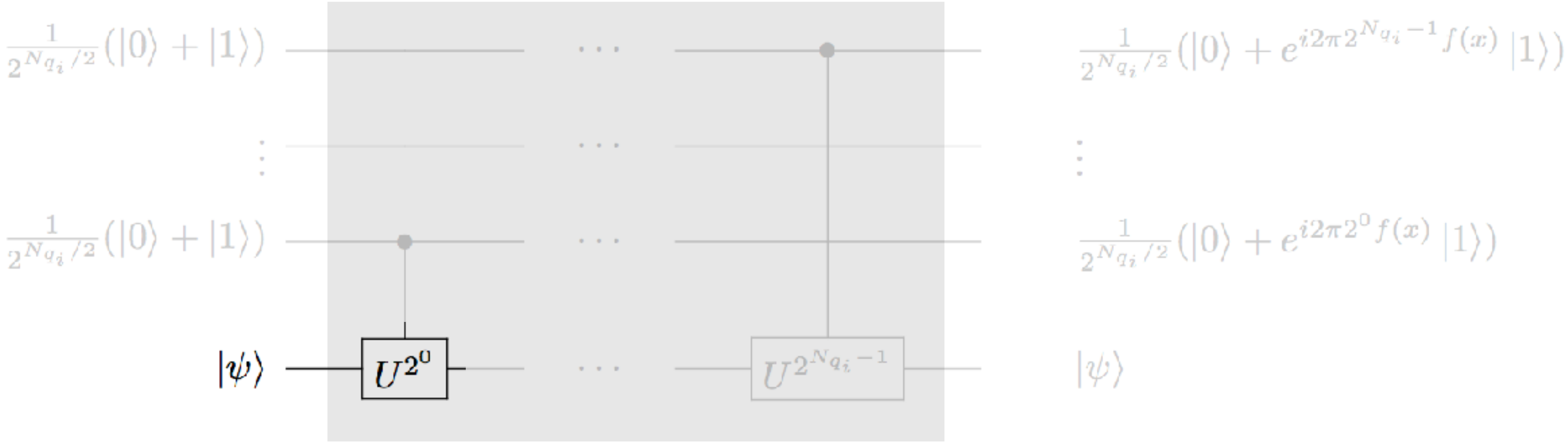
Construct U such that

$$U^{2^j}|\psi\rangle = e^{i2\pi 2^j f(x)}|\psi\rangle \tag{2}$$

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

phase kickback



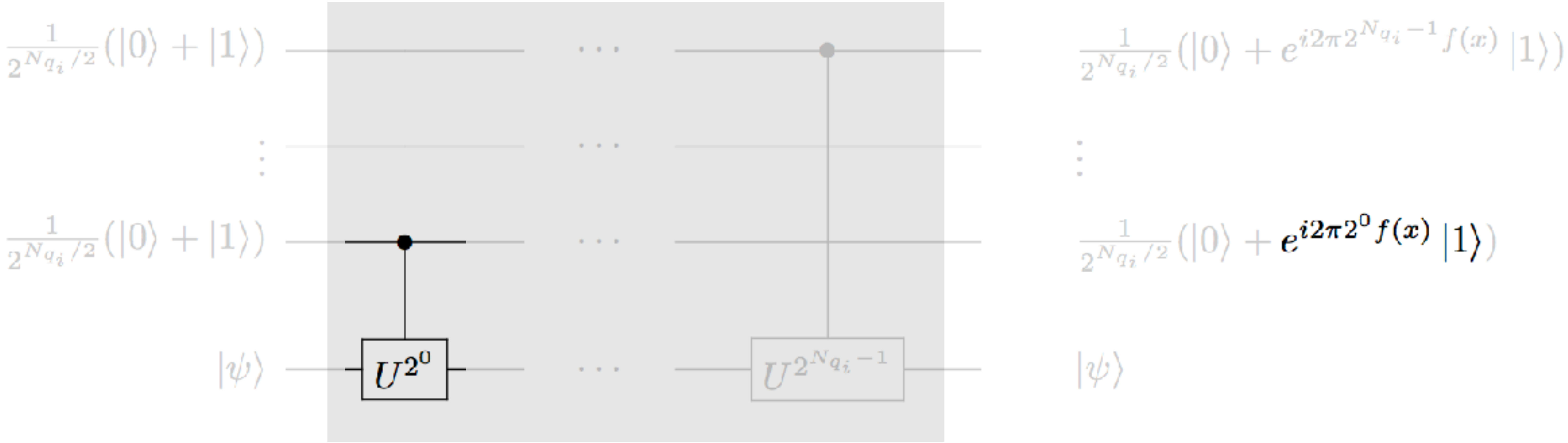
Construct U such that

$$U^{2^j}|\psi\rangle = e^{i2\pi 2^j f(x)}|\psi\rangle \tag{2}$$

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

phase kickback



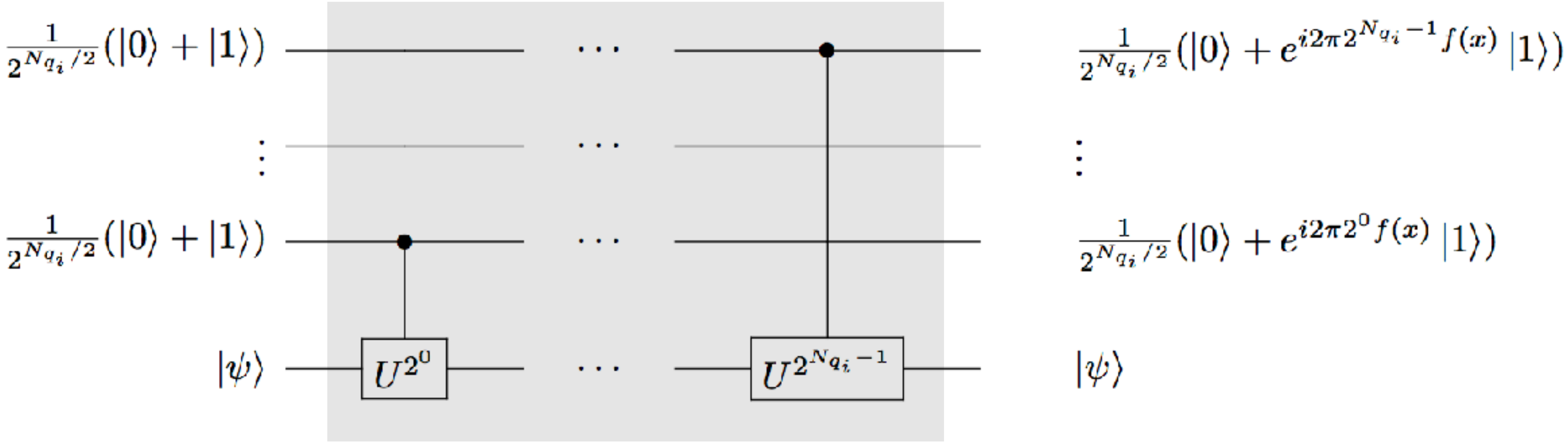
Construct U such that

$$U^{2^j}|\psi\rangle = e^{i2\pi 2^j f(x)}|\psi\rangle \tag{2}$$

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

phase kickback



Construct U such that

$$U^{2^j} |\psi\rangle = e^{i2\pi 2^j f(x)} |\psi\rangle \tag{2}$$

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

phase kickback

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

```
from grove.alpha.phaseestimation.phase_estimation import controlled
from grove.qft.fourier import inverse_qft

# encode f_h into CPHASE gate
U = np.array([[1, 0],
              [0, np.exp(1.0j * np.pi * f_h)]])
p_kickback = pq.Program()
# apply c-U^{2^j} to ancilla register
for i in input_qubits:
    if i > 0:
        U = np.dot(U, U)
    cU = controlled(U)
    name = "c-U{0}".format(2 ** i)
    p_kickback.defgate(name, cU)
    p_kickback.inst((name, i, ancilla_qubits[0]))
# iqft to pull out fractional component of eigenphase
p_kickback += inverse_qft(input_qubits)
return p_kickback
```

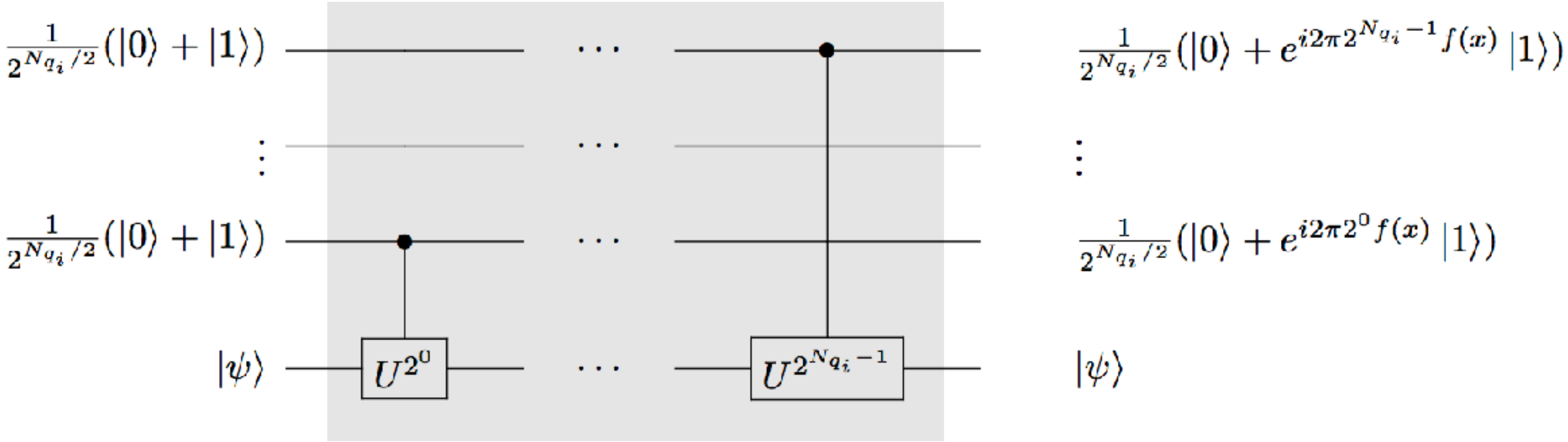
phase kickback

Hence, we may use the PHASE gate for U

$$U_{2\pi f(x)}^{2^j} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \quad (3)$$

```
from grove.alpha.phaseestimation.phase_estimation import controlled
from grove.qft.fourier import inverse_qft

# encode f_h into CPHASE gate
U = np.array([[1, 0],
              [0, np.exp(1.0j * np.pi * f_h)]])
p_kickback = pq.Program()
# apply c-U^{2^j} to ancilla register
for i in input_qubits:
    if i > 0:
        U = np.dot(U, U)
    cU = controlled(U)
    name = "c-U{0}".format(2 ** i)
    p_kickback.defgate(name, cU)
    p_kickback.inst((name, i, ancilla_qubits[0]))
# iqft to pull out fractional component of eigenphase
p_kickback += inverse_qft(input_qubits)
return p_kickback
```

phase kickback



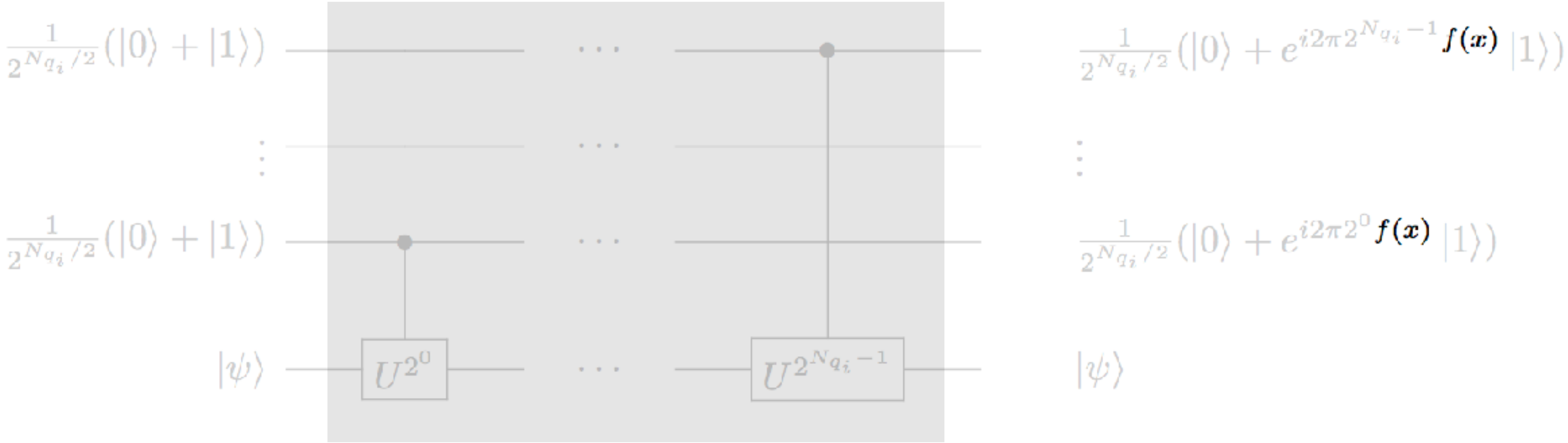
Construct U such that

$$U^{2^j} |\psi\rangle = e^{i2\pi 2^j f(x)} |\psi\rangle \tag{2}$$

Hence, we may use the PHASE gate for U

$$U^{2^j}_{2\pi f(x)} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i2\pi 2^j f(x)} \end{bmatrix} \tag{3}$$

phase kickback



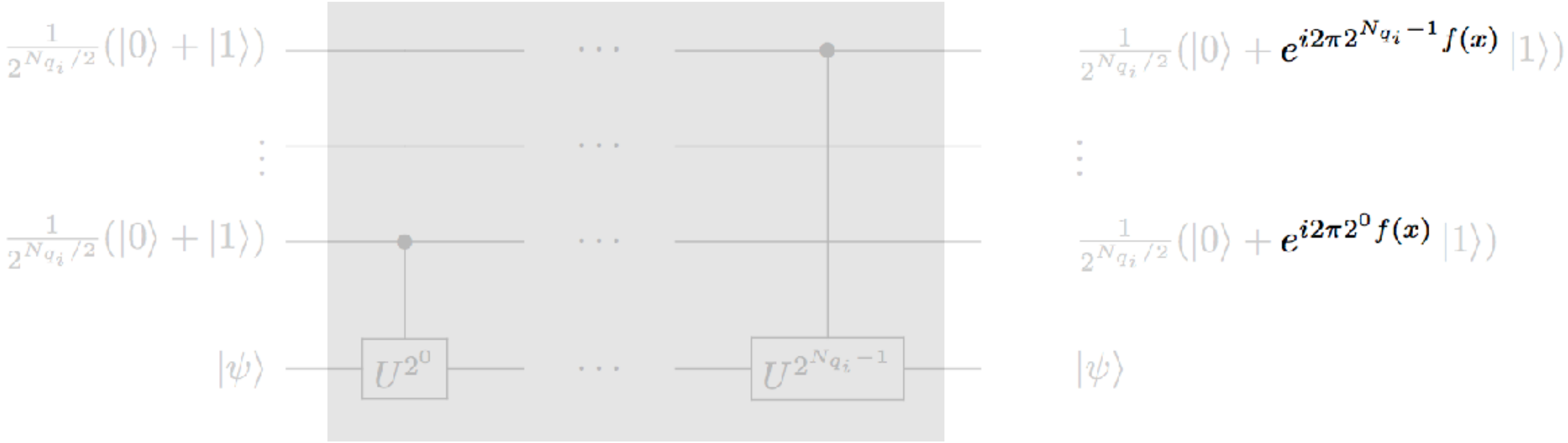
Sample $f(h)$. For small h, recall

$$f(h) \approx f(0) + h\nabla f(0) \tag{4}$$

So

$$e^{i2\pi 2^j f(h)} \approx e^{i2\pi 2^j f(0)} e^{i2\pi 2^j h\nabla f(0)} \tag{5}$$

phase kickback



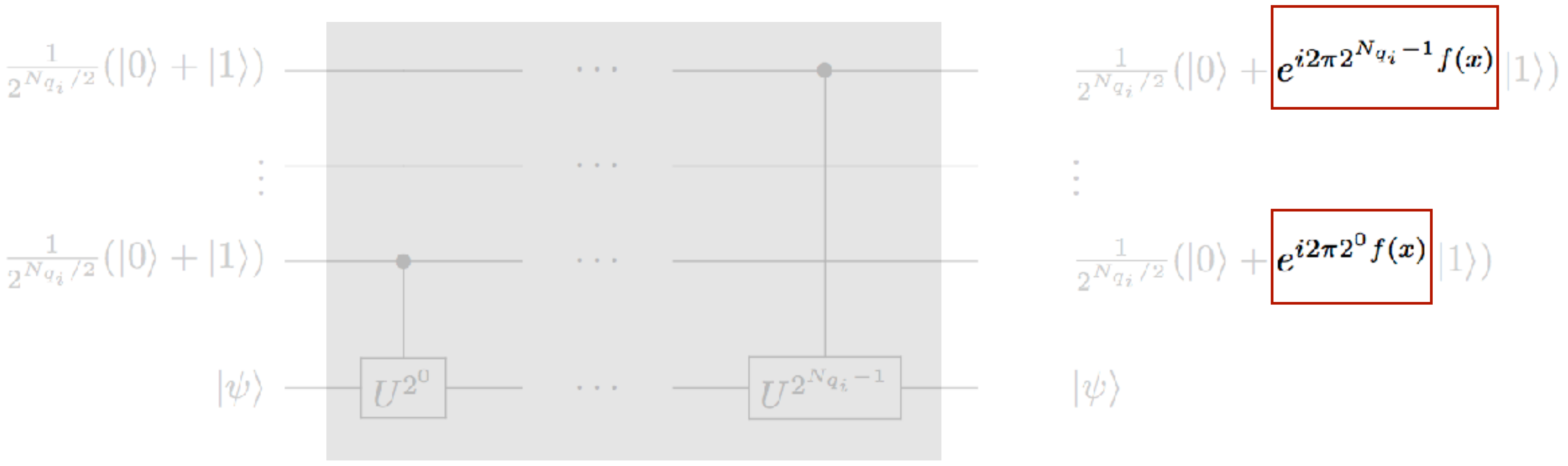
Sample $f(h)$. For small h, recall

$$f(h) \approx f(0) + h\nabla f(0) \tag{4}$$

So

$$e^{i2\pi 2^j f(h)} \approx e^{i2\pi 2^j f(0)} e^{i2\pi 2^j h\nabla f(0)} \tag{5}$$

phase kickback



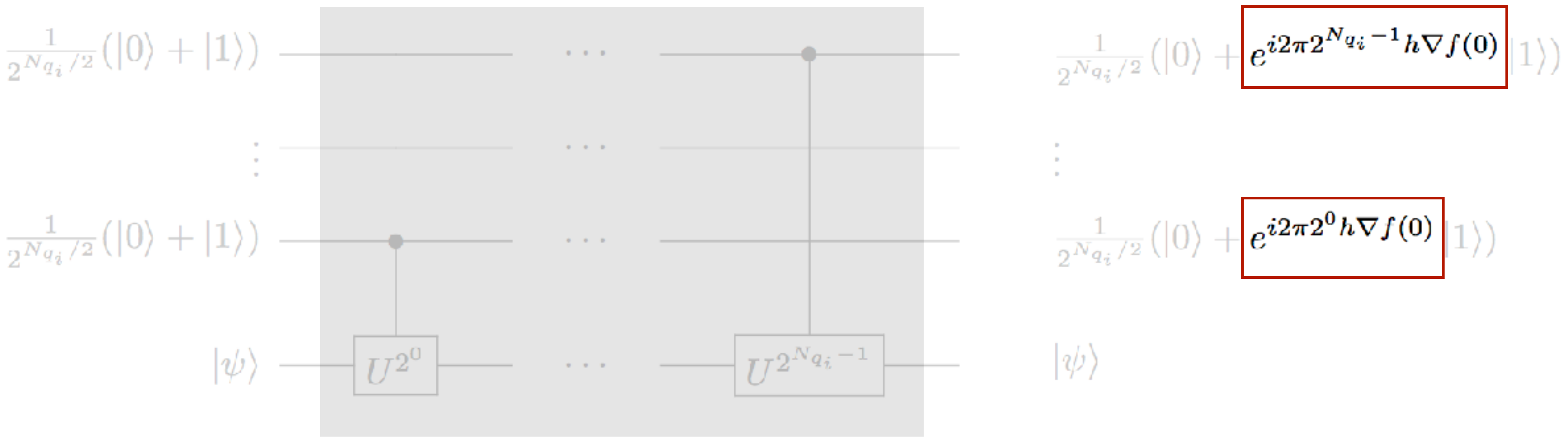
Sample $f(h)$. For small h, recall

$$f(h) \approx f(0) + h\nabla f(0) \tag{4}$$

So

$$e^{i2\pi 2^j f(h)} \approx e^{i2\pi 2^j f(0)} e^{i2\pi 2^j h\nabla f(0)} \tag{5}$$

phase kickback



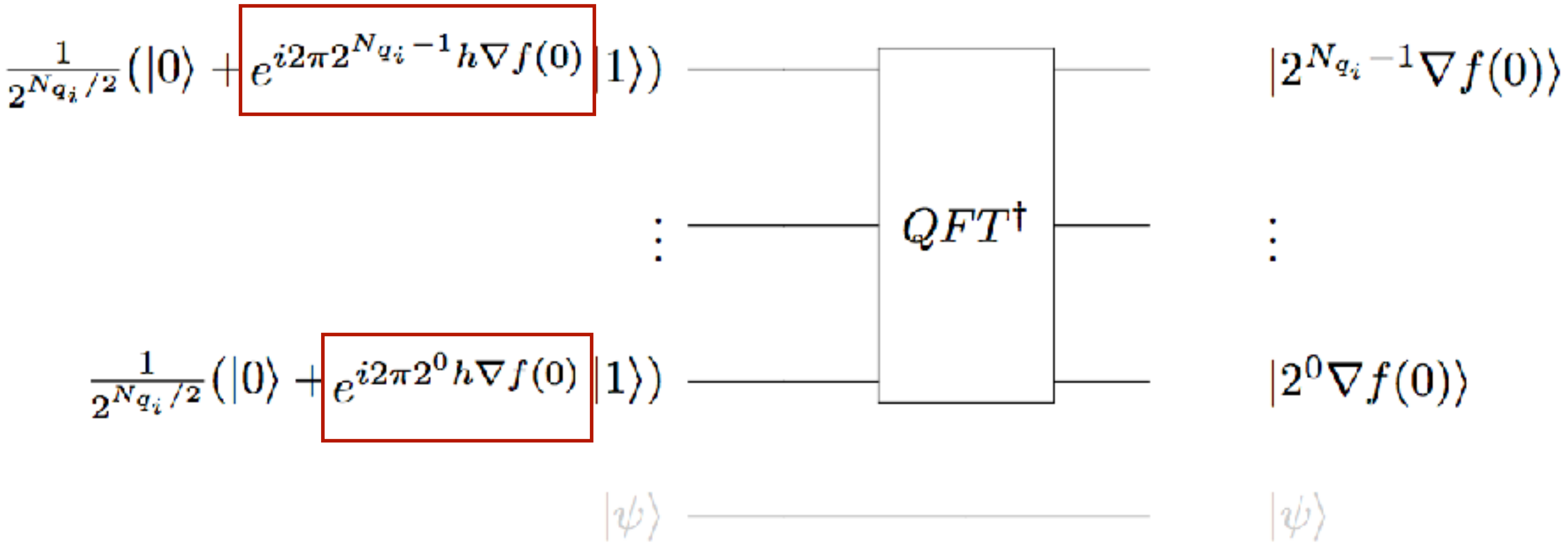
Sample $f(h)$. For small h, recall

$$f(h) \approx f(0) + h\nabla f(0) \tag{4}$$

So

$$e^{i2\pi 2^j f(h)} \approx e^{i2\pi 2^j f(0)} e^{i2\pi 2^j h\nabla f(0)} \tag{5}$$

phase kickback



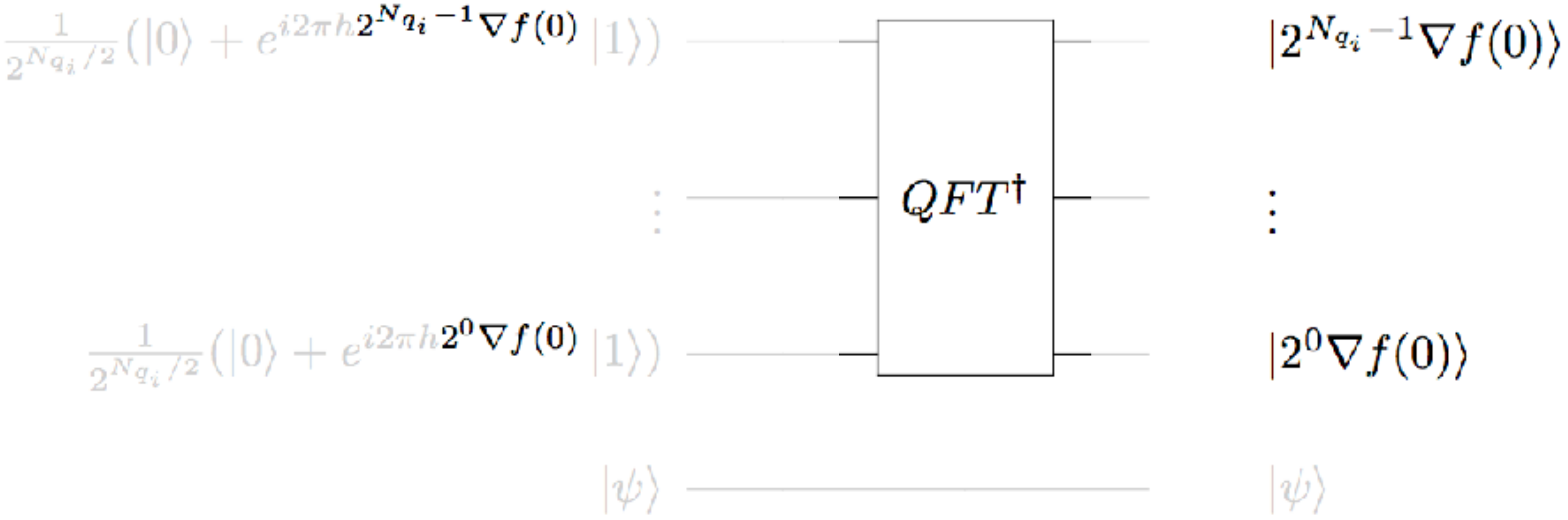
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0 \cdots b_{n-1})} &= e^{i2\pi b_0 \cdots b_j} e^{i2\pi b_{j+1} \cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1} \cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1} \cdots b_{n-1}} \tag{8}$$

phase kickback



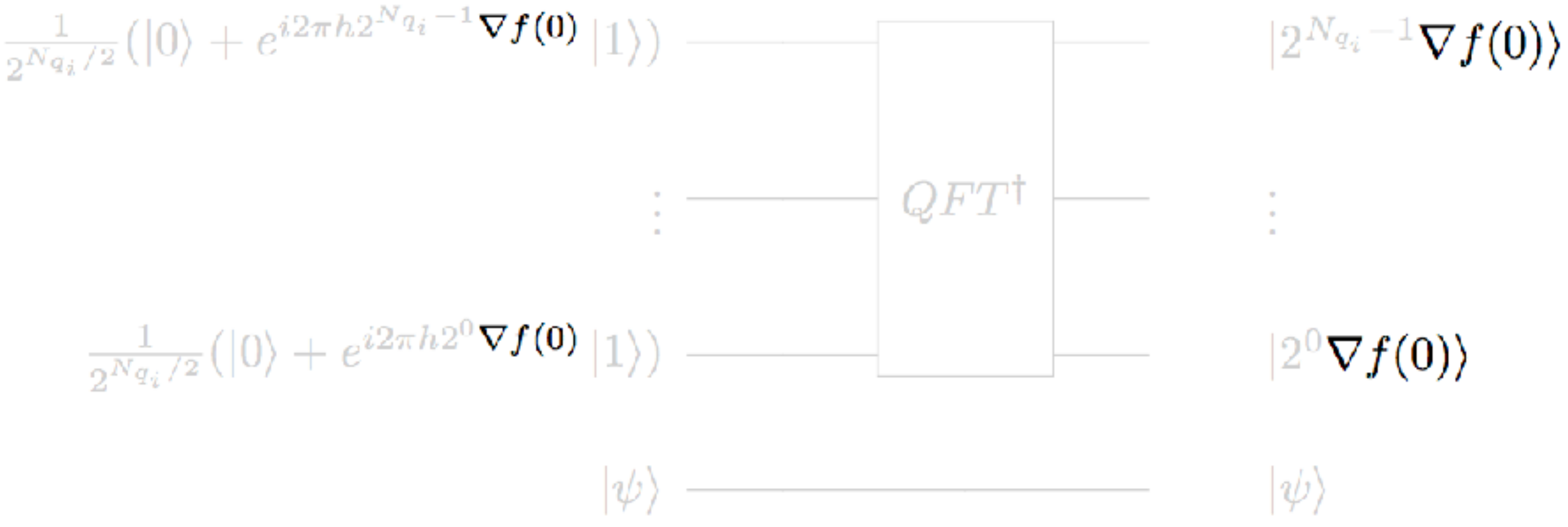
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0\cdots b_{n-1})} &= e^{i2\pi b_0\cdots b_j} e^{i2\pi b_{j+1}\cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1}\cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1}\cdots b_{n-1}} \tag{8}$$

phase kickback



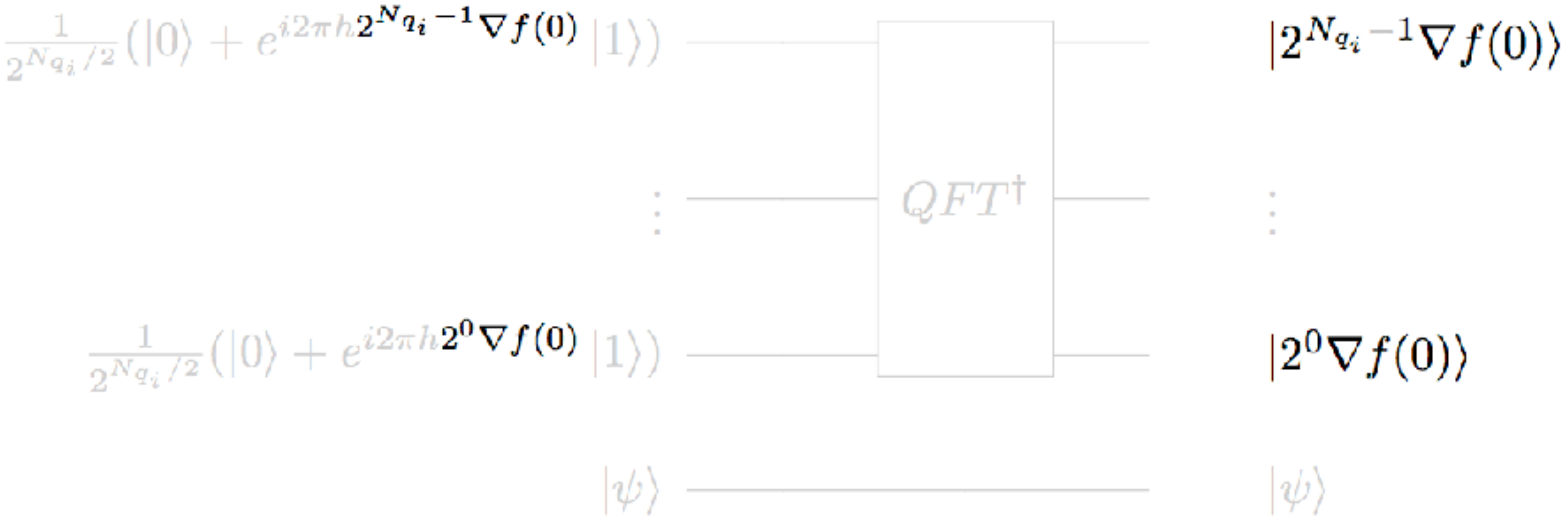
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0\cdots b_{n-1})} &= e^{i2\pi b_0\cdots b_j} e^{i2\pi b_{j+1}\cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1}\cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1}\cdots b_{n-1}} \tag{8}$$

phase kickback



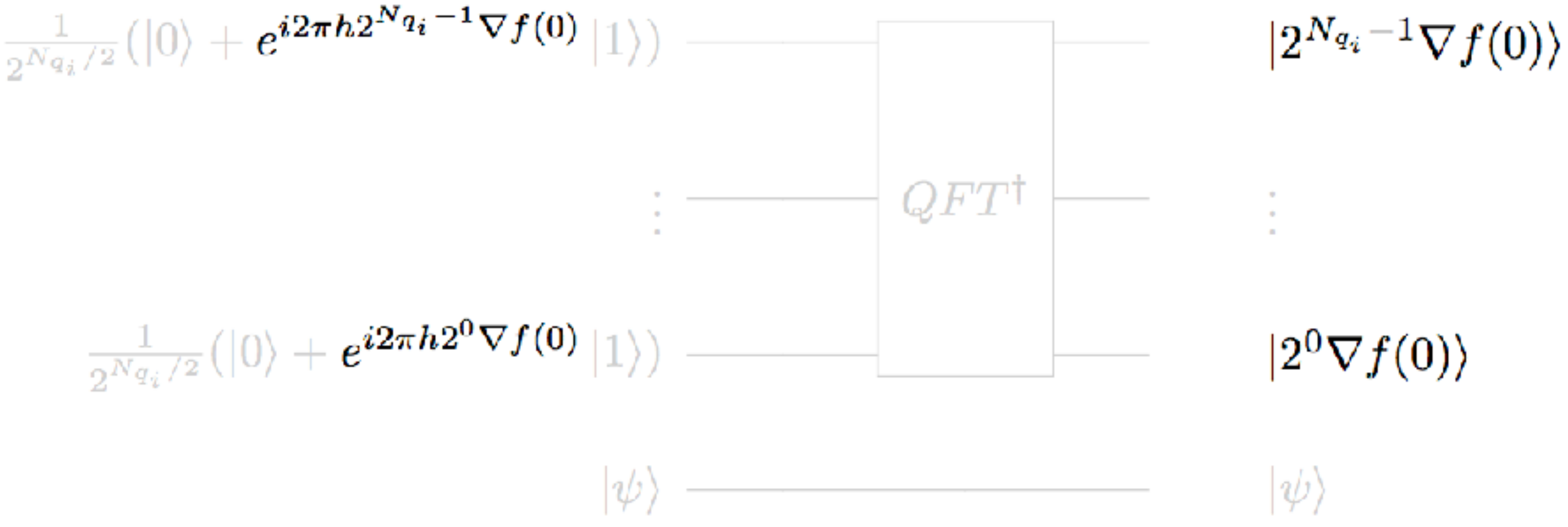
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0 \cdots b_{n-1})} &= e^{i2\pi b_0 \cdots b_j} e^{i2\pi b_{j+1} \cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1} \cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1} \cdots b_{n-1}} \tag{8}$$

phase kickback



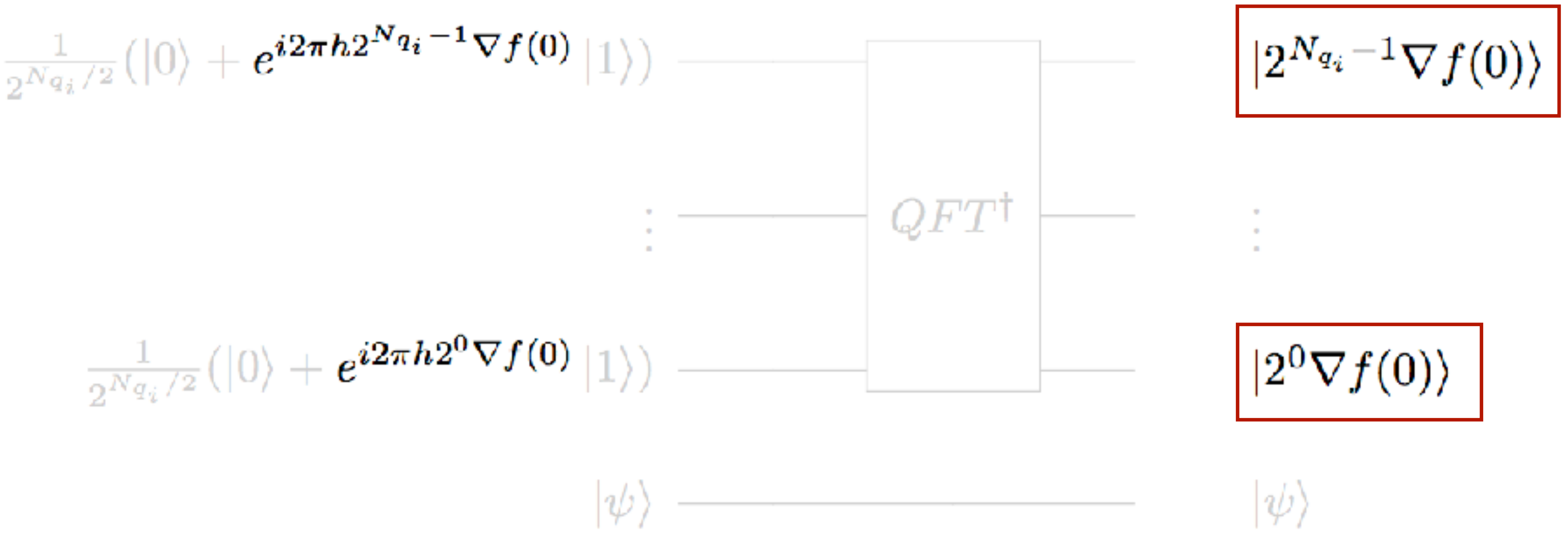
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0 \cdots b_{n-1})} &= e^{i2\pi b_0 \cdots b_j} e^{i2\pi b_{j+1} \cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1} \cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1} \cdots b_{n-1}} \tag{8}$$

phase kickback



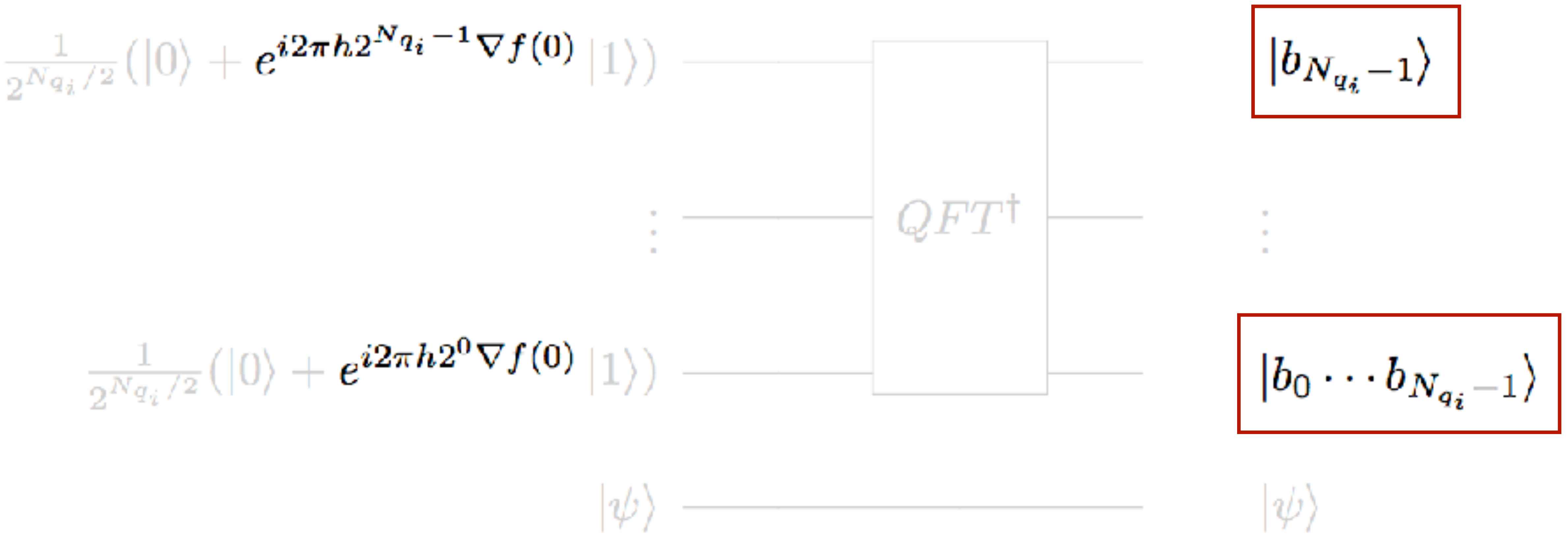
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j(0.b_0\cdots b_{n-1})} &= e^{i2\pi b_0\cdots b_j} e^{i2\pi b_{j+1}\cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1}\cdots b_{n-1}} \end{aligned}$$

$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1}\cdots b_{n-1}} \tag{8}$$

phase kickback



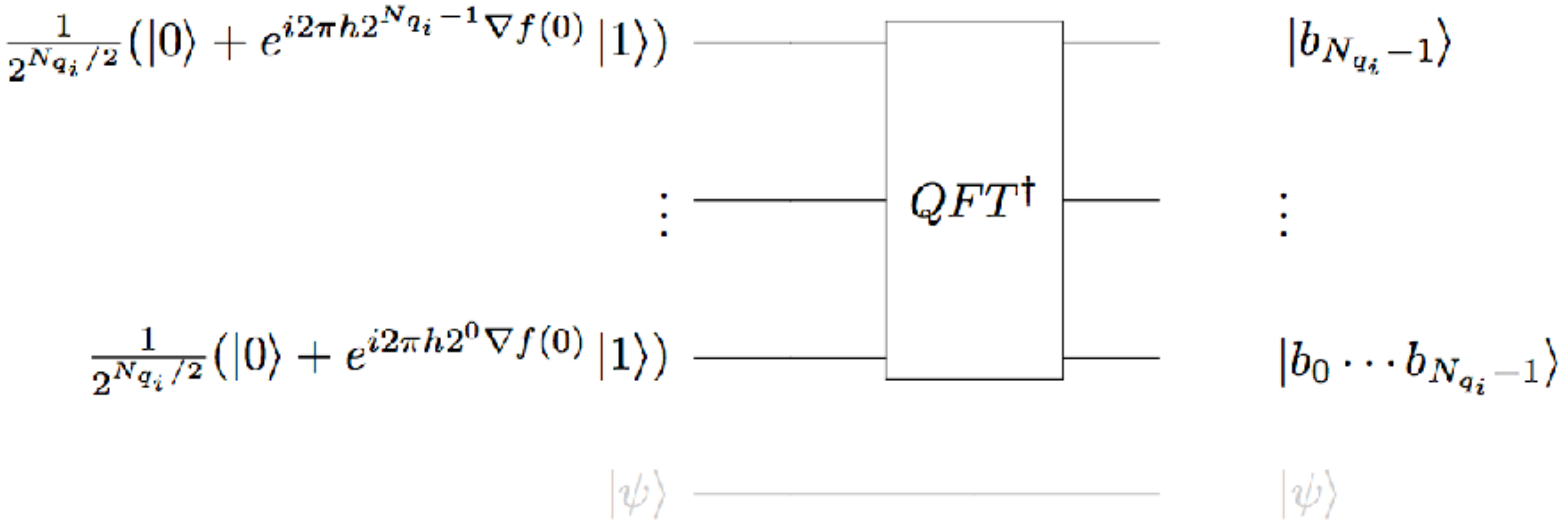
$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1} \tag{6}$$

$$2^j(0.b_0 \cdots b_{n-1}) = b_0 \cdots b_j.b_{j+1} \cdots b_{n-1} \tag{7}$$

$$\begin{aligned} e^{i2\pi 2^j (0.b_0 \cdots b_{n-1})} &= e^{i2\pi b_0 \cdots b_j} e^{i2\pi b_{j+1} \cdots b_{n-1}} \\ &= e^{i2\pi b_{j+1} \cdots b_{n-1}} \end{aligned}$$

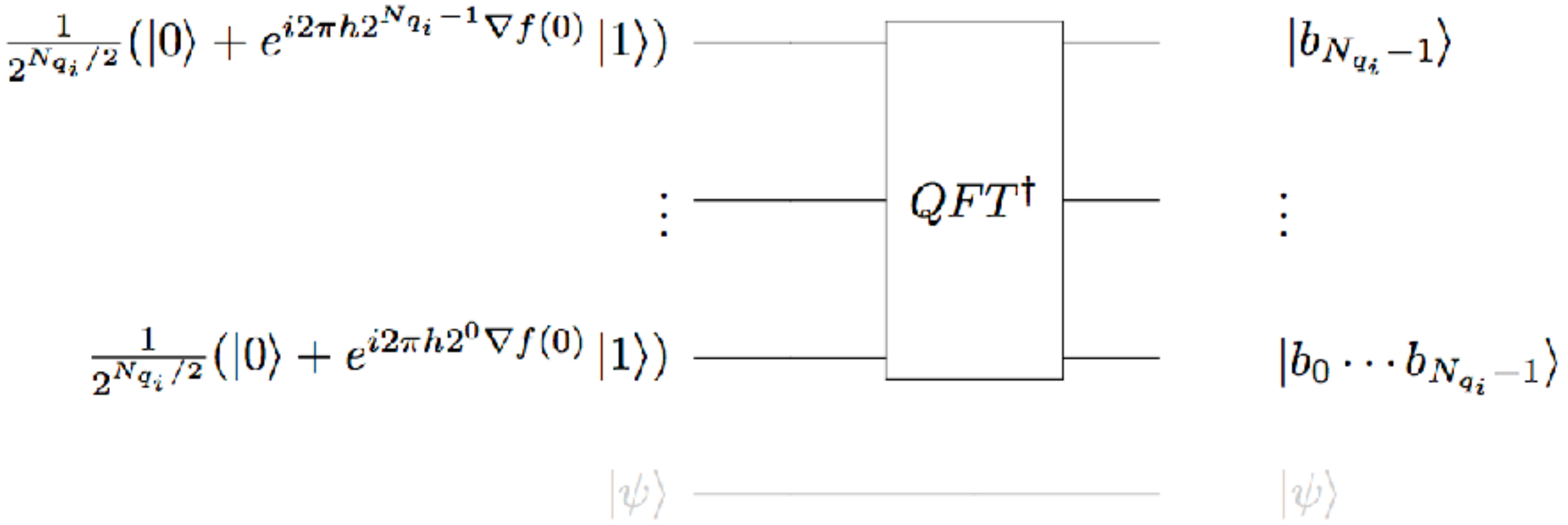
$$e^{i2\pi 2^j \nabla f(0)} \approx e^{i2\pi b_{j+1} \cdots b_{n-1}} \tag{8}$$

phase kickback




```
from grove.alpha.phaseestimation.phase_estimation import controlled
from grove.qft.fourier import inverse_qft

for i in input_qubits:
    if i > 0:
        U = np.dot(U, U)
    cU = controlled(U)
    name = "c-U{0}".format(2 ** i)
    p_kickback.defgate(name, cU)
    p_kickback.inst((name, i, ancilla_qubits[0]))
# iqft to pull out fractional component of eigenphase
p_kickback += inverse_qft(input_qubits)
return p_kickback
```

phase kickback




```
from grove.alpha.phaseestimation.phase_estimation import controlled
from grove.qft.fourier import inverse_qft

for i in input_qubits:
    if i > 0:
        U = np.dot(U, U)
    cU = controlled(U)
    name = "c-U{0}".format(2 ** i)
    p_kickback.defgate(name, cU)
    p_kickback.inst((name, i, ancilla_qubits[0]))
# iqft to pull out fractional component of eigenphase
p_kickback += inverse_qft(input_qubits)
return p_kickback
```

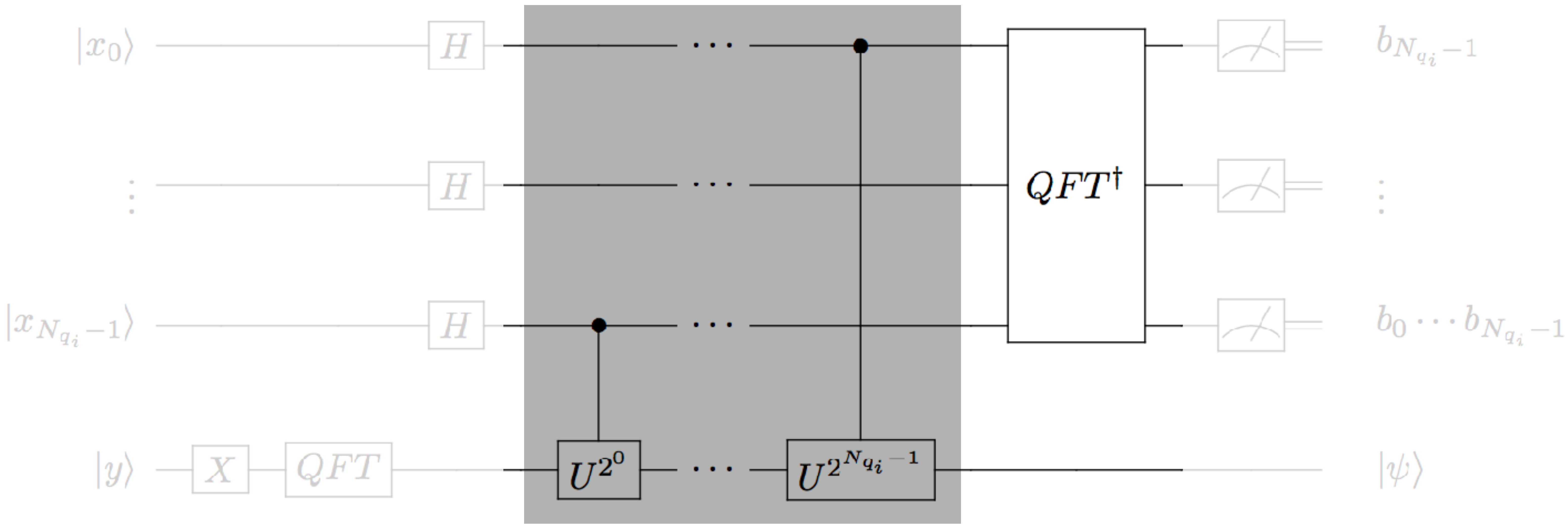
phase kickback

```
def phase_kickback(f_h, input_qubits, ancilla_qubits, precision):
    """ Phase kickback of f_h
```

program	ancilla qubits	input qubits
phase kickback	U	IQFT

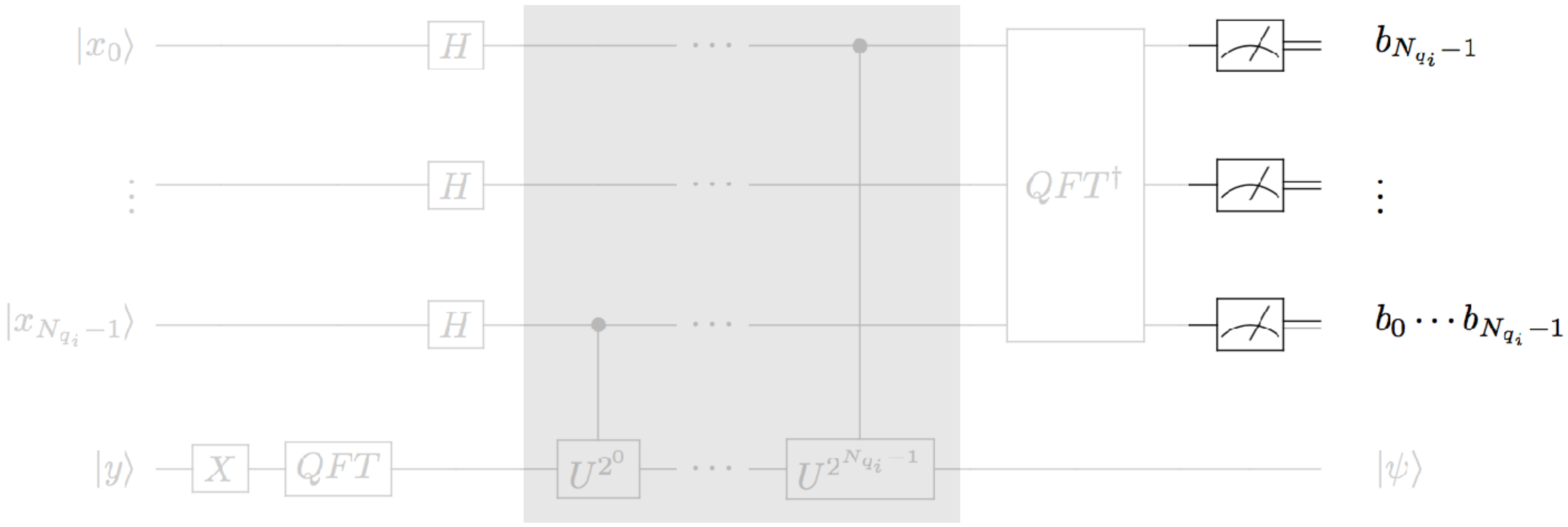
```
    """

    # encode f_h into CPHASE gate
    U = np.array([[1, 0],
                  [0, np.exp(1.0j * np.pi * f_h)]])
    p_kickback = pq.Program()
    # apply c-U^{2^j} to ancilla register
    for i in input_qubits:
        if i > 0:
            U = np.dot(U, U)
        cU = controlled(U)
        name = "c-U{0}".format(2 ** i)
        p_kickback.defgate(name, cU)
        p_kickback.inst((name, i, ancilla_qubits[0]))
    # iqft to pull out fractional component of eigenphase
    p_kickback += inverse_qft(input_qubits)
    return p_kickback
```

Jordan gradient estimation



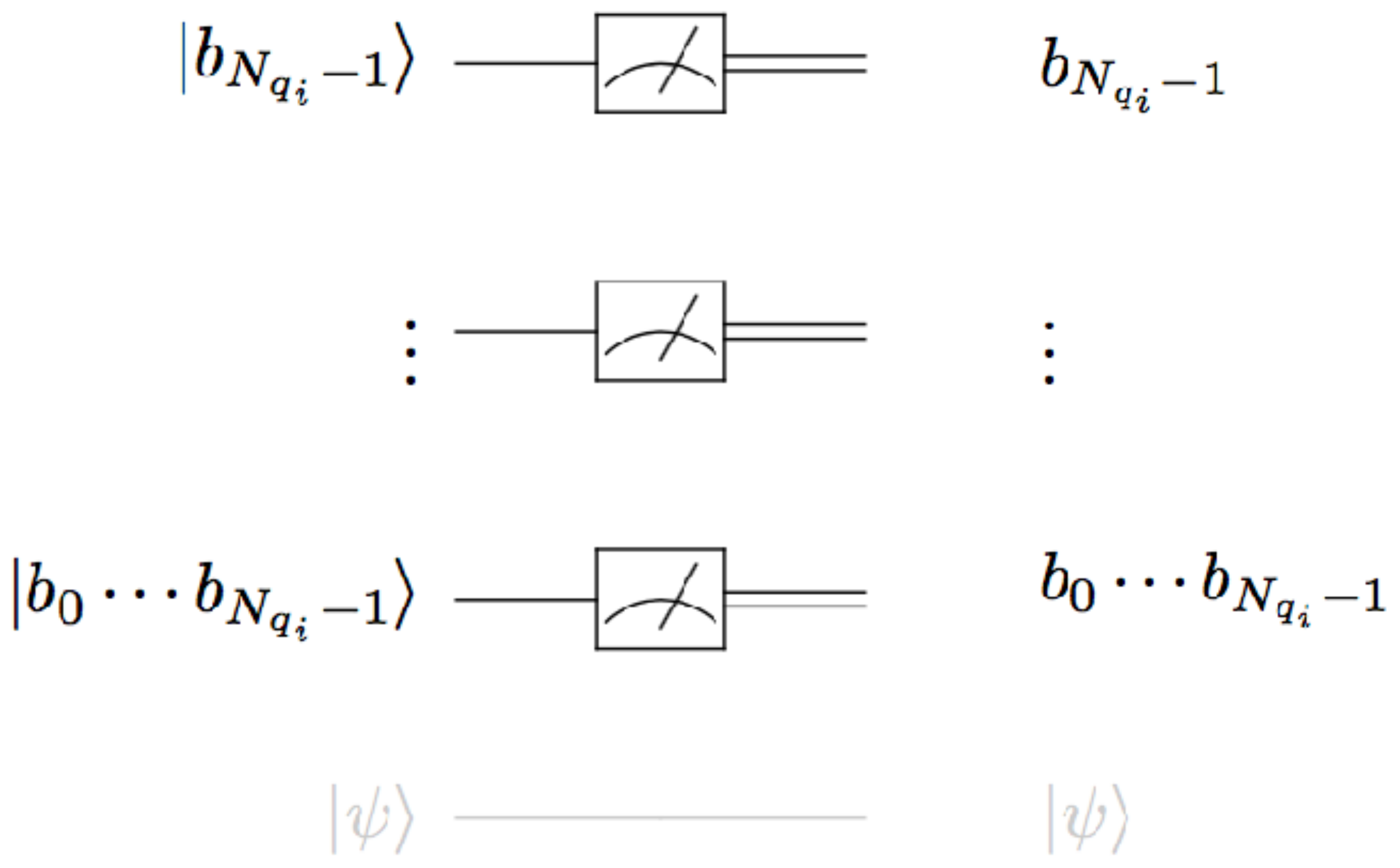
program	ancilla qubits	input qubits
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

Jordan gradient estimation



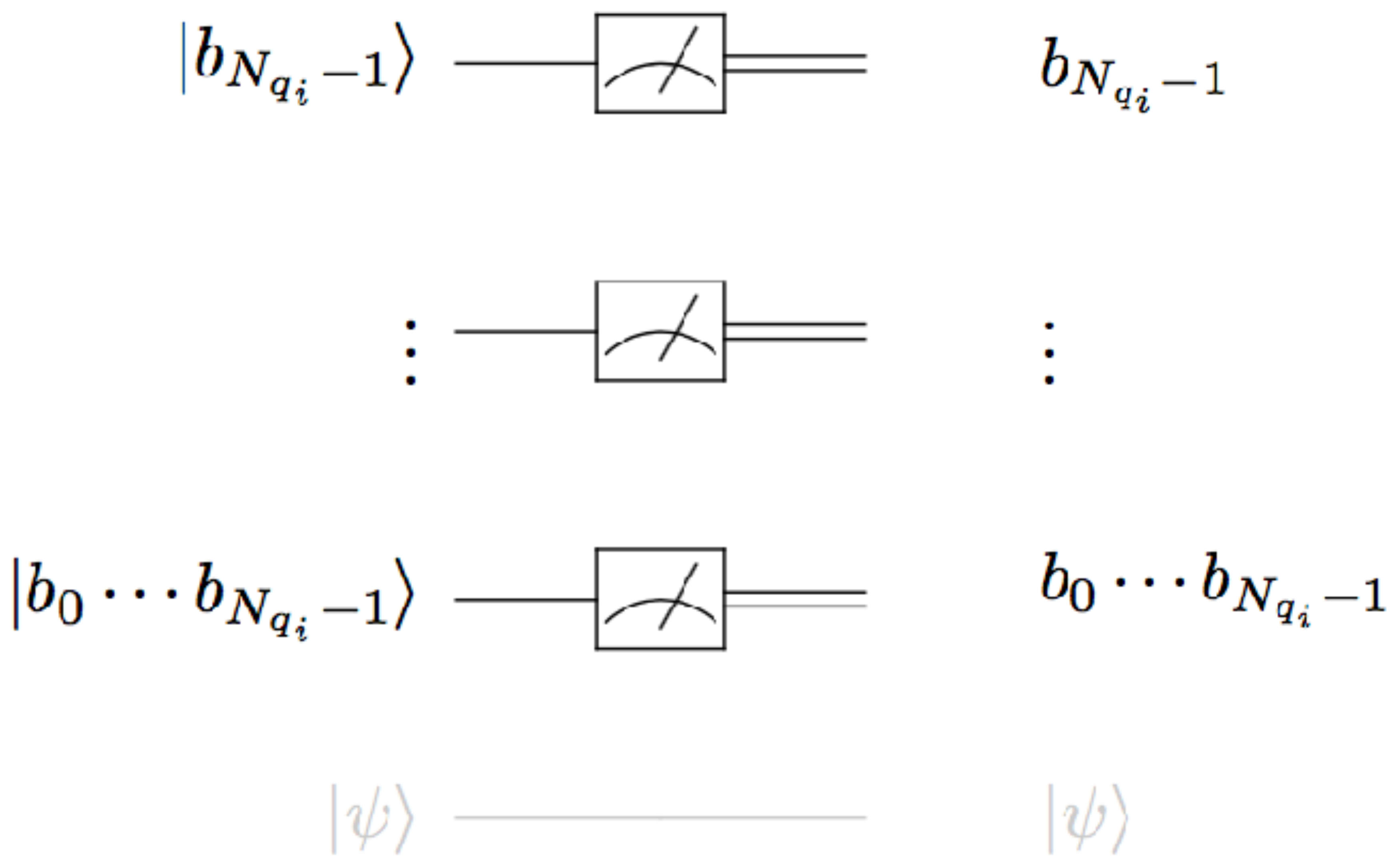
program	ancilla qubits	input qubits
initialize system	QFT, X	H
phase kickback	U	IQFT
measure	-	M

measure



$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1}$$

measure



$$\nabla f(0) \approx 0.b_0 \cdots b_{n-1}$$

```
from pyquil.api import SyncConnection
qvm = SyncConnection()
measurement = qvm.run_and_measure(p_gradient, input_qubits)
```

Running this algorithm

```
from jordan_gradient import gradient_estimator
p_gradient = gradient_estimator(f_h, input_qubits, ancilla_qubits,
        precision=precision)

from pyquil.api import SyncConnection
qvm = SyncConnection()
measurement = qvm.run_and_measure(p_gradient, input_qubits)
```

Running this algorithm

```
from jordan_gradient import gradient_estimator
p_gradient = gradient_estimator(f_h, input_qubits, ancilla_qubits,
        precision=precision)

from pyquil.api import SyncConnection
qvm = SyncConnection()
measurement = qvm.run_and_measure(p_gradient, input_qubits)
```

wrapper function

```python
def gradient_estimator(f_h, input_qubits, ancilla_qubits, precision=16):
    """ Gradient estimation via Jordan's algorithm
    10.1103/PhysRevLett.95.050501

    :param np.array f: Oracle outputs.
    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :param int precision: Bit precision of gradient.
    :return Program p_gradient: Quil program to estimate gradient of f.
    """

    # intialize input and output registers
    p_ic = initialize_system(input_qubits, ancilla_qubits)
    # encode oracle values into phase
    p_kickback = phase_kickback(f_h, input_qubits, ancilla_qubits, precision)
    # combine steps of algorithm into one program
    p_gradient = p_ic + p_kickback
    return p_gradient
```

wrapper function

```python
def gradient_estimator(f_h, input_qubits, ancilla_qubits, precision=16):
    """ Gradient estimation via Jordan's algorithm
    10.1103/PhysRevLett.95.050501

    :param np.array f: Oracle outputs.
    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :param int precision: Bit precision of gradient.
    :return Program p_gradient: Quil program to estimate gradient of f.
    """

    # intialize input and output registers
    p_ic = initialize_system(input_qubits, ancilla_qubits)
    # encode oracle values into phase
    p_kickback = phase_kickback(f_h, input_qubits, ancilla_qubits, precision)
    # combine steps of algorithm into one program
    p_gradient = p_ic + p_kickback
    return p_gradient
```

wrapper function

```
def gradient_estimator(f_h, input_qubits, ancilla_qubits, precision=16):
    """ Gradient estimation via Jordan's algorithm
    10.1103/PhysRevLett.95.050501

    :param np.array f: Oracle outputs.
    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :param int precision: Bit precision of gradient.
    :return Program p_gradient: Quil program to estimate gradient of f.
    """

    # intialize input and output registers
    p_ic = initialize_system(input_qubits, ancilla_qubits)
    # encode oracle values into phase
    p_kickback = phase_kickback(f_h, input_qubits, ancilla_qubits, precision)
    # combine steps of algorithm into one program
    p_gradient = p_ic + p_kickback
    return p_gradient
```

wrapper function

```
def gradient_estimator(f_h, input_qubits, ancilla_qubits, precision=16):
    """ Gradient estimation via Jordan's algorithm
    10.1103/PhysRevLett.95.050501

    :param np.array f: Oracle outputs.
    :param list input_qubit: Qubits of input registers.
    :param list ancilla_qubits: Qubits of output register.
    :param int precision: Bit precision of gradient.
    :return Program p_gradient: Quil program to estimate gradient of f.
    """

    # intialize input and output registers
    p_ic = initialize_system(input_qubits, ancilla_qubits)
    # encode oracle values into phase
    p_kickback = phase_kickback(f_h, input_qubits, ancilla_qubits, precision)
    # combine steps of algorithm into one program
    p_gradient = p_ic + p_kickback
    return p_gradient
```

Running this algorithm

```
from jordan_gradient import gradient_estimator
p_gradient = gradient_estimator(f_h, input_qubits, ancilla_qubits,
        precision=precision)

from pyquil.api import SyncConnection
qvm = SyncConnection()
measurement = qvm.run_and_measure(p_gradient, input_qubits)
```

Demo

Thank you

References

Fast Quantum Algorithm for Numerical Gradient Estimation
Stephen P. Jordan
10.1103/PhysRevLett.95.050501

Quantum Algorithm for Molecular Properties and Geometry Optimization
Ivan Kassal, Alán Aspuru-Guzik
10.1063/1.3266959

Code

github.com/rigetticomputing/grove
github.com/kmckiern/grove/blob/master/grove/alpha/jordan_gradient