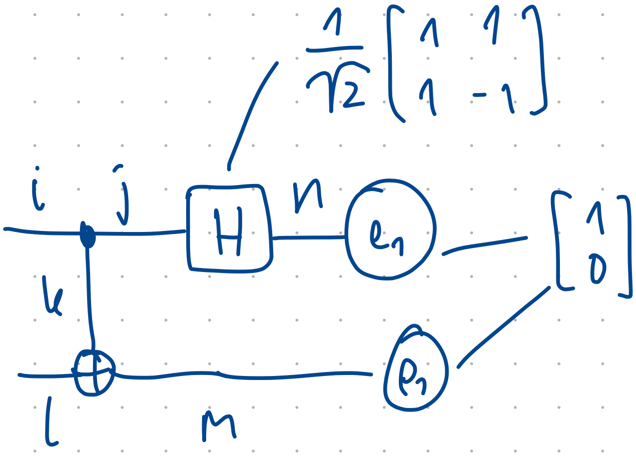
4.1

Group 2:
- Armin Ettenhofer
- Atul Agarwal
- Johannes Spies

- $C :\hat{=}$ 

- $C = \left(\sum_{j,k,m,n} \delta_{ijk} \cdot H_{jn} \cdot \oplus_{lkm} e_n \cdot e_m \right)_{il} = \left(\sum_{j,k} \delta_{ijk} \cdot H_{j0} \cdot \oplus_{lk0} \right)_{il}$

- $c_{0l} = \left(\sum_{j,k} \delta_{0jk} \cdot H_{j0} \cdot \oplus_{lk0} \right)_l = \left(\underbrace{\delta_{000}}_{1} \underbrace{H_{00}}_{\frac{1}{\sqrt{2}}} \oplus_{l00} \right)_l = \left(\frac{1}{\sqrt{2}} \oplus_{l00} \right)_l$

  $c_{1l} = \left(\sum_{j,k} \delta_{1jk} \cdot H_{j0} \cdot \oplus_{lk0} \right)_l = \left(\underbrace{\delta_{111}}_{1} \cdot \underbrace{H_{10}}_{1/\sqrt{2}} \cdot \oplus_{l10} \right)_l = \left(\frac{1}{\sqrt{2}} \oplus_{l10} \right)_l$

  $c_{00} = \frac{1}{\sqrt{2}} \oplus_{000} = \frac{1}{\sqrt{2}}$ $\quad$ $c_{01} = \frac{1}{\sqrt{2}} \oplus_{100} = 0$

  $c_{10} = \frac{1}{\sqrt{2}} \oplus_{010} = 0$ $\quad$ $c_{11} = \frac{1}{\sqrt{2}} \oplus_{110} = \frac{1}{\sqrt{2}}$

- $\Rightarrow \quad C = \frac{1}{\sqrt{2}} I_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

4.2

```python
#!/usr/bin/env python3
import numpy as np

def compact(A):
    """Calculate the compact SVD representation for a matrix A and retain r singular values"""
    # singular value decomposition
    V, S, Wh = np.linalg.svd(A)
    r = np.count_nonzero(S)

    # truncate matrices
    V_tilde = V[:, :r]
    S_tilde = S[:r]
    Wh_tilde = Wh[:r, :]

    return V_tilde, S_tilde, Wh_tilde, r

def decompose(U):
    """Construct a compact representation for a unitary matrix U"""
    T = np.reshape(U, (2, 2, 2, 2))
    T = np.transpose(T, (0, 2, 1, 3))
    U_tilde = np.reshape(T, (4, 4))

    V, S, W, r = compact(U_tilde)

    V = np.reshape(V, (2, 2, r))
    W = np.reshape(W, (r, 2, 2))
    return V, S, W, r

def reconstruct(V, S, W, r):
    """Reconstruct a matrix from compact SVD"""
    return sum(S[j] * np.kron(V[:, :, j], W[j, :, :]) for j in range(r))

if __name__ == "__main__":
    # specify CNOT gate
    U_CNOT = np.eye(4)
    U_CNOT[2][2] = 0
    U_CNOT[3][3] = 0
    U_CNOT[2][3] = 1
    U_CNOT[3][2] = 1
    CNOT_decomp = decompose(U_CNOT)
    CNOT_recons = reconstruct(*CNOT_decomp)
    assert np.allclose(U_CNOT, CNOT_recons)

    # specify fSim gate
    theta, phi = np.pi / 3, np.pi / 4
    U_fSim = np.array([
        [1.0, 0.0, 0.0, 0.0],
        [0.0, np.cos(theta), -1j*np.sin(theta), 0.0],
        [0.0, -1j*np.sin(theta), np.cos(theta), 0.0],
        [0.0, 0.0, 0.0, np.exp(-1j*phi)],
    ])
    fSim_decomp = decompose(U_fSim)
    fSim_recons = reconstruct(*fSim_decomp)
    assert np.allclose(U_fSim, fSim_recons)
```