

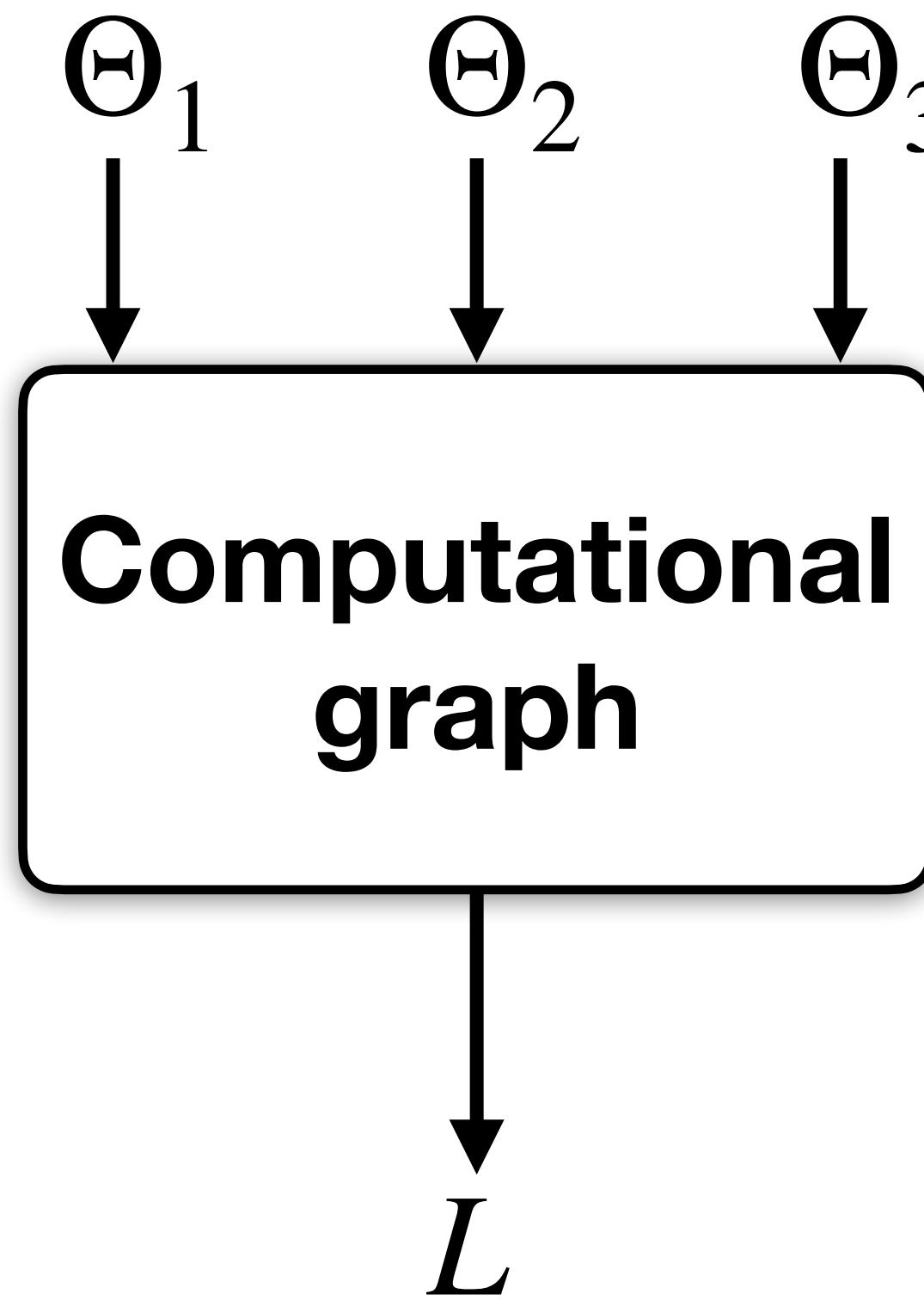


Rapid prototyping of quantum information processing tasks with deep learning libraries

Ilia Luchnikov

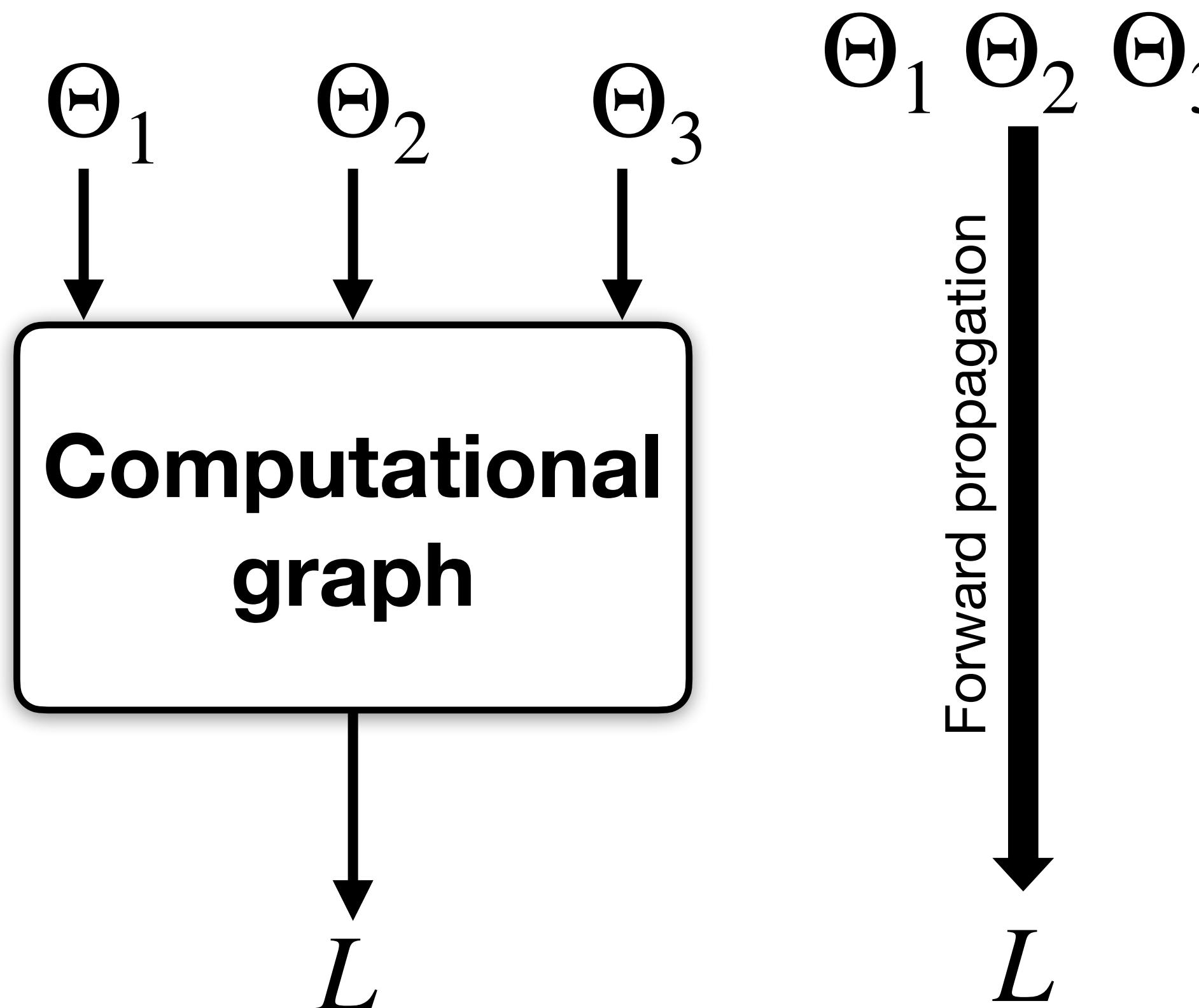


Key feature of deep learning (or AD) libraries: automatic differentiation



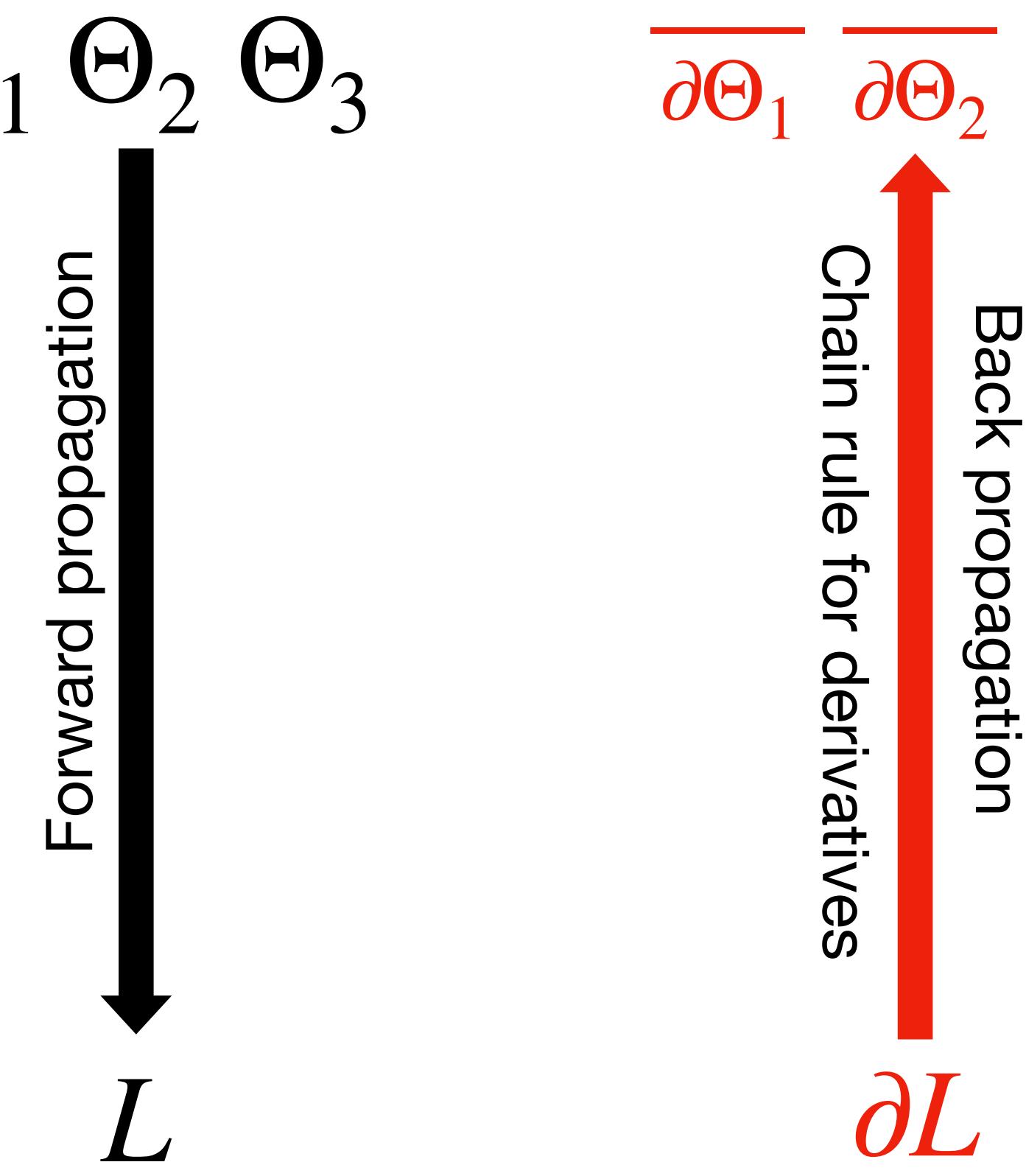
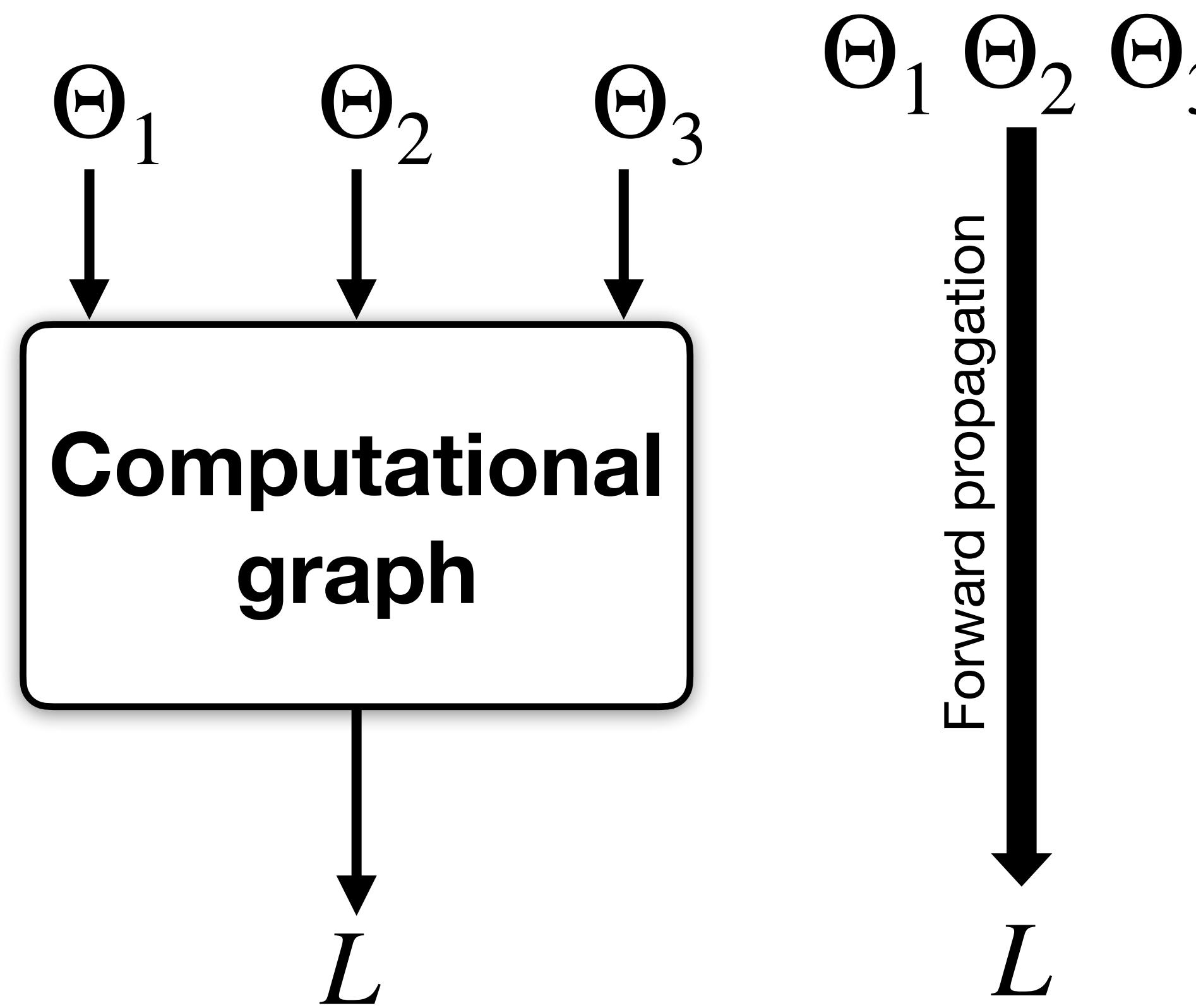


Key feature of deep learning (or AD) libraries: automatic differentiation





Key feature of deep learning (or AD) libraries: automatic differentiation



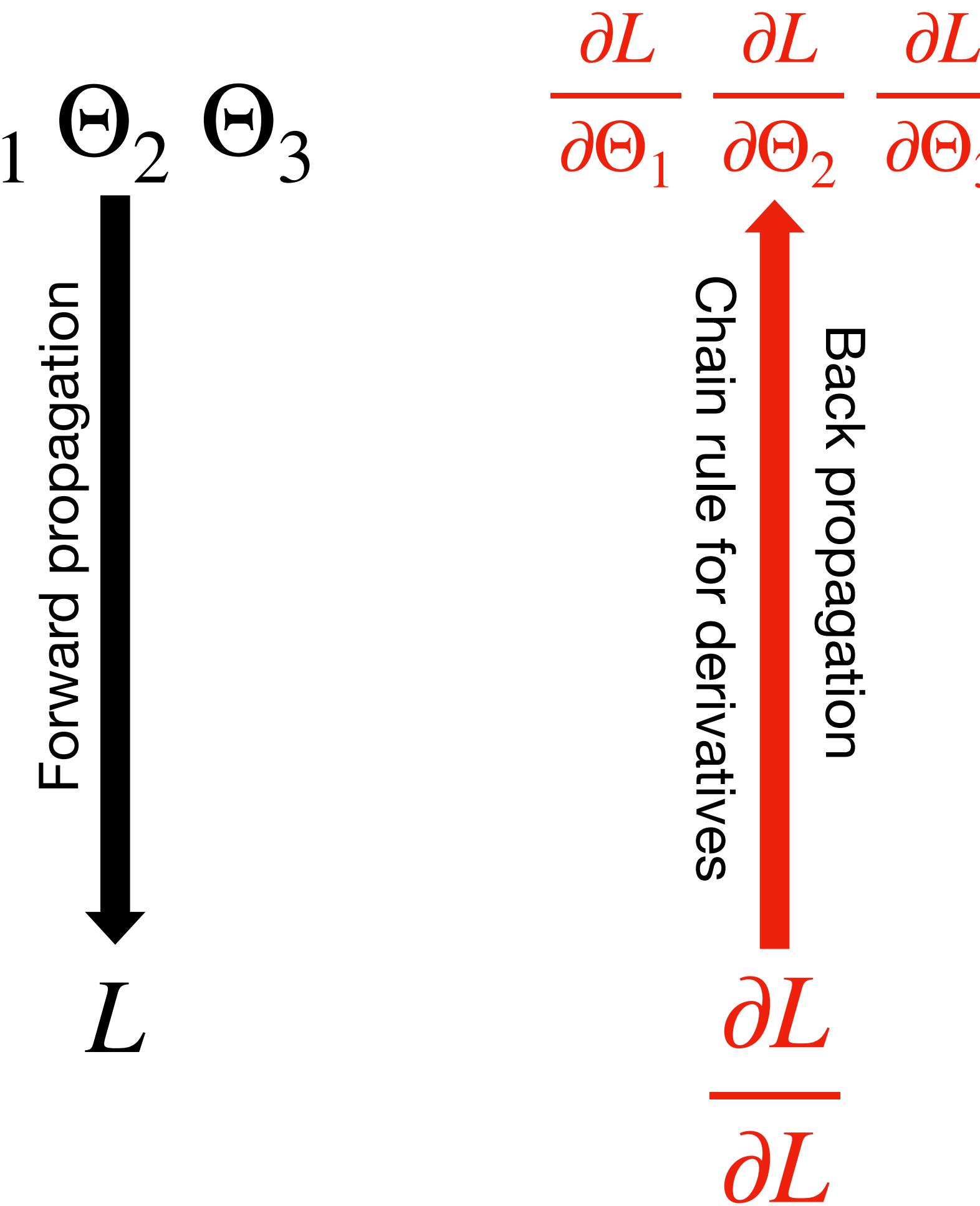
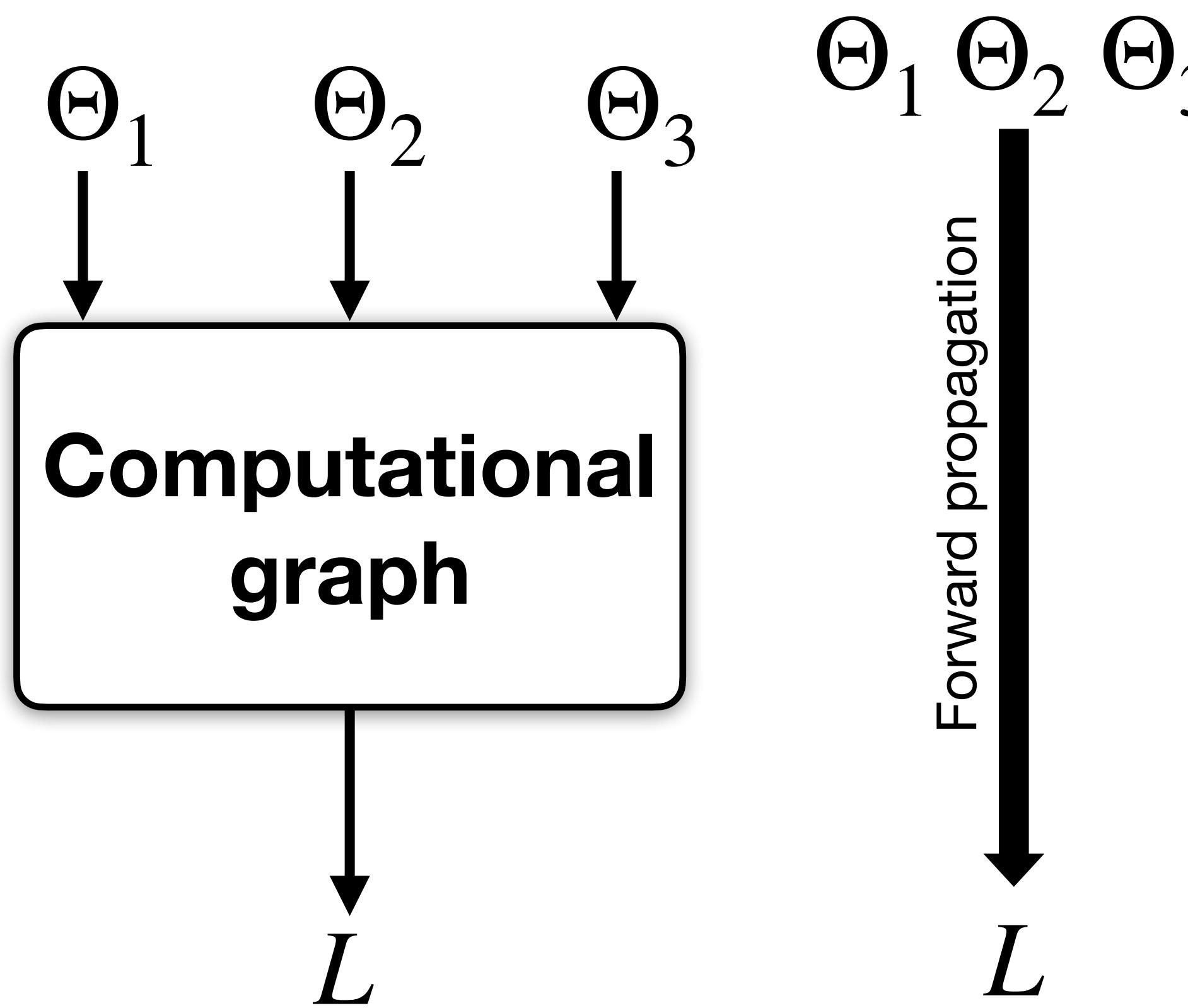
$$\frac{\partial L}{\partial \Theta_1} \quad \frac{\partial L}{\partial \Theta_2} \quad \frac{\partial L}{\partial \Theta_3}$$

↑
Chain rule for derivatives
↓

$$\frac{\partial L}{\partial L}$$



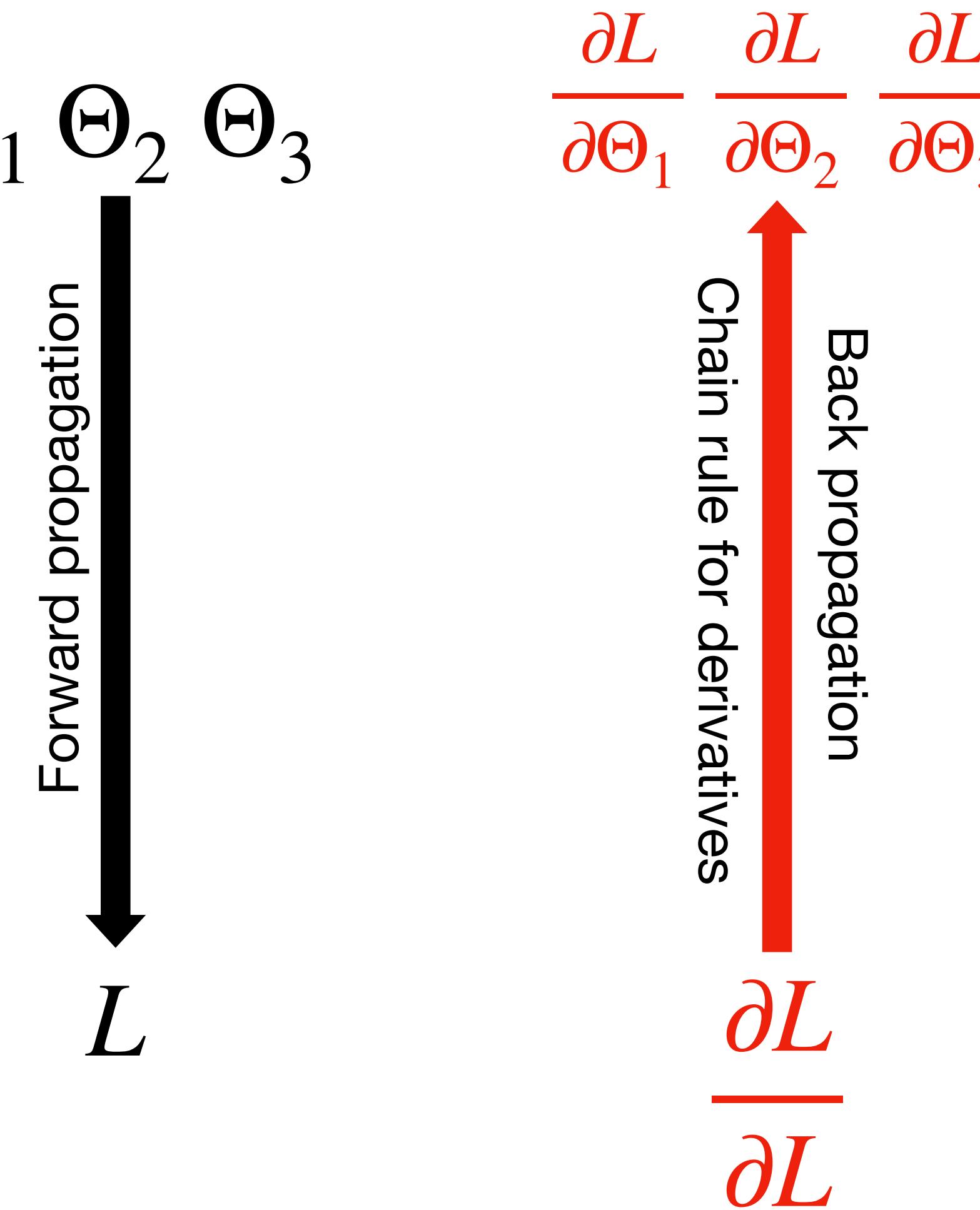
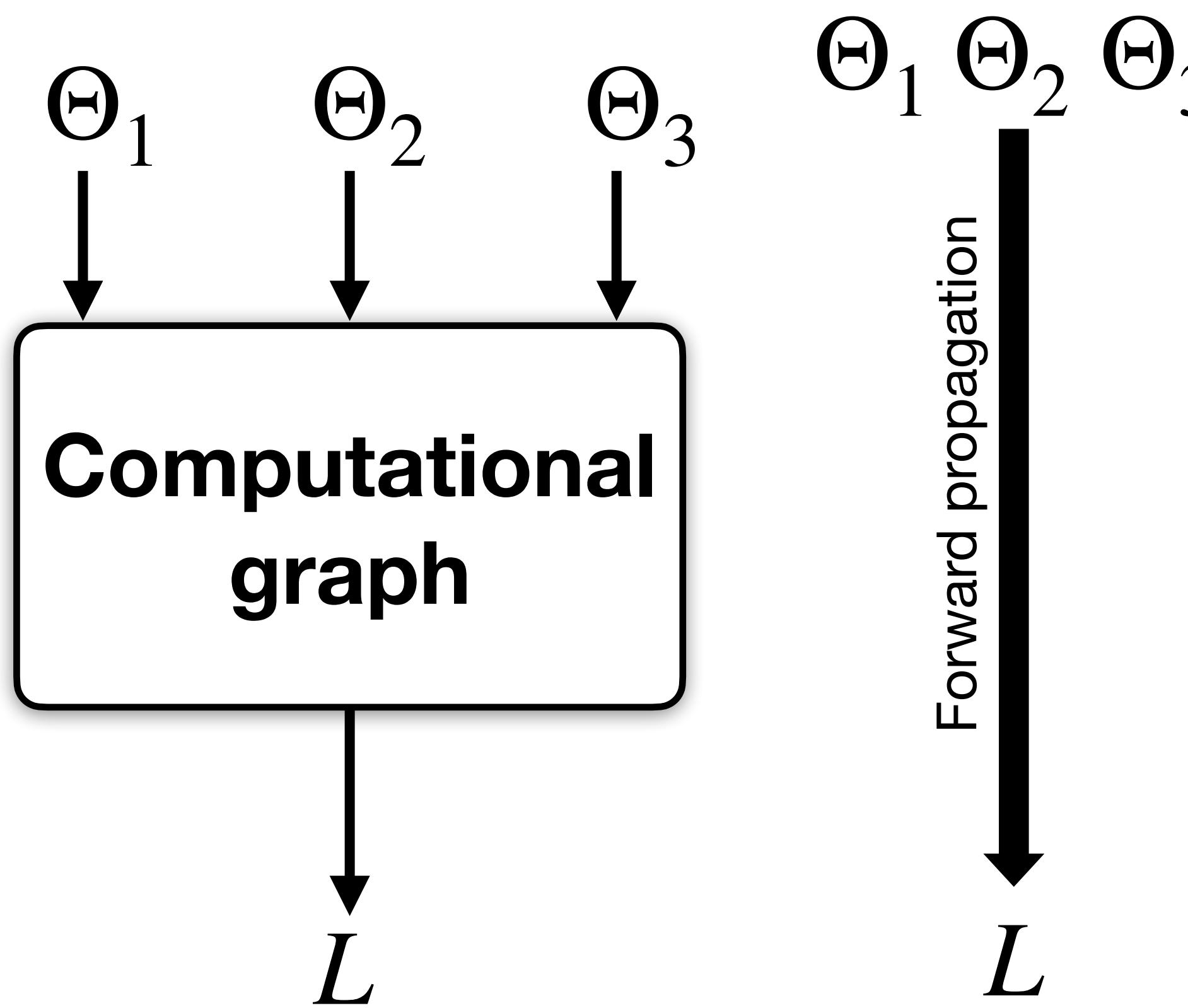
Key feature of deep learning (or AD) libraries: automatic differentiation



- **Caffe**
- **PyTorch**
- **TensorFlow**
- **Jax**
- **Flux**
- etc



Key feature of deep learning (or AD) libraries: automatic differentiation



- Caffe
- PyTorch
- **TensorFlow**
- Jax
- Flux
- etc



The extended version of all pieces of code is available here

[https://github.com/LuchnikovI/
RQC School Rapid prototyping](https://github.com/LuchnikovI/RQC_School_Rapid_prototyping)



Initialization of tensors



```
1   T = tf.zeros((2, 3, 4), dtype=tf.complex128)
```



Initialization of tensors



```
1   T = tf.zeros((2, 3, 4), dtype=tf.complex128)
```

$$T \in \mathbb{C}^{2 \times 3 \times 4}$$

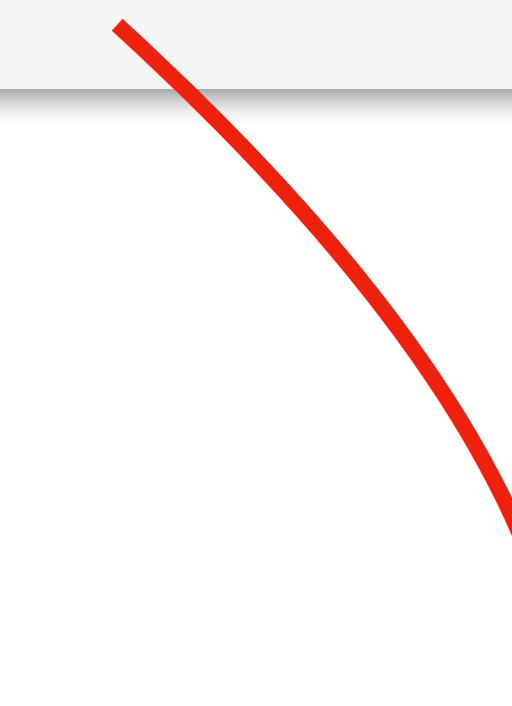


Initialization of tensors



```
1   T = tf.zeros((2, 3, 4), dtype=tf.complex128)
```

$$T \in \mathbb{C}^{2 \times 3 \times 4}$$



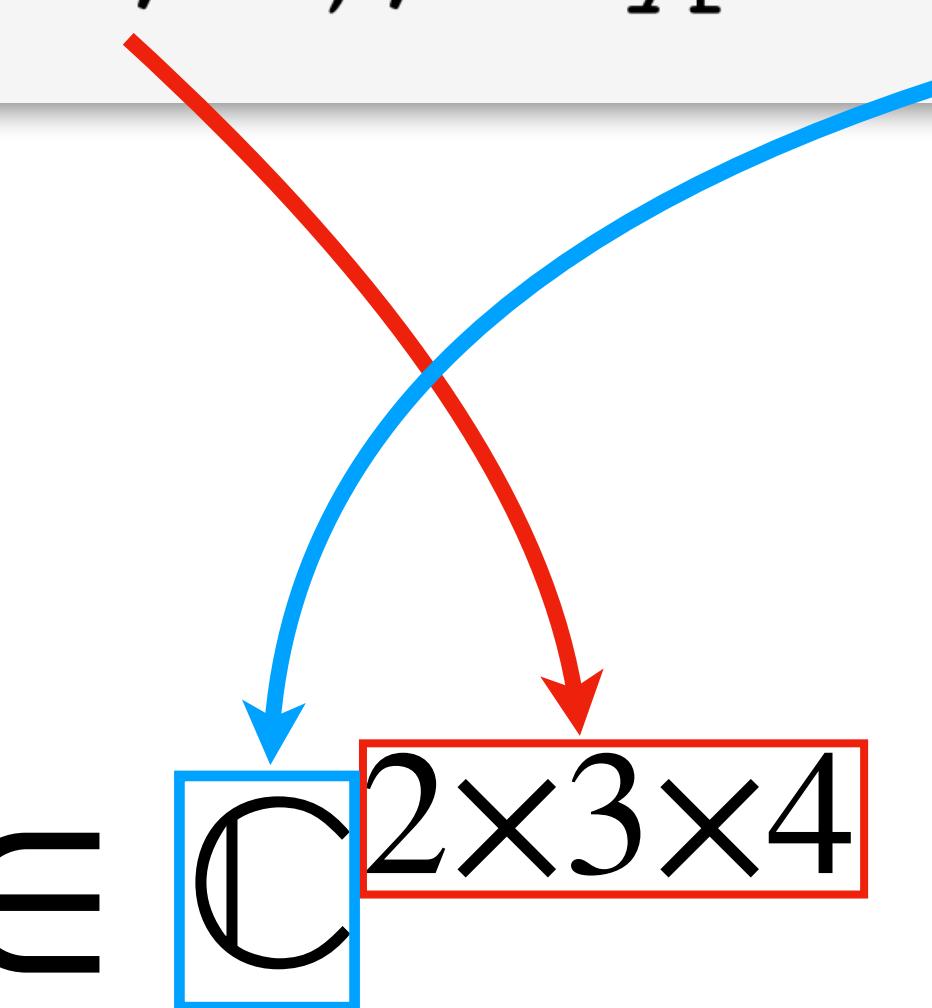


Initialization of tensors



```
1   T = tf.zeros((2, 3, 4), dtype=tf.complex128)
```

$$T \in \mathbb{C}^{2 \times 3 \times 4}$$





Initialization of tensors



```
1   T = tf.zeros((2, 3, 4), dtype=tf.complex128)
```

$$T \in \mathbb{C}^{2 \times 3 \times 4}$$

```
1   print(T.shape)
2   print(T.dtype)
```

```
(2, 3, 4)
<dtype: 'complex128'>
```



Initialization of tensors



$T \in \mathbb{R}^{2 \times 3 \times 4}, T \sim N(0, I)$

```
1   T = tf.random.normal((2, 3, 4), dtype=tf.float64)
```



Initialization of tensors



$T \in \mathbb{R}^{2 \times 3 \times 4}, T \sim N(0, I)$

```
1   T = tf.random.normal((2, 3, 4), dtype=tf.float64)
```

$T = I \in \mathbb{C}^{4 \times 4}$

```
1   T = tf.eye(4, dtype=tf.complex128)
```



Initialization of tensors



$T \in \mathbb{R}^{2 \times 3 \times 4}, T \sim N(0, I)$

```
1   T = tf.random.normal((2, 3, 4), dtype=tf.float64)
```

$T = I \in \mathbb{C}^{4 \times 4}$

```
1   T = tf.eye(4, dtype=tf.complex128)
```

```
1   T = tf.constant([1, 2, 5, 10], dtype=tf.complex128)
```



Initialization of tensors



$T \in \mathbb{R}^{2 \times 3 \times 4}, T \sim N(0, I)$

```
1   T = tf.random.normal((2, 3, 4), dtype=tf.float64)
```

$T = I \in \mathbb{C}^{4 \times 4}$

```
1   T = tf.eye(4, dtype=tf.complex128)
```

```
1   T = tf.constant([1, 2, 5, 10], dtype=tf.complex128)
```

etc

$\check{\hbar}$

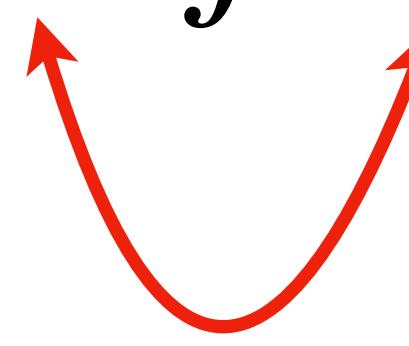
Transpose tensors

$$T_{i \ j \ k \ l}$$

$\begin{bmatrix} \dot{q} \end{bmatrix}$

\check{h}

Transpose tensors

$$T_{i \ j \ k \ l}$$


\dot{g}

$\check{\hbar}$

Transpose tensors

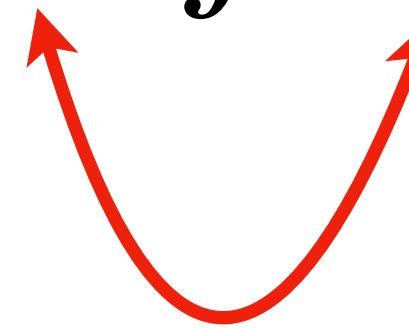


\check{g}

$\check{\hbar}$

Transpose tensors

$$T_{i \ j \ k \ l} \longrightarrow T_{k \ j \ i \ l}$$



$$\begin{array}{c} i \\ j \\ k \\ l \end{array} \longleftrightarrow \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array}$$

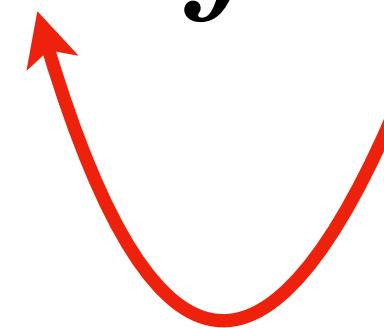
\check{g}

h

Transpose tensors

g

$$T_{i \ j \ k \ l} \longrightarrow T_{k \ j \ i \ l}$$



$$\begin{array}{c} i \\ j \\ k \\ l \end{array} \longleftrightarrow \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array}$$

$$k \ j \ i \ l$$

```
1     T = tf.transpose(T, (2, 1, 0, 3))
```

h̄

Reshape tensors

$$T \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$

ḡ



Reshape tensors



$$T \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$

$$\underbrace{T_{i\ j\ k\ l}}_q$$

\check{h}

Reshape tensors

\dot{g}

$$T \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$

$$\underbrace{T_{i \ j \ k \ l}}_q \xrightarrow{\hspace{10em}} T_{q \ k \ l} \in \mathbb{R}^{6 \times 3 \times 2}$$



Reshape tensors



$$T \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$

$$\underbrace{T_{i \ j \ k \ l}}_q \xrightarrow{\hspace{10em}} T_{q \ k \ l} \in \mathbb{R}^{6 \times 3 \times 2}$$

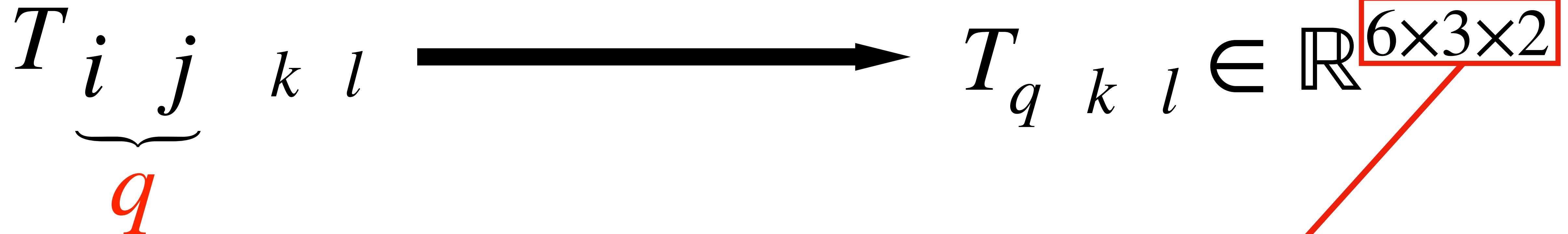
```
1     T = tf.reshape(T, (6, 3, 2))
```

h

Reshape tensors

g

$$T \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$



```
1     T = tf.reshape(T, (6, 3, 2))
```

h

Reshape tensors

g

$$T \in \mathbb{R}^{6 \times 3 \times 2}$$

$$T_{q \ k \ l} \xrightarrow{\quad} T_{i \ j \ k \ l} \in \mathbb{R}^{3 \times 2 \times 3 \times 2}$$

```
1   T = tf.reshape(T, (3, 2, 3, 2))
```

Partial density matrices

Bell state initialization

$$|\psi\rangle = (|1\rangle \otimes |1\rangle + |0\rangle \otimes |0\rangle)$$

```
1    psi = tf.constant([1, 0, 0, 1], dtype=tf.complex128)
```

Partial density matrices

Bell state initialization

$$|\psi\rangle = (|1\rangle \otimes |1\rangle + |0\rangle \otimes |0\rangle)$$

```
1    psi = tf.constant([1, 0, 0, 1], dtype=tf.complex128)
```

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|1\rangle \otimes |1\rangle + |0\rangle \otimes |0\rangle)$$

```
1    psi = psi / tf.linalg.norm(psi)
```

Partial density matrices

Bell state initialization

$$|\psi\rangle = (|1\rangle \otimes |1\rangle + |0\rangle \otimes |0\rangle)$$

```
1    psi = tf.constant([1, 0, 0, 1], dtype=tf.complex128)
```

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|1\rangle \otimes |1\rangle + |0\rangle \otimes |0\rangle)$$

```
1    psi = psi / tf.linalg.norm(psi)
```

$$\rho = |\psi\rangle\langle\psi|$$

```
1    rho = tf.tensordot(psi, tf.math.conj(psi), axes=0)
```



Partial density matrices



$$\rho_{i\,j}$$

$\check{\hbar}$

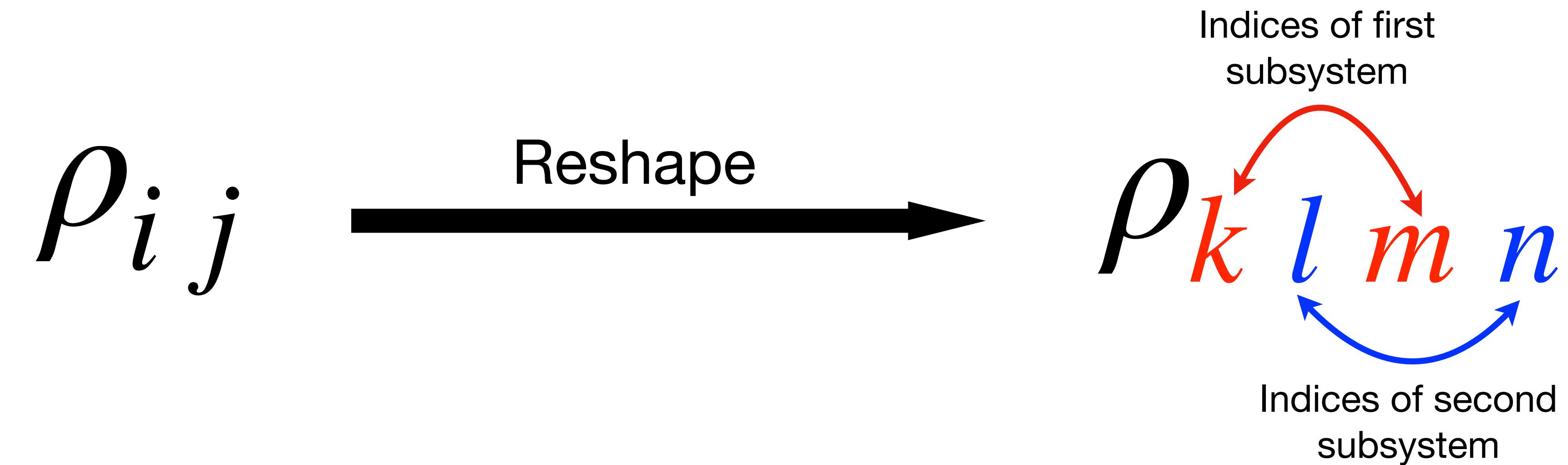
Partial density matrices

\check{g}

$$\rho_{ij} \xrightarrow{\text{Reshape}} \rho_{\underbrace{k \ l}_{i} \ \underbrace{m \ n}_{j}}$$

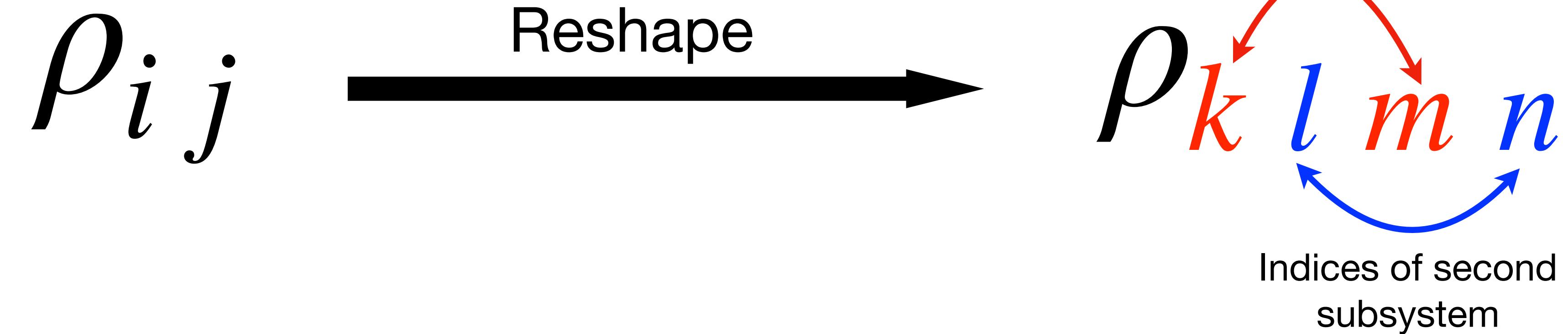


Partial density matrices





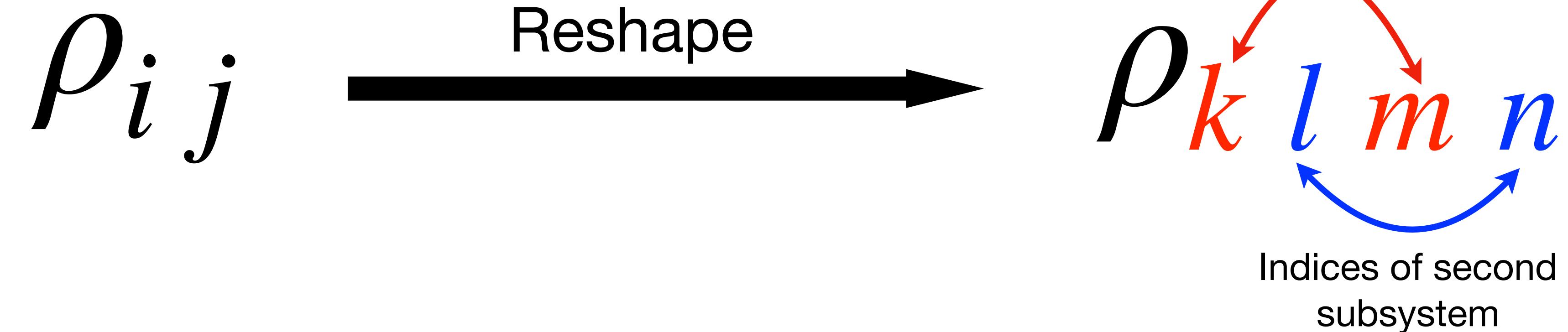
Partial density matrices



```
1   rho_resh = tf.reshape(rho, (2, 2, 2, 2))
```



Partial density matrices



```
1 rho_resh = tf.reshape(rho, (2, 2, 2, 2))
```

$$\rho_{km}^L = \sum_l \rho_{k l m l}$$

```
1 rhoL = tf.einsum('klml->km', rho_resh)
```



Partial density matrices



$$\rho_{ij}$$

Reshape

$$\rho_{\color{red}k \ l \ m \ n}$$

Indices of first subsystem

Indices of second subsystem

```
1 rho_resh = tf.reshape(rho, (2, 2, 2, 2))
```

$$\rho_{km}^L = \sum_l \rho_{\color{red}k \ l \ m \ l}$$

```
1 rhoL = tf.einsum('klml->km', rho_resh)
```

$$\rho_{ln}^R = \sum_k \rho_{\color{red}k \ l \ k \ n}$$

```
1 rhoR = tf.einsum('klkn->ln', rho_resh)
```



Partial density matrices



```
1 rhoL = tf.einsum('klml->km', rho_resh)
```



Partial density matrices



```
1 rhoL = tf.einsum('klml->km', rho_resh)
```



Sum over
repeated
indices



Partial density matrices



```
1 rhoL = tf.einsum('klml->km', rho_resh)
```

Remaining
indices

Sum over
repeated
indices



Partial density matrices



```
1 rhoL = tf.einsum('klml->km', rho_resh)
```

Remaining
indices



Sum over
repeated
indices

```
1 print(rhoL)

tf.Tensor(
[[0.5+0.j 0. +0.j]
 [0. +0.j 0.5+0.j]], shape=(2, 2), dtype=complex128)
```

```
1 print(rhoR)

tf.Tensor(
[[0.5+0.j 0. +0.j]
 [0. +0.j 0.5+0.j]], shape=(2, 2), dtype=complex128)
```



Partial density matrices



```
1 rhoL = tf.einsum('klml->km', rho_resh)
```

$$\rho^L = \rho^R = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

Remaining
indices



Sum over
repeated
indices

```
1 print(rhoL)

tf.Tensor(
[[0.5+0.j 0. +0.j]
 [0. +0.j 0.5+0.j]], shape=(2, 2), dtype=complex128)
```

```
1 print(rhoR)

tf.Tensor(
[[0.5+0.j 0. +0.j]
 [0. +0.j 0.5+0.j]], shape=(2, 2), dtype=complex128)
```

h

Slicing

g

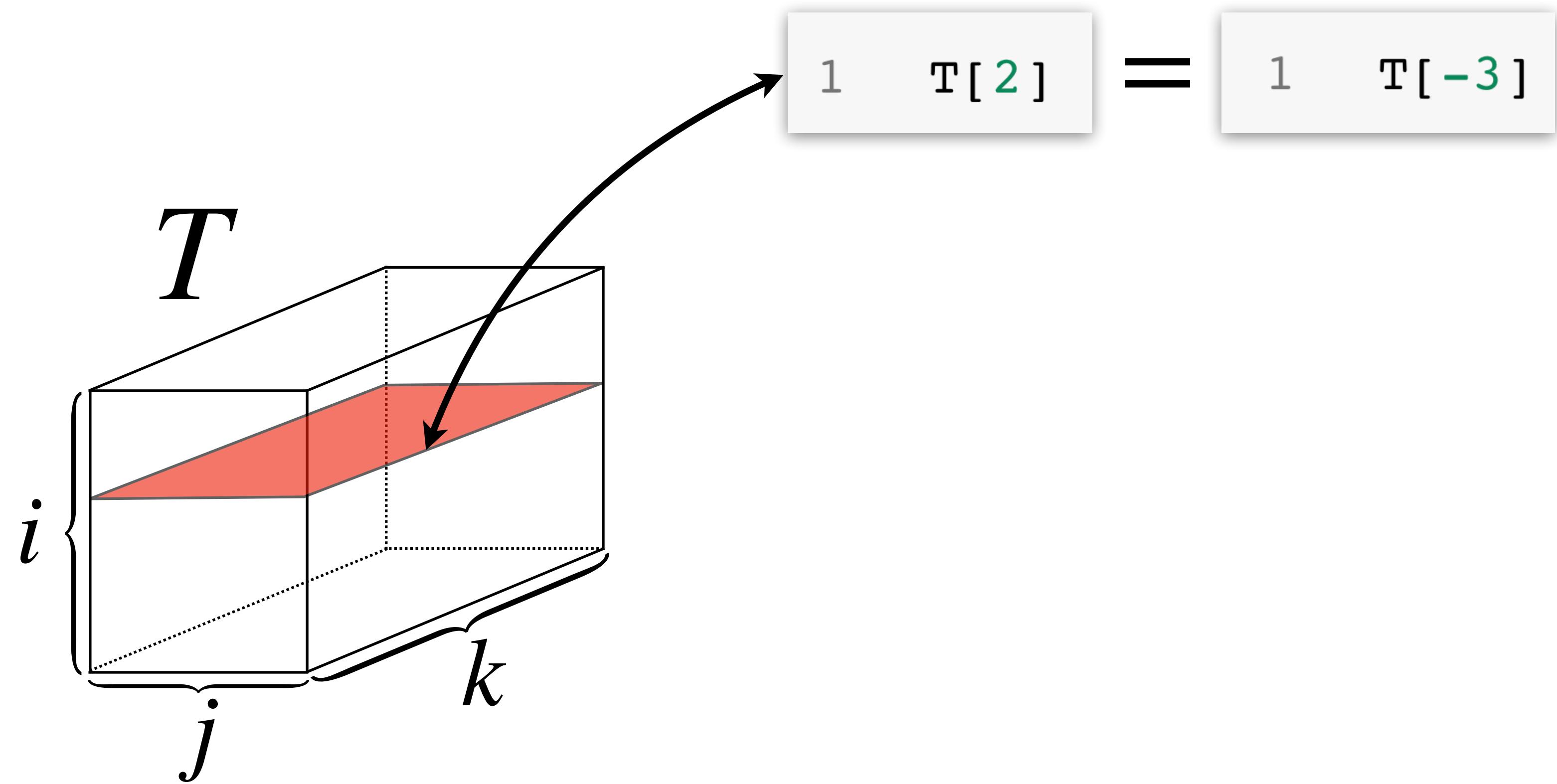
$$T \in \mathbb{R}^{5 \times 3 \times 8}$$



Slicing



$$T \in \mathbb{R}^{5 \times 3 \times 8}$$

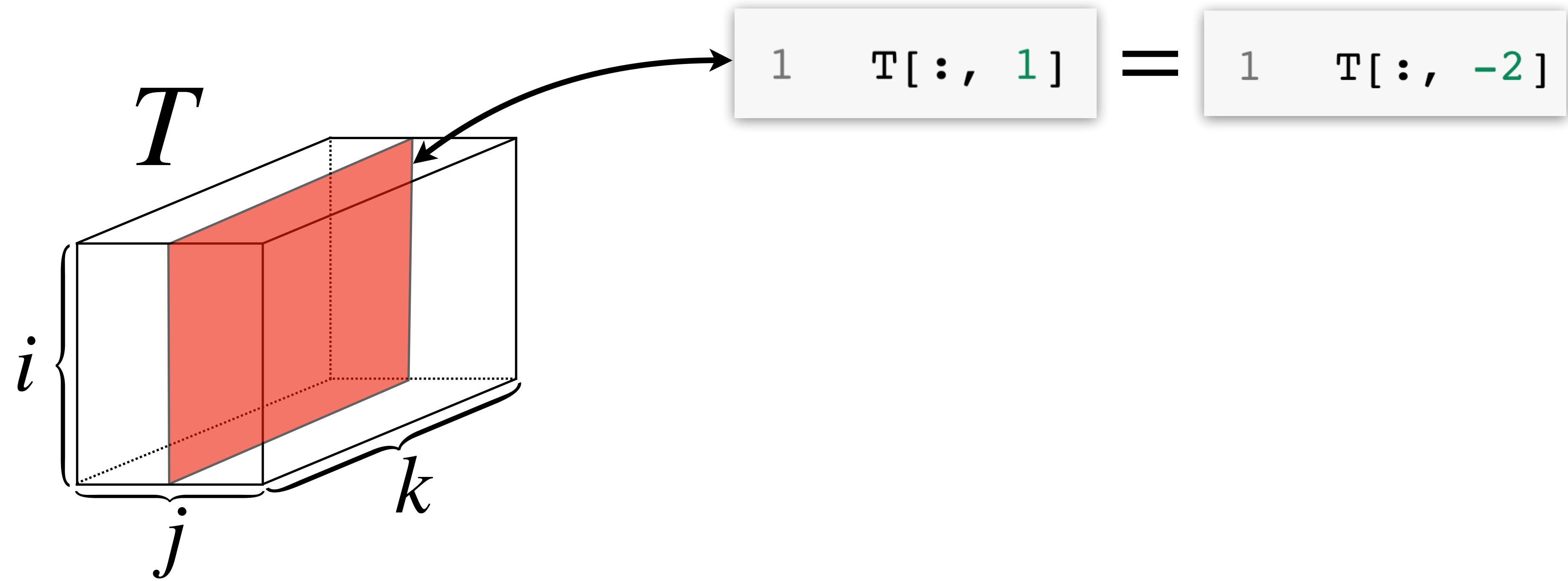




Slicing



$$T \in \mathbb{R}^{5 \times 3 \times 8}$$

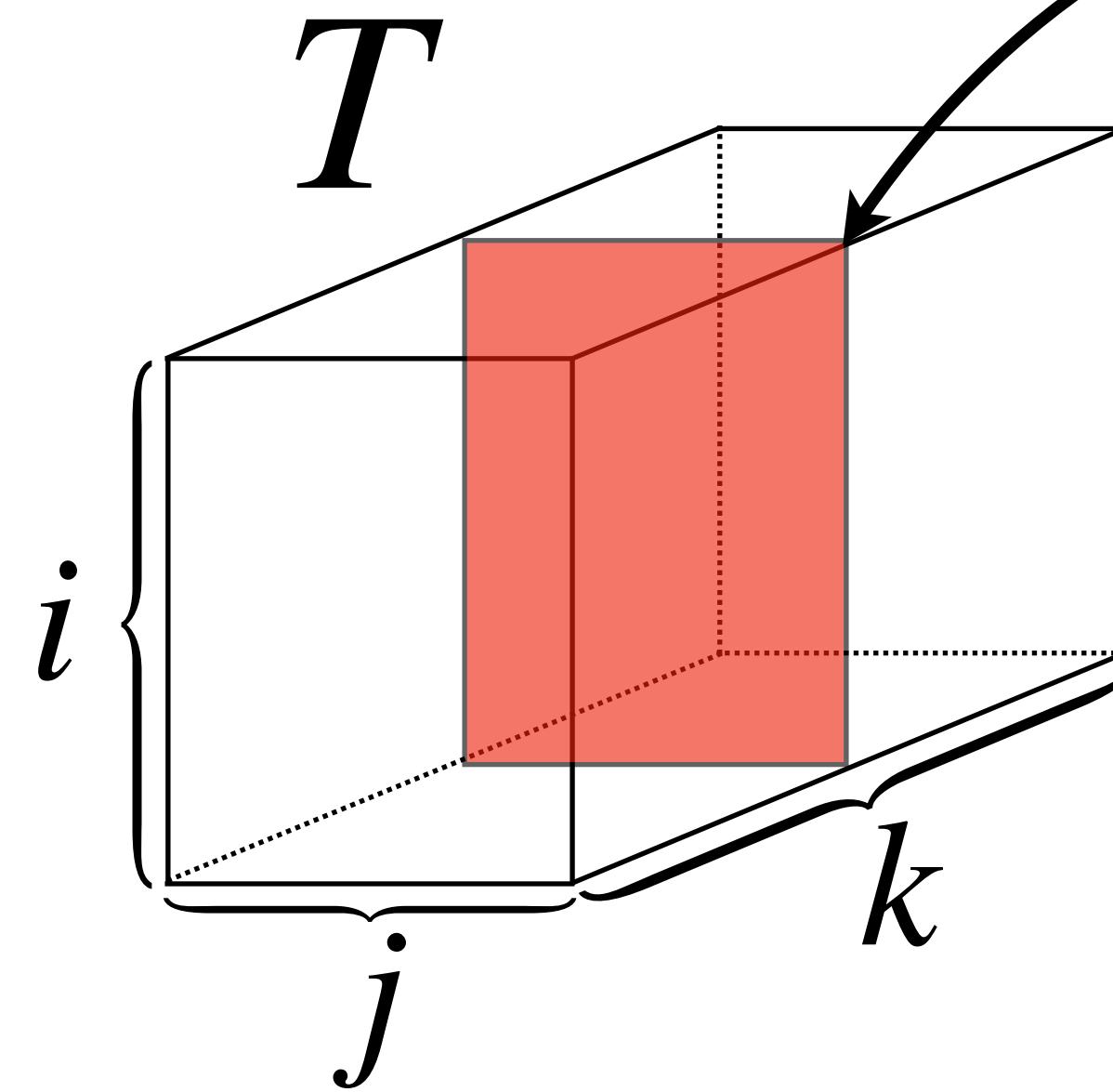




Slicing



$$T \in \mathbb{R}^{5 \times 3 \times 8}$$



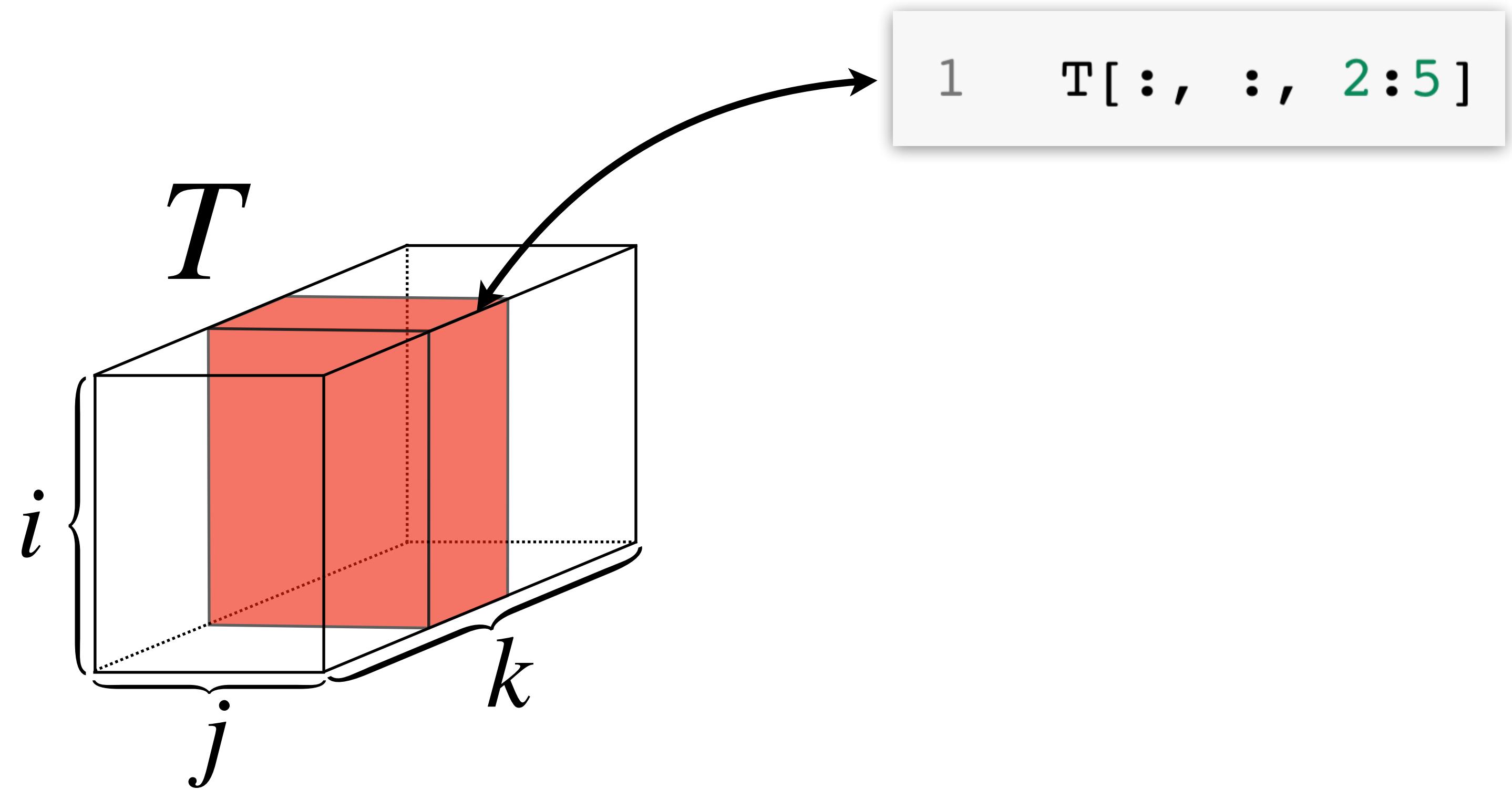
1 $T[:, :, 4]$ = 1 $T[:, :, -4]$



Slicing



$$T \in \mathbb{R}^{5 \times 3 \times 8}$$





Element-wise operations



If the shape of D is equal to the shape of E

```
1 D * E # element wise multiplication
2 D + E # element wise summation
3 D - E # element wise subtraction
4 D / E # element wise division
5 D ** E # element wise exponentiation
6 D > E # element wise comparison, returns tensor of type bool
7 D < E # element wise comparison, returns tensor of type bool
8 D == E # element wise comparison, returns tensor of type bool
9 D != E # element wise comparison, returns tensor of type bool
```

etc



Element-wise operations



```
1 tf.math.exp(D)  # element wise exponent  
2 tf.math.log(D)  # element wise logarithm  
3 tf.math.atan(D) # element wise atan  
4 tf.math.sqrt(D) # element wise sqrt
```

etc



Linear algebra



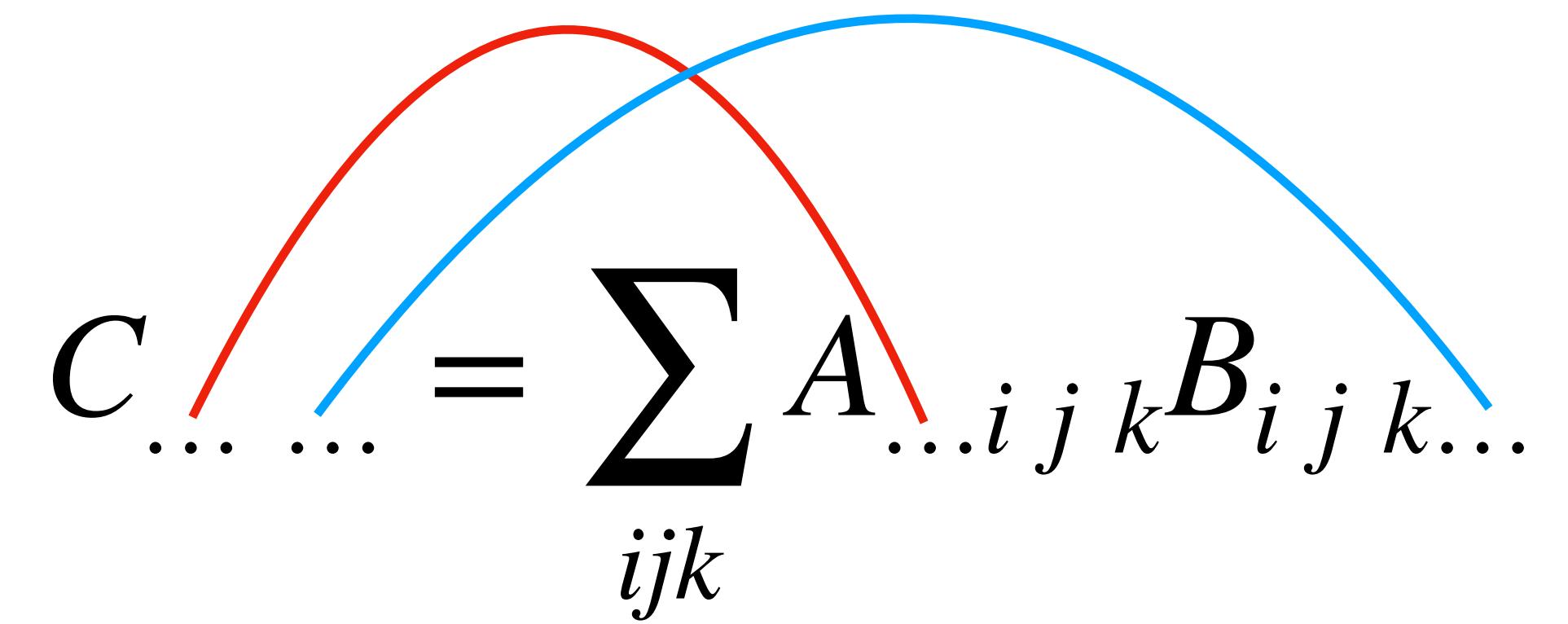
```
1 Q, R = tf.linalg.qr(K) # QR decomposition of a matrix
2 S, U, V = tf.linalg.svd(K) # SVD decomposition of a matrix
3 eigvals, right_eigvecs = tf.linalg.eig(K) # eigen decomposition of a matrix
4 expC = tf.linalg.expm(K) # matrix exponent
5 C_inv = tf.linalg.inv(K) # inverse matrix
6 K_adj = tf.linalg.adjoint(K) # conjugate transpose matrix
7 det = tf.linalg.det(K) # determinant of a matrix
8 sqrt_K = tf.linalg.sqrtm(K) # matrix sqrt
```

etc



Convolutions of tensors



$$C_{\dots \dots} = \sum_{ijk} A_{\dots i j k} B_{i j k \dots}$$




Convolutions of tensors



$$C_{\dots \dots} = \sum_{ijk} A_{\dots i j k} B_{i j k \dots}$$

```
1     C = tf.tensordot(A, B, axes=3)
```



Convolutions of tensors



$$C_{\dots \dots} = \sum A_{\dots i j k} B_{i j k \dots}$$

ijk

Sum over 3
neighboring
indices

```
1   C = tf.tensordot(A, B, axes=3)
```



Convolutions of tensors



$$\sum_{ln} A_{j\ l\ n} B_{n\ i\ l} = C_{i\ j}$$



Convolutions of tensors



$$\sum_{ln} A_{j\ l\ n} B_{n\ i\ l} = C_{i\ j}$$

```
1   C = tf.einsum('jln,nil->ij', A, B)
```



Convolutions of tensors



$$\sum_{ln} A_{j\ l\ n} B_{n\ i\ l} = C_{i\ j}$$

```
1   C = tf.einsum('jln,nil->ij', A, B)
```



Convolutions of tensors

$$\sum_{ln} A_{j\ l\ n} B_{n\ i\ l} = C_{i\ j}$$

```
1   C = tf.einsum('jln, nil->ij', A, B)
```

There is no Red indices there, it means that we summed them out



Matrix multiplication



$$C_{ij} = \sum_k A_{ik}B_{kj}$$

1 $C = A @ B$



Matrix multiplication



$$C_{ij} = \sum_k A_{ik} B_{kj}$$

$$1 \quad C = A @ B$$

But!

$$C_{aij} = \sum_k A_{aik} B_{akj}$$

$$1 \quad C = A @ B$$



Matrix multiplication



$$C_{aij} = \sum_k A_{aik} B_{akj}$$

1 C = A @ B

Only last two
indices are
considered as
matrix indices!



Matrix multiplication



```
1 A = tf.random.normal((2, 3, 4, 5))  
2 B = tf.random.normal((2, 3, 5, 4))  
3 print((A @ B).shape)
```

```
(2, 3, 4, 4)
```



Matrix multiplication



Aren't
touched

```
1 A = tf.random.normal((2, 3, 4, 5))  
2 B = tf.random.normal((2, 3, 5, 4))  
3 print((A @ B).shape)
```

(2, 3, 4, 4)



Matrix multiplication



Aren't
touched mat.
 mul.

```
1 A = tf.random.normal((2, 3, 4, 5))  
2 B = tf.random.normal((2, 3, 5, 4))  
3 print((A @ B).shape)
```

(2, 3, 4, 4)

```
1 A = tf.random.normal((100, 100, 5, 5))  
2 s, u, v = tf.linalg.svd(A)  
3 print(s.shape)  
4 print(u.shape)  
5 print(v.shape)
```

```
(100, 100, 5)  
(100, 100, 5, 5)  
(100, 100, 5, 5)
```



Broadcasting



```
1 A = tf.random.normal((3, 2, 1, 5, 6))  
2 B = tf.random.normal((3, 1, 4, 5, 1))  
3 print((A + B).shape)
```

```
(3, 2, 4, 5, 6)
```



Broadcasting



Shapes are different

```
1 A = tf.random.normal((3, 2, 1, 5, 6))
2 B = tf.random.normal((3, 1, 4, 5, 1))
3 print((A + B).shape)
```

```
(3, 2, 4, 5, 6)
```



Broadcasting



Shapes are different

```
1 A = tf.random.normal((3, 2, 1, 5, 6))
2 B = tf.random.normal((3, 1, 4, 5, 1))
3 print((A + B).shape)
```

(3, 2, 4, 5, 6)

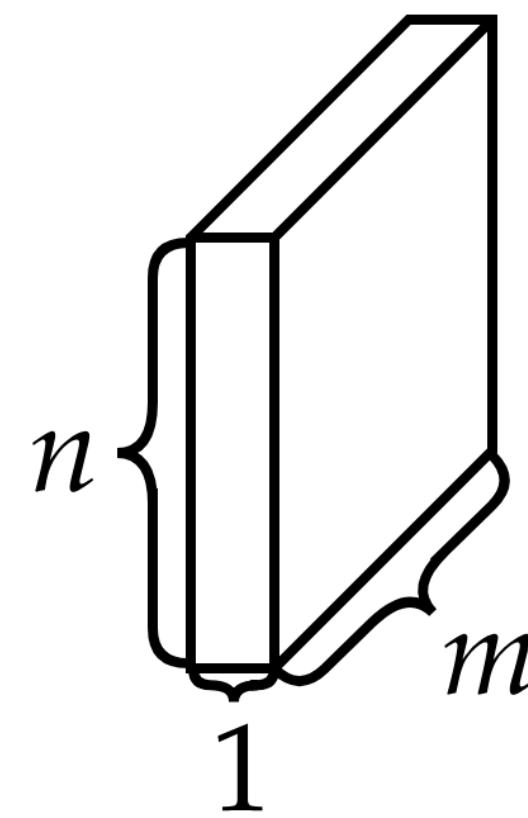
Why don't we have a mistake???



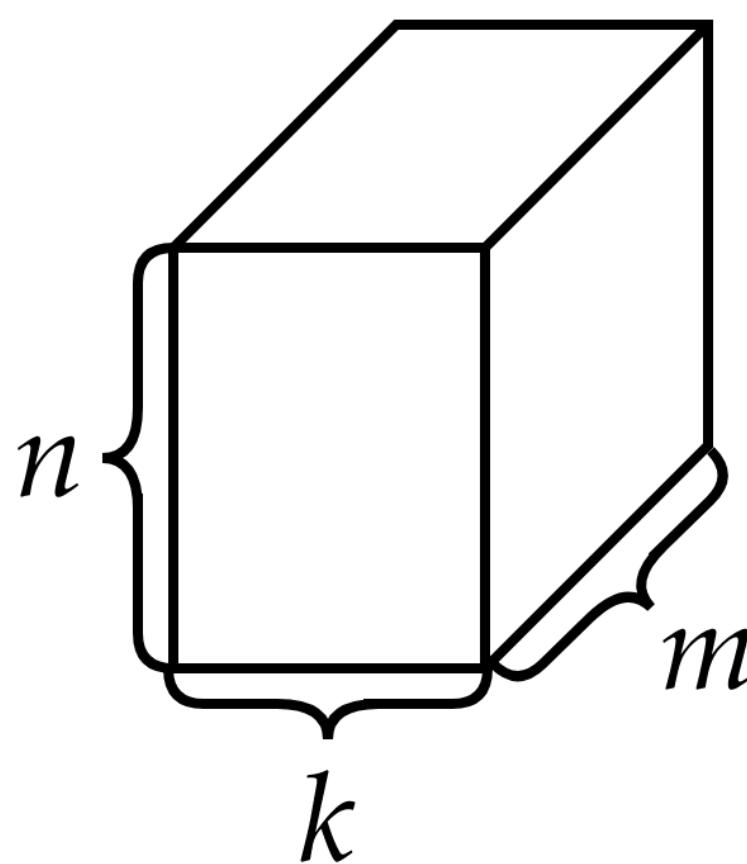
Broadcasting



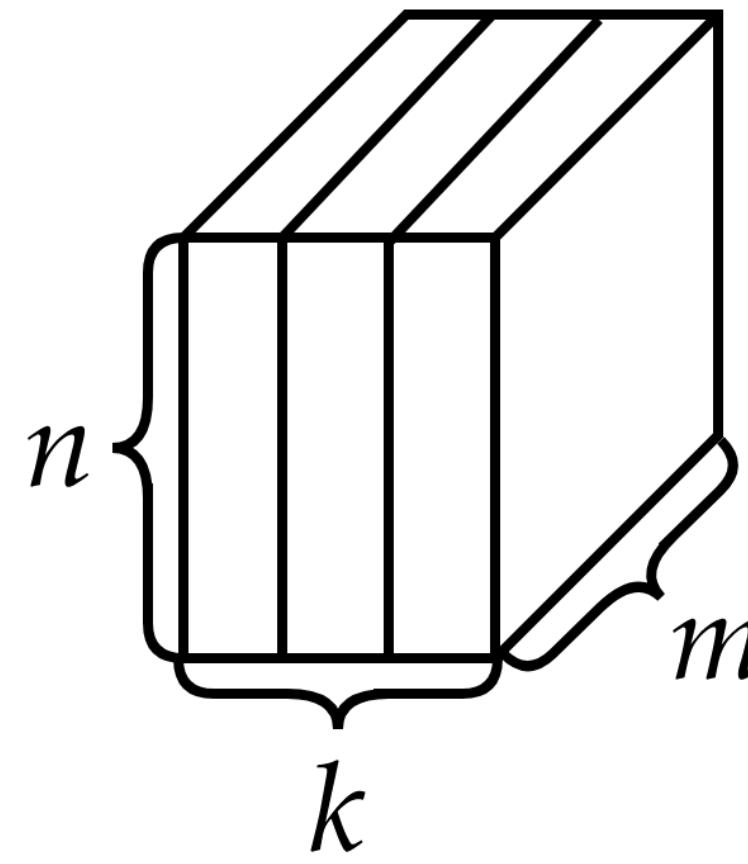
shape = (n, 1, m)



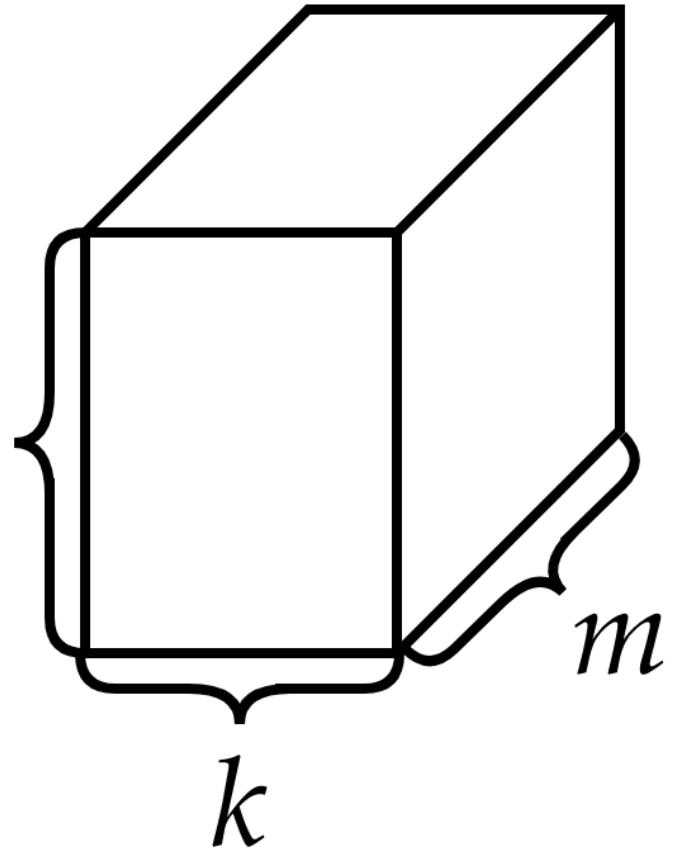
shape = (n, k, m)



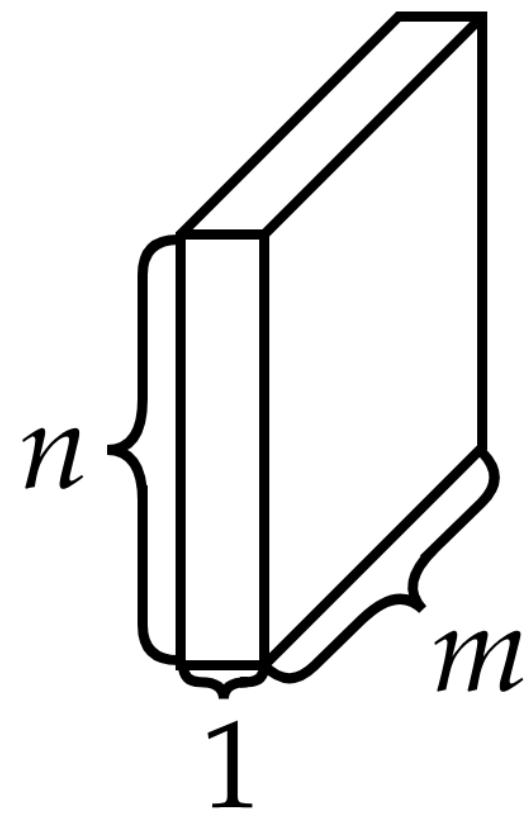
shape = (n, k, m)



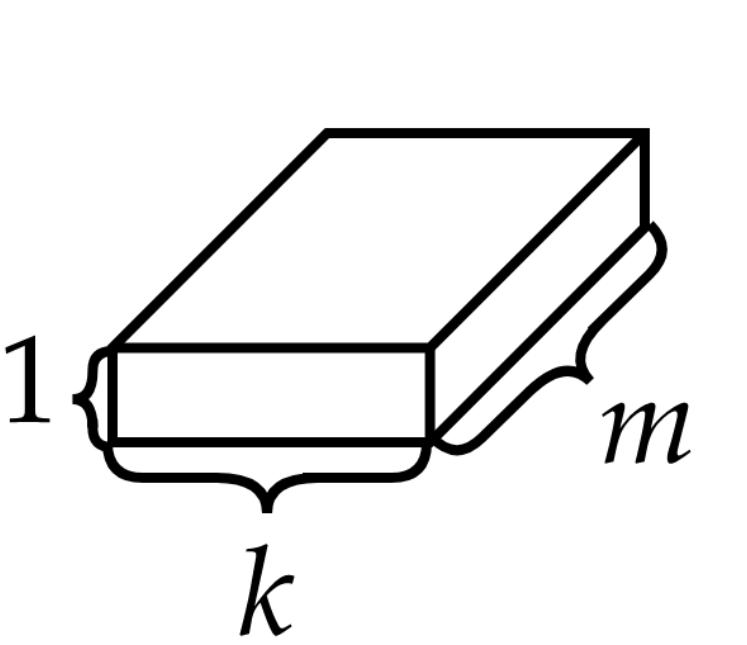
shape = (n, k, m)



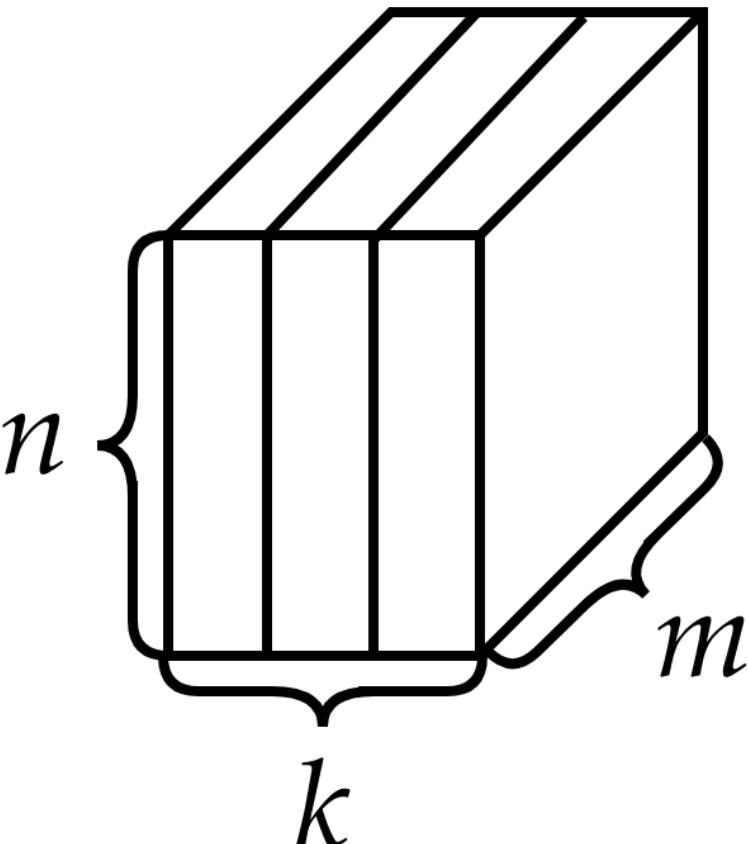
shape = (n, 1, m)



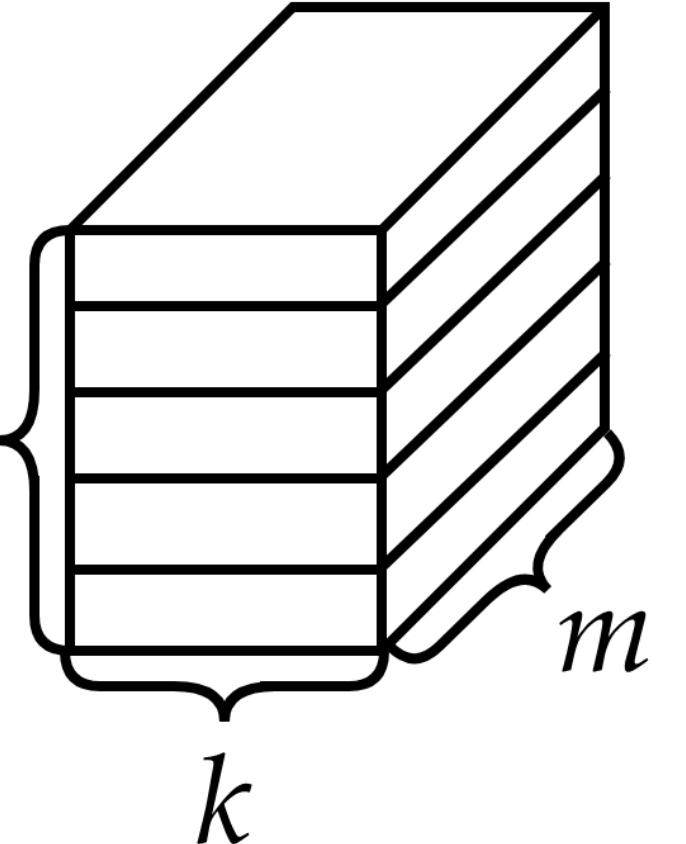
shape = (1, k, m)



shape = (n, k, m)



shape = (n, k, m)





Broadcasting



```
1 A = tf.random.normal((3, 2, 1, 5, 6))  
2 B = tf.random.normal((3, 1, 4, 5, 1))  
3 print((A + B).shape)
```

(3, 2, 4, 5, 6)

Four copies
along this axis

Two copies
along this axis

Six copies
along this axis



Tensor product of Pauli matrices



$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



Tensor product of Pauli matrices



$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
1 # Pauli matrices
2 sigma_x = tf.constant([[0, 1], [1, 0]], dtype=tf.complex128)
3 sigma_y = tf.constant([[0 + 0j, -1j], [1j, 0 + 0j]], dtype=tf.complex128)
4 sigma_z = tf.constant([[1, 0], [0, -1]], dtype=tf.complex128)
5
6 # All Pauli matrices in one tensor of shape (3, 2, 2)
7 sigma = tf.concat([sigma_x[tf.newaxis],
8 |                         sigma_y[tf.newaxis],
9 |                         sigma_z[tf.newaxis]], axis=0)
```



Tensor product of Pauli matrices

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Want to calculate

$$\sigma_x \otimes \sigma_x, \sigma_y \otimes \sigma_y, \sigma_z \otimes \sigma_z$$

```
1 # Pauli matrices
2 sigma_x = tf.constant([[0, 1], [1, 0]], dtype=tf.complex128)
3 sigma_y = tf.constant([[0 + 0j, -1j], [1j, 0 + 0j]], dtype=tf.complex128)
4 sigma_z = tf.constant([[1, 0], [0, -1]], dtype=tf.complex128)
5
6 # All Pauli matrices in one tensor of shape (3, 2, 2)
7 sigma = tf.concat([sigma_x[tf.newaxis],
8 |                         sigma_y[tf.newaxis],
9 |                         sigma_z[tf.newaxis]], axis=0)
```



Tensor product of Pauli matrices



Auxiliary indices, new shape is (3, 2, 2, 1, 1)

```
1 left_sigma = sigma[:, :, :, tf.newaxis, tf.newaxis]
```



Tensor product of Pauli matrices

```
1 left_sigma = sigma[:, :, :, tf.newaxis, tf.newaxis]
```

Auxiliary indices, new shape is (3, 2, 2, 1, 1)

```
2 right_sigma = sigma[:, tf.newaxis, tf.newaxis]
```

Auxiliary indices, new shape is (3, 1, 1, 2, 2)



Tensor product of Pauli matrices

```
1 left_sigma = sigma[:, :, :, tf.newaxis, tf.newaxis]
```

Auxiliary indices, new shape is (3, 2, 2, 1, 1)

```
2 right_sigma = sigma[:, tf.newaxis, tf.newaxis]
```

Auxiliary indices, new shape is (3, 1, 1, 2, 2)

```
3 tensor_prod = right_sigma * left_sigma
```

Broadcasting, new shape is (3, 2, 2, 2, 2)



Tensor product of Pauli matrices

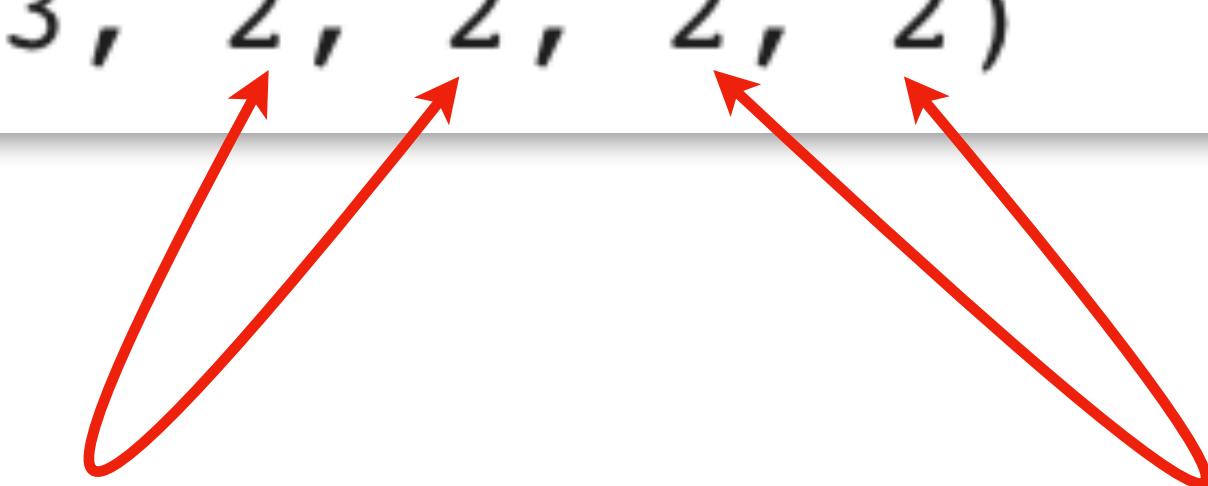


```
1 print(tensor_prod.shape)
```

```
(3, 2, 2, 2, 2)
```

Indices of a first
Pauli matrix

Indices of a second
Pauli matrix





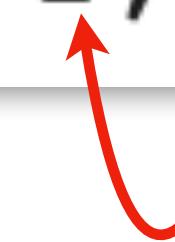
Tensor product of Pauli matrices



```
1 print(tensor_prod.shape)
```

```
(3, 2, 2, 2, 2)
```

Let us swap these
two indices





Tensor product of Pauli matrices

```
1 print(tensor_prod.shape)
```

```
(3, 2, 2, 2, 2)
```



Let us swap these
two indices

```
1 tensor_prod = tf.transpose(tensor_prod, (0, 1, 3, 2, 4))
```



Tensor product of Pauli matrices

```
1 print(tensor_prod.shape)
```

```
(3, 2, 2, 2, 2)
```

Let us swap these
two indices

```
1 tensor_prod = tf.transpose(tensor_prod, (0, 1, 3, 2, 4))
```

Reshape to a set of matrices

```
1 tensor_prod = tf.reshape(tensor_prod, (3, 4, 4))
```



Tensor product of Pauli matrices



```
1 print(tensor_prod[0])  
  
tf.Tensor(  
[[0.+0.j 0.+0.j 0.+0.j 1.+0.j]  
 [0.+0.j 0.+0.j 1.+0.j 0.+0.j]  
 [0.+0.j 1.+0.j 0.+0.j 0.+0.j]  
 [1.+0.j 0.+0.j 0.+0.j 0.+0.j]], shape=(4, 4), dtype=complex128)
```

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} = \sigma_x \otimes \sigma_x$$

```
1 print(tensor_prod[1])  
  
tf.Tensor(  
[[ 0.+0.j  0.-0.j  0.-0.j -1.+0.j]  
 [ 0.+0.j  0.+0.j  1.-0.j  0.-0.j]  
 [ 0.+0.j  1.-0.j  0.+0.j  0.-0.j]  
 [-1.+0.j  0.+0.j  0.+0.j  0.+0.j]], shape=(4, 4), dtype=complex128)
```

$$= \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} = \sigma_y \otimes \sigma_y$$

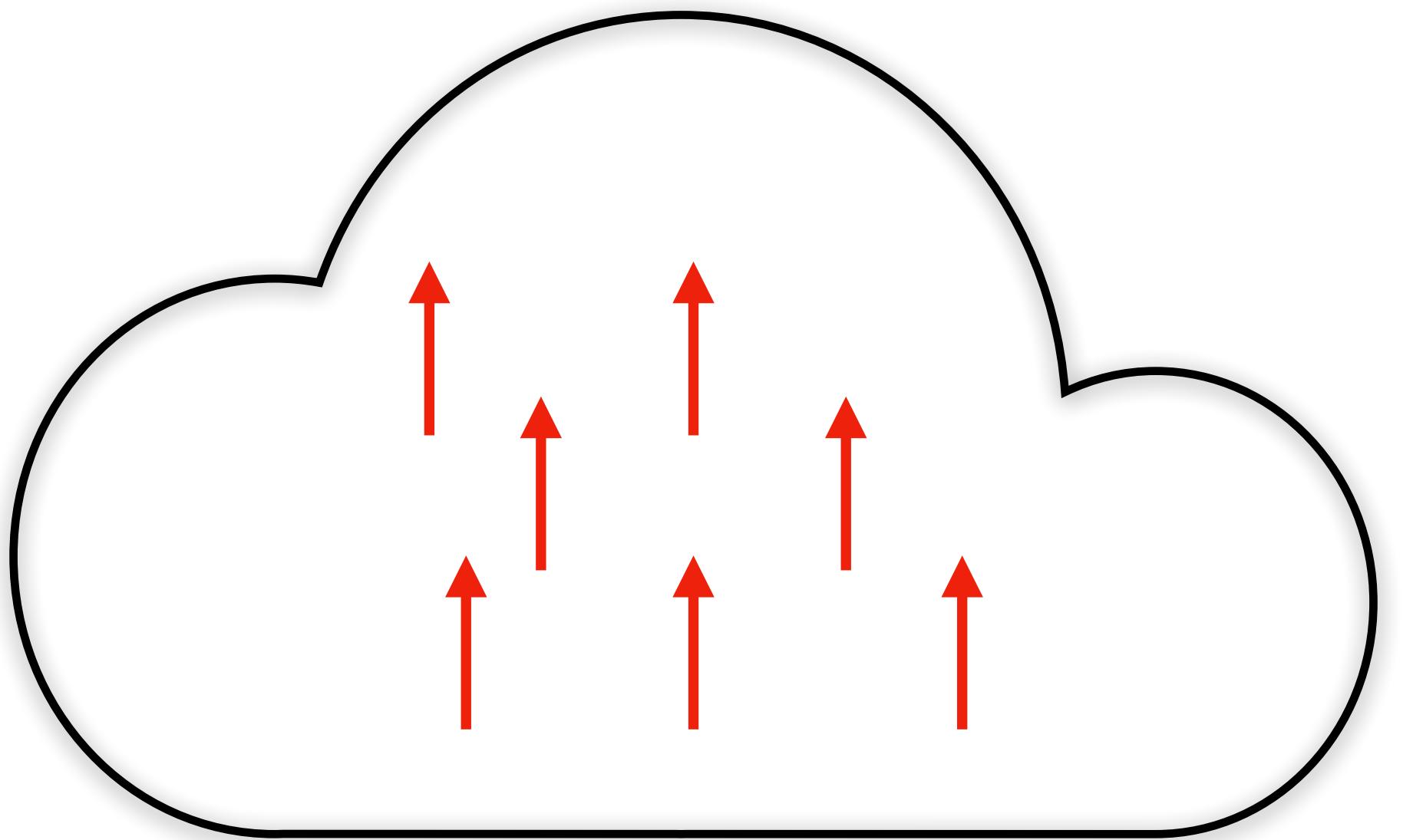
```
1 print(tensor_prod[2])  
  
tf.Tensor(  
[[ 1.+0.j  0.+0.j  0.+0.j  0.+0.j]  
 [ 0.+0.j -1.+0.j  0.+0.j -0.+0.j]  
 [ 0.+0.j  0.+0.j -1.+0.j -0.+0.j]  
 [ 0.+0.j -0.-0.j -0.+0.j  1.-0.j]], shape=(4, 4), dtype=complex128)
```

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \sigma_z \otimes \sigma_z$$

\hbar

Spin echo

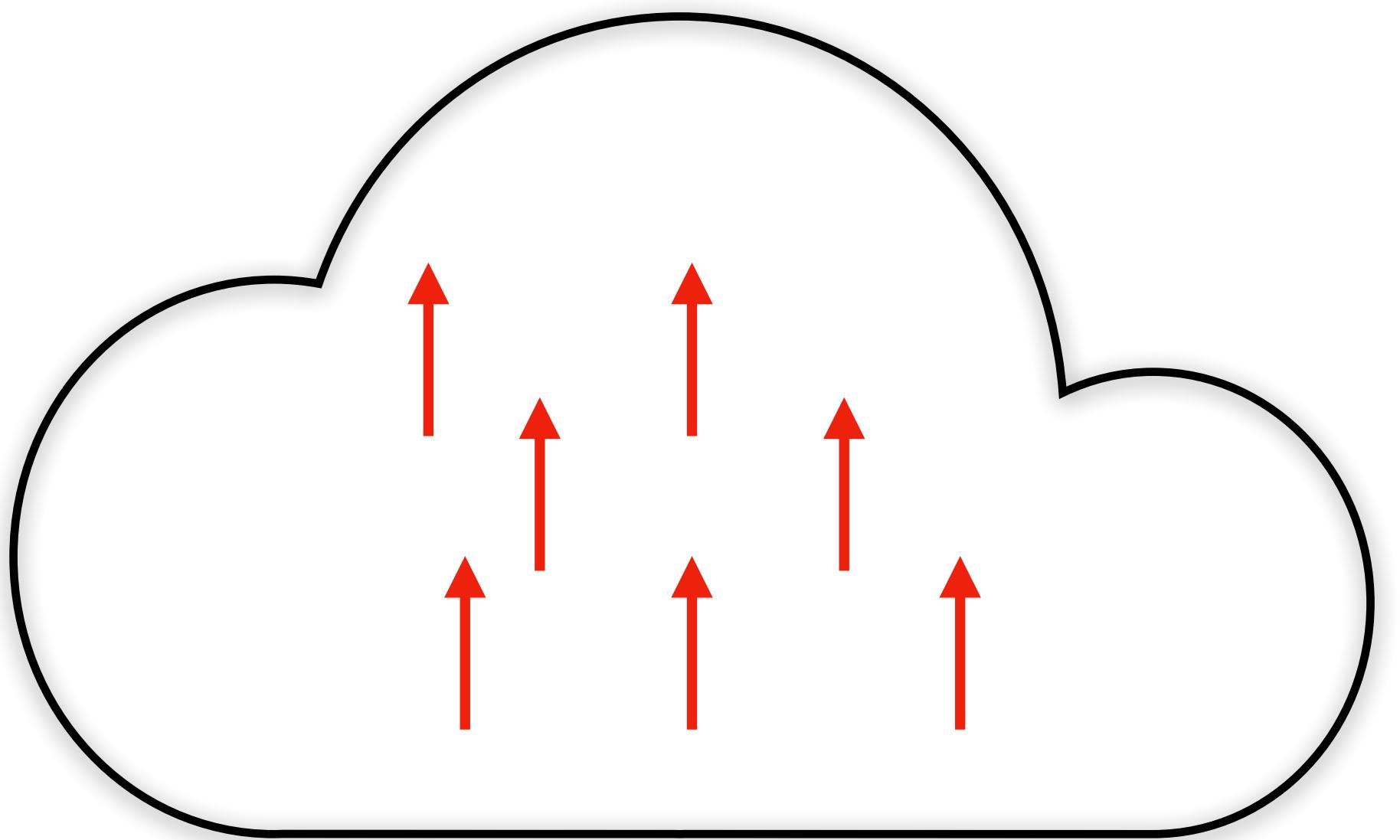
\dot{g}



\hbar

Spin echo

\dot{g}

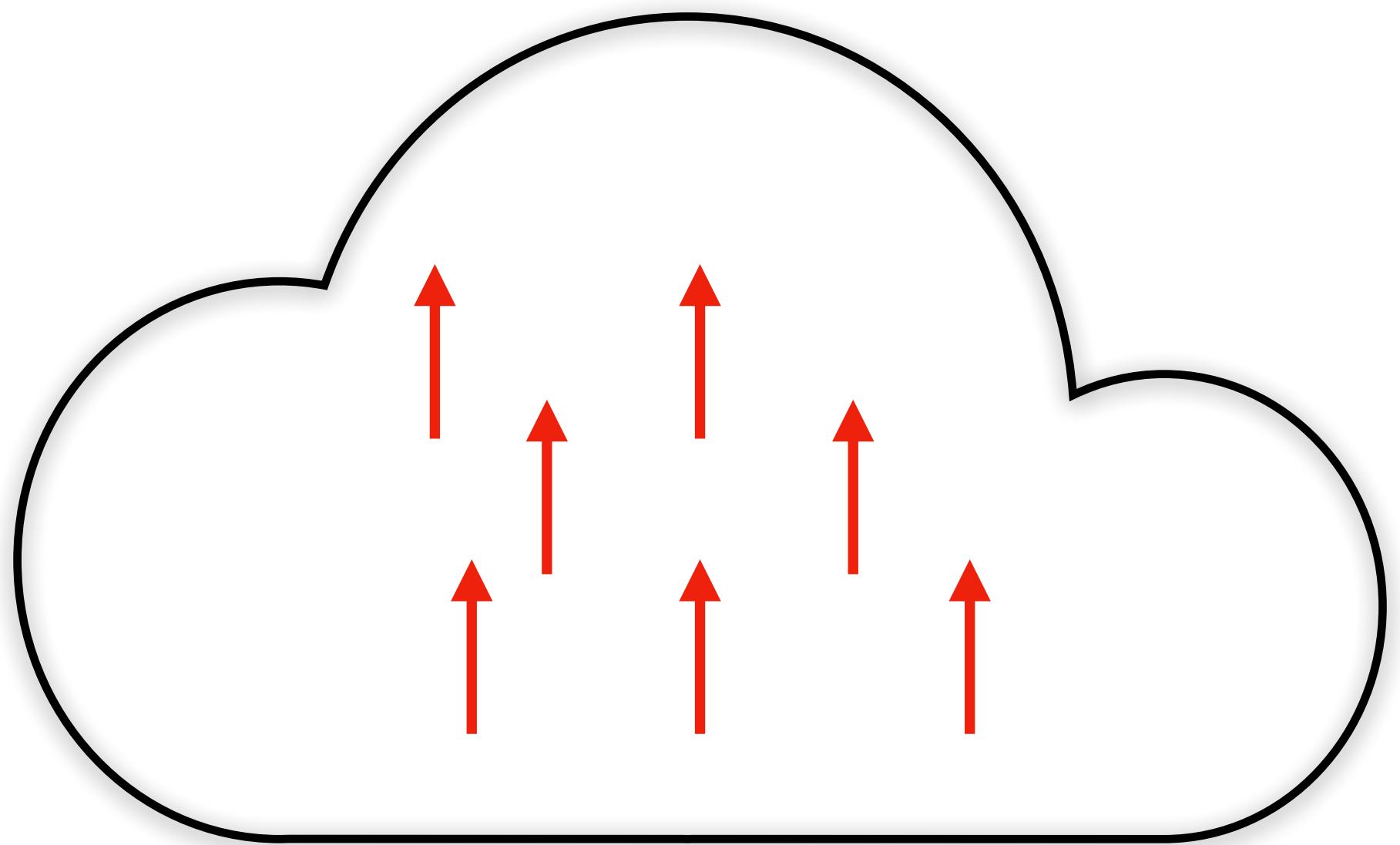


$$|\psi_{\text{in}}\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

\hbar

Spin echo

\dot{q}



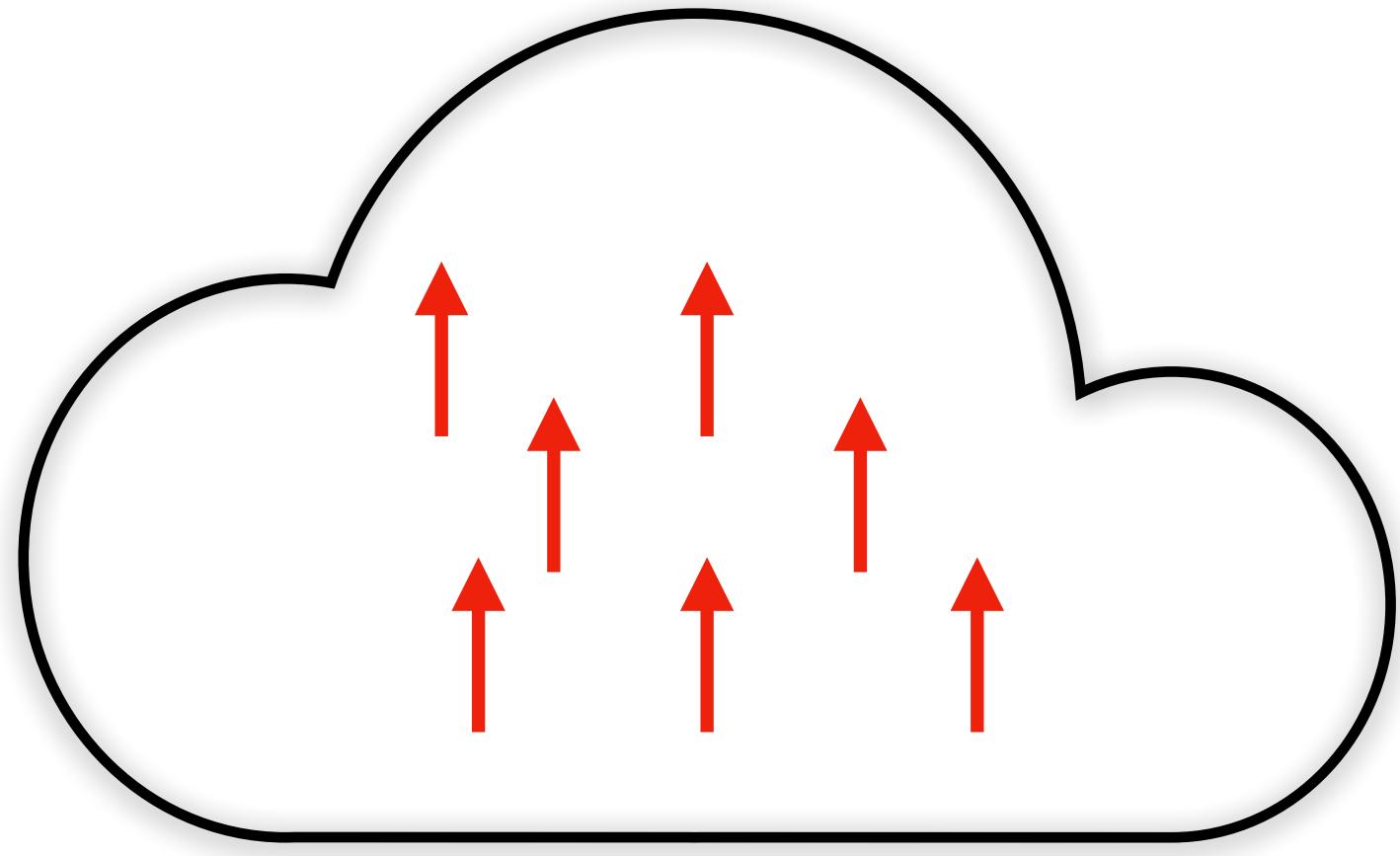
$$H = \epsilon \sigma_z, \quad \epsilon \sim N(0,1)$$

$$|\psi_{\text{in}}\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

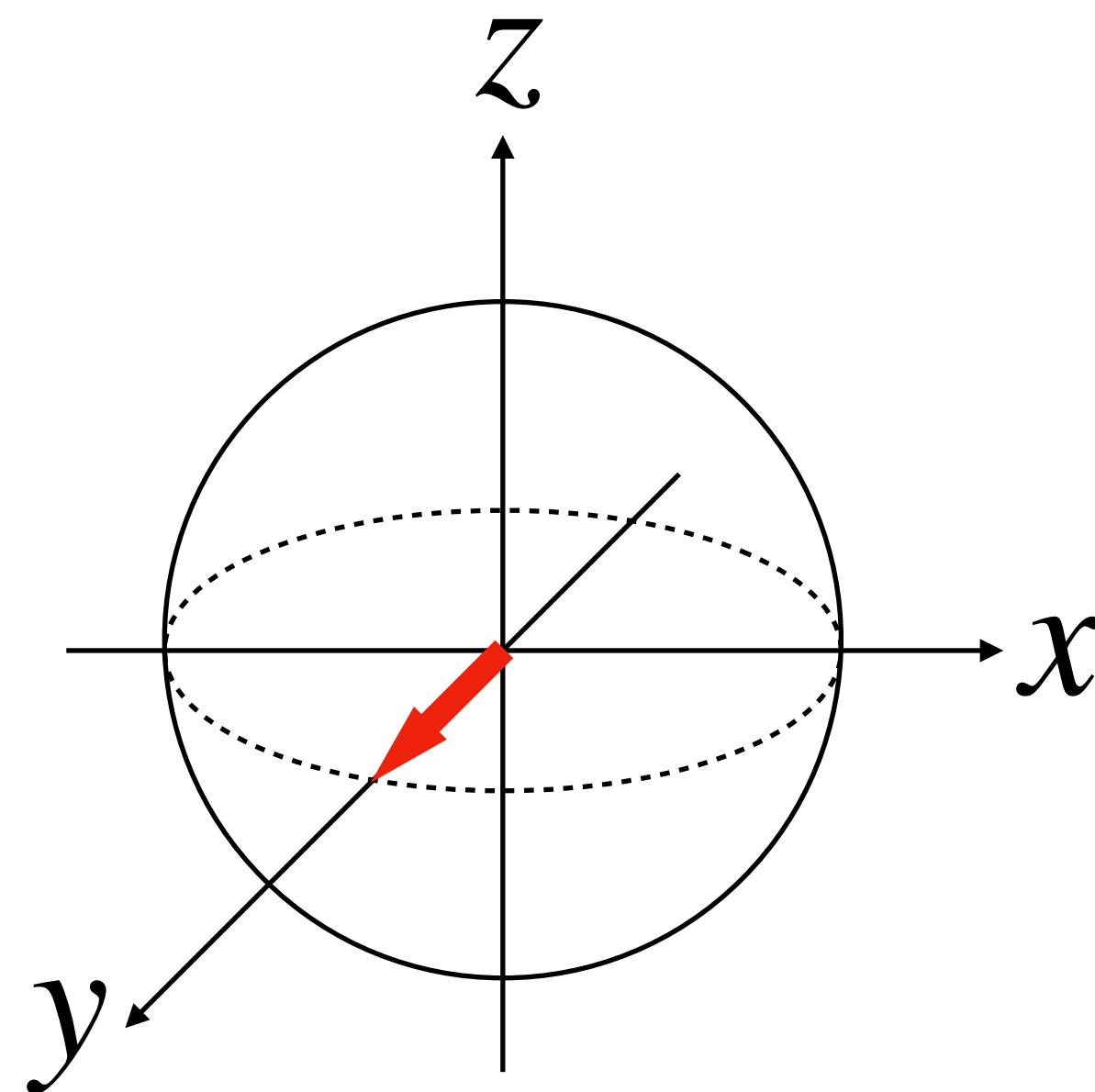
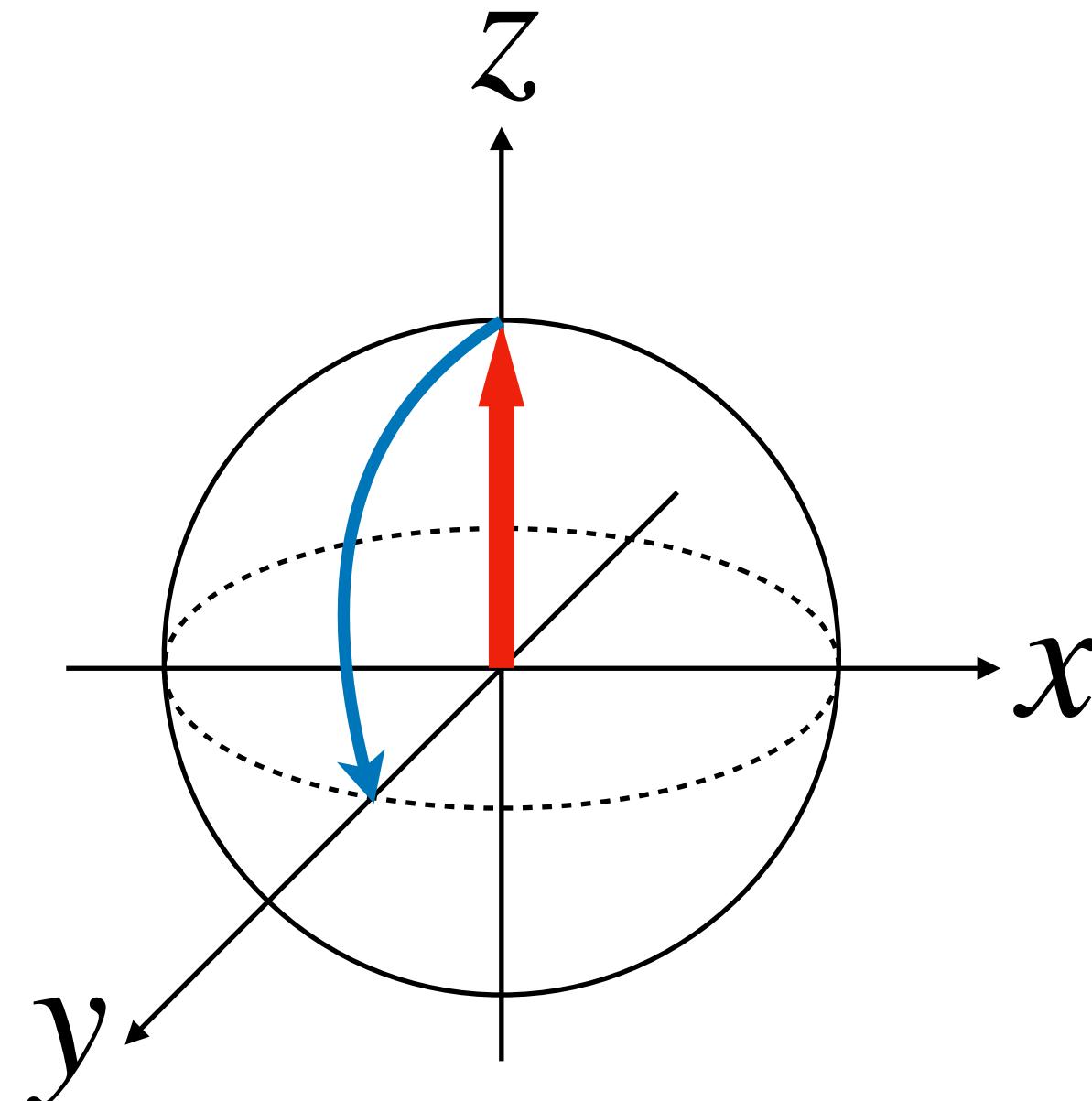
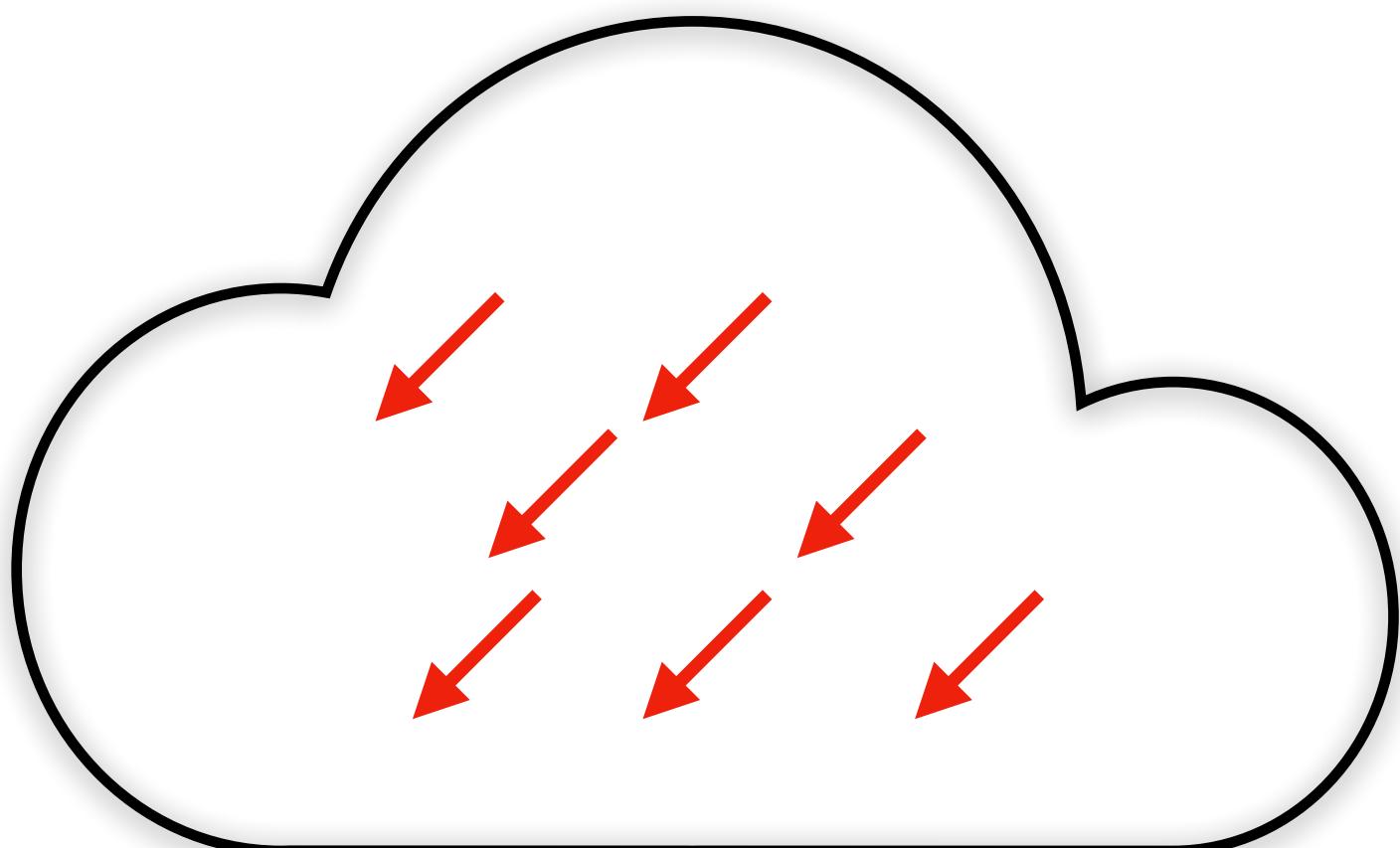
\hbar

Spin echo

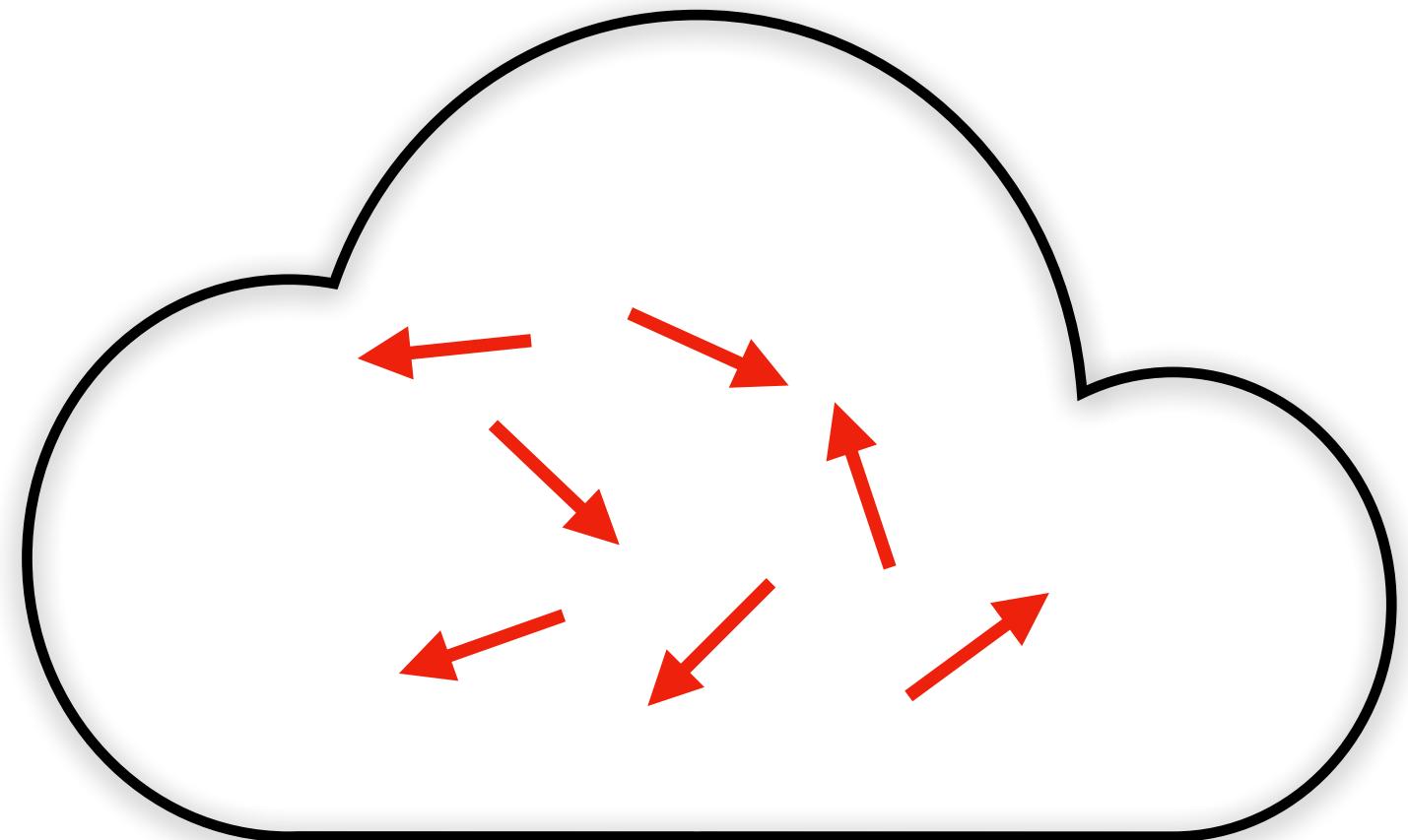
\dot{q}



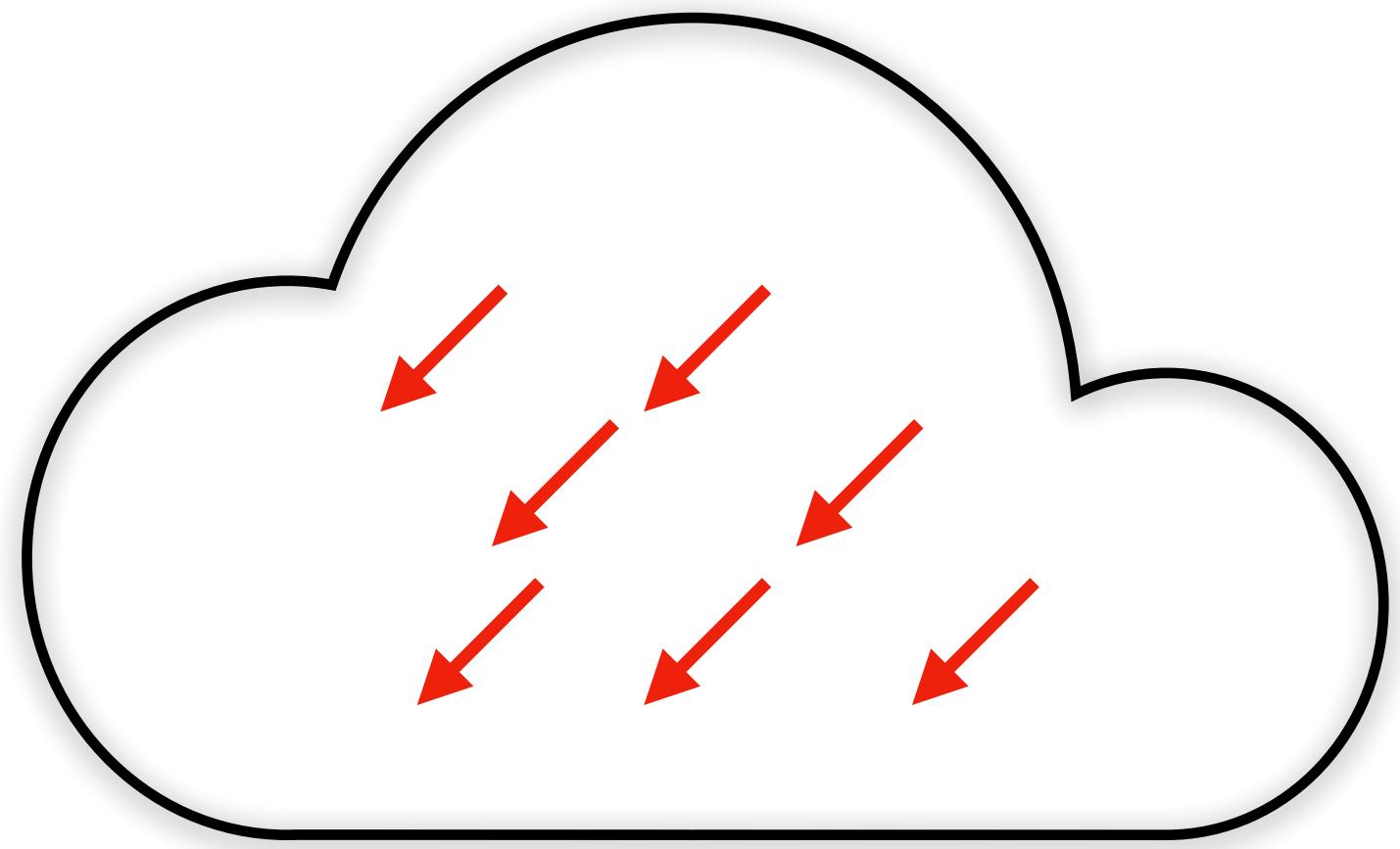
$$H_c = \epsilon \sigma_z - \frac{\pi}{4T_c} \sigma_x$$



\hbar

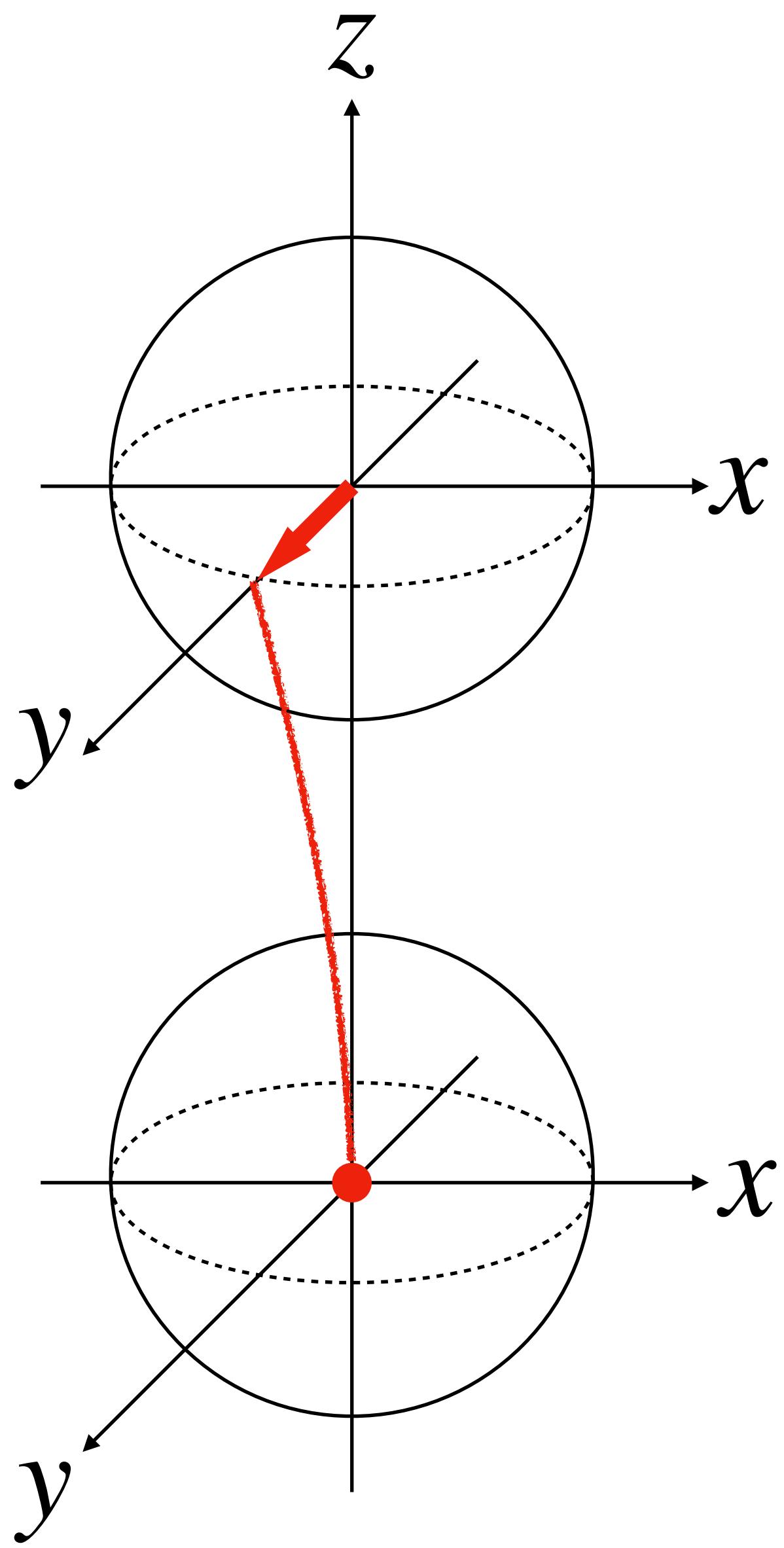


$$H_c = \epsilon \sigma_z$$

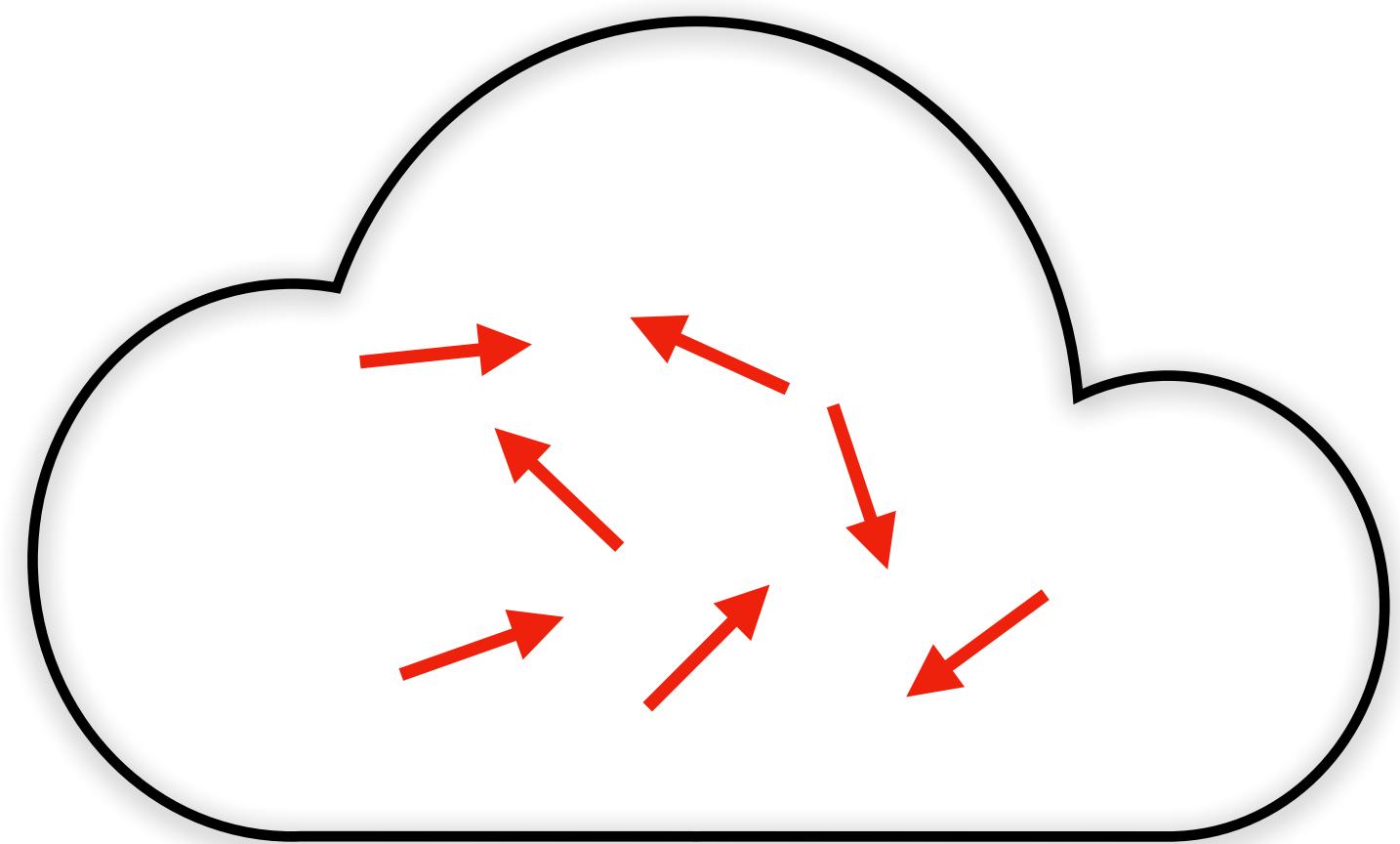


Spin echo

\dot{q}



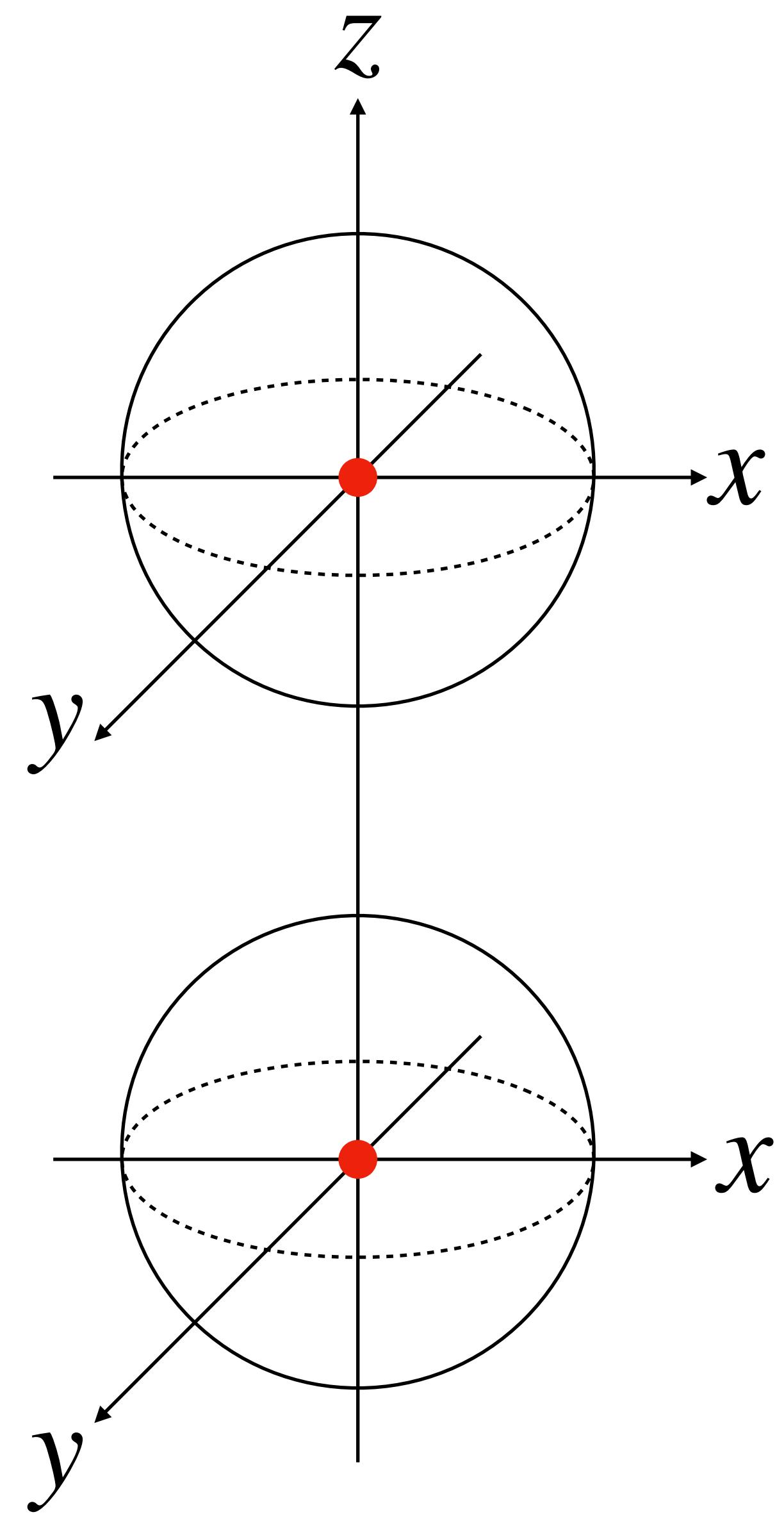
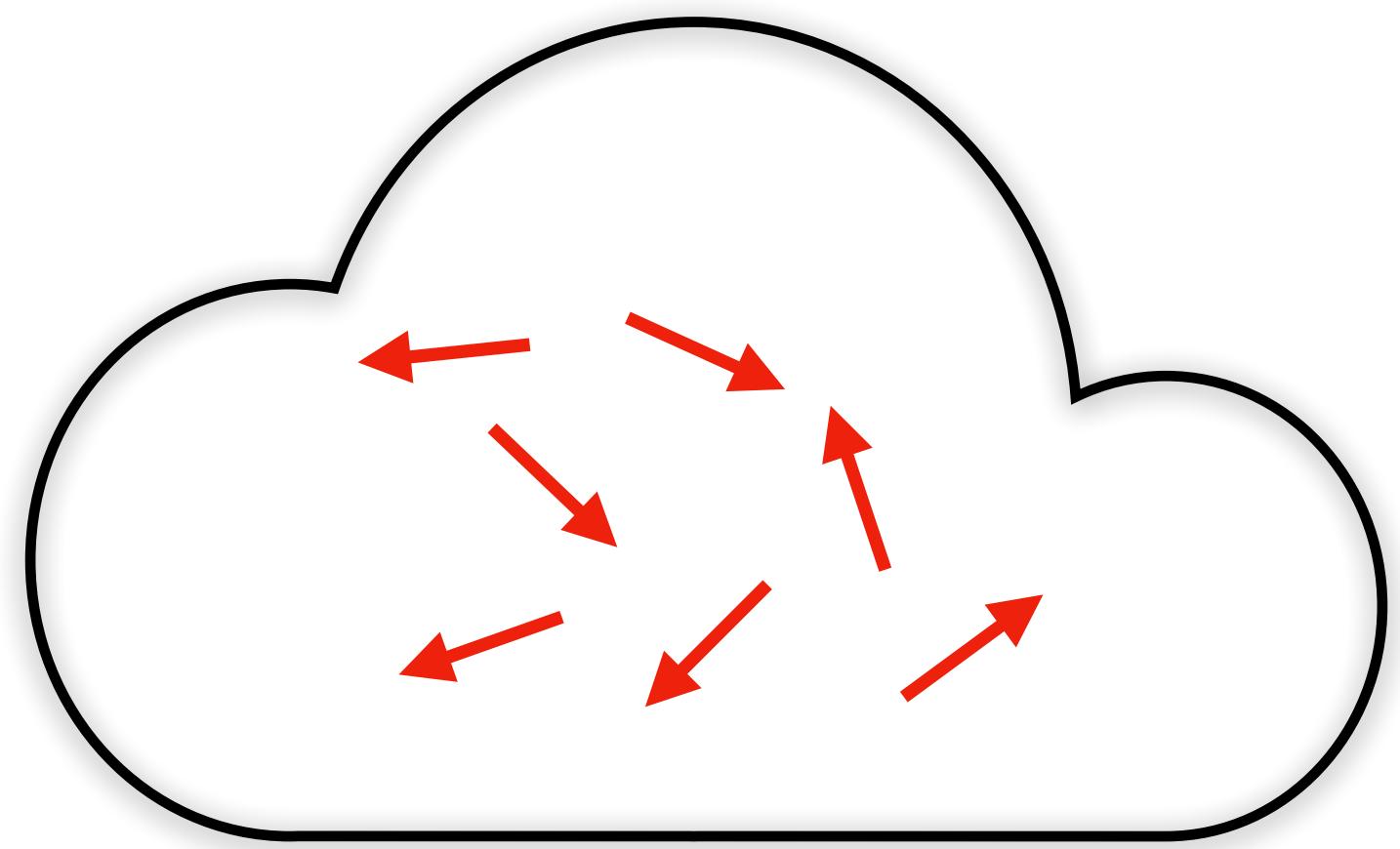
\hbar



$$H_c = \epsilon \sigma_z - \frac{\pi}{2T_c} \sigma_x$$

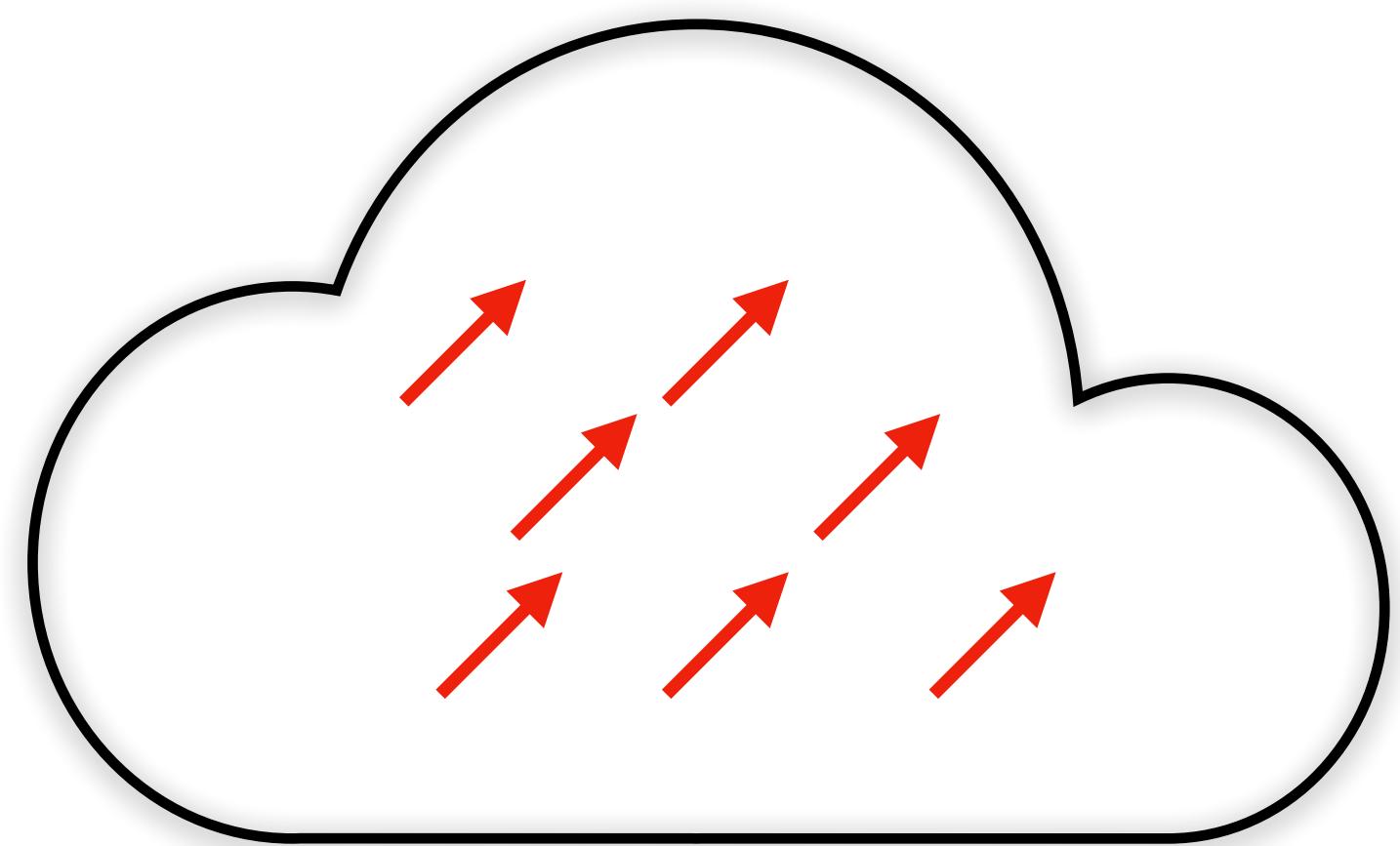
Spin echo

Swap directions
of all spins

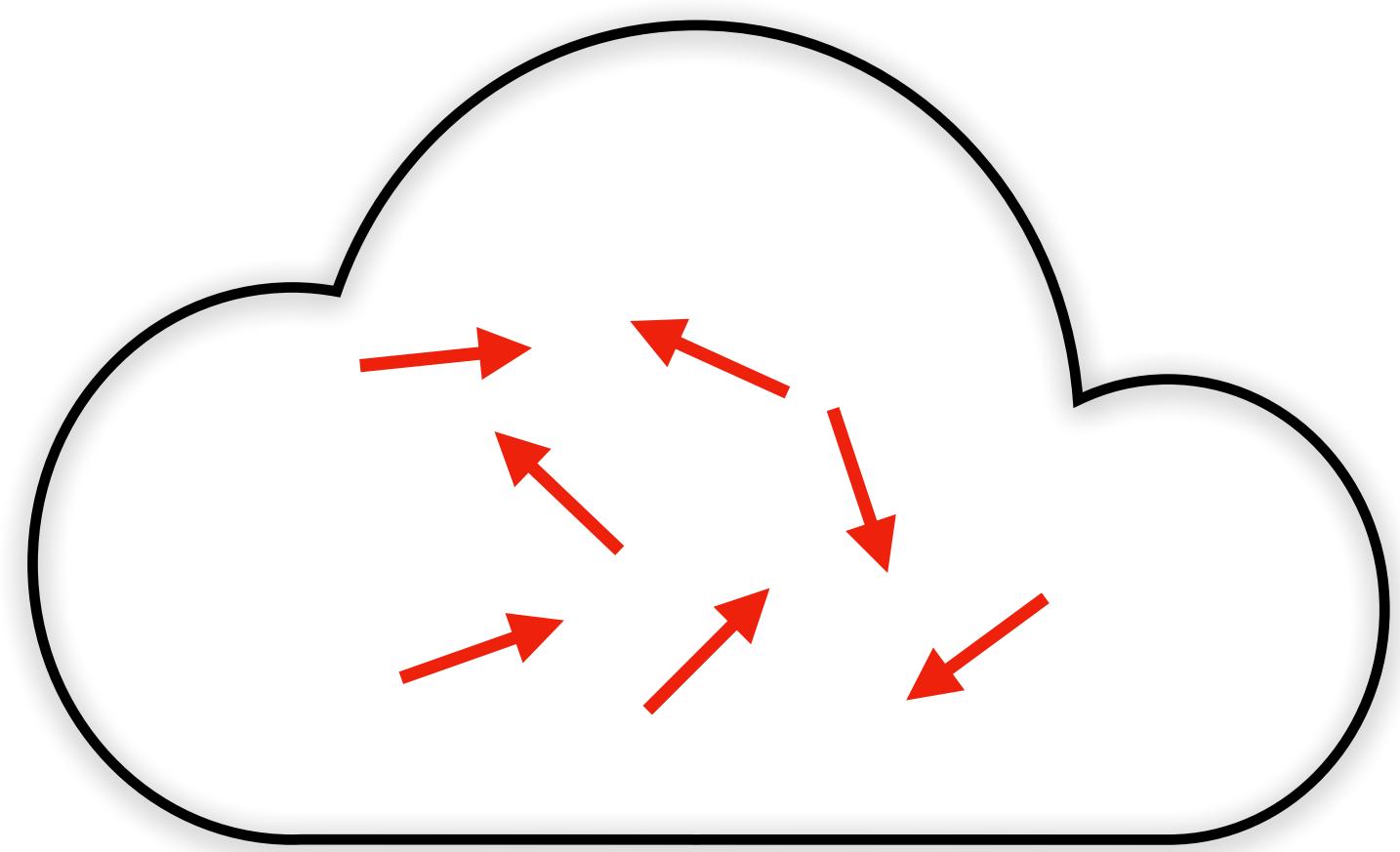


\dot{q}

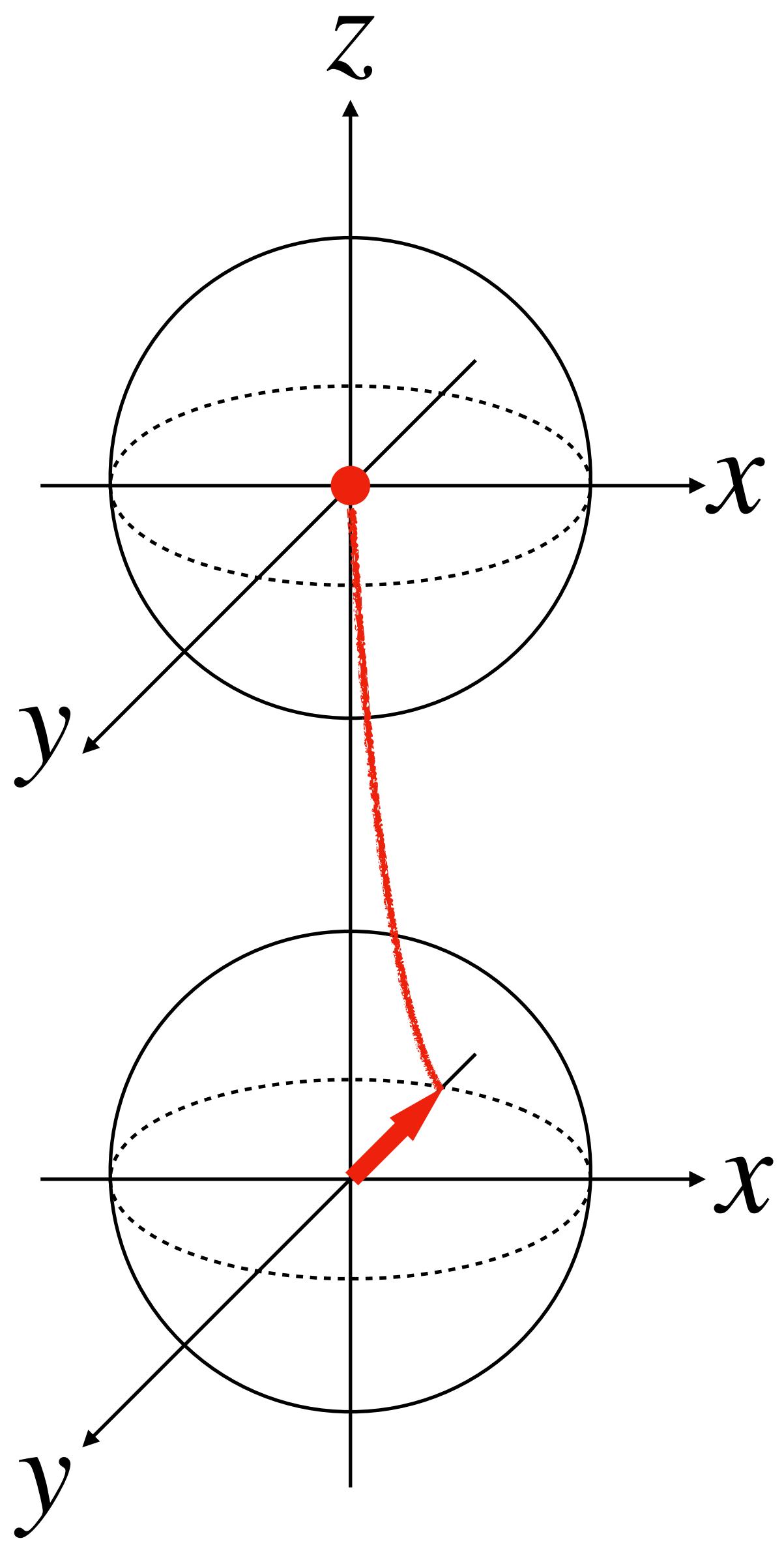
\hbar



$$H_c = \epsilon \sigma_z$$



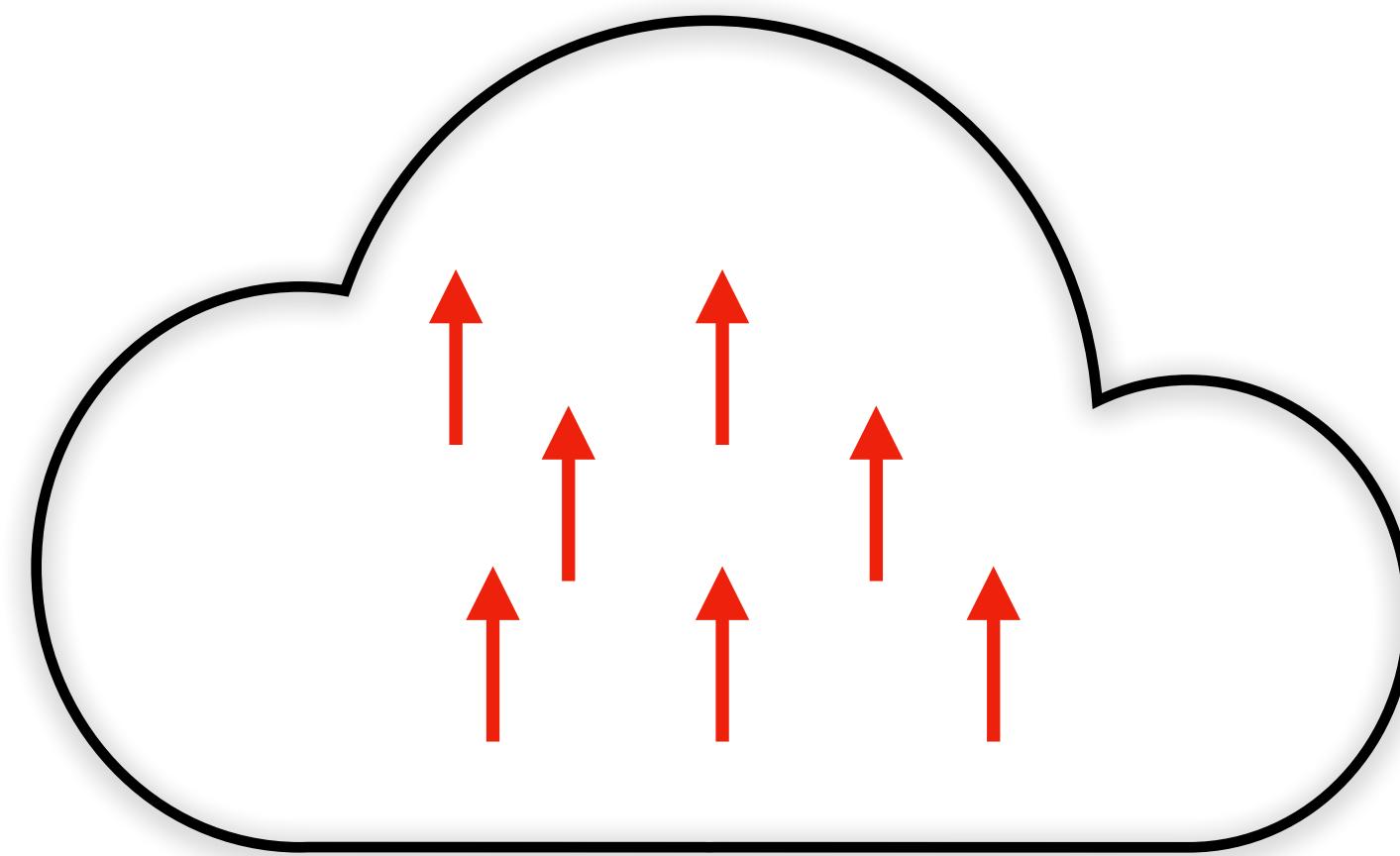
Spin echo



\dot{q}



Spin echo simulation



Initial density matrices

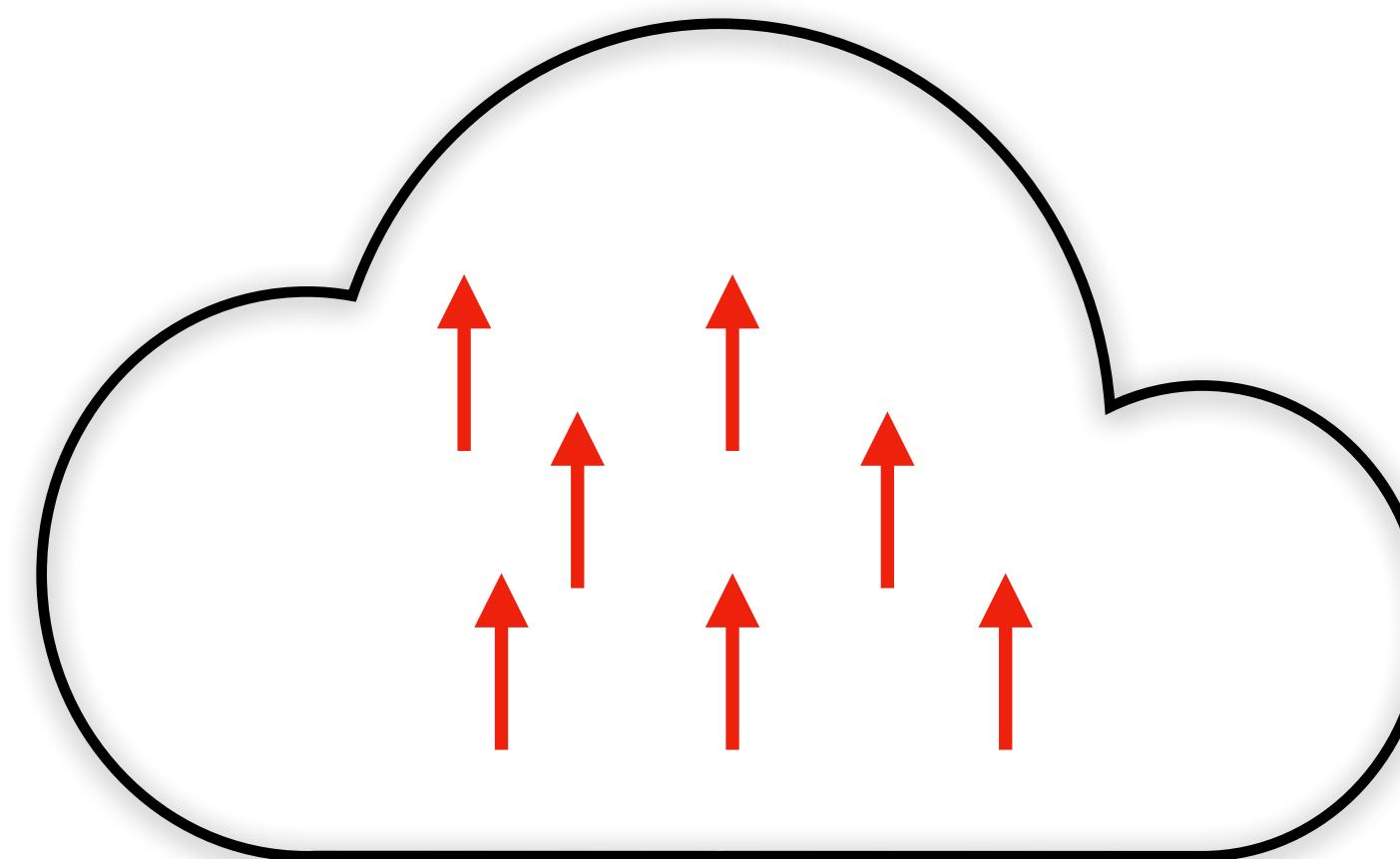
```
1 rho = tf.constant([[1, 0], [0, 0]], dtype=tf.complex128)
2 rho = rho[tf.newaxis]
3 rho = tf.tile(rho, (10000, 1, 1))
```



Last function makes 10000 copies of density matrix and stack them together. The resulting tensor has shape (10000, 2, 2)



Spin echo simulation



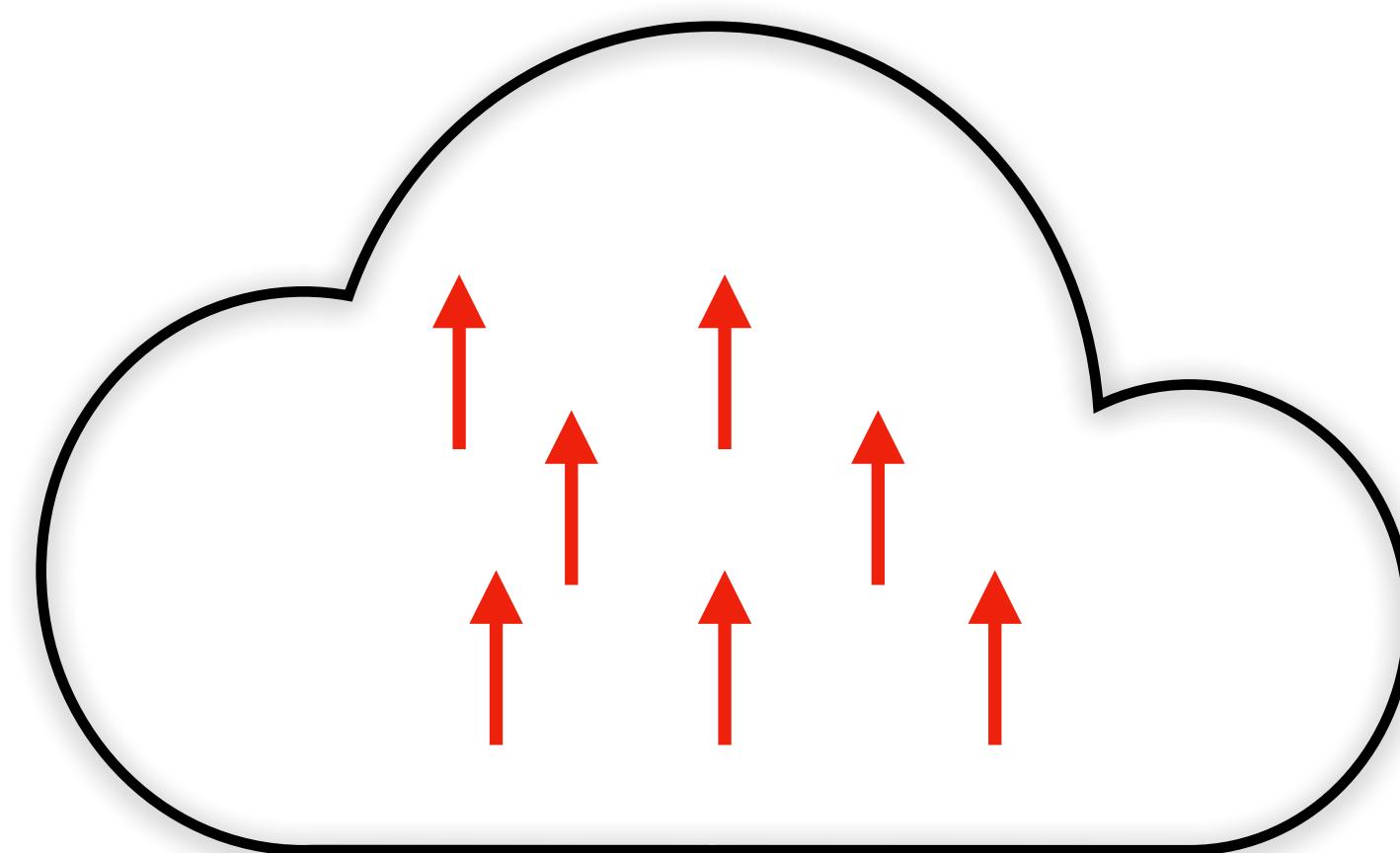
```
1 H = tf.constant([[0.5, 0], [0, -0.5]], dtype=tf.complex128)
2 delta = tf.random.normal((N, 1, 1))
3 delta = tf.cast(delta, dtype=tf.complex128)
4 H = H * delta
```

Disordered Hamiltonians

Resulting tensor has shape (10000, 2, 2)



Spin echo simulation



```
1 H = tf.constant([[0.5, 0], [0, -0.5]], dtype=tf.complex128)
2 delta = tf.random.normal((N, 1, 1))
3 delta = tf.cast(delta, dtype=tf.complex128)
4 H = H * delta
```

Disordered Hamiltonians

Resulting tensor has shape (10000, 2, 2)

Corresponding unitary transformations

Time step

```
1 U = tf.linalg.expm(-1j * H * 0.1)
```

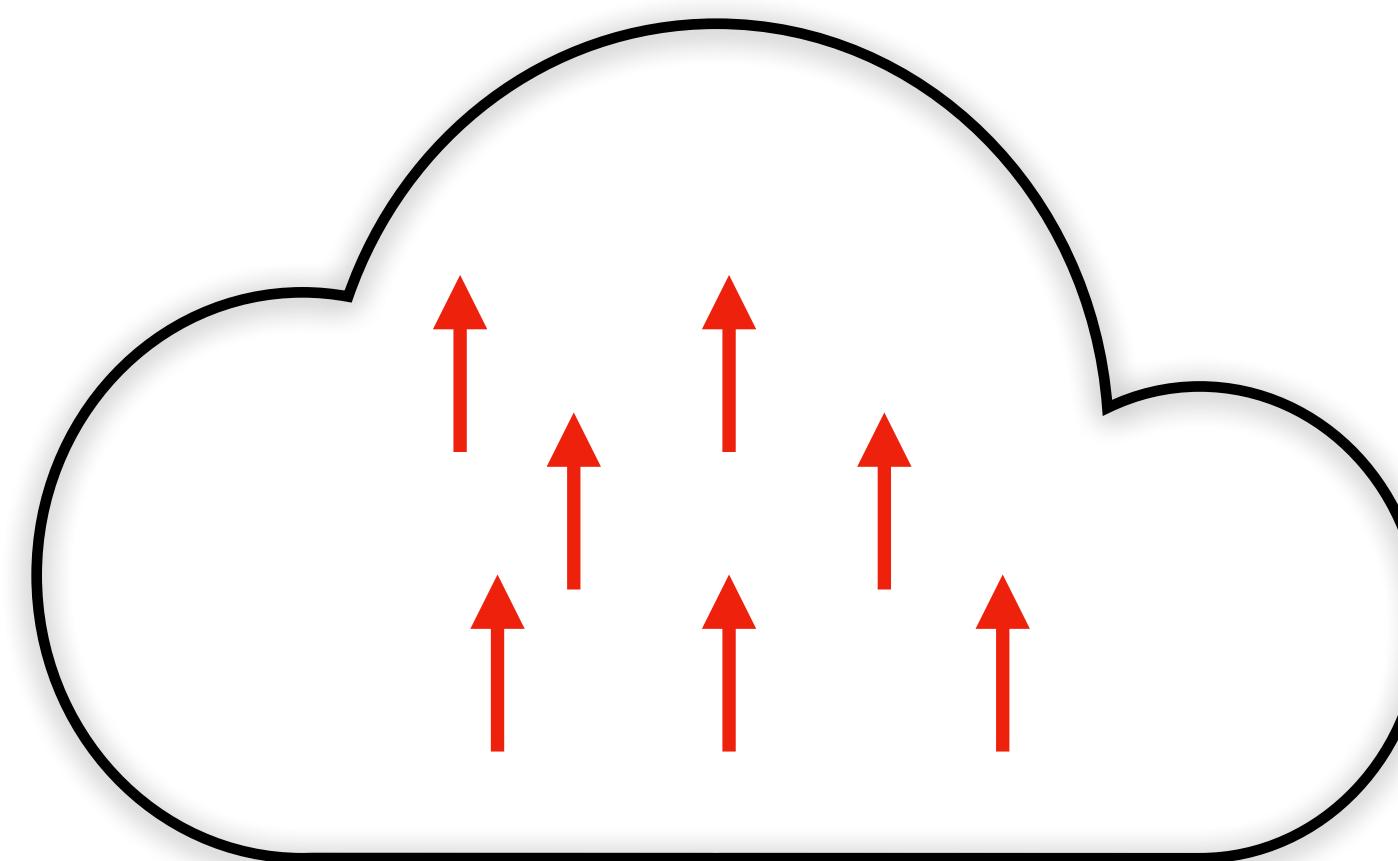
Resulting tensor has shape (10000, 2, 2)



Spin echo simulation



Disordered Hamiltonians with control

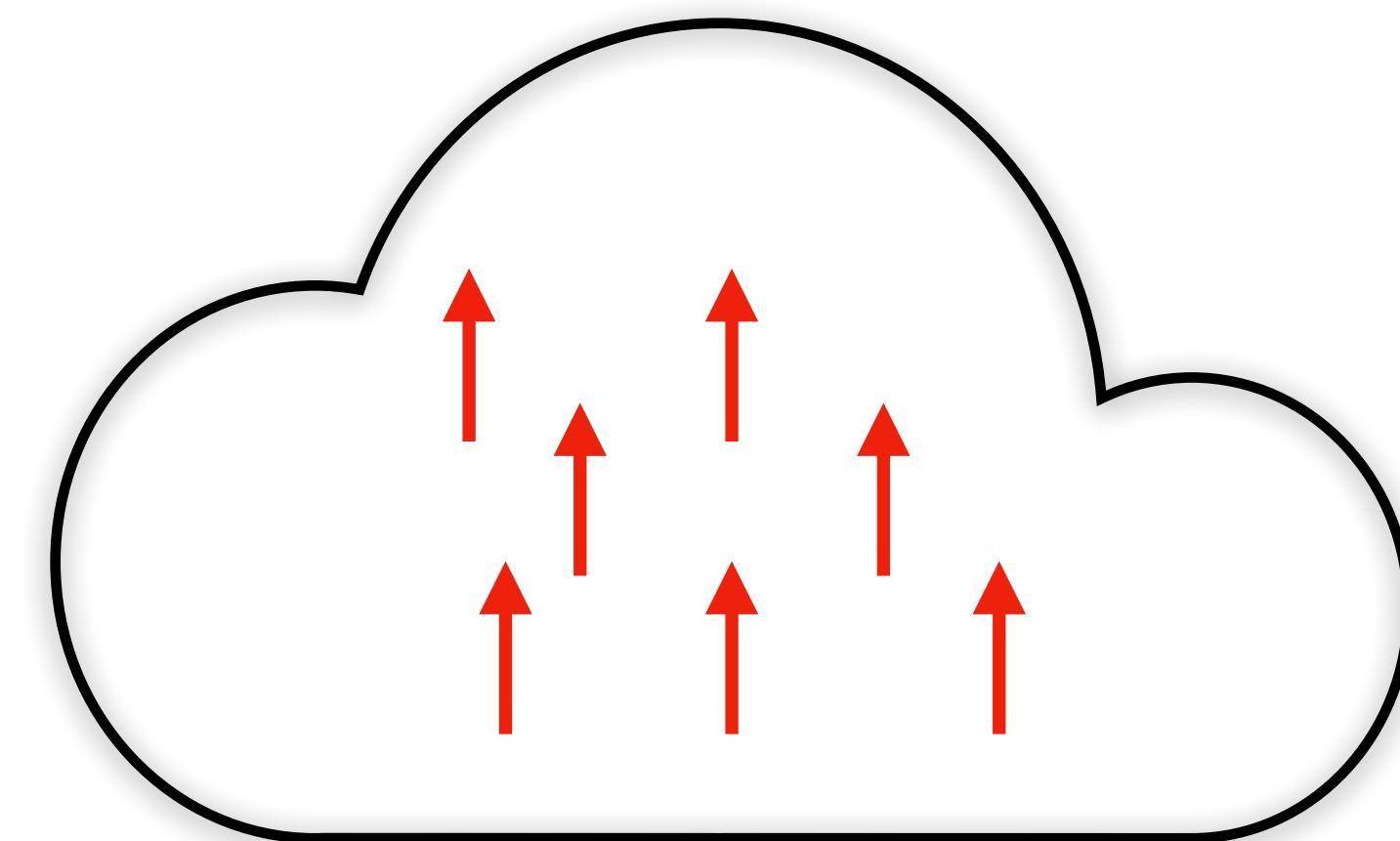


```
1 H_c = H - 0.5 * math.pi * sigma_x
```

Resulting tensor has shape (10000, 2, 2)



Spin echo simulation



Disordered Hamiltonians with control

```
1 H_c = H - 0.5 * math.pi * sigma_x
```

Resulting tensor has shape (10000, 2, 2)

Corresponding unitary transformations

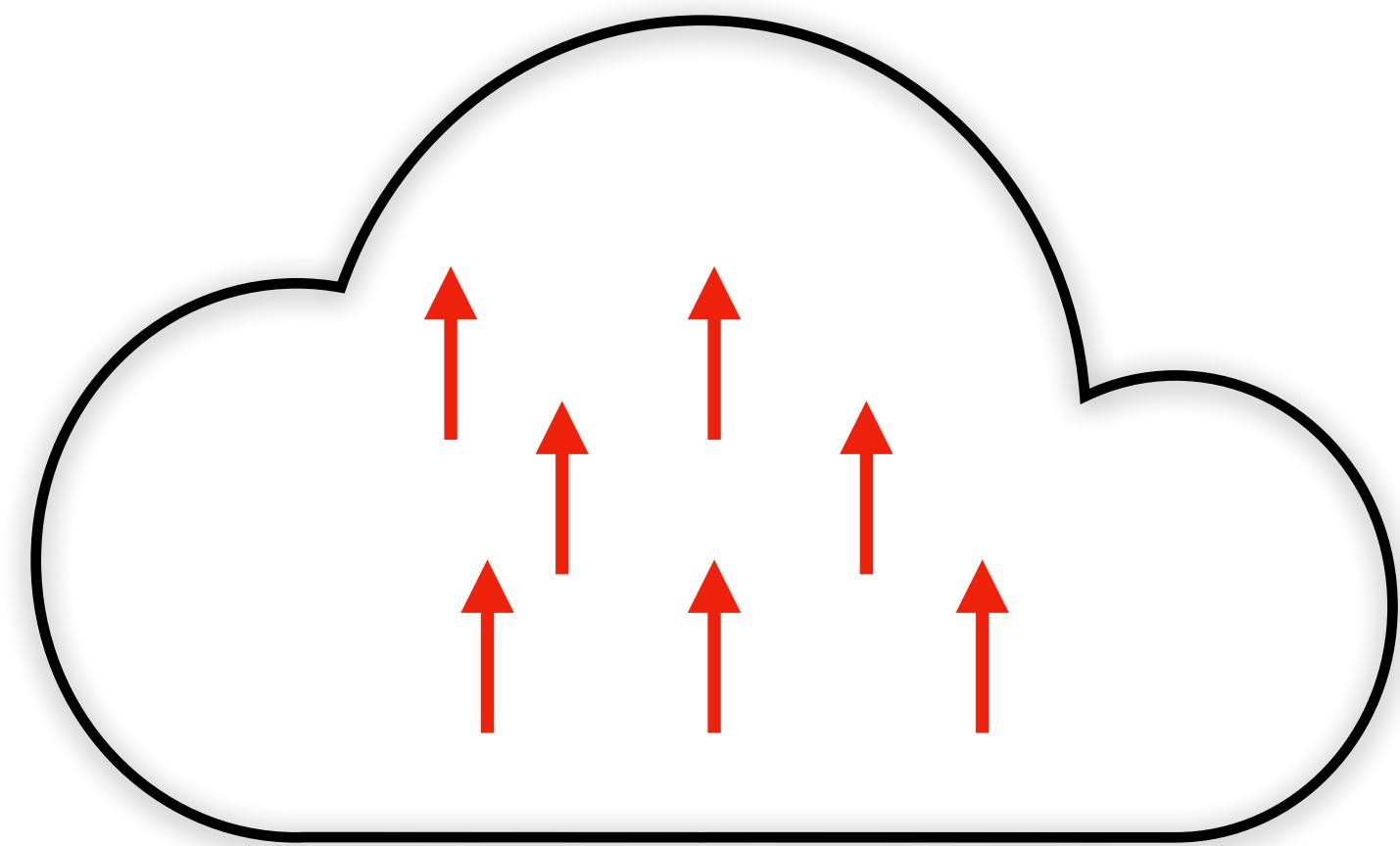
Time step

```
1 U_c = tf.linalg.expm(-1j * H_c * 0.1)
```

Resulting tensor has shape (10000, 2, 2)



Spin echo simulation



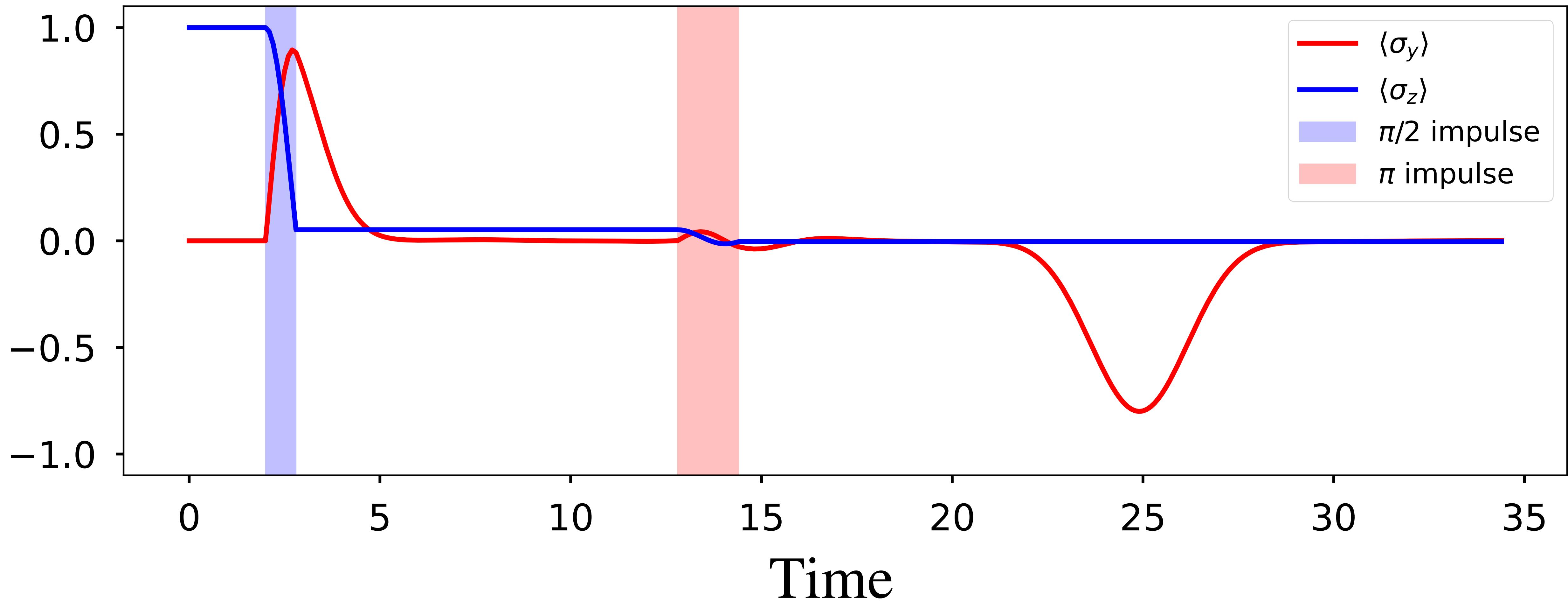
Dynamics of **whole ensemble**

```
1 rho = U_c @ rho @ tf.linalg.adjoint(U_c)
```

$\check{\hbar}$

Spin echo simulation

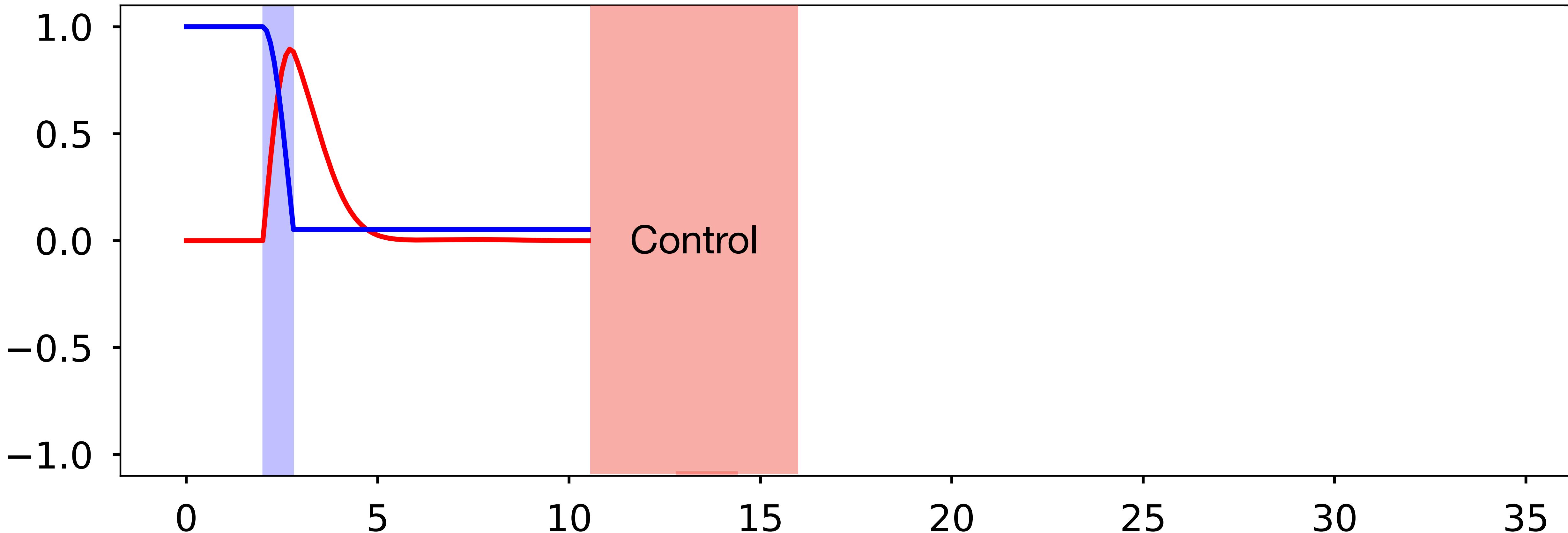
\check{g}





Spin echo optimization

$$H_c(t) = \epsilon\sigma_z + A(t)\sigma_x$$





Spin echo optimization



```
1 control = tf.Variable(tf.random.normal((30,), dtype=tf.float64))
```

The value of `tf.Varianle` can be
changed using one of the
assign methods

Number of time steps



Spin echo optimization



```
1 control = tf.Variable(tf.random.normal((30,), dtype=tf.float64))
```

The value of `tf.Varianle` can be changed using one of the assign methods

Number of time steps

```
1 gated_control = tf.nn.tanh(control)
```

Restrict control on the interval
 $(-1, 1)$



Spin echo optimization



```
1 complex_gated_control = tf.cast(gated_control, dtype=tf.complex128)
```



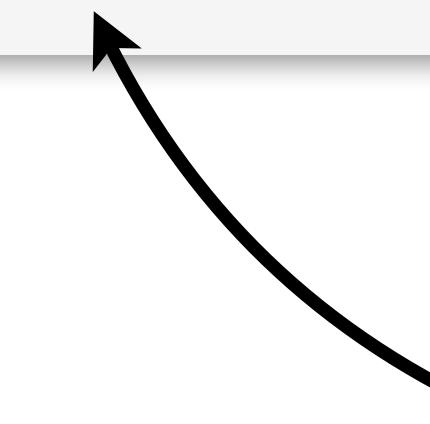
Spin echo optimization



```
1 complex_gated_control = tf.cast(gated_control, dtype=tf.complex128)
```

```
1 H_c = tf.tensordot(complex_gated_control, sigma_x, axes=0)
```

Control Hamiltonian of shape
(30, 2, 2)

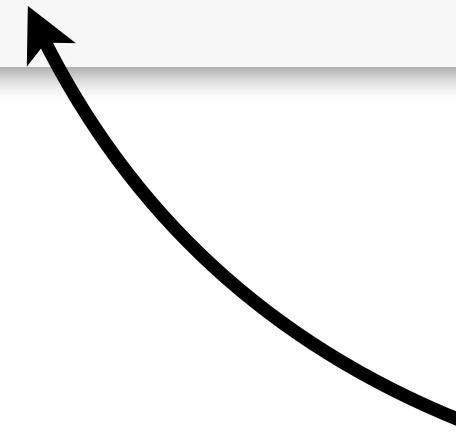




Spin echo optimization



```
1     H_full = H[:, tf.newaxis] + H_c
```



Full Hamiltonian of shape
(10000, 30, 2, 2)



Spin echo optimization



```
1 control
```



```
1 with tf.GradientTape() as tape:
```

Dynamics calculation



```
1 last_y
```



Spin echo optimization



```
1 control
```



```
1 with tf.GradientTape() as tape:
```

Dynamics calculation



```
1 last_y
```

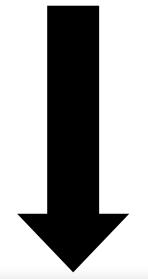
```
1 loss = L1 + last_y
```



Spin echo optimization

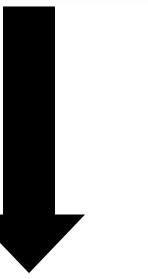


```
1 control
```



```
1 with tf.GradientTape() as tape:
```

Dynamics calculation



```
1 last_y
```

```
1 opt = tf.optimizers.Adam(0.1)
```

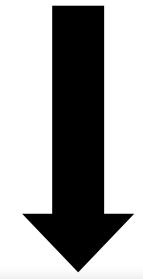
```
1 loss = L1 + last_y
```



Spin echo optimization



```
1 control
```



```
1 with tf.GradientTape() as tape:
```

Dynamics calculation



```
1 last_y
```

```
1 opt = tf.optimizers.Adam(0.1)
```

```
1 grad = tape.gradient(loss, control)
```

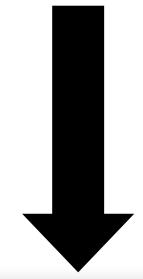
```
1 loss = L1 + last_y
```



Spin echo optimization



```
1 control
```



```
1 with tf.GradientTape() as tape:
```

Dynamics calculation



```
1 last_y
```

```
1 loss = L1 + last_y
```

```
1 opt = tf.optimizers.Adam(0.1)
```

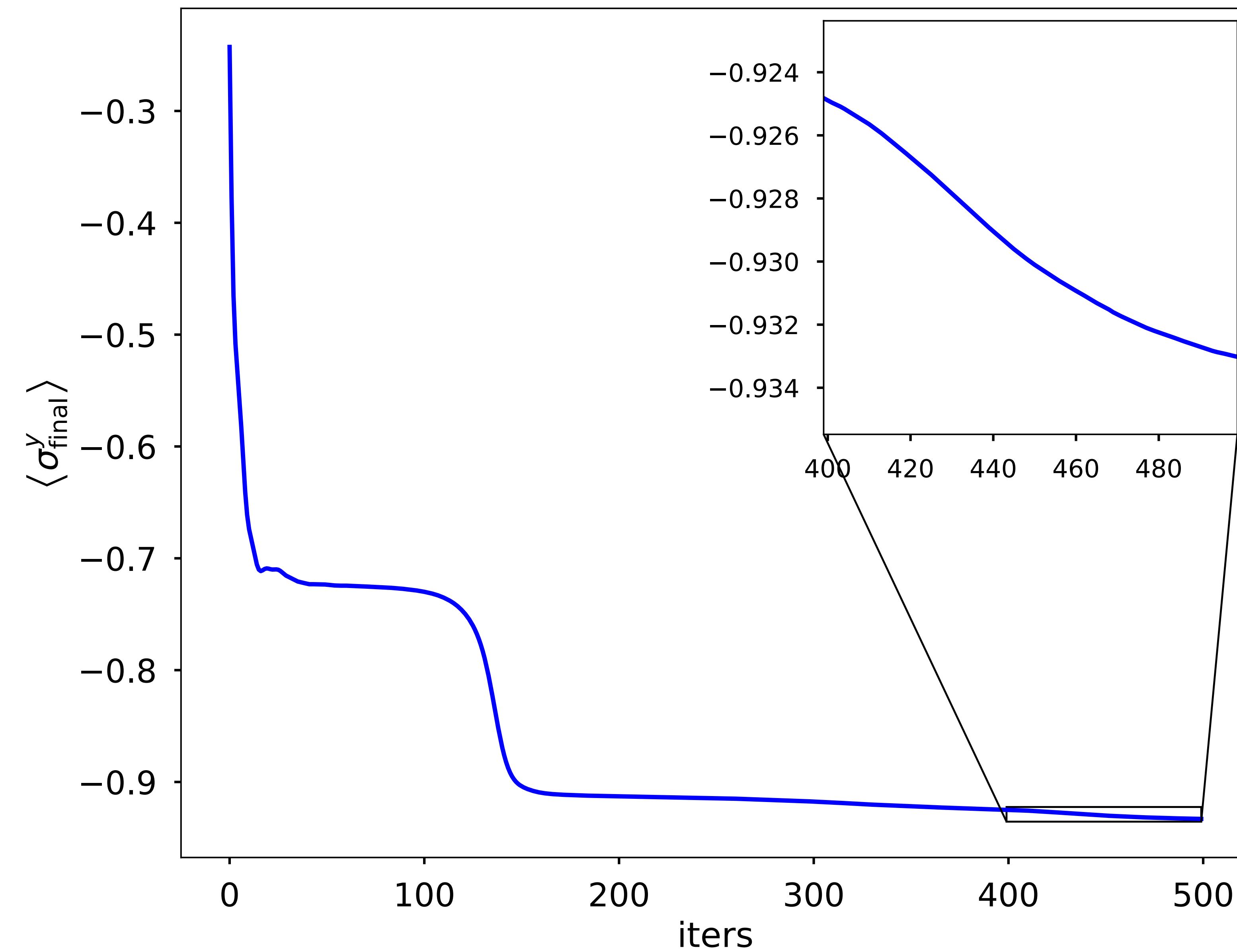
```
1 grad = tape.gradient(loss, control)
```

```
1 opt.apply_gradients(zip([grad], [control]))
```

ħ

Spin echo optimization

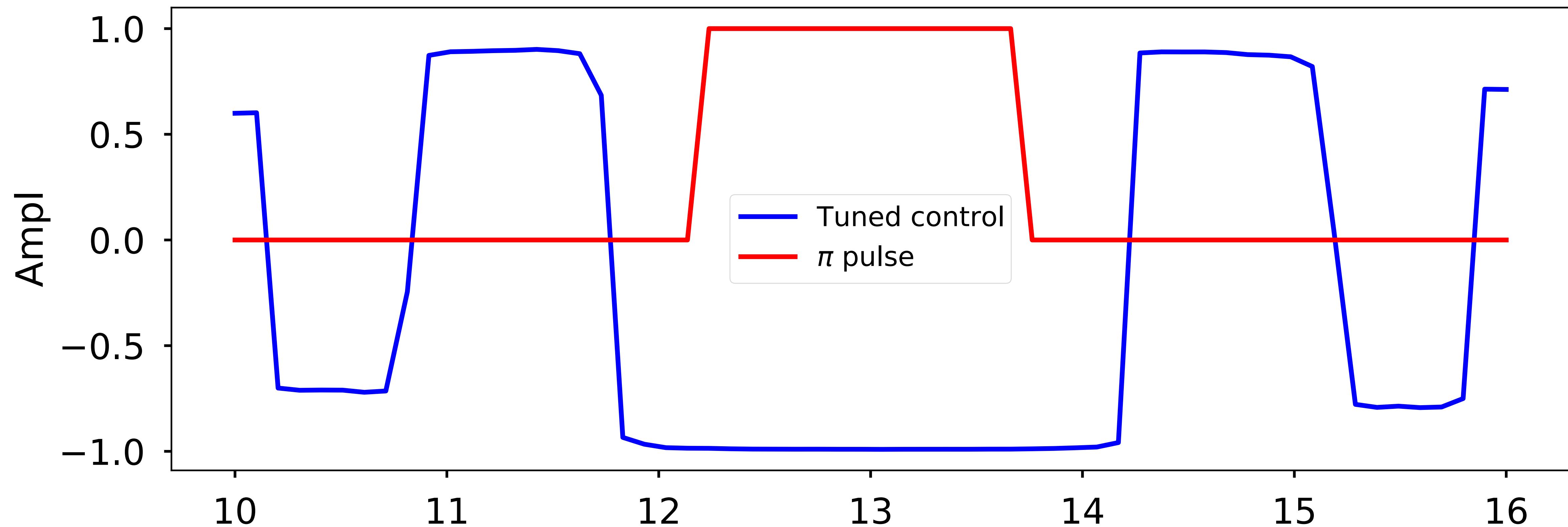
g



\hbar

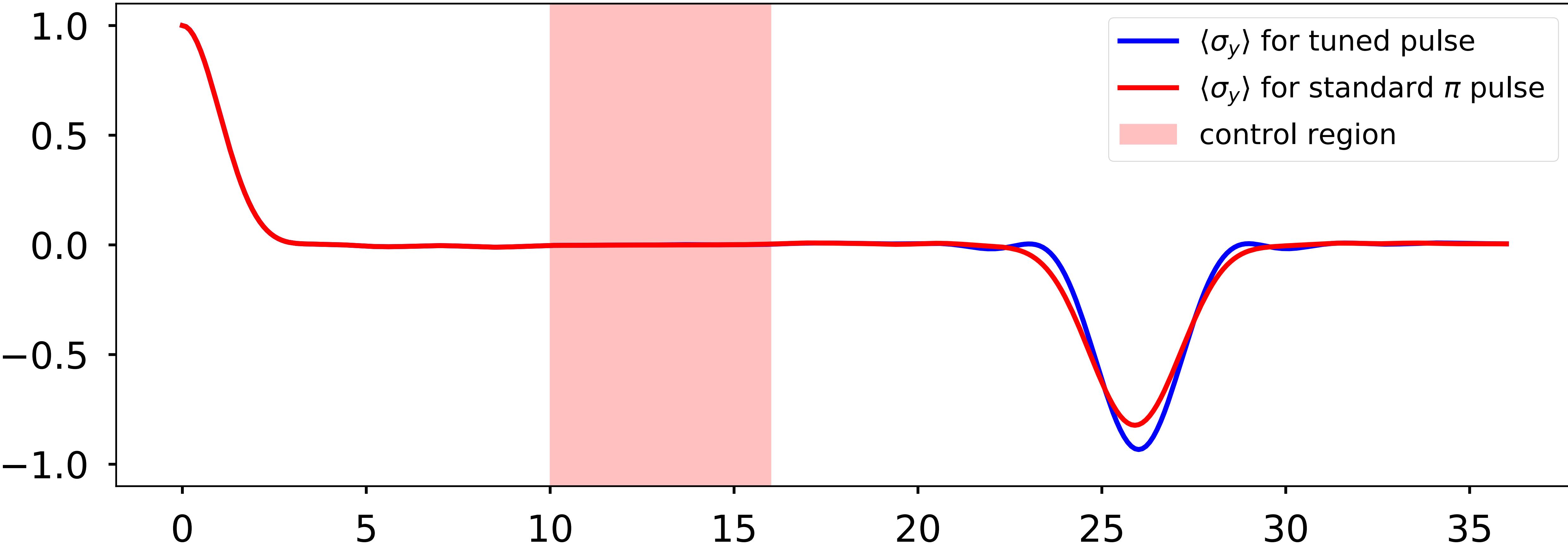
Spin echo optimization

\dot{g}



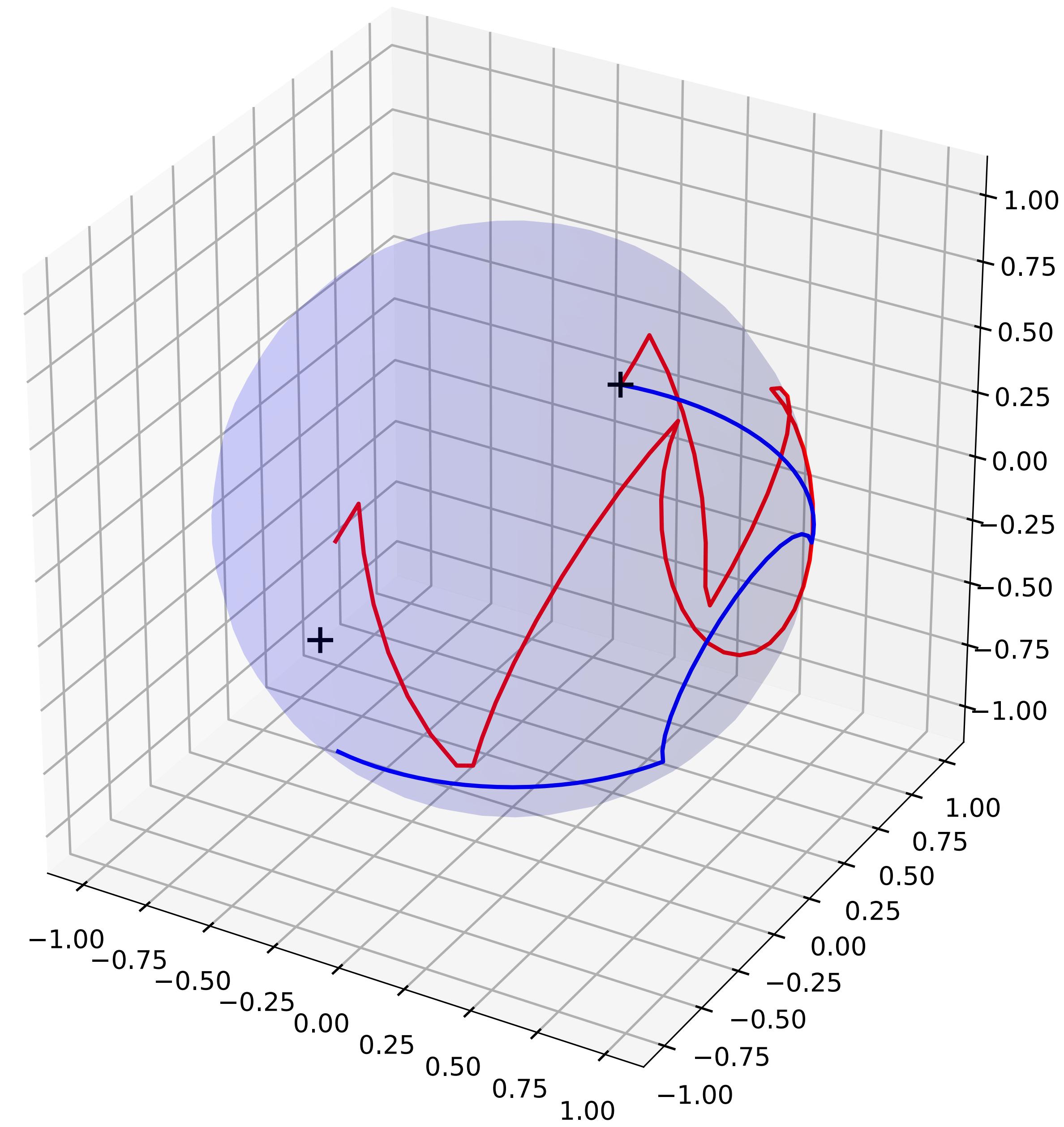


Spin echo optimization





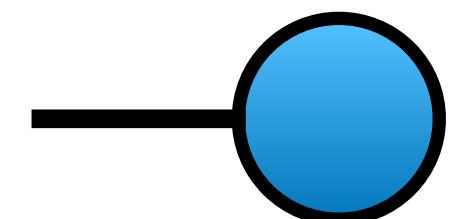
Spin echo optimization



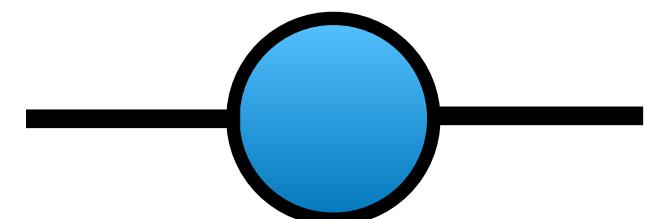
\hbar

Penrose graphical notation

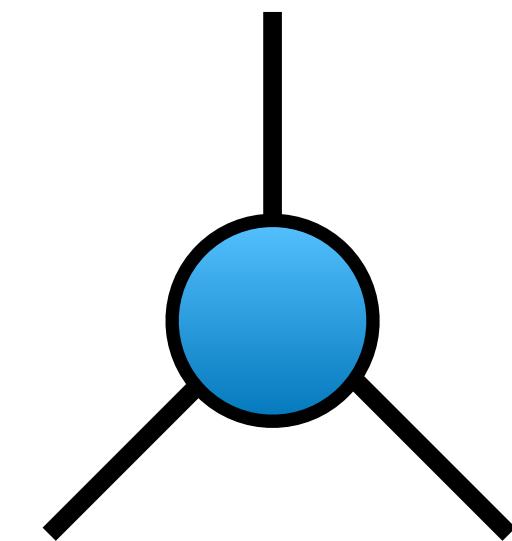
g



Vector



Matrix



Tensor

$$\sum_j A_{ij} a_j = \text{---}_i \text{---}_j \text{---}_a$$

$$\sum_j A_{ij} B_{jk} = \text{---}_i \text{---}_j \text{---}_B \text{---}_k$$

$$\text{Tr}(A) = \text{---}_A \text{---}$$

$$\sum_{jk} C_{ijk} D_{ljk} = \text{---}_i \text{---}_j \text{---}_k \text{---}_l$$

ħ

Matrix product state

g

$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} \psi_{i_1, i_2, \dots, i_n} |i_1\rangle \otimes |i_2\rangle \otimes \dots \otimes |i_n\rangle$$

$\check{\hbar}$

Matrix product state

\check{g}

$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} \psi_{i_1, i_2, \dots, i_n} |i_1\rangle \otimes |i_2\rangle \otimes \dots \otimes |i_n\rangle$$

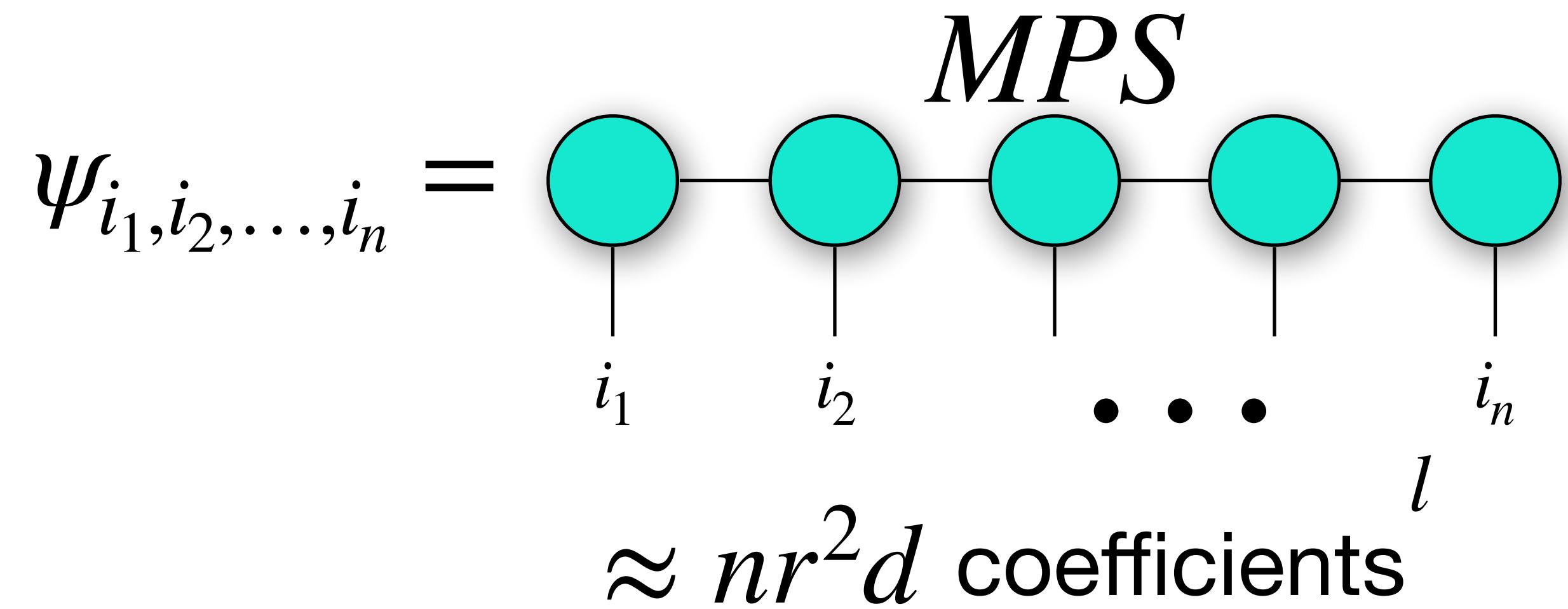
d^n coefficients

$\check{\hbar}$

Matrix product state

\check{g}

$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} \psi_{i_1, i_2, \dots, i_n} |i_1\rangle \otimes |i_2\rangle \otimes \dots \otimes |i_n\rangle \quad d^n \text{ coefficients}$$



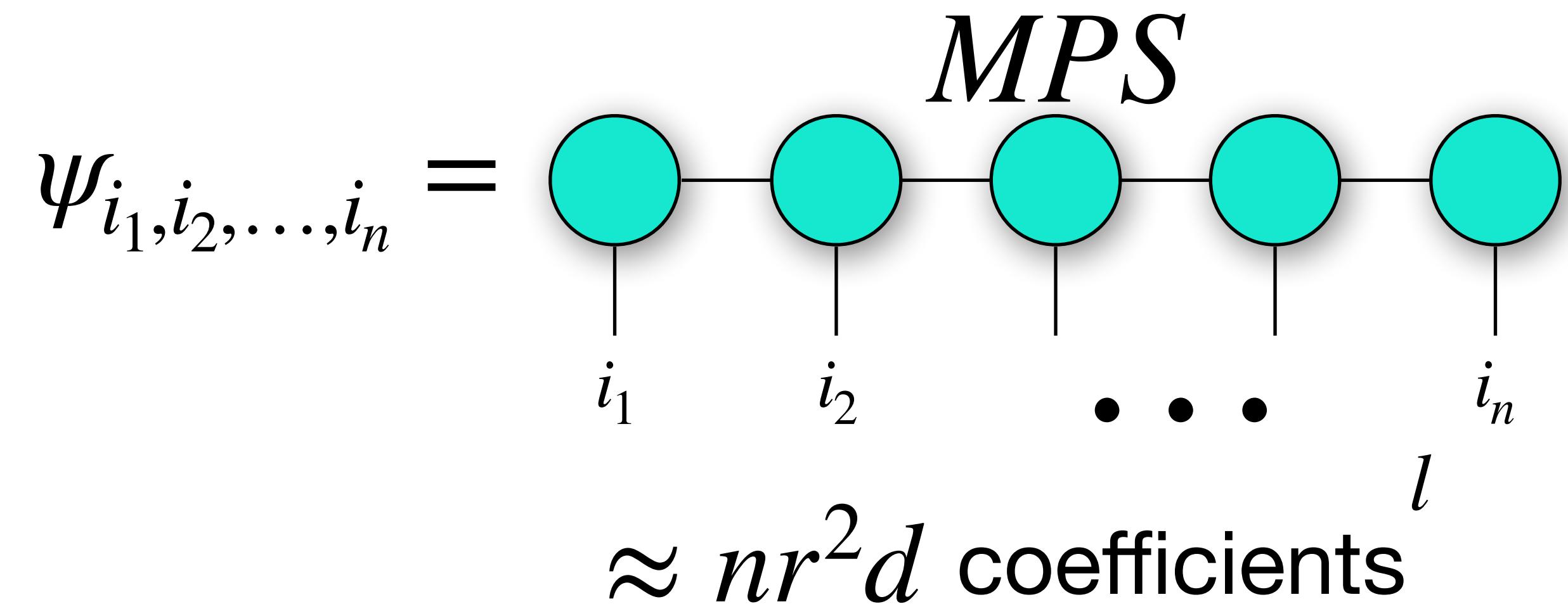


Matrix product state



$$|\psi\rangle = \sum_{i_1, i_2, \dots, i_n} \psi_{i_1, i_2, \dots, i_n} |i_1\rangle \otimes |i_2\rangle \otimes \dots \otimes |i_n\rangle \quad d^n \text{ coefficients}$$

Orús R. A practical introduction to tensor networks:
Matrix product states and projected entangled pair
states. Annals of Physics. 2014 Oct 1;349:117-58.





Ground state of TFI hamiltonian



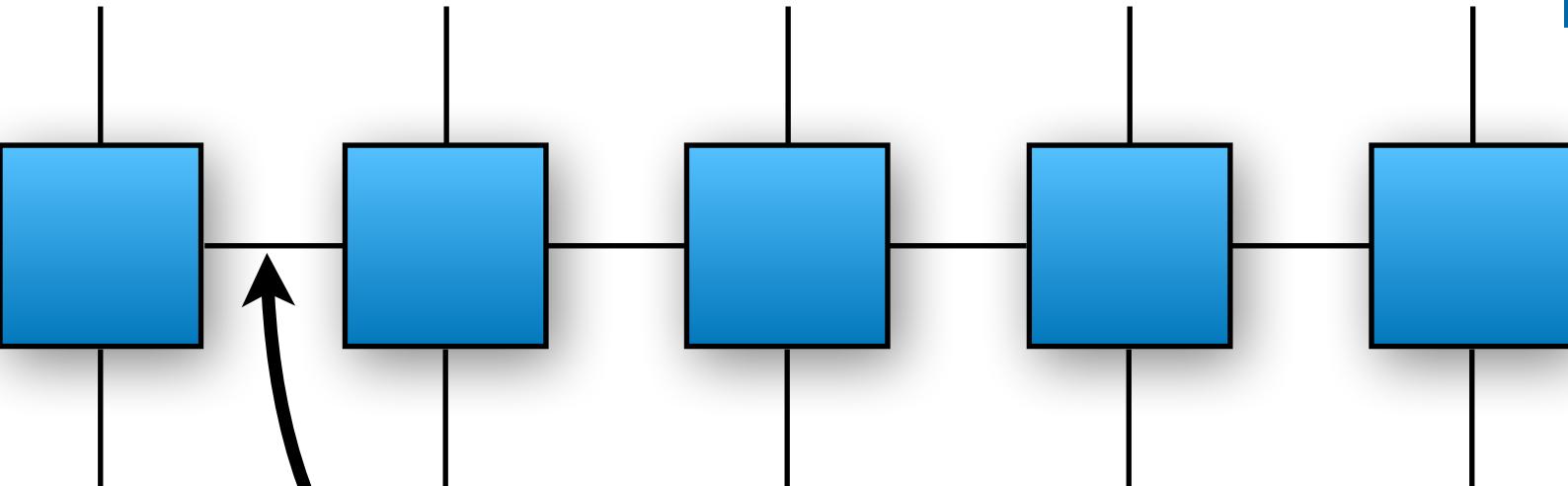
$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i$$



Ground state of TFI hamiltonian



$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i =$$



$$r = 3$$

Schollwöck U. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*. 2011 Jan 1;326(1):96-192.

Ground state of TFI hamiltonian

$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i = \begin{array}{c} \text{Diagram of five blue rectangular blocks connected horizontally, each with a vertical line above it. A curved arrow points from the bottom of the second block to the right, labeled } r = 3. \end{array}$$

Hamiltonian in MPO representation

Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Annals of physics. 2011 Jan 1;326(1):96-192.

Ground state of TFI hamiltonian

$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i = \begin{array}{c} \text{Diagram of five blue squares connected horizontally, each with a vertical line above it. A curved arrow points from the second square to the third square.} \\ r = 3 \end{array}$$

Hamiltonian in MPO representation

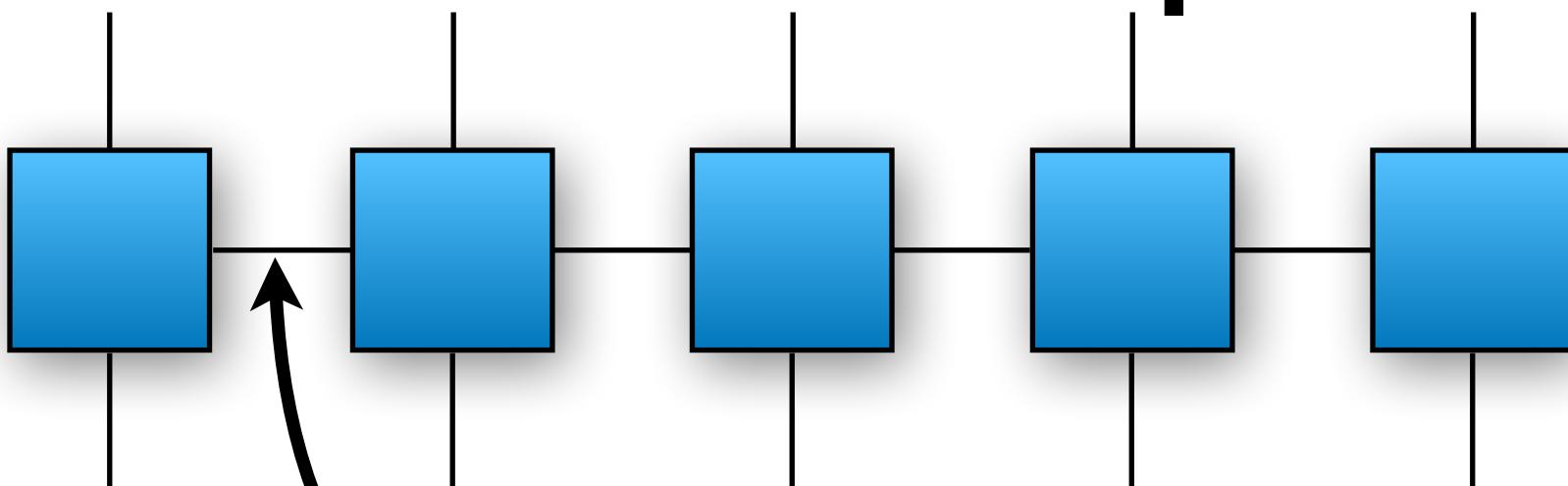
Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Annals of physics. 2011 Jan 1;326(1):96-192.

$$E = \frac{\langle \text{mps} | H | \text{mps} \rangle}{\langle \text{mps} | \text{mps} \rangle}$$

Ground state of TFI hamiltonian

Hamiltonian in MPO representation

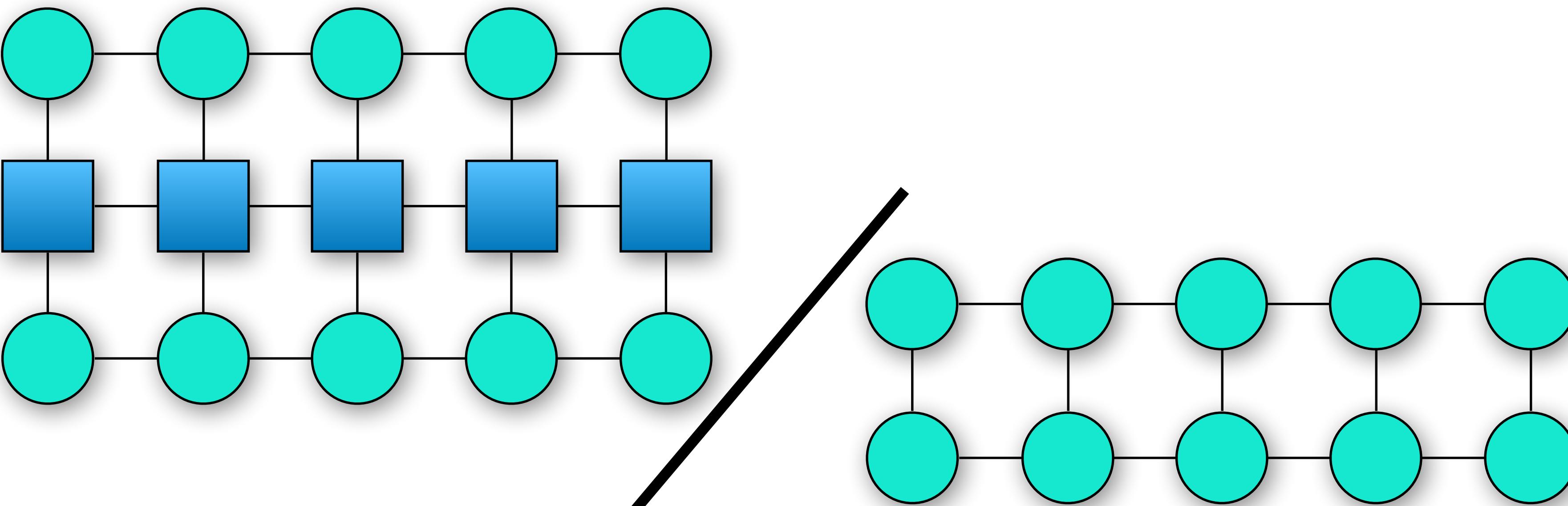
$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i$$



$$r = 3$$

Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Annals of physics. 2011 Jan 1;326(1):96-192.

$$E = \frac{\langle \text{mps} | H | \text{mps} \rangle}{\langle \text{mps} | \text{mps} \rangle}$$



Ground state of TFI hamiltonian

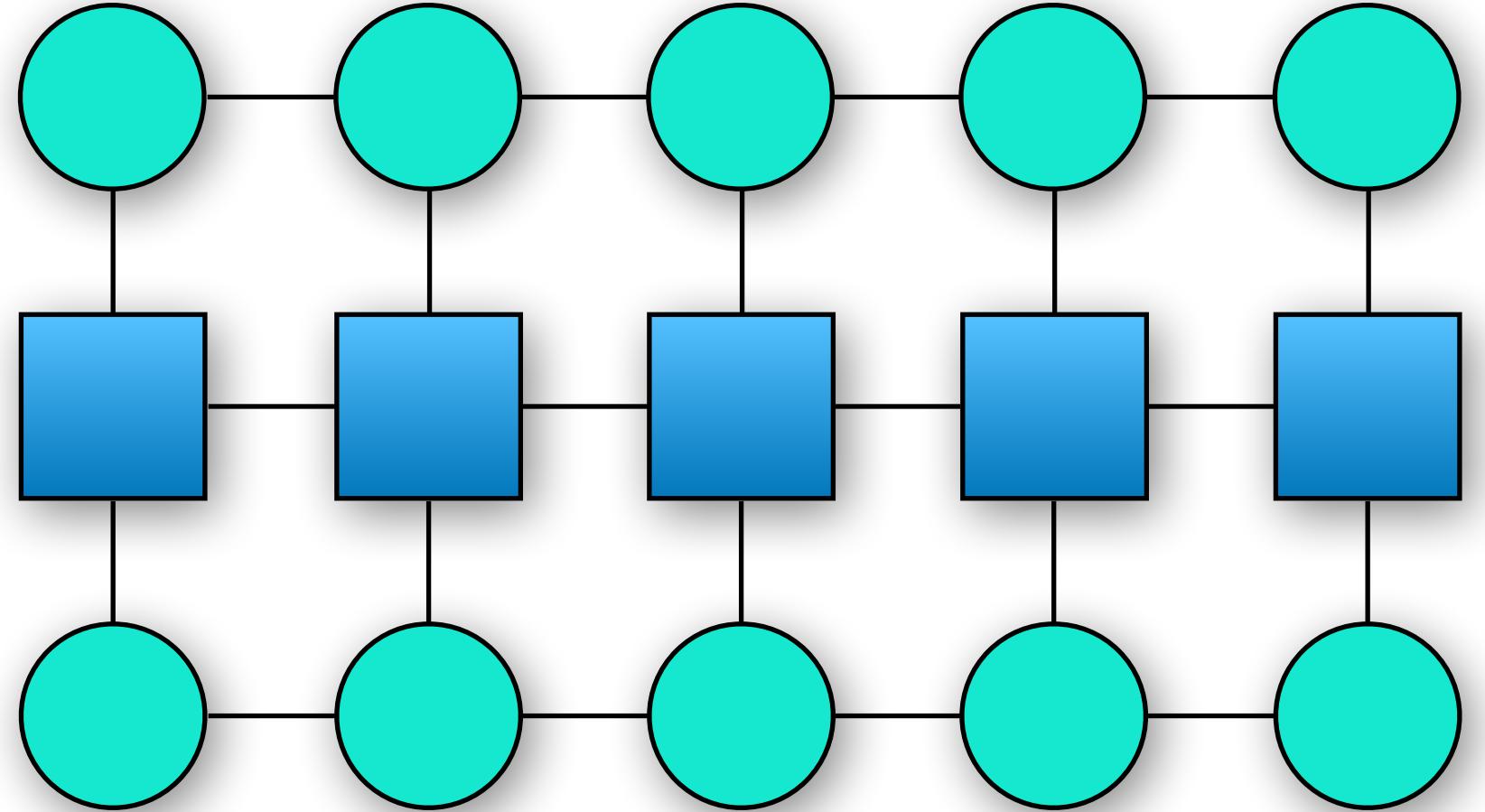
$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i = \begin{array}{c} \text{Diagram of a 1D chain of five sites, each represented by a blue square. Horizontal lines connect adjacent sites. A curved arrow from the right points to the third site, labeled } r = 3. \\ \text{Hamiltonian in MPO representation} \end{array}$$

Schollwöck U. The density-matrix renormalization group in the age of matrix product states. Annals of physics. 2011 Jan 1;326(1):96-192.

$$|\Omega\rangle \approx \operatorname{argmin}_{|\text{mps}\rangle} \frac{\langle \text{mps} | H | \text{mps} \rangle}{\langle \text{mps} | \text{mps} \rangle}$$

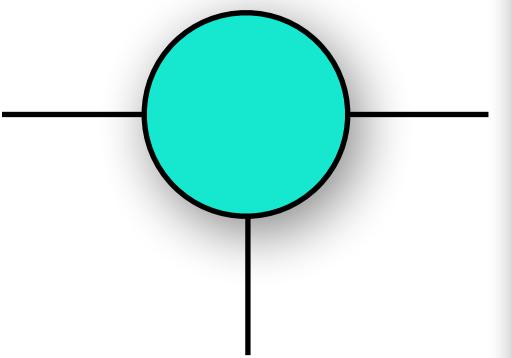
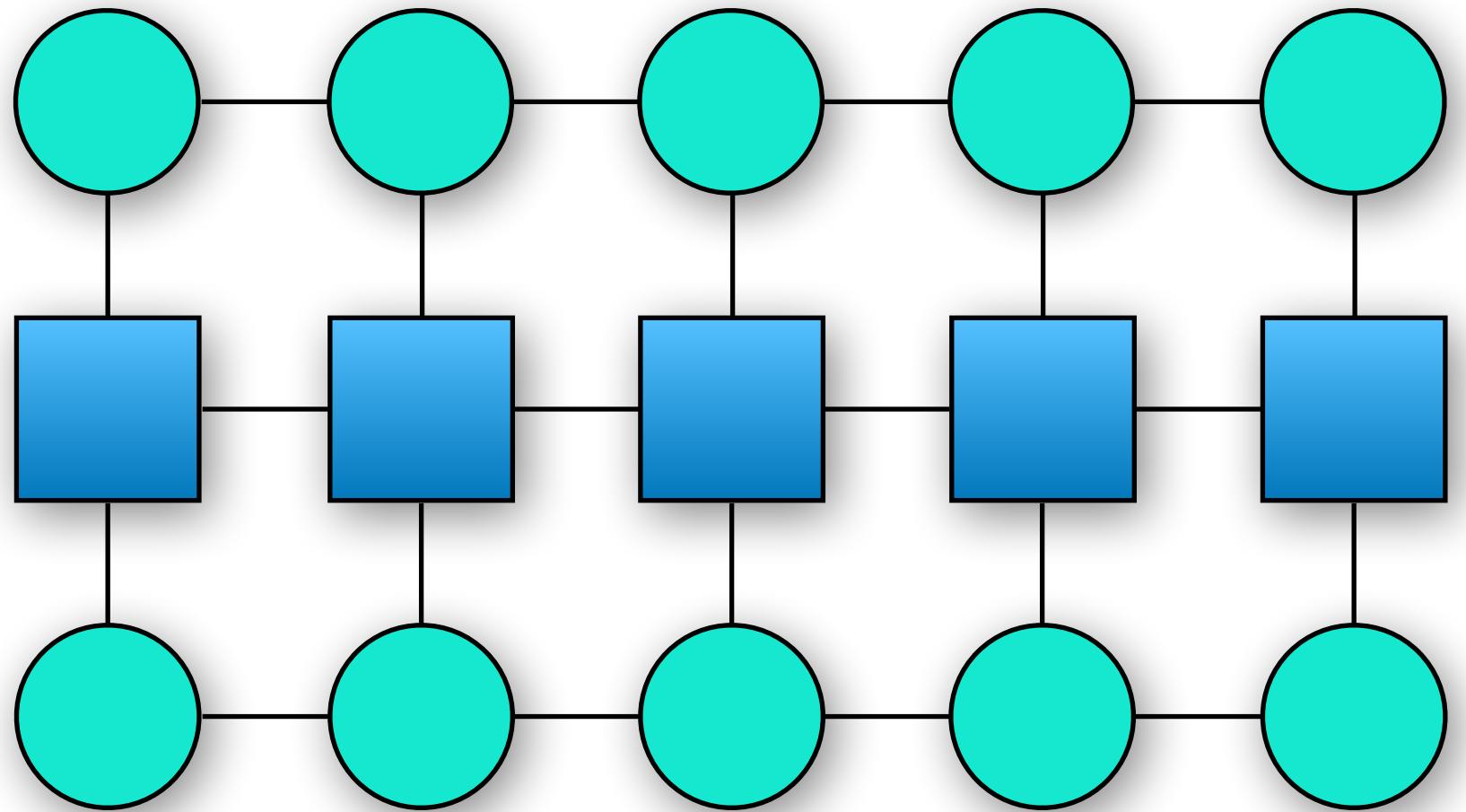


Programming of tensor networks





Programming of tensor networks

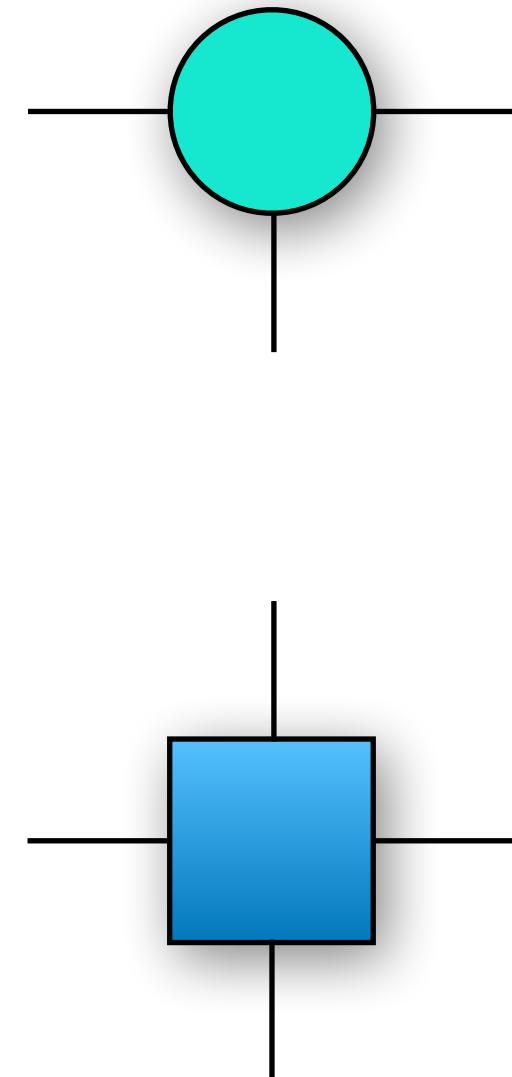
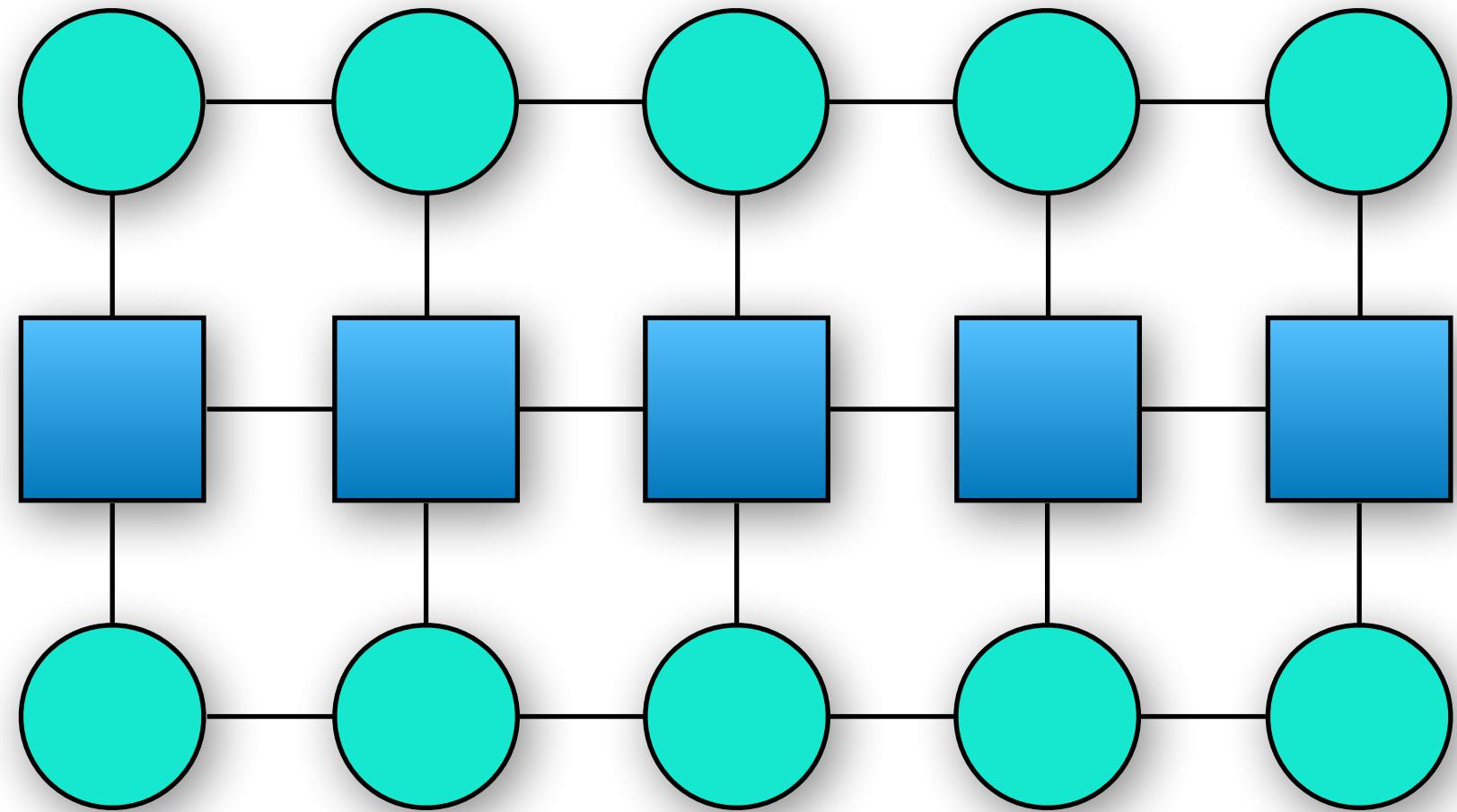


```
1 print(A.shape)
2 print(A.dtype)
```

(28, 2, 28)
<dtype: 'complex128'>



Programming of tensor networks



```
1 print(A.shape)
2 print(A.dtype)
```

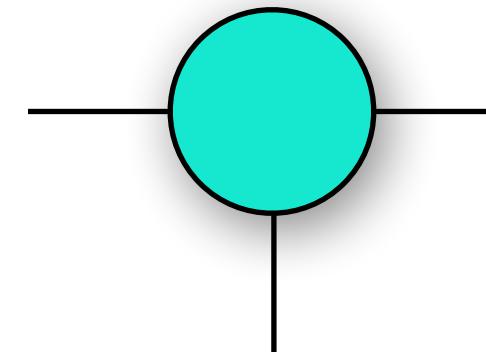
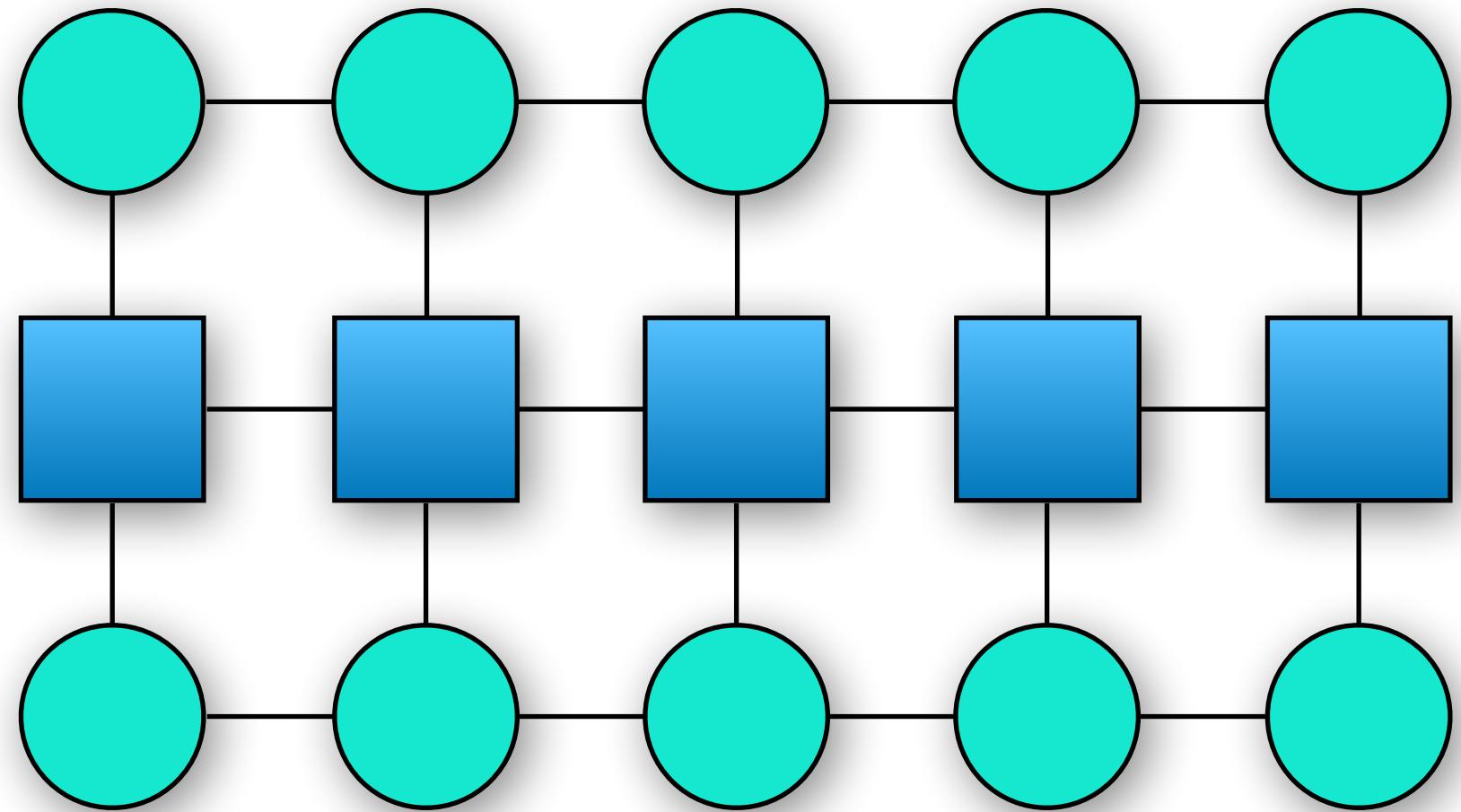
```
(28, 2, 28)
<dtype: 'complex128'>
```

```
1 print(H.shape)
2 print(H.dtype)
```

```
(3, 2, 3, 2)
<dtype: 'complex128'>
```

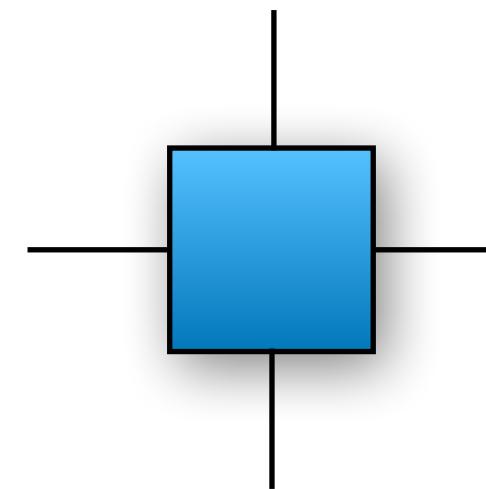


Programming of tensor networks



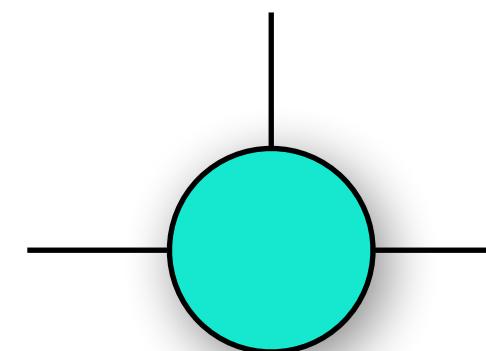
```
1 print(A.shape)
2 print(A.dtype)
```

(28, 2, 28)
<dtype: 'complex128'>



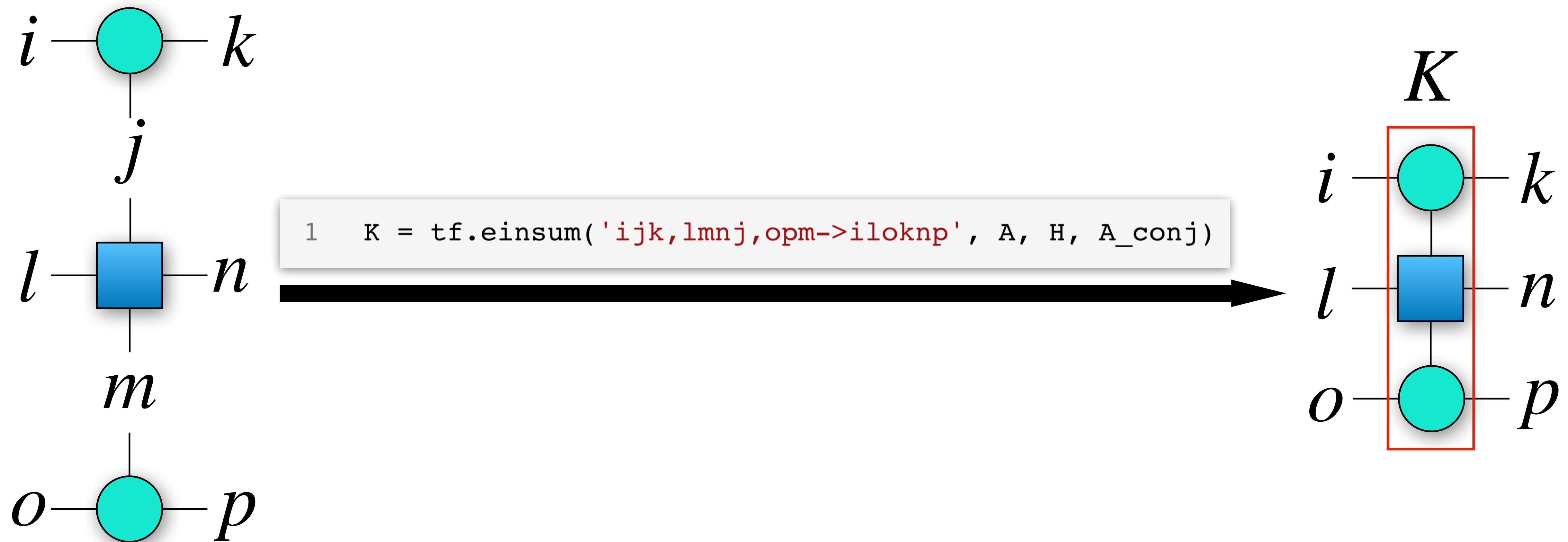
```
1 print(H.shape)
2 print(H.dtype)
```

(3, 2, 3, 2)
<dtype: 'complex128'>



```
1 A_conj = tf.math.conj(A)
```

Programming of tensor networks



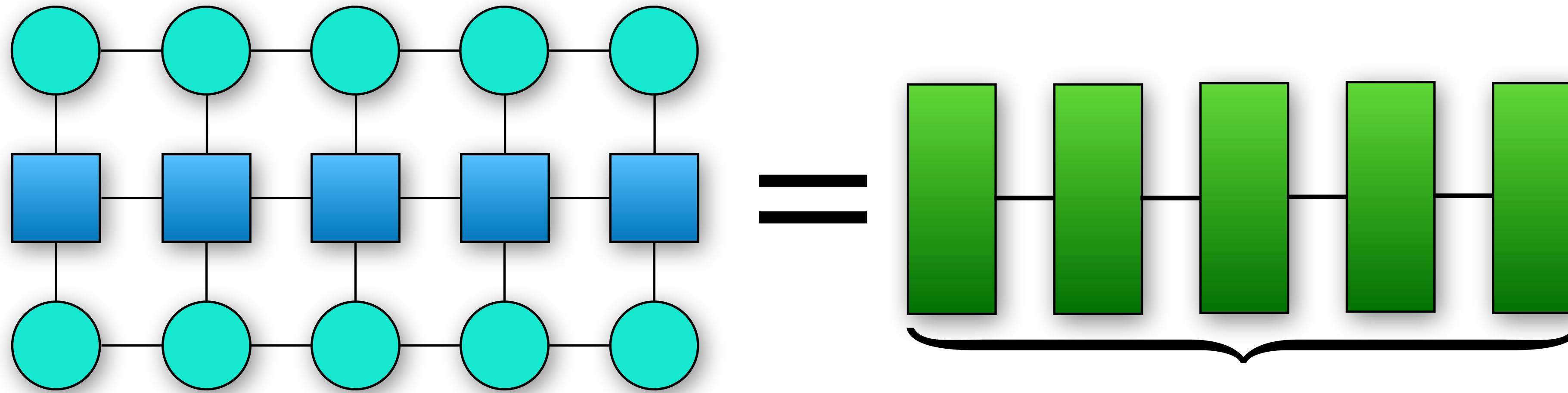


Programming of tensor networks





Programming of tensor networks



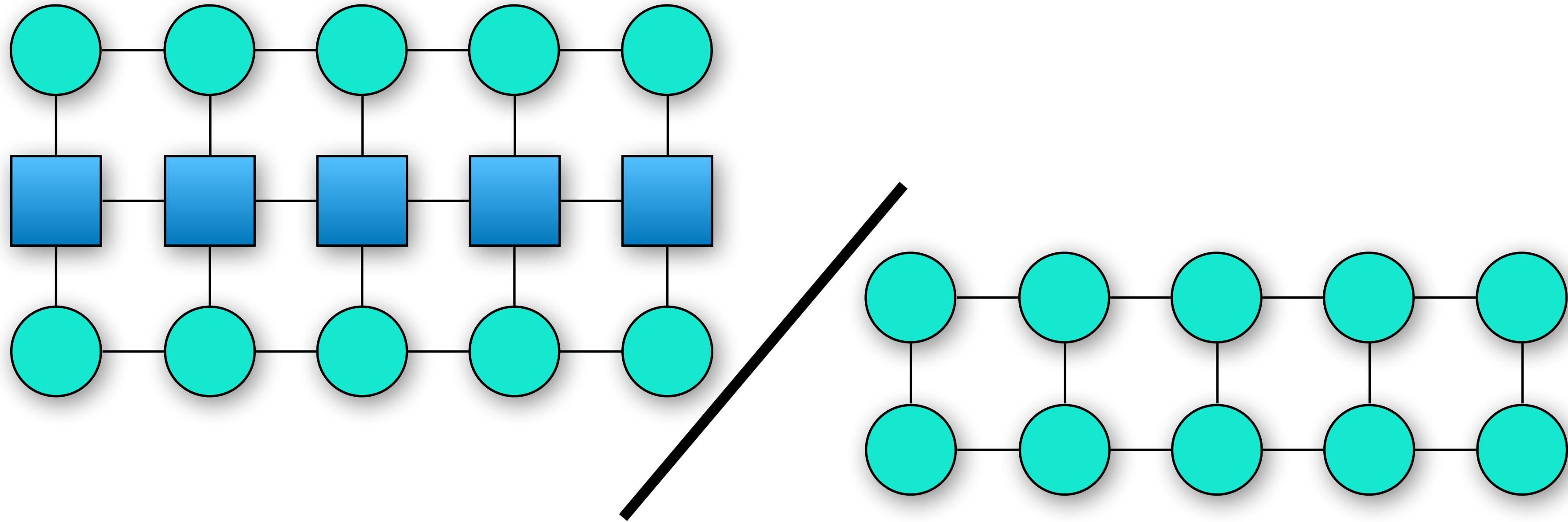
**Many vector by matrix multiplications.
One can calculate by applying
tf.tensordot several times**



Programming of tensor networks



$$E = \frac{\langle \text{mps} | H | \text{mps} \rangle}{\langle \text{mps} | \text{mps} \rangle}$$





Ground state of TFI hamiltonian



$$\left\{ \operatorname{Re} (A_i) \right\}_{i=1}^N, \left\{ \operatorname{Im} (A_i) \right\}_{i=1}^N$$



```
1 with tf.GradientTape() as tape:
```

Energy calculation



E

$\check{\hbar}$

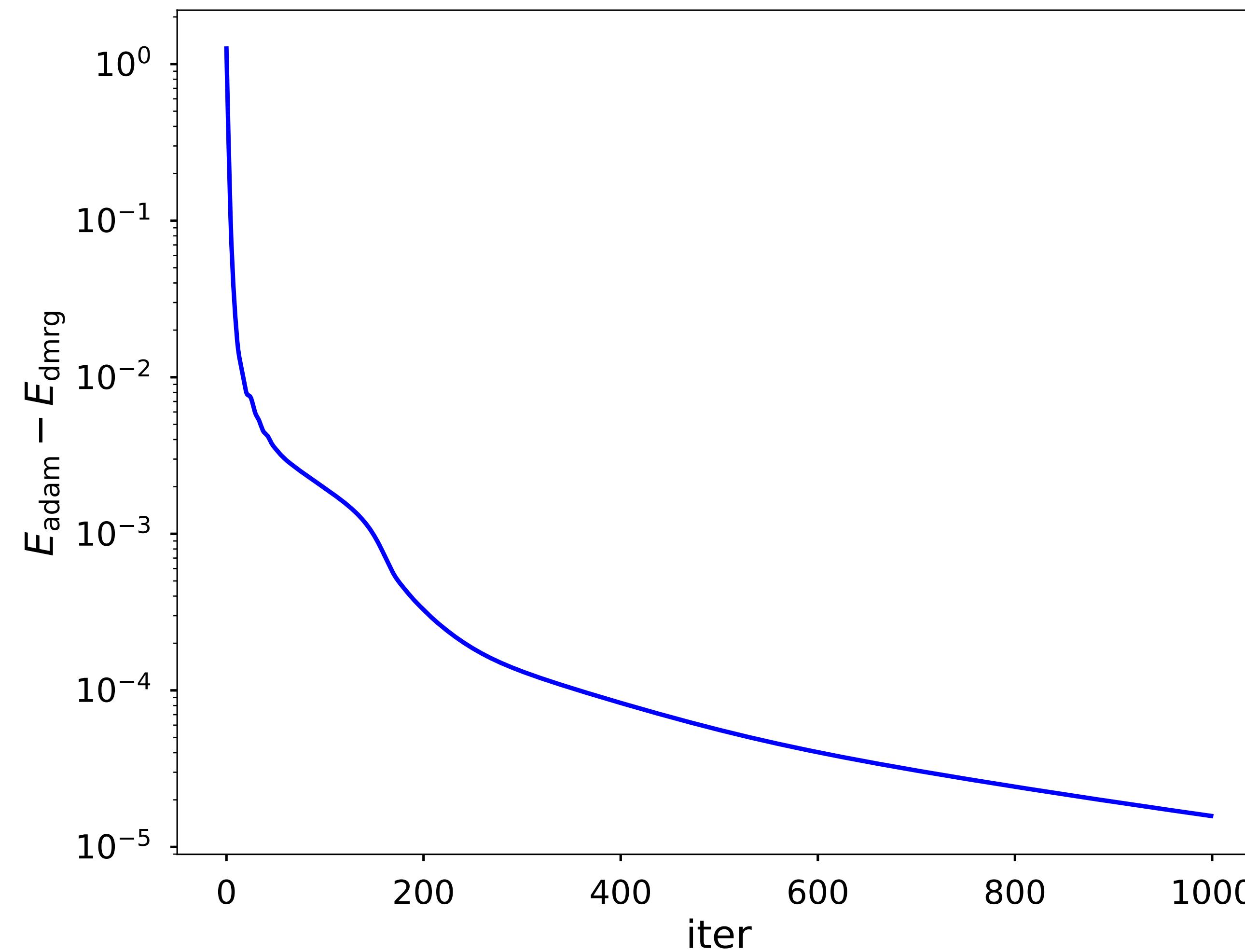
Ground state of TFI hamiltonian

$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i$$

$$N = 32$$

$$h_x = 1$$

$$h_z = 0.3$$



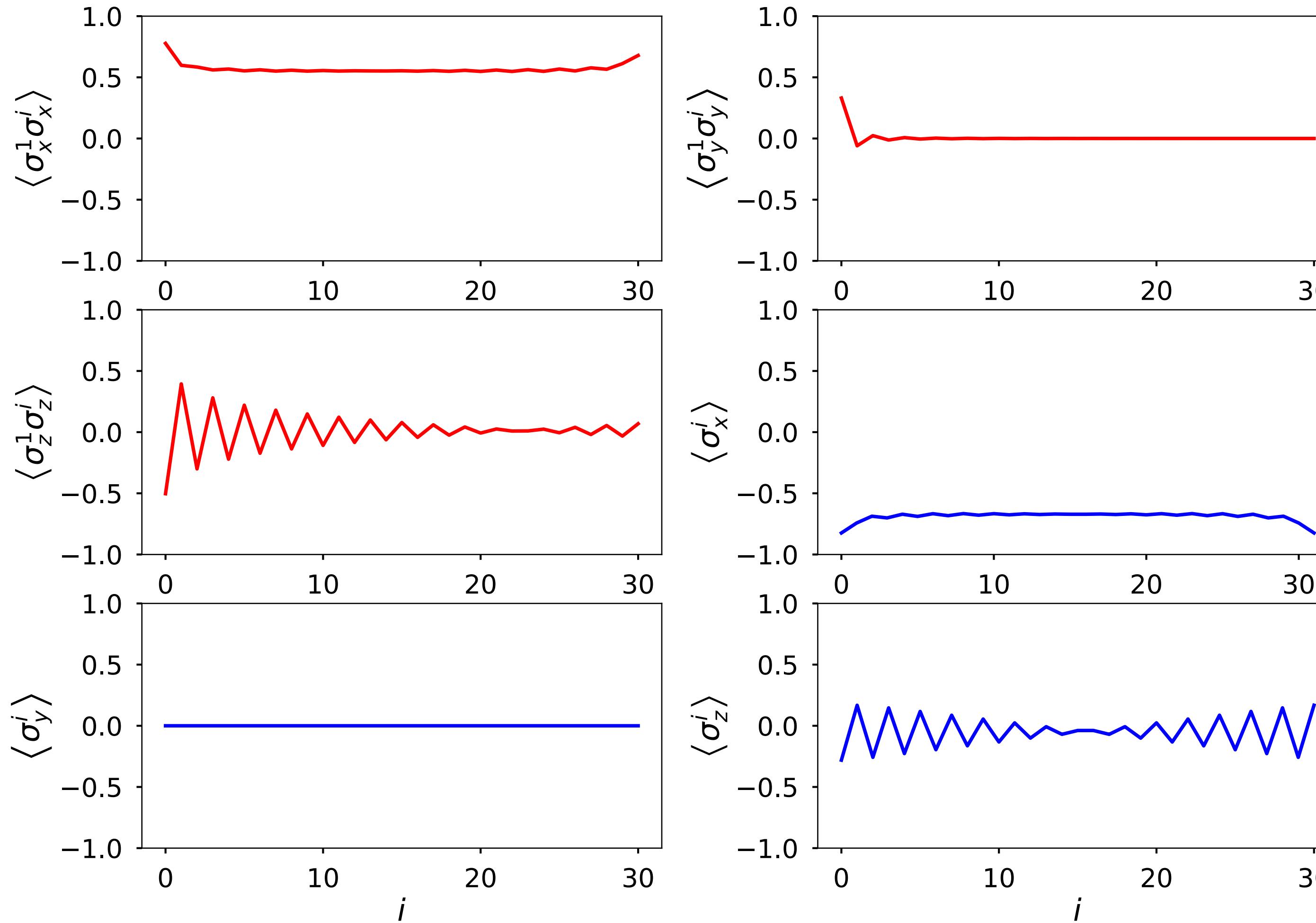
\dot{g}

$\check{\hbar}$

Ground state of TFI hamiltonian

\check{g}

$$H = \sum_{i=1}^{N-1} \sigma_z^i \sigma_z^{i+1} + \sum_{i=1}^N h_x \sigma_x^i + \sum_{i=1}^N h_z \sigma_z^i$$



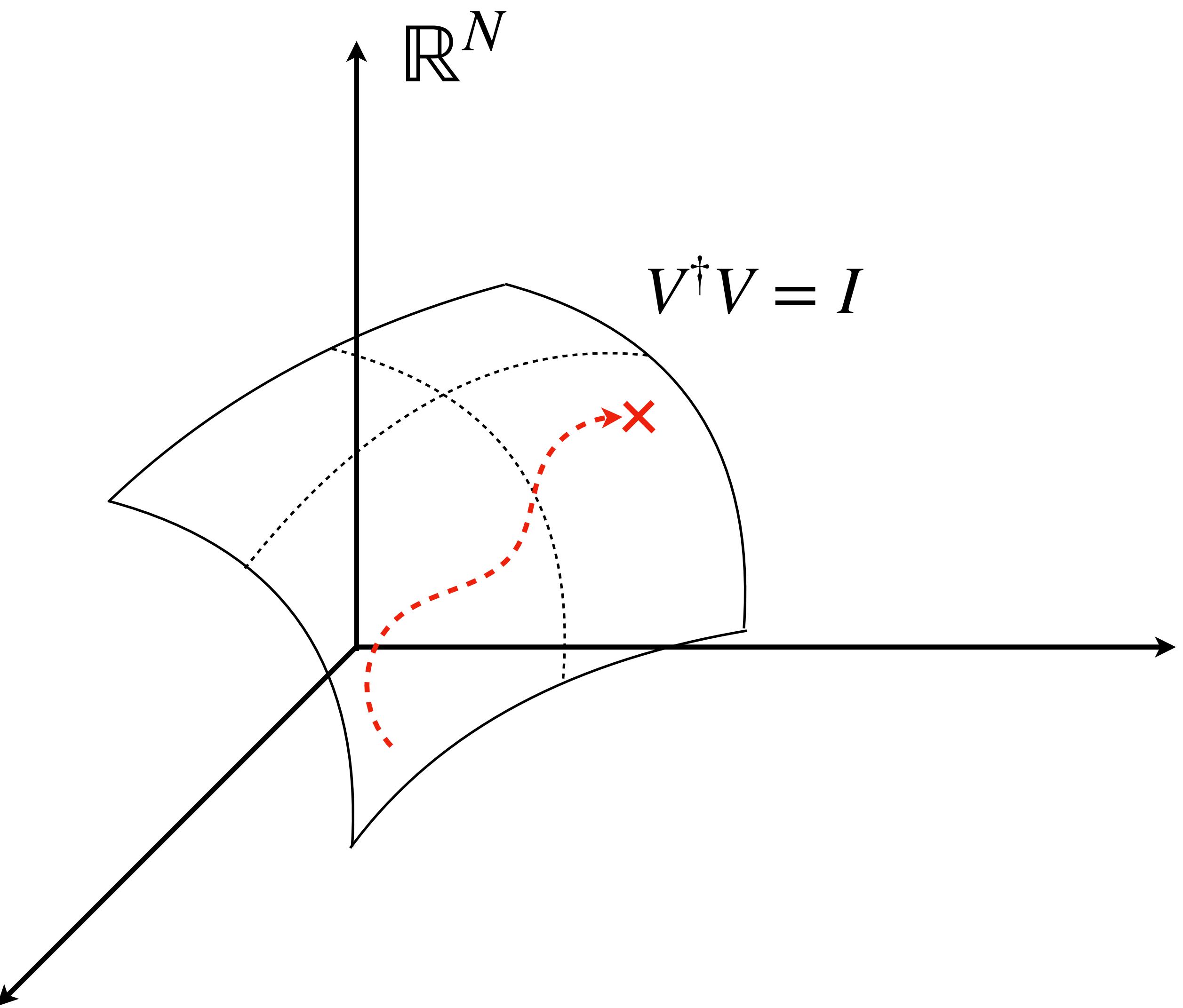
$$N = 32$$

$$h_x = 1$$

$$h_z = 0.3$$



Constraint optimization





QGOpt



<https://qgopt.readthedocs.io/en/latest/>

<https://github.com/LuchnikovI/QGOpt>

Luchnikov I, Krechetov M, Filippov S. Riemannian
optimization and automatic differentiation for complex
quantum architectures. arXiv preprint arXiv:2007.01287.
2020 Jul 2.

Constraint optimization with QGOpt

$V_{n,p} = \{Z \in \mathbb{C}^{n \times m} \mid n \geq m, Z^\dagger Z = I\}$ – Stiefel manifold

Edelman, A., Arias, T. A., & Smith, S. T. (1998). The geometry of algorithms with orthogonality constraints. *SIAM journal on Matrix Analysis and Applications*, 20(2), 303-353.

$S_{++}^n = \{S \in \mathbb{C}^{n \times n} \mid S^\dagger = S, S \succeq 0\}$ – psd matrices

$\rho_{++}^n = \{\rho \in \mathbb{C}^{n \times n} \mid \rho^\dagger = \rho, \rho \succeq 0, \text{Tr}(\rho) = 1\}$ – density matrices

$C_{++}^{n^2} = \left\{ C \in \mathbb{C}^{n^2 \times n^2} \mid C^\dagger = C, C \succeq 0, \text{Tr}_2(C) = I \right\}$ – Choi matrices

$\text{POVM}^{n,m} = \left\{ \{E_i\}_{i=1}^n \mid E_i \in \mathbb{C}^{m \times m}, E_i^\dagger = E_i, E_i \succeq 0 \text{ for } i = 1, \dots, n, \sum_i E_i = I \right\}$ – POVMs

$H_n = \{H \in \mathbb{C}^{n \times m} \mid H = H^\dagger\}$ – Hermitian matrices

All of them are implemented in QGOpt!

See also <https://manopt.org>



Constraint optimization with QGOpt



```
1 stiefel = qgo.manifolds.StiefelManifold()
2 density_matrix = qgo.manifolds.DensityMatrix()
3 choi_matrix = qgo.manifolds.ChiMatrix()
4 hermitian_matrix = qgo.manifolds.HermitianMatrix()
5 positive_cone = qgo.manifolds.PositiveCone()
6 povm = qgo.manifolds.POVM()
```



Constraint optimization with QGOpt



```
1 stiefel = qgo.manifolds.StiefelManifold()
2 density_matrix = qgo.manifolds.DensityMatrix()
3 choi_matrix = qgo.manifolds.ChiMatrix()
4 hermitian_matrix = qgo.manifolds.HermitianMatrix()
5 positive_cone = qgo.manifolds.PositiveCone()
6 povm = qgo.manifolds.POVM()
```

```
1 opt = qgo.optimizers.Adam(m, lr)
```



Constraint optimization with QGOpt



```
1 stiefel = qgo.manifolds.StiefelManifold()
2 density_matrix = qgo.manifolds.DensityMatrix()
3 choi_matrix = qgo.manifolds.ChiMatrix()
4 hermitian_matrix = qgo.manifolds.HermitianMatrix()
5 positive_cone = qgo.manifolds.PositiveCone()
6 povm = qgo.manifolds.POVM()
```

```
1 opt = qgo.optimizers.Adam(m, lr)
```



Constraint optimization with QGOpt



```
1 stiefel = qgo.manifolds.StiefelManifold()
2 density_matrix = qgo.manifolds.DensityMatrix()
3 choi_matrix = qgo.manifolds.ChiMatrix()
4 hermitian_matrix = qgo.manifolds.HermitianMatrix()
5 positive_cone = qgo.manifolds.PositiveCone()
6 povm = qgo.manifolds.POVM()
```

```
1 opt = qgo.optimizers.Adam(m, lr)
```

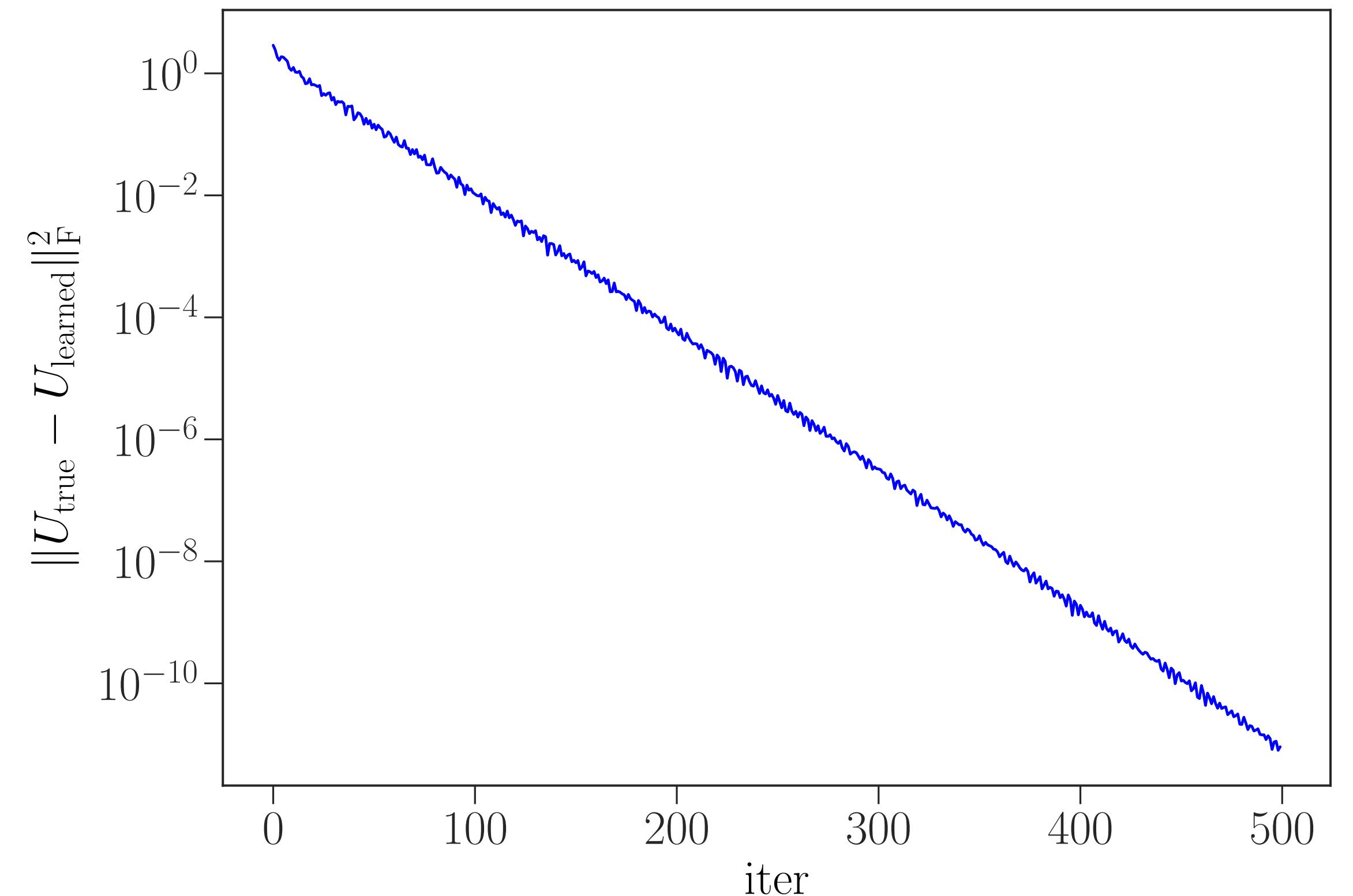
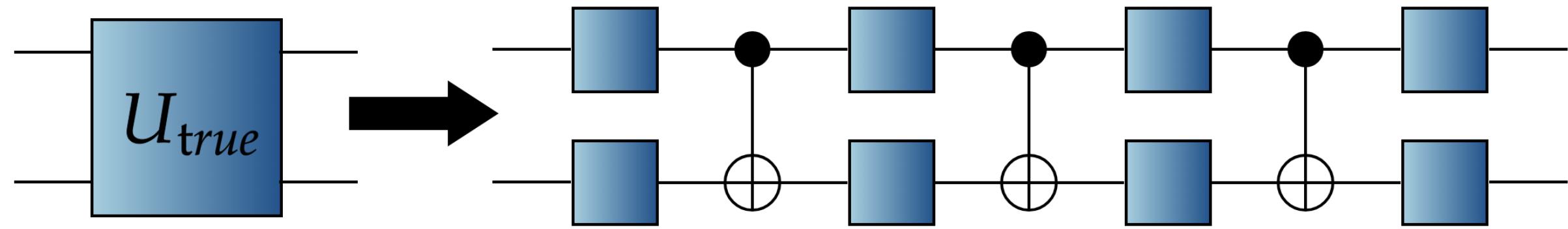
```
1 opt.apply_gradients(zip(loss, grad))
```

\check{h}

Gate decomposition

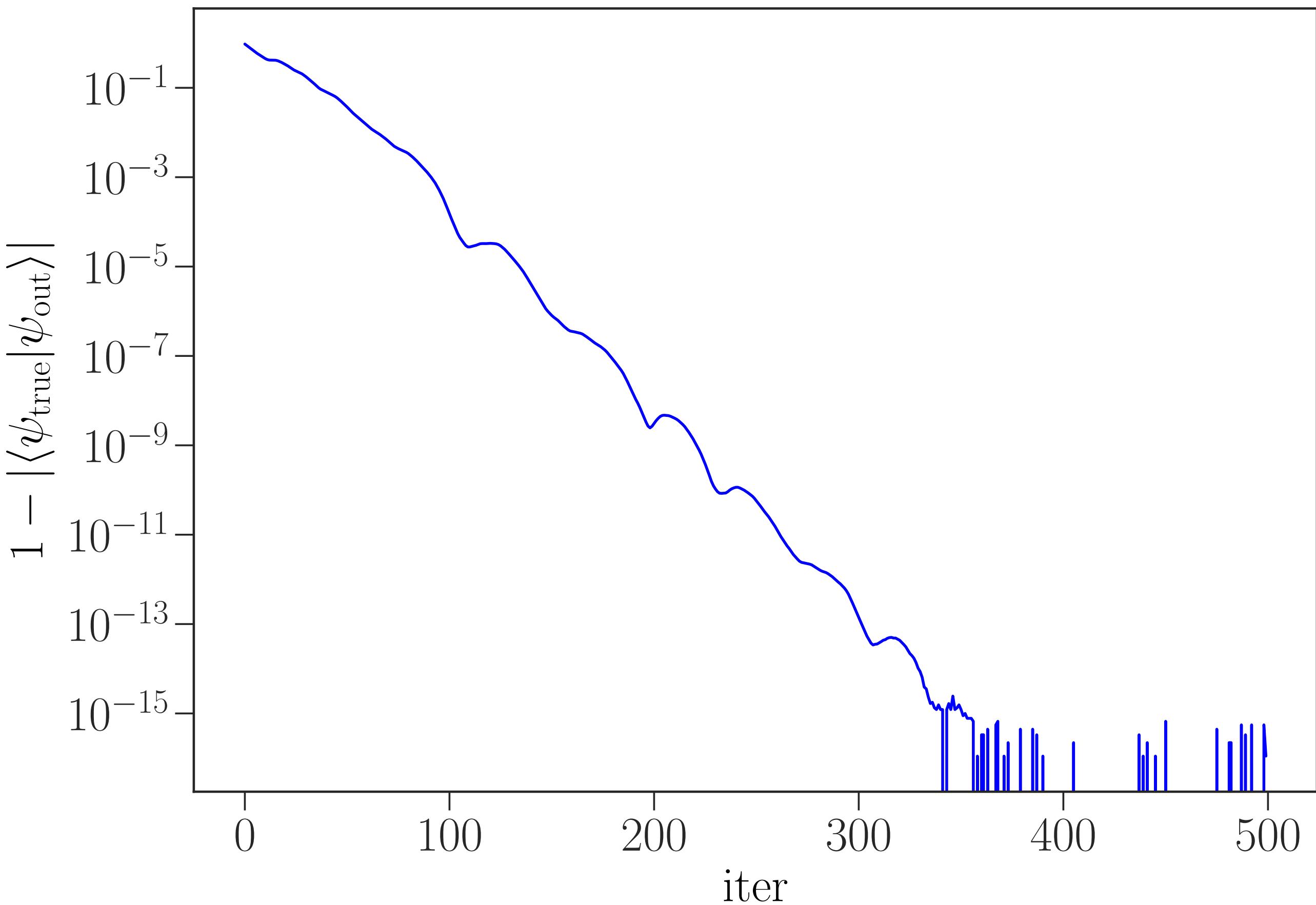
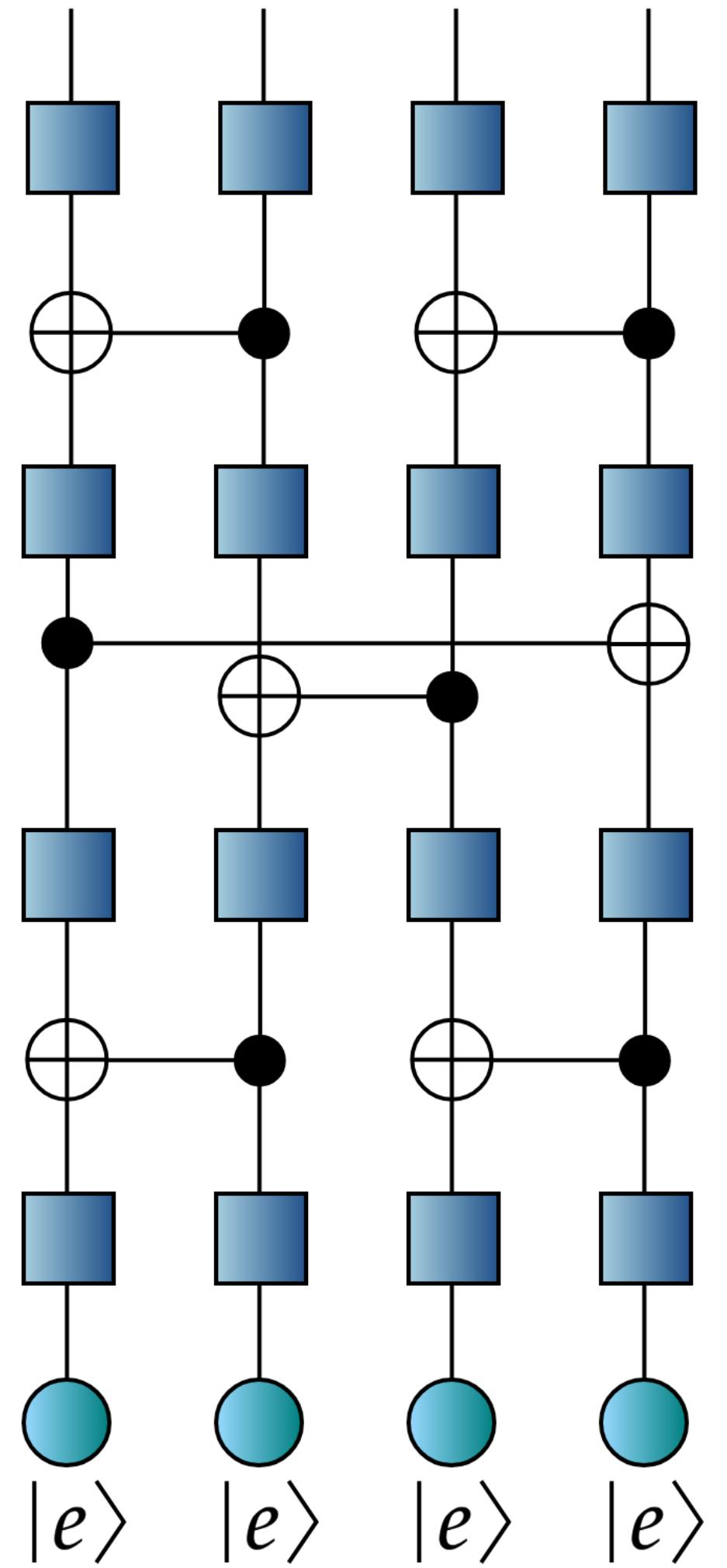
\dot{g}

Optimization over Stiefel Manifold



State preparation

Optimization over Stiefel Manifold



Entanglement renormalization

a) $H^{(0)} = \boxed{\text{---}} = \boxed{\text{---}} + \boxed{\text{---}} + \dots$

$$h_{i,i+1}^{(0)} = \boxed{\text{---}}$$

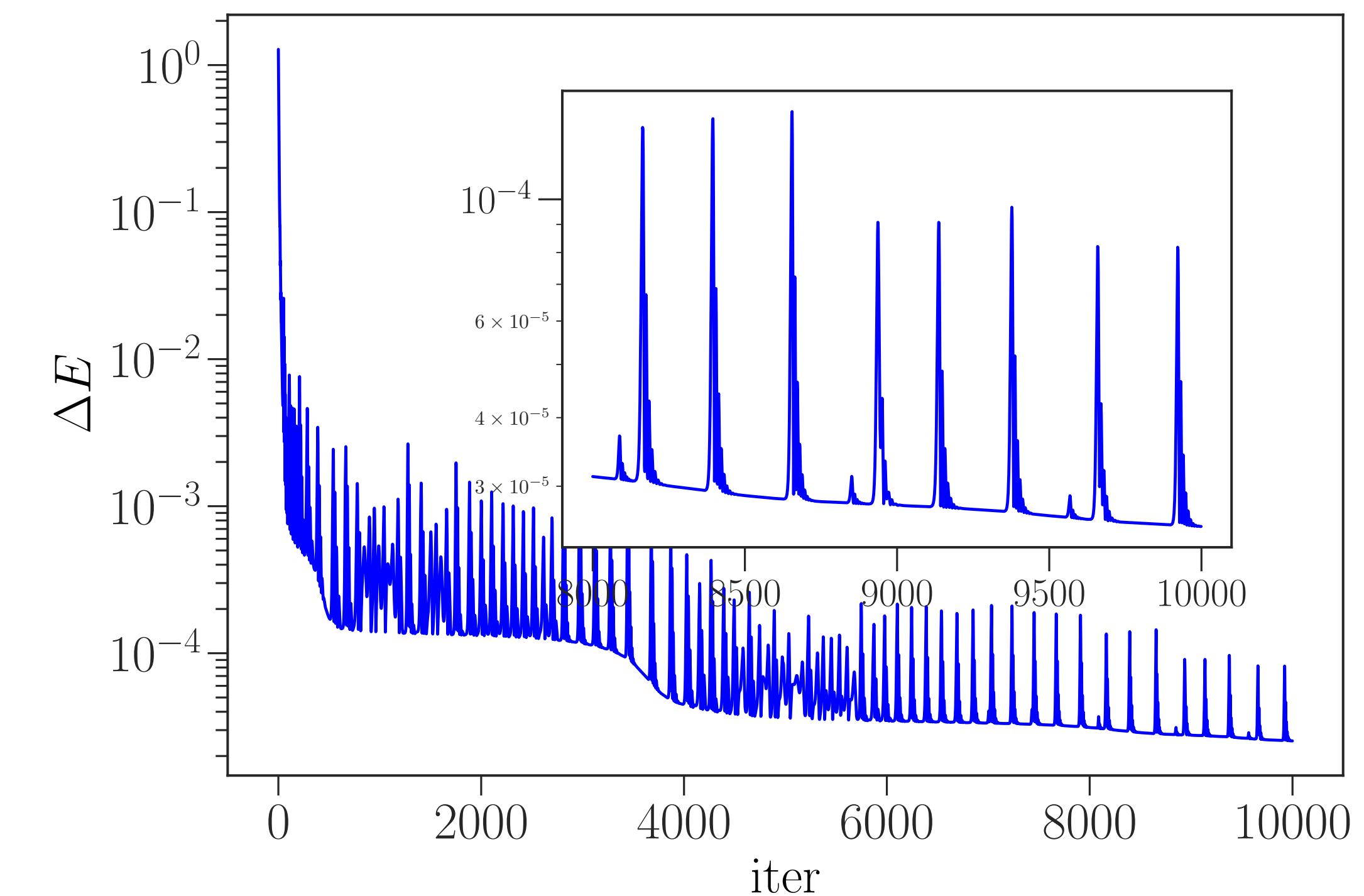
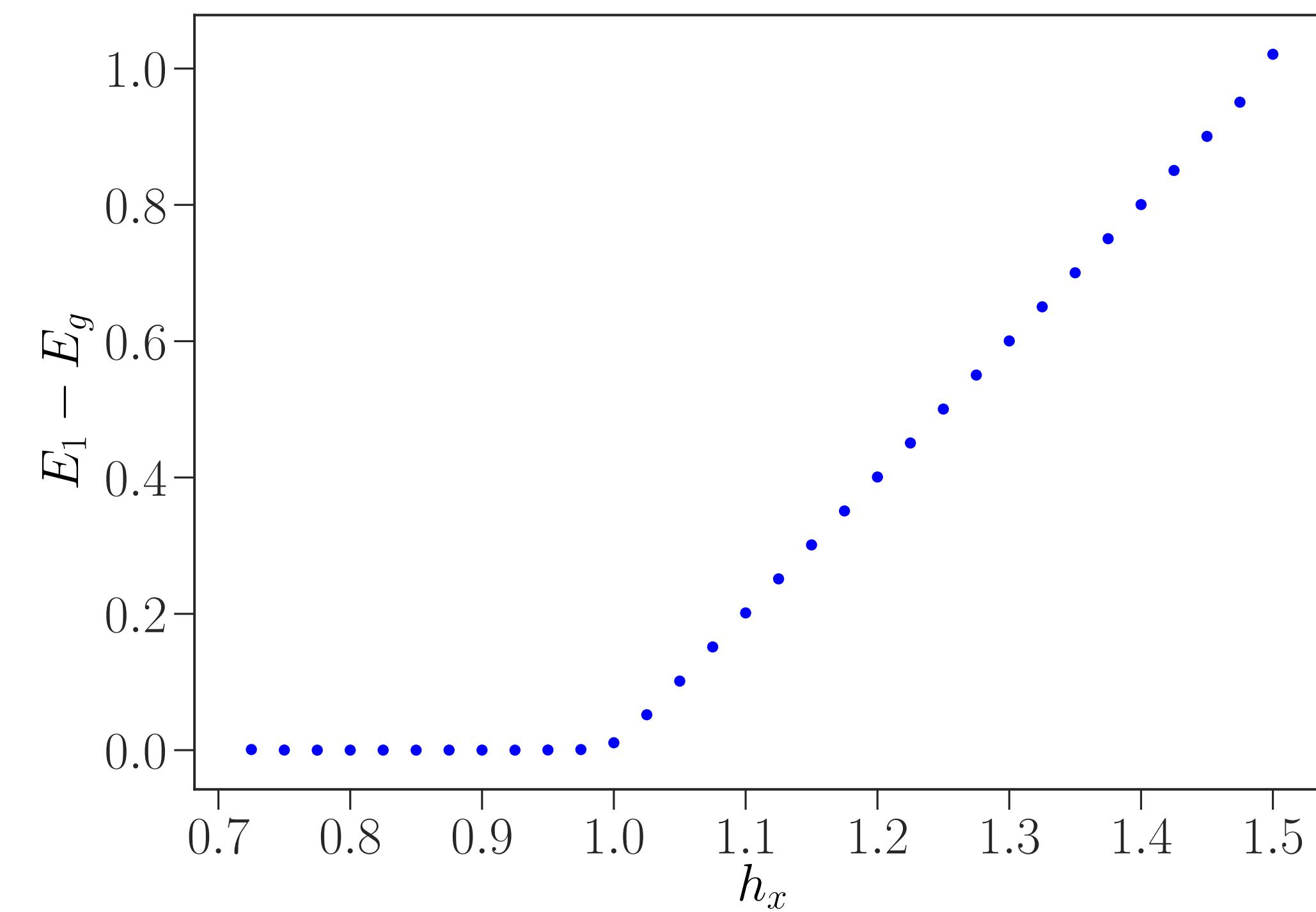
b)

$$H^{(1)} = \boxed{\text{---}} = \boxed{\text{---}} + \boxed{\text{---}} + \boxed{\text{---}} + \dots + \boxed{\text{---}} + \dots$$

486 quantum spins TFI model

$$H = \sigma_N^z \sigma_1^z + \sum_{i=1}^{N-1} \sigma_i^z \sigma_{i+1}^z + h_x \sum_{i=1}^N \sigma_i^x$$

Evenbly, G., & Vidal, G. (2009). Algorithms for entanglement renormalization. *Physical Review B*, 79(14), 144108.



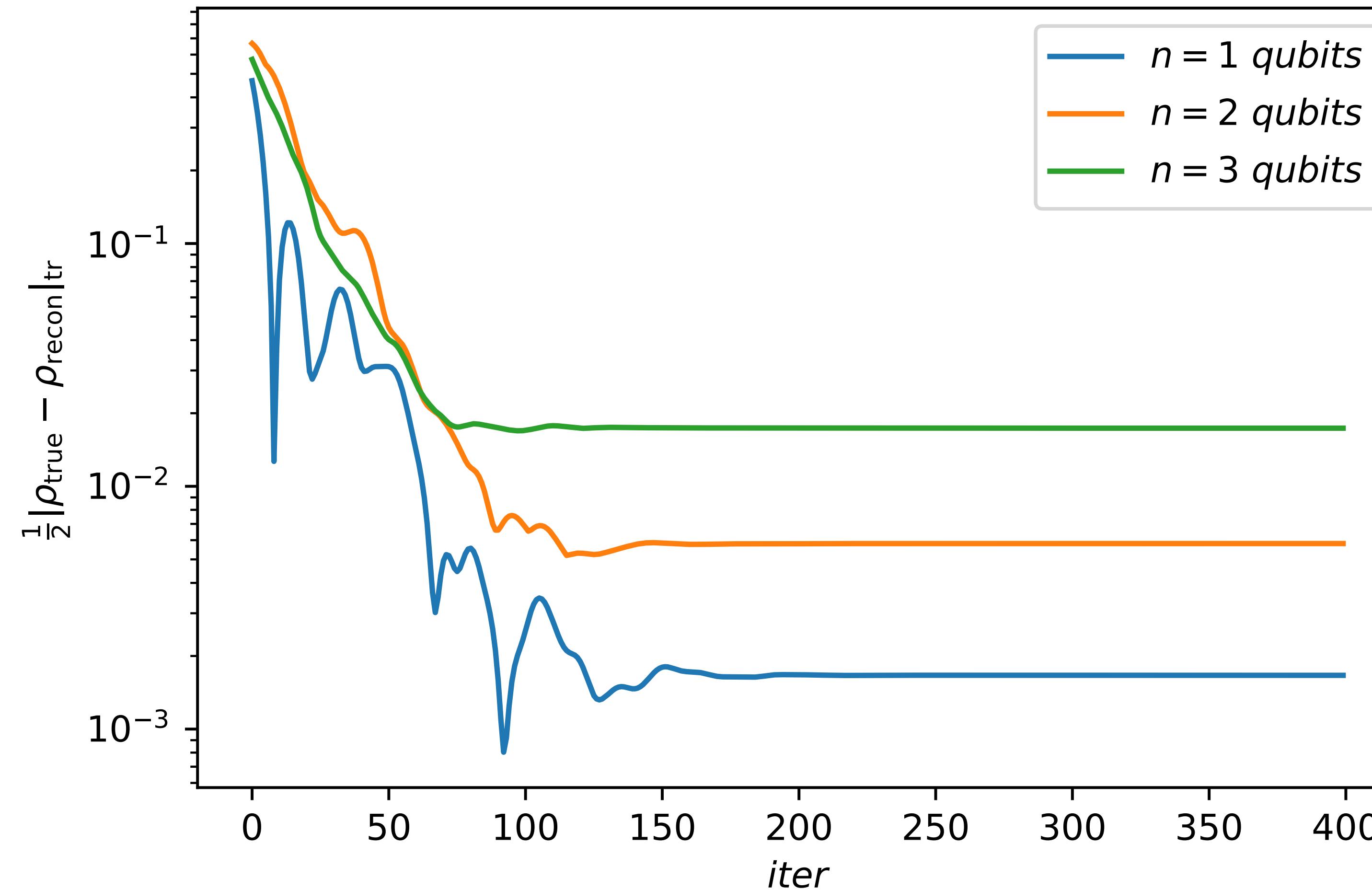


Tomography of quantum states



600 000 measurements, tetrahedral povm

Optimization over manifold of density matrices



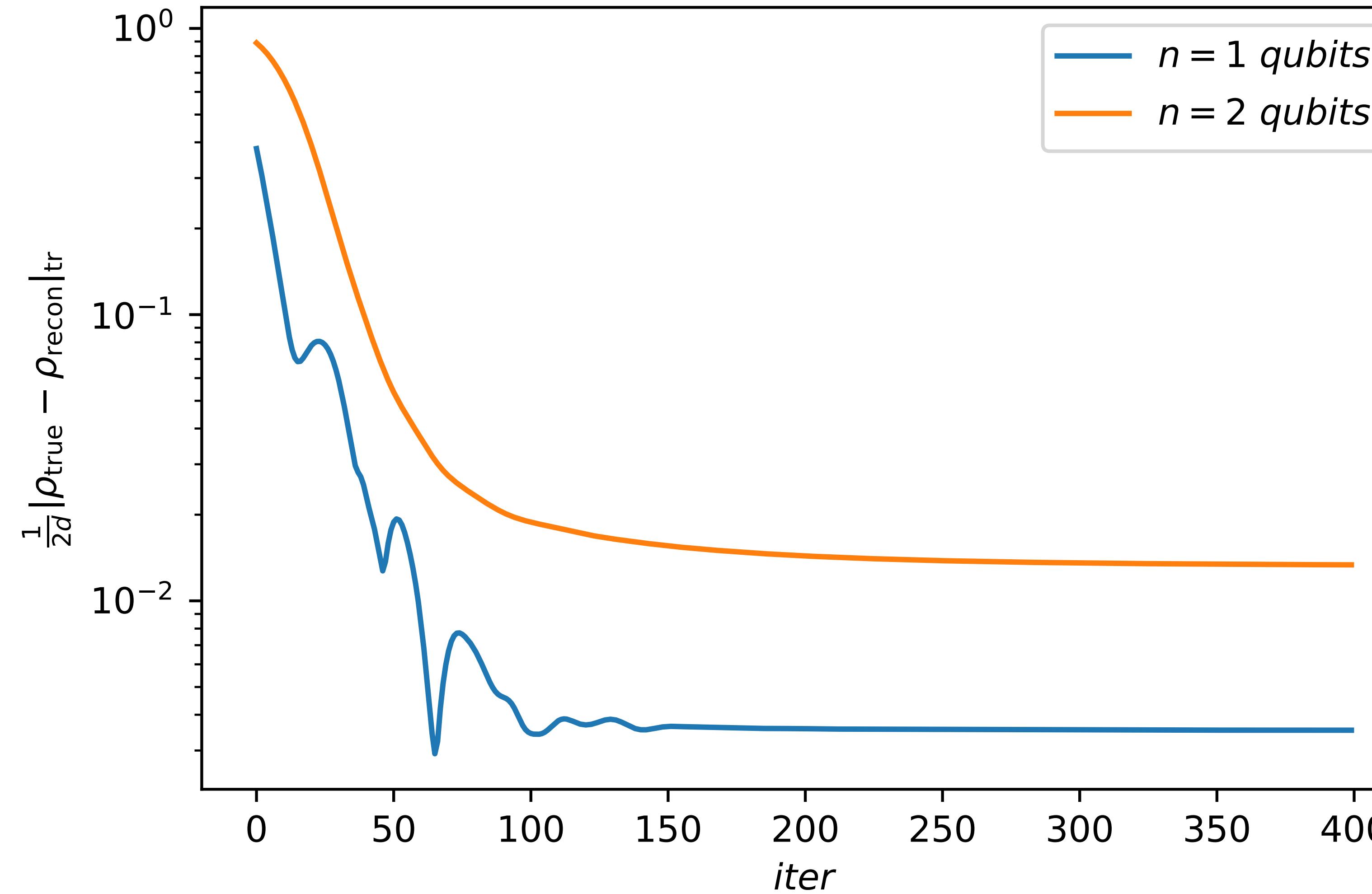


Tomography of quantum channels



600 000 measurements, tetrahedral povm and random input state

Optimization over manifold of Choi matrices





QGOpt tutorials



To get familiar with QGOpt API please visit

https://qgopt.readthedocs.io/en/latest/quick_start.html



Summary



- Machine learning with tensor networks
- Quantum control
- Model identification
- Tomography of states and channels
- Many body systems, quantum phase transitions
- Adaptive quantum tomography
- Combinations of approaches (like in deep learning)

h

g

Thank you for your attention!