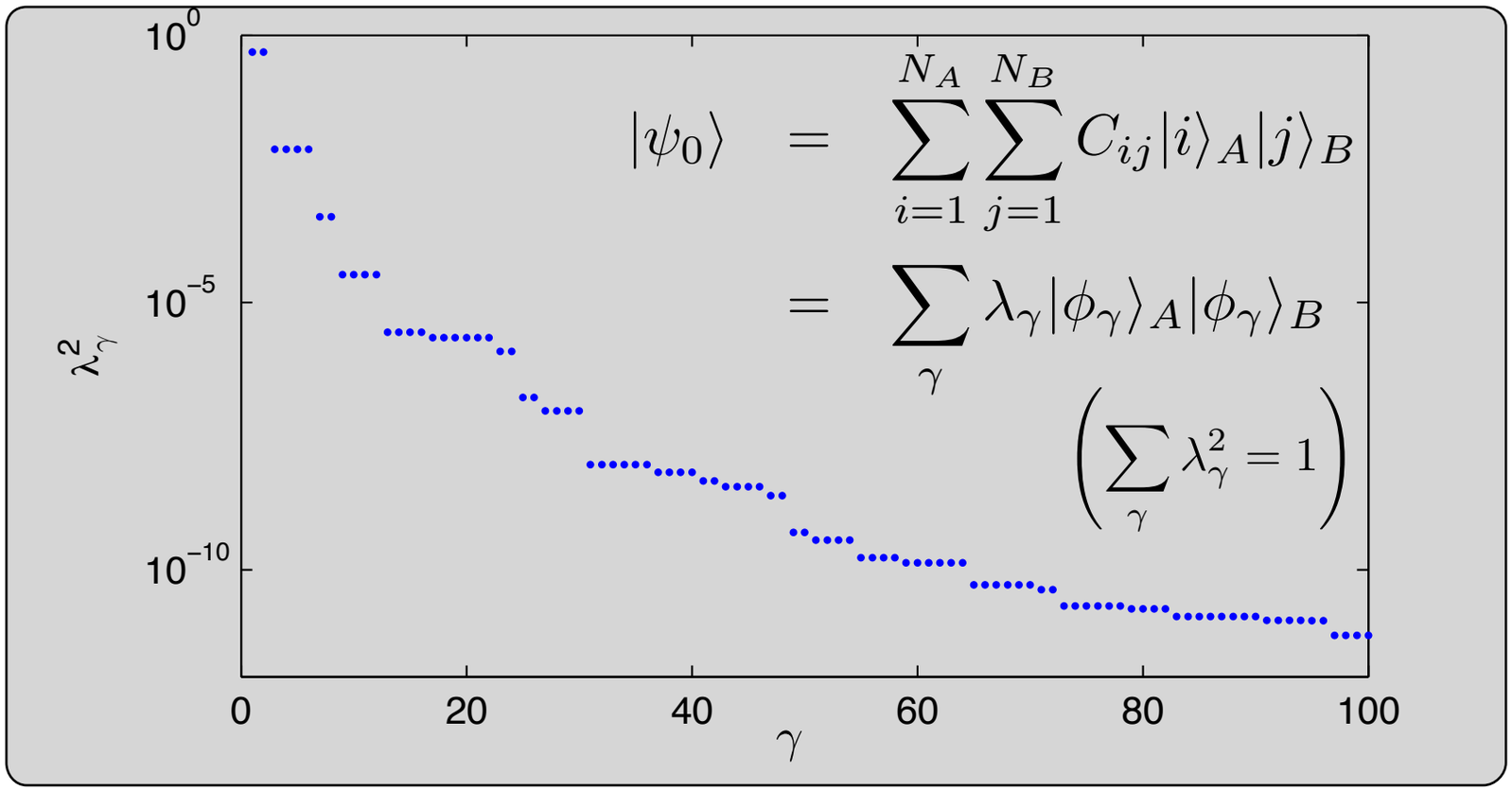
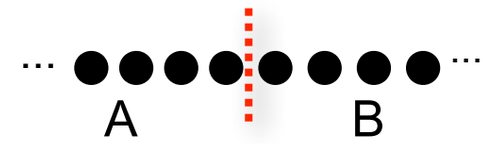
Entanglement

- Example: Spin-1 Heisenberg chain $H = \sum_j \vec{S}_j \cdot \vec{S}_{j+1}$





- **Schmidt values decay rapidly:** Almost the entire weight is contained in only **few important states (AREA LAW!)**

Time evolution of Matrix Product States

- Apply Suzuki-Trotter decomposition of order p

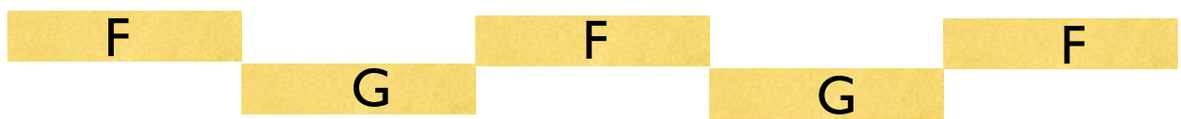
  $$\exp\left(-i(F+G)\delta t\right) \approx f_p\left[\exp(-F\delta t), \exp(-G\delta t)\right]$$

  with $f_1(x,y) = xy$, $f_2(x,y) = x^{1/2}yx^{1/2}$, etc.

- Two chains of two-site gates

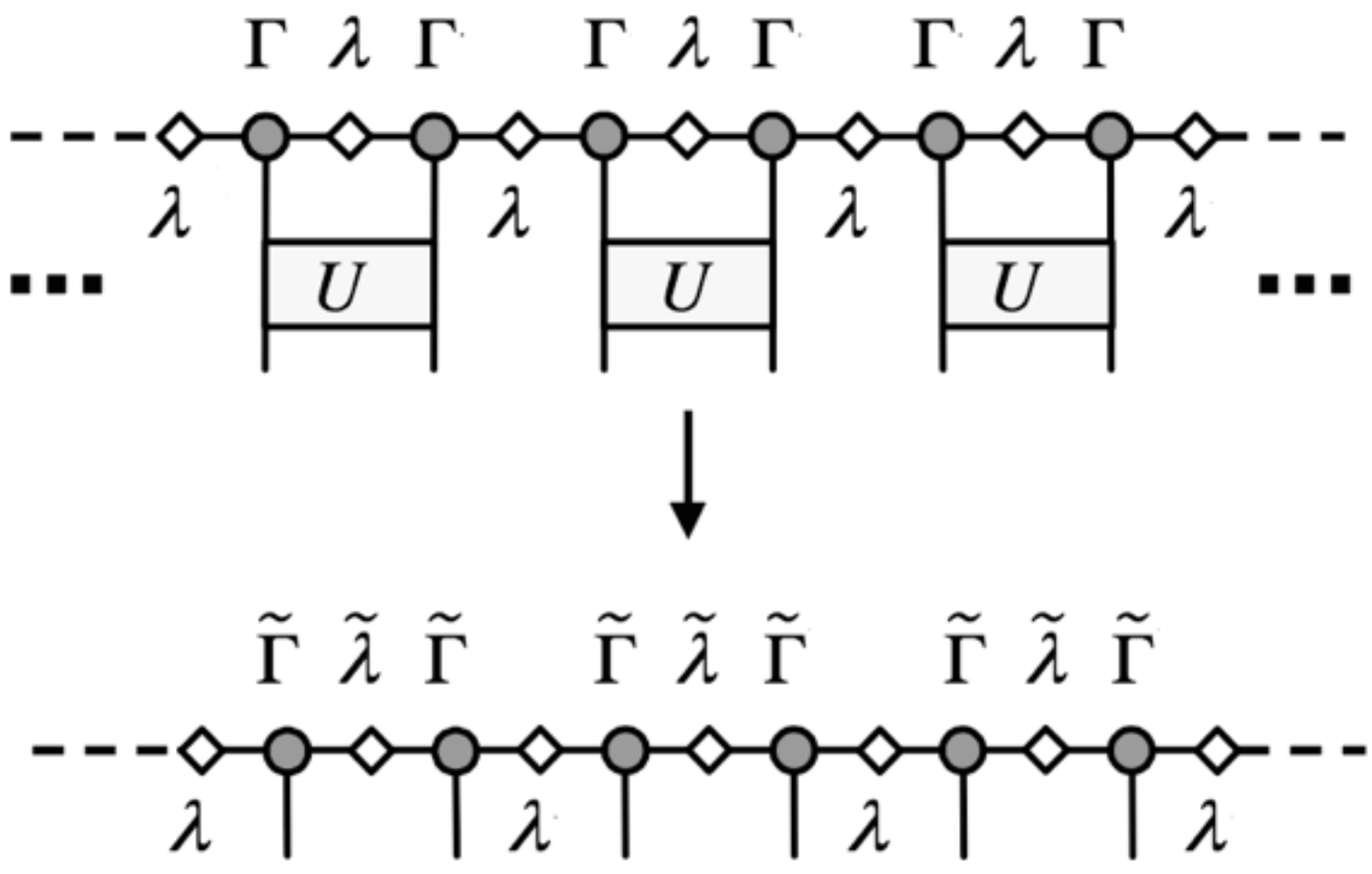
$$U_F = \prod_{\text{even } r} \exp(-iF^{[r]}\delta t)$$

$$U_G = \prod_{\text{odd } r} \exp(-iG^{[r]}\delta t)$$



Time evolution of Matrix Product States

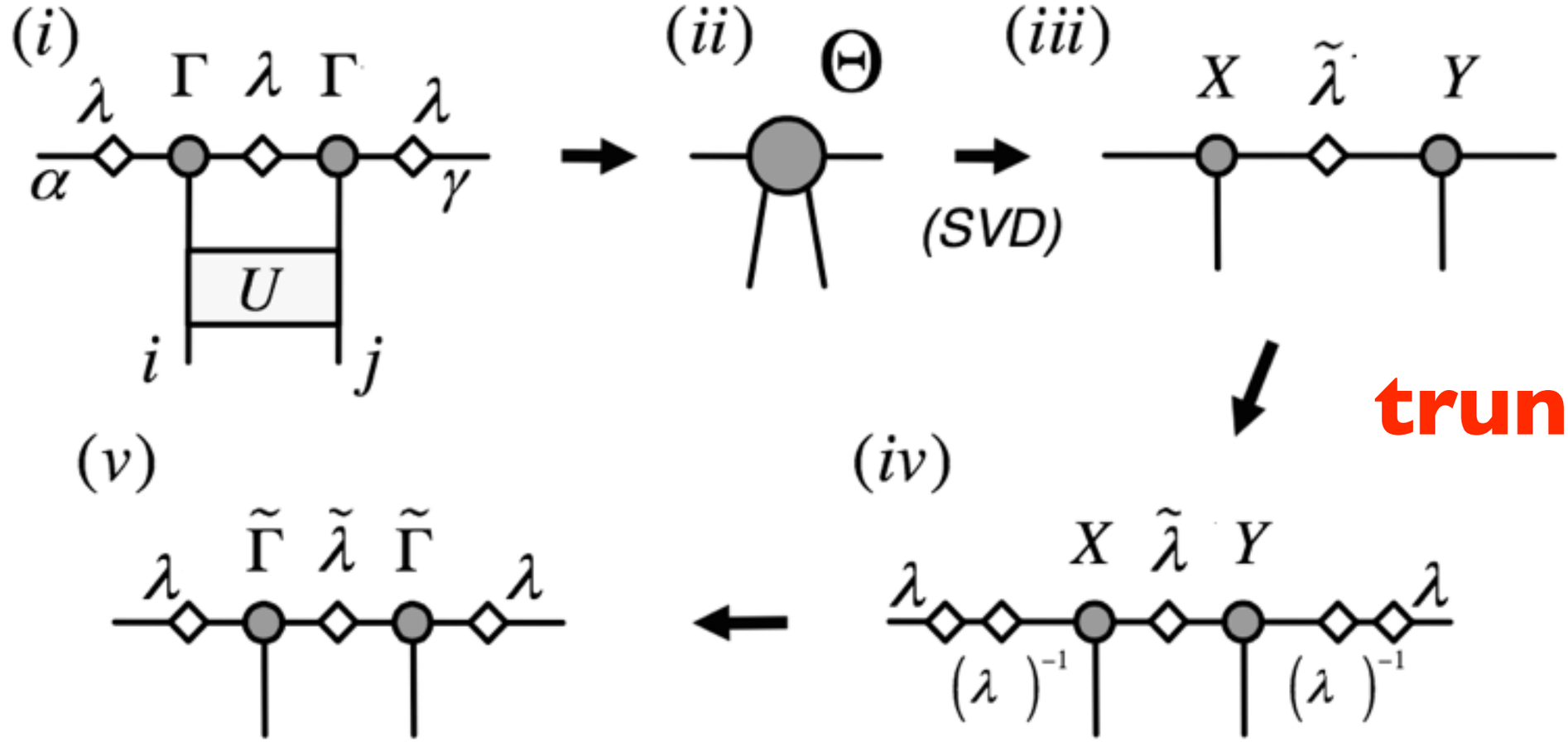
- Time Evolving Block Decimation algorithm [Vidal 03]



- How do we get the original form back?

Time evolution of Matrix Product States

- Time Evolving Block Decimation algorithm [Vidal 03]



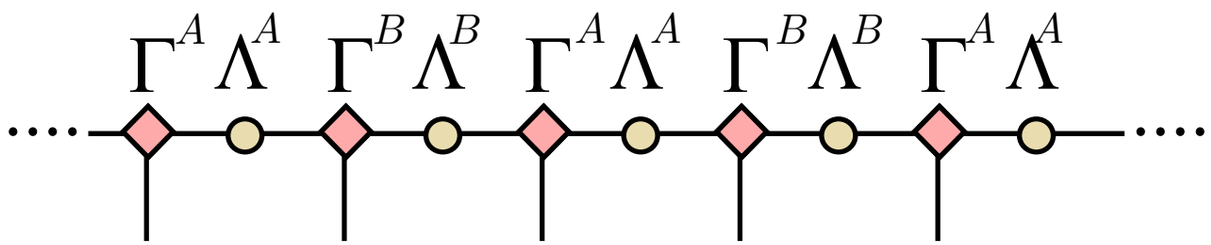
- Scales with the matrix dimension as χ^3

Time evolution of Matrix Product States

iTEBD algorithm [Vidal 07]:

- Assume that $|\psi\rangle$ is translational invariant and $N = \infty$
- Partially break translational symmetry to simulate the action of the gates

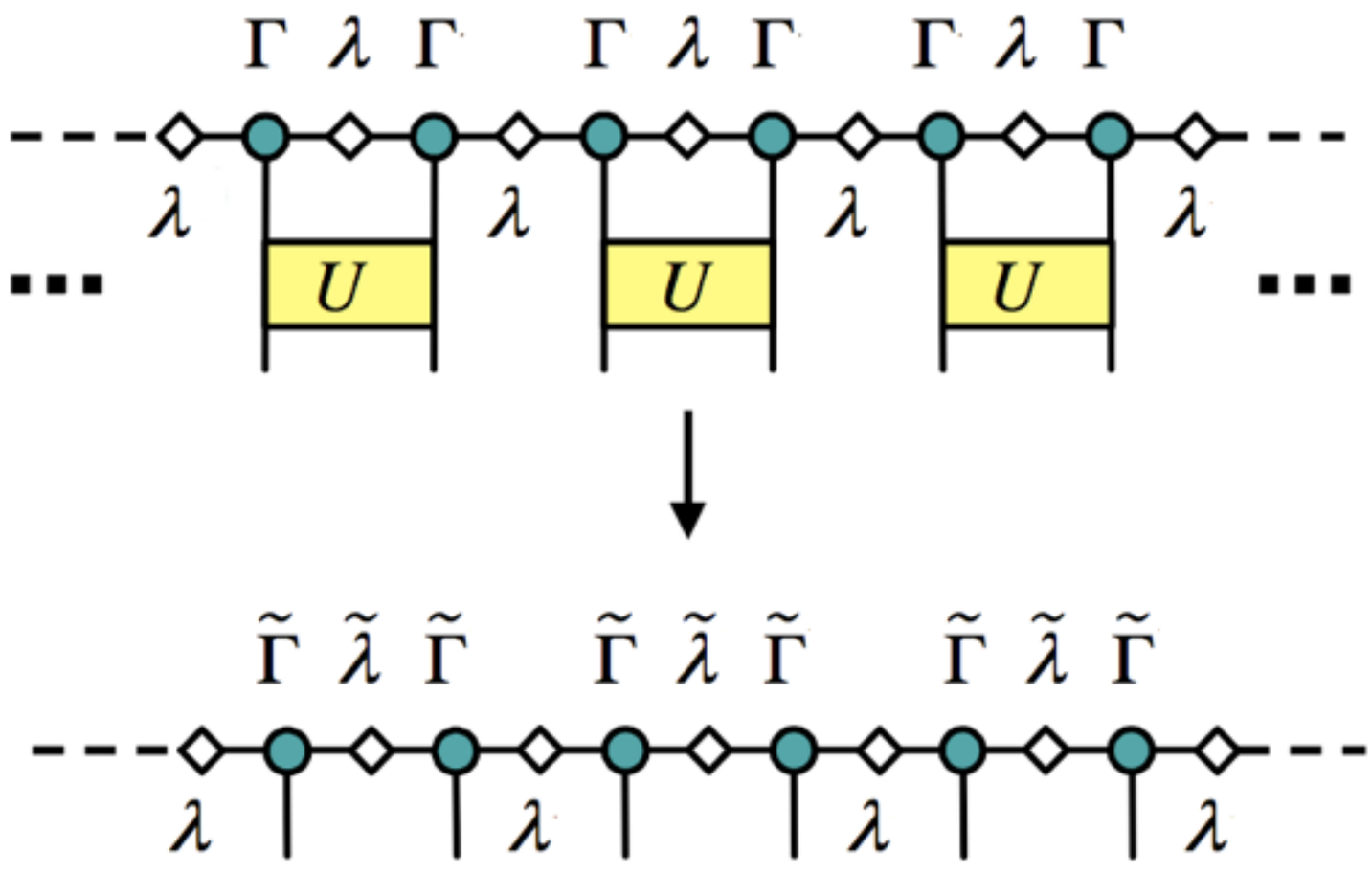
$$\Gamma^{[2r]} = \Gamma^A,\ \lambda^{[2r]} = \lambda^A,\ \Gamma^{[2r+1]} = \Gamma^B,\ \lambda^{[2r+1]} = \lambda^B$$



- **Time evolution achieved by repeated local application of gates (parallel)**
- Computational space / time are $O(d^2\chi^2)$ / $O(d^3\chi^3)$

Time evolution of Matrix Product States

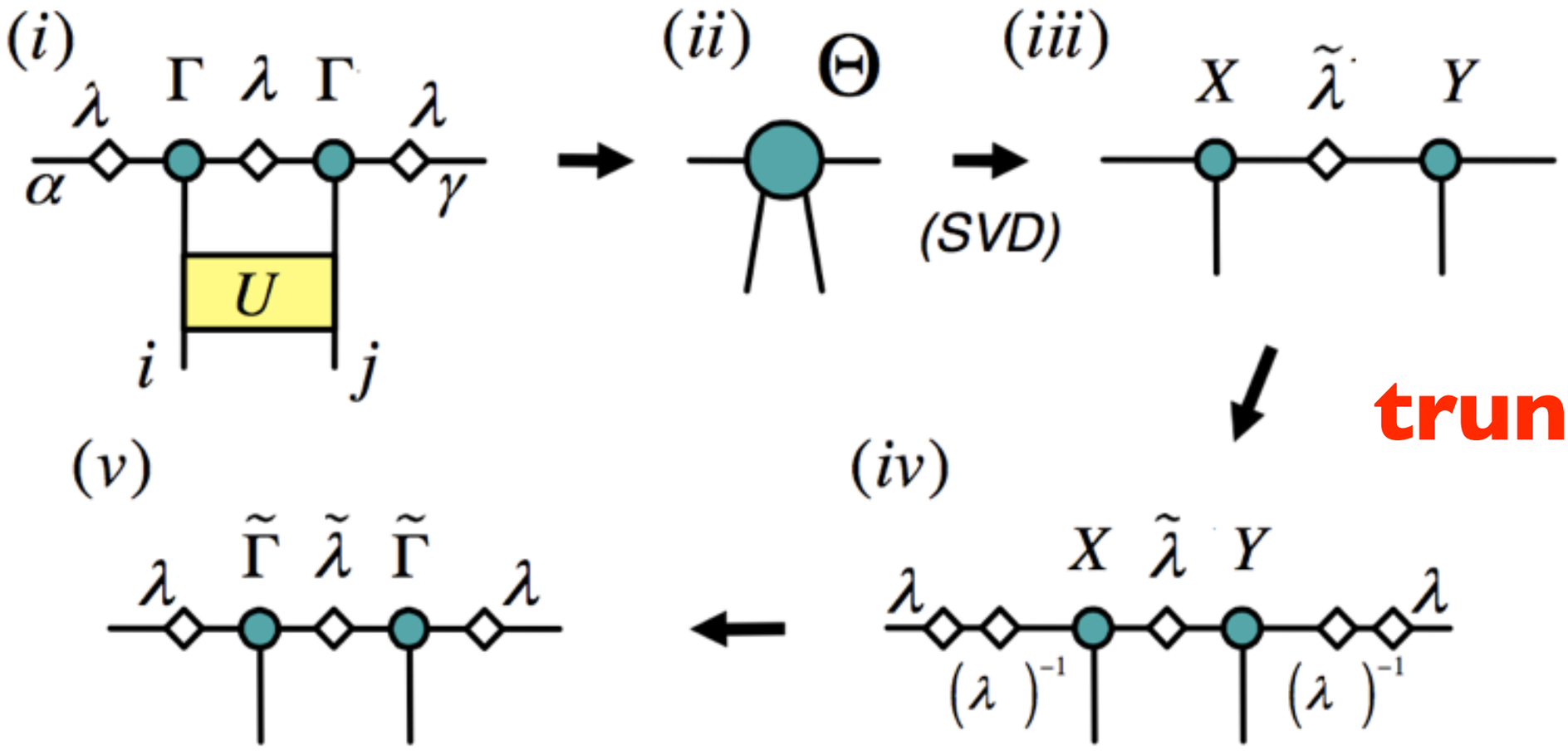
- Time Evolving Block Decimation algorithm [Vidal 03]



- How do we get the original form back?

Time evolution of Matrix Product States

- Time Evolving Block Decimation algorithm [Vidal 03]



Python!

- Python + numpy provide nice tools to simply implement the algorithm as it contains the key functions already

```
X=tensordot(Y,Z,axes=(1,0))
```

$$X_{ijk} = \sum_m Y_{im} Z_{mjk}$$

```
X=reshape(X,(dim1*dim2,dim3))
```

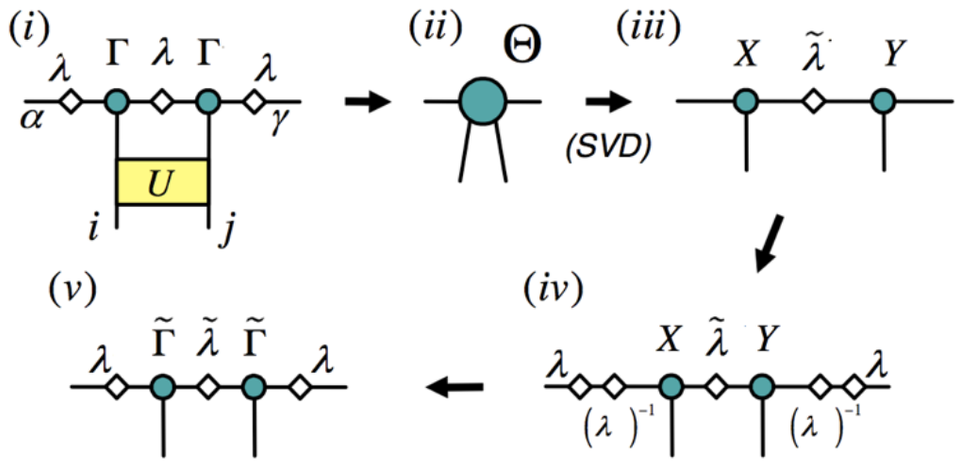
$$X_{ijk} \rightarrow X_{(ij)k}$$

```
X=transpose(X,(0,2,1))
```

$$X_{ijk} \rightarrow X_{ikj}$$

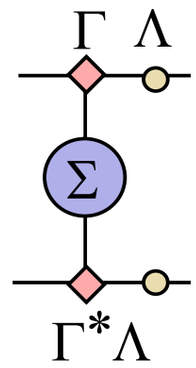
Python : TEBD

```
# First define the parameters of the model / simulation
J=1.0; g=0.5; chi=5; d=2; delta=0.01; N=1000;
G = np.random.rand(2,d,chi,chi);l = np.random.rand(2,chi)

# Generate the two-site time evolution operator
H = np.array( [[J,-g/2,-g/2,0], [-g/2,-J,0,-g/2], [-g/2,0,-J,-g/2], [0,-g/2,-g/2,J]] )
U = np.reshape(expm(-delta*H),(2,2,2,2))

# Perform the imaginary time evolution alternating on A and B bonds
for step in range(0, N):
    A = np.mod(step,2); B = np.mod(step+1,2)

    # Construct theta
    theta = np.tensordot(np.diag(l[B,:]),G[A,:,:,:],axes=(1,1))
    theta = np.tensordot(theta,np.diag(l[A,:],0),axes=(2,0))
    theta = np.tensordot(theta,G[B,:,:,:],axes=(2,1))
    theta = np.tensordot(theta,np.diag(l[B,:],0),axes=(3,0))

    # Apply U
    theta = np.tensordot(theta,U,axes=([1,2],[0,1]))

    # SVD
    theta = np.reshape(np.transpose(theta,(2,0,3,1)),(d*chi,d*chi))
    X, Y, Z = np.linalg.svd(theta); Z = Z.T

    # Truncate
    l[A,0:chi]=Y[0:chi]/np.sqrt(sum(Y[0:chi]**2))

    X=np.reshape(X[0:d*chi,0:chi],(d,chi,chi))
    G[A,:,:,:]=np.transpose(np.tensordot(np.diag(l[B,:]**(-1)),X,axes=(1,1)),(1,0,2))

    Z=np.transpose(np.reshape(Z[0:d*chi,0:chi],(d,chi,chi)),(0,2,1))
    G[B,:,:,:]=np.tensordot(Z,np.diag(l[B,:]**(-1)),axes=(2,0))

print "E_iTEBD =", -np.log(np.sum(theta**2))/delta/2
```



Python : Projective representation

```
def representation(B_list,R,chi):
    # Construct the mixed transfermatrix
    T = np.tensordot(R,B_list[0],axes=(0,0))
    T = np.tensordot(T,np.conj(B_list[0]),axes=(0,0))
    T = np.tensordot(T,B_list[1],axes=(1,1))
    T = np.tensordot(R,T,axes=(0,3))
    T = np.tensordot(T,np.conj(B_list[1]),axes=([0,3],[0,1]))
    T = np.reshape(T,(chi**2,chi**2))

    # Obtain the dominant eigenvector
    eta,v = np.linalg.eig(T)
    idx = np.argsort(eta)[::-1]
    eta = eta[idx]
    v = v[:,idx]
    return eta[0],np.reshape(v[:,0],(chi,chi))
```




```
############### Find the Ground state ########################
for step in range(1, N):
    tebd(B_list,s_list,U,chi_list,chi_max)

############### Get the topological invariant ###############
eta_x,U_x = representation(B_list,expm(1j*np.pi*Sx),chi_max)
eta_z,U_z = representation(B_list,expm(1j*np.pi*Sz),chi_max)
I = np.trace(np.dot(np.dot(np.dot(U_x,U_z),np.conj(U_x.T)),np.conj(U_z.T)))
```

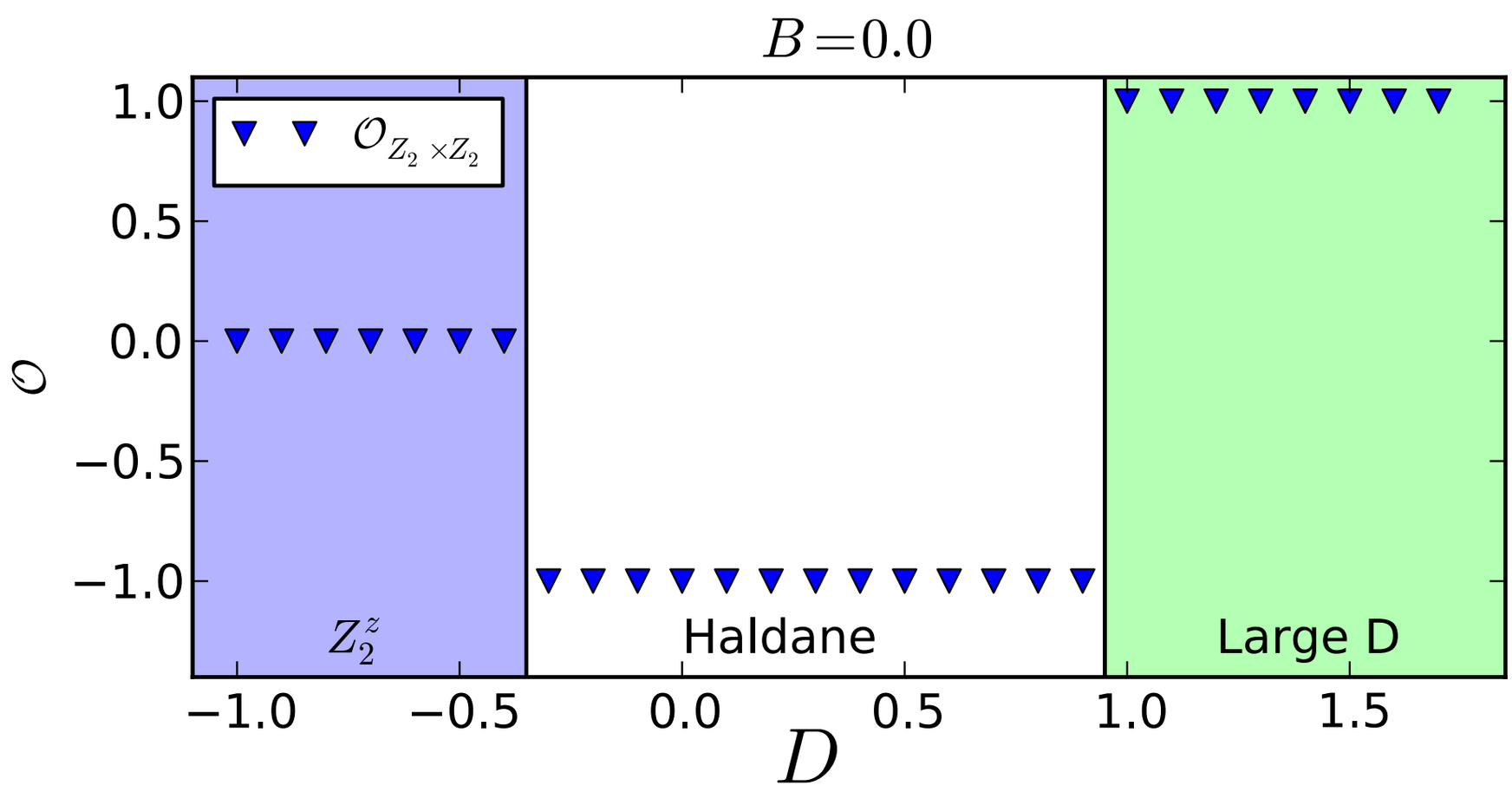
$$U_x U_z U_x^\dagger U_z^\dagger$$

1D Symmetry protected topological phases

- S=1 chain $H = \sum_j \vec{S}_j \cdot \vec{S}_{j+1} + D \sum_j (S_j^z)^2$

- $\mathbb{Z}_2 \times \mathbb{Z}_2$ stabilizes Haldane phase

$$\mathcal{O}_{Z_2 \times Z_2} = \begin{cases} 0 & \text{if symmetry broken} \\ \frac{1}{\chi} \mathrm{tr} \left(U_x U_z U_x^\dagger U_z^\dagger \right) & \text{if symmetry not broken} \end{cases} .$$



iMPS obtained using the iTEBD / iDRMG method

1D Symmetry protected topological phases

- **Non-local order parameter** "detects" cohomology class from ground state wave function

- **Inversion** symmetry based order parameter distinguishes Haldane and trivial Phase

$$H = \sum_j \vec{S}_j \cdot \vec{S}_{j+1} + D \sum_j (S_j^z)^2$$

$$\mathcal{O}_I =$$