Hands-on DMRG methods with

ITensor

http://itensor.org

ITensor

http://itensor.org

C++ library for tensor network wavefunctions

Includes tensor classes, matrix product states, DMRG

Useful for “post DMRG” methods:
- MPS algorithms (time evolution, METTS)
- MERA
- PEPS

Contains both complete algorithms and “building blocks” - customizable at every level

C++

lines end with ;

Basic data types:

```
int i = 5;

Real r = 2.3456;

string s = “some string”;
```

Printing:

```
println(i); //prints “5”

println(r); //prints “2.3456”
```

C++

User defined types (objects)

Construct an object of type MyClass:

```
MyClass m("MyClass m", 5);
```

Objects define various methods:

```
m.doThing();

m.setValue(6);

println(m.name());
```

C++

Some objects can be called like functions:

```
FType f;

int j = f(5);
```

Other objects can be used like numbers:

```
Numerical x(1.),y(2.);

Numerical r = x + y;

println(r.value());  //prints 3
```

01 One Site

Start with a single-site wavefunction,
for example a spin 1/2.

Single-site basis:

$$|\mathrm{s}{=}1\rangle = |\!\uparrow\rangle$$

$$|\mathrm{s}{=}2\rangle = |\!\downarrow\rangle$$

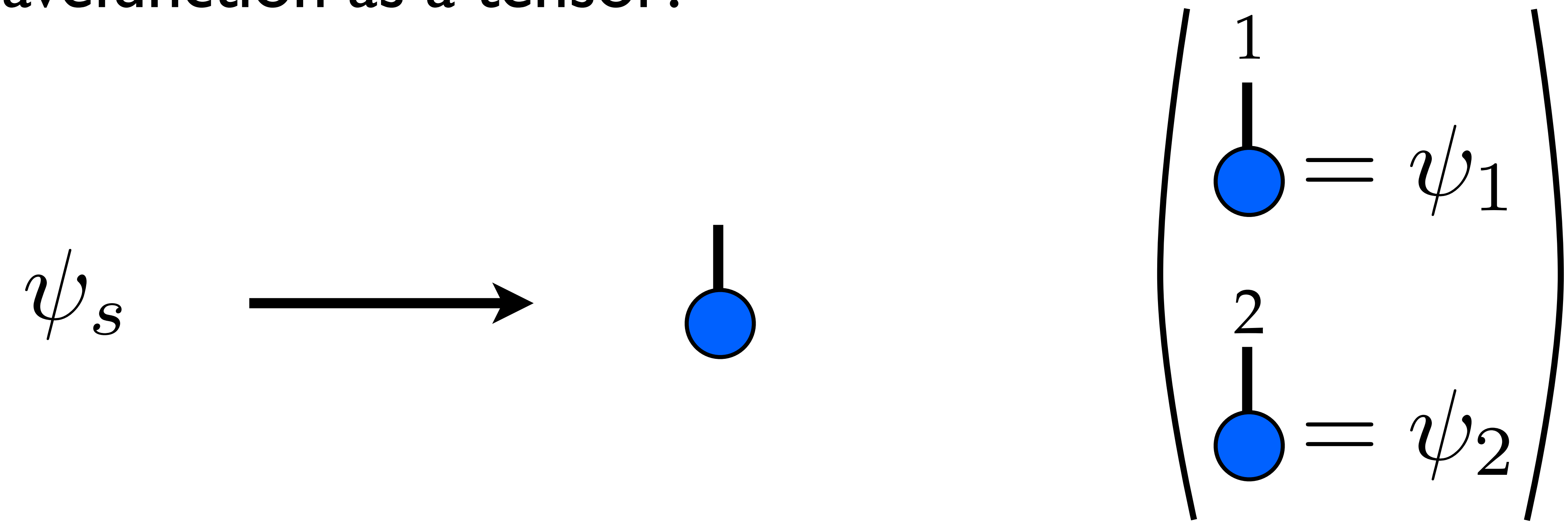
Most general wavefunction for a spin 1/2:

$$|\psi\rangle = \sum_{s=1}^{2} \psi_s |s\rangle$$

The ψ_s are complex numbers.

Slight abuse of notation, may refer to either $|\psi\rangle$ or ψ_s as the wavefunction.

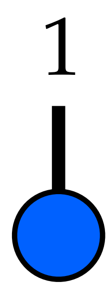
Single-site wavefunction as a tensor:



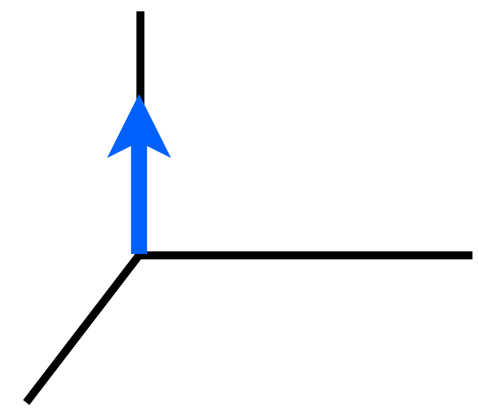
Using ITensor:

```
Index s("s",2);

//"s" gives the name of the Index when printed
// 2 is the dimension/range of the Index

ITensor psi(s); //default initialized to zero
```

Now initialize ψ_s. First choose $|\psi\rangle = |\uparrow\rangle$



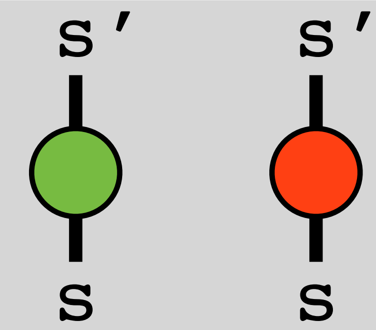
$= 1$



```
Index s("s",2);
ITensor psi(s);      // Prints:
                     // psi =
psi(s(1)) = 1;       // ITensor r=1: s/Link-79180:2
                     //   (1) 1.0000000000
PrintData(psi);
```

Make some operators:

```
ITensor Sz(s,prime(s));
ITensor Sx(s,prime(s));
```



New ITensors start out full out zeros

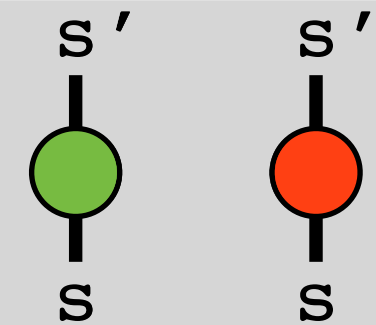
What does "prime" mean?

`prime(s)` returns copy of `s` with a "prime level" of `1`

Could use different indices (say `s` and `t`),
but `s'` convenient - can easily remove prime later

Our operators:

```
ITensor Sz(s,prime(s)),
        Sx(s,prime(s));
```

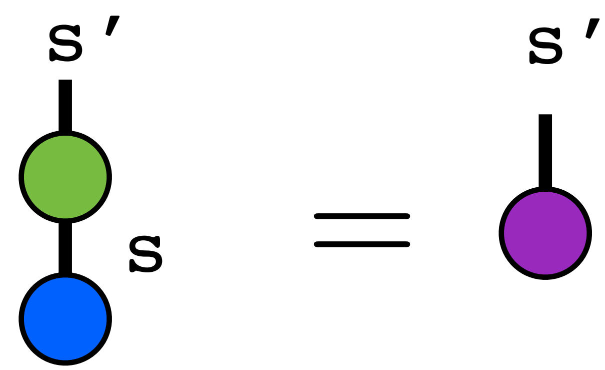


Set their components:

```
commaInit(Sz,s,prime(s)) =  0.5, 0.0,
                            0.0,-0.5;

commaInit(Sx,s,prime(s)) =  0.0, 0.5,
                            0.5, 0.0;
```

Let's multiply $\hat{S}_x|\psi\rangle$

$$(\hat{S}_x)_{s'}{}^{s}\;\psi_s =$$



In code,

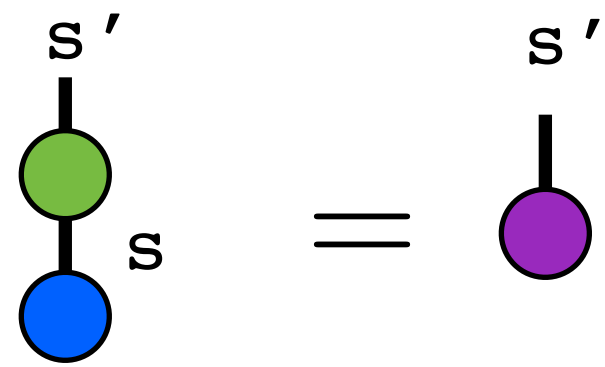
```
ITensor phi = Sx * psi;
```

Easy!

`*` operator contracts matching indices.

Indices `s` and `s'` don't match because of different prime levels.

What state is `phi` ?

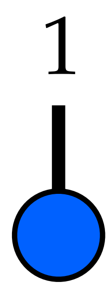
$$(\hat{S}_x)_{s'}{}^{s}\ \psi_s =$$ 


```
ITensor phi = Sx * psi;
PrintData(phi);
```

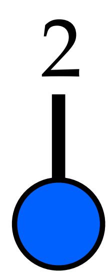
Prints:

```
phi =
ITensor r = 1: s'/Link'-######:2
  (2) 0.50000
```

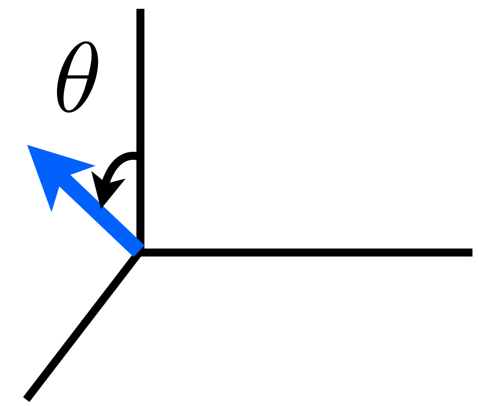
More interesting ψ_s: choose $\theta = \pi/4$ and



$= \cos\theta/2$

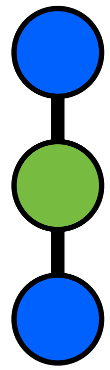


$= \sin\theta/2$




```
Real theta = Pi/4;                  // Prints:
                                    // psi =
psi(s(1)) = cos(theta/2);           // ITensor r = 1:
psi(s(2)) = sin(theta/2);           //     s/Link-1185:2
                                    //  (1) 0.9238795325
PrintData(psi);                     //  (2) 0.3826834324
```

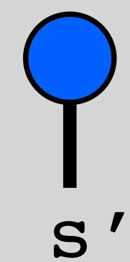
Diagrammatically, measurements (expectation values) look like:



$$\langle\psi|\hat{S}_z|\psi\rangle$$

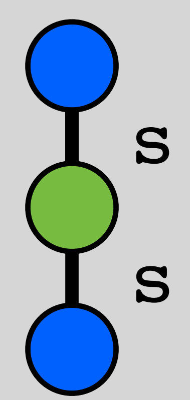
For convenience, make:

```
ITensor cpsi = dag(prime(psi));
```



Calculate expectation values:

```
Real zz = (cpsi * Sz * psi).toReal();
Real xx = (cpsi * Sx * psi).toReal();
```



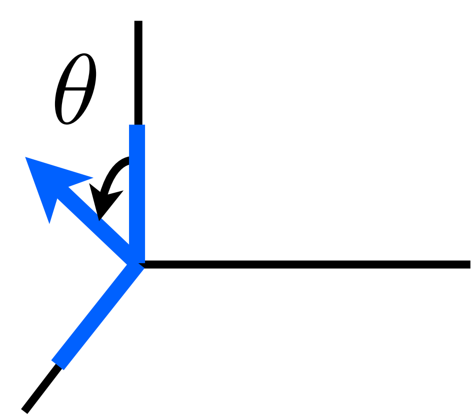
```
Real zz = (cpsi * Sz * psi).toReal();
Real xx = (cpsi * Sx * psi).toReal();
```

Printing the results,

```
println(“<Sz> = “,zz);
println(“<Sx> = “,xx);
```

we get the output

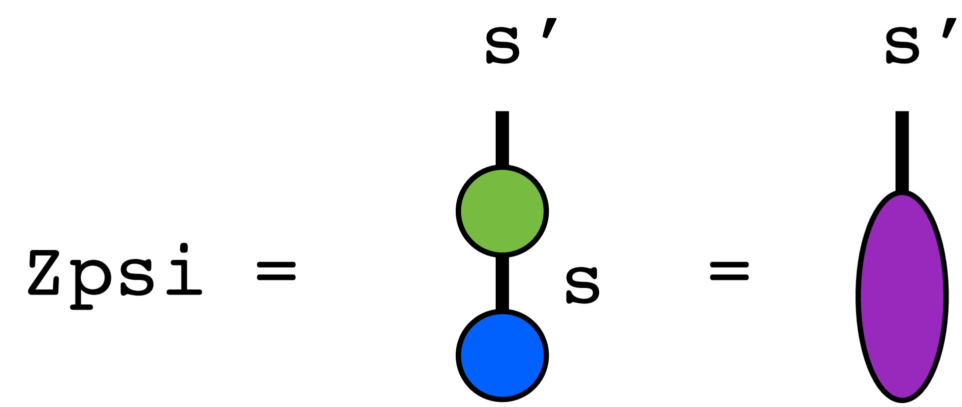
```
<Sz> = 0.35355
<Sx> = 0.35355
```



$$\sqrt{(0.35355)^2 + (0.35355)^2} = 1/2$$

Take a closer look at the tensor contractions:

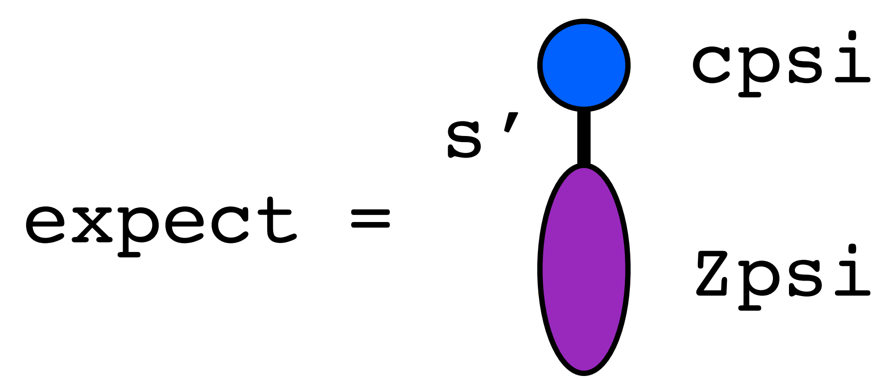
```
ITensor Zpsi = Sz * psi;
```



Index `s` matches, so it's automatically contracted.

`Zpsi` and `cpsi` share Index `s'`
`*` contracts it, leaving a scalar ITensor

```
ITensor expect = cpsi * Zpsi;
Real zz = expect.toReal();
```



Review:

- Construct an Index using `Index a("a",4);`

- Construct ITensor using indices a, b, c

```
ITensor T(a,b,c);
```

- Set ITensor components using

```
T(a(2),b(1),c(3)) = 5;
```

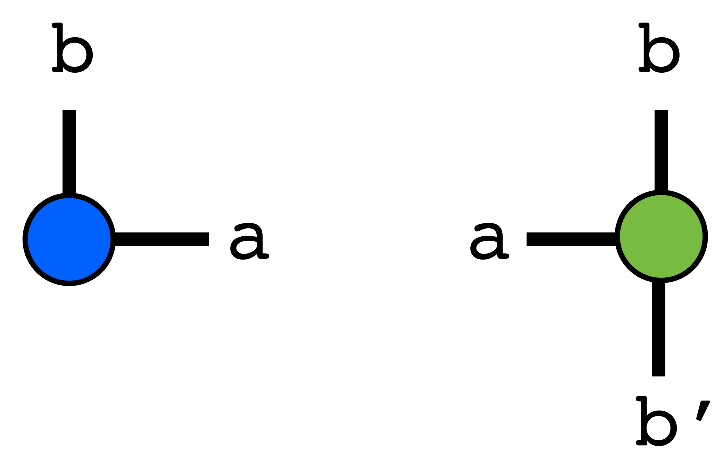
- We can prime an Index b $\longrightarrow$ b' using

```
prime(b)
```

- The `*` operator automatically contracts matching Index pairs

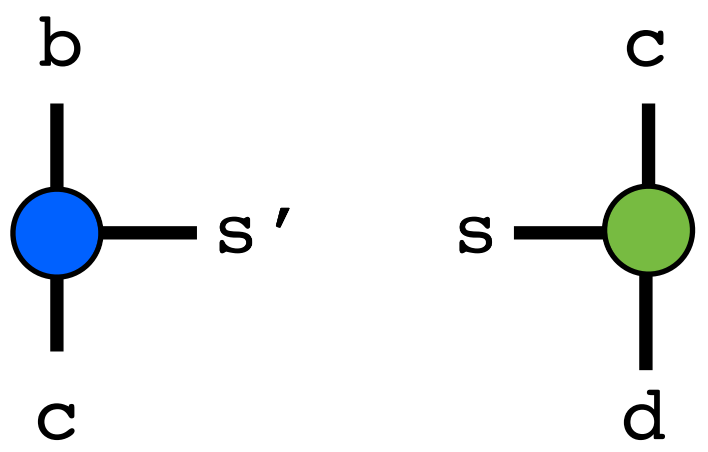
Quiz:

If we * the following tensors,
how many indices remain?



Quiz:

If we * the following tensors,
how many indices remain?



Code hands-on session:

```
<library folder>/tutorial/01_one_site
```

1. Compile by typing "**`make`**" then run by typing "**`./one`**"

2. Change psi to be an eigenstate of S_x $$|\psi\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$

3. Compute overlap of $|\psi\rangle$ with $|\phi\rangle = \hat{S}_x|\psi\rangle$:

```
Real olap = (dag(phi)*psi).toReal();
```

Try also normalizing $|\phi\rangle$ first using the code

```
phi *= 1/phi.norm();
```

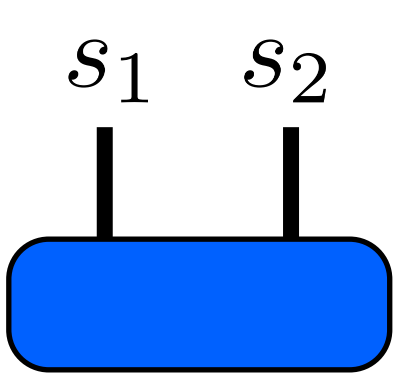
02 Two Sites

Most general two-spin wavefunction is

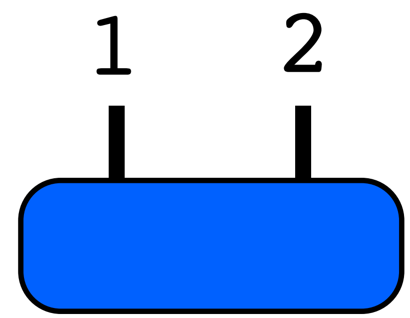
$$|\Psi\rangle = \sum_{s_1, s_2 = 1}^{2} \psi_{s_1 s_2} |s_1\rangle |s_2\rangle$$

Amplitudes a rank-2 tensor

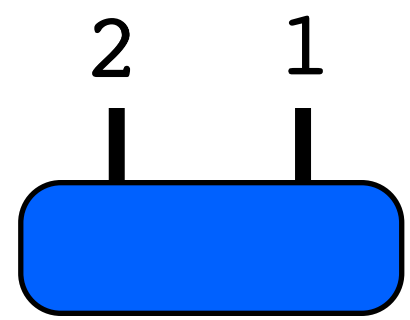
$$\psi_{s_1 s_2} =$$



Let's make a singlet



$$= 1/\sqrt{2}$$



$$= -1/\sqrt{2}$$

Using ITensor:

```
Index s1("s1",2,Site), s2("s2",2,Site);
ITensor psi(s1,s2); //default initialized to zero
psi(s1(1),s2(2)) =  1./sqrt(2);
psi(s1(2),s2(1)) = -1./sqrt(2);
```

Why **`Site`** tag in Index constructor?

```
Index s1("s1",2,Site),
      s2("s2",2,Site);
```

Two Index types: **`Link`** (default) and **`Site`**.

Useful for priming just one type of Index, for example.

Let's make the Heisenberg Hamiltonian $\hat{H} = \mathbf{S}_1 \cdot \mathbf{S}_2$

$$\hat{H} = S_1^z S_2^z + \frac{1}{2} S_1^+ S_2^- + \frac{1}{2} S_1^- S_2^+$$

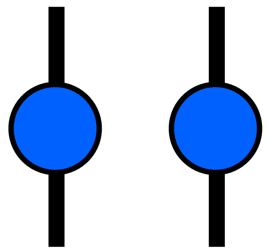
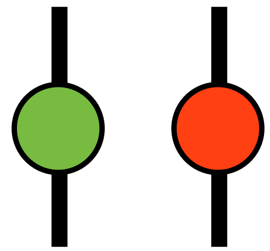
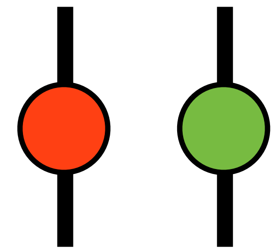
First create operators, for example S^+

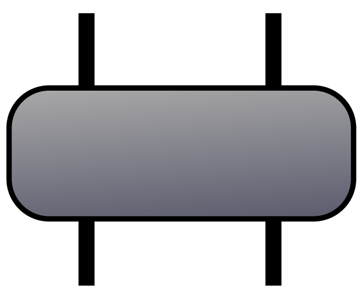
```
ITensor Sp1(s1,prime(s1));
commaInit(Sp1,s1,prime(s1)) = 0, 1,
                              0, 0;
```

Multiply and add operators to make H:

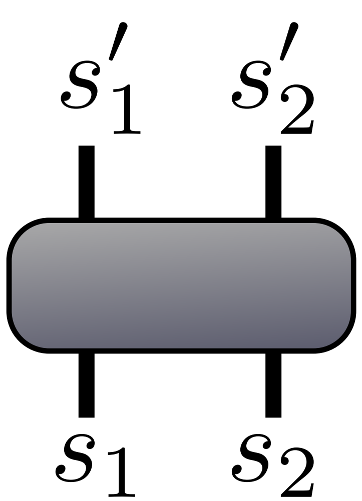
```
ITensor H = Sz1*Sz2 + 0.5*Sp1*Sm2 + 0.5*Sm1*Sp2;
```

Tensor form of H

$\hat{H} =$  $+\frac{1}{2}$  $+\frac{1}{2}$ 

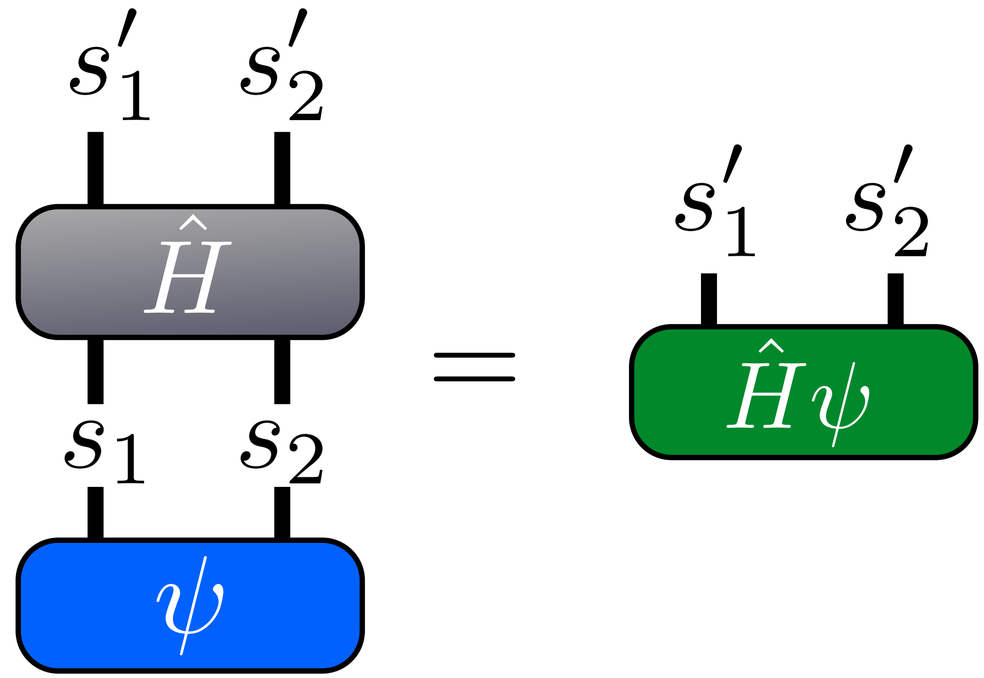
$=$ 

Showing Index labels

$\hat{H} =$ 

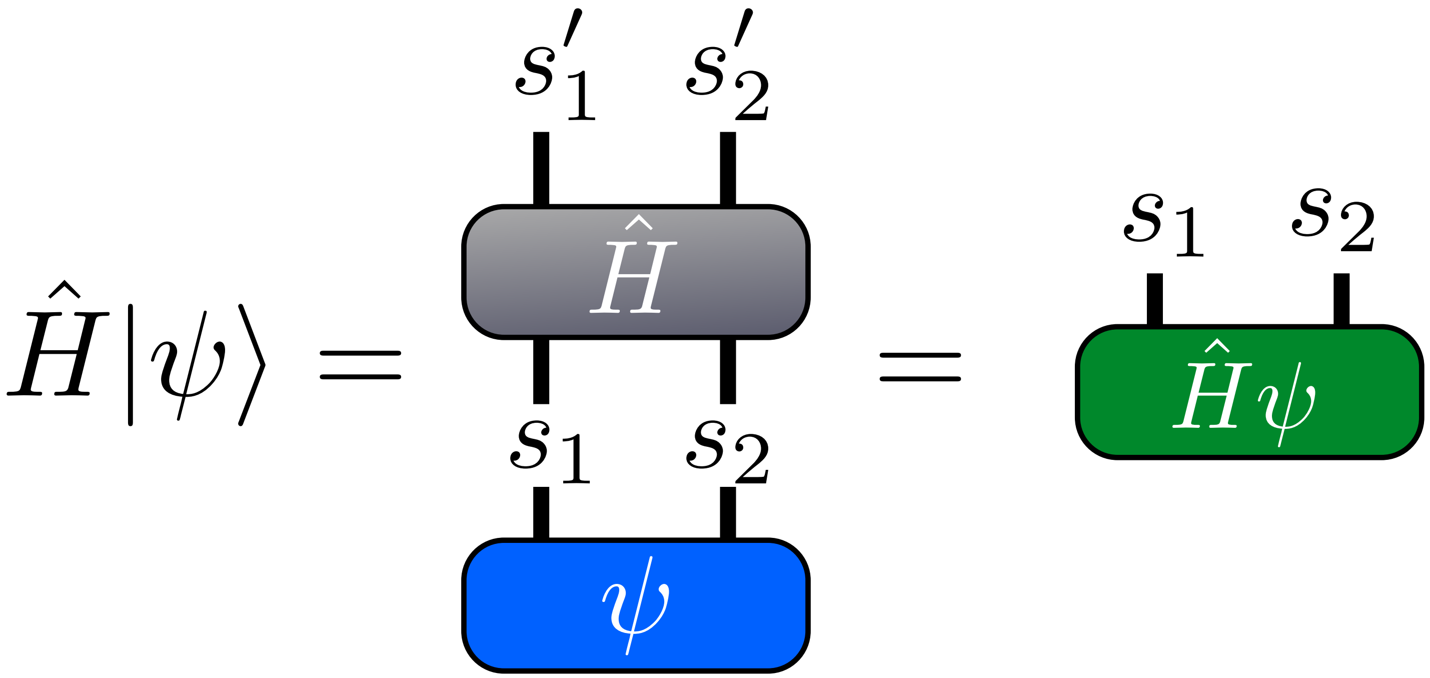
Compute singlet energy with this Hamiltonian:

$\hat{H}|\psi\rangle =$



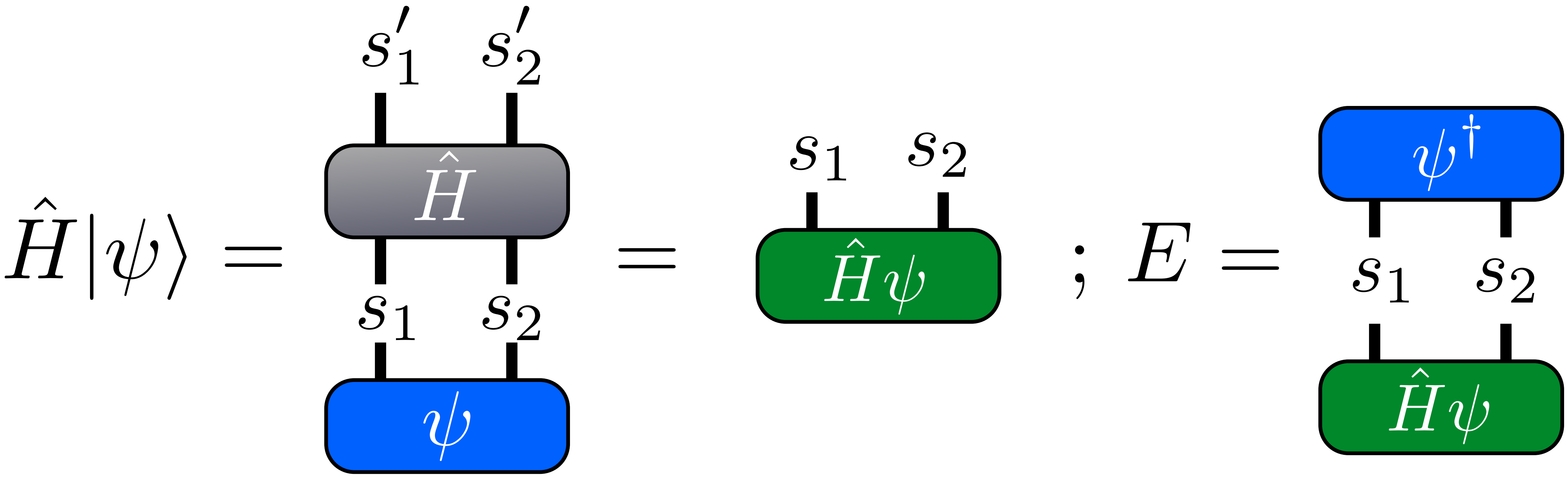

```
ITensor Hpsi = H * psi;
```

Compute singlet energy with this Hamiltonian:



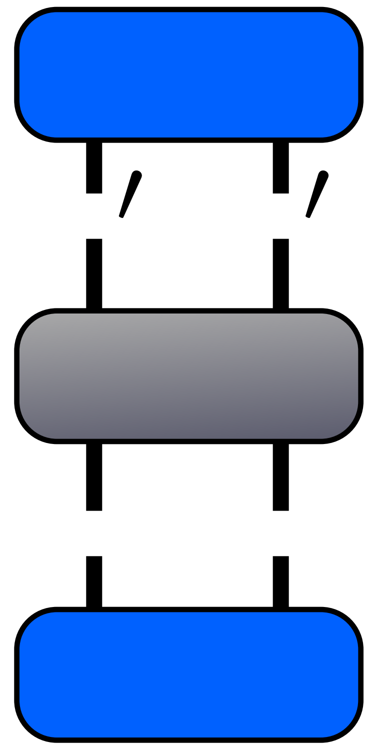

```
ITensor Hpsi = H * psi;

Hpsi.mapprime(1,0);
```

Compute singlet energy with this Hamiltonian:




```
ITensor Hpsi = H * psi;

Hpsi.mapprime(1,0);

Real E = (dag(psi) * Hpsi).toReal();
Print(E);
//Prints:
//E = -0.75
```

Or compute energy in one shot:

$E_{\mathrm{sing}} =$ 


```
Real E = (dag(prime(psi)) * H * psi).toReal();
Print(E);
//Prints:
//E = -0.75
```

We’ll use imaginary time evolution to find this Hamiltonian’s ground state

$$e^{-\beta H/2}|0\rangle \propto |\Psi_0\rangle$$

```
<library folder>/tutorial/02_two_sites
```

1. Read through **`two.cc`**, compile and run by typing “**`make two`**” then run by typing “**`./two`**”

2. Open **`imag_tevol.cc`** and implement the code to make $e^{-\beta H}$ using a Taylor series (summed using a recursive formula)

3. Try increasing β, compile, and re-run the code until it converges to the ground state

Solution for missing code (near line 120 of `imag_tevol.cc`):

```
for(int ord = max_order-1; ord >= 1; --ord)
    {
    expH = expH * (x/ord);
    expH.mapprime(2,1);
    expH += I;
    }
```

03 SVD

The density matrix renormalization group (DMRG) works with a variational wavefunction known as a **matrix product state** (MPS).

Matrix product states arise from compressing one-dimensional wavefunctions through the **singular-value decomposition** (SVD).

Let's see how this works...

Recall:
Singular-value decomposition

Given rectangular (4x3) matrix M

$$M = \begin{bmatrix} 0.435839 & 0.223707 & 0.10 \\ 0.435839 & 0.223707 & -0.10 \\ 0.223707 & 0.435839 & 0.10 \\ 0.223707 & 0.435839 & -0.10 \end{bmatrix}$$

Can decompose as

$$\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & 0.200 \end{bmatrix} \begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & 0.200 \end{bmatrix} \begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A D B

Matrices A and B one-sided unitaries (isometries):

$$A^\dagger A = \mathbf{1}$$

$$BB^\dagger = \mathbf{1}$$

D diagonal

Elements of D can be chosen:

(I) Real

(II) Positive semi-definite

(III) Decreasing order

Keep fewer and fewer elements of D:

$$\underset{A}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}} \underset{D}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & 0.200 \end{bmatrix}} \underset{B}{\begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$= M = \begin{bmatrix} 0.435839 & 0.223707 & 0.10 \\ 0.435839 & 0.223707 & -0.10 \\ 0.223707 & 0.435839 & 0.10 \\ 0.223707 & 0.435839 & -0.10 \end{bmatrix}$$

$$||M - M||^2 = 0$$

Keep fewer and fewer elements of D:

$$\overset{A}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}} \overset{D}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0.300 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \overset{B}{\begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$= M_2 = \begin{bmatrix} 0.435839 & 0.223707 & 0 \\ 0.435839 & 0.223707 & 0 \\ 0.223707 & 0.435839 & 0 \\ 0.223707 & 0.435839 & 0 \end{bmatrix}$$

$$||M_2 - M||^2 = 0.04 = (0.2)^2$$

Keep fewer and fewer elements of D:

$$\underset{A}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}} \underset{D}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \underset{B}{\begin{bmatrix} 0.707107 & 0.707107 & 0 \\ -0.707107 & 0.707107 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$= M_3 = \begin{bmatrix} 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \\ 0.329773 & 0.329773 & 0 \end{bmatrix}$$

$$||M_3 - M||^2 = 0.13 = (0.3)^2 + (0.2)^2$$

Keep fewer and fewer elements of D:

$$\underset{A}{\begin{bmatrix} 1/2 & -1/2 & 1/2 \\ 1/2 & -1/2 & -1/2 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 \end{bmatrix}} \underset{D}{\begin{bmatrix} 0.933 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \underset{B}{\begin{bmatrix} 0.707107 & 0.707107 & 0 & \\ -0.707107 & 0.707107 & 0 & \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$= M_3 =$$

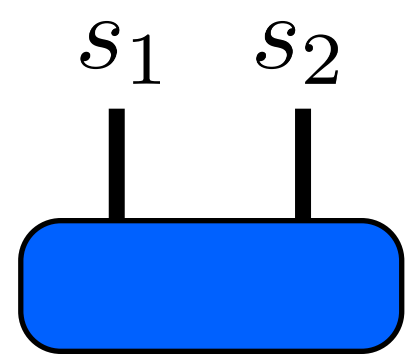
Truncating SVD =

Controlled approximation for M

$$||M_3 - M||^2 = 0.13 = (0.3)^2 + (0.2)^2$$

Recall:

Most general two-spin wavefunction

$$\psi_{s_1 s_2} =$$ 

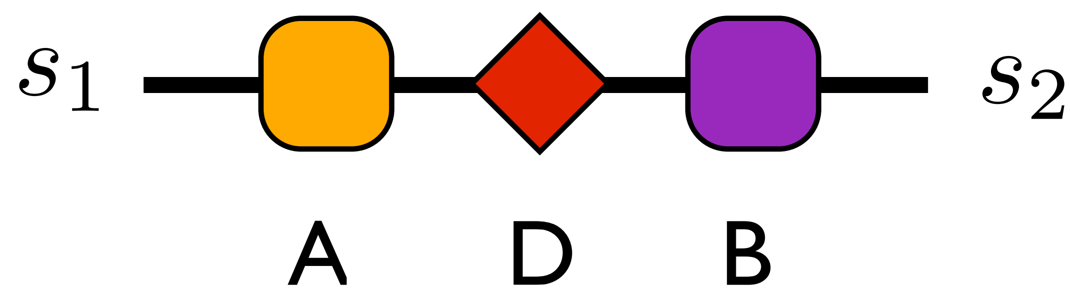
Can treat as a matrix:

$$\psi_{s_1 s_2} =$$ 

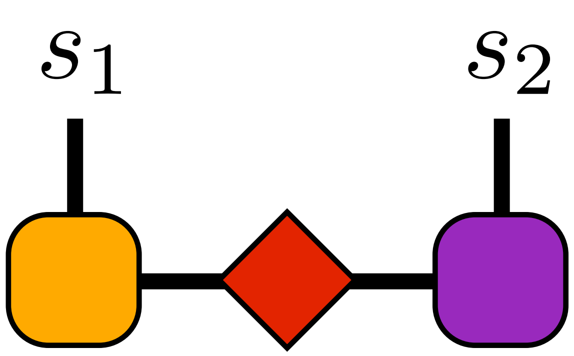
SVD this matrix:

$$\psi_{s_1 s_2} =$$

$$=$$

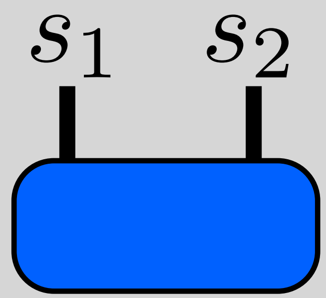


Bend lines back to look like wavefunction:

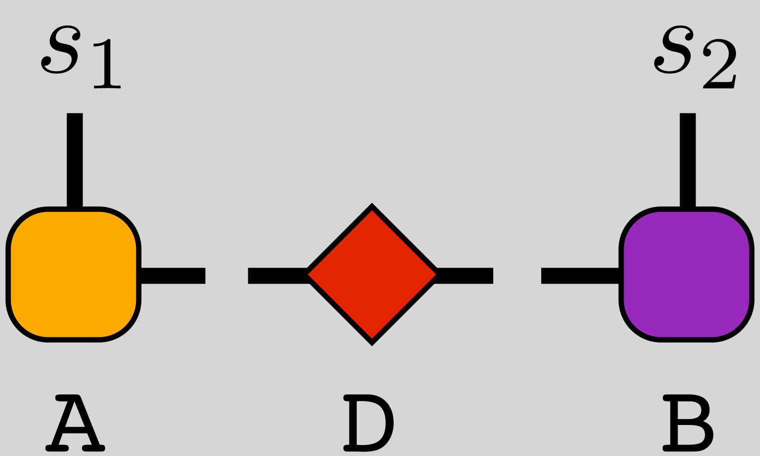


Using ITensor:

```
//Say we have a two-site ITensor psi
```

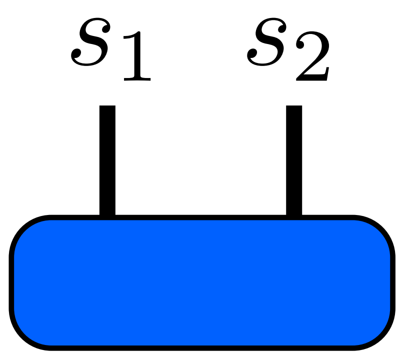



```
//Declare ITensors
//to hold results
ITensor A(s1),D,B;    //Indices of psi present
                      //on A remain, others
                      //put onto B


//Call svd method
svd(psi,A,D,B);
```



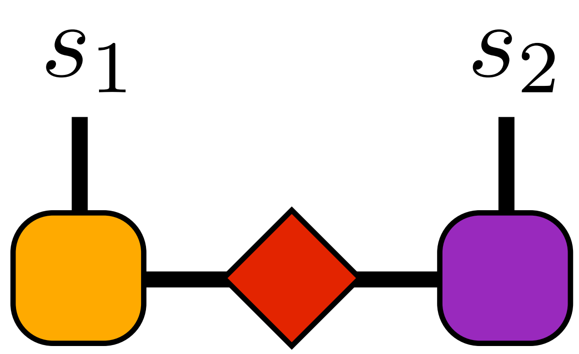
What have we gained from SVD?

Generic two-spin wavefunction (say spin S):



$(2S+1)^2$ parameters
Not clear which parameters important, unimportant

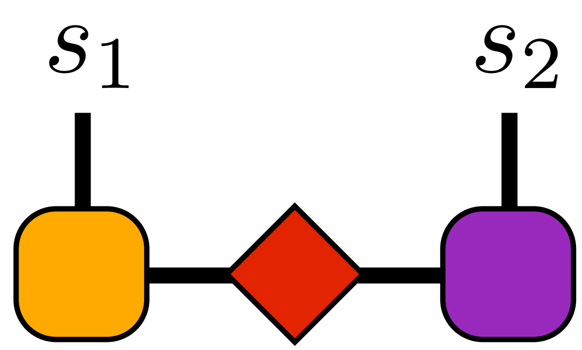
Compressed wavefunction:



SVD tells us which parameters are important, might be very few!

Later see that # parameters also scales much better

This form of wavefunction known as **matrix product state** (MPS)

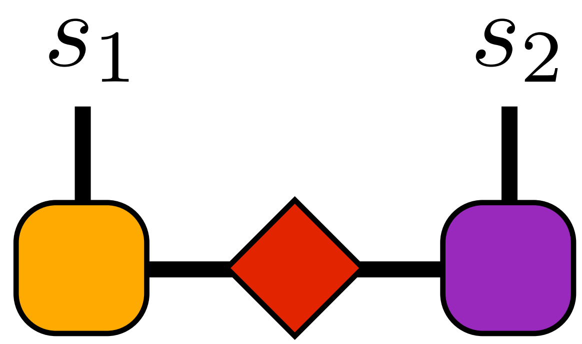


Why? Amplitude a product of matrices:

$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle |s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"

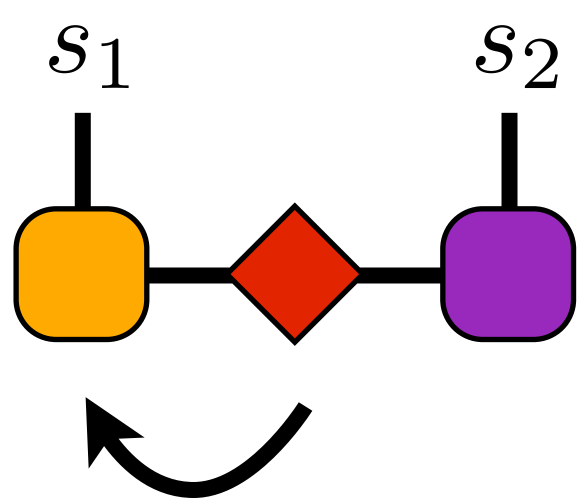


Canonical form

$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle|s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"

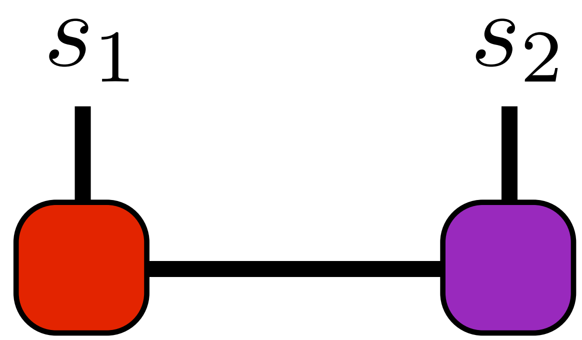


Left-canonical

$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle |s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"

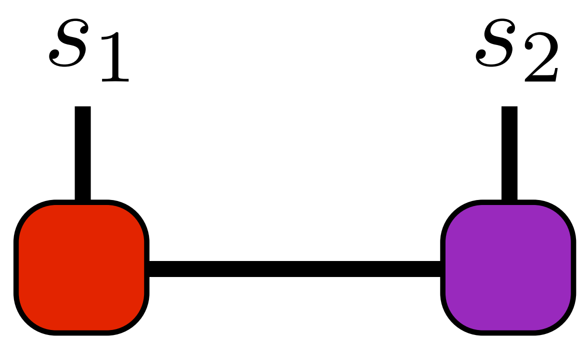


Left-canonical

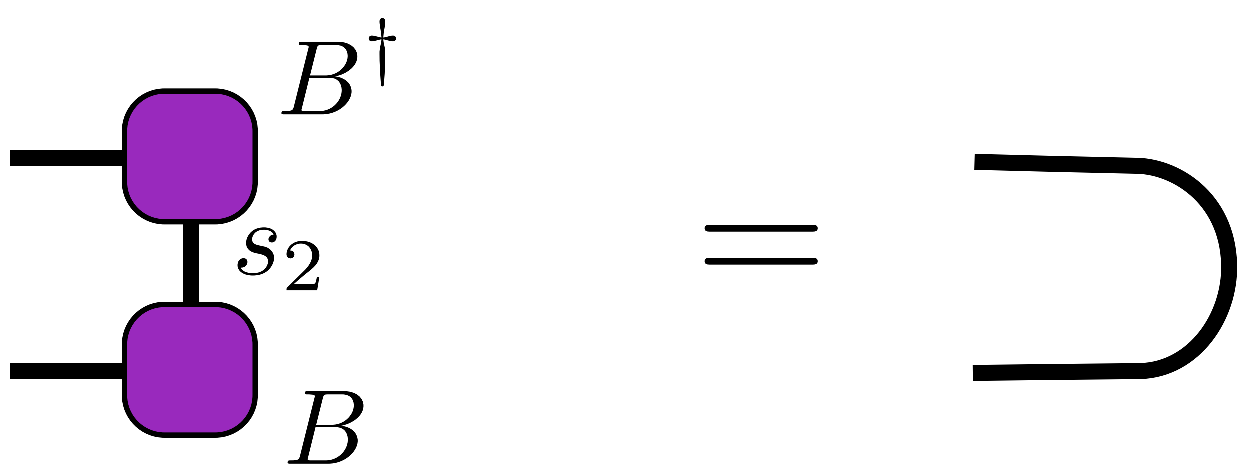
$$|\Psi\rangle = \sum_{s_1,\alpha',s_2} \psi_{s_1\alpha'} B_{\alpha' s_2} |s_1\rangle|s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"



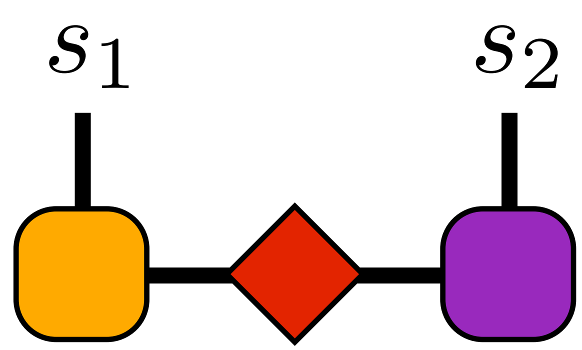
Matrix B is "right orthogonal" (from SVD)



$$BB^\dagger = I$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or “gauges”

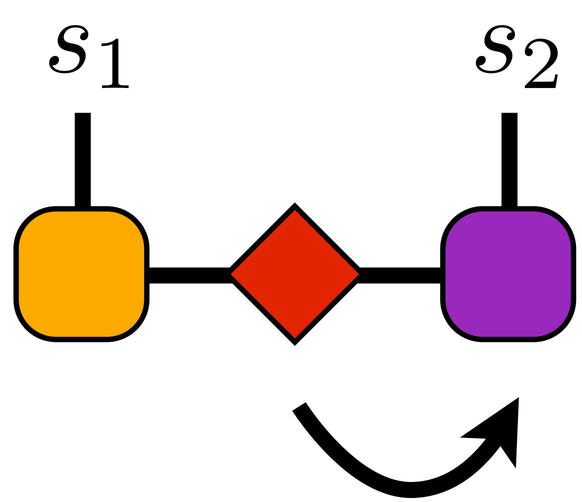


Canonical form

$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle|s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

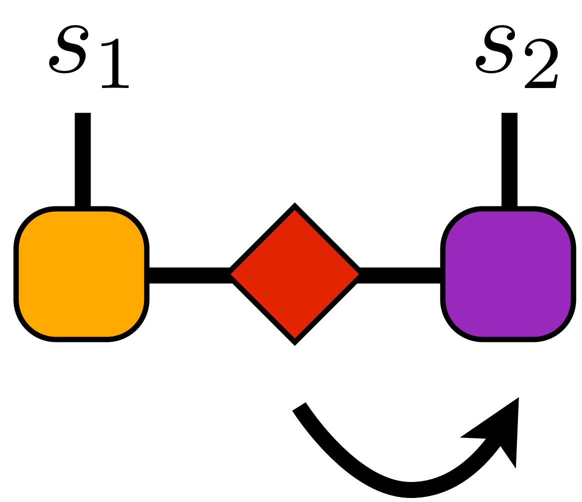
MPS have different equivalent forms, or "gauges"



$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle |s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"

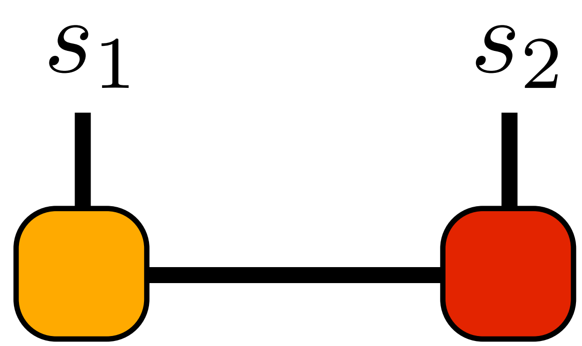


Right-canonical

$$|\Psi\rangle = \sum_{s_1,\alpha,\alpha',s_2} A_{s_1\alpha} D_{\alpha\alpha'} B_{\alpha' s_2} |s_1\rangle |s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"

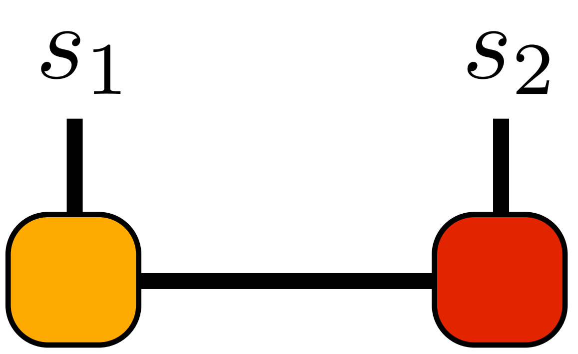


Right-canonical

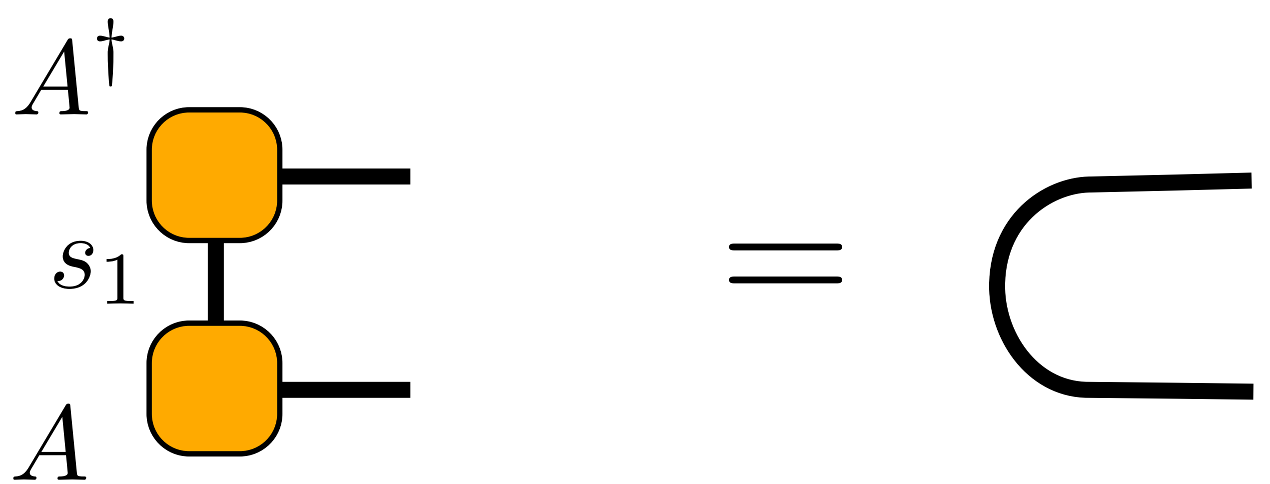
$$|\Psi\rangle = \sum_{s_1,\alpha,s_2} A_{s_1\alpha}\psi_{\alpha s_2}|s_1\rangle|s_2\rangle$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

MPS have different equivalent forms, or "gauges"



Matrix A is "left orthogonal" (from SVD)



$$A^\dagger A = I$$

Schollwöck, Ann. of Phys. **326**, 96 (2011)

We'll use the SVD to study the entanglement of a two-site wavefunction

`<library folder>/tutorial/03_svd`

1. Read through **`svd.cc`**; compile; and run

2. Make a *normalized* wavefunction that is the sum
   `(1-mix)*prod + mix*sing`

3. SVD this wavefunction

```
ITensor A(s1),D,B;
Spectrum spec = svd(psi,A,D,B);
```

3. Compute the entanglement entropy using the eigenvalue spectrum "spec" returned by svd.

   n^{th} eigenvalue:

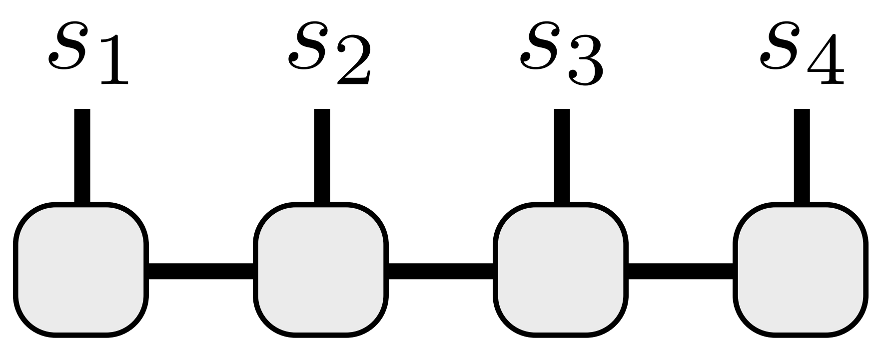
```
spec.eig(n);
```

   number of eigenvalues:

```
spec.numEigsKept();
```

04 Four

Say we have a 4-site MPS.
What can we do with it?

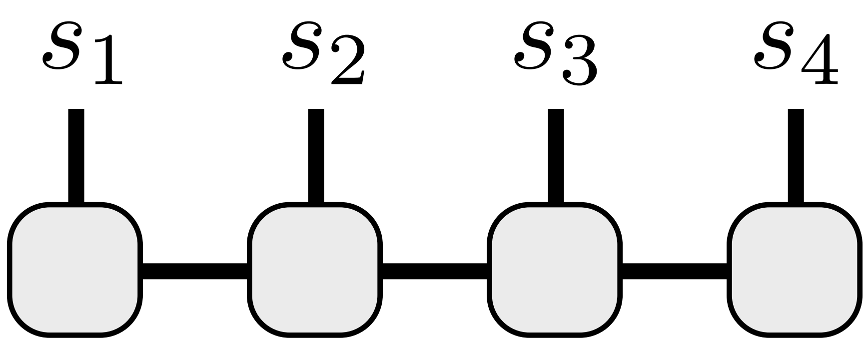


Depends on the gauge!

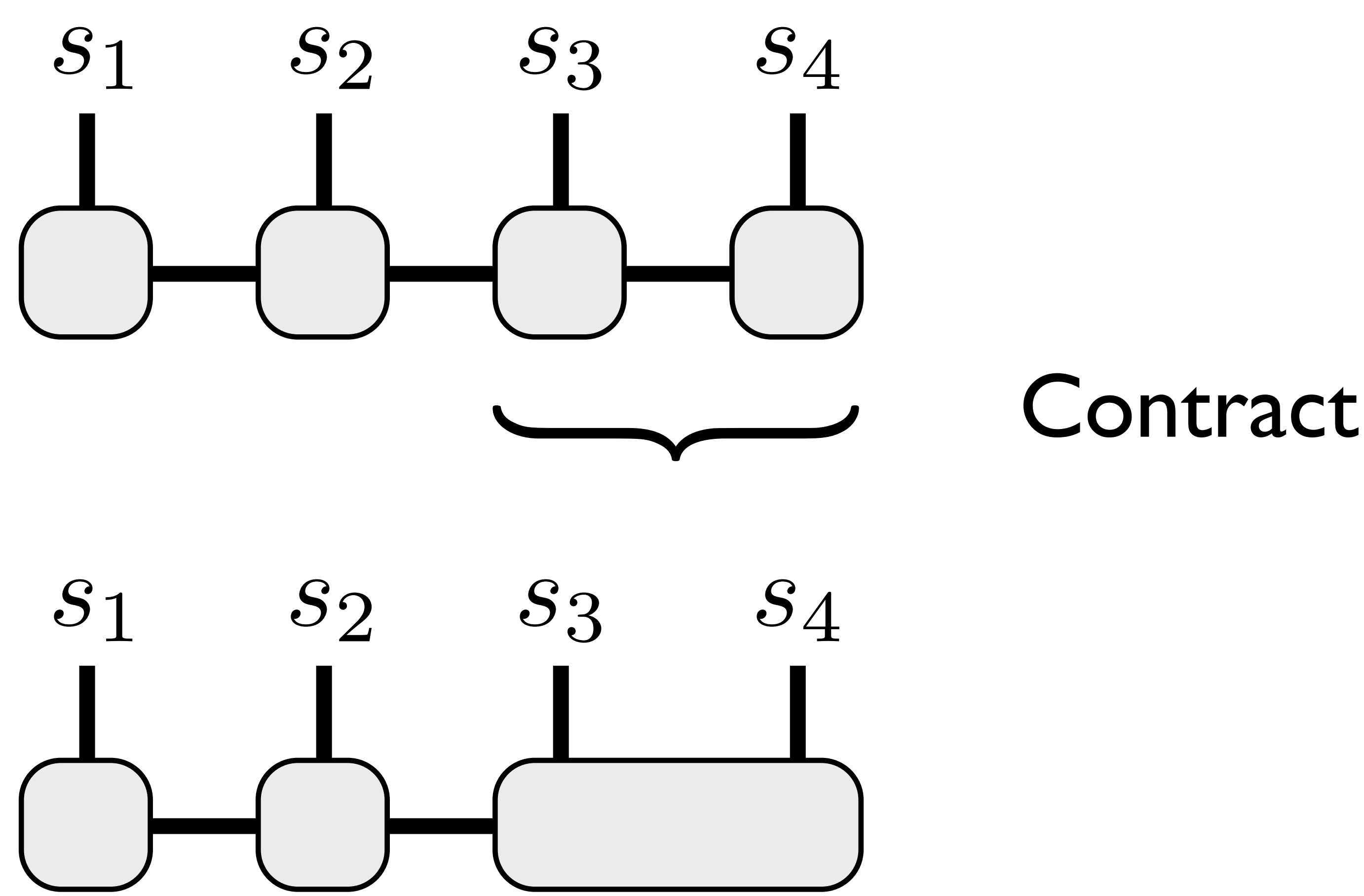
$$|\Psi\rangle = \sum_{\{s\},\{\alpha\}} M^{s_1}_{\alpha_1} M^{s_2}_{\alpha_1\alpha_2} M^{s_3}_{\alpha_2\alpha_3} M^{s_4}_{\alpha_3} |s_1 s_2 s_3 s_4\rangle$$

Assume we know nothing about the MPS

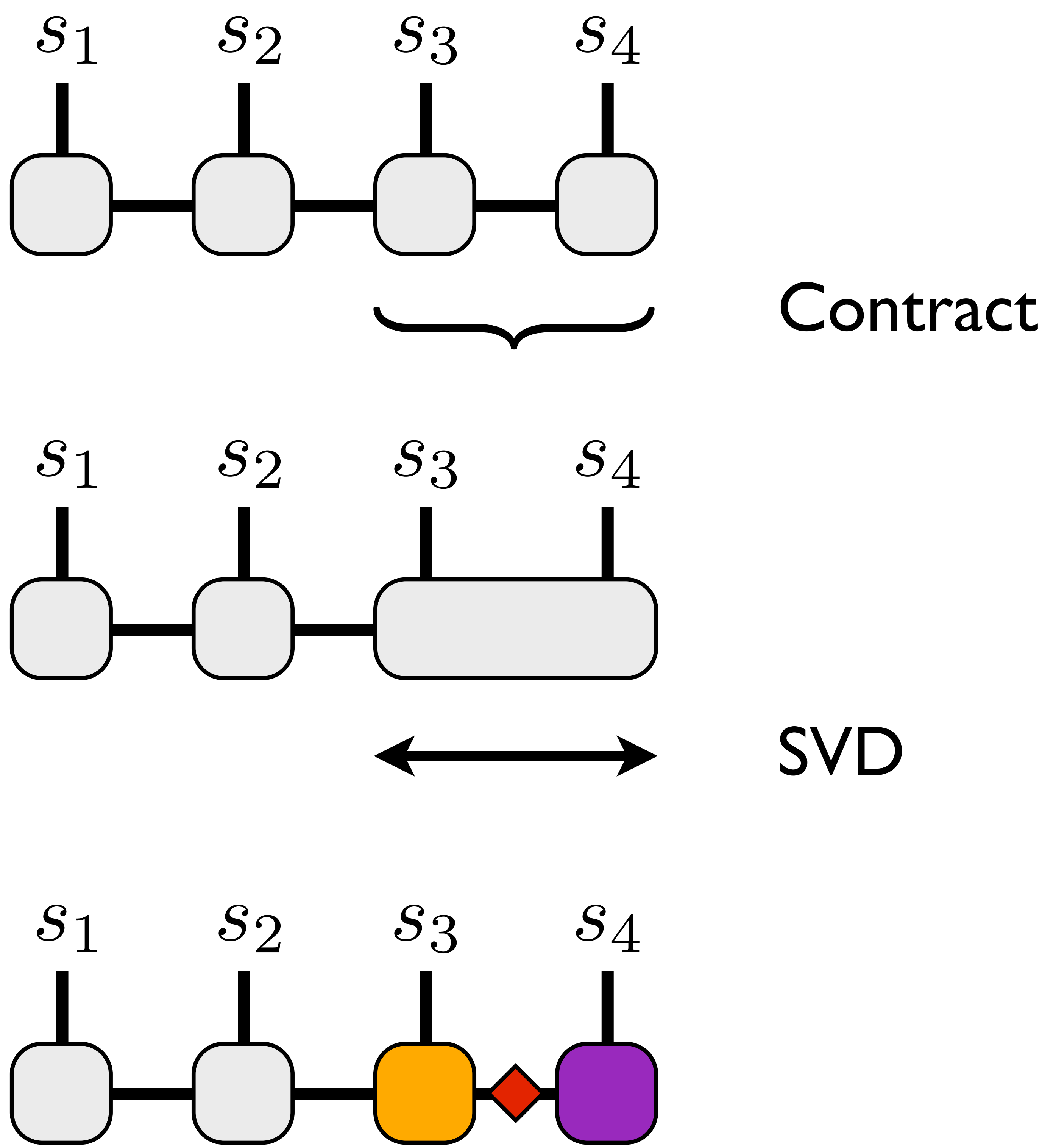
Put it in a useful gauge:



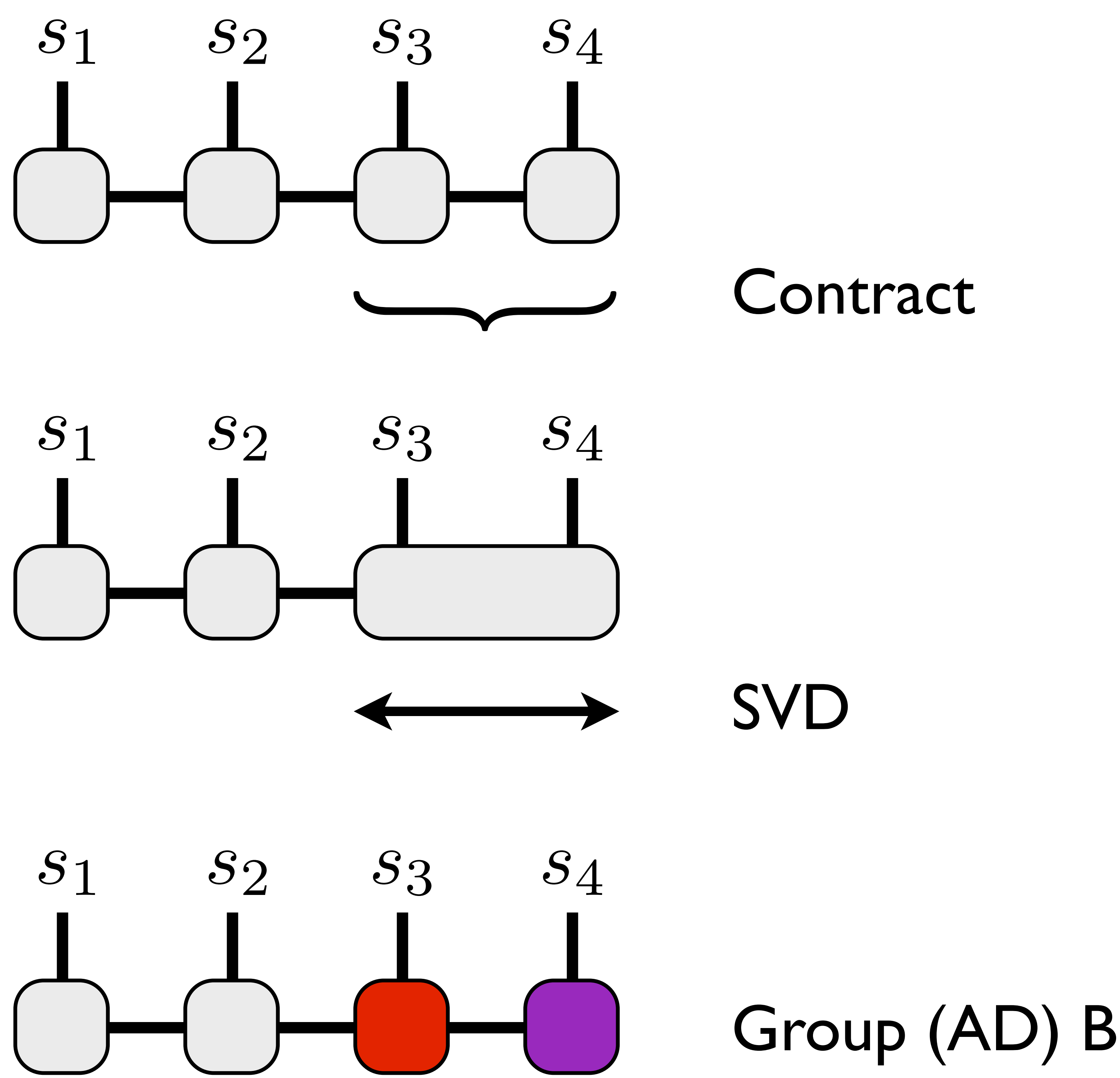
Assume we know nothing about the MPS
Put it in a useful gauge:



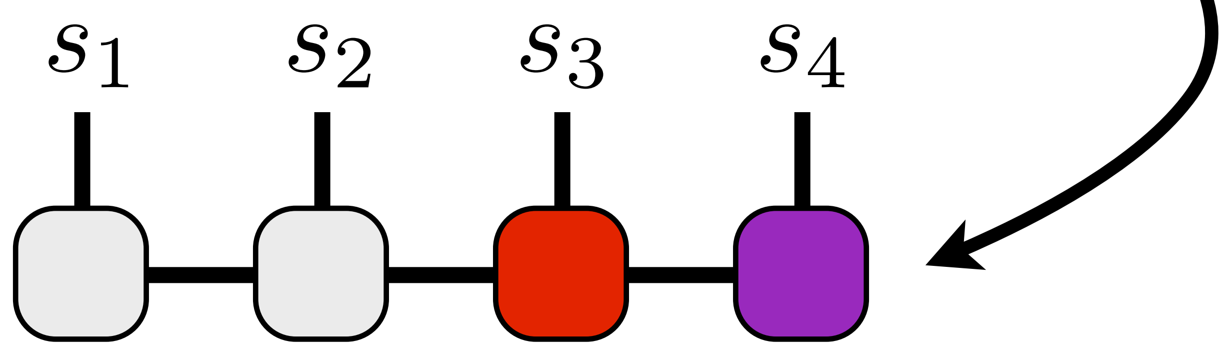
Assume we know nothing about the MPS
Put it in a useful gauge:



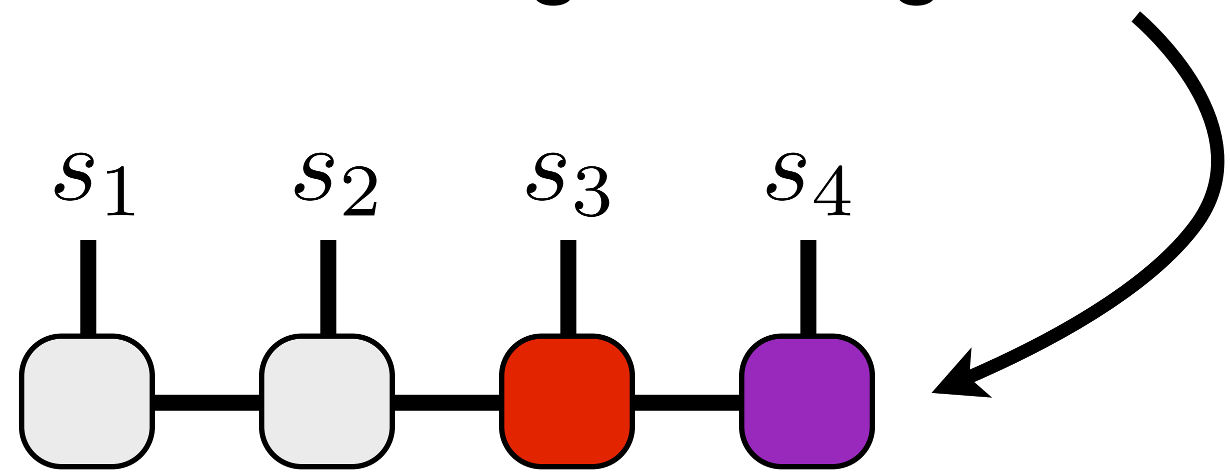
Assume we know nothing about the MPS
Put it in a useful gauge:



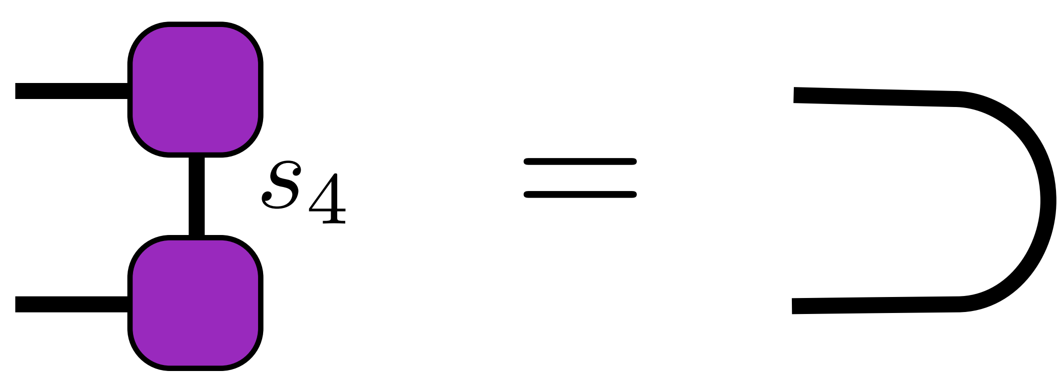
Note that site 4 tensor now right orthogonal



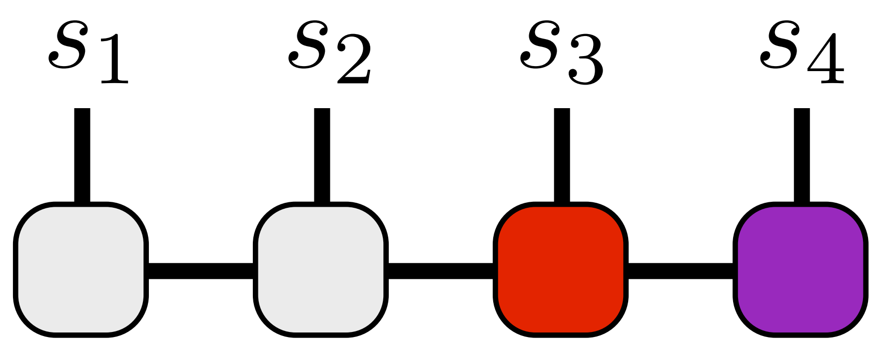
Note that site 4 tensor now right orthogonal



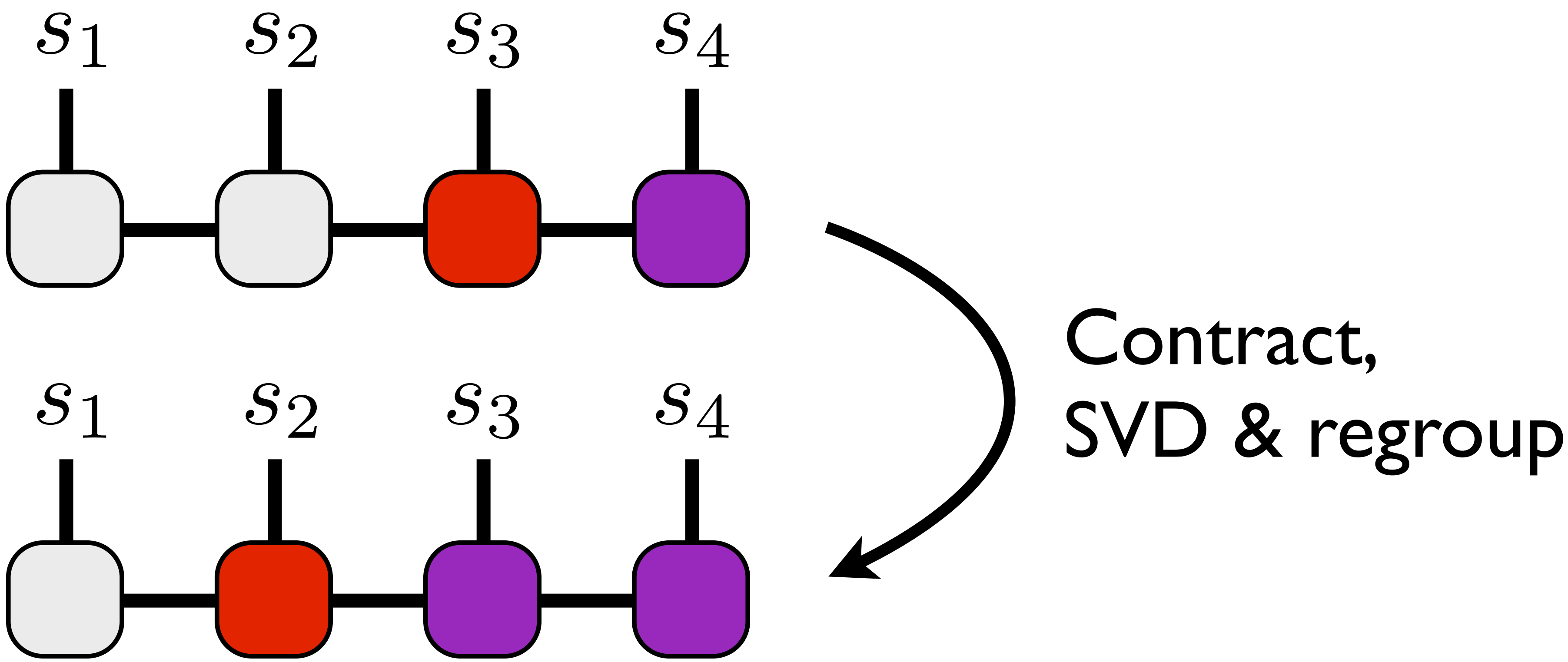
Recall this means



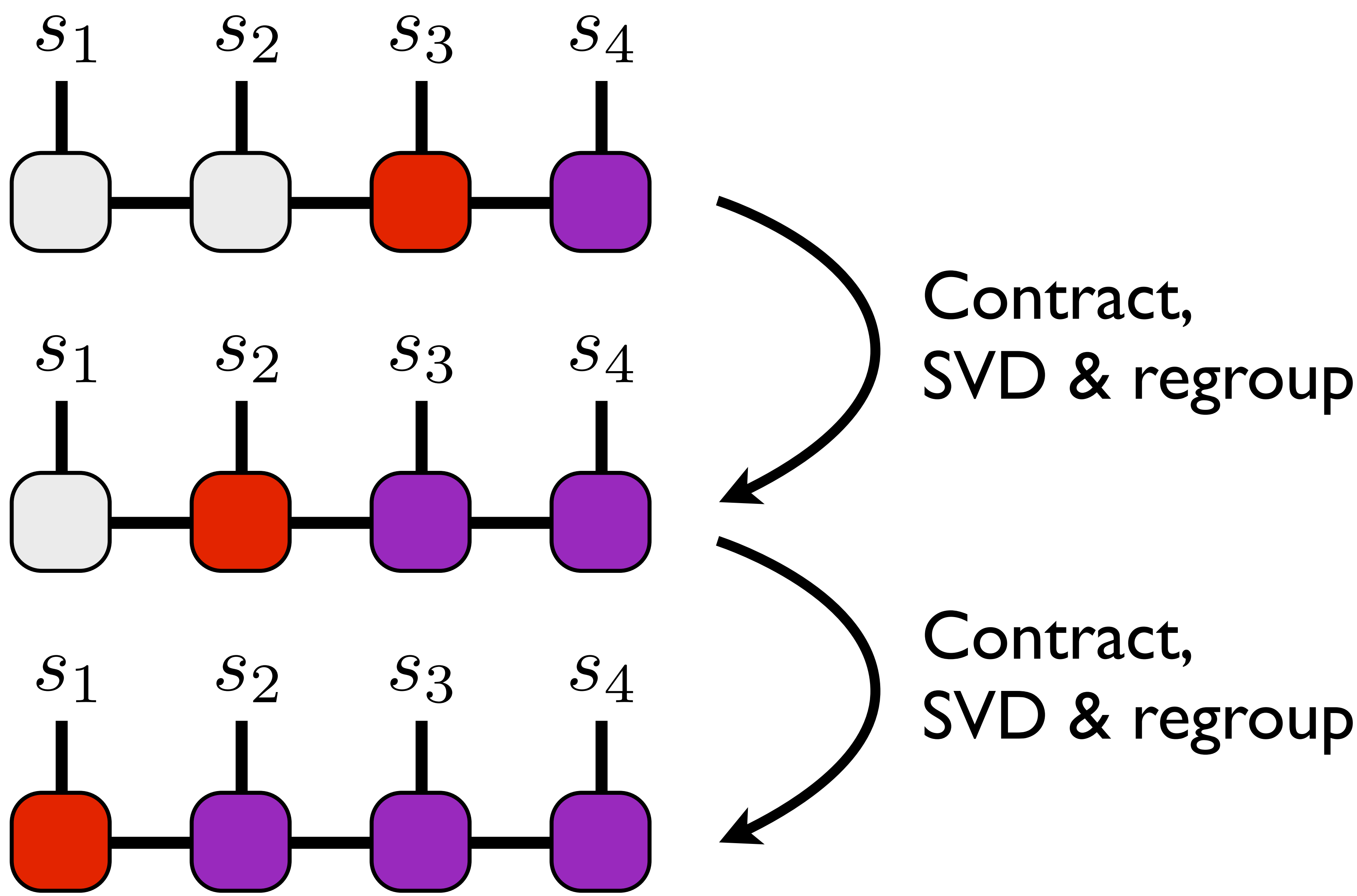
Can repeat gauge transformation (repeated SVD)



Can repeat gauge transformation (repeated SVD)

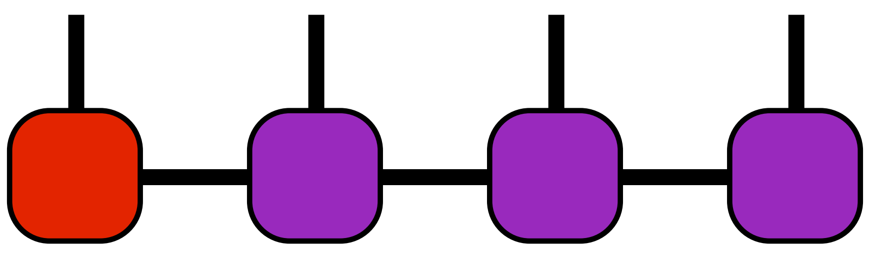


Can repeat gauge transformation (repeated SVD)



What have we gained?

Consider measuring an operator on site 1

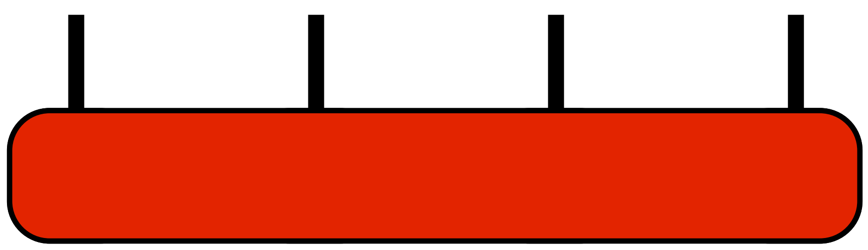


What have we gained?

Consider measuring an operator on site 1

First, general wavefunction:

$$|\Psi\rangle$$



What have we gained?

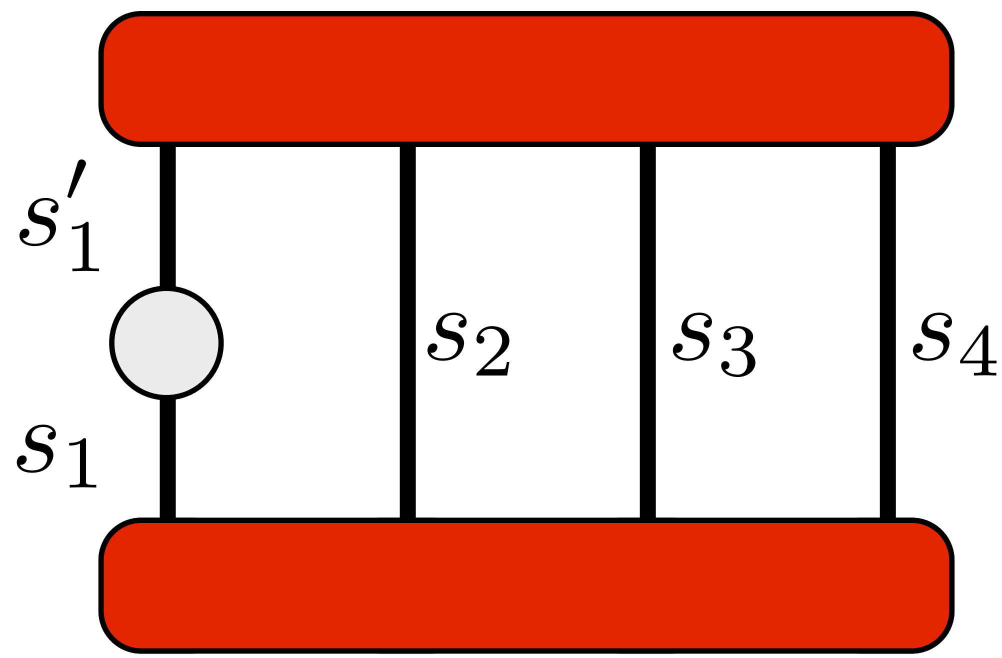
Consider measuring an operator on site 1

First, general wavefunction:

$\langle \Psi |$

$\hat{A}_1$

$| \Psi \rangle$



Cost scales exponentially!

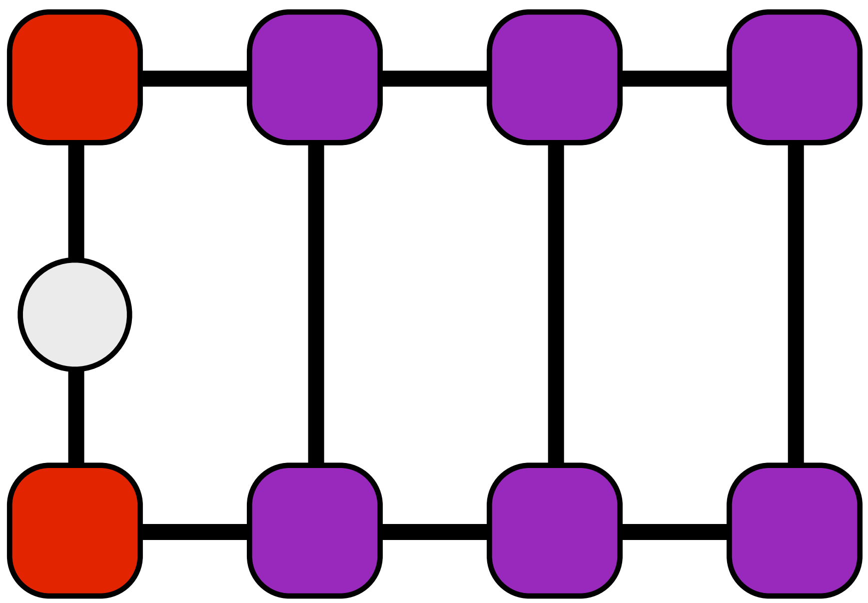
2^4 in this case

$$\langle \hat{A}_1 \rangle = \sum_{\{s\}} \bar{\psi}_{s_1' s_2 s_3 s_4} A_{s_1' s_1} \psi_{s_1 s_2 s_3 s_4}$$

What have we gained?

Consider measuring an operator on site 1

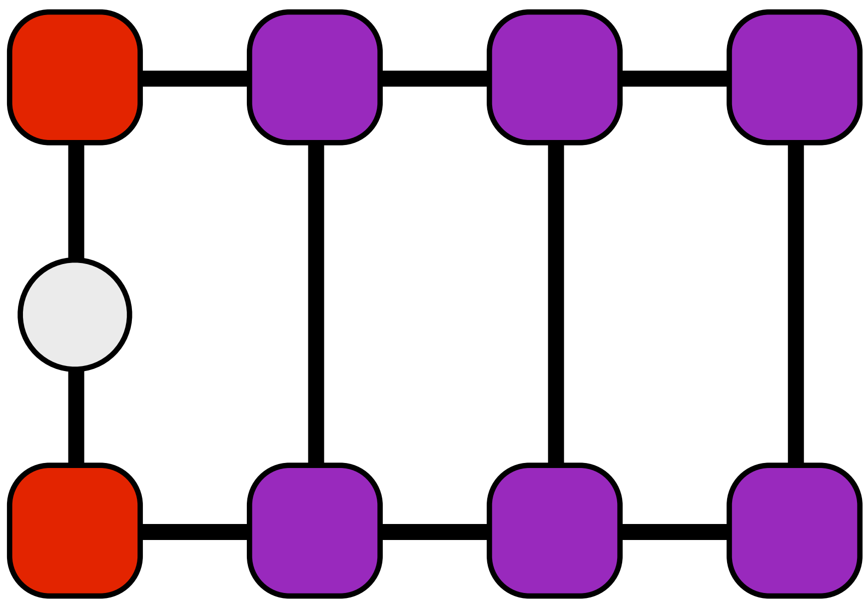
Now gauged MPS:



What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

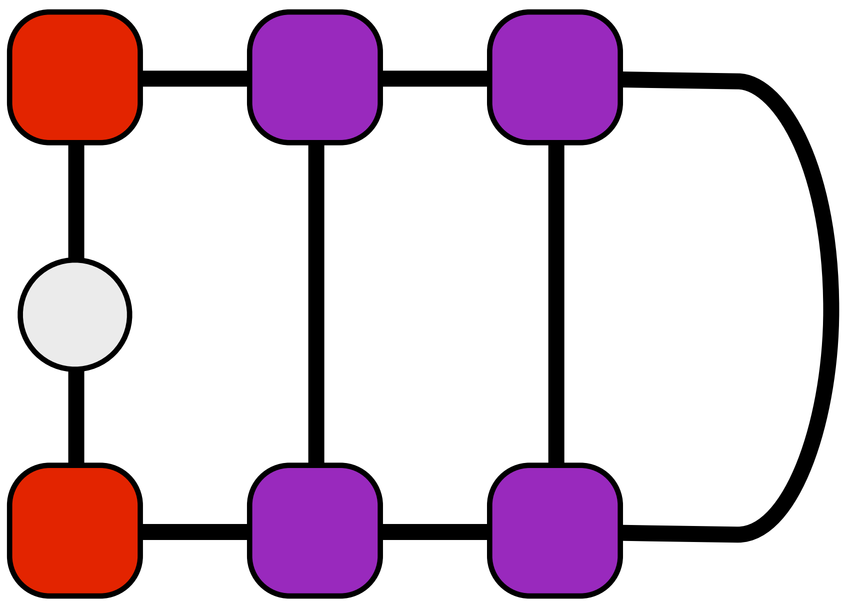


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

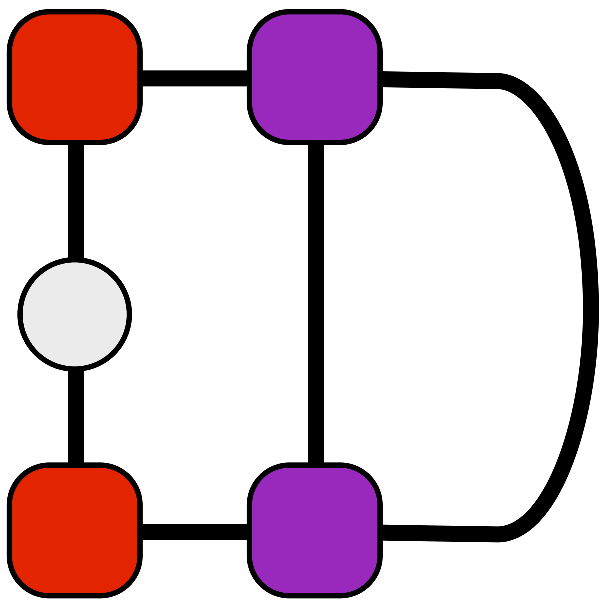


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

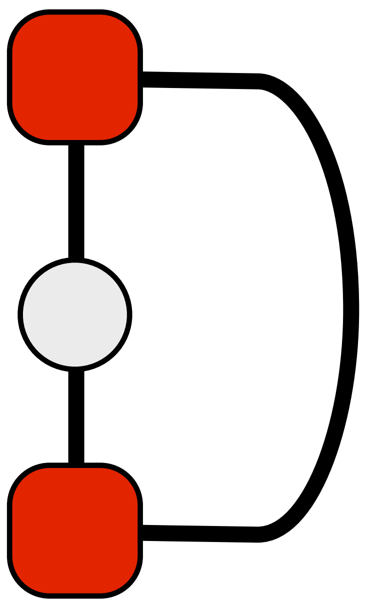


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:



Use right orthogonality

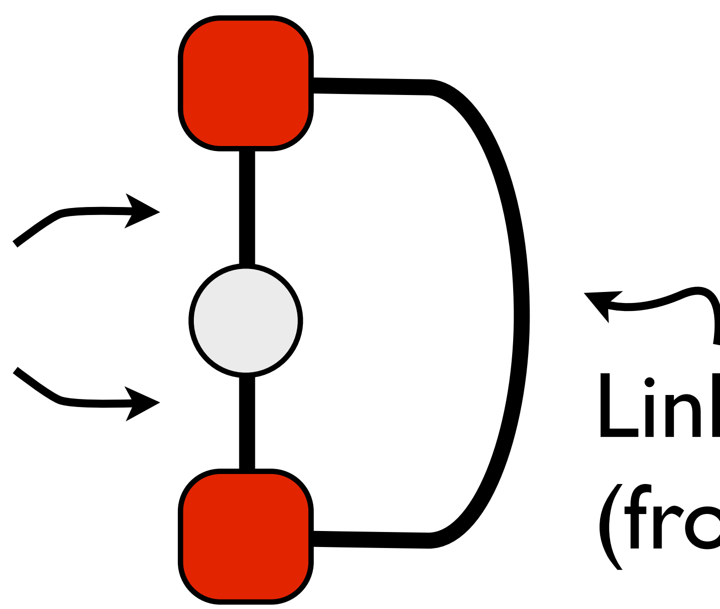
Much simpler computation!

What have we gained?

How much simpler a computation?

Choose always $\leq m$ singular values
in each SVD

Site indices
run from 1...d



Link index runs from 1...m
(from SVD)

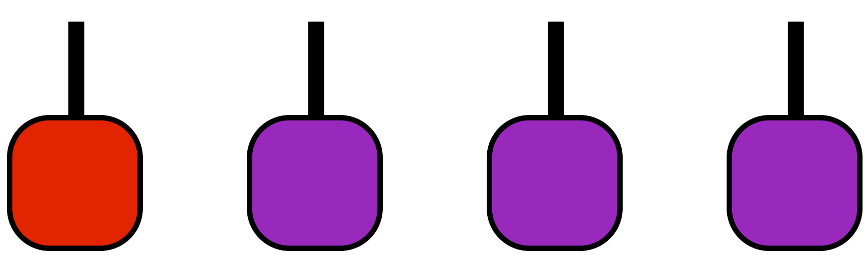
Computational cost ~ d^2 m   (compared to d^4)

Gauging an MPS Using ITensor:

```
//Define lattice sites
SpinHalf sites(N);
```

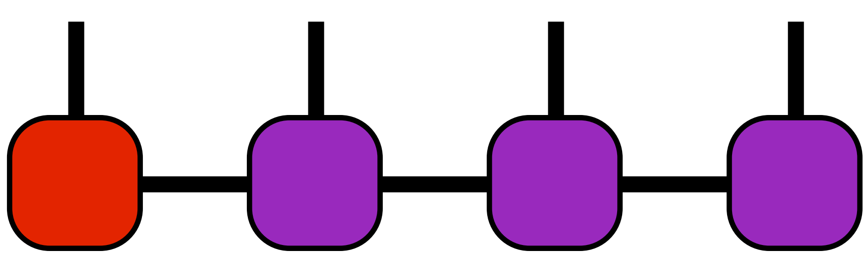
Gauging an MPS Using ITensor:

```
//Define lattice sites
SpinHalf sites(N);
MPS psi(sites);
```



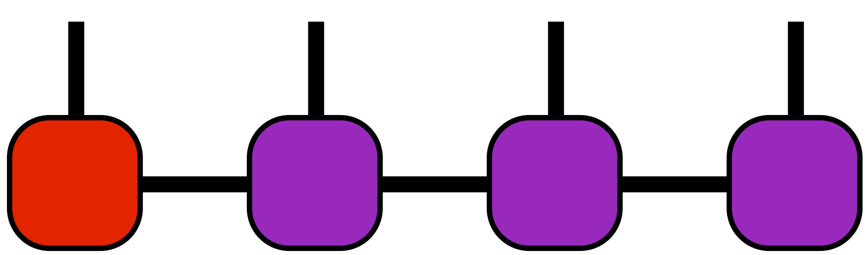
Gauging an MPS Using ITensor:

```
//Define lattice sites
SpinHalf sites(N);
MPS psi(sites);
computeGroundState(H,psi); //optimize psi
```



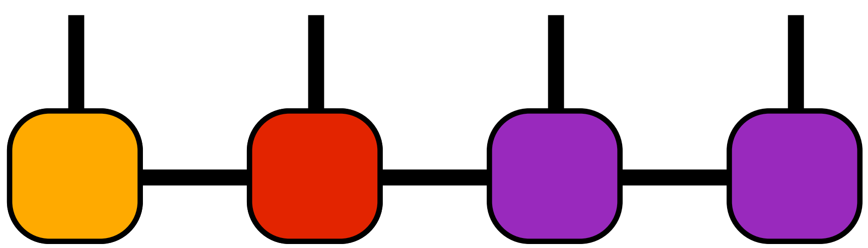
Gauging an MPS Using ITensor:

```
//Define lattice sites
SpinHalf sites(N);
MPS psi(sites);
computeGroundState(H,psi); //optimize psi

//Gauge MPS to second site
psi.position(2);
```



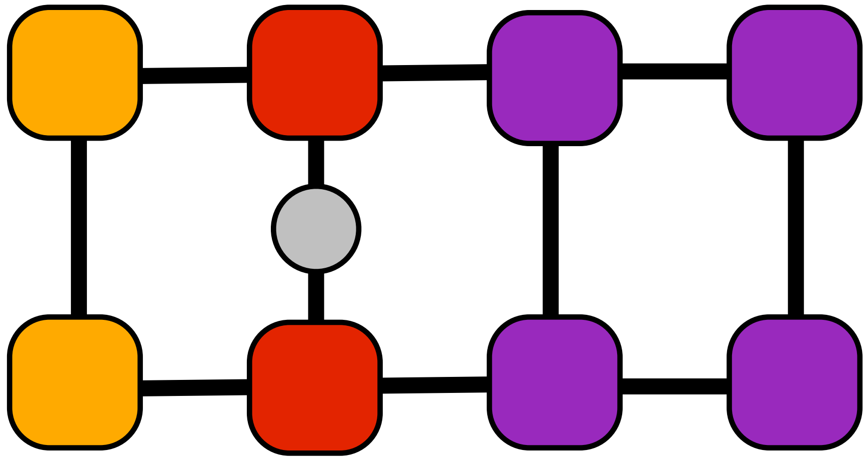
Gauging an MPS Using ITensor:

```
//Define lattice sites
SpinHalf sites(N);
MPS psi(sites);
computeGroundState(H,psi); //optimize psi


//Gauge MPS to second site
psi.position(2);
```

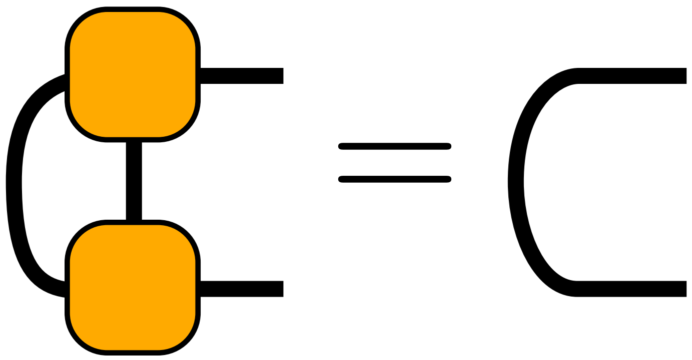


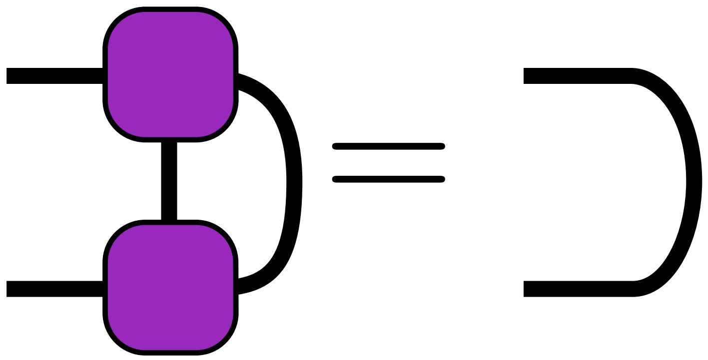
Measuring an MPS Using ITensor:

```
//Measure Sz on second site
```



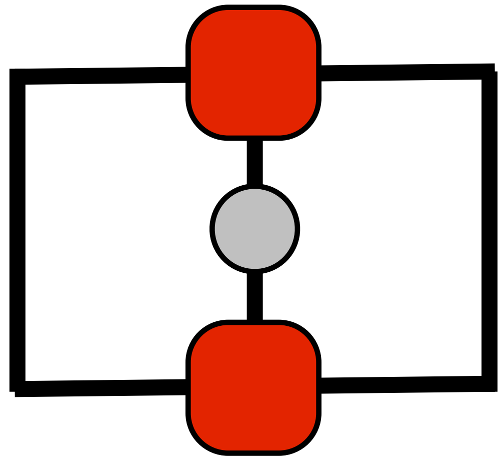
Recall:



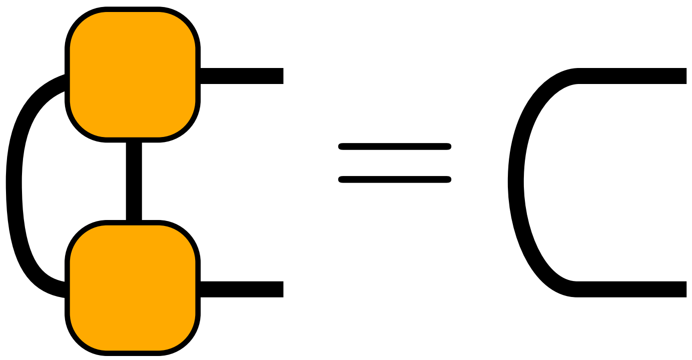


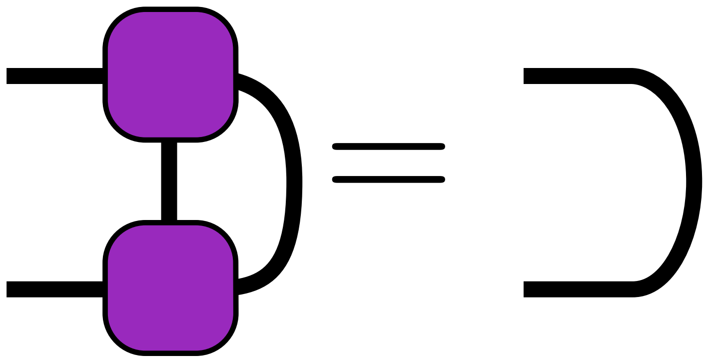
Measuring an MPS Using ITensor:

```
//Measure Sz on second site
```



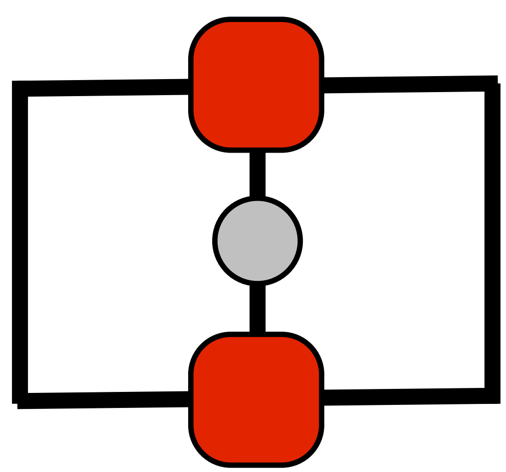
Recall:



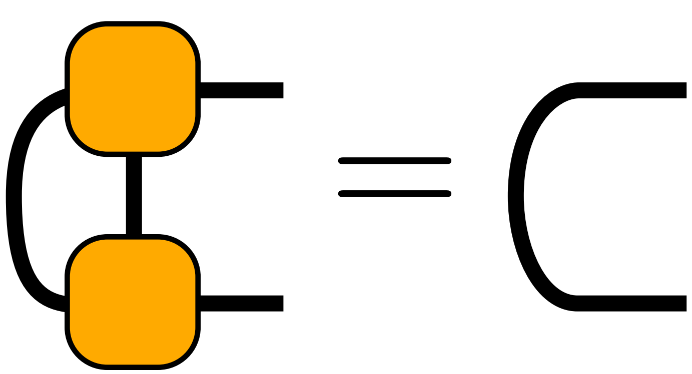


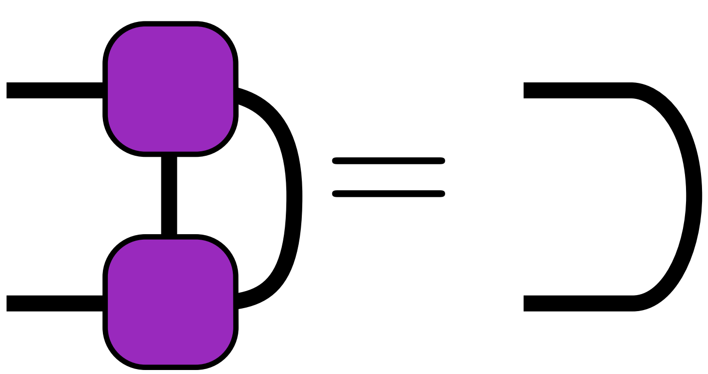
Measuring an MPS Using ITensor:

```
//Measure Sz on second site
Real sz_expect = (dag(prime(psi.A(2),Site))
                  * sites.op("Sz",2)
                  * psi.A(2)).toReal();
```



Recall:





We'll measure the dimer order of the J_1-J_2 model

`<library folder>/tutorial/04_mps`

1. Read through **`j1j2.cc`**; compile; and run

2. Call `psi.position(N/2);` to gauge the MPS to site `N/2`

3. Measure $\hat{B}_{N/2} = \mathbf{S}_{N/2} \cdot \mathbf{S}_{N/2+1}$

```
ITensor wf = psi.A(N/2)*psi.A(N/2+1);
Real b = (dag(prime(wf,Site))*B(sites,N/2)*wf).toReal();
```

4. Repeat for bonds $(\mathrm{N}/2\text{-}1)$ and $(\mathrm{N}/2+1)$. (Don't forget to call `psi.position(b);` to include the "gauge center" `b` in each bond!!) Use to compute and save dimer order parameter:

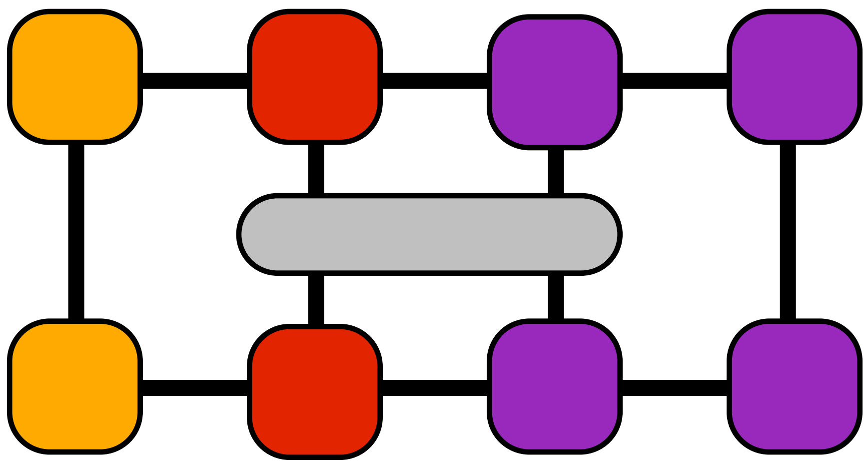
$$D = \langle \hat{B}_{N/2} \rangle - \frac{1}{2}\langle \hat{B}_{N/2-1} \rangle - \frac{1}{2}\langle \hat{B}_{N/2+1} \rangle$$

Solution for missing code (near line 40 of `j1j2.cc`):

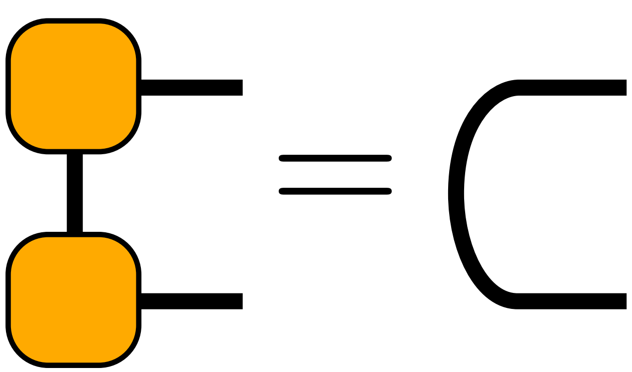
```
psi.position(N/2-1);
ITensor wf = psi.A(N/2-1)*psi.A(N/2);
val += -0.5*(dag(prime(wf,Site))*B(sites,N/2-1)*wf).toReal();

psi.position(N/2);
wf = psi.A(N/2)*psi.A(N/2+1);
val += (dag(prime(wf,Site))*B(sites,N/2)*wf).toReal();

psi.position(N/2+1);
wf = psi.A(N/2+1)*psi.A(N/2+2);
val += -0.5*(dag(prime(wf,Site))*B(sites,N/2+1)*wf).toReal();
```

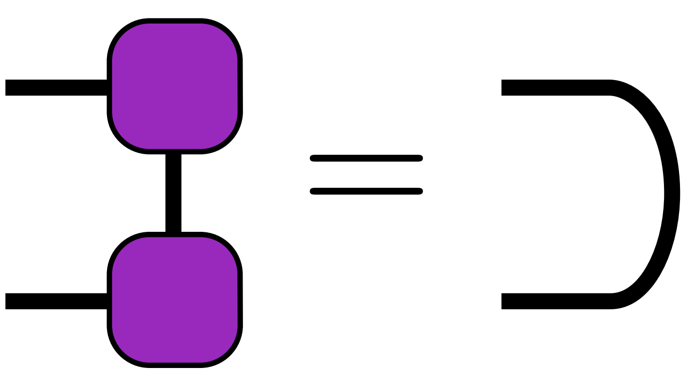
05 Trotter

Just as we can measure one-site operators,
can measure two-site operators

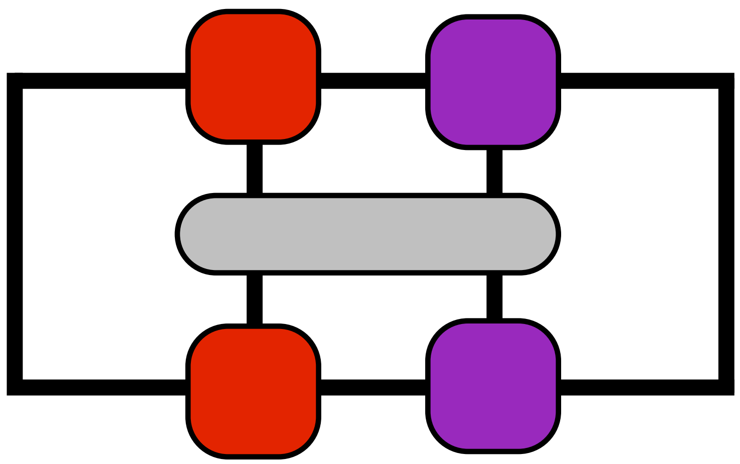


Recall:

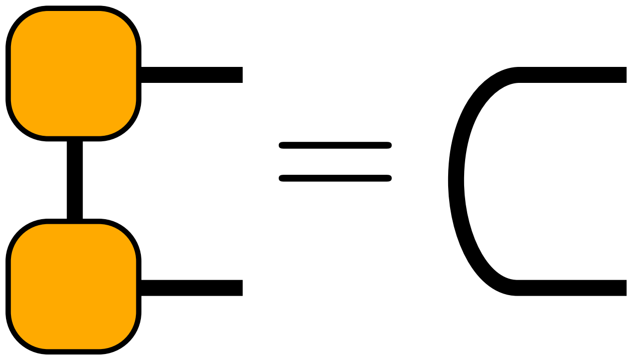


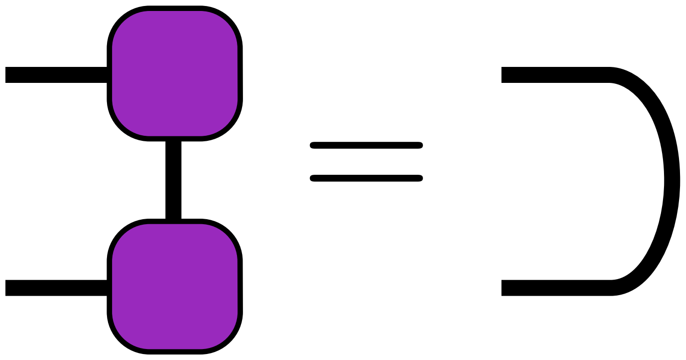


Just as we can measure one-site operators, can measure two-site operators

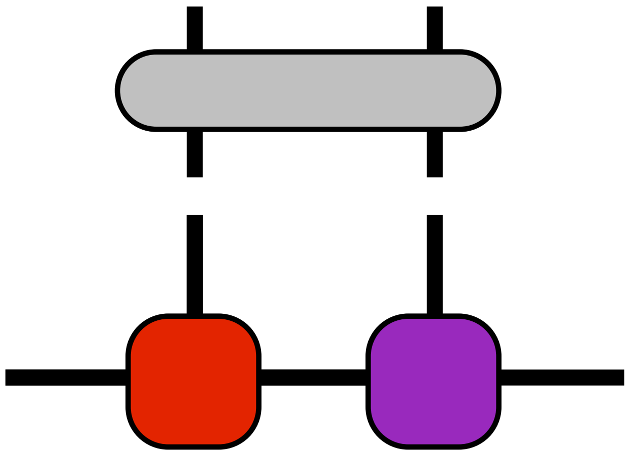


Recall:

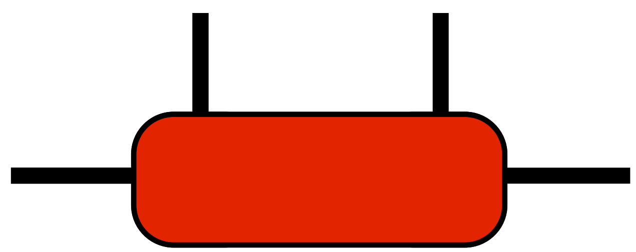




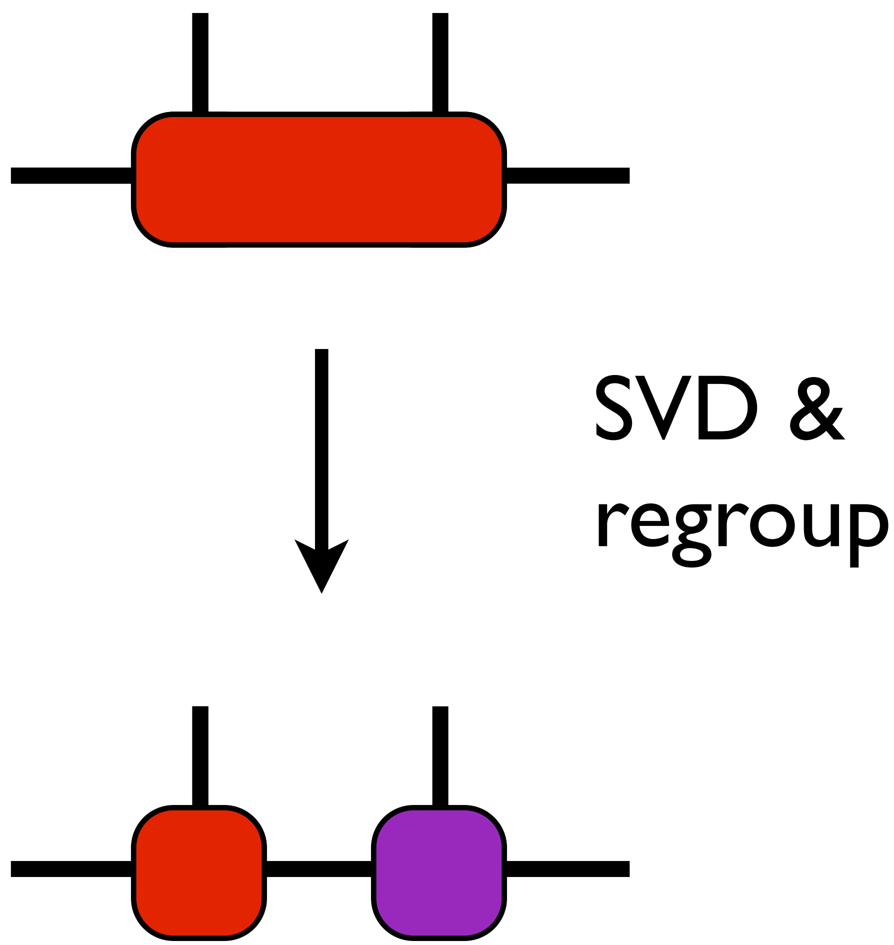
Since two "center" sites have orthogonal environment, ok to apply operators:



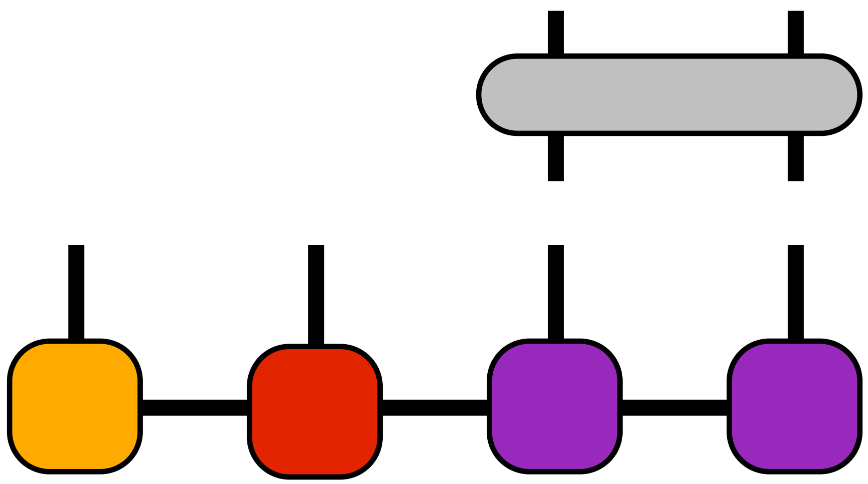
Since two “center” sites have orthogonal environment, ok to apply operators:



Since two "center" sites have orthogonal environment, ok to apply operators:

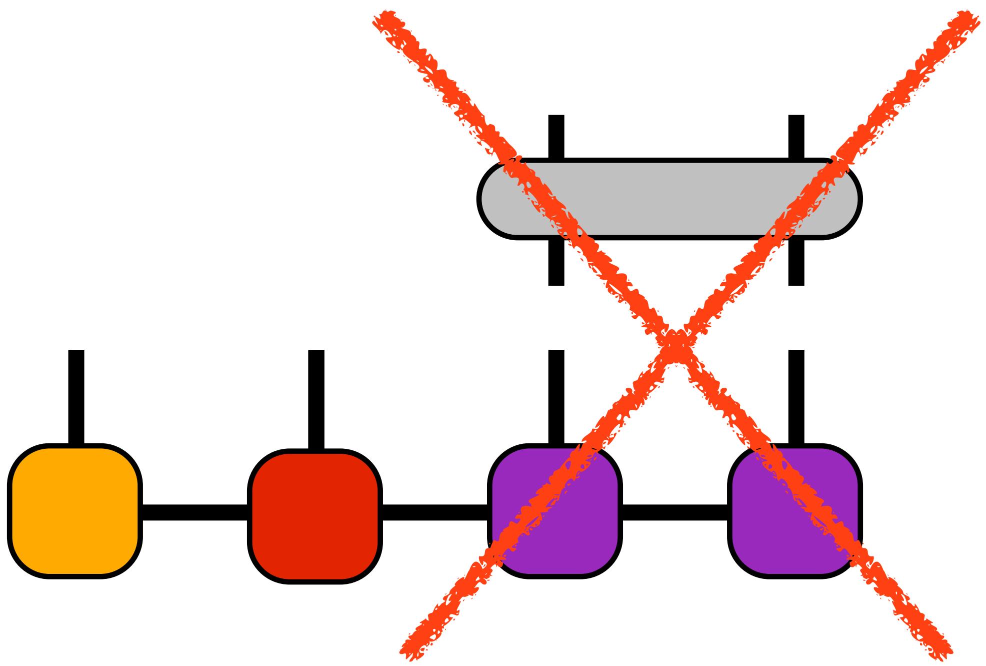


Would NOT be ok on another bond without regauging



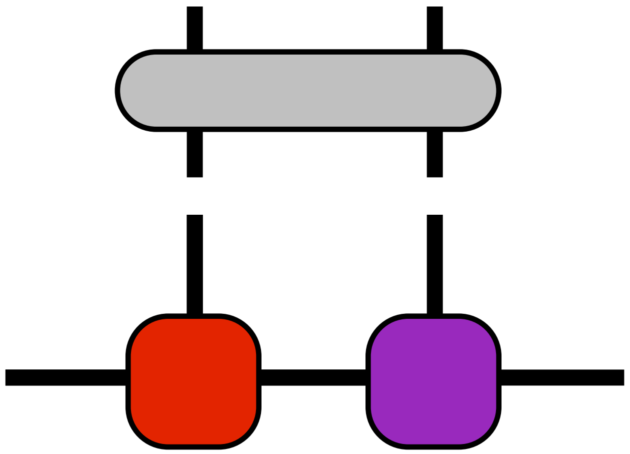
Truncating SVD not globally optimal except at orthogonality center

Would NOT be ok on another bond without regauging



Truncating SVD not globally optimal except at orthogonality center

Q: What can we do with this capability?



A: For short-ranged Hamiltonians, can time evolve

Trick is to use Trotter decomposition

Useful for Hamiltonians of the form

$$H = H_1 + H_2 + H_3 + \ldots$$

For example

$$H = \sum_j \mathbf{S}_j \cdot \mathbf{S}_{j+1}$$

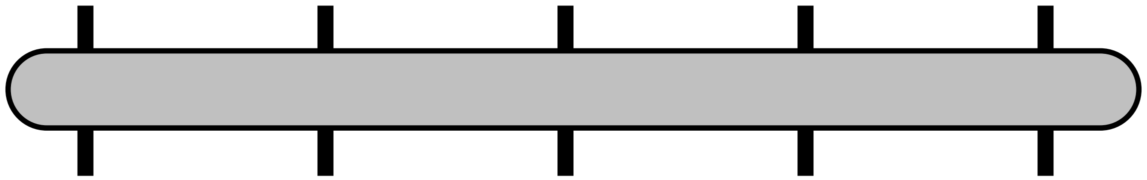
$$= (\mathbf{S}_1 \cdot \mathbf{S}_2) + (\mathbf{S}_2 \cdot \mathbf{S}_3) + (\mathbf{S}_3 \cdot \mathbf{S}_4)$$

For a small time step τ

$$e^{-\tau H} \simeq e^{-\tau H_1/2} e^{-\tau H_2/2} e^{-\tau H_3/2} \ldots$$

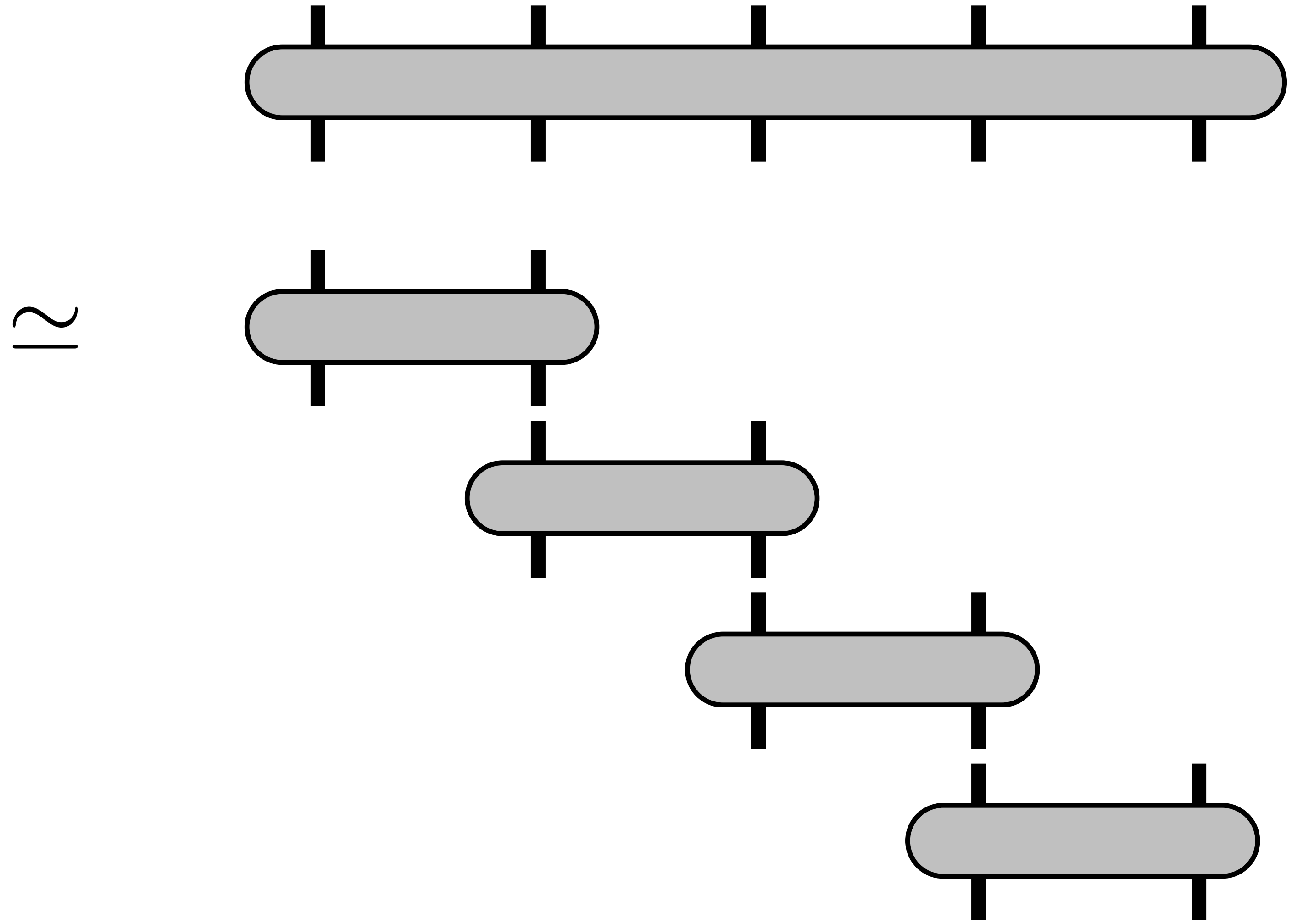
$$\cdots e^{-\tau H_3/2} e^{-\tau H_2/2} e^{-\tau H_1/2} + \mathcal{O}(\tau^3)$$

Diagramatically,

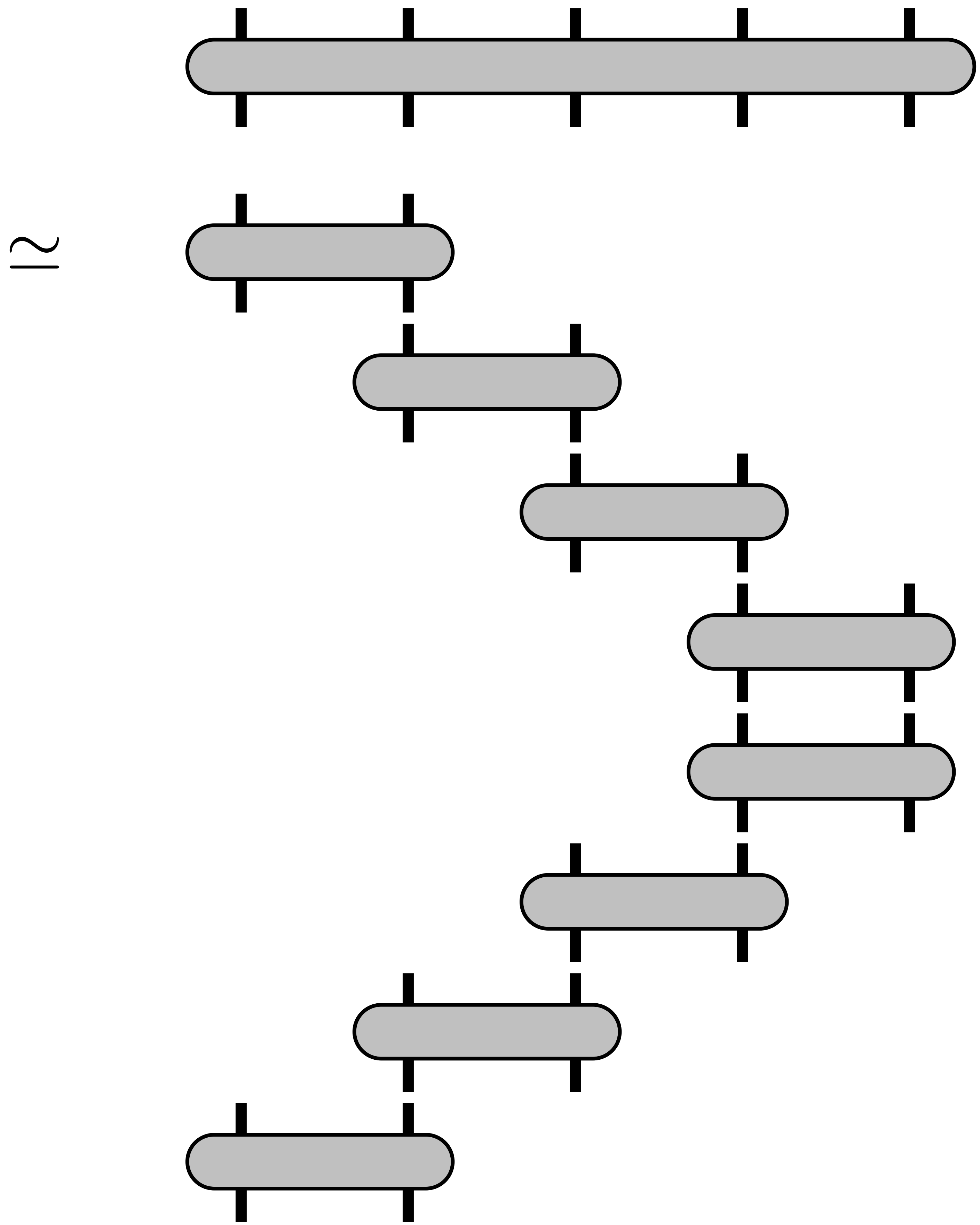


$$\simeq$$

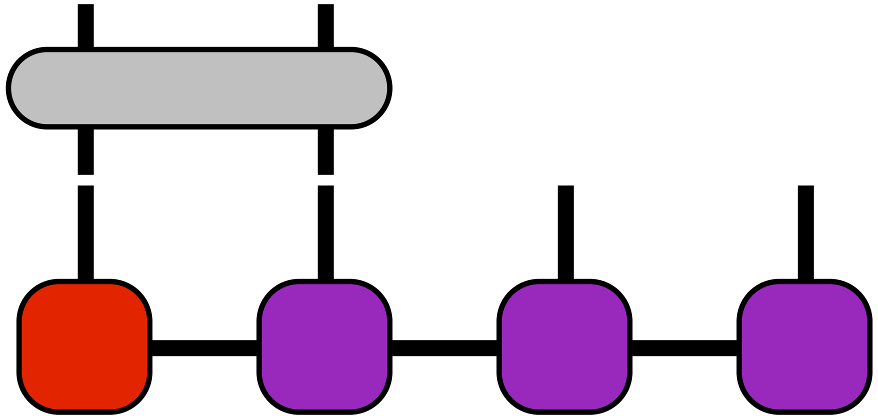
Diagramatically,



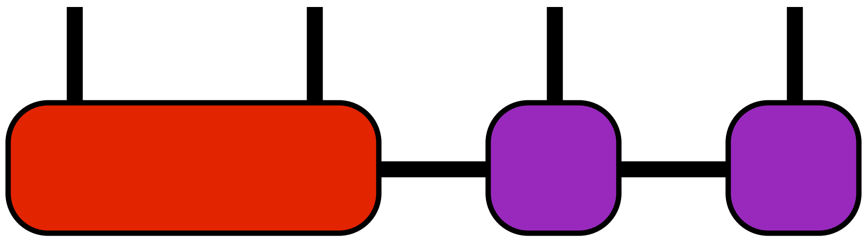
Diagramatically,



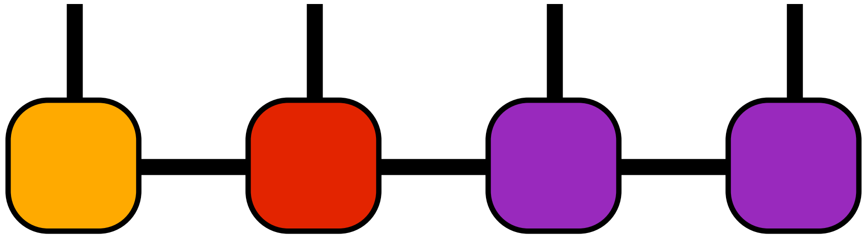
Apply to MPS as follows:



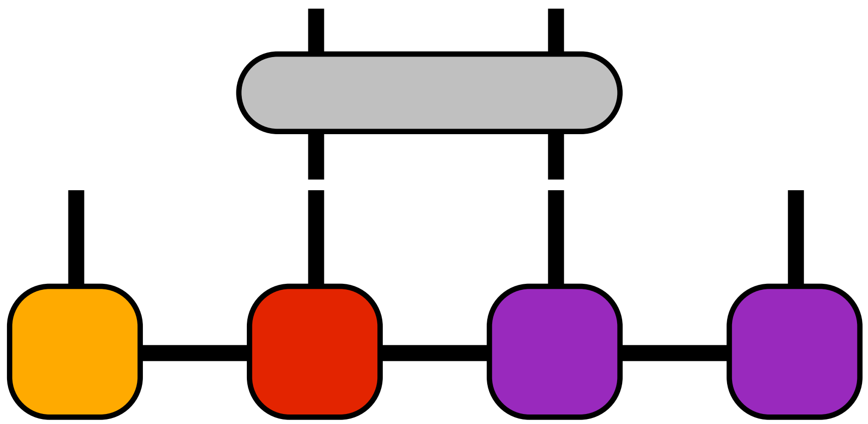
Apply to MPS as follows:



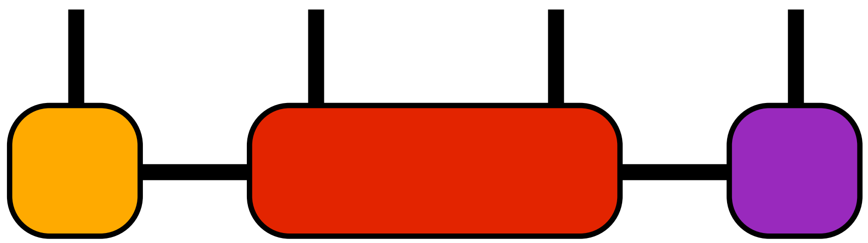
Apply to MPS as follows:



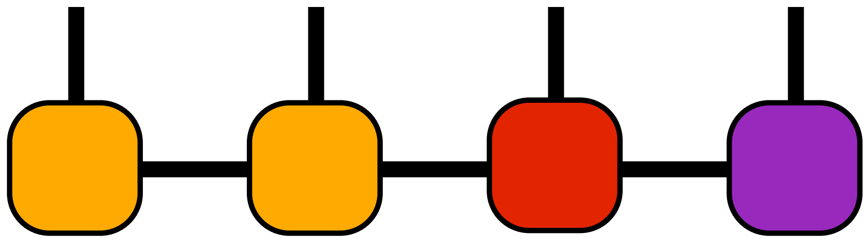
Apply to MPS as follows:



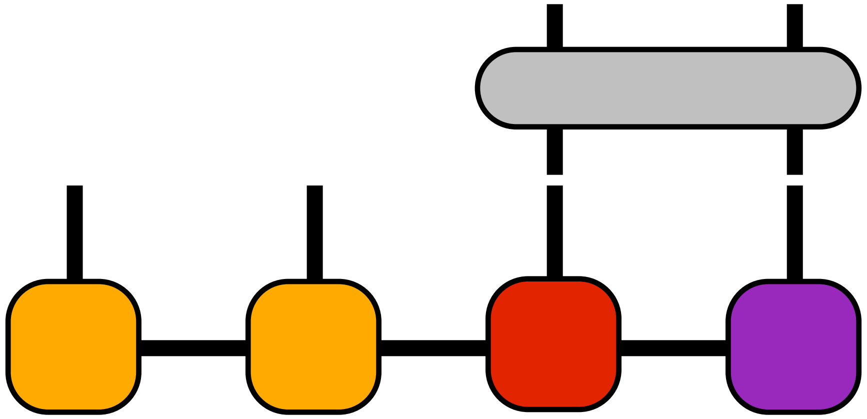
Apply to MPS as follows:



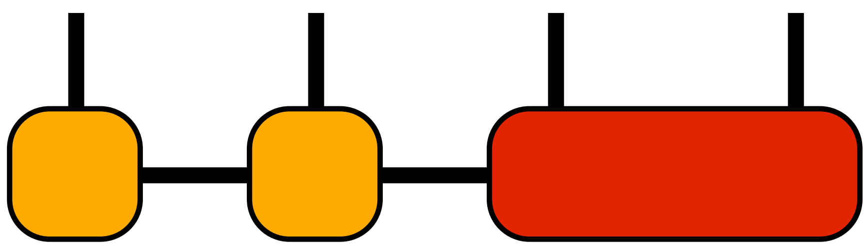
Apply to MPS as follows:



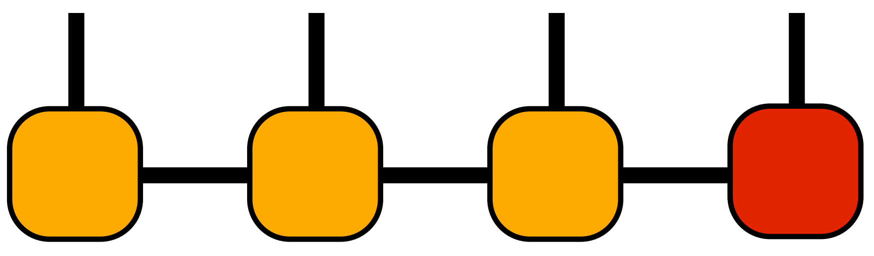
Apply to MPS as follows:



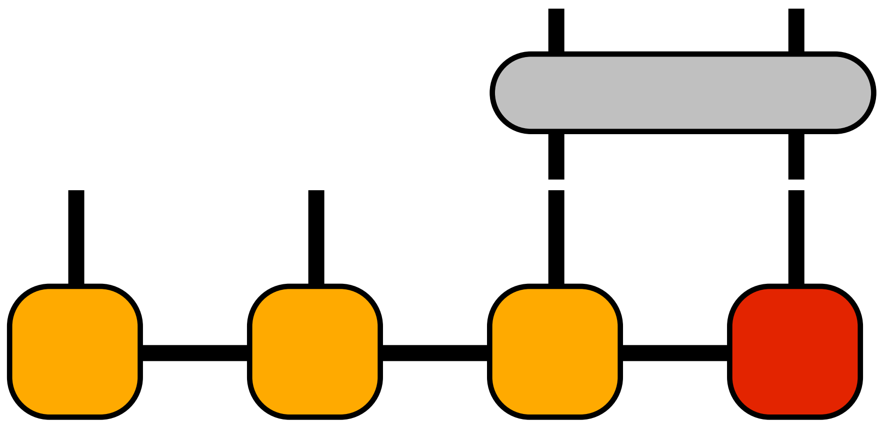
Apply to MPS as follows:



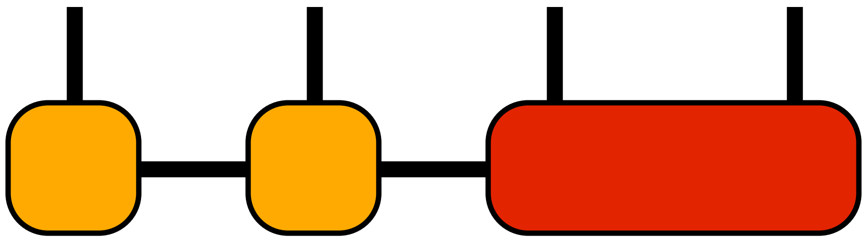
Apply to MPS as follows:



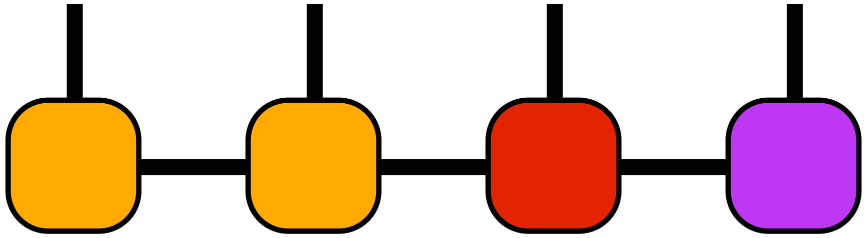
Apply to MPS as follows:



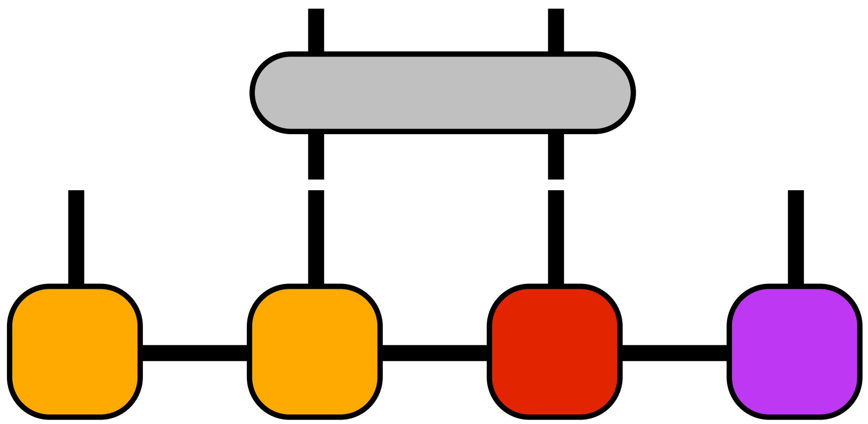
Apply to MPS as follows:



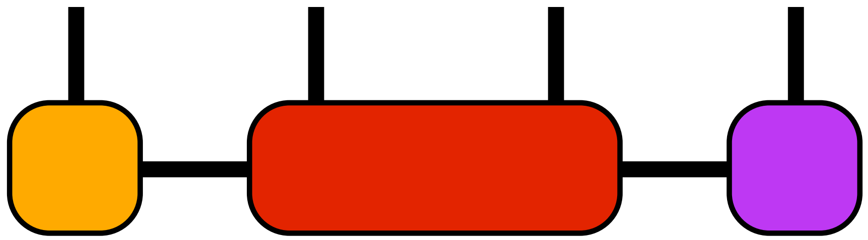
Apply to MPS as follows:



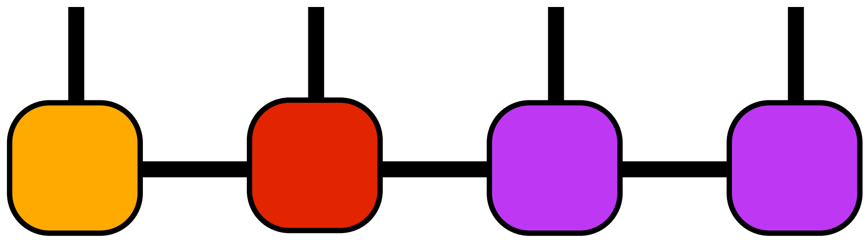
Apply to MPS as follows:



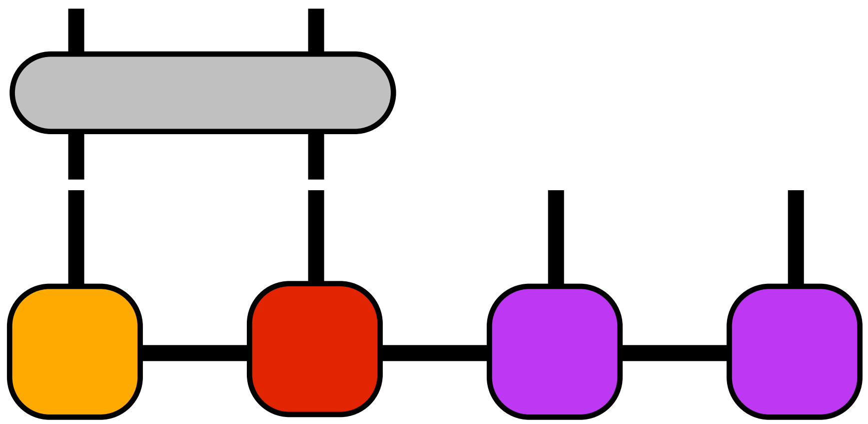
Apply to MPS as follows:



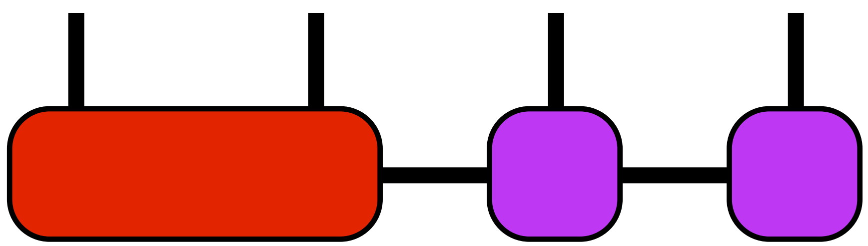
Apply to MPS as follows:



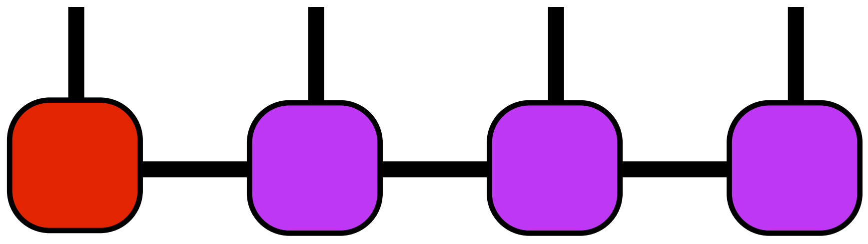
Apply to MPS as follows:



Apply to MPS as follows:



Apply to MPS as follows:



Interesting applications: $$|\psi'\rangle = e^{-\tau H}|\psi\rangle$$

If τ real (imaginary time evolution), enough steps will give **ground state**

If τ imaginary, evolve in real time, study **dynamics** [1]

Evolving through imaginary time $\beta/2 = 1/(2T)$ simulates **finite temperature** [2]

[1] White, Feiguin PRL **93**, 076401 (2004)

[2] White PRL **102**, 190601 (2009)

We'll implement time evolution for the Heisenberg chain

`<library folder>/tutorial/05_gates`

1. Read through **`gates.cc`**; compile; and run

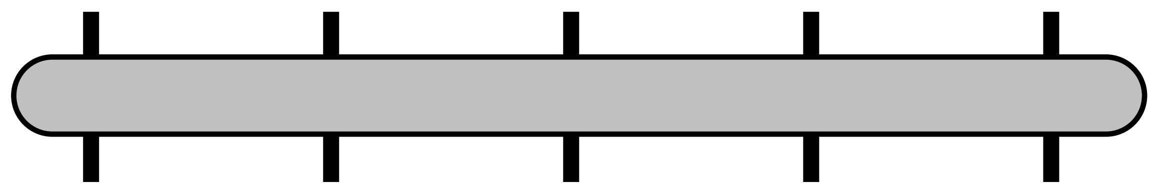
2. Apply the gate G to the MPS bond tensor AA.
   The gate G can be multiplied times AA as if it's an ITensor.

3. Reset the prime level back to zero using AA's
   `.noprime()` class method.

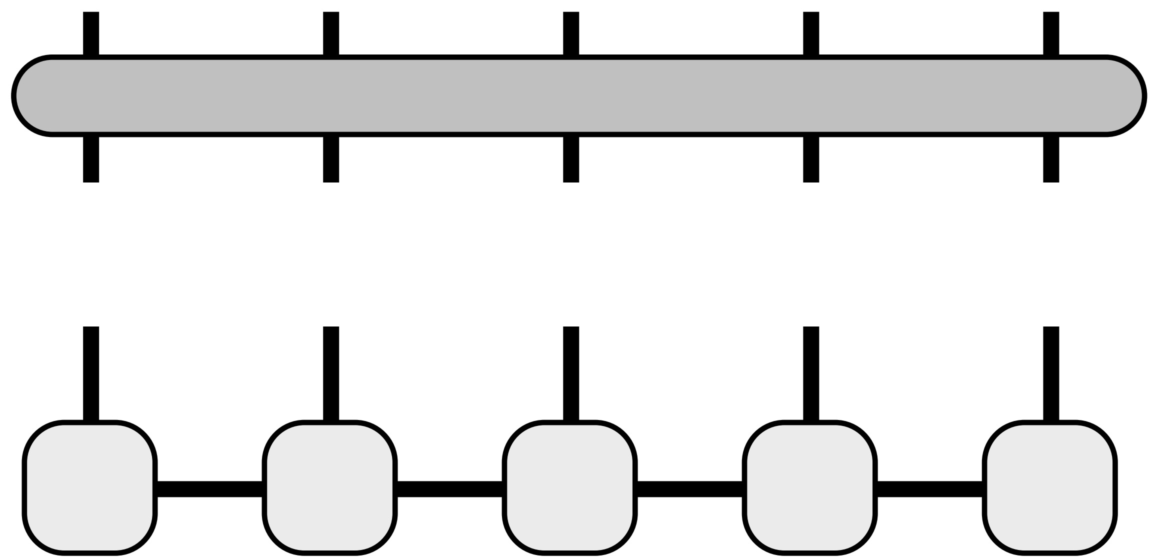
3. Try increasing the total time "ttotal" to imaginary time evolve
   toward the ground state.
   (Exact energy for 20 sites: $E_0 = -8.6824733317$)

05 MPO

We have seen a Hamiltonian looks like this:



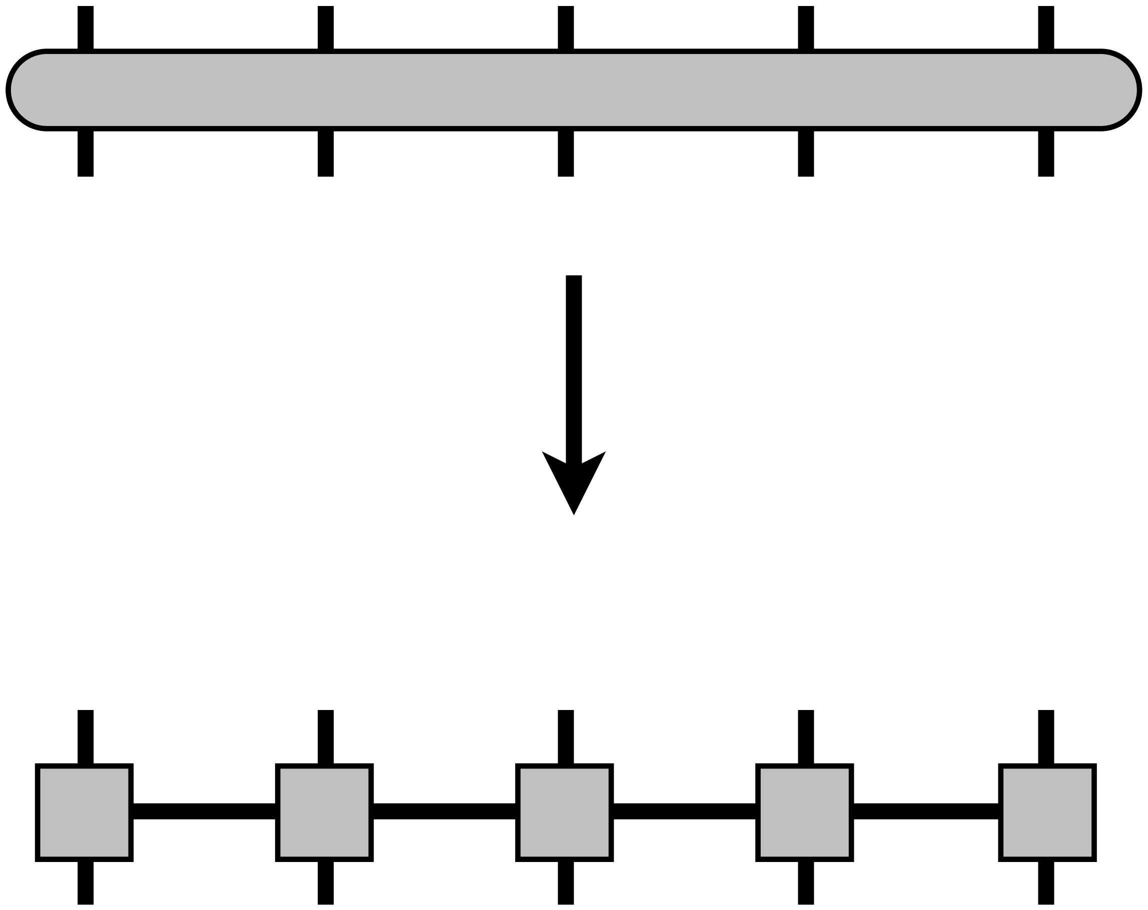
We have seen a Hamiltonian looks like this:



$$\hat{H}|\Psi\rangle$$

Does a 1d Hamiltonian have a local form/factorization like an MPS?

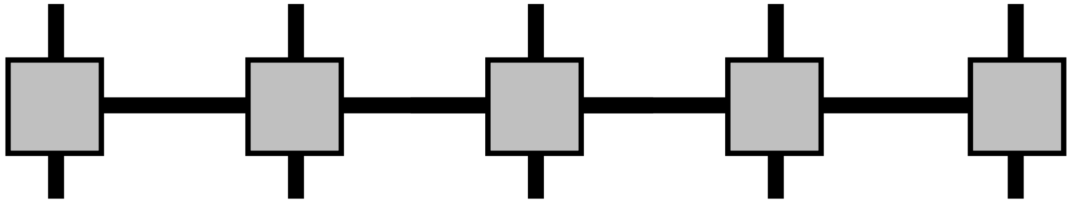
Want something like



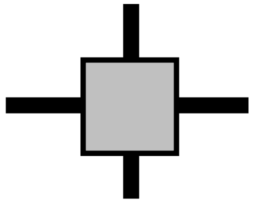
Operator (H) as product of "matrices"

matrix product operator

Focus on just one tensor

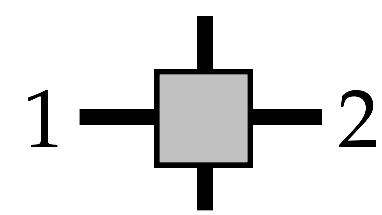


Focus on just one tensor



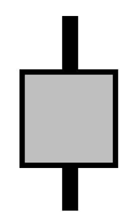
Focus on just one tensor

Specific values for horizontal bonds
gives site operator



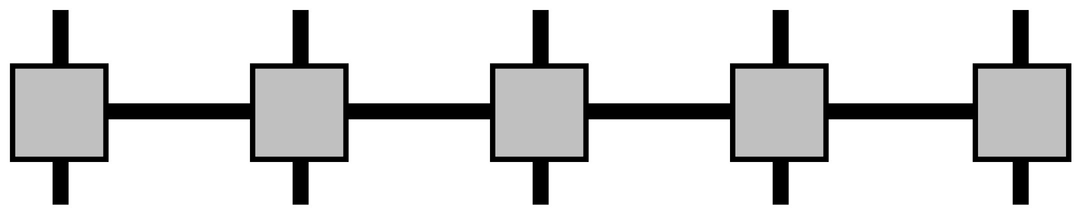
Focus on just one tensor

Specific values for horizontal bonds
gives site operator



Focus on just one tensor

Specific values for horizontal bonds
gives site operator



$\longrightarrow$ Each tensor a matrix of site operators!

$\longrightarrow$ Each tensor a matrix of site operators!

Hamiltonians can be written

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$\longrightarrow$ Each tensor a matrix of site operators!

Multiply out

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} \hat{I} \\ \hat{\sigma}^z \end{bmatrix}$$

$$\hat{\sigma}_1^z \otimes \hat{I}_2 + \hat{I}_1 \otimes \hat{\sigma}_2^z$$

$$
\begin{bmatrix} 0 & 1 \end{bmatrix}
\begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix}
\begin{bmatrix} \hat{I} & \\ \hat{\sigma}^z & \hat{I} \end{bmatrix}
\begin{bmatrix} 1 \\ 0 \end{bmatrix}
$$

This Hamiltonian is

$$
H = \sum_i \hat{\sigma}_i^z
$$

More complicated example

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}} \quad 3 \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \quad \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix}^{1} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

More complicated example

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}} \quad 3 \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \quad \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} 1 \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$\hat{\sigma}^z$

$\bullet$ $\bullet$

More complicated example

$$
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}}
\quad
\begin{array}{cc}
 & \begin{array}{ccc} \phantom{-h\hat{\sigma}^x} & 2 & \phantom{\hat{I}} \end{array} \\
\begin{array}{c} \\ \\ 3 \end{array} & \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix}
\end{array}
\quad
\begin{array}{ccc}
 & \begin{array}{ccc} 1 & \phantom{\hat{\sigma}^z} & \phantom{\hat{I}} \end{array} & \\
\begin{array}{c} \\ 2 \\ \\ \end{array} & \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} & \begin{array}{c} 1 \\ \\ \\ \end{array}
\end{array}
\quad
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
$$

$\hat{\sigma}^z$ $\hat{\sigma}^z$

More complicated example

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}} \quad 3 \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \quad \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} 1 \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\bullet \qquad \bullet$$

More complicated example

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}} \quad \begin{array}{c} \\ \\ 3 \end{array} \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \quad \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{array}{c} 1 \\ \\ \\ \end{array} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$-h\hat{\sigma}^x$$

More complicated example

$$
\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}}
\quad
\begin{array}{cc}
 & \begin{array}{ccc} 1 & & \end{array} \\
\begin{array}{c} \\ \\ 3 \end{array} &
\begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix}
\end{array}
\quad
\begin{array}{ccc}
 & \begin{array}{ccc} 1 & & \end{array} & \\
\begin{array}{c} 1 \\ \\ \\ \end{array} &
\begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} &
\begin{array}{c} 1 \\ \\ \\ \end{array}
\end{array}
\quad
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
$$

$-h\hat{\sigma}^x$ $\qquad\qquad$ $\hat{I}$

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} \hat{I} & & \\ \hat{\sigma}^z & 0 & \\ -h\hat{\sigma}^x & \hat{\sigma}^z & \hat{I} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Hamiltonian is

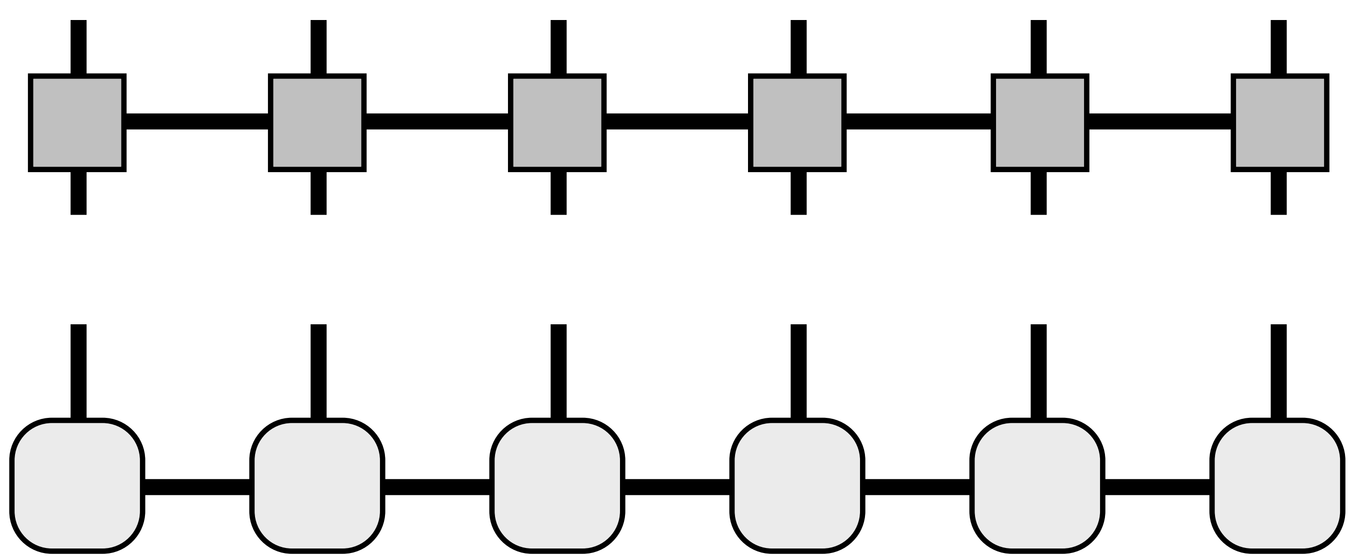
$$\hat{H} = \sum_j \hat{\sigma}_j^z \sigma_{j+1}^z - h\hat{\sigma}_j^x$$

05 DMRG

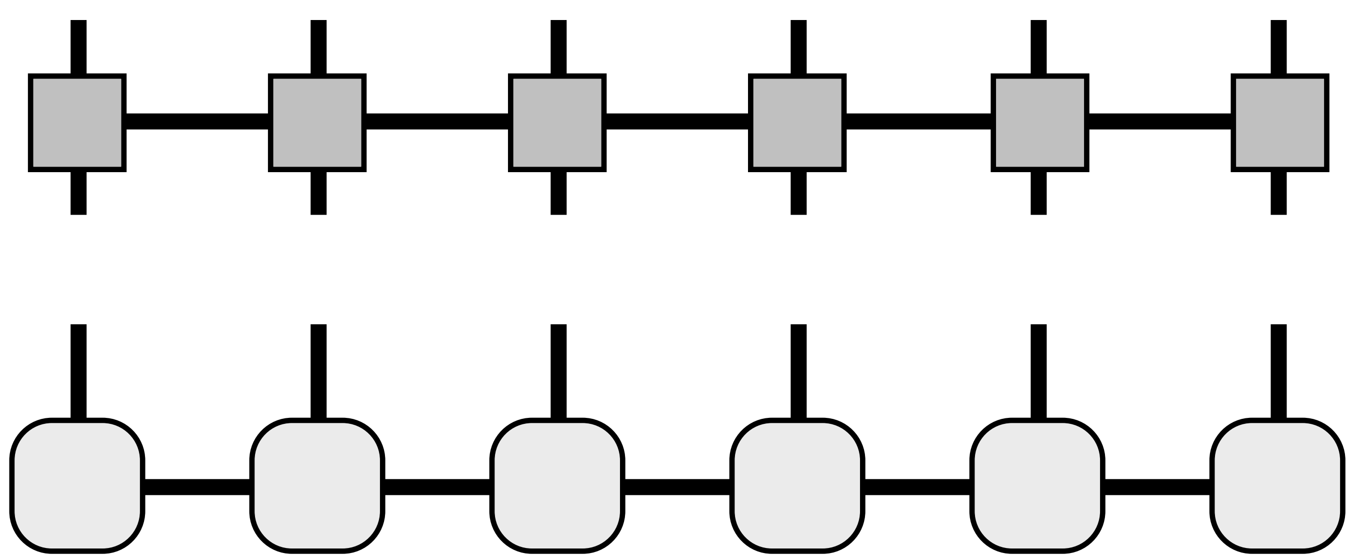
DMRG is the best method for finding ground states of 1d Hamiltonians

Want to solve $H|\Psi\rangle = E|\Psi\rangle$

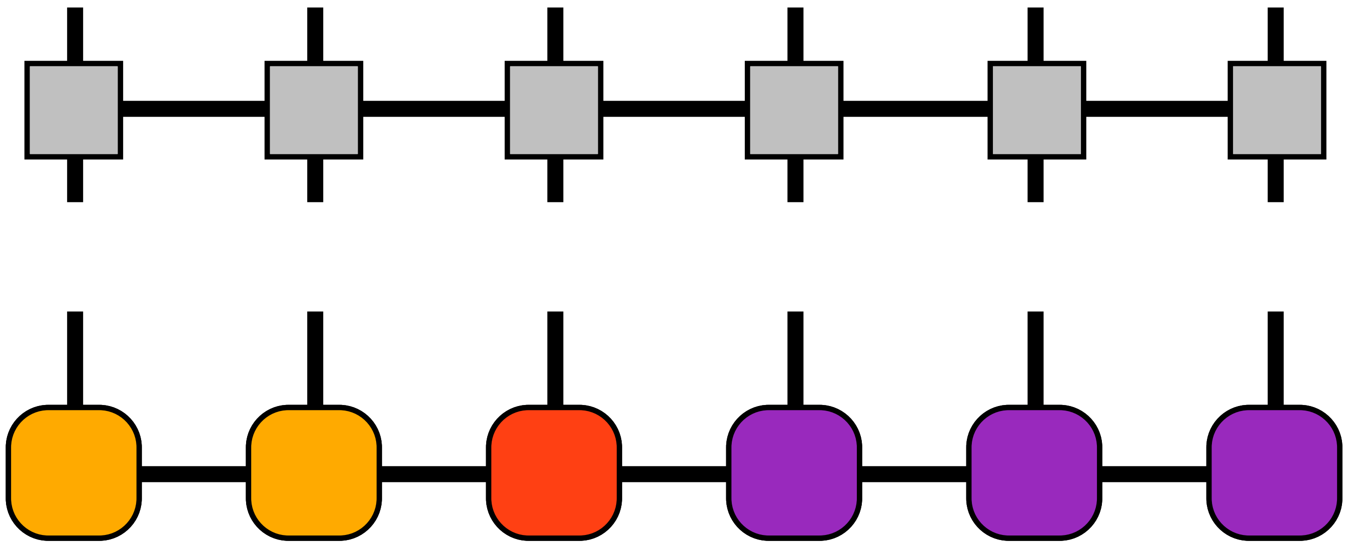
Think of H as MPO



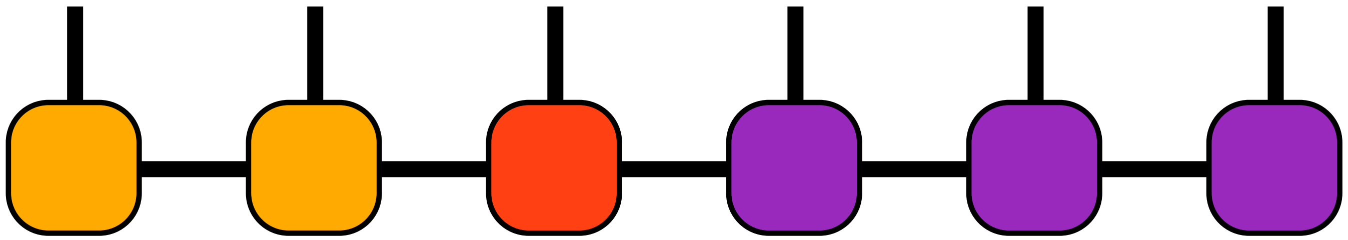
Important: MPS should be in definite gauge
I.e. most tensors unitary



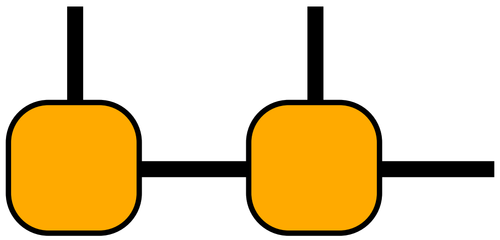
Important: MPS should be in definite gauge
I.e. most tensors unitary



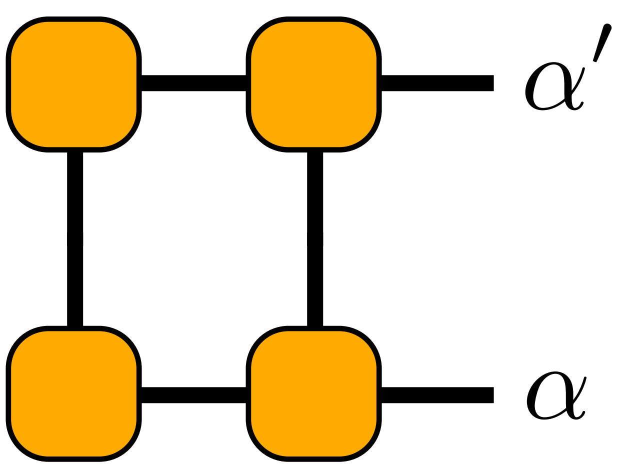
This way, tensors left/right of center define orthonormal bases



This way, tensors left/right of center define orthonormal bases

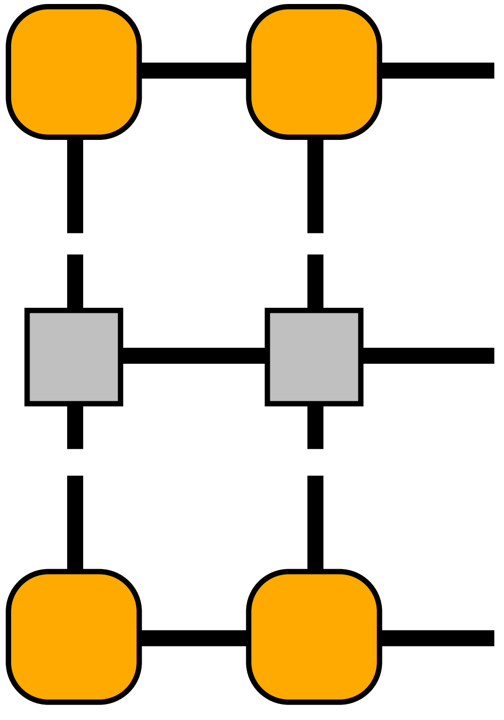


This way, tensors left/right of center define orthonormal bases

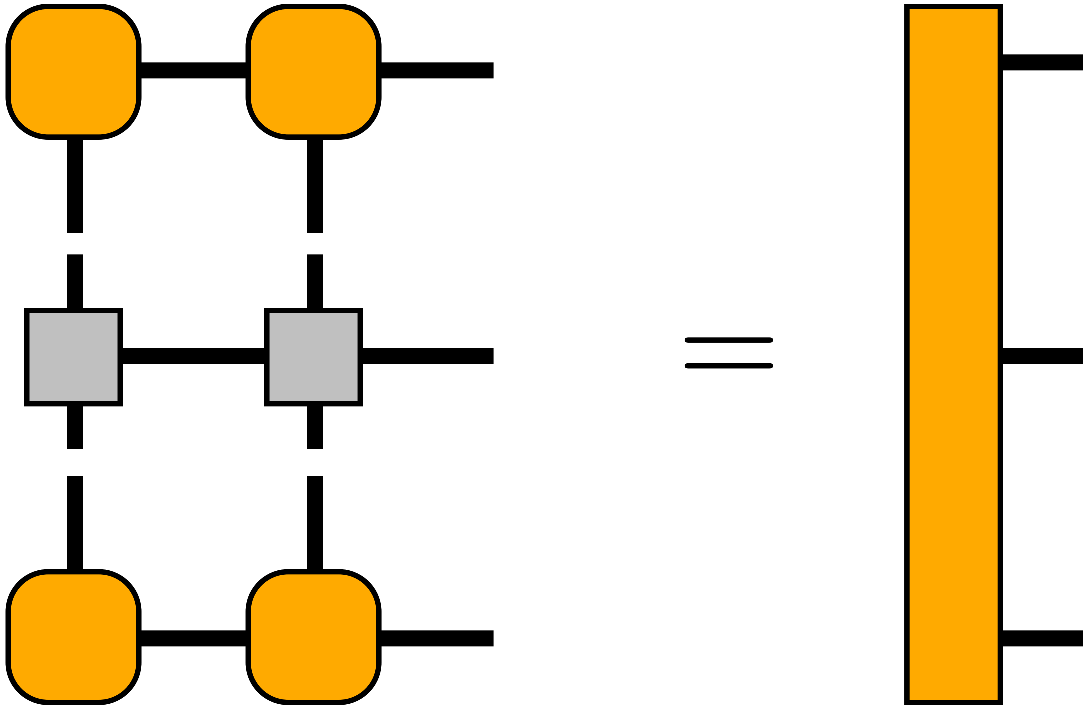


$$= \delta_{\alpha}^{\alpha'}$$

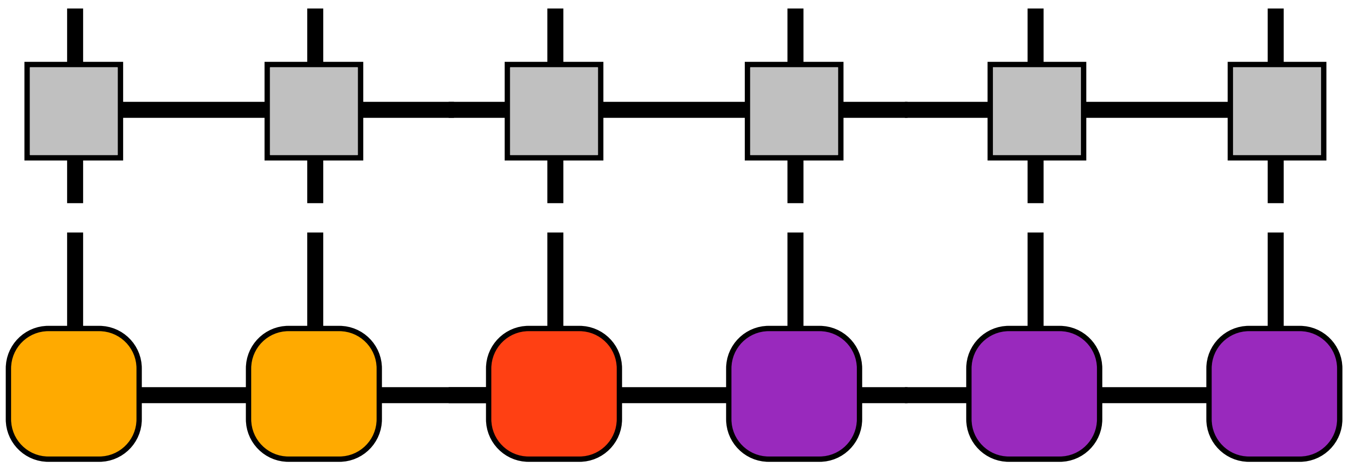
Can project Hamiltonian into this basis



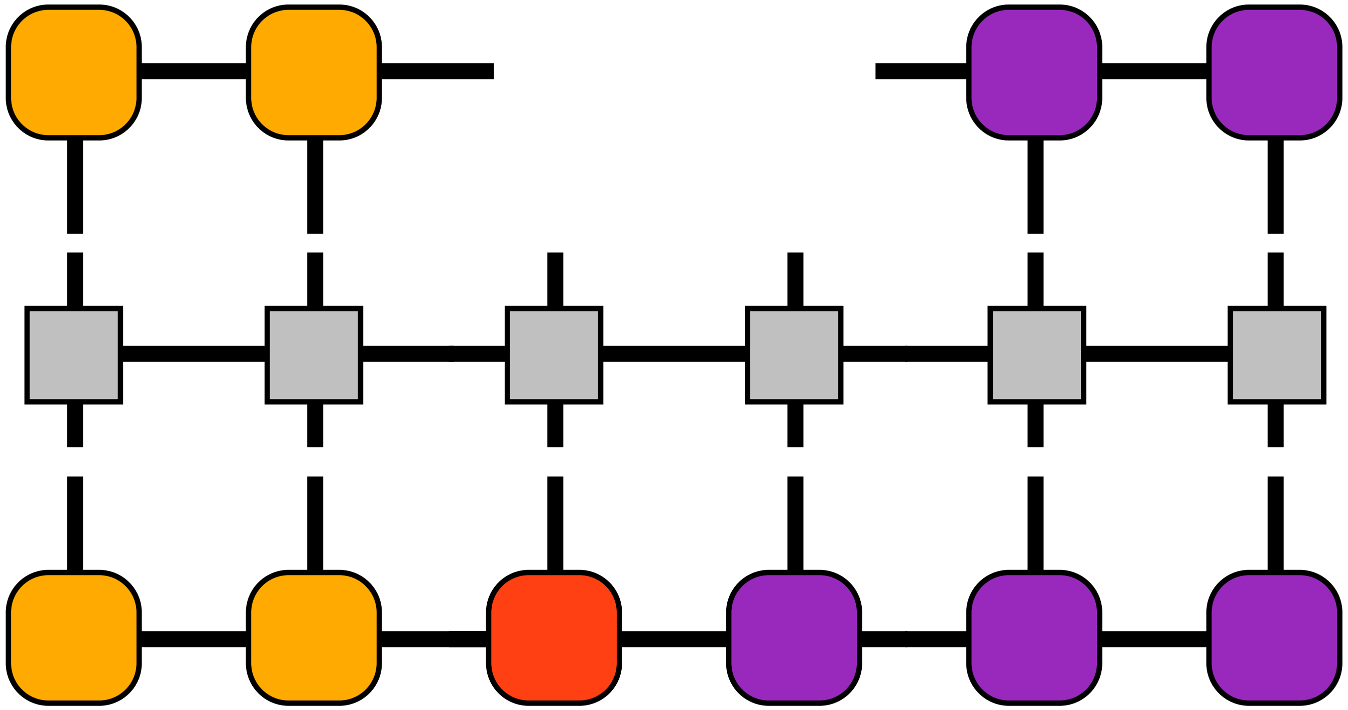
Can project Hamiltonian into this basis



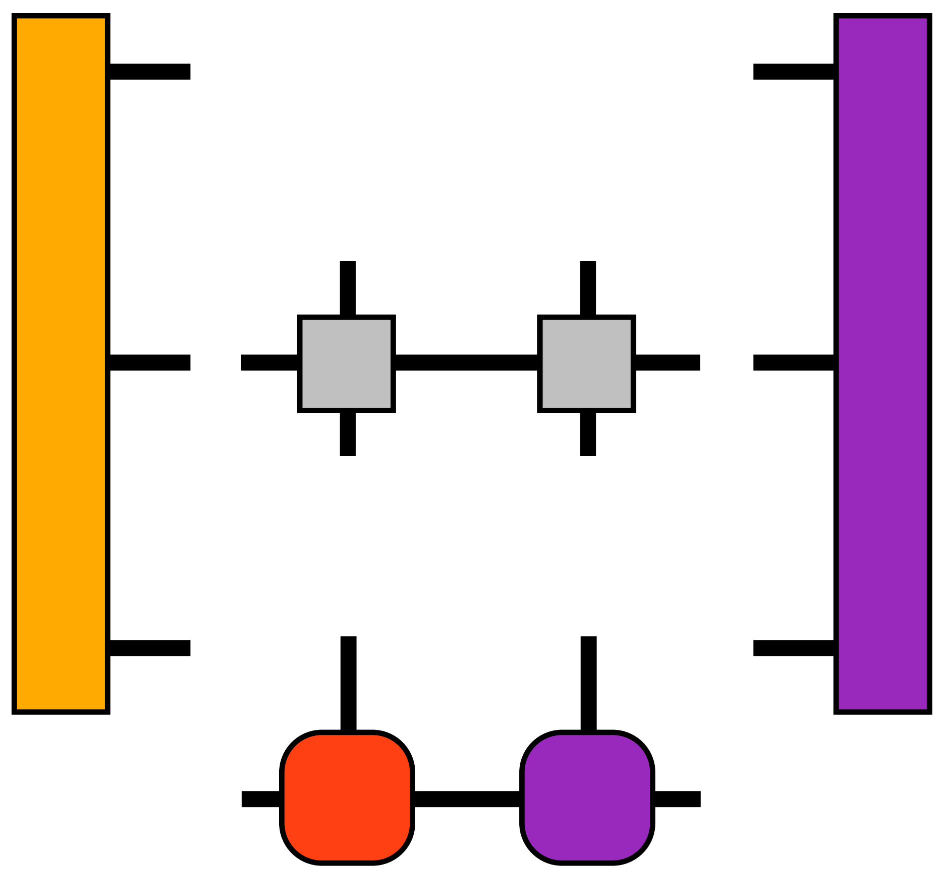
Doing the same on the right gives



Doing the same on the right gives



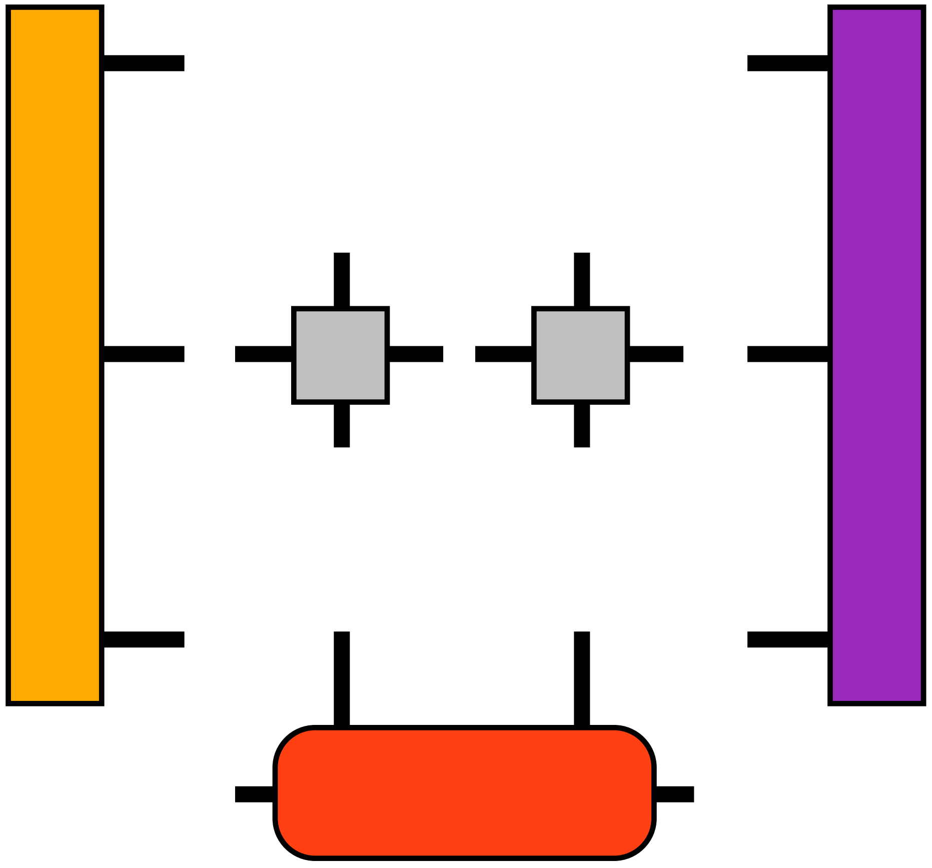
Doing the same on the right gives



$$\tilde{H}|\tilde{\Psi}\rangle = \tilde{E}|\tilde{\Psi}\rangle$$

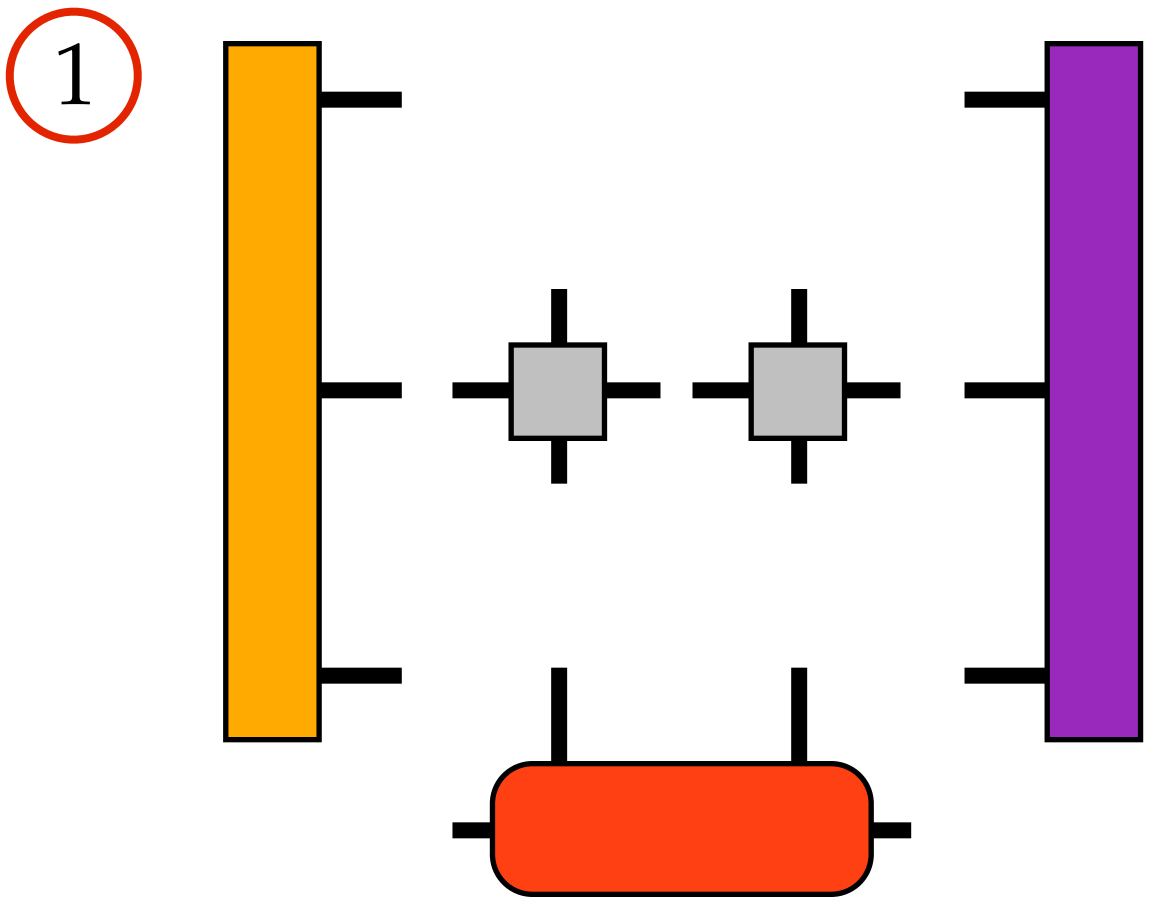
Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

Order important!



Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

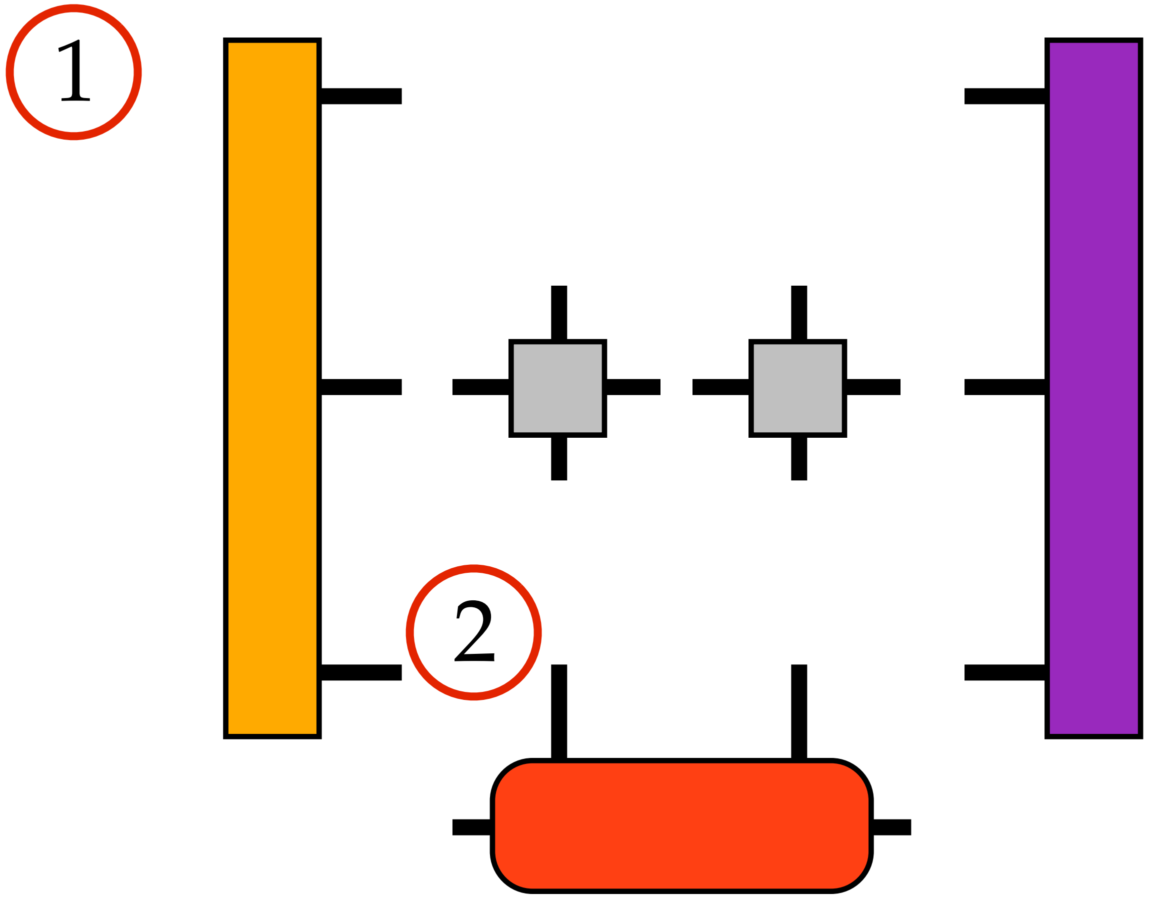
Order important!



Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

Order important!

$2 \sim m^3$

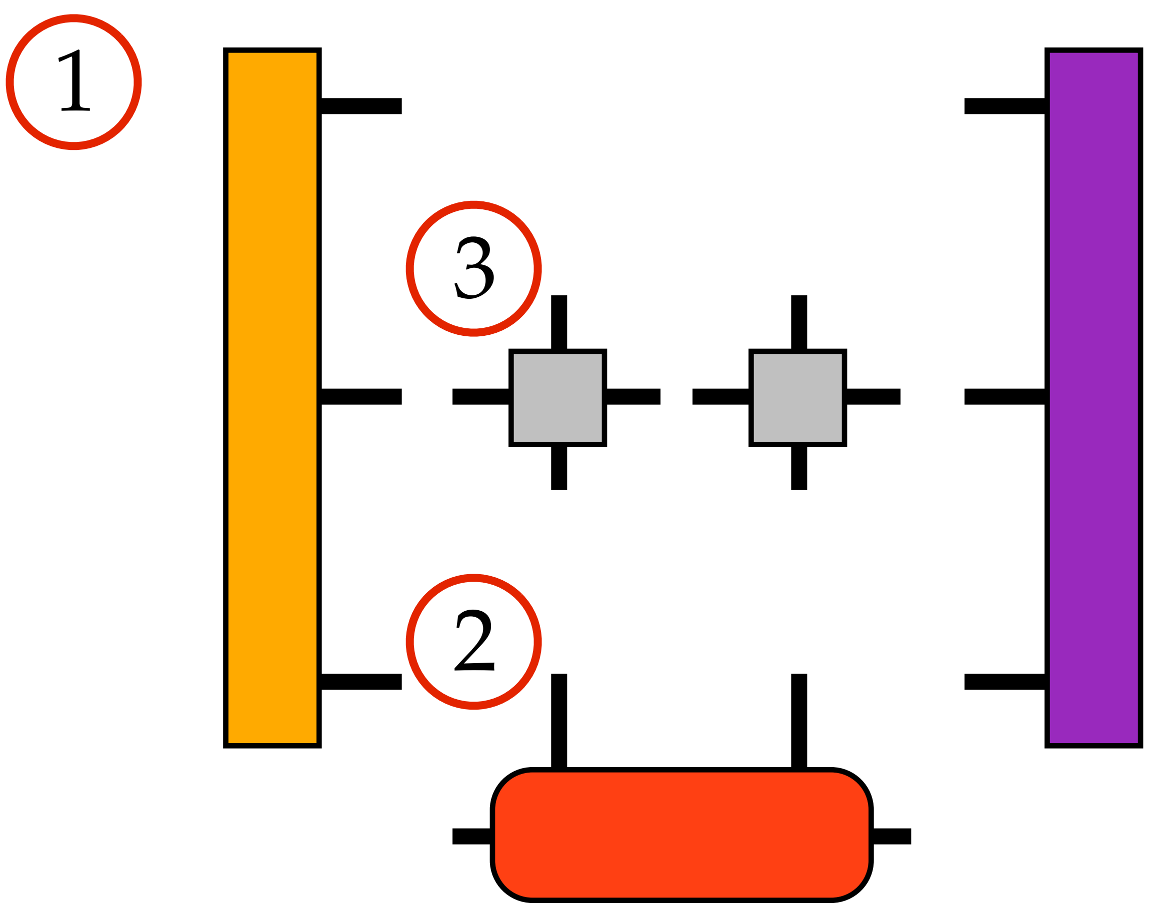


Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

Order important!

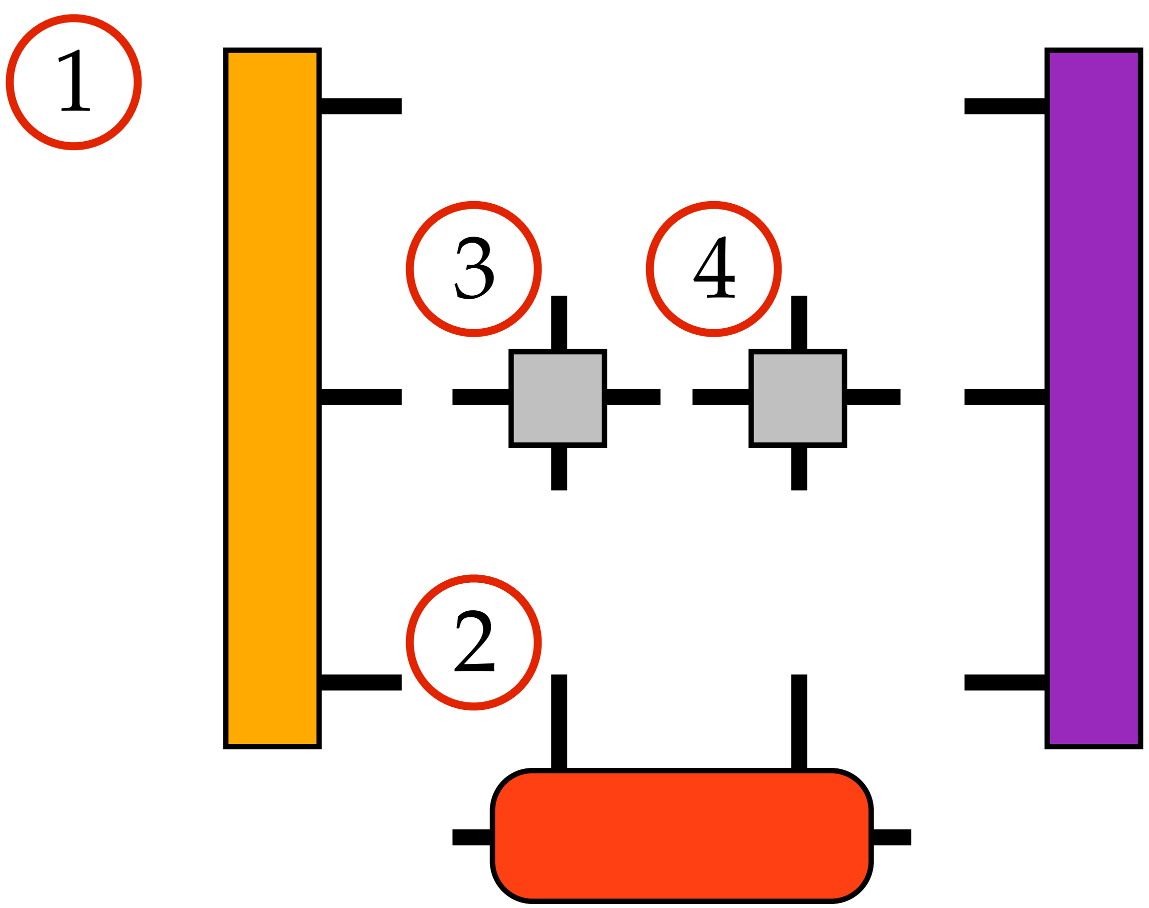
$2 \sim \mathrm{m}^3$

$3 \sim \mathrm{m}^2$



Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

Order important!



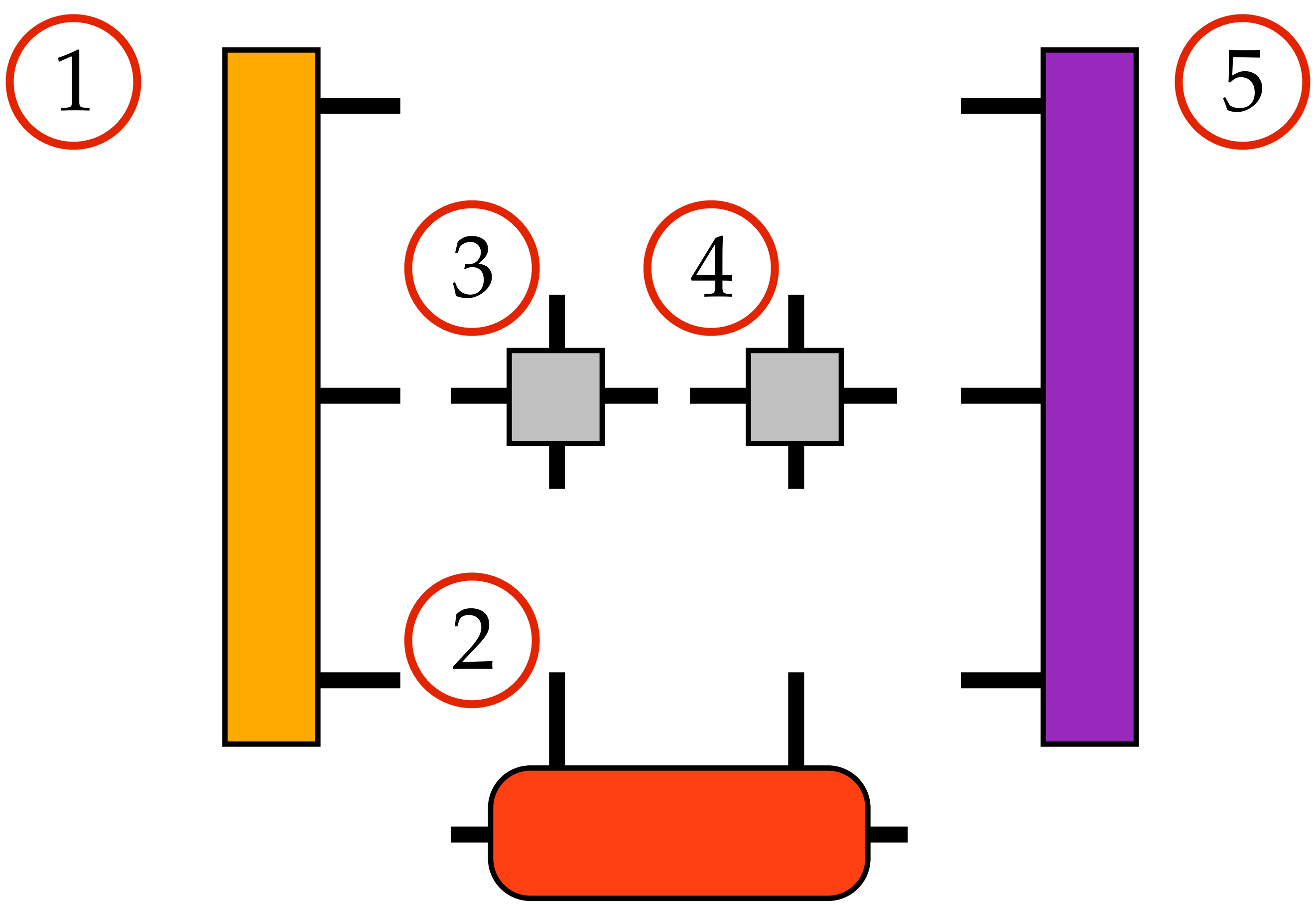
$2 \sim m^3$

$3 \sim m^2$

$4 \sim m^2$

Can efficiently multiply effective $\tilde{H}$ times $|\tilde{\Psi}\rangle$

Order important!



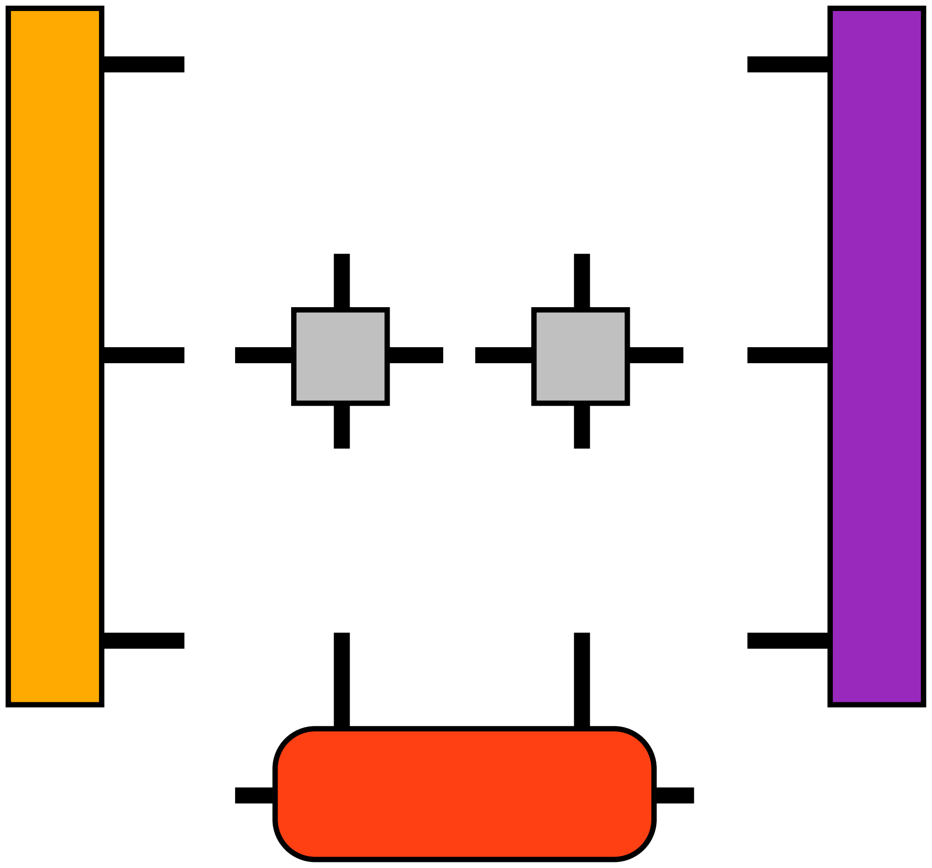
$2 \sim m^3$

$3 \sim m^2$

$4 \sim m^2$

$5 \sim m^3$

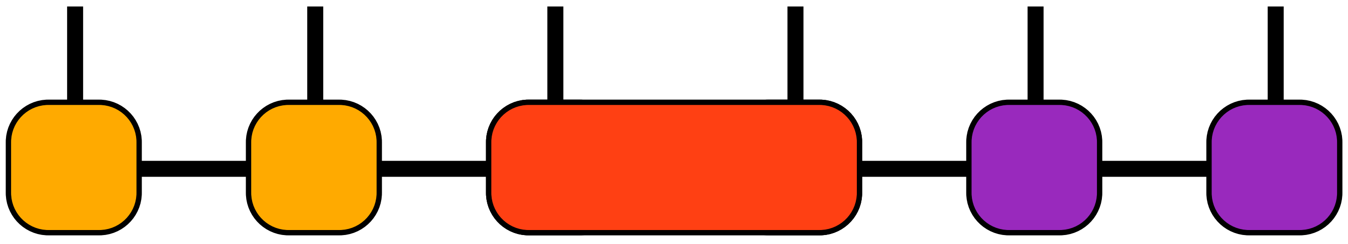
Use Lanczos/Davidson to solve
(sparse matrix eigensolver)



Noack, Manmana, AIP Conf. Proc. **789**, 93 (2005)

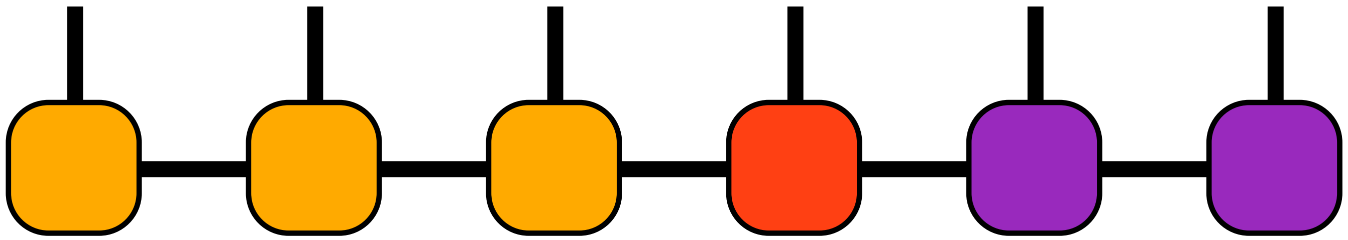
Now, with improved wavefunction,
shift orthogonality center
(using SVD)

Important to truncate to m singular values
("number of states kept" in DMRG)

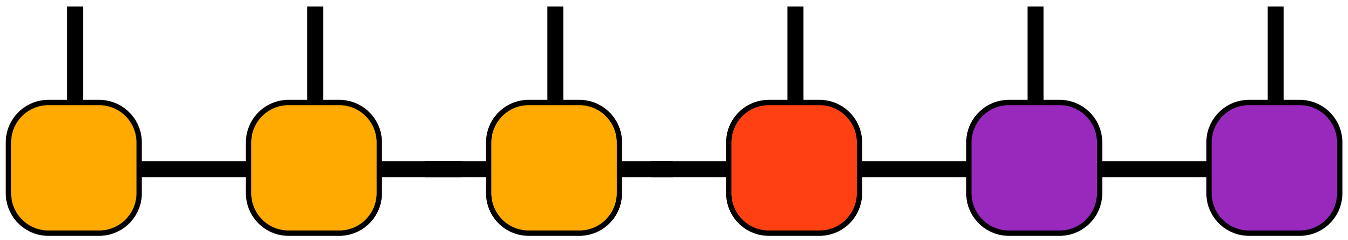


Now, with improved wavefunction,
shift orthogonality center
(using SVD)

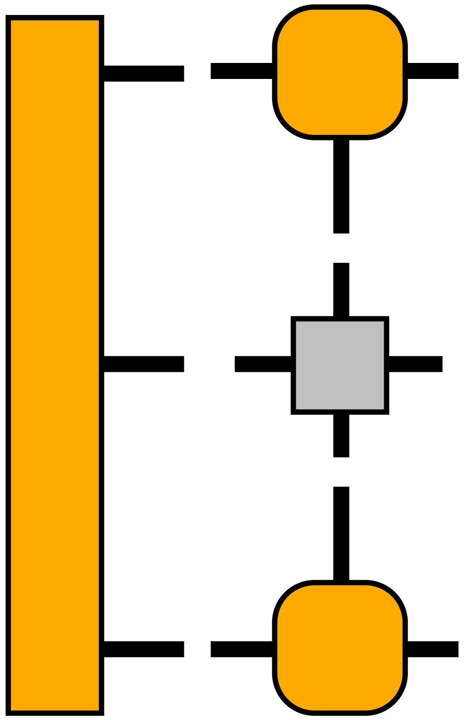
Important to truncate to m singular values
("number of states kept" in DMRG)



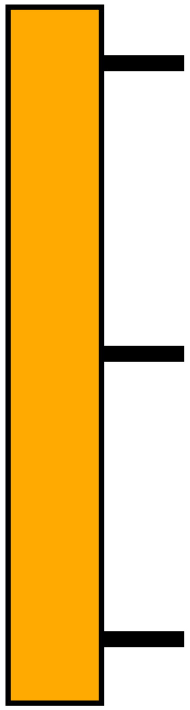
Grow projected Hamiltonian



Grow projected Hamiltonian

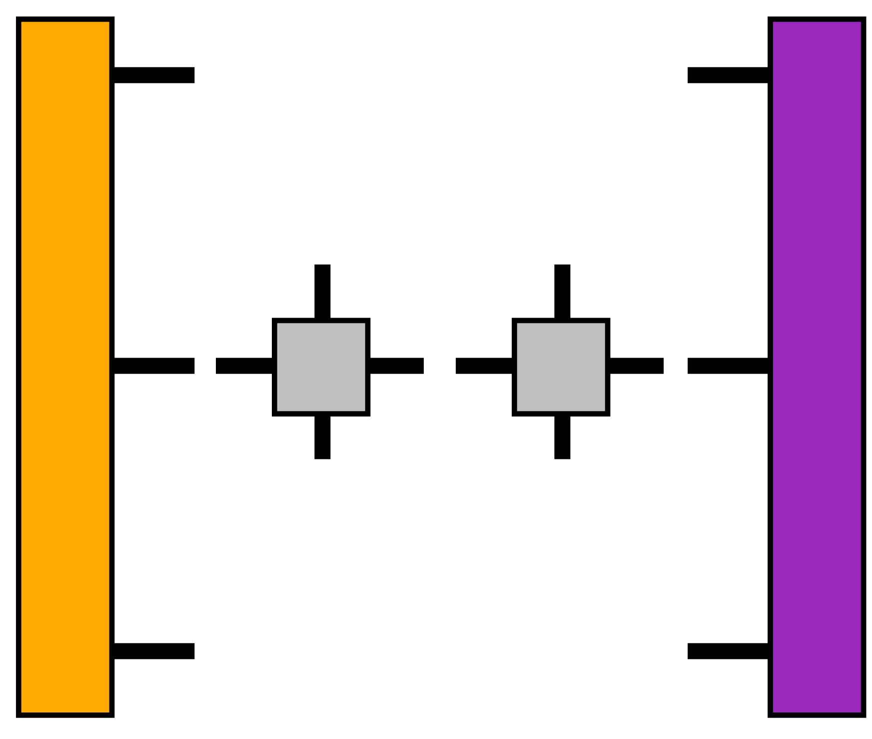


Grow projected Hamiltonian

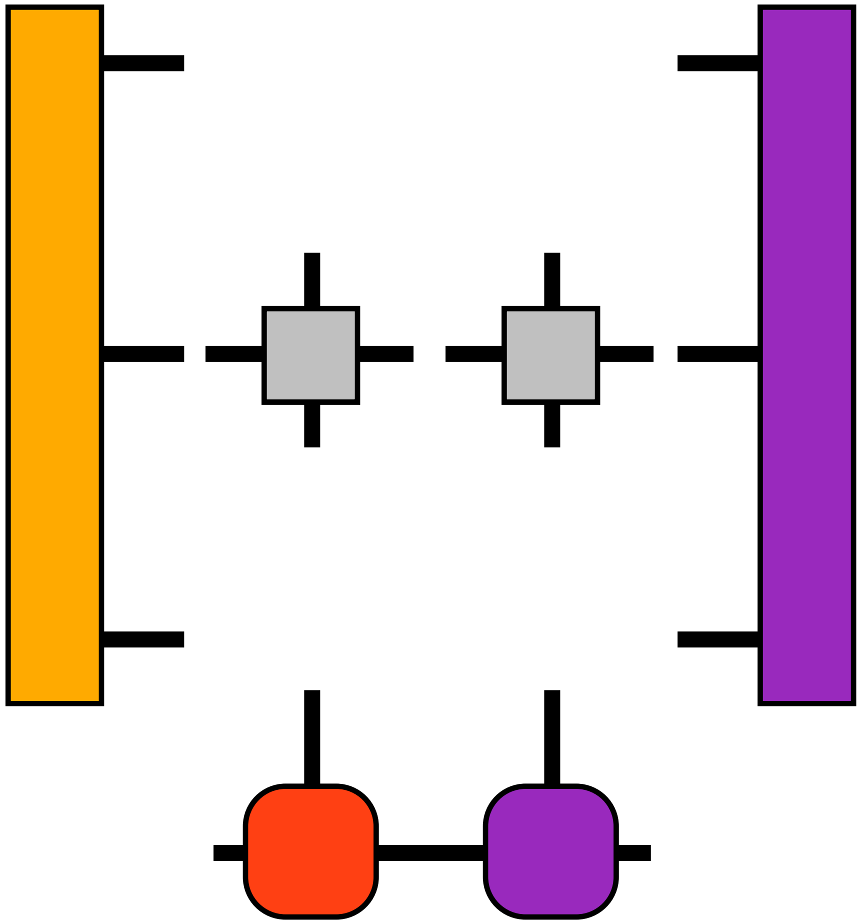


Grow projected Hamiltonian

Recover older projected Hamiltonian saved in memory

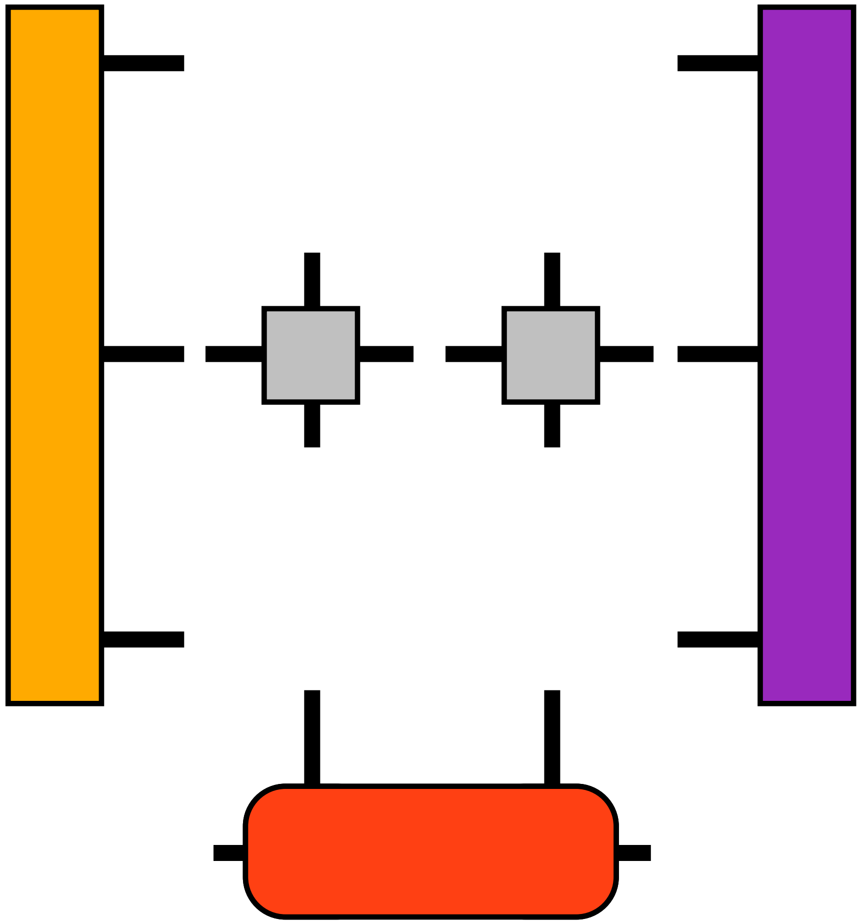


Iterating leads to sweeping procedure



Iterating leads to sweeping procedure

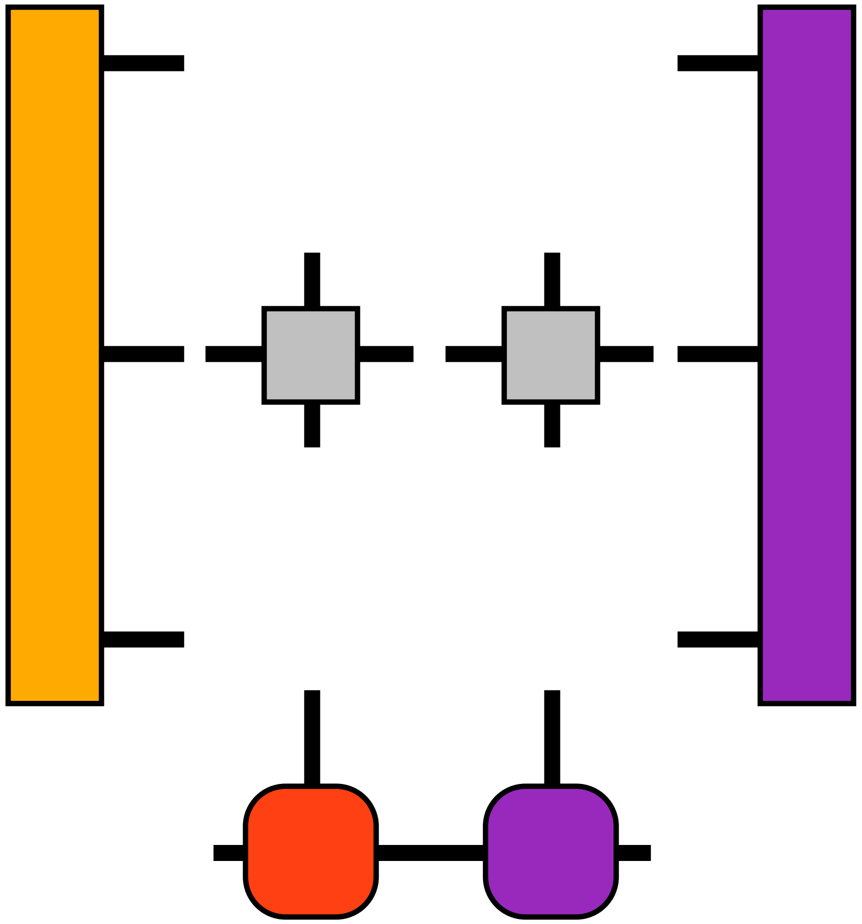
1. Solve eigenproblem



Iterating leads to sweeping procedure

I. Solve eigenproblem

II. SVD wavefunction

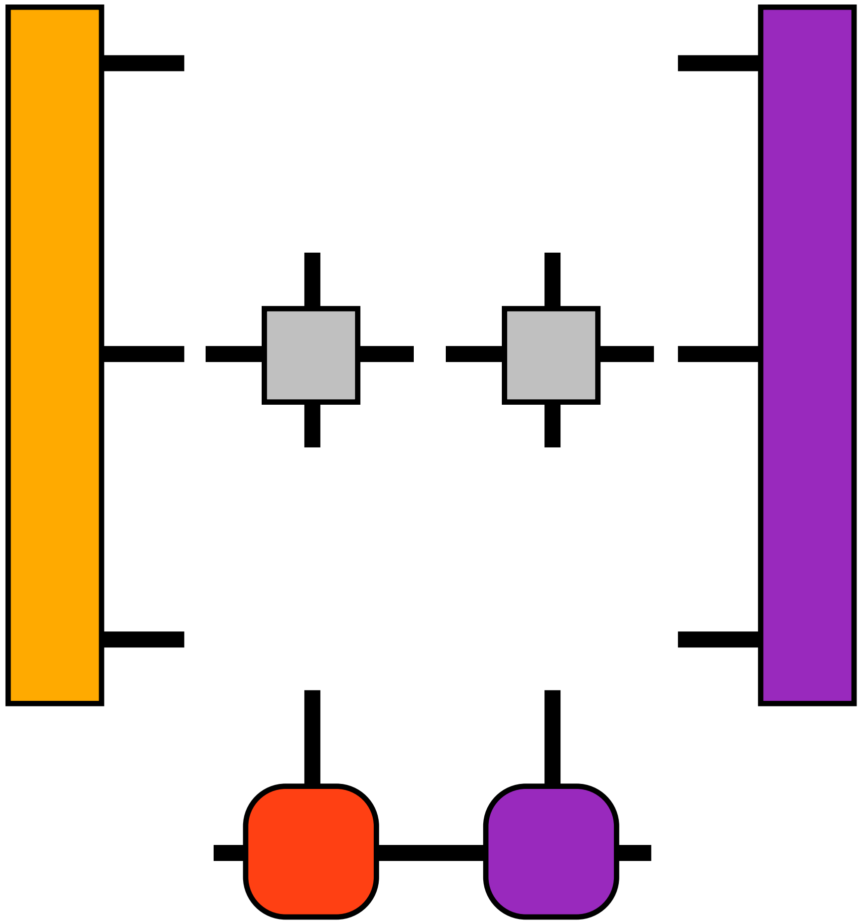


Iterating leads to sweeping procedure

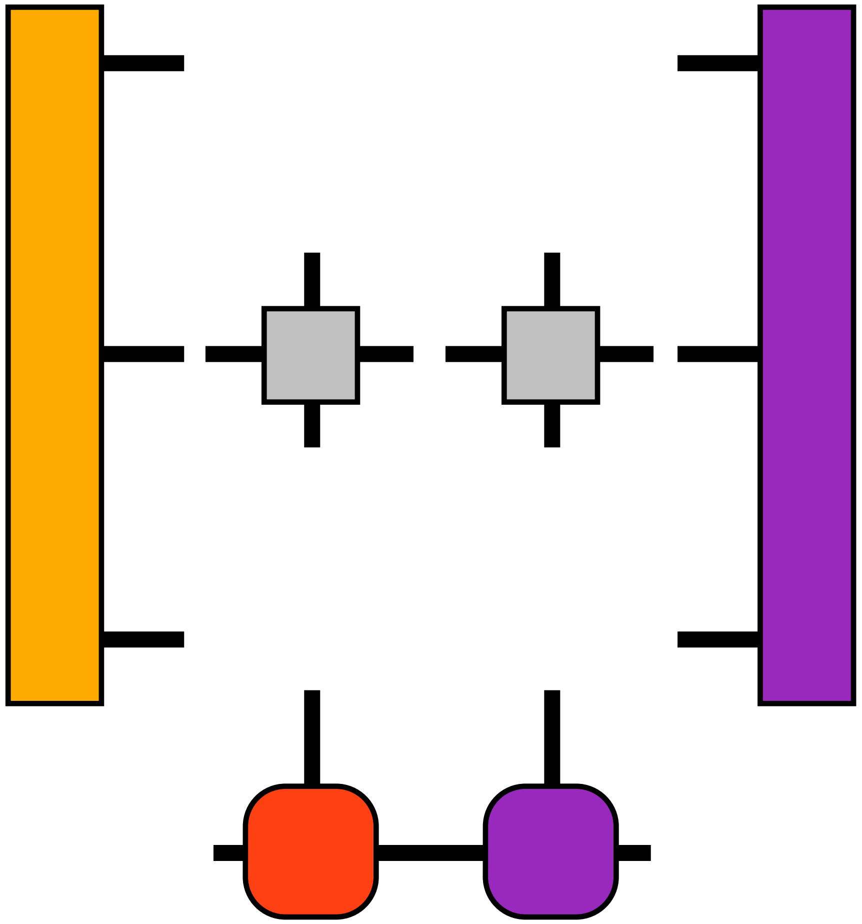
I. Solve eigenproblem

II. SVD wavefunction

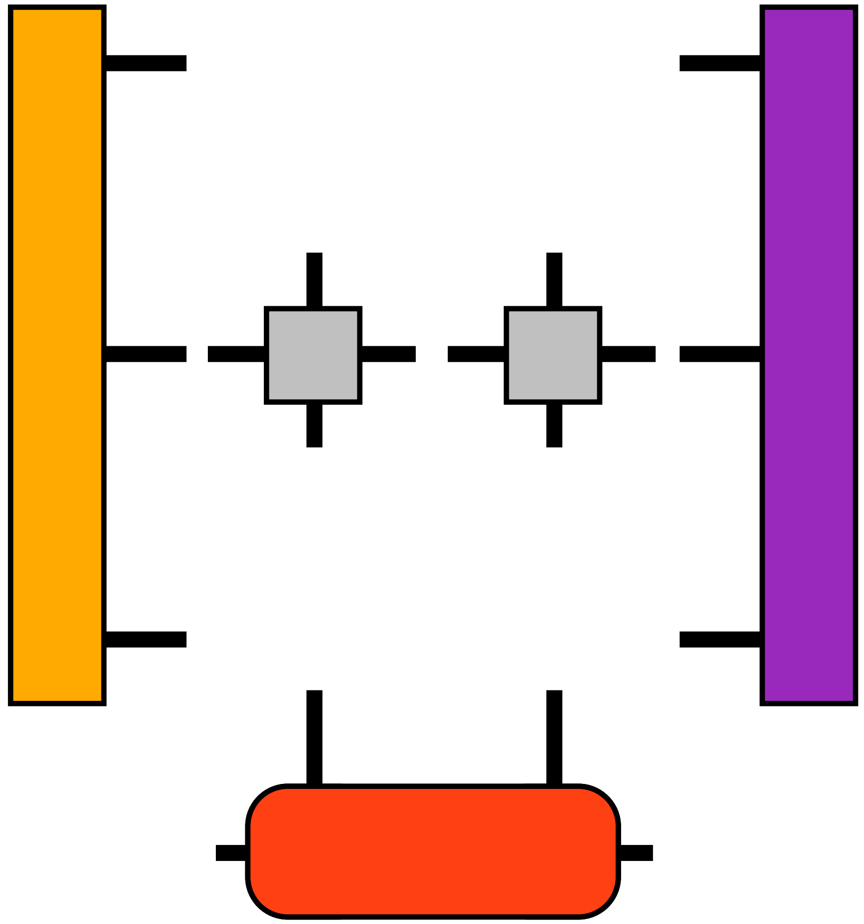
III. Grow effective H



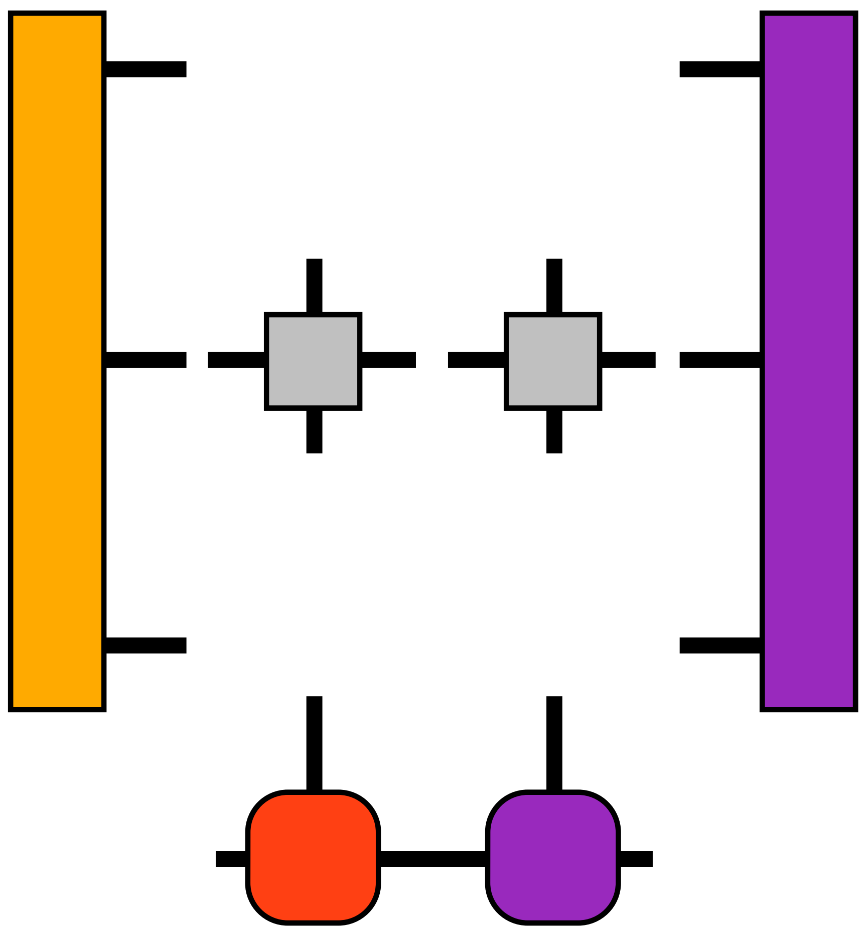
Iterating leads to sweeping procedure



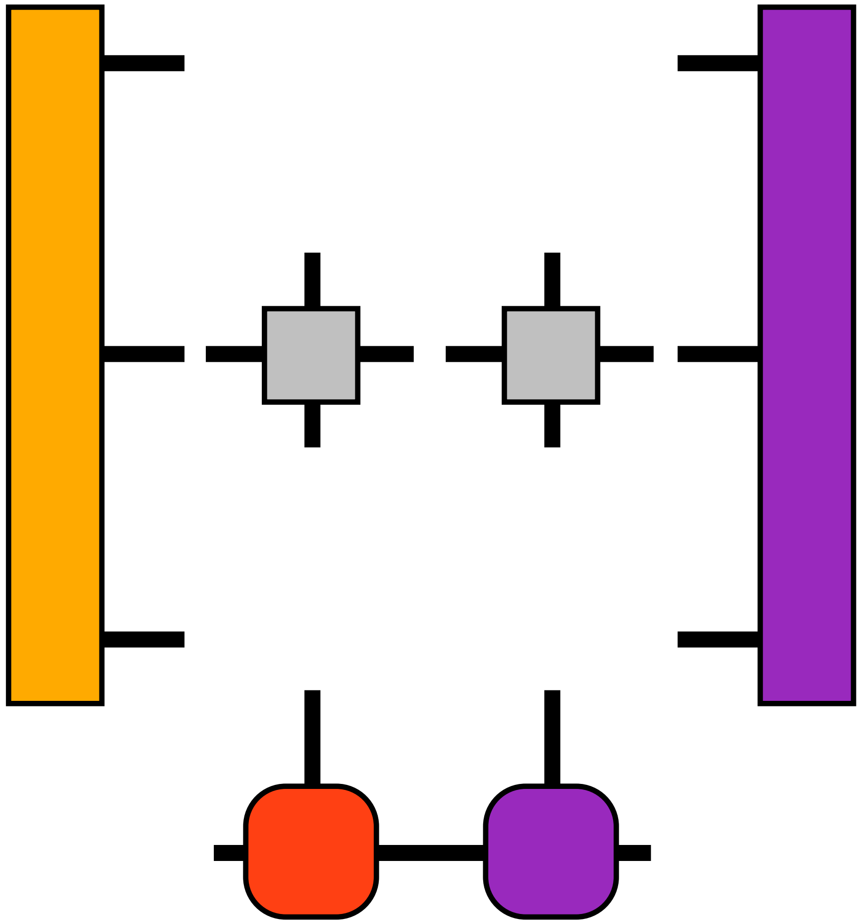
Iterating leads to sweeping procedure



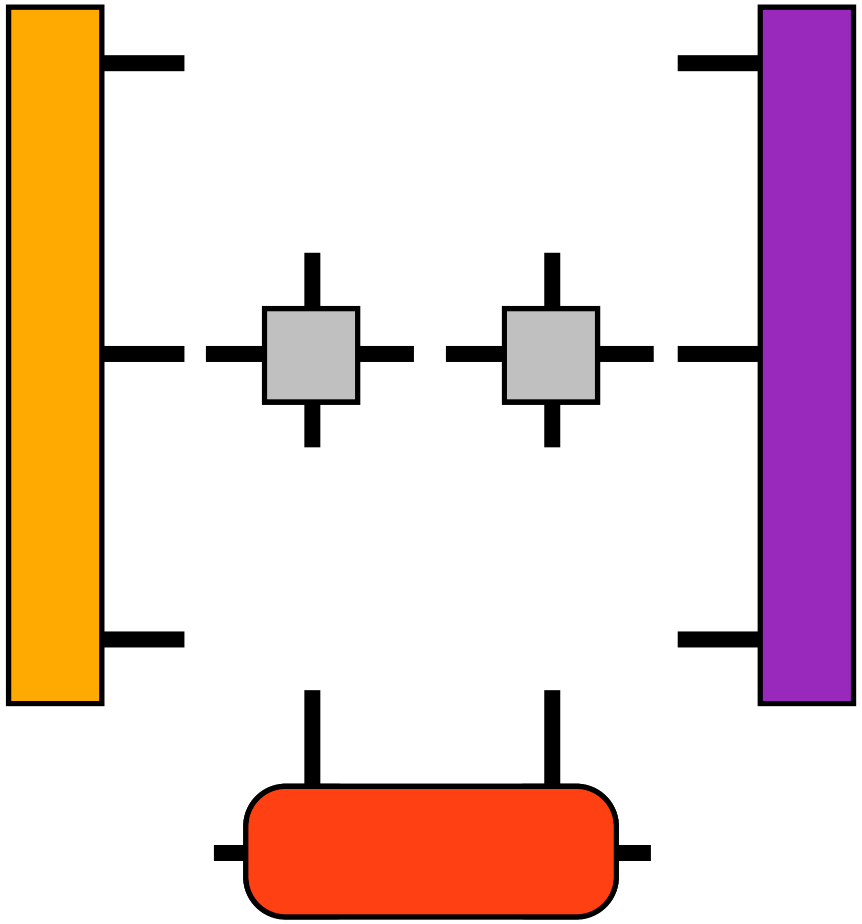
Iterating leads to sweeping procedure



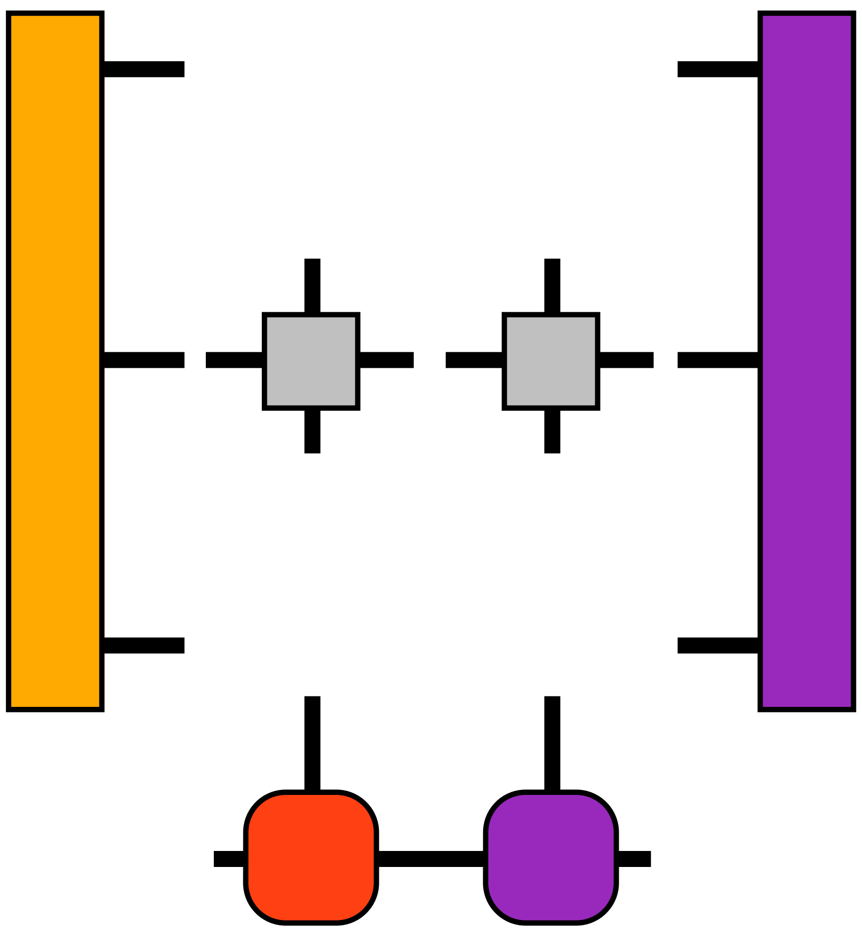
Iterating leads to sweeping procedure



Iterating leads to sweeping procedure



Iterating leads to sweeping procedure



We'll implement a key missing step of the DMRG algorithm

`<library folder>/tutorial/06_DMRG`

1. Read through **`dmrg.cc`**; compile; and run

2. SVD the two-site tensor phi into factors A, D, B
   The last argument to svd should be "opts" to pass
   through parameters controlling truncation:

```
svd(phi, ... , opts);
```

3. Multiply the singular-value tensor D back into A or B
   as appropriate to shift orthogonality center of MPS.

4. Add code to print out the energy at each step (or even
   to measure other local operators).