Hands-on with

ITensor

http://itensor.org

ITensor

http://itensor.org

C++ library for tensor networks

Tensors, matrix product states (MPS), DMRG

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

Much more than a "black box" for DMRG

ITensor website http://itensor.org/

- Home
- News
- Learn
- Discuss
- Contributors

ITENSOR

Introduction

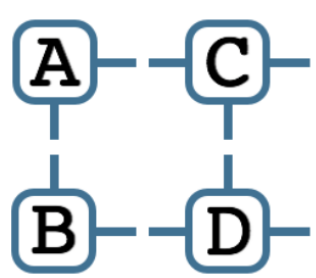
ITensor—Intelligent Tensor—is a C++ library for implementing tensor product wavefunction calculations. It is efficient and flexible enough to be used for research-grade simulations.

Features include:

- Named indices; no need to think about index ordering
- Full-featured matrix product state and DMRG layer
- Quantum number conserving (block-sparse) tensors; same interface as dense tensors
- Complex numbers handled lazily: no efficiency loss if real
- Easy to install; only dependencies are BLAS/LAPACK and C++11

ITensors have an Einstein summation interface making them nearly as easy to multiply as scalars: tensors indices have unique identities and matching indices automatically contract when two ITensors are multiplied. This type of interface makes it simple to transcribe tensor network diagrams into correct, efficient code.

For example, the diagram below (resembling the overlap of matrix product states) can be converted to code as



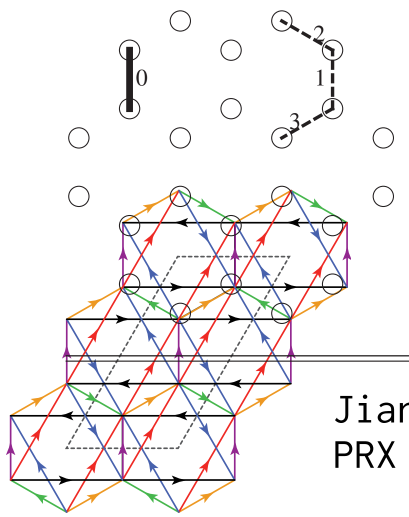
= A*B*C*D

Latest version is v2.0.8
Clone from github (preferred)
Download: tar.gz, zip
Report bugs: code
website
Follow: @ITensorLib

Recent News

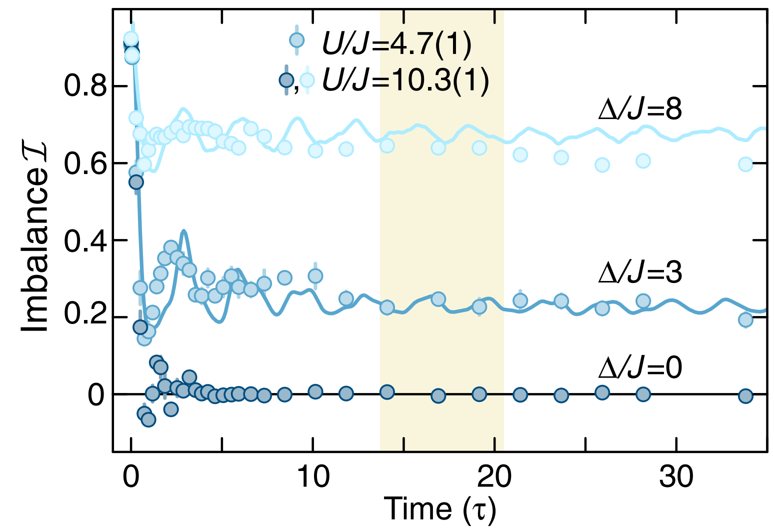
- Tutorial on Fermions and Jordan-Wigner Mapping
- New Discussion Forum
- Version 2.0 released!
- ITensor at 2016 Sherbrooke Summer School

Selected applications of ITensor:

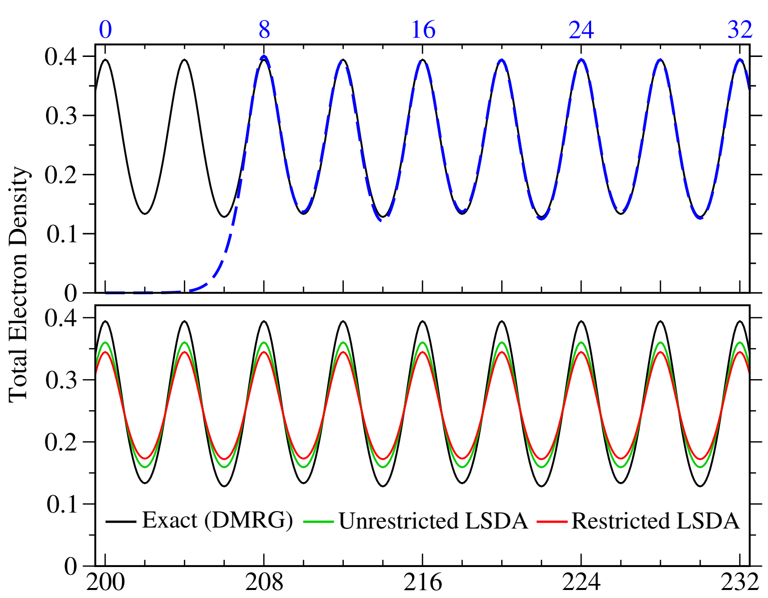


J/t	$\arg(\Delta_{01})/2\pi$	$\arg(\Delta_{02})/2\pi$
2.0	−0.0025(9)	0.327(1)
	−0.022(5)	0.313(6)
1.5	0.001(1)	0.322(2)
	0.000(7)	0.311(7)
1.0	0.001(2)	0.296(2)
	0.000(7)	0.305(8)
0.78	−0.001(2)	0.289(2)
	0.003(7)	0.304(9)
0.5	−0.005(3)	0.245(3)
	0.02(2)	0.28(1)
0.2	−0.008(4)	0.119(6)
0.1	−0.011(4)	0.065(5)

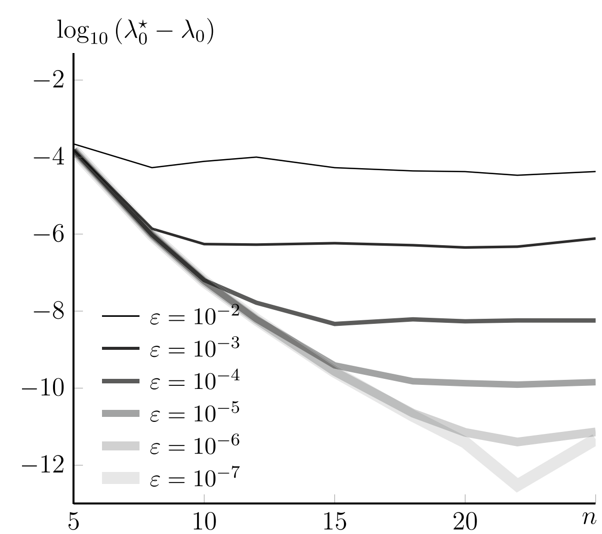
Jiang, Mesaros, Ran,
PRX 4, 031040 [2014]



Schreiber et al.,
“Observation of many-body localization...”,
arxiv:1501.05661 [2015]



Stoudenmire, Wagner, White, Burke,
PRL 109, 056402 [2012]



Dolgov, Khoromskij, Oseledets, Savostyanov
CPC 185, 1207 [2014]

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 is ",r); //prints “r is 2.3456”
```

C++

User defined types (objects)

Construct an object of type MyClass:

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

Objects can have methods:

```
m.setValue(6);

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

C++

Some objects can be called like functions:

```
auto f = FType();

int j = f(5);
```

Other objects behave like numbers:

```
auto x = Numerical(1.);
auto y = Numerical(2.);

auto r = x + y;

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

Structure of an ITensor C++ program:

```
#include “itensor/all.h”

using namespace itensor;

int main()
{

//
// Code will go here
//

return 0;
}
```

01 Single Site Wavefunction

Consider a single-site wavefunction,
for example a spin 1/2

Single-site basis:

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

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

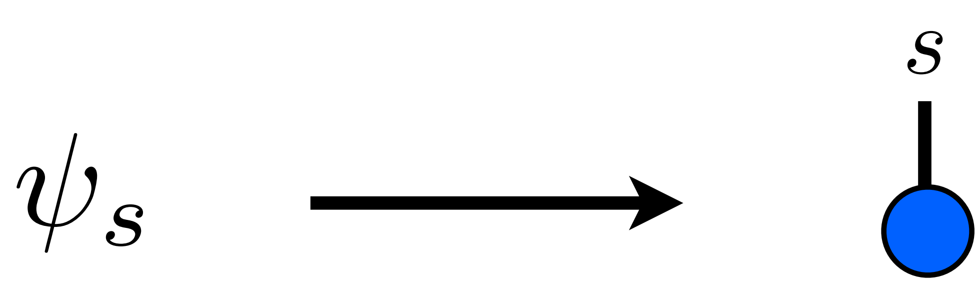
Most general wavefunction for a spin 1/2:

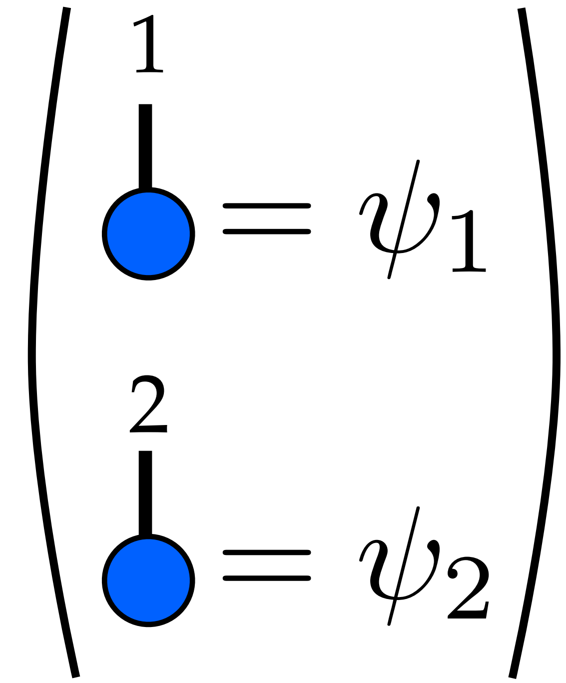
$$|\psi\rangle = \sum_{s=1}^{2} \psi_s |s\rangle$$

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:

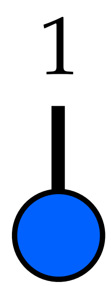




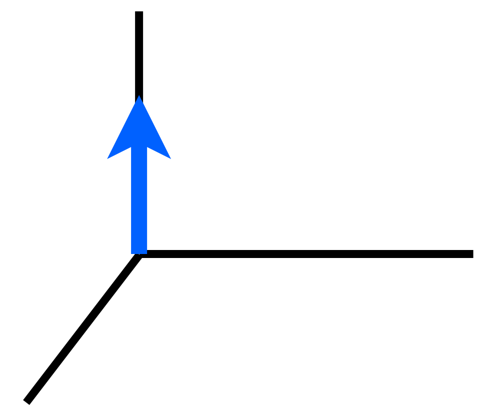
Using ITensor:

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

auto psi = ITensor(s);
```

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

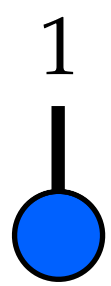


$= 1$

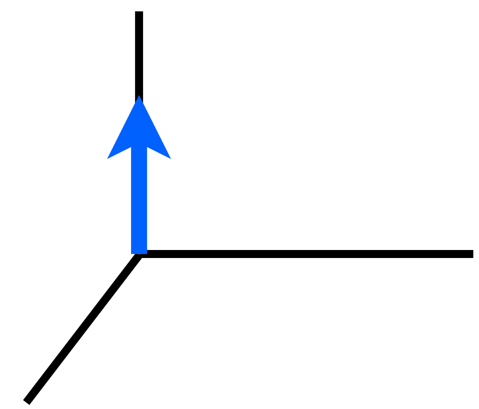


```
auto s = Index("s",2);
auto psi = ITensor(s);

psi.set(s(1), 1.0);

PrintData(psi);
```

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



$= 1$

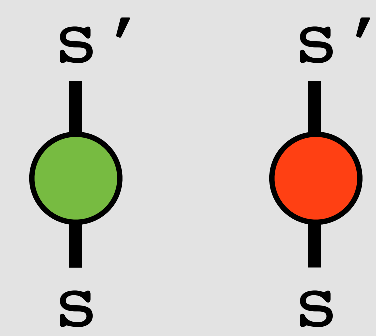


```
auto s = Index("s",2);
auto psi = ITensor(s);

psi.set(s(1), 1.0);

PrintData(psi);
```

```
psi =
ITensor r=1: (s,2,Link,273)
(1) 1.00
```

Make some operators:

```
auto Sz = ITensor(s,prime(s));

auto Sx = ITensor(s,prime(s));
```



New ITensors start out set to zero

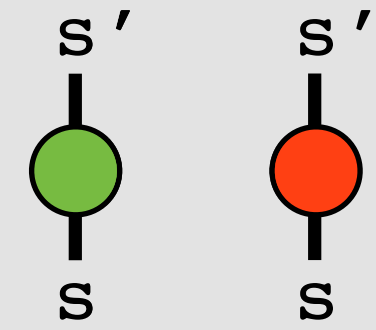
What does "prime" do?

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

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

Our operators:

```
auto Sz = ITensor(s,prime(s));

auto Sx = ITensor(s,prime(s));
```

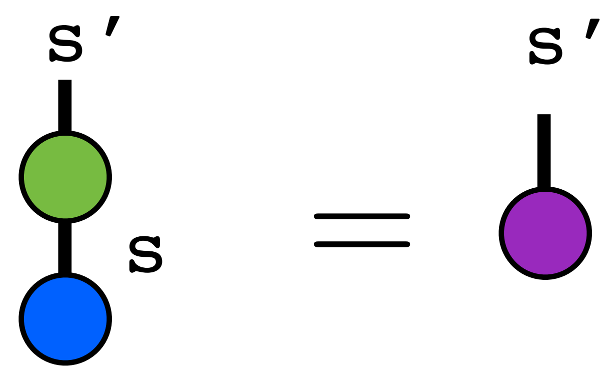


Set their components:

```
Sz.set(s(1),prime(s)(1), +0.5);
Sz.set(s(2),prime(s)(2), -0.5);


Sx.set(s(1),prime(s)(2), +0.5);
Sx.set(s(2),prime(s)(1), +0.5);
```

Let's compute $\hat{S}_x|\psi\rangle = |\phi\rangle$

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



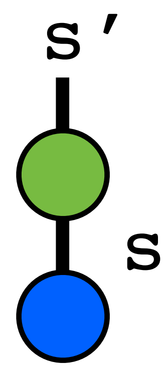
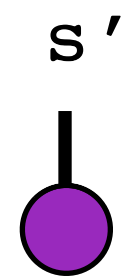
In code,

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

* operator contracts matching indices.

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

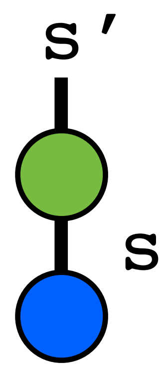
What state is phi ?

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

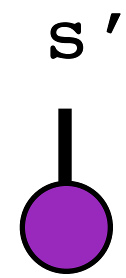

```
ITensor phi = Sx * psi;

PrintData(phi);
```

What state is phi ?

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



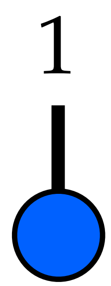
$$=$$



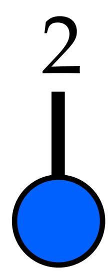

```
ITensor phi = Sx * psi;

PrintData(phi);
```

```
phi =
ITensor r=1: (s,2,Link,273)'
(2) 0.500
```

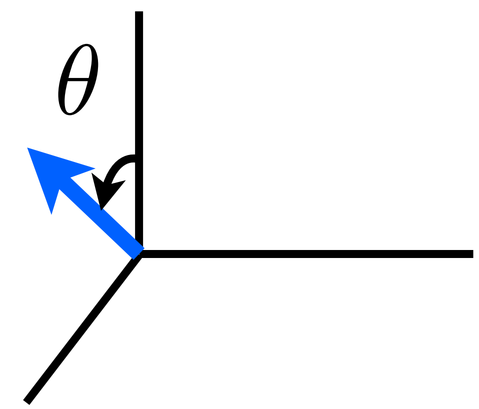
More interesting ψ_s : choose $\theta = \pi/4$ and



$= \cos\theta/2$

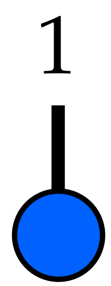


$= \sin\theta/2$

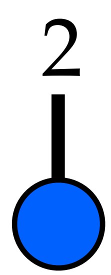



```
Real theta = Pi/4.;

psi.set(s(1),cos(theta/2));
psi.set(s(2),sin(theta/2));

PrintData(psi);
```

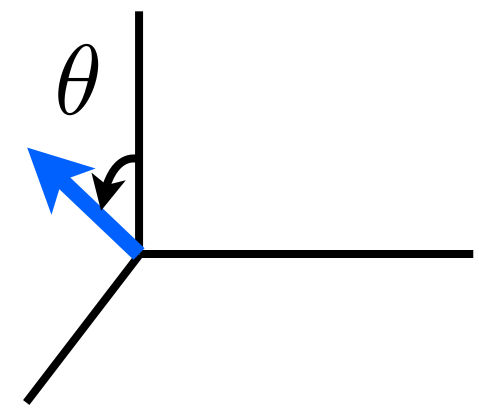
More interesting ψ_s : choose $\theta = \pi/4$ and



$= \cos\theta/2$



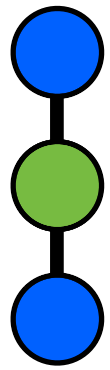
$= \sin\theta/2$




```
Real theta = Pi/4.;

psi.set(s(1),cos(theta
psi.set(s(2),sin(theta

PrintData(psi);
```

```
psi =
ITensor r=1: (s,2,Link,273)
(1) 0.92388
(2) 0.38268
```

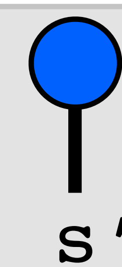
Diagrammatically, measurements (expectation values) look like:



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

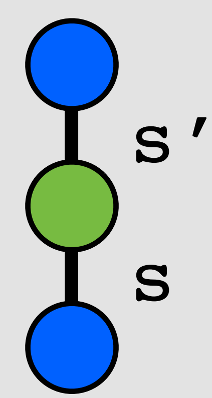
For convenience, make:

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



Calculate expectation values:

```
auto zz = (cpsi * Sz * psi).real();

auto xx = (cpsi * Sx * psi).real();
```



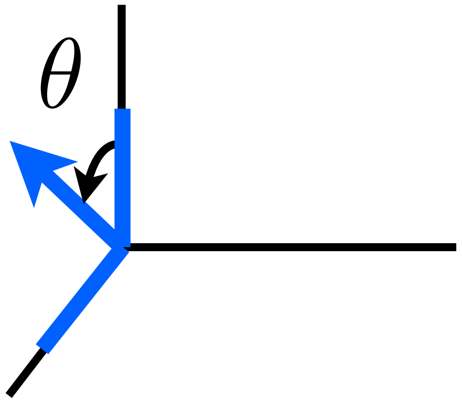
```
auto zz = (cpsi * Sz * psi).real();

auto xx = (cpsi * Sx * psi).real();
```

Printing the results,

```
println("<Sz> = ",zz);

println("<Sx> = ",xx);
```

we get the output

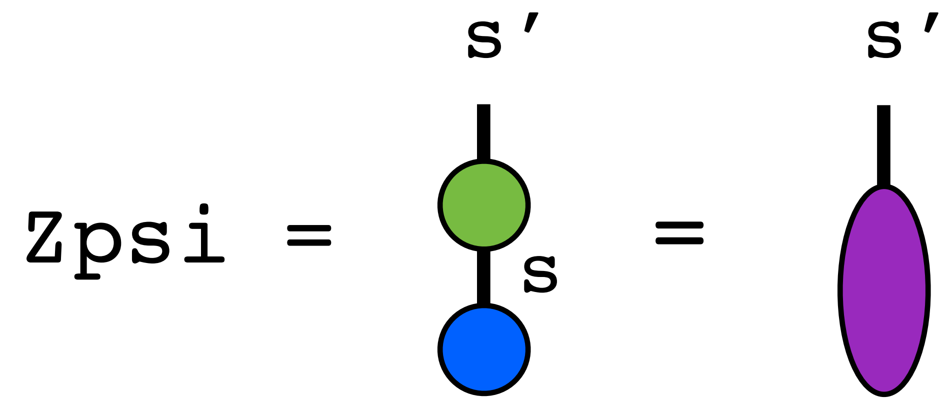
```
<Sz> = 0.35355
<Sx> = 0.35355
```



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

More slowly:

```
auto Zpsi = Sz * psi;
```

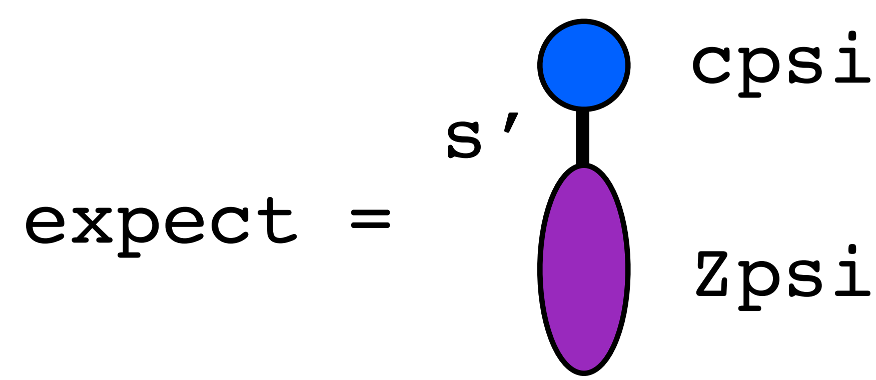


Index s matches, so it's automatically contracted.

`Zpsi` and `cpsi` share Index s'

* contracts it, leaving a scalar ITensor

```
auto expect = cpsi * Zpsi;

auto zz = expect.real();
```



Review:

- Construct an Index

```
auto a = Index("index a",4);
```

- Construct ITensor (indices a, b, c)

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

- Set ITensor components

```
T.set(a(2),c(3),b(1), 7.89);
```

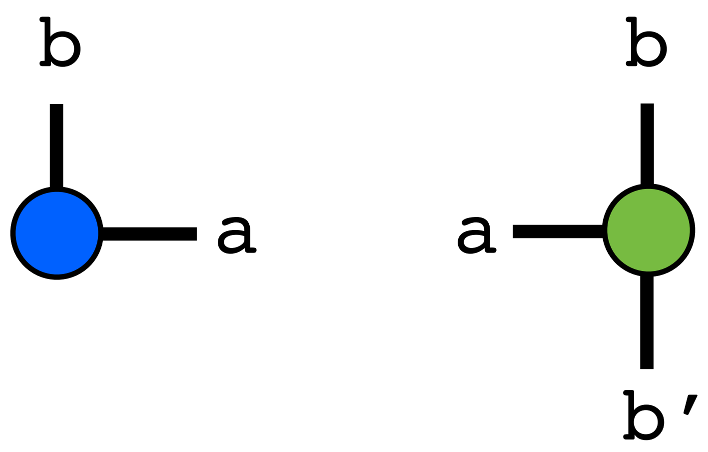
- Prime an Index b $\longrightarrow$ b′

```
prime(b)
```

- The `*` operator automatically contracts matching Index pairs

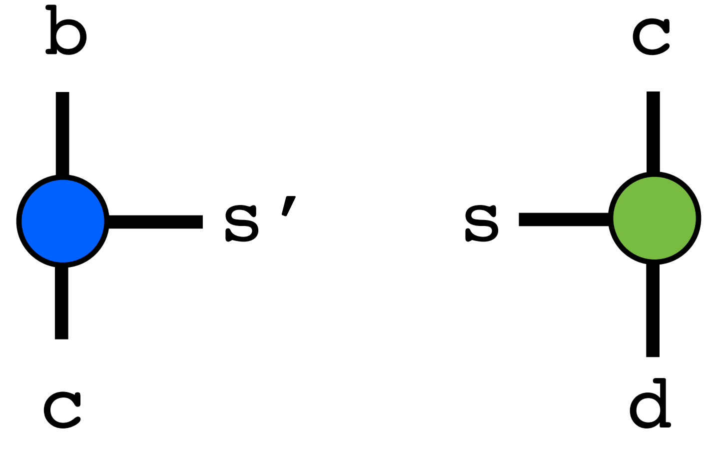
Quiz:

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



Quiz:

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



Code hands-on session:

`itensor_tutorial/01_one_site`

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

2. Change psi (line 22) to be an eigenstate of S_x

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle)$$

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

```
auto olap = (dag(psi)*phi).real();
```

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

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

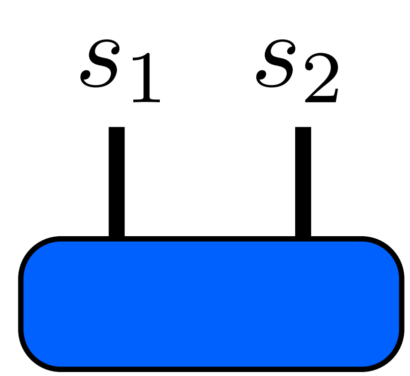
02 Two Site Wavefunction

Most general two-site wavefunction is

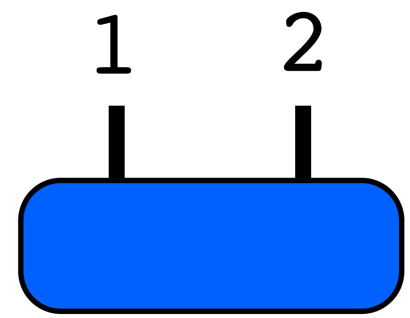
$$|\Psi\rangle = \sum_{s_1, s_2 = 1}^{2} \psi_{s_1 s_2} |s_1\rangle |s_2\rangle$$

Amplitudes are a rank-2 tensor

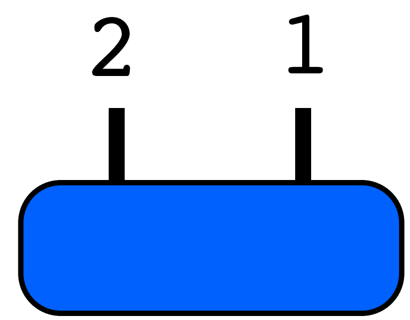
$$\psi_{s_1 s_2} =$$



Let's make a singlet



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



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

Using ITensor:

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

auto psi = ITensor(s1,s2);

psi.set(s1(1),s2(2),+1./sqrt(2));
psi.set(s1(2),s2(1),-1./sqrt(2));
```

Why `Site` tag in Index constructor?

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

Index objects can have an optional `"IndexType"` tag

Useful for priming just one type of Index, for example

Default type is `Link`, physical indices of type `Site`

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

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

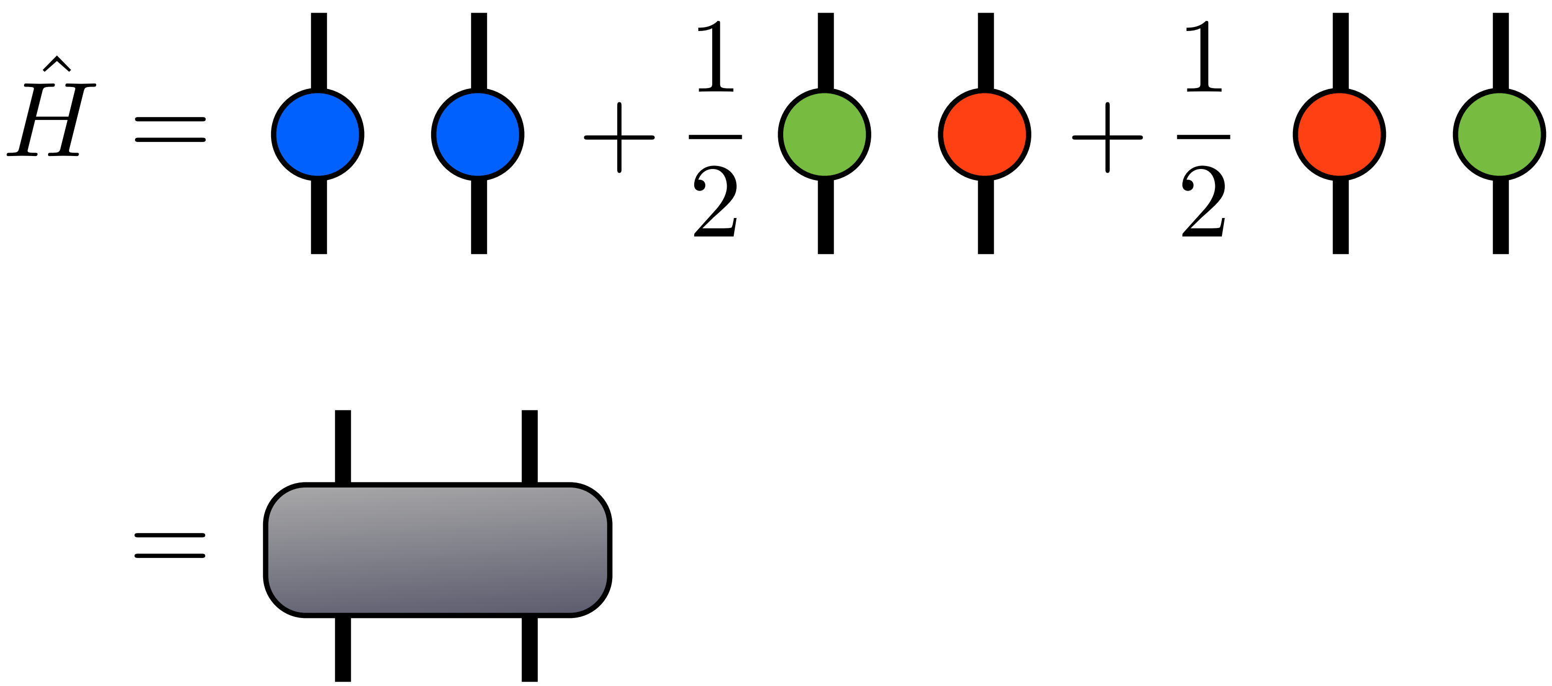
First create operators, for example S^+

```
auto Sp1 = ITensor(s1,prime(s1));
Sp1.set(s1(2),prime(s1)(1), 1);
```

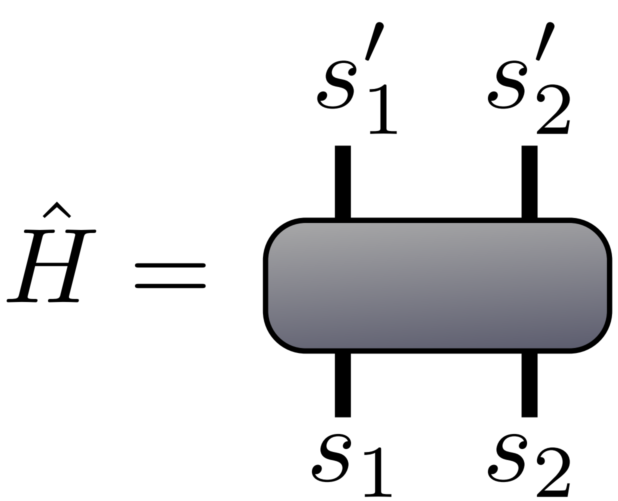
Multiply and add operators to make H:

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

Tensor form of H

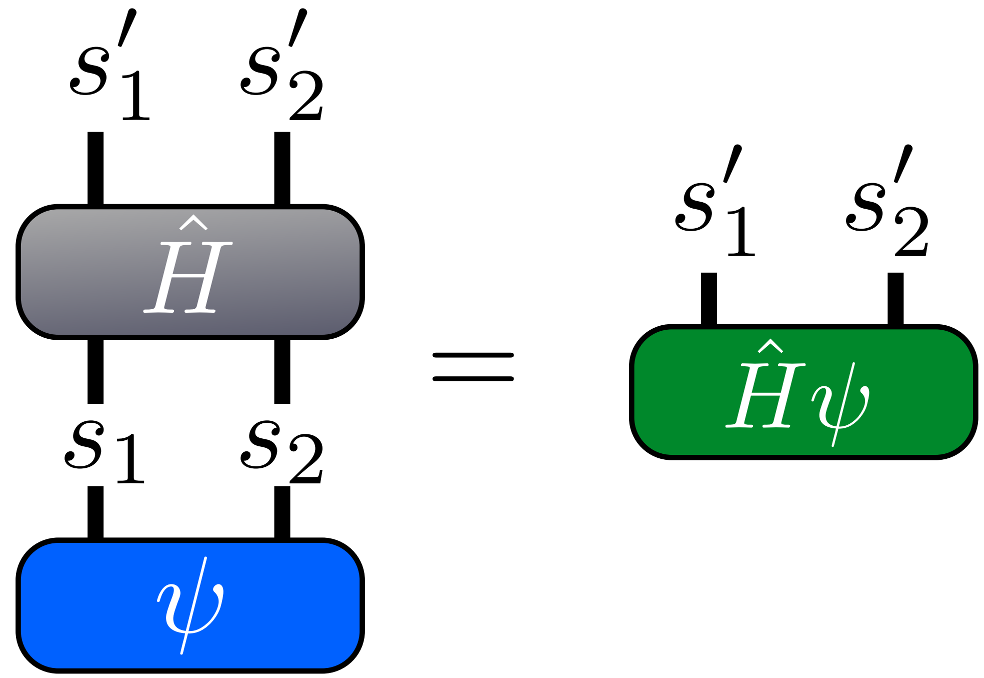


Showing Index labels



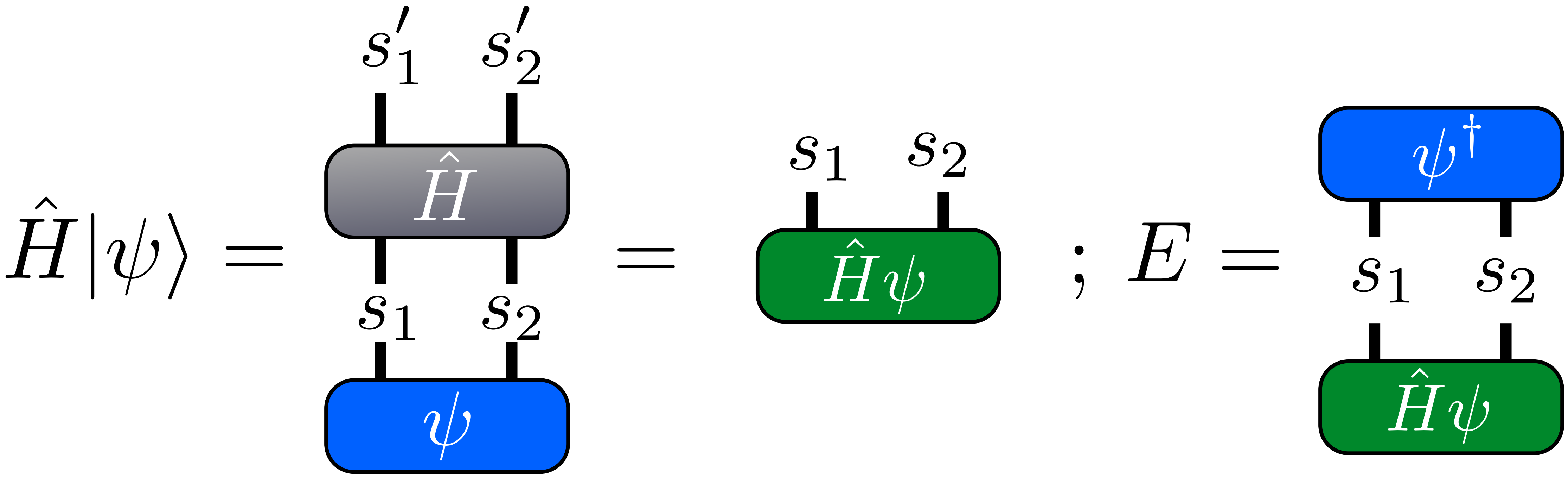
Compute singlet energy with this Hamiltonian:

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




```
auto Hpsi = H * psi;
Hpsi.mapprime(1,0);

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

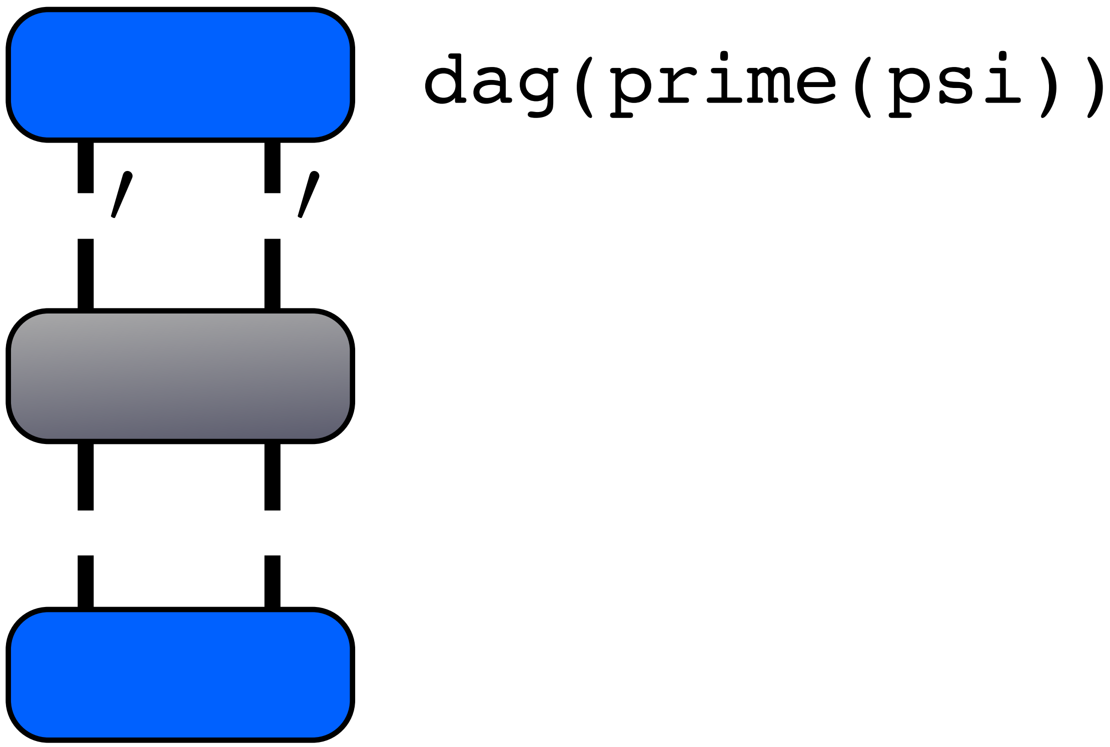
Compute singlet energy with this Hamiltonian:




```
auto Hpsi = H * psi;
Hpsi.mapprime(1,0);

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

Or compute energy in one shot:

$$E_{\text{sing}} =$$




```
Real E = (dag(prime(psi)) * H * psi).real();

Print(E);
//prints: E = -0.75
```

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

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

```
itensor_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 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 * bH;
    expH /= ord;
    expH.mapprime(2,1);
    expH = expH + Id;
    }
```

03 SVD

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

Matrix product states arise from compressing a one-dimensional wavefunction using 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

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

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

Matrices A and B are "isometries":

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

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

D diagonal

Elements of D can be chosen:

(1) Real

(2) Positive semi-definite

(3) 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:

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

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

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

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:

A D B

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

$$= M_3 =$$

Truncating SVD =

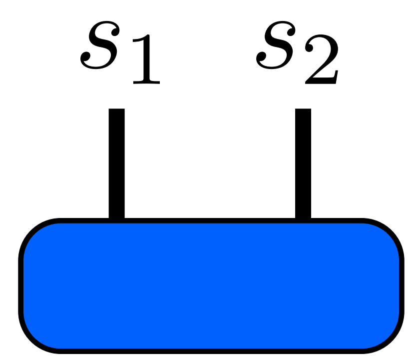
Controlled approximation for M

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

Recall:

Most general two-spin wavefunction

$$\psi_{s_1 s_2} =$$



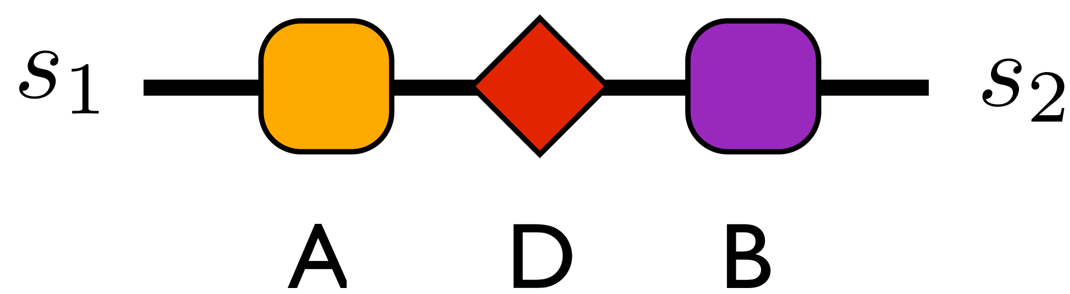
Can treat as a matrix:

$$\psi_{s_1 s_2} =$$

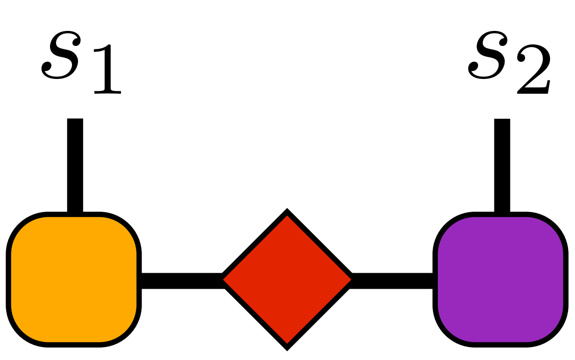
SVD this matrix:

$\psi_{s_1 s_2} =$

$=$

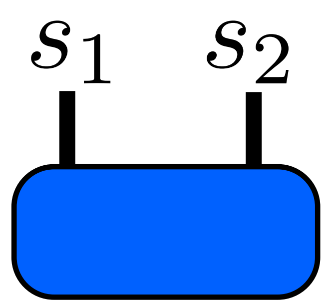


Bend lines back to look like wavefunction:



Using ITensor:

Say we have a two-site wavefunction psi

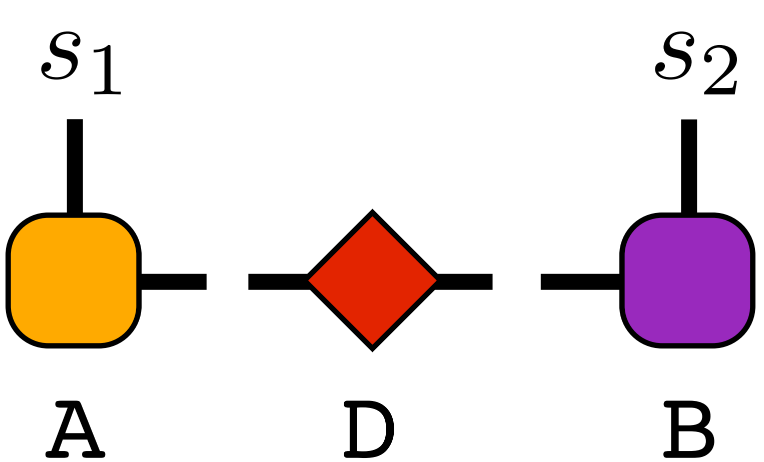


Declare A,D,B to hold results of SVD

```
auto A = ITensor(s1)
ITensor D,B;
```

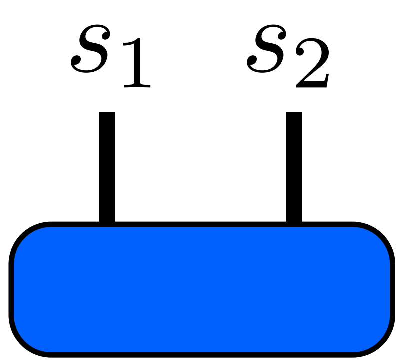
Call SVD function

```
svd(psi,A,D,B);
```



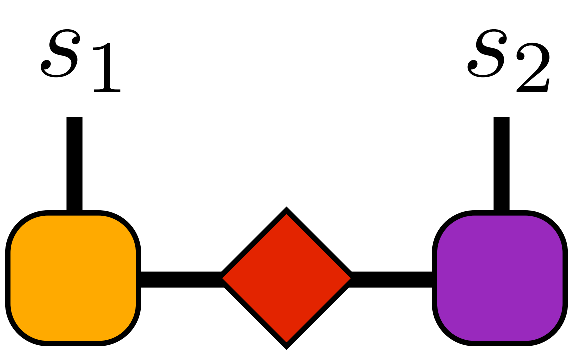
What have we gained from SVD?

Generic two-spin wavefunction (say spin S):



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

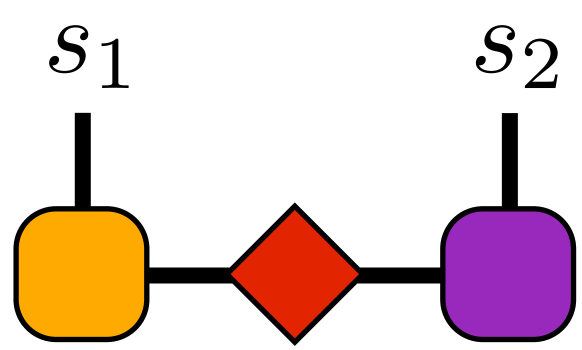
Compressed wavefunction:



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)

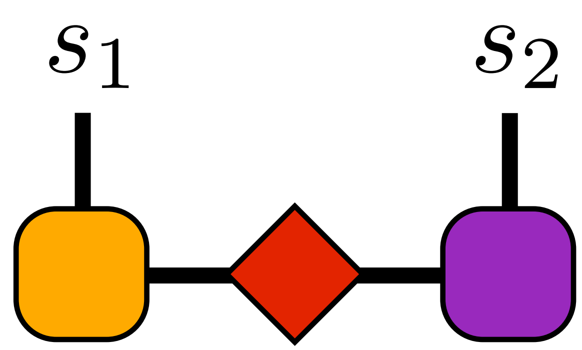


Why? Amplitude a product of matrices:

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

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

MPS have different equivalent forms, or "gauges"

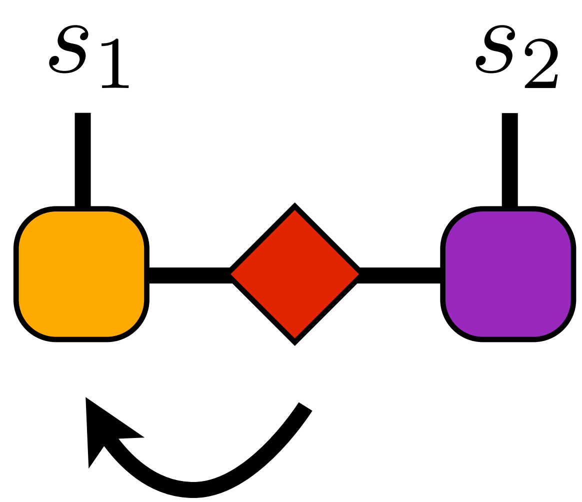


Canonical form

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

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

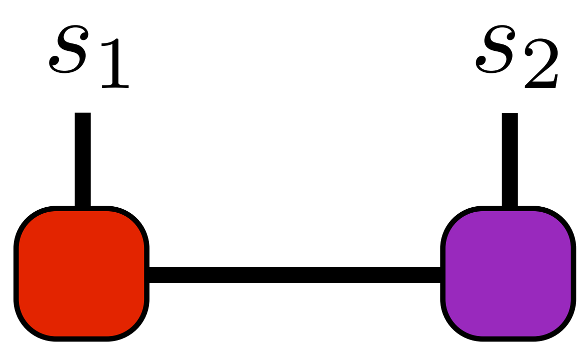
MPS have different equivalent forms, or “gauges”



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

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

MPS have different equivalent forms, or "gauges"

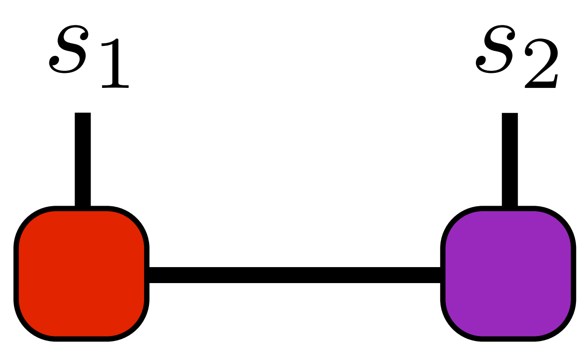


Left-canonical

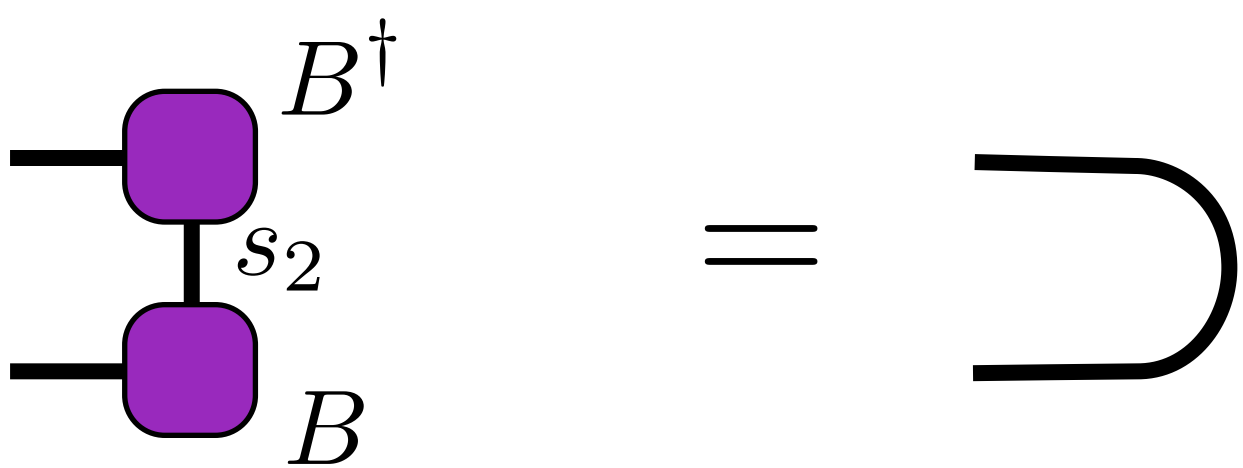
$$|\Psi\rangle = \sum_{s_1,\alpha',s_2} \psi_{s_1\alpha'} B_{\alpha' s_2} |s_1\rangle |s_2\rangle$$

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

MPS have different equivalent forms, or "gauges"



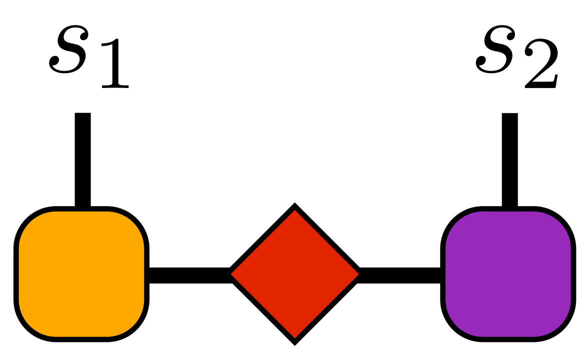
Matrix B is "right orthogonal" (from SVD)



$$BB^\dagger = I$$

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

MPS have different equivalent forms, or "gauges"

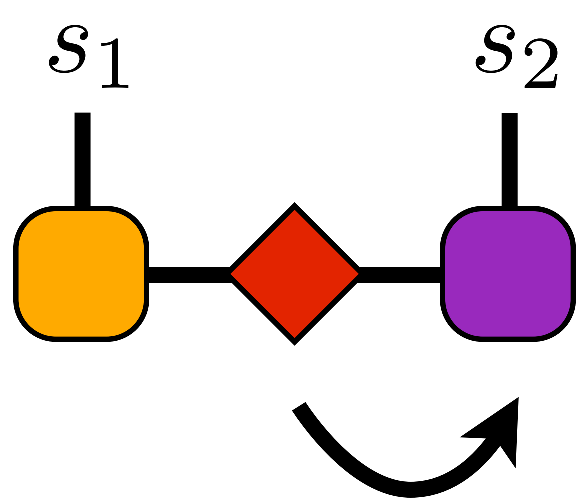


Canonical form

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

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

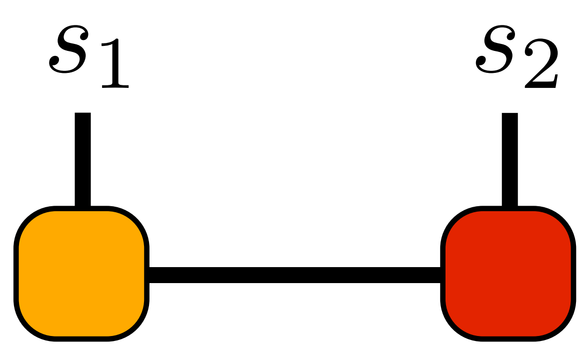
MPS have different equivalent forms, or "gauges"



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

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

MPS have different equivalent forms, or “gauges”

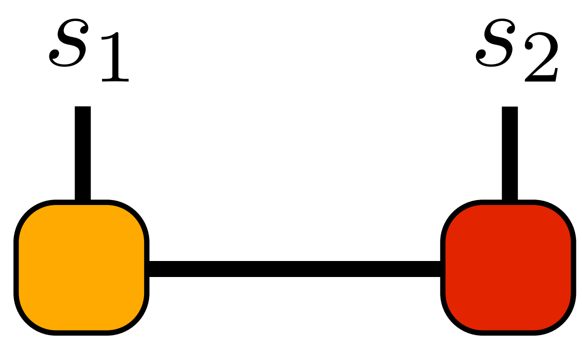


Right-canonical

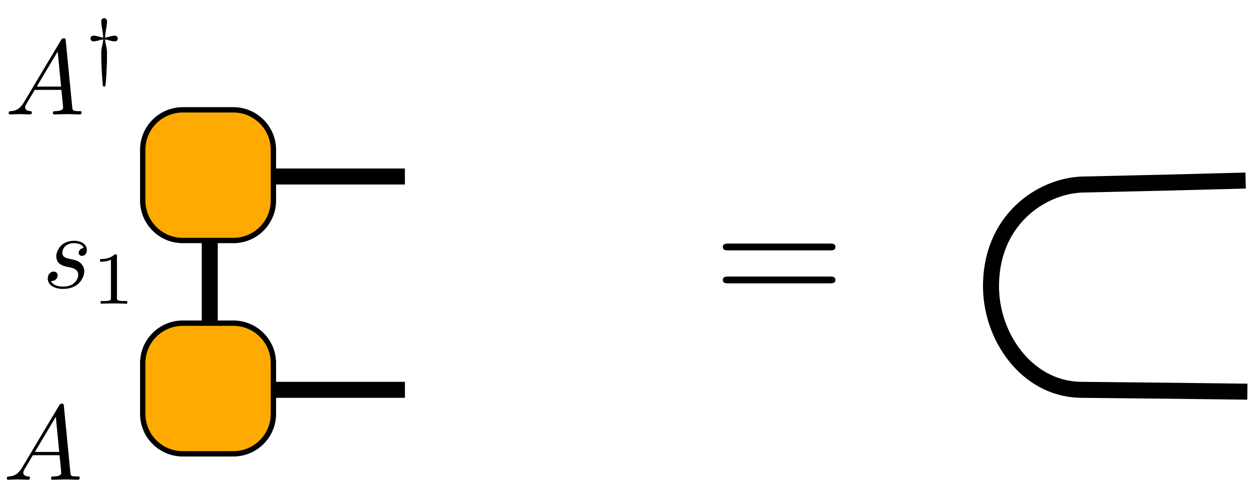
$$|\Psi\rangle = \sum_{s_1,\alpha,s_2} A_{s_1\alpha}\psi_{\alpha s_2}|s_1\rangle|s_2\rangle$$

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

```
itensor_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;
auto spectrum = svd(psi,A,D,B);
```

3. Compute the entanglement entropy using the density matrix spectrum returned by svd.

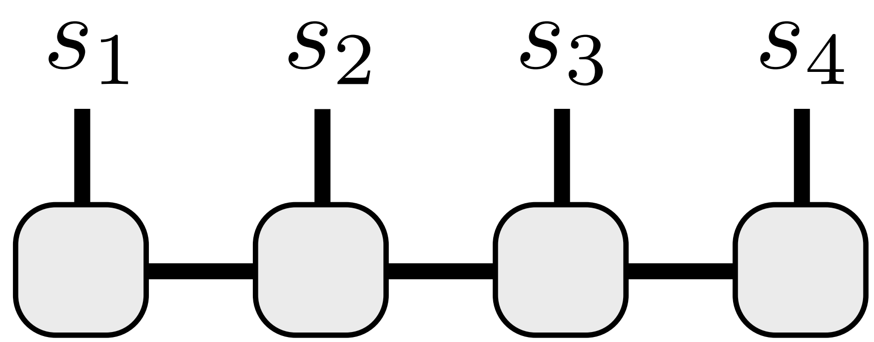
n^{th} eigenvalue:
number of eigenvalues:

```
spec.eig(n); //n=1,2,3,...
spec.size();
```

04 Four Sites

Say we have a 4-site MPS.

How efficiently can we compute properties?

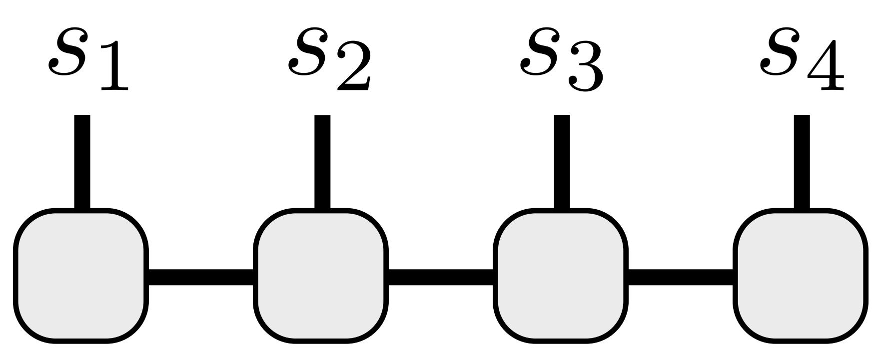


Depends on the gauge!

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

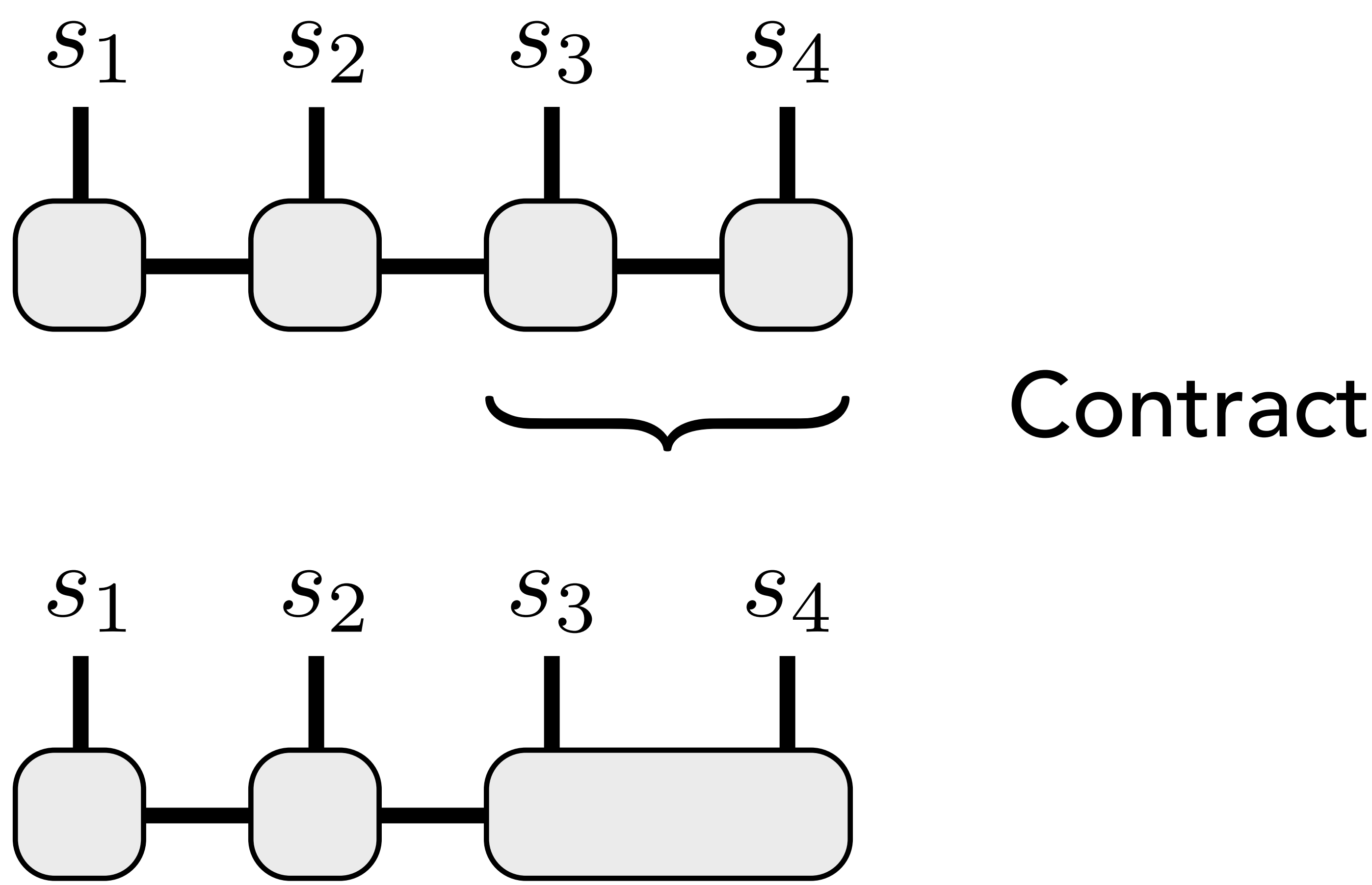
Assume we know nothing about the MPS

Put it in a useful gauge:



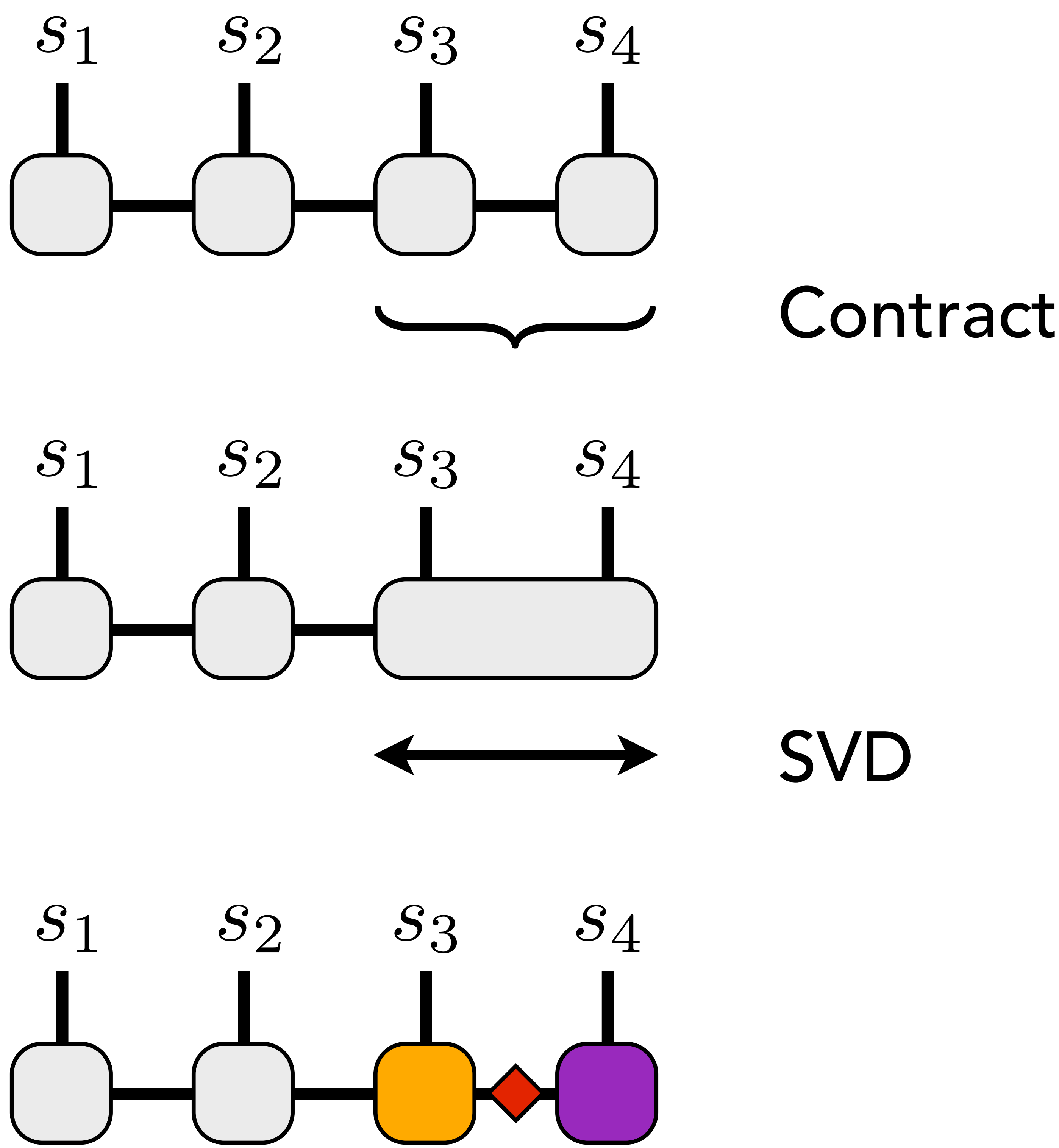
Assume we know nothing about the MPS

Put it in a useful gauge:



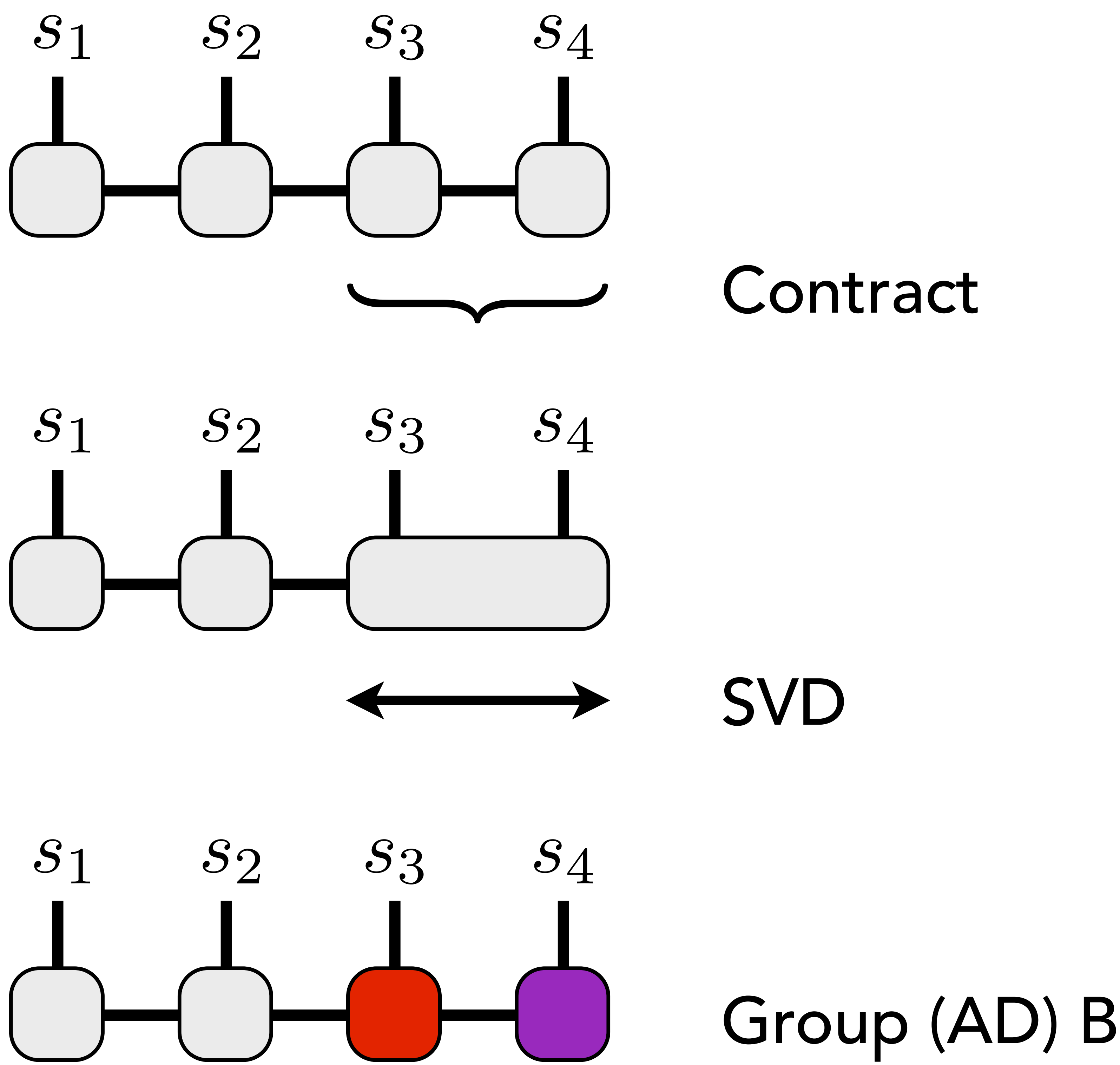
Assume we know nothing about the MPS

Put it in a useful gauge:

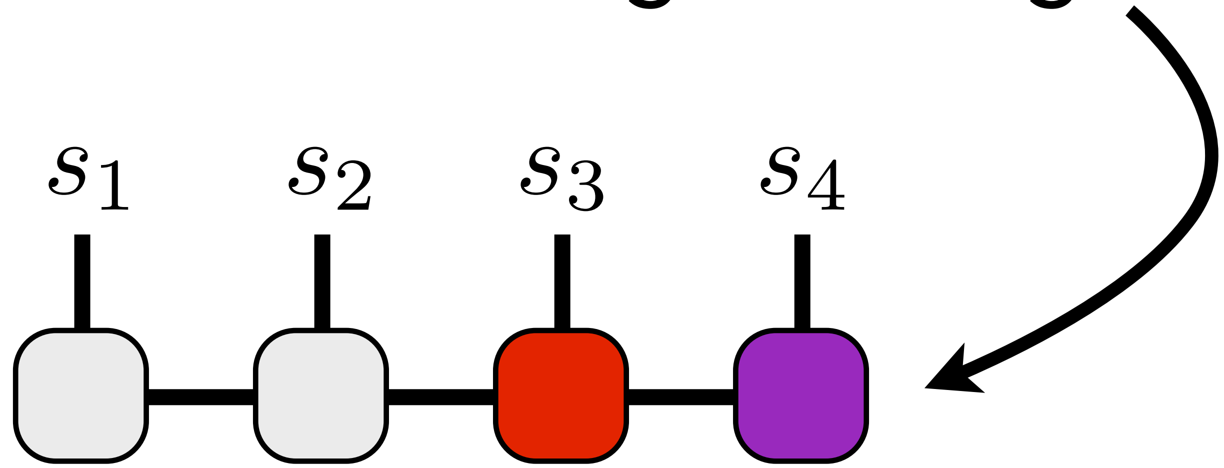


Assume we know nothing about the MPS

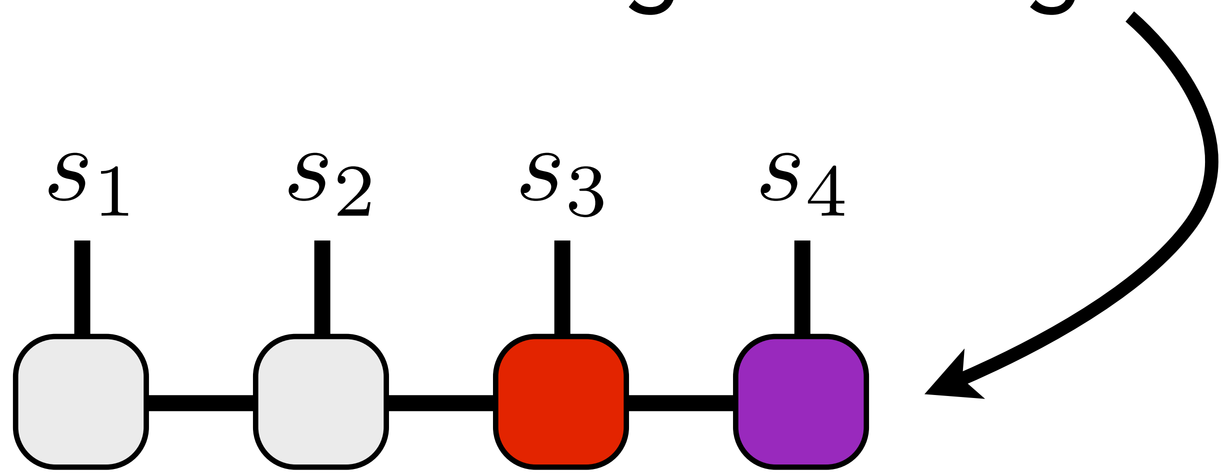
Put it in a useful gauge:



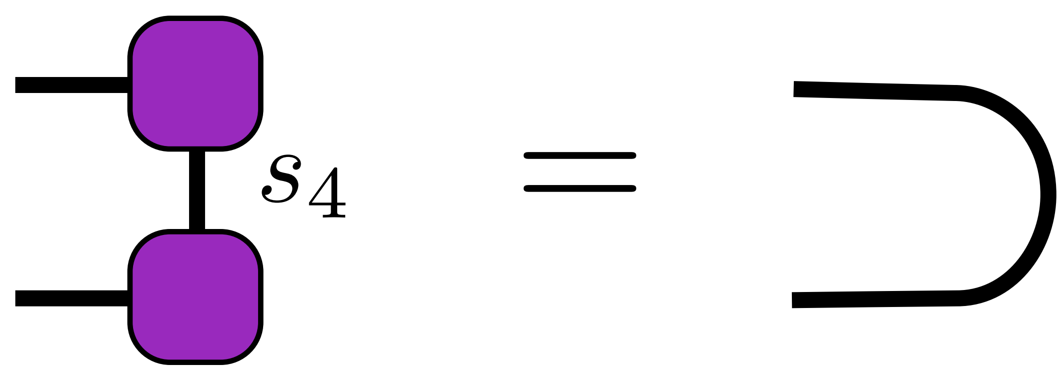
Note that site 4 tensor now right orthogonal



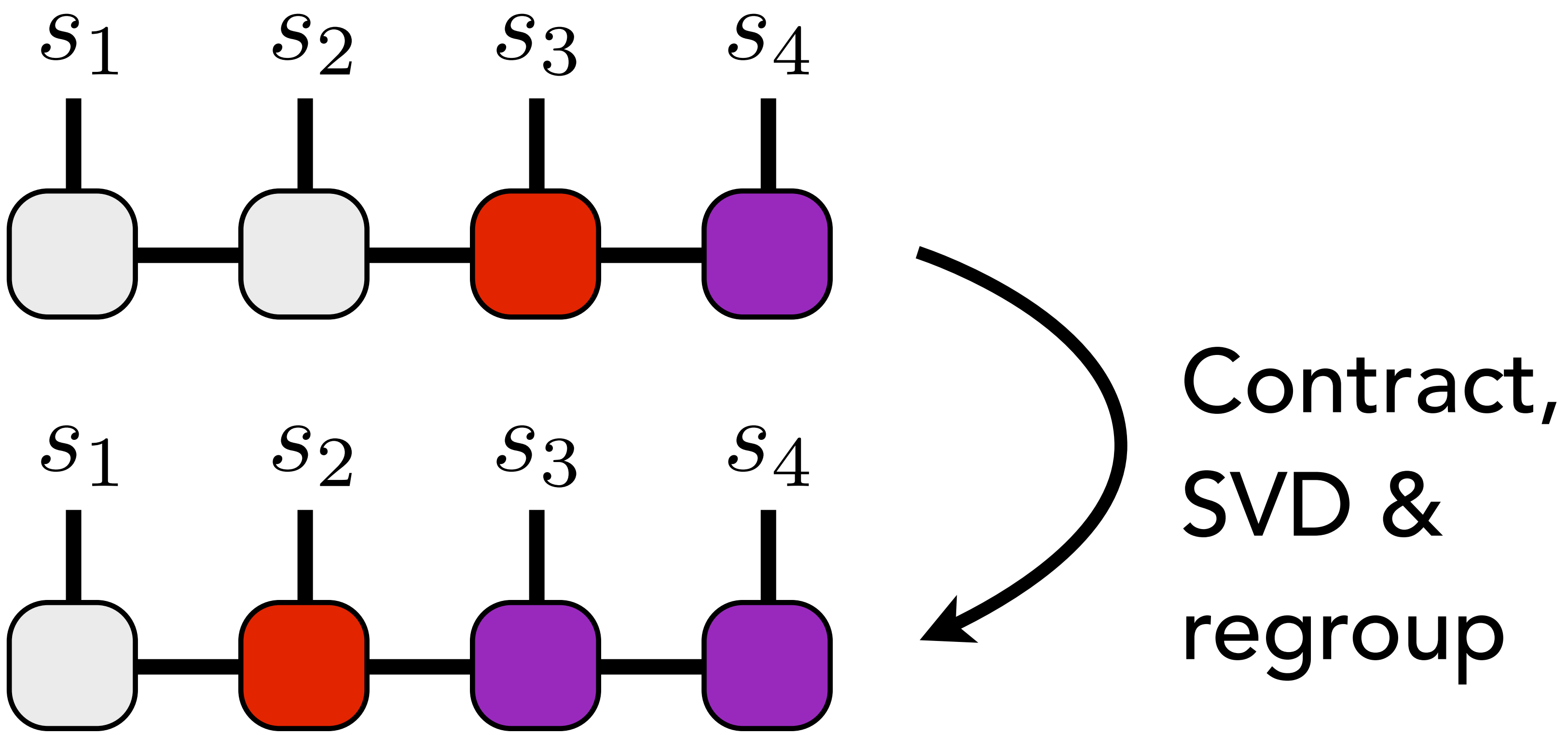
Note that site 4 tensor now right orthogonal



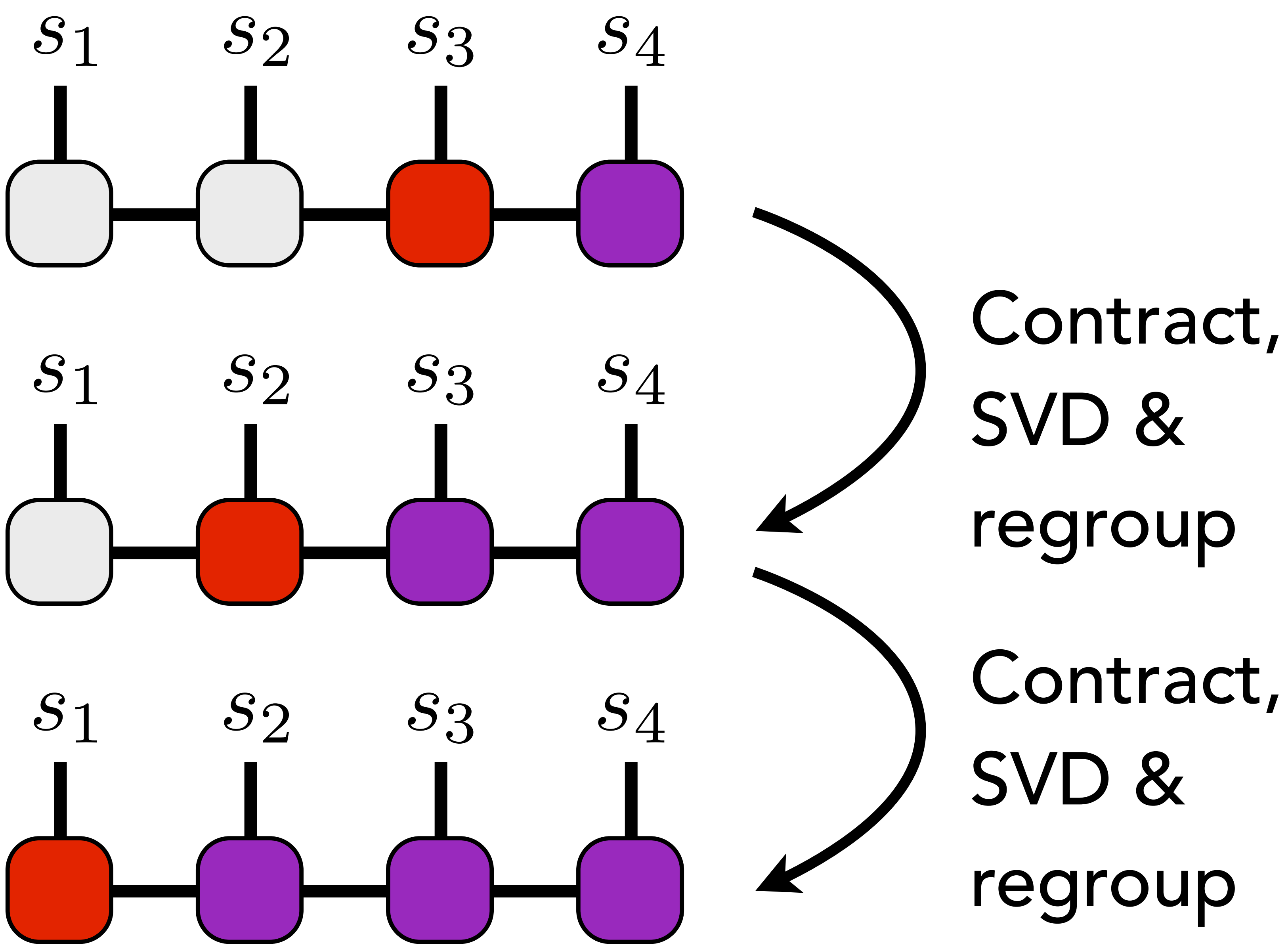
Recall this means



Can repeat gauge transformation (repeated SVD)

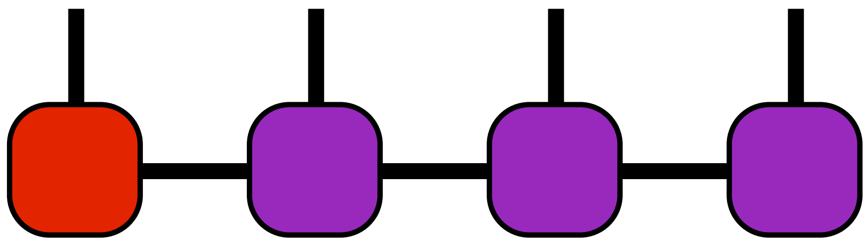


Can repeat gauge transformation (repeated SVD)



What have we gained?

Consider measuring an operator on site 1

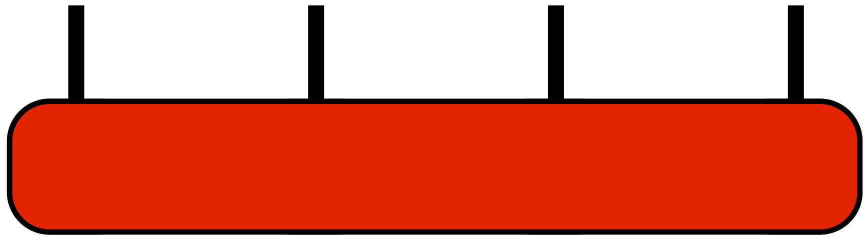


What have we gained?

Consider measuring an operator on site 1

First, general wavefunction:

$$|\Psi\rangle$$



What have we gained?

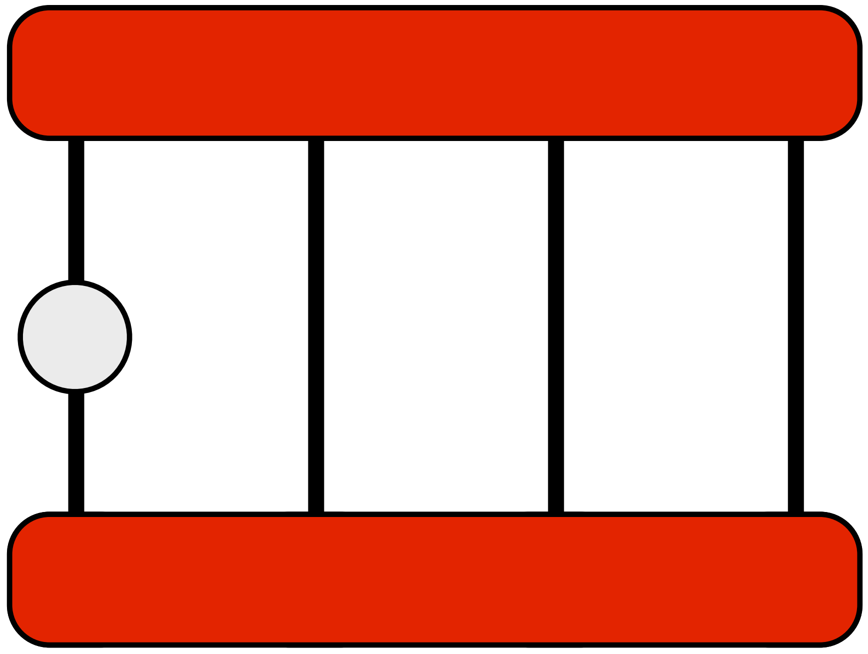
Consider measuring an operator on site 1

First, general wavefunction:

$\langle \Psi |$

$\hat{A}_1$

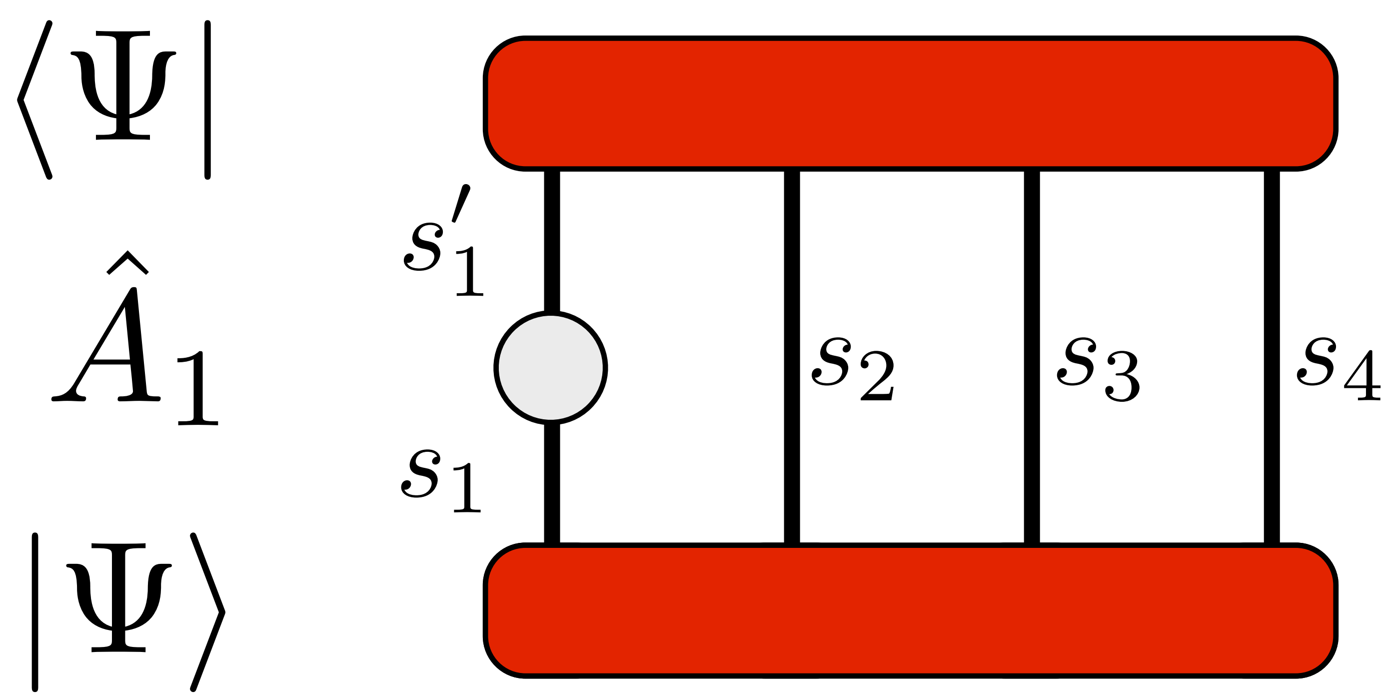
$| \Psi \rangle$



What have we gained?

Consider measuring an operator on site 1

First, general wavefunction:

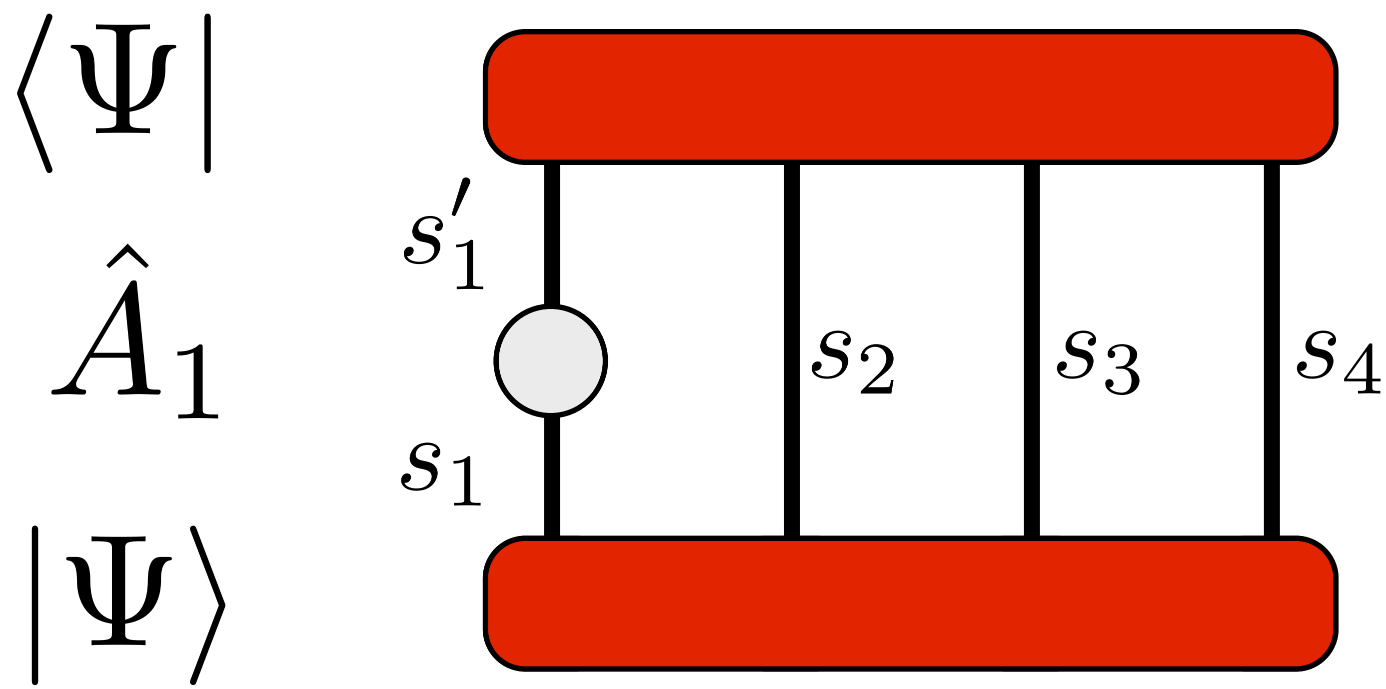


$$\langle \hat{A}_1 \rangle = \sum_{\{s\}} \bar{\psi}_{s_1' s_2 s_3 s_4} A_{s_1' s_1} \psi_{s_1 s_2 s_3 s_4}$$

What have we gained?

Consider measuring an operator on site 1

First, general wavefunction:



Cost scales exponentially!

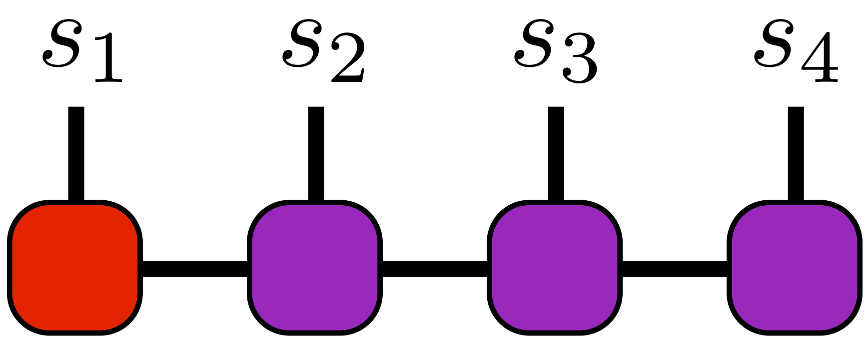
2^4 in this case

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

What have we gained?

Consider measuring an operator on site 1

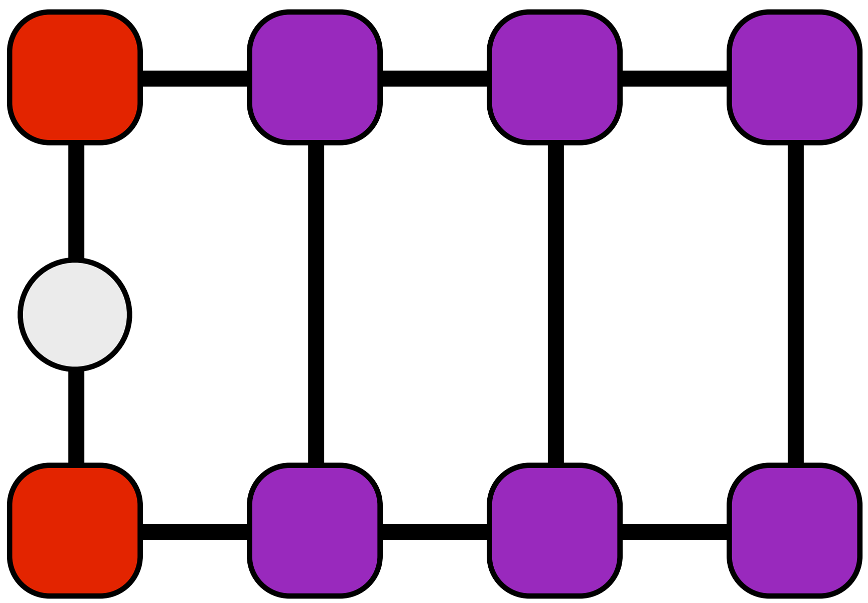
Now gauged MPS:



What have we gained?

Consider measuring an operator on site 1

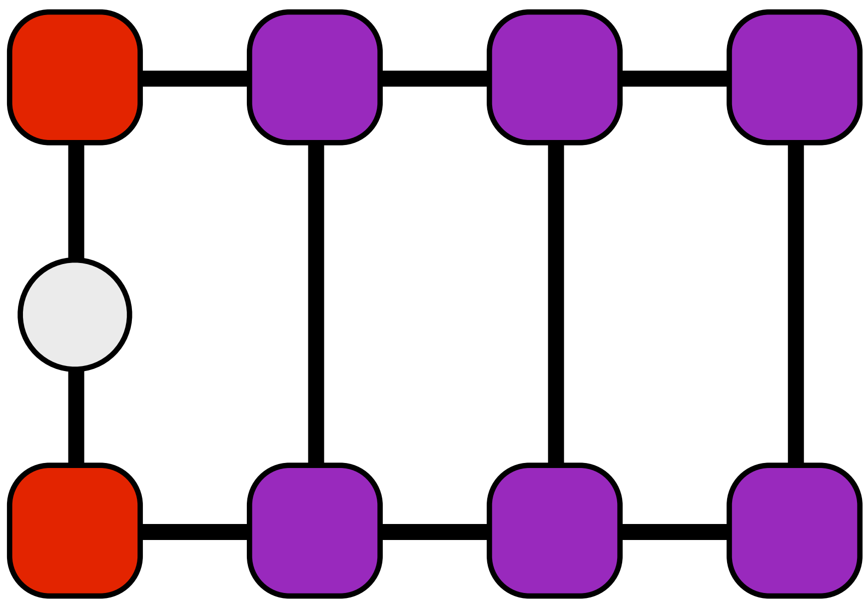
Now gauged MPS:



What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

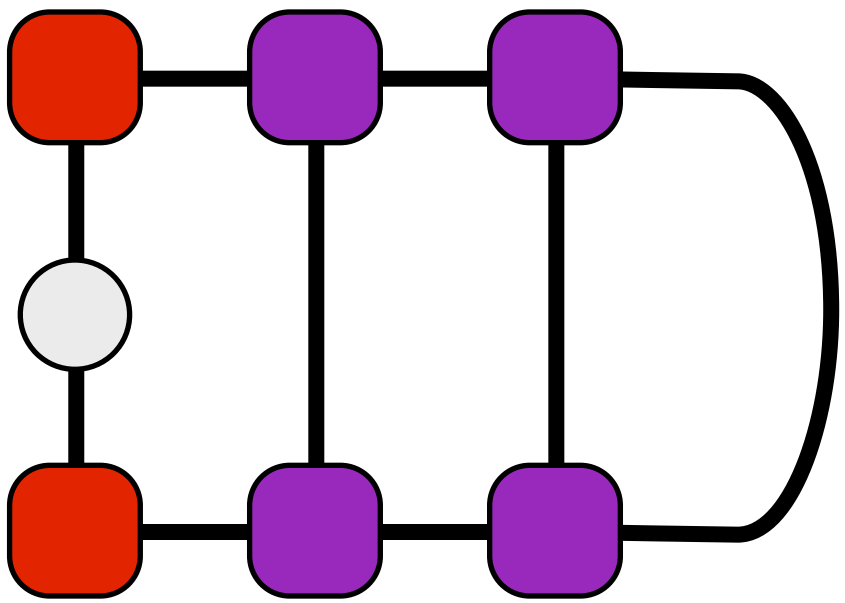


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

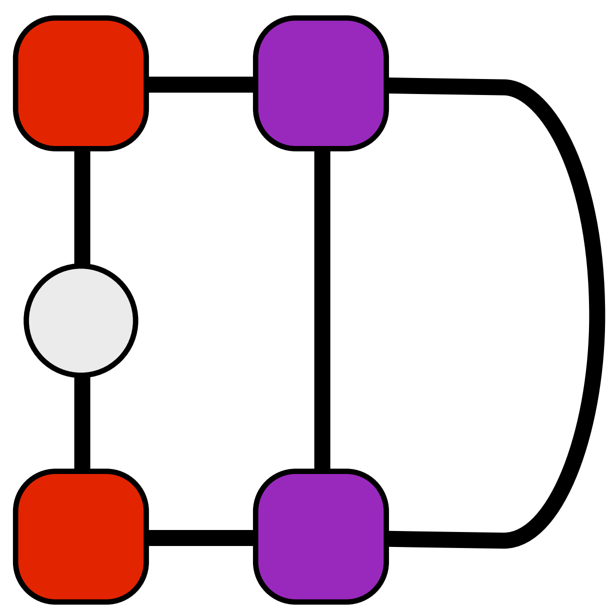


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:

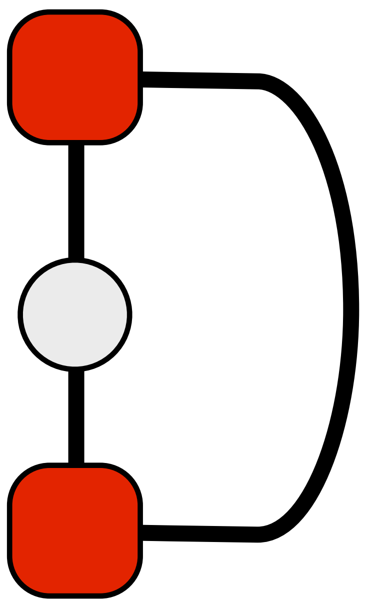


Use right orthogonality

What have we gained?

Consider measuring an operator on site 1

Now gauged MPS:



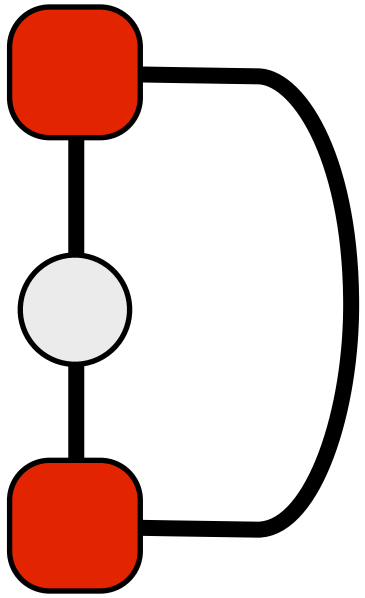
Use right orthogonality

Much simpler computation!

What have we gained?

How much simpler a computation?

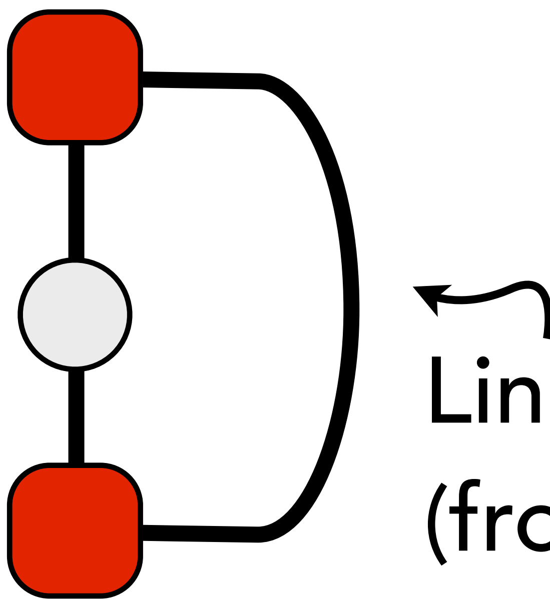
Choose always $\leq m$ singular values in each SVD



What have we gained?

How much simpler a computation?

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

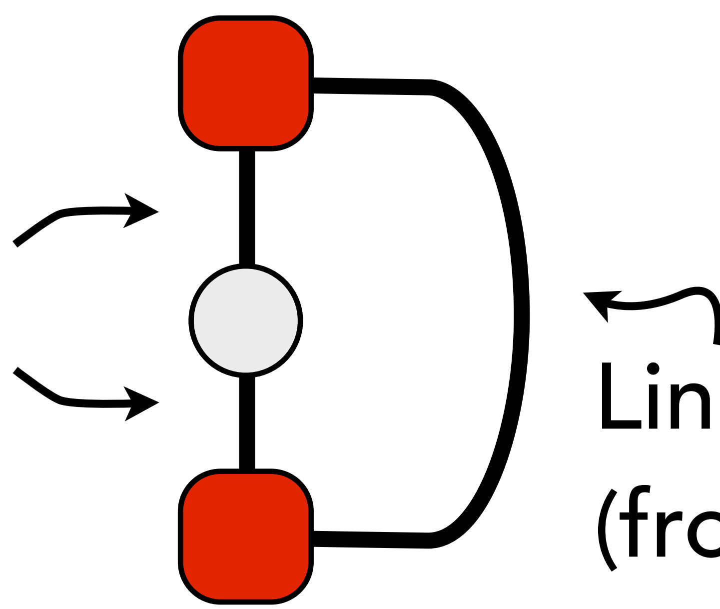


What have we gained?

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)

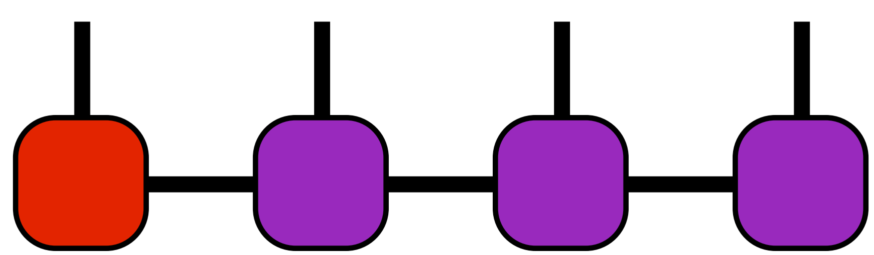
Gauging an MPS Using ITensor:

Create lattice sites and MPS

```
auto sites = SpinHalf(N);
auto psi = MPS(sites);
computeGroundState(psi);
```

Gauge to site number 2

```
psi.position(2);
```



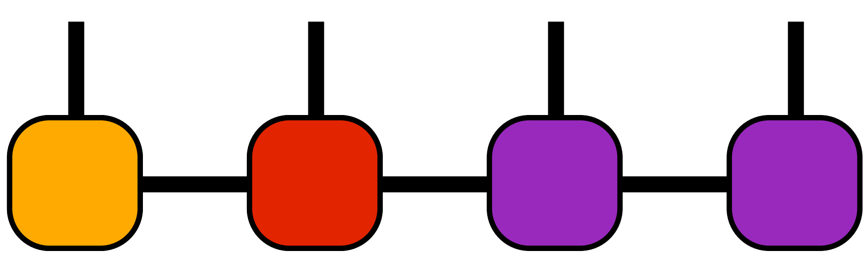
Gauging an MPS Using ITensor:

Create lattice sites and MPS

```
auto sites = SpinHalf(N);
auto psi = MPS(sites);
computeGroundState(psi);
```

Gauge to site number 2

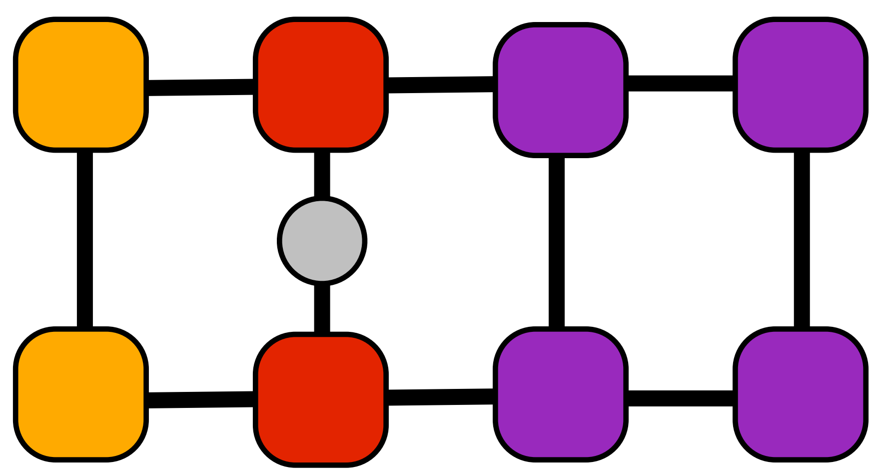
```
psi.position(2);
```



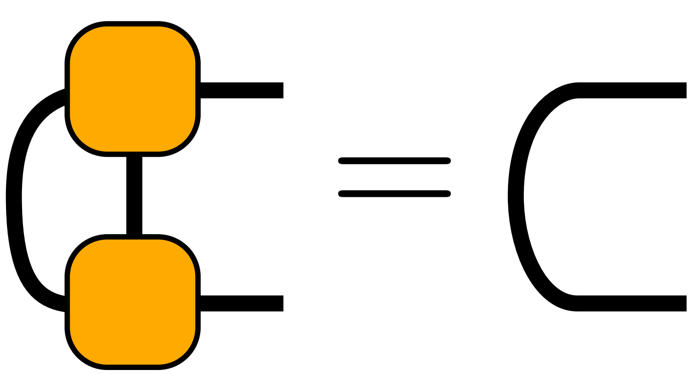
Measuring an MPS Using ITensor:

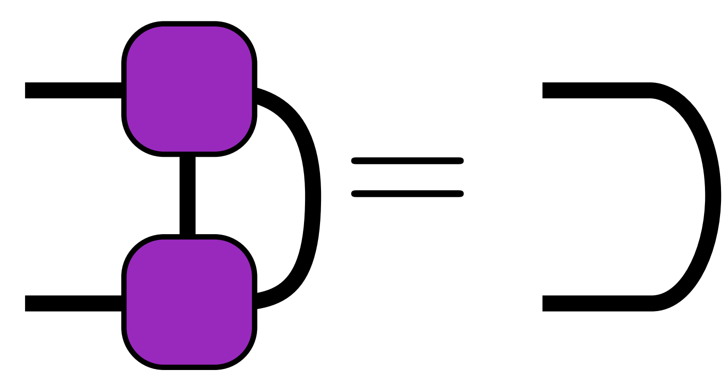
Measure Sz on second site

```
auto sz2 = (dag(prime(psi.A(2),Site))
           * sites.op("Sz",2)
           * psi.A(2)).real();
```



Recall:

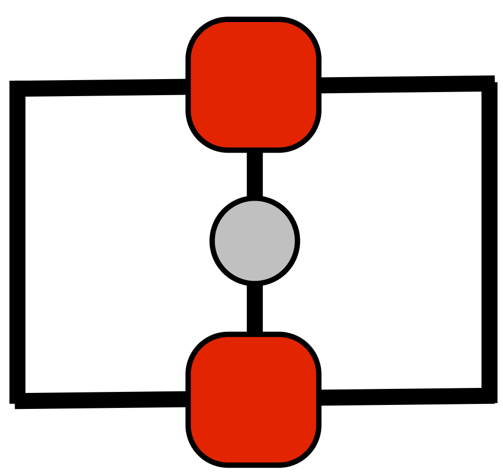




Measuring an MPS Using ITensor:

Measure Sz on second site

```
auto sz2 = (dag(prime(psi.A(2),Site))
            * sites.op("Sz",2)
            * psi.A(2)).real();
```



Recall:

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

`itensor_tutorial/04_mps`

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

2. Study the code [lines 46-51] which measures $\hat{B}_{N/2} = \mathbf{S}_{N/2} \cdot \mathbf{S}_{N/2+1}$

3. Create similar code that measures the bond strength on bonds (N/2-1) and (N/2+1) [lines 62 and 73]
   These are combined into the "dimer order parameter"
   $$D = \langle \hat{B}_{N/2} \rangle - \frac{1}{2}\langle \hat{B}_{N/2-1} \rangle - \frac{1}{2}\langle \hat{B}_{N/2+1} \rangle$$

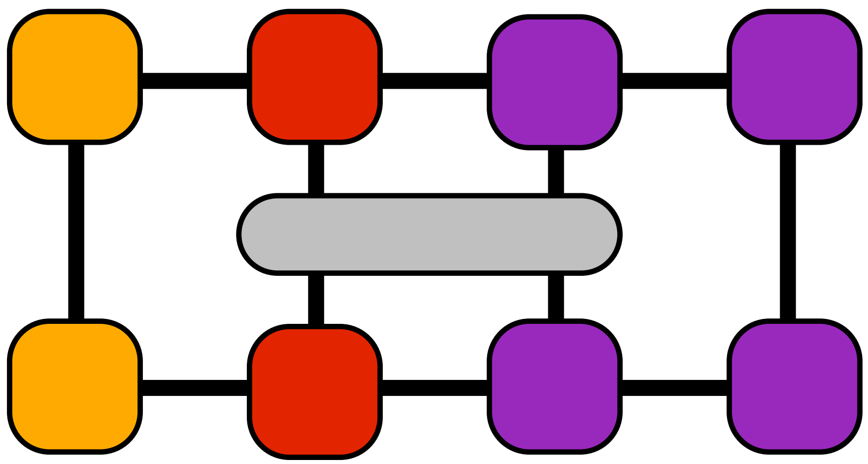
4. Run the code for various sizes N and plot the results.
   How does large J_2/J_1 differ from small J_2/J_1 ?

Solution for missing lines of **j1j2.cc:**

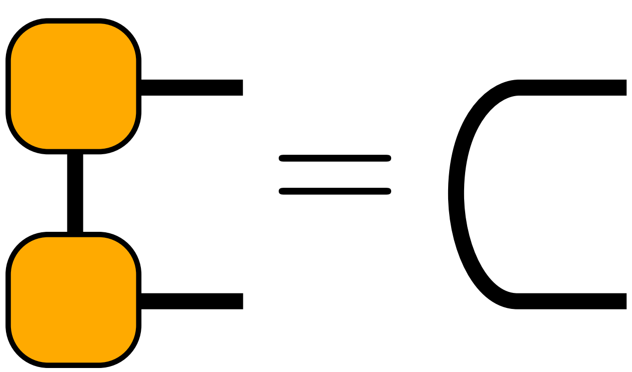
```
val += -0.5 * (dag(prime(wf2,Site)) * B2 * wf2).real();


val += -0.5 * (dag(prime(wf3,Site)) * B3 * wf3).real();
```

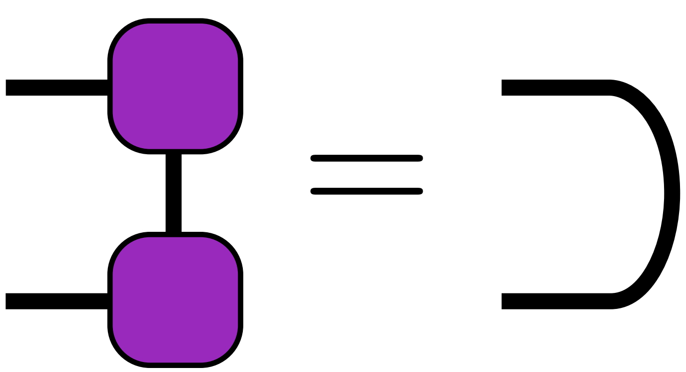
05 Trotter

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

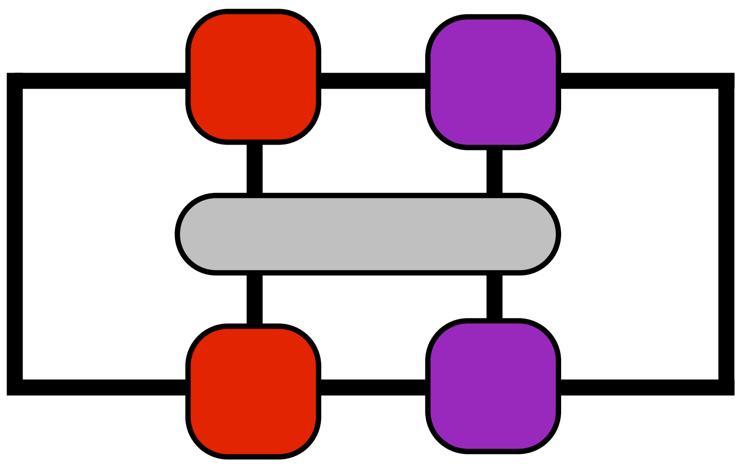


Recall:

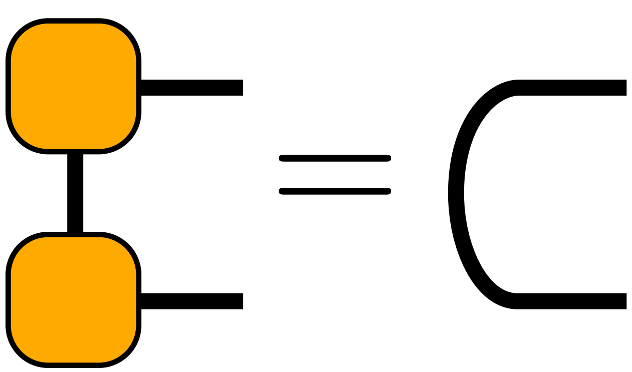


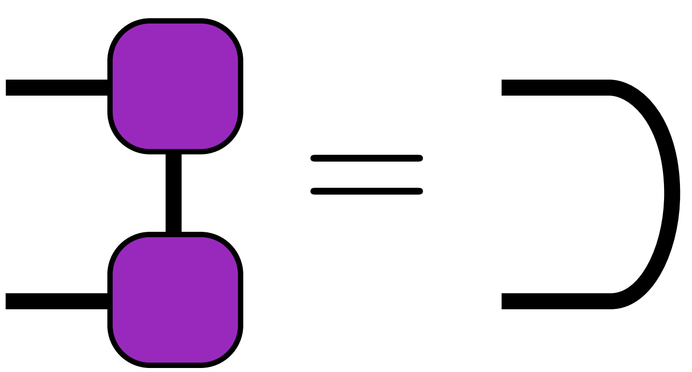


Just as we can measure one-site operators,
can measure two-site operators

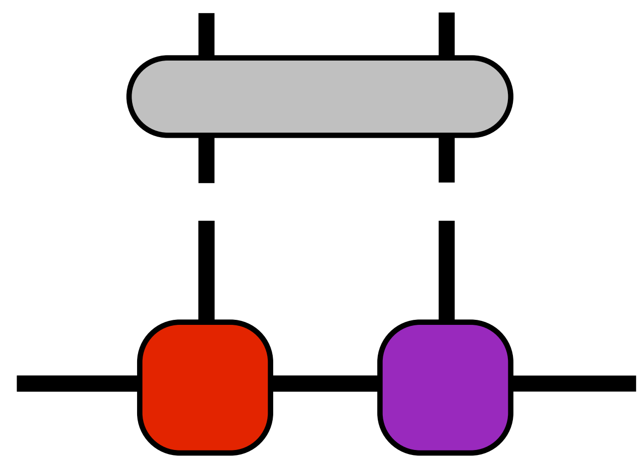


Recall:

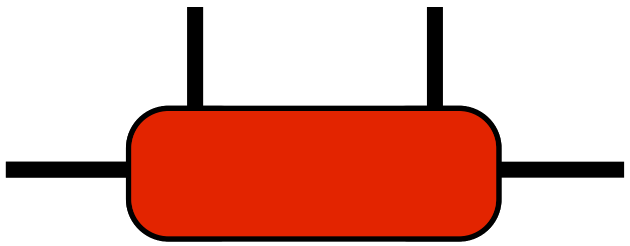




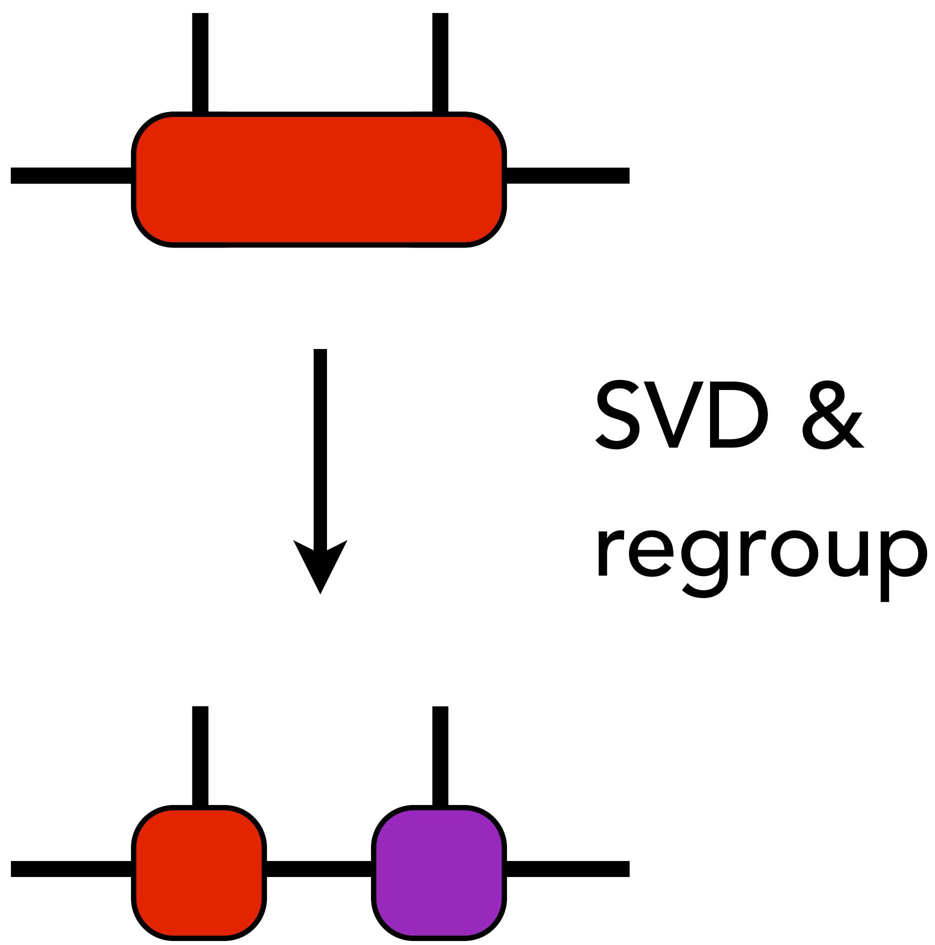
Since two "center" sites have orthogonal environment, ok to apply operators:



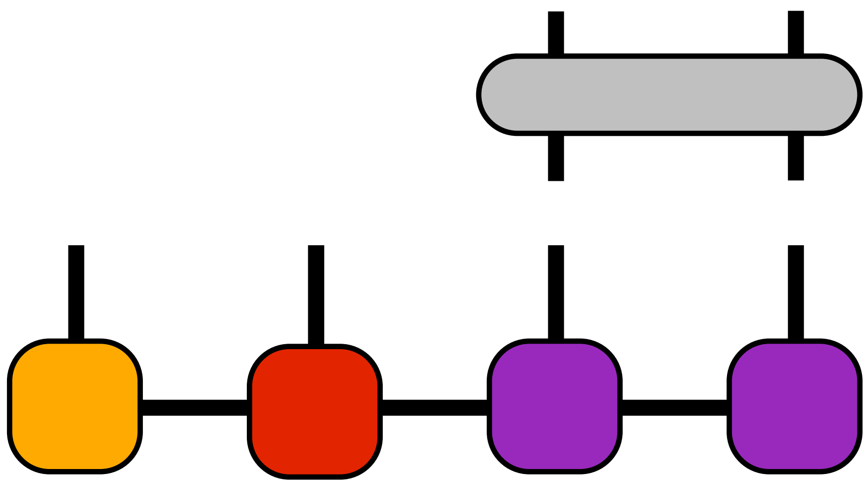
Since two “center” sites have orthogonal environment, ok to apply operators:



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

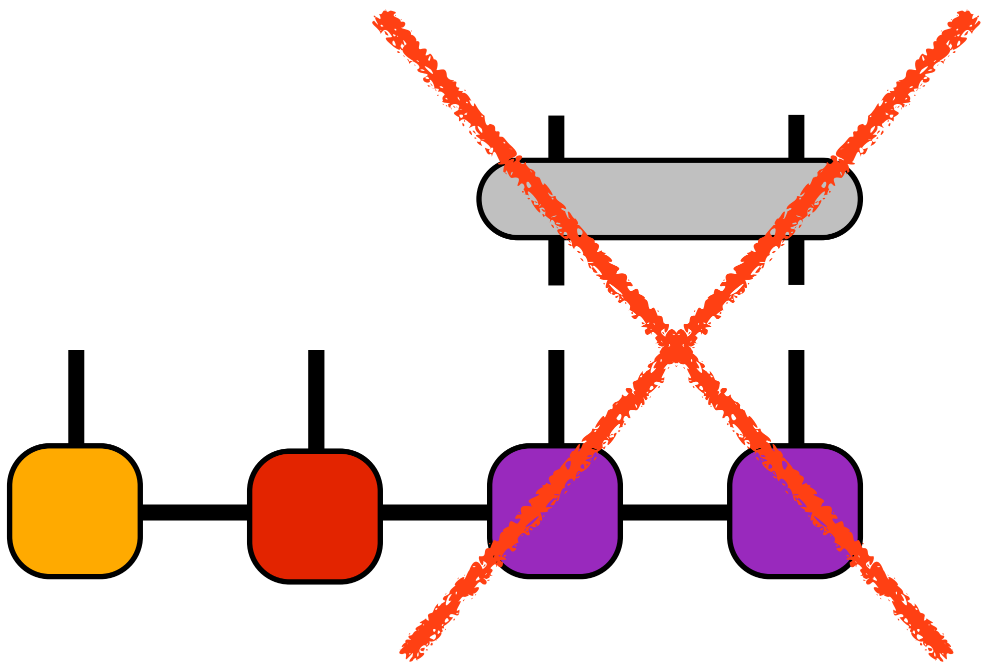


Would NOT be ok on another bond without regauging



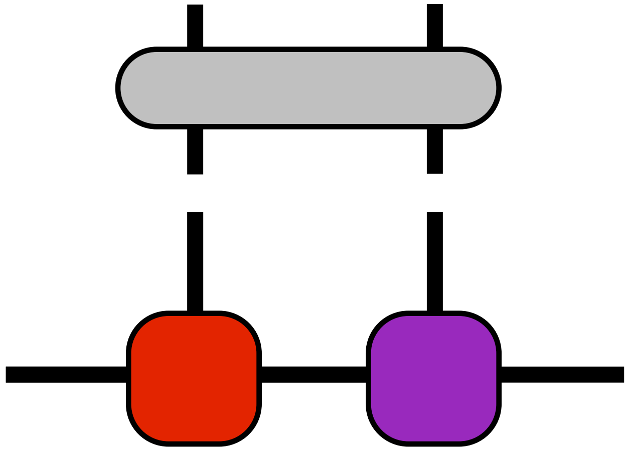
SVD truncation not globally optimal except at orthogonality center

Would NOT be ok on another bond without regauging

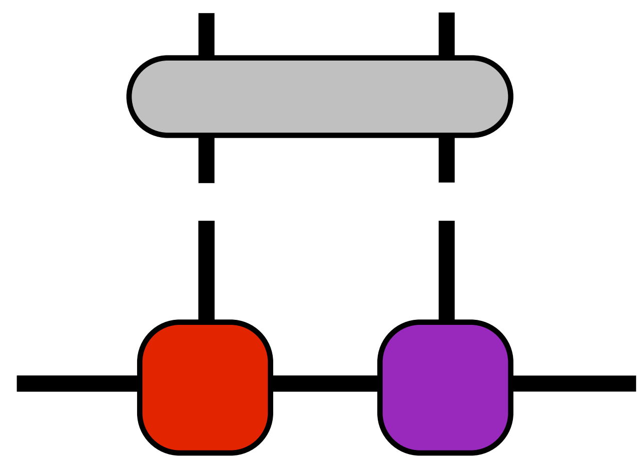


SVD truncation not globally optimal except at orthogonality center

Q: What can we do with this capability?



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

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

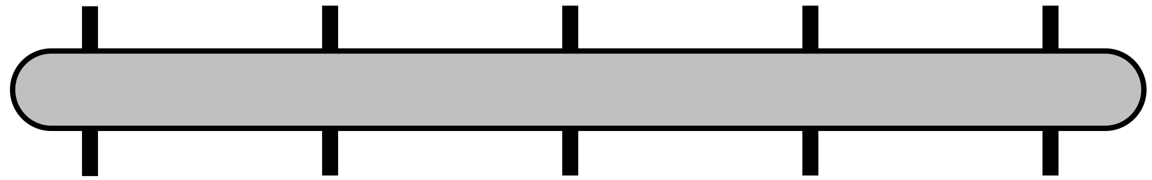
$$= (\mathbf{S}_1 \cdot \mathbf{S}_2) + (\mathbf{S}_2 \cdot \mathbf{S}_3) + (\mathbf{S}_3 \cdot \mathbf{S}_4)$$

For a small time step τ

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

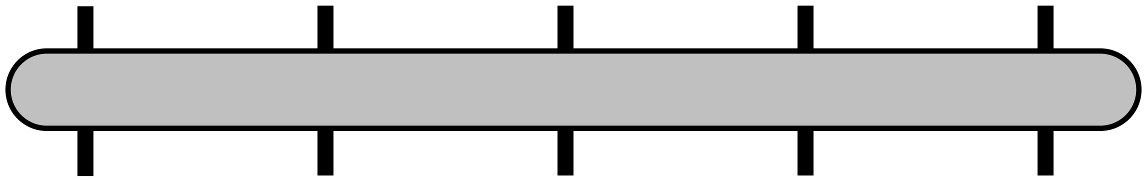
$$\cdots e^{-\tau H_3/2} e^{-\tau H_2/2} e^{-\tau H_1/2} + \mathcal{O}(\tau^3)$$

Diagramatically,

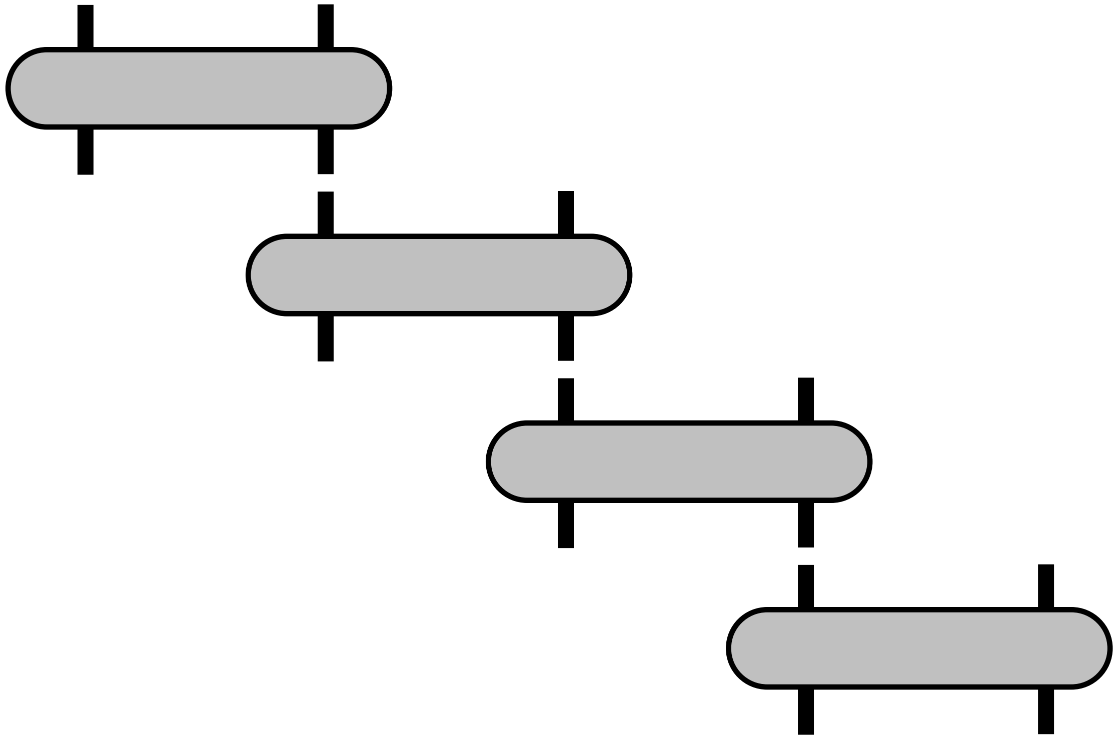


$$\simeq$$

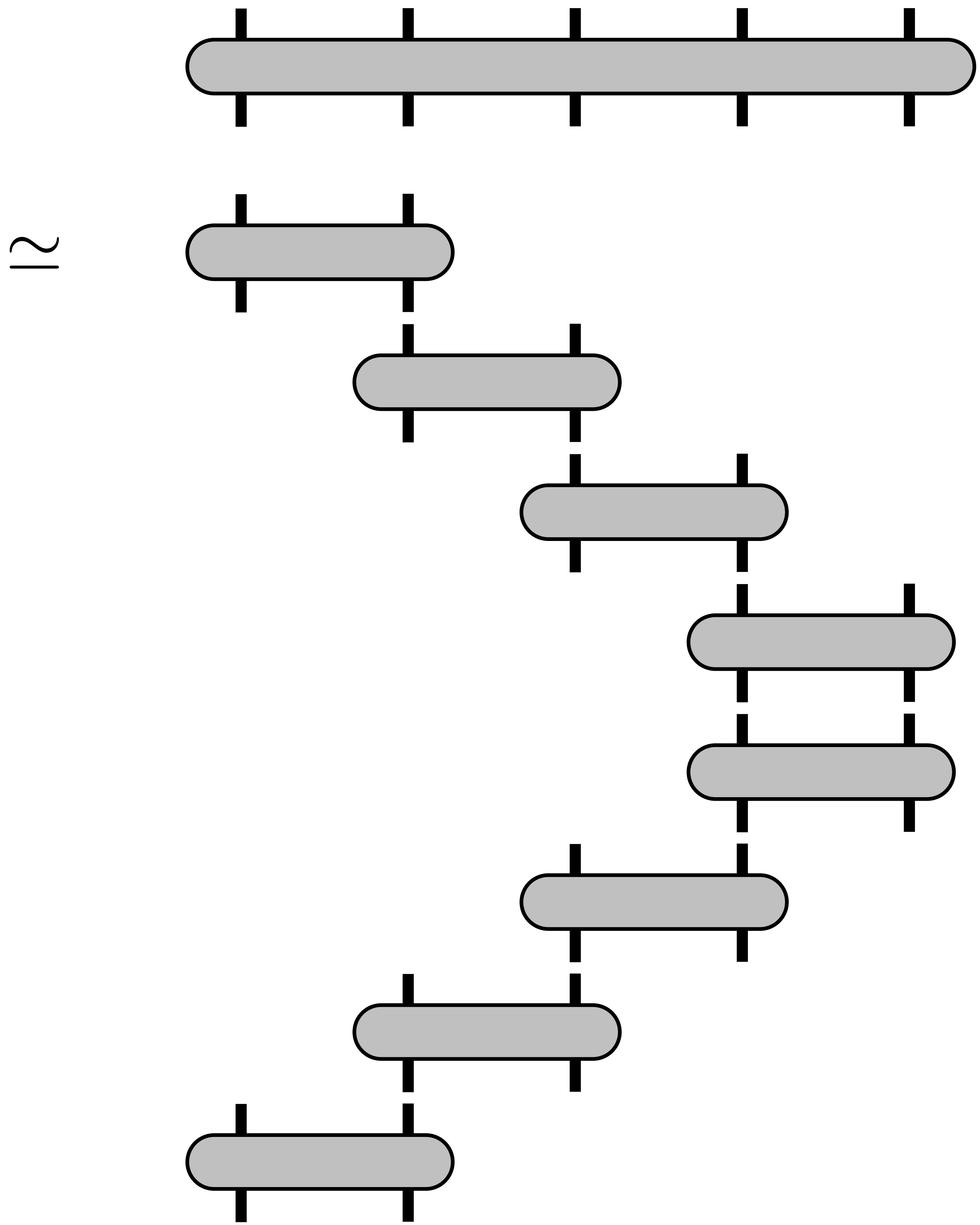
Diagramatically,



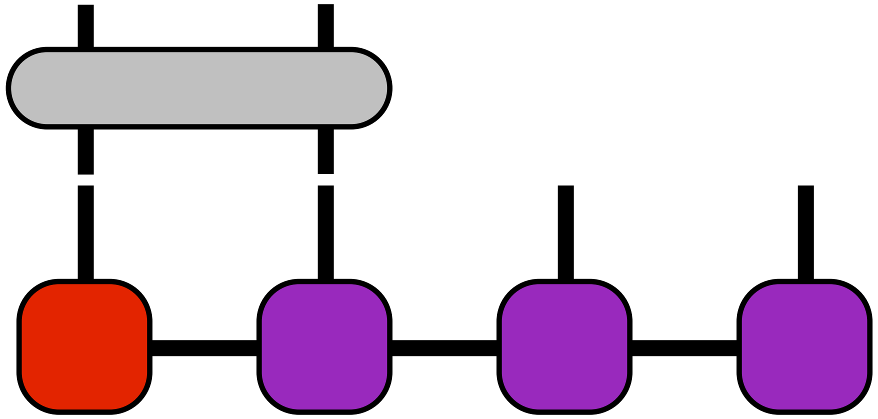
$\simeq$



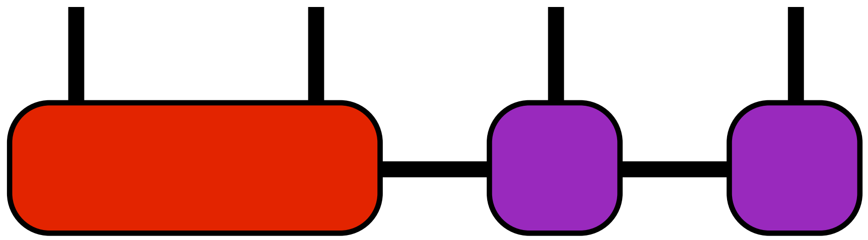
Diagramatically,



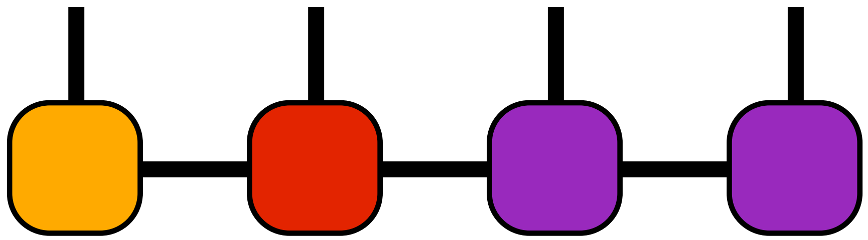
Apply to MPS as follows:



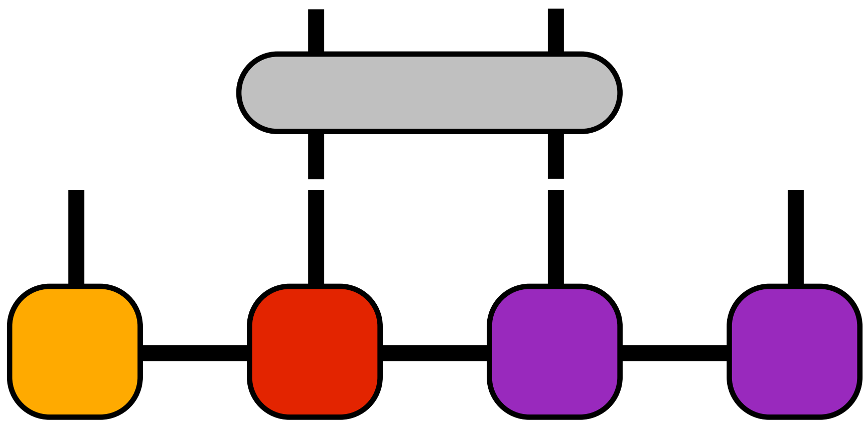
Apply to MPS as follows:



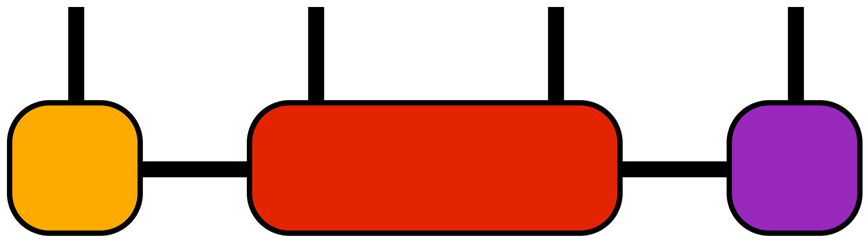
Apply to MPS as follows:



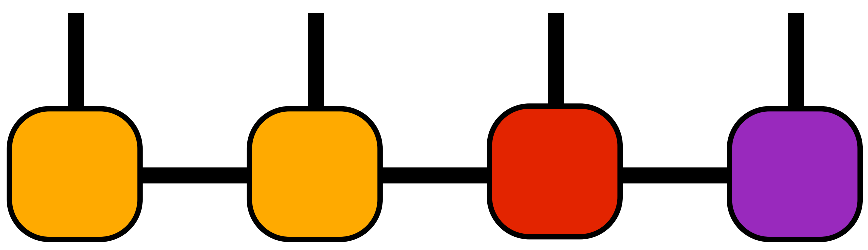
Apply to MPS as follows:



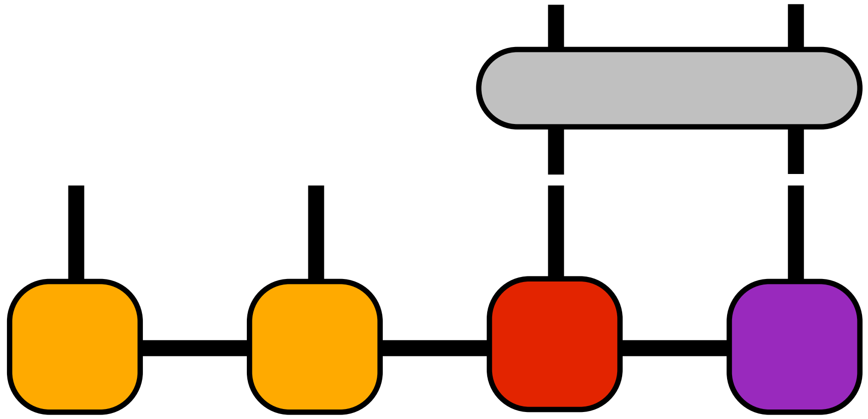
Apply to MPS as follows:



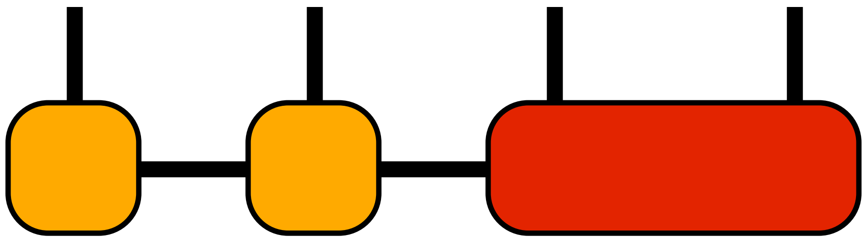
Apply to MPS as follows:



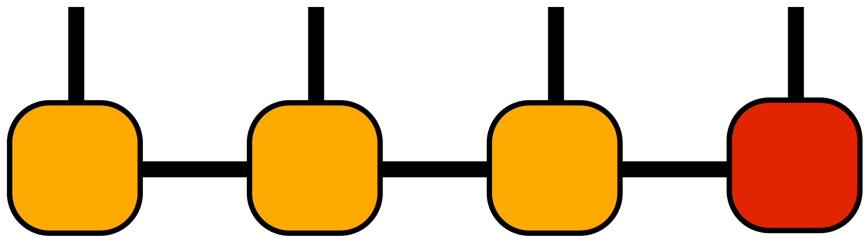
Apply to MPS as follows:



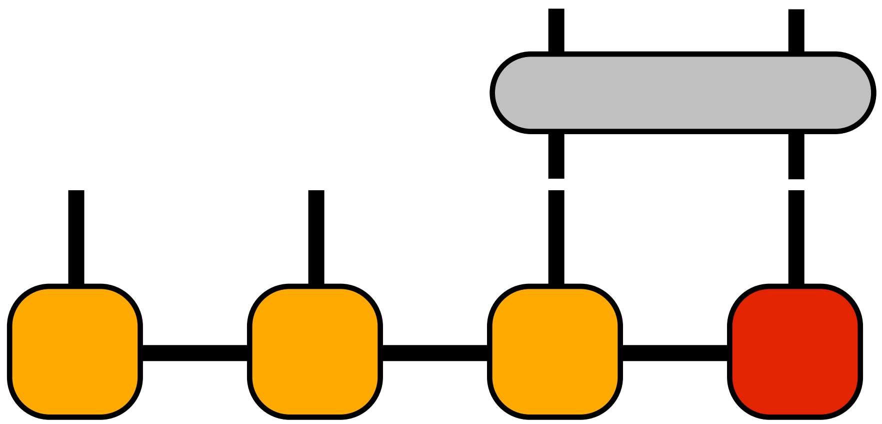
Apply to MPS as follows:



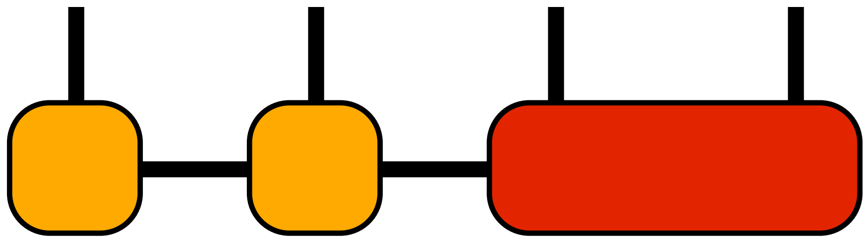
Apply to MPS as follows:



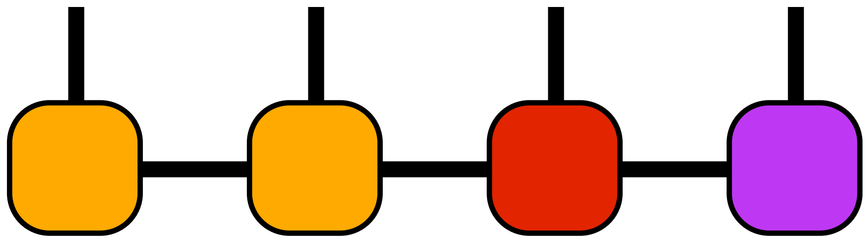
Apply to MPS as follows:



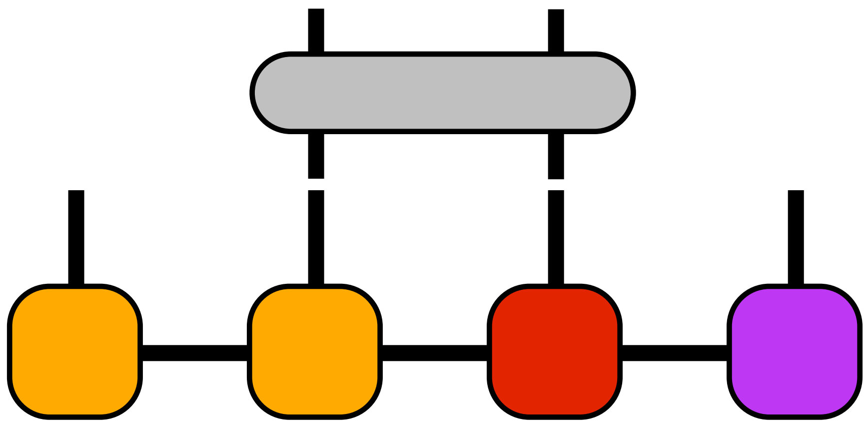
Apply to MPS as follows:



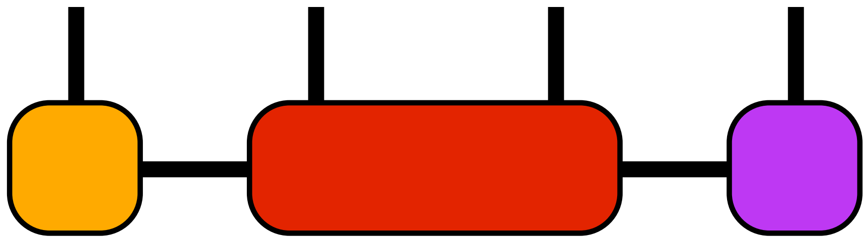
Apply to MPS as follows:



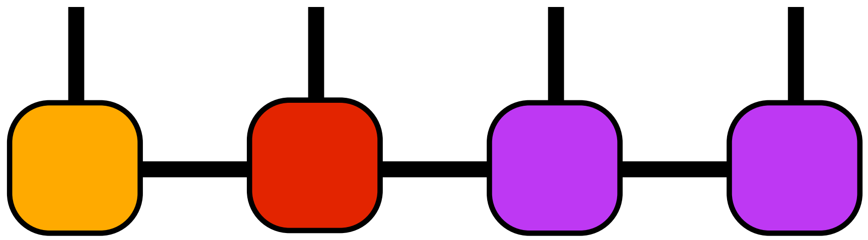
Apply to MPS as follows:



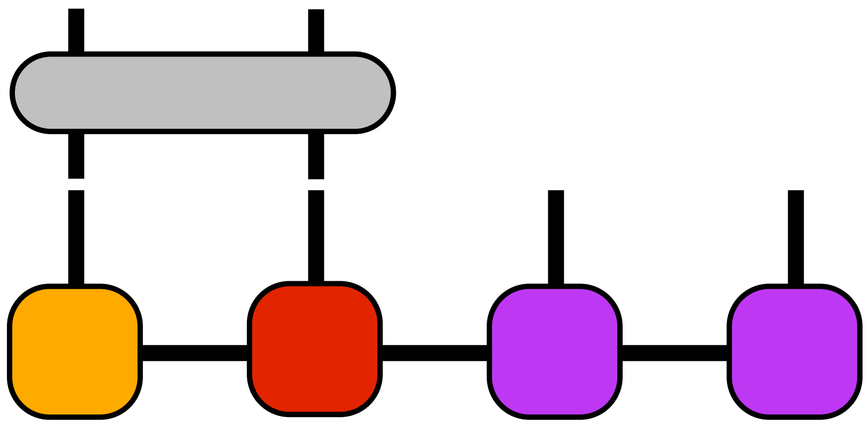
Apply to MPS as follows:



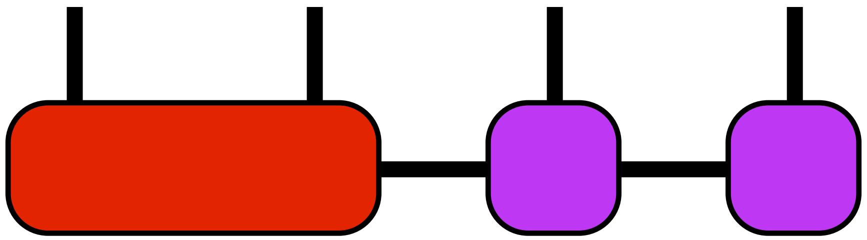
Apply to MPS as follows:



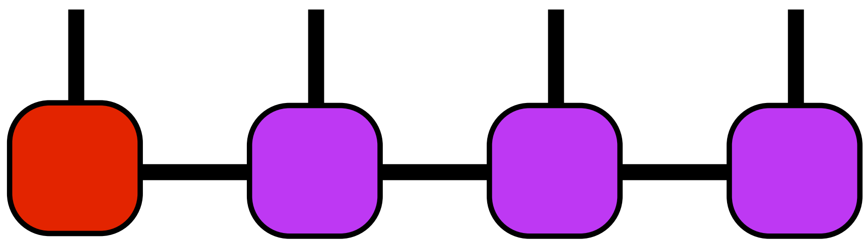
Apply to MPS as follows:



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

`itensor_tutorial/05_gates`

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

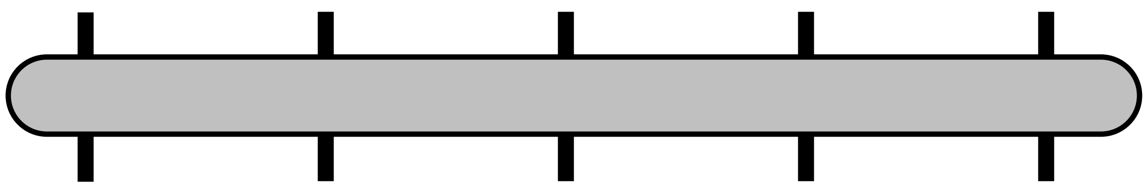
2. Apply the gate G to the MPS bond tensor AA.
   The gate G can be multiplied times AA as if it's an ITensor

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

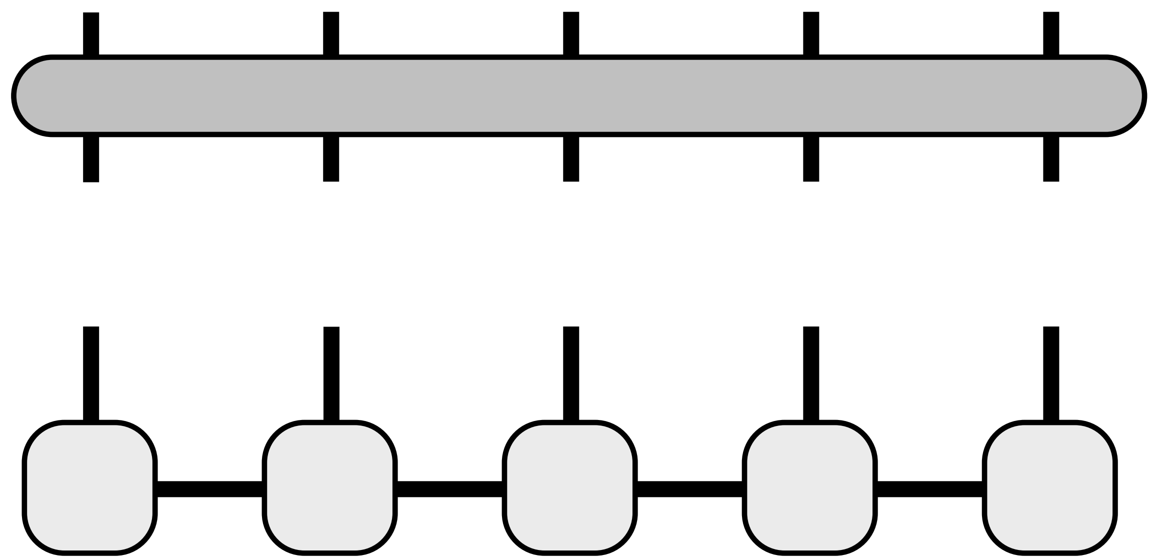
3. Try increasing the total time "ttotal" to imaginary time
   evolve toward the ground state.
   (Exact energy for 20 sites: E_0 = -8.6824733317)

05 MPO

We have seen a Hamiltonian looks like this:



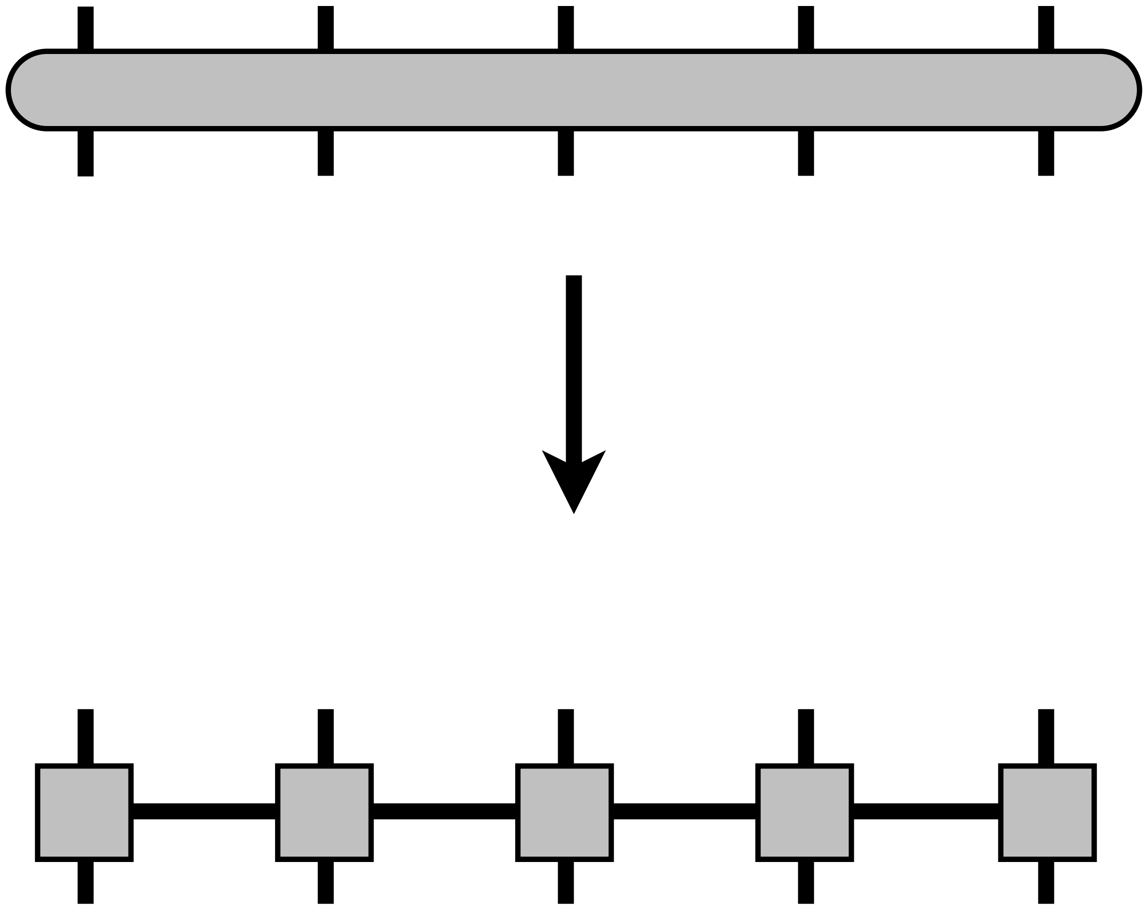
We have seen a Hamiltonian looks like this:



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

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

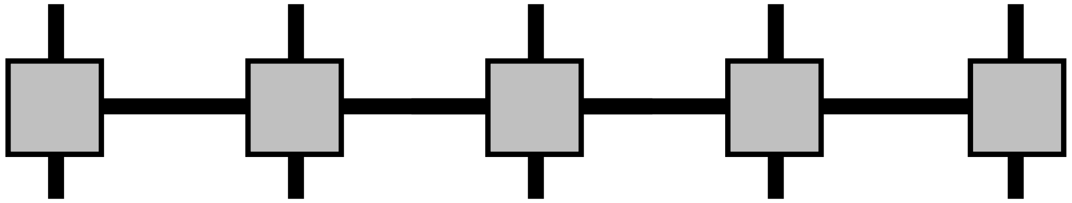
Want something like



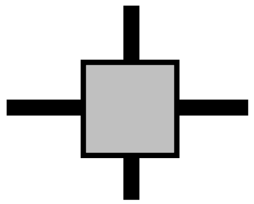
Operator (H) as product of “matrices”

matrix product operator

Focus on just one tensor

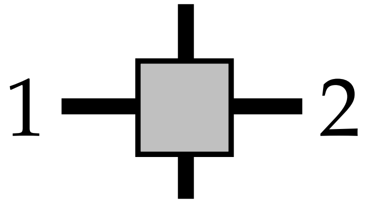


Focus on just one tensor



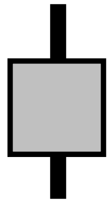
Focus on just one tensor

Specific values for horizontal bonds gives site operator



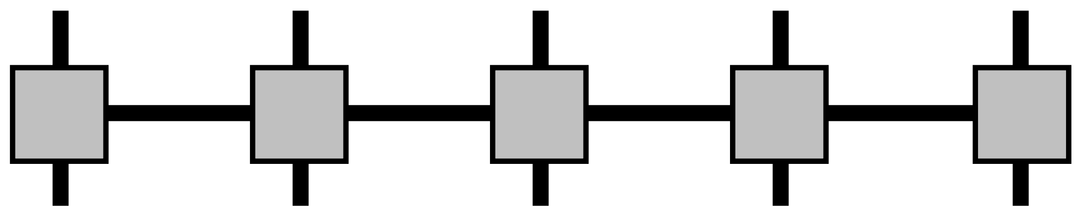
Focus on just one tensor

Specific values for horizontal bonds gives site operator



Focus on just one tensor

Specific values for horizontal bonds
gives site operator



⟶ Each tensor a matrix of site operators!

⟶ Each tensor a matrix of site operators!

Hamiltonians can be written

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

→ Each tensor a matrix of site operators!

Multiply out

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

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

Multiply out

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

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

→ Each tensor a matrix of site operators!

Multiply out

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

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

$$\hat{\sigma}^z_1 \otimes \hat{I}_2 + \hat{I}_1 \otimes \hat{\sigma}^z_2$$

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

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

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

$\hat{\sigma}^z$

More complicated example

2 1

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

3 2 1

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

● ●

More complicated example

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

More complicated example

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

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

More complicated example

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

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

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

Hamiltonian is

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

New AutoMPO feature of ITensor:

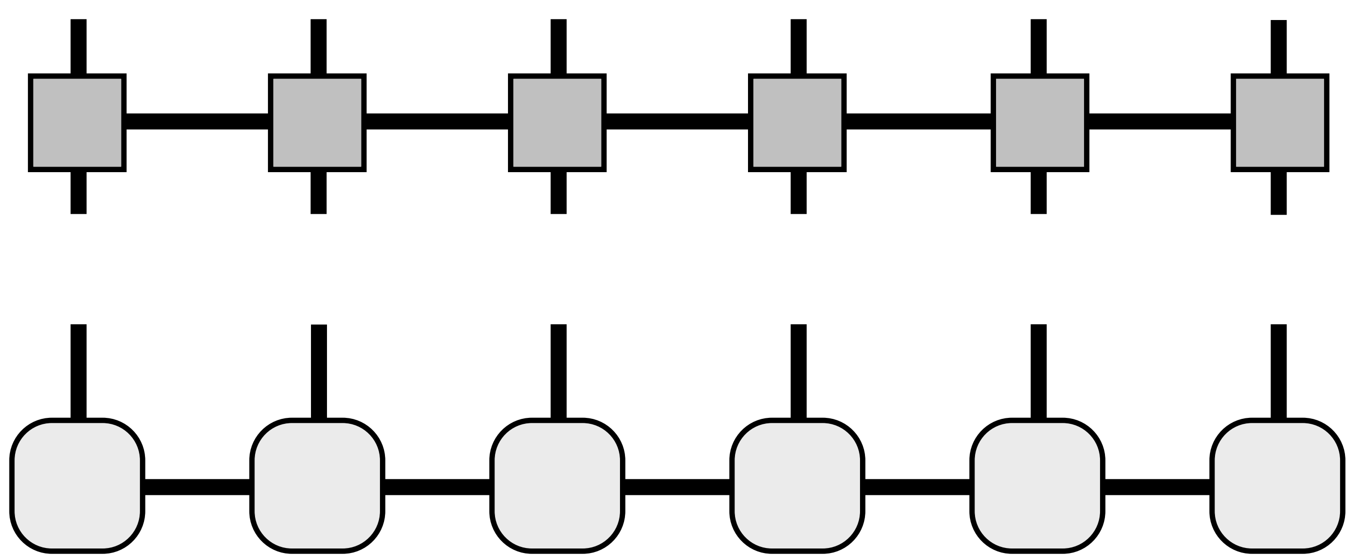
```
auto sites = SpinOne(N);

auto ampo = AutoMPO(sites);
for(int j = 1; j < N; ++j)
    {
    ampo += 0.5,"S+",j,"S-",j+1;
    ampo += 0.5,"S-",j,"S+",j+1;
    ampo +=     "Sz",j,"Sz",j+1;
    }
auto H = MPO(ampo);

auto psi = MPS(sites);

dmrg(psi,H,sweeps);
```

05 DMRG

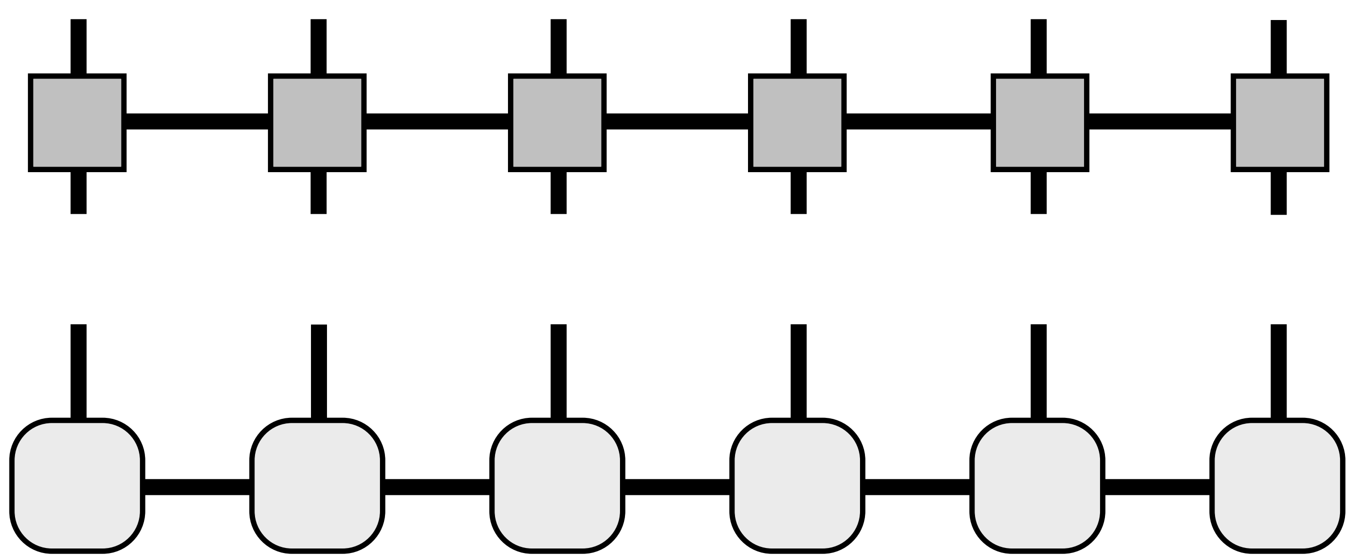
DMRG is typically the best method for finding ground states of 1d Hamiltonians

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

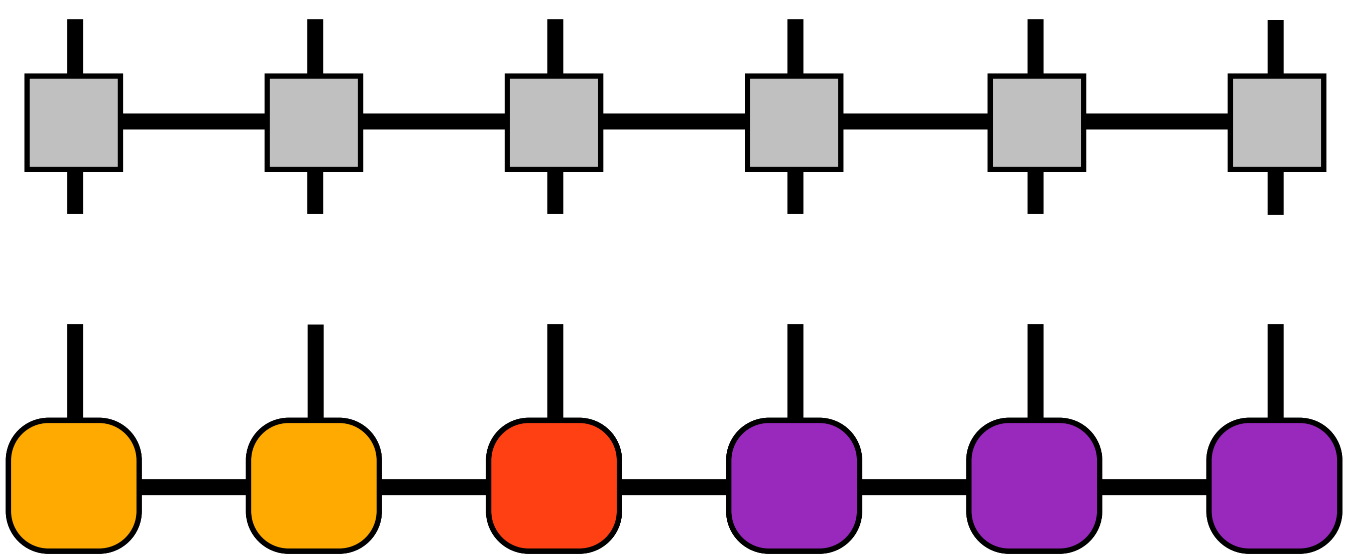
Think of H as MPO



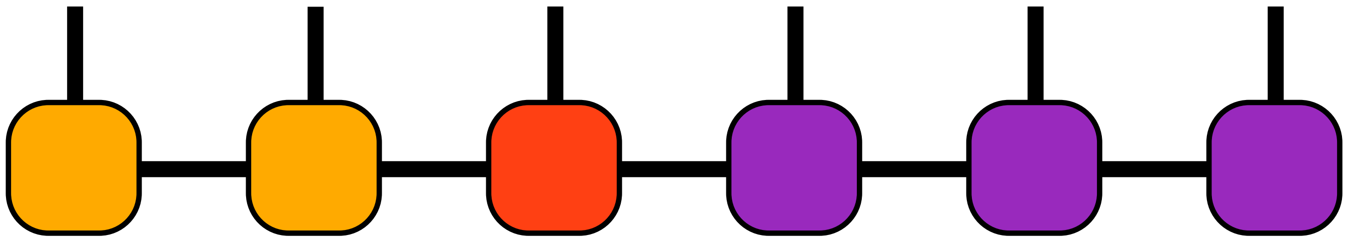
Important: MPS should be in definite gauge
I.e. most tensors unitary



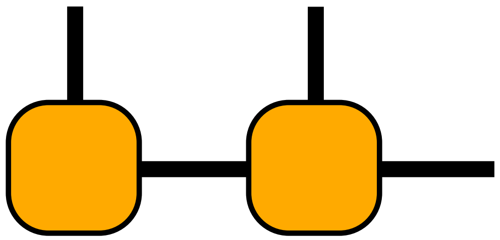
Important: MPS should be in definite gauge
I.e. most tensors unitary



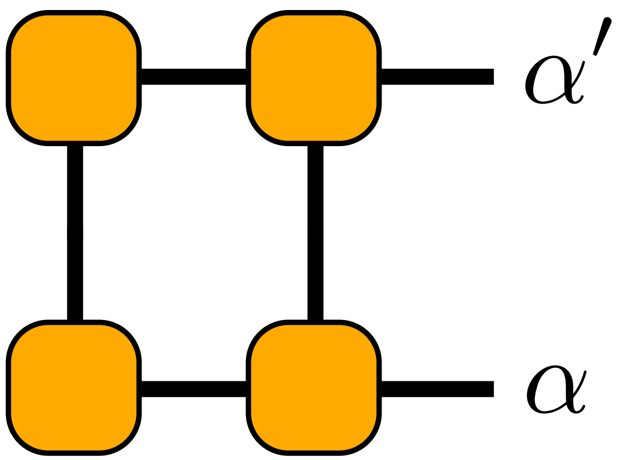
This way, tensors left/right of center define orthonormal bases



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

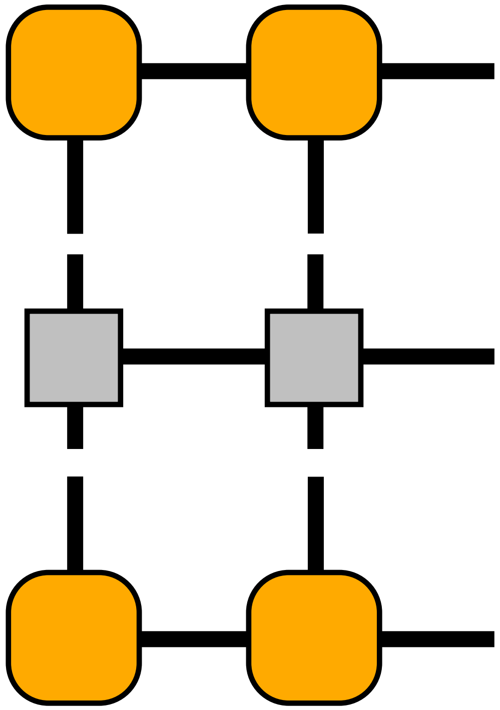


This way, tensors left/right of center define orthonormal bases

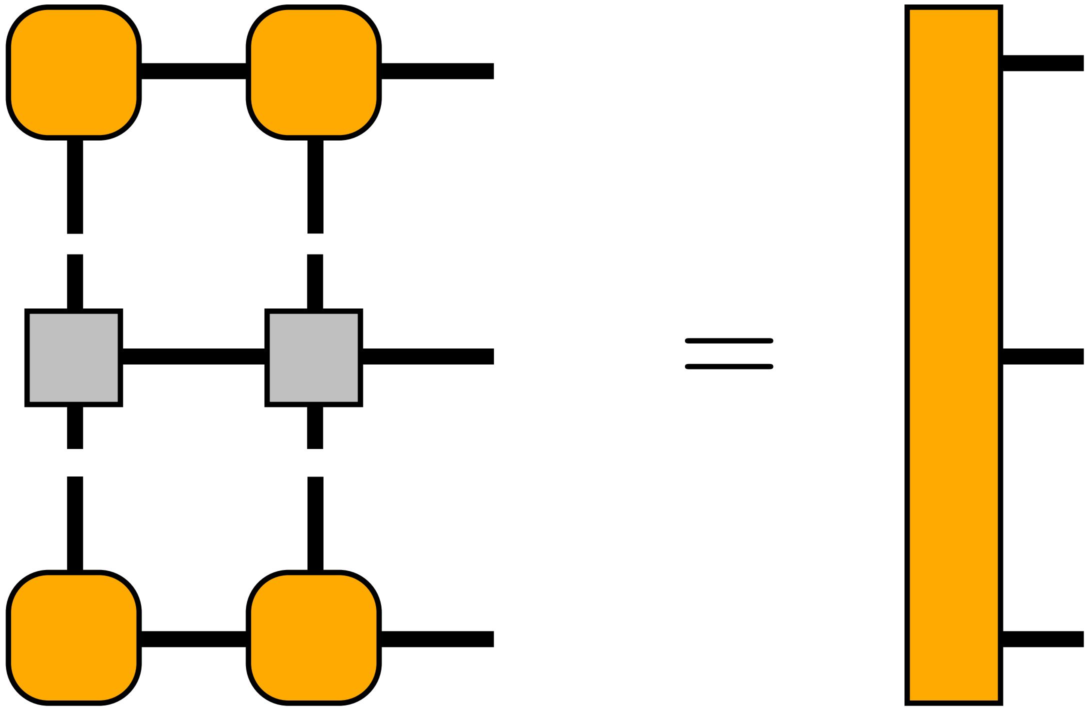


$$= \delta_{\alpha}^{\alpha'}$$

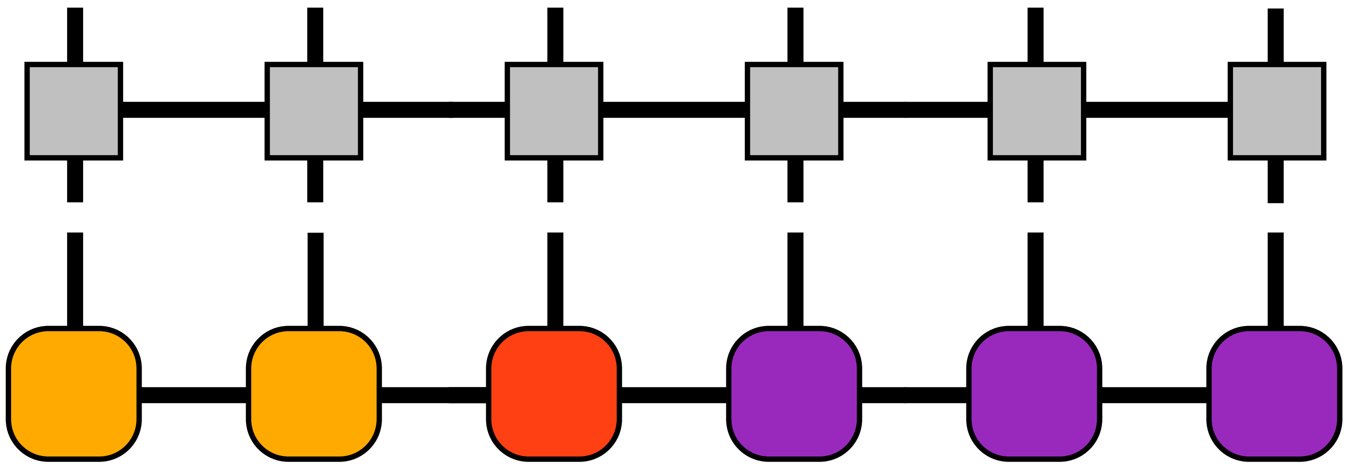
Can project Hamiltonian into this basis



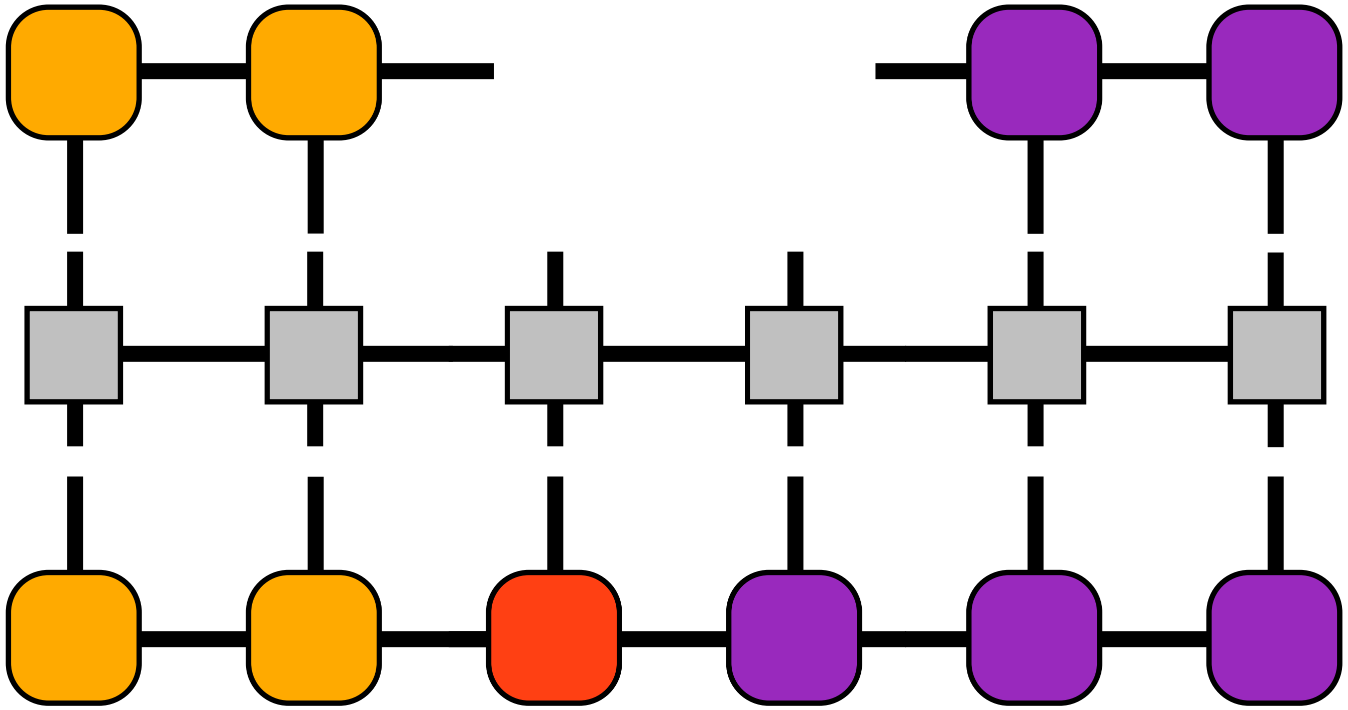
Can project Hamiltonian into this basis



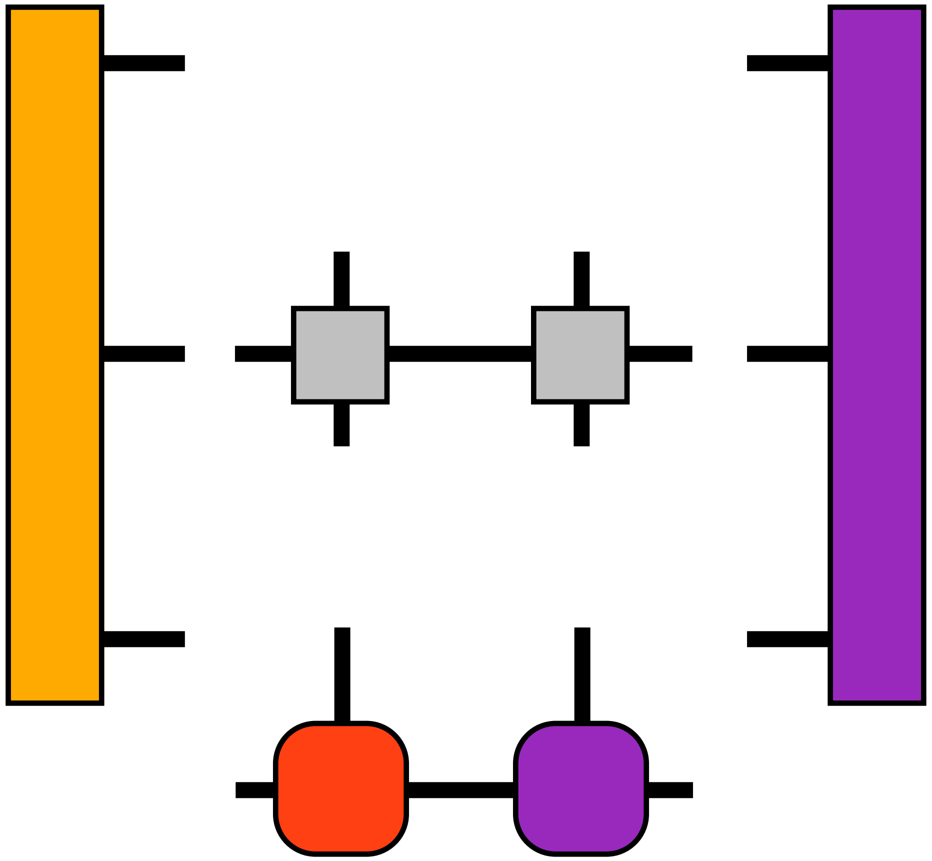
Doing the same on the right gives



Doing the same on the right gives



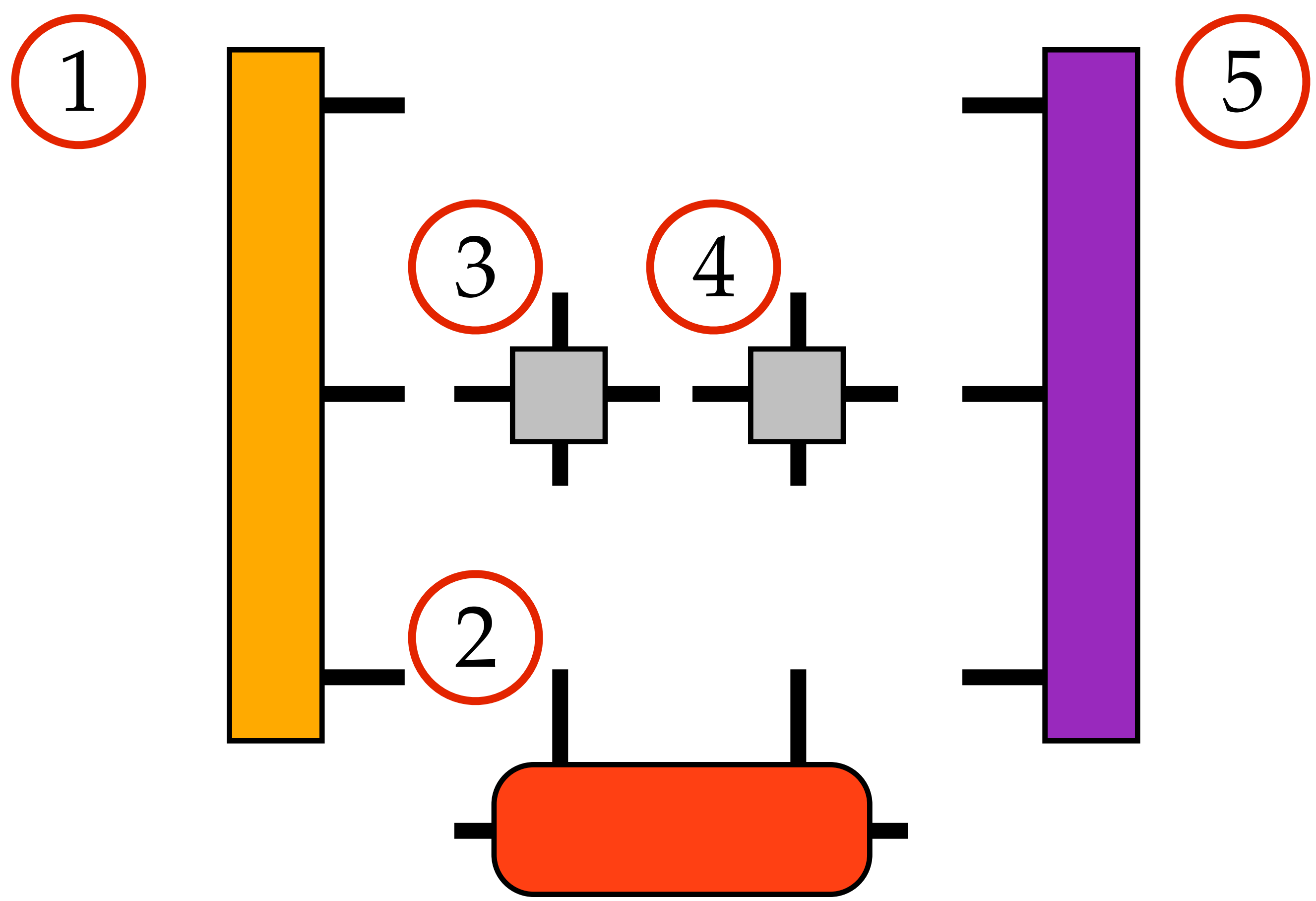
Doing the same on the right gives



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

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

Order important!



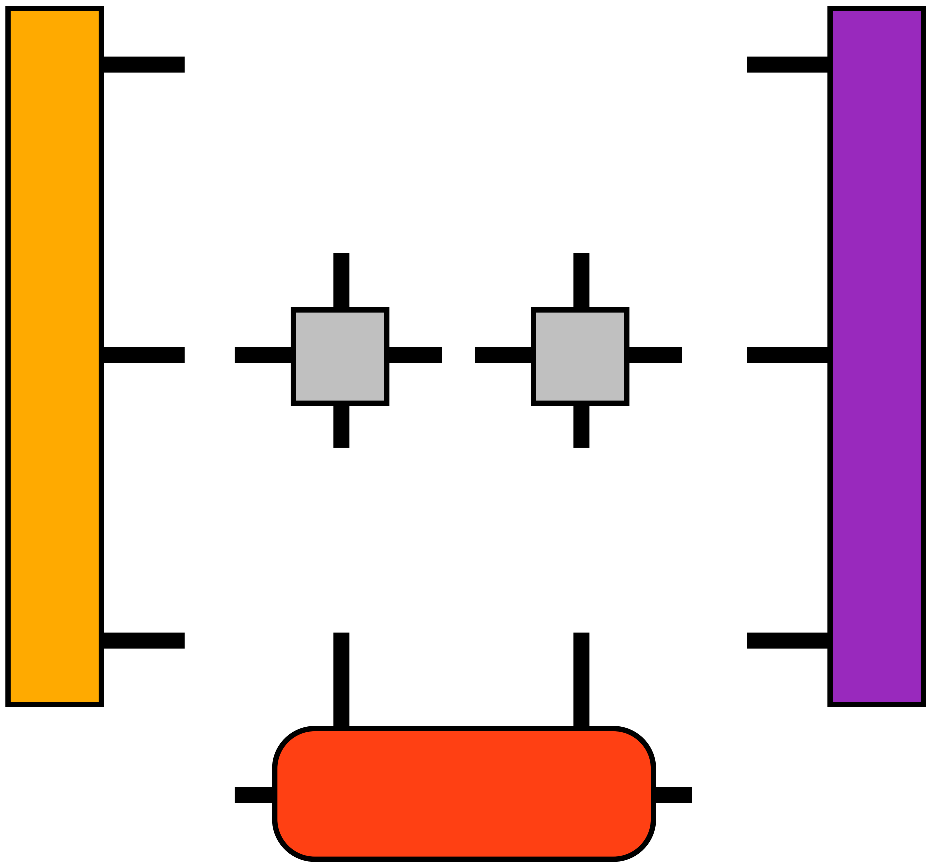
2 ~ m^3

3 ~ m^2

4 ~ m^2

5 ~ m^3

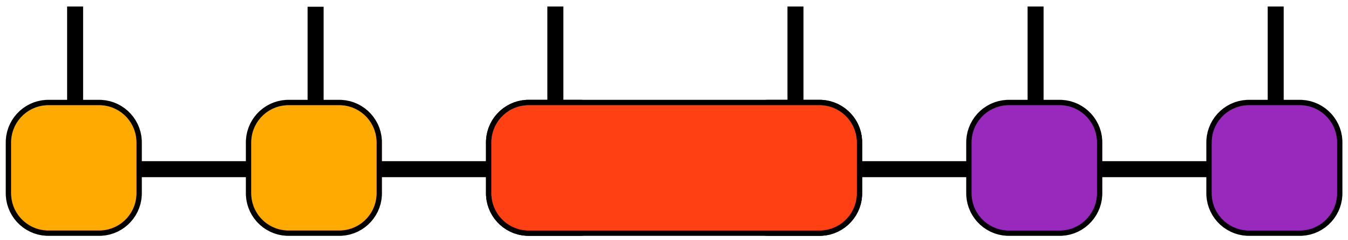
Use Lanczos/Davidson to solve
(sparse matrix eigensolver)



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

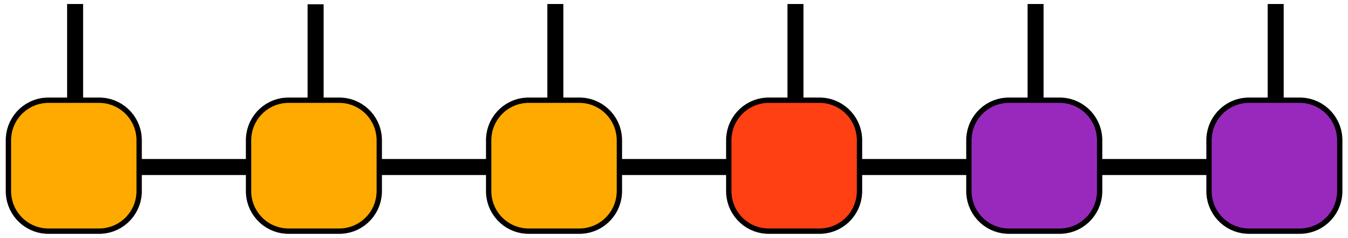
Now, with improved wavefunction, shift orthogonality center (using SVD)

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

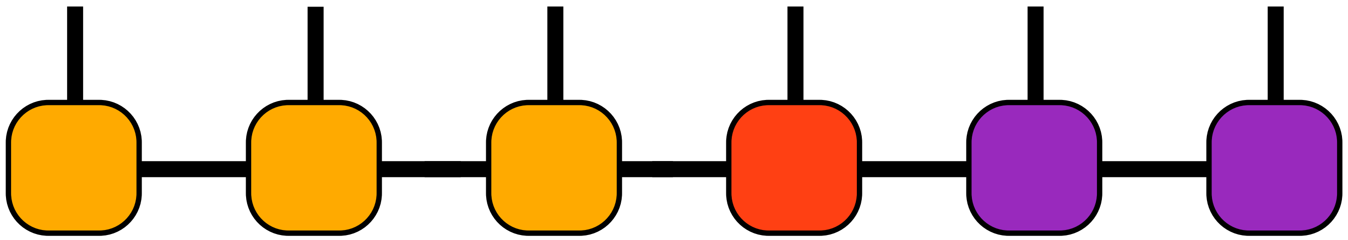


Now, with improved wavefunction,
shift orthogonality center (using SVD)

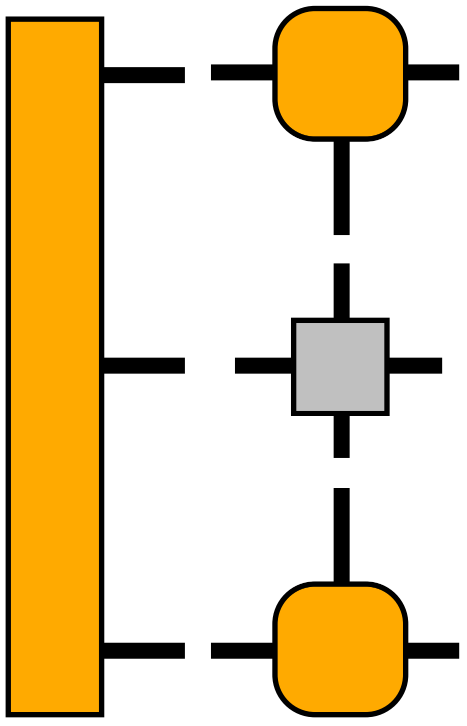
Important to truncate to m singular values
("number of states kept" in DMRG)



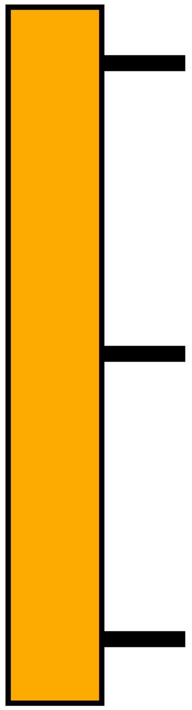
Grow projected Hamiltonian



Grow projected Hamiltonian

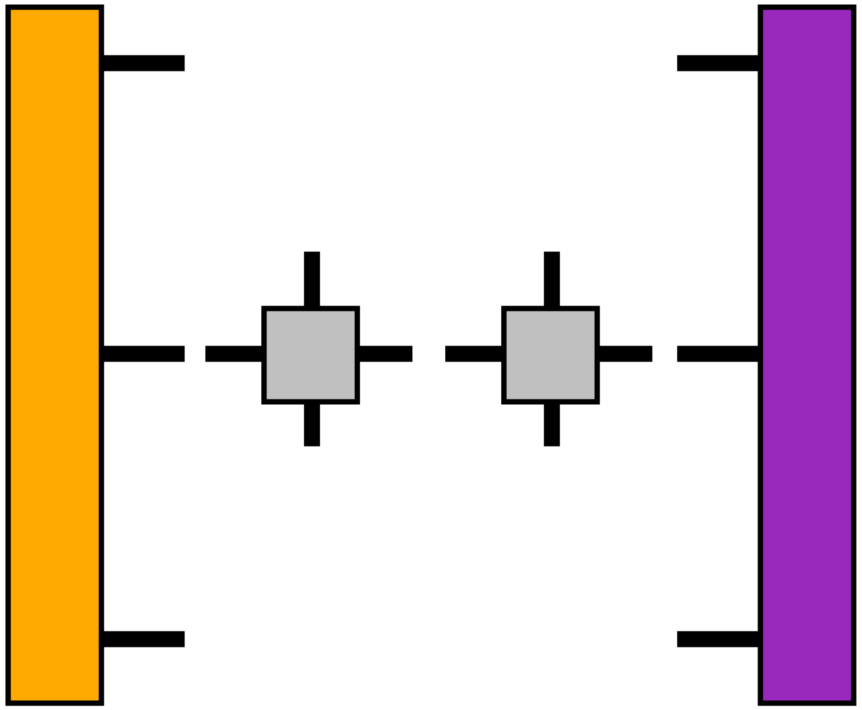


Grow projected Hamiltonian

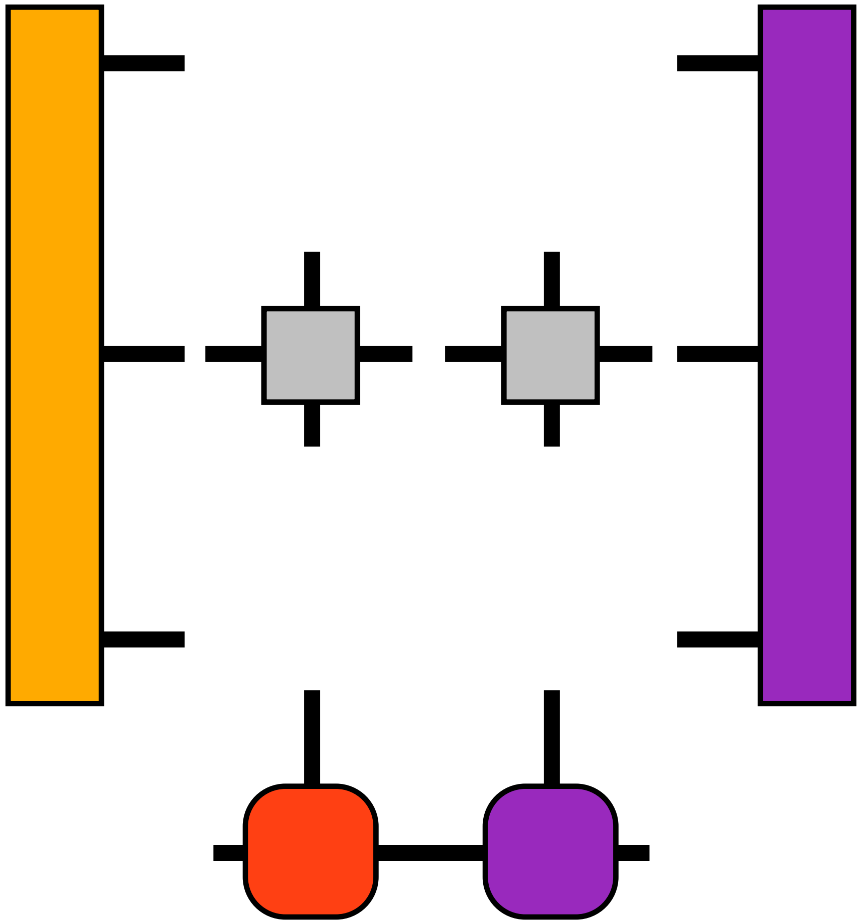


Grow projected Hamiltonian

Recover older projected Hamiltonian saved in memory

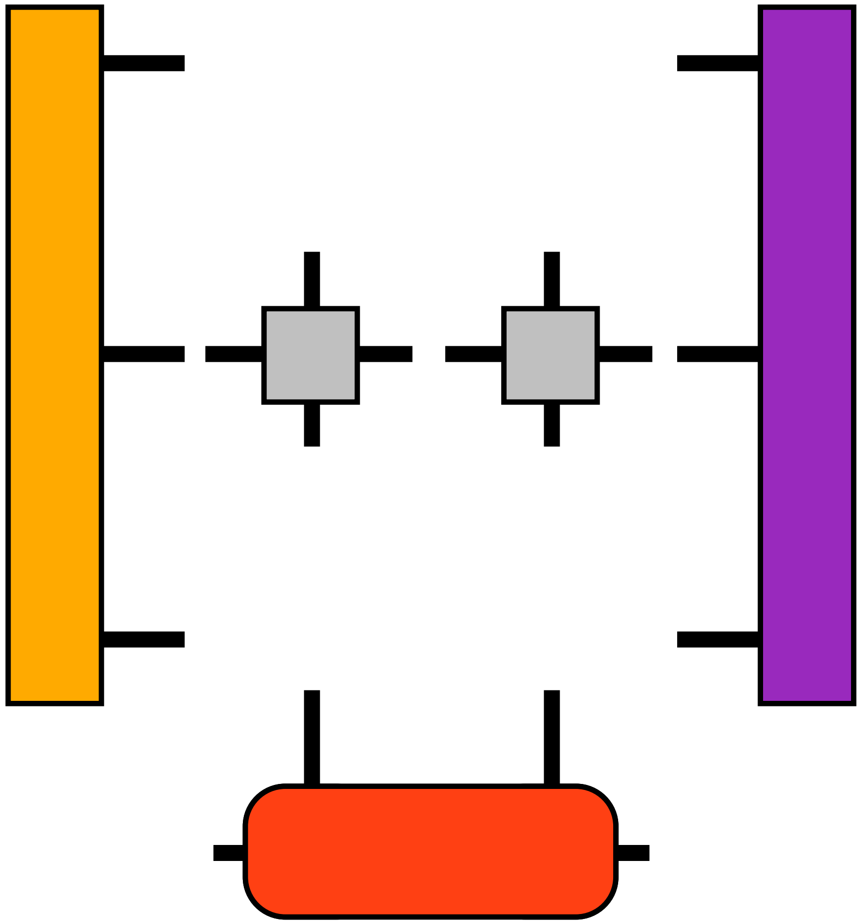


Iterating leads to sweeping procedure



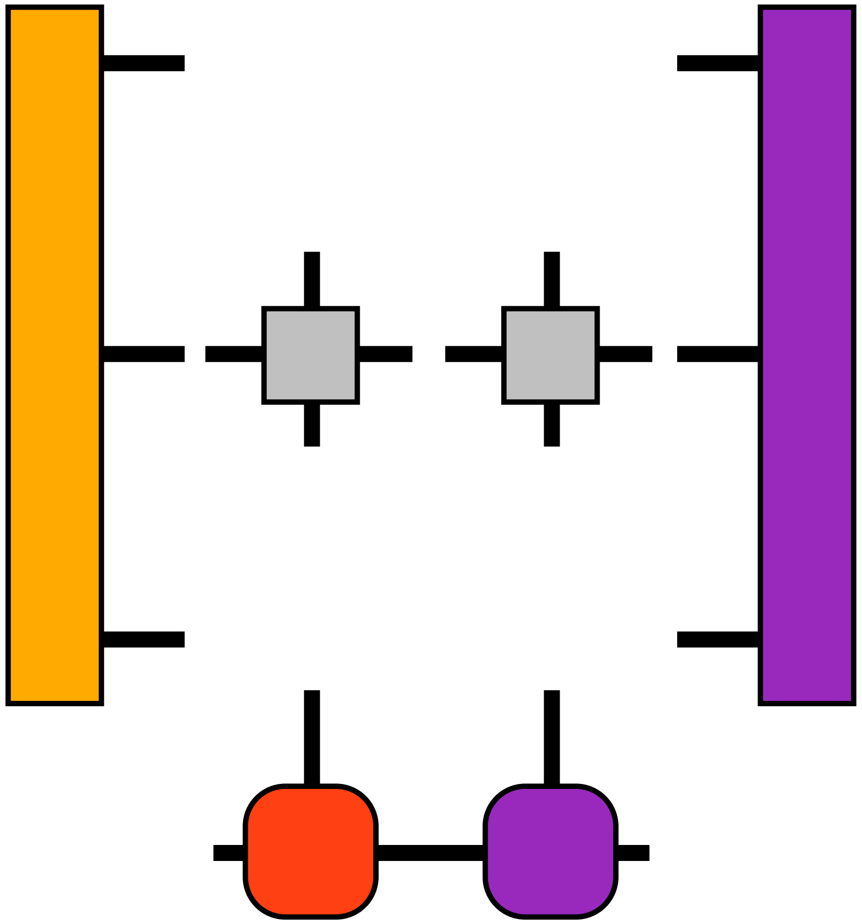
Iterating leads to sweeping procedure

1. Solve eigenproblem



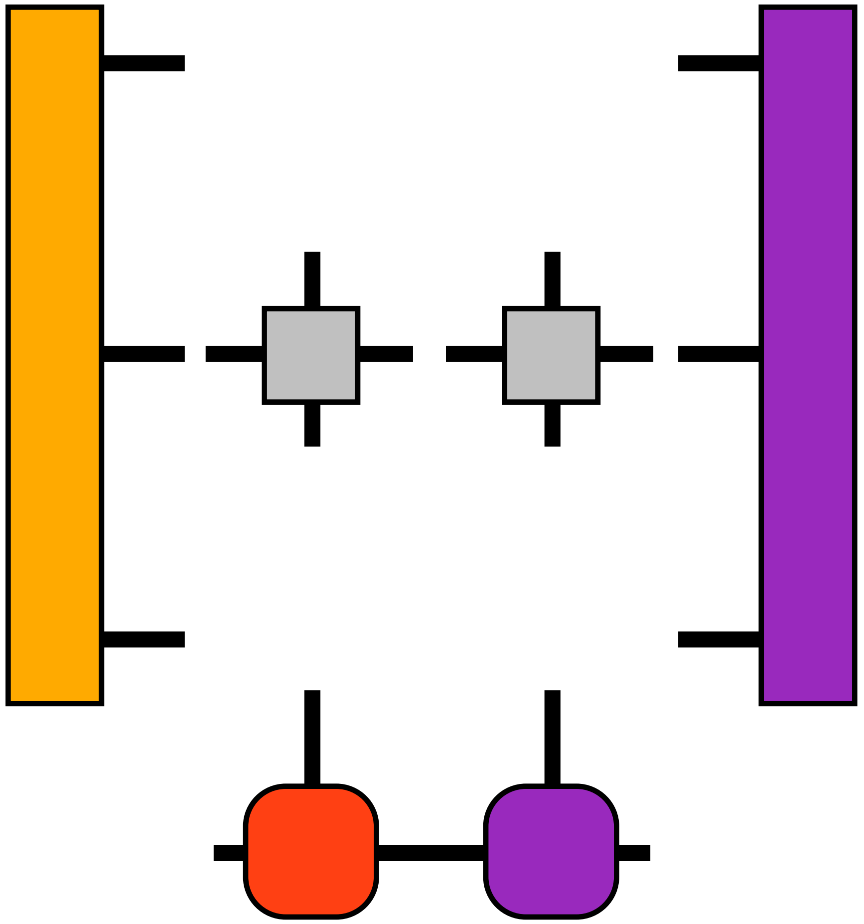
Iterating leads to sweeping procedure

1. Solve eigenproblem
2. SVD wavefunction

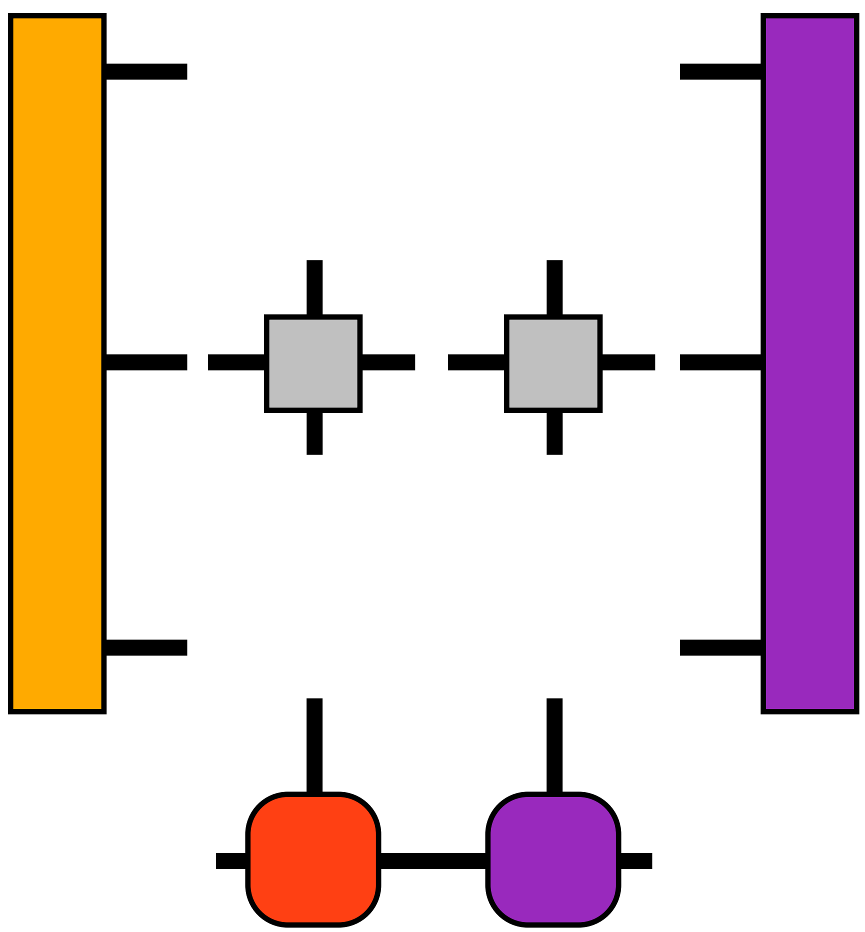


Iterating leads to sweeping procedure

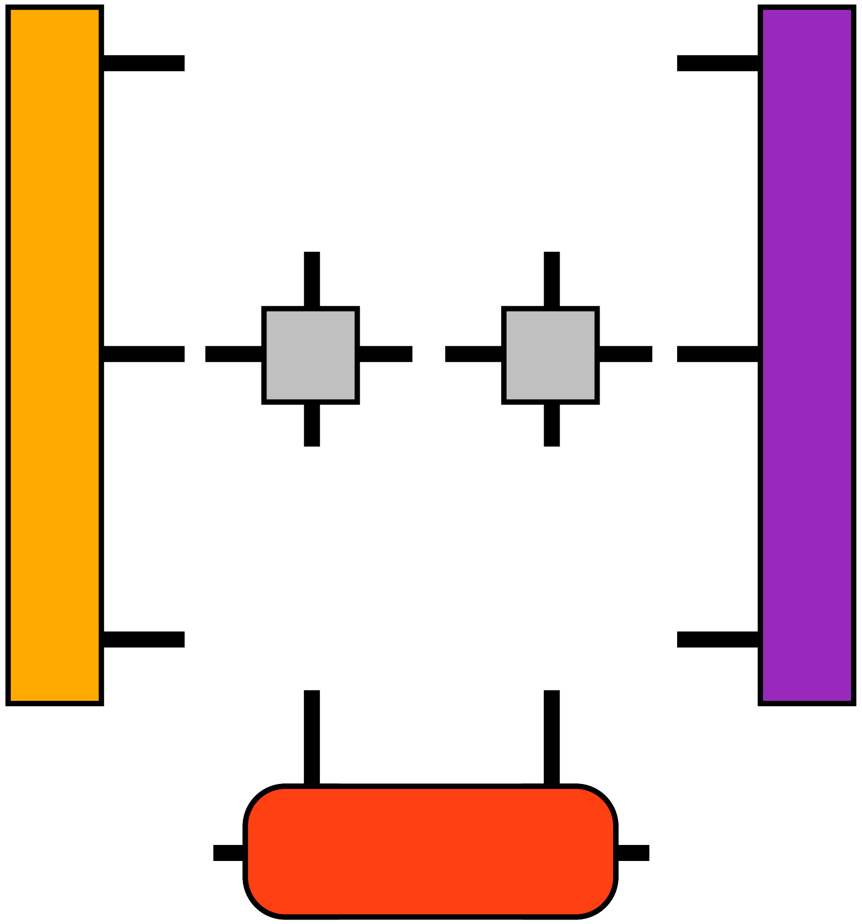
1. Solve eigenproblem
2. SVD wavefunction
3. Grow effective H



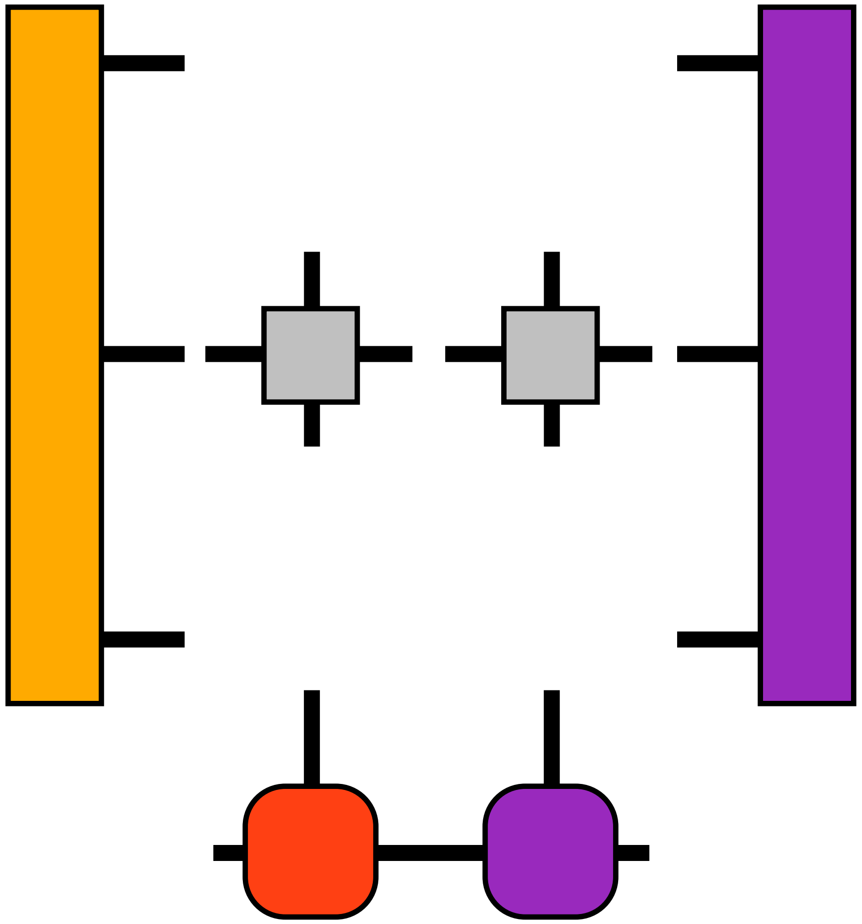
Iterating leads to sweeping procedure



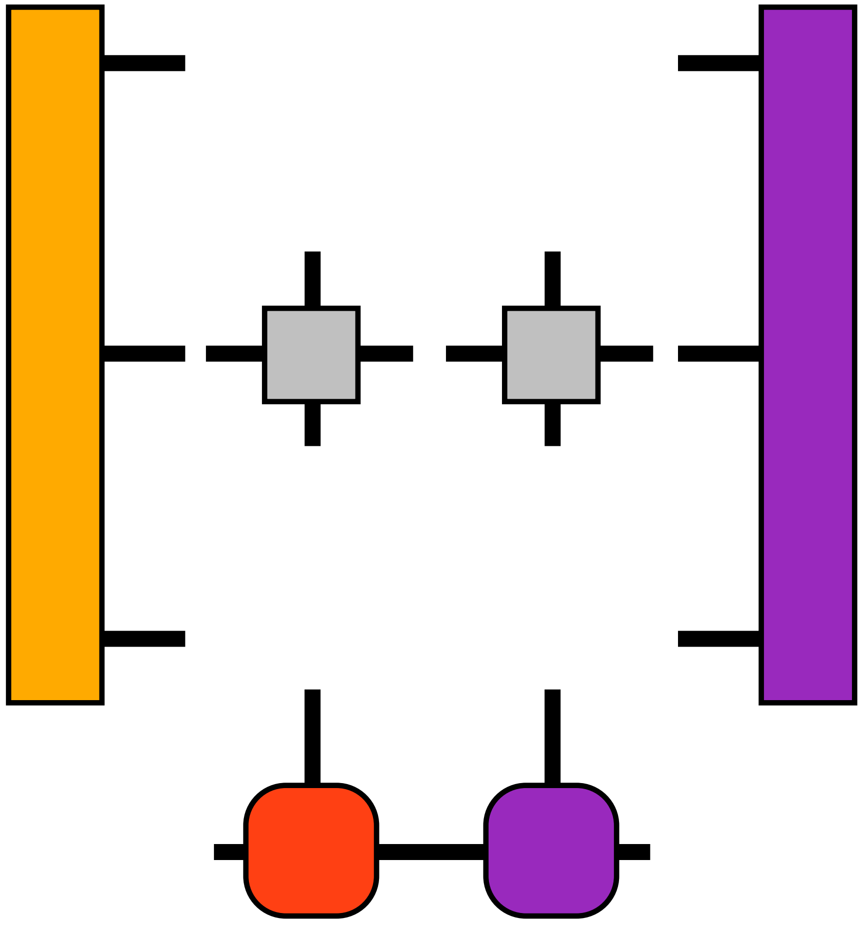
Iterating leads to sweeping procedure



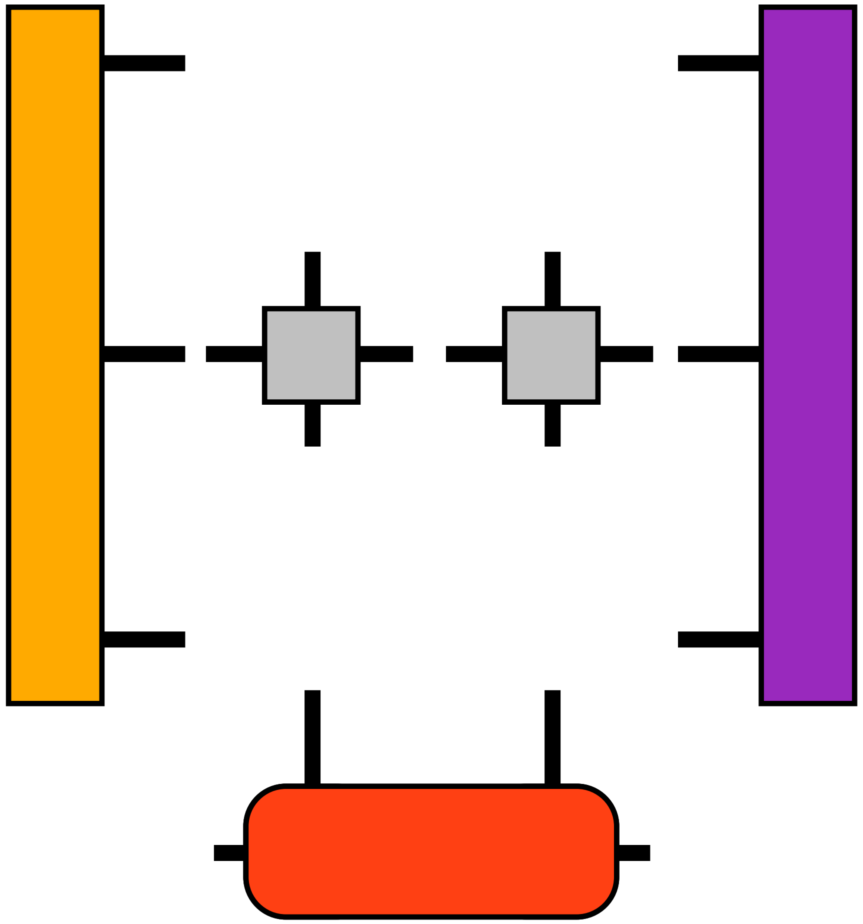
Iterating leads to sweeping procedure



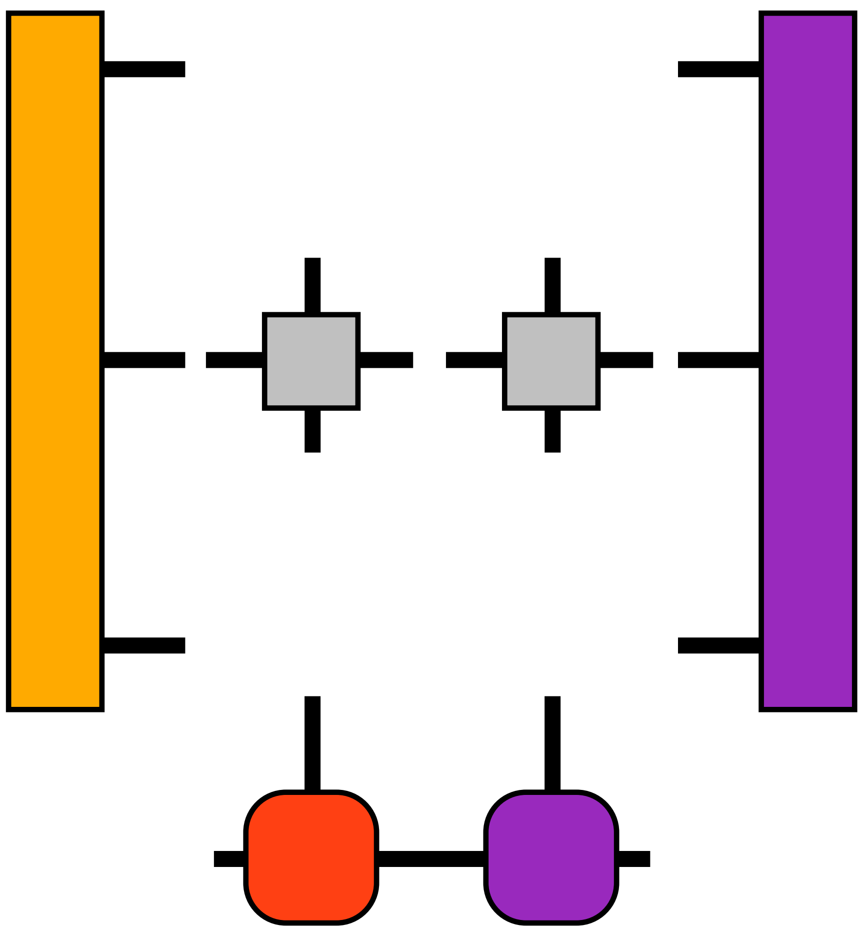
Iterating leads to sweeping procedure



Iterating leads to sweeping procedure



Iterating leads to sweeping procedure



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

`itensor_tutorial/06_DMRG`

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

2. (Line 65) SVD the two-site tensor phi into factors A, D, B. The last argument to svd should be "args" in order to pass truncation parameters:

```
svd(...,args);
```

3. (Lines 75, 85) 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).