01 One Site

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

Single-site basis:

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

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

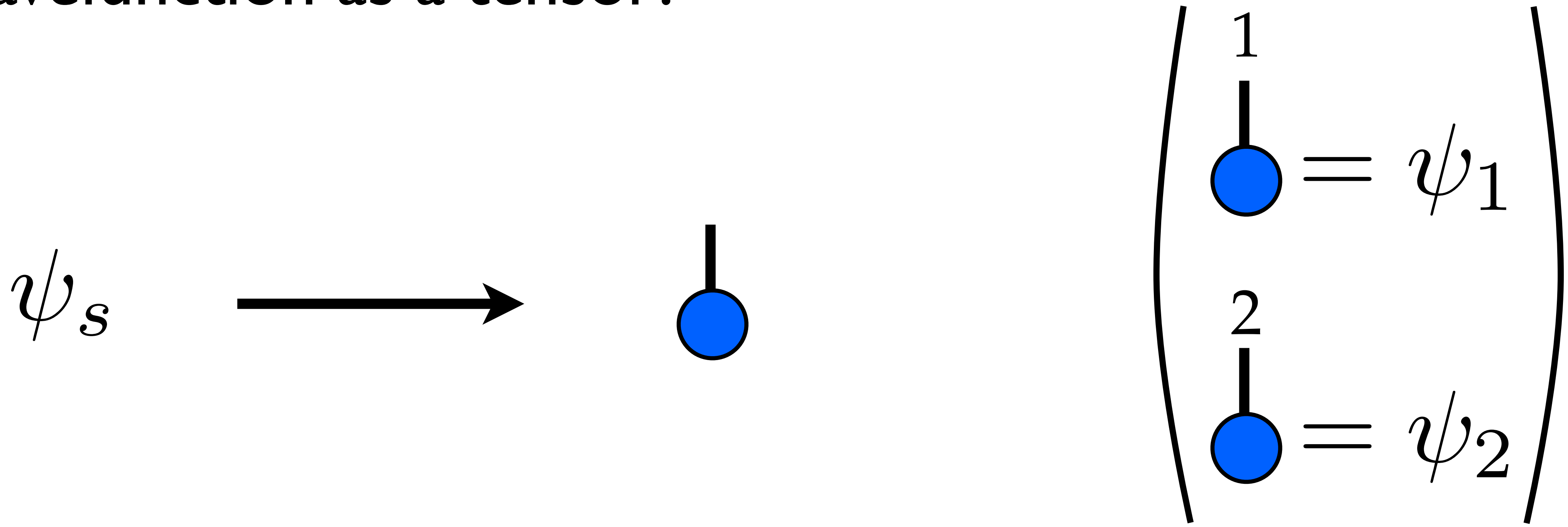
Most general wavefunction for a spin 1/2:

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

The ψ_s are complex numbers.

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

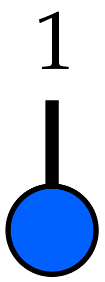
Single-site wavefunction as a tensor:



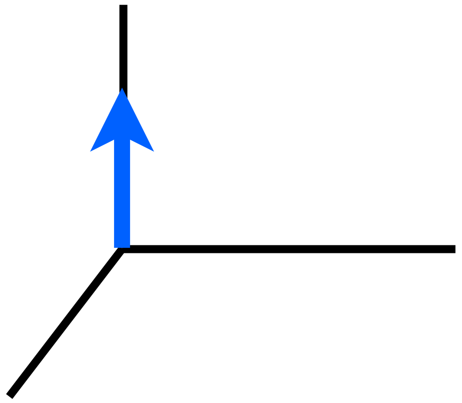
Using ITensor:

```
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 $|\psi\rangle = |\uparrow\rangle$



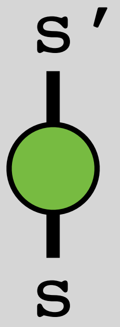
$= 1$



```
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 an operator:

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

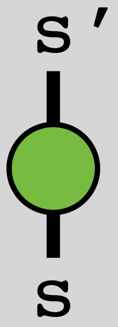


`prime(s)` returns copy of `s` with a “prime level” of `1`

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

Our operator:

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

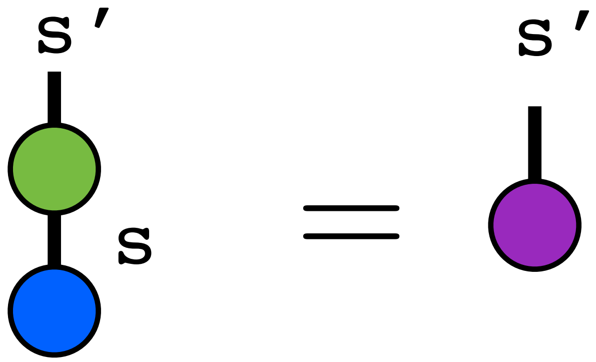


Set its components:

```
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,

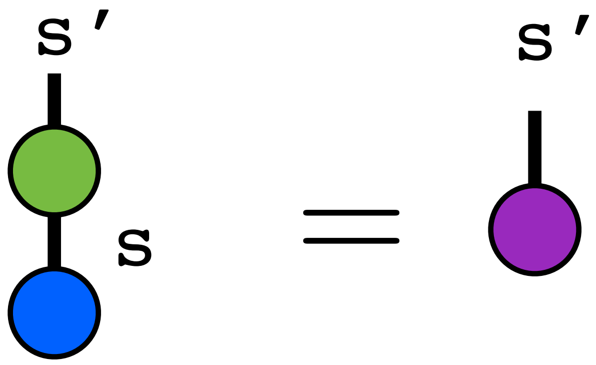
```
ITensor phi = Sx * psi;
```

`*` operator contracts matching indices.

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

What state is phi ?

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

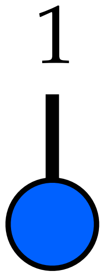


```
ITensor phi = Sx * psi;
PrintData(phi);
```

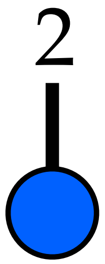
Prints:

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

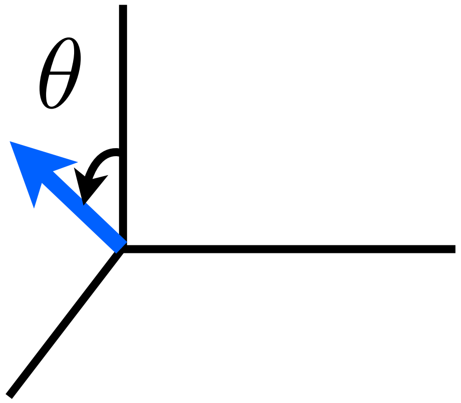
More interesting ψ_s: choose $\theta = \pi/4$ and



$= \cos\theta/2$

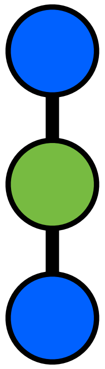


$= \sin\theta/2$



```
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
```

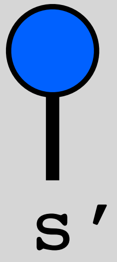
Diagrammatically, measurements (expectation values) look like:



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

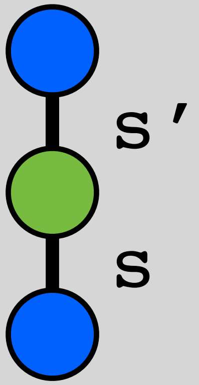
For convenience, make:

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



Calculate expectation values:

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



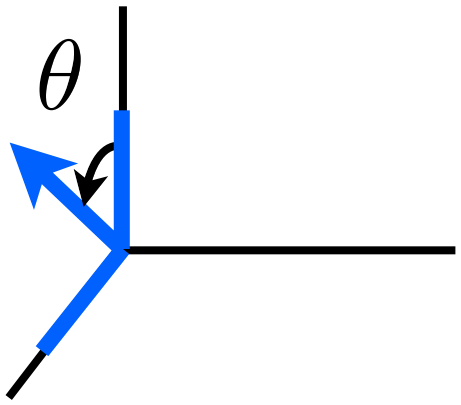
```
Real zz = (cpsi * Sz * psi).toReal();
Real xx = (cpsi * Sx * psi).toReal();
```

Printing the results,

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

we get the output

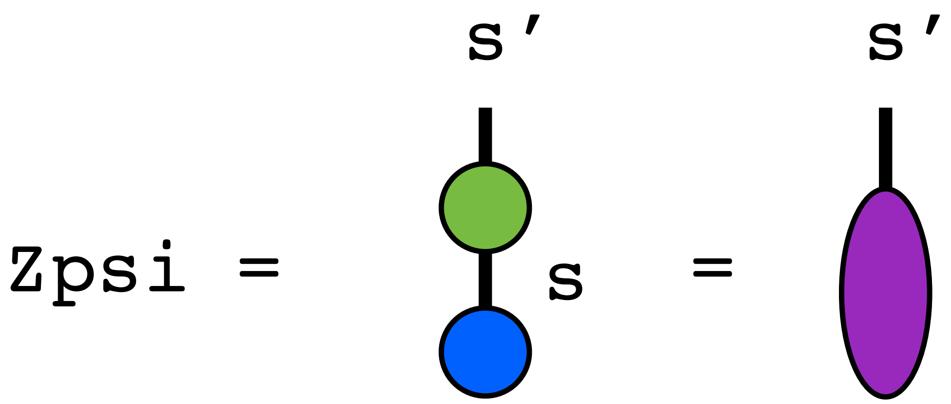
```
<Sz> = 0.35355
<Sx> = 0.35355
```



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

Take a closer look at the tensor contractions:

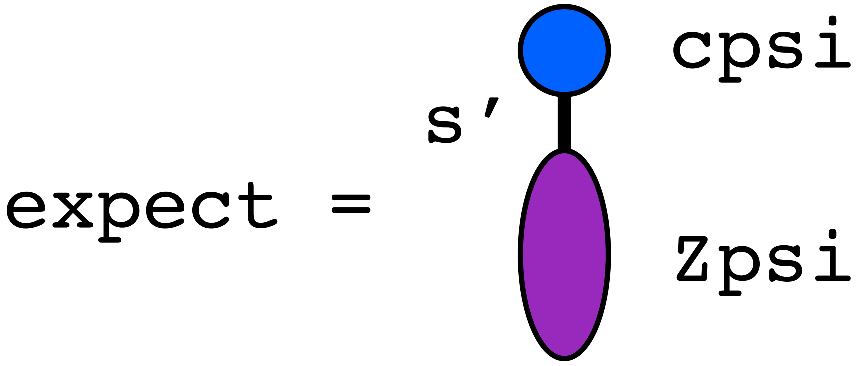
```
ITensor Zpsi = Sz * psi;
```



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;
```

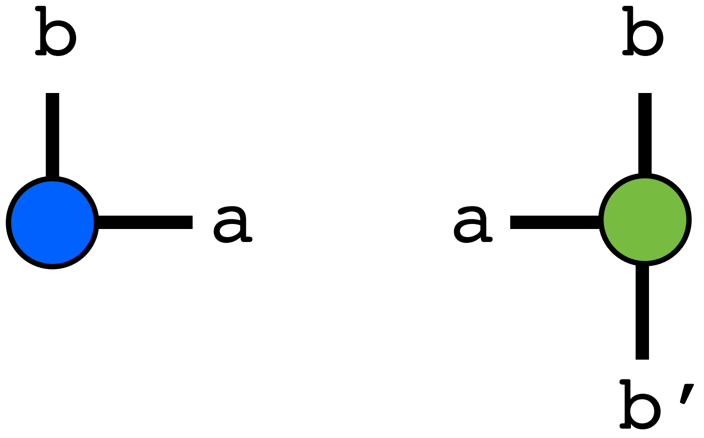
- We can prime an Index b ⟶ b’ using

```
prime(b)
```

- The `*` operator automatically contracts matching Index pairs

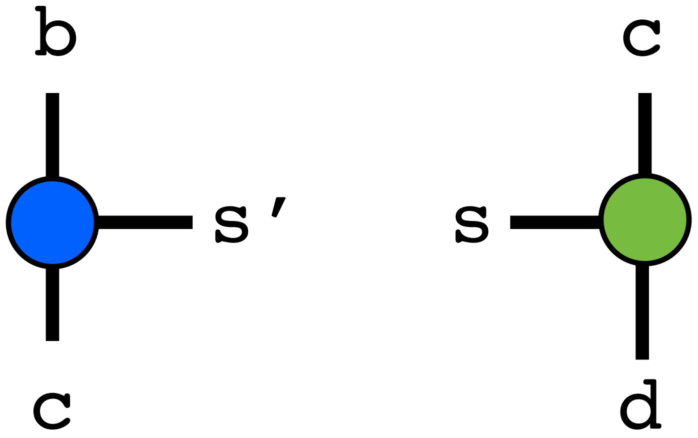
Quiz:

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();
```