Neutral atoms/Rydberg qubits

VQD setup

Set the main directory as the current directory

```
In[56]:= SetDirectory[NotebookDirectory[]];
```

Load the QuESTLink package
One may also use the off-line questlink.m file, change it to the location of the local file

```
In[57]:= Import["https://qtechtheory.org/questlink.m"]
```

This will download a binary file **quest_link** if there is no such file found
Otherwise, use a locally-compiled that called **quest_link***

```
In[58]:= (* Search for existing files that match the pattern "quest_link*" *)
With[{questLinkFiles = Sort@FileNames["quest_link*", {NotebookDirectory[]}]}
  ,
  If[Length[questLinkFiles] > 0,
    (* If one or more matching files are found,
    use the first one alphabetically *)
    Print["Using the existing link file: ", First@questLinkFiles];
    CreateLocalQuESTEnv[First@questLinkFiles];
    ,
    (*If no matching files are found, download the link file*)
    Print["No link file found, download quest_link"];
    CreateDownloadedQuESTEnv[];
   ];
]
```

```
Using the existing link file: /home/cica/VQD/devices/quest_link
```

Load the **VQD** package; must be loaded after QuESTlink is loaded

```
In[59]:= Get["../vqd.wl"]
```

Set the default configuration of the netural atom device

frequency unit: ***MHz***
time unit: ***μs***
distance unit: ***μm*** *(the VQD accepts 2 or 3 dimensional coordinates)*

```
In[60]:= (* some examples of arrays *)
         (* 2d-array of 9 atoms*)
         locs2 = Association@
            MapThread[# → #2 &, {Range[0, 8], Flatten[Table[{i, j}, {i, 0, 2}, {j, 0, 2}], 1]}];
         (* 3d-array of 8 atoms *)
         locs3 = Association@MapThread[# → #2 &,
             {Range[0, 7], Flatten[Table[{i, j, k}, {i, 0, 1}, {j, 0, 1}, {k, 0, 1}], 2]}];

In[62]:= Options[RydbergHub] = {
            (* The total number of atoms/qubit*)
            QubitNum → 9
            ,
            (*Physical location on each qubit described with a 2D- or 3D-vector*)
            AtomLocations → locs2
            ,
            (* It's presumed that T2* has been echoed out to T2 *)
            T2 → 100*10^6
            ,
            (* The life time of vacuum chamber,
            where it affects the coherence time: T1=τvac/N  *)
            VacLifeTime → 100*10^6
            ,
            (* Rabi frequency of the atoms. We assume the duration of multi-
             qubit gates is as long as 4π pulse of single-qubit gates *)
            RabiFreq → 0.1
            ,
            (* Asymmetric bit-flip error probability;
            the error is acquired during single qubit operation *)
            ProbBFRot → <|10 → 0.015, 01 → 0.025|>
            ,
            (* Unit lattice in μm. This will be the unit the lattice and coordinates *)
            UnitLattice → Sqrt@2
            ,
            (* blockade radius of each atom *)
```

```
  BlockadeRadius → 2
  ,
  (* The factor that estimates accelerated dephasing due to moving the
   atoms. Ideally, it is calculated from the distance and speed. *)
  HeatFactor → 10
  ,
  (* Leakage probability during initalisation process *)
  ProbLeakInit → 0.01
  ,
  (* duration of moving atoms;
  we assume SWAPLoc and ShiftLoc take this amount of time: 100 μs *)
  DurMove → 100
  ,
  (* duration of lattice initialization which involves
   the atom loading (~50%) and rearranging the optical tweezer *)
  DurInit → 5 * 10^5
  ,
  (* measurement fidelity and duration, were it induces atom loss afterward *)
  FidMeas → 0.987
  ,
  DurMeas → 10
  ,
  (* The increasing probability of atom loss on each
   measurement. The value keeps increasing until being initialised *)
  ProbLossMeas → 0.05
  ,
  (* leak probability of implementing multi-qubit gates *)
  ProbLeakCZ → <|01 → 0.001, 11 → 0.001 |>
 };
```

Elementary guide

Native gates

Operators

Initialisation and readout

Init_q, M_q

Single-qubit gates

$\text{Rx}_q[\theta]$, $\text{Ry}_q[\theta]$, $\text{Rz}_q[\theta]$, H_q, $\text{SRot}_q\left[\phi, \Delta, \text{dt}\right]$

Two-qubit gates
$\text{CZ}_{\text{q1,q2}}$

Multi-qubit gates
$C_{\text{q1,q2,...}}\left[Z_{\text{qt}}\right]$, $C_{\text{qc}}\left[Z_{\text{q1,q2,...}}\right]$

Register reconfiguration: swap the location of two atoms and shift the location of a bunch of atoms
$\text{SWAPLoc}_{\text{q1,q2}}$, $\text{ShiftLoc}_{\text{q1,q2,...}}$

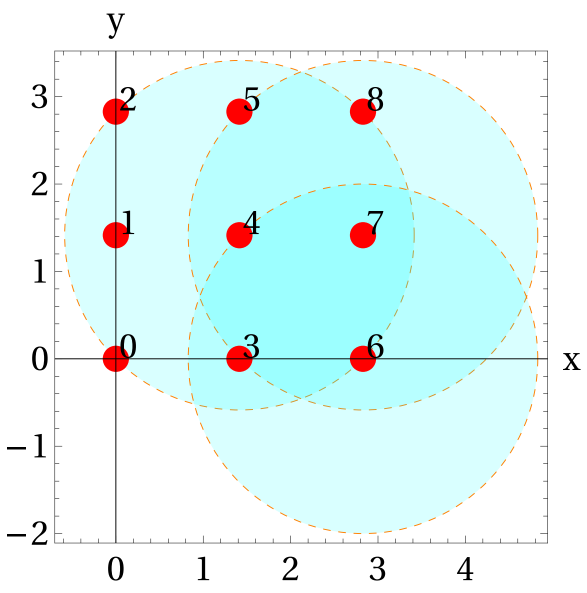
others: doing nothing
$\text{Wait}_q[\text{duration}]$

The 2D- and 3D- dimensional arrays

```
In[63]:= device1 = RydbergHub[];
```

```
In[64]:= PlotAtoms[device1, ImageSize → 300,
          BaseStyle → Directive[18, FontFamily → "Times"], ShowBlockade → {4, 7, 6}]
```

Out[64]=

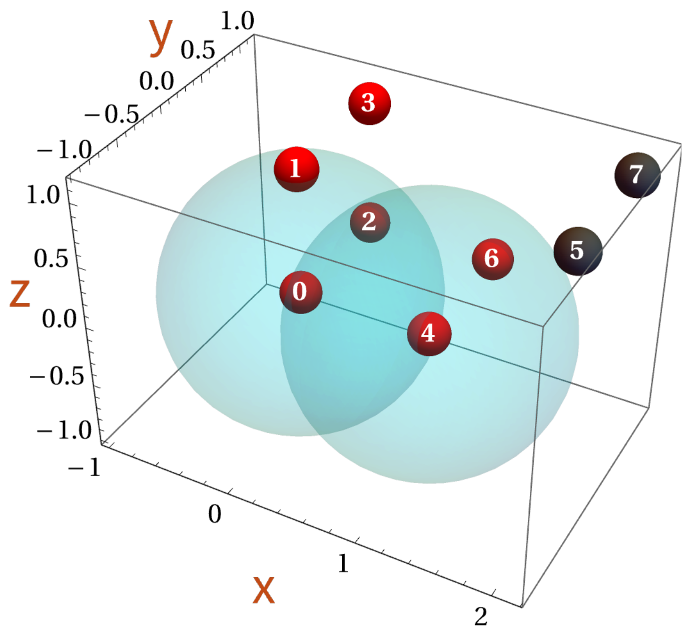


A 3D configuration. Here, we set the loss probability of measurement into 100%, thus, after measuring the atom is lost to the environment.
Set **ShowLossAtoms** to True to show the last position of the atoms before gone missing.

```
In[65]:= device2 = RydbergHub[QubitNum → 8, BlockadeRadius → 1,
             UnitLattice → 1, AtomLocations → locs3, ProbLossMeas → 1];
         InsertCircuitNoise[{ShiftLoc`и"[{1, 0, 0}], M`, M"}, device2];
```

```
In[67]:= plot = PlotAtoms[device2, ImageSize → 350,
           BaseStyle → Directive[14, FontFamily → "Times"],
           ShowBlockade → {0, 4}, ShowLossAtoms → True, LabelStyle → "Section"]
```

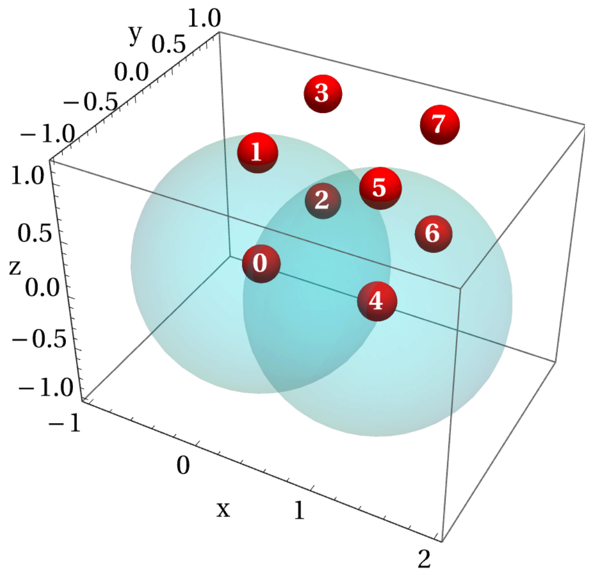
Out[67]=



```
In[68]:= (*Export["rydberg3d.pdf",
          Row@{Show[plot,ViewPoint→{1,-1.9,0}],Show[plot,ViewPoint→{0,-2,1.1}]}]*)
```

Initialisation will put back the atom to the tweezer at the initial configuration

```
In[69]:= InsertCircuitNoise[{Init , Init }, device2];
         PlotAtoms[device2, ImageSize → 300, BaseStyle → Directive[14, FontFamily → "Times"],
          ShowBlockade → {0, 4}, ShowLossAtoms → True]
Out[70]=
```



Show the atoms and reconfiguring the register: PlotAtoms[]

Spatial locations accept 2D and 3D arrangements. Set **ShowBlockade → {qubits}** to show the blockade radius.
Use command **Options[function]**, to see what options that are available to a function.
Also type **?function** to see a short help about the function.

```
In[71]:= Options@PlotAtoms
Out[71]=
         {ShowBlockade → {}, ShowLossAtoms → False}
```

Here we change the number of qubits, location, and the unit of lattice on the fly

```
In[72]:= locs = Association@MapThread[# → #2 &,
            {Range[0, 7], Flatten[Table[{i, j, k}, {i, 0, 1}, {j, 0, 1}, {k, 0, 1}], 2]}]
Out[72]=
         <|0 → {0, 0, 0}, 1 → {0, 0, 1}, 2 → {0, 1, 0}, 3 → {0, 1, 1},
          4 → {1, 0, 0}, 5 → {1, 0, 1}, 6 → {1, 1, 0}, 7 → {1, 1, 1}|>
```

Atoms cannot be moved if place is occupied already. Notice that the atoms moved experience enhance dephasing

```
In[73]:= dev3 = RydbergHub[QubitNum → 8,
            AtomLocations → locs, ProbLossMeas → 1, UnitLattice → 2.1];
         InsertCircuitNoise[{ShiftLoc   [{1, 0, 0}]}, dev3]
```

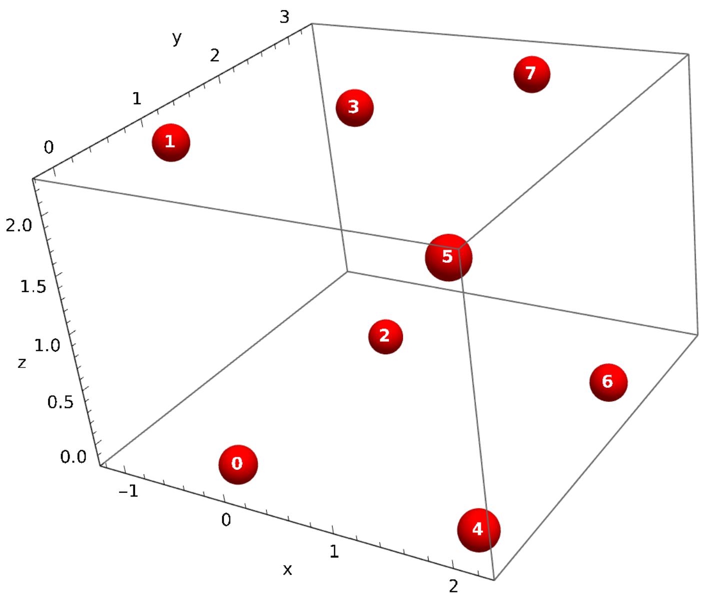
InsertCircuitNoise: Encountered gate $ShiftLoc_{1,7}$[{1, 0, 0}] which is not supported by the given device specification. Note this may be due to preceding gates, if the spec contains constraints which depend on dynamic variables. See ?GetUnsupportedGates.

```
Out[74]=
         $Failed

In[75]:= dev3 = RydbergHub[QubitNum → 8,
            AtomLocations → locs, ProbLossMeas → 1, UnitLattice → 2.1];
         InsertCircuitNoise[{ShiftLoc   [{-.5, .5, 0}]}, dev3]
Out[76]=
```

{{0,
{$Depol_1[5.99998\times10^{-6}]$, $Deph_1[4.99998\times10^{-6}]$, $Depol_7[5.99998\times10^{-6}]$, $Deph_7[4.99998\times10^{-6}]$},
{$Depol_0[5.99998\times10^{-6}]$, $Deph_0[5.\times10^{-7}]$, $Depol_1[0.]$, $Deph_1[0.]$, $Depol_2[5.99998\times10^{-6}]$,
$Deph_2[5.\times10^{-7}]$, $Depol_3[5.99998\times10^{-6}]$, $Deph_3[5.\times10^{-7}]$, $Depol_4[5.99998\times10^{-6}]$,
$Deph_4[5.\times10^{-7}]$, $Depol_5[5.99998\times10^{-6}]$, $Deph_5[5.\times10^{-7}]$,
$Depol_6[5.99998\times10^{-6}]$, $Deph_6[5.\times10^{-7}]$, $Depol_7[0.]$, $Deph_7[0.]$}}, {100, {}, {}}}

```
In[77]:= PlotAtoms[dev3]
Out[77]=
```

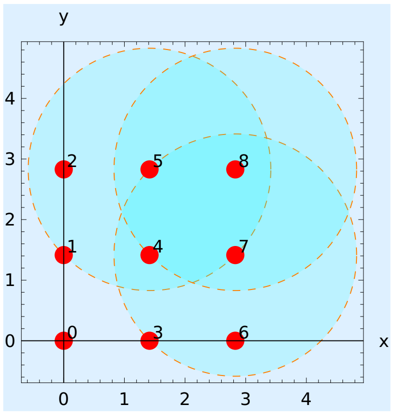


One can modify the plots using **Graphics** options

```
In[78]:= Options@PlotAtoms
Out[78]=
         {ShowBlockade → {}, ShowLossAtoms → False}
```

In[79]:=
```
Options@Graphics
```
Out[79]=
```
{AlignmentPoint → Center, AspectRatio → Automatic, Axes → False,
 AxesLabel → None, AxesOrigin → Automatic, AxesStyle → {}, Background → None,
 BaselinePosition → Automatic, BaseStyle → {}, ColorOutput → Automatic,
 ContentSelectable → Automatic, CoordinatesToolOptions → Automatic,
 DisplayFunction :→ $DisplayFunction, Epilog → {}, FormatType :→ TraditionalForm,
 Frame → False, FrameLabel → None, FrameStyle → {}, FrameTicks → Automatic,
 FrameTicksStyle → {}, GridLines → None, GridLinesStyle → {}, ImageMargins → 0.,
 ImagePadding → All, ImageSize → Automatic, ImageSizeRaw → Automatic,
 LabelStyle → {}, Method → Automatic, PlotLabel → None, PlotRange → All,
 PlotRangeClipping → False, PlotRangePadding → Automatic,
 PlotRegion → Automatic, PreserveImageOptions → Automatic,
 Prolog → {}, RotateLabel → True, Ticks → Automatic, TicksStyle → {}}
```

In[80]:=
```
PlotAtoms[RydbergHub[], ShowBlockade → {5, 7, 8},
 ImageSize → 200, Background → LightBlue]
```
Out[80]=



Arbitrary single rotation

```
Hadamard : ϕ → 0, Δ → Ω, t → π/Ω̃
```

*Here, I assign **Ω** with the default value of **RabiFreq** for practicality. Then I check what matrix produced with **SRot[]** gate given value.*
*I access **Aliases** definition to replace **SRot[]** definition since it is not a native QuESTLink gate by replace command **/.***

In[81]:=
```
Ω = OptionValue[RydbergHub, RabiFreq]
```
Out[81]=
```
0.1
```

```
In[82]:= CalcCircuitMatrix[SRot₀[0, Ω, π / Sqrt[2 Ω²]] /. RydbergHub[][Aliases]] // Chop // MatrixForm
```

Out[82]//MatrixForm=

$$\begin{pmatrix} 0.-0.707107 i & 0.-0.707107 i \\ 0.-0.707107 i & 0.+0.707107 i \end{pmatrix}$$

Rotation around x – axis via **SRot[$\phi \to 0$, $\Delta \to 0$, $t \to \theta / \Omega$]** or directly using **Rx[θ].**

Chop[] *is called to remove the thrilling zeros*

```
In[83]:= CalcCircuitMatrix[Rx₀[π / Ω]] // MatrixForm
```

Out[83]//MatrixForm=

$$\begin{pmatrix} -1.+0. i & 0.-6.12323\times10^{-16} i \\ 0.-6.12323\times10^{-16} i & -1.+0. i \end{pmatrix}$$

```
In[84]:= CalcCircuitMatrix[Rx₀[π]] // Chop // MatrixForm
```

Out[84]//MatrixForm=

$$\begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix}$$

```
In[85]:= CalcCircuitMatrix[SRot₀[0, 0, π / Ω] /. RydbergHub[][Aliases]] // Chop // MatrixForm
```

Out[85]//MatrixForm=

$$\begin{pmatrix} 0 & 0.-1. i \\ 0.-1. i & 0 \end{pmatrix}$$

Multi-qubit gates must fulfill blockade requirement

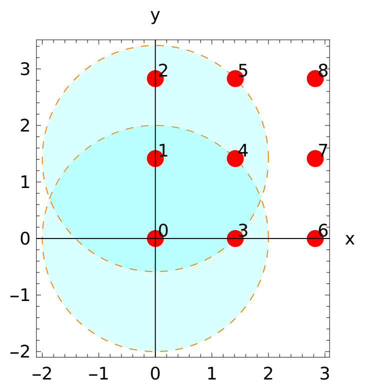
The operation controlled-Z up to a single **qubit** phase ϕ : **inside blockade** vs **outside blockade**

```
In[86]:= CalcCircuitMatrix[CZ   [ϕ] /. RydbergHub[][Aliases]] // MatrixForm
```

Out[86]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{i\phi} & 0 & 0 \\ 0 & 0 & e^{i\phi} & 0 \\ 0 & 0 & 0 & e^{i(-\pi+2\phi)} \end{pmatrix}$$

```
In[87]:= PlotAtoms[RydbergHub[], ShowBlockade → {0, 1}, ImageSize → Small]
Out[87]=
```

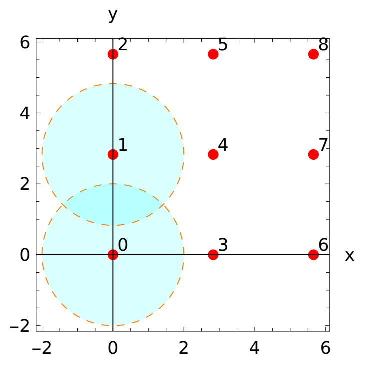


```
In[88]:= InsertCircuitNoise[{CZ_{0,1}[ϕ]}, device1]
Out[88]=
{{0, {CZ_{0,1}[ϕ],
    KrausNonTP_{0,1}[{{{1, 0, 0, 0}, {0, 0.9995, 0, 0}, {0, 0, 0.9995, 0}, {0, 0, 0, 0.9995}}}]},
  {Depol_0[0.], Deph_0[0.], Depol_1[0.], Deph_1[0.], Depol_2[8.48225 × 10^-6], Deph_2[6.28318 × 10^-7],
    Depol_3[8.48225 × 10^-6], Deph_3[6.28318 × 10^-7], Depol_4[8.48225 × 10^-6],
    Deph_4[6.28318 × 10^-7], Depol_5[8.48225 × 10^-6], Deph_5[6.28318 × 10^-7],
    Depol_6[8.48225 × 10^-6], Deph_6[6.28318 × 10^-7], Depol_7[8.48225 × 10^-6],
    Deph_7[6.28318 × 10^-7], Depol_8[8.48225 × 10^-6], Deph_8[6.28318 × 10^-7]}}, {125.664, {}, {}}}
```

*The device **dev** below has a more separated lattice. The atoms are not in the blockade radii, thus, $CZ_{0,1}[\phi]$ gate application becomes illegal and returns error.*

```
In[89]:= dev = RydbergHub[UnitLattice → 0.00001 + 2 × √2];

In[90]:= PlotAtoms[dev, ShowBlockade → {0, 1}, ImageSize → Small]
Out[90]=
```



In[91]:=
```
InsertCircuitNoise[{CZ_{0,1}[ϕ]}, dev]
```

InsertCircuitNoise: Encountered gate $CZ_{0,1}[\phi]$ which is not supported by the given device specification. Note this may be due to preceding gates, if the spec contains constraints which depend on dynamic variables. See ?GetUnsupportedGates.

Out[91]=
```
$Failed
```

Multiqubit gates $C_{c\_}[Z_{t\_}]$ or $C_{c\_\_}[Z_{t\_\_}]$,

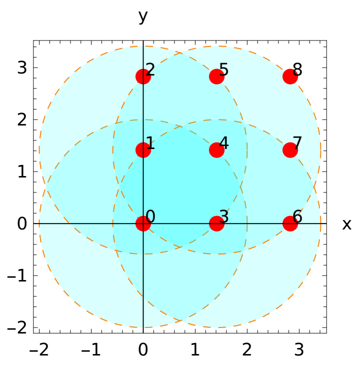
every qubit in cq and tq must be in each other in the blockade radius.

In the following example, qubits {0, 1, 3, 4}, {3, 4, 6, 7}, {5, 4, 7, 8} have overlapping blockade radius; they must produce legit multi – qubit gates.

side note : Variable **j_** accepts input with 1 entry. **j__** accepts input with at least one entry

In[92]:=
```
dev = RydbergHub[];
PlotAtoms[dev, ShowBlockade → {0, 1, 3, 4}, ImageSize → Small]
```

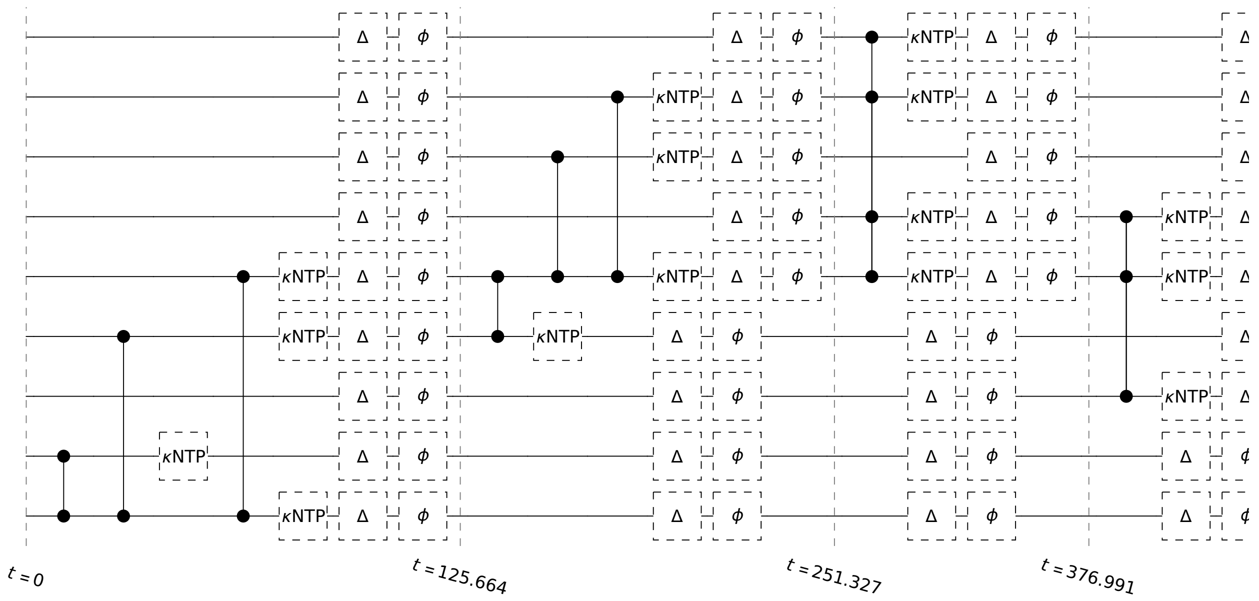
Out[93]=



In[94]:=
```
InsertCircuitNoise[{C_0[Z_{1,3,4}], C_4[Z_{3,6,7}], C_{4,5,7}[Z_8], C_{2,5}[Z_4]}, dev];
```

variable % is useful to pass the outcome of previous executed command

In[95]:=
```
DrawCircuit@%
```
Out[95]=

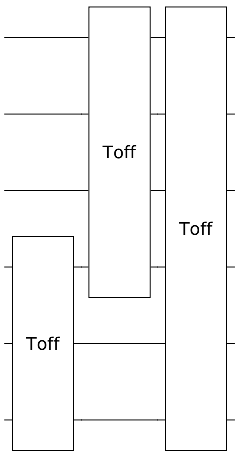


Operations outside native gates and how to verify

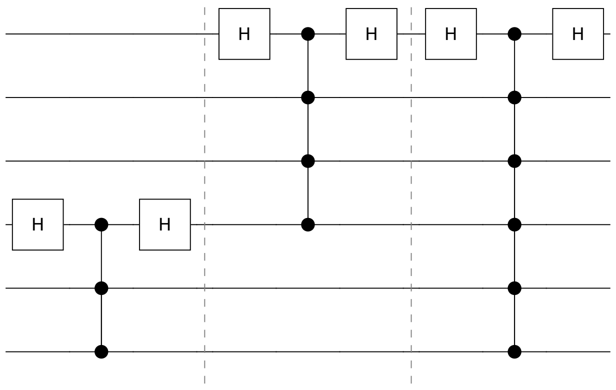
We can define an arbitrary gates above this layer straightforwardly using **ReplaceAll[]**. For example, I will replace simple Toffoli (where the last qubit becomes the target) with Rydberg native multi-z gate and hadamard.

In[96]:=
```
gateRule = {Toff_q_ :→ With[{c = Sequence @@ ({q}⟦ ;; -2⟧), t = Last@{q}}, {H_t, C_c[Z_t], H_t}]};
```

In[97]:=
```
circ = {Toff_{0,1,2}, Toff_{2,3,4,5}, Toff_{0,1,2,3,4,5}};
DrawCircuit[%]
```
Out[98]=



In[99]:=
```
DrawCircuit[circ /. gateRule]
```
Out[99]=



Note that, QuEST is using Least significant bit! so be careful with the indices!

For example, in the case of CNOT gate one commonly sees:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

that is because the indices arranged from behind: **{q0q1...qn},** e.g matrix above has basis {00,01,10,11}. QuEST arrangement is **{qn...q1q0}**! Thus, instead, you will see

In[100]:=
```
cnot = CalcCircuitMatrix[C0[X1]];
```

In[101]:=
```
cnot // MatrixForm
```
Out[101]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

For instance, here is my function to rearrange the matrix to looks like the commonly defined order

In[102]:=
```
rearrange[mat_] := With[{d = Length@mat, nq = Log2[Length@mat]},
  Table[mat⟦Sequence @@ (1 + {FromDigits[Reverse@IntegerDigits[r, 2, nq], 2],
          FromDigits[Reverse@IntegerDigits[c, 2, nq], 2]})⟧, {r, 0, d - 1}, {c, 0, d - 1}]
 ]
```

In[103]:=
```
rearrange[cnot] // MatrixForm
```
Out[103]//MatrixForm=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Before rearrange

In[104]:=
```
CalcCircuitMatrix[Toff0,1,2,3 /. gateRule] // MatrixForm
```
Out[104]//MatrixForm=
```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
```

After rearrange

In[105]:=
```
CalcCircuitMatrix[Toff0,1,2,3 /. gateRule] // rearrange // MatrixForm
```
Out[105]//MatrixForm=
```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Spatial operations

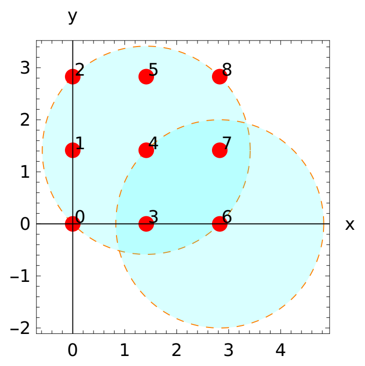
We can do consecutive commands. The instance **dev** will store previous state from the lass **InsertCircuitNoise[]** call.

In[106]:=
```
dev = RydbergHub[];
```

In[107]:=

```
PlotAtoms[dev, ImageSize → Small, ShowBlockade → {4, 6}]
```

Out[107]=



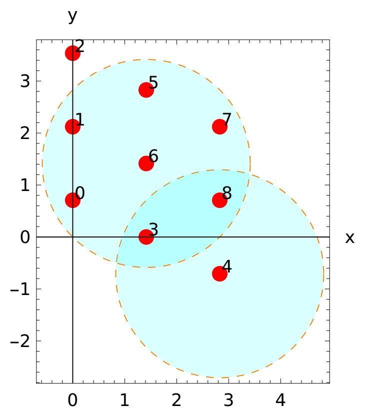
In[108]:=

```
InsertCircuitNoise[
  {ShiftLoc_{0,1,2}[{0, 0.5}], SWAPLoc_{8,7}, Wait_0[.1], ShiftLoc_{7,8,6}[{0, -0.5}], SWAPLoc_{4,6}}, dev];
```

In[109]:=

```
PlotAtoms[dev, ImageSize → Small, ShowBlockade → {4, 6}]
```

Out[109]=



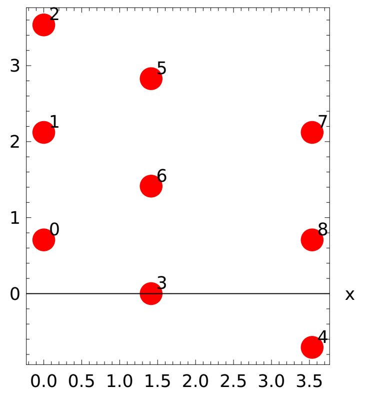
In[110]:=

```
InsertCircuitNoise[{ShiftLoc_{7,8,4}[{0.5, 0}]}, dev];
```

In[111]:=

```
PlotAtoms[dev, ImageSize → Small]
```

Out[111]=

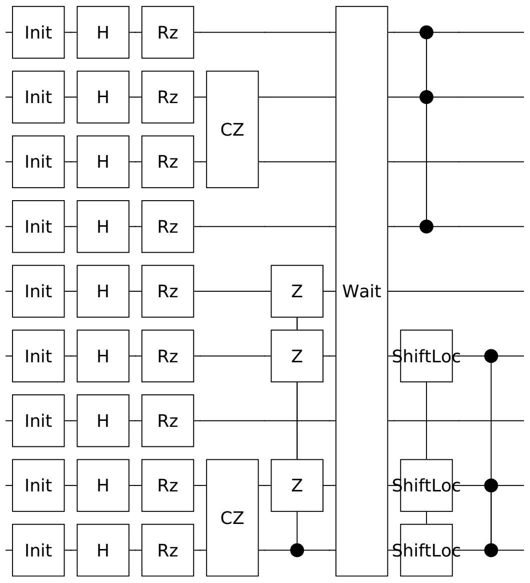


Scheduling : Rearrange circuit by parallel and serial

In[112]:=

```
circ = Flatten@{{Init#, H#, Rz#[π / (# + 1)]} & /@ Range[0, 8], CZ6,7[π],
    CZ0,1[π], C0[Z1,3,4], WaitRange[0,8][ϕ], ShiftLoc0,1,3[{-1, 0}], C0,1[Z3], C5,8[Z7]};
DrawCircuit@%
```

Out[113]=



Parallel excution, where paralellism applies when the operations are done without non-overlaping blockade radius (future)
At the moment, it applies when it is not a multi-qubit gate. But initialisation is in parallel in nature.

In[114]:=

```
Options@CircRydbergHub
```

Out[114]=

```
{Parallel → False}
```

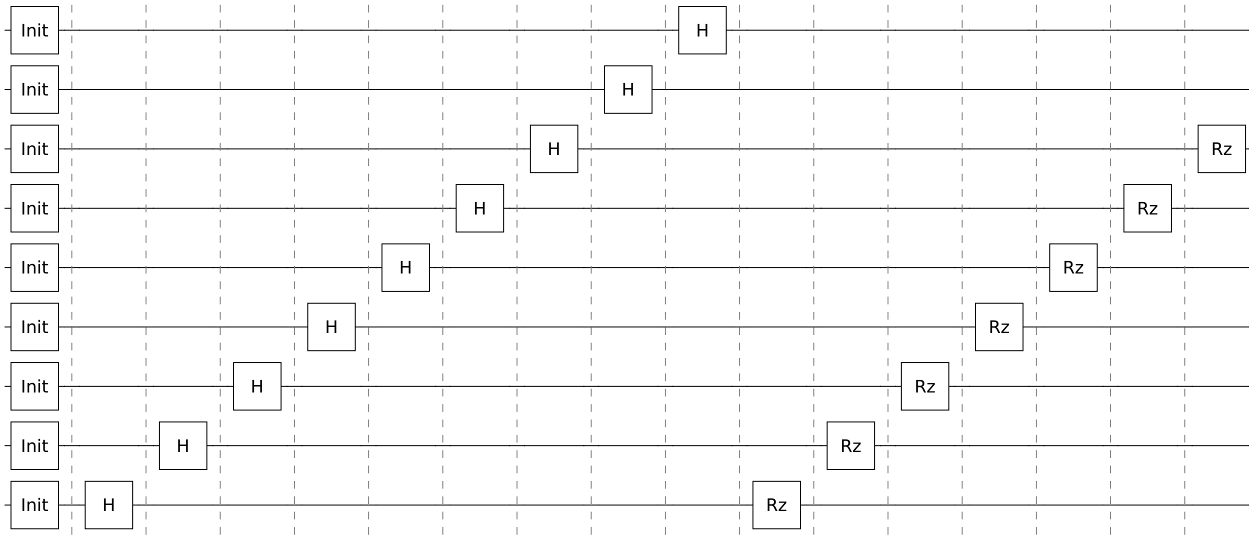
In[115]:=

```
serialcirc = CircRydbergHub[circ, RydbergHub[]];
```

In[116]:=

```
DrawCircuit@serialcirc
```

Out[116]=



Serial excution. We rearrange a list of gates into a list of list {{... }, {...}, ... }.
The gates within the same inner list { ... } are executed in parallel. The schedule time is taken based on the maximal duration of the gates among the inner list.
One may edit this manually.

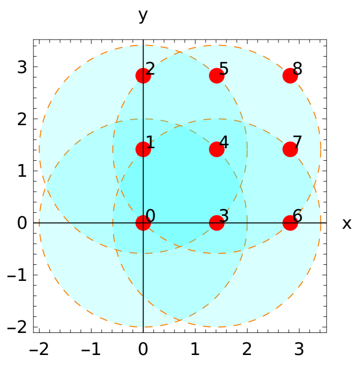
In[117]:=

```
dev = RydbergHub[];
```

In[118]:=

```
PlotAtoms[dev, ImageSize → Small, ShowBlockade → {0, 1, 3, 4}]
```

Out[118]=



In[119]:=

```
parallelcirc = CircRydbergHub[circ, dev, Parallel → True]
```

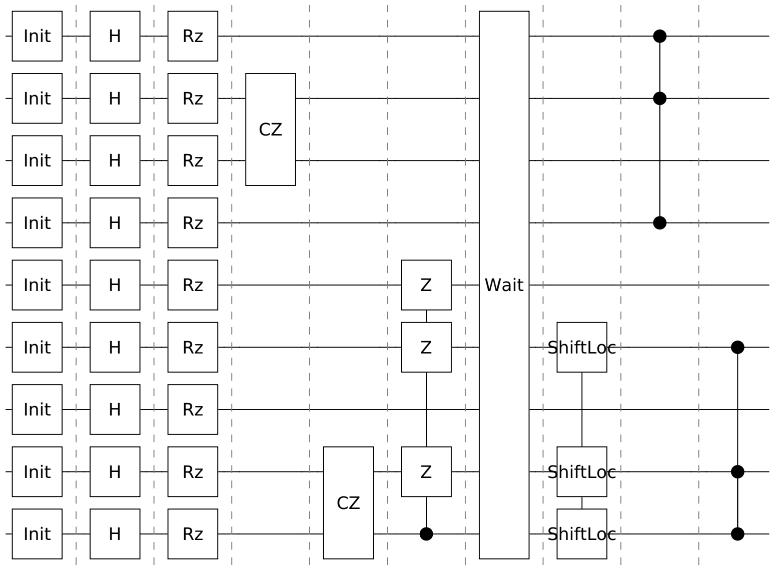
Out[119]=

$\{\{Init_0, Init_1, Init_2, Init_3, Init_4, Init_5, Init_6, Init_7, Init_8\},$
$\{H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8\},$
$\{Rz_0[\pi], Rz_1[\frac{\pi}{2}], Rz_2[\frac{\pi}{3}], Rz_3[\frac{\pi}{4}], Rz_4[\frac{\pi}{5}], Rz_5[\frac{\pi}{6}], Rz_6[\frac{\pi}{7}], Rz_7[\frac{\pi}{8}], Rz_8[\frac{\pi}{9}]\}, \{CZ_{6,7}[\pi]\},$
$\{CZ_{0,1}[\pi]\}, \{C_0[Z_{1,3,4}]\}, \{Wait_{\{0,1,2,3,4,5,6,7,8\}}[\phi]\}, \{ShiftLoc_{0,1,3}[\{-1, 0\}]\}, \{C_{5,8}[Z_7]\}, \{C_{0,1}[Z_3]\}\}$

In[120]:=

```
DrawCircuit@%
```

Out[120]=



For example, I execute the hadamards in pair for some reason.

In[121]:=

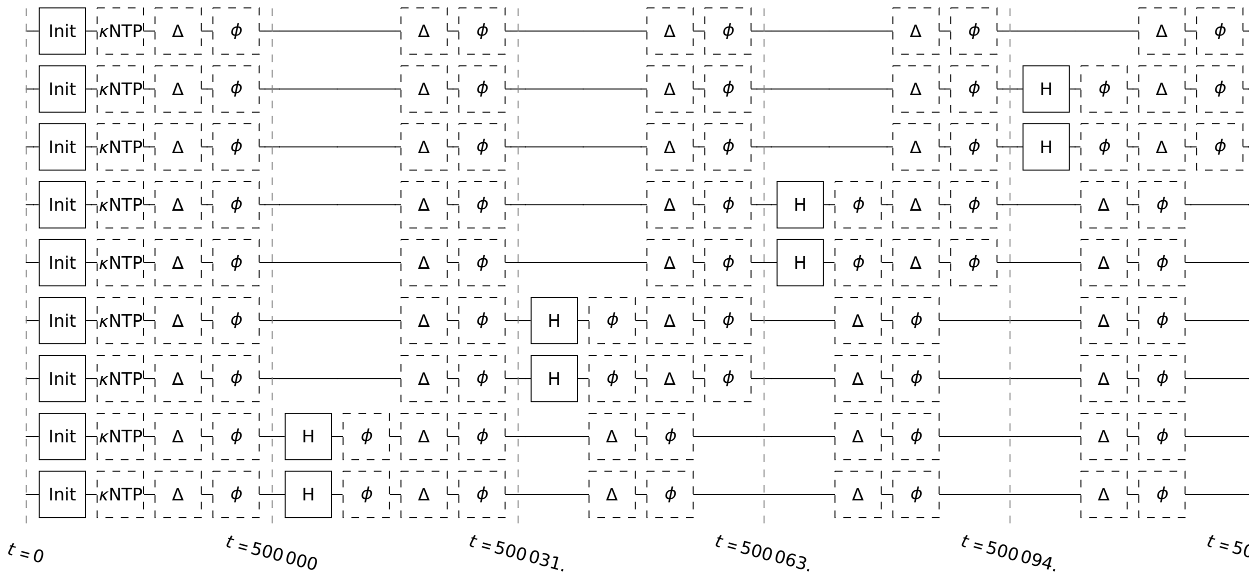
serialcirc2 = $\{\{Init_0, Init_1, Init_2, Init_3, Init_4, Init_5, Init_6, Init_7, Init_8\},$
$\{H_0, H_1\}, \{H_2, H_3\}, \{H_4, H_5\}, \{H_6, H_7\}, \{H_8\}, \{Rz_0[\pi]\}, \{Rz_1[\frac{\pi}{2}]\}, \{Rz_2[\frac{\pi}{3}]\}, \{Rz_3[\frac{\pi}{4}]\},$
$\{Rz_4[\frac{\pi}{5}]\}, \{Rz_5[\frac{\pi}{6}]\}, \{Rz_6[\frac{\pi}{7}]\}, \{Rz_7[\frac{\pi}{8}]\}, \{Rz_8[\frac{\pi}{9}]\}, \{CZ_{0,1}[\pi]\}, \{CZ_{6,7}[\pi]\},$
$\{C_0[Z_{1,3,4}]\}, \{Wait_{\{0,1,2,3,4,5,6,7,8\}}[\phi]\}, \{ShiftLoc_{0,1,3}[\{-1, 0\}]\}, \{C_{5,8}[Z_7]\}, \{C_{0,1}[Z_3]\}\};$

Get noise-decorated circuit from rearranged circuit, where hadamard are run in pairs, in parallel

In[122]:=

```
DrawCircuit@InsertCircuitNoise[serialcirc2, dev]
```

Out[122]=



Example: quantum simulation on creating 9-GHZ

In[123]:=

```
ghz = {
    Init₀, Init₁, Init₂, Init₃, Init₄, Init₅, Init₆, Init₇, Init₈,
    H₀,
    H₁, H₃, H₄, C₀[Z₁,₃,₄], H₁, H₃, H₄,
    H₆, H₇, C₃[Z₆,₇], H₆, H₇,
    H₅, H₈, C₇[Z₅,₈], H₅, H₈,
    H₂, C₁[Z₂], H₂
  };
```

In[124]:=

```
(* allocate memory *)
ψ = CreateQureg[9];
ρ = CreateDensityQureg[9];
```

The non-native questlink gates are defined in the Aliases, so we need to replace those aliases into the native questlink gates. Apply the circuit in the state vector, noiseless case (note that we remove the damping here because it's vector simulation)

In[126]:=

```
ApplyCircuit[InitZeroState@ψ, Flatten[ghz /. dev[Aliases] /. {Damp_q_[_] :→ Id_q}]];
```

Initialise the qubits in a random mixed state: extremely low fidelity. Note that CalcFidelity accepts

density matrix and state vector. It cannot compare two density matrices.

In[127]:=
```
SetQuregMatrix[ρ, RandomMixState[9]];
CalcFidelity[ρ, ψ]
```
Out[128]=
```
0.00194946
```

Then apply the circuit in serial manner

In[129]:=
```
dev = RydbergHub[];
ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[
    CircRydbergHub[ghz, dev, Parallel → False], dev, ReplaceAliases → True]];
```
In[131]:=
```
CalcFidelity[ρ, ψ]
```
Out[131]=
```
0.907664
```

Paper Supplement

Here we replicate the experiment in *: www.nature.com/articles/s41586-022-04592-6*
See **Graphstate1D.nb** and **Steane7.nb** in folder **supplement/GraphStatesonRydbergHub** for the complete simulation code

1D graph state generation

In[132]:=
```
(* memory initialisation*)
DestroyAllQuregs[];
ρ = CreateDensityQureg[12];
ρinit = CreateDensityQureg[12];
ρwork = CreateDensityQureg[12];
```

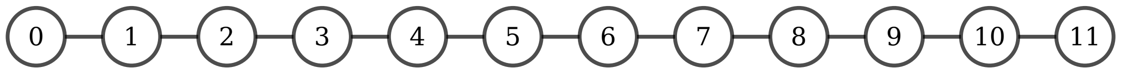
Plots

In[136]:=
```
(*returns graph state stabilizer of a node*)
stabgs[graph_, node_] :=
 With[{neig = Complement[VertexList@NeighborhoodGraph[graph, node], {node}]},
  ToExpression[StringRiffle[Join[{X_node}, Z_# & /@ neig]]]]
```

In[137]:=

```
g = Graph[# ⟷ # + 1 & /@ Range[0, 10]];
Graph[g, VertexSize → 0.6, VertexStyle → Directive[White, EdgeForm[Thick]],
 BaseStyle → {13, FontFamily → "Serif"}, ImageSize → 600,
 EdgeStyle → Directive[Black, Thick], VertexLabels → Placed[Automatic, Center]]
(*Export["graph1d.pdf",%]*)
```

Out[138]=



Default device configuration

In[139]:=

```
Options[RydbergHub] = {
   QubitNum → 12,
   AtomLocations → Association@Table[j → {j, 0}, {j, 0, 11}],
   T2 → 1.5 * 10^6,
   VacLifeTime → 48 * 10^6,
   RabiFreq → 1,
   ProbBFRot → <|10 → 0.001, 01 → 0.03|>,
   UnitLattice → 3,
   BlockadeRadius → 1,
   ProbLeakInit → 0.001,
   DurInit → 5 * 10^5,
   DurMove → 100,
   HeatFactor → 10,
   FidMeas → 0.975,
   DurMeas → 10,
   ProbLossMeas → 0.0001,
   ProbLeakCZ → <|01 → 0.01, 11 → 0.0001|>
  };
```

Plots generation

In[140]:=

```
ClearAll[showgs]
```

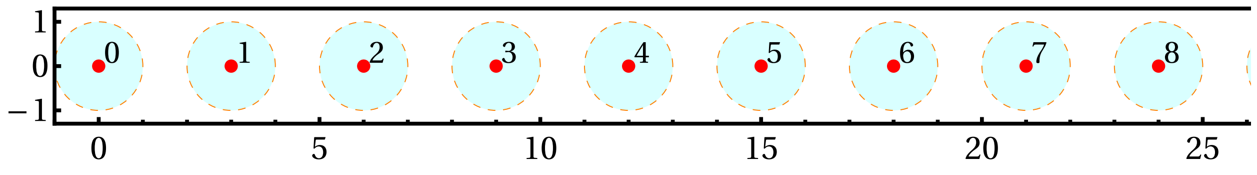
In[141]:=

```
showgs[title_ : "", opt_ : {}] := PlotAtoms[devGS, Sequence @@ opt, ImageSize → 900,
   ShowBlockade → Range[0, 11], LabelStyle → Directive[17, Black],
   BaseStyle → {16, FontFamily → "Times"}, PlotRange → {{-1, 37}, {-1.3, 1.3}},
   Epilog → Inset[Style[title, {Purple, Italic}], Scaled[{0.96, 0.15}]],
   Frame → True, Axes → False, FrameStyle → Directive[Black, Thick],
   FrameTicks → {{{-1, 0, 1}, None}, {Automatic, None}}
  ];
```

In[142]:=

```
devGS = RydbergHub[];
showgs["init"]
```

Out[143]=



In[144]:=

```
circ1 = CircRydbergHub[
   Flatten@{{Init#, Ry#[π / 2]} & /@ Range[0, 11]}, RydbergHub[], Parallel → True];
circ2 = {{ShiftLocSequence@@Range[1,11,2][{-0.75, 0}]}};
circ3 = {C#[Z#+1] & /@ Range[0, 10, 2]};
circ4 = {{ShiftLocSequence@@Range[1,11,2][{1.5, 0}]}};
circ5 = {C#[Z#+1] & /@ Range[1, 9, 2]};
```

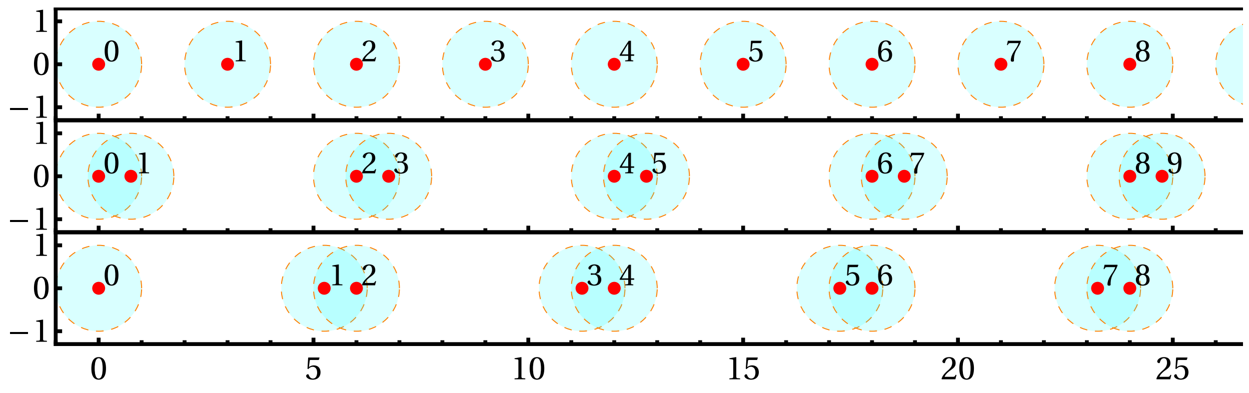
In[149]:=

```
devGS = RydbergHub[];
f1 = showgs["initial", {ImagePadding → {{30, 20}, {0, 0}}}];
noisycirc1 = InsertCircuitNoise[circ1, devGS];
noisycirc2 = InsertCircuitNoise[circ2, devGS];
f2 = showgs["move 1", {ImagePadding → {{30, 20}, {0, 0}}}];
noisycirc3 = InsertCircuitNoise[circ3, devGS];
noisycirc4 = InsertCircuitNoise[circ4, devGS];
f3 = showgs["move 2", {ImagePadding → {{30, 20}, {20, 0}}}];
noisycirc5 = InsertCircuitNoise[circ5, devGS];
```

In[158]:=

```
Column[{f1, f2, f3}, Spacings → 0]
(*Export["rydberg_graph.pdf",%]*)
```

Out[158]=



Results

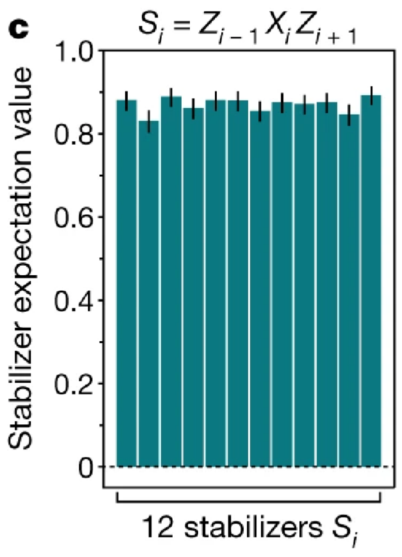
Modules related to displaying the results

In[159]:=

```
chartGraph1D[res_, expresults_] := With[
  {scount = res["scount"], nshots = res["outeven"] // Length, stabideal = res["sideal"]},
  Show[
   BarChart[Values@scount / nshots,
    ChartLabels → (ToString["S"_#, TraditionalForm] & /@ Range[0, 11]),
    Frame → True, FrameStyle → Directive[Black, Thick],
    AspectRatio → 1.2,
    ChartStyle → ColorData["DeepSeaColors"][0.7],
    PlotRange → {Automatic, {-0.05, 1}}]
   ,
   BarChart[expresults, ChartStyle → Directive[Opacity[0], EdgeForm[{Dashed, Thick}]]]
   ,
   ListPlot[stabideal, Joined → True, PlotMarkers → {"■", 15}, PlotStyle → 🟥]
   ,
   ImageSize → {Automatic, 400}, Background → White,
   LabelStyle → {16, FontFamily → "Serif"}, ImagePadding → {{30, 5}, {30, 10}}
  ]
 ]

showResultGraph1D[res_, expresults_] := With[
  {dev = RydbergHub[Sequence @@ res[["opt"]]], nshots = res["outeven"] // Length}
  ,
  <|"nshots" → ToString@nshots,
   "chart" → chartGraph1D[res, expresults],
   "benchmark" → Table[Between[res[["scount"]][j - 1] / nshots,
      Sort[expresults[[j]][[1]] + {1, -1} * expresults[[j]][[2]]]], {j, Length@expresults}],
   "erravg" → Mean@Abs[N[Values@res[["scount"]] / nshots] - expresults],
   "errmax" → Max@Abs[N[Values@res[["scount"]] / nshots] - First /@ expresults],
   "stabavg" → N@Mean@Values@res[["scount"]] / nshots,
   "nospamavg" → Mean@res[["sideal"]] |>
 ]
```

Quoted from the paper to compare

c


In[161]:=

```
cs1dmean = {0.8814814814814814, 0.8314814814814814, 0.8888888888888887,
   0.8629629629629629, 0.8814814814814814, 0.8796296296296295,
   0.8555555555555554, 0.8759259259259258, 0.8722222222222221,
   0.8777777777777775, 0.8462962962962962, 0.8925925925925925};
cs1dminus = {0.8537037037037036, 0.8018518518518518, 0.8629629629629629,
   0.8333333333333333, 0.8537037037037036, 0.8537037037037036,
   0.8277777777777777, 0.8481481481481481, 0.8444444444444444,
   0.8481481481481481, 0.8185185185185184, 0.8685185185185184};
cs1dplus = {0.9018518518518518, 0.8574074074074074, 0.912962962962963,
   0.8870370370370371, 0.9037037037037035, 0.9037037037037035,
   0.8814814814814814, 0.898148148148148, 0.8962962962962961,
   0.898148148148148, 0.8722222222222221, 0.9148148148148149};
```

In[164]:=

```
cs1d = Around[#[[1]], #[[2 ;;]] - #[[1]]] & /@ Transpose[{cs1dmean, cs1dminus, cs1dplus}]
```

Out[164]=

$\{0.881^{+0.020}_{-0.028}, 0.831^{+0.026}_{-0.030}, 0.889^{+0.024}_{-0.026}, 0.863^{+0.024}_{-0.030}, 0.881^{+0.022}_{-0.028}, 0.880^{+0.024}_{-0.026},$
$0.856^{+0.026}_{-0.028}, 0.876^{+0.022}_{-0.028}, 0.872^{+0.024}_{-0.028}, 0.878^{+0.020}_{-0.030}, 0.846^{+0.026}_{-0.028}, 0.893^{+0.022}_{-0.024}\}$

Results from simulation

In[165]:=

```
grahstate1d << "supplement/GraphStatesonRydbergHub/graphstate1d.mx";
```

In[166]:=

```
graphstate1d // Length
```

Out[166]=

```
47
```

In[167]:=

```
allres = showResultGraph1D[#, cs1d] & /@ graphstate1d;
```

In[168]:=

```
(* best
 results: 11/12 stabilizer measurements agree with the experimental results*)
Count[#, True] & /@ allres[[All, "benchmark"]]
best = Flatten@Position[%, x_ /; x ≥ 11]
```

Out[168]=

```
{7, 8, 8, 7, 8, 6, 3, 8, 9, 11, 9, 6, 7, 7, 6, 8, 7, 11, 6, 4, 8, 4, 5,
 8, 9, 9, 8, 8, 6, 6, 7, 8, 8, 9, 10, 8, 9, 8, 8, 10, 8, 7, 10, 8, 9, 9, 10}
```

Out[169]=

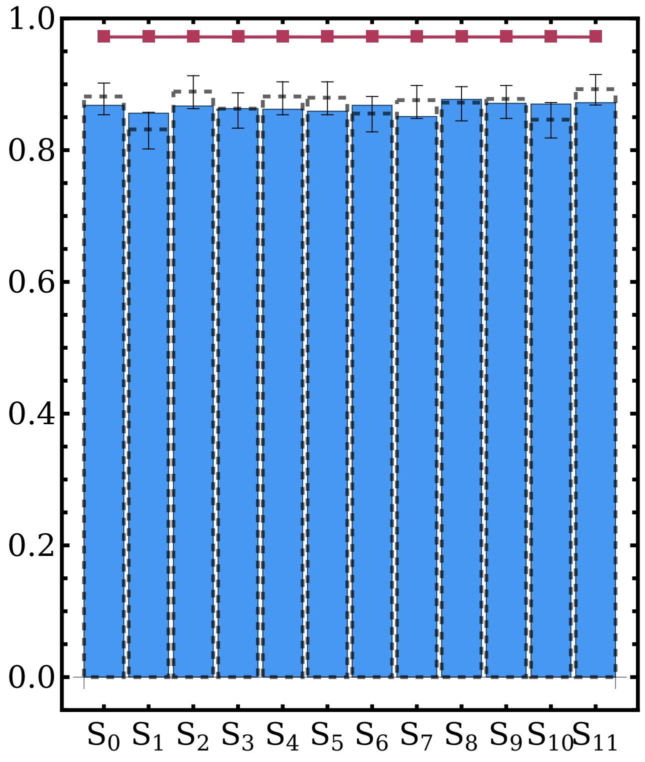
```
{10, 18}
```

In[170]:=

```
(* the result shown in the paper *)
showResultGraph1D[graphstate1d[[18]], cs1d]
```

Out[170]=

```
⟨|nshots → 2000, chart →
```



```
,
 benchmark → {True, True, True, True, True, True, True, False, True, True, True, True},
 erravg → 0.016^{+0.007}_{-0.008}, errmax → 0.0249259, stabavg → 0.865333, nospamavg → 0.971812|⟩
```

In[171]:=

```
(*Export["stab_gs.pdf",showResultGraph1D[graphstate1d[[18]],cs1d][["chart"]]]*)
```

Steane code

Default device configuration

In[172]:=

```
Options[RydbergHub] = {
    QubitNum → 7,
    AtomLocations →
      <|6 → {0, 1}, 5 → {1, 1}, 2 → {2, 1}, 1 → {4, 1}, 4 → {2, 0}, 0 → {4, 0}, 3 → {5, 0}|>,
    T2 → 1.5 * 10′ ,
    VacLifeTime → 48 * 10′ ,
    RabiFreq → 1,
    ProbBFRot → <|10 → 0.001, 01 → 0.03|>,
    UnitLattice → 3,
    BlockadeRadius → 1,
    ProbLeakInit → 0.001,
    DurInit → 5 * 10` ,
    DurMove → 100,
    HeatFactor → 10,
    FidMeas → 0.975,
    DurMeas → 10,
    ProbLossMeas → 0.0001,
    ProbLeakCZ → <|01 → 0.01, 11 → 0.0001|>
  };
```

In[173]:=

Plots generation

In[174]:=

```
devst = RydbergHub[];
```

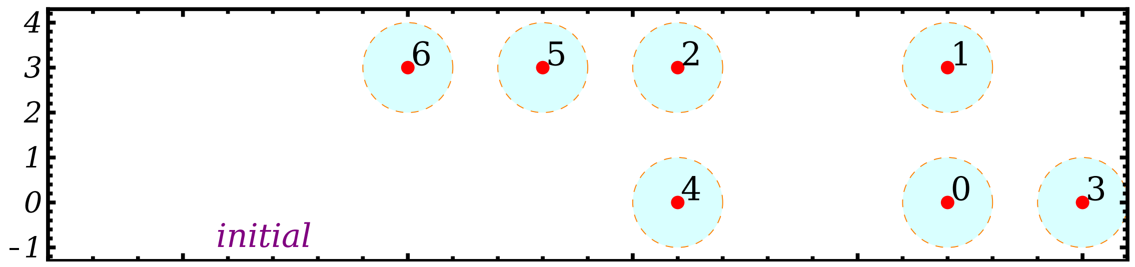
In[175]:=

```
ClearAll[showst]
showst[title_ : "", opt_ : {}] := PlotAtoms[devst, Sequence @@ opt, ImageSize → 600,
    ShowBlockade → Range[0, 6], LabelStyle → Directive[Italic, 15, Black],
    BaseStyle → {17, FontFamily → "Serif"}, PlotRange → {{-8, 16}, {-1.3, 4.3}},
    Epilog → Inset[Style[title, {Purple, Italic}], Scaled[{0.2, 0.1}]],
    Frame → True, FrameStyle → Directive[Black, Thick], Axes → False];
```

In[177]:=

```
move0 = showst["initial", {ImagePadding → {{20, 18}, {0, 18}}}]
```

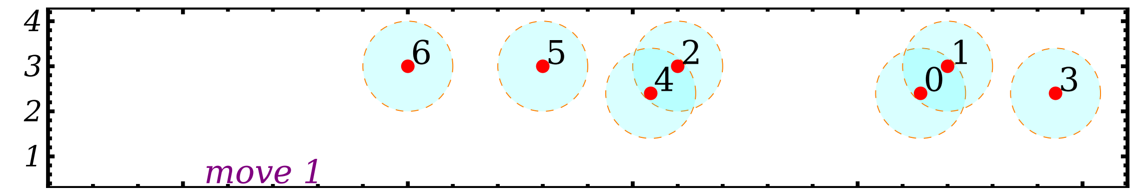
Out[177]=



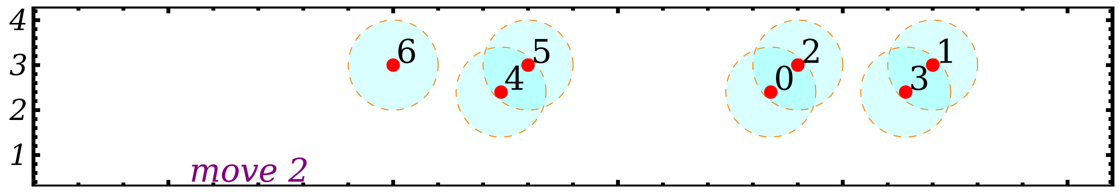
In[178]:=

```
devst = RydbergHub[];
circ0 = {Init# & /@ Range[0, 6], Ry#[π / 2] & /@ Range[0, 6]};
circ1 = {{ShiftLoc‹и˛и›[{-0.2, 0.8}]}, {C›[Z‹], C˛[Zι]}};
circ2 = {{ShiftLoc›[{1, 0}], ShiftLoc‹и˛и›[{-1, 0}]}, {C‹[Z›], C›[Z˛], Cι[Z›]}};
circ3 = {{ShiftLoc‹и˛и›[{-1, 0}]}, {C‹[Z‹], C›[Z›]}};
circ4 = {{ShiftLoc‹и˛и›[{-2, 0}]}, {C˛[Z‹], C›[Z›]}, Ry#[π / 2] & /@ {0, 3, 4}};
circ5 = {{ShiftLoc‹и˛и›[{-2, 0}]}, {C˛[Z‹], C›[Z›]}, Ry#[π / 2] & /@ {1, 2, 5, 6}};
InsertCircuitNoise[circ1, devst];
move1 =
 showst["move 1", {ImagePadding → {{20, 18}, {0, 0}}, PlotRange → {{-8, 16}, {0.3, 4.3}}}]
InsertCircuitNoise[circ2, devst];
move2 =
 showst["move 2", {ImagePadding → {{20, 18}, {0, 0}}, PlotRange → {{-8, 16}, {0.3, 4.3}}}]
InsertCircuitNoise[circ3, devst];
move3 =
 showst["move 3", {ImagePadding → {{20, 18}, {0, 0}}, PlotRange → {{-8, 16}, {0.3, 4.3}}}]
InsertCircuitNoise[circ4, devst];
move4 =
 showst["move 4", {ImagePadding → {{20, 18}, {18, 0}}, PlotRange → {{-8, 16}, {0.3, 4.3}}}]
```

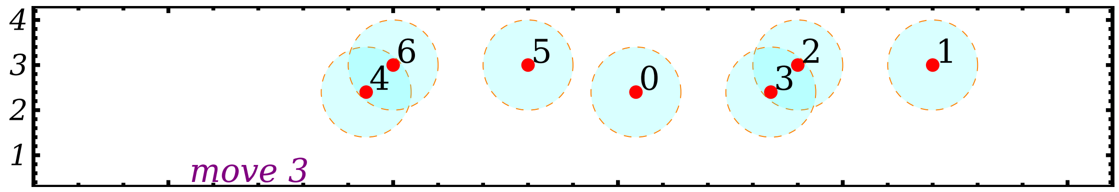
Out[186]=



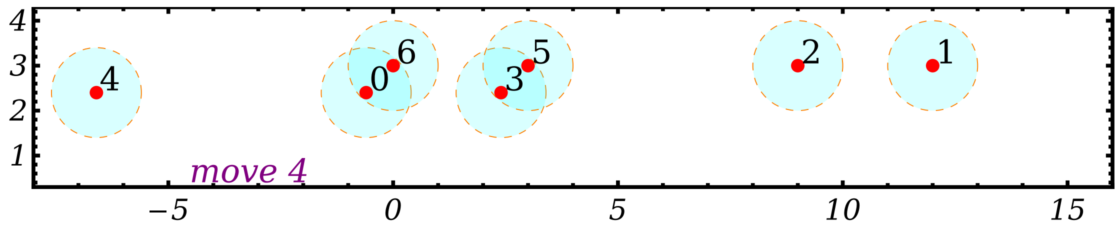
Out[188]=



Out[190]=



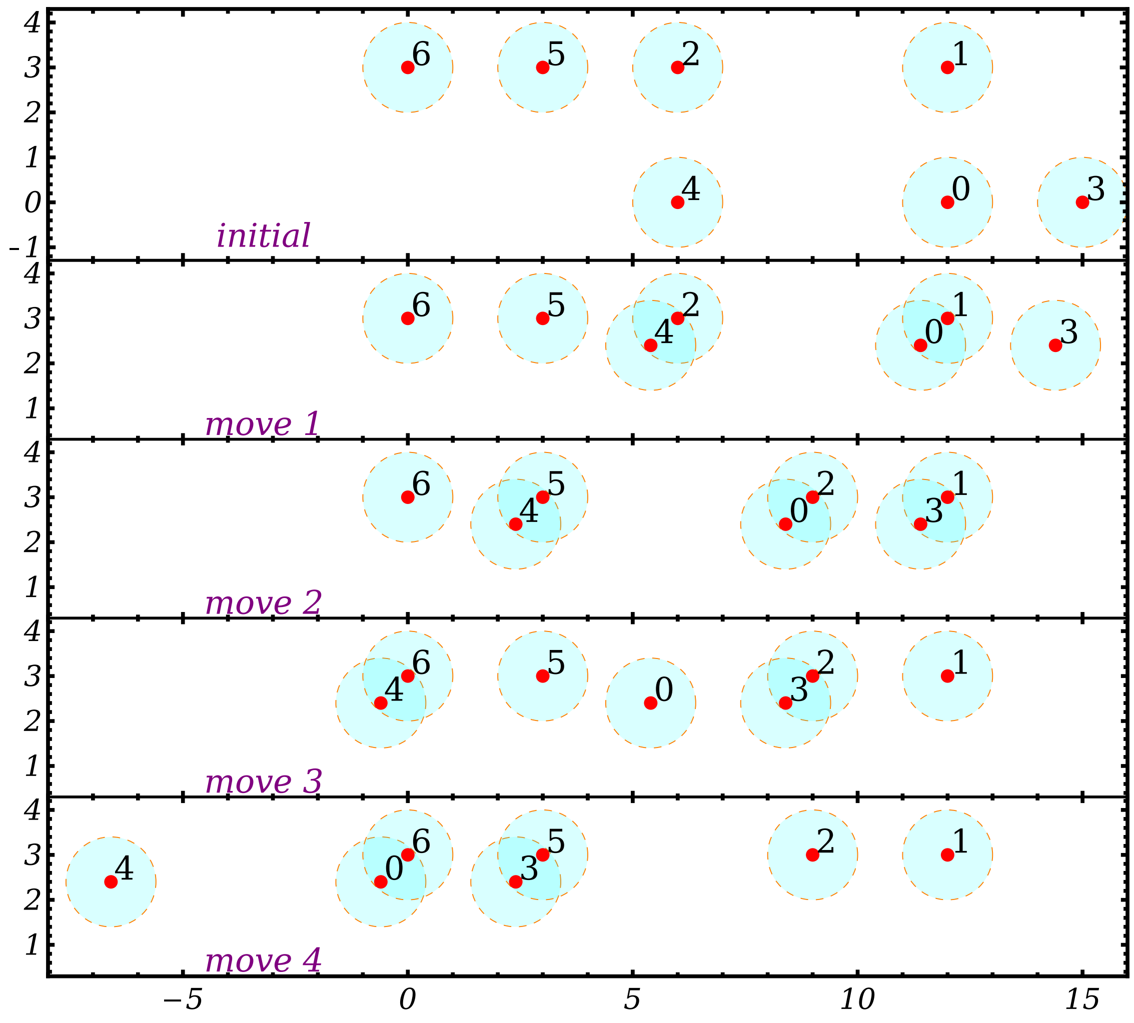
Out[192]=



In[193]:=

```
Column[{move0, move1, move2, move3, move4}, Spacings → -0.1]
(*Export["rydberg_steane.pdf",%]*)
```

Out[193]=



Entire simulation circuit

In[194]:=

```
stabx = {X X X X , X X X X , X X X X };
stabz = {Z Z Z Z , Z Z Z Z , Z Z Z Z };
xlogic = X X X ;
zlogic = Z Z Z ;
(* returns indices of the involved stabilizers *)
stabindex[stab_] := Level[stab, 1] /. Subscript[_, j_] :→ j
```

In[199]:=

```
DestroyAllQuregs[]
{ρ, ρinit, ρwork} = CreateDensityQuregs[7, 3];
```

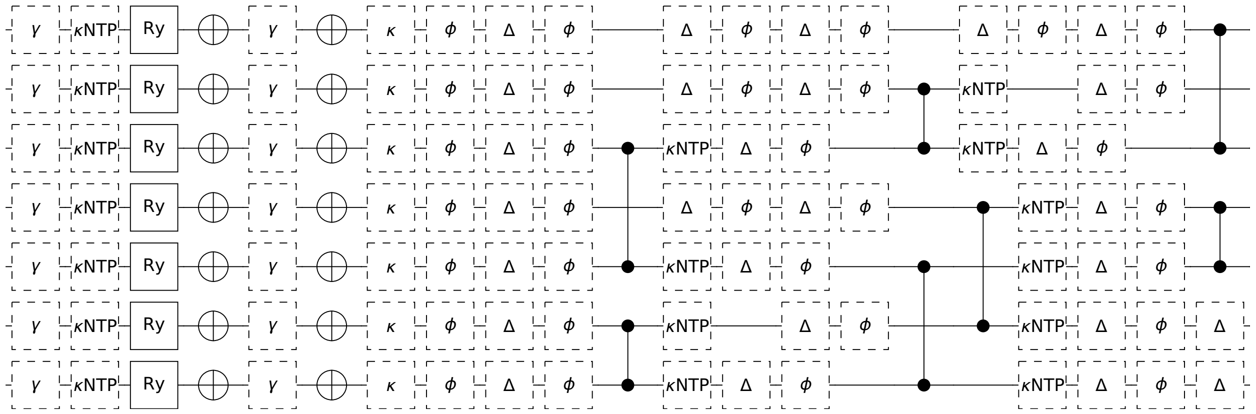
In[201]:=

```
noisycirc = ExtractCircuit@InsertCircuitNoise[
    Join[circ0, circ1, circ2, circ3, circ4], RydbergHub[], ReplaceAliases → True];
(*simplify, and remove zero-parameterised operations *)
simpncirc = noisycirc;
(simpncirc = DeleteCases[simpncirc, #]) & /@ {Depol_[0.], Deph_[0.], Damp_[0.]};
```

In[204]:=

```
DrawCircuit@simpncirc
```

Out[204]=

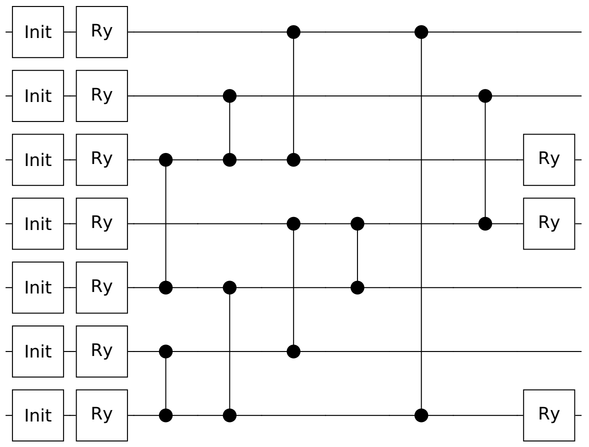


Logical |+>

In[205]:=

```
DrawCircuit@
 DeleteCases[Flatten@Join[circ0, circ1, circ2, circ3, circ4], ShiftLoc_[_]]
```

Out[205]=



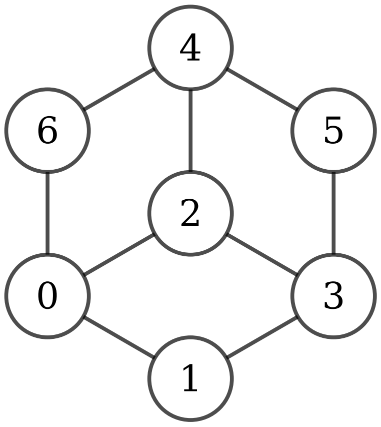
In[206]:=

```
ApplyCircuit[SetQuregMatrix[ρ, IdentityMatrix[2^[illegible]]/2^[illegible]], simpncirc]
```

Out[206]=

```
{}
```

In[207]:=
```
Graph[{2 ⟷ 4, 0 ⟷ 1, 4 ⟷ 5, 2 ⟷ 0, 1 ⟷ 3, 4 ⟷ 6, 2 ⟷ 3, 0 ⟷ 6, 3 ⟷ 5},
 VertexSize → 0.5, VertexStyle → Directive[White, EdgeForm[Thick]],
 BaseStyle → {19, FontFamily → "Serif"},
 ImageSize → 200, EdgeStyle → Directive[Black, Thick],
 VertexLabels → Placed[Automatic, Center], GraphLayout → "TutteEmbedding"]
(*Export["graphsteane.pdf",%]*)
```
Out[207]=



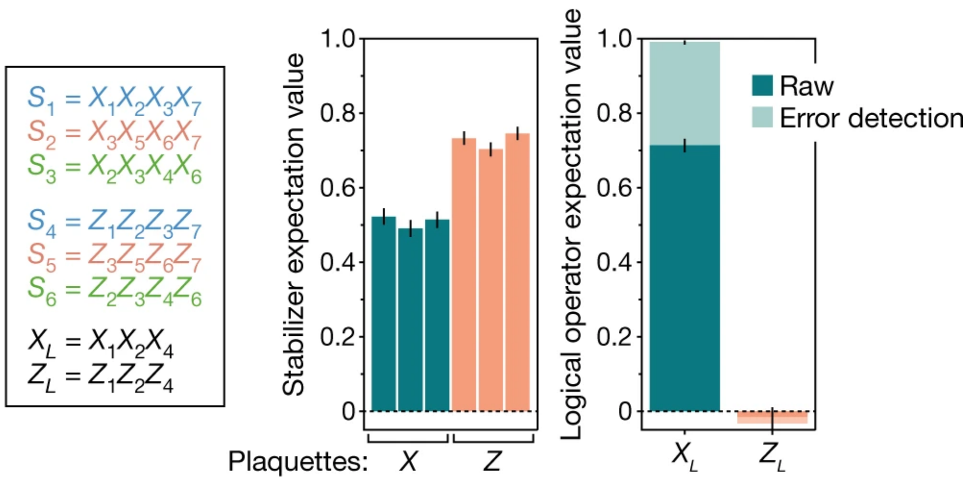
Results

Modules related to displaying the results

In[208]:=
```
chartSteane[res_, expstabs_, explogic_] := With[
  {
   sxcount = res["sxcount"],
   szcount = res["szcount"],
   nshots = Length@res["outx"],
   sxideal = res["sxideal"],
   szideal = res["szideal"],
   logiccount = res["logiccount"],
   logicideal = res["logicideal"],
   cols = {🟩, 🟧},
   size = 400
  },
  Row@{
    Show[
       BarChart[
      Flatten@{Values@sxcount / nshots, Values@szcount / nshots},
      ChartLabels → (ToString["S"#, TraditionalForm] & /@ Range[0, 5]),
      Frame → True, FrameStyle → Directive[Black, Thick], AspectRatio → 2.5,
      ChartStyle → Flatten@{ConstantArray[cols⟦1⟧, 3], ConstantArray[cols⟦2⟧, 3]},
      PlotRange → {Automatic, {-0.05, 1}},
      Background → White, ImageSize → {Automatic, size},
      ImagePadding → {{30, 0}, {30, 10}}, BaseStyle → {16, FontFamily → "Serif"}],
     ListPlot[{Join[sxideal, szideal], Join[sxideal, szideal]},
      Joined → True, PlotMarkers → {"■", 15}, PlotStyle → 🟥],
     BarChart[Flatten@expstabs,
      ChartStyle → Directive[Opacity[0], EdgeForm[{Dashed, Thick}]]]
    ],
    Show[
     BarChart[Values@logiccount / nshots, ChartLabels → {"X_L", "Z_L"},
      Frame → True, FrameStyle → Directive[Black, Thick],
      AspectRatio → 5, ChartStyle → cols, PlotRange → {{0., 3}, {-0.05, 1}},
      FrameTicks → {Automatic, Automatic}, BaseStyle → {16, FontFamily → "Serif"}],
     BarChart[explogic, ChartStyle → Directive[Opacity[0], EdgeForm[{Dashed, Thick}]]],
     Background → White, ImageSize → {Automatic, size}, ImagePadding → {{0, 0}, {30, 10}}
    ]
   }
 ]


(*
sumarise the result
*)
showResultSteane[res_, expstabs_, explogic_] :=
 With[{dev = RydbergHub[Sequence @@ res⟦"opt"⟧], nshots = res["outx"] // Length},
  <|
   "nshots" → nshots,
   "avgstab" → <|"x" → N@Mean@Values@res["sxcount"] / nshots,
     "z" → N@Mean@Values@res["szcount"] / nshots,
     "xl" → N[res["logiccount"]["x"] / nshots], "zl" → N[res["logiccount"]["z"] / nshots]|>,
   "errmaxstab" → <|"x" → N@Max[Abs[res["sxcount"] / nshots - First /@ First@expstabs]],
     "z" → N@Max[Abs[res["szcount"] / nshots - First /@ Last@expstabs]]|>,
   "erravgstab" → <|"x" → N@Mean[Abs[Values@res["sxcount"] / nshots - First@expstabs]],
     "z" → N@Mean[Abs[Values@res["szcount"] / nshots - Last@expstabs]]|>,
   "benchmarkstab" → <|
     "x" → Table[Between[res⟦"sxcount"⟧[j - 1] / nshots, Sort[First[expstabs]⟦j⟧⟦1⟧ +
          {1, -1} * First[expstabs]⟦j⟧⟦2⟧]], {j, Length@First@expstabs}],
     "z" → Table[Between[res⟦"szcount"⟧[j - 1] / nshots, Sort[Last[expstabs]⟦j⟧⟦1⟧ +
          {1, -1} * Last[expstabs]⟦j⟧⟦2⟧]], {j, Length@Last@expstabs}]|>,
   "benchmarklogic" → <|"x" → Between[
       res⟦"logiccount"⟧["x"] / nshots, Sort[explogic⟦1⟧⟦1⟧ + {1, -1} * explogic⟦1⟧⟦2⟧]],
     "z" → Between[res⟦"logiccount"⟧["z"] / nshots,
       Sort[explogic[[2]][[1]] + {1, -1} * explogic[[2]][[2]]] |> ,
   "errlogic" → <|"x" → Abs[res["logiccount"]["x"] / nshots - First@explogic],
     "z" → Abs[res["logiccount"]["z"] / nshots - Last@explogic] |> ,
   "idealavg" → <|"x" → Mean@res["sxideal"], "z" → Mean@res["szideal"],
     "xl" → res["logicideal"]["x"], "zl" → res["logicideal"]["z"] |> ,
   "chart" → chartSteane[res, expstabs, explogic]
   |>
 ]
```

Quoted from the paper to compare



In[210]:=

```
steanemean = {0.5246212121212122, 0.4924242424242424, 0.5170454545454546,
    0.7329545454545454, 0.7045454545454546, 0.7462121212121211};
steaneminus = {0.5, 0.46780303030303033, 0.4905303030303031,
    0.7121212121212122, 0.6799242424242425, 0.7234848484848485};
steaneplus = {0.5454545454545455, 0.5151515151515151, 0.5378787878787878,
    0.7518939393939393, 0.7234848484848485, 0.7651515151515151};
```

In[213]:=

```
lsteanemean = {0.7134502923976608, -0.015594541910331362};
lsteaneminus = {0.6939571150097467, -0.050682261208577};
lsteaneplus = {0.7290448343079922, 0.009746588693957212};
```

In[216]:=

```
steane = Partition[
  Around[#⟦1⟧, #⟦2 ;;⟧ - #⟦1⟧] & /@ Transpose[{steanemean, steaneminus, steaneplus}], 3]
lsteane =
 Around[#⟦1⟧, #⟦2 ;;⟧ - #⟦1⟧] & /@ Transpose[{lsteanemean, lsteaneminus, lsteaneplus}]
```

Out[216]=

$\{\{0.525^{+0.021}_{-0.025}, 0.492^{+0.023}_{-0.025}, 0.517^{+0.021}_{-0.027}\}, \{0.733^{+0.019}_{-0.021}, 0.705^{+0.019}_{-0.025}, 0.746^{+0.019}_{-0.023}\}\}$

Out[217]=

$\{0.713^{+0.016}_{-0.019}, -0.016^{+0.025}_{-0.035}\}$

In[218]:=

```
(*stabsteane={{0.52,0.49,0.51},{0.732,0.7,0.75}};*)
logsteane = {0.71, -0.02};
cols = {🟩, 🟧};
```

Results from simulation

In[220]:=

```
steane7 << "supplement/GraphStatesonRydbergHub/steane7.mx";
```

In[221]:=

```
steane7 // Length
```

Out[221]=

```
216
```

In[222]:=

```
truth = Table[
   out = Values@
     showResultSteane[res, steane, lsteane]⟦{"benchmarkstab", "benchmarklogic"}⟧;
   Flatten@{Values@out⟦1⟧, Values@out⟦2⟧}
   , {res, steane7}];

truecount = Count[#, True] & /@ truth;
Max@truecount
```

Out[224]=

```
7
```

Take the best result

In[225]:=

```
best = Flatten@Position[truecount, x_ /; x ≥ 7]
```

Out[225]=

```
{28, 70, 80, 84, 86, 96, 98, 102, 103, 105, 115, 128, 142, 144, 148,
 150, 156, 160, 163, 168, 170, 173, 177, 181, 189, 201, 205, 212, 216}
```

In[226]:=

```
bestres = showResultSteane[steane7[[#]], steane, lsteane] & /@ best;
```

In[227]:=

```
(* minimum by the distance to the average given in the experiment *)
Ordering[Total@Abs[# - {0.51, 0.73, 0.71, -0.02}] & /@
  Flatten[Values /@ Values /@ bestres[[All, {"avgstab"}]], 1], 3]
```

Out[227]=

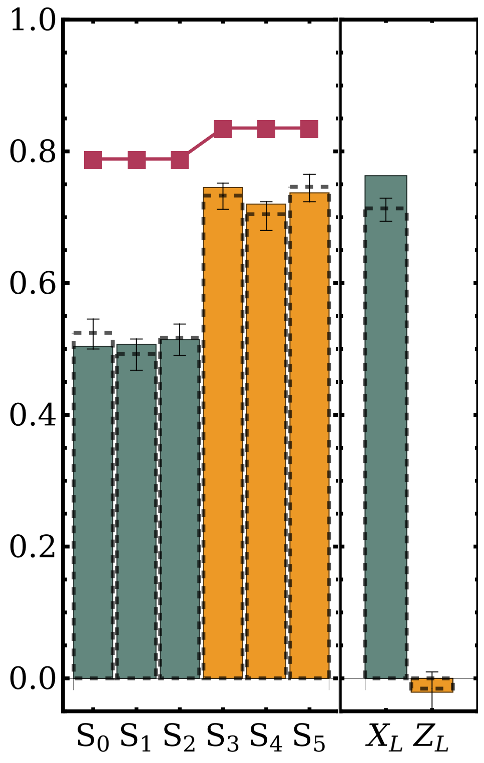
```
{2, 18, 21}
```

In[228]:=

```
(* the result shown in the paper *)
bestres[[18]]
```

Out[228]=

```
<|nshots → 2000, avgstab → <|x → 0.508333, z → 0.734, xl → 0.763, zl → -0.021|>,
 errmaxstab → <|x → 0.0215758, z → 0.0404545|>,
 erravgstab → <|x → 0.013^{+0.012}_{-0.015}, z → 0.012^{+0.011}_{-0.013}|>,
 benchmarkstab → <|x → {True, True, True}, z → {True, True, True}|>,
 benchmarklogic → <|x → False, z → True|>, errlogic → <|x → 0.050^{+0.016}_{-0.019}, z → 0.005^{+0.025}_{-0.035}|>,
 idealavg → <|x → 0.788493, z → 0.835445, xl → 0.835995, zl → -0.0000278615|>,
 chart →
```



```
|>
```

In[229]:=

```
(*Export["stab_steane.pdf",bestres[[18]][["chart"]]]*)
```