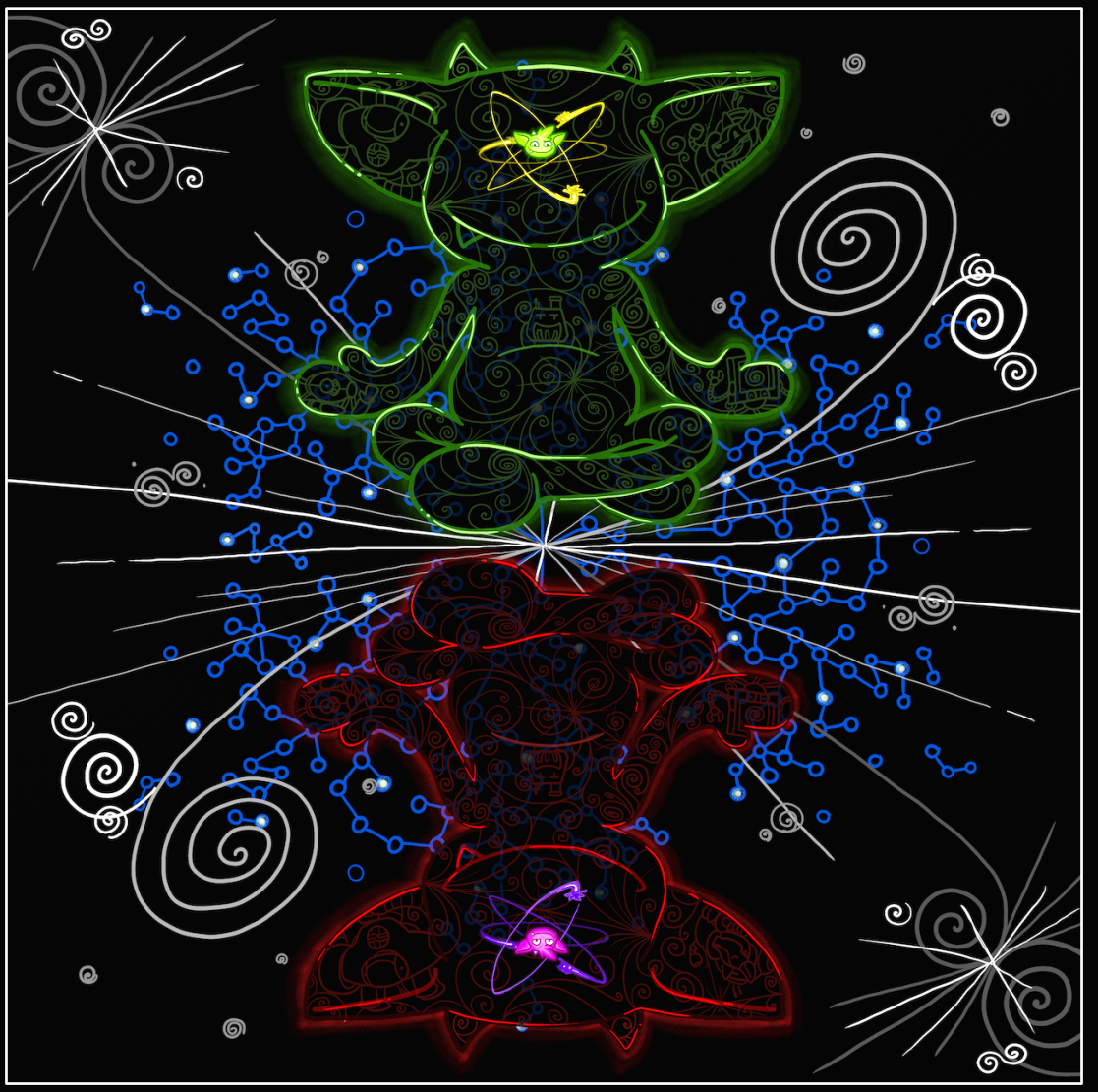
Quantum Processes in Graph Computing

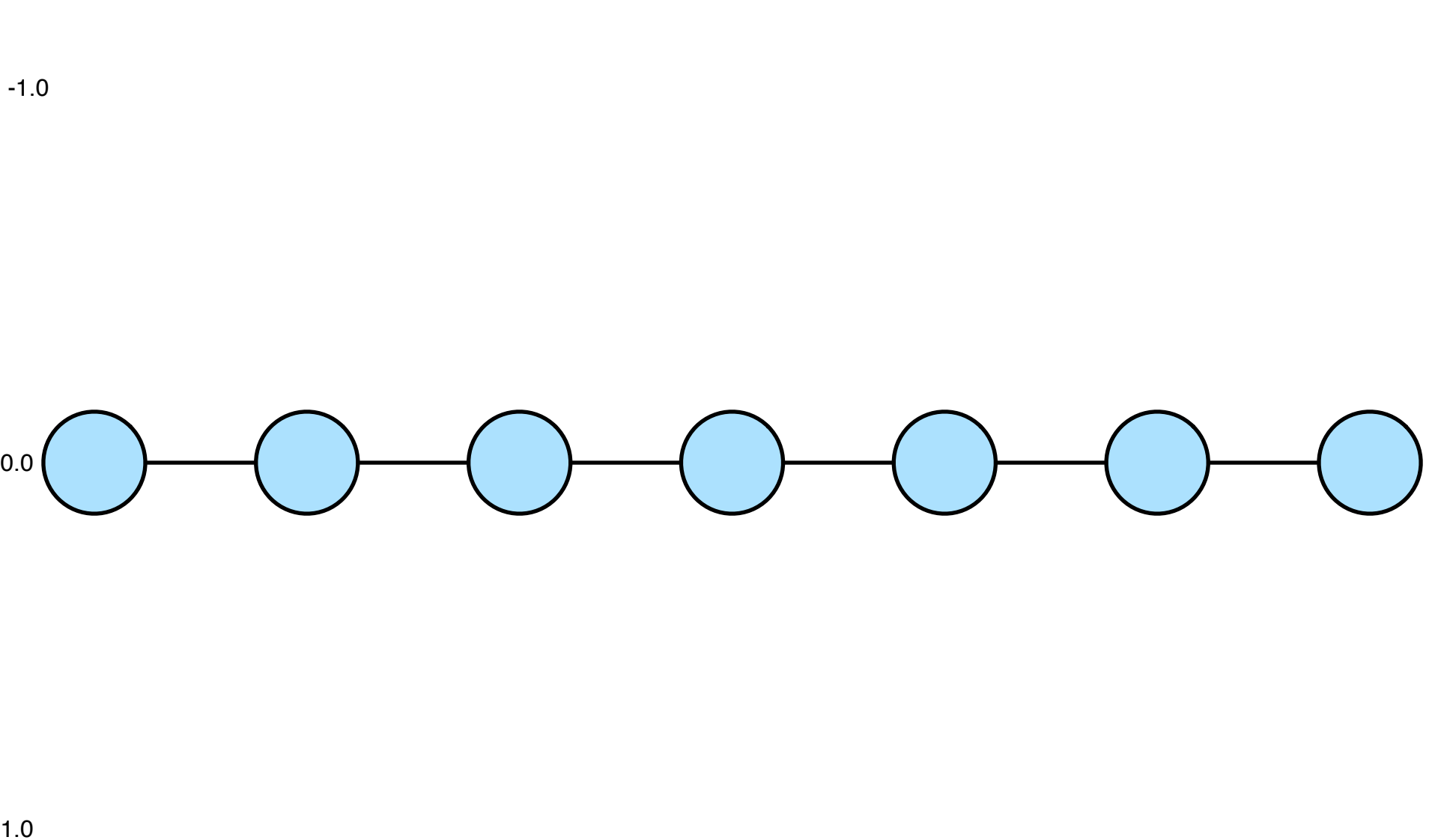


Dr. Marko A. Rodriguez
DataStax and Apache TinkerPop

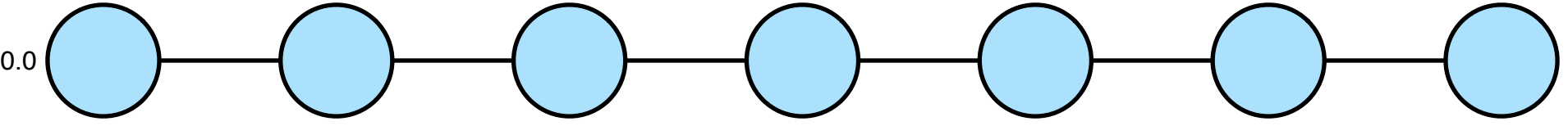
Rodriguez, M.A., Watkins, J.H., "Quantum Walks with Gremlin," GraphDay '16, pp. 1-16, Austin, Texas, January 2016.
http://arxiv.org/abs/1511.06278

Part 1

Wave Dynamics in Graphs

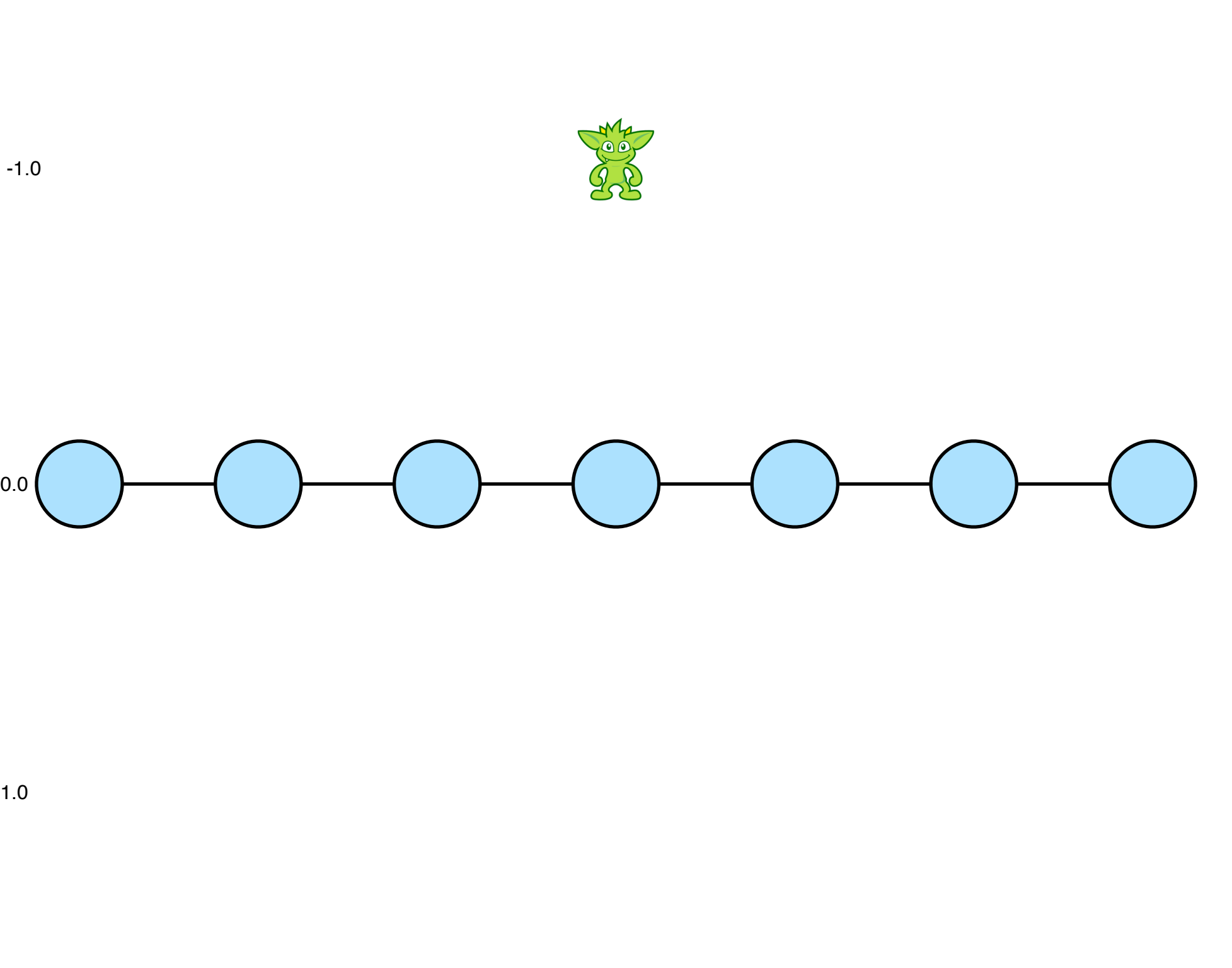

-1.0
0.0
1.0

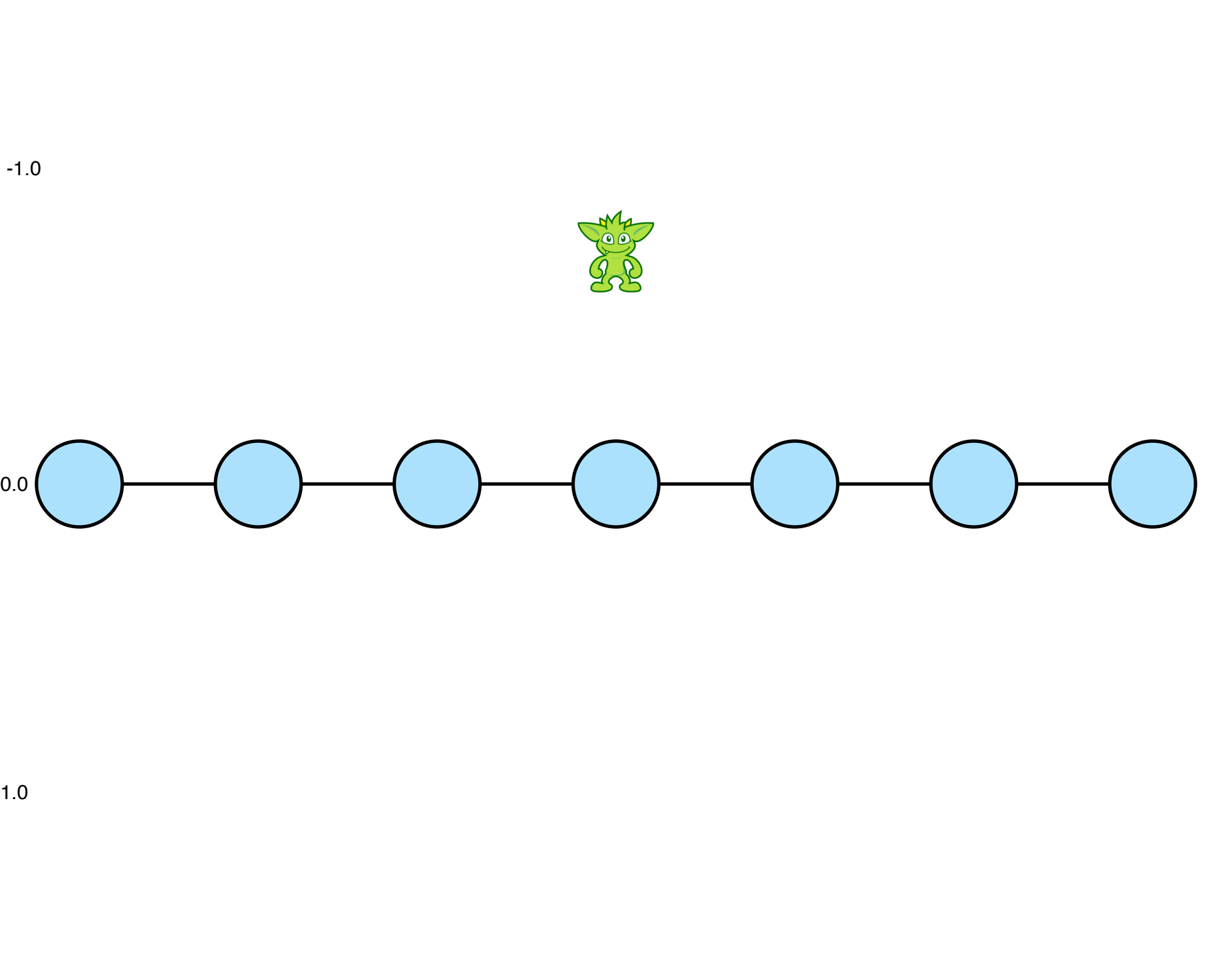
-1.0

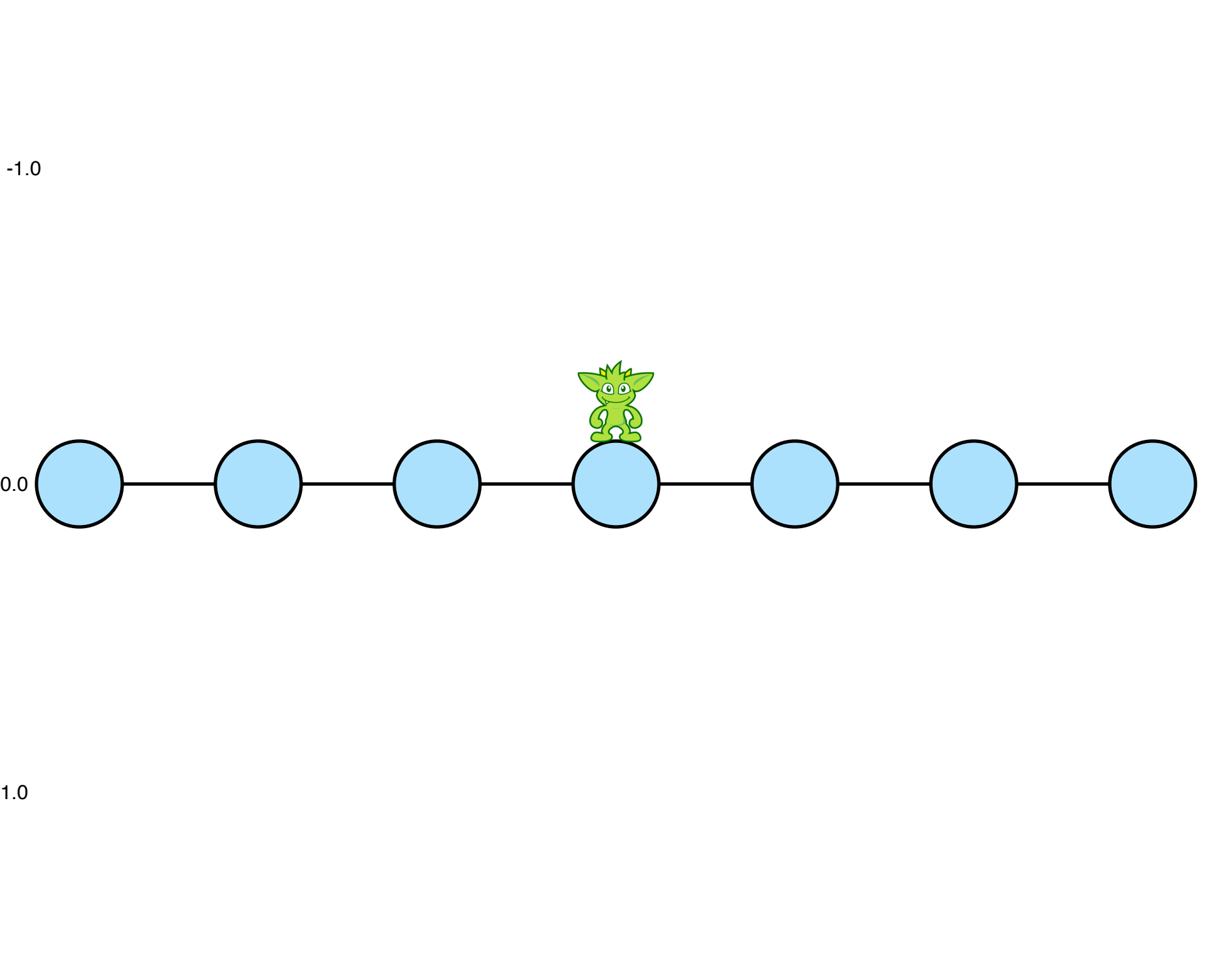


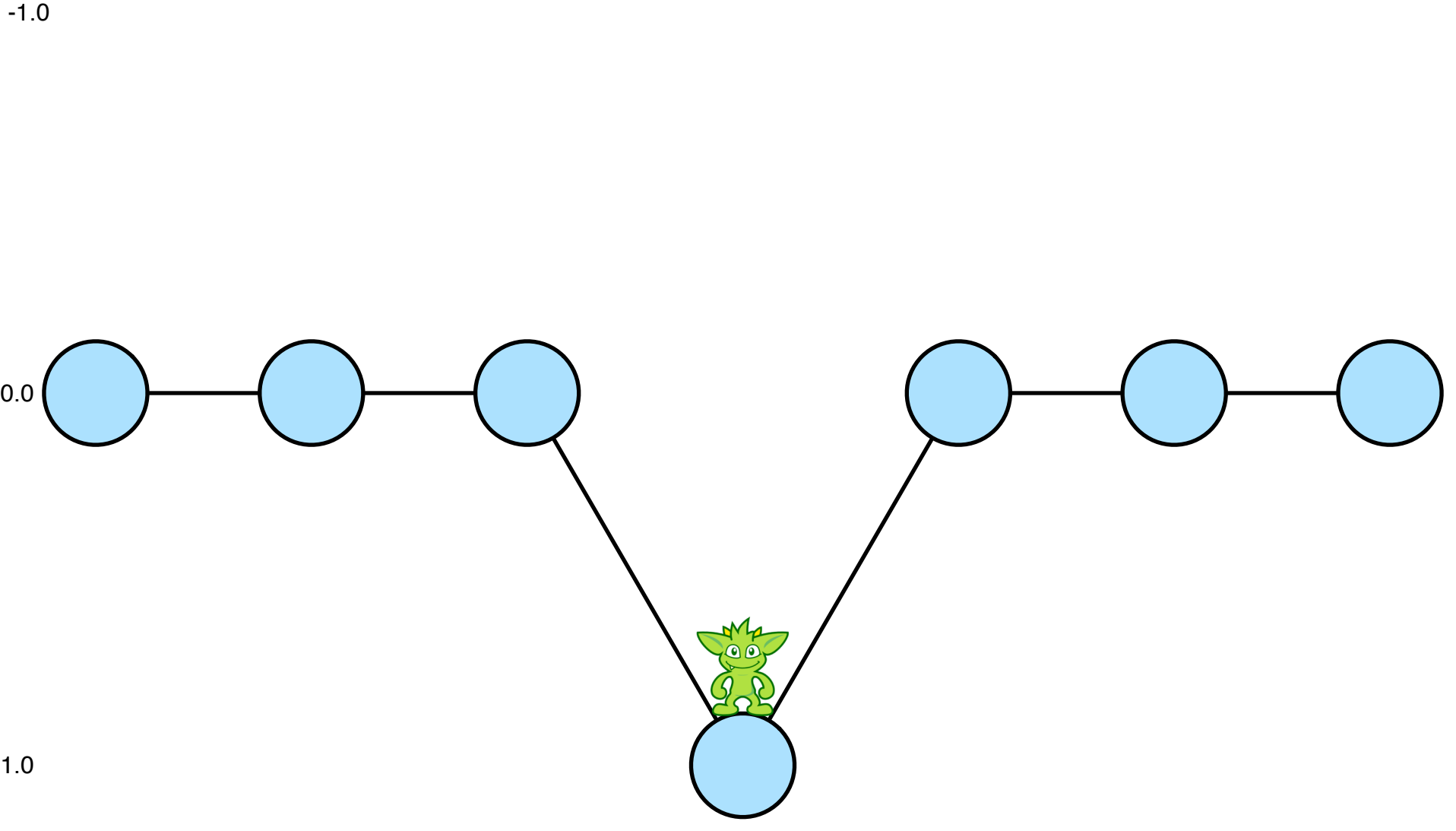
0.0

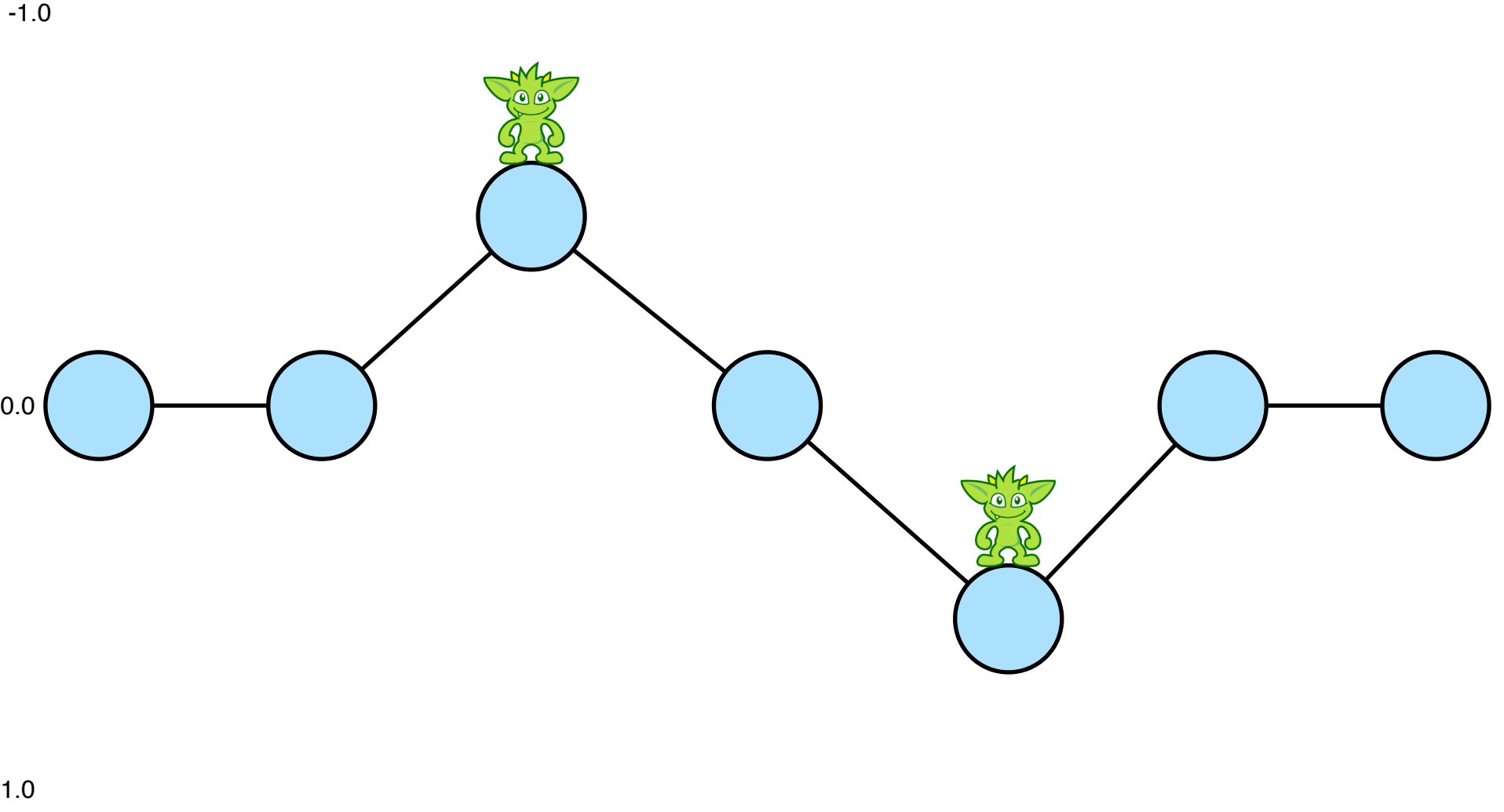
1.0

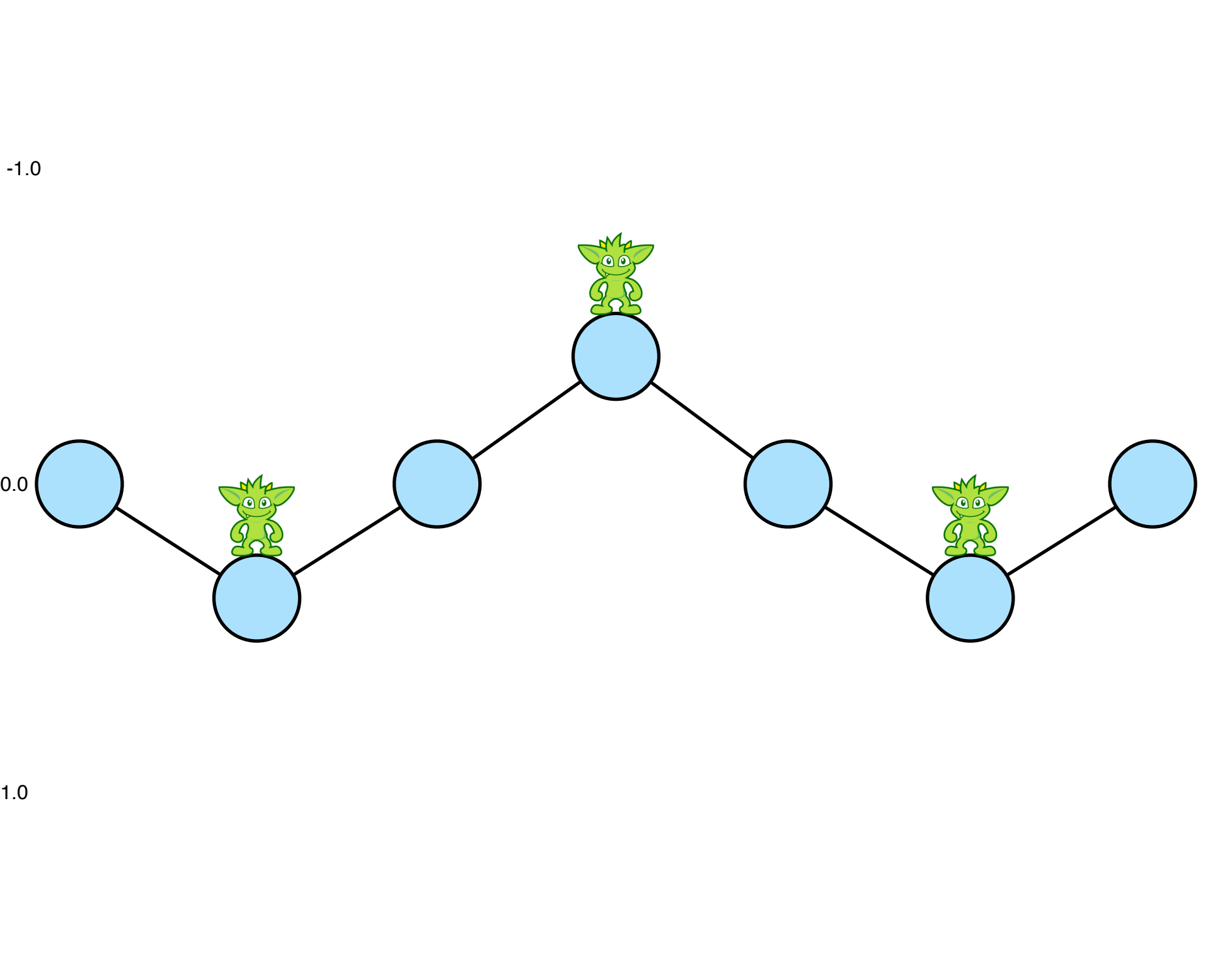

-1.0
0.0
1.0

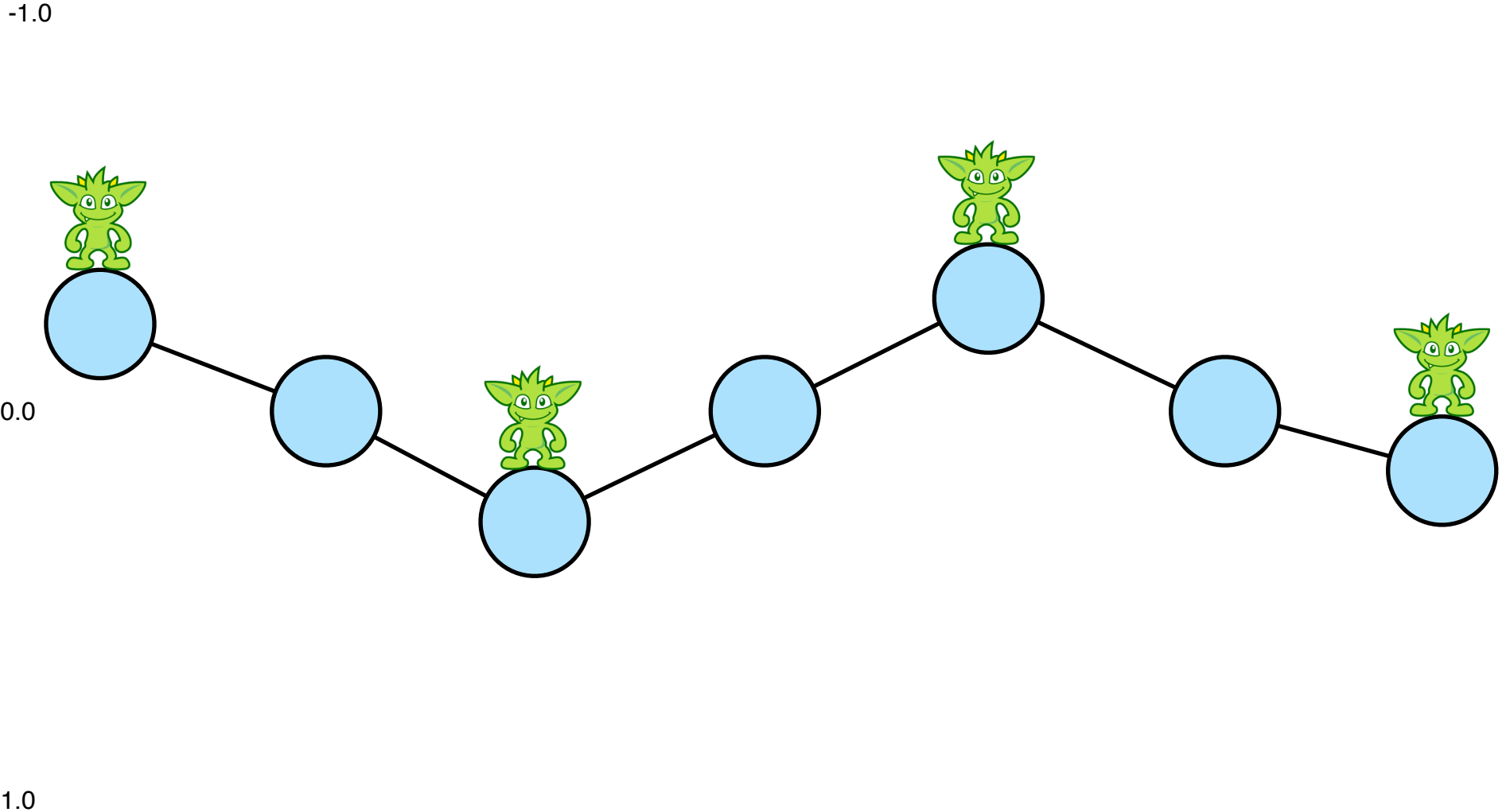

-1.0
0.0
1.0

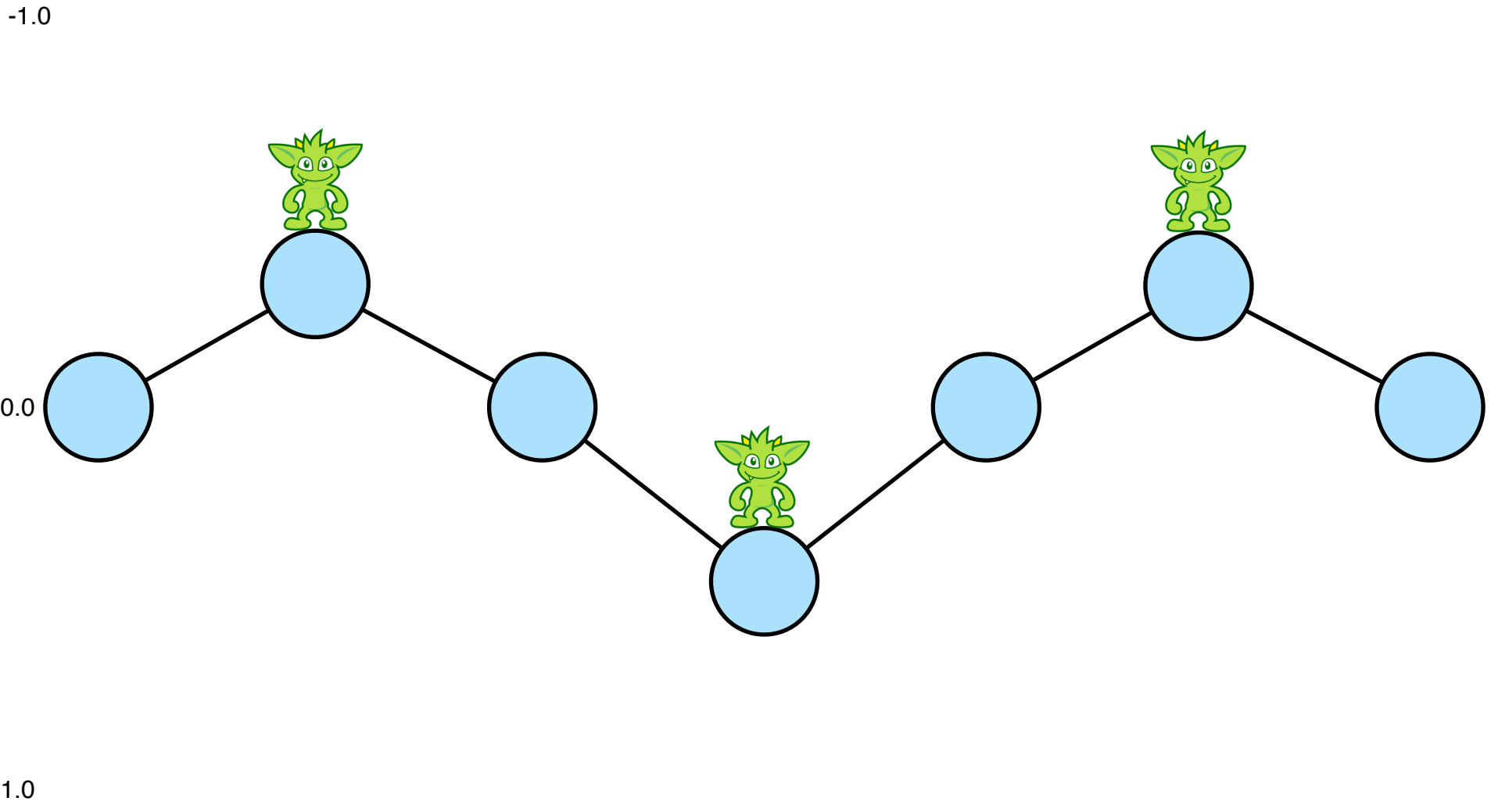

-1.0
0.0
1.0

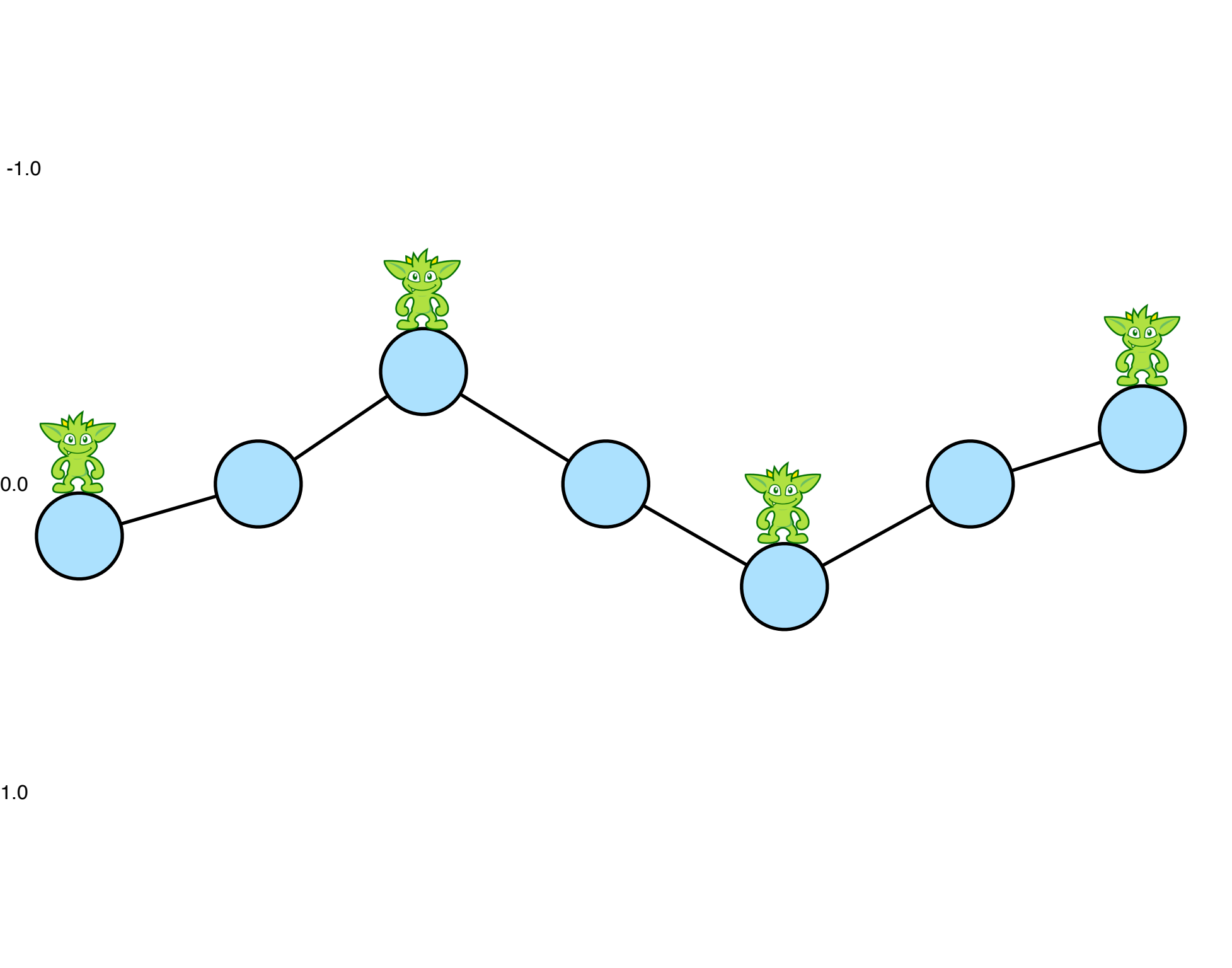

-1.0
0.0
1.0

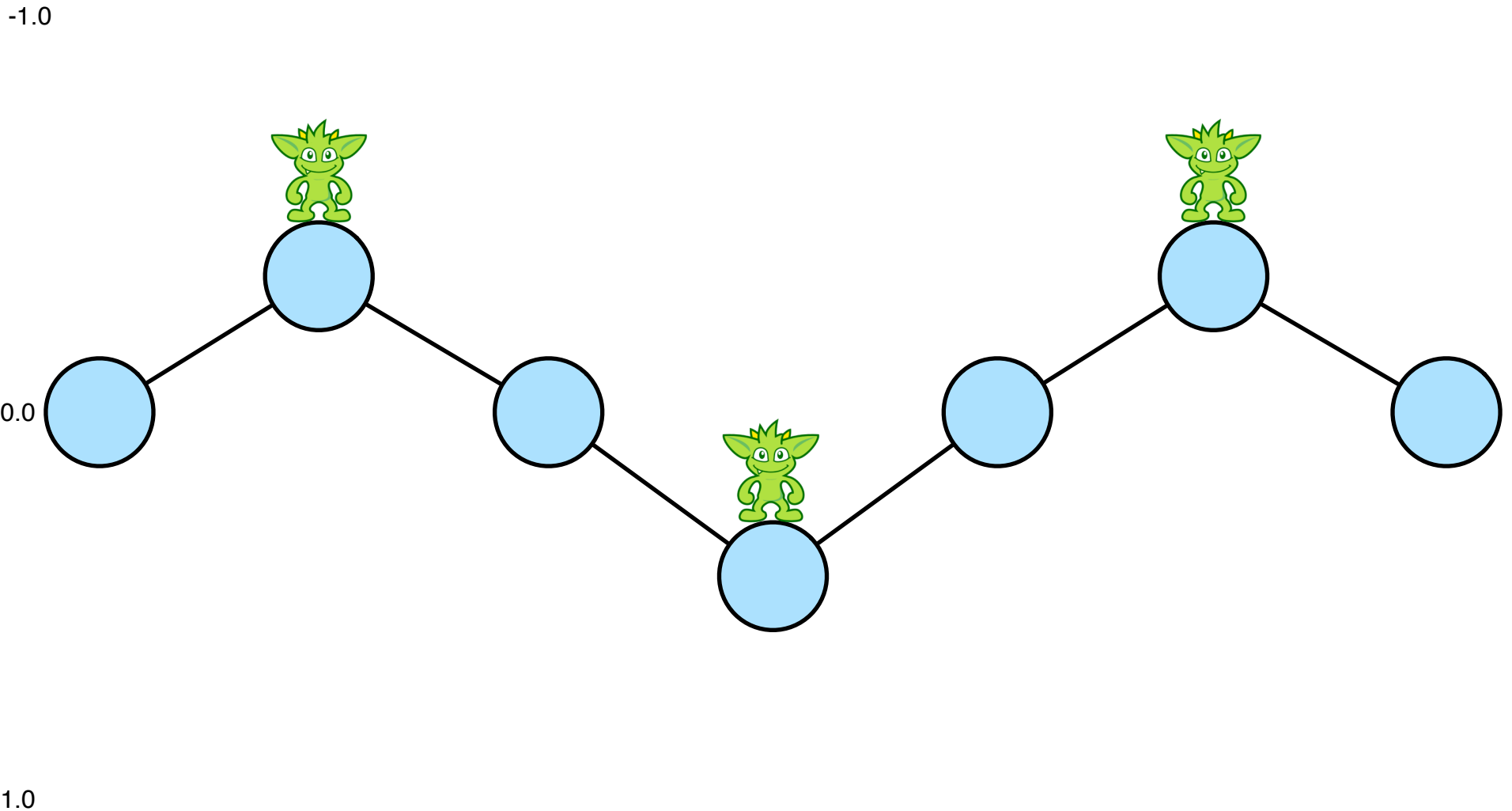

-1.0
0.0
1.0

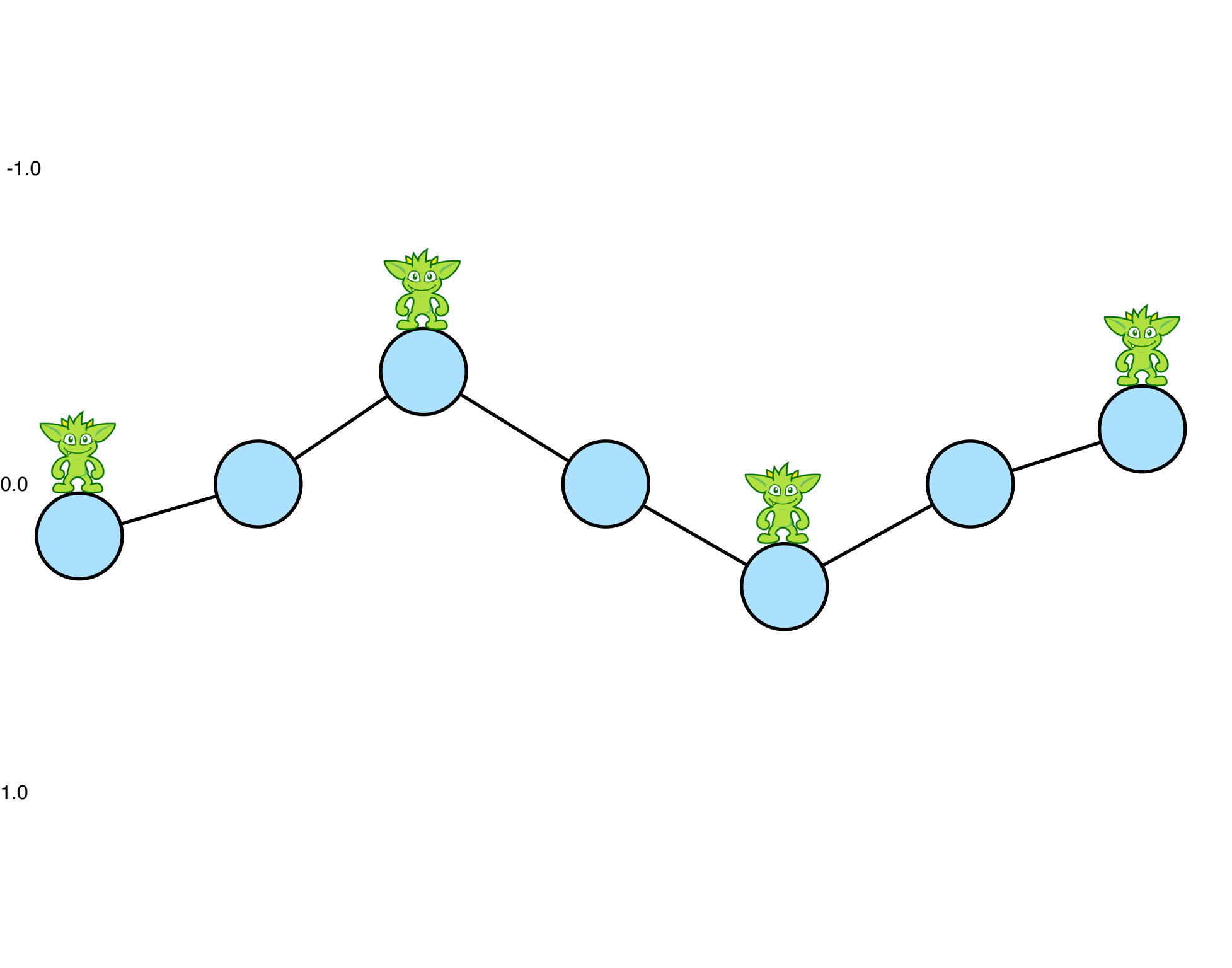

-1.0
0.0
1.0


-1.0
0.0
1.0


-1.0
0.0
1.0


-1.0
0.0
1.0


-1.0
0.0
1.0


-1.0
0.0
1.0

However, vertices don't "move."
The graph topology is the only space.
There is no "space" outside the graph.

"Then how do you model an oscillating vertex?"

By computing the amount of "energy"
at the vertex over time.

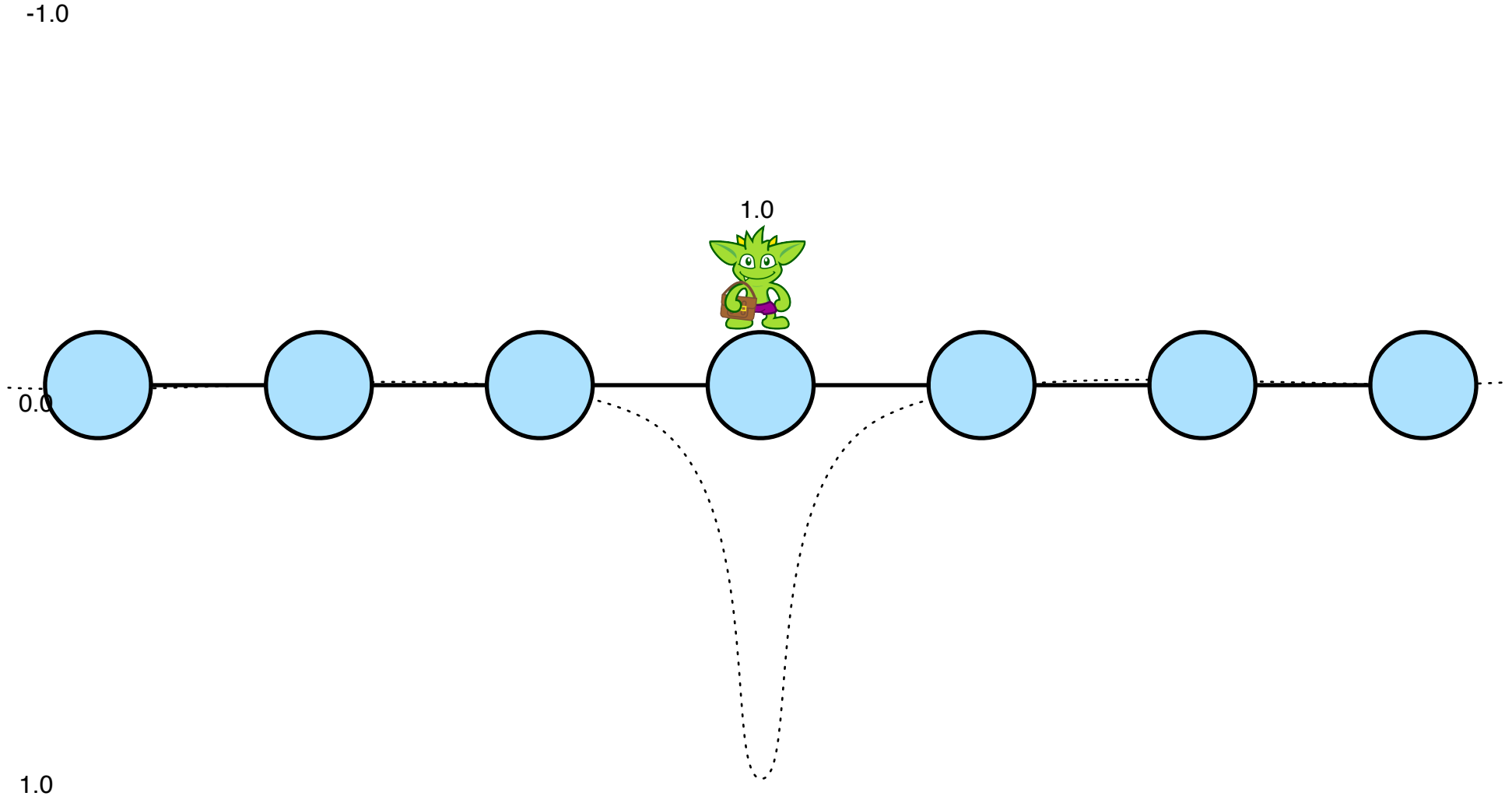
"But traversers are indivisible objects!
Energy must be able to be
divided indefinitely and take negative values."

Lets look at Gremlin Sacks.

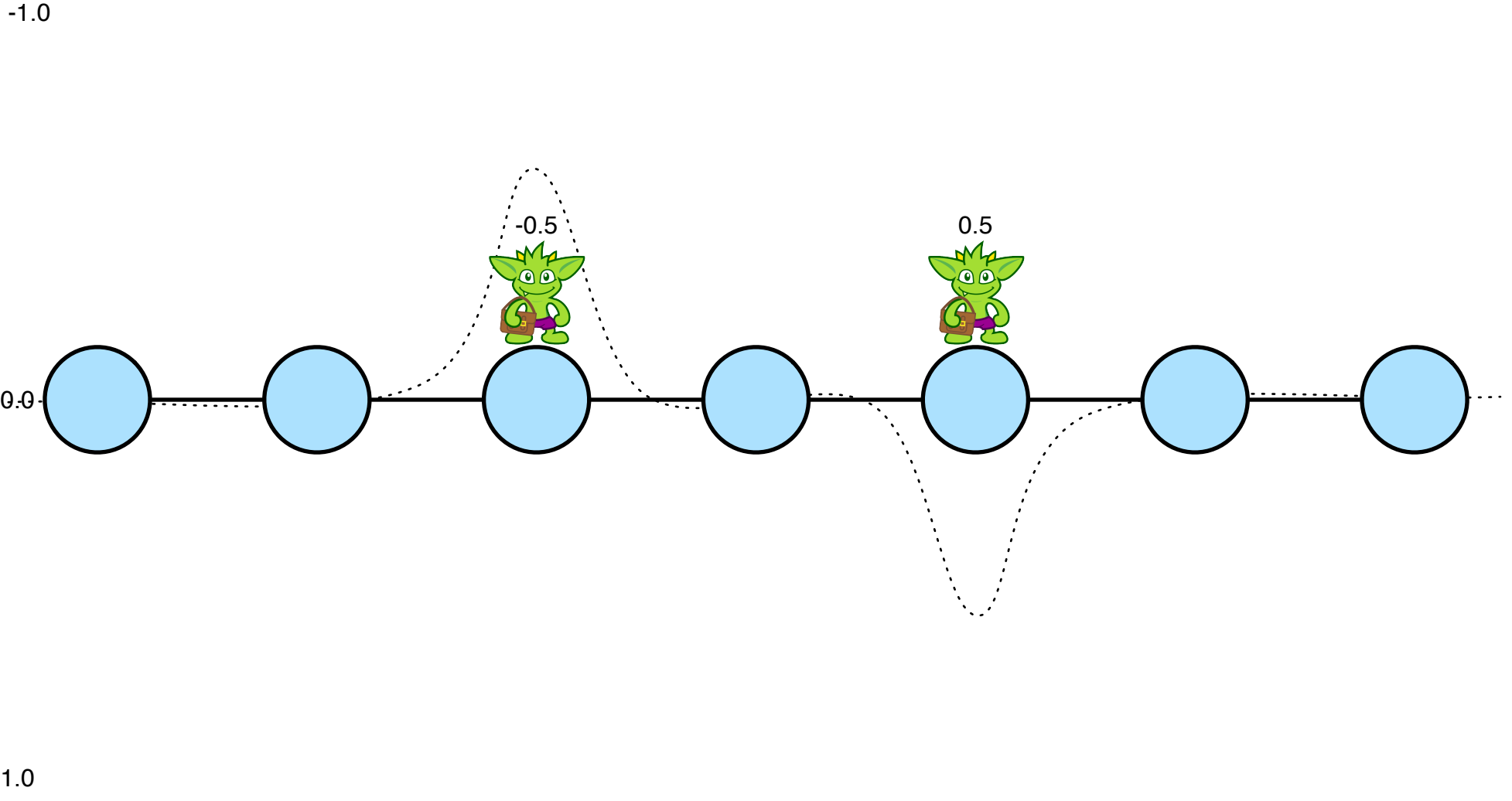
The traverser is not divisible, but that
double floating point number in his sack is!

Gremlin Sacks

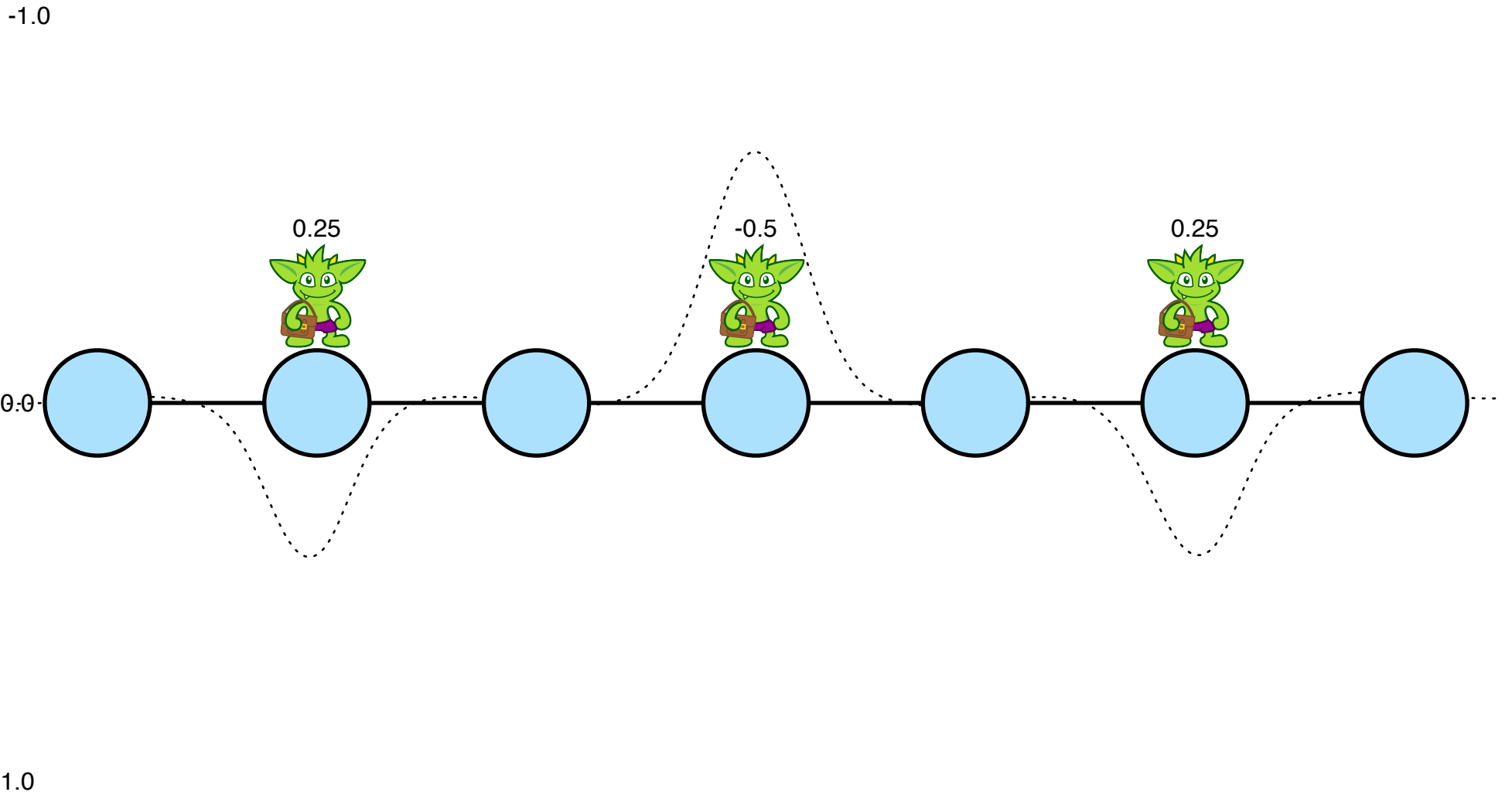
(a data structure local to the traverser)
(each traverser can carry an object along its way)



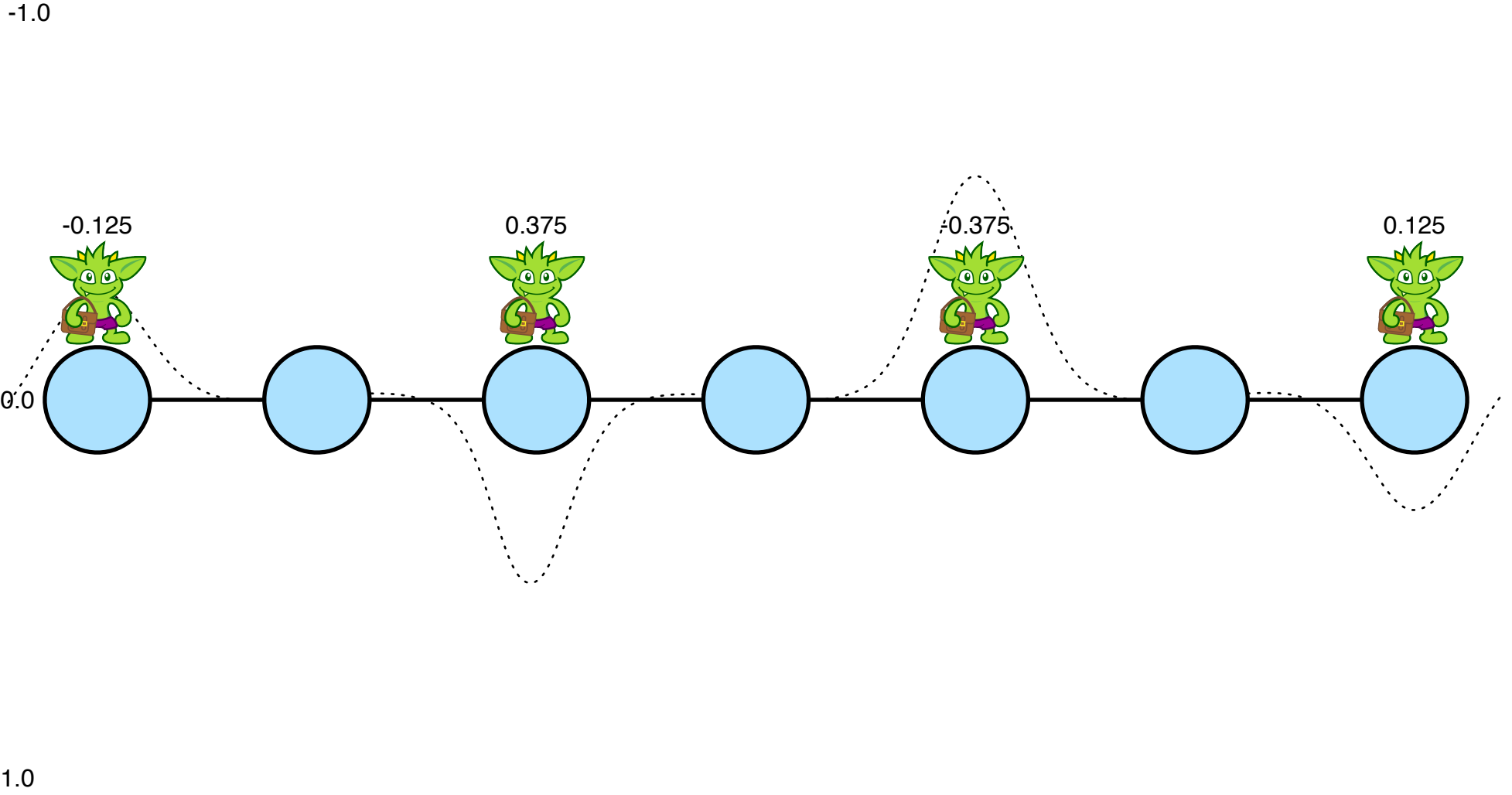
Don't worry about the algorithm, just watch the visualization for now.



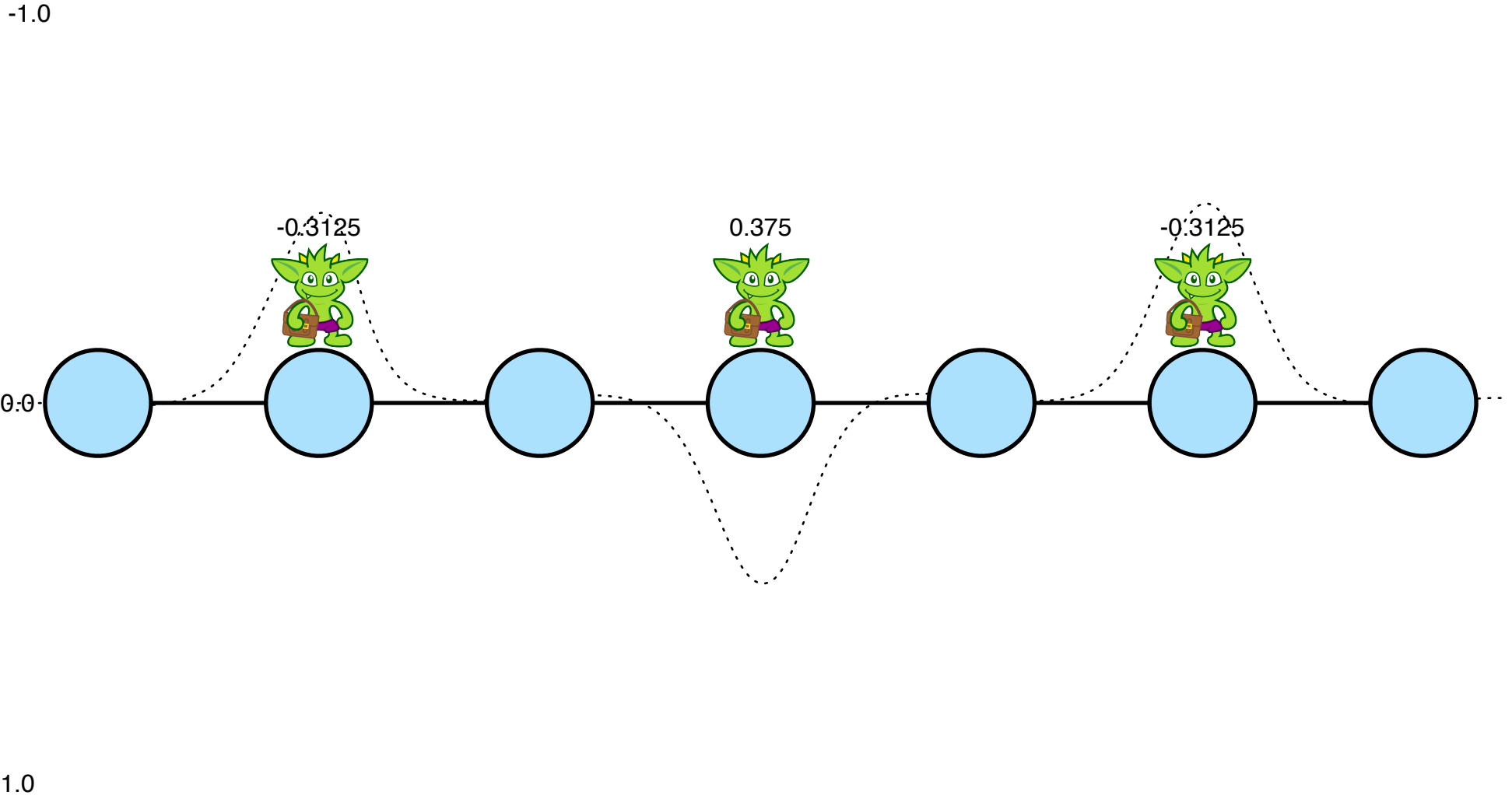
Don't worry about the algorithm, just watch the visualization for now.



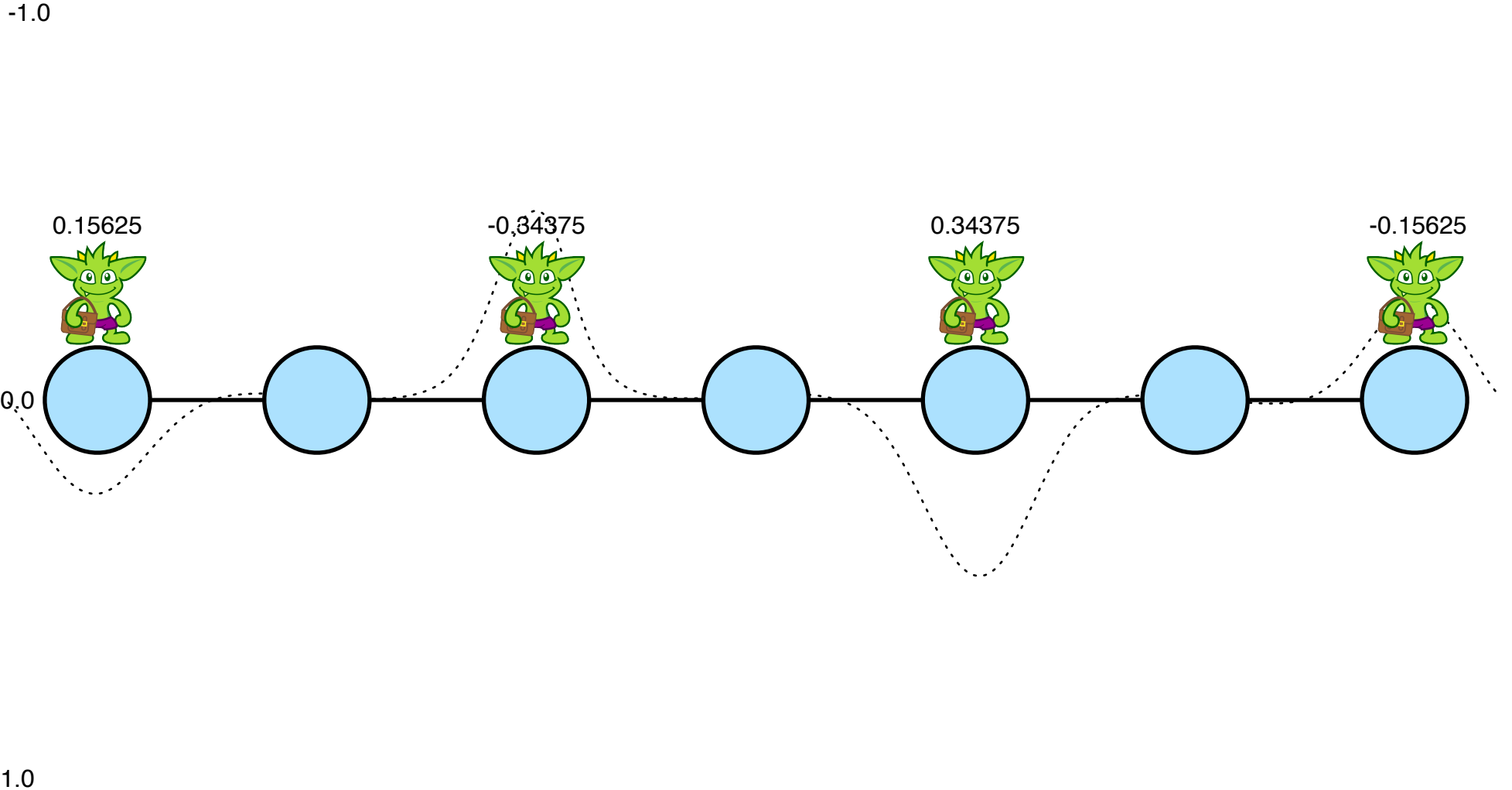
Don't worry about the algorithm, just watch the visualization for now.



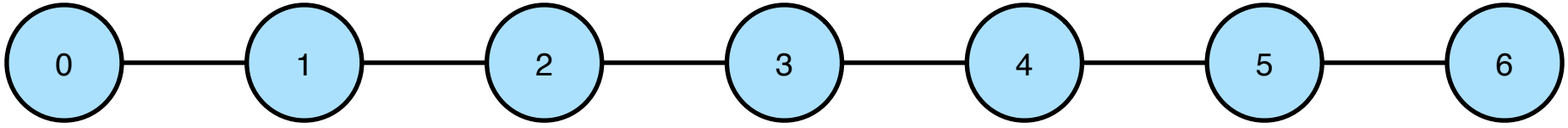
Don't worry about the algorithm, just watch the visualization for now.



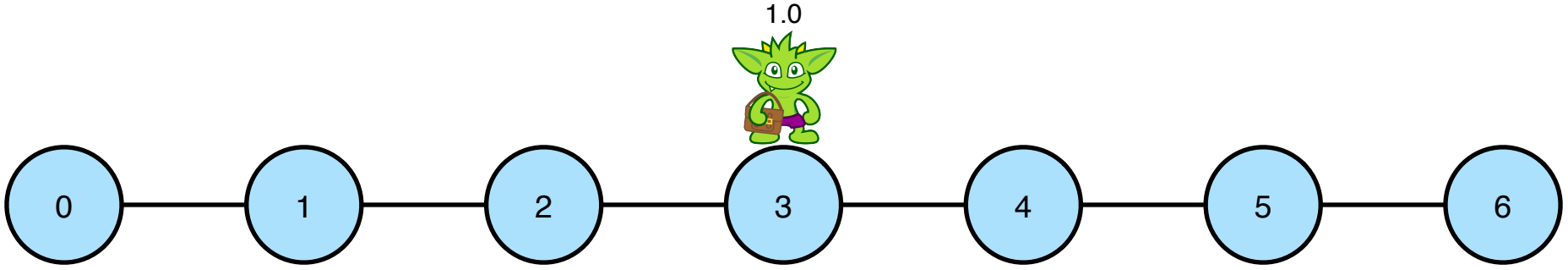
Don't worry about the algorithm, just watch the visualization for now.



Don't worry about the algorithm, just watch the visualization for now.

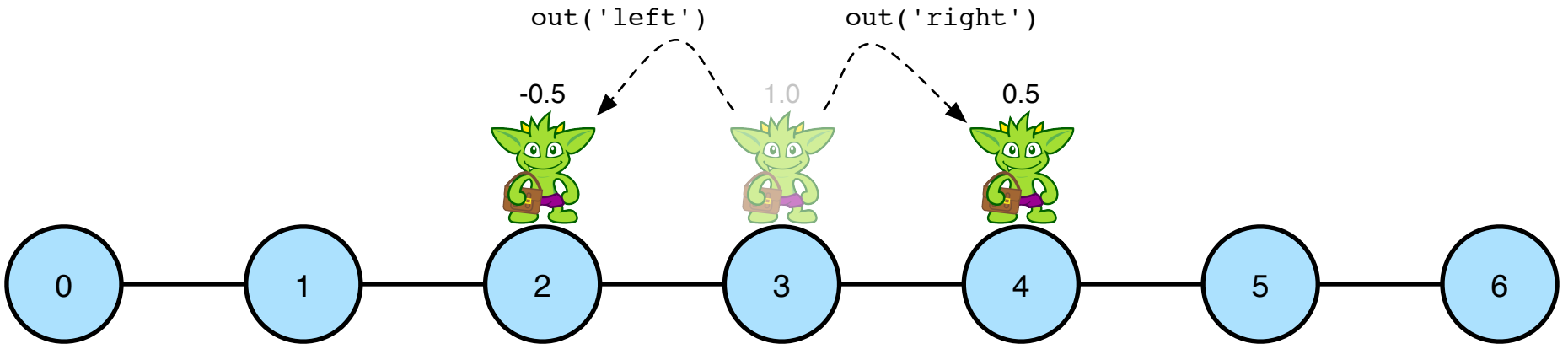


The graph.



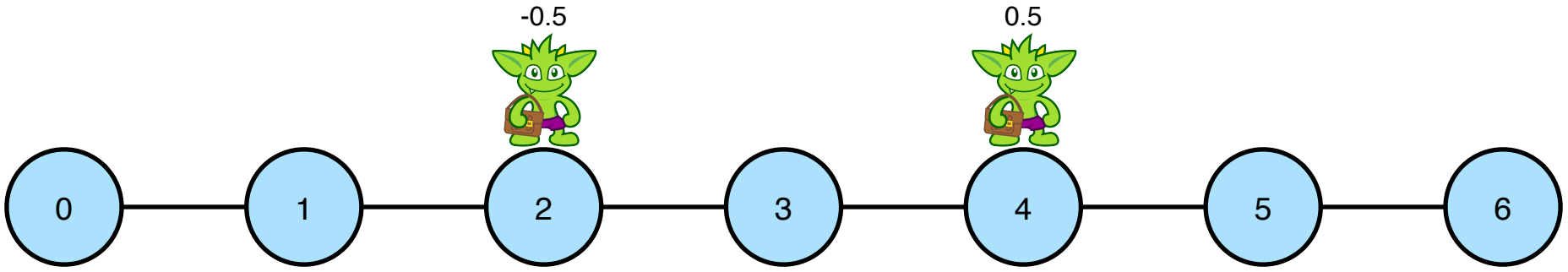

```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

An initial energy of 1.0 perturbs the graph.




```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

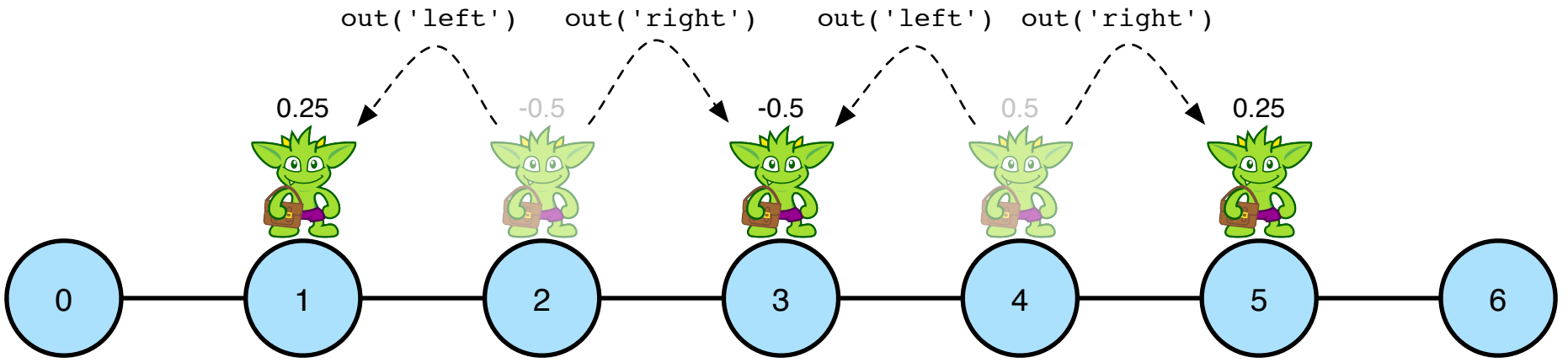
The energy diffuses left and right along the graph.




```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

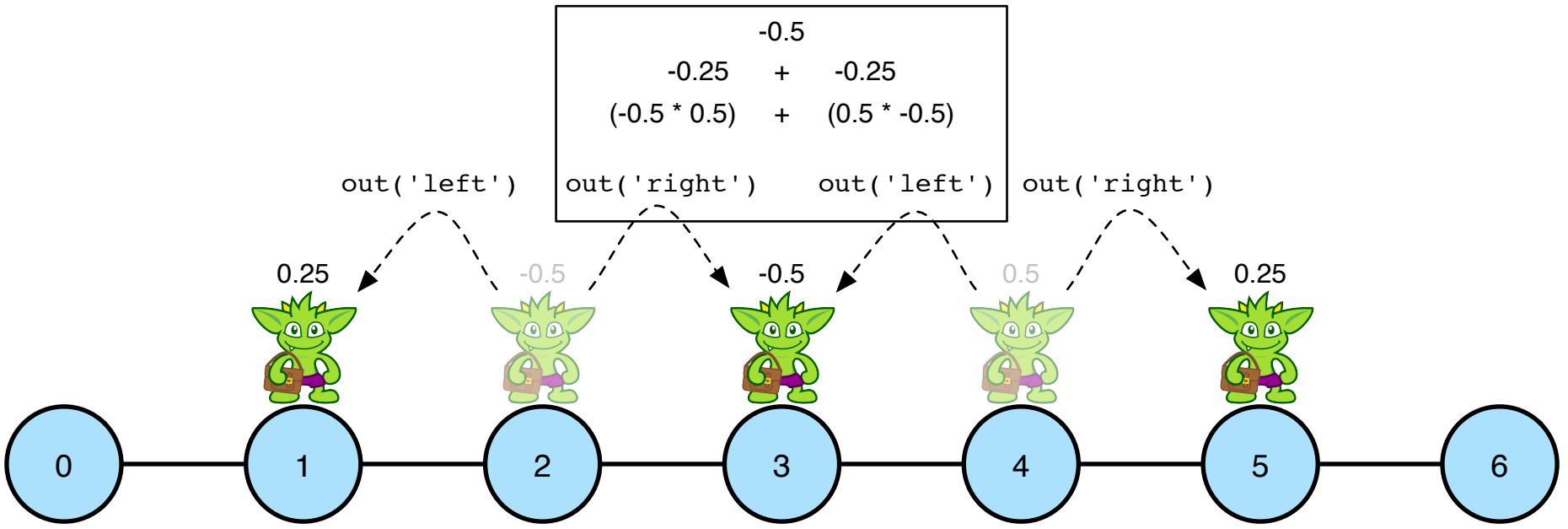
1 time

The total energy in the system is always 1.



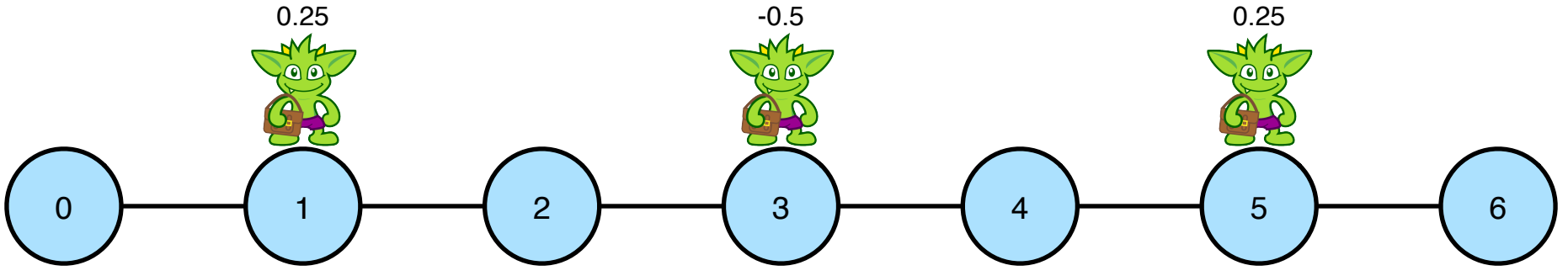

```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

Energy waves reverberate and thus, prior "left energy" can go right.



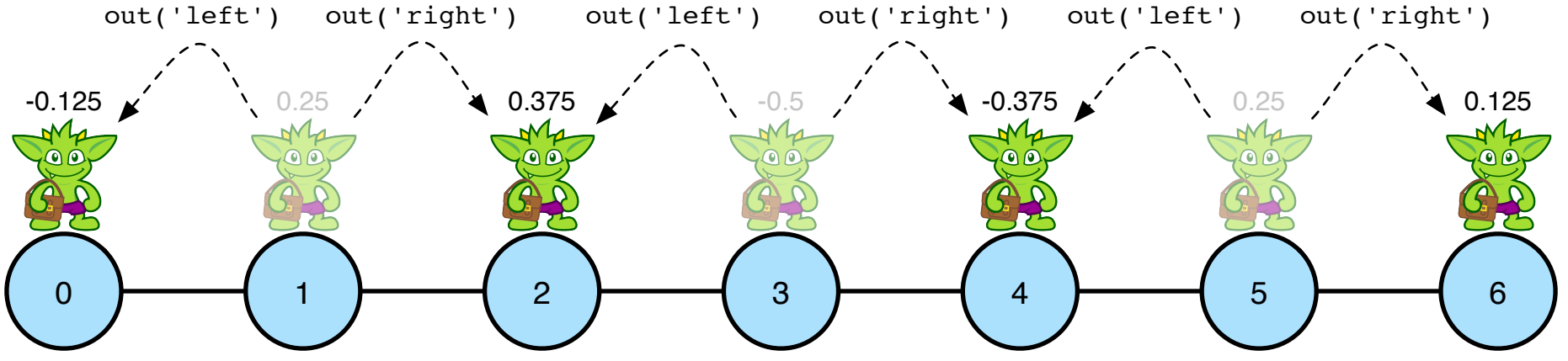

```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

Energy constructively and destructively interferes.

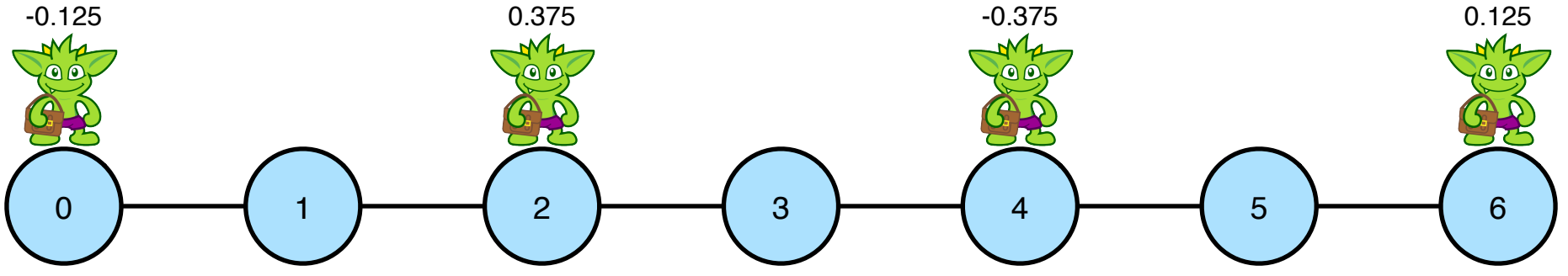



```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

2 times




```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```




```
g.withSack(1.0,sum).V(3).
  repeat(
   union(
     sack(mult).by(-0.5).out('left'),
     sack(mult).by(0.5).out('right')
   )
  ).times(3)
```

3 times

A wave is a diffusion of energy within a space.

The space is the graph.

The energy is contained within the traversers (via their sacks).

When a traverser splits, its sack energy is divided amongst its children traversers' sacks.

When traversers merge, their sack energy is summed to a single traverser sack.

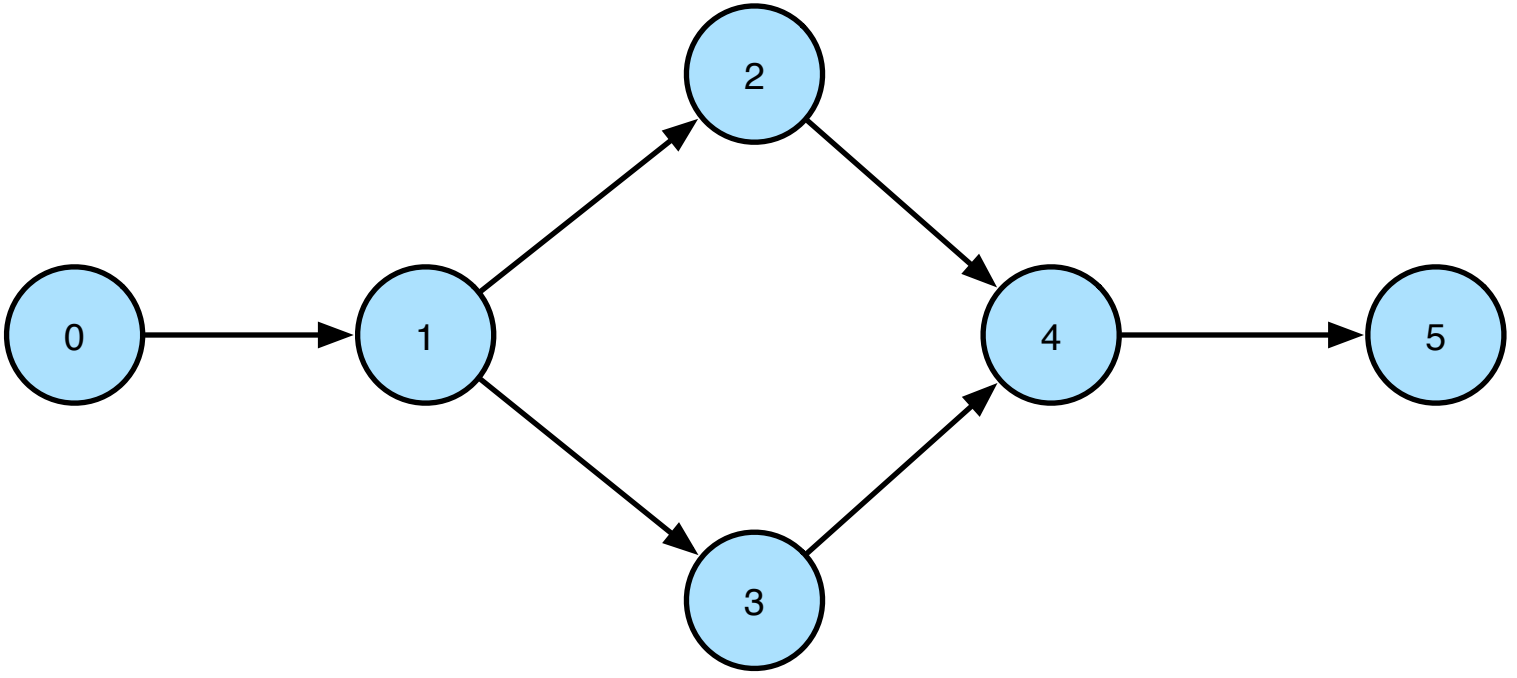
If two energy sacks with the same parity are merged, *constructive interference*.

If a negative and a positive energy sack are merged, *destructive interference*.

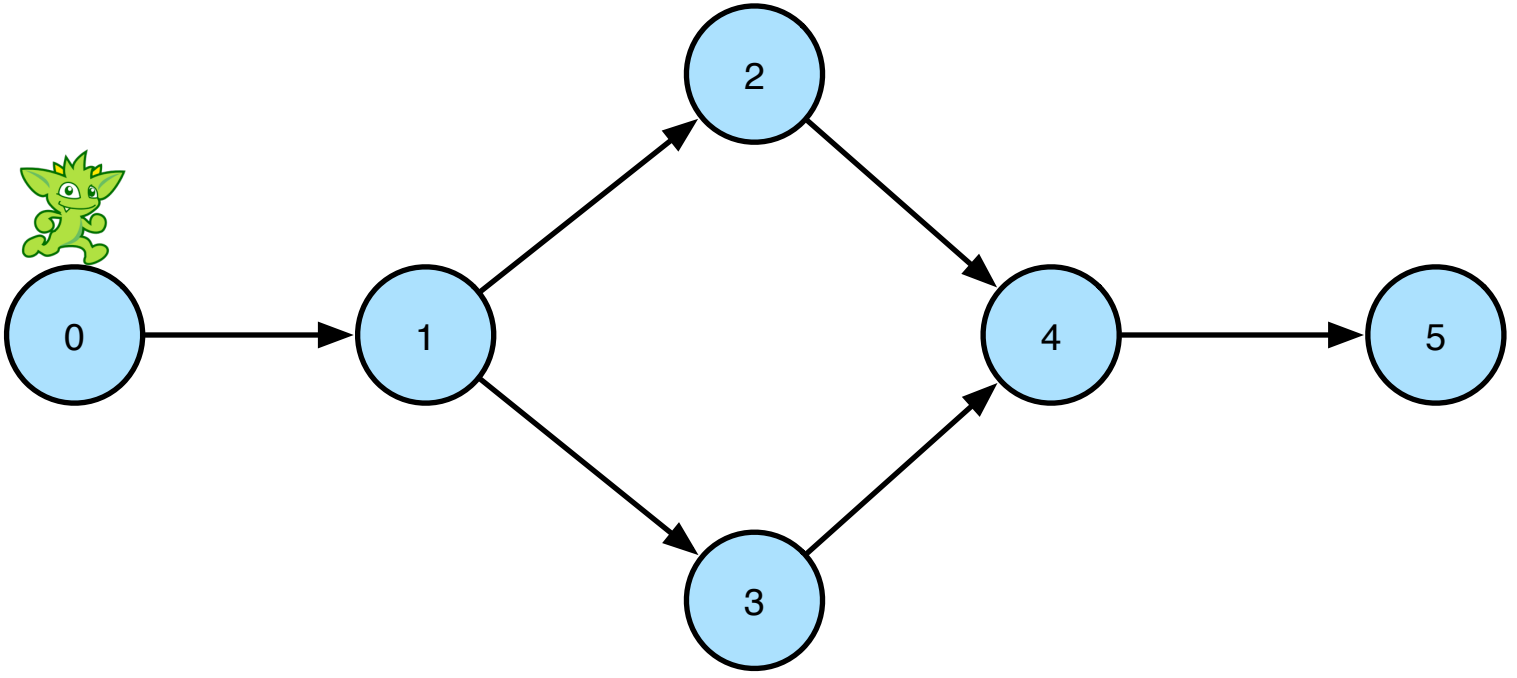
Part 2

Quantum Mechanics via the Copenhagen Interpretation

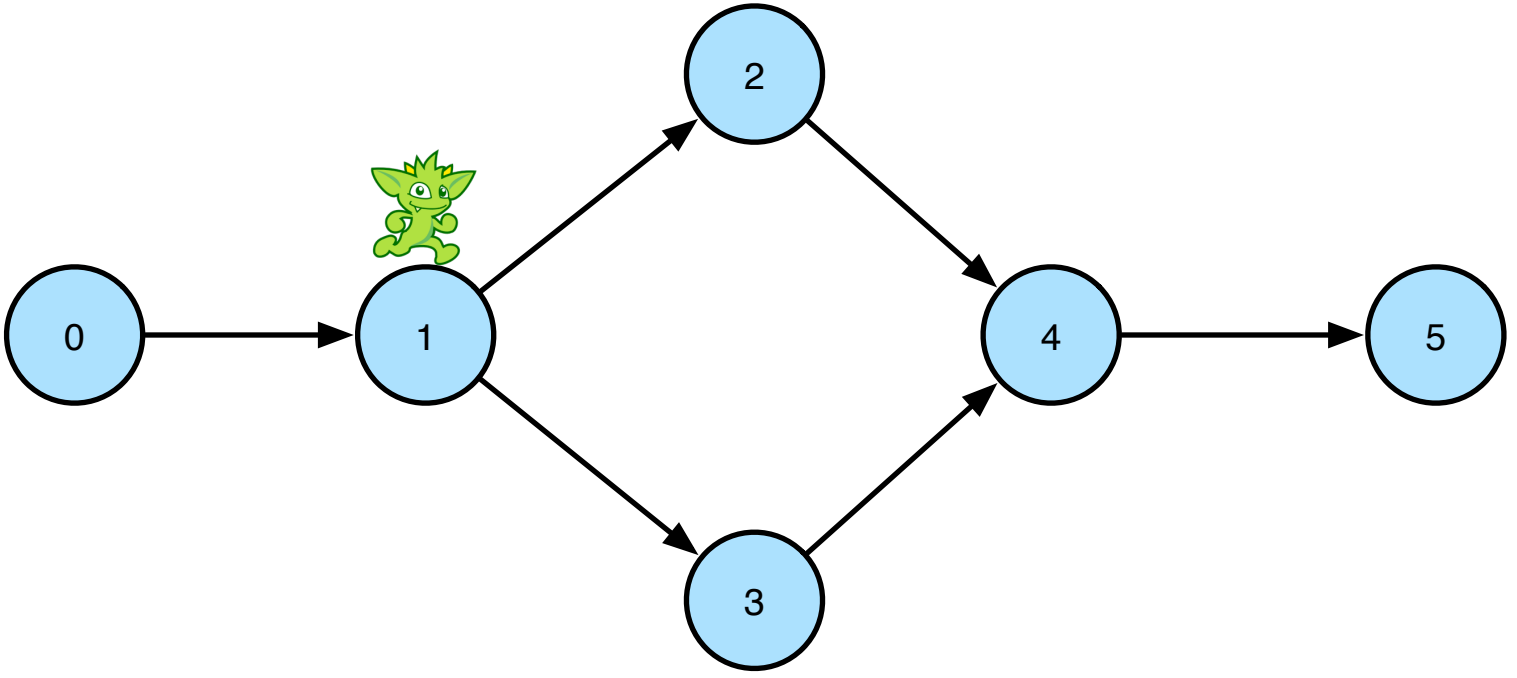
Space = Graph



Particle = Traverser

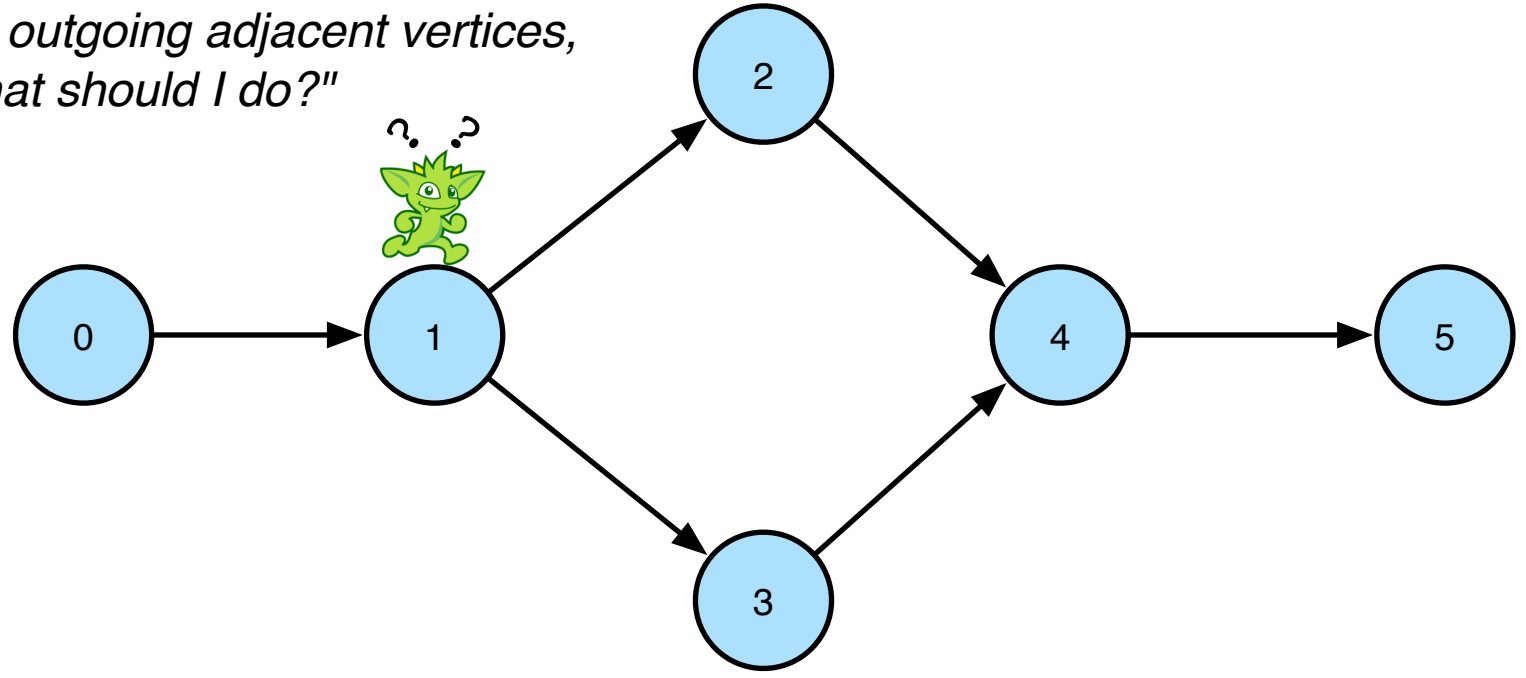



```
gremlin> g.V(0)
==>v[0]
```

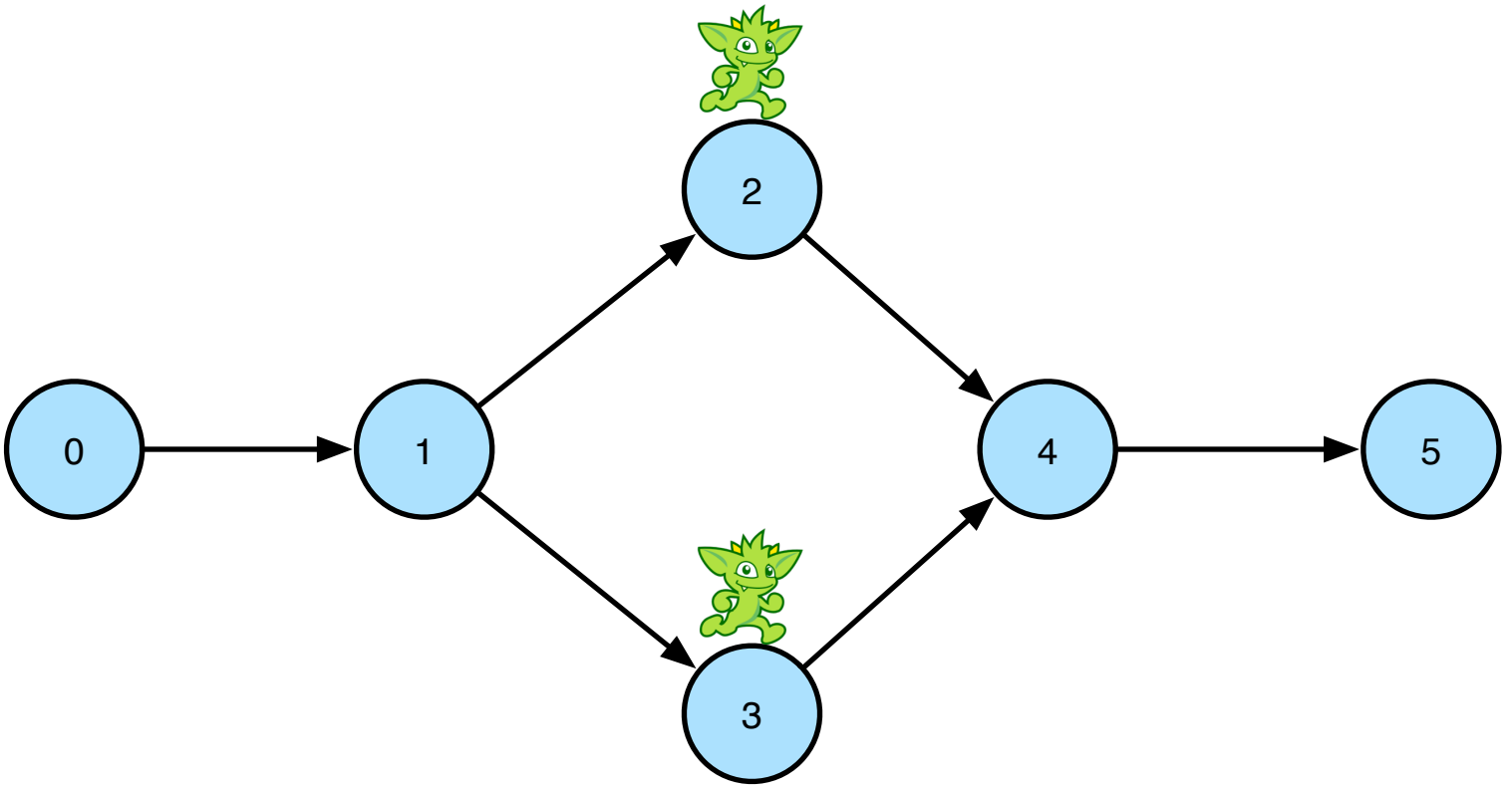



```
gremlin> g.V(0).out()
==>v[1]
```

"Two outgoing adjacent vertices, what should I do?"



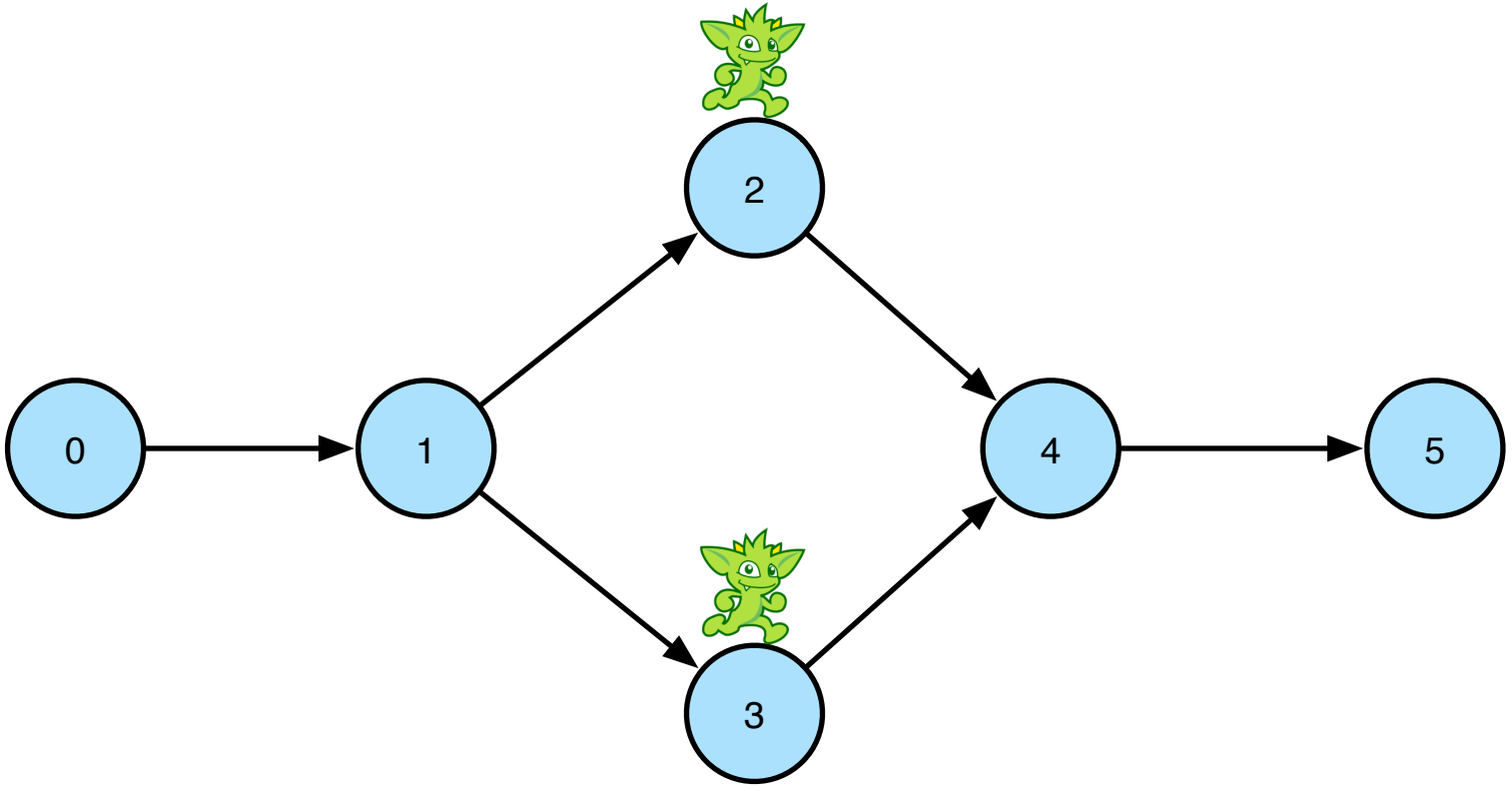

```
gremlin> g.V(0).out().out()
==>??
```




```
gremlin> g.V(0).out().out()
==>v[2]
==>v[3]
```

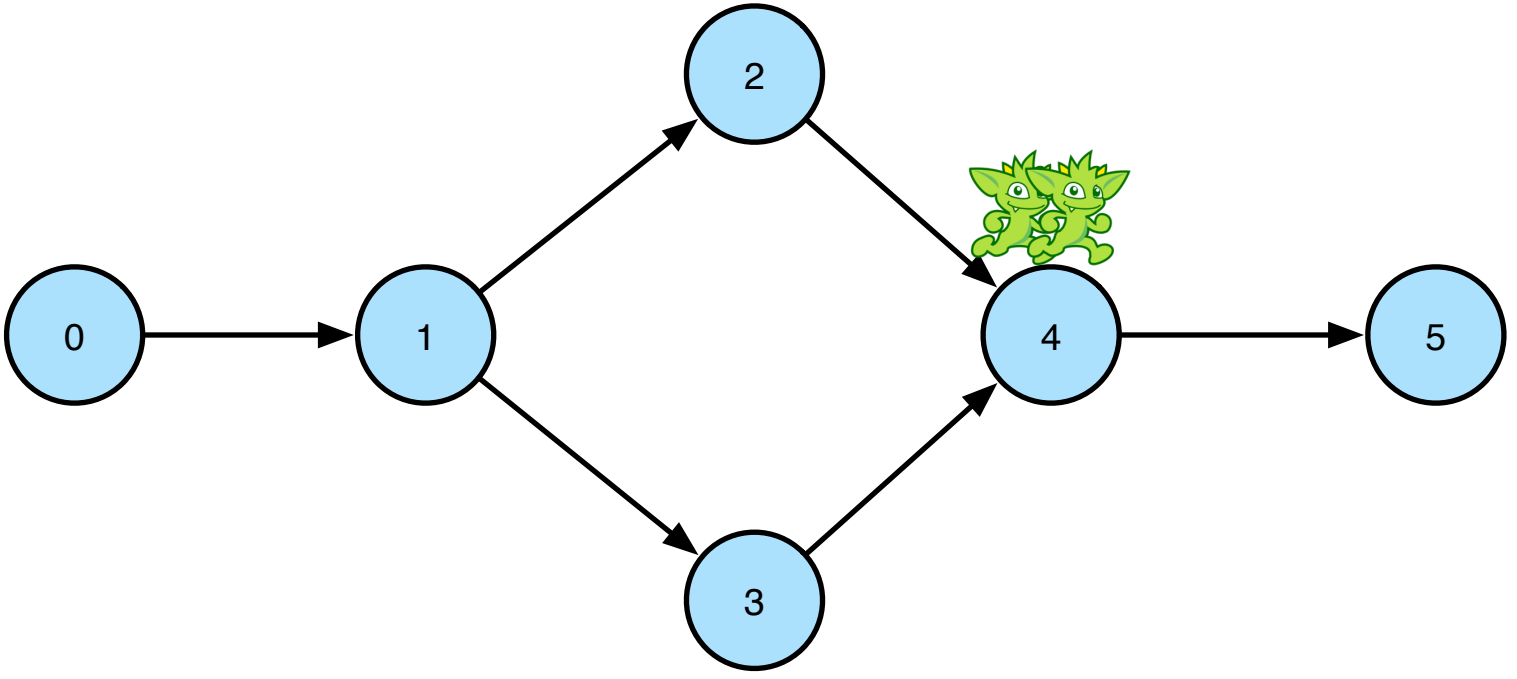
2 particles are created from 1.

In graph computing: "There are two legal paths, so take both."
In physical computing: "Violates conservation of energy law (matter is created)."



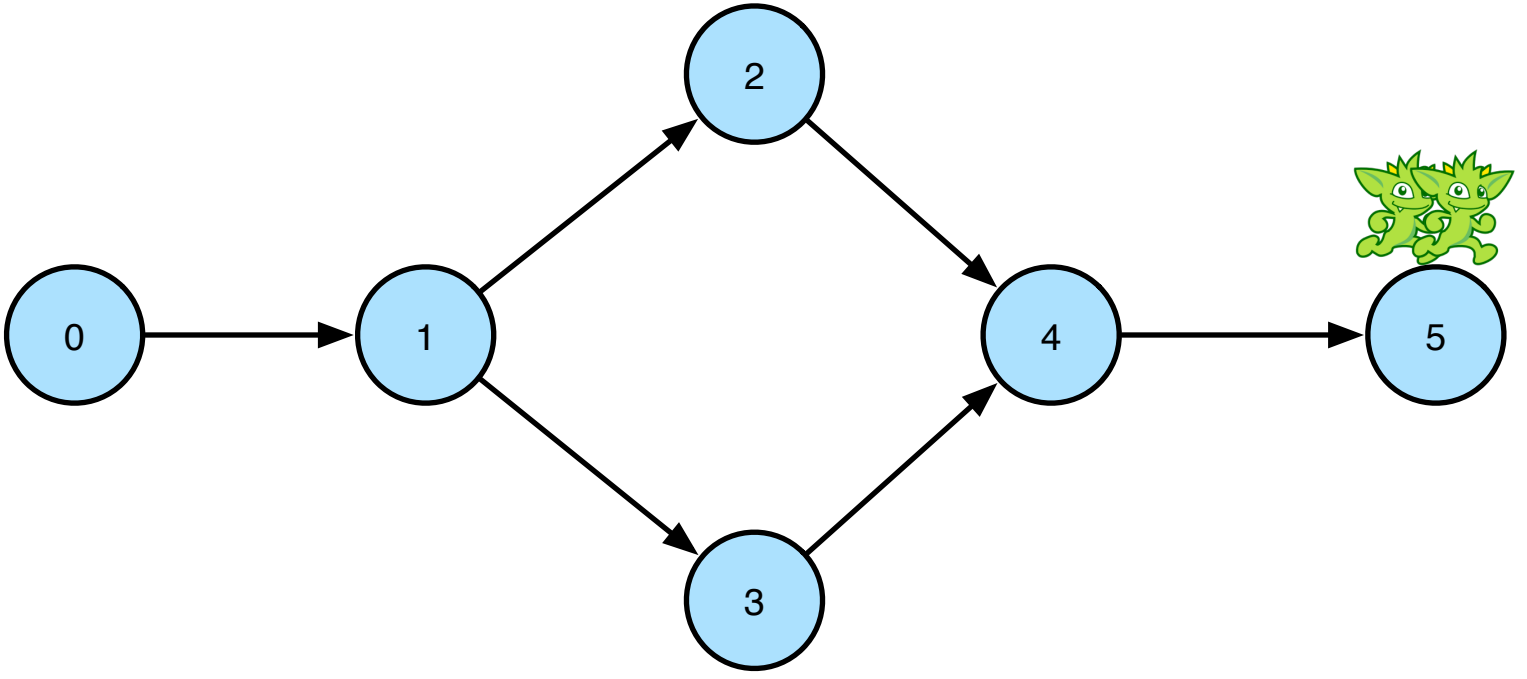

```
gremlin> g.V(0).out().out()
==>v[2]
==>v[3]
```

In graph computing: "Two length 3 paths lead to vertex 4."
In physical computing: "The twin particles coexist at vertex 4."



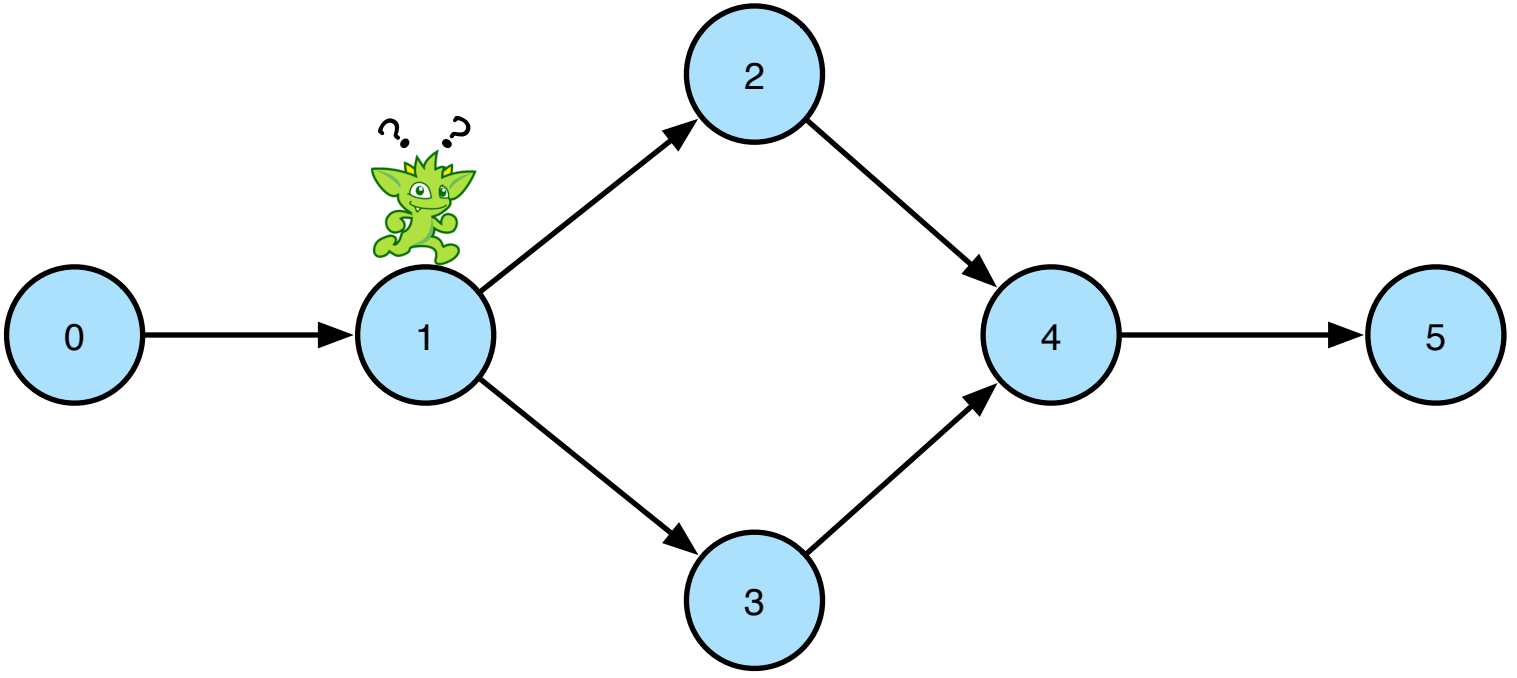

```
gremlin> g.V(0).out().out().out()
==>v[4]
==>v[4]
```

In graph computing: "Two length 3 paths lead to vertex 5."
In physical computing: "The twin particles coexist at vertex 5."

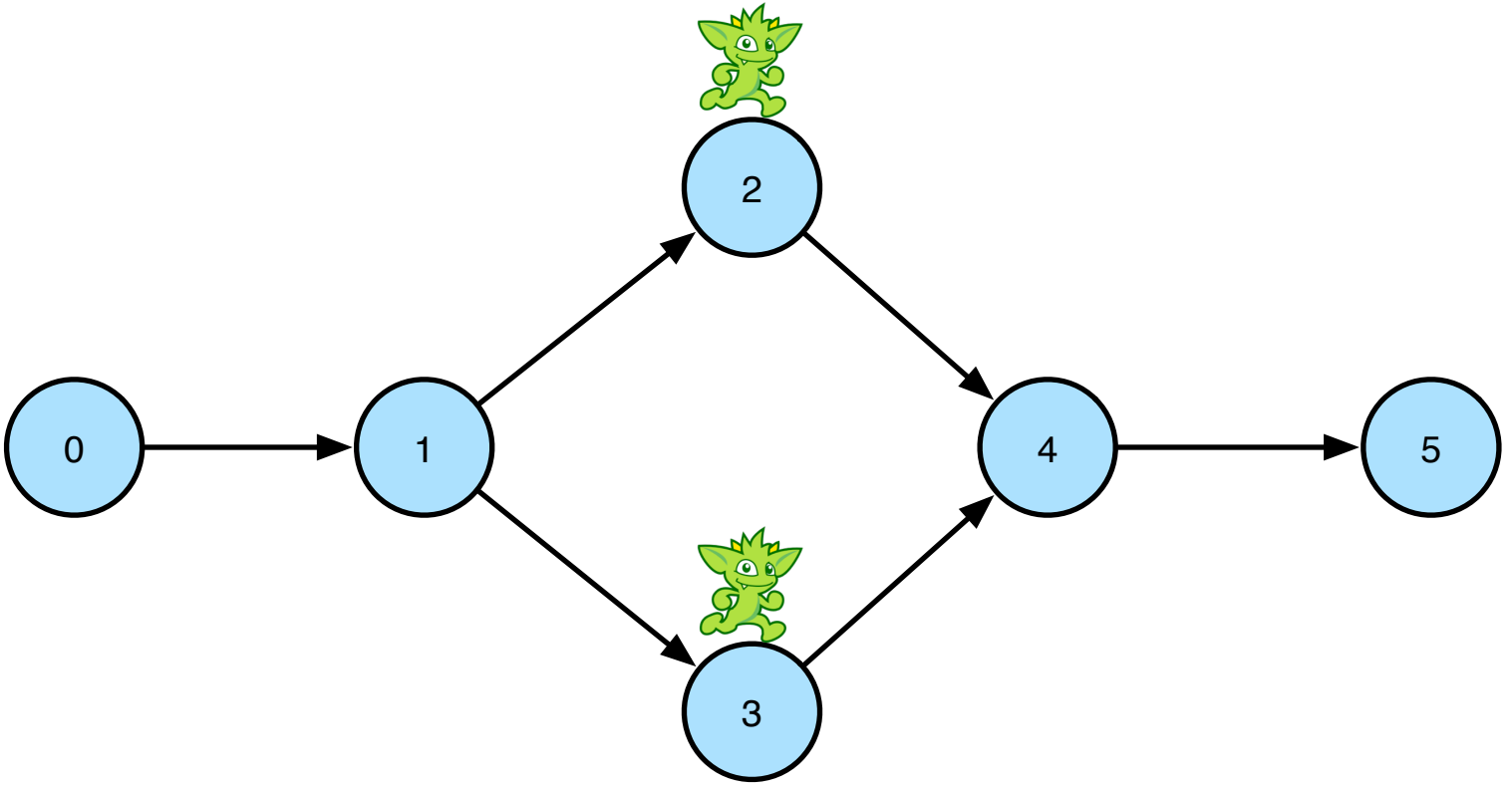



```
gremlin> g.V(0).out().out().out().out()
==>v[5]
==>v[5]
```

What does a **physical particle** do when it is faced with a choice?



The particle takes both options!

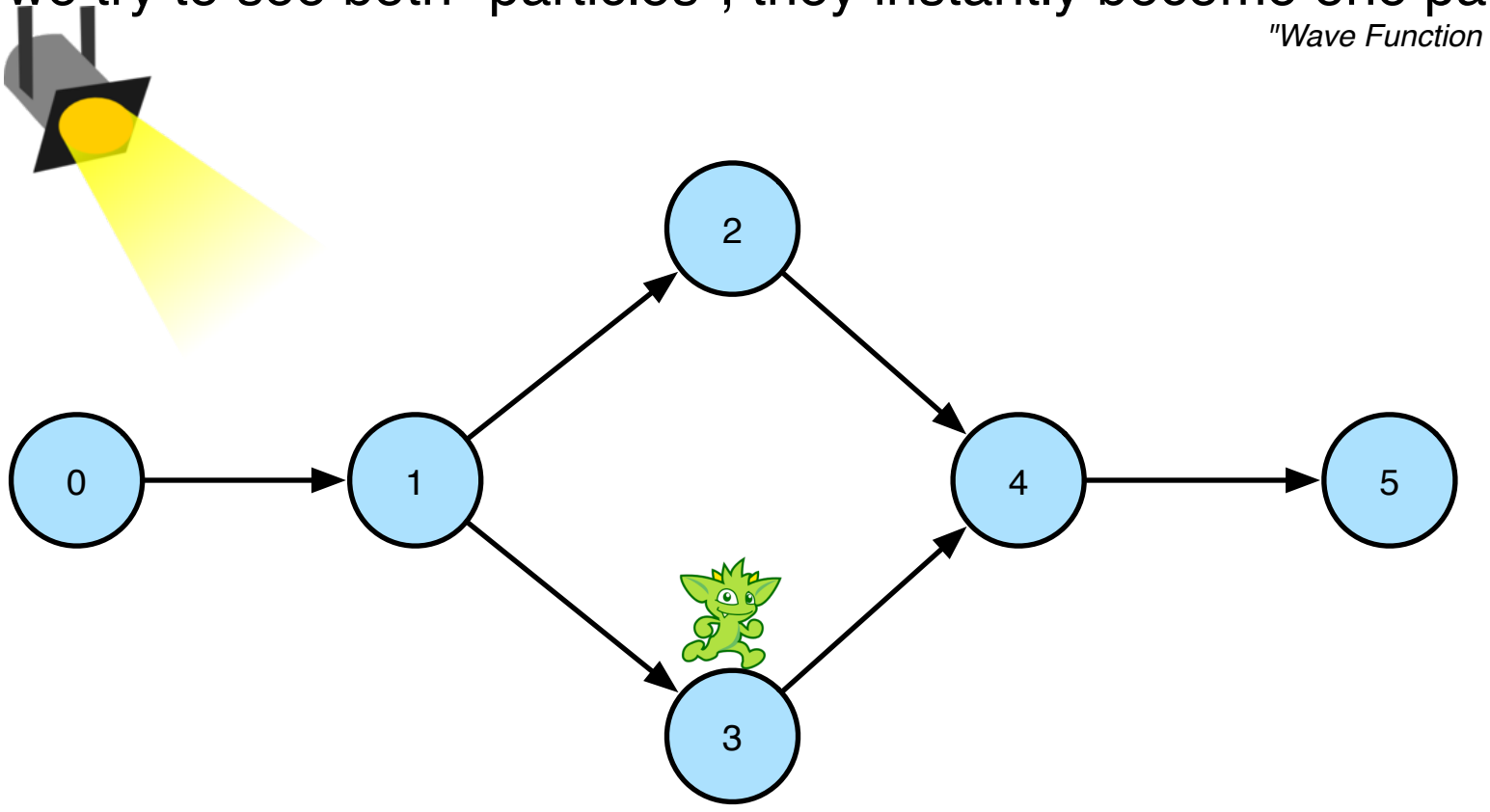


The indivisible "particle" divides!

Two particles can't be "seen," but we know both options are taken.

When we try to see both "particles", they instantly become one particle.

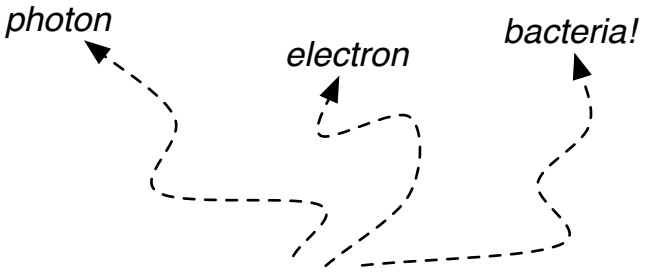
"Wave Function Collapse"



"Why do you say the particle took both paths?
If you find it only on a single path, then wasn't only one path taken?"

Explanation coming soon...

Wave interference!



The smallest "quanta" allowed is the particle.

Only one particle is ever directly observed.

However, *prior to observation*, the particle splits into pieces (**a wave**).

When we *observe* the system, the pieces becomes one again (**a particle**).

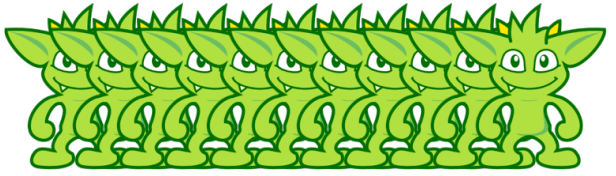
Other Interpretations of Quantum Mechanics
Multi-World Interpretation
Pilot Wave Interpretation

Part 3

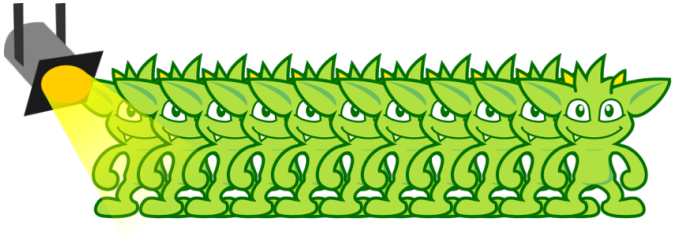
The Quantum Wave Function

Classical Particle
(A single unique "thing" exists)

The Particle Diffuses over Space as a Wave
(The particle is in a quantum superposition)



The Wave Function is Perturbed
(e.g., light tries to "see" the particle)



The Wave Function Collapses to a Classical Particle
(The wave turns into a particle)

Observed

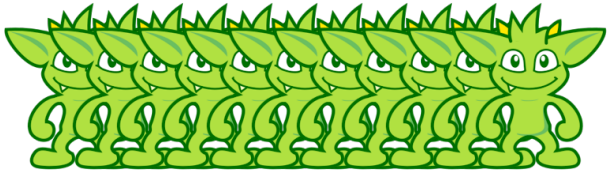
Classical Particle

(A single unique "thing" exists)

Inferred

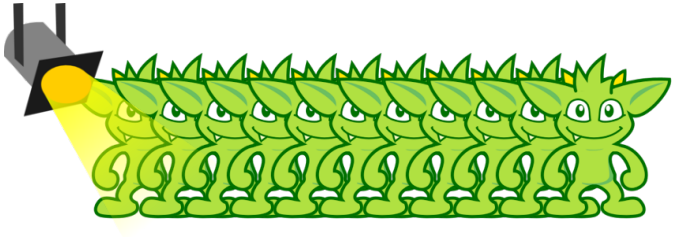
The Particle Diffuses over Space as a Wave

(The particle is in a quantum superposition)



The Wave Function is Perturbed

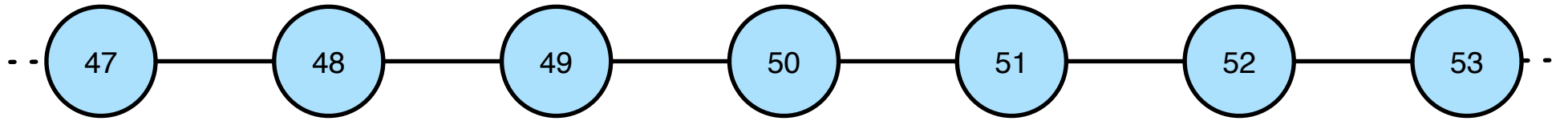
(e.g., light tries to "see" the particle)



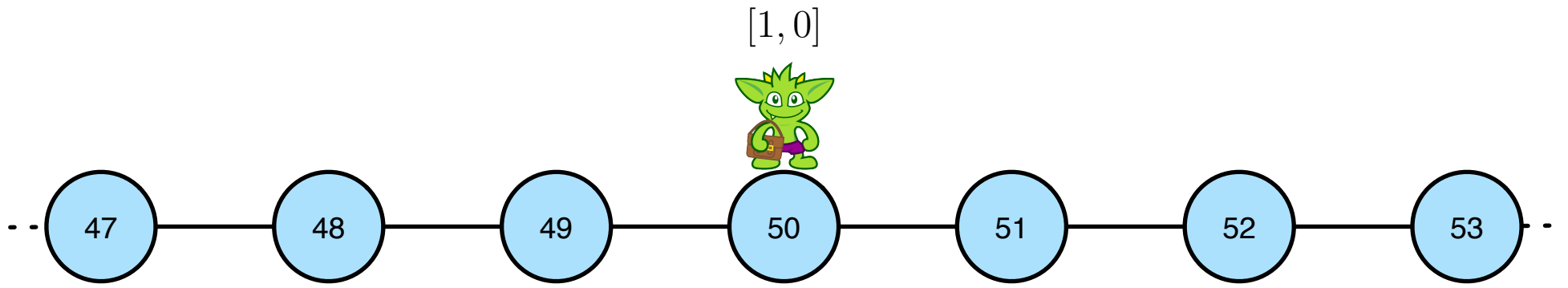
Observed

The Wave Function Collapses to a Classical Particle

(The wave turns into a particle)



A 101 vertex line graph with the center being vertex 50.



The particle has a **definite location** and a **spin**/potential to the left.

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

left-ness

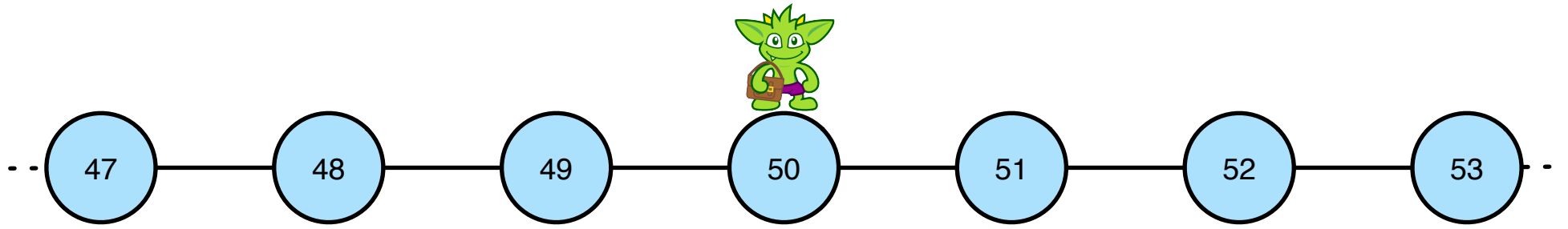
right-ness

For each option (degree of freedom),
we need a vector component.

In the article, we call it "traversal superposition."

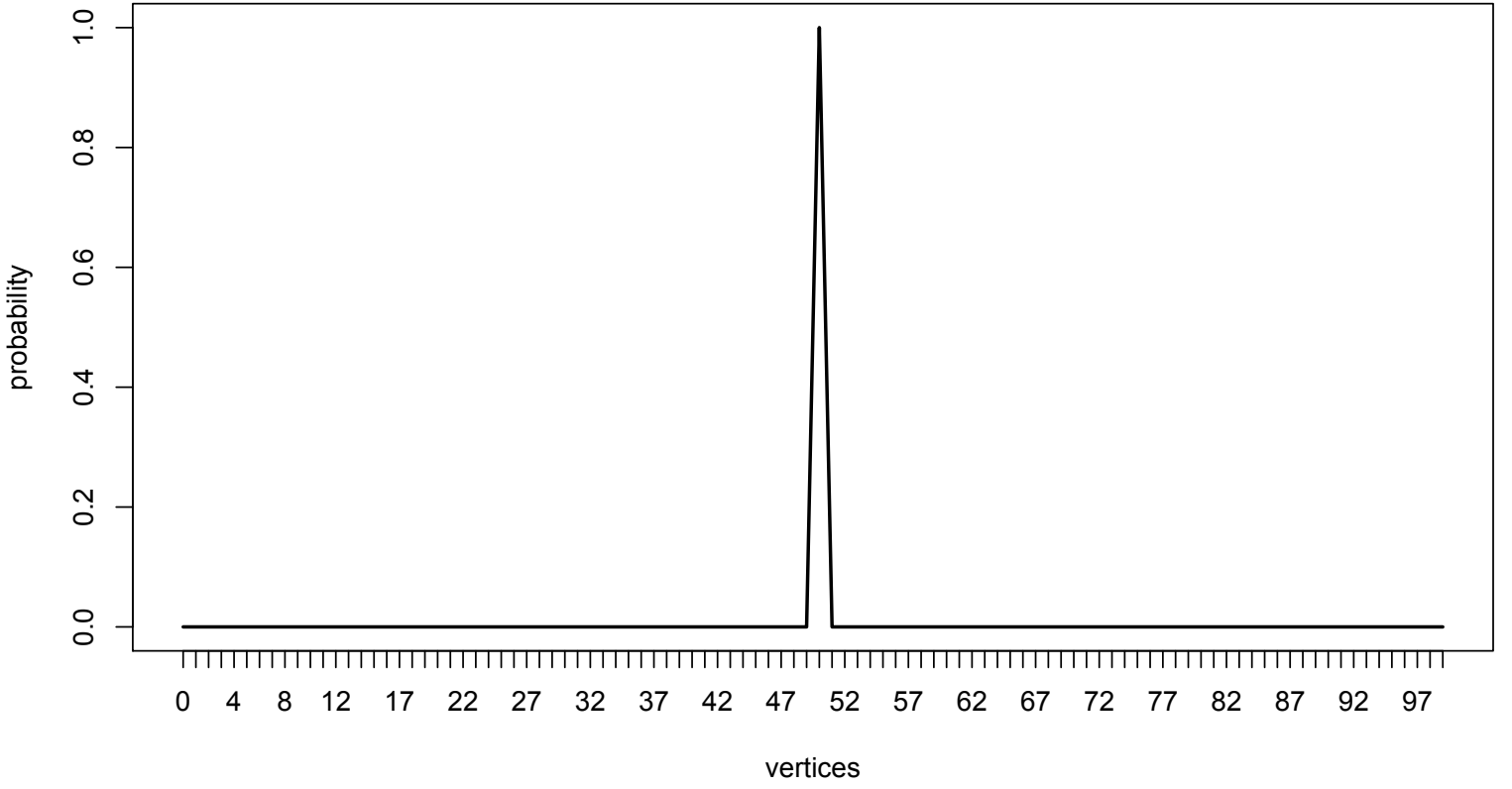
Classical Initial Step

$$[1, 0]$$



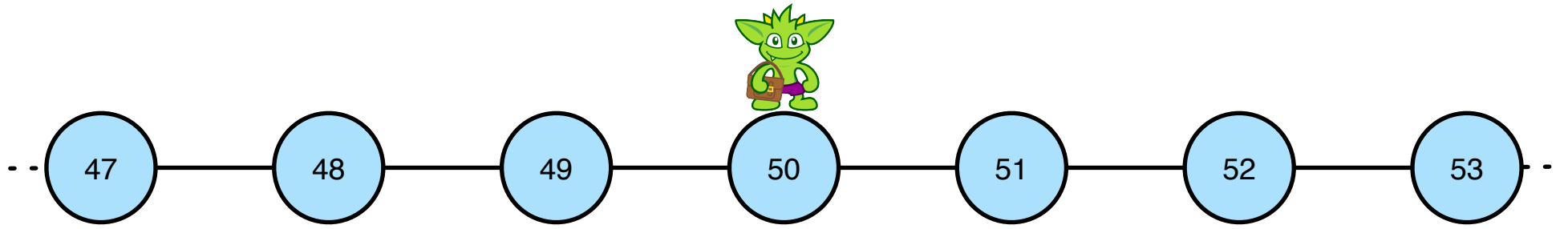
$$1^2 + 0^2 = 1$$

The probability of seeing the particle at vertex 50 is 100%.



Wave Function to Probability Distribution

$$[1, 0] \rightarrow \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$



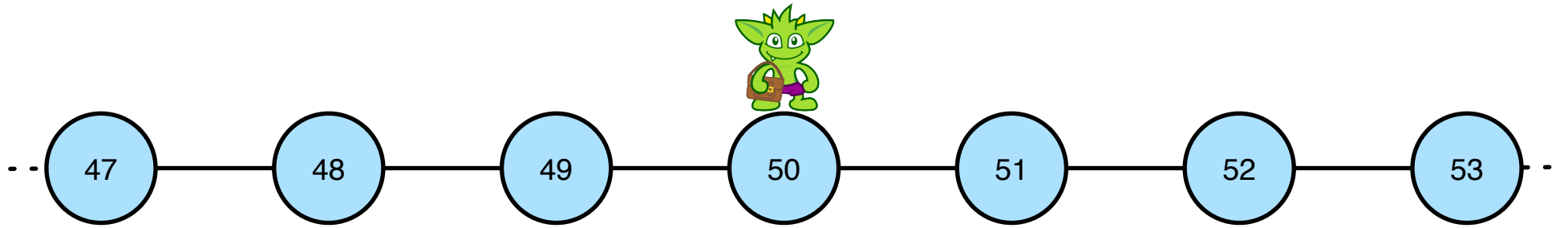
It has the option to go left or right so it goes both!

The particle's spin is put into superposition.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Coin Step

$$[1, 0] \rightarrow \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$



The particle's spin is put into superposition.

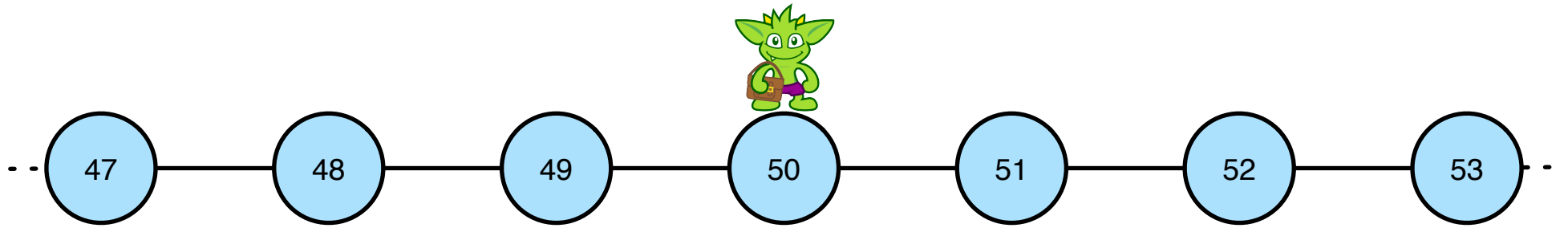
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

"left particles" going left or right keep their sign.

"right particles" going right flip their sign.

Coin Step

$$[1,0] \rightarrow \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$

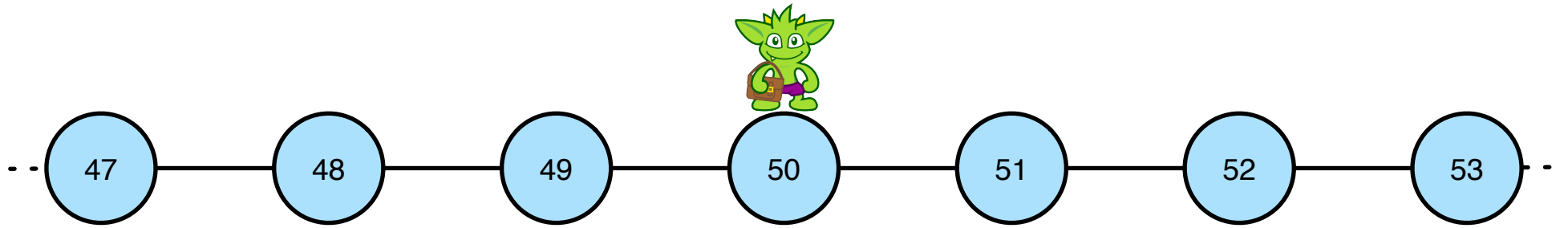


The particle's spin is put into superposition.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot [1,0] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \cdot [1,0] = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$

$$[1, 0] \rightarrow \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$



The particle's spin is put into superposition.

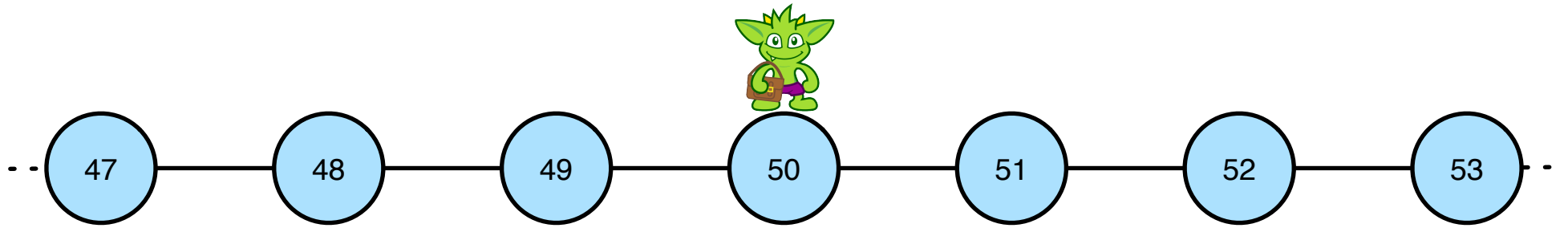
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot [1, 0] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \cdot [1, 0] = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$

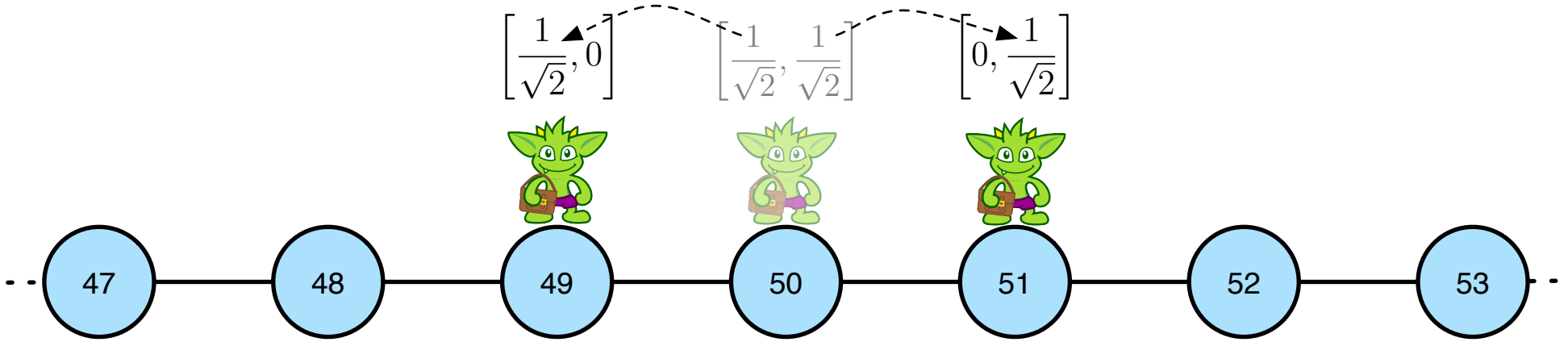
$$\left[\left(\left(1 \cdot \frac{1}{\sqrt{2}}\right) + \left(0 \cdot \frac{1}{\sqrt{2}}\right)\right), \left(\left(1 \cdot \frac{1}{\sqrt{2}}\right) + \left(0 \cdot -\frac{1}{\sqrt{2}}\right)\right)\right] = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$

Coin Step

$$\left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$$

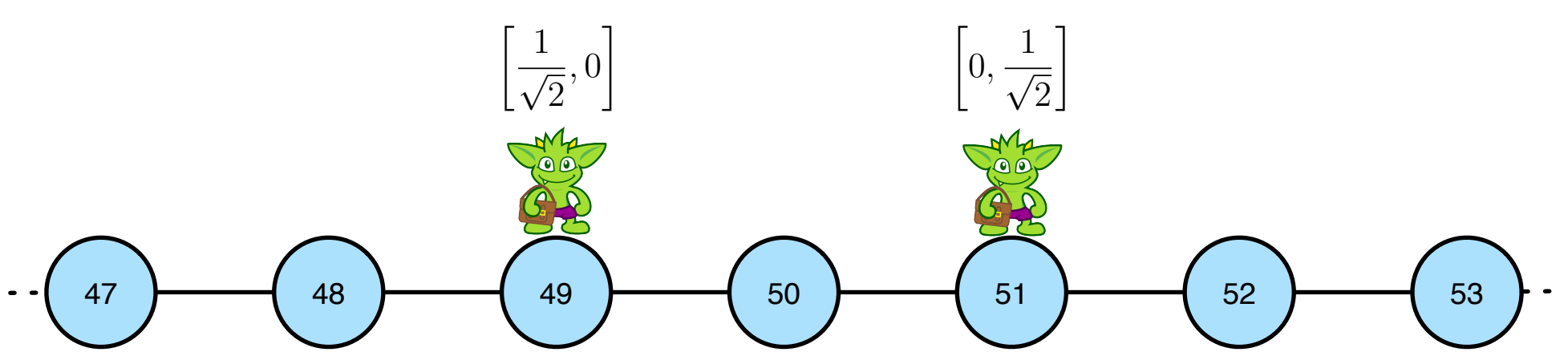


Coin Step

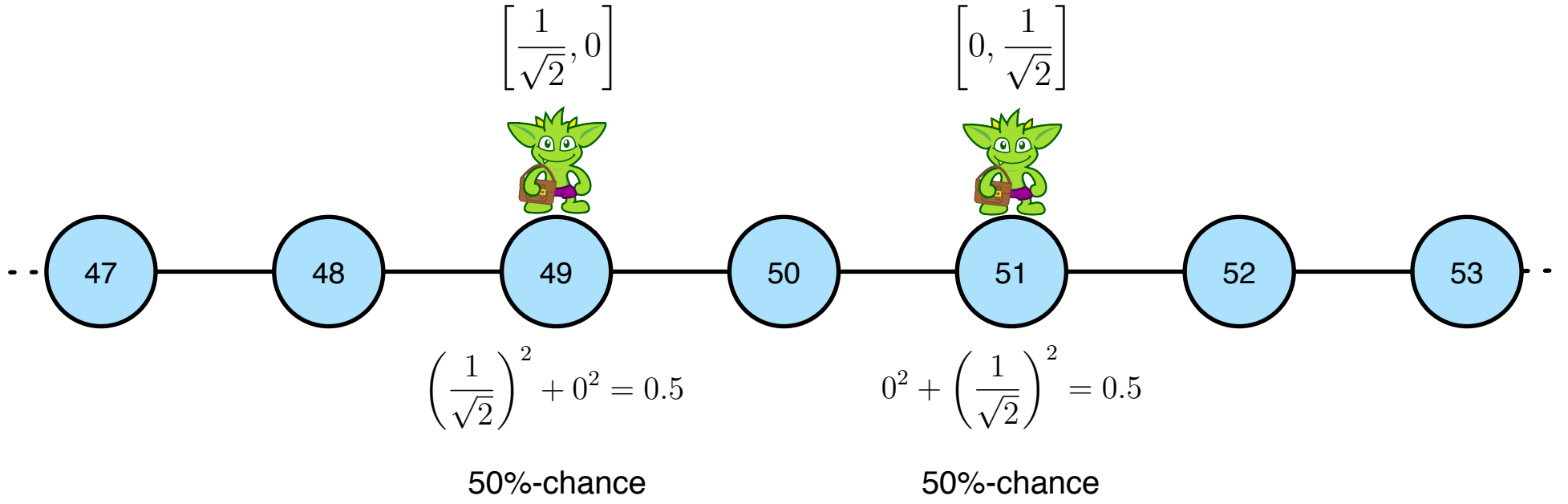


The "particle" spinning left goes left and
the "particle" spinning right goes right.

Shift Step


$\left[\frac{1}{\sqrt{2}}, 0\right]$
$\left[0, \frac{1}{\sqrt{2}}\right]$
47
48
49
50
51
52
53

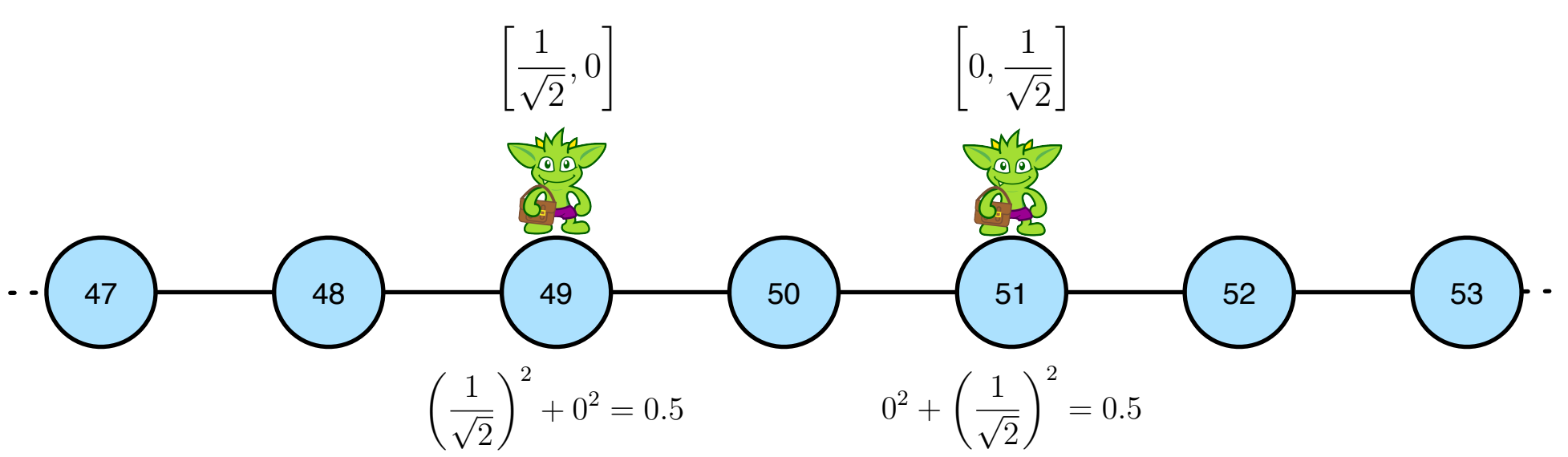
Shift Step



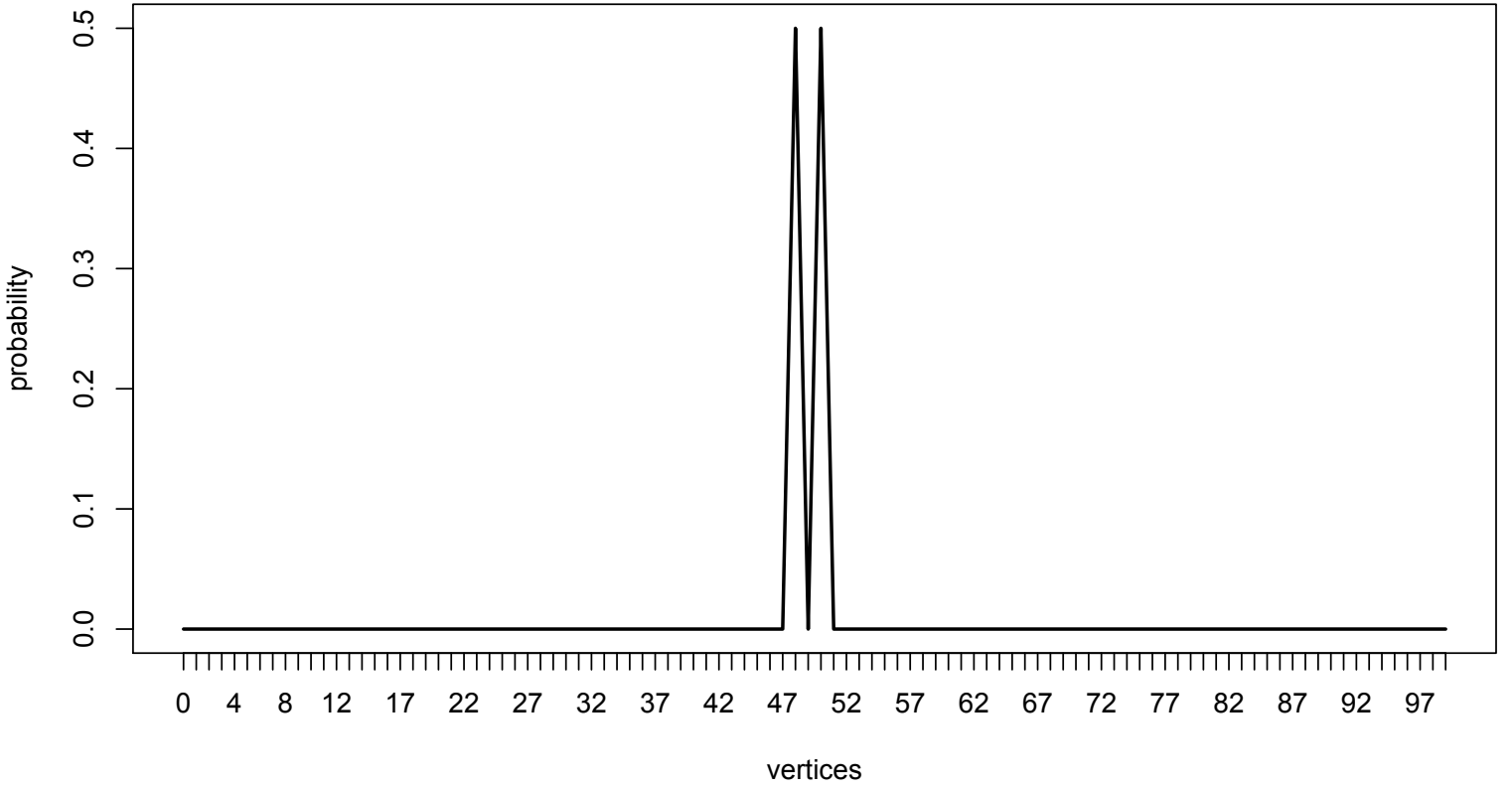
$$P(v) = |c_0|^2 + |c_1|^2$$

The probability of locating the classical particle each vertex is a function of the total wave amplitude at that vertex.

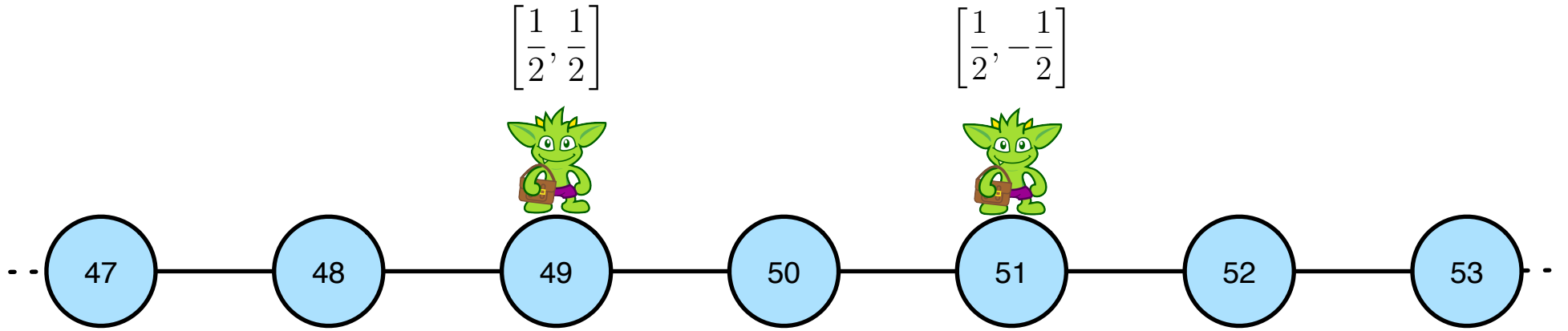
Wave Function to Probability Distribution



The probability of seeing the particle at vertex 49 or 51 is 50%.

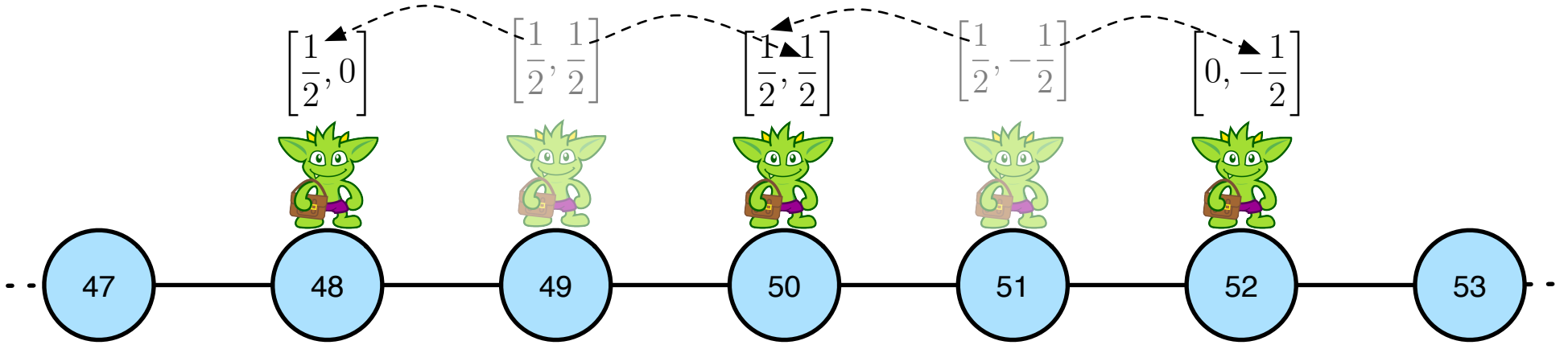


Wave Function to Probability Distribution



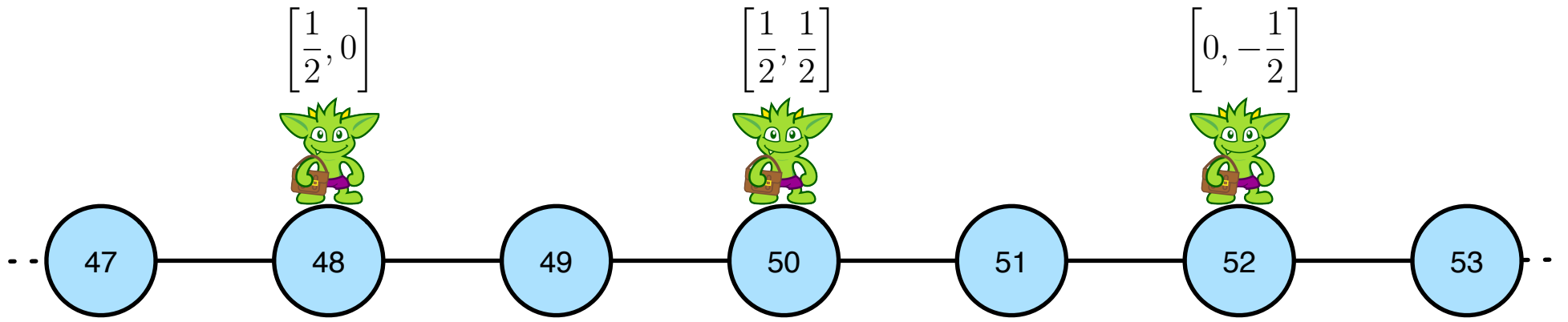
Each "particles"' spin is put into superposition.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$



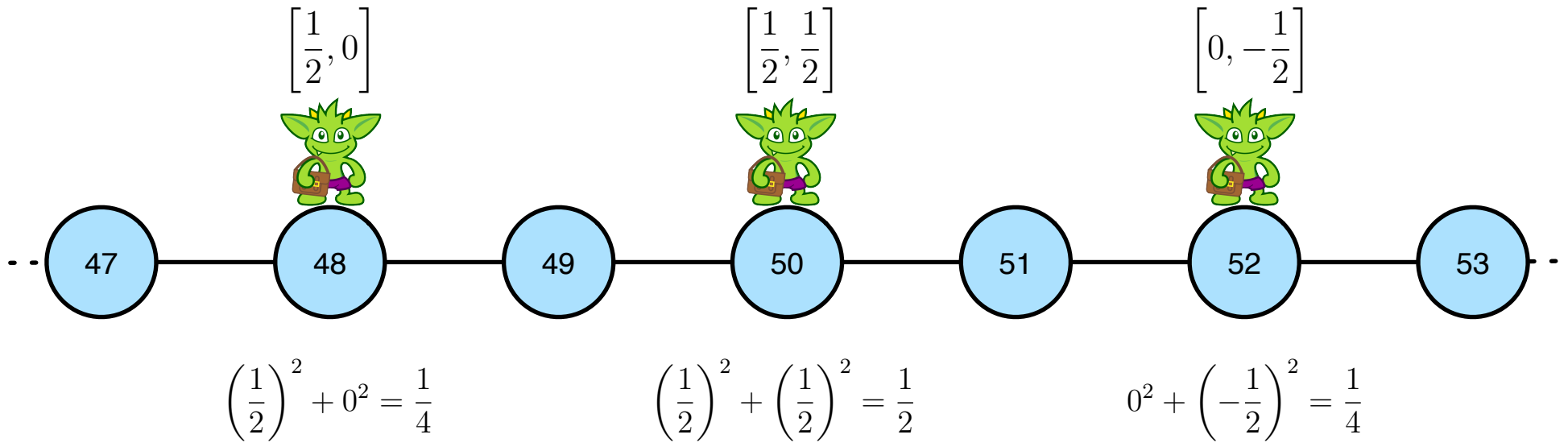
The "particles" spinning left go left and
the "particles" spinning right go right.

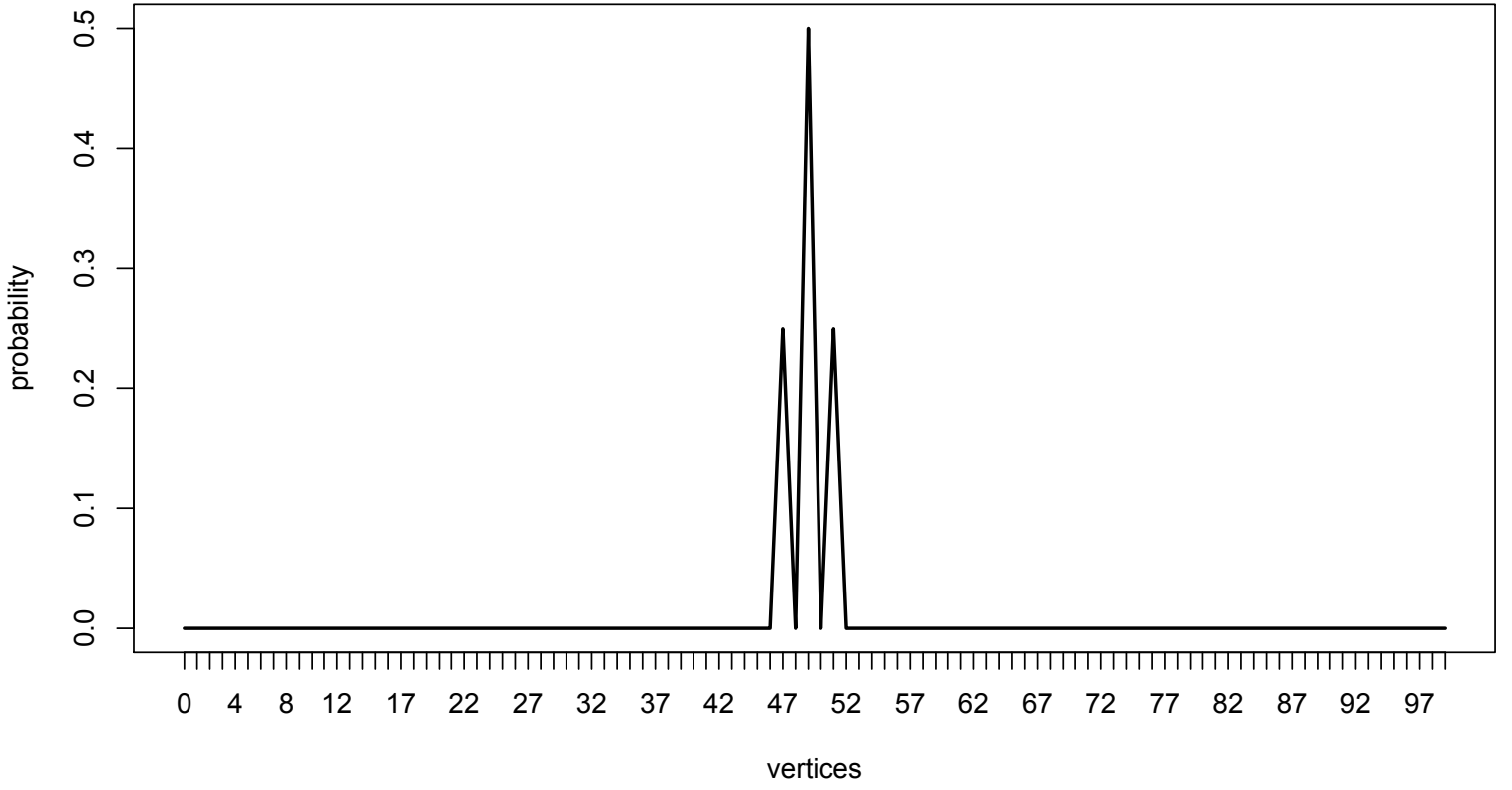
Shift Step



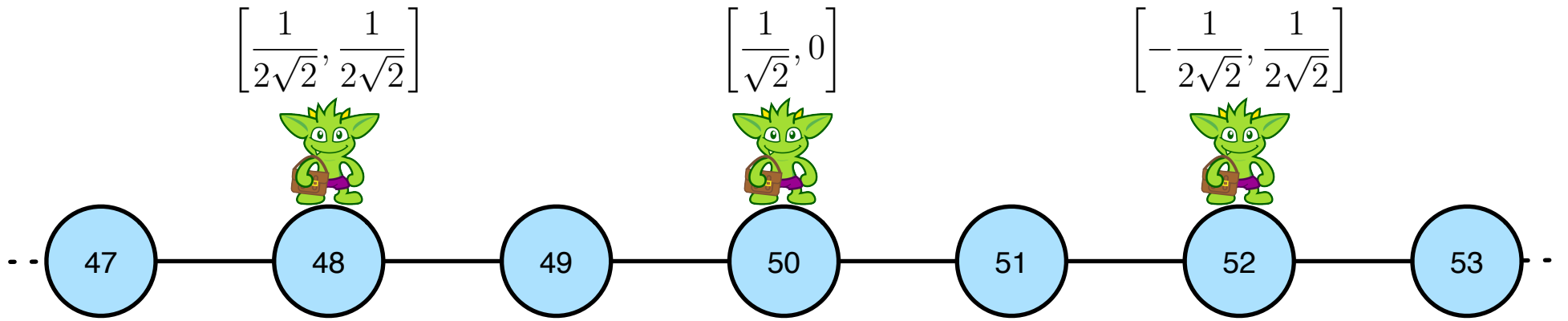
Constructive Interference

Shift Step



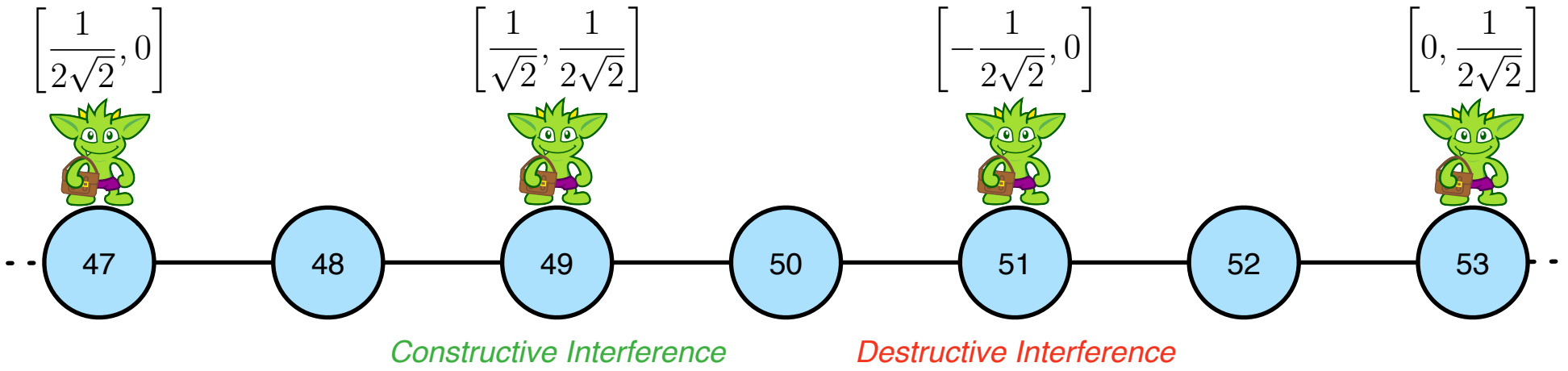


Wave Function to Probability Distribution



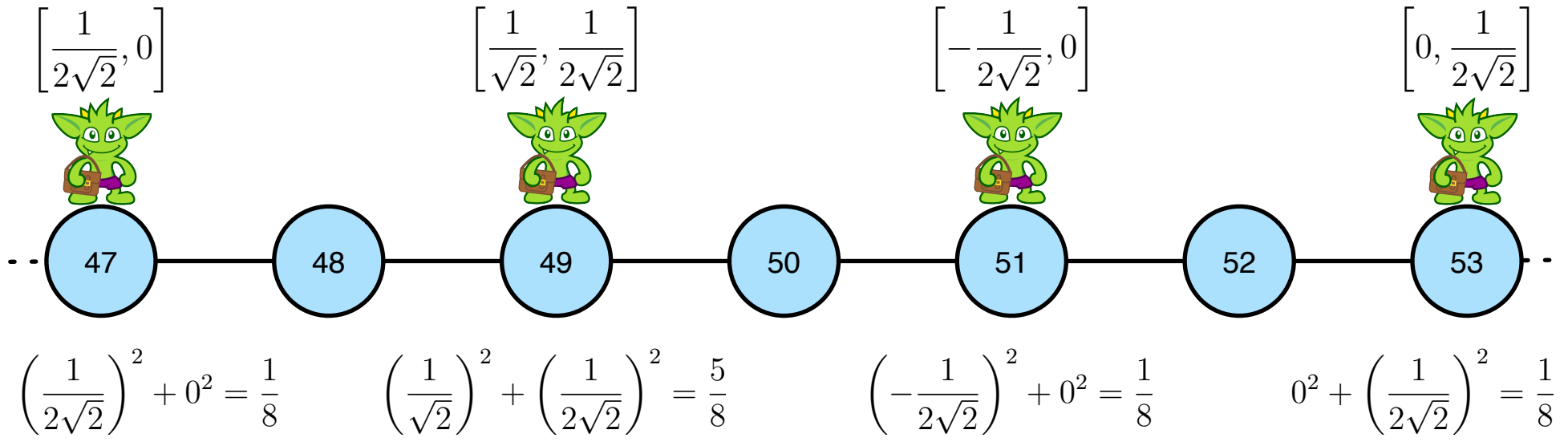
The "particles'" spin is put into superposition.

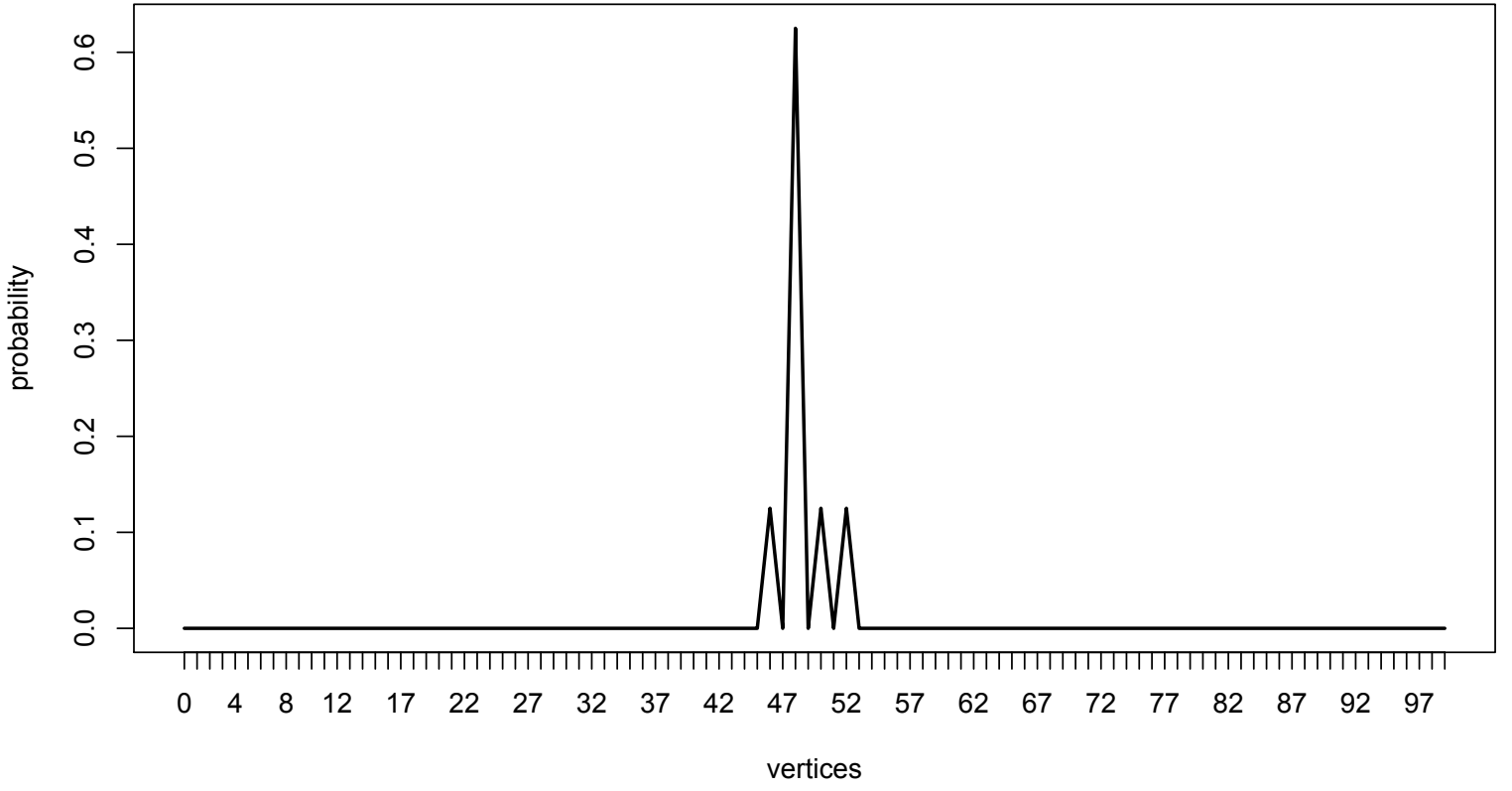
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$



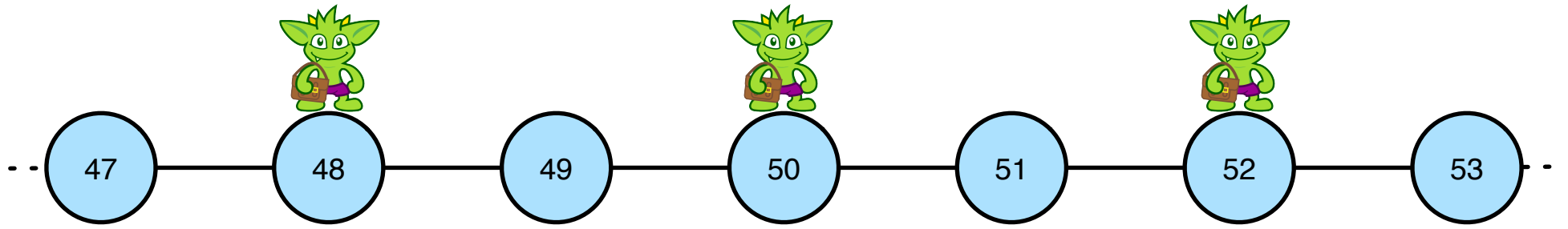
The "particles" spinning left go left and the "particles" spinning right go right.

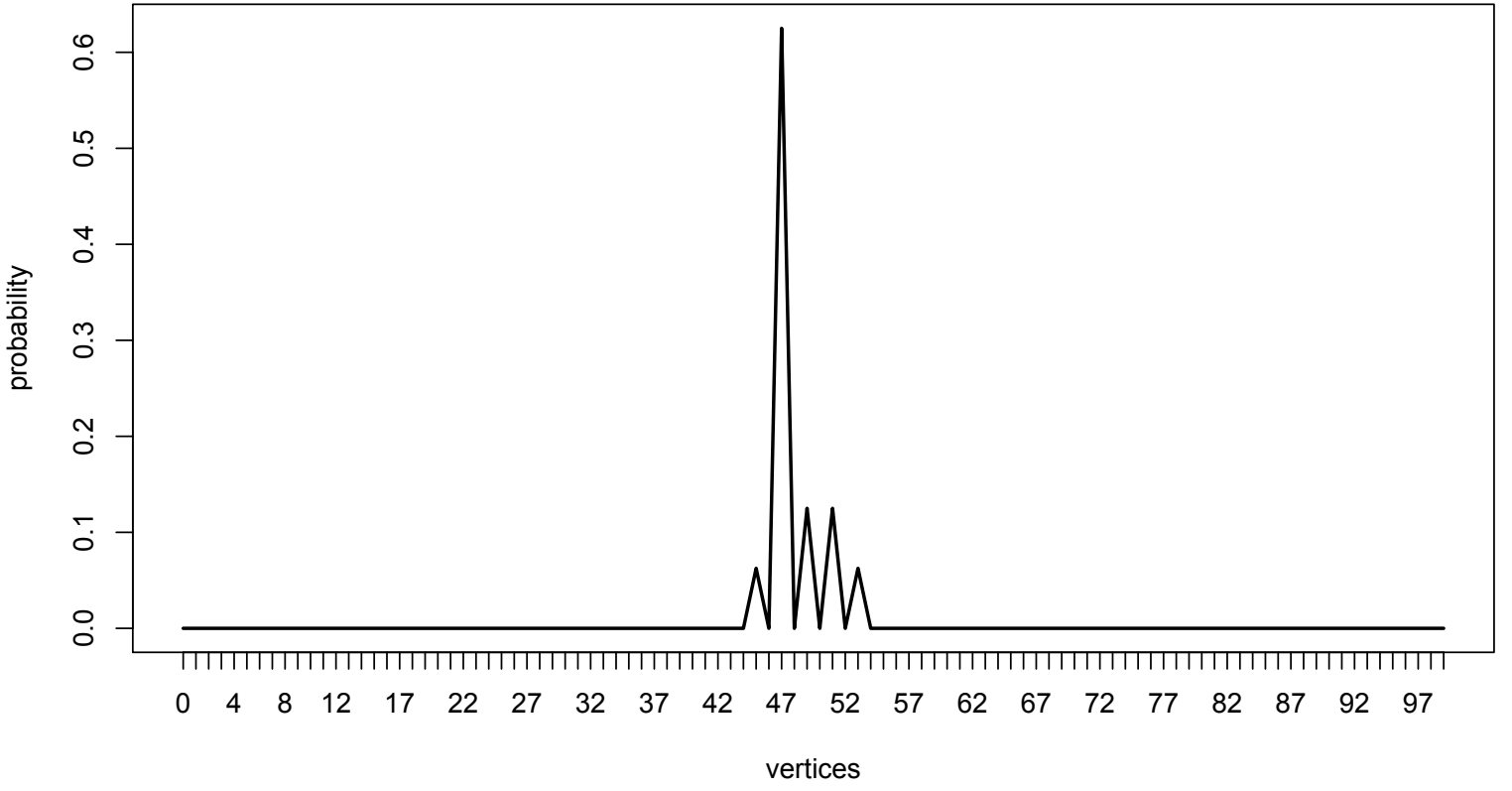
Shift Step



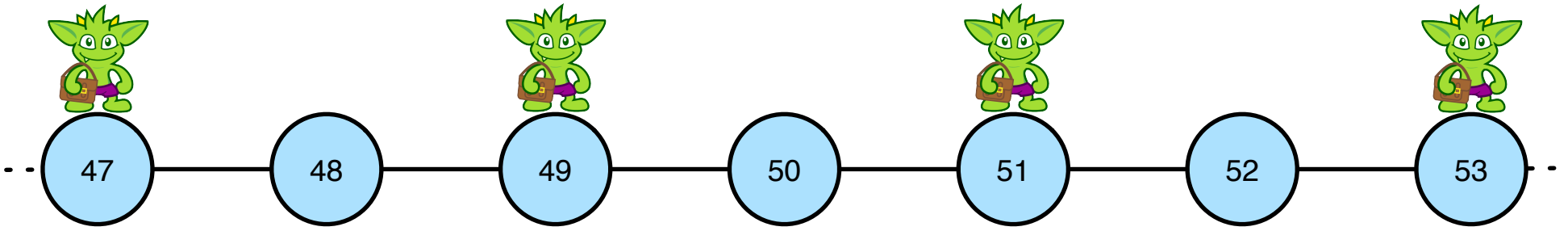


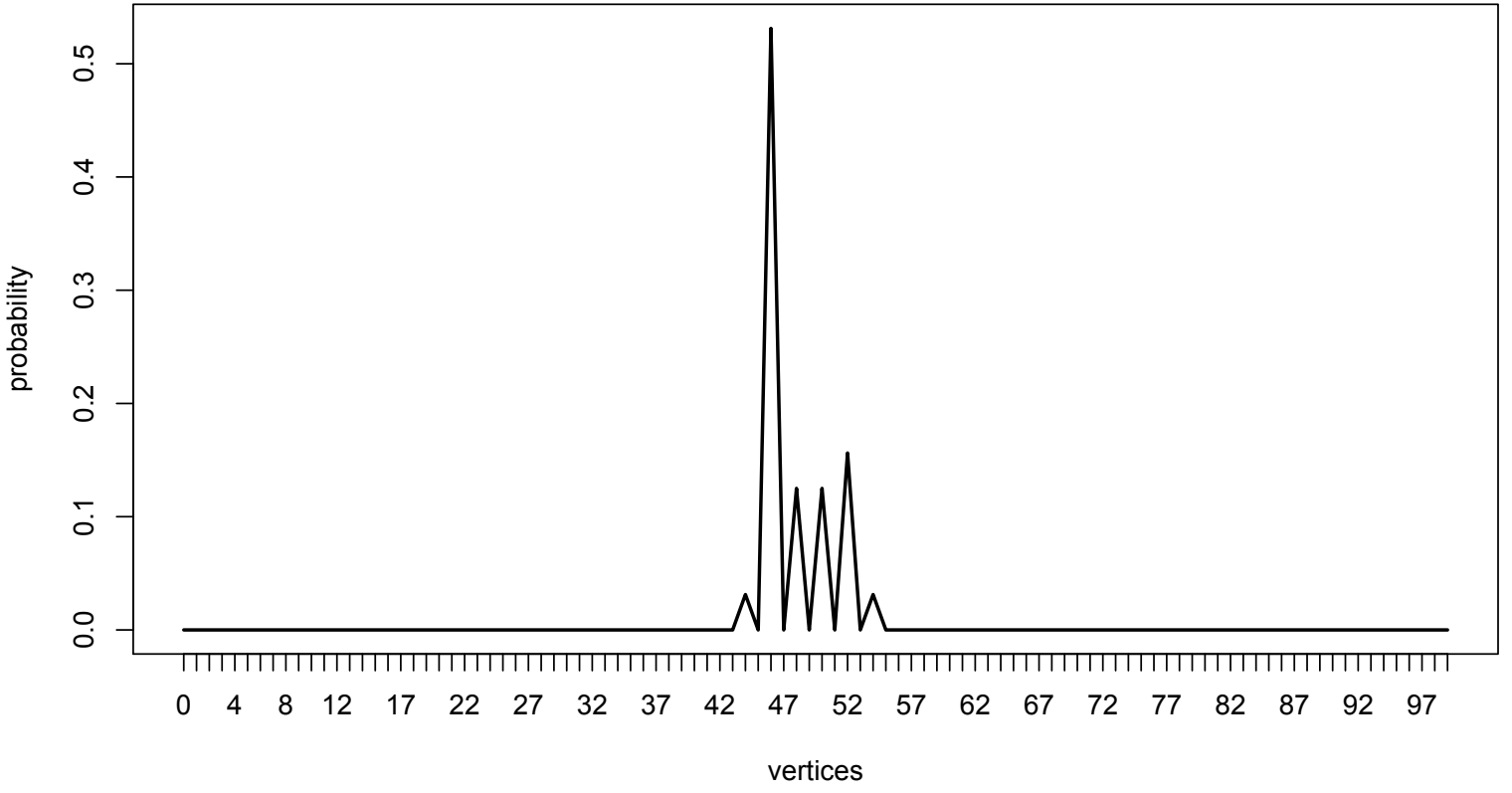
Wave Function to Probability Distribution



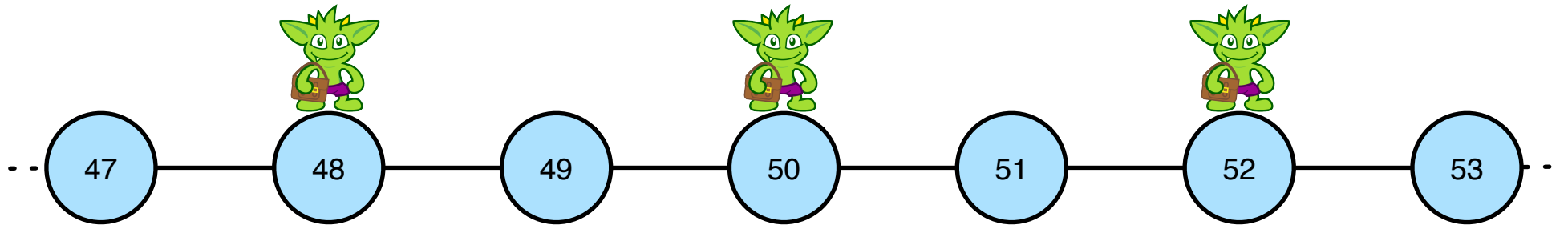


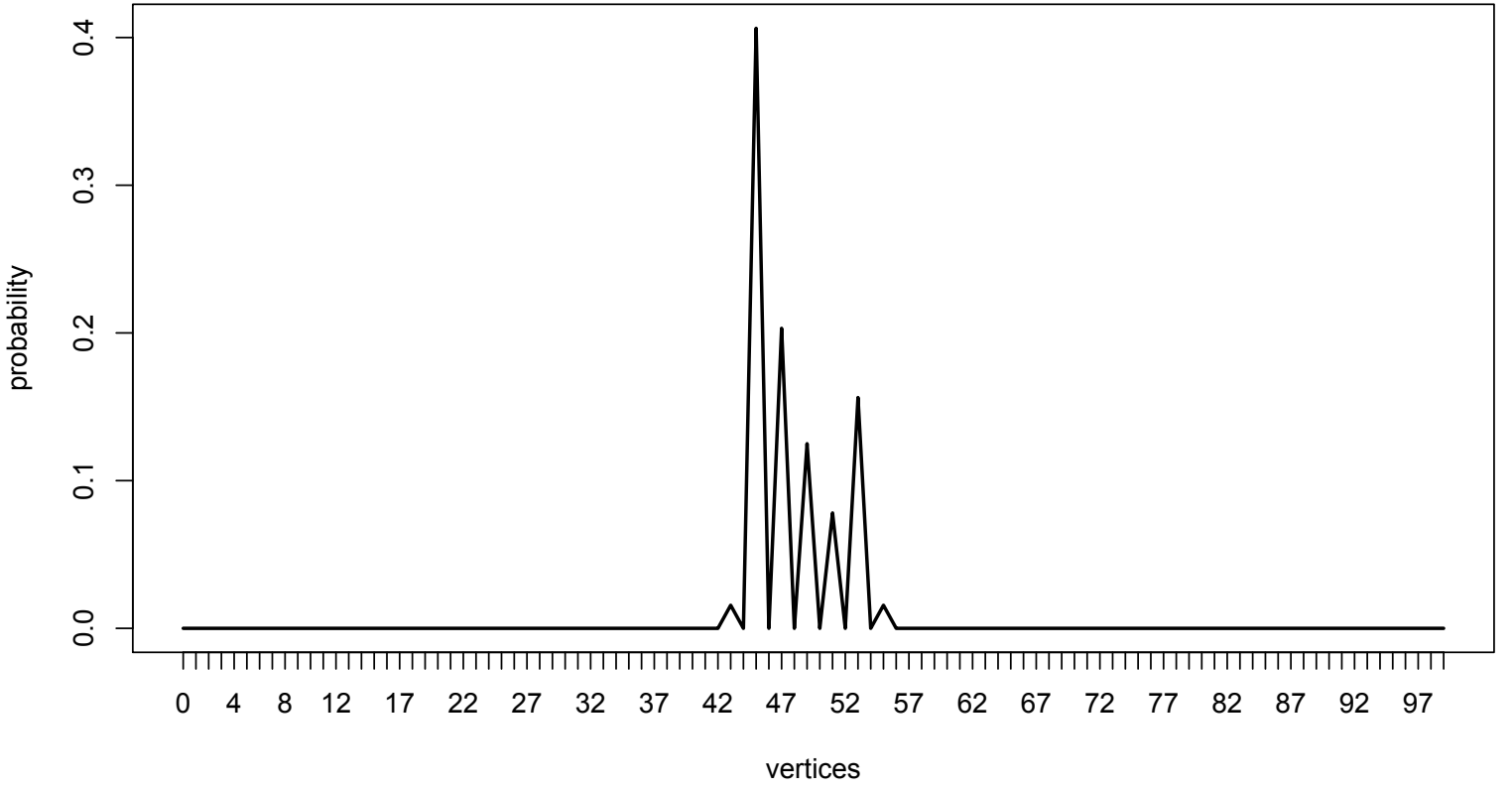
Wave Function to Probability Distribution



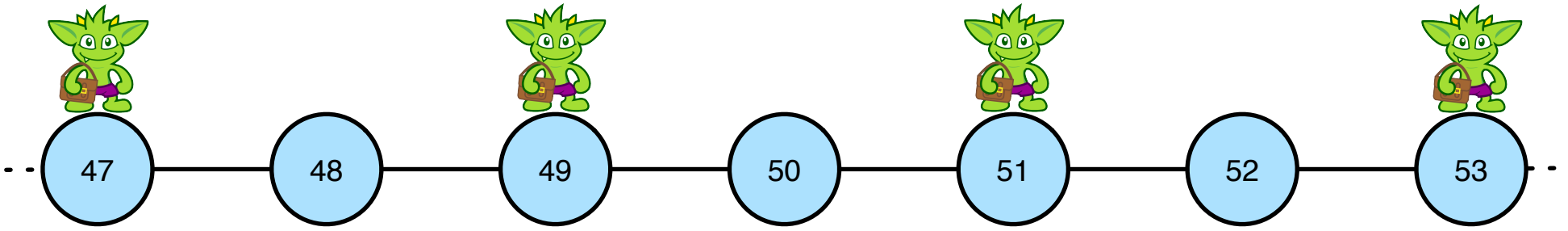


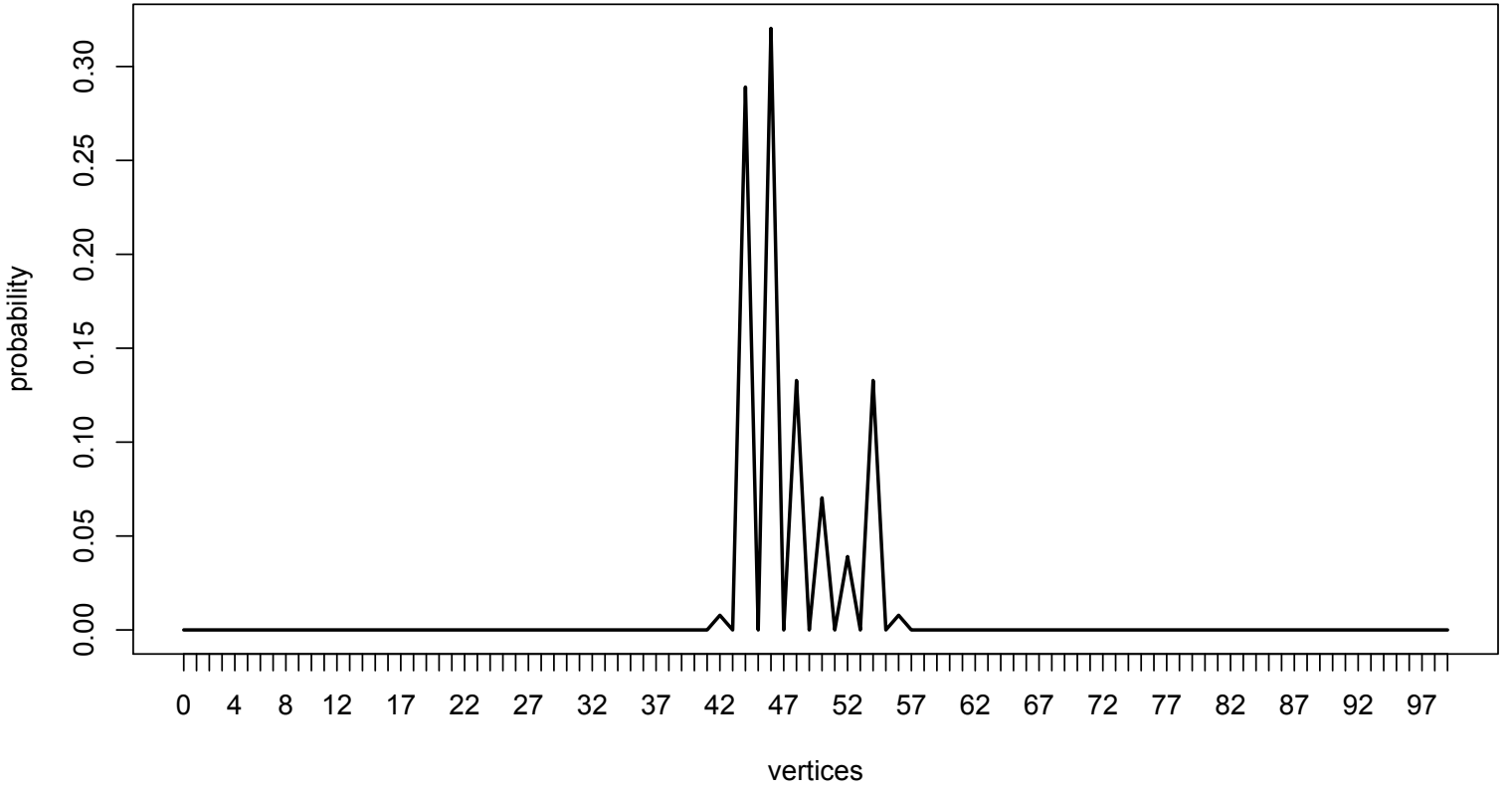
Wave Function to Probability Distribution



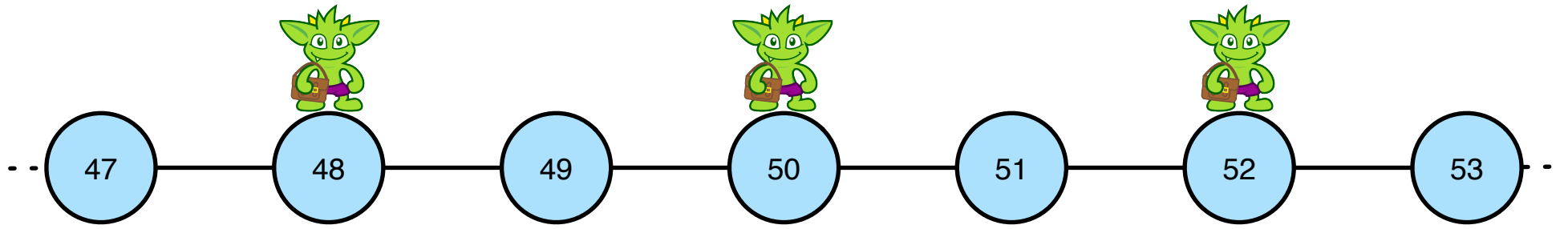


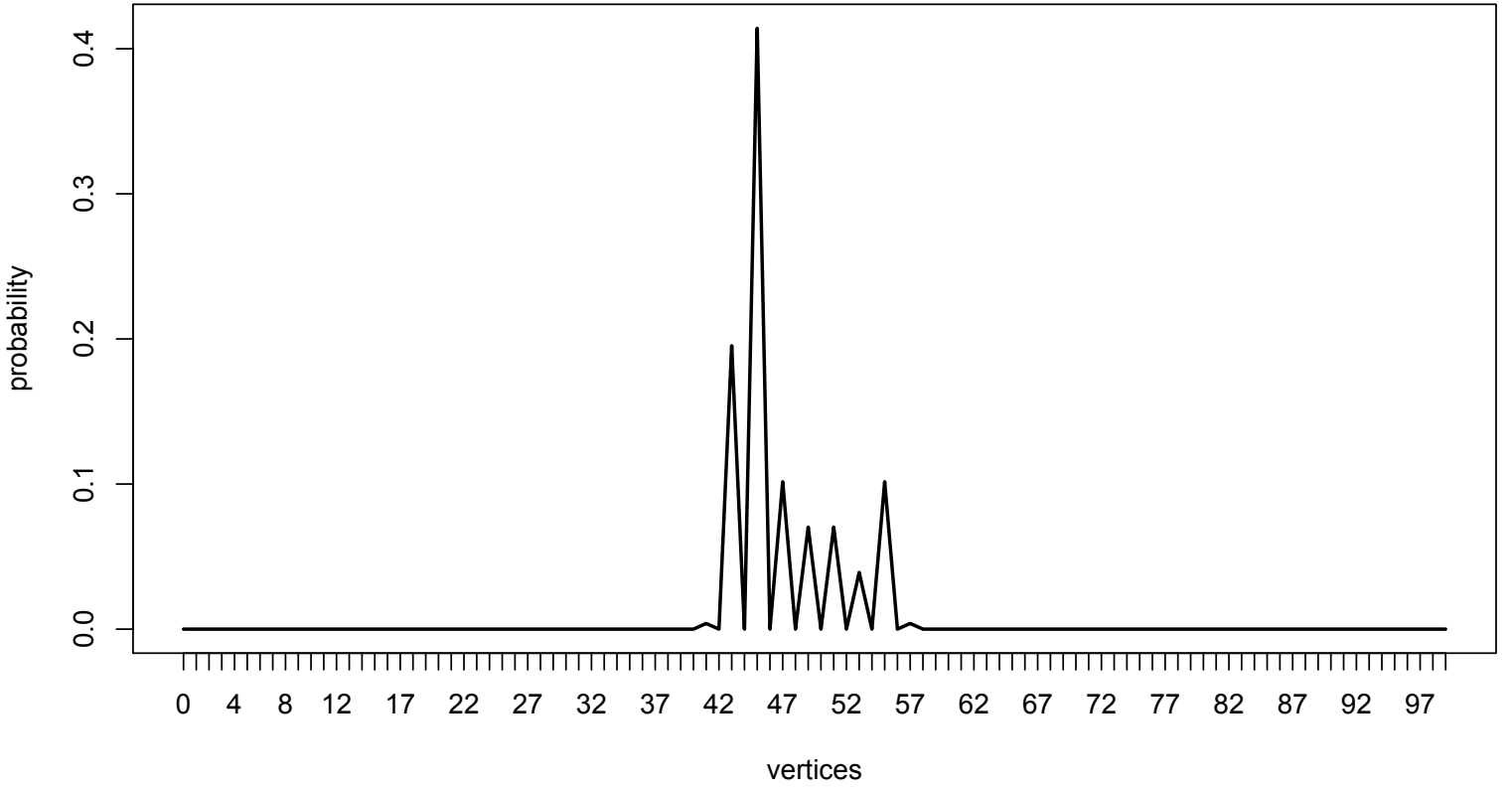
Wave Function to Probability Distribution



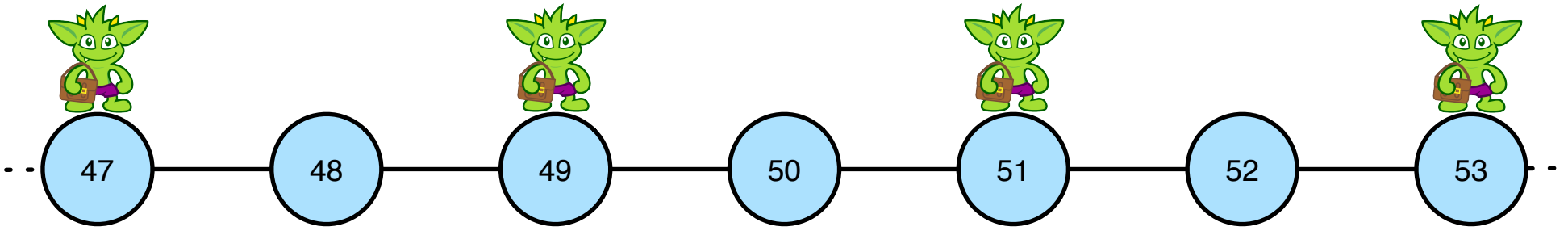


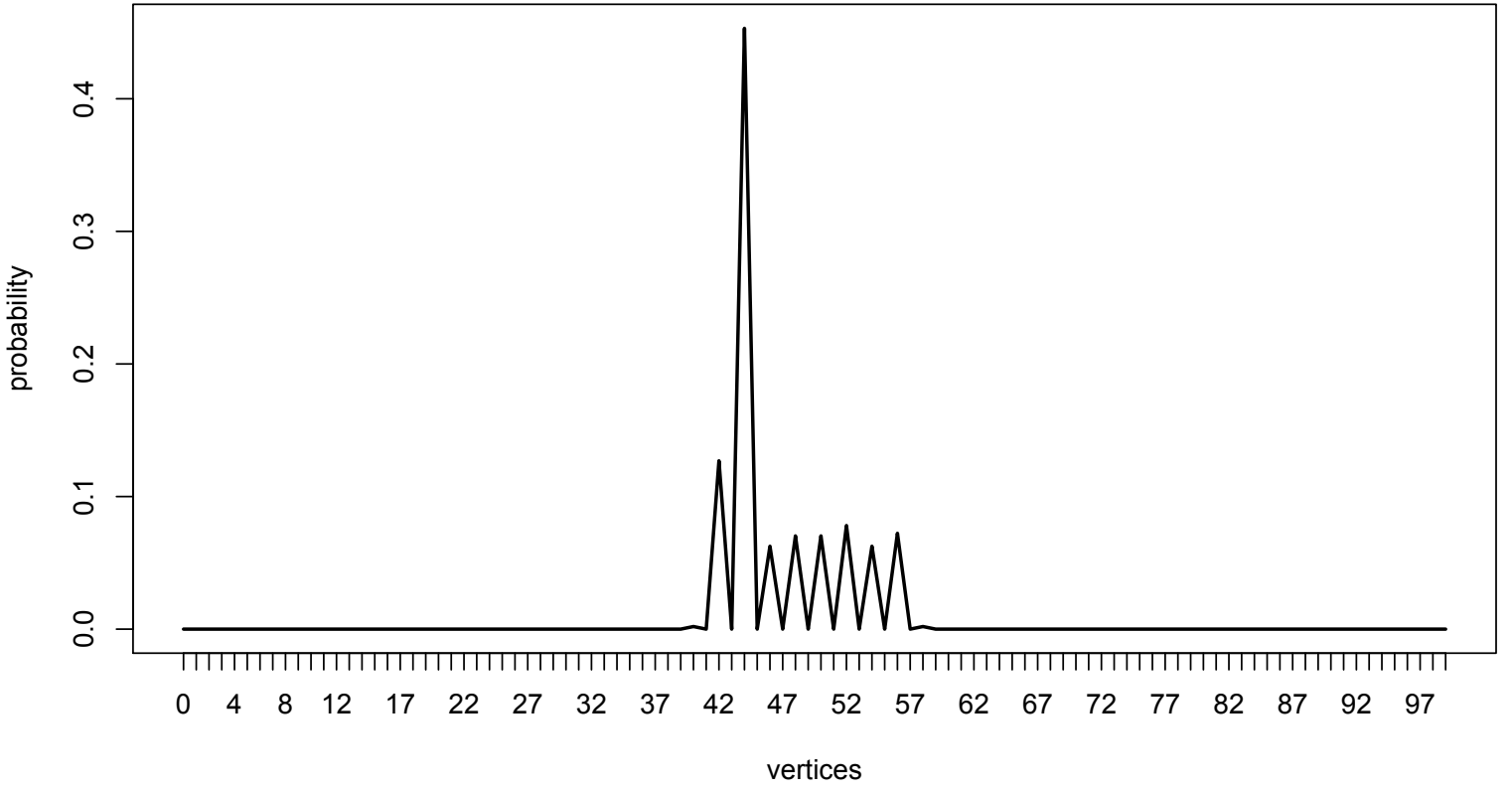
Wave Function to Probability Distribution



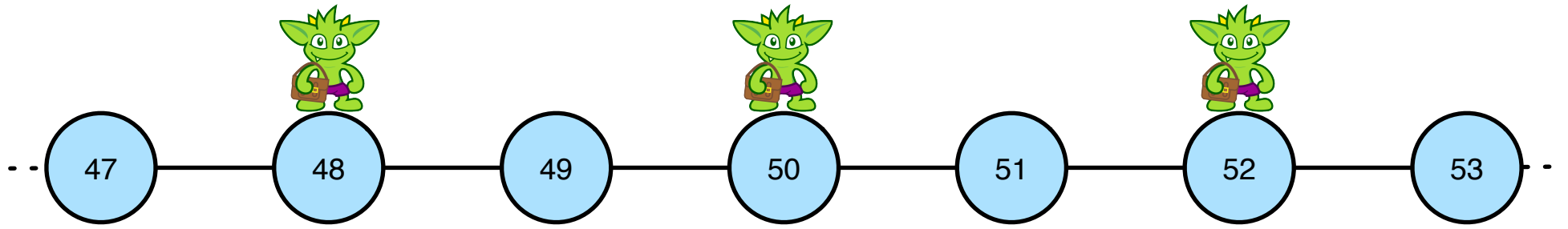


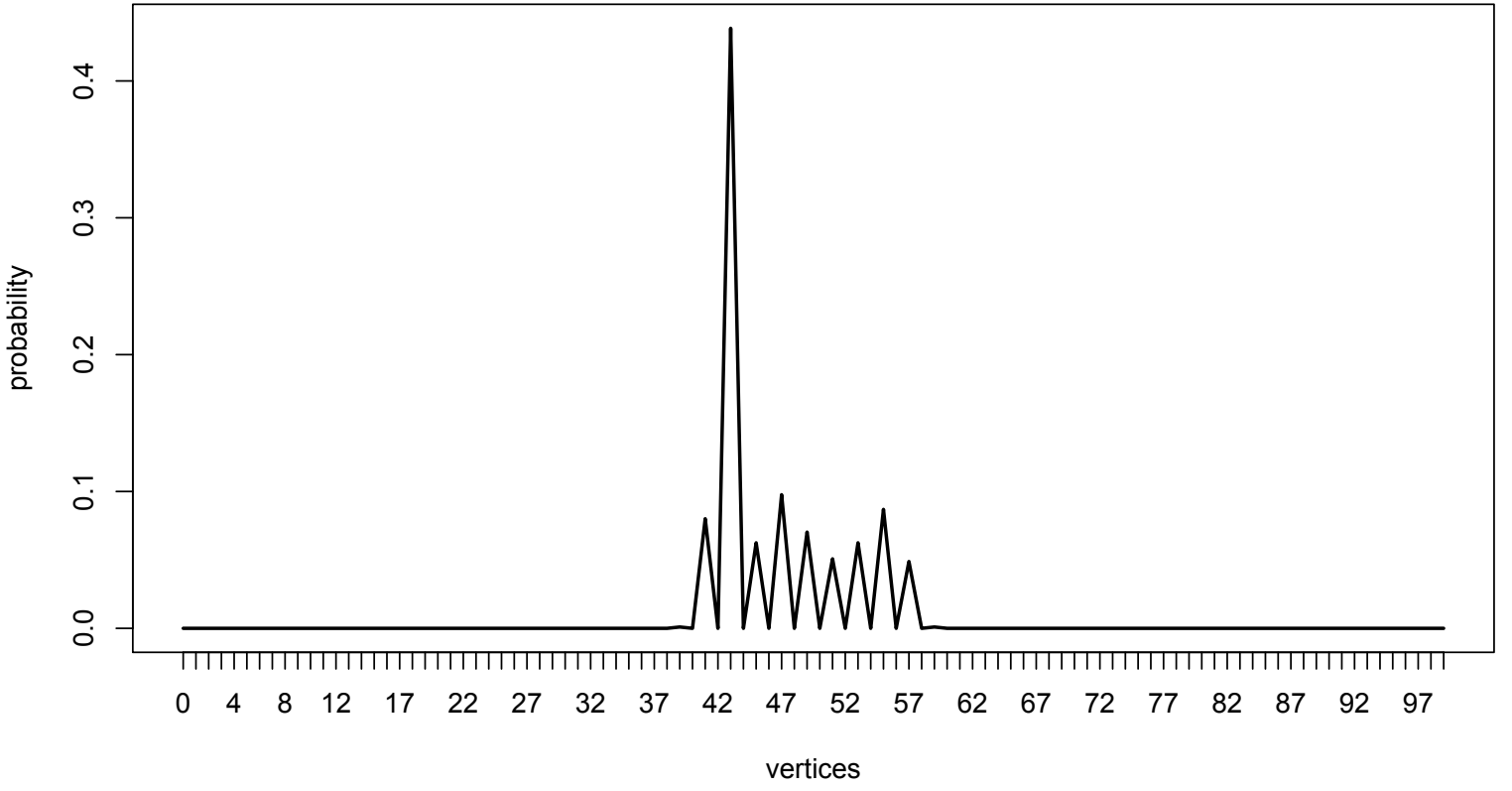
Wave Function to Probability Distribution



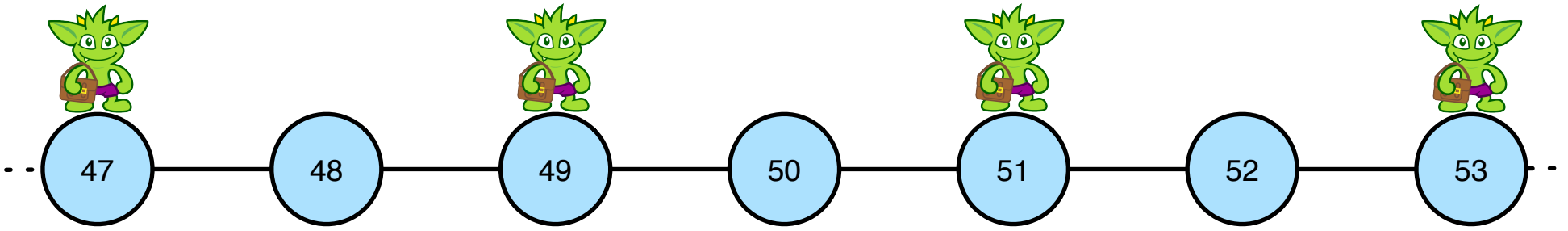


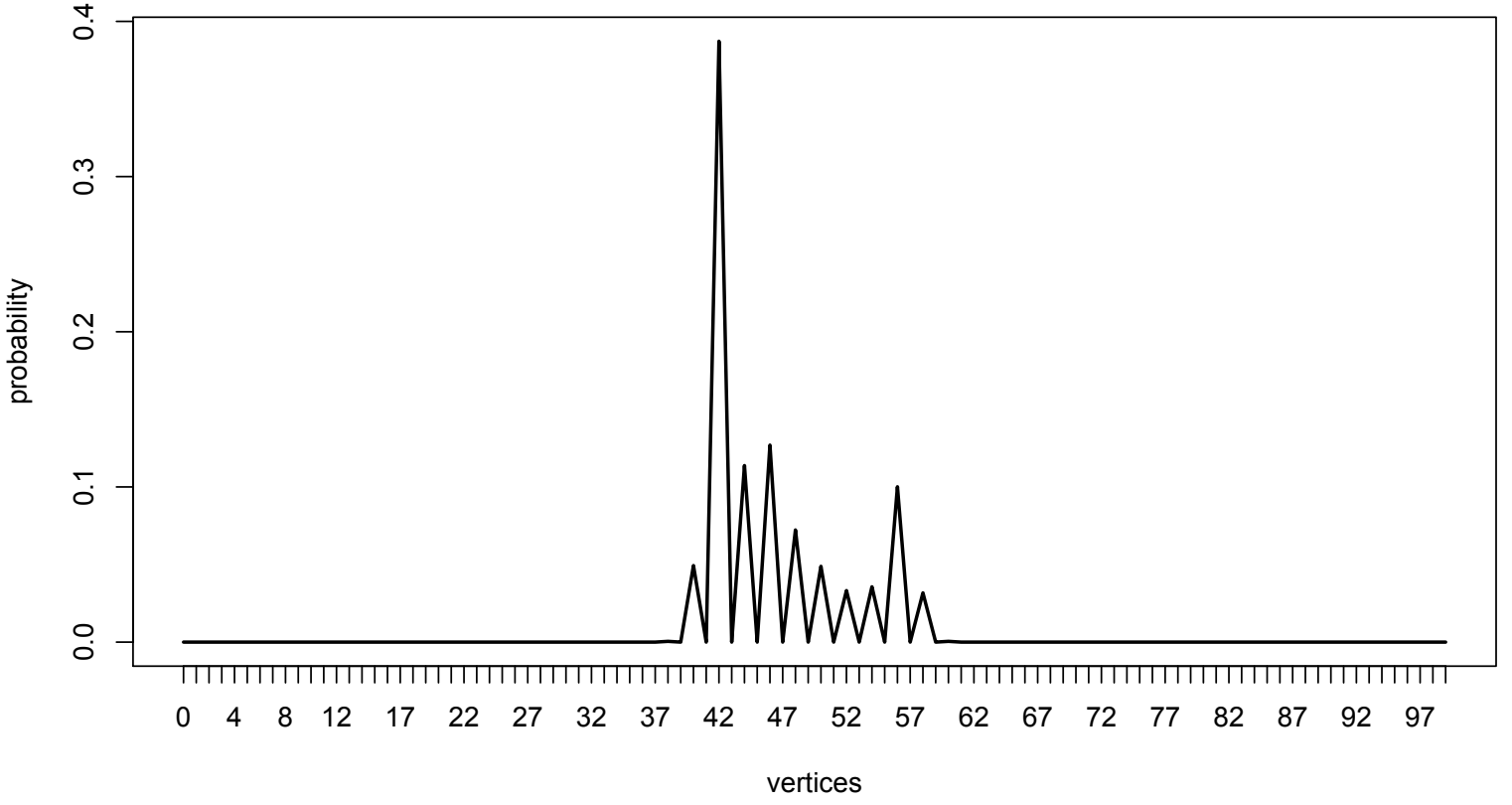
Wave Function to Probability Distribution



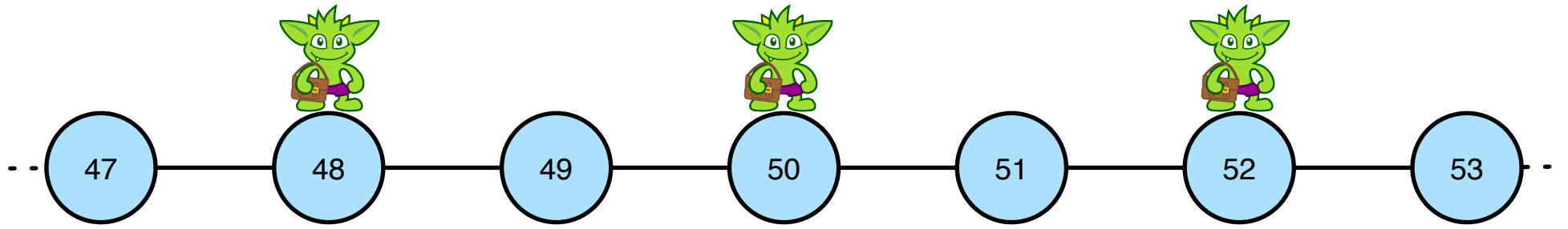


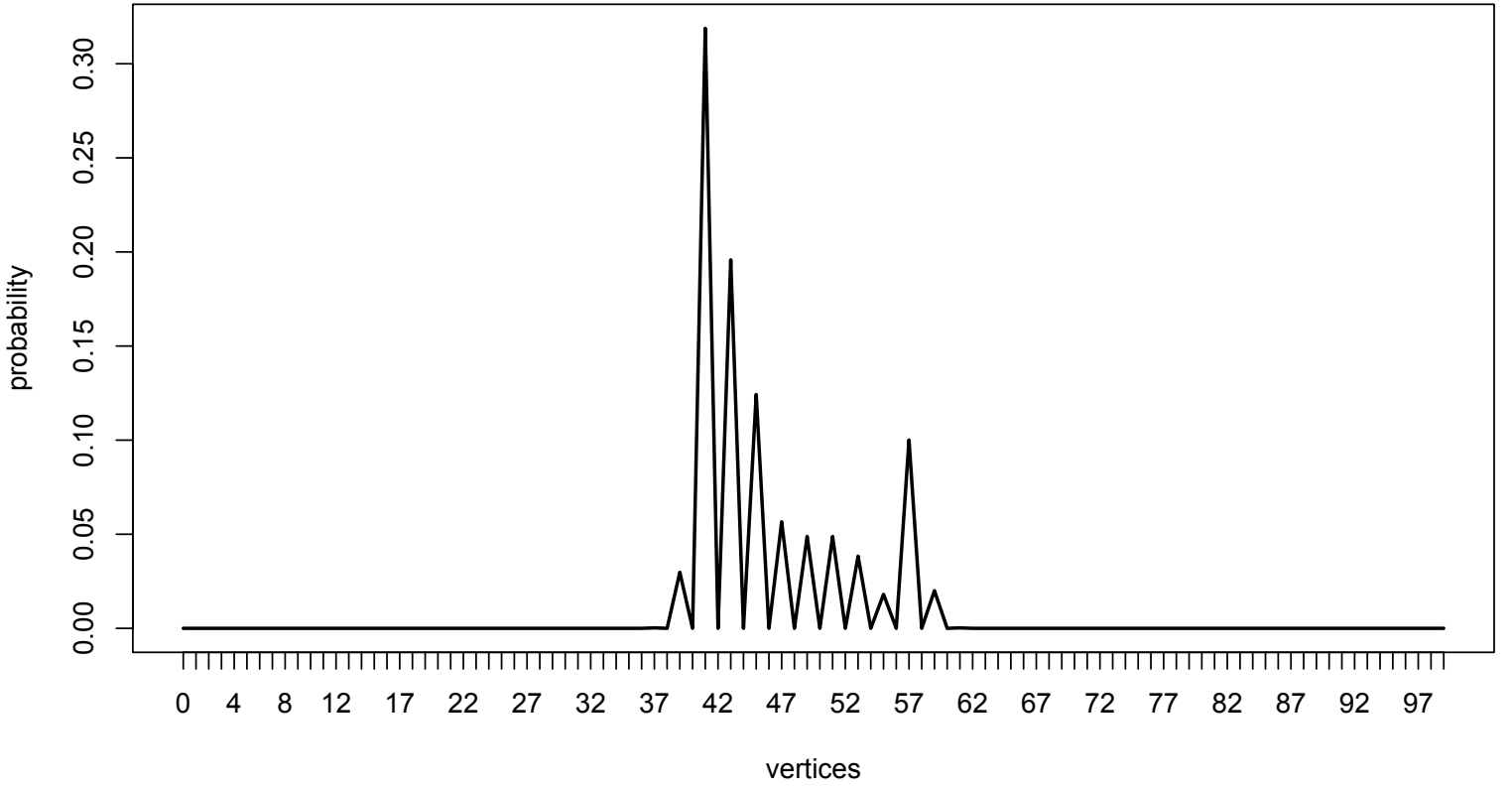
Wave Function to Probability Distribution



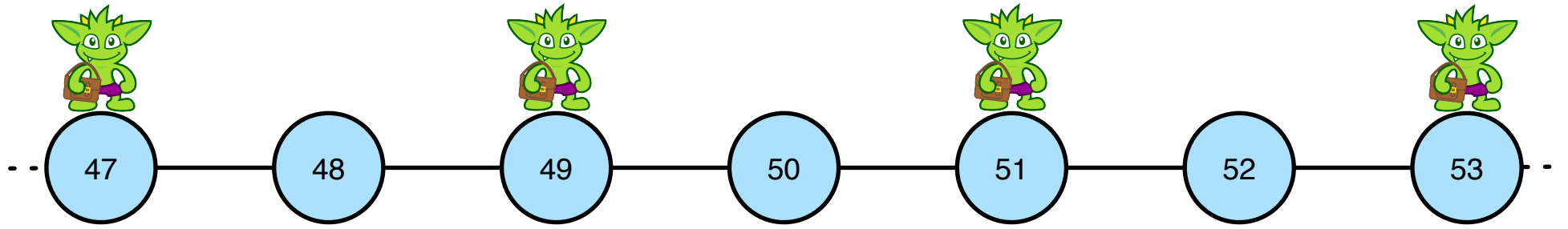


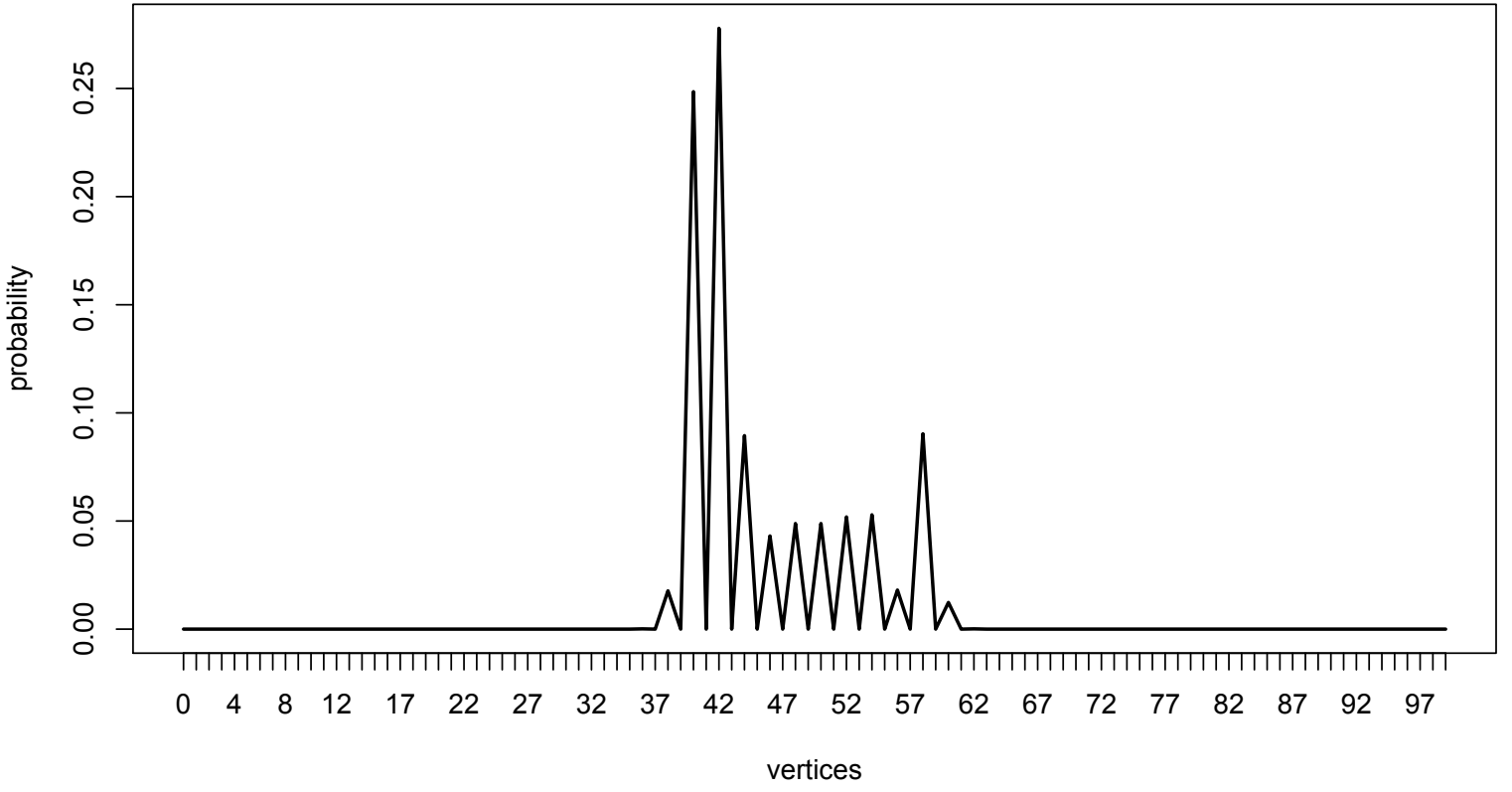
Wave Function to Probability Distribution



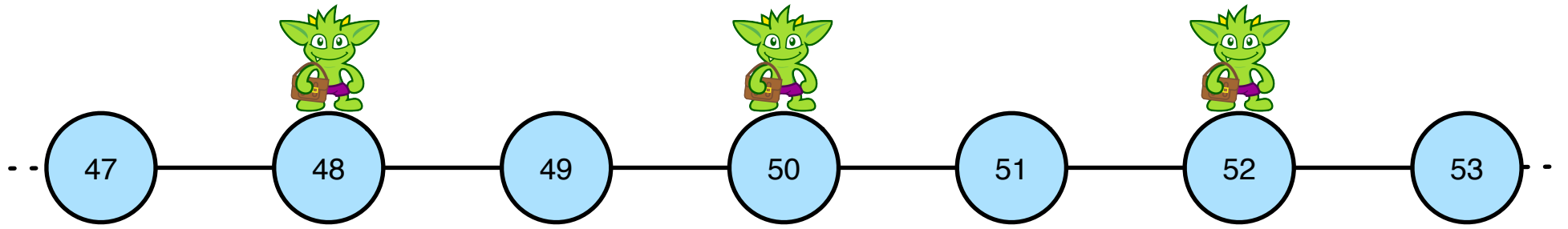


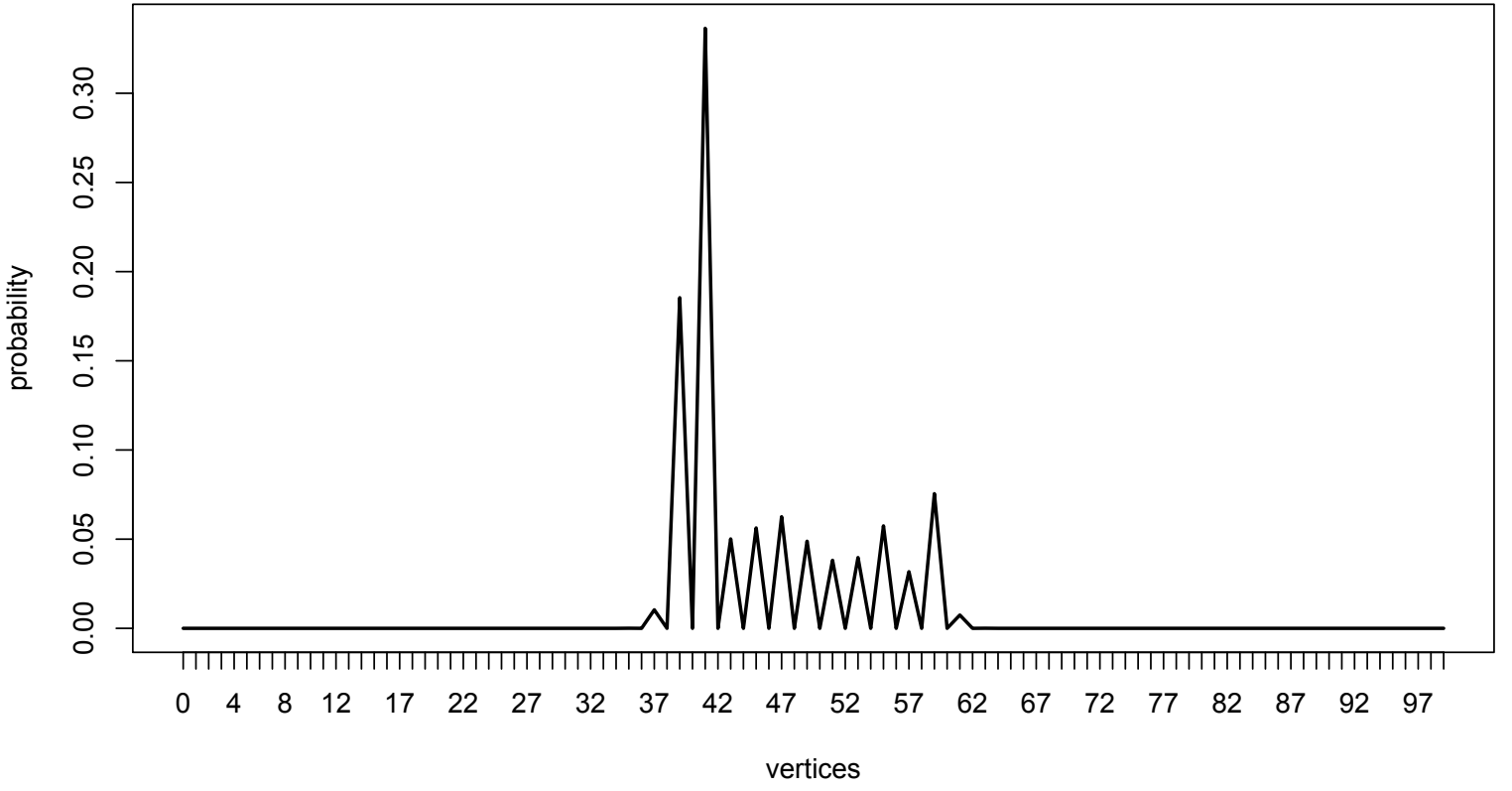
Wave Function to Probability Distribution



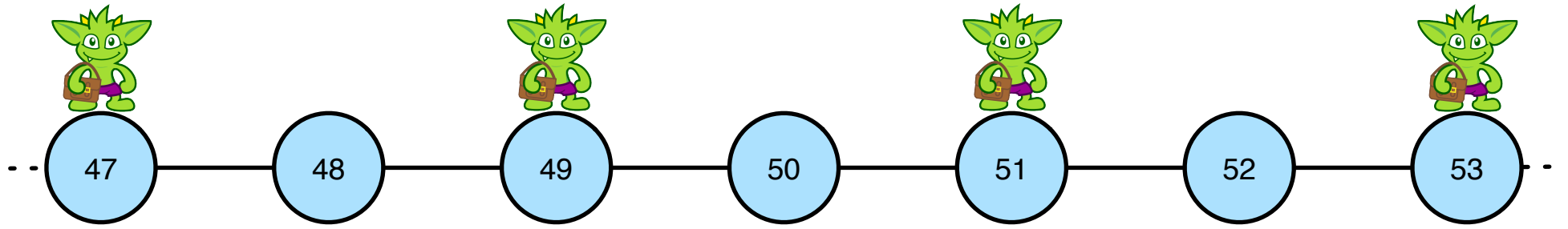


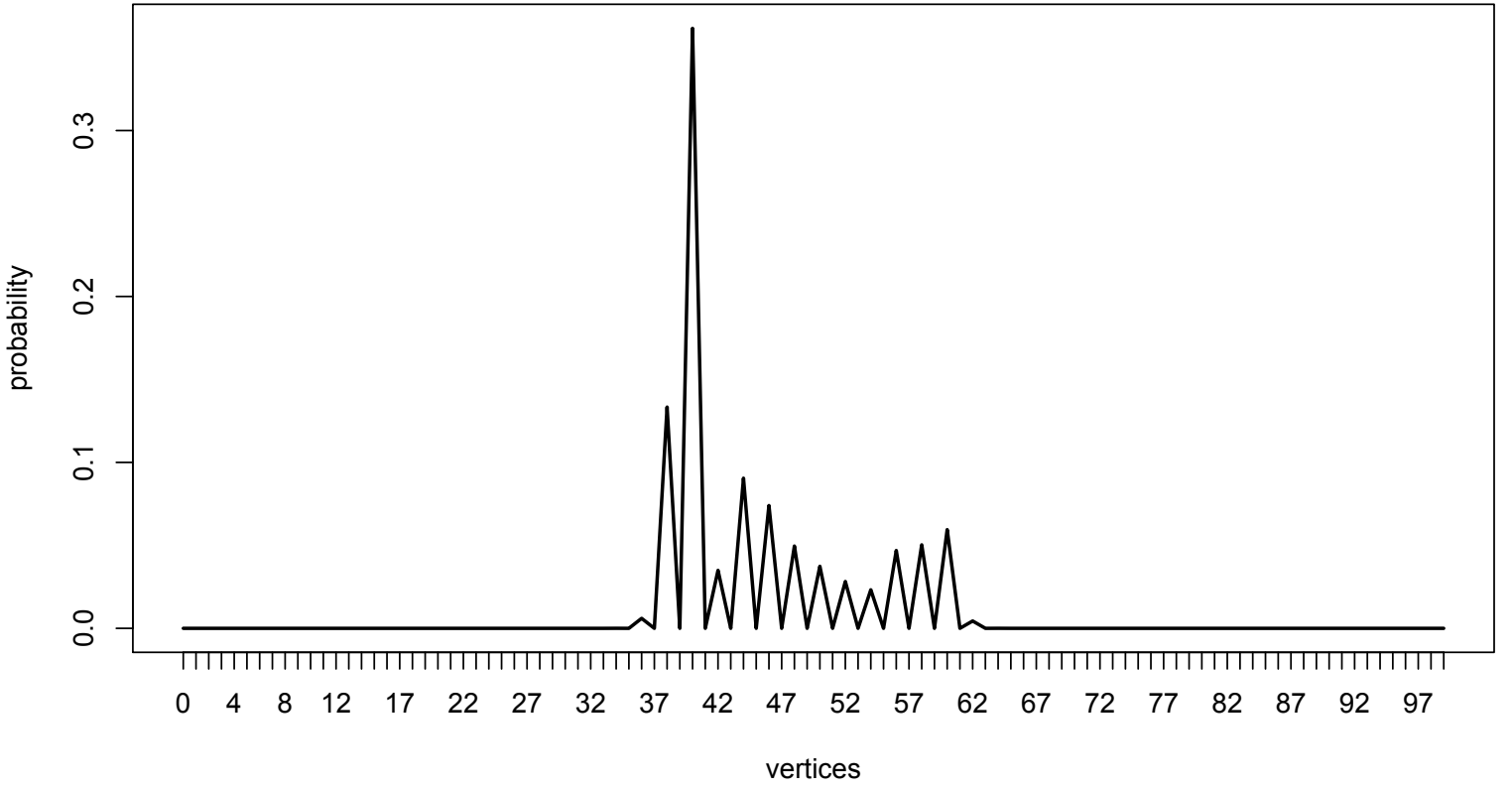
Wave Function to Probability Distribution





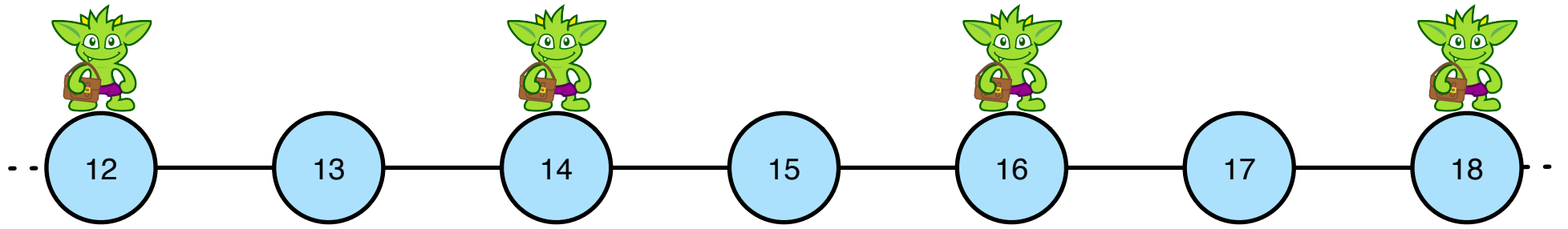
Wave Function to Probability Distribution

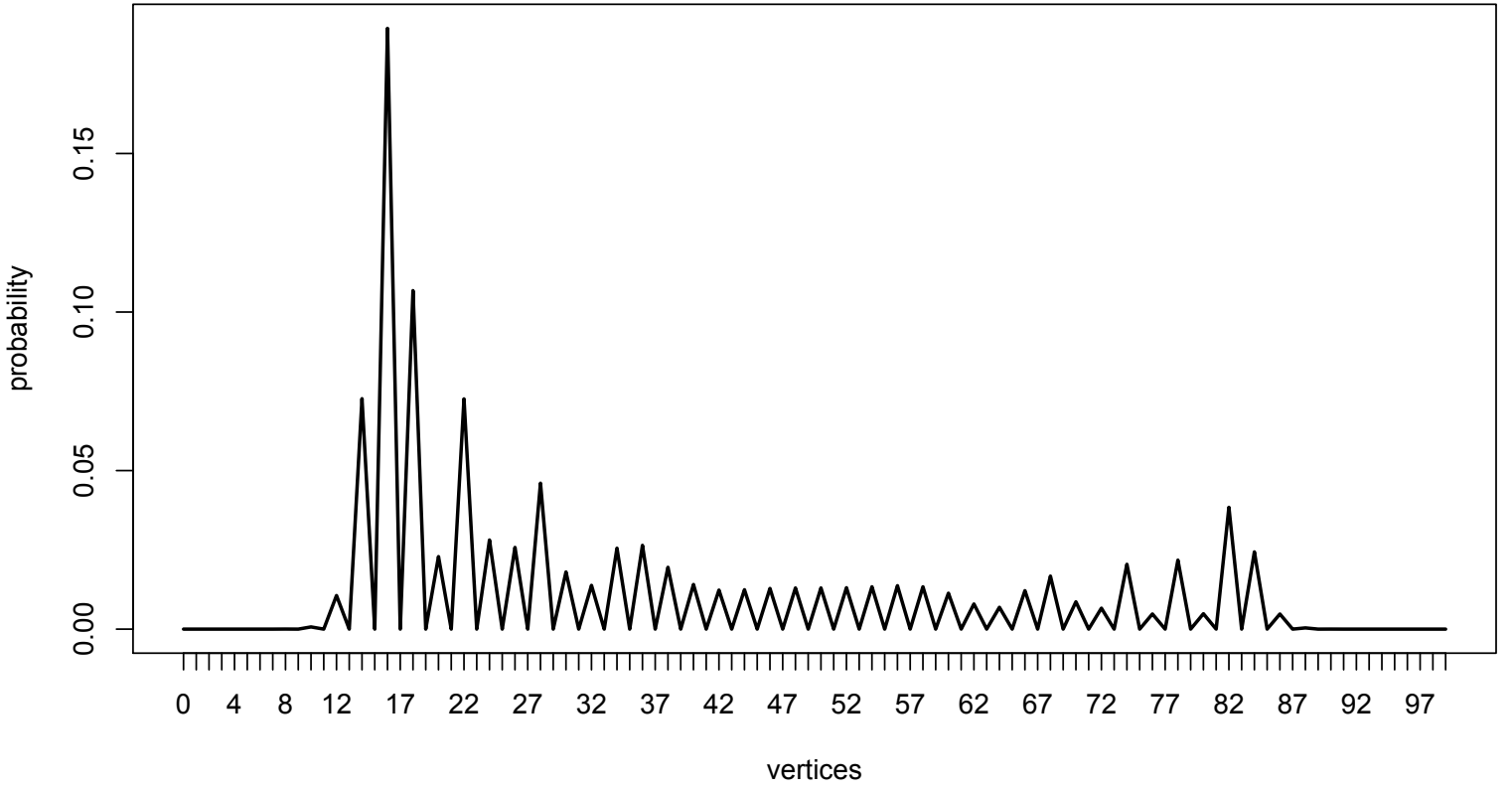




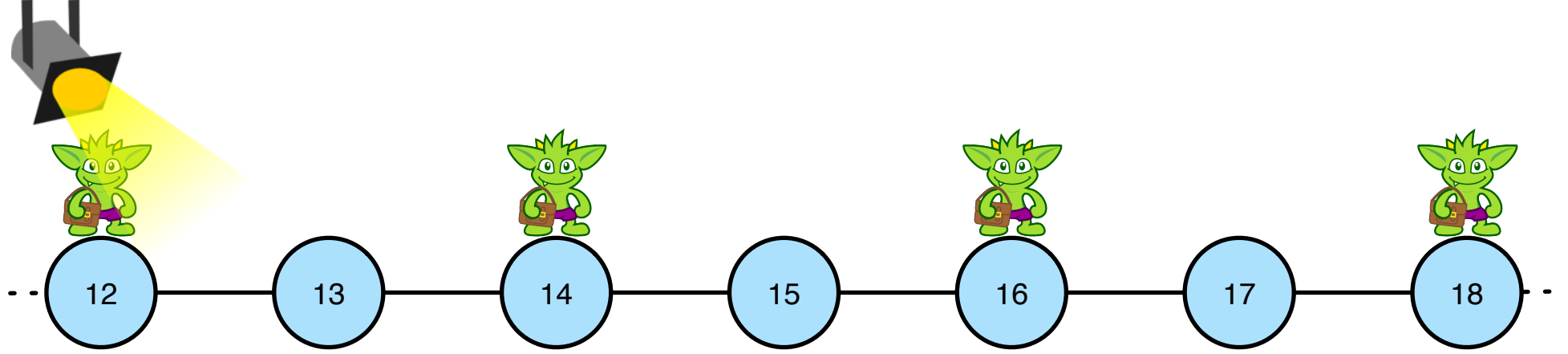
Wave Function to Probability Distribution

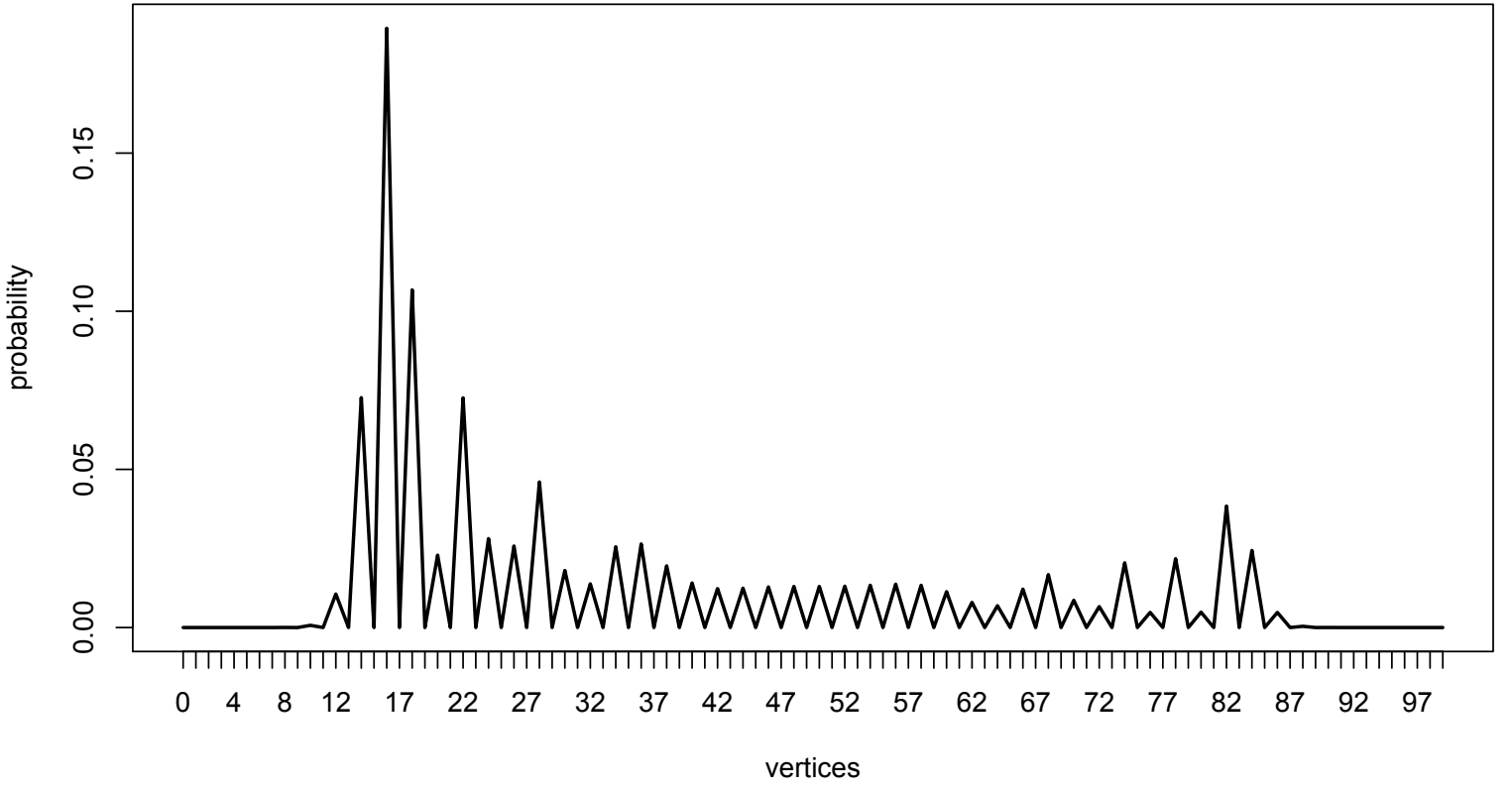
Fast forward to iteration 50.



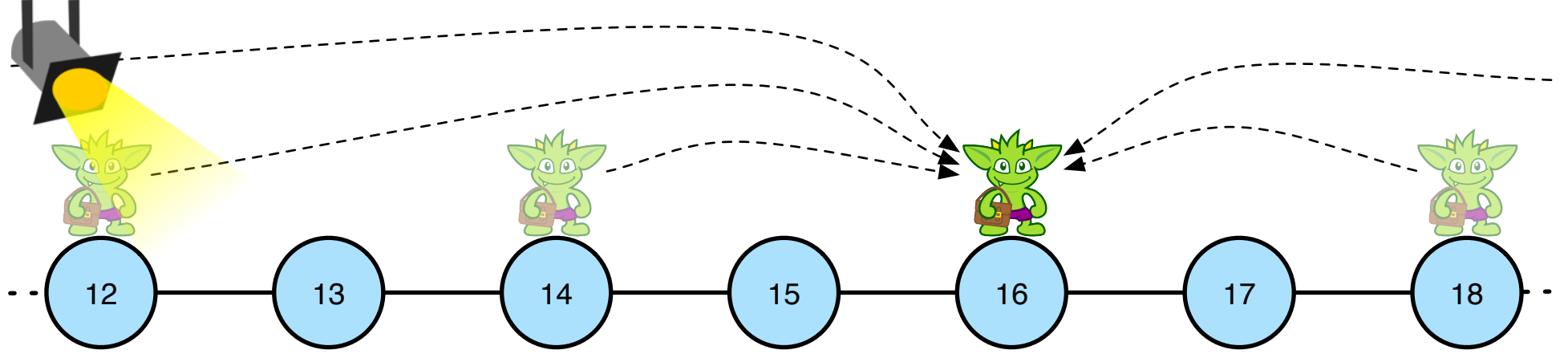


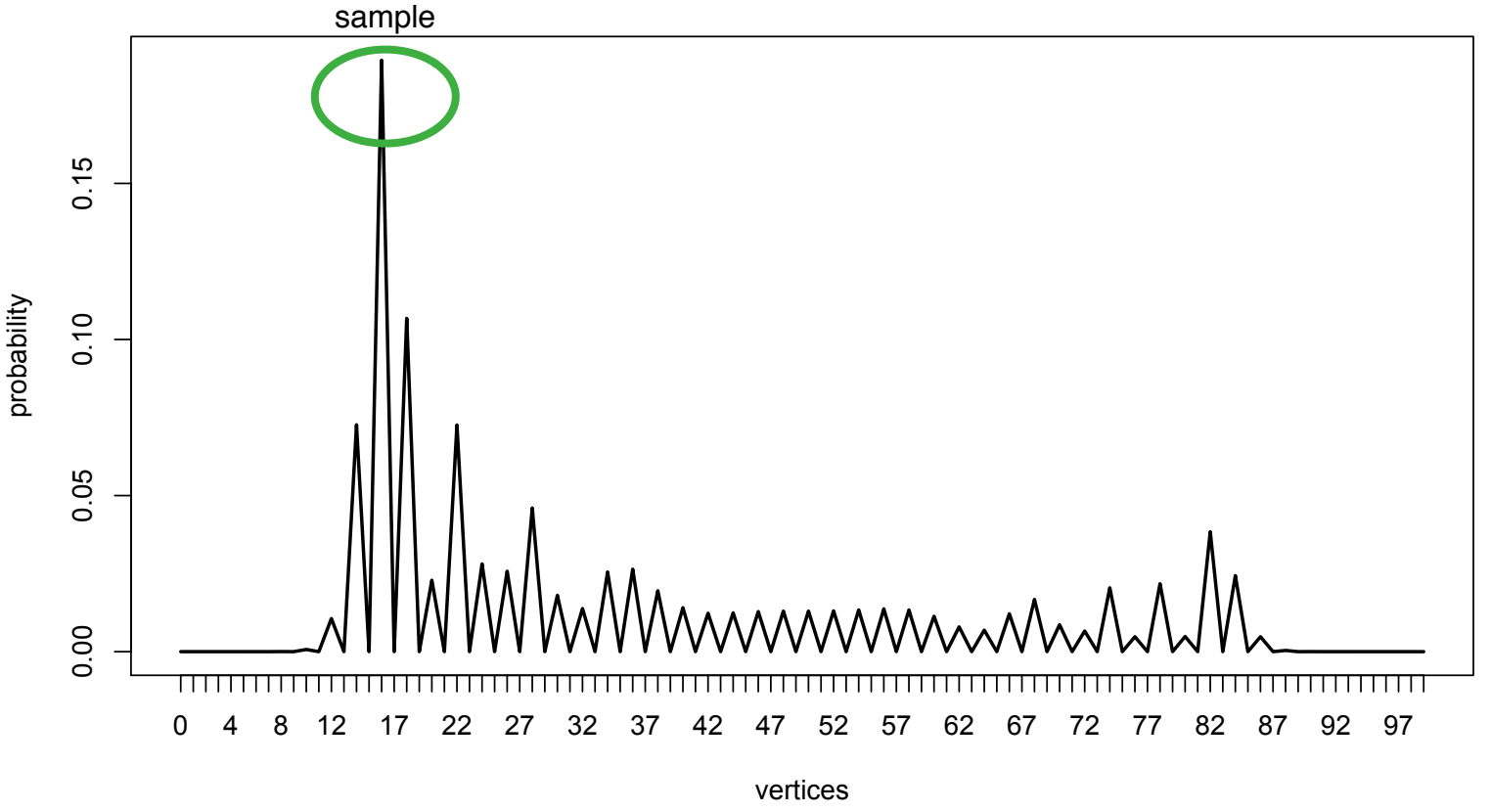
Wave Function to Probability Distribution



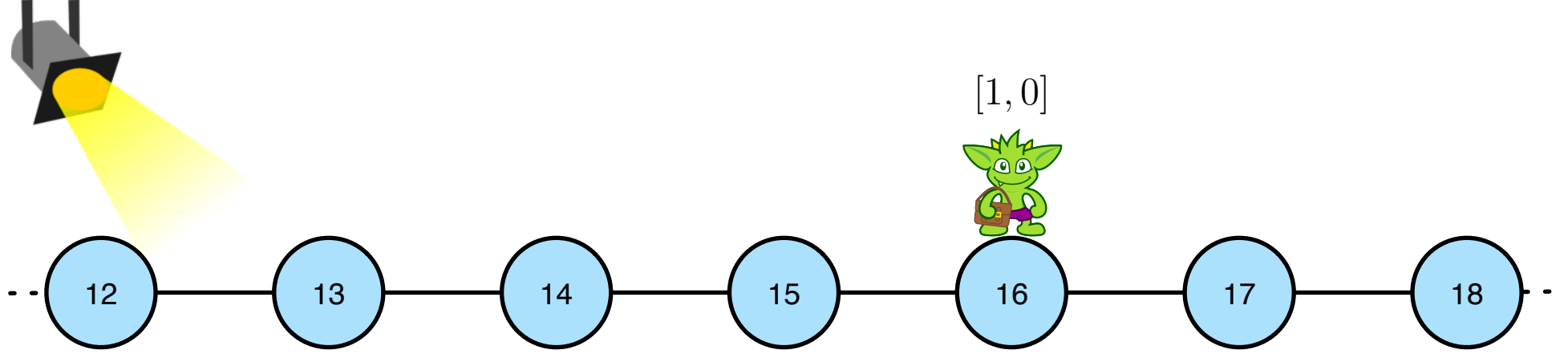


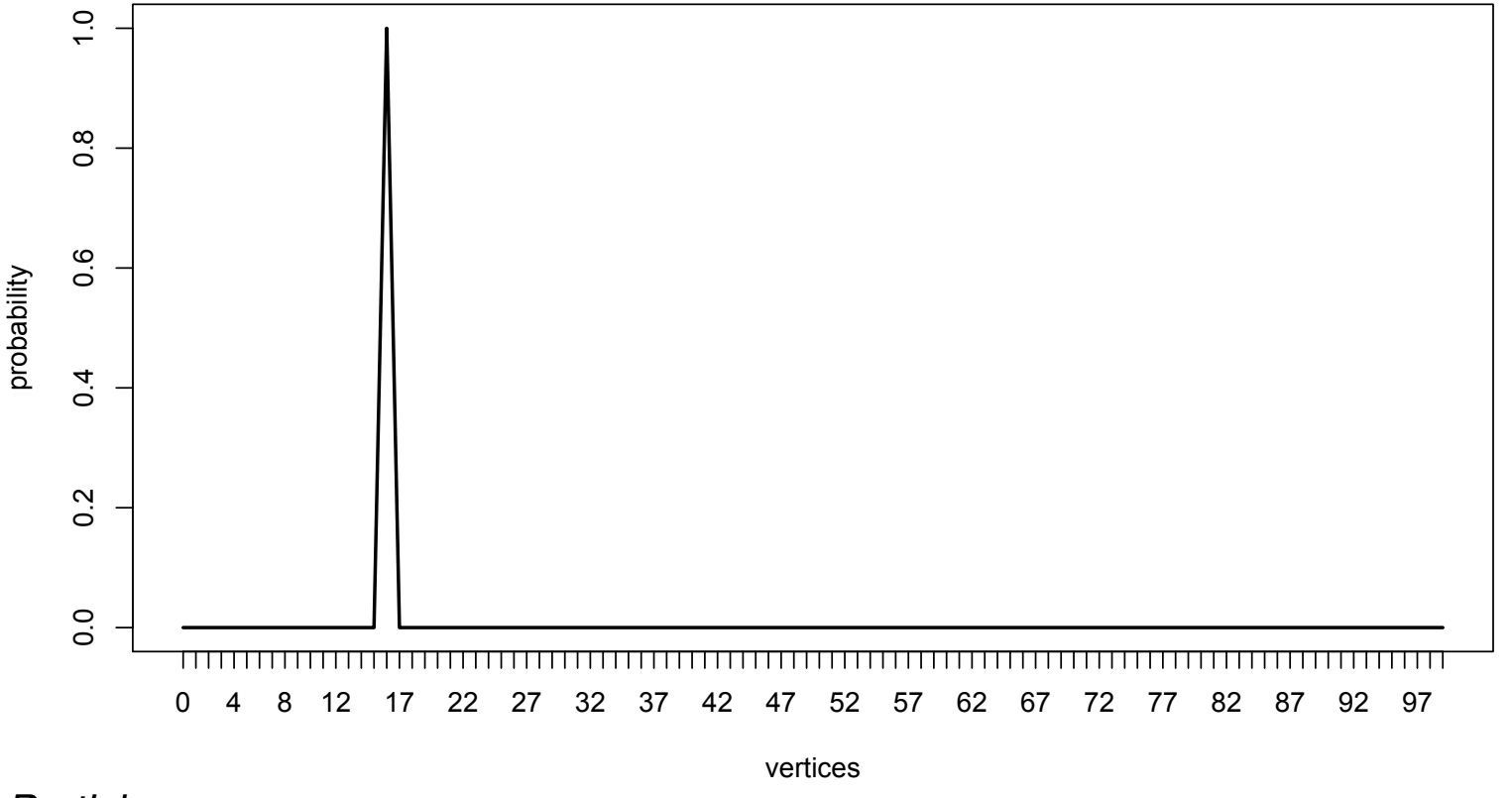
Wave Function is Perturbed by Light



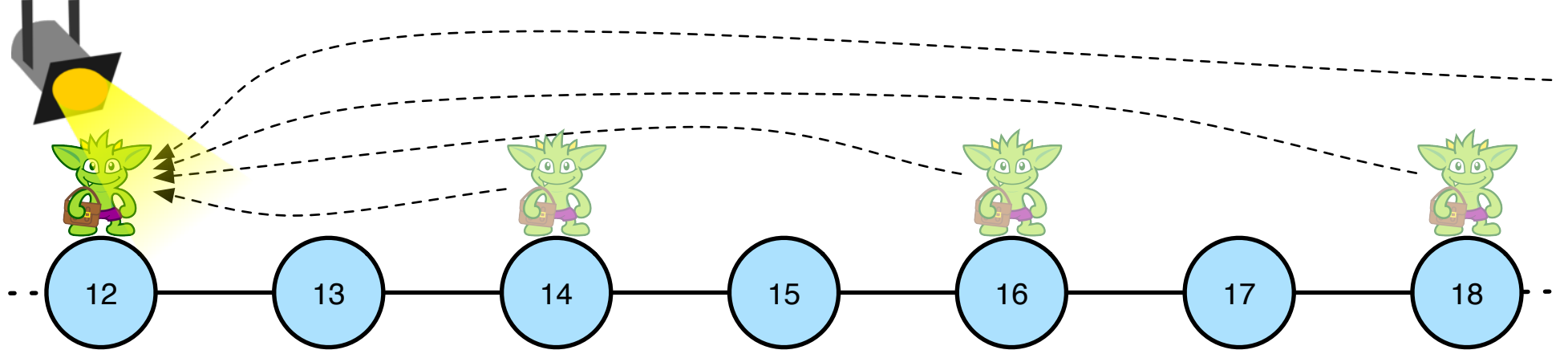


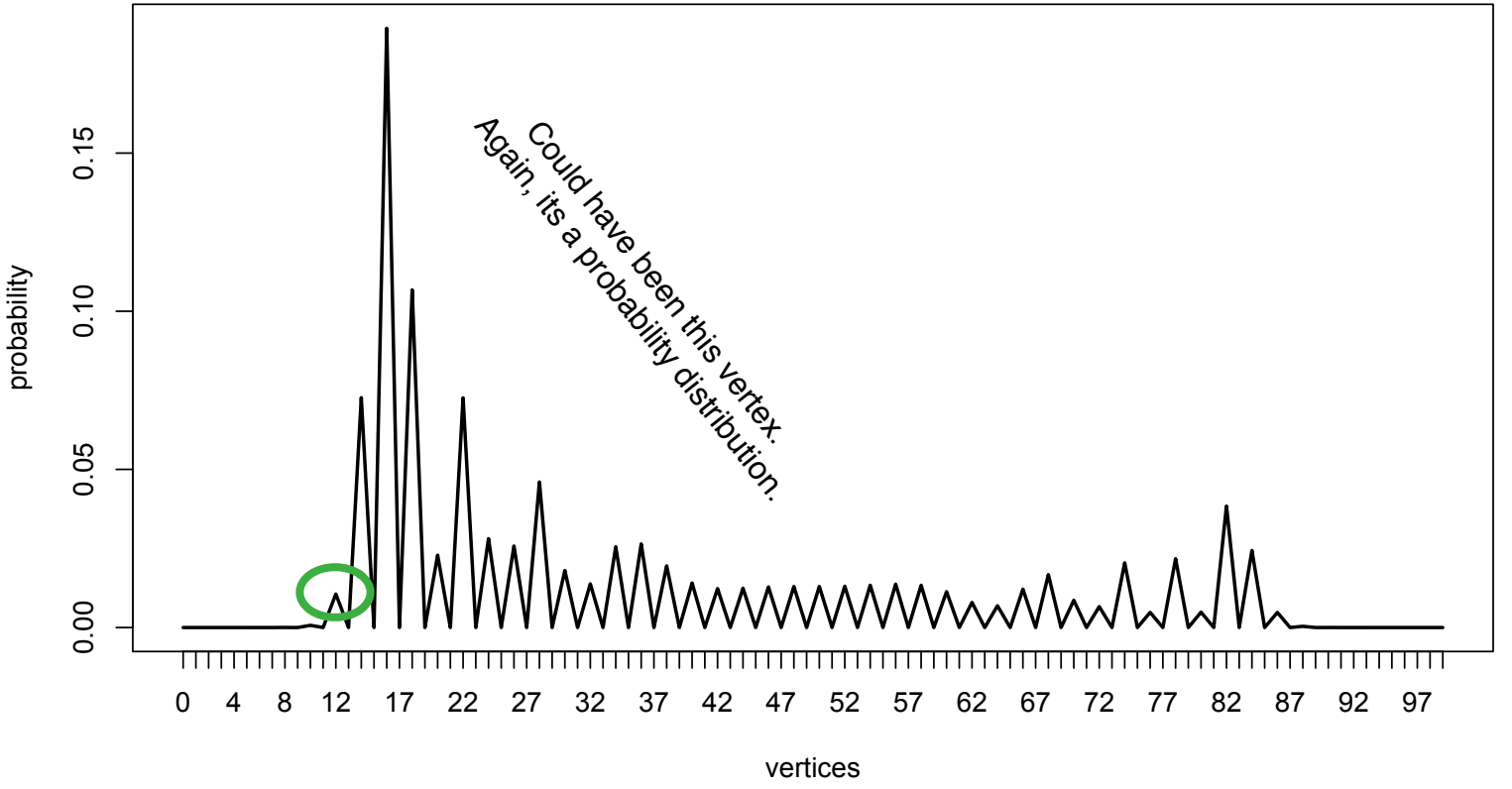
Wave Function Collapse



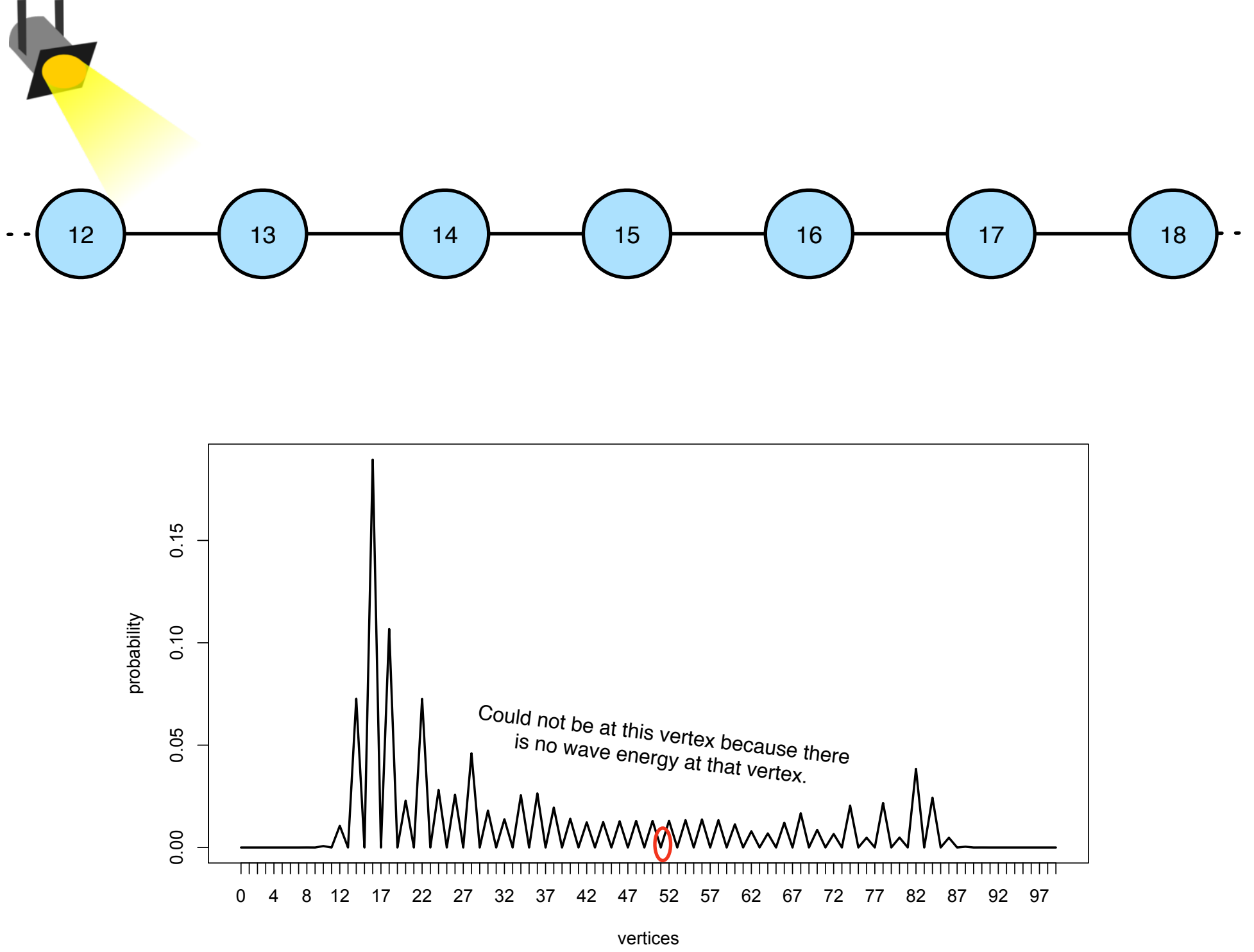


Classical Particle

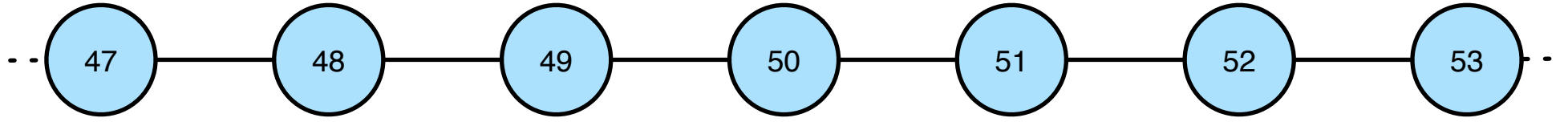




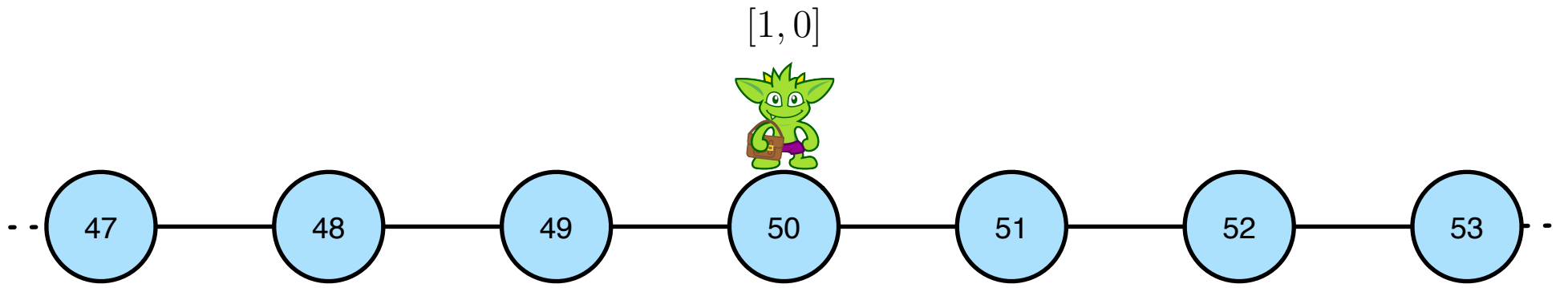
Wave Function Collapse



Wave Function Collapse



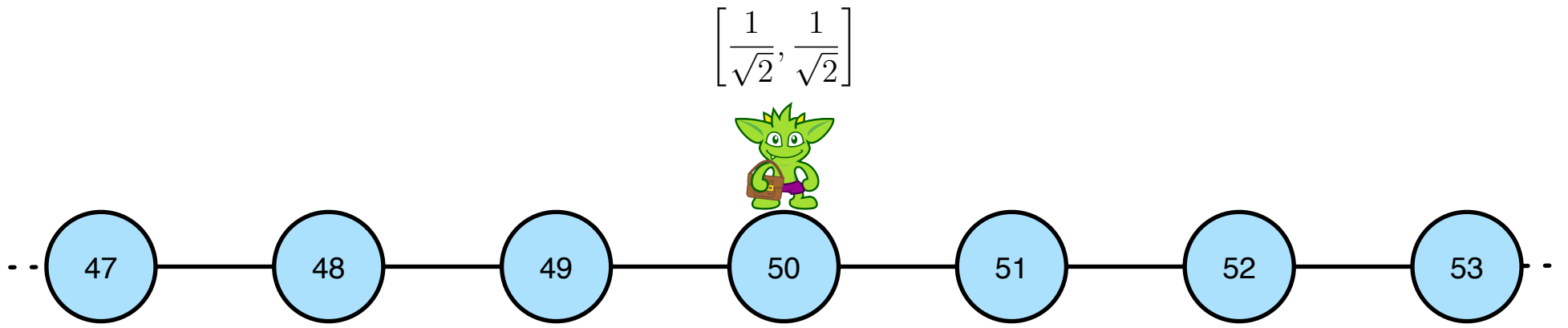

```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```




```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

```
sackSum = { a,b -> [a[0] + b[0],a[1] + b[1]] }
```

Classical Initial Step

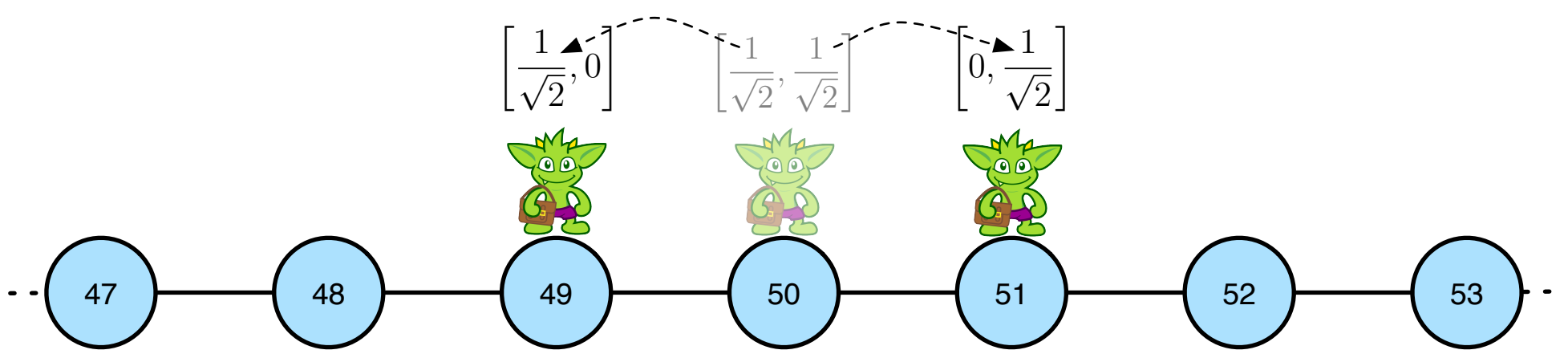



```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

```
hadamard = { a,b ->
  [(1/Math.sqrt(2)) * (a[0] + a[1]), (1/Math.sqrt(2)) * (a[0] - a[1])]
}
```

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

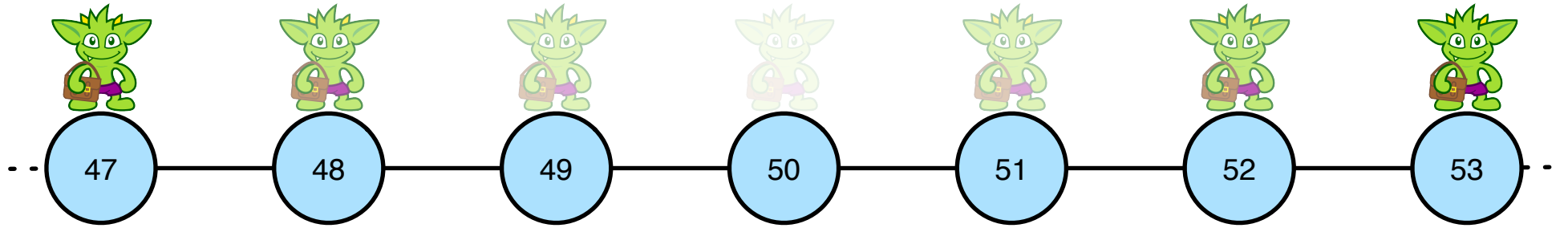
Coin Step




```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

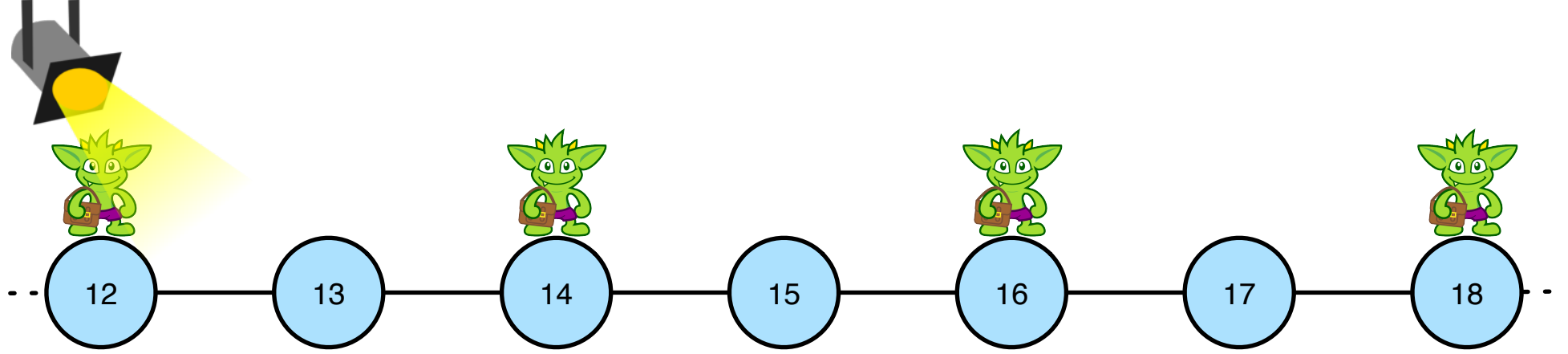
```
project = { a,b -> [a[0] * b[0], a[1] * b[1]] }
```

Shift Step




```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

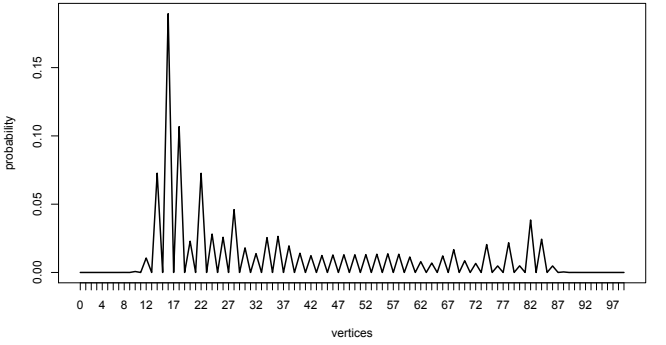
Diffuse the wave function for 50 steps



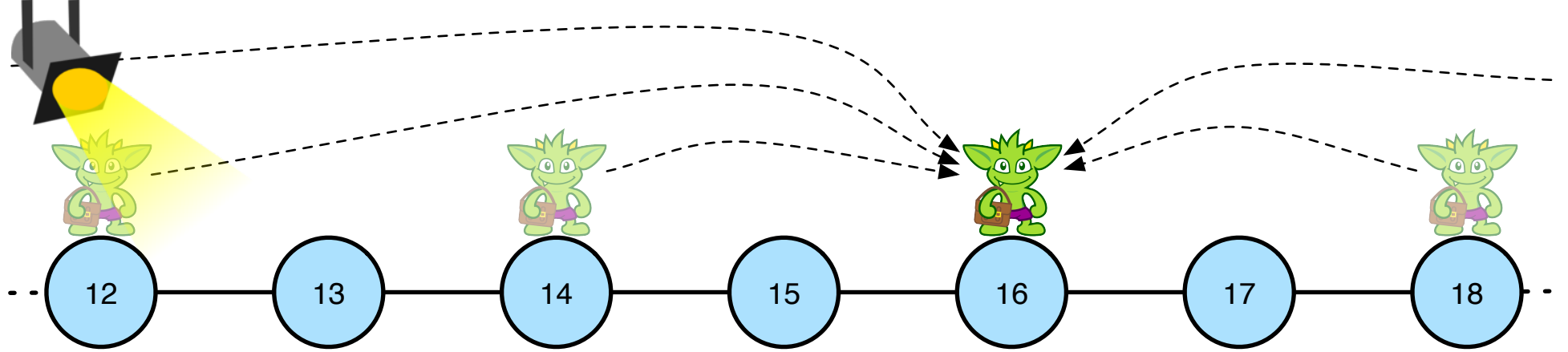

```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

```
norm = { Math.pow(it.get()[0],2) +
         Math.pow(it.get()[1],2) }
```

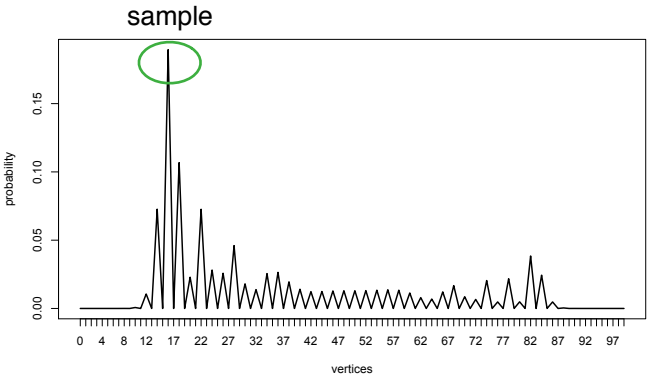
$$P(v) = |c_0|^2 + |c_1|^2$$



Wave Function to Probability Distribution




```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```



Wave Function Collapse (Classical Particle Manifested)

A Quantum Walk on a Line with Gremlin

```
g.withSack([1,0],sackSum).V(50).
  repeat(
    sack(hadamard).
    union(
      sack(project).by(constant([1,0])).out('left'),
      sack(project).by(constant([0,1])).out('right')
    )
  ).times(50).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

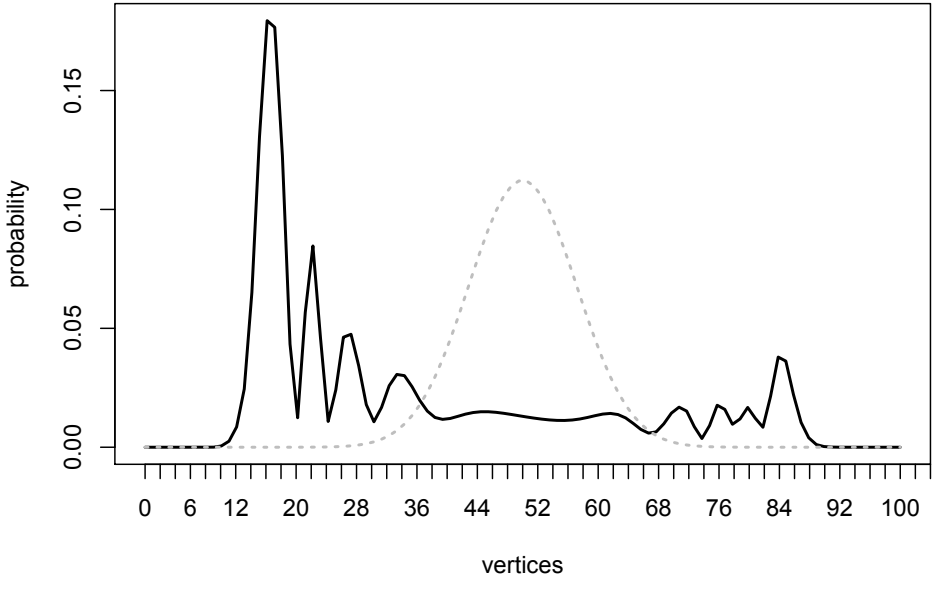
$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

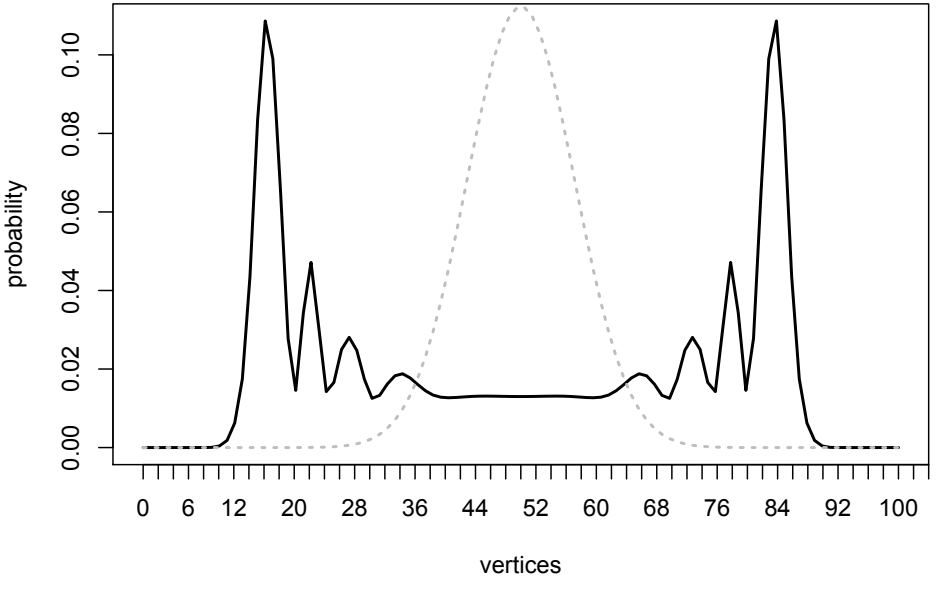
Unbalanced Unitary Operator

$i^2 = -1$ complex numbers

$$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

Balanced Unitary Operator





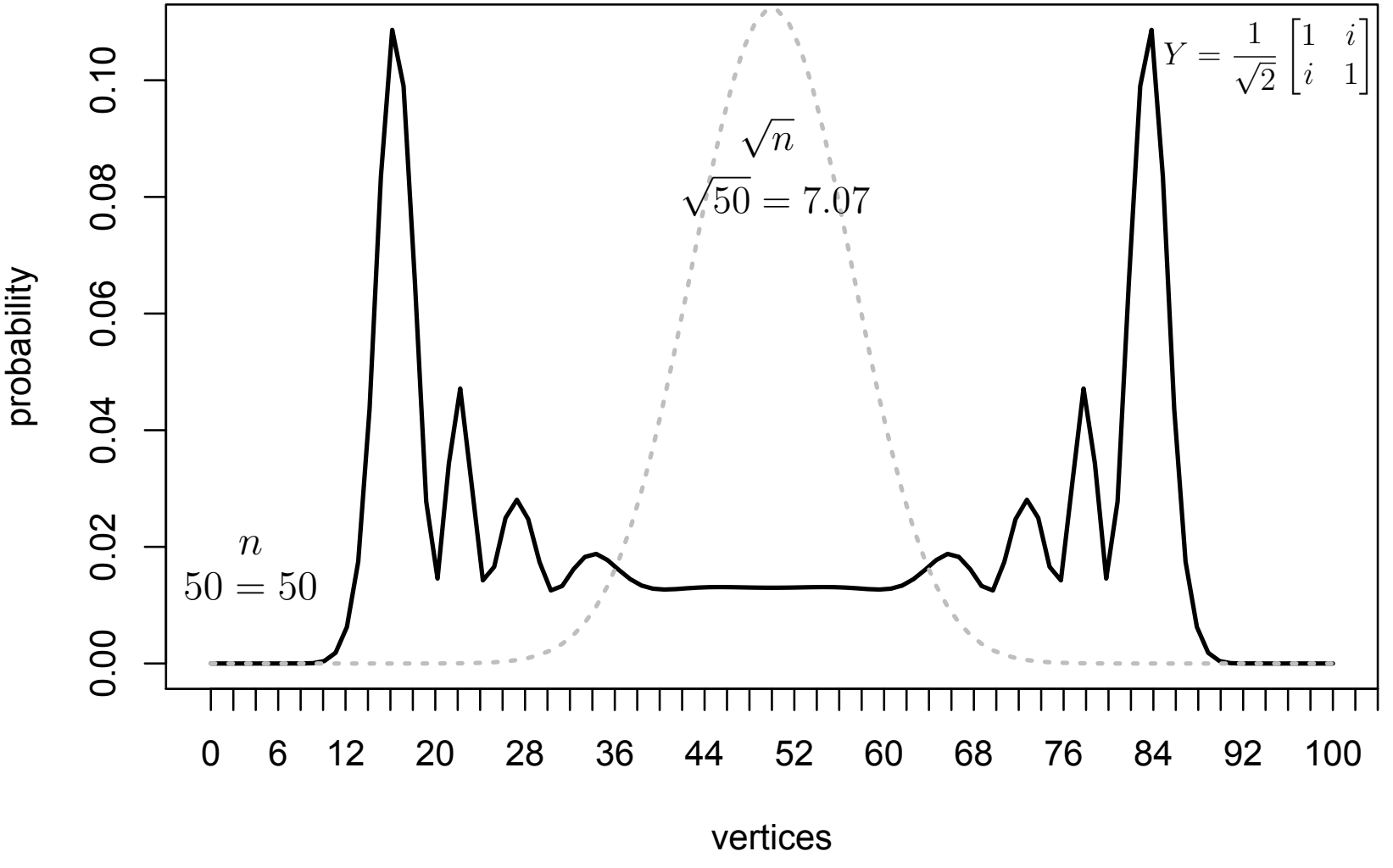
Any energy split operator can be used as long as:

1. The operation conserves the total energy in the system. no friction
2. It is possible to reverse the operation. no information loss

$$U \cdot U^* = I$$

In general, a **unitary operator** is used to diffuse energy.

The probability of seeing the particle at a particular vertex after 50 iterations for both a quantum walk and a classical walk.



Quantum walks can explore more graph in a shorter amount of time!

Part 4

The Double Slit Experiment

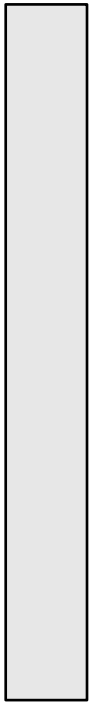
Thomas Young (1773-1829)

Question

"Is light a wave or a particle?"

continuous light source

double-slit screen

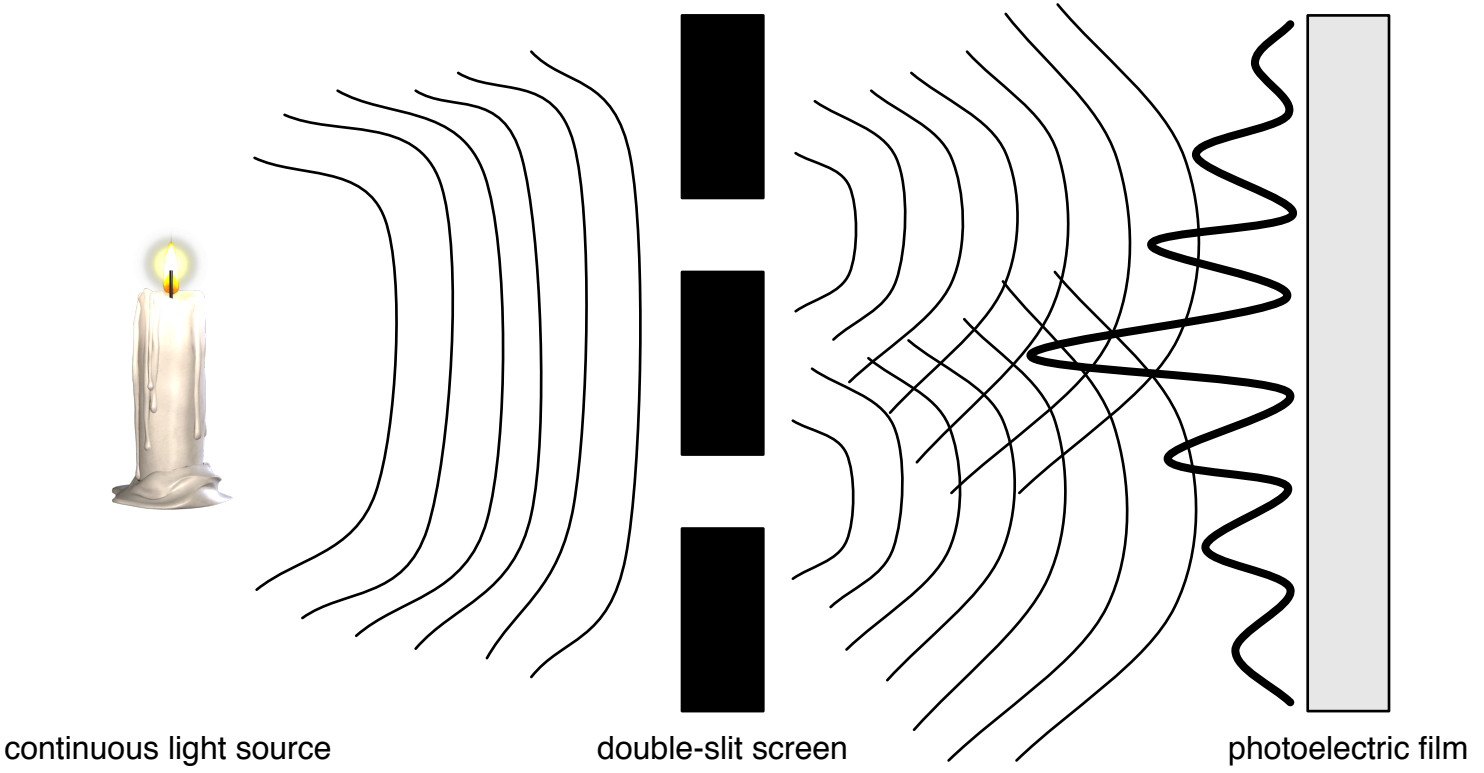


photoelectric film

Thomas Young (1773-1829)

Hypothesis

"If light is a wave, then the film will show a constructive and destructive interference pattern."



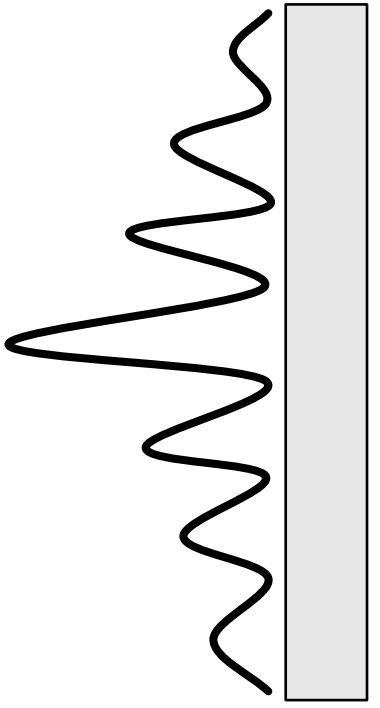
Thomas Young (1773-1829)

Result

"Light must be a wave because it has interference and refraction patterns like other natural waves such as water waves."

continuous light source

double-slit screen



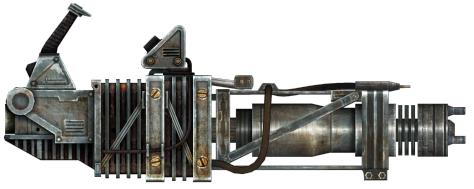
photoelectric film

G. I. Taylor (1886-1975)

Question

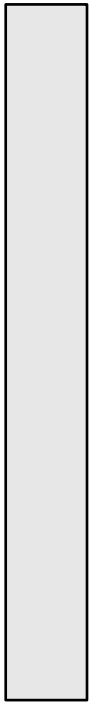
"What happens when only a single photon of light is emitted?"

The most indivisible amount of light.



single photon light source

double-slit screen

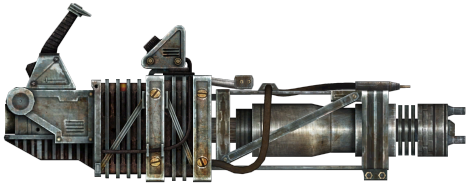


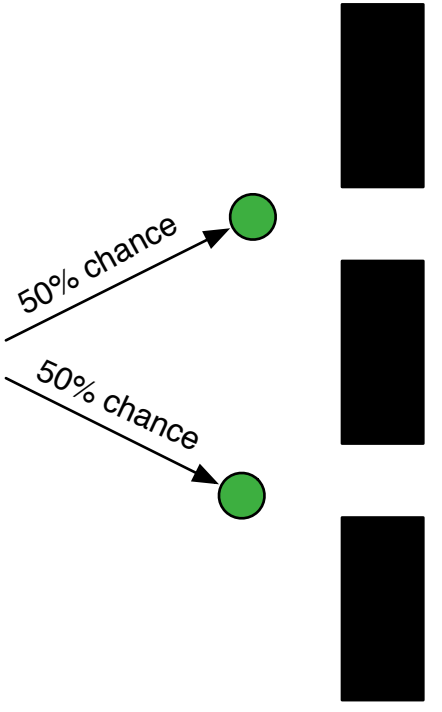
photoelectric film

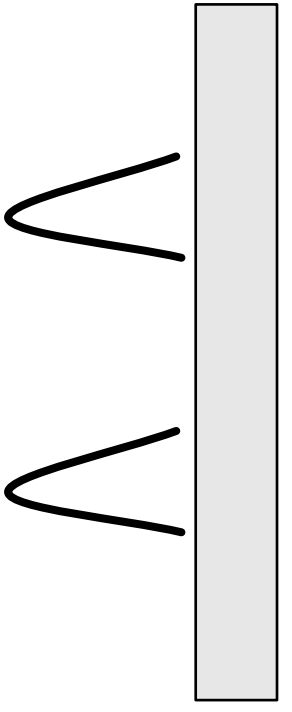
G. I. Taylor (1886-1975)

Hypothesis

"The photons must act like bullets and hit directly behind the slits."







single photon light source

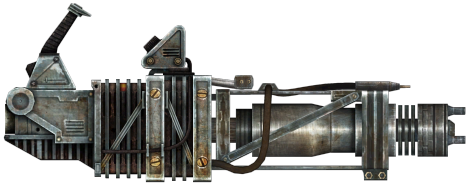
double-slit screen

photoelectric film

G. I. Taylor (1886-1975)

Experiment

"Shoot photon #1!"



single photon light source

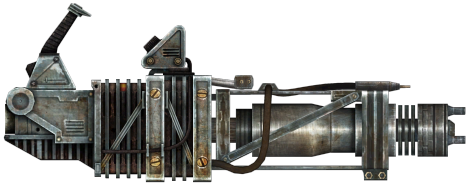
double-slit screen

photoelectric film

G. I. Taylor (1886-1975)

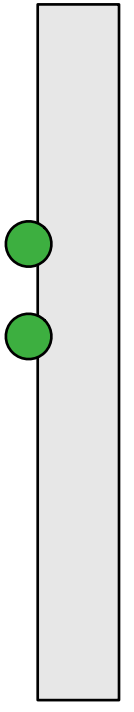
Experiment

"Shoot photon #2!"



single photon light source

double-slit screen

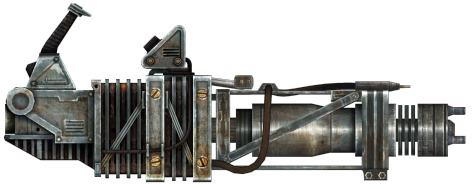


photoelectric film

G. I. Taylor (1886-1975)

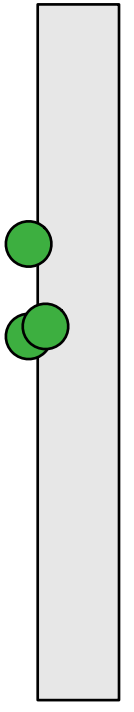
Experiment

"Shoot photon #3!"



single photon light source

double-slit screen

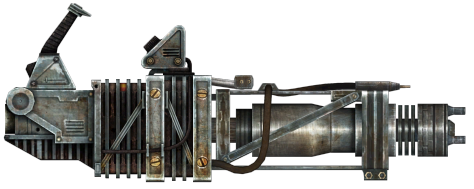


photoelectric film

G. I. Taylor (1886-1975)

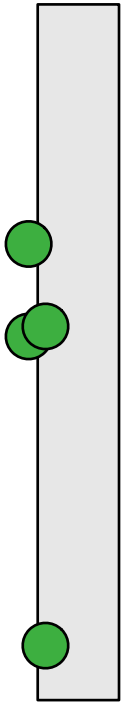
Experiment

"Shoot photon #4!"



single photon light source

double-slit screen



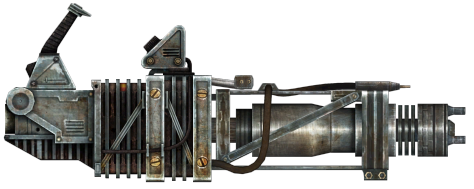
photoelectric film

Fast forward to photon number 20.

G. I. Taylor (1886-1975)

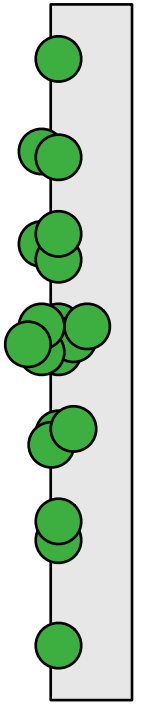
Experiment

"Shoot photon #20!"



single photon light source

double-slit screen

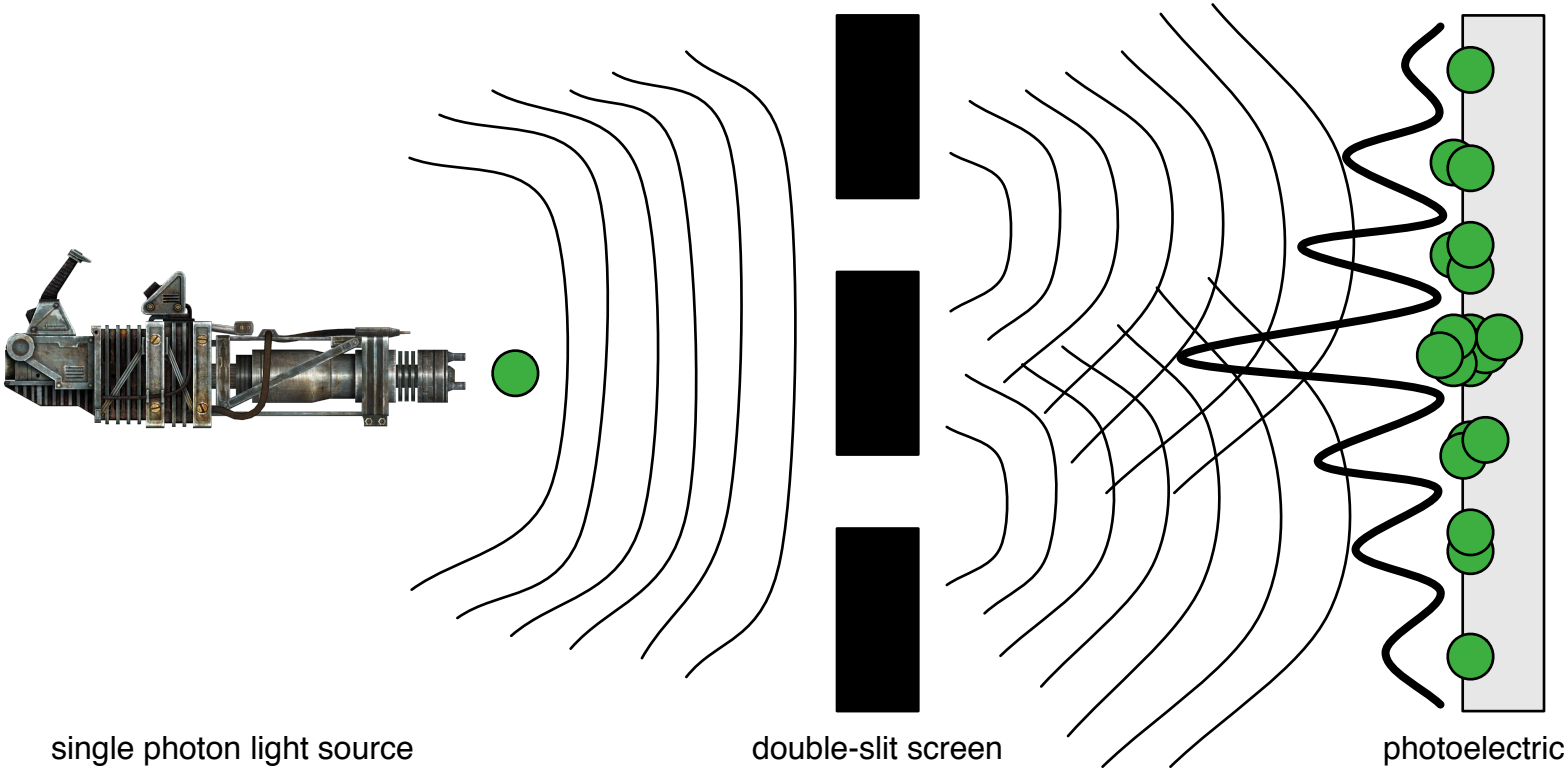


photoelectric film

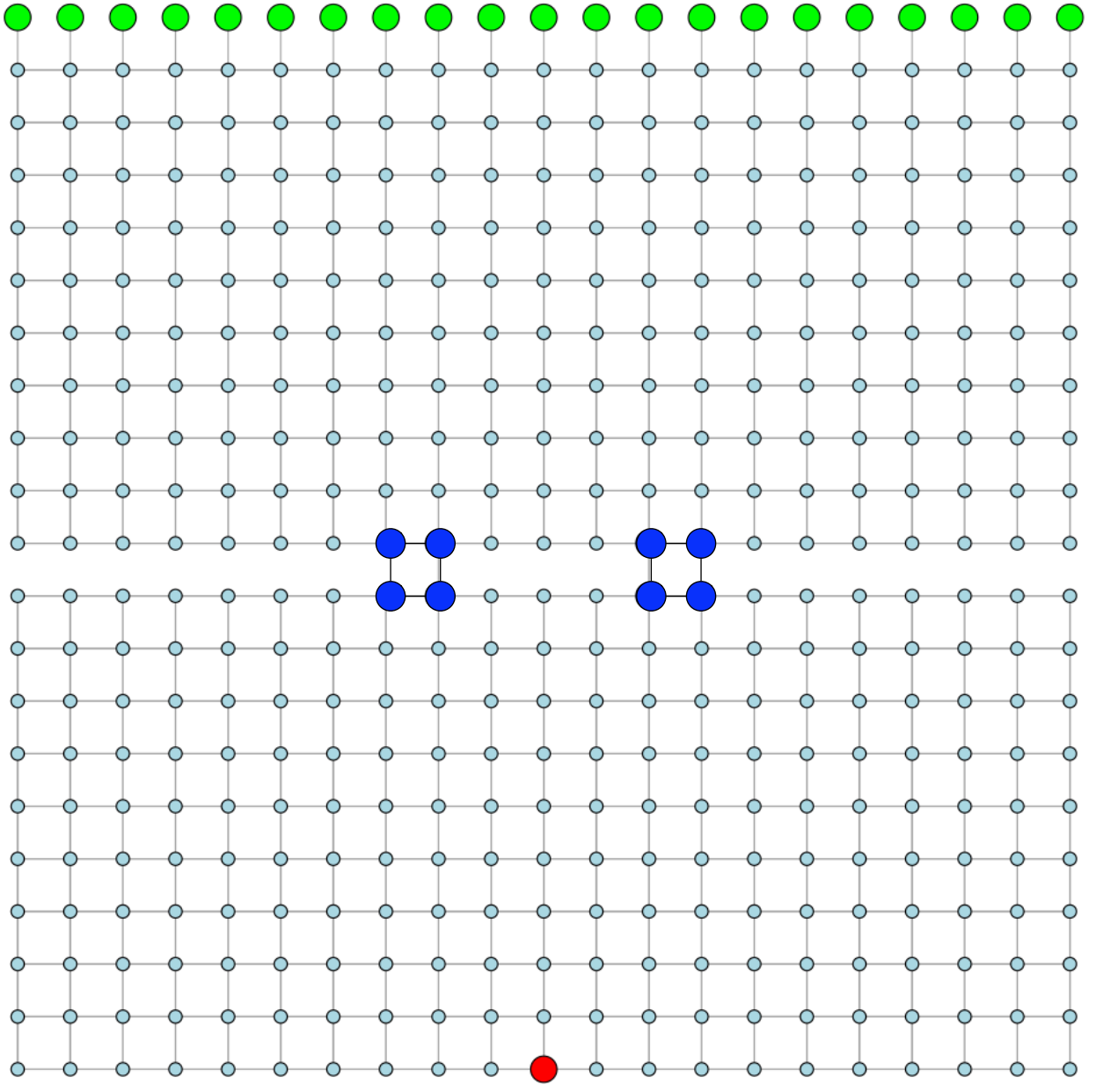
G. I. Taylor (1886-1975)

Result

"A single quantum of light (a photon) turns into a wave after emission and then is localized back to a particle at the film based on the wave function."

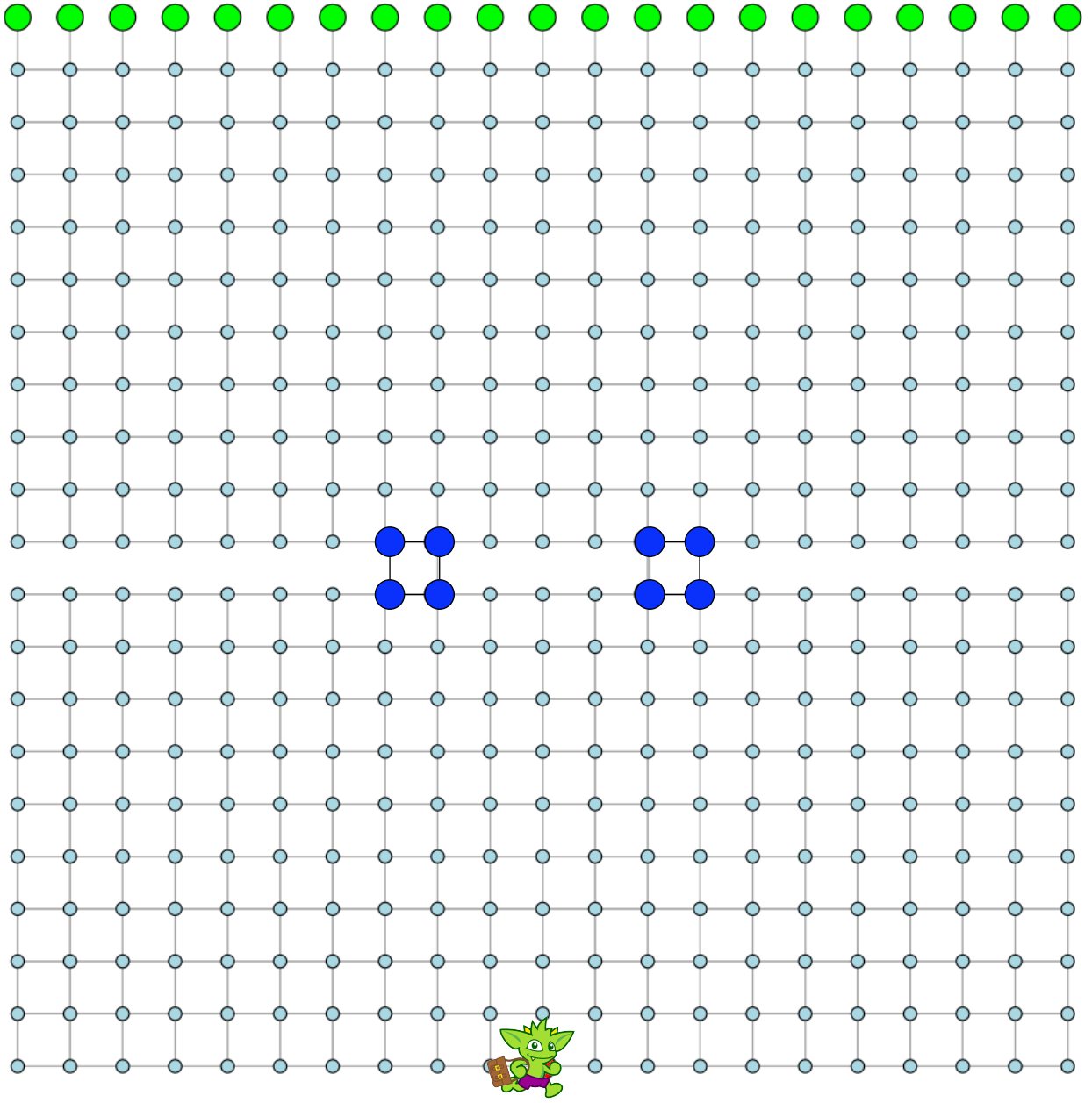


photoelectric film

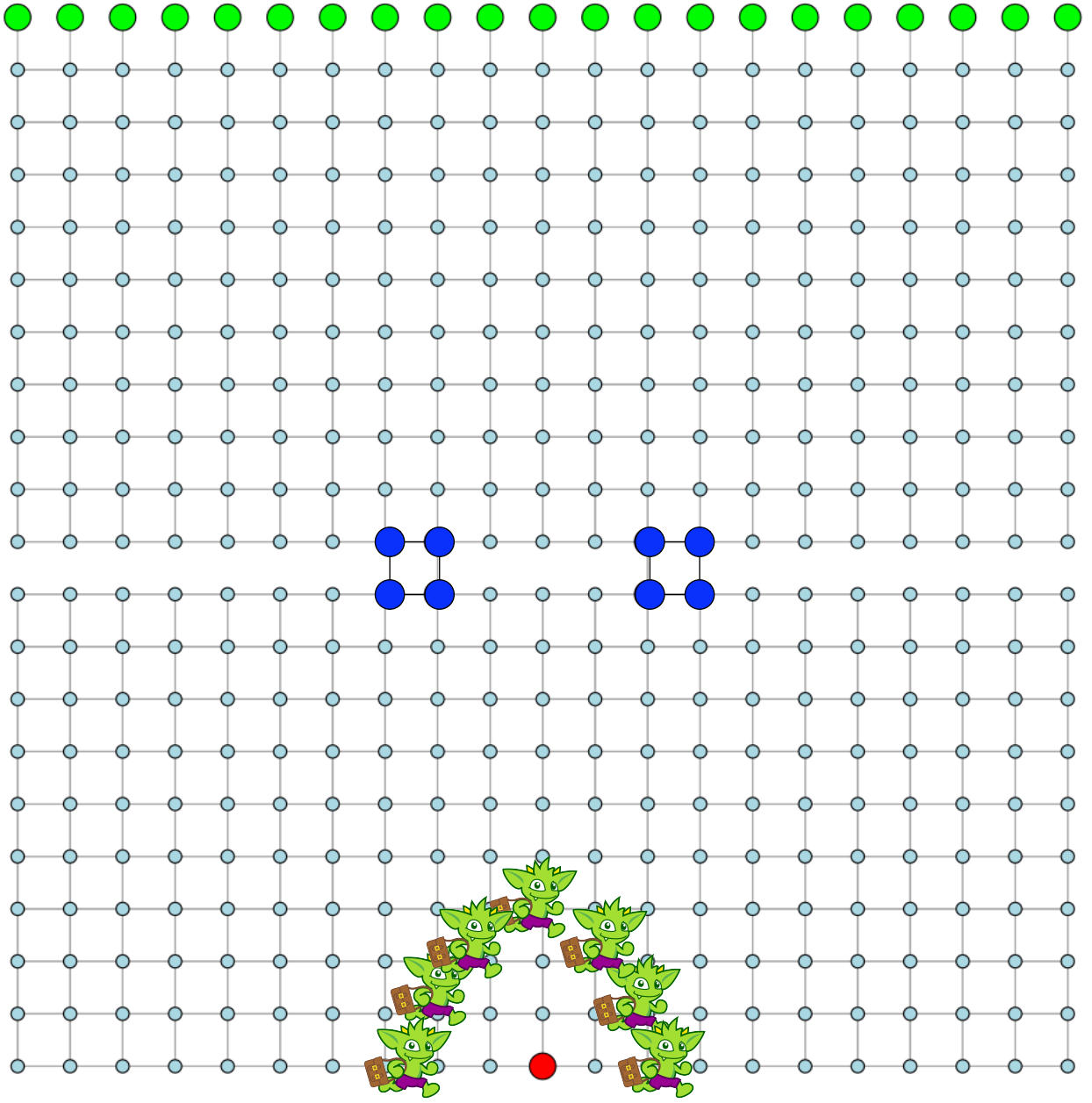


double-slit screen

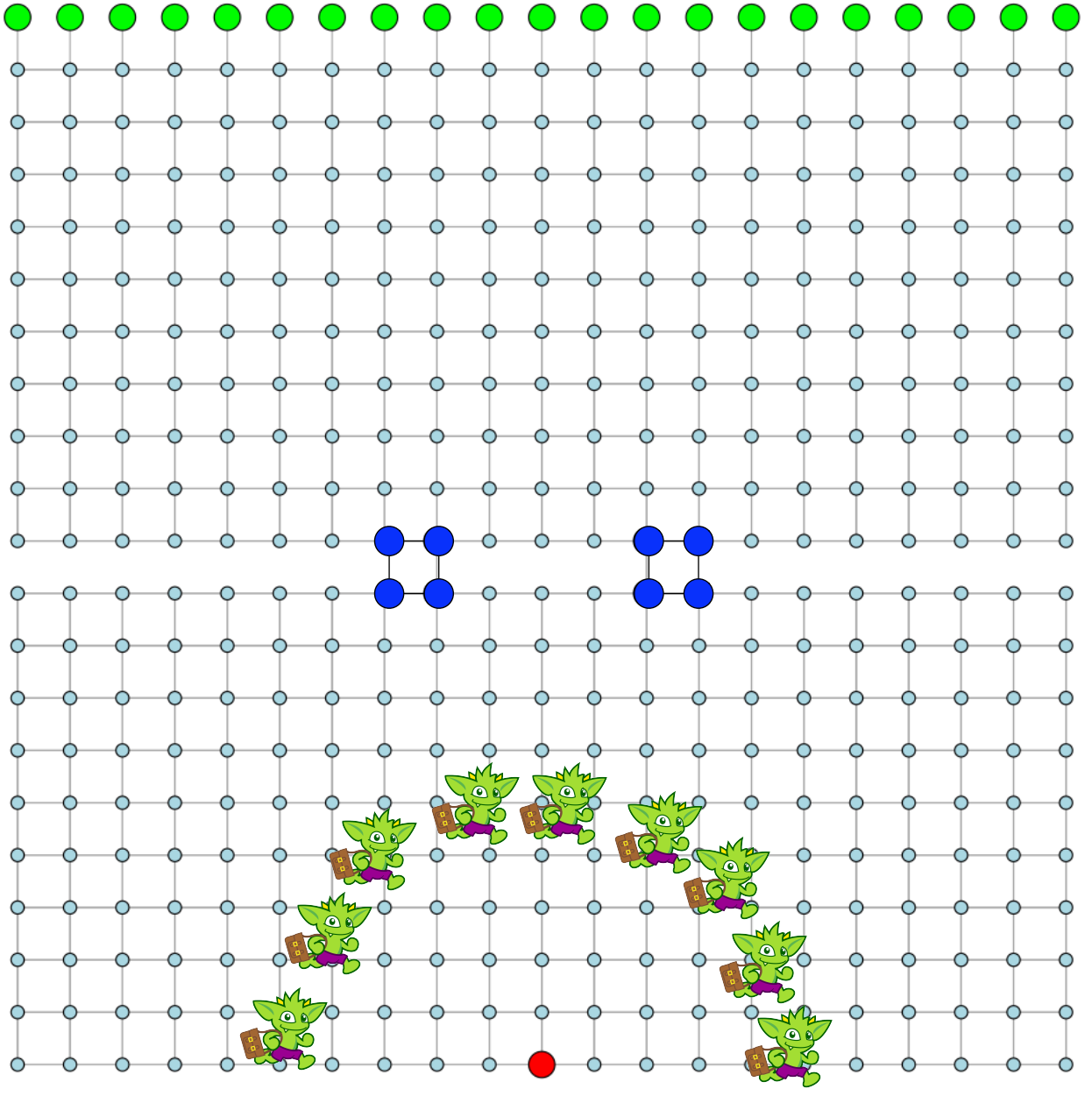
single photon light source

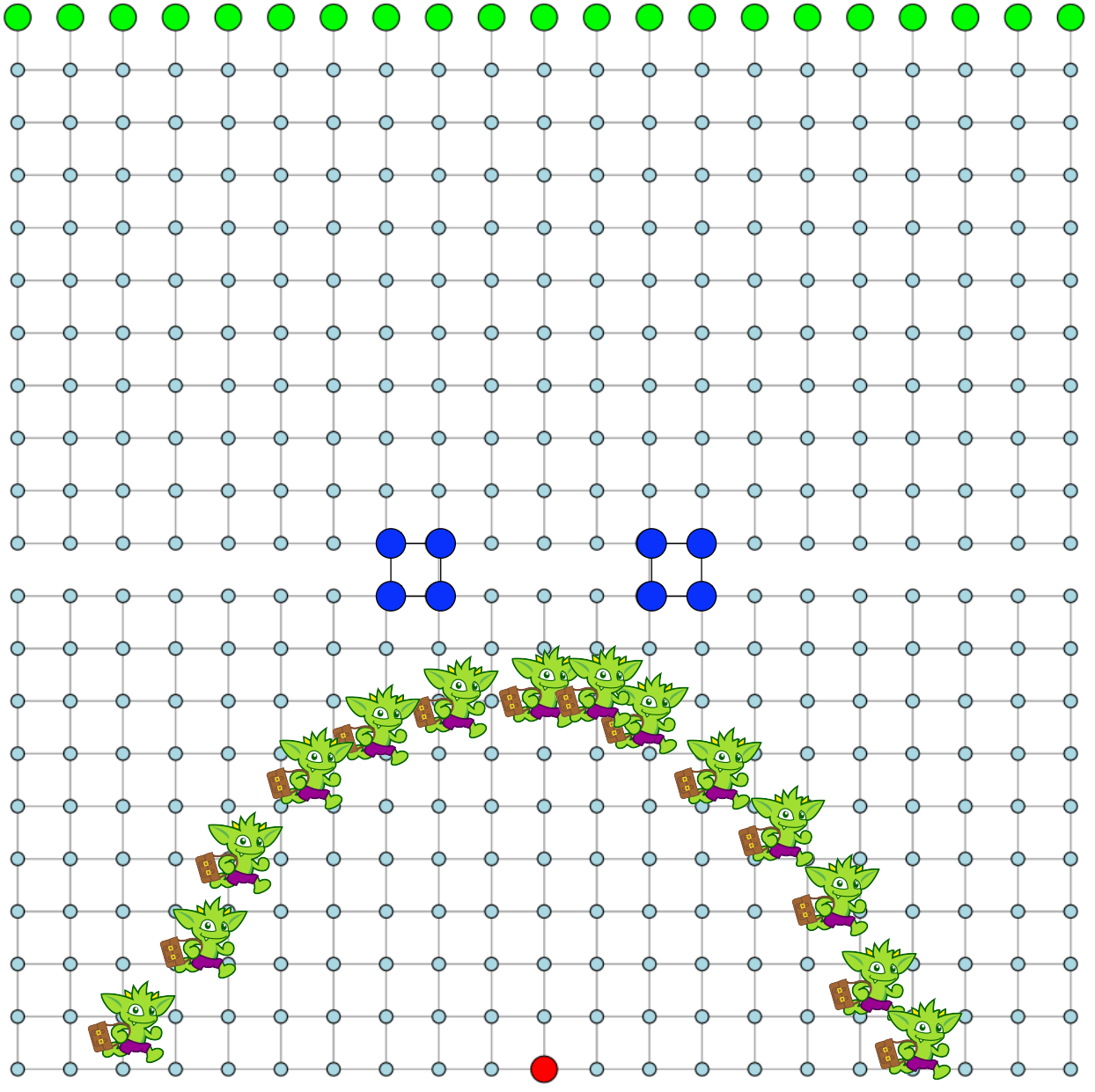


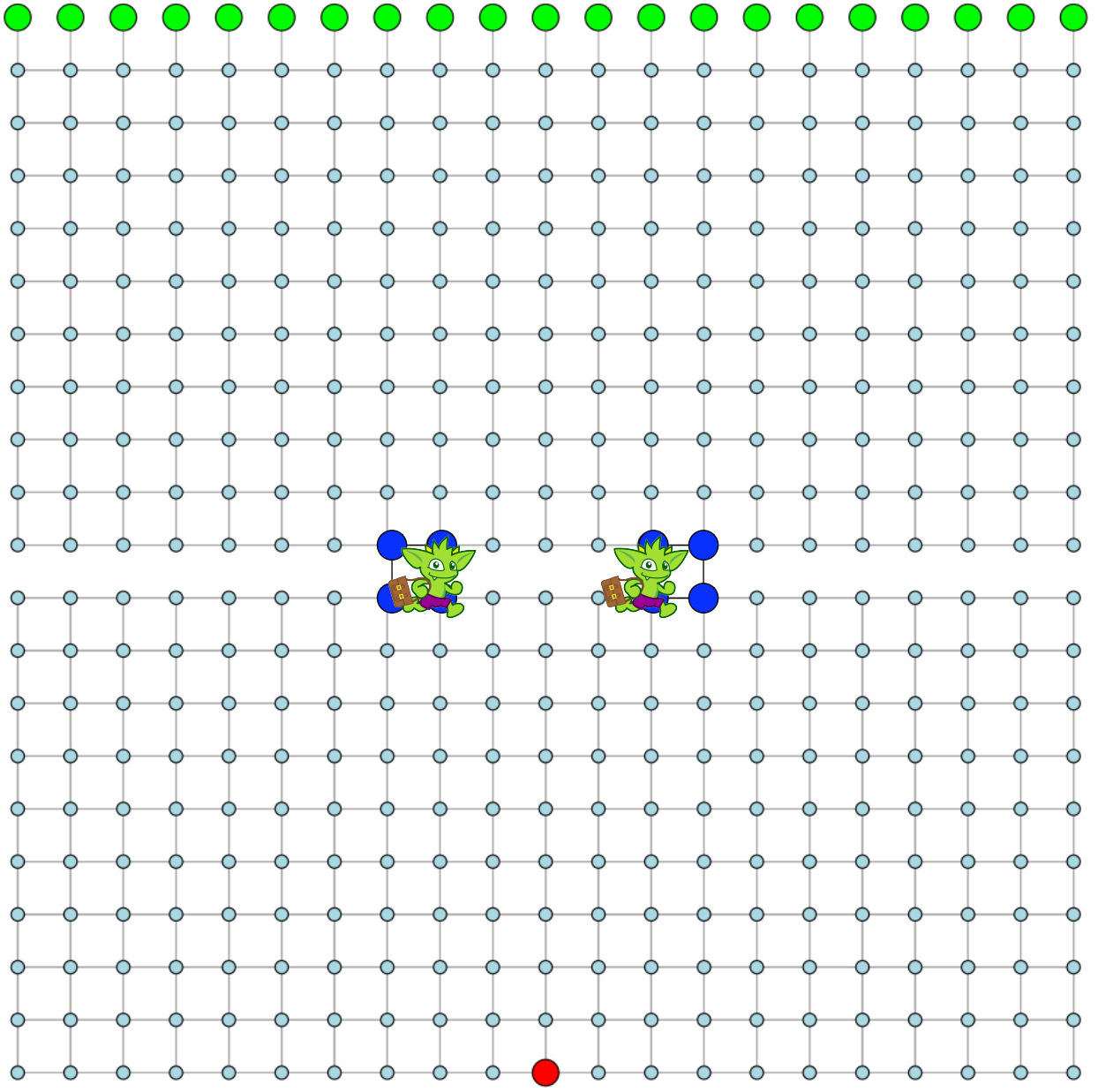
Classical particle.



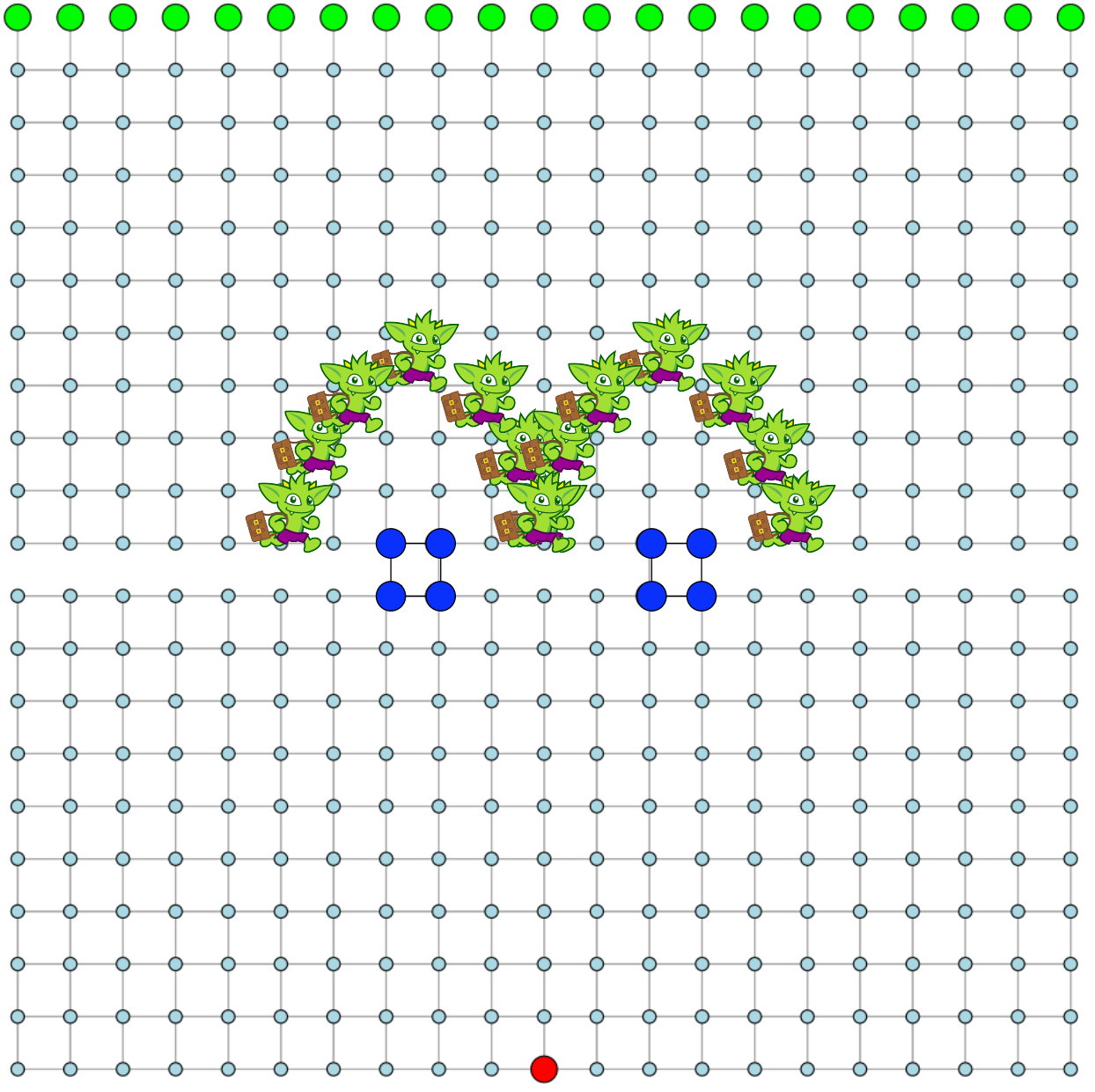
Particle becomes a wave.

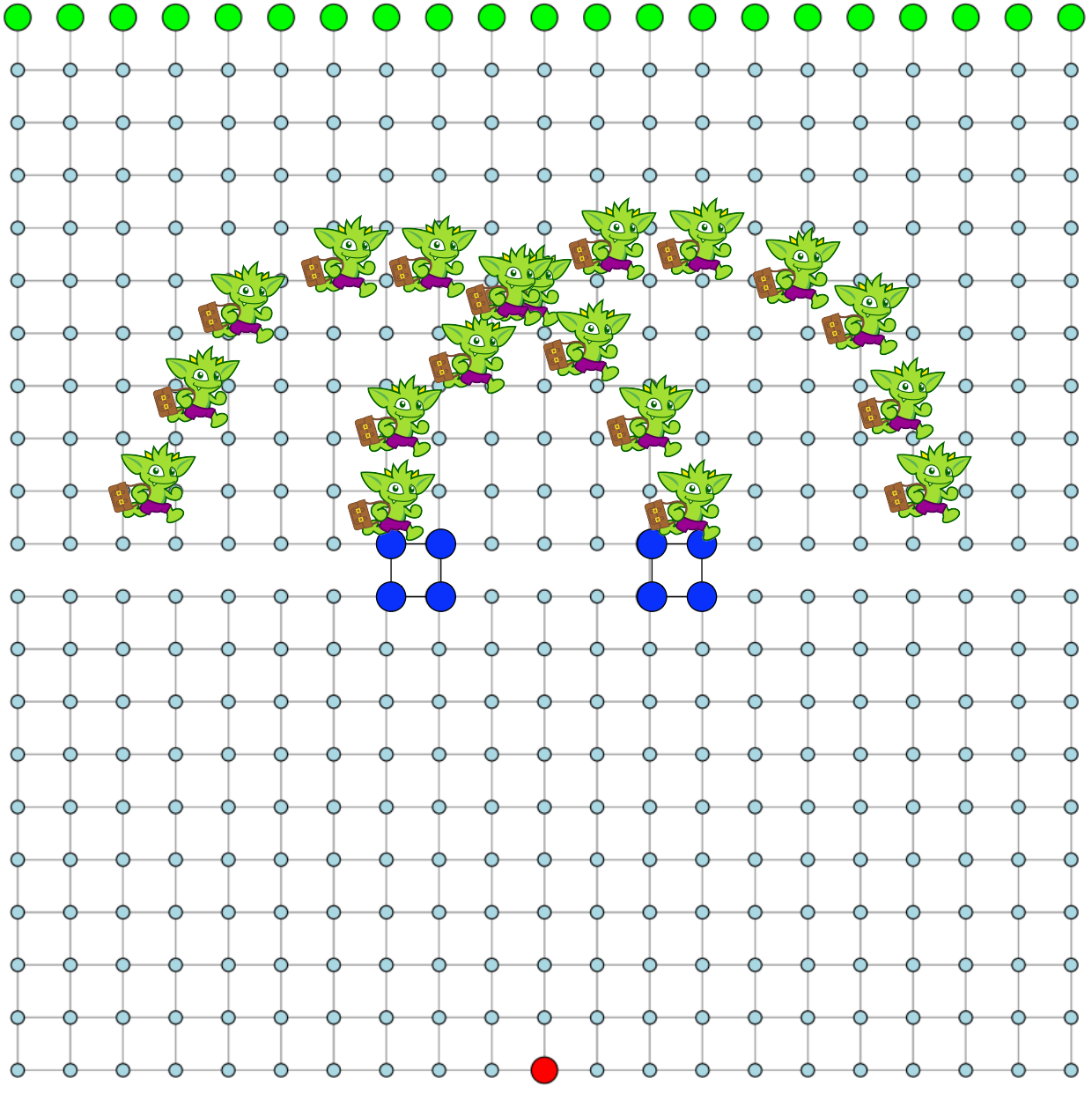




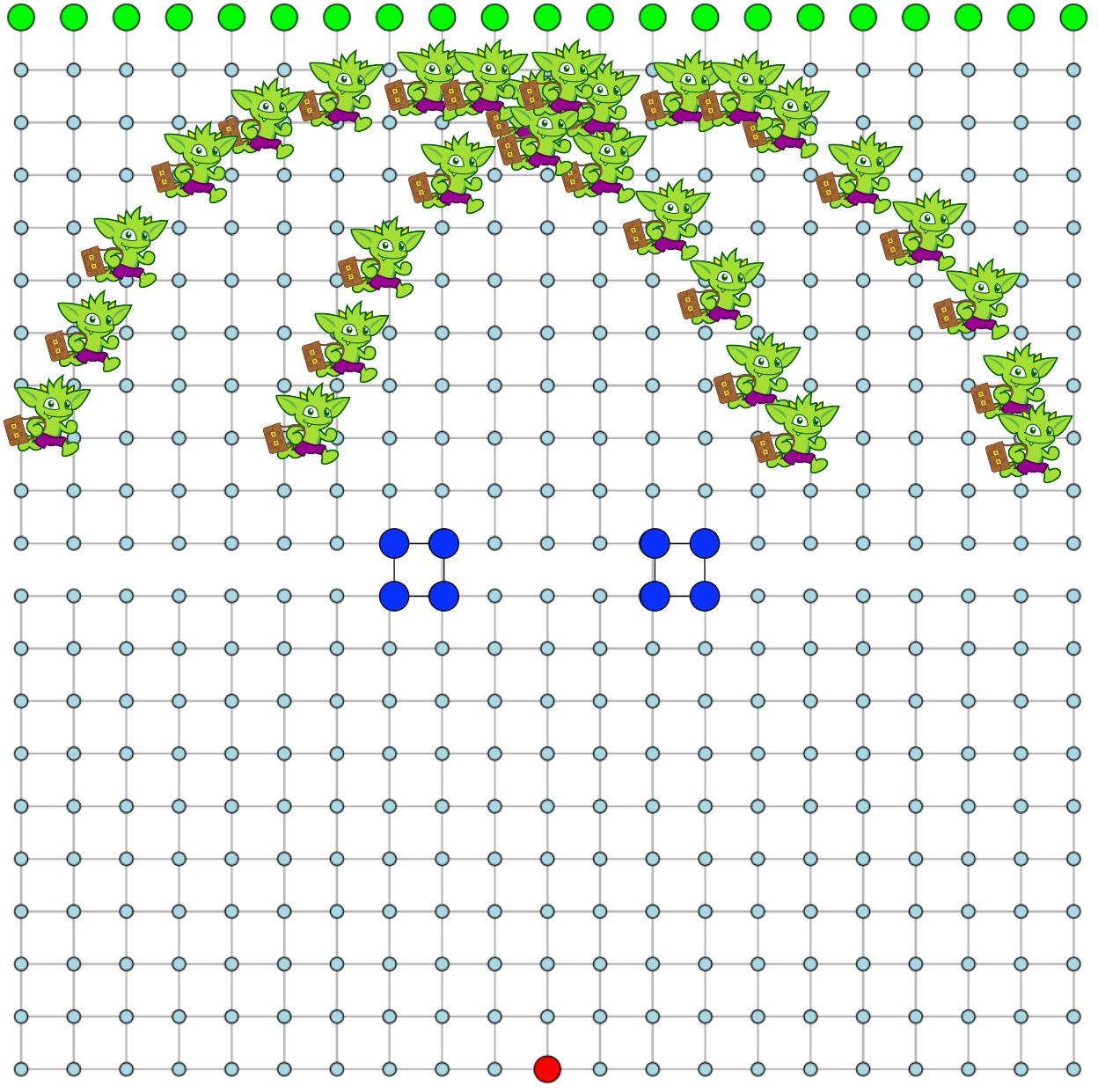


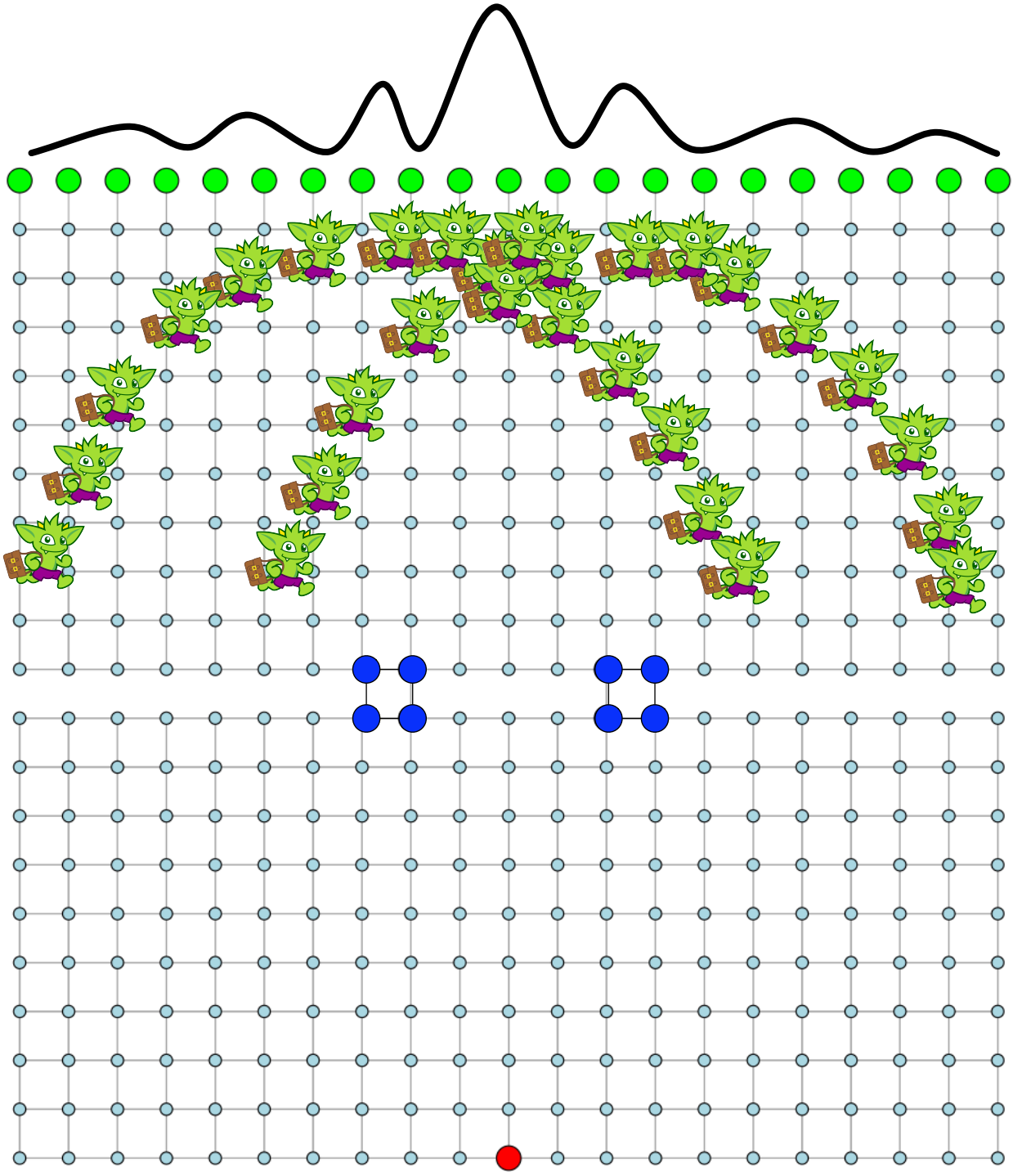
"Particle" takes both options.



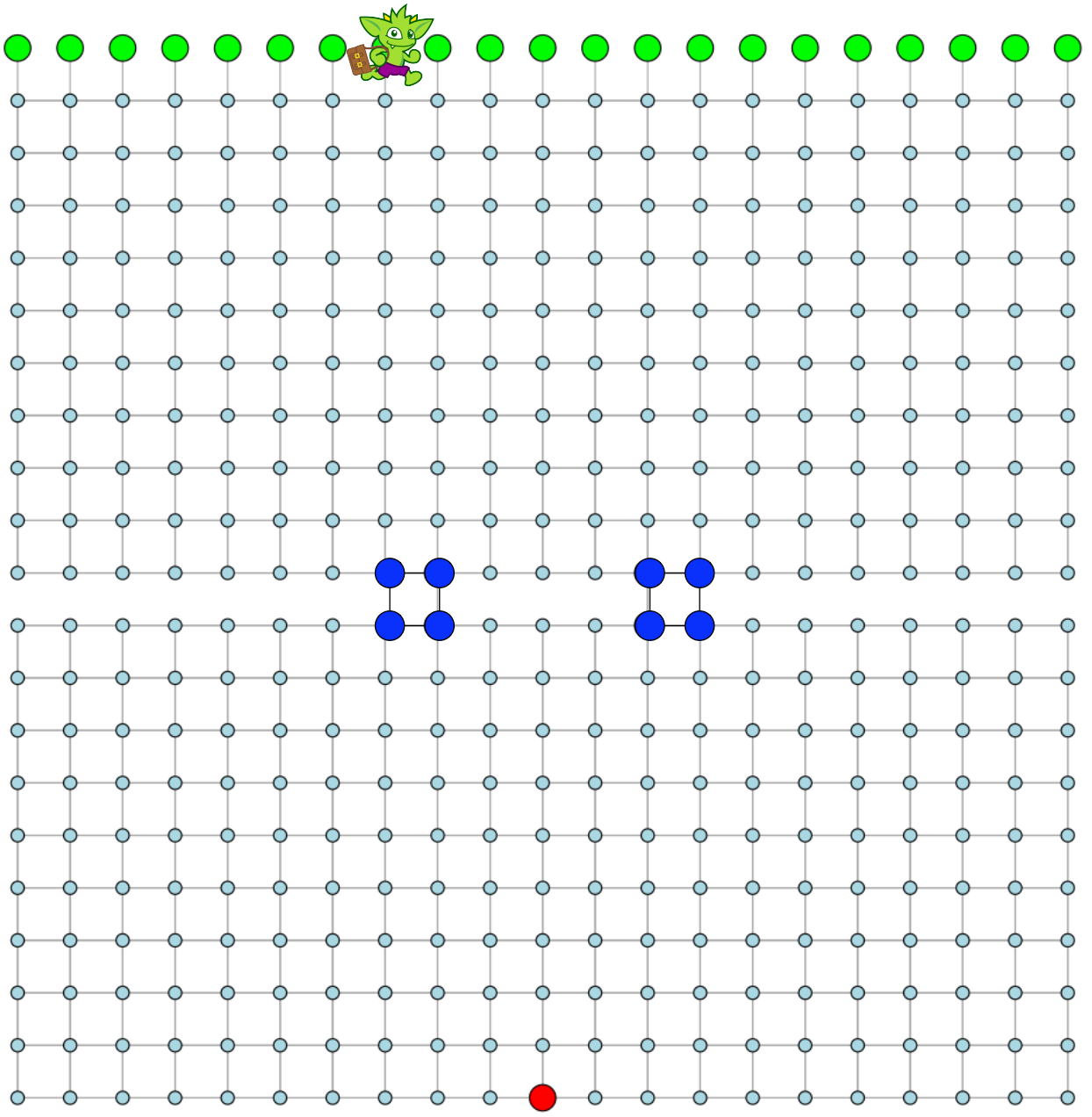


Wave interference.

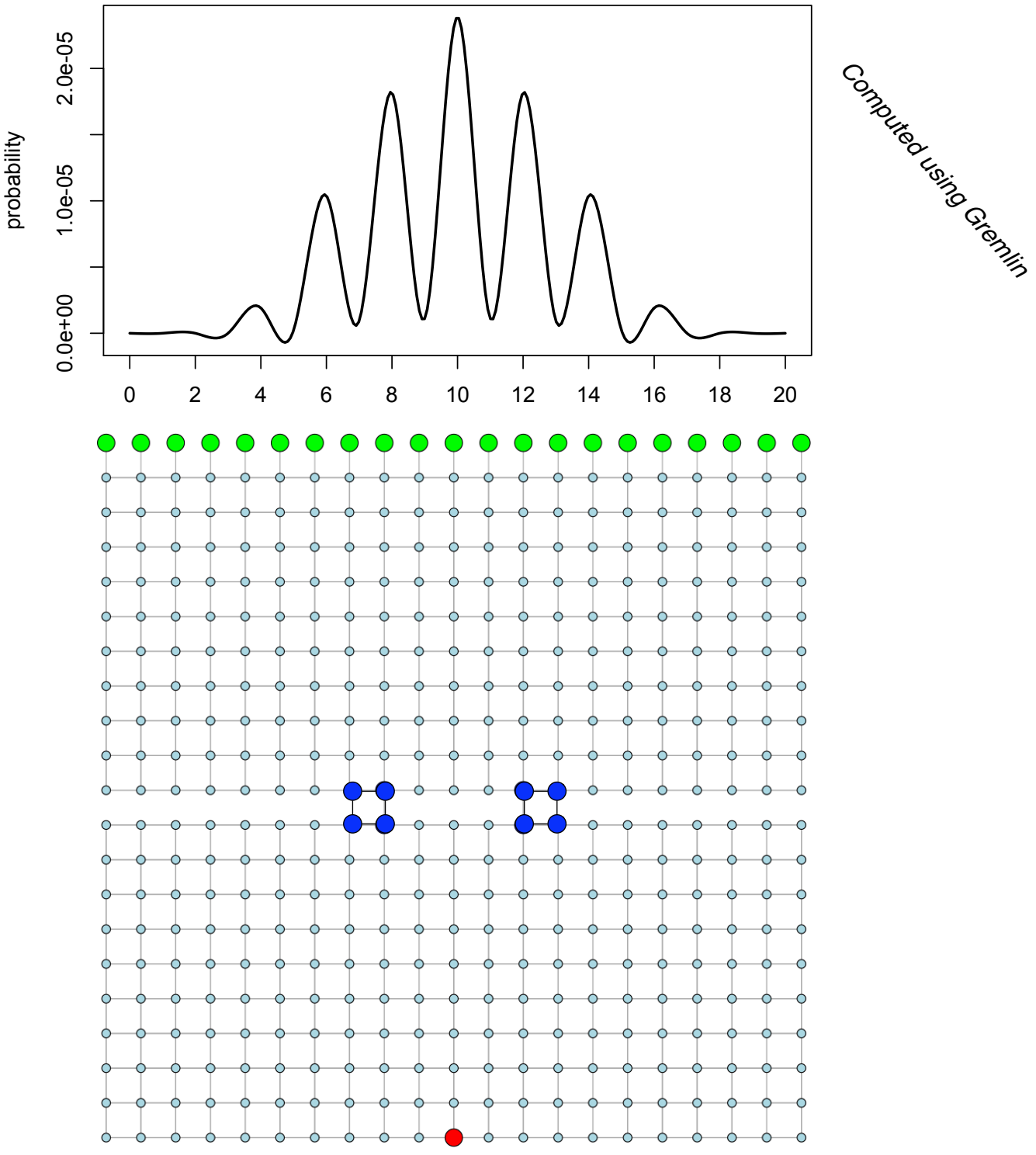




Probability distribution generated from the wave function.



Wave function collapses back to a classical particle.


probability
0.0e+00
1.0e-05
2.0e-05
0
2
4
6
8
10
12
14
16
18
20
Computed using Gremlin

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

Four degrees of freedom means you have a four entry energy vector.

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

Bottom-middle vertex. The single photon light source.

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

A 4x4 unitary operator called the Grover coin.

$$R = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

Diffuse the wave in all four directions on the lattice.

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

If there is a "wall," reflect the energy back. No decoherence/friction.

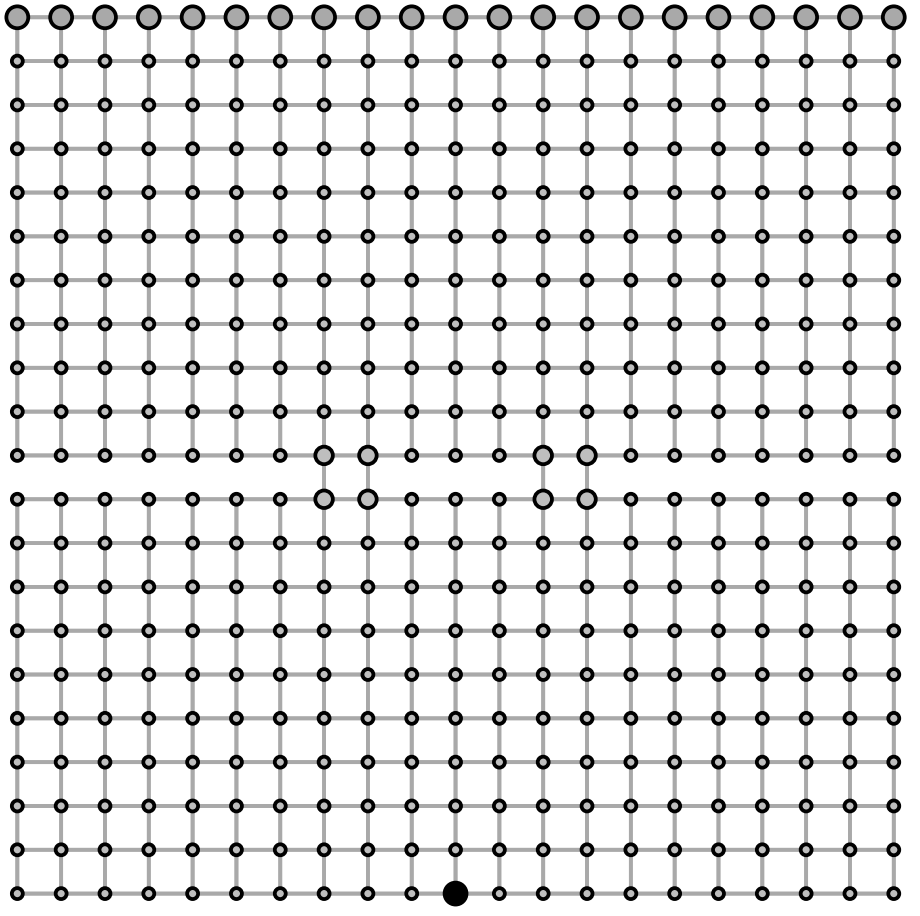
The article presents the details around reflection.

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

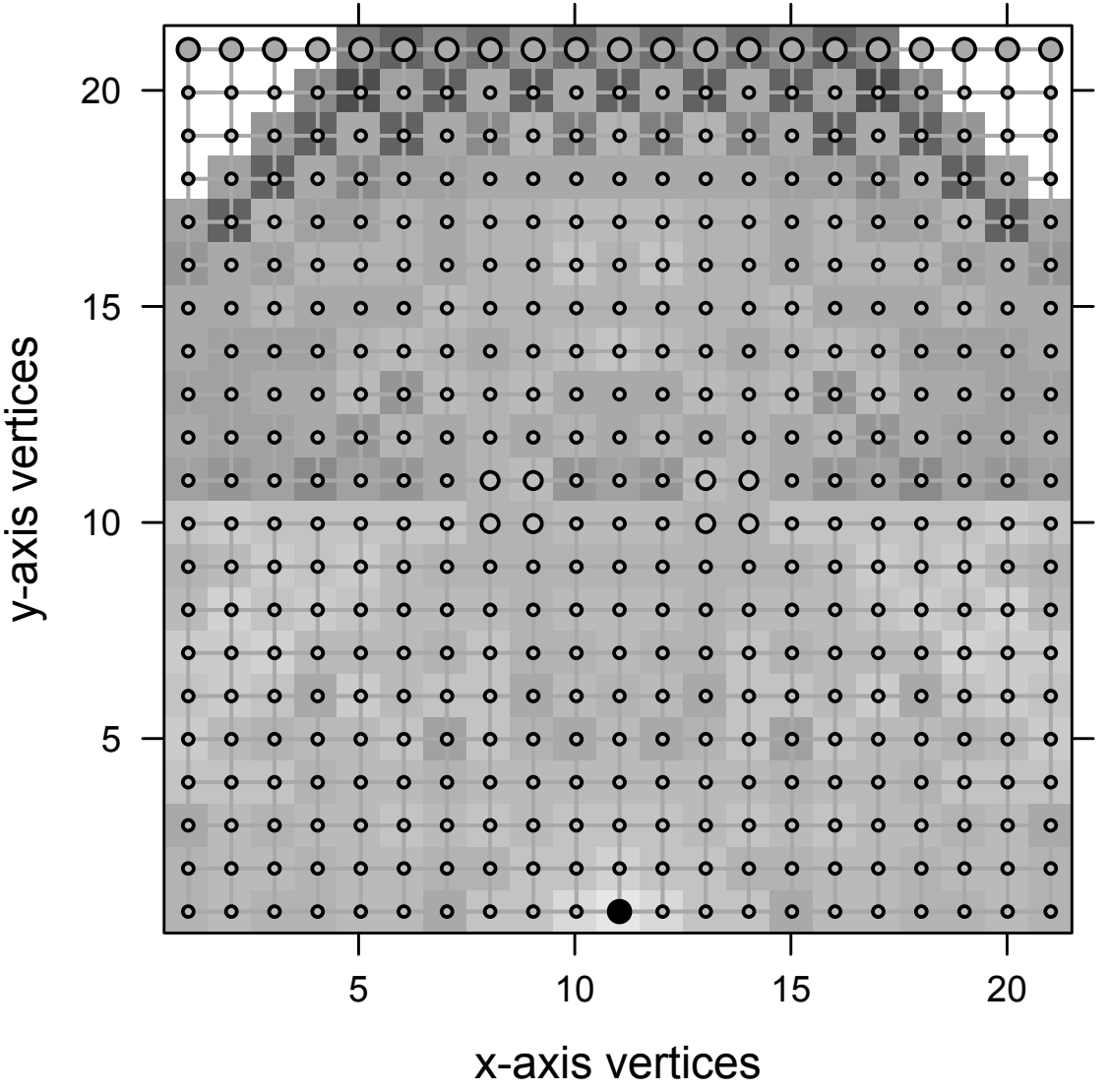
As before, generate the probability distribution and sample it (i.e. collapse the wave form).

```
g.withSack([0,0,1,0],sackSum).V(10).
  repeat(
    sack(grover).
    union(
      choose(out('left'),
        out('left').sack(project).by(constant([1,0,0,0])),
        identity().sack(project).by(constant([1,0,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('right'),
        out('right').sack(project).by(constant([0,1,0,0])),
        identity().sack(project).by(constant([0,1,0,0])).
          sack(flip).by(constant("lr"))),
      choose(out('up'),
        out('up').sack(project).by(constant([0,0,1,0])),
        identity().sack(project).by(constant([0,0,1,0])).
          sack(flip).by(constant("ud"))),
      choose(out('down'),
        out('down').sack(project).by(constant([0,0,0,1])),
        identity().sack(project).by(constant([0,0,0,1])).
          sack(flip).by(constant("ud"))))).
  times(22).
  group().by(id).by(sack().map(norm)).
  unfold().sample(1).by(values)
```

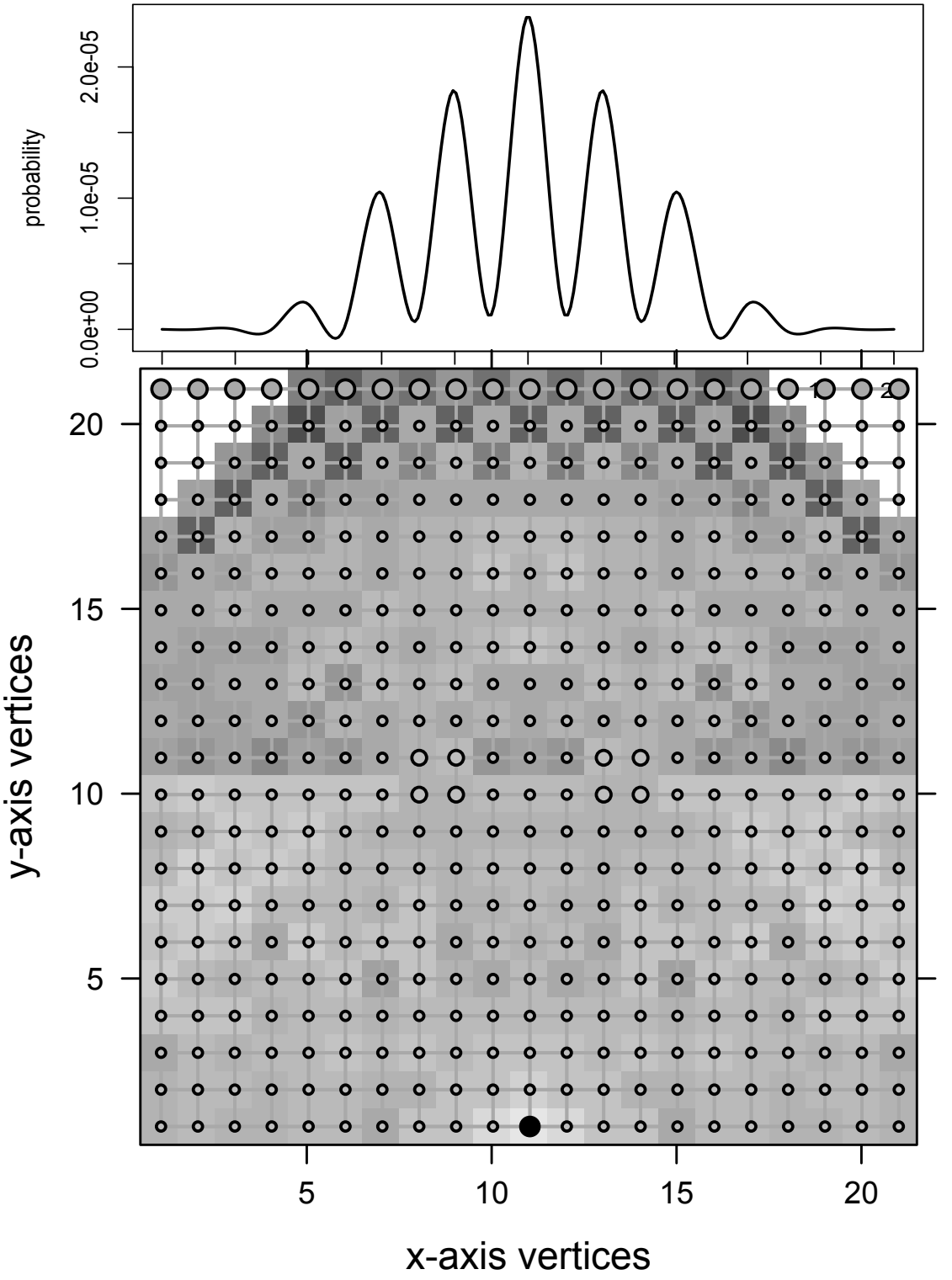
A Quantum Walk on a Bounded Lattice with Gremlin



Step 0



Step 22

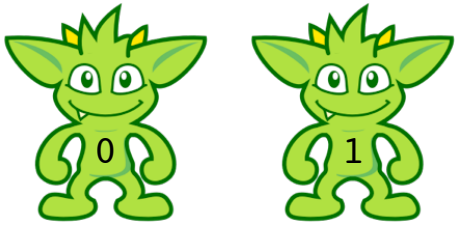

probability
0.0e+00
1.0e-05
2.0e-05
y-axis vertices
5
10
15
20
x-axis vertices
5
10
15
20

Step 22

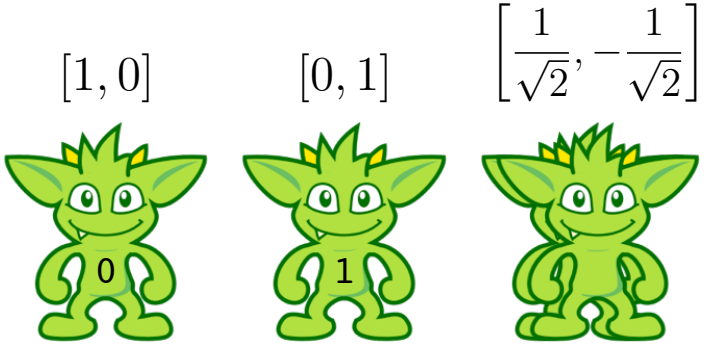
Part 5

Quantum Computing

In **classical computing**, a bit is either 0 or 1.

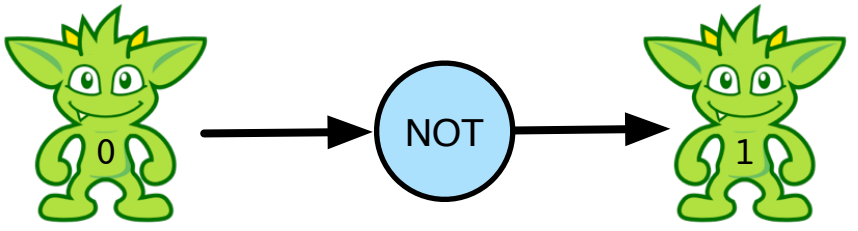


In **quantum computing**, a *qubit* can be 0, 1, or a superposition of both.



In **classical computing**, a gate takes a {0,1}* and outputs {0,1}*.

$f(0) \rightarrow 1$

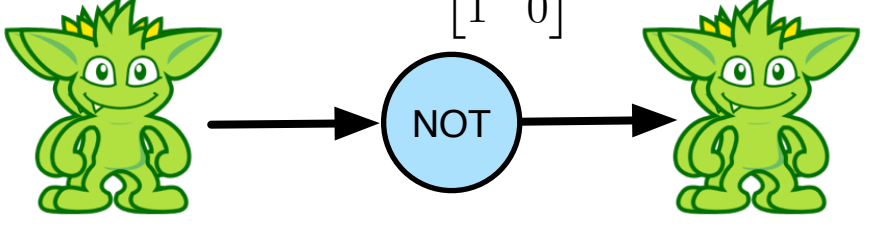


In **quantum computing**, a quantum gate takes a superposition of {0,1}* and outputs a superposition of {0,1}*.

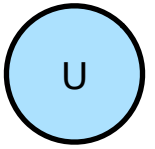
$f(0,1) \rightarrow 1,0$

Two function evaluations for the cost of one!

$\left[\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right]$ $\quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $\quad \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$

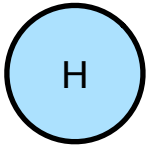


Every quantum gate is a unitary operator.

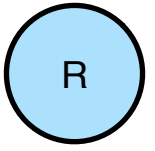


$$U \cdot U^* = I$$

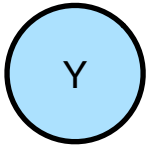
Not all gates have to be the same operator.



$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$



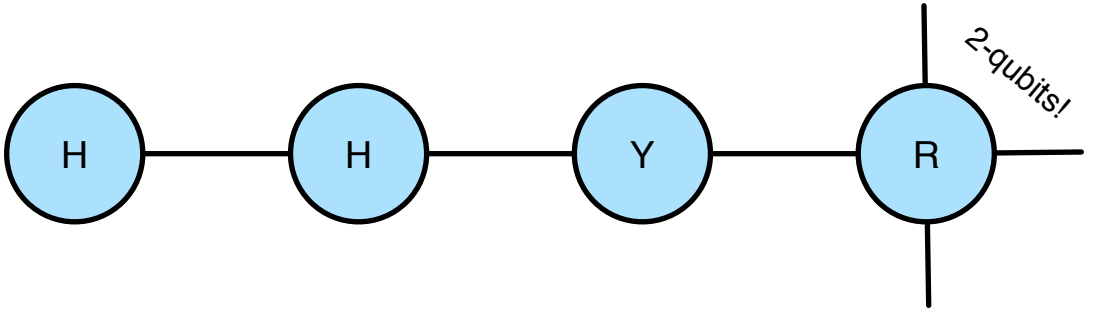
$$R = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$



$$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$$

Its all just vertices and edges!

Gates are connected by "wires."



breadth-first search

A quantum algorithm (circuit) can evaluate all possible inputs at once.

5-qubit register, where each qubit is both a 0 and 1 (thus, all 32 possible states at once).

Breadth-first memory requirements not needed!

* Read about the multi-world interpretation of quantum mechanics.

depth-first search

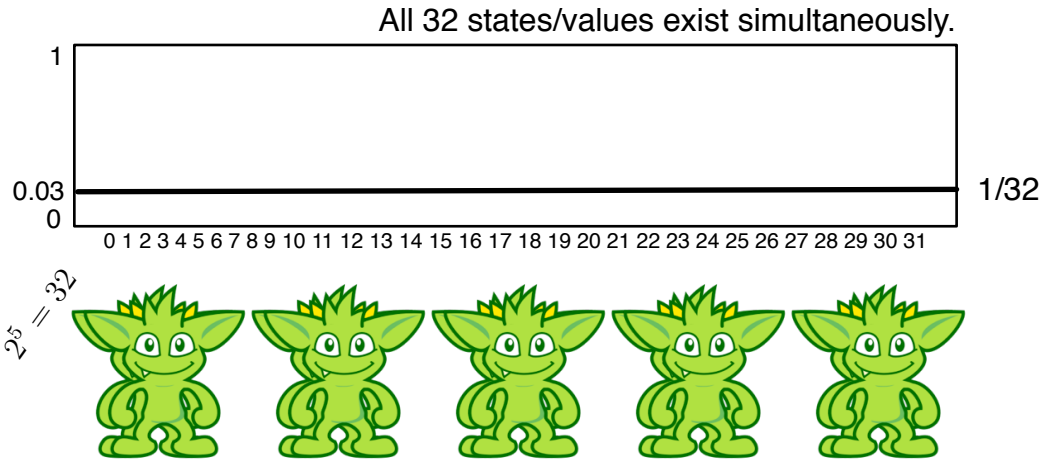
The algorithm's purpose is to guide the wave function to yield the correct answer to the problem with a high-probability (near 100%) at the time of sampling (i.e. wave function collapse).

We can never observe a superposition of outputs.
Thus, our quantum algorithms must ensure wave collapse to the correct classical state.

Depth-first speed enabled!

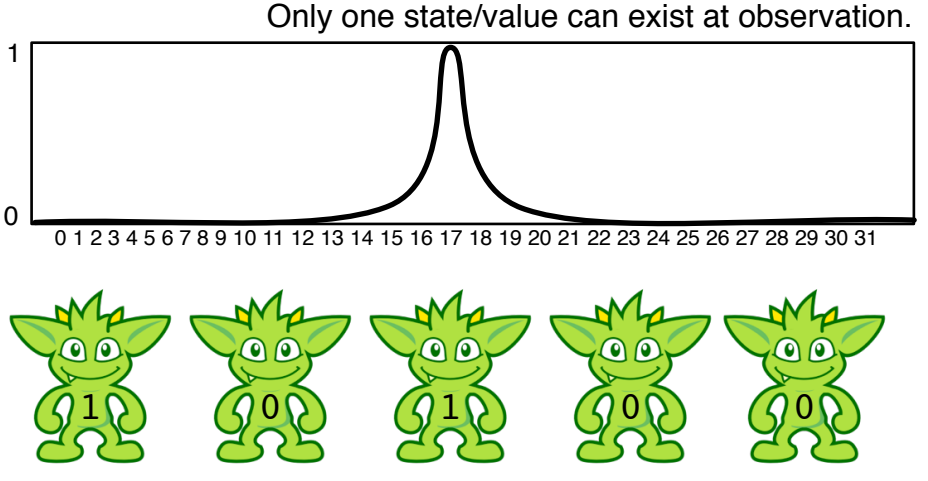
* Read about the pilot-wave interpretation of quantum mechanics.

breadth-first search



Total wave energy is split equally amongst all states/values.
If the wave function collapses at this point in time, only one number (0-31) will manifest itself in reality with each being equally likely.

depth-first search

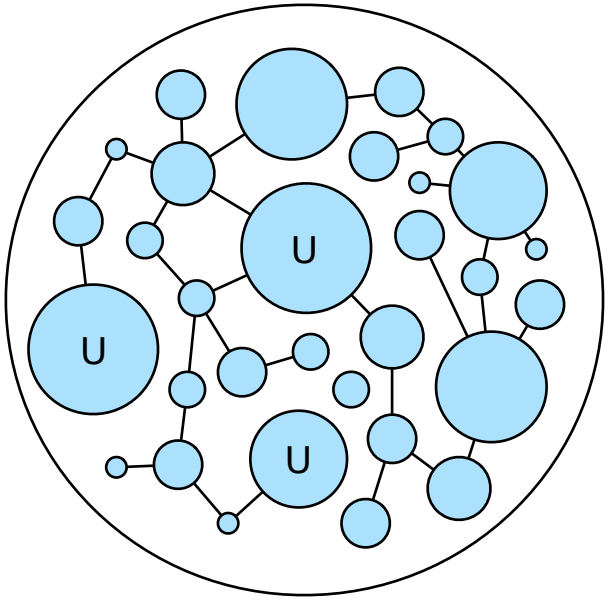


A quantum algorithm leverages simultaneous states, but can ultimately only yield one output state (a collapsed wave).
Thus, if you don't want a random answer, the algorithm must focus the wave energy on the correct solution.
Waves are controlled via constructive and destructive interference as the various parallel computations can interact with one another.

Quantum Graph Computing

Quantum Circuit

Structure



Graph

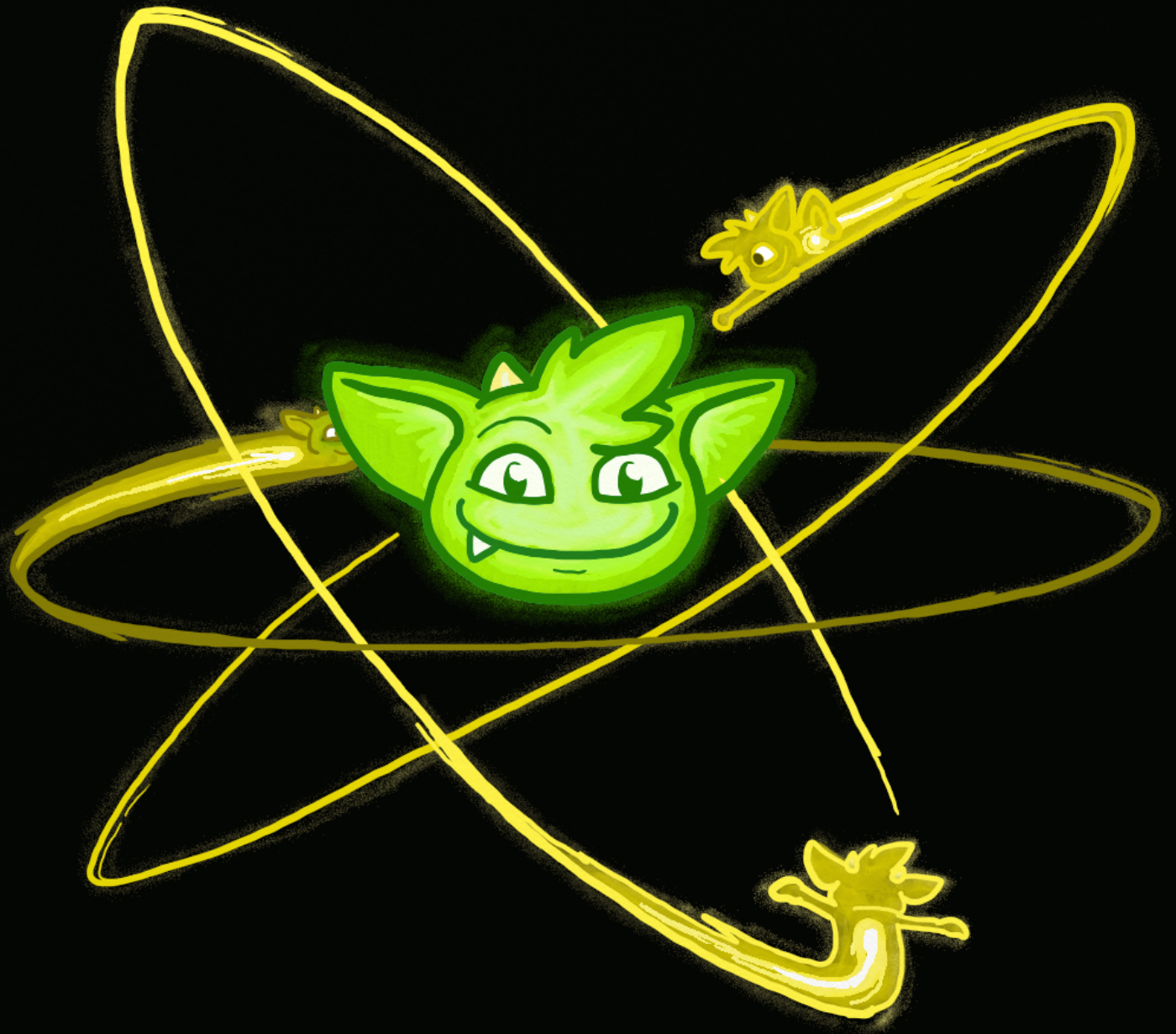
+

Quantum Wave Function

Process

Traversal

Fin.



Rodriguez, M.A., Watkins, J.H., "Quantum Walks with Gremlin," GraphDay '16, pp. 1-16, Austin, Texas, January 2016.
http://arxiv.org/abs/1511.06278