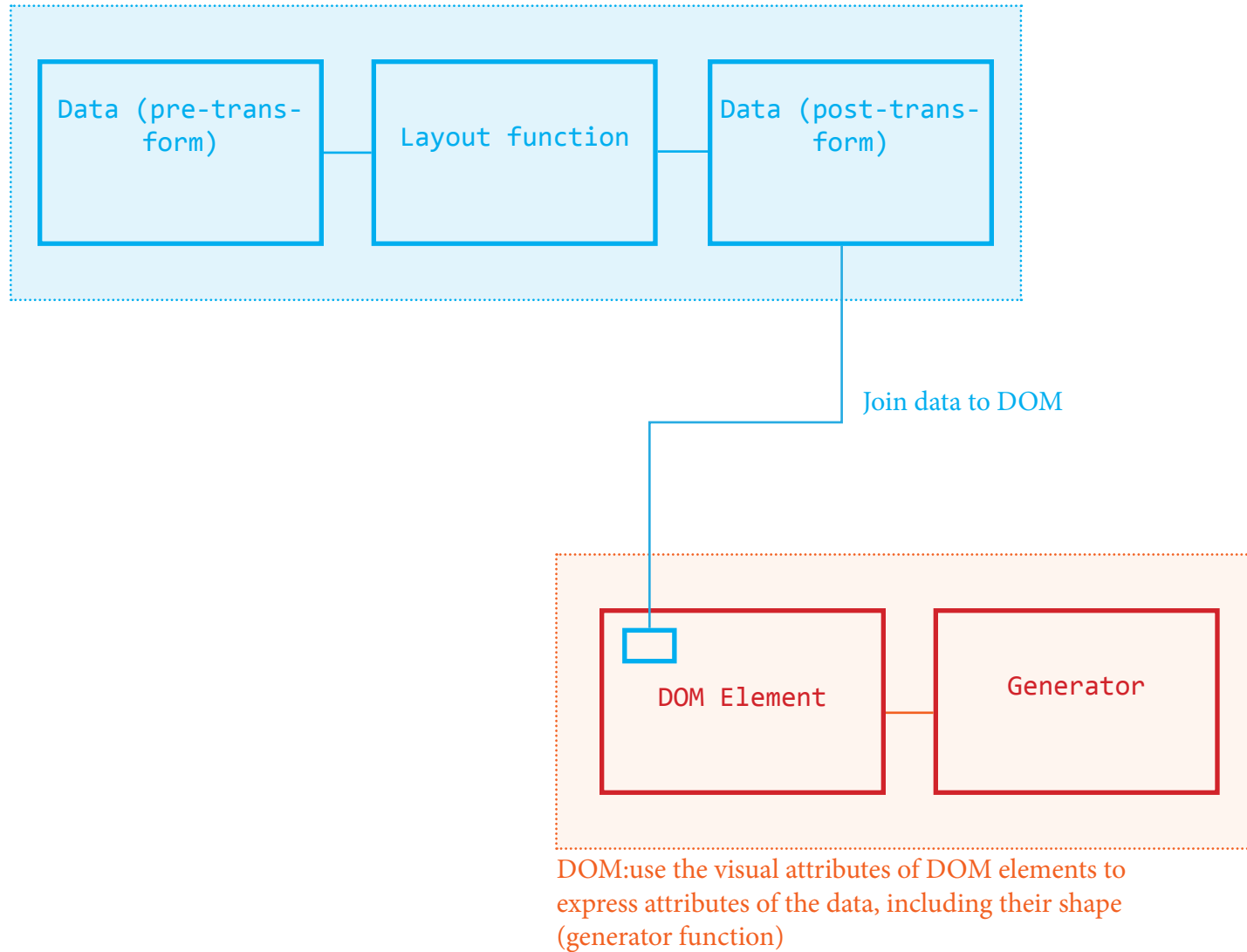
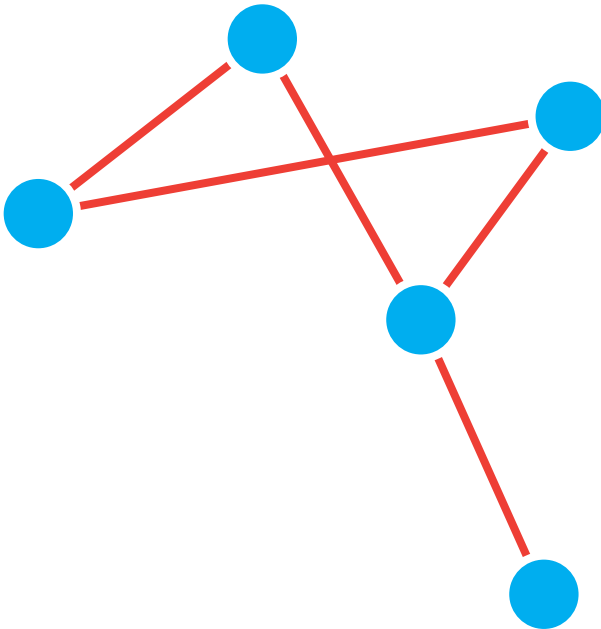


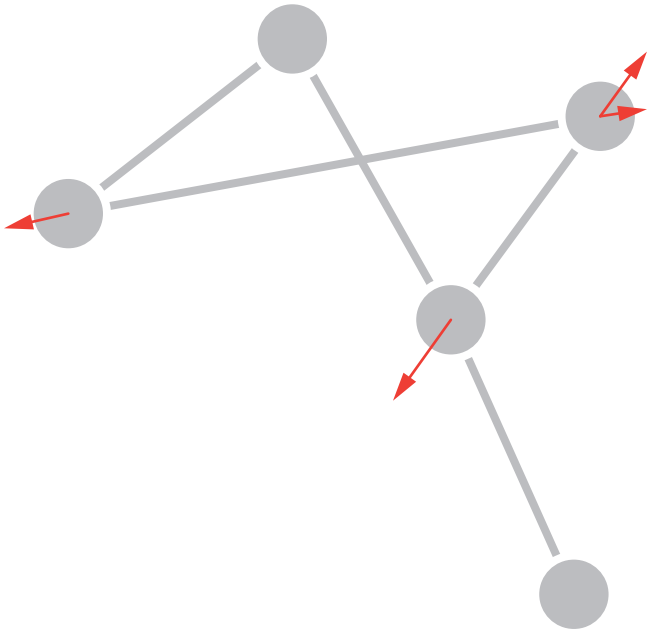
Week 11

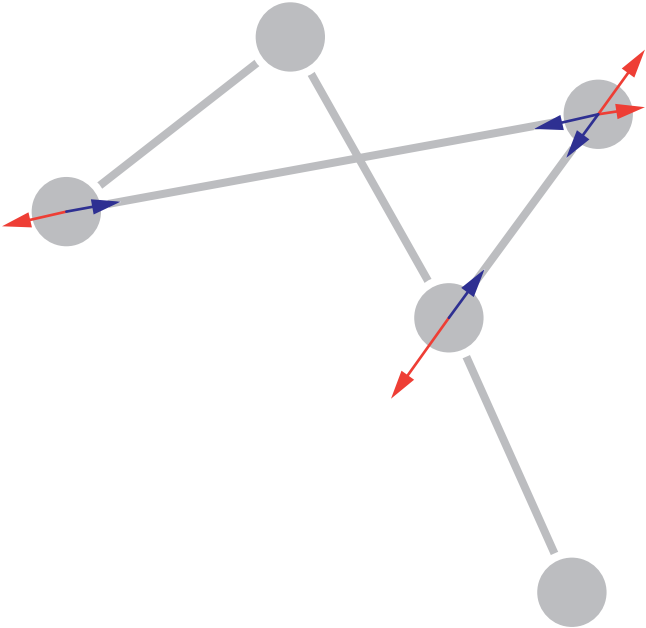
FORCE LAYOUT

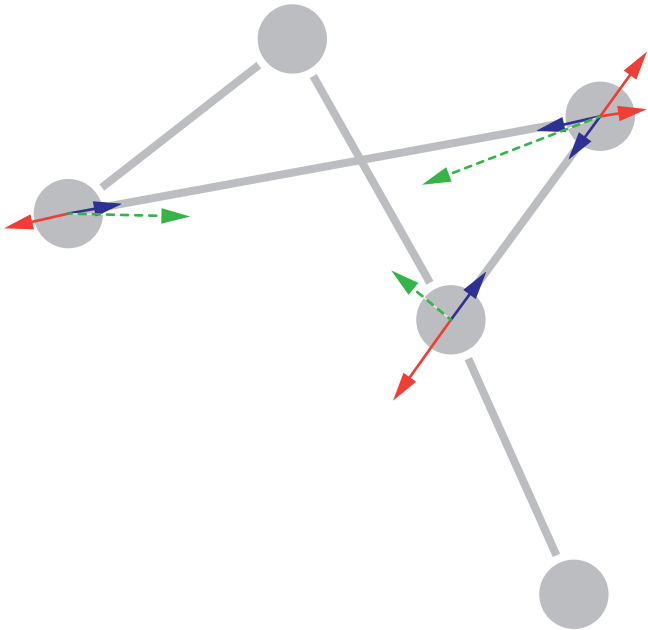
Data:
Transform data to have the right structure











HOW IS FORCE LAYOUT DIFFERENT?

1. Different data format

Force layout requires that the input data set have a very specific format.

2. Force layout modifies data in place

Force layout DOES NOT return an entirely new data set. It modifies the existing input data set in-place.

3. Force layout is dynamic

It doesn't return a fixed result, but rather incrementally moves towards it in a series of step a.k.a **ticks**.

1. DIFFERENT DATA FORMAT

Because force layout so often represents networks, it requires two sets of data inputs: nodes and links, both of which are **arrays**.

```
//nodes array
var people = [
  {
    name: "Siqu",
  },
  {
    name: "Chris",
  },
  {
    name: "Jake"
  },
  {
    name: "Christina"
  }
]
```

Must be called
"source" and "target"


```
//links array
var connections = [
  {
    source: { name: "Siqu"},
    target: { name: "Chris"}
  },
  {
    source: { name: "Jake" },
    target: { name: "Siqu" }
  }
]
```


1. DIFFERENT DATA FORMAT

An important nuance is that “source” and “target” must be the actual element in the nodes array--no substitutes!

```
var people = [  
  {  
    name: "Siqui",  
  },  
  {  
    name: "Chris",  
  },  
  {  
    name: "Jake",  
  },  
  {  
    name: "Christina",  
  }  
]
```

```
var connections = [  
  {  
    source: { name: "Siqui"},  
    target: { name: "Chris"}  
  },  
  {  
    source: { name: "Jake" },  
    target: { name: "Siqui" }  
  }  
]
```



1. DIFFERENT DATA FORMAT

An important nuance is that “source” and “target” must be the actual element in the nodes array--no substitutes!

```
var people = [  
  {  
    name: "Siqui",  
  },  
  {  
    name: "Chris",  
  },  
  {  
    name: "Jake",  
  },  
  {  
    name: "Christina"  
  }  
]
```

This does NOT work.

```
var connections = [  
  {  
    source: "Siqui",  
    target: "Chris"  
  },  
  {  
    source: "Jake",  
    target: "Siqui"  
  }  
]
```

1. DIFFERENT DATA FORMAT

One shortcut is that “source” and “target” can be array indices. During the force layout, the array indices will be replaced by the actual element.

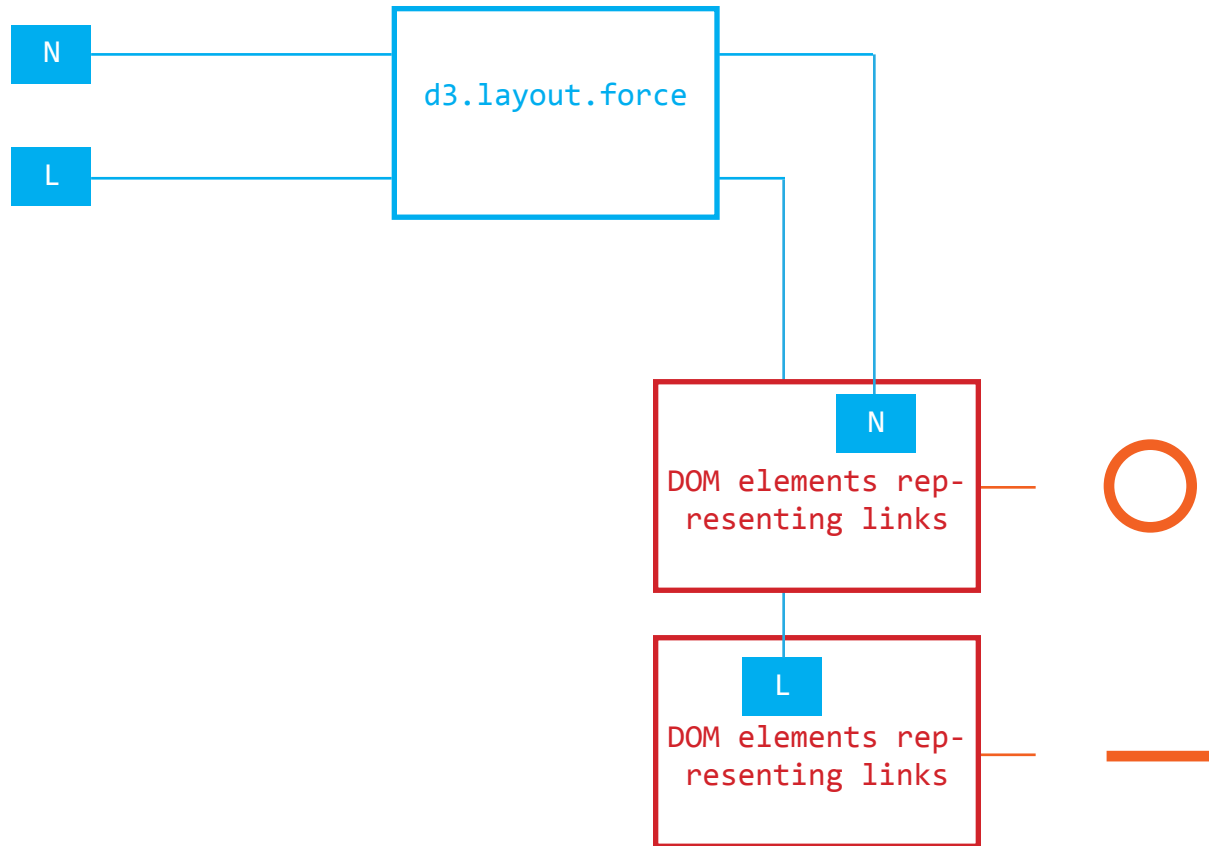
```
var people = [  
  {  
    name: "Siqi",  
  },  
  {  
    name: "Chris",1  
  },  
  {  
    name: "Jake"2  
  },  
  {  
    name: "Christina"3  
  }  
]
```

```
var connections = [  
  {  
    source: 0,  
    target: 1  
  },  
  {  
    source: 2,  
    target: 0  
  }  
]  
  
This will work  
fine.
```

HOW DO WE SET UP A FORCE LAYOUT?

```
var people = [...], connections = [...];  
  
var forceLayout = d3.layout.force()  
  .size([width,height])  
  /*  
  additional code to set up physical properties of the layout  
  */  
  .nodes(people)  
  .links(connections)
```

Two data arrays can be passed through the force layout: nodes and links



2. DATA IS MODIFIED IN PLACE

The **original nodes and links arrays** will be modified, and acquire a whole lot of new attributes:

Before layout

```
var people = [  
  {  
    name: "Siqi",  
  },  
  ...  
]
```

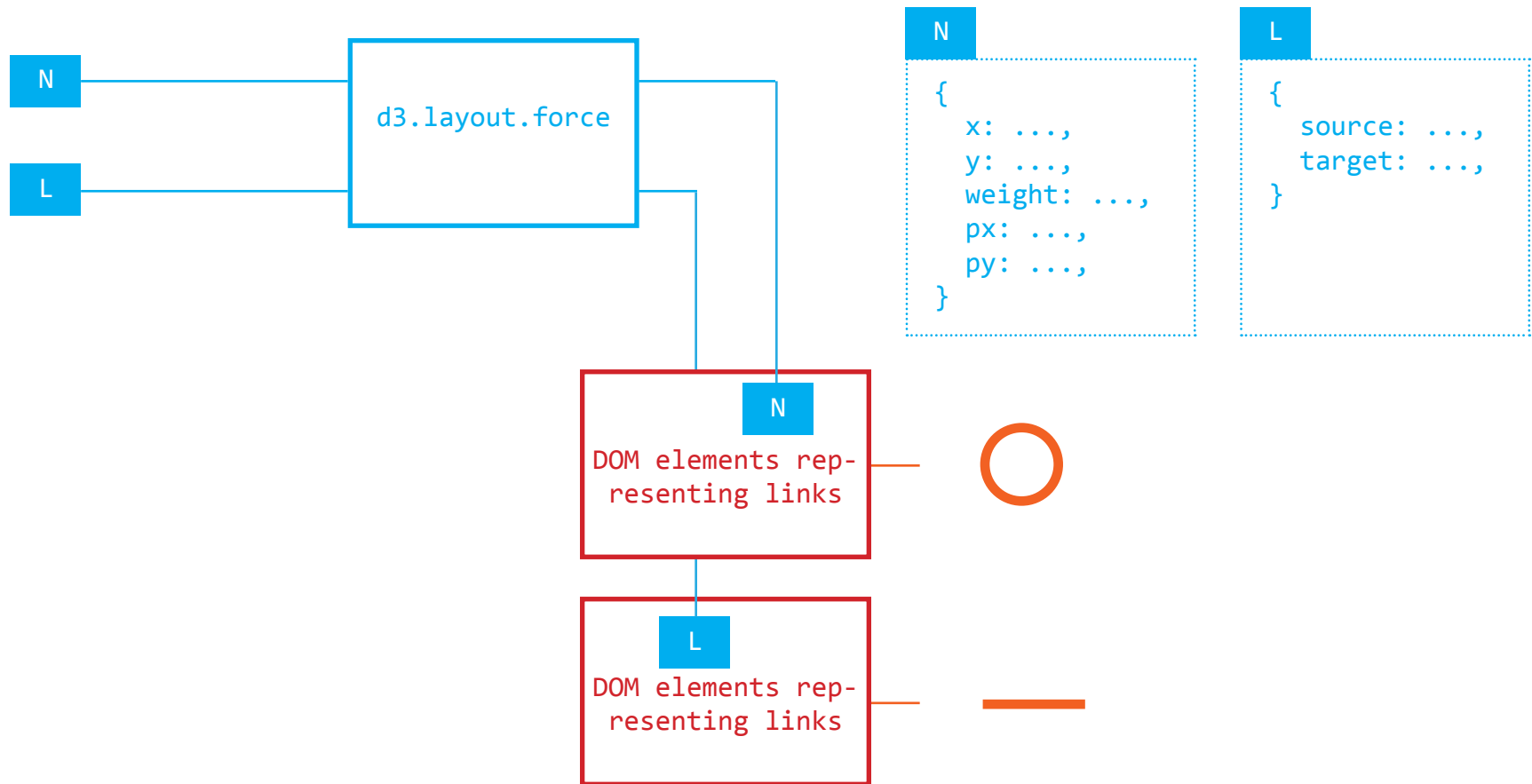
After layout

```
var people = [  
  {  
    name: "Siqi",  
    x: 234.22,  
    y: 454.12,  
    weight: 7,  
    px: 255.22,  
    py: 567.23  
  },  
]
```

2. DATA IS MODIFIED IN PLACE

At this point, we'll have enough data to visually represent these nodes and links in screen space.

Two data arrays can be passed through the force layout: nodes and links



3. FORCE LAYOUT IS DYNAMIC

Force layout doesn't return a static output. Rather, once you **start** a force layout, it will compute and try to optimize the layout in a series of steps, called **ticks**.

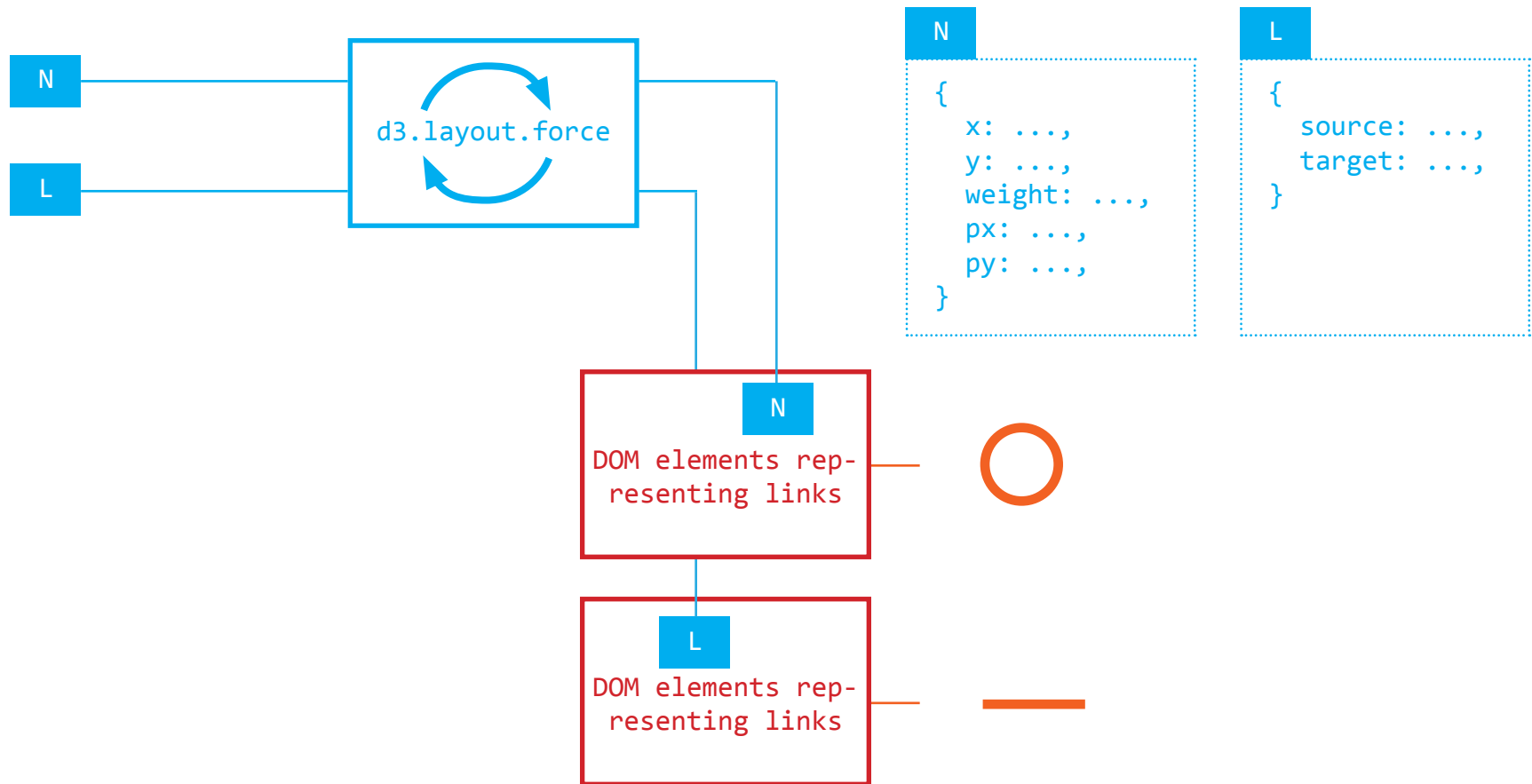
3. FORCE LAYOUT IS DYNAMIC

Force layout doesn't return a static output. Rather, once you **start** a force layout, it will compute and try to optimize the layout in a series of steps, called **ticks**.

At a certain point, the layout will **end**. But you can always manually re-start, pause, and stop the layout.

```
var forceLayout = d3.layout.force();  
  
forceLayout.start();  
forceLayout.stop();  
forceLayout.on('tick', function(e){  
    //this block of code runs many, many times, until the layout ends  
});
```

Computation is iterative. Nodes and links are optimized through a series of “ticks”.



HOW DOES THIS WORK IN PRACTICE?

How do we integrate force layout into our enter/exit/update DOM workflow?

HOW DOES THIS WORK IN PRACTICE?

Step 1: create force layout, and set up its physical properties (we'll review what these are in a minute). We probably want to assign it to a global variable.

```
var forceLayout = d3.layout.force()  
    .size(...)  
    .gravity(...)  
    .charge(...)  
    .linkDistance(...)  
    .friction(...)  
    .linkStrength(...);
```

HOW DOES THIS WORK IN PRACTICE?

Step 2: feed nodes and links array into the layout (assuming they are in the right format);

```
forceLayout
  .nodes(nodesArray)
  .links(linksArray);
```

HOW DOES THIS WORK IN PRACTICE?

Step 3: bind nodesArray and linksArray to DOM elements, creating new ones if necessary using `.enter()`

```
forceLayout
  .nodes(nodesArray)
  .links(linksArray)
```

```
var circles = svg.selectAll('circle')
  .data(nodesArray)
  .enter()
  .append('circle')
```


HOW DOES THIS WORK IN PRACTICE?

Step 4: setting visual attributes for the DOM elements should take place within each tick event, because data is being re-computed at each tick.

```
forceLayout
  .nodes(nodesArray)
  .links(linksArray)
  .on('tick', onTick);
```

```
var circles = svg.selectAll('circle')
  .data(nodesArray)
  .enter()
  .append('circle')
```

```
function onTick(e){
  circles
    .attr('cx',function(d){
      return d.x;
    })
    .attr('cy',function(d){
      return d.y;
    })
}
```




HOW DOES THIS WORK IN PRACTICE?

Step 5: Finally, the layout only starts when you call `.start()` on the force layout

```
forceLayout
  .nodes(nodesArray)
  .links(linksArray)
  .on('tick', onTick)
  .start();
```

```
var circles = svg.selectAll('circle')
  .data(nodesArray)
  .enter()
  .append('circle')
```

```
function onTick(e){
  circles
    .attr('cx',function(d){
      return d.x;
    })
    .attr('cy',function(d){
      return d.y;
    })
}
```



HOW DOES THIS WORK IN PRACTICE?

Bonus: call `force.drag` on the DOM selection will enable drag behavior.

```
forceLayout
  .nodes(nodesArray)
  .links(linksArray)
  .on('tick', onTick)
  .start();
```

```
var circles = svg.selectAll('circle')
  .data(nodesArray)
  .enter()
  .append('circle')
  .call(forceLayout.drag);

function onTick(e){
  circles
    .attr('cx',function(d){
      return d.x;
    })
    .attr('cy',function(d){
      return d.y;
    })
}
```

WHAT ELSE?

Force layout is not deterministic i.e. the same data and same layout will not produce the same visual output.

Force layout assigns the initial positions of the nodes randomly.

BUT, you can manually assign the initial x- and y- positions for the nodes array. This will in fact improve performance.

LET'S CREATE OUR OWN!

ADVANCED USE CASE 1: MULTIPLE FOCI

Refer to 11_2 script-4 for an example. In particular, we can change how we write the tick function to customize behavior at each step.

ADVANCED USE CASE 2: COLLISION DETECTION

See this example:

<http://bl.ocks.org/mbostock/3231298>

A special case of this is a dorling cartogram:

<http://bl.ocks.org/mbostock/4055892>