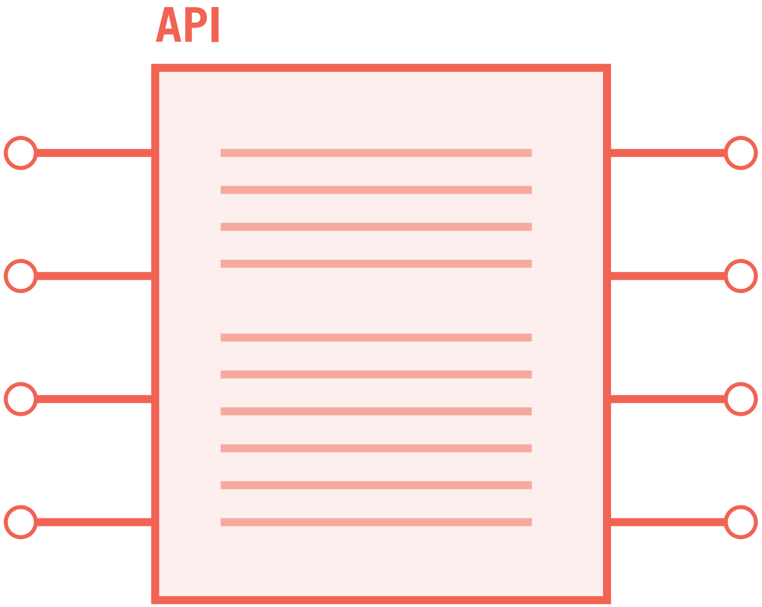
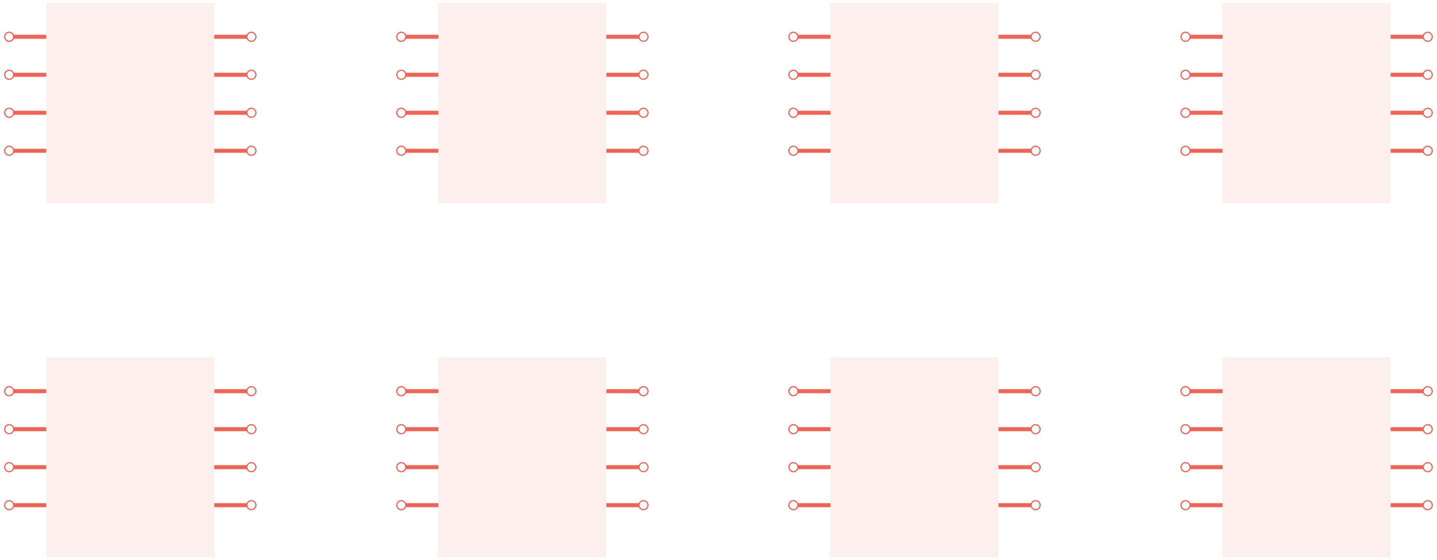
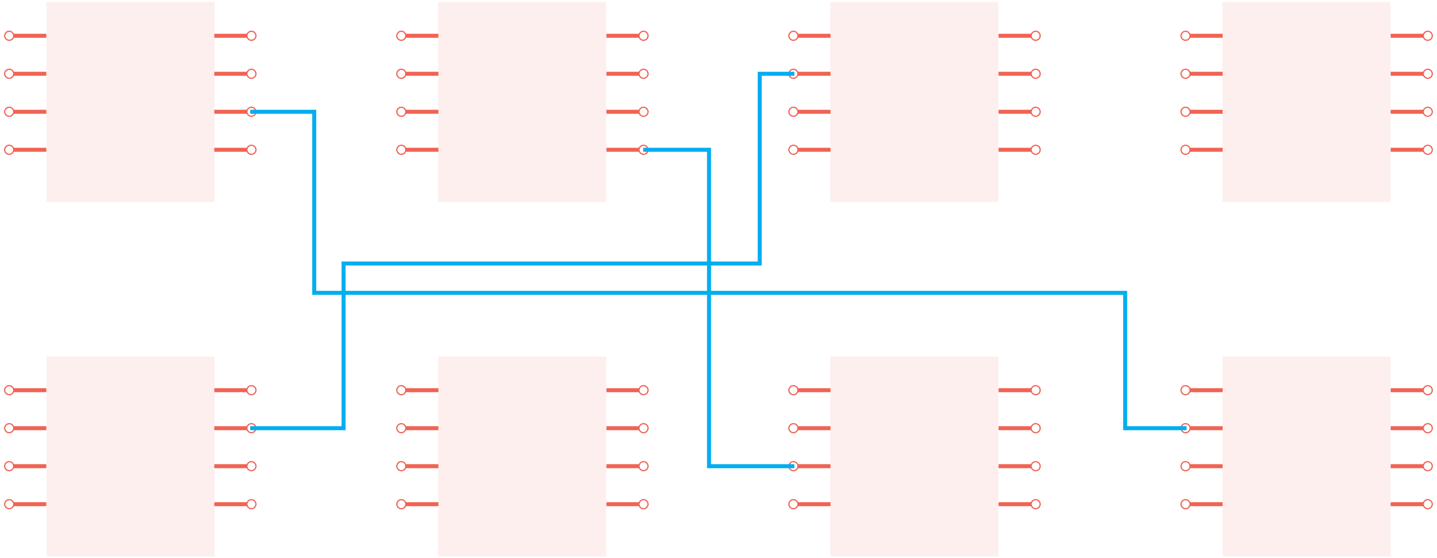
Event Handling

Goal of the Class

- Refresh memory of event handling in d3
- The special case of interacting with line type graphs
- Custom events using `d3.dispatch()`


API





Native event handling: `selection.on( )`

```
selection.on(eventType, listener);
```

`eventType:` a DOM event such as `“mouseenter”`, `“mouseleave”`, `“click”` etc.

`listener`: a function; anytime the event is fired, this function is invoked with current datum `d` and index `i`

`this`: the DOM element that emitted the event

Special consideration 1: `mouseenter` vs. `mouseover`

http://bl.ocks.org/mbostock/5247027

Special consideration 2: multiple listeners

```
selection1.on(‘click’, function(d,i){
   console.log(‘First function’);
});
selection1.on(‘click’, function(d,i){
   console.log(‘Second function);
});
```

Special consideration 2: multiple listeners

```
selection1.on(‘click.first’, function(d,i){
   console.log(‘First function’);
});
selection1.on(‘click.second’, function(d,i){
   console.log(‘Second function);
});
```

`d3.mouse( )` **and** `d3.event`

`d3.event` provides specific information about each event within its listener

`d3.mouse(node)` returns the [x, y] coordinate of the mouse within mouse events, relative the DOM node.

The special case of interacting with line type graphs

```
svg.append(‘path’)
    .datum(array)
    .on(‘mousemove”,function(d){
        ...
    });
```

The special case of interacting with line type graphs

```
svg.append(‘path’)
    .datum(array)
    .on(‘mousemove”,function(d){

        ---
        1. using d3.mouse( ), discover current mouse position
        2. use mouse position to find the nearest point in the array
        3. visualize tooltip using the nearest point
        ---

    });
```

d3.bisector()

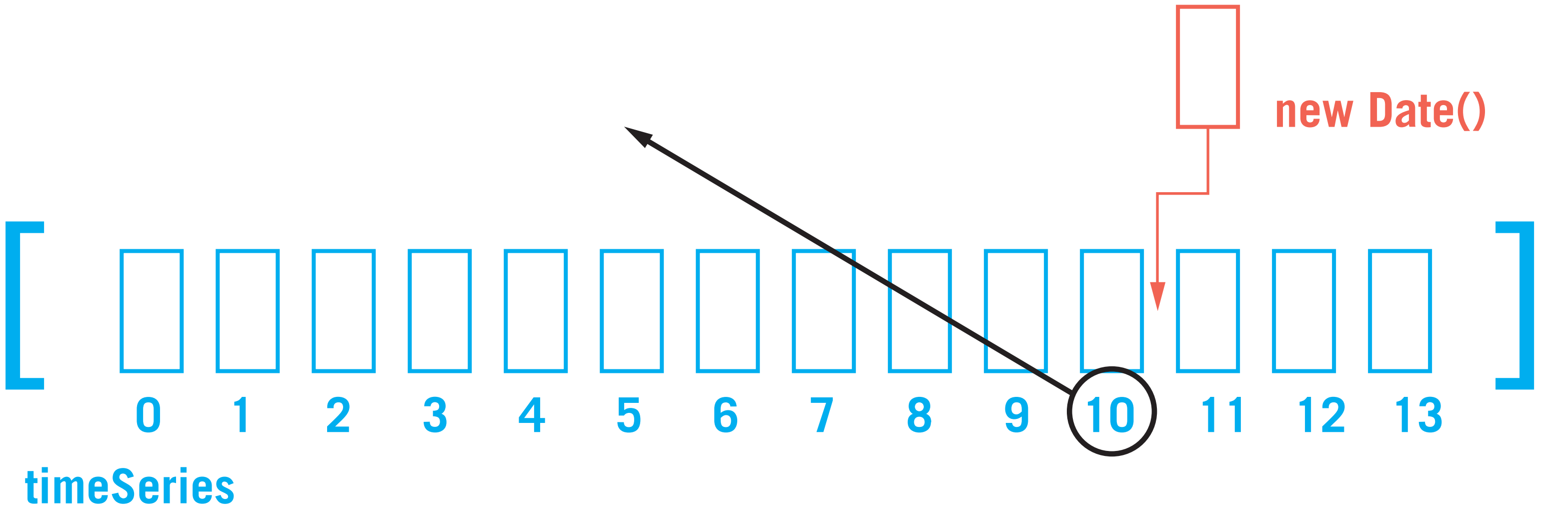
Returns a **bisect** function

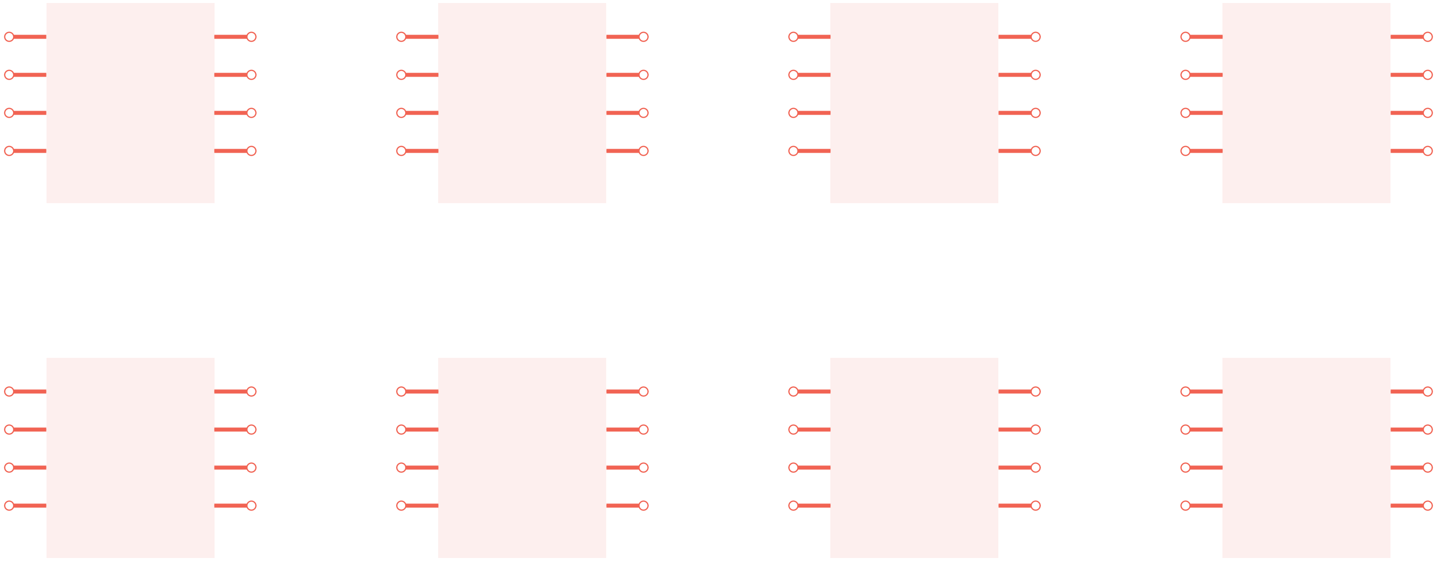
Assuming array is **sorted**, the bisect function returns an insertion point index.

d3.bisector()

```
var bisect = d3.bisector(function(d){return
d.x}).left;

var index = bisect(timeSeries, new
Date(2012,2,3));
```





Advanced events: `d3.dispatch( )`

- Allows us to design custom events
- Emitter / subscriber model

Basic usage of `d3.dispatch( )`

1. Create an event dispatcher, and register some custom events

```
var dispatcher1 = d3.dispatch(‘customEvent1’,
‘customEvent2’);
```

2. This dispatcher allows us to emit custom events with any arbitrary arguments

```
dispatcher1.customEvent1(arg1, arg2, arg3...);
```

3. Subscribe to custom events

```
dispatcher1.on(‘customEvent1’, function(arg1,
arg2, arg3){
    ...
});
```

Basic usage of `d3.dispatch( )`

1. Create an event dispatcher, and register some custom events

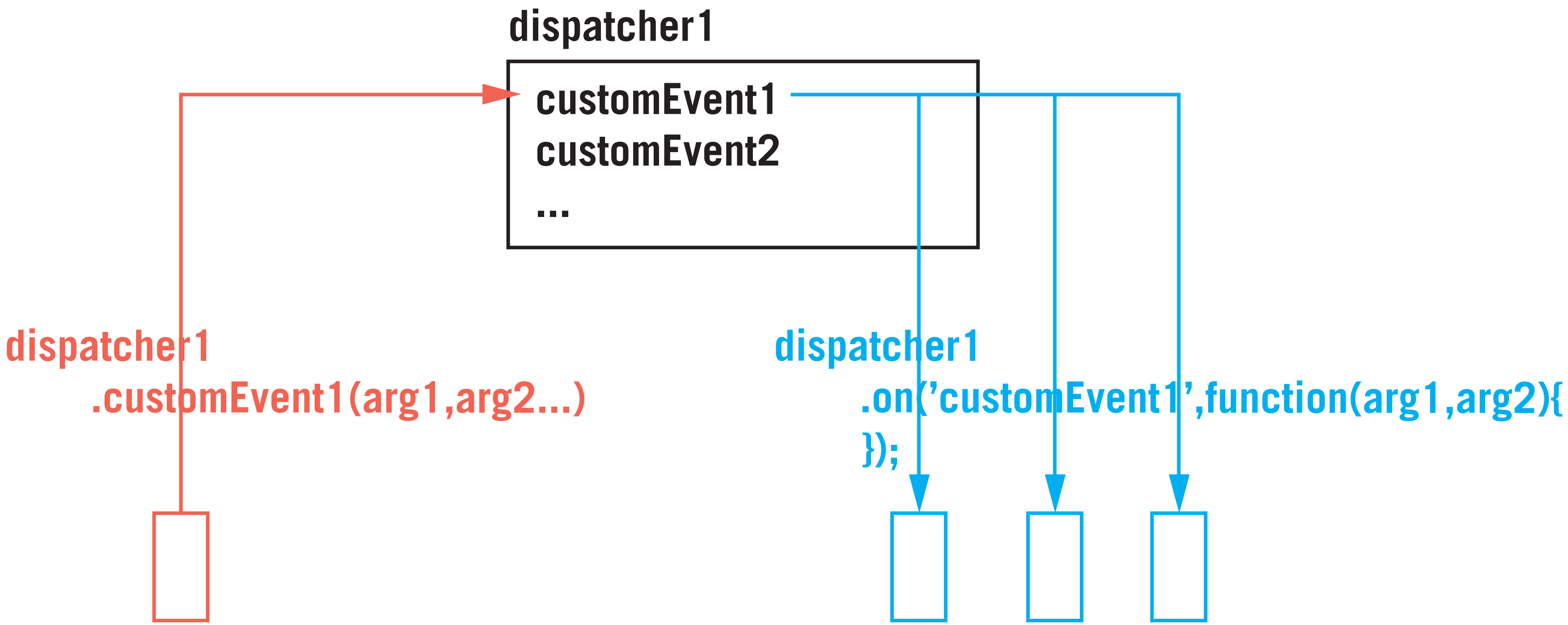
```
var dispatcher1 = d3.dispatch(‘customEvent1’,
‘customEvent2’);
```

2. This dispatcher allows us to emit custom events with any arbitrary arguments

```
dispatcher1.customEvent1(arg1, arg2, arg3...);
```

3. Subscribe to custom events

```
dispatcher1.on(‘customEvent1’, function(arg1,
arg2, arg3){
    ...
});
```

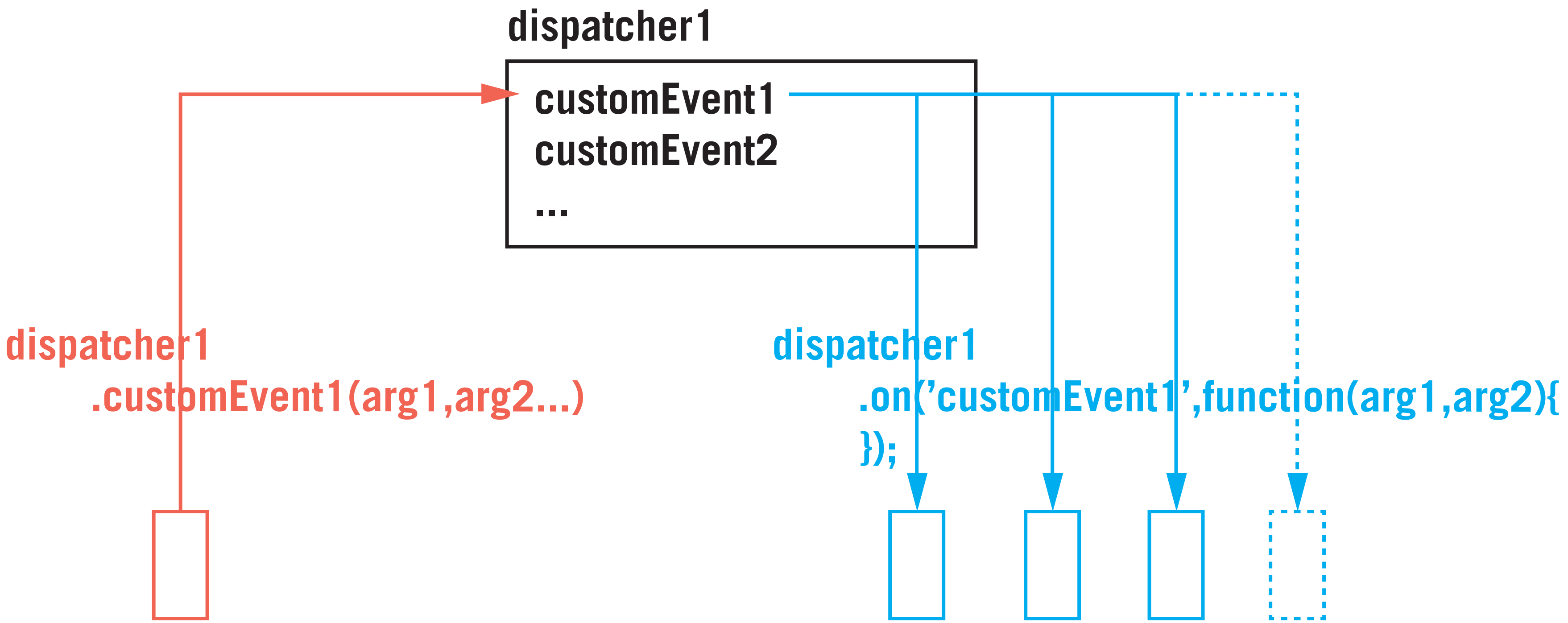


dispatcher1

customEvent1
customEvent2
...

```
dispatcher1
    .customEvent1(arg1,arg2...)
```

```
dispatcher1
    .on(’customEvent1’,function(arg1,arg2){
    });
```



dispatcher1

customEvent1
customEvent2
...

```
dispatcher1
    .customEvent1(arg1,arg2...)
```

```
dispatcher1
    .on('customEvent1',function(arg1,arg2){
    });
```