Advanced Event Handling

Goal of the Workshop

- Become familiar with `d3.dispatch( )` for handling events and interactions between loosely coupled components
- Combine `d3.dispatch()` with previously developed reusable modules

Quick Review of DOM Event Handling in d3

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

Events are triggered by DOM elements, and listeners react to these events.

https://developer.mozilla.org/en-US/docs/Web/Events

Quick Review of DOM Event Handling in d3

```
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

Quick Review of DOM Event Handling in d3

```
selection.on(eventType, function(d,i){
    console.log(this);
});
```

*For each eventType, we can only bind one event listener**

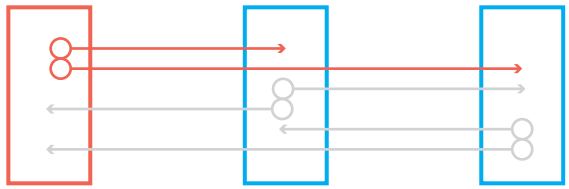
** We can get around this by "**namespacing**" events

Quick Review of DOM Event Handling in d3

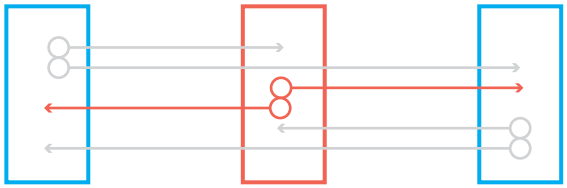
```
selection.on('click.random', function(d,i){
    console.log(this);
});
selection.on('click.random2', function(d,i){
    ...
});
```

Let's quickly refresh our memory by doing a quick exercise.

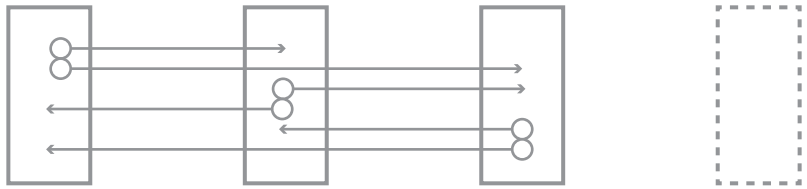
These can be modules,
UI element etc.

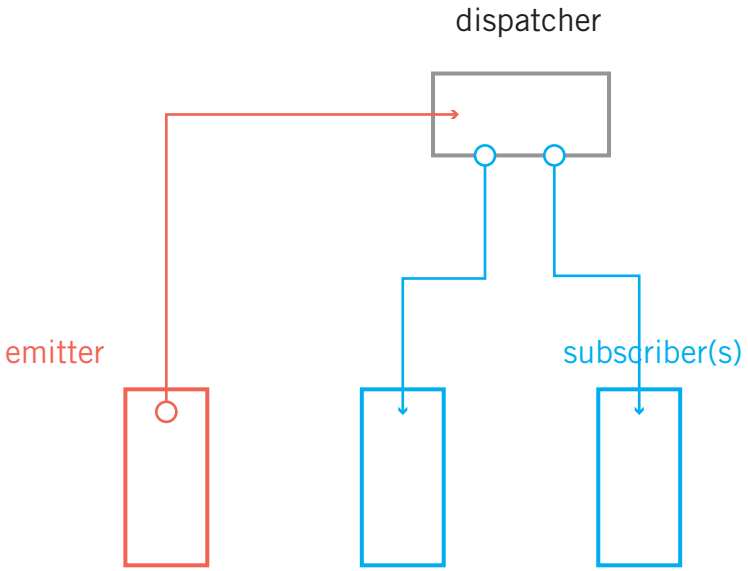


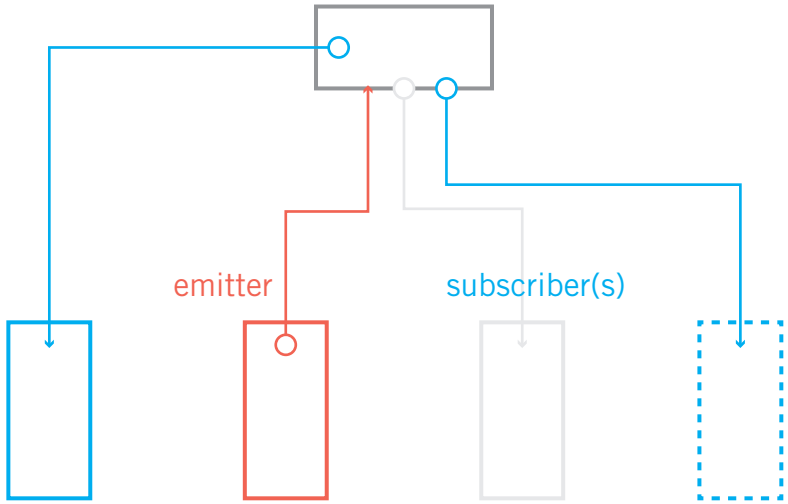
selection1.on(...)



selection2.on(...)




dispatcher
emitter
subscriber(s)


emitter
subscriber(s)

Advanced events: `d3.dispatch( )`

- Allows us to design custom events
- Emitter / subscriber model allows components to be loosely coupled -- more flexible and scalable

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');
```

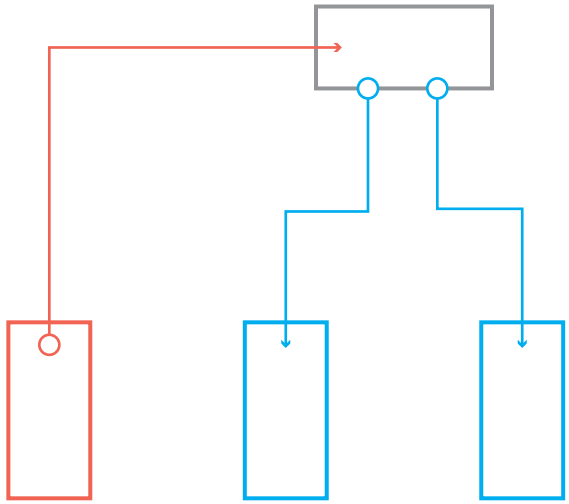
2. This dispatcher allows us to emit custom events with any arbitrary arguments

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

3. Subscribe to custom events

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

dispatcher = d3.dispatch('event1','event2'...)

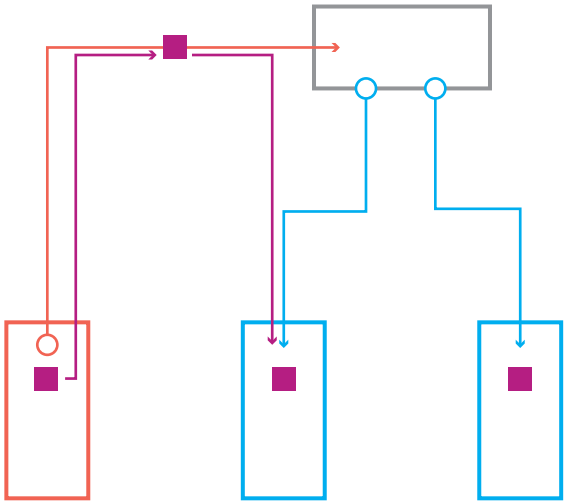


dispatcher.event1()

dispatcher.on('event1',function(..){

})

dispatcher = d3.dispatch('event1','event2'...)

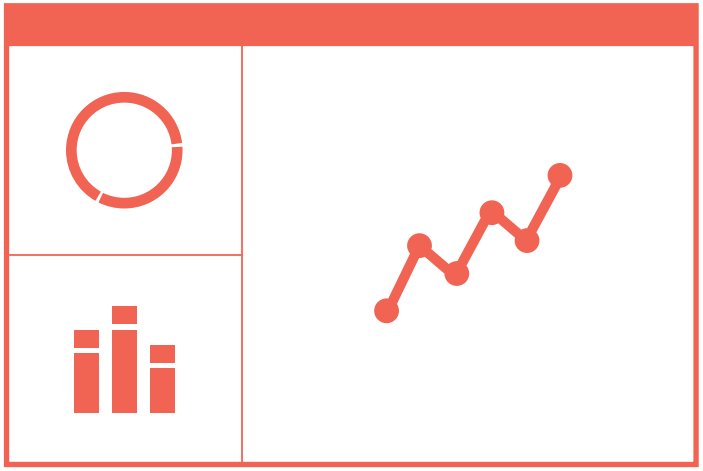


dispatcher.event1(arg1...)

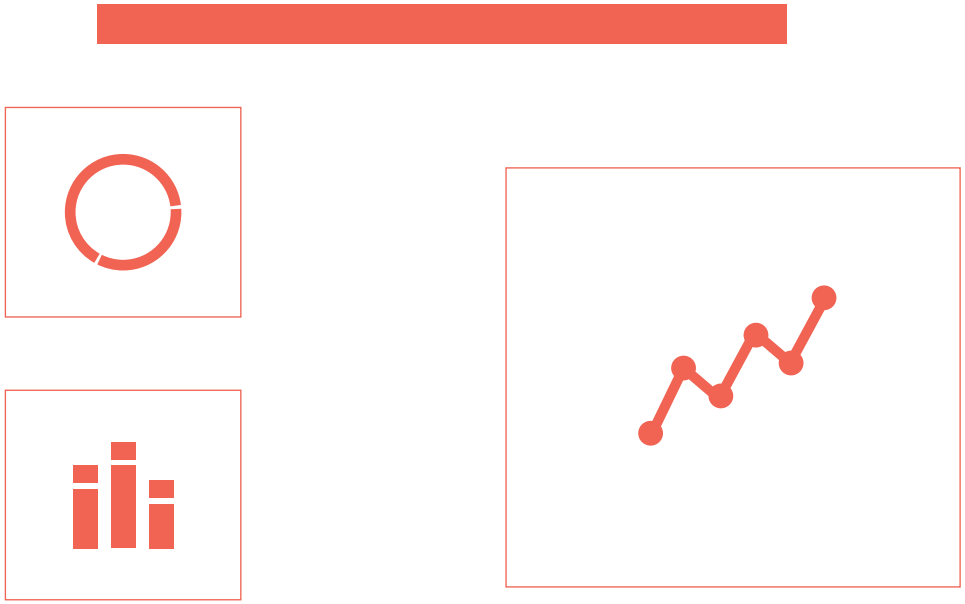
dispatcher.on('event1',function(arg1){

})

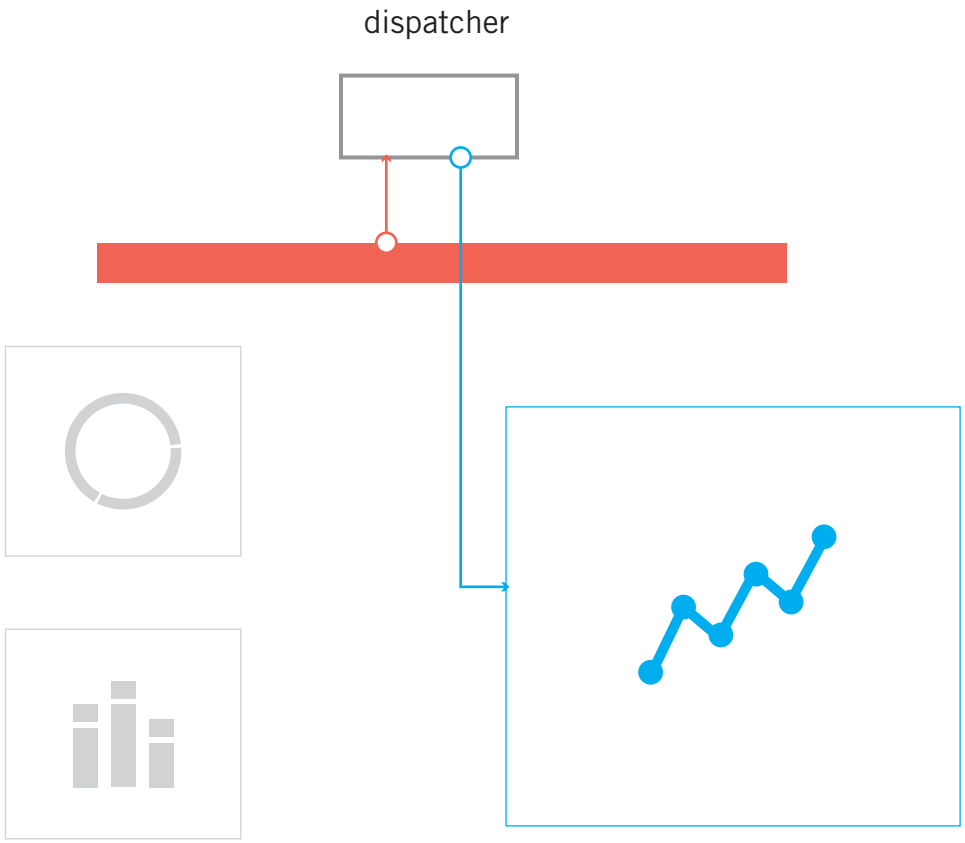
How does d3.dispatch( ) work with modular development?



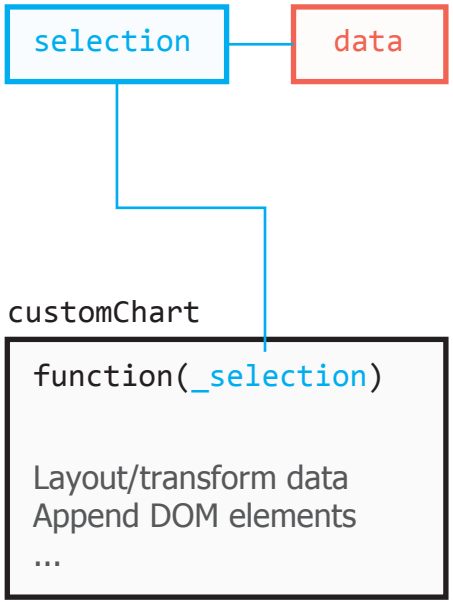
How does d3.dispatch( ) work with modular development?



Case 1: UI and Modules



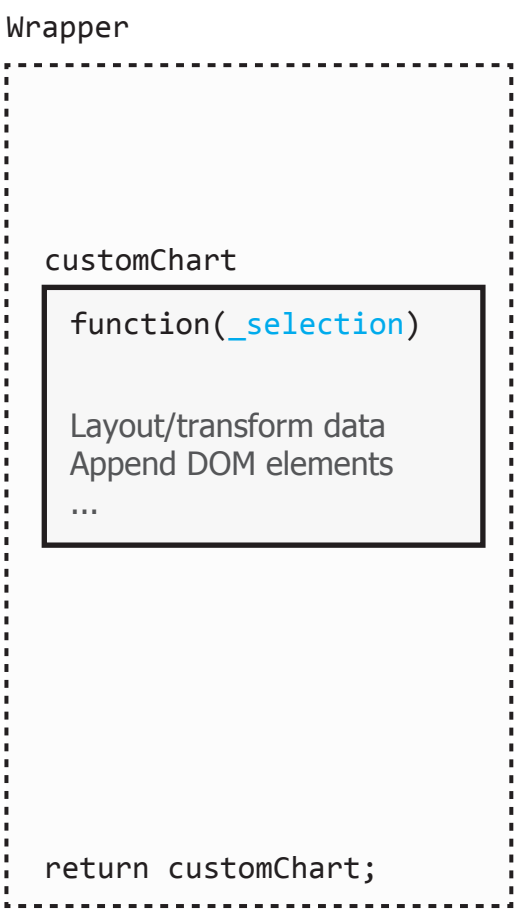
Review: Basic Thought Process of Developing a Module




```
selection
    .datum(...)
    .call(customChart);
```

But this implementation of `customChart` isn't configurable! i.e. we can't change its attributes!

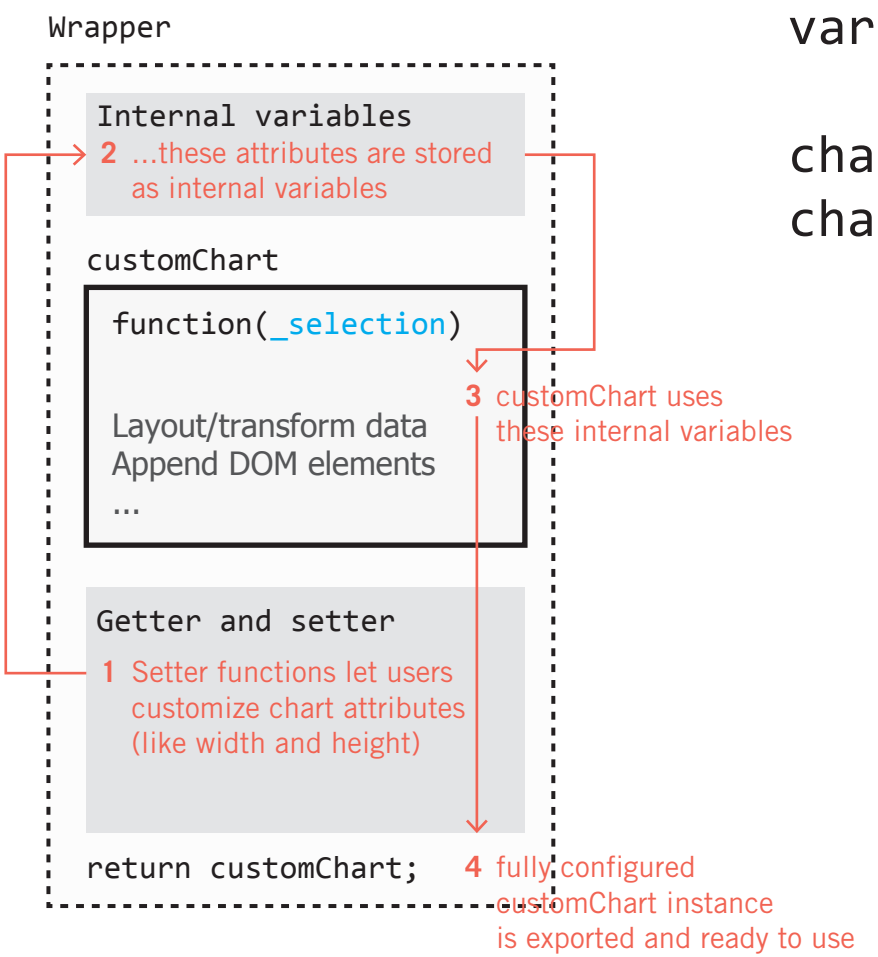
Review: Basic Thought Process of Developing a Module



The solution: wrap `customChart` in a `wrapper` function.

Use the `wrapper` function to configure, and export an instance of, `customChart`

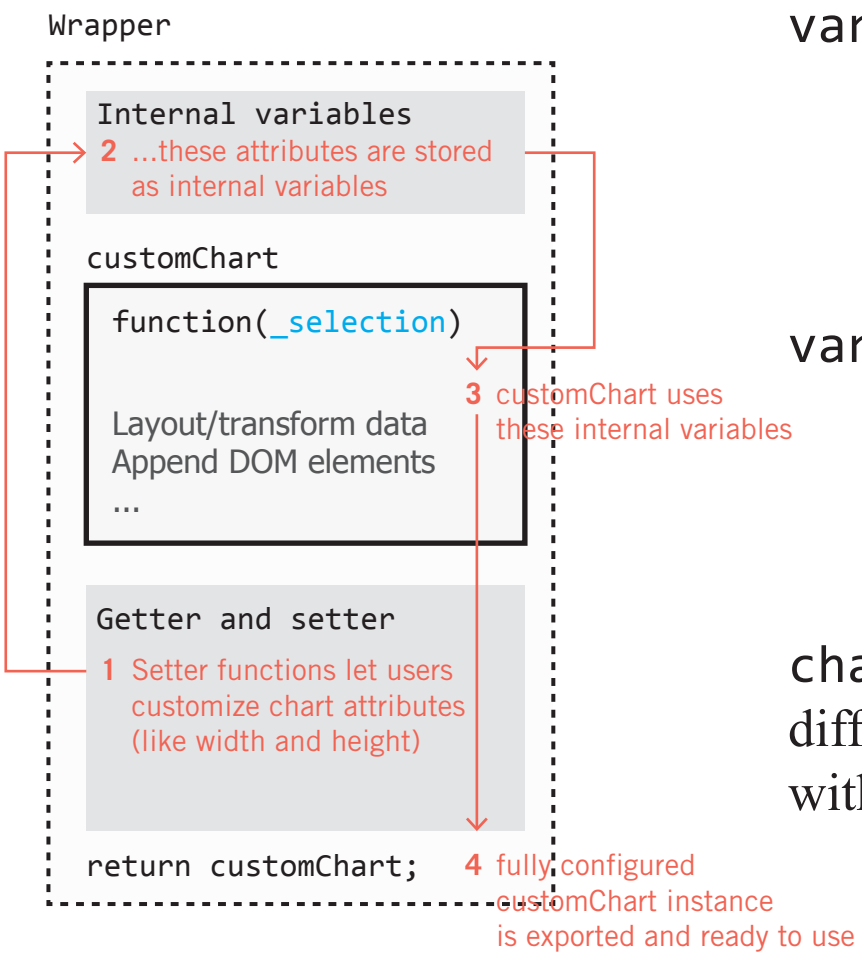
Review: Basic Thought Process of Developing a Module




```
var chart1 = wrapper();

chart1.width(400);
chart1.height(600);
```

Review: Basic Thought Process of Developing a Module

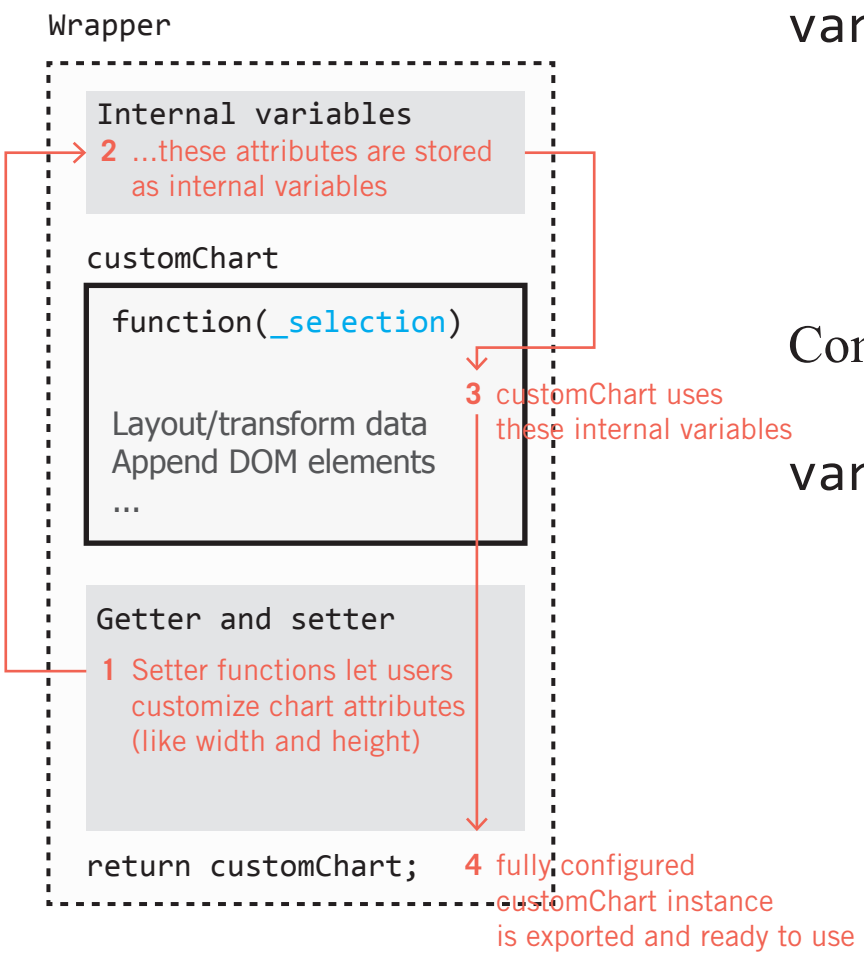



```
var chart1 = wrapper()
    .width(400)
    .height(600)
    ...

var chart2 = wrapper()
    .width(300)
    .height(400)
    ...
```

`chart1` and `chart2` are two different instances of `customChart`, with different configurations.

Review: Basic Thought Process of Developing a Module




```
var chart1 = wrapper()
    .width(400)
    .height(600)
    ...
```

Compare this to something familiar:

```
var axisX = d3.svg.axis()
    .scale(...)
    .ticks(...)
```

Additional Considerations: Updates to the Chart

```
var chart1 = wrapper()
    .width(400)
    .height(600);
d3.select('div.chart')
    .datum(someData)
    .call(chart1);

chart1.width(500);
d3.select('div.chart')
    .datum(newData)
    .call(chart1);
```

If chart configuration changes, or new data is bound, we should be able to make updates by simply calling the function again.

Our implementation of the module should support this behavior.

Case 2: Communication between modules