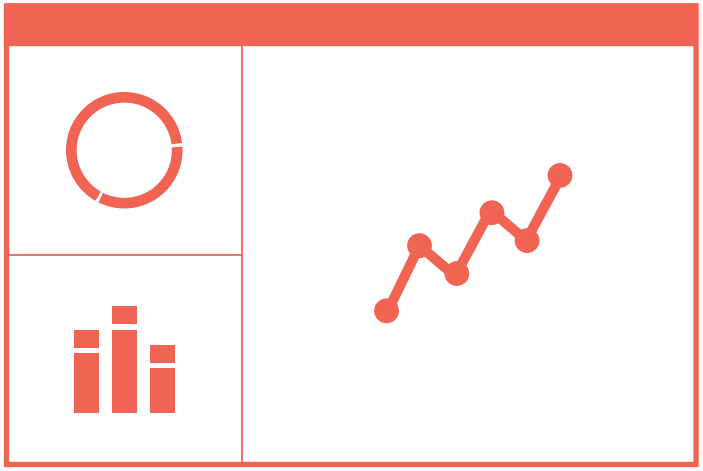
Brushing, Crossfilter & MVC Paradigm

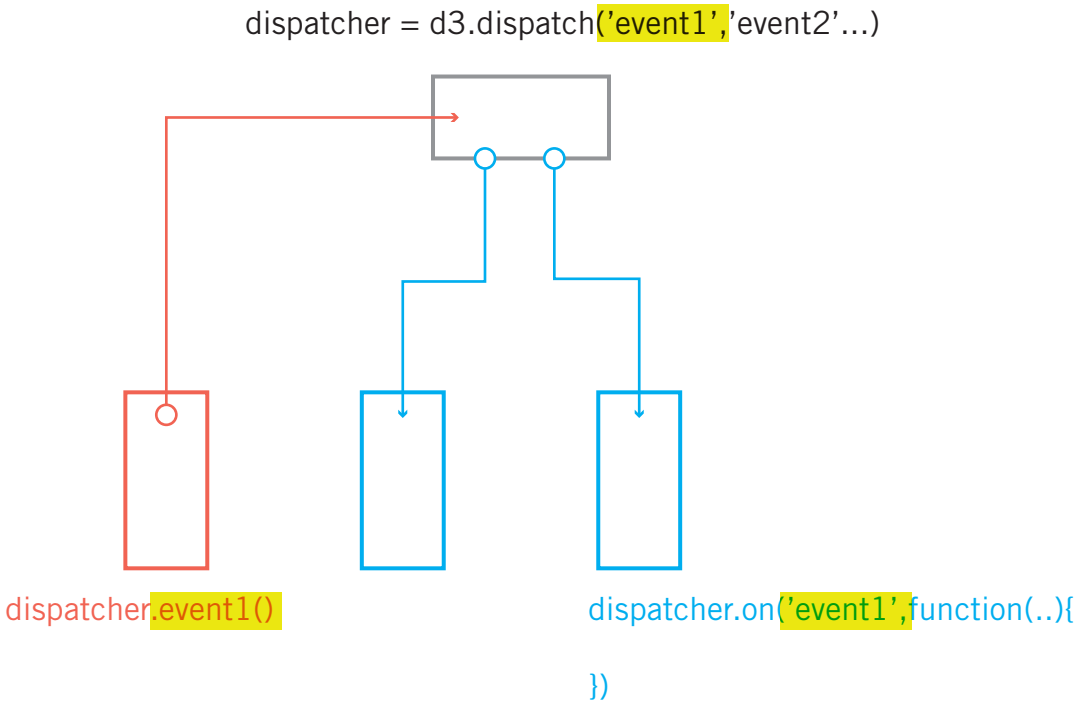
Goal of the Workshop

- Further familiarize with event dispatching
- Become proficient with brushing interaction
- Use crossfilter to manipulate large datasets
- Putting it together: model / view / controller

Review overall motivation: an exploratory visualization with multiple connected components

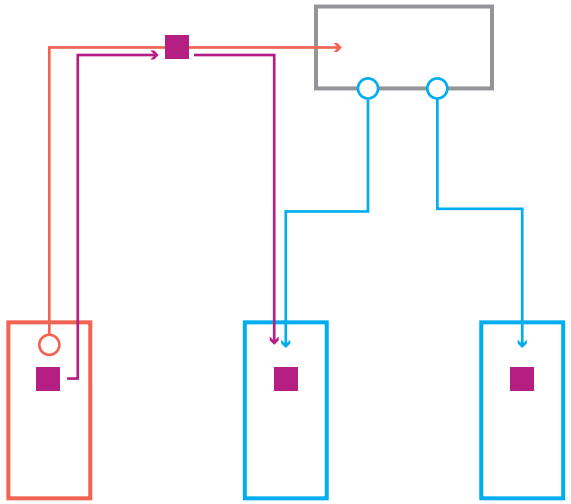


Quick review of dispatch



Quick review of dispatch

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



dispatcher.event1(arg1...)

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

})

Task 1: d3.dispatch() warm-up

- Note the use of <input> elements and the "change" event they trigger
- Note how Bootstrap styles <input> elements
  http://v4-alpha.getbootstrap.com/components/input-group/

Brushing interaction

An interaction pattern that nicely dovetails with filtering data

Implemented in d3 as a reusable module

Creating a brush

First, instantiate an instance of a brush module

```
var newBrush = d3.svg.brush( )
    .x( scaleX );
```

Then, call the module on a new selection

```
plot.append('g')
    .call( newBrush );
```

Creating a brush (cont'd)

This creates a number of (at first invisible) DOM elements:

```
<g class= 'brush' >
   <rect class = 'background' ></rect>
   <rect class = 'extent' ></rect>
   ...
</g>
```

They are invisible because they have initial width/height set to 0.

Brush interaction

When you click, drag or move the brush extent, several events fire in sequence:

```
brushstart
brush
brushend
```

You can capture these events using event listeners.

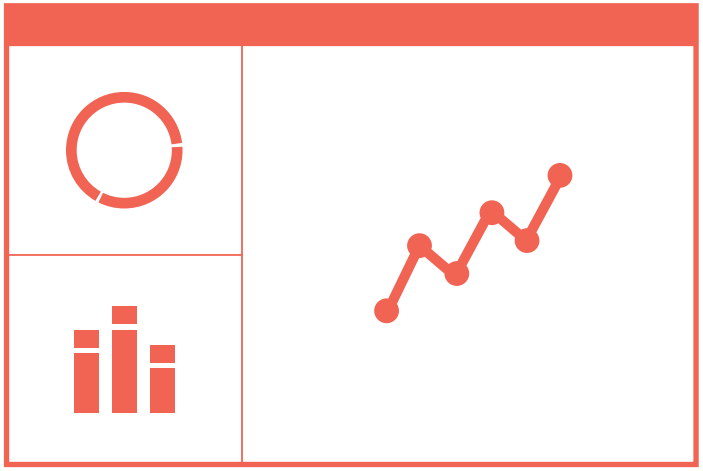
Brush interaction (cont'd)

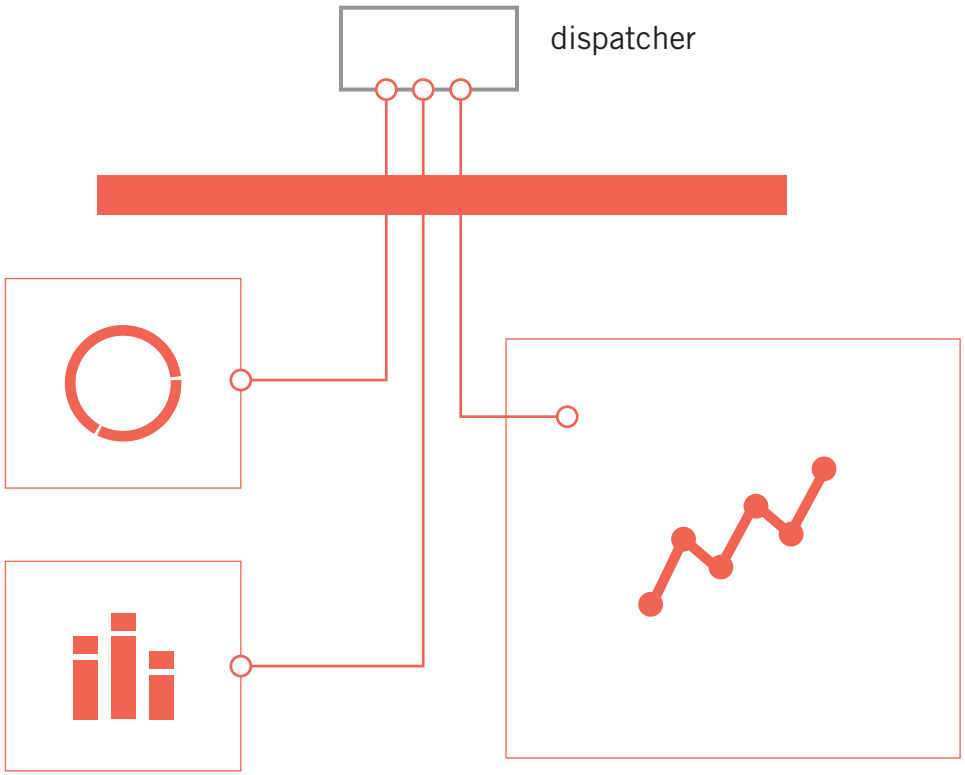
```
var newBrush = ...
    .on('brush', brushed);

function brushed(){
    console.log( newBrush.extent( ) );
    console.log( newBrush.empty( ) );
}
```

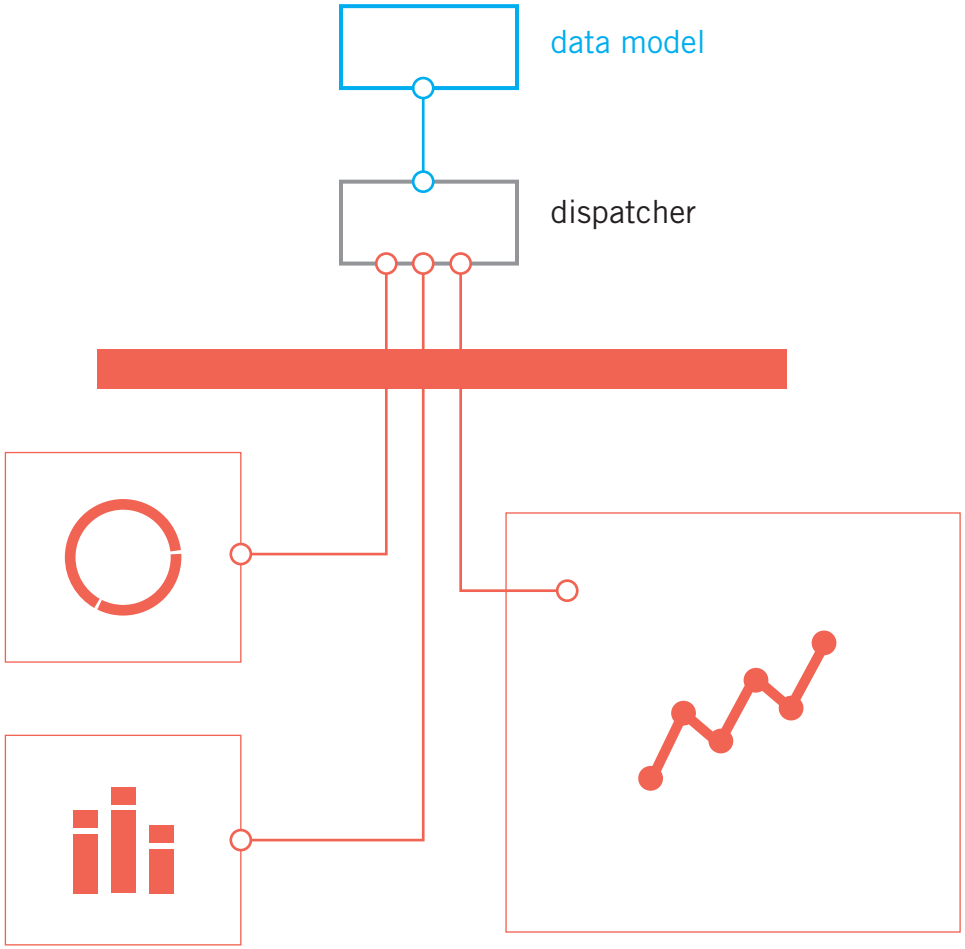
Task 2: practicing with brush

How do we use the brush interactions to filter data?

Let's review the structure of our visualization

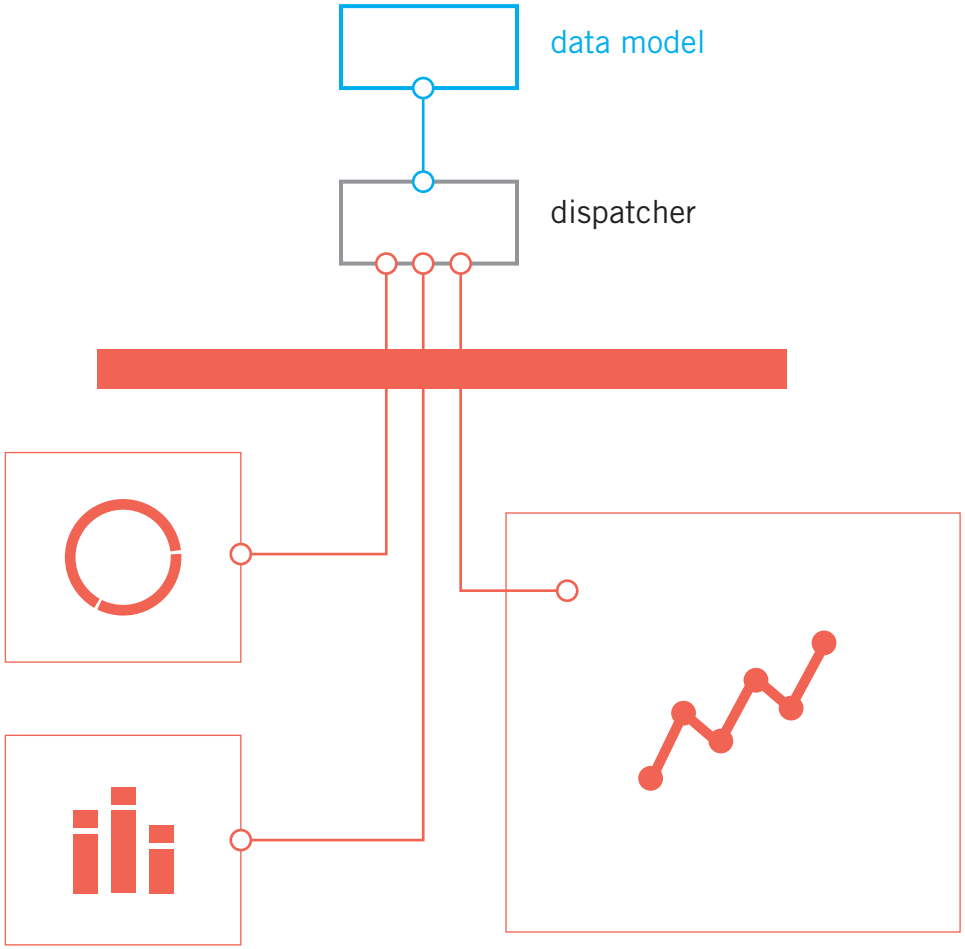



dispatcher


data model
dispatcher

In this paradigm...

"Views" (i.e. individual visualizations) are representations of the same underlying data "Model".


data model
dispatcher

In this paradigm...

"Views" (i.e. individual visualizations) are representations of the same underlying data "Model".

Views can trigger changes (filtering, for example) of the model.

Changes in the model are then propogated to the views.

Crossfilter.js

A library to filter large datasets

Crossfilter works with multi-dimensional datasets

```
var trips = crossfilter( rows );
```

id	col2	col3	col4

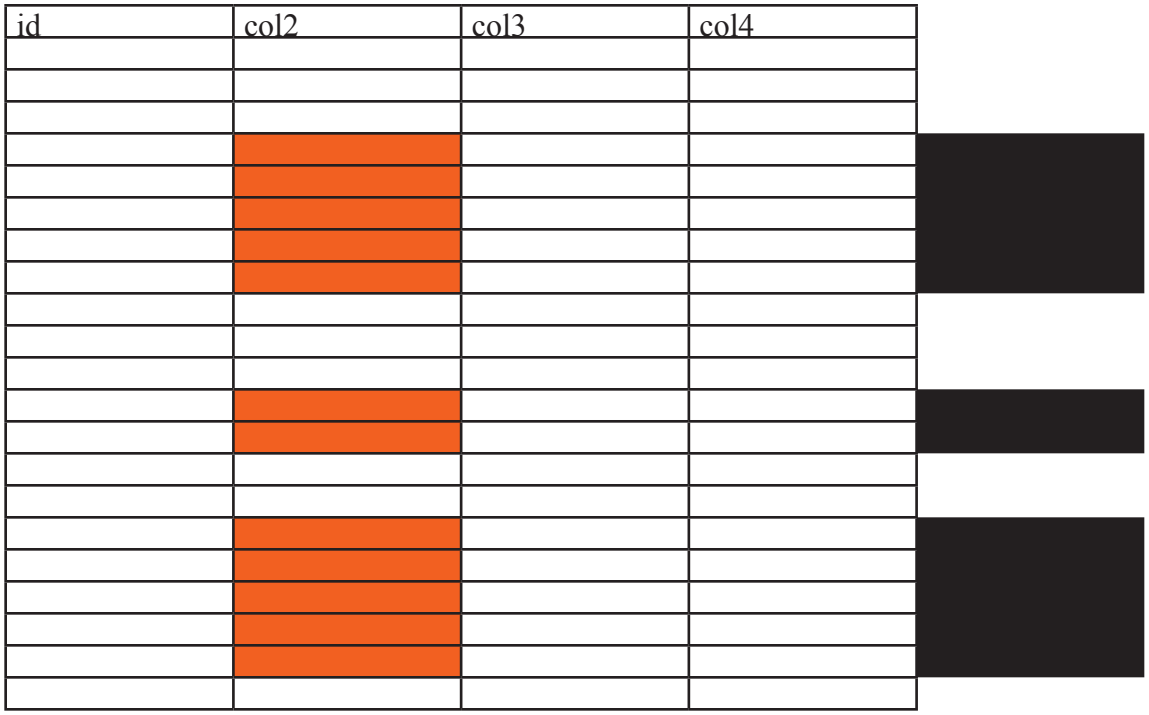
Crossfilter: create new dimensions

```
var tripsByDuration = trips.dimension(function(row){
     return row.duration});
```

id	col2	col3	col4

Crossfilter: filtering

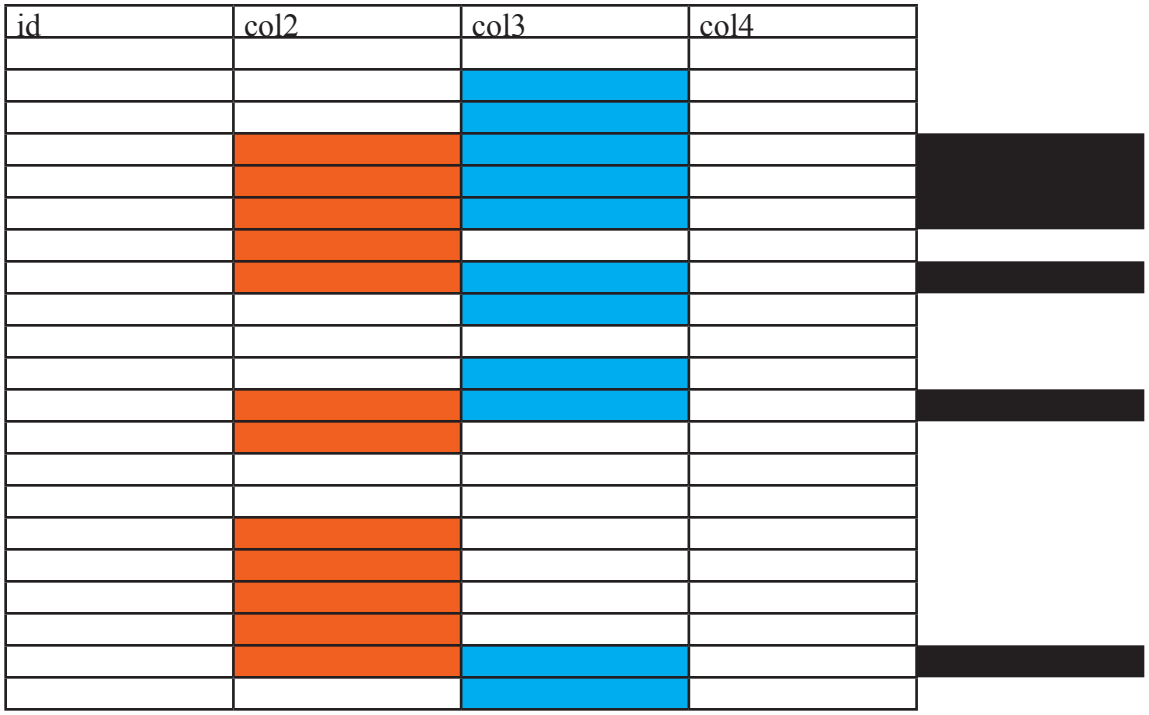
```
tripsByDuration.filter([0,200]);
```



Crossfilter: filtering

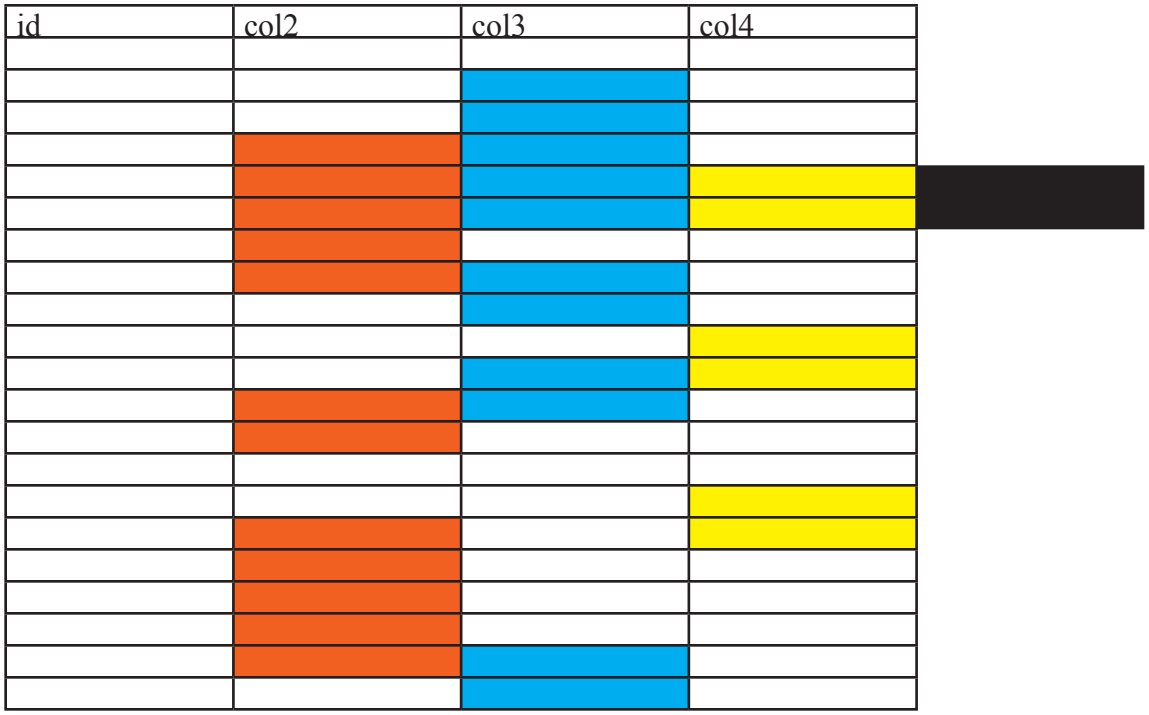
You may have multiple dimensions, but each dimension can only have one active filter:

```
tripsByDuration.filter([0,200]);
tripsByGender.filter('Female');
```



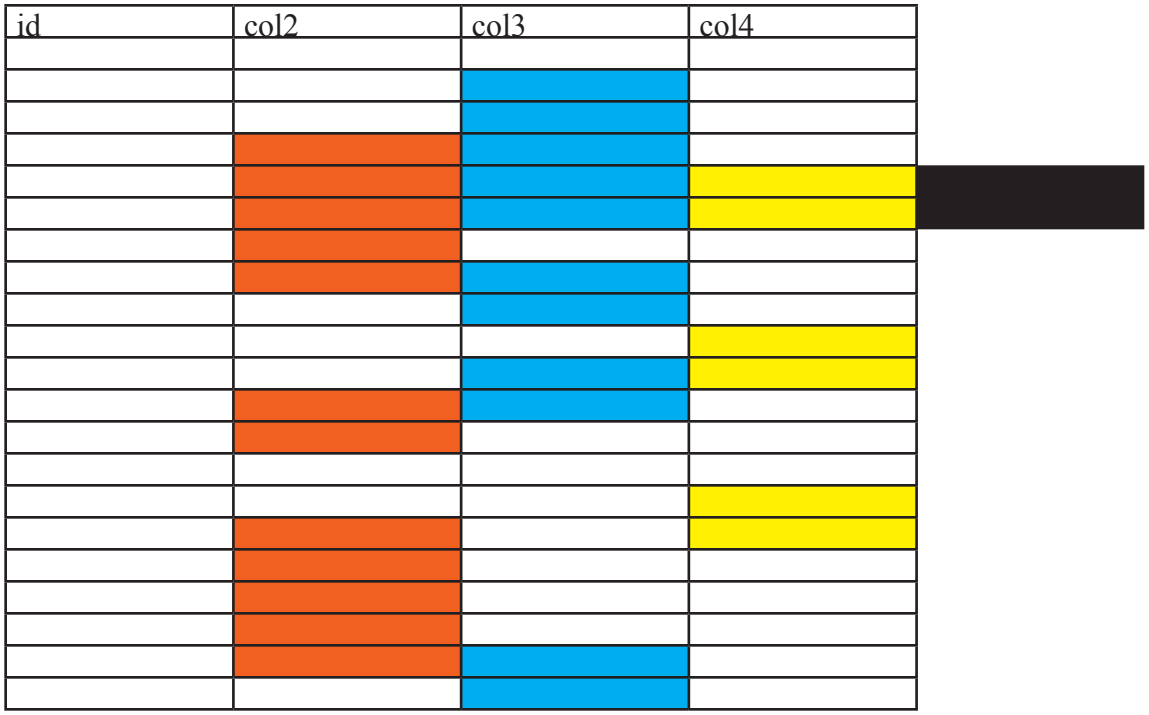
Crossfilter: filtering

Once again: updates to filters on dimension A will not undo filters on dimensions B, C, D etc.



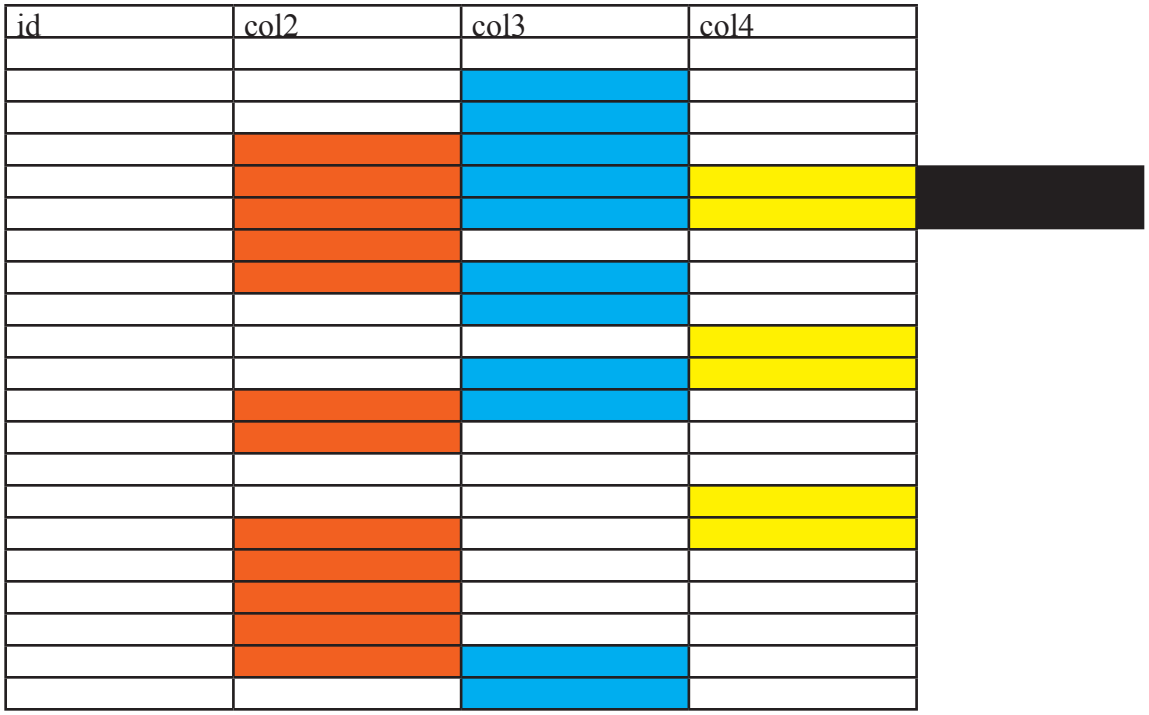
Crossfilter: filtering

To get the results of filtering by all current active filters, use dimension.top( ) or dimension.bottom( );



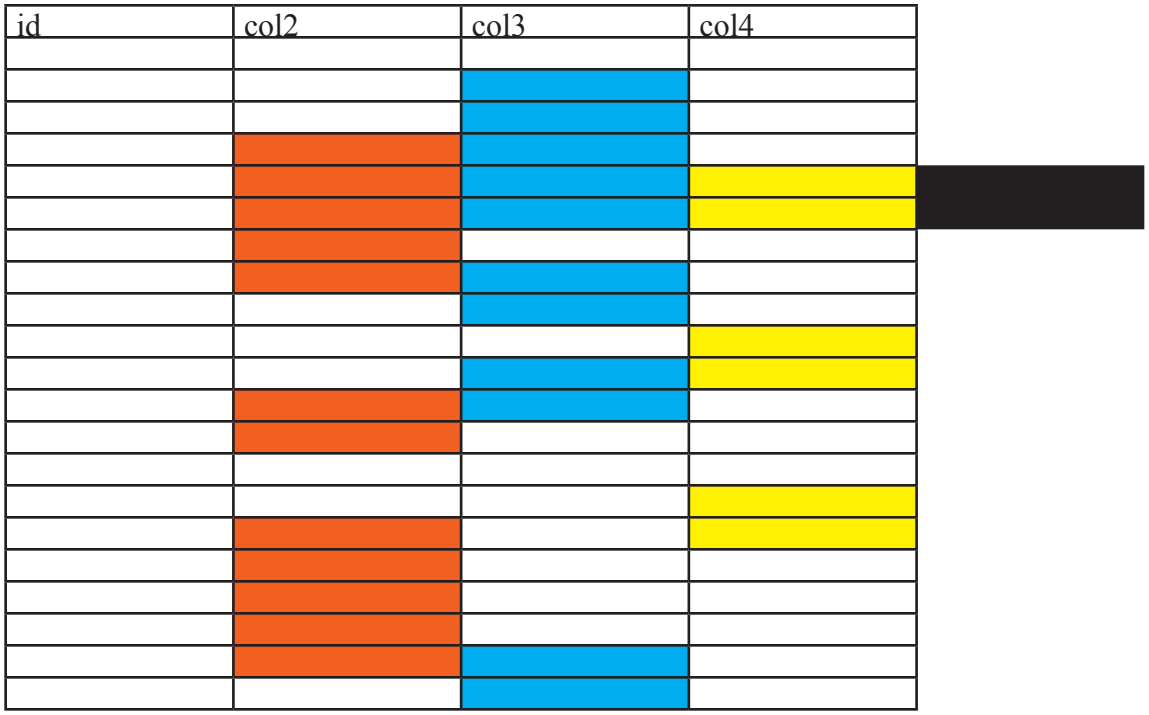
Crossfilter: filtering

```
tripsByDuration.top(3); //?
tripsByGender.top(Infinity); //?
```



Crossfilter: group by dimension

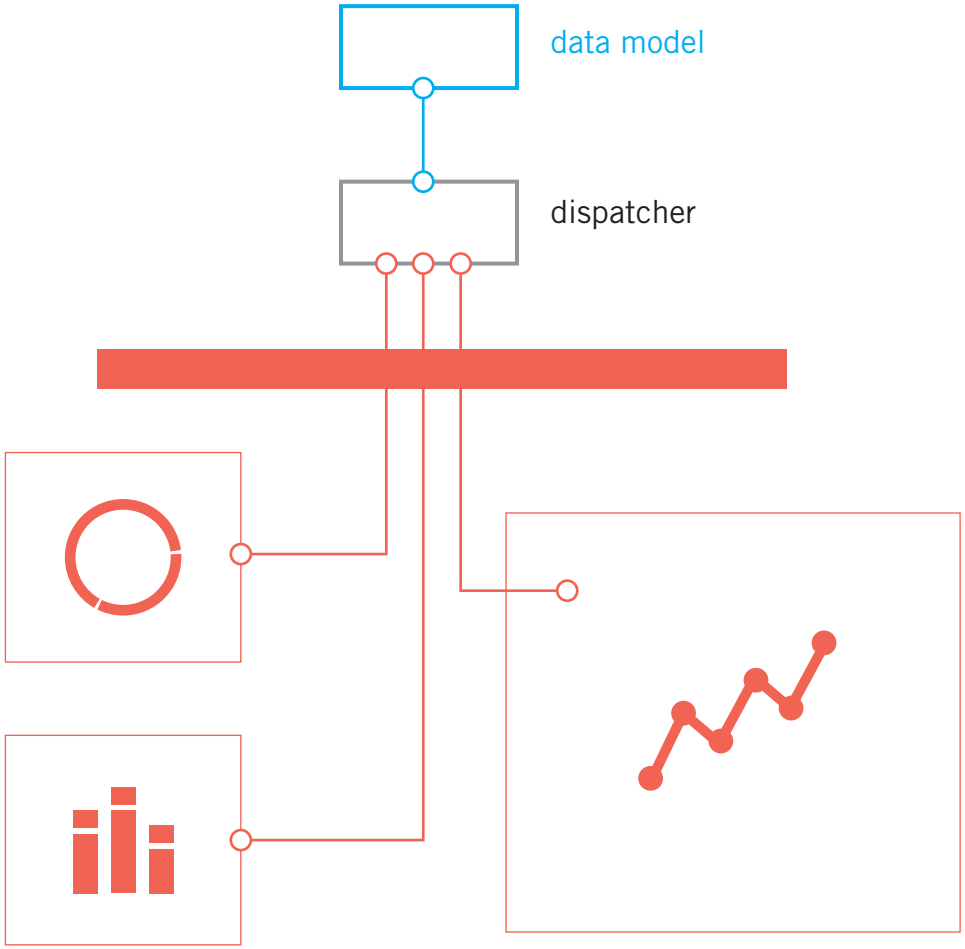
```
var tripsGroupByDuration = tripsByDuration.
group(function(d){ return Math.floor(d/60)});
```



Goal of the Workshop

- Further familiarize with event dispatching
- Become proficient with brushing interaction
- Use crossfilter to manipulate large datasets
- Putting it together: model / view / controller

To Review:



Homework

- Assignment 2-A
- Review other use cases of brushing
- Review modules, event dispatching, and crossfilter before next week's class