Performance of big data apps in AngularJS

Eyal (Jo) Arubas

Performance depends on…

- Number of data items
- Complexity of each item
- Structure of data
- Interactions with data
- Responsive UI

complex ui, need to visualize data and display it in an organized way.

ui needs to be responsive and fast.

javascript is single threaded, everything happens serially.

Performance depends on…

- ~~Number of data items~~
- ~~Complexity of each item~~
- **Structure of data**
- ~~Interactions with data~~
- ~~Responsive UI~~

usually only structure of data we can control, as it’s part of our design.

all the others are features of our application.

How do we monitor changes in data?

Hint: We $watch it

why do we want to monitor data? because it changes.

changes in data can have various sources: user, server, time

in angularjs, we have built-in tools to monitor and respond to changes in data.

AngularJS basics

- Dirty checks
- Digest cycle
- View-Model bindings

a loop which checks what has changes and updates everything.

Example, please.

```
<input ng-model="message">
<div>{{message}}</div>
```

$watch is behind the scenes

…and in front of them

Link

Data is never that simple

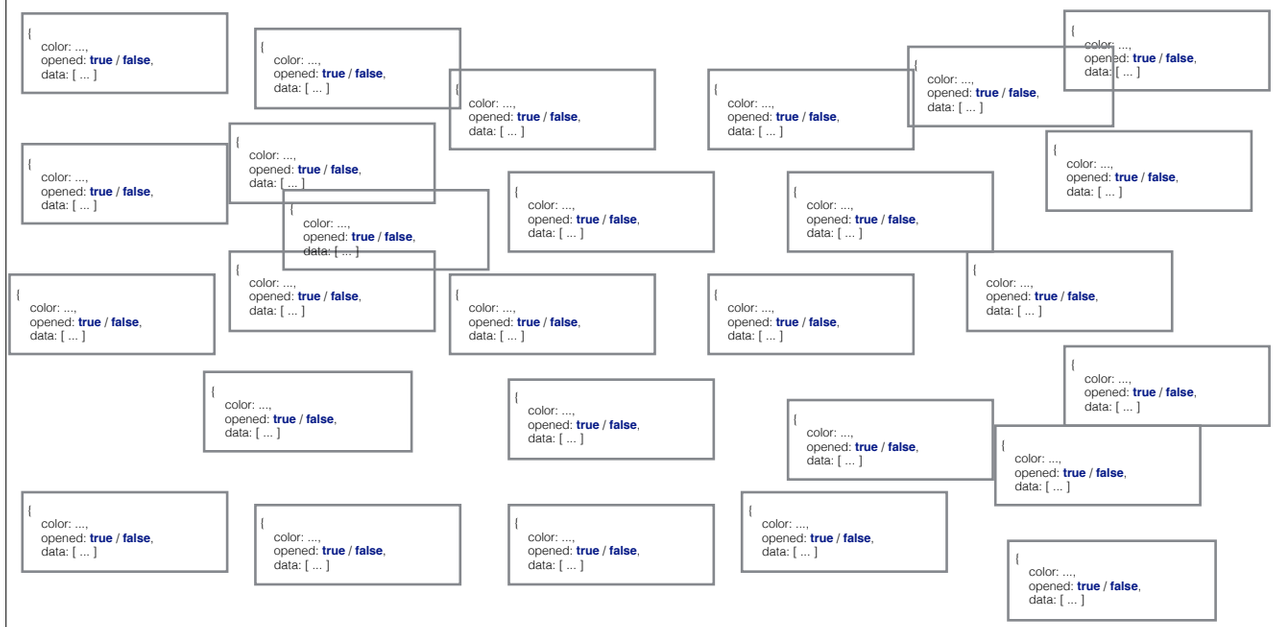
A Box

```
{
    color: ...,
    opened: true / false,
    data: [ ... ]
}
```

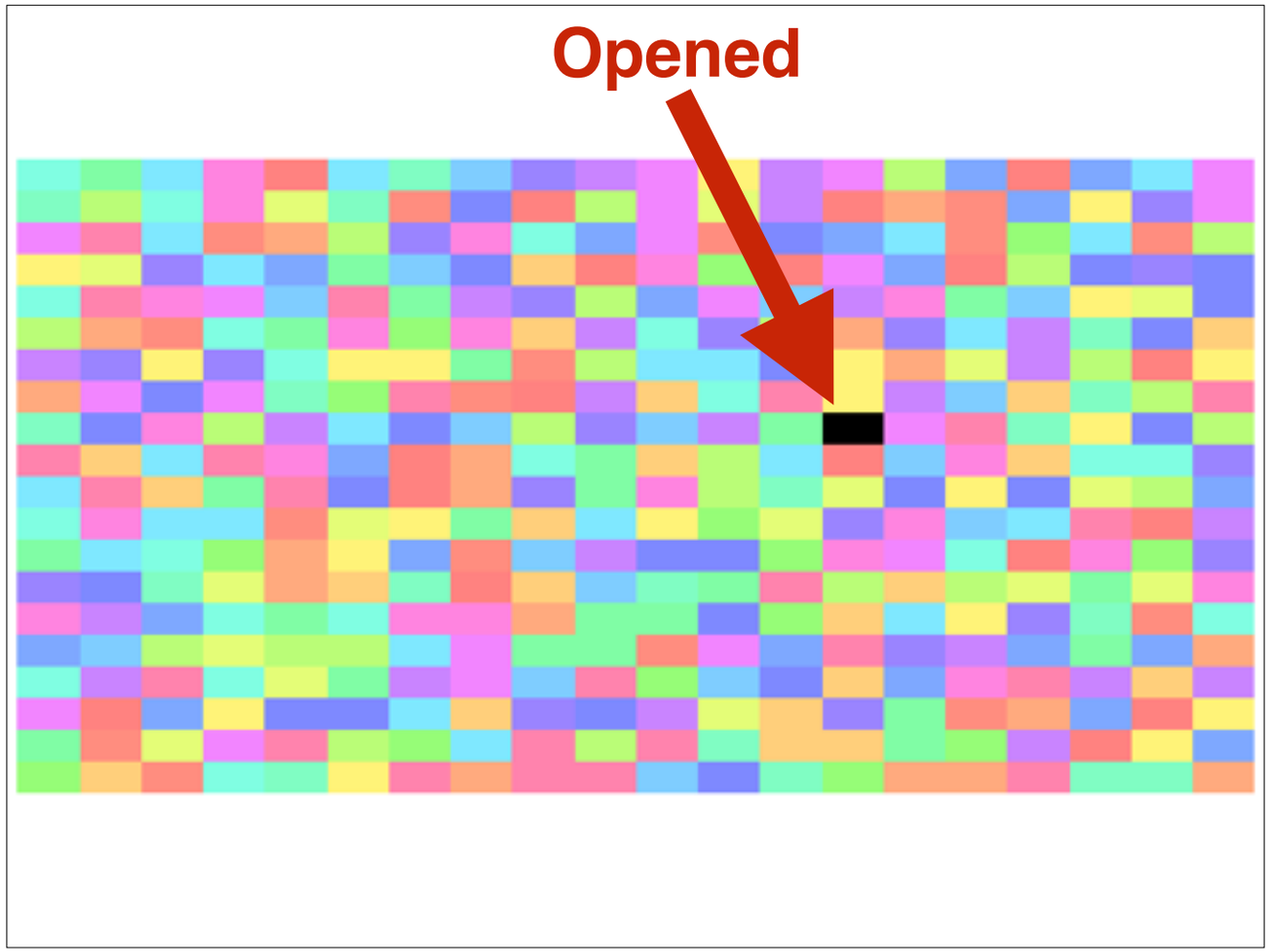
define a box

10,000 boxes

```
var boxes = [{ ... }, { ... }, { ... }, { ... }, ...];
```



Let’s say we want to draw boxes. Each box contains information which is a part of it.


Opened

A box is dynamic

- Open it
- Close it
- Change its color
- Modify the data it contains

Data, in general, is dynamic

- How do we organize the data?
- How do we structure it?
- How do we detect changes?
- How do we update the view?

$watch (carefully)

Watch what?

- Fields inside the objects?
- Just the containing array?
- References to the objects?

We need to define how our data changes, and then we can know how to efficiently watch it

Dynamic data

```
var i = 1;
$interval(function() {
    open($scope.boxes[i]);
    close($scope.boxes[i - 1]);
    i = (i + 1) % n;
}, 30);
```

First try

```
scope.$watch('boxes',
    function(boxes){
        /* update the DOM */
    });
```

Link

Nothing happens. Why?

Because we only watch the reference to the ‘boxes’ array. it never changes.

Second try

```
scope.$watch('boxes',
    function(boxes){
        /* update the DOM */
    }, true);
```

Link

Too slow. Why?

Because we watch the internals of the objects, some of which we don't care about. There's lots of data which is very expensive to watch.

Third try

```
scope.
$watchCollection('boxes',
    function(boxes){
        /* update the DOM */
    });
```

Link

Good. Why?

We watch the references to the objects inside the array. It’s cheap.

We can only achieve that if we carefully think how the data is updated.

We need to replace objects in the array, and not just update the internal fields.

```
function open(box){
    // 1. set box.opened = true?
    // 2. replace the box with an
    //     identical, but opened, box?
}
```

Performance
vs.
complexity

In this case, the tradeoff is having a more complex implementation of data update, and gaining performance.