Angular 1.x

Don’t let Angular do magic when you don’t need it

Agenda

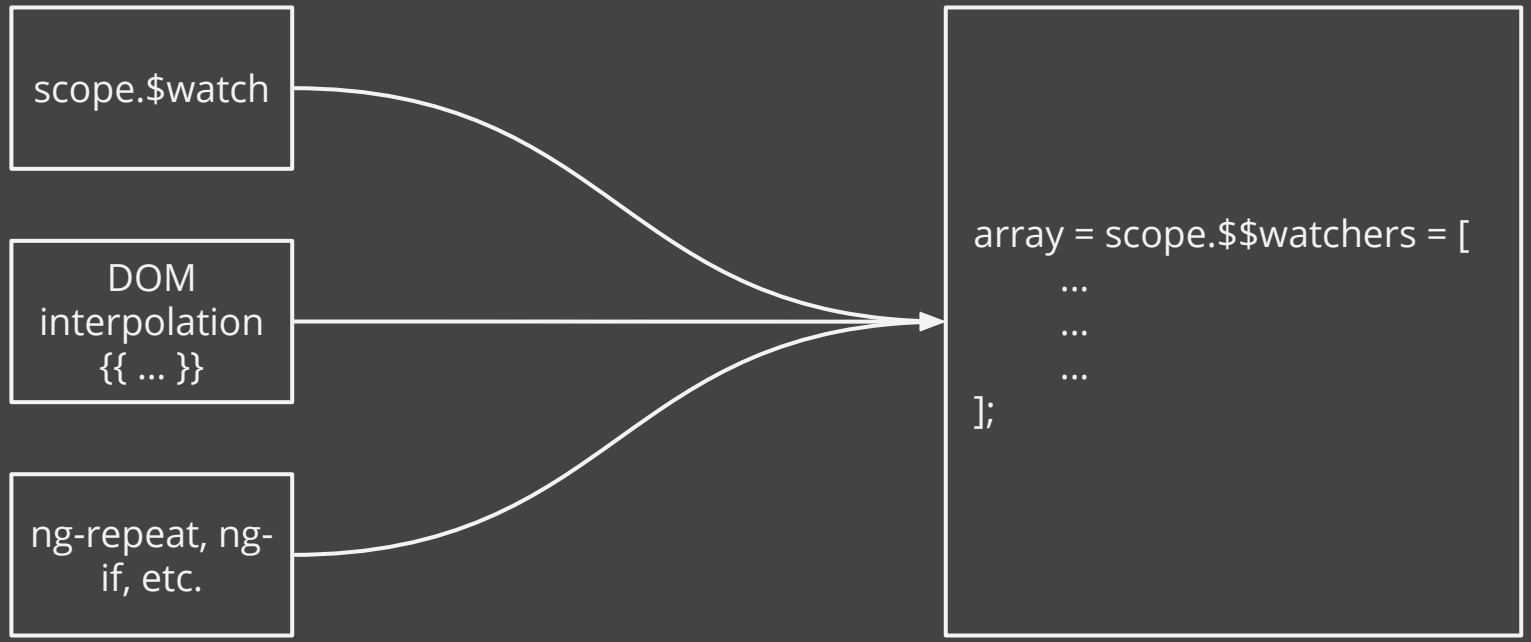
1. Important concepts (watch, **digest cycle**)
2. Rules;
3. Examples from Web app;

Goal: understand the inner workings of AngularJS to write better code.

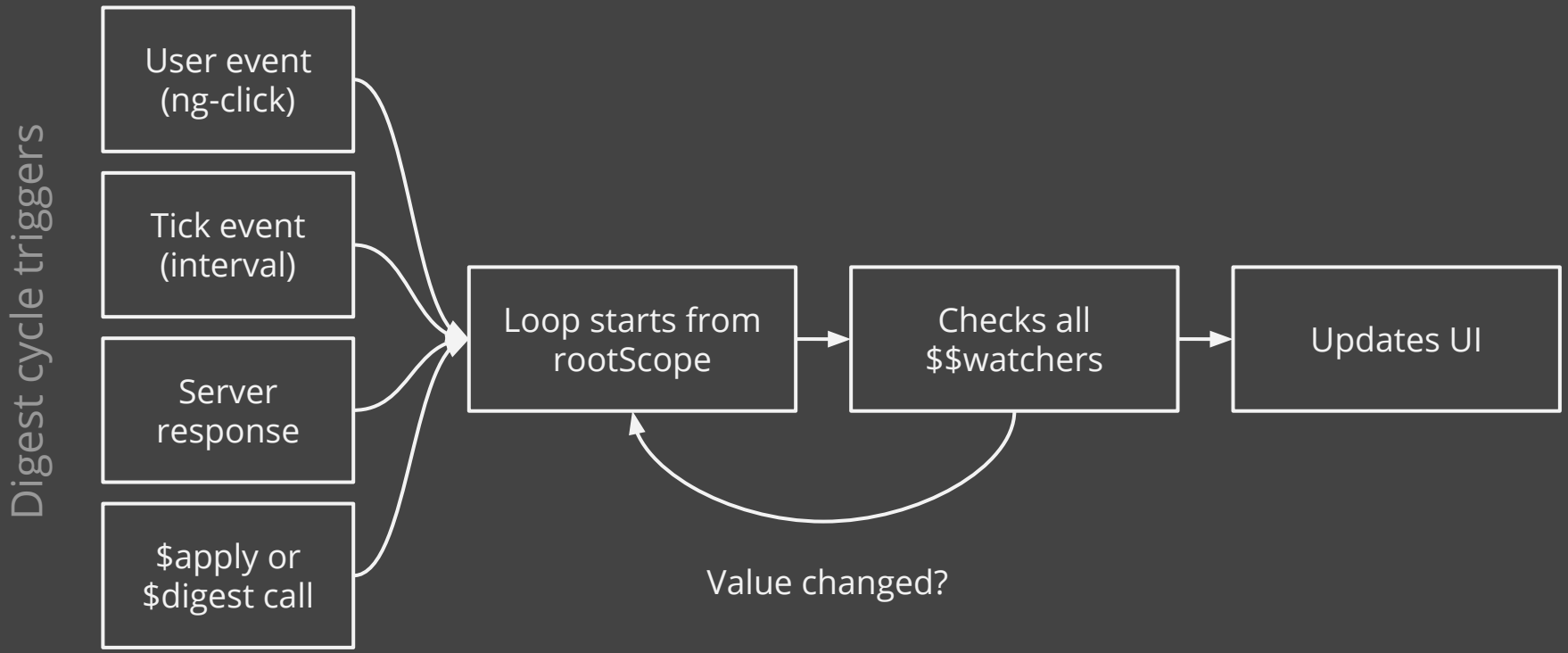
Important concepts

Important concepts: $watch

All watched elements are added to $$watchers array.



Important concepts: digest cycle



Important concepts: $watch

$$watchers must be re-evaluated with each run of the $digest loop.

Problem:

JavaScript is single threaded, therefore the digest loop blocks UI until the loop completes.

Rules

Rule #1

Use one-time binding when possible

Example 1, 3

Why?

A signal to Angular that you don’t need to watch data.

Does not register watcher.

How? + example

```
<div>{{::ctrl.name}}</div>
<div>{{::ctrl.i18n.label}}</div>
```

```
<div ng-class="::{ red: ctrl.colors.red }">
    One time red
</div>
```

Rule #2

Avoid using ng-repeat for large datasets;
Always use 'track by' to max performance;

Example 2

Why?

Refreshing ng-repeat would remove all elements and recreate them again which is costly DOM operation.

How?

```
<div ng-repeat="item in ctrl.arr track by item.id">
    {{ item }}
</div>
```

If you don't have a unique identifier, track by $index can also provide a performance boost.

```
<div ng-repeat="item in ctrl.arr track by $index">
    {{ item }}
</div>
```

Rule #3

Never bind anything (ng-show, ng-repeat, etc.) directly to a function.

```
<div ng-show="myFunction()"> … </div>
```

Example 4

Why?

This function will run on every digest cycle, possibly slowing your application.

How?

```
// Don’t:
<div ng-show="ctrl.showIt()">"Hello World!"</div>


// Do
<div ng-show="ctrl.showItRight">"Hello World!"</div>
```

Rule #4

Avoid using filters if at all possible

Why?

They are run twice per digest cycle, once when anything changes, and another time to collect further changes.

How?

Angular includes a $filter provider for JS before parsing into the DOM.

Preprocess our data before sending it to the View.

Rule #5

scope.$watch indicates bad architecture

Why?

Combination of services and reference bindings can achieve the same results with lower overhead.

How?

If you must create a watcher, always remember to unbind it at the first available opportunity.

```
var myWatcher = scope.$watch('scope', function(newVal, oldVal){
    // do smth
});

unbinder(); // deregisters the watch from $$watchers.
```

Rule #6

Minimize number of digest cycles with ng-model-options

Example 5

Why?

The more watchers you have the longer the digest cycle is.

Less digests => less loops to run.

How?

```
<input type="search"
       ng-model="ctrl.searchQuery"
       ng-model-options="{ debounce: 3000 }">

<input type="search"
       ng-model="ctrl.searchQuery"
       ng-model-options="{
           updateOn: 'default blur',
           debounce: { 'default': 3000, 'blur': 0 }
       }">
```

Rule #7

Use ng-if/ng-switch instead of ng-show

Why?

Fewer bindings, content won’t be compiled until it’s visible.

Rule #8

Remove elements not visible in viewport and use pagination when you need to represent a big chunk of data

Demo

Why?

Less $watches, thus less work js performs during $digest

Rule #9

Avoid core directives and use $digest() instead of $apply()

Example 6

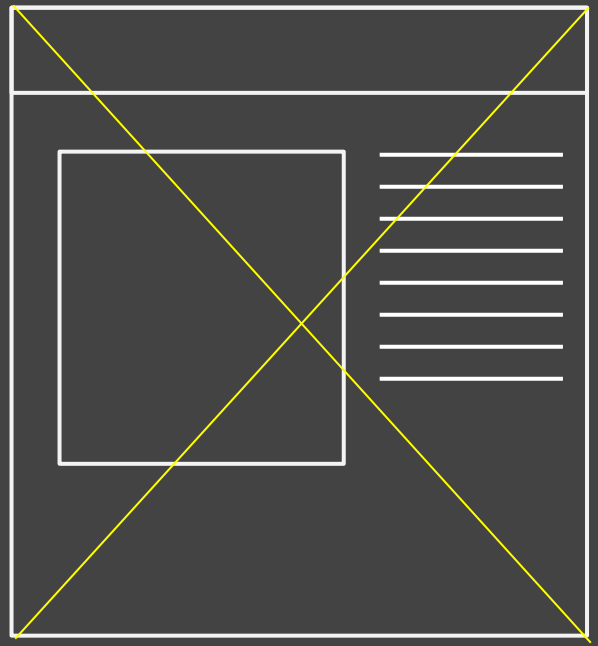
Why?

Each ngEventDirectives (click, dblclick, mousedown, mouseup, etc.) calls $apply which starts from $rootScope downwards.

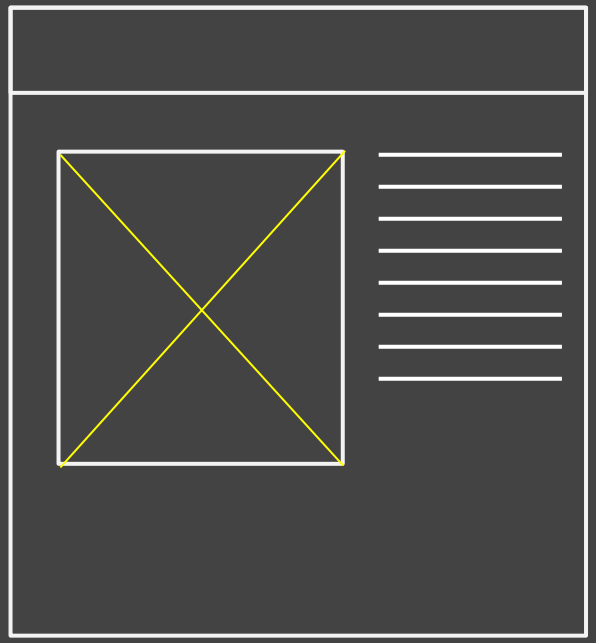
Angular 1.3.6 source code

Why?

$rootScope.$digest();



$digest();



How?

Write custom directives, catch the events you need and trigger local $digests instead of global $apply.

Rule #10

Eval your code

How?

+ Batarang
+ Chrome profiler (memory, performance)
+ Performance.now()

Rule #11

Clean after yourself

Why?

To ensure safety and flag your scope for garbage collection more rapidly.

Not doing so will cause memory leaks in older browsers, and slow down your Garbage Collector in modern browsers.

How?

Flush intervals, cancel timeouts, destroy new scopes, unbind all your watchers and event listeners.

How?

Flush $interval / $timeout:

```
var stop = $interval(function() {
    // do something
}, 100);

$interval.cancel(stop);
```

How?

Destroy new scope:

```
if (_newScope) {
    _newScope.$destroy();
}

var _newScope = $scope.$new();
```

How?

Unbind watchers:

```
// When you call the $watch() method, AngularJS returns an
unbind function that will kill the $watch()

var unbindWatcher = $scope.$watch("scope", function() {
    // when there's no more need to watch the change
    unbindWatcher();
});
```

How?

Always explicitly call $on('$destroy') in custom directives.

```
scope.$on("$destroy", function( event ) {
    $timeout.cancel(timer);
    $interval.cancel(interval);
    unbindWatcher();
});
```

Web app examples (#3, #7)

```
<div class="form-group"
       ng-show="hasStateProp('backgroundColor')">
       <label class="control-label col-xs-4">
           <span ng-bind="::l10n.color.background.label">
           </span>
           <i class="fa fa-tint fa-fw"></i>
       </label>
</div>
```

Web app examples (#11)

Web Client - running $interval;
directives.js :122

Web Config - new scope causes memory leak
directives.js :3069

Q&A