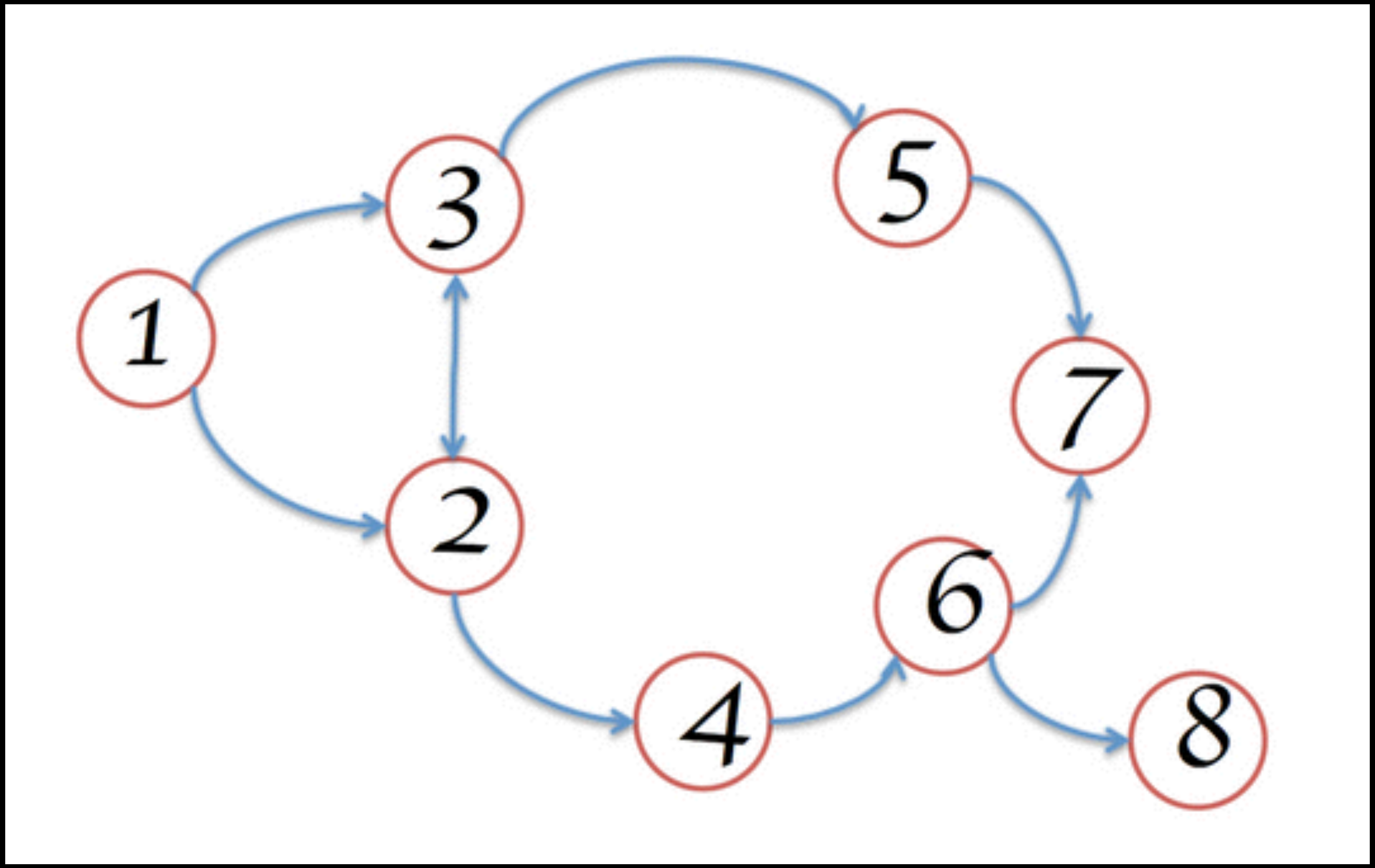
JS Performance & Memory Leaks

Keep Yo Angular Running Snappy

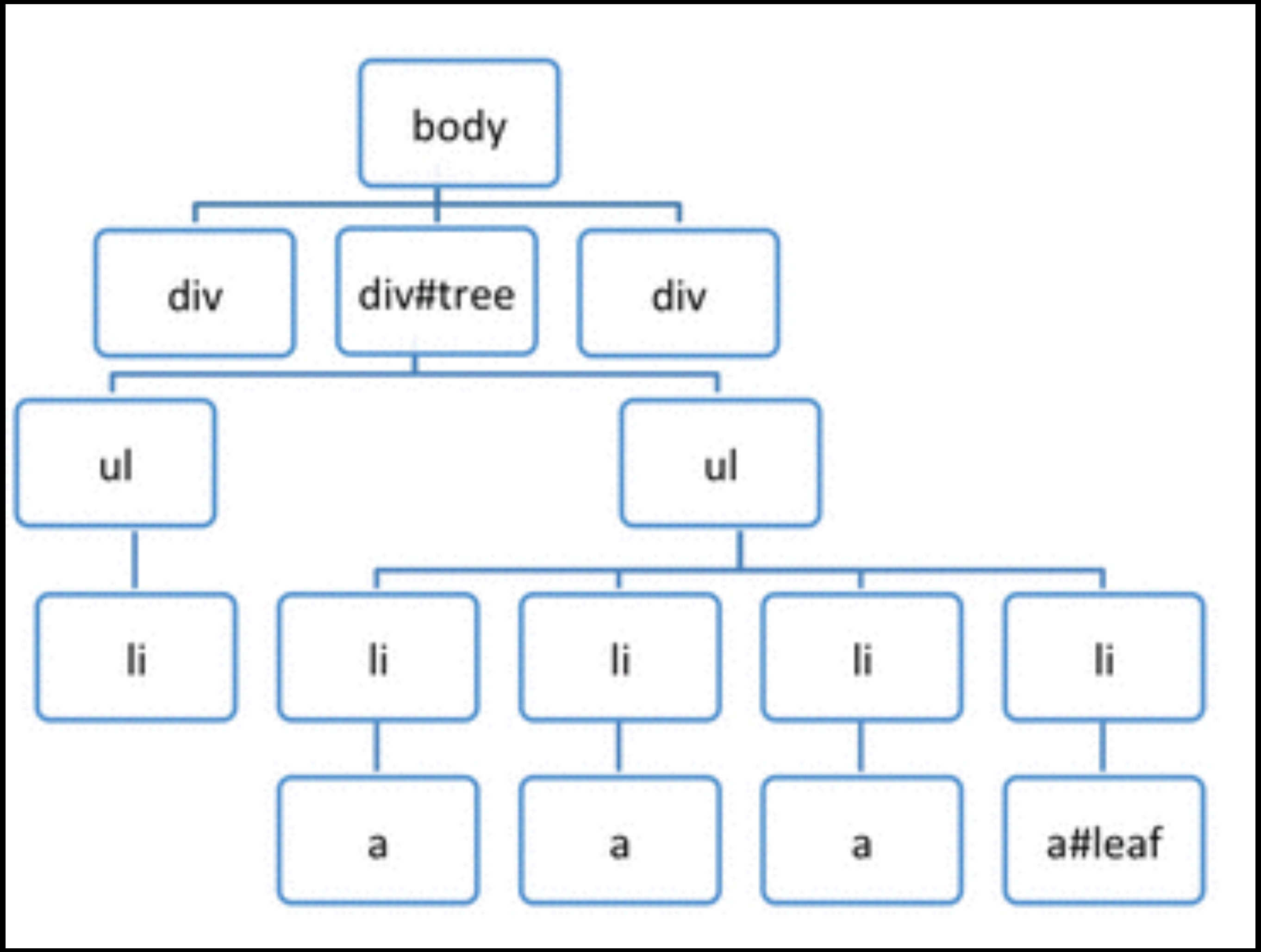
How To Think of Memory

- It a graph!



How To Think of Memory

- Something a little more visual



Common Memory Leak Cases

Them Dom Leaks

```
someDiv = document.createElement(“div”);
display.appendChild(someDiv);

//Some other code
display.removeAllChildern();

// Oh no zombie div, it’s still alive!
```

Common Memory Leak Cases

<u>Closures are awesome till they arent</u>

```
var a = function () {
    var largeStr = new Array(1000000).join('x');
    return function () {
        return largeStr;
    };
}();

// largeStr can stick around
```

Common Memory Leak Cases

Those Damn Timeouts

```
var myObj = {
    callMeMaybe: function () {
        var myRef = this;
        var val = setTimeout(function () {
            myRef.callMeMaybe();
        }, 1000);
    }
};
myObj.callMeMaybe();
myObj = null; // This aint gonna cut it
```

Solving Memory Leaks in AngularJS

Use $destroy to clean up!

```
$scope.on('$destroy', function(){
    // KILL
    // ALL
    // REFERENCES
    // NOW
});
```

Solving Memory Leaks in AngularJS

Use $destroy to clean up!

- Unbind event-listeners: element.off('click')
- Kill your watchers:
    - var unwatch = scope.$watch(…
    - unwatch(); // Watcher is dead

Solving Memory Leaks in AngularJS

Use $destroy to clean up!

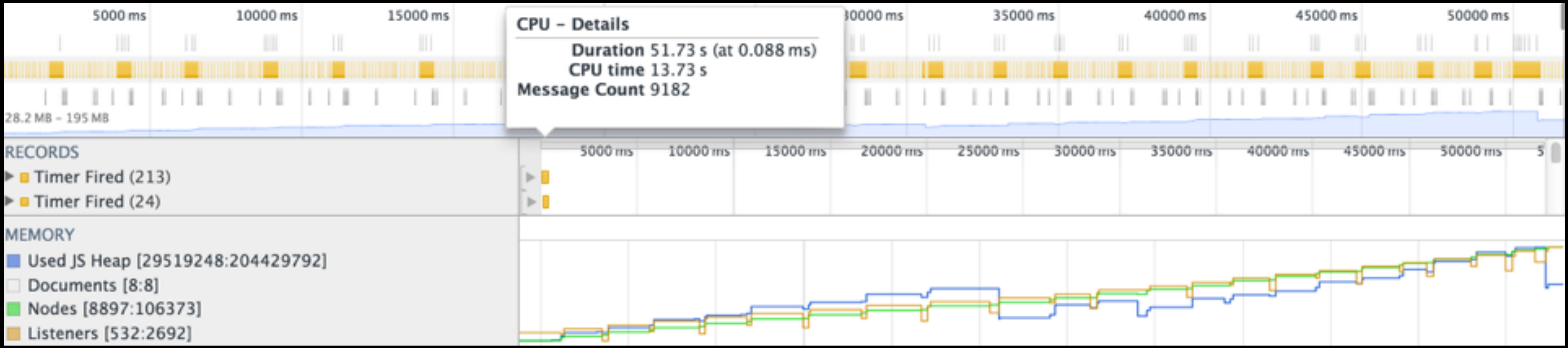
```
var button = scope.button = {
            selected: false,
            callback: scope.onSelect || angular.noop
          };
};

scope.$destroy(…
    button = null;
…);
```

How to Find Memory Issues

- ***CHROME DEV TOOLS!!!!***
- ***https://developer.chrome.com/devtools/docs/javascript-memory-profiling***

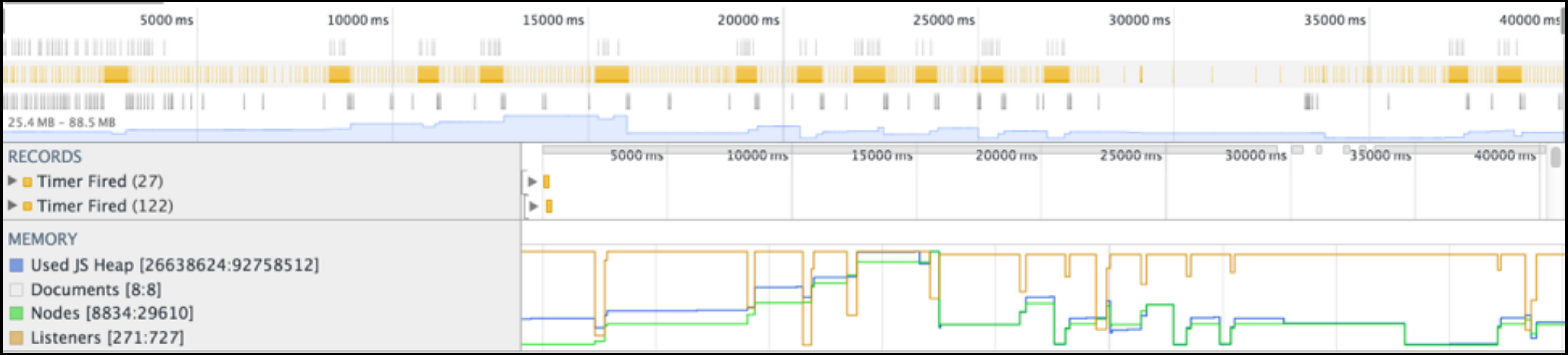
Fruits of our Efforts



CPU – Details
Duration 51.73 s (at 0.088 ms)
CPU time 13.73 s
Message Count 9182

RECORDS
▸ Timer Fired (213)
▸ Timer Fired (24)

MEMORY
🟦 Used JS Heap [29519248:204429792]
⬜ Documents [8:8]
🟩 Nodes [8897:106373]
🟧 Listeners [532:2692]



RECORDS
▸ Timer Fired (27)
▸ Timer Fired (122)

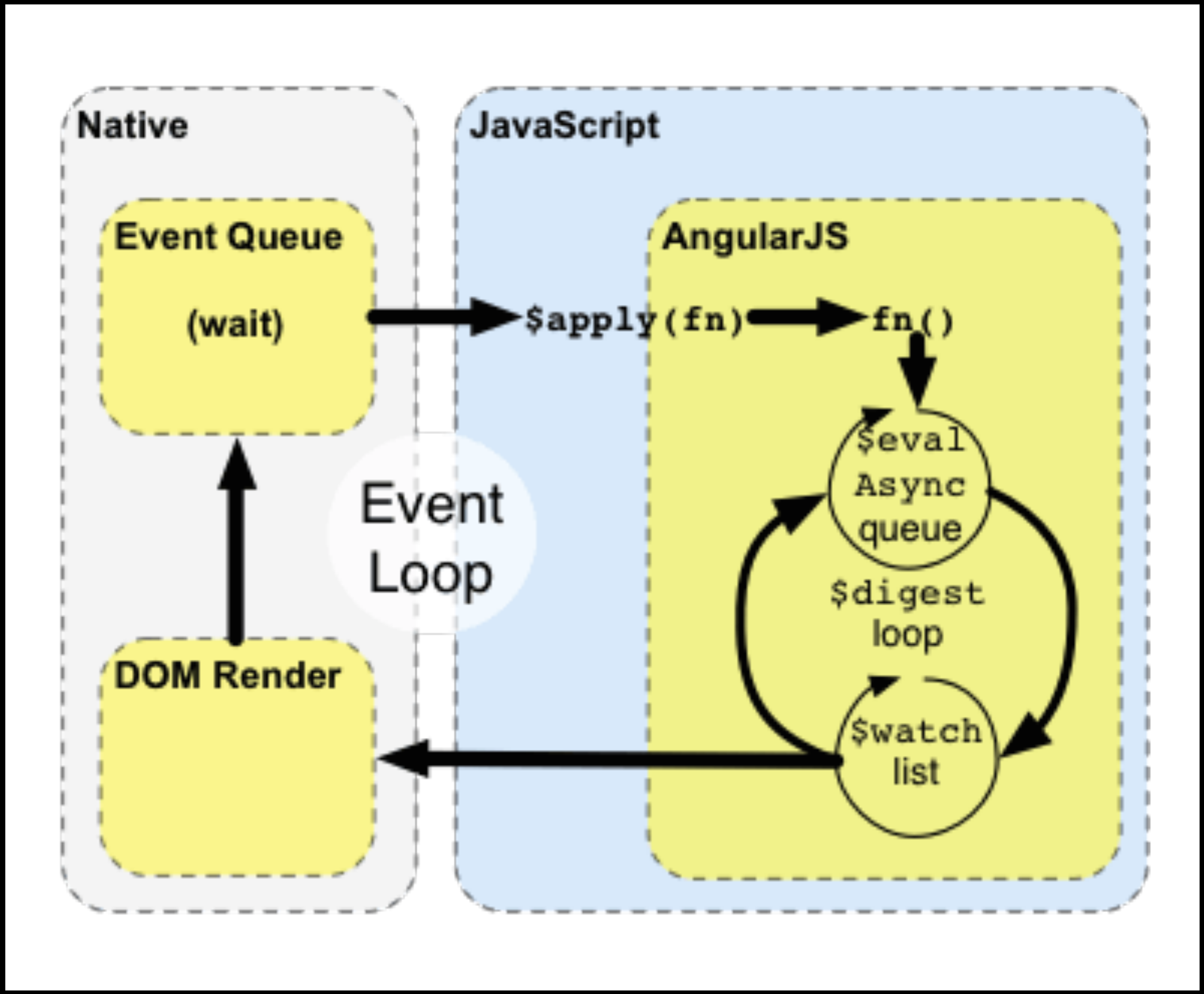
MEMORY
🟦 Used JS Heap [26638624:92758512]
⬜ Documents [8:8]
🟩 Nodes [8834:29610]
🟧 Listeners [271:727]

Performance

How do we keep Angular snappy?

Understanding the Angular Digest Cycle

Triggers: $apply, $digest, $timeout, ngClick



Psst… Dont use mouse move events (or them debounce)

Performance Tips

<u>Using $digest() V.S. $apply() -> $$watchers</u>

Think of scopes and watcher like a tree from the $rootScope

$digest triggers digest cycle in current scope and below

V.S.

$apply starts at $rootScope and goes down

Try $applyAsync([exp]);
This can be used to queue up multiple expressions which need to be evaluated in the same digest.

Performance Tips

Watch your Watchers

```
var unbinder = scope.$watch('scopeValueToBeWatcher',
function(newVal, oldVal){})
```

- Avoiding creating a Watcher programmatically
- watchers > 2000 = caution zone // code smell
- Try services or event dispatching
- Were using ngStats to count that
  - DEMO!

Performance Tips

Use One-Way Bindings!!

- Binds once and then deregisters watcher
- Dont use it when you expect the value to change

```
{{::omgOneAndDoneBinding}}
```

Performance Tips

<u>$broadcast V.S. $emit</u>

$broadcast calls all registered listeners from scope DOWN
V.S.
$emit calls all registered listeners from scope UP

- Dont use them
- If you have to: $rootScope.$emit(…);
- What we did: event-dispatch.js
  - Doesnt rely on digest cycle
  - Dispatcher/Callback register
  - Dispatcher.listen('MediaFilter:Filtered', func…);

Performance Tips

<u>$filter</u>

- Dont use them on the DOM
- They are run twice per digest cycle, once when anything changes, and another time to collect further changes, and do not actually remove any part of the collection from memory

- BLAH -> {{ array | filter }}

  - Do it in the controller -> $filter('filter')(array)

Performance Tips

ngShow/ngHide V.S ngIf/ngSwitch

- ng-hide and ng-show simply toggle the CSS display property.
  - What that means everything is just hiding but the $$watchers are still registered
- ng-if remove elements off the DOM
  - That means anything inside is gone along with the $$watchers

Performance Tips

Crazy DOM Logic

- Have crazy logic using ng-if?
- Try ng-include!

```
%ng-include(src="show.template")

Show Logic:
if ( item.sucks() ) {
    show.template = ‘sucks.html';
} else if ( item.awesome() ) {
    show.template = ‘awesome.html';
}
```

Performance Tips

Crazy DOM Logic For Directives

Use attributes passed to directives to choose template

```
templateUrl: function(tElement, tAttrs) {

    if (tAttrs === ‘photo’) {
        return ‘somePhotoTemplate’;
    } else {
        return ‘otherTemplate’;

    }


    …

}
```

Performance Tips

$http Performance Boost

-
  ```
  app.config(function ($httpProvider) {
              $httpProvider.useApplyAsync(true);
  });
  ```

- Configure $http service to combine processing of multiple http responses received at around the same time via $rootScope.$applyAsync. This can result in significant performance improvement for bigger applications that make many HTTP requests concurrently (common during application bootstrap).

Performance Tips

ng-repeat Can Get Nasty

- Mo’ DOM elements mo’ problems (watchers)
- ng-repeat="model in collection track by model.id”
  - ngRepeat will not have to rebuild the DOM elements for already rendered items, even if the JavaScript objects in the collection have been substituted
- angular-viewport-watch to the rescue
  - http://.github.com/shahata/angular-viewport-watch
  - Hide them $$watchers
- DEMO!

Keeping Digest Cycle Fast

- Keeping watcher count down
  - Avoid making new $watchers
- Use on way bindings
  - {{::oneWay}}
- Logic triggered by digest cycle should be fast
  - ng-repeat="a in getItems()"
- Avoid creating new scopes, mo’ scope mo’ slow
- ngIf over ngShow
- Avoid $emit and $broadcast
- Watchers and Digest cycles arent evil just have to use them wisely

fin