HTML5 Client

Performance

Ben Aston, August 2015

1. Context
2. Identifying Problems
3. Solutions
4. Summary

1

Context

Context : What the User Sees

"A script on this page may be busy, or it may have stopped responding. You can stop the script now, or you can continue to see if the script will complete."

Context : Runtime

- Single-threaded
- Run to completion semantic
- Cooperative multi-tasking
- Script execution blocks UI rendering!

```
while(1) {
  // Do nothing else (literally).
}
```

2

Identifying Problems

Identifying Problems

- Custom instrumentation
- Chrome developer tools

Chrome Developer Tools

3

Solutions

Caveat

“Premature optimisation is the root of all evil."

Solutions : Yielding the Event Loop

```
const start = performance.now();

function sampleFps(period, d=Q.defer(), fps=0) {
  requestAnimationFrame(now => {
    if (now-start >= period) {
      return d.resolve((fps+1) * 1000/(now-start));
    }
    sampleFps(period, d, ++fps);
  });

  return d.promise;
}
```

Solutions : Yielding the Event Loop

```
const

function sampleFps(period, d=Q.defer(), fps

       return d.resolve((fps+1) * 1000

    sampleFps(period, d, ++fps



  return d.promise
}
```

```
sampleFrameRate(10000)
  .then(fps=> console.log(fps));
```

Solutions : Yielding the Event Loop

```
const start = performance.now();

function sampleFps(period, d=Q.defer(), fps=0) {
  requestAnimationFrame(now => {
    if (now-start >= period) {
      return d.resolve((fps+1) * 1000/(now-start));
    }
    sampleFps(period, d, ++fps);
  });

  return d.promise;
}
```

Solutions : Reduce Job Queue Length

```
const scheduler = {};
let latch = false;
const callQueue = [];

scheduler.schedule = function(options) {
  callQueue.push(getFnForQueue(options));

  if(!latch) {
    return callQueue.shift()();
  }
}
```

Solutions : Reduce Job Queue Length

```
  const
  let
  const



  scheduler.schedule = function(options) {


      return callQueue.shift()();


  }
```

```
function openLatchAndDrainOne() {
  latch = false;
  (callQueue.length && callQueue.shift()());
}
```

Solutions : Reduce Job Queue Length

```
const scheduler = {};
let latch = false;
const callQueue = [];

scheduler.schedule = function(options) {
  callQueue.push(getFnForQueue(options));

  if(!latch) {
    return callQueue.shift()();
  }
}
```

Solutions : Choose Your Moment

```
var service = require('my-service');

function MyCtor() {
  this.foo = service.doSomethingExpensive();
};
```

Solutions : Choose Your Moment

```
MyCtor.prototype = {
  get foo() {
    return service.doSomethingExpensive();
  }
};
```

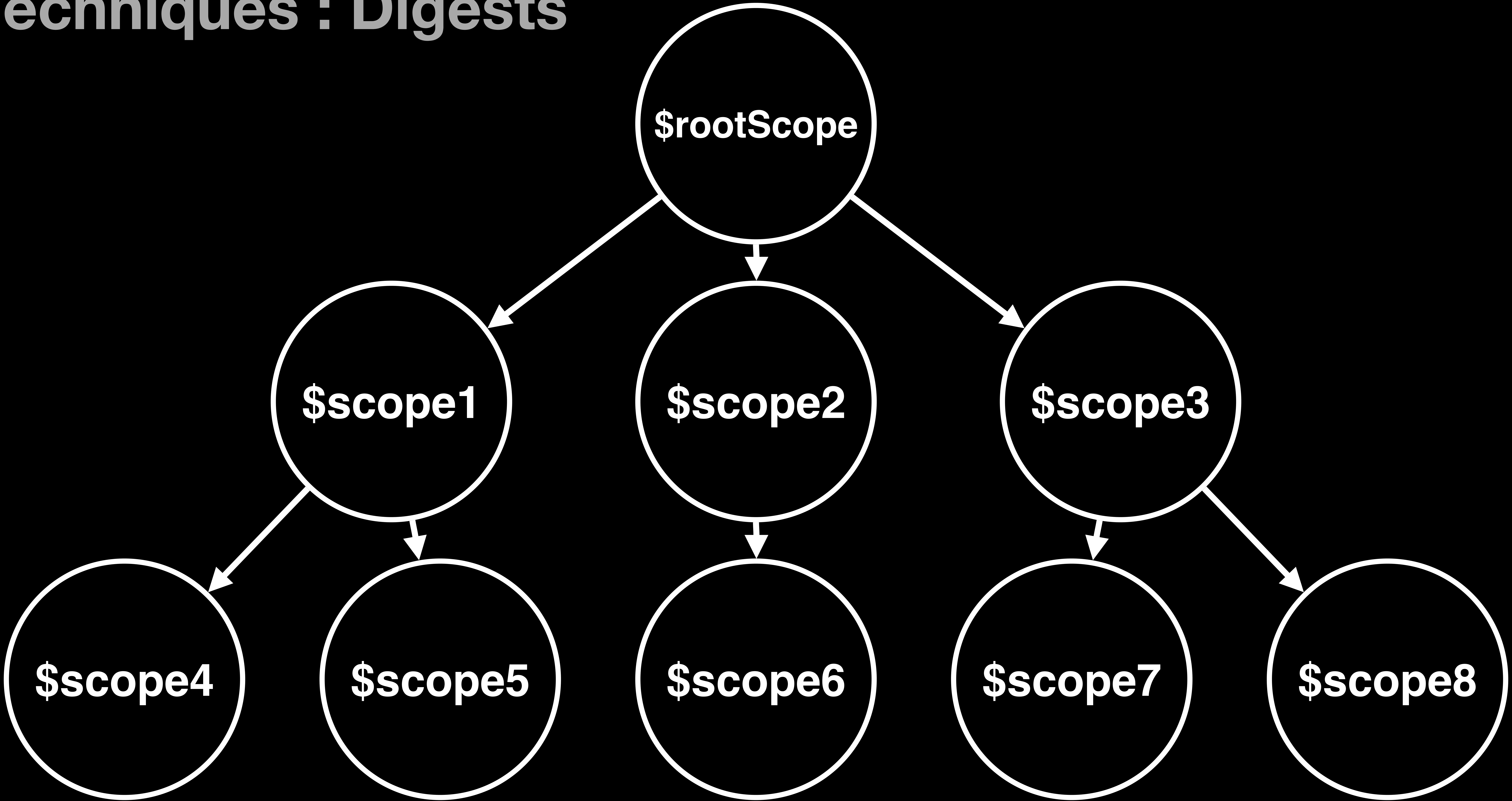
Solutions : Choose Your Moment

```
MyCtor.prototype = {
  get foo() {
    return this._foo ||
     this._foo = service.doSomethingExpensive();
  }
};
```

Solutions : Choose Your Moment

```
MyCtor.prototype = {
    foo: _.memoize(function() {
        return service.doSomethingExpensive();
    })
};
```

Techniques : Digests



Techniques : Digests : Tips

- Choose carefully the scope to perform your digest on
- Place guards around manual digests
- Consider rendering directly to the DOM
- Favour `$scope.$digest()` over `$scope.$apply()`

4 Summary

Summary

- Write for humans first, then the computer
- Do not start by optimising
- Identify low hanging fruit using Chrome dev tools
- Mind the AngularJS digest