Making ExtJS Perform

How to avoid speed impact

Presented by Brandon Ryall

Question: What types of things can impact our speed?

Question: What types of things can impact our speed?

- Network latency
  - A slow network can have a major impact on our usability, many things such as rendering to data load.

Question: What types of things can impact our speed?

- Network latency
  - A slow network can have a major impact on our usability, many things such as rendering to data load.
- CSS Selectors
  - Improper selectors can slow down our DOM traversal

Question: What types of things can impact our speed?

- Network latency
    - A slow network can have a major impact on our usability, many things such as rendering to data load.
- CSS Selectors
    - Improper selectors can slow down our DOM traversal
- Dom Manipulation
    - Creating panels within panels create unnecessary DOM nesting

Question: What types of things can impact our speed?

- Network latency
  - A slow network can have a major impact on our usability, many things such as rendering to data load.
- CSS Selectors
  - Improper selectors can slow down our DOM traversal
- Dom Manipulation
  - Creating panels within panels create unnecessary DOM nesting
- Slow JavaScript Execution
  - Unoptimized javascript can make the app sluggish and create a terrible user-experience

Network Issues

One of the

biggest

issues we run into when using libraries and frameworks are

slow

load times.

Network Issues

Why?

- We’re loading tons of JavaScript files at once
- We’re loading tons of CSS at once
- We’re loading tons of images at once

Network Issues

So then if we have all of these slow loading issues, why do we continue to use it?

Network Issues

So then if we have all of these slow loading issues, why do we continue to use it?

The simple answer is that it works. And it works well.

Network Issues

So then if we have all of these slow loading issues, why do we continue to use it?

The simple answer is that it works. And it works well.

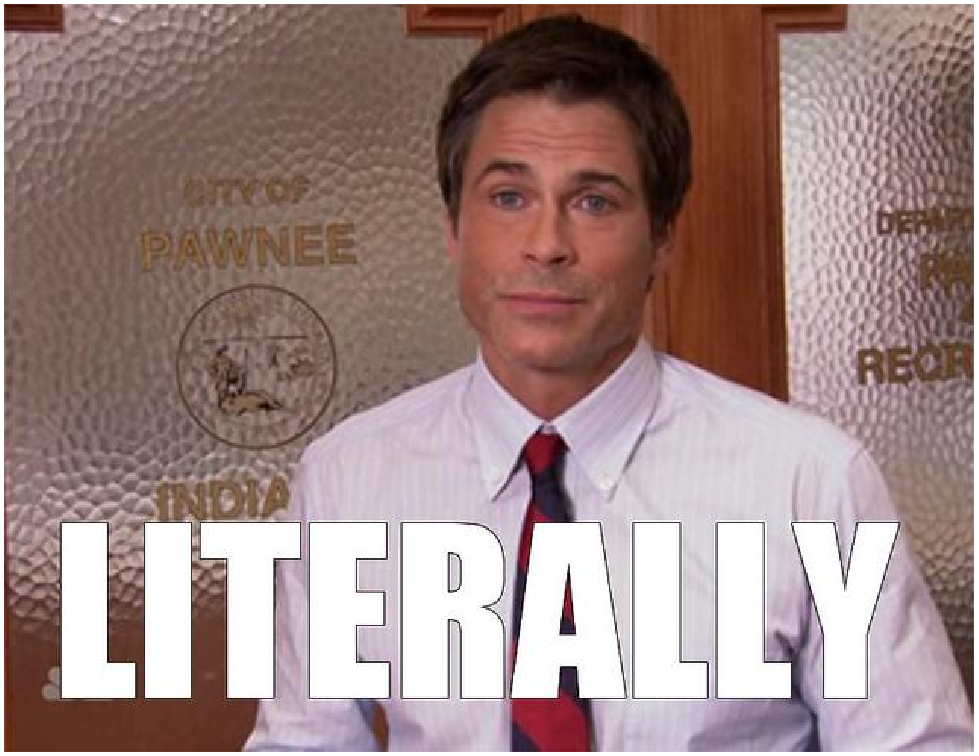
The problem is that we're just

doing it wrong.

Network Issues

Really really really wrong.

Like really.



Network Issues

But we can fix it!

Network Issues

But we can fix it!

How?

- Use the bootstrap, it is your friend

Network Issues

But we can fix it!

How?

- Use the bootstrap, it is your friend
- Reduce the number of JS files needed to be loaded by using the Ext.Loader and Ext.require

Network Issues

But we can fix it!

How?

- Use the bootstrap, it is your friend
- Reduce the number of JS files needed to be loaded by using the Ext.Loader and Ext.require
- Compress and obfuscate the files into a single cacheable file

Network Issues

But we can fix it!

How?

- Use the bootstrap, it is your friend
- Reduce the number of JS files needed to be loaded by using the Ext.Loader and Ext.require
- Compress and obfuscate the files into a single cacheable file
- Use SASS to compress and compile that CSS into a single file

Network Issues

But we can fix it!

How?

- Use the bootstrap, it is your friend
- Reduce the number of JS files needed to be loaded by using the Ext.Loader and Ext.require
- Compress and obfuscate the files into a single cacheable file
- Use SASS to compress and compile that CSS into a single file
- Turn all of those images you're loading into a sprite

Network Issues

A real word example

When the wizard v2 was created, one of the things done was introduce compression/obfuscation

The results were pretty noticeable:

Network Issues

A real word example

When the wizard v2 was created, one of the things done was introduce compression/obfuscation

The results were pretty noticeable:

- Uncompressed development version
  - All JS loaded: ~3.2 seconds
  - Total load: ~9 seconds

Network Issues

A real word example

When the wizard v2 was created, one of the things done was introduce compression/obfuscation

The results were pretty noticeable:

- Uncompressed development version
  - All JS loaded: ~3.2 seconds
  - Total load: ~9 seconds
- Compressed/Light Obfuscation version
  - All JS loaded: ~400 milliseconds
  - Total load: ~6 seconds

CSS Selectors

Question: How will the browser handle the selector below?

```
div.nav > div[title=boo] span
```

CSS Selectors

Question: How will the browser handle the selector below?

```
div.nav > div[title=boo] span
```

CSS selectors read right to left, so `span` would be our key selector.

CSS Selectors

Question: How will the browser handle the selector below?

```
div.nav > div[title=boo] span
```

CSS selectors read right to left, so `span` would be our key selector.

Is our key selector efficient?

CSS Selectors

Question: How will the browser handle the selector below?

```
div.nav > div[title=boo] span
```

CSS selectors read right to left, so `span` would be our key selector.

Is our key selector efficient?

CSS Selectors

So what is the point of having a key selector?

CSS Selectors

So what is the point of having a key selector?

Key selectors help the browser quickly filter out a mismatched result to your selector

CSS Selectors

So what is the point of having a key selector?

Key selectors help the browser quickly filter out a mismatched result to your selector

So why is our key selector of `span` bad?

CSS Selectors

So what is the point of having a key selector?

Key selectors help the browser quickly filter out a mismatched result to your selector

So why is our key selector of `span` bad?

We’re essentially hitting every single span in the DOM.

It’s like trying to find your best friend Greg, when all Gregs in the world have no middle name nor last name.

It’s going to take a long time and slow you down.

CSS Selectors

So key selectors are pretty important.

“This is the key to dramatically increasing performance. The fewer rules required to check for a given element, the faster style resolution will be.”

David Hyatt, Mozilla

CSS Selectors

So which selectors are most efficient?

1. id (#myID)
2. class (.myClass)
3. tag (div, span, h1)
4. adjacent sibling (div + p)
5. child (ul > li)
6. descendant (li a)
7. universal (*)
8. attribute (a[title=booger])
9. pseudo-class & pseudo-element (a:hover, li:first)

(ordered best to worst)

For ExtJS, we should try to use a class for most of our components since we do not use IDs.

A Better DOM

Question: What is wrong with the Ext structure below?

```
...
{
       xtype: ‘panel’,
       anchor: ‘100%’,
       layout: ‘hbox’,
       items: [
              {
                     xtype: ‘panel’,
                     flex: 1,
                     html: ‘Panel 1’
              },
              {
                     xtype: ‘container’,
                     flex: 2,
                     html: ‘Panel 2’
              }
       ]
}
...
```

A Better DOM

Question: What is wrong with the Ext structure below?

```
...
{
       xtype: 'panel',
       anchor: '100%',
       layout: 'hbox',
       items: [
              {
                     xtype: 'panel',
                     flex: 1,
                     html: 'Panel 1'
              },
              {
                     xtype: 'container',
                     flex: 2,
                     html: 'Panel 2'
              }
       ]
}
...
```

Why use a panel with extra overhead we don't need?

A Better DOM

Question: What is wrong with the Ext structure below?

```
...
{
        xtype: 'panel',
        anchor: '100%',
        layout: 'hbox',
        items: [
                {
                        xtype: 'panel',
                        flex: 1,
                        html: 'Panel 1'
                },
                {
                        xtype: 'container',
                        flex: 2,
                        html: 'Panel 2'
                }
        ]
}
...
```

Why use a panel with extra overhead we don't need?

When we have no need for extra features (ie: docking bars, extra event listeners, title) then we should be using a simple container. Containers are essentially a single div and free up a lot of the overhead a panel creates.

A Better DOM

It’s important that we always remember to reduce our container/component nesting to the shallowest level possible to prevent redundant layout runs.

A Better DOM

It’s important that we always remember to reduce our container/component nesting to the shallowest level possible to prevent redundant layout runs.

Remember:

A Better DOM

It’s important that we always remember to reduce our container/component nesting to the shallowest level possible to prevent redundant layout runs.

Remember:

Layout calls are

expensive!

A Better DOM

Question: What is wrong with this Ext structure?

```
…
{
        xtype: 'tabpanel',
        items: [
                {
                        title: 'Tab 1',
                        items: [
                                {
                                        xtype: 'grid',
                                        ...
                                }
                        ]
                }
        ]
}
...
```

A Better DOM

Question: What is wrong with this Ext structure?

```
…
{
        xtype: ‘tabpanel’,
        items: [
                {
                        title: ‘Tab 1’,
                        items: [
                                {
                                        xtype: ‘grid’,
                                        ...
                                }
                        ]
                }
        ]
}
...
```

Do I really need this panel?

A Better DOM

Question: What is wrong with this Ext structure?

```
…
{
        xtype: ‘tabpanel’,
        items: [
                {
                        title: ‘Tab 1’,
                        items: [
                                {
                                        xtype: ‘grid’,
                                        ...
                                }
                        ]
                }
        ]
}
...
```

Do I really need this panel?

Nope.

A Better DOM

Question: What is wrong with this Ext structure?

```
…
{
        xtype: ‘tabpanel’,
        items: [
                {
                        title: ‘Tab 1’,
                        items: [
                                {
                                        xtype: ‘grid’,
                                        ...
                                }
                        ]
                }
        ]
}
...
```

So how do I fix this guy then?

A Better DOM

Question: What is wrong with this Ext structure?

```
...
{
      xtype: 'tabpanel',
      items: [
            {
                  xtype: 'grid',
                  title: 'Tab 1',
                  ...
            }
      ]
}
...
```

So how do I fix this guy then?

- Remove the panel
- Move our title to our grid

A Better DOM

“But Brandon, why is this important!”

- all of you right now.

A Better DOM

“But Brandon, why is this important!”

- all of you right now.

This is very important because we have to remember, each component has to render itself and run the layout manager. Having these nested structures is **expensive** and time consuming.

A Better DOM

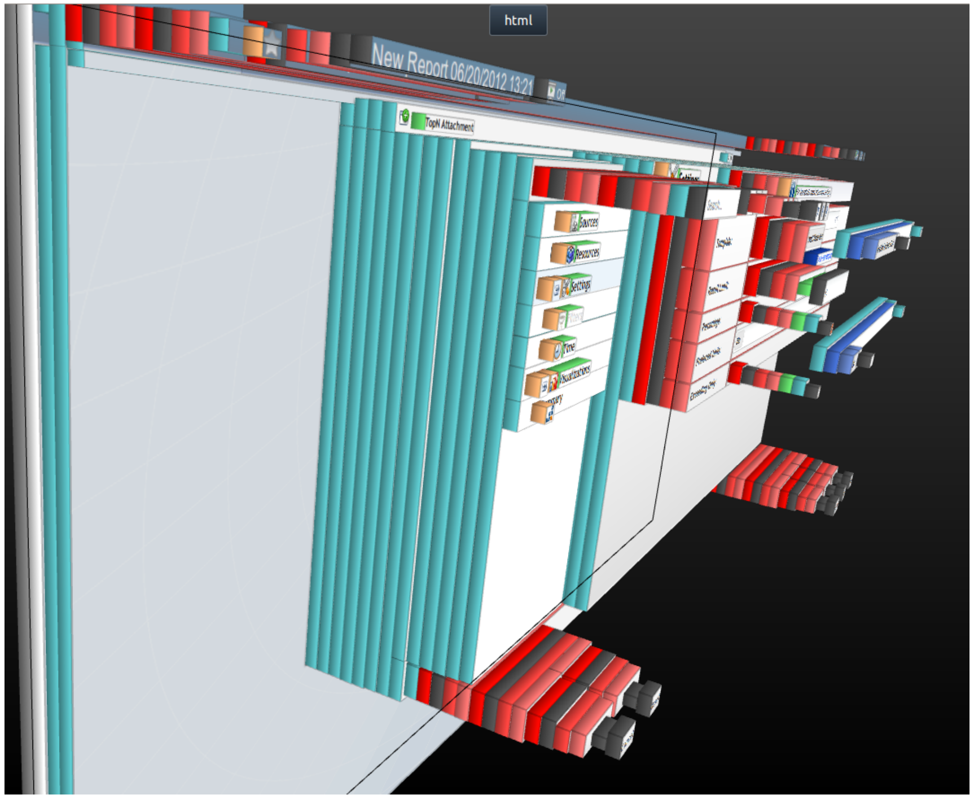
Things to remember and avoid:

- shrink wrapping (auto sizing based on content) where possible
- size constraints (minHeight, maxHeight, minWidth, maxWidth) where possible

A Better DOM

How about a real world example?

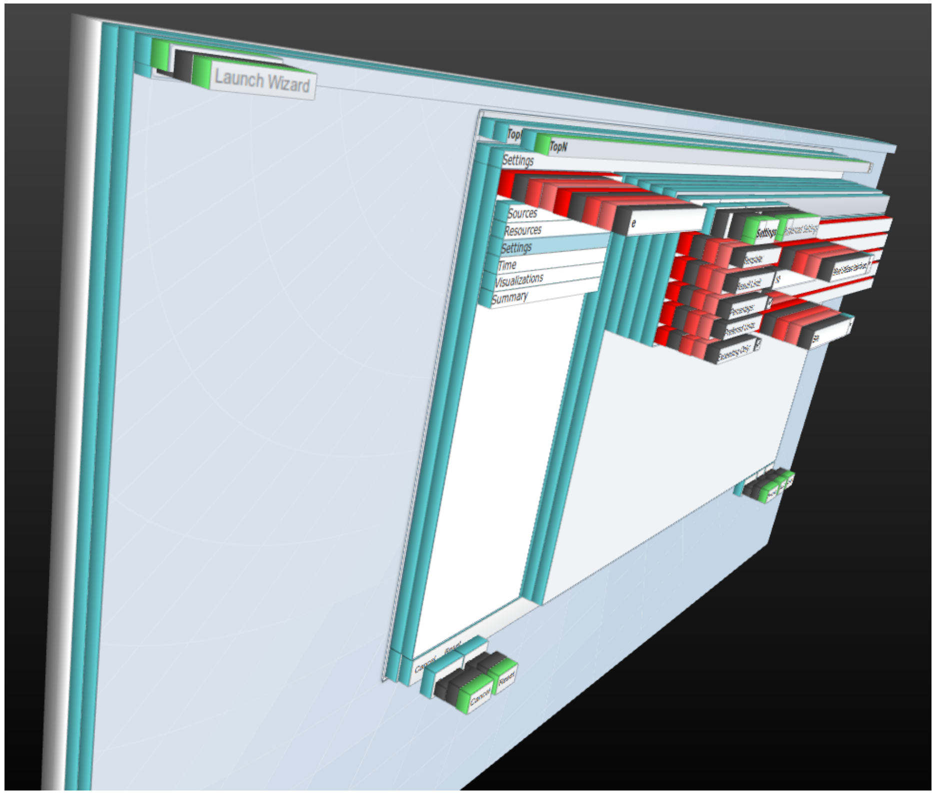
Here is our wizard version 1, using ext3.



A Better DOM

How about a real world example?

And here we are in ext4 using goodness



A Better DOM

Awesome sauce, now we're in business!

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently
- When a method is executed during render or layout time

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently
- When a method is executed during render or layout time
- ...Although it’s probably best to not do extra activity during render

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently
- When a method is executed during render or layout time
- ...Although it’s probably best to not do extra activity during render
- Cache variables when possible

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently
- When a method is executed during render or layout time
- ...Although it’s probably best to not do extra activity during render
- Cache variables when possible
- Avoid using Ext.each

Optimizing JavaScript

Question: When would be the best time to optimize our JavaScript?

- When methods are repeated frequently
- When a method is executed during render or layout time
- ...Although it’s probably best to not do extra activity during render
- Cache variables when possible
- Avoid using Ext.each
- Others that I can’t think of off the top of my head

Optimizing JavaScript

What’s the deal about Ext.each()?

Optimizing JavaScript

Here at SevOne, we pride ourselves on our speed: True or False?

Optimizing JavaScript

Here at SevOne, we pride ourselves on our speed: **True** or False?

...So why would we use the worst looping constructs we can?

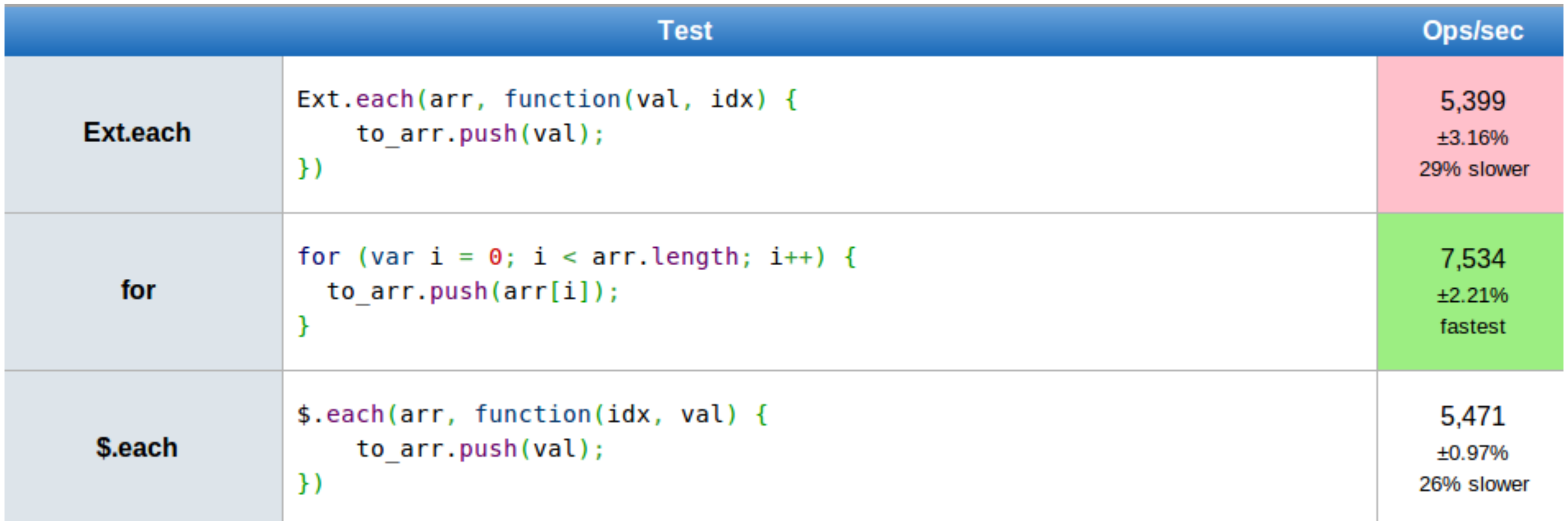
Optimizing JavaScript

Lets look at the numbers...

Optimizing JavaScript

Lets look at the numbers…

Ext.each() - for - $.each

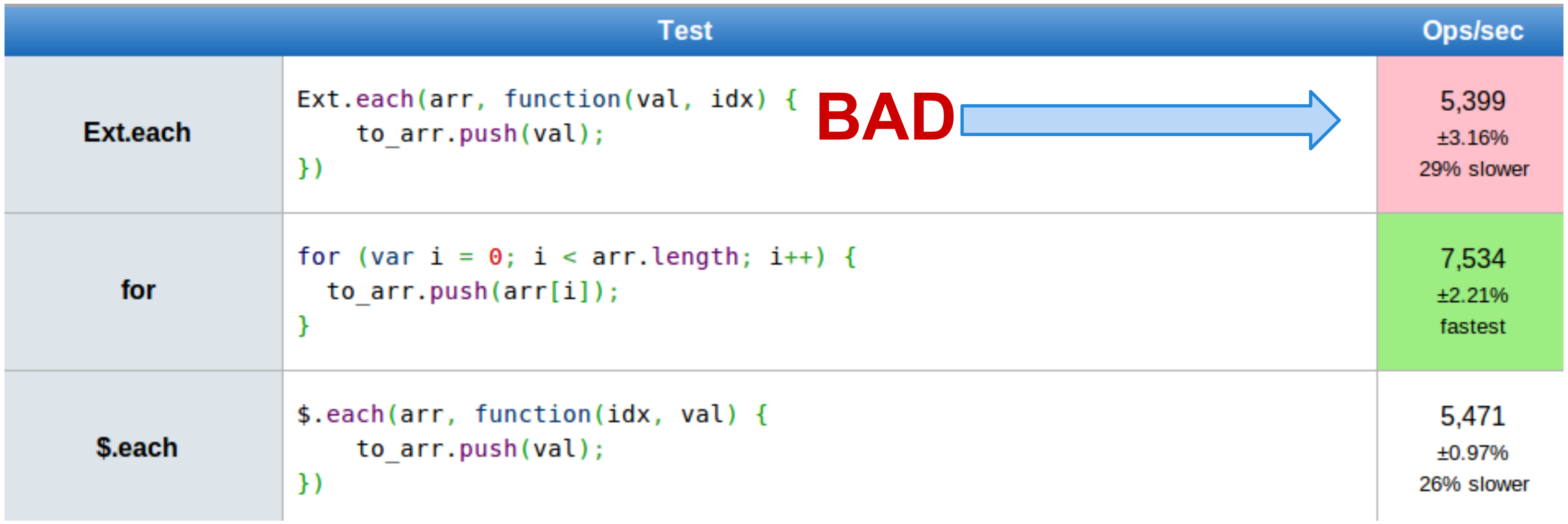


Test		Ops/sec
Ext.each	`Ext.each(arr, function(val, idx) { to_arr.push(val); })`	5,399 ±3.16% 29% slower
for	`for (var i = 0; i < arr.length; i++) { to_arr.push(arr[i]); }`	7,534 ±2.21% fastest
$.each	`$.each(arr, function(idx, val) { to_arr.push(val); })`	5,471 ±0.97% 26% slower

Optimizing JavaScript

Lets look at the numbers…

Ext.each() - for - $.each



	Test	Ops/sec

Ext.each	`Ext.each(arr, function(val, idx) {`
	`    to_arr.push(val);`
	`})`
		5,399
		±3.16%
		29% slower

for	`for (var i = 0; i < arr.length; i++) {`
	`  to_arr.push(arr[i]);`
	`}`
		7,534
		±2.21%
		fastest

$.each	`$.each(arr, function(idx, val) {`
	`    to_arr.push(val);`
	`})`
		5,471
		±0.97%
		26% slower

BAD

Optimizing JavaScript

So knowing that Ext.each is ridiculously slow, how can we improve the code below?

```
var myStore = Ext.getStore('Devices');
myStore.each(function() {
    console.log( this );
});
```

Optimizing JavaScript

So knowing that Ext.each is ridiculously slow, how can we improve the code below?

```
var myStore = Ext.getStore(‘Devices’),
    data = myStore.data;
for( var i = 0; i < data.length; i++ ) {
    console.log( data[ i ] );
}
```

Optimizing JavaScript

Optimizing JavaScript

So knowing that Ext.each is ridiculously slow, how can we improve the code below?

```
var myStore = Ext.getStore('Devices'),
    data = myStore.data,
    len = data.length;
for( var i = 0; i < len; i++ ) {
    console.log( data[ i ] );
}
```

Optimizing JavaScript

Wait...is there a faster loop?

Optimizing JavaScript

Wait...is there a faster loop?

There might be.

Mind if I have a buddy of mine look at it?

Optimizing JavaScript

Every time browsers change their engines, the speed differences appear to change.

Optimizing JavaScript

For instance, decrementing while loops were a power player for a long time

```
var i = 10000;
while( i-- ) {
   console.log( i );
}
```

However, this is no longer the case…

Optimizing JavaScript

Current benchmarks show that cached for loops are the most powerful.

```
var arr = [ … ]; // assume lots...
for (var i = 0, len = arr.length; i < len; i++) {
   console.log( i );
}
```

I’ll always love you while loops...

Optimizing JavaScript

So to reiterate some rules for optimizing

Optimizing JavaScript

So to reiterate some rules for optimizing

- Use the best loops possible

Optimizing JavaScript

So to reiterate some rules for optimizing

- Use the best loops possible
- Cache variables, components, etc. when possible

Optimizing JavaScript

So to reiterate some rules for optimizing

- Use the best loops possible
- Cache variables, components, etc. when possible
- Optimize frequently repeated methods

Optimizing JavaScript

So to reiterate some rules for optimizing

- Use the best loops possible
- Cache variables, components, etc. when possible
- Optimize frequently repeated methods
- Avoid strenuous actions during render

Almost there...

Practices to Avoid

- Unnecessary nesting of component structures

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure
- Global variables

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
    - Example: Not caching a window component
- Huge controllers
    - Remember, each view should really have its own
- Bad folder structure
- Global variables
- Using id

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure
- Global variables
- Using id
- Not creating references
  - By creating refs you are essentially caching the component

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure
- Global variables
- Using id
- Not creating references
  - By creating refs you are essentially caching the component
- Not sticking with a standard naming convention

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure
- Global variables
- Using id
- Not creating references
  - By creating refs you are essentially caching the component
- Not sticking with a standard naming convention
- Making your code overly complicated
  - D.R.Y and K.I.S.S

Practices to Avoid

- Unnecessary nesting of component structures
- Creating memory leaks created by improper component usage
  - Example: Not caching a window component
- Huge controllers
  - Remember, each view should really have its own
- Bad folder structure
- Global variables
- Using id
- Not creating references
  - By creating refs you are essentially caching the component
- Not sticking with a standard naming convention
- Making your code overly complicated
  - D.R.Y and K.I.S.S
- Forgetting to add the MVC files to Ext.require

That’s it

Questions?