

One of the great features of the modern web is that sites work on any device with a browser. This session will focus on creating UIs for the cross-device web. We will cover building web sites that support multiple device form factors (responsive and non-responsive approaches), discuss single page sites and some of the layout features in modern mobile browsers, and do a deep dive into multi-touch input on the web. Finally, we'll show some of the awesome new mobile debugging tools in Chrome and Chrome for Android



About Boris Smus

[View full profile](#)

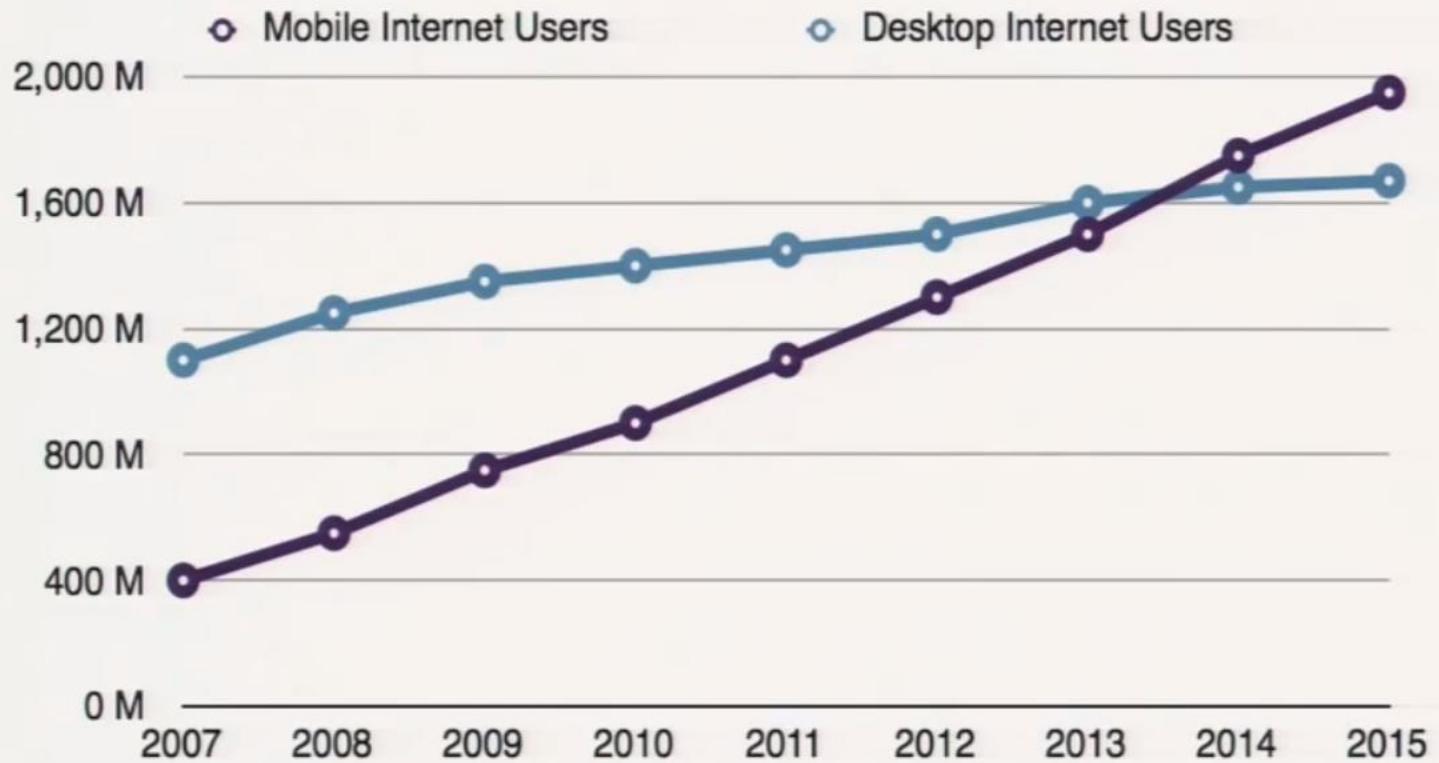
Boris is a UI engineer on the Google Chrome developer relations team. Before joining Google, he lived on a tropical island in the Atlantic.



# Fast user interfaces for the cross-device web

Boris Smus  
Developer Programs Engineer, Google

# Who cares?



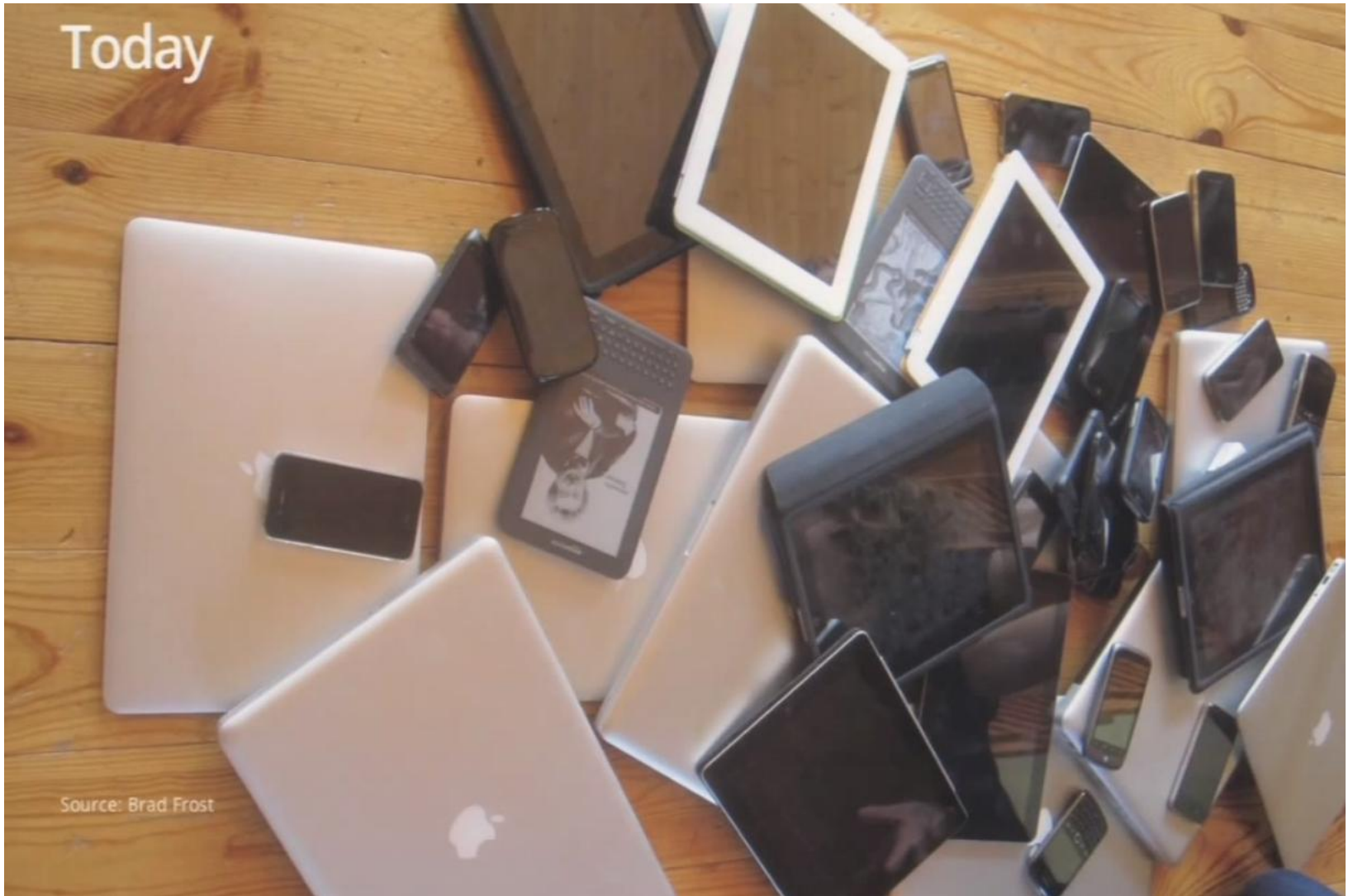
Source: [Morgan Stanley Research](#)

## Terminology: "cross-device"

Questions about "mobile":

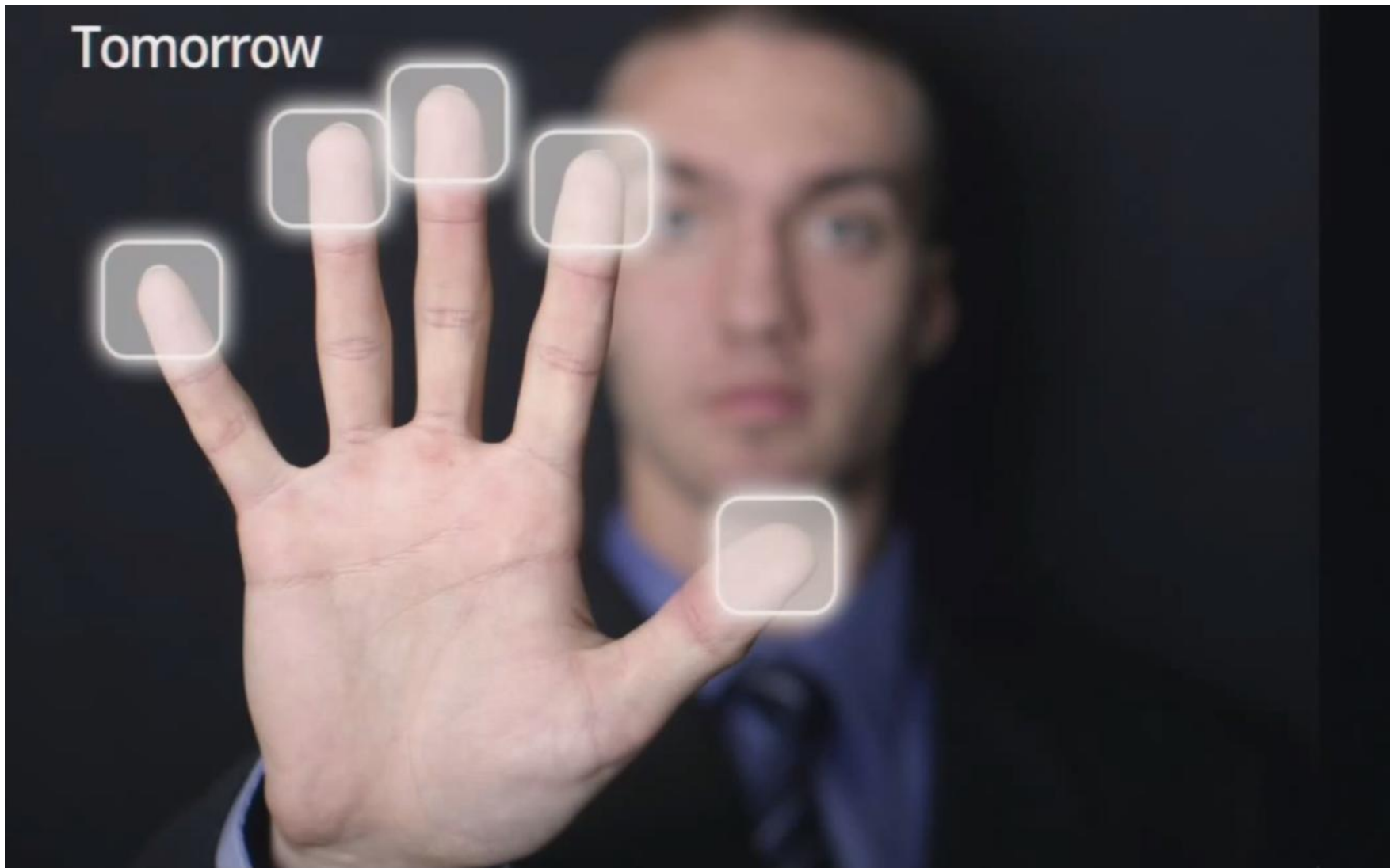
- What about tablets?
- What about other kinds of devices like TV/Cars/etc?
- What's the opposite of mobile?

Today



Source: Brad Frost

Tomorrow



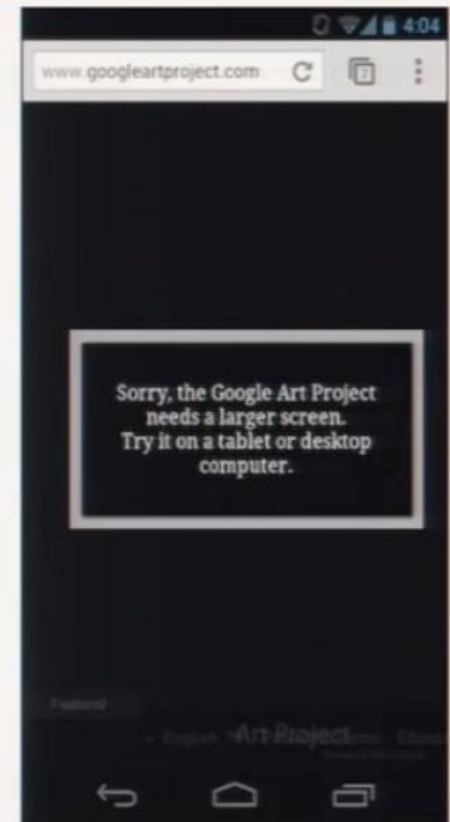
# What are the issues?

- Variety of form factors.
- CPUs and networks are slow.
- New kinds of input (multi-touch).
- Development with devices.

Supporting multiple devices



# Extreme 1: One version to rule them all



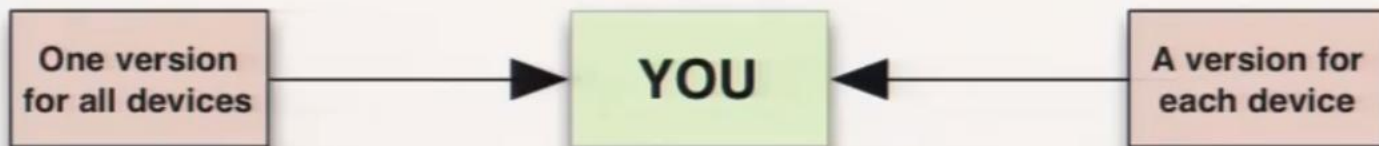
## Extreme 2: A version for each device

- HTC One
- Galaxy Nexus
- HTC Rezound
- iPhone
- Blackberry Bold
- Blackberry Curve
- iPad
- Amazon Kindle
- Sony W810i
- Nexus S
- Lumia 900
- ... (\* browsers)

And many many more...

# Seek middle ground

The more versions you create, the better each of them can be from a UX and performance perspective, but the more overall effort it will require.



- Cross-device means both platforms and form factors.
- Which one can we save on?

# Platforms

Human-interface guidelines (HIG) and UI frameworks are great!

- Frameworks make it easy.
- HIGs make it consistent.

Goal: pit of success!



# Beware emulating platform guidelines on web

Problems of emulating native UI on the web:

- Hard to implement.
- Perpetually *slightly off*.
- Doesn't look right on other platforms.

iOS navigation bar Back button



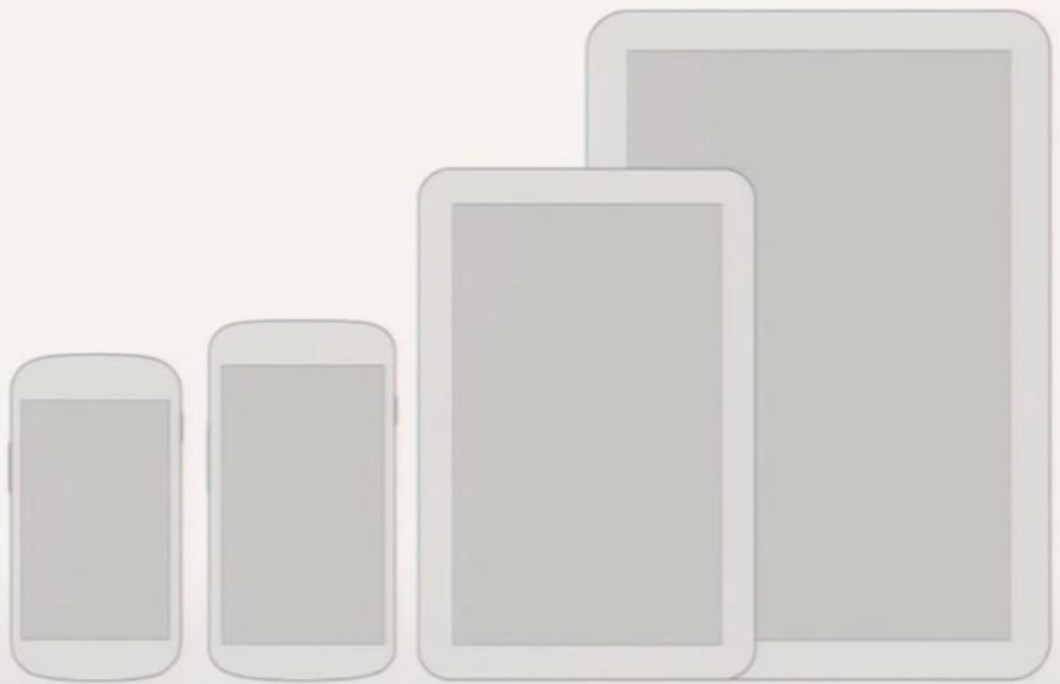
CSS

Native

Detailed study trying to do this for iOS: read the article on [cheeaun.com](http://cheeaun.com).

# Form factor differences

- Variable usage patterns (one vs two handed).
- Significant variation in screen real estate.



# Compromise: phone tablet desktop



- Approach 1: form-factor tweaks to a single version.
- Approach 2: form-factor specific versions.

# Approach 1: One adaptive version

Tweak layout with media queries:

```
@media screen and (max-width: 1000px) {  
  #sidebar { display: none; }  
}
```

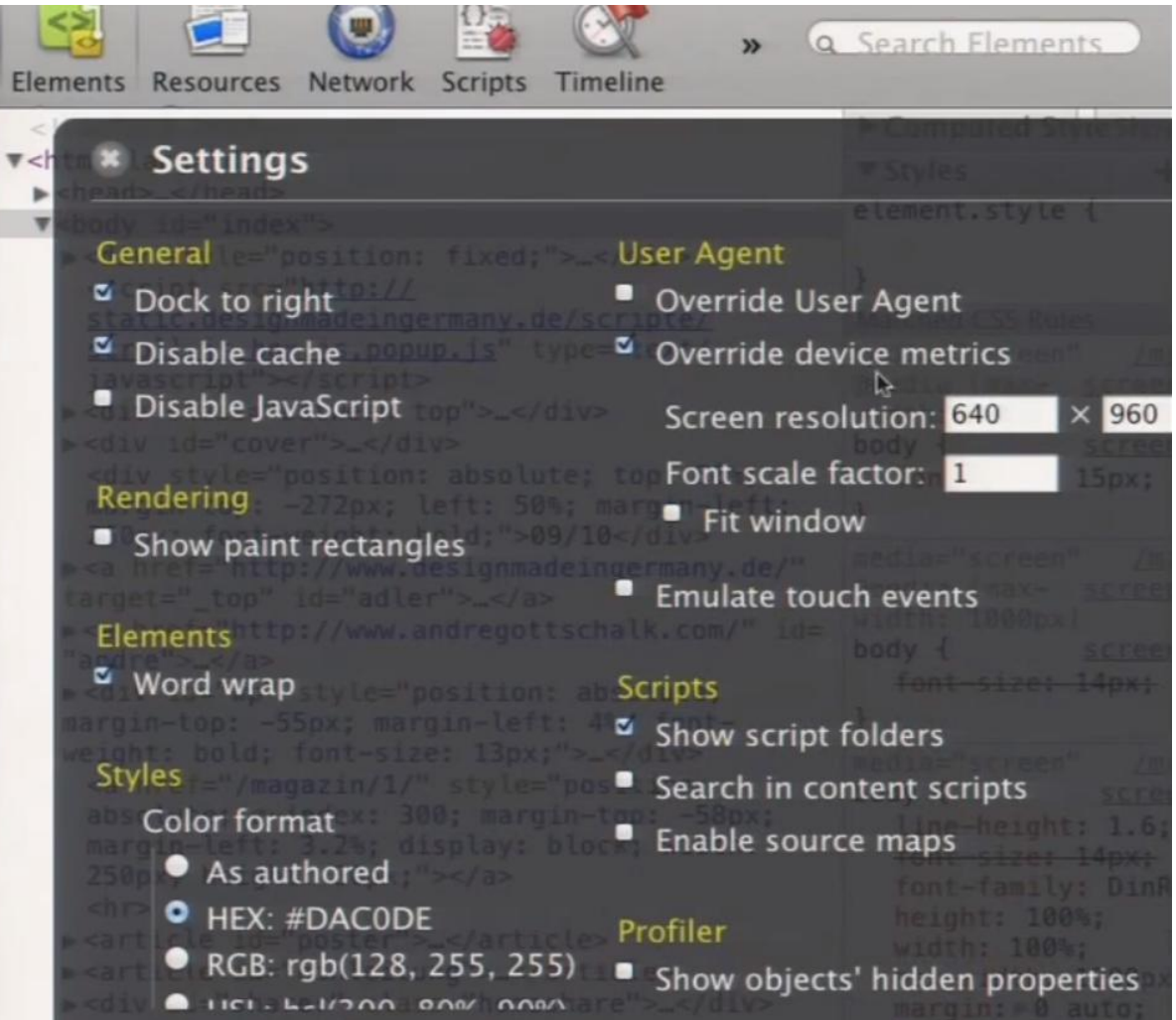
CSS

[Drastic example of responsive web design.](#)



09/10





# Media query limitations

- Shared DOM, so making big changes between form factors is hard:



- How to load additional form-factor specific functionality?

# Media queries in JavaScript

`window.matchMedia` lets you evaluate arbitrary media queries:

```
var result = window.matchMedia('(min-width: 500px)');  
result.matches // True iff width > 500px.
```

JAVASCRIPT

You can also listen for changes to match:

```
function onOrientationChange(mql) {  
  var isLandscape = mql.matches;  
}  
var mql = window.matchMedia("(orientation: landscape)");  
mql.addListener(onOrientationChange);
```

JAVASCRIPT

## Approach 2: Separate versions

JAVASCRIPT

```
function detectFormFactor() {  
  var device = DESKTOP;  
  if (hasTouch()) {  
    device = isSmall() ? PHONE : TABLET;  
  }  
  return device;  
}  
  
function hasTouch() {  
  return Modernizr.touch; // Soon: (hover: 0) and (pointer: coarse)  
}  
  
function isSmall() {  
  return window.matchMedia('(min-device-width: ???)').matches;  
}
```

(hover: 0) and (pointer: coarse)



# Distinguishing tablets from phones

'\*' denotes double-density devices

device pixels != css pixels

`devpx = window.devicePixelRatio * csspx`

How about landscape mode?



# Draw the line at 650px



# Correctly specifying multiple versions

HTML

```
<head>
  <link rel="alternate" href="http://foo.com" id="desktop"
        media="only screen and (hover: 1) and (pointer: fine)">
  <link rel="alternate" href="http://m.foo.com" id="phone"
        media="only screen and (max-device-width: 650px)">
  <link rel="alternate" href="http://tablet.foo.com" id="tablet"
        media="only screen and (min-device-width: 651px)">
</head>
```

Read these Google Webmaster [smartphone guidelines](#).



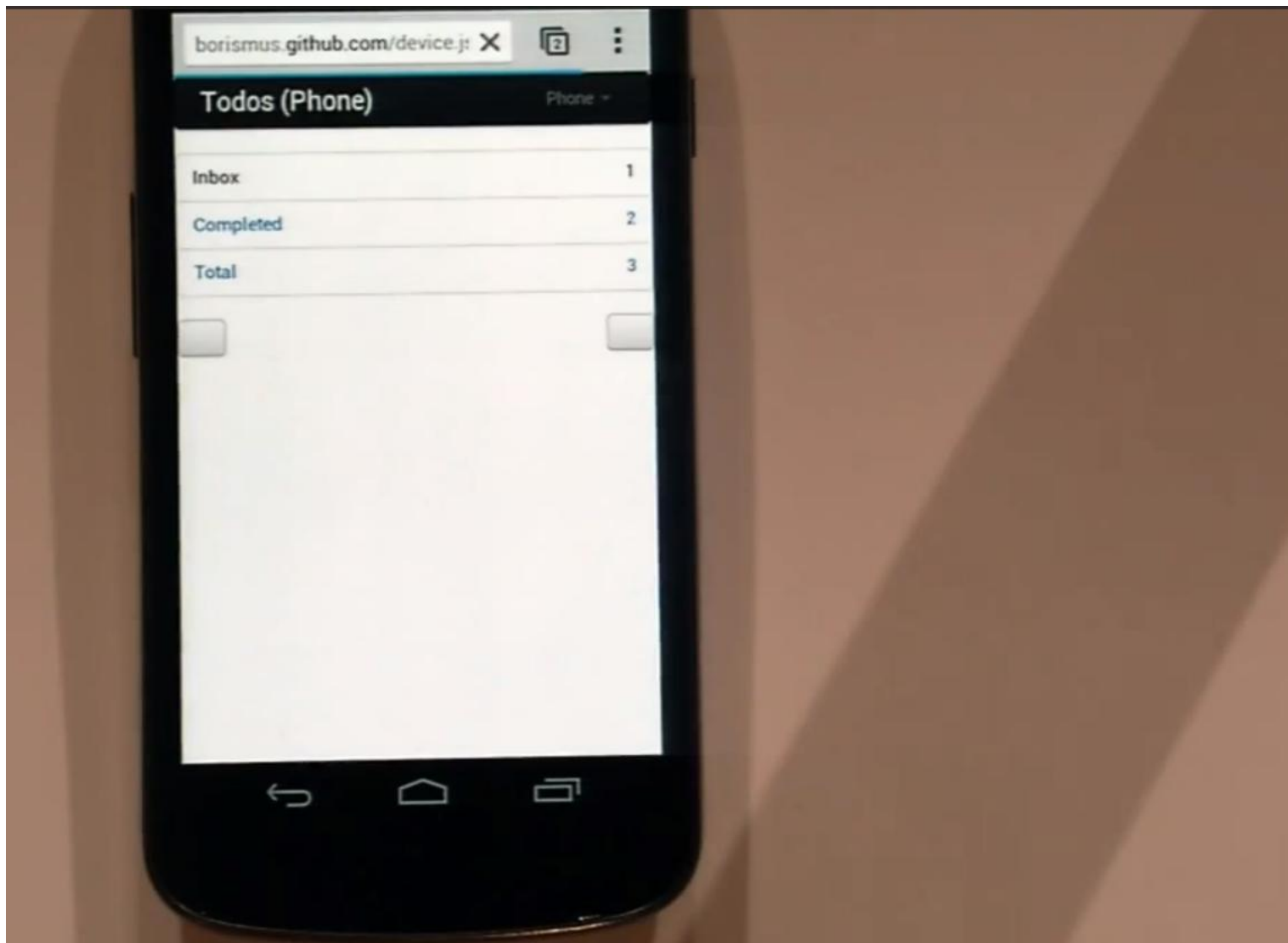
## Device.js: formalizing this approach

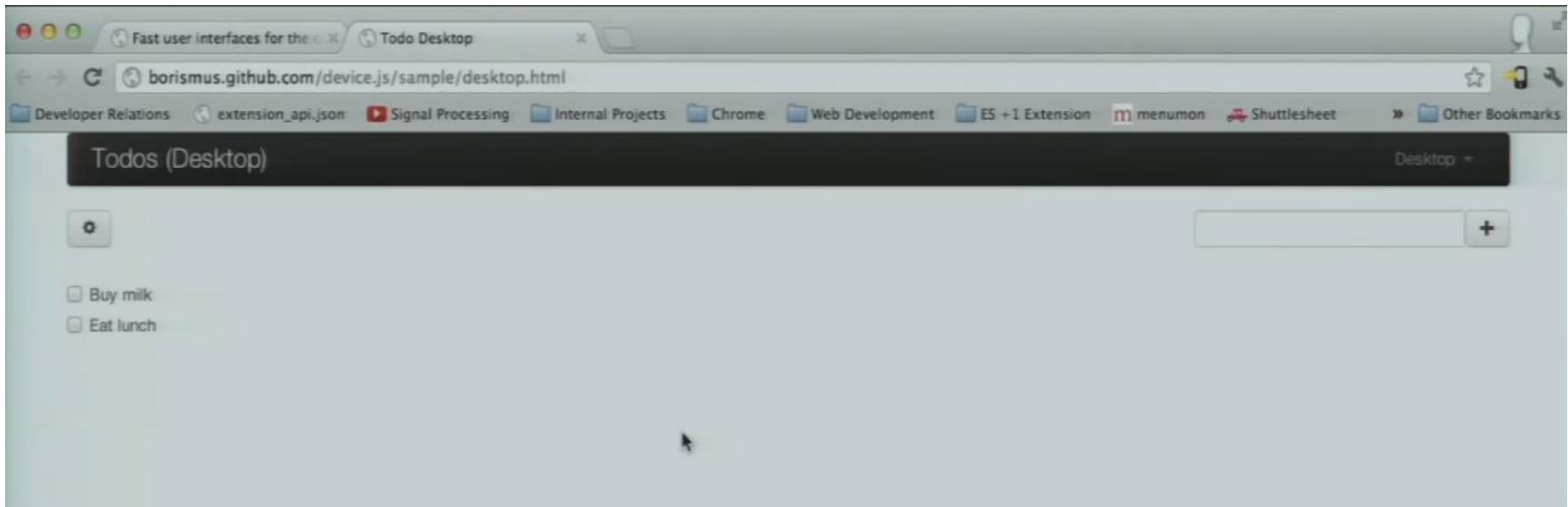
Use it to automatically redirect to the correct version based on `<link rel="alternate">` declarations.

- Built-in version forcing, and intelligent redirection.

[In action!](#)

For details, visit [github.com/borismus/device.js](https://github.com/borismus/device.js).





## Todos (Phone)

Phone -

Inbox

1

Completed

2

Total

3



# On the server

All we have is the User-Agent string.

**Mozilla/5.0 (Linux; Android 4.0.4; Galaxy Nexus Build/IMM76B)**

**AppleWebKit/535.19 (KHTML, like Gecko)**

**Chrome/18.0.1025.133 Mobile**

**Safari/535.19**

...and the user agent string was a complete mess, and near useless, and everyone pretended to be everyone else, and confusion abounded." - Aaron Andersen ([How the User Agent got its String](#))

[DevTools Demo.](#)



# Device databases

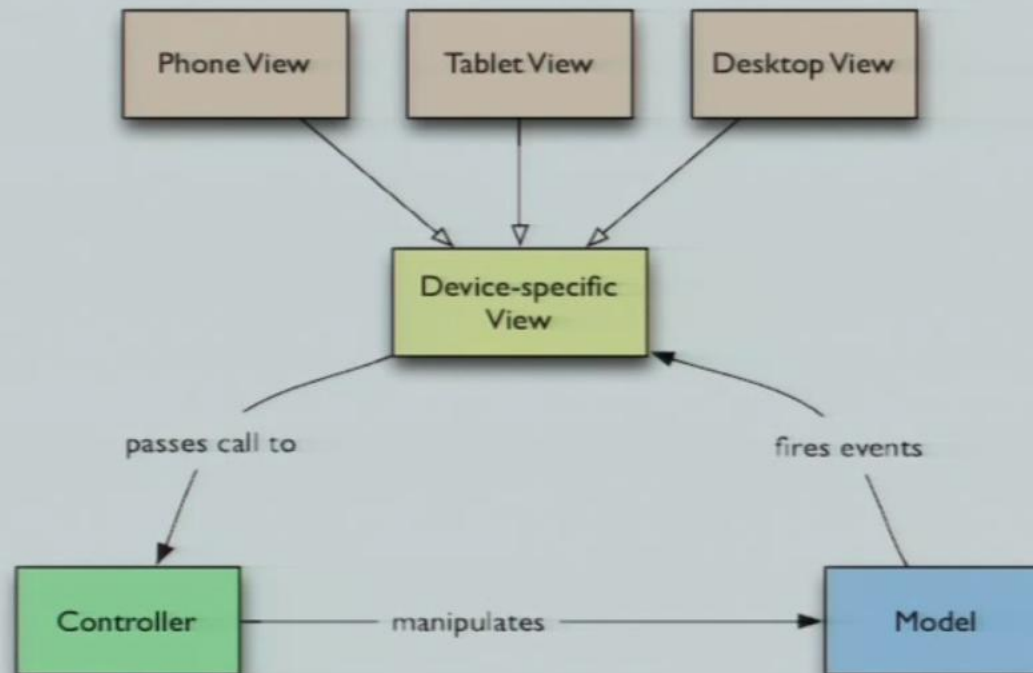
Solutions that make dealing with User-Agent strings easier (standalone and as a service API).

- DeviceAtlas at [deviceatlas.com](https://deviceatlas.com)
- WURFL at [wurfl.sourceforge.net](https://wurfl.sourceforge.net)

Drawbacks:

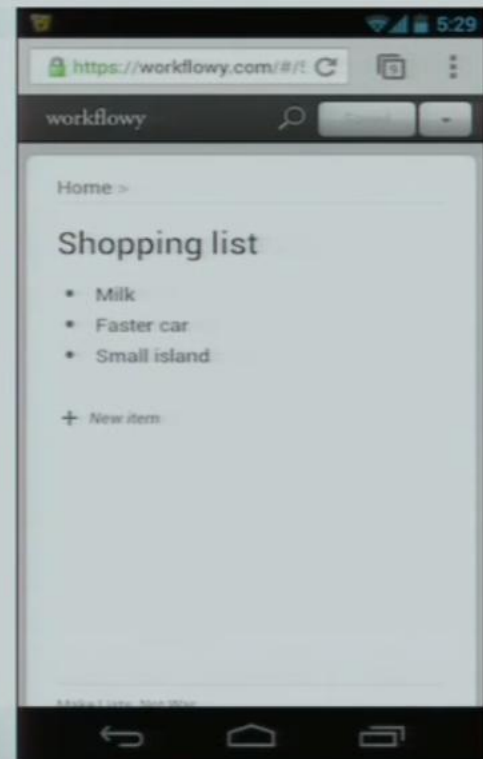
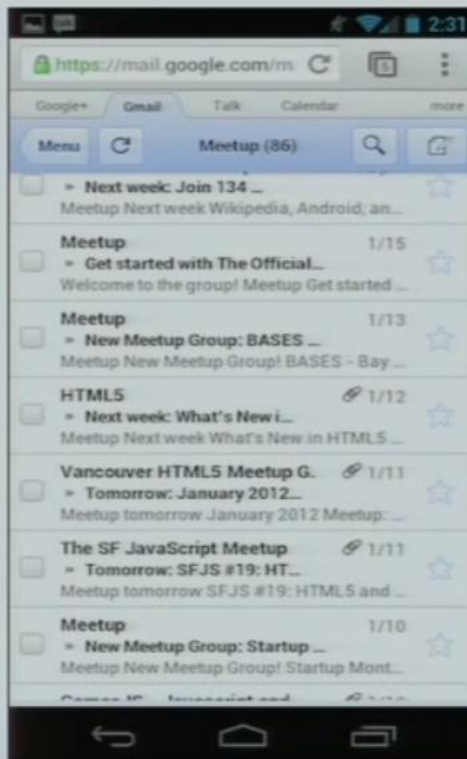
- Non-free [for commercial use](#).
- Not too simple to setup.

## Custom views → code reuse



# Single page sites

- Limited zooming.
- Fixed headers/footers.
- Smooth transitions.





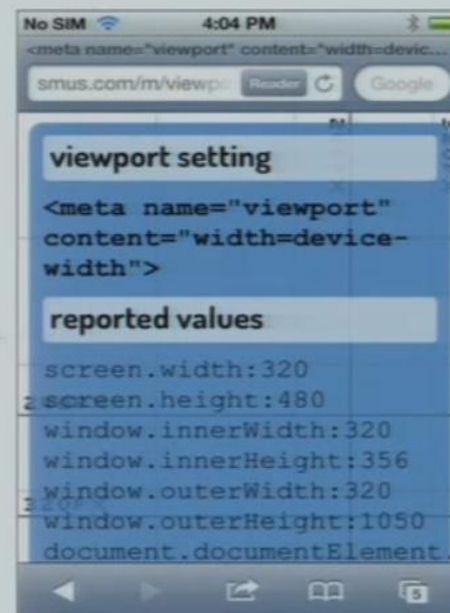
# Basic viewport

`<meta name="viewport" content="...">`. At minimum, set `width=device-width`.



← No viewport

Viewport →



Source: [Andreas Bovens](#)

# Prevent zooming

**Goal:** serve page at the right size, eliminating the need to pinch-zoom.

Use `initial-scale=1`, `minimum-scale=1` and `maximum-scale=1`:

```
<head>  
  <meta name="viewport"  
        content="width=device-width,initial-scale=1,minimum-scale=1,maximum-scale=1">  
</head>
```

HTML

Avoid `user-scalable=no`, since it's not well supported.

Use commas, not semicolons;

# Jank-free UIs

## Limit of human perception: 60 FPS

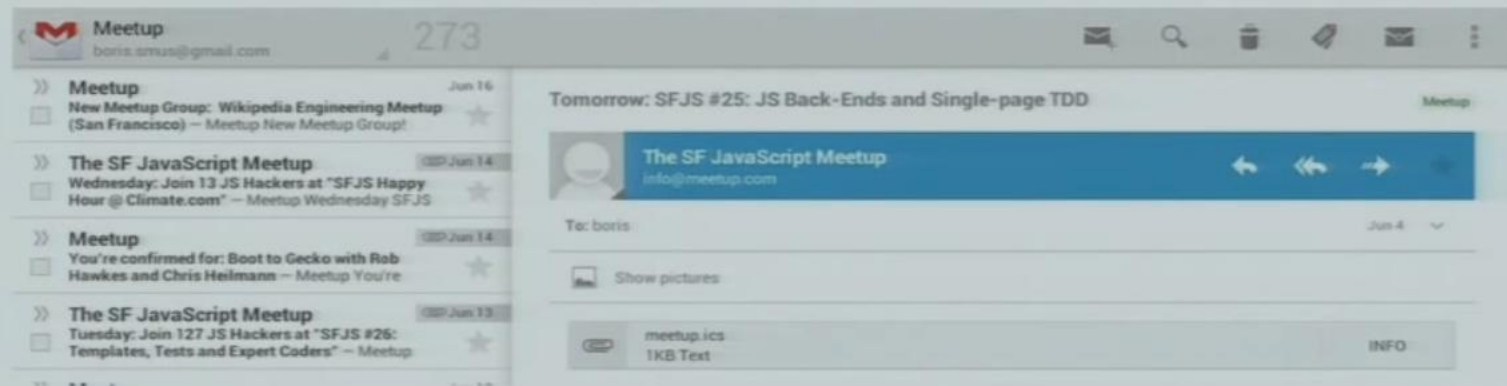
Bottom line: utilize hardware acceleration as much as possible.

Example: use `position: fixed` to keep things fixed to viewport.



# Scrolling sub-elements

For in-element scrolling, `overflow: auto;`



`-webkit-overflow-scrolling: touch;` forces hardware acceleration.

[In action!](#)

**overflow: hidden;**



**overflow: auto;**



**overflow: auto; -webkit-overflow-scrolling: touch;**



smus.com/m/scroll/

overflow: hidden; overflow: auto;

overflow: auto; -webkit-overflow-scrolling: touch;



Bookmarks > Mobile Bookmarks > IO Talk



Pointer.js  
demo



Request  
Animation



CSS  
transitions



Mobile Web  
Scrolling



Device.js





# Advanced transforms

3D transformations, and perspective!

```
matrix3d(m00, m01, m02, m03,  
         m10, m11, m12, m13,  
         m20, m21, m22, m23,  
         m30, m31, m31, m33)
```

CSS

CSS interpolates for you: `transition: transform 1s ease-in-out;`

Note: `transition`, `transform` still need vendor prefixes!

[In action!](#)



## Touch != mouse

- No hover state
  - Multiple points
  - Less precise input
- 
- Fingers aren't (x, y), but also have shape
  - Future: Pressure? Haptic feedback? Hover?

# Touch events

events: touchstart, touchmove, touchend

properties: touches, targetTouches, changedTouches

There's a [specification](#)!

# Overriding the browser

Browser has built-in touch behavior (scrolling, tab switching).

How to override?

```
var el = document.querySelector('#my .selector');  
el.addEventListener('touchmove', function(event) {  
  event.preventDefault();  
});
```

JAVASCRIPT

In IE:

```
#my .selector {  
  -ms-touch-action: none;  
}
```

CSS

Some things can't be overridden, such as bevel swipe to switch tabs.

## Touch performance

Beware click delays!

The `click` and `mouse*` events are delayed by 300ms.

Use `touchend` for [faster buttons](#).

# Advanced touch performance

Multi-touch events come in faster than 60 FPS. Decouple input and draw:

JAVASCRIPT

```
var touches = [];  
canvas.addEventListener('touchmove', function(event) {  
    touches = event.touches;  
}, false);  
  
function draw() {  
    // Touch rendering code goes here...  
}  
  
// Setup a 60 FPS timer.  
var timer = setInterval(function() {  
    draw();  
}, 1000/60);
```

# Request Animation Frame

In Chrome, use `window.webkitRequestAnimationFrame` instead (see [shim](#)).

```
function draw(time) {  
  // Touch rendering code goes here...  
  requestId = window.requestAnimationFrame(draw);  
}  
function start() {  
  requestId = window.requestAnimationFrame(draw);  
}  
function stop() {  
  window.cancelAnimationFrame(requestId);  
}  
start();
```

JAVASCRIPT

[Demo!](#)

# Problem: Gestures are difficult to implement

Unfortunately `gesture*` events aren't widely implemented.

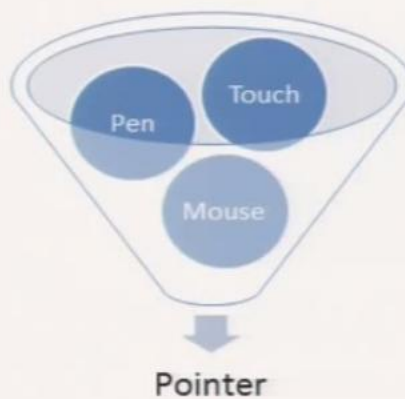
Robust pinch-zoom based on `touch*` events:

```
/**  
 * Gesture recognizer for compound multi-touch transformations.  
 *  
 * 1. pinch/zoom/scale gesture.  
 * 2. rotate gesture.  
 */  
  
function TransformRecognizer(element) {  
  // Reference positions for the start of the transformation.  
  this.referencePair = null;  
  
  // Bind touch event handlers to this element.  
  element.addEventListener('touchstart', this.touchStartHandler.bind(this));
```

JAVASCRIPT

# MSPointer events

Idea: consolidated input model.



Problems:

- Not well-specified yet.
- Another input mechanism to deal with (now `mouse*`, `touch*`, *and* `MSPointer*`)



# Pointer.js

Idea: consolidate input model across browsers (`pointerdown`, `pointermove`, `pointerup`).

```
var el = document.querySelector(mySelector);  
el.addEventListener('pointerdown', function(event) {  
  // ...  
});
```

JAVASCRIPT

Basically, abstraction layer around `mouse*`, `touch*` and `MSPointer*` events.

Get original event source was (touch or mouse) via `event.pointerType`.

[In action!](#)

# Gestures in Pointer.js

Pointer.js also provides gesture events built on top of pointer events.

```
var el = document.querySelector(mySelector);  
el.addEventListener('gesturescale', function(e) {  
  console.log('scale: ' + e.scale);  
});
```

JAVASCRIPT

For details, visit [github.com/borismus/pointer.js](https://github.com/borismus/pointer.js).

# Mobile web development

# Chrome DevTools ♥ mobile

Easiest path: emulate mobile on desktop.

- Emulate screen size.
- Override Chrome's User-Agent string.
- Emulate (single) touch events.

## Emulate multi-touch events on mac

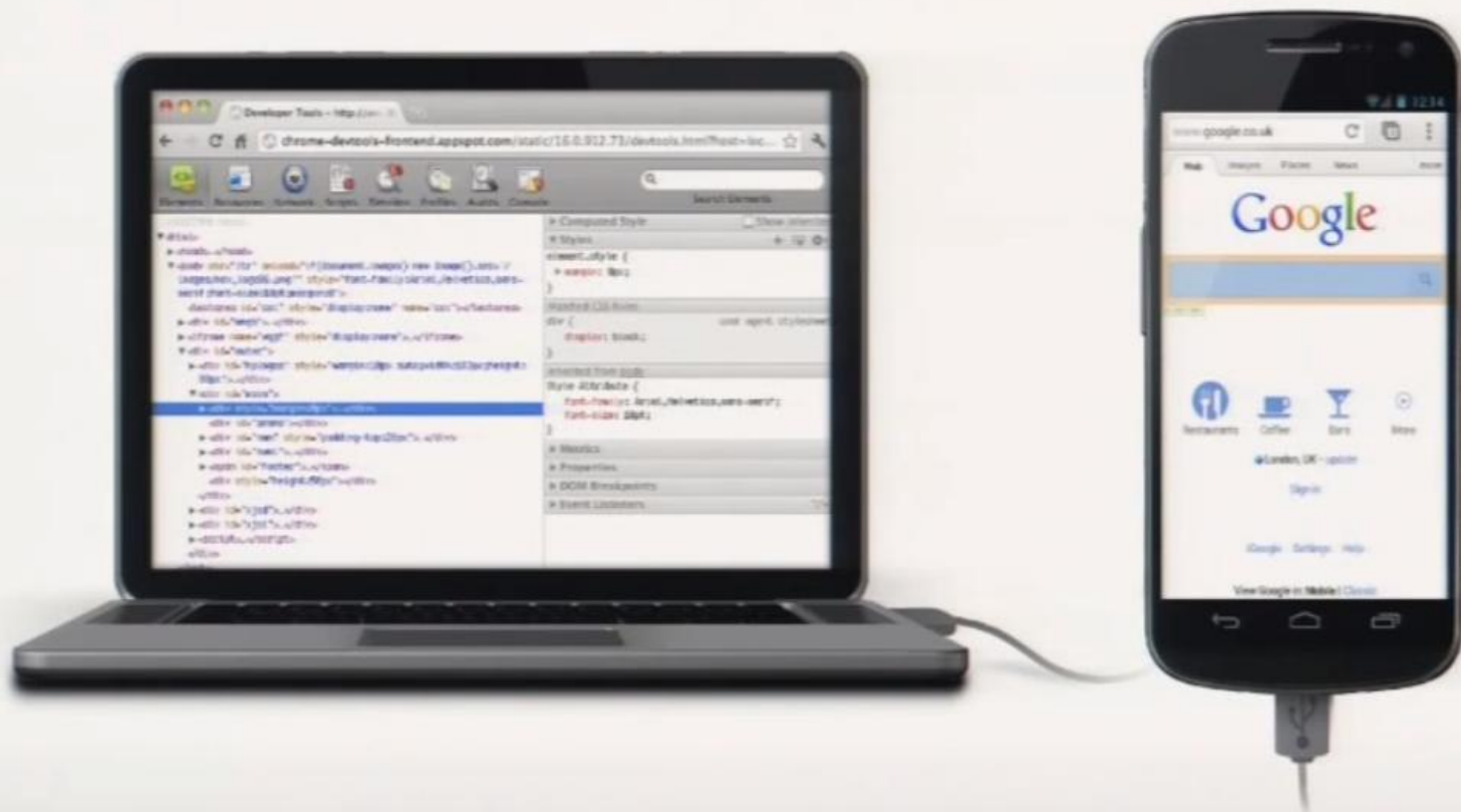
- Install NPAPI plugin.
- Run tracker application.
- Include `<script>` and `<object>` tags.
- Slice some browser logos.

In action!

For details, visit [github.com/borismus/magictouch](https://github.com/borismus/magictouch).

# Remote debugging!

Ultimately, you need to test on the device. [More info.](#)



# Thanks for coming!

- Chrome for Android [page on Google Developers](#)
- HTML5Rocks [mobile landing page](#)
- Stack Overflow questions [tagged android and google-chrome](#)
- [+Google Chrome Developers](#)

[+Boris Smus](#) / [@borismus](#)