ENGINEERING HTML5 APPLICATIONS FOR BETTER PERFORMANCE

LAURI SVAN @LAURISVAN
SC5 ONLINE @SC5

HTML5 EXPERTISE AT YOUR SERVICE

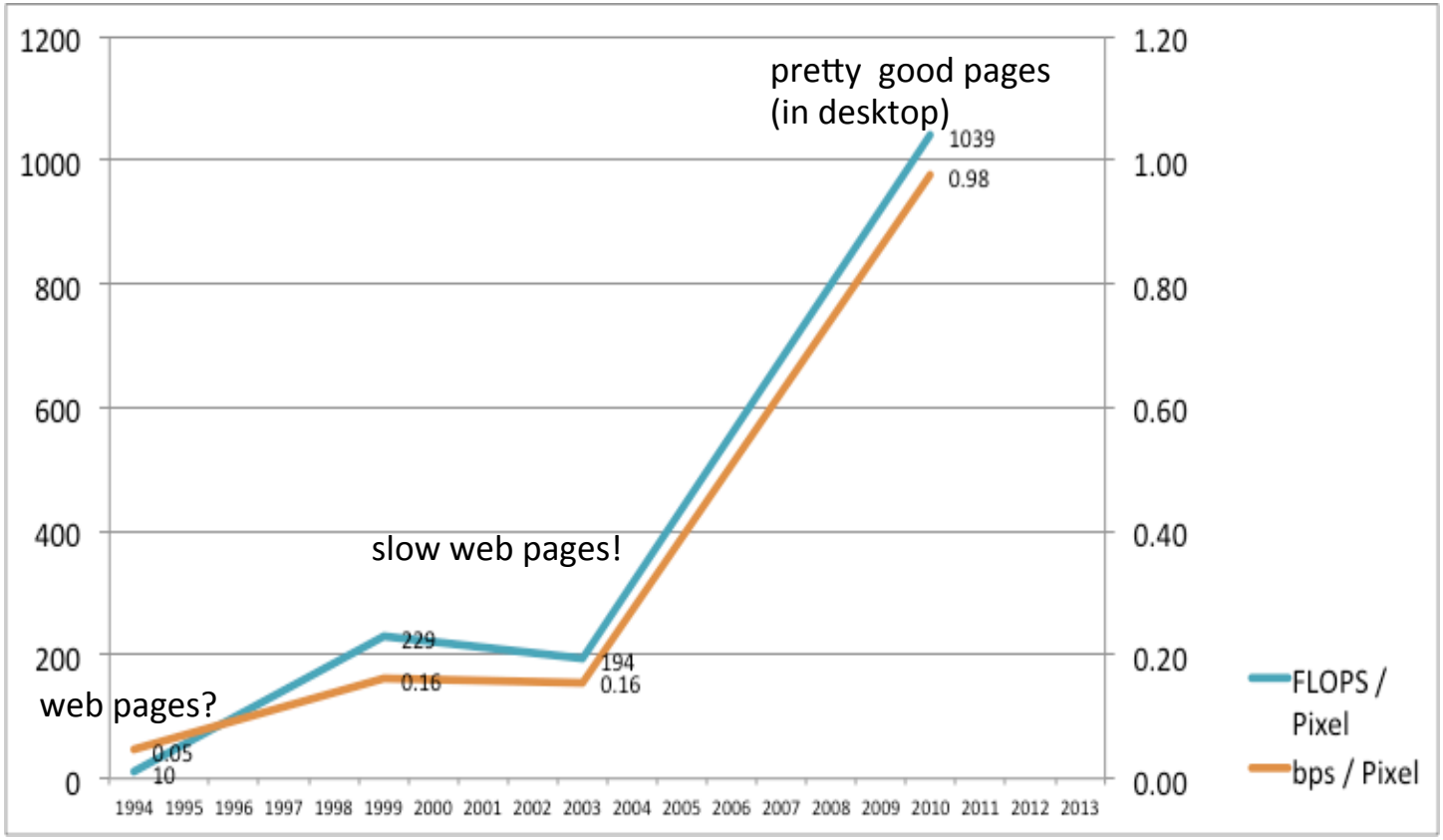
“GIVE ME SOMETHING THAT I CAN USE”

15 YEARS RETROSPECTIVE OF MY PERSONAL GEAR

1000 TIMES THE SPEED IN 15 YEARS

	CPU	Display	Connection	Network Speed / bps	CPU / MFLOPS	Pixels
1994	80486	640x480	modem	14,400	3	307,200
1999	AMD Athlon 500	1024x768	ISDN	128,000	180	786,432
2003	AMD Athlon Tbird	1600x1200	cable modem	300,000	373	1,920,000
2010	Intel Core i5	HDMI	cable modem	4,000,000	4,256	4,096,000

1000 TIMES THE CPU, 20 TIMES THE BANDWIDTH TO DRAW A PIXEL



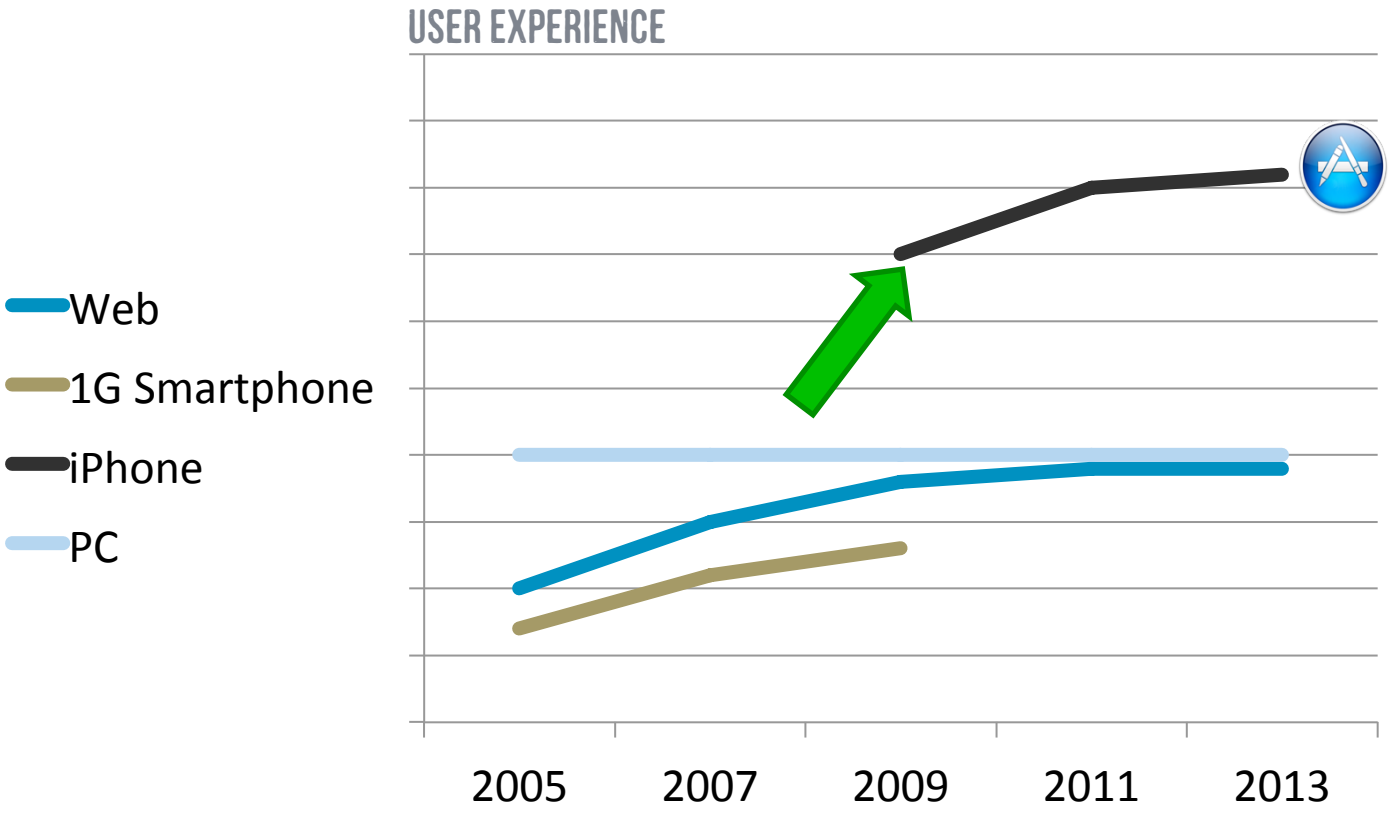
SO DO WE FEEL OUR APPS RUN 1000 TIMES FASTER?

OR EVEN 20 TIMES FASTER?

3 TIMES FASTER?

WE GOT USED TO CREATING SLUGGISH WEB SERVICES

AT THE SAME TIME, APPLE DELIVERED A SLICK APP EXPERIENCE

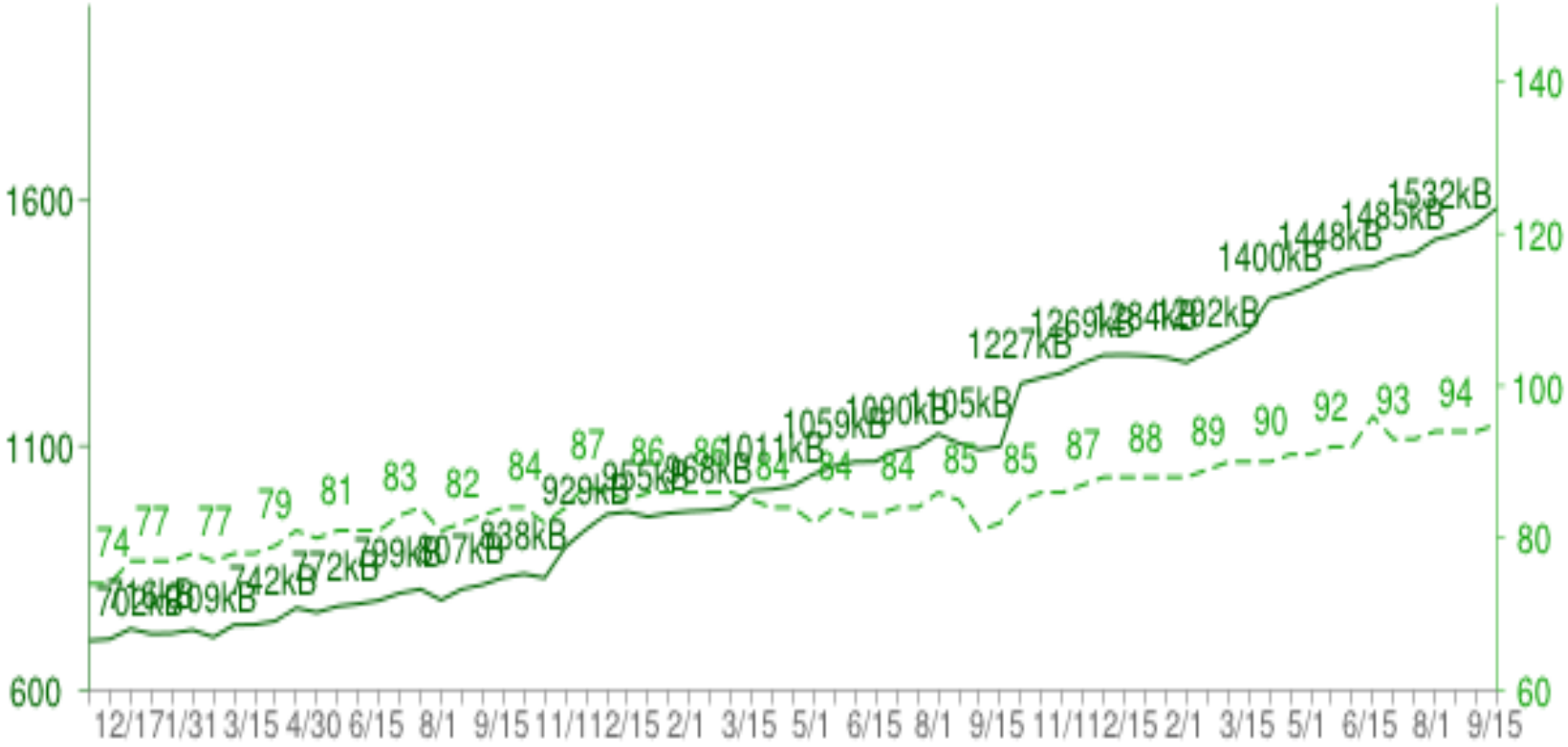


WE HAVE TRADED PERFORMANCE FOR EVERYTHING ELSE

- Distributing the data & computation far away
  - personal computing → web services → mobile & cloud
- Consuming increasingly rich data
  - Text → Images → Videos
- Moving to highly abstracted & interpreted languages
  - C → Java → JavaScript

WEB SITES STILL GET BIGGER

NOV 2010- SEPT 2013 STATS, HTTP ARCHIVE / STEVE SOUDERS

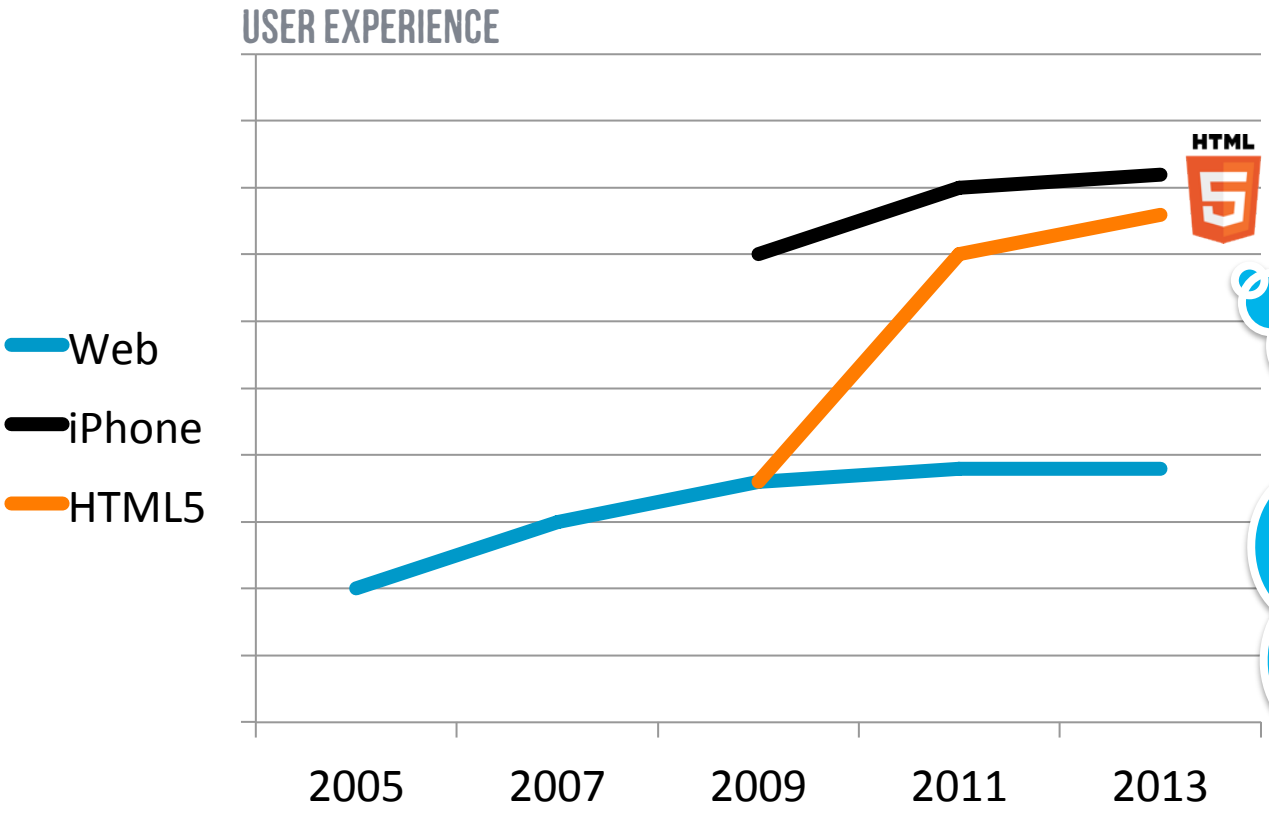


MOBILE DEVICES - 5 YEARS BACK IN CPU SPEED, 10 YEARS BACK IN CONNECTION BANDWIDTH

	Samsung Galaxy S4	My Machine (2003)	My Machine (2010)
CPU / MFLOPS	1500	373	4,256
Display / Pixels	2073600	1,920,000	4,096,000
Connection Speed / bps	384000*	300,000	4,000,000

* Assuming 3G/UMTS safe speed, due to low adoption of 4G networks

HTML5 & WEB APPS TO CHALLENGE NATIVE APPS IN SLICKNESS



Utilizing HW accelerated graphics, offline assets, advanced gestures and high perf JS engines

LET'S TARGET FOR MOBILE APP PERFORMANCE

- 1s app startup time (first page load time)
- 1s for any subsequent view/page (reasonable delay)
- 100ms UI response time (noticeable delay)
- 16ms paints (LCDs will refresh 50-60Hz, the rest is surplus)

WEB ENGINES ARE QUITE FAST!

LET'S USE THE SAME YARDSTICK WHEN MEASURING

HTML5 App

- 1000 DOM elements
- 1Mb of images and 100 network requests on page load

Native App

- 1000 widgets?
- 1Mb of images and 100 network requests on app startup?

How about these?

- 1s first page fold
- 1s application install?

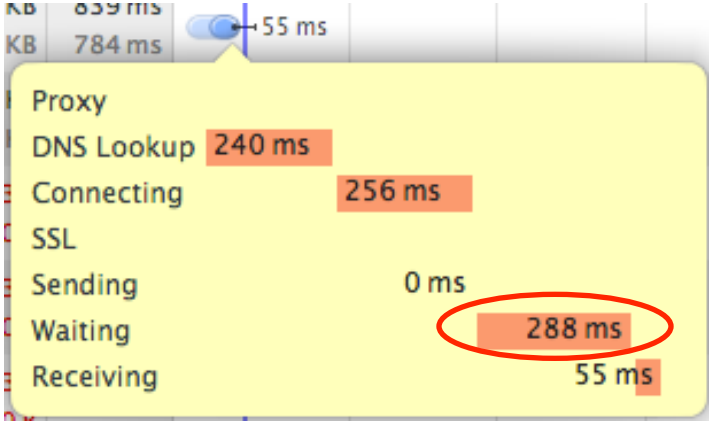
IF YOUR EMULATOR RUNS 20 TIMES FASTER THAN THE TARGET, WHAT CAN YOU EXPECT ABOUT PERFORMANCE?

LET'S ENGINEER THE WEB APPS THE SAME WAY APPS ARE ENGINEERED

ONE-SECOND PAGE LOADS

ONE SECOND BUDGET

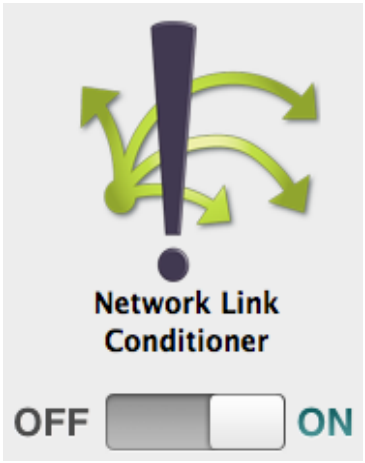
• Turn on radio		300ms
• DNS lookup	+	150ms
• SSL handshake	+	200ms
• Process & load HTML & other resources	+	200ms
• Parse → Layout → Paint	=	150ms

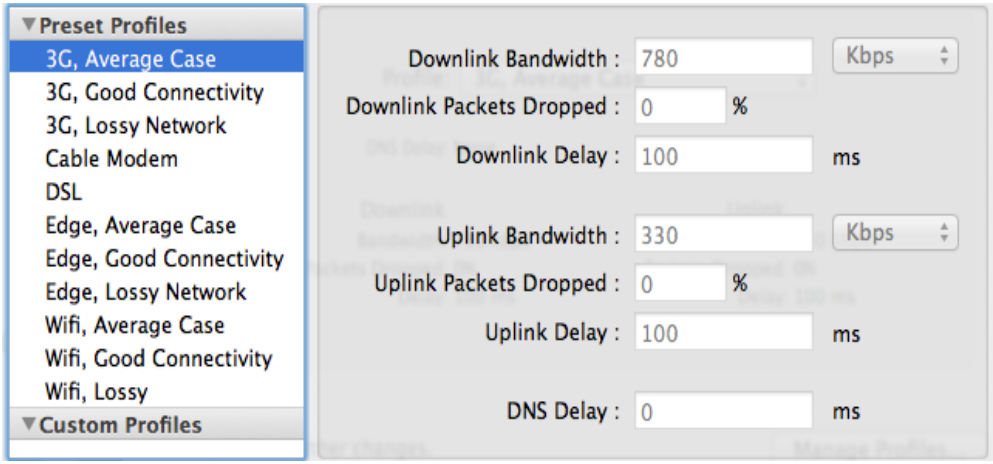


SIMULATE SLOW PAGE SPEEDS

- Developers tend to have ultra-fast computers & networks and forget the Egde & 3G network

- Tip: Network Link Conditioner XCode Plugin & ipfw rules





A SIMPLE SIMULATION SHELL SCRIPT

```
#!/bin/sh
#
# simulate_3g.sh - Simulate a sluggish 3G network with delays & packet loss
# Usage: simulate_3g.sh 8080 8081

# Make sure only root can run our script
if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root" 1>&2
   exit 1
fi

# Simulate http over 3G, 300kbit/s with 5% packet loss and 200ms delays
# into all ports given as a parameter
ipfw pipe 1 config delay 200 plr 0.1 bw 300kbit/s
for var in "$@"
do
    ipfw add 1 pipe 1 dst-port $var
done
```

COMBINE AND COMPRESS YOUR RESOURCES

- Combine and minify your CSS and JavaScript
- Combine small icons into a sprite sheet
- Squeeze the last bits from your images: It is much easier to remove 100k from your images than your JavaScript code!

- Tip: You should automate this, e.g. using Grunt

CACHING THE ASSETS

- CDNs for caching assets close to the user
- Varnish, Squid, Nginx etc. in front of your app server
- Having the CDNs and caches working requires good headers from your app server, too
- Tip: Let your Apache/Nginx reverse proxy care for your headers, they usually do it much better than you do

```
Cache-Control: public, max-age=0
Etag: "91580-1380653643000“
Last-Modified: Tue, 01 Oct 2013 18:54:03 GMT
```

SHARD FOR SEVERAL DOMAINS

- Modern browsers limit to 6 connections per host your non-scripted resources (e.g. CSS, images) from several hosts
- Severe problem because HTTP 1.1 requires the resources to be sent in the order they were requested

- Note: Remember the browser security rules, particularly CORS and Same-origin policy

IF YOU STILL GOT TIME TO OPTIMIZE PAGE LOADS

- **Optimize for the first page fold**: Critical CSS and JS embeded
- **Optimize for connection drops & offline**: Application Cache
- Batch your API calls
- Serve responsive images, prepare for the W3C adaptive images extension

```
<img alt="The Breakfast Combo” src="banner.jpeg”
   srcset="banner-HD.jpeg 2x,
   banner-phone.jpeg 100w,
   banner-phone-HD.jpeg 100w 2x”>
```

60 FRAMES SCROLL AND ANIMATION

OLD-SCHOOL OPTIMIZATIONS WON'T HELP YOU

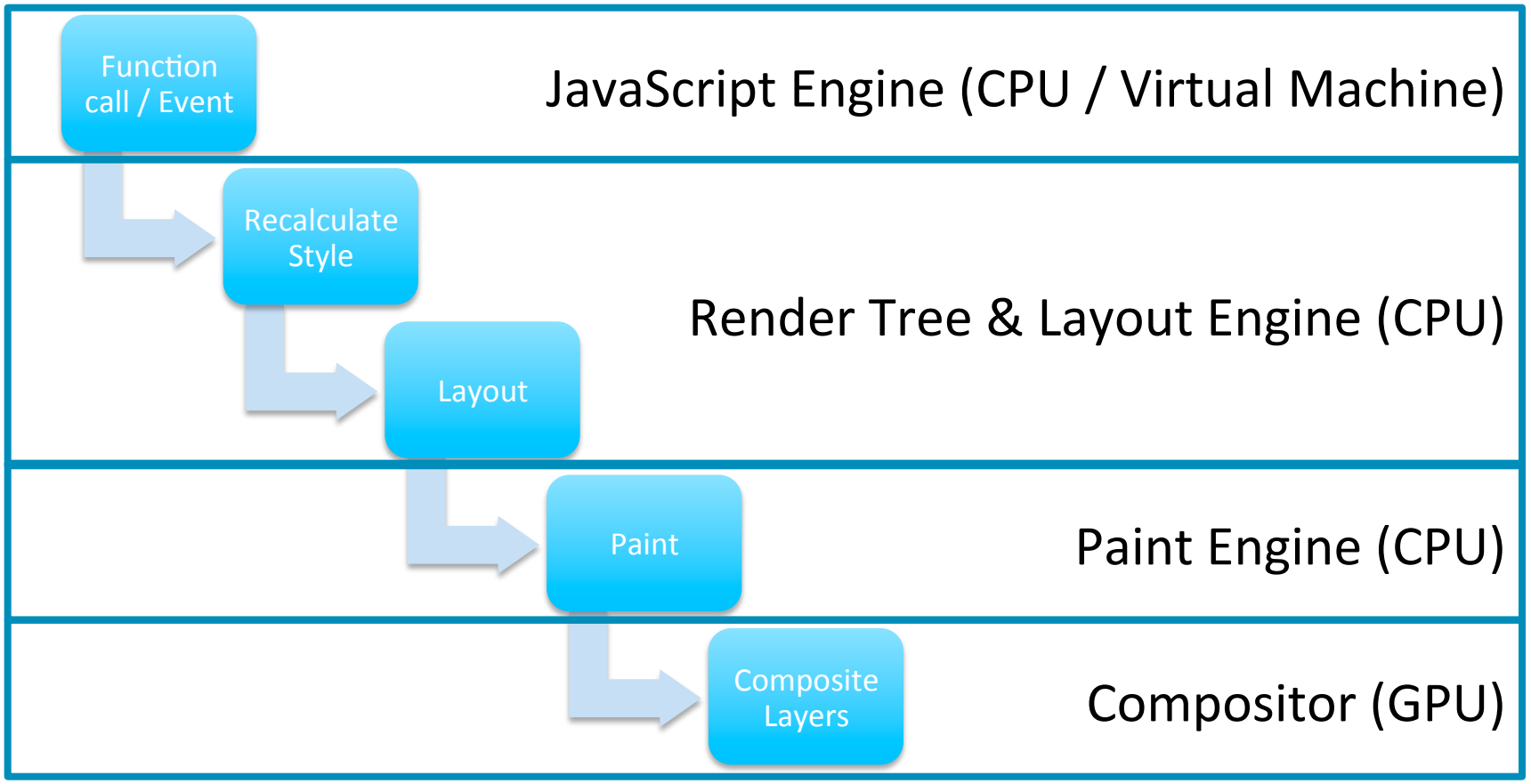
- JavaScript is typically not your problem
- CSS selector lengths typically have only a minor impact
- Browsers spend ~90% of its computation layouting and painting

What you want to track is

- The causes of relayout and repainting
- What their costs are

- Tip: Watch out adding/removing classes & hovering

"TYPICAL PAINT LOOP"



ALL CSS OPERATIONS AREN'T EQUAL

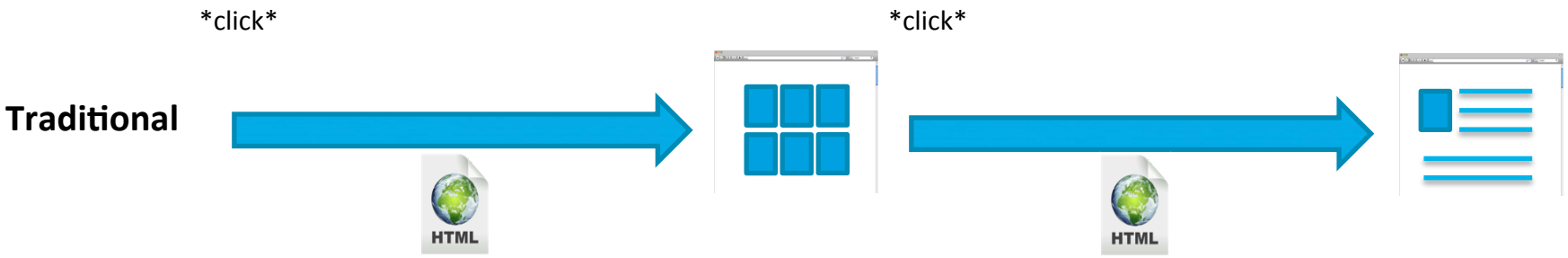
- Geometry changing ops: Layout, repaint, compositing
  - width, height etc…
- Paint-only ops: Repaint & compositing only
  - borders, outlines, box shadow, etc…
- Composition only (or less): Things that are 100% in GPU
  - CSS3 Transforms, Opacity

OPTIMIZING LAYOUTS & PAINTS

- Avoid DOM reads after geometry changing DOM operations
- Avoid a few expensive paint operations (shadows, border radius, flexbox etc...)
- Use `translateZ(0)` hack if you need a HW accelerated element
- Use CSS3 transforms for animating, they will not cause reflows

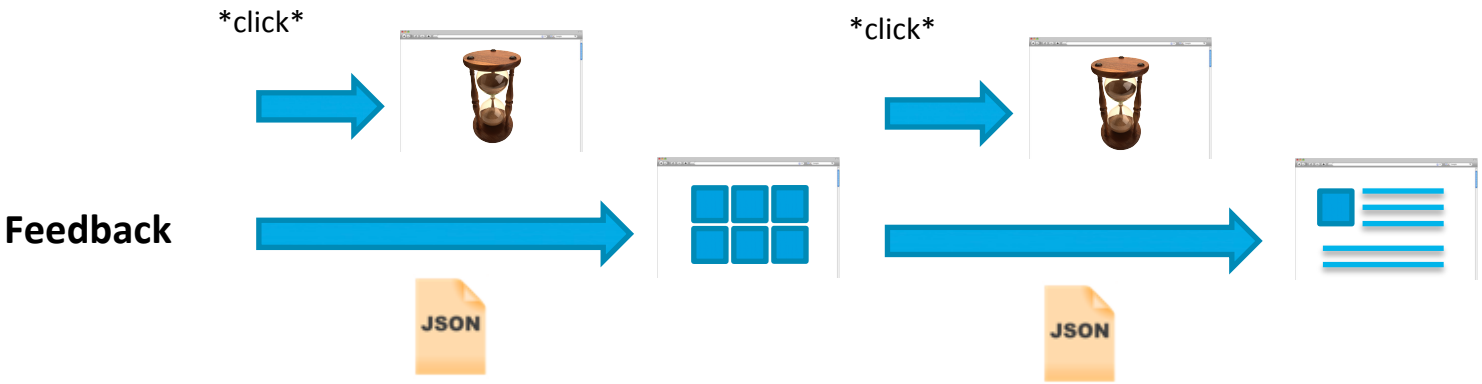
1/10 SECOND TO RESPOND,
ONE SECOND TO SHOW THE RESULTS

TRADITIONAL PAGES DON'T DO ANYTHING UNTIL YOU TELL



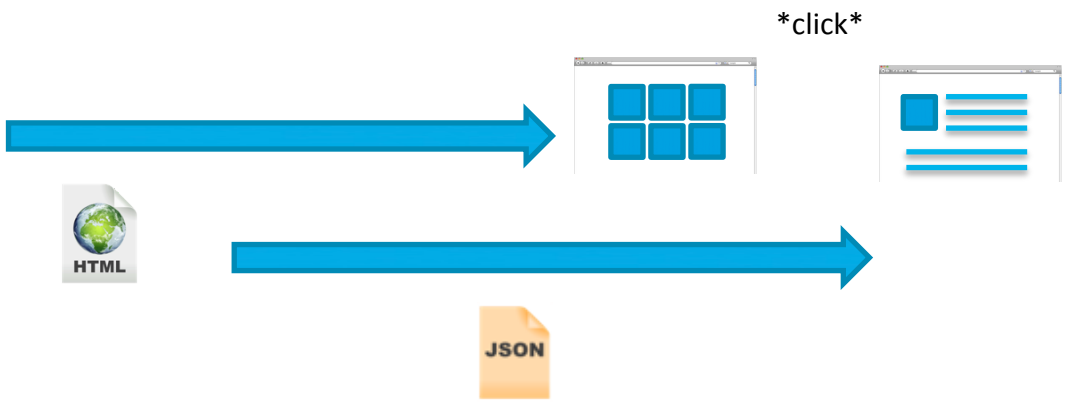
IMMEDIATE FEEDBACK BUYS YOU TIME

OVERALL PERFORMANCE MAY GET FASTER, TOO

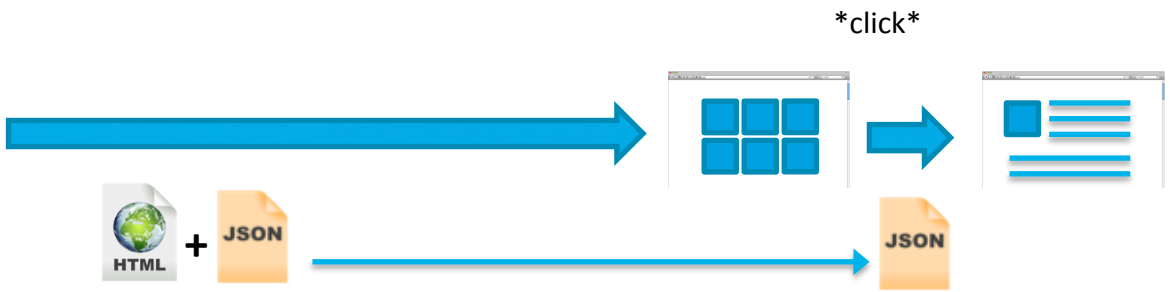


PRELOADING & BOOTSTRAPPING TO SHOW THE NEXT PAGE FASTER YOU MAY HURT YOURSELF BAD!

Preloading

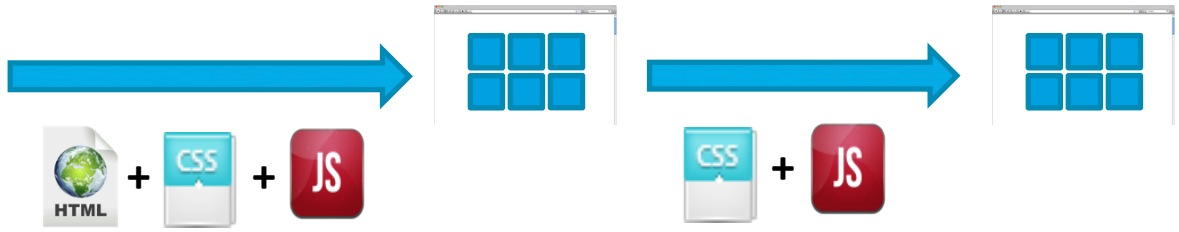


Bootstrapping



LAZY LOAD - OPTIMIZING FOR THE FIRST PAGE FOLD

Lazy Load



1st fold inlined

Rest of the page as separate resources

QUALITY ATTRIBUTES OF YOUR APP ARE SET BY YOUR ARCHITECTURE.

PERFORMANCE IS NO EXCEPTION. AT SOME POINT YOUR ARCHITECTURE WILL FIGHT AGAINST YOU.

RECAP: GETTING TO THE PERFORMANCE TARGETS

- Set the performance goals, prepare for tradeoffs
- Track the goals from the beginning
- Don’t guess; measure
- Simulate the target performance as part of your daily work
- Keep your code simple, don’t trade it for performance
- Perfect is the enemy of the good

THAT'S ALL!

ANY QUESTIONS?

THANK YOU !

LAURI SVAN

Software Architect, SC5 Online Ltd

https://github.com/laurisvan

https://twitter.com/laurisvan