MOBILE CHANGES EVERYTHING.
WHAT IMPACTS LOADING?
2017 Best Practices For Loading

Compress diligently! (GZip, Brotli)

Cache effectively (HTTP, Service Workers)

Minify & optimize *.*

Preresolve DNS for critical origins

Preload critical resources

Respect data plans

Stream HTML responses

Make fewer HTTP requests

Have a Font loading strategy

Send less JavaScript (code-splitting)

Lazy-load non-critical resources

Route-based chunking

Library sharding

PRPL pattern

Tree-shaking (Webpack, RollUp)

Serve modern browsers ES2015 (babel-preset-env)

Scope hoisting (Webpack)

Don’t ship DEV code to PROD
We want happy users.
Evolving

RESPONSE ANIMATION IDLE LOAD RAIL
User happiness metrics

- First Paint
- First Contentful Paint
- First Meaningful Paint
- Time To Interactive
“Networks, CPUs and disks all hate you. On the client, you pay for what you send in ways you can't easily see”

- Alex Russell, Chrome
JavaScript has a cost.

Fast = Fast at Download Parse Eval

On mobile devices
2017 JavaScript Parse Costs

~1MB JavaScript (uncompressed)
## JavaScript Parse Cost On Mobile - CNN

### Mobile cnn.com browser main thread time (Safari and Chrome)

<table>
<thead>
<tr>
<th>Device</th>
<th>Script</th>
<th>Layout</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone 8 (A11)</td>
<td>3967</td>
<td>2693</td>
<td>1148</td>
</tr>
<tr>
<td>iPhone 7+ (A10)</td>
<td>5669</td>
<td>2002</td>
<td>2582</td>
</tr>
<tr>
<td>iPhone 6s (A9)</td>
<td>6074</td>
<td>4483</td>
<td>1737</td>
</tr>
<tr>
<td>iPhone SE (A9)</td>
<td>5476</td>
<td>4464</td>
<td>1713</td>
</tr>
<tr>
<td>iPhone 6 (A8)</td>
<td>12038</td>
<td>5614</td>
<td>3069</td>
</tr>
<tr>
<td>iPhone 5c (A6)</td>
<td>16035</td>
<td>8403</td>
<td>5002</td>
</tr>
<tr>
<td>Samsung S7 (Exynos 8890)</td>
<td>14354</td>
<td>3250</td>
<td>1048</td>
</tr>
<tr>
<td>Moto G4 (Snapdragon 617)</td>
<td>13355</td>
<td>4107</td>
<td>1002</td>
</tr>
<tr>
<td>Thinkpad T430 (Core i5 3320M)</td>
<td>7179</td>
<td>4955</td>
<td>2851</td>
</tr>
<tr>
<td>Thinkpad Yoga (Core i7 6600U)</td>
<td>2891</td>
<td>1243</td>
<td>422</td>
</tr>
<tr>
<td>Desktop (Core i7-5930K)</td>
<td>2061</td>
<td>818265</td>
<td>2652</td>
</tr>
</tbody>
</table>

~9s difference to the A11

With thanks to Pat Meenan
PRPL Pattern

Push the minimal code for the initial route

Request
- index.html
- parsing
- entry
- parse/compile
- vendor
- parse/compile

Push/Preload critical scripts

Render route & get interactive

Navigate next route

Pre-cache using Service Workers
- scripts
- images
- styles

Cache remaining resources

Lazy-load async (split) routes

Navigate to another route

route-1
- parse/compile

route-2
Tools with a baseline that is fast by default provide the best chance of success.
Where do mobile sites spend their time loading?

With thanks to Camillo and Mathias @ V8
Caching

In most cases, when a web page needs a resource, Chrome starts by looking it up in the Memory cache. If the Memory cache doesn't have it, Chrome will then ask the network stack to handle the request. The network stack will eventually process the request and will start by looking for the resource in the HTTP cache. If the HTTP cache doesn’t have it, the network stack will then issue an actual network request.
# Chrome’s Cache Hit Rates

<table>
<thead>
<tr>
<th>Type</th>
<th>Hit rate for “Used as-is” (higher is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>End to end</td>
</tr>
<tr>
<td></td>
<td>Android</td>
</tr>
<tr>
<td>CSS</td>
<td>95%</td>
</tr>
<tr>
<td>JS</td>
<td>76.3%</td>
</tr>
<tr>
<td>Fonts</td>
<td>67.2%</td>
</tr>
<tr>
<td>Images</td>
<td>80.2%</td>
</tr>
</tbody>
</table>

* Chrome has 4+ caches. The above reflects the main two - the HTTP and memory caches.
EVERYONE IS RESPONSIBLE FOR performance.
Time to Interactive Budget

5s

1.6s  3.4s

DNS LOOKUP  TCP HANDSHAKE  HTTPS HANDSHAKE

At 400Kbps we can send 3.4 x 50KB = 170KB

Baseline is ~$200 Android phone
On a slow 3G network, emulated at:

400ms RTT, 400Kbps transfer
170KB gzipped JS
= ~0.8-1MB decompressed
= ~1s to parse/compile

File size
Budget

Critical-path
JS/CSS/HTML

Framework
Router,
State management,
Utilities

App
All bytes are not equal

170KB JS ≠ 170KB JPEG
“File-size isn't just about the download. Byte-for-byte, JavaScript is more expensive for the browser to process than the equivalently sized image or Web Font.”

- Tom Dale, Glimmer & Ember
Operations that must happen on the main-thread

JavaScript

Parse
Compile

JavaScript Execution

Construction of DOM
Layout
Processing input (including scrolling w/ active touch listeners)
Evaluating the performance of Web Frameworks

NETWORK TRANSFER

PARSE/COMPILE

RUNTIME COST

https://twitter.com/kristoferbaxter/status/908144931125858304
GOOD OPTIONS FOR MOBILE WEB

*WITH CODE-SPLITTING AND A PERF BUDGET*

AND OTHERS, LIKE STENCILJS.
Recipe for building good web sites

Dev on an Average Phone
- Thermally throttled.
- Near-0 L2/L3 cache
- Feel CPU & GPU limits.

Keep JavaScript Parse/Eval Low
- HUGE contribs to slowdown.
- Heavily impacted by size.

Interactive < 5s
Weight < 170KB

Have a Performance Budget
- Make perf accountable.
- Automate auditing.
Performance Budget Tools

CALIBRE  SPEEDCURVE  BUNDLESIZE

Budgets

Total Image transferred must be less than 204.88 KB

Moto G4

JS Size for Polaris has gone **under** your set budget of **200KB**. 1 other test also crossed this budget threshold.
REAL-WORLD WEB PERF BUDGETS

![Chart showing time (s) vs data (KB) with预算: 5 seconds, 131 KB, and 170 KB highlighted.]
HEALTH OF THE WEB
Web Performance Tooling

Synthetic lab conditions

Chrome DevTools

Lighthouse

Real-world

WebPageTest

Synthetic lab conditions

Real-world

WebPageTest

Synthetic lab conditions

Real-world

WebPageTest

Synthetic lab conditions

Real-world

WebPageTest
EXPLORE
The health of the web as a whole with HTTP Archive
RESPONSE BODIES
LIGHTHOUSE REPORTS
BLINK FEATURE COUNTERS
NEW PERFORMANCE METRICS

..AND IT’S ALL QUERYABLE!
STATE OF JAVASCRIPT ON MOBILE

SITES ARE SENDING USERS...

P90: ~1MB
P75: 0.6MB

OF JS SPENDING ~4s ON PARSE/COMPILE

Using Dev Tools mobile emulation, Moto G4 calibrated CPU, Cable (5/1mbps, 28ms)
Minify _everything_
Babelified ES5 w/Uglify
ES2015+ with Babili
css-loader + minimize:true

Code-splitting
Dynamic import()
Route-based chunking

Tree-shaking
Webpack 2+ with Uglify
RollUp
DCE w/ Closure Compiler

Optimize “Vendor” libs
NODE_ENV=production
CommonsChunk + HashedModuleIdsPlugin()

Transpile less code
babel-preset-env + modules:false
Browserlist
useBuiltIns: true

Scope Hoisting:
Webpack 3
RollUp

Strip unused Lodash modules
lodash-webpack-plugin
babel-plugin-lodash

Fewer Moment.js locales
ContextReplacementPlugin()
Is all of that ~1MB used upfront?
JS CODE COVERAGE OF TOP 50 SITES

SITES MAY USE ONLY 40% OF THE JAVASCRIPT THEY LOAD UPFRONT.

With thanks to fmeawad@chromium.org
Removing unused code can reduce network transmission times, CPU-intensive code parsing, and memory overhead.
SITES ARE SENDING USERS...

**P90**: 5.4MB

**P75**: 2.9MB

P90 3.8MB (70%) of this is images

P90 1MB (18%) of this is JS
70% OF THIS IS IMAGES. OPTIMIZE THEM.

Choose the right format
Size appropriately
Adapt intelligently
Compress carefully
Prioritize critical images
Lazy-load the rest
Take care with tools

https://images.guide
WEB SPEED METRICS ON MOBILE

SITES ARE INTERACTIVE IN...

P90: 35s
P75: 22s
P90: 11s before First Meaningful Paint

http://beta.httparchive.org
Queryable RUM for the web?

Ilya Grigorik @igrigorik  Bryan McQuade @bryancmcquade
Chrome User Experience Report

- Real world performance as experienced by Chrome users
  - User experience metrics
  - Origin-level resolution

- Initial release is...
  - Focusing on metrics that capture loading experience
  - Provides a sample of 10K origins

Available as a public dataset on Google BigQuery.
<table>
<thead>
<tr>
<th>origin</th>
<th>form factor</th>
<th>ECT</th>
<th>bin.start (ms)</th>
<th>bin.end (ms)</th>
<th>bin. density</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone</td>
<td>4G</td>
<td>0</td>
<td>200</td>
<td>0.02</td>
<td>...</td>
</tr>
<tr>
<td><a href="https://example.com">https://example.com</a></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
<td>...</td>
</tr>
<tr>
<td>desktop</td>
<td>3G</td>
<td>0</td>
<td>200</td>
<td>0.03</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Keyed by origin
<table>
<thead>
<tr>
<th>origin</th>
<th>form factor</th>
<th>ECT</th>
<th>first_contentful_paint</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://example.com">https://example.com</a></td>
<td>phone</td>
<td>4G</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>desktop</td>
<td></td>
<td>3G</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

phone, desktop, tablet
<table>
<thead>
<tr>
<th>origin</th>
<th>form factor</th>
<th>ECT</th>
<th>first_contentful_paint</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://example.com">https://example.com</a></td>
<td>phone</td>
<td>4G</td>
<td>bin.start (ms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>desktop</td>
<td>desktop</td>
<td>3G</td>
<td>bin.start (ms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

effectiveConnectionType
NetInfo API
<table>
<thead>
<tr>
<th>origin</th>
<th>form factor</th>
<th>ECT</th>
<th>bin.start (ms)</th>
<th>bin.end (ms)</th>
<th>bin. density</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://example.com">https://example.com</a></td>
<td>phone</td>
<td>4G</td>
<td>0</td>
<td>200</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>desktop</td>
<td>3G</td>
<td>0</td>
<td>200</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Paint API: First Paint, First Contentful Paint + domContentLoaded, onload
<table>
<thead>
<tr>
<th>origin</th>
<th>form factor</th>
<th>ECT</th>
<th>bin.start (ms)</th>
<th>bin.end (ms)</th>
<th>bin. density</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone</td>
<td>4G</td>
<td></td>
<td>0</td>
<td>200</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>desktop</td>
<td>3G</td>
<td></td>
<td>0</td>
<td>200</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Histogram: [start, end, density]

....
### New Query

```sql
SELECT *
FROM `chrome-ux-report.chrome_ux_report.281710`
WHERE origin = 'https://www.google.com';
```

#### Results

<table>
<thead>
<tr>
<th>Row</th>
<th>origin</th>
<th>effective_connection_type.name</th>
<th>form_factor.name</th>
<th>first_paint.histogram.bin.start</th>
<th>first_paint.histogram.bin.end</th>
<th>first_paint.histogram.bin.density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="https://www.google.com">https://www.google.com</a></td>
<td>4G</td>
<td>phone</td>
<td>0</td>
<td>200</td>
<td>0.0198</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>400</td>
<td>0.0648</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>400</td>
<td>600</td>
<td>0.0577</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>600</td>
<td>800</td>
<td>0.0237</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>800</td>
<td>1000</td>
<td>0.0135</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td>1200</td>
<td>0.009</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1200</td>
<td>1400</td>
<td>0.0055</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1400</td>
<td>1600</td>
<td>0.0038</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1600</td>
<td>1800</td>
<td>0.0027</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1800</td>
<td>2000</td>
<td>0.0021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2000</td>
<td>2200</td>
<td>0.0013</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2200</td>
<td>2400</td>
<td>9.0E-4</td>
</tr>
</tbody>
</table>
SELECT 
  origin, 
  ROUND(100* 
    SUM(( 
      SELECT SUM(bin.density) 
        FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 100)) AS fast_percent 
  FROM 
  'chrome-ux-report.chrome_ux_report.201710' 
  WHERE origin = 'https://www.google.com' 
GROUP BY origin;
### Percent of fast loads on https://www.google.com

```sql
SELECT origin, ROUND(100 * SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000)) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin = 'https://www.google.com'
GROUP BY origin;
```

<table>
<thead>
<tr>
<th>Row</th>
<th>origin</th>
<th>fast_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="https://www.google.com">https://www.google.com</a></td>
<td>81.0</td>
</tr>
</tbody>
</table>
Percent of fast loads on https://www.google.com

```sql
SELECT origin, 
    ROUND(100* 
        SUM(( 
            SELECT SUM(bin.density) 
            FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000)) AS fast_percent 
    ) 
FROM chrome-ux-report.chrome_ux_report.201710 
WHERE origin = 'https://www.google.com' 
GROUP BY origin;
```
SELECT
  origin,
  ROUND(100 *
    SUM(
      SELECT SUM(first_contentful_paint) AS fast_percent
      FROM UNNEST histogram.bin) bin WHERE bin.end <= 1000
    ) AS fast_percent
FROM
  'chrome-ux-report.chrome_ux_report.201710'
WHERE origin = 'https://www.google.com'
GROUP BY origin;
```
SELECT origin, 
    ROUND(100* 
        SUM(
            SELECT SUM(bin.density) 
                FROM UNNEST(first_contentful_paint.histogram.bin) bin 
                WHERE bin.end <= 1000) 
        AS fast_percent 
    FROM `chrome-ux-report.chrome_ux_report.201710` 
    WHERE origin = 'https://www.google.com' 
GROUP BY origin;
```
SELECT
  origin,
  ROUND(100 * SUM(
      SELECT SUM(bin.density)
      FROM UNNEST(first_contentful_paint.histogram.bin) bin
      WHERE bin.end <= 1000
    ) AS fast_percent
  )
FROM
  `chrome-ux-report.chrome_ux_report.201710`
WHERE origin = 'https://www.google.com'
GROUP BY origin;
SELECT origin, ROUND(100 * SUM((
  SELECT SUM(bin.density)
  FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000)) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin = 'https://news.google.com'
GROUP BY origin;
SELECT origin,
    ROUND(100* SUM(
        SELECT SUM(bin.density)
        FROM UNNEST(first_contentful_paint.histogram.bin) bin
        WHERE bin.end <= 1000)
    ) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin = 'https://news.google.com'
GROUP BY origin;
SELECT origin, 
    form_factor.name as form_factor, 
    ROUND(100* 
      SUM(( 
        SELECT SUM(bin.density) 
        FROM UNNEST(first_contentful_paint.histogram.bin) bin 
        WHERE bin.end <= 1000)) / 
      SUM(( 
        SELECT SUM(bin.density) 
        FROM UNNEST(first_contentful_paint.histogram.bin) bin)) AS fast_percent 
FROM 'chrome-ux-report.chrome_ux_report.201710' 
WHERE origin = 'https://news.google.com' 
GROUP BY origin, form_factor;
SELECT
origin,
form_factor.name as form_factor,
ROUND(100 *
  SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin
    WHERE bin.end <= 1000)
  / SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin)
) AS fast_percent
FROM `chrome-ux-report.chrome_ux_report.201710`
WHERE origin = 'https://news.google.com'
GROUP BY origin, form_factor;
SELECT
    origin,
    form_factor.name as form_factor,
    ROUND(100 *
        SUM(
            SELECT SUM(bin.density)
            FROM UNNEST(first_contentful_paint.histogram.bin) bin
            WHERE bin.end <= 1000)
        / SUM(
            SELECT SUM(bin.density)
            FROM UNNEST(first_contentful_paint.histogram.bin) bin)) AS fast_percent
FROM chrome-ux-report.chrome_ux_report.201710
WHERE origin = 'https://news.google.com'
GROUP BY origin, form_factor;
```
SELECT
  origin,
  form_factor.name as form_factor,
  ROUND(100 * SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin
    WHERE bin.end <= 1000)) / SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin
  ) AS fast_percent
FROM `chrome-ux-report.chrome_ux_report.201710`
WHERE origin = 'https://news.google.com'
GROUP BY origin, form_factor;
```
SELECT origin, form_factor.name as form_factor, 
  ROUND(100* SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000)) / 
  SUM(
    SELECT SUM(bin.density)
    FROM UNNEST(first_contentful_paint.histogram.bin))) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin = 'https://news.google.com'
GROUP BY origin, form_factor;
SELECT origin,
    form_factor.name as form_factor,
    ROUND(100 * SUM(
        SELECT SUM(bin.density)
        FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000) /
        SUM(
            SELECT SUM(bin.density)
            FROM UNNEST(first_contentful_paint.histogram.bin) bin))) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin like 'https://%.google.com'
GROUP BY origin, form_factor
ORDER by fast_percent DESC;
### Percent of fast loads on Google origins

```sql
SELECT
    origin,
    form_factor.name as form_factor,
    ROUND(100*SUM(
        SELECT SUM(bin.density)
        FROM UNNEST(first_contentful_paint.histogram.bin) bin WHERE bin.end <= 1000)
    / SUM(
        SELECT SUM(bin.density)
        FROM UNNEST(first_contentful_paint.histogram.bin))
) AS fast_percent
FROM 'chrome-ux-report.chrome_ux_report.201710'
WHERE origin like 'https://%2.google.com'
GROUP BY origin, form_factor
ORDER by fast_percent DESC;
```

<table>
<thead>
<tr>
<th>origin</th>
<th>form_factor</th>
<th>fast_percent</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.google.com">https://www.google.com</a></td>
<td>phone</td>
<td>82.0</td>
</tr>
<tr>
<td><a href="https://scholar.google.com">https://scholar.google.com</a></td>
<td>desktop</td>
<td>82.0</td>
</tr>
<tr>
<td><a href="https://www.google.com">https://www.google.com</a></td>
<td>desktop</td>
<td>81.0</td>
</tr>
<tr>
<td><a href="https://scholar.google.com">https://scholar.google.com</a></td>
<td>phone</td>
<td>68.0</td>
</tr>
<tr>
<td><a href="https://translate.google.com">https://translate.google.com</a></td>
<td>desktop</td>
<td>62.0</td>
</tr>
<tr>
<td><a href="https://support.google.com">https://support.google.com</a></td>
<td>desktop</td>
<td>57.0</td>
</tr>
</tbody>
</table>
Analysis best practices

When comparing origin A vs. origin B consider...

- Population differences
- Population size differences
- Chrome population differences

e.g. *small site visited by “fast” users*  

*vs.*  

*large service visited by wide distribution of users*
Getting started

Looking for your feedback, suggestions & ideas!

bit.ly/introducing-crux
GIVING DEVELOPERS MORE control.
If my Web Fonts can’t load quickly, don’t load them at all.

```css
@font-face {
  font-family: 'Roboto';
  font-display: optional;
  src: url(Roboto.woff) format('woff'),
       url(Roboto.eot) format('eot');
  font-weight: 400;
  font-style: normal;
}
```
I want to adapt serving based on estimated network quality

```javascript
// Network type that browser uses
navigator.connection.type
> 'wifi'

// New: Effective connection type
// using rtt and downlink values
navigator.connection.effectiveType
> '2G'
```

For more on `navigator.connection.*`
See ‘Building a modern media experience’
<link rel="preload" as="script" href="bundle.js">
I have critical resources I want to load earlier than discovery.

```html
<link rel="preload" href="movies.json" as="fetch" crossorigin="use-credentials" />
<script>
(async () => {
  try {
    const response = await fetch(new Request("movies.json", {credentials: "include"}));
    const data = await response.json();
    console.log(data);
  } catch (exception) {
    console.log("Booo");
  }
})();
</script>
```
addEventListener('activate', event => {
  event.waitUntil(async function() {
    // Feature-detect
    if (self.registration.navigationPreload) {
      // Enable navigation preloads!
      await self.registration.navigationPreload.enable();
    }
  }());
});

I want to start network requests while the Service Worker is still booting up.

Saves 1 RTT

Early numbers suggest a 20% improvement to page load time at PC95.

Chrome 59
Many sites optimize for the Lowest Common Denominator

Most users end up being deployed ES2015 polyfills
Deploying ES2015+ JavaScript in 2017

babel-preset-env + <script type=module>

<!-- Browsers with ES module support load this file. -->
<script type="module" src="main.js"></script>

<!-- Older browsers load this file (and module-supporting -->
<!-- browsers know *not* to load this file). -->
<script nomodule src="main-legacy.js"></script>

<table>
<thead>
<tr>
<th>Version</th>
<th>Size (minified)</th>
<th>Size (minified + gzipped)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES2015+ (main.js)</td>
<td>80K</td>
<td>21K</td>
</tr>
<tr>
<td>ES5 (main-legacy.js)</td>
<td>175K</td>
<td>43K</td>
</tr>
</tbody>
</table>
Deploying ES2015+ JavaScript in 2017

main-legacy.js

```javascript
module: {
  rules: [{
    test: '/\.js$/',
    use: {
      loader: 'babel-loader',
      options: {
        presets: [
          ['env', {
            modules: false,
            useBuiltIns: true,
            targets: {
              browsers: [
                '> 1%',
                'last 2 versions',
                'Firefox ESR',
              ],
            }
          },
        ],
      }
    }
  }]
}
```

main.js

```javascript
rules: [{
  test: '/\.js$/',
  use: {
    loader: 'babel-loader',
    options: {
      presets: [
        ['env', {
          modules: false,
          useBuiltIns: true,
          targets: {
            browsers: [
              'Chrome >= 60',
              'Safari >= 10.1',
              'iOS >= 10.3',
              'Firefox >= 54',
              'Edge >= 15',
            ],
          }
        },
      ]
    }
  }
}
```
FUTURE? BETTER PERF.

Modules.

Service Workers.

Navigation Architecture.

TODAY, STILL NEED TO BUNDLE FOR PRODUCTION

OFF-MAIN THREAD FETCH
SCRIPT STREAMING

PlzNavigate
PROGRESSIVE WEB APPS ARE THE NEW normal.

but...we have some new ones to share!
Old Mobile Site - 1st load

First Paint: 4.2s
First Meaningful Paint: 6.2s
Time To Interactive: 23s
New Mobile Site - 1st load

First Paint: 1.8s
First Meaningful Paint: 5.1s
Time To Interactive: 5.6s

JS Bundles: 620KB ➔ 150KB
CSS Bundles: 150KB ➔ 6KB inline
P90 for Pin pages: 20s ➔ 6.5s
New Mobile Site - Repeat Loads

First Paint: 0.6s
First Meaningful Paint: 3.5s
Time To Interactive: 3.9s
Size: Comparing the PWA to the native apps

- Android: 9.6MB
- iOS: 56MB
- PWA: 150KB home feed load
Comparing old mobile web to new mobile web

<table>
<thead>
<tr>
<th>Metric</th>
<th>New Mobile Web</th>
<th>Old Mobile Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Spent &gt; 5 minutes</td>
<td>+40%</td>
<td></td>
</tr>
<tr>
<td>User-generated Ad ($)</td>
<td>+44%</td>
<td></td>
</tr>
<tr>
<td>Ad Clickthroughs</td>
<td>+50%</td>
<td></td>
</tr>
<tr>
<td>Core Engagements</td>
<td>+60%</td>
<td></td>
</tr>
</tbody>
</table>

Comparing across web/native

<table>
<thead>
<tr>
<th>Metric</th>
<th>Web/native</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Spent &gt; 5 minutes</td>
<td>+5%</td>
<td>+2%</td>
</tr>
<tr>
<td>User-generated Ad ($)</td>
<td>+2%</td>
<td></td>
</tr>
<tr>
<td>Ad Clickthroughs</td>
<td>+0%</td>
<td>+2-3%</td>
</tr>
<tr>
<td>Core Engagements</td>
<td>+2-3%</td>
<td>+2-3%</td>
</tr>
</tbody>
</table>
JavaScript Bundle Splitting Strategy

How did Pinterest handle code-splitting?

- **Vendor**: React, Redux, React Router etc (73KB)
- **Entry**: Main shell, Core logic, Redux store (59KB)
- **Async**: Async route chunks (13-18KB)
const chunkPlugins = [
    new webpack.optimize.CommonsChunkPlugin({
        name: 'vendor-mweb',
        minChunks: Infinity,
        chunks: ['entryChunk-mobile']
    }),
    new webpack.optimize.CommonsChunkPlugin({
        name: 'entryChunk-webpack',
        minChunks: Infinity,
        chunks: ['vendor-mweb']
    }),
];

const bundles = {
    'vendor-mweb': ['app/mobile/polyfills.js', 'intl', 'normalizr', 'react-dom', 'react-redux', 'react-router-dom', 'react', 'redux'],
    'entryChunk-webpack': 'app/mobile/runtime.js',
    'entryChunk-mobile': 'app/mobile/index.js'
};
// Create a loader
const Closeup = () => import(/* webpackChunkName: "CloseupPage" */
'app/mobile/routes/CloseupPage');

// Register it to the route
route('/pin/:pinId', routes.Closeup, { name: 'Closeup' }),

// Render a react-router-v4 Route with the route bundle loader
<Route exact key="matched-route" path={path} render={matchProps =>
  <PageRoute
    bundleLoader={loader}
    routeName={name}
    {...matchProps}
    {...props}
  />}
/>
// Async load the route bundle
class PageRoute extends PureComponent {
  render() {
    const { bundleLoader, ...props } = this.props;
    return <Loader loader={{bundleLoader}} {...props} />
  }
}

// Load it and render
class Loader extends PureComponent {
  componentDidMount() {
    this.props.loader().then(module => {
      this.setState({ LoadedComponent: module.default });
    });
  }
}
Webpack Bundle Analyzer: Before splitting out common async route code
Webpack Bundle Analysis

Tackling dupes across lazily loaded routes

- Moved common code into the Entry chunk
- 20% increase in size of Entry (59KB -> 71KB)
- 60-90% decrease in size of async route chunks
- e.g: Homefeed (13.9KB -> 1KB), Closeup (18KB -> 7KB)
Webpack Bundle Analyzer: After moving out common code from async chunks into entryChunk

- 60-90% decrease in size of async route chunks (e.g., 13.9KB ➔ 1KB)
- 20% increase in size of entry (59KB ➔ 71KB)
Service Workers

*Caching runtime & static assets offline*

**Start**
- Runtime caching async JS chunks (for V8 bytecode cache)
- Precaching vendor & entry chunks
- Precaching most used routes (e.g., Pins)
- Generating a SW for each locale bundle

**Today**
- Cache all JS/CSS cache-first
- Cache the Application Shell
- Precache bundle loaded by the shell
  - Webpack runtime, vendor, entry
  - Named chunks to cache async routes
/* global $VERSION, $Cache, importScripts, WorkboxSW */
importScripts('https://unpkg.com/workbox-sw@1.1.0/build/importScripts/workbox-sw.prod.v1.1.0.js');

// Add app shell to the webpack-generated precache list
$Cache.precache.push({ url: 'sw-shell.html', revision: $VERSION });

// Register precache list with Workbox
const workbox = new WorkboxSW({ handleFetch: true, skipWaiting: true, clientClaim: true });
workbox.precache($Cache.precache);

// Runtime cache all js
workbox.router.registerRoute(/webapp/js/.*\.js/, workbox.strategies.cacheFirst());

// Prefer app-shell for full-page loads
workbox.router.registerNavigationRoute('sw-shell.html', {
  blacklist: [
    // bunch of non-app routes
  ],
});
Future

- Web Push notifications
- Fixing slow API responses (home-feed takes 1s on Fast 3G)
- Optimize server latency and response sizes
- Adding `<link rel=preload as=script>` for preloading bundles
Size: Comparing the PWA to the native apps

- Android: 30MB
- PWA: 2.8MB
Before

First meaningful paint: 7,770 ms
First meaningful paint measures when the primary content of a page is visible. Learn more.

First Interactive (beta): 7,770 ms
The first point at which necessary scripts of the page have loaded and the CPU is idle enough to handle most user input.

Consistently Interactive (beta): 7,770 ms
The point at which most network resources have finished loading and the CPU is idle for a prolonged period.

After

- First meaningful paint 5,850 ms
- First Interactive (beta) 6,240 ms
- Consistently Interactive (beta) 6,240 ms
Performance Budgets

Budgets Tinder tries not to exceed

- **Vendor**: 155KB
- **Async**: 55KB
- **Other**: 35KB
- **CSS**: 20KB
import A from '../A';
import B from '../B';

const route = [
  {
    route: '/'
    regions: {
        side: A,
        main: B
    }
  }
];
import Loadable from 'react-loadable';

class A {
  static loadable = Loadable({
    loader: () => import('./A') /* webpackChunkName: 'pc-r-A' */,
    loading: () => null
  });
}

class B {
  static loadable = Loadable({
    loader: () => import('./B') /* webpackChunkName: 'pc-r-B' */,
    loading: () => null
  });
}

const route = [
  {
    route: '/',
    regions: {
      side: A.loadable,
      main: B
    },
    preload: [ /* next page chunk to preload*/ ]
  }
]
JavaScript Route-based code-splitting

Preloading more page chunks with React Loadable

```javascript
const LoadableComponent = Loadable({ ... });

LoadableComponent.preload();
```
JavaScript Route-based code-splitting

**Before**
- Main bundle size: 166kb
- DOMContentLoaded: 5.46s
- Load: 11.91s

**After**
- Main bundle size: 101kb
- DOMContentLoaded: 4.69s
- Load: 4.69s
<link rel=preload as=script> before

Reduce first paint by 500ms, load time by 1 second

<link rel=preload as=script> after
new BundleAnalyzerPlugin({
  analyzerMode: 'server',
  analyzerPort: 8888,
  reportFilename: 'report.html',
  openAnalyzer: true,
  generateStatsFile: false,
  statsFilename: 'stats.json',
  statsOptions: null
})

- core-js + babel-preset-env to drop unused polyfills
- Use lodash-webpack-plugin to reduce bundle size
- Replaced localForage with IndexedDB
- Split non-critical components not used for First Paint
- Removed critical CSS from bundle (SSRs already)
new LodashModuleReplacementPlugin({
caching: true,
collections: true,
paths: true,
shorthands: true
}),

Lodash Savings

The following features are removed by default (biggest savings first):

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shorthands</td>
<td>Iteratee shorthands for underscore.property, underscore.matches, &amp; underscore.matchesProperty.</td>
</tr>
<tr>
<td>cloning</td>
<td>Support “clone” methods &amp; cloning source objects.</td>
</tr>
<tr>
<td>currying</td>
<td>Support “curry” methods.</td>
</tr>
<tr>
<td>caching</td>
<td>Caches for methods like underscore.cloneDeep, underscore.isEqual, &amp; underscore.uniq.</td>
</tr>
<tr>
<td>collections</td>
<td>Support objects in “Collection” methods.</td>
</tr>
<tr>
<td>exotics</td>
<td>Support objects like buffers, maps, sets, symbols, typed arrays, etc.</td>
</tr>
<tr>
<td>guards</td>
<td>Guards for host objects, sparse arrays, &amp; other edge cases.</td>
</tr>
<tr>
<td>metadata</td>
<td>Metadata to reduce wrapping of bound, curried, &amp; partially applied functions. (requires currying)</td>
</tr>
<tr>
<td>deburring</td>
<td>Support deburring letters.</td>
</tr>
<tr>
<td>unicode</td>
<td>Support Unicode symbols.</td>
</tr>
</tbody>
</table>
CSS Loading Strategy

Before

After

Tinder
new webpack.optimize.ModuleConcatenationPlugin(),
new webpack.DefinePlugin({
  'process.env': {
    // Env stuff...
  },
});

Webpack 2 -> 3 reduced JS parsing time by 8% (250ms -> 230ms)
React 15.x -> React 16: reduced vendor chunk size by ~6.7%
new workboxWebpackPlugin({
    injectManifest: true,
    swSrc: paths.serviceWorkerSrcPath,
    swDest: paths.serviceWorkerDestPath,
    globDirectory: paths.staticPath,
    globPatterns: [
        '**/*.{js,html,css,svg,woff2}'
    ],
    templatedUrls: {
        'index.html': [
            '../public/static/build/main-*.js',
            '../public/static/build/vendor-*.js',
            '../public/static/build/manifest-*.js',
            '../public/static/build/style.*.css'
        ]
    },
    maximumFileSizeToCacheInBytes: 4194304
})
IMPROVING PERFORMANCE IS A JOURNEY. LOTS OF SMALL CHANGES CAN LEAD TO BIG GAINS.
Links

Web Fundamentals
developers.google.com/web

Chrome User Experience Report
bit.ly/introducing-crux

React Perf Case Studies
medium.com/@addyosmani

Thanks!

Addy Osmani
@addyosman

Ilya Grigorik
@igrigorik

Bryan McQuade
@bryanmcquade

Come chat with us at the demo area!