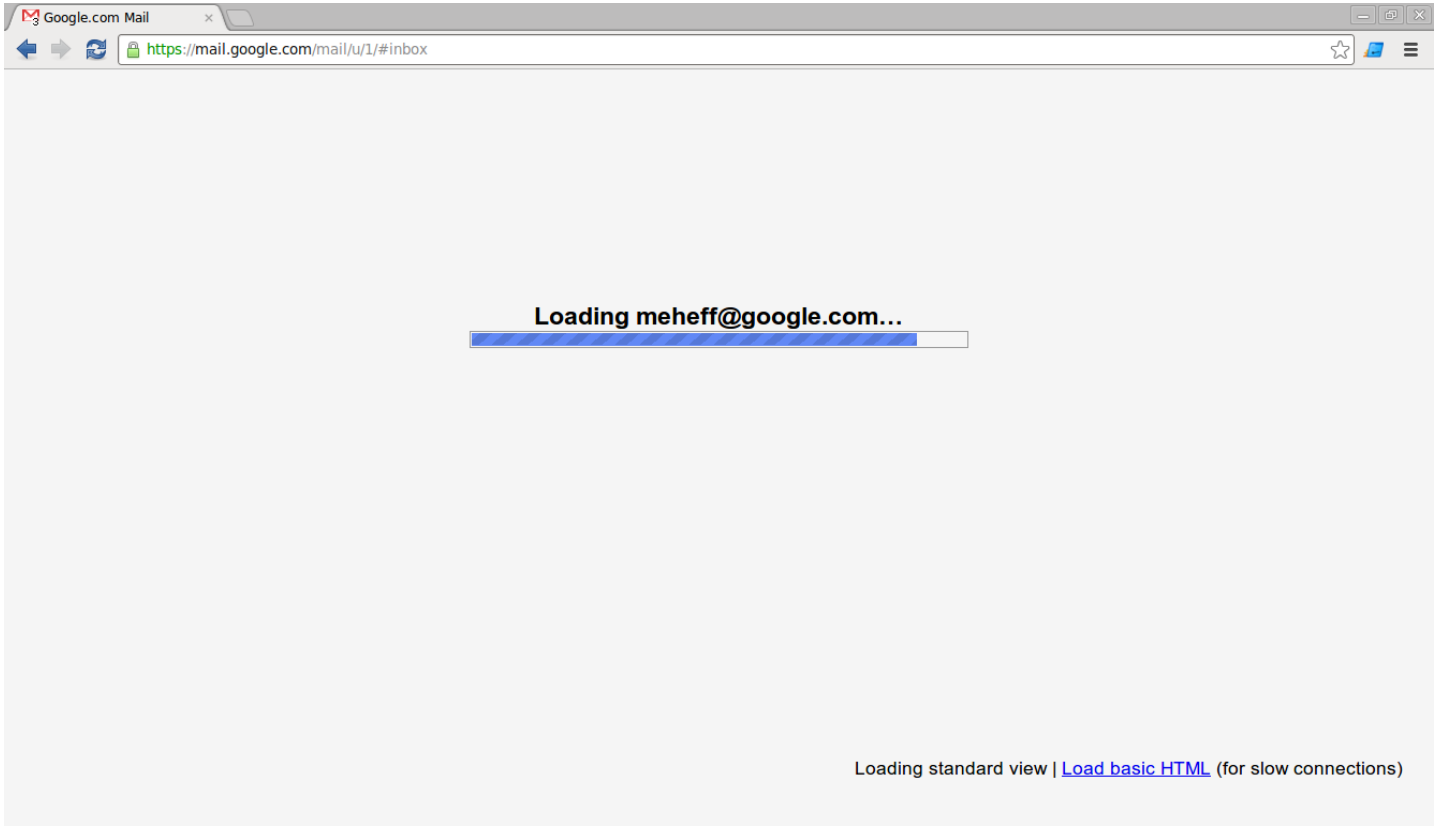
Browser Enhancements to Help Improve Page Load Performance Using Delta Delivery

W3C Performance Working Group
November 8th, 2012

Robert Hundt (rhundt@google.com)
Mark Heffernan (meheff@google.com)
Ian Flanigan (flan@google.com)

Google

Every day Gmail subjects users to a combined 61 years of this...



Initial Load

Initial load latency: Time from the user starting to navigating to App to rendering complete.

Gmail, Docs, Sheets

Median: Low seconds
90%'ile: Low 10's of seconds
99%'ile: Low minutes (in slow geographies)

Initial load is by far the slowest action our users commonly perform.

Initial Load

Robert's Goals

- **Bring initial load to < 1 sec at median**
- **Stretch: Bring initial load to < 1 sec at 90%-tile**

Initial Load

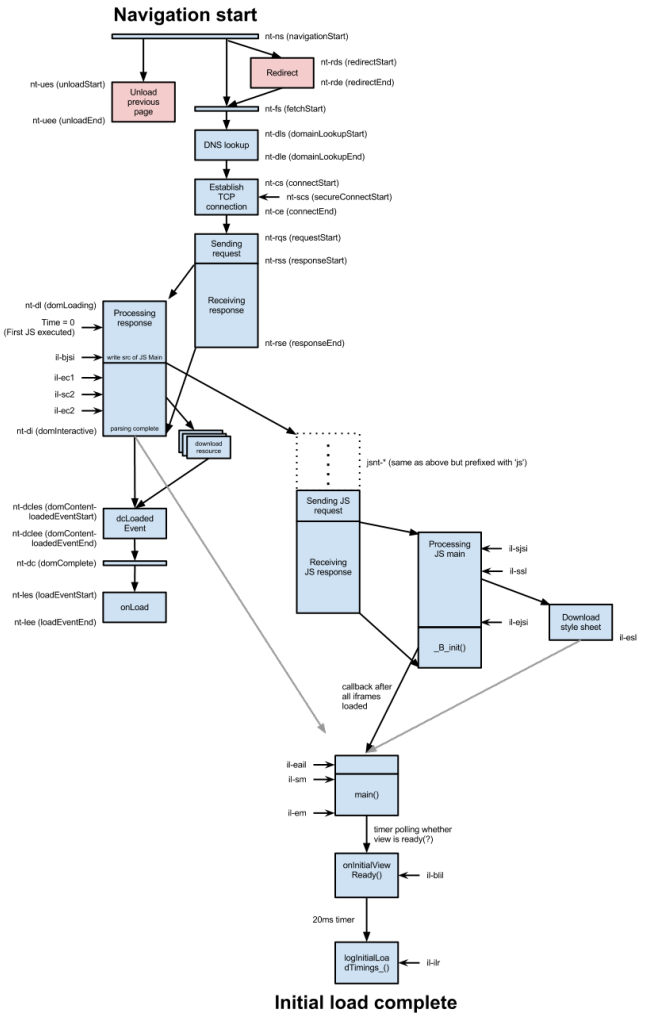
Robert's Goals

- **Bring initial load to < 1 sec at median**
- **Stretch: Bring initial load to < 1 sec at 90%-tile**

Ideas in this presentation won't quite get us there, but it'll be a big improvement.

Basics of Initial Load (Gmail)

Initial load sequence is complicated!



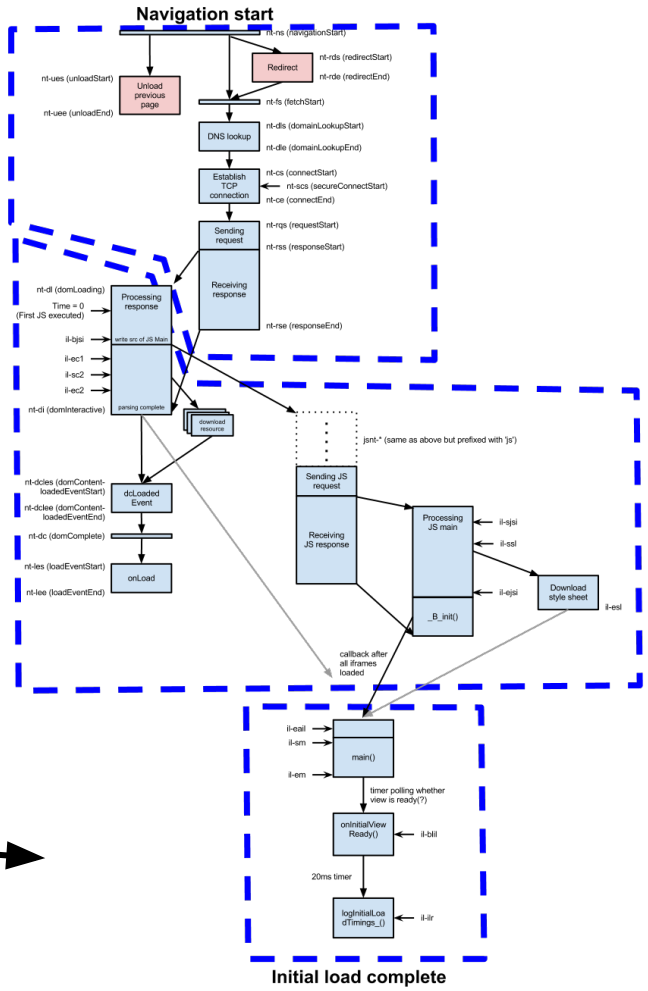
Basics of Initial Load

Three phases:

Phase 1
Redirects, establish connection

Phase 2
Bootstrap, download resources

Phase 3
Execute JS, build view

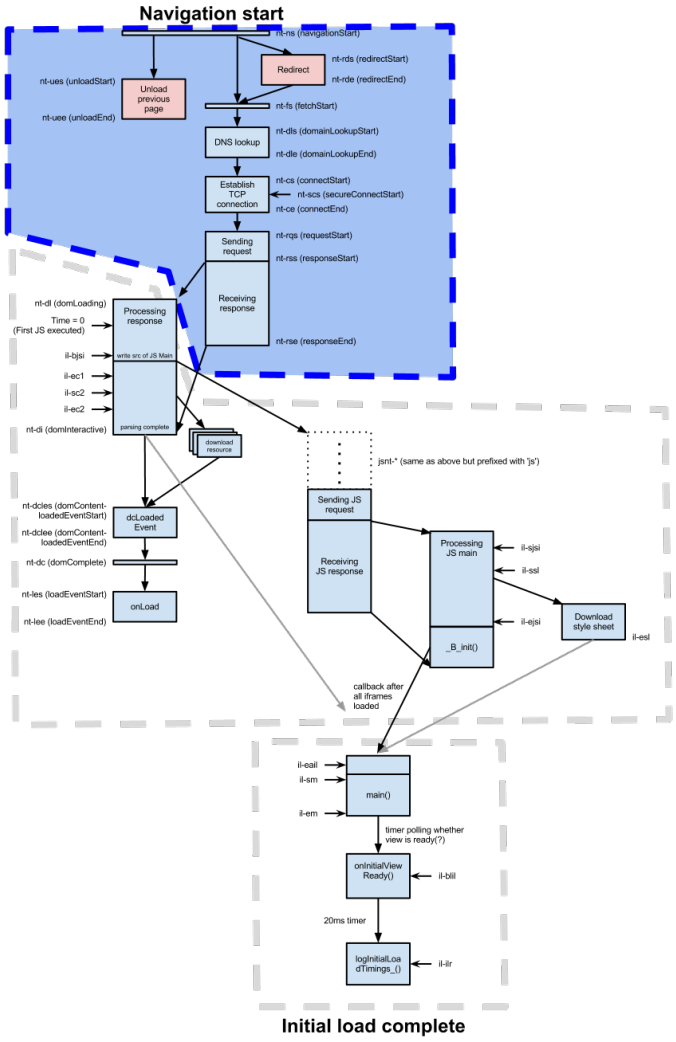


Initial load: Phase 1

Phase 1
Redirects, establish connection

Phase 2
Bootstrap, download resources

Phase 3
Execute JS, build view



Initial load: Phase 1

From navigation start to execution of first byte of JS.

Includes redirects, establishing connection(s).

Median	<2s
90%-tile	a few seconds
99%-tile	a couple of tens of seconds

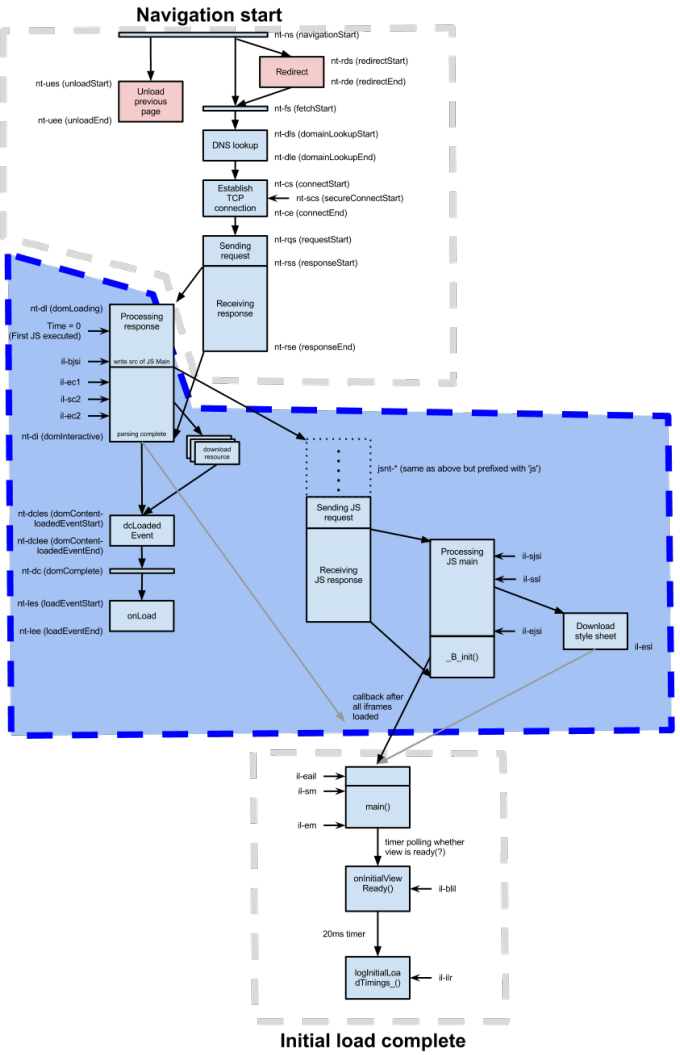
Measured with W3C Navigation Timing API.

Initial load: Phase 2

Phase 1
Redirects, establish connections

Phase 2
Bootstrap, download resources

Phase 3
Execute JS, build view



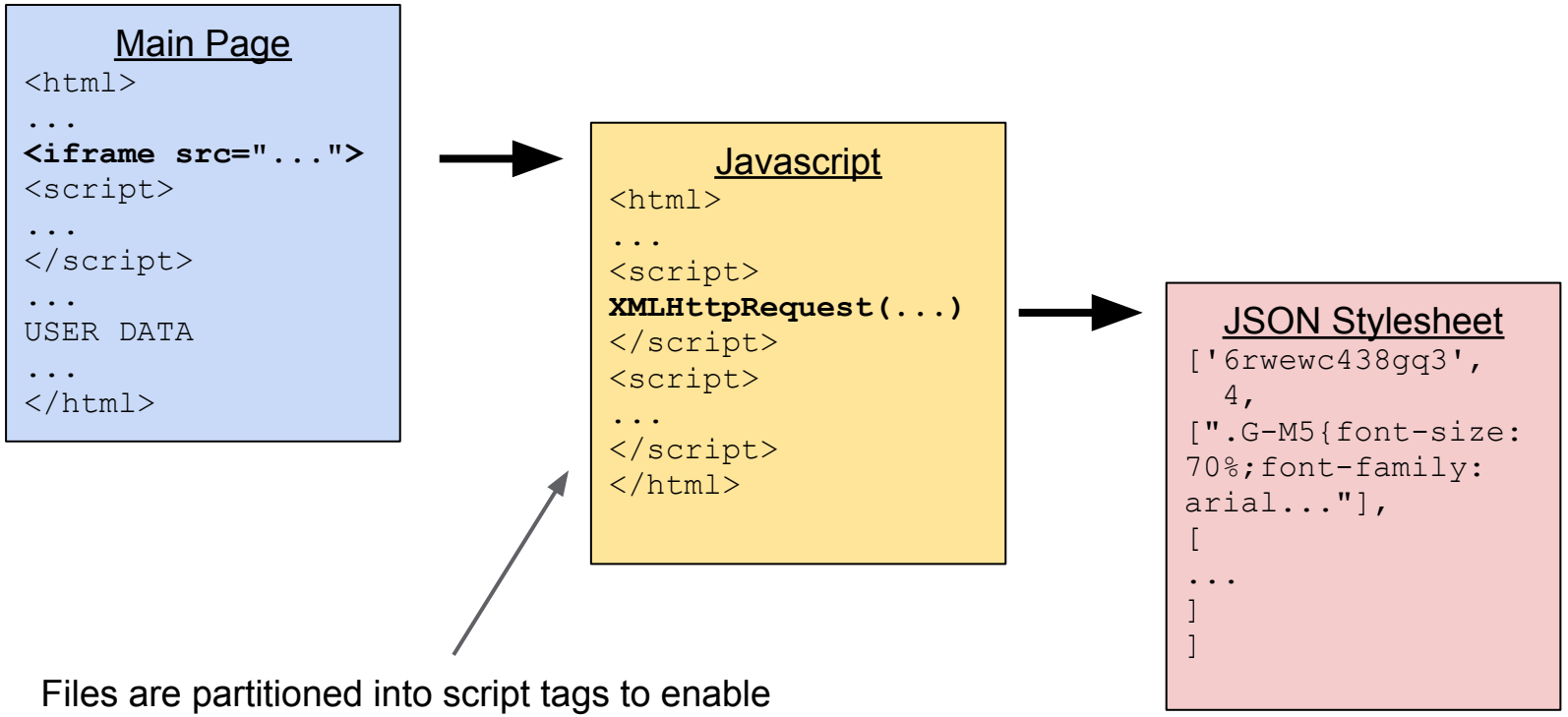
Initial load: Phase 2

Primarily three components:

- Main page: bootstrap and userdata (100kB), **not cacheable**
- Main Javascript (1.1MB), **cacheable**
- Stylesheet (300kB), **cacheable**

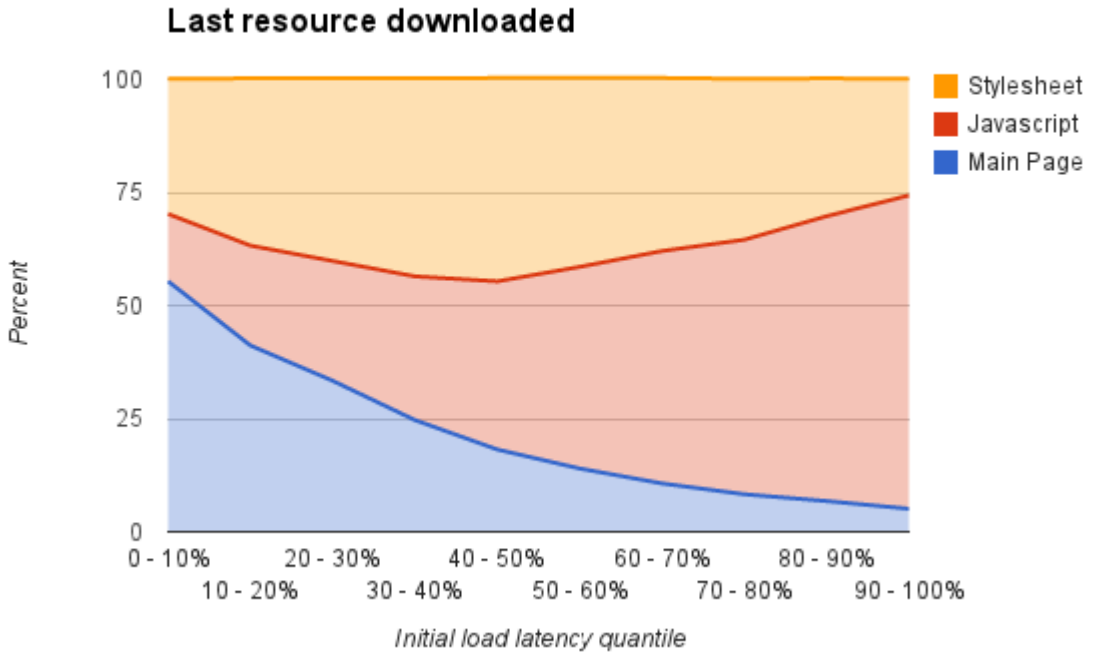
Initial load: Phase 2

Three resources are downloaded in a cascade:



Files are partitioned into script tags to enable pipelined parsing and execution.

Where is the bottleneck?



fast ~1s ⟸ ~5s ⟹ ~60s *slow*

- **Main page dominates faster initial loads.**
- **Javascript and stylesheet dominate slower initial loads.**

Reducing size of Javascript and Stylesheet

- Server doesn't send the entire resource.
- Instead, sends only the difference (delta) between what the client has (in cache or local storage) and the latest version.
- Delta is encoded with fast, efficient scheme called VCDIFF.
- **Delta can be *much* smaller than the entire resource**

We looked at:

RFC3229, SDCH, Bunch of internal implementations, proposals.

Google

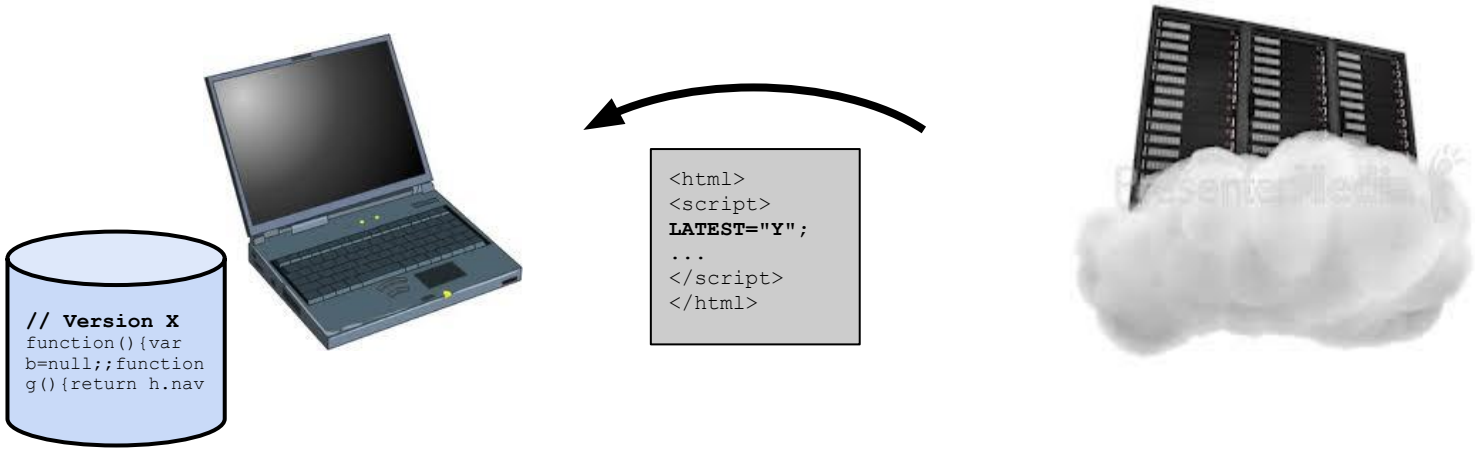
DeltaJS Basics

Client has Javascript of version X from previous session.

DeltaJS Basics

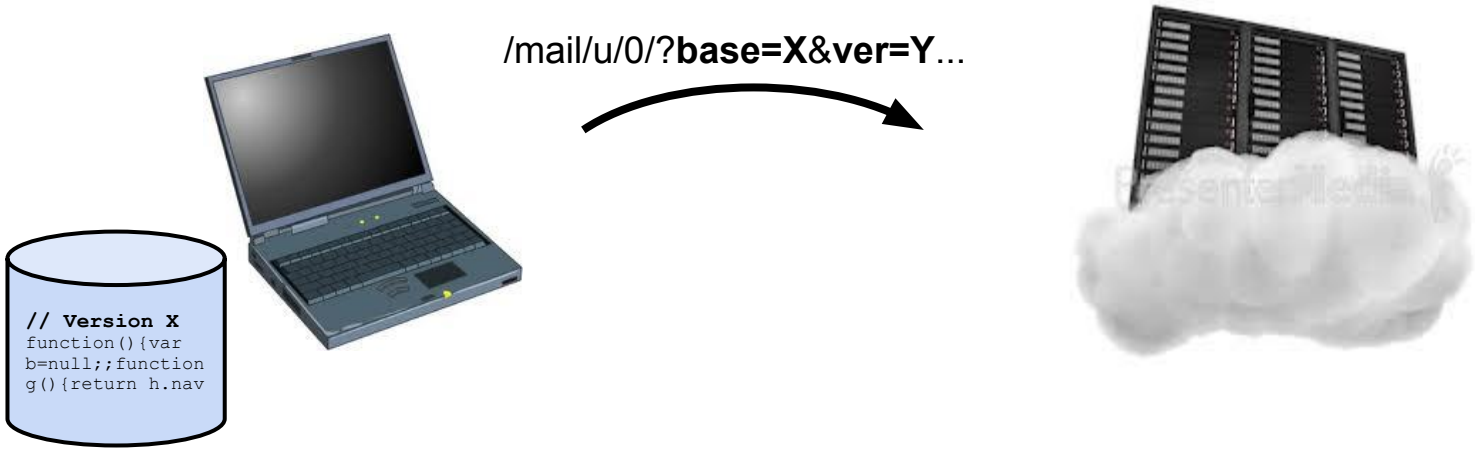
Client starts initial load with request to server.

DeltaJS Basics



Server response includes the latest version number (Y).

DeltaJS Basics



Client requests delta from X to Y.

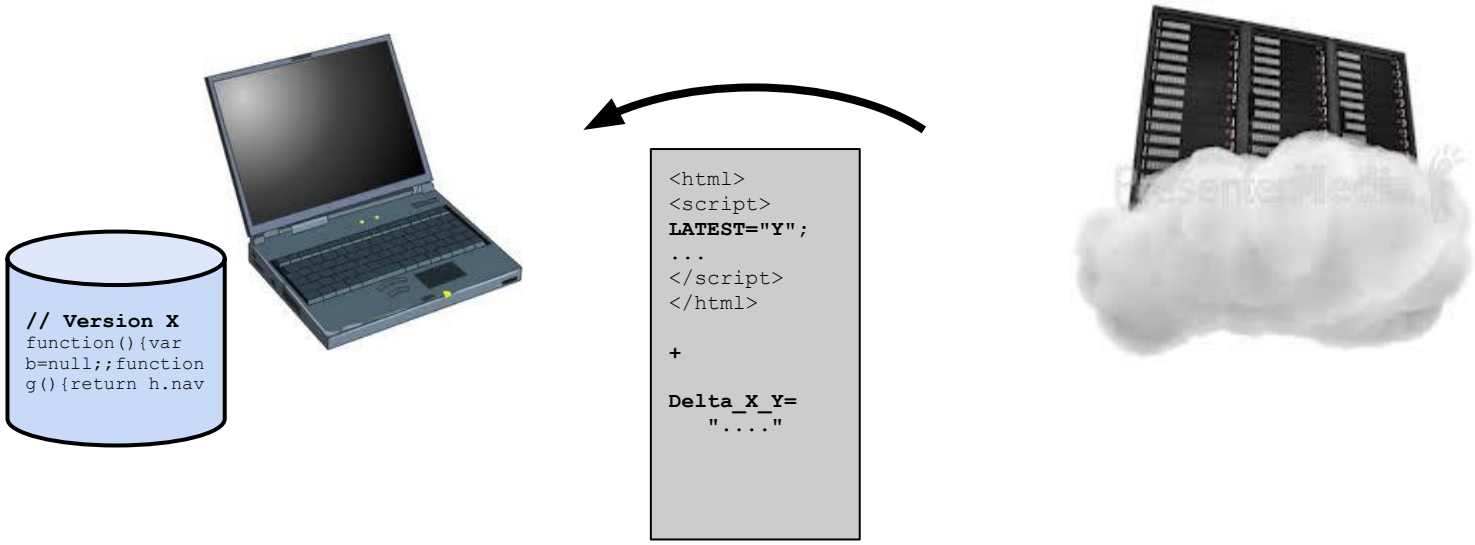
DeltaJS Basics

Server computes and returns encoded delta between version X and Y which is **much** smaller than Y itself.

DeltaJS Basics

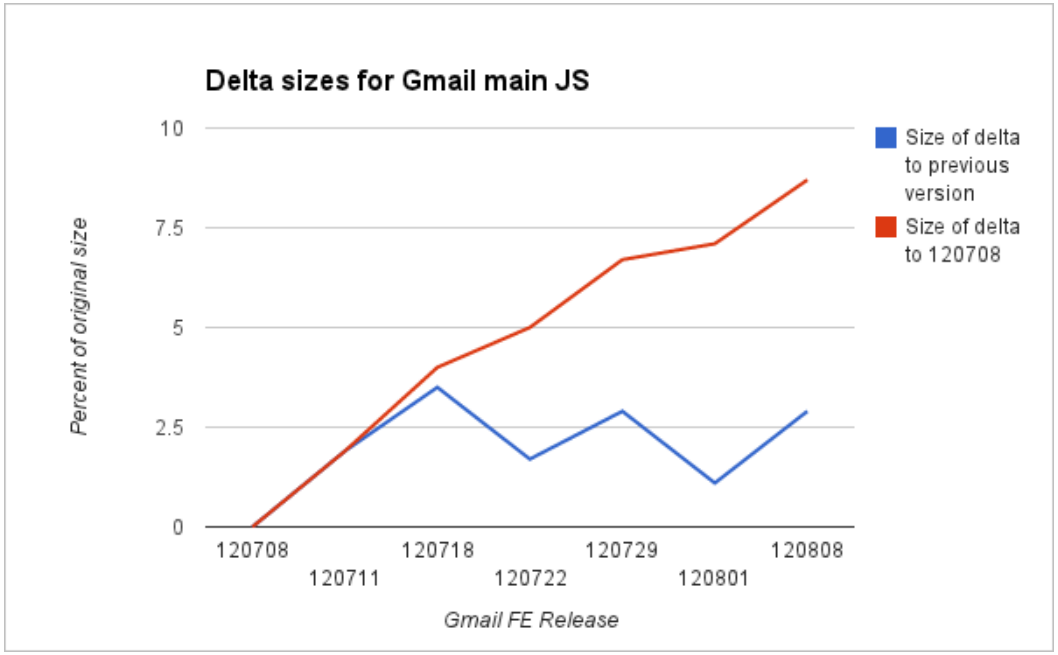
Client applies delta updating version X to Y and starts executing version Y.

Or... Speculative All-In-One



Initial server response includes the latest version number (Y), as well as the Delta to Last Used (X).

DeltaJS for Gmail



- **Version-to-versions deltas are ~2% original size (20kB).**
- **A whole month's worth of changes is <9% of original size.**

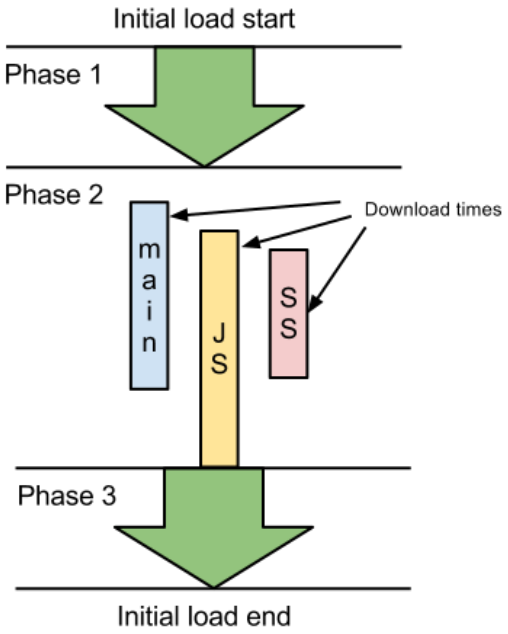
Google

DeltaJS for Gmail: Impact

Can we estimate the potential benefit?

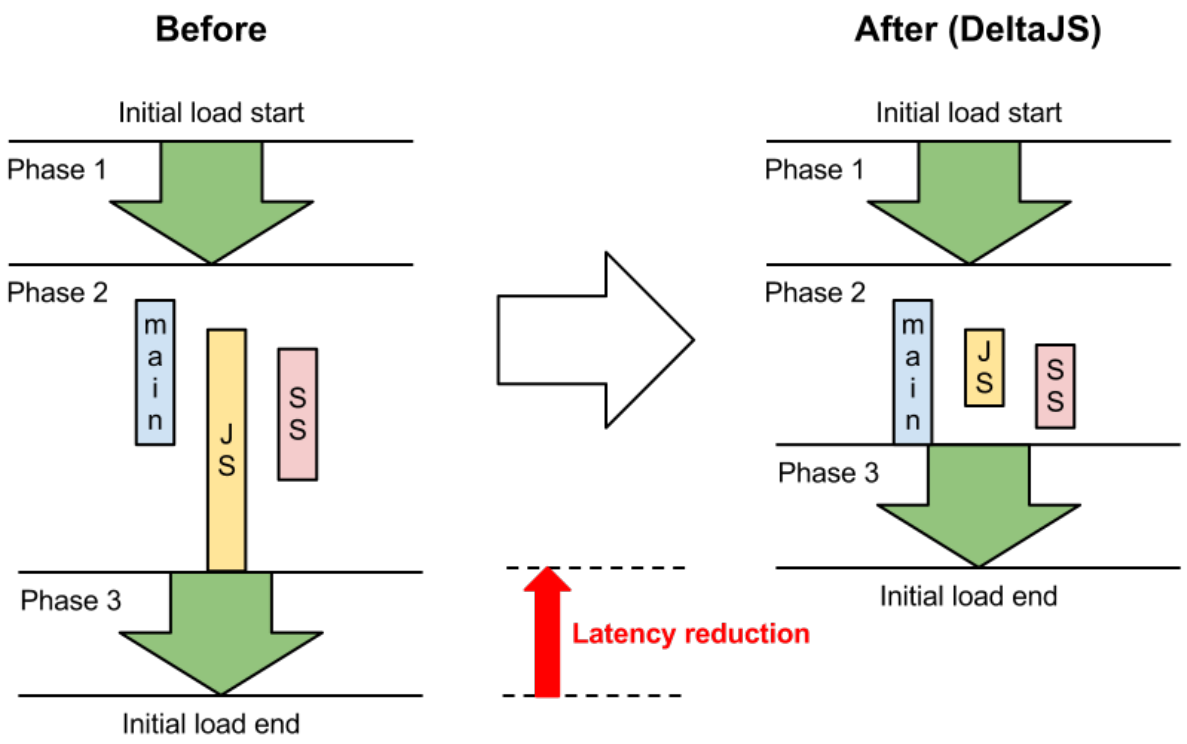
Model:

- Assume one of the three major resources (main page, JS, SS) is always on critical path.
- Simulate reduced download time* of JS and SS for a large sample of initial loads.



* Time from first byte received to last byte received.

DeltaJS for Gmail: Impact



Estimate is conservative: doesn't account for faster main page downloading by no longer sharing bandwidth with JS and SS.

DeltaJS for Gmail: Impact

What if JS and SS download times are reduced by 95%?

	improvement
Median	**11%**
90%-tile	**30%**
90%-tile India	**48%**
99%-tile	**42%**
99%-tile India	**52%**

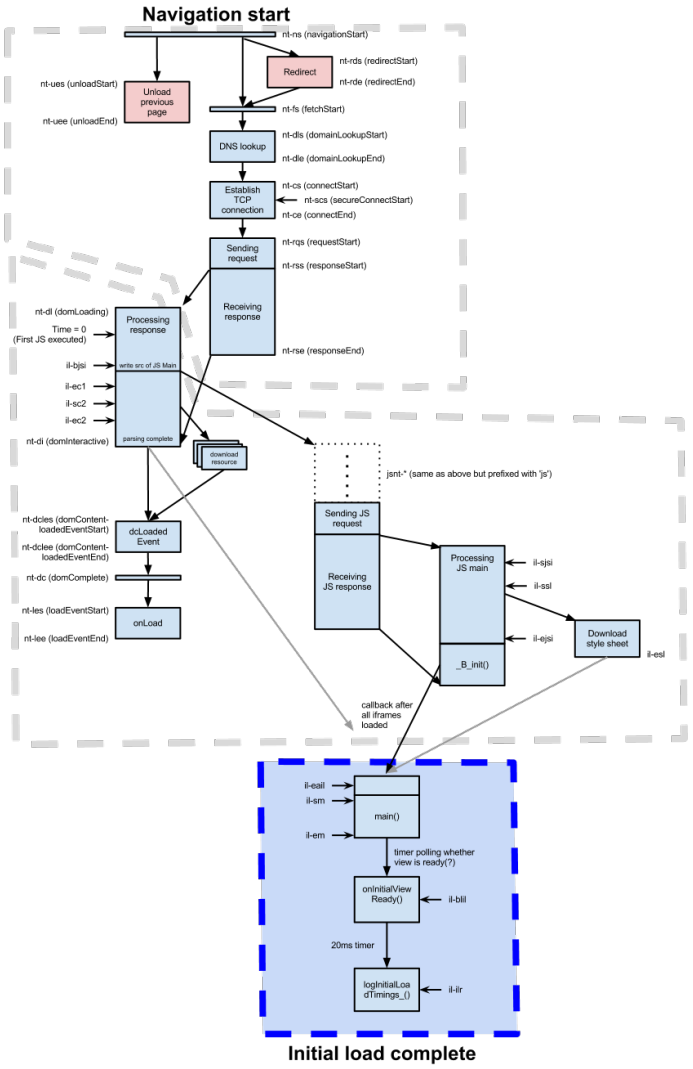
Initial load latency could be reduced by about half for high percentiles!

Initial load: Phase 3

Phase 1
Redirects, establish connections

Phase 2
Bootstrap, download resources

Phase 3
Execute JS, build view



Google

Browser Platform Enhancements

Challenge: Avoid regressions on Fast Path

We're thinking of 4 key enhancements

1. Sha256, VCDiff in browser
2. Pre-Heat JS engine
3. Protocol Change
4. Streaming VCDiff

Browser/Platform Enhancements - 1 / 4

Sha256 in Chrome

- Hashing 1.1MB JS: **~300ms**
- Hashing 1.1MB C++: **~10ms**, *30x faster*

Proposal:

- Expose JS interface to Sha256 (crypto API?)
- Same thing for VCDIFF

Browser/Platform Enhancements - 2 / 4

Pre-Heat Storage / JS engine during connection wait time:

- Pre-Load - "all" of LocalStorage
- Pre-Parse - load and pre-parse JS (find function start/end points)
- Pre-Compile - load, pre-parse, and pre-compile JS

And maybe even:

- Pre-Run - load, compile, and run. Might introduce fantastically complicated side-effects.

Browser/Platform Enhancements - 3 / 4

Delta-Application via HTTP protocol change (vague idea):

- Server tells browser to explicitly cache/retain resource, e.g., via response header

```
X-Delta-Version: 12345abc
```

- When requesting (cached) resource, browser adds:

```
X-Delta-Previous-Version: 12345abc
```

- Server returns full resource, or Delta marked as:

```
Content-Type: delta/x-vcdiff
```

with additional headers, e.g.:

```
C-Delta-Version: 1ontuhno
X-Delta-SHA256: a3498249ebbeec23
```

- Big Benefit: Transparent to users (client code)

Browser/Platform Enhancements - 4 / 4

Streaming Delta Application

- Pipeline parallelism between
  - JS download
  - VCDIFF
  - and parsing

- Modify delta format and add "safe points":
  "it is safe to parse/execute up to this point in the original/modified resource"

Google

Thanks. Q / A

4 key proposed enhancements

1. Sha256, VCDiff in browser
2. Pre-Heat JS engine
3. Protocol Change (vague)
4. Streaming VCDiff