
Google™
10

Architecting for Performance with Google Web Toolkit

Adam Schuck (Wave), Joel Webber (GWT)

May 2010

View live notes and ask questions about this session on Google Wave

http://bit.ly/io2010-gwt3

Why architect for performance?

- Speed matters!
- Response time limits:
  - **0.1 seconds:** user feels the system is instantaneous
  - **1 second:** user's flow of thought stays uninterrupted
  - **10 seconds:** keeps the user's attention focused

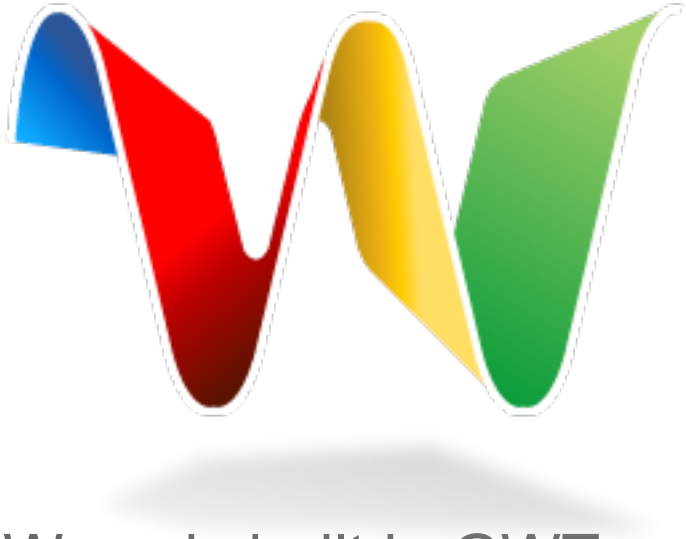
Source: http://www.useit.com/papers/responsetime.html

Theory and Practice

- The GWT and Wave perspectives...
- Joel: the toolkit creator

- Making GWT faster
- Making *your app* faster!

- Adam: the toolkit user



- Wave is built in GWT
- Lessons from the the trenches
- Wave team still learning! (Demo)

Today's Talk

- Using GWT to improve your app's slow points:
  - Startup
  - Fetching data
  - Rendering
  - User interactions
- Performance measurement

Startup

Startup

Where does the time go?

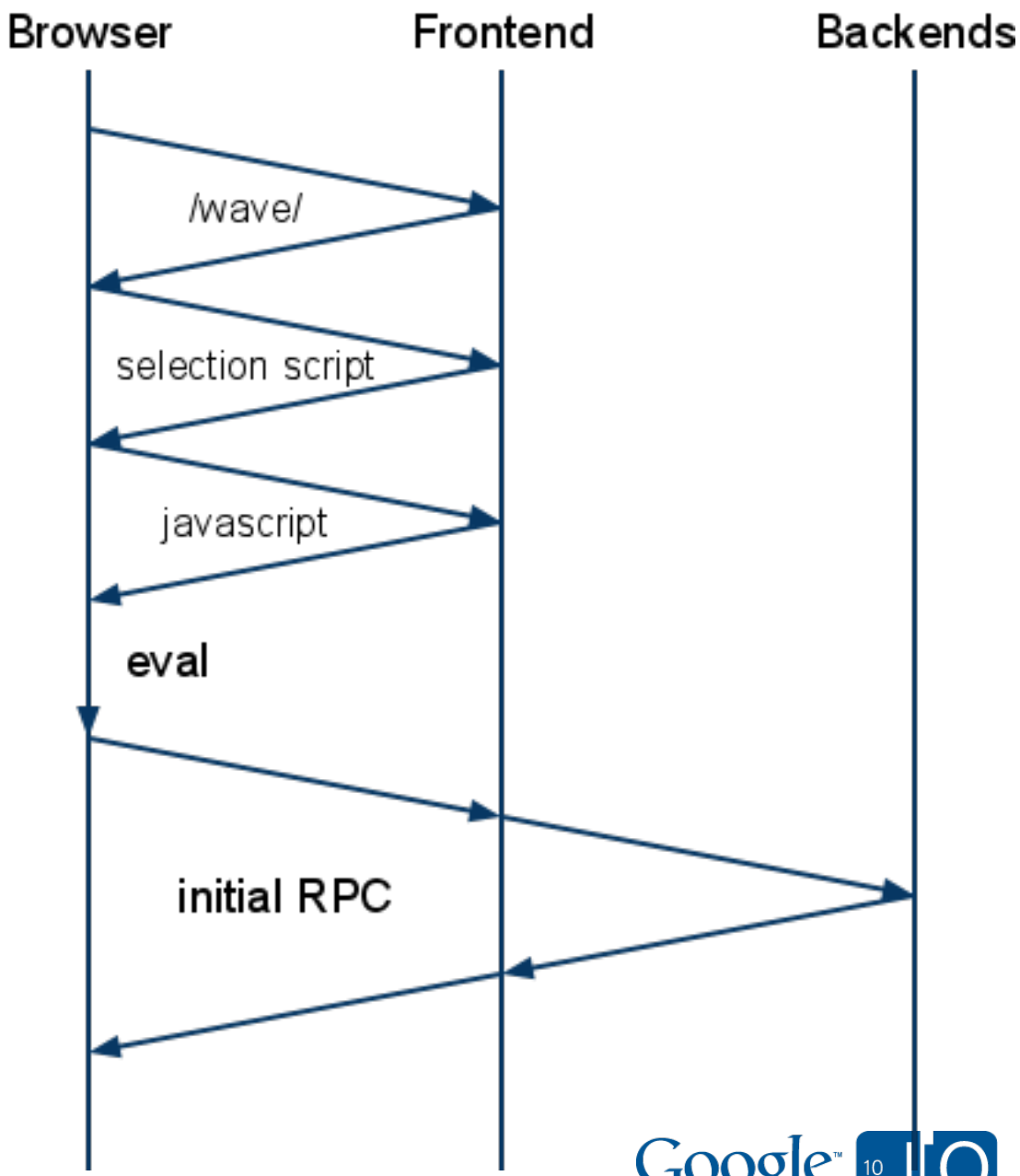
- Fetching script
- Evaluating script
- Fetching initial data
- Building application structure

Startup Sequence

Default GWT setup:

- Serial execution
- 4 round trips
  - Host page
  - Selection Script
  - Compiled Script
  - Initial Data

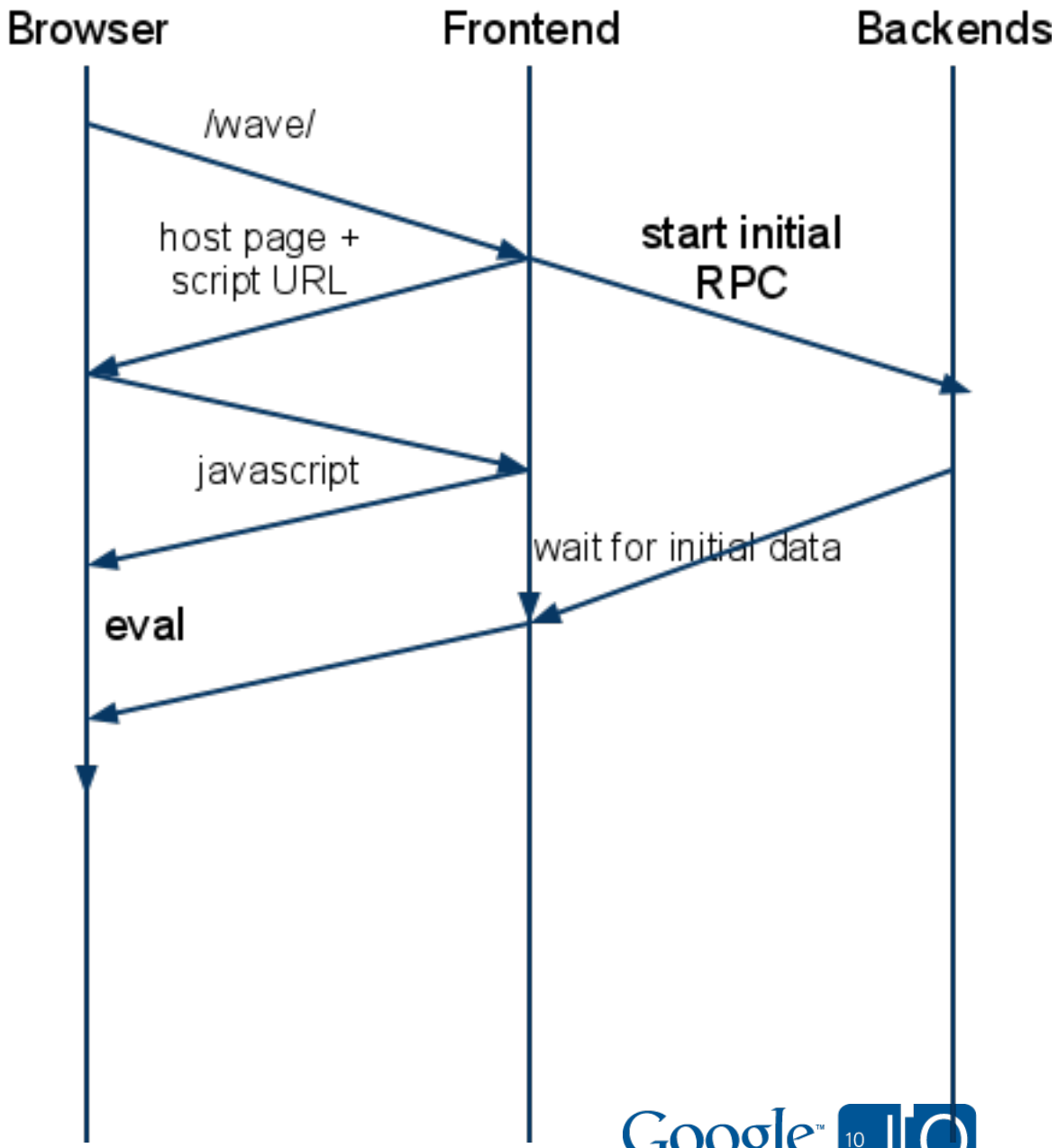
How can we do better?



Startup Sequence

Better:

- 2 round trips
  - First:
    - Host Page
    - Script URL
    - Initial Data
  - Second:
    - Compiled Script
- JS + data in parallel
- Simple data format w/o requiring RPC system



Code Splitting

- What is it?
- Download as little as possible to get started
  - But not *too* little!
  - Goal: One fragment for the initial page
  - Prefetch the rest
- Demo
- Wave experience
  - Optimized for loaded inbox
  - Use GWT Compile Reports

Wave's Startup

- Preloading data: "Fast Start" Data
  - Server anticipates initial requests from client
  - Initial download page uses chunked HTTP response
  - Send the data when it is available from the servers
  - Sequence in increasing order of expected load speed
  - In practice: we optimize for inbox, so that goes first
- Code splitting:
  - Initial inbox view all in initial download
- Results:
  - Median: was 5 secs, now 2 secs
  - 90th %ile: was 16 secs, now 7 secs

Future Work

- Server-side script selection
  - Instead of using JS sniffer code to determine permutation, save a round trip
  - Requires choosing permutation based on HTTP headers only
- Currently must be hand-rolled in a linker
  - Working on bringing to GWT proper

Fetching Data

Fetching Data

Where does the time go?

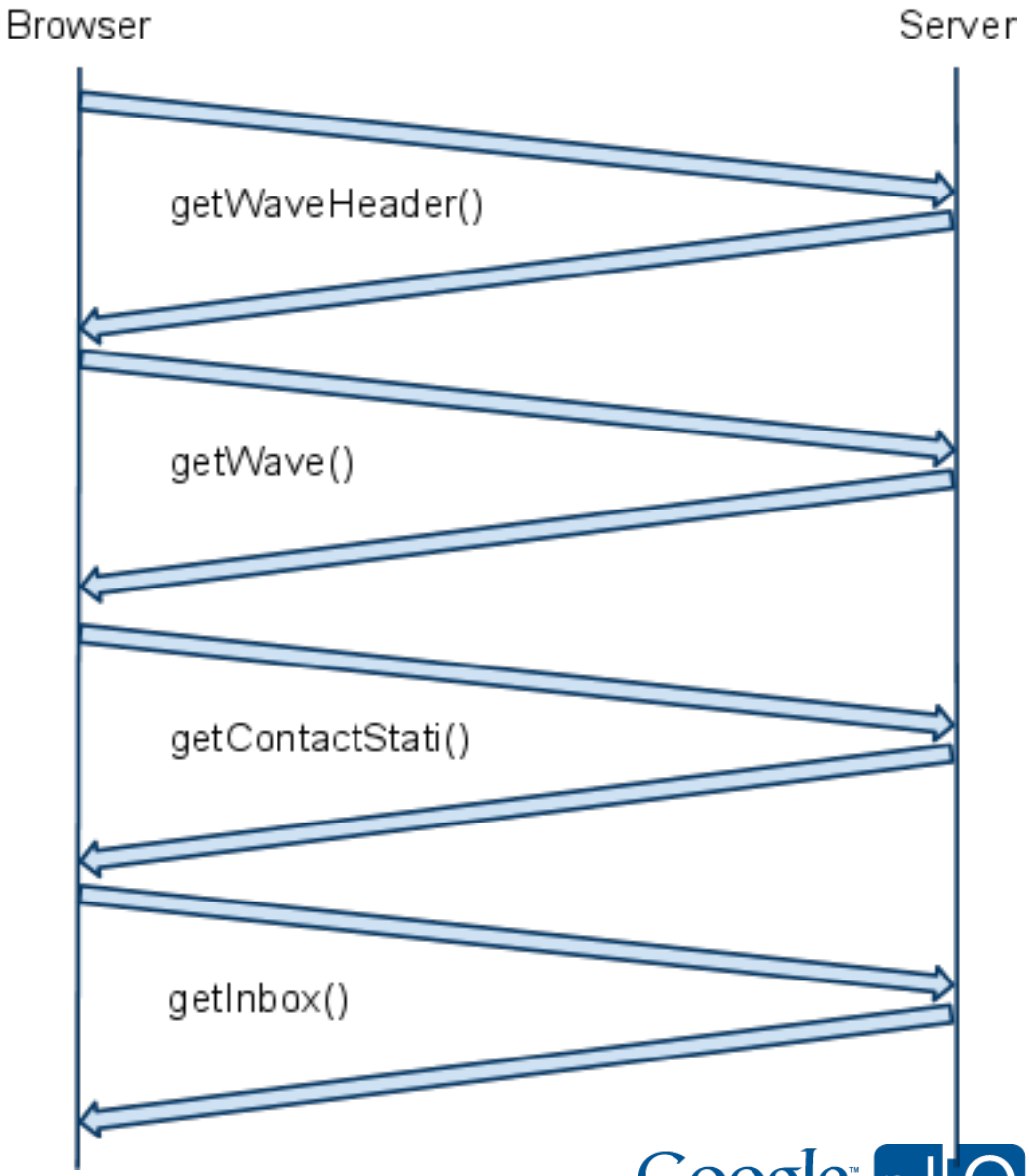
- Fetching data you don't need
- Too many HTTP requests

Design RPC interfaces carefully

- Design server interface to support the UI
- Overly-generic services lead to unnecessary data and requests
- Fetch only what you need
- Pay attention to types being serialized by RPC

Batch Requests

```
interface Service {
  WaveHeader getWaveHeader(id);
  Wave getWave(id);
  Status[] getContactStati();
  Header[] getInbox();
}
```

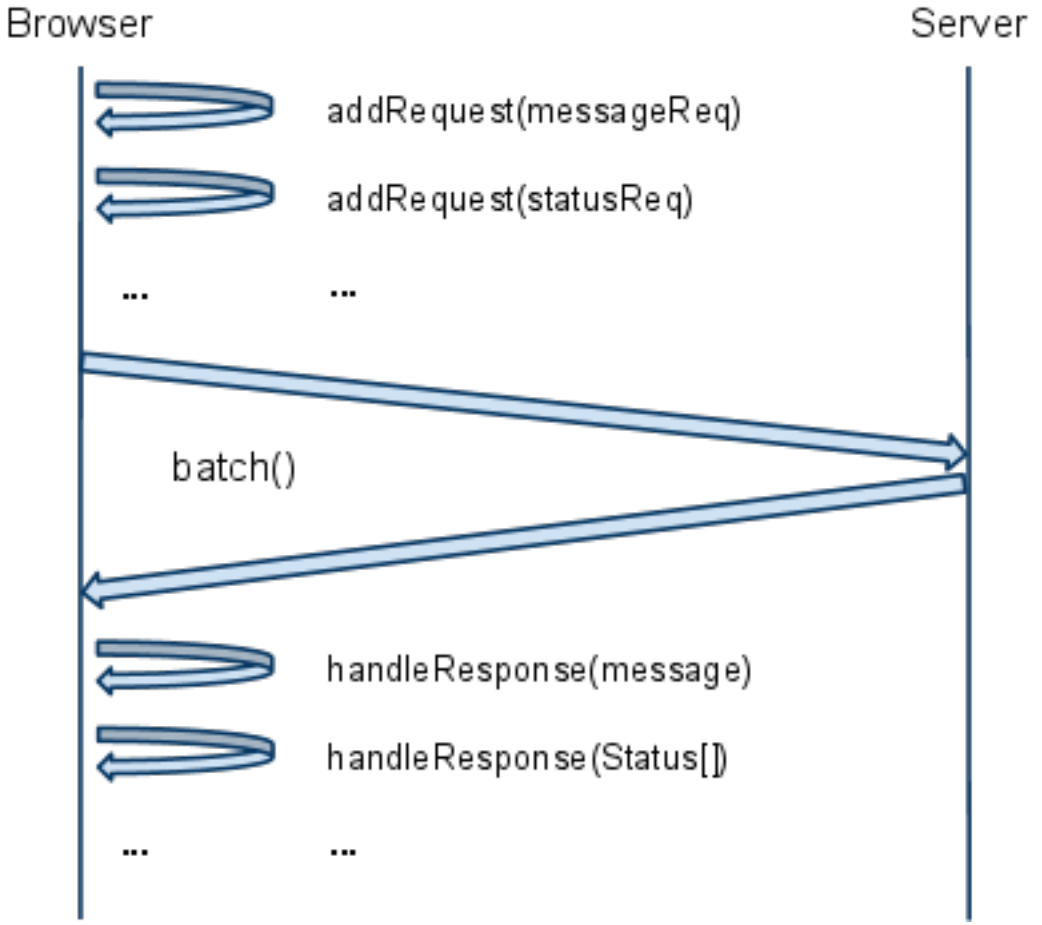


Four serialized HTTP requests!

Batch Requests

```
interface Service {
 Response[] batch(Request[]);
}

void addRequest(Request req) {
 // use DeferredCommand to queue
 // requests
}
```



One HTTP request per user action

Rendering

Rendering

Where does the time go?

- Creating widgets
  - Too early
  - When not needed
- Populating widgets with data

Lazy Initialization

- If you don't need it yet, defer it
- LazyPanel encodes this pattern

```
LazyPanel lazy = new LazyPanel() {
  protected Widget createWidget() {
    return new MonsterWidget();
  }
};
tabPanel.add(lazy, "monster");
```

Using GWT Widgets

When should I use widgets?

- When a component must receive events AND
- There's no way to catch events in the parent widget

When should I *not* use widgets?

- When HTML elements will suffice
- UiBinder can help!

How to eliminate widgets

- Use UiBinder to replace widgets with HTML
- Cell-based Lists, Tables, and Trees
  - Use flyweight "cells" for rendering items efficiently with innerHTML
  - Middle-ground between widgets and HTML
- See "GWT's UI Overhaul" for more information
  - Tomorrow at 10:15am

UiBinder

- Templates can use both HTML and Widgets
- Use HTML wherever possible
- Use optimized CSS (easy with UiBinder)
  - Simple CSS rules are the fastest
    - #foo { }
    - .bar { }
  - Descendant selectors can be quite slow
    - .foo .bar div { }

HTML & CSS with UiBinder

```
<div ui:field='container' class='{css.container}'>
  <div ui:field='avatar' class='{css.avatar}'>
    <div ui:field='imagePlaceholder'/>
    <div ui:field='status'/>
  </div>
  <div class='{css.details}'>
    <div class='{css.name}' ui:field='name'/>
    <div class='{css.mood}' ui:field='message'/>
  </div>
  <div class='{css.tick}'/>
</div>
```

HTML & CSS with UiBinder

```
<div ui:field='container' class='{css.container}'>
  <div ui:field='avatar' class='{css.avatar}'>
    <div ui:field='imagePlaceholder'/>
    <div ui:field='status'/>
  </div>
  <div class='{css.details}'>
    <div class='{css.name}' ui:field='name'/>
    <div class='{css.mood}' ui:field='message'/>
  </div>
  <div class='{css.tick}'/>
</div>
```

```
@UiField Element container;
@UiField Element avatar;
@UiField Element imagePlaceholder;
@UiField Element status;
@UiField Element name;
@UiField Element message;
```

HTML & CSS with UiBinder

```
<div ui:field='container' class='{css.container}'>
  <div ui:field='avatar' class='{css.avatar}'>
    <div ui:field='imagePlaceholder'/>
    <div ui:field='status'/>
  </div>
  <div class='{css.details}'>
    <div class='{css.name}' ui:field='name'/>
    <div class='{css.mood}' ui:field='message'/>
  </div>
  <div class='{css.tick}'/>
</div>
```

```
<g:style>
  .avatar { /* ... */ }  .name { /* ... */ } .container { /* ... */ }
  .details { /* ... */ } .mood { /* ... */ } .tick { /* ... */ }
</g:style>
```

Demo: Expenses Sample

Future Work

- On-demand rendering (aka "infinite scrolling")
  - Idea: keep the number of visible items bounded by screen real-estate
  - Can help with large collections
  - Wave has implemented for blips and inbox
  - New list widgets pave the way for doing this in GWT proper
  - 4x improvement for large waves for this approach

User Interactions

User Interactions

How fast should client-side interactions be?

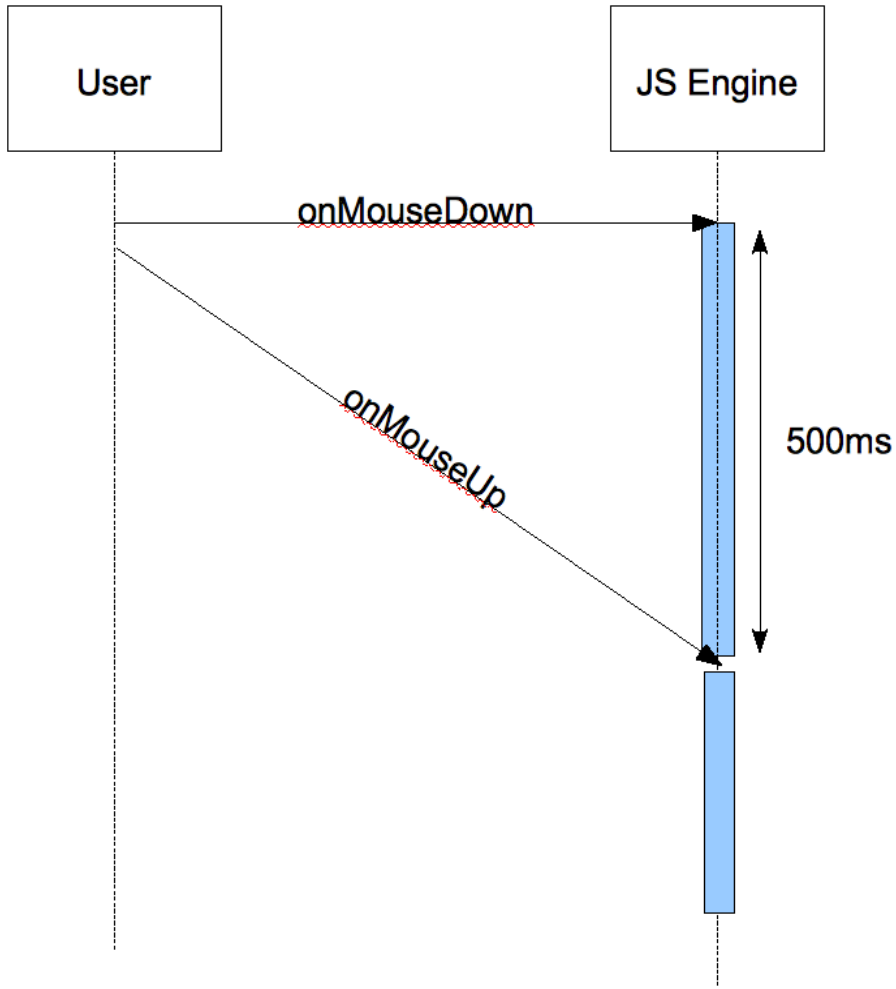
- 100ms or less
- See Kelly Norton's I/O 2009 talk, "Measuring in Milliseconds"

Where does the time go?

- Slow event handlers block the UI thread!
- Click events: buttons should do something quickly
- Mouse events: slow dragging and hover effects
- Key events: slow typing and navigation
- Layout: sluggish rendering and window resizing

Keep the application responsive

- JS is single threaded
- Events cannot be processed if JS is already running
- DeferredCommand allows displaying changes to the user before processing



Faster Layout

- LayoutPanels
  - Work around inefficient layout mechanisms
  - Leverages the browser for 99% of layout work
  - More predictable than old table-based layout
- See "GWT's UI Overhaul" for more information
  - Tomorrow at 10:15am

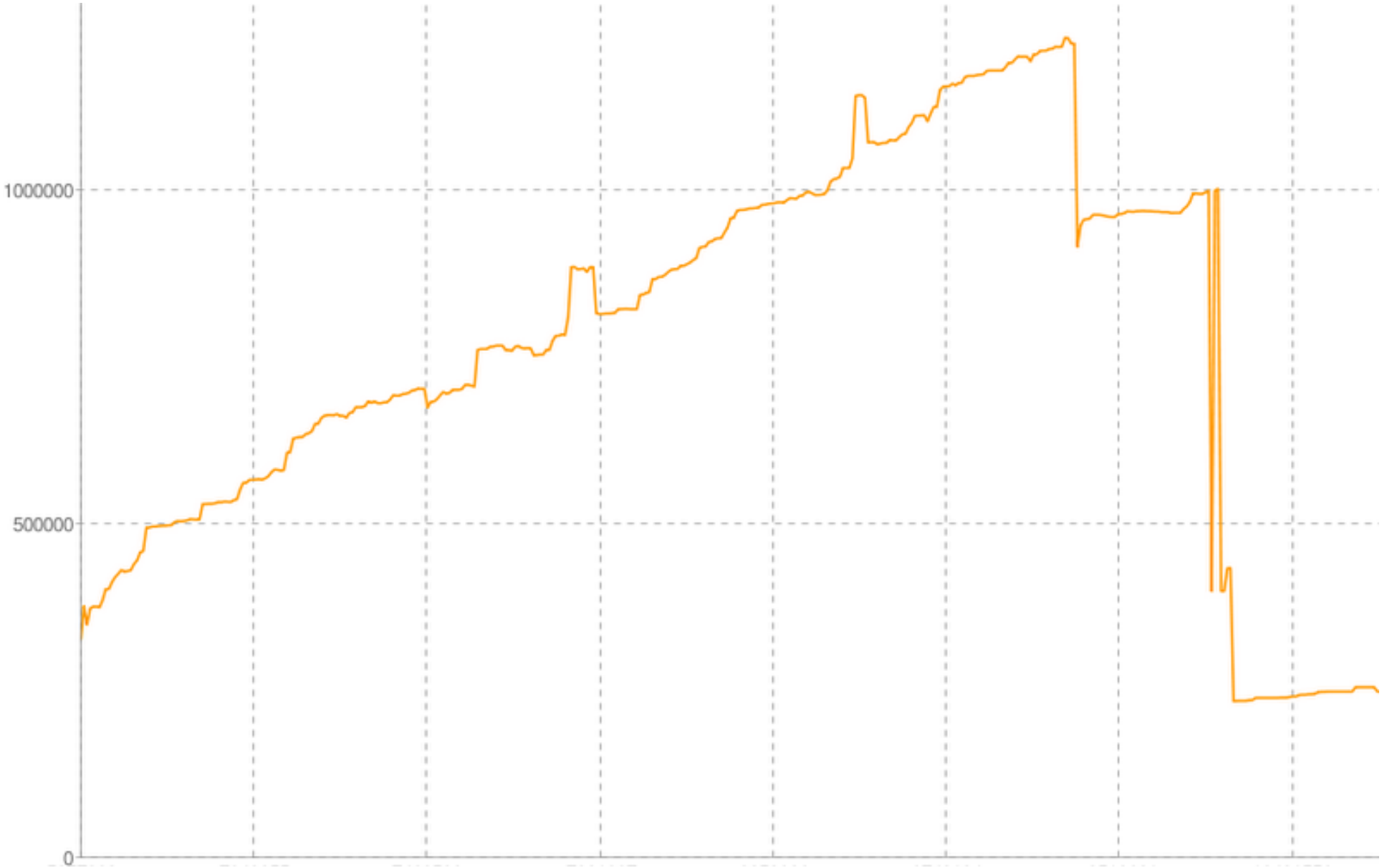
Performance Measurement

How can I keep my app fast?

Latency regression testing

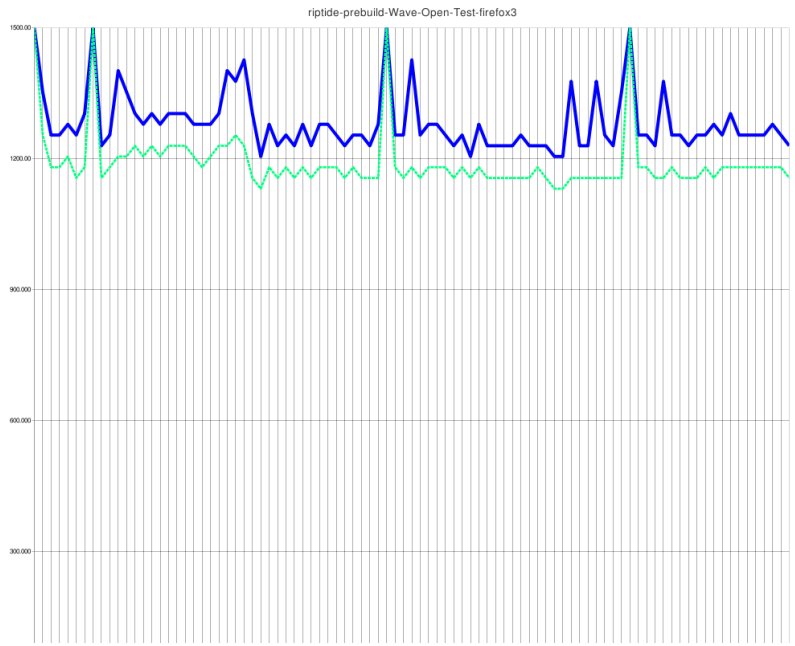
- Plot your download size (initial & total)
- Measure your key timings
  - in production conditions, and
  - in lab conditions

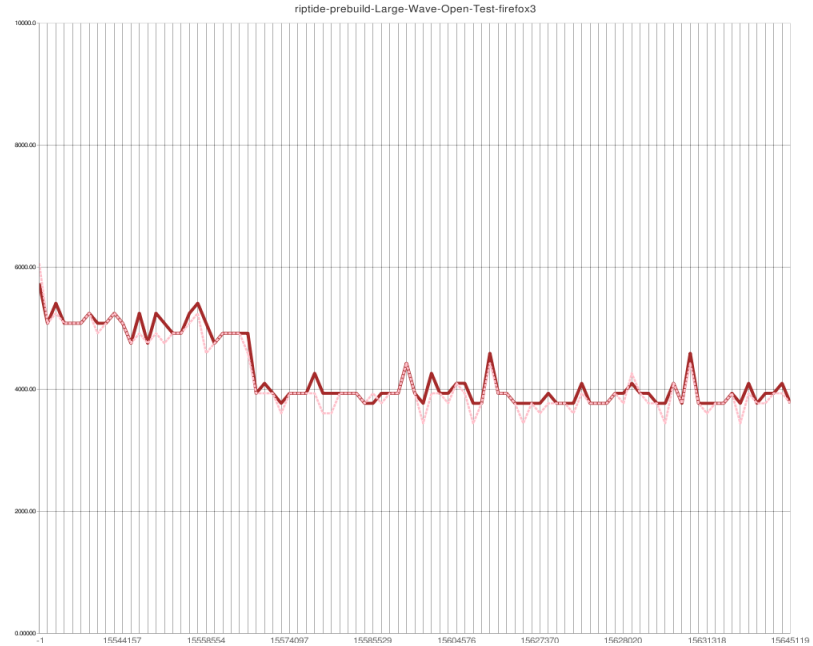
Plot your download size

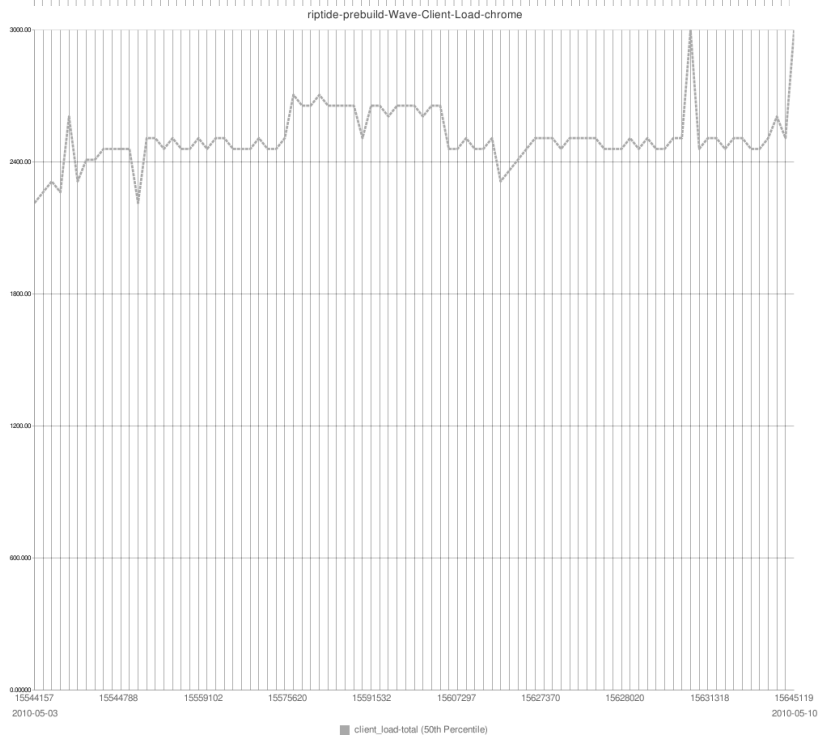


Production Timings

Lab Timings


riptide-prebuild-Wave-Open-Test-firefox3
1500.00
1200.00
900.000
600.000
300.000


riptide-prebuild-Large-Wave-Open-Test-firefox3
10000.0
8000.00
6000.00
4000.00
2000.00
0.00000
-1
15544157
15558554
15574097
15585529
15604576
15627370
15628020
15631318
15645119


riptide-prebuild-Wave-Client-Load-chrome
3000.00
2400.00
1800.00
1200.00
600.000
0.00000
15544157
15544788
15559102
15575620
15591532
15607297
15627370
15628020
15631318
15645119
2010-05-03
2010-05-10
client_load-total (50th Percentile)


riptide-prebuild-Wave-Client-Load-firefox3
3000.00
2400.00
1800.00
1200.00
600.000
0.00000
-1
15544157
15558554
15574097
15585529
15604576
15627370
15628020
15631318
15645119
2010-05-07
2010-05-10
client_load-total (50th Percentile)

How can I keep my app fast?

Tools

- Speed Tracer
  - Great for debugging deep browser behavior
- GWT Inspector Widget
  - How many widgets are on this page?
- Page Speed
  - Provides advice on startup performance

Demo: Wave Open Timing

Odds and ends

Compiler options

- Try to make all optimizations default
- Some non-default (minor Java semantic violations)
  - -XdisableClassMetadata
  - -XdisableCastChecking
  - Possibly more to come
- See Ray Cromwell's talk "Faster Apps Faster"
  - Wednesday at 12:30, Room 7

Go forth and speed up!

- Use GWT to improve your key timings:
  - Startup
  - Data fetching
  - Rendering
  - User interactions
- Keep it fast: Measure and track.

View live notes and ask questions about this session on Google Wave

http://bit.ly/io2010-gwt3


Google™
10