Why is this web app running slowly?

Performance Engineering: Theory & Practice

Web Application Performance

- Topics:
  - Building scalable web applications
  - YSlow scalability model: page composition
  - Page Load Time and Round trips
  - W3C Nav/Timing API
  - Google Analytics & RUM
  - ETW-based web application performance monitoring

  - Case study
    - a server-side ASP.NET application (with lots of buggy JavaScript)

Web Application Performance

- Themes:
  - The Value of Response Time measurements
    - Service Level reporting
      - Application response time measurements correlate with measures of customer productivity & satisfaction
    - Queuing models, decomposition & other analytic techniques

    - Obstacles:
      - acquiring important Measurement data
      - Request-Response boundaries are blurred in many AJAX applications
      - Understanding how to set good response time objectives
      - Since human beings are adaptable, “Good” and “Bad” response times are often relative to the application context
        - See *Engineering Time*, by Dr. Steve Seow

Web Application Performance

- Themes:
  - Integrating software performance engineering into the software development life cycle
    - Create a positive feedback loop using instrumentation to measure performance and reinforce software quality
    - Understand the relationship between responsiveness and customer satisfaction
    - Application code needs to be instrumented so that performance can be tracked against the goals during development, test, QA testing, and Production
      - Once the code is instrumented properly, *any* test can be a performance test
      - The instrumentation remains useful in production (DevOps)
        - Diagnose performance problems once the app hits production
        - Uncover and diagnose performance regressions during subsequent development cycles

Web Application Performance

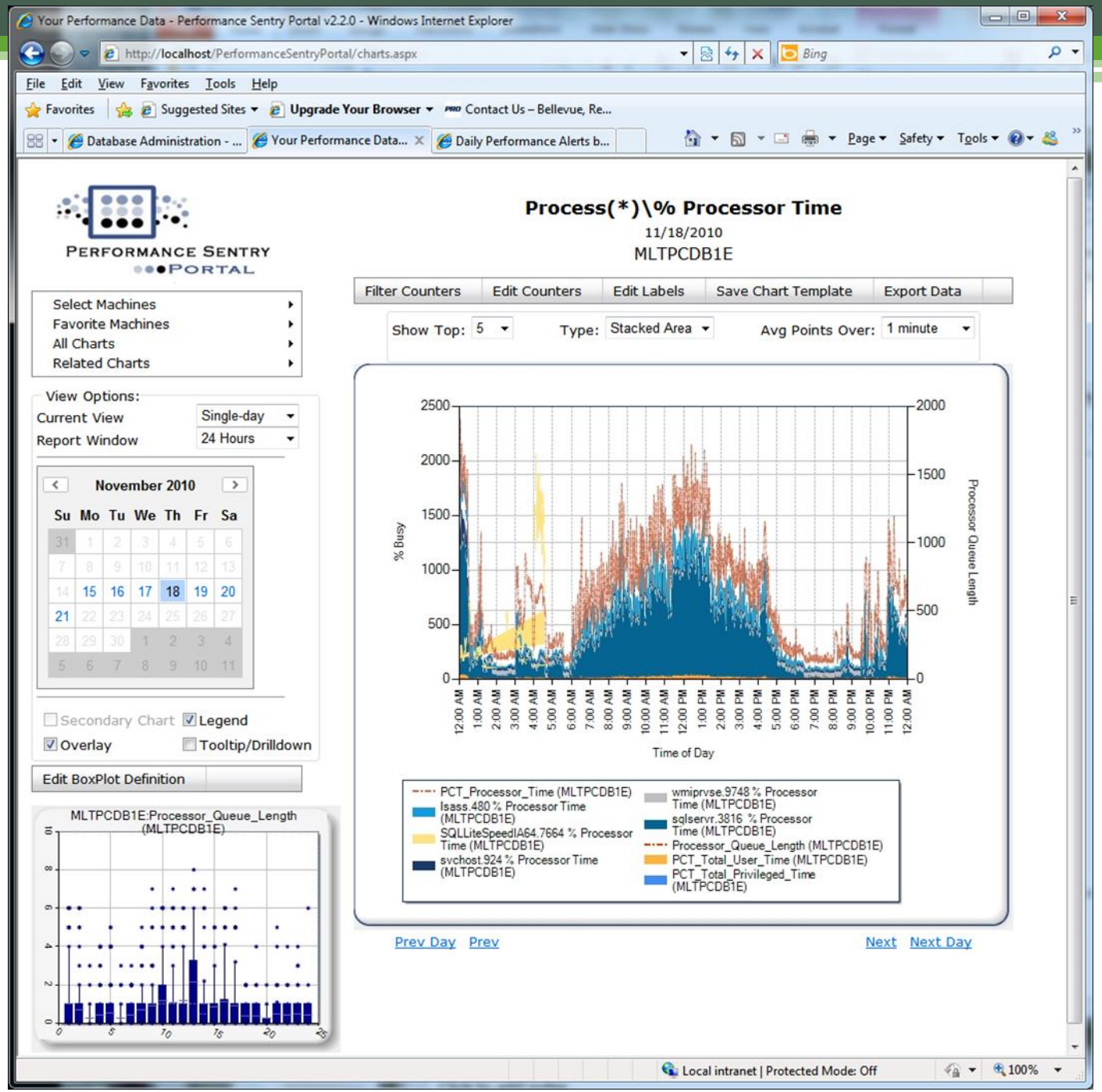
- Themes:
  - Web application programming models and fashion change faster than tools can adapt
    - AJAX
      - e.g., Auto-complete in Google Search
      - achieved using client-side JavaScript & Asynchronous Http web service Requests
    - High Availability and Scalability using n-tiered architectures
      - typically, a Presentation Layer, Business Objects layer, and a Data Access Layer
    - the latest JavaScript frameworks
    - HTML5
    - HTTP/2
    - Effective performance tools are usually one or two releases behind emerging technology

Scalability models

- Fundamental concept in software performance engineering (SPE)
- *namely,* $f(x)$,
  such that $f(x)$ reliably predicts Response time or Thruput.
- Factors can be
  - linear (m + n…)
  - multiplicative (m * n)
  - exponential (m^n)

Why is my web app running slowly?

Case Study:


Your Performance Data - Performance Sentry Portal v2.2.0 - Windows Internet Explorer
http://localhost/PerformanceSentryPortal/charts.aspx
File Edit View Favorites Tools Help
Favorites Suggested Sites Upgrade Your Browser Contact Us – Bellevue, Re...
Database Administration - ... Your Performance Data... Daily Performance Alerts b...
Page Safety Tools
PERFORMANCE SENTRY PORTAL
Select Machines
Favorite Machines
All Charts
Related Charts
View Options:
Current View Single-day
Report Window 24 Hours
November 2010
Su Mo Tu We Th Fr Sa
Secondary Chart Legend
Overlay Tooltip/Drilldown
Edit BoxPlot Definition
MLTPCDB1E:Processor_Queue_Length (MLTPCDB1E)
Process(*)\% Processor Time
11/18/2010
MLTPCDB1E
Filter Counters Edit Counters Edit Labels Save Chart Template Export Data
Show Top: 5 Type: Stacked Area Avg Points Over: 1 minute
% Busy
Processor Queue Length
Time of Day
PCT_Processor_Time (MLTPCDB1E)
lsass.480 % Processor Time (MLTPCDB1E)
SQLLiteSpeedIA64.7664 % Processor Time (MLTPCDB1E)
svchost.924 % Processor Time (MLTPCDB1E)
wmiprvse.9748 % Processor Time (MLTPCDB1E)
sqlservr.3816 % Processor Time (MLTPCDB1E)
Processor_Queue_Length (MLTPCDB1E)
PCT_Total_User_Time (MLTPCDB1E)
PCT_Total_Privileged_Time (MLTPCDB1E)
Prev Day Prev Next Next Day
Local intranet | Protected Mode: Off
100%

Why is this web app running slowly?

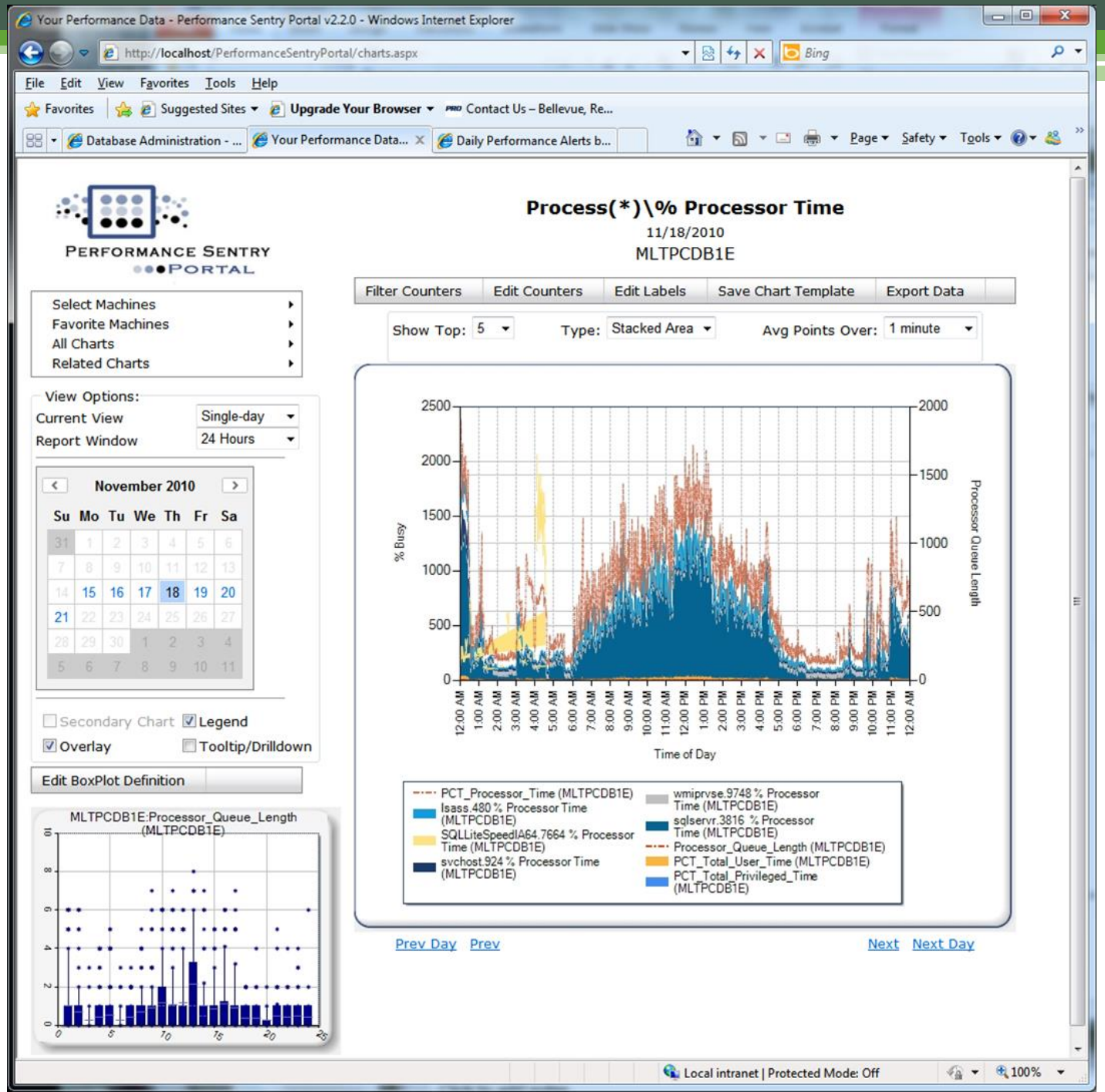
- Database Query with graphical reporting
  - MS SQL Server DB containing Windows performance counter data, gathered once per minute
    - # of machines scales to 10,000+ machines
- Developed using ASP.NET (Server-side controls) + JavaScript

```
<asp:DropDownList ID="ddlCurrentView" runat="server" Width="100">
      <asp:ListItem Text="Single-day" Value="Single-day"></asp:ListItem>
      <asp:ListItem Text="Multi-day" Value="Multi-day"></asp:ListItem>
 </asp:DropDownList>
```

Note: `runat="server"` with `autopostback="true"` requires Http GET Requests to be sent to IIS/ASP.NET for each UI interaction

Why is this web app running slowly?

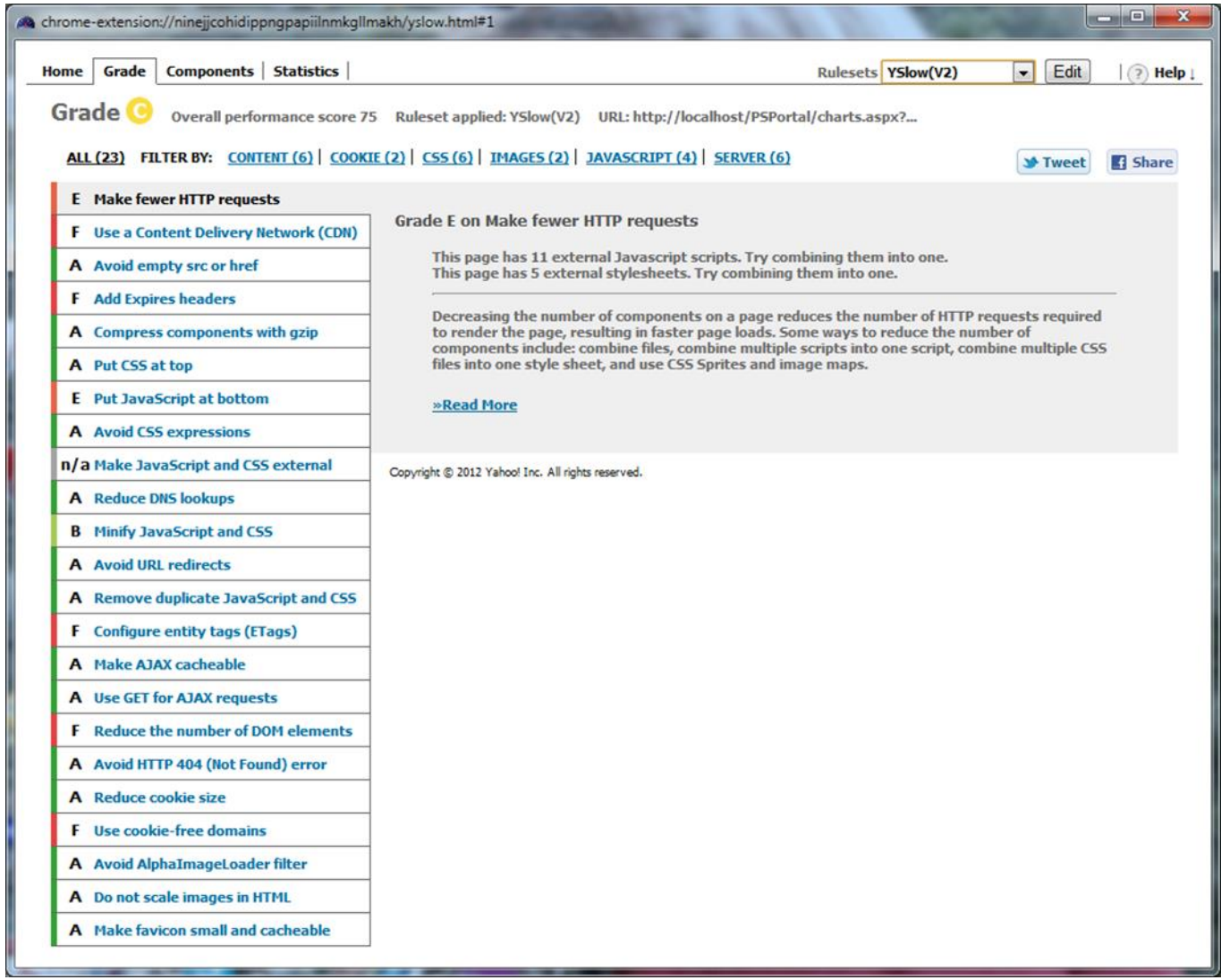
- Developed using ASP.NET (Server-side controls)
  - Multiple tiers: presentation/business objects/data layer
  - Uses the Model-View-Controller (MVC) pattern
  - Key elements of the web Page
    - data-rich Charting component (.NET Chart, based on the Dundas component)
    - Chart definition used to generate the DB Query and the results are mapped to a Chart instance
      - Library of Chart templates
      - Machine selection
      - Date/Time selection


Your Performance Data - Performance Sentry Portal v2.2.0 - Windows Internet Explorer
http://localhost/PerformanceSentryPortal/charts.aspx
File Edit View Favorites Tools Help
Favorites Suggested Sites Upgrade Your Browser Contact Us – Bellevue, Re...
Database Administration - ... Your Performance Data... Daily Performance Alerts b...
Page Safety Tools
PERFORMANCE SENTRY PORTAL
Select Machines
Favorite Machines
All Charts
Related Charts
View Options:
Current View Single-day
Report Window 24 Hours
November 2010
Su Mo Tu We Th Fr Sa
Secondary Chart Legend
Overlay Tooltip/Drilldown
Edit BoxPlot Definition
MLTPCDB1E:Processor_Queue_Length (MLTPCDB1E)
Process(*)\% Processor Time
11/18/2010
MLTPCDB1E
Filter Counters Edit Counters Edit Labels Save Chart Template Export Data
Show Top: 5 Type: Stacked Area Avg Points Over: 1 minute
% Busy
Processor Queue Length
Time of Day
PCT_Processor_Time (MLTPCDB1E)
lsass.480 % Processor Time (MLTPCDB1E)
SQLLiteSpeedIA64.7664 % Processor Time (MLTPCDB1E)
svchost.924 % Processor Time (MLTPCDB1E)
wmiprvse.9748 % Processor Time (MLTPCDB1E)
sqlservr.3816 % Processor Time (MLTPCDB1E)
Processor_Queue_Length (MLTPCDB1E)
PCT_Total_User_Time (MLTPCDB1E)
PCT_Total_Privileged_Time (MLTPCDB1E)
Prev Day Prev Next Next Day
Local intranet | Protected Mode: Off
100%

Why is this web app running slowly?

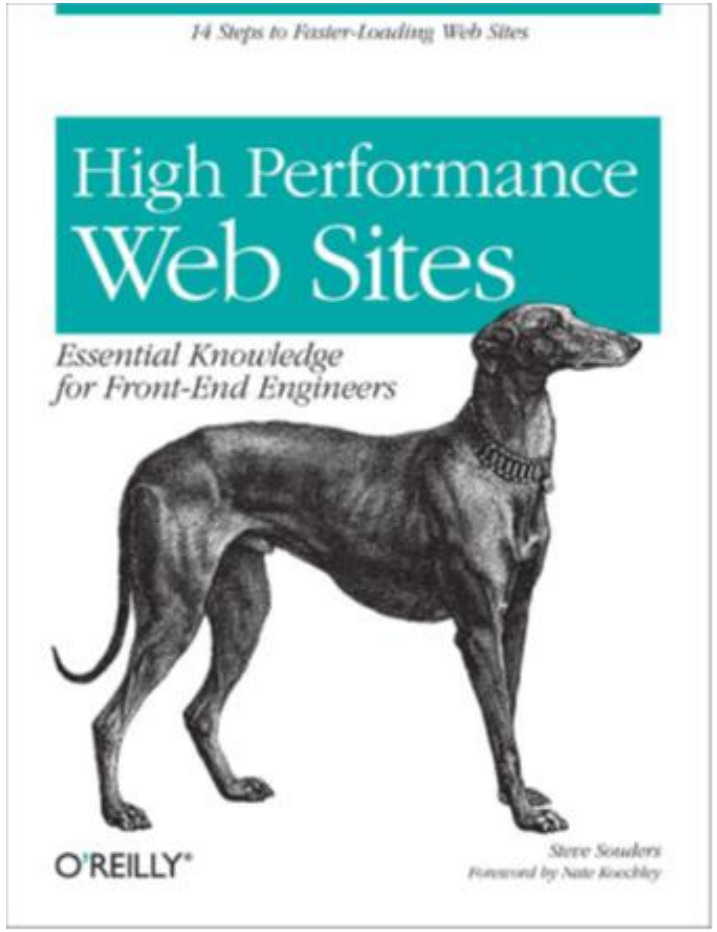
Ask an expert:

YSlow



What is YSlow?

- Based on the influential work of Steve Souders*
  - originally at Yahoo
  - since migrated to Google
    - Google Chrome extension
  - Rule-based
  - Influenced:
    - Chrome PageSpeed Insights
    - IE Developer Tools
    - Fiddler
    - Glimpse
    - etc.

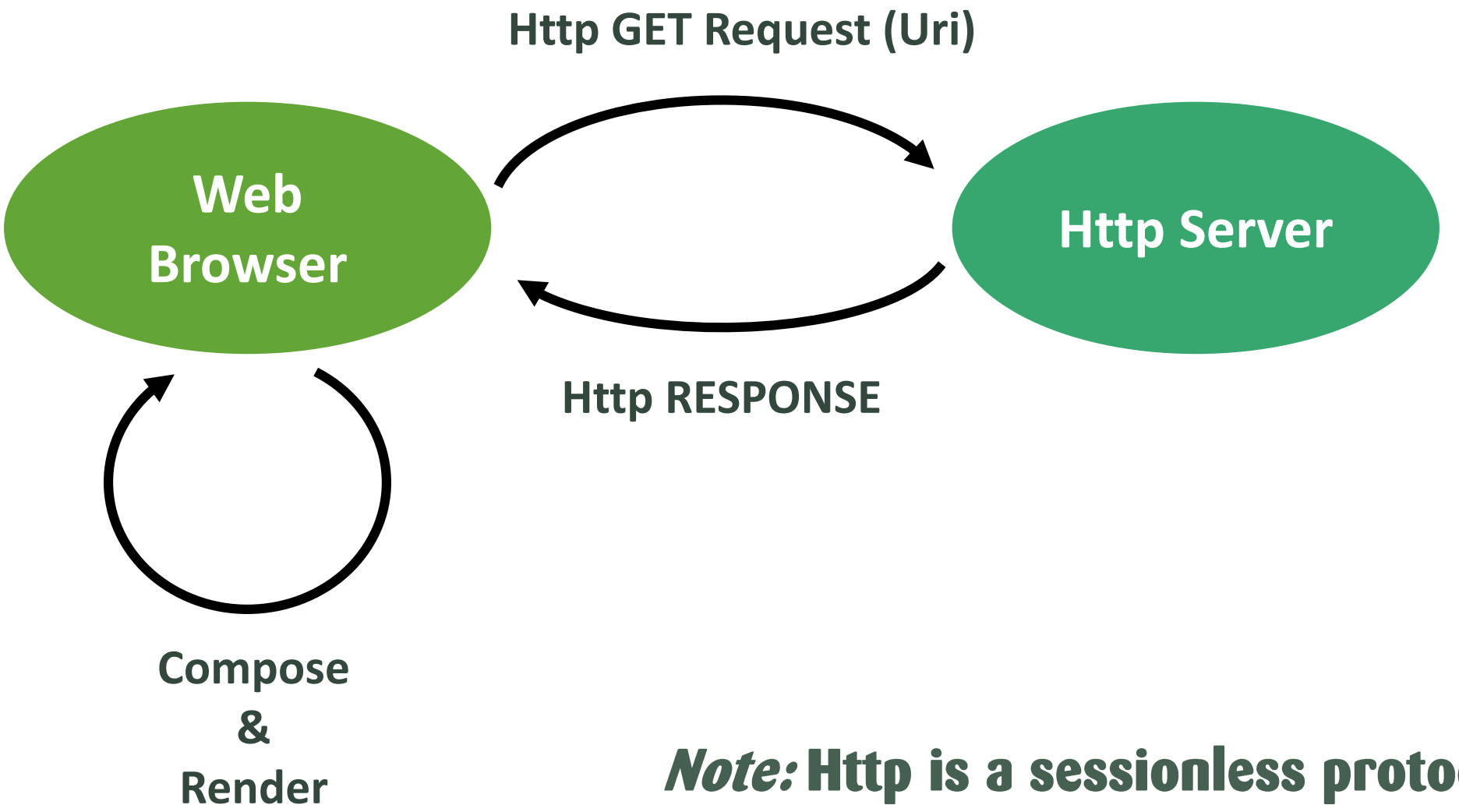


* *High Performance Web Sites*, O'Reilly Media, 2007

What is YSlow?

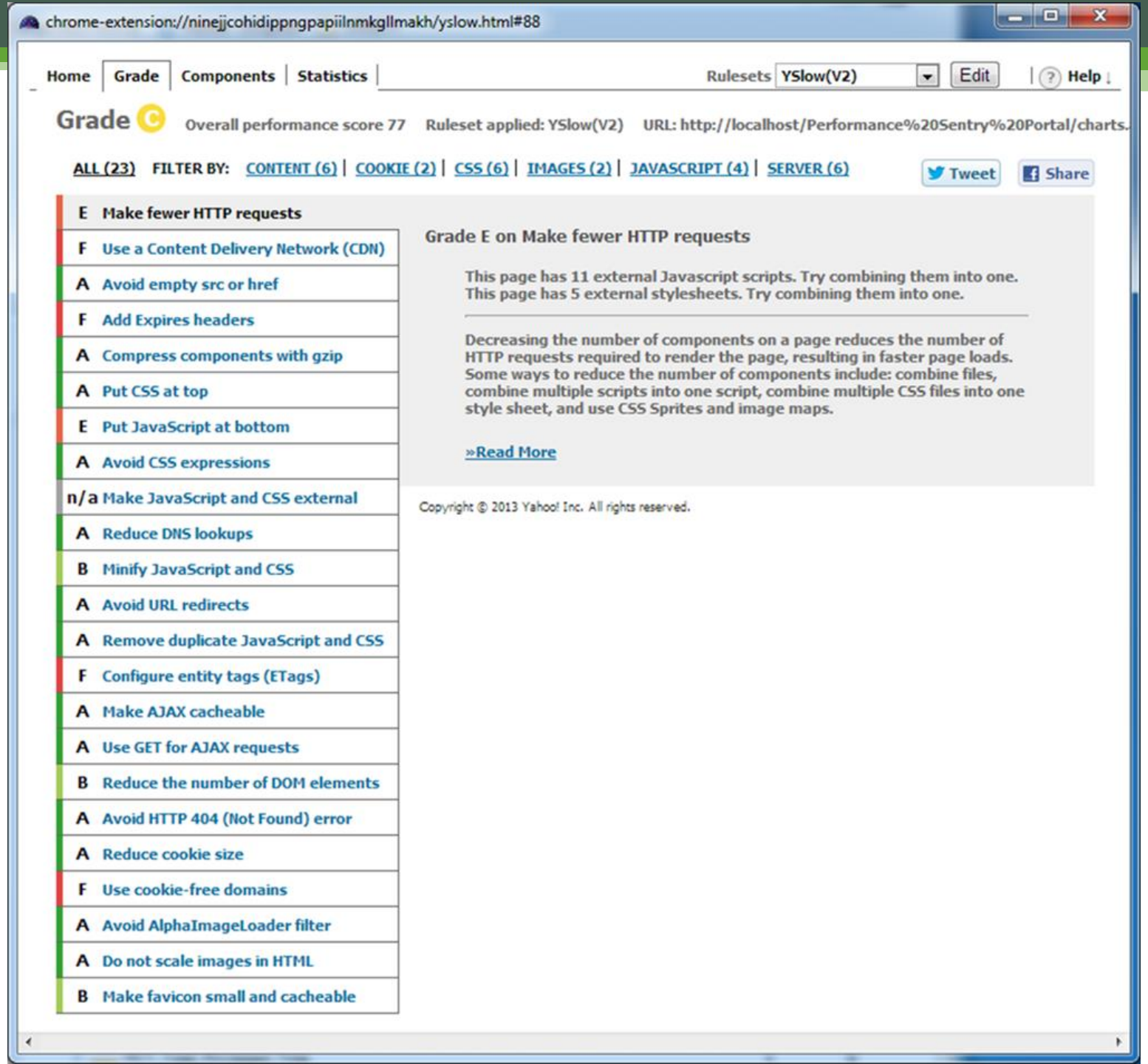
- Optimize for **Page Load Time**
  - Makes specific recommendations on how to improve Page Load Time for a specific web page
    - Document Object Model (DOM) for rendering a web page
    - Request.Start ⇨ DOM.Complete
    - Standardization effort to wire performance timing data to the DOM & create a consistent way to access it
      - Navigation Time, Performance Timing & a High Resolution Clock

Page Load Time

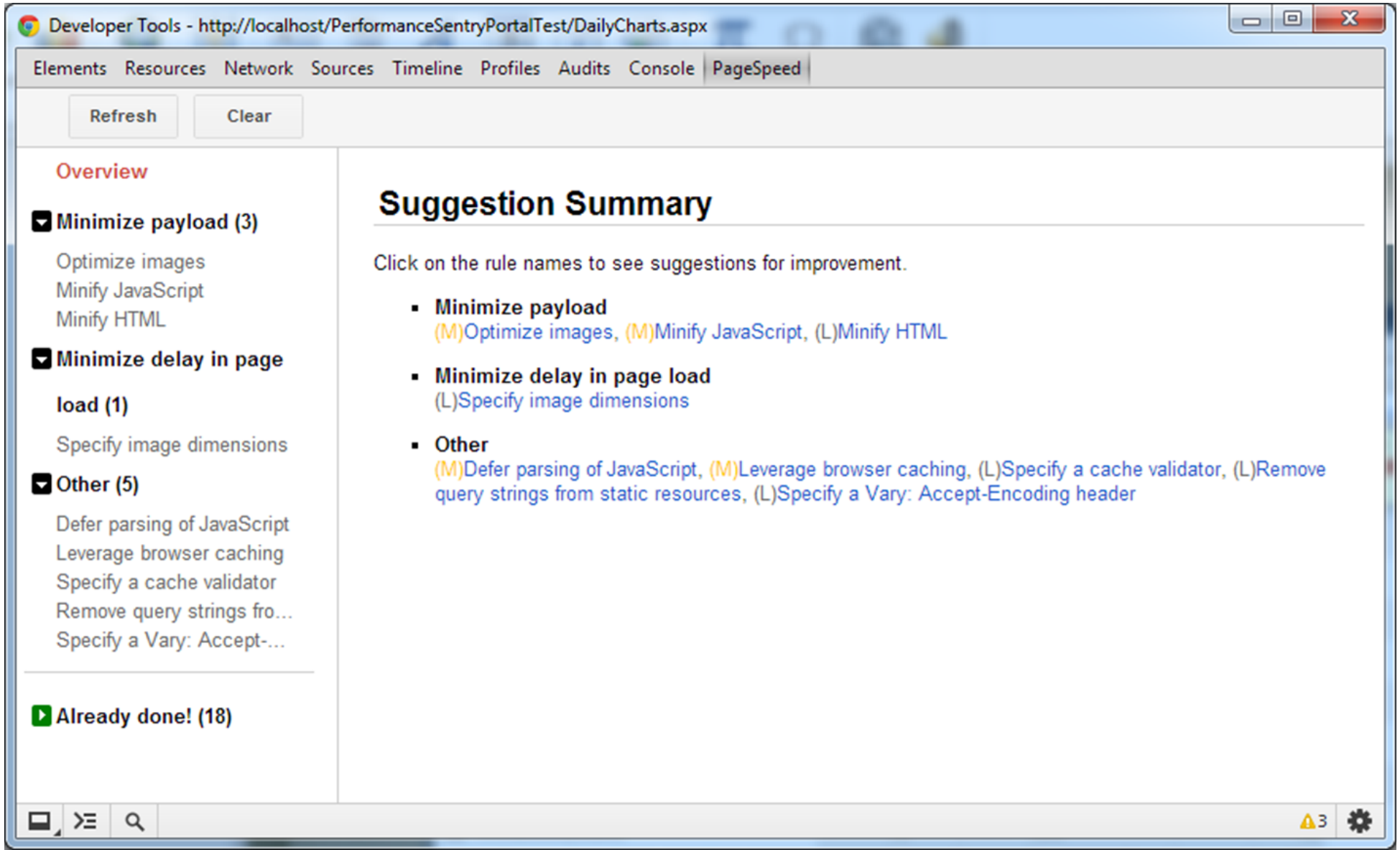


Note: **Http is a sessionless protocol**

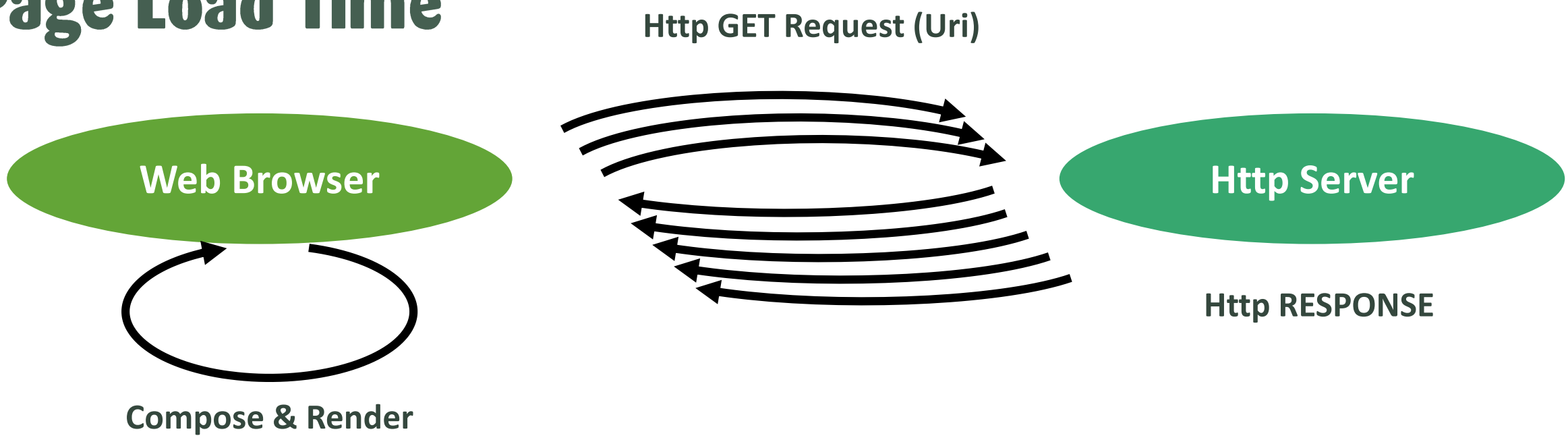
YSlow Chrome extension



PageSpeed Chrome extension



Page Load Time

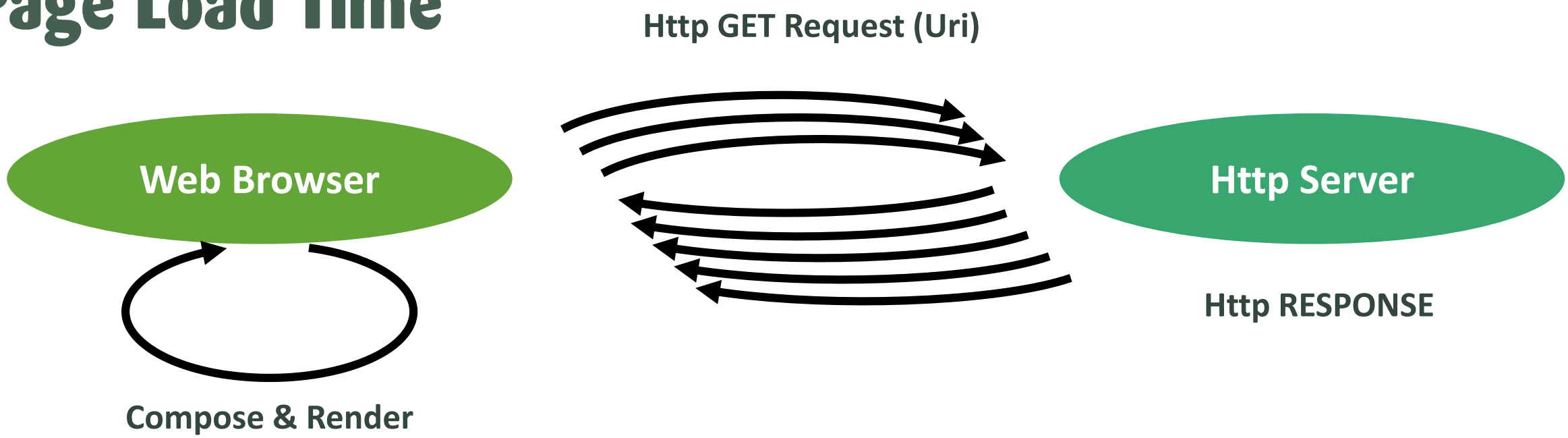


Response message often contains embedded references to additional resources needed to render the Page

e.g.,

- **image files**
- **style sheets**
- **JavaScript files**

Page Load Time



- HTTP interacts with the underlying TCP/IP networking protocols
  - HTTP Response messages > *Ethernet packet size* (~ 1500 bytes) require multiple IP packets
  - With large cookies and a large number of parms, GET Request messages can also exceed the Ethernet packet size

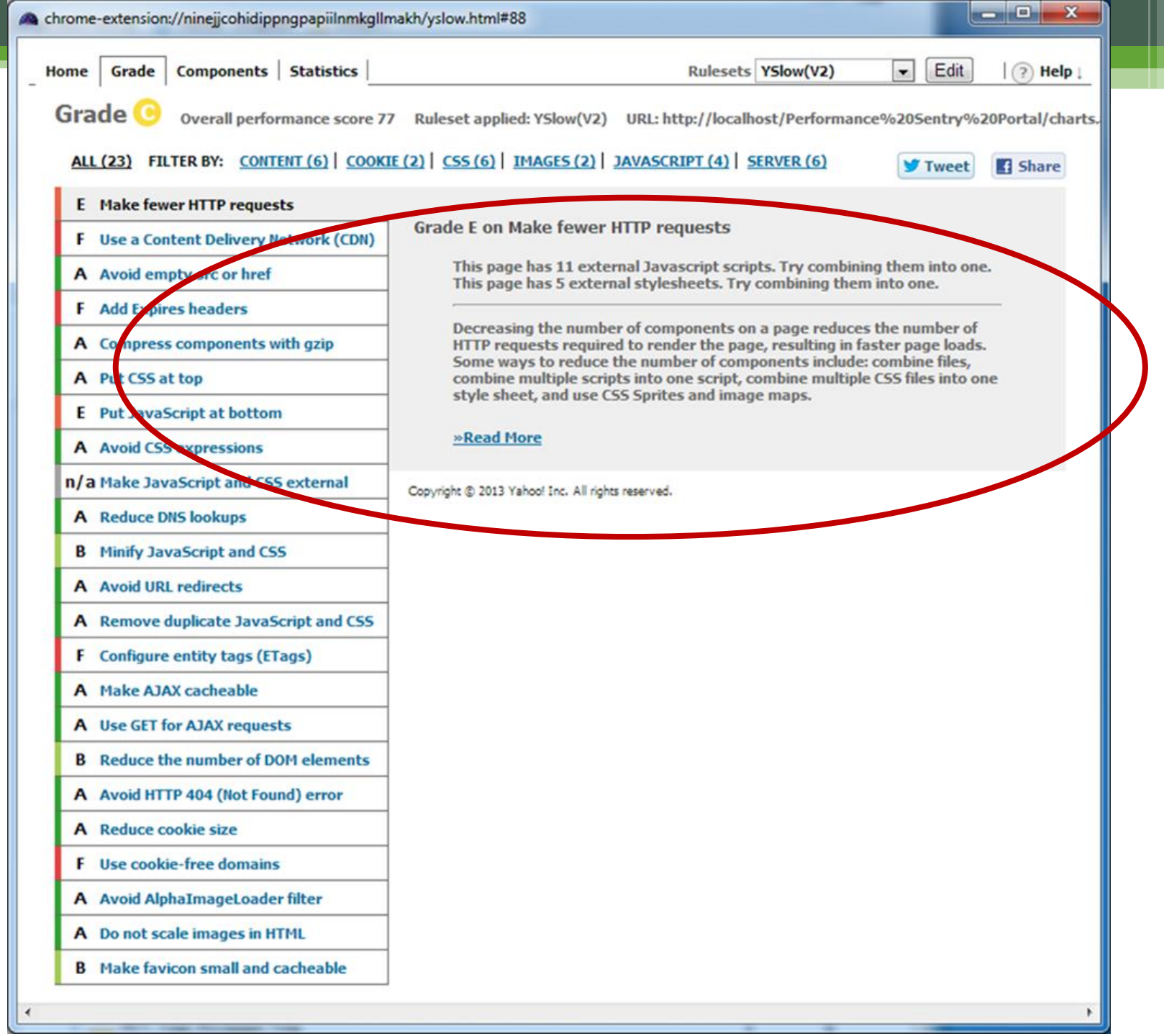
Page Load Time

- YSlow scalability model:

$$\text{Render Time} \approx \text{RoundTrips} * \text{RTT}$$

$$\text{RoundTrips} = \sum_{i=1}^{n} \frac{httpObjectSize_i}{packetsize}$$

Ask YSlow


chrome-extension://ninejjcohidippngpapiilnmkgllmakh/yslow.html#88
Home
Grade
Components
Statistics
Rulesets
YSlow(V2)
Edit
Help
Grade C
Overall performance score 77
Ruleset applied: YSlow(V2)
URL: http://localhost/Performance%20Sentry%20Portal/charts.
ALL (23)
FILTER BY:
CONTENT (6)
COOKIE (2)
CSS (6)
IMAGES (2)
JAVASCRIPT (4)
SERVER (6)
Tweet
Share
E Make fewer HTTP requests
F Use a Content Delivery Network (CDN)
A Avoid empty src or href
F Add Expires headers
A Compress components with gzip
A Put CSS at top
E Put JavaScript at bottom
A Avoid CSS expressions
n/a Make JavaScript and CSS external
A Reduce DNS lookups
B Minify JavaScript and CSS
A Avoid URL redirects
A Remove duplicate JavaScript and CSS
F Configure entity tags (ETags)
A Make AJAX cacheable
A Use GET for AJAX requests
B Reduce the number of DOM elements
A Avoid HTTP 404 (Not Found) error
A Reduce cookie size
F Use cookie-free domains
A Avoid AlphaImageLoader filter
A Do not scale images in HTML
B Make favicon small and cacheable
Grade E on Make fewer HTTP requests
This page has 11 external Javascript scripts. Try combining them into one.
This page has 5 external stylesheets. Try combining them into one.
Decreasing the number of components on a page reduces the number of HTTP requests required to render the page, resulting in faster page loads. Some ways to reduce the number of components include: combine files, combine multiple scripts into one script, combine multiple CSS files into one style sheet, and use CSS Sprites and image maps.
»Read More
Copyright © 2013 Yahoo! Inc. All rights reserved.

YSlow scalability model:

$$Browser\ Render\ Time \approx RoundTrips * RTT$$

- Web Browser performs ***page composition*** using the Document Object Model (DOM)
  - YSlow Rule: Make fewer HTTP requests
  - YSlow Rule: Improve cache effectiveness
  - YSlow Rule: Reduce the number of DOM elements
  - YSlow Rule: Compress the objects the page does need to load

- Tuning is a process that attempts to drive

RoundTrips ⇨ 0
RoundTripTime ⇨ 0

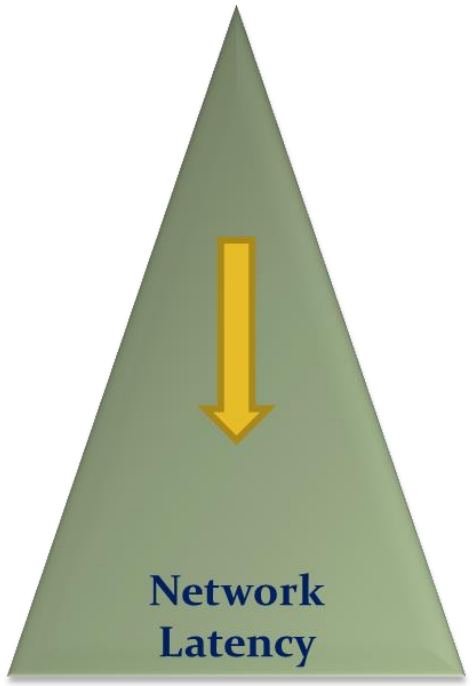
YSlow scalability model:

$$Browser\ Render\ Time \approx RoundTrips * RTT$$

- Round Trip Time (RTT), rather than network bandwidth, is determinative for internetworking across long distances

- **Strategies to reduce RTT**
  - TCP session-oriented behavior requires that each packet sent must be ACKed by the Receiver
  - YSlow Rule: Use a Content Delivery Network (e.g., Akamai)

YSlow scalability model



- Increase cache effectiveness
  - Static content readily cached:
    - Images files, Style sheets, Scripts
  - Multiple caching services
    - Browser resident (e.g., Temporary Internet Files)
    - Content Delivery network
    - IIS
  - Caching content generated dynamically by ASP.NET programs is more difficult!
    - ASP.NET
      - Cache API
      - Output Cache

YSlow scalability model complications

- Web Browsers create multiple TCP sessions to download referenced objects in parallel
  - YSlow Rule: take advantage of parallel sessions by loading scripts last
    - Accurate rendering of the DOM requires that downloading a JavaScript file **blocks** parallel downloads
      - Script may add elements to the DOM dynamically, call other scripts or reference additional resources
    - Browser assumes DOM rendering can only resume ***after*** the Javascript code executes

```
Page Load Time =   Browser Render Time +
                   Script execution Time +
                   (RoundTrips * RTT)/Sessions
```

YSlow scalability model complications

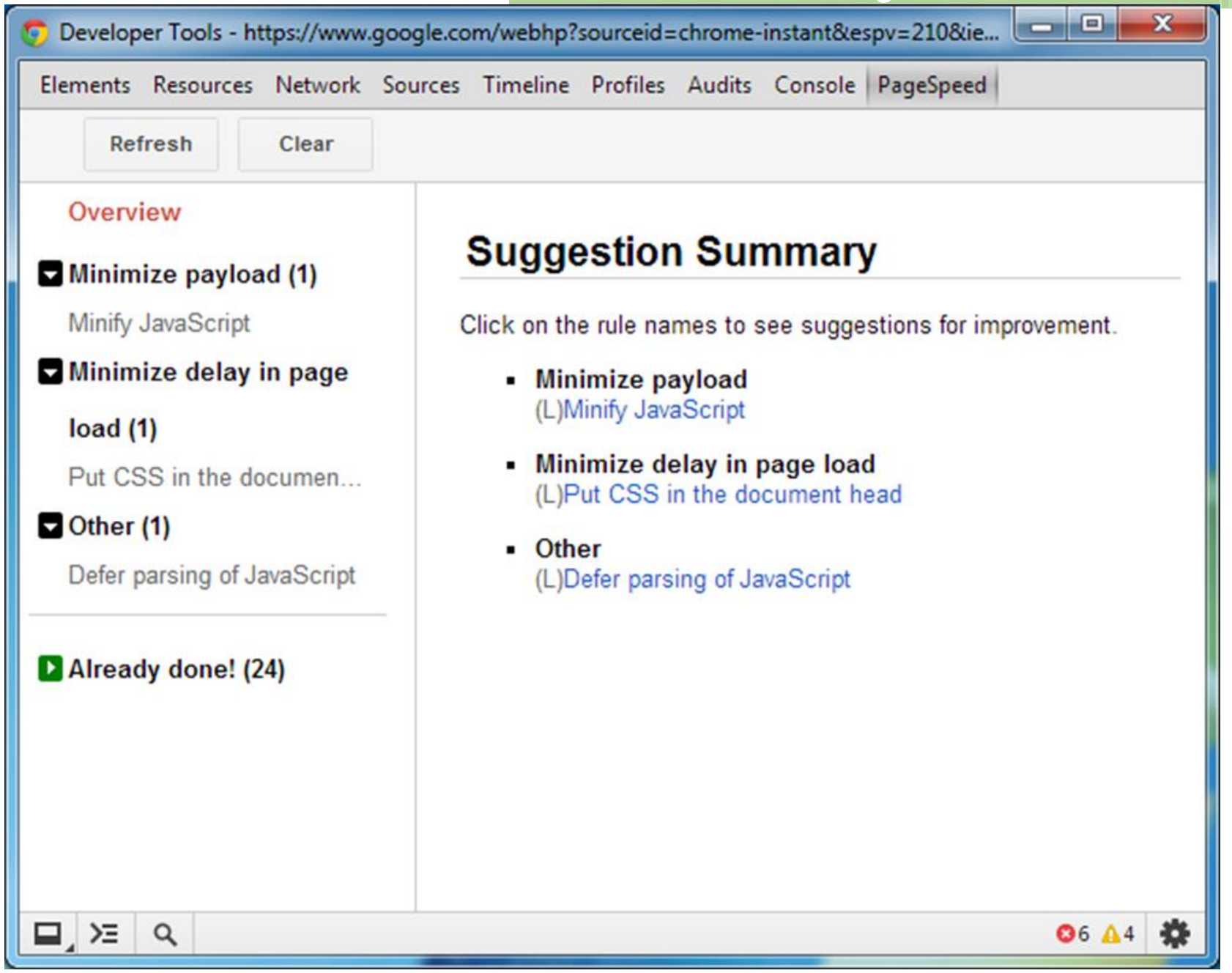
- RTT may vary across Requests (using different TCP sessions)
  - Objects can be geographically dispersed
    - Local cluster, remote, cloud
    - e.g., referencing 3rd-party, advertising services
  - TCP adaptive window scaling and other network **congestion avoidance** strategies
  - YSlow never actually measures RTT, but other related tools can

- Ignores variability in the execution time of client and server-side code
  - e.g., `sort()` a large list of elements, or
  - a hi-res visualization component that scales **nonlinearly** with the size of the result set
- Compression recommendations can be at odds with code maintainability Best Practices

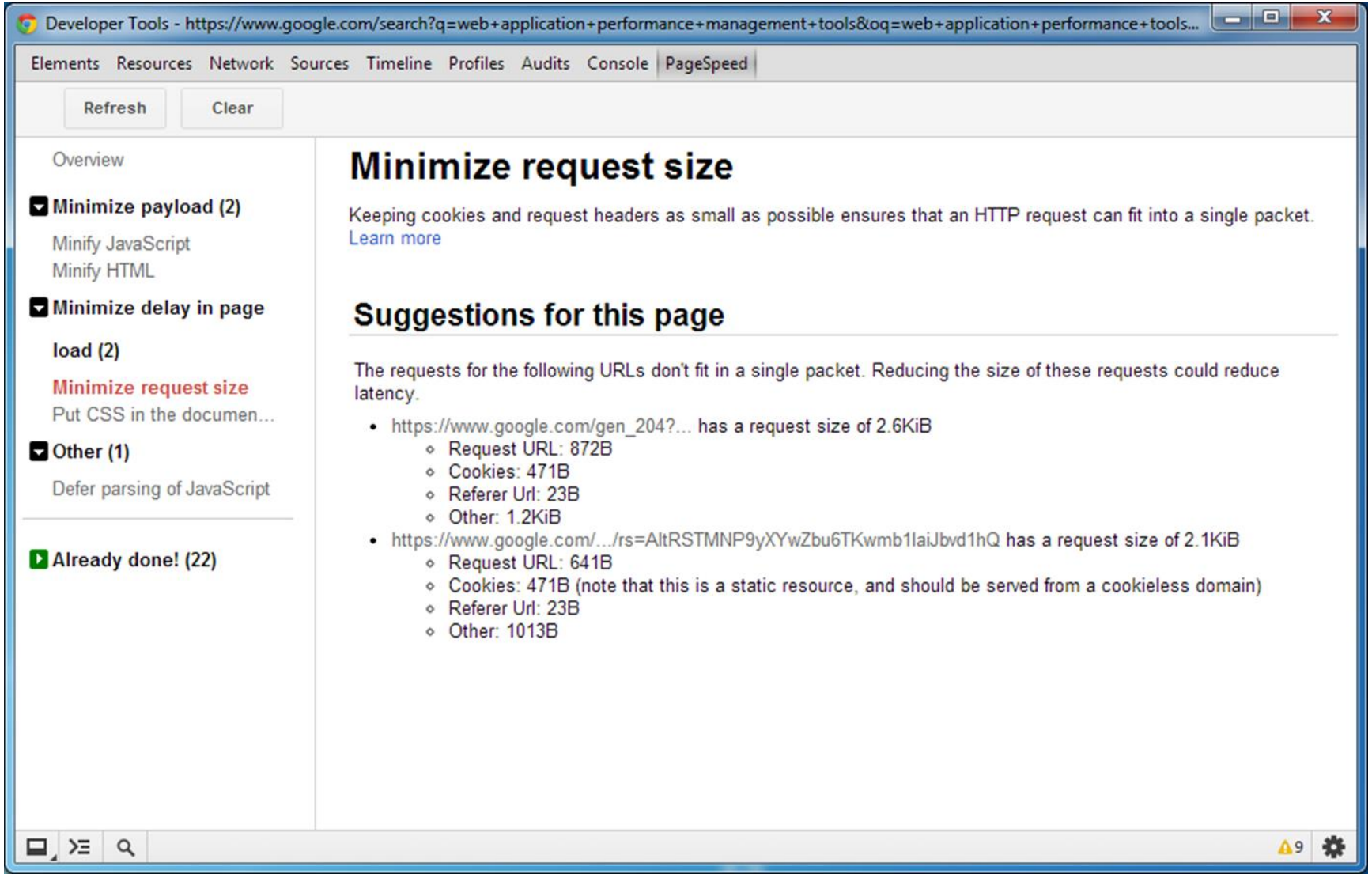
YSlow scalability model

- Despite the many complications, the YSlow scalability model has proved very influential
  - Browser Page Load Time is an **end-to-end** measurement of service time, which is apt to be correlated with customer satisfaction
    - see, for example, "37 Lessons I've Learned on the Performance Front Lines"
  - or, even more important for e-commerce sites, fulfillment rates
    - see, e.g., https://searchengineland.com/how-not-to-get-lost-in-the-performance-oriented-web-303071
  - Google, for example, practices what it preaches
    - Uses very simple Response messages (although they have grown more complicated due to competition from Bing)

Google Search landing page



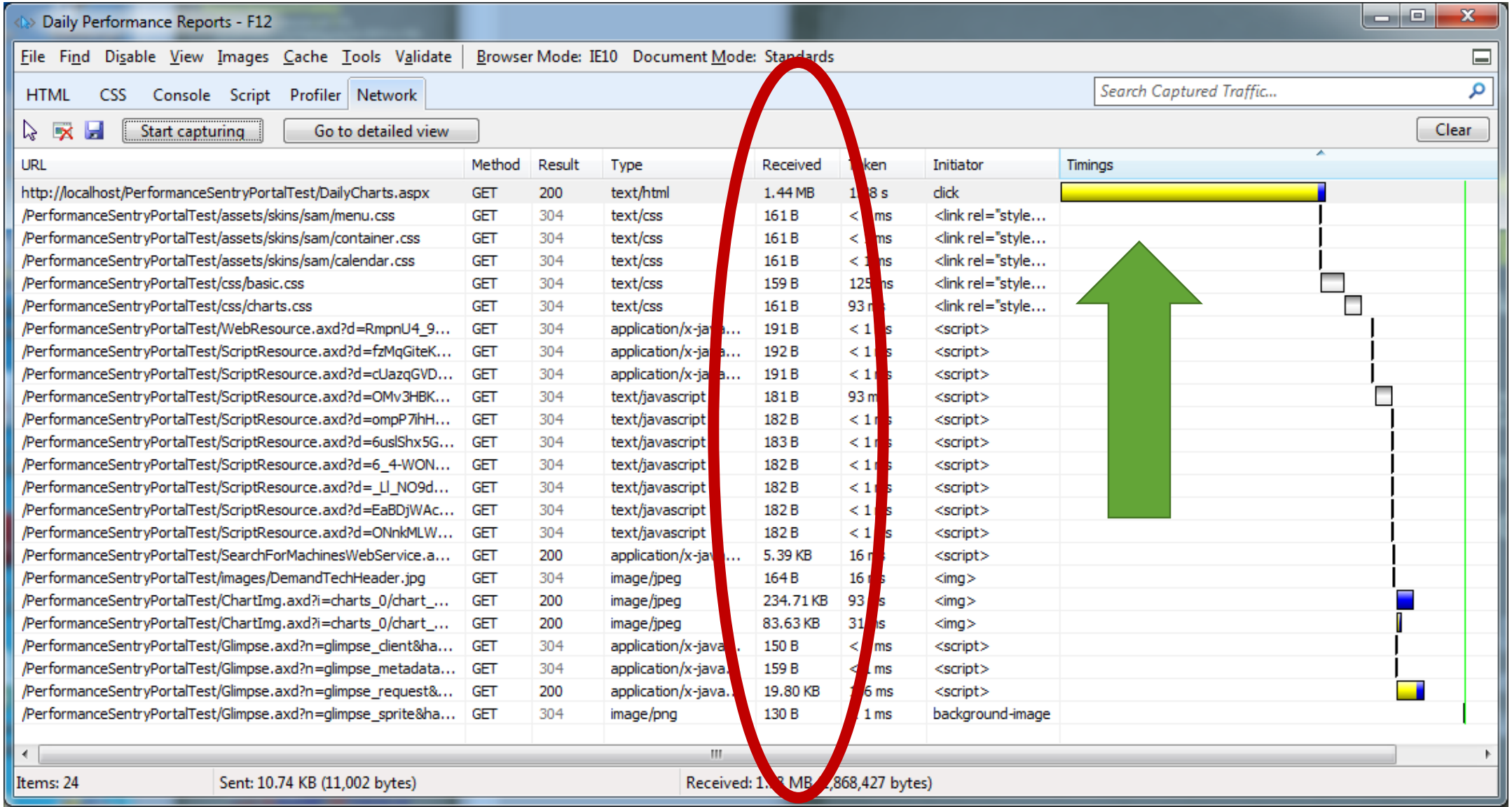
Google Search landing page



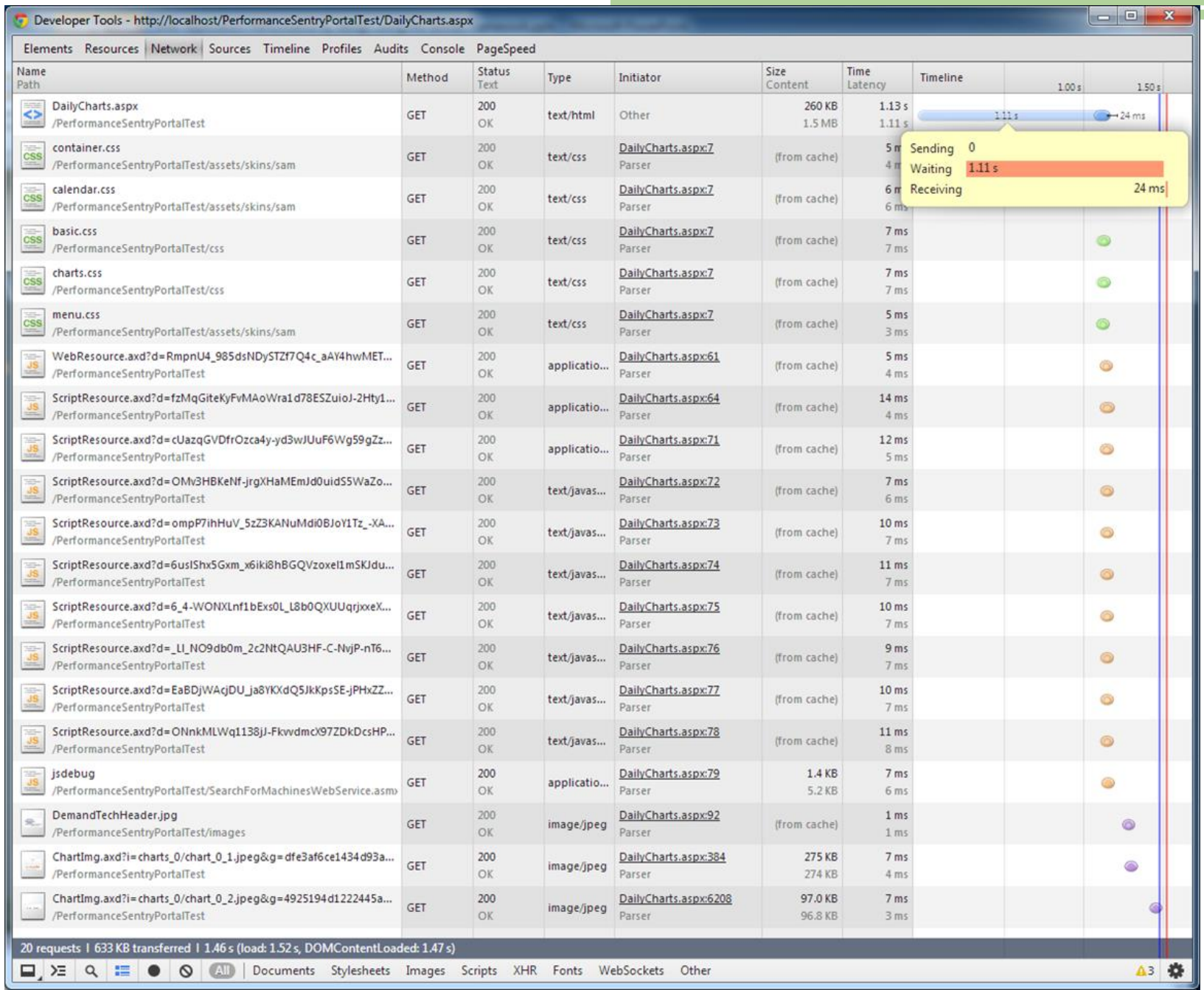
YSlow scalability model

- Despite the many complications, the YSlow scalability model has proved very influential
  - Sparked development of web page performance tools that do measure Page Load Times
    - Browser-based Developer Tools
    - Waterfall (timeline) graph
    - e.g., Chrome, Edge “Network” View
    - Note: simple analysis of network traffic cannot account for any JavaScript execution time delays

YSlow Scalability model assessment



from the Case Study…



Architecting High-Performance web sites

- Scale: multiple-tiers, clustering
- Geographical diversity
- take advantage of parallel TCP sessions
  - sharding monolithic content; HTTP/2 vs. HTTP/1
- Cache effectiveness
  - Optimizing for browser-based cache
    - what about Mobile?
  - edge networks (CDNs)
- Connected native applications vs. browser-based

YSlow scalability model

- Despite its many complications, the YSlow scalability model has proved very influential
  - Evolved from a "Rules and Recipes" analysis of static DOM elements approach to a dynamic, measurement approach
    - Standardized instrumentation so developers could access the PLT measurements reliably across Browsers
    - Creation of standard DOM performance objects that are accessible using JavaScript
  - Once actual web application performance data becomes readily available from the web browser, web developers face a problem of how to get that data back to my data center for optimization & capacity planning?

YSlow scalability model

- Despite its many complications, the YSlow scalability model has proved very influential
- Evolved from a "Rules and Recipes" analysis of static DOM elements approach to a dynamic, measurement-oriented approach
- But "Rules and Recipes" still has its adherents:
  - see "The Low Hanging Fruit of Web Performance" for a recent example

YSlow scalability model

- Addressing the limitations of the YSlow "Rules and Recipes" approach was the impetus behind adding standardized instrumentation to the browsers so developers could access the PLT measurements
  - Creation of standard DOM performance objects that are accessible using JavaScript
  - Once actual web application performance data becomes readily available from the web browsers, developers then face a problem of how to get that data back to my data center for optimization & capacity planning studies?

Measuring Web application response time

- Applying the Rules and Recipes approach without recourse to measurements is problematic
  - How much improvement from implementing the Rules to
    - cache all static content?
    - compress all images files > packet size?
    - defer all JavaScript loads?
- But simple measurement approaches have their own limitations:
  - Measure the impact of caching at the client & using a CDN?
  - RTTs vary geographically!

Web Application measurement & tuning

- Web Pages contain:
  - Html
  - Hyperlinks
  - CSS
  - JavaScript
  - Images
  - Audio
  - Video
  - 3rd party Ad servers
    - Banner ads
  - etc.
- Browser ⇨ composition, layout
- Page Load time can be improved using parallelism
  - loading a JavaScript file requires serialization because the script can modify the DOM
    - see Souders
  - AJAX: foreground-background
  - web browser support for multiple TCP sessions that execute in parallel
  - HTTP/2

Web Application measurement & tuning

- consider the web app Scalability model:

$$\mathbf{PDT} \approx \text{client render time} + \text{script execution time} + \text{RTT} * \left(\sum_{i=1}^{n} \frac{httpObjectSize_i}{packetsize} \right) / \text{sessions}$$

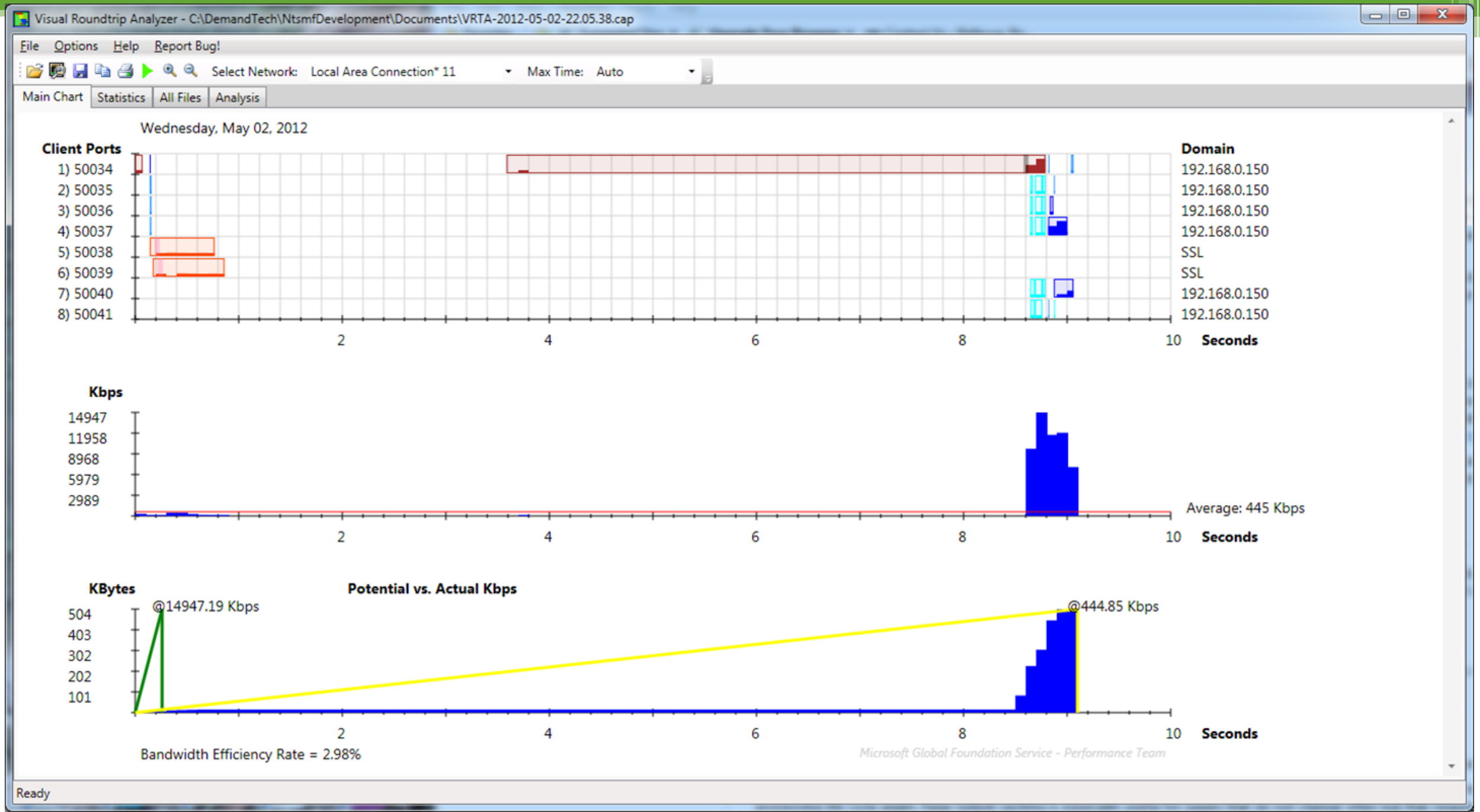
- Measuring Page Load Time
  - Time to First Byte
  - Time to Ready (for User Input/Interaction)
  - Measurement tools
    - Network Monitor capture tools
    - browser Developer tools
    - RUM & other external tools

Web Application measurement & tuning

- A survey of measurement tools
  - Network Monitor capture tools
  - 3rd party ETE monitoring tools
  - browser Developer tools
  - external developer tools
  - RUM

Web Application measurement tools

- Network packet trace capture tools
  - Extensions to existing network monitors
  - Stitch together all network traffic associated with a single web request into a coherent set of measurements
    - Network timeline (or waterfall) view
    - identify multiple sessions associated with a single, originating IP address
  - Limitations:
    - data is captured from inside the web server network
    - large amount of packets captured


Visual Roundtrip Analyzer - C:\DemandTech\NtsmfDevelopment\Documents\VRTA-2012-05-02-22.05.38.cap
File
Options
Help
Report Bug!
Select Network:
Local Area Connection* 11
Max Time:
Auto
Main Chart
Statistics
All Files
Analysis
Wednesday, May 02, 2012
Client Ports
1) 50034
2) 50035
3) 50036
4) 50037
5) 50038
6) 50039
7) 50040
8) 50041
Domain
192.168.0.150
192.168.0.150
192.168.0.150
192.168.0.150
SSL
SSL
192.168.0.150
192.168.0.150
2
4
6
8
10
Seconds
Kbps
14947
11958
8968
5979
2989
Average: 445 Kbps
2
4
6
8
10
Seconds
KBytes
Potential vs. Actual Kbps
@14947.19 Kbps
@444.85 Kbps
504
403
302
202
101
2
4
6
8
10
Seconds
Bandwidth Efficiency Rate = 2.98%
Microsoft Global Foundation Service - Performance Team
Ready

Web Application measurement tools

- End-to-End (ETE) real-time monitoring tools
  - 3rd party tools that generate and analyze synthetic Requests
    - monitor availability and performance
    - using geographically distributed client machines
  - see, e.g., Dynatrace Digital Experience Insights product
    - formerly Keynote Systems

Web Application measurement & tuning

- consider an enhanced YSlow web app Scalability model:

$$\text{PDT} \approx \text{client render time} + \text{script execution time} + \text{RTT} * \left(\sum_{i=1}^{n} \frac{httpObjectSize_i}{packetsize} \right) / \text{sessions}$$

- Measuring Page Load Time
  - Time to First Byte
  - Time to Ready (for User Input/Interaction)
  - Measurement tools
    - Network Monitor capture tools
    - browser Developer tools
    - RUM & other external tools

Web Application measurement & tuning

- consider an enhanced YSlow web app Scalability model:

$$\mathbf{PDT} \approx \text{client render time} + \text{script execution time} + \text{RTT} * \left(\sum_{i=1}^{n} \frac{httpObjectSize_i}{packetsize} \right) / \text{sessions}$$

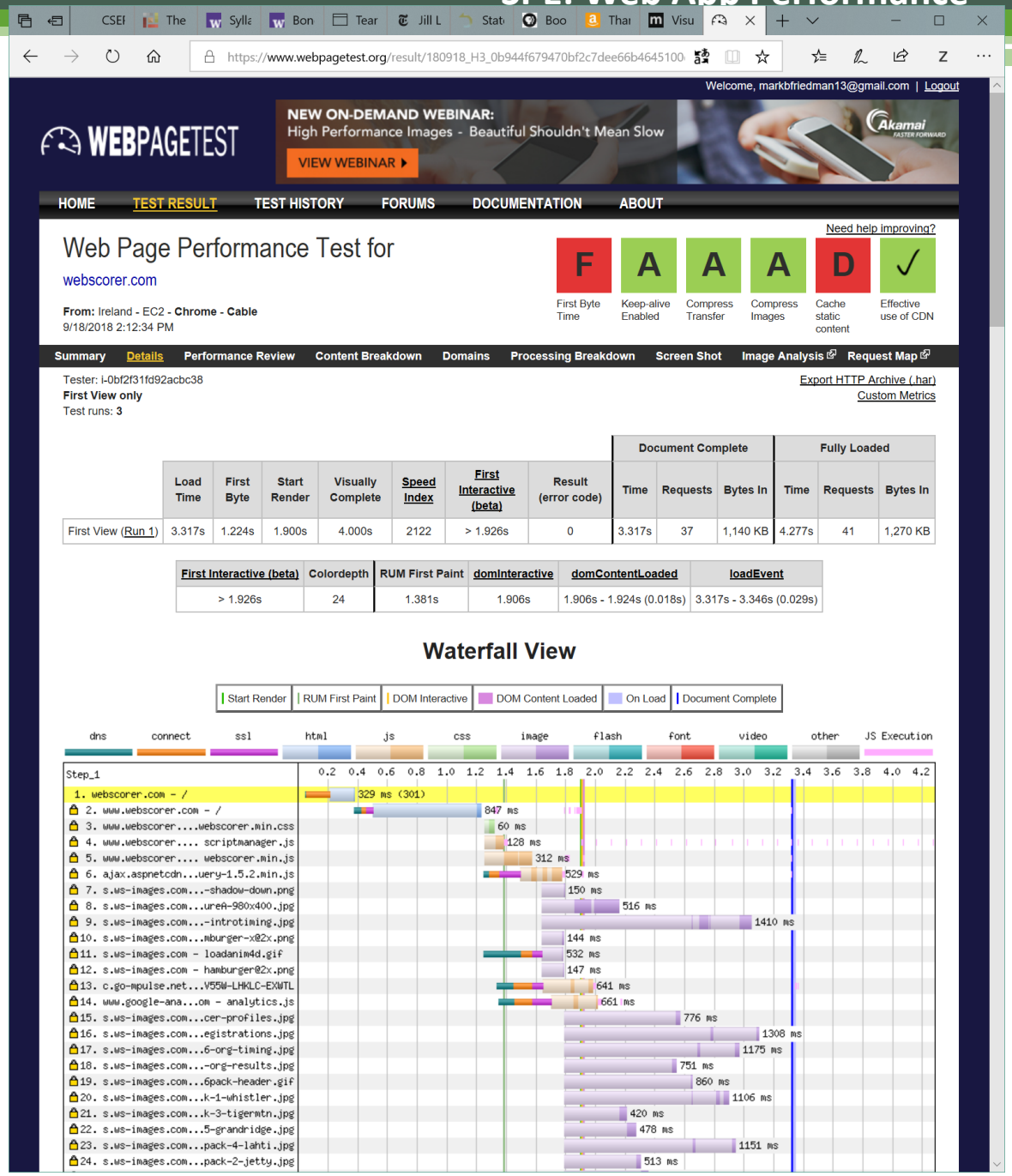
- Rule violations that do not assess how much improvement to expect from following them are not very useful
- Measuring Page Load Time complications
  - RTTs are dependent on where the client is located
  - client execution time depends on the capability of the client machine

Web Application measurement tools

- web browser Developer tools
  - Chrome
  - Edge
  - etc.
- related developer tools
  - Fiddler
  - Glimpse
- external developer tools
  - WebPageTest : a direct outgrowth from YSlow

WebPageTest

demo:



Real User Measurements (RUM)

- Leverage instrumentation in the web browser
  - the DOM's window.Load() event handler
  - retain the value of the previous Page's window.unload() event

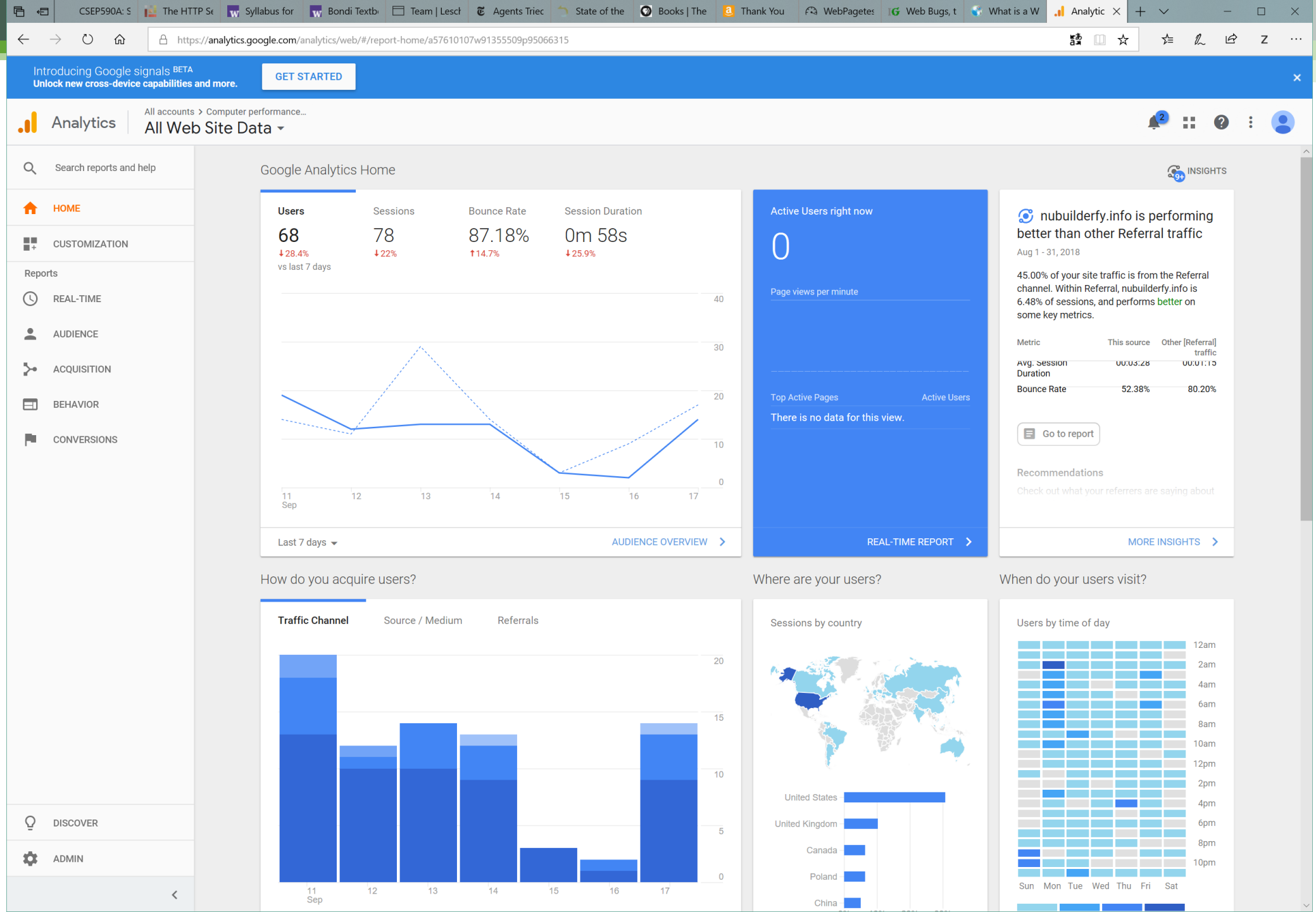
- the role of Google Analytics
  - evolved from the need to help buyers of Google Ad words understand how effective their ad buys were
  - inject the analytics.js script into web pages to instrument them
  - transmit the measurements back to Google using web beacons

Google Analytics

- Leverage instrumentation in the web browser
  - the DOM's window.Load() event handler
  - retain the value of the previous Page's window.unload() event

- the role of Google Analytics
  - evolved from the need to help buyers of Google Ad words understand how effective their ad buys were
  - inject the analytics.js script into web pages to instrument them
  - transmit the measurements back to Google using web beacons
  - report client visits, referrers, landing pages, bounce-backs, conversions, and abandonment

Google Analytics demo:



Google Analytics

- JavaScript instrumentation in the web browser
  - hook the DOM's window.Load() event handler
  - retain the value of the previous Page's window.unload() event
  - transmit the measurements back to Google using web beacons
- Chrome extensions to make gathering web application timing data easier and more reliable
  - subsequently adopted as W3C standards

- Google's "Need for Speed" initiative attempts to correlate PageLoad times with other browsing behavior (like conversions)

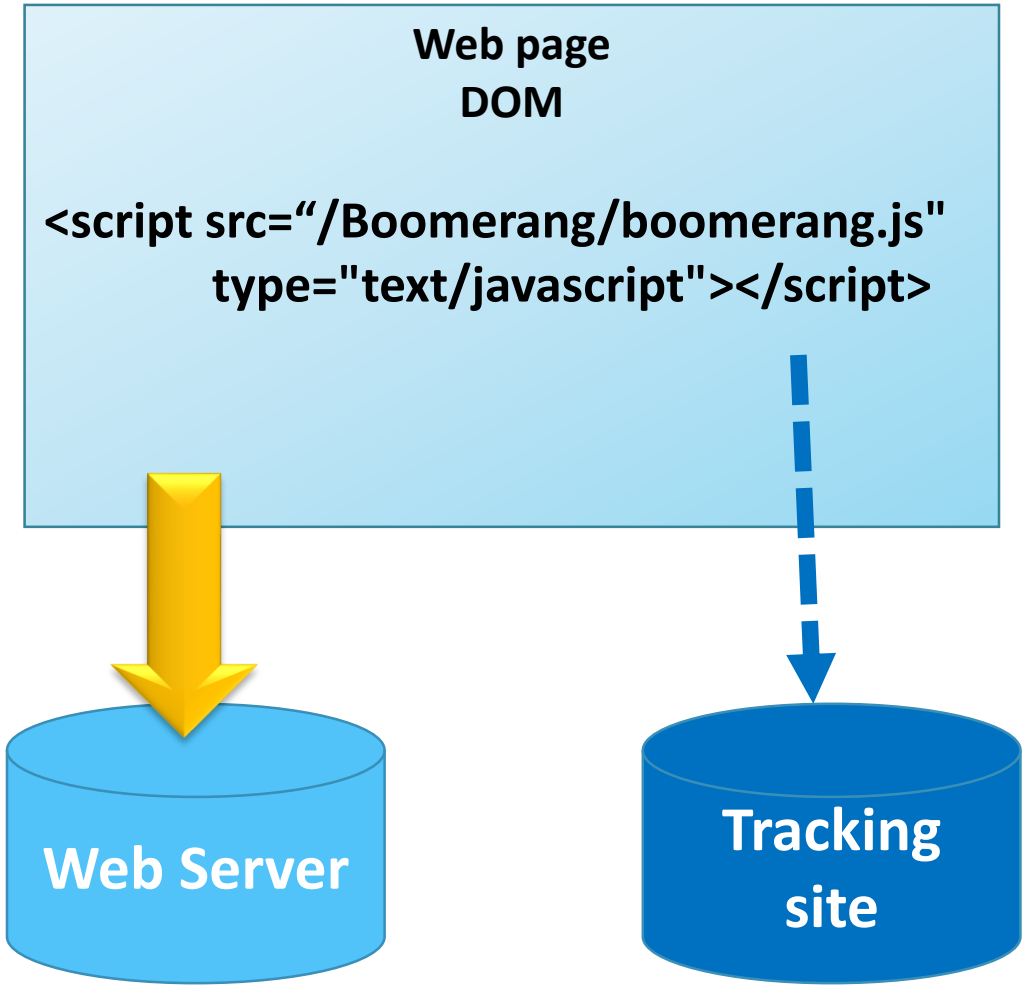
Web analytics

- Google Analytics is free for small volumes of data
  - Large customers need to pay for the reporting service
- Alternative products exist for large web sites whose owners do not want to send their usage information to Google

- Or, you can roll your own…
  - see, e.g., boomerang beacons project

Web analytics using web beacons

- the beacon is a trivial GET Request sent from the browser to an interested 3rd party
  - often for a one-pixel transparent .gif
  - the parms in the Request carry the data payload
- Beacons are the technique used by web usage tracking programs to transmit data from the client machine



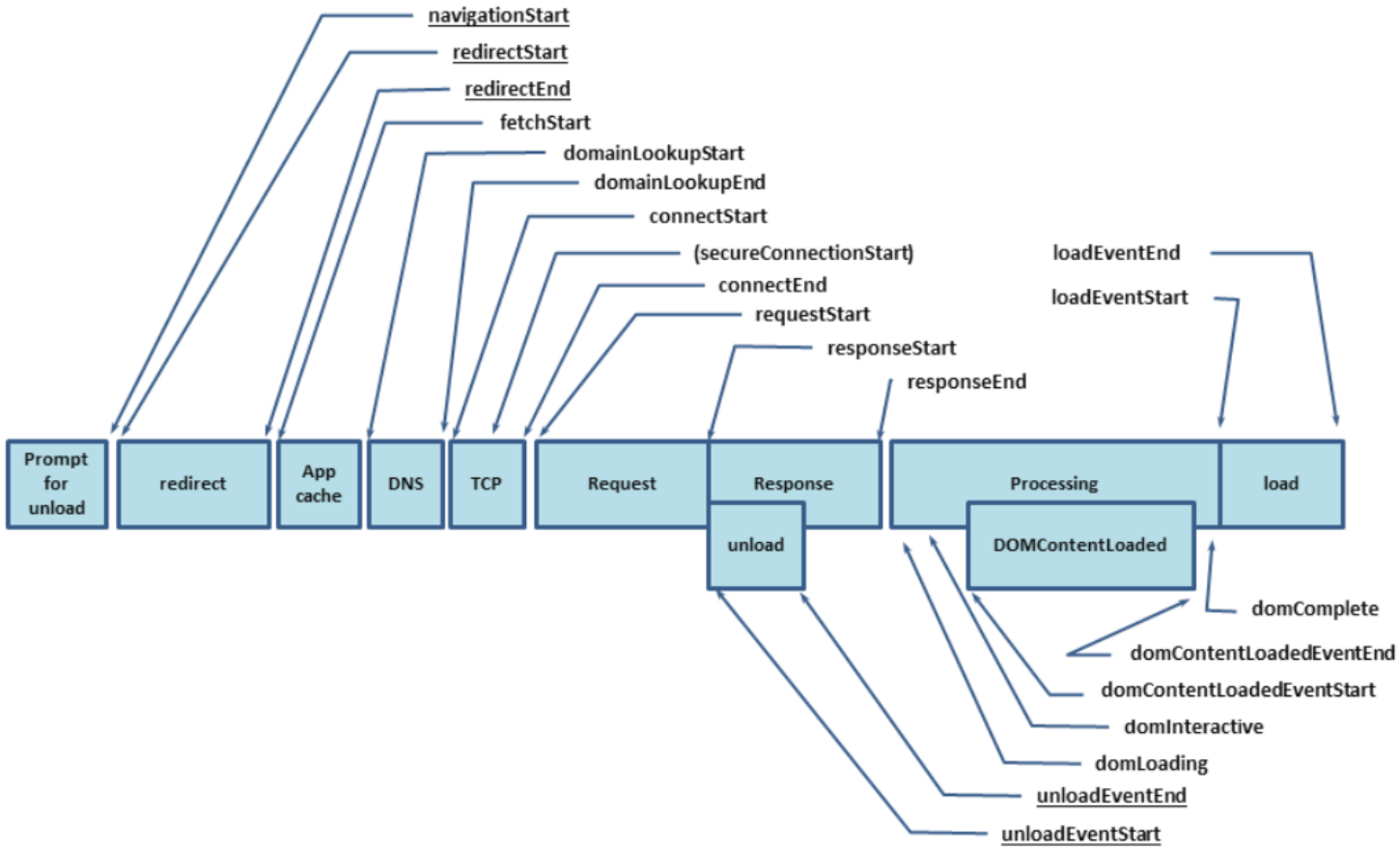
Boomerang web beacons

- boomerang.js
  - JavaScript, embedded in an HTML document, that gathers browser timing data and returns it to a specified URL using a web beacon
  - originally developed at Yahoo
  - adopted by SOASTA for use in mPulse, a web analytics reporting tool
    - Akamai, a leading 3rd party CDN, then purchased SOASTA

```
<script src="boomerang.js"></script>
<script src="plugins/rt.js"></script>
<script>
      BOOMR.init({
            beacon_url: "http://yoursite.com/beacon/"
});
</script>
```

Page Load Time Measurements

- **Http Request/Response and Rendering events**

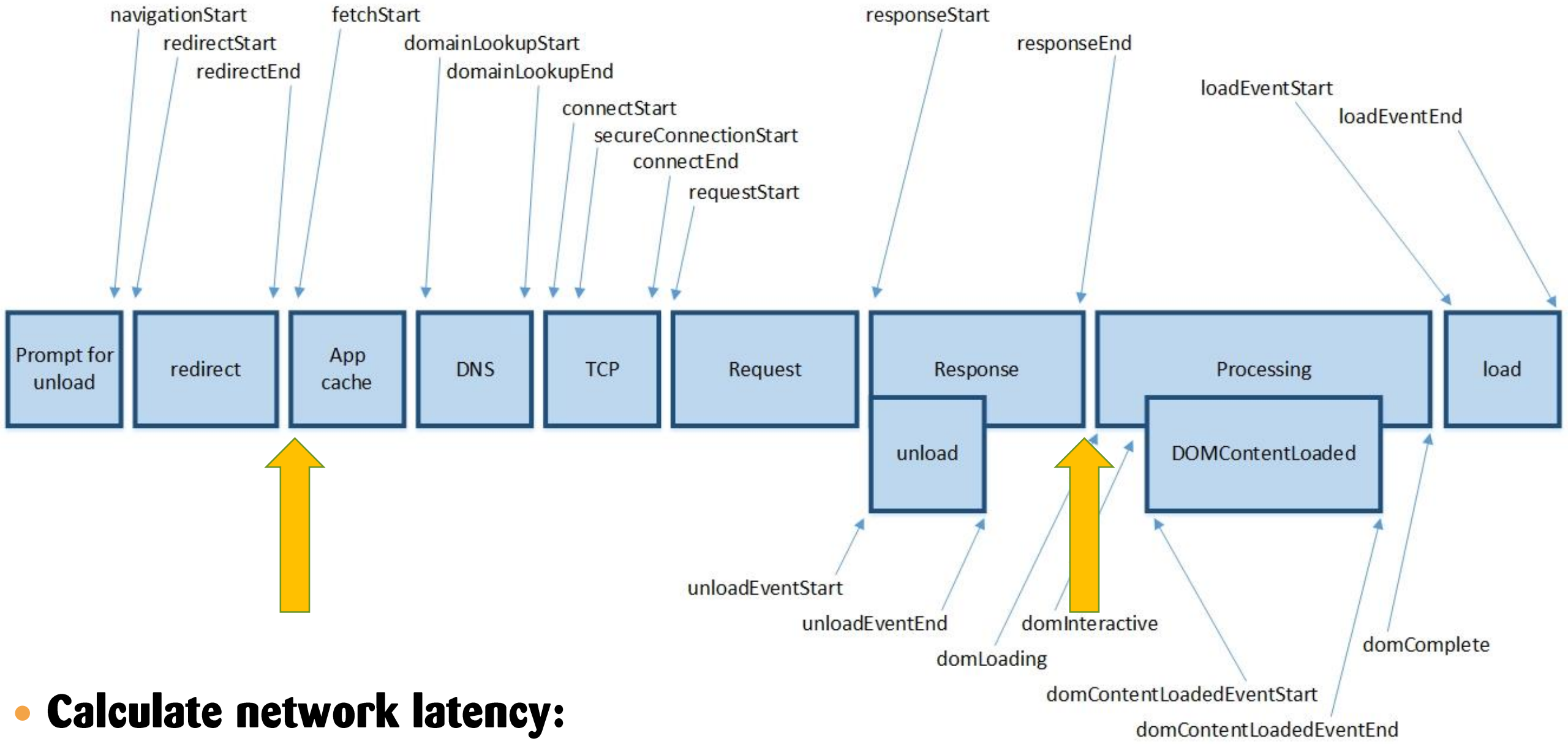


Navigation Timing spec: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html#processing-model

Page Load Time Measurements

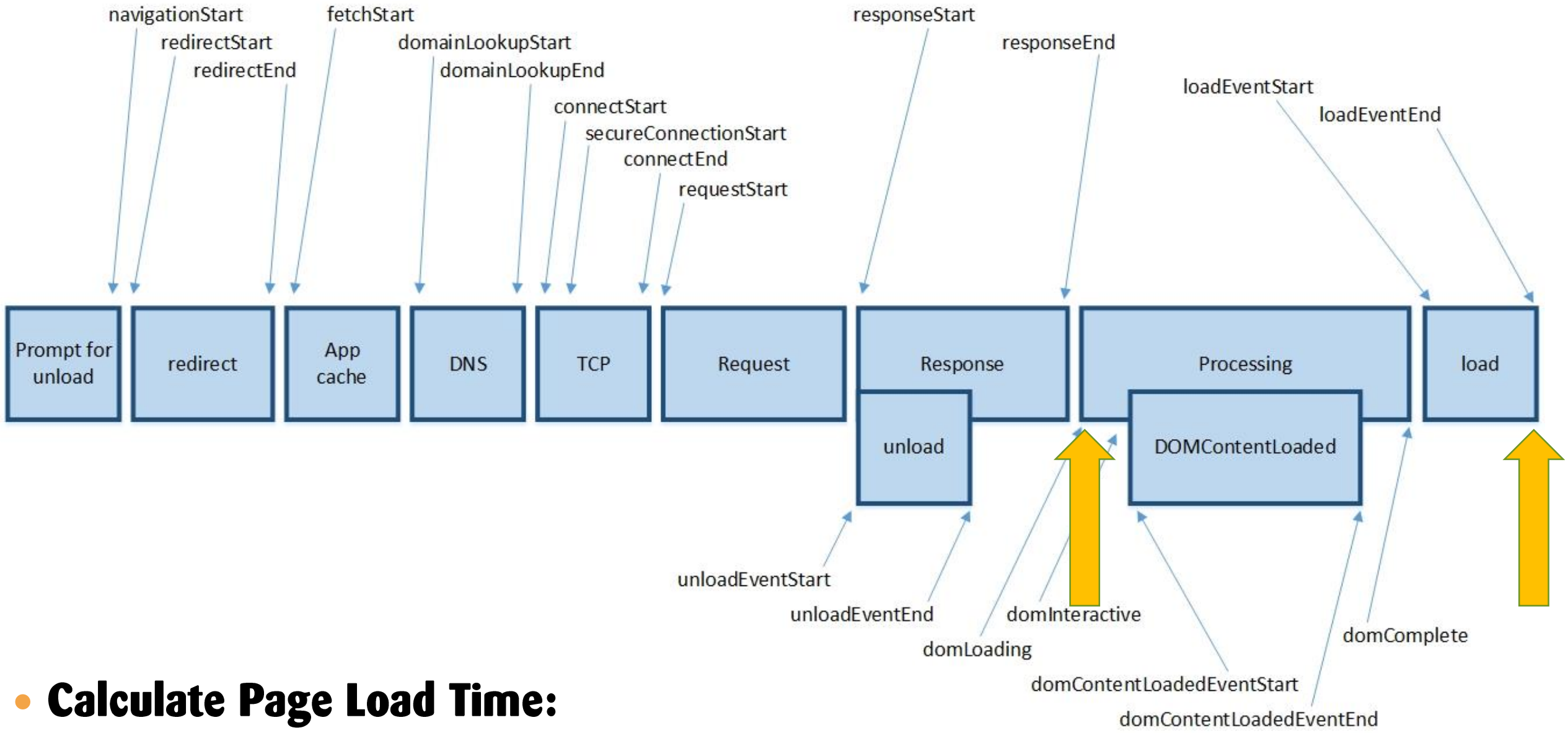
- **Http Request/Response and Rendering** events
- **Web Performance Working Group**
- Performance Timeline, Navigation Timing, and High Resolution Time specification
  - Supported in all major browsers
  - Event timings are used to calculate Real User Measurements (RUM):
    - **Network latency** (from the standpoint of the web client)
    - **Page Load time** (once the page components are received from the server)
    - Entire sequence from navigation to page load completion:

Navigation Timing spec: https://dvcs.w3.org/hg/webperf/raw-file/tip/specs/NavigationTiming/Overview.html#processing-model



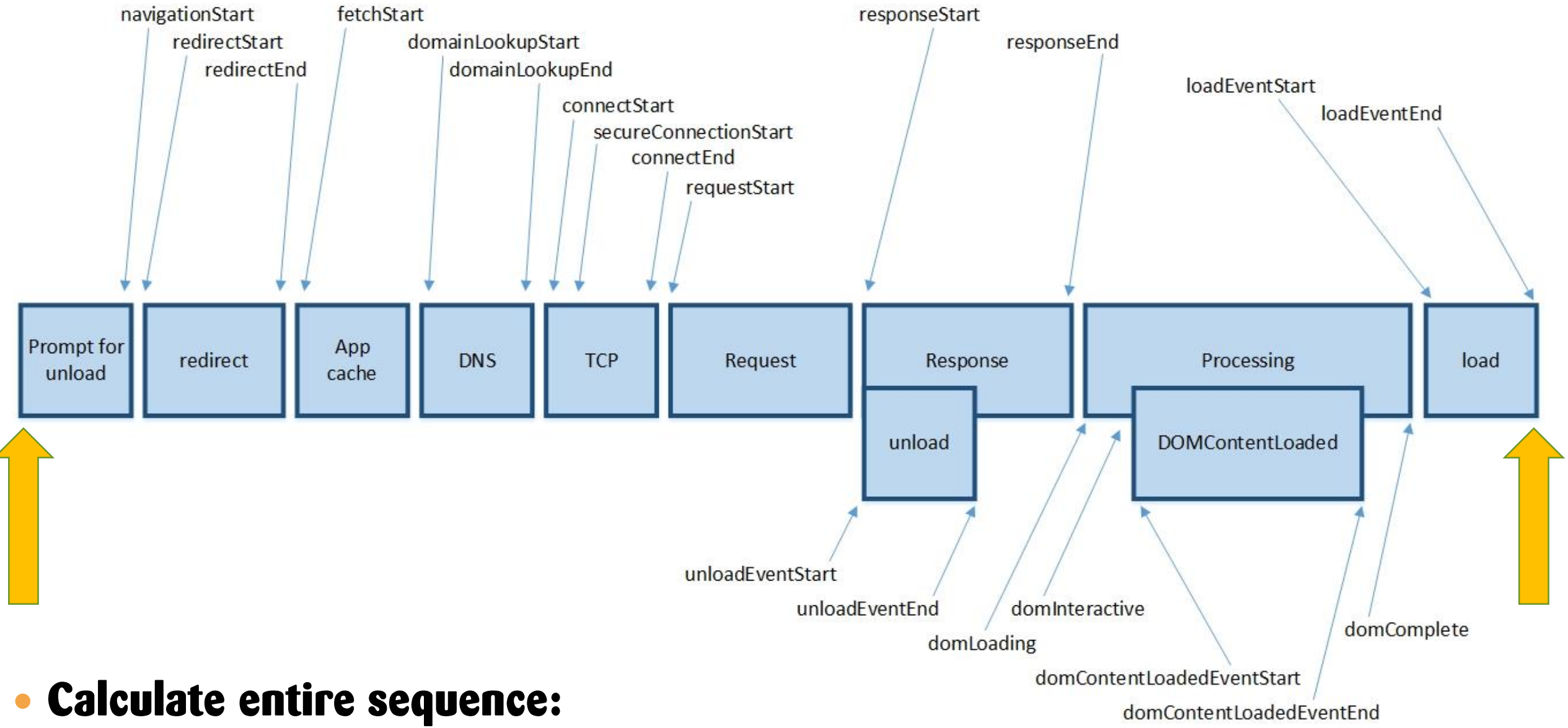
- **Calculate network latency:**

```
responseEnd - fetchStart
```



- **Calculate Page Load Time:**

```
loadEventEnd - responseEnd
```



- **Calculate entire sequence:**

```
loadEventEnd - navigationStart
```

window.performance.timing

DOM Event	(ms)	Δ	
navigationStart	1380736168062		Time after the previous document begins unload.
fetchStart	1380736168064	2	Time when the resource starts being fetched.
domainLookupStart	1380736168065	1	Time just before domain name lookup.
domainLookupEnd	1380736168065	0	Time after domain name lookup.
connectStart	1380736168065	0	Time just before server connection begins.
connectEnd	1380736168065	0	Time when server connection is finished.
requestStart	1380736168065	0	Time just before a server request.
responseStart	1380736168065	0	Time just before the start of a response.
responseEnd	1380736168065	0	Time after the end of a response or connection.
domLoading	1380736168065	0	Time just before readiness set to loading.
unloadEventStart	1380736168071	6	Time just before the unload event is fired.
unloadEventEnd	1380736168071	0	Time after the previous document is unloaded.
domInteractive	1380736168077	6	Time just before readiness set to interactive.
domContentLoadedEventStart	1380736168107	30	Time just before DOMContentLoaded starts.
domContentLoadedEventEnd	1380736168107	0	Time after DOMContentLoaded event completes.
domComplete	1380736168107	0	Time just before document readiness completes.
loadEventStart	1380736168112	5	Time just before the load event is fired.
loadEventEnd	1380736168115	3	Time when the load event is complete.

Boomerang web beacons

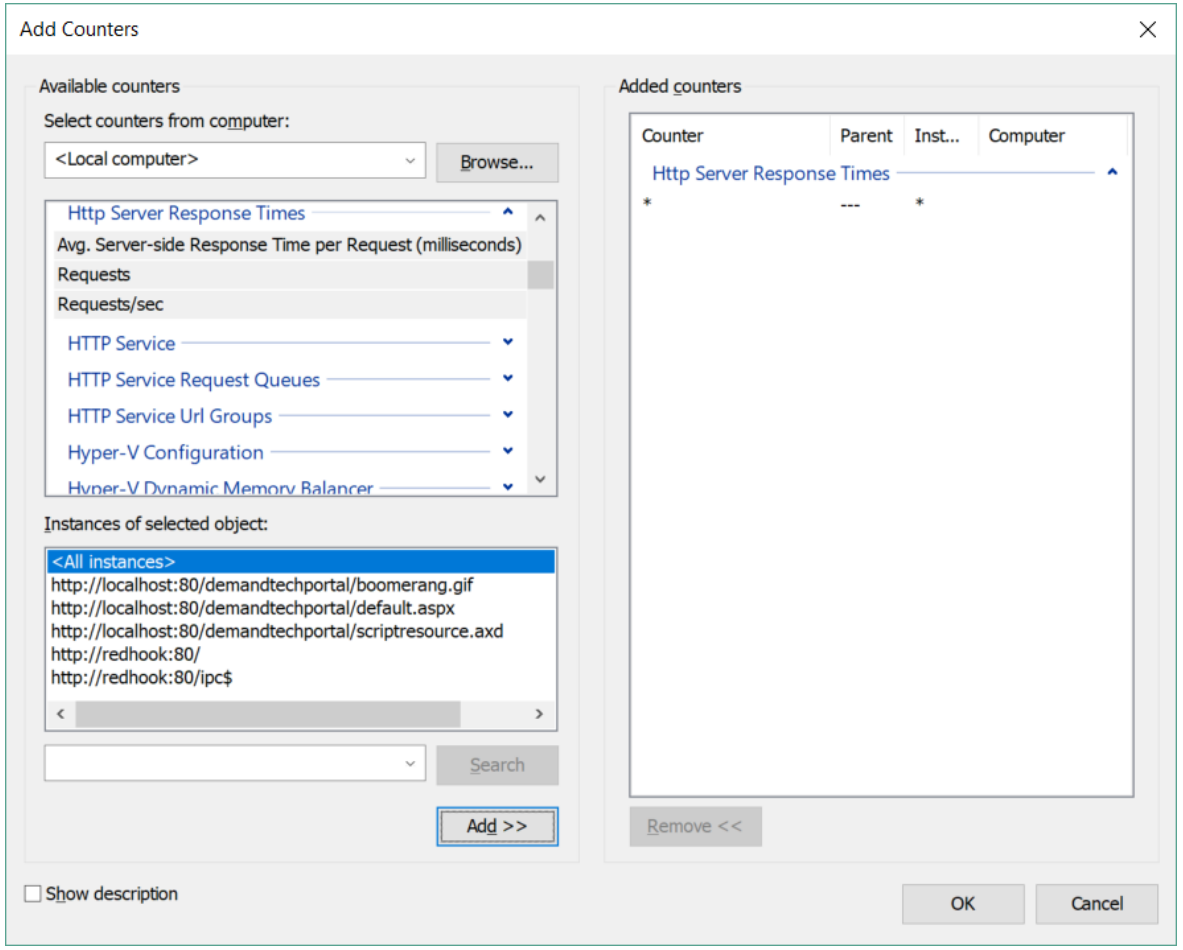
- boomerang.js
  - rt.js plug-in gathers the performance timing data and maps it into a series of name:value parameters added to the GET Request

```
<script src="boomerang.js"></script>
<script src="plugins/rt.js"></script>
```

Parm	Value
t_done	Perceived load time of the page
t_page	Time taken from the head of the page to the page_ready event
t_resp	Time taken from the user initiating the request to the first byte of the response
r	URL
r2	Referring URL
rt.tstart	start timestamp
rt.end	Note: t_done = rt.end - rt.tstart

Boomerang web beacons

- Intercept the beacons directed to your Web server
  - Process the timing data passed via the parms
  - In ASP.NET, e.g., implement the IHttpModule interface
    - generate ETW events
    - an ETW Listener that transforms them into performance counters and generates Alerts



window.performance.timing

Network latency: responseEnd - fetchStart

Page Load time: loadEventEnd - responseEnd

Entire sequence: loadEventEnd - navigationStart

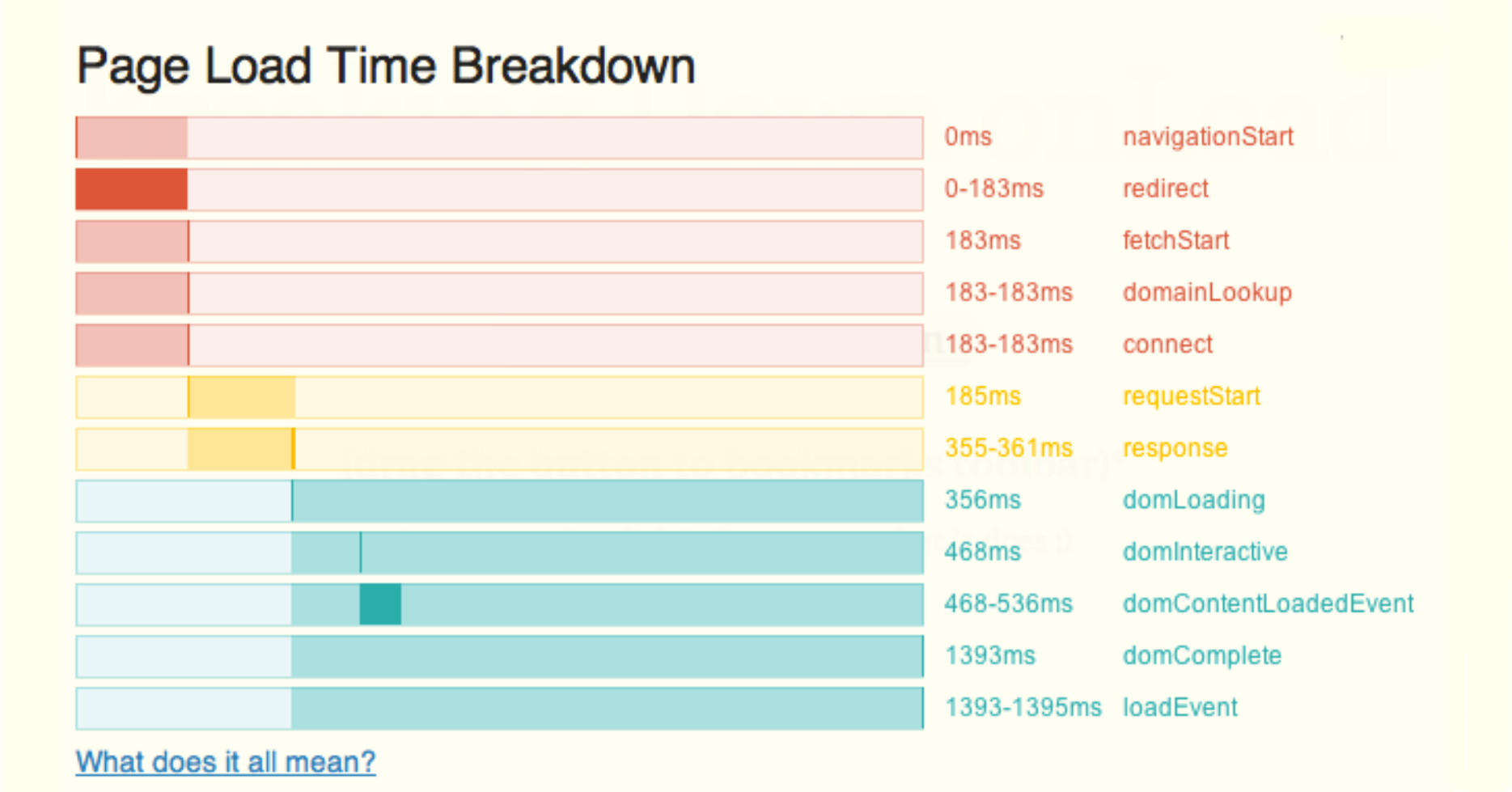
DOM Event	(ms)	Δ	
navigationStart	1380736168062		Time after the previous document begins unload.
fetchStart	1380736168064	2	Time when the resource starts being fetched.
domainLookupStart	1380736168065	1	Time just before domain name lookup.
domainLookupEnd	1380736168065	0	Time after domain name lookup.
connectStart	1380736168065	0	Time just before server connection begins.
connectEnd	1380736168065	0	Time when server connection is finished.
requestStart	1380736168065	0	Time just before a server request.
responseStart	1380736168065	0	Time just before the start of a response.
responseEnd	1380736168065	0	Time after the end of a response or connection.
domLoading	1380736168065	0	Time just before readiness set to loading.
unloadEventStart	1380736168071	6	Time just before the unload event is fired.
unloadEventEnd	1380736168071	0	Time after the previous document is unloaded.
domInteractive	1380736168077	6	Time just before readiness set to interactive.
domContentLoadedEventStart	1380736168107	30	Time just before DOMContentLoaded starts.
domContentLoadedEventEnd	1380736168107	0	Time after DOMContentLoaded event completes.
domComplete	1380736168107	0	Time just before document readiness completes.
loadEventStart	1380736168112	5	Time just before the load event is fired.
loadEventEnd	1380736168115	3	Time when the load event is complete.

Network & Server Latency

Page Load Time

window.performance.timing

see, for example, http://kaaes.github.io/timing/



performance.now High Resolution Clock

see https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

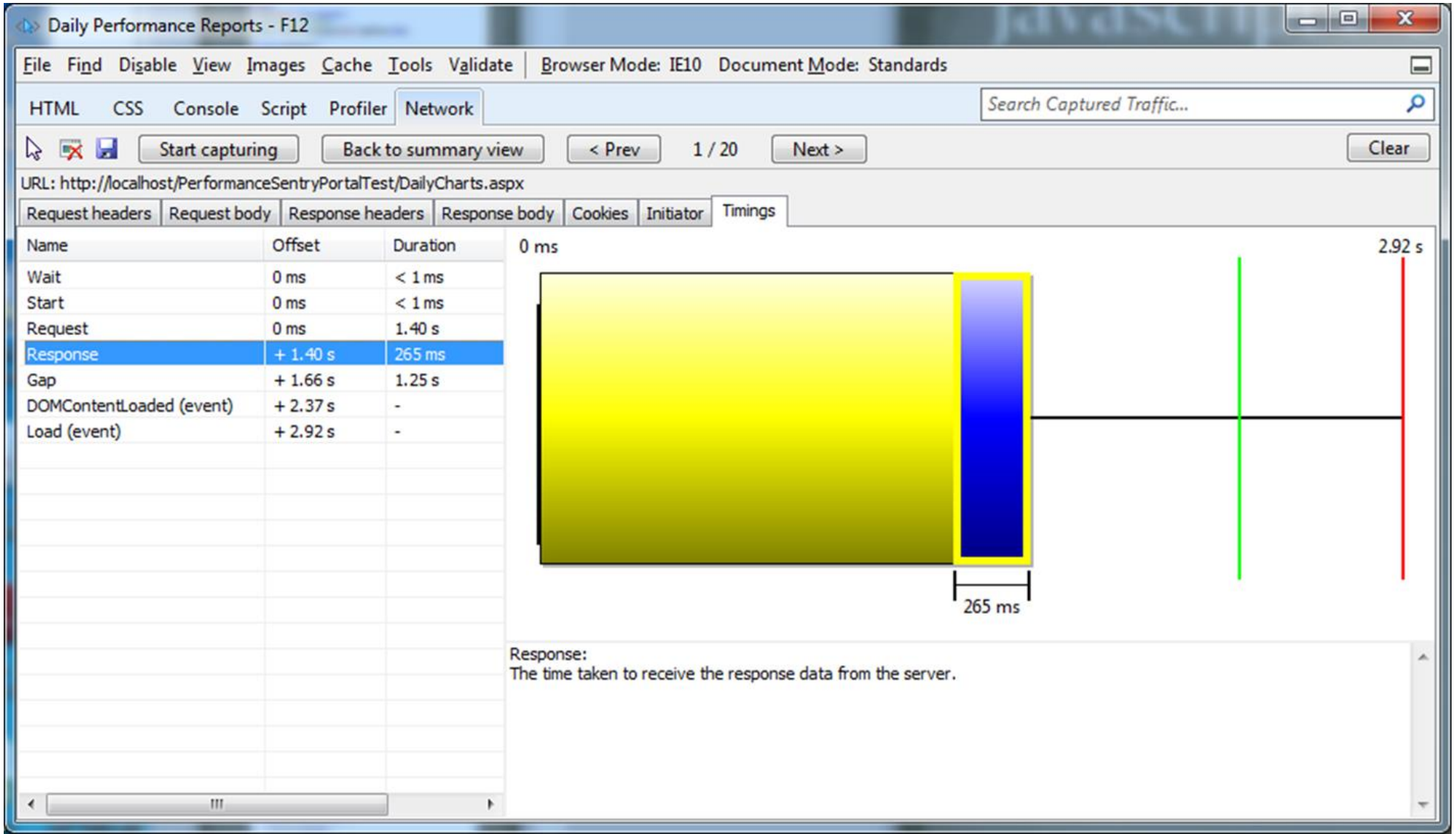
```
<html>
<head>
<script type="text/javascript">
// Add load event listener.
window.addEventListener("load", loadTime, false);

function loadTime() {
   // Get current time.
   var now = window.performance.now();
   // Calculate page load time.
   var page_load_time = now - performance.timing.navigationStart;
   // Write the load time to the F12 console.
   if (window.console)  console.log(page_load_time);
}
</script>
</head><body>
<!- Main page body is here. --> </body>
 </html>
```

Additional RUM resources

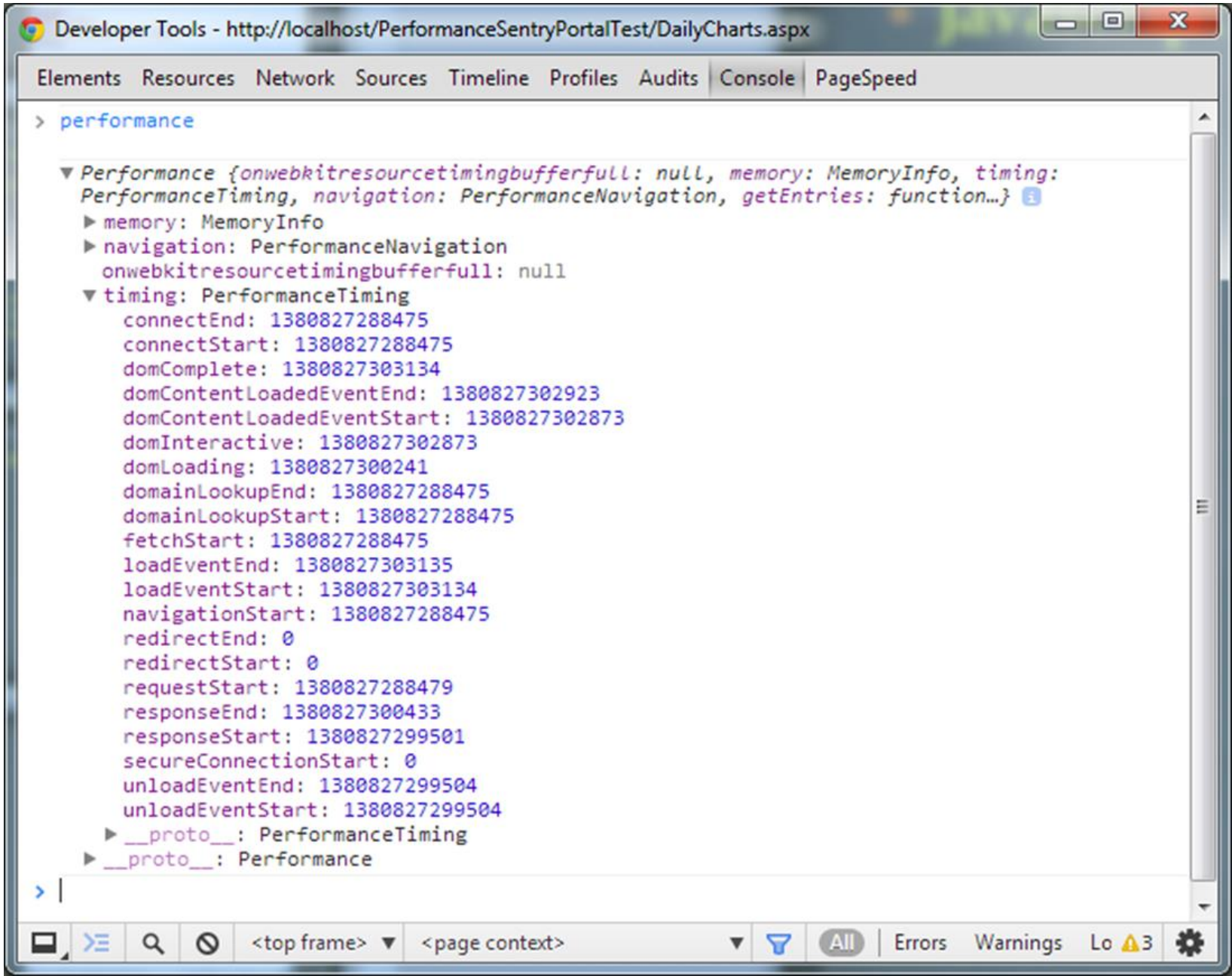
- W3C Web Performance Working Group
  - see http://www.w3.org/2010/webperf/ for latest docs
- IE doc:
  - Timing and Performance APIs: see http://msdn.microsoft.com/
  - coding examples:
    - http://msdn.microsoft.com/en-us/library/ie/hh673552(v=vs.85).aspx
    - http://ie.microsoft.com/testdrive/Performance/msPerformance/Default.html

- see http://www.html5rocks.com/en/tutorials/
- see also Andrea Trasatti's blog
- see also http://kaaes.github.io/timing/info.html

Case Study: IE Developer Tools



Case Study: IE Developer Tools



Proposed PerformancePaintTiming Interface

- Latest Google initiative:
  - https://w3c.github.io/paint-timing/#sec-PerformancePaintTiming
  - adds new timing data to the DOM Performance object
  - browser First Paint
  - browser First Contentful Paint
    - first time when users can start to consume page content – they understand that the Page is responding & something good is happening
- Available in Google's PageSpeed Insights tool
  - https://developers.google.com/speed/pagespeed/insights/

Page Load times

Is it happening?	Has the server responded?	First Paint
Is it useful?	Has enough content rendered that users can engage with it?	First Contentful Paint
Is it useable?	Can the user interact with the page?	Document Content Load
Is it delightful?	Are the interactions smooth and natural?	Animations, video streaming, etc.

- New browser instrumentation:
  - PerformanceObserver, PerformanceEntry, DOMHighResTimeStamp

- See User-centric Performance Metrics

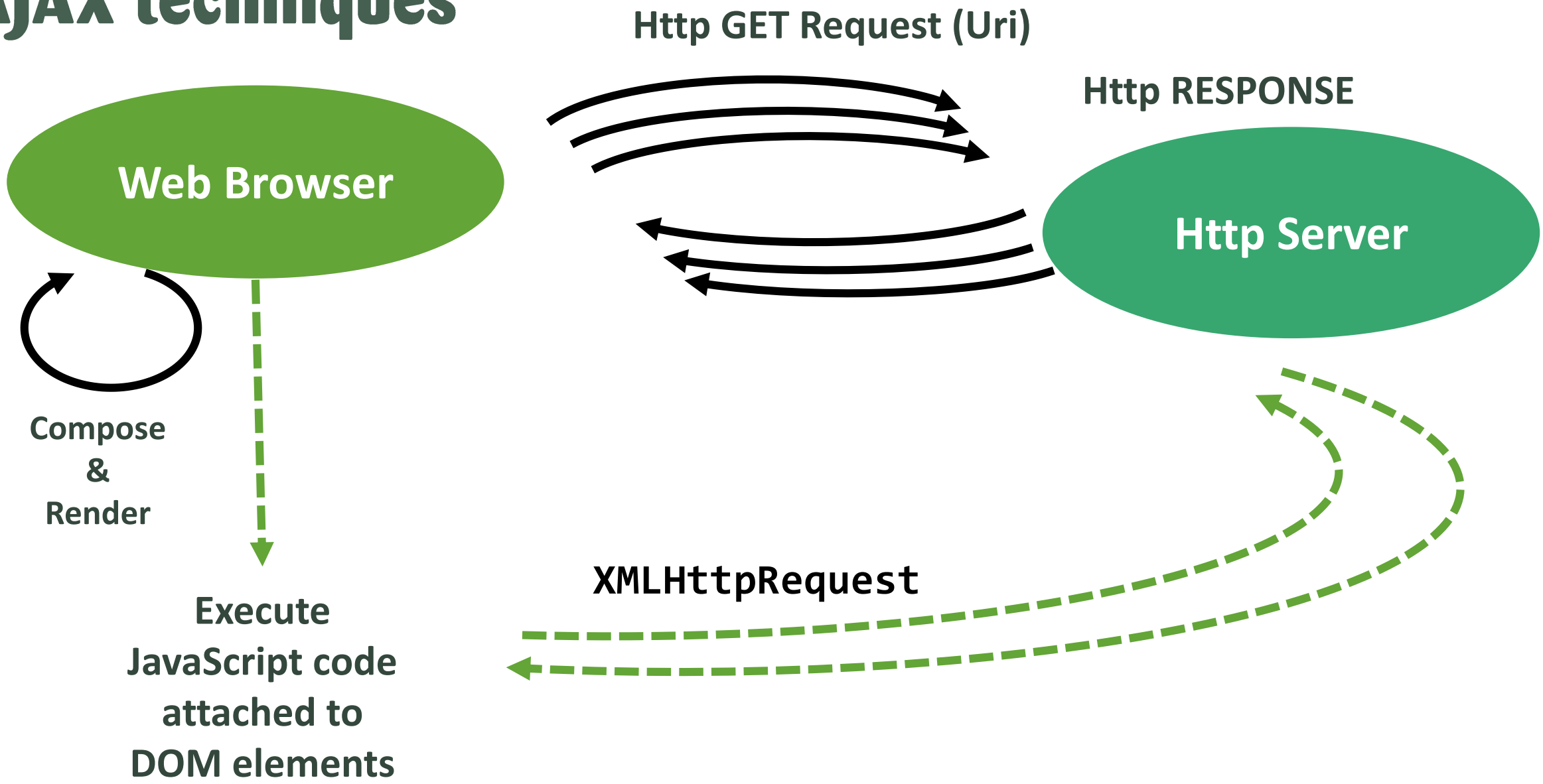
YSlow scalability model complications

- JavaScript interactivity makes it difficult to fix the Request:Response boundary
  - e.g., self-contained script code to handle mouseover or button click events
  - e.g., AutoComplete script used by the Search engines
    - self-contained when the Search history can be cached in local storage on the client machine
    - autocompletion hints based on input data required accessing frequent search result sets from the web server (AJAX)
  - Naturally, any Requests for long running web tasks should not tie up the foreground UI (and should use AJAX instead)

YSlow scalability model complications

- JavaScript interactivity makes it difficult to fix the Request:Response boundary
  - AJAX techniques
    - Asynchronous JavaScript and XML
    - Asynchronous `XMLHttpRequests` for JSON or XML objects run in background threads
    - while JavaScript interactions can continue to run concurrently in the foreground

AJAX techniques



YSlow Scalability model assessment

- The YSlow rules do apply to many commercial apps & portals…
  - Pages composed of many, rapidly changing, modular elements
    - Different page elements may be retrieved from different web servers (e.g., ad servers)
  - Customization based on identity and location
  - Implications:
    - More frequency, but smaller, Request/Response sequences
    - Caching strategies
    - Size of client-resident cookies and other techniques for managing the state of customer interaction

YSlow Scalability model assessment

- The YSlow scalability model is useful, but it is not adequate for many web applications
  - Applies mainly to traditional web browser client interactions
  - May not be adequate for many ASP.NET applications that generate Http Response messages dynamically
  - May not be adequate for web browser apps that make extensive use of web services (AJAX)
  - Not adequate for animation and streaming apps

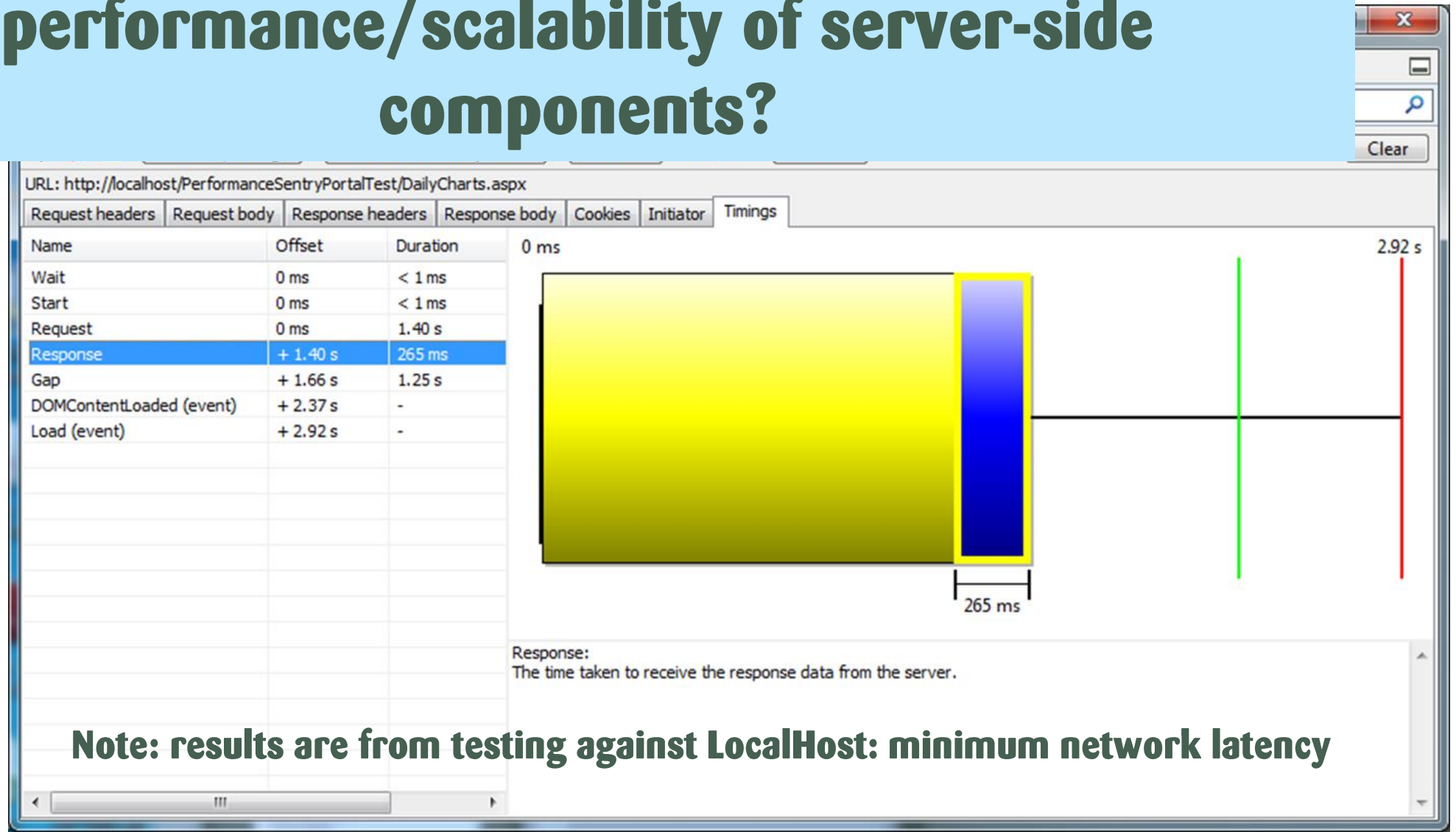
  - Does not apply to native client apps that make use of web services

YSlow Scalability model assessment

- Page Load time is a measure of end-to-end response time
  - Navigation Timing measurements decompose overall response time into Network, Server, and Client (i.e., web browser) components

- YSlow is silent:
  - the **scalability** of server-side components
  - the **diversity** of web client hardware (PCs, tablets, phones) & software (iOS, Android, Windows)
  - the performance of the client's **network connection**
    - *e.g.,* Internet vs. Intranet connections

What if we need to look at the performance/scalability of server-side components?



Note: results are from testing against LocalHost: minimum network latency

Assignment

- Evaluate the waterfall graph of Page Load time from:
  - Developer Tools in your favorite web browser, or
  - WebPageTest

to measure Page Load times for three of your favorite web sites

- Report back to the class on your experience

  - Clear Cache and compare cold-start to warm-start Page Load times
  - Things to Look for:
    - YSlow Best Practices in min.js, image compression, etc.
    - parallel TCP sessions
    - cached content in edge networks (CDNs)

Questions

References

- Friedman, “Why is this web app running slowly?”
- Steve Souders, *High Performance Web Sites: Essential Knowledge for Front-End Engineers.*
- Steve Souders, *Even Faster Web Sites: Performance Best Practices for Web Developers.*