Web Performance

Sergey Chernyshev

March '09 New York Web Standards Meetup

New York, NY

March 19th, 2009

About presenter

- Doing web stuff since 1995
- Director, Web Systems and Applications at truTV
- Personal projects
  - *TechPresentations.org*
  - *MediaWikiWidgets.org*
  - *SharingButtons.org*
  - *HowWebWorks.com (in works)*

What is this talk about?

- Performance is how fast your site works
- **NOT** how many users it can serve (Scalability)
- **NOT** how often it's down (Reliability)

Why do we need to care?

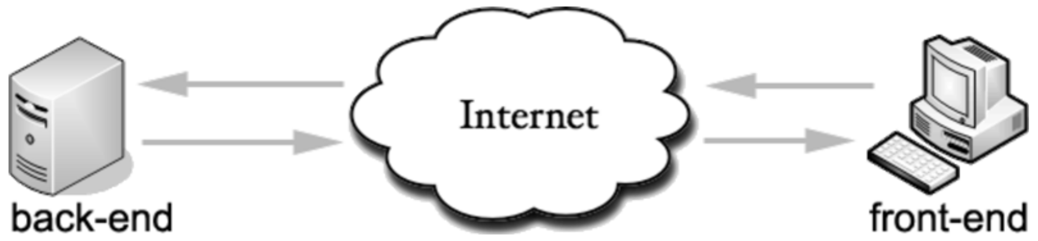
- User experience
- Money!!!
    - ***+100ms → -1% sales (Amazon) ****
    - ***+400ms → -5-9% full-page traffic (Yahoo! Autos front page) *****
    - ***+500ms → -20% searches (Google) ******

* Make Data Useful, Greg Linden at Stanford Data Mining class, fall 2006

** YSlow 2.0 early preview in China, Stoyan Stefanov, December 6th, 2008

*** Scaling Google for Every User, Marissa Mayer at Google Seattle Conference on Scalability 2007

How web works?



Backend

- Usually planned for along with scalability and reliability
- Usual tree-tier
  - *Browser (front-end)*
  - *Web/app server*
  - *Relational database*

DataBase performance

- use INDEXES!
- Study set theory basics and use JOINs instead of cursors or code-level iterations
- Use correct datatypes to put as much stuff into memory as possible
- Use query cache
- Read documentation for your RDBMS *

* For MySQL, 15 Ways to Kill Your MySQL Application Performance by Jay Pipes at PHP Tek 2007

Compile

- Compiled code is faster then interpreted
- C/C++ web apps are rare
- ASP.NET and Java compile into bytecode
- But most of the web is interpreted, use opcode caches
    - *APC for PHP **
    - *mod_perl for perl*

* APC at Facebook by Brian M. Shire and Facebook Performance Caching by Lucas Nealan at PHP Tek 2008

Cache

- Cache results
  - *Shared memory (APC and EAccelerator for PHP)*
  - *memcached*
- Cache not even data, but chunks od HTML
- Reverse proxies (nginx, Squid)

CDNs or lightweight web servers for static content

- CDNs are closer to the user and remove load your app servers
    - *Akamai (also has Edge Suite - reverse proxy)*
    - *Limelight*
- *mystaticfiles.net* with lightweight servers
    - *nginx (also reverse proxy)*
    - *lighttpd*

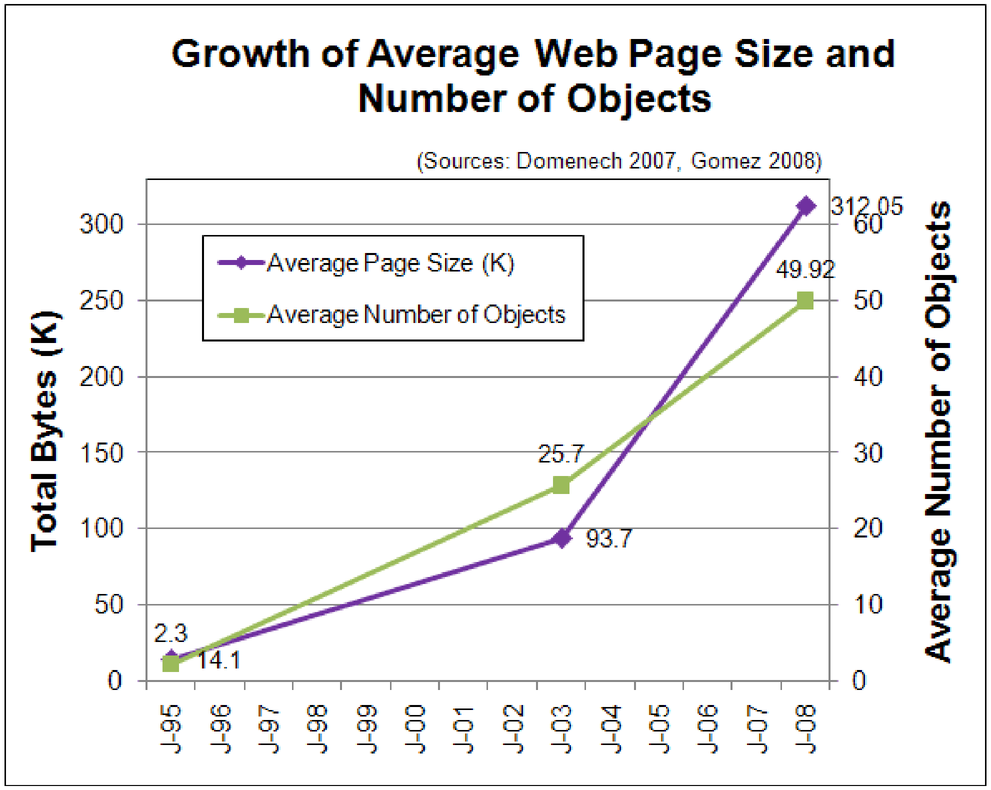
Front-end

- Face of the web
- Heavily influenced by marketing, brand and etc.
- Performance wasn't planned for until Web 2.0
- Slowness became noticable by users

Front-end: Start Here!

Amount of media and requests per page grew exponentially.

1995 - 2008

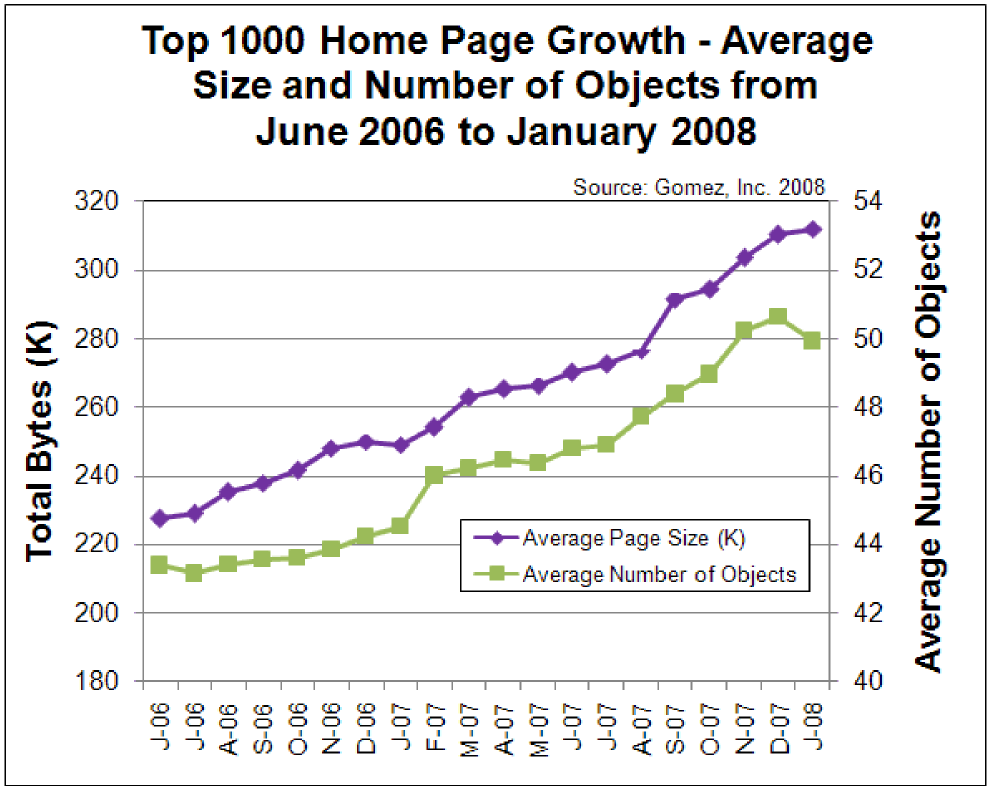


* Average Web Page Size Triples Since 2003, as of end of 2007 (via Nicole Sullivan)

Front-end: Start Here!

Top 1000 Home pages, ~25-30 growth in just 1.5 years.

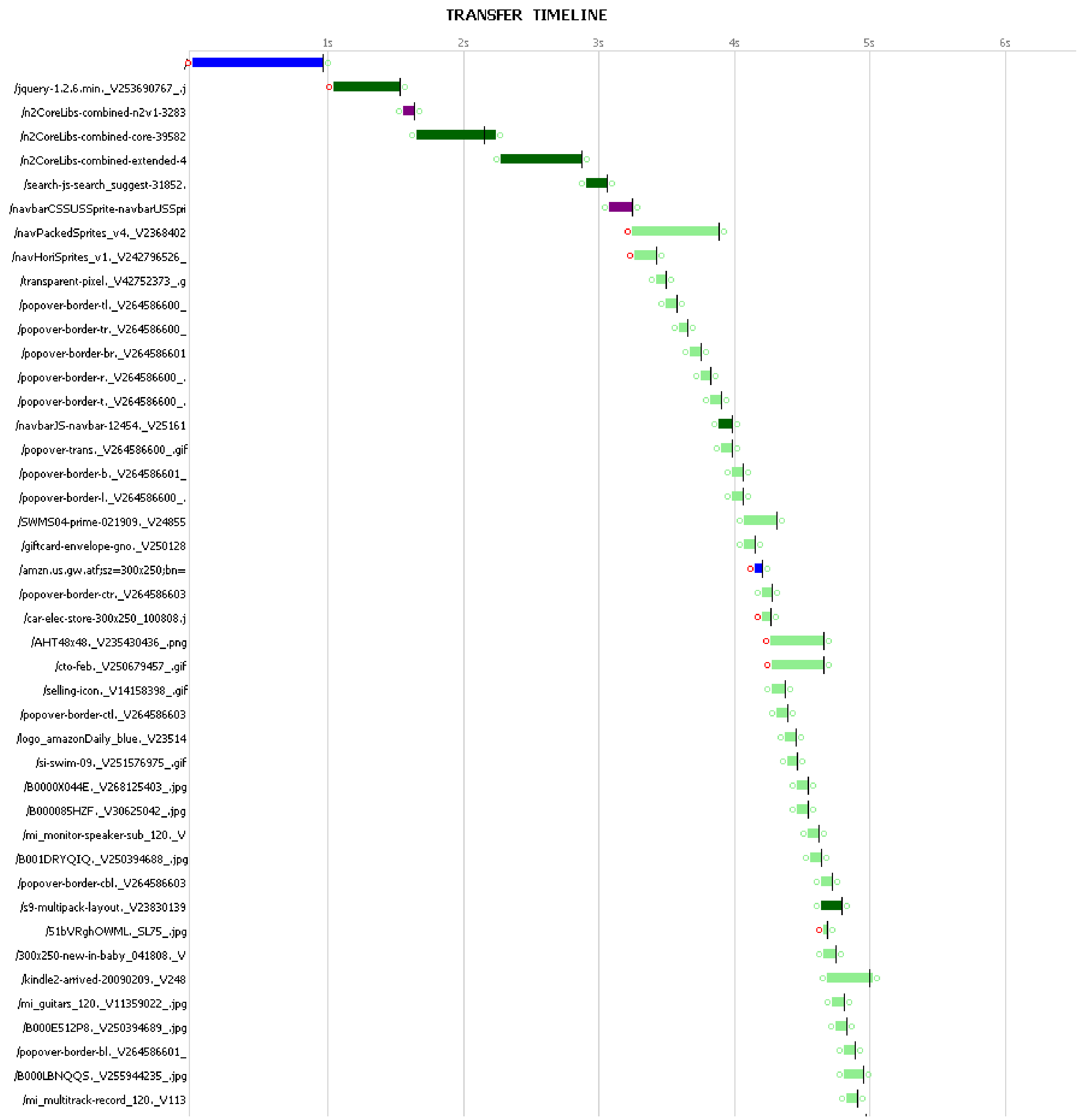
Jun 2006 - Jan 2008



* Average Web Page Size Triples Since 2003, as of end of 2007 (via Nicole Sullivan)

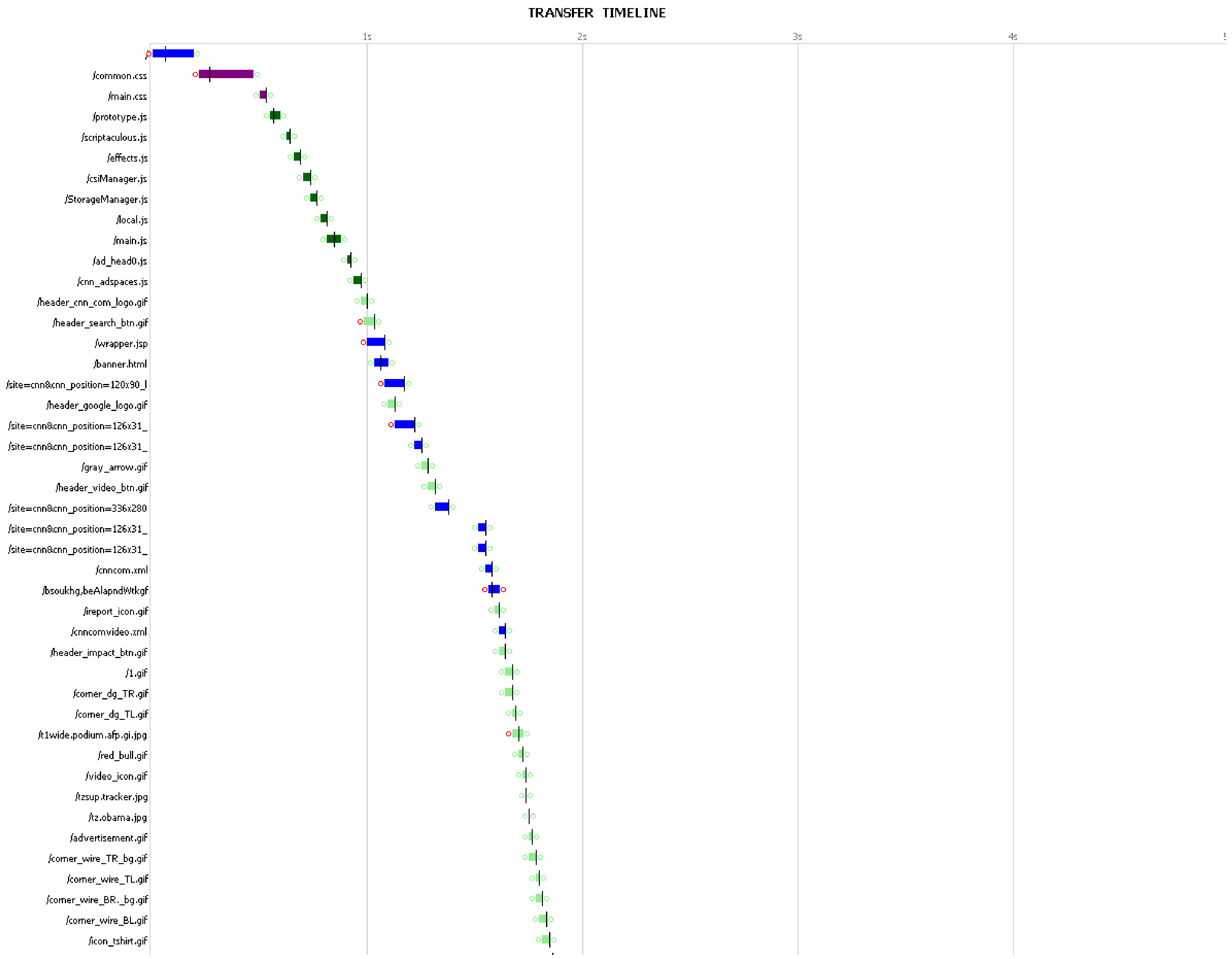
Amazon Waterfall!

- Total Requests: 88
- Total Time: 6.344 seconds
- Back-end Time: 0.968 seconds = **just 15%**


TRANSFER TIMELINE
1s
2s
3s
4s
5s
6s
/jquery-1.2.6.min._V253690767_.j
/n2CoreLibs-combined-n2v1-3283
/n2CoreLibs-combined-core-39582
/n2CoreLibs-combined-extended-4
/search-js-search_suggest-31852.
/navbarCSSUSSprite-navbarUSSpi
/navPackedSprites_v4._V2368402
/navHoriSprites_v1._V242796526_
/transparent-pixel._V42752373_.g
/popover-border-tl._V264586600_
/popover-border-tr._V264586600_
/popover-border-br._V264586601
/popover-border-r._V264586600_.
/popover-border-t._V264586600_.
/navbarJS-navbar-12454._V25161
/popover-trans._V264586600_.gif
/popover-border-b._V264586601_
/popover-border-l._V264586600_.
/SWMS04-prime-021909._V24855
/giftcard-envelope-gno._V250128
/amzn.us.gw.atf;sz=300x250;bn=
/popover-border-ctr._V264586603
/car-elec-store-300x250_100808.j
/AHT48x48._V235430436_.png
/cto-feb._V250679457_.gif
/selling-icon._V14158398_.gif
/popover-border-ctl._V264586603
/logo_amazonDaily_blue._V23514
/si-swim-09._V251576975_.gif
/B0000X044E._V268125403_.jpg
/B000085HZF._V30625042_.jpg
/mi_monitor-speaker-sub_120._V
/B001DRYQIQ._V250394688_.jpg
/popover-border-cbl._V264586603
/s9-multipack-layout._V23830139
/51bVRghOWML._SL75_.jpg
/300x250-new-in-baby_041808._V
/kindle2-arrived-20090209._V248
/mi_guitars_120._V11359022_.jpg
/B000E512P8._V250394689_.jpg
/popover-border-bl._V264586601_
/B000LBNQQS._V255944235_.jpg
/mi_multitrack-record_120._V113

CNN Waterfall!

- Total Requests: 174
- Total Time: 4.406 seconds
- Back-end Time: 0.171 seconds = **less then 4%**


TRANSFER TIMELINE
1s
2s
3s
4s
5
/
/common.css
/main.css
/prototype.js
/scriptaculous.js
/effects.js
/csiManager.js
/StorageManager.js
/local.js
/main.js
/ad_head0.js
/cnn_adspaces.js
/header_cnn_com_logo.gif
/header_search_btn.gif
/wrapper.jsp
/banner.html
/site=cnn&cnn_position=120x90_l
/header_google_logo.gif
/site=cnn&cnn_position=126x31_
/site=cnn&cnn_position=126x31_
/gray_arrow.gif
/header_video_btn.gif
/site=cnn&cnn_position=336x280
/site=cnn&cnn_position=126x31_
/site=cnn&cnn_position=126x31_
/cnncom.xml
/bsoukhg.beAlapndWtkgf
/ireport_icon.gif
/cnncomvideo.xml
/header_impact_btn.gif
/1.gif
/corner_dg_TR.gif
/corner_dg_TL.gif
/t1wide.podium.afp.gi.jpg
/red_bull.gif
/video_icon.gif
/tzsup.tracker.jpg
/tz.obama.jpg
/advertisement.gif
/corner_wire_TR_bg.gif
/corner_wire_TL.gif
/corner_wire_BR_bg.gif
/corner_wire_BL.gif
/icon_tshirt.gif

Yahoo!

- Developed YUI design patterns and JavaScript library
- Had to pay attention to performance as part of experience
- Exceptional Performance Team at Yahoo!
- Steve Souders* (now at Google)
- Developed prioritized list of best practices
- Developed YSlow** to test performance and promote to business within Yahoo!

* Steve Souders @ TechPresentations

** YSlow @ TechPresentations

Best Practices (34 already)

- Make Fewer HTTP Requests
- Use a Content Delivery Network
- Add an Expires or a Cache-Control Header
- Gzip Components
- Put Stylesheets at the Top
- Put Scripts at the Bottom
- Avoid CSS Expressions
- Make JavaScript and CSS External
- Reduce DNS Lookups
- Minify JavaScript and CSS
- Avoid Redirects
- Remove Duplicate Scripts
- Configure ETags
- Make Ajax Cacheable
- Flush the Buffer Early
- Use GET for AJAX Requests
- Post-load Components
- Preload Components
- Reduce the Number of DOM Elements
- Split Components Across Domains
- Minimize the Number of iframes
- No 404s
- Reduce Cookie Size
- Use Cookie-free Domains for Components
- Minimize DOM Access
- Develop Smart Event Handlers
- Choose <link> over @import
- Avoid Filters
- Optimize Images
- Optimize CSS Sprites
- Don't Scale Images in HTML
- Make favicon.ico Small and Cacheable
- Keep Components under 25K
- Pack Components into a Multipart Document

Top Best Practices

Most effective, tested by YSlow

- Make Fewer HTTP Requests
- Use a Content Delivery Network
- Add an Expires or a Cache-Control Header
- Gzip Components
- Put Stylesheets at the Top
- Put Scripts at the Bottom

Top Best Practices (cont'd)

- Avoid CSS Expressions
- Make JavaScript and CSS External
- Reduce DNS Lookups
- Minify JavaScript and CSS
- Avoid Redirects
- Remove Duplicate Scripts
- Configure ETags

More Front-end Best Practices

- Flush the Buffer Early
- Post-load Components
- Preload Components
- Reduce the Number of DOM Elements
- Minimize the Number of iframes
- Optimize Images
- Optimize CSS Sprites

Demo

- YSlow (Firefox)
- Firebug's Net panel (Firefox)
- Fiddler 2 Beta (IE)
- AOL Page Test / WebPageTest.org (IE)

How do I start?

How do I start? Business people

- use metrics that include site performance, tie it in to your bottom line (e.g. can you afford loosing 20% of your traffic to 500ms slowness? What does it cost you to bridge this gap?)
- incorporate performance testing in QA process (developers should not care more then you do). Use YSlow - it speaks business language.
- but be careful, it might be too expensive to excel on all levels (still cheap to get a lot of improvement)

How do I start? Designers

- watch Design Fast Websites presentation by Nicole Sullivan (Yahoo!, co-author of Smoosh.it)
- start designing experience, not digital paper
- use consistent styles (reusable CSS, cached)
- don't mandate effects that require heavy lifting on browser side
  - *Rounded corners can still be fast*
  - *Transparent PNGs can still be fast*
- use graceful degradation for less capable browsers (IE6)

How do I start? Front-end developers

- most of it is on your shoulders
- include YSlow grades in your year end review and resume
- include performance into your definition of good craftmanship (it's a rare case when you can use TDD effectively)
- learn how browsers work, DOM, events, if that trick of yours really makes it faster
- reduce amount of requests, use one CSS file (load it first), one JS file (load as late as possible), CSS sprites
- "smush" images, compress CSS and JS, automate it
- change your code and publishing process to allow infinite expiration of assets (if you can do that for HTML, ask to double your bonus)

How do I start? Backend developers

- you should already know what to do - performance culture is ages old here
- still heavily depends on your apps and connected to scalability and reliability
- use indexes in your databases! Monitor how they perform as you dataset grows
- use caches (RAM is cheap, disks are even cheaper)

How do I start? System administrators

- talk to designers and developers and even business people
- read RFC 2616 (HTTP) and Apache manual
- configure gzipping (mod_deflate) and expires
- install APC, memcached, Squid, nginx

More info

- TechPresentations.org/Performance
- Yahoo! Exceptional Performance Team

Tools

- Diagnostic Tools
  - *YSlow for Firebug*
  - *Fiddler2 (WinINET)*
  - *WebPageTest.org and AOL Page Test*
  - *HttpWatch ($$$)*
- Content Compressors
  - *JSMin by Douglas Crockford*
  - *YUI Compressor - for JS and CSS*
  - *SmushIt.com - for images (web interface + API, not Open Source yet)*
- Server-side
  - *APC for PHP*
  - *memcached*
  - *mod_deflate for Apache2*
  - *nginx - lightweight reverse proxy*
  - *Squid - caching reverse proxy*
  - *Akamai, Limelight - CDNs*

About these slides

These slides use an HTML presentation tool Slidy.

They are valid XHTML document with rich metadata embedded using RDFa.

Licensing & Attribution