

facebook

(c) 2009 Facebook, Inc. or its licensors. "Facebook" is a registered trademark of Facebook, Inc.. All rights reserved. 1.0

facebook

Making Facebook faster

Frontend performance engineering

David Wei and Changhao Jiang

Velocity 2009
Jun 24, 2009 San Jose, CA

Agenda

- 1** Site speed matters
- 2** Performance monitoring
- 3** Static resource management
- 4** Ajaxification
- 5** Client side cache

Site speed matters!

Site speed matters: large scale

200 million users, more than 4 billion page views / day

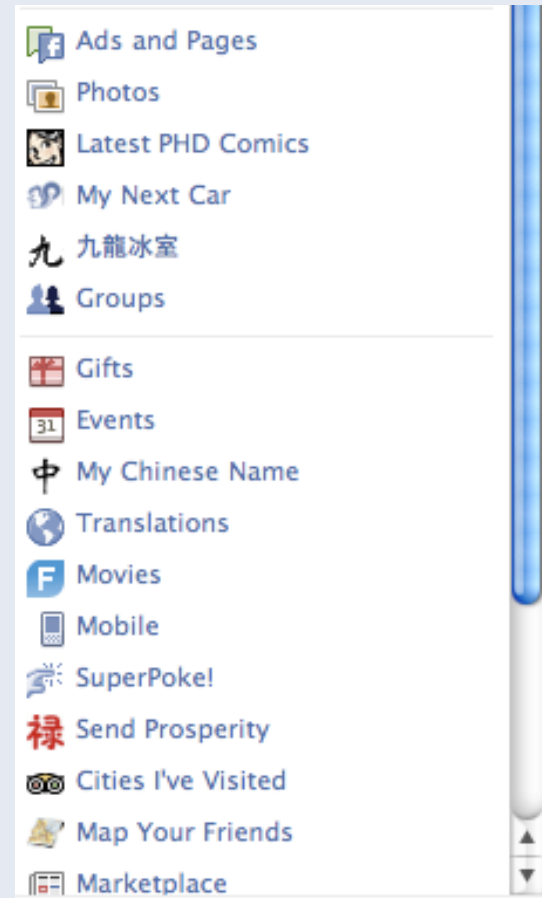
- 10ms per page = more than 1 man-year per day
= more than 5 human-life of time per year

Site speed matters: emerging challenges

- Agile development

Site speed matters: emerging challenges

- Agile development
- Deep integration



Site speed matters: emerging challenges

- Agile development
- Deep integration
- Viral adoption



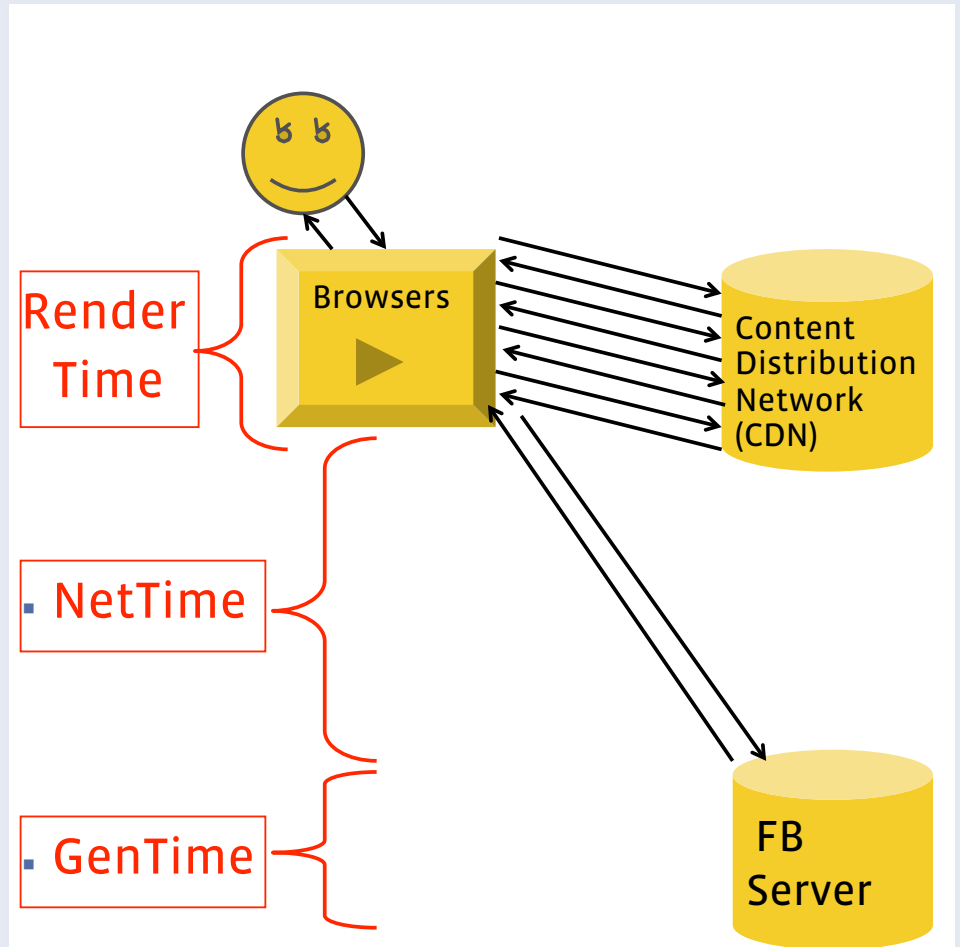
- Agile development
- Deep integration
- Viral adoption
- Heavily interactive

Site speed matters: emerging challenges

- Agile development
- Deep integration
- Viral adoption
- Heavily interactive

Site speed: end-to-end latency experienced by users

- From a user request to the presentation of the page at the browser, interactive:
 - Network Transfer Time
 - Server Generation Time
 - Client Render Time



Site speed: end-to-end latency experienced by users

$$\text{User latency} = \text{RenderTime} + \text{NetTime} + \text{GenTime}$$

- **RenderTime: ~50% of end-user latency**
- **NetTime: ~25% of end-user latency**
- **GenTime: ~25% of end-user latency**

Site speed: end-to-end latency experienced by users

$$\text{User latency} = \text{RenderTime} + \text{NetTime} + \text{GenTime}$$

- **RenderTime: ~50% of end-user latency**
- **NetTime: ~25% of end-user latency**
- **GenTime: ~25% of end-user latency**

Cavalry: Site speed monitoring

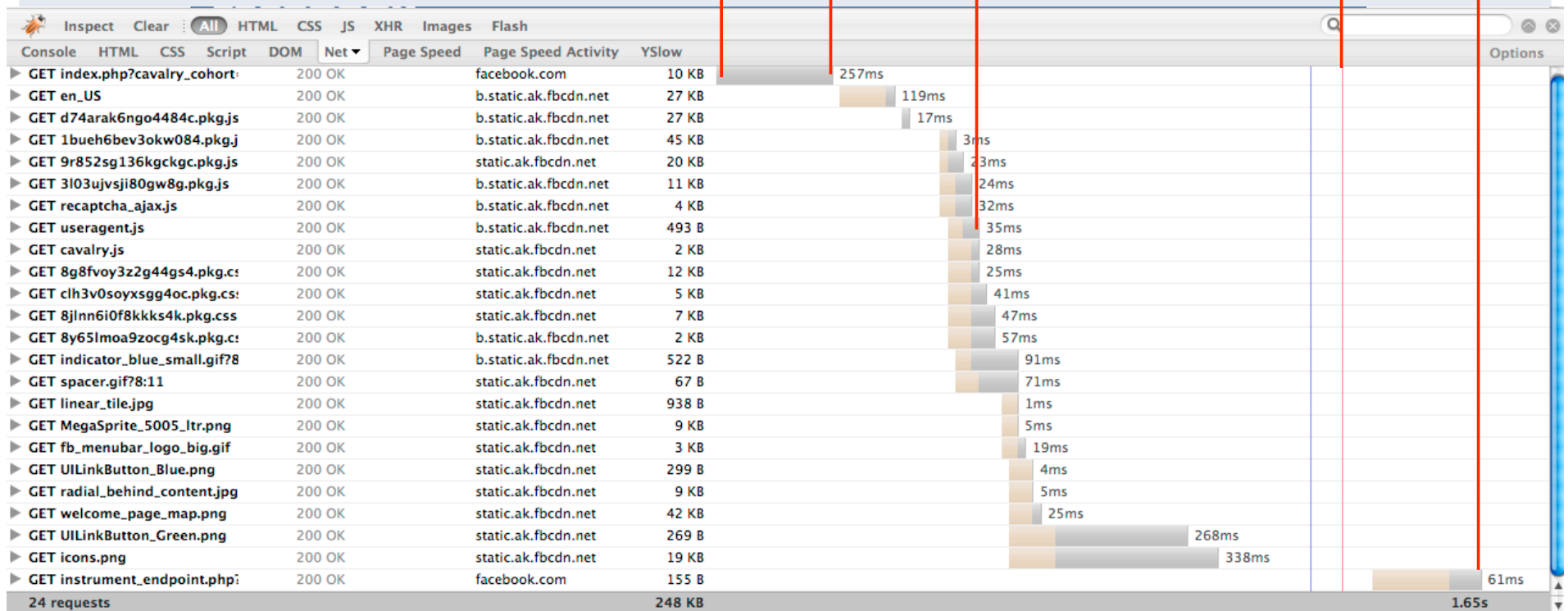
User-based measurement

What's our speed?

- sampling 1/10000 page loads

All content loaded,
Page Interactive

Server
First bytes of HTML
JS
Report



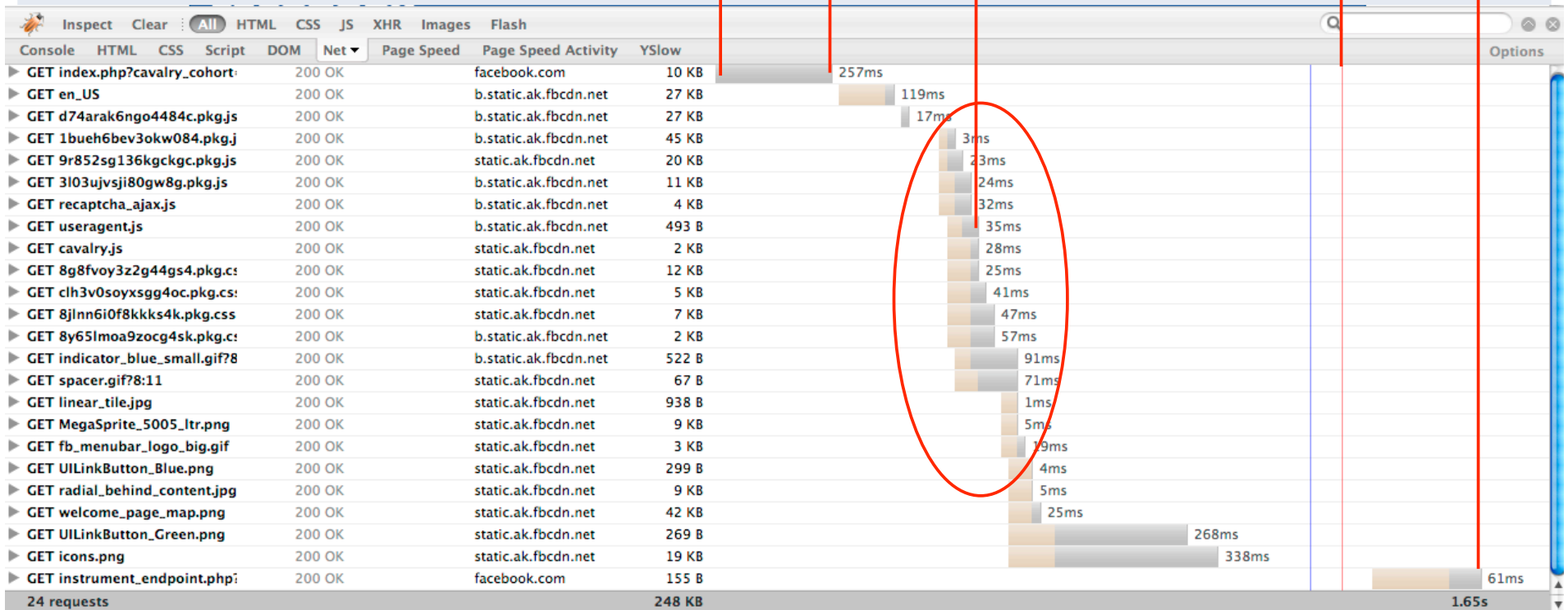
User-based measurement

What's our speed?

- sampling 1/10000 page loads

All content loaded,
Page Interactive

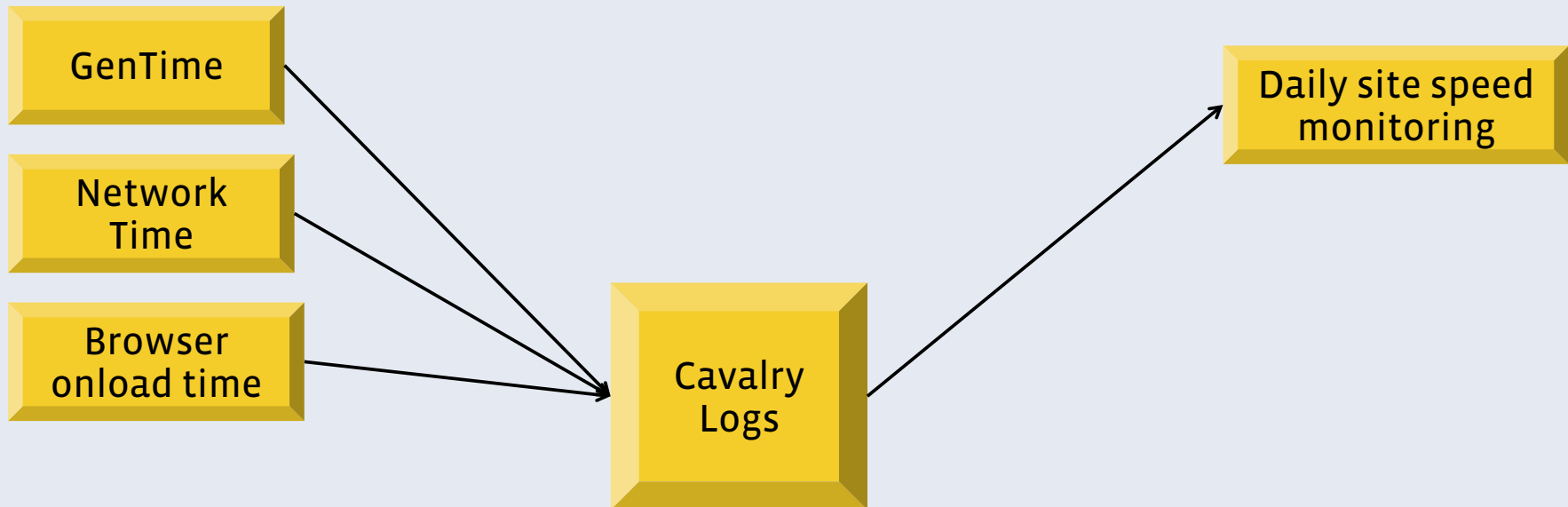
Server
First bytes of HTML
JS
Report



Cavalry: Day-to-day monitoring

What's our speed?

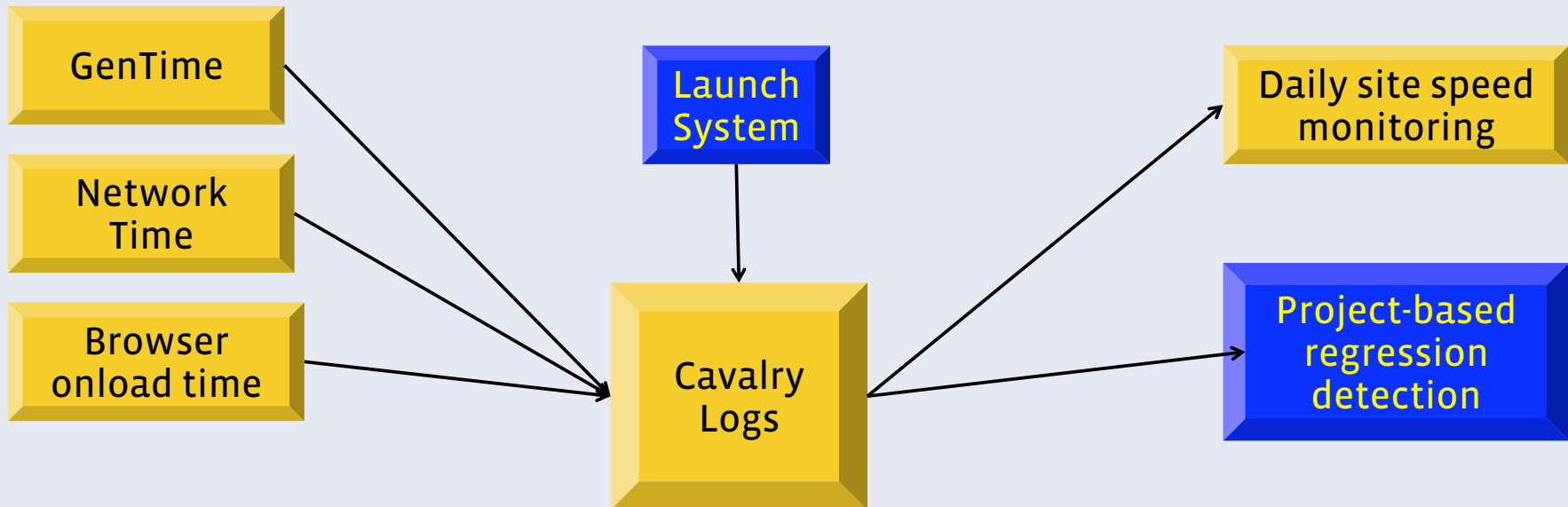
- Collect gen time / network transfer time and render time



Cavalry: Project-based analysis

Who made it faster / slower?

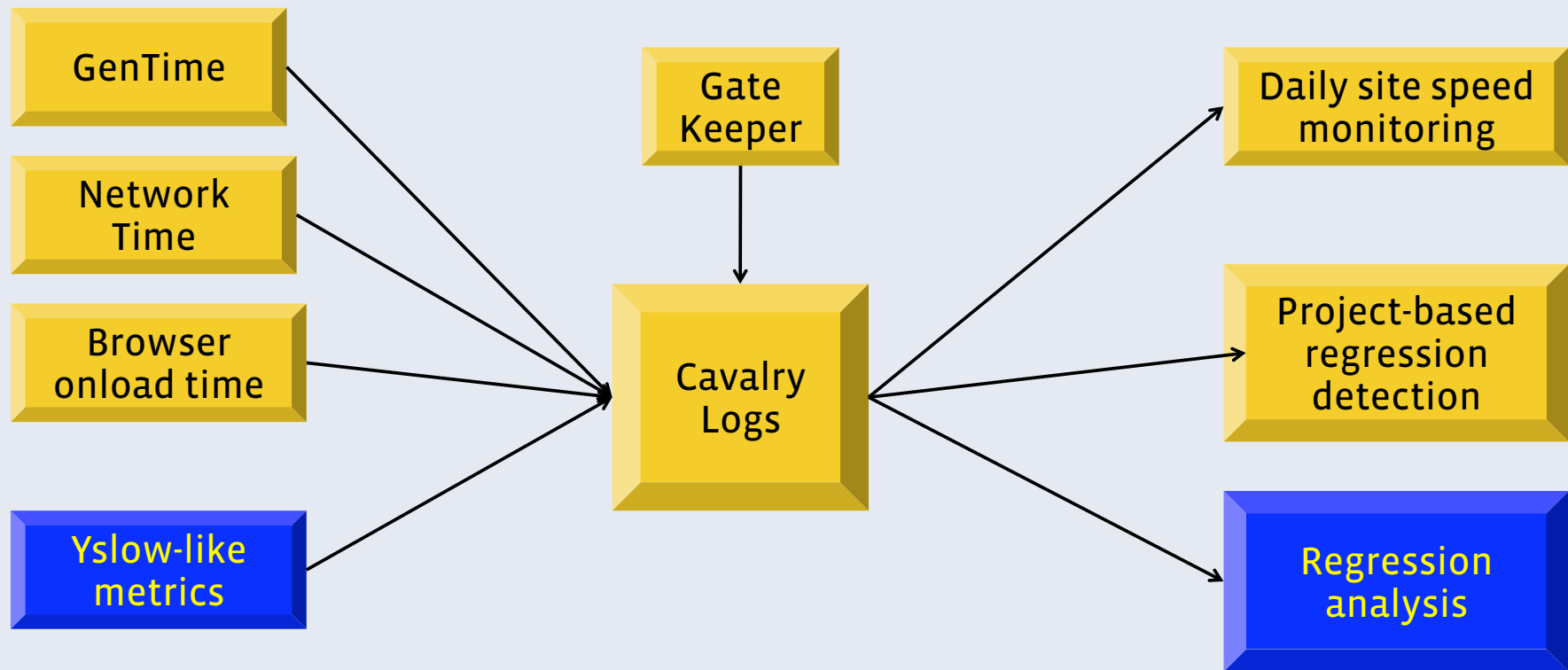
- Integrated with Launch System



Cavalry: Numeric metrics

Why are we fast / slow? How can I fix it?

- YSlow-like technical metrics



“WWW” in performance monitoring:

What? Who? Why?

- User-based measurement: unbiased, representative results
- Feature-launch integration: identify the regression
- Technical metrics: define actionable items for improvement

Haste: Static resource management

Why we need SR Management?

- Day 1: Some smart engineers start a project!

<Print css tag for feature A>

<Print css tag for feature B>

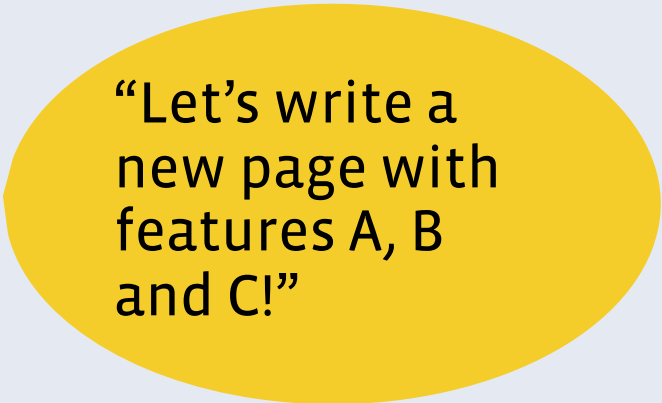
<Print css tag for feature C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

...



“Let’s write a new page with features A, B and C!”

Why we need SR Management?

- Day 2: Some smart engineers run PageSpeed and thinks...

<Print css tag for feature A>

<Print css tag for feature B>

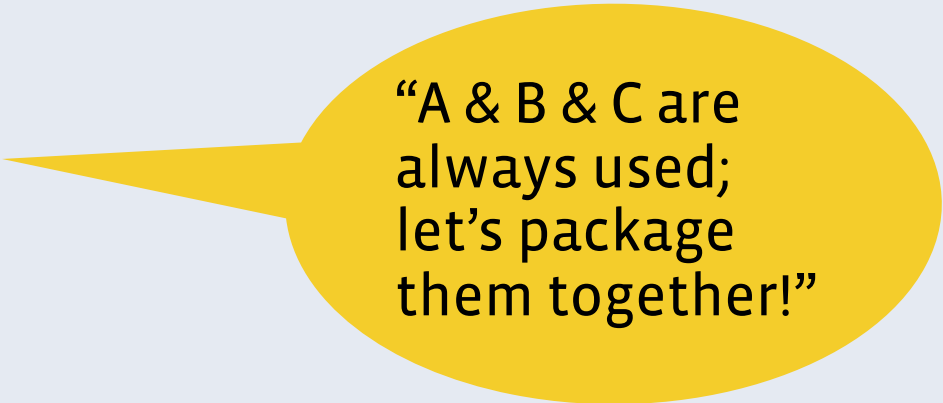
<Print css tag for feature C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

...



“A & B & C are always used; let’s package them together!”

Why we need SR Management?

- Day 2: Awesome!

<Print css tag for feature A&B&C>

<print HTML of feature A>

<print HTML of feature B>

<print HTML of feature C>

...

Why we need SR Management?

- Day 3: feature C evolves...

<Print css tag for feature A & B & C>

<print HTML of feature A>

<print HTML of feature B>

If (users_signup_for_C()) { <print HTML of feature C> }

...

Why we need SR Management?

- Day 3:

<Print css tag for feature A & B & C>

A&B are always used, while C is not. ...

<print HTML of feature A>

<print HTML of feature B>

If (users_signup_for_C()) { <print HTML of feature C> }

...

Why we need SR Management?

- Day 4: feature C is deprecated

<Print css tag for feature A & B & C>

<print HTML of feature A>

<print HTML of feature B>

// no one uses C { <print HTML of feature C> }

...

Why we need SR Management?

- Day 4: we start to send unused bits

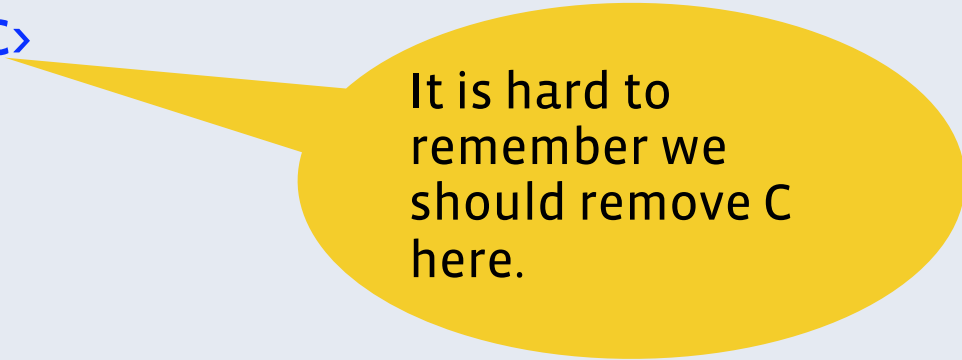
<Print css tag for feature A & B & C>

<print HTML of feature A>

<print HTML of feature B>

// no one uses C { <print HTML of feature C> }

...



It is hard to remember we should remove C here.

Why we need SR Management?

- One months later...

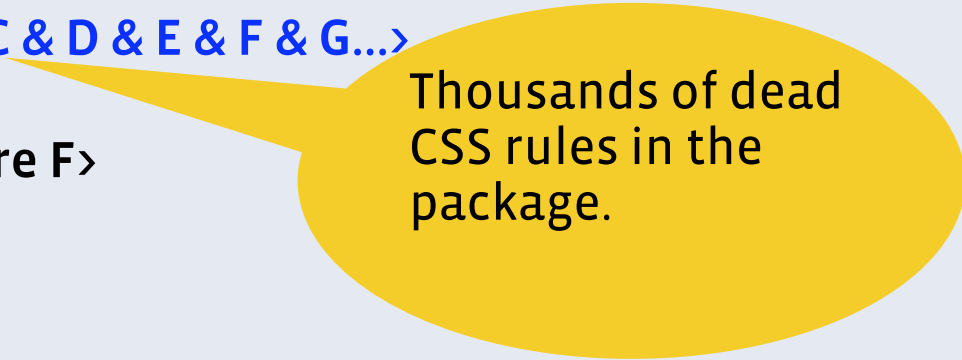
<Print css tag for feature A & B & C & D & E & F & G...>

if (F is used) <print HTML of feature F>

<print HTML of feature G>

if (F is not used) { <print HTML of feature E> }

...



Thousands of dead CSS rules in the package.

Static Resource Management @ Facebook

Challenges:

- Deep Integration
- Viral Adoption
- Agile Development

Responses:

- Separate requirement declaration and delivery of static resources
- Requirement declaration: lives with HTML generation
- Delivery: Globally optimized

Haste: Static Resource Management

Separate *Declaration*
from actual *Delivery*

Back to Day 1:

require_static(A_css); <render HTML of feature A>

require_static(B_css); <render HTML of feature B>

require_static(C_css); <render HTML of feature C>

<deliver all required CSS>

<print all rendered HTML>

Requirement Declaration
lives with HTML

Global Optimization on
Delivery

Haste: Global Optimization

Online process

require_static(A_css); <render HTML of feature A>

require_static(B_css); <render HTML of feature B>

require_static(C_css); <render HTML of feature C>

<deliver all required CSS>

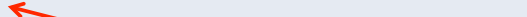
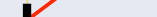
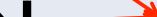
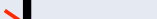
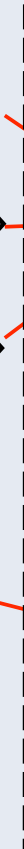
<print all rendered HTML>

Offline analysis

Usage Pattern logs

Clustering algorithms

“Optimal” packages



Haste: Trace-based Packaging

Nov 2008 => May 2009

Date	# of JS files	# of JS bytes	# of pkg at a home.php	# of bytes at a home.php
Nov 2008	461	4.4 MB	29	629 KB
May 2009	729	5.9 MB	14	560 KB

Haste: Trace-based Packaging

Nov 2008 => May 2009

Date	# of JS files	# of JS bytes	# of pkg at a home.php	# of bytes at a home.php
Nov 2008	461	4.4 MB	29	629 KB
May 2009	729	5.9 MB	14	560 KB

- 'js/careers/jobs.js',
- 'js/lib/ui/timeeditor.js',
- 'resume/js/resumepro.js',
- 'resume/js/resumesection.js'

Haste: Trace-based Packaging

Nov 2008 => May 2009

Date	# of JS files	# of JS bytes	# of pkg at a home.php	# of bytes at a home.php
Nov 2008	461	4.4 MB	29	629 KB
May 2009	729	5.9 MB	14	560 KB

Date	# CSS files	# of CSS bytes	# of pkg at a home.php	# of bytes at a home.php
Nov 2008	487	1.7 MB	24	69 KB
May 2009	706	1.9 MB	15	64 KB

Haste: Trace-based Analysis

Potentials for image sprites too!

- Thousands of virtual gifts with static images, which to sprite?



Haste: Trace-based Analysis

Potentials for image sprites too!

- The answer is...



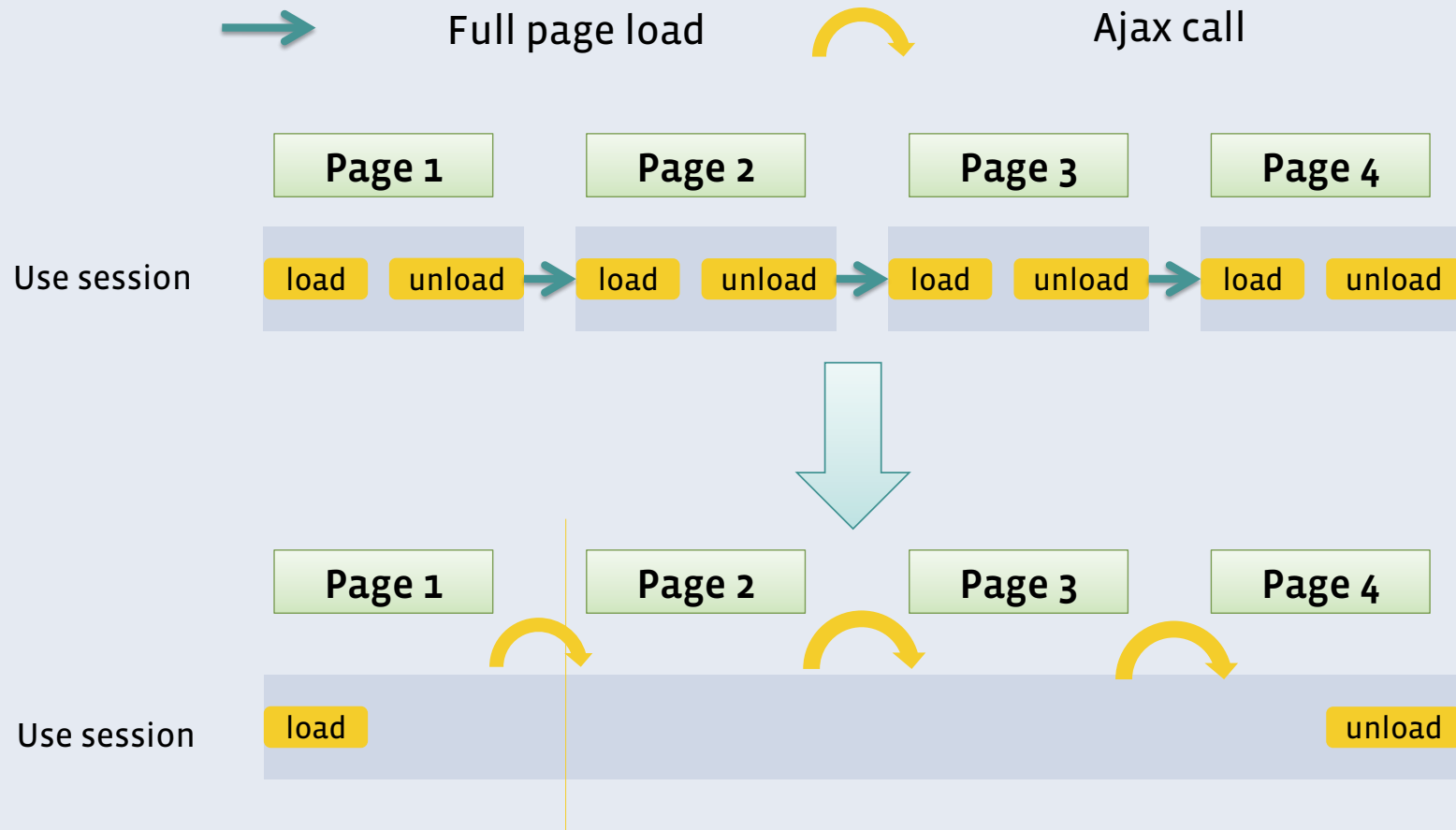
Haste: Trace-based Analysis

Adaptive Performance Optimization

- JS / CSS package optimization
- Guidance for image spriting
- Guidance of progressive rendering

Quickling: Ajaxify the Facebook site

Remove redundant work via Ajax



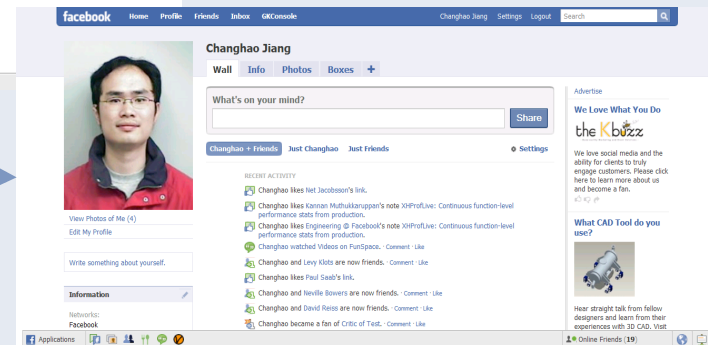
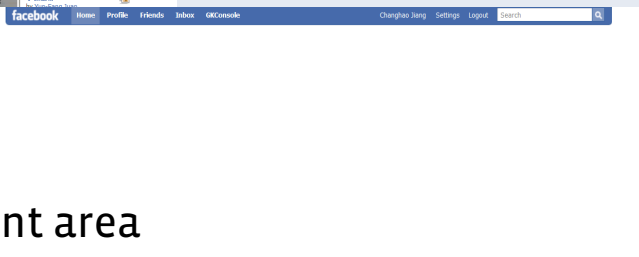
How Quickling works?

1. User clicks a link or back/forward button
2. Quickling sends an ajax to server
3. Response arrives

4. Quickling blanks the content area

5. Download javascript/CSS

6. Show new content



LinkController

Intercept user clicks on links

- Dynamically attach a handler to all link clicks:

```
$(‘a’).click(function() {  
  
    // ‘payload’ is a JSON encoded response from the server  
    $.get(this.href, function(payload) {  
  
        // Dynamically load ‘js’, ‘css’ resources for this page.  
        bootload(payload.bootload, function() {  
  
            // Swap in the new page’s content  
            $('#content').html(payload.html)  
  
            // Execute the onloadRegister’ed js code  
            execute(payload.onload)  
        });  
    });  
});
```

HistoryManager

Enable 'Back/Forward' buttons for AJAX requests

- Set target page URL as the fragment of the URL
 - <http://www.facebook.com/home.php>
 - <http://www.facebook.com/home.php#/cjiang?ref=profile>
 - <http://www.facebook.com/home.php#/friends/?ref=tn>

Bootloader

Load static resources via 'script', 'link' tag injection

```
function requestResource(type, source) {
  var h = document.getElementsByTagName('head')[0];
  switch (type) {
    case 'js':
      var script = document.createElement('script');
      script.src = source;
      script.type = 'text/javascript';
      h.appendChild(script);
      break;
    case 'css':
      var link = document.createElement('link');
      link.rel = "stylesheet";
      link.type = "text/css";
      link.media = "all";
      link.href = source;
      h.appendChild(link);
      break;
  }
}
```

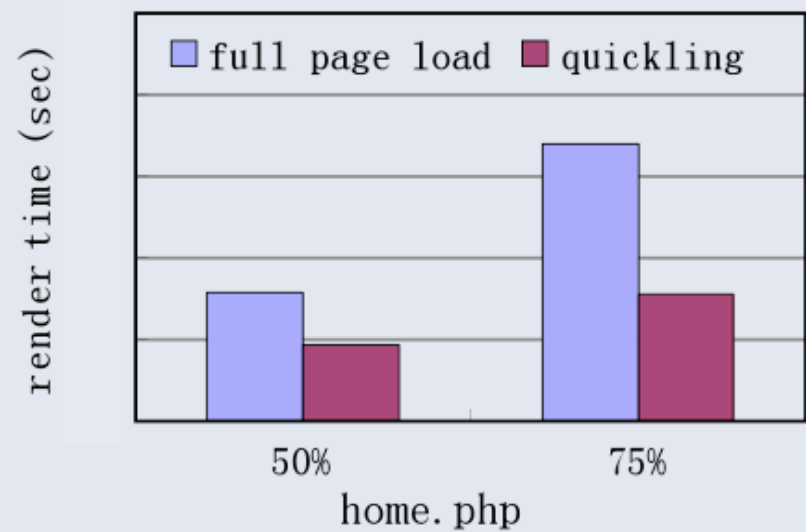
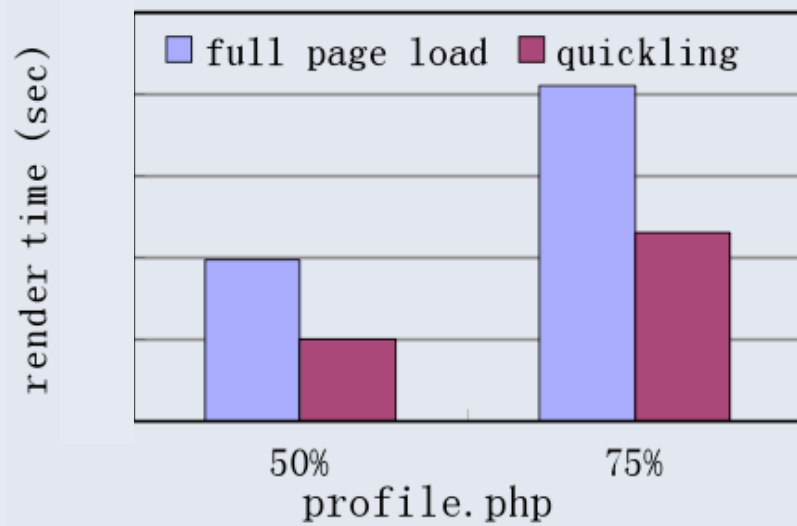
Other details

- All pages now share a single global javascript scope:
 - Explicitly reclaim resources or reset states before leaving a page
 - Stub out setTimeout and setInterval
- All CSS rules will be accumulated
 - Name-spacing CSS rules with page-specific information
- Busy indicator
 - iframe transport
- Permanent link
 - prelude inlined js code to redirect if necessary

Current status

- Turned on for FireFox and IE users: (>90% users)
- ~60% of page hits to Facebook site are Quickling requests

Performance improvement



40% ~ 50% reduction in render time

PageCache: Cache visited pages at client side

PageCache

Cache user visited pages in browsers

- Motivation:
 - A typical user session:
 - *home -> profile -> photo -> home -> notes -> home -> photo -> photo*
 - Some pages are likely to be revisited soon (temporal locality)
 - Home page visited every 3 ~ 5 page views
 - Back/Forward button

How PageCache works?



1. User clicks a link or back button

2. Quick Page is sent to the server

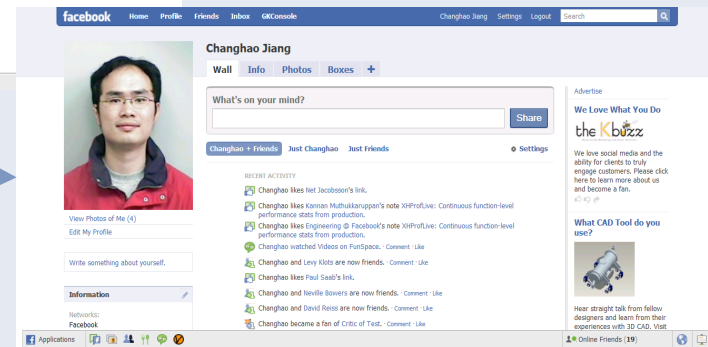
3. Response arrives

3.5 Save response in cache

4. Quickling blanks the content area

5. Download javascript/CSS

6. Show new content



Cache consistency 1: Incremental updates



Cached version



Restored version

Cache consistency 1: Incremental updates

Poll server for incremental updates via ajax calls.

- Allow registering javascript functions to be called right before cached page is shown.
- Used by home page to refresh 'ads', fetch latest stories



Cached version



Restored version

Cache consistency 2: In-page writes

A screenshot of a Facebook post from Doug Beaver. The post features two images of a red car. Below the images, there are two comments: one from Liz Doughty Tan and one from Kimberly Algeri-Wong. The post is labeled as a 'Cached version' in a green box at the bottom.



A screenshot of the same Facebook post, but in its 'Restored version'. The content is identical to the cached version, but a new comment from Changhao Jiang has been added and is highlighted with a red border. The post is labeled as a 'Restored version' in a green box at the bottom.

Cache consistency 2: In-page writes

Record and replay

- Automatically record all state-changing operations in a cached page
- Automatically replay those operations when cached page is restored.

The screenshot shows a Facebook post by Doug Beaver with two photos of a red car. Below the photos are comments from Yishan Wong, Liz Doughty Tan, and Kimberly Algeri-Wong. The post is labeled as 'Cached version' in a green box at the bottom.

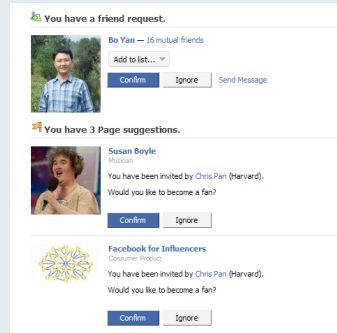


The screenshot shows the same Facebook post as the cached version, but with a new comment from Changhao Jiang: "haha, me too!". This comment is highlighted with a red box. The post is labeled as 'Restored version' in a green box at the bottom.

Cache consistency 3: Cross-page writes



Cached version



State-changing op



Restored version

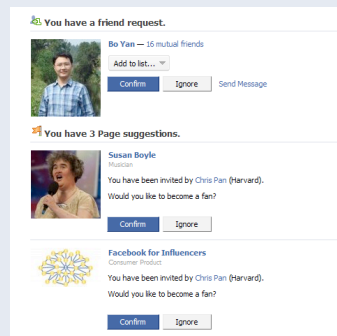
Cache consistency 3: Cross-page writes

Server side invalidation

- Instrument server-side database access API, whenever a write operations is detected, send a signal to the client to invalidate the cache.



Cached version



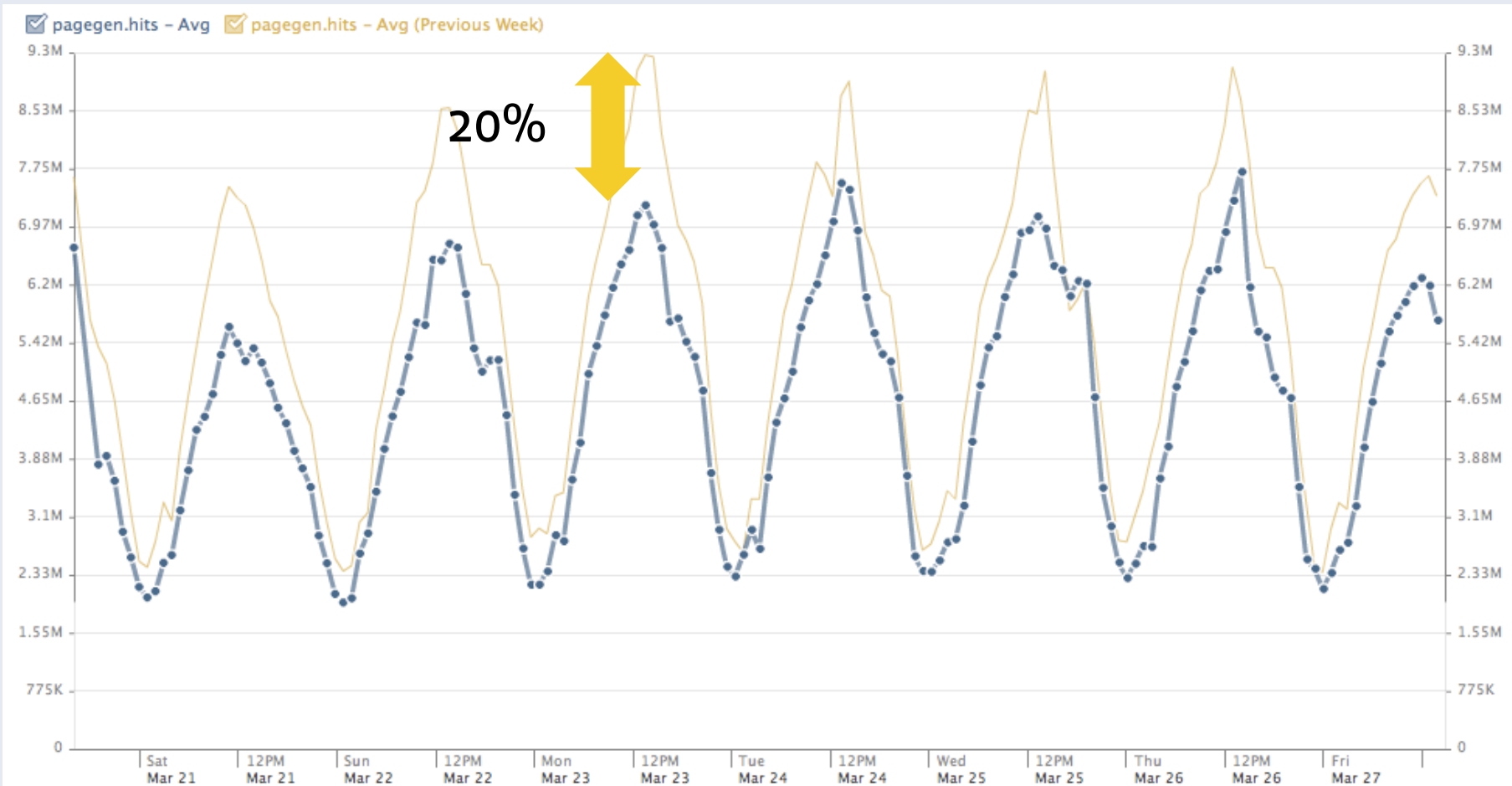
State-changing op



Restored version

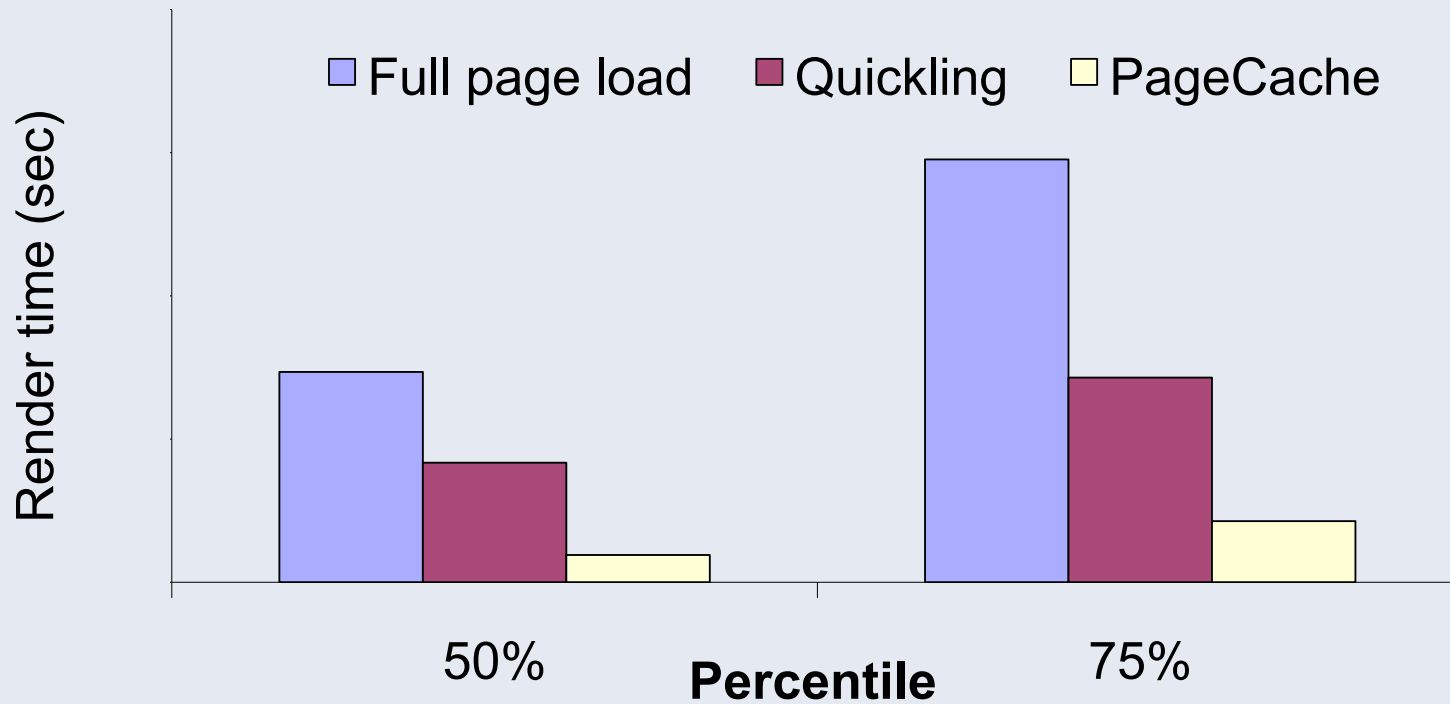
Current status

- Deployed on production
- Only cache in memory
- Only turned on for home page



~20% savings on page hits to home page

Performance improvement



3X ~ 4X speedup in render time vs Quickling

Summary

Summary

- Performance monitoring: What, Who, and Why (“WWW”)
- Static resource management: Adaptive to fast evolution
- Ajaxify the website.
- Client side caching of user visited pages

Thank you!