Arquitectura de Sistemas de Software

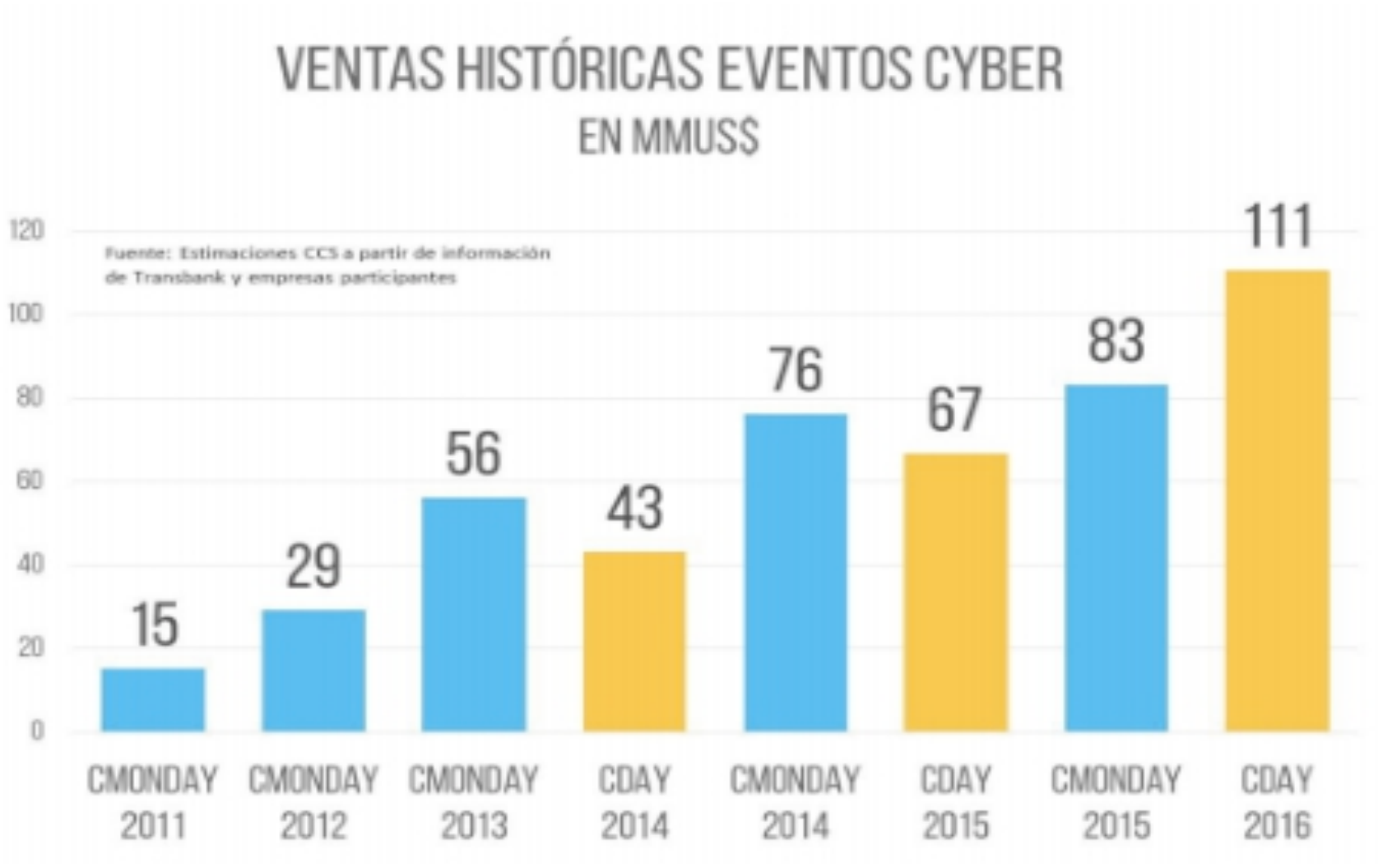
Departamento de Ciencia de la Computación
Escuela de Ingeniería – PUC
Hans Findel {hjfindel@uc.cl}

Performance

Adopción


NUMBER OF YEARS IT TOOK FOR EACH PRODUCT TO REACH 50 MILLION USERS
Automobile
62 years
Telephone
50 years
Electricity
46 years
Credit Card
28 years
Television
22 years
ATM
18 years
Debit Card
12 years
Internet
7 years
PayPal
5 years
YouTube
4 years
Facebook
3 years
Twitter
2 years

Compras


VENTAS HISTÓRICAS EVENTOS CYBER
EN MMUS$
Fuente: Estimaciones CCS a partir de información de Transbank y empresas participantes
120
100
80
60
40
20
0
15
29
56
43
76
67
83
111
CMONDAY 2011
CMONDAY 2012
CMONDAY 2013
CDAY 2014
CMONDAY 2014
CDAY 2015
CMONDAY 2015
CDAY 2016

Performance

- Es una atributo de calidad observable cuando el sistema está en ejecución (dinámica)
  - Igual que seguridad, disponibilidad, y usabilidad
- Propiedades no observables en tiempo de ejecución (estática):
  - Modificabilidad, portabilidad, reusabilidad, integrabilidad, “testeabilidad”
- Capacidad del software para dar una respuesta
  - Depende de la comunicación e interacción entre componentes del sistema

Performance

- Cómo medir performance
  - Experimental:
    - Mediciones de requests, transacciones, etc…
  - Teórico:
    - Modelo estocástico de colas del sistema basado en escenarios de carga de trabajo
      - Técnica analítica
        - Tasa de llegada y distribución de solicitudes de servicios, tiempos de procesamiento, tamaños de colas, y latencia
      - Simulación

Estrategias para mejorar el desempeño

- Considerar el hardware
- Considerar un “optimizador” automático de código
  - JVM
- Usar caches
- Haz menos!
  - Evita correr código innecesario
  - Evita instanciar objetos innecesarios
  - Evita abstracciones innecesarias
  - Toma el camino más corto posible (reduce el stack!)

Hardware

- Lectura de memoria
  - CAS Latency: Tiempo de transferencia entre el momento en que el controlador de memoria indica al módulo de memoria acceder a una posición de la RAM y el momento en que la data esta disponible (ha sido leída)

Memory timing examples (CAS latency only)									
Generation	**Type**	**Data rate**	**Bit time**	**Command rate**	**Cycle time**	**CL**	**First word**	**Fourth word**	**Eighth word**
SDRAM	PC100	100 MT/s	10 ns	100 MHz	10 ns	2	20 ns	50 ns	90 ns
	PC133	133 MT/s	7.5 ns	133 MHz	7.5 ns	3	22.5 ns	45 ns	75 ns
DDR SDRAM	DDR-333	333 MT/s	3 ns	166 MHz	6 ns	2.5	15 ns	24 ns	36 ns
	DDR-400	400 MT/s	2.5 ns	200 MHz	5 ns	3	15 ns	22.5 ns	32.5 ns
						2.5	12.5 ns	20 ns	30 ns
						2	10 ns	17.5 ns	27.5 ns

Hardware

- Operaciones de punto flotante (lentas) Vs. operaciones con enteros
  - Depende del tipo de operaciones, de la arquitectura del CPU (a veces se hacen en paralelo), etc…
- ***Disco duro***
- CPU (reloj)
- RAM

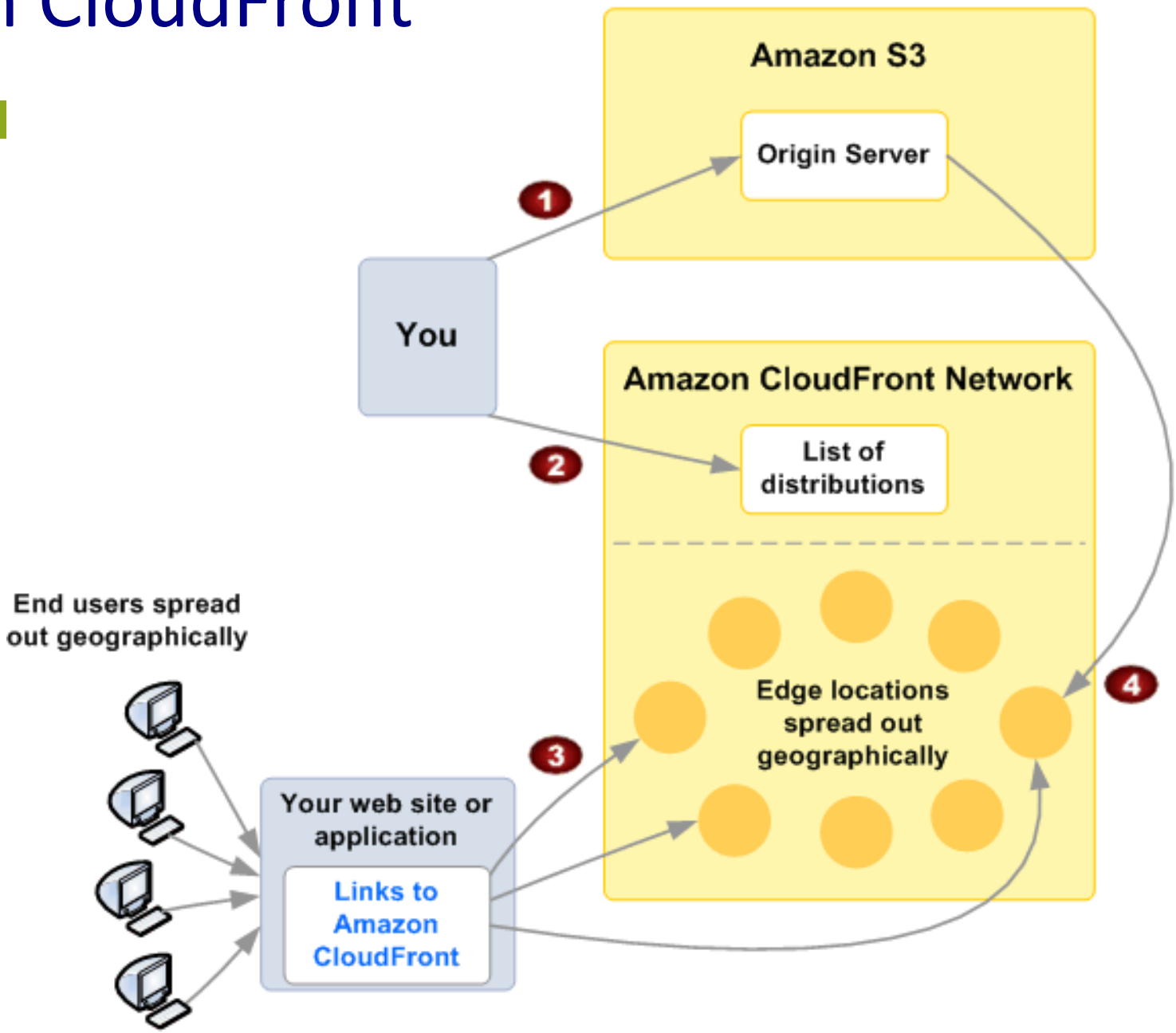
Hardware

- Ancho de banda
  - Tráfico de internet:
    - Streaming video/audio, protocolos codiciosos (bit torrent), etc…
  - Incluso en cloud computing
    - Replicación y redundancia agrega los retrasos en las lecturas/***escrituras*** en disco
    - Aumenta el consumo de ancho de banda
    - Peor si es sobre Internet! Con datos masivos! (ej. salud)

Cache

- En lugar de descargar contenido desde un servidor remoto, los navegadores lo descargan de un servidor geográficamente cercano o conectado por un enlace de red rápida (corto)
  - Almacenamiento en caché del navegador
    - Copias cercanas y rápidas, pero descargadas por un sólo usuario
  - Almacenamiento en caché de una red (Proxy cache, interception proxy)
    - Copias cercanas por cada usuario que descargó el contenido (ahorra ancho de banda)
  - Almacenamiento en caché de una red de distribución de contenido (CDN, reverse proxy, gateway cache, surrogate cache)
    - Usado por empresas (Webmasters) que quieren que su contenido sea accesible rápidamente en todas partes
    - Akamai

Amazon CloudFront



Performance: Cache

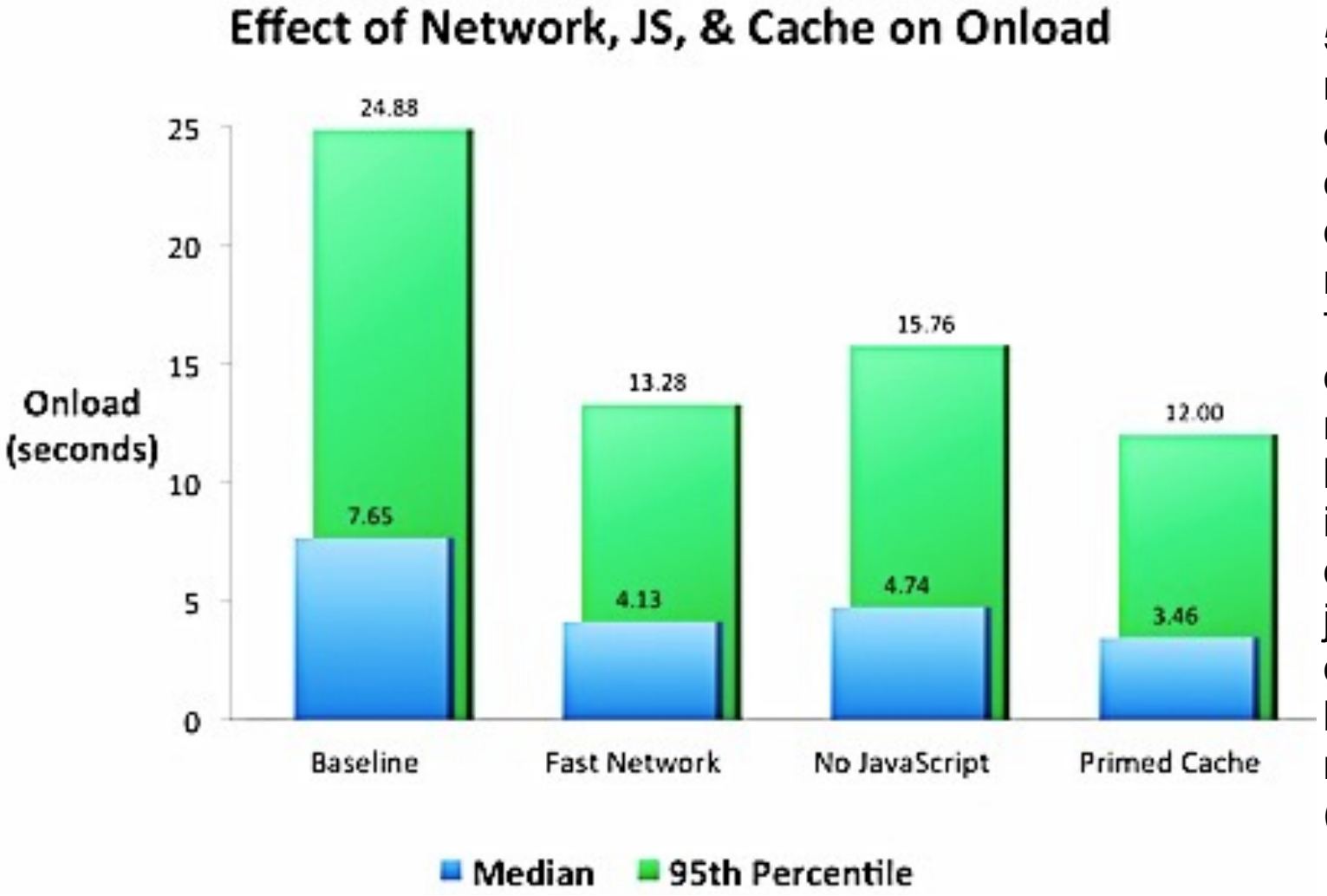
- http://www.stevesouders.com/blog/2012/10/11/cache-is-king/

- **Ajax:** Prefetching hace que la lectura de las páginas siguientes sea más rápida (la primera es más lenta)
- **Optimizar Javascript:** Hace que la primera página y las siguientes sean más rápidas
- **Cache:** Está en el medio, la primera visita real no es rápida pero las siguientes si, incluso se siente en la siguiente visita (después de cerrar el browser)

Performance

- Baseline:
  - WebPageTest en Alexa Top 1.000 con IE9 y conexión DSL simulada: 1.5Mbps down, 384 Kbps up, 50ms RTT)
  - Test: tiempo de carga a la primera visita (cache vacío)
- Red Rápida:
  - Igual, conexión FIOS simulada: 20Mbps down, 5Mbps up, 4ms RTT
- No Javascript:
  - Baseline + noscript (ignorar HTTP requests generados por Javascript)
- Primed Cache:
  - Baseline, pero solo se ven las visitas a vistas cacheadas

Performance (window.onload)



59% de los requests estaban configurados con cache y max-age > 0 Tiempo de cache 10mins o menos, como la lectura fue inmediata, el cache entro en juego completamente bajando los 90 requests a 53 (59%)

Cómo manejar los Caches de la Web?

- Mike Nothingham
  - Chief Architect Akamai
  - http://www.mnot.net/cache_docs/

- Objetivo del cache:
  - Reducir latencia
  - Reducir tráfico de red
- Efectos secundarios:
  - “Ocultan” el tráfico real de un sitio Web
  - Sirven contenido añejo (*stale)*

Reglas de los Caches

- HTTP 1.0 / 1.1

1. Si el response Header dice No-Cache, no se cachea
2. Si el request es autenticado o seguro (HTTPS), no se cachea
3. Una representación cacheada es *fresca* (se sirve del cache) si:
   - Tiene fecha de expiración o header Age-Control < actual
   - El cache ha visto la representación hace poco y fue modificada hace tiempo (Last-Modified)
4. Si una representación es añeja (*stale*), se pedirá al servidor de origen *validarla*
5. Bajo ciertas circunstancias (offline) se servirá la respuesta añeja sin chequear con el servidor de origen

¿Cómo?

- HTML Headers: no es una buena idea (sólo afecta a algunos browsers)
- HTTP Headers

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

- **Expires** (Response):
  - Afecta a todos los caches (pero Cache-Control tiene precedencia)
  - Fecha absoluta, última fecha de acceso, o fecha de última modificación
  - Sincronización de relojes
  - Problema: Si no se actualiza el Expires, hay un aumento de visitas!

Cache-Control (Request / Response)

- El ISP (Internet Service Provider) debe poder soportarlo
- Atributos:
  - max-age=[seconds] por cuánto tiempo desde el último request estará fresco?
  - s-maxage=[seconds] igual a max-age pero para caches compartidos (ej. Proxy)
  - public marca a respuestas autenticadas como cacheables
  - private los caches de usuario (browser) pueden cachear la respuesta, los caches compartidos no
  - no-cache fuerza a los caches a enviar el request al servidor de origen
  - no-store los caches no deben guardar una copia de la representación
  - must-revalidate fuerza al cache a validar con el servidor de origen si la copia está fresca
  - proxy-revalidate igual a must-revalidate pero con los proxy caches

Validadores

- Last-Modified (Response) Tiempo de última modificación
- If-Modified-Since (Request) se usa para preguntar al servidor de origen si la copia se ha modificado desde esa fecha
- ETag (Response) (HTTP 1.1) Identificador de versión
  - If-Match (Request) Efectuar una acción si coincide ETag
  - If-None-Match (Request) Aceptará una respuesta *304 Not Modified* si el contenido no ha cambiado

Diseñando sitios Web sensibles al cache

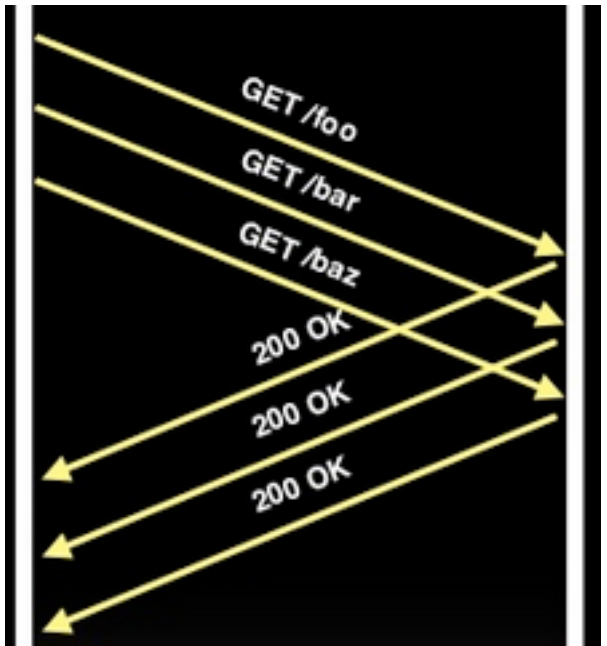
- Usar URLs de manera consistente
- Usar librerías de imágenes comunes
- Marcar imágenes y páginas que no cambian a menudo con *Cache-Control : max-age* y un valor grande
- Ajustar el valor de *max-age* o *Expires* apropiado para los otros recursos
- No actualice archivos de manera innecesaria (falso *Last-Modified*)
- Usar cookies cuando sea necesario (son difíciles de cachear, sólo usar en páginas dinámicas)
- Minimice SSL
- Valide sus páginas con http://redbot.org/

HTTP2.0

- http://www.slideshare.net/mnot/what-http20-will-do-for-you
- IETF HTTPbis WG
  - Clarificar RFC2616 (HTTP 1.1)
  - Roy Fielding, Julian Reschke
- SPDY, Google
  - Firefox 1.1, Nginx, Akamai, Chrome
- HTTPbis -> HTTP/2.0 basado en SPDY

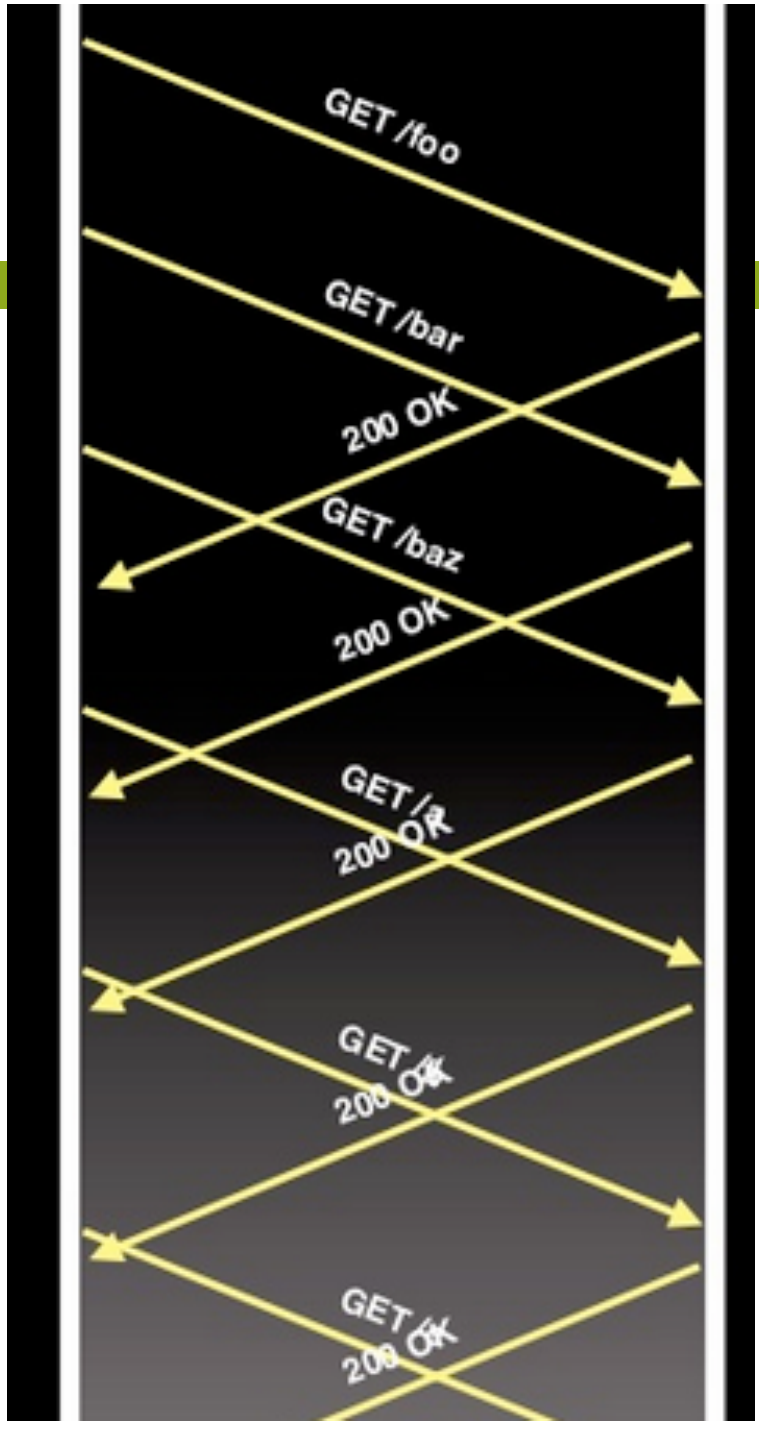
HTTP/2.0 multiplexing

- HTTP/1.0 Un request por conexión TCP !
- HTTP/1.0 con *Connection: keep-alive*
  - Se pueden hacer varios requests uno después de otro
  - Minimiza el costo de abrir conexiones
  - Pero es *bloqueante* 1 *outstanding* request a la vez
- HTTP/1.1 con Pipelining
  - Request múltiples, ordenados
  - La respuesta, sin embargo, sigue siendo bloqueante (Head Of Line, HOL) y ordenada (FIFO)
  - Browsers mantienen 4-8 conexiones



SPDY multiplexing

- Una conexión
- Varios requests
- Priorización
- Respuestas fuera de orden
- Entrelazadas



Header compression

- HTTP headers en una conexión abierta (incluso multiplexada) son muy similares
  - Request URI
  - User-Agent
  - Cookies
  - *Referer*

- Server push!!!

NodeJS

Node.js

- Servidores Web altamente escalables
- Motor de Javascript server-side: Google V8 + libs
- Ryan Dahl, 2009
- Objetivo: Sitios Web con push

  - Escritos en Javascript
  - Guiado por ***eventos,*** I/O ***asíncrono***, ***no bloqueantes***
  - Conexiones con alta concurrencia
  - No multi-threads: Un sólo thread que maneja loop de eventos + 1 cola

Node.js - Arquitectura

Javascript			C/C++	
Librería estándar de Node				
Bindings node (socket, http, etc.)				
V8	Pool de threads (libelo)	Loop de eventos (libdev)	DNS (c-ares)	crypto (OpenSSL)

Ejemplo

```
var http = require('http');

http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello World\n');
}).listen(8000);

console.log('Server running at http://localhost:8000/');
```

Quién usa Node?

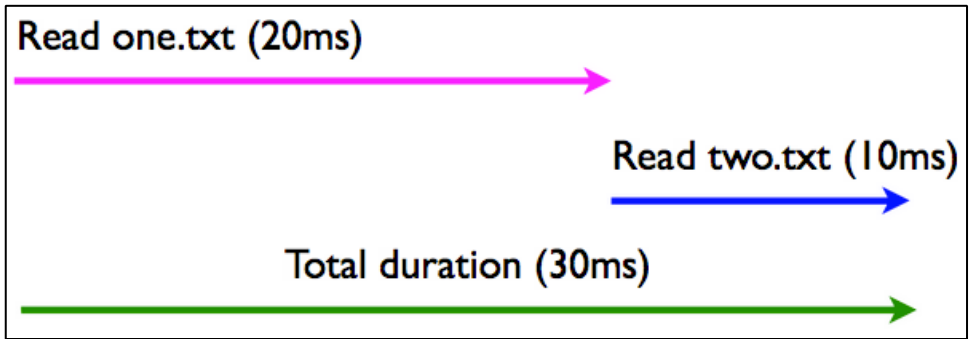
- Linkedin (Mobile Web App 20x faster)
- Pinterest
  - http://highscalability.com/blog/2012/2/16/a-short-on-the-pinterest-stack-for-handling-3-million-users.html
- Ebay (Data retrieval gateway)
- GitHub (downloads)
- Palm/HP (WebOS)
- Yahoo! Mail
- Rackspace
- …

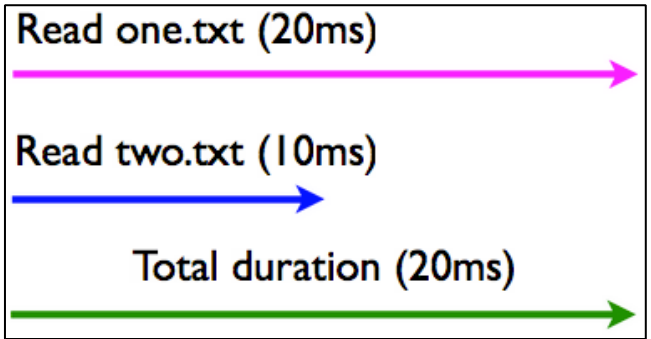
I/O No Bloqueante

read_file_async.js

```
var fs = require('fs');
fs.readFile('one.txt', 'utf-8', function(err, data) {
  console.log('Read file one');
});
fs.readFile('two.txt', 'utf-8', function(err, data) {
  console.log('Read file two');
});
```

```
$ node read_file_async.js
Read file two
Read file one
```





Librerías / aplicaciones interesantes

- Web sockets
  - HTML5 Web Sockets
  - Conexiones persistentes entre browser/server
  - Push
  - Socket.io
    - WebSocket
    - Adobe Flash Socket
    - AJAX long polling
    - AJAX multipart streaming
    - Forever Iframe
    - JSONP Polling

Librerías / aplicaciones interesantes

- Streaming
  - Readable, writable
- Zombie.js
- npm
- +Express, Restify (CRUD)
- + redis (DB muy rápida)
  - NoSQL, in-memory, key/value, replicación master/slave, pub/sub
- Mobile
  - SenchaTouch2 + node.js + socket.io

Ventajas / Desventajas

- Ventajas:
  - Aplicaciones intensas en I/O
    - 6000 http requests x sec x core CPU (1 kb response)
    - Miles de conexiones concurrentes activas
    - V8 -> Javascript to “Assembler”
  - Websockets/Push
  - Proxy de data streams
  - Backend de aplicaciones Web simples (una página)
  - Spawn de otros programas
  - Paralelización de I/O

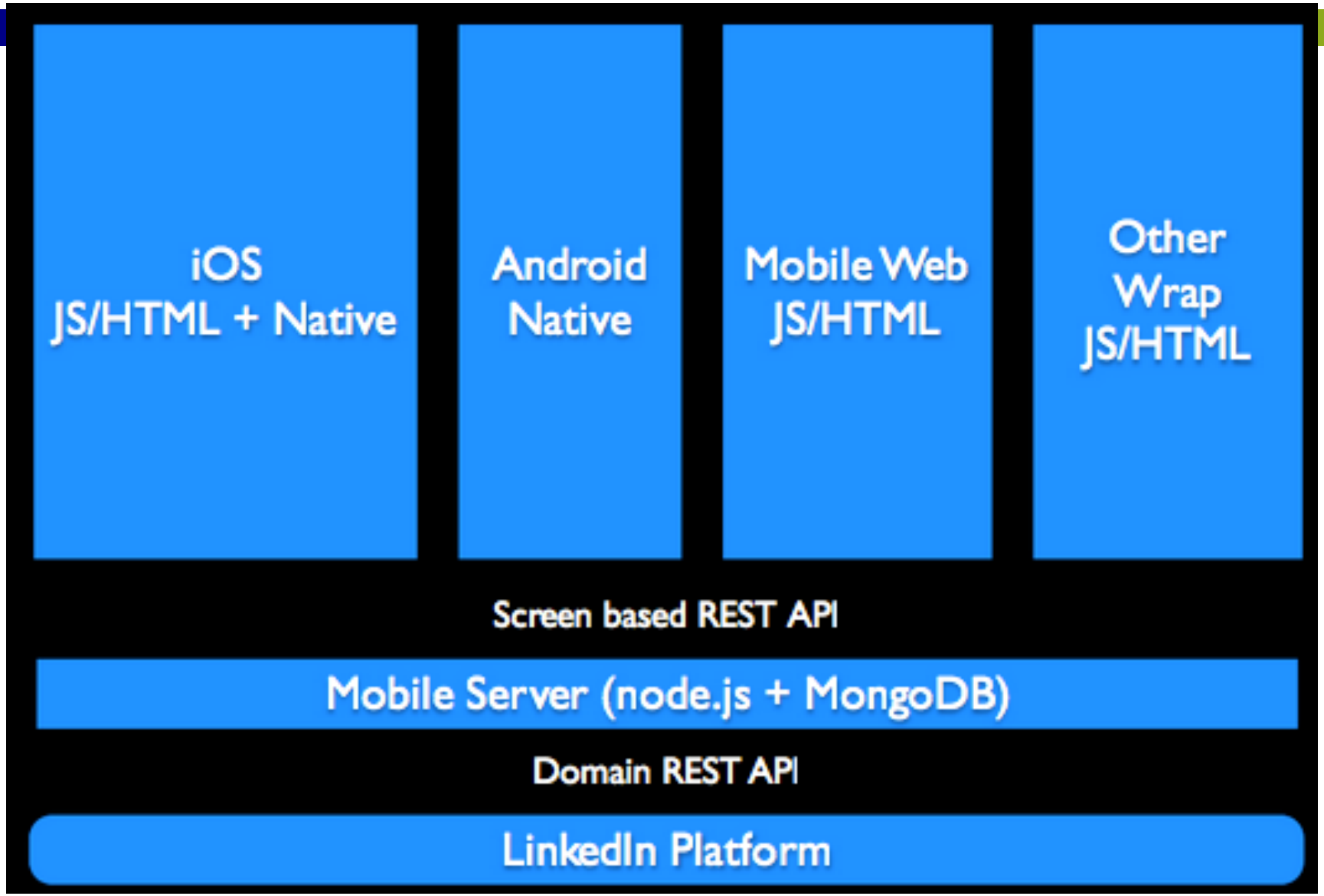
Ventajas / Desventajas

- Desventajas:
  - Sistemas de tiempo real estricto
  - Procesamiento de números
  - Datasets grandes en memoria (V8)
  - No es bueno con aplicaciones intensas en CPU

Caso1: Linkedin Mobile
De Rails a Node: 20x faster, 30 servers a 3

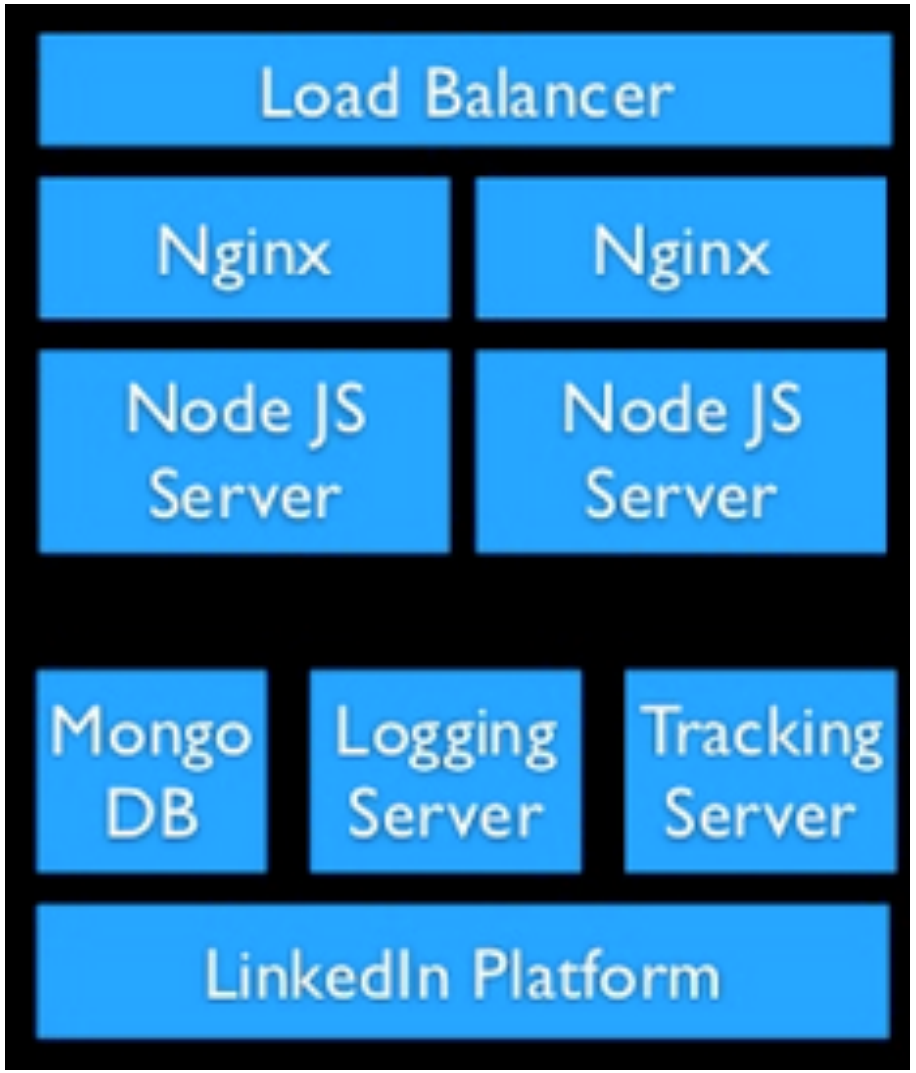
- Aprovechar la plataforma
  - 70/60% HTML (look & feel, texto, imágenes, layouts)
  - 30% nativo (iOS), 40% (Android)
    - Listas (largas) de contenido, scroll
    - Listas infinitas (dinámicas)
- Usar Websockets
  - Comunicación HTML – nativo, muy rápido, inestable
- Servidor HTTP (REST) en el mobile: Node.js
  - Seguridad: Sólo localhost
  - Background? Suspendido

Linkedin Mobile: Arquitectura



Linkedin Mobile Server

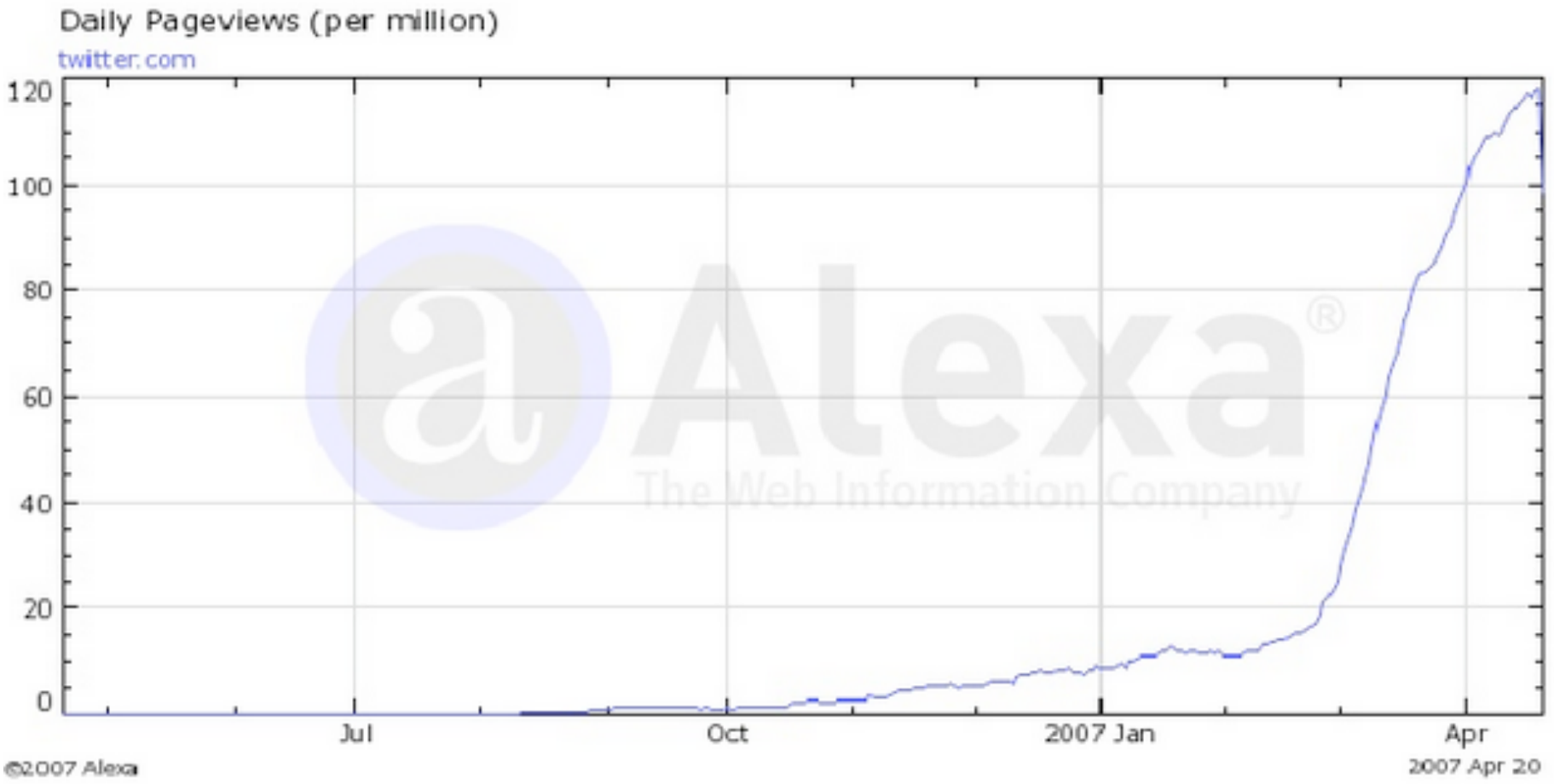
- Stateless
- Transporte: JSON (in/out)
- NGINX como servidor
- CDN para contenido estático
- Logs! Trackers!



Caso2: Twitter

- 2007
  - 600 req x sec
  - 180 Rails (Mongrel)
  - 1 DB (MySQL) + 1 slave
  - 30 procesos misceláneos
  - 8 Sun X4100s
  - Creciendo rápido!

twitter


Daily Pageviews (per million)
twitter.com
120
100
80
60
40
20
0
Jul
Oct
2007 Jan
Apr
©2007 Alexa
2007 Apr 20

Caso2: Twitter

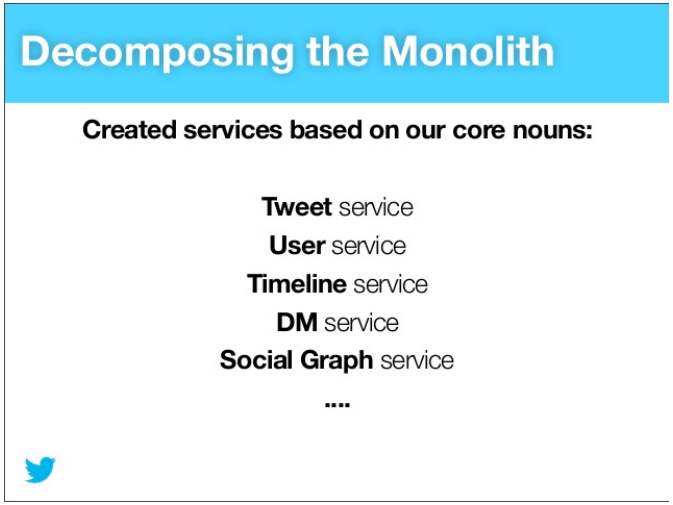
- 2015
  - 500.000.000 Tweets/día (~600.000 Tweet/s [10x 2006])
    - 2013 record: 143,199 TPS (https://blog.twitter.com/2013/new-tweets-per-second-record-and-how)
  - 100+ Proyectos de código abierto (open source)
  - 230M+ Usuarios activos
  - 75+ Lenguajes soportados

Caso 2 - Twitter

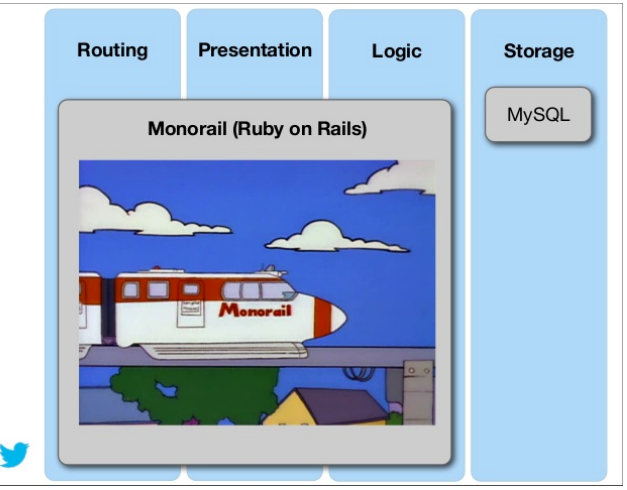
- Inicialmente: Gran aplicación con Rails
- Problemas:
  - Lanzando más máquinas a problema, en vez de dar una solución de ingeniería
  - Optimización de código vs legibilidad y flexibilidad
  - Recursos de la máquinas siendo el cuello de botella

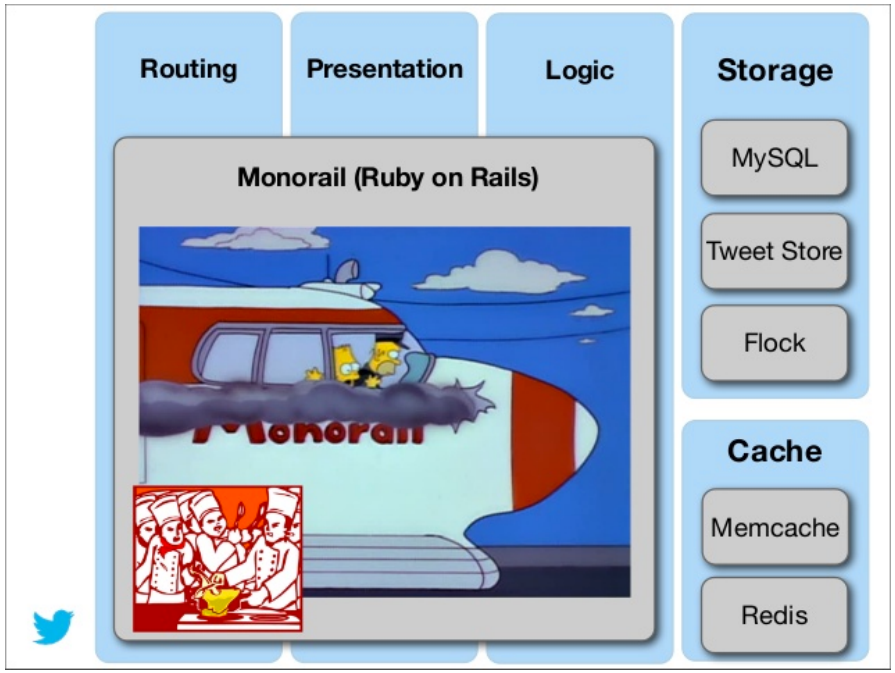
Caso 2 - Twitter

- Cambio de Ruby VM a JVM
  - Search (Java via Lucene)
  - FlockDB (Social Graph - Scala)
- Buena experiencia - migrar todo a servicios aislados
  - Tweet Service
  - User Service
  - Timeline Service
  - DM Service
  - Social Graph Service
  - …

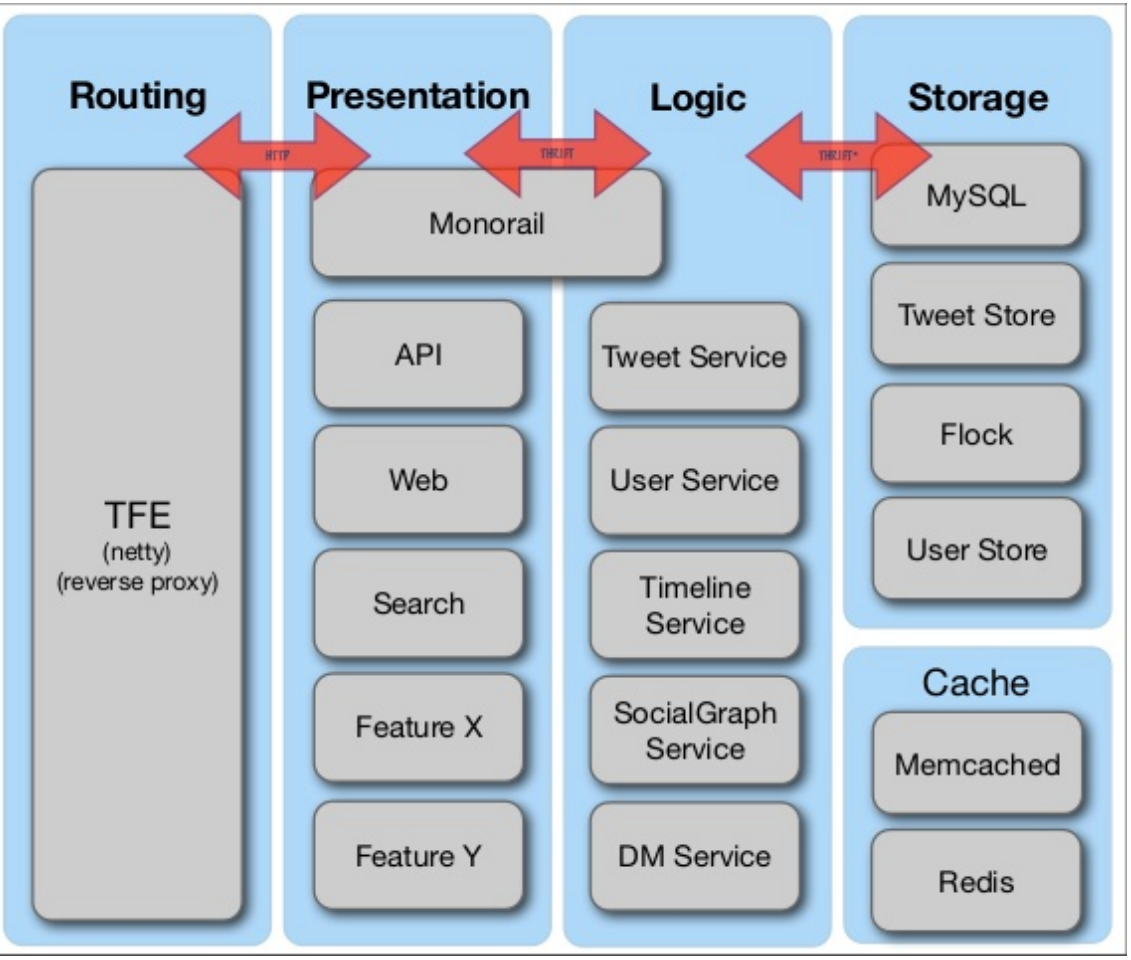


Caso 2 - Twitter





Caso 2 - Twitter



Borg and The Birth of Mesos

Google was generations ahead with Borg/Omega

"The Datacenter as a Computer"

http://research.google.com/pubs/pub35290.html (2009)

engineers focus on resources needed; mixed workloads possible

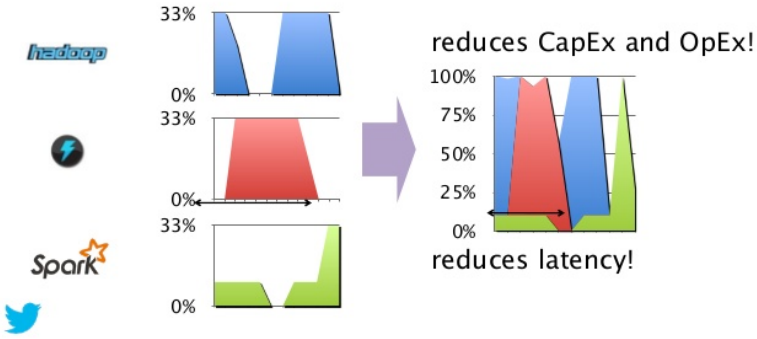
Learn from Google and work w/ university research!

http://wired.com/wiredenterprise/2013/03/google-borg-twitter-mesos

Data Center Computing

Reduce CapEx/OpEx via efficient utilization of HW

http://mesos.apache.org

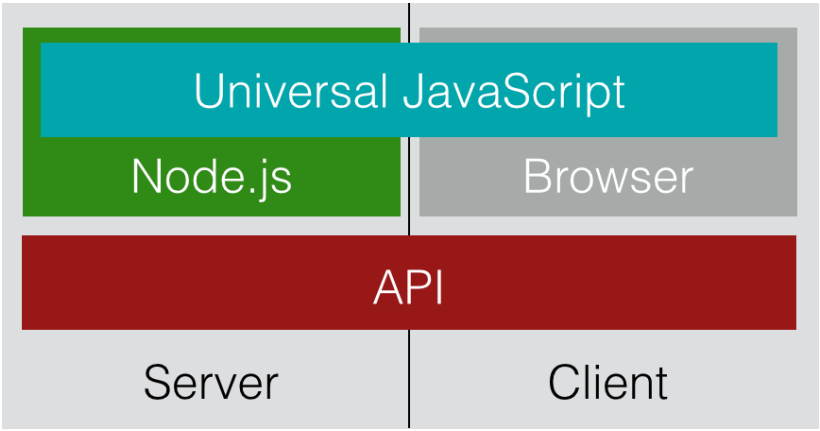


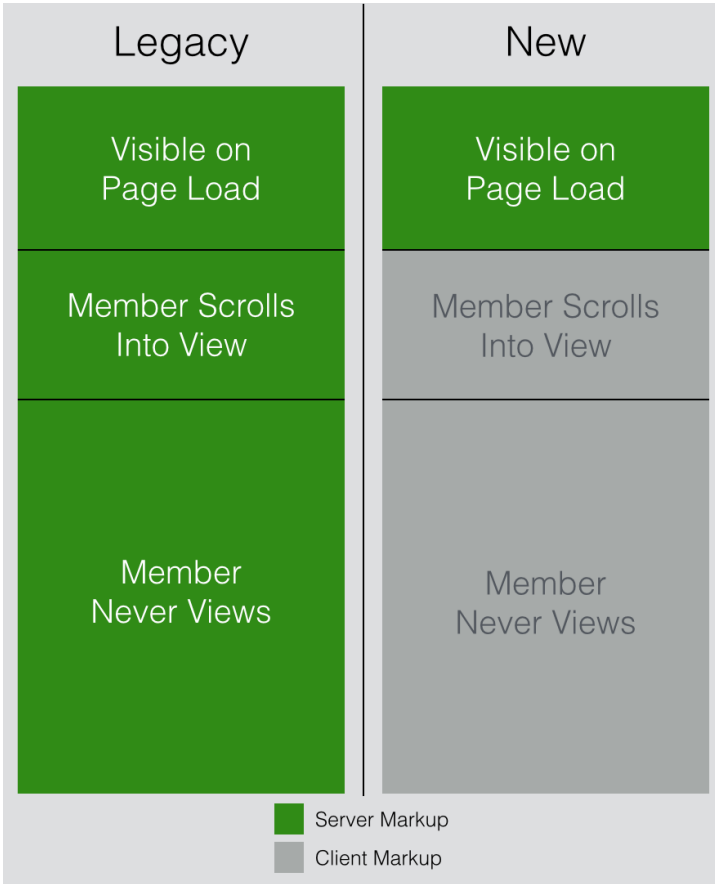
Caso2: Twitter - Lecciones

- Open Source
- Cambios incrementarles
  - Menor riesgo
  - Asociado a arquitecturas poco acopladas…
- Infraestructura
  - Data Center as a Computer (borg, mesos)

Caso 3: Netflix

- http://techblog.netflix.com/2015/08/making-netflixcom-faster.html
  - Server and Client Rendering
  - Universal JavaScript
  - JavaScript Payload Reductions





Analiza!

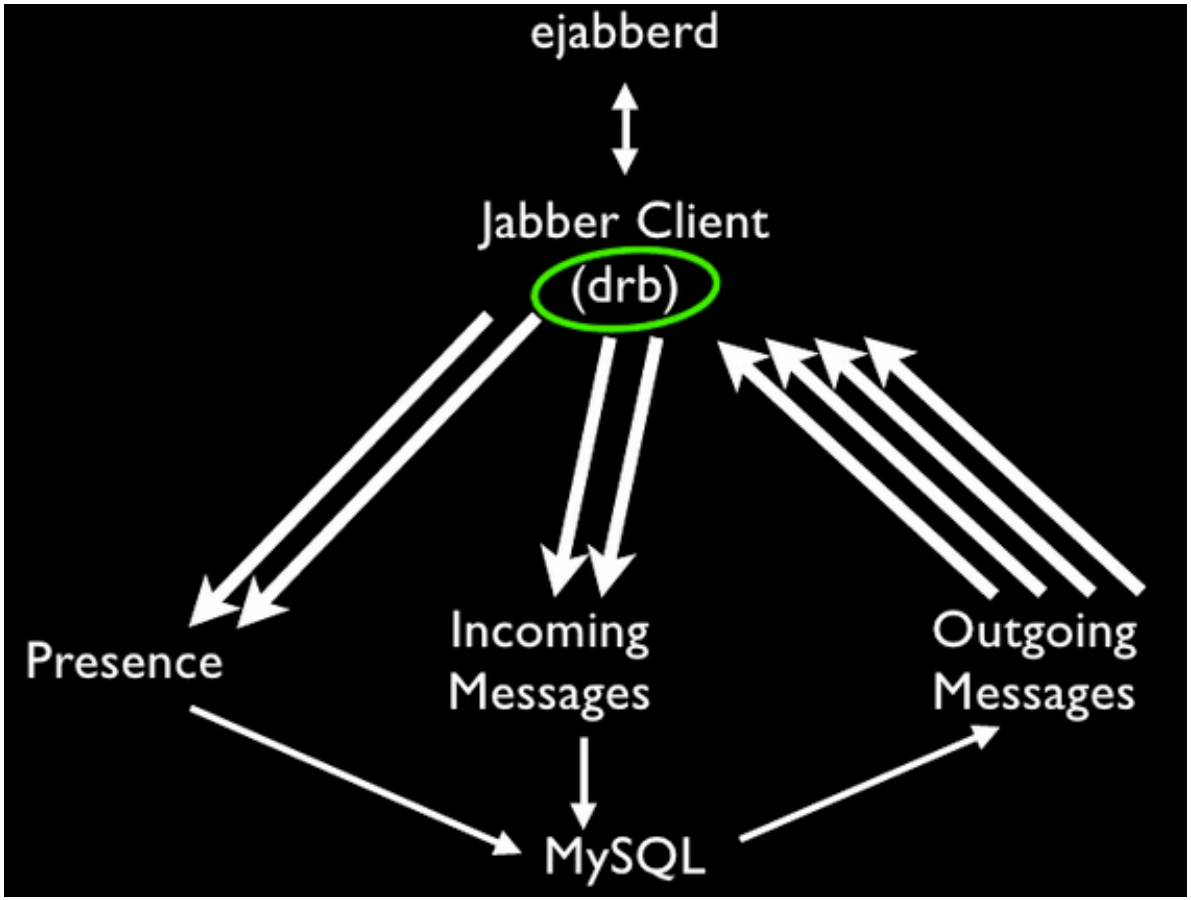
- Munin, Nagios, AWStats, Google Analytics
- Excepciones, Loggers
- Test!
- Agile!
- Benchmarks (data de users!)

Datos

- Rails con ActiveRecord
  - Indexar todo manualmente!
  - Denormalizar
  - Simplificar
- Cache: MemCache
  - ActiveRecord en memoria
- 90% API Request
  - Cache!

Conectores de mensajería

- Colas!
  - Fácil, rápido
  - Inestable
  - Sin redundancia
  - Acoplado
- DRb (Distributed Ruby)



Conectores de mensajería

- Rinda
  - Sistema distribuído para Ruby
  - Cola compartida (replicada en varias máquinas)
  - La base es DRb
  - RingyDingy (gema: registro de máquinas)
  - O(N) para lectura!

- Escaló! … 1 semana
- http://www.slideshare.net/markykang/drb-and-rinda
- http://www.druby.org/imaco_doc/ijpp_text_en.html

Conectores de mensajería

- Alternativas
  - Colas “tradicionales”
    - ActiveMQ (Java)
    - RabbitMQ (erlang)
    - MySQL + locks
    - ?

…

- Erlang! (whattsapp)
  - 4.000 transactions x s

…

- 9.000 “amigos” en 24 horas! … la historia continúa

Links

- http://www.slideshare.net/phegaro/linkedin-mobile-how-do-we-do-it
- http://arstechnica.com/information-technology/2012/10/a-behind-the-scenes-look-at-linkedins-mobile-engineering/2/
- http://engineering.linkedin.com/nodejs/blazing-fast-nodejs-10-performance-tips-linkedin-mobile
- http://www.infoq.com/presentations/Performance-V8-Dart
- http://techblog.netflix.com/2015/08/making-netflixcom-faster.html