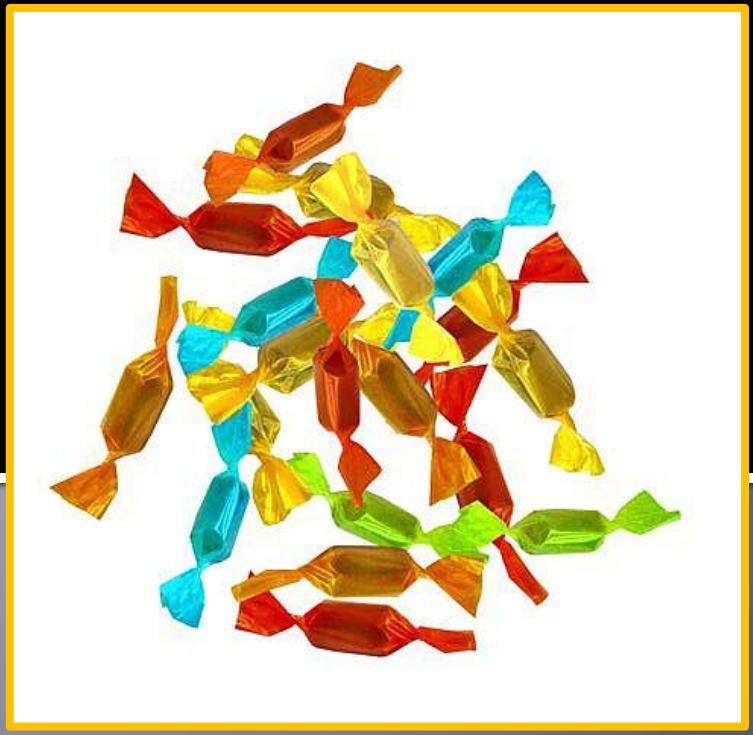
Tips *(mainly)* by Google and Kamil Majdanik

Improving website performance

Cukierki



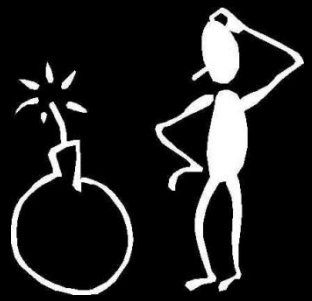
- Bonus za to, że w ogóle jesteście

- Dodatkowo można zarobić cukierka za:
  - **Inteligentne** pytania **na temat**
  - **Inteligentne** uwagi **na temat**
  - Zastrzegam prawo do nie dania cukierka dla osób rozmawiających
- Przepraszam jeżeli zapomnę
  - W razie czego proszę się upominać ☺

Agenda

- Problem
  - Ale czy to takie ważne?
  - Backend VS Frontend
- Techniki poprawiające wydajność
  - Co optymalizaujemy?
  - Cache
  - Minimalizacja czasu round-trip (RTT)
  - Zmniejszenie rozmiaru request'ów
  - Zmniejszenie ilości przesyłanych danych
  - Renderowanie
  - Optymalizacja dla urządzeń mobilnych
  - Inne techniki
- Projekty Google
  - PageSpeed
  - Google Public DNS
  - SPDY
  - WebP
- Bibliografia
- Pytania

Problem



Fred Wilson tworząc „10 Złotych Zasad tworzenia aplikacji internetowych” powiedział:

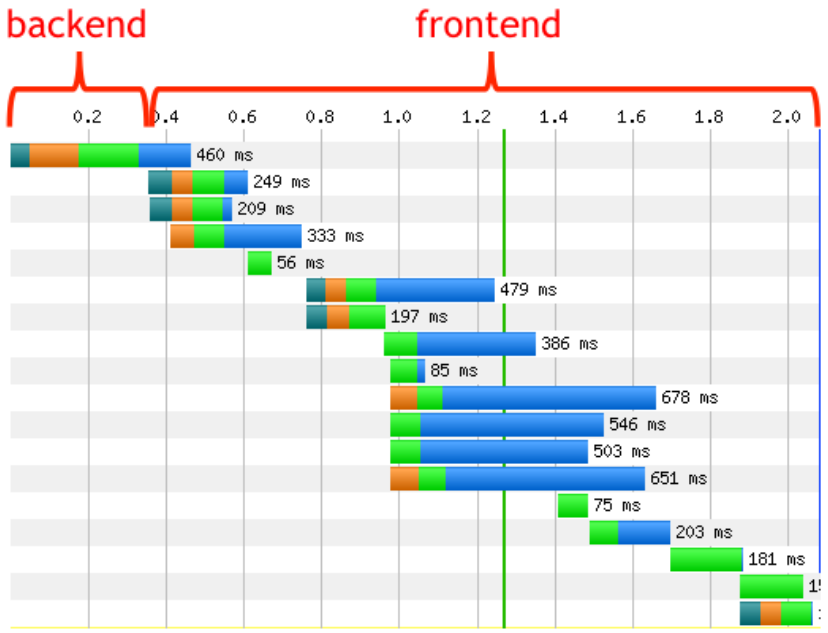
First and foremost, we believe that speed is more than a feature.
Speed is the most important feature. ***If your application is slow, people won't use it.***

Ale czy to takie ważne?

- **Amazon:**
  - 100ms nadmiernego czasu ładowania to 1% spadek sprzedaży *(Greg Linden, Amazon)*
- **Google:**
  - 500ms nadmiernego czasuładowania to 20% mniej zapytań wyszukiarki *(Marrissa Mayer, Google)*
- **Yahoo!:**
  - 400ms nadmiernego czasu ładowania powoduje 5-9% wzrost liczby kliknięć "Wstecz" przed zakończeniem ładowania strony *(Nicole Sullivan, Yahoo!)*

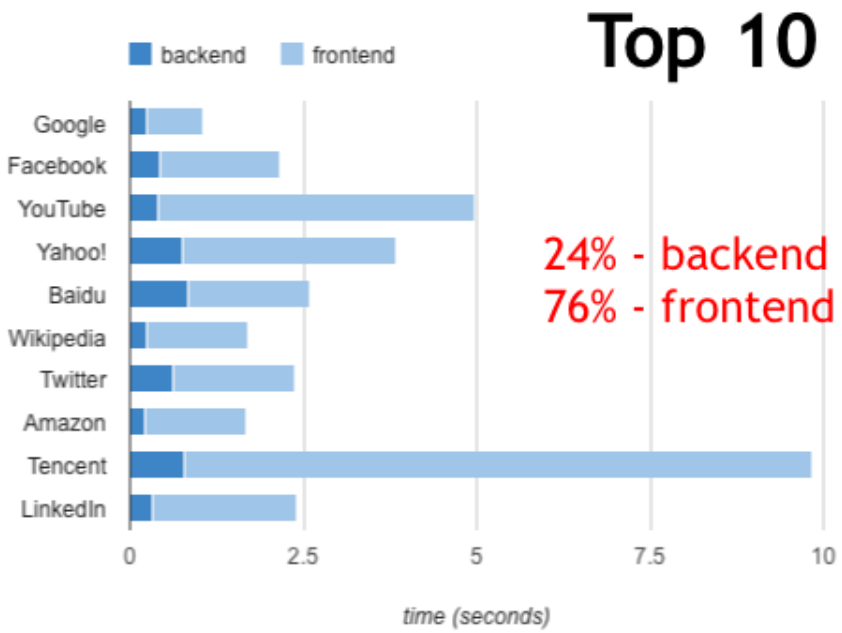
Backend vs Frontend (1)

- Backend
  - Bardzo ważny przy dużych serwisach
  - Istotny ze względu na skalowalność
- Frontend
  - Łatwiej poprawić niż backend
  - Więcej można zyskać
  - Prawie zawsze daje efekty

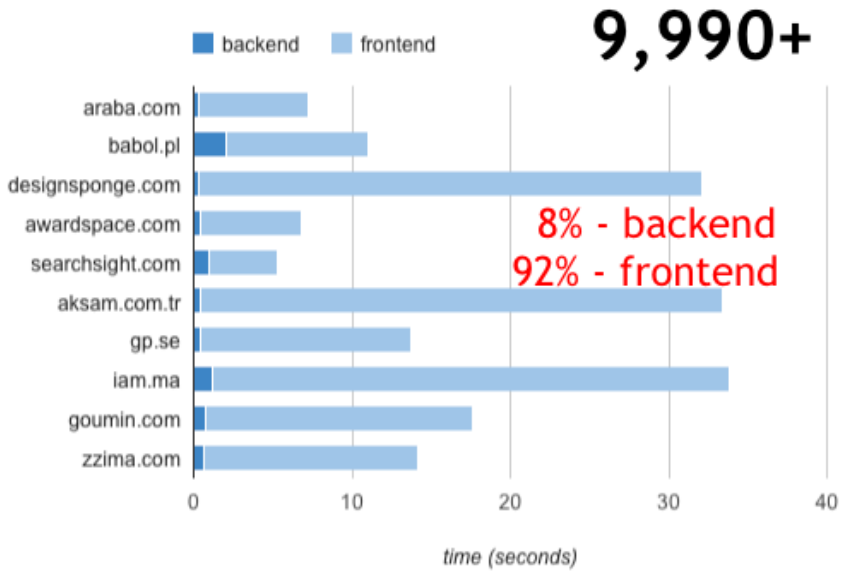


Backend vs Frontend (2)

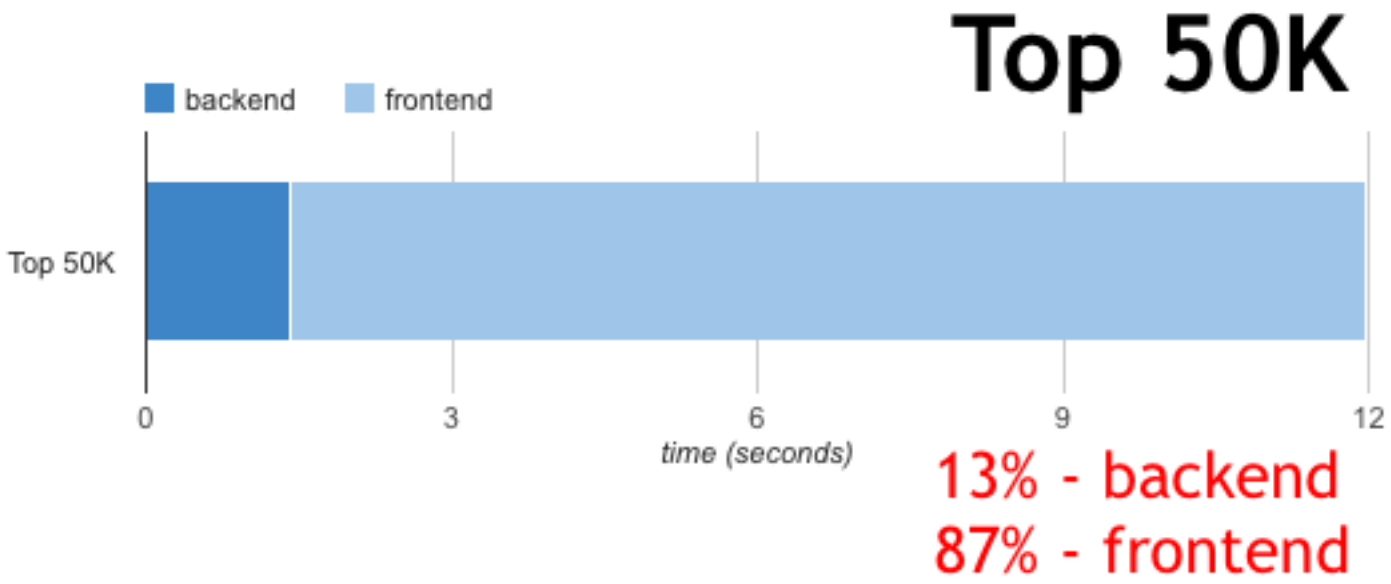
TOP10 WEBSITES



9 900+ ON RANKING WEBSITES



Backend vs Frontend (3)



Techniki poprawiające wydajność

Co optymalizujemy?

- Z punktu widzenia użytkownika
- Czas od wysłania żądania do pełnego załadowania strony, m.in.:
  - zapytania DNS
  - nawiązanie połączeń TCP
  - przesył nagłówków HTTP
  - pobieranie danych (lub użycie cache)
  - prasowanie i wykonywanie skryptów
  - renderowanie obiektów na stronie

Cache:
Empty Cache VS Full Cache

Empty Cache		Full Cache	
28.0K	1 HTML document		
1.9K	1 Style Sheet File		
59.5K	4 JavaScript Files	28.0K	1 HTML document
78.7K	24 Images	0.1K	2 Images
168.1K	Total size	28.1K	Total size
30	HTTP requests	3	HTTP requests
2.4s	Response time*	0.9s	Response time*

Optymalizacja cache: Cache przeglądarki (1)

- Co robią przeglądarki w SSL:
  - Dane bez nagłówków cache
    - Najnowsze (IE7, Chrome) heurystycznie zgaduję jak długo trzymać konkretne dane w cache
  - Inne w ogóle nie cache'ują
- Należy cache'ować wszystko co się da:
  - JS, CSS, obrazki, PDF'y, Flash, etc

Optymalizacja cache: Cache przeglądarki (2)

- Dane nie będą ściągane na pewno
  - „Expires” i „Cache-Control: max-age”
- „Być może”:
  - „Last-Modified” (data)
    - używane są heurystyki (czasem zapytania warunkowe GET)
  - „ETag” (np. wersja, hash)
- Powinniśmy ustawić
  - ("Expires" OR "Cache-Control: max-age”) AND ("Last-Modified" OR "Etag")
- Redundantne jest wstawienie:
  - ("Expires" AND "Cache-Control: max-age ") LUB ("Last-Modified" AND "Etag”)

Optymalizacja cache: Cache przeglądarki (3)

- Zalecenia:
  - Statyczne dane:
    - „Expires” na miesiąc – rok (bliżej roku)
    - „Last-Modified”
  - Dane zmieniane okazjonalnie
    - Fingerprint w adresie URL
  - Dla IE – najlepiej usunąć „Vary”
  - URL’e powinny różnić się o co najmniej 8 znaków
    - Funkcja hash’ująca w Firefox może powodować kolizje (po restarcie przeglądarki)
  - „Cache control: public” dla Firefox w HTTPS

Optymalizacja cache: Cache proxy

- Dane mogą być cache'owane w publicznych serwerach proxy (np. u ISP)
- Użytkownik może użyć cache nawet jeżeli wchodzi pierwszy (bo inny użytkownik zapisał do cache)
- „Cache-control: public"
- Zalecenia:
  - Nie używać „?" w URL'u dla danych statycznych
    - Squid od 3.0 nie cache'uje
  - Dla danych używających cookie ustawić „Cache-control: private"
  - Bugi w cache proxy
    - User dostaje skompresowane dane i nie umie ich zdekompresować
    - „Vary: Accept-encoding" – dwie wersje (skompresowana i nieskompresowana)

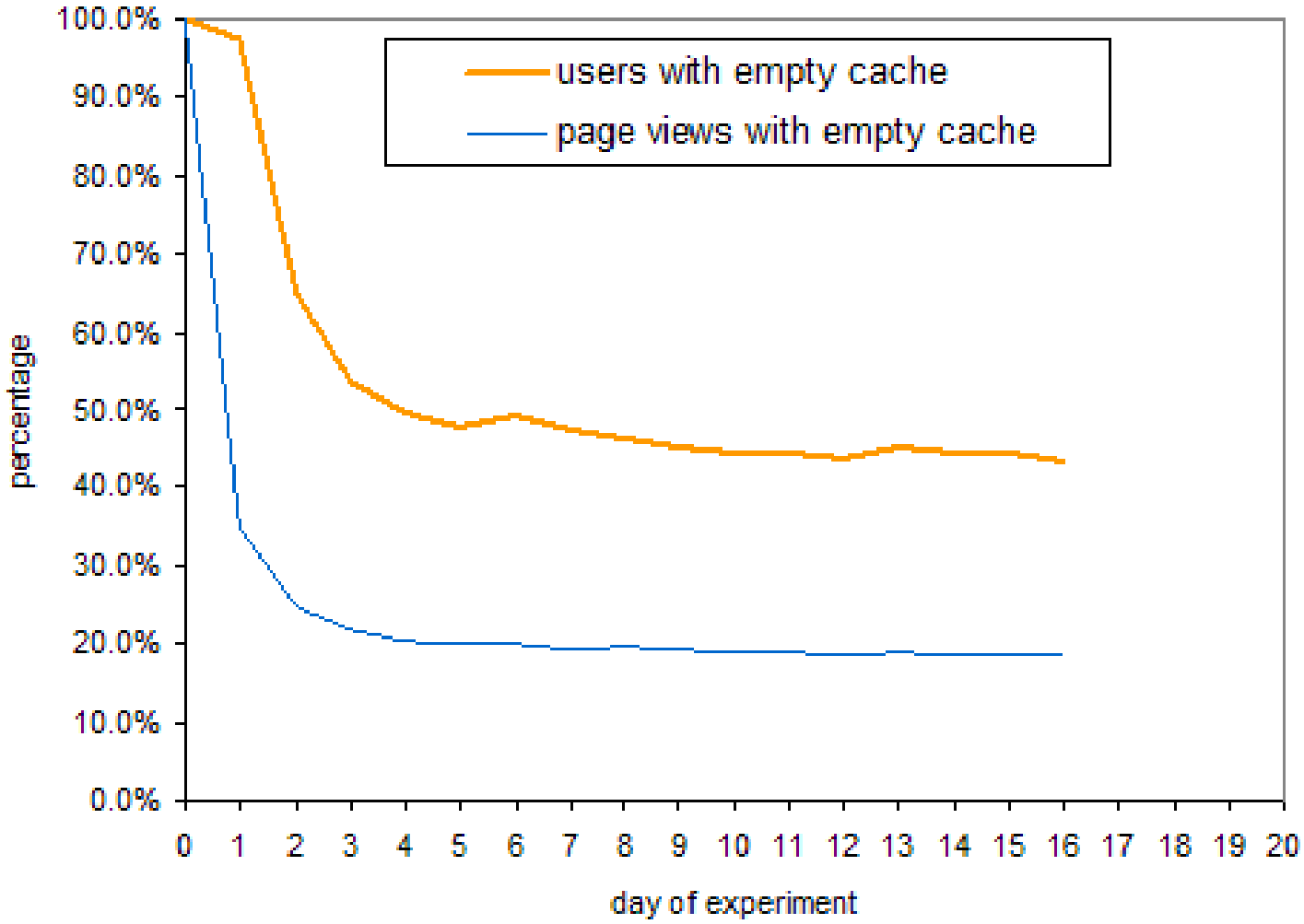
Cache: Frontend VS Backend

TABLE 1

Percentage of time spent on the back end

Web Site	Empty Cache	Primed Cache
http://www.aol.com/	3%	3%
http://www.ebay.com/	5%	19%
http://www.facebook.com/	5%	19%
http://www.google.com/search?q=flowers	53%	100%
http://search.live.com/results.aspx?q=flowers	33%	100%
http://www.msn.com/	2%	6%
http://www.myspace.com/	2%	2%
http://en.wikipedia.org/wiki/Flowers	6%	9%
http://www.yahoo.com/	3%	4%
http://www.youtube.com/	2%	3%

Cache: Zmniejszenie ruchu



Minimalizacja czasu round-trip (RRT)

- Czas od wysłania żądania do otrzymania odpowiedzi
  - Nie licząc transferu danych
- Przykładowo dla pierwszego połączenia: 3 RRT
  - Po jednym RRT na: DNS, TCP, HTTP
- Zazwyczaj 1ms (dla LAN) do 1s (inny kontynent)
- Bardzo istotny dla danych o małym rozmiarze
  - Np. dane wyszukiwarki
- Zazwyczaj optymalizacja = zmniejszenie liczb żądań HTTP (lub zrównoleglić je)

Minimalizacja czasu round-trip (RRT): Zapytania DNS (1)

- Jeżeli serwer DNS nie ma danych w cache to 1RRT może oznaczać ich kilka
- Często mamy małe TTL (5 minut – 1 dzień)
  - Niektóre serwery i tak trzymają dłużej (nawet 30 minut)
  - f4
- Warto:
  - zwiększyć TTL
  - zmniejszyć liczbę rekordów CNAME (ten sam IP dla kilku hostname)
  - Replikacja DNS w kilku regionach

Minimalizacja czasu round-trip (RRT): Zapytania DNS (2)

- Warto cd.:
  - Najłatwiej: zmniejszyć liczbę hostów dla danych (pamiętając o równoległości)
    - To także mniej połączeń TCP
    - Zwiększenie hit-ratio dla cache'u DNS
    - Optymalnie to 1-5 hostów
  - Zmniejszyć żądania z „critical-path" (to co potrzebne do załadowania pierwszej „wersji" strony)
    - CSS'y, JS'y powinny być z tego samego hosta
    - Jeżeli musimy użyć kilku hostów, to ładujmy na końcu strony

Minimalizacja czasu round-trip (RRT): Liczba przekierowań (1)

- Często przekierowania używane są do:
  - Dane zmieniły położenie
  - Gromadzenie statystyk
  - Poprawianie literówek
  - Przekierowanie
    - na inną domenę (np. kraj)
    - protokół (HTTP/HTTPS)
    - autoryzowane strony
- Powoduje to dodatkowe zapytania HTTP
  - Starajmy się to zminimalizować

Minimalizacja czasu round-trip (RRT): Liczba przekierowań (2)

- Zalecenia:
  - Jak zasób zmienia położenie
    to zmieniamy wszystkie odnośniki
    (a nie robimy przekierowanie)
  - Unikać wielokrotnych przekierowań
    - Jeżeli A*->C, B->C, to nie róbmy A->B->C
  - Używać „rewrite” na serwerze (mod_rewrite)
    - Lepiej nie dla danych cache’owanych
      (cache będzie trzymać kilka kopii)
  - Starać używać się przekierowań HTTP, a nie JS
    - 301 lub 302 HTTP („moved permanently”)
      - To może nawet być cache’owane
    - „<meta http-equiv="refresh">”

Minimalizacja czasu round-trip (RRT): Liczba przekierowań (3)

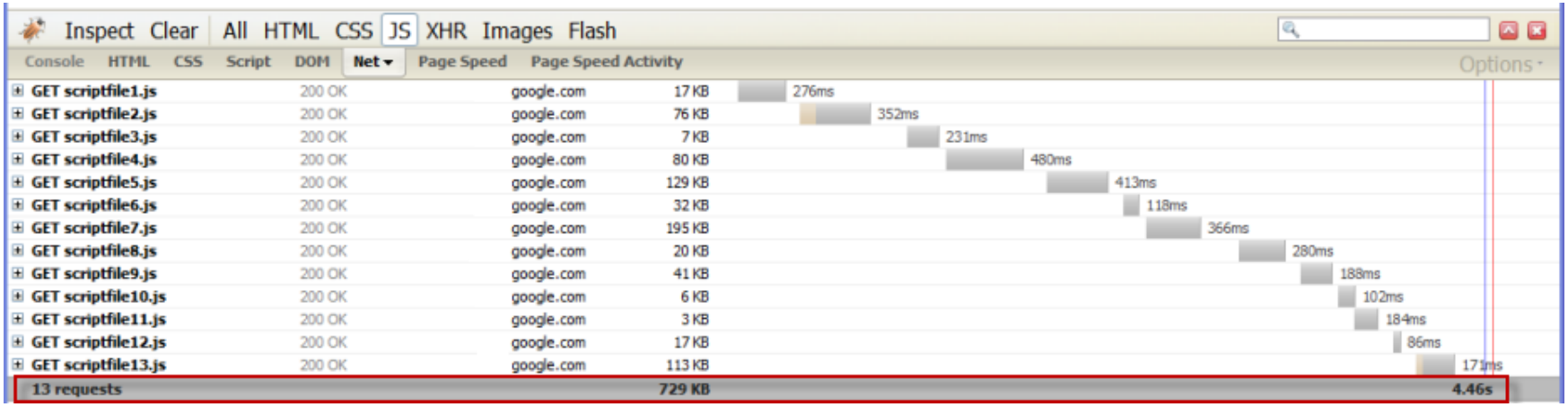
- Dla celów statystycznych nie robić przekierowań
  - A najlepiej w tle – na dole strony (tak robit Google Analytics):

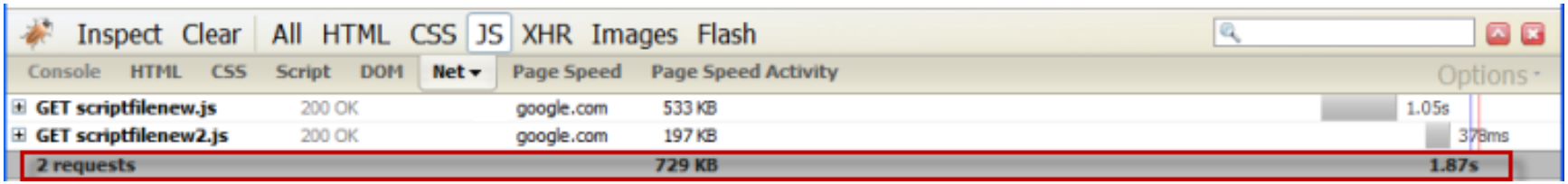
```
<script type="text/javascript">
var thisPage = location.href;
var referringPage = (document.referrer) ?
     document.referrer : "none";
var beacon = new Image();
beacon.src =
     "http://www.example.com/logger/beacon
     .gif?page=" + encodeURI(thisPage)
     + "&ref=" + encodeURI(referringPage);
</script>
```

Minimalizacja czasu round-trip (RRT): Bad request

- „Bad request” = błędy 404/410
- Są automatyczne narzędzia do wykrywania

Minimalizacja czasu round-trip (RRT): Połączyć zewnętrzny JS/CSS (1)

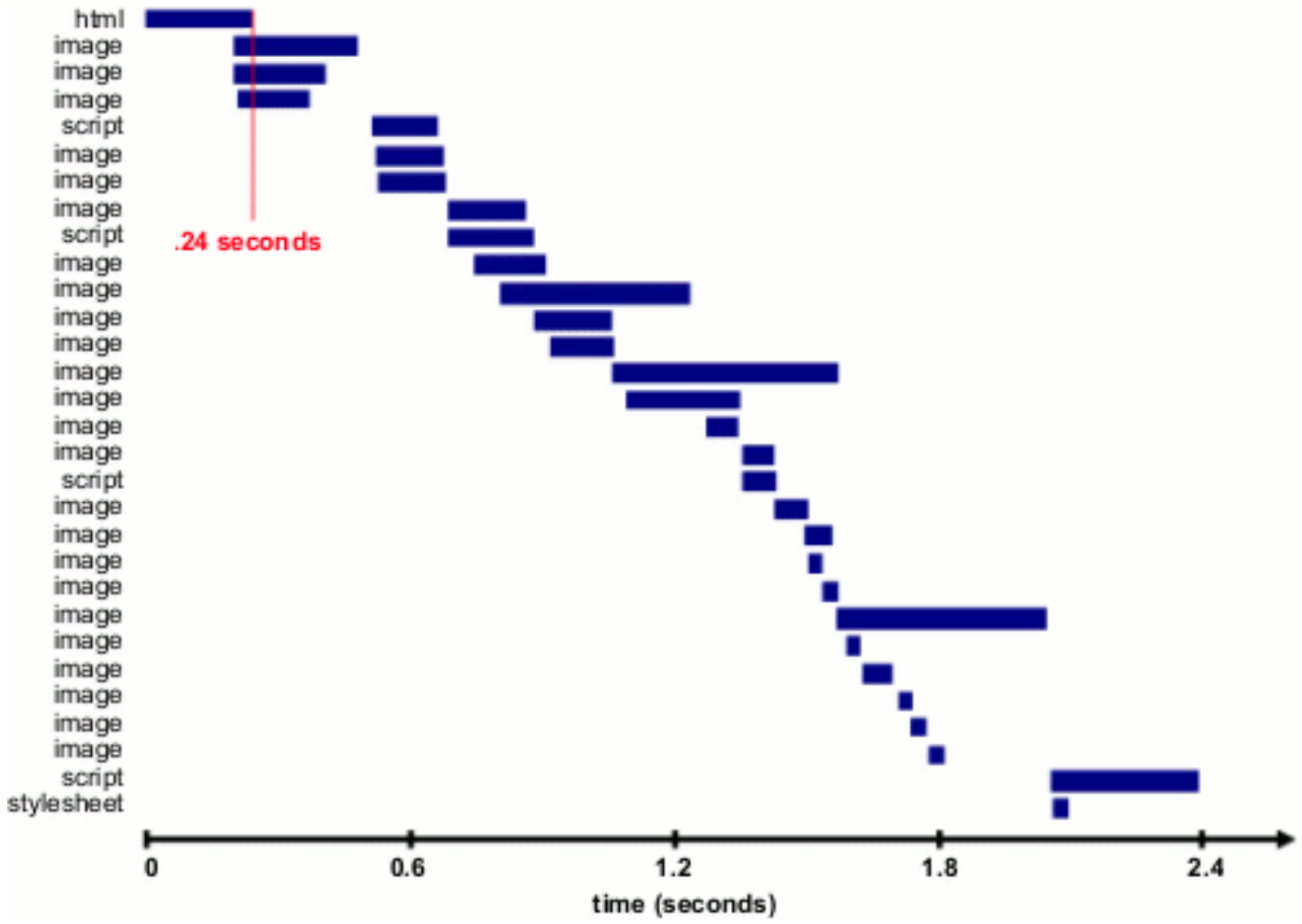




Minimalizacja czasu round-trip (RRT): Połączyć zewnętrzny JS/CSS (2)

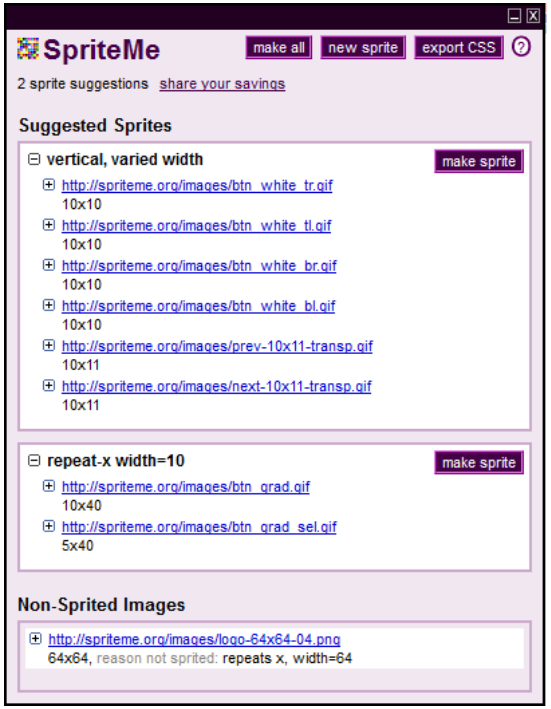
- Z drugiej strony wiele plików to:
  - Modularyzacja kodu
  - Wersjonowanie
  - Kod, który chcemy załadować na końcu strony
- Zalecenia:
  - W miarę możliwości podzielić kod na 2 pliki:
    - 1. plik na startup, 2. plik na końcu strony
  - Include'y w „<head>"
  - Rzadko używany kod w jednym pliku
  - Dla niewielkiego kodu, który nie powinien być cache'owany:
    - Umieścić go w HTML'u
    - Odpowiednia kolejność include'ów
      - O tym później

Minimalizacja czasu round-trip (RRT): CSS spirte'y (1)



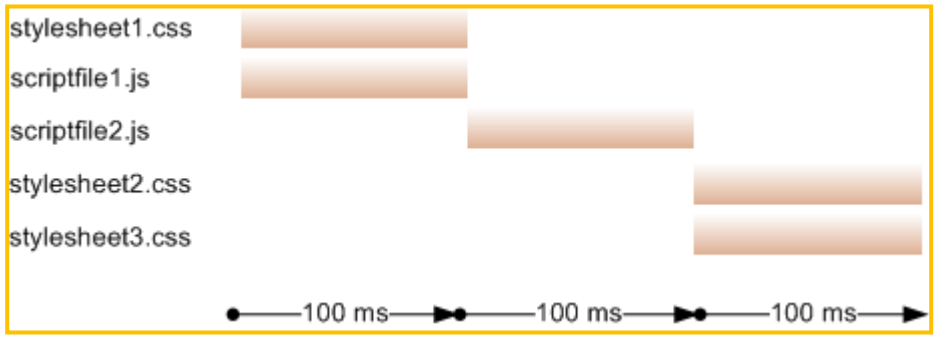
Minimalizacja czasu round-trip (RRT): CSS spirte'y (2)

- Zalecenia:
  - Obrazki ładowane na raz w jednym pliku
    - Zmieniające się dynamicznie NIE:
      - Np. obrazki profilowe
  - GIF i PNG dobrze się nadają do tego
    - Grupujemy po kolorach
  - Najistotniejsze są obrazki:
    - Małe
    - Długo cache'owane
  - Zmniejszyć puste miejsce
  - Użyć SpriteMe

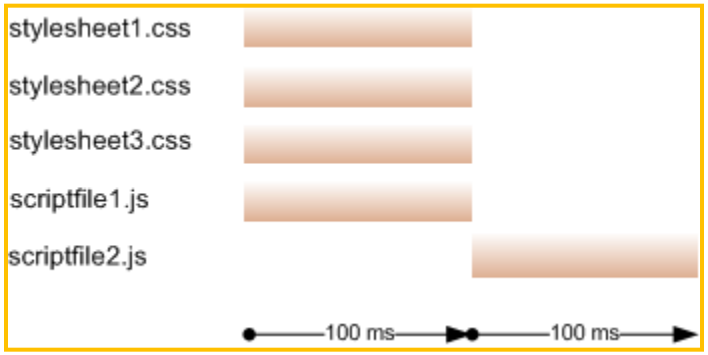


Minimalizacja czasu round-trip (RRT): Kolejnosć include'ów JS i CSS

- Include JS czeka na poprzednie
  - Nie można wykonać JS jak nie wykonano poprzednich
  - Niektóre przeglądarki:
    - Stop do momentu ściągnięcia i wykonania JS
- Zalecenia:
  - CSS zawsze przed JS
  - Kod CSS/JS inline'owany w HTML na końcu



VS



Minimalizacja czasu round-trip (RRT): document.write

- Unikać! ☺
- Przeglądarki nie mogą „przewidzieć” co tam będzie bez sparsowania tego
- Zalecenia:
  - Nie robić tak:

```
<script>
document.write('<script
src="example.js"><\/script>');
</script>
```

  - Rozważyć „<iframe>”
    - Wtedy cała strona nie czeka na załadowanie ramki

Minimalizacja czasu round-trip (RRT): Ładowanie asynchroniczne

- Np.

```
<script>
var node = document.createElement('script');
node.type = 'text/javascript';
node.async = true;
node.src = 'example.js';
// Now insert the node into the DOM, perhaps using
insertBefore()
</script>
```

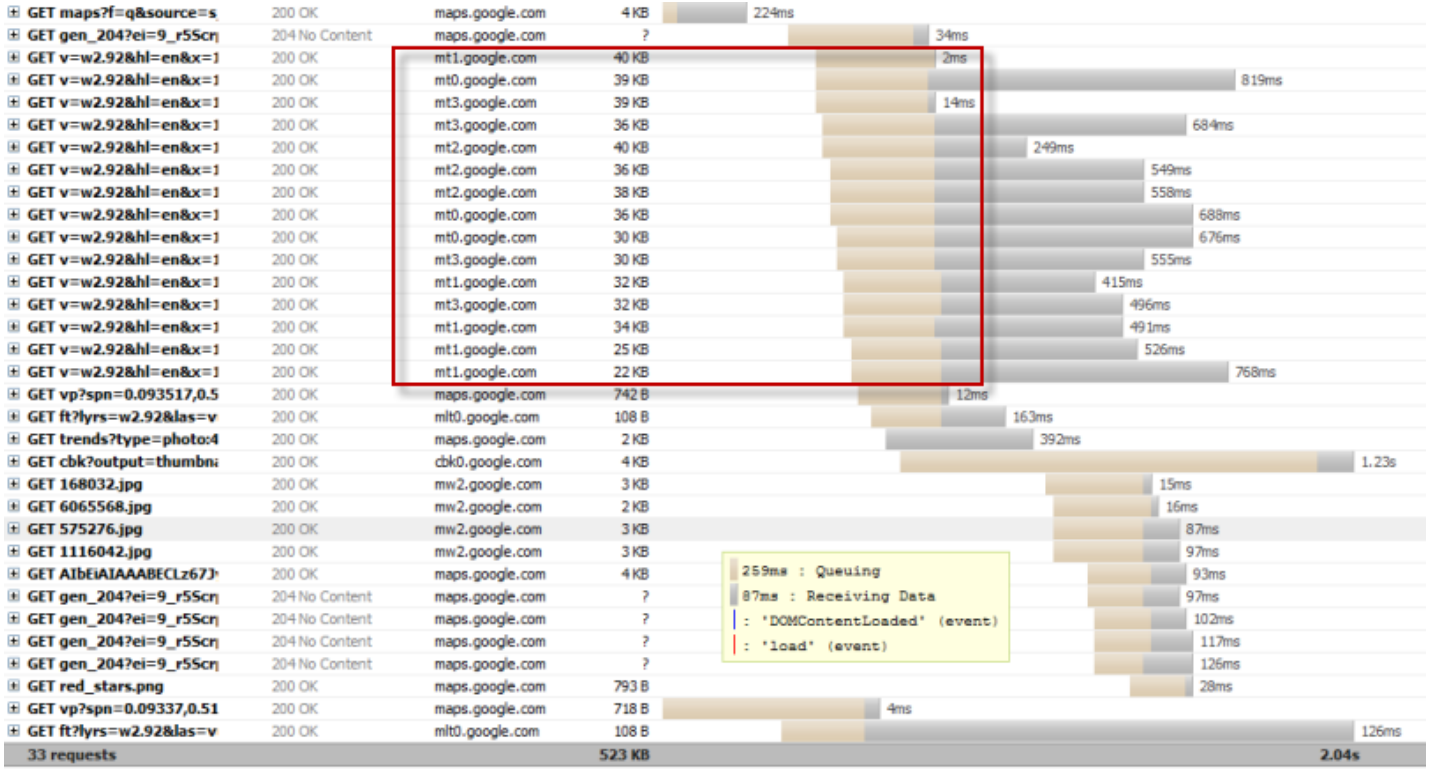
  - Powyższe działa w:
    - IE, FF, Chrome, Safari
  - „<script async= " true " >” działa tylko w:
    - Chrome, FF >3.6

Minimalizacja czasu round-trip (RRT): Kilka hostów (1)

- Równoległe pobieranie z kilku hostów
- W HTTP 1.1 jest limit na 2 równoległe połączenia z jednym hostem
  - Kolejka FIFO
- Więcej hostów = więcej równoległości
  - Ale więcej zapytań TCP, DNS…
  - No i CPU u użytkownika
  - Optymalnie 2-5 hostów
  - Dobrze nadaje się do danych statycznych (np. obrazki)
  - Można użyć CNAME
  - Może CDN (Content delivery network)
  - 1 zasób = 1 host

Minimalizacja czasu round-trip (RRT): Kilka hostów (2)

- Przykład: Google Maps
  - Mapa podzielona na kafelki



Zmniejszenie rozmiaru request'ów (1)

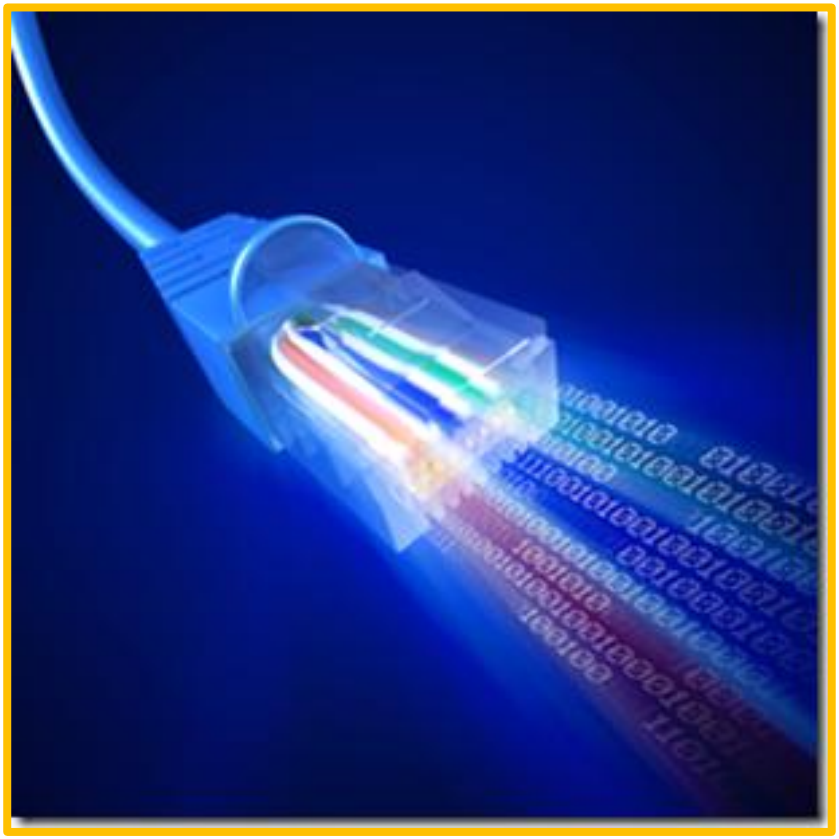
- Każde żądanie HTTP wysyła:
  - Ciasteczka (limit 1KB)
  - Dane użytkownika (user agent)
  - Dane GET i POST
  - URL
- Łącza są asynchroniczne: 1:4 do 1:20
  - 1KB wysłany = 10KB odebranych (przez użytkwnika)
  - Albo i jeszcze więcej:
    - dane nie są kompresowane
    - „slow start" TCP (czekamy na odpowiedź)

Zmniejszenie rozmiaru request'ów (2)

- Zalecenia:
  - W ciasteczku trzymamy tylko ID
    - Coś w stylu sesji
  - Ciasteczka wszystkich ścieżek „/…" zapisywać do globalnej „/"
    - Analogicznie w drugą stronę
  - Dane statyczne serwujemy z „cookieless domains"
    - JS, CSS, obrazki nie potrzebują ciasteczek

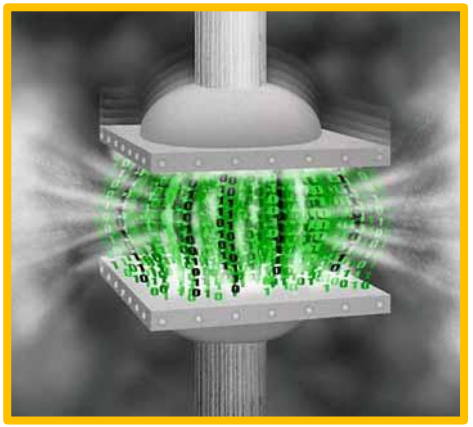
Zmniejszenie ilości przesyłanych danych

- Przepustowość łącz jest ograniczona
  - Szczególnie w mobilnym internecie
  - Maksymalne MTU (maksymalny rozmiar pakietu) to 1,5KB dla Ethernet

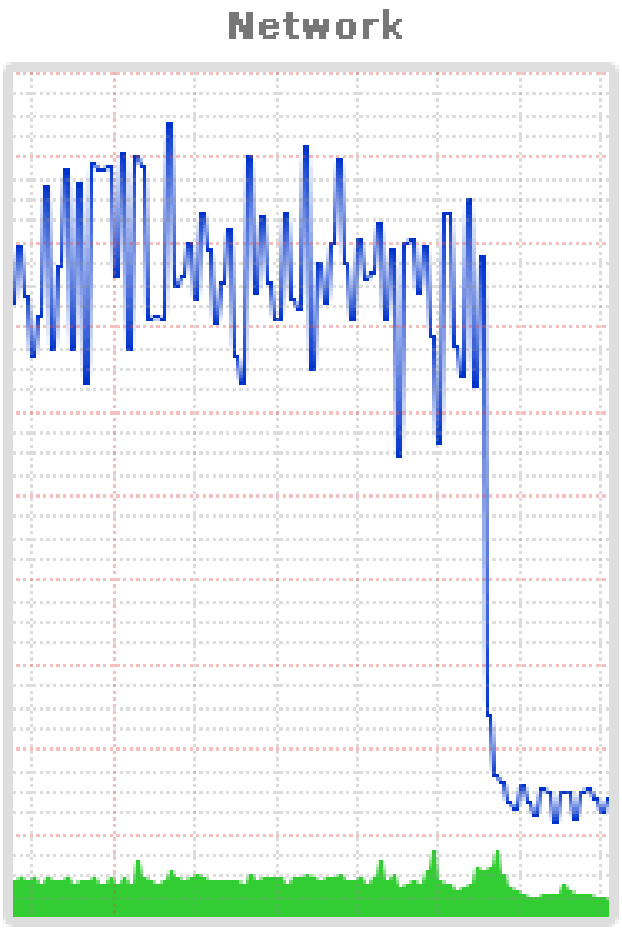


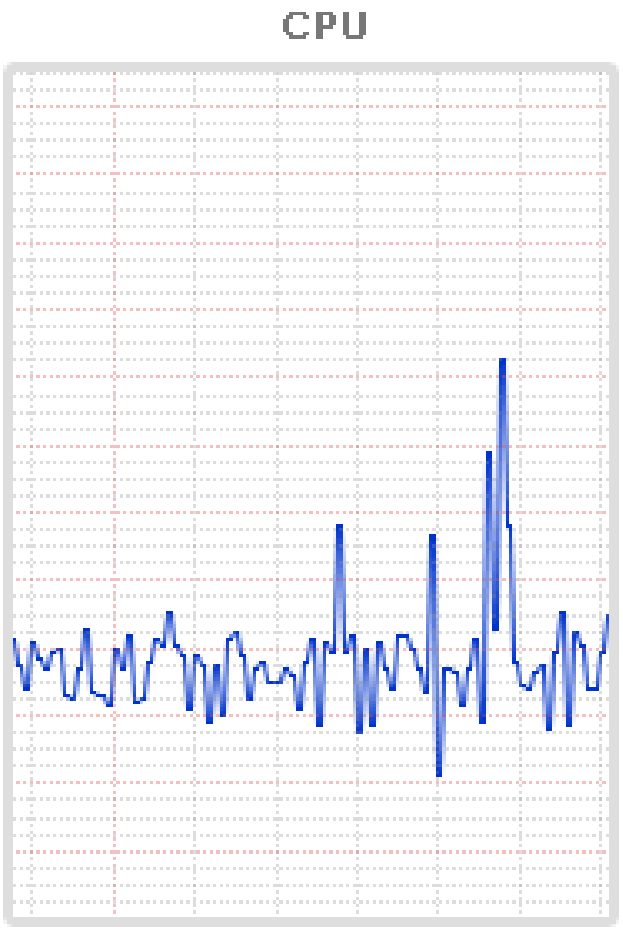
Zmniejszenie ilości przesyłanych danych: Kompresja (1)

- gzip / deflate
- Dobrze kompresuje się HTML, CSS, JS
  - Nie używać do danych binarnych
- „Content-encoding: gzip”
- Dane dużego rozmiaru (minimum to 0,15KB-1KB)
- Zalecenia (zwiększające kompresję):
  - Kolejność alfabetyczna:
    - Selektorów CSS
    - Atrybutów w znacznikach
      - W Google Search zmniejszyło o 1,5% rozmiar
  - Lowercase
  - Jednolicie: cudzysłowia albo apostrofy



Zmniejszenie ilości przesyłanych danych: Kompresja (2)





Zmniejszenie ilości przesyłanych danych: Rozmiar JS/CSS/HTML

- Należy zmniejszyść rozmiar
- Gotowe narzędzia:
  - JavaScript:
    - Closure Compiler, JSMin, YUI Compressor
  - CSS:
    - YUI Compressor, cssmin.js
- Dane większe niż 4KB
- HTML
  - Problem z zgodnością ze standardami

Zmniejszenie ilości przesyłanych danych: Inne

- Opóźnienie ładowania danych
  - Często część JS możemy załadować na końcu
  - Obrazki po scroll'u
- Odptymalizacja obrazków
  - Małe obrazki = GIF
    - 10x10 pikseli, < 3 kolory
  - Zapomnijmy o BMP, TIFF
  - Dobrać odpowiedni rozmiar kompresji
    - Metoda prób i błędów
  - Nie przesadzać z skalowaniem obrazków w HTML
- Ten sam URL dla jednego zasobu (jeżeli jest na kilku hostach)
  - *forum.example.com* i *developer.example.com* używają pliku *www. example.com/logo.gif*

Renderowanie: Wydajny CSS (1)

- Algorytm:
  - Dla każdego taga:
    - Przechodzimy regułę po selektorach od prawej do lewej
    - Np.
      ** html #atticPromo ul li a {…}*
      #footer h3 {…}
      …
- Unikajmy:
  - *, tag jako klucz
    - *body* ******* *{}*, *body* ***table*** *{}*
  - Wyrażeń CSS

```
#myDiv {
  left:   expression(document.body.offsetWidth  - 110 + "px");
  width:  expression(ieBox ? "100px" : "80px");
}
```

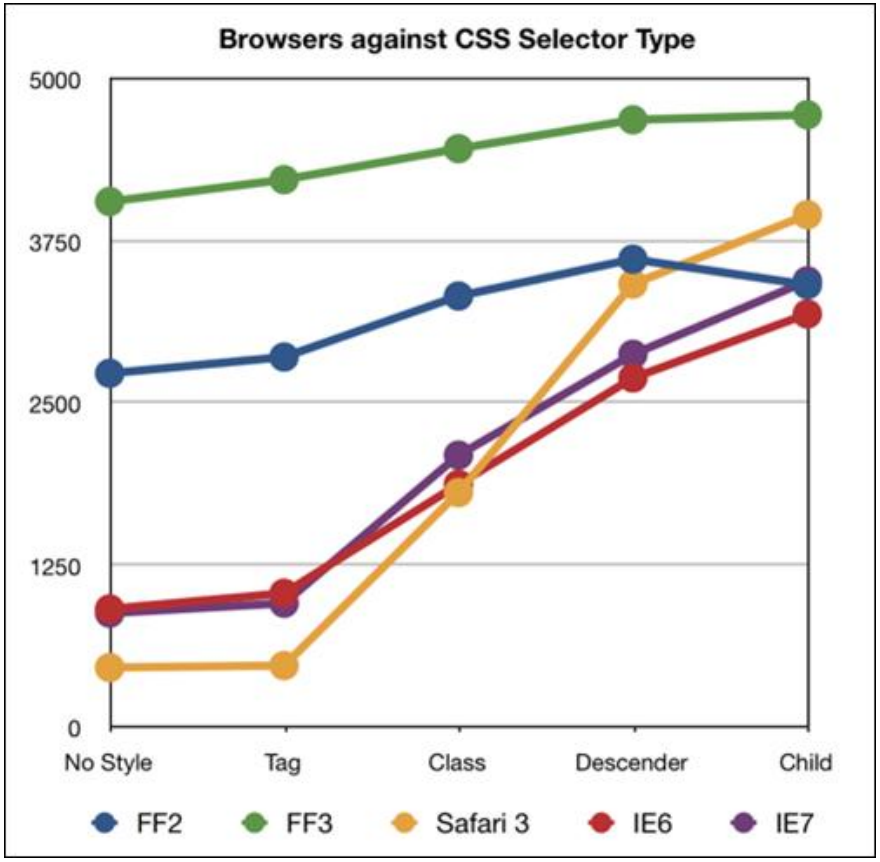
    - Są wolniejsze niż JS i niekompatybilne z starymi przeglądarkami

Renderowanie: Wydajny CSS (2)

- Zalecenia:
  - # lub . wystarczy
    - button#backButton VS #backButton
    - .menu-left#newMenuIcon VS #newMenuIcon
  - Najbardziej konkretnie, mniej selektorów
    - treeitem[mailfolder="true"] > treerow > treecell VS .treecell-mailfolder
  - Selektor potomka jest wolny
    - treehead treerow treecell VS treehead > treerow > treecell
  - Używanie dziedziczenia (# jest konkretniejszy niż .)
    - #bookmarkMenuItem > .menu-left VS #bookmarkMenuItem
  - *:hover, :visited* tylko do elementów, które to obsługują
    - Spowalnia to IE
- Nie uzywać @import

Renderowanie: Wydajny CSS (3)

- Test:
  - 20 000 elementów A
  - Legenda
    - No style (brak arkusza CSS)
    - Tag (*A)*
    - Class (*.class)*
    - Descender (DIV DIV DIV P)
    - Child (DIV > DIV > DIV > P)



Renderowanie: Wydajny CSS (4)

	# Rules	# elements	Avg Depth
AOL	2289	1628	13
eBay	305	588	14
Facebook	2882	1966	17
Google Search	92	552	8
Live Search	376	449	12
MSN.com	1038	886	11
MySpace	932	444	9
Wikipedia	795	1333	10
Yahoo!	800	564	13
YouTube	821	817	9
average	1033	923	12

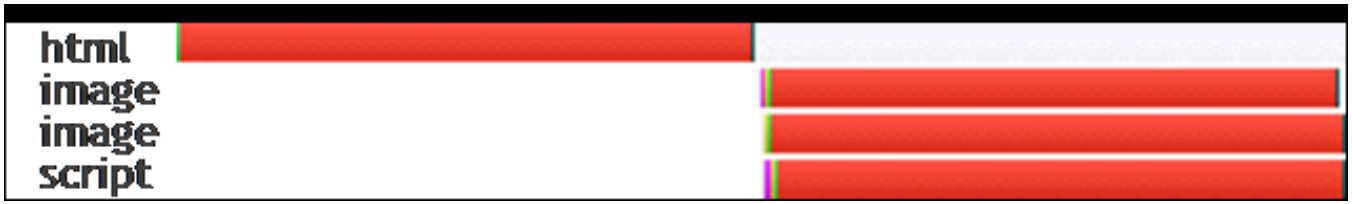
Renderowanie: Inne

- Ustalać rozmiary obrazków
  - Może być w HTML lub CSS
- Ustalać kodowanie w HTTP
  - Jeżeli jest w „<meta>”, to musi zgadywać i jak się pomyli od nowa parsować
  - Jak w ogóle nie podamy to też zgaduje (ale jeszcze wolniej)

Optymalizacja dla urządzeń mobilnych

- Dlaczego?
  - Wolne CPU (w porównaniu do PC ☺)
  - Wolne połączenia sieci mobilnych (RTT)
  - 100KB JS = 0,1s
- Zalecenia:
  - To co dla zwykłych stron
  - Opóźnienie ładowania JS
  - Ładowanie kodu JS jako string (jako zmienna albo w komentarzu)
    - a potem odpalenie *eval()*

Inne techniki (1)



VS

- <!-- css, js -->
  </head>
  <?php flush(); ?>
  <body>
  <!-- content -->

Inne techniki (2)

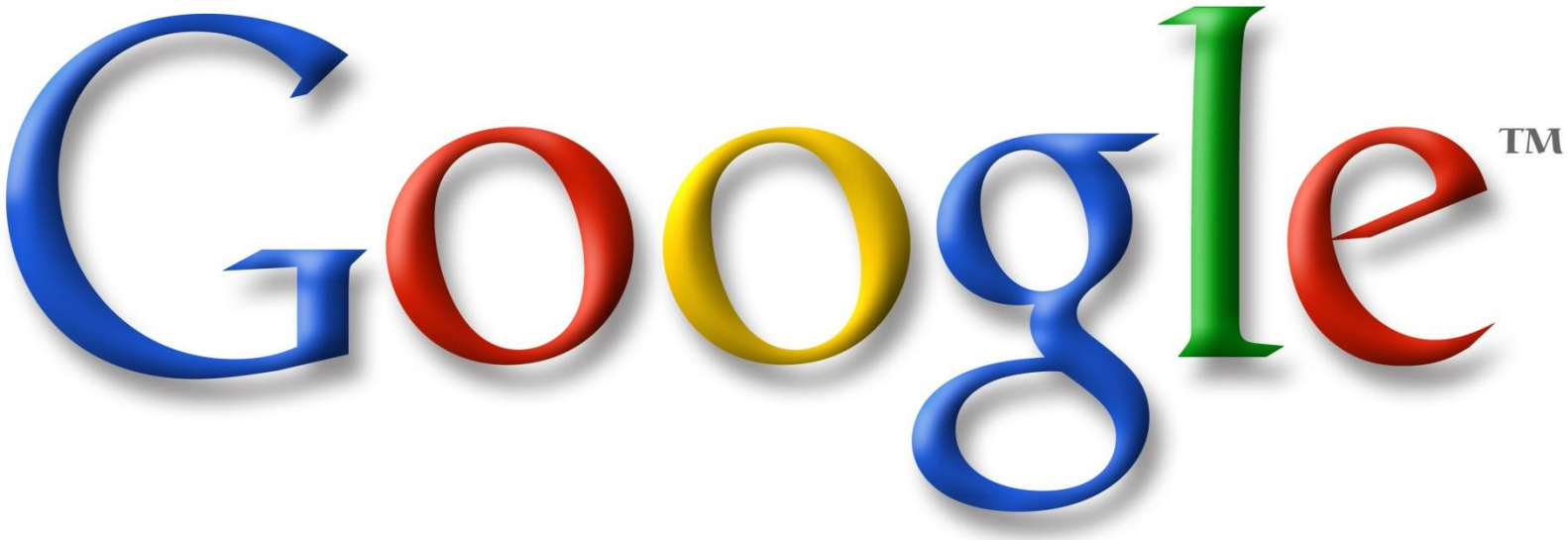
- GET dla żądań Ajax
  - POST jest dzielone na dwie części (nagłówki i dane)
    - GET nie
  - Maksymalna długość GET = 2K
- Ładowanie na żądanie „post-load” (np. obrazki)
  - YUI Image Loader (np. strona główna Yahoo)

Inne techniki (3)

- Ładowanie „na przód”
  - Bezwarunkowe – od razu
  - Warunkowe
    - np. jak user coś zrobi: wpisze kilka znaków do „<input>”
  - „*Anticipated*” – po przebudowie strony
    - Cache jest pusty – uzpełnijmy go gdy user nic nie robi
- Mniej elementów w DOM
  - Np. „<table>” VS „ ”
  - Strona główna Yahoo ma tylko 700 tagów!
- Rozmiar komponentów < 25KB
  - iPhone ich nie cache’uje

Projekty Google

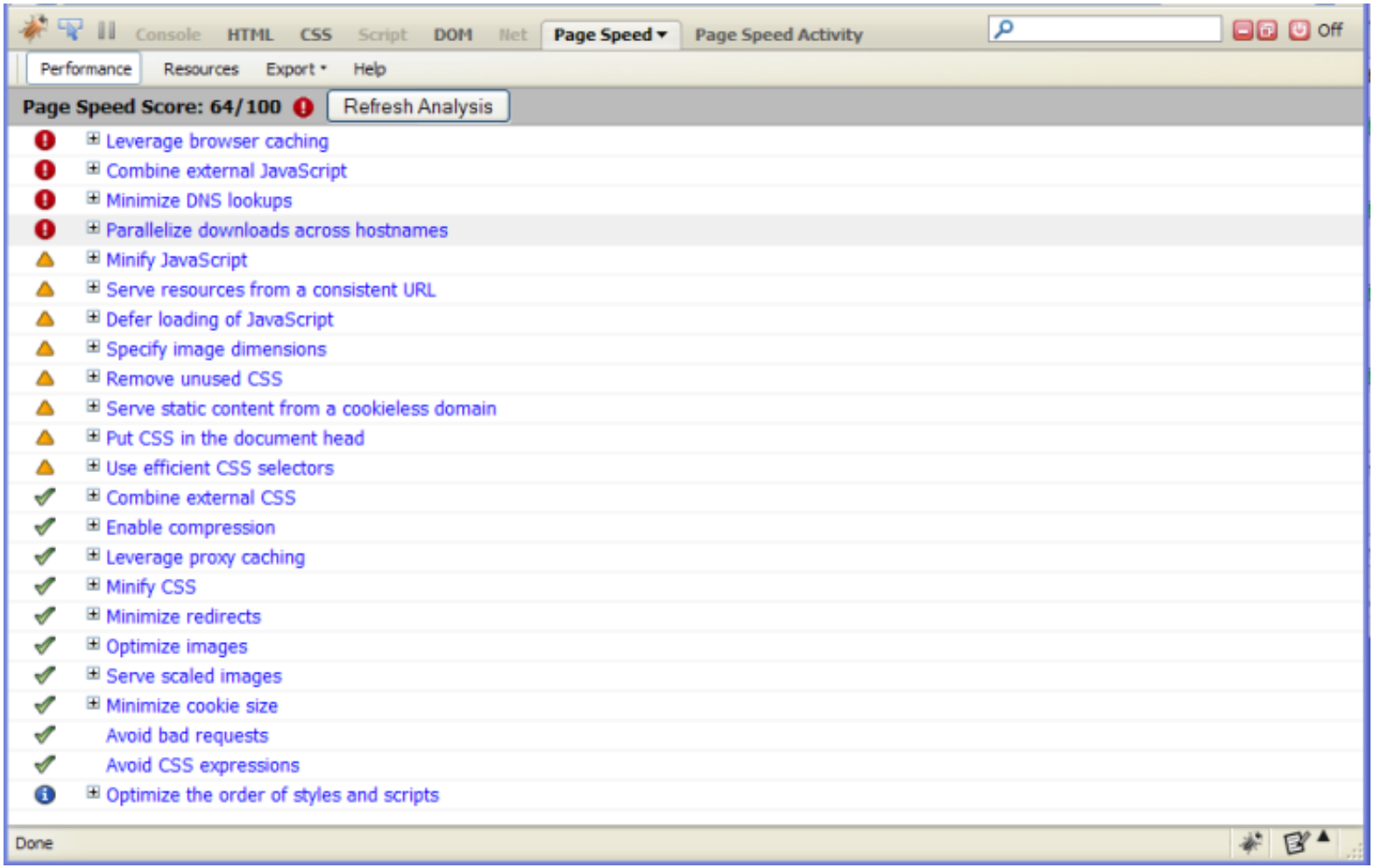
Wpomagające poprawianie wydajności



PageSpeed (1)

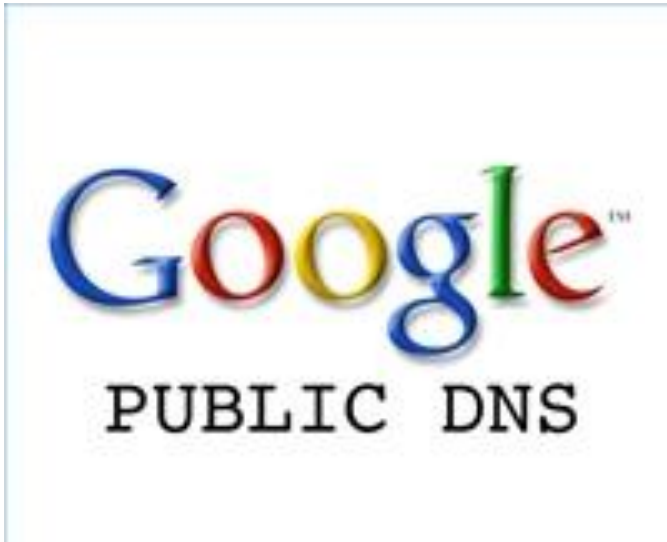
- Realizuje większość wspomnianych wcześniej wskazówek
- Podpowiada:
  - Dodatek do Firefoxa oraz Chrome
    - Dostępna także wersja webowa usługi
- Faktyczne poprawianie działania:
  - Moduł Apache'a
    - Używa zalecanych praktyk Google

PageSpeed (2)


Console
HTML
CSS
Script
DOM
Net
Page Speed
Page Speed Activity
Off
Performance
Resources
Export
Help
Page Speed Score: 64/100
Refresh Analysis
Leverage browser caching
Combine external JavaScript
Minimize DNS lookups
Parallelize downloads across hostnames
Minify JavaScript
Serve resources from a consistent URL
Defer loading of JavaScript
Specify image dimensions
Remove unused CSS
Serve static content from a cookieless domain
Put CSS in the document head
Use efficient CSS selectors
Combine external CSS
Enable compression
Leverage proxy caching
Minify CSS
Minimize redirects
Optimize images
Serve scaled images
Minimize cookie size
Avoid bad requests
Avoid CSS expressions
Optimize the order of styles and scripts
Done

Google Public DNS (1)

- Załadowanie jednej strony to wiele zapytań DNS
  - Statystyczny user 100 zapytań / dzień
- Wyszukiwarki przechowują i cache'ują dane DNS
  - Pomysł Google: udostępnić swoje rozwiązania innym
- Największy publiczny DNS
  - 70 bilionów żądań / dzień
- Czym nie jest Google Public DNS
  - TLD (top level domain)
  - Usługą DNS taką jak np. DynDNS
  - Authoritative name server
  - Usługą blokującą malware
    - Brak blokowania i filtrowania

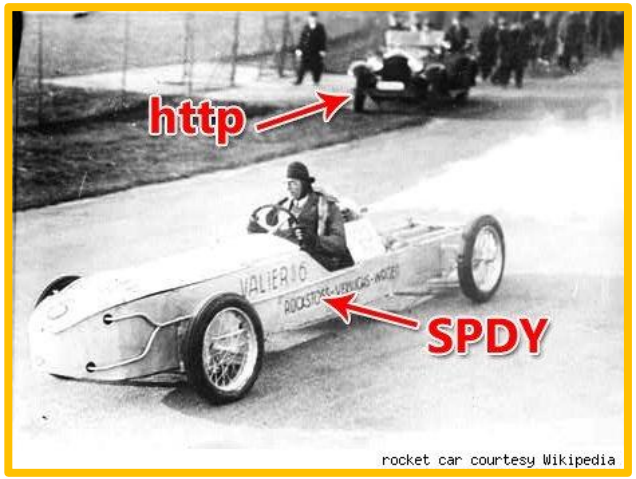


Google Public DNS (2)

- Zalety
  - Wydajność (w większości przypadków)
    - Wiele serwerów nie jest przystosowanych do dużych przepustowości i cache'owania
    - Google posiada bardzo dobre rozwiązania w tej dziedzinie
  - Bezpieczeństwo
    - Zawsze aktualne rozwiązania zabezpieczające m.in. przed „cache poisoning", DoS
    - Zapowiadany atak Anonymous 31. marca 2012
  - Zmniejszenie obciążenia serwerów dostawców
    - Google posiada infrastrukturę na całym świecie
    - Mniejszy koszt dla ISP = niższe ceny usług
- Prywatność?

SPDY (1)

- Jeden z projektów Chromium
- Protokół warstwy aplikacji
  - Warstwa sesji (pomiędzy transportową a prezentacji)
  - Bazowany na TCP
- Przyśpiesza HTTP
  - Rozszerza HTTP – a nie zastępuje
- Kompresja żądań/odpowiedzi HTTP (np. cookie)
- Wiele żądań za pomocą jednego połączenia
  - Dwa wątki:
    - Odebranie połączenia
    - Obsługa połączenia



SPDY (2)

- Możliwość wysłania danych (np. CSS/JS), które użytkownik i tak będzie potrzebował (nie czeka na prośbę o nie)
- Priorytetyzacja
- Kompresja gzip
- Chrome używa go gdy korzystamy z:
  - Wyszukiwarki
  - Gmail'a
  - Reklamy Google
  - Chrome sync
  - Twitter
  - Amazon
  - *Tutaj powoli lista się kończy…*

SPDY (3)

- Ale coraz więcej serwerów i projektów:
  - Jetty 7.6.2
  - node.js
  - curl
  - nginx
- Firefox 11 też obsługuje
  - Obsługa domyślnie wyłączona
- Moduł Apache'a mod_spdy
  - Całkiem świeży: kwiecień 2012
- Pomysł o włączeniu do standardu HTTP 2.0

SPDY (4)

WebP (1)

- Rastrowy format plików graficznych
  - Kompresja stratna i bezstratna
- Mniejszy rozmiar (ten sam indeks SSIM)
  - 28% do PNG
  - 25-34% do JPG
- Limit rozmiaru 16383 pixel'i (14 bitów)
- Features:
  - Obsługa przeźroczystości alpha
    - Dodatkowe 22%
  - Animacje
  - Profil ICC
  - XMP meta dane

WebP (2)

- Używa kodowania VP8 (kodowanie ramek wideo)
  - Przewidujemy kolor bloku na podstawie sąsiednich (3 bloki wyżej,1 na lewo)
  - Kodujemy jedynie różnicę pomiędzy przewidywaną, a rzeczywistą wartością
  - Dużo zer, potem standardowe metody kompresji
- Wszystko zapkowane za pomocą RIFF
- Biblioteka libwebp
  - konwertery linii poleceń: cwebp, dwebp

WebP: Przykłady (1)

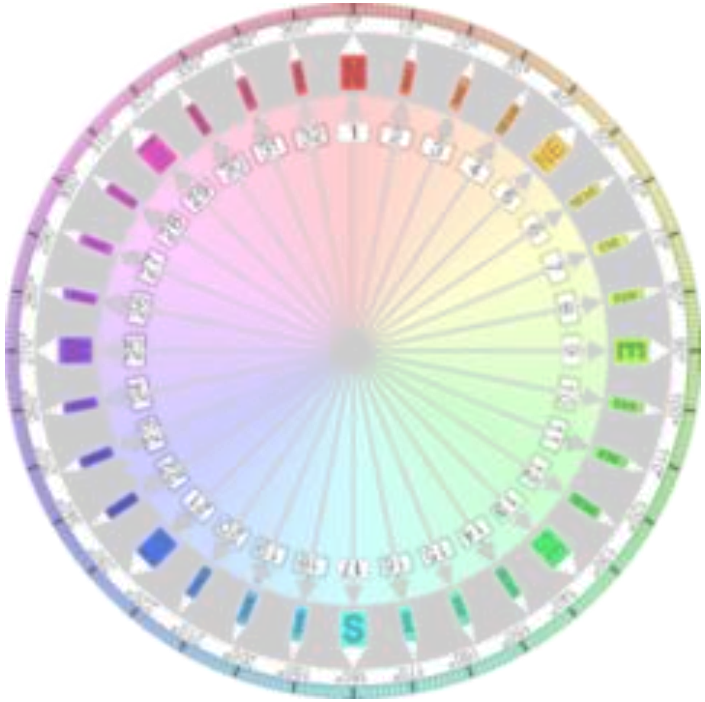
- Zdjęcie PNG , quality 90
- PNG 118,5KB, WebP-bs: 88,1KB, WebP-s: 23,4KB

WebP: Przykłady (2)

- Zwykły PNG , quality 90
- PNG: 40,5KB, WebP-bs: 27,5KB, WebP-s: 17,3KB

WebP: Przykłady (3)

- Przeźroczystość alpha, quality 90
- PNG: 127,1KB, WebP-bs: 100KB, WebP-s: 67,7KB



WebP: Przykłady (4)

- JPEG: 43,84KB, WebP: 29,61KB

WebP: Podsumowanie

- Obsługa
  - Gmail, Picasa, plany do AppEngine
  - Przeglądarki: Chrome, plugin do IE, Opera 11, Android Ice Cream Sandwich
  - Edytory: Pixelmator, ImageMagic, GIMP, Leptonica, XnConvert, ReaConverter, Konvertor, XnView, IrfanView, GDAL
    - plugin do: Photoshop'a, Windows Photo Viewer, MS Office 2010 (Windows Imaging Component)
- Wady:
  - Obrazki są zblurowane bardziej niż np. w JPEG, x264, Theora
    - VP8 używa bloków 4x4, a JPEG 8x8
- Na razie brak sukcesu
  - JPEG-2000, MS JPEG XR też poległy

Bibliografia

- https://developers.google.com/speed
- http://developer.yahoo.com/performance/rules.html
- http://www.yuiblog.com/blog
- http://www.stevesouders.com/blog/2012/02/10/the-performance-golden-rule/
- Wkipedia
- Obrazki z wyszukiwarki Google
- „Wydajność aplikacji webowych (1)”
  - Paweł Bedyński 19/11/2009

Pytania

Cukierki czekają… ☺