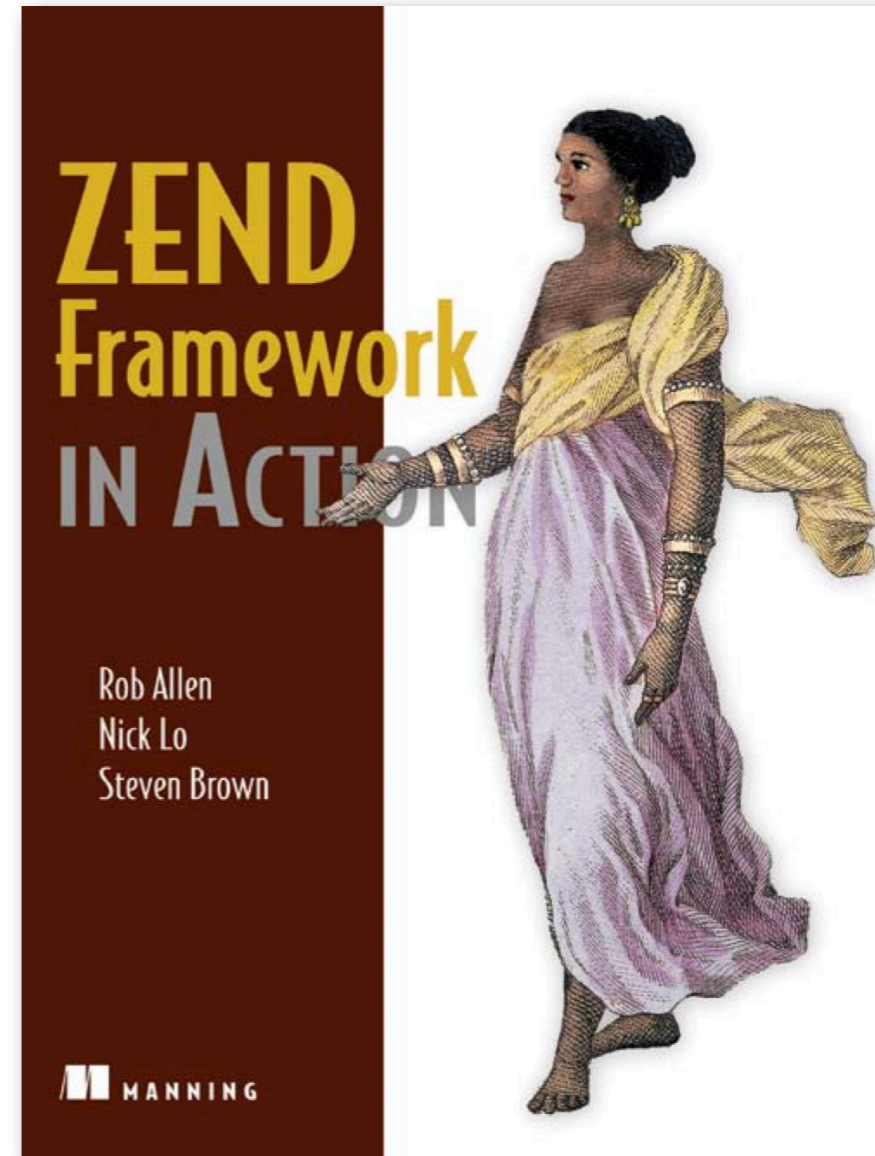


# Caching for performance

Rob Allen

# Rob Allen?



# Caching for performance?

# Measuring performance

# Siege

<http://www.joedog.org/index/siege-home>

edit ~/.siegerc

verbose = false

logging = false *or* logfile=~/.siege.log

concurrent = 5

benchmark = true

# Running siege

```
siege -t 30s http://localhost/info.php
```

Time based: `-t NUMx` where `x = S,M or H`

Request based: `-r NUM`

Add `-c NUM` for the number of concurrent users

# Siege output for info.php

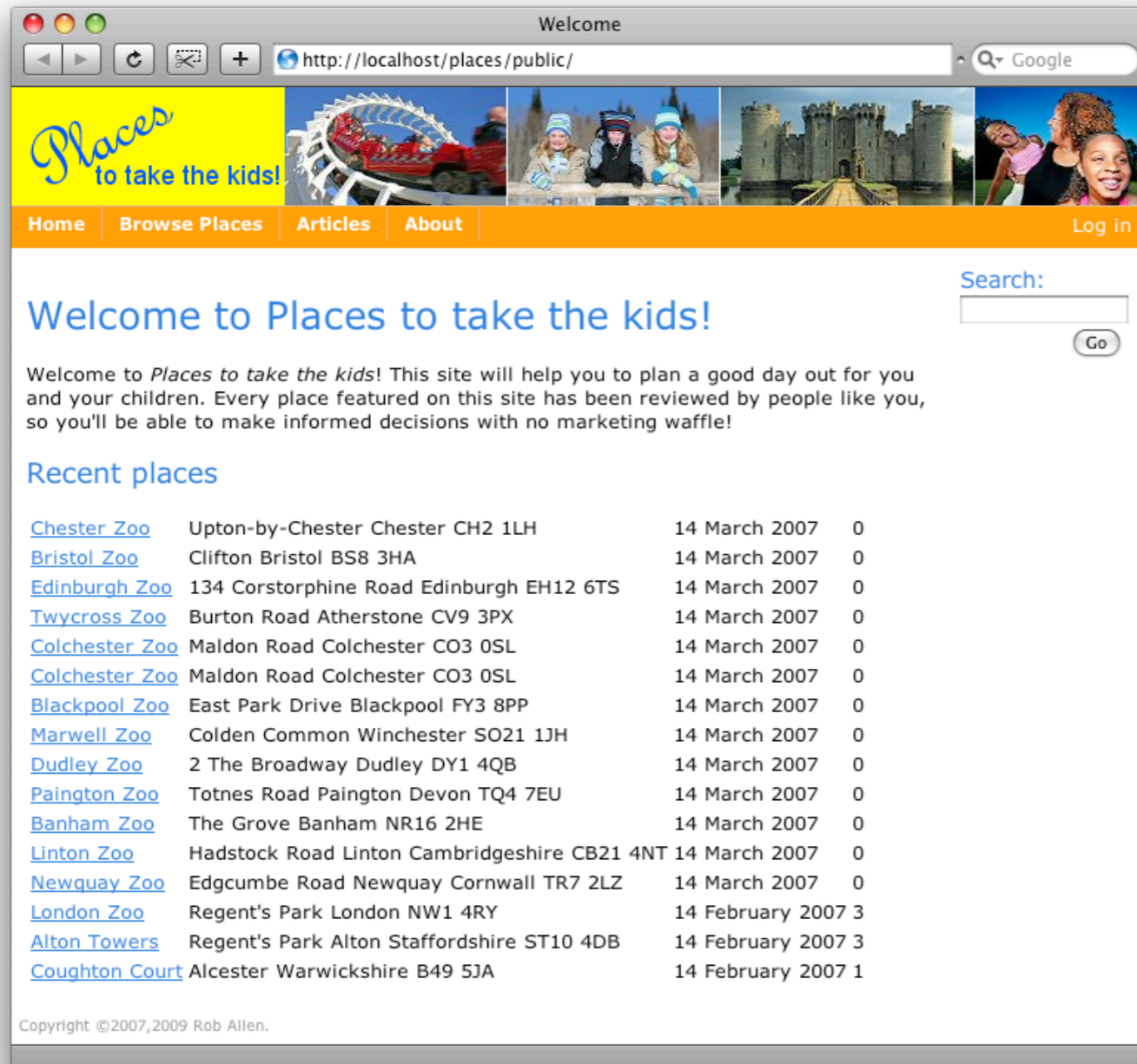
```
$siege -t 30s http://localhost/info.php
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:                    26241 hits
Availability:                    100.00 %
Elapsed time:                    29.45 secs
Data transferred:                1491.16 MB
Response time:                  0.01 secs
Transaction rate:              891.04 trans/sec
Throughput:                      50.63 MB/sec
Concurrency:                    4.99
Successful transactions:         26241
Failed transactions:             0
Longest transaction:             0.04
Shortest transaction:            0.00
```

# Siege output for info.html

```
$siege -t 30s http://localhost/info.html
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:                    233746 hits
Availability:                    100.00 %
Elapsed time:                    29.45 secs
Data transferred:               10228.57 MB
Response time:                  0.00 secs
Transaction rate:             7937.05 trans/sec
Throughput:                     347.32 MB/sec
Concurrency:                    4.91
Successful transactions:        233746
Failed transactions:            0
Longest transaction:            1.08
Shortest transaction:           0.00
```



# A “real” page



- Zend Framework website
- Some CSS and images!
- Zend\_Application
- View uses Zend\_Layout
- Zend\_Db\_Table

# Siege output

## for a real-world page

```
$siege -t 30s http://localhost/places/public/
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:                   375 hits
Availability:                   100.00 %
Elapsed time:                   30.13 secs
Data transferred:               1.88 MB
Response time:                 0.40 secs
Transaction rate:             12.45 trans/sec
Throughput:                     0.06 MB/sec
Concurrency:                    4.94
Successful transactions:        375
Failed transactions:            0
Longest transaction:            1.60
Shortest transaction:           0.15
```

# Profiling

# Xdebug profiler

<http://www.xdebug.org>

Output file format compatible with:

- KCachegrind (Linux)
- WinCacheGrind (Windows)
- MacCallGrind (OS X)
- WebGrind (PHP)

# Xdebug profiler

php.ini:

```
xdebug.profiler_enable = 0
```

```
xdebug.profiler_enable_trigger = 1
```

```
xdebug.profiler_output_dir = "/Users/rob/xdebug"
```

Enable per request:

[http://localhost/places/public/?XDEBUG\\_PROFILE=1](http://localhost/places/public/?XDEBUG_PROFILE=1)

# Simple example

```
<?php
function onesecond() {sleep(1);}
function twoseconds() {sleep(2);}
function threeseconds() {sleep(3);}
function fourseconds() {sleep(4);}
onesecond();
twoseconds();
threeseconds();
fourseconds();
```

[http://localhost/profiletest.php?XDEBUG\\_PROFILE=1](http://localhost/profiletest.php?XDEBUG_PROFILE=1)

# Webgrind display

Number of times called

Time to run this function

Time to run this function and every function it calls

webgrind v1.0  
profiling in the browser

Show 100% of Auto (newest) in milliseconds update  
 Hide PHP functions

/www/profiletest.php  
cachegrind.out.86869 @ 2009-04-29 21:29:30

6 different functions called in 10003 milliseconds (1 runs, 6 shown)

Function	Invocation Count	Total Self Cost	Total Inclusive Cost
{main}	1	0	10003
php::sleep	4	10001	10001
fourseconds	1	0	4000
threeseconds	1	0	3000
twoseconds	1	0	2000
onesecond	1	1	1002

You have the latest version.  
Copyright © 2008 Jacob Oettinger & Joakim Nygård. [webgrind homepage](http://webgrind.org)

# Look for:

- Functions with unexpectedly large call counts
- Functions with large self-times
- Unexpected calls to functions



# Real-world example

webgrind of /www/places/public/index.php

http://localhost/webgrind/

/www/places/public/index.php  
cachegrind.out.3870 @ 2009-05-02 21:29:14

846 different functions called in 476 milliseconds ( runs, 846 shown)

Function	Invocation Count	Total Self Cost	Total Inclusive Cost
{main}	1	1085	476463
Zend_Application->run	1	56	395293
Zend_Application_Bootstrap_Bootstrap->run	1	142	395233
Zend_Controller_Front->dispatch	1	3146	394831
Zend_Controller_Dispatcher_Standard->dispatch	1	301	358470
Zend_Controller_Action->dispatch	1	262	347706
IndexController->indexAction	1	171	297461
Places->fetchLatest	1	742	293909
Places->numReviews	16	1517	277210
Place->numberOfReviews	16	1043	182694
Place->reviews	16	876	181301
Zend_Db_Table_Row_Abstract->findDependentRowset	16	5937	180299

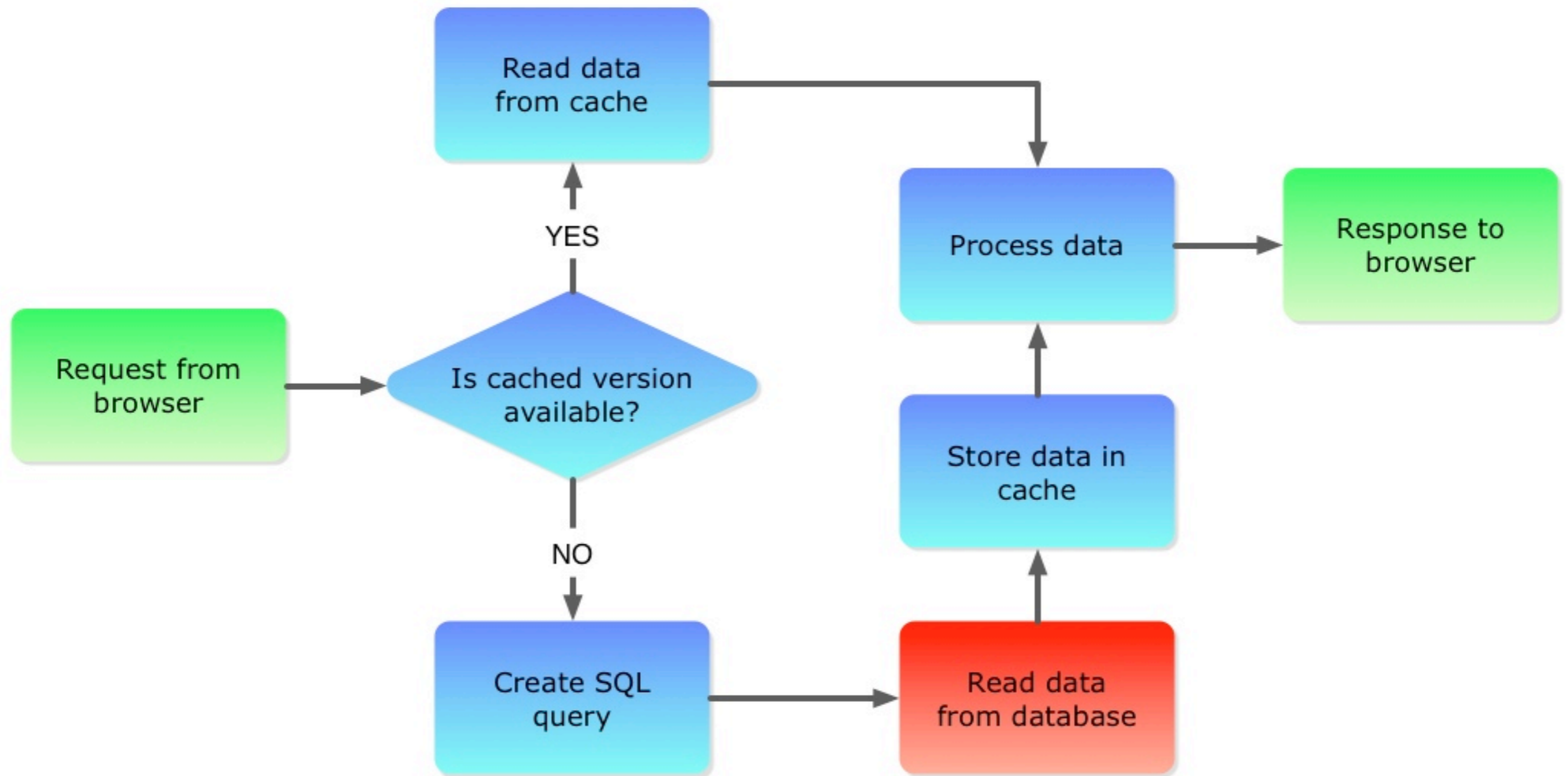
# Principles of caching

1. Don't execute code unless you need to
2. Get data from the fastest place you can
3. Don't get the same data twice

# Save the output of PHP code

AKA Don't execute code unless you need to

# Code caching



# The current code

```
public function fetchLatest($count = 50)
{
    $rows = $this->fetchAll(null, 'date_created DESC', $count);
    $rows = $rowset->toArray();
    foreach($rows as &$row) {
        $row['numberOfReviews'] = $this->numReviews($row['id']);
    }
    return $rows;
}
```

# Adding a cache

Many choices: Zend\_Cache, PEAR::Cache\_Lite, ezcCacheManger and others

1. Set up the cache
2. Wrap cache loading around database query
3. That's it!

# Zend\_Cache set-up

```
function _initCache($cacheDir)
{
    $frontendOptions = array(
        'lifetime' => '7200',
        'automatic_serialization'=>true);
    $backendOptions = array(
        'cache_dir' => $cacheDir);
    $cache = Zend_Cache::factory('Core', 'File',
        $frontendOptions, $backendOptions);
    return $cache;
}
```

# Zend\_Cache in use

```
public function fetchLatest($count = 50)
{
    $cacheId = 'latestNews_'. $count;
    $rows = $this->_cache->load($cacheId);
    if ($rows === false) {
        $rows = $this->fetchAll(null, 'date_created DESC', $count);
        $rows = $rowset->toArray();
        foreach($rows as &$row) {
            $row['numberOfReviews'] = $this->numReviews($row['id']);
        }
        $this->_cache->save($rows, $cacheId, array('places'));
    }
    return $rows;
}
```

Unique id

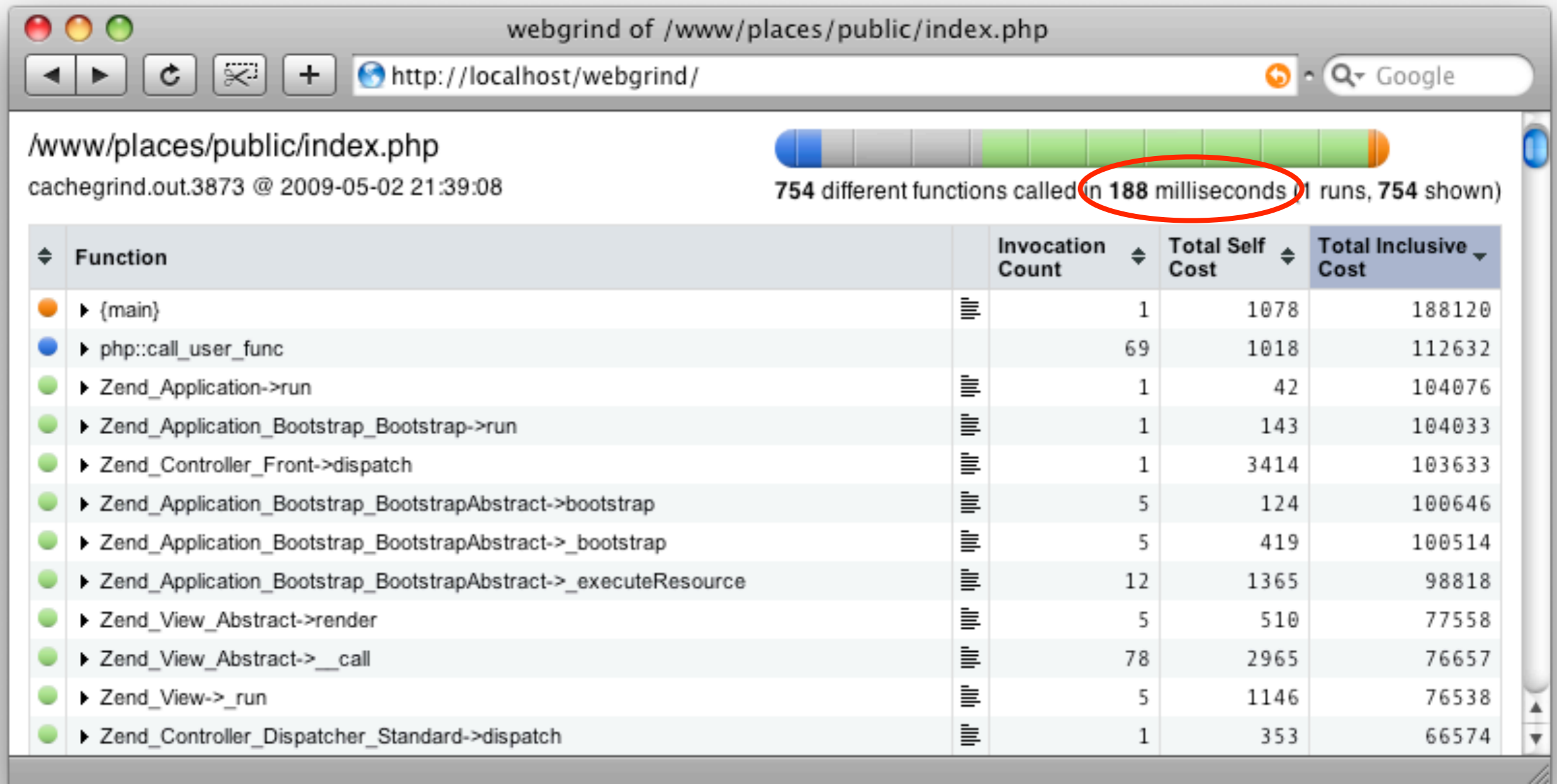
Store to cache

Tag

Re-use existing code



# Re-profile



# Siege output

for code cached real page

```
$siege -t 30s http://localhost/places/public/
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:          875 hits
Availability:          100.00 %
Elapsed time:          29.80 secs
Data transferred:     4.51 MB
Response time:        0.17 secs
Transaction rate:    29.36 trans/sec
Throughput:           0.15 MB/sec
Concurrency:          4.97
Successful transactions: 875
Failed transactions:   0
Longest transaction:  3.73
Shortest transaction: 0.06
```

# Emptying Zend\_Cache

```
public function cleanCacheByTag($tag)
{
    return $this->_cache->clean(
        Zend_Cache::CLEANING_MODE_MATCHING_TAG,
        array($tag));
}

public function cleanAllCache()
{
    return $this->_cache->clean(Zend_Cache::CLEANING_MODE_ALL);
}
```

# Page caching

Why stop at just the database?  
Let's cache the entire page!

# Setup a page cache

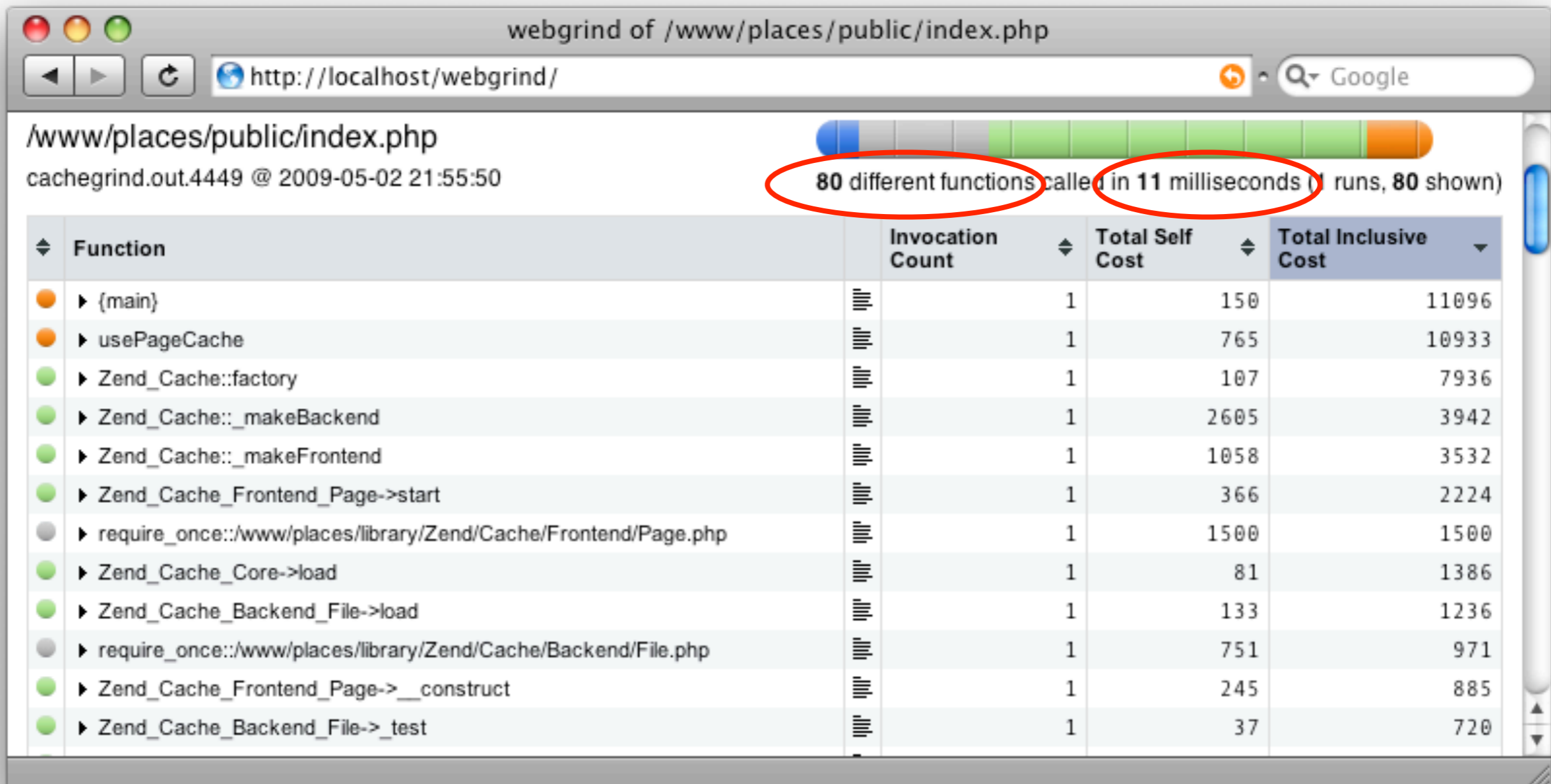
```
function usePageCache($cacheDir)
{
    $frontendOptions = array(
        'lifetime' => 3600,
        'default_options' => array('cache' => false),
        'regexps' => array(
            '^/$' => array('cache' => true),
            '^/places/' => array('cache' => true),
        )
    );
    $backendOptions = array('cache_dir' => $cacheDir);
    $cache = Zend_Cache::factory('Page', 'File',
        $frontendOptions, $backendOptions
    );
    $cache->start();
    return $cache;
}
```

# Siege output

## for a page cache

```
$siege -t 30s http://localhost/places/public/
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:                  10738 hits
Availability:                  100.00 %
Elapsed time:                   29.90 secs
Data transferred:              29.27 MB
Response time:                 0.01 secs
Transaction rate:             359.13 trans/sec
Throughput:                    0.98 MB/sec
Concurrency:                   4.94
Successful transactions:       10738
Failed transactions:           0
Longest transaction:           1.46
Shortest transaction:          0.00
```

# Profile for a page cache

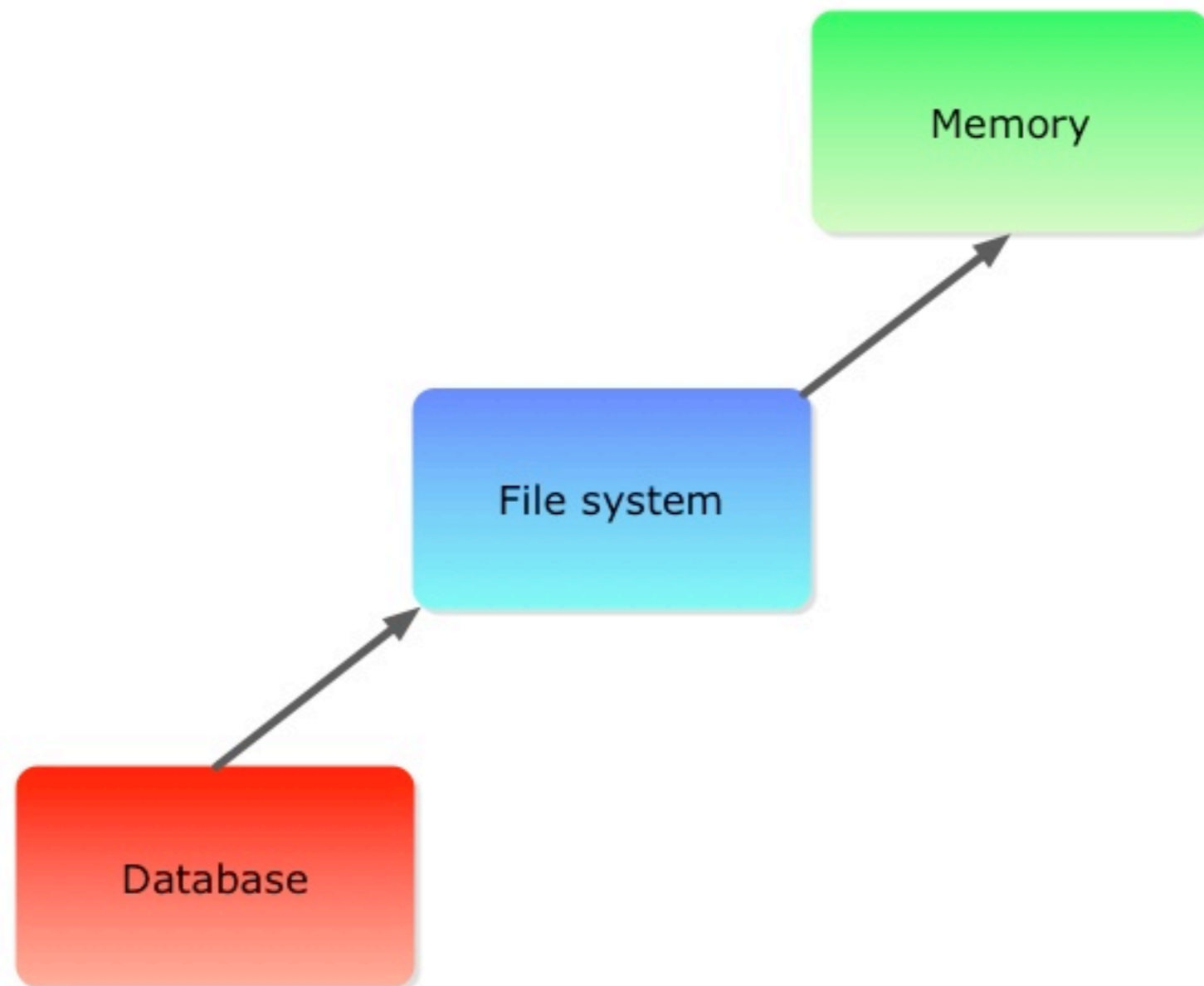


# Choose the right storage layer

AKA Get data from the fastest place you can



# Where is your cache?



# memcached

Cache to memory rather than disk.

memcached server:

```
./configure && make && make install  
memcached -d -l 10.0.148.121 -m 512
```

PECL memcache extension:

```
pecl install memcache
```

php.ini:

```
extension=memcache.so
```

# Zend\_Cache set-up

```
function _initCache($cacheDir)
{
    $frontendOptions = array(
        'lifetime' => '7200',
        'automatic_serialization'=>true);
    $backendOptions = array(
        'servers' => array(array('host' => '10.16.148.121')));
    $cache = Zend_Cache::factory('Core', 'Memcached',
        $frontendOptions, $backendOptions);
    return $cache;
}
```

# Siege output

for memcached code cached page

```
$siege -t 30s http://localhost/places/public/
```

```
** SIEGE 2.68
```

```
** Preparing 5 concurrent users for battle.
```

```
The server is now under siege...
```

```
Lifting the server siege... done.
```

```
Transactions: 1824 hits
```

```
Availability: 100.00 %
```

```
Elapsed time: 30.29 secs
```

```
Data transferred: 4.97 MB
```

```
Response time: 0.08 secs
```

```
Transaction rate: 60.22 trans/sec
```

```
Throughput: 0.16 MB/sec
```

```
Concurrency: 4.99
```

```
Successful transactions: 1824
```

```
Failed transactions: 0
```

```
Longest transaction: 2.88
```

```
Shortest transaction: 0.00
```

Don't forget that PHP  
is interpreted...

AKA: Don't execute code unless you need to

# Op-code caching

Cache your php byte code using APC, Zend Optimizer+, XCache, eAccelerator, etc...

APC:

```
pecl install apc
```

php.ini:

```
extension=apc.so
```

# Siege output

for real page (no other caching) & APC

```
$siege -t 30s http://localhost/places/public/
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:                    574 hits
Availability:                    100.00 %
Elapsed time:                    29.42 secs
Data transferred:                3.14 MB
Response time:                  0.25 secs
Transaction rate:              19.51 trans/sec
Throughput:                      0.11 MB/sec
Concurrency:                     4.96
Successful transactions:          574
Failed transactions:              0
Longest transaction:             1.05
Shortest transaction:            0.10
```

# Siege output

for code cached page with APC

```
$siege -t 30s http://localhost/places/public/
** SIEGE 2.68
** Preparing 5 concurrent users for battle.
The server is now under siege...
Lifting the server siege...      done.
Transactions:          8766 hits
Availability:          100.00 %
Elapsed time:          30.25 secs
Data transferred:     23.89 MB
Response time:        0.02 secs
Transaction rate:    289.65 trans/sec
Throughput:           0.79 MB/sec
Concurrency:          4.98
Successful transactions: 8766
Failed transactions:   0
Longest transaction:  0.63
Shortest transaction: 0.00
```



# Siege output

for a page cache with APC

```
$siege -t 30s http://localhost/places/public/
```

```
** SIEGE 2.68
```

```
** Preparing 5 concurrent users for battle.
```

```
The server is now under siege...
```

```
Lifting the server siege... done.
```

```
Transactions: 94834 hits
```

```
Availability: 100.00 %
```

```
Elapsed time: 29.35 secs
```

```
Data transferred: 258.53 MB
```

```
Response time: 0.00 secs
```

```
Transaction rate: 3231.14 trans/sec
```

```
Throughput: 8.81 MB/sec
```

```
Concurrency: 4.84
```

```
Successful transactions: 94834
```

```
Failed transactions: 0
```

```
Longest transaction: 0.56
```

```
Shortest transaction: 0.00
```

# The browser's cache

AKA Don't get the same data twice

# HTTP headers

- Expires
- Cache-Control

# Expires

`Expires: Sat, 13 June 2019 14:45:00 GMT`

- Use for static components (CSS, JS, images)
- Encode version into filename
- Set via Apache's `ExpiresByType` directive

# ExpiresByType

```
<IfModule mod_expires.c>
```

```
ExpiresActive On
```

```
ExpiresByType text/css "now plus 2 years"
```

```
ExpiresByType text/javascript "now plus 2 years"
```

```
ExpiresByType image/jpeg "now plus 2 years"
```

```
ExpiresByType image/gif "now plus 2 years"
```

```
ExpiresByType image/png "now plus 2 years"
```

```
ExpiresByType application/x-shockwave-flash "now plus 2  
years"
```

```
</IfModule>
```

# Cache-Control

Cache-Control: max-age=3600, must-revalidate

- Set relative expiry time
- Configure expiry of proxy caches separately
- Ability to prevent caching completely

# Validation headers

```
Last-Modified: Sun, 31 May 2009 10:21:09 GMT  
ETag: 49705dbc03db2832844362b3950bfd0
```

- Browser checks before serving cached copy
- ETag must be quoted
- Respond to: `HTTP_IF_MODIFIED_SINCE` & `HTTP_IF_NONE_MATCH` (in `$_SERVER`)  
Send back header( 'HTTP/1.0 304 Not Modified' );

# Rob's Recommendations

1. Run an op code cache
2. Cache slow code (usually db calls)
3. Cache CSS/JS/images in the browser



# Thank you

Please provide feedback on this talk: <http://joind.in/1040>