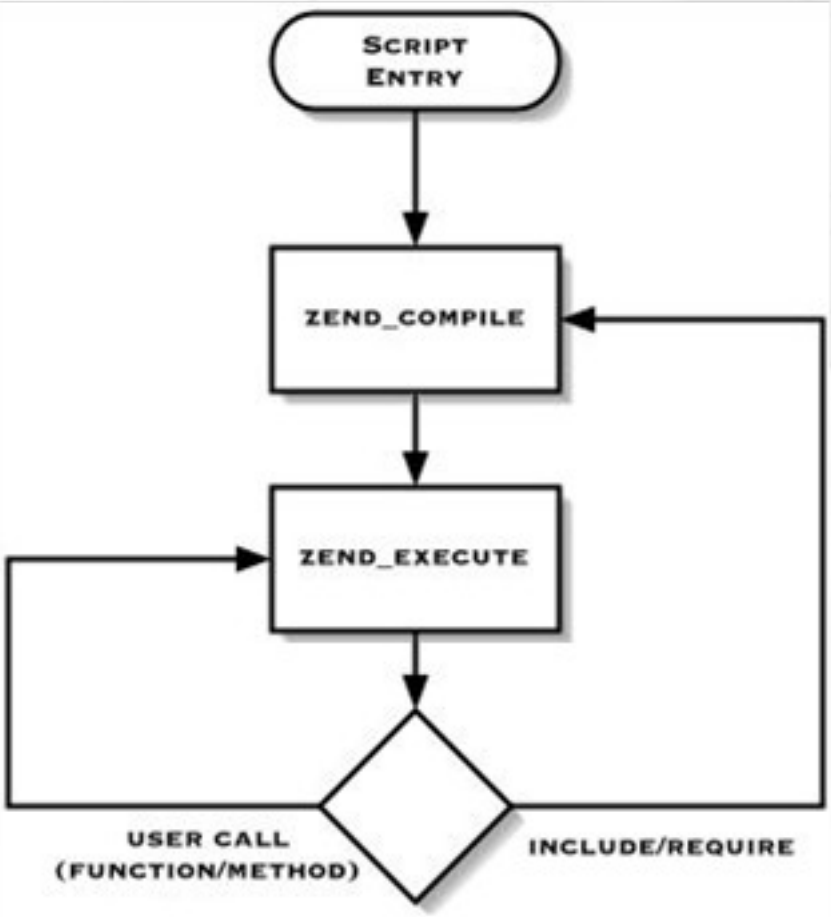
PHP & Performance

By: Ilia Alshanetsky

Compiler/Opcode Caches

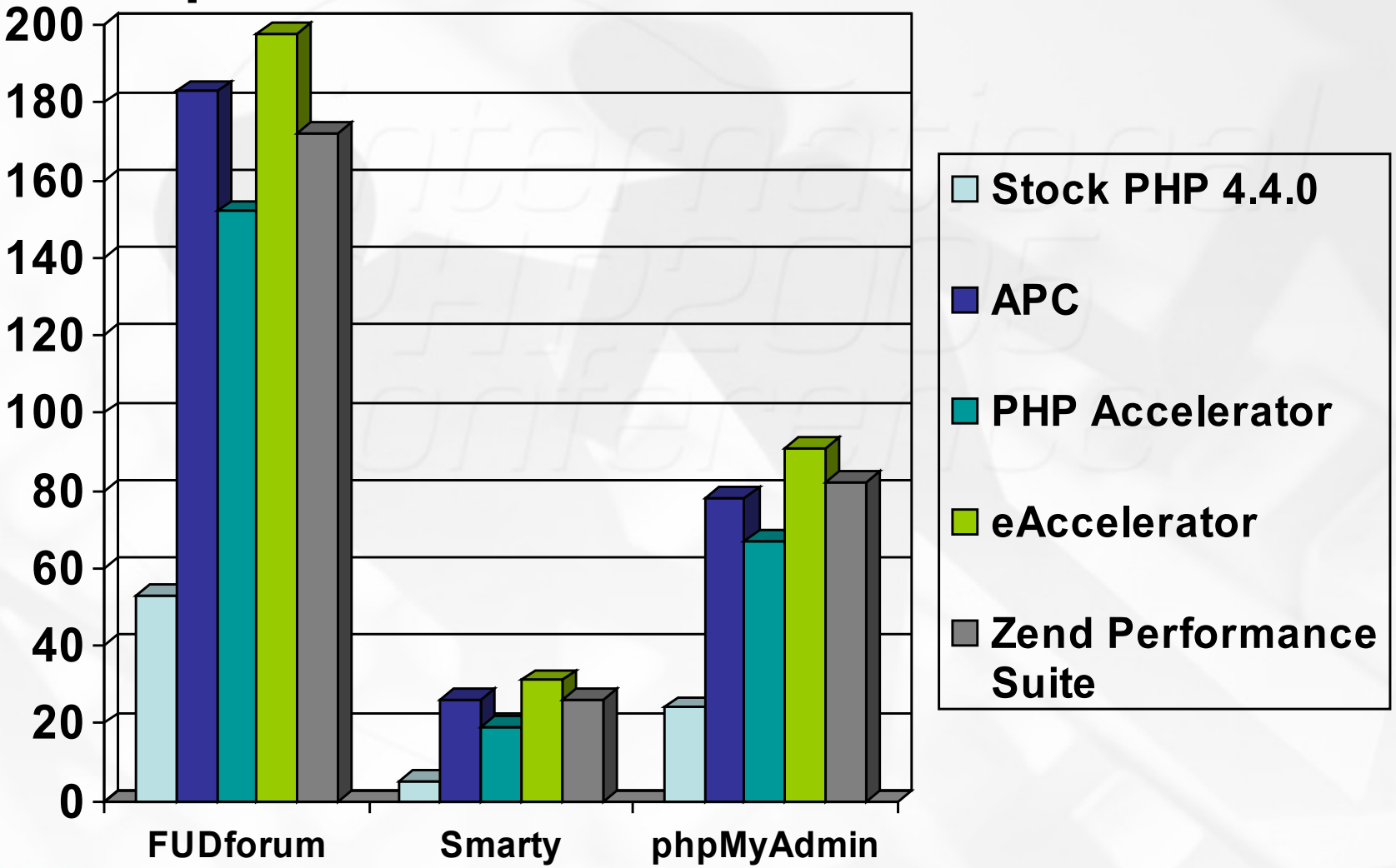


- This cycle happens for every include file, not just for the "main" script.
- Compilation can easily consume more time than execution.

Compiler/Opcode Caches

- Each PHP script is compiled only once for each revision.
- Reduced File IO, opcodes are being read from memory instead of being parsed from disk.
- Opcodes can optimised for faster execution.

Opcode Cache Benchmarks



Compiler Optimisations

- For absolute maximum performance, ensure that all of the software is compiled to take advantage of the available hardware.
  - Enable all compiler optimizations with `-O3`
  - Tune the code to your CPU via `-march -mcpu`
  - CPU specific features `-msse -mmmx -mfpmath=sse`
  - Drop debug data `-fomit-frame-pointer`

```
export CFLAGS="-O3 -msse -mmmx -march=pentium3 \
-mcpu=pentium3 -funroll-loops -mfpmath=sse \
-fomit-frame-pointer"
```

Reduce Binary/Library Size

- Eliminate waste by removing debugging symbols from object files using the `strip` utility.
  - Saves disk space.
  - Reduces memory needed to load the binary.
- Stripping PHP binaries and/or modules on average makes them 20-30% smaller.
- Very useful for CLI/CGI PHP binaries.

Web Server: File IO

- Keep `DirectoryIndex` file list as short as possible.
- Whenever possible disable `.htaccess` via `AllowOverride none`.
- Use `Options FollowSymLinks` to simplify file access process in Apache.
- If logs are unnecessary disable them.
- If logging is a must, log everything to 1 file and break it up during the analysis stage.

Web Server: Syscalls

- Syscall is function executed by the Kernel. The goal is to minimise the number of these calls needed to perform a request.
  - Do not enable `ExtendedStatus`.
  - For Deny/Allow rules use IPs rather then domains.
  - Do not enable `HostnameLookups`.
  - Keep `ServerSignature` off

Web Server: KeepAlive

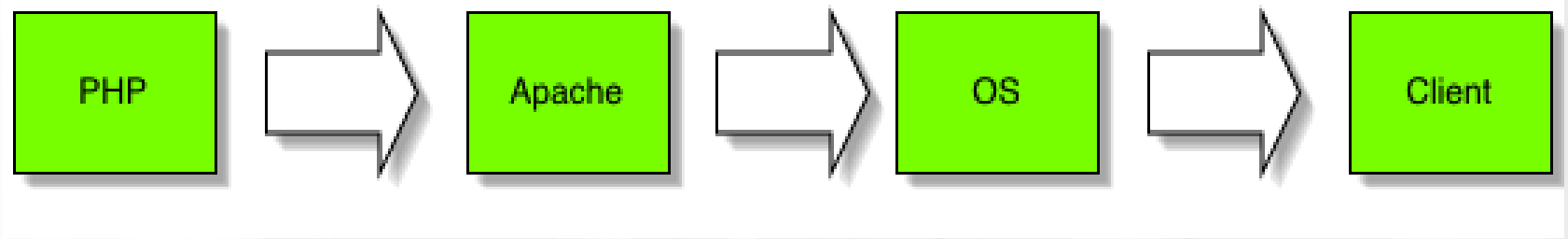
- In theory `KeepAlive` is supposed to make things faster, however if not used carefully it can cripple the server.
  - In Apache set `KeepAlive` timeout, `KeepAliveTimeout` as low as possible. Suggested value: 10 seconds.
  - If the server is only serving dynamic requests, disable `KeepAlive` all together.

Static Content Serving

- While Apache is great for dynamic requests, static requests can be served WAY FASTER by other web servers.
  - `lighttpd`
  - `Boa`
  - `Tux`
  - `thttpd`

- For static requests these servers can be 300-400% faster then Apache.

Matching Your IO Sizes



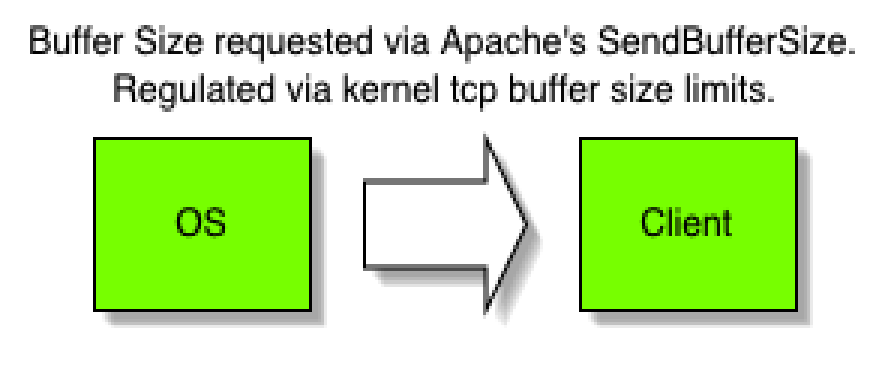
- The goal is to pass off as much work to the kernel as efficiently as possible.
- Optimizes PHP to OS Communication
- Reduces Number Of System Calls

Output Buffering

- Efficient
- Flexible
- In your script, with `ob_start()`
- Everywhere, with `output_buffering` = On
- Improves browser's rendering speed

Output Buffering

- The idea is to hand off entire page to the kernel without blocking.



- In Apache:

```
SendBufferSize = PageSize
```

Network Buffer Sizing Cont.

OS (Linux)

```
/proc/sys/net/ipv4/tcp_wmem
4096    16384   maxcontentsize
min      default      max

/proc/sys/net/ipv4/tcp_mem
(maxcontentsize * maxclients) / pagesize
```

Be careful on low memory systems!

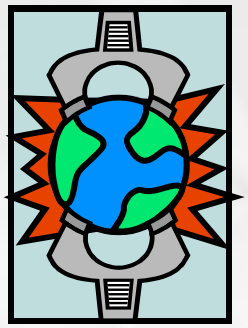
Bandwidth Optimizations

- **Less output is good because…**
  - Saves server bandwidth (saves $$ too).
  - Reduces server resource usage (CPU/Memory/Disk)
  - Pages load faster for clients.

  - Reduces network IO high traffic sites, where it is the primary bottleneck in most cases.
  - Reduces probability of partial page downloads.

Content Compression

- Most browser support retrieval of compressed pages and then decompressing them prior to rendering.
- Compressed pages are on average are 7-10 times smaller.
  - Implementations:
    - Apache 1 (`mod_gzip / mod_deflate`)
    - Apache 2 (`mod_deflate`)
    - PHP
      - From PHP configuration `zlib.output_compression=1`
      - From inside the script `ob_start("ob_gzhandler")`



- Compression will utilize 3%-5% of CPU.

Content Reduction

- Use post-processor such as the `tidy` extension to eliminate white-space and any unnecessary components from final HTML output.

```
<?php
$o = array("clean" => true,
  "drop-proprietary-attributes" => true,
  "drop-font-tags" => true,
  "drop-empty-paras" => true,
  "hide-comments" => true,
  "join-classes" => true,
  "join-styles" => true
);

$tidy = tidy_parse_file("php.html", $o);
tidy_clean_repair($tidy);
echo $tidy;
?>
```

```
clean=1
drop-proprietary-attributes=1
drop-font-tags=1
drop-empty-paras=1
hide-comments=1
join-classes=1
join-styles=1
```

```
<?php
ini_set("tidy.default_config",
   /path/to/compact_tidy.cfg");
ini_set("tidy.clean_output", 1);
?>
```

Tuning PHP Configuration

- register_globals = Off **
- magic_quotes_gpc = Off
- expose_php = Off
- register_argc_argv = Off
- always_populate_raw_post_data = Off **
- session.use_trans_sid = Off **
- session.auto_start = Off **
- session.gc_divisor = 1000 or 10000
- output_buffering = 4096

** Off by default

Profiling & Benchmarking

- Identify Bottlenecks
- Track Resource Usage
- Generate Call Trees
- Create Progress Tracking Data

Helpful Tools

- Benchmarking content serving
  - Apache Bench (http://apache.org)
  - httperf (http://freshmeat.net/projects/httperf/)
- PHP Profilers
  - DBG (http://dd.cron.ru/dbg/)
  - APD (pear install apd)
  - Xdebug (http://xdebug.org/)

Web Server Testing

```
Server Software:                Apache
Server Hostname:                localhost
Server Port:                    80
Document Path:                  /php.php
Document Length:                46844 bytes

Concurrency Level:              10
Time taken for tests:           0.265 seconds
Complete requests:              100
Failed requests:                0
Broken pipe errors:             0
Total transferred:              5077082 bytes
HTML transferred:               5061168 bytes
Requests per second:            377.36 [#/sec] (mean)
Time per request:               26.50 [ms] (mean)
Time per request:               2.65 [ms] (mean, across all concurrent requests)
Transfer rate:                  19158.80 [Kbytes/sec] received

Connection Times (ms)  min  mean[+/-sd] median   max
Connect:                 0    8    5.2      8    20
Processing:             22   16    5.2     16    25
Waiting:                 3   14    5.5     14    24
Total:                  22   24    3.2     24    44
```

PHP Profilers (APD)

- PHP profilers come in a form of Zend modules that sit around the executor and collect information about the executed functions & methods.
- Installation Procedure
  - `pear install apd`
  - Modify `php.ini` with

```
zend_extension=/path/to/apd.so
```

Generating A Trace

- Profiling of a script starts from the point when the `apd_set_pprof_trace()` function is called.
  - All code executed prior, will not be profiled.

```
$parts = preg_split("!\s!", "a b c");
function test(&$var) {
        $var = base64_encode(trim($var));
}
apd_set_pprof_trace();
array_walk($parts, 'test');
```

- Use the `auto_append_file` php.ini setting to activate profiling for an entire application.

Understanding The Trace

Real %Time	User (excl/cumm)	System (excl/cumm)	secs/ (excl/cumm)	cumm Calls	call	s/call	Name
82.4	0.00 0.00	0.00 0.00	0.00 0.00	1	0.0007	0.0007	apd_set_pprof_trace
10.2	0.00 0.00	0.00 0.00	0.00 0.00	3	0.0000	0.0000	trim
4.3	0.00 0.00	0.00 0.00	0.00 0.00	3	0.0000	0.0000	base64_encode
1.9	0.00 0.00	0.00 0.00	0.00 0.00	3	0.0000	0.0000	test
0.6	0.00 0.00	0.00 0.00	0.00 0.00	1	0.0000	0.0001	array_walk
0.6	0.00 0.00	0.00 0.00	0.00 0.00	1	0.0000	0.0008	main

Tuning PHP File Access

- Whenever opening files or including scripts into the main script try to specify a full path or at least an easily resolvable partial path.

Inefficient Approach:

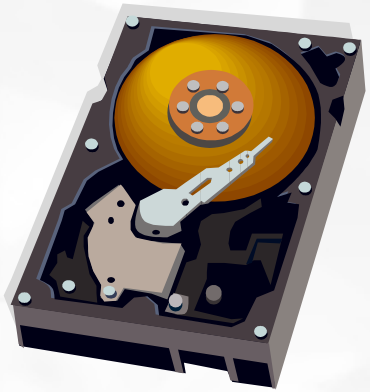
```
<?php include "file.php"; ?>
```

Performance Friendly Approach:

```
<?php
include "/path/to/file.php";
// or
include "./file.php";
?>
```

Drive Tuning

- Hard-drive is in most cases the slowest part of the system, yet all the data eventually comes from it.
- By adjust the drive configuration parameters you can help your OS get the most out of it.



Drive Tuning Parameters

- Use the `hdparm` utility to adjust settings.
  - `-c1` - set IDE 32-bit I/O setting
  - `-d1` - enable DMA
  - `-u1` - enable IRQ unmasking
  - `-m16` - turn on multicount
  - `-X 34|66|100|133` - transfer mode
- Benchmark the affect of the changes using:
  - `hdparm -tT /dev/[drive]`

RAM Disk

- One way to accelerate File IO operations is by moving the files and directories to a RAM disk.
- On Linux this is extremely simple to do using via `tmpfs`.

```
# Speed Up /tmp Directory
mount --bind -ttmpfs /tmp /tmp

# Accelerate Scripts Directory
mount --bind -ttmpfs /home/webroot /home/webroot
```

Session Storage

- PHP's session extension by default stores each session inside a separate file.
  - Many files in one directory reduce access speed.
    - Assign each user their own session directory
    - Split sessions into multiple directories
      session.save_path = "N;/path"
  - File system is slow, especially for random access.
    - Use alternate session storage mechanism like shared memory via "mm" session handler.

Regular Expressions

- While very useful tool for string manipulation, regex leave much to be desired when it comes to performance.

```
// Slow
if (preg_match("!^foo_!i", "FoO_")) { }
// Much faster
if (!strncasecmp("foo_", "FoO_", 4)) { }

// Slow
if (preg_match("![a8f9]!", "sometext")) { }
// Faster
if (strpbrk("a8f9", "sometext")) { }
```

Optimizing str_replace()

- The `str_replace()` function in PHP can be slow, due it's duplication of data even if no replacement is being performed.

```
$src_str = file_get_contents("BIG_FILE");
$src = array('abc', 123, 'text');
$dst = array('cba', 321, 'txet');

// eliminate unnecessary replacement attempts
foreach ($src as $k => $v)
        if (strpos($src_str, $src) === FALSE)
                unset($src[$k], $dst[$k]);

if ($src) $new_str = str_replace($src, $dst, $src_str);
```

strtr() vs str_replace()

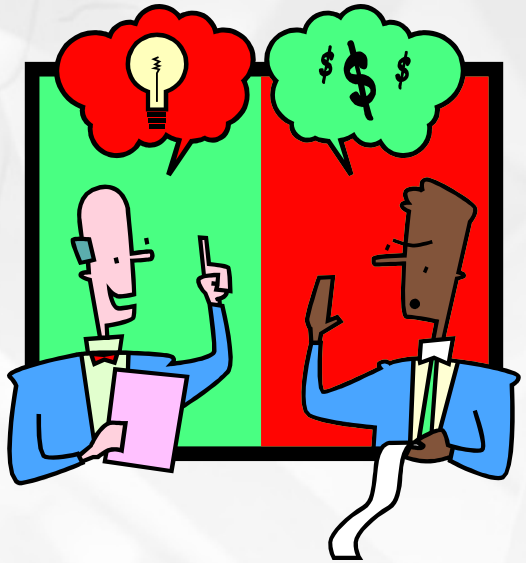
```
$src_str = file_get_contents("some_big_file");

$src = array('abc', 123, 'text');
$dst = array('cba', 321, 'txet');

$s = microtime(1);
for ($i = 0; $i < 10000; $i++)
        str_replace($src, $dst, $src_str);
$e = microtime(1);
echo ($e - $s) . "\n"; // 5.69 seconds

$new_rep = array_combine($src, $dst);

$s = microtime(1);
for ($i = 0; $i < 10000; $i++)
        strtr($src_str, $new_rep);
$e = microtime(1);
echo ($e - $s) . "\n"; // 54.42 seconds
```



Don’t Reinvent the Wheel

- PHP includes hundreds of functions, always check if the desired operation is already natively implemented.

```
$data = '';
$fp = fopen("some_file", "r");
while ($fp && !feof($fp)) {
        $data .= fread($fp, 1024);
}
fclose($fp);

// vs the much simpler & faster
$data = file_get_contents("some_file");
```

Handy New Functions

- `file_put_contents()`
  - Append data to files or create new files in one shot.
- `microtime()` and `gettimeofday()`
  - Return floats when passed `TRUE` as a 1st argument.
- `mkdir()`
  - Can create directory trees, when 3rd argument is set to `TRUE`.

Handy New Functions

- `convert_uuencode,convert_uudecode`
  - Fast UU encoding/decoding mechanism.
- `http_build_query()`
  - Build GET/POST query based on associated array.
- `substr_compare()`
  - strcmp/strncasecmp/etc… from an offset.
- `array_walk_recursive()`
  - Recursively iterate through an array.

Reference Tricks

- References can be used to simply & accelerate access to multi-dimensional arrays.

```
$a['b']['c'] = array();
// slow 2 extra hash lookups per access
for($i = 0; $i < 5; $i++)
        $a['b']['c'][$i] = $i;

// much faster reference based approach
$ref =& $a['b']['c'];
for($i = 0; $i < 5; $i++)
        $ref[$i] = $i;
```

What Is Caching?

Caching is the recognition and exploitation of the fact that most "dynamic" data does not change every time you request it.

Content Caching

```
function cache_start()
{
    global $cache_file_name;

    // a superbly creative way for creating cache files
    $cache_file_name = __FILE__ . '_cache';

    $age = 600; // default cache age

    // check if cache exists and is valid
    if (@filemtime($cache_file_name) + $age > time()) {
        // Yey! cache hit, output cached data and exit
        readfile($cache_file_name);
        unset($cache_file_name); exit;
    }
    ob_start(); // nothing in cache or cache is too old
}
```

Content Caching

```
function cache_end()
{
  global $cache_file_name;
  // nothing to do
  if (empty($cache_file_name)) return;
  // fetch output of the script
  $str = ob_get_clean();
  echo $str; // output data to the user right away
  // write to cache
  fwrite(fopen($cache_file_name.'_tmp', "w"), $str);
  // atomic write
  rename($cache_file_name.'_tmp', $cache_file_name);
}
cache_start();
// set cache termination code as the exit handler
// this way we don't need to modify the script
register_shutdown_function("cache_end");
```

Content Caching

```
<?php
require "./cache.php"; // our cache code
// Simple guestbook script.
$db = new sqlite_db("gb.sqlite");
$r = $db->array_query("SELECT * FROM guestbook");
foreach ($r as $row)
 echo $r->user . ' wrote on ' .
      date("Ymd", $r->date) . ":<br />\n" .
      $r->message . "<hr /><hr />";
?>
```

- Implementing cache without modifying the script

```
# Add to .htaccess
php_value auto_prepend_file "/path/to/cache.php"

# Or to virtual host entry in httpd.conf
php_admin_value auto_prepend_file "/path/to/cache.php"
```

On-Demand Caching

Set up a 404 error handler in `.htaccess`:

```
ErrorDocument 404 /index.php
DirectoryIndex index.php
```

```
if (!empty($_SERVER['REDIRECT_URL'])) {
 // This is the requested page that caused the error
 $current_page = get_page_name($_SERVER['REDIRECT_URL']);
}


// content generation
if (!FORCE_DYNAMIC) {
 echo $contents = ob_get_clean();
 file_put_contents($lang."/".$current_page.".html", 'w');
}
```

Header Caching

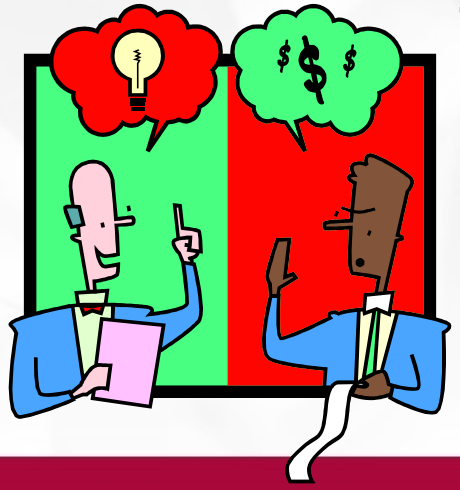
- Most modern browsers can and will cache page's data given the "permission" to do so.
  - For dynamic (PHP) generated pages, caching headers are not sent automatically.
  - However, you can send them yourself, telling browsers to cache page data.

```
header("Expires: ".gmdate("r", time() + 600)." GMT");
```

Pros and Cons of Caching

- Increase in performance
- Reduces resource usage
- Improved user experience.

- Architectural Complexity
- Potential for Stale or Inconsistent Data
- Reduced output control.



SQL & Performance

Most large applications will end up using databases for information storage. Improper use of this resource can lead to significant and continually increasing performance loss.

Check Your Queries

- Most databases offers tools for analyzing query execution.

SLOW

```
EXPLAIN select * from users where login LIKE '%ilia%';
+----------+------+---------------+------+---------+------+------+------------+
| table    | type | possible_keys | key  | key_len | ref  | rows | Extra      |
+----------+------+---------------+------+---------+------+------+------------+
| mm_users | ALL  | NULL          | NULL |    NULL | NULL | 27506 | where used |
+----------+------+---------------+------+---------+------+------+------------+
```

FAST

```
EXPLAIN select * from users where login LIKE 'ilia%';
+----------+-------+---------------+-------+---------+------+------+------------+
| table    | type  | possible_keys | key   | key_len | ref  | rows | Extra      |
+----------+-------+---------------+-------+---------+------+------+------------+
| mm_users | range | login         | login |      50 | NULL |    2 | where used |
+----------+-------+---------------+-------+---------+------+------+------------+
```

Bitwise Option Packing

- Rather then creating a column for every Boolean option, you can pack 32 of them into a single integer field.

```
CREATE TABLE users (
        is_active INT,
        is_banned INT,
        is_admin INT,
        ...
);
```

```
CREATE TABLE users (
        user_opt INT,
        ...
);

user_opt & 1 // active
user_opt & 2 // banned
user_opt & 4 // admin
```

Database Systems

PHP can work with many database systems. A poorly chosen system can add significant overhead to the application.

Declare Your Statics!

- When object properties and methods will only be accessed statically, be sure to declare them as `static`.
  - Improved performance (50-75%).
  - Clearer Code.

KISS = Performance

- The simpler the code, the faster it runs, it really is that simple.
  - Syntactic sugar.
  - Unnecessary wrappers.
  - Wrapping one liners in functions.
  - OO for the sake of OO.

Questions

Resources

- http://ilia.ws/ (These Slides) http://pecl.php.net/apc (APC)
- http://pecl.php.net/apd (APD)
- http://xdebug.org (Xdebug)
- http://www.lighttpd.net (Lighttpd)
- http://sysoev.ru/en/ (mod_deflate for Ap1)

<?php include “/book/plug.inc”; ?>