Building Scalable PHP Applications

George Schlossnagle

What is Scalability

- Systemwide Metrics
  - Ability to accept increased traffic in a graceful and controlled manner
  - Efficient delivery of content and services.
- Individual Metrics
  - Minimal Interdependencies
  - Low Latency
  - Fast Delivery
- This is often used synonymously with 'performant', but they aren't necessarily the same.

Why Performance Is Important to You

- Efficient resource utilization
- More satisfying user experience
- Easier to manage

Why PHP

(or: Shouldn't we be using Java for this?)

- PHP is a completely runtime language.
- Compiled, statically typed languages are faster.

BUT

- Most bottlenecks are not in user code.
- PHP's heavy lifting is all done in C.
- PHP is fast to learn.
- PHP is fast to write.
- PHP is easy to extend.

Knowing When To Start

Premature optimization is the root of all evil
- Donald Knuth

- Without direction and goals, optimization will only make your code obtuse with a minimal chance of actual improvement.
- Design for easy refactoring, but follow the YAGNI principle.

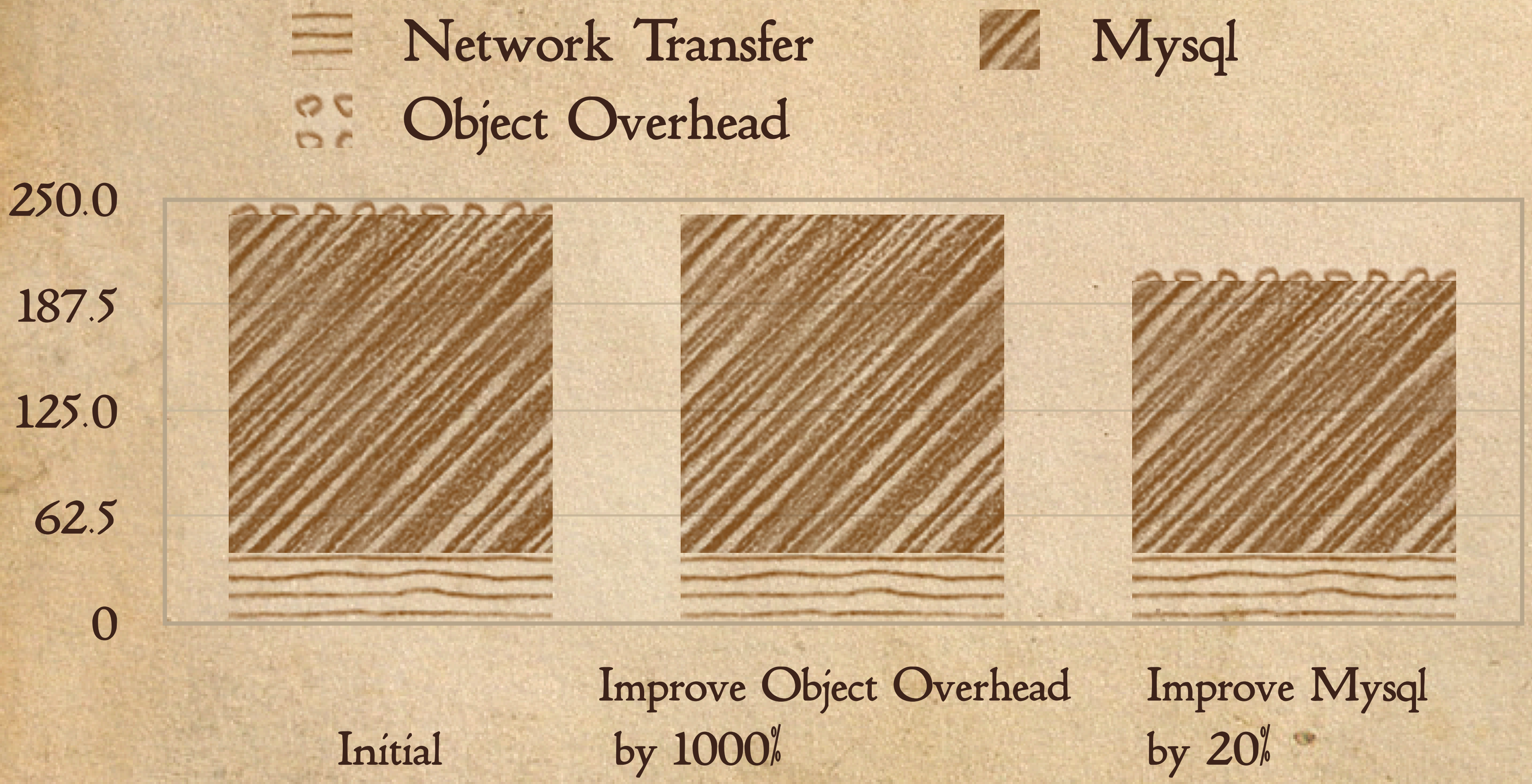
Knowing When to Stop

- Optimizations get exponentially more expensive as they are accrued.
- Striking a balance between performance and features.
- Unless you can 'go live', all the performance in the world is useless.

No Fast ≠ True

- Optimization takes work.
- There are some optimizations which are easy, but there is no 'silver bullet' to make your site faster.
- Be prepared to get your hands dirty.

Ahmdal's Law



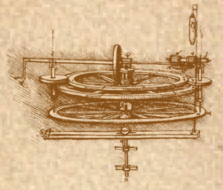
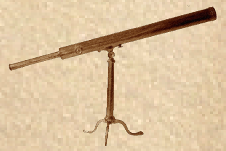
Network Transfer
Mysql
Object Overhead
250.0
187.5
125.0
62.5
0
Initial
Improve Object Overhead by 1000%
Improve Mysql by 20%

A Lesson From Open Source

Even if your projects aren't open source, you can try and take a lesson from the community:

Every part of the system can be open for change, so look for the greatest impact you can make, whether it's in PHP, Apache, your RDBMS, or wherever.

Contents

Best Practices

10 Best Practices

1. Use a compiler cache.
2. Control your include trees to no more than 10 includes. (Yes, that number is made-up.)
3. Be mindful of how you use your RDBMS.
4. Be mindful of all network resources.
5. Use regular expressions cautiously.
6. Always build with caching in mind.
7. Output buffering and compression are good.
8. Watch for resource exhaustion.
9. Profile early, profile often.
10. Dev-Ops cooperation is essential.

1. Compiler Caches

- On every request, PHP will compile and execute the indicated scripts.
- This compilation stage is expensive, and can rightfully be avoided.

2. Control Your Includes

- Including a file is expensive, even when using a compiler cache.
- Path resolution must be performed, resulting in at least one stat() and a realpath() call.
- A reasonable number of includes is fine (and promotes good code organization), but 25, 50, 100+ will impose a real performance penalty.
- I stick to 10 as my personal metric.

3. Mind Your

- The number one bottleneck I see in most systems is making poorly tuned (or poorly strategized) queries.
- No matter how was your PHP code, if it take a second to pull your data from your database, your scripts will take at least as long to execute.

4. Mind Your Network

- Don't forget that reaching network-available data is expensive. SOAP and other RPC mechanisms may be sexy, but they:
  - are slow
  - tie your quality of service to external services.

5. Use Regexs Cautiously

With great power comes great responsibility - Uncle Ben

- Regular expressions are an extremely powerful mini-language for matching and modifying text.
- As such, they can be very slow if you evaluate complex patterns.
- Nevertheless, they are tuned for the job they do. Use them but be mindful.

6. Proactively Support

- Caching (at the PHP level) is your most powerful optimization tool.
- While you want to avoid premature optimization, make sure that you write your applications so as to make integrating caching later on easy.
- Modifiable, refactorable code is good.

7. Small is Beautiful

Output buffering and compression allow you to optimize the amount of content you send to your clients.

This has two positive effects:

- Less bandwidth => lower costs
- Smaller pages => faster transfer times
- Smaller pages => optimized networking

8. Exhaustion

It is easy to exhaust resources, even without being CPU bound

- Having your Apache processes due something slow.
- Surpassing the concurrency setting on your RDBMS.
- Saturating your internal network.

9. Profile Often

Profiling is essential to understand how an application will perform under load, and to make sound design choices.

- Benchmarking is good.
- Use realistic data.
- If possible, profile on ‘live’ data as well.

10. Dev-Ops Cooperation

- Production problems are both time-critical and almost always hard to diagnose.
- Build team unity before emergencies, not during them.
- It is essential that operations staff provide good feedback and debugging information to developers regarding what code is working and what is not.
- Similarly, it is essential that development staff heed both pre-emptive and post-facto warnings from operations staff.
- Strict launch time-windows and oncall developer escalations help ease angsty operations teams.

Low-Hanging Fruit

Compile-Time Options

Compile Options

(modules)

- For fastest results either:
  - Compile mod_php statically
  - Compile as a DSO, but with

    --prefer-non-pic
- I prefer non-PIC DSOs, as they have equivalent speed metrics to the static libraries while still allowing flexible recompilation of PHP and extension addition.

Compile Options

(arch specific)

Distro vendors tend to ship code that will run on a wide variety of platforms, and thus must avoid architecture specific optimizations.

- -O3 (full optimizations)
- -arch/-mcpu
- -funroll_loops

Compile Options

(be a minimalist)

- --disable-all is a good place to start to decrease your instance memory footprint.
- Since I run non-PIC DSOs, I compile in only core extensions I know I need everywhere, and load others through php.ini.

INI

Optimizations

Minimize Variable Setting

- variables_order = 'GPC'
- register_argc_argv = Off
- register_globals = Off
- always_populate_raw_post_data = Off

Minimize Variable Manipulation

- magic_quotes_gpc = Off
- Filter data with a custom treat_data function.

Optimize File Searches

- Keep include_path to a minimum.
- Fully qualify all pathnames where possible
  include_once("$LIB/Auth.php");
  vs.
  include_once("Auth.php");
- open_basedir ≠ Off

Minimize Error Logging

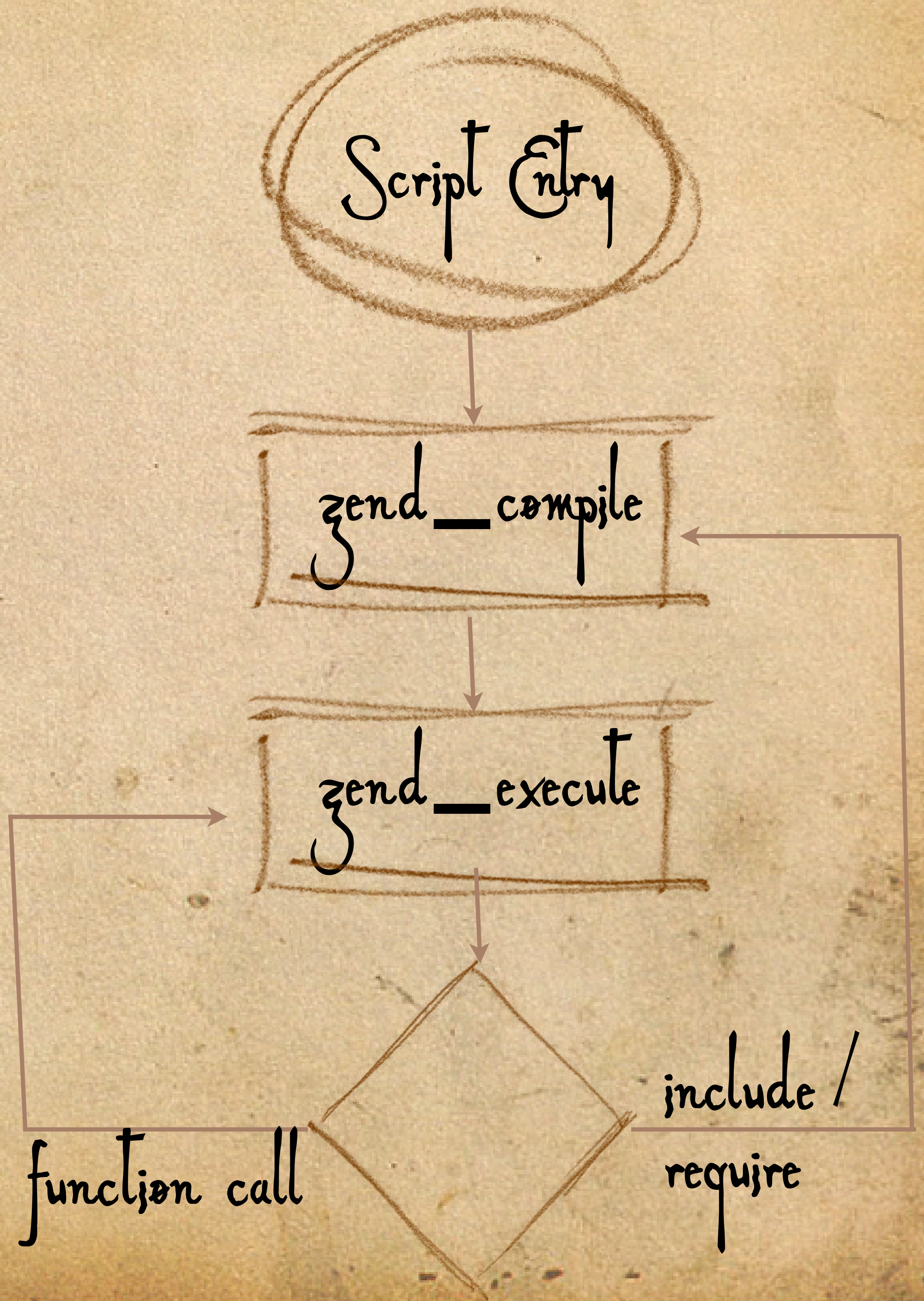
- Error logging, especially to syslog, is quite slow. Enable logging on your development server, and rectify all warnings there. Disable it in production.

Compiler Caches

1. Compiler Caches

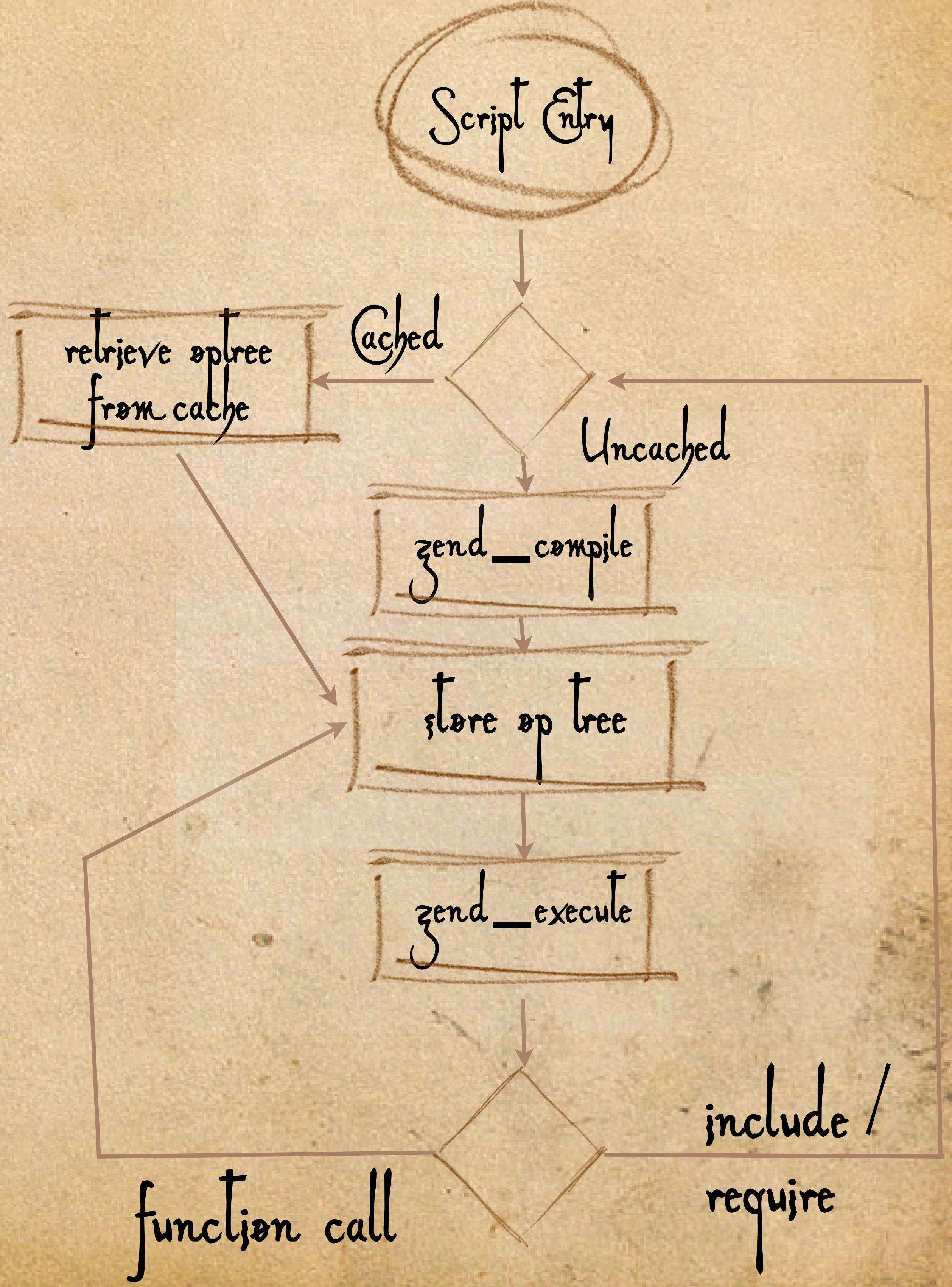
How PHP Works

- PHP is a 'runtime' language in that all scripts are compiled and executed anew on every request.
- Compilation is not cheap, but is amortizable.



1. Compiler Caches

- PHP's internal compilation calls are intercepted and checked for the cached compiled version, which is stored in shared memory.
- There are still some associated costs:
  - Reinstantitition
  - Path resolution
  - include guards



1. Compiler Caches

There is a smorgasborg of compiler caches:

- ZPS
- Turck MMCache
- IonCube
- APC

Apache Optimizations

Never do DNS Lookups inside Apache

- DNS resolution for logging should always be done as a post-process.
  - HostnameLookups Off
- Also, use IPs in mod_access acls.
- This applies to PHP scripts as well. If you need to reference network entities (RDBMSs, for instance) it is more efficient to use IPs.

Avoid Excessive Path Exploration

- Eliminate (potentially recursive) .htaccess searching with AllowOverride None
- Avoid expensive file permission checks with Options FollowSymLinks

Process Sizing

- Determining an optimal setting for MaxClients is difficult. Ideally you want to size it so that the server is almost fully CPU-utilized when that many clients are performing average accesses. This is highly application-dependent.
- As a rule of thumb I start with 25*#cpus, and work up from there, looking for the load to hover around #cpus while the system is heavily utilized.

Process Sizing

- To prevent a thundering-herd effect, set StartServers and MinSpareServers high. Ideally processes should always be created in advance.
- Set MaxRequestsPerChild to a large number.

Minimize Logging

- Disable discretionary logging where possible. Verbose logs are nice for debugging, but resign that to your development server and true emergencies.

Disable Keepalives

- HTTP/1.1 keepalives are designed to enhance performance by avoiding the setup cost on TCP connections for subsequent requests. Unfortunately, if you have more active clients than MaxClients, it is also a fabulous way to DoS yourself.

The Keepalive Problem

Let's optimize based on average page service time for a user. Assume:

- N objects on a page.
- t1 seconds for TCP connection.
- t2 seconds per page.
- K seconds keepalive timeout.

Non-keepalive: $N*(t1 + t2)$

Keepalive: $N*(t2) + t1 + K$

So, for keepalive connections to be a win: $\underline{K < t1*(N - 1)}$

lingerd

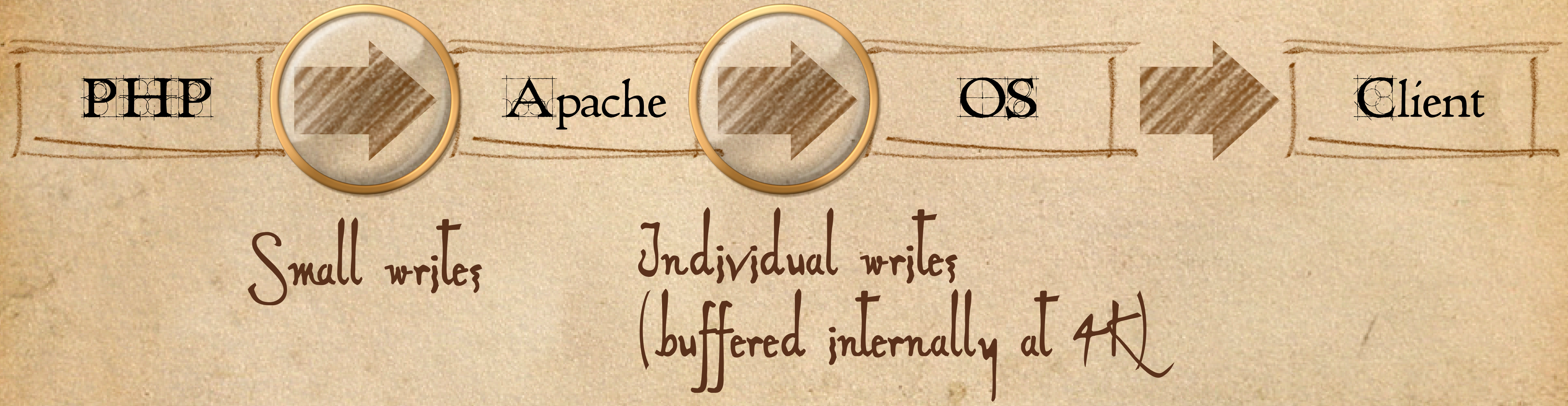
- Due to some implementations in TCP/IP, when an Apache request has been served, its socket can not be immediately closed. Instead Apache must linger on the socket to ensure its send buffer is successfully sent.
- lingerd allows Apache to hand off the lingering socket to a simple daemon whose sole purpose is handling closes (and thus can do so very efficiently).

Aligning Output Buffers

Matching Your IO Sizes

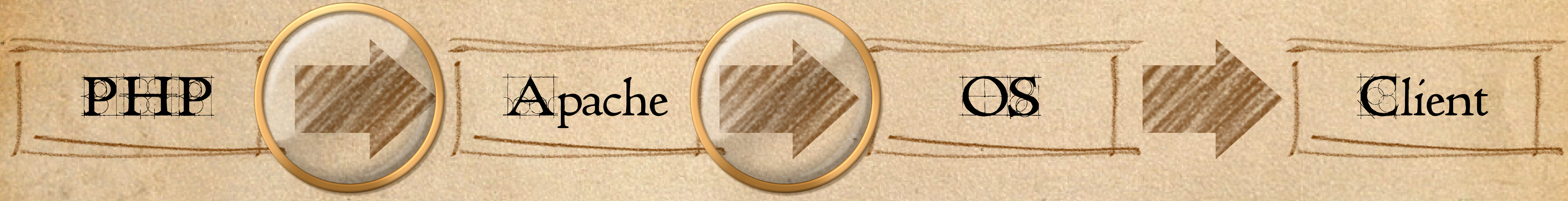
- The goal is to pass off as much work to the kernel as efficiently as possible.
- Optimizes PHP<->OS Communication
- Reduces Number Of System Calls

The Path Of Data in PHP



Unbuffered Writes

The Path Of Data in PHP

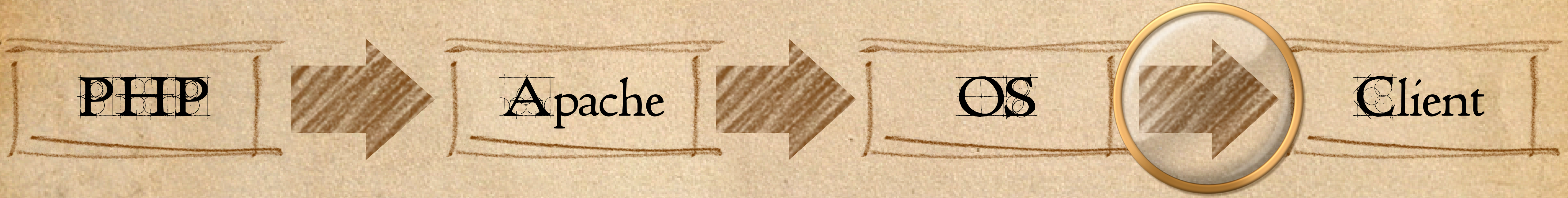


Large writes

Triggers use of writev() (more efficient)

Buffered Writes

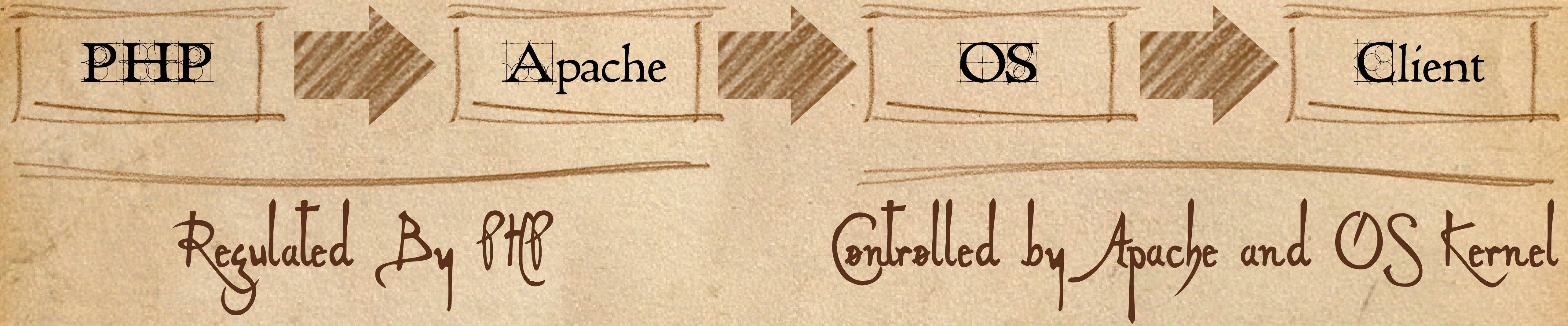
The Path Of Data in PHP



Regulated by OS tcp Buffer Size

OS › Client Communciation

The Path Of Data in PHP



The Final Picture

Ouput Buffering

- Efficient
- Flexible
- In your script with ob_start()
- Everywhere with output_buffering = On (php.ini)

Compressing Content

Content Compression

- Most modern browsers support the ability to receive content compressed with gzip or compress and to decompress it for display.
- Browsers advertise this support with the Accept-Encoding header.
- Compressing content costs in CPU (up to 10% more CPU intensive), but can shrink text-type contents by up to 90%. This allows for more aggressive buffer sizing and fewer packets on the wire (i.e. faster downloads!)

Content Compression (the PHP way)

In php.ini

- zlib.output_compression = On

or

- output_handler = ob_gzhandler

Handling compression inside PHP is convenient and efficient, but lacks the flexibility of an external solution.

Content Compression (mod_gzip)

mod_gzip is an Apache module that allows for highly configurable content compressions.
In addition to negotiated sessions, you can modify it's behavior based on file names, browser settings and MIME types.

Content Compression (other resources)

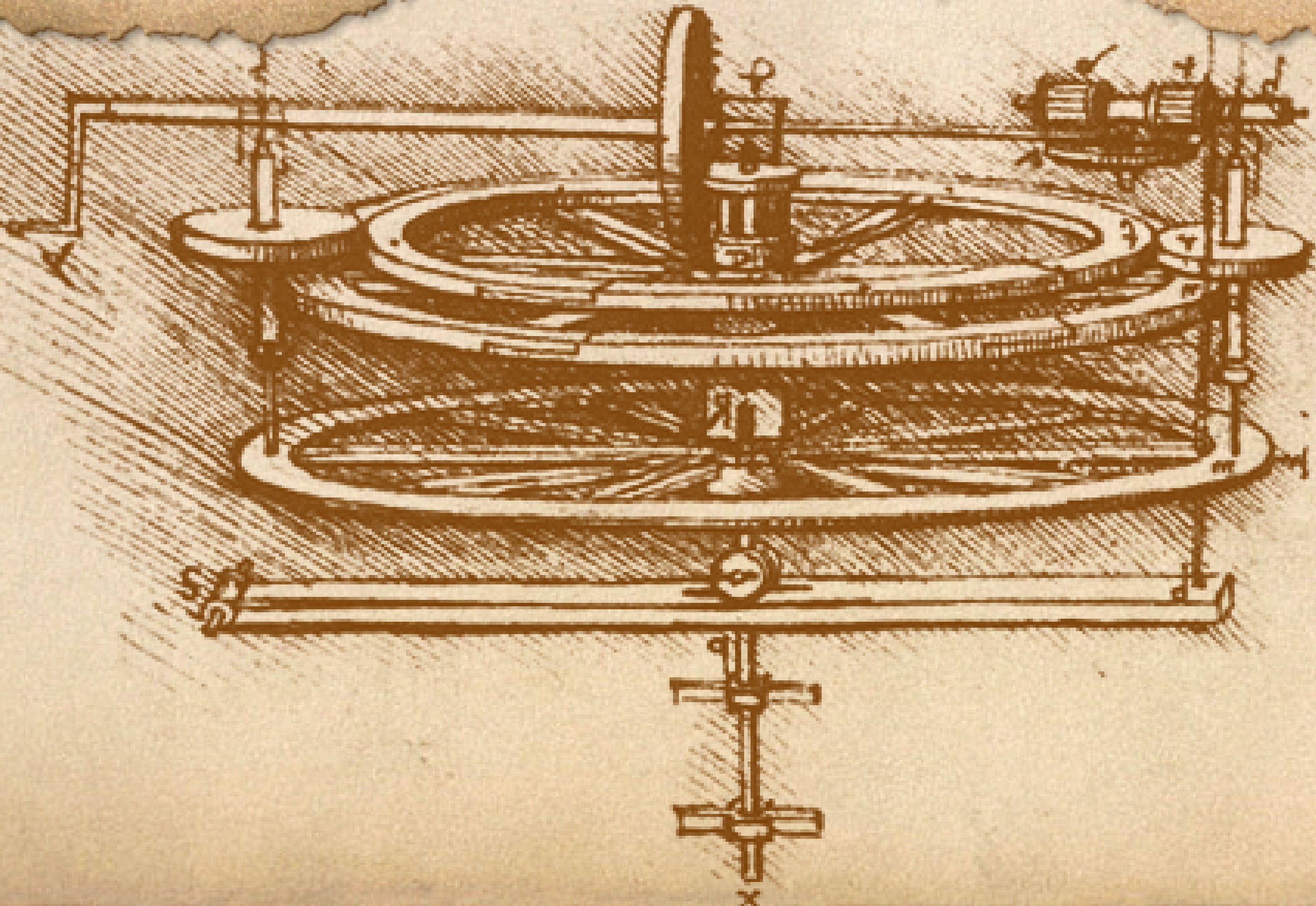
- In Apache 1.3:
  - mod_deflate
- In Apache 2.0
  - mod_gz
  - mod_deflate

Optimizing Content

Optimizing HTML

- Optimizing content the 'old-fasioned way' is benefitial, even in conjunction with content compression.
  - Use CSS (often reduces page sizes by 30 %).
  - Remove comments and whitespace.
  - Use Javascript to generate repetitive HTML.
  - Use shortened URLs.
  - Cut corners on well-formedness.

General Techniques



Architectural Concerns
Static vs. Dynamic Content

Static vs. Dynamic

- mod_php is not a lightweight process. Each Apache child uses:
  - A fair chunk of memory.
  - Persistent resources like DB connections.
- mod_php is optimized for serving dynamic content. Serving static content with it results in the expensive portions of the process being squandered

Static vs. Dynamic

- Ideally, all static content should be served off of a server optimized for that task.
    - thttpd
    - tux
    - X15
    - ZPS

Static vs. Dynamic

Even if you don't have the resources now, you can prepare yourself fro serving static content separately as follows:

```
$STATIC = "http://www.example.com";
<img src="<? $STATIC ?>/sample.gif" />
```

This simple technique will save you massive amounts of heartache if you ever decide to serve static content independently. Just change the value of **$STATIC** in one place and you're done.

Architectural Concerns Shared Nothing

What is Shared Nothing?

- Shared Nothing is a buzz-word.
- Shared Nothing isn't an architecture.
- Shared Nothing is the philosophy that a web application should not maintain it's own statefulness.
- Shared Nothing says that statefulness and inter-request communication should be done through the data storage layer.

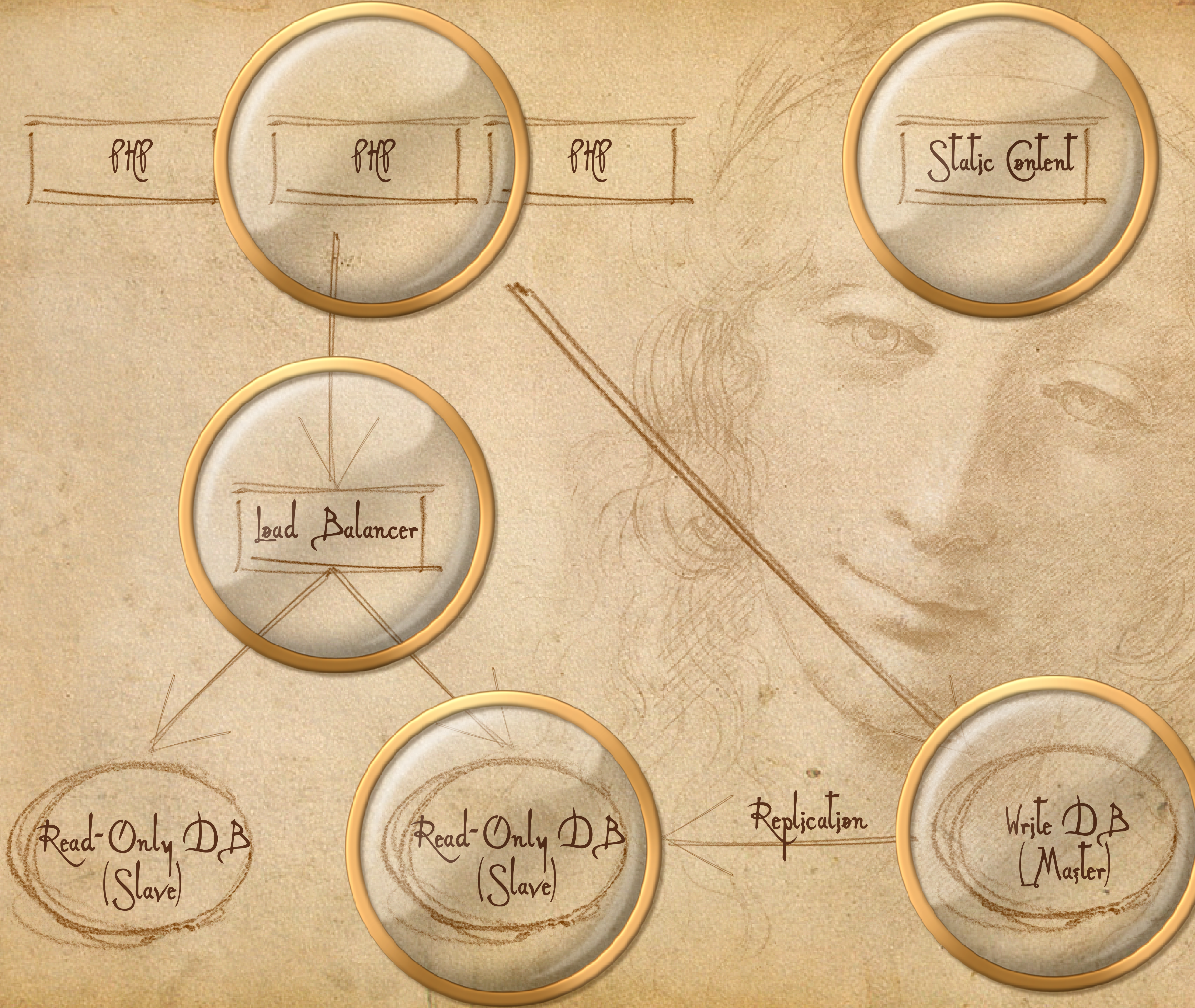
Shared Nothing is a Lie

(kinda)

Shared nothing is a bit tricksy. It says that you shouldn't maintain statefulness in PHP apps, but instead do it through things like the file system or a database. This may seem like an evasion of responsibility (it is), but it is also a sound idea. The point is that RDBMS vendors spend huge amounts of time and effort solving the general problem of making data visible to clients in a consistent fashion. Its hubris (and a waste of time) to try and tackle the problem more efficiently.

What Does Shared Nothing Buy You?

- Effectively infinte horizontal scalability (assuming your data store can scale with you).
- Fully transparent failover capability.
- Less hardcore business logic in your code.


PHP
PHP
PHP
Static Content
Load Balancer
Read-Only DB (Slave)
Read-Only DB (Slave)
Replication
Write DB (Master)

$_SESSION

Cautious Sessions

PHPs session extension does not violate Shared Nothing, but it certainly leads you down the path of temptation.

Standard session handlers use fast local storage (files, shared memory) to handle session data.

To move from one machine to many and still have sessions work, you need to move to a centralized storage system which may not be as fast.

Playing Safe With Sessions

- Never use session.auto_start.
- Never set session.use_trans_sid.
- Only use sessions when necessary.

Alternative: Use cookies as your session data store!

Use Internal Functions

Internal Functions

- Internal functions are implemented in C, and thus always faster than functions written in PHP to do the same job.
- This is true of any VM: executing on the underlying hardware machine will always be faster.

Internal Functions

To demonstrate the difference, let's compare a hand-coded version of bin2hex() to the real thing.

```
function mybin2hex ($temp) {
  $len = strlen($temp);
  $data = '';
  for ($i=0; $i<$len; $i++) {
    $data.=sprintf("%02x",ord(substr($temp,$i,1)));
  }
  return $data;
}
```

I would claim this was contrived if I hadn't pulled the function from a recent posting to the PHP user manual notes.

Benchmark_Iterate

Benchmark_Iterate is a nice PEAR class for comparing the performance of function implementations.

```
require_once("Benchmark/Iterate.php");
foreach(array('mybin2hex', 'bin2hex') as $func) {
        $b = new Benchmark_Iterate;
        $b->run('1000', $func, $test_str);
        $result = $b->get();
        print "$func\t";
        printf("Clock Time: %1.6f\n",$result['mean']);
}
```

The Results!

Testing this on a random 512 byte string, the following results are quite telling:

```
$test_str = '';
for($i=0; $i < 512; $i++) {
  $test_str .= chr(rand(0, 128));
}

...

mybin2hex        Clock Time: 0.006157
bin2hex          Clock Time: 0.000069
```

Regexes: Not Your Enemy

Regular Expressions

Regular expressions are unfairly maligned. PCREs are a mini-language to themselves. They breed the same bad code as any other language.

```
“(\w+|\s{1,2})*“
```

Matches words and spaces inside a quoted string.

```
“(\w+|\s{1,2})*+“
```

The same, but with backtracking disabled. Much faster on partially successful matches.

Databases

Minimize Round Trips

- Avoid database (and any external resource) lookups whenever possible.
- If you store configuration data in your DB, use a caching scheme to manage it.

Fetch Only What You Need

- Lazy initialization is your friend - if you aren't sure you're going to need it, don't pull it.
- Beware of platform-specific nuances:
  - In MySQL fetching a column value forces a read of the entire row, so aggressively fetching contents there makes sense.
  - In Oracle, you can return a indexed column in an IOT without ever looking in the table proper, and CLOBs are stored out-of-line, so it is cheaper to be selective in what you fetch.

Use Prepared Statements

- On systems that support them at the database/driver level, prepared statements can give a significant performance boost.
- On all systems, prepared statements can help protect you against SQL injection attacks by managing the escaping of your inputs.

EXPLAIN

EXPLAIN is the SQL keyword for instructing the RDBMS to show you how it plans to execute a query

```
mysql> explain SELECT itemid FROM member_queue WHERE member_id = "4001" ORDER BY rank;+--------------+------
+---------------+------+---------+------+--------+-----------------------------+
| table        | type | possible_keys | key  | key_len | ref  | rows   | Extra                       |
+--------------+------+---------------+------+---------+------+--------+-----------------------------+
| member_queue | ALL  | NULL          | NULL |    NULL | NULL | 110123 | Using where; Using filesort |
+--------------+------+---------------+------+---------+------+--------+-----------------------------+
1 row in set (0.00 sec)
```

With an Index

```
mysql> create index mem_id on member_queue(member_id);
Query OK, 110123 rows affected (4.32 sec)
Records: 110123  Duplicates: 0  Warnings: 0

mysql> explain select itemid from member_queue where member_id = "4001" ORDER BY rank;
+--------------+------+---------------+--------+---------+-------+------+-----------------------------+
| table        | type | possible_keys | key    | key_len | ref   | rows | Extra                       |
+--------------+------+---------------+--------+---------+-------+------+-----------------------------+
| member_queue | ref  | mem_id        | mem_id |       5 | const |    1 | Using where; Using filesort |
+--------------+------+---------------+--------+---------+-------+------+-----------------------------+
1 row in set (0.00 sec)
```

Non-Indexed Joins can be Disastorous

Here we have a join on two tables where the pivot is not on either table, This results in n*m (or 206160500000) rows being scanned.

```
mysql> explain select orderdetail.decription from orders, orderdetail where orders.userid = 1001 and orders.orderid =
orderdetail.orderid and available = 'y';
+-------------+------+---------------+------+---------+------+--------+-------------+
| table       | type | possible_keys | key  | key_len | ref  | rows   | Extra       |
+-------------+------+---------------+------+---------+------+--------+-------------+
| orders      | ALL  | NULL          | NULL |    NULL | NULL | 500000 | Using where |
| orderdetail | ALL  | NULL          | NULL |    NULL | NULL | 412321 | Using where |
+-------------+------+---------------+------+---------+------+--------+-------------+
```

How To Find Bad Queries

- MySQL
  - Enable slow query logging
  - --log-long-format will report all non-indexed queries.
- Oracle
  - Query against v$session_wait for current queries
  - Query against v$sqlarea for cpu and i/o intensive queries.

Indexing Gotchas

- The LIKE operator will only hit an index on its leading static part.
- Only a single index will be used for a given table during query execution. If you need index hits on more than one column, you need a multi-column index.
- Sorting based on a function is slow (usually requires a result set scan unless your database supports function-based indexes)
- Outer joins are much more costly than inner joins.
- Indexes make lookups faster, but writes slower.

External Network Resources

Referencing External Data

- SOAP
- XML-RPC
- Trackback/Pingback
- Content validation
- Co-Registration

Asynchronous is Good

The goal with handling any external resource should be to make it asynchronous. This decouples your display functionality and web cluster resources from the third-party data source.

If you can't decouple the data fetch from your application, you're in bad shape. This can easily de-stabilize your application as all your resources become allocated to handling these slow-feeders.

If the data is for display only, one last hope is to remove all semblance of proxying from your application and have it included for display via Javascript or some other client-side language.

Caching

Caching Categories

- Methodologies
  - Cache-on-Write
  - Cache-on-Demand
- Scope
  - Full Page
  - Partial Page
  - Algorithmic

Cache on Demand PHP Passthru

Here we use PEAR's Cache_Lite to cache the entire page:

```
$cache = new Cache_Lite_Output($options);
if(!$cache->start(__FILE__)) {
  // perform page logic here
}
```

We should also incorporate important $_GET variables here.

Cache on Demand Writing out Static Files

This is a classic PHP 'trick'. It's usually done with an Apache ErrorDocument handler, but that does funny things to your logs, requires you to manually set many headers, and is generally inflexible.

mod_rewrite was made for this sort of task:

```
RewriteEngine On
RewriteCond /path/to/docroot/%{REQUEST_FILENAME} !-f
RewriteRule ^/(.*).html /generate.php?page=$1
```

Caching with APC

APC also provides functions for storing and fetching user content from its shared memory cache:

```
if($data = apc_fetch($key)) {
  // generate $data
} else {
  apc_store($key, $data);
}
```

also primitives for storing constants:

```
if(!apc_load_constants("CONST::".__FILE)) {
  $constants = array('const1' => $value);
  apc_define_constants("CONST::".__FILE__, $constants);
}
```

Caching with APC

You could alos use these to implement a simple content cache.

```
if($page = apc_fetch("PAGE::".__FILE__)) {
  echo $page;
  exit;
}
ob_start();
// do normal work
$page = ob_get_flush();
apc_store("PAGE::".__FILE__, $page);
```

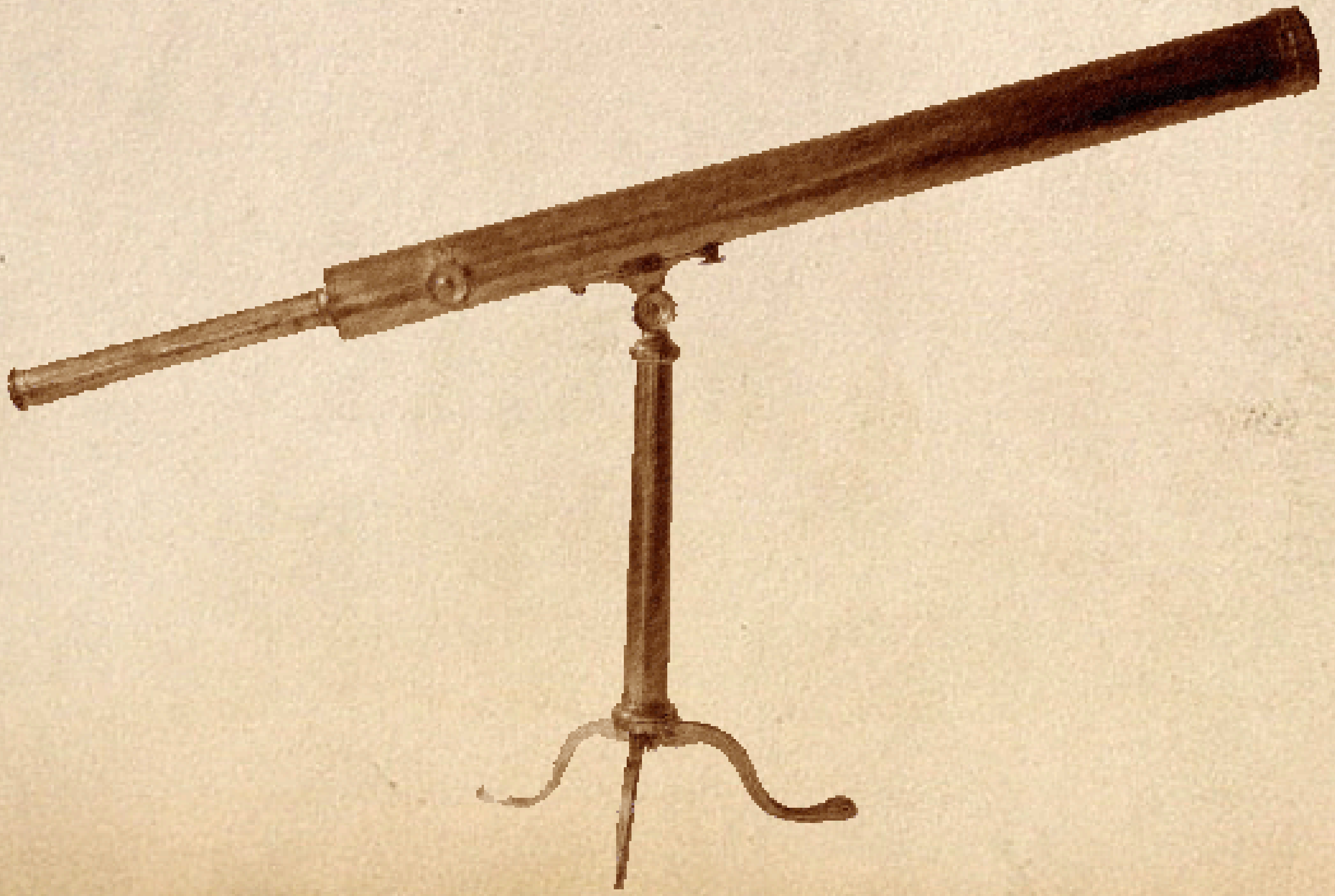
Distributed Caching With memcached

In some applications, cluster-wide cache coherency is critical. In these situations, local caches are difficult to use because they cannot be centrally expunged.

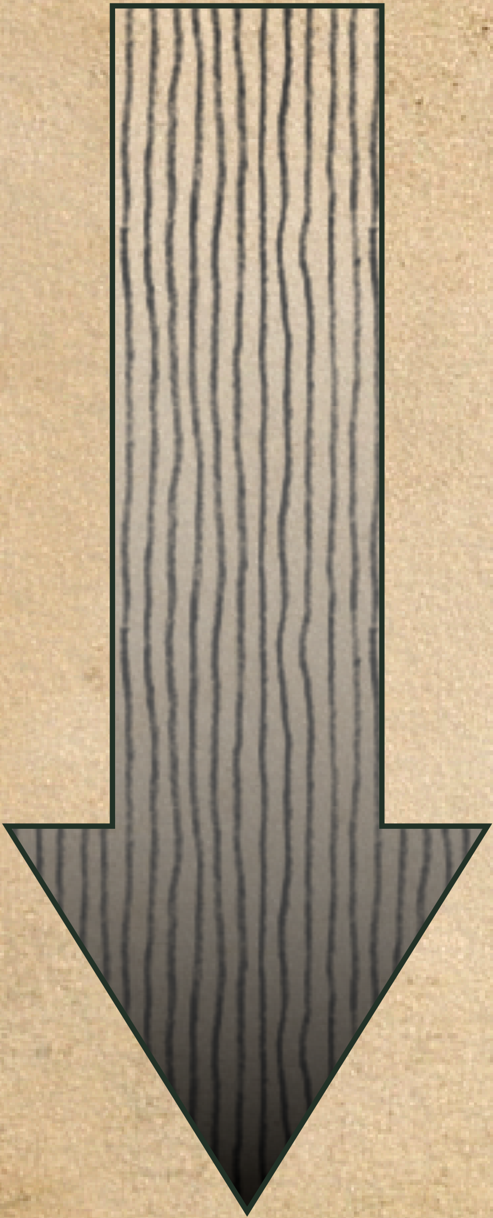
memcached is a network caching server that stores basic key/value pairs. It's quite fast, and very popular, especially for page fragment caching.

```
$memcache = memcache_connect('localhost', 11211);
if($memcache && ($fragment = $memcache->get($key))) {
  // do something with fragment
} else {
  // generate fragment
  $memcache && $memcache->set($key, $fragment);
}
```

Profiling



Stages of Profiling



Systems Investigation

Script Identification (logs / strace)

Script Profiling (APD / XDebug / DTrace / Strace)

Why Profiling Helps

- Profiling targets your efforts by finding the expensive portions of your code.
- Even if your code was tuned when you wrote it, changing data disposition can render old tuning decisions obsolete.
- Profiling helps you understand how your application works in practice.

Essential qualities.

- Transparency.
- Low overhead.
- Global overview statistics.
- In-depth local statistics.

PHP Profiling tools

- APD
- mod_log_config
- XDebug
- Zend IDE
- strace
- Benchmark_Profiler

PECL Install

- Almost as simple as:
  pear install apd
- It's a Zend extension, so you need to add in your ini file:
  zend_extension=/path/to/apd.so

First Example

An RSS Reader

```
require_once 'Onyx/RSS.php';
function rss_entries($url)
{
  $feed = array();
  $parser =  &new Onyx_RSS;
  $parser->parse($url);
  $meta = $parser->getData(ONYX_META);
  $feed['title'] = $meta['title'];
  while($item = $parser->getNextItem()) {
    $entry = array();
    if(isset($item['pubdate'])) $date = $item['pubdate'];
    else if (isset($item['dc:date'])) $date = $item['dc:date'];
    else $date = "now";
    $entry['ts'] = strtotime($date);
    $entry['date'] = gmdate('Y-m-j H:i:00+0000', $entry['ts']);
    if(isset($item['description']))
      $entry['description'] = $item['description'];
    else if(isset($item['content:encoded']))
      $entry['description'] = $item['content:encoded'];
    $entry['title'] = (string) $item['title'];
    $entry['link'] = (string) $item['link'];
    $feed['items'][] = $entry;
  }
  return $feed;
}
```

Analyzing Its Performance

- To profile it, add the apd_set_pprof_trace() call.
- This will profile the script from that point forward, and dump a trace file in your dumpdir.

```
<?php

apd_set_pprof_trace();
require_once 'Onyx/RSS.php';
rss_entries("gs.rss");
?>
```

```
> ls /tmp/traces/
pprof.25401.0
```

Parsing The Tracefile

```
> pprofp -R /tmp/traces/pprof.24917.0

Trace for /Users/george/phpworks/02.php
Total Elapsed Time = 0.45
Total System Time  = 0.02
Total User Time    = 0.26


         Real         User        System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call    s/call  Name
--------------------------------------------------------------------------------------
100.0 0.00 0.45    0.00 0.26    0.00 0.02      1    0.0001   0.4480   main
 97.3 0.00 0.44    0.00 0.24    0.00 0.02      1     0.0000   0.4361   rss_entries
 93.7 0.00 0.42    0.00 0.24    0.00 0.02      1     0.0000   0.4199   ONYX_RSS->parse
 93.1 0.00 0.42    0.00 0.24    0.00 0.02      7     0.0000   0.0596   xml_parse
 65.6 0.05 0.29    0.01 0.20    0.01 0.02   1051  0.0001   0.0003   ONYX_RSS->cdata
 30.2 0.14 0.14    0.09 0.09    0.00 0.00   1051  0.0001   0.0001   trim
 25.6 0.00 0.11    0.01 0.03    0.00 0.00    164   0.0000   0.0007   ONYX_RSS->tag_open
```

Generating a Calltree

```
> pprofp -cmT /tmp/traces/pprof.24917.0
...
0.02         ONYX_RSS->cdata    C: ./Onyx/RSS.php:133
0.02           trim             C: ./Onyx/RSS.php:203
0.02           strlen           C: ./Onyx/RSS.php:203
0.02         ONYX_RSS->tag_open C: ./Onyx/RSS.php:133
0.02           strtolower       C: ./Onyx/RSS.php:176
0.02           sizeof           C: ./Onyx/RSS.php:191
0.02         ONYX_RSS->cdata    C: ./Onyx/RSS.php:133
0.02           trim             C: ./Onyx/RSS.php:203
...
```

Looking into the source

- Here is the location of the call in question. Luckily, trim() is used improperly here. What this code wants to do is test if $cdata contains non-whitespace. A regex is appropriate for this.

```
function cdata($parser, $cdata)
{
  if(strlen(trim($cdata)) && $cdata != "\n")
    switch ($this->type)
    {
```

Correct and Re-Profile

```
function cdata($parser, $cdata)
{
  if(preg_match('/\S/',$cdata))
    switch ($this->type)
    {
```

Replacing the strlen(trim()) calls with a non-whitespace regex match, yields almost a 30% speed-up.

```
Trace for /Users/george/phpworks/02.php
Total Elapsed Time = 0.23
Total System Time  = 0.02
Total User Time    = 0.18
          Real         User        System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls     call     s/call  Name
--------------------------------------------------------------------------------------
100.0 0.00 0.23     0.00 0.18    0.00 0.02       1     0.0001   0.2299 main
 92.1 0.00 0.21     0.00 0.17    0.00 0.02       1     0.0000   0.2118 rss_entries
 89.3 0.00 0.21     0.00 0.17    0.00 0.01       1     0.0000   0.2054 ONYX_RSS->parse
 88.0 0.00 0.20     0.00 0.17    0.00 0.01       7     0.0000   0.0289 xml_parse
 73.8 0.04 0.17     0.05 0.14    0.00 0.01    1051     0.0000   0.0002 ONYX_RSS->cdata
 34.2 0.08 0.08     0.07 0.07    0.01 0.01    1051     0.0001   0.0001 preg_match
```

Lessons to Learn

- The right tool is usually the one designed for the job. Regexes get a bad rap, but if you need their functionality, they are almost always faster than cobbling that functionality together by hand.
- Always measure your changes. A poor optimization can reduce your performance. Without testing, you may never know.

Configuration Options

pprofp summary flags

- -R Sort by real time, and include all child calls. This is useful for finding top-level routines that take a long time.
- -r Sort by real time, excluding child calls. This is useful for identifying base-level functions which are expensive.
- -Z Sort by (user+system) time, including child calls. This finds ***computationally*** expensive code blocks, and can filter out noise from slow network readers or process contention.
- -z Like -Z but excluding child calls.
- -u,-U,-s,-S Sort on user or system time respectively.
- -l Sort by number of calls

pprofp calltree flags

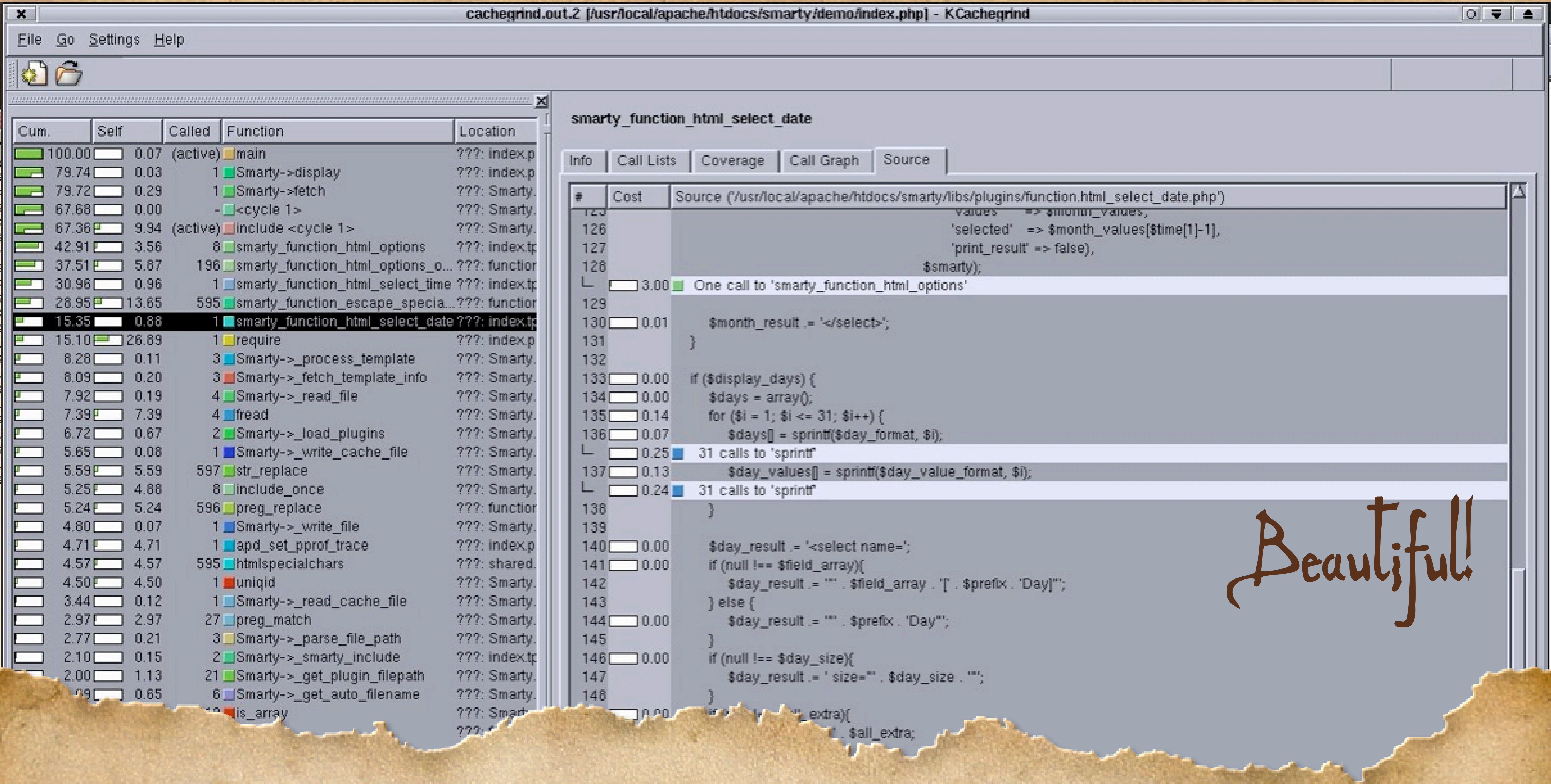
- -T Display an un-compressed calltree.
- -t Display a calltree, compressing repeated calls to the same function.
- -c Display the execution times alongside the calltree listing.
- -m Display call location (__FILE__:__LINE__) in the calltree.

pprofp INI Options

- apd.dumpdir

  The location where trace files will be dumped.
- apd.statement_tracing

  Enable tracing on a per-statement level, instead of per-function. The default is Off. This is currently only used in the kcachegrind viewer.

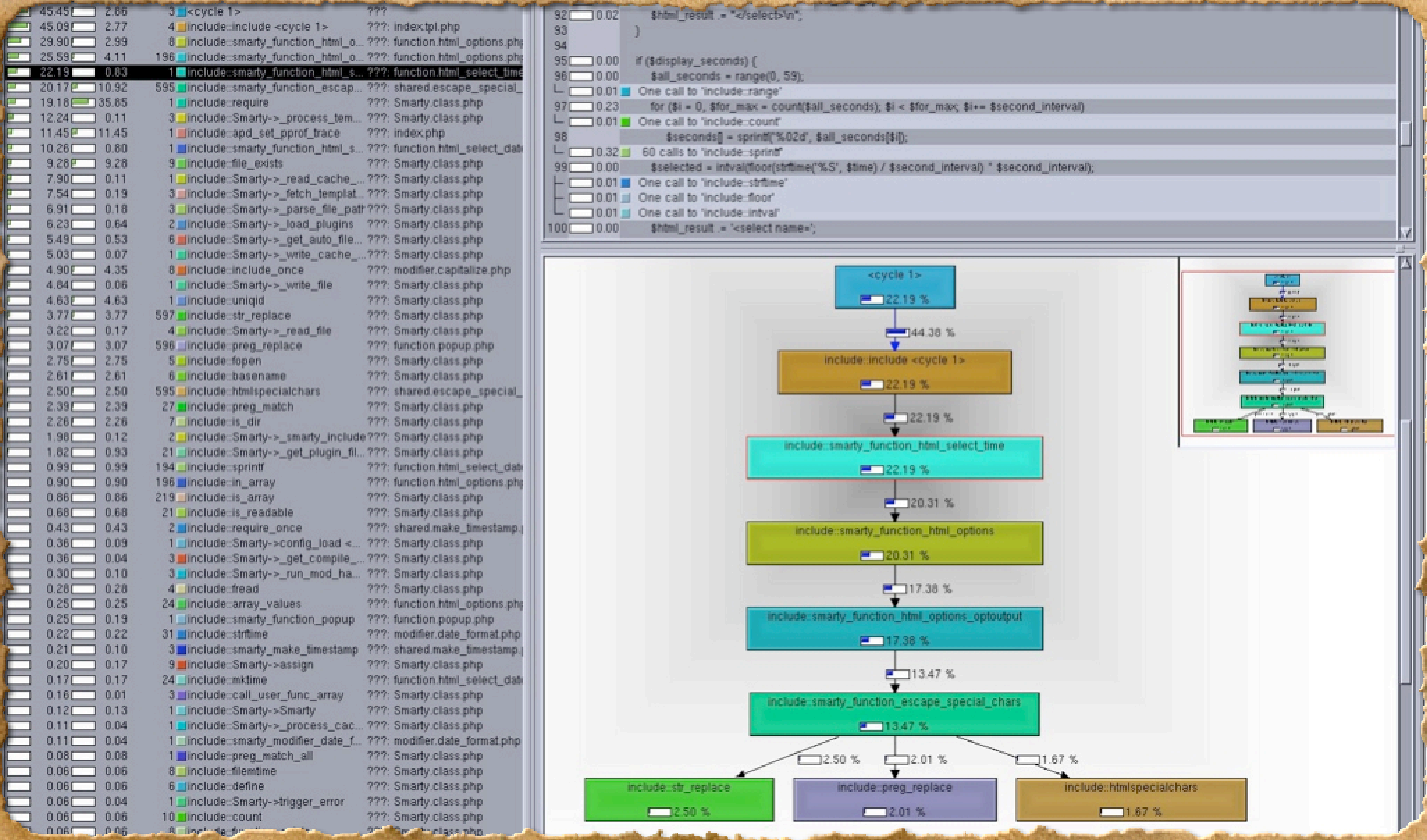

cachegrind.out.2 [/usr/local/apache/htdocs/smarty/demo/index.php] - KCachegrind
File Go Settings Help
smarty_function_html_select_date
Info
Call Lists
Coverage
Call Graph
Source

```
Cum.    Self   Called   Function                          Location
100.00   0.07  (active) main                              ???: index.p
 79.74   0.03       1   Smarty->display                   ???: index.p
 79.72   0.29       1   Smarty->fetch                     ???: Smarty.
 67.68   0.00       -   <cycle 1>                         ???: Smarty.
 67.36   9.94  (active) include <cycle 1>                 ???: Smarty.
 42.91   3.56       8   smarty_function_html_options      ???: index.tp
 37.51   5.87     196   smarty_function_html_options_o... ???: functior
 30.96   0.96       1   smarty_function_html_select_time  ???: index.tp
 28.95  13.65     595   smarty_function_escape_specia...  ???: functior
 15.35   0.88       1   smarty_function_html_select_date  ???: index.tp
 15.10  26.89       1   require                           ???: index.p
  8.28   0.11       3   Smarty->_process_template         ???: Smarty.
  8.09   0.20       3   Smarty->_fetch_template_info      ???: Smarty.
  7.92   0.19       4   Smarty->_read_file                ???: Smarty.
  7.39   7.39       4   fread                             ???: Smarty.
  6.72   0.67       2   Smarty->_load_plugins             ???: Smarty.
  5.65   0.08       1   Smarty->_write_cache_file         ???: Smarty.
  5.59   5.59     597   str_replace                       ???: Smarty.
  5.25   4.88       8   include_once                      ???: Smarty.
  5.24   5.24     596   preg_replace                      ???: functior
  4.80   0.07       1   Smarty->_write_file               ???: Smarty.
  4.71   4.71       1   apd_set_pprof_trace               ???: index.p
  4.57   4.57     595   htmlspecialchars                  ???: shared.
  4.50   4.50       1   uniqid                            ???: Smarty.
  3.44   0.12       1   Smarty->_read_cache_file          ???: Smarty.
  2.97   2.97      27   preg_match                        ???: Smarty.
  2.77   0.21       3   Smarty->_parse_file_path          ???: Smarty.
  2.10   0.15       2   Smarty->_smarty_include           ???: index.tp
  2.00   1.13      21   Smarty->_get_plugin_filepath      ???: Smarty.
    09   0.65       6   Smarty->_get_auto_filename        ???: Smarty.
                        is_array                          ???: Smar
                                                          ???:
```

```
#    Cost   Source ('/usr/local/apache/htdocs/smarty/libs/plugins/function.html_select_date.php')
126                                       'selected'    => $month_values[$time[1]-1],
127                                       'print_result' => false),
128                                    $smarty);
└     3.00   One call to 'smarty_function_html_options'
129
130   0.01       $month_result .= '</select>';
131        }
132
133   0.00     if ($display_days) {
134   0.00       $days = array();
135   0.14       for ($i = 1; $i <= 31; $i++) {
136   0.07           $days[] = sprintf($day_format, $i);
└     0.25   31 calls to 'sprintf'
137   0.13           $day_values[] = sprintf($day_value_format, $i);
└     0.24   31 calls to 'sprintf'
138          }
139
140   0.00       $day_result .= '<select name=';
141   0.00       if (null !== $field_array){
142                  $day_result .= '"' . $field_array . '[' . $prefix . 'Day]"';
143              } else {
144   0.00           $day_result .= '"' . $prefix . 'Day"';
145              }
146   0.00       if (null !== $day_size){
147                  $day_result .= ' size="' . $day_size . '"';
148              }
```

Beautiful!

KCachegrind



```
45.45  2.86    3 <cycle 1>                             ???
45.09  2.77    4 include::include <cycle 1>            ???: index.tpl.php
29.90  2.99    8 include::smarty_function_html_o...    ???: function.html_options.php
25.59  4.11  196 include::smarty_function_html_o...    ???: function.html_options.php
22.19  0.83    1 include::smarty_function_html_s...    ???: function.html_select_time
20.17 10.92  595 include::smarty_function_escap...     ???: shared.escape_special_
19.18 35.85    1 include::require                      ???: Smarty.class.php
12.24  0.11    3 include::Smarty->_process_tem...      ???: Smarty.class.php
11.45 11.45    1 include::apd_set_pprof_trace          ???: index.php
10.26  0.80    1 include::smarty_function_html_s...    ???: function.html_select_dat
 9.28  9.28    9 include::file_exists                  ???: Smarty.class.php
 7.90  0.11    1 include::Smarty->_read_cache_...      ???: Smarty.class.php
 7.54  0.19    3 include::Smarty->_fetch_templat...    ???: Smarty.class.php
 6.91  0.18    3 include::Smarty->_parse_file_path     ???: Smarty.class.php
 6.23  0.64    2 include::Smarty->_load_plugins        ???: Smarty.class.php
 5.49  0.53    6 include::Smarty->_get_auto_file...    ???: Smarty.class.php
 5.03  0.07    1 include::Smarty->_write_cache_...     ???: Smarty.class.php
 4.90  4.35    8 include::include_once                 ???: modifier.capitalize.php
 4.84  0.06    1 include::Smarty->_write_file          ???: Smarty.class.php
 4.63  4.63    1 include::uniqid                       ???: Smarty.class.php
 3.77  3.77  597 include::str_replace                  ???: Smarty.class.php
 3.22  0.17    4 include::Smarty->_read_file           ???: Smarty.class.php
 3.07  3.07  596 include::preg_replace                 ???: function.popup.php
 2.75  2.75    5 include::fopen                        ???: Smarty.class.php
 2.61  2.61    6 include::basename                     ???: Smarty.class.php
 2.50  2.50  595 include::htmlspecialchars             ???: shared.escape_special_
 2.39  2.39   27 include::preg_match                   ???: Smarty.class.php
 2.26  2.26    7 include::is_dir                       ???: Smarty.class.php
 1.98  0.12    2 include::Smarty->_smarty_include      ???: Smarty.class.php
 1.82  0.93   21 include::Smarty->_get_plugin_fil...   ???: Smarty.class.php
 0.99  0.99  194 include::sprintf                      ???: function.html_select_dat
 0.90  0.90  196 include::in_array                     ???: function.html_options.php
 0.86  0.86  219 include::is_array                     ???: Smarty.class.php
 0.68  0.68   21 include::is_readable                  ???: Smarty.class.php
 0.43  0.43    2 include::require_once                 ???: shared.make_timestamp.
 0.36  0.09    1 include::Smarty->config_load <...     ???: Smarty.class.php
 0.36  0.04    3 include::Smarty->_get_compile_...     ???: Smarty.class.php
 0.30  0.10    3 include::Smarty->_run_mod_ha...       ???: Smarty.class.php
 0.28  0.28    4 include::fread                        ???: Smarty.class.php
 0.25  0.25   24 include::array_values                 ???: function.html_options.php
 0.25  0.19    1 include::smarty_function_popup        ???: function.popup.php
 0.22  0.22   31 include::strftime                     ???: modifier.date_format.php
 0.21  0.10    3 include::smarty_make_timestamp        ???: shared.make_timestamp.
 0.20  0.17    9 include::Smarty->assign               ???: Smarty.class.php
 0.17  0.17   24 include::mktime                       ???: function.html_select_dat
 0.16  0.01    3 include::call_user_func_array         ???: Smarty.class.php
 0.12  0.13    1 include::Smarty->Smarty               ???: Smarty.class.php
 0.11  0.04    1 include::Smarty->_process_cac...      ???: Smarty.class.php
 0.11  0.04    1 include::smarty_modifier_date_f...    ???: modifier.date_format.php
 0.08  0.08    1 include::preg_match_all               ???: Smarty.class.php
 0.06  0.06    8 include::filemtime                    ???: Smarty.class.php
 0.06  0.06    6 include::define                       ???: Smarty.class.php
 0.06  0.04    1 include::Smarty->trigger_error        ???: Smarty.class.php
 0.06  0.06   10 include::count                        ???: Smarty.class.php
```

```
 92  0.02       $html_result .= "</select>\n";
 93         }
 94
 95  0.00     if ($display_seconds) {
 96  0.00         $all_seconds = range(0, 59);
 └   0.01   One call to 'include::range'
 97  0.23         for ($i = 0, $for_max = count($all_seconds); $i < $for_max; $i+= $second_interval)
 └   0.01   One call to 'include::count'
 98               $seconds[] = sprintf('%02d', $all_seconds[$i]);
 └   0.32   60 calls to 'include::sprintf'
 99  0.00         $selected = intval(floor(strftime('%S', $time) / $second_interval) * $second_interval);
 ├   0.01   One call to 'include::strftime'
 ├   0.01   One call to 'include::floor'
 └   0.01   One call to 'include::intval'
100  0.00         $html_result .= '<select name=';
```

<cycle 1>
22.19 %
44.38 %
include::include <cycle 1>
22.19 %
22.19 %
include::smarty_function_html_select_time
22.19 %
20.31 %
include::smarty_function_html_options
20.31 %
17.38 %
include::smarty_function_html_options_optoutput
17.38 %
13.47 %
include::smarty_function_escape_special_chars
13.47 %
2.50 %
2.01 %
1.67 %
include::str_replace
2.50 %
include::preg_replace
2.01 %
include::htmlspecialchars
1.67 %

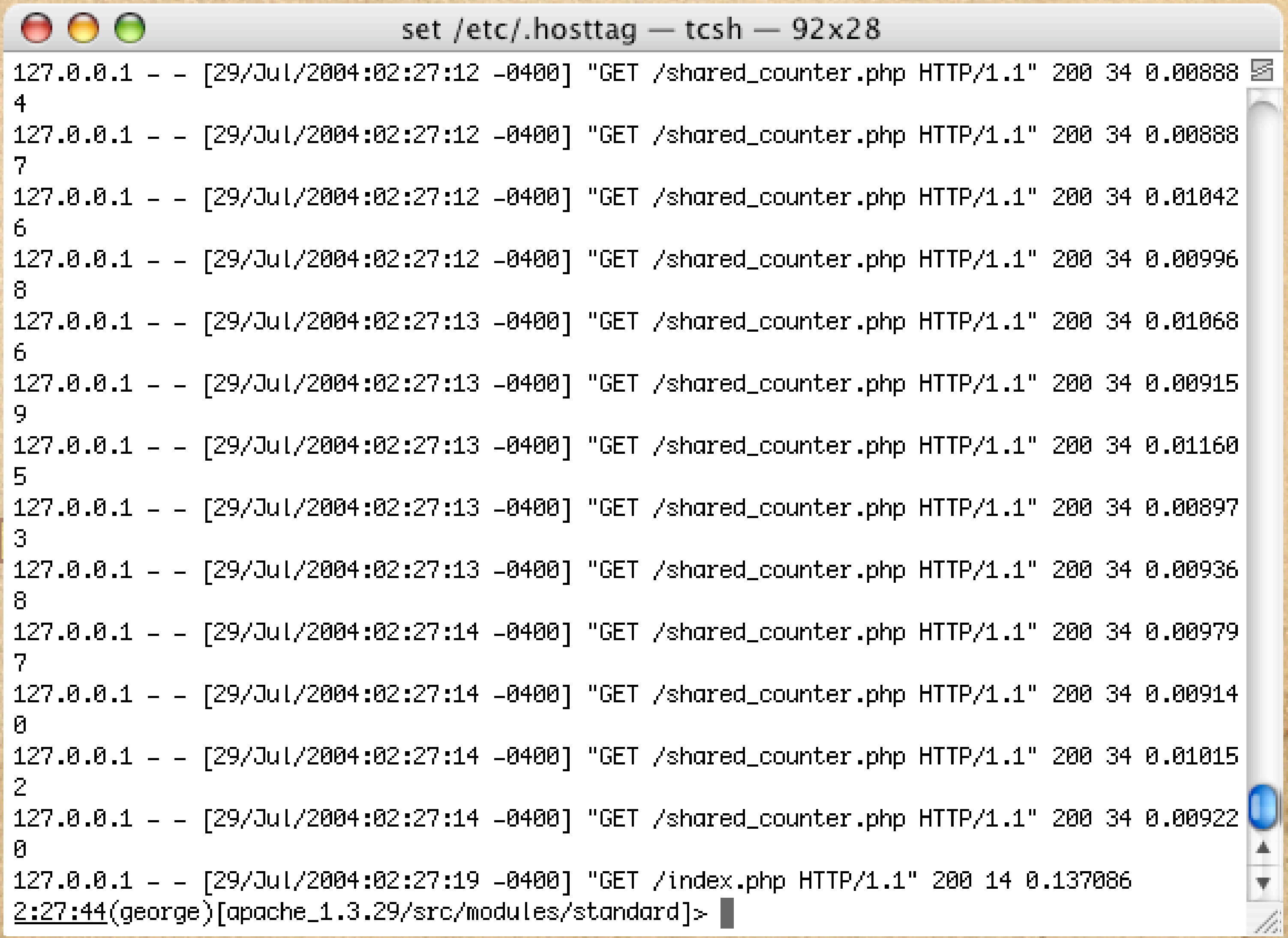
Log Analysis

- Websites have more pages than you think. Remember that performance effects are aggregate.
  - Mild performance issues on frequently accessed pages.
  - Bad performance issues on infrequently accessed pages.
  - Poisoning shared resources (database buffer caches, etc.)

Pinpointing Pages with mod_log_config

Apache 1.3 only supports low resolution timings, but you can patch it.

Apache 2.0 natively supports fine grain timings.



```
127.0.0.1 - - [29/Jul/2004:02:27:12 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00888
4
127.0.0.1 - - [29/Jul/2004:02:27:12 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00888
7
127.0.0.1 - - [29/Jul/2004:02:27:12 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.01042
6
127.0.0.1 - - [29/Jul/2004:02:27:12 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00996
8
127.0.0.1 - - [29/Jul/2004:02:27:13 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.01068
6
127.0.0.1 - - [29/Jul/2004:02:27:13 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00915
9
127.0.0.1 - - [29/Jul/2004:02:27:13 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.01160
5
127.0.0.1 - - [29/Jul/2004:02:27:13 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00897
3
127.0.0.1 - - [29/Jul/2004:02:27:13 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00936
8
127.0.0.1 - - [29/Jul/2004:02:27:14 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00979
7
127.0.0.1 - - [29/Jul/2004:02:27:14 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00914
0
127.0.0.1 - - [29/Jul/2004:02:27:14 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.01015
2
127.0.0.1 - - [29/Jul/2004:02:27:14 -0400] "GET /shared_counter.php HTTP/1.1" 200 34 0.00922
0
127.0.0.1 - - [29/Jul/2004:02:27:19 -0400] "GET /index.php HTTP/1.1" 200 14 0.137086
2:27:44(george)[apache_1.3.29/src/modules/standard]>
```

Some Real World Examples

A Tricky Case

The Initial Profile

```
> pprofp -R pprof.07384.44

Trace for /reports/headlines.php
Total Elapsed Time = 1.50
Total System Time  = 0.01
Total User Time    = 0.11
        Real         User        System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call    s/call  Name
--------------------------------------------------------------------------------------
100.0 0.00 1.50    0.00 0.11    0.00 0.01      1    0.0003   1.4981 main
 99.6 0.00 1.49    0.00 0.11    0.00 0.01      1    0.0000   1.4926 include
 66.1 0.00 0.99    0.00 0.04    0.00 0.00    202    0.0000   0.0049 db_mysqlstatement::fetch_assoc
 65.2 0.98 0.98    0.03 0.03    0.00 0.00    207    0.0047   0.0047 is_resource
 29.3 0.44 0.44    0.00 0.00    0.00 0.00      2    0.2195   0.2195 mysql_query
 29.3 0.00 0.44    0.00 0.00    0.00 0.00      1    0.0000   0.4387 db_mysql::execute
```

That can't be right.

```
65.2 0.98 0.98   0.03 0.03    0.00 0.00    207    0.0047   0.0047 is_resource
```

- is_resource() does an extremely simple check. There's no way that it can be consuming 60% of a real script.
- Is APD broken?

pprof format

- To investigate, we'll need to peak into the raw trace file.

Start Token	Meaning
!	A file is encountered. It is assigned an index (1 here).
&	A function is encountered, it is assigned an index (1) and is noted as a userspace function (2).
+	A function is called. Function 1 called from file 1 at line 2.
@	A timing is recorded at file 1, line 2. 3999 user usecs, 1000 system usecs, 5203 wall-clock usecs.
-	A function call ends. Function (2). Script memory usage is also recorded (but near worthless)

```
#Pprof [APD] v0.9.1
caller=/reports/headlines.php

END_HEADER
! 1 /reports/headlines.php
& 1 main 2
+ 1 1 2
& 2 apd_set_pprof_trace 1
+ 2 1 2
@ 1 2 3999 1000 5203
- 2 50331648
```

Looking into the trace

Looking for is_resource, we find it's declaration:

```
& 13 is_resource 1
```

Next we go looking for it's actual calls. There are many (207), but here is one:

```
+ 22 7 224
+ 13 7 113
@ 7 113 0 0 226
- 13 50331648
```

APD times on function exit, so it's possible that function 22 (`mysql_db::fetch_assoc`)is leaking in a bit of time, but this is still not near the 4700 usec average quoted back from APD.

Finding the Outliers

Let's look for the outliers that are weighting the average time. Here are a sample of the timings that are being reported for is_resource().

```
@ 7 113 0 0 196
...
@ 7 113 999 0 229
...
@ 7 113 0 0 196
...
@ 7 113 1000 0 286
...
@ 7 113 0 0 197
...
@ 7 113 0 0 197
...
@ 7 113 0 0 274140
...
@ 7 113 0 0 123
```

Looking at caller's context

Here is the context that is calling is_resource():

```
function fetch_assoc() {
  if(!is_resource($this->result)) {
    return false;
  }
  return mysql_fetch_assoc($this->result);
}
```

That looks fine, let's go one step up the callstack:

```
while($y = $sth->fetch_assoc()) {
  //  lots of stuff
  // lots of printing
}
```

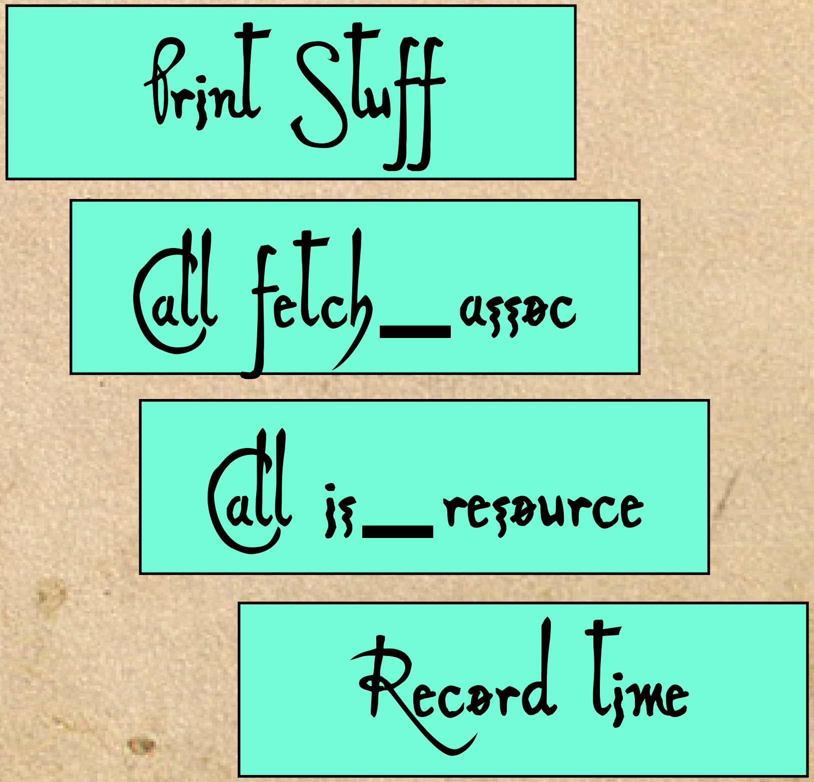
Aha!

To solve our mystery we need two hints:

- (Without statement tracing) APD traces function calls, not language constructs.
- Timings are done at function exit.

So there is a cost in print which is being miscategorized into is_resource().

```
while($y = $sth->fetch_assoc()) {
  //  lots of stuff
  // lots of printing
}
```



Buffering Issues

So, the problem is that PHP is blocking while the OS flushes it's TCP buffer on the client socket. To handle this you can:

- Enable output buffering in PHP and attempt to size the buffer chain (PHP => Apache => OS => Client) to allow the entire page to fit in a single TCP buffer.
- Enable output compression to help the pages fit into a reasonable buffer size.

After Adding Buffering

Always measure your changes! Here is the same page with output buffering and compression on:

```
Trace for /reports/headlines.php
Total Elapsed Time = 0.15
Total System Time  = 0.00
Total User Time    = 0.08
```

Much better!

Proactive Profiling

The Initial Call

```
> pprofp -R pprof.10089.0

Trace for /reports/bank/us/index.php
Total Elapsed Time = 0.07
Total System Time  = 0.01
Total User Time    = 0.06
         Real          User         System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls     call     s/call  Name
--------------------------------------------------------------------------------------
 97.3 0.00 0.07    0.00 0.05    0.00 0.01       1     0.0008    0.0693 main
 48.3 0.00 0.03    0.00 0.03    0.00 0.00       1     0.0000    0.0344 generatepagexsl
 48.2 0.01 0.03    0.01 0.03    0.00 0.00       4     0.0029    0.0086 require_once
 32.7 0.02 0.02    0.02 0.02    0.00 0.00       1     0.0233    0.0233
fastxsl_prmcache_transform
 11.9 0.01 0.01    0.01 0.01    0.00 0.00       1     0.0085    0.0085 fastxsl_xml_parsestring
 10.8 0.01 0.01    0.01 0.01    0.00 0.00       1     0.0077    0.0077 apd_set_pprof_trace
  7.9 0.00 0.01    0.00 0.00    0.00 0.00       5     0.0000    0.0011 db_mysql::execute
  6.2 0.00 0.00    0.00 0.00    0.00 0.00       1     0.0000    0.0044 report->getalertsxml
  5.9 0.00 0.00    0.00 0.00    0.00 0.00       6     0.0007    0.0007 mysql_query
  5.6 0.00 0.00    0.00 0.00    0.00 0.00       1     0.0001    0.0040 generatepagexml
```

Subsequent Calls

```
> pprofp -R pprof.10089.6

Trace for /reports/bank/us/index.php
Total Elapsed Time = 0.05
Total System Time  = 0.01
Total User Time    = 0.04

         Real         User          System             secs/    cumm
%Time (excl/cumm)  (excl/cumm)  (excl/cumm) Calls    call     s/call  Name
--------------------------------------------------------------------------------------
 92.1 0.00 0.04    0.00 0.03    0.00 0.01       1    0.0011    0.0418 main
 19.7 0.00 0.01    0.00 0.01    0.00 0.00       1    0.0001    0.0089 generatepagexsl
 14.8 0.00 0.01    0.00 0.01    0.00 0.00       1    0.0000    0.0067 report->getalertsxml
 14.0 0.00 0.01    0.00 0.00    0.00 0.00       5    0.0000    0.0013 db_mysql::execute
 12.2 0.00 0.01    0.00 0.00    0.00 0.00       1    0.0000    0.0055 generatepagexml
```

19.7% of script runtime in XSLT Templatization.

Not Shabby.

Thank You!

Questions? feel free to mail me anytime at

george@omniti.com