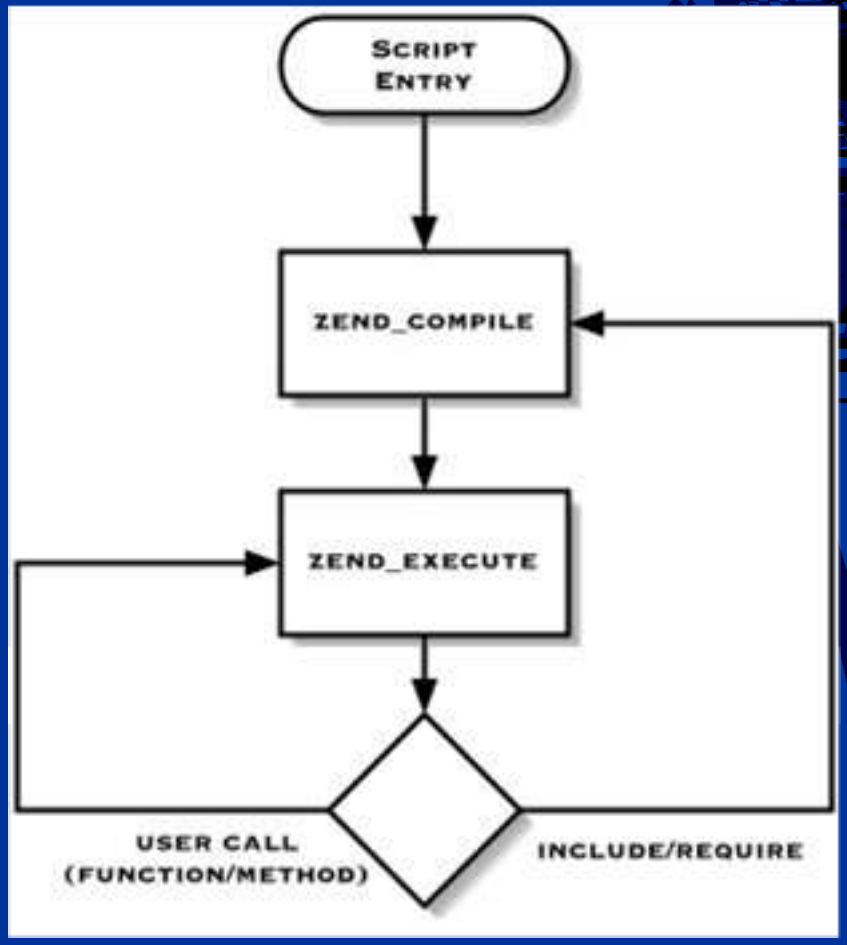
Accelerating PHP Applications

Ilia Alshanetsky

ilia@ilia.ws

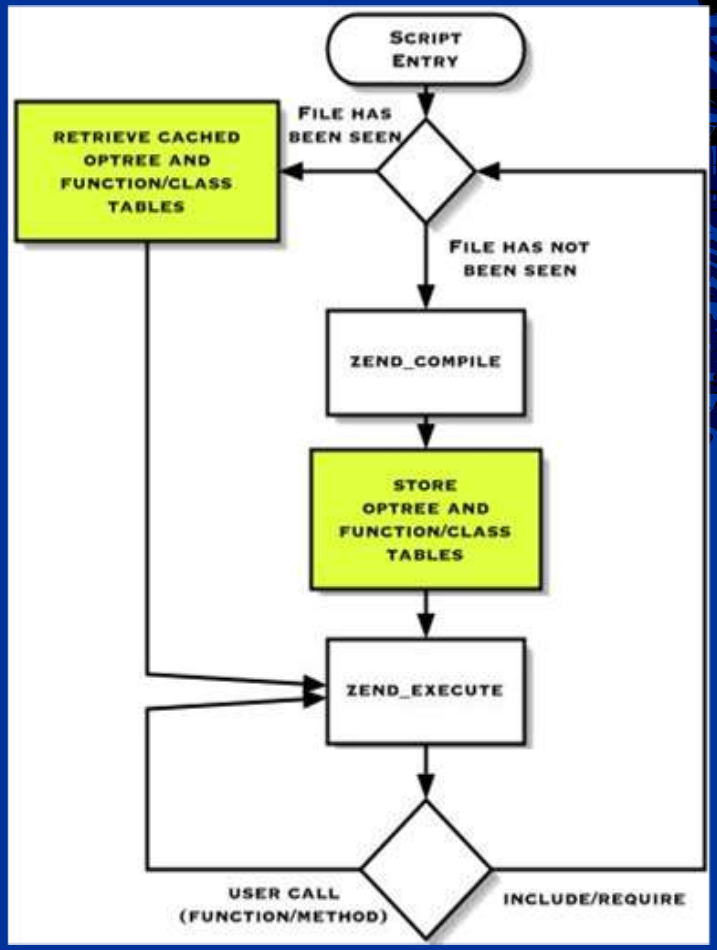
Bytecode/Opcode Caches



This cycle happens for every include file, not just for the "main" script.

Compilation can easily consume more time than execution.

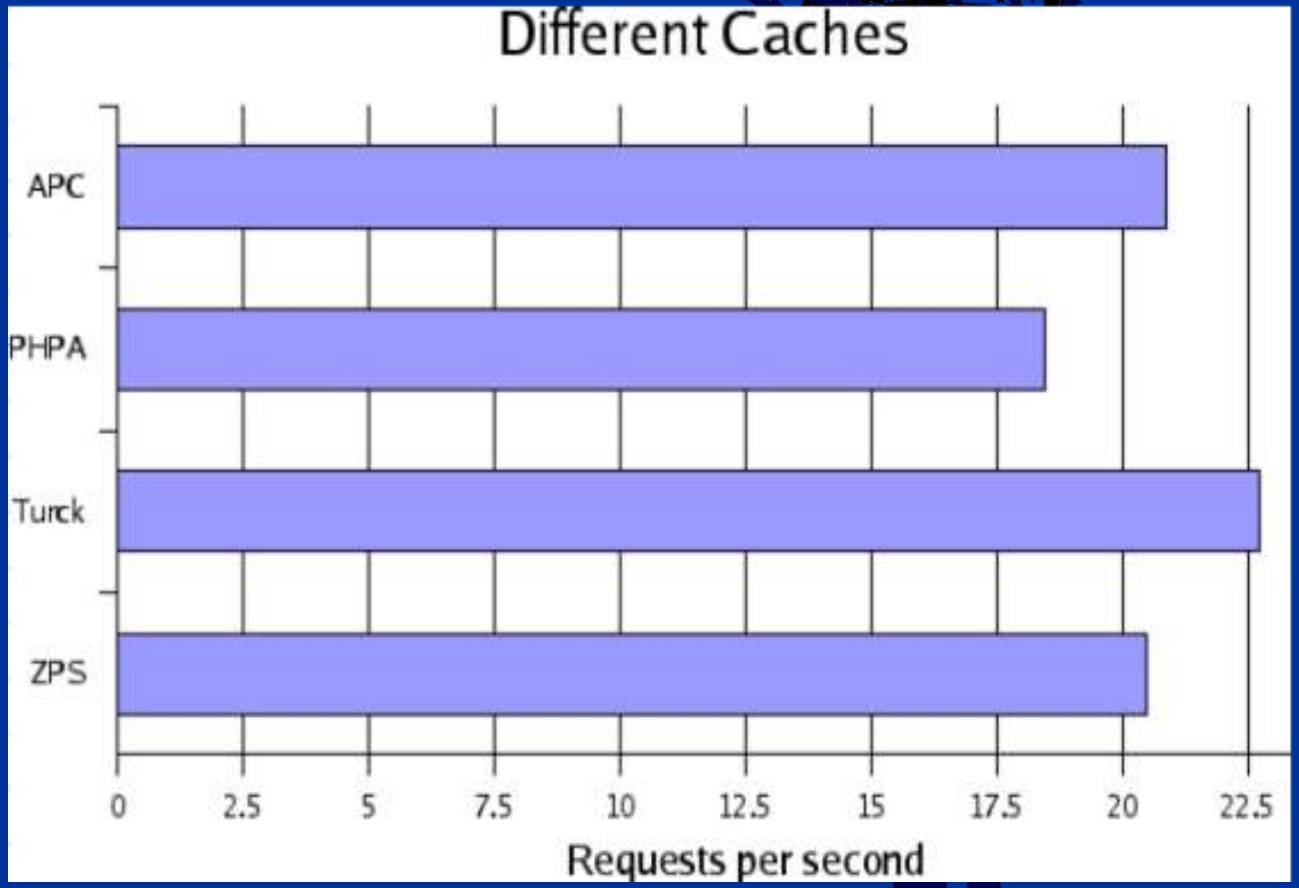
Bytecode/Opcode Caches



Benefits:

- Each PHP script is compiled only once for each revision.
- Reduced File IO thanks to opcodes being read from memory rather then being parsed from disk.
- Since compilation is one time event, generated opcodes can optimised for faster execution.

Cache Implementations



Implementations:

- APC
- Zend Performance Suite
- Turck MM Cache / eAccelerator
- PHP Accelerator

Compiler Optimisations

For absolute maximum performance it may be a good idea to ensure that all software is compiled to the take maximum advantage of the available hardware.

- Enable all compiler optimizations with -O3
- Make the compiler tune the code to your CPU via -march -mcpu (use caution).
- Try to make the compiler use CPU specific features -msse -mmmx -mfpmath=sse

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

Reduce Binary/Library Size

Eliminate waste by removing symbols from object files using **strip** utility.

Saves disk space and more importantly memory needed to load the library or run the binary. In case of PHP makes the binary or Apache loadable module 20-30% smaller.

Very useful for CLI/CGI PHP binaries.

Apache/PHP Integration

For maximum performance compile PHP statically into Apache (up to 30% speed increase). Or use PHP 4.3.11+ where **--prefer-non-pic** is default.

How to compile PHP statically into Apache

```
# PHP
./configure --with-apache=/path/to/apache_source

# Apache
./configure --activate-module=src/modules/php4/libphp4.a
```

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 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

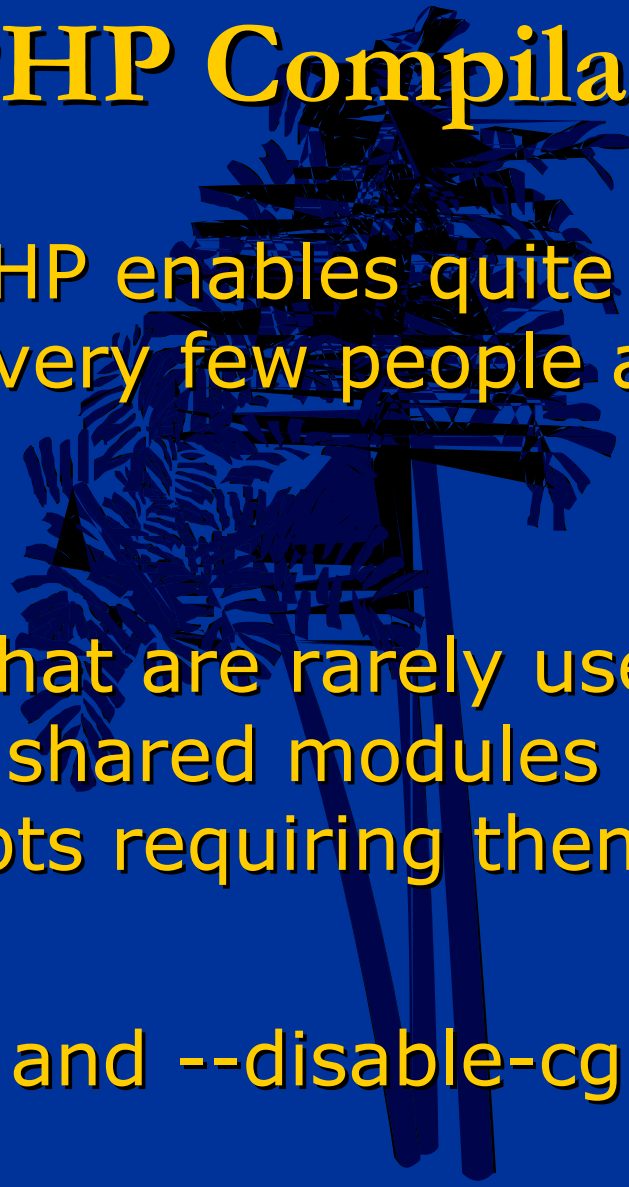
Web Server: KeepAlive

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 (10 secs).

If the server is only serving dynamic requests, disable KeepAlive.

PHP Compilation

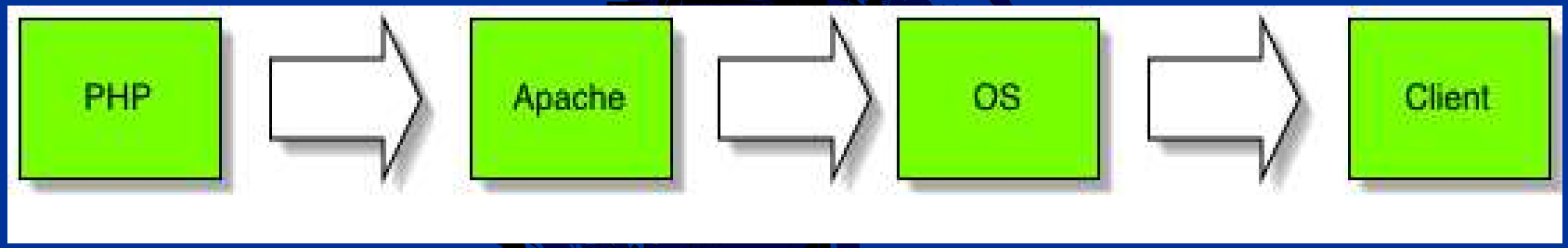


By default PHP enables quite a few extensions, very few people actually need them.

Extensions that are rarely used, should be compiled as shared modules and loaded by the few scripts requiring them.

--disable-all and --disable-cgi simplify PHP builds.

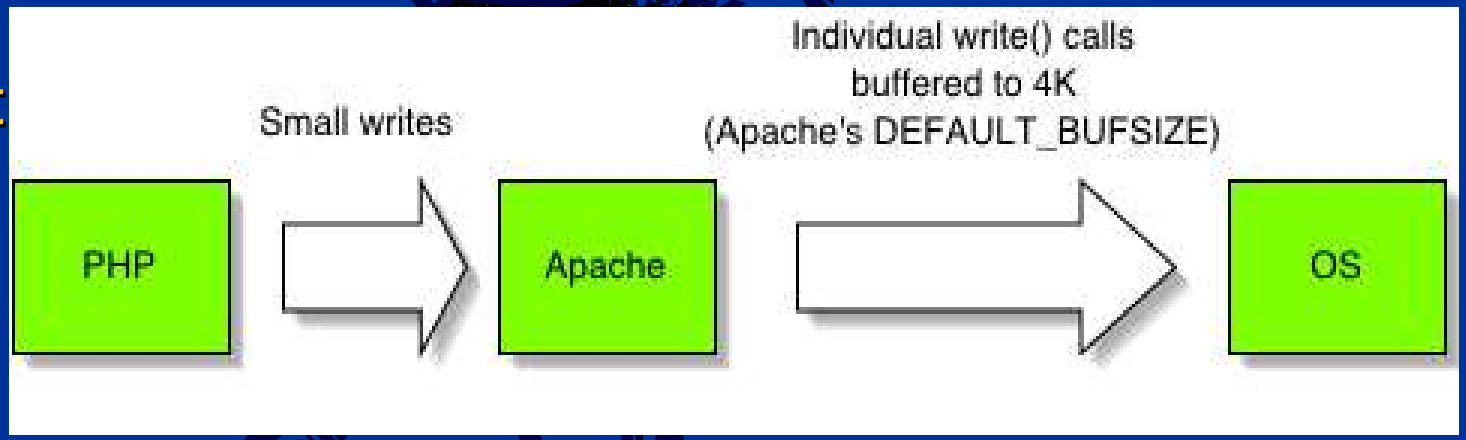
Matching Your IO Sizes



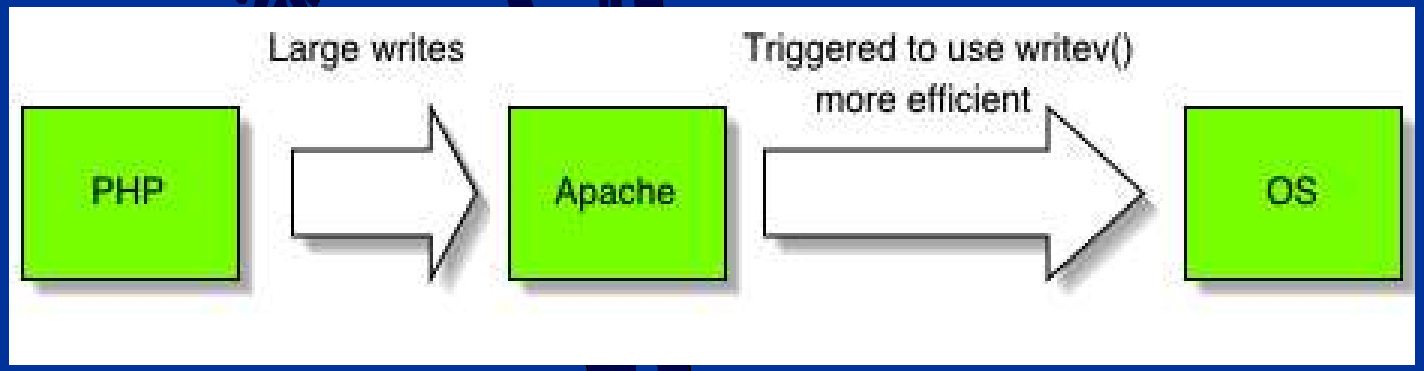
- The goal is to pass off as much work to the kernel as efficiently as possible.
- Optimizes PHP <--> OS Communication
- Reduces Number Of System Calls

Output Buffering

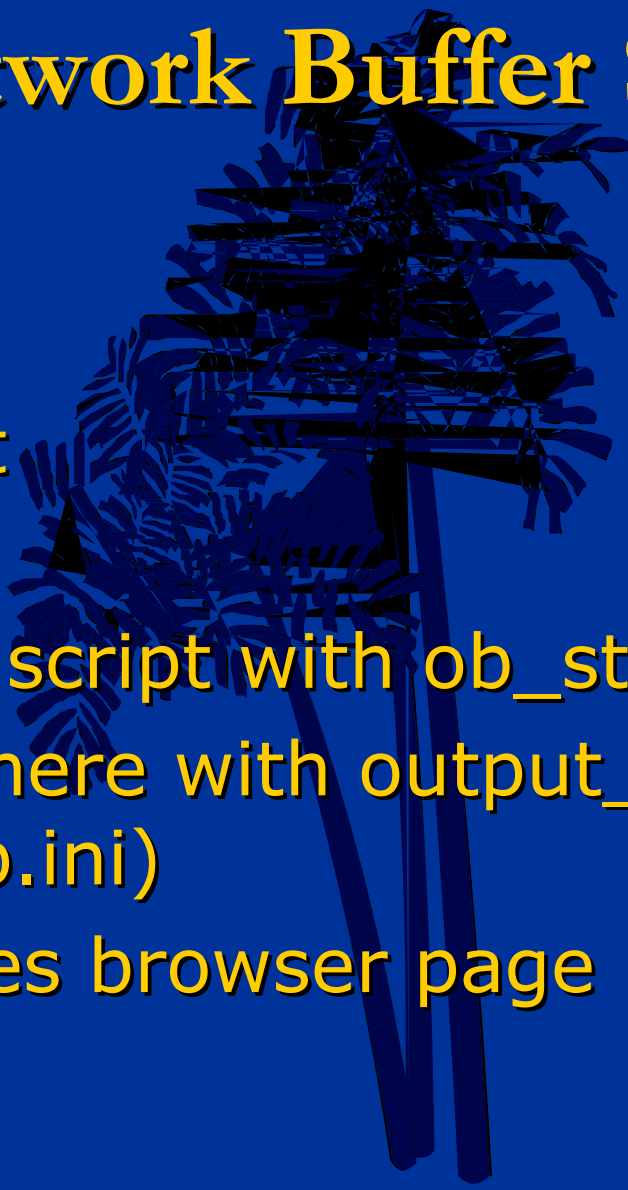
Without Output Buffer



With Output Buffer



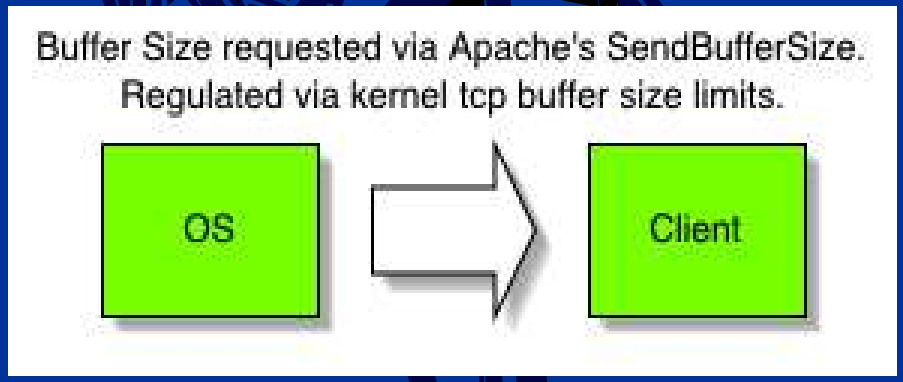
Network Buffer Sizing



- Efficient
- Flexible
- In your script with ob_start()
- Everywhere with output_buffering = On (php.ini)
- Improves browser page rendering speed.

Output Buffering

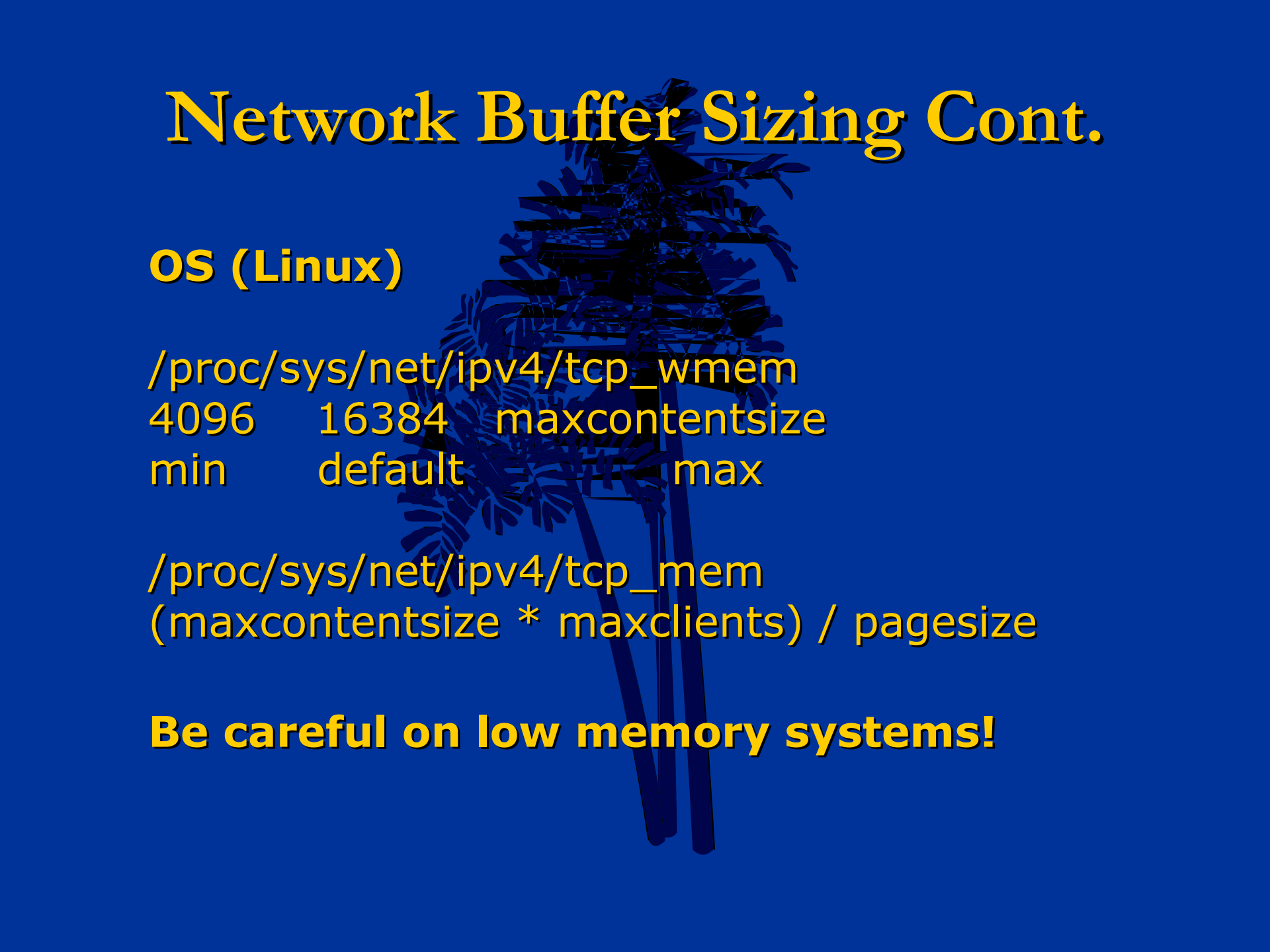
Idea is to hand off entire page to the kernel without blocking.



In Apache:

SendBufferSize = PageSize

Network Buffer Sizing Cont.



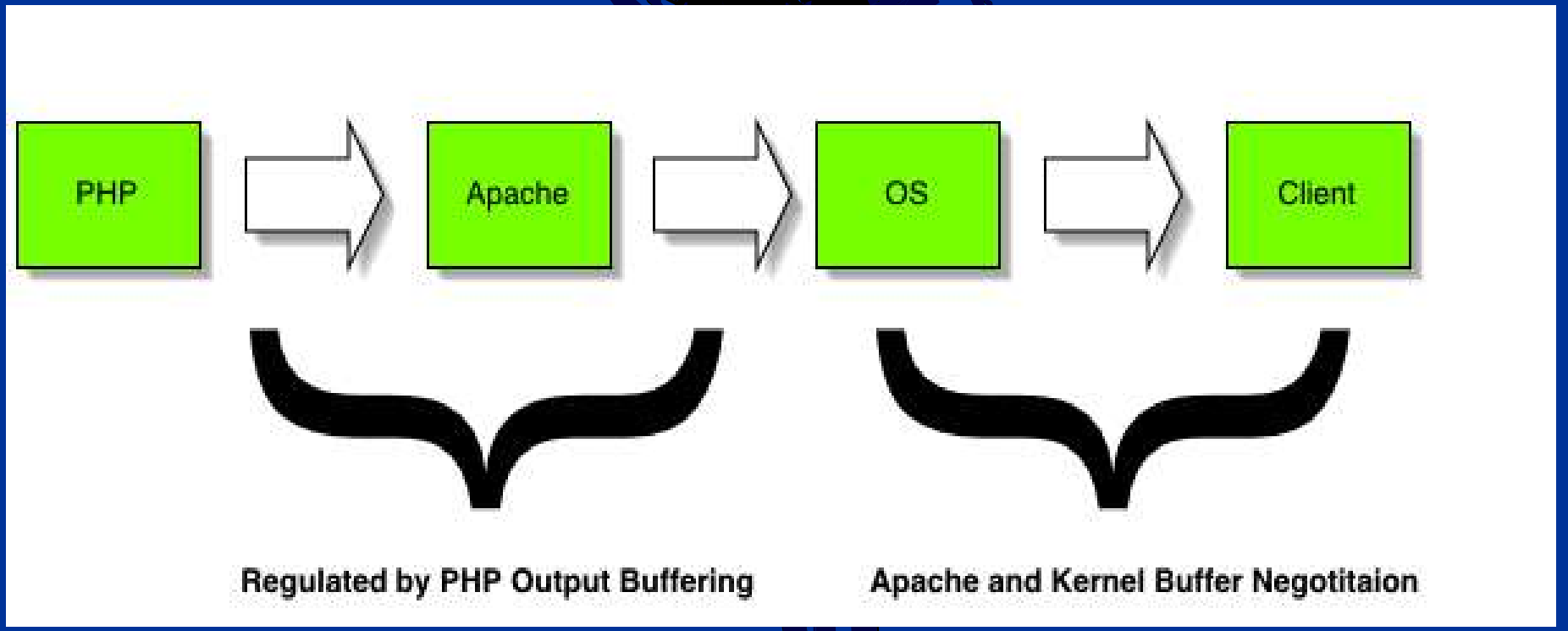
OS (Linux)

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

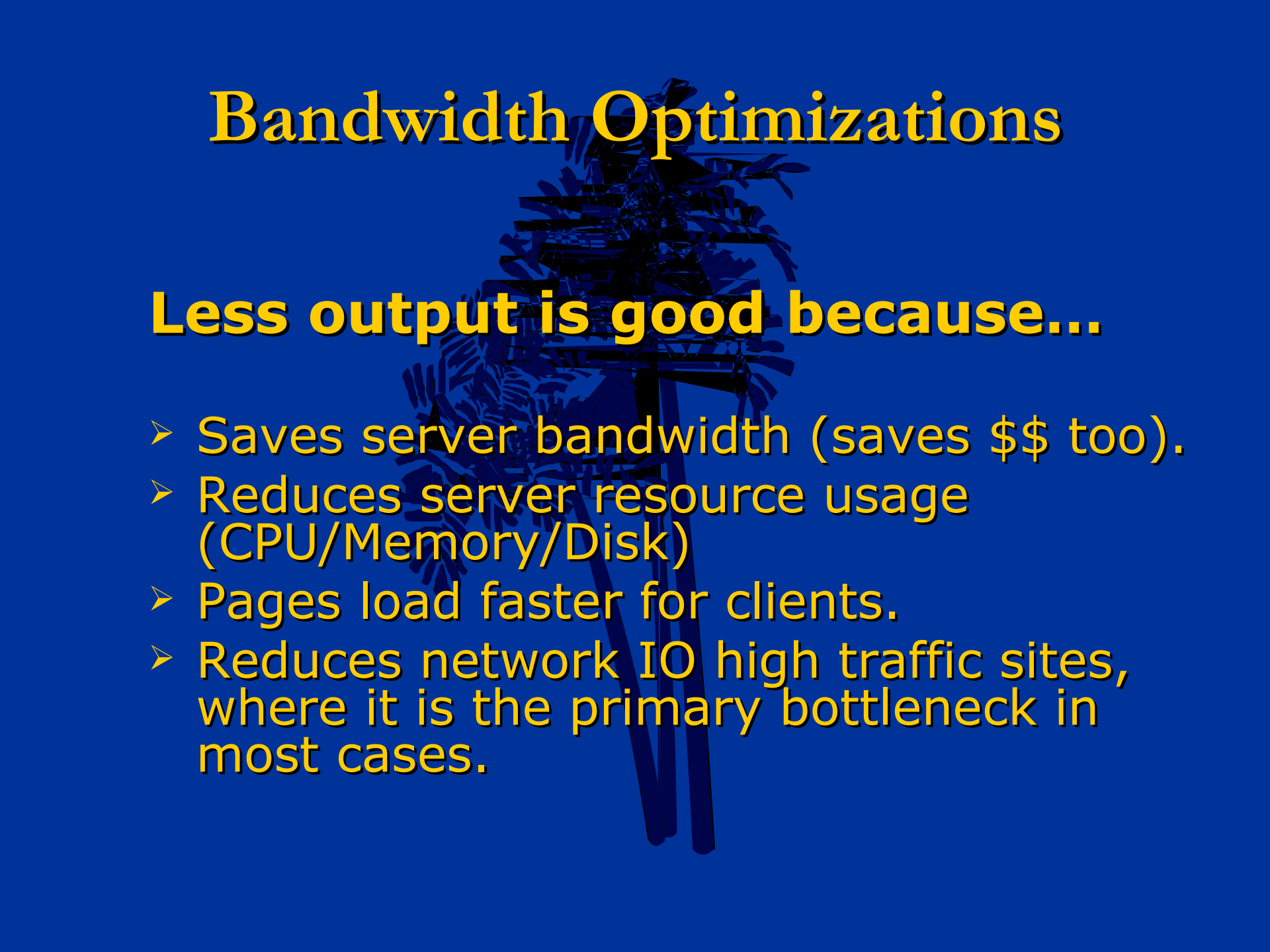
/proc/sys/net/ipv4/tcp_mem
(maxcontentsize * maxclients) / pagesize

Be careful on low memory systems!

Final Picture



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.

Content Compression

- Most browser support retrieval of compressed pages decompressing them before rendering.
- Compressed pages are on average are 7-10 times smaller, however compression can take 3%-5% of CPU.

Implementations:

- Apache 1 (mod_gzip)
- Apache 2 (mod_deflate)
- PHP
  - php.ini (zlib.output_compression=1)
  - script (ob_start("ob_gzhandler"))

Content Reduction

Use preprocessor such as **tidy** extension to eliminate white-space and any unnecessary components from final HTML output.
(5-10% reduction on average)

```
<?php
$opts = 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", $opts);

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

Tools

- Profiling/Benchmarking Web Server
  - Apache Bench (http://apache.org)
  - httperf (http://freshmeat.net/projects/httperf/)
- PHP Profilers
  - DBG (http://dd.cron.ru/dbg/)
  - APD (PECL)
  - Xdebug (http://xdebug.org/)

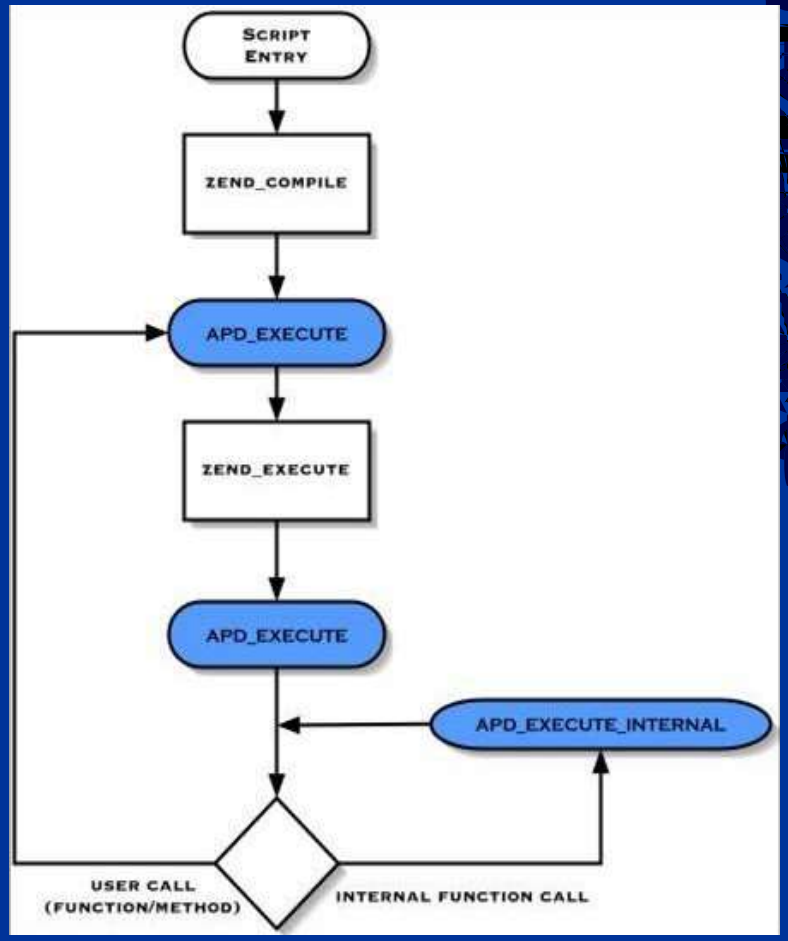
Web Server Testing

```
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.

How to Install:

pear install apd

Then add the following to your php.ini

zend_extension=/path/to/apd.so

Generating A Trace

Profiling of script is started from the point when The **apd_set_pprof_trace()** function is called. All code executed prior will not be profiled.

```
<?php
$parts = preg_split("!\s!", "a b c");

function test(&$var) {
    $var = base64_encode(trim($var));
}

apd_set_pprof_trace();

array_walk($parts, 'test');
?>
```

To avoid having to modify every file in the application, you can use the **auto_append_file** php.ini setting to activate profiling for entire application.

Understanding The Trace

%Time	Real (excl/cumm)	User (excl/cumm)	System (excl/cumm)	secs/ call (excl/cumm)	Calls	cumm call	s/call	Name
82.4	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	3	0.0000	0.0000	trim
4.3	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	3	0.0000	0.0000	test
0.6	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	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.

Bad Approach:

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

Performance Friendly Approach:

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

Regular Expressions

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

```php
<?php
// 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")) { }
?>
```

Don't Reinvent the Wheel

PHP has hundreds functions and tens of extension. Rather then implementing functionality in PHP, try to see if the task can be done via a native function.

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

// vs the much simpler

$data = file_get_contents("some_file");
?>
```

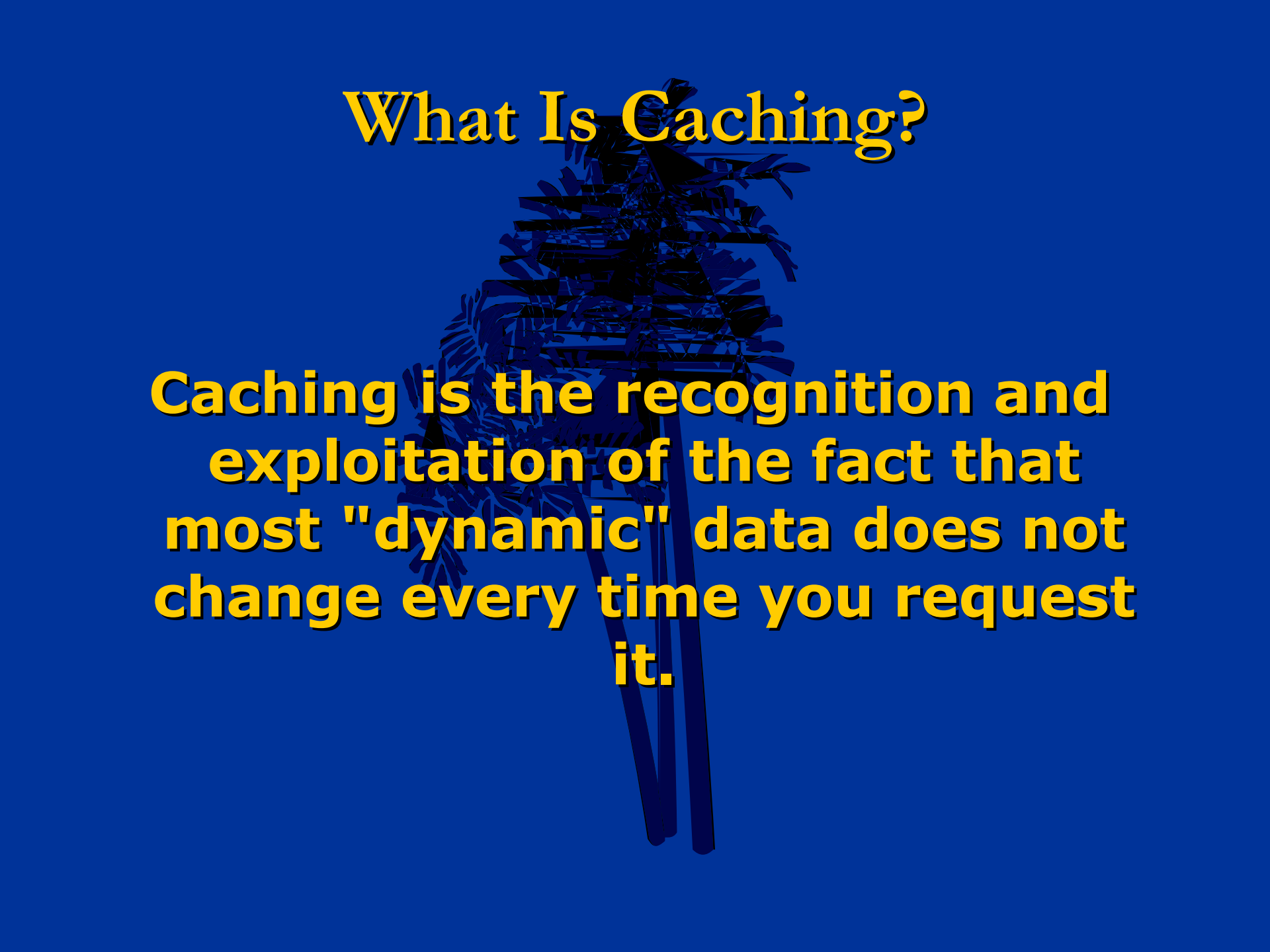
Reference Tricks

References can be a valuable tool to simplify and accelerate access to complex data types as well as a memory saving tool.

```php
<?php
$a['b']['c'] = array();
// slow 2 extra hash lookups
per access
for($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;
?>
```

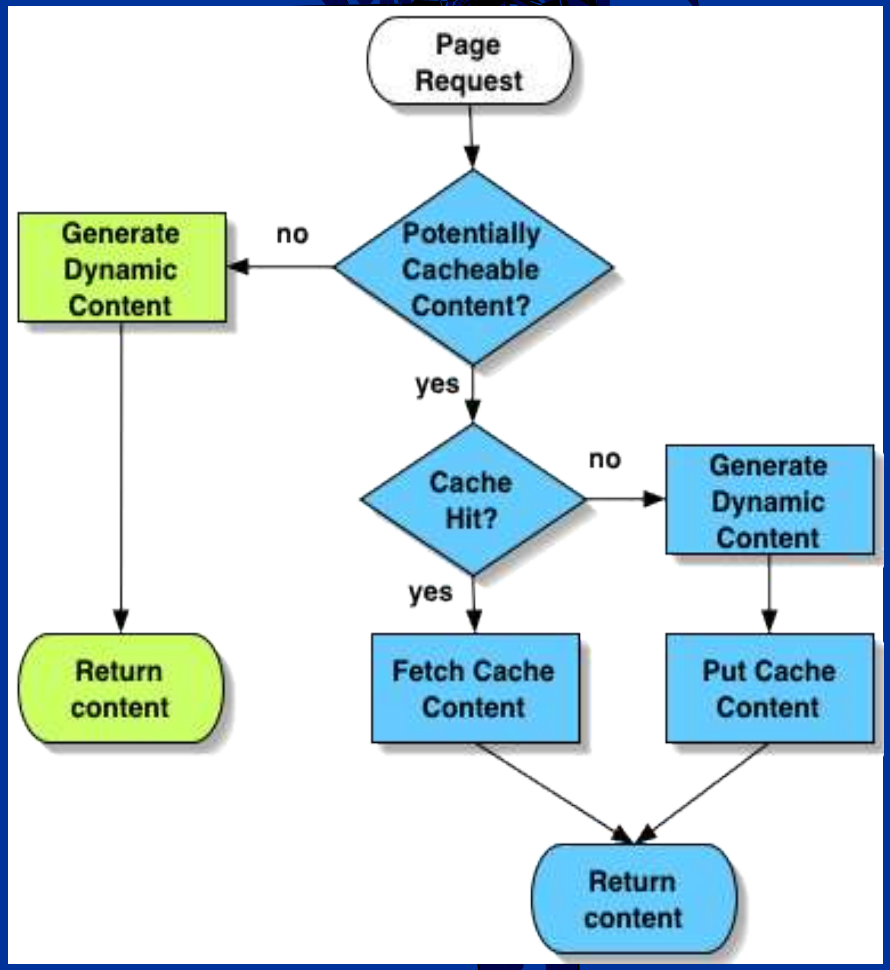
```php
<?php
$a = "abc";
// memory intensive
solution
function a($str) {
    return $str . "d";
}
$a = a($a);
// more effecient approach
function b(&$str) {
    $str .= "d";
}
b($a);
?>
```

What Is Caching?



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

How Does It Work?



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

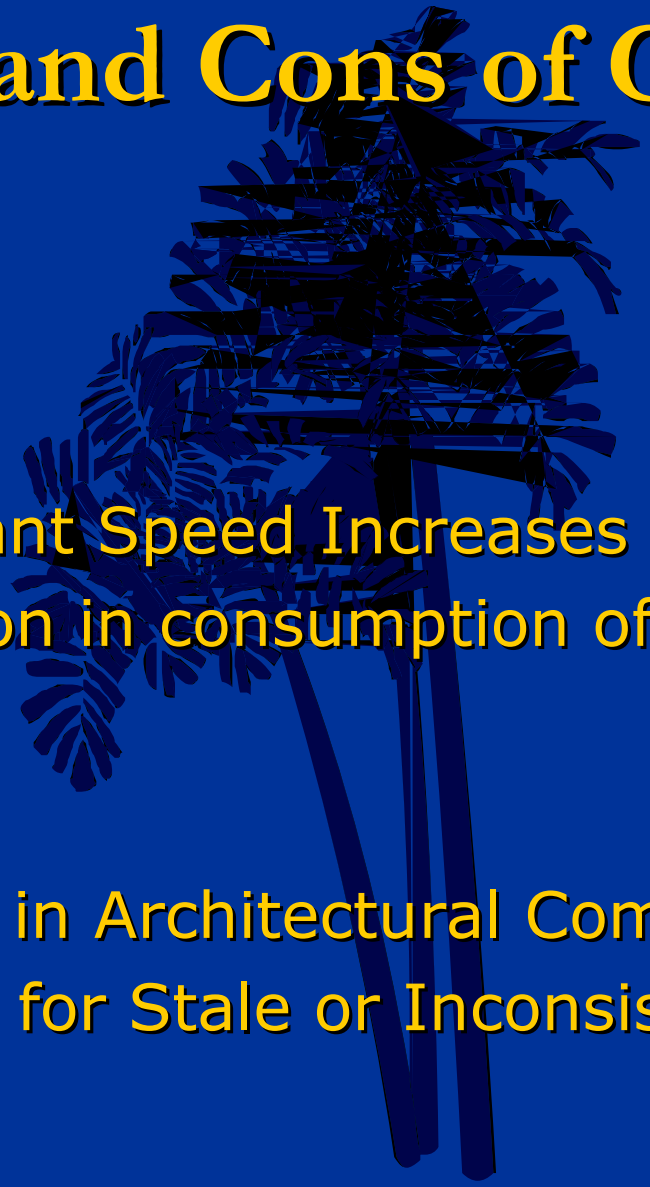
```php
<?php
require "./cache.php"; // our cache code
// Simple guestbook script.
$db = new sqlite_db("gb.sqlite");
$r = $db->array_query("SELECT * FROM guestbook");
foreach ($r as $row)
 echo $r->user . ' wrote on ' .
      date("Ymd", $r->date) . ":<br />\n" .
      $r->message . "<hr /><hr />";
?>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

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"
```

Pros and Cons of Caching



- Pros:
  - Significant Speed Increases
  - Reduction in consumption of some resources
- Cons:
  - Increase in Architectural Complexity
  - Potential for Stale or Inconsistent Data

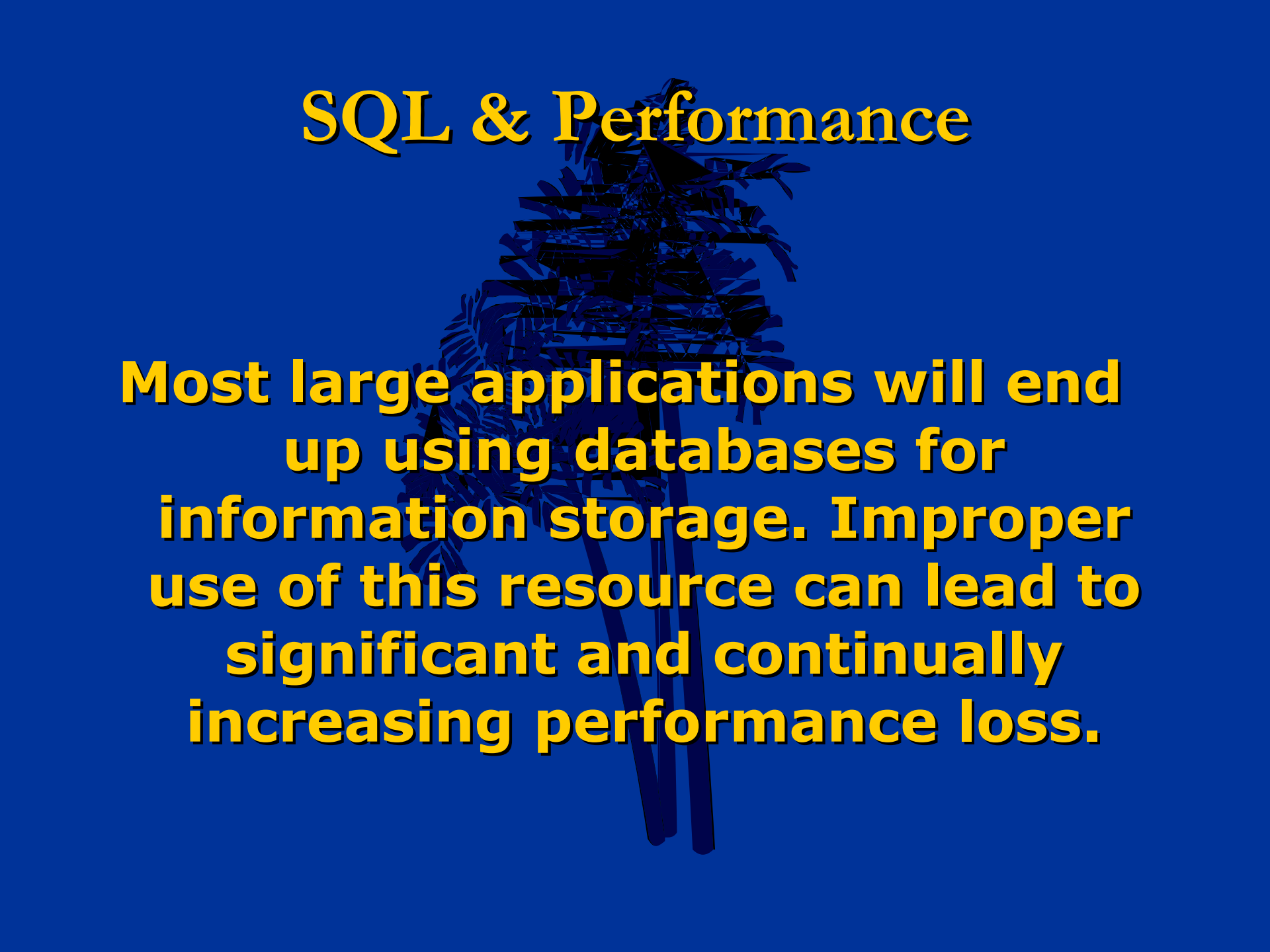
On-Demand Caching

Set up a 404 error handler in .htaccess:

```
RewriteEngine on
RewriteRule /.*\.[^h][^t][^m][^l]$ /$1.html
ErrorDocument 404 /index.php
DirectoryIndex index.php
```

```php
<?php
if (!empty($_SERVER['REDIRECT_URL'])) {
 // This is the requested page that caused the error
 $current_page = substr($_SERVER['REDIRECT_URL'], strlen
(WEBBASE));
}
/* content generation */
if (!FORCE_DYNAMIC) {
 echo $contents = ob_get_clean();
 file_put_contents($lang."/".$current_page.".html", 'w');
}
?>
```

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 offer mechanisms to analyze query execution and determine if it's running in an optimal manner.

SLOW

```
EXPLAIN select * from mm_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 mm_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 |
+----------+-------+---------------+-------+---------+------+------+------------+
```

Optimize Query Execution

Executing one query at a time is boring (and slow), chain them and execute many queries at once quickly.

```php
<?php
// Slow Approach
for ($i = 0; $i < 10; $i++)
	mysql_query("INSERT INTO foo VALUES({$i})");

// Faster MySQL Specific approach
$query = "INSERT INTO foo VALUES";
$query = "(" . implode("),(", array_keys(array_fill(0, 10, 1))).")";
mysql_query($query); // INSERT INTO foo VALUES (0), (1), (2)

// Query Chaining for DBs that support it PostgreSQL, MSSQL, SQLite
$query = '';
for ($i = 0; $i < 10; $i++)
	$query .= "INSERT INTO foo VALUES({$i});";
mysql_query($query);
?>
```

Use Joins

Usage of joins allows simplification & acceleration Of the script by moving portions of the logic to the database engine.

```
<?php
// slow join less approach
$a = sqlite_fetch_single($db,
      "SELECT id FROM foo WHERE name='ilia'");
$b = sqlite_array_query($db,
      "SELECT * FROM bar WHERE id={$a}");
// Fast Join implementation
$b = sqlite_array_query($db,
    "SELECT b.* FROM foo f INNER JOIN bar b ON f.id=b.id
WHERE f.name='ilia'");
?>
```

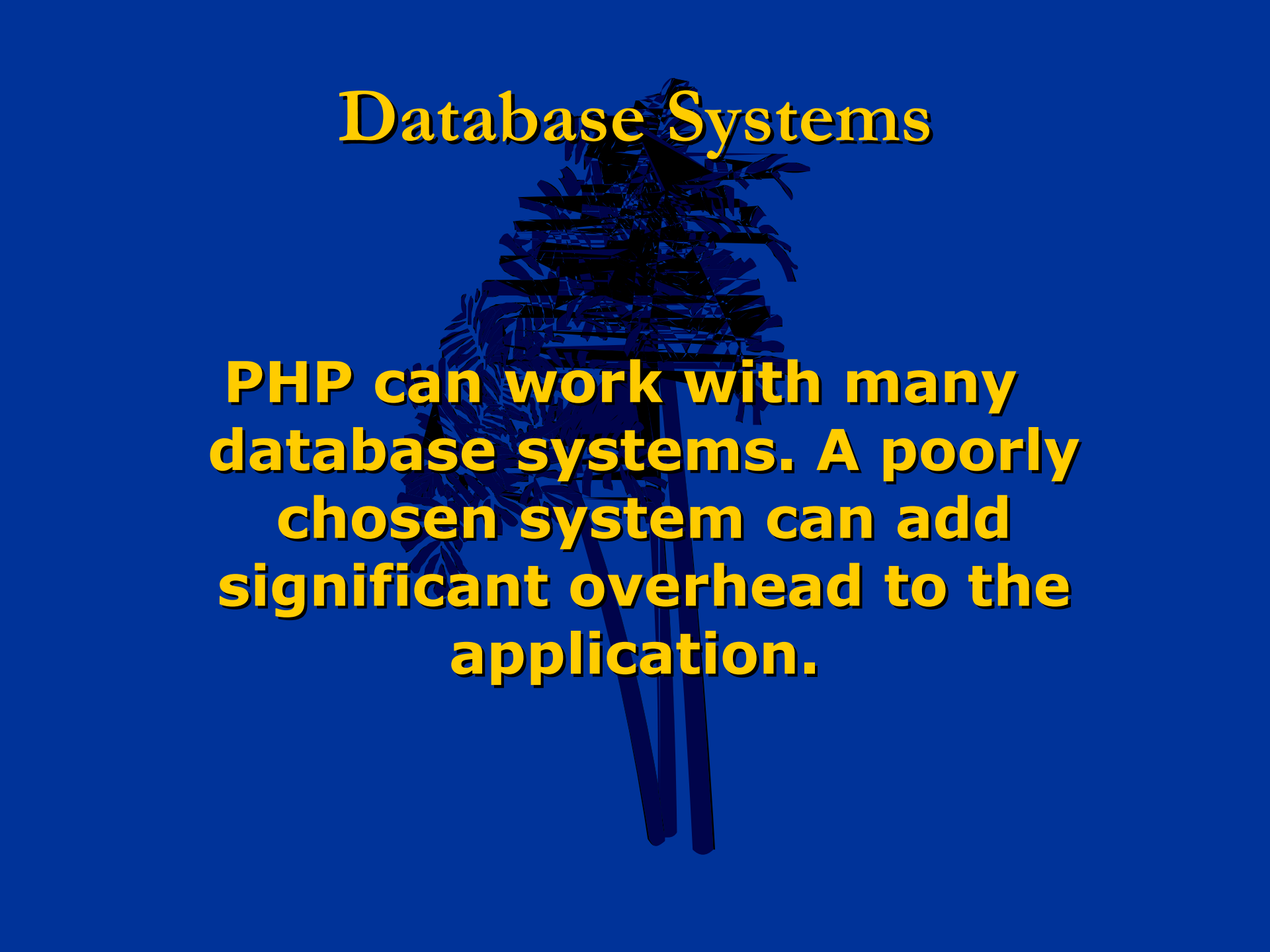
Sub-Queries

Like Joins, Sub-queries can be used to move some Of the logic from PHP into the database engine.

```
<?php
$b = sqlite_array_query($db,
    "SELECT * FROM bar WHERE id=(SELECT id FROM foo
WHERE name='ilia')");
?>
```

While sub-queries save you from the complexity of joins, they are often slower then equivalent joins.

Database Systems



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

Questions