



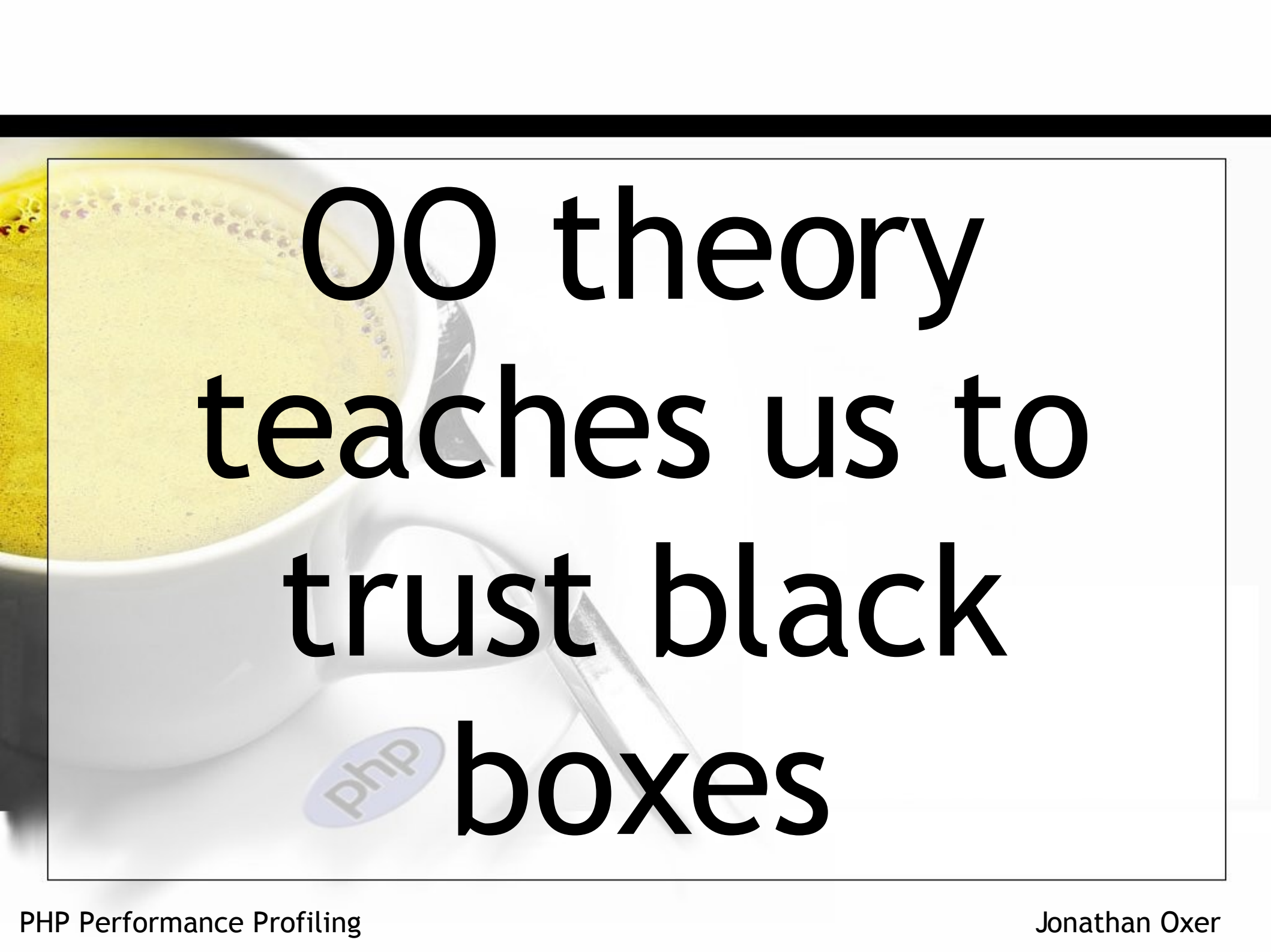
PHP Performance Profiling

Jonathan Ozer


January 30th, 2007
Internet Vision Technologies
Melbourne, Australia

A white coffee cup filled with a frothy beverage, topped with a spoon. The cup sits on a white saucer with a blue oval logo containing the letters 'PHP'. The entire scene is set against a background of a white surface. A large, black-bordered text box is superimposed over the center of the image, containing the text 'Do you know what your code is doing? Really?' in a bold, black, sans-serif font.

**Do you know
what your code
is doing? Really?**

A white coffee cup filled with a frothy beverage, with a silver spoon resting on the saucer. A blue oval logo with the letters 'PHP' is visible on the saucer. The background is a light, neutral color. Overlaid on this image is large, bold, black text.


OO theory
teaches us to
trust black
boxes



**Function
libraries have
the same
problem**



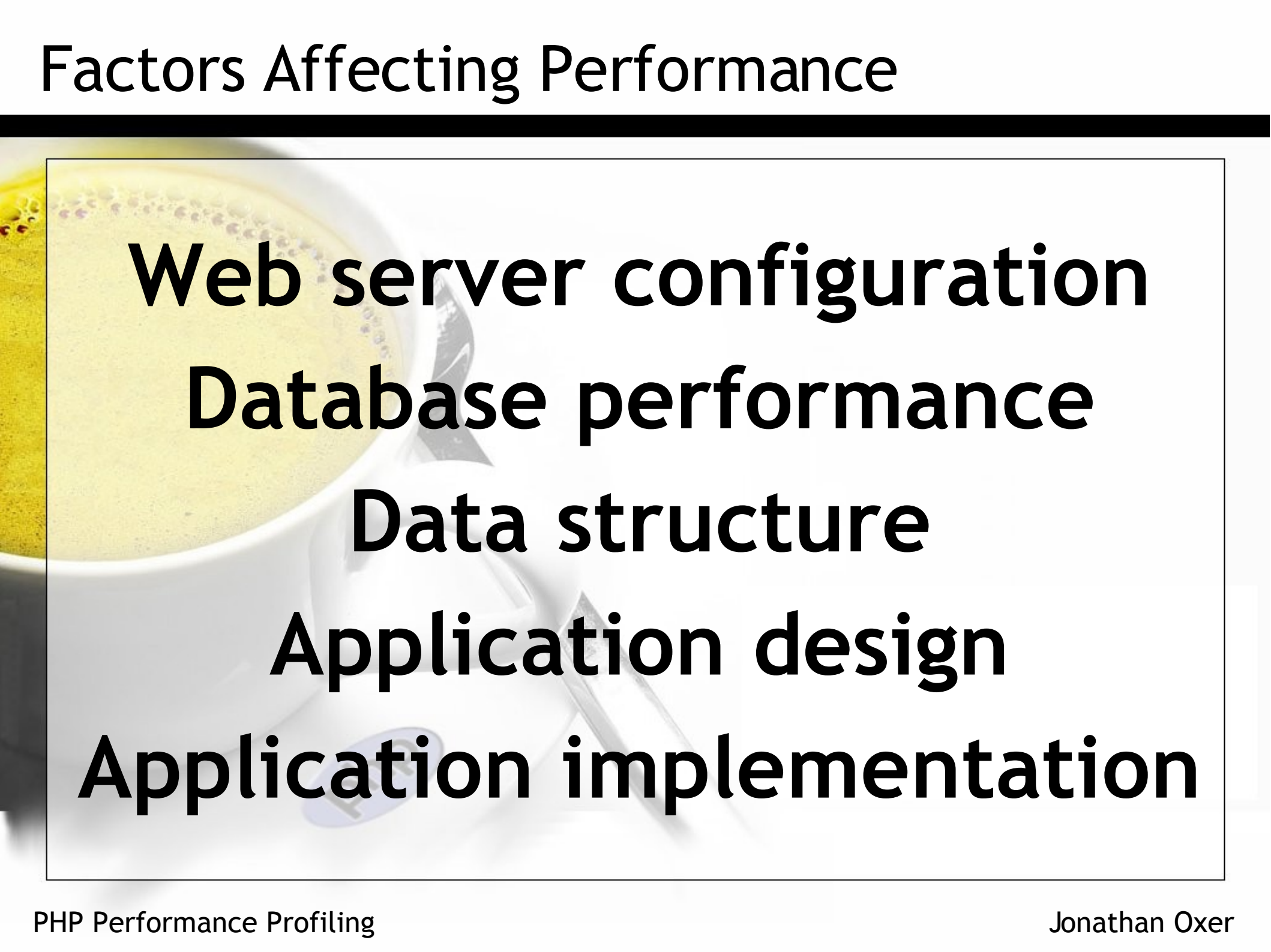
Result: performance problems

commodore  VIC 20


POWER



Factors Affecting Performance

A background image showing a white cup of coffee with a light brown foam on top. A silver pen and a white notepad are visible behind the cup. The text is overlaid on this image.

Web server configuration
Database performance
Data structure
Application design
Application implementation



**Don't guess.
Measure.**

Profiling: Objective Analysis

“Profiling” means running your code in a controlled environment to see what *actually* happens, not what you *think* happens.

Profiling Tools



“userspace”

or

“in-engine”

Profiling Tools

- Assorted timing classes
- Zend Studio
www.zend.com
- DBG
dd.cron.ru/dbg
- XDebug
www.xdebug.org
- APD (Advanced PHP Debugger)
pear.php.net/apd

Getting APD

- Source:
pear.php.net/apd
- PEAR:

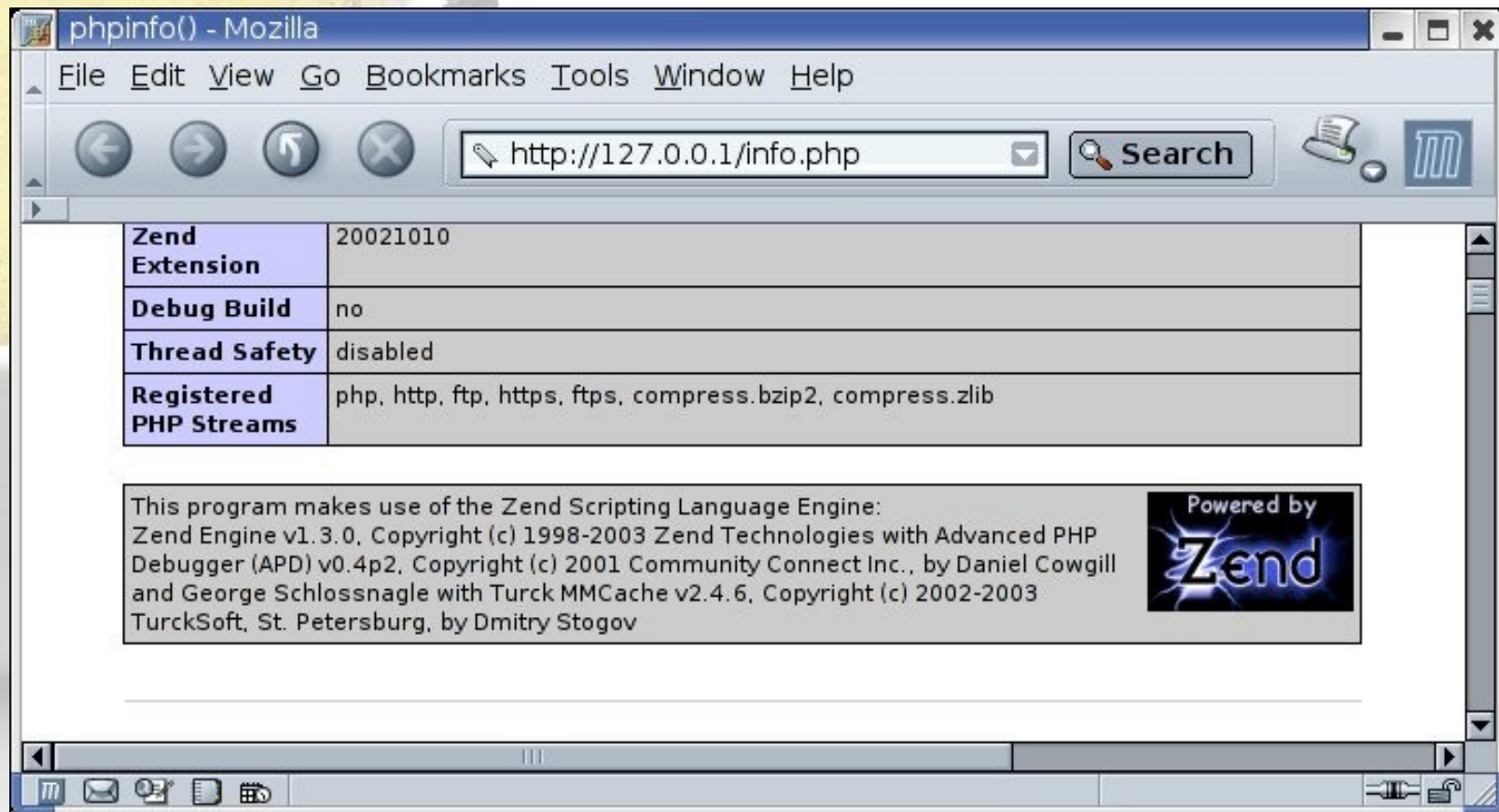
```
# pear install apd
```
- Debian package:

```
# apt-get install php4-apd
```

Checking Your Installation

APD's presence will be reported by phpinfo():

```
# echo "<?php phpinfo(); ?>" > /var/www/info.php
```



Zend Extension	20021010
Debug Build	no
Thread Safety	disabled
Registered PHP Streams	php, http, ftp, https, ftps, compress.bzip2, compress.zlib

This program makes use of the Zend Scripting Language Engine:
Zend Engine v1.3.0, Copyright (c) 1998-2003 Zend Technologies with Advanced PHP Debugger (APD) v0.4p2, Copyright (c) 2001 Community Connect Inc., by Daniel Cowgill and George Schlossnagle with Turck MMCache v2.4.6, Copyright (c) 2002-2003 TurckSoft, St. Petersburg, by Dmitry Stogov

Powered by
Zend

Your First Test

Add a call to APD to start the profiler at the top of your script:

```
apd_set_pprof_trace();
```

Then load the page.

Gathering Data

APD stores raw data as tracefiles in “`apd.dumpdir`”, which is defined in `php.ini`. Typically:

`/var/log/php4-apd/`

Tracefiles are not for human consumption.

Gathering Data

Tracefiles are intended to be machine-parsed to generate reports:

```
pprofp -z <tracefile>
```

pprofp Sort Options

- -a: sort by alphabetic names of subroutines.
- -l: sort by number of calls to subroutines
- -m: sort by memory used in a function call.
- -r: sort by real time spent in subroutines.
- -R: sort by real time spent in subroutines (inclusive of child calls).
- -s: sort by system time spent in subroutines.
- -S: sort by system time spent in subroutines (inc child calls).
- -u: sort by user time spent in subroutines.
- -U: sort by user time spent in subroutines (inc child calls).
- -v: sort by average amount of time spent in subroutines.
- -z: sort by user+system time spent in subroutines. (default)

pprofp Display Options

- -c: display real time elapsed alongside call tree.
- -i: suppress reporting for PHP built-in functions
- -O <cnt>: specify max number of subroutines to display. (default 15)
- -t: display compressed call tree.
- -T: display uncompressed call tree.

Function Call Tree

```
# pprof -t /var/log/php4-apd/pprof.2463
```

```
main
require_once
  php_self
require_once (2x)
  session_is_registered
  require_once
require_once
  require_once (3x)
    require_once (2x)
require_once
  require_once
    require_once
      require_once
        is_array
        use_plugin
        file_exists
        include_once
        function_exists
        ... etc
```

Call Tree Timing Display

Useful to see where large time increments exist:

```
# pprof -Tc /var/log/php4-apd/pprof.2463
```



```
jon@svn.ivt.com.au: /home/jon
0.87      dbase_sql->fetch_array
0.87      mysql_fetch_array
0.87      dbase_sql->fetch_array
0.87      mysql_fetch_array
0.87      dbase_sql->fetch_array
0.87      mysql_fetch_array
0.87      dbase_sql->fetch_array
0.87      mysql_fetch_array
0.87      dbase_sql->fetch_array
0.87      mysql_fetch_array
0.87      contact->staff_array
0.87      strlen
0.87      dbase_sql->query
0.87      mysql_query
1.08      mysql_errno
1.08      mysql_error
1.08      dbase_sql->fetch_array
1.08      mysql_fetch_array
1.08      knowledge->article_display
1.08      ereg_replace
1.08      nl2br
1.08      eval
1.08      ob_get_contents
1.08      ob_end_clean
```

APD Function Reference

`apd_set_pprof_trace()`

The most useful APD function as far as profiling is concerned, this dumps a tracefile named “pprof.<pid>” in your `apd.dumpdir`.

The tracefile is a machine-parsable output file that can be processed with the “`pprofp <tracefile>`” command.

APD Function Reference

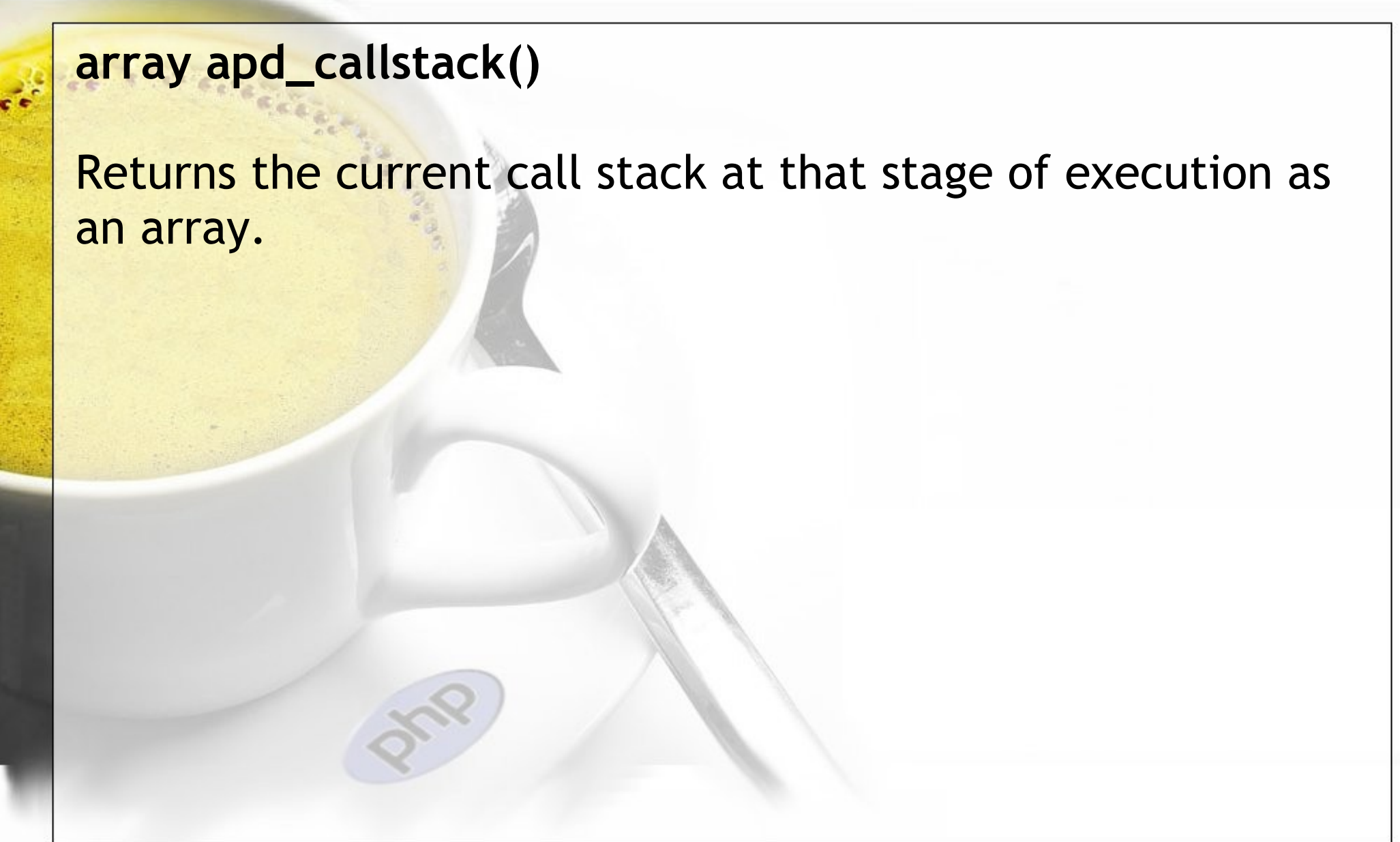
`apd_set_session_trace(N)`

Similar to `apd_set_pprof_trace()`, but it dumps a human-readable session trace named “`apd_dump_<pid>`” in your `apd.dumpdir`. This is the old way of doing things, noted here because it still works (for now). It's been deprecated, so it's better to use a pprof trace instead. `N` is an integer that sets the items to be traced: use a value of 99 to turn on all implemented options.

APD Function Reference

array apd_callstack()

Returns the current call stack at that stage of execution as an array.



APD Function Reference

`apd_cluck([error],[delimiter])`

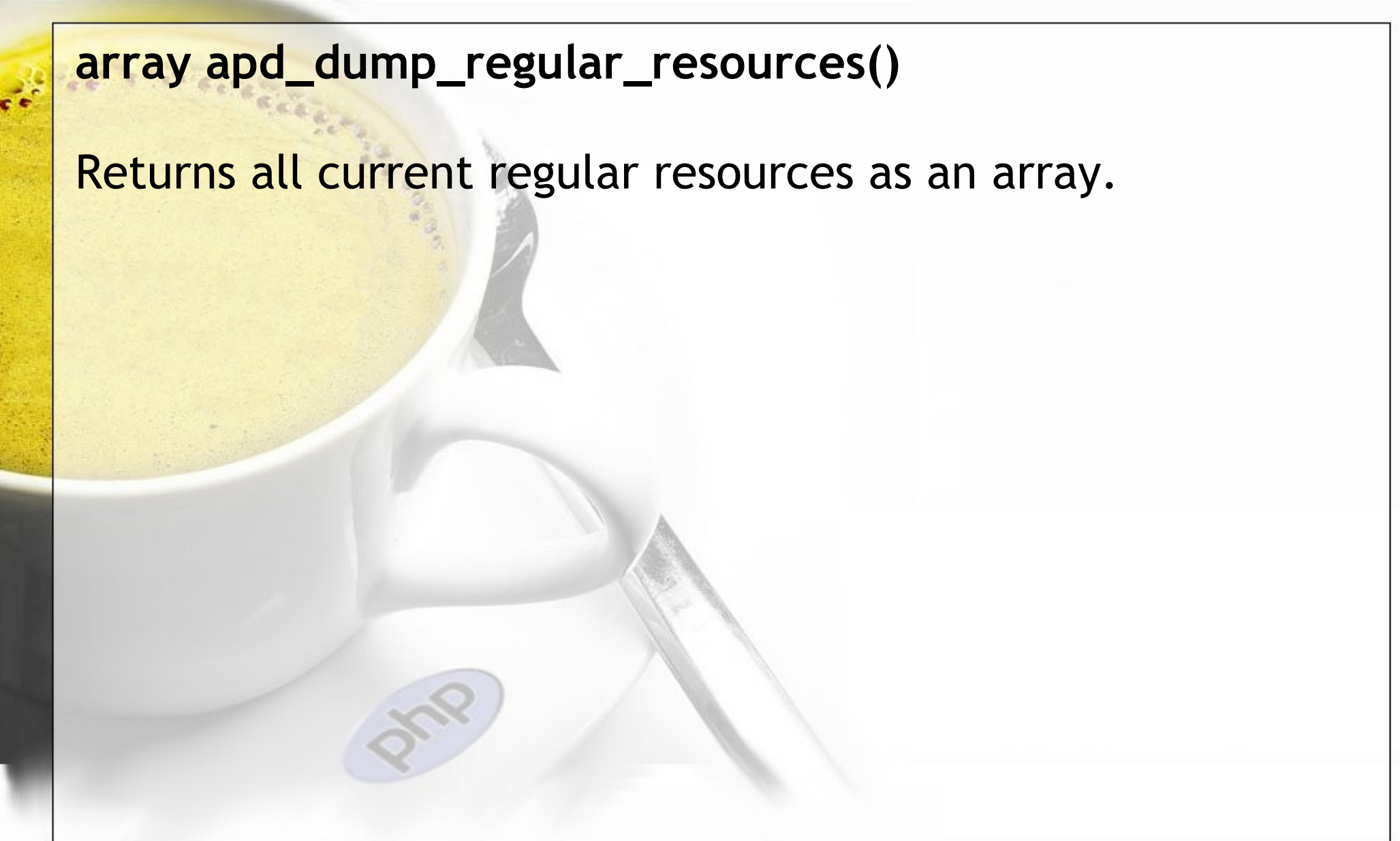
Behaves like Perl's `Carp::croak` module. Throws an error, a callstack and then exits.

The default line delimiter is “`
\n`”. This function is deprecated for users of PHP4.3+: use the internal `debug_backtrace()` and `debug_print_backtrace()` instead.

APD Function Reference

`array apd_dump_regular_resources()`

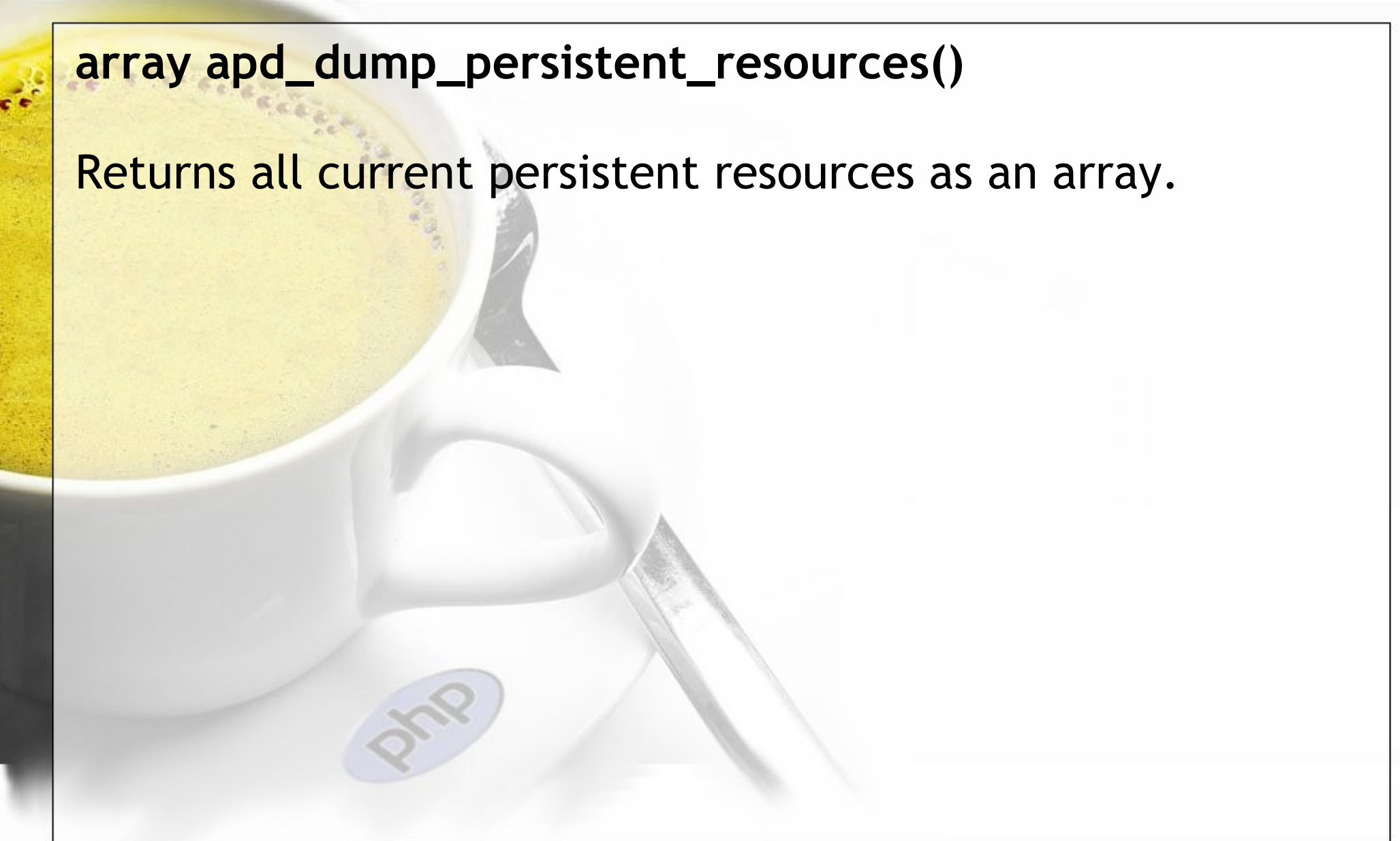
Returns all current regular resources as an array.



APD Function Reference

array apd_dump_persistent_resources()

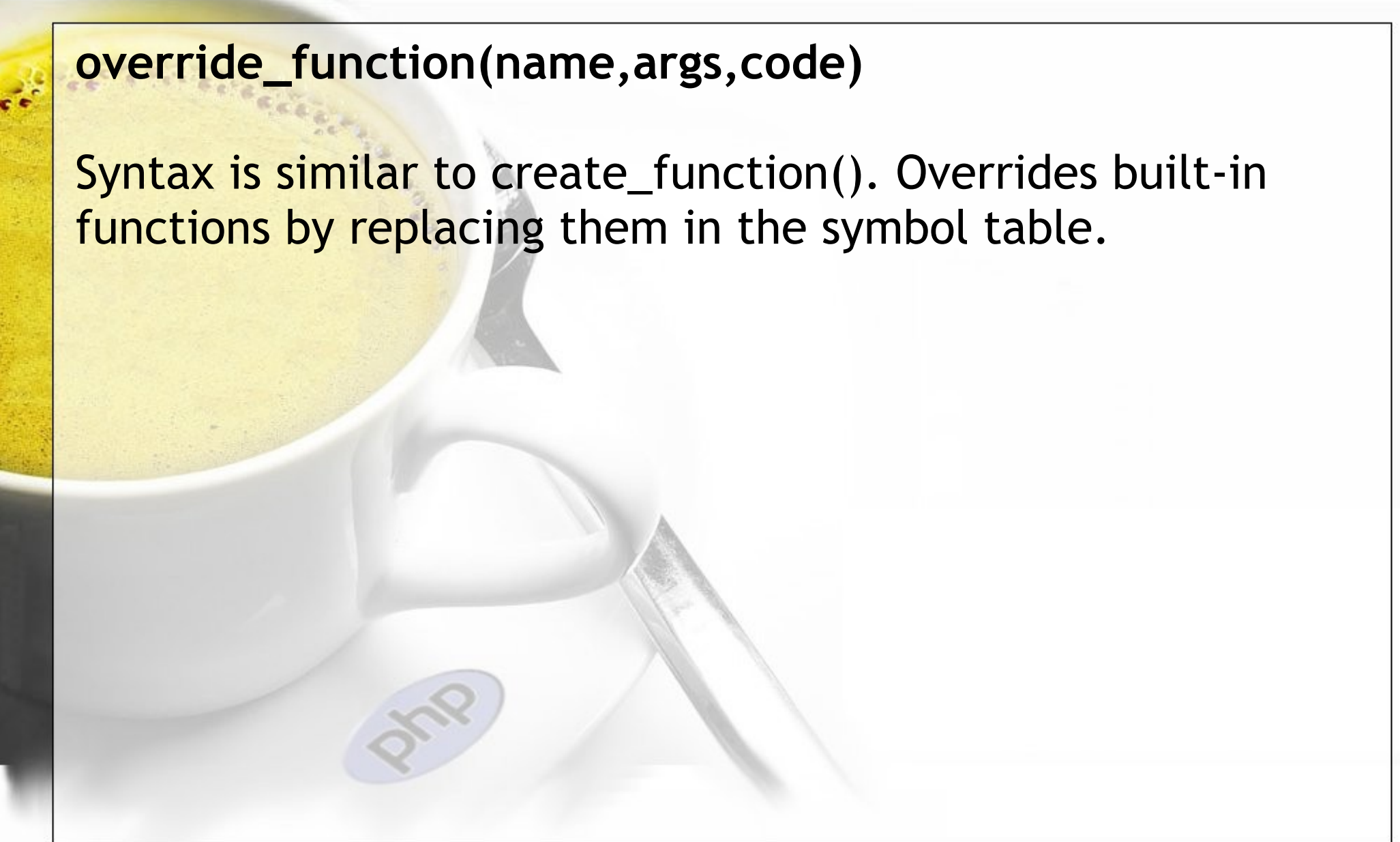
Returns all current persistent resources as an array.



APD Function Reference

`override_function(name, args, code)`

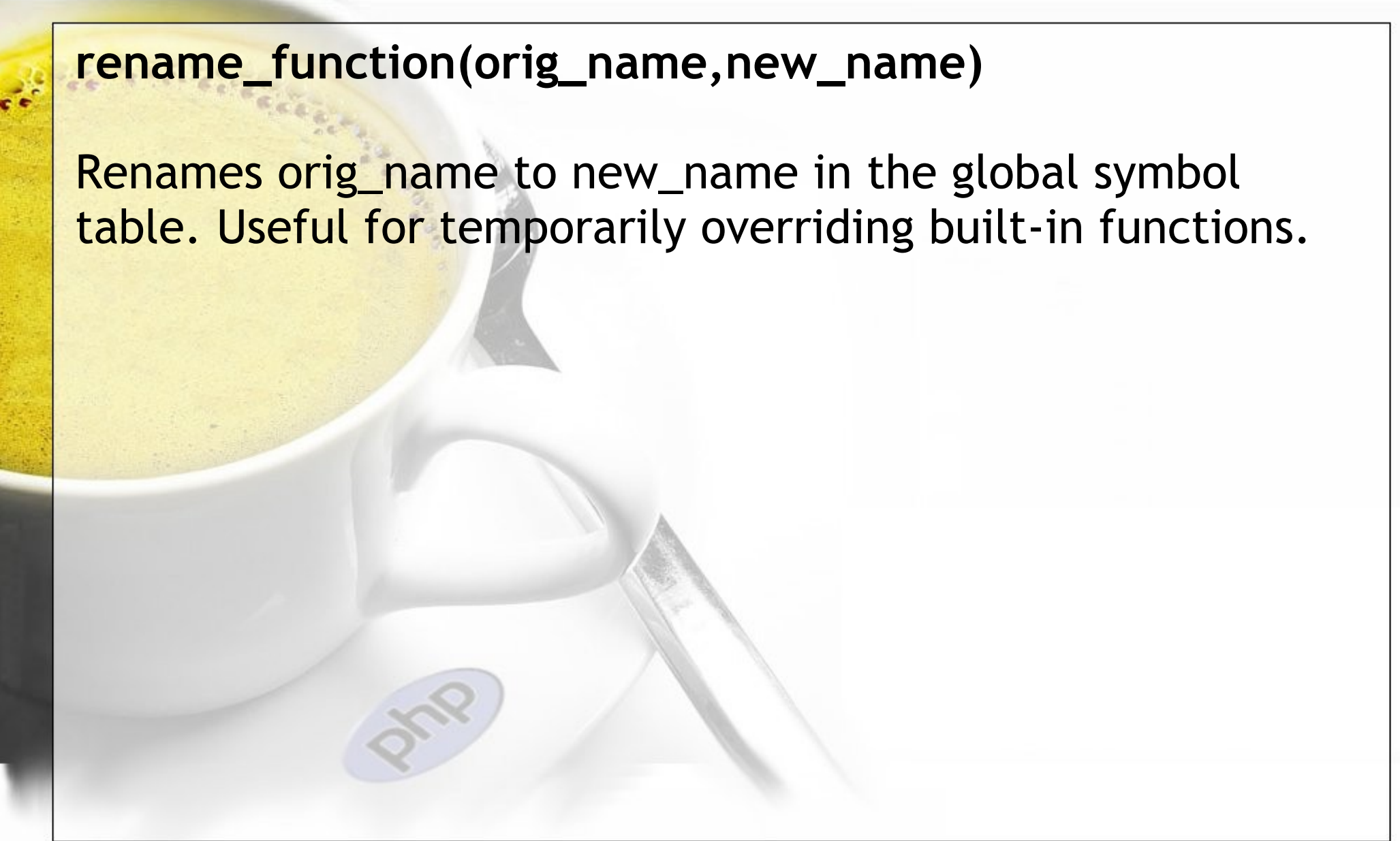
Syntax is similar to `create_function()`. Overrides built-in functions by replacing them in the symbol table.



APD Function Reference

`rename_function(orig_name,new_name)`

Renames `orig_name` to `new_name` in the global symbol table. Useful for temporarily overriding built-in functions.



Profiling A Live Site

Sometimes we don't have the luxury of an isolated test environment with a complete dataset and identical hardware to the production system. But profiling slows down the site and generates a lot of data, so it's a bad idea on production servers.

Solution:

```
<?php
$DEBUGIPS = array('202.91.207.51', '192.168.0.23');
if(array_search($_SERVER[REMOTE_IP], $DEBUGIPS))
    apd_set_pprof_trace();
?>
```

Editor Integration

Profiling is a pain

1. Set trace
2. Load page
3. Find tracefile
4. Process with pprof
5. Read output
6. Try to understand it
7. Tweak your code
8. Go to 2.

Editor Integration

If it's a pain, you won't do it.

Andy Jeffriess, author of gPHPEdit (www.gphpedit.org), is looking at integrating APD calls directly into gPHPEdit.

Extreme Funkiness: APD / Kcachegrind

Kcachegrind provides a graphical representation of profiling data.

pprof2calltree processes the pprof tracefile and generates an inputfile for Kcachegrind.

Getting pprof2calltree: .deb



```
apt-get install \  
kcachegrind \  
kcachegrind-converter
```

Converting A Tracefile

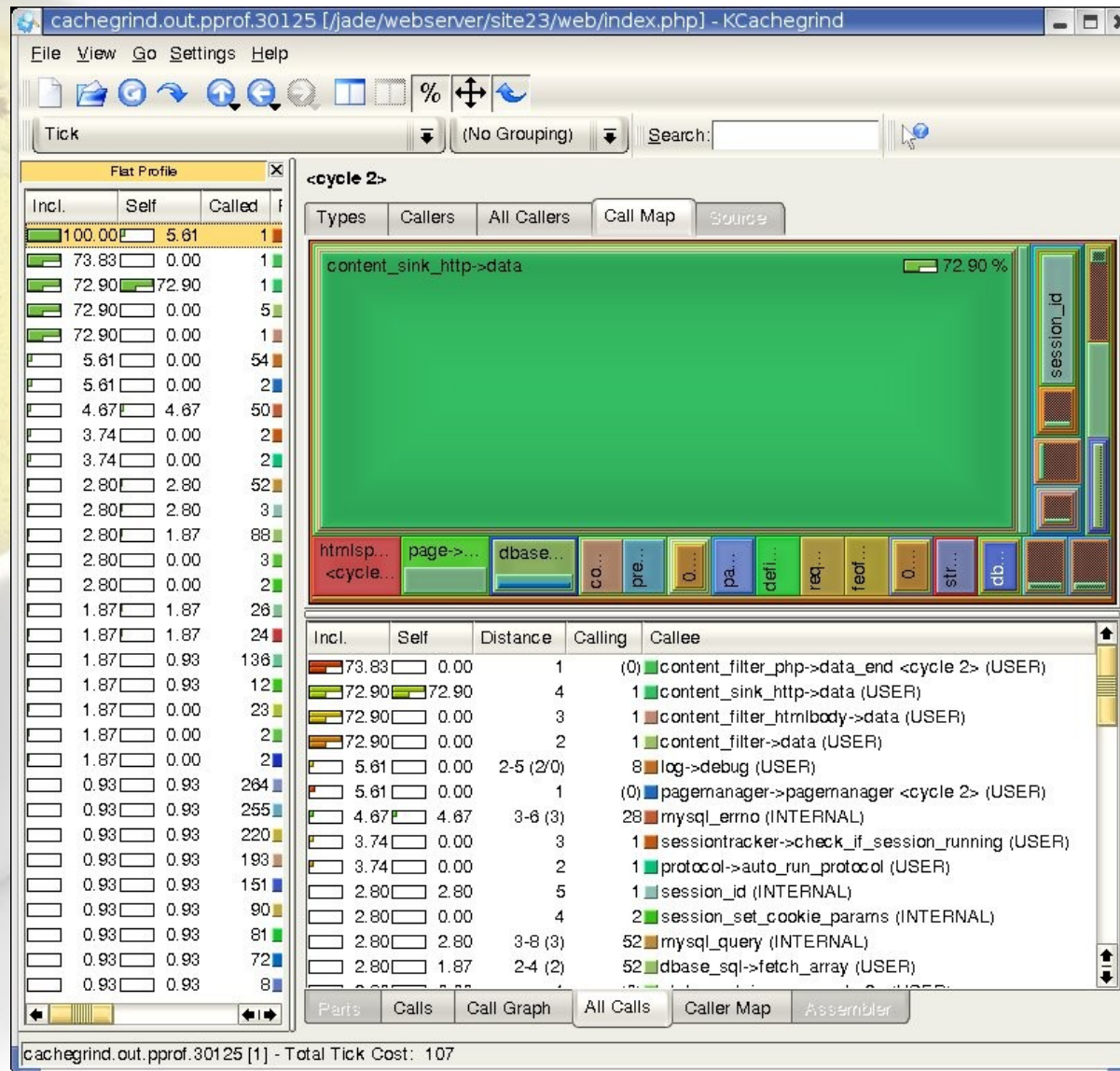
Convert the tracefile to a call tree:

```
pprof2calltree -f /var/log/php4-apd/pprofp.5232
```

Then feed it to Kcachegrind:

```
kcachegrind cachegrind.out.pprof.5232
```

Kcachegrind Output



More Information



These slides are online at:
jon.oxer.com.au/talks

Introductory article in Linux Journal:
www.linuxjournal.com/article.php?sid=7213

Xdebug documentation:
www.xdebug.org/docs-profiling.php