



PHP Performance Profiling

Jonathan Oxe
LinuxTag 2004, Karlsruhe

Do you know what your code really does?

Are you sure?

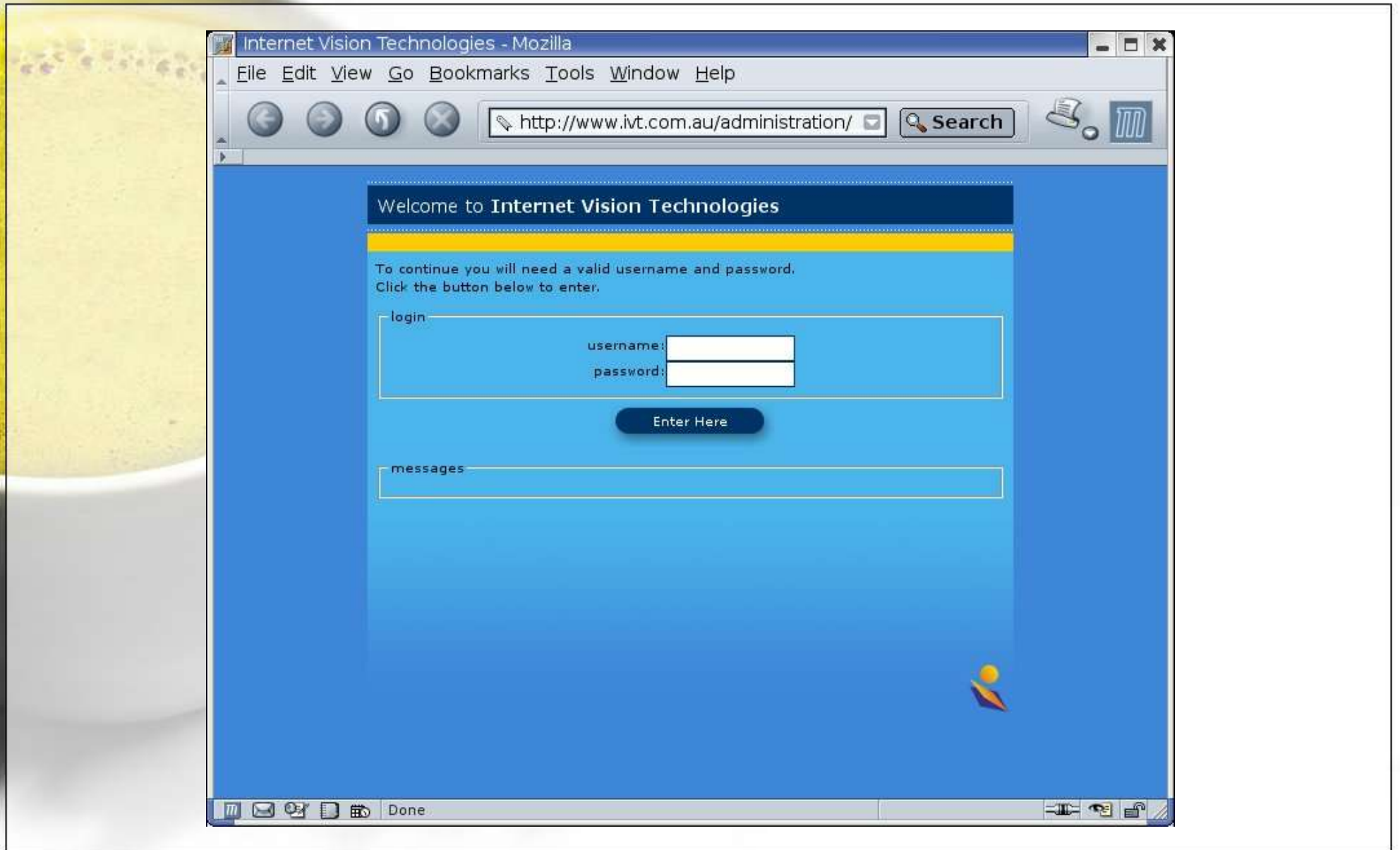
Much of our code is opaque, even to us. OOP theory teaches us to treat objects as “black boxes”, and libraries are often used the same way. We usually have no idea of the true cost of using certain functionality.

Profiling: Objective Analysis

Since we don't know what really happens to our code we need to measure it.

“Profiling” means running our code in a controlled environment to see what happens.

First Shock: 273 Function Calls???



Factors Affecting Performance

- Web server configuration
- Database performance
- Data structure
- Application design
- Application implementation

Profiling Tools

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

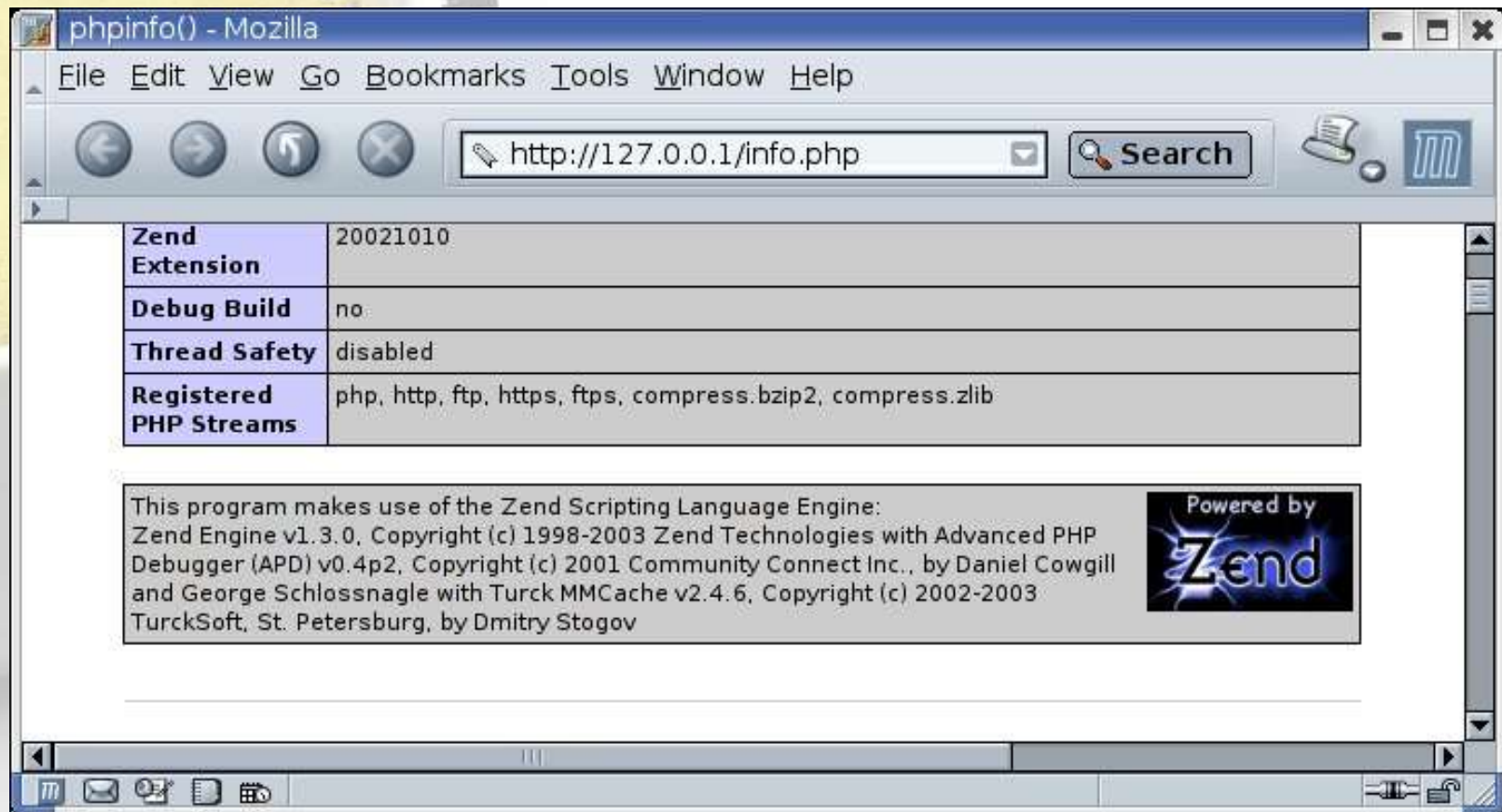
Getting APD

- 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



Your First Test

Find a convenient PHP script for a test. I'll use Squirrelmail.

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

```
apd_set_pprof_trace();
```

Then load the page.

Gathering Some Data

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

Tracefiles are not for human consumption: they are intended to be machine-parsed to generate reports.

“`pprofp <tracefile>`” to process the tracefile.

Intepreting The pprof Tracefile

“pprof” is a script to process the raw tracefile and generate a human-readable report:

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

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

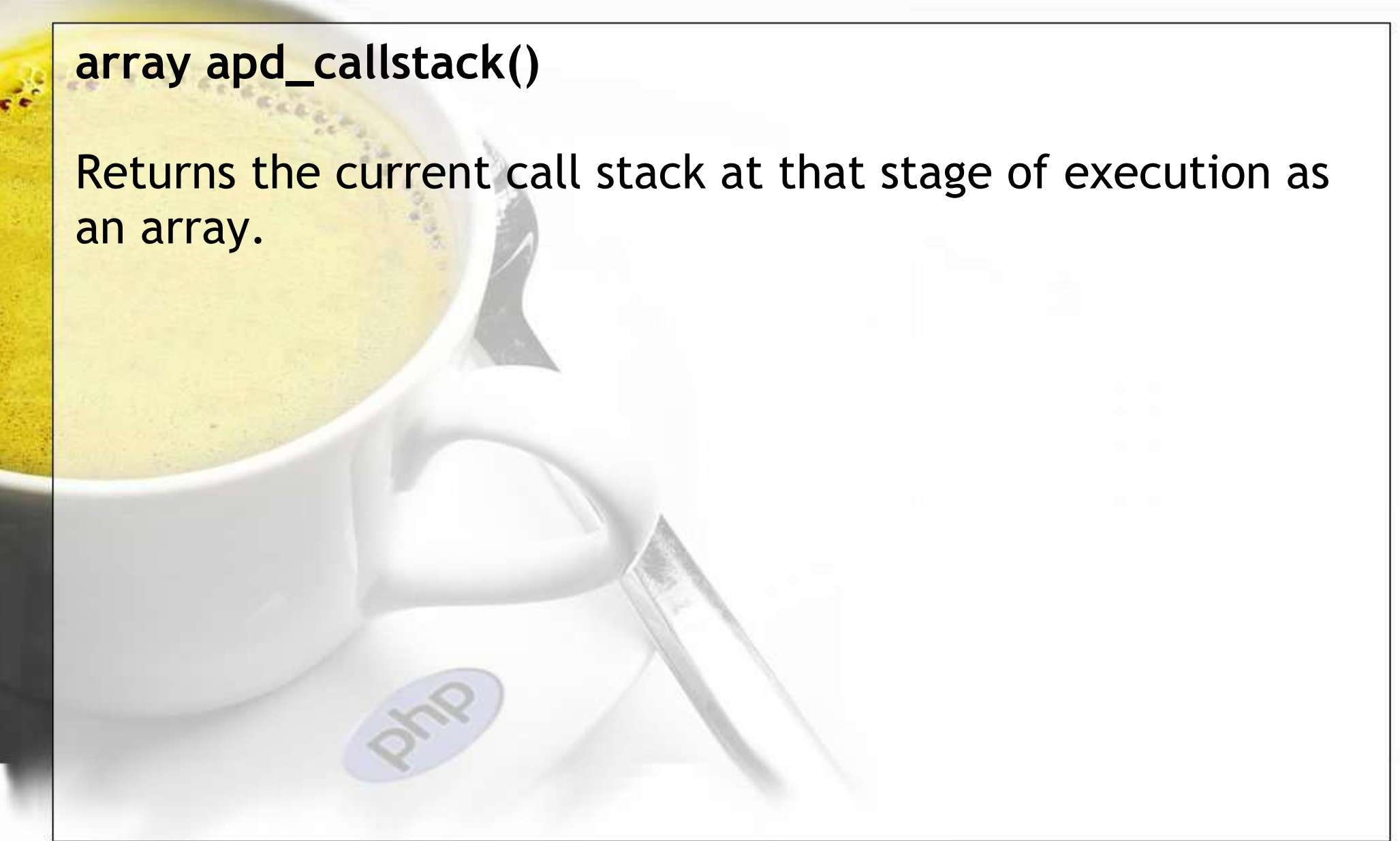
`apd_set_session_trace(N)`

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])`

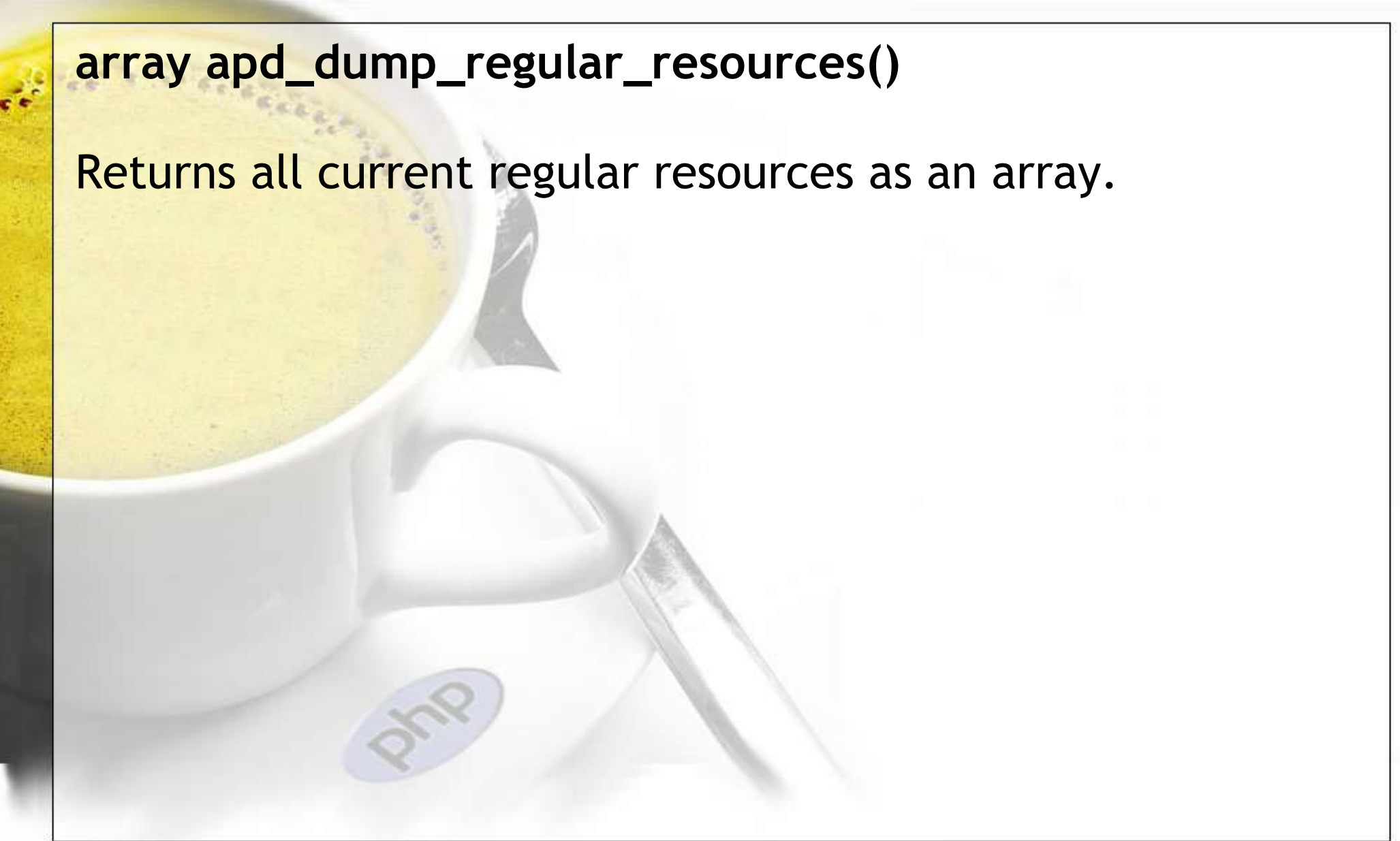
Behaves like Perl's `Carp::croak` module. Throws an error, a callstack and then exits.

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()

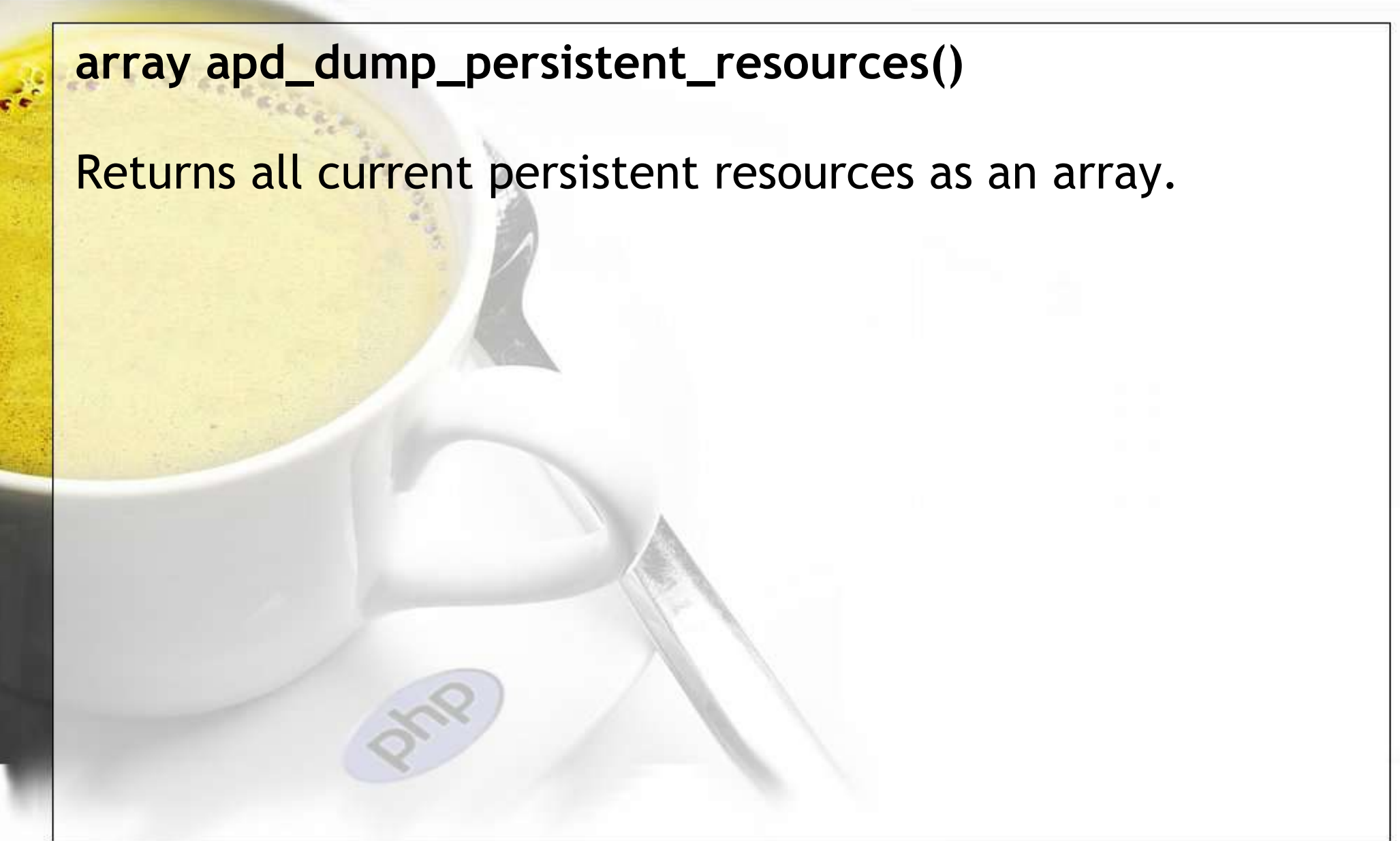
Returns all current regular resources as an array.



APD Function Reference

array apd_dump_persistent_resources()

Returns all current persistent resources as an array.



APD Function Reference

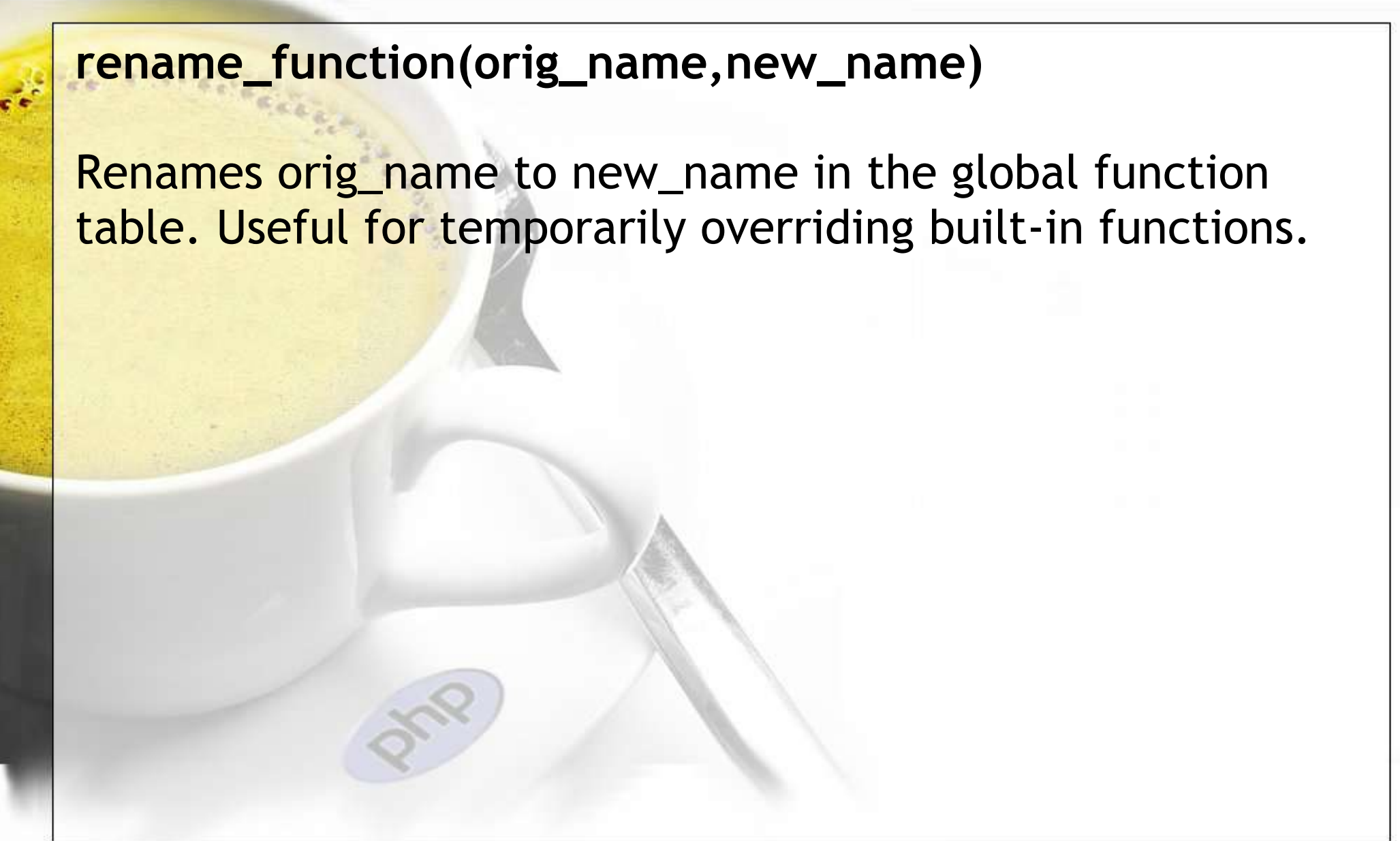
`override_function(name, args, code)`

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 function 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, identical hardware to the production system. But profiling slows down the site and generates a lot of data.

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

Using APD is a pain: set trace, load page, find tracefile, process with pprof, read output, try to understand it, blah blah yawn. Tweak. Do it all over again. Yawn more. Reach for the \$favourite_caffeinated_beverage.

If it's a pain, you won't do it. At least not very often.

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

Extreme Funkiness: APD / Kcachegrind

Kcachegrind provides a graphical representation of profiling data. However, it requires input to be provided in a very specific format.

Solution: pprof2calltree, a script that processes the pprof tracefile and generates an inputfile for Kcachegrind. It's only in the PHP5 version of APD in CVS though.

Converting A Tracefile

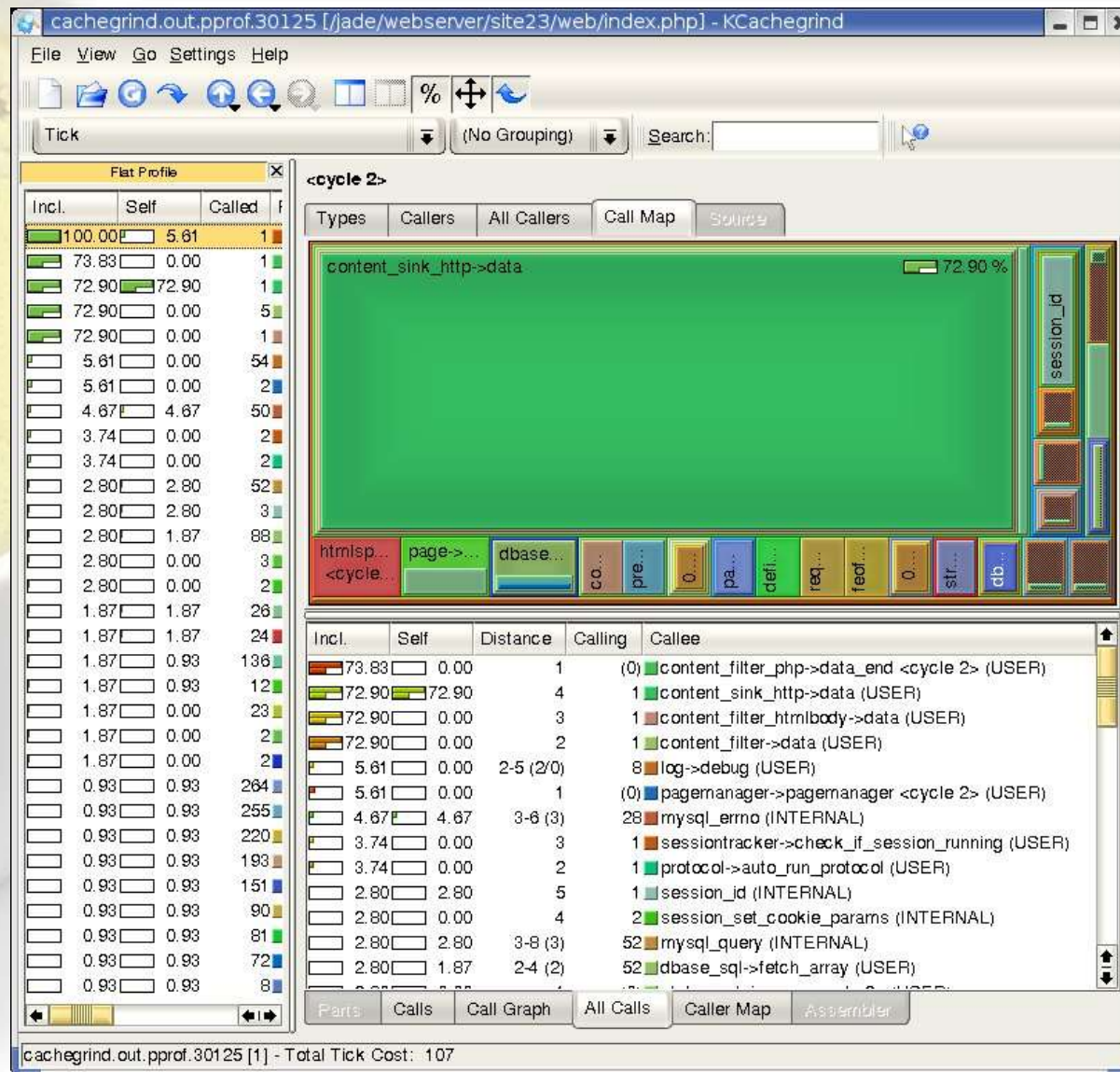
You don't need to re-install all of APD. Just use the pprof2calltree script from the tree you checked out:

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

Now you'll have a new file in a format that Kcachegrind can understand:

```
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