*Write your own **SAPI***

Andrey Hristov
Oracle Corp.

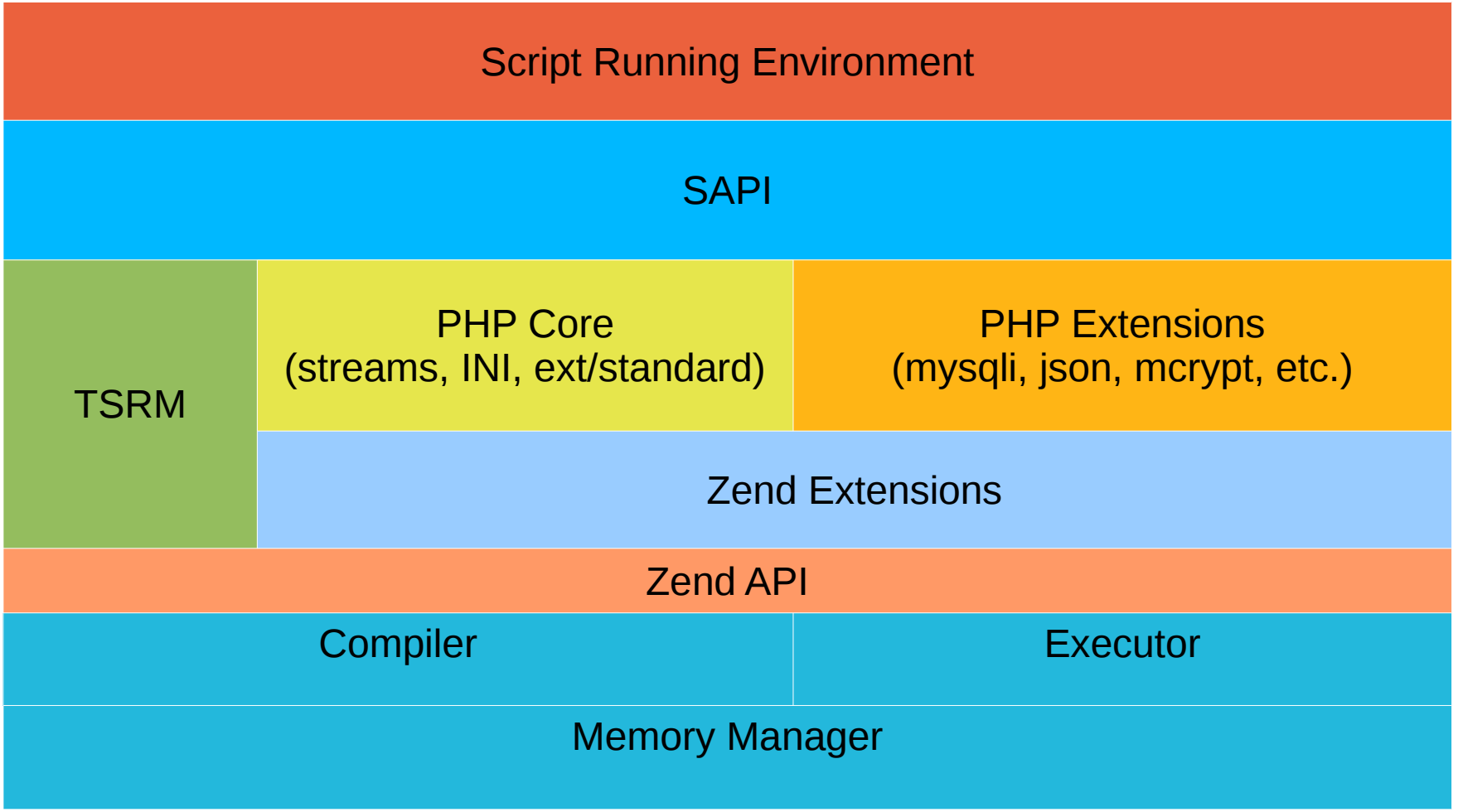
Agenda

- The usual info
- The PHP architecture
- Zend, what are you?
- SAPI, who are you?
- TSRM
- Let's invite PION to the stage
- Special guest this afternoon: PHP request
- Let PHP greet the world!
- Final words

The usual info

- Andrey is a software developer at Oracle, the red company (ex-SUN, ex-ex-MySQL – both blue companies)
- His job is to give you the best MySQL experience in the PHP universe
- (He) started to work with PHP just before 4.0.0 was released, and can be blamed for some PHP core functions and functionality to be found in Syberia (of PHP).
- Disclaimer : “*Past Performance Is No Guarantee of Future Results*” ;)

PHP Architecture



What is the Zend Engine?

- PHP is like Java, compiled to bytecode, ran by a virtual machine
- Zend is not PHP but Zend runs PHP
- Zend parses the scripts and produces bytecode
- The Zend Executor runs the bytecode
- Zend has a hookable API, i.e the parser and the executor can be customized through Zend Extensions (APC, Optimizer, xdebug)

SAPI

high level

- Server API
- Bonding between PHP and the world around
- Another kind of extension
- C API (main/SAPI.h)
- Found under *sapi/*
- (Fast-)CGI, CLI, Apache (1.x and 2), FPM, Embed, Vulcan SRM
- CLI & Embed are definitely not servers :)
- Since 5.4.0 CLI has an embedded web server, good for running the test suite or quickly checking things

ZTS

- Zend Thread Safety mode
- The actual work is done by TSRM
- Written by Sascha Schumann, Zeev and Andi
- Has own license, due to Sascha
- Thin thread abstraction layer
- However, mainly used as thread local storage (TLS). Accessed through a macro like EG, PG, MySG, etc.
- Macro is expanded depending on ZTS setting

TSRM primitives

- Found in TSRM/TSRM.h
- `TSRMLS_D, TSRMLS_DC` (definition; def + comma)
- `TSRMLS_C, TSRMLS_CC` (call; call + comma)
- `TSRMLS_FETCH()` - only in rare cases
- `tsrm_startup()/ tsrm_shutdown()`
- `tsrm_mutex_alloc()/_free()/ _lock()/_unlock()`

```
/* #define TSRMG(id, type, element) (((type)(*((void ***)tsrm_ls))[TSRM_UNSHUFFLE_RSRC_ID(id)])->element)
   #define TSRMLS_D void ***tsrm_ls
    For extensions generated by ext_skeleton */
#ifdef ZTS
#define MYSQLND_G(v) TSRMG(mysqlnd_globals_id, zend_mysqlnd_globals *, v)
#else
#define MYSQLND_G(v) (mysqlnd_globals.v)
#endif
```

PION

“pion-net is not intended to implement yet another web server, but to provide HTTP(S) functionality to new or existing C++ applications. If you're looking for a full-featured server application, check out Apache or lighttpd. If you're working on a Boost C++ application and would just like to use HTTP to provide a simple user interface or interact with run-time data, then pion-net is a clean and simple solution.

Pion Network Library uses the Boost and asio libraries for multi-threading and asynchronous I/O. This allows servers implemented using pion-net to handle many thousands of connections simultaneously with a single physical Server.”

Pion-net debian package description

PION required Boost modules

- boost_date_time
- boost_filesystem
- boost_iostreams
- boost_regex
- boost_signals
- boost_system
- boost_thread

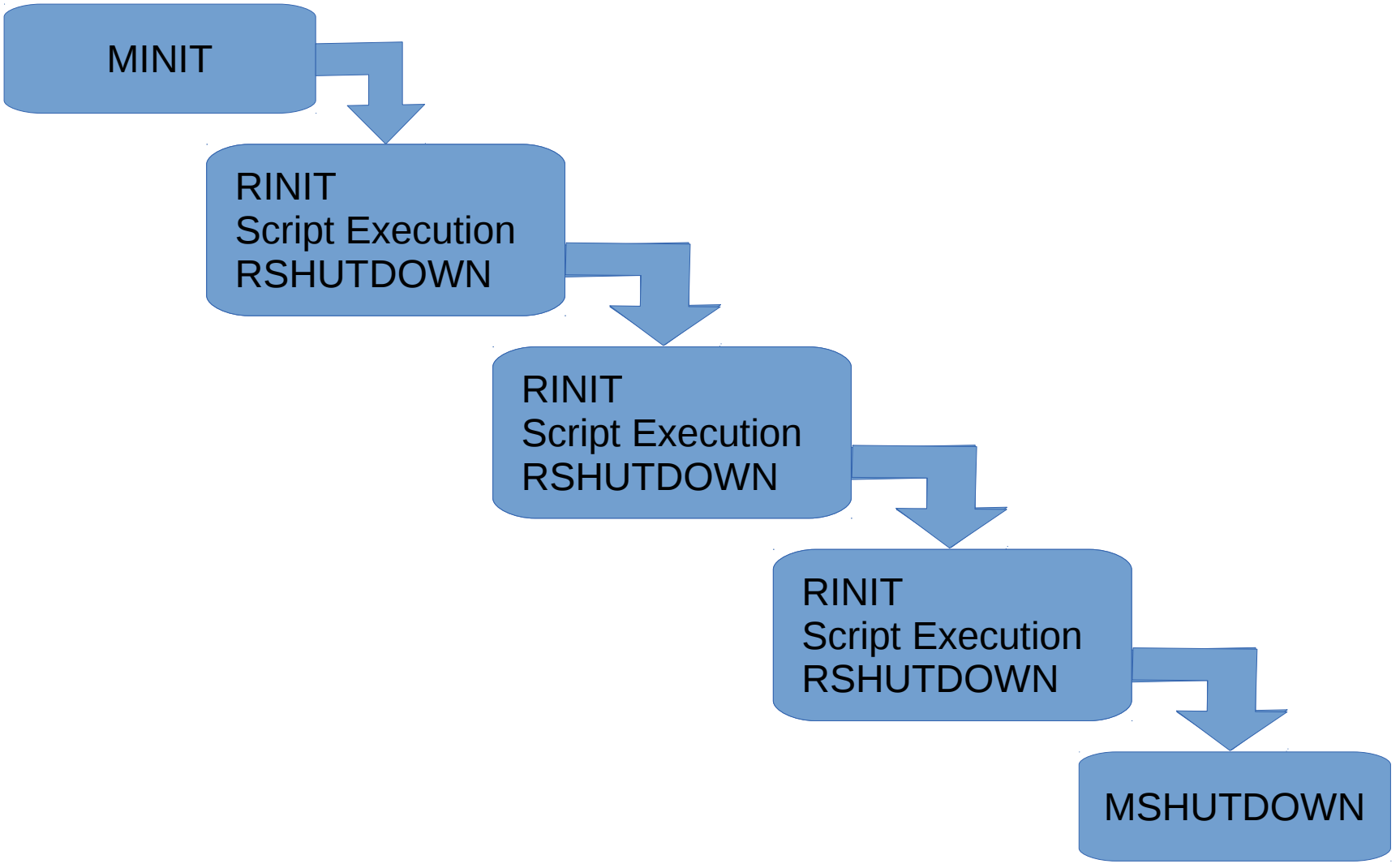
PION is easy!

```
#include <iostream>
#include <pion/net/HTTPServer.hpp>
#include <pion/net/HTTPTypes.hpp>
#include <pion/net/HTTPRequest.hpp>
#include <pion/net/HTTPResponseWriter.hpp>

using namespace std;
using namespace pion;
using namespace pion::net;

int main (int argc, char *argv[])
{
    static const unsigned int DEFAULT_PORT = 8080;
    try {
        HTTPServerPtr simple_server(new HTTPServer(DEFAULT_PORT));
        simple_server->addResource("/", &handleRootURI);
        simple_server->start();
        std::cout << "PION SiS v 1.0 Port: " << DEFAULT_PORT << "\n";
        sleep(1000);
        simple_server->stop();
        simple_server->clear();
    } catch (std::exception& e) {
        std::cerr << "Simple Server failed: " <<  e.what();
    }
    return 0;
}
```

PION is easy!

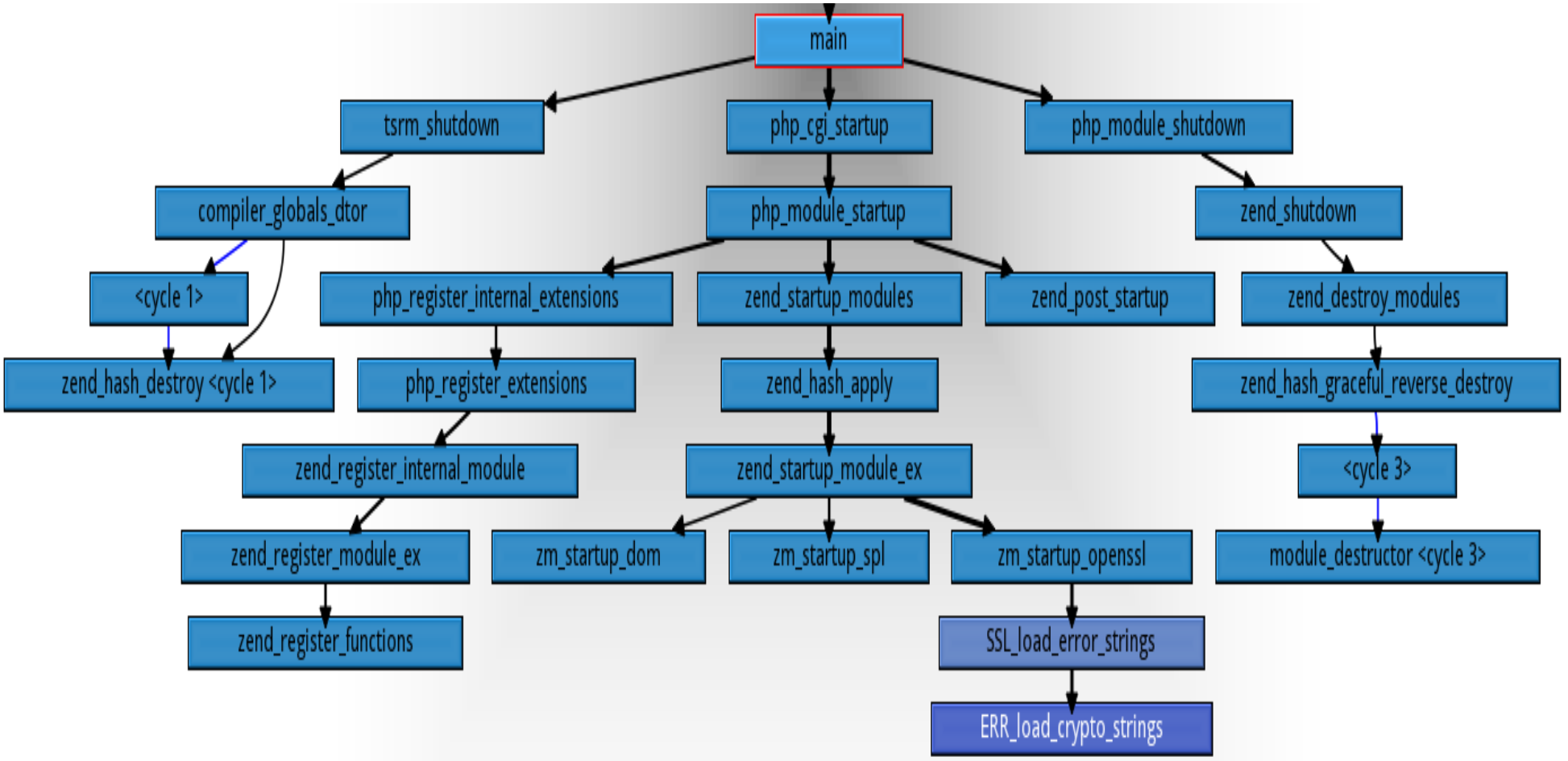
```
void handleRootURI(HTTPRequestPtr& http_request, TCPConnectionPtr& tcp_conn)
{
    HTTPResponseWriterPtr response_writer(
        HTTPResponseWriter::create(tcp_conn, *http_request,
                                   boost::bind(&TCPConnection::finish, tcp_conn));

    HTTPResponse& response = response_writer->getResponse();
    response_writer->write("You requested: ");
    response_writer->write(http_request->getResource());

    HTTPTypes::QueryParams& params = http_request->getQueryParams();
    if (params.size() > 0) {
        response_writer->write("\nAnd passed the following parameters:\n");
        HTTPTypes::QueryParams::const_iterator i;
        for (i = params.begin(); i != params.end(); ++i) {
            response_writer->write(i->first);
            response_writer->write("=");
            response_writer->write(i->second);
            response_writer->write("\n");
        }
    } else {
        response_writer->write("\nAnd passed no parameters.\n");
    }

    response.setStatusCode(HTTPTypes::RESPONSE_CODE_OK);
    response.setStatusMessage(HTTPTypes::RESPONSE_MESSAGE_OK);
    response_writer->send();
}
```

Apache Process Lifetime



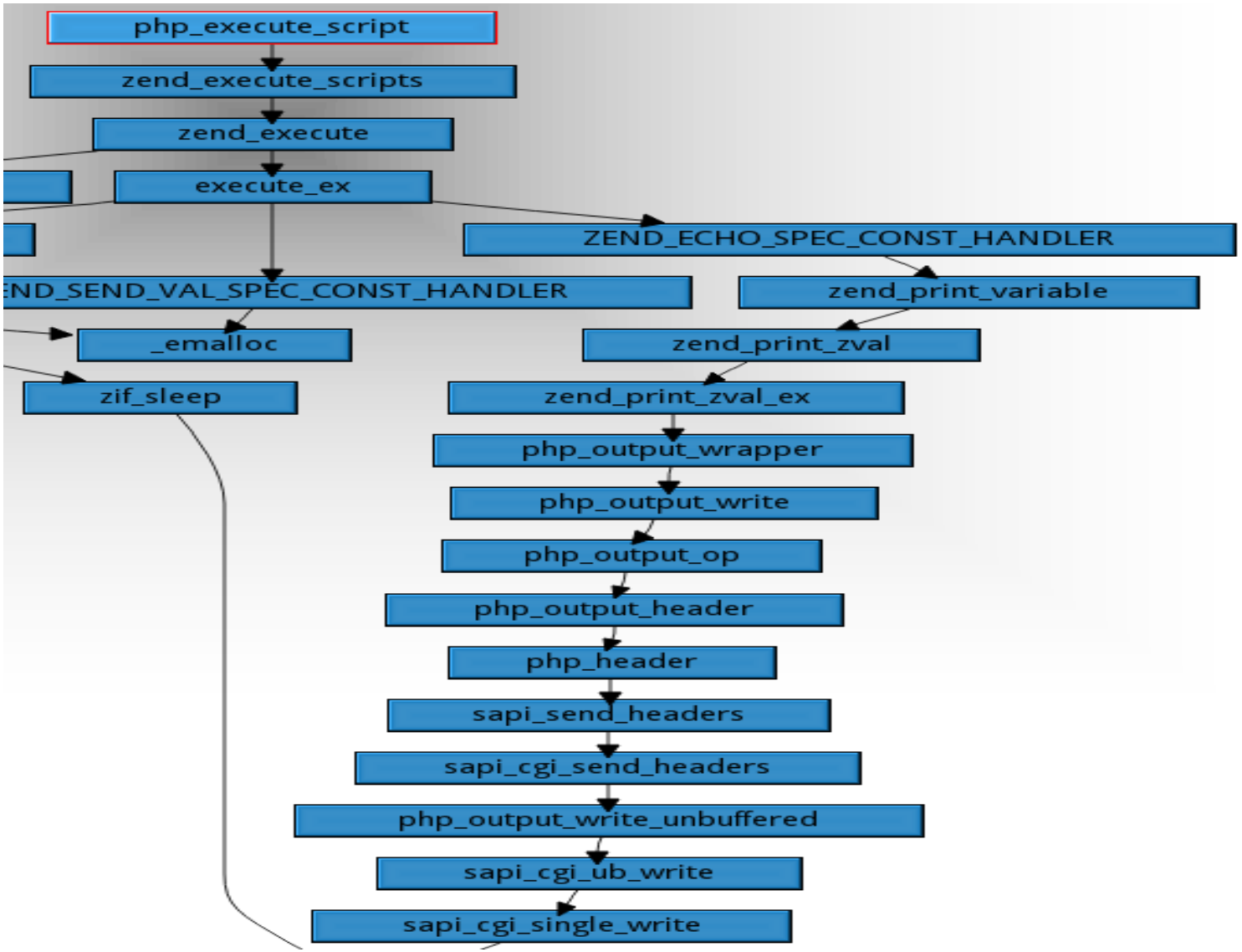
By a popular request - CGI

high level

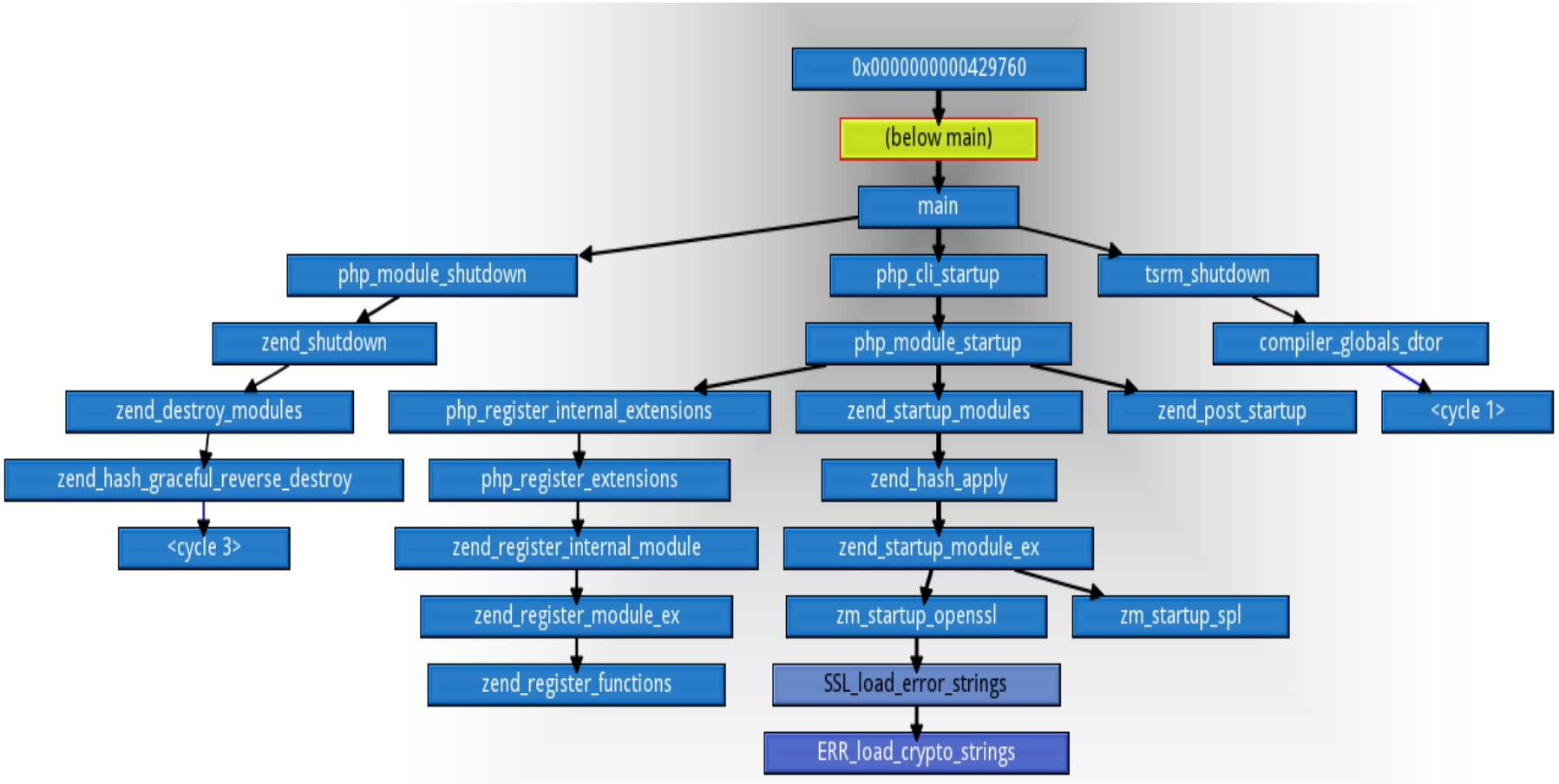


By a popular request - CGI

execute (<?php echo “Hello World”; ?>)



By a popular request - CLI



SAPI

(low level)

```
struct _sapi_module_struct {
  char *name;                   /* php_sapi_name() */
  char pretty_name;             /* phpinfo() */

  int (*startup)( );             /* MINIT              */
  int (*shutdown)( );           /* MSHUTDOWN */

  int (*activate)( );           /* RINIT              */
  int (*deactivate)( );         /* RSHUTDOWN */

  int (*ub_write)( );           /* unbuffered write   */
  void (*flush)( );
  struct stat *(*get_stat)( );  /* script file info   */
  char *(*getenv)( );           /* env variable read */

  void (*sapi_error)( );        /* error handler      */

  int (*header_handler)( );     /* add/update header */
  int (*send_headers)( );       /* either this is NULL  */
  void (*send_header)( );       /* or this is NULL */

  int (*read_post)( );          /* decode POST data */
  char *(*read_cookies)( );     /* import cookies */

  void (*register_server_variables)();/*$_SERVER*/
  void (*log_message)();            /* error logging    */
  double (*get_request_time)( );  /* for Apache       */
  void (*terminate_process)( );   /* used by apache*/
```

```
  char *php_ini_path_override; /* different INI path*/

  void (*block_interruptions)(void); /* for hash & MM*/
  void (*unblock_interruptions)(void);

  void (*default_post_reader)( );/* POST+Filtering */
  void (*treat_data)( );              /* Input Filtering    */
  char *executable_location; /*PHP_BINARY const*/

  int php_ini_ignore;               /* use php.ini or not */
  int php_ini_ignore_cwd;           /* no INI in CWD      */

  int (*get_fd)( );                 /* conn's FD          */
  int (*force_http_10)( );          /* force HTTP/1.0 */

  int (*get_target_uid)( );         /* current user id    */
  int (*get_target_gid)( );         /* current group id */

  unsigned int (*input_filter)( ); /* input filtering     */

  void (*ini_defaults)( );          /* some INI defaults*/
  int phpinfo_as_text;              /* Text or HTML       */

  char *ini_entries;                /* INI entries w/o file*/

  /* for loading PHP_FUNCTIONs */
  const zend_function_entry * additional_functions;
  unsigned int (*input_filter_init)( ); /* IF init */
};
```

SAPI

(low level)

```
static sapi_module_struct cgi_sapi_module= {
  "cgi-fcgi",                    /* name - php_sapi_name() */
  "CGI/FastCGI",                 /* pretty name - phpinfo() SAPI name*/

  php_cgi_startup,               /* startup */
  php_module_shutdown_wrapper,   /* shutdown */

  sapi_cgi_activate,             /* activate */
  sapi_cgi_deactivate,           /* deactivate */
  sapi_cgi_ub_write,             /* unbuffered write*/
  sapi_cgi_flush,                /* flush */
  NULL,                          /* get uid */
  sapi_cgi_getenv,               /* getenv */

  php_error,                     /* error handler*/

  NULL,                          /* header handler */
  sapi_cgi_send_headers,         /* send headers */
  NULL,                          /* send header */
  sapi_cgi_read_post,            /* read POST data */
  sapi_cgi_read_cookies,         /* read Cookies */

  sapi_cgi_register_variables,   /* register server variables */
  sapi_cgi_log_message,          /* Log message */

  NULL,                          /* Get request time */
  NULL,                          /* Child terminate */

  STANDARD_SAPI_MODULE_PROPERTIES
};
```

How to structure?

- PION is C++ but SAPIs are in C, usually, as are the normal extensions. The pass word is PHP_REQUIRE_CXX.
- We need to extern “C” the hooks
- No such tricks if your environment/server is C based.
- Some void * tricks are needed to have C API between the environment and the SAPI.
- Our SAPI should live under sapi/pion in the PHP sources. Built and installed from there.

config.m4

```
PHP_ARG_ENABLE(pion,,
[  --enable-pion[=TYPE] EXPERIMENTAL: PION SAPI TYPE=[shared|static],no, no)
AC_MSG_CHECKING([for PION headers])
if test "$PHP_PION" != "no"; then
  PHP_REQUIRE_CXX

  SEARCH_PATH="/usr/local/include /usr/include"
  AC_MSG_CHECKING([for PION files in default path])
  for i in $SEARCH_PATH ; do
    if test -r $i/pion/net/HTTPServer.hpp; then
      PHP_PION_DIR=$i
    fi
  done
  if test -z "PHP_PION_DIR"; then
     AC_MSG_ERROR([Please install pion-net])
  fi
  PION_INCLUDE_DIR="-I$PHP_PION_DIR/pion -I$PHP_PION_DIR/pion/net"
   case "$PHP_PION" in
    yes|shared)
      PHP_PION_TYPE=shared
      INSTALL_IT="\$(mkinstalldirs) \$(INSTALL_ROOT)\$(prefix)/lib; \$(INSTALL) -m 0755 $SAPI_SHARED \$(INSTALL_ROOT)\$
(prefix)/lib"
      ;;
    static)
      PHP_PION_TYPE=static
      INSTALL_IT="\$(mkinstalldirs) \$(INSTALL_ROOT)\$(prefix)/lib; \$(INSTALL) -m 0644 $SAPI_STATIC \$(INSTALL_ROOT)\$
(prefix)/lib"
      ;;
    *)
      PHP_PION_TYPE=no
      ;;
   esac
   if test "$PHP_PION_TYPE" != "no"; then
     PHP_ADD_LIBRARY_WITH_PATH(stdc++, "", PION_SHARED_LIBADD)
     PHP_SELECT_SAPI(pion, $PHP_PION_TYPE, php_pion.cc, $PION_INCLUDE_DIR)
     PHP_INSTALL_HEADERS([sapi/pion/php_pion.h])
   fi
  AC_MSG_RESULT([$PHP_PION_TYPE])
else
  AC_MSG_RESULT(no)
fi
```

sapi/pion/php_pion.h

```
#ifndef _PHP_PION_H_
#define _PHP_PION_H_

#include <main/SAPI.h>

#ifndef PHP_WIN32
#define PION_SAPI_API SAPI_API
#else
#define PION_SAPI_API
#endif

BEGIN_EXTERN_C()
PION_SAPI_API int php_pion_minit();
PION_SAPI_API void php_pion_mshutdown();
PION_SAPI_API int php_pion_run_script(void * ctx,
                                      const char * const fname);
PION_SAPI_API void * php_pion_create_context(
              void /*HTTPRequestPtr*/ * req,
              void /*HTTPResponseWriterPtr*/ * writer);
PION_SAPI_API void php_pion_free_context(void * ctx);
extern PION_SAPI_API sapi_module_struct php_pion_module;
END_EXTERN_C()

#endif /* _PHP_PION_H_ */
```

Boot Time!

```
static const char HARDCODED_INI[] =
"html_errors=0\nregister_argc_argv=1\nimplicit_flush=1\noutput_buffering=0\n"
"max_execution_time=0\nmax_input_time=-1\n";

PION_SAPI_API int php_pion_minit()
{
#ifdef ZTS
    tsrm_startup(1, 1, 0, NULL);
#endif

#if defined(HAVE_SIGNAL_H) && defined(SIGPIPE) && defined(SIG_IGN)
    signal(SIGPIPE, SIG_IGN); /* ignore SIGPIPE in standalone mode so that sockets created
         via fsockopen() don't kill PHP if the remote site closes it. In apache|apxs mode
         apache does that for us!  thies@thieso.net 20000419 */
#endif
    sapi_startup(&php_pion_module);

#ifdef PHP_WIN32
    _fmode = _O_BINARY;                 /*sets default for file streams to binary */
    setmode(_fileno(stdin), O_BINARY);   /* make the stdio mode be binary */
    setmode(_fileno(stdout), O_BINARY);  /* make the stdio mode be binary */
    setmode(_fileno(stderr), O_BINARY);  /* make the stdio mode be binary */
#endif

    php_pion_module.ini_entries = malloc(sizeof(HARDCODED_INI));
    memcpy(php_pion_module.ini_entries, HARDCODED_INI, sizeof(HARDCODED_INI));

    if (php_pion_module.startup(&php_pion_module) == FAILURE) {
        return FAILURE;
    }
    return SUCCESS;
}
```

Feierabend!

```
PION_SAPI_API void php_pion_mshutdown()
{
    TSRMLS_FETCH();
    php_module_shutdown(TSRMLS_C);
    sapi_shutdown();
#ifdef ZTS
     tsrm_shutdown();
#endif
    if (php_pion_module.ini_entries) {
        free(php_pion_module.ini_entries);
        php_pion_module.ini_entries = NULL;
    }
}
```

RUN PION, RUN!

```
PION_SAPI_API int php_pion_run_script(void* sctx, const char* const filename)
{
    int retval = FAILURE; /* failure by default */
    zend_file_handle file_handle;
    TSRMLS_FETCH();

    /* Set some PHP defaults */
    SG(options) |= SAPI_OPTION_NO_CHDIR;
    SG(headers_sent) = 0;
    SG(server_context) = sctx;

    pion_fill_request_info(SG(server_context), &SG(request_info));

    if (php_request_startup(TSRMLS_C)==FAILURE) {
        return FAILURE;
    }
    php_register_variable("PHP_SELF", "pion", NULL TSRMLS_CC);

    file_handle.type = ZEND_HANDLE_FP;
    file_handle.filename = filename;
    file_handle.opened_path = NULL;
    file_handle.free_filename = 0;
    if ((file_handle.handle.fp = fopen(filename, "rb"))) {
        zend_first_try {
            retval = php_execute_script(&file_handle TSRMLS_CC);
        } zend_end_try();
    }

    /* expects dummy pointer */
    php_request_shutdown((void *) 0);

    return (retval == SUCCESS) ? SUCCESS : FAILURE;
}
```

Cookies are yummy, headers are first

```
static char * php_pion_read_cookies(TSRMLS_D)
{
    psctx_t * sctx = (psctx_t *) SG(server_context);
    HTTPRequestPtr * request = sctx->http_request;
    std::string cook("Cookie", sizeof("Cookie") - 1);
    if ((*request)->hasHeader(cook)) {
        return (char *) (*request)->getHeader(cook).c_str();
    }
    return NULL;
}

static void php_pion_send_header(sapi_header_struct * sapi_header,
                                 void * server_context TSRMLS_DC)
{
  if (sapi_header && server_context) {
    psctx_t * sctx = (psctx_t *) server_context;
    const char * const h_str = sapi_header->header;
    size_t h_len = sapi_header->header_len;
    const char * colon = strchr(h_str, ':');
    if (colon) {
      std::string h_key(h_str, colon - h_str);
      /* skip whitespace */
      do { colon++; } while (h_str[colon - h_str] == ' ');
      std::string h_val(colon, h_len - (colon - h_str));
      (*(sctx->response_writer))->getResponse().changeHeader(h_key,
h_val);
    }
  }
}
```

Write Data & read POST

```
static int
php_pion_ub_write(const char * str, unsigned int str_len TSRMLS_DC)
{
    psctx_t * sctx = (psctx_t *) SG(server_context);
    HTTPResponseWriterPtr * writer = sctx->response_writer;
    (*writer)->write(str, (size_t) str_length);
    return str_length; /* HTTPResponseWriter returns void */
}


static int
php_pion_read_post(char * buffer, unsigned int count_bytes TSRMLS_DC)
{
    psctx_t * ctx = (psctx_t *)SG(server_context);
    HTTPRequestPtr * http_request = ctx->http_request;
    size_t content_len = (*http_request)->getContentLength();
    size_t to_read = (ctx->post_read_offset + count_bytes <=
content_len)?
                        count_bytes:
                        content_len - ctx->post_read_offset;
    if (to_read) {
        const char * content = (*http_request)->getContent();
        memcpy(buffer, content + ctx->post_read_offset, to_read);
        ctx->post_read_offset+= to_read;
    }
    return to_read;
}
```

Extract request info

```
static void
pion_fill_request_info(void * sctx, sapi_request_info * ri)
{
    psctx_t * context = (psctx_t *)sctx;
    HTTPRequestPtr * req = context->http_request;

    ri->request_method = (*req)->getMethod().c_str();
    ri->proto_num =
        (*req)->getVersionMajor()*100 + (*req)->getVersionMinor();
    ri->request_uri = (char*) (*req)->getResource().c_str();

    ri->path_translated = NULL;
    ri->query_string = (char*) (*req)->getQueryString().c_str();
    ri->post_data = (*req)->getContent();
    ri->content_length = (*req)->getContentLength();
    ri->auth_user = NULL;
    if ((*req)->hasHeader("Content-Type")) {
        ri->content_type =
(*req)->getHeader("Content-Type").c_str();
    }
}
```

One liners

```
static int php_pion_deactivate(TSRMLS_D)
{
    return SUCCESS;
}

static void php_pion_flush(void * server_context)
{
    TSRMLS_FETCH();
    if (!SG(headers_sent)) {
        sapi_send_headers(TSRMLS_C);
        SG(headers_sent) = 1;
    }
}

static void php_pion_log_message(char * message TSRMLS_DC)
{
    fprintf(stderr, "%s\n", message);
}

static void php_pion_register_variables(zval* tvarr TSRMLS_DC)
{
    php_import_environment_variables(tvarr TSRMLS_CC);
}

static int php_pion_startup(sapi_module_struct * sapi_module)
{
    return php_module_startup(sapi_module, NULL, 0);
}
```

And now the SAPI struct

```
PION_SAPI_API sapi_module_struct php_pion_module = {
  "pion",                         /* name */
  "Pion",                         /* pretty name */
  php_pion_startup,               /* startup */
  php_module_shutdown_wrapper,    /* shutdown */

  NULL,                           /* activate */
  php_pion_deactivate,            /* deactivate */

  php_pion_ub_write,              /* unbuffered write */
  php_pion_flush,                 /* flush */
  NULL,                           /* get uid */
  NULL,                           /* getenv */

  php_error,                      /* error handler */

  NULL,                           /* header handler */
  NULL,                           /* send headers handler */
  php_pion_send_header,           /* send header handler */

  php_pion_read_post,             /* read POST data */
  php_pion_read_cookies,          /* read Cookies */

  php_pion_register_variables,    /* register server vars */
  php_pion_log_message,           /* Log message */
  NULL,                           /* Get request time */
  NULL,                           /* Child terminate */
  STANDARD_SAPI_MODULE_PROPERTIES
};
```

Let's kick it

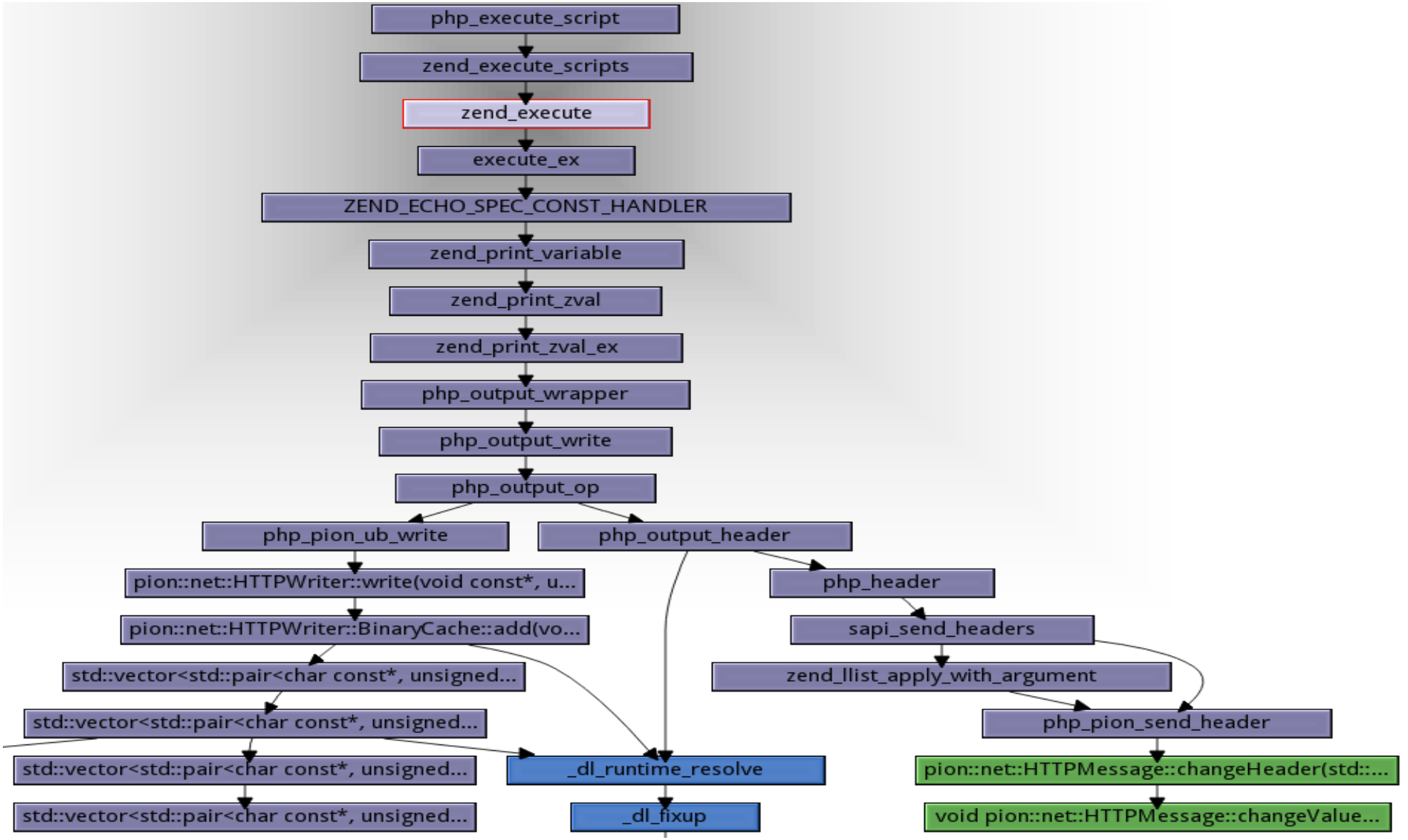
```
void handlePHPURI(HTTPRequestPtr& http_request,
                  HTTPResponseWriterPtr & response_writer)
{
    HTTPResponse& response = response_writer->getResponse();

    std::string filename = http_request->getResource();
    filename.insert(0, PION_DOCUMENT_ROOT);

    /*check yourself or Zend will cry*/
    FILE * fd = fopen(filename.c_str(), "r");
    if (fd) {
        fclose(fd);
        response.setStatusCode(HTTPTypes::RESPONSE_CODE_OK);
        response.setStatusMessage(HTTPTypes::RESPONSE_MESSAGE_OK);
        {
            void * ctx = php_pion_create_context(
                (void *) &http_request, (void *) &response_writer);
            php_pion_run_script(ctx, filename.c_str());
            php_pion_free_context(ctx);
        }
    } else {
        response.setStatusCode(HTTPTypes::RESPONSE_CODE_NOT_FOUND);
        response.setStatusMessage(HTTPTypes::RESPONSE_MESSAGE_FOUND);
    }
    response_writer->send();
}
```

What happens

```
andrey@poohie:/work/dev/pion$ ./hello_service
Hello, the server is running now on port 8080
```

```
andrey@poohie:/work/dev/pion$ cat /tmp/a.php
<?php
echo "Hello PION\n";
?>andrey@poohie:/work/dev/pion$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /a.php HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 11
Content-type: text/html
Connection: close
X-Powered-By: PHP/5.6.0-dev

Hello PION
Connection closed by foreign host.
```

By a popular request – PION



More?

- The source is the best documentation but you might try first the following:
  - “Advanced PHP Programming” by George Schlossnagle, ISBN: 0672325616
  - “Extending and Embedding PHP” by Sara Golemon, ISBN: 067232704X
  - *Wigwam fans:*
    - *http://httpd.apache.org/docs/2.4/developer/modguide.html*
    - *http://www.linuxuser.co.uk/tutorials/develop-apache-modules*
- internals@lists.php.net

¿Questions?

- Shoot now!
  - Or talk to me after this session
    - The shy guys can drop me an email at andrey *ät php.net*