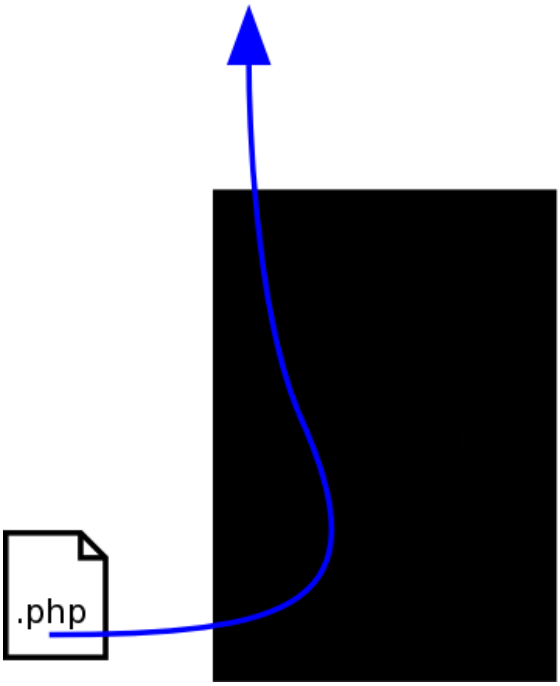
PHP Internals

International PHP Conference 2003

November 5, 2003. Frankfurt, Germany

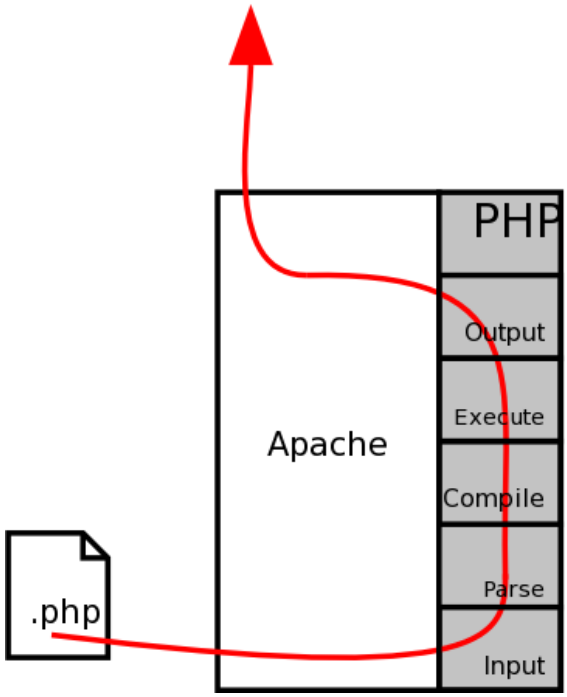
Derick Rethans <derick@php.net>

Introduction



Black box?

Introduction (2)



No! Clearness

Input: Apache module

- PHP registers handlers for:
- Mime types:
- application/x-httpd-php
- application/x-httpd-php-source
- Others:
- text/html (Apache xbithack)
- php-script (Apache 2)
- On a request, Apache:
- checks if there is a handler for the mime type
- opens the file
- fills a meta data record
- hands PHP a filepointer

Input: Apache/CGI

- CGI binary is linked to mime-type:

```
ScriptAlias /php/ /usr/local/bin/php-cgi
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

- On a request, Apache:
- checks if there is a handler for the mime type
- fills in needed environment variables
- executes the CGI processor

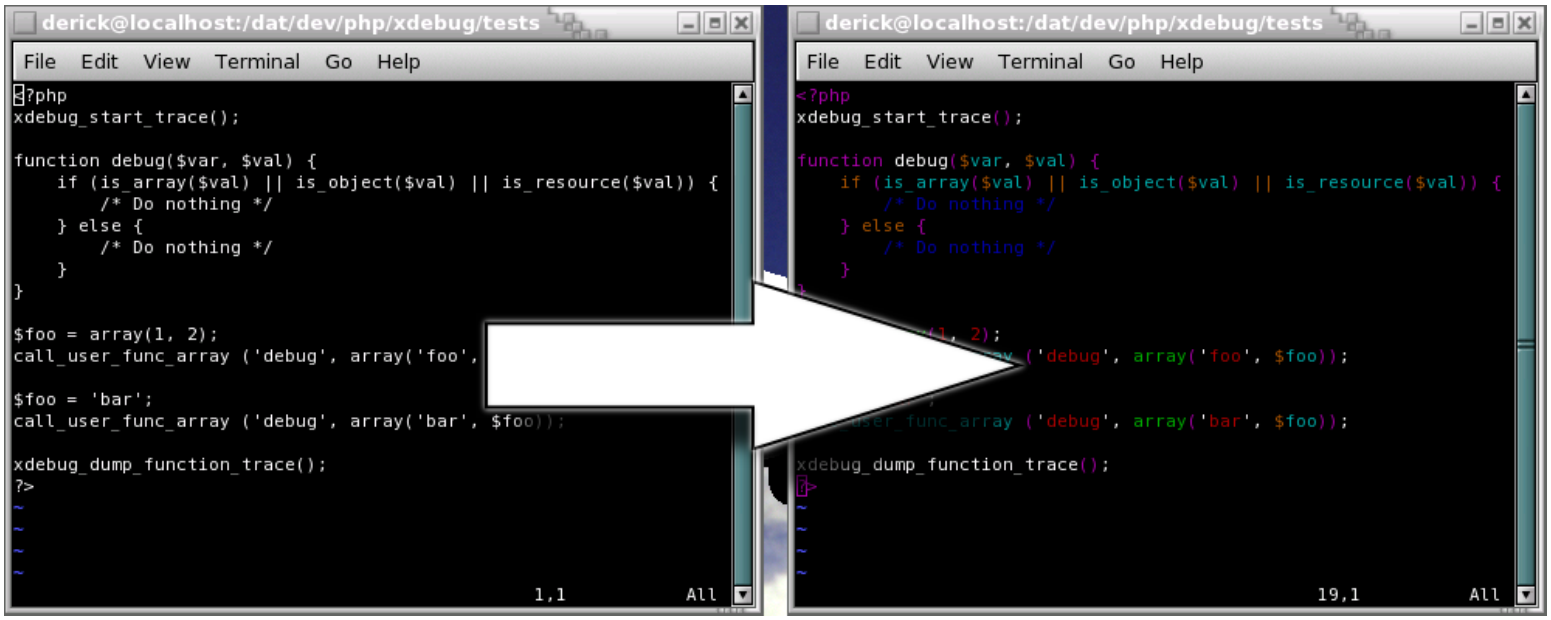
Parsing

- Lexical analyze script source
- Divide into logical blocks of characters
- Give special blocks a meaning
- flex

Our friend Mr Parse Error:

```
Parse error: parse error,
unexpected T_CLASS, expecting ',' or ';' in - on line 2
```

Parsing: Tokenize



Syntax highlighting = Tokenizing

Parsing: Tokens

- The building blocks of a script
- Strings, variables, keywords
- Action rules:

```
<ST_IN_SCRIPTING>"::" {
    return T_PAAMAYIM_NEKUDOTAYIM;
}

<ST_IN_SCRIPTING>{DNUM}|{EXPONENT_DNUM} {
    zendlval->value.dval = strtod(yytext, NULL);
    zendlval->type = IS_DOUBLE;
    return T_DNUMBER;
}
```

Parsing: Tokens in PHP

From Zend/zend_language_parser.y:

```
%left T_INCLUDE T_INCLUDE_ONCE T_EVAL T_REQUIRE T_REQUIRE_ONCE
%left ','
%left T_LOGICAL_OR
%left T_LOGICAL_XOR
%left T_LOGICAL_AND
%right T_PRINT
%left '=' T_PLUS_EQUAL T_MINUS_EQUAL T_MUL_EQUAL T_DIV_EQUAL T_CONCAT_EQUAL
     T_MOD_EQUAL T_AND_EQUAL T_OR_EQUAL T_XOR_EQUAL T_SL_EQUAL T_SR_EQUAL
%left '?' ':'
%left T_BOOLEAN_OR
%left T_BOOLEAN_AND
%left '|'
%left '^'
%left '&'
%nonassoc T_IS_EQUAL T_IS_NOT_EQUAL T_IS_IDENTICAL T_IS_NOT_IDENTICAL
%nonassoc '<' T_IS_SMALLER_OR_EQUAL '>' T_IS_GREATER_OR_EQUAL
%left T_SL T_SR
%left '+' '-' '.'
left '*' '/' ''
%right '!' '~' T_INC T_DEC T_INT_CAST T_DOUBLE_CAST T_STRING_CAST T_ARRAY_CAST
    T_OBJECT_CAST T_BOOL_CAST T_UNSET_CAST '@'
%right '['
%nonassoc T_NEW T_INSTANCEOF
%token T_EXIT
%token T_IF
%left T_ELSEIF
%left T_ELSE
%left T_ENDIF
%token T_LNUMBER
%token T_DNUMBER
%token T_STRING
%token T_STRING_VARNAME
%token T_VARIABLE
%token T_NUM_STRING
%token T_INLINE_HTML
%token T_CHARACTER
%token T_BAD_CHARACTER
%token T_ENCAPSED_AND_WHITESPACE
%token T_CONSTANT_ENCAPSED_STRING
%token T_ECHO
%token T_DO
%token T_WHILE
%token T_ENDWHILE
%token T_FOR
%token T_ENDFOR
%token T_FOREACH
%token T_ENDFOREACH
%token T_DECLARE
%token T_ENDDECLARE
%token T_INSTANCEOF
%token T_AS
%token T_SWITCH
%token T_ENDSWITCH
%token T_CASE
%token T_DEFAULT
%token T_BREAK
%token T_CONTINUE
%token T_FUNCTION
%token T_CONST
%token T_RETURN
%token T_TRY
%token T_CATCH
%token T_THROW
%token T_USE
%token T_GLOBAL
%right T_STATIC T_ABSTRACT T_FINAL T_PRIVATE T_PROTECTED T_PUBLIC
%token T_VAR
%token T_UNSET
%token T_ISSET
```

```
%token T_EMPTY
%token T_CLASS
%token T_INTERFACE
%token T_EXTENDS
%token T_IMPLEMENTS
%token T_OBJECT_OPERATOR
%token T_DOUBLE_ARROW
%token T_LIST
%token T_ARRAY
%token T_CLASS_C
%token T_FUNC_C
%token T_LINE
%token T_FILE
%token T_COMMENT
%token T_DOC_COMMENT
%token T_OPEN_TAG
%token T_OPEN_TAG_WITH_ECHO
%token T_CLOSE_TAG
%token T_WHITESPACE
%token T_START_HEREDOC
%token T_END_HEREDOC
%token T_DOLLAR_OPEN_CURLY_BRACES
%token T_CURLY_OPEN
%token T_PAAMAYIM_NEKUDOTAYIM
}
```

Parsing: Example (1)

Tokenizer extension:

magic.php:

```
<?php
    function apply_magic(&$a) {
        $a = $a + rand(0,3);
    }
? >
```

tokenize.php:

```
<?php
    foreach (token_get_all($script) as $token) {
        if (count($token) == 2) {
            printf ("-25s [s]\n", token_name($token[0]), $token[1]);
        } else {
            printf ("-25s [s]\n", "", $token[0]);
        }
    }
? >
```

Parsing: Example (2)

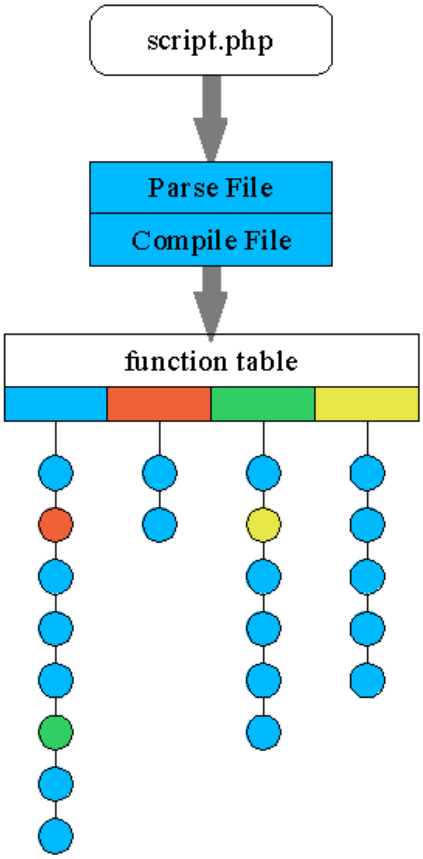
```
<?php
    function apply_magic(&$a) {
        $a = $a + rand(0,3);
    }
? >
```

```
T_OPEN_TAG                [<?php\n]
T_WHITESPACE              [    ]
T_FUNCTION                [function]
T_WHITESPACE              [ ]
T_STRING                  [apply_magic]
                          [(]
                          [&]
T_VARIABLE                [$a]
                          [)]
T_WHITESPACE              [ ]
                          [{]
T_WHITESPACE              [\n        ]
T_VARIABLE                [$a]
T_WHITESPACE              [ ]
                          [=]
T_WHITESPACE              [ ]
T_VARIABLE                [$a]
T_WHITESPACE              [ ]
                          [+]
T_WHITESPACE              [ ]
T_STRING                  [rand]
                          [(]
T_LNUMBER                 [0]
                          [,]
T_LNUMBER                 [3]
                          [)]
                          [;]
T_WHITESPACE              [\n    ]
                          [}]
T_WHITESPACE              [\n]
T_CLOSE_TAG               [? >]
```

Compiling

- Interpreting tokens
- Generating execution units
- Generating class and function tables

Compiling: Diagram



Compiling: Oparrays

- Oparrays
- Compiled code
- One for every script element

- Opcode
- Basic execution unit
- Two operands
- One result

Compiling: Example

fibonacci.php:

```
<?php
    $cache = array();

    function fibonacci($nr) {
        global $cache;

        if (isset($cache[$nr])) {
            return $cache[$nr];
        }
        switch ($nr) {
            case 0:
                die("Invalid Nr\n");
            case 1:
                return 1;
            case 2:
                return 1;
            default:
                $r = fibonacci($nr - 2) + fibonacci($nr - 1);
                $cache[$nr] = $r;
                return $r;
        }
    }

    echo fibonacci($argv[1])."\n";
? >
```

Compiling: Break-down

```
       compile.php
name:  (null)
 ops:  11
#  op                        operands           code
-------------------------------------------------------------------------------
0  INIT_ARRAY                ~1, ,                    $cache = array();
1  FETCH_W                   $0, 'cache'
2  ASSIGN                    $0, ~1
3  NOP                                                function fibonacci($nr) {
4  FETCH_R                   $3, 'argv'               echo fibonacci($argv[1])."\n";
5  FETCH_DIM_R               $4, $3, 1
6  SEND_VAR                  $4
7  DO_FCALL                  $5, 'fibonacci', 0
8  CONCAT                    ~6, $5, '%0A'
9  ECHO                      ~6
0  RETURN                    1

fibonacci:
       compile.php
name:  fibonacci
 ops:  58
#  op                        operands           code
-------------------------------------------------------------------------------
0  FETCH_W                   $0, 'nr'                 function fibonacci($nr) {
1  RECV                      $0, 1
2  FETCH_W                   $1, 'cache'                  global $cache;
3  FETCH_W                   $2, 'cache'
4  ASSIGN_REF                $2, $1
5  FETCH_R                   $4, 'nr'                     if (isset($cache[$nr])) {
6  FETCH_IS                  $3, 'cache',
7  FETCH_DIM_IS              $5, $3, $4
8  ISSET_ISEMPTY             ~6, $5,
9  JMPZ                      ~6, ->15
0  FETCH_R                   $8, 'nr'                         return $cache[$nr];
1  FETCH_R                   $7, 'cache'
2  FETCH_DIM_R               $9, $7, $8
3  RETURN                    $9
4  JMP                       ->15                         }
5  FETCH_R                   $10, 'nr'                    switch ($nr) {
6  BOOL                      ~11,
7  CASE                      ~11, $10, 0                      case 0:
8  JMPZ                      ~11, ->21
9  EXIT                      'Invalid+Nr%0A',                     die("Invalid Nr\n");
0  JMP                       ->23
1  CASE                      ~11, $10, 1                      case 1:
2  JMPZ                      ~11, ->26
3  SWITCH_FREE               $10,                             return 1;
4  RETURN                    1
5  JMP                       ->28
6  CASE                      ~11, $10, 2                      case 2:
7  JMPZ                      ~11, ->31
8  SWITCH_FREE               $10,                             return 1;
9  RETURN                    1
0  JMP                       ->33                         default:
1  JMP                       ->55
2  BOOL                      ~11,
3  INIT_FCALL_BY_NAME        'fibonacci'                      $r = fibonacci($nr - 2) + fibonacci($nr
4  FETCH_R                   $13, 'nr'
5  SUB                       ~14, $13, 2
6  SEND_VAL                  ~14
7  DO_FCALL_BY_NAME          $15, 'fibonacci', 0
8  INIT_FCALL_BY_NAME        'fibonacci'
9  FETCH_R                   $16, 'nr'
0  SUB                       ~17, $16, 1
1  SEND_VAL                  ~17
2  DO_FCALL_BY_NAME          $18, 'fibonacci', 0
3  ADD                       ~19, $15, $18
4  FETCH_W                   $12, 'r'
5  ASSIGN                    $12, ~19
6  FETCH_R                   $22, 'nr'                        $cache[$nr] = $r;
7  FETCH_R                   $24, 'r'
```

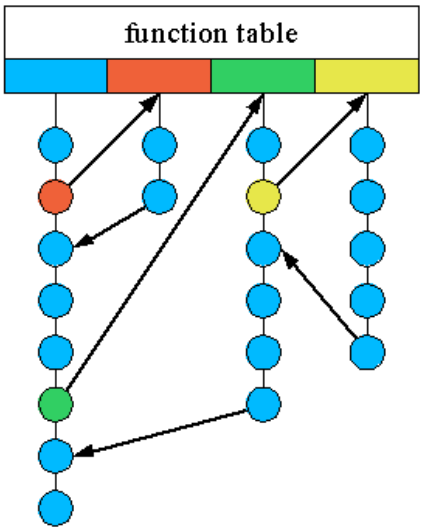
Compiling: Disassembler

- Disassembler: vld
- Dumps oparray per element:
- main script
- function
- class method

Executing

- Executor executes opcodes
- 116 or 140 different opcodes
- Per file execution

Executing: Diagram



Executing: Example (1)

test.inc:

```
<?php
    function foo () {
        echo "bar\n";
    }
    echo "inc\n";
? >
```

test.php

```
<?php
    echo "foo\n";
    include "test.inc";
    foo();
    echo "more bar\n";
? >
```

Executing: Example (2)

The engine:

- parses and compiles test.php
- starts executing test.php
- parses and compiles test.inc when the include() statement is encountered
- starts executing test.inc
- resumes executing test.php

Questions

?

Resources

These Slides: http://pres.derickrethans.nl

VLD site: http://www.derickrethans.nl/vld.php

PHP: http://www.php.net

Index