PHP быстрый или медленный ?

Dmitry Stogov (dmitry@zend.com)

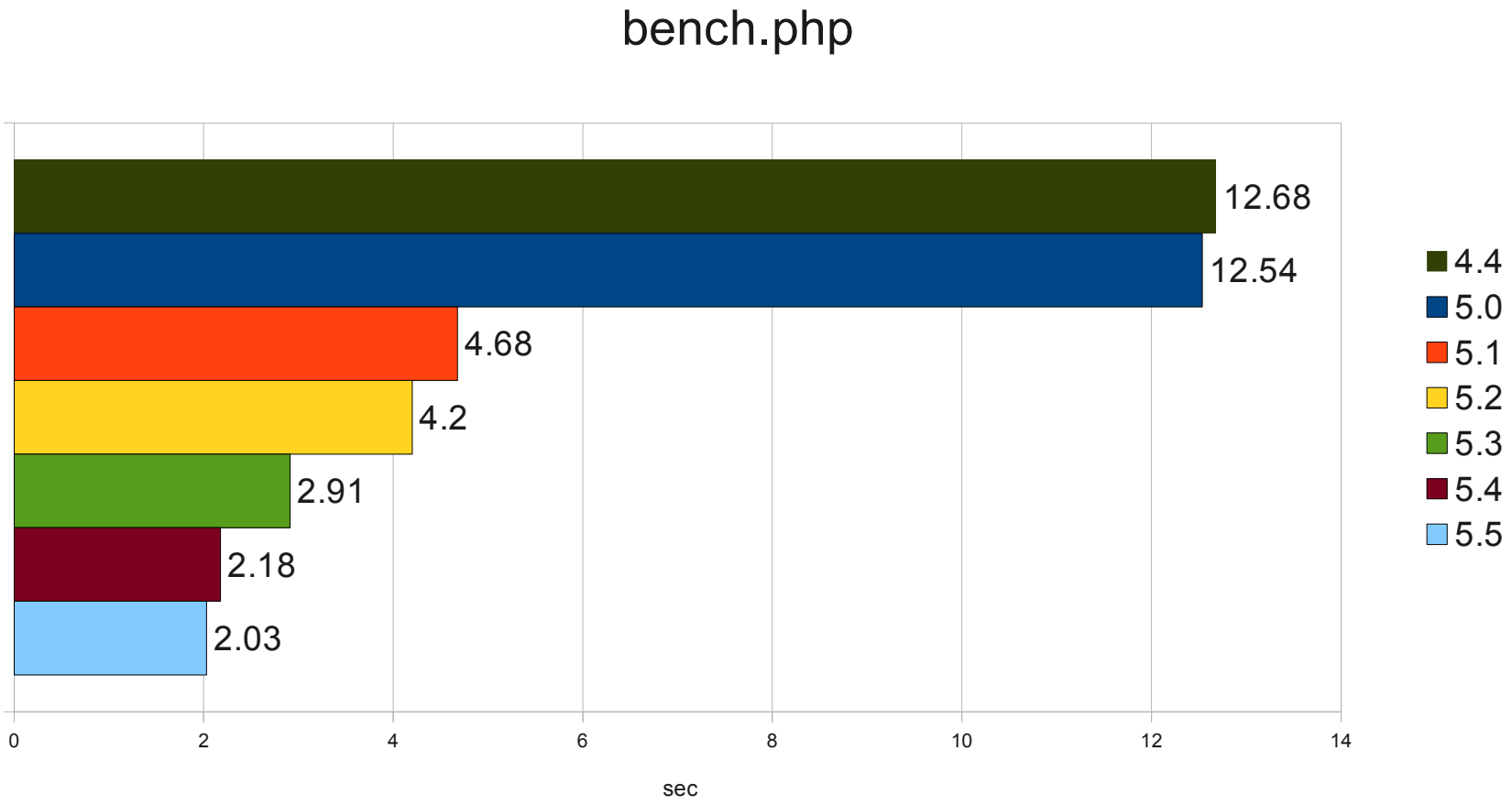
Кто Я?

- Работаю в области IT с 1991
- Первое знакомство с PHP в 2002
- Автор Turck MMCache (eAccelerator)
- Работяю в Zend Technologies с 2004
- Автор ext/soap
- Автор pecl/perl
- Один из ведущих разработчиков PHP
- Майнтейнер Zend OPcache
-

Кто Я?

- Работаю в области IT с 1991
- Первое знакомство с PHP в 2002
- Автор Turck MMCache (eAccelerator)
- Работяю в Zend Technologies с 2004
- Автор ext/soap
- Автор pecl/perl
- Один из ведущих разработчиков PHP
- Майнтейнер Zend OPcache
-

Изменение производительности PHP



PHP 5.0

- Выпущен в июле 2004
- Нововведения
  - Новая “настоящая” объектная модель
  - Деструкторы
  - Исключения (try/catch exception handling)
- Производительность
  - Call-Threaded интерпретатор

Call-Threaded интерпретатор

```
void execute(zend_op_array *op_array)
{
   ...
   EX(opline) = op_array->opcodes;
   while (1) {
      switch (EX(opline)->opcode) {
         ...
         case ZEND_ADD:
            ...
            EX(opline)++;
            break;
         ...
         case ZEND_RETURN:
            ...
            return;
         ...
      }
   }
}
```

```
int ZEND_ADD_HANDLER()
{
    ...
    EX(opline)++;
    return 0;
}

int ZEND_RETURN_HANDLER()
{
    ...
    return 1;
}

void execute(zend_op_array *op_array)
{
    ...
    EX(opline) = op_array->opcodes;
    while (1) {
        if (EX(opline)->handler()) {
            return;
        }
    }
}
```

PHP 5.1

- Выпущен в ноябре 2005
- Нововведения
  - Контроль типа аргумента для массивов (array type hinting)
  - Новая реализация FastCGI
- Производительность
  - Скомпилированные переменные (IS_CV)
  - Специализированный интерпретатор
  - Кэширование настоящих имен файлов

Скомпилированные переменные (IS_CV)

```php
<?php
function add($a, $b) {
    $c = $a + $b;
    return $c;
}
```

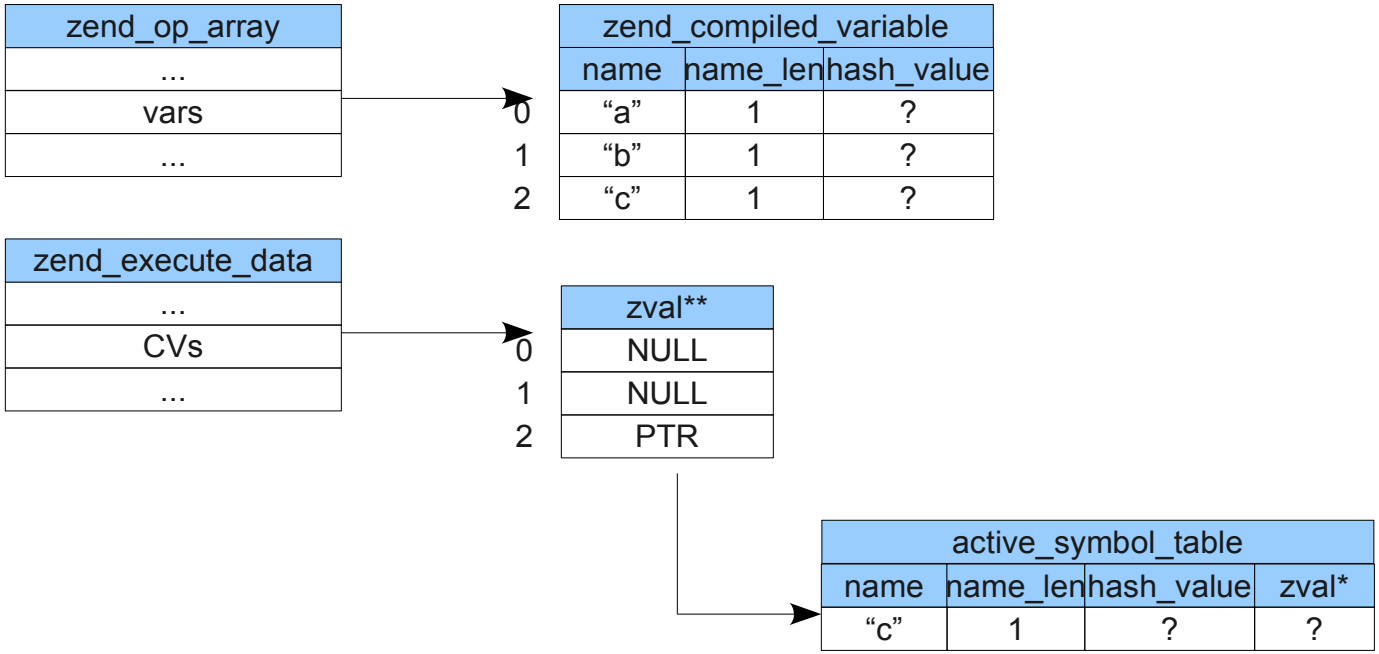
```
FETCH_W  “a”            -> var(0)
RECV     1, var(0)
FETCH_W  “b”            -> var(1)
RECV     2, var(1)
FETCH_R  “a”            -> var(2)
FETCH_R  “b”            -> var(3)
ADD      var(2), var(3) -> tmp(4)
FETCH_W  “c”            -> var(5)
ASSIGN   var(5), tmp(4)
FETCH_R  “c”            -> var(6)
RETURN   var(6)
```

- 11 инструкций

```
RECV    1, cv(0,“a”)

RECV    2, cv(1,“b”)


ADD     cv(0,”a”), cv(1,”b”) -> tmp(0)

ASSIGN  cv(2,”c”), tmp(0)

RETURN  cv(2,“c”);
```

- 5 инструкций

Скомпилированные переменные (IS_CV)



Скомпилированные переменные (IS_CV)

```php
<?php
function add($a, $b) {
    $c = $a + $b;
    return $c;
}
```

```
FETCH_W  “a”            -> var(0)
RECV     1, var(0)
FETCH_W  “b”            -> var(1)
RECV     2, var(1)
FETCH_R  “a”            -> var(2)
FETCH_R  “b”            -> var(3)
ADD      var(2), var(3) -> tmp(4)
FETCH_W  “c”            -> var(5)
ASSIGN   var(5), tmp(4)
FETCH_R  “c”            -> var(6)
RETURN   var(6)
```

- 11 инструкций
- 6 чтений
- 3 записи

```
RECV    1, cv(0,“a”)

RECV    2, cv(1,“b”)


ADD     cv(0,”a”), cv(1,”b”) -> tmp(0)

ASSIGN  cv(2,”c”), tmp(0)

RETURN  cv(2,“c”);
```

- 5 инструкций
- 3 “быстрых” чтения
- 3 “быстрых” записи

Специализированный интерпретатор

```
int zend_add_handler()


{



  add_function(
    &EX_T(opline->result.u.var).tmp_var,
    get_zval_ptr(
      &opline->op1, EX(Ts),
      &EG(free_op1), BP_VAR_R),
    get_zval_ptr(
      &opline->op2, EX(Ts),
      &EG(free_op2) BP_VAR_R)
    TSRMLS_CC);
  FREE_OP(
    EX(Ts),
    &opline->op1,
    EG(free_op1));
  FREE_OP(
    EX(Ts),
    &opline->op2,
    EG(free_op2));
  NEXT_OPCODE();
}
```

```
ZEND_VM_HANDLER(1, ZEND_ADD,
  CONST|TMP|VAR|CV,
  CONST|TMP|VAR|CV)
{
  zend_op *opline = EX(opline);
  zend_free_op free_op1, free_op2;

  add_function(
    &EX_T(opline->result.u.var).tmp_var,
    GET_OP1_ZVAL_PTR(BP_VAR_R),


    GET_OP2_ZVAL_PTR(BP_VAR_R)


    TSRMLS_CC);
  FREE_OP1();



  FREE_OP2();



  ZEND_VM_NEXT_OPCODE();
}
```

Специализированный интерпретатор

```
int zend_add_handler()


{



  add_function(
    &EX_T(opline->result.u.var).tmp_var,
    get_zval_ptr(
      &opline->op1, EX(Ts),
      &EG(free_op1), BP_VAR_R),
    get_zval_ptr(
      &opline->op2, EX(Ts),
      &EG(free_op2) BP_VAR_R)
    TSRMLS_CC);
  FREE_OP(
    EX(Ts),
    &opline->op1,
    EG(free_op1));
  FREE_OP(
    EX(Ts),
    &opline->op2,
    EG(free_op2));
  NEXT_OPCODE();
}
```

```
int ZEND_ADD_SPEC_CONST_CV_HANDLER()


{
  zend_op *opline = EX(opline);


  add_function(
    &EX_T(opline->result.u.var).tmp_var,
    &opline->op1.u.constant,


    _get_zval_ptr_cv(
      &opline->op2, EX(Ts),
      BP_VAR_R TSRMLS_CC)
    TSRMLS_CC);









  ZEND_VM_NEXT_OPCODE();
}
```

Кэширование настоящих имен файлов

```
function php_realpath($path) {
    static $cache = arrya();
    if (isset($cache[$path])) {
        return $cache[$path];
    } else {
        $realpath = realpath($path);
        if ($realpath) {
            $cache[$path] = $realpath;
        }
        return $realpath;
    }
}
```

PHP 5.2

- Выпущен в ноябре 2006
- Нововведения
  - ?
- Производительность
  - Новый менеджер памяти (Zend MM)
  - Оптимизация компиляции строк
  - Оптимизация некоторых внутренних функций для работы с массивами

Новый менеджер памяти (Zend MM)

- Хорошо оптимизированный классический алгоритм Doug Lea (dlmalloc)
- Быстрое освобождение памяти по окончанию HTTP запроса
- Прозрачное кэширование блоков наиболее часто используемых размеров
- Кажды поток (thread) использует свой собственный менеджер

PHP 5.3

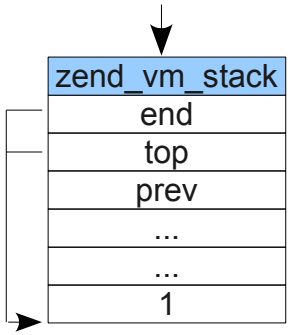
- Выпущен в июне 2009
- Нововведения
  - Пространства имен (namespaces)
  - Безымянные функции (closures)
  - Познее связывание классов
  - Оператор GOTO
  - NOWDOC
  - Сканер переписан с flex на re2c
  - Сборщик мусора (garbage collector)

PHP 5.3

- Производительность
  - Сегментированный стек VM
  - Исполнение без стека CPU (stackless VM)
  - Подстановка констант во время компиляции
  - Создание таблиц символов только в случае необходимости (lazy initialization)
  - Улучшенный кэш настоящих имен файлов

Сегментированный стек VM

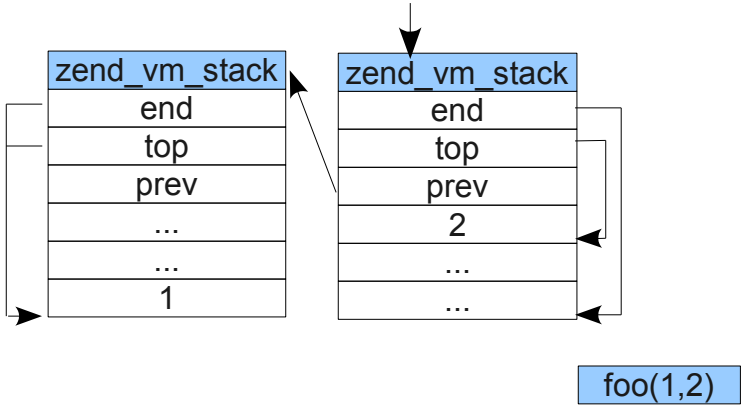
- Исключает перемещение стека (realloc)
- Фактические параметры всегда передаются в одном сегменте и могут быть доступпны по смещению



foo(1,2)

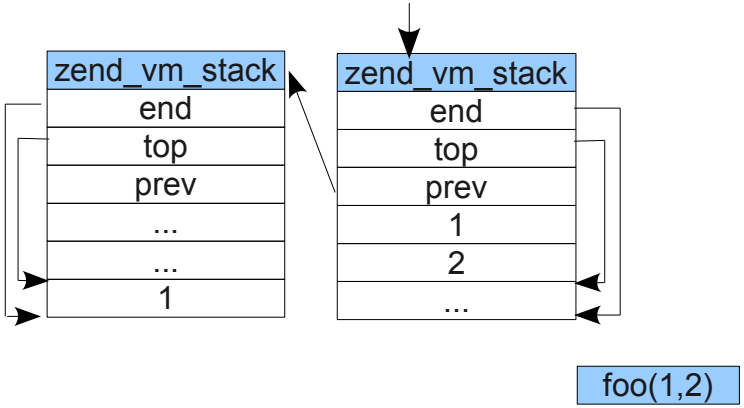
Сегментированный стек VM

- Исключает перемещение стека (realloc)
- Фактические параметры всегда передаются в одном сегменте и могут быть доступпны по смещению



Сегментированный стек VM

- Исключает перемещение стека (realloc)
- Фактические параметры всегда передаются в одном сегменте и могут быть доступпны по смещению



Stackless VM

```
int DO_FCALL_HANDLER()
{
  PRE_CALL();
  execute();
  POST_CALL()
  return 0;
}

int RETURN_HANDLER() {
  ...
  DESTROY_FRAME()
  return 1;
}

void execute(zend_op_array
    *op_array)
{
  INIT_FRAME()
  while (1) {
    if (EX(opline)->handler()) {
      return;
    }
  }
}
```

```
int DO_FCALL_HANDLER()
{
  PRE_CALL();
  INIT_FRAME();
  return 0;
}

int RETURN_HANDLER() {
  ...
  DESTROY_FRAME()
  POST_CALL();
  return 0;
}

void execute(zend_op_array
    *op_array)
{
  INIT_FRAME()
  while (1) {
    if (EX(opline)->handler()) {
      return;
    }
  }
}
```

Stackless VM

```
int DO_FCALL_HANDLER()
{
  PRE_CALL();
  execute();
  POST_CALL()
  return 0;
}

int RETURN_HANDLER() {
  ...
  DESTROY_FRAME()
  return 1;
}

void execute(zend_op_array
    *op_array)
{
  INIT_FRAME()
  while (1) {
    if (EX(opline)->handler()) {
      return;
    }
  }
}
```

```
int DO_FCALL_HANDLER()
{
  PRE_CALL();
  INIT_FRAME();
  return 0;
}

int RETURN_HANDLER() {
  ...
  DESTROY_FRAME()
  POST_CALL();
  return 0;
}

void execute(zend_op_array
    *op_array)
{
  INIT_FRAME()
  while (1) {
    if (EX(opline)->handler()) {
      return;
    }
  }
}
```

Подстановка констант во время компиляции

```php
<?php
function foo() {
    return false;
}
```

```
FETCH_CONSTANT  “false” -> tmp(0)
RETURN          tmp(0)
```

- 2 инструкции
- 1 чтение из таблицы

```
RETURN          bool(false)
```

- 1 инструкция
- 0 чтений

Ленивая инициализация таблиц символов

```php
<?php
function add($a, $b) {
    $c = $a + $b;
    return $c;
}
```

```
FETCH_W  “a”             -> var(0)
RECV     1, var(0)
FETCH_W  “b”             -> var(1)
RECV     2, var(1)
FETCH_R  “a”             -> var(2)
FETCH_R  “b”             -> var(3)
ADD      var(2), var(3)  -> tmp(4)
FETCH_W  “c”             -> var(5)
ASSIGN   var(5), tmp(4)
FETCH_R  “c”             -> var(6)
RETURN   var(6)
```

- 11 инструкций
- 6 чтений
- 3 записи

```
RECV    1, cv(0,“a”)

RECV    2, cv(1,“b”)


ADD     cv(0,”a”), cv(1,”b”) -> tmp(0)

ASSIGN  cv(2,”c”), tmp(0)

RETURN  cv(2,“c”);
```

- 5 инструкций
- 3 “быстрых” чтения
- 3 “быстрых” записи

Ленивая инициализация таблиц символов

```php
<?php
function add($a, $b) {
    $c = $a + $b;
    return $c;
}
```

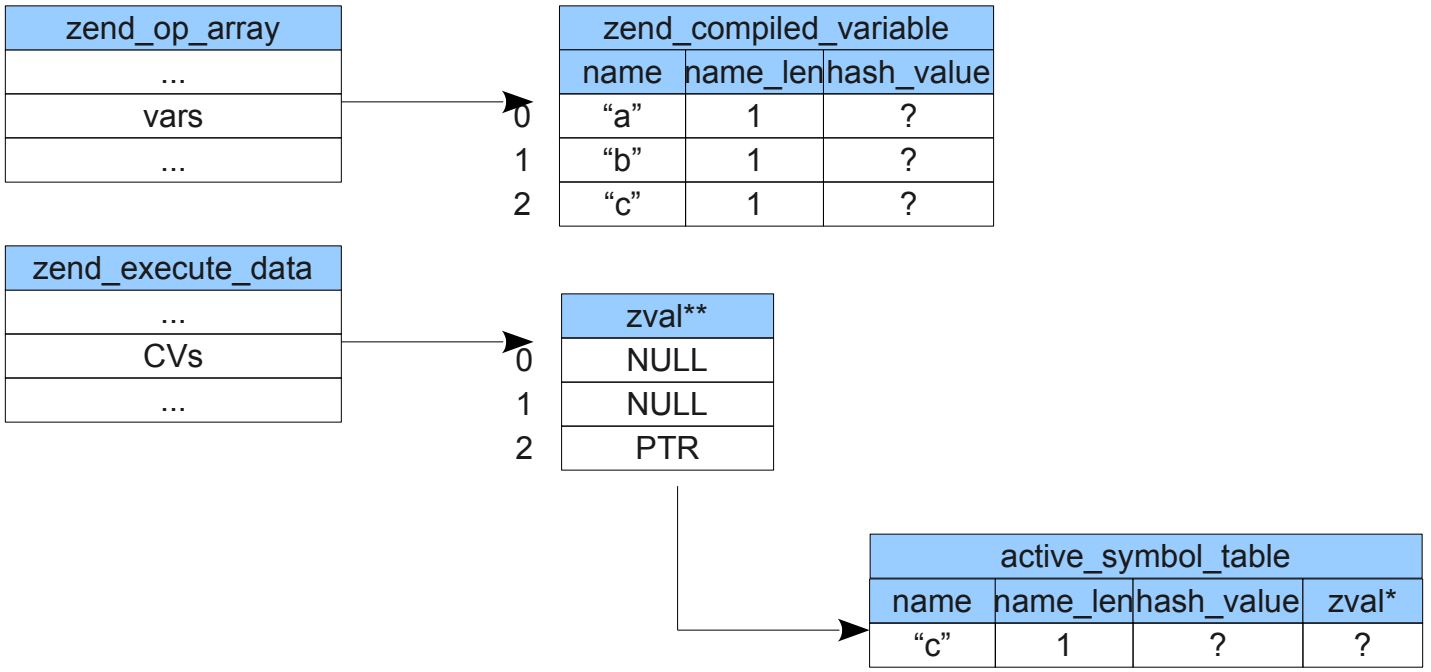
```
FETCH_W  “a”             -> var(0)
RECV     1, var(0)
FETCH_W  “b”             -> var(1)
RECV     2, var(1)
FETCH_R  “a”             -> var(2)
FETCH_R  “b”             -> var(3)
ADD      var(2), var(3)  -> tmp(4)
FETCH_W  “c”             -> var(5)
ASSIGN   var(5), tmp(4)
FETCH_R  “c”             -> var(6)
RETURN   var(6)
```

- 11 инструкций
- 6 чтений
- 3 записи

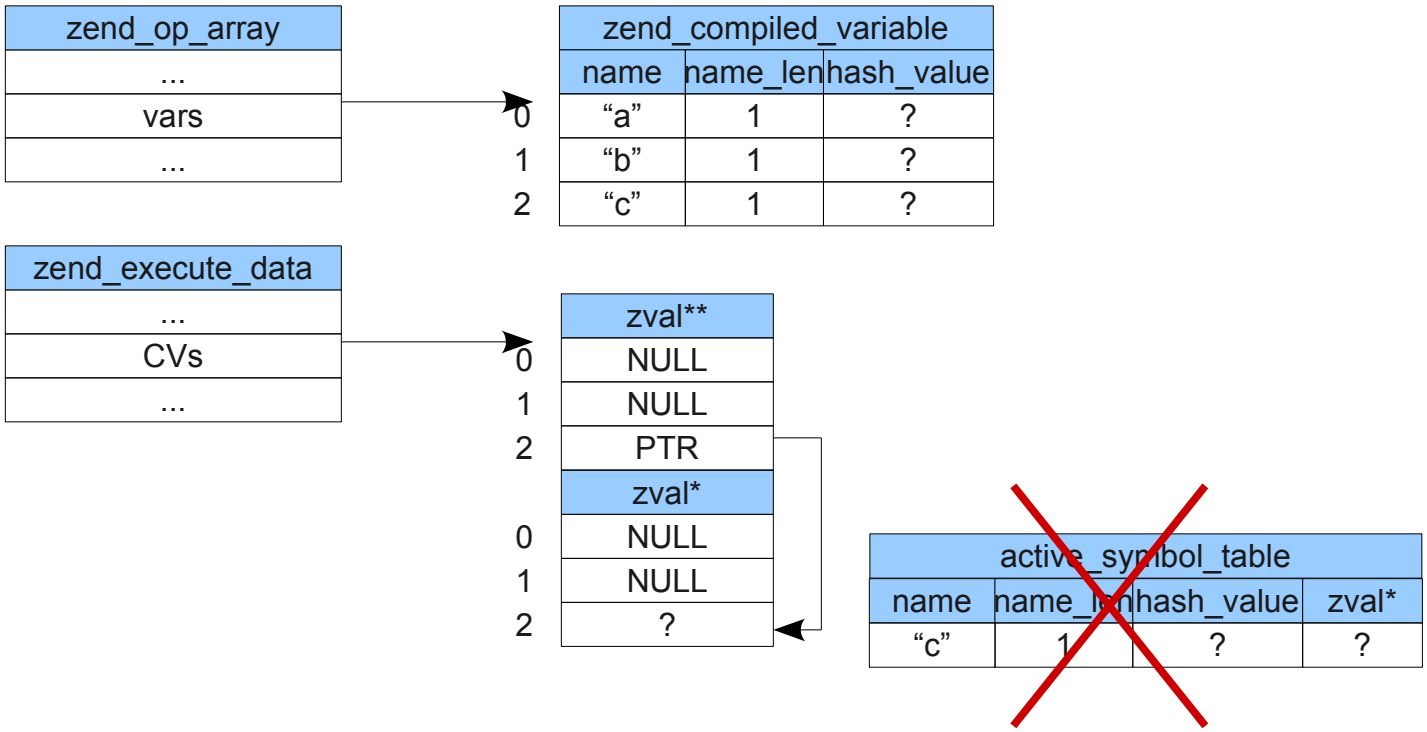
```
RECV    1, cv(0,“a”)

RECV    2, cv(1,“b”)


ADD     cv(0,”a”), cv(1,”b”) -> tmp(0)

ASSIGN  cv(2,”c”), tmp(0)

RETURN  cv(2,“c”);
```

- 5 инструкций
- 0 “быстрых” чтений
- 0 “быстрых” записей

Ленивая инициализация таблиц символов



Ленивая инициализация таблиц символов



Ленивая инициализация таблиц символов

- Некоторые динамические возможности PHP все таки требуют создания таблицы символов
  - доступ по имени ($$a)
  - eval(), include()
  - некоторые внутренние функции (parse_str, extract, compact, etc)
  - При выполнении top-level кода скрипта используется глобальная таблица символов

Улучшенный кэш настоящих имен файлов

```
function php_realpath($path) {
    static $cache = array();
    if (isset($cache[$path])) {
        return $cache[$path];
    }
    if (!lstat($path, $buf)) {
        return false; // file doesn't exists
    }
    if (S_ISLINK($buf['st_mode'])) {
        $realpath =
            php_realpath(read_symlink($path));
    } else {
        $realpath = php_realpath(dirname($path))
            . '/' . filename($path);
    }
    $cache[$path] = $realpath;
    return $realpath;
}
```

Улучшенный кэш настоящих имен файлов

```php
<?php
realpath(“/var/www/x1.php”);
realpath(“/var/www/x2.php”);
```

```
lstat64("/var", {...})
lstat64("/var/www", {...})
lstat64("/var/www/x1.php", {...})

lstat64("/var", {...})
lstat64("/var/www", {...})
lstat64("/var/www/x2.php", {...})
```

```
lstat64("/var/www/x1.php", {...})
lstat64("/var/www", {...})
lstat64("/var", {...})

lstat64("/var/www/x2.php", {...})
```

PHP 5.4

- Выпущен в марте 2012
- Нововведения
  - Traits
  - Array dereferencing - foo()[]
  - FPM SAPI

PHP 5.4

- Производительность
  - Отложеное выделение памяти под хэш таблицы (HashTable)
  - Таблицы констант
  - Кэширование связывания имен (run-time binding caching, inline caching)
  - Interned Strings
  - Уменьшение потребления памяти
  - Оптимизация последовательностей инициализации и завершения запроса

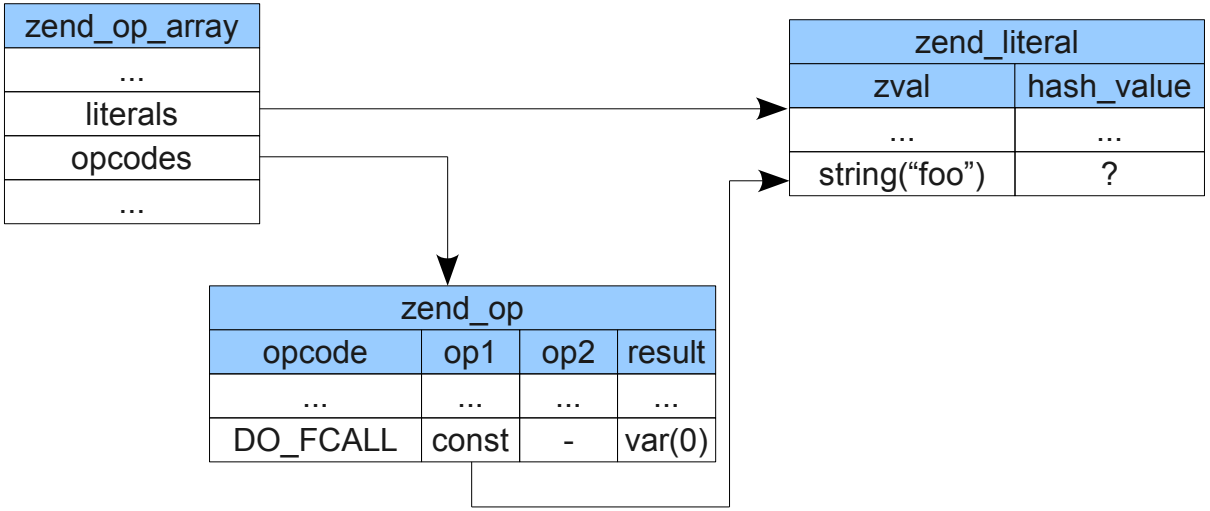
Отложенное выделение памяти под хэш таблицы

- Многие хэш таблицы всегда оставались пустыми (для примера многие классы не имеют статических членов или констант)
- Мы прозрачно откладываем реальное выделение памяти под хэш таблицы до вставки первого элемента
- Таким образом мы сохраняем память и время затраченное на бесполезные операции

Таблицы констант

- Людая инструкция VM может принимать константу в качестве операнда
- Вместо того что бы резервирывать место под константы в каждой инструкции мы помещаем все константы используемые функцией в отдельную таблицу
- Во время компиляции мы адресуем константы по индексу, а во время исполнения прямой ссылкой
- Размер инстукии уменшен с 76 до 28 байт (x86)

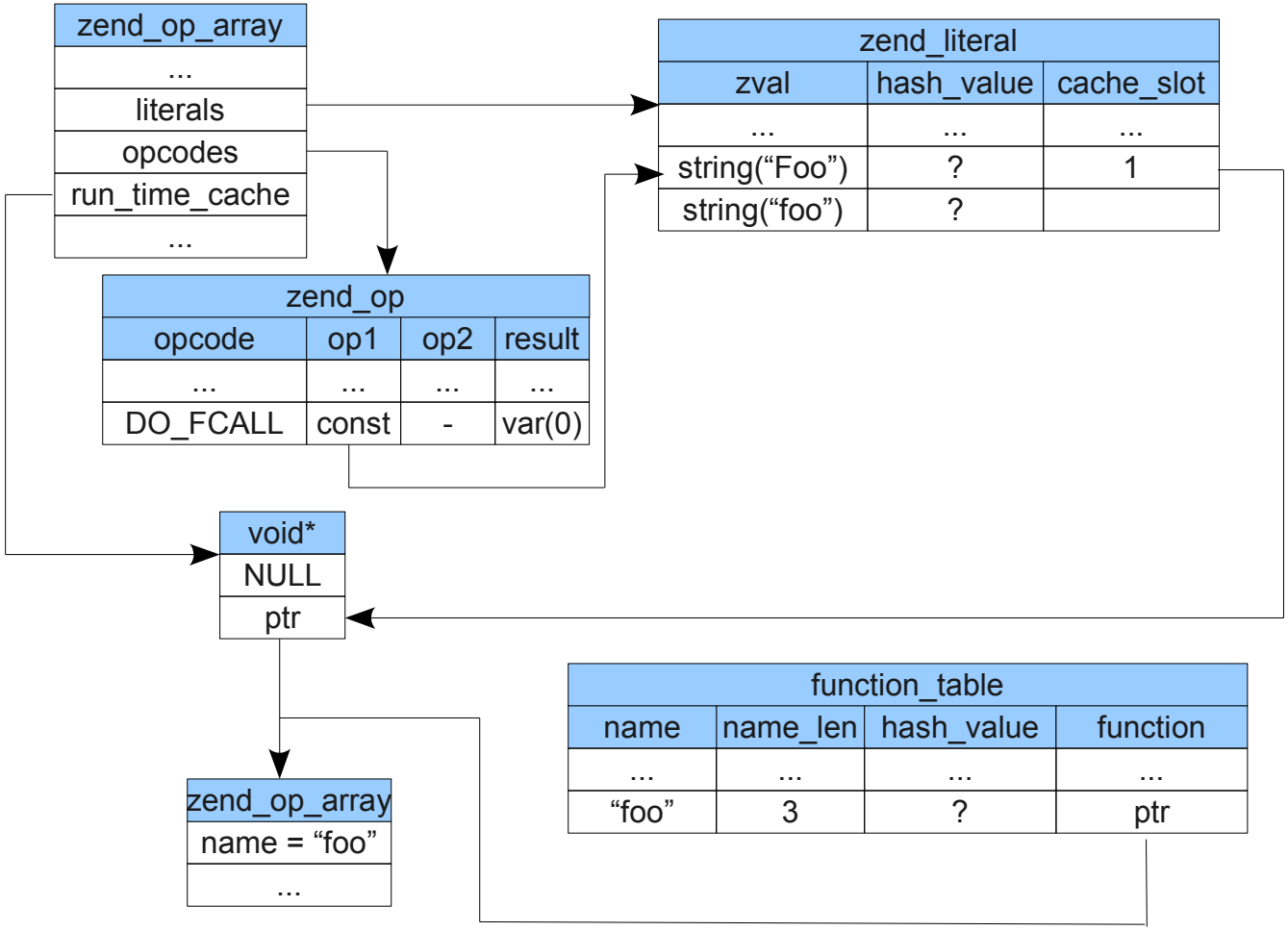
Таблицы констант



Run-Time Binding Cache

- Идея позаимствована у inline и polymorphic inline cache. Используется практически во всех JIT копиляторах для динамических и OO языков.
- Когда имя (например имя функции) связывается с реальной функцией мы запоминаем эту связь в кэше.
- Для виртуальных функций мы так же запоминаем к объекту какого класса мы обращались

Run-Time Binding Cache



Доступ к элементам объектов и классов

- Методы объектов определяются путем поиска (hash lookup) в таблице методов, найденное значкние сохраняется в кэше вместе с актуальным классом
- Определенные элементы объектов адресуются по смещению, это смещение находится поиском в таблице элементов (properties), найденное значкние сохраняется в кэше вместе с актуальным классом
- Доступ к динамические элементы объектов и элементам массивов не кэшируется

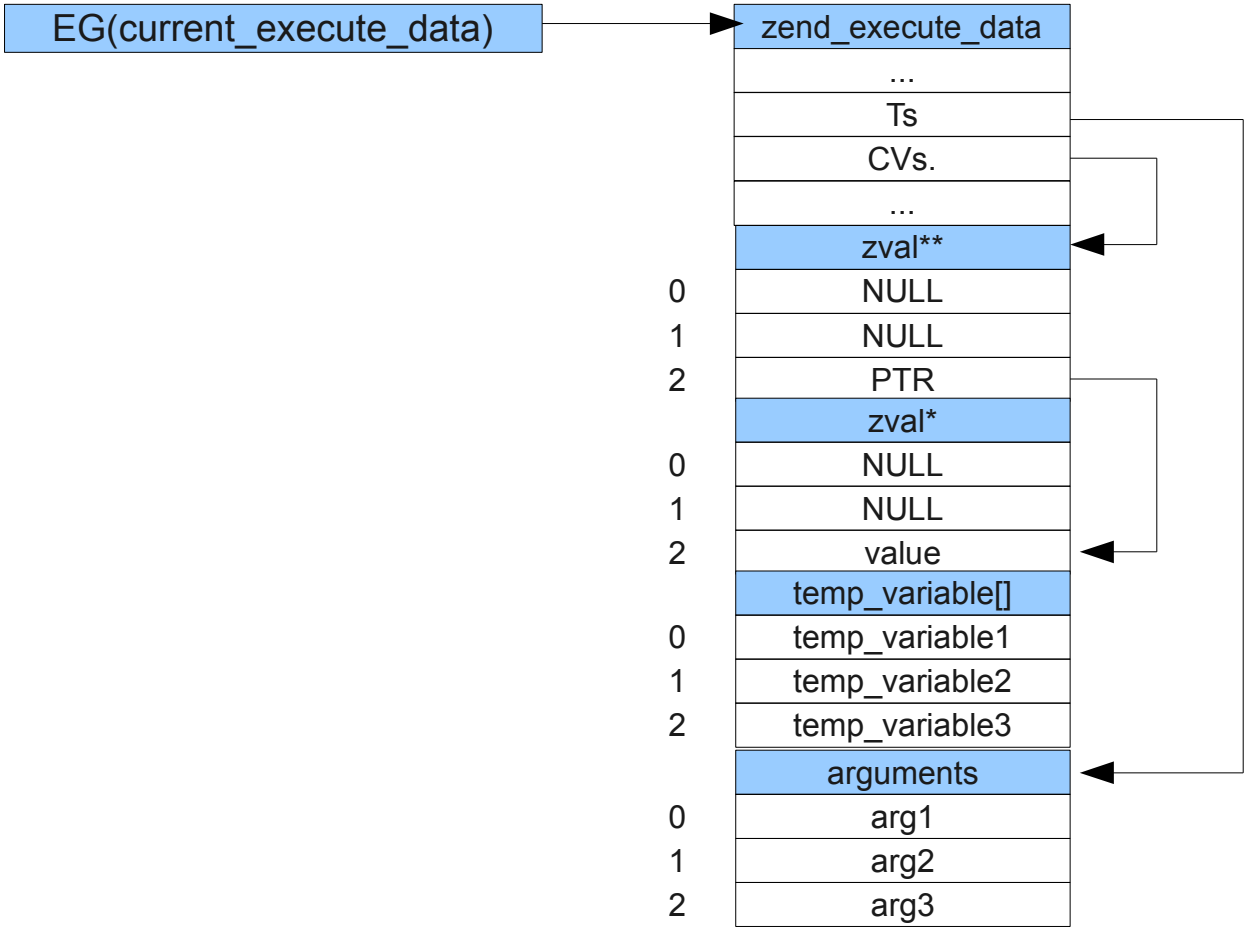
Interned Strings

- Сохраняем только одну копию каждой конеретной строки
- Мы можем сравнивать на равенство указатели вместо вызова strcmp()
- Больше не надо копировать строки
- Сохраняем длины строк и hash_value вместе со строками, исключаем повторяющиеся вычисления
- Используем Interned Strings только для строк известных во время компиляции
-

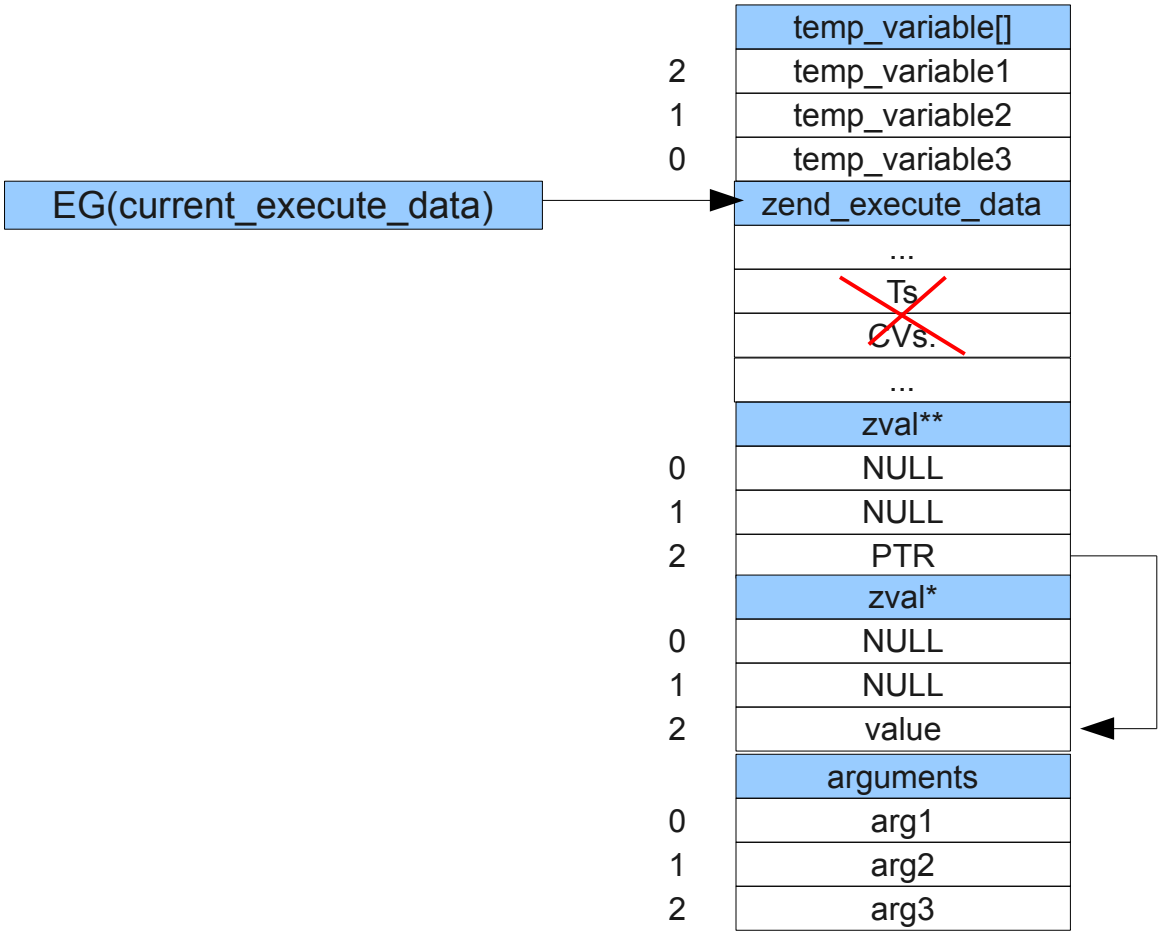
PHP 5.5

- Будет выпущен в 2013
- Нововведения
  - Генераторы и сопроцессы (yield)
  - Finally блок при обработке исключений
- Производитеьность
  - Улучшение во фрейме активации функций
  - Zend OPcache

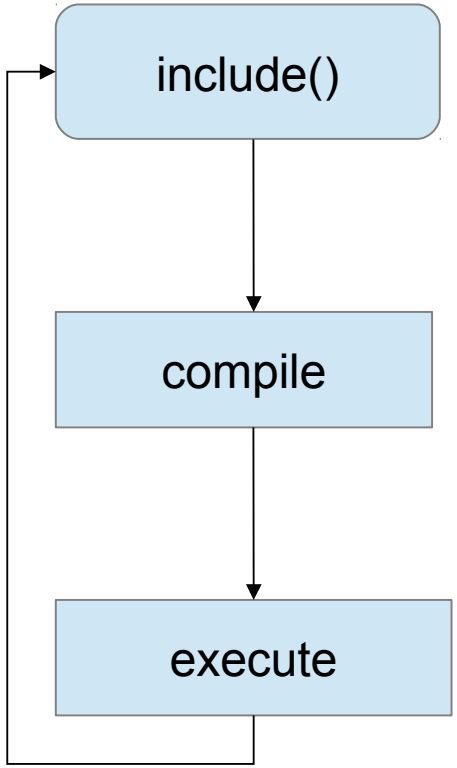
Улучшения во фрейме активации функции



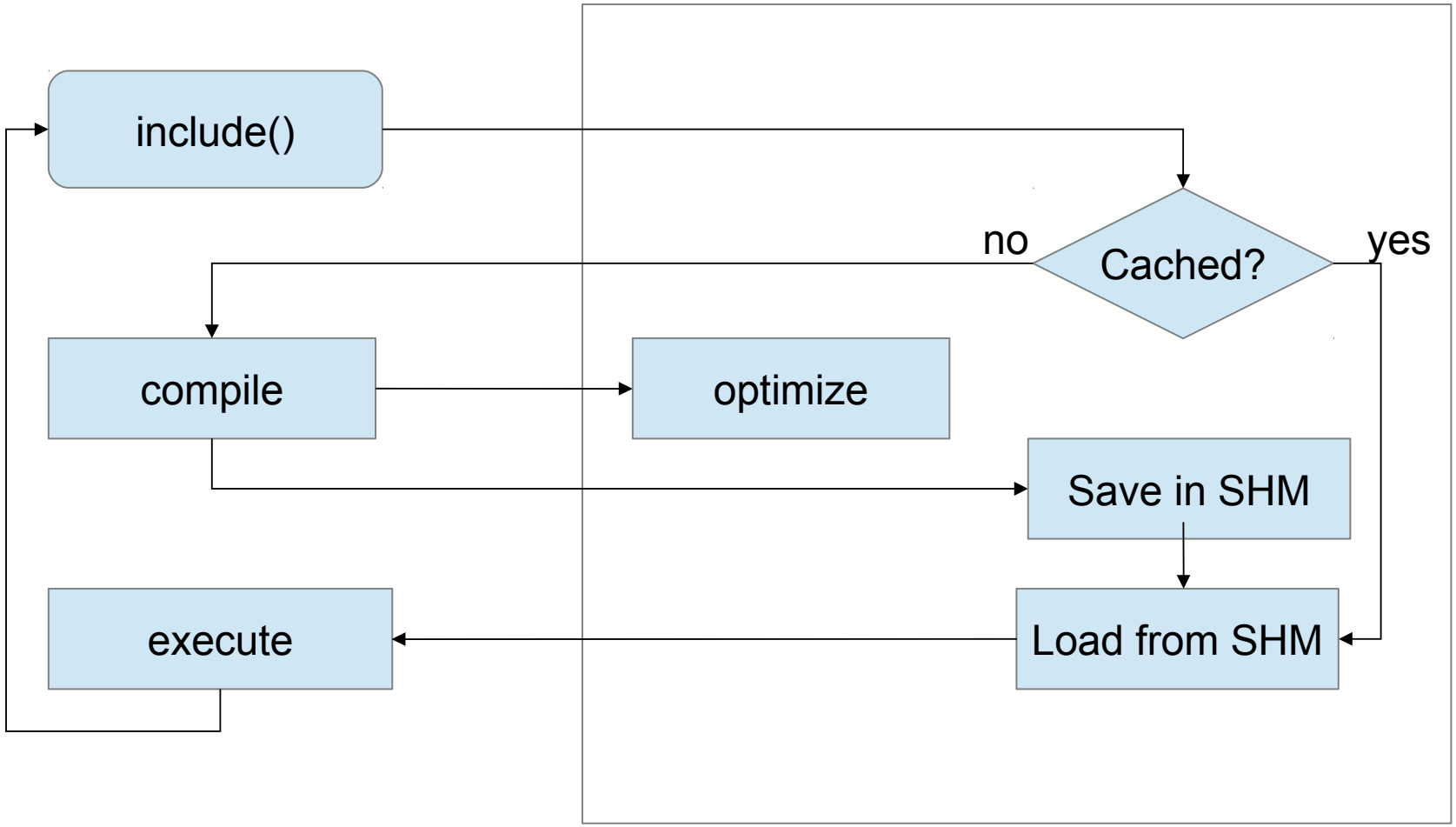
Улучшения во фрейме активации функции



Zend OPcache



Zend OPcache



Zend Opcache: Оптимизация байт-кода

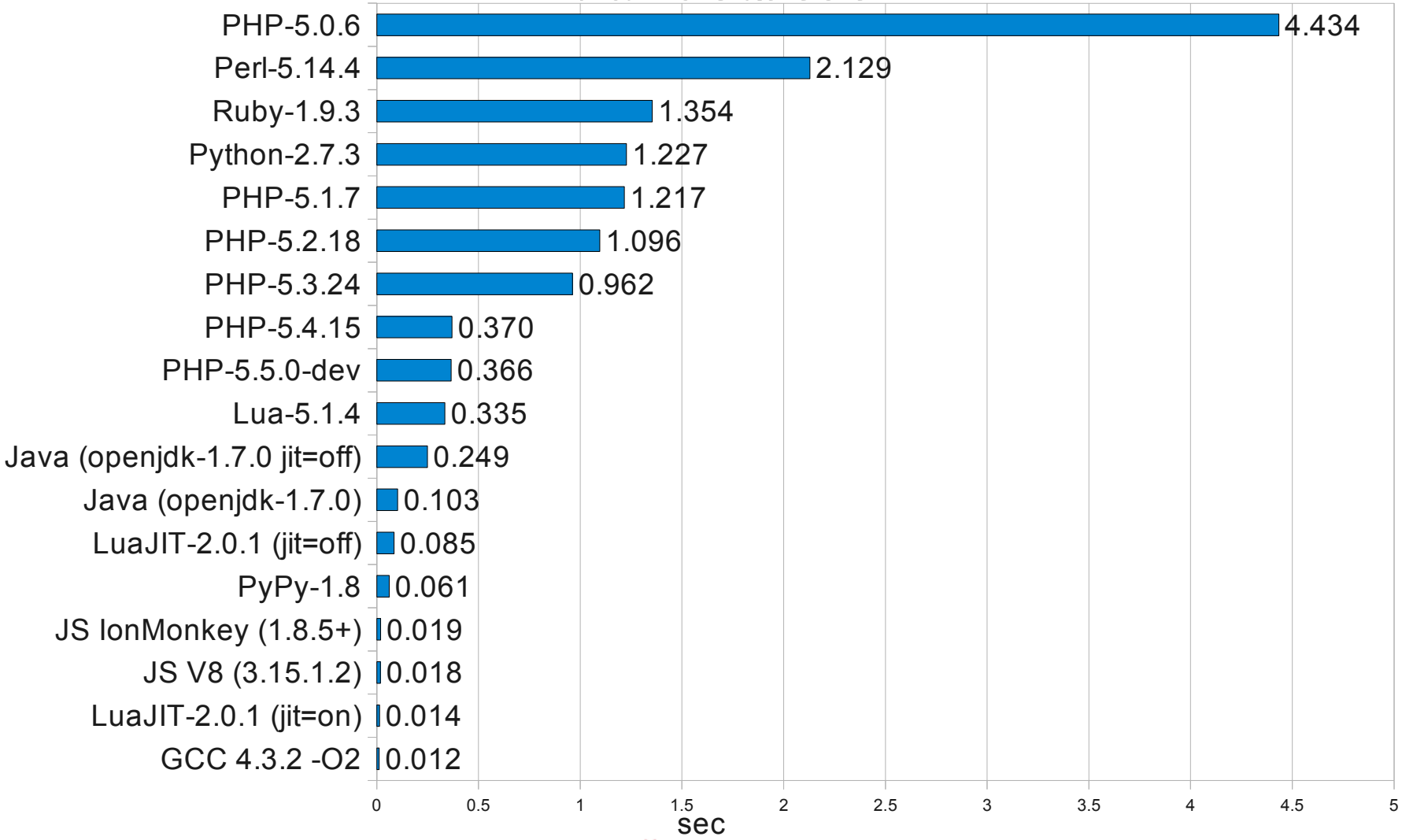
- Вычесление константных выражений
- Упрощение выражений
- Оптимизация переходов
- Повторное использывание временных переменных
- Слияние идентичных констант
- Удаление неиспользуемового кода

Zend OPcache

	PHP 5.5 (req/sec)	PHP & OPcache (req/sec)	Speedup (times)
blog	42.9	101.5	2.4
drupal	594.1	1653.3	2.8
fw	14.8	217.7	14.7
hello	7486.57	12851.8	1.7
qdig	250.3	439.8	1.8
typo3	90.9	575	6.3
wordpress	63.6	183.2	2.9
xoops	51.5	130.8	2.5
scrum	54.2	175.1	3.2

Где мы в сравненни с другими языками?

Mandelbtot



Что дальше?

- Использывание более агрессиыных методов оптимизации байт-кода
- Интер-процедурная оптимизация
- Более тесная интеграция OPcache в PHP
- Компиляция
- JIT

Вопросы?