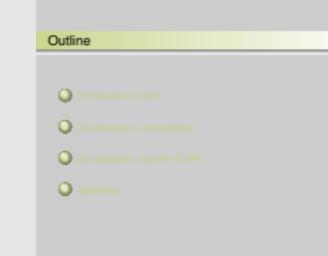


# A Practical Solution for Scripting Language Compilers

Paul Biggar, Edsko de Vries and David Gregg

Department of Computer Science and Statistics  
Trinity College Dublin

SAC '09: 11th March, 2009

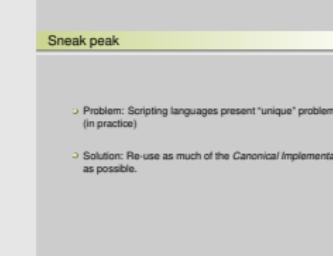
 Outline

Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

# Outline

- 1 Introduction to phc
- 2 Challenges to compilation
- 3 phc solution: use the C API
- 4 Speedup

- └ Sneak peak



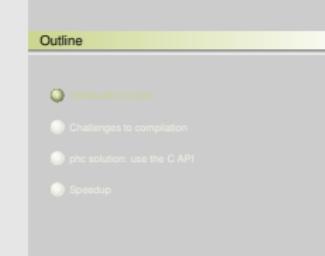
Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Sneak peak

- Problem: Scripting languages present “unique” problems (in practice)
- Solution: Re-use as much of the *Canonical Implementation* as possible.

- └ Introduction to phc

- └ Outline



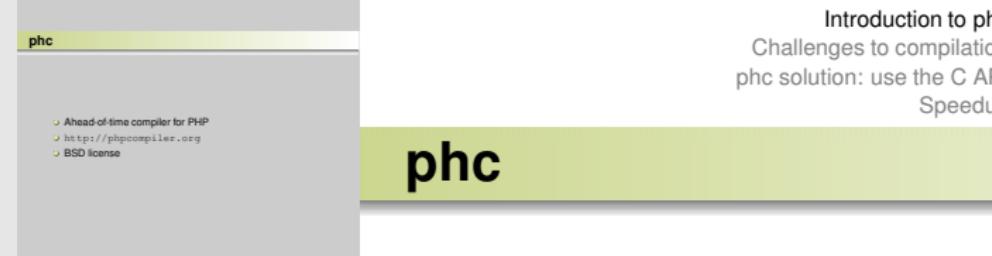
Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Outline

- 1 Introduction to phc
- 2 Challenges to compilation
- 3 phc solution: use the C API
- 4 Speedup

└ Introduction to phc

└ **phc**

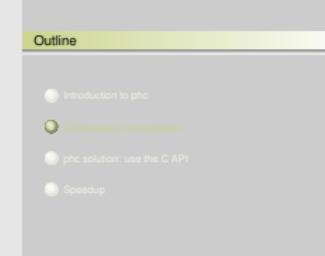


## 1. BSD licence useful since its easy to extend

- Ahead-of-time compiler for PHP
- <http://phpcompiler.org>
- BSD license

└ Challenges to compilation

└ Outline



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Outline

1 Introduction to phc

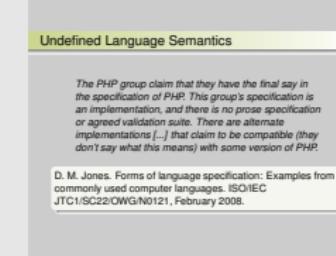
2 Challenges to compilation

3 phc solution: use the C API

4 Speedup

- Challenges to compilation

- Undefined Language Semantics



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Undefined Language Semantics

*The PHP group claim that they have the final say in the specification of PHP. This group's specification is an implementation, and there is no prose specification or agreed validation suite. There are alternate implementations [...] that claim to be compatible (they don't say what this means) with some version of PHP.*

D. M. Jones. Forms of language specification: Examples from commonly used computer languages. ISO/IEC JTC1/SC22/OWG/N0121, February 2008.

- Challenges to compilation

- Batteries included

- all written in C, not PHP
- Mike Furr earlier: 1000 methods/classes in C
- 4870 functions, 1000 methods



Introduction to phc  
 Challenges to compilation  
 phc solution: use the C API  
 Speedup

## Batteries included

abs()	apc_load_constants()	array_intersect()	array_values()
acos()	apc_sma_info()	array_intersect_assoc()	array_walk()
acosh()	apc_store()	array_intersect_key()	array_walk_recursive()
addslashes()	apd_breakpoint()	array_intersect_uassoc()	ArrayIterator::current()
addslashes()	apd_callstack()	array_intersect_ukey()	ArrayIterator::key()
aggregate()	apd_clunk()	array_key_exists()	ArrayIterator::next()
aggregate_info()	apd_continue()	array_keys()	ArrayIterator::rewind()
aggregate_methods()	apd_croak()	array_map()	ArrayIterator::seek()
aggregate_methods_by_list()	apd_dump_function_table()	array_merge()	ArrayIterator::valid()
aggregate_methods_by_regexp()	apd_dump_persistent_resources()	array_merge_recursive()	ArrayObject::__construct()
aggregate_properties()	apd_dump_regular_resources()	array_multisort()	ArrayObject::append()
aggregate_properties_by_list()	apd_echo()	array_pad()	ArrayObject::count()
aggregate_properties_by_regexp()	apd_get_active_symbols()	array_pop()	ArrayObject::getIterator()
aggregation_info()	apd_set_pprof_trace()	array_product()	ArrayObject::offsetExists()
apache_child_terminate()	apd_set_session()	array_push()	ArrayObject::offsetGet()
apache_get_modules()	apd_set_session_trace()	array_rand()	ArrayObject::offsetSet()
apache_get_version()	apd_set_socket_session_trace()	array_reduce()	ArrayObject::offsetUnset()
apache_getenv()	array()	array_reverse()	arsort()
apache_lookup_uri()	array_change_key_case()	array_search()	ascii2ebcdic()
apache_note()	array_chunk()	array_shift()	asin()
apache_request_headers()	array_combine()	array_slice()	asinh()
apache_reset_timeout()	array_count_values()	array_splice()	asort()
apache_response_headers()	array_diff()	array_sum()	aspell_check()
apache_setenv()	array_diff_assoc()	array_udiff()	aspell_check_raw()
apc_add()	array_diff_key()	array_udiff_assoc()	aspell_new()
apc_cache_info()	array_diff_uassoc()	array_udiff_uassoc()	aspell_suggest()
apc_clear_cache()	array_diff_ukey()	array_uintersect()	assert()
apc_compile_file()	array_fill()	array_uintersect_assoc()	assert_options()
apc_define_constants()	array_filter()	array_uintersect_uassoc()	atan()
apc_delete()	array_flip()	array_unique()	atan2()
apc_fetch()	array_unshift()	array_unshift()	atanh()

Jeff Atwood, Coding Horror, May 20th, 2008

<http://www.codinghorror.com/blog/archives/001119.html>

└ Challenges to compilation

└ Change between releases



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Change between releases

```
<?php
var_dump (0x9fa0ff0b);
?>
```

PHP 5.2.1 (32-bit)  
int(2147483647)

PHP 5.2.3 (32-bit)  
float(2678128395)

└ Challenges to compilation

  └ Run-time code generation

The slide has a green header bar with the title 'Run-time code generation'. Below the header are two code snippets:

```
Run-time code generation
```

```
<?php
eval ($argv[1]);
?>
```

```
<?php
include ("mylib.php");
...
include ("plugin.php");
...
?>
```

1. scripting langs are typically made for interpreters
2. can do source inclusion at compile time
3. same mechanism for plugins

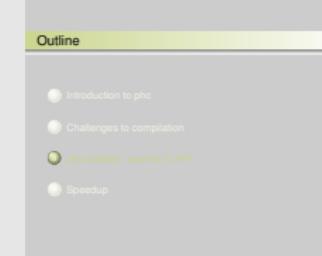
```
<?php
eval ($argv[1]);
?>
```

```
<?php
include ("mylib.php");
...
include ("plugin.php");
...
?>
```

## Run-time code generation

└ phc solution: use the C API

└ Outline



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

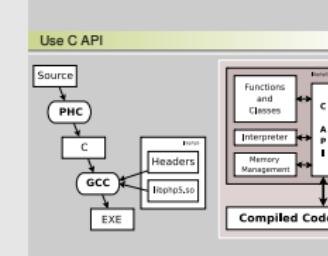
## Outline

- 1 Introduction to phc
- 2 Challenges to compilation
- 3 phc solution: use the C API
- 4 Speedup

## A Practical Solution for Scripting Language Compilers

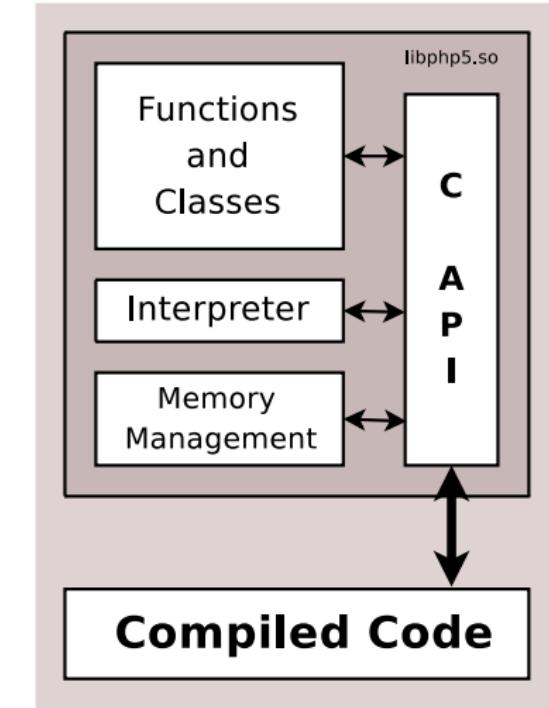
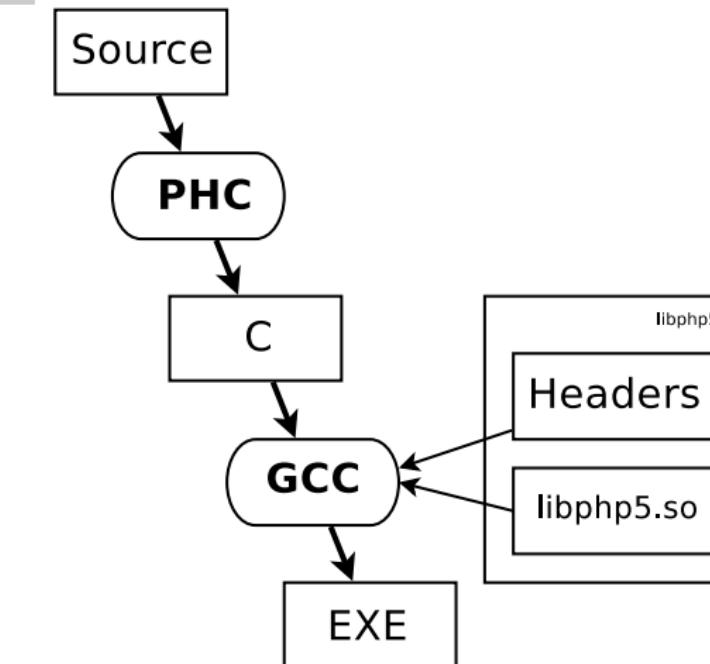
- └ phc solution: use the C API

- └ Use C API



## Use C API

1. RTCG
2. Functions
3. Changes between releases: also use C API at compile-time



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## A Practical Solution for Scripting Language Compilers

└ phc solution: use the C API

└ More detail

More detail	
PHP	zval
Python	PyObject
Ruby	VALUE
Lua	TValue

H. Muhammad and R. Ierusalimschy. C APIs in extension and extensible languages. Journal of Universal Computer Science, 13(6):839–853, 2007.

1. C API is just zval + macros and functions
2. Use (target) PHP's C API at run-time

## More detail

Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

PHP	zval
Python	PyObject
Ruby	VALUE
Lua	TValue

H. Muhammad and R. Ierusalimschy. C APIs in extension and extensible languages. Journal of Universal Computer Science, 13(6):839–853, 2007.

└ phc solution: use the C API

└ Applicability



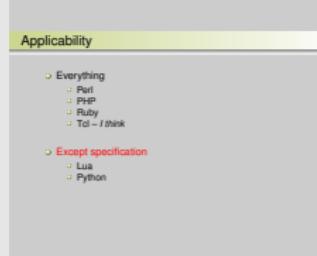
## Applicability

- Everything

- Perl
- PHP
- Ruby
- Tcl – *I think*

└ phc solution: use the C API

└ Applicability



## Applicability

- Everything

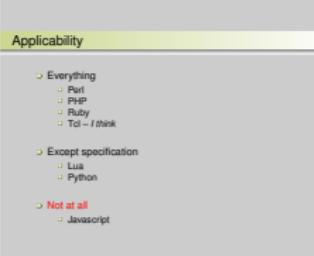
- Perl
- PHP
- Ruby
- Tcl – *I think*

- Except specification

- Lua
- Python

└ phc solution: use the C API

└ Applicability



## Applicability

- Everything

- Perl
- PHP
- Ruby
- Tcl – *I think*

- Except specification

- Lua
- Python

- Not at all

- Javascript

└ phc solution: use the C API

└ Simple listings: \$i = 0

The screenshot shows a terminal window with the title 'Simple listings: \$i = 0'. The output of the phc compiler is displayed, which is a series of C-like statements that implement the assignment operation.

```
// $i = 0;
{
    zval* p_i;
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);
    php_destruct (p_i);
    php_allocate (p_i);
    ZVAL_LONG (*p_i, 0);
}
```

## Simple listings: \$i = 0

```
// $i = 0;
{
    zval* p_i;
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);
    php_destruct (p_i);
    php_allocate (p_i);
    ZVAL_LONG (*p_i, 0);
}
```

## A Practical Solution for Scripting Language Compilers

- └ phc solution: use the C API

- └ Example: \$i = 0

```
Example: $i = 0

// $i = 0
if (local_i == NULL)
{
    local_i = EG (uninitialized_zval_ptr);
    local_i->refcount++;
}
zval **p_lhs = &local_i;

zval *value;
if ((*p_lhs)->is_ref)
{
    // Always overwrite the current value
    value = *p_lhs;
    zval_dtor (value);
}
else
{
    ALLOC_INIT_ZVAL (value);
    zval_ptr_dtor (p_lhs);
    *p_lhs = value;
}

ZVAL_LONG (value, 0);
```

Introduction to phc  
 Challenges to compilation  
**phc solution: use the C API**  
 Speedup

## Example: \$i = 0

```
// $i = 0;
{
    if (local_i == NULL)
    {
        local_i = EG (uninitialized_zval_ptr);
        local_i->refcount++;
    }
    zval **p_lhs = &local_i;

    zval *value;
    if ((*p_lhs)->is_ref)
    {
        // Always overwrite the current value
        value = *p_lhs;
        zval_dtor (value);
    }
    else
    {
        ALLOC_INIT_ZVAL (value);
        zval_ptr_dtor (p_lhs);
        *p_lhs = value;
    }

    ZVAL_LONG (value, 0);
```

## A Practical Solution for Scripting Language Compilers

- └ phc solution: use the C API

- └ Example: \$i = \$j



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Example: \$i = \$j

```
// $i = $j;
{
    if (local_i == NULL)
    {
        local_i = EG(uninitialized_zval_ptr);
        local_i->refcount++;
    }
    zval **p_lhs = &local_i;

    zval *rhs;
    if (local_j == NULL)
        rhs = EG(uninitialized_zval_ptr);
    else
        rhs = local_j;

    if (*p_lhs != rhs)
    {
        if ((*p_lhs)->is_ref)
        {
            // First, call the destructor to remove any data structures
            // associated with lhs that will now be overwritten
            zval_dtor(*p_lhs);
            // Overwrite LHS
            (*p_lhs)->value = rhs->value;
            (*p_lhs)->type = rhs->type;
            zval_copy_ctor(*p_lhs);
        }
        else
        {
            zval_ptr_dtor(p_lhs);
            if (rhs->is_ref)
            {
                // Take a copy of RHS for LHS
                *p_lhs = zvp_clone_ex(rhs);
            }
            else
            {
                // Share a copy
                rhs->refcount++;
                *p_lhs = rhs;
            }
        }
    }
}
```

## A Practical Solution for Scripting Language Compilers

- └ phc solution: use the C API

- └ Example: printf (\$f)

```
Example: printf ($f)
```

The screenshot shows a terminal window with the title "Example: printf (\$f)". The window displays the output of the command "printf '%s\n' \$f", which prints the contents of the file "f" followed by a new line. The file "f" contains the text "Hello, world!". The terminal window has a light gray background and a dark gray header bar.

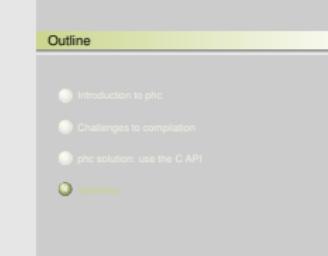
Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Example: printf (\$f)

```
./phc ./src/printf.c -O3 -fPIC -o ./bin/printf.o
./phc ./src/printf.o -fPIC -o ./bin/printf
./bin/printf > ./bin/f
cat ./bin/f
Hello, world!
```

- Speedup

- Outline



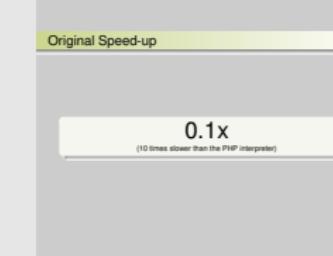
Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Outline

- 1 Introduction to phc
- 2 Challenges to compilation
- 3 phc solution: use the C API
- 4 Speedup

- Speedup

- Original Speed-up



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Original Speed-up

1. Why is experimental evaluation a speedup?
2. That's an interesting result. Shouldnt compilers always be faster!!!
3. PHP's interpreter isnt slowed by interpreter loop. Rather its the level of dynamicism.

**0.1x**

(10 times slower than the PHP interpreter)

- └ Speedup

- └ The problem with copies

1. each statement is pretty high level

The problem with copies

```
<?php
for ($i = 0; $i < $n; $i++)
    $str = $str . "hello";
?>

<?php
for ($i = 0; $i < $n; $i++)
{
    $T = $str . "hello";
    $str = $T;
}
?>
```

Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

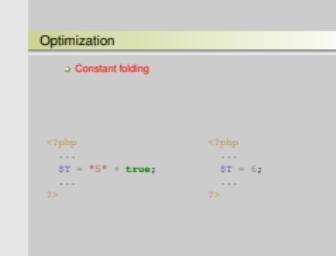
## The problem with copies

```
<?php
for ($i = 0; $i < $n; $i++)
    $str = $str . "hello";
?>
```

```
<?php
for ($i = 0; $i < $n; $i++)
{
    $T = $str . "hello";
    $str = $T;
}
?>
```

- └ Speedup

- └ Optimization



## Optimization

### Constant folding

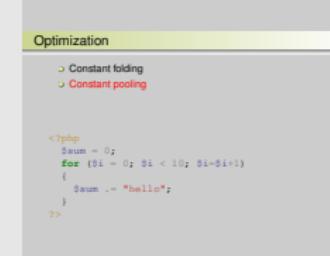
1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

```
<?php
...
$T = "5" + true;
...
?>
```

```
<?php
...
$T = 6;
...
?>
```

- └ Speedup

- └ Optimization



1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

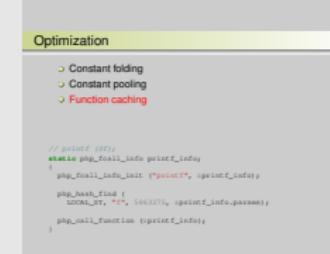
## Optimization

- Constant folding
- Constant pooling

```
<?php
$sum = 0;
for ($i = 0; $i < 10; $i=$i+1)
{
    $sum .= "hello";
}
?>
```

- └ Speedup

- └ Optimization



1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Optimization

- Constant folding
- Constant pooling
- Function caching

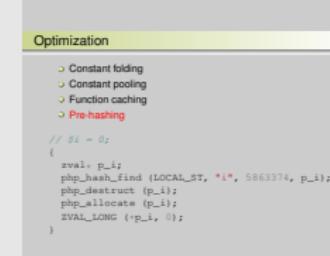
```
// printf ($f);
static php_fcall_info printf_info;
{
    php_fcall_info_init ("printf", &printf_info);

    php_hash_find (
        LOCAL_ST, "f", 5863275, &printf_info.params);

    php_call_function (&printf_info);
}
```

- └ Speedup

- └ Optimization



1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

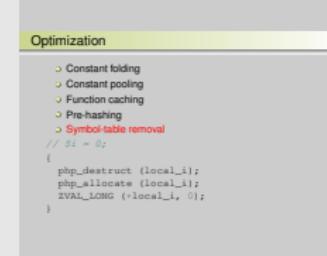
## Optimization

- Constant folding
- Constant pooling
- Function caching
- Pre-hashing

```
// $i = 0;
{
    zval* p_i;
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);
    php_destruct (p_i);
    php_allocate (p_i);
    ZVAL_LONG (*p_i, 0);
}
```

- └ Speedup

- └ Optimization



1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

## Optimization

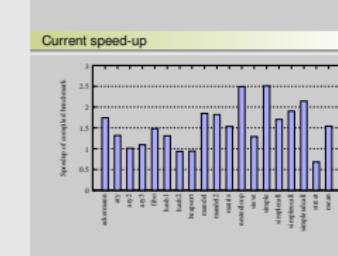
- Constant folding
- Constant pooling
- Function caching
- Pre-hashing
- **Symbol-table removal**

```
// $i = 0;
{
    php_destruct (local_i);
    php_allocate (local_i);
    ZVAL_LONG (*local_i, 0);
}
```

- Speedup

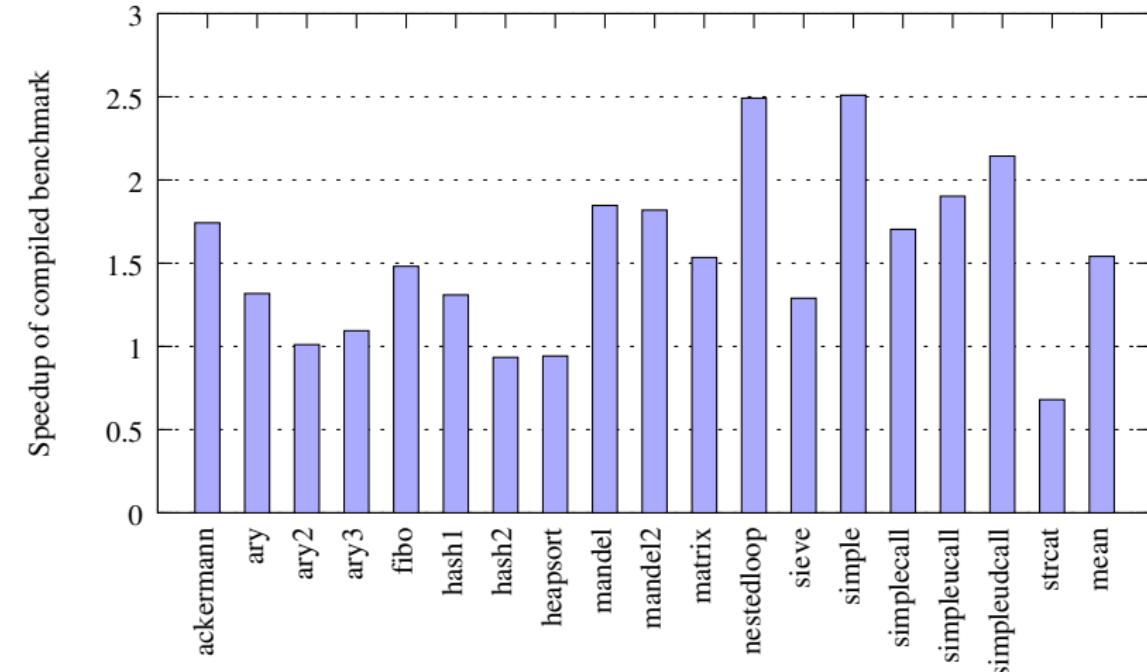
- Current speed-up

1. Explain how to read graph
2. Much better than 0.1x
3. C compiler: be 5x faster
4. PHP 40x-70x slower



Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Current speed-up



- Speedup

- Summary

The screenshot shows a presentation slide with a green header bar containing the word 'Summary'. Below the header is a list of bullet points:

- Scripting languages pose new problems for compilers
- Solution: Re-use existing run-time
  - Speed-ups of 1.5x
  - Future work: Precise optimization required for speed
- <http://phpcompiler.org>

Introduction to phc  
Challenges to compilation  
phc solution: use the C API  
Speedup

## Summary

- Scripting languages pose new problems for compilers
- Solution: Re-use existing run-time
  - Speed-ups of 1.5x
  - Future work: Precise optimization required for speed
- <http://phpcompiler.org>