Building a JIT compiler for PHP in 2 days

Nuno Lopes
nuno.lopes@ist.utl.pt
Instituto Superior Técnico
Technical University of Lisbon

Outline

- Overview of the Zend VM
- Design Rationale
- Implementation
- Results
- Future Work

Overview of the Zend VM

Overview of the Zend VM

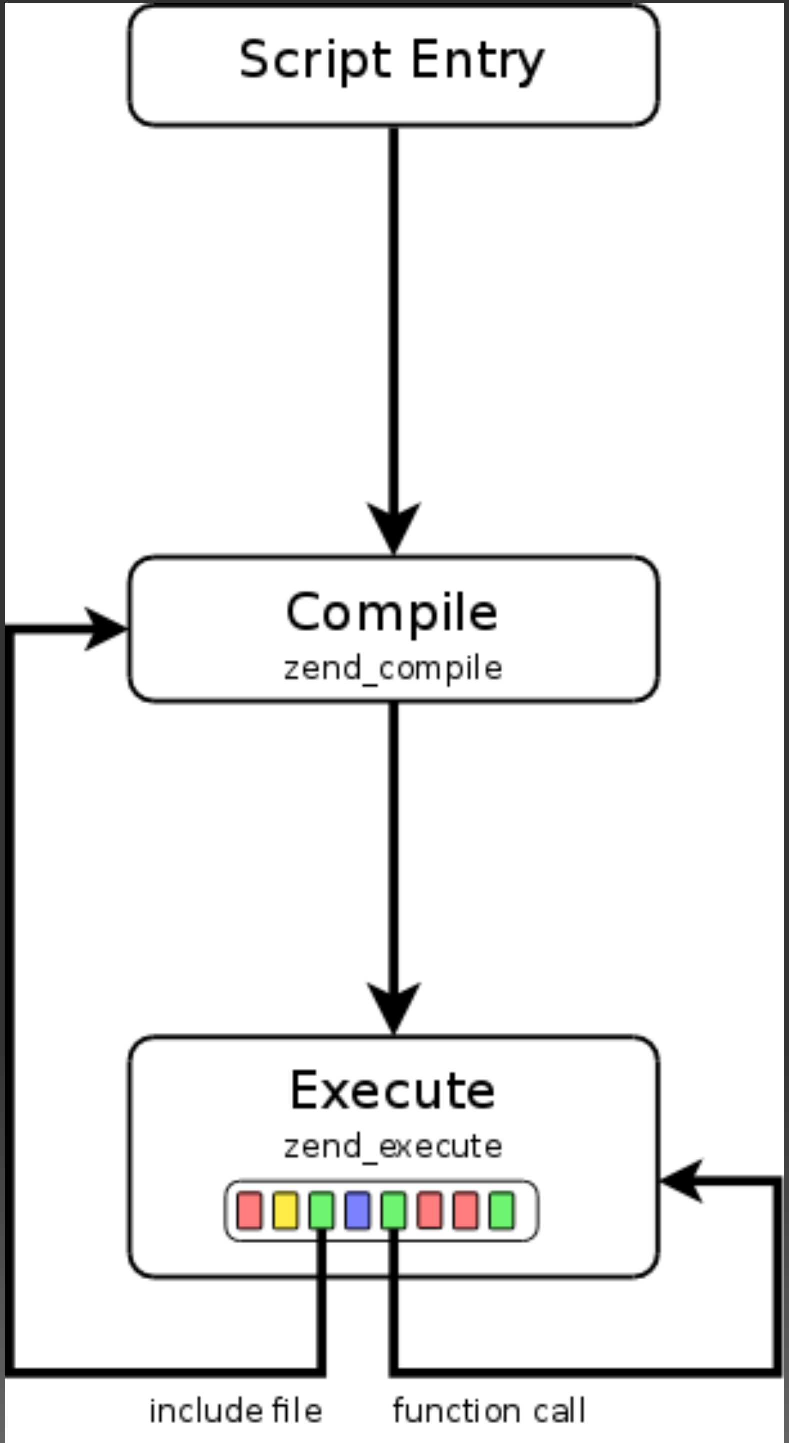
- Syntax-directed translation

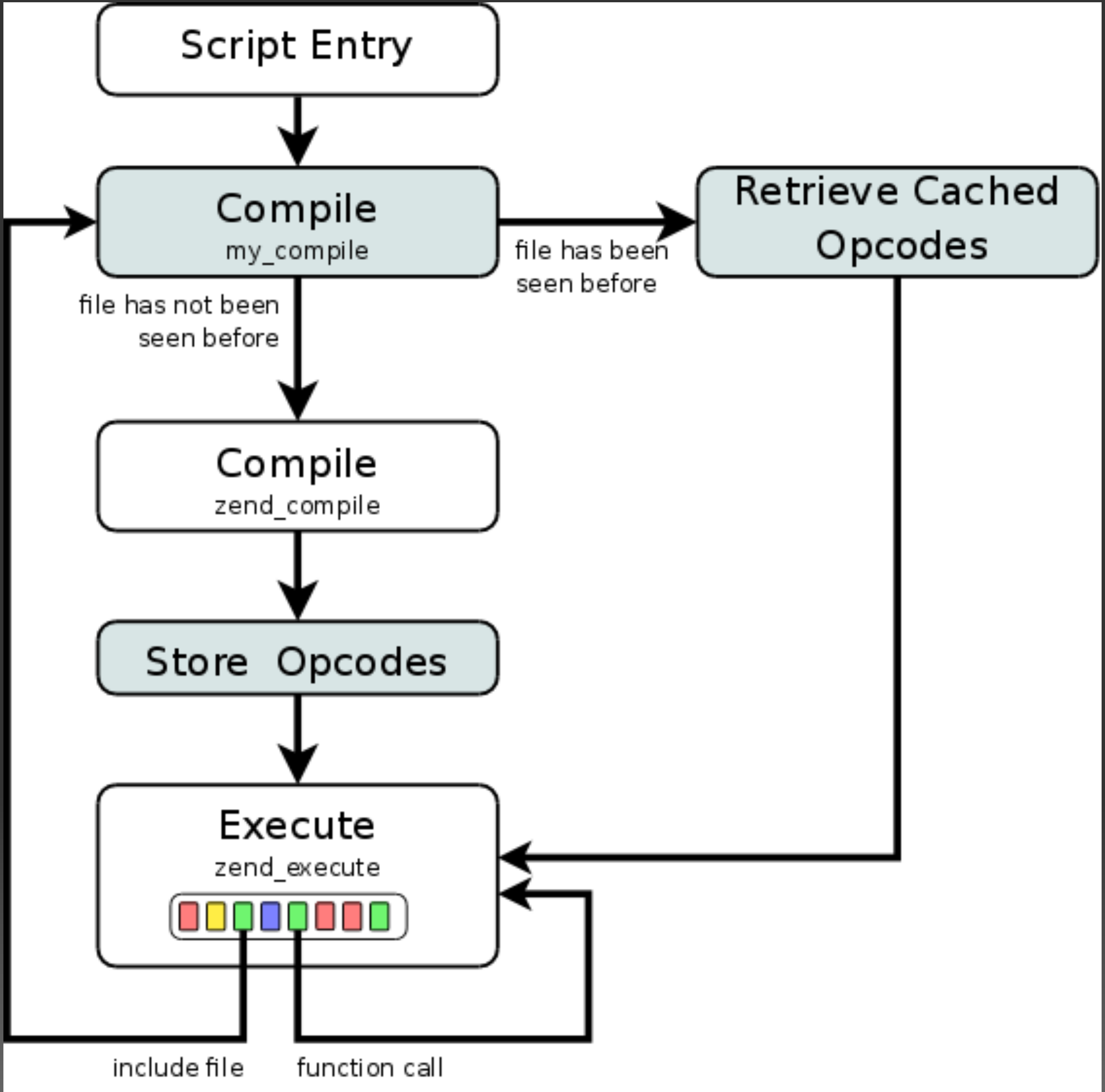
Overview of the Zend VM

- Syntax-directed translation
- Interprets bytecode

Overview of the Zend VM

- Syntax-directed translation
- Interprets bytecode
- No code optimizations


Script Entry
Compile
zend_compile
Execute
zend_execute
include file
function call


Script Entry
Compile
my_compile
Retrieve Cached Opcodes
file has been seen before
file has not been seen before
Compile
zend_compile
Store Opcodes
Execute
zend_execute
include file
function call

PHP bytecode

PHP bytecode

- Memory based (vs register or stack based)

PHP bytecode

- Memory based (vs register or stack based)
- No standard representation

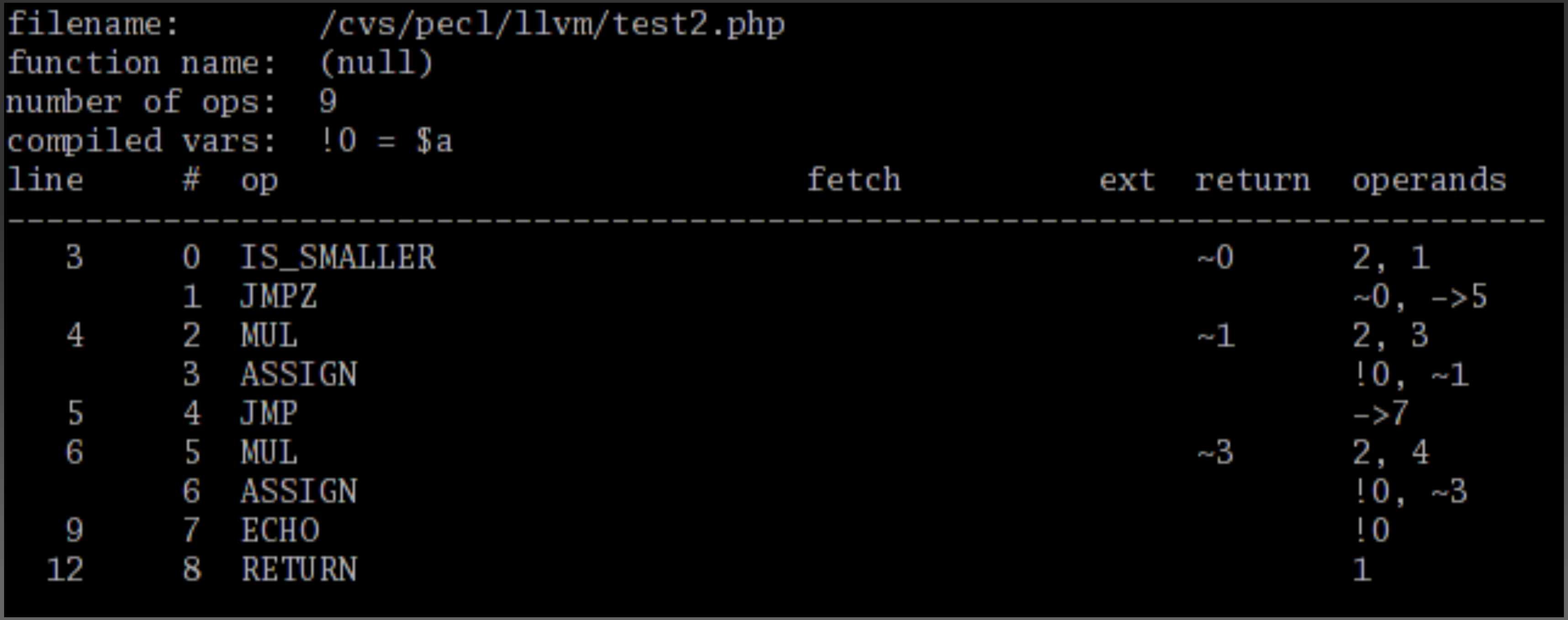
PHP bytecode

- Memory based (vs register or stack based)
- No standard representation
- Designed to be executed and discarded

PHP bytecode

- Memory based (vs register or stack based)
- No standard representation
- Designed to be executed and discarded
- Some information is not stored in bytecode (e.g. class definitions)

```
<?php
if (1 > 2)
        $a = 2 * 3;
else
        $a = 2 * 4;

echo $a;
?>
```



```
filename:       /cvs/pecl/llvm/test2.php
function name:  (null)
number of ops:  9
compiled vars:  !0 = $a
line     #  op                           fetch          ext  return  operands
-------------------------------------------------------------------------------
   3     0  IS_SMALLER                                       ~0      2, 1
         1  JMPZ                                                     ~0, ->5
   4     2  MUL                                              ~1      2, 3
         3  ASSIGN                                                   !0, ~1
   5     4  JMP                                                      ->7
   6     5  MUL                                              ~3      2, 4
         6  ASSIGN                                                   !0, ~3
   9     7  ECHO                                                     !0
  12     8  RETURN                                                   1
```

Design Rationale

Design Rationale

- Do not rewrite the whole VM from scratch

Design Rationale

- Do not rewrite the whole VM from scratch
- Have a proof-of-concept working ASAP

Design Rationale

- Do not rewrite the whole VM from scratch
- Have a proof-of-concept working ASAP
- Leave room for future optimizations

Implementation

Implementation

- Works as a Zend VM extension ("a speedup plugin")

Implementation

- Works as a Zend VM extension ("a speedup plugin")
- Hooks as the bytecode executor

Implementation

- Works as a Zend VM extension ("a speedup plugin")
- Hooks as the bytecode executor
- Updates the state of the VM

Implementation

- Works as a Zend VM extension ("a speedup plugin")
- Hooks as the bytecode executor
- Updates the state of the VM
- Can be used along with the old interpreter

Implementation #2

Implementation #2

- Offline compilation of Zend VM bytecode handlers to LLVM

Implementation #2

- Offline compilation of Zend VM bytecode handlers to LLVM
- Translation of bytecodes to handler calls

Implementation #2

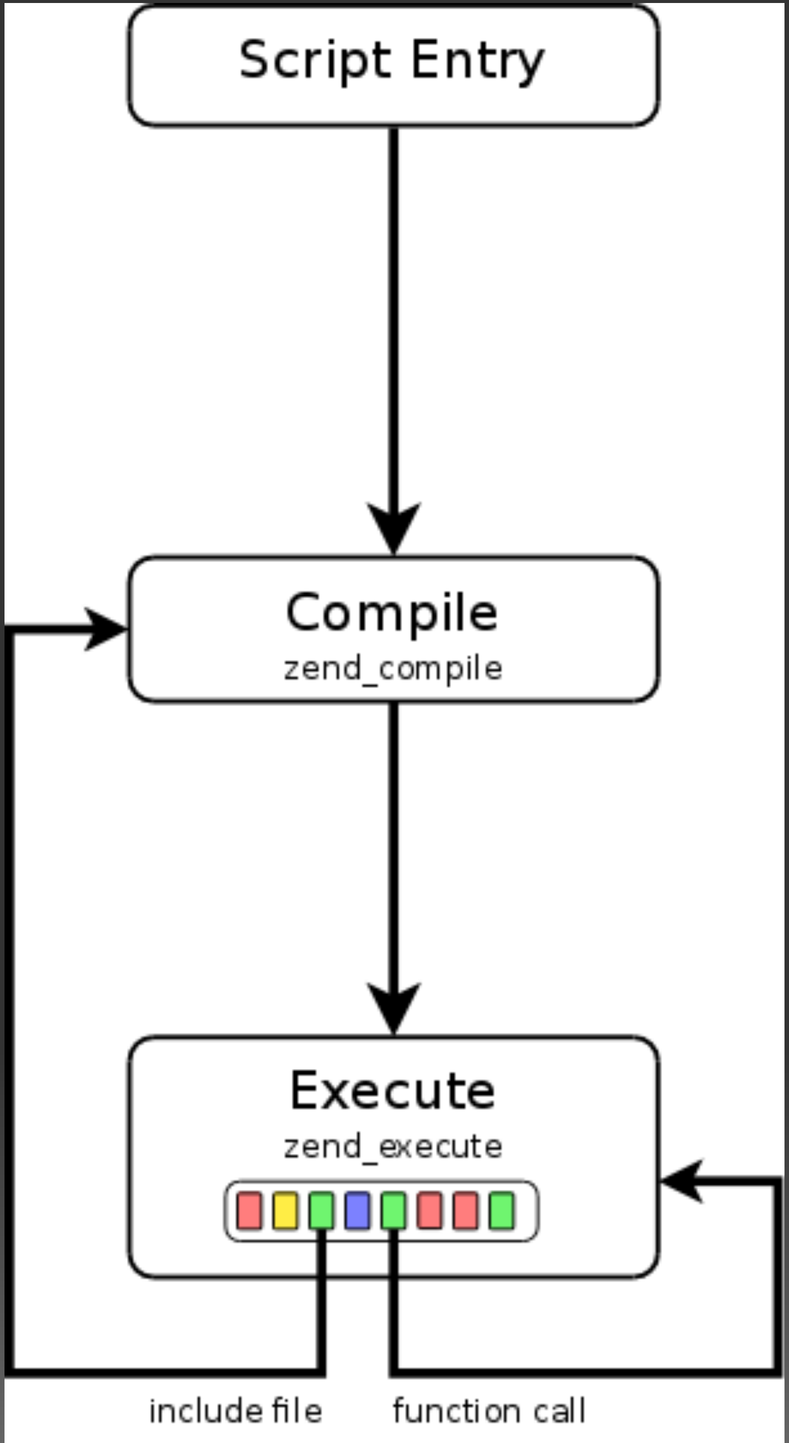
- Offline compilation of Zend VM bytecode handlers to LLVM
- Translation of bytecodes to handler calls
- JIT compilation of one function at a time

Implementation #2

- Offline compilation of Zend VM bytecode handlers to LLVM
- Translation of bytecodes to handler calls
- JIT compilation of one function at a time
- Performs simple optimizations (including inlining)

Implementation #2

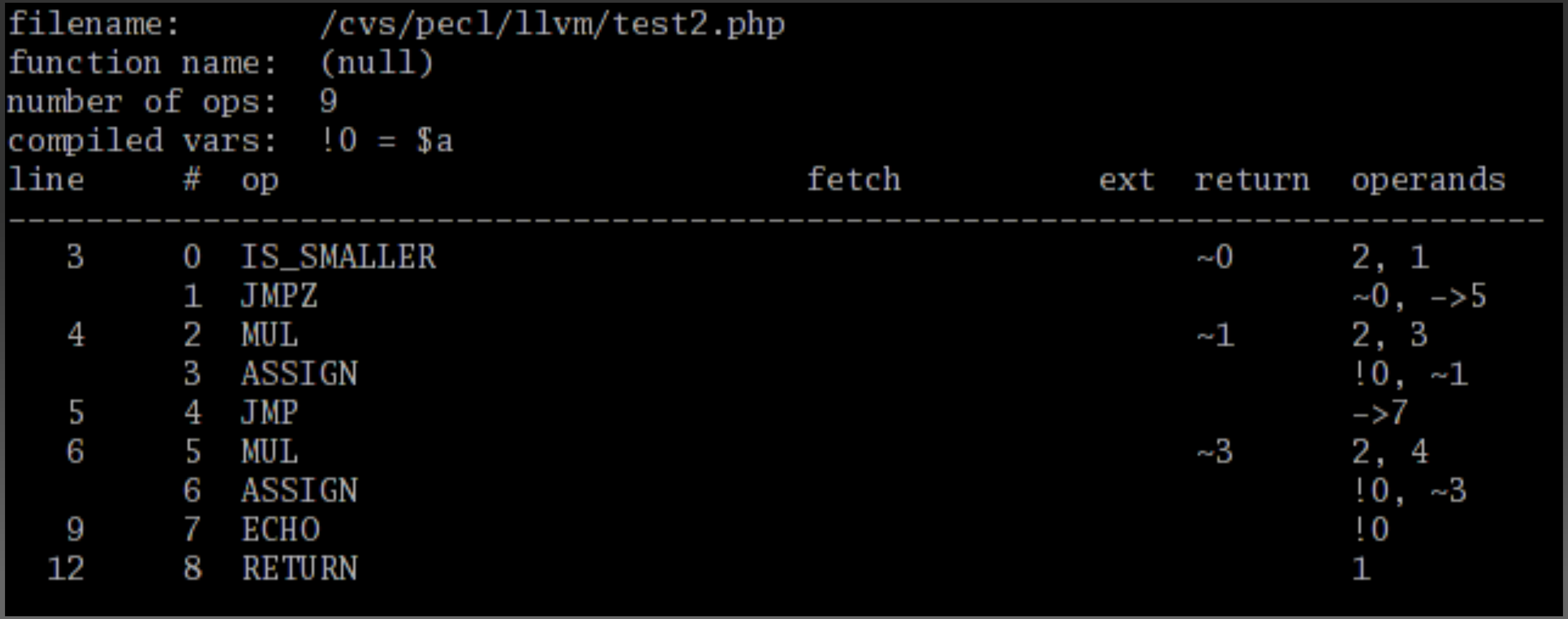
- Offline compilation of Zend VM bytecode handlers to LLVM
- Translation of bytecodes to handler calls
- JIT compilation of one function at a time
- Performs simple optimizations (including inlining)
- Uses a small runtime "library"


Script Entry
Compile
zend_compile
Execute
zend_execute
include file
function call

zend_execute()

```
while (1) {
    int ret;

    if ((ret = EX(opline)->handler(data)) > 0) {
        switch (ret) {
            ...
        }
    }
}
```

```
<?php
if (1 > 2)
        $a = 2 * 3;
else
        $a = 2 * 4;

echo $a;
?>
```



```
filename:       /cvs/pecl/llvm/test2.php
function name:  (null)
number of ops:  9
compiled vars:  !0 = $a
line     #  op                           fetch          ext  return  operands
-------------------------------------------------------------------------------
   3     0  IS_SMALLER                                       ~0      2, 1
         1  JMPZ                                                     ~0, ->5
   4     2  MUL                                              ~1      2, 3
         3  ASSIGN                                                   !0, ~1
   5     4  JMP                                                      ->7
   6     5  MUL                                              ~3      2, 4
         6  ASSIGN                                                   !0, ~3
   9     7  ECHO                                                     !0
  12     8  RETURN                                                   1
```

LLVM bitcode

```
op_block:

%execute_data = call @phpllvm_get_execute_data(%1)

%execute_result = call
@ZEND_IS_SMALLER_HANDLER(%execute_data)

switch i32 %execute_result, label %op_block1 [
    i32 1, label %pre_vm_return
    i32 2, label %pre_vm_enter
    i32 3, label %pre_vm_leave
]
```

LLVM bitcode

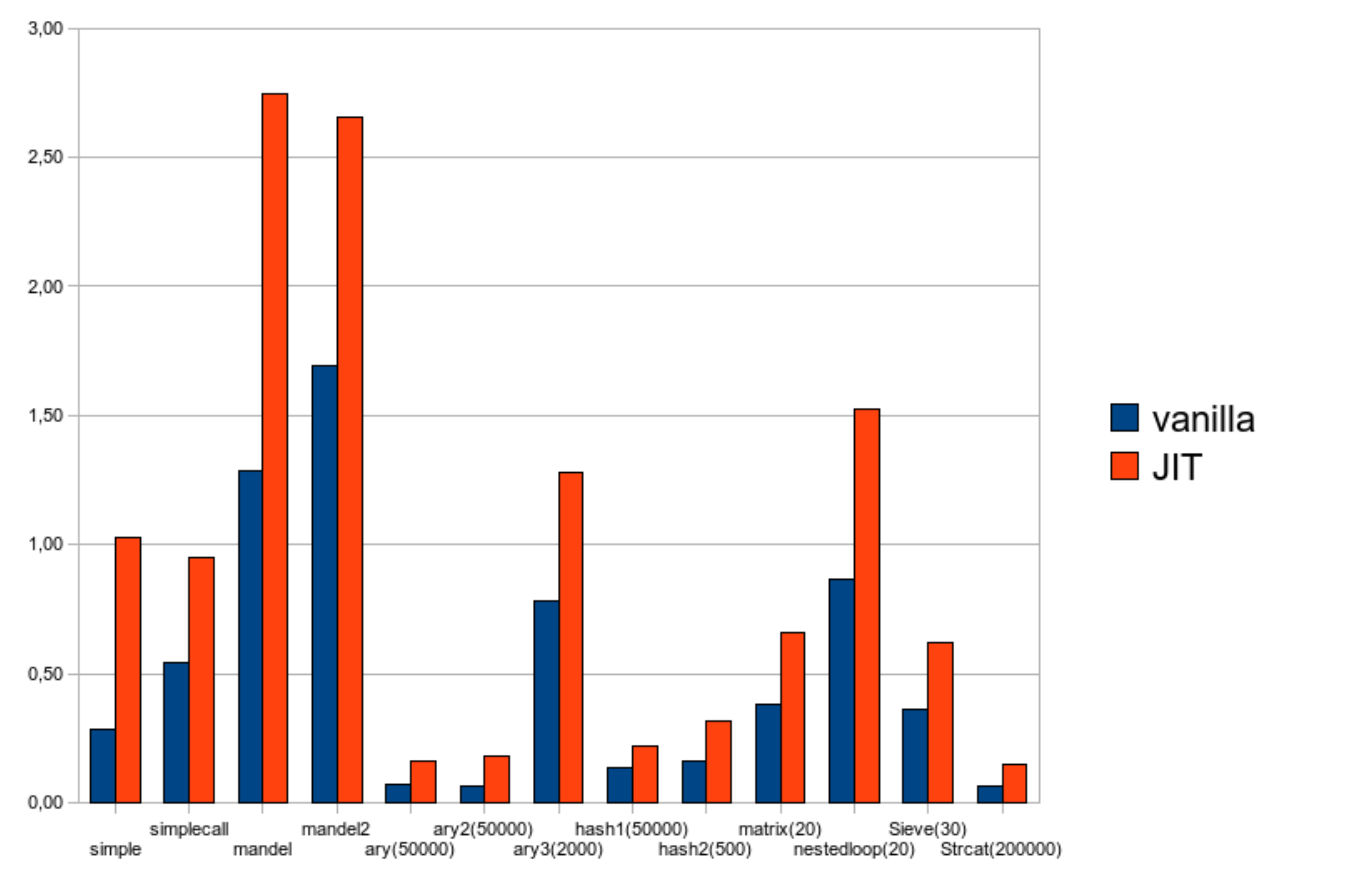
```
op_block1:

%execute_data = call @phpllvm_get_execute_data(%1)

%execute_result = call  @ZEND_JMPZ_HANDLER(%execute_data)

%current = call i32 @phpllvm_get_opline_number(%1)

switch i32 %current, label %ret [
    i32 5, label %op_block5
    i32 2, label %op_block2
]
```

Results of "Hello World"

- Vanilla: 0.03s
- JIT Debug: 2.5s
- JIT Release: 0.68s
- JIT Release+no asserts: 0.64s

Slowdown: 21x

Results


3,00
2,50
2,00
1,50
1,00
0,50
0,00
simple
simplecall
mandel
mandel2
ary(50000)
ary2(50000)
ary3(2000)
hash1(50000)
hash2(500)
matrix(20)
nestedloop(20)
Sieve(30)
Strcat(200000)
vanilla
JIT

Future Work

Future Work

- Compiled code caching and sharing

Future Work

- Compiled code caching and sharing
- Self-executable apps ("normal", GTK, etc..)

Future Work

- Compiled code caching and sharing
- Self-executable apps ("normal", GTK, etc..)
- Self-contained webapps (with e.g. Apache)

Future Work

- Compiled code caching and sharing
- Self-executable apps ("normal", GTK, etc..)
- Self-contained webapps (with e.g. Apache)
- Optimizations (lots of them :)

Questions?