

KEEP GETTING FASTER

THE NEXT GENERATION OF PHP



Laruence
PHP DEVELOPER



SELF INTRODUCTION

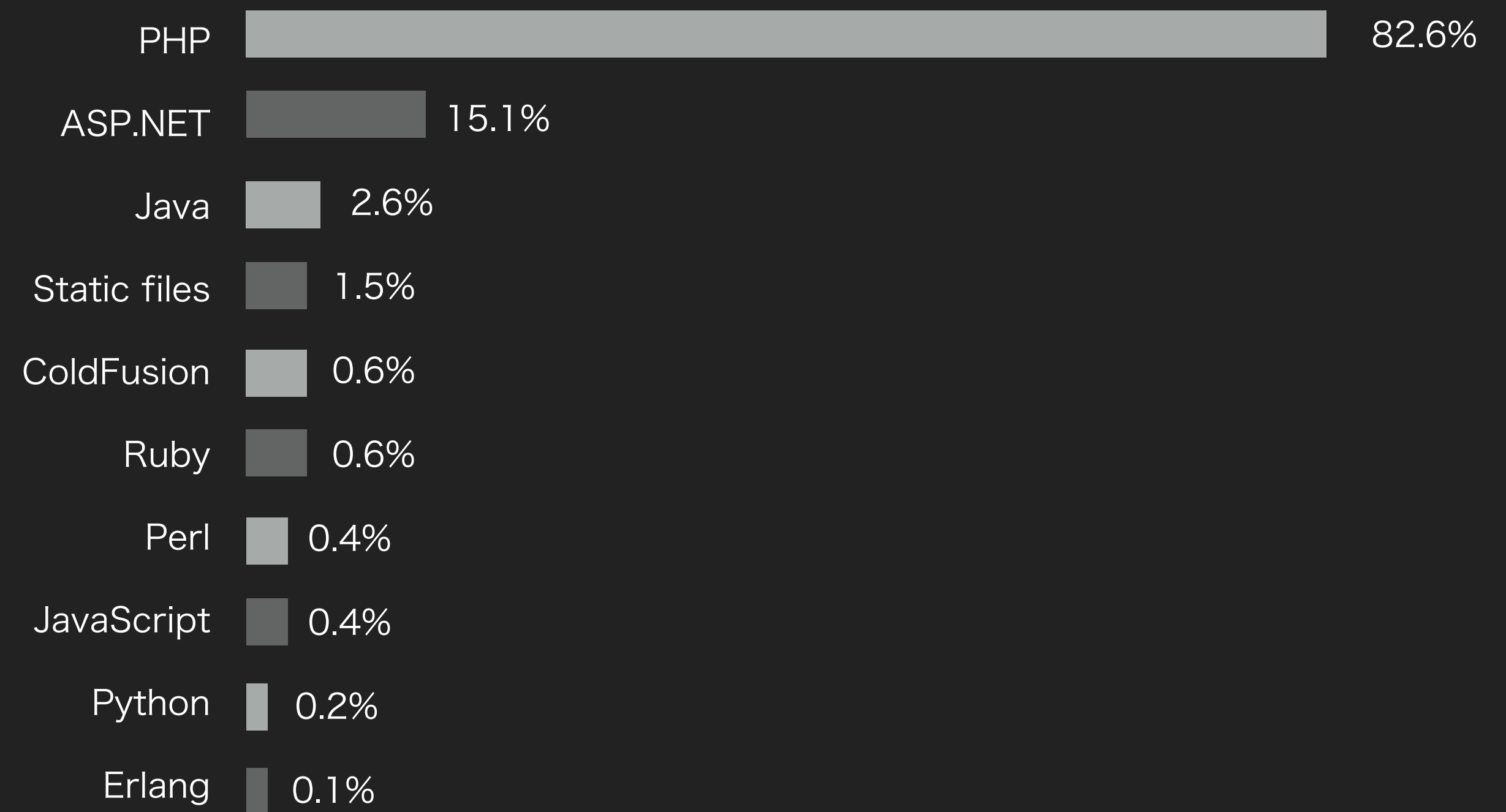
A PROGRAMMER

- Author Of Yaf, Yar, Yac, Yaconf, Taint Projects
- Maintainer Of Opcache, Msgpack Projects
- PHP Core Developer Since 2011
- Zend Consultant Since 2013
- PHP7 Core Developer
- Chief Software Architect At Lianjia Since 2015

PHP HISTORY

BORN FOR WEB

- Created In 1994 By Rasmus Lerdorf
- 20+ Years Programming Language
- Most Popular Web Service Program Language
- Php7 Is Released At 3 Dec 2015
- Latest Version Is PHP7.2

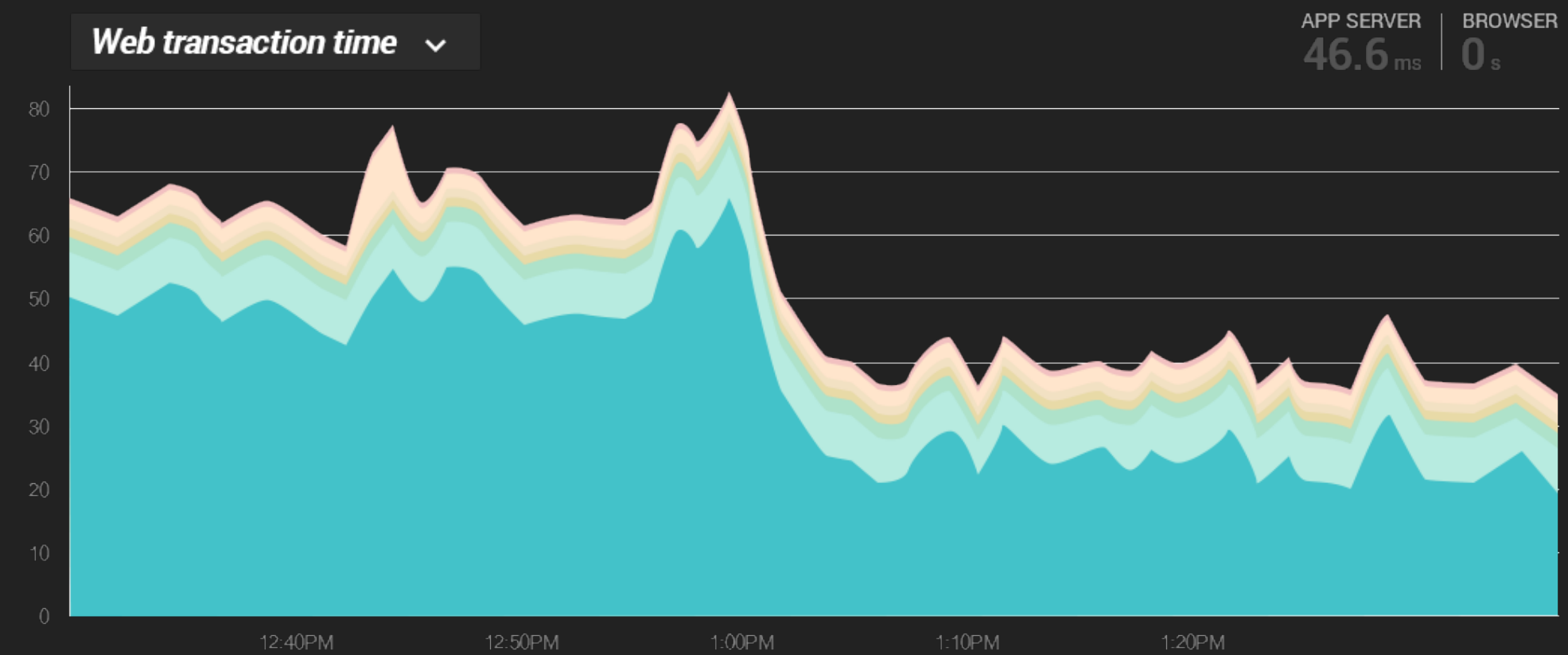
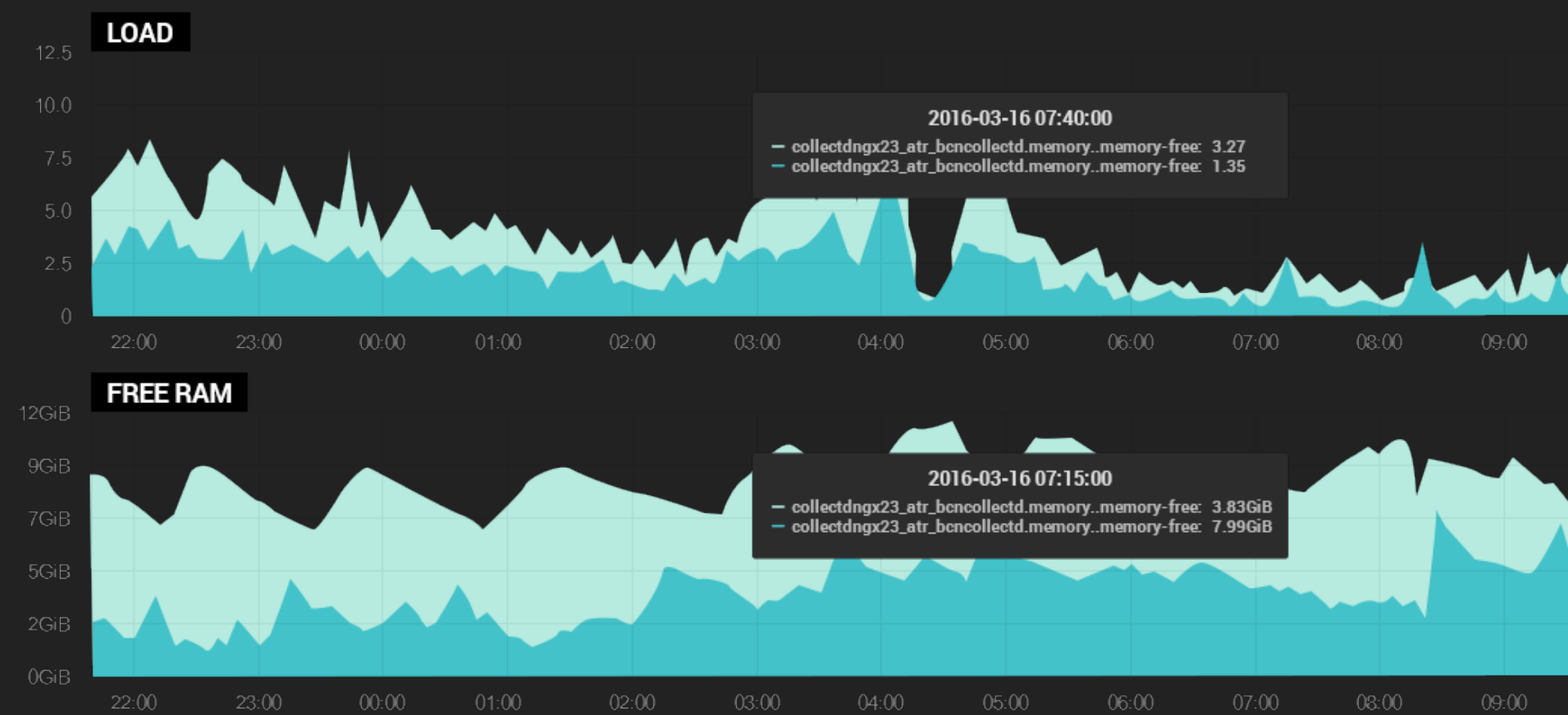


W3Techs.com, 7 June 2017

PHP7

THE FASTEST PHP

- The Biggest Performance Boost Version
- Over 100 % Performance Boosted
- Released Two Years Ago



PHP7.1

DATAFLOW ANALYSIS

- Static Single Assignment IR(SSA)
- Data Flow Analysis Optimization
- Type Inference System
 - Enhancement Of Range Inference
 - Enhancement Of Type Inference Using Pi-Node

```
function calc($a, $b) {  
    $a = $a * 2 % 1000;  
    $b = $b * 3 % 1000;  
    return $a + $b;  
}
```

```
function calc($a1, $b2) { // $a1: [ANY], $b2: [ANY]  
    $T3 = $a1 * 2;         // $T3: [LONG, DOUBLE]  
    $a4 = $T3 % 1000;      // $a4: [LONG]  
    $T5 = $b2 * 3;         // $T5: [LONG, DOUBLE]  
    $b6 = $T5 % 1000;      // $b6: [LONG]  
    $T7 = $a4 + $b6;       // $T7: [LONG, DOUBLE]  
    return $T7;  
}
```

```
php -d opcache.opt_debug_level=-1 calc.php
```

PHP7.1

TYPE SPECIAL OPCODE HANDLER

- Type Special Opcode Handler
 - Avoiding Type-Check In Runtime
 - Make Fast-Path Faster

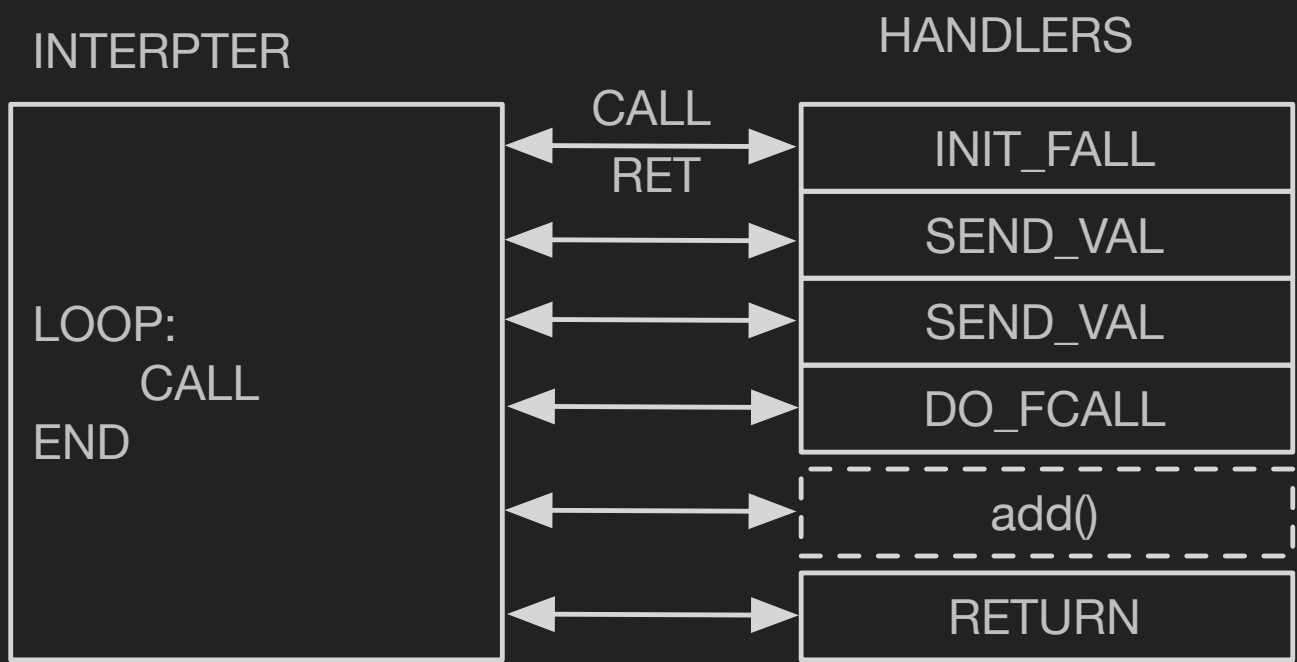
```
function calc($a1, $b2) {    //$a1: [ANY], $b2: [ANY]
    $T3 = $a1 * 2;           //$T3: [LONG, DOUBLE]
    $a4 = $T3 % 1000;        //$a4: [LONG]
    $T5 = $b2 * 3;           //$T5: [LONG, DOUBLE]
    $b6 = $T5 % 1000;        //$b6: [LONG]
    $T7 = $a4 + $b6;         // ZEND_ADD_LONG
    return $T7;
}
```

PHP7.2

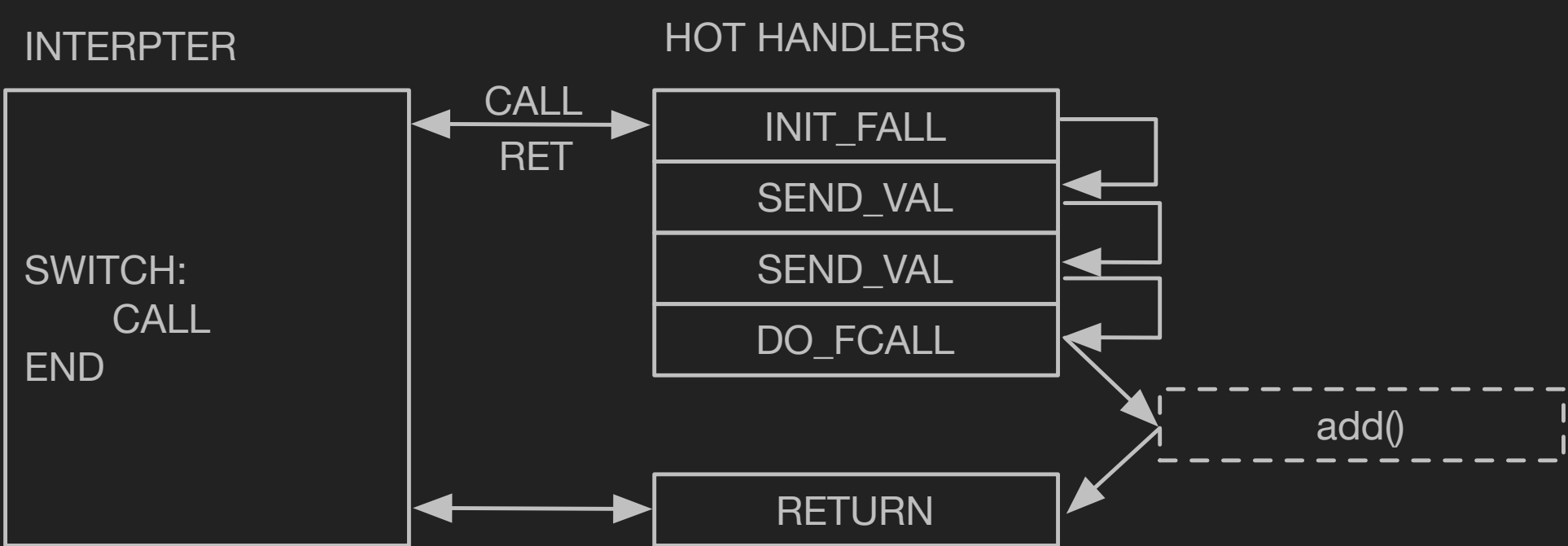
HYBRID VM

- Hybrid Vm Engine (Default Vm)
- Hybrid With Call+Goto
- Based On GCC Computed Goto^[1]
- Less Instruction And Friendly For Branch Prediction

```
function add($a, $b) {  
    return $a + $b;  
}  
  
add(1, 2);
```



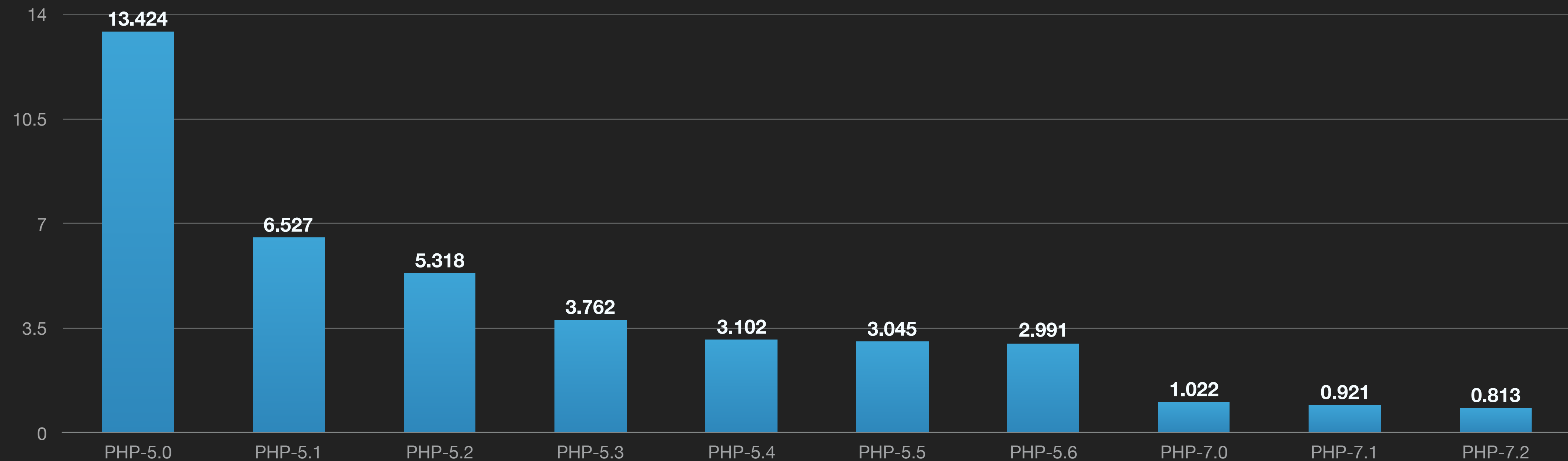
CALL VM



HIBRID VM

PERFORMANCE EVOLUTION

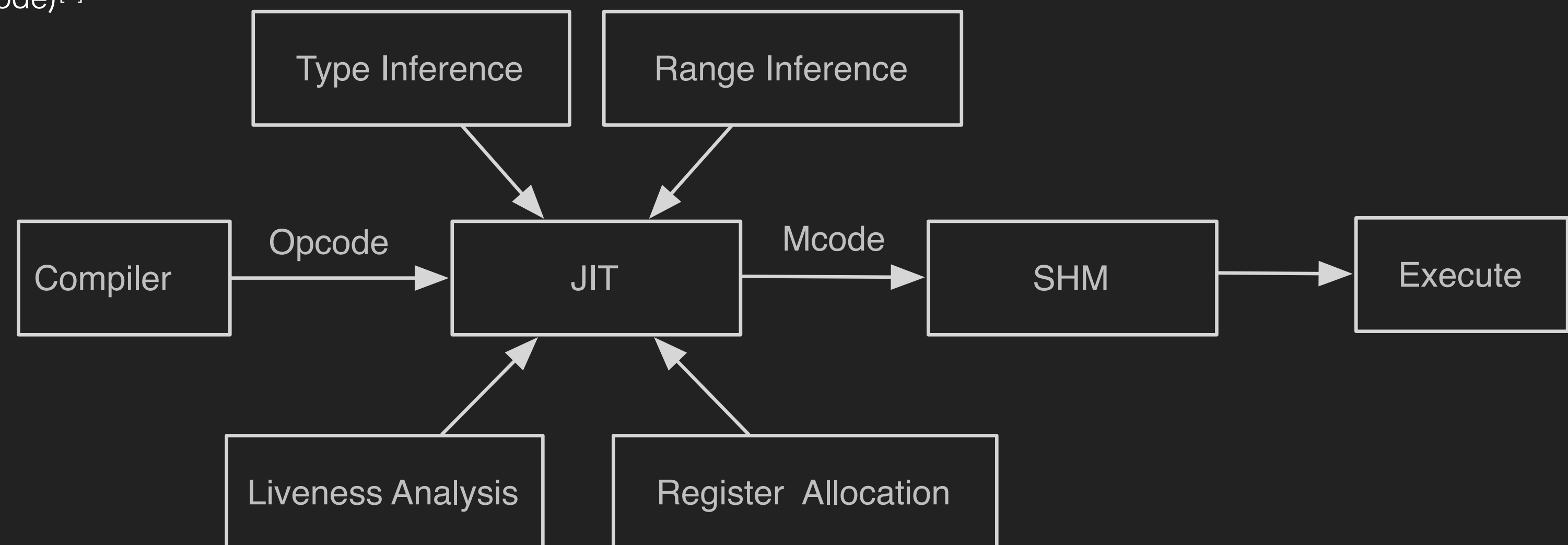
KEEP GETTING FASTER



JUST-IN-TIME COMPILER

BRAND NEW WAY

- JIT For PHP7
- Require Gcc 4.8 +(Global Register)
- X86-32/64 Supported
- Using Dynasm To Emitting Machine Codes(Mcode)^[2]
- Already Opened Source^[3]



JUST-IN-TIME COMPILER

CONFIGURATION(TEMPORARY)

opcache.jit=1 2 0 5

JIT Strategy
↓
Use AVX
↓
1

2
↑
Register alloc

0
↑
JIT Level

5
↑
JIT Level

0: JIT functions on load

1: JIT functions on execute

2: JIT most frequently called functions on first request

3: JIT functions after N calls or loop iterations

4: JIT functions with @jit annotation

0: JIT off

1: Minimal optimization

2: Basic optimization

3: optimize based on type-inference

4: optimize based on type-Inference and call-tree

5: optimize based on type-Inference and inner-procedure analyses

JUST-IN-TIME COMPILER

BASIC OPTIMIZATION

- Inline Opcodes Dispatch
- opcache.jit=1201

```
function calc($a, $b) {  
    $a = $a * 2 % 1000;  
    $b = $b * 3 % 1000;  
    return $a + $b;  
}
```

#0	RECV	1		\$a
#1	RECV	2		\$b
#2	MUL	\$a	2	~1
#3	MOD	~1	1000	~0
#4	ASSIGN	\$a	~0	
#5	MUL	\$b	3	~1
#6	MOD	~1	1000	~0
#7	ASSIGN	\$b	~0	
#8	ADD	\$a	\$b	~0
#9	RETURN	~0		

phpdbg -p* calc.php

```
call ZEND_MUL_SPEC_CV_CONST_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_MOD_SPEC_TMPVAR_CONST_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_ASSIGN_SPEC_CV_TMP_RETVAL_UNUSED_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_MUL_SPEC_CV_CONST_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_MOD_SPEC_TMPVAR_CONST_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_ASSIGN_SPEC_CV_TMP_RETVAL_UNUSED_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler  
call ZEND_ADD_LONG_SPEC_TMPVARCV_TMPVARCV_HANDLER  
cmp $0x0, EG(exception)  
jnz JIT$$exception_handler
```

php -dopcache.jit_debug=1 calc.php



JUST-IN-TIME COMPILER

TYPE CHECK AVOIDING

- PHP Is Weak-Type Language
- Lots Of Type-Checks In Runtime

```
function calc($a, $b) {  
    $a = $a * 2 % 1000;  
    $b = $b * 3 % 1000;  
    return $a + $b;  
}
```

#0	RECV	1		\$a
#1	RECV	2		\$b
#2	MUL	\$a	2	~1
#3	MOD	~1	1000	~0
#4	ASSIGN	\$a	~0	
#5	MUL	\$b	3	~1
#6	MOD	~1	1000	~0
#7	ASSIGN	\$b	~0	
#8	ADD	\$a	\$b	~0
#9	RETURN	~0		

phpdbg -p* calc.php

```
ZEND_VM_HANDLER(1, ZEND_ADD, CONST|TMPVAR|CV, CONST|TMPVAR|CV)  
{  
    USE_OPLINE  
    zend_free_op free_op1, free_op2;  
    zval *op1, *op2, *result;  
  
    op1 = GET_OP1_ZVAL_PTR_UNDEF(BP_VAR_R);  
    op2 = GET_OP2_ZVAL_PTR_UNDEF(BP_VAR_R);  
    if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_LONG)) {  
        if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {  
            result = EX_VAR(opline->result.var);  
            fast_long_add_function(result, op1, op2);  
            ZEND_VM_NEXT_OPCODE();  
        } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {  
            result = EX_VAR(opline->result.var);  
            ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));  
            ZEND_VM_NEXT_OPCODE();  
        }  
    } else if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_DOUBLE)) {  
        if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {  
            result = EX_VAR(opline->result.var);  
            ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));  
            ZEND_VM_NEXT_OPCODE();  
        } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {  
            result = EX_VAR(opline->result.var);  
            ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));  
            ZEND_VM_NEXT_OPCODE();  
        }  
    }  
  
    SAVE_OPLINE();  
    if (OP1_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op1) == IS_UNDEF)) {  
        op1 = GET_OP1_UNDEF_CV(op1, BP_VAR_R);  
    }  
    if (OP2_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op2) == IS_UNDEF)) {  
        op2 = GET_OP2_UNDEF_CV(op2, BP_VAR_R);  
    }  
    add_function(EX_VAR(opline->result.var), op1, op2);  
    FREE_OP1();  
    FREE_OP2();  
    ZEND_VM_NEXT_OPCODE_CHECK_EXCEPTION();  
}
```



JUST-IN-TIME COMPILER

TYPE CHECK AVOIDING

· Only Run The Necessary Codes

· opcache.jit=1205

```
function calc($a1, $b2) { // $a1: [ANY], $b2: [ANY]
    $T3 = $a1 * 2; // $T3: [LONG, DOUBLE]
    $a4 = $T3 % 1000; // $a4: [LONG]
    $T5 = $b2 * 3; // $T5: [LONG, DOUBLE]
    $b6 = $T5 % 1000; // $b6: [LONG]
    $T7 = $a4 + $b6; // $T7: [LONG, DOUBLE]
    return $T7;
}
```

php -d opcache.opt_debug_level=-1 calc.php

#0	RECV	1		\$a
#1	RECV	2		\$b
#2	MUL	\$a	2	~1
#3	MOD	~1	1000	~0
#4	ASSIGN	\$a	~0	
#5	MUL	\$b	3	~1
#6	MOD	~1	1000	~0
#7	ASSIGN	\$b	~0	
#8	ADD	\$a	\$b	~0
#9	RETURN	~0		

phpdbg -p* calc.php

```
mov 0x50(%r14), %rax
add 0x60(%r14), %rax
jo .L27
mov %rax, 0x70(%r14)
mov $0x4, 0x78(%r14)
```

php -d opcache.jit_debug=1 calc.php



JUST-IN-TIME COMPILER

REGISTERS ALLOCATION

- Registers Allocation For Long/Double
- Linear Scan Register Allocation(Toggled Between Opcache.Jit=1205 And1005)^[4]

```
function loop() {  
    for ($i = 0; $i < 1000; $i++) {  
    }  
}
```

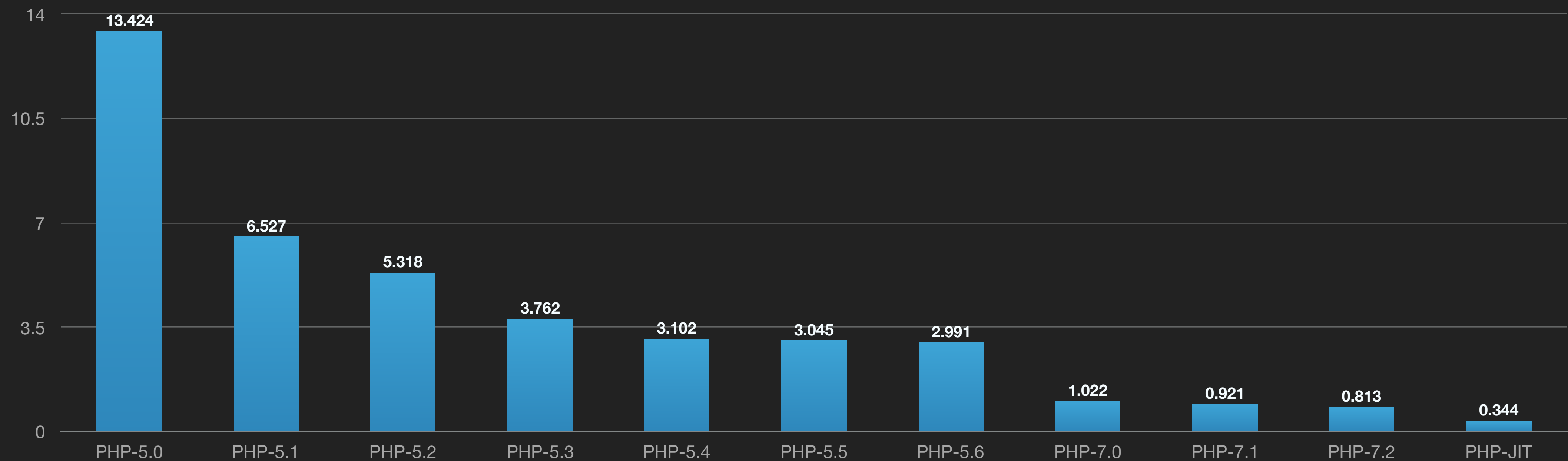
```
mov $0x0, 0x50(%r14)      xor %rdx, %rdx  
mov $0x4, 0x58(%r14)      jmp .L2  
jmp .L2  
.L1:                      add $0x1, %rdx  
add $0x1, 0x50(%r14)      .L2:  
.L2:                      cmp $0x0, EG(vm_interrupt)  
cmp $0x0, EG(vm_interrupt) jnz JIT$$interrupt_handler  
jnz JIT$$interrupt_handler cmp $0x3e8, %rdx  
cmp $0x3e8, 0x50(%r14)    j1 .L1  
j1 .L1
```

php -dopcache.jit_debug=1 loop.php

JUST-IN-TIME COMPILER

BENCHMARK

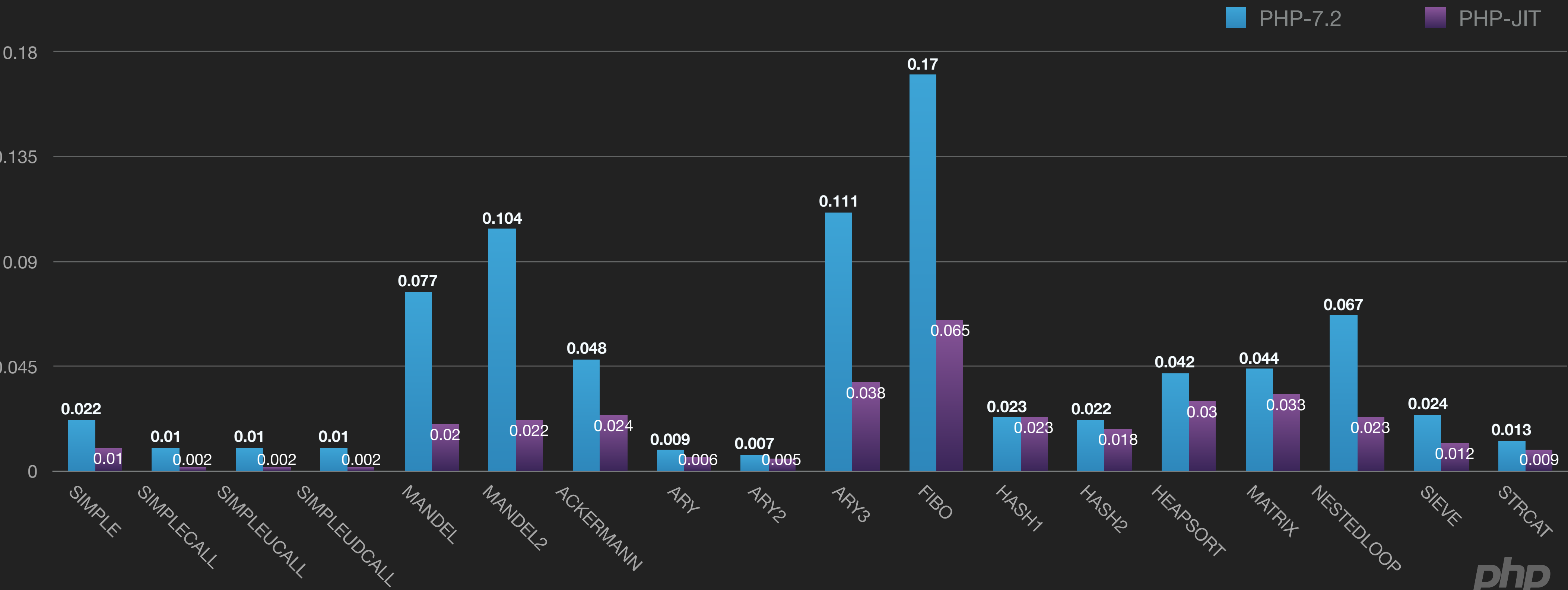
· Over 100% Improvement In Bench VS PHP-7.2



JUST-IN-TIME COMPILER

BENCHMARK

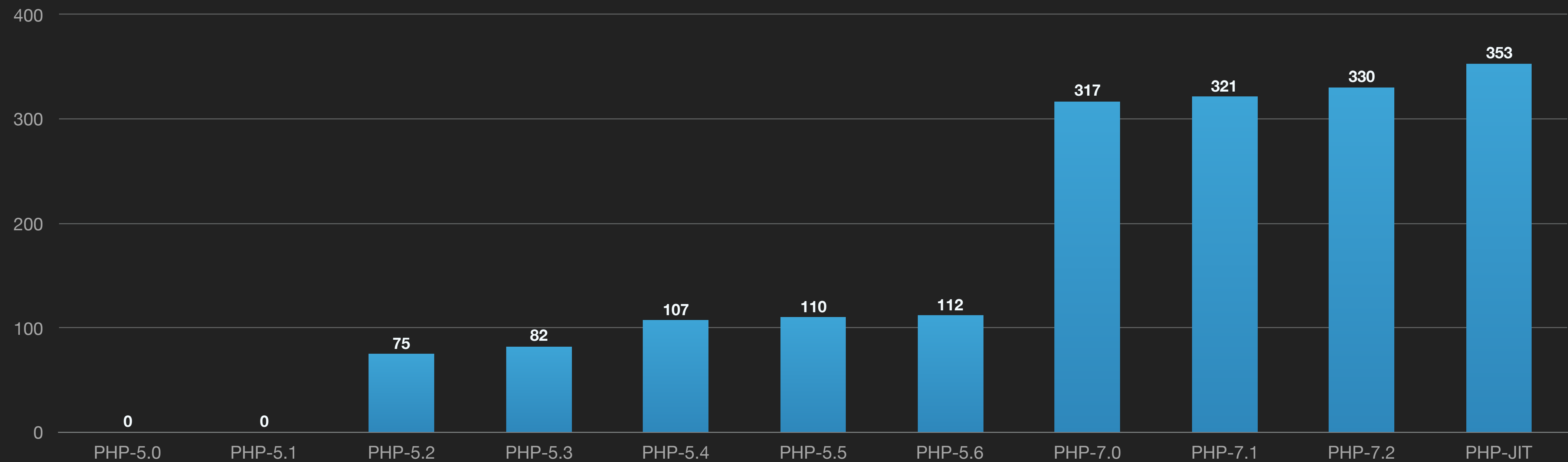
· Benchmark Details VS PHP-7.2



JUST-IN-TIME COMPILER

WORDPRESS HOMEPAGE BENCHMARK

· 7% Qps Improvement In Wordpress Homepage VS PHP-7.2



JIT CONSTRAINT

PSEUDO-MAIN SCOPE

- Volatile Variable In Main Scope
- We Don't Optimize Codes In Main Scope

```
set_error_handler(function() {  
    $GLOBALS['a'] = "str";  
});
```

```
$a = 1;  
$b = $c;
```

JIT CONSTRAINT

DYNAMIC SCOPE INTROSPECTION

- Dynamic Scope Modification
- Can Not Optimize Codes Which Include Such Usage

```
function example() {  
    $a = 1;  
    $b = "a";  
    ${ $b } = "str";  
}
```

```
function example() {  
    $a = 1;  
    extract(array("a" => "str"));  
}
```

JIT CONSTRAINT

REFERENCE

- Reference Arguments Passing Can Only Be See In Callee-Side

```
function fn(&$val) {  
    $val = "str";  
}
```

```
function example() {  
    $a = 1;  
    fn($a);  
}
```

JIT CONSTRAINT

INDEPENDENT SCRIPTS

- Inter-Script Optimization Is Impossible

```
if ($_GET["first"]) {  
    include "a.php";  
} else {  
    include "b.php";  
}
```

```
function example() {  
    $a = fn();  
}
```

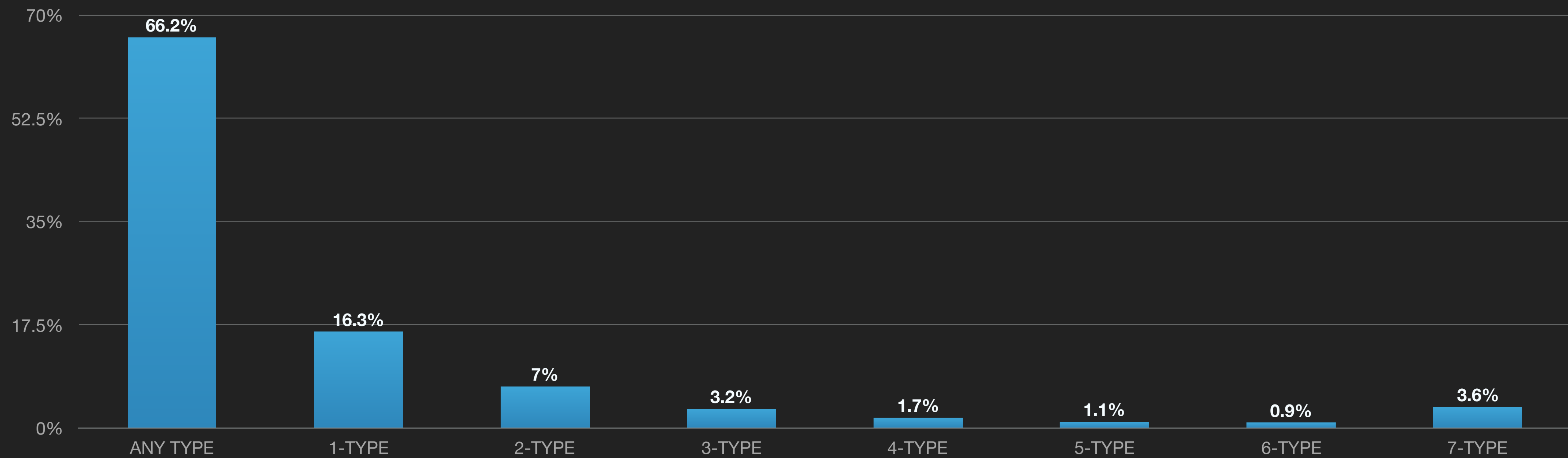
```
function fn() {  
    return "str";  
}
```

```
function fn() {  
    return array();  
}
```

JIT CONSTRAINT

THATS WHY

· Wordpress Homepage Type Inference Result



JIT CONSTRAINT

THATS WHY

- Visible Improvement On Old Applications
- Significant Improvement On Well-Written Codes
- Type Inference Plays Very Important Role In Jit
- More Type Info == More Performance Improvement
- Use Type-Hints If You Could

```
function add(int $a, int $b) :int {  
  
}
```

LINKS

- Computed GOTO: <https://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Labels-as-Values.html>
- The Unofficial DynAsm Documentation: <http://corsix.github.io/dynasm-doc/index.html>
- Zend JIT Repo: <https://github.com/zendtech/php-src>
- Linear Scan Register Allocation on SSA Form: <http://www.christianwimmer.at/Publications/Wimmer10a/Wimmer10a.pdf>

THANKS

PHP: KEEP GETTING FASTER

