Code Generation in PHP

c9s

Yo-An Lin
@c9s

The Problems

Web Frameworks have many conditions for different environment.

And many dynamic mechanisms

Framework conditions

- Decide which statements to be run in production / development.
- Dynamically setter/getter dispatching in ORM (keys can't be analyzed)
- Check which implementation is supported. (e.g. extensions, PHP VM versions....)

As the framework is getting bigger and bigger, the more conditions will need to be added into the application.

1. Detecting Environment in Frameworks.

Detecting Environment

```
<?php
$environment = $_ENV['PHIFTY_ENV'];
if ($environment === "dev") {
    // do something for development env
} else if ($environment === "testing") {
    // do something for testing env
} else if ($environment === "production") {
    // do something for production env
}
```

Detecting Environment

```php
<?php
if ($environment === "dev") {
    $event->bind("before_route", function() { /* ... */ });
    $event->bind("finalize", function() { /* ... */ });
} else if ($environment === "production") {
    $event->bind("before_route", function() { /* ... */ });
    $event->bind("finalize", function() { /* ... */ });
}
```

Detecting Environment

```php
<?php
if ($environment == "dev") {
    require "environment/dev.php";
} else if ($environment == "production") {
    require "environment/production.php";
}
```

2. Checking Implementations

Checking Implementation

```php
<?php
use Symfony\Component\Yaml\Dumper;

function encode($data) {
    if (extension_loaded('yaml')) {
        return yaml_emit($data);
    }

    // fallback to pure PHP implementation
    $dumper = new Dumper();
    return $dumper->dump($array);
}
```

3. Integrating Config Values

Integration Config Values

```php
<?php
if (extension_loaded('mongo')) {
    $container->mongo = function() use ($someConfigArray) {
        if (isset($someConfigArray['mongo_host'])) {
            return new MongoClient($someConfigArray['mongo_host']);
        }
        return new MongoClient('....');
    };
}
```

4. Magic Setters/Getters

Magic Setters/Getters

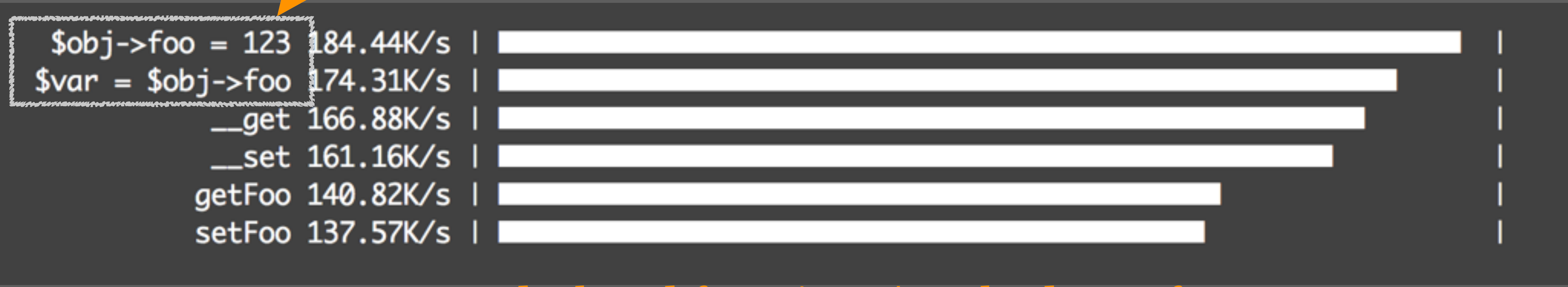
```
<?php

class MyArray
{
    protected $data = [];

    public function __set($key, $value)
    {
        $this->data[ $key ] = $value;
    }

    public function __
    {
        return $this->
    }
}
```

CAN'T BE AUTO-COMPLETED
IF WE'VE KNOWN THE
KEYS DEFINED IN SCHEMA

Magic Setters/Getters

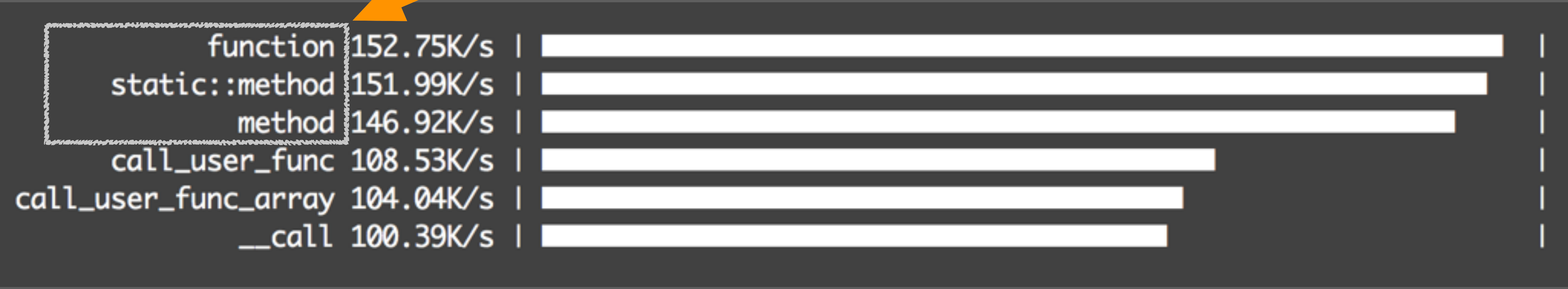
declared properties are faster

PHP 5.6.10



```
  $obj->foo = 123 184.44K/s |                                        |
  $var = $obj->foo 174.31K/s |                                       |
             __get 166.88K/s |                                       |
             __set 161.16K/s |                                       |
            getFoo 140.82K/s |                                       |
            setFoo 137.57K/s |                                       |
```

declared functions/methods are faster



```
          function 152.75K/s |                                       |
    static::method 151.99K/s |                                       |
            method 146.92K/s |                                       |
    call_user_func 108.53K/s |                                       |
call_user_func_array 104.04K/s |                                     |
            __call 100.39K/s |                                       |
```

Magic Setters/Getters

```php
<?php
class Foo
{
    protected $name;

    protected $price;

}
```

```php
<?php
class Foo
{
    protected $name;

    protected $price;

    public function getName()
    {
        return $this->name;
    }

    public function getPrice()
    {
        return $this->price;
    }
}
```

Doctrine can generates getter/setter methods for entities.

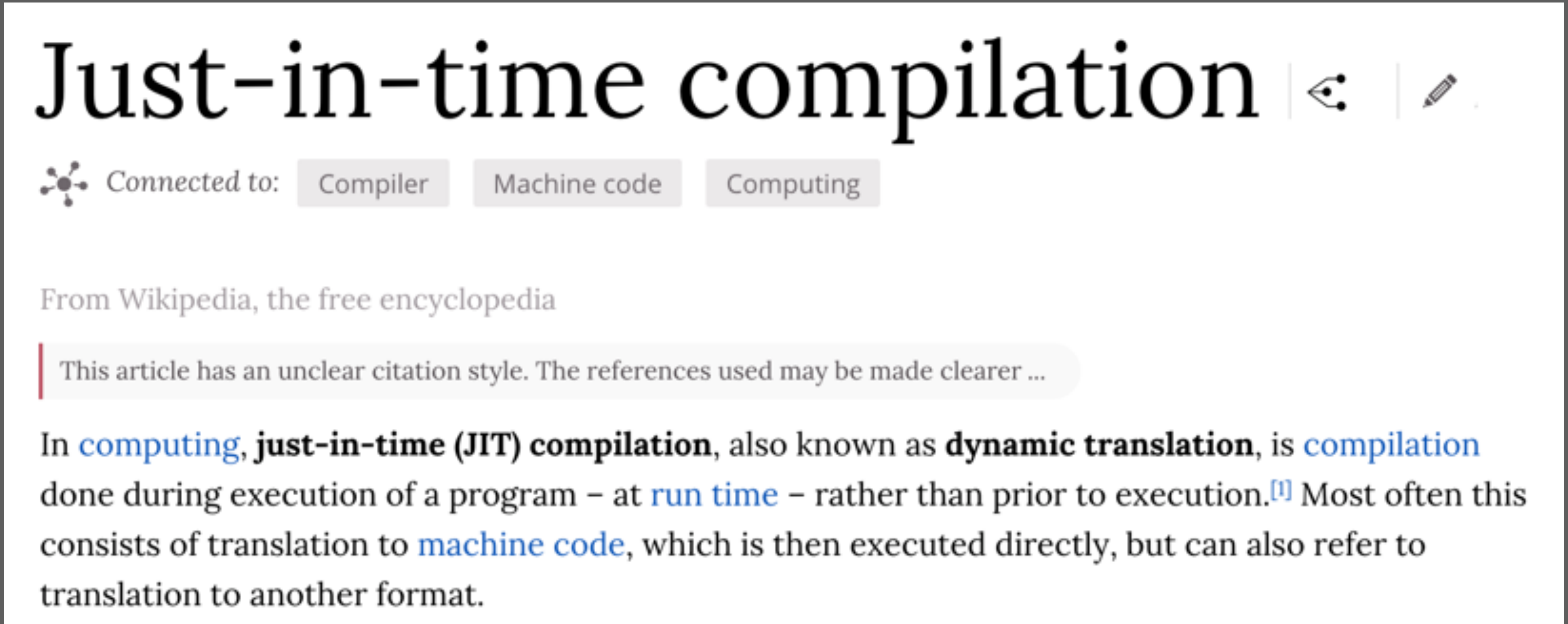
Types of Code Generation

Types of Code Generation

- Low Level Code Generation: JIT (Just-in-time compiler)
- High Level Code Generation: PHP to PHP, reducing runtime costs.

Low Level Code Generation

JIT (Just-in-time compilation)

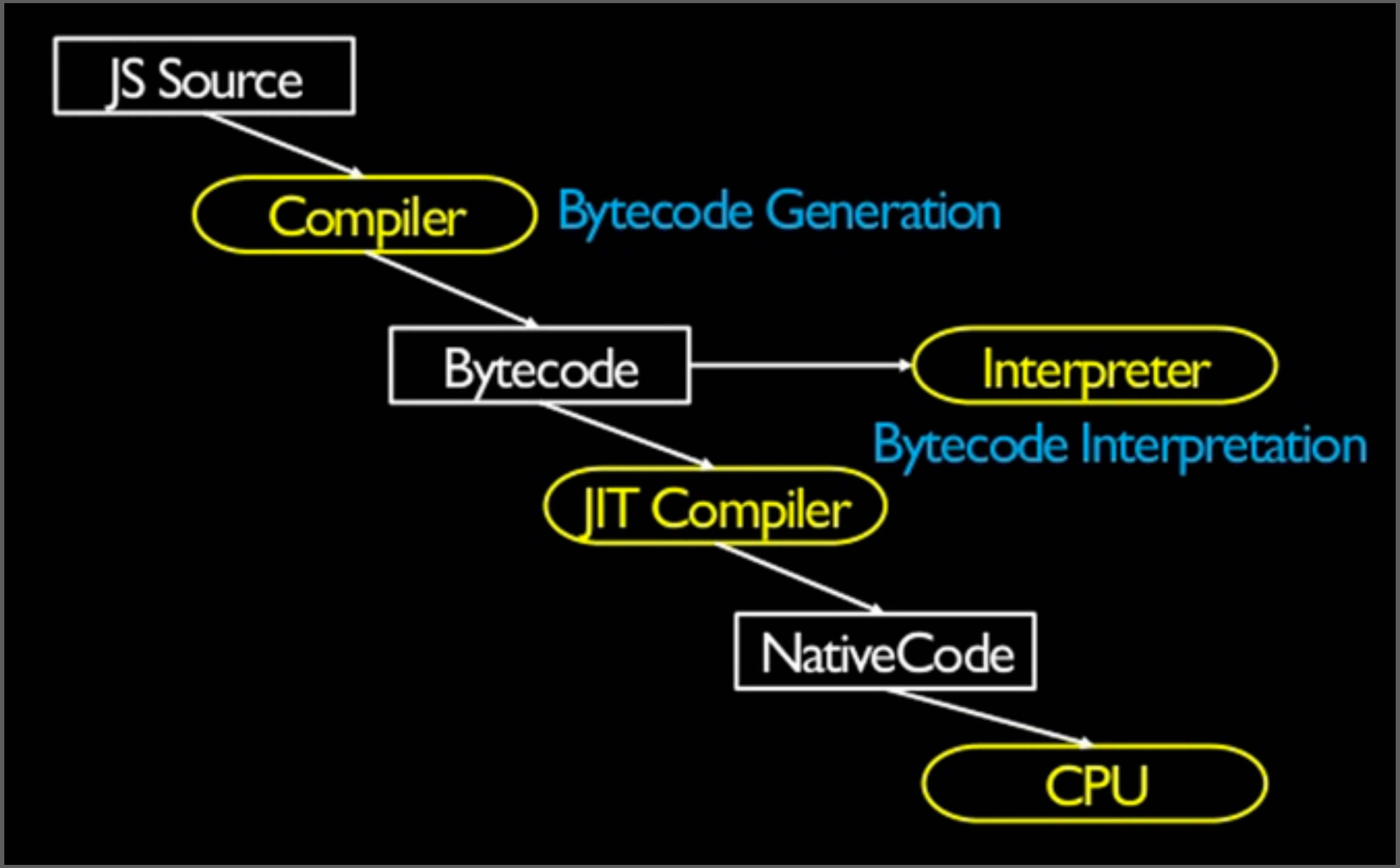


Just-in-time compilation

From Wikipedia, the free encyclopedia

This article has an unclear citation style. The references used may be made clearer ...

In computing, **just-in-time (JIT) compilation**, also known as **dynamic translation**, is compilation done during execution of a program – at run time – rather than prior to execution.[1] Most often this consists of translation to machine code, which is then executed directly, but can also refer to translation to another format.



JS Source

Compiler

Bytecode Generation

Bytecode

Interpreter

Bytecode Interpretation

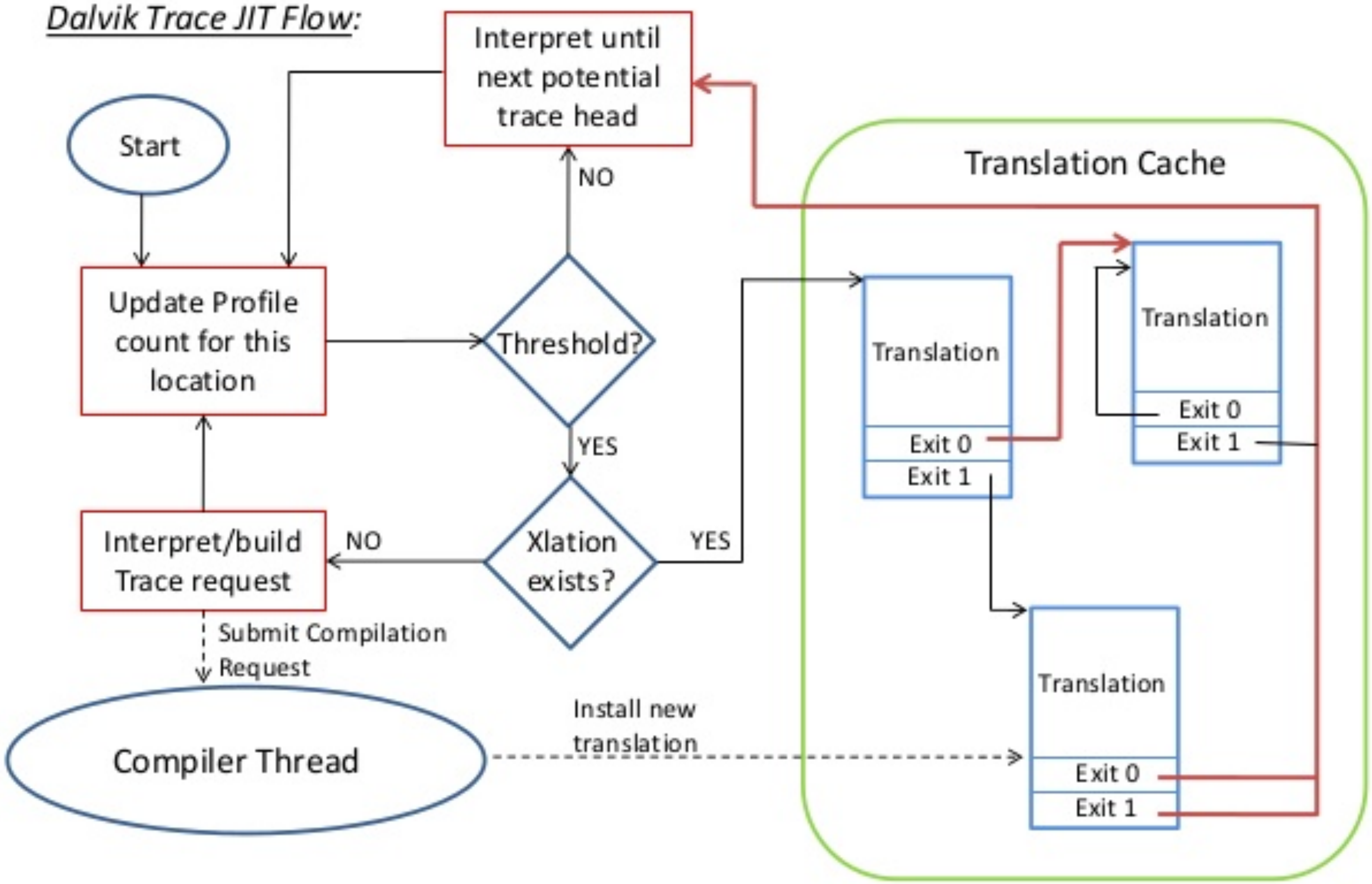
JIT Compiler

NativeCode

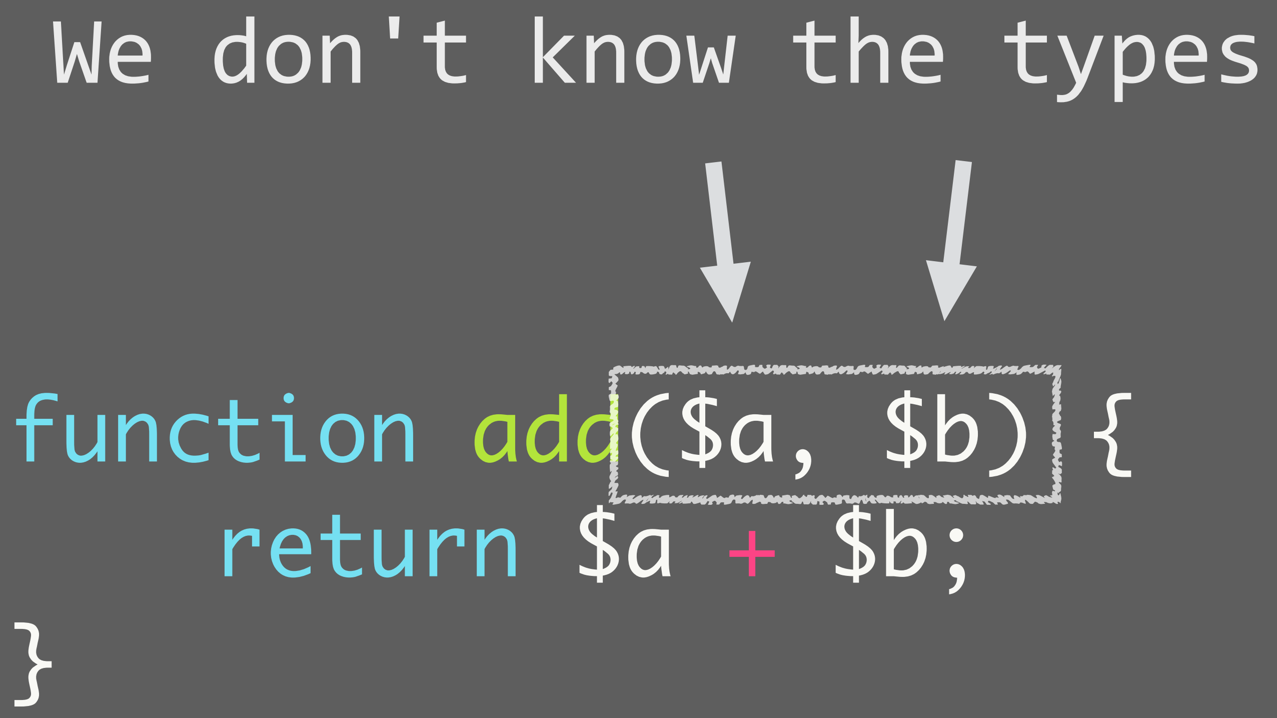
CPU

Dalvik JIT (Contd.):

Dalvik Trace JIT Flow:



Why Types Are Important?



We don't know the types

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

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

ZEND_ADD

```
ZEND_VM_HANDLER(1, ZEND_ADD, CONST|TMPVAR|CV, CONST|TMPVAR|CV)
```

http://www.opensource.apple.com/source/apache_mod_php/apache_mod_php-44/php/Zend/README.ZEND_VM long + long or long + double

```
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();
}
```

double + double | double + long

```
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();
}
```

```
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();
}
```

for other types

long + long

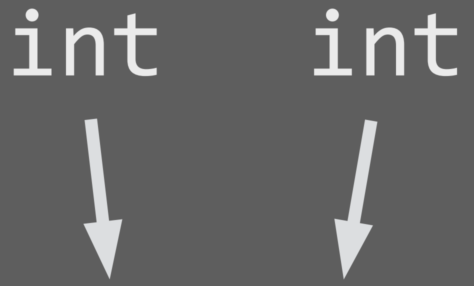
```c
ZEND_API int ZEND_FASTCALL add_function(zval *result, zval *op1, zval *op2) /* {{{ */
{
    zval op1_copy, op2_copy;
    int converted = 0;

    while (1) {
        switch (TYPE_PAIR(Z_TYPE_P(op1), Z_TYPE_P(op2))) {
            case TYPE_PAIR(IS_LONG, IS_LONG): {
                zend_long lval = Z_LVAL_P(op1) + Z_LVAL_P(op2);

                /* check for overflow by comparing sign bits */
                if ((Z_LVAL_P(op1) & LONG_SIGN_MASK) == (Z_LVAL_P(op2) & LONG_SIGN_MASK)
                    && (Z_LVAL_P(op1) & LONG_SIGN_MASK) != (lval & LONG_SIGN_MASK)) {

                    ZVAL_DOUBLE(result, (double) Z_LVAL_P(op1) + (double) Z_LVAL_P(op2));
                } else {
                    ZVAL_LONG(result, lval);
                }
                return SUCCESS;
            }

            case TYPE_PAIR(IS_LONG, IS_DOUBLE):
                ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_LONG):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_DOUBLE):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_ARRAY, IS_ARRAY):
                if ((result == op1) && (result == op2)) {
                    /* $a += $a */
                    return SUCCESS;
                }
                if (result != op1) {
                    ZVAL_DUP(result, op1);
                }
                zend_hash_merge(Z_ARRVAL_P(result), Z_ARRVAL_P(op2), zval_add_ref, 0);
                return SUCCESS;

            default:
                if (Z_ISREF_P(op1)) {
                    op1 = Z_REFVAL_P(op1);
```

long + double
double + long
double + double

```
ZEND_API int ZEND_FASTCALL add_function(zval *result, zval *op1, zval *op2) /* {{{ */
{
    zval op1_copy, op2_copy;
    int converted = 0;

    while (1) {
        switch (TYPE_PAIR(Z_TYPE_P(op1), Z_TYPE_P(op2))) {
            case TYPE_PAIR(IS_LONG, IS_LONG): {
                zend_long lval = Z_LVAL_P(op1) + Z_LVAL_P(op2);

                /* check for overflow by comparing sign bits */
                if ((Z_LVAL_P(op1) & LONG_SIGN_MASK) == (Z_LVAL_P(op2) & LONG_SIGN_MASK)
                    && (Z_LVAL_P(op1) & LONG_SIGN_MASK) != (lval & LONG_SIGN_MASK)) {

                    ZVAL_DOUBLE(result, (double) Z_LVAL_P(op1) + (double) Z_LVAL_P(op2));
                } else {
                    ZVAL_LONG(result, lval);
                }
                return SUCCESS;
            }

            case TYPE_PAIR(IS_LONG, IS_DOUBLE):
                ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_LONG):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_DOUBLE):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_ARRAY, IS_ARRAY):
                if ((result == op1) && (result == op2)) {
                    /* $a += $a */
                    return SUCCESS;
                }
                if (result != op1) {
                    ZVAL_DUP(result, op1);
                }
                zend_hash_merge(Z_ARRVAL_P(result), Z_ARRVAL_P(op2), zval_add_ref, 0);
                return SUCCESS;

            default:
                if (Z_ISREF_P(op1)) {
                    op1 = Z_REFVAL_P(op1);
```

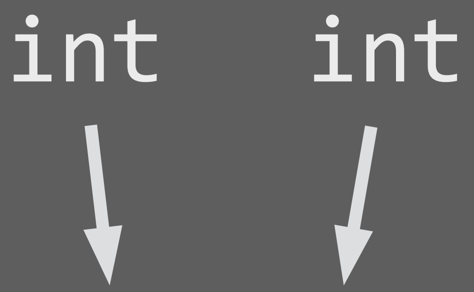
```
ZEND_API int ZEND_FASTCALL add_function(zval *result, zval *op1, zval *op2) /* {{{ */
{
    zval op1_copy, op2_copy;
    int converted = 0;

    while (1) {
        switch (TYPE_PAIR(Z_TYPE_P(op1), Z_TYPE_P(op2))) {
            case TYPE_PAIR(IS_LONG, IS_LONG): {
                zend_long lval = Z_LVAL_P(op1) + Z_LVAL_P(op2);

                /* check for overflow by comparing sign bits */
                if ((Z_LVAL_P(op1) & LONG_SIGN_MASK) == (Z_LVAL_P(op2) & LONG_SIGN_MASK)
                    && (Z_LVAL_P(op1) & LONG_SIGN_MASK) != (lval & LONG_SIGN_MASK)) {

                    ZVAL_DOUBLE(result, (double) Z_LVAL_P(op1) + (double) Z_LVAL_P(op2));
                } else {
                    ZVAL_LONG(result, lval);
                }
                return SUCCESS;
            }

            case TYPE_PAIR(IS_LONG, IS_DOUBLE):
                ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_LONG):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));
                return SUCCESS;

            case TYPE_PAIR(IS_DOUBLE, IS_DOUBLE):
                ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));
                return SUCCESS;

            case TYPE_PAIR(IS_ARRAY, IS_ARRAY):
                if ((result == op1) && (result == op2)) {
                    /* $a += $a */
                    return SUCCESS;
                }
                if (result != op1) {
                    ZVAL_DUP(result, op1);
                }
                zend_hash_merge(Z_ARRVAL_P(result), Z_ARRVAL_P(op2), zval_add_ref, 0);
                return SUCCESS;

            default:
                if (Z_ISREF_P(op1)) {
                    op1 = Z_REFVAL_P(op1);
```

array + array



int int

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


add(1,2);
```



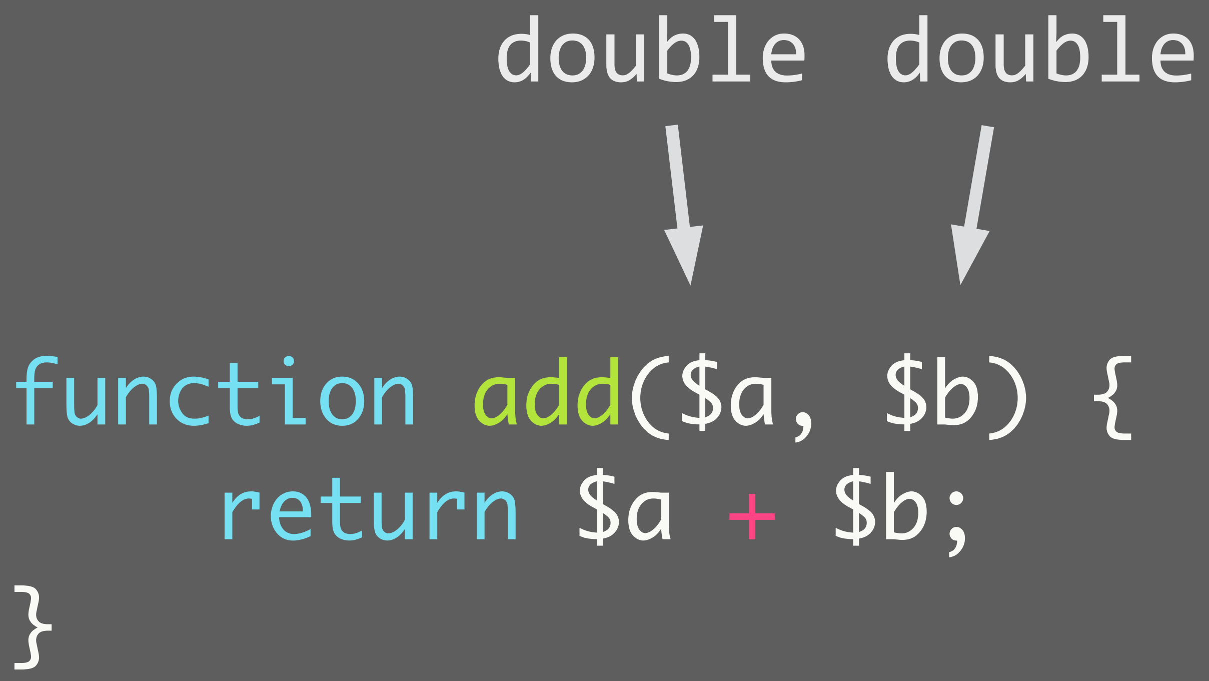
int int

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


add(1,2);
add(1,2);    x N
add(1,2);
```

```
               int   int

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

OK Enough, Let's compile a function:
add(int a, int b)

```
movl (address of a), %eax
movl (address of b), %ebx
addl %ebx, %eax
```



double double

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

```
add(1.3, 3.4);
```

libjit

PHPPHP

https://github.com/ircmaxell/PHPPHP

Anthony Ferrara

@ircmaxell

A PHP VM implementation written in PHP. This is a basic VM implemented in PHP using the AST generating parser developed by @nikic

recki-ct

https://github.com/google/recki-ct

Recki-CT is a set of tools that implement a compiler for PHP, and is written in PHP! Specifically, Recki-CT compiles a subset of PHP code. The subset is designed to allow a code base to be statically analyzed.

High Level Code Generation

Compile PHP to PHP

Compile PHP to Faster PHP

nikic/PHP-Parser

c9s/CodeGen

c9s/ClassTemplate

ActionKit