Code Generation in PHP

c9s

Yo-An Lin
@c9s

PHPBrew, R3, Pux, ...

@c9s

@c9s_en

C9S Hacker News

A long time ago in a galaxy far, far away....

Someone started a web framework.....

For the web, of course.

It was started from small

2m

3 years later, it looks like this

The Problems

Web Frameworks have too many conditions for different environment.

Including dynamic mechanism & feature checking

Frameworks usually do this

- Integrate configuration file & default configuration.
- 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....)
- etc...

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
<?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";
}
```

Environment checking is everywhere in frameworks.

for example

database connection configuration

template engine configuration (cache, recompile or not)

whether to cache database queries or not..

etc....

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

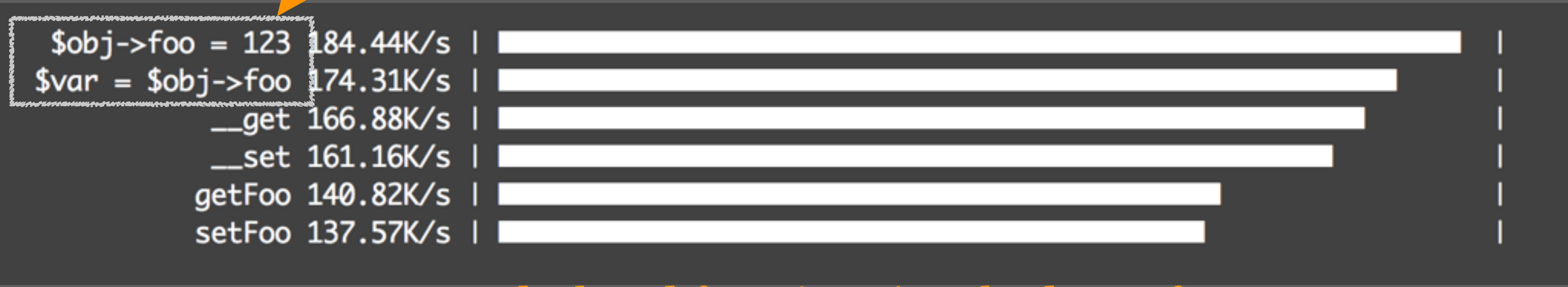
```php
<?php

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

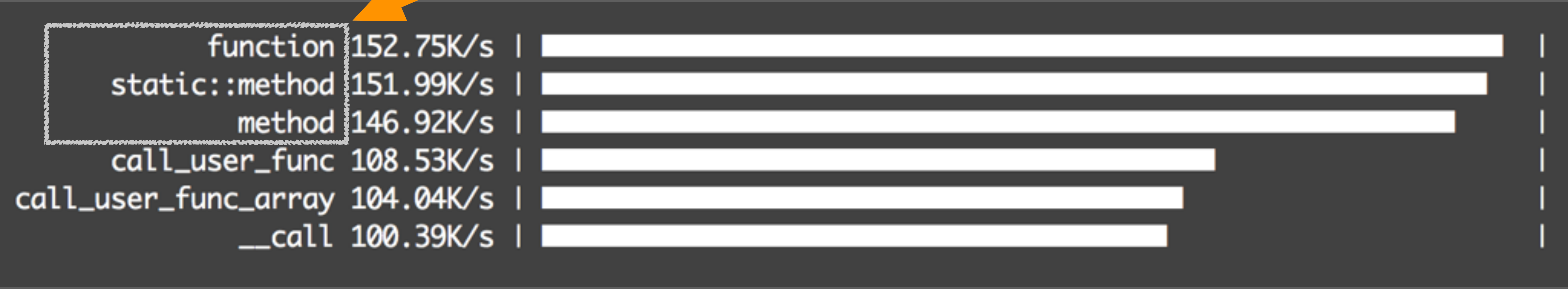
declared properties are faster

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.

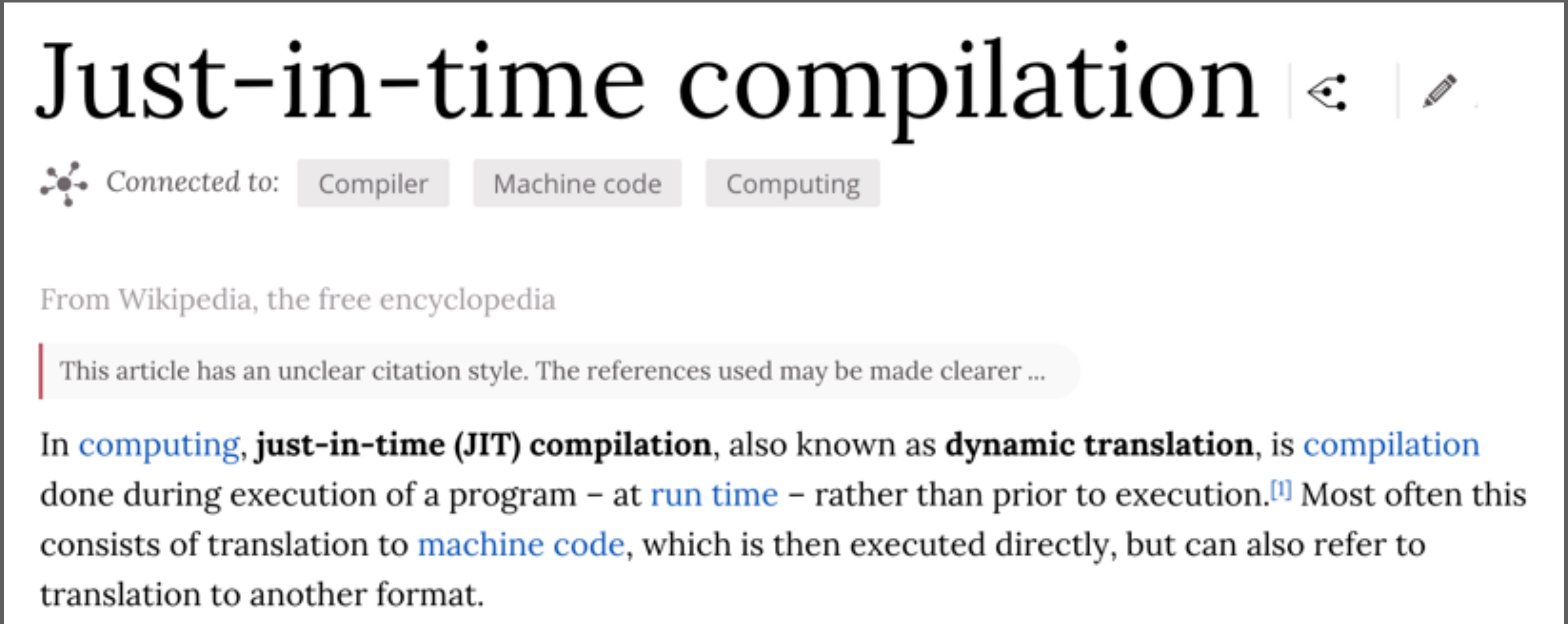
Types of Code Generation

Types of Code Generation

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

1. Low Level Code Generation

JIT (Just-in-time compilation)



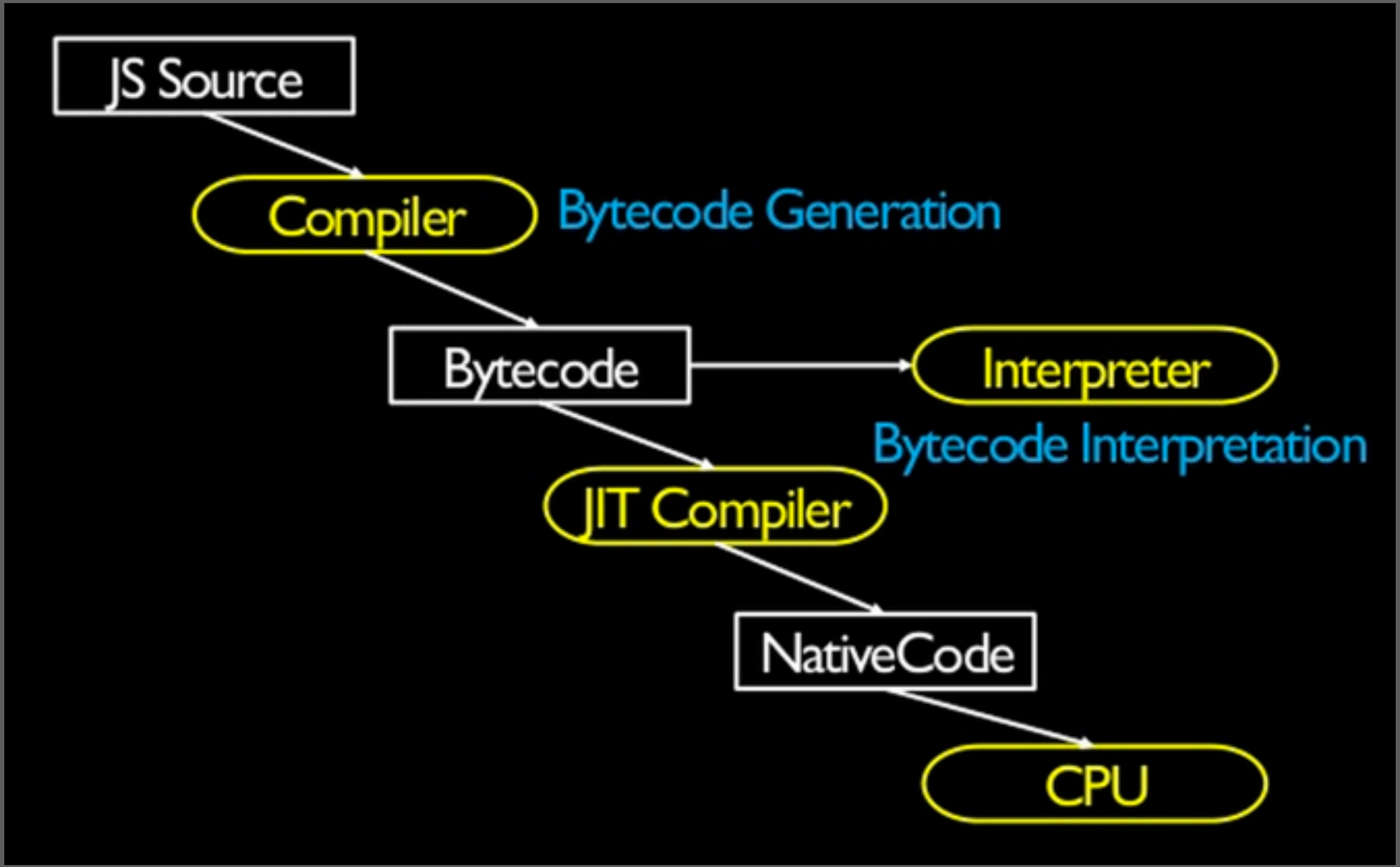
Just-in-time compilation

Connected to: Compiler Machine code Computing

From Wikipedia, the free encyclopedia

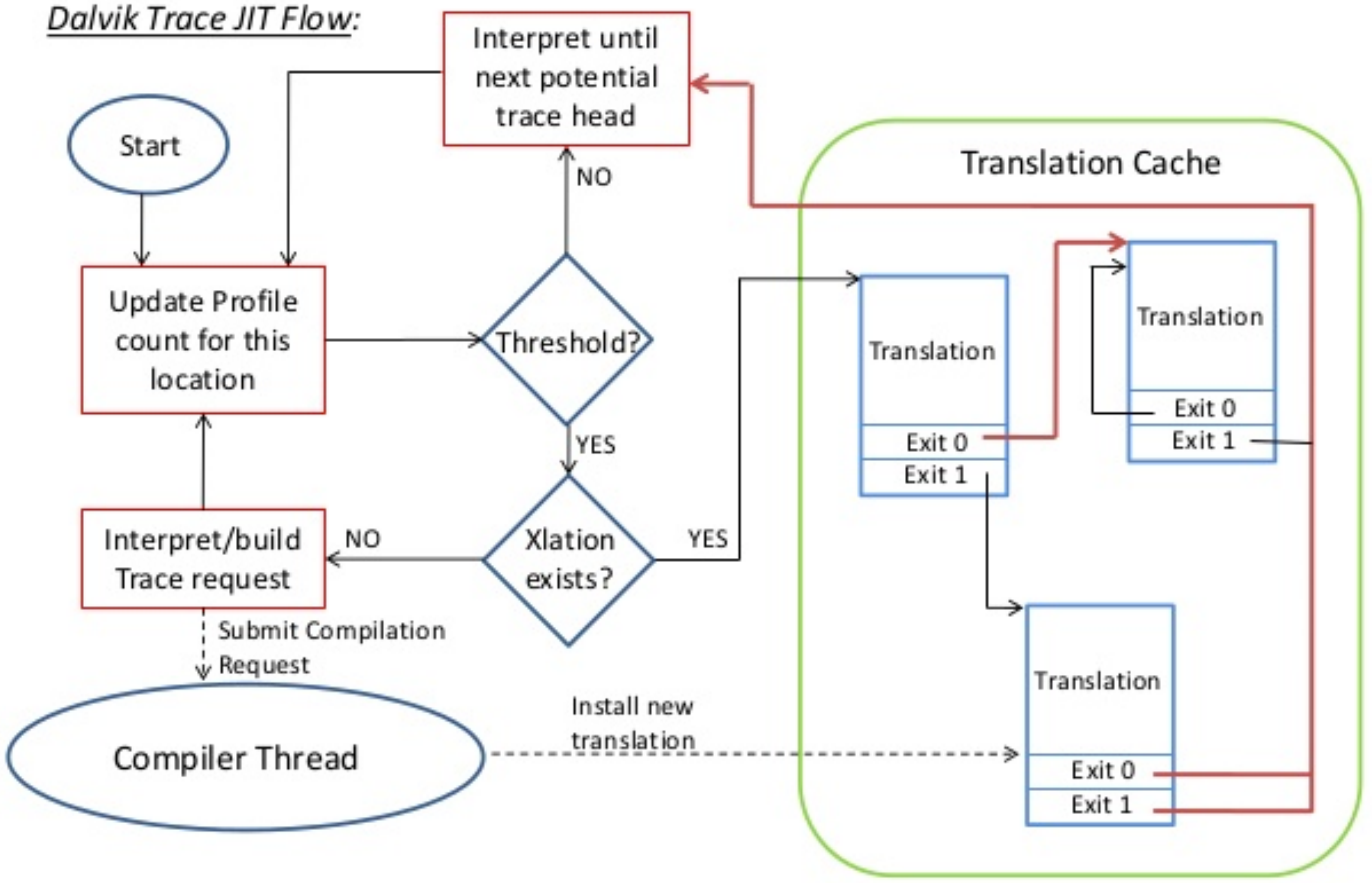
This article has an unclear citation style. The references used may be made clearer ...

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 to the Runtime System of VMs?

We don't know the types

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

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

ZEND_ADD

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

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

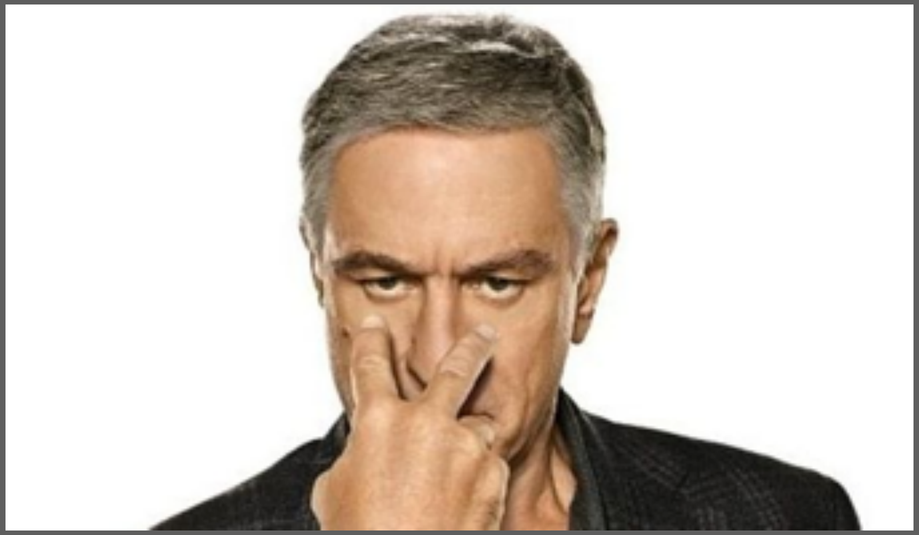
More details: Getting into the Zend Execution engine (PHP 5)

http://jpauli.github.io/2015/02/05/zend-vm-executor.html

We don't know the types

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

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

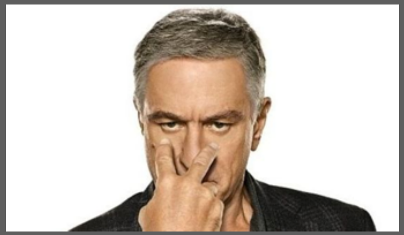


OK, Launch a thread for watching the types of function arguments

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

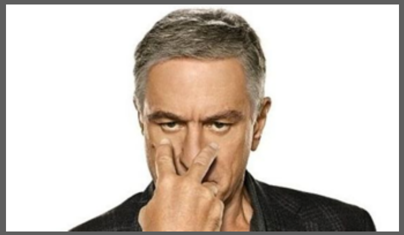
This is so called Trace-Based JIT Compilation. (also implemented in V8)

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



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

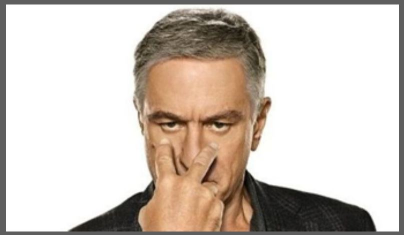
```
                 int    int
                  ↓      ↓

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



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

..... x N as a threshold

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



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

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

libjit

http://www.gnu.org/software/libjit/

LibJIT is a library that provides generic Just-In-Time compiler functionality independent of any particular byte-code, language, or runtime.

```c
int mul_add(int x, int y, int z)
{
    return x * y + z;
}
```

```c
#include <jit/jit.h>

jit_context_t context;
...
context = jit_context_create();
jit_context_build_start(context);
```

```
jit_function_t function;
...
function = jit_function_create(context, signature);
```

```
jit_type_t params[3];
jit_type_t signature;
...
params[0] = jit_type_int;
params[1] = jit_type_int;
params[2] = jit_type_int;
signature = jit_type_create_signature
    (jit_abi_cdecl, jit_type_int, params, 3, 1);
```

```
jit_value_t x, y, z;
...
x = jit_value_get_param(function, 0);
y = jit_value_get_param(function, 1);
z = jit_value_get_param(function, 2);
```

```
jit_value_t temp1, temp2;
...
temp1 = jit_insn_mul(function, x, y);
temp2 = jit_insn_add(function, temp1, z);
jit_insn_return(function, temp2);
```

```
jit_function_compile(function);
jit_context_build_end(context);
```

很難捉摸呀
It's not predictable.

Joe Watkins

@krakjoe

jitfu php extension

https://github.com/krakjoe/jitfu

Creating native instructions in PHP since 2014.

JIT-Fu is a PHP extension that exposes an OO API for the creation of native instructions to PHP userland, using libjit.

```php
<?php
use JITFU\Context;
use JITFU\Type;
use JITFU\Signature;
use JITFU\Func;
use JITFU\Value;

$context  = new Context();
$integer  = Type::of(Type::int);
$function = new Func($context, new Signature($integer, $integer,
$integer), function($args) use($integer) {
    $this->doReturn(
        $this->doAdd($this->doMul($args[0], $args[1]), $args[2])
    );
});
```

Pretty much simpler, isn't it?

You can get it through phpbrew

```
phpbrew ext install github:krakjoe/jitfu \
           -- --with-libjit=/opt/local
```

Related Projects

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

Anthony Ferrara

@ircmaxell

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.

LLVM vs LIBJIT?

http://eli.thegreenplace.net/2014/01/15/some-thoughts-on-llvm-vs-libjit

2. High Level Code Generation

Compile PHP to PHP

Compile PHP to Faster PHP

CodeGen

github.com/c9s/CodeGen

CodeGen transforms your dynamic calls to static code

Framework Bootstrap Script

Phifty Framework Bootstrap Script

https://github.com/c9s/Phifty/blob/master/src/Phifty/Bootstrap.php

```php
<?php
use ConfigKit\ConfigCompiler;
use ConfigKit\ConfigLoader;
// get PH_ROOT from phifty-core
defined( 'PH_ROOT' )      || define( 'PH_ROOT', getcwd() );
defined( 'PH_APP_ROOT' ) || define( 'PH_APP_ROOT' , getcwd() );
defined( 'DS' )          || define( 'DS' , DIRECTORY_SEPARATOR );
function initConfigLoader()
{
    // We load other services from the defi
    // Simple load three config files (frame
    $loader = new ConfigLoader;
    if ( file_exists( PH_APP_ROOT . '/config/framework.yml') ) {
        $loader->load('framework', PH_APP_ROOT . '/config/framework.yml');
    }
    // This is for DatabaseService
    if ( file_exists( PH_APP_ROOT . '/db/config/database.yml') ) {
        $loader->load('database', PH_APP_ROOT . '/db/config/database.yml');
    } elseif ( file_exists( PH_APP_ROOT . '/config/database.yml') ) {
        $loader->load('database', PH_APP_ROOT . '/config/database.yml');
    }
    // Config for application, services does not depends on this config file.
    if ( file_exists( PH_APP_ROOT . '/config/application.yml') ) {
        $loader->load('application', PH_APP_ROOT . '/config/application.yml');
    }
    // Only load testing configuration when environment
    // is 'testing'
    if ( getenv('PHIFTY_ENV') === 'testing' ) {
        if ( file_exists( PH_APP_ROOT . '/config/testing.yml' ) ) {
            $loader->load('testing', ConfigCompiler::compile(PH_APP_ROOT . '/config/testing.yml') );
        }
    }
    return $loader;
}
```

A lot of dynamic checking

Dynamic initialization

```php
<?php
$kernel = new \Phifty\Kernel;
$kernel->prepare(); // prepare constants
$composerLoader = require PH_ROOT . '/vendor/autoload.php';
$kernel->registerService( new \Phifty\Service\ClassLoaderService(getSplClassLoader()));
$configLoader = initConfigLoader();
$kernel->registerService( new \Phifty\Serv
$kernel->registerService( new \Phifty\Serv
// if the framework config is defined.
if ( $configLoader->isLoaded('framework') ) {
    // we should load database service before other services
    // because other services might need database service
    if ( $configLoader->isLoaded('database') ) {
        $kernel->registerService( new \Phifty\Service\DatabaseService );
    }
    if ( $appconfigs = $kernel->config->get('framework','Applications') ) {
        foreach ($appconfigs as $appname => $appconfig) {
            $kernel->classloader->addNamespace( array(
                $appname => array( PH_APP_ROOT . '/applications' , PH_ROOT . '/applications' )
            ));
        }
    }
    if ( $services = $kernel->config->get('framework','Services') ) {
        foreach ($services as $name => $options) {
            // not full qualified classname
            $class = ( false === strpos($name,'\\') ) ? ('Phifty\\Service\\' . $name) : $name;
            $kernel->registerService( new $class , $options );
        }
    }
}
$kernel->init();
```

~1000 lines to bootstrap

Laravel Framework Bootstrapping

https://github.com/laravel/framework/blob/5.1/src/Illuminate/Foundation/Application.php

```
public function __construct($basePath = null)
{
    $this->registerBaseBindings();
    $this->registerBaseServiceProviders();
    $this->registerCoreContainerAliases();
    if ($basePath) {
        $this->setBasePath($basePath);
    }
}
```

```
/**
    * Register the core class aliases in the container.
    *
    * @return void
    */
public function registerCoreContainerAliases()
{
    $aliases = [
        'app'                  => ['Illuminate\Foundation\Application', 'Illuminate\Contracts
\Container\Container', 'Illuminate\Contracts\Foundation\Application'],
        'auth'                 => 'Illuminate\Auth\AuthManager',
        'auth.driver'          => ['Illuminate\Auth\Guard', 'Illuminate\Contracts\Auth\Guard'],
        'auth.password.tokens' => 'Illuminate\Auth\Passwords\TokenRepositoryInterface',
        'url'                  => ['Illuminate\Routing\UrlGenerator', 'Illuminate\Contracts\Routin
\UrlGenerator'],
.....
20 lines cut
.....
        'validator'            => ['Illuminate\Validation\Factory', 'Illuminate\Contracts
\Validation\Factory'],
        'view'                 => ['Illuminate\View\Factory', 'Illuminate\Contracts\View\Factory'],
    ];
    foreach ($aliases as $key => $aliases) {
        foreach ((array) $aliases as $alias) {
            $this->alias($key, $alias);
        }
    }
}
```

```php
public function detectEnvironment(Closure $callback)
{
    $args = isset($_SERVER['argv']) ? $_SERVER['argv'] : null;
    return $this['env'] = (new EnvironmentDetector())->detect($callback, $args);
}
```

Illuminate\Foundation\Bootstrap\ConfigureLogging ~120 lines
Illuminate\Foundation\Bootstrap\DetectEnvironment ~ 29 lines
Illuminate\Foundation\Bootstrap\HandleExceptions
Illuminate\Foundation\Bootstrap\LoadConfiguration
Illuminate\Foundation\Bootstrap\RegisterFacades
Illuminate\Foundation\Bootstrap\RegisterProviders

~3000 lines of code to bootstrap an application

Using CodeGen to reduce checks and remove conditions

Declaring Block

```php
use CodeGen\Block;
use CodeGen\Comment;
use CodeGen\CommentBlock; $block = new Block;
$block[] = '<?php';
$block[] = new CommentBlock([
    "This file is auto-generated through 'bin/phifty bootstrap' command.",
    "Don't modify this file directly",
    "",
    "For more information, please visit https://github.com/c9s/Phifty",
]);
```

Declaring Require Statement

```
// Generates: $kernel->registerService(new \Phifty\ServiceProvider
\EventServiceProvider());
$block[] = new Comment('The event service is required for every component.');
$block[] = new RequireClassStatement('Phifty\\ServiceProvider\\EventServiceProvider');
$block[] = new Statement(new MethodCall('$kernel', 'registerService', [
    new NewObject('\\Phifty\\ServiceProvider\\EventServiceProvider'),
]));
```

Declaring Conditional Statement

```
$stmt = new ConditionalStatement($foo == 1, '$foo = 1');
$stmt->when($foo == 2, function() {
    return '$foo = 2;';
});
$stmt->when($foo == 3, function() {
    return '$foo = 3;';
});
```

```php
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/ActionServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\ActionServiceProvider(array (
  'DefaultFieldView' => 'ActionKit\\FieldView\\BootstrapFieldView',
)));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/PuxRouterServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\PuxRouterServiceProvider(array ()));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/LibraryServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\LibraryServiceProvider(array ()));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/ViewServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\ViewServiceProvider(array (
  'Backend' => 'twig',
  'Class' => 'App\\View\\PageView',
)));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/MailerServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\MailerServiceProvider(array (
  'Transport' => 'MailTransport',
)));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/MongodbServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\MongodbServiceProvider(array ( 'DSN' => 'mongodb://
localhost',)));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/CacheServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\CacheServiceProvider(array ()));
require '___/___/vendor/corneltek/phifty-core/src/Phifty/ServiceProvider/LocaleServiceProvider.php';
$kernel->registerService(new Phifty\ServiceProvider\LocaleServiceProvider(array (
  'Directory' => 'locale',
  'Default' => 'zh_TW',
  'Domain' => '___',
  'Langs' =>
  array (
    0 => 'en',
    1 => 'zh_TW',
  ),
)));
```

Integrating PHP Parser for CodeGen with Annotation

nikic/PHP-Parser

PHP-Parser

https://github.com/nikic/PHP-Parser

a PHP 5.2 to PHP 5.6 parser written in PHP. Its purpose is to simplify static code analysis and manipulation.

```
// @codegen
if ($environment == "development") {
    $handler = new DevelopmentHandler;
} else {
    $handler = new ProductionHandler;
}
```

```php
$handler = new DevelopmentHandler;
```

LazyRecord

https://github.com/c9s/LazyRecord

ORM implemented with Code Generation Technologies

```php
<?php
namespace LazyRecord\Schema\Factory;
use ClassTemplate\TemplateClassFile;
use ClassTemplate\ClassFile;
use LazyRecord\Schema\SchemaInterface;
use LazyRecord\Schema\DeclareSchema;
use Doctrine\Common\Inflector\Inflector;

class BaseModelClassFactory
{
    public static function create(DeclareSchema $schema, $baseClass) {
        $cTemplate = new ClassFile($schema->getBaseModelClass());
        $cTemplate->addConsts(array(
            'schema_proxy_class' => $schema->getSchemaProxyClass(),
            'collection_class'   => $schema->getCollectionClass(),
            'model_class'        => $schema->getModelClass(),
            'table'              => $schema->getTable(),
            'read_source_id'     => $schema->getReadSourceId(),
            'write_source_id'    => $schema->getWriteSourceId(),
            'primary_key'        => $schema->primaryKey,
        ));

        $cTemplate->addMethod('public', 'getSchema', [], [
            'if ($this->_schema) {',
            '   return $this->_schema;',
            '}',
            'return $this->_schema = \LazyRecord\Schema\SchemaLoader::load(' . var_export($schema-
>getSchemaProxyClass(),true) .  ');',
        ]);

        $cTemplate->addStaticVar('column_names',  $schema->getColumnNames());
        $cTemplate->addStaticVar('column_hash',  array_fill_keys($schema->getColumnNames(), 1 ) );
        $cTemplate->addStaticVar('mixin_classes', array_reverse($schema-
>getMixinSchemaClasses()) );
```

```php
<?php
namespace UserBundle\Model;
use LazyRecord\BaseModel;
class UserBase
    extends BaseModel
{
    const schema_proxy_class = 'UserBundle\\Model\\UserSchemaProxy';
    const collection_class = 'UserBundle\\Model\\UserCollection';
    const model_class = 'UserBundle\\Model\\User';
    const table = 'users';
    const read_source_id = 'default';
    const write_source_id = 'default';
    const primary_key = 'id';
    public static $column_names = array (
      0 => 'id',
      1 => 'password',
      2 => 'auth_token',
      3 => 'account',
      4 => 'confirmed',
      5 => 'email',
      6 => 'name',
      7 => 'cellphone',
      8 => 'phone',
      9 => 'role',
      10 => 'company',
      11 => 'receive_email',
      12 => 'receive_sms',
      13 => 'remark',
      14 => 'org_id',
    );
    public static $column_hash = array (
      'id' => 1,
      'password' => 1,
      'auth_token' => 1,
      'account' => 1,
      'confirmed' => 1,
      'email' => 1,
```

ActionKit

github.com/c9s/ActionKit

ActionKit handles your PHP web application logics and record relationships

Generating CRUD Handler automatically in the Runtime

App\Model\Product

ActionKit Generates API classes automatically

```
App\Action\CreateProduct
App\Action\UpdateProduct
App\Action\DeleteProduct
```

Trigger ActionKit ActionGenerator by SPL autoloader

```
use App\Action\CreateProduct;

$create = new CreateProduct(['name' => 'Product I', 'sn' =>
'PN-12345677']);
$success = $create->invoke();
```

The SPL autoloader generates the action class in cache directory automatically.

```php
<?php
/**
This is an auto-generated file,
Please DO NOT modify this file directly.
*/
namespace App\Action;

use ActionKit\Action;
use ActionKit\RecordAction\BaseRecordAction;
use ActionKit\RecordAction\UpdateRecordAction;

class UpdateStore  extends UpdateRecordAction {


public $recordClass = 'App\\Model\\Product';


}
```

ConfigKit

https://github.com/c9s/ConfigKit

The optimized config loader

```
use ConfigKit\ConfigLoader;

$loader = new ConfigLoader();
if (file_exists($baseDir.'/config/framework.yml')) {
    $loader->load('framework', $baseDir.'/config/framework.yml');
}

// This is for DatabaseService
if (file_exists($baseDir.'/db/config/database.yml')) {
    $loader->load('database', $baseDir.'/db/config/database.yml');
} elseif (file_exists($baseDir.'/config/database.yml')) {
    $loader->load('database', $baseDir.'/config/database.yml');
}
```

```php
use CodeGen\Generator\AppClassGenerator;
use ConfigKit\ConfigLoader;
$configLoader = new ConfigLoader;
$configClassGenerator = new AppClassGenerator([
    'namespace' => 'App',
    'prefix' => 'App'
]);
$configClass = $configClassGenerator->generate($configLoader);
$classPath = $configClass->generatePsr4ClassUnder('app');
$block[] = new RequireStatement(PH_APP_ROOT . DIRECTORY_SEPARATOR .
$classPath);
```

Thank You