

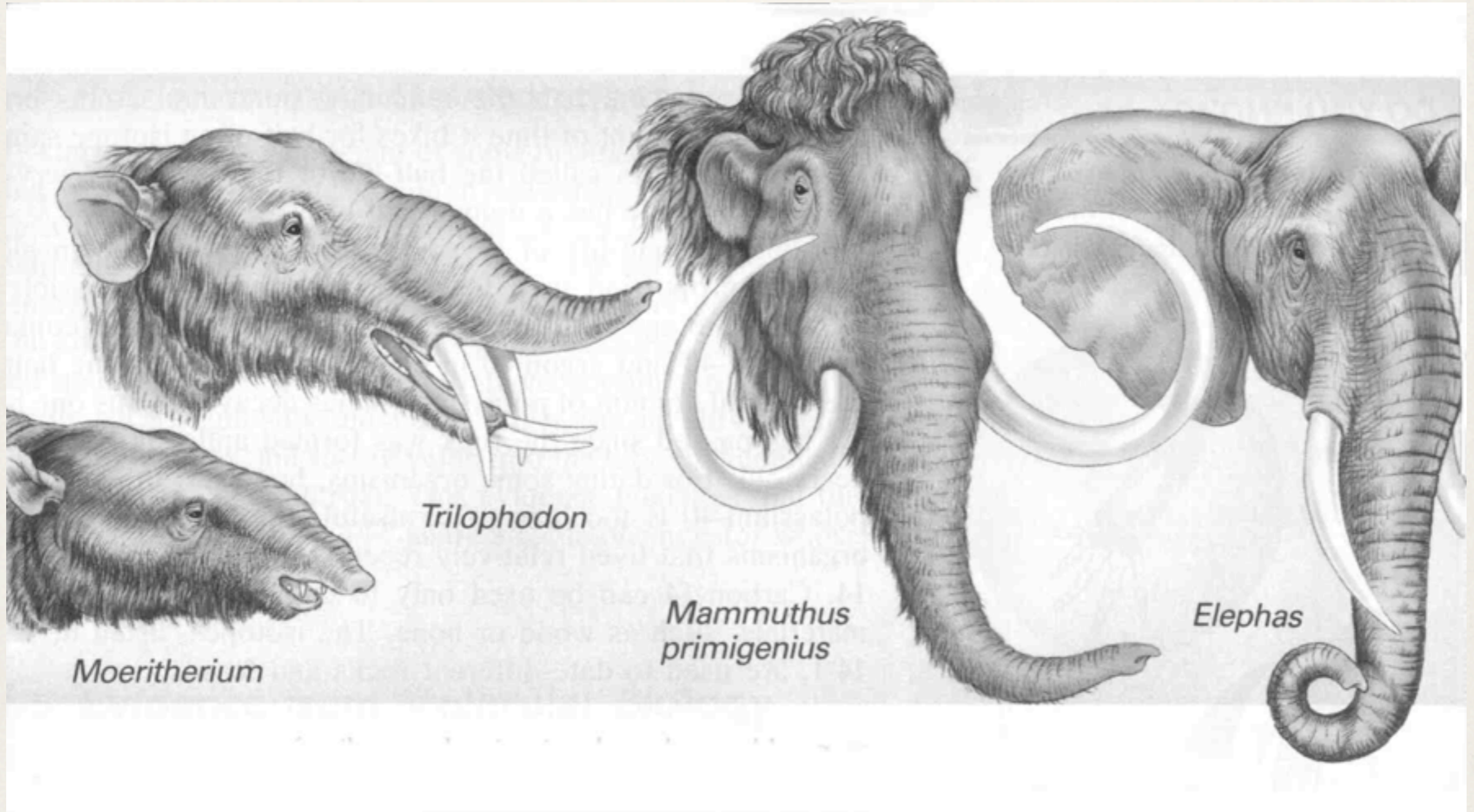
Introducing PHP 5.4

Ilia Alshanetsky - @iliaa

whois: Ilia Alshanetsky

- ❖ PHP Core Developer since 2001
- ❖ Author of “Guide to PHP Security”
- ❖ CIO @ Centah Inc.
- ❖ Occasional Photographer ;-)

Evolutionary Progression





The little things ...

Array Dereferencing (finally!!)

You can finally put the temporary variables to rest and retrieve values from returned arrays directly.

```
$a = "hello world";  
echo explode(" ", $a)[0]; // prints hello
```

```
function test() {  
    return array("test" =>  
                array("foo" => "bar"));  
}
```

```
echo test()['test']['foo']; // prints bar
```

<?= “they always work now” ?>

<?= "no more ini settings to worry about" ?>

Compact Array Syntax

```
$a = [1, 2, 3];
```

```
// Equivalent of array(1, 2, 3);
```

```
$b = ['a' => 'b', 'x' => 'y'];
```

```
// Same as array('a' => 'b', 'x' => 'y');
```

There are only **0b10** kinds of people; those who understand binary and those who don't.

And now, so can PHP.

```
$a = 0b101; // integer(5)
```


JSON Serialization Helper

via `jsonSerializable` interface

```
class myClass implements JsonSerializer {
    private $data, $multiplier;

    public function __construct($a, $b) {
        $this->data = $a;
        $this->multiplier = $b;
    }

    public function jsonSerialize() {
        return array_fill(
            0, $this->multiplier,
            $this->data);
    }
}
```

```
echo json_encode(new myClass(123, 3));
// will print [123,123,123]
```

Native Session Handler Interface

```
session_set_save_handler(  
    array($this, "open"),  
    array($this, "close"),  
    array($this, "read"),  
    array($this, "write"),  
    array($this, "destroy"),  
    array($this, "gc")  
);
```

Ugly

SessionHandler implements SessionHandlerInterface {

/* Methods */

public int close (void)

public int destroy (string \$sessionid)

public int gc (int \$maxlifetime)

public int open (string \$save_path , string \$sessionid)

public string read (string \$sessionid)

public int write (string \$sessionid , string \$sessiondata)

}

session_set_save_handler(new MySessionHandler);

Objects as Functions

via `__invoke()` callback

```
class doStuff {  
    function __invoke($var) {  
        return $var * $var;  
    }  
}
```

```
$o = new doStuff();
```

```
echo $o(10);
```

```
// will print 100
```

Callable Type-Hint



```
function doSomething(callable $x) {  
    return $x();  
}
```

```
doSomething(function () { });  
doSomething("function_name");  
doSomething(['class', 'staticMethodName']);  
doSomething([$object, 'methodName']);  
doSomething($invokableObject);
```


\$this in Anonymous Functions

```
class foo {  
    function test() {  
        echo "Foo walks into a bar...";  
    }  
  
    function anonFunc() {  
        return function() { $this->test(); };  
    }  
}
```

```
class bar {  
    public function __construct(foo $o) {  
        $a = $o->anonFunc(); $a();  
    }  
}
```

```
new bar(new foo); // prints "Foo walks into a bar..."
```

Initialized High Precision Timer

```
// < 5.4
```

```
$start = microtime(1);
```

```
/* your code here */
```

```
echo "took: ", (microtime(1) - $start);
```

```
// >= 5.4
```

```
/* your code here */
```

```
echo "took: ",  
    (microtime(1) -  
     $_SERVER[ 'REQUEST_TIME_FLOAT' ] );
```



The Big Stuff ...

Traits

a.k.a. Horizontal Reuse

a.k.a. Multiple Inheritance

```
trait Hello {  
    public function hello() {  
        return "Hello";  
    }  
}  
  
trait City {  
    public function city($name) {  
        return $name;  
    }  
}  
  
class Welcome {  
    use Hello, City;  
}  
  
$c = new Welcome();  
echo $c->hello(), ' ', $c->city("Montreal");  
// prints "Hello Montreal"
```

```
trait Hello {
    public function hello() { return "Hello"; }
}

trait City {
    public function city($name) { return $name; }
}

trait Greeting {
    use Hello, City;

    public function greeting($name) {
        echo $this->hello(), ' ', $this->city($name);
    }
}

class Welcome { use Greeting; }

(new Welcome())->greeting("Montreal");
// prints "Hello Montreal"
```



```
trait Hello {  
    private function hello($city) {  
        echo "Hello " , $city;  
    }  
}
```

```
class Welcome {  
    use Hello {  
        hello as public;  
    }  
}
```

```
(new Welcome())->hello("Montreal");  
// prints "Hello Montreal"
```

```
trait Who {  
    public function msg($name) { return $name; }  
}
```

```
trait Hello {  
    private function msg() { return "Hello"; }  
}
```

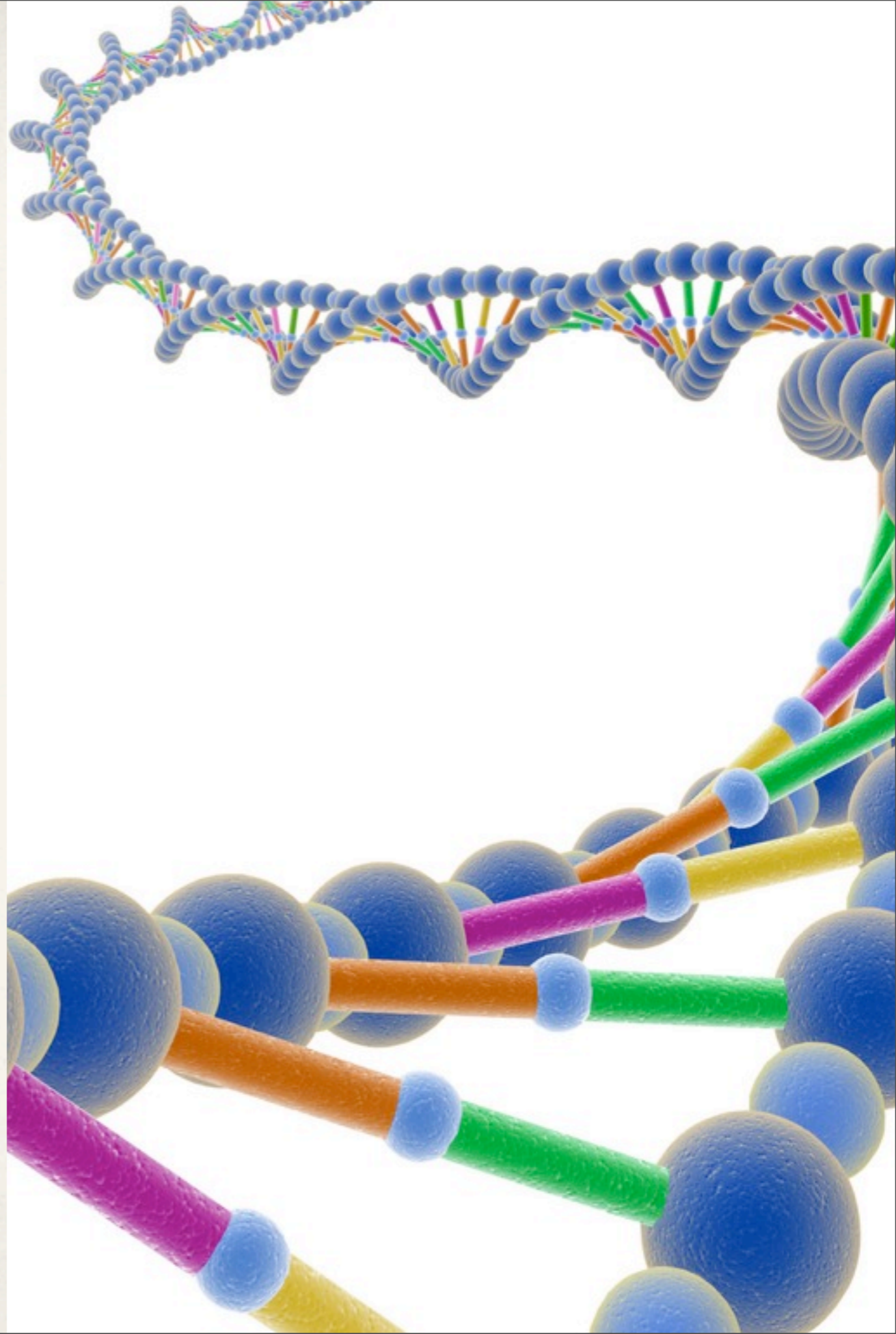
```
class Welcome {  
    use Who, Hello {  
        hello::msg as public hi;  
        Who::msg insteadof Hello;  
    }  
}
```

```
$w = new Welcome();
```

```
echo $w->hi(), ' ', $w->msg("Montreal");  
// prints "Hello Montreal"
```

Identifying Traits

The consequence of the “copy & paste” implementation of traits, makes them a little hard to detect.



```

trait Hello {
    public function hi() { return 'Hello'; }
}

class Welcome {
    use Hello { hello::hi as public hello; }
}

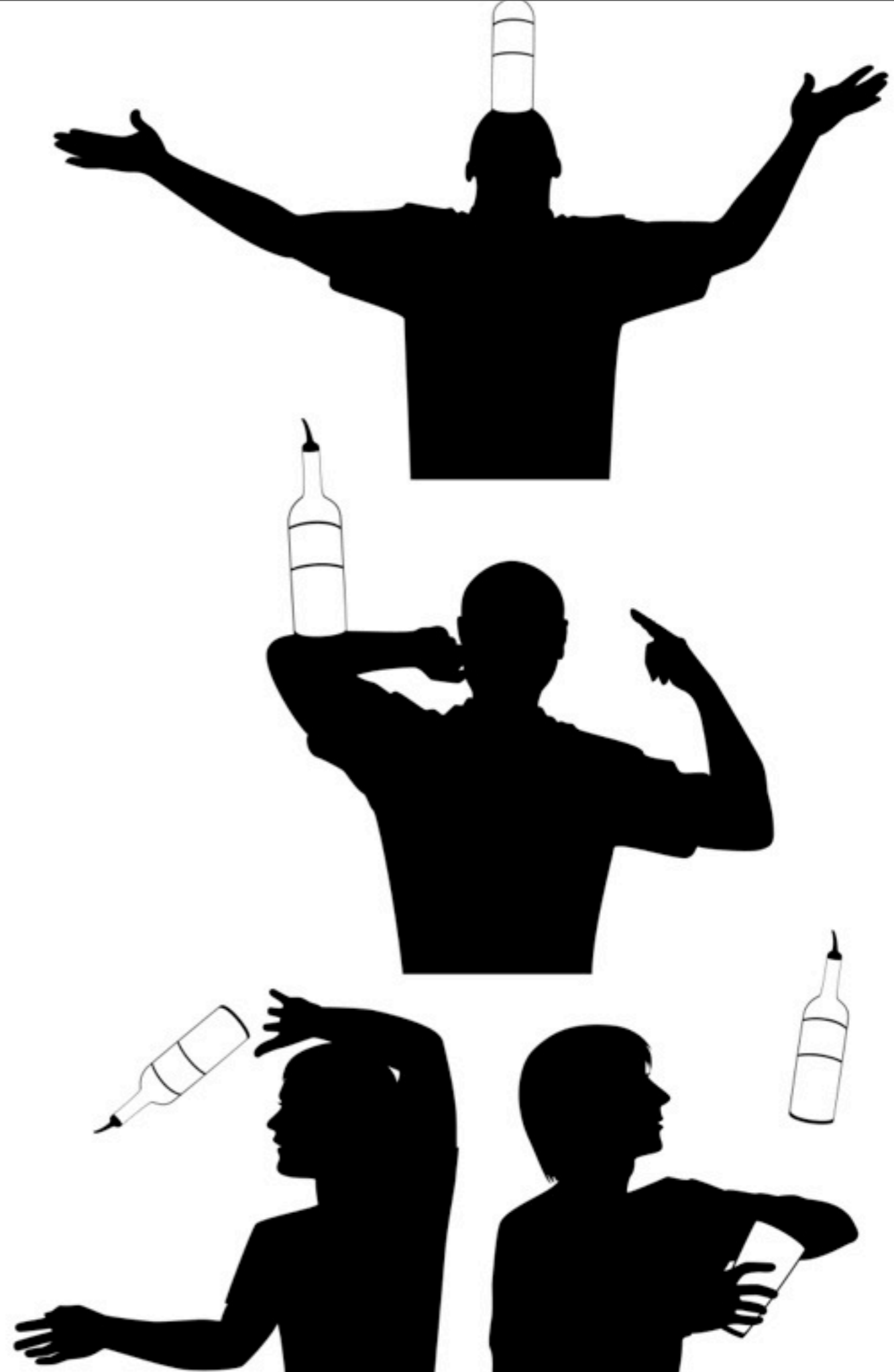
$src = new ReflectionClass('Welcome');
if (!$src->isTrait()) {
    echo $src->getTraitNames()[0]; // prints "Hello"

    echo $src->getTraitAliases()['hello'];
    // prints "hello::hi"

    foreach ($src->getTraits() as $v) {
        echo $v->getMethods()[0]->class, ' ',
            $v->getMethods()[0]->name;
        // prints "hello hi"
    }
}

```

Built-in CLI Web-Server



```
ilia@s3 ~ $ php -S localhost:8080 -t /tmp/web router.php
```

```
PHP 5.4.0RC8-dev Development Server started at Sat Feb 11 12:17:50 2012
```

```
Listening on localhost:8080
```

```
Document root is /tmp/web
```

```
Press Ctrl-C to quit.
```

```
[Sat Feb 11 12:18:26 2012] 127.0.0.1:45018 [200]: /logo.png
```

```
[Sat Feb 11 12:18:33 2012] 127.0.0.1:45021 [200]: /index.php
```

Basic Request Router

```
if (preg_match('!\.php$', $_SERVER["REQUEST_URI"])) {
    require basename($_SERVER["REQUEST_URI"]);
} else if (strpos($_SERVER["REQUEST_URI"], '.') {
    return false; // serve the requested file as-is.
} else {
    Framework::Router($_SERVER["REQUEST_URI"]);
}
```

Notes & Disclaimers

- ❖ Brand new, so **use with caution**
- ❖ Default index file is “index.php”
- ❖ No support for source highlighting via “.phps” (for now)
- ❖ Intended for Development, NOT for production.
- ❖ No SSL support



Performance Improvements

Lots of improvements

- * Replaced `zend_function.pass_rest_by_reference` by `ZEND_ACC_PASS_REST_BY_REFERENCE` in `zend_function.fn_flags`.
- * Replaced `zend_function.return_reference` by `ZEND_ACC_RETURN_REFERENCE` in `zend_function.fn_flags`.
- * Removed `zend_arg_info.required_num_args` as it was only needed for internal functions. Now the first `arg_info` for internal functions (which has special meaning) is represented by `zend_internal_function_info` structure.
- * Moved `zend_op_array.size`, `size_var`, `size_literal`, `current_brk_cont`, `backpatch_count` into `CG(context)` as they are used only during compilation.
- * Moved `zend_op_array.start_op` into `EG(start_op)` as it's used only for 'interactive' execution of single top-level op-array.
- * Replaced `zend_op_array.done_pass_two` by `ZEND_ACC_DONE_PASS_TWO` in `zend_op_array.fn_flags`.
- * `op_array.vars` array is trimmed (reallocated) during `pass_two`.
- * Replaced `zend_class_entry.constants_updated` by `ZEND_ACC_CONSTANTS_UPDATED` in `zend_class_entry.ce_flags`.
- * Reduced the size of `zend_class_entry` by sharing the same memory space by different information for internal and user classes. See `zend_class_entry.info` union.
- * Reduced size of `temp_variable`.
- * Changed the structure of `op_array.opcodes`. The constant values are moved from opcode operands into a separate literal table.
- * Inlined most probable code-paths for arithmetic operations directly into executor.
- * Eliminated unnecessary iterations during request startup/shutdown.
- * Changed `$GLOBALS` into a JIT autoglobal, so it's initialized only if used. (this may affect opcode caches!)
- * Improved performance of `@` (silence) operator.
- * Simplified string offset reading. `$str[1][0]` is now a legal construct.
- * Added caches to eliminate repeatable run-time bindings of functions, classes, constants, methods and properties.
- * Added concept of interned strings. All strings constants known at compile time are allocated in a single copy and never changed.
- * Added an optimization which saves memory and `emalloc/efree` calls for empty HashTables.
- * `ZEND_RECV` now always has `IS_CV` as its result.
- * `ZEND_CATCH` now has to be used only with constant class names.
- * `ZEND_FETCH_DIM_?` may fetch array and dimension operands in different order.
- * Simplified `ZEND_FETCH_*_R` operations. They can't be used with the `EXT_TYPE_UNUSED` flag any more. This is a very rare and useless case. `ZEND_FREE` might be required after them instead.
- * Split `ZEND_RETURN` into two new instructions `ZEND_RETURN` and `ZEND_RETURN_BY_REF`.
- * Optimized access to global constants using values with pre-calculated `hash_values` from the literals table.
- * Optimized access to static properties using executor specialization.
- * A constant class name may be used as a direct operand of `ZEND_FETCH_*` instruction without previous `ZEND_FETCH_CLASS`.
- * `zend_stack` and `zend_ptr_stack` allocation is delayed until actual usage.
- * Zend Signal Handling

Lies, Damn Lies & Statistics

- ❖ 5.4 is the fastest PHP version yet!
 - ❖ On average real-life applications are 5-20% faster
 - ❖ Static benchmarks show 15-20% speed improvement
 - ❖ Memory usage reduced at least 25% in most real-life applications



Cleanup ...

False Security “Features”

- ❖ Safe Mode
 - ❖ Not really “safe” and caused many problems
- ❖ Magic Quotes
 - ❖ Poor mechanism for securing SQL data

Welcome to 2012

- ❖ Register Globals & Register long arrays (HTTP_GET_VARS)
- ❖ y2k_compliance option for cookies
- ❖ Allow-call-time pass reference [foo(&\$bar)]
- ❖ import_request_variables() function
- ❖ Session extension's bug compatibility options
- ❖ SQLite 2 extension (replaced with SQLite 3)



Backwards Compatibility Breaks

Internal entities functions



Consequences For:

- ▶ **Input Processing** (GET / POST / `default_charset` INI)
- ▶ **Data Storage** (Database charset)
- ▶ **Output** (`htmlspecialchars` / `htmlentities`)

Removed “Features”

- ❖ Time-zone guessing code, now defaults to UTC
- ❖ `break $foo;` and `continue $bar;`
- ❖ Setting of Timezones via `putenv(“TZ=...”)`
- ❖ `define_syslog_variables()` (use pre-defined constants)

I'd love to hear your feedback

<http://joind.in/6063>

Ilia Alshanetsky

ilia@ilia.ws

<http://ilia.ws/>

@iliaa