HHVM - PHP++

PAUL TARJAN
HHVM OPEN SOURCE

SARA GOLEMON
HHVM OPEN SOURCE

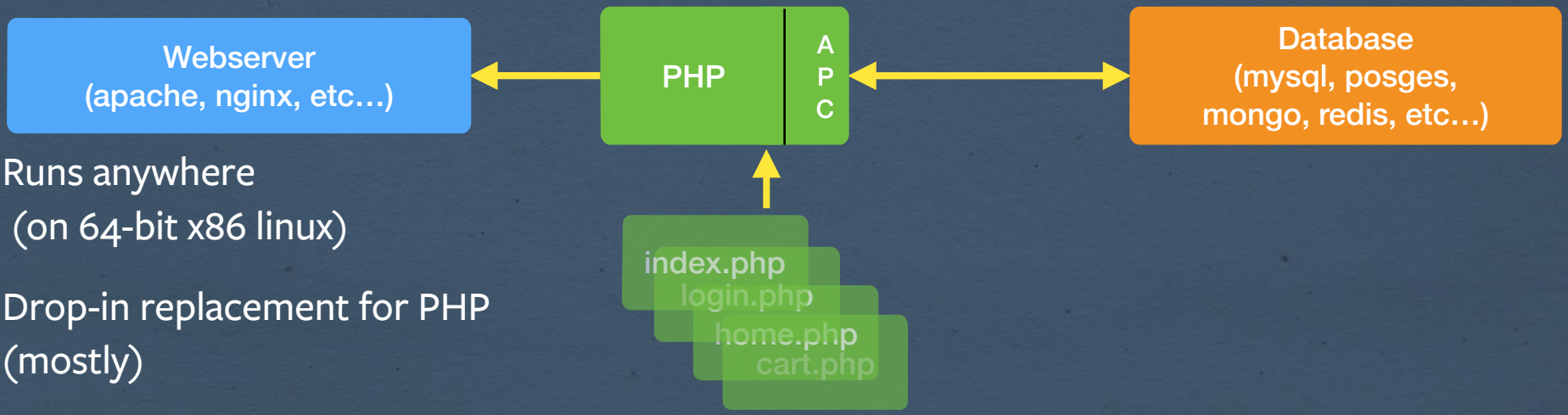
Facebook
Open Source

What is HHVM?

HHVM is not a source transformer

That was HPHPc, it's dead.

- Runs your PHP pages live, just like PHP
- Uses standard FastCGI transport

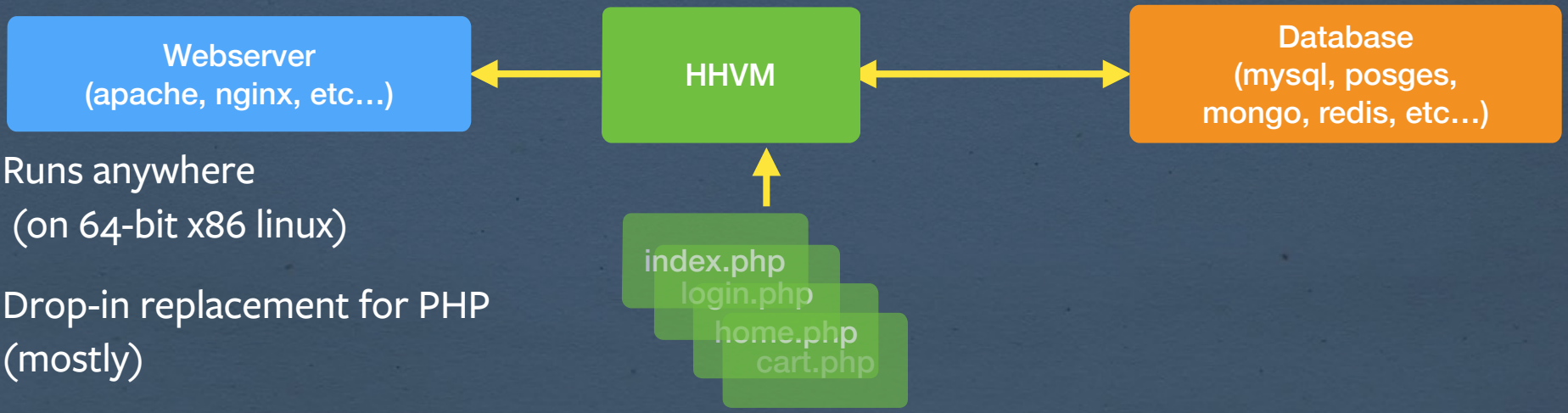


- Runs anywhere (on 64-bit x86 linux)
- Drop-in replacement for PHP (mostly)

HHVM is not a source transformer

That was HPHPc, it's dead.

- Runs your PHP pages live, just like PHP
- Uses standard FastCGI transport



- Runs anywhere (on 64-bit x86 linux)
- Drop-in replacement for PHP (mostly)

HHVM supports all PHP syntax

Tracking HEAD

- Splat & Variadics
- Finally
- Generators
- Namespaces

And some of its own

- Scalar type hint
  (and much much more)
- Async co-routines
- Generics
- Collections (smart arrays)
- XHP (Integrated XHTML)
- User Attributes

HHVM's Parity Gap

- Only about 60% of PHP's unit tests pass
  - Missing Extensions
  - Minor differences in error message output
  - Most private extensions need a rewrite
- 20 top frameworks (and more) do pass
  - So I wouldn't sweat the small stuff

HHVM is easy to install

If you're on Ubuntu

- `deb http://dl.hhvm.com/ubuntu trusty main`
- `apt-get update`
- `apt-get install hhvm (or hhvm-nightly)`
- Provides one binary covering cli, debugger, & fcgi server

Or something Debianish...

- Coming very soon to a Debian near you!

HHVM is buildable

On other linux distros (Mac in interp mode)

- http://hhvm.com/repo/wiki
- gcc 4.8 or later (We love C++11)
- Boost 1.49 or later
- Lots of other dependencies....
- ```
  git clone git@github.com:facebook/hhvm
  cmake .
  make –j
  ```
- hphp/hhvm/hhvm

xkcd.org/303
  ```

Running a FastCGI server

nginx

```
server {
  server_name www.example.com;

  root /var/www;
  index index.php;

  location ~ \.php$ {
    fastcgi_pass unix:/var/run/hhvm.sock
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME
      /var/www$fastcgi_script_name;
    include fastcgi_param;
  }
}
```

HHVM

```
hhvm.server.file_socket = /var/run/hhvm.sock
hhvm.server.type = fastcgi
hhvm.server.source_root = /var/www

hhvm.log.level = Error
hhvm.log.user_log_file = true
hhvm.log.file = /var/log/hhvm-error.log

hhvm.log.access.0.file = /var/log/hhvm-access.log
hhvm.log.access.0.format=%h %l %u %t \"%r\" %>s %b



; hhvm -m daemon -c /etc/hhvm/hhvm.ini -u www-data
```

Open Source means something

HHVM's walled garden

HPHPc in 2010 - 2011

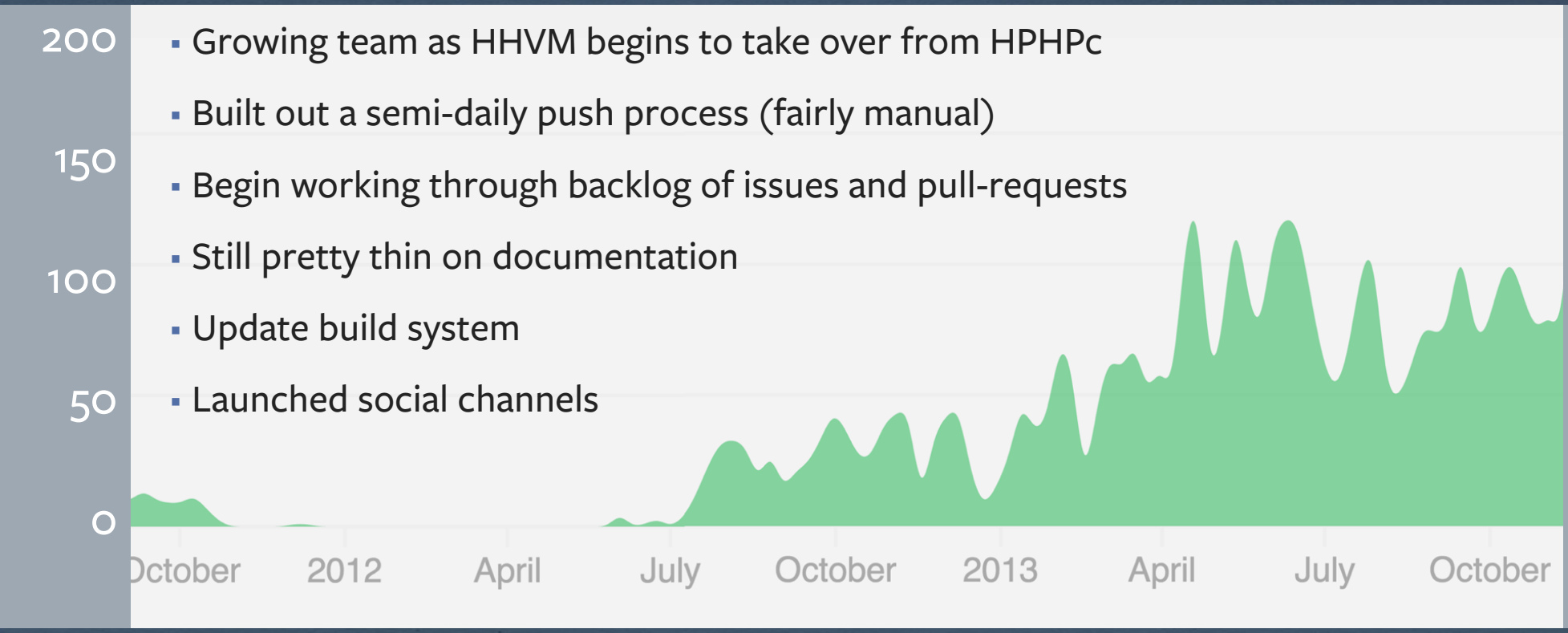
Throw it over the wall (FOSDEM 2010)

- Handful of internal developers
- Irregular “when we get to it” pushes to github
- Little curation of tasks/pull-requests
- Nearly zero documentation
- Brittle build system
- Went dark in late 2011

HipHop in 2012

"Oh right, we have an OSS project..."

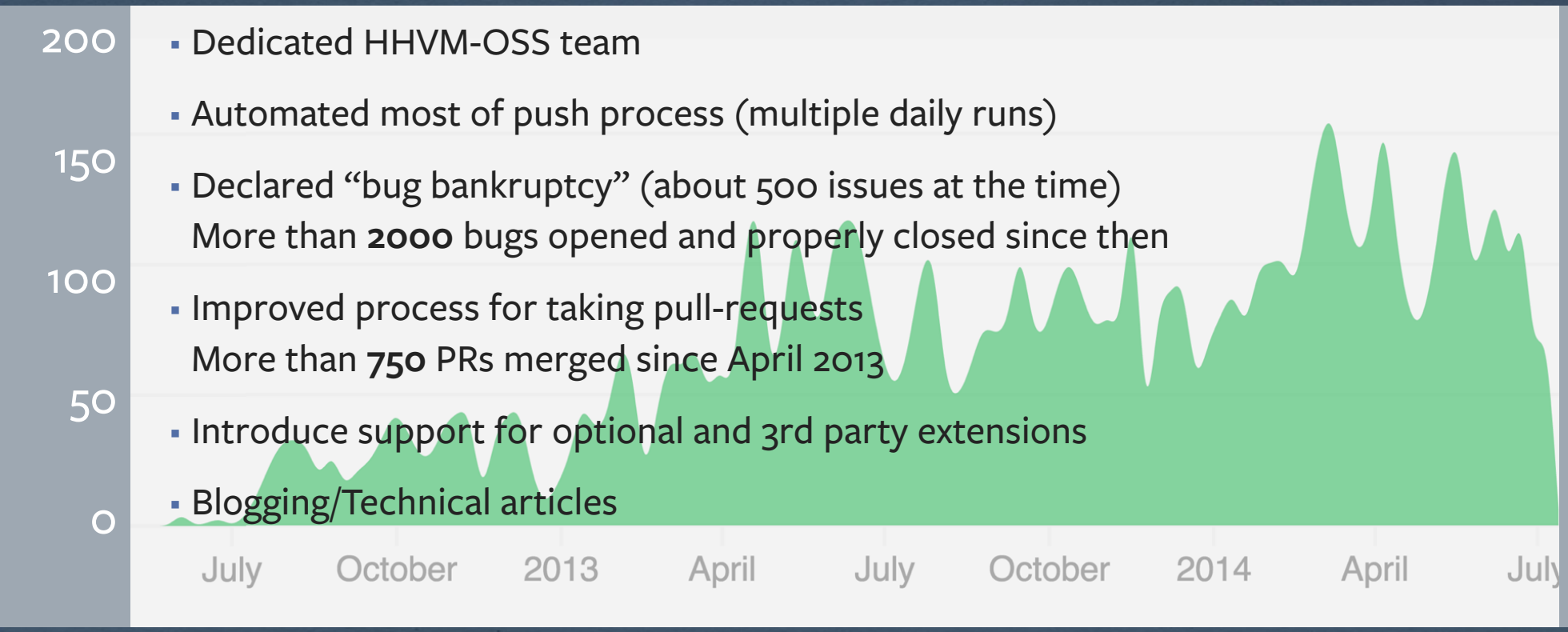
- Growing team as HHVM begins to take over from HPHPc
- Built out a semi-daily push process (fairly manual)
- Begin working through backlog of issues and pull-requests
- Still pretty thin on documentation
- Update build system
- Launched social channels



HHVM in 2013

Making Open Source HHVM a priority

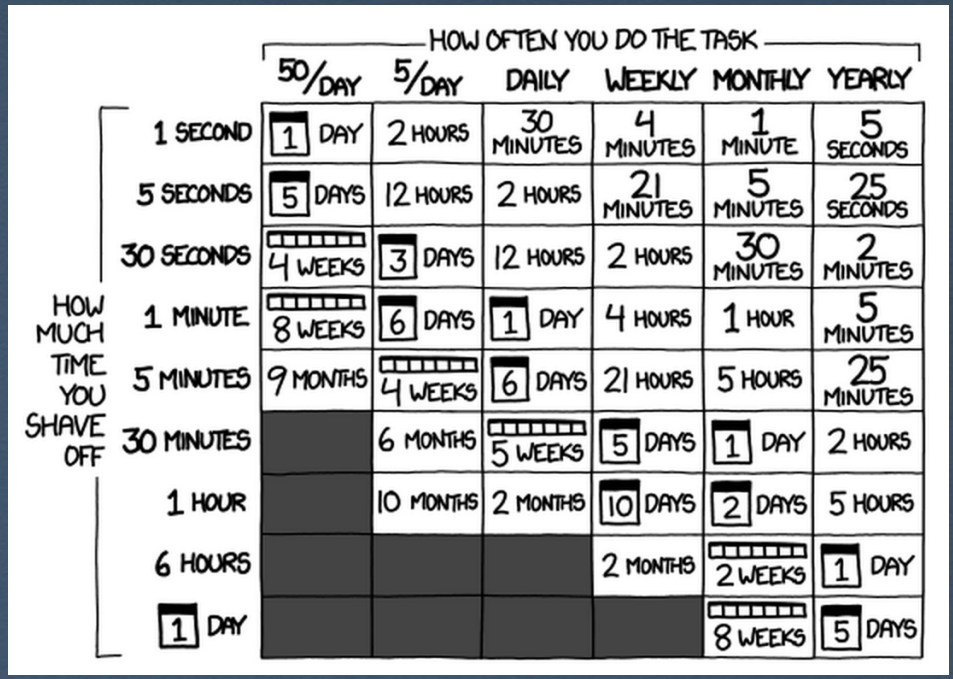
- Dedicated HHVM-OSS team
- Automated most of push process (multiple daily runs)
- Declared “bug bankruptcy” (about 500 issues at the time)
  More than **2000** bugs opened and properly closed since then
- Improved process for taking pull-requests
  More than **750** PRs merged since April 2013
- Introduce support for optional and 3rd party extensions
- Blogging/Technical articles



HHVM in 2014

Bringing down the walls

- OSS build integrated with FB's CI testing
- Immediate* mirroring of FB->Github repo
- Proper documentation! - docs.hhvm.com
- Hack programming language announced
- Automated import of github PRs
- Internal diffs developed in the open
- Inclusion in standard distros**
- Composer support for community-driven extensions**



xkcd.org/1205

* 30 min push window

** Coming in 2nd half of 2014

HHVM Community

HHVM Community

Who's already running it?

- Facebook: 100% Production traffic since Feb 2013
- Baidu: "Most products migrated to HHVM"
- Pocket Rent: github.com/PocketRent/BeatBox
- Escalate.EU: Cloud monitoring/alerts
- GREE: Games Developer

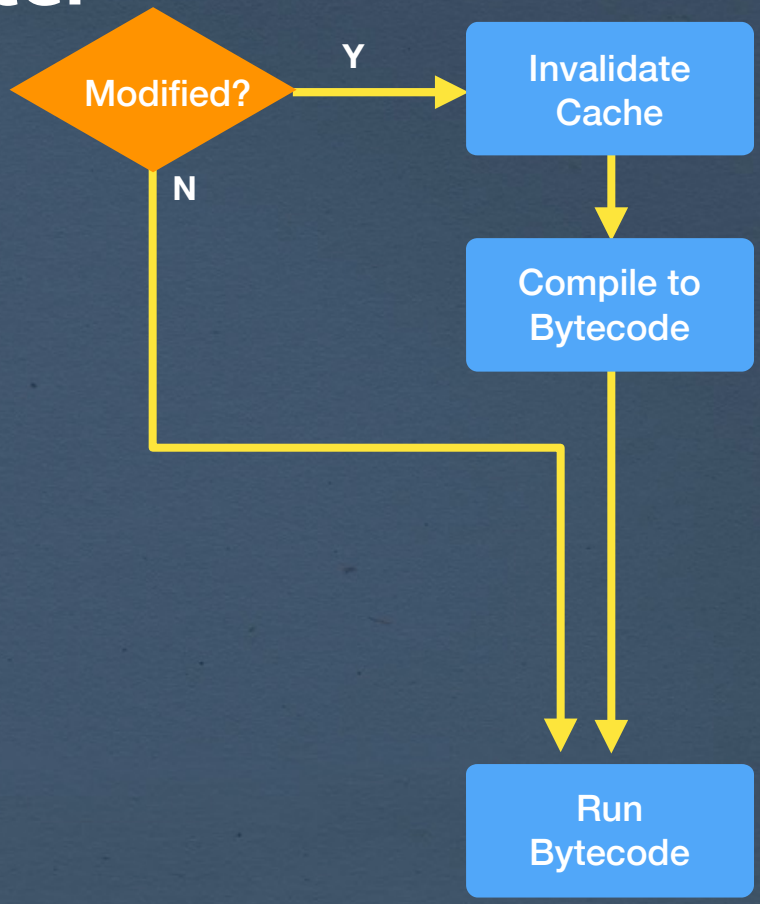
- Clearly a lot of people at least trying: 30K downloads in June

PHP is webscale

HHVM's JIT is the secret sauce

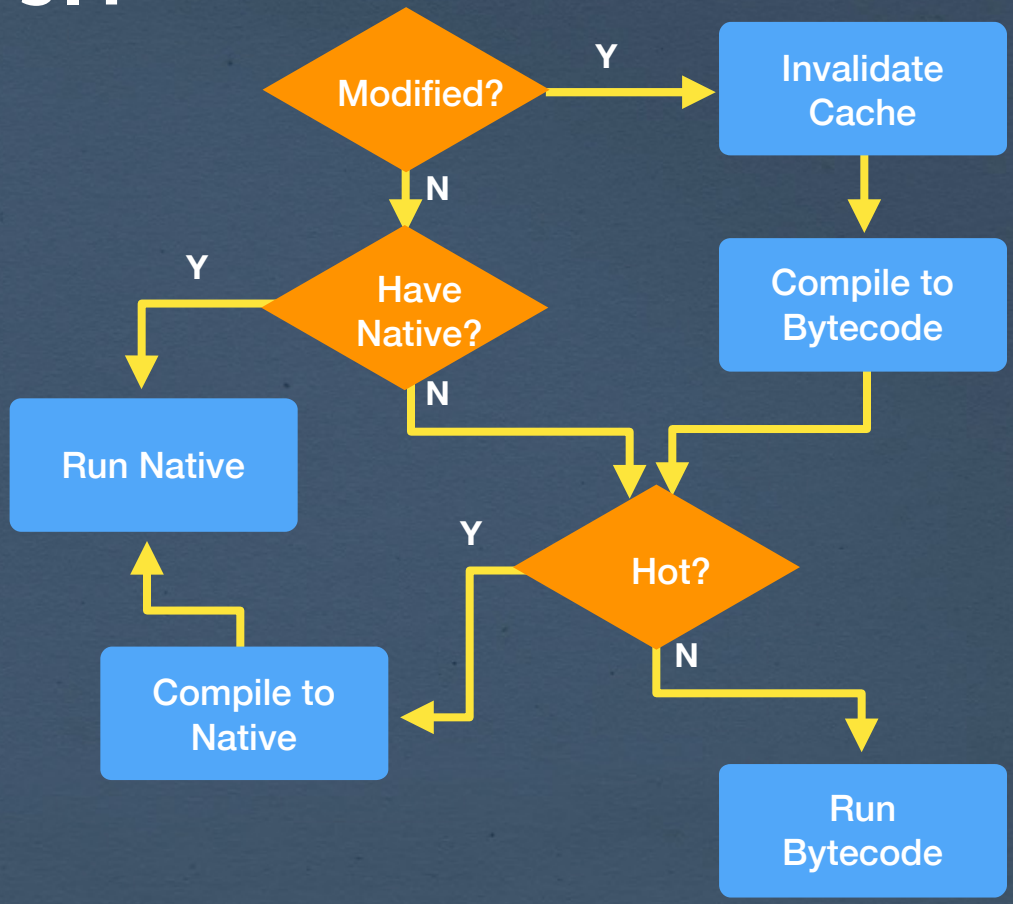
HHVM – Bytecode interpreter

- PHP5 style bytecode execution
- APC-like caching of bytecodes
- Perf about on par with PHP5



HHVM – Native code JIT

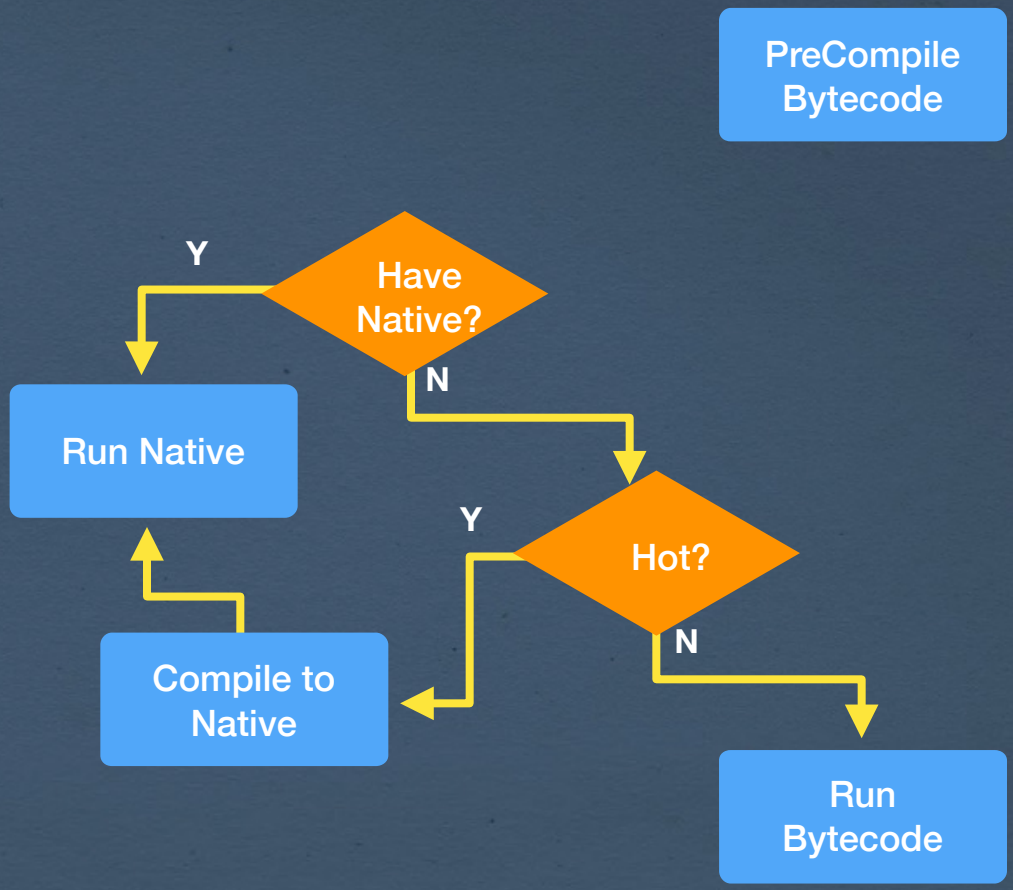
- Bytecodes run a few times “cold”
- Variable type inference
- Hotpath detection
- Transform to native code



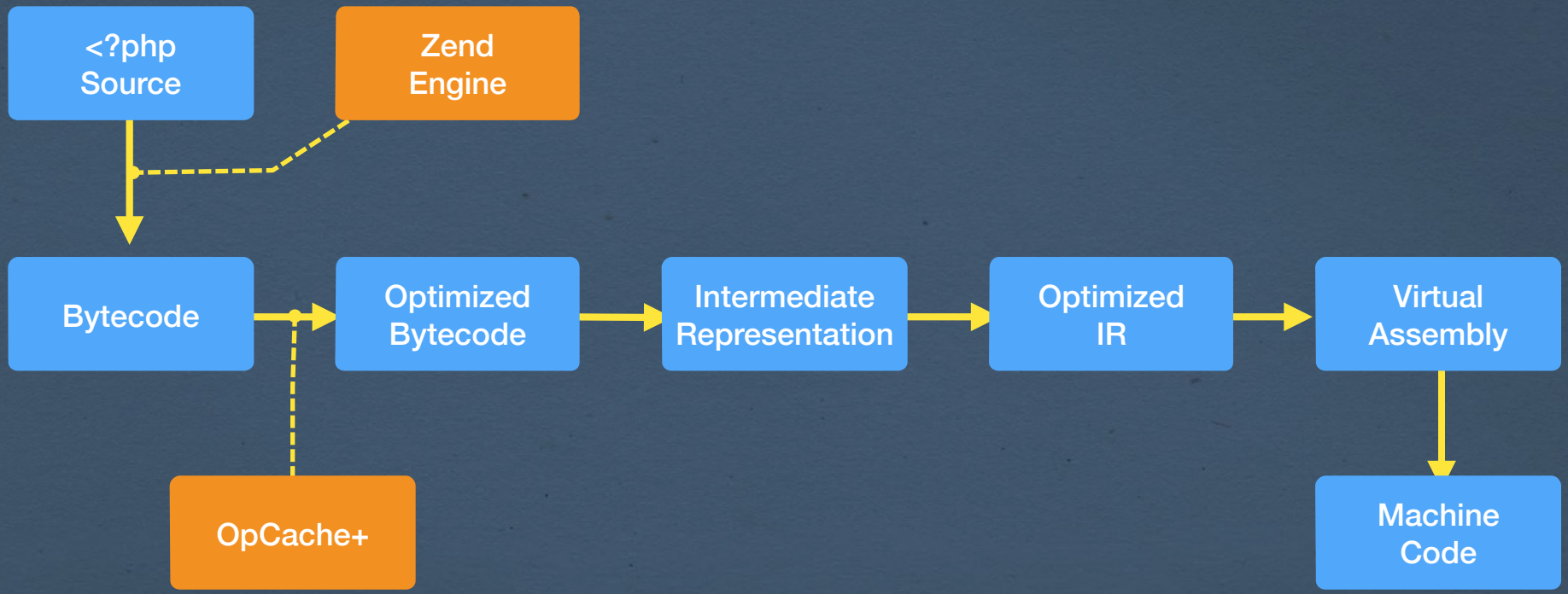
HHVM – Repo Authoritative Mode

- "Production Mode"
- Improved offline pre-analysis
- Assumes no changes

- Another 10-25% perf gain



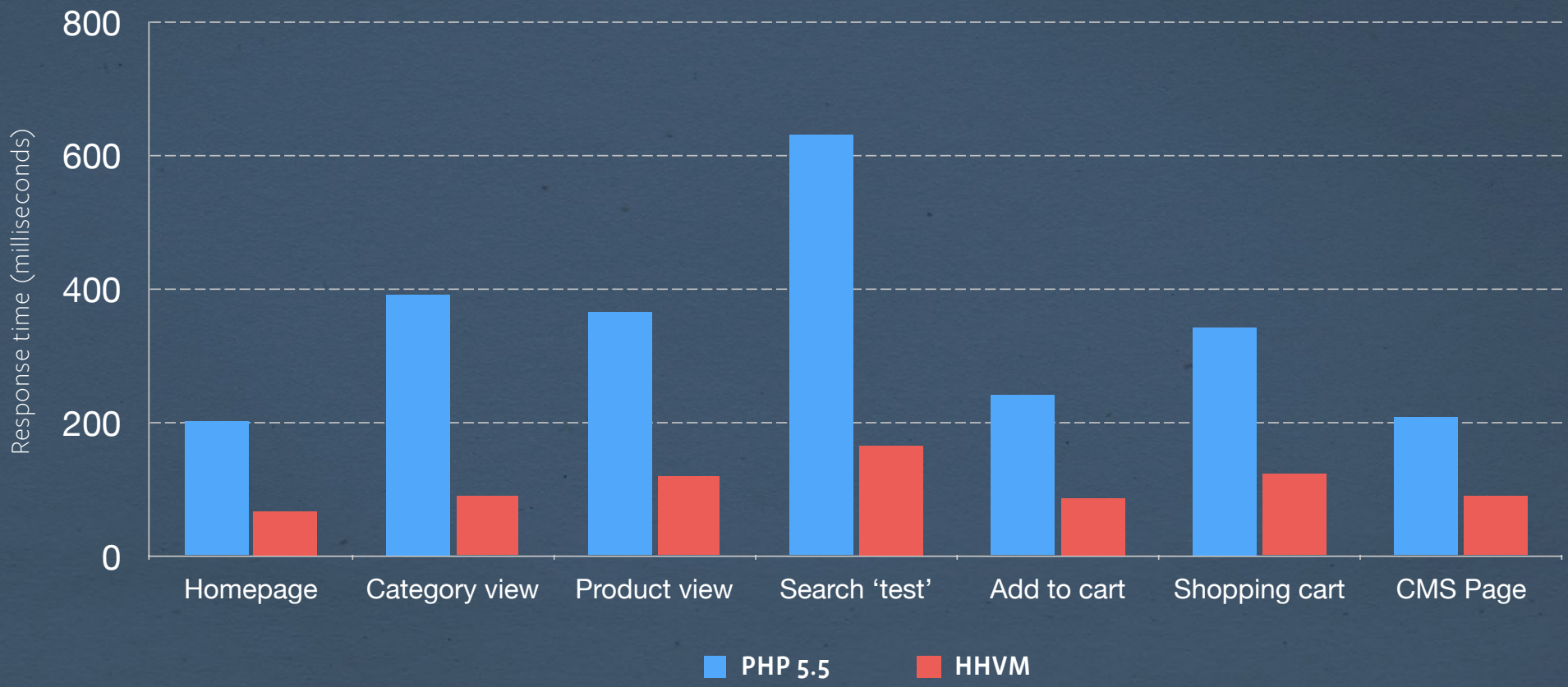
HHVM – Compiling to Native



But don’t take my word for it…

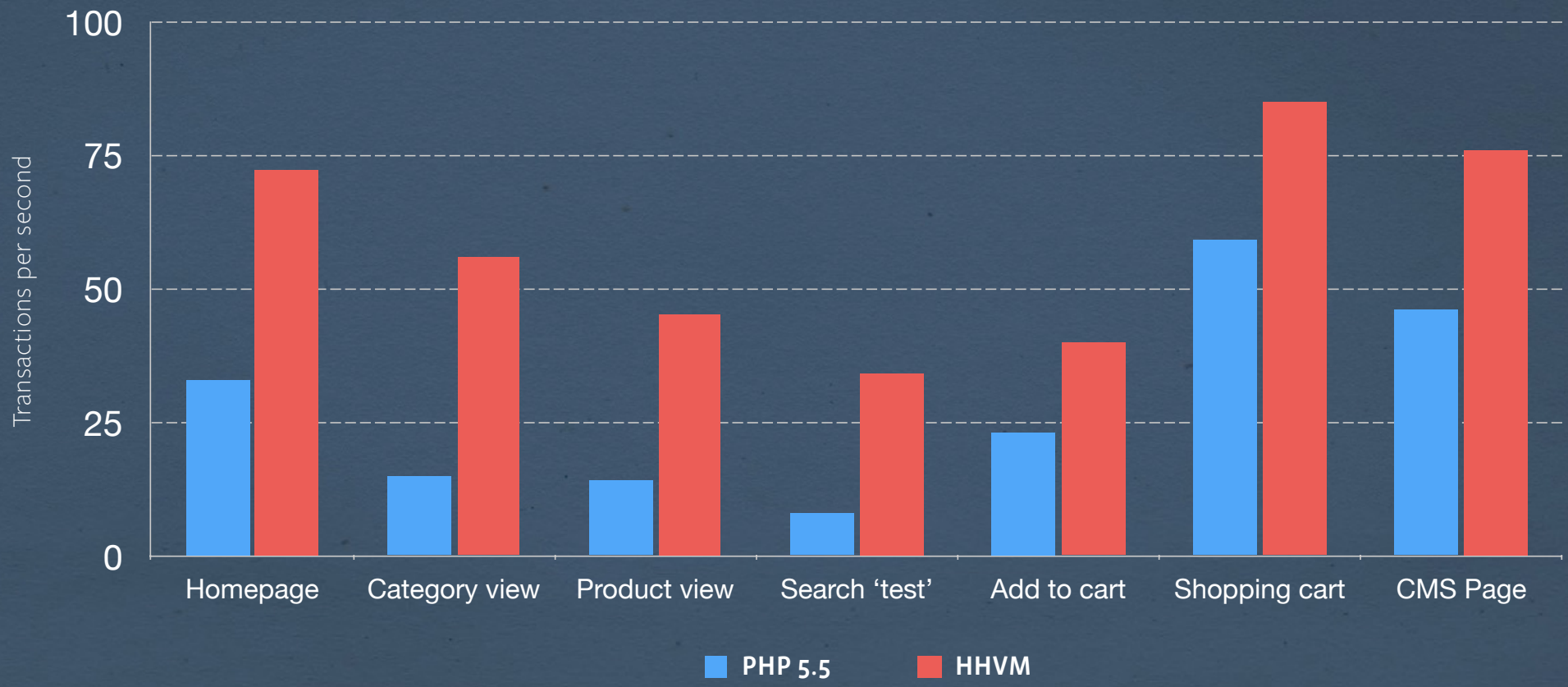
Magento (Daniel Sloof)

Response time (lower is better)



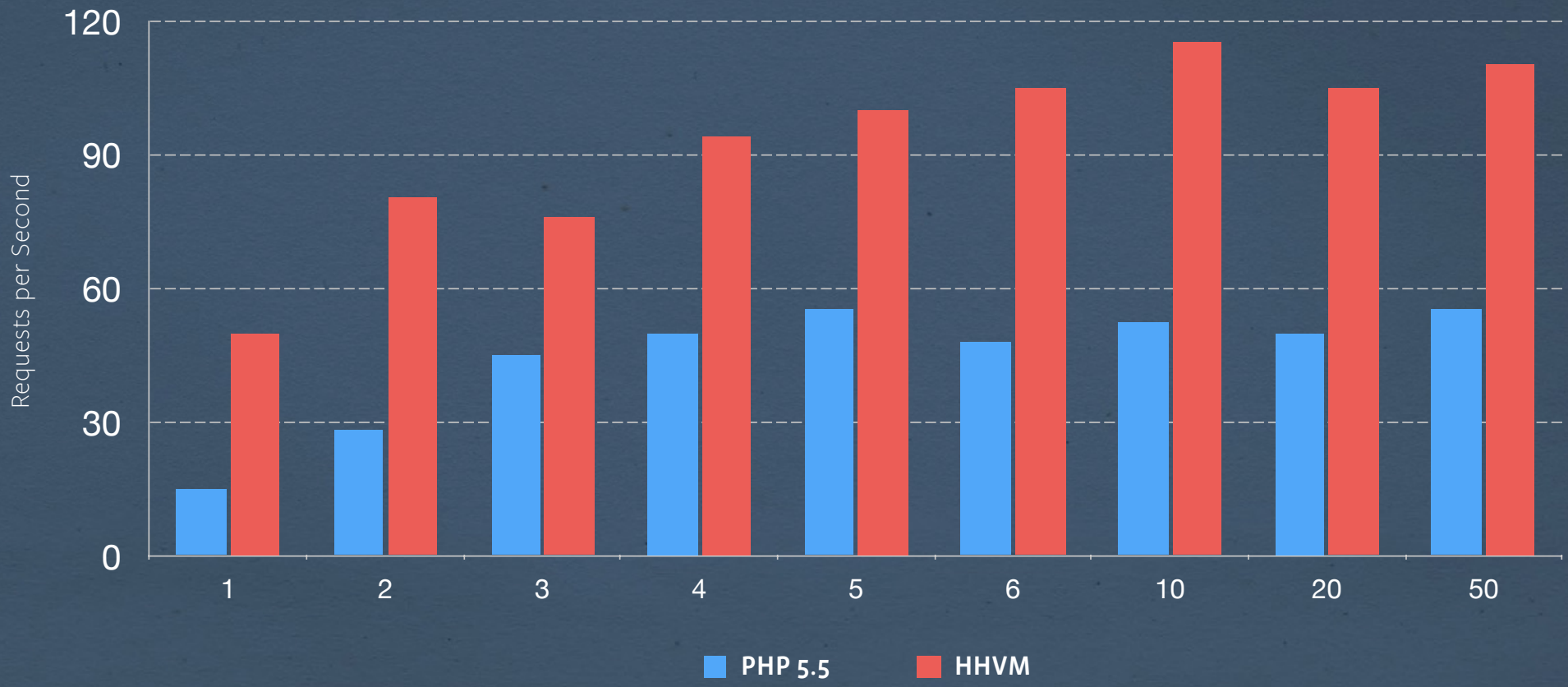
Magento (Daniel Sloof)

Transaction rate (higher is better)



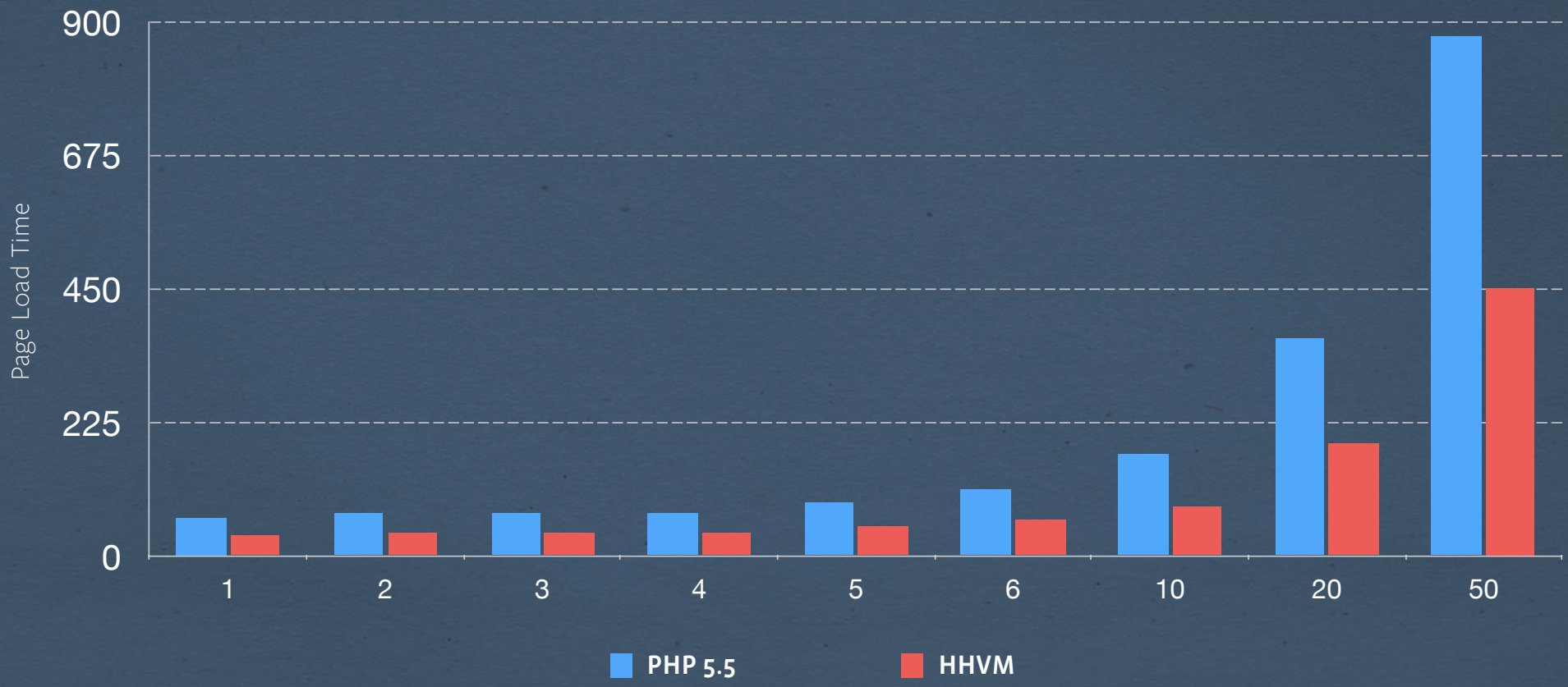
Symfony (Christian Stocker)

Requests per Second (_____



Symfony (Christian Stocker)

Page Load Time (_____



Sure it's fast,
but **what else** can it do?

HACK (yes, it's a horrible name...)

Type Hinting gone mad

- Scalars
  - `bool, int, float, num, string`
- Return typehints
- Typed properties
- Constructor arg promotion
- Specialization
  - `array<int>`
  - `array<string,array<string,stdClass>>`

Static Analysis

- Analyze entire code tree without running
- Report type errors like a strictly typed language
- Fallback on loose typing when needed
  (because that's what PHP is good at)

HACK

```
<?hh

class Foo {
  private int $num = 123;

  public function add(int $delta): Foo {
    $this->num += $delta;
    return $this;
  }

  public function get(): int {
    return $this->num;
  }

  public function __construct(int $num): void {
    $this->num = $num;
  }
}

$f = new Foo(123);
$f->add(456);
$f->add("banana");
```

Basic Hack

- Static Analyzer traces value through whole program
- Looks for mismatches and reports errors
- "Foo::add() expected int"
  "Foo::add() received string"

HACK – Constructor arg promotion

```
<?hh

class Foo {

  public function add(int $delta): Foo {
    $this->num += $delta;
    return $this;
  }

  public function get(): int {
    return $this->num;
  }

  public function __construct(private int $num)
                                      :void { }
}

$f = new Foo(123);
echo $f->add(456)->get();
```

Avoid repeated patterns

- public/protected/private as __construct arg modifiers
- Declares the property and initializes from value
- Less boilerplate, more coding

HACK – Generics

```
<?hh

class Foo<T> {
  public function add(T $delta): Foo {
    $this->num += $delta;
    return $this;
  }

  public function get(): T {
    return $this->num;
  }

  public function __constructor(private T $num)
                                    :void { }

}

$i = new Foo(123); // Makes a Foo<int>
echo $i->add(456)->get();

$f = new Foo(3.14); // Makes a Foo<float>
echo $f->add(2.17)->get();
```

Specialize common code

- T is replaced w/ specialization type at instantiation
- Type-checker propagates replacement through generic
- *Less boilerplate, more coding*

HACK – Collections

```
<?hh

function foo(Vector<int> $nums,
             Set<string> $names,
             Map<int,string> $numNameMap,
             FrozenVector<int> $nums2): bool {

  foreach($nums as $num) {
    $mappedName = $numNameMap[$num];
    if ($names->contains($mappedName)) {
      return true;
    }
  }

  if ($nums2->count() == 0) return true;
  return false;
}
```

Specialized array objects

- `Vector, Set, Map, StableMap`
- `Frozen*`
  `Mutable* (default)`
- Support ArrayAccess
- Extensive library of methods

Async Functions

```
<?hh

async function getPage($url) :
           Awaitable<string> {
  $fp = fopen($url, 'r');
  return await async_get_contents($fp);
}

// Sends all requests in parallel,
// blocks for response
$pages = await [
  getPage('http://php.net'),
  getPage('http://example.com'),
  getPage('http://hhvm.com'),
];
```

Cooperative Multitasking

- Parallelizing made easy
- Allow multiple functions to run cooperatively
- While one is blocking, the other executes

hh_server convert - The Hackificator

```
<?hh

function f(     $x):       {
  if ($x) {
    return new Foo();
  } else {
    return null;
  }
}
```

```
function g($y):       {
  f(42);
  return f(103);
}

function h():       {
  g('hello world');
  g(44);
}
```

hh_server convert - The Hackificator

```
<?hh

function f(@int $x): @?Foo {
  if ($x) {
    return new Foo();
  } else {
    return null;
  }
}
```

```
function g($y):       {
  f(42);
  return f(103);
}

function h(): @void {
  g('hello world');
  g(44);
}
```

hh_server convert - The Hackificator

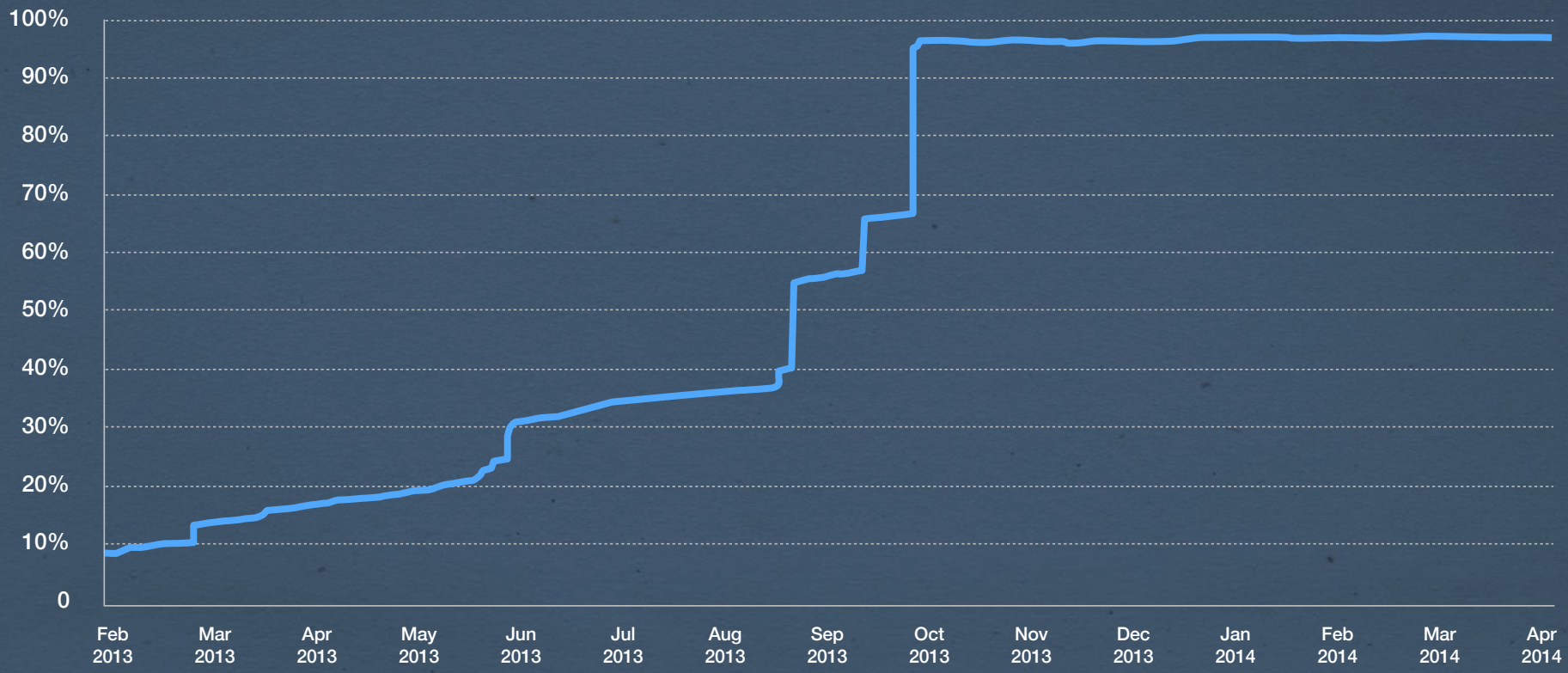
```
<?hh

function f(@int $x): @?Foo {
  if ($x) {
    return new Foo();
  } else {
    return null;
  }
}
```

```
function g($y): @?Foo {
  f(42);
  return f(103);
}

function h(): @void {
  g('hello world');
  g(44);
}
```

Hackificating



User Attributes

```
<?php

<<Author("sgolemon"),Clowny>>
function life() {
  return 42;
}

$rf = new ReflectionFunction('life');
print_r($rf->getAttributes());
print_r($rf->getAttribute('Author'));


Array (
  [Author] => Array (
    [0] => sgolemon
  )
  [Clowny] => Array ( )
)
Array (
  [0] => sgolemon
)
```

User-defined metadata

- Arbitrary labels surfaced via reflection
- Basically what it says on the tin

XHP – XHtml for PHP

```
<?php

  if (validate($_GET['id'], $_GET['pwd']))
    loginAndRedirectToHome();
    exit;
  }
?><html>
 <head><title>Login</title></head>
 <body>
  <form>
   ID: <input type=text name="username"
                                value="{$_GET['id']}" />
    Pass: <input type=password name="password"
                                value="{$_GET['pwd']}" />
  </form>
  </body>
</html>
```

Easy noob mistake

- http://example.com/?id="><script>alert("Gotcha!")</script>

- Trivial vector for stealing user's cookies (and login credentials)

XHP – XHtml for PHP

```
<?php

 if (validate($_GET['id'], $_GET['pwd']))
   loginAndRedirectToHome();
   exit;
 }

 include 'xhp/init.php';

 $form =
   <form>
     ID: <input type=text name="username"
                              value={$_GET['id']} />
    Pass: <input type=password name="password"
                              value={$_GET['pwd']} />
  </form>;

 echo <html>
          <head><title>Login</title></head>
          <body>{$form}</body>
        </html>;
```

Markup as 1st class syntax

- Parser can tell what's userdata, and what isn't
- Catch most XSS automatically
- Formalizes component modularity

XHP – XHtml for PHP

```
<?php

class :mysite:footer extends :xhp:html-element {
  attribute enum {'en','es'} lang = 'en';
  attribute string prefix;
  public function stringify() {
    $home = ($this->lang == 'es') ? 'Casa' : 'Home';
    $footer =
      <div class="footer">
        <a href={$this->prefix}>$home</a>
        <a href={$this->prefix}/contact.php>
                    Contact Us</a>
                    etc...
      </div>;
    return $footer;
  }
}

echo <html>
        <body>
          Blah Blah
          <mysite:footer lang="es" />
        </body>
      </html>;
```

Markup as 1st class syntax

- Custom components
- Compose from other tags or generate custom xhtml

XHP – XHtml for PHP

```
<?php

class :blink extends :xhp:html-element {
  category %flow, %phrase;
  children (pcdata | %phrase);
  protected $tagName = 'blink';
}

echo <html>
        <body>
          <blink>UNDER CONSTRUCTION</blink>
          <mysite:footer lang="es" />
        </body>
      </html>;
```

Markup as 1st class syntax

- Can also contain other tags
- Default :xhp:html-element behavior makes simple tags easy
- Replace :blink's stringify to use javascript since browsers disable

HPHPd – HPHP Debugger

```
Welcome to HipHop Debugger!
Type "help" or "?" for a complete list of commands.

hphpd> =str_rot13("Hello World!");
Uryyb Jbeyq!
hphpd> $fp = fopen("php://stdout", "w");

hphpd> fwrite($fp, "Example\n");
Example

hphpd> $a = 123;

hphpd> $b = 456;

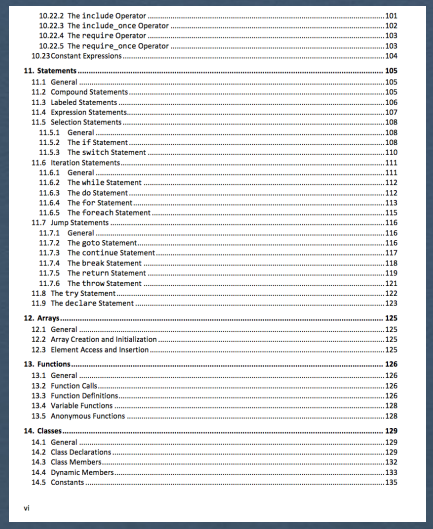
hphpd> =($a+$b);
579
hphpd>
```

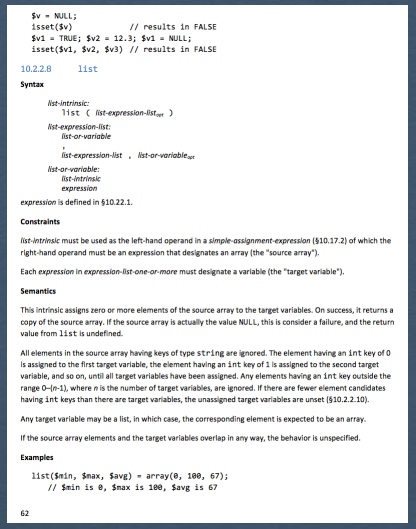
- Interactive shell
- GDB-like debugging
- Standalone or w/ server
- Breakpoints
- Watches
- Macros
- xdebug - coming soon

One more thing…

PHP Language Specification

- Formal standard - Defines "Proper PHP"
- Based on ZendEngine 2.6 (PHP5.6) behavior, focus on syntax
- Living document, curated by PHP Community




```
$v = NULL;
isset($v)           // results in FALSE
$v1 = TRUE; $v2 = 12.3; $v1 = NULL;
isset($v1, $v2, $v3) // results in FALSE
```

10.2.2.8 list

Syntax

list-intrinsic:
    list (*list-expression-list*$_{opt}$)

list-expression-list:
    list-or-variable
    ,
    list-expression-list , *list-or-variable*$_{opt}$

list-or-variable:
    list-intrinsic
    expression

expression is defined in §10.22.1.

Constraints

list-intrinsic must be used as the left-hand operand in a *simple-assignment-expression* (§10.17.2) of which the right-hand operand must be an expression that designates an array (the "source array").

Each *expression* in *expression-list-one-or-more* must designate a variable (the "target variable").

Semantics

This intrinsic assigns zero or more elements of the source array to the target variables. On success, it returns a copy of the source array. If the source array is actually the value NULL, this is consider a failure, and the return value from list is undefined.

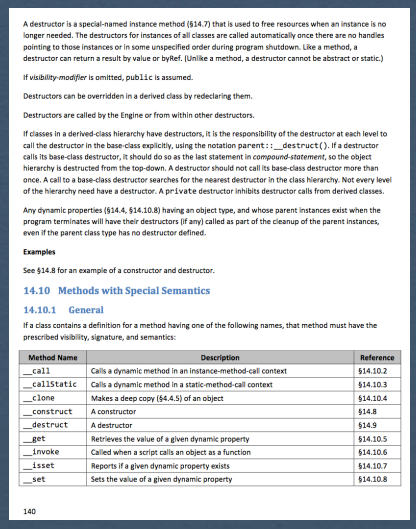
All elements in the source array having keys of type string are ignored. The element having an int key of 0 is assigned to the first target variable, the element having an int key of 1 is assigned to the second target variable, and so on, until all target variables have been assigned. Any elements having an int key outside the range 0–(*n*-1), where *n* is the number of target variables, are ignored. If there are fewer element candidates having int keys than there are target variables, the unassigned target variables are unset (§10.2.2.10).

Any target variable may be a list, in which case, the corresponding element is expected to be an array.

If the source array elements and the target variables overlap in any way, the behavior is unspecified.

Examples

```
list($min, $max, $avg) = array(0, 100, 67);
    // $min is 0, $max is 100, $avg is 67
```



A destructor is a special-named instance method (§14.7) that is used to free resources when an instance is no longer needed. The destructors for instances of all classes are called automatically once there are no handles pointing to those instances or in some unspecified order during program shutdown. Like a method, a destructor can return a result by value or byRef. (Unlike a method, a destructor cannot be abstract or static.)

If *visibility-modifier* is omitted, public is assumed.

Destructors can be overridden in a derived class by redeclaring them.

Destructors are called by the Engine or from within other destructors.

If classes in a derived-class hierarchy have destructors, it is the responsibility of the destructor at each level to call the destructor in the base-class explicitly, using the notation parent::__destruct(). If a destructor calls its base-class destructor, it should do so as the last statement in *compound-statement*, so the object hierarchy is destructed from the top-down. A destructor should not call its base-class destructor more than once. A call to a base-class destructor searches for the nearest destructor in the class hierarchy. Not every level of the hierarchy need have a destructor. A private destructor inhibits destructor calls from derived classes.

Any dynamic properties (§14.4, §14.10.8) having an object type, and whose parent instances exist when the program terminates will have their destructors (if any) called as part of the cleanup of the parent instances, even if the parent class type has no destructor defined.

Examples

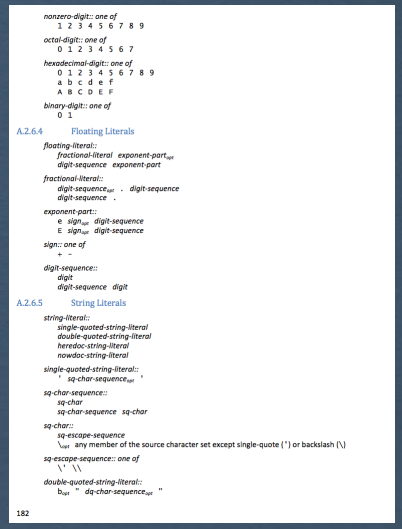
See §14.8 for an example of a constructor and destructor.

14.10 Methods with Special Semantics

14.10.1 General

If a class contains a definition for a method having one of the following names, that method must have the prescribed visibility, signature, and semantics:

Method Name	Description	Reference
__call	Calls a dynamic method in an instance-method-call context	§14.10.2
__callStatic	Calls a dynamic method in a static-method-call context	§14.10.3
__clone	Makes a deep copy (§4.4.5) of an object	§14.10.4
__construct	A constructor	§14.8
__destruct	A destructor	§14.9
__get	Retrieves the value of a given dynamic property	§14.10.5
__invoke	Called when a script calls an object as a function	§14.10.6
__isset	Reports if a given dynamic property exists	§14.10.7
__set	Sets the value of a given dynamic property	§14.10.8


nonzero-digit:: one of
    1 2 3 4 5 6 7 8 9

octal-digit:: one of
    0 1 2 3 4 5 6 7

hexadecimal-digit:: one of
    0 1 2 3 4 5 6 7 8 9
    a b c d e f
    A B C D E F

binary-digit:: one of
    0 1

A.2.6.4 Floating Literals

floating-literal::
    fractional-literal exponent-part$_{opt}$
    digit-sequence exponent-part

fractional-literal::
    digit-sequence$_{opt}$. *digit-sequence*
    digit-sequence .

exponent-part::
    e *sign*$_{opt}$ *digit-sequence*
    E *sign*$_{opt}$ *digit-sequence*

sign:: one of
    + -

digit-sequence::
    digit
    digit-sequence digit

A.2.6.5 String Literals

string-literal::
    single-quoted-string-literal
    double-quoted-string-literal
    heredoc-string-literal
    nowdoc-string-literal

single-quoted-string-literal::
    b$_{opt}$ ' *sq-char-sequence*$_{opt}$ '

sq-char-sequence::
    sq-char
    sq-char-sequence sq-char

sq-char::
    sq-escape-sequence
    any member of the source character set except single-quote (') or backslash (\)

sq-escape-sequence:: one of
    \' \\

double-quoted-string-literal::
    b$_{opt}$ " *dq-char-sequence*$_{opt}$ "

Questions / Resources

- http://hhvm.com/repo
- http://hhvm.com/blog
- http://hhvm.com/twitter
- http://docs.hhvm.com

- hphp/doc in git repository
  for Options and Technical... Stuff

- Freenode / #hhvm
- http://hhvm.com/fb/page
- http://hhvm.com/fb/general

- @HipHopVM
- @HackLang