TAKING PHP SERIOUSLY

Keith Adams

Facebook

Strange Loop 2013

Why PHP?

What this talk is

- Experience report
- Apologia
- Qualified advocacy
- A surprise.

What this talk is *not*

- “Network effects”/”Legacy”
- “Right tool for the job”
  - tautological
  - some tools really are *bad*
  - PHP might be such a tool
- “Worse is better”
  - *pace* Richard Gabriel
  - Better is better
  - Most people think of UNIX as “better” nowadays

Recent changes

- Traits (ala Scala)
- Closures
- Generators (`yield` statement)

- The HipHop VM (hhvm) is fast
  - https://github.com/facebook/hiphop-php/
  - https://www.hhvm.com
- ...and we want it to run your code
  - http://www.hhvm.com/blog/?p=875

Conventional Wisdom on PHP

- “PHP: A fractal of bad design”
  - http://me.veekun.com/blog/2012/04/09/php-a-fractal-of-bad-design/
- “[ ] You have reinvented PHP better, but that’s still no justification”
  - http://colinm.org/language_checklist.html
- Etc.

And yet...

- A lot of software that has changed the world has been rendered in PHP
    - Mediawiki
    - Facebook
    - Wordpress
    - Drupal

- This is at least *interesting*
- Should they *really* have been written in Haskell?
- Does PHP make projects more or less successful?

Facebook's PHP Codebase

- *x* * 10^5 *files*
- *y* * 10^7 LoC
- 10 releases per week

- Anecdotally, good engineers are *astonishingly* productive in PHP

The Case Against PHP

- Unexpected behaviors

```
$x / 0                  // => bool(false)

“11abcd” + “1xy” // => int(12)

“0123” + “3456”  // => int(3579)
“0123” | “3456”  // => string(“3577”)
```

The Case Against PHP (2)

- Schizophrenia about value/reference semantics

```
/*
 * Probably copy $a into foo’s 0’th param.
 * Unless $a is a user-defined object; and unless
 * foo’s definition specifies that arg 0 is by
 * reference.
 */
foo($a);
```

The Case Against PHP (3)

- Reliance on reference-counting
  - String, array need O(1) logical copies
  - User-defined classes have destructors that run at a deterministic time
  - Some programs use the RAII idiom from C++
- Heavily constrains implementation

The Case Against PHP (4)

- Inconsistent, dangerous standard library
    - array_map vs. array_reduce argument orders
    - array_merge
    - mysql_escape_string vs. (*sigh*) mysql_real_escape_string

The Case Against PHP: “Guilty”

- It’s all true!
- These are “unforced errors”
- Most other languages do better
- You would want to avoid them in a PHP Reboot

In Defense of PHP

- PHP gets three important things really right
  - Programmer workflow
  - State
  - Concurrency

Workflow

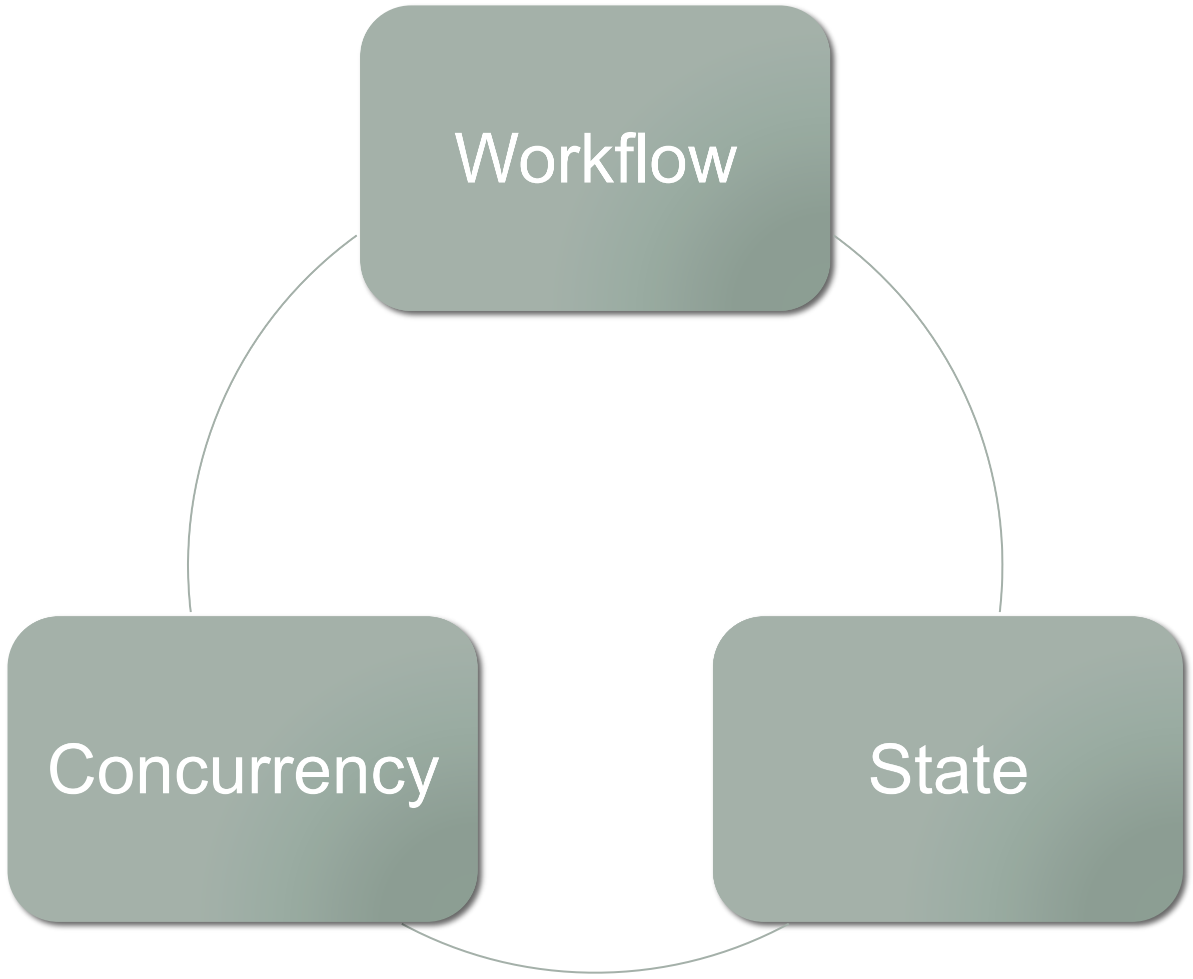
- Save, reload-the-page
- Short feedback cycle
- Optimizes most precious resource of all: programmer short-term memory

State

- PHP requests always start with empty heap, namespace
- Cross-request state must be saved explicitly
    - Filesystem, memcache, APC
    - Affirmative virtue
- Typical FB requests spend 10ms initializing
- Reduces the cost of bugs
    - Requests interact in limited ways
    - Natural boundary for failure isolation

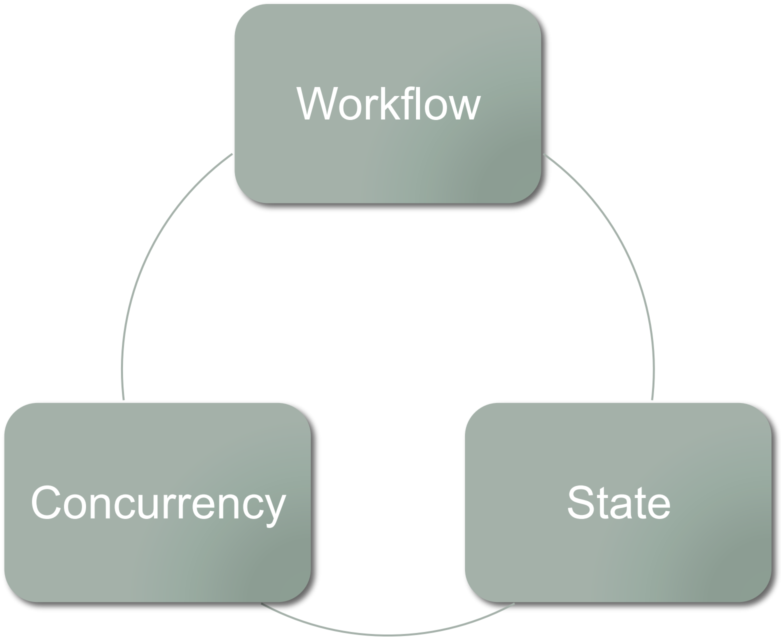
Concurrency

- PHP requests execute in a single thread
- Concurrency happens via recursive web requests
  - shared-nothing
  - inputs/outputs copied
- Limits PHP's applicable domain
  - That's actually good.


Workflow
Concurrency
State

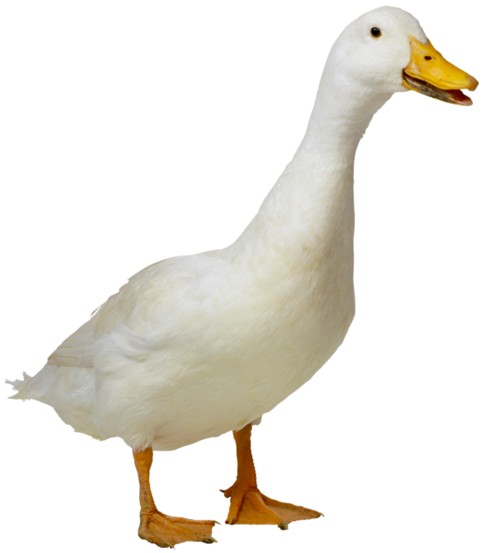
The limits of conscious design

- *Discovered* or *invented?*
- Shrug
- In my opinion, more important than PHP's problems
- They're not available anywhere else



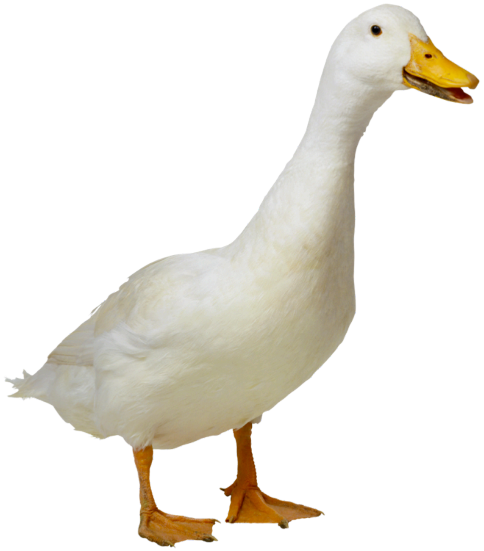
Pushing PHP further

- PHP engineer dare: rename this method
- Reorder the parameters for this method
- Remove this method that *we think* is not called anywhere



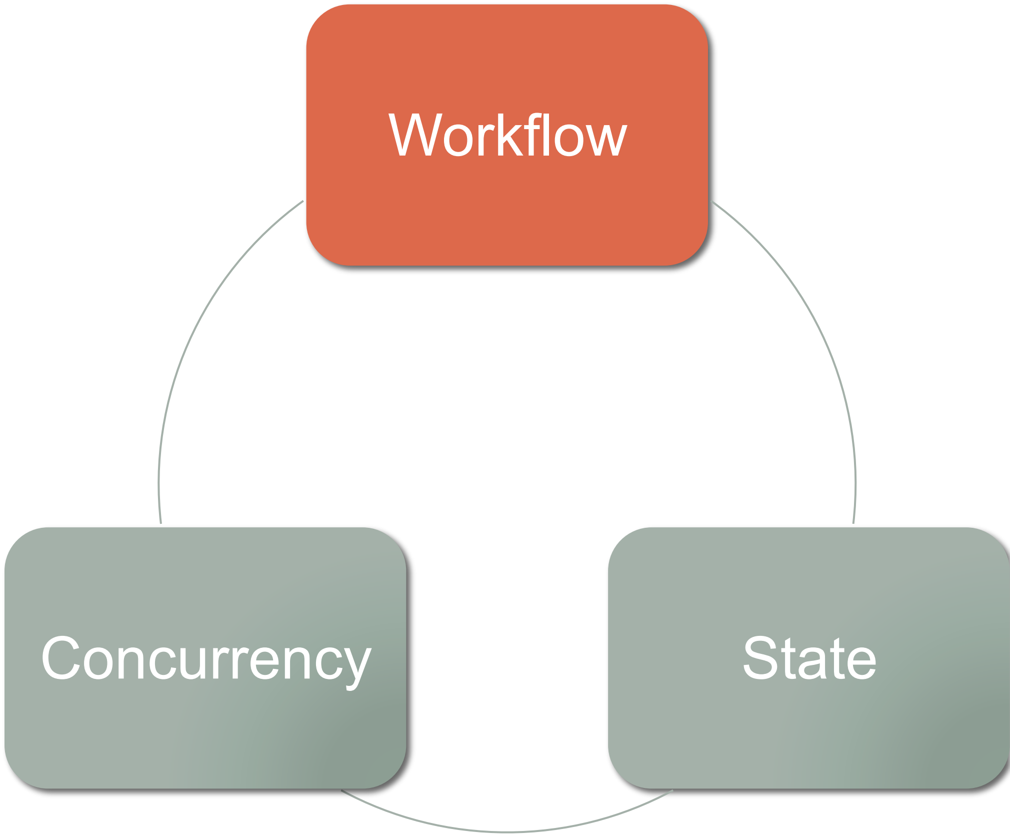
Pushing PHP further (2)

- Enforce invariants:
  - Only escaped strings are passed to `build_query`
  - A certain array() maps strings to Widgets



Wait...

- A static type system?
- Verbose types, or incomprehensible error messages
- Either way hoses programmer productivity
- Millions of lines to migrate



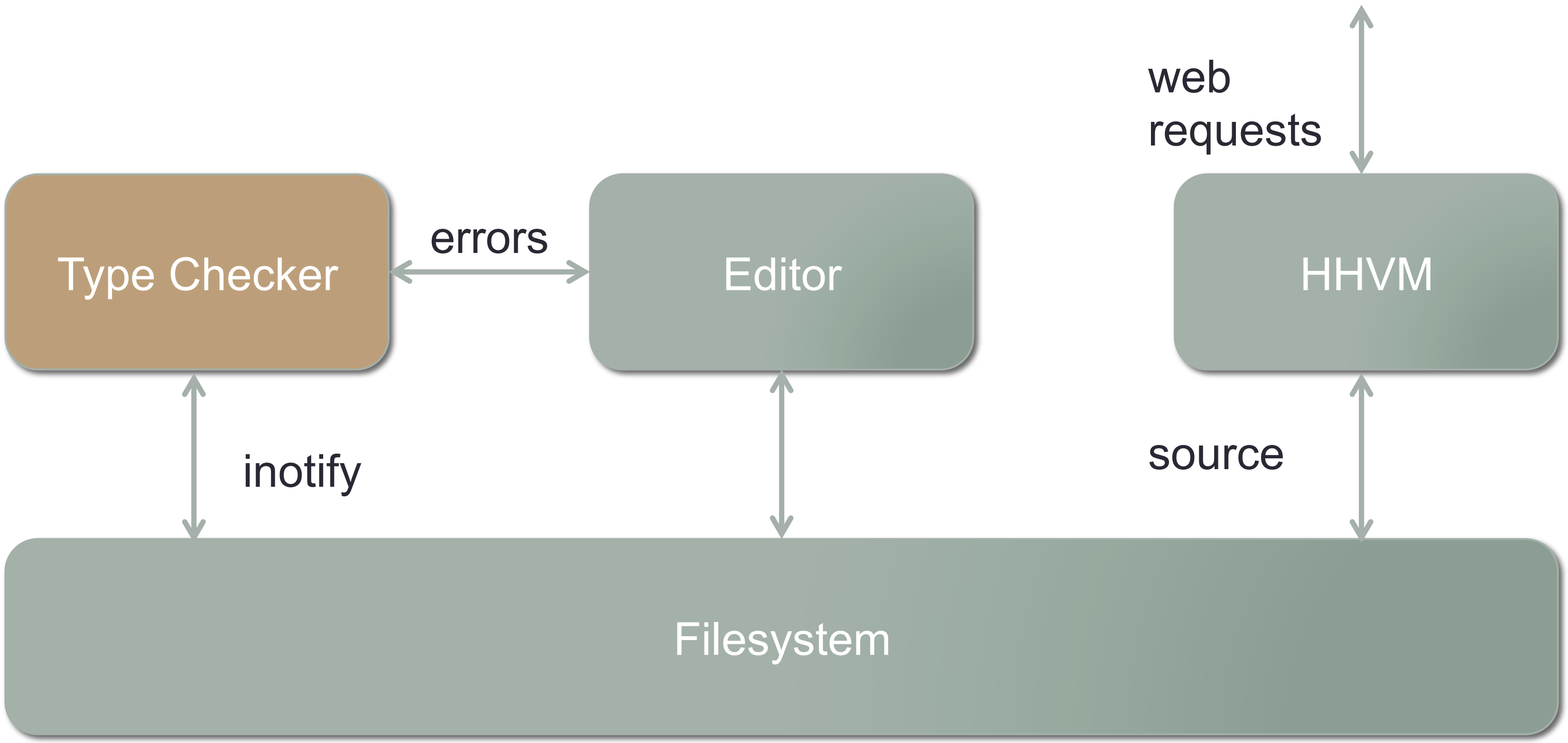
We think we've solved this problem

- Introducing Hack
- Gradual typing for PHP
- Novel type inference system
- Real-time type-checking preserves PHP workflow
- Credit: Julien Verlaguet

Hack

- Opt into typing via <?hh (instead of <?php)
- `<?hh // strict`
  - Almost-totally sound analysis
  - Requires transitive closure of code has been hackified
- `<?hh`
  - Tolerates missing annotations
  - Assumes undeclared classes/functions exist, behave as implied by any types
- Disallows most “silly” PHP-isms

Hack implementation



Changes from PHP

```
<?hh
class Point2 {
  public float $x, $y;
  function __construct(float $x, float $y) {
    $this->x = $x;
    $this->x = $y;
  }
}
```

Changes from PHP

```
<?hh
class Point2 {
  public float $x, $y;
  function __construct(float $x, float $y) {
    $this->x = $x;
    $this->x = $y; // Whoopsy. Didn’t init y
  }
}
```

Changes from PHP

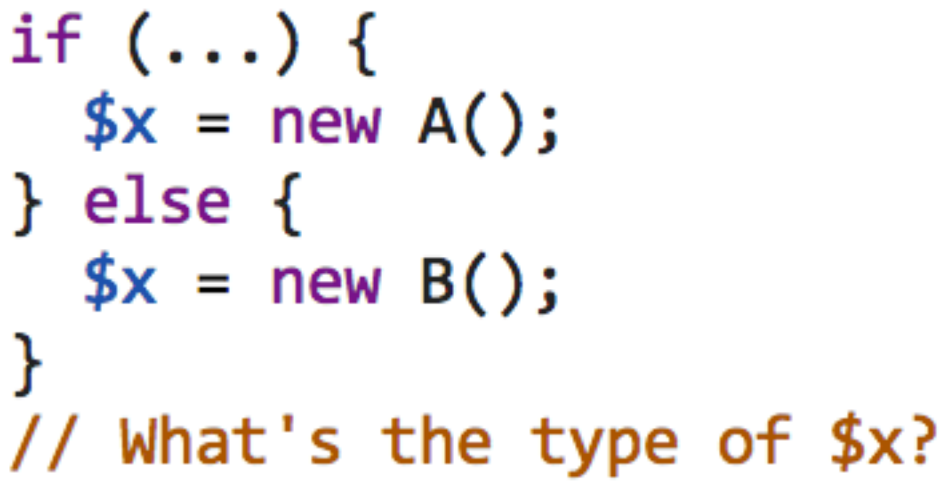
```
<?hh
...
function meanOrigDistance(Point $p, Point $q)
  : float {
  $distf = function(Point $p) : float {
    return sqrt($p->x * $p->x + $p->y * $p->y);
  };
  $pdist = $distf($p);
  $qdist = $distf($q);
  return ($pdist + $qdist) / 2;
}
```

Hack Type Cheatsheet

- Base PHP types: `int, MyClassName, array, ...`
- Nullable: `?int, ?MyClassName`
- Mixed: anything (careful)
- Tuples: `(int, bool, X)`
- Closures: `(function(int): int)`
- Collections: `Vector<int>, Map<string, int>`
- Generics: `A<T>, foo<T>(T $x): T`
- Constraints: `foo<T as A>(T $x): T`

Hack Type Inference (1)

- Let’s infer the type of `$x`:



```
if (...) {
  $x = new A();
} else {
  $x = new B();
}
// What's the type of $x?
```

Hack Type Inference (2)

- How does a type-system normally work?
  - Type-variables are introduced
  - A unification algorithm solves the type-variables (usually noted α)

```
                                    type($x) = α
if (…) {
    $x = new A();          →        unify(α, A) => α = A
} else {
    $x = new B();          →        unify(α, B) => α = B
}                                                   ERROR
```

Type inference in Hack

- Hack introduces unresolved types (noted U)

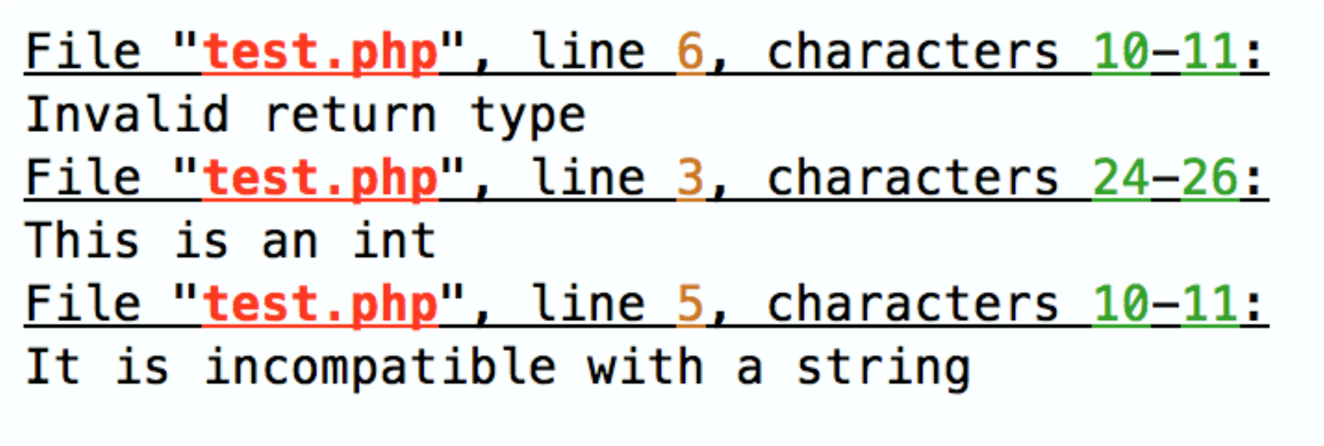
```
if (…) {
    $x = new A();
} else {
    $x = new B();
}

takesAnIFace($x);
```

```
type($x)  = α = U()

$x = α = U(A);

$x = α = U(A, B);
```

\$x = α = U(A, B) = IFace
with (A ≤ IFace, B ≤ IFace)

Error messages

- We can’t expect the user to understand all the type-inference
- The solution: keep the reason why we deduced a type and expose it to the user



```
File "test.php", line 6, characters 10-11:
Invalid return type
File "test.php", line 3, characters 24-26:
This is an int
File "test.php", line 5, characters 10-11:
It is incompatible with a string
```

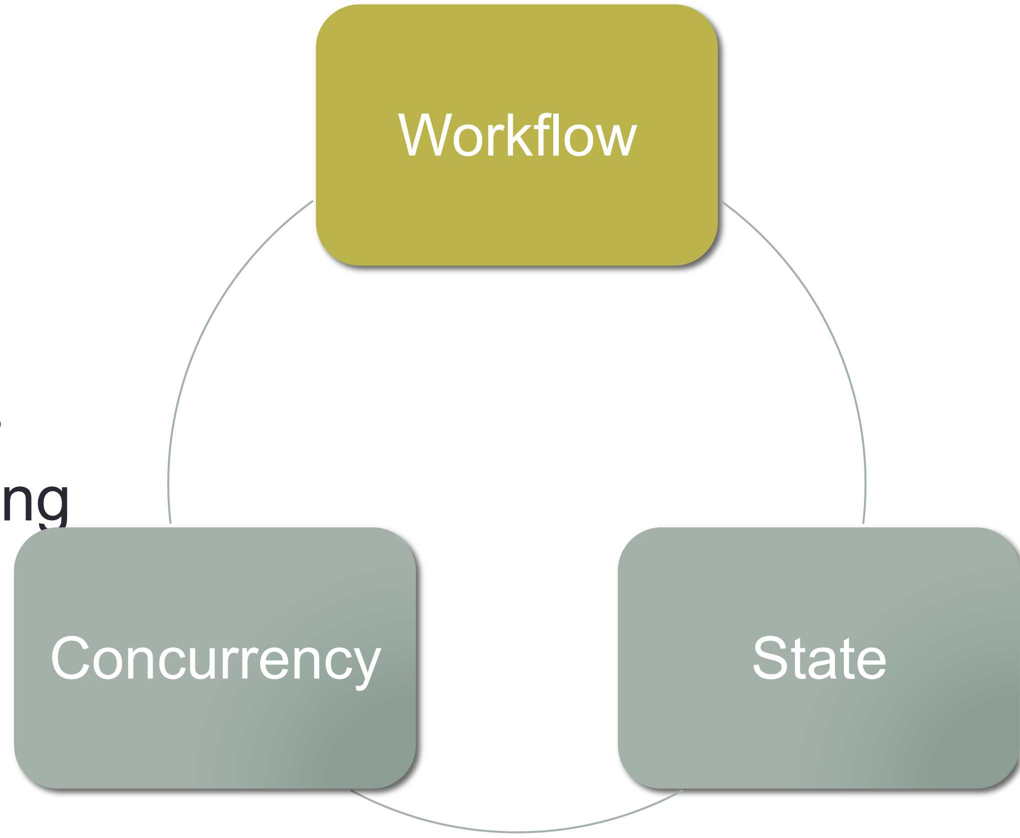
Hack

- “[X] You have reinvented PHP better, but that’s still no justification
- [X] The name of your language makes it impossible to find on Google”

- Many millions of lines converted
- Most new code in Hack
- Most PHP users at Facebook regularly check in Hack

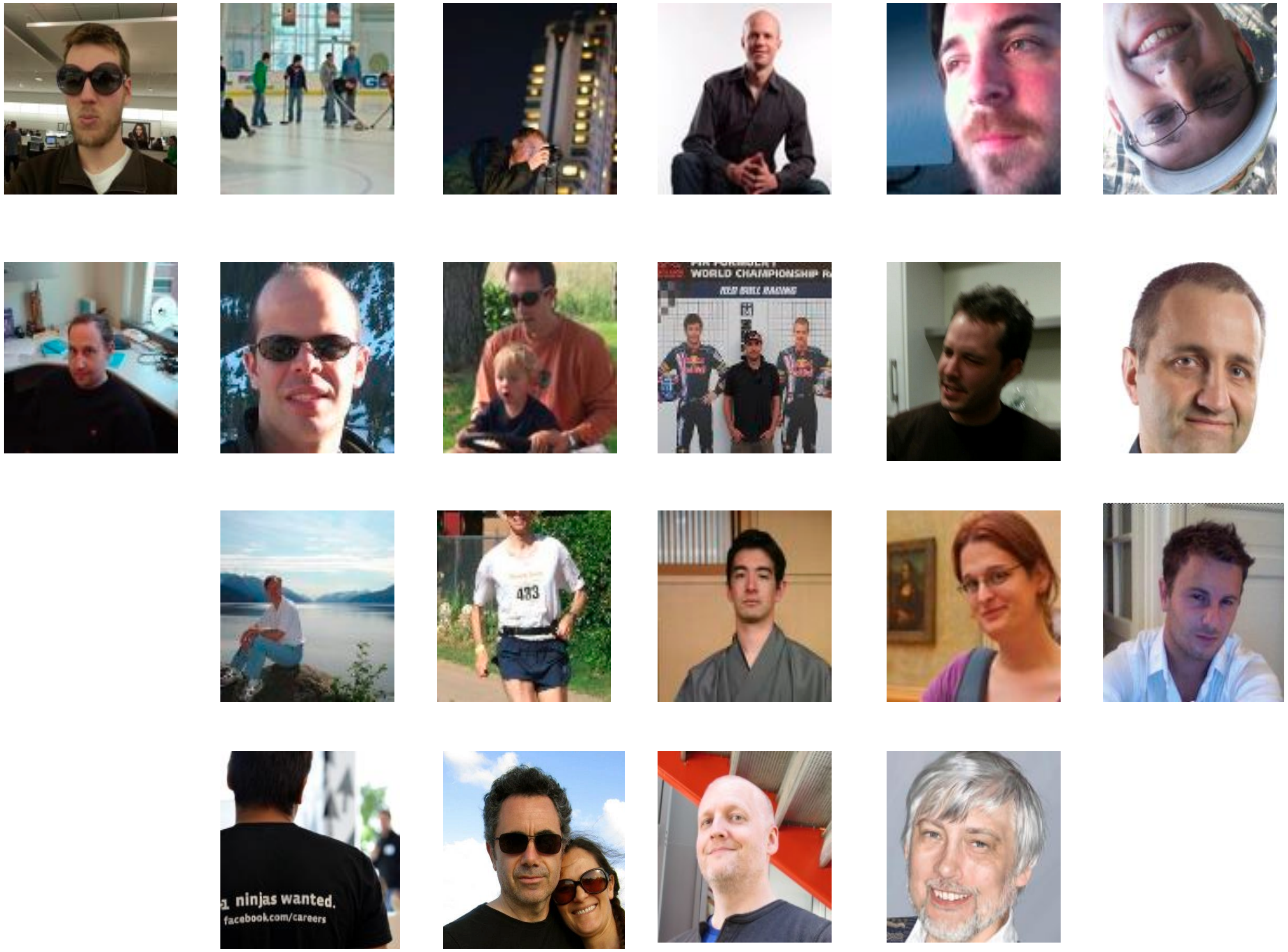
Postmodern PHP (2014-...)

- HipHop project provides great tools
  - Fast VM
  - Debugger
  - Profiler
  - Integrations with editors/IDEs
- Hack is a SoA gradual typing system
- Maintains all of PHP's strengths
- Compare to your favorite "Dynamic Algol"



When PHP?

- Any time you might consider another “Dynamic Algol” language
  - Python, Lua, JavaScript, Perl, Ruby, ...
- Server-side
- Request-oriented
- ...but want to preserve some of the option value of “BigLangs”
  - Type system
  - High-performance implementations


WORLD CHAMPIONSHIP
RED BULL RACING
483
ninjas wanted.
facebook.com/careers

Backup

Everyone's favorite generics slide

- (Remember, "covariance" refers to type specifications for Type that accept T >= Type. "Contravariance" means Type that accept T <= Type.)
- We allow:
  - Covariant function parameters
  - Covariant arrays
  - Constraints on type parameters (Foo<T as IFace> will error if T does not implement IFace)
- We don't allow
  - Contravariant function params (they don't make sense)
  - Covariant type parameters
- Remember, runtime throws everything away anyway, so perfwise, it's type erasure.