

Bringing Functional PHP into the **Fold**

Tom Harding (and hecklers)

@am_i_tom

github.com/i-am-tom/php-folding-talk

CODE WRITTEN IN HASKELL
IS GUARANTEED TO HAVE
NO SIDE EFFECTS.

...BECAUSE NO ONE
WILL EVER RUN IT?





Almost everyone

Units

Immutability

Almost everyone

Generators



Controlled side-effects

Promises

Map/reduce

Single Static Assignment

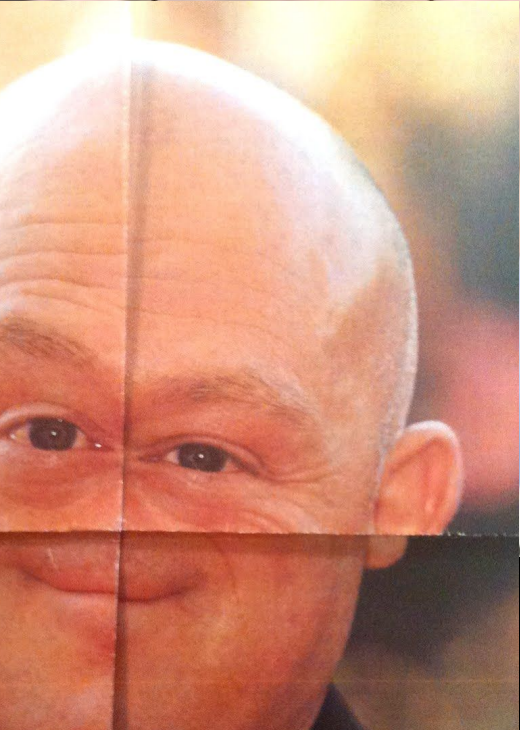


Single Static Assignment

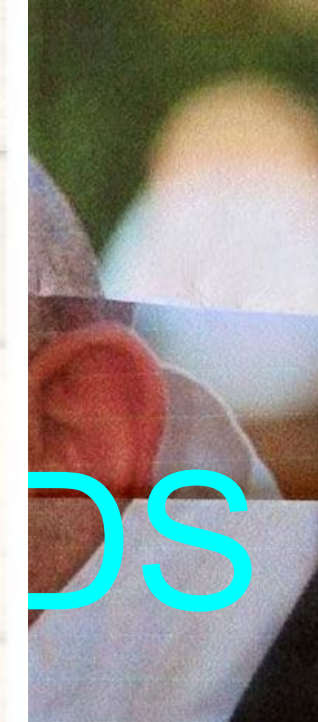
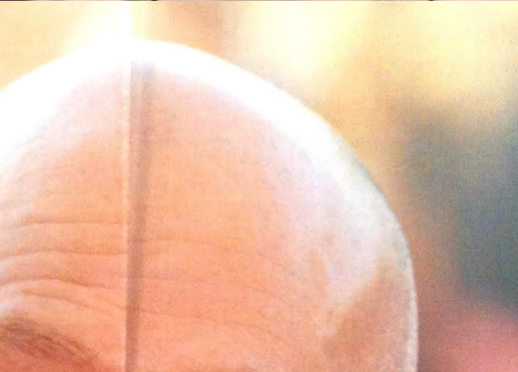
A portrait of a man with a surprised expression, wearing a dark suit, white shirt, and red tie. The image is divided into vertical strips by several fold lines, creating a collage effect. The word "FOLDS" is written in large, bold, cyan capital letters across the center of the image, partially covering the man's face and the fold lines.

FOLDS



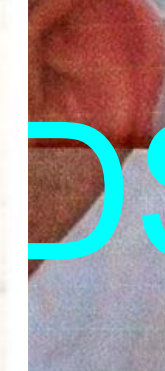
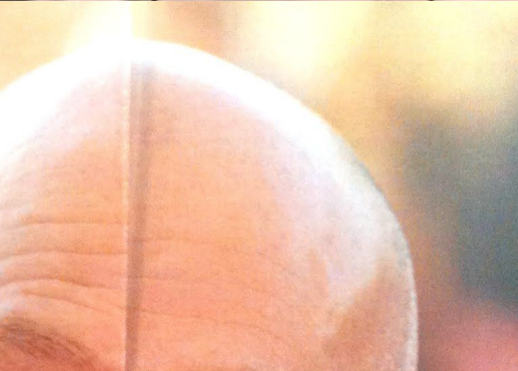


DS



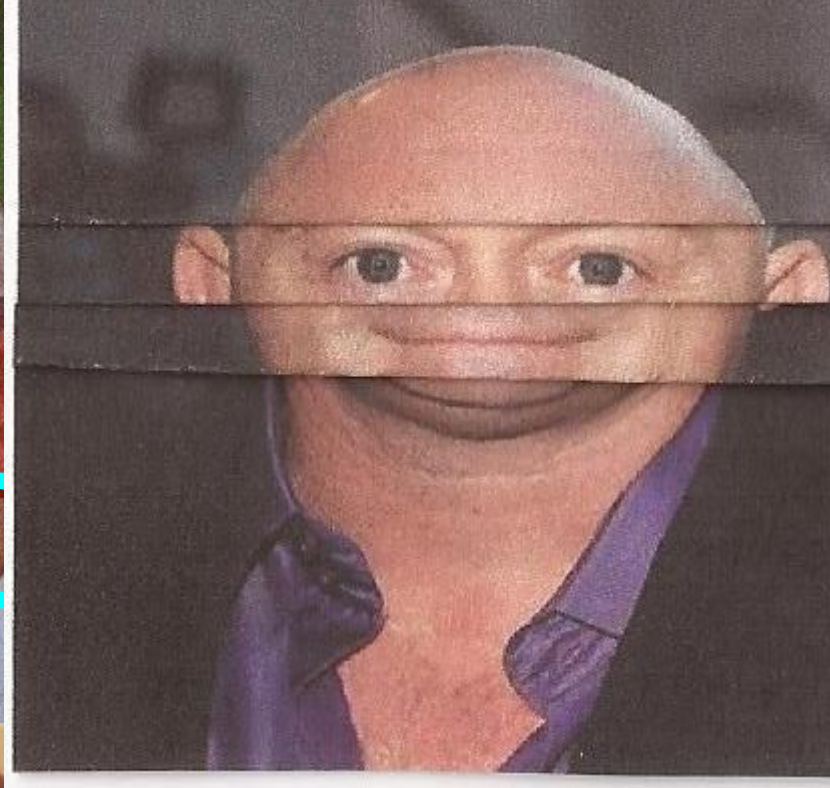
DS





DS









`array_reduce`

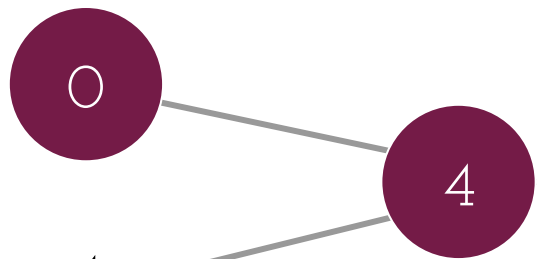


4

9

2

6

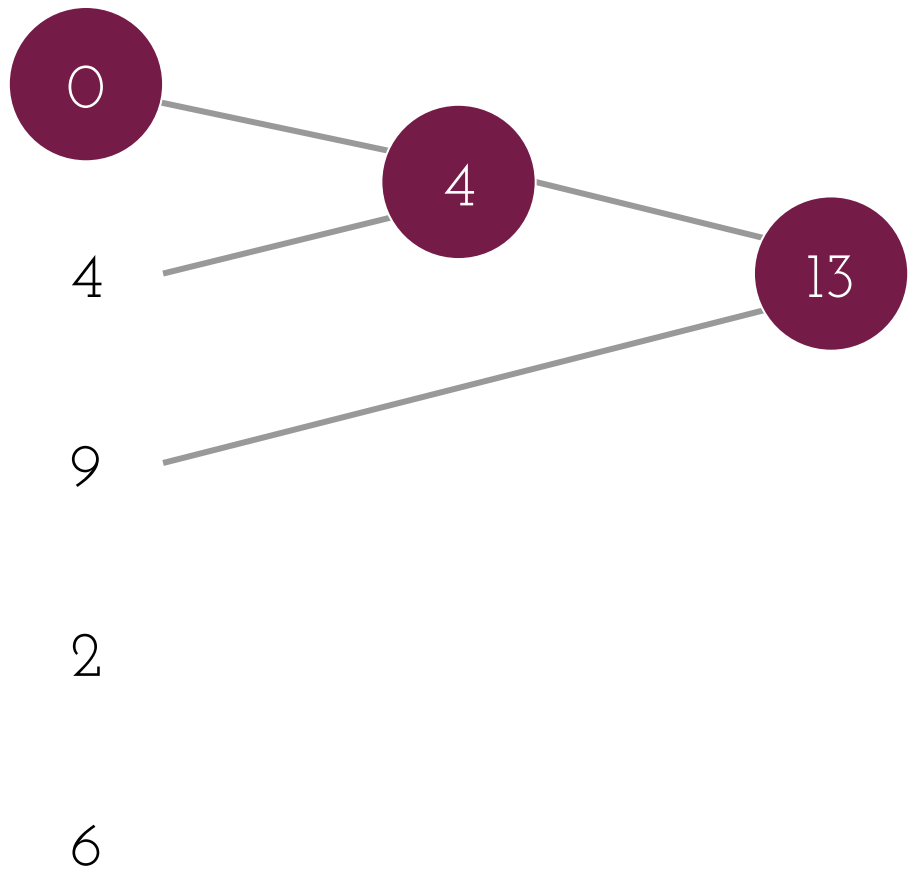


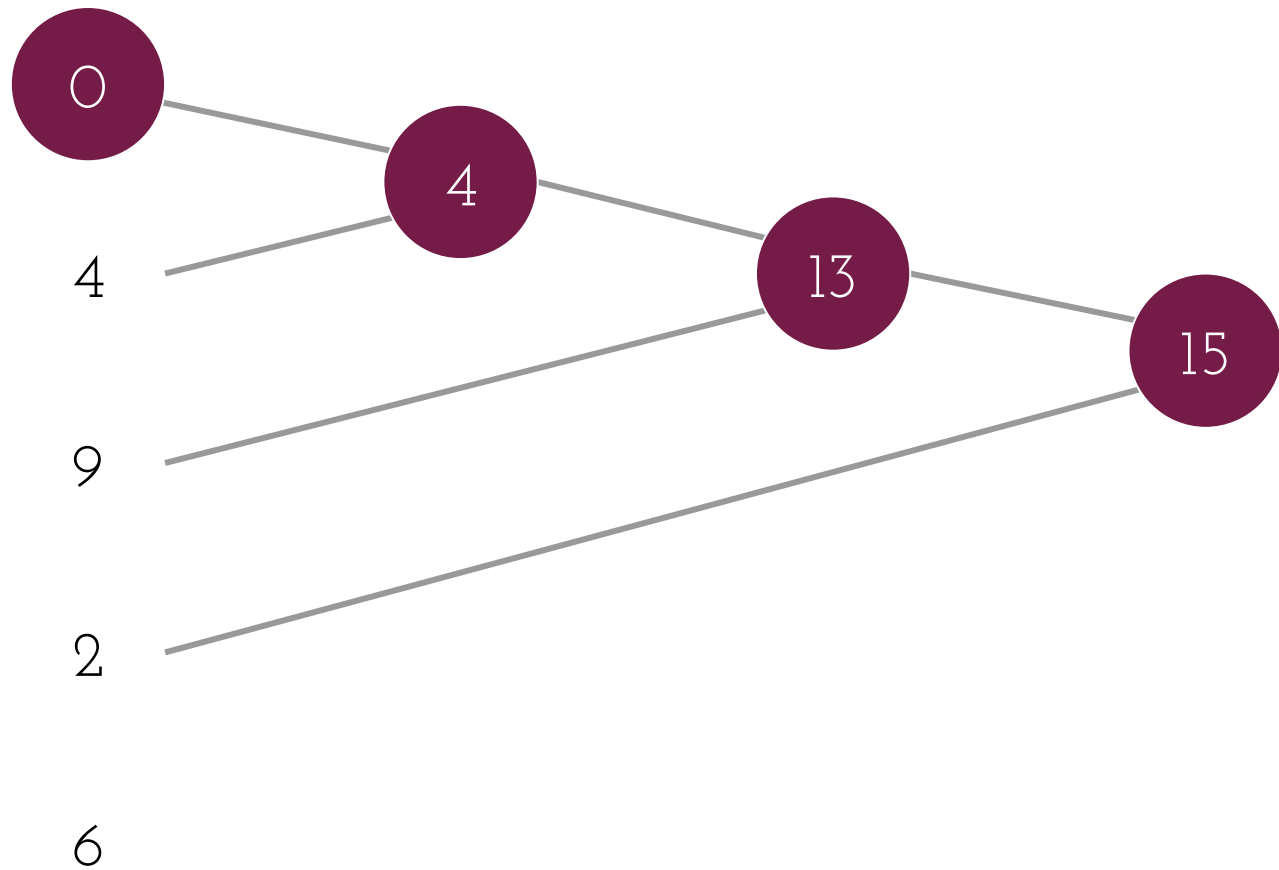
4

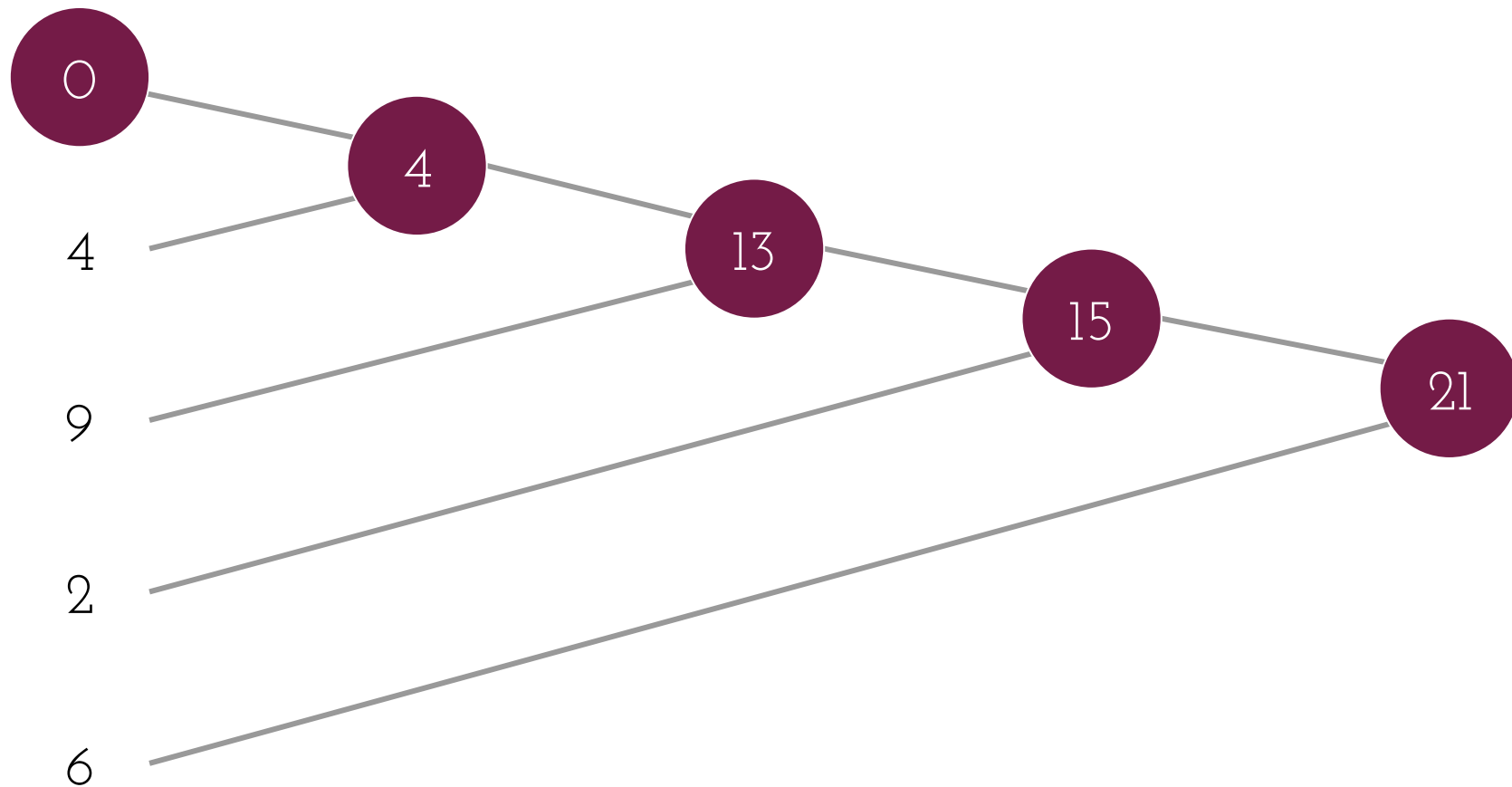
9

2

6









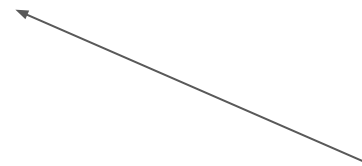
4

9

2

6

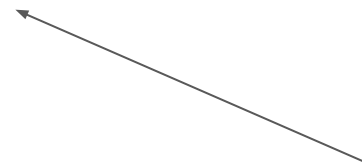
$$A + (B + C) = (A + B) + C$$



Associativity



$$A + (B + C) = (A + B) + C$$



Associativity

4

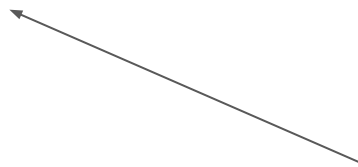
9

2

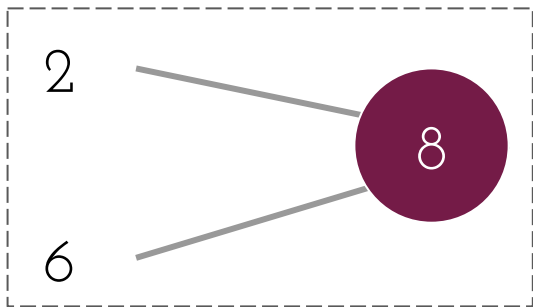
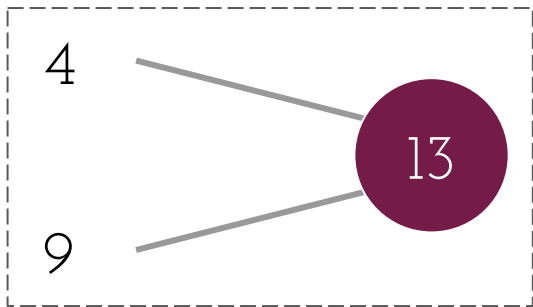
6



$$A + (B + C) = (A + B) + C$$



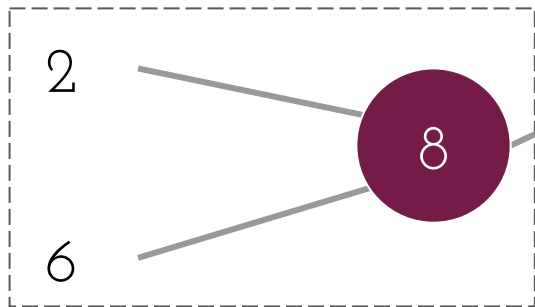
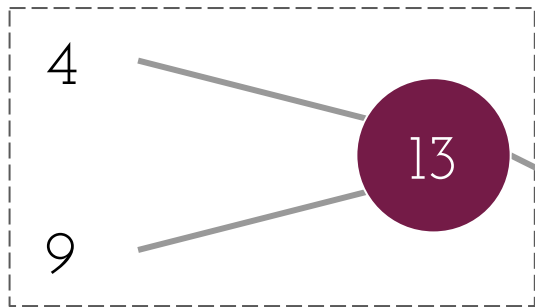
Associativity



0

$$A + (B + C) = (A + B) + C$$

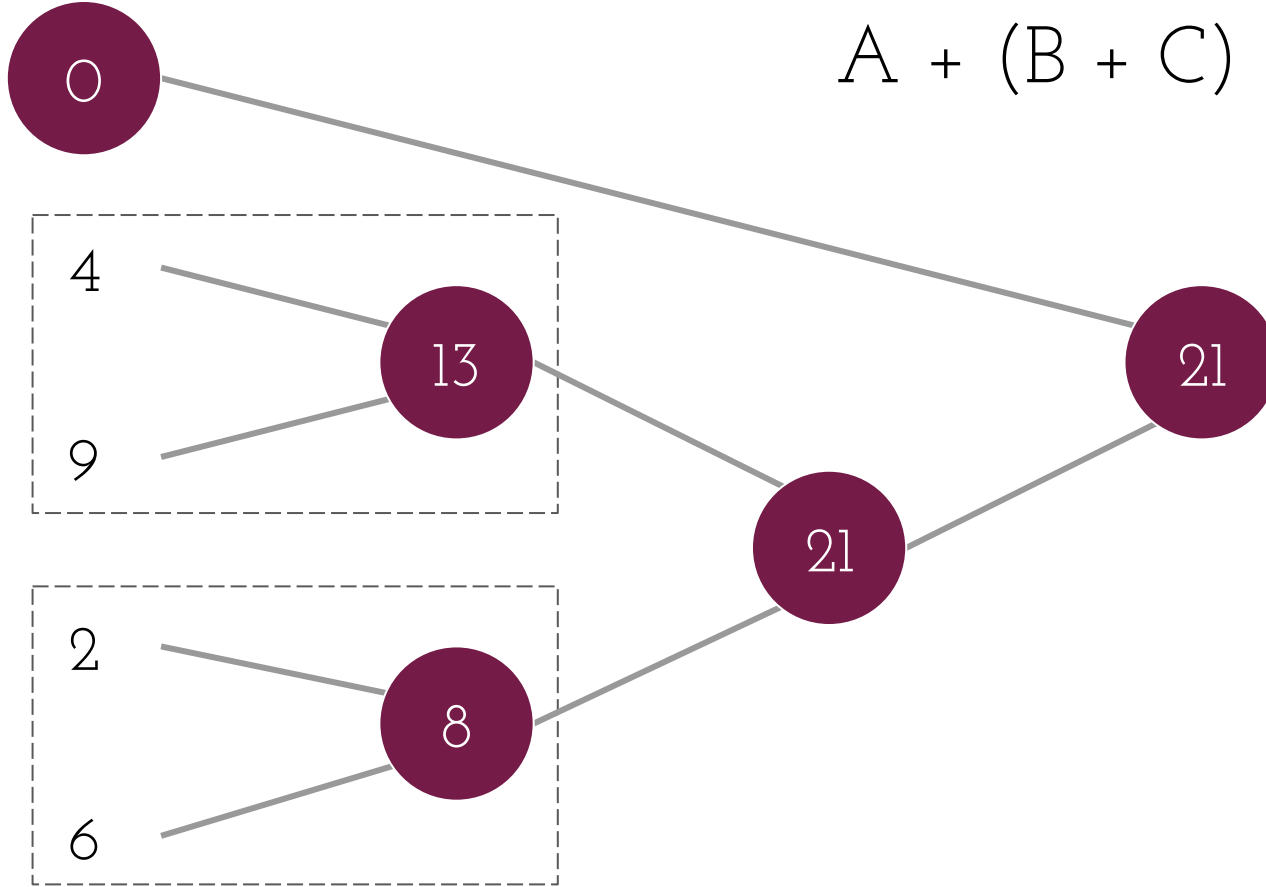
Associativity



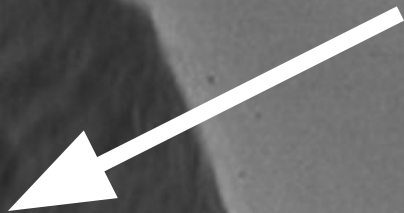
21

$$A + (B + C) = (A + B) + C$$

Associativity



Monoids



Monoid

Semigroup with identity

```
class X implements Monoid
{
    public function append(X $that) : X
    {
        /* ... */
    }

    public static function empty() : X
    {
        /* ... */
    }
}
```

Semigroup: **append** for combining two monoids of the same type.

Identity: a value that can be **appended** to another monoid without changing the other's value.

```
class Sum
{
    public function __construct(float $value)
    {
        $this->value = $value;
    }

    public function append(Sum $that) : Sum
    {
        return new Sum(
            $this->value + $that->value
        );
    }

    public static function identity() : Sum
    {
        return new Sum(0);
    }
}
```

Construct as usual

appending one sum to another
produces the sum of the sums.

The identity is 0:

$$\forall x . x + 0 = x = 0 + x$$


```
class Prod
{
    public function __construct(float $value)
    {
        $this->value = $value;
    }

    public function append(Prod $that) : Prod
    {
        return new Prod(
            $this->value * $that->value
        );
    }

    public static function identity() : Prod
    {
        return new Prod(1);
    }
}
```

Construct as usual

appending one product to
another produces the product
of the products.

The identity is 1:

$$\forall x . x * 1 = x = 1 * x$$

Monoid

Laws in PHP

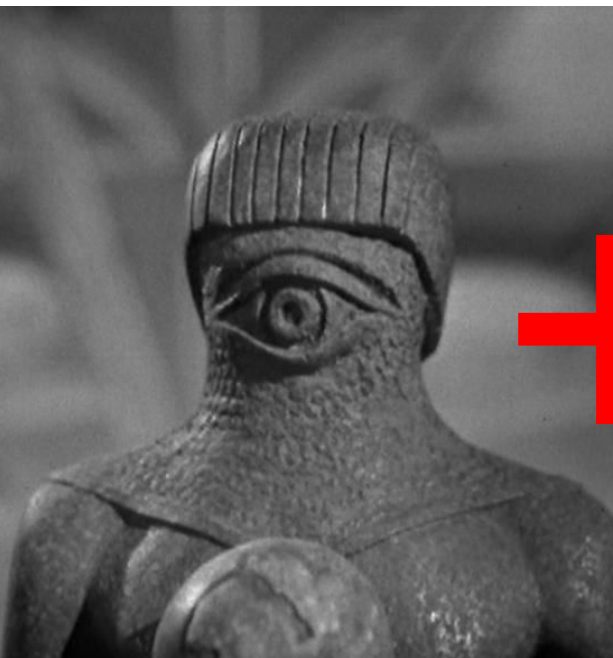
Associativity

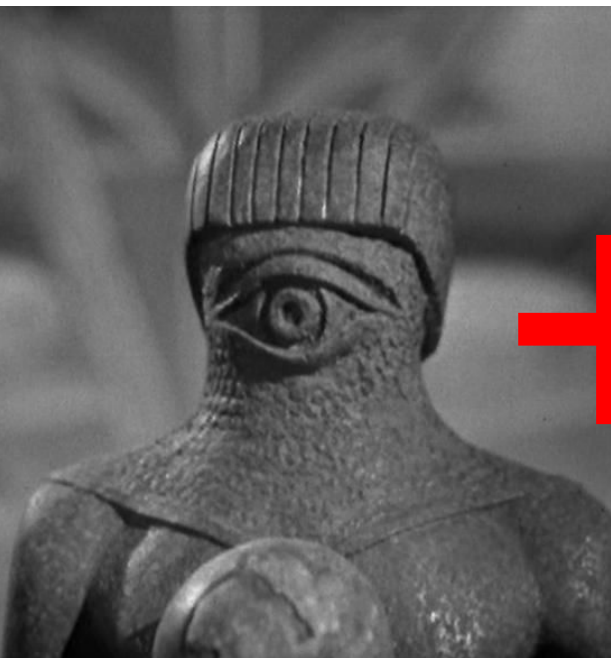


```
$x->append($y)->append($z) == $x->append($y->append($z));
```

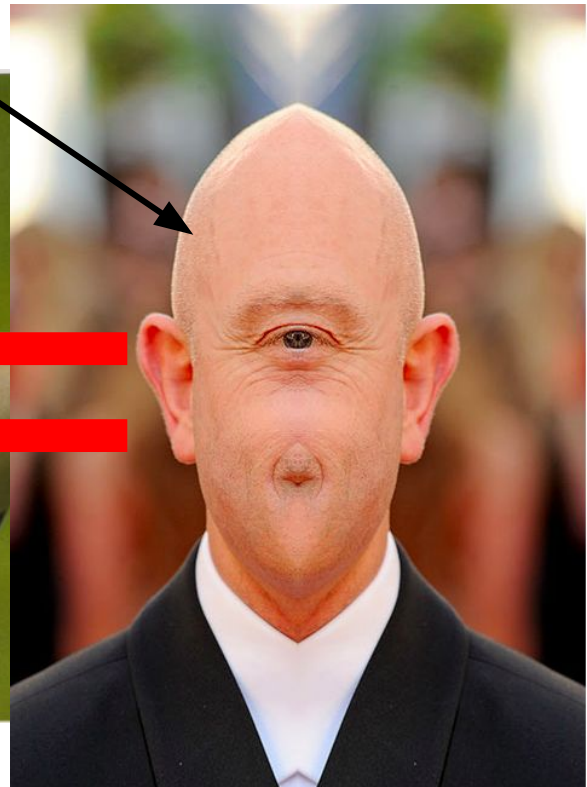
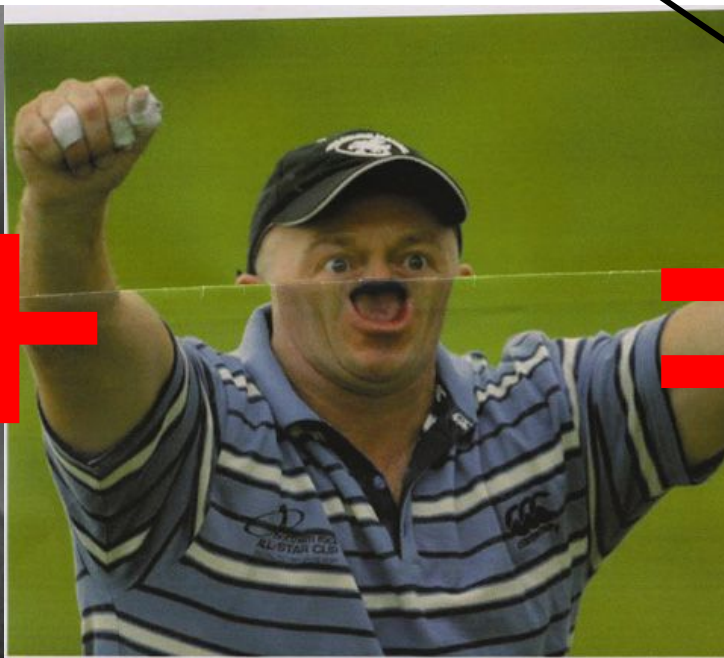
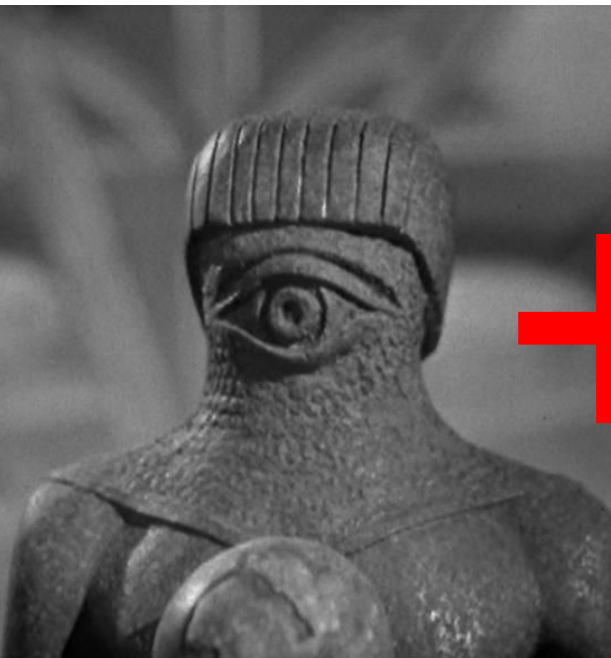
```
M::identity()->append($y) == $y == $y->append(M::identity());
```

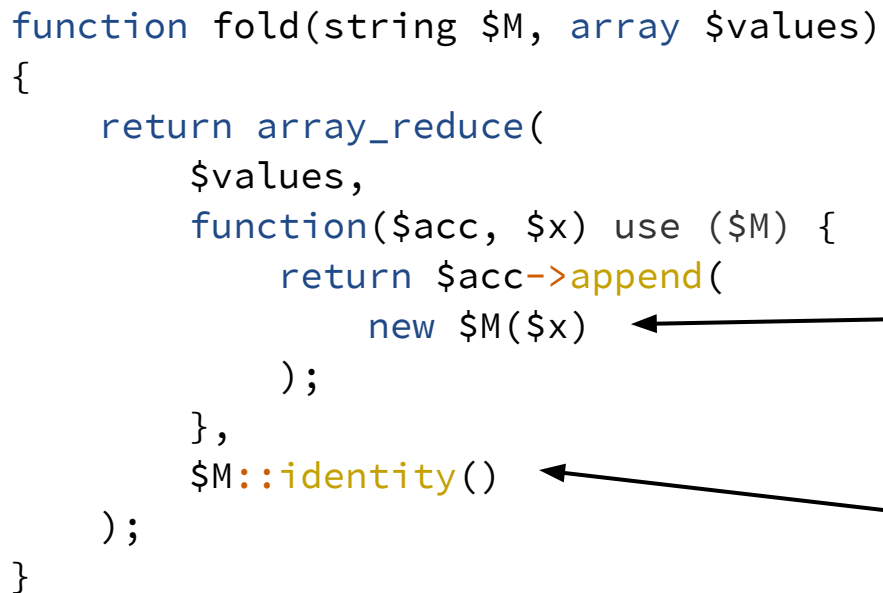






The limit of my
Photoshop abilities





```
function fold(string $M, array $values)
{
    return array_reduce(
        $values,
        function($acc, $x) use ($M) {
            return $acc->append(
                new $M($x)
            );
        },
        $M::identity()
    );
}
```

The diagram illustrates a PHP fold function. A black dot at the top center has two arrows pointing downwards. One arrow points to the text 'PHP's type system :(' on the right. The other arrow points to the function definition. Within the function, two more arrows point from the right to specific parts of the code: one to '\$M(\$x)' in the 'append' call, and another to '\$M::identity()'.

PHP's type system :(

Wrap and **append**

Start with **identity**

Example

```
fold(Sum::class, range(0, 50)) -> value;
```

Example

```
fold(Sum::class, range(0, 50)) -> value;
```

1275

Example

`fold(Sum::class, range(0, 50)) -> value;`

1275

Example

```
fold(Sum::class, range(0, 50))
```

1275

1
2
3
4
5

5
6
7
8
9
10

Example 2

```
$myCollection->fold(Sum::class)->value;
```

Example 2

```
$myCollection->fold(Sum::class)->value;
```



Declarative

Example 2

Open/Closed



```
$myCollection->fold(Sum::class)->value;
```



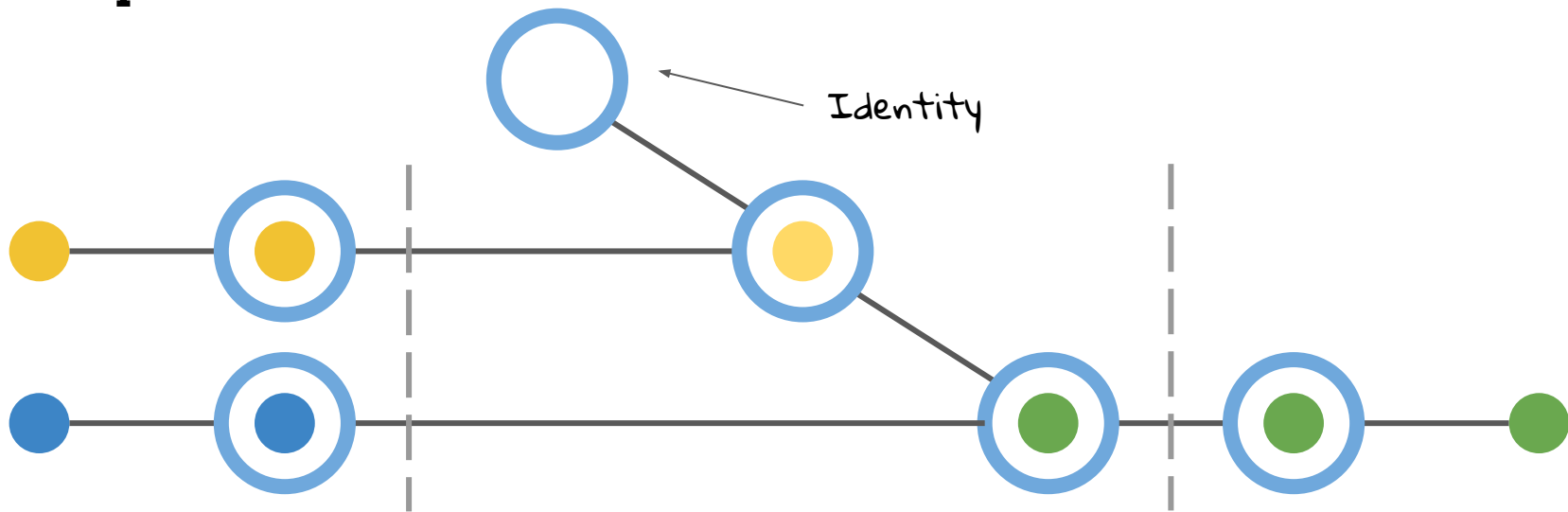
Declarative

Monoid Folds

Wrap the values

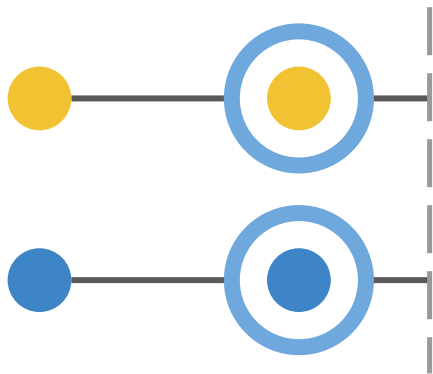
Reduce

Retrieve the value

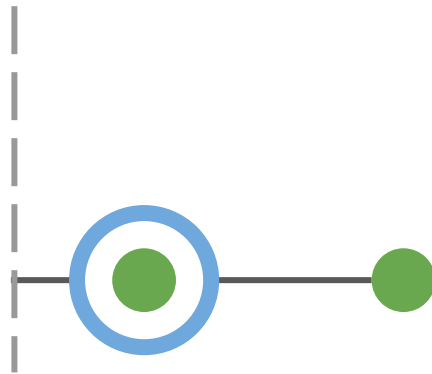


Monoid Folds

Wrap the values



Retrieve the value

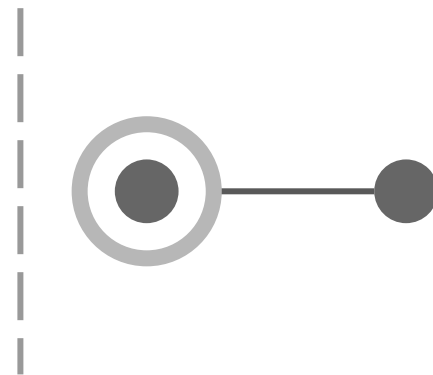
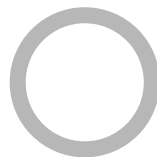
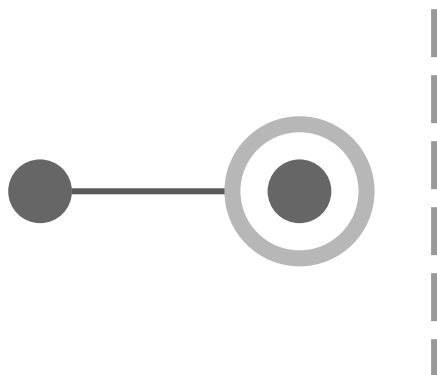


Monoid Folds

Wrap

(monoid)

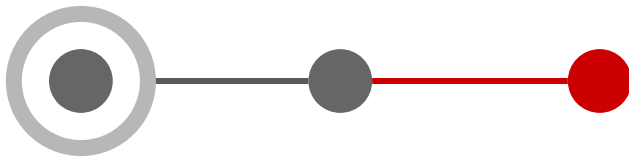
Retrieve



Covariant
Functor



Map



```
$stringJoiner->map('strtoupper');
```

Contramap

Contravariant
Functor



```
$length = $sum->contramap(function ($_) { return 1; });
```

Applicative



Ap

```
$average = $sum->map(function ($x) {  
    return function ($y) use ($x) {  
        return $x / $y;  
    };  
})->ap($length);
```

Applicative



Ap

```
$average = $sum->divideBy($length);
```

Grand Finale

```
$sumSq = $sum->contramap(function ($x) { return pow($x, 2); });
```

```
$standardDev = $sumSq  
    ->divideBy($length)  
    ->minus($average->toThePowerOf(2))  
    ->sqrt();
```

```
$standardDev->fold(range(0, 10)); // 3.162...
```


Questions?

`rosskempfolds.tumblr.com`

`@am_i_tom`

`github.com/i-am-tom/php-folding-talk`