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

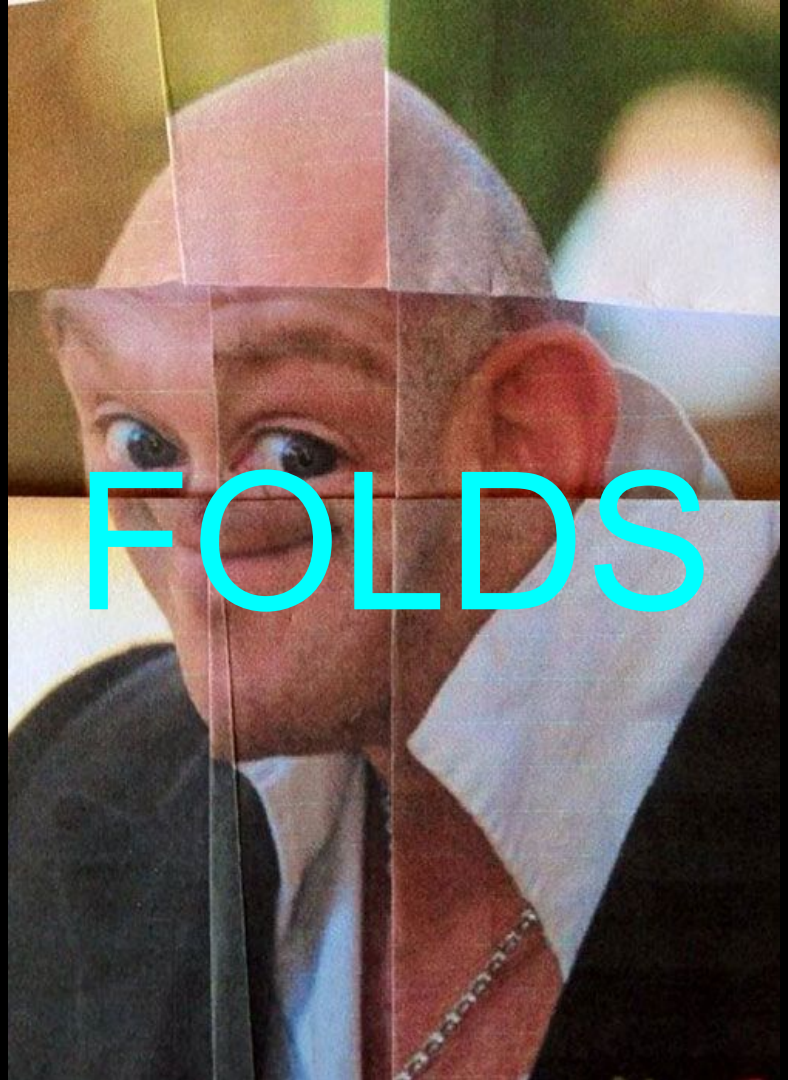
Controlled side-effects

Promises

Map/reduce

Single Static Assignment

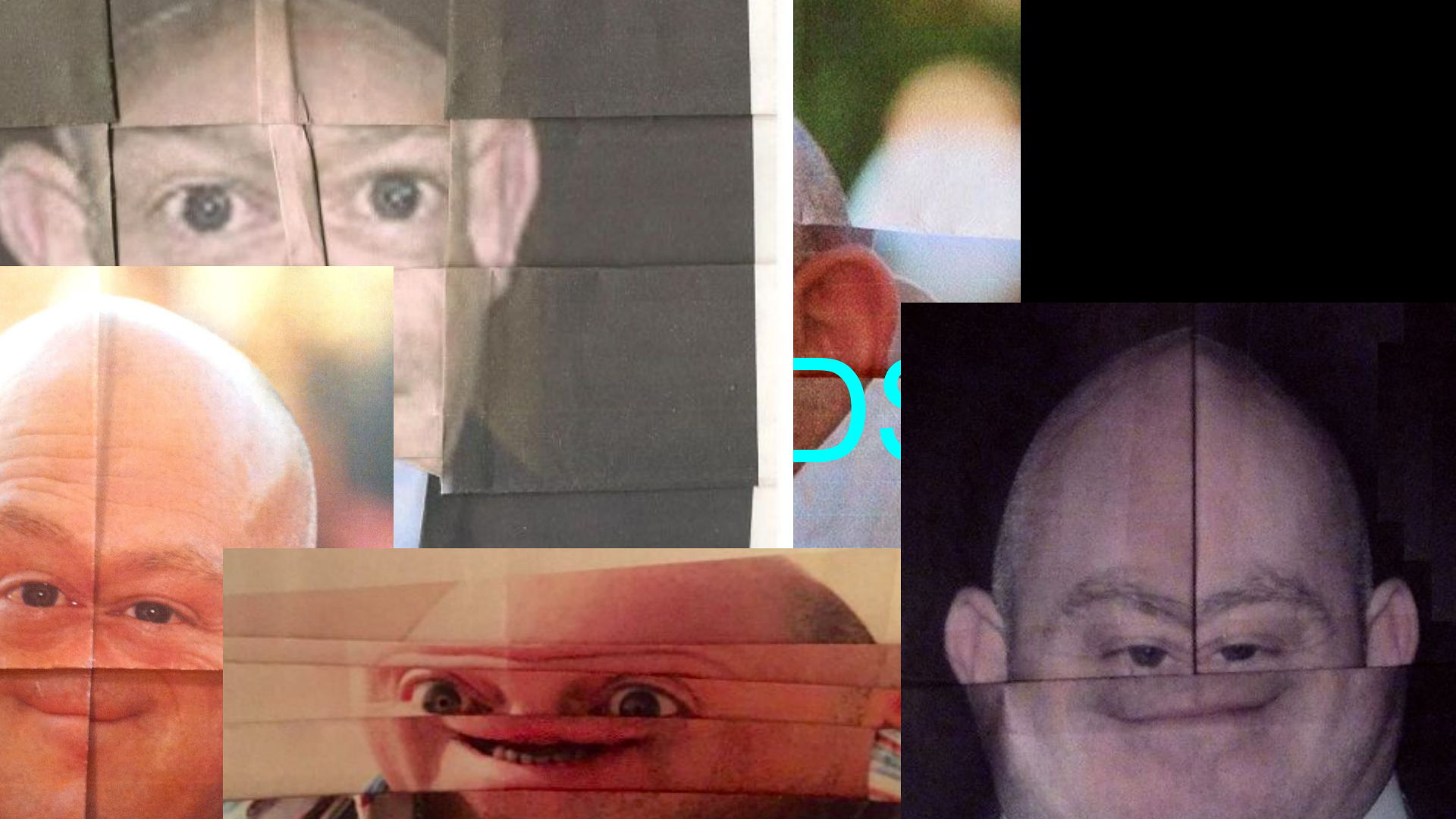
Single Static Assignment


FOLDS


DS


DS


DS




array_reduce

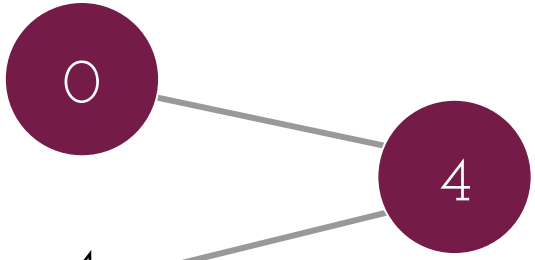
0

4

9

2

6

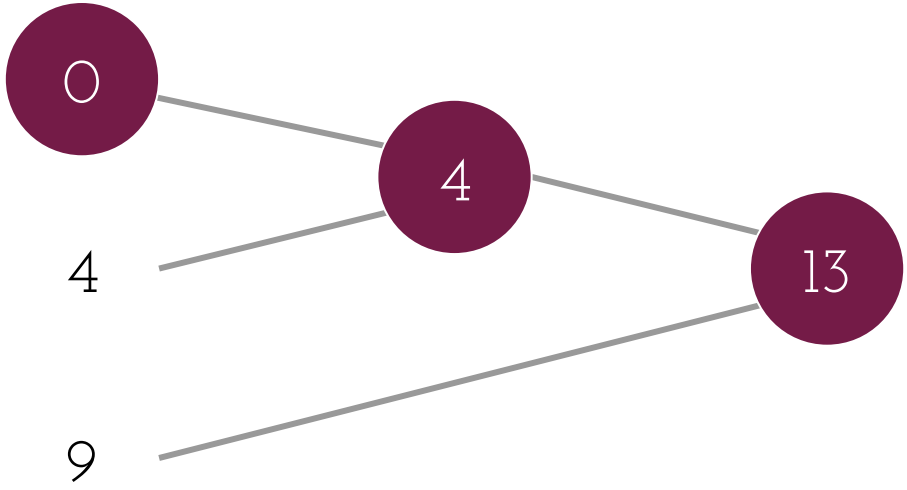


4

9

2

6

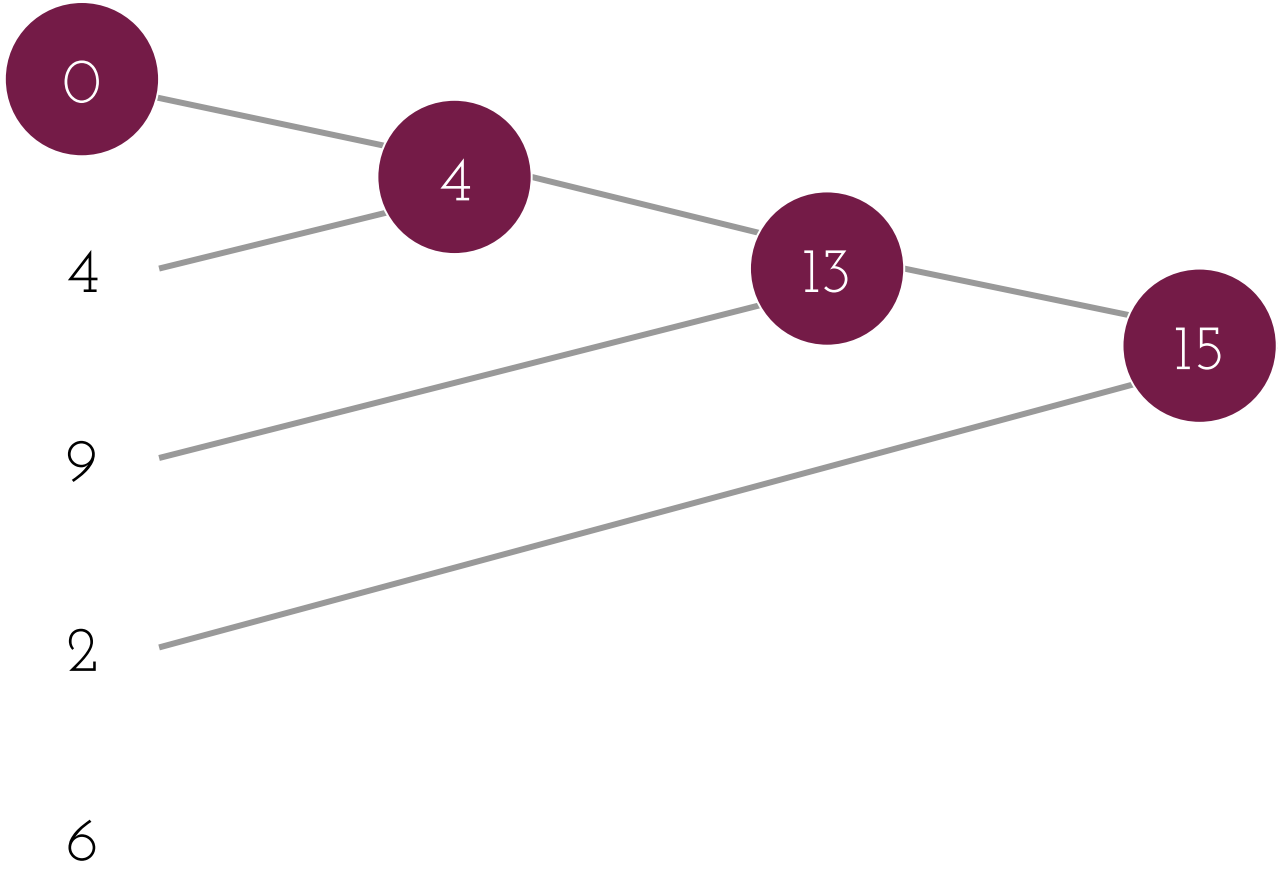


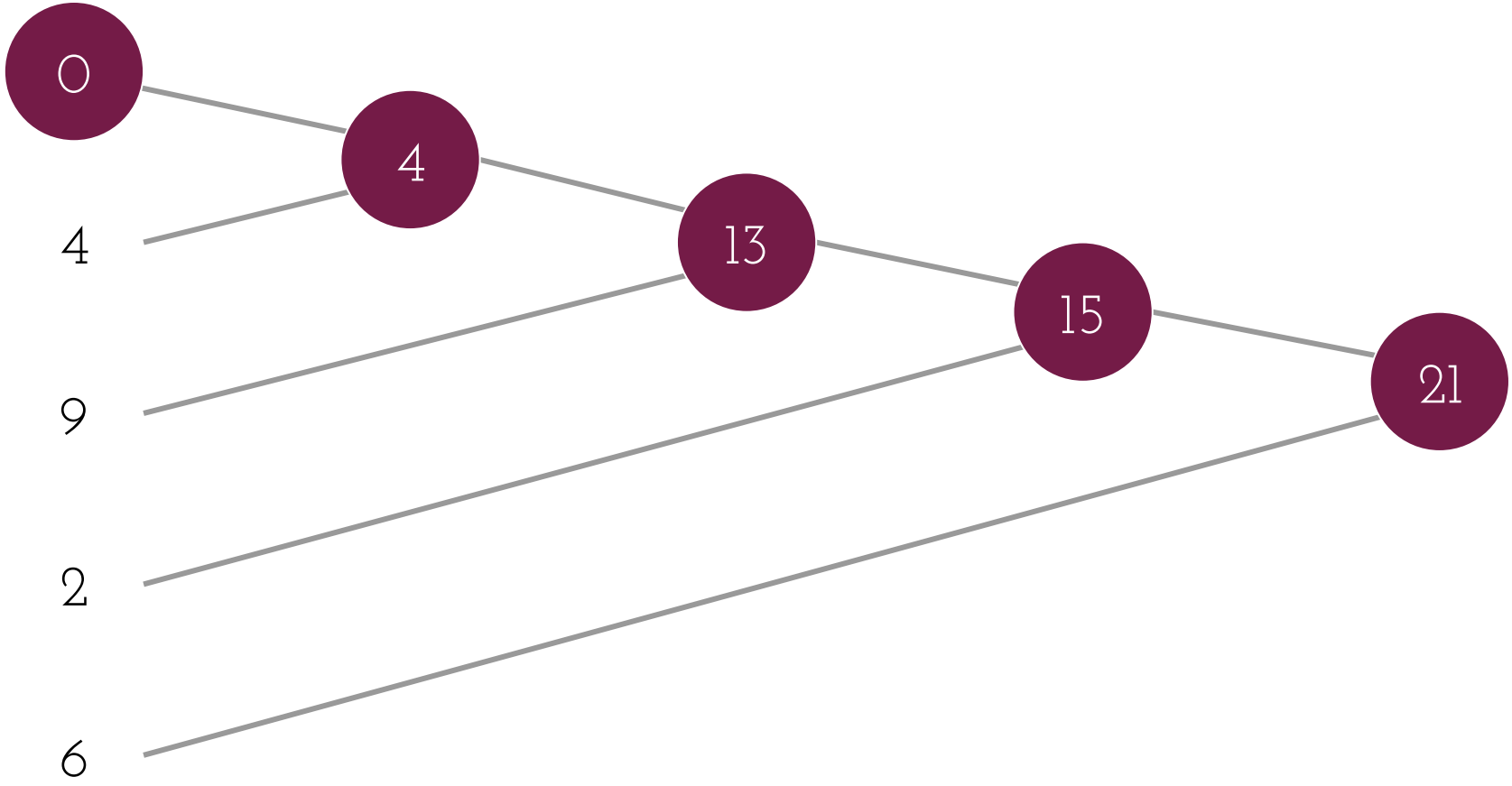
4

9

2

6


0
4
13
15
4
9
2
6


0
4
13
15
21
4
9
2
6

0

4

9

2

6

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

Associativity

0

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

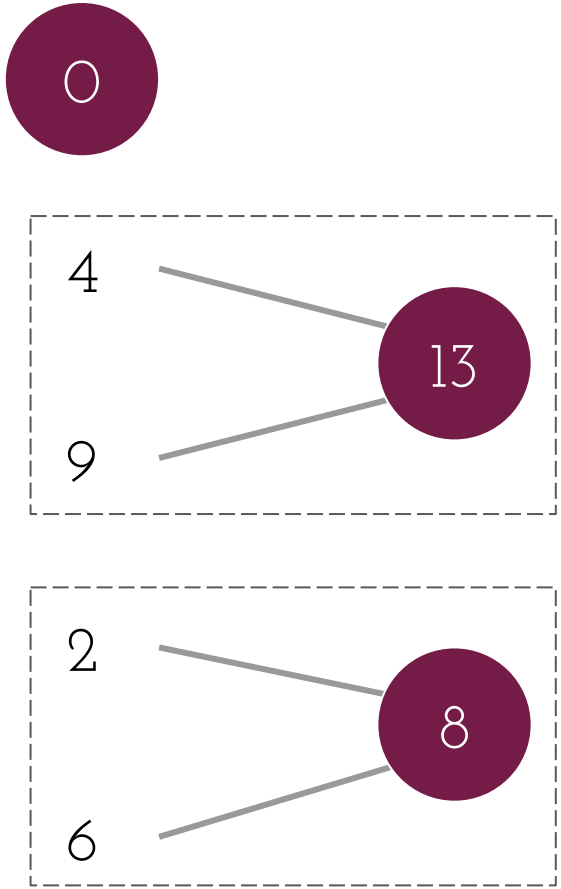
Associativity

4

9

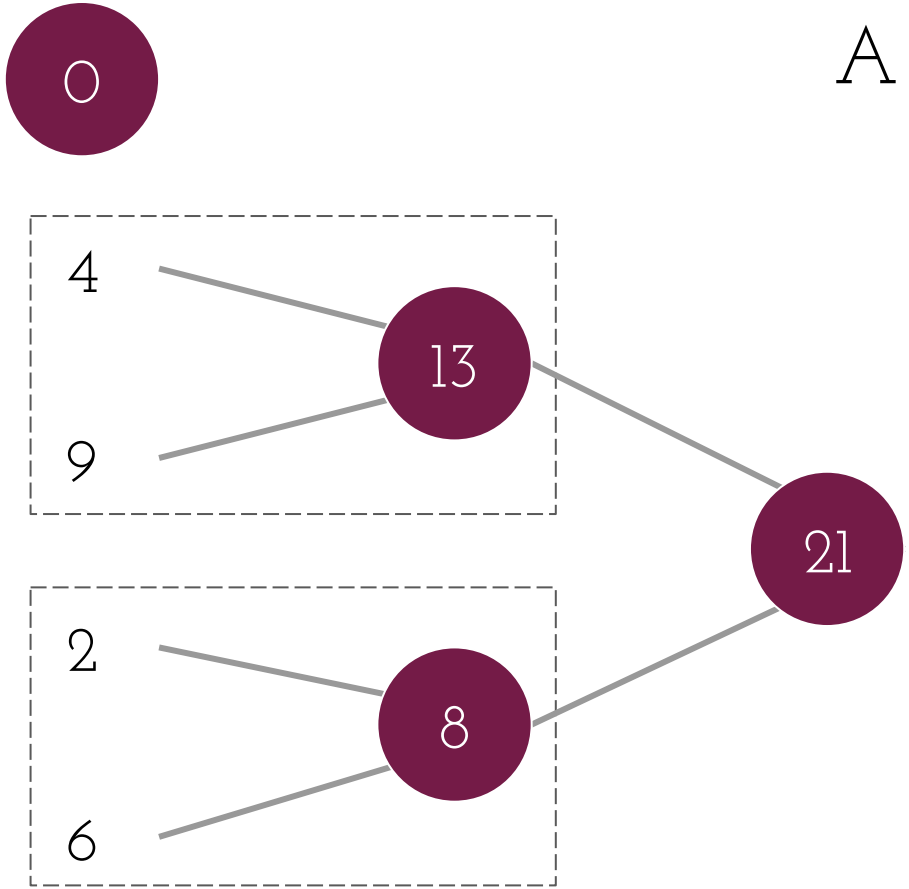
2

6



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

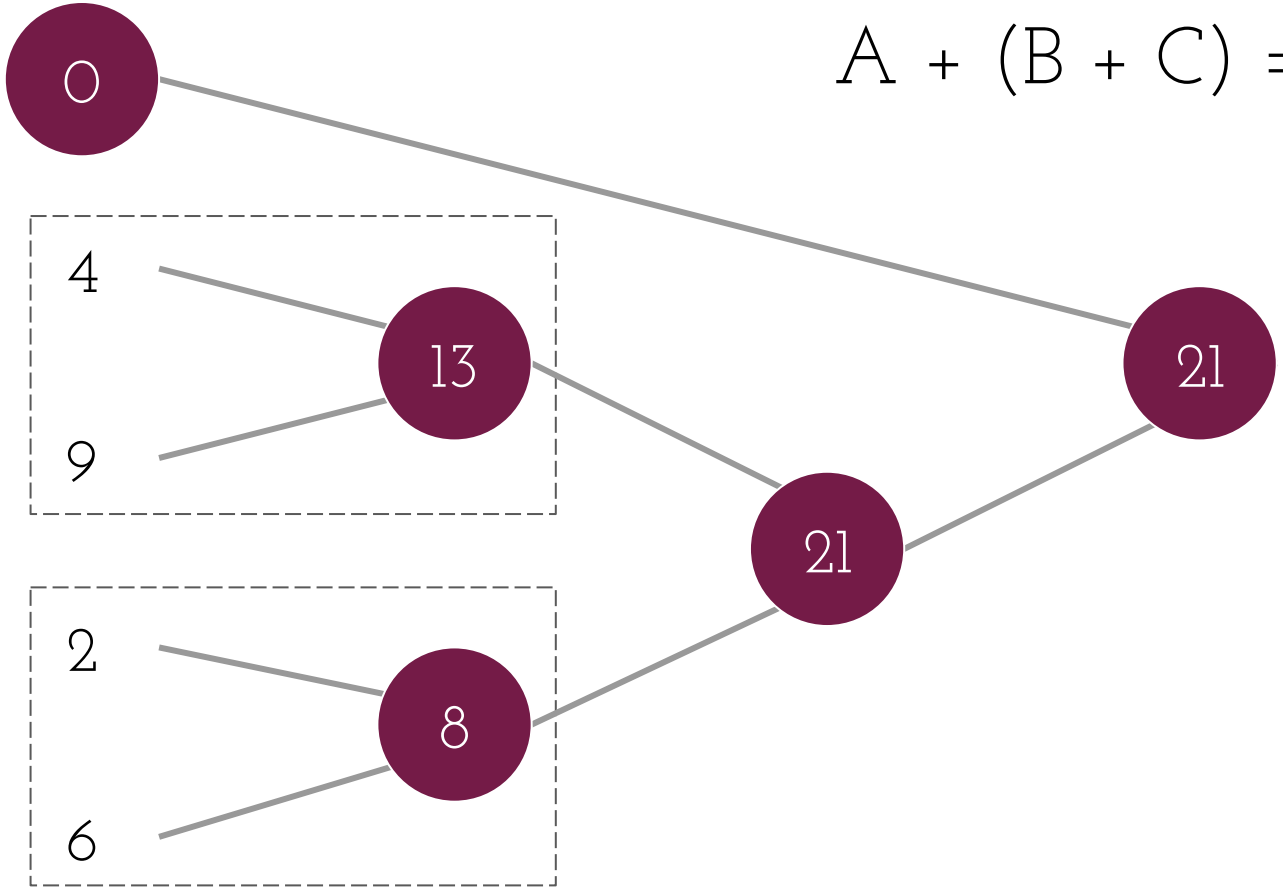
Associativity



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

Associativity

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



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.

```php
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:

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

```php
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**:

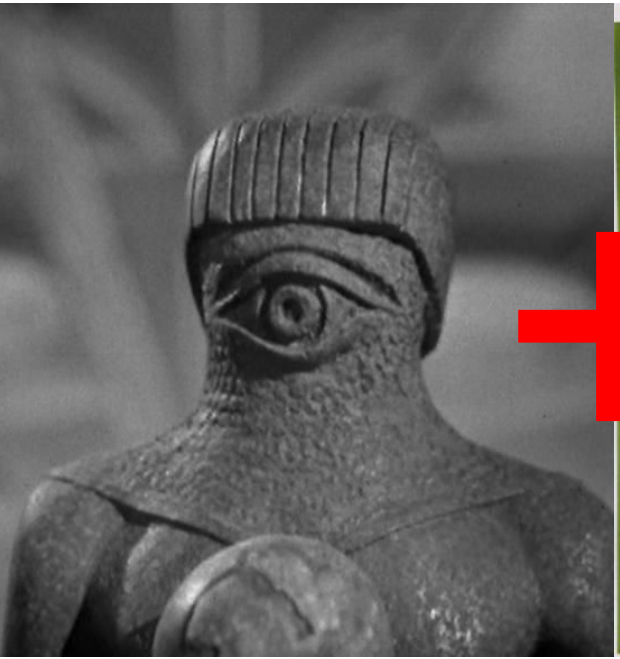
$\forall x \; . \; x * 1 = x = 1 * x$

Monoid

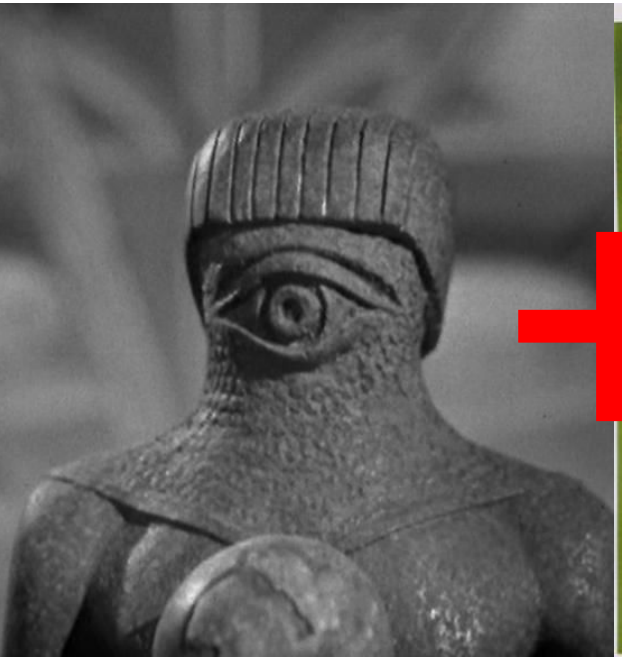
Laws in PHP

Associativity

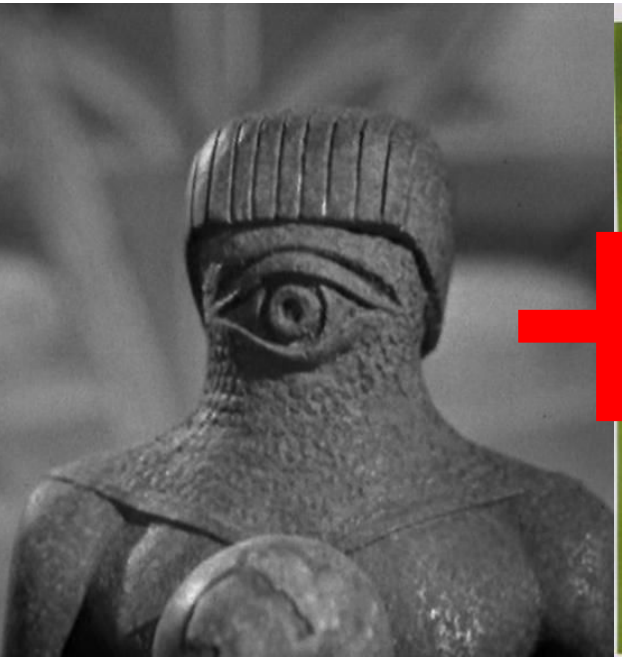
```
$x->append($y)->append($z) == $x->append($y->append($z));

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



+

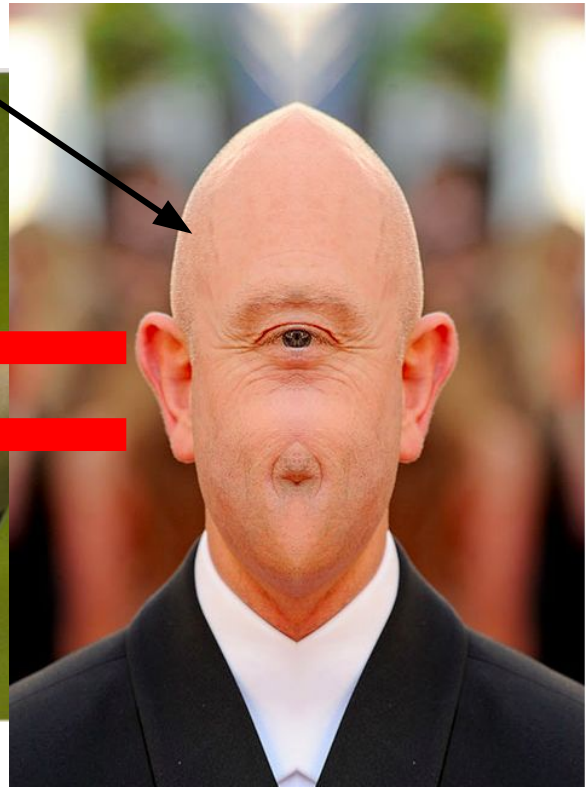


The limit of my
Photoshop abilities



+

=



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

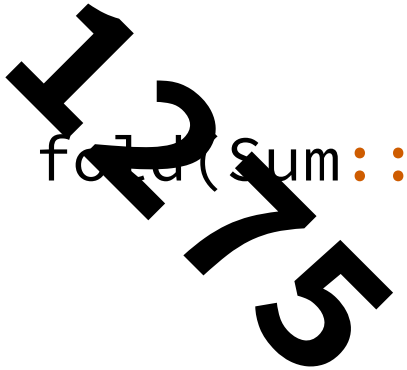
Example

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

1275

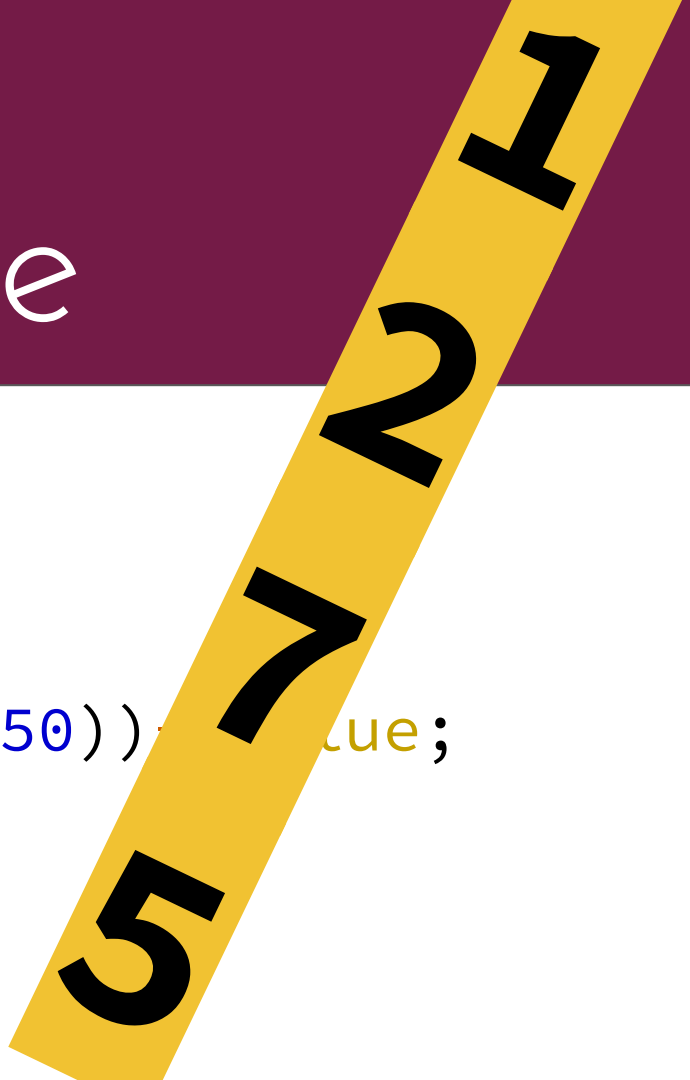
1275

Example



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

1275



Example 2

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

Example 2

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

Declarative

Example 2

Open/Closed

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

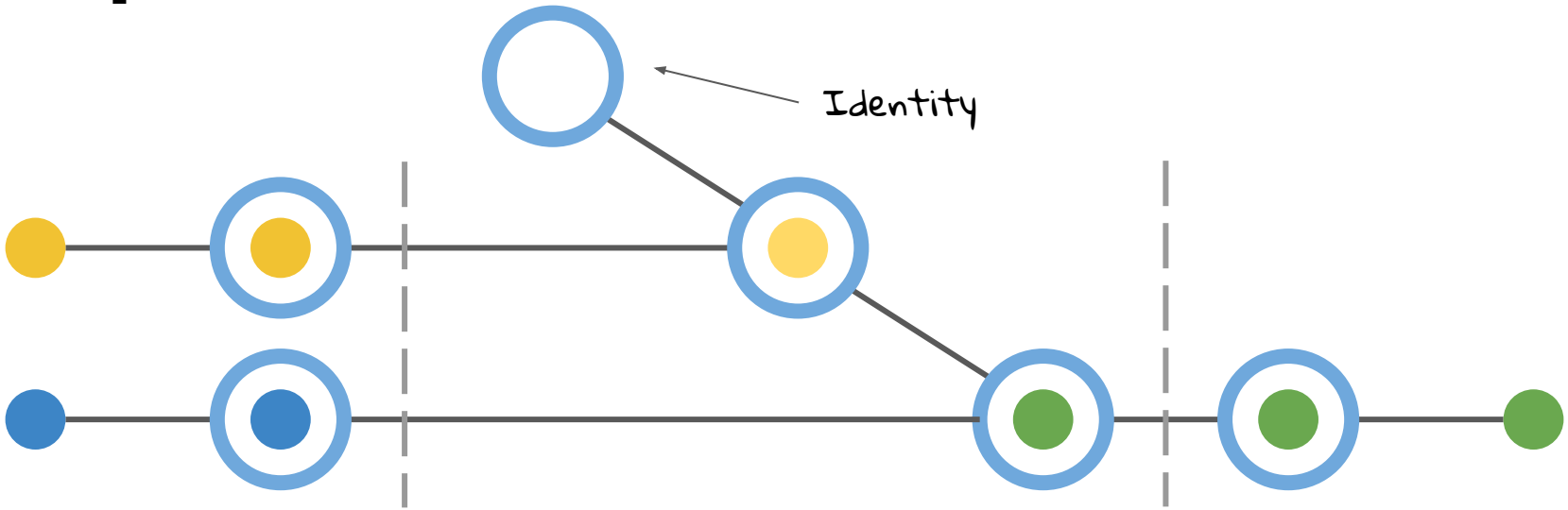
Declarative

Monoid Folds

Wrap the values

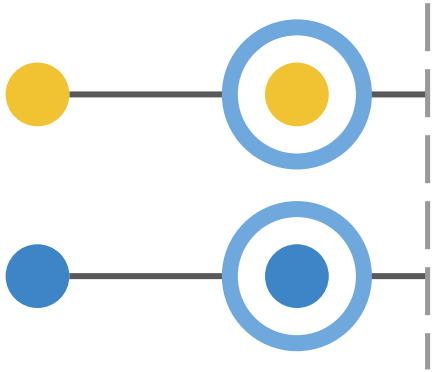
Reduce

Retrieve the value

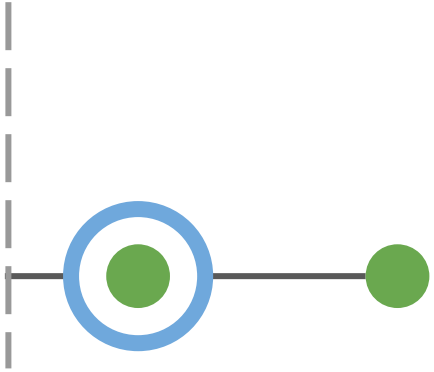


Monoid Folds

Wrap the values

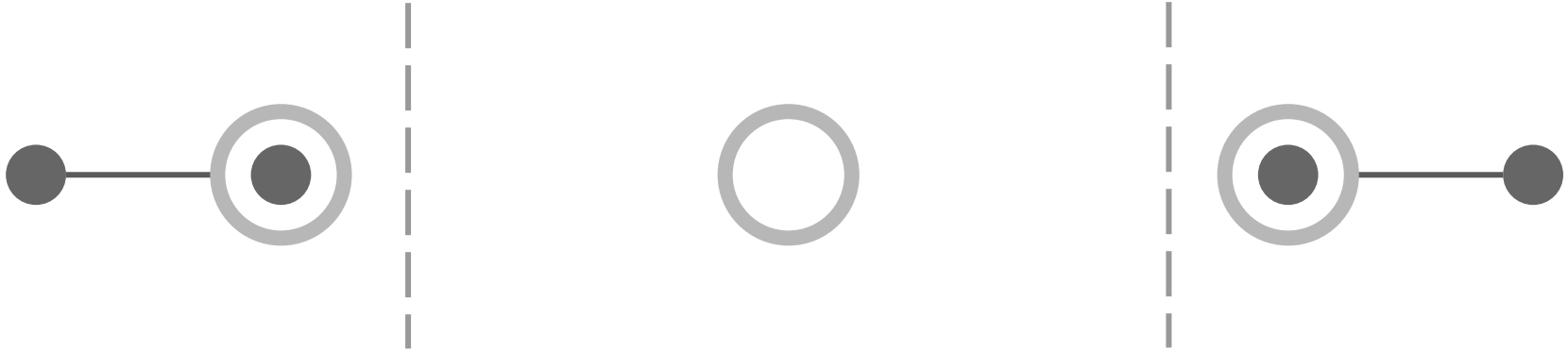


Retrieve the value



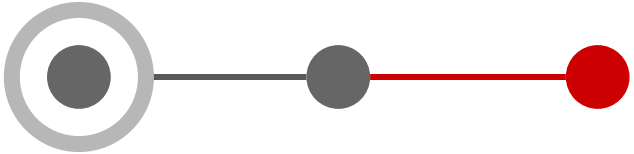
Monoid Folds

Wrap **(monoid)** **Retrieve**



Covariant Functor

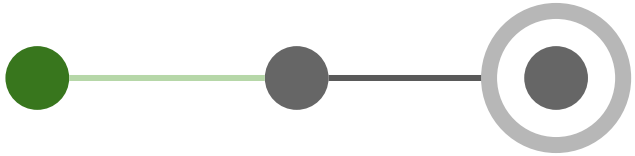
Map



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

Contravariant Functor

Contramap



```
$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