Async PHP with AMP

https://amphp.org/amp/

But why?

- Performance (app is already spun up, single process)
- Server type tasks
- Streaming
- Basically anything that benefits long running management and knowledge of of state in a master process (PHP doesn’t normally do this)
- Really cool tests

It’s hard

- We think sync
- Time and order of operations (sequence) is complex
- PHP is short lived, no loop

It can be done

But how?

- Event Loop
- Promises (React or AMP)
- Async Iterators
- Coroutines! (ticket right out of callback hell)
- Process management

Event Loop

- Something's gotta block something (the unmoved mover); holds process up
- Manages events, scheduling
- General control paradigm around event driven programming
- Entry point for running your app

'The fair treatment of clients is thus the responsibility of your application' - node.js

Event Loop

```php
<?php

use Amp\Loop;

$myText = null;

function onInput($watcherId, $stream)
{
    global $myText;

    $myText = fgets($stream);
    stream_set_blocking(STDIN, true);

    Loop::cancel($watcherId);
    Loop::stop();
}

Loop::run(function () {
    echo "Please input some text: ";
    stream_set_blocking(STDIN, false);

    // Watch STDIN for input
    Loop::onReadable(STDIN, "onInput");

    // Impose a 5-second timeout if nothing is input
    Loop::delay($msDelay = 5000, "Amp\\Loop::stop");
});

var_dump($myText); // whatever you input on the CLI

// Continue doing regular synchronous things here.
```

Promises

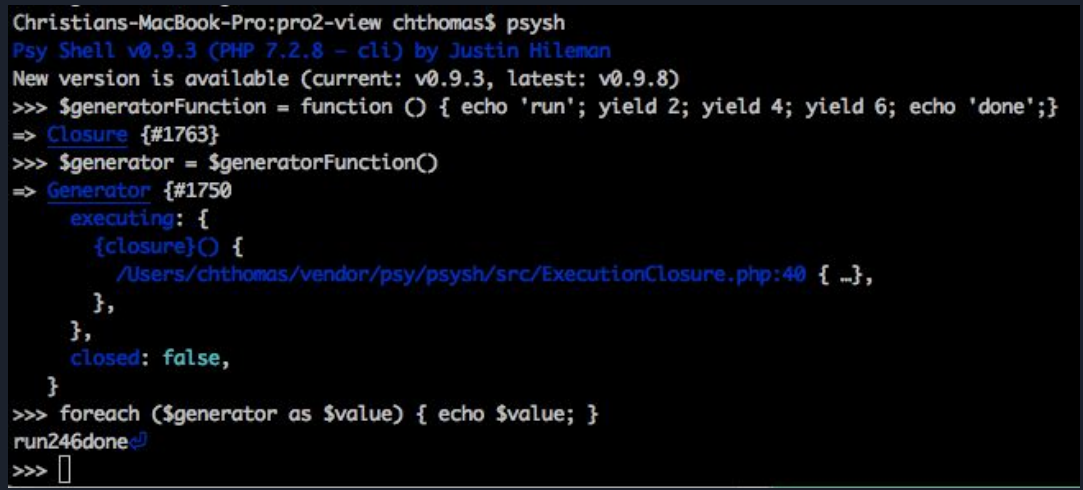
- Front developers hate this one simple trick
- A spec for contracting out work to do eventually (eventual return values)
- Can substitute ReactPHP promises in AMP
- Yielded to get values from

Promises

```php
<?php // Example async producer using promisor

use Amp\Loop;

function asyncMultiply($x, $y)
{
    // Create a new promisor
    $deferred = new Amp\Deferred;

    // Resolve the async result one second from now
    Loop::delay($msDelay = 1000, function () use ($deferred, $x, $y) {
        $deferred->resolve($x * $y);
    });

    return $deferred->promise();
}

$promise = asyncMultiply(6, 7);
$result = Amp\Promise\wait($promise);

var_dump($result); // int(42)
```

Generators: a quick word

- Between callable and generators, the backbone of work orchestration
- They are so cool!
- send() throw()
- yield, yield from (generator delegation)
- Great with memory usage
- JIT



Async Iterators

- Iterators that return promises!

```
namespace Amp;

interface Iterator
{
    public function advance(): Promise;
    public function getCurrent();
}
```

Coroutines

- Interruptible functions whenever there is a ‘yield’

```php
// Fetches a resource with Artax and returns its body.
$promise = Amp\call(function () use ($http) {
    try {
        // Yield control until the generator resolves
        // and return its eventual result.
        $response = yield $http->request("https://example.com/");

        $body = yield $response->getBody();

        return $body;
    } catch (HttpException $e) {
        // If promise resolution fails the exception is
        // thrown back to us and we handle it as needed.
    }
});
```

OK! Let’s build something async

OK! Let’s build something in…parallel

Questions?

Thanks?