Mod_Perl

And why I don’t care about your scorn.

By Julian Brown
Developer @ cPanel
Thursday Sept 14th, 2017

When I say mod_perl, think mod_perl2

It’s about Trade Offs

- **I use and prefer to use Mod_Perl in a CGI style setup.**
- **I believe it is the least overhead per request.**
- **But the trade off? My full Perl app is built into httpd.**
- **Hence I am trading memory footprint for perl script startup time.**
- **So let’s take a brief look at mod_perl’s other CGI style cousins.**

mod_perl's cousins are:

- **CGI**
- **FastCGI**
- **PHP-FPM**

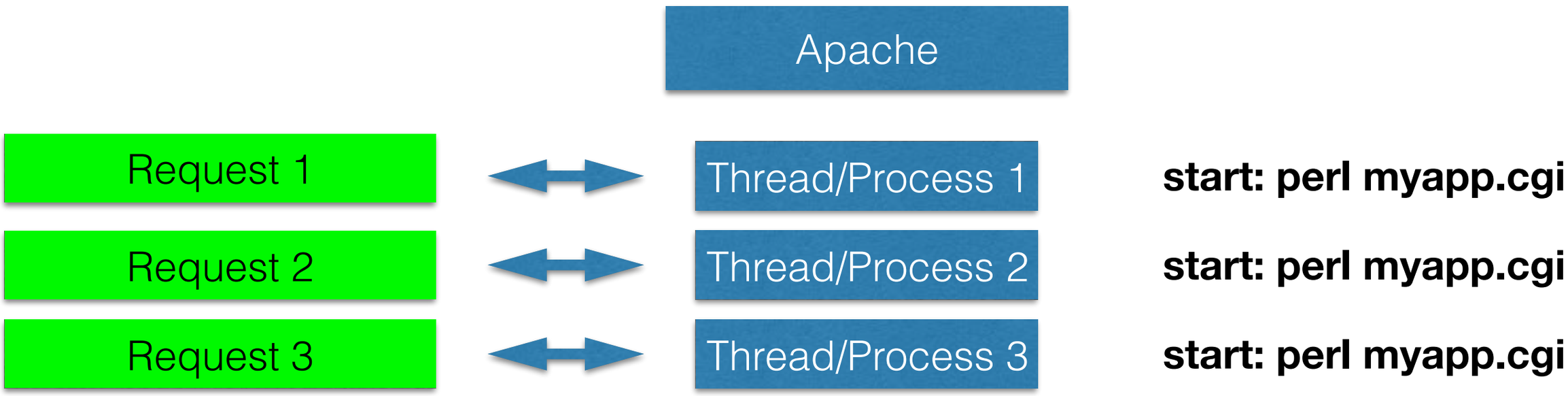
I will briefly discuss and diagram each of these.

CGI

Apache Worker Pool (you see many :)

Send Request

Launches Perl interpreter to process.



As each request comes in, a perl interpreter starts parsing the script and it processes

Works well in a lightly used server.

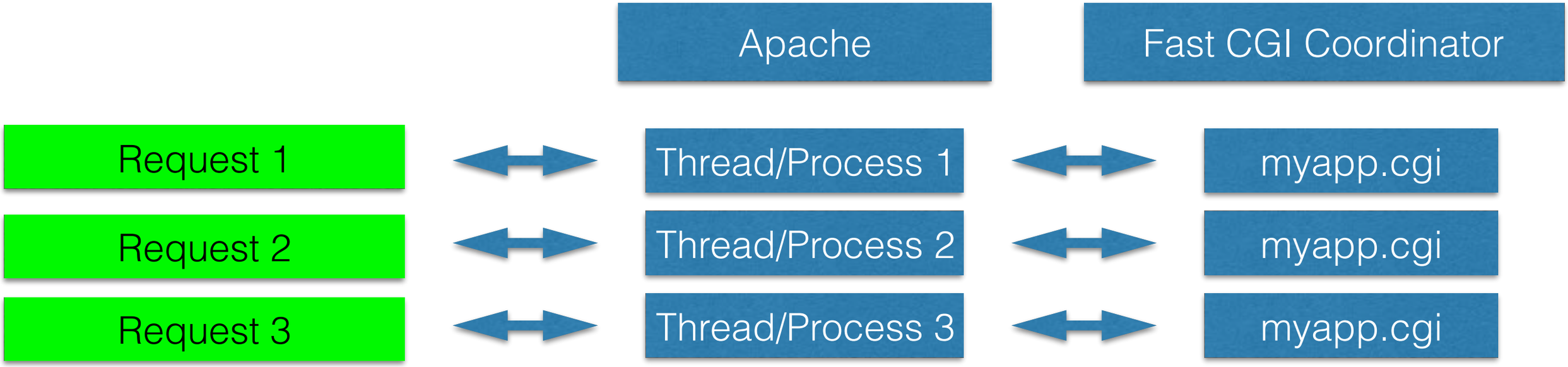
FastCGI

Apache Worker Pool (you see many :)

Send Request

Receives Request

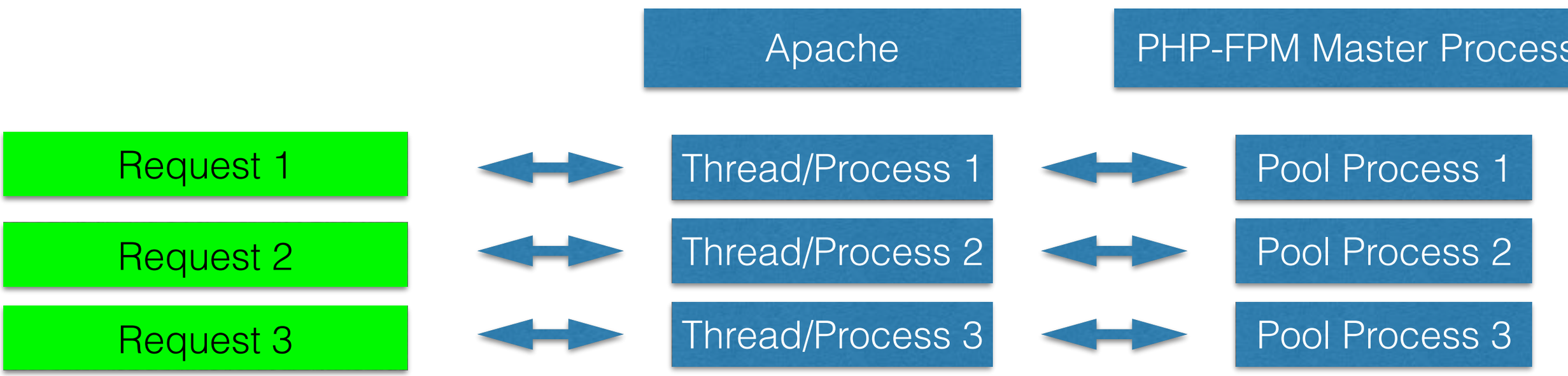
Fast CGI Pool



In this scenario:

- **Request comes into Apache pool**
- **Apache sends request to Fast CGI pool via unix socket**
- **The Fast CGI pool has already parsed myapp.cgi and is frozen and ready to work**
- **I do not need to pay the startup cost for myapp.**

PHP-FPM



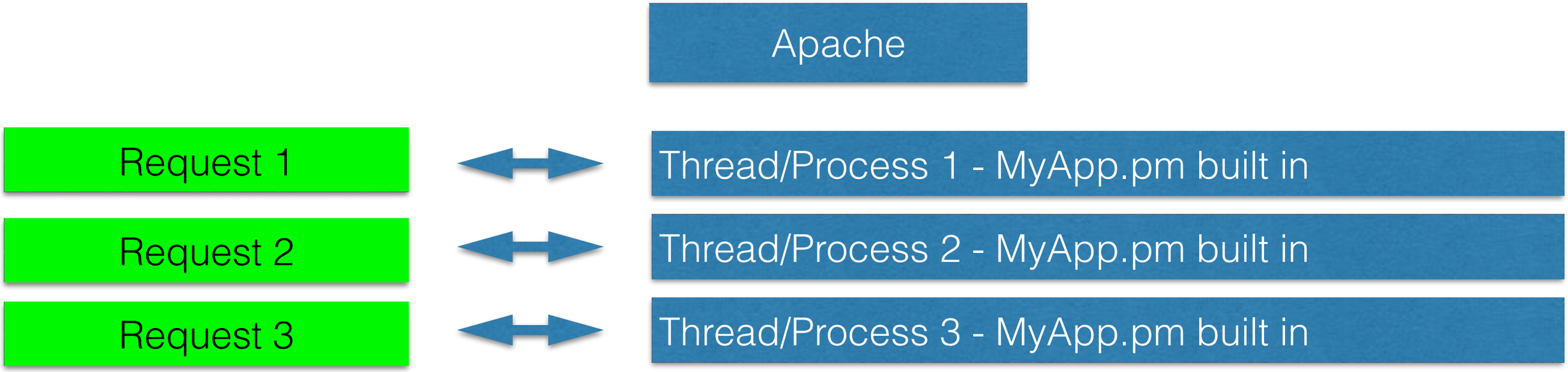
FPM stands for FastCGI Process Manager. Hence PHP-FPM is really FastCGI for PHP. The difference is FPM is built and optimized for PHP, but the benchmark differences between FastCGI and FPM are minimal.

mod_perl

Apache Worker Pool (you see many :)

Send Request

Receives Request



In this scenario:

- **Request comes into Apache pool**
- **Apache runs MyApp::handler**
- **Fast CGI is GREAT! but mod_perl has it already.**
- **No need for second pool.**

Mod Perl Cons

- These apply to FastCGI as well.

- DBH connections can timeout and need to be refreshed periodically if the instance remains idle long enough. It is fairly easy to work around though.

- Resources are not necessarily released when I want them gone. I also need to review global resources to make sure they are properly managed.

- The instances run for very long times, so if a bug is present that prevents an the app to continue, this can be catastrophic. With CGI this is not much of an issue, assuming the first response is correct.

Cool PHP-FPM Parameters

- In my work with PHP-FPM I love some parameters that they offer. Some can be done with FastCGI or ModPerl but not all.
- On Demand processing.
- Max Children, defines maximum number of PHP-FPM processes in the pool.
- Process Idle Timeout, if this pool process is idle this long it will be reaped.
- Max Requests, maximum number of requests before this process is recycled, limiting memory leak damage.

mod_perl : The How

Simple to setup and use

/etc/apache2/sites-enabled/000-default.conf

```
root@julian-Lubuntu:/etc/apache2/sites-enabled# cat 000-default.conf
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    <Location /myapp>
        SetHandler perl-script
        PerlResponseHandler  PerlMongers::MyApp
    </Location>
</VirtualHost>
```

/etc/apache2/apache2.conf at the bottom

```
/etc/apache2/apache2.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

PerlRequire /var/www/perl/startup.pl
```

/var/www/perl/startup.pl

```
use lib qw(/var/www/perl);
1;
```

```
root@julian-Lubuntu:/var/www# tree perl
perl
├── PerlMongers
│   └── MyApp.pm
└── startup.pl
```

My Angular app POST's a JSON document instead of the normal ?field=val&field=val
Nothing wrong with the latter, but I prefer JSON.

```
$scope.openchart_actual = function (chart_num)
{
    $scope.mt_chart_tool_message= "loading chart data ...";

    console.log ("load");
   $scope.my_chart_tool_message = "Button Pressed";

    $http.post('/myapp',
    {
        DO: 'CHARTTOOL_CHART',
        my_chart: chart_num
    }).
    success(function (data, status, headers, config)
    {
        console.log ("success");
        console.log (data);

       var my_chart = data.my_chart;
       $scope.my_chart_tool_message = "mod_perl returned my_chart :" + my_chart + ":";
    }).
    error(function (data, status, headers, config)
    {
       $scope.my_chart_tool_message = "mod_perl big error :" + data + ":";
    });
}
```

```
package PerlMongers::MyApp;

use strict;
use warnings;

use Apache2::RequestRec ();
use Apache2::RequestIO ();
use Apache2::Request ();

use Apache2::Const -compile => qw(OK);

use Data::Dumper;
use JSON;
```

Note: you could use CGI.pm or some similar to process the POST.

If you do:

$json_doc = $cgi->param('POSTDATA');

Also Note, you reuse this module over and over again so if you use persistent db connections you may have to refresh them as stale ones often get disconnected by the db.

```
sub handler {
    my $r = shift;
    my $buffer = "";
    my $my_chart = 0;

    $r->content_type('application/json');

    if ($ENV{'REQUEST_METHOD'} eq "POST")
    {
        while (<STDIN>)
        {
            $buffer .= $_;
        }

        my $json = JSON->new->allow_nonref;
        my $ref = $json->decode( $buffer );

        if (defined $ref && exists $ref->{'my_chart'}
        {
            $my_chart = $ref->{'my_chart'};
        }
}

    print qq~{
        "my_chart" : $my_chart
}
~;

    return Apache2::Const::OK;
}

1;
```

My Chart Tool

localhost/charttool/index.html

My Chart App

- Open Chart 1
- Open Chart 2

Nothing Pressed Yet

My Chart Tool

localhost/charttool/index.html

My Chart App

- Open Chart 1
- Open Chart 2

mod_perl returned my_chart :2:

My Chart Tool

localhost/charttool/index.html

My Chart App

- Open Chart 1
- Open Chart 2

mod_perl returned my_chart :1:

Further Research

- I may be wrong but the other Web Frameworks seem to be based on PSGI/Plack
- So I looked at this page:
- http://plackperl.org/

Servers

Plack (web server adapters)
Plack core includes a CGI runner (for running any PSGI application as a CGI script), a FastCGI daemon and mod_perl handlers for Apache1 and 2.

So in reality, my focus is on lower level efficient access. The other frameworks use this concept and build from there. We are not far from each other.