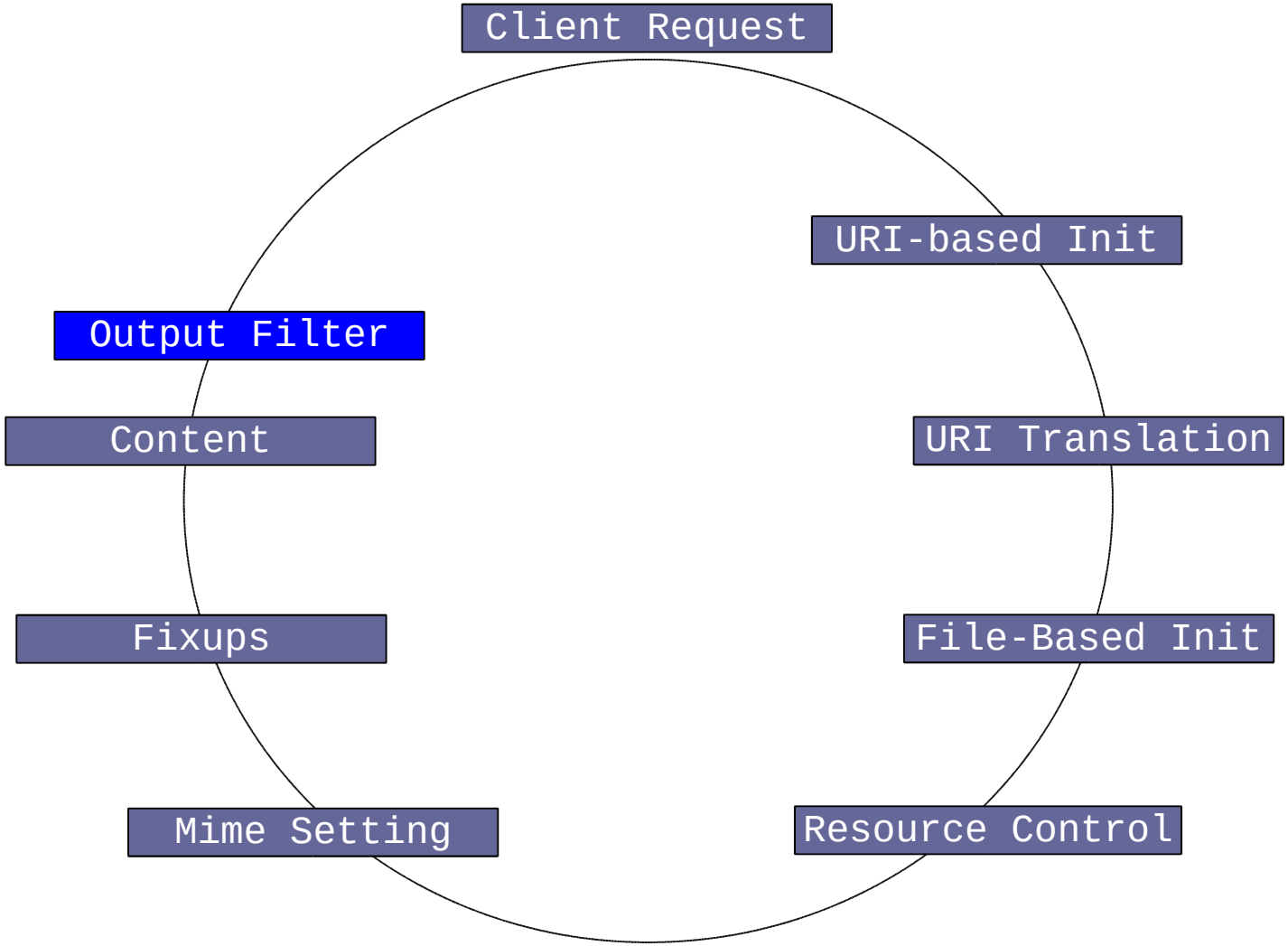
mod_perl 2.0 Filters

Geoffrey Young

geoff@modperlcookbook.org

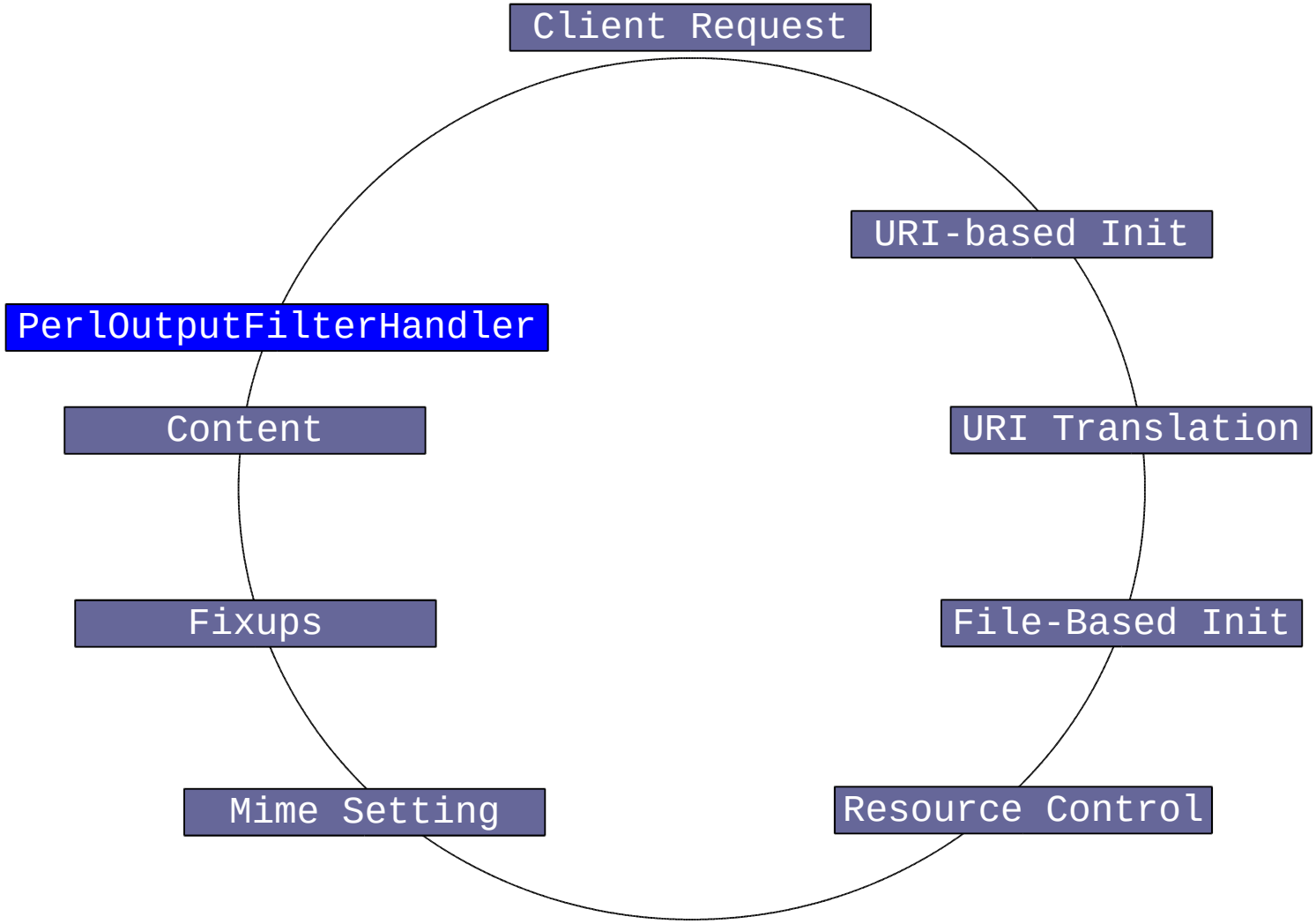
Output Filters



Output Filters

- New in Apache 2.0
- Allow you to post-process content *after* the content phase has run
- mod_perl has been able to filter content for years
  - limited to mod_perl generated content
    - mod_perl can do lots, like CGI and SSI
- Output filters let you filter *everything*
  - no matter who generates the content

PerlOutputFilterHandler



```
package My::Filter;

use Apache2::Filter ();

use Apache2::Const qw(OK);

sub handler {
  my $f = shift;
  while ($f->read(my $buffer, 1024)) {
    # do something with $buffer
    $f->print($buffer);
  }
  return OK;
}

1;
```

httpd.conf

```
PerlOutputFilterHandler My::Filter
```

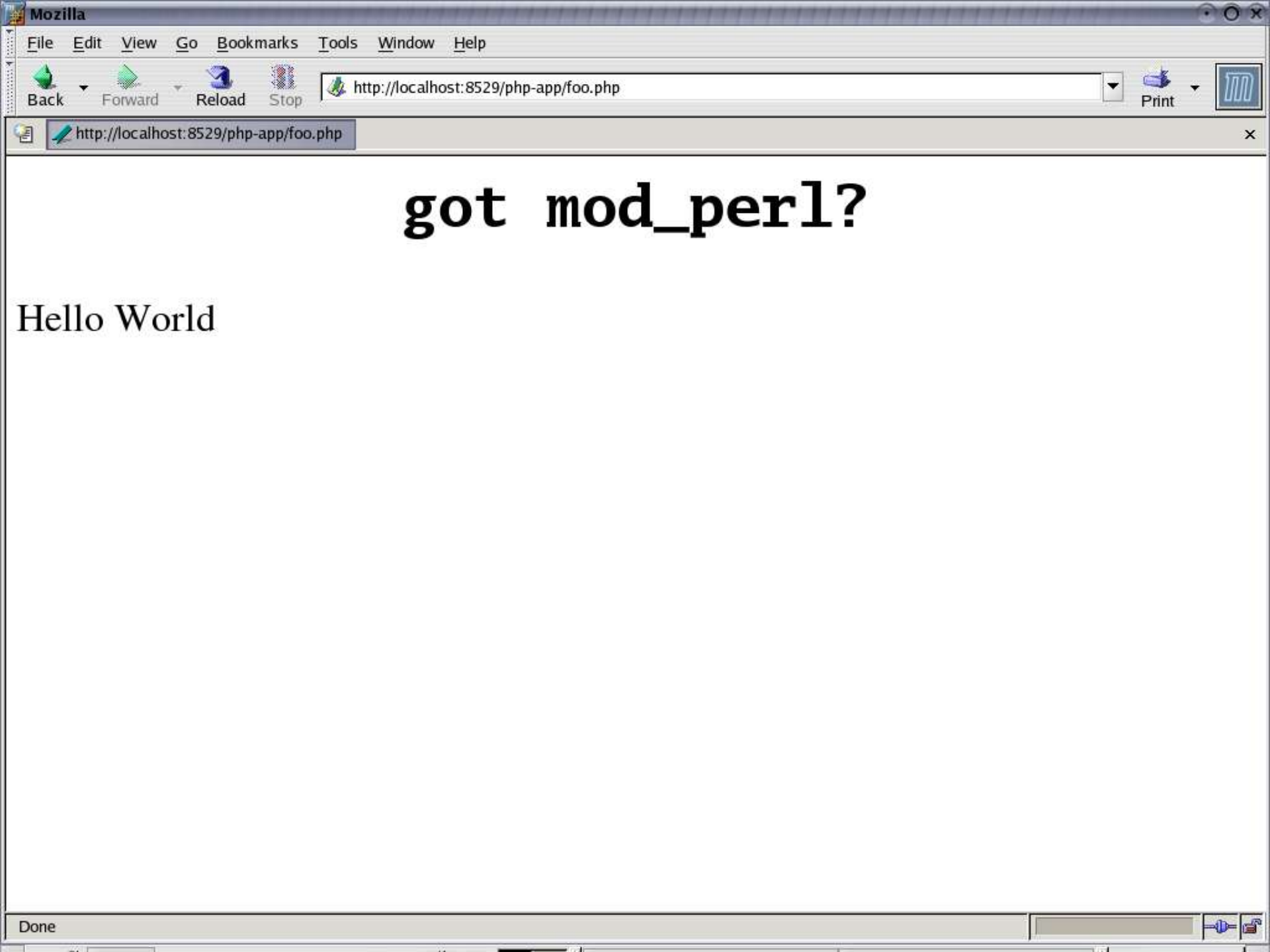
Fun with PHP

- That filter wasn't all that interesting
- Let's mess with PHP

httpd.conf

```
# alter _all_ PHP pages (just a bit)
PerlOutputFilterHandler Apache::Hijack
```

```
package Apache::Hijack;

use Apache2::Filter ();
use Apache2::RequestRec ();

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

use strict;

sub handler {

  my $f = shift;
  my $r = $f->r;

  return Apache2::Const::DECLINED
    unless $r->handler eq 'php-script' or
           $r->handler eq 'application/x-httpd-php';

  while ($f->read(my $buffer, 1024)) {
    $buffer =~ s!(<body>)!$1<h1>got mod_perl?</h1>!i;

    $f->print($buffer);
  }

  return Apache2::Const::OK;
}
1;
```


Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8529/php-app/foo.php
Print
http://localhost:8529/php-app/foo.php
got mod_perl?
Hello World
Done

```
package Apache::Hijack;

use Apache2::Filter ();
use Apache2::RequestRec ();

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

use strict;

sub handler {

  my $f   = shift;
  my $r   = $f->r;

  return Apache2::Const::DECLINED
    unless $r->handler eq 'php-script' or
           $r->handler eq 'application/x-httpd-php';

  my $extra;

  while ($f->read(my $buffer, 1024)) {

    $buffer = $extra . $buffer if $extra;

    $buffer = substr($buffer, 0, - length($extra))
      if (($extra) = $buffer =~ m/(<[^>]*)$/);

    $buffer =~ s!(<body>)!$1<h1>got mod_perl?</h1>!i;

    $f->print($buffer) unless $extra;
  }

  return Apache2::Const::OK;
}
1;
```

```
package Apache::Hijack;

use Apache2::Filter ();
use Apache2::RequestRec ();
use APR::Table ();

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

use strict;

sub handler {

  my $f   = shift;
  my $r   = $f->r;

  return Apache2::Const::DECLINED
    unless $r->handler eq 'php-script' or
           $r->handler eq 'application/x-httpd-php';

  $r->headers_out->unset('Content-Length');
  $r->headers_out->set('X-Powered-By' => 'mod_perl 2.0');

  my $extra;

  while ($f->read(my $buffer, 1024)) {
    $buffer = $extra . $buffer if $extra;

    $buffer = substr($buffer, 0, - length($extra))
      if (($extra) = $buffer =~ m/(<[^>]*)$/);

    $buffer =~ s!(<body>)!$1<h1>got mod_perl?</h1>!i;

    $f->print($buffer) unless $extra;
  }

  return Apache2::Const::OK;
}
1;
```

```
package Apache::HijackFull;

use Apache2::Filter ();
use Apache2::RequestRec ();
use APR::Table ();

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

use strict;

sub handler {

  my $f   = shift;
  my $r   = $f->r;

  return Apache2::Const::DECLINED
    unless $r->handler eq 'php-script' or
           $r->handler eq 'application/x-httpd-php';

  my $context;

  unless ($f->ctx) {
    $r->headers_out->unset('Content-Length');
    $r->headers_out->set('X-Powered-By' => 'mod_perl 2.0');

    $context = { extra => undef };
  }

  $context ||= $f->ctx;

  while ($f->read(my $buffer, 1024)) {

    $buffer = $context->{extra} . $buffer if $context->{extra};

    if (($context->{extra}) = $buffer =~ m/(<[^>]*)$/) {
      $buffer = substr($buffer, 0, - length($context->{extra}));
    }

    $buffer =~ s!(<body>)!$1<h1>got mod_perl?</h1>!i;

    $f->print($buffer) unless $context->{extra};
  }

  $f->ctx($context) unless $f->seen_eos;

  return Apache2::Const::OK;
}
1;
```

A Better Example

- OK, that was fun, but not very useful
- Let's take a close look at something real...

Apache::Clean

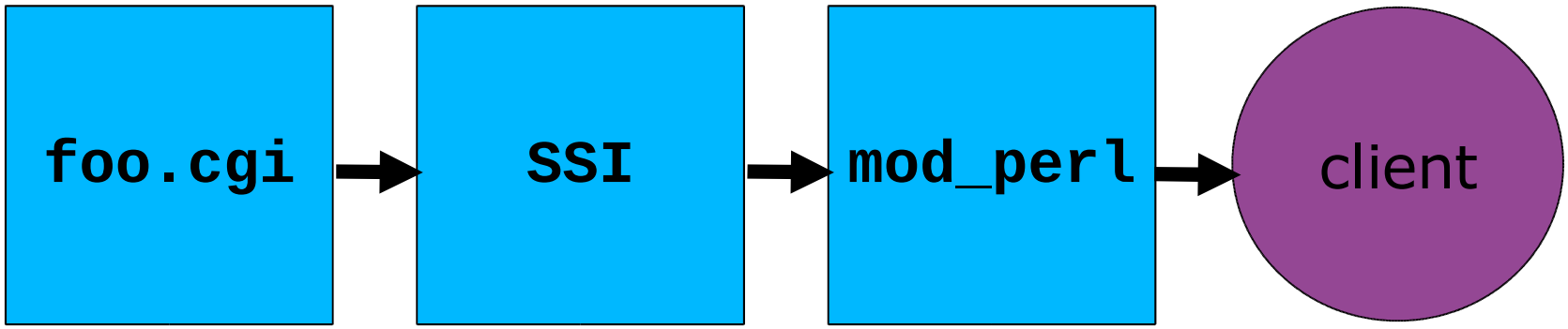
- Apache::Clean is a mod_perl 2.0 filter that scours HTML content and makes it smaller
  - changes to <b>
  - changes < to <
  - and so on
- Uses HTML::Clean from CPAN
- Operates on and and all content
  - mod_perl-generated or otherwise

Configuration

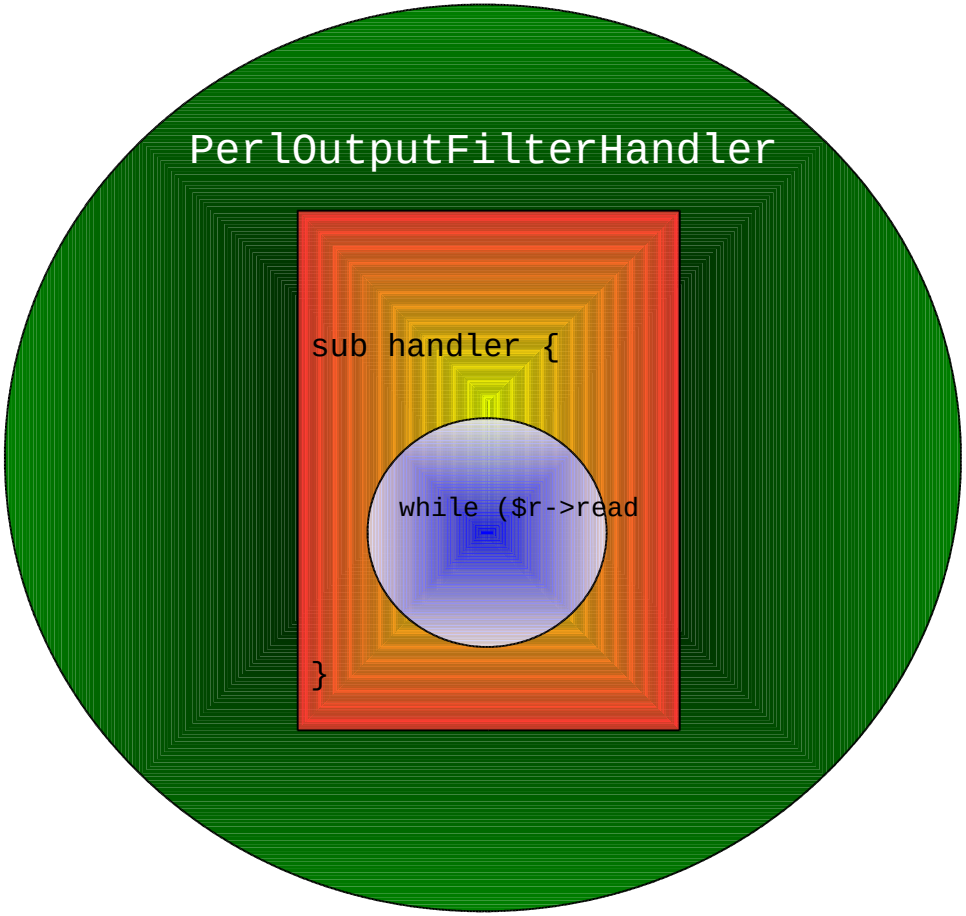
- Here's a sample `httpd.conf`

```
Alias /cgi-bin /usr/local/apache2/cgi-bin

<Location /cgi-bin>
  SetHandler cgi-script

  SetOutputFilter INCLUDES
  PerlOutputFilterHandler Apache::Clean

  PerlSetVar CleanOption shortertags
  PerlAddVar CleanOption whitespace

  Options +ExecCGI +Includes
</Location>
```

Pipeline



Filter Context



Filter Context

- Filters are called more than once per request
  - unlike everybody else
- Sometimes this matters, sometimes it doesn't
- Filter context is the way to communicate between the multiple calls

```
package Apache::Clean;

use Apache2::Filter ();       # $f
use Apache2::RequestRec ();   # $r
use Apache2::RequestUtil (); # $r->dir_config()
use Apache2::Log ();          # $log->info()
use APR::Table ();            # dir_config->get()

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

use HTML::Clean ();

use strict;

sub handler {

  my $f   = shift;

  my $r   = $f->r;

  my $log = $r->server->log;

  # we only process HTML documents
  unless ($r->content_type =~ m!text/html!i) {
    $log->info('skipping non-html document', $r->uri);

    return Apache2::Const::DECLINED;
  }
```

```
my $ctx;

unless ($f->ctx) {
  # these are things we only want to do once no matter how
  # many times our filter is invoked per request

  # parse the configuration options
  my $level = $r->dir_config->get('CleanLevel') || 1;


  my %opt = map {$_ => 1} $r->dir_config->get('CleanOption');

  # store the configuration
  $ctx = { level   => $level,
           options => \%opt,
           extra   => undef };

  # output filters that alter content are responsible for
  # removing the Content-Length header, but we only need
  # to do this once.
  $r->headers_out->unset('Content-Length');
}

# retrieve the filter context
$ctx ||= $f->ctx;
```

```
# now, filter the content
while ($f->read(my $buffer, 1024)) {

  # prepend any tags leftover from the last buffer
  $buffer = $ctx->{extra} . $buffer if $ctx->{extra};

  # if our buffer ends in a split tag (eg '<strong')
  # save processing the tag for later
  if (($ctx->{extra}) = $buffer =~ m/(<[^>]*)$/) {
    $buffer = substr($buffer, 0, - length($ctx->{extra}));
  }

  my $h = HTML::Clean->new(\$buffer);

  $h->level($ctx->{level});

  $h->strip($ctx->{options});

  $f->print(${$h->data});
}
```

```
  if ($f->seen_eos) {
    # we've seen the end of the data stream

    # print any leftover data
    $f->print($ctx->{extra}) if $ctx->{extra};
  }
  else {
    # there's more data to come

    # store the filter context, including any leftover data
    # in the 'extra' key
    $f->ctx($ctx);
  }

  return Apache2::Const::OK;
}

1;
```