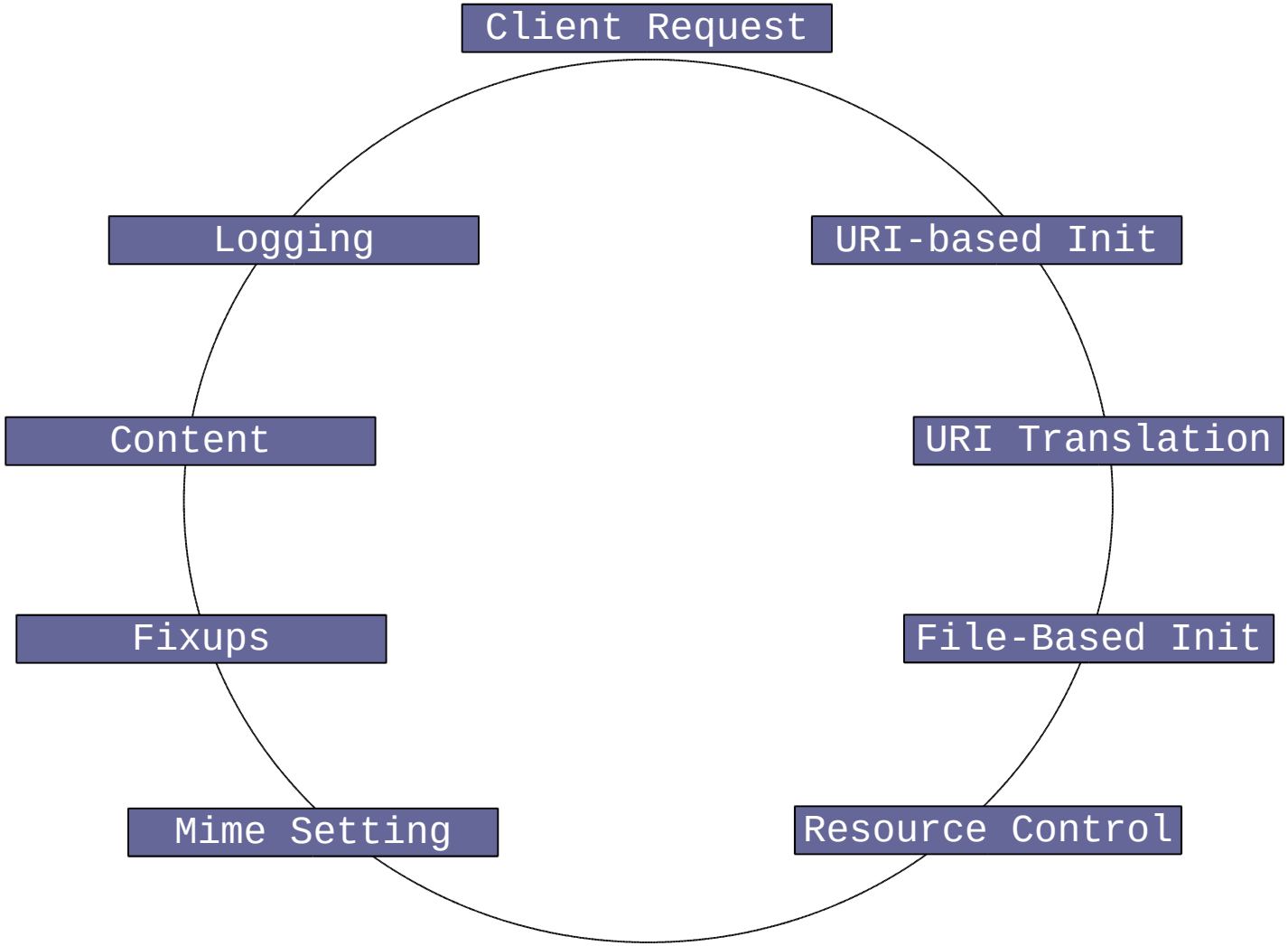
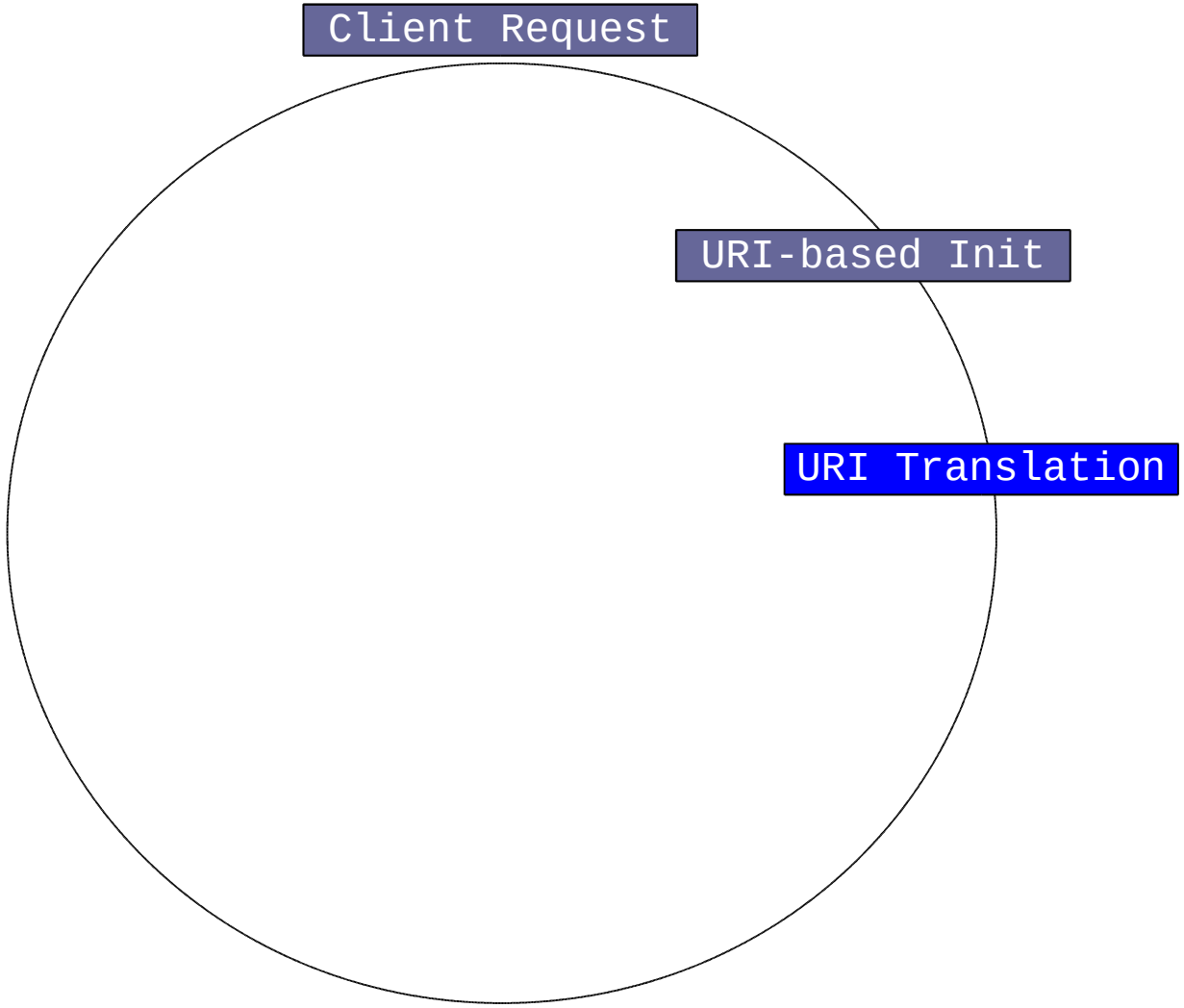
All Your URI are Belong to Us

Geoffrey Young
geoff@modperlcookbook.org

Apache Request Cycle



Apache Request Cycle



URI Translation

- Apache needs to map the URI to a physical file on disk
- Default is to prepend `DocumentRoot` to the URI

```
DocumentRoot /usr/local/apache/htdocs
```

URI Translation

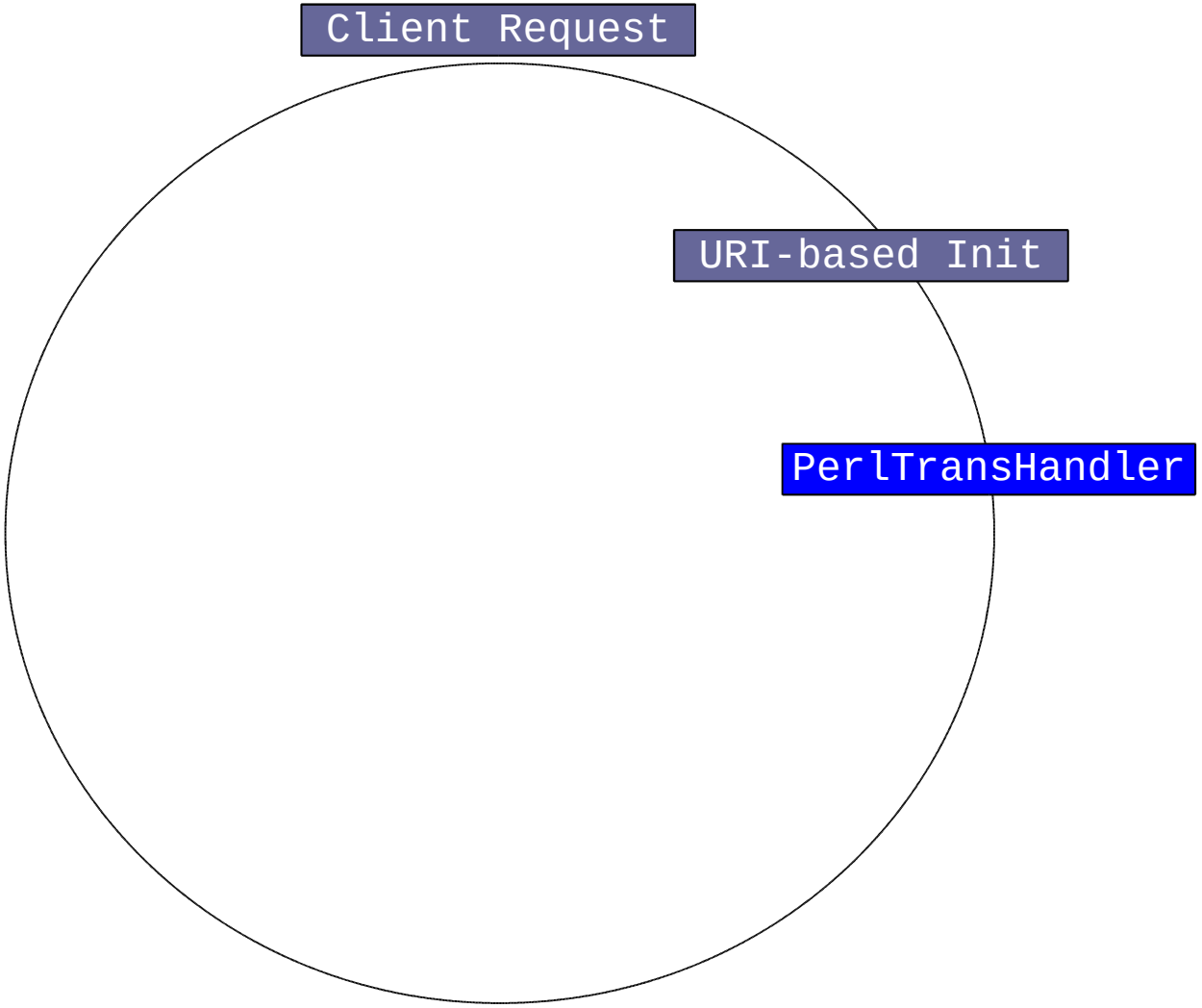
- Directives like `Alias` override the default

```
DocumentRoot /usr/local/apache/htdocs

Alias /manual/ /usr/local/apache/manual/

<Directory /usr/local/apache/manual>
  ...
</Directory>
```

- Some URIs have no associated file, but Apache tries anyway

```
<Location server-status>
  ...
</Location>
```

PerlTransHandler



PerlTransHandler

- Useful for overriding the Apache default
- Allows you to be extremely devious
- There are a few pitfalls of which to be aware

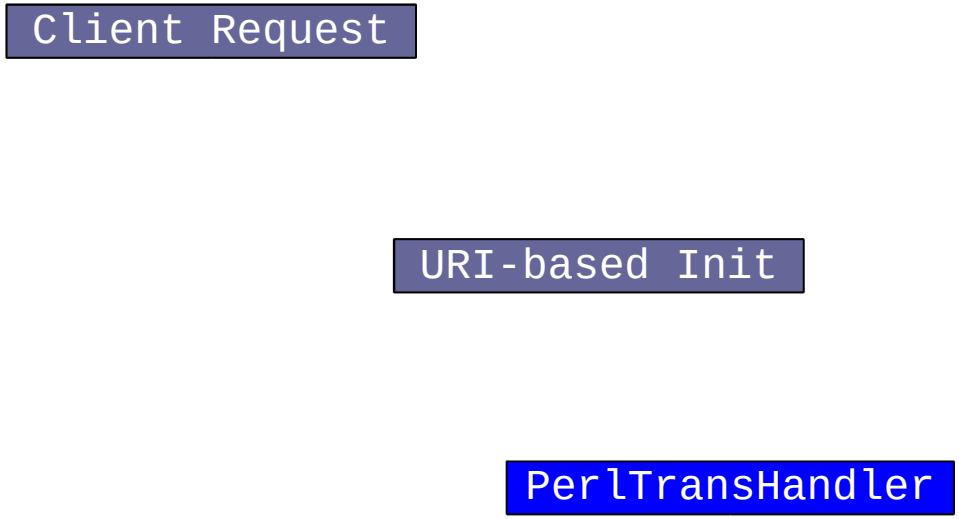
Simple `PerlTransHandler`

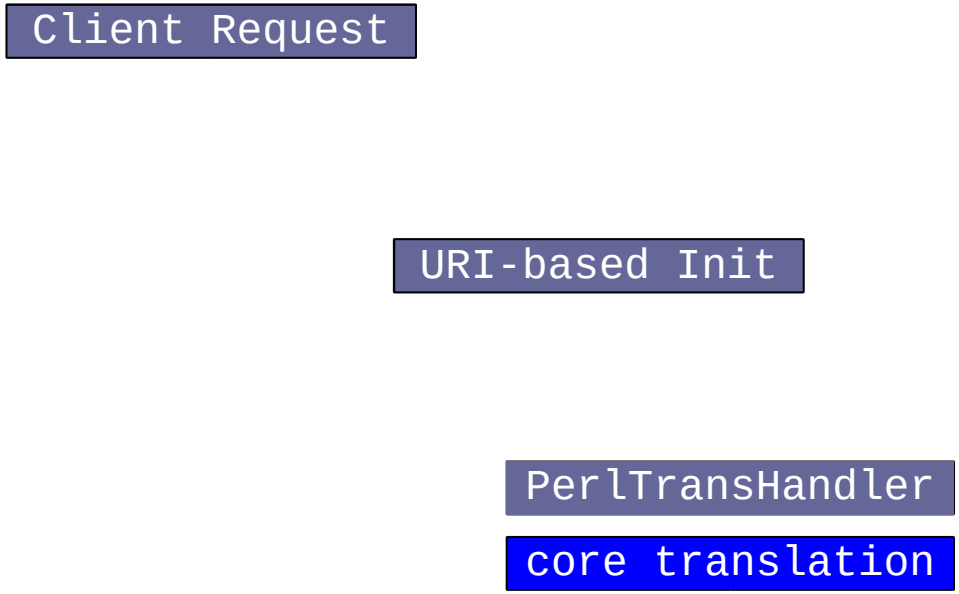
- Be rid of those silly `favicon.ico` requests that end up `404`
- Translate the incoming URI to a common place if it matches `favicon.ico`

```
package Cookbook::Favicon;

use Apache::Constants qw(DECLINED);

use strict;

sub handler {

  my $r = shift;

  $r->uri("/images/favicon.ico")
    if $r->uri =~ m!/favicon\.ico$!;

  return DECLINED;
}

1;
```

Client Request

```
GET /foo/bar/baz/biff/favicon.ico HTTP/1.1
Host: www.example.com
```


Client Request
URI-based Init
PerlTransHandler
URI: /foo/bar/baz/biff/favicon.ico



FILE: /usr/local/apache/htdocs/images/favicon.ico

Setup

- add `Favicon.pm` to `@INC`

  `ServerRoot``/lib/perl/Cookbook/Favicon.pm`

- add to `httpd.conf`

```
PerlModule Cookbook::Favicon
PerlTransHandler Cookbook::Favicon
```

- that's it!

Winner Takes All

- The first URI translation handler to return `OK` ends the phase
  - mod_perl or otherwise
- Perl handlers usually return `DECLINED`
  - lets Apache's core translation engine do the filesystem mapping
- Return `OK` only when you map the file to disk yourself

Why Not Use mod_rewrite?

- our `Cookbook::Favicon` is pretty much the same as

```
RewriteRule /favicon.ico$ /images/favicon.ico
```

- We did it in Perl
- With Perl comes great power

URI-based Sessions

- A simple cookie-less session scheme stores the session in the URI
- Storage can occur in the `PATH_INFO`

```
http://manual/index.html/a92c5e
```

- Or at the start of the URI

```
http://a92c5e/manual/index.html
```

- The second form has some distinct advantages

```
package Cookbook::URISessionManager;

use Apache::Constants qw(DECLINED OK);
use Apache::URI;

sub get_session {

  my $r = shift;

  my $uri = $r->parsed_uri;

  # Separate the MD5 session from the real path.
  my ($session, $path) = $uri->path =~ m!^/([a-fA-F0-9]{32})(/.*)!;

  return DECLINED unless $session;

  # Now, put the session in a note...
  $r->notes(SESSION => $session);

  # ... and set the URI to the proper path.
  $r->uri($path);

  return DECLINED;
}

1;
```

Advantages

- Session parsing is done up front
  - everyone else gets the session from `notes`

```
my $session = $r->notes('SESSION');
```

  - even C handlers!
  - true even if you use `PATH_INFO`
- Browsers take care of adding the session to relative links *for you*

Disadvantages

- The main problem with URI-based sessions is "session leakage"
- Session data will show up in Referer logs whenever someone clicks offsite
- A simple `PerlTransHandler` takes care of that

```
package Cookbook::URISessionManager;

sub get_session {
  ...
}

sub clean_uri {

  my $r = shift;


  my ($uri) = $r->uri =~ m!.*(http://.*)!;

  $r->send_http_header('text/html');

  print<<EOF;
<html>
  <head>
    <meta http-equiv="refresh" content="0;URL=$uri">
  </head>
  <body>
    you should be going <a href="$uri">here</a> soon
  </body>
</html>
EOF

  return OK;
}
1;
```

httpd.conf

```
PerlModule Cookbook::URISessionManager
PerlTransHandler Cookbook::URISessionManager::get_session

<Location /goodbye>
  SetHandler perl-script
  PerlHandler Cookbook::URISessionManager::clean_uri
</Location>
```

Mischievous Behavior

- Simple URI re-mapping is only the beginning
- Apache has this neat, built-in functionality called *proxying*
  - provided you have mod_proxy installed
- With mod_perl and mod_proxy you can proxy just about anything...

Advanced `PerlTransHandler`

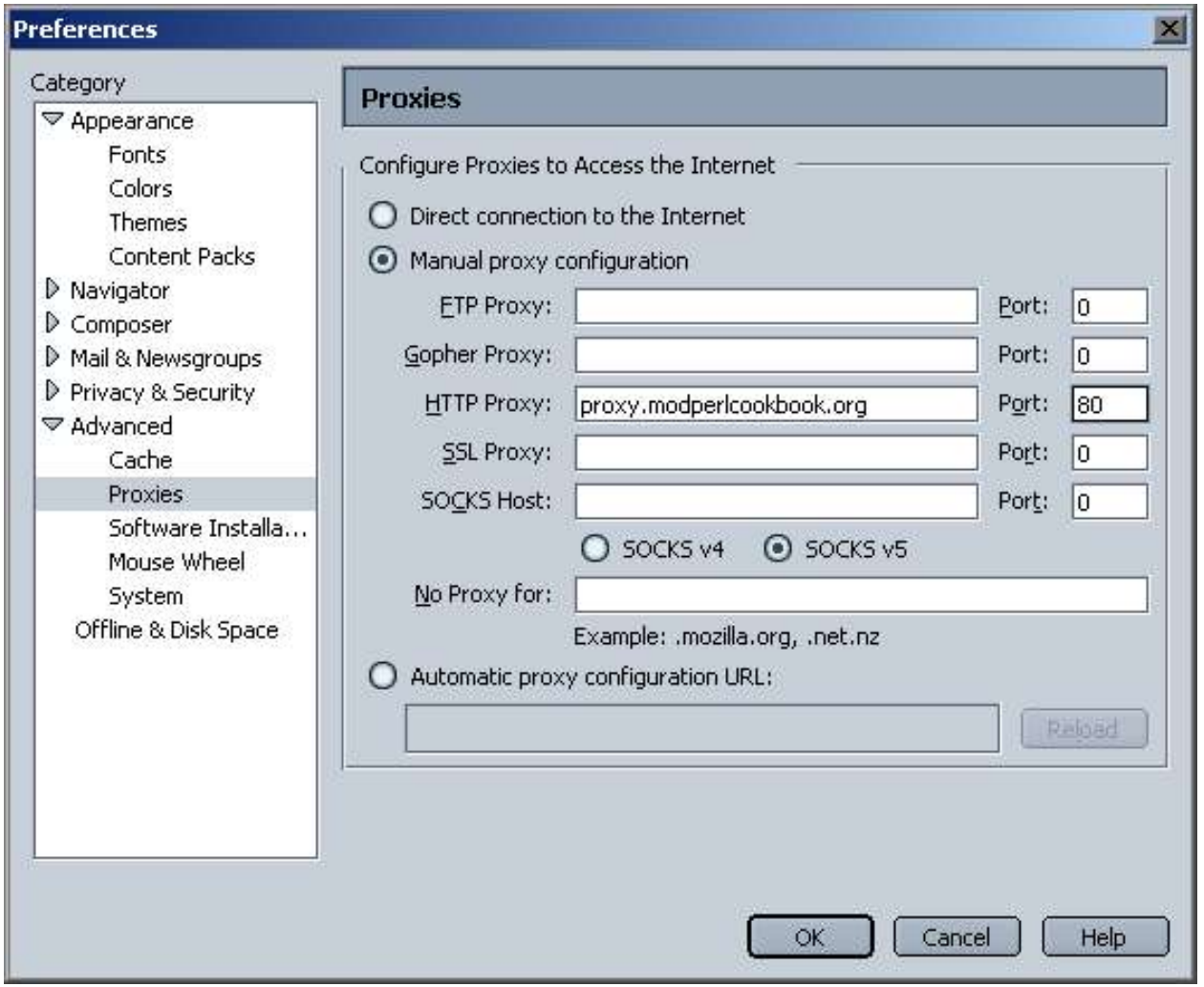
- Create a proxy that uses our local Apache documentation instead of ASF servers
- Intercept proxy requests and silently replace calls to

  `http://httpd.apache.org/docs`

  with

  `/usr/local/apache/htdocs/manual`

Client Setup



Server Setup

- Add this to `httpd.conf`

```
PerlModule My::ManualProxy
PerlTransHandler My::ManualProxy
```

```
package My::ManualProxy;

use Apache::Constants qw(OK DECLINED);

use strict;

sub handler {

  my $r = shift;

  return DECLINED unless $r->proxyreq;

  my (undef, $file) = $r->uri =~ m!^http://(www|httpd).apache.org/(.*)!;

  if ($file =~ m!^docs/!) {

    $file =~ s!^docs/!manual/!;

    $file = join "/", ($r->document_root, $file);

    if (-f $file) {

      $r->filename($file);  # use local disk

      return OK;
    }
  }

  return DECLINED;
}
1;
```

Apache Request Cycle