A Few Cool Things About mod_perl

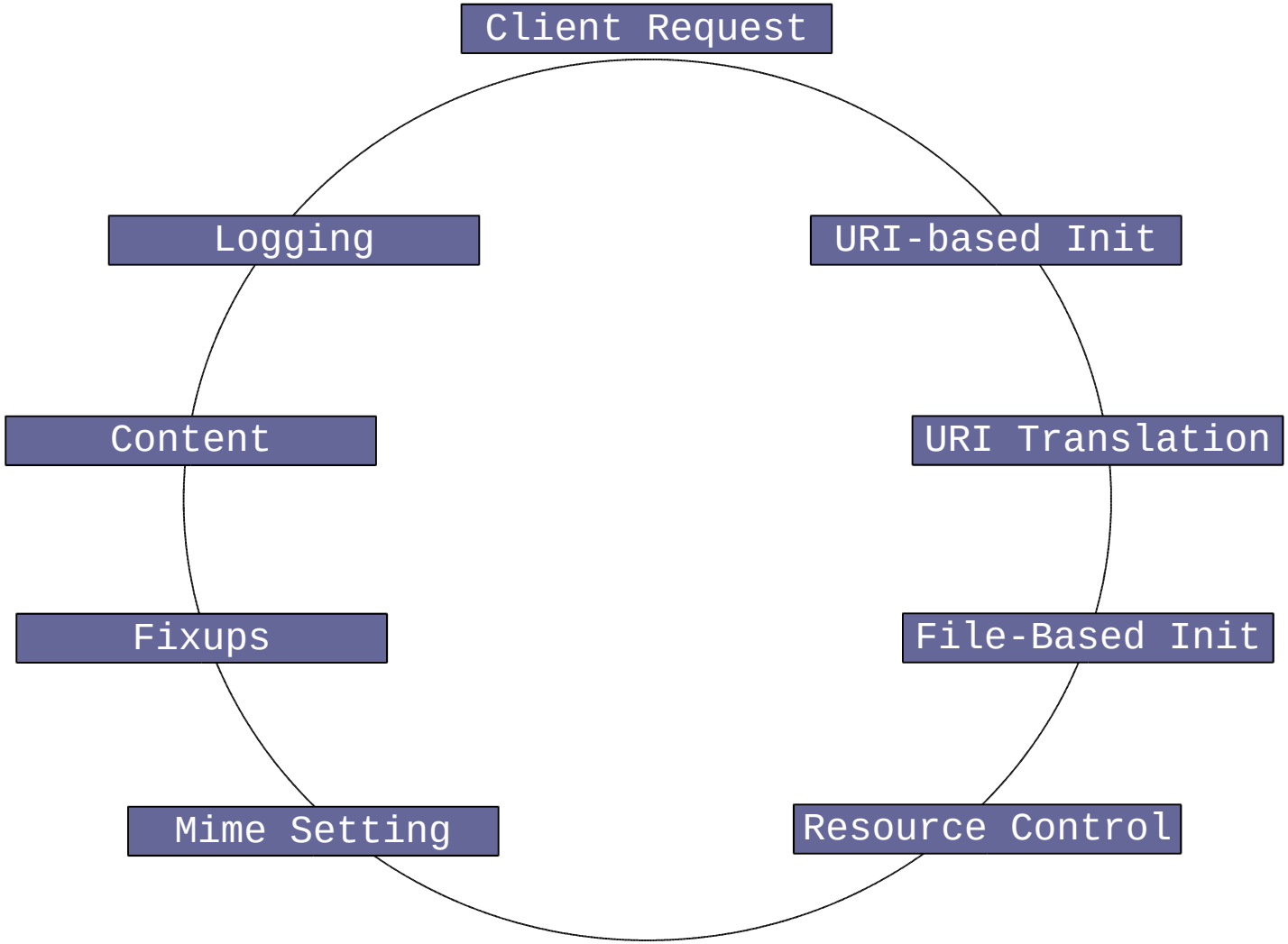
Geoffrey Young

geoff@modperlcookbook.org

Request Phases

- Apache breaks down request processing into separate, logical parts called phases
- each request is stepped through the phases until...
  - all processing is complete
  - somebody throws an "error"
- developers are given the chance to hook into each phase to add custom processing

Apache Request Cycle



So What?

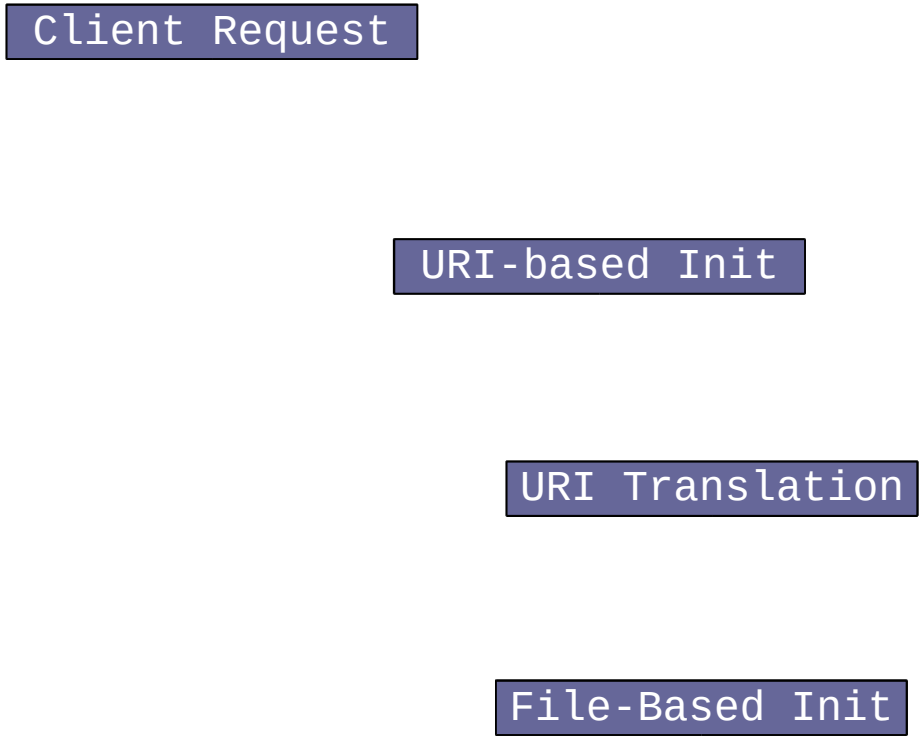
- most Apache users don't worry about the request cycle too much...
- ...but they do use modules that plug into it

... for instance

mod_rewrite:

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

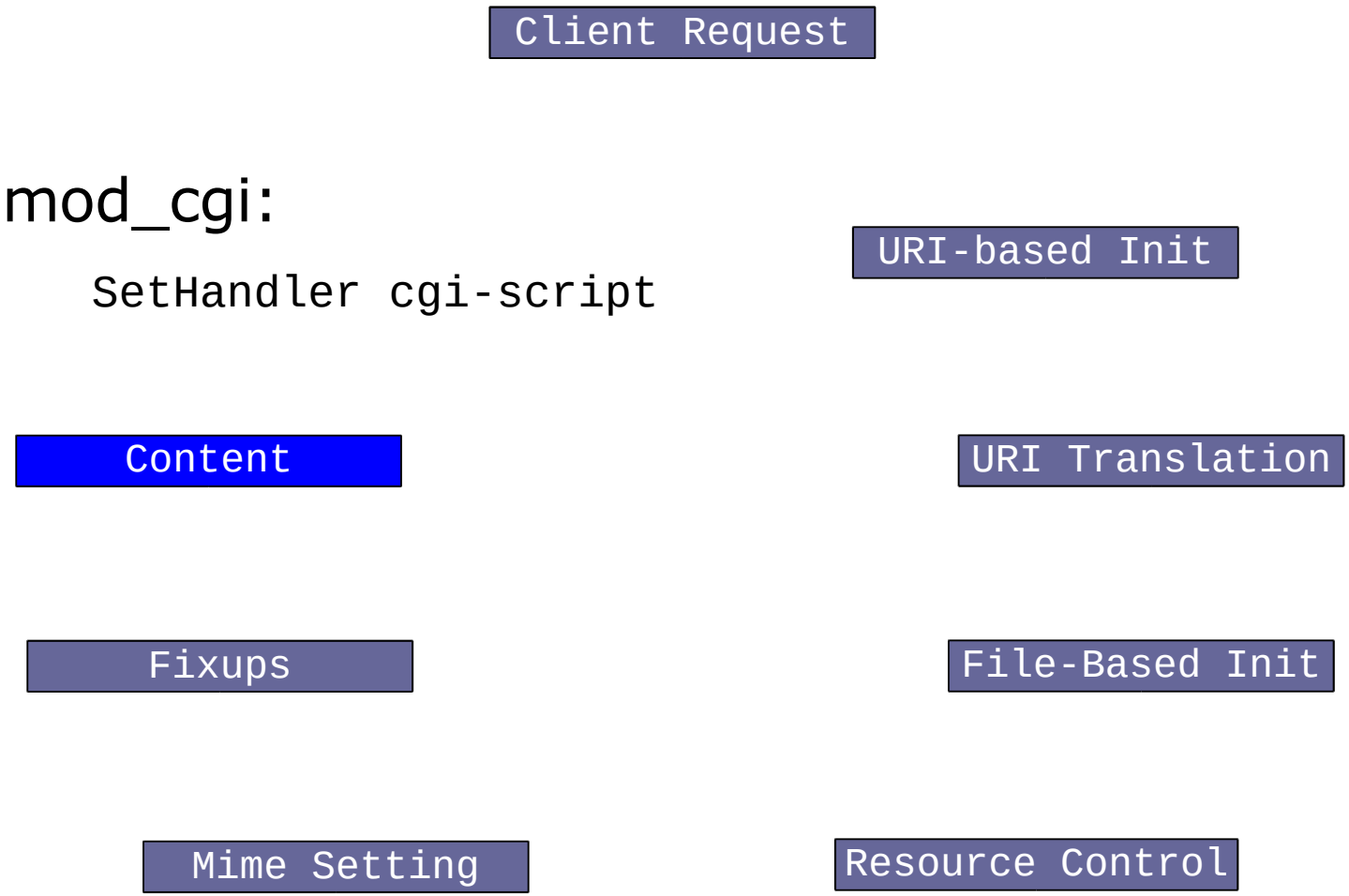
... for instance



mod_auth:

```
AuthUserFile .htpasswd
```

... for instance



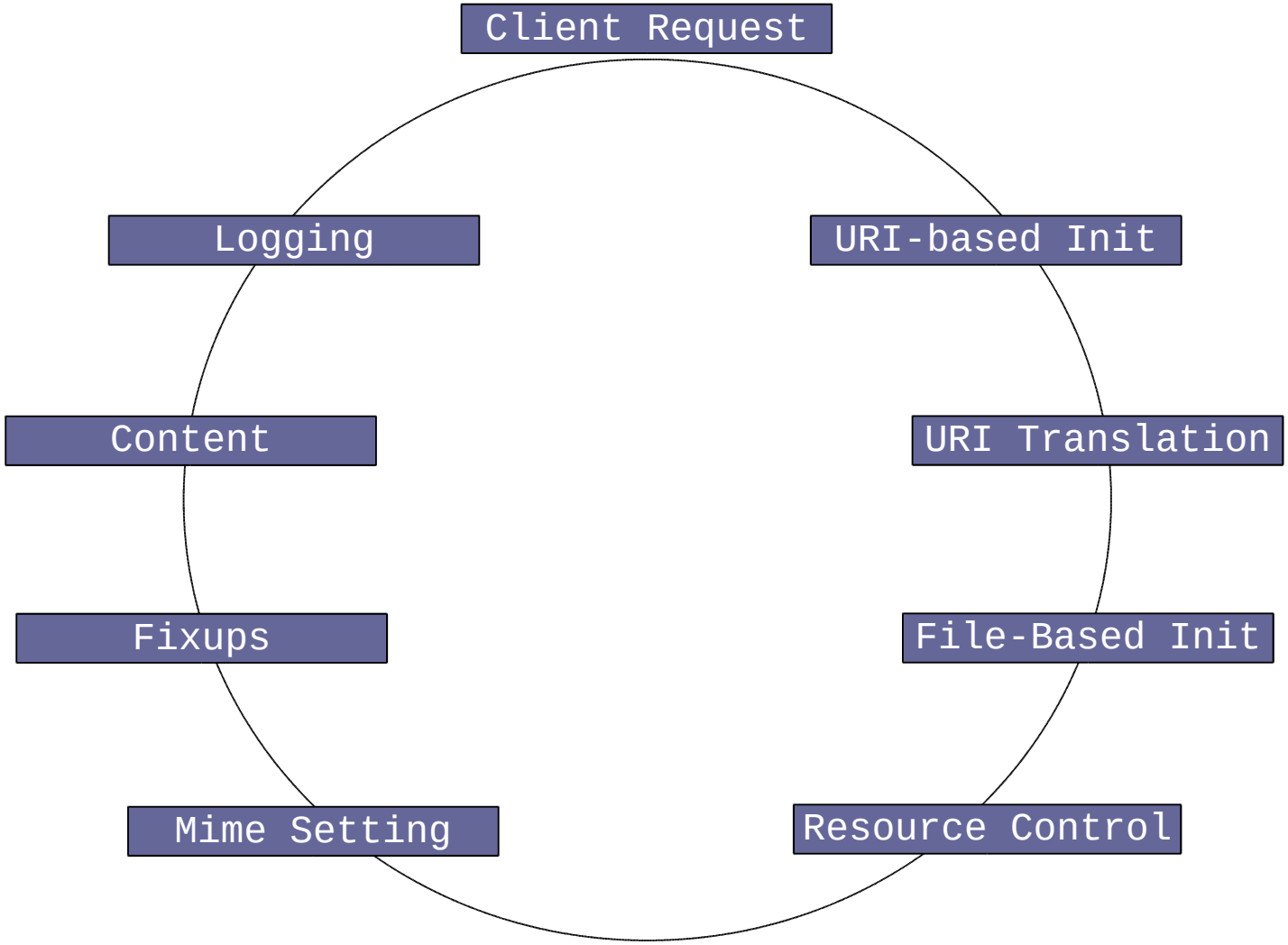
That's great, but...

- breaking down the request into distinct phases has many benefits
  - gives each processing point a role that can be easily managed and programmed
  - makes Apache more like an application framework rather than a content engine
- but you have to code in C

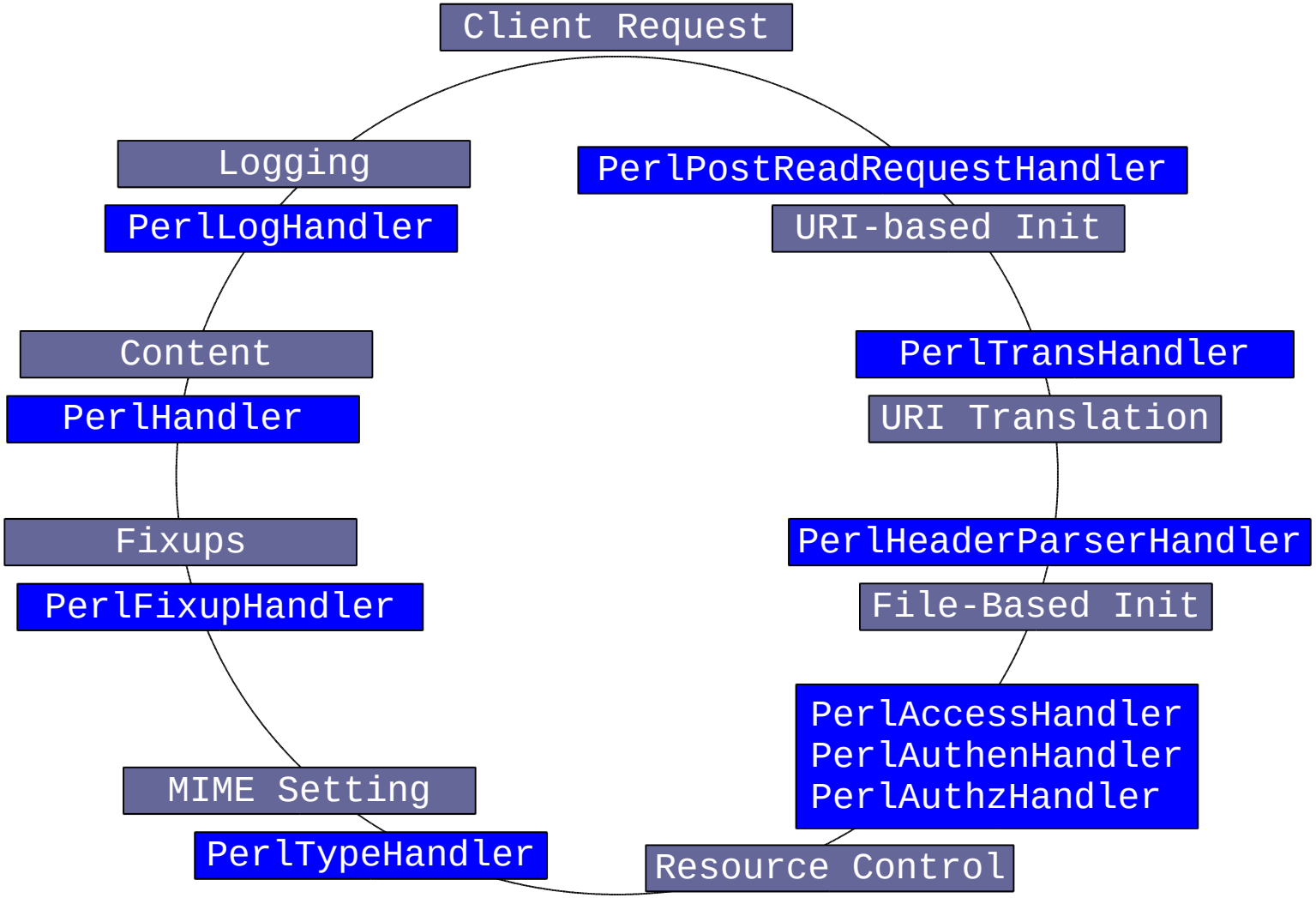
Enter mod_perl

- mod_perl offers an interface to each phase of the request cycle
- opens up the Apache API to Perl code
- allows you to program targeted parts of the request cycle using Perl
- we like Perl

Apache Request Cycle



mod_perl Interface



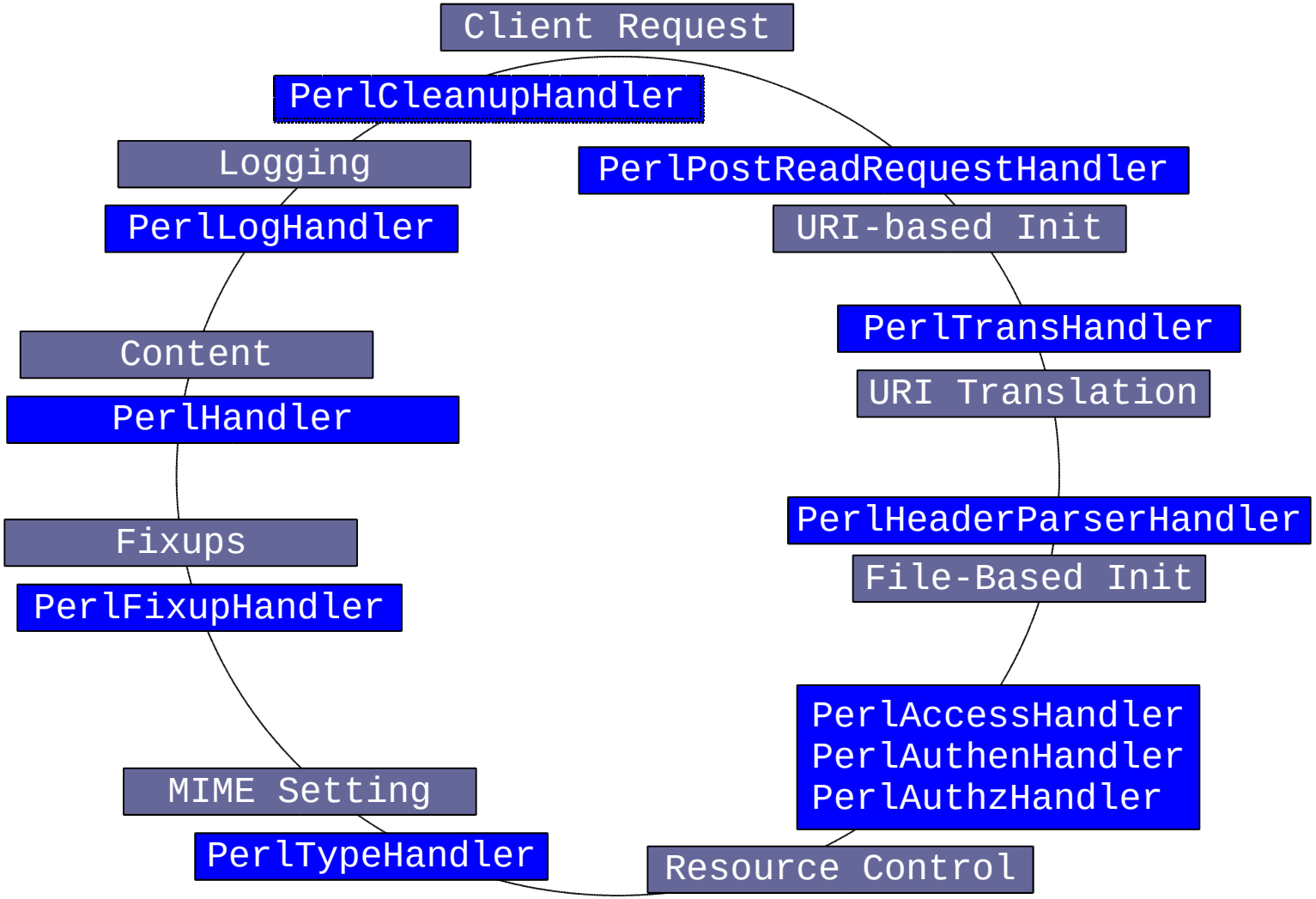
What is mod_perl?

- mod_perl is the Perl interface to the Apache API
- a C extension module, just like mod_cgi or mod_rewrite
- creates a persistent perl environment embedded within Apache

What's the Big Deal?

- mod_perl allows you to interact with and directly alter server behavior
- gives you the ability to "program *within* Apache's framework instead of *around* it"
- allows you to intercept basic Apache functions and replace them with your own (sometimes devious) Perl substitutes
- let's you do it in Perl instead of C

mod_perl Runs First



What is a Handler?

- the term handler refers to processing that occurs during any of the Apache runtime phases
- this includes the request-time phases as well as the other parts of the Apache runtime, such as restarts
- the use of "handler" for all processing hooks is mod_perl specific

Why use Handlers?

- gives each processing point a role that can be easily managed and programmed
  - process request at the proper phase
  - meaningful access to the Apache API
  - break up processing into smaller parts
  - modularize and encapsulate processing
- makes Apache more like an application framework rather than a content engine
- CPAN

For Example...

- Object: to protect our name-based virtual hosts from `HTTP/1.0` requests Apache can't handle

HTTP/1.0 and Host

- HTTP/1.0 does *not* require a Host header
- assumes a "one host per IP" configuration
- this limitation "breaks" name-based virtual host servers for browsers that follow HTTP/1.0 to the letter
  - most send the Host header, so all is well

Let's Fix It!

- Intercept *every* request prior to content-generation and return an error unless...
  - there is a `Host` header
  - the request is an absolute URI

```
package Cookbook::TrapNoHost;

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

use strict;

sub handler {

  my $r = shift;

  # Valid requests for name based virtual hosting are:
  # requests with a Host header, or
  # requests that are absolute URIs.

  unless ($r->headers_in->get('Host') || $r->parsed_uri->hostname) {


    $r->custom_response(BAD_REQUEST,
                        "Oops!  no a Host header?");

    return BAD_REQUEST;
  }

  return DECLINED;
}
1;
```

Setup

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

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

- add to `httpd.conf`

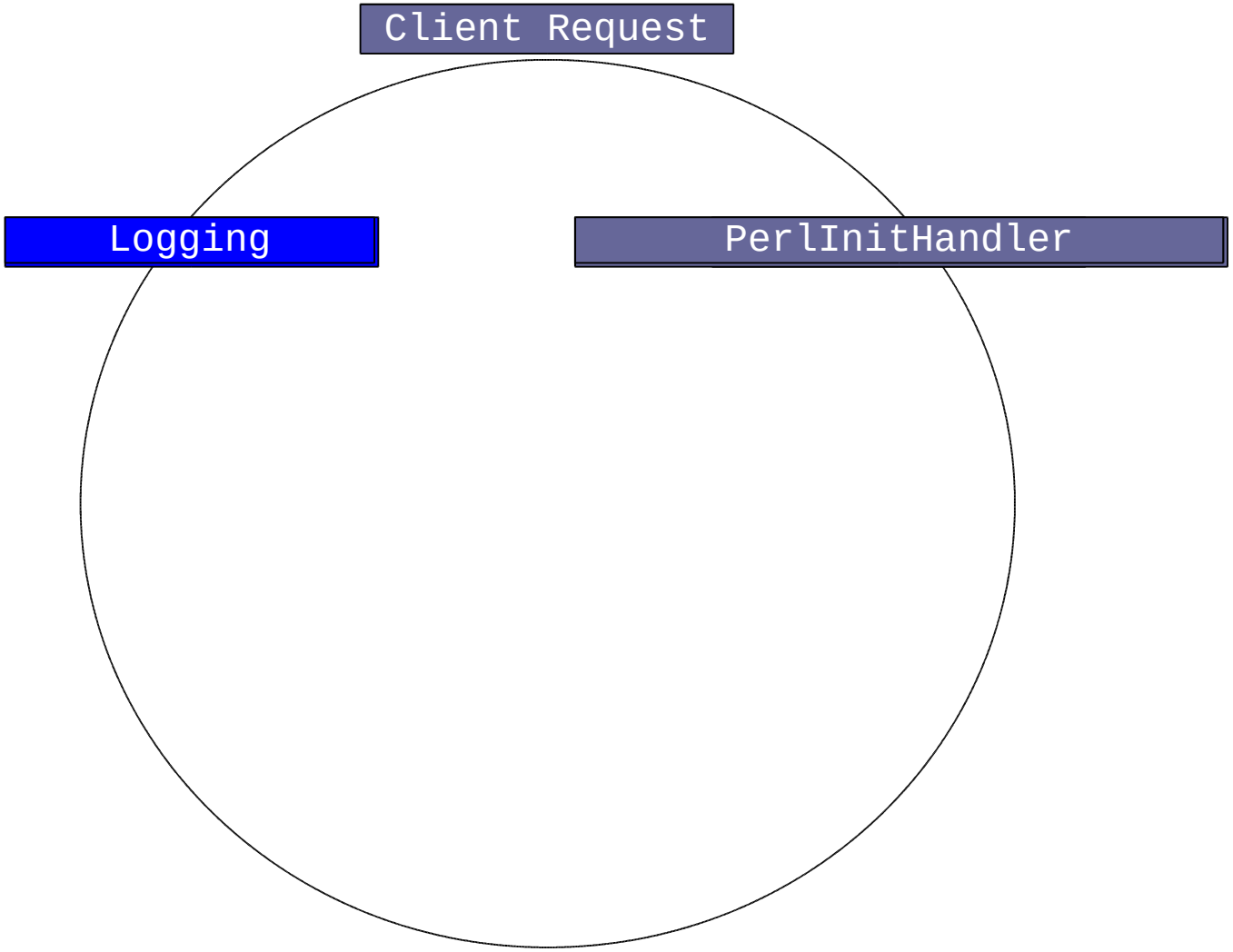
```
PerlModule Cookbook::TrapNoHost
PerlInitHandler Cookbook::TrapNoHost
```

- that's it!

Intercept the Request


```
HTTP/1.1 400 Bad Request
Date: Tue, 04 Jun 2002 01:17:52 GMT
Server: Apache/1.3.25-dev (Unix) mod_perl/1.27_01-dev Perl/v5.8.0
Connection: close
Content-Type: text/html; charset=iso-8859-1

Oops!  no Host header?
```

Errors Skip to Logging



Anatomy of a Handler

- a mod_perl handler is just an ordinary Perl module
- visible through `@INC`
  - including mod_perl extra paths
- contains a `package` declaration
- has at least one subroutine
  - typically the `handler()` subroutine

Apache Request Object

- $r
- passed to handlers as the first subroutine argument

```
$r = shift;  # from @_ passed to handler()
```

- provides access to the Apache class, which provides access to request attributes
- singleton-like constructor, always returning the same object

Request Attributes

- `$r` gives you access to underlying per-request information
  - `$r->headers_in()  # request headers`
  - `$r->headers_out()  # response headers`
  - `$r->user()  # authenticated user`
- `$r` is also the doorway to the official C API
  - `$r->document_root() # ap_document_root`
  - `$r->update_mtime()  # ap_update_mtime`
  - `$r->send_fd()  # ap_send_fd_length`

Apache::Constants

- Apache::Constants class provides over 90 runtime constants

```
use Apache::Constants qw(DECLINED BAD_REQUEST);
```

- the most common are:

```
OK                  # 0
DECLINED            # -1
SERVER_ERROR        # 500
REDIRECT            # 302
```

Return Values

- handlers are expected to return a value
- the return value of the handler defines the status of the request
- Apache defines three "good" return values

  `OK` – all is well

  `DECLINED` – forget about me

  `DONE` – we're finished, start to log

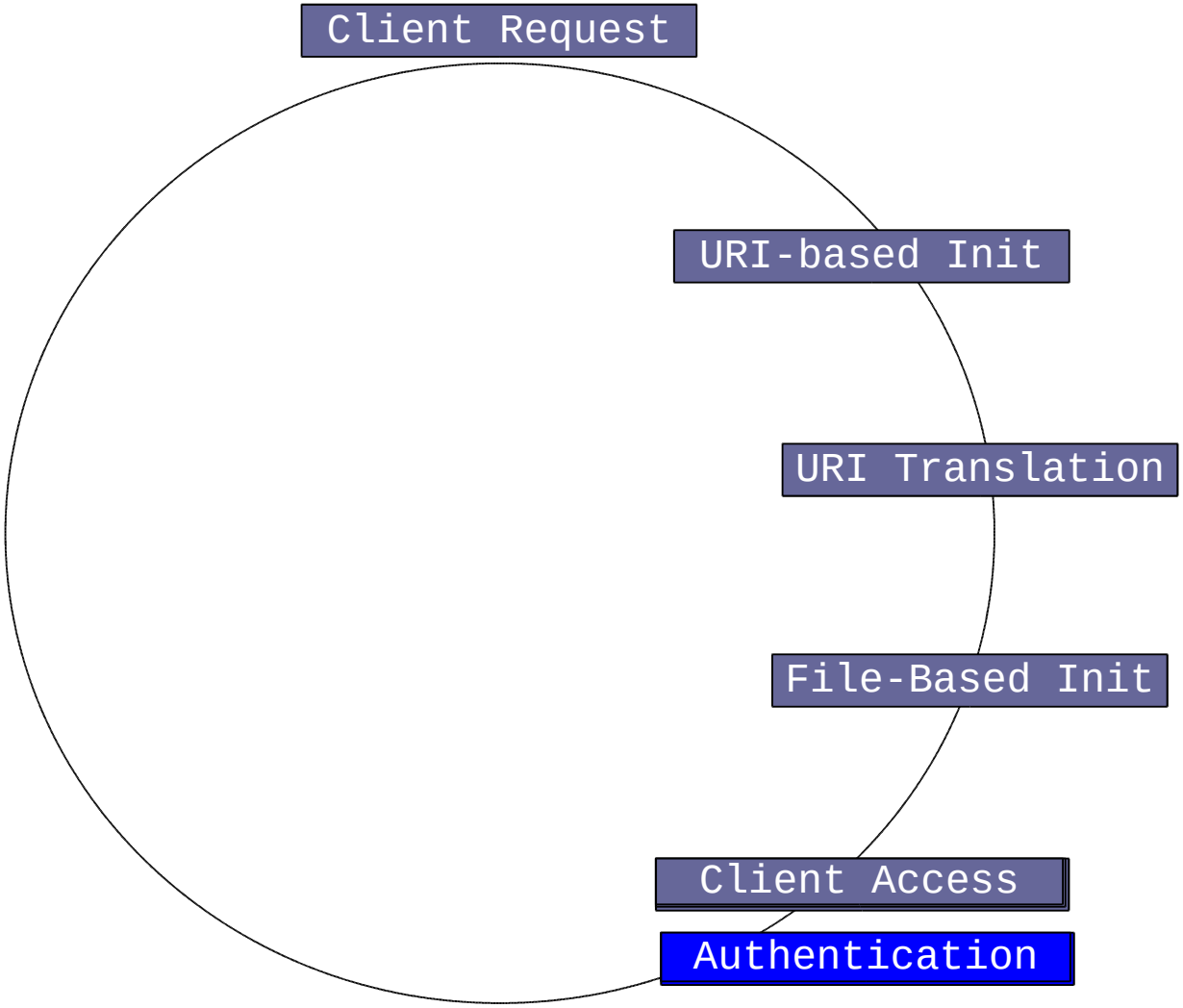
- All other values are "errors" and trigger the `ErrorDocument` cycle

mod_perl 2.0

- Basic mod_perl concepts apply to both 1.0 and 2.0
- The majority of the API is the same
  - method names
- Where the API lives is different
  - package namespaces
- *Lots* of preloading is now required

```
package Cookbook::TrapNoHost;

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

use Apache2::RequestRec ();  # for $r->headers_in()
use APR::Table ();           # for headers_in->get()
use Apache2::Response ();    # for $r->custom_response()
use Apache2::URI ();         # for $r->parsed_uri()
use APR::URI ();             # for parsed_uri->hostname()

use strict;

sub handler {

  my $r = shift;

  unless ($r->headers_in->get('Host') || $r->parsed_uri->hostname) {

    $r->custom_response(Apache2::Const::HTTP_BAD_REQUEST,
                        "Oops!  no Host header?");

    return Apache2::Const::HTTP_BAD_REQUEST;
  }

  return Apache2::Const::DECLINED;
}

1;
```

Resource Control



User Authentication

- Apache default authentication mechanism is mod_auth
- uses a password file generated using Apache's `htpasswd` utility

```
geoff:zzpEyL0tbgwwk
```

User Authentication

- configuration placed in `.htaccess` file or `httpd.conf`

```
AuthUserFile .htpasswd
AuthName "cookbook"
AuthType Basic
Require valid-user
```

How Authentication Works

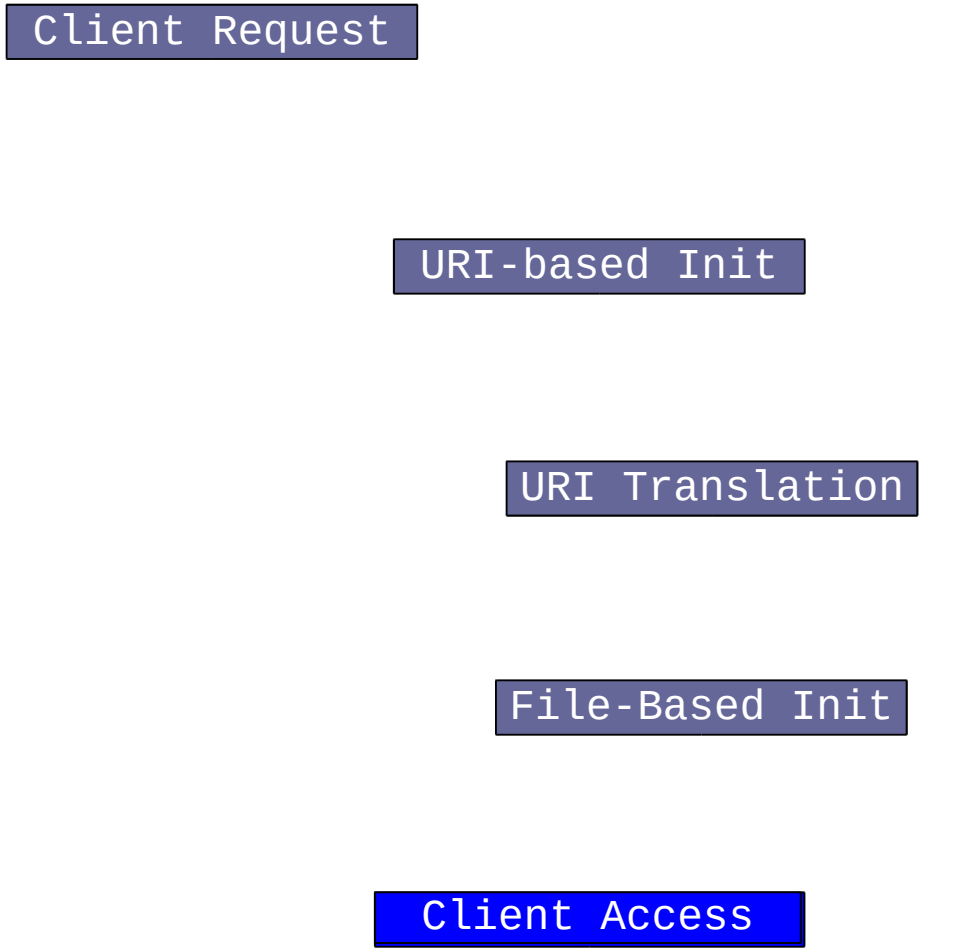
- client requests a document

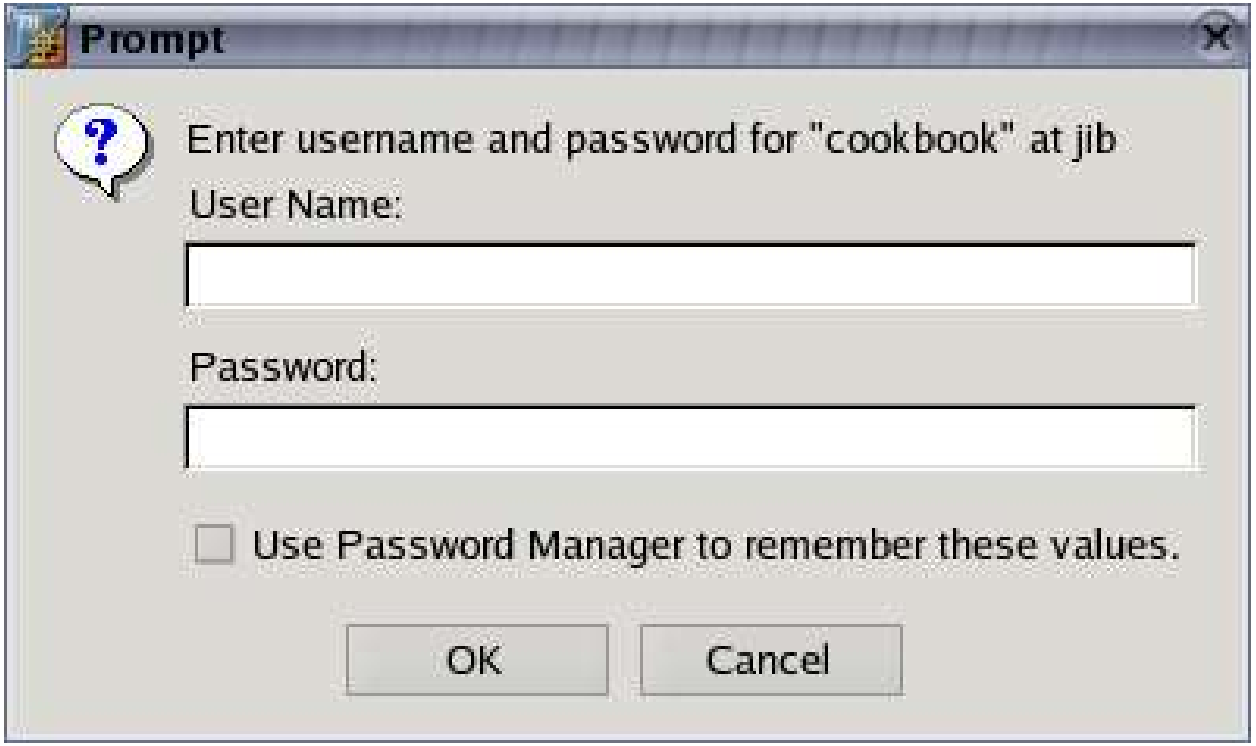
```
GET /perl-status HTTP/1.1
Accept: text/xml, image/png, image/jpeg, image/gif, text/plain
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Language: en-us
Connection: keep-alive
Host: www.example.com
Keep-Alive: 300
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US)
```

- server denies request

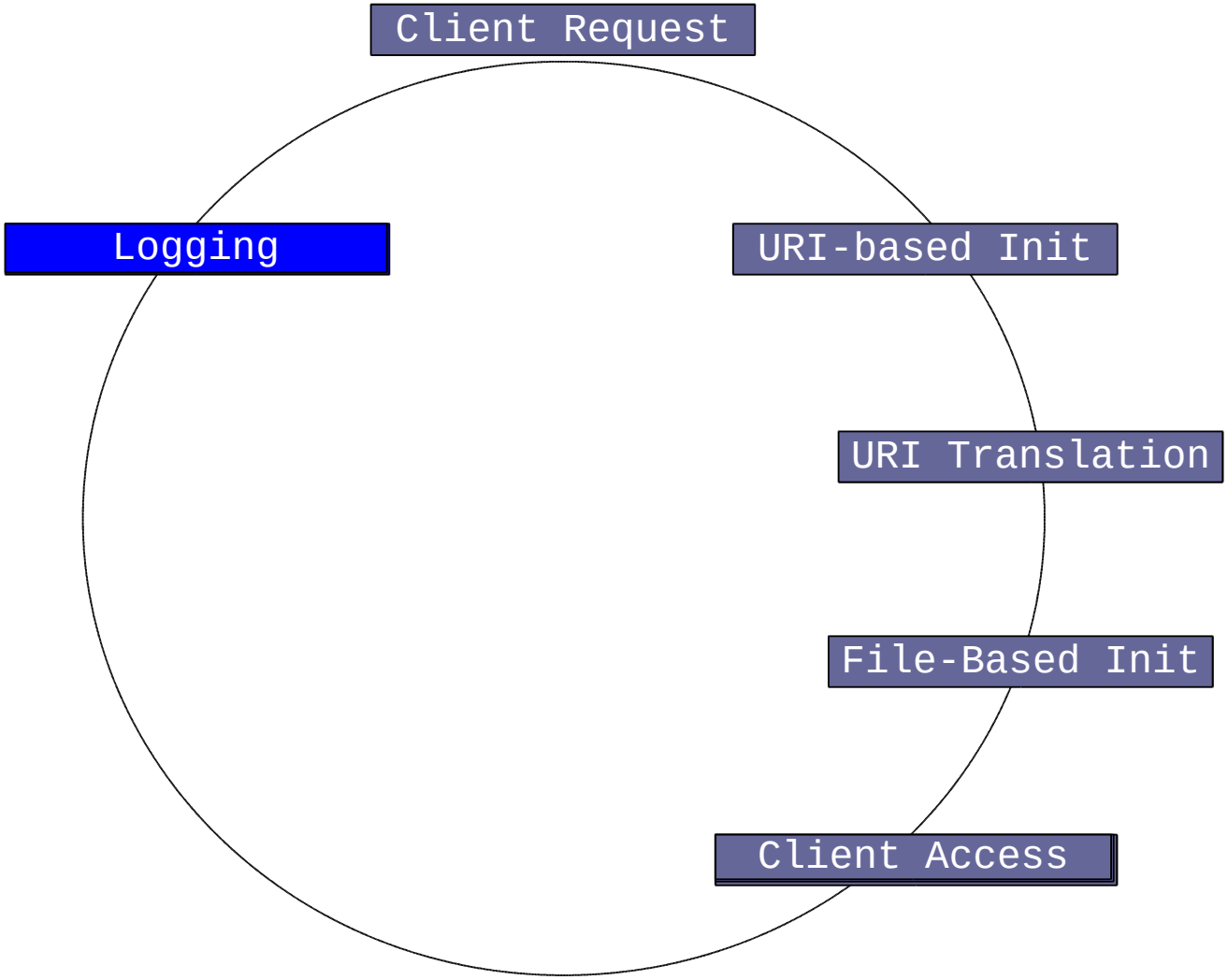
```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="cookbook"
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=iso-8859-1
```

Resource Control




Prompt
Enter username and password for "cookbook" at jib
User Name:
Password:
Use Password Manager to remember these values.
OK
Cancel

Resource Control



How Authentication Works

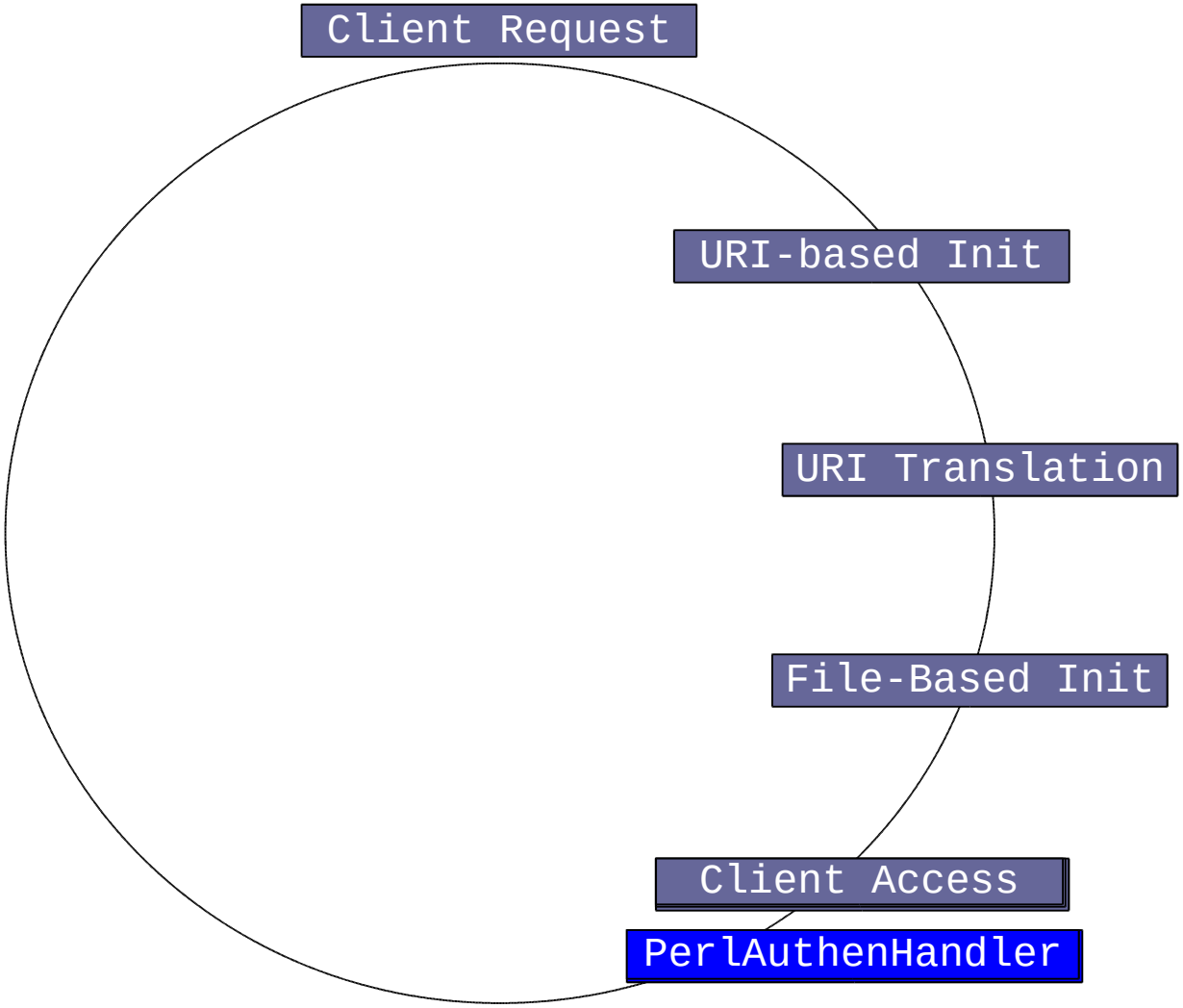
- client sends a new request

```
GET /perl-status HTTP/1.1
Accept: text/xml, image/png, image/jpeg, image/gif, text/plain
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Language: en-us
Authorization: Basic Z2VvZmY6YWZha2VwYXNzd29yZA==
Connection: keep-alive
Host: www.example.com
Keep-Alive: 300
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US)
```

- server sends document

```
HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
```

PerlAuthenHandler



Who Uses Flat Files?

- flat files are limiting, hard to manage, difficult to integrate, and just plain boring
- we can use the Apache API and Perl to replace flat files with our own authentication mechanism

Do it in Perl

- since mod_perl gives us the ability to intercept the request cycle *before* Apache, we can authenticate using Perl instead
- Apache provides an API, making the job easy
- mod_perl provides access to the Apache API

```
package My::Authenticate;

use Apache::Constants qw(OK AUTH_REQUIRED);

use strict;

sub handler {

  my $r = shift;

  # Get the client-supplied credentials.
  my ($status, $password) = $r->get_basic_auth_pw;

  return $status unless $status == OK;

  # Perform some custom user/password validation.
  return OK if authenticate_user($r->user, $password);

  # Whoops, bad credentials.
  $r->note_basic_auth_failure;
  return AUTH_REQUIRED;
}

1;
```

Configuration

- change

```
AuthUserFile .htpasswd
AuthName "cookbook"
AuthType Basic
Require valid-user
```

- to

```
PerlAuthenHandler My::Authenticate
AuthName "cookbook"
AuthType Basic
Require valid-user
```

The Choice is Yours

- how you decide to authenticate is now up to you

```
sub authenticate_user {

  my ($user, $pass) = @_;

  return $user eq $pass;
}
```

- are you seeing the possibilities yet?

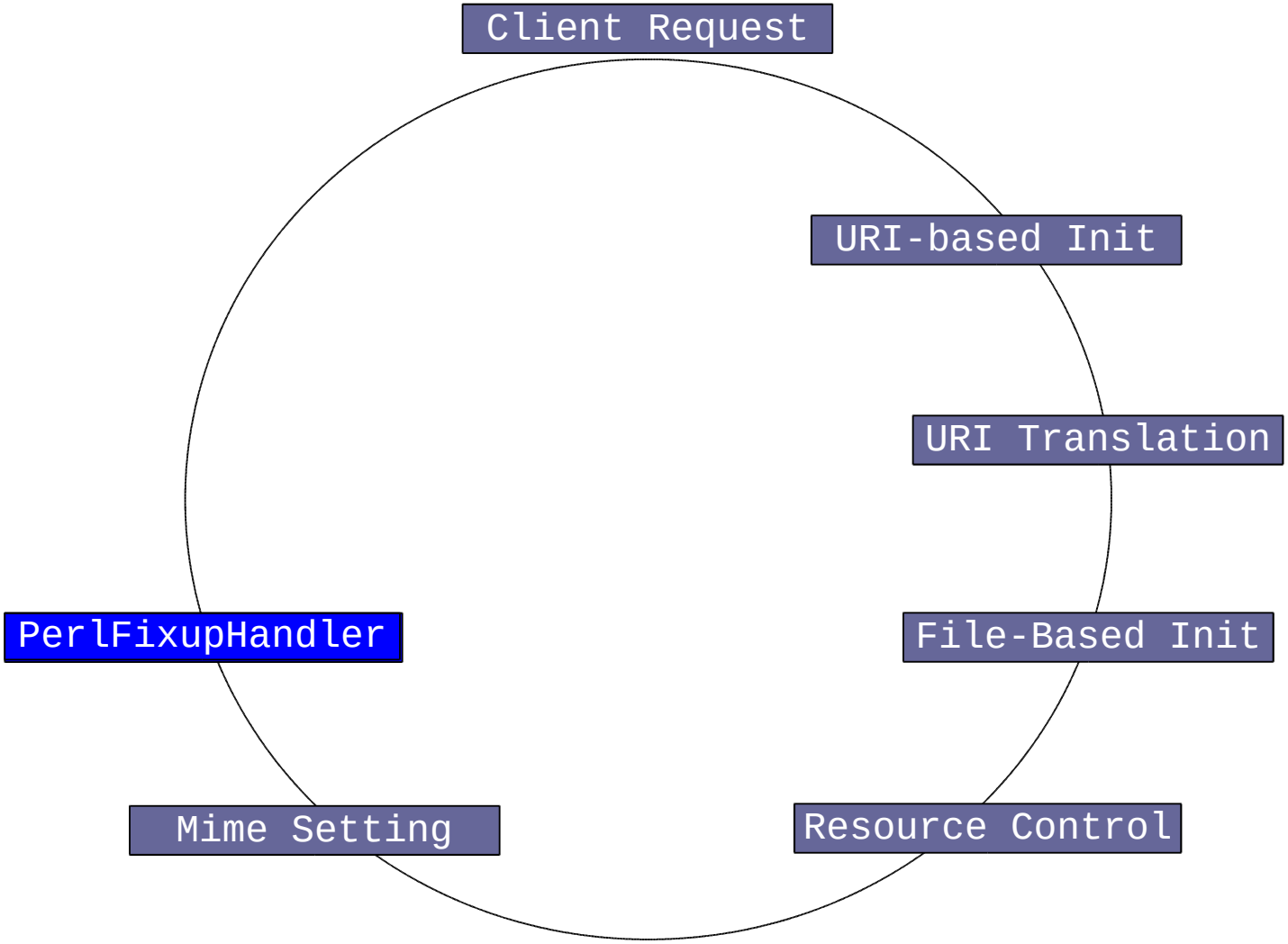
The Power of CPAN

- over 25 Apache:: shrink-wrapped modules on CPAN for authentication
  - SecureID
  - Radius
  - SMB
  - LDAP
  - NTLM

To Infinity and Beyond!

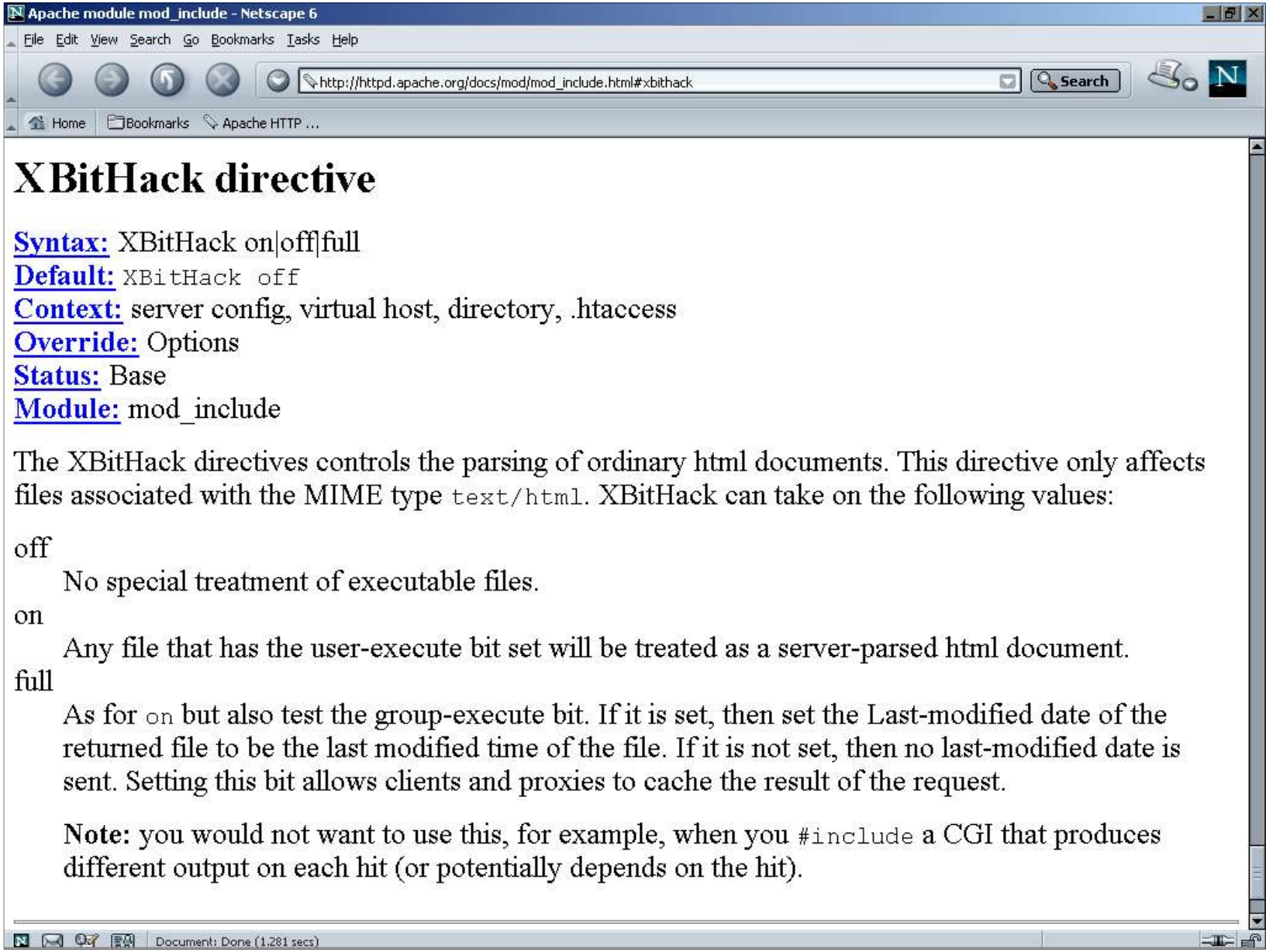
- this example only covered Basic authentication via popup box
- the same techniques can be used to authenticate via a login form plus cookies, munged URLs, or hidden fields
- extended to use Digest authentication as well

PerlFixupHandler



PerlFixupHandler

- Generic phase with no Apache default behavior
- The final chance to fiddle with the request before content is written to the client


Apache module mod_include - Netscape 6
File Edit View Search Go Bookmarks Tasks Help
http://httpd.apache.org/docs/mod/mod_include.html#xbithack
Search
Home
Bookmarks
Apache HTTP ...
XBitHack directive
Syntax: XBitHack on|off|full
Default: XBitHack off
Context: server config, virtual host, directory, .htaccess
Override: Options
Status: Base
Module: mod_include
The XBitHack directives controls the parsing of ordinary html documents. This directive only affects files associated with the MIME type text/html. XBitHack can take on the following values:
off
No special treatment of executable files.
on
Any file that has the user-execute bit set will be treated as a server-parsed html document.
full
As for on but also test the group-execute bit. If it is set, then set the Last-modified date of the returned file to be the last modified time of the file. If it is not set, then no last-modified date is sent. Setting this bit allows clients and proxies to cache the result of the request.
Note: you would not want to use this, for example, when you #include a CGI that produces different output on each hit (or potentially depends on the hit).
Document: Done (1.281 secs)

mod_include's `XBitHack`

- mod_include's `XBitHack` provides an alternate way to turn on the SSI engine
- if the file is html and the user execute bit is on, mod_include parses the file
- if the user and group execute bits are on, it sends a `Last-Modified` header

Our own XBitHack

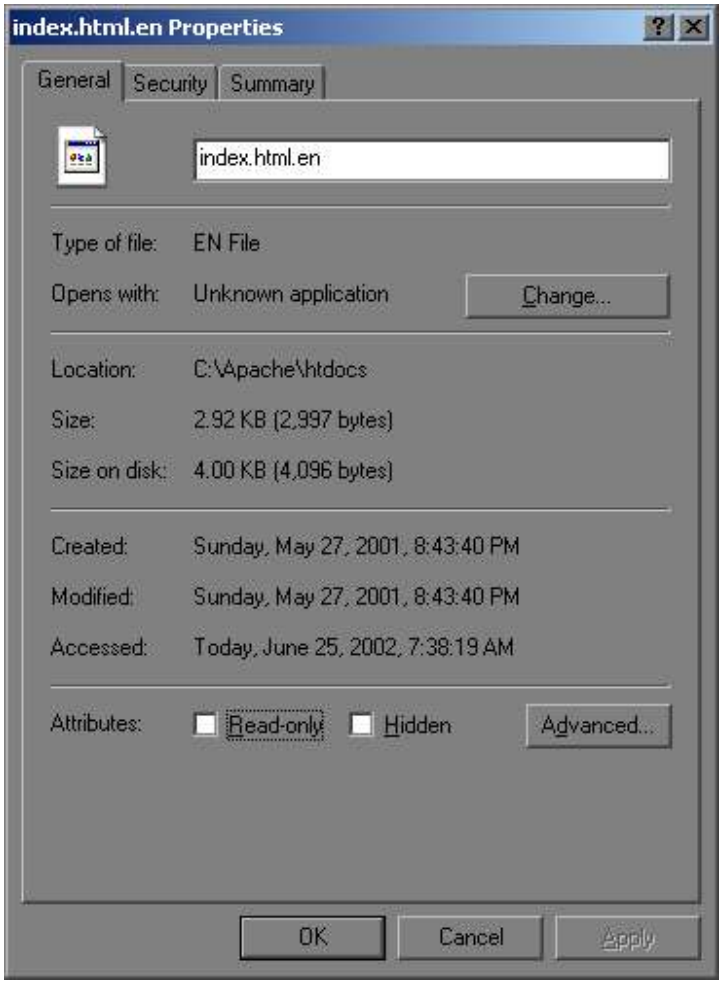
- Let's re-implement XBitHack in our own Perl module
- Make it Apache on Win32 specific

The Win32 Problem

- `XBitHack` uses the concept of "user executable"
- The only "user executable" files on Win32 are `.exe` and `.bat` files
- So, unless you want to call your files `index.bat` or `index.exe` you can't use `XBitHack`
- Behold the power of mod_perl!

```
package Cookbook::WinBitHack;

use Apache::Constants qw(OK DECLINED OPT_INCLUDES);
use Apache::File;

use Win32::File qw(READONLY ARCHIVE);

sub handler {

  my $r = shift;

  return DECLINED unless (
     -f $r->finfo                     &&  # the file exists
     $r->content_type eq 'text/html' &&  # and is HTML
     $r->allow_options & OPT_INCLUDES);  # and we have Options +Includes

  # Gather the file attributes.
  my $attr;
  Win32::File::GetAttributes($r->filename, $attr);

  # Return DECLINED if the file has the ARCHIVE attribute set
  return DECLINED if $attr & ARCHIVE;

  # Set the Last-Modified header unless the READONLY attribute is set.
  $r->set_last_modified((stat _)[9]) unless $attr & READONLY;

  # Make sure mod_include picks it up.
  $r->handler('server-parsed');

  return OK;
}
```

Win32 File Attributes



For the Record

- That was only a partial solution
- I can actually override Apache's `XBitHack` directive at configuration parse time

```
XBitHack Full
```

- But that's a story for another time

Apache Request Cycle