Presentation Slides: mod_perl 2.0, the Next Generation

by **Stas Bekman**
http://stason.org/
<stas@stason.org>
TicketMaster

OSCON US 2003
Wed, July 9 2003
Portland, Oregon, USA

This tutorial is available from: http://stason.org/talks/

Last modified Fri Jul 18 08:57:33 2003 GMT

1 The Next Generation: mod_perl 2.0

1.1 About

- Why rewrite?
- What’s new in Apache 2.0
- What’s new in Perl 5.6.0 - 5.8.0
- What’s new in mod_perl 2.0
- Installing mod_perl 2.0
- Configuring mod_perl 2.0
- Migrating from 1.0 to 2.0

- New Phases
- Protocol Handlers
- Filter Handlers

Thank you:

ticketmaster

for sponsoring my work on mod_perl for the 2nd year!

TicketMaster rules!!!

1.2 Versioning Convention

- To make things simple here and in the new docs:
- mod_perl:
    - mod_perl 1.0 (not mod_perl 1.xx)
    - mod_perl 2.0 (not mod_perl 2.0.xx)
- Apache:
    - Apache 1.3
    - Apache 2.0

1.3 Which mod_perl Version Do I Run

```
% perl -MApache2 -Mmod_perl -le ’print mod_perl->VERSION’
1.9910

% perl           -Mmod_perl -le ’print mod_perl->VERSION’
1.2701
```

1.4 Why the 2.0 Rewrite?

- Too patchy (6 years!), backward compatibility with:
    - Apache 1.3.0 - 1.3.27
    - Perl 5.003 - 5.8.0
- mod_perl 2.0 starts afresh with:
    - Apache 2.0 (incompatible with Apache 1.3)
    - Perl 5.6.1 (has semi-thread-safe Perl Interpreters)
    - Threaded mpms: 5.8.0 (really thread-safe)

- A new build system autogenerates the code used to
    - autogenerates the code that is used to ...
    - ...
    - which generates the final code ...
    - ... and it all works
- Automatically supports new Apache APIs

1.4.1 Apache::Test

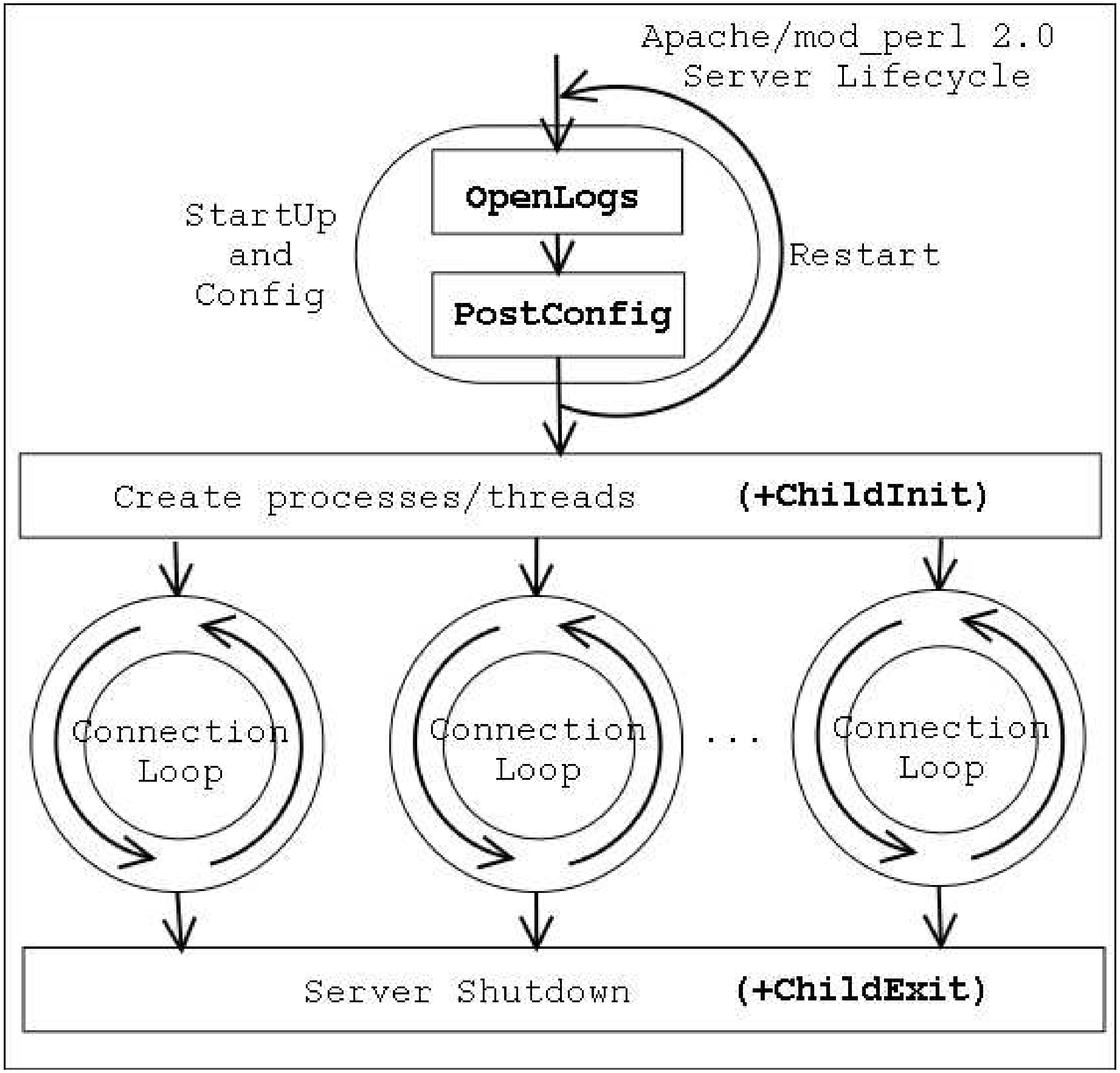
- The mod_perl 2.0 core:

```
make test
All tests successful
Files=126, Tests=653, 166 wallclock secs...                          #

cd ModPerl-Registry && make test
All tests successful.
Files=10, Tests=51, 18 wallclock secs...
```

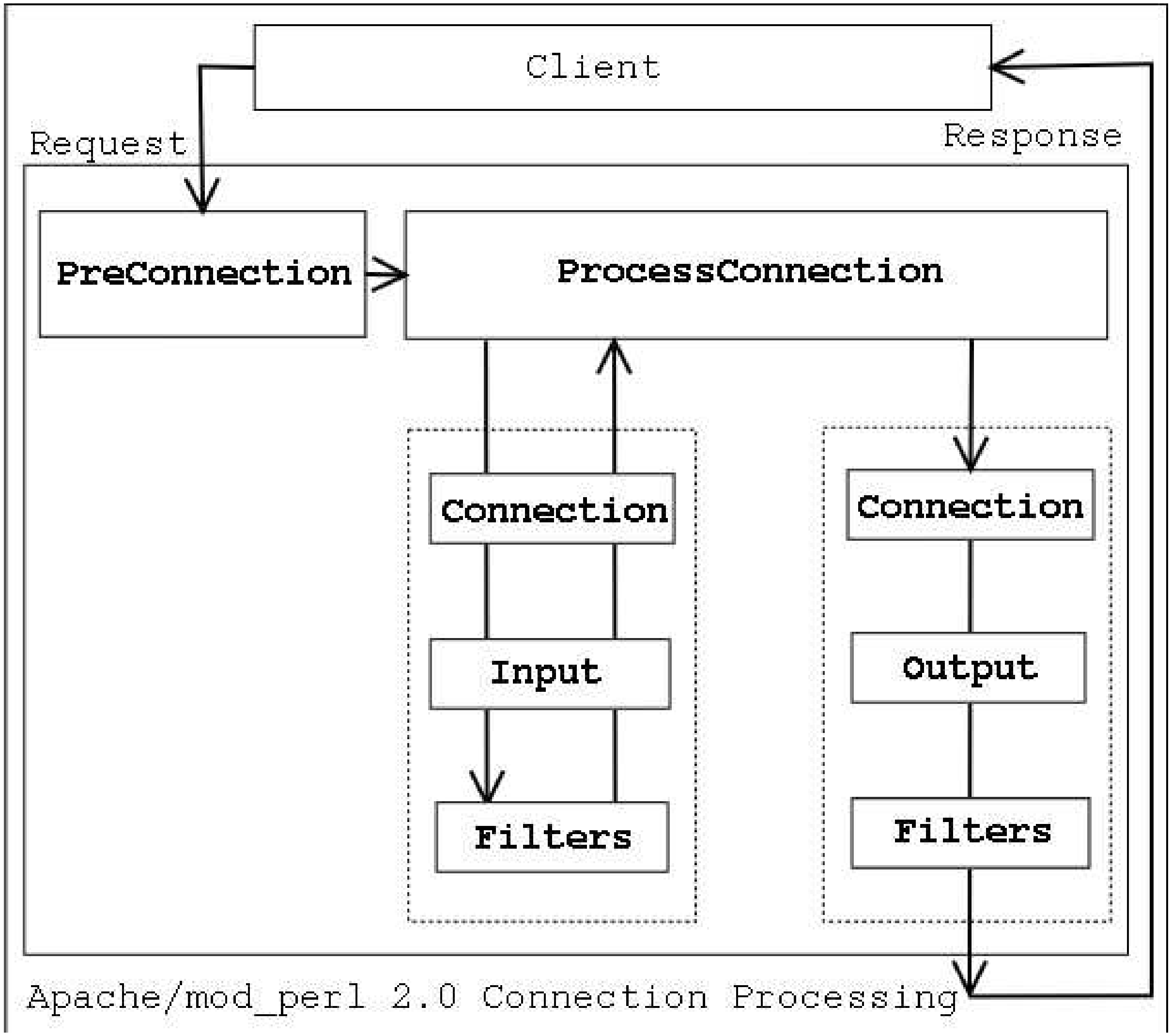
- Any Perl module needing mod_perl 1.0 or 2.0
- Any Apache module (both 1.3 and 2.0), PHP, Python, C...
- Already used by httpd-test to test Apache 1.3 and 2.0!!!

1.5 New in Apache 2.0

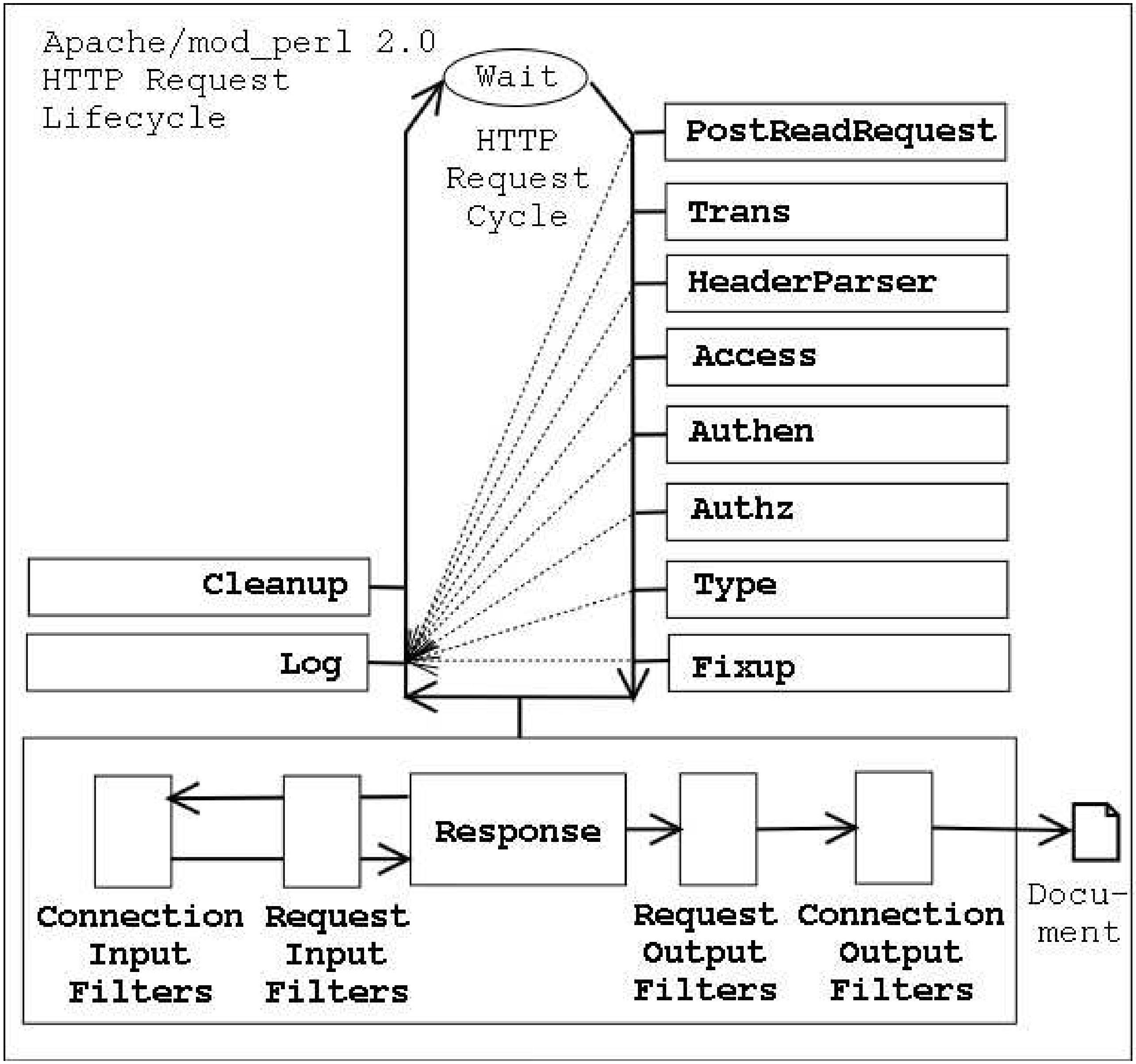
- APR - Apache Portable Runtime
- Multi Processing Model modules (MPMs).
    - processes: prefork
    - threads: worker, leader, perchild...
    - os: mpmt_os2, netware, winnt, beos...


Apache/mod_perl 2.0
Server Lifecycle
StartUp
and
Config
OpenLogs
PostConfig
Restart
Create processes/threads
(+ChildInit)
Connection
Loop
Connection
Loop
...
Connection
Loop
Server Shutdown
(+ChildExit)

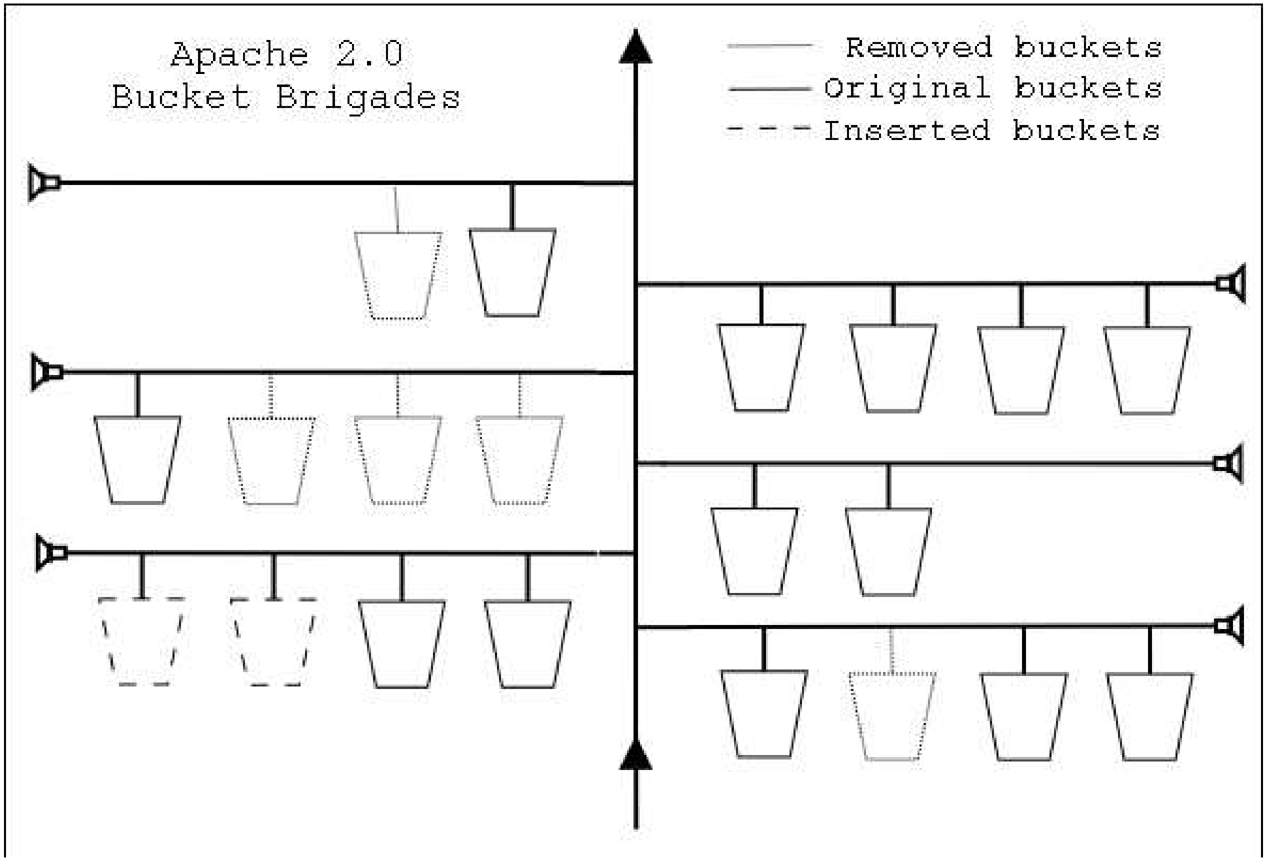
Protocol Modules (HTTP, POP3, SMTP...)



I/O Filtering



Bucket Brigades



New in Apache 2.0:

- Parsed Configuration Tree
- New Hook Scheme (Flexible, Order-able)
- Optional Functions

1.6 New in Perl m/5\.(6|8)\.\d/

- Thread-safe Interpreter (5.8.0) via perl_clone()

- Subroutine attributes:

```
sub handler : FilterRequestHandler { ... }
```

- `CORE::GLOBAL::` subs overriding `CORE::`

- PerlIO layers => `APR::PerlIO`:

```
open my $fh, "<:APR", $file, $r;
```

- I18n: Unicode, UTF...

- Safe signal handling (5.8.0)
- Large file support (files > 2 gigabytes)
- `XSLoader`, a lighter alternative to `DynaLoader`
- fine tuned warnings control:

```
use warnings FATAL => 'all';
```

- Lots of performance enhancements. Though threads slow things down, **if enabled w/o a need**.
- Numerous memory leaks and bugs were fixed

1.7 New in mod_perl 2.0

- All the new Apache 2.0 and Perl 5.6.0+ features
- Plus its own new features

1.7.1 Threads Support

- Thread Interpreters Pool
- scalar @perl_interpreters != scalar @apache_threads
  - no need for front-end/back-end separation

- Two classes of interpreters: *parent* and *clone*
- parent: preload modules and *perl_clone()* clones
- clones: do the real work
  - mutable data is copied by the clone
  - read-only data such as the syntax tree is shared
  - clone pools are FIFO => memory re-use

1.7.2 Thread-safety

- Manipulating Perl data is thread-safe (5.8.0)

```
push(), map(), chomp(), ...
```

- The rest, depends on the underlying implementation

```
localtime(), readdir(), srand(), ...
```

- Thread-safe but Process-scoped

```
chdir(), umask(), chroot(), ...
```

- See perlthrtut(3)

1.7.3 Perl interface to the APR and Apache APIs

- `Apache::` API, which handles issues specific to the web server.
- `APR::` API, which implements a portable and efficient API to handle generically work with files, threads, processes, shared memory, etc.
- `APR::` API, can be used outside of Apache as well.

```
% perl -MApache2 -MAPR -le ’use APR::Table; use APR::Pool; \
 $table = APR::Table::make(APR::Pool->new, 2);’
```

1.7.4 Other New Features

- Protocol modules
- Simplified stream-oriented filtering API
- etc...

1.7.5 Optimizations

- Inlined `{Apache|APR|ModPerl}/*.xs` calls
- Use of Apache Pools for memory allocations

1.8 MPMs and Performance

- Performance may depend on the used mpm
- Nevertheless: CPAN modules should work with any mpm if possible

1.8.1 Memory footprint

prefork mpm:

- same as mod_perl 1.0, relying on shared memory
- The API is spread across many DSO modules
  - loading only the needed modules speeds up the startup
  - saves some memory (not much)
- easy to limit memory usage, `Apache::SizeLimit`, etc.

threaded mpm:

- OS-level memory sharing is not applicable for threads
    - but Perl interpreters share the opcode tree
    - result: memory usage doesn’t grow with time the same way
- FIFO Interpreter pools, optimal usage
    - e.g. can run 2 Perl interpreters to run only trans handler with 256 static threads! no more need for the frontend server.
- Limiting memory usage is under question (GQ thread?)

1.8.2 DataBase Connection Pooling / Apache::DBI

- **prefork** - works as in mod_perl 1.0
- **threaded** - same, plus DBI::Pool will allow sharing across threads of the same process

1.9 Installing mod_perl 2.0 and its prerequisites

- Get the stable sources
  - mod_perl 2.0 from *http://perl.apache.org/dist/*.
  - Apache 2.0 from *http://httpd.apache.org/dist/*.
  - Perl 5.8.0 from *http://cpan.org/src/*.

1.9.1 Installing Apache

```
% cd httpd-2.0.xx
% ./configure --prefix=/home/httpd/httpd-2.0 --with-mpm=prefork
% make && make install
```

1.9.2 Installing Perl

- Perl:

```
% cd perl-5.8.0
% ./Configure -des -Dprefix=$HOME/perl/perl-5.8.0 -Dusethreads
```

- Do not add `-Dusethreads` if you don’t plan on using threads!

```
% make && make test && make install
```

- Prerequisites

```
% $HOME/perl/perl-5.8.0/bin/perl -MCPAN -e ’install("LWP")’
```

1.9.3 Installing mod_perl 2.0

```
% cd mod_perl-2.0.x
% perl Makefile.PL MP_AP_PREFIX=/home/stas/src/httpd-2.0.xx
% make && make test && make install
```

- If `make  test` fails complete a bug report and send it to the mod_perl list. Start with the template:

```
% mp2bug > bug_report.txt
```

```
% t/REPORT > bug_report.txt
```

1.9.4 Binaries

- Apache 2.0 binaries: *http://httpd.apache.org/dist/binaries/*.
- Perl 5.6.1 or 5.8.0 binaries: *http://cpan.org/ports/index.html*.
- mod_perl 2.0 (only win32) by Randy Kobes: http://perl.apache.org/download/binaries.html

1.10 Configuring mod_perl 2.0

- DSO:

```
LoadModule perl_module modules/mod_perl.so
```

- Static: nada

1.10.1 Accessing 2.0 Modules

- mod_perl 2.0 Perl libs go to *Apache2/*
- Adjust `@INC`. Originally:

```
/usr/lib/perl5/site_perl/5.8.0/i686-linux-thread-multi       #
```

- Load *Apache2.pm*:

```
use Apache2 ();
```

- Now `@INC` is:

```
/usr/lib/perl5/site_perl/5.8.0/i686-linux-thread-multi/Apache2
/usr/lib/perl5/site_perl/5.8.0/i686-linux-thread-multi
```

1.10.2 PerlRequire’d Startup File

```
use Apache2 ();
# use Apache::compat (); # 1.0 compat                                  #

# preload all mp2 modules
# use ModPerl::MethodLookup;
# ModPerl::MethodLookup::preload_all_modules();

use lib qw(/home/httpd/perl);

use ModPerl::Util (); #for CORE::GLOBAL::exit

use Apache::RequestRec ();
use Apache::RequestIO ();
use Apache::RequestUtil ();

use Apache::Server ();
use Apache::ServerUtil ();
use Apache::Connection ();
use Apache::Log ();

use APR::Table ();
use ModPerl::Registry ();

use Apache::Const -compile => ':common';
use APR::Const    -compile => ':common';

1;
```

1.10.3 Perl's Command Line Switches

- `PerlSwitches` passes any Perl switches
- e.g., enable warnings and taint checking:

```
PerlSwitches -wT
```

- adjust `@INC` values:

```
PerlSwitches -I/home/stas/modperl
```

1.10.4 mod_perl 2.0 Core Handlers

```
<Location /perl>
  SetHandler perl-script
  PerlResponseHandler ModPerl::Registry
</Location>

<Location /perl2>
  SetHandler modperl
  PerlResponseHandler ModPerl::Registry
</Location>
```

1.10.5 "perl-script"

```
SetHandler perl-script
```

- As in mod_perl 1.0
- Unless set to `-GlobalRequest` assumes

```
PerlOptions +GlobalRequest
```

- Unless set to `-SetupEnv` assumes

```
PerlOptions +SetupEnv
```

- Tied `STDIN` and `STDOUT`

```
my $line = <STDIN>;
print "Dahuuuuut!";
```

- on each request restores `%ENV`, `@INC`, `$/`, `STDOUT`'s `$|` and `END` blocks

1.10.6 "modperl"

```
SetHandler modperl
```

- calls the `Perl*Handler`'s callback func.
- sets `MOD_PERL`, `GATEWAY_INTERFACE`, `PATH` and `TZ` env vars.
- No tied IO handles:

```
$r->read($line, $len, $offset);
$r->print("Dahuuuuut!");
```

1.10.6.1 A Simple Response Handler Example

- Printout environment variables ala `perl-script` core handler

```
PerlModule MyApache::PrintEnv1
<Location /print_env1>
    SetHandler perl-script
    PerlResponseHandler MyApache::PrintEnv1
</Location>
```

```
package MyApache::PrintEnv1;                                                  #

use strict;
use warnings;

use Apache::RequestRec (); # for $r->content_type
use Apache::RequestIO ();  # for print
use Apache::Const -compile => ’OK’;

sub handler {
    my $r = shift;

    $r->content_type(’text/plain’);
    for (sort keys %ENV){
        print "$_ => $ENV{$_}\n";
    }

    return Apache::OK;
}
1;
```

- Printout environment variables ala `modperl` core handler

```
PerlModule MyApache::PrintEnv2
<Location /print_env2>
    SetHandler modperl
    PerlResponseHandler MyApache::PrintEnv2
</Location>
```

```
package MyApache::PrintEnv2;                                       #

use strict;
use warnings;

use Apache::RequestRec (); # for $r->content_type
use Apache::RequestIO ();  # for $r->print
use Apache::Const -compile => ’OK’;

sub handler {
    my $r = shift;

    $r->content_type(’text/plain’);
    $r->subprocess_env;
    for (sort keys %ENV){
        $r->print("$_ => $ENV{$_}\n");
    }
    return Apache::OK;
}
1;
```

- Instead of calling:

```
$r->subprocess_env;
```

- could configure the location with:

```
PerlOptions +SetupEnv
```

1.10.7 `PerlOptions` Directive

- Disable mod_perl for a given `VirtualHost`:

```
<VirtualHost ...>
    PerlOptions -Enable
</VirtualHost>
```

- Give the `VirtualHost` its own interpreter pool.

```
<VirtualHost ...>
    PerlOptions +Clone
    PerlInterpStart 2
    PerlInterpMax 2
</VirtualHost>
```

- Run different versions of the same module:

```
<VirtualHost ...>
    ServerName dev1
    PerlOptions +Parent
    PerlSwitches -Mlib=/home/dev1/lib/perl
</VirtualHost>

<VirtualHost ...>
    ServerName dev2
    PerlOptions +Parent
    PerlSwitches -Mlib=/home/dev2/lib/perl
</VirtualHost>
```

- disallow certain handlers/options

```
<VirtualHost ...>
    PerlOptions -Authen -Authz -Access -Sections
</VirtualHost>
```

- Or maybe everything but the response handler:

```
<VirtualHost ...>
    PerlOptions None +Response
</VirtualHost>
```

- Resolve `Perl*Handlers` at startup time:

```
PerlOptions +Autoload
PerlResponseHandler MyApache::Magick
```

- Disable the global `request_rec` (`Apache->request`)

```
<Location ...>
    SetHandler perl-script
    PerlOptions -GlobalRequest
    ...
</Location>
```

- s/PerlSendHeader On/PerlOptions +ParseHeaders/
- Merge Handlers

```
PerlFixupHandler MyApache::FixupA
<Location /inside>
    PerlOptions +MergeHandlers
    PerlFixupHandler MyApache::FixupB
</Location>
```

- s/PerlSetupEnv Off/PerlOptions -SetupEnv/

1.10.8 Threads Mode Specific Directives

- `PerlInterpStart`
- `PerlInterpMax`
- `PerlInterpMinSpare`
- `PerlInterpMaxSpare`
- `PerlInterpMaxRequests`

`PerlInterpScope`:

- Use for the lifetime of the request (default):

```
PerlInterpScope request
```

- Use a separate interpreter in subrequests:

```
PerlInterpScope subrequest
```

- Use a separate interpreter for each handler:

```
PerlInterpScope handler
```

1.10.9 Retrieving Server Startup Options

```
% httpd -DONE_PROCESS
```

- Retrieve:

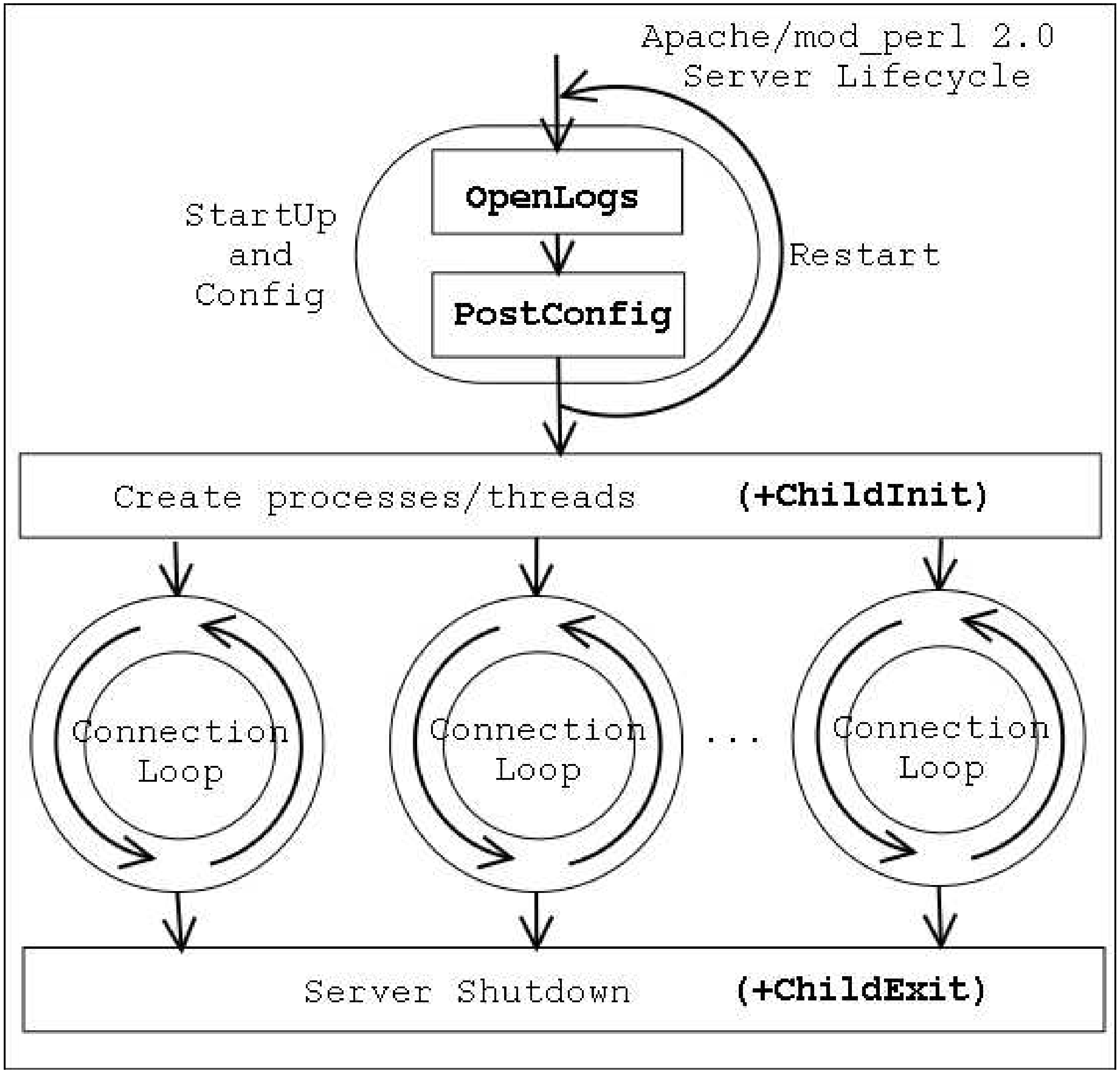
```
if (Apache::exists_config_define("ONE_PROCESS")) {
    print "Running in a single mode";
}
```

1.11 New Apache Phases

- Several new phases were added in Apache 2.0

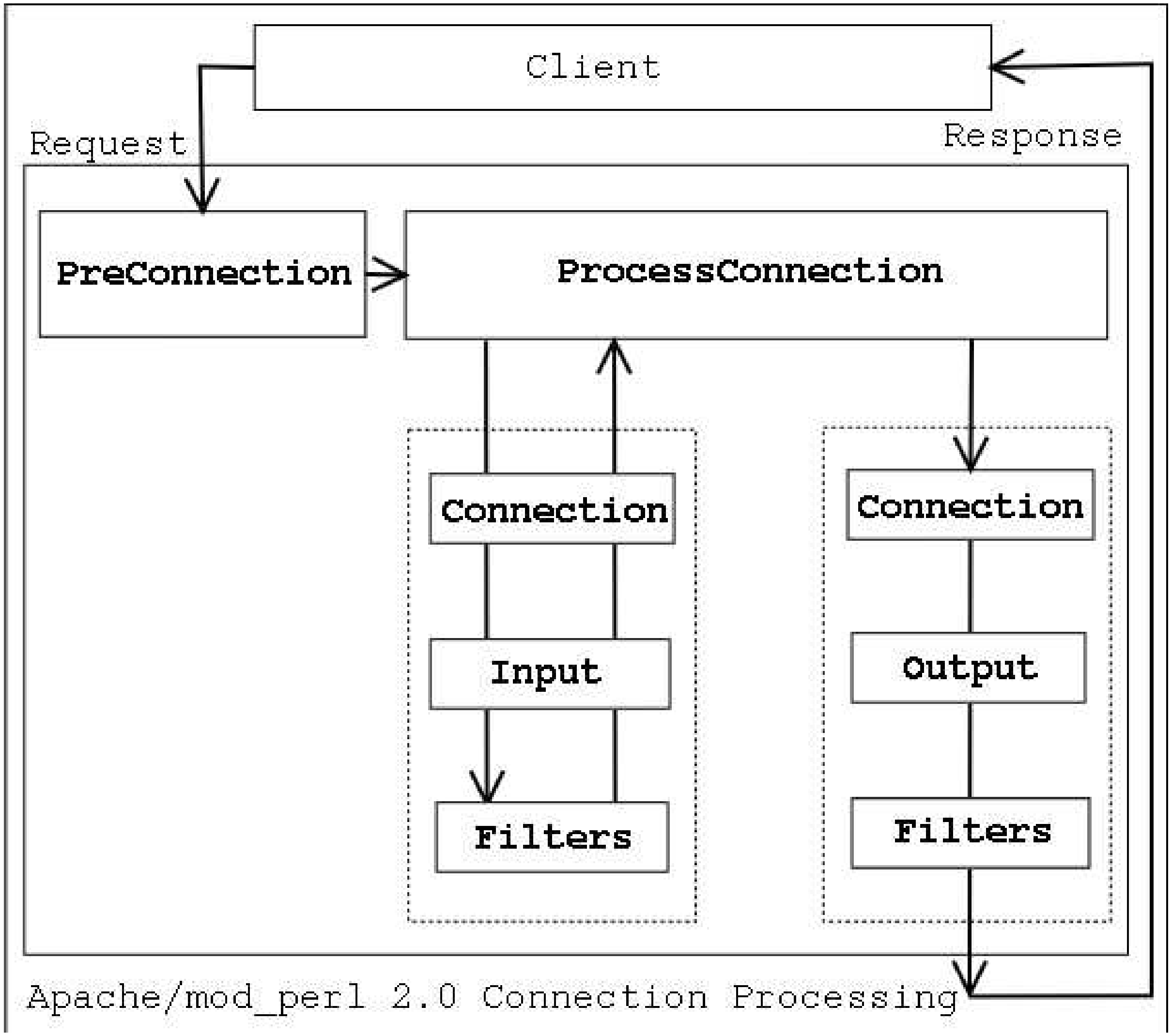
1.11.1 Server Configuration (Startup) Phases

- `PerlOpenLogsHandler`
- `PerlPostConfigHandler`


Apache/mod_perl 2.0
Server Lifecycle
StartUp
and
Config
OpenLogs
PostConfig
Restart
Create processes/threads
(+ChildInit)
Connection
Loop
Connection
Loop
...
Connection
Loop
Server Shutdown
(+ChildExit)

1.11.2 Connection Phases

- `PerlPreConnectionHandler`
- `PerlProcessConnectionHandler`


Client
Request
Response
PreConnection
ProcessConnection
Connection
Input
Filters
Connection
Output
Filters
Apache/mod_perl 2.0 Connection Processing

1.11.2.1 `PerlPreConnectionHandler`

- The *pre_connection* phase happens just after the server accepts the connection, but before it is handed off to a protocol module to be served.
- It gives modules an opportunity to modify the connection as soon as possible and insert filters if needed.
- The core server uses this phase to setup the connection record based on the type of connection that is being used.
- mod_perl itself uses this phase to register the connection input and output filters.

- Can run `Apache::Reload` at this phase for non-HTTP protocol modules

Block by IP example:

```
package MyApache::BlockIP2;                                          #

use Apache::Connection ();

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

my %bad_ips = map {$_ => 1} qw(127.0.0.1 10.0.0.4);

sub handler {
    my Apache::Connection $c = shift;

    my $ip = $c->remote_ip;
    if (exists $bad_ips{$ip}) {
        warn "IP $ip is blocked\n";
        return Apache::FORBIDDEN;
    }

    return Apache::OK;
}
1;
```

- Configuration

```
PerlPreConnectionHandler MyApache::BlockIP2
```

- Apache simply drops the connection if the IP is blacklisted
- Almost no resources are wasted

1.11.2.2 PerlProcessConnectionHandler

- The *process_connection* phase is used to process incoming connections.
- Only protocol modules should assign handlers for this phase, as it gives them an opportunity to replace the standard HTTP processing with processing for some other protocols (e.g., POP3, FTP, etc.).
- Protocol modules can work:
    - directly with a socket, filters don’t work
    - manipulating bucket brigades, filters work

1.11.2.2.1 MyApache::Eliza Protocol Module

- An example of a protocol working directly with a socket
- Very simple implementation, but cannot use filters

- Configuration:

```
Listen 8084
<VirtualHost _default_:8084>
    PerlModule MyApache::Eliza
    PerlProcessConnectionHandler MyApache::Eliza
</VirtualHost>
```

```
package MyApache::Eliza;                                            #

use strict;
use warnings FATAL => ’all’;

use Apache::Connection ();
use APR::Socket ();

require Chatbot::Eliza;

use Apache::Const -compile => ’OK’;

use constant BUFF_LEN => 1024;

my $eliza = new Chatbot::Eliza;
```

```
sub handler {                                                      #
    my Apache::Connection $c = shift;
    my APR::Socket $socket = $c->client_socket;

    my $buff;
    my $last = 0;
    while (1) {
        my($rlen, $wlen);
        $rlen = BUFF_LEN;
        $socket->recv($buff, $rlen);
        last if $rlen <= 0;

        # \r is sent instead of \n if the client is talking over telnet
        $buff =~ s/[\r\n]*$//;
        $last++ if $buff =~ /good bye/i;
        $buff = $eliza->transform( $buff ) . "\n\n";
        $socket->send($buff, length $buff);
        last if $last;
    }

    Apache::OK;
}
1;
```

- And we give it a whirl:

```
% telnet localhost 8084                                           #
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is ’^]’.
Hello Eliza
How do you do. Please state your problem.

Well, emails take over my work day.
Is it important to you that your work day?

Yes, because I want to have a life too!
Why don’t you tell me a little more about this.

I’m not sure you know what life is
We were discussing you -- not me.

That’s true. But it’s because of programs like you we programmers have no life.
You’re not really talking about me -- are you?

Yes I’m talking about you!
Please go on.

Sorry, Eliza, my day is over again, have to leave now. good bye.
Your day is over again?

Connection closed by foreign host.
```

1.11.2.2.2 MyApache::Eliza2 Protocol Module

- An example of a protocol working directly with bucket brigades
- More complicated implementation, but can use filters
- Here we use a lowercase output filter that lowers the case of the response sent by Eliza.

```
package MyApache::Eliza2;                                           #

use strict;
use warnings FATAL => 'all';

use Apache::Connection ();
use APR::Bucket ();
use APR::Brigade ();
use APR::Util ();

require Chatbot::Eliza;

use APR::Const -compile => qw(SUCCESS EOF);
use Apache::Const -compile => qw(OK MODE_GETLINE);

my $eliza = new Chatbot::Eliza;
```

```
sub handler {                                                     #
    my Apache::Connection $c = shift;

    my $bb_in  = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $bb_out = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $last = 0;

    while (1) {
        my $rv = $c->input_filters->get_brigade($bb_in,
                                                Apache::MODE_GETLINE);

        if ($rv != APR::SUCCESS or $bb_in->empty) {
            my $error = APR::strerror($rv);
            unless ($rv == APR::EOF) {
                warn "[eliza] get_brigade: $error\n";
            }
            $bb_in->destroy;
            last;
        }
```

```
while (!$bb_in->empty) {                                    #
    my $bucket = $bb_in->first;
    $bucket->remove;

    if ($bucket->is_eos) {
        $bb_out->insert_tail($bucket);
        last;
    }

    my $data;
    my $status = $bucket->read($data);
    return $status unless $status == APR::SUCCESS;

    if ($data) {
        $data =~ s/[\r\n]*$//;
        $last++ if $data =~ /good bye/i;
        $data = $eliza->transform( $data ) . "\n\n";
        $bucket = APR::Bucket->new($data);
    }

    $bb_out->insert_tail($bucket);
}

my $b = APR::Bucket::flush_create($c->bucket_alloc);
$bb_out->insert_tail($b);
```

```
          $c->output_filters->pass_brigade($bb_out);
          last if $last;
      }

      Apache::OK;
  }
```

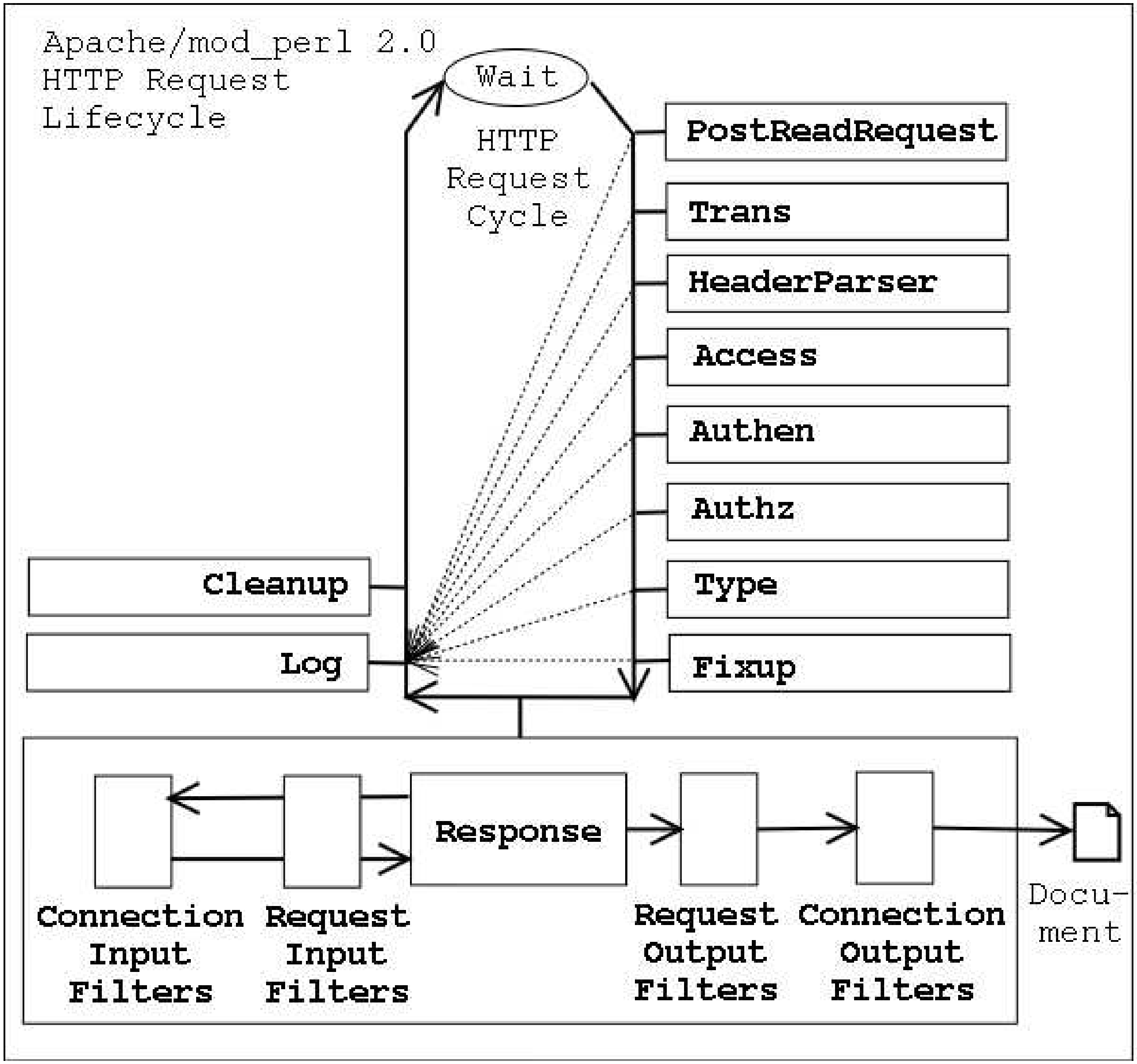
```
use base qw(Apache::Filter);                                #
use constant BUFF_LEN => 1024;

sub lowercase : FilterConnectionHandler {
    my $filter = shift;

    while ($filter->read(my $buffer, BUFF_LEN)) {
        $filter->print(lc $buffer);
    }

    return Apache::OK;
}

1;
```

- Configuration

```
Listen 8085
<VirtualHost _default_:8085>
    PerlModule MyApache::Eliza2
    PerlProcessConnectionHandler MyApache::Eliza2
    PerlOutputFilterHandler MyApache::Eliza2::lowercase
</VirtualHost>
```

1.11.3 Request Phases

- `PerlResponseHandler`



Apache/mod_perl 2.0
HTTP Request
Lifecycle
Wait
HTTP
Request
Cycle
PostReadRequest
Trans
HeaderParser
Access
Authen
Authz
Type
Fixup
Cleanup
Log
Response
Connection Input Filters
Request Input Filters
Request Output Filters
Connection Output Filters
Docu-
ment

1.11.4 I/O Filtering Phases

- `PerlInputFilterHandler`
- `PerlOutputFilterHandler`
- mod_perl provides two interfaces to filtering:
  - a direct mapping to buckets and bucket brigades
  - and a simpler, stream-oriented interface

- mod_perl can do connection and request filtering.
- (Apache supports several other types)
- subroutine attributes set the filter type

```
sub handler : FilterRequestHandler    { ... }
sub handler : FilterConnectionHandler { ... }
```

1.11.4.1 `PerlInputFilterHandler`

- poor man’s s/GET/HEAD/ rewrite
- The handler looks for data like:

```
GET /perl/test.pl HTTP/1.1
```

- and turns it into:

```
HEAD /perl/test.pl HTTP/1.1
```

```
package MyApache::InputFilterGET2HEAD;                            #

use strict;
use warnings;

use base qw(Apache::Filter);

use Apache::RequestRec ();
use Apache::RequestIO ();
use APR::Brigade ();
use APR::Bucket ();

use Apache::Const -compile => ’OK’;
use APR::Const -compile => ’:common’;
```

```
sub handler : FilterConnectionHandler {                           #
    my($filter, $bb, $mode, $block, $readbytes) = @_;

    my $c = $filter->c;
    my $ctx_bb = APR::Brigade->new($c->pool, $c->bucket_alloc);
    my $rv = $filter->next->get_brigade($ctx_bb, $mode, $block, $readbytes);
    return $rv unless $rv == APR::SUCCESS;

    while (!$ctx_bb->empty) {
        my $bucket = $ctx_bb->first;
        $bucket->remove;

        if ($bucket->is_eos) {
            $bb->insert_tail($bucket);
            last;
        }

        my $data;
        my $status = $bucket->read($data);
        return $status unless $status == APR::SUCCESS;

        if ($data and $data =~ s|^GET|HEAD|) {
            $bucket = APR::Bucket->new($data);
        }

        $bb->insert_tail($bucket);
    }
```

```
      Apache::OK;
  }
  1;
```

```
Listen 8005
<VirtualHost _default_:8005>
    PerlInputFilterHandler +MyApache::InputFilterGET2HEAD

    <Location />
        SetHandler modperl
        PerlResponseHandler +MyApache::RequestType
    </Location>
</VirtualHost>
```

1.11.4.2 `PerlOutputFilterHandler`

- A stream oriented output filter
- `MyApache::ROT13` implements the simple Caesar-cypher encryption
- so that *"mod_perl 2.0 rules!"* becomes *"zbq_crey 2.0 ehyrf!"*

```
% perl -le ’print length join "", "a" .. "z"’
26
```

- therefore the ROT13 encryption is self-inverse,
- so the same code can be used for encoding and decoding

```
package MyApache::ROT13;                                                  #
use strict;

use Apache::RequestRec ();
use Apache::RequestIO ();
use Apache::Filter ();

use Apache::Const -compile => ’OK’;

use constant BUFF_LEN => 1024;

sub handler {
    my $filter = shift;

    while ($filter->read(my $buffer, BUFF_LEN)) {
        $buffer =~ y/A-Za-z/N-ZA-Mn-za-m/;
        $filter->print($buffer);
    }

    return Apache::OK;
}
1;
```

Configuration:

```
PerlModule MyApache::ROT13
Alias /perl-rot13/ /home/httpd/perl/
<Location /perl-rot13>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    PerlOutputFilterHandler MyApache::ROT13
    Options +ExecCGI
    #PerlOptions +ParseHeaders
</Location>
```

1.12 Migrating from mod_perl 1.0 to mod_perl 2.0

- Several configuration directives were renamed or removed.
- Several APIs have changed, renamed, removed, or moved to new packages.
- Certain functions while staying exactly the same as in mod_perl 1.0, now reside in different packages.

1.12.1 The Shortest Migration Path from 1.0

```
use Apache2;
use Apache::compat;
```

- Apache2.pm helps to have 1.0 and 2.0 code coexist by installing modules with the same name into different dirs.
- Certain Configuration directives and APIs have changed

`http://perl.apache.org/docs/2.0/user/compat/compat.html`

`http://perl.apache.org/docs/2.0/user/compat/porting.html`

1.12.2 Migrating Configuration Files

```
PerlHandler          => PerlResponseHandler                          #

PerlSendHeader On  => PerlOptions +ParseHeaders
PerlSendHeader Off => PerlOptions -ParseHeaders

PerlSetupEnv On    => PerlOptions +SetupEnv
PerlSetupEnv Off   => PerlOptions -SetupEnv

PerlTaintCheck     => PerlSwitches -T
PerlWarn           => PerlSwitches -w

PerlFreshRestart   => /dev/null (gone)
```

1.12.3 `ModPerl::Registry` Family

- s/Apache::Registry/ModPerl::Registry/

```
Alias /perl/ /home/httpd/perl/
<Location /perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    Options +ExecCGI
    PerlOptions +ParseHeaders
</Location>
```

- Cook your own registry with `Apache::RegistryCooker`

1.12.4 Method Handlers

- The `($$)` prototyping doesn’t work since some callbacks accepts more than 2 args

```
package Bird;
@ISA = qw(Eagle);

sub handler : method {
    my($class, $r) = @_;
    ...;
}
```

- See the *attributes* manpage.

1.12.5 `Apache::StatINC` Replacement

- `Apache::StatINC` has been replaced by `Apache::Reload`, which works for both mod_perl generations and provides extra functionality
- To migrate simply replace:

```
PerlInitHandler Apache::StatINC
```

- with:

```
PerlInitHandler Apache::Reload
```

1.12.6 ModPerl::MethodLookup

- Map methods to their modules

```
# error_log
Can’t locate object method "sendfile" via package "Apache::RequestRec"

% perl -MApache2 -MModPerl::MethodLookup -e print_method sendfile
to use method ’sendfile’ add:
        use Apache::RequestIO ();

% alias lookup "perl -MApache2 -MModPerl::MethodLookup -e print_method"
% lookup sendfile
  to use method ’sendfile’ add:
        use Apache::RequestIO ();
```

- Map modules to their methods

```
% perl -MApache2 -MModPerl::MethodLookup -e print_module Apache::RequestIO

Module ’Apache::RequestIO’ contains the following XS methods:

Method                Invoked on object type
--------------------------------------------------------------------------
BINMODE               Apache::RequestRec
CLOSE                 Apache::RequestRec
FILENO                Apache::RequestRec
GETC                  Apache::RequestRec
OPEN                  Apache::RequestRec
PRINT                 Apache::RequestRec
[...]
discard_request_body  Apache::RequestRec
get_client_block      Apache::RequestRec
print                 Apache::RequestRec
printf                Apache::RequestRec
puts                  Apache::RequestRec
read                  Apache::RequestRec
rflush                Apache::RequestRec
sendfile              Apache::RequestRec
setup_client_block    Apache::RequestRec
should_client_block   Apache::RequestRec
write                 Apache::RequestRec
```

- Map objects to their methods

```
% perl -MApache2 -MModPerl::MethodLookup -e print_object APR::Table

Objects of type 'APR::Table' can invoke the following XS methods:

Method   Module
--------------------------------------------------------------------------
CLEAR    APR::Table
DELETE   APR::Table
EXISTS   APR::Table
FETCH    APR::Table
FIRSTKEY APR::Table
NEXTKEY  APR::Table
STORE    APR::Table
add      APR::Table
clear    APR::Table
copy     APR::Table
do       APR::Table
make     APR::Table
merge    APR::Table
overlap  APR::Table
overlay  APR::Table
set      APR::Table
unset    APR::Table
```

1.12.7 How `Apache::MP3` was Ported to mod_perl 2.0

- I started porting `Apache::MP3` version 3.03
  http://search.cpan.org/CPAN/authors/id/L/LD/LDS/Apache-MP3-3.03.tar.gz

Preparations: *httpd.conf*

- empty to a bare minimum, then add a few things:
  - `Apache::Reload` is a must
  - add the `Apache::MP3` config
  - enable warnings/ taint mode

```
Listen 127.0.0.1:8002
#... standard Apache configuration bits omitted ...                      #

LoadModule perl_module modules/mod_perl.so

PerlSwitches -wT
PerlRequire "/home/httpd/2.0/perl/startup.pl"

PerlModule Apache::Reload
PerlInitHandler Apache::Reload
PerlSetVar ReloadAll Off
PerlSetVar ReloadModules "ModPerl::* Apache::*"
PerlSetVar ReloadConstantRedefineWarnings Off

AddType audio/mpeg     mp3 MP3
AddType audio/playlist m3u M3U
AddType audio/x-scpls  pls PLS
AddType application/x-ogg ogg OGG
<Location /mp3>
  SetHandler perl-script
  PerlResponseHandler Apache::MP3
  PerlSetVar PlaylistImage playlist.gif
  PerlSetVar StreamBase http://localhost:8002
  PerlSetVar BaseDir /mp3
</Location>
```

Preparations: *startup.pl*

- keep only the bare minimum

```
use Apache2 ();
use lib qw(/home/httpd/2.0/perl);
use Apache::compat;
```

Preparations: enable warnings in *Apache/MP3.pm*

```
--- Apache/MP3.pm.orig 2003-06-03 18:44:21.000000000 +1000
+++ Apache/MP3.pm      2003-06-03 18:44:47.000000000 +1000
@@ -4,2 +4,5 @@
 use strict;
+use warnings;
+no warnings ’redefine’;  # XXX: remove when done with porting
+
```

Preparations: command-line client and *error_log*

- Console 1: command-line client

```
% lynx --dump http://localhost:8002/mp3/
```

- Console 2: keeping the *error_log* open

```
% err2
```

- which expands to:

```
% tail -f ~/httpd/prefork/logs/error_log
```

Porting: method handlers

```
[Thu Jun 05 15:29:45 2003] [error] [client 127.0.0.1]
Usage: Apache::RequestRec::new(classname, c, base_pool=NULL)
at .../Apache/MP3.pm line 60.
```

- Looking at the code:

```
58: sub handler ($$) {
59:   my $class = shift;
60:   my $obj = $class->new(@_) or die "Can’t create object: $!";
```

- referring to the 2.0 migration reference, see that methods declaration is different

```
-sub handler ($$) {
+sub handler : method {
```

Porting: bug in Apache

- as of 2.0.46 `r->path_info` return ” all the time when the location maps to an existing directory, which is how `Apache::MP3` works.

```
 -  unless ($self->r->path_info){
 +  unless ($self->r->path_info eq ’’){
```

Porting: Fixing warnings

- Since I have enabled warnings, I had to take care of many warnings. e.g. abstracting the dir_config() usage, which returns undef when not set:

```
sub get_config {
    my $val = shift->r->dir_config(shift);
    return defined $val ? $val : ’’;
}

sub config_yes { shift->get_config(shift) !~ /$YES/oi; }
sub config_no  { shift->get_config(shift) !~ /$NO/oi; }
```

Porting: API change

- When trying to stream a song I get:

```
[Fri Jun 06 15:33:33 2003] [error] [client 127.0.0.1] Bad arg length
for Socket::unpack_sockaddr_in, length is 31, should be 16 at
.../5.9.0/i686-linux-thread-multi/Socket.pm line 370.
```

- looked up the reference and fixed:

```
-  my $r = $self->r;
-  my ($serverport,$serveraddr) = sockaddr_in($r->connection->local_addr);
-  my ($remoteport,$remoteaddr) = sockaddr_in($r->connection->remote_addr);
-  return $serveraddr eq $remoteaddr;
+  my $c = $self->r->connection;
+  require APR::SockAddr;
+  return $c->local_addr->ip_get eq $c->remote_addr->ip_get;
```

Porting: Getting rid of `Apache::compat`

```
use Apache2 ();
use lib qw(/home/httpd/2.0/perl);
#use Apache::compat ();
```

Porting: Ensuring that no-one loads `Apache::compat`

```
--- Apache/MP3.pm.5     2003-06-06 16:17:50.000000000 +1000
+++ Apache/MP3.pm       2003-06-06 16:21:14.000000000 +1000
@@ -3,2 +3,6 @@

+BEGIN {
+    die "Apache::compat is loaded loaded" if $INC{'Apache/compat.pm'};
+}
+
 use strict;
```

- and indeed something has loaded `Apache::compat` indirectly.

```
--- Apache/compat.pm.orig   2003-06-03 16:11:07.000000000 +1000
+++ Apache/compat.pm        2003-06-03 16:11:58.000000000 +1000
@@ -1,5 +1,9 @@
 package Apache::compat;

+BEGIN {
+    use Carp;
+    Carp::cluck("Apache::compat is loaded by");
+}
```

- Found the guilty party: CGI.pm 2.89. Updated to 2.93 - the problem has gone

Porting: ModPerl::MethodLookup

- *startup.pl*

```
use Apache2 ();
use lib qw(/home/httpd/2.0/perl);
{
  package ModPerl::MethodLookupAuto;
  use ModPerl::MethodLookup;

  use Carp;
  sub handler {
      # look inside mod_perl:: Apache:: APR:: ModPerl:: excluding DESTROY   #
      my $skip = ’^(?!DESTROY$’;
      *UNIVERSAL::AUTOLOAD = sub {
          my $method = $AUTOLOAD;
          return if $method =~ /DESTROY/;
          my ($hint, @modules) =
              ModPerl::MethodLookup::lookup_method($method, @_);
          $hint ||= "Can’t find method $AUTOLOAD";
          croak $hint;
      };
      return 0;
  }
}
1;
```

- *httpd.conf*:

```
PerlChildInitHandler ModPerl::MethodLookupAuto
```

- restart the server:

```
[Fri Jun 06 16:28:32 2003] [error] failed to resolve handler ‘Apache::MP3’
[Fri Jun 06 16:28:32 2003] [error] [client 127.0.0.1] Can’t locate
object method "boot" via package "mod_perl" at .../Apache/Constants.pm
line 8. Compilation failed in require at .../Apache/MP3.pm line 12.
```

- Fix: use the new module

```
-use Apache::Constants qw(:common REDIRECT HTTP_NO_CONTENT DIR_MAGIC_TYPE HTTP_NOT_MODIFIED);
+use Apache::Const -compile => qw(:common REDIRECT HTTP_NO_CONTENT
+                                 DIR_MAGIC_TYPE HTTP_NOT_MODIFIED);
```

- Adjust constants, automatic:

```
% perl -pi -e ’s/return\s(OK|DECLINED|FORBIDDEN| \
  REDIRECT|HTTP_NO_CONTENT|DIR_MAGIC_TYPE| \
  HTTP_NOT_MODIFIED)/return Apache::$1/xg’ Apache/MP3.pm
```

and manual:

```
-    push(@directories,$d) if !$seen{$d}++ && $mime eq DIR_MAGIC_TYPE;
+    push(@directories,$d) if !$seen{$d}++ && $mime eq Apache::DIR_MAGIC_TYPE;
```

- the next error is:

```
[Fri Jun 06 17:28:00 2003] [error] [client 127.0.0.1]
Can’t locate object method "header_in" via package
"Apache::RequestRec" at .../Apache/MP3.pm line 85.
```

- go to the reference, find that the API have changed:

```
$r->header_in($foo);           =>   $r->headers_in->{$foo};
$r->header_out(foo => $bar);  =>   $r->headers_out->{foo} = $bar;
```

- Fix:

```
% perl -pi -e ’s|header_in\((.*?)\)|headers_in->{$1}|g’ Apache/MP3.pm
% perl -pi -e ’s|header_out\((.*?)\s*=>\s*(.*?)\);|headers_out->{$1} = $2;|g’ Apache/MP3.pm
```

- Next error:

```
[Fri Jun 06 18:36:35 2003] [error] [client 127.0.0.1]
to use method ’FETCH’ add:
      use APR::Table ();
at .../Apache/MP3.pm line 85
```

- what used to be:

```
[Fri Jun 06 18:35:53 2003] [error] [client 127.0.0.1]
Can’t locate object method "FETCH" via package "APR::Table"
at .../Apache/MP3.pm line 85.
```

- fix:

```
+use APR::Table ();
```

- More similar errors which only require loading of the module, `ModPerl::MethodLookup` tells me what I should load:

```
+use Apache::Connection ();
+use Apache::SubRequest ();
+use Apache::Access ();
+use Apache::RequestIO ();
+use Apache::RequestUtil ();
+use Apache::RequestRec ();
+use Apache::ServerUtil ();
+use Apache::Log;
```

- Next error:

```
[Fri Jun 06 18:40:34 2003] [error] [client 127.0.0.1]
Don’t know anything about method ’send_http_header’
at .../Apache/MP3.pm line 498
```

- Find in the reference that the API has changed:

```
-  $self->r->send_http_header( $self->html_content_type );
+  $self->r->content_type( $self->html_content_type );
```

- Next a problem with send_fd(), not in API, fixing:

```
-    if($r->request($r->uri)->content_type eq ’audio/x-scpls’){
-      open(FILE,$r->filename) || return 404;
-      $r->send_fd(\*FILE);
-      close(FILE);
+
+    if($r->content_type eq ’audio/x-scpls’){
+      $r->sendfile($r->filename) || return Apache::NOT_FOUND;
```

- Finally `log_reason` is now `log_error`:

```
  -  $self->r->log_reason('Invalid parameters -- possible attempt to circumvent checks.');
  +  $r->log_error('Invalid parameters -- possible attempt to circumvent checks.')
;
```

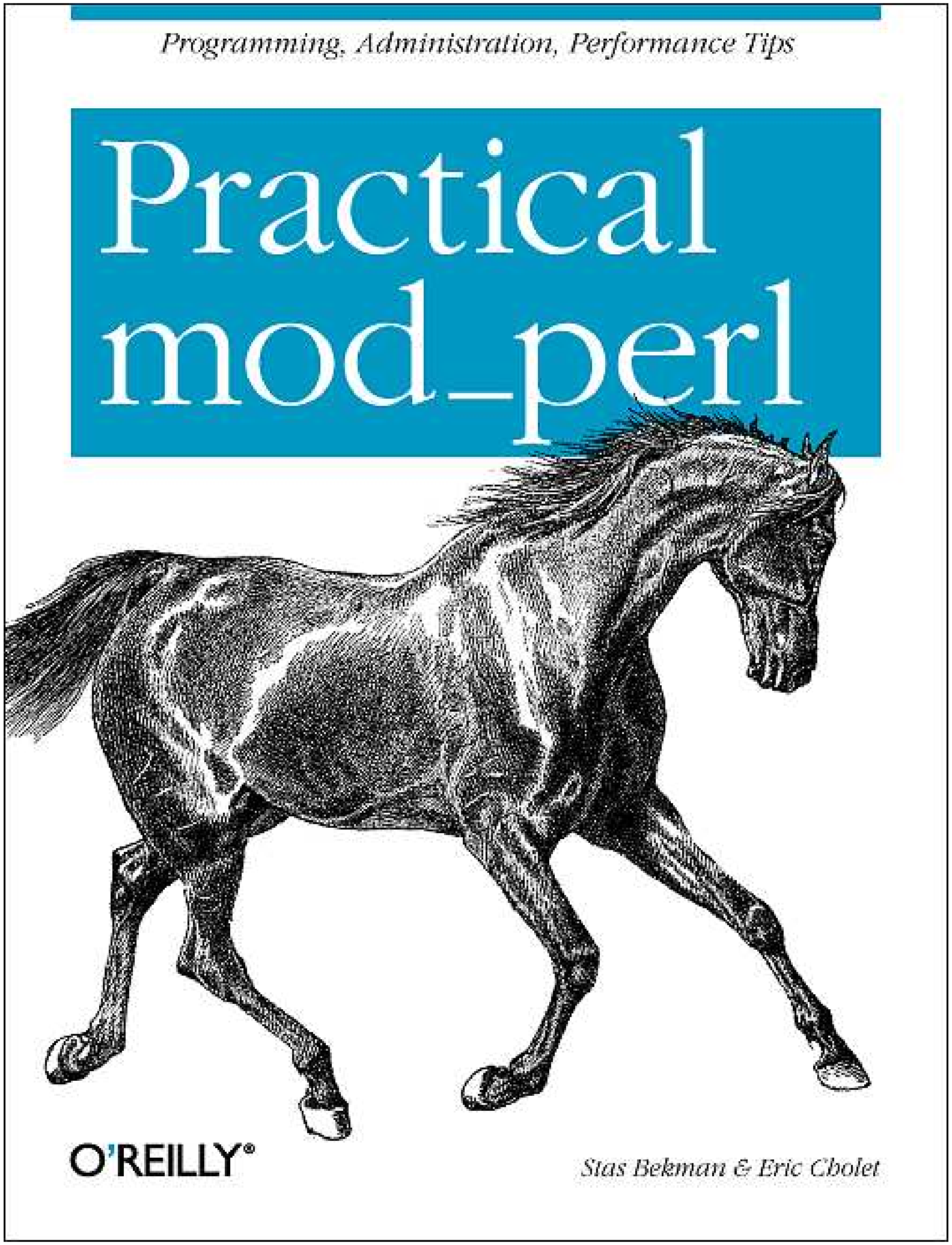
1.13 References

- All the information can be found at:

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

- Further Questions?
  - Grab me at the corridor and demand answers
  - Ask at modperl@perl.apache.org

1.14 A shameless plug



Programming, Administration, Performance Tips

Practical mod_perl

O'REILLY®

Stas Bekman & Eric Cholet