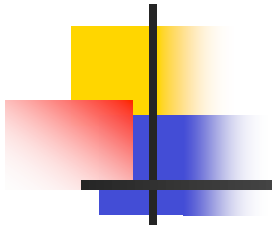
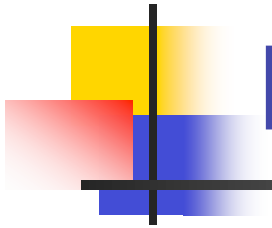


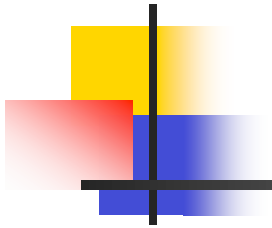
Advanced Web Server Programming

(Week 11 Lectures)



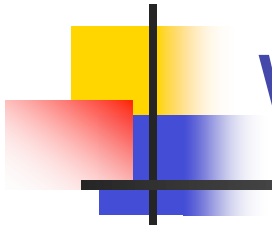
Learning Objectives

- Learn about the various ways of programming a web server.
- Look at an example case study with Apache's modules.



Lecture Outline

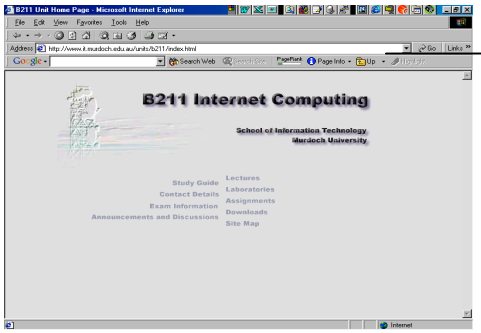
- Web application programming.
- Approaches to extending a web server's functionality.
  - CGI
  - Server APIs.
  - Embedded interpreters.
  - Server-side Includes.



Web Application Technologies

- Today, we refer to web application languages as being client-side or server-side languages.
- Some common client-side languages:
  - JavaScript
  - VBScript
- Some common server-side languages:
  - CGI with Perl
  - PHP (PHP Hypertext Preprocessor)
  - Microsoft's ASP (Active Server Pages) and ASP.NET
  - JSP (Java Server Pages) and Java Servlets
  - Macromedia's ColdFusion

How Client-side Languages Work



1. Browser requests URL

```
...HTML...
...script code...
...HTML...
...script code...
```

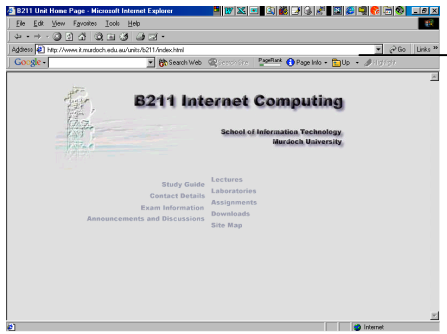
2. Web server sends HTML and script code back to browser

3. Browser interprets the script code and displays appropriately

Client-side

Server-side

How Server-side Languages Work



1. Browser requests URL

```
...HTML...
...script code...
...HTML...
...script code...
```

2. Web server invokes interpreter to turn the script code into HTML

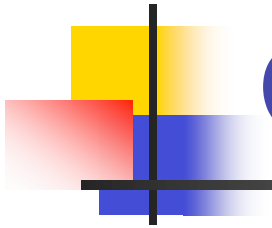
```
...HTML...
...HTML...
...HTML...
...HTML...
```

3. Web server sends HTML back to browser

4. Browser reads and displays HTML

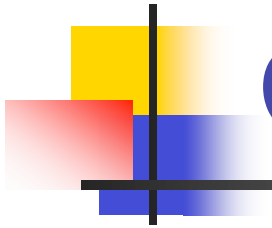
Client-side

Server-side



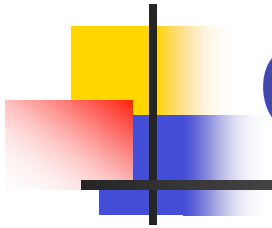
Client vs Server Side Languages

- This emphasis on client versus server side belies the true complexity of web applications.
  - Most web applications involves some processing on both sides.
  - Eg. PHP scripts processed on the server-side generally produces JavaScript code to be processed on the client-side.



Complexity of Web Applications

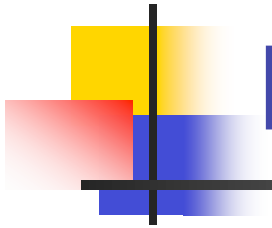
- The client vs server side emphasis also doesn't show the true nature of web server programming.
  - Generally, client and server-side languages concentrates on producing web-site content.
  - As we have in the first part of the semester, a web server does a lot more than just produce web-site content.



Complexity of Web Applications

- In this lecture, we will look at how web servers are extended to support programming web applications.

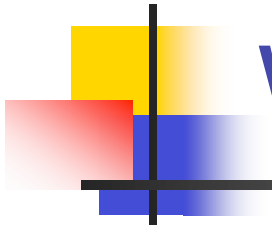
- First, a look at the the evolution of web application programming…



Before the WWW…

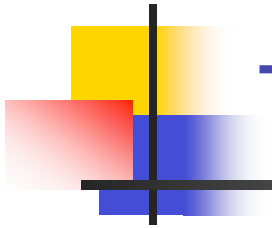
- Before the World-Wide-Web, programming networked applications requires very in-depth programming skills.

- Need to develop
  - A communication protocol with ways of transferring messages.
  - Server to listen to ports and service requests.
  - Client able to construct requests, process results, etc.
  - A user-interface with buttons, input textboxes, etc.
  - Some of these tasks you have touched on in Assignment 1!



With the WWW…

- With the World-Wide-Web, developing network applications becomes a lot simpler.
  - Make use of APIs in PHP, ASP.NET, etc.
  - Heralded a new generation of "web programmer" that can develop quite sophisticated networks applications without needing that deep programming knowledge.
  - Concentrates on dealing with content.
  - Can depend on existing web browsers and web servers to do a lot of the tasks mentioned in the previous overhead.



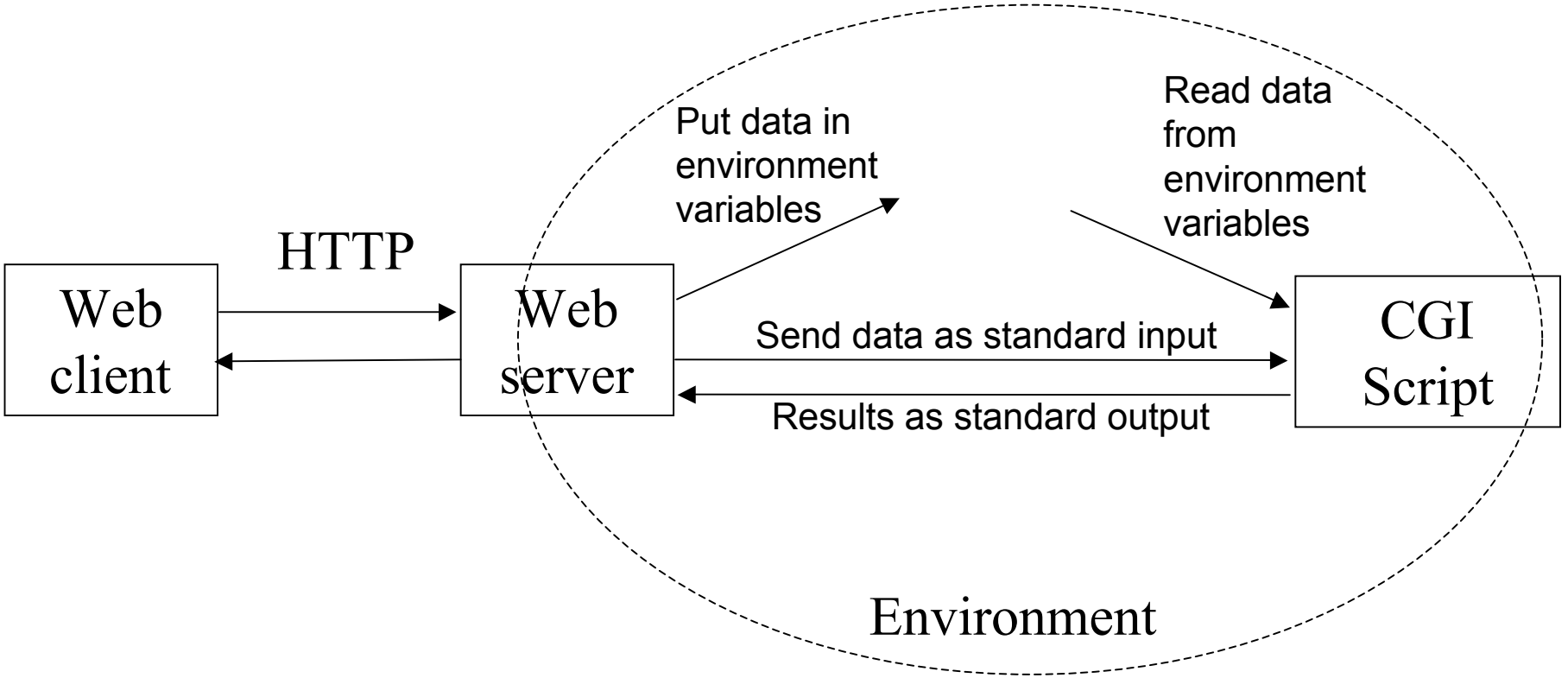
The Common Gateway Interface (CGI)

- The capability for web servers to support general "web applications" were started by the the definition of the Common Gateway Protocol (CGI) in the early 1990's.

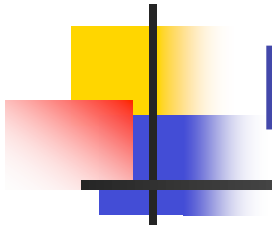
- CGI is platform-independent interface specification used to run software in conjunction with an HTTP server.

  - It acts as a "gateway" between non-HTTP servers/software with the HTTP (web) server.

CGI: The Process



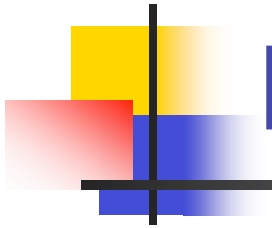
Further details of CGI specifications, see hoohoo.ncsa.uiuc.edu/cgi/interface.html and www.w3.org/CGI/



Popularity of CGI

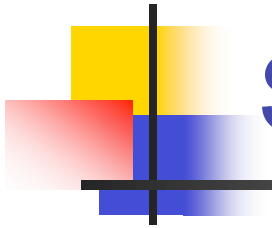
- CGI became very popular and was supported by many web servers as the WWW became popular.

- It was simple, cross-platform way of putting a network application together.



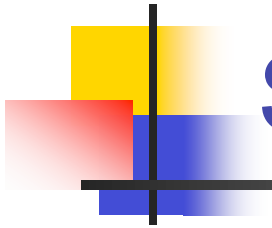
Problems with CGI

- The problems with CGI were with its performance.
- Everytime a CGI application is run, the server has to
  - Set up the environment and environment variables.
  - Read the script into memory.
  - Execute the script.



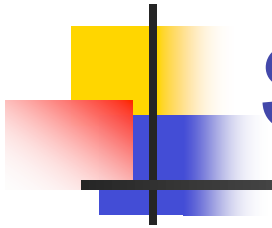
Solutions to performance problems

- Developments in response to CGI performance problems have led to the myriads of web application programming options today.
- The solutions are mainly in getting the web server in a state where it is ready to produce the results as efficiently as possible.
  - Do as little processing for a single application as possible.



Solutions to performance problems

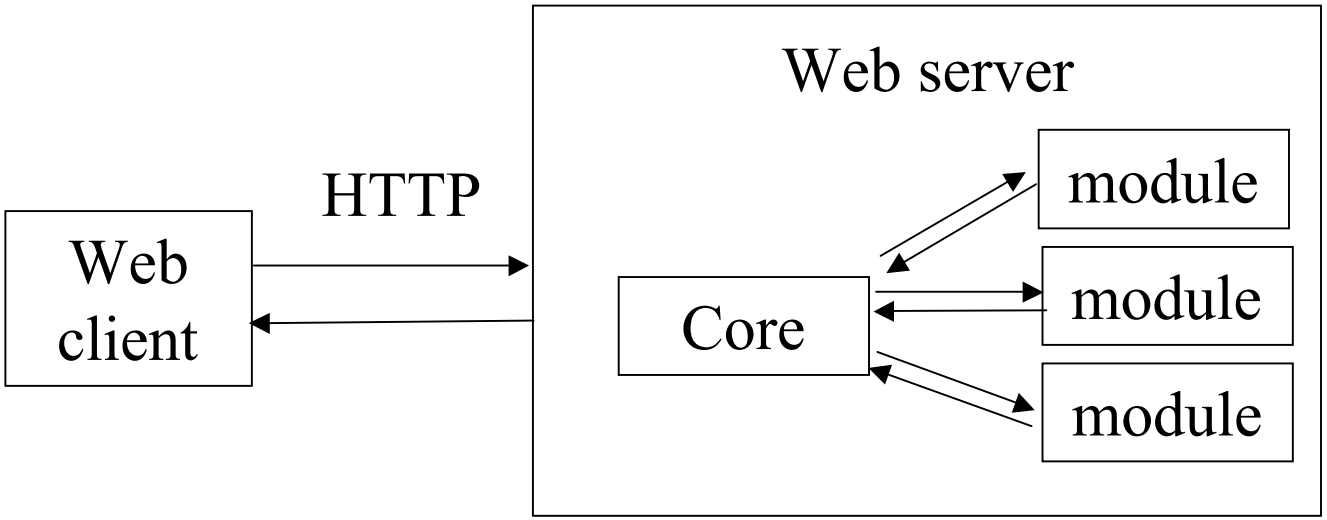
- Some example solutions:
  - Pushing processing to client-side.
    - Eg. generating HTML with JavaScript code.
  - Server APIs.
    - Eg. Netscape's NSAPI, Microsoft's ISAPI supported in IIS, Apache API.
  - Embedded interpreters.
    - Enbedded Python, Java servlet API.
  - Server-side Includes.
    - Eg. Apache's XSSI, Netscape's server-side JavaScript.
    - Can consider PHP, ASP and Coldfusion as advanced examples of this.



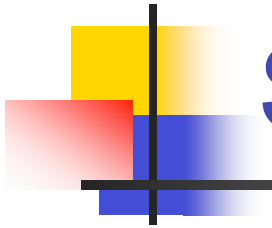
Server API

- To reduce the overhead of starting new applications using methods like CGI, web servers started supporting their own proprietary API to allow developers to extend the web server by directly linking into the server's executable.

Server API

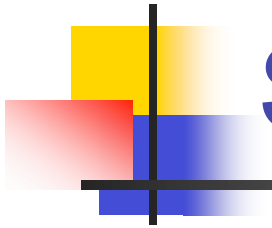


- Compare with CGI.
- Save much overheads.



Server API

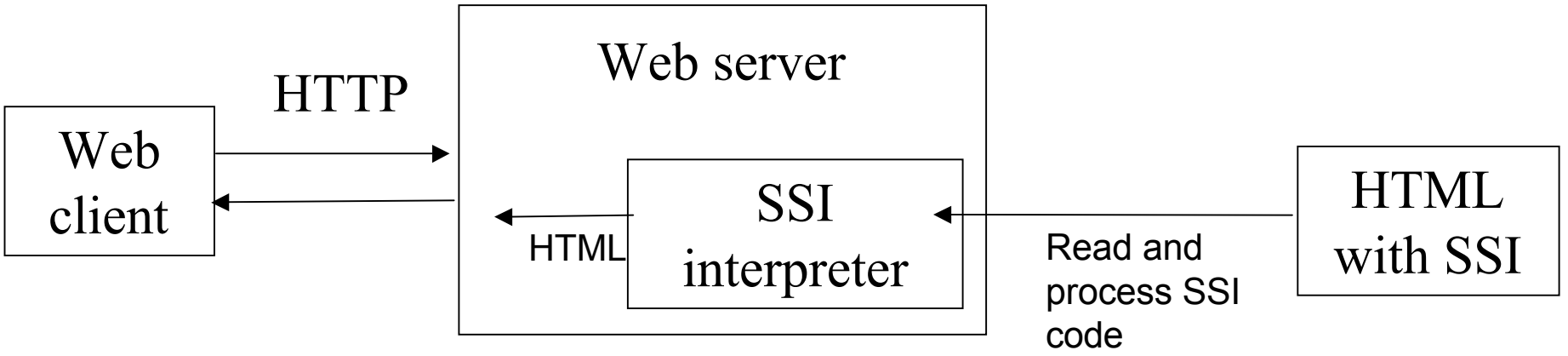
- Advantage:
  - Much more efficient in execution.
- Disadvantage:
  - Closely tied to particular web server - not portable.
  - Big learning curve to learn to link into web server.

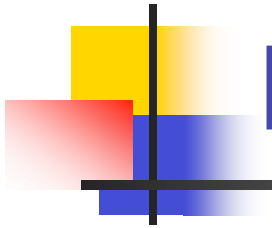


Server-Side Includes

- Another solution is Server-side Includes.
- As we have seen with Apache's XSSI, it can used as a lightweight server-side processing language.
- More flexible compared to Server APIs, since developers can develop code without having to link them into the web server everytime.

Server-side Includes

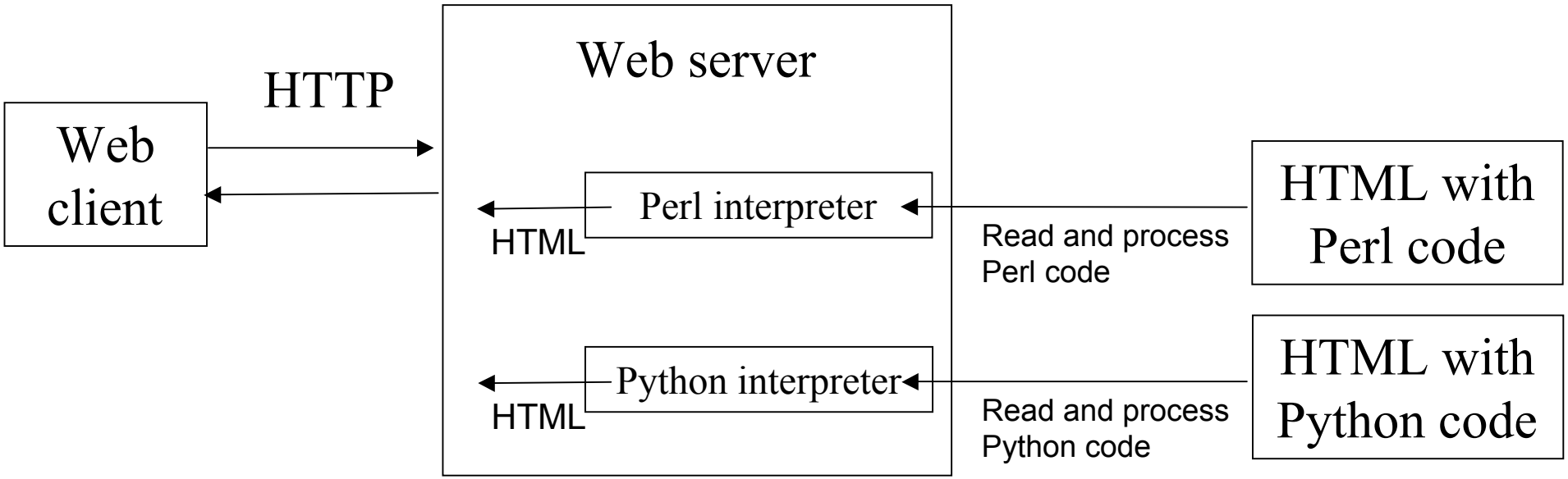


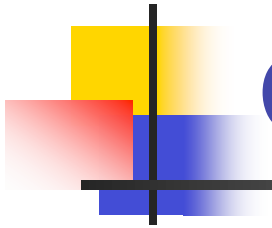


Embedded Interpreter

- Embedding interpreters is the same approach as SSI, but embeds general language interpreters into the web server.
- Developers can develop in languages of their choice, instead of being restricted by what is define in the SSI language.
- However, the processing of the language code is still done by a part of the web server, instead of a completely in different process in the OS.
  - Therefore faster compared to CGI.

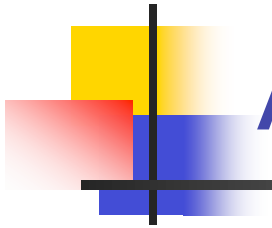
Server-side Includes





Case Study: Apache's Module API

- An example of Server API is the Apache's API for modules.
  - It is an API to allow you to write your own Apache modules.
  - Modules we covered at the beginning of the semester like `mod_status` were created with this API.



Apache APIs

- The Apache's module APIs provides a facility to extend and control all aspects of the Apache Request Loop.

Apache's Request Loop

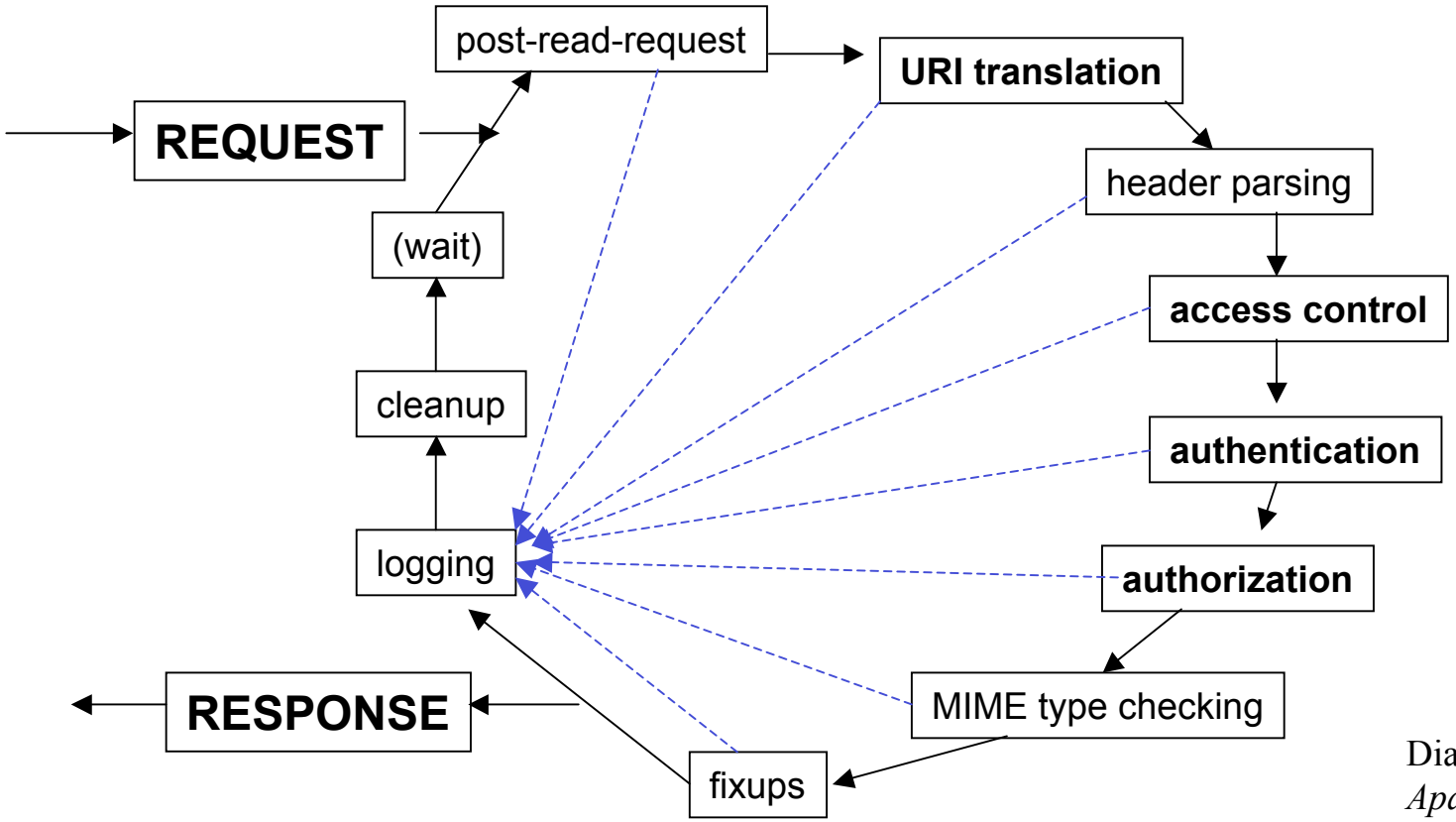


Diagram modified from *Writing Apache Modules with Perl and C*, Stein & MacEachern, 1999, page 60.

Example Module

Hello.pm in module directory

```
package Apache::Hello;
# file: Apache/Hello.pm

use strict vars;
use Apache::Constants ':common';

sub handler {
    my $r = shift;
    $r->content_type('text/html');
    $r->send_http_header;
    return OK if $r->header_only;

    my $remote_host = $r->get_remote_host;
    my $user_agent  = $r->header_in('User-Agent');

    $r->print(<<END);
<html>
<header>
<title>Greetings</title>
</header>
<body>
<h1>Greetings from Perl</h1>
<p> Hello to you, <b>$remote_host</b>.</p>
<p> You are the lucky user of a <i>$user_agent</i></p>
<hr>
END

   $r->printf("<p>Path info = <b>%s</b></p><hr>\n",$r->path_info)
         if $r->path_info;
   $r->printf("<p>Query = <b>%s</b></p><hr>\n",$r->args)
         if $r->args;

print <<END;
</body>
</html>
END

    return OK;
}
1;
```

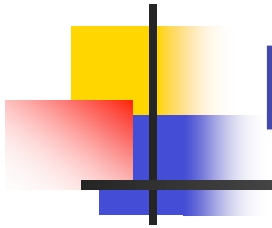
Example modified from: http://stein.cshl.org/~lstein/talks/perl_conference/apache_api/mp_dynamic.html

Example Module

In httpd.conf

```
<Location /hello_perl>
    SetHandler  perl-script
    PerlHandler Apache::Hello
</Location>
```

Example taken from:
http://stein.cshl.org/~lstein/talks/perl_conference/apache_api/mp_dynamic.html



Further Reading

- Further information on Apache's module APIs
  - http://httpd.apache.org/docs/misc/API.html
  - http://stein.cshl.org/~lstein/talks/perl_conference/apache_api/index.html