# Lecture 11
# Performance

Play

# Outline

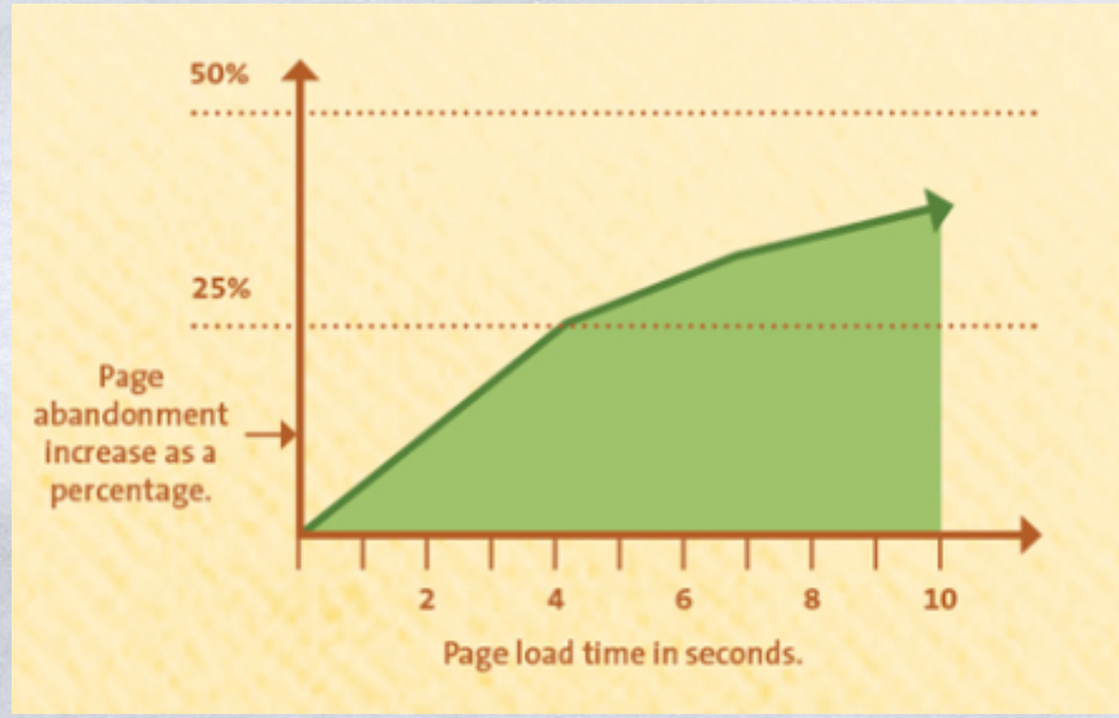
- Web Performance Optimization
- Large Scale Web Application

# What happened?

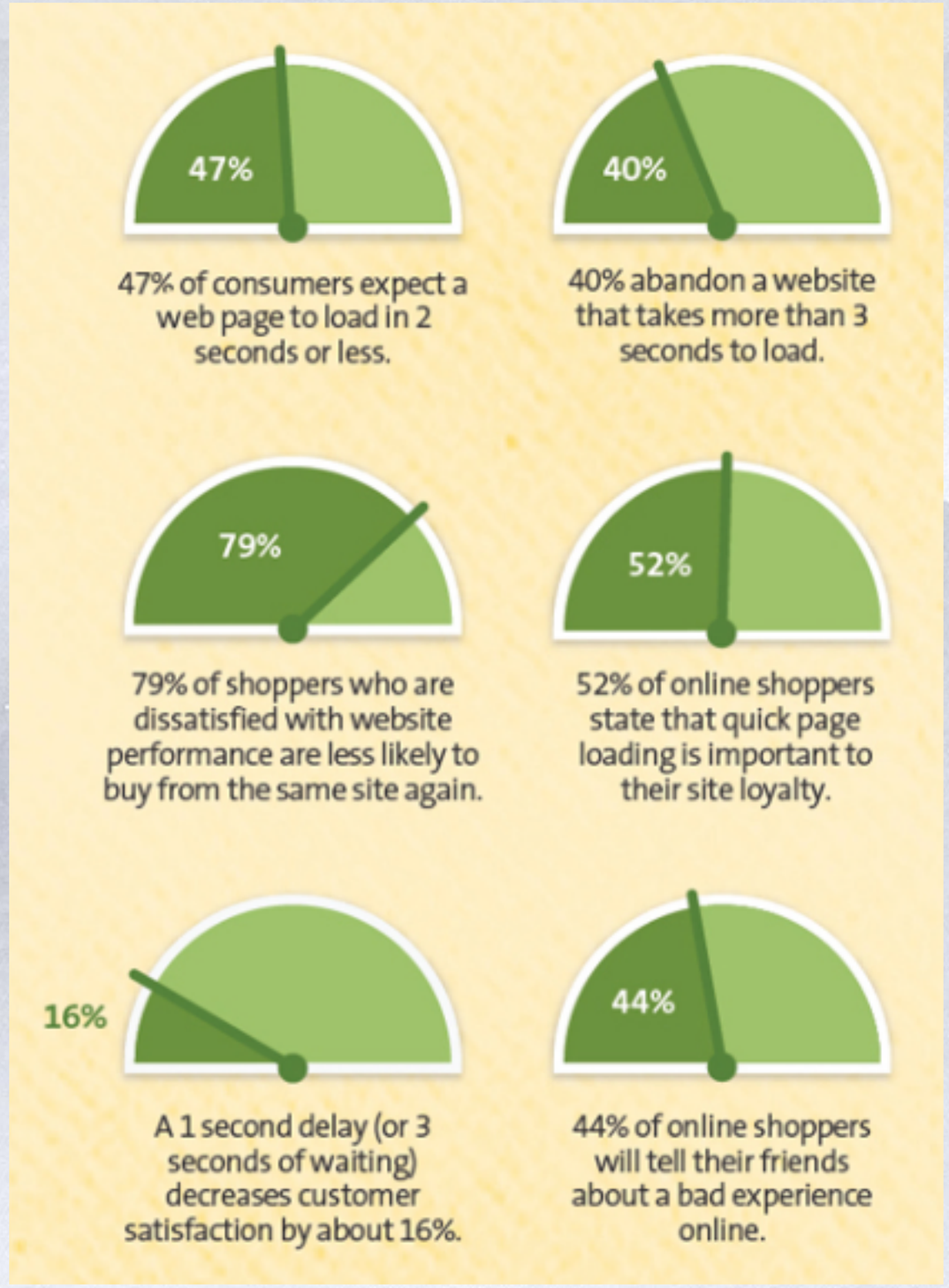
- Service is too busy

# Why Website Performance Matters?

- 40% Abandon a site after waiting 3 seconds for a page to load.
- 80% of the people will NOT return
- Almost half will tell others about their negative experience.
- Amazon: 1% Revenue increase for every 100MS of improvement



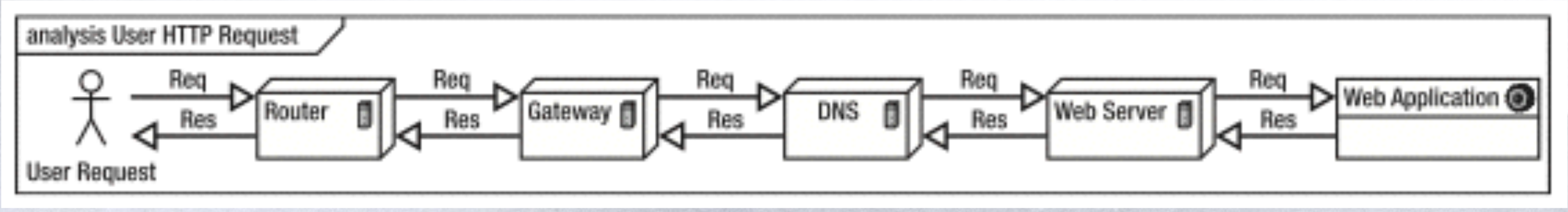

# How Website Performance Affects Shopping Behavior



# Website Speed and SEO

- Website speed is important to your SEO efforts because faster websites are:
  - Easier to crawl
  - Easier to access
  - More likely to rank (although this is marginal)
  - Most importantly, more likely to keep visitors around!

# HTTP request lifecycle



# 网站性能优化思路

网页通过HTTP获得,在浏览器解析,那么网页性能优化思路:

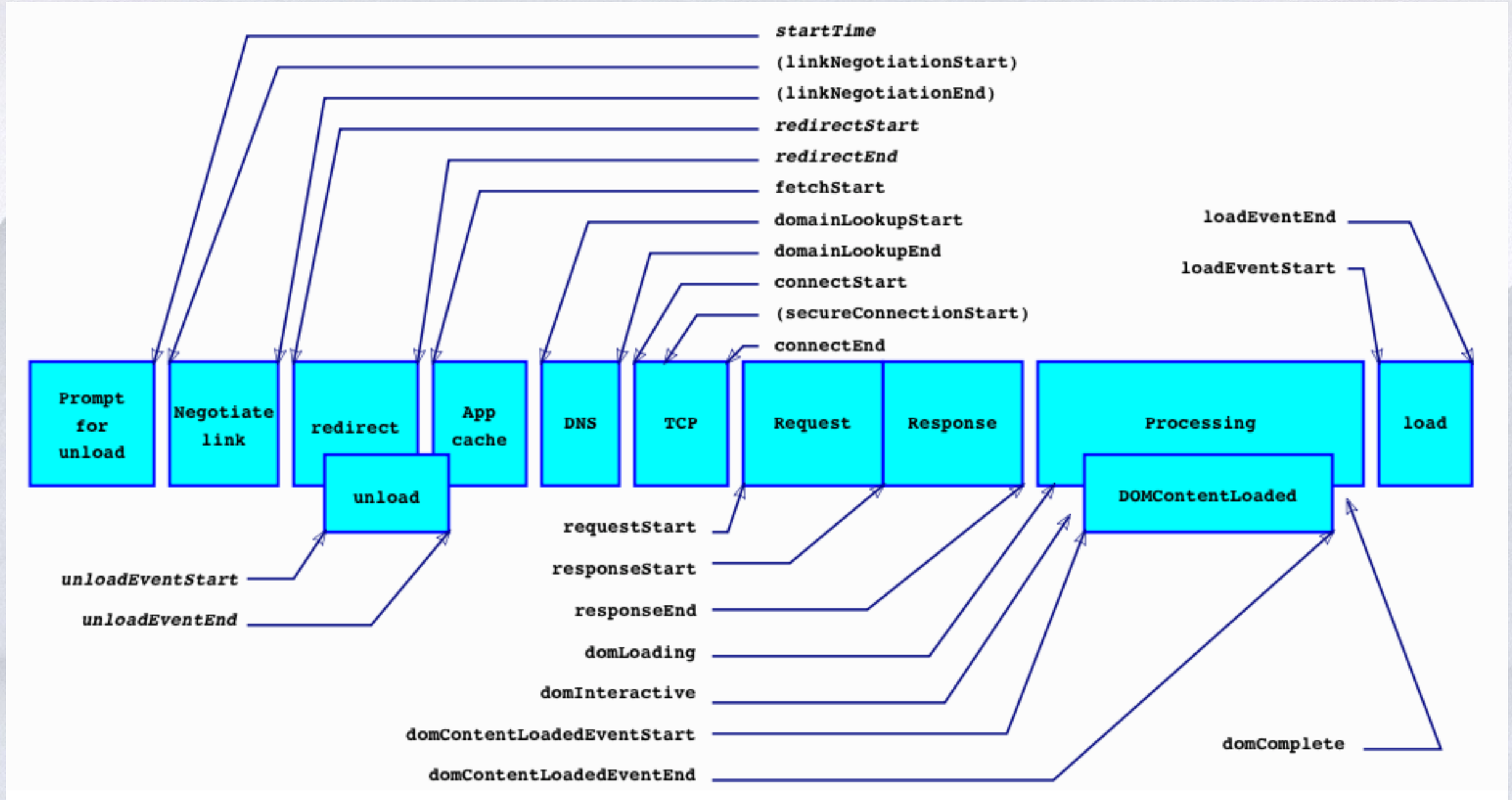
# Benchmarking Utilities

- Apache Bench
  - ab utility bundled with Apache
- Siege
  - http://www.joedog.org/JoeDog/Siege
- http_load (Excellent for latency tests)
  - http://www.acme.com/software/http_load/

# Affecting Your Benchmark Figures

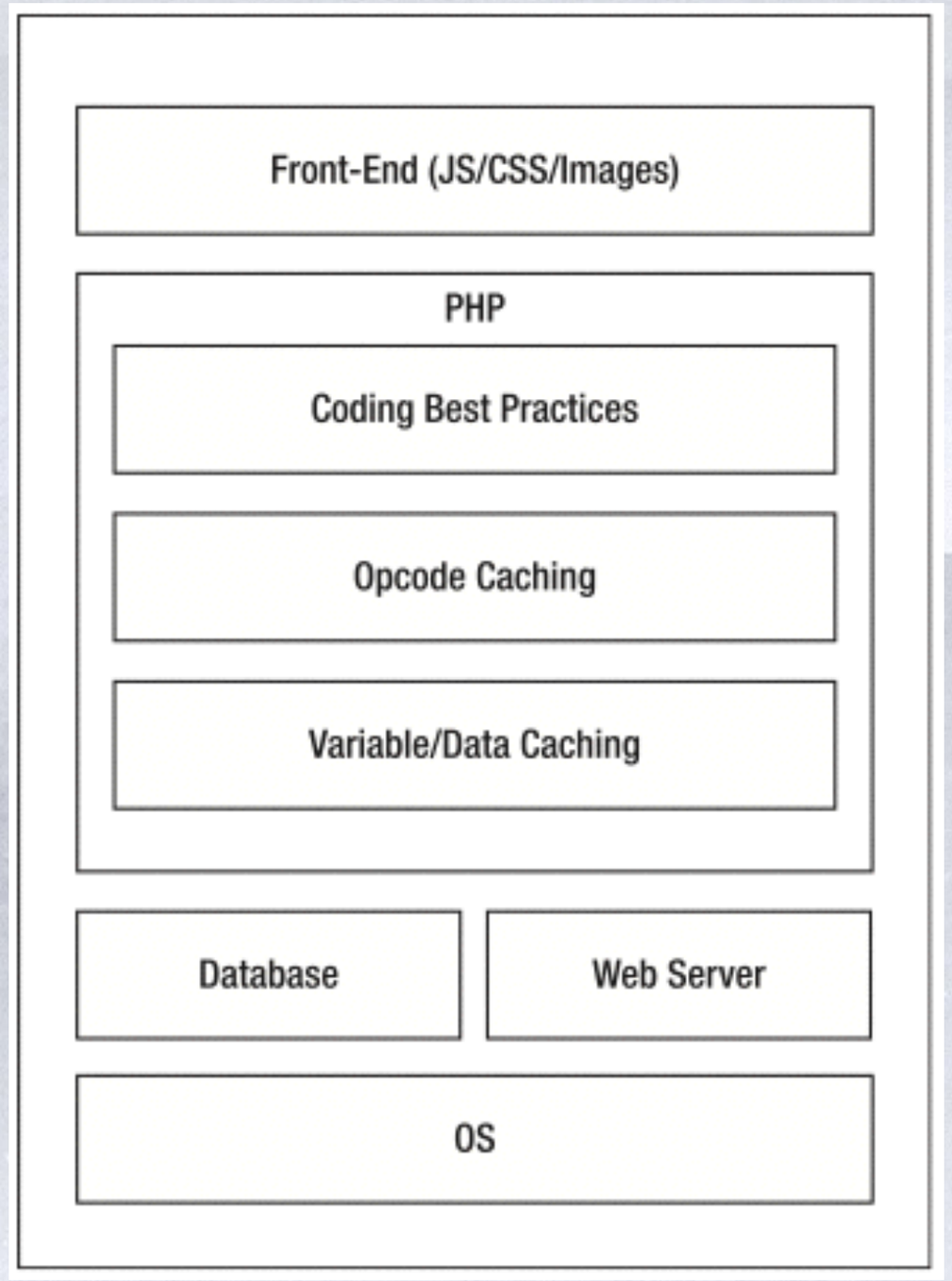
- Geographical location
- Network issues
- Response size
- Code processing
- Browser behavior
- Web server configuration

# Navigation Timing 2



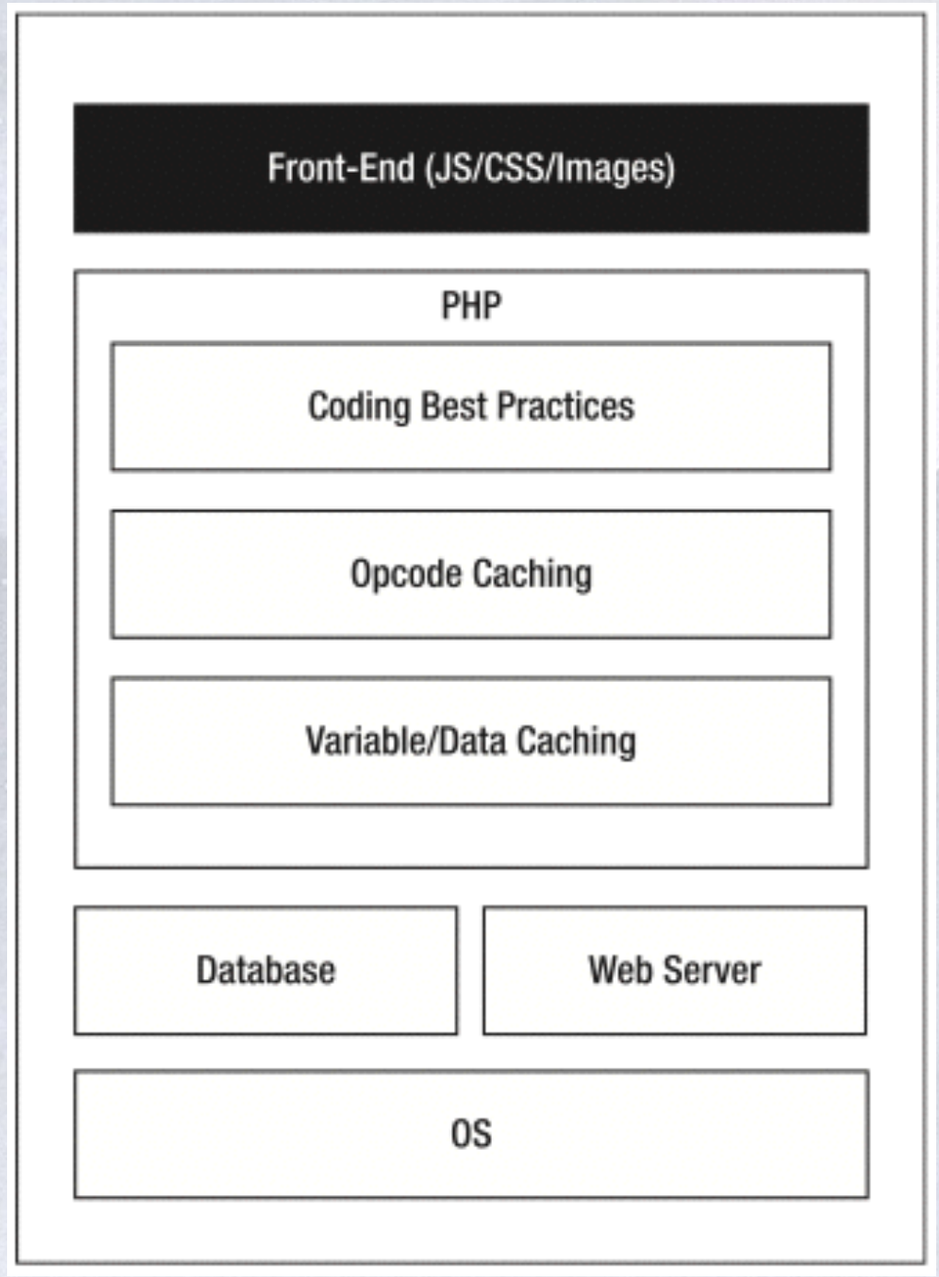

W3C Web Performance Working Group

# The PHP Application Stack



# Improving Client Download and Rendering Performance

- Firebug, YSlow, and Page Speed
  - An in-depth look at the response from a web server
  - Profile front-end logic within your JavaScript
  - An itemized list of resources that the browser will fetch
  - The length of time the browser took to fetch and receive the resource
  - Suggestions on how to optimize the response
- YUI Compressor, Closure Compiler, and Smush.it
  - optimize the response



# YSlow

- YSlow v2's 22 web optimization rules cover the following:
  - CSS optimization
  - Image optimization
  - JavaScript optimization
  - Server optimization

# CSS Optimization Rules

- Place the CSS styles at the top of the HTML document.
- Avoid certain CSS expressions.
- Minify the CSS files.

# Image Optimization Rules

- Use desired image sizes instead of resizing within HTML using width and height attributes.
- Create sprites when possible.

# JavaScript Optimization

- Place JavaScript at the bottom of the HTML.
- Minify JavaScript.
- Make JavaScript external.

# Server Optimization

- Whether the server utilizes Gzip/bzip2 compression
- Whether DNS lookups are reduced
- Whether ETags are implemented

# Optimize HTML

- Standards Compliant
- Remove whitespace
- Simple structures
- Browser Compatibility
- Mobile Compatibility

- Note：Web Standards

# Use Text Instead of Images

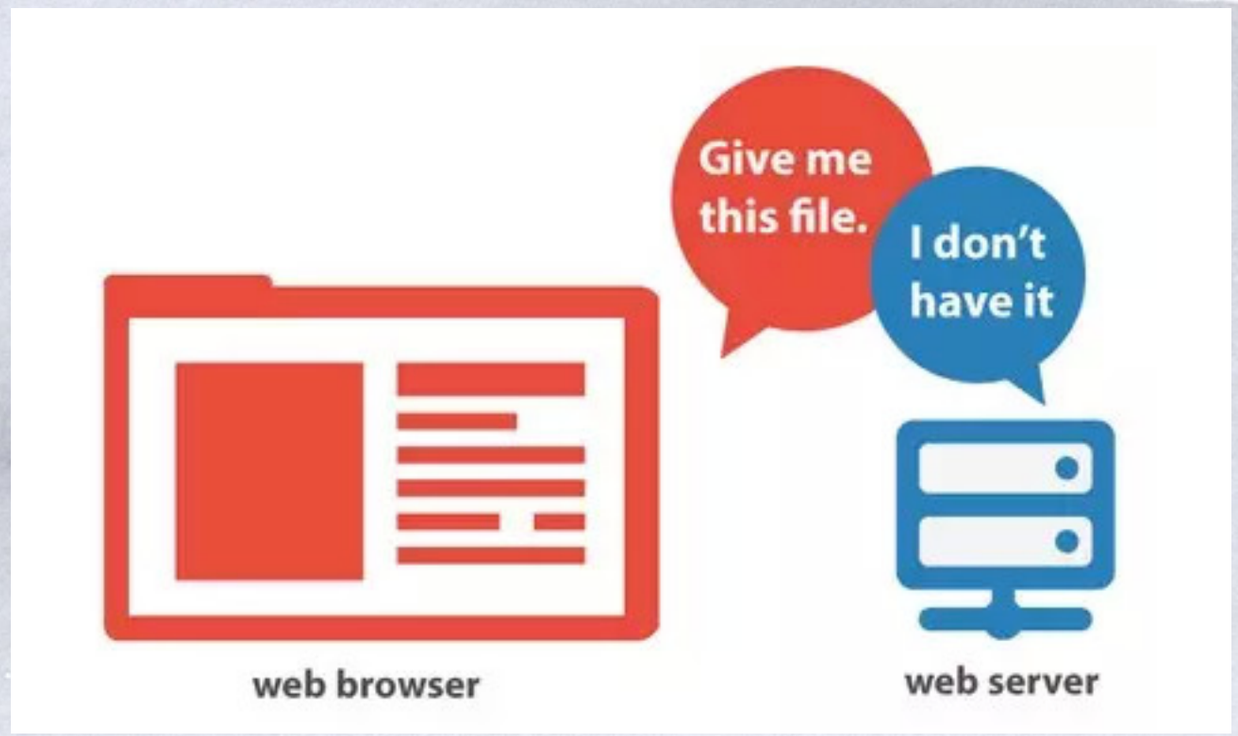
- Use CSS and HTML
- Avoid Images for Headers
- Text is always better for SEO
- Text will always render faster
- Balance good design with great performance

# Cache Your Output

- Minimize Server Processing
- Server Caching for Dynamic Content
- Browser Caching for Static Resources
- Cache Ajax Requests

# Avoid Bad Requests

- Check for 404 Errors
- Avoid Redirects on Resources
- Monitor Server Errors



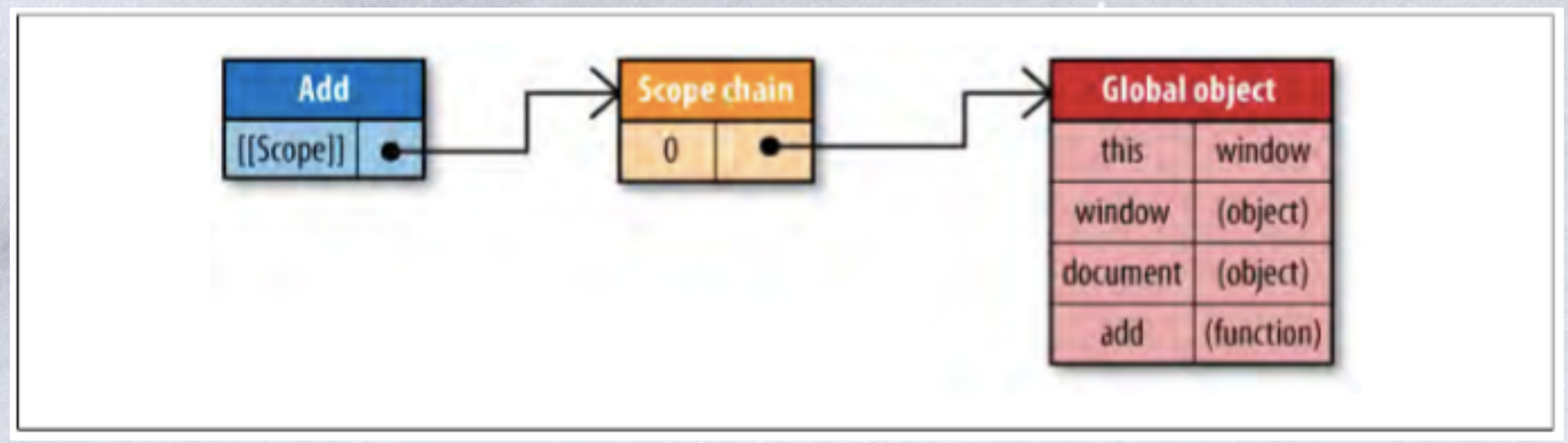

# defer vs async

```
<script src=“main.js”></script>

<script async src="main.js"></script>

<script defer src="main1.js"></script>
<script defer  src="main2.js"></script>
```

# Data Access

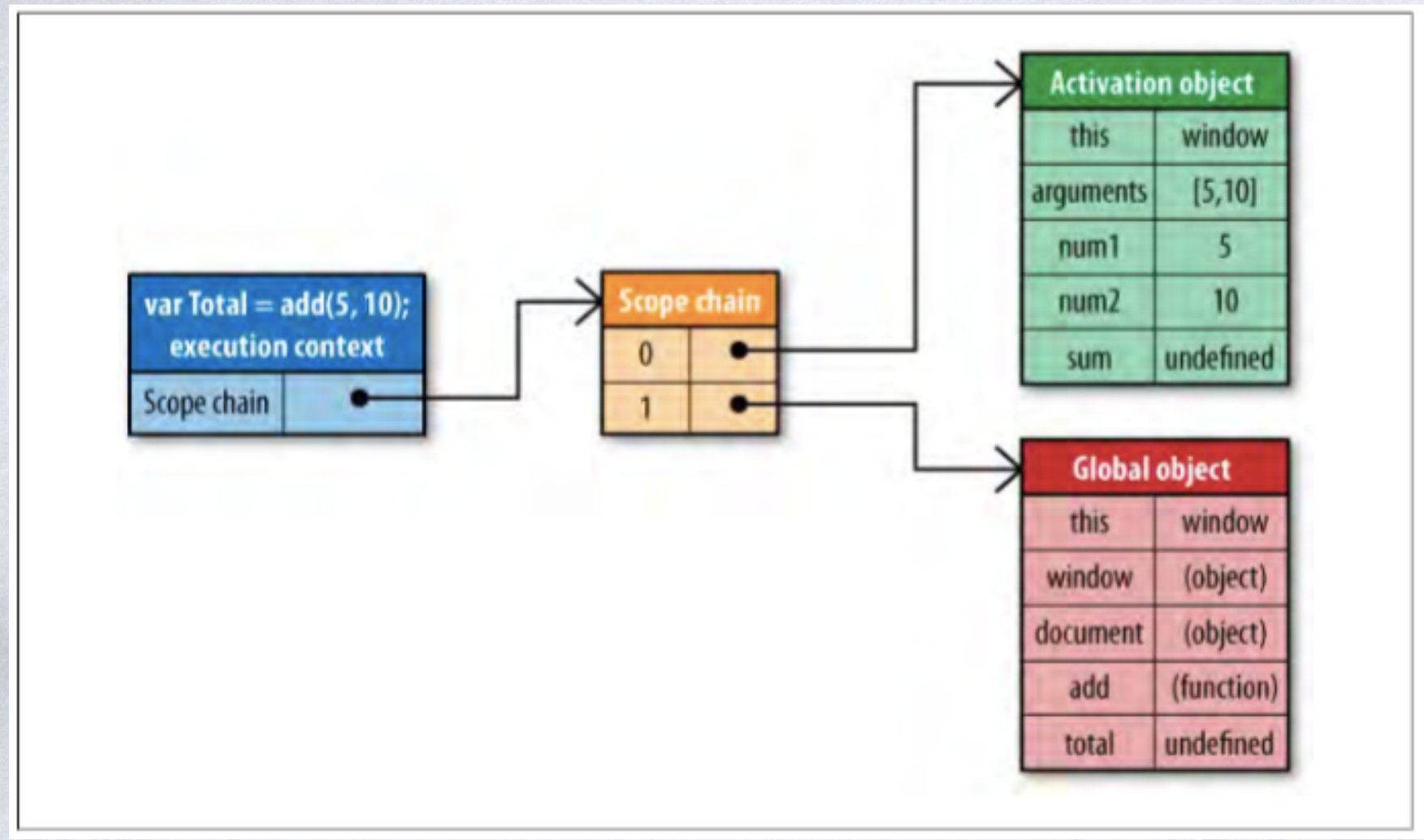
- Scope chain

```
function add(num1, num2){
    var sum = num1 + num2;
    return sum;
}
```



# Executing the add function

- execution context and its scope chain



# For example

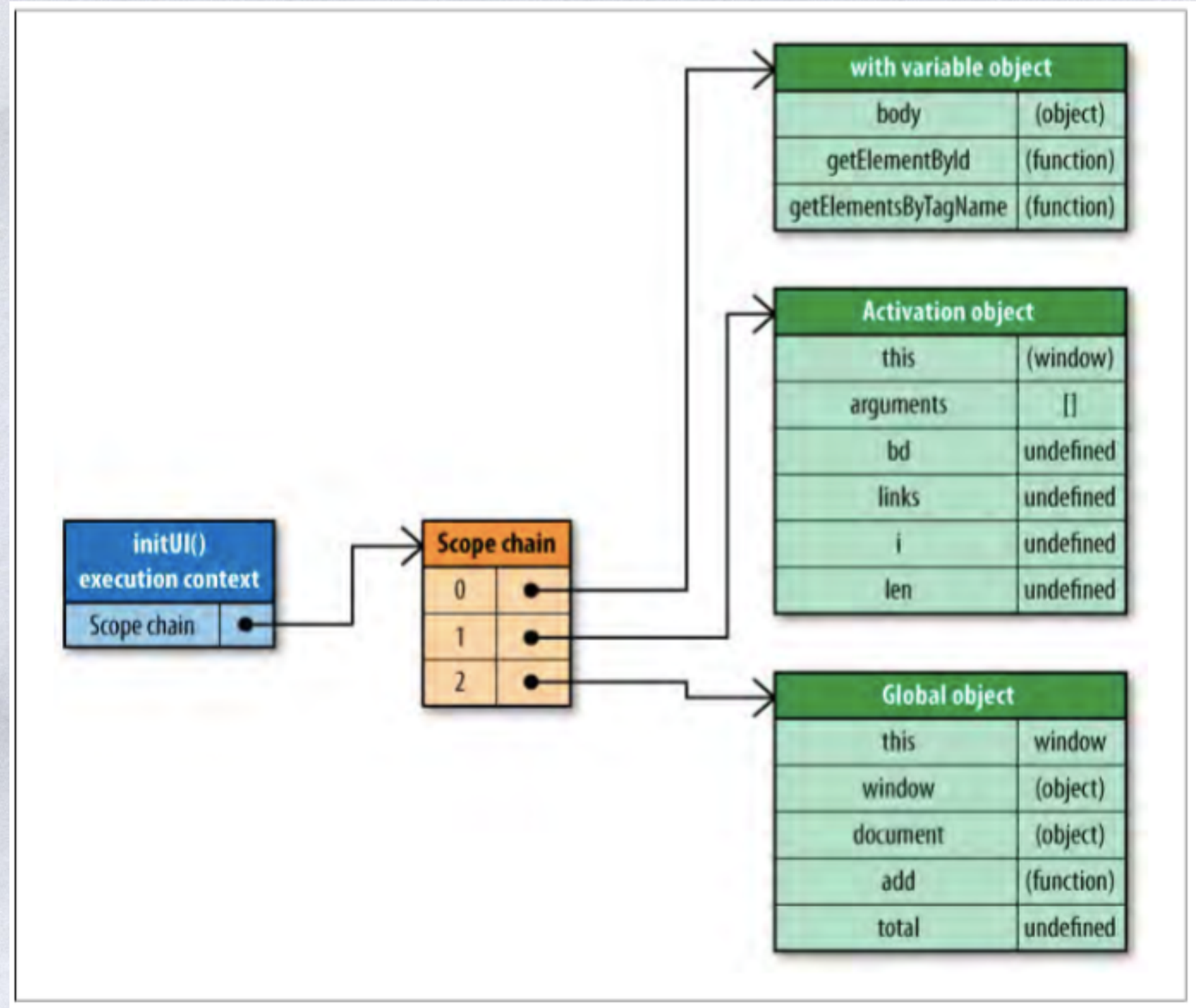
```
function initUI(){
  var bd = document.body,
      links = document.getElementsByTagName("a"),
      i= 0,
      len = links.length;

  while(i < len){
        update(links[i++]);
  }
  document.getElementById("go-btn").onclick = function(){
        start();
  };
  bd.className = "active";
}
```

```
function initUI(){
     var doc = document,
          bd = doc.body,
          links = doc.getElementsByTagName("a"),
          i= 0,len = links.length;
.....}
```

# cope Chain Augmentation

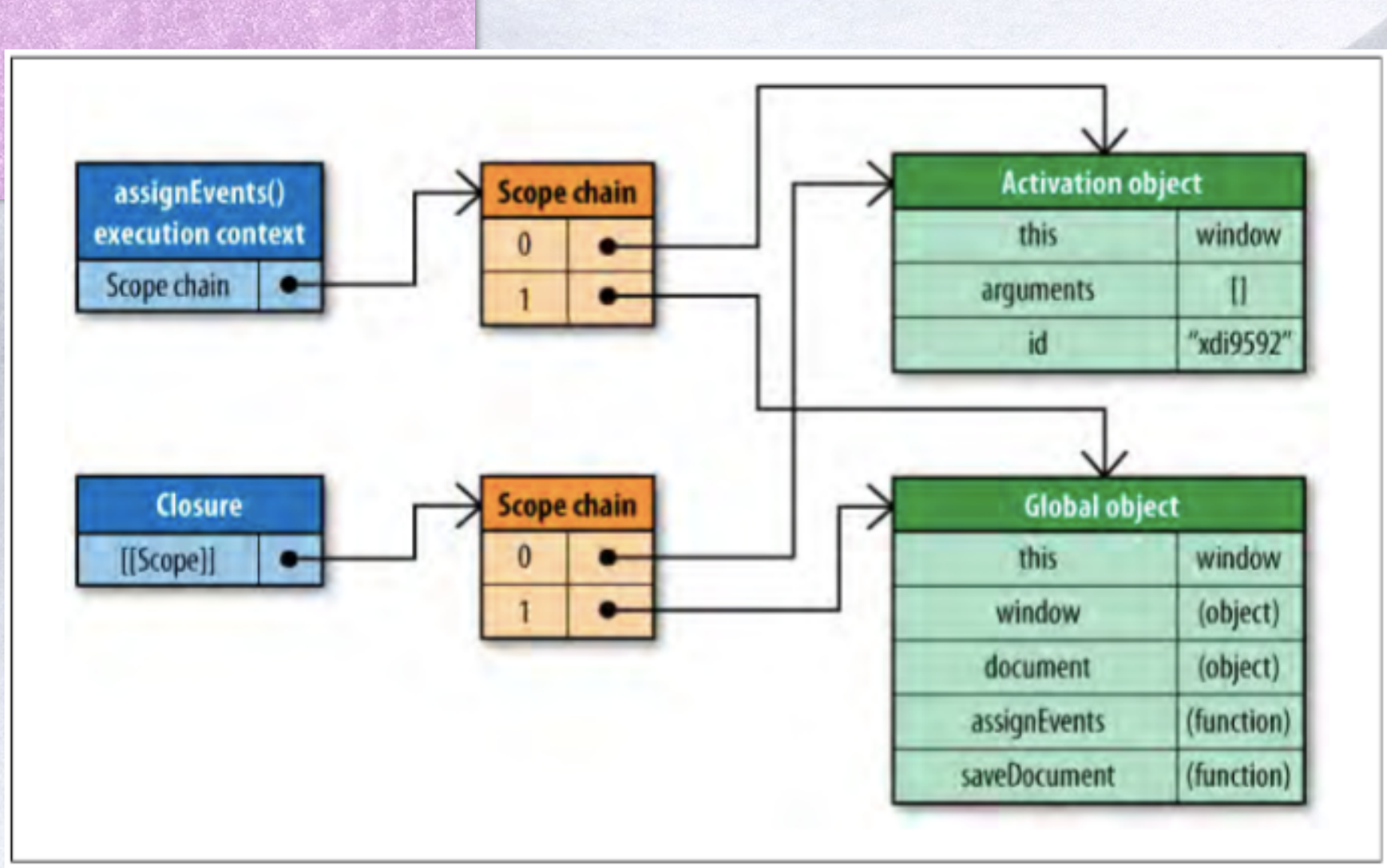
```
function initUI(){
    with (document){ //avoid!
        var bd = body,
            links = getElementsByTagName("a"),
            i= 0,
            len = links.length;
        while(i < len){
            update(links[i++]);
        }
        getElementById("go-btn").onclick = function(){
            start();
        };
        bd.className = "active";
    }
}
```

# Scope Chain Augmentation

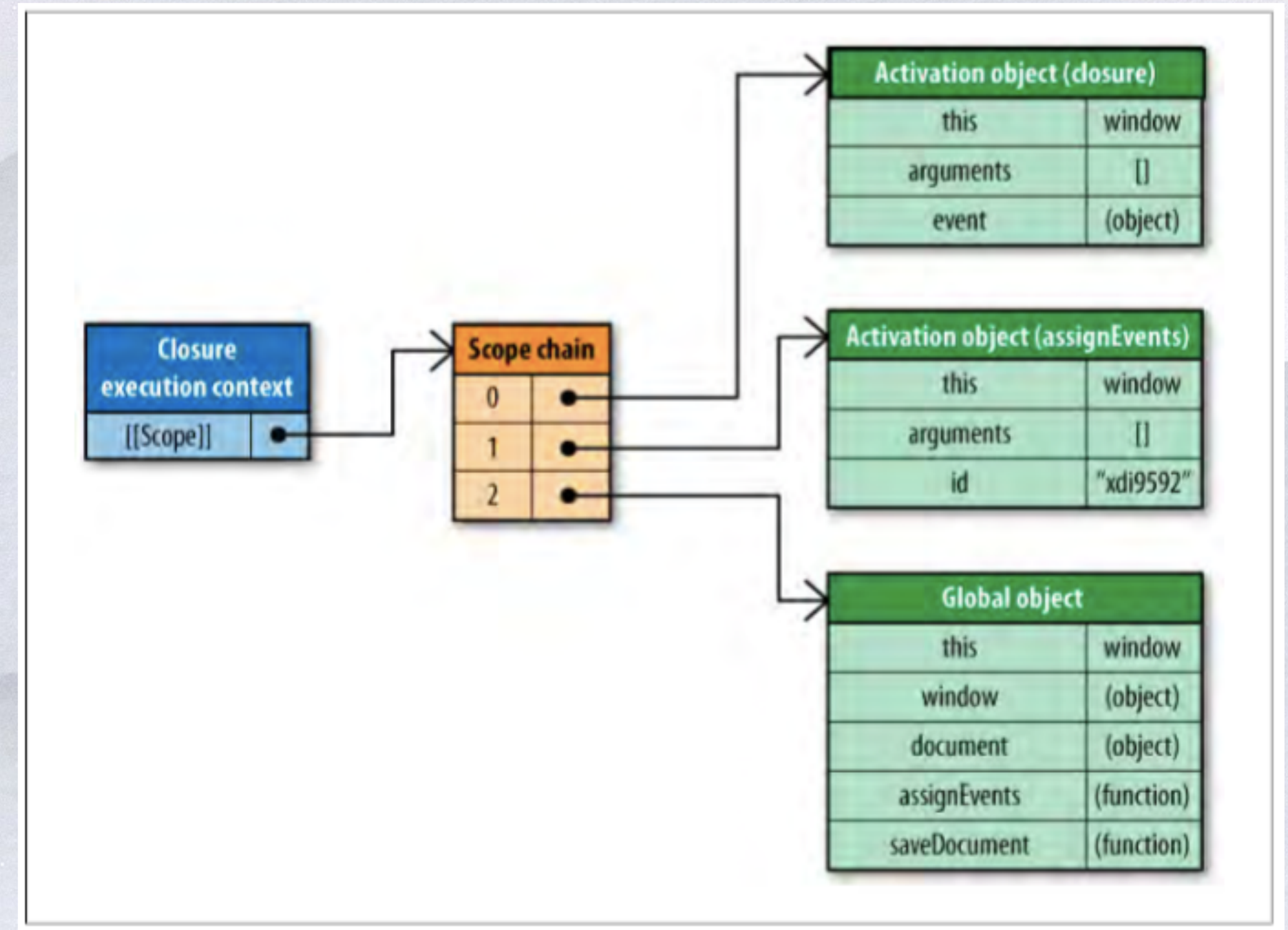


# Closures, Scope, and Memory

```
function assignEvents(){
  var id = "xdi9592";
  document.getElementById(“save-btn”)
      .onclick = function(event){
  saveDocument(id); };
}
```



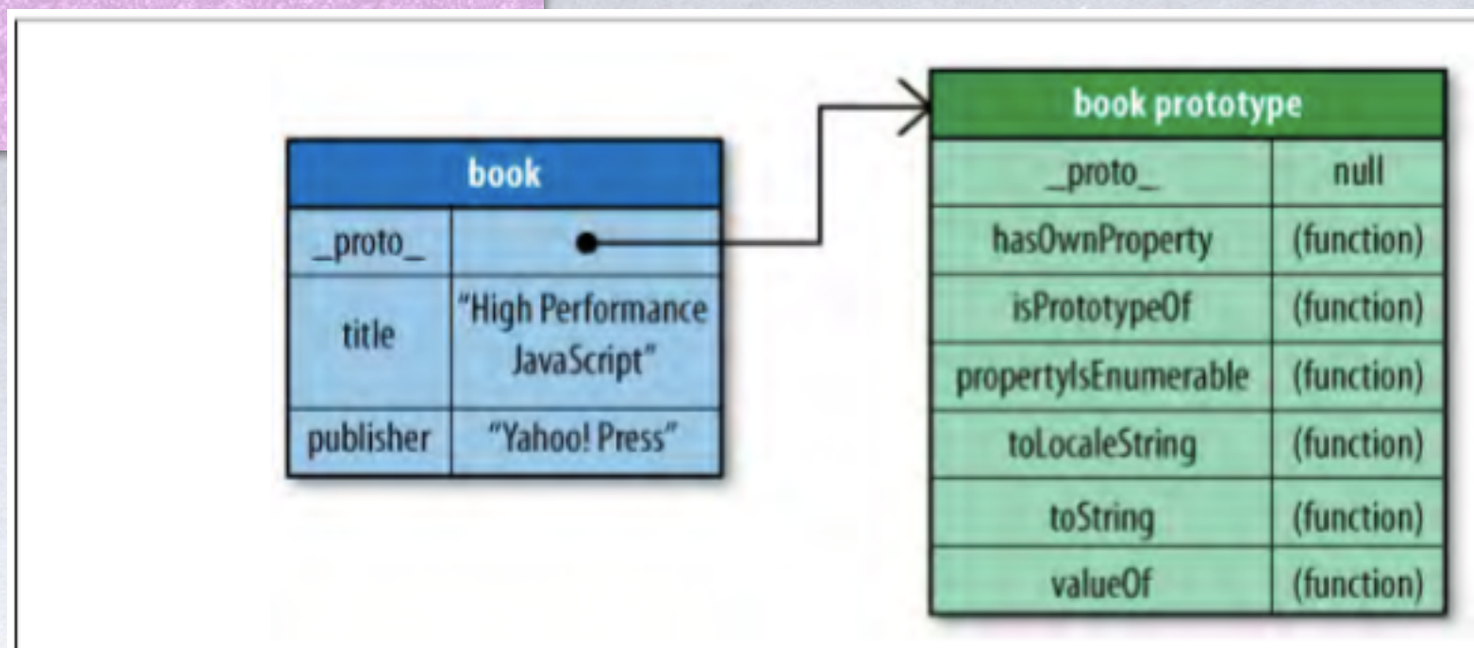
# Executing the closure



# Prototypes

- Relationship between an instance and prototype

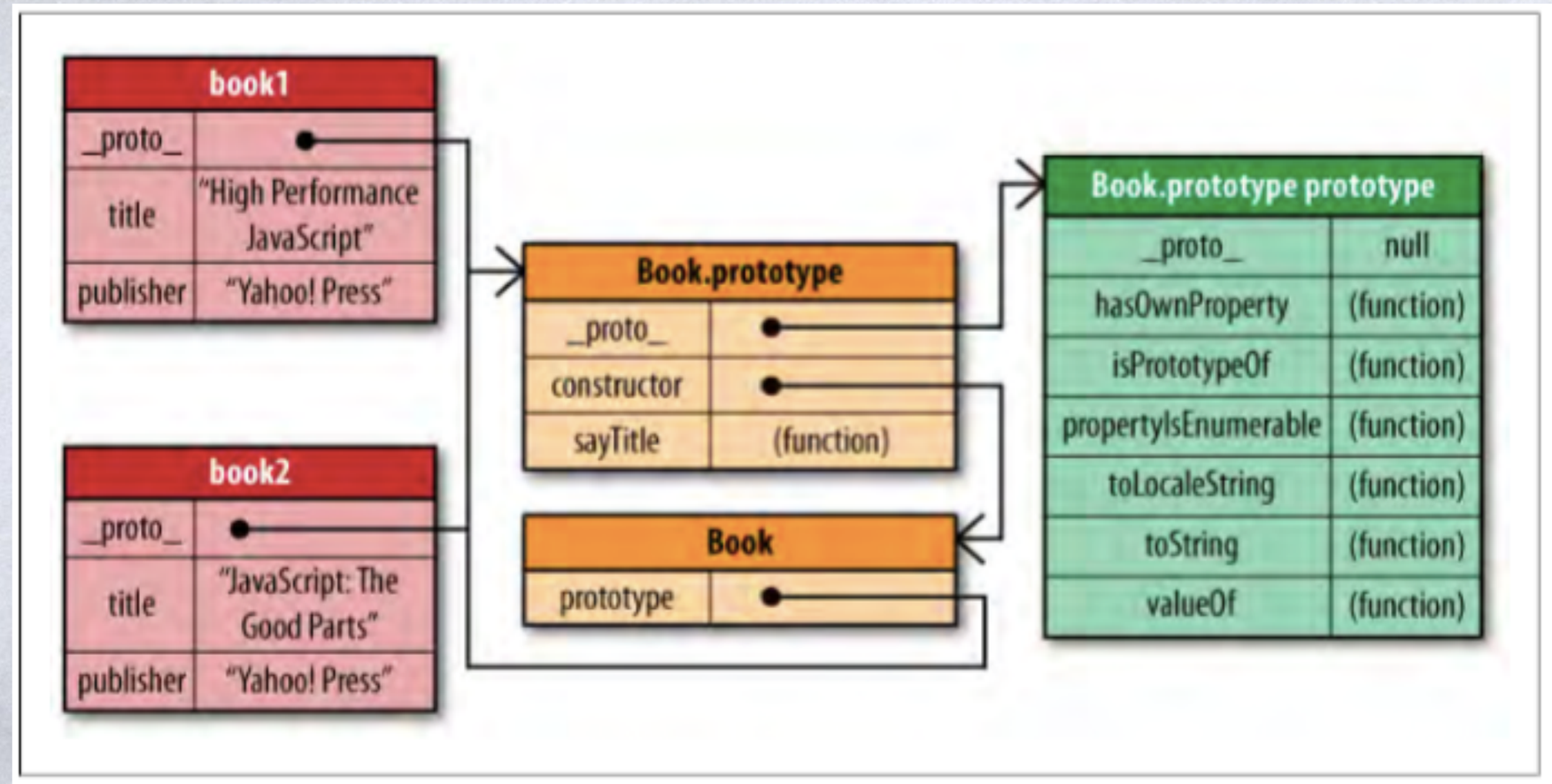
```
var book = {
    title: "High Performance JavaScript",
    publisher: "Yahoo! Press"
};

alert(book.toString()); //"[object
Object]"
```



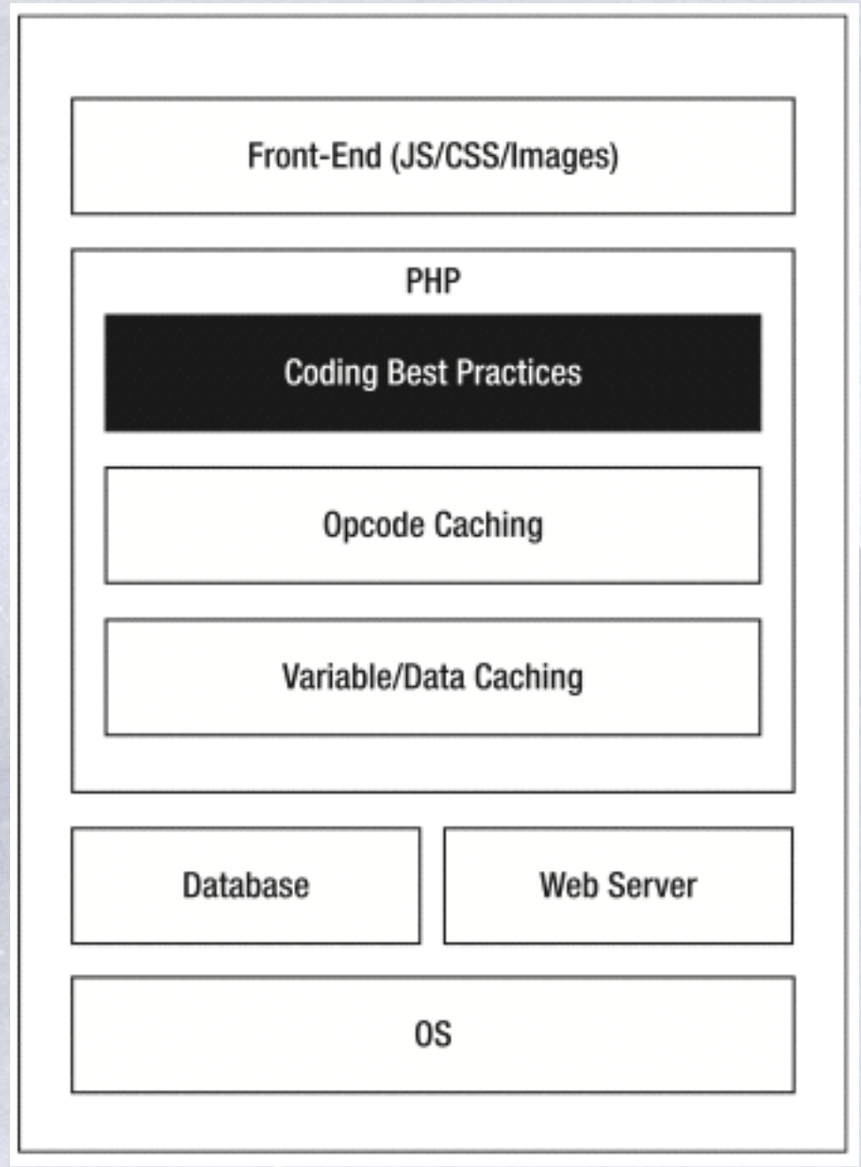
# Prototype Chains - example

```
function Book(title, publisher){
    this.title = title;
    this.publisher = publisher;
}
Book.prototype.sayTitle = function(){
      alert(this.title);
};

var book1 = new Book("High Performance JavaScript", "Yahoo!
Press");
var book2 = new Book("JavaScript: The Good Parts", "Yahoo!
Press");

alert(book1 instanceof Book); //true
alert(book1 instanceof Object); //true

book1.sayTitle(); //"High Performance JavaScript"
alert(book1.toString()); //"[object Object]"
```

# Prototype Chains - Figure



# PHP Code Optimization



# Profiling PHP Code

- APD (Pure profiler)
  - http://pecl.php.net/apd
- DBG (Profiler & Debugger)
  - http://dd.cron.ru/dbg/

# OOP Tweaks

- Always declare your statics! Dynamic methods accessed statically are 50%+ slower.
  - class::attribute/function

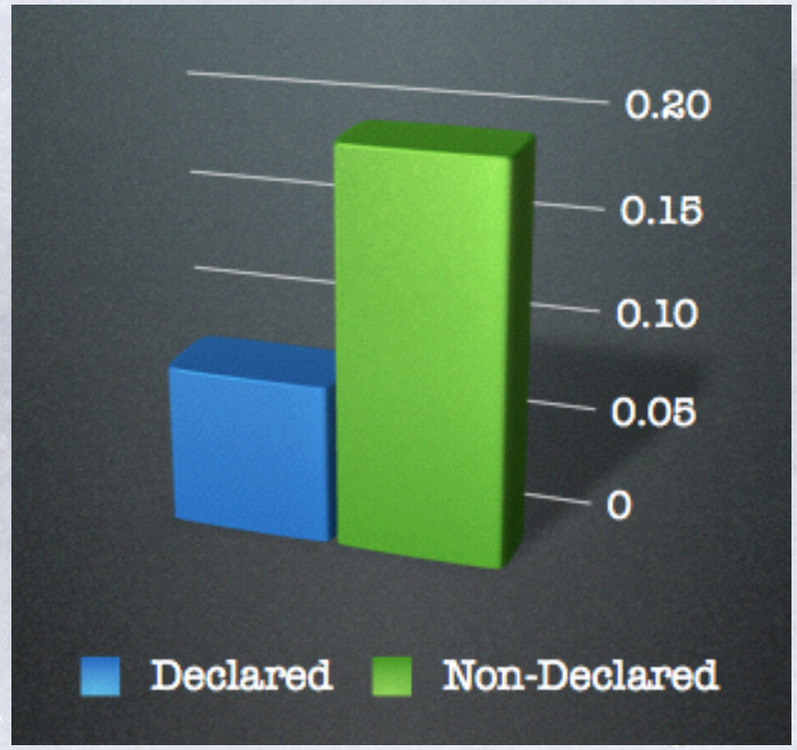
```
protected static function loadExternalFile( $file )
    {
        if ( file_exists( $file ) )
        {
            require( $file );
        } else {
            throw new ezcBaseFileNotFoundException( $file );
        }
    }
```

# Quick Static Benchmark

```php
<?php
 class bench {
      public function a() { return 1; }
      public static function b() { return 1; }
 }
 $s = microtime(1);
 for ($i = 0; $i < 100000; $i++) bench::a();
 $e = microtime(1);
 echo "Dynamic Static Method: ".($e - $s)."\n";
 $s = microtime(1);
 for ($i = 0; $i < 100000; $i++) bench::b();
 $e = microtime(1);
 echo "Declared Static Method: ".($e - $s)."\n";
```

# Conclusion?

- Declaring static methods as was done gives us a 60% speed boost.



# Use Class Constants!

```
class ezcInputForm
 {
      const VALID = 0;
      const INVALID = 1;
      const DEF_NO_ARRAY =1;
      const DEF_EMPTY =2;
      const DEF_ELEMENT_NO_DEFINITION_ELEMENT = 3;
      const DEF_NOT_REQUIRED_OR_OPTIONAL =5;
      const DEF_WRONG_FLAGS_TYPE =6;
      const DEF_UNSUPPORTED_FILTER =7;
      const DEF_FIELD_NAME_BROKEN =8;
}
```

# Advantages

- Parsed at compile time, no execution overhead.
- Faster lookups due to a smaller hash.
- “Namespacing” & shorter hash names.
- Cleaner code speeds up debugging.

# require vs. require_once

```php
<?php
require_once("ClassA.php");
require_once("ClassB.php");
require_once("ClassC.php");
require_once("ClassD.php");
echo "Only testing require_once.";
```

VS.

```php
<?php
require("ClassA.php");
require("ClassB.php");
require("ClassC.php");
require("ClassD.php");
echo "Only testing require.";
```

```php
<?php
class A{
}
class B {
}
class C {
}
class D {
}
```

# Calculating Loop Length in Advance

Un-optimized for Loop

```
<?php
$items = array(1,2,3,4,5,6,7,8,9,10);
for($i=0; $i<count($items); $i++)
{
    $x = 10031981 * $i;
    }
```

Optimized for Loop

```
<?php
$items = array(1,2,3,4,5,6,7,8,9,10);
$count = count($items);
for($i=0; $i<$count; $i++) {
$x = 10031981 * $i;
}
```

# Accessing Array Elements Using foreach vs. for vs. while

- using a foreach loop instead of either a while or for loop

# File Access

- Using fread() for small file data access
- file_get_contents for large files

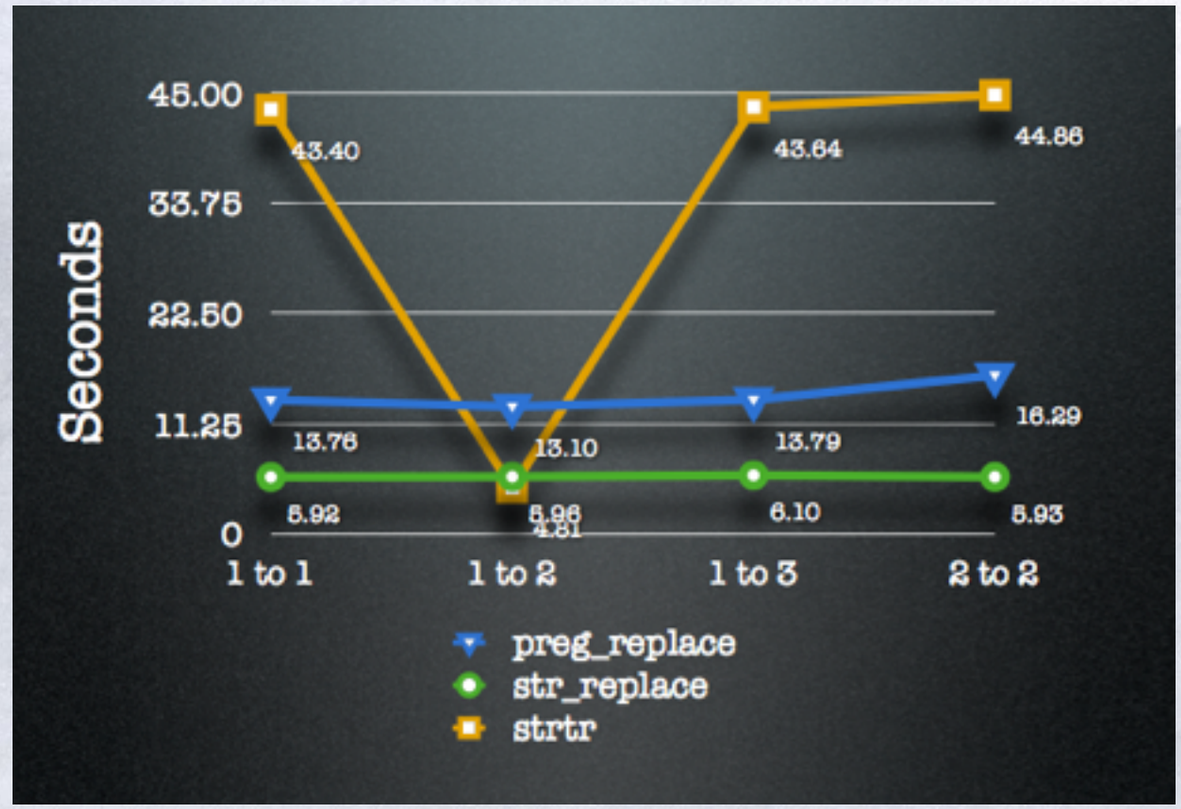
# Faster Access to Object Properties

```php
<?php
class Person
{
    private $_gender = NULL;
    private $_age = NULL;
    public function getGender() {
        return $this->_gender;
     }
    public function getAge() {
        return $this->_age;
        }
    public function setGender($gender) {
        $this->_gender = $gender;
    }
    public function setAge($age) {
        $this->_age = $age; }
}

$personObj = new Person();
$start = microtime();
for($i=0; $i<100000; $i++) {
    $personObj->getGender();
}
echo microtime()-$start.'ms';
```

```php
<?php
class Person
{
    public $_gender = NULL;
    public $_age = NULL;
}

$personObj = new Person();
$start = microtime();
for($i=0; $i<100000; $i++) {
    //Average: 0.0205617ms
    $personObj->_gender;
    }

echo microtime()-$start.'ms';
```

# PCRE is slow

```
<?php
preg_replace( "/\n/", "\\n", $text);
str_replace( "/\n/", "\\n", $text);

preg_match(“/^foo_/i", "FoO_")
!strncasecmp("foo_", "FoO_", 4)

preg_match(“/[abce]/", “string")
strpbrk(“string", “abcd")

preg_match("!string!i", "text")
stripos("text", "string")
?>
```



# When Not to Use Regular Expressions

```
endsWithSemicolon = /;$/.test(str);

—>

endsWithSemicolon = str.charAt(str.length - 1) == ";";
```

# More Ways to Improve Regular Expression Efficiency

- Start regexes with simple, required tokens
  - Good starting tokens: anchors (^ or $), specific characters (x), character classes ([a-z] or \d), and word boundaries (\b).
  - avoid starting regexes with groupings or optional tokens, and avoid top-level alternation such as /one|two/.
- Reduce the amount and reach of alternation
  - cat|bat —> [cb]at
  - red|read —> rea?d

- Expose required tokens
  - /(^ab|^cd)/ ——> / ^(ab|cd)/
- Reuse regexes by assigning them to variables
- Split complex regexes into simpler pieces

```
while (/regex1/.test(str1)) {
        /regex2/.exec(str2);
          ...
}
Do this instead:
var regex1 = /regex1/,
    regex2 = /regex2/;
while (regex1.test(str1)) {
    regex2.exec(str2);
    ...
}
```

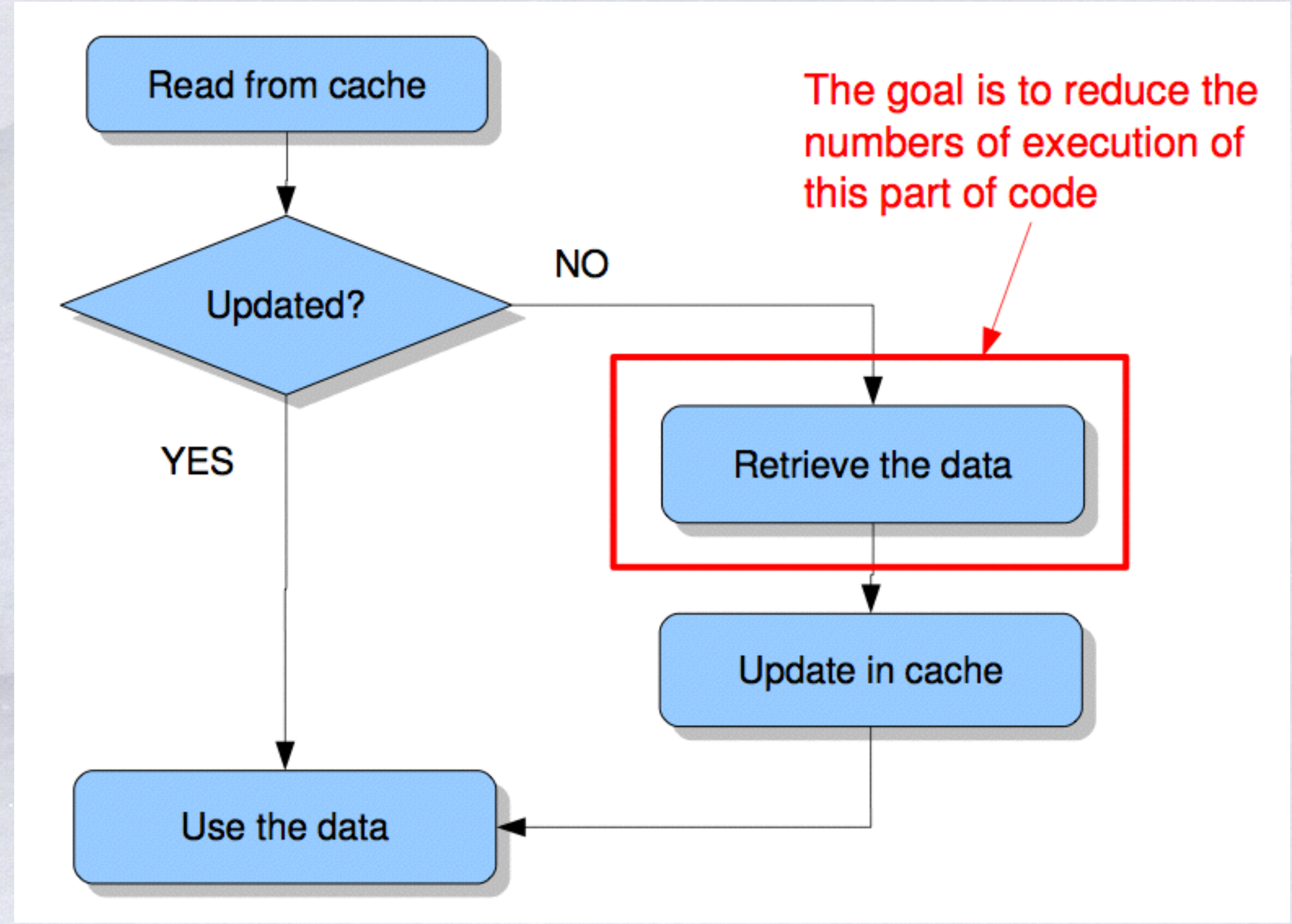
# Looking Under the Hood

- VLD
  - Reviewing Opcode Functions
- strace
  - C-level Tracing
- Xdebug
  - Memory consumption of your PHP script
  - Total number of calls made to a function
  - Total time spent within the function
  - Complete stack trace for a function

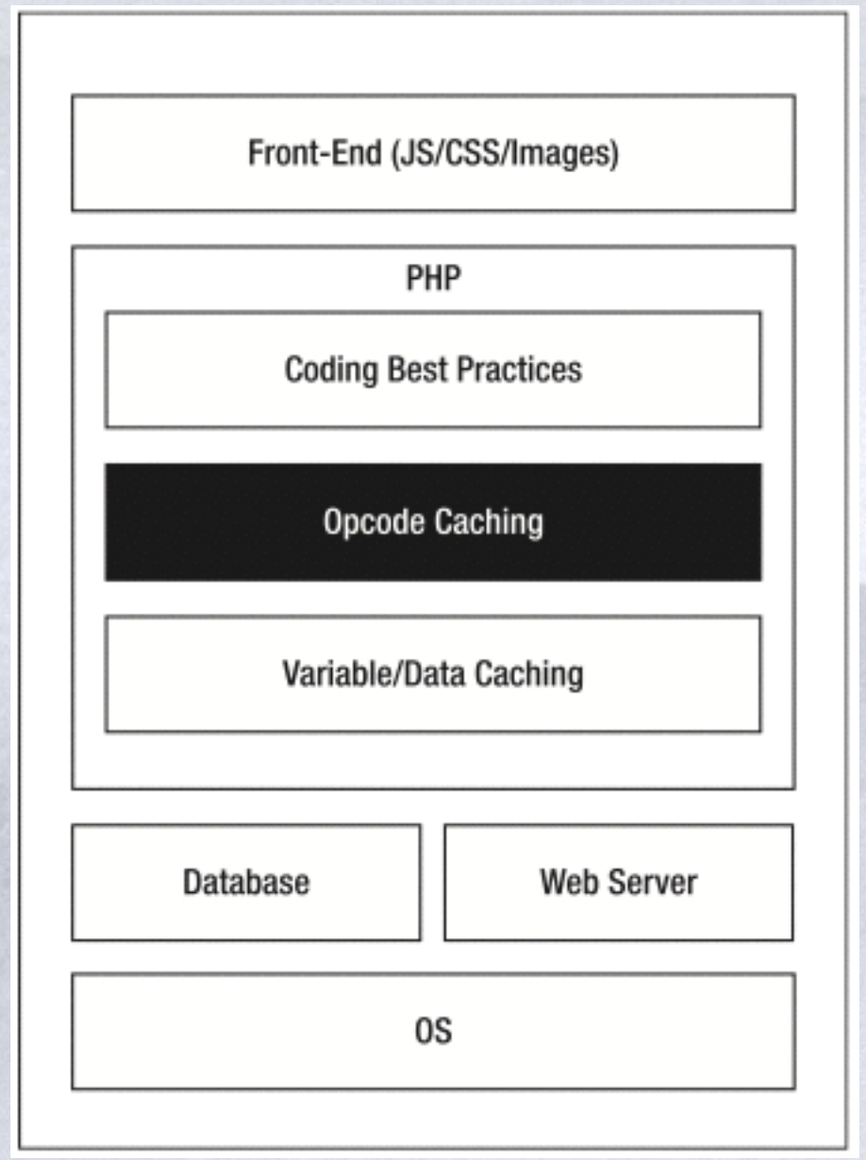
# Caching in PHP

- In PHP we can use different kinds of cache systems:
  - File, by hand using the fwrite, fread, readfile, etc
  - Pear::Cache_Lite, using the files as cache storage
  - APC, using the apc extension (apc_add, apc_fetch, etc)
  - Xcache, from the lighttpd web server project
  - Memcached, using the memcached extension (Memcached::add, Memcached::get, etc)
  - Zend Server/CE, using Zend Server API (zend_shm_cache_fetch, zend_shm_cache_store, etc)

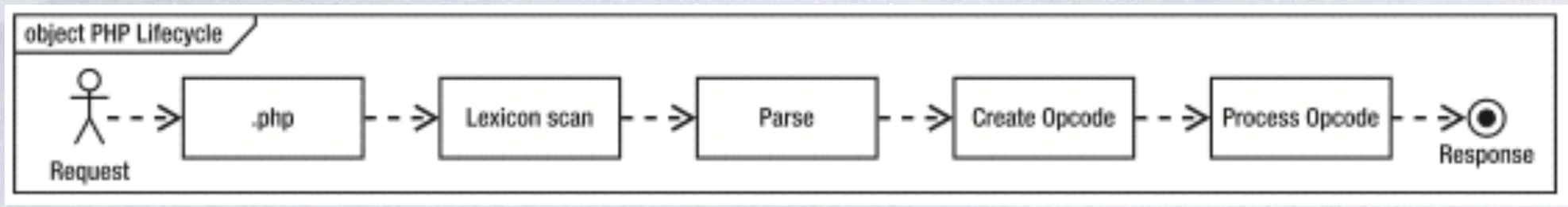
# Conditional execution

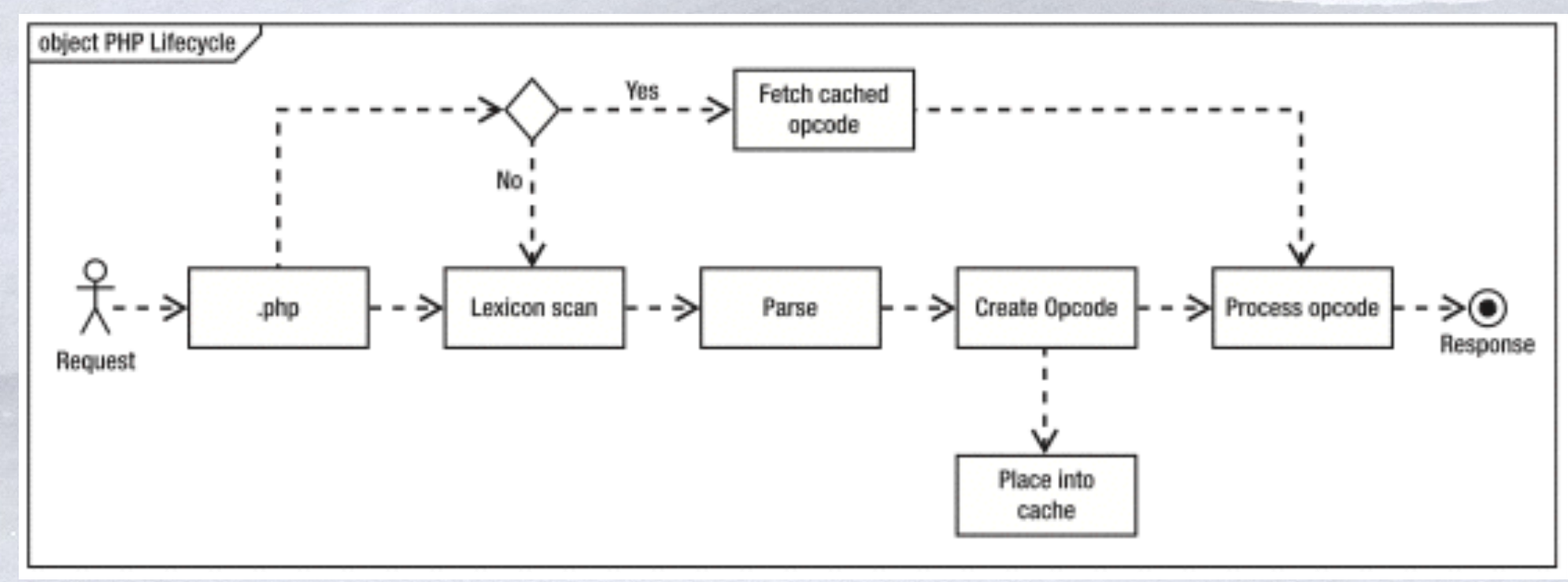


# Opcode Caching



# The PHP Life Cycle



# Alternative PHP Cache(APC)

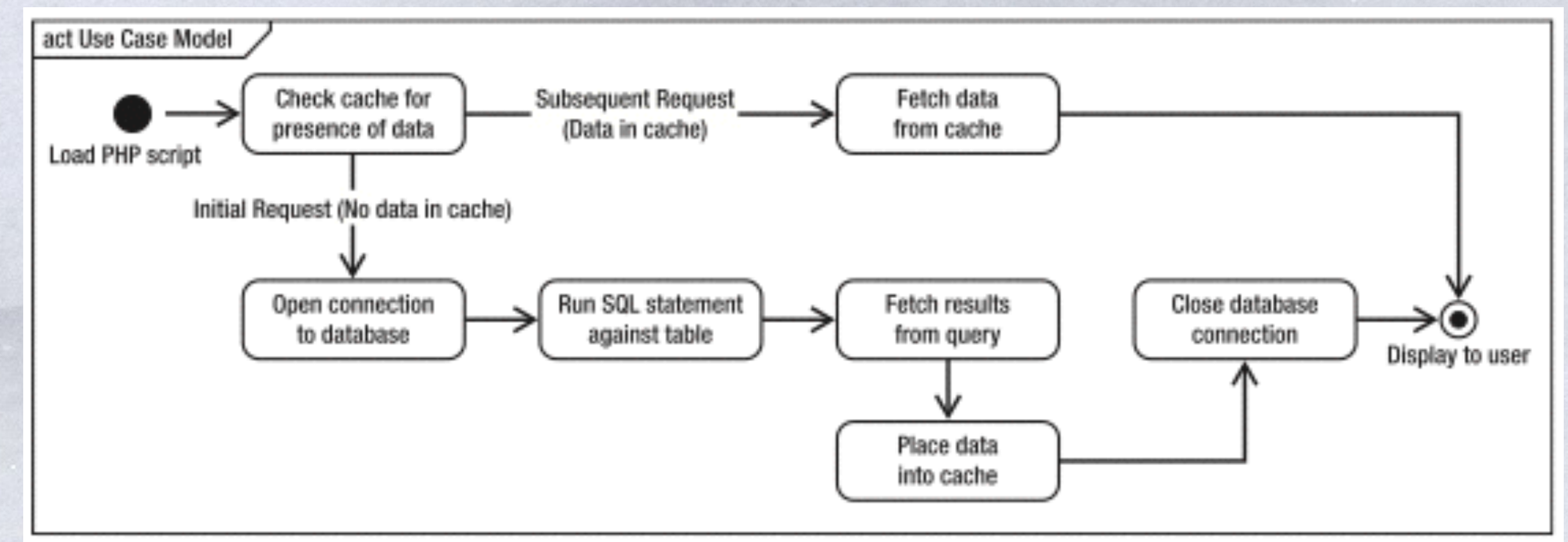
- A free, open source and robust framework for caching and optimizing PHP intermediate code.
- maintained by core PHP developers
- users
  - Yahoo!
  - Facebook
  - Wikipedia and many more

# XCache

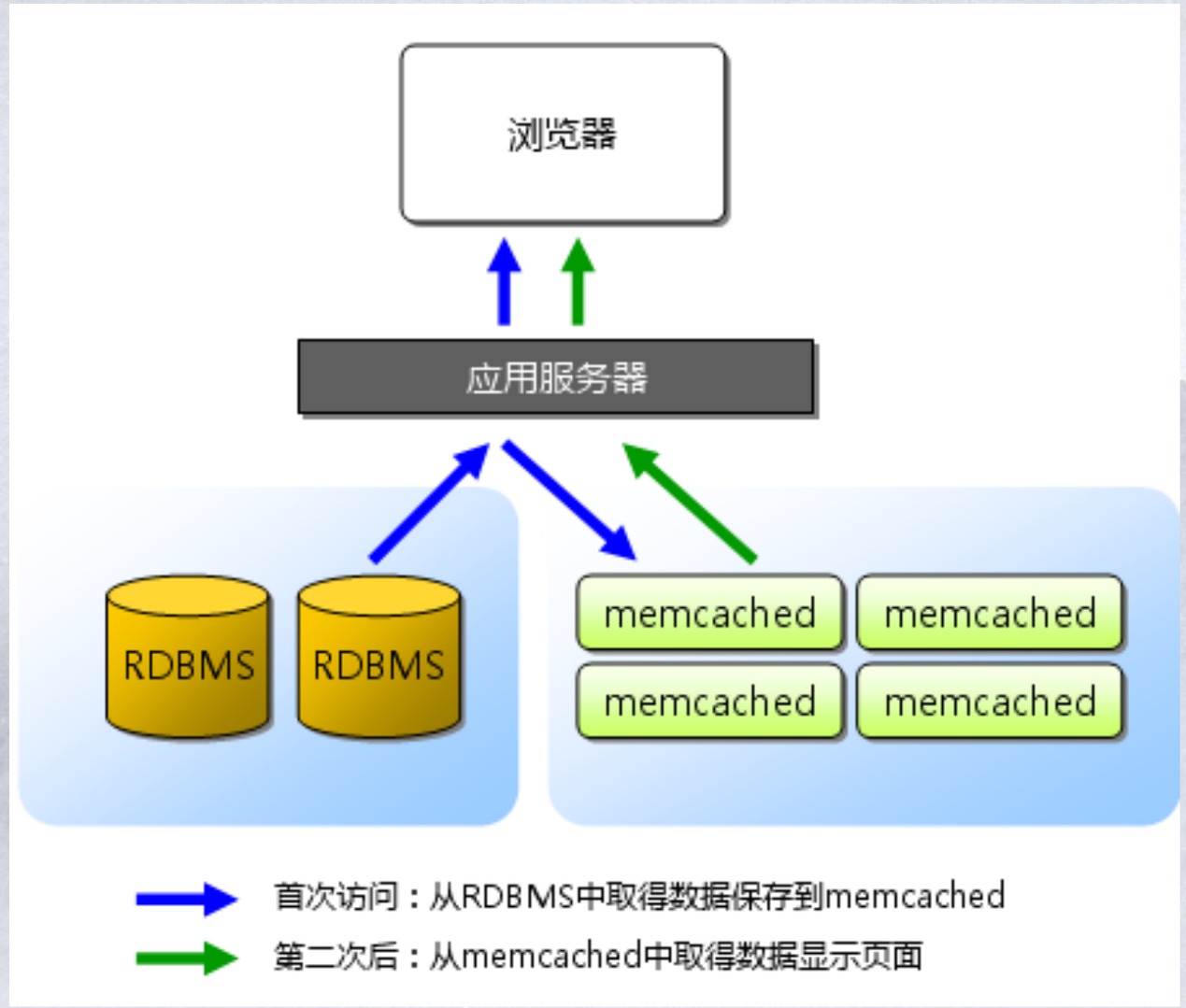
- another Opcode caching tool that is used by PHP. XCache, like APC, uses shared memory to store the Opcode and uses this cached Opcode to satisfy a request for a PHP script.
- Like APC, XCache is also available for both Unix-based systems and Windows.

# Variable Caching

## The Value of Implementing Variable Caching

# Memcached

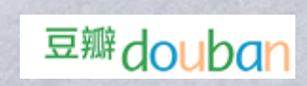

浏览器
应用服务器
RDBMS
RDBMS
memcached
memcached
memcached
memcached
首次访问：从RDBMS中取得数据保存到memcached
第二次后：从memcached中取得数据显示页面

# Users of Memcached



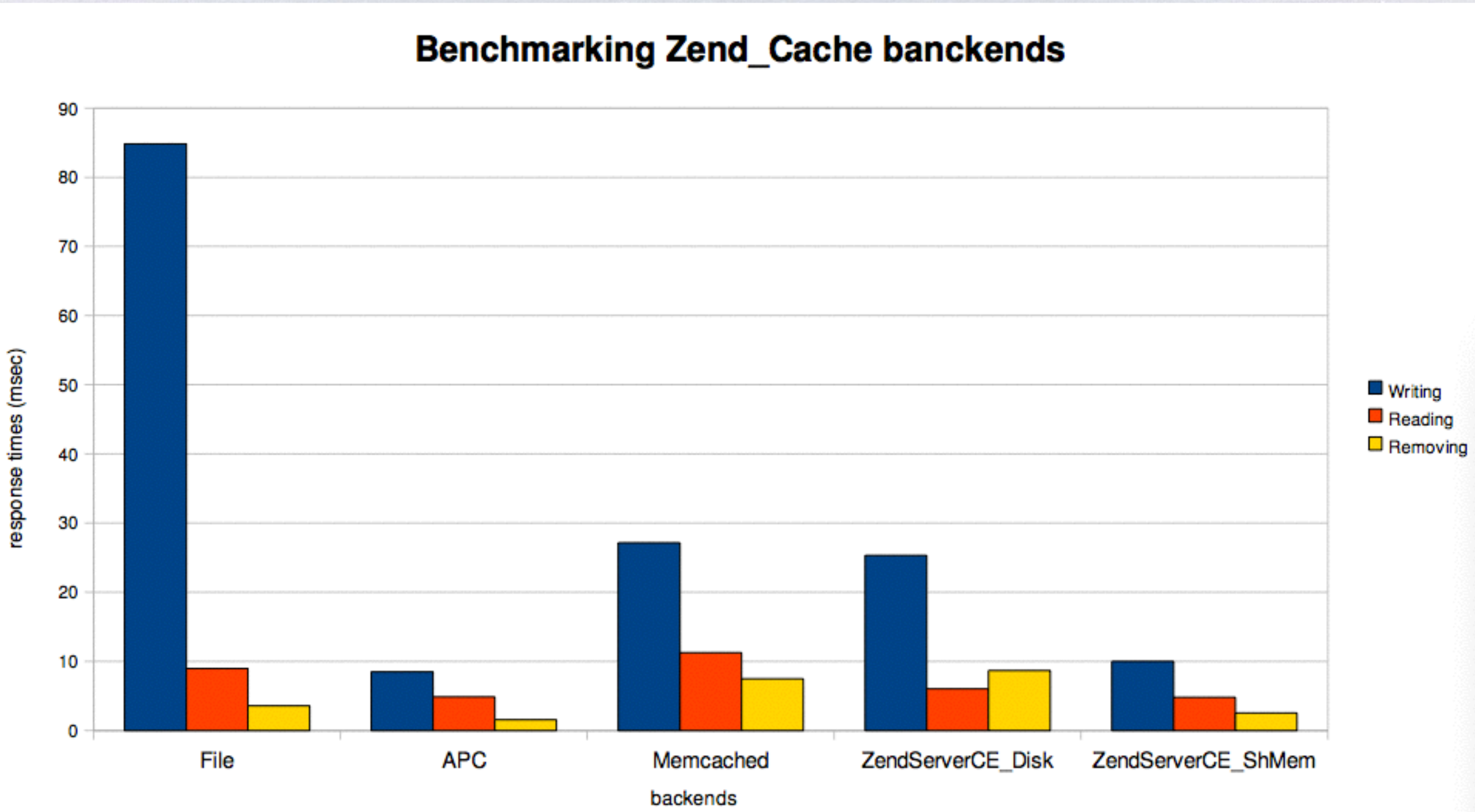




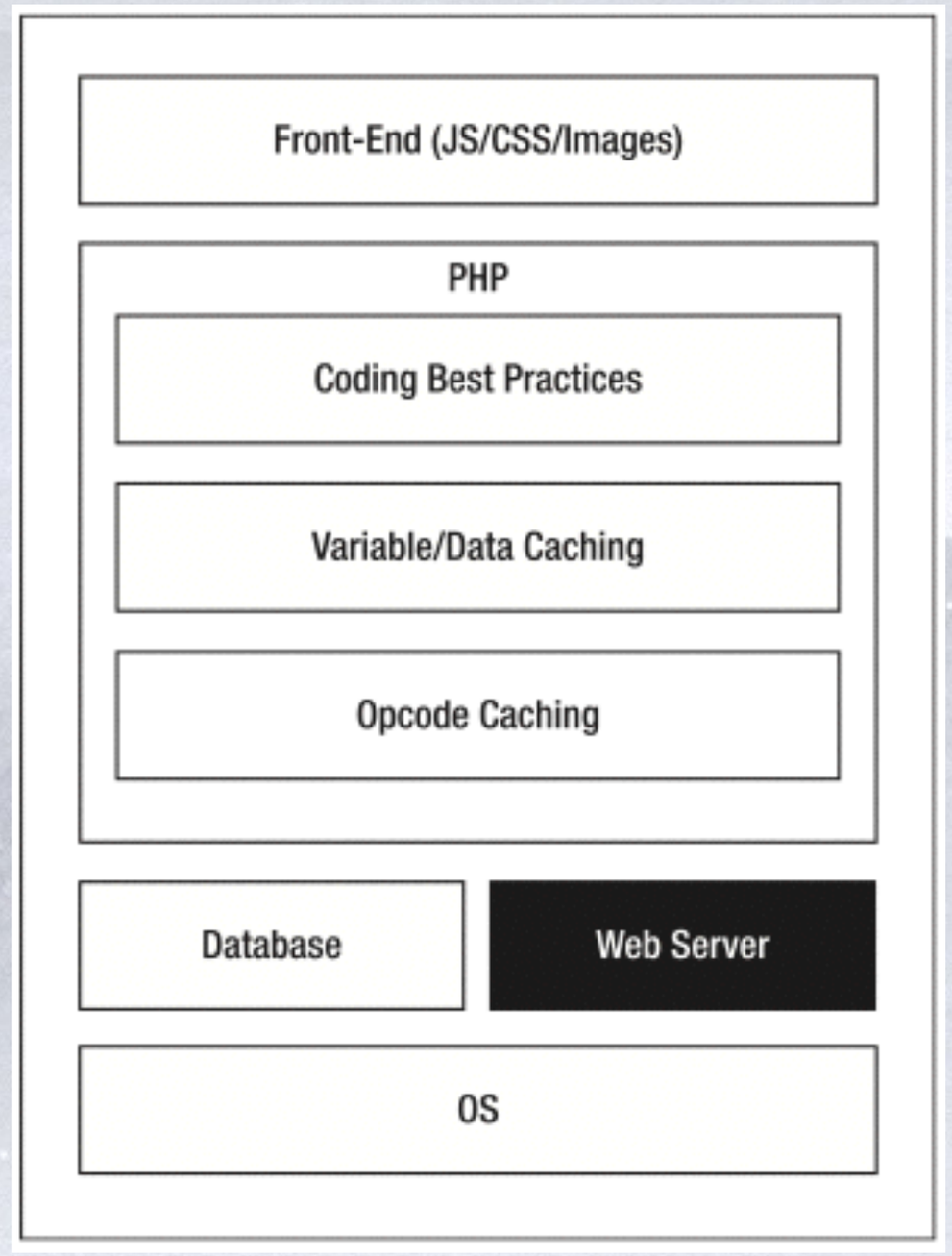








Benchmarking Zend_Cache banckends
response times (msec)
90
80
70
60
50
40
30
20
10
0
File
APC
Memcached
ZendServerCE_Disk
ZendServerCE_ShMem
backends
Writing
Reading
Removing

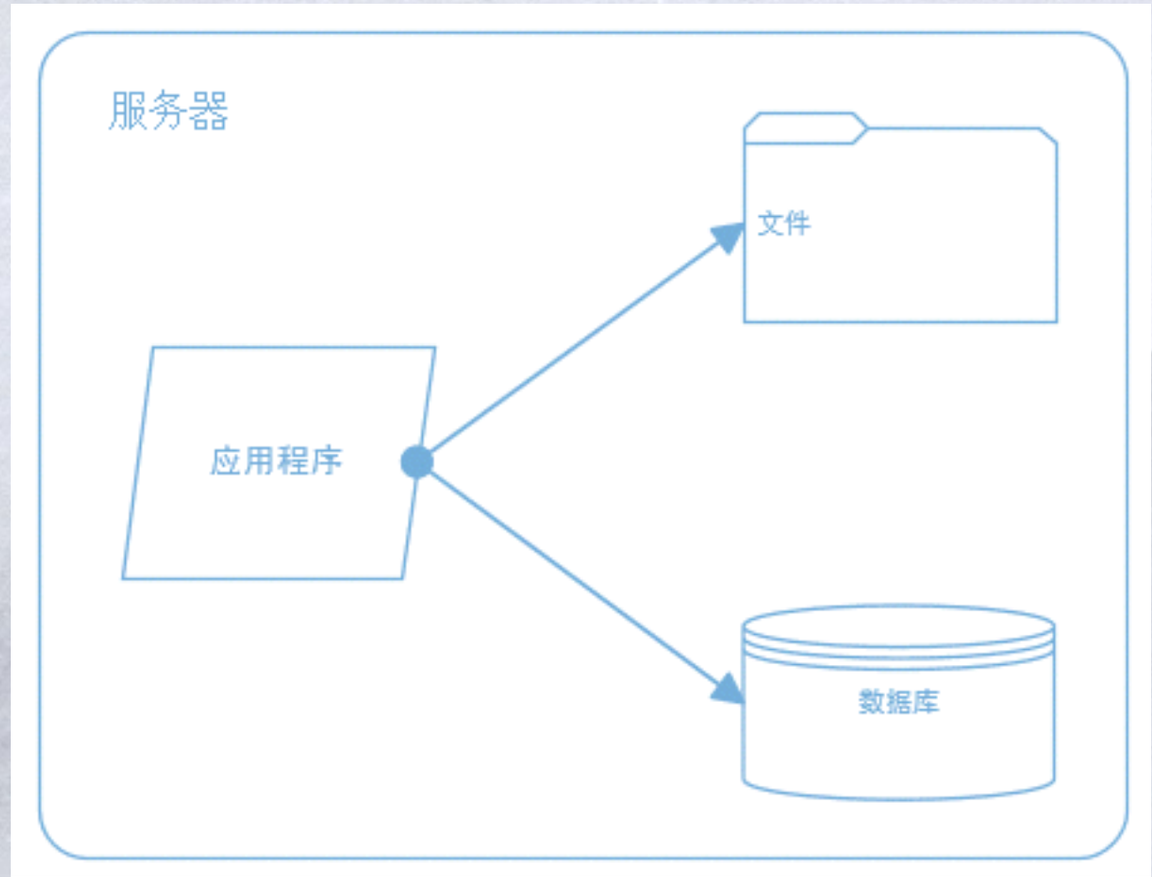
# Choosing the Right Web Server



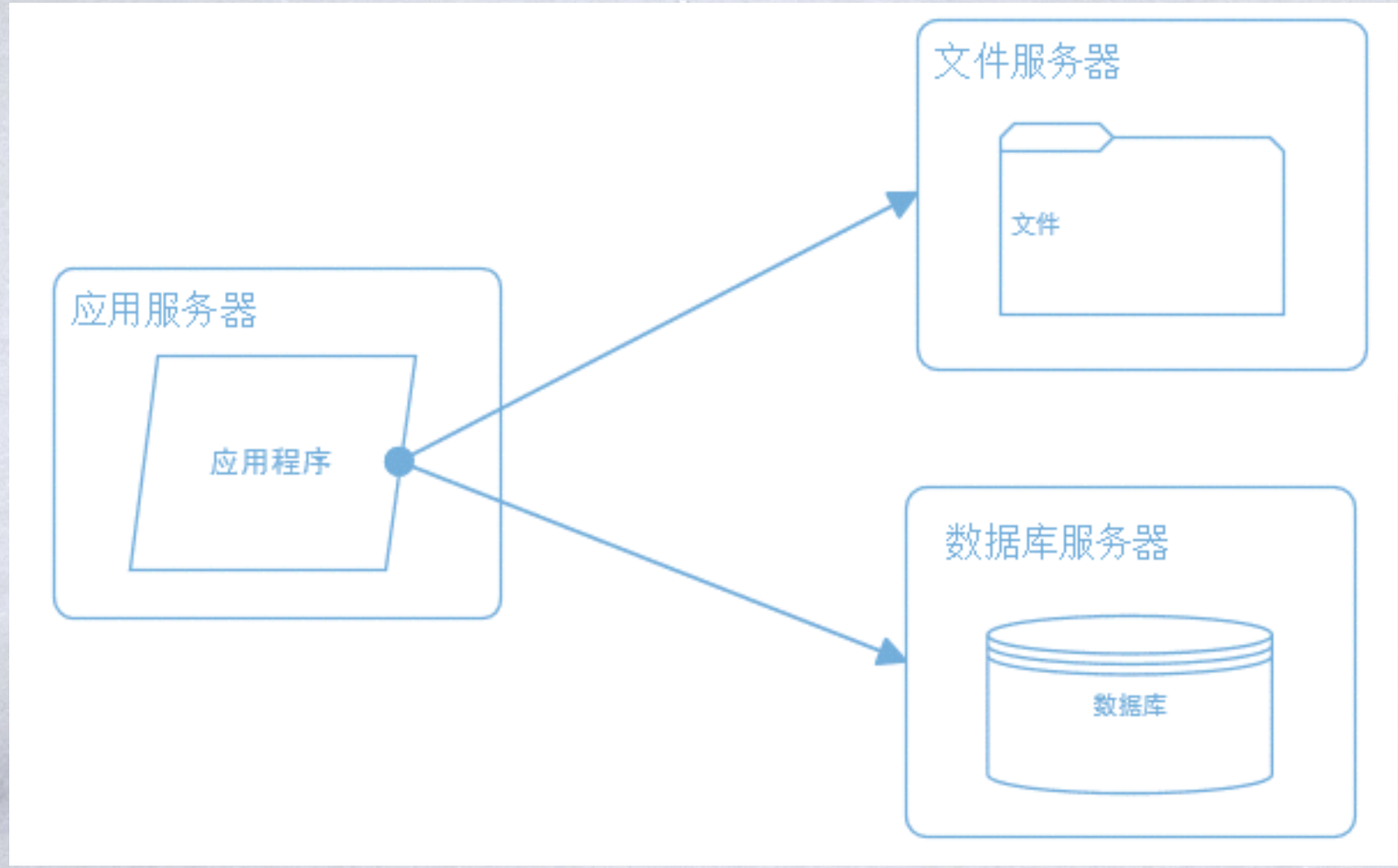
# Outline

- Web Application Performance
- Large Scale Web Application

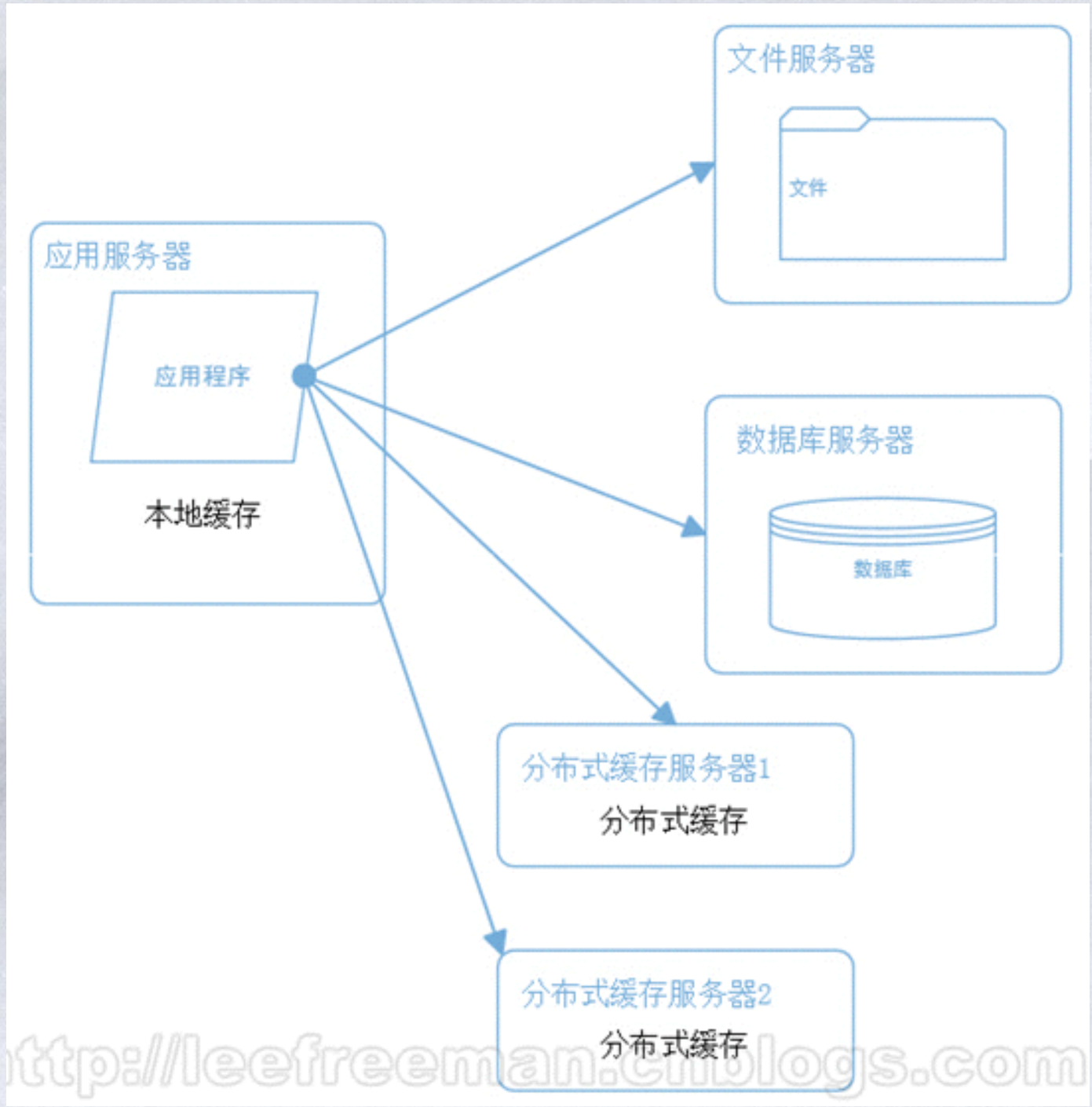
# 一、最开始的网站架构



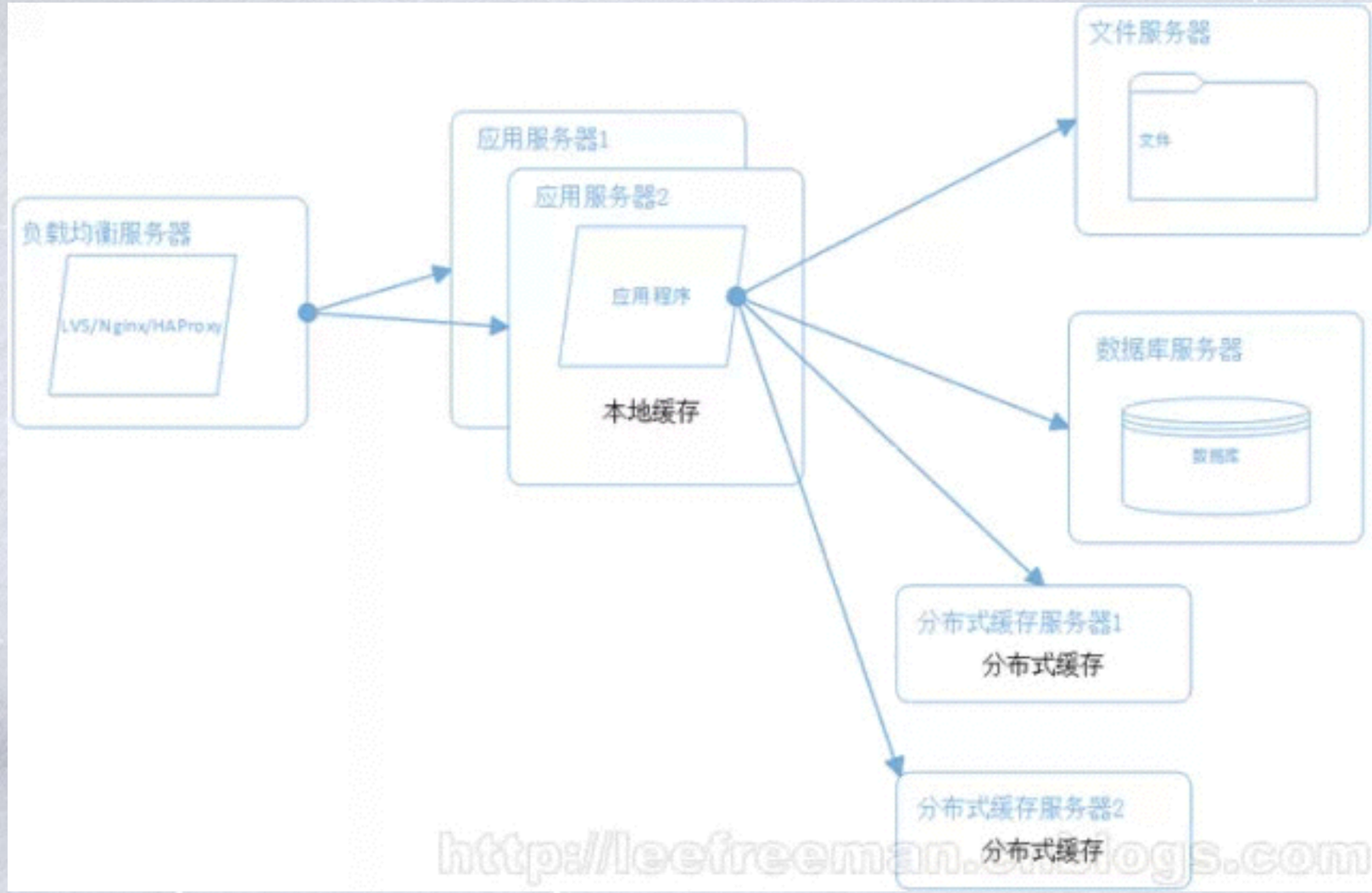
# 二、应用、数据、文件分离



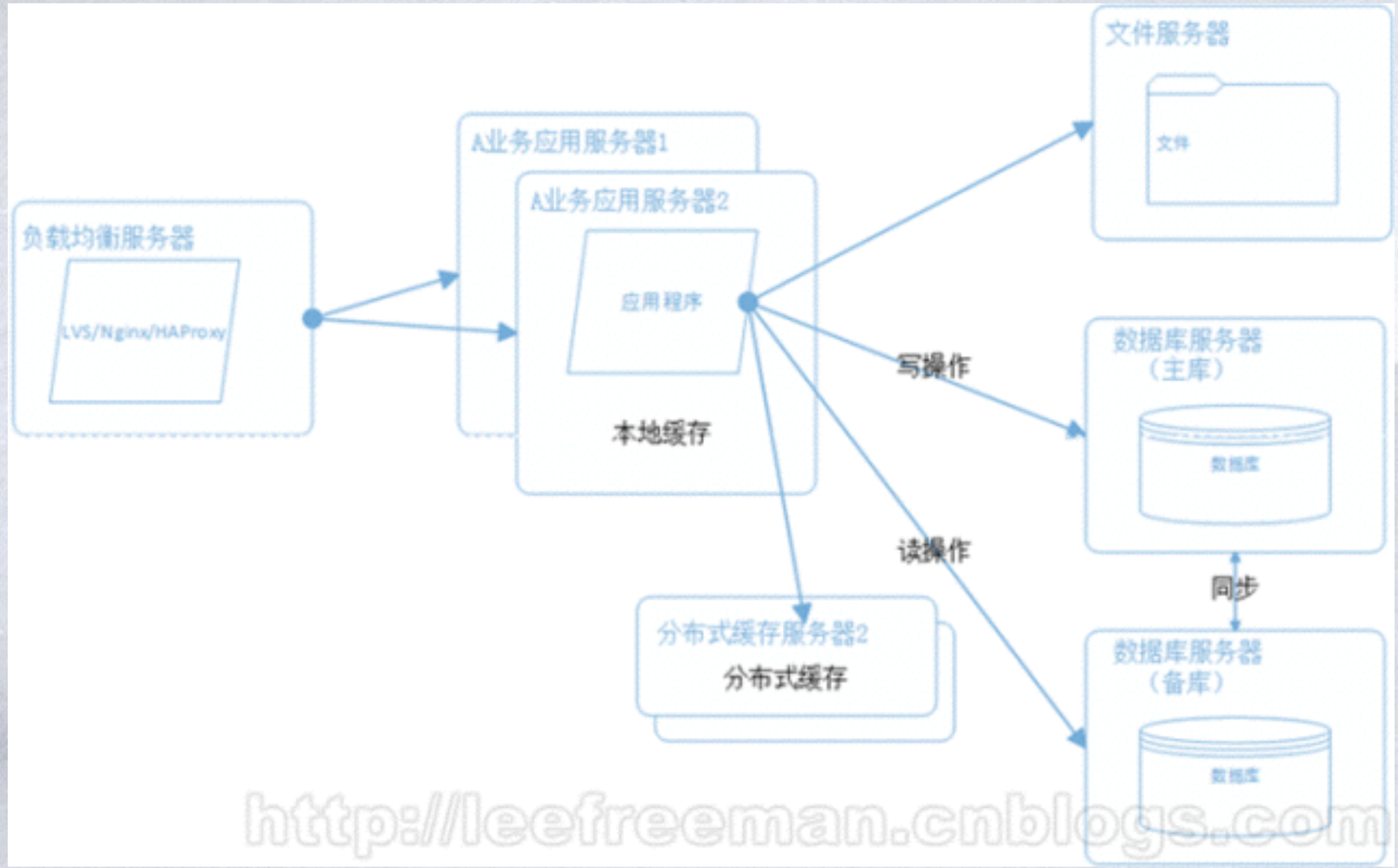
# 三、利用缓存改善网站性能



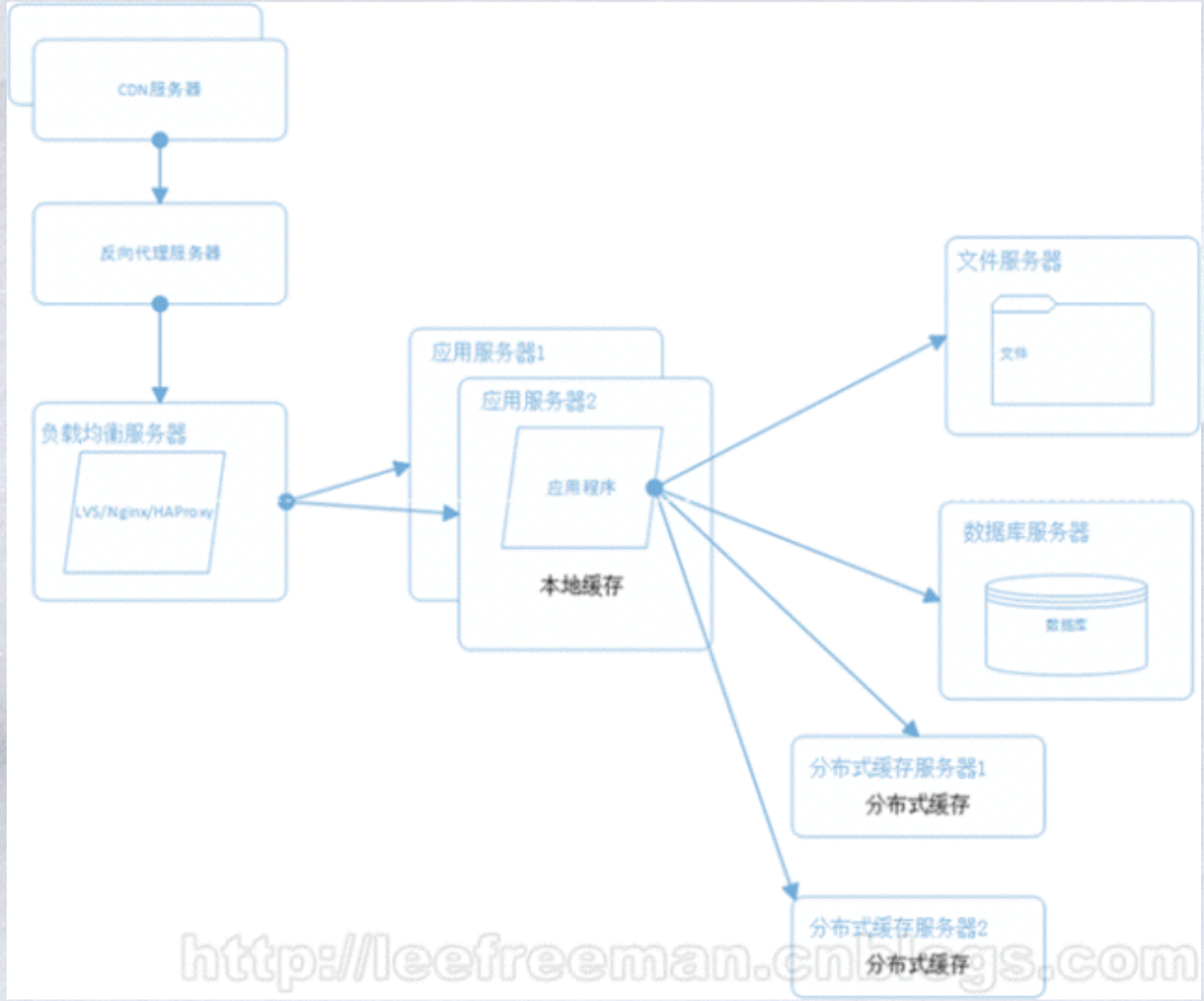
# 四、使用集群改善应用服务器性能



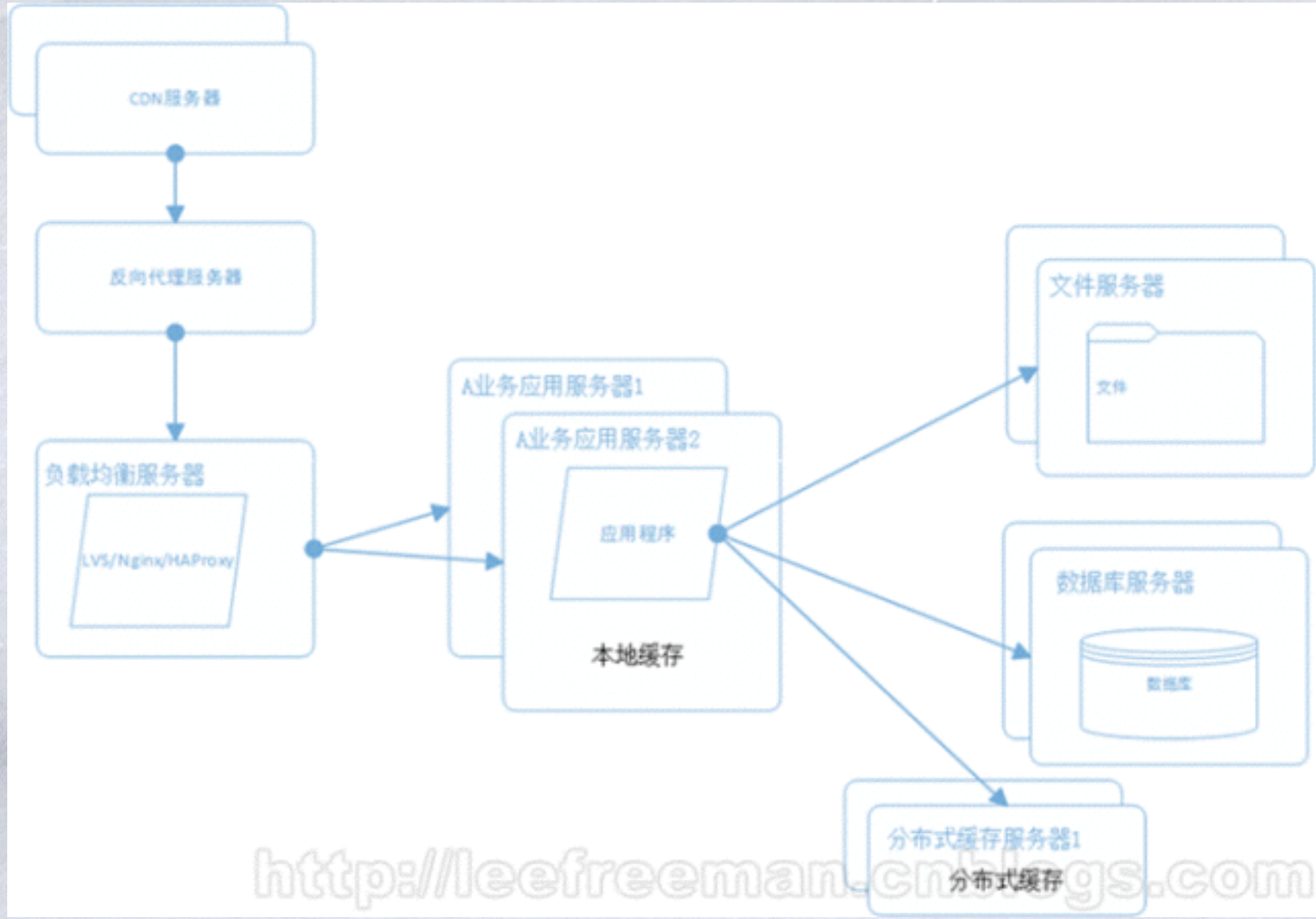
# 五、数据库读写分离和分库分表



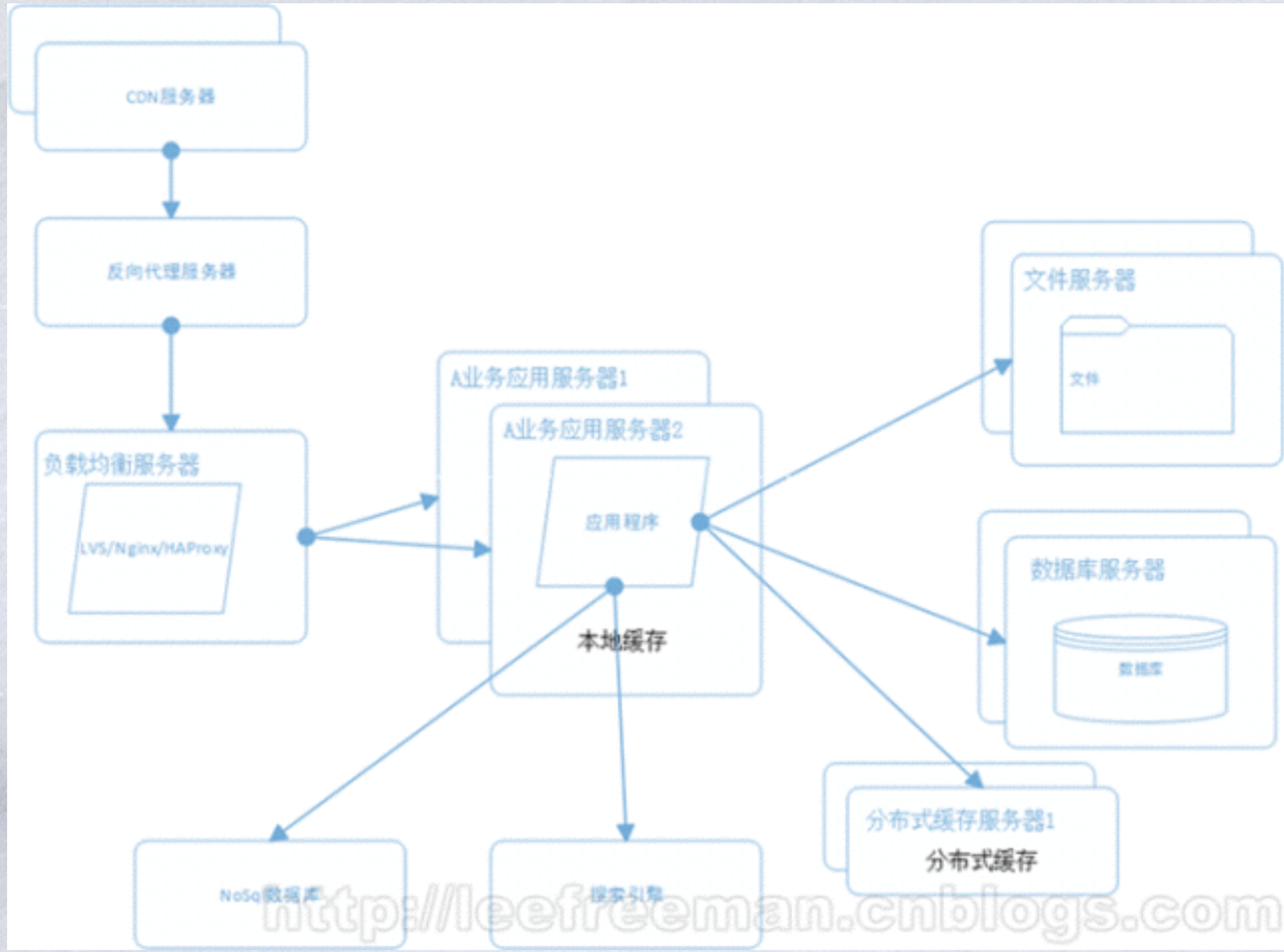
# 六、使用CDN和反向代理提高网站性能



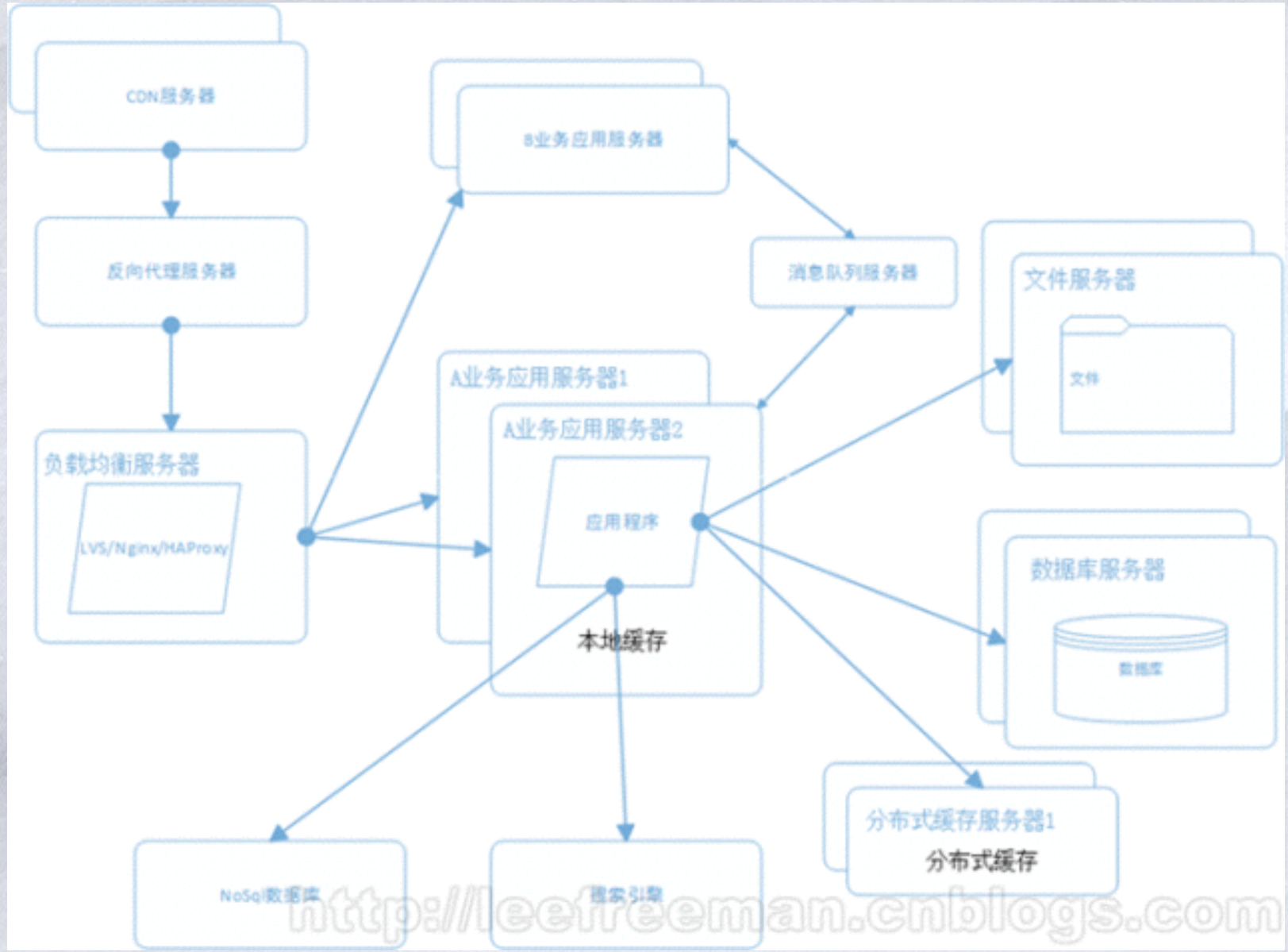
# 七、使用分布式文件系统


CDN服务器
反向代理服务器
负载均衡服务器
LVS/Nginx/HAProxy
A业务应用服务器1
A业务应用服务器2
应用程序
本地缓存
文件服务器
文件
数据库服务器
数据库
分布式缓存服务器1
分布式缓存
http://leefreeman.cnblogs.com

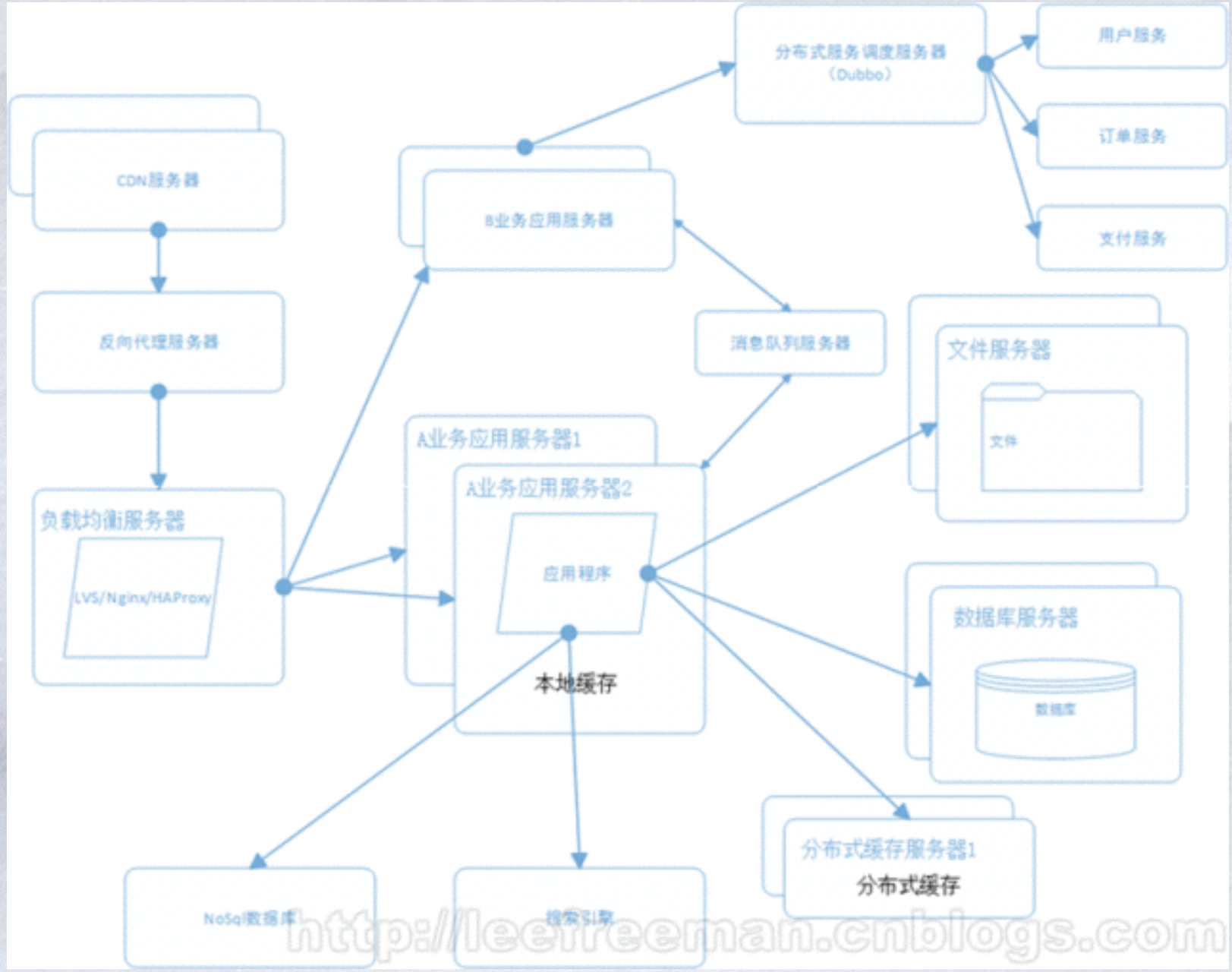
# 八、使用NoSql和搜索引擎



# 九、将应用服务器进行业务拆分



# 十、搭建分布式服务


分布式服务调度服务器
（Dubbo）
用户服务
订单服务
支付服务
CDN服务器
反向代理服务器
负载均衡服务器
LVS/Nginx/HAProxy
B业务应用服务器
消息队列服务器
A业务应用服务器1
A业务应用服务器2
应用程序
本地缓存
文件服务器
文件
数据库服务器
数据库
分布式缓存服务器1
分布式缓存
NoSql数据库
搜索引擎
http://leefreeman.cnblogs.com

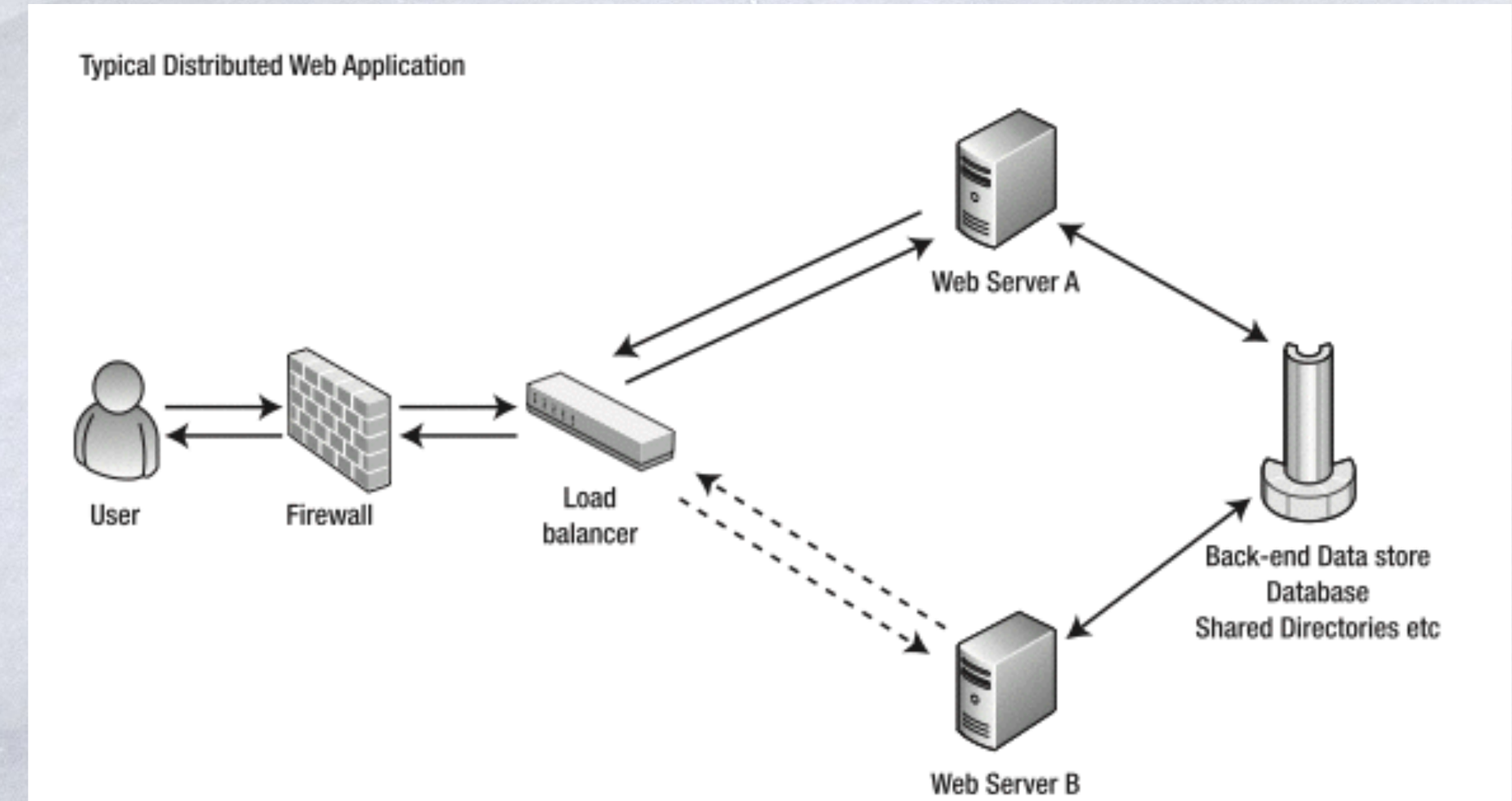
# Static Content Serving

- While Apache is great for dynamic requests, static requests can be served WAY FASTER by other web servers.
  - lighttpd
  - Boa
  - Tux
  - thttpd
- For static requests these servers are easily 300-400% faster then Apache 1 or 2.

# Scaling Beyond a Single Server

- Using Round-Robin DNS
- Using a Load Balancer
- Using Direct Server Return
- Sharing Assets with a Content Distribution Network

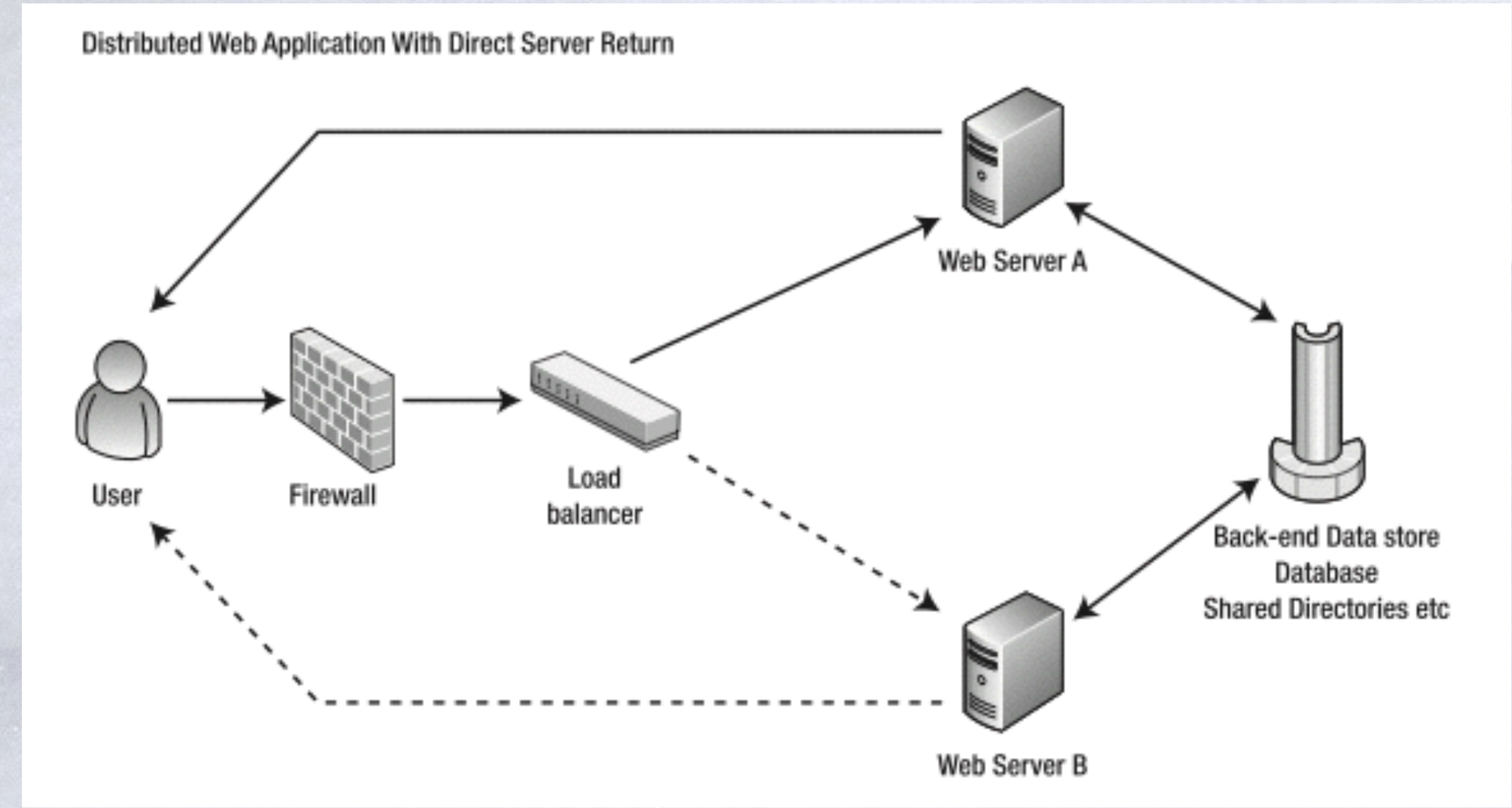
# Using a Load Balancer



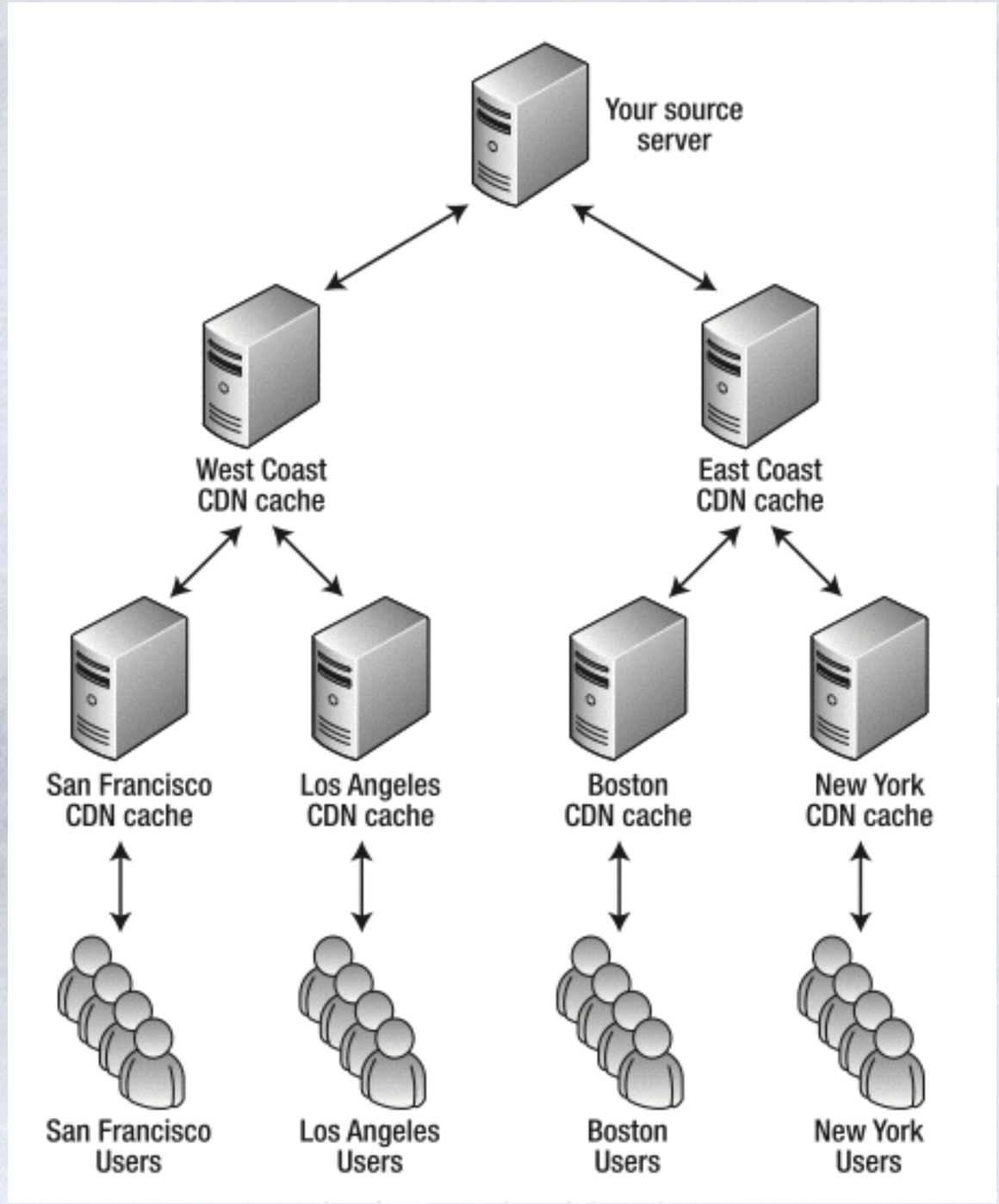
# Categories of Load Balancers

- There are many kinds of load balancers available, both hardware- and software-based. Generally load balancers fall into four categories.
  - Totally software-based solutions
  - Software solutions using a separate load balancing server
  - Physical load balancing appliances
  - Load balancing services

# Using Direct Server Return



# Sharing Assets with a Content Distribution Network



# References

- Pro PHP Application Performance
- 大规模Web服务开发技术
- http://talks.php.net/show/digg/
- 构建高性能web站点

# Tools to Help Measure Performance

- Google PageSpeed
  https://developers.google.com/speed/pagespeed/
- Yahoo YSlow
  http://developer.yahoo.com/yslow/
- Pingdom
  http://www.pingdom.com/
- Web Developer Checklist
  http://webdevchecklist.com/

# Thanks!!!