Improve WordPress performance with caching and deferred execution of code

Danilo Ercoli

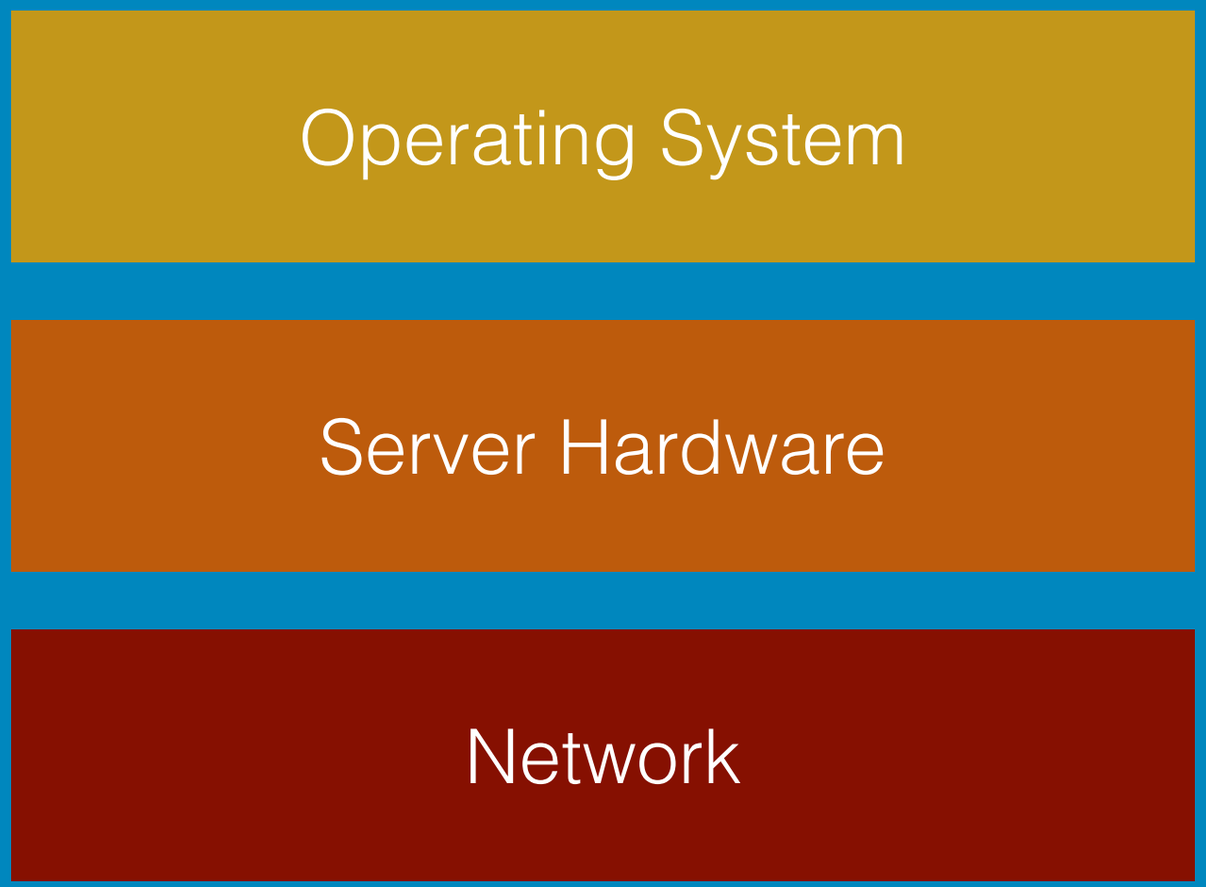
Software Engineer

http://daniloercoli.com

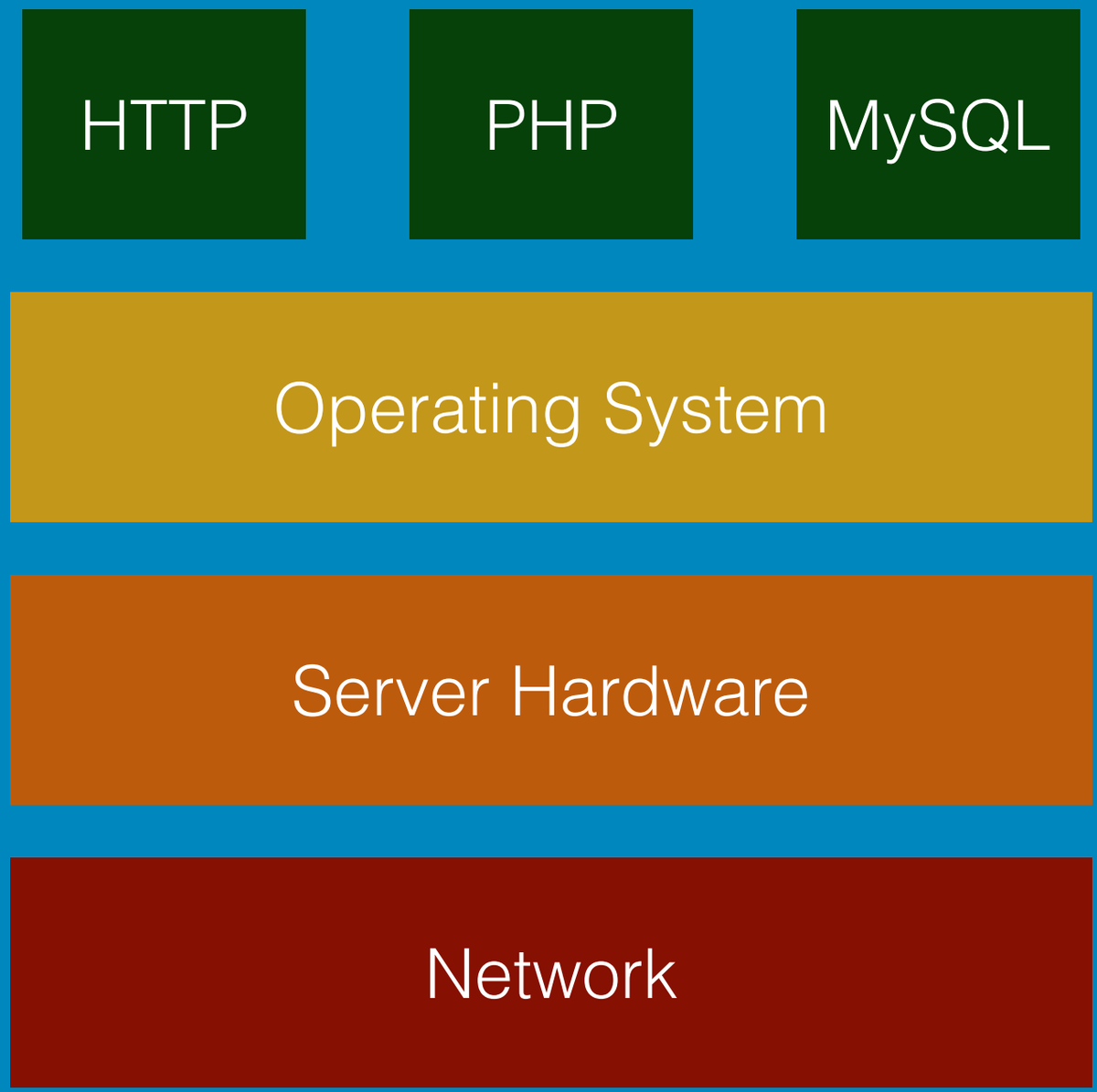
Agenda

- PHP Caching
- WordPress Page Caching
- WordPress Object Caching
- Deferred Execution of code

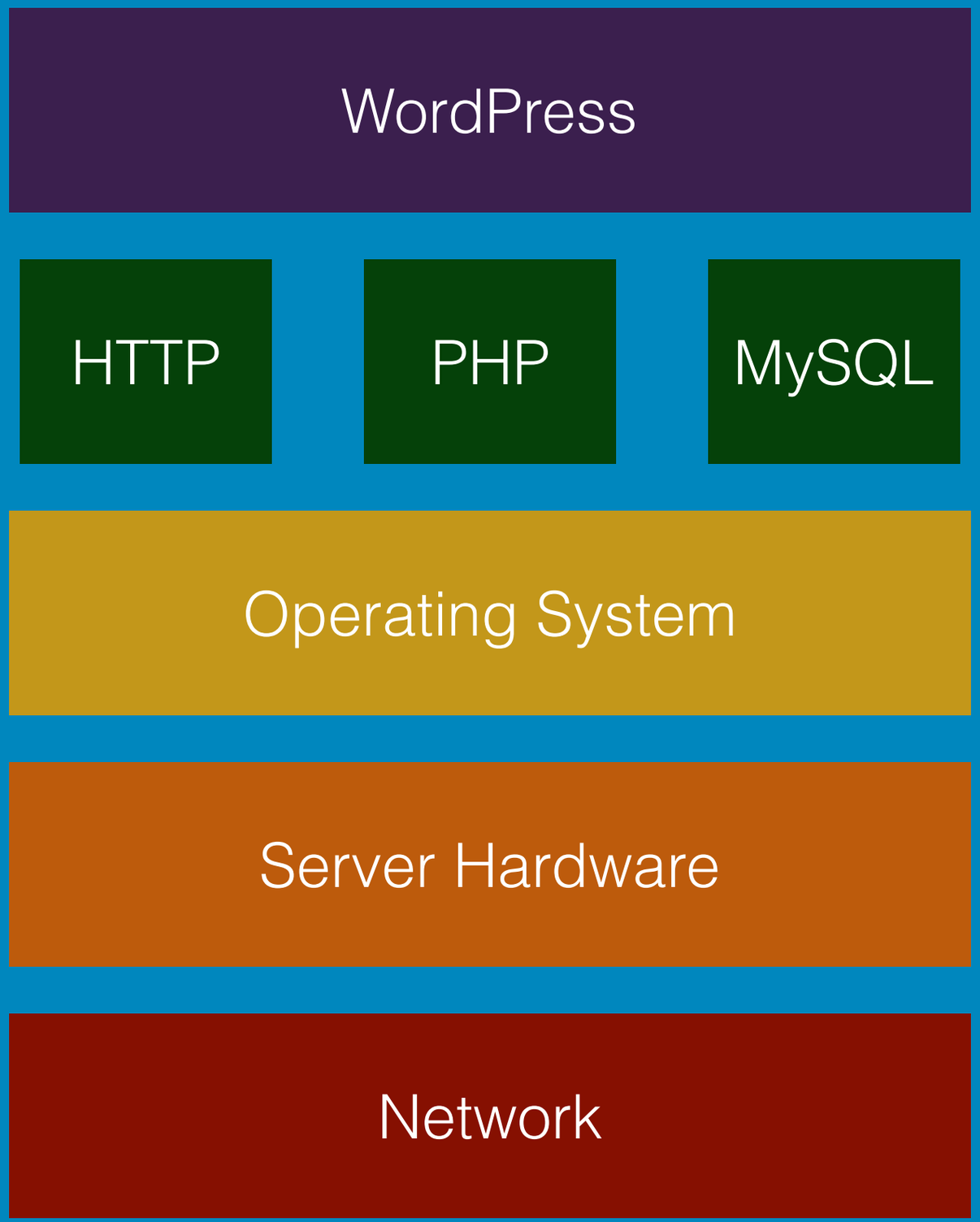
Layers



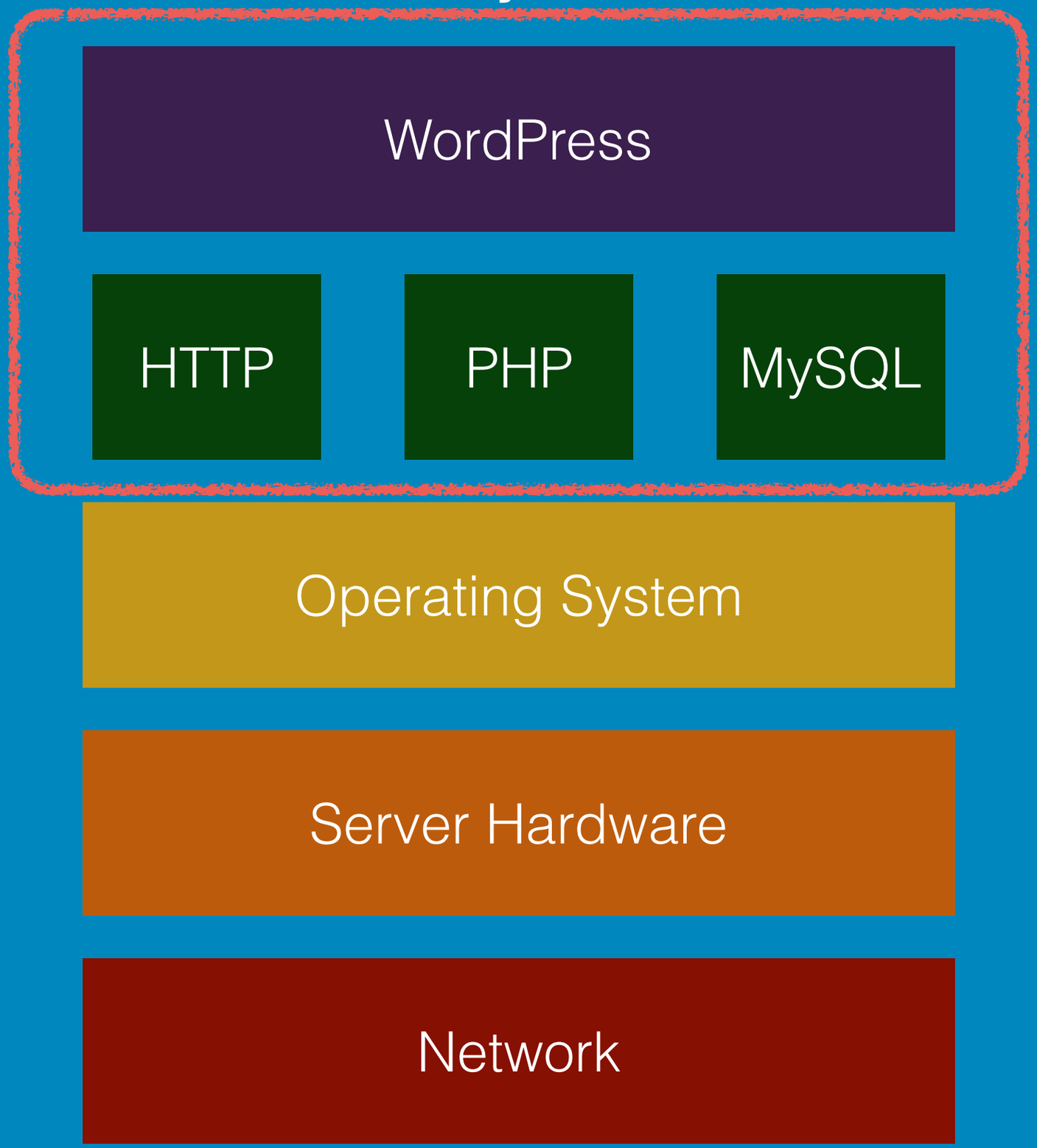
Layers



Layers


WordPress
HTTP
PHP
MySQL
Operating System
Server Hardware
Network

Layers


WordPress
HTTP
PHP
MySQL
Operating System
Server Hardware
Network

MySQL

- **Use InnoDB**
  unless you have a good reason not to

- **Use the slow query log**

- **Use EXPLAIN on queries,**
  learn what the output means

- **Double, Triple check your indexes**

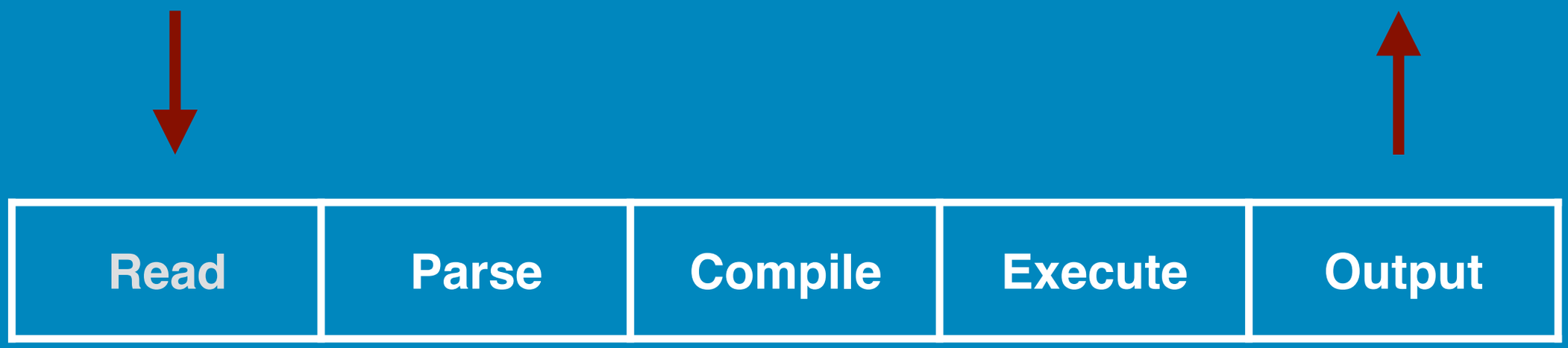
- **Learn about quirks**

PHP

- Use an opcode cache (Zend Opcache, XCache, APC)
- Use newer versions
- Don’t use on static files
- Be careful with preg_* functions
- Profile your code (Xdebug,VLD, XHProf)
- Don’t re-invent native PHP functions

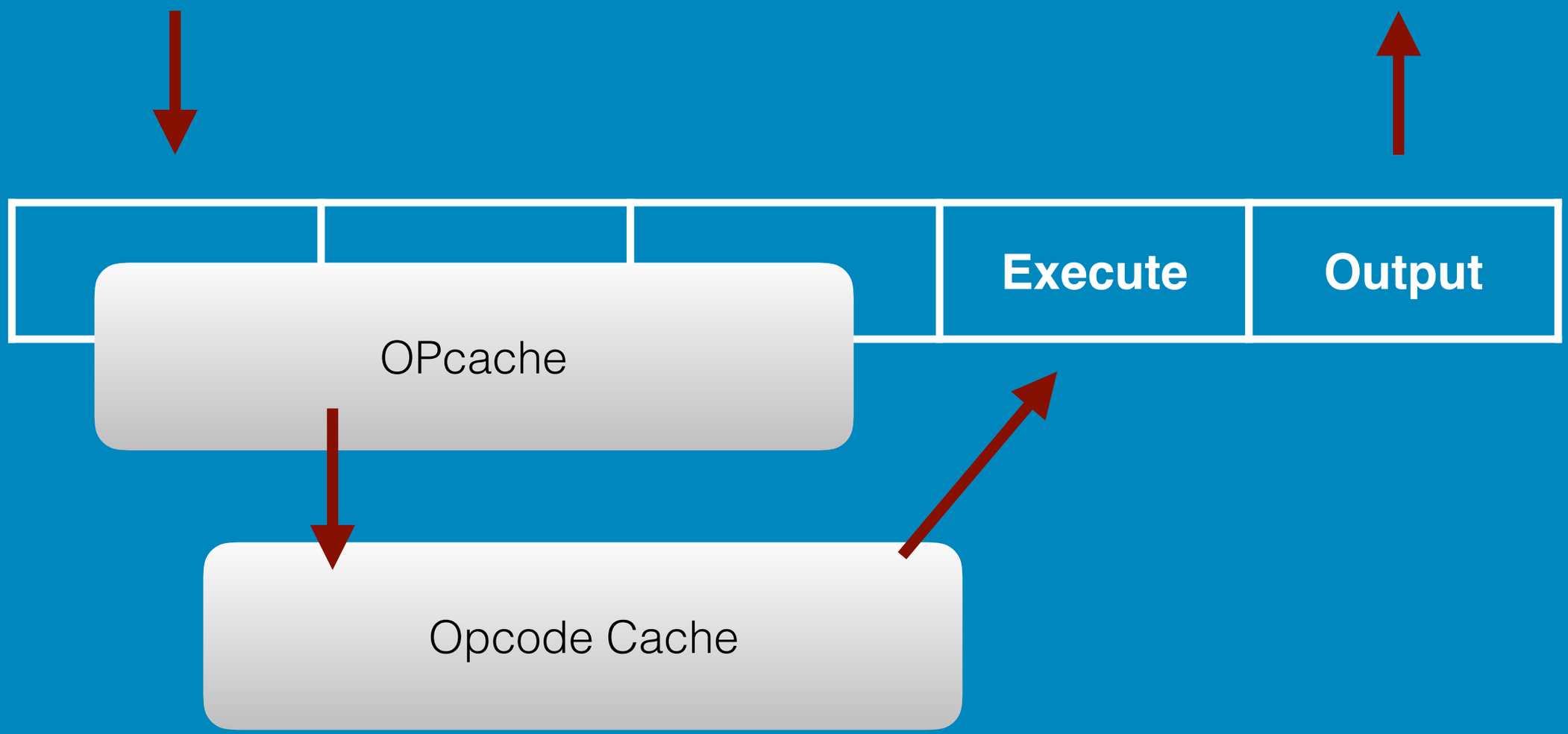
PHP - opcode caching

.php



PHP - opcode caching

.php



PHP - opcode caching

- Reduce time per request by 4X
- Increase requests per second by 4x
- Less work, for the same result means more capacity

WordPress Native Caching APIs

Transients

- Persistent out of the box
- Stored in wp_options: _transient_{key}
- WordPress uses them for certain internal functions
- set_, get_, and delete_transient()

Object Cache

- Not persistent without a plugin, such as W3 Total Cache or Memcached Object Cache
- Storage depends on server's and plugin's capabilities
- Used extensively within WordPress Cache objects can be grouped wp_cache_add(), _set, _get, _delete

Object cache - plugins

- **W3 Total Cache** provides object level caching using disk, opcode or memcache(d) memory stores.
- **Memcached Object Cache** provides a persistent backend for the WordPress object cache. A memcached server and the PECL memcached extension are required.
- **APC Object Cache** provides a persistent backend for the WordPress object cache using APC, the default opcode and key/value cache of PHP.
- **Redis Object Cache** supports the use of Predis (PHP client library), HHVM's Redis extension and the PECL Redis extension to provide a persistent backend for the WordPress object cache. Redis is required.
- **WP File Cache** implements object level persistent caching by shifting the load from your database to your disk/file system.

Memcached

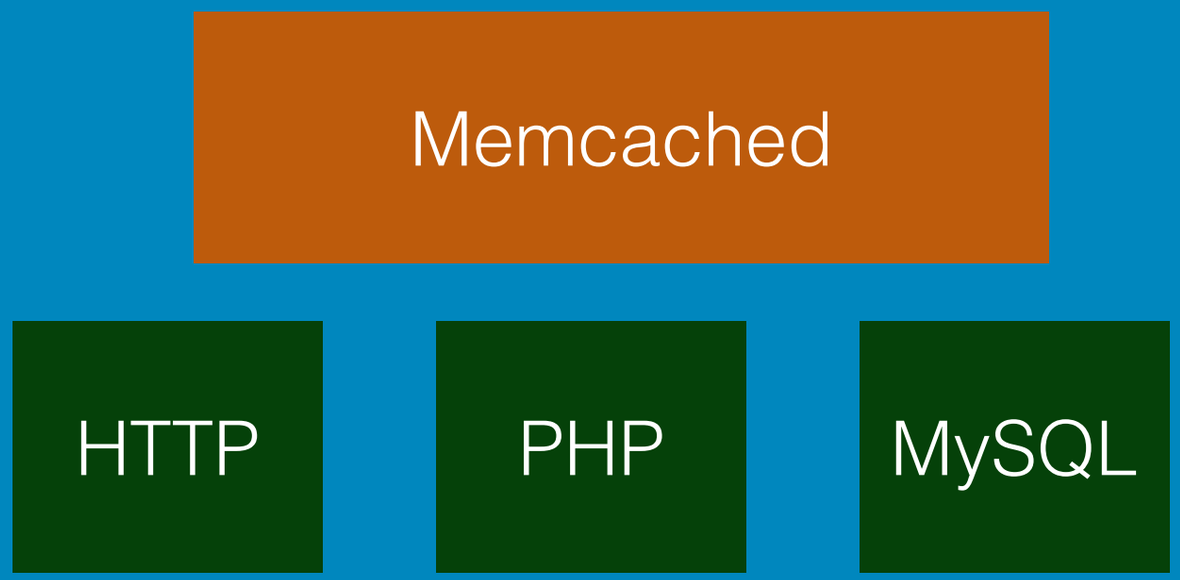
- Free & open source
- High-performance, distributed memory object caching system
- Key / Value data storage
- In memory only, won’t survive a reboot
- http://memcached.org/

WordPress + Memcached

- **WordPress Object Cache**

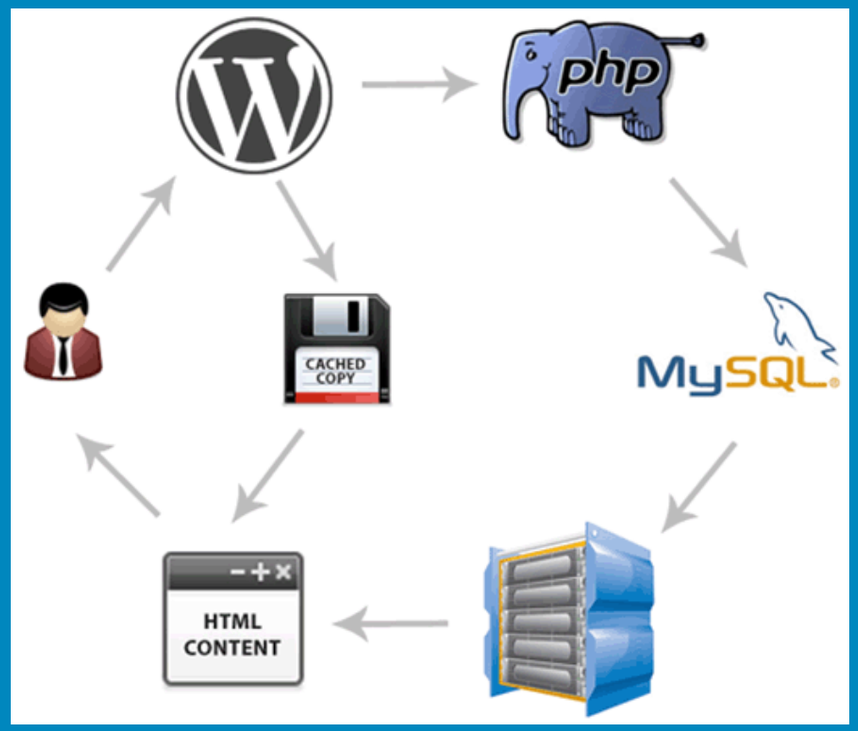
Reduces the number of database queries
http://wordpress.org/extend/plugins/memcached/

WordPress + Memcached



Full Page Caching

- Batcache
- WP Super Cache
- W3 Total Cache



Full Page Caching - Batcache

Batcache is a plugin to store and serve cached versions of rendered pages.

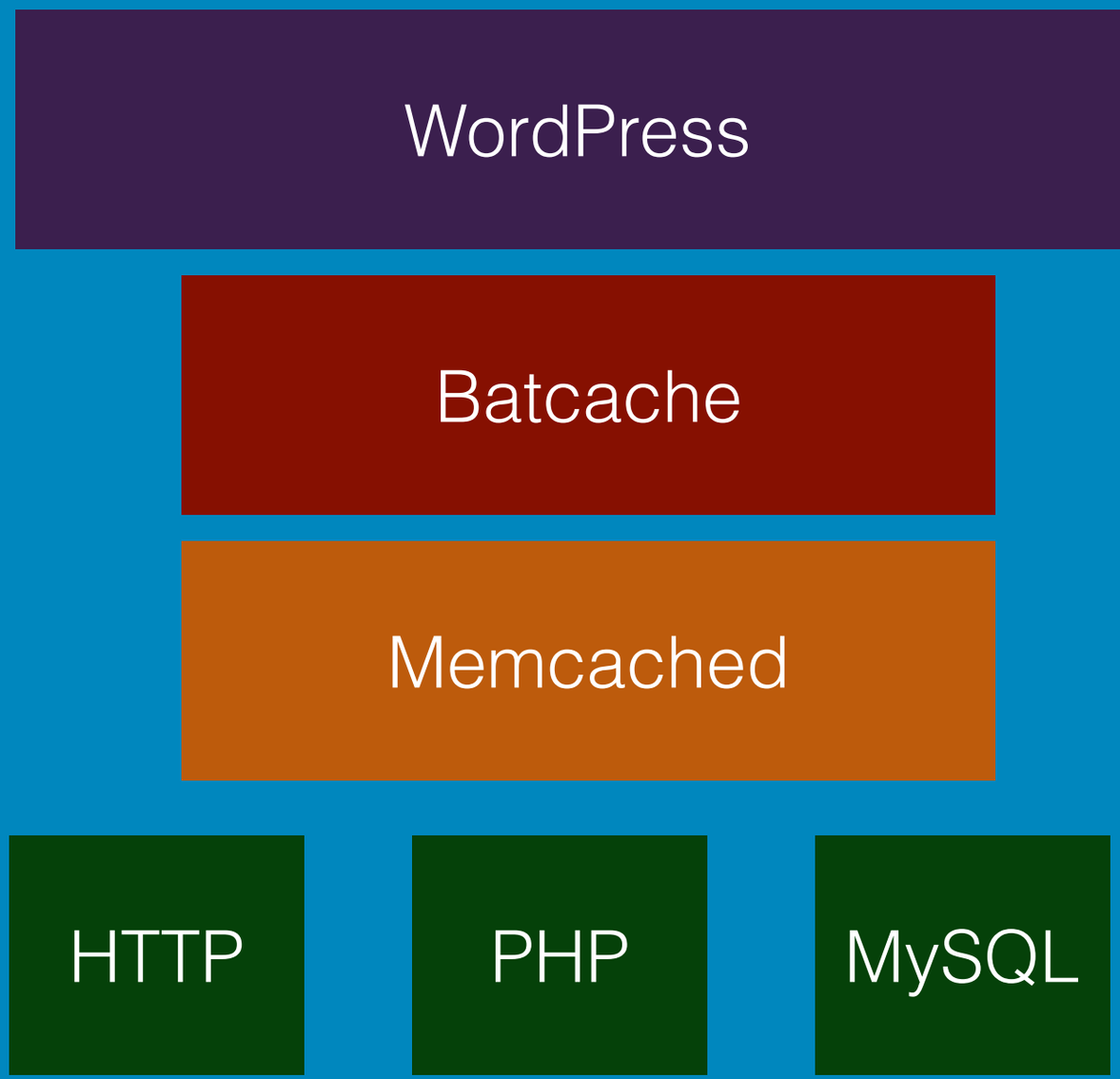
- Batcache uses memcached as its storage and is aimed at preventing a flood of traffic from breaking your site.
- It does this by serving old pages to new users.
- This reduces the demand on the web server CPU and the database.
- It also means some people may see a page that is up to 5 minutes old.

Full Page Caching - Batcache

Who receives a cached pageview?

- By default, all new users receive a cached pageview.
- New users are defined as anybody who hasn’t interacted with your domain —once they’ve left a comment or logged in, their cookies will ensure they get fresh pages.
- Note that URLs with query strings are automatically exempt from Batcache.

WordPress + Memcached



Full Page Caching - Batcache

Because Batcache caches fully rendered pages, per-user interactions on the server-side can be problematic.

This means usage of objects/functions like:

- $_COOKIE
- setcookie
- $_SERVER['HTTP_USER_AGENT']
- Anything that’s unique to an individual user

cannot be relied on, as the values may be cached and cross-pollution can occur.

In most cases, any user-level interactions should be moved to client-side using JavaScript.

WP Home page (not logged)	Requests/sec (mean)
PHP	0.89
PHP + OPCODE	1.05
PHP + OPCODE + OBJECT CACHE (MEMCACHED OC + MEMCACHE SERVER)	7.33
PHP + OPCODE + OBJECT CACHE + PAGE CACHE (W3 Total Cache + MEMCACHE SERVER)	29.17
PHP + OPCODE + OBJECT CACHE + PAGE CACHE (W3 Total Cache on Disk)	29.49
PHP + OPCODE + OBJECT CACHE + PAGE CACHE (MEMCACHED OC + BATCACHE + MEMCACHE SERVER)	29.72
PHP + OPCODE + PAGE CACHE (WP Super Cache)	29.57

Deferred execution of code - Asynchronous Jobs Queue

Deferred execution of code

PHP executes in a single threaded manner.

That is the interpreter does A, then B, then C, then D, then finishes, and wipes its slate clean (shared nothing).

It does this same things for every page view on your site.

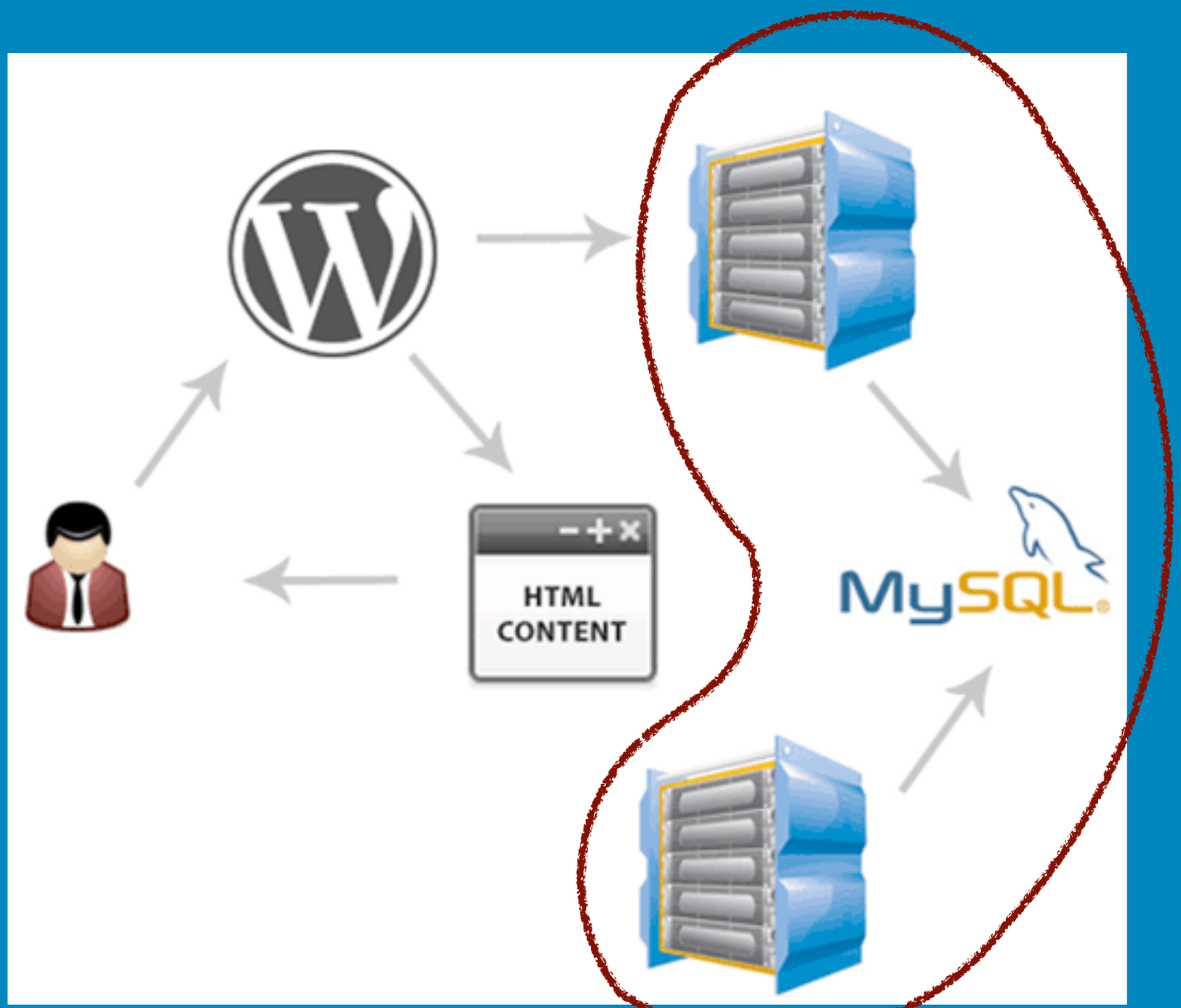
Deferred execution of code

- **Everything from spam checking, through sending emails, Push Notifications, is *not necessary to render the HTML page.***

- **Those tasks could be stashed in the Jobs system.**

- **Another process elsewhere on your network will pick up the job, pull out the message, check it for duplication, spam, language, stick it in the db if that all passes, and then finally send out emails / PNs to all the recipients.**

Deferred execution of code



Jobs system

Deferred execution of code

- **Lots of options** (WP Async Task library, WP Background Processing)
- **We built one for WordPress.com**

- http://code.trac.wordpress.org/browser/jobs
- http://code.trac.wordpress.org/wiki/JobsDocs
- All PHP + MySQL
- Good for anything that isn’t required for rendering a page

Deferred execution of code - Send Push Notifications to Users

- **WordPress**
  - A new comment is posted on the blog
  - WordPress does follow the normal flow, and enqueue a new Job in the Jobs system
- **The Jobs Systems**
  - Loads the job from the DB
  - Checks the userID of the author of the post.
  - Checks that the Author of the post has a device registered
  - Check user preferences
  - Checks the type of the device (Android, iOS, Windows)
  - Creates the PN object by using the device type information, and send it to the correct remote server
  - Handles errors and re-enqueue it if something went wrong

AUTOMATTIC