



# High Performance PHP & MySQL Scaling Techniques

Elliott White III – *Eli*

<http://eliw.com/>

# *What's all this then?*

- Introduction
- Standard Solution
- Quick PHP Solutions
- APC User Variables
- Memcached
- Purpose Driven Database Servers
- Database Partitioning

# *Introduction*

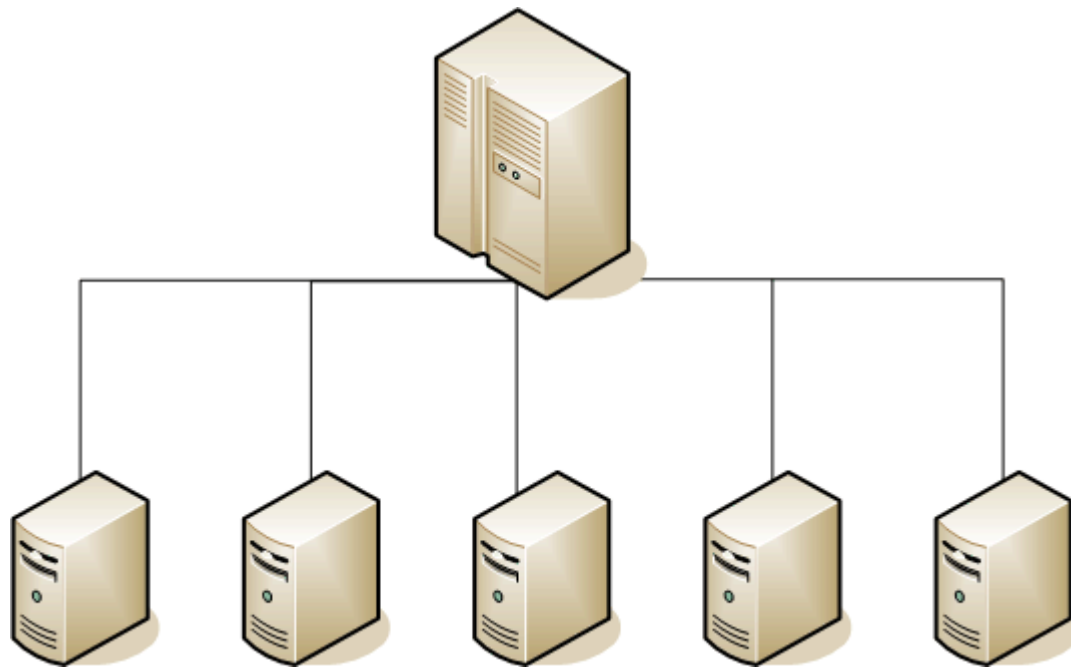
- Performance is a problem
- Scaling your performance is a bigger problem

# *Standard Solution*

How most people setup a basic solution that scales 'so far'.

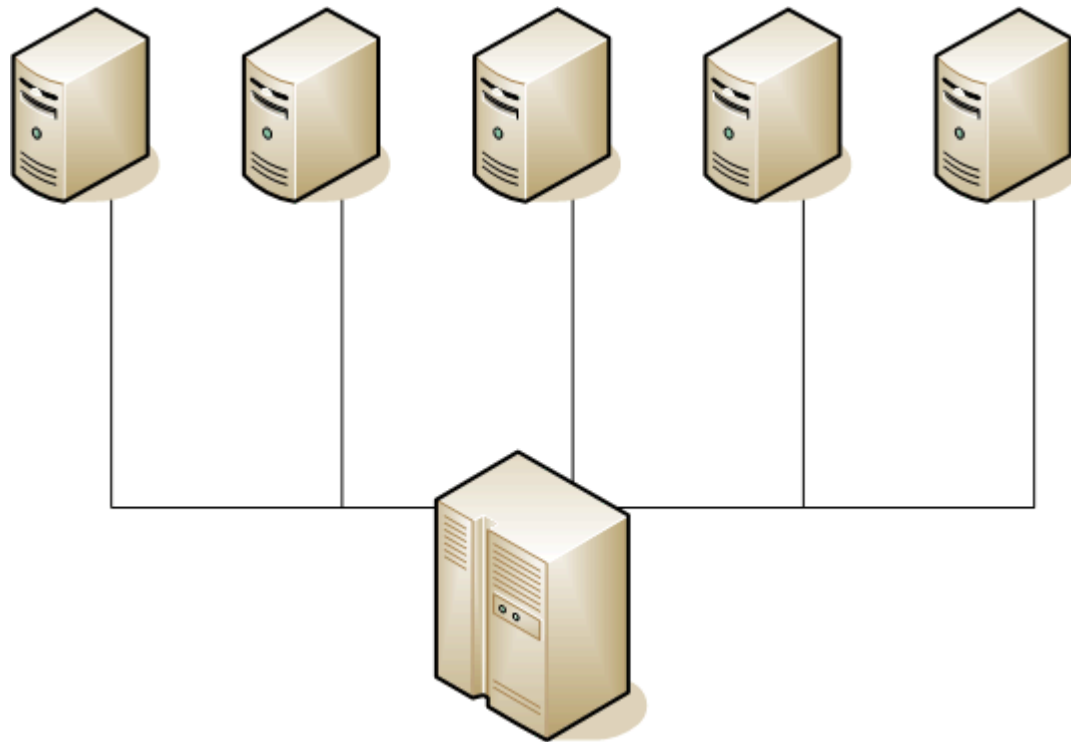
# *Standard Solution*

Many PHP Servers behind a load balancer:



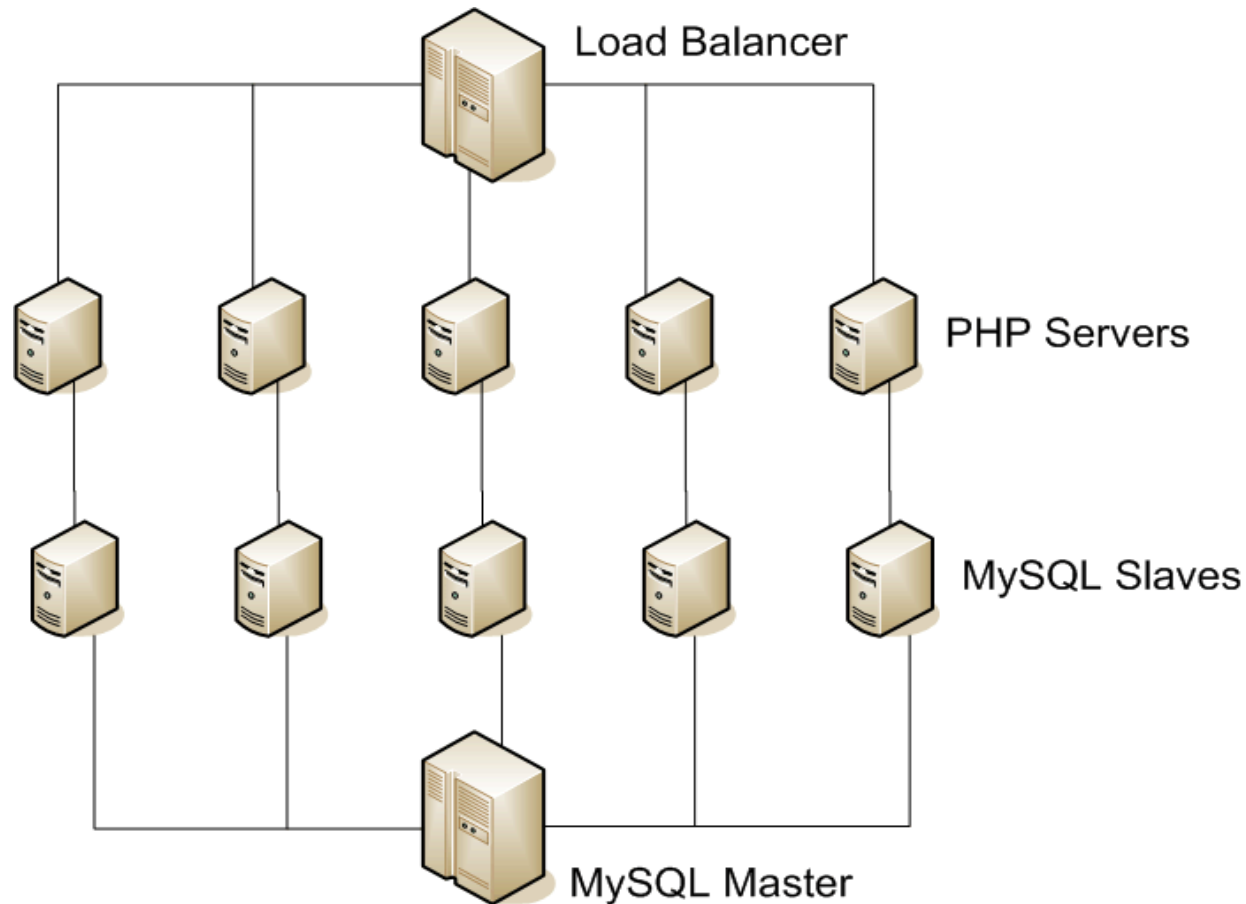
# *Standard Solution*

Many MySQL slaves, talking to a master



# Standard Solution

Randomized or 'planned' PHP to MySQL relations



# *Quick PHP Solutions*

A number of things that will speed up PHP,  
if that is your bottleneck.

# *Use an opcode cache*

PHP by default recompiles every page, every request.

APC (Alternative PHP Cache)  
<http://pecl.php.net/package/APC>

# *Stop using PHP*

Specifically move to faster server software, such as thttpd for static HTML pages, images, etc.

# *Pregenerate Content*

If pages do not need to be instantly updated, generate them on a regular basis.

# *Cache content*

Half-way between dynamic and pregenerated.

Cache it as you create it.

Example: jpcache

<http://www.jpcache.com/>

Or Smarty does this for you.

# *APC User Variables*

What is it?

# *APC User Variables Pros & Cons*

## Advantages:

- Allows complicated processing to be done once.
- Stores data as native PHP types in local memory.

## Limitations:

- Data that is stored is local to that web server.
- Has to share memory resources with web server.

# *Memcached*

What is it?

# *Memcached Performance gains*

Caching certain chunks of data that might be used on many different pages.

From that, still being able to dynamically create the page, but using some cached data.

# *Memcached Server Farm*



- Setting up a pool of servers
  - PHP Provides the basics of distributing load across servers.
- Taking it to the next level
  - Failover protection, Redundancy, etc.

# *Memcached disadvantages / issues*

- Coding the actual caching decisions
- Out of date / Old data
- Perpetuating slave lag
- Scaling it further / Getting the most out of caching
- Balancing the farm load

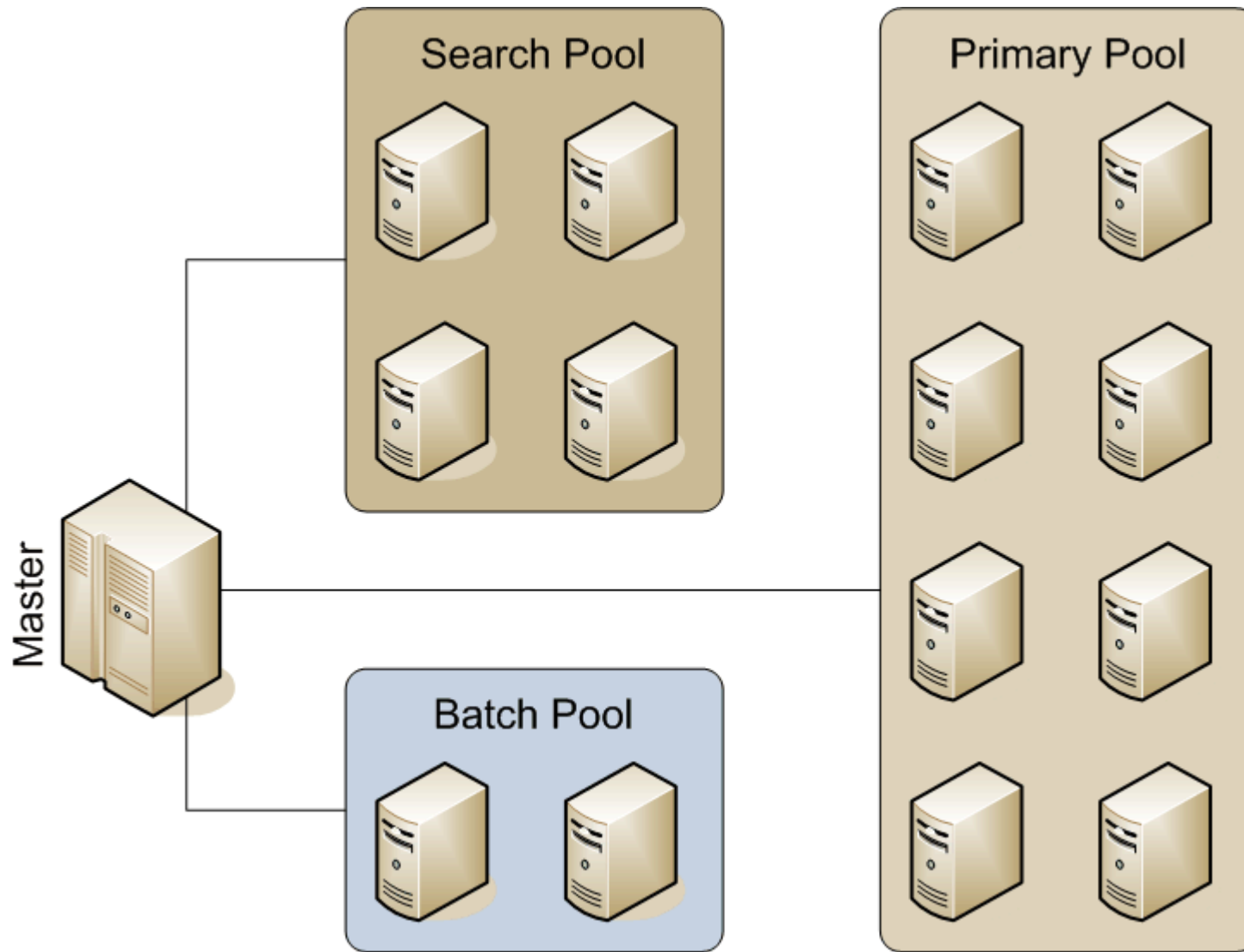
# *Creating Generic Memcached Solutions*

- Create generic/abstract system (classes) to hide connections, load balancing, fail over, and server farm aspects for you.
  - You only ever say 'store' or 'retrieve'
- Next Step: Create a system (classes) to even abstract that further. To completely hide how the data is stored, retrieved, and cached.
  - You just 'ask for the data', and the classes handle everything.

# *Purpose Driven MySQL Pools*

Creating separate slave pools, that are close to identical in order to isolate high database load.

# Purpose Driven Pool Example



# *Database Partitioning*

What is it?

Simplest Definition:

Breaking up your database into a number of smaller ones.

(And I'm not talking about built-in versions)

# *Pros & Cons of Partitioning*

## Pros

- Greater performance
- Tweakable / Scalable

## Cons

- Loss of direct SQL support
- Increased PHP load
- Complicated programming

# *Main Types of Partitioning*

Horizontal

Vertical

Application Level

Discussion topic: Partitioning within same database

# *Horizontal Partitioning*

“Moving various rows of your table into different tables”

Various methodologies:

- Range Based
- Date Based
- Interlaced
- User Based
- Partial partitioning works well here

# *Vertical Partitioning*

“Moving various columns of your table into different tables”

Various methodologies:

- Move rarely used columns into auxiliary table
- Move often empty columns into auxiliary table
- Move columns that are not used in where clauses

# *Application Level Partitioning*

“Moving various tables of your DB onto different servers”

Various methodologies:

- Move single tables to specific servers
- Move groups of related tables together to allow joining

# *Generic code to handle partitioning*

Coding to partitions can get complicated.

Make a set of functions/classes that understand the shards so that you don't have to.

Your code, again, should only be concerned with:  
Give me the data!

# *Digg is Hiring!*

Digg.com is hiring experienced PHP programmers!

<http://digg.com/jobs>  
[jobs@digg.com](mailto:jobs@digg.com)

# Any Questions?

For this presentation and more:

<http://eliw.com/>

<http://digg.com/users/EliW>

The Digg logo, consisting of the word "digg" in a white, lowercase, sans-serif font, centered on a blue rectangular background.