Caching with Memcached

Ilia Alshanetsky
@iliaa

Memcached

* an elephant that uses Memcache is actually quite forgetful.

- Interface to Memcached - a distributed, in-memory caching system
- Provides a simple Object Oriented interface
- Offers a built-in session handler
- Purpose built, so lots of nifty features

Memcache

Memcached

Igbinary serializer

Memcache

Faster

Binary protocol support

Buffered writes

Memcached

FastLz compression

Multi-Server interface

Delayed fetches

Basics in Practice

```php
$mc = new MemCached();

// Configure memcache to connect
$mc->addServer('localhost', '11211');

// try to add an array using “key” for 1 day
if (!$mc->add('key', array(1,2,3), 86400)) {
    // if already exists, let's replace it
    if (!$mc->replace('key', array(1,2,3), 86400)) {
        die("Critical Error");
    }
}

// let's fetch our data
if (($data = $mc->get('key')) !== FALSE) {
    // let's delete it now
    $mc->delete('key'); // RIGHT NOW!
}
```

Data Retrieval Gotcha

```php
$mc->add('key', FALSE);

if (($data = $mc->get('key')) !== FALSE) {
  die("Not Found?"); // not true
  // The value could be FALSE, 0, array(), NULL, ""
}

// The "correct" way!
if (
        (($data = $mc->get('key')) === FALSE)
        &&
        ($mc->getResultCode() != MemCached::RES_SUCCESS)
) {
  die("Not Found");
}
```

Interface Basics Continued...

```php
$mc = new MemCached();
// on local machine we can connect via Unix Sockets for better speed
$mc->addServer('/var/run/memcached/11211.sock', 0);

// add/or replace, don't care, just get it in there
// without expiration parameter, will remain in cache “forever”
$mc->set('key1', array(1,2,3));

$key_set = array('key1' => “foo”, 'key2' => array(1,2,3));
```

```php
// store multiple keys at once for 1 hour
$mc->setMulti($key_set, 3600);

// get multiple keys at once
$data = $mc->getMulti(array_keys($key_set));
/* array(
    'key1' => ‘foo’
    'key2' => array(1,2,3)
) */
```

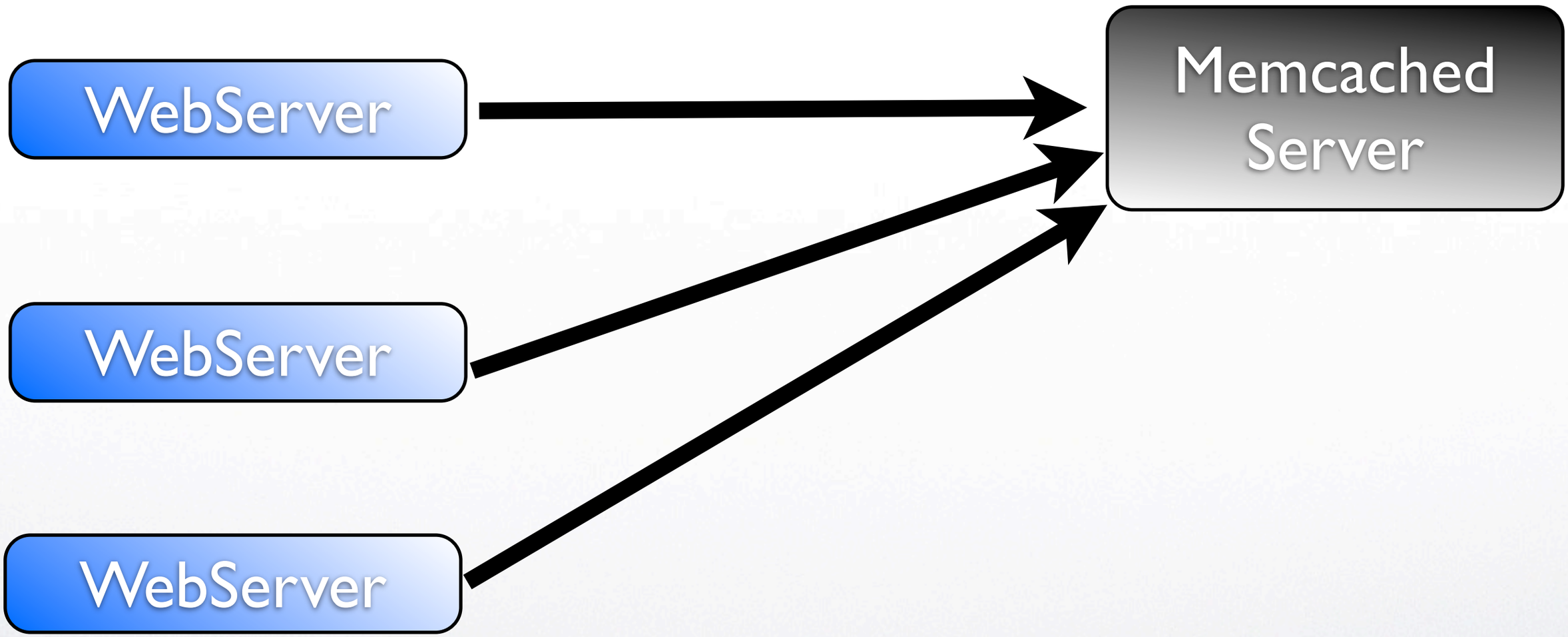
For multi-(get|set), all ops must succeed for successful return.

Multiple Servers

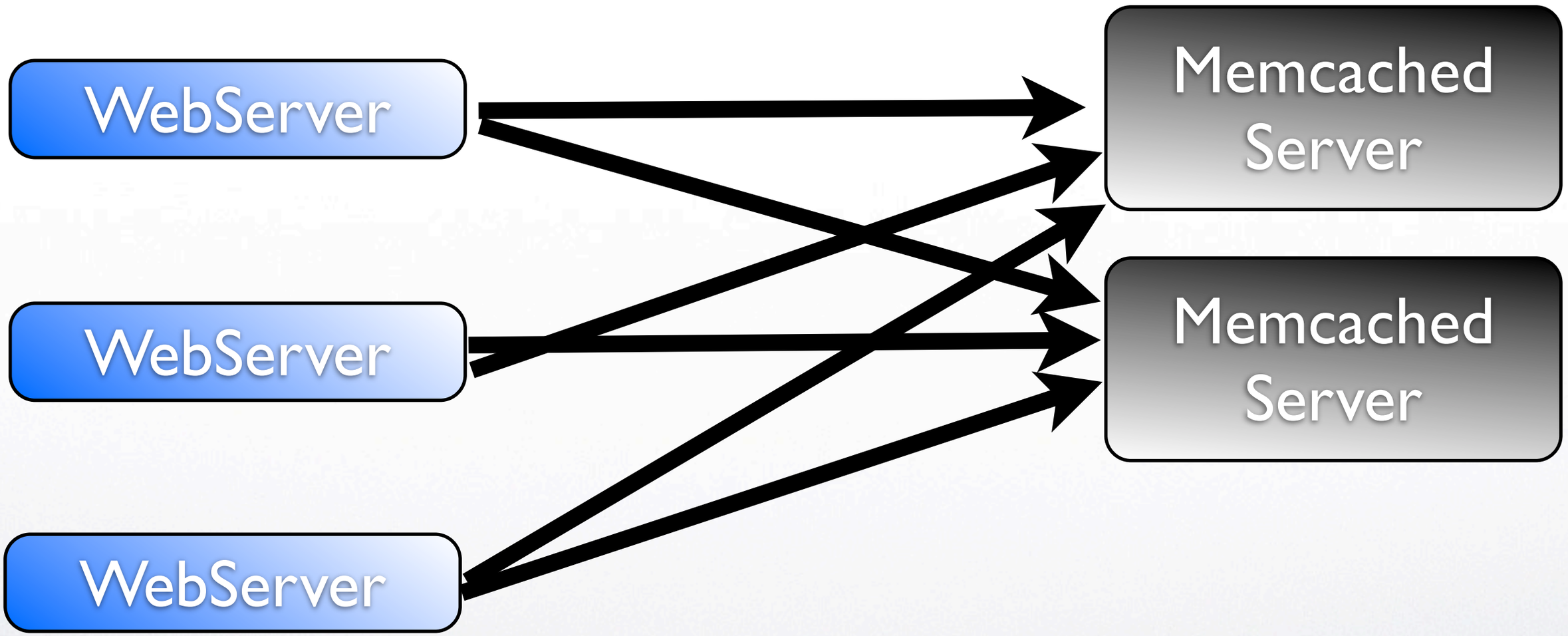
```php
$mc = new MemCached();

// add multiple servers to the list
// as many servers as you like can be added
$mc->addServers(array(
      array('localhost', 11211, 80), // high-priority 80%
      array('192.168.1.90', 11211, 20)// low-priority 20%
));

// You can also do it one at a time, but this is not recommended
$mc->addServer('localhost', 11211, 80);
$mc->addServer('192.168.1.90', 11211, 20);

// Get a list of servers in the pool
$mc-> getServerList();
// array(array('host' => … , 'port' => … 'weight' => …))
```

Architecture...

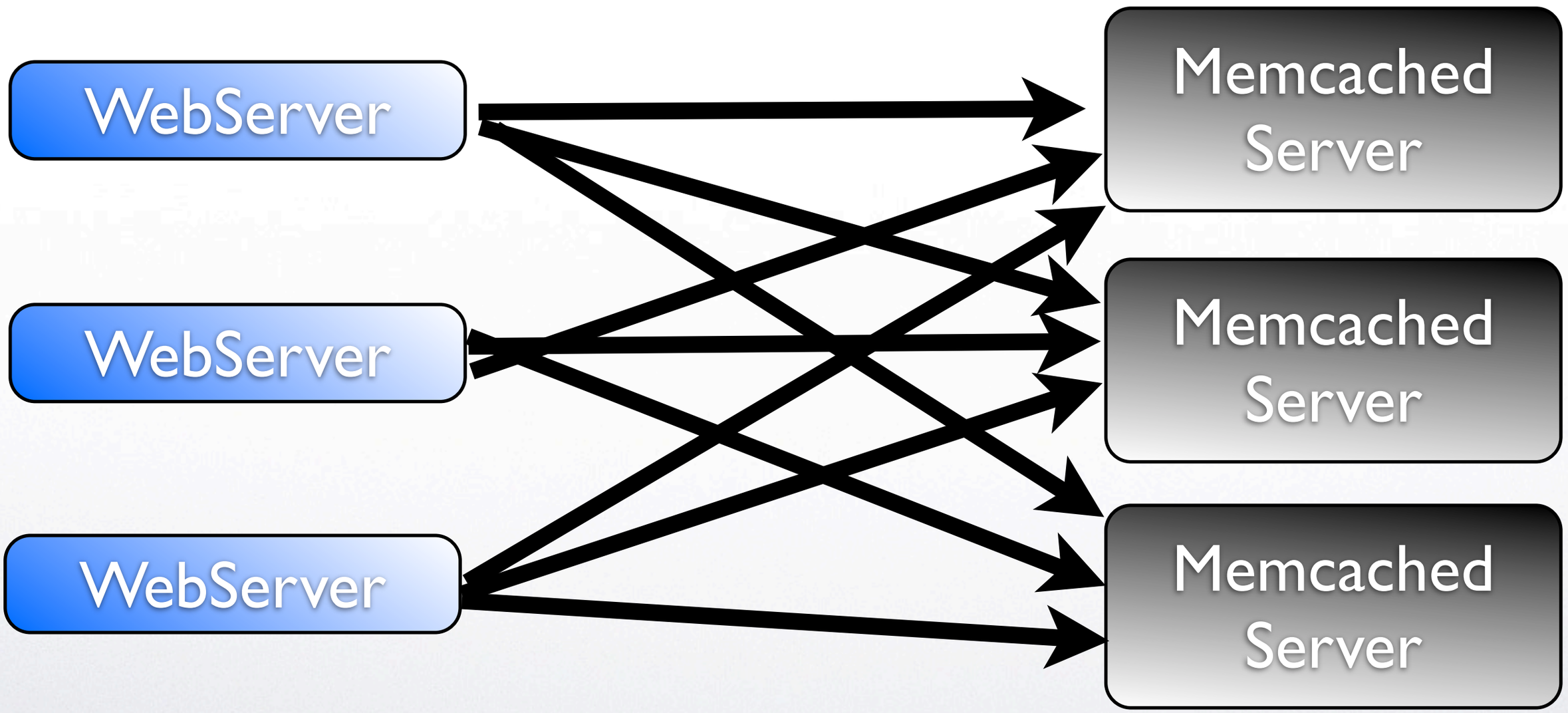
Architecture...



Architecture...



Architecture...



Modulo Approach

The modulo, or "naive approach" used to distribute keys across many servers is pretty simple.

```
$server_idx = crc32('my_key') % $num_servers;
```

Convert key to an integer and do modulo by the # of servers in the pool.

Modulo Approach

The modulo, or "naive approach" used to distribute keys across many servers is pretty simple.

```
$server_idx = crc32('my_key') % $num_servers;
```

Convert key to an integer and do modulo by the # of servers in the pool.

But what if the number of servers changes?

Modulo Approach

The modulo, or "naive approach" used to distribute keys across many servers [illegible]

```
$server_[illegible] = [illegible]c32([illegible]key') % $num_servers;
```

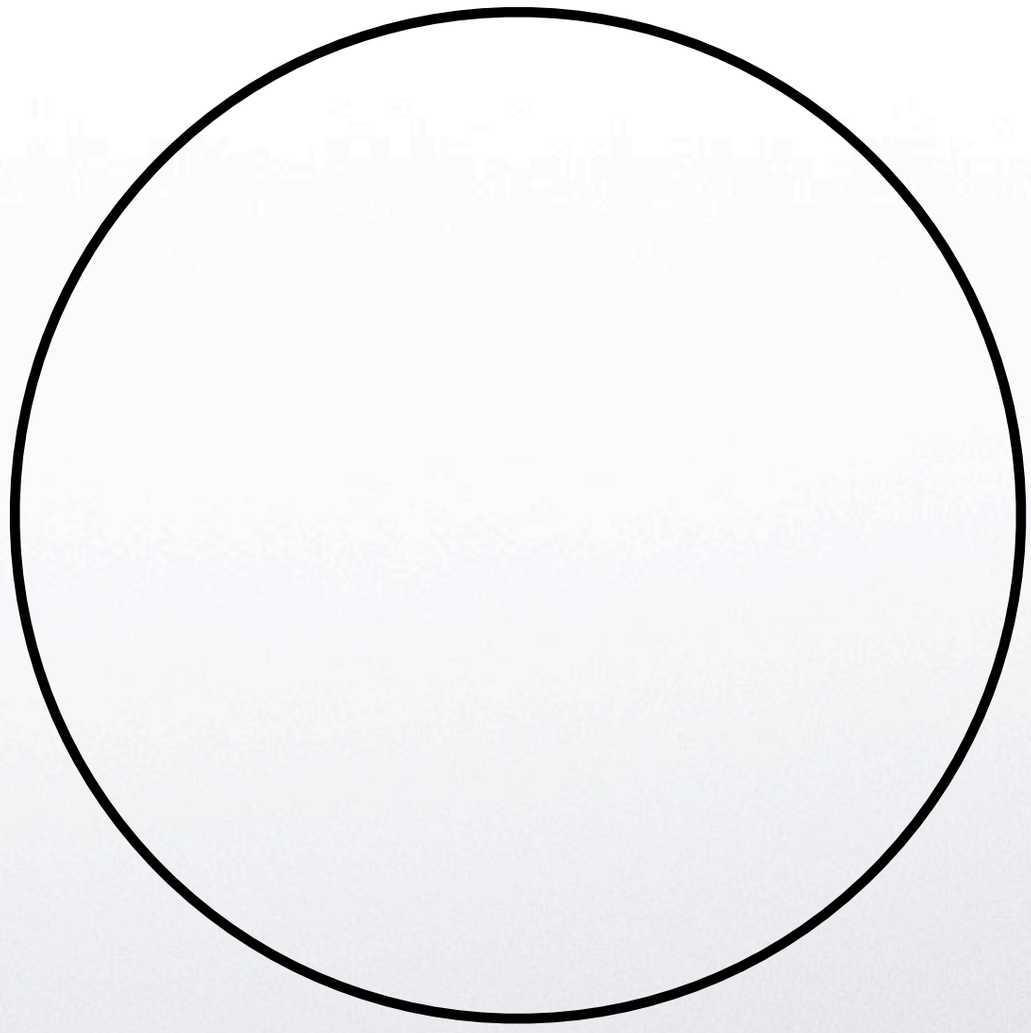
Convert key to an integer and do modulo by the # of servers in the pool.

But what if the number of servers changes?

FAIL

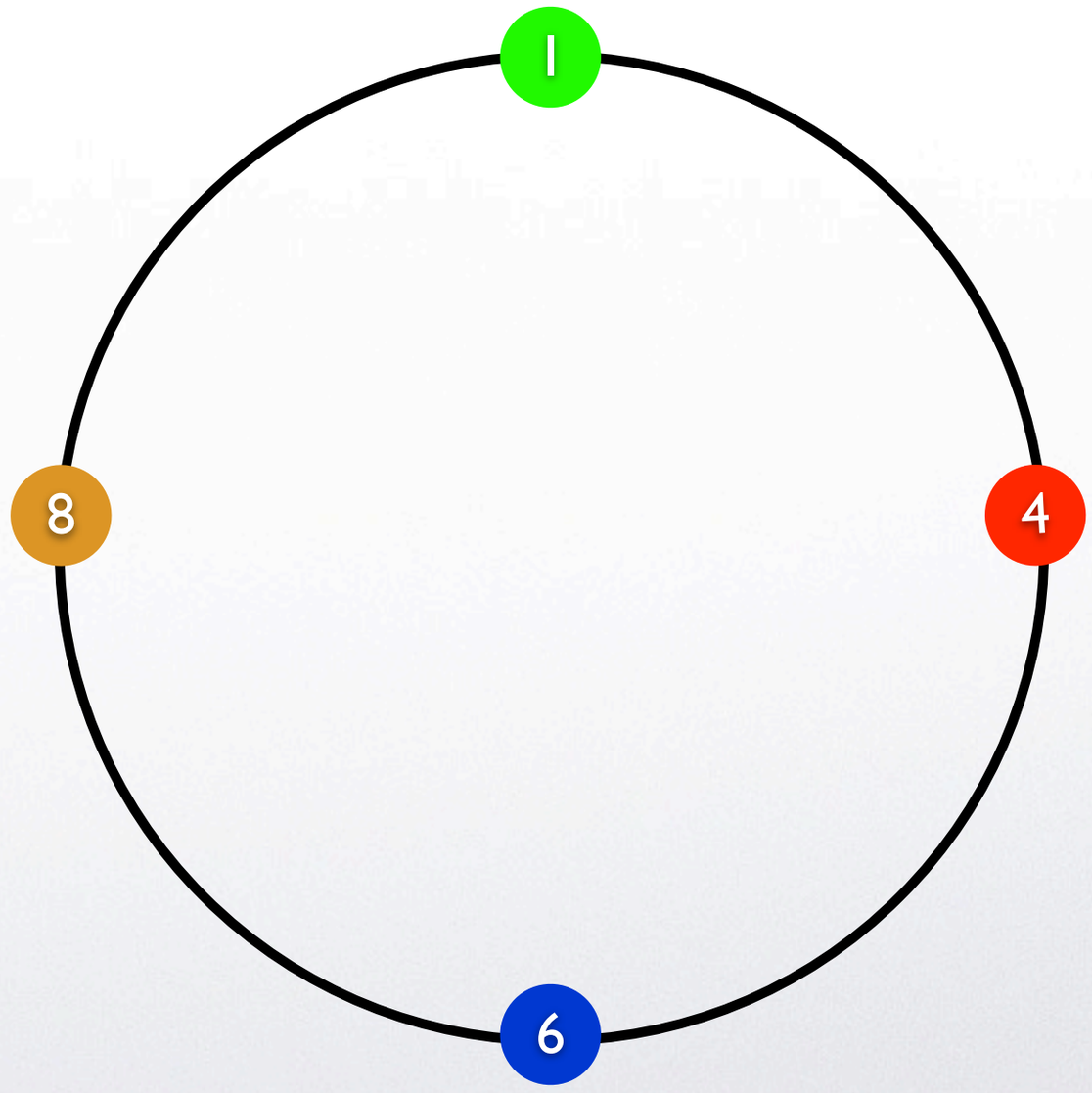
Consistent Hashing

Select **N** random integers for each server and sort them into an array of values **N * # of servers**



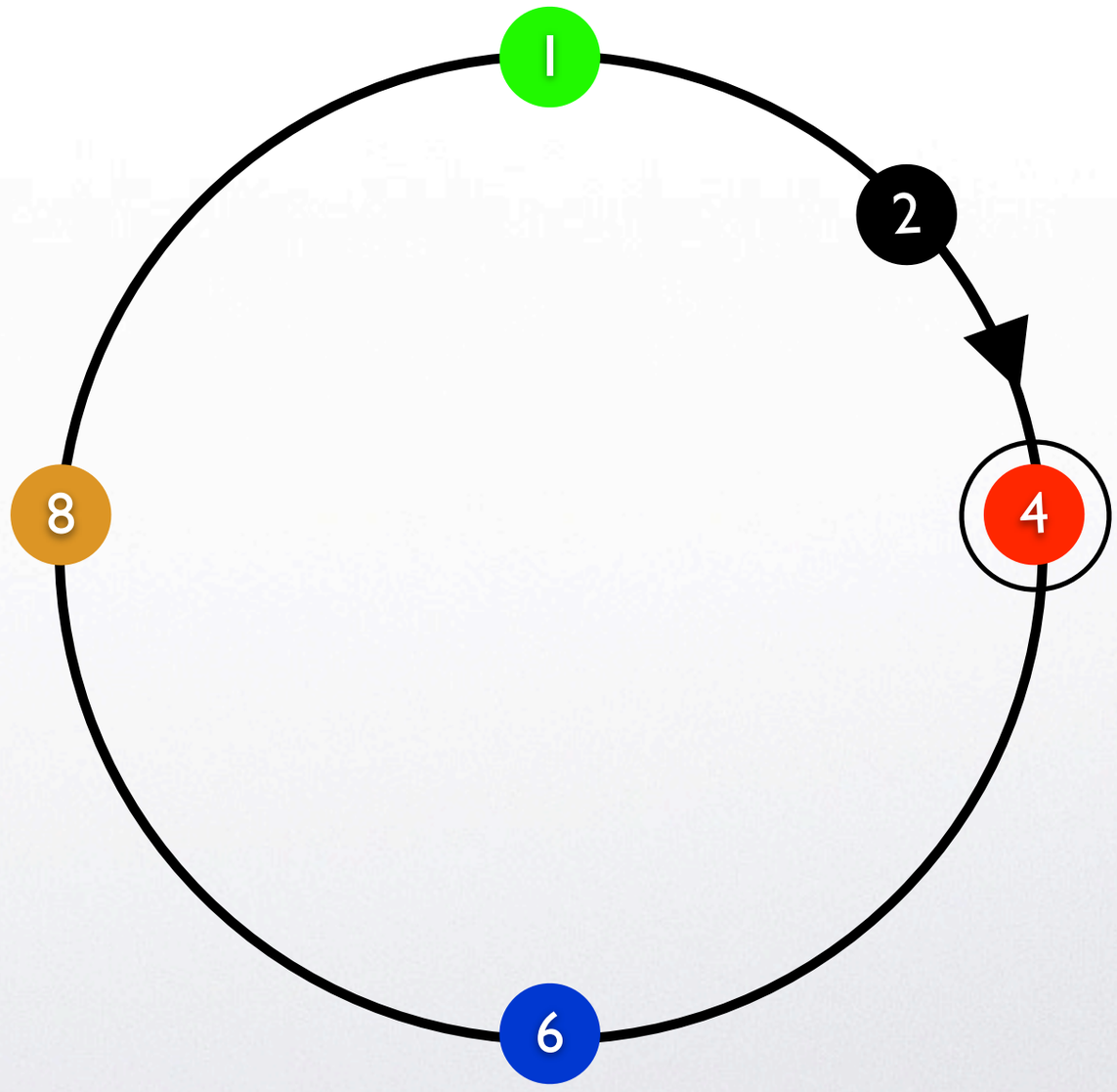
Consistent Hashing

Select **N** random integers for each server and sort them into an array of values **N * # of servers**



Consistent Hashing

Select **N** random integers for each server and sort them into an array of values **N * # of servers**



Lookup key's int hash proximity to randomly picked values in clock-wise manner.

Consistent Hashing

```
// default, modulo distribution mode
$mem->setOption(
  Memcached::OPT_DISTRIBUTION,
  Memcached::DISTRIBUTION_MODULA
);
```

```
// consistent hasing
$mem->setOption(
  Memcached::OPT_DISTRIBUTION,
  Memcached::DISTRIBUTION_CONSISTENT
);
```

Data Segmentation

Memcached interface allows you to store certain types of data on specific servers

```
$mc = new MemCached();
$mc->addServers( … );

// Add data_key with a value of “value” for 10 mins to
// server identified by “server_key”
$mc->addByKey('server_key', 'data_key', 'value', 600);

// Fetch key from specific server
$mc->getByKey('server_key', 'data_key');

// Add/update key on specific server
$mc->setByKey('server_key', 'data_key', 'value', 600);

// Remove key from specific server
$mc->deleteByKey('server_key', 'data_key');
```

Fail-Over Callbacks

```
$m = new Memcached();
$m->addServer('localhost', 11211);


$data = $m->get('key',
    function (Memcached $memc, $key, &$value) {
        $value = 'retrieve value';
        $memc->add($key, $value);
      return $value;
    }
);
```

Only supported for get() & getByKey() methods

Delayed Data Retrieval

One of the really neat features of Memcached extension is the ability to execute the "fetch" command, but defer the actual data retrieval until later.

Particularly handy when retrieving many keys that won't be needed until later.

Delayed Data Retrieval

```php
$mc = new MemCached();
$mc->addServer('localhost', '11211');

$mc->getDelayed(array('key'));
// parameter is an array of keys

/* some PHP code that does “stuff” */
```

```php
// Fetch data one record at a time
while ($data = $mc->fetch()) { ... }

// Fetch all data in one go
$data = $mc->fetchAll();
```

Delayed Result C.B.

The delayed result callback allows execution of code upon successful delayed retrieval.

Delayed Result C.B.

```php
$m = new Memcached();
$m->addServer('localhost', 11211);

$m->getDelayed(
    array('footer','header'), false, 'cb');

function cb(Memcached $m, $data) {
    //$data = array('key' => '...', 'value' => '...');
    layout::$data['key']($data['value']);
}
```

Callback will be called individually for every key

Namespacing w/Counters

```php
$mc = new MemCached();
$mc->addServer('localhost', 11211);

// add key position if does not already exist
if (!$mc->add('key_pos', 1)) {
    // otherwise increment it
    $position = $mc->increment('key_pos');
} else {
    $position = 1;
}

// add real value at the new position
$mc->add('key_value_' . $position, array(1,2,3));
```

Simplifies cache invalidation and reduces lock contention

Global Namespacing

Global key namespacing allows rapid invalidation of all keys, on major changes, such as a software version upgrade.

```php
$mem->setOption(
    Memcached::OPT_PREFIX_KEY,
    "_" . PHP_VERSION . "_"
);

$mem->set("foo", "bar");
// actual key is _5.3.3-pl1-gentoo_foo

$mem->get("foo");
// gets value from _5.3.3-pl1-gentoo_foo
```

Buffered Writes

When doing many consecutive writes, or writing large data blocks, use buffered writes.

```
$m->setOption(Memcached::OPT_BUFFER_WRITES, true);
```

Significant performance increase...

Data Compression

In many cases performance can be gained by compressing large blocks of data. Since in most cases network IO is more expensive then CPU speed + RAM.

```
$mc = new MemCached();
$mc->addServer('localhost', 11211);
// enable compression
$mc->setOption(Memcached::OPT_COMPRESSION, TRUE);
```

Data Compression

Related INI settings (INI_ALL)

Other possible value is zlib
memcached.compression_type=fastlz

minimum compression rate
memcached.compression_factor=1.3

minimum data size to compress
memcached.compression_threshold=2000

PHP Serialization

If you are using Memcache to store complex data types (arrays & objects), they will need to be converted to strings for the purposes of storage, via serialization.

Memcached can make use of igbinary serializer that works faster (~30%) and produces more compact data set (up-to 45% smaller) than native PHP serializer.

http://github.com/igbinary

Enabling Igbinary

Install Memcached extension with

--enable-memcached-igbinary

```
$mem = new MemCached();
$mem->addServer('localhost', 11211);
```

```
// use Igbinary serializer
$mem->setOption(
    Memcached::OPT_SERIALIZER,
    Memcached::SERIALIZER_IGBINARY
);
```

Utility Methods

```php
$mc = new MemCached();
$mc->addServer('localhost', 11211);

// memcached statistics gathering
$mc->getStats();
```

```
Array
(
    [server:port] => Array
        (
            [pid] => 4933
            [uptime] => 786123
            [threads] => 1
            [time] => 1233868010
            [pointer_size] => 32
            [rusage_user_seconds] => 0
            [rusage_user_microseconds] => 140000
            [rusage_system_seconds] => 23
            [rusage_system_microseconds] => 210000
            [curr_items] => 145
            [total_items] => 2374
            [limit_maxbytes] => 67108864
            [curr_connections] => 2
            [total_connections] => 151
            [connection_structures] => 3
            [bytes] => 20345
            [cmd_get] => 213343
            [cmd_set] => 2381
            [get_hits] => 204223
            [get_misses] => 9120
            [evictions] => 0
            [bytes_read] => 9092476
            [bytes_written] => 15420512
            [version] => 1.2.6
        )
)
```

```php
// clear all cache entries
$mc->flush();

// clear all cache entries
// in 10 minutes
$mc->flush(600);
```

Installing Memcached

Memcached - www.memcached.org

libMemcached - libmemcached.org/

pecl install memcached (2.1.0 released 4 days ago)

Enable Memcached from your php.ini file

Memcached & Sessions

```
# Session settings

session.save_handler # set to “memcached

session.save_path # set to memcache host server:port

memcached.sess_prefix # Defaults to memc.sess.key.
```

Thank You For Listening

Slides will be available at:

http://ilia.ws

Please give feedback at:

http://joind.in/6822