



# Memcached, the Better Memcache Interface

---

International PHP Conference 2011

Ilia Alshanetsky

@iliaa



# root@foo \$: whois ilia

PHP: Core Developer Since 2001  
Release Manager of 4.3, 5.1 and 5.2 branches  
Author of a “few” extensions

Work: CIO at Centah Inc.

Hobbies: Photography, Biking

Dev. Interests: Performance & Security



# 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



```
$MC = NEW MEMCACHED();
```

```
// CONNECT TO MEMCACHE ON LOCAL MACHINE, ON DEFAULT PORT  
$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



```
$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...

```
$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", 'KEY1' => ARRAY(1,2,3));  
  
// 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



```
$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...



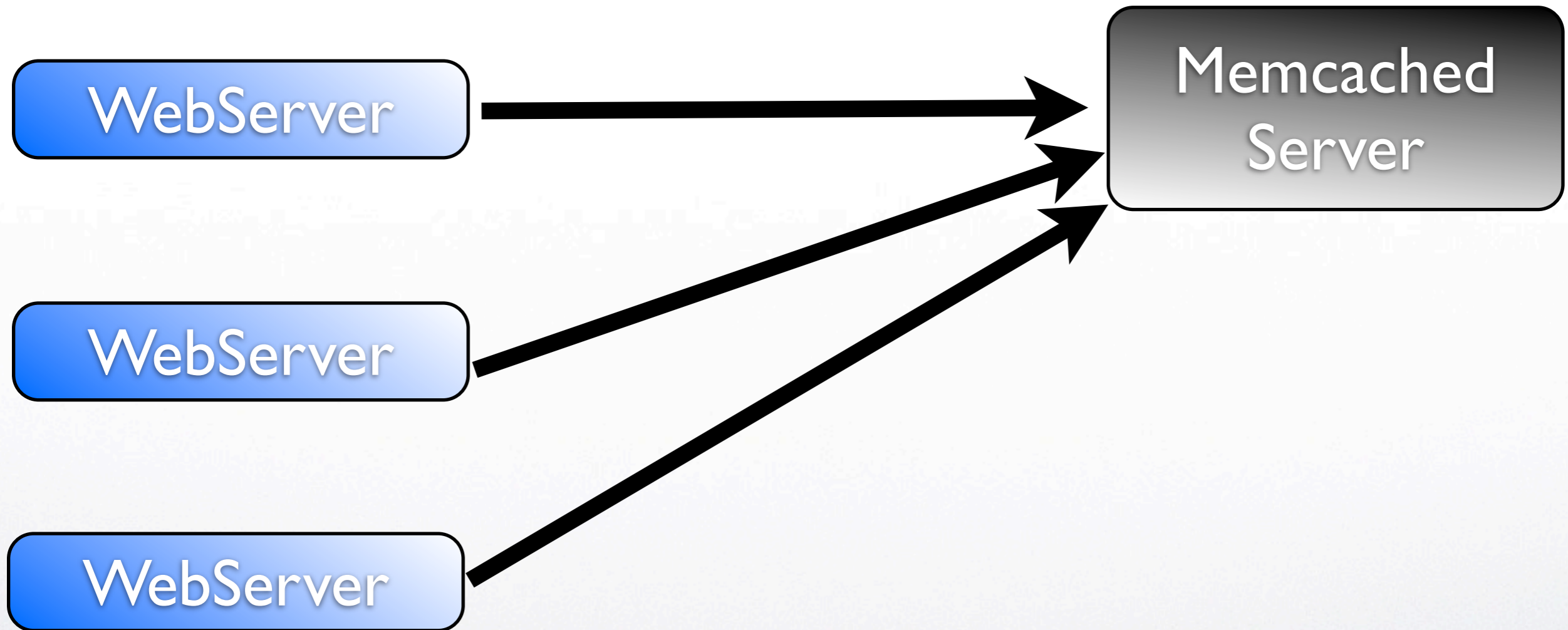
WebServer



Memcached  
Server

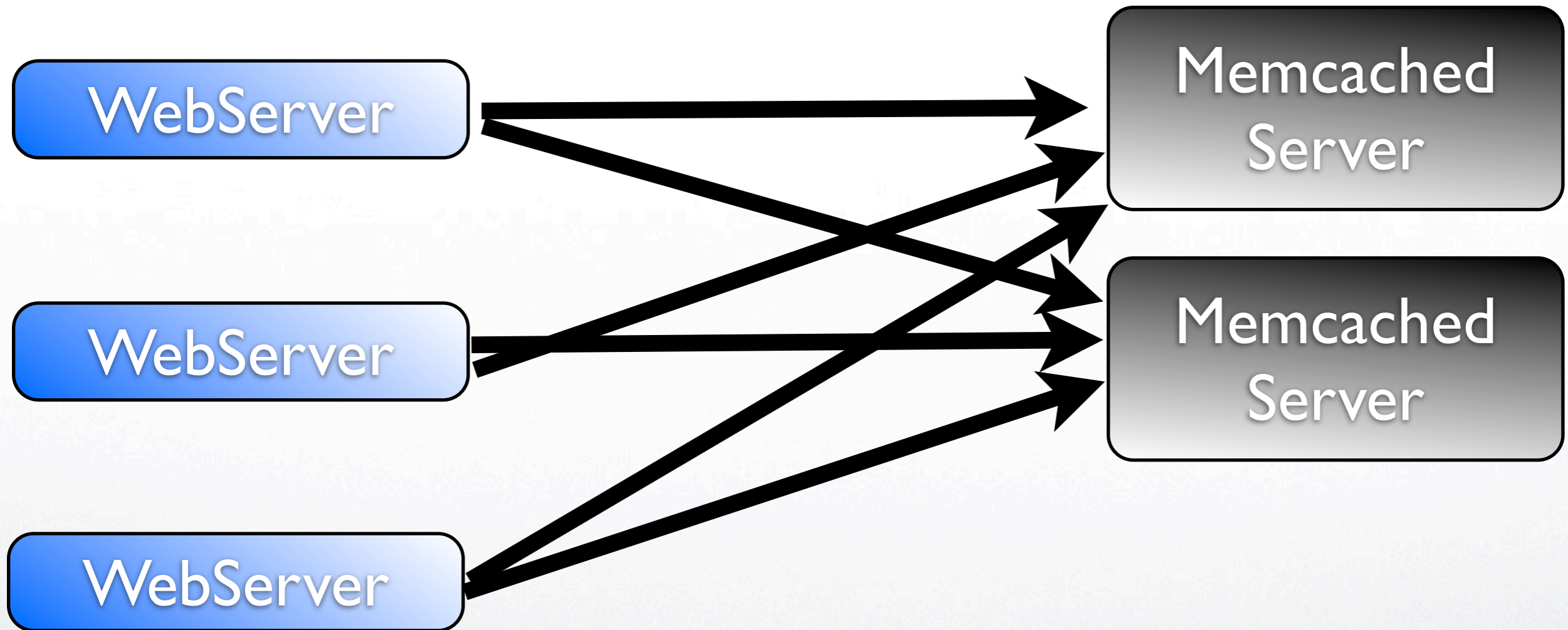


# Architecture...



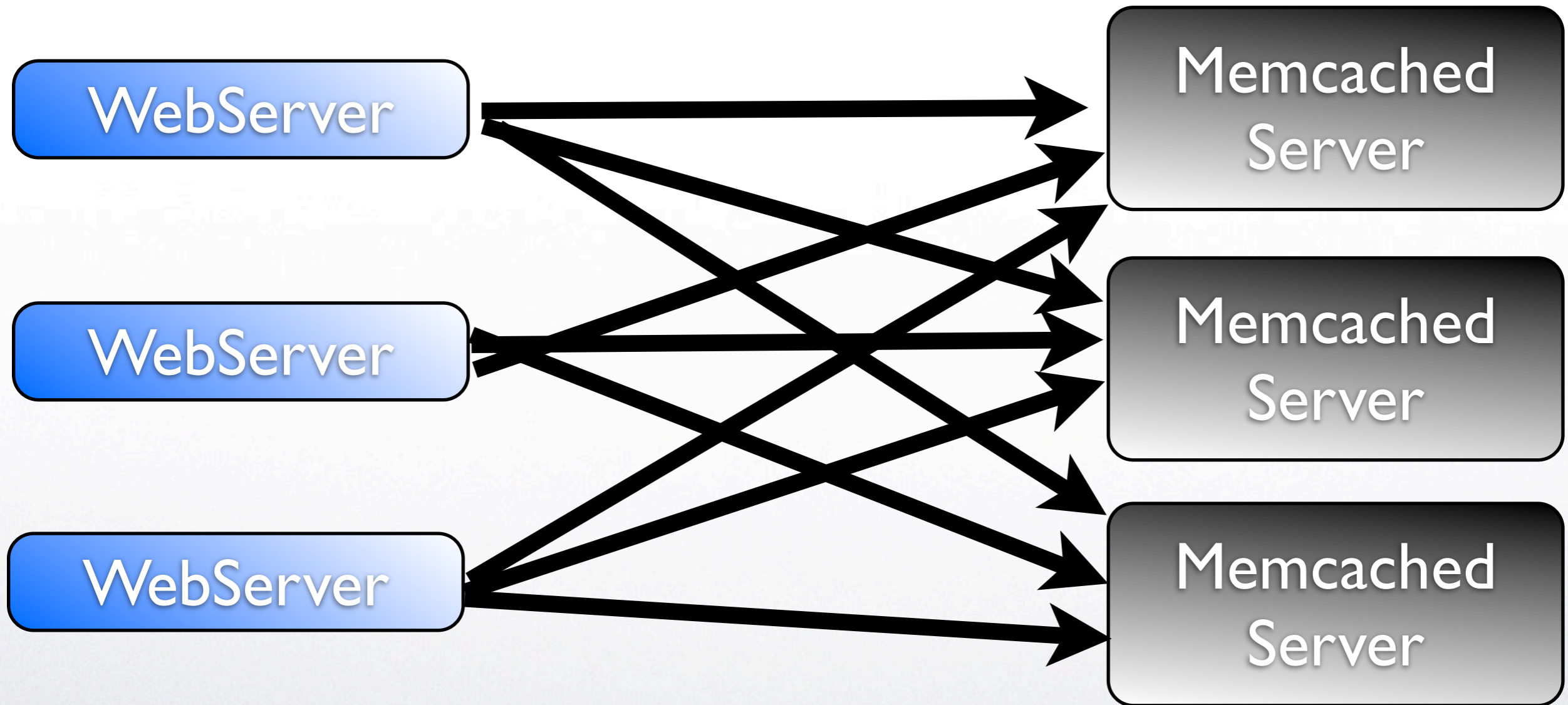
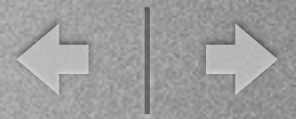


# Architecture...





# 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 is pretty simple.

```
$server_index = (crc32('key') % $num_servers);
```

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

**But what if the number of servers changes?**

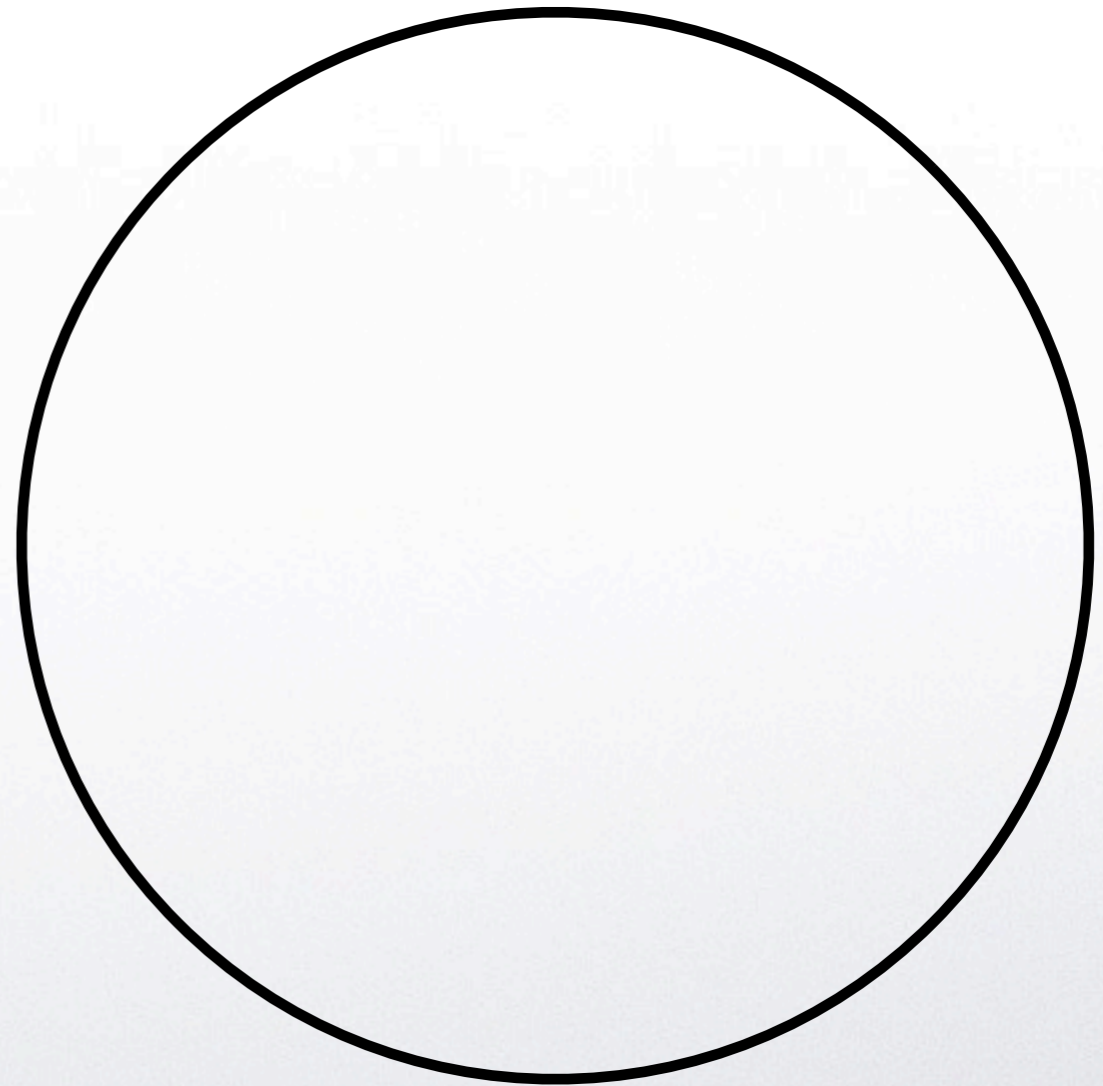




# Consistent Hashing



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

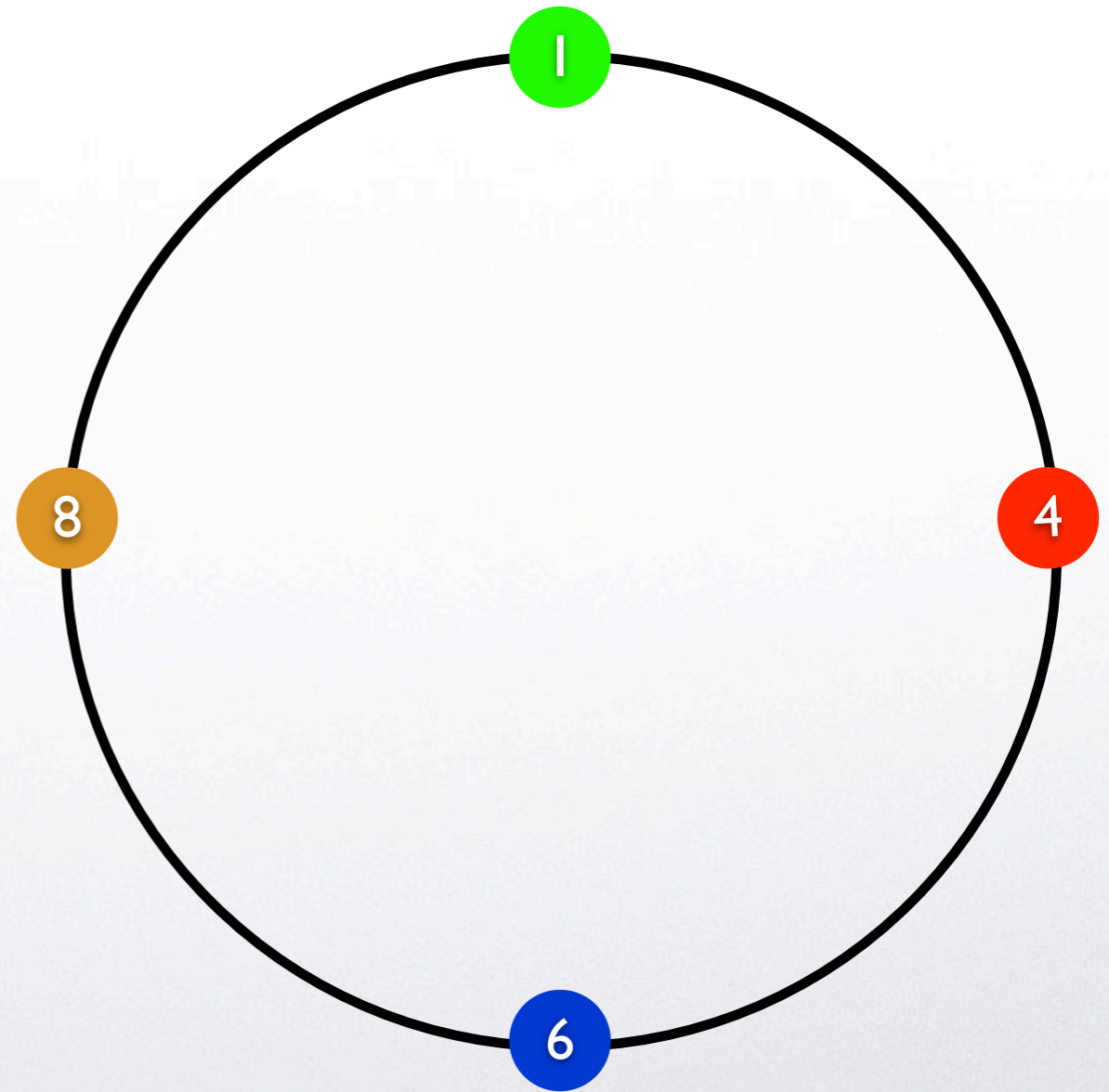




# Consistent Hashing



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



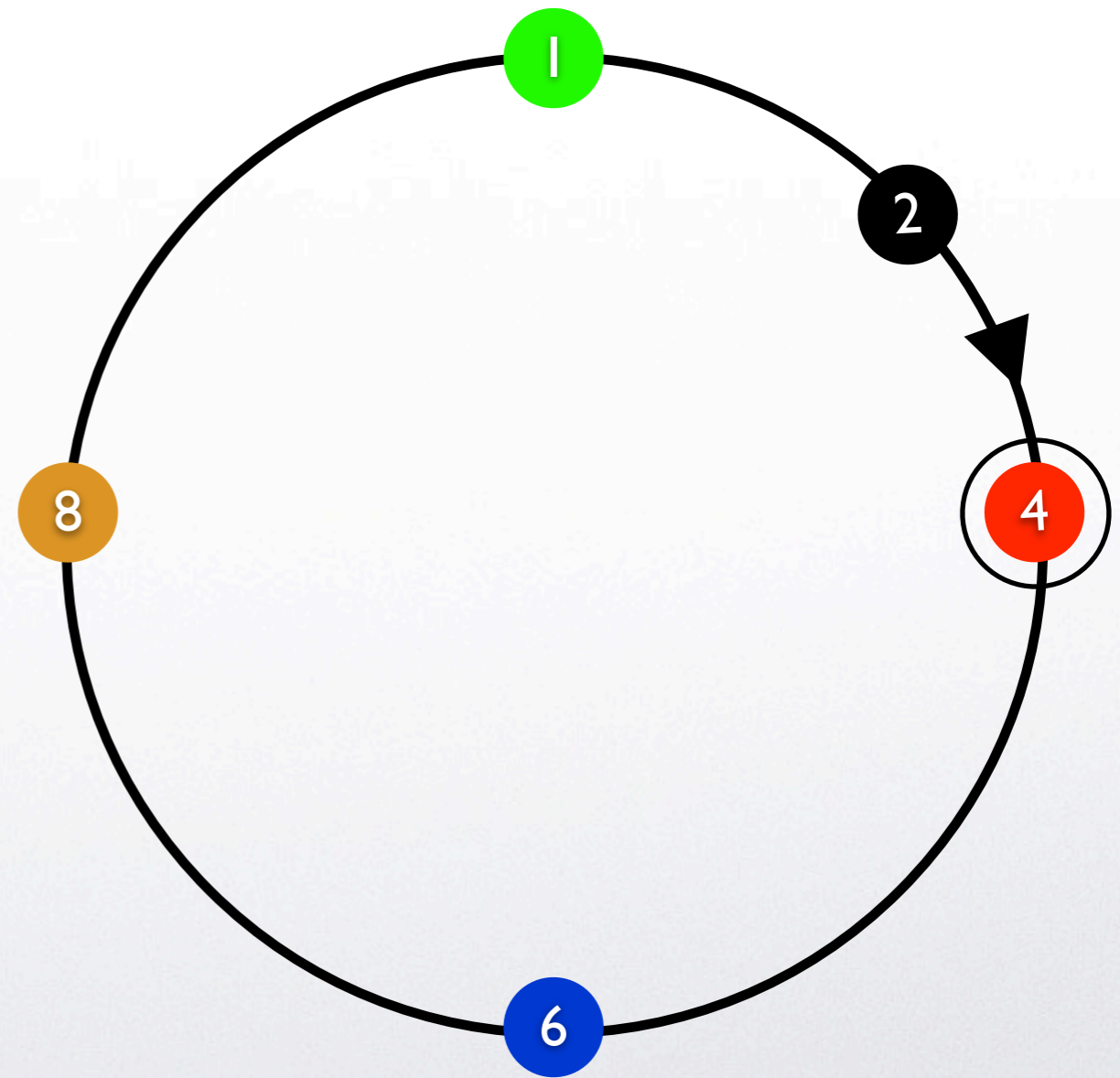


# Consistent Hashing



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');
```



# And there is more ...



The specific-server interface also supports multi-(get|set)

```
$MC = NEW MEMCACHED();  
$MC->ADDSERVERS( ... );
```

```
$KEY_SET = ARRAY('KEY1' => "FOO", 'KEY2' => ARRAY(1,2,3));
```

```
// STORE MULTIPLE KEYS AT ONCE FOR 1 HOUR
```

```
$MC->SETMULTIBYKEY('SERVER_KEY', $KEY_SET, 3600);
```

```
// GET MULTIPLE KEYS AT ONCE
```

```
$DATA = $MC->GETMULTIBYKEY('SERVER_KEY',  
                            ARRAY_KEYS($KEY_SET));
```



# 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

```
$MC = NEW MEMCACHED();  
$MC->ADDSERVER('LOCALHOST', '11211');  
  
$MC->GETDELAYED(ARRAY('KEY'));  
// PARAMETER IS AN ARRAY OF KEYS  
  
/* SOME PHP CODE THAT DOES "STUFF" */  
  
// 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.



```
$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



# Compare & Swap



Compare & Swap (CAS) is a mechanism for updating a value, unless it was changed by someone else already.

```
$cas = null;  
  
// fetch last_seen value,  
// retrieving CAS into $cas by-ref  
$value = $mem->get("last_seen", null, $cas);  
  
// update last_seen, unless updated  
// by someone else already  
$mem->cas($cas, "last_seen", time());
```



# Namespacing w/Counters



```
$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.

```
$mem->setOption(  
    Memcached::OPT_PREFIX_KEY,  
    "_" . PHP_VERSION . "_"  
);  
  
$mem->set("foo", "bar");  
// actual key is _5.3.3-p11-gentoo_foo  
  
$mem->get("foo");  
// gets value from _5.3.3-p11-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 than CPU speed + RAM.

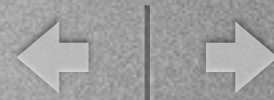
```
$MC = NEW MEMCACHED();
```

```
$MC->ADDSERVER('LOCALHOST', 11211);
```

```
// ENABLE COMPRESSION
```

```
$MC->SETOPTION(MEMCACHED::OPT_COMPRESSION, TRUE);
```





## 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



```
$MC = NEW MEMCACHED();  
$MC->ADDSERVER('LOCALHOST', 11211);
```

```
// MEMCACHED STATISTICS GATHERING  
$MC->GETSTATS();
```

```
// CLEAR ALL CACHE ENTRIES  
$MC->FLUSH();
```

```
// CLEAR ALL CACHE ENTRIES  
// IN 10 MINUTES  
$MC->FLUSH(600);
```

```
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  
  )  
)  
)
```



# Installing Memcached



Download memcached from <http://www.memcached.org> and compile it.

Download libmemcached from <http://tangent.org/552/libmemcached.html> and compile it.

`pecl install memcached` (configure, make, make install)

Enable Memcached from your `php.ini` file



# Installing Memcached



If you want the latest Memcached sources checkout Github:

<http://github.com/php-memcached-dev>



# 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.

## # LOCKING CONTROLS

# WHETHER TO ENABLE SESSION LOCK, ON BY DEFAULT

**MEMCACHED.SESS\_LOCKING**

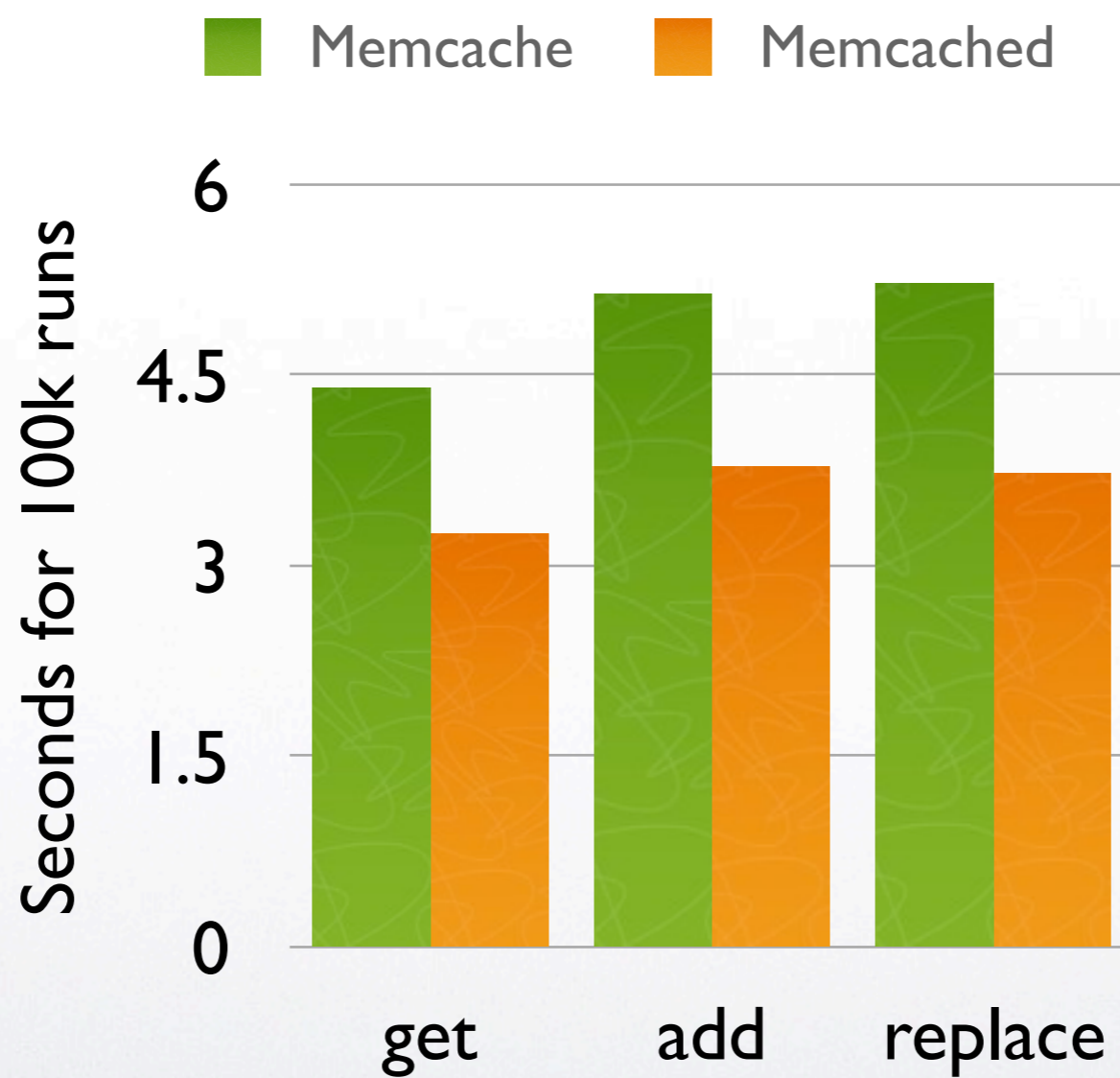
# MAXIMUM NUMBER OF MICROSECONDS TO WAIT ON A LOCK

**MEMCACHED.SESS\_LOCK\_WAIT**





# Performance







# Thank You For Listening

---

Slides will be available  
at **<http://ilia.ws>**

Please give me your feedback  
**<http://joind.in/3509>**