

# Give Your Site A Boost With Memcache

Ben Ramsey  
July 23, 2008

# Why cache?

To make it faster.

“A cache is a collection of data duplicating original values stored elsewhere or computed earlier, where the original data is expensive to fetch (owing to longer access time) or to compute, compared to the cost of reading the cache. In other words, a cache is a temporary storage area where frequently accessed data can be stored for rapid access.”

— [Wikipedia](#)

# Why cache?

- You want to reduce the number of retrieval queries made to the database
- You want to reduce the number of external requests (retrieving data from other web services)
- You want to cut down on filesystem access

# Caching options

- Flat file caching
- Caching data in the database
- MySQL 4.x query caching
- Shared memory (APC)
- RAM disk
- memcached

# What is memcached?

- Distributed Memory Object Caching System
- Caching daemon
- Developed by Danga Interactive for LiveJournal.com
- Uses RAM for storage
- Acts as a dictionary of stored data with key/value pairs

# Is memcached fast?

- Stored in memory (RAM), not on disk
- Uses non-blocking network I/O (TCP/IP)
- Uses libevent to scale to any number of open connections
- Uses its own slab allocator and hash table to ensure virtual memory never gets externally fragmented and allocations are guaranteed  $O(1)$



# General usage

1. Set up a pool of memcached servers
2. Assign values to keys that are stored in the cluster
3. The memcache client hashes the key to a particular machine in the cluster
4. Subsequent requests for that key retrieve the value from the memcached server on which it was stored
5. Values time out after the specified TTL

# Memcached principles

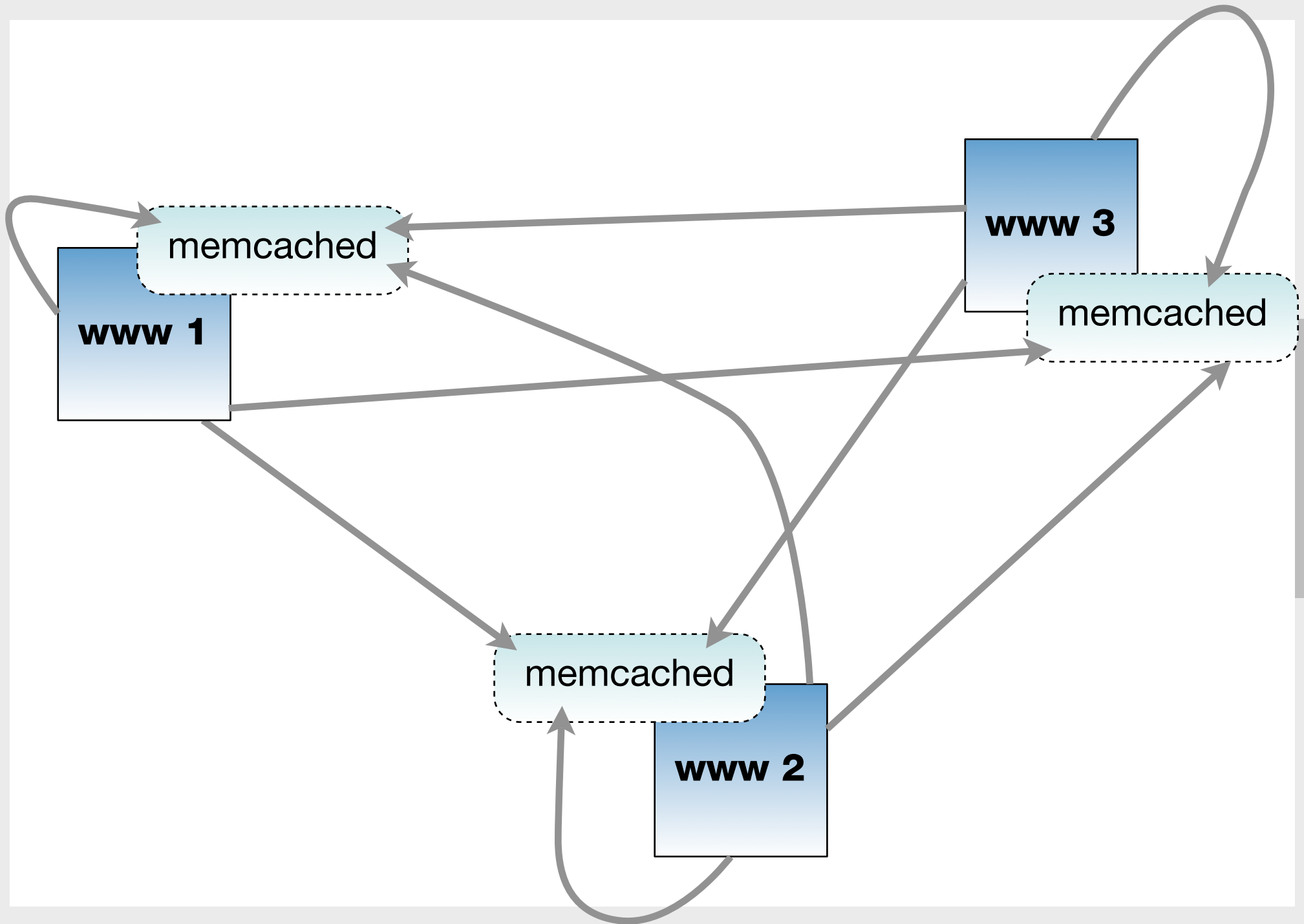
- It's a non-blocking server
- It is not a database
- It does not provide redundancy
- It doesn't handle failover
- It does not provide authentication

# Memcached principles

- Data is not replicated across the cluster
- Works great on a small and local-area network
- A single value cannot contain more than 1MB of data
- Keys are strings limited to 250 characters

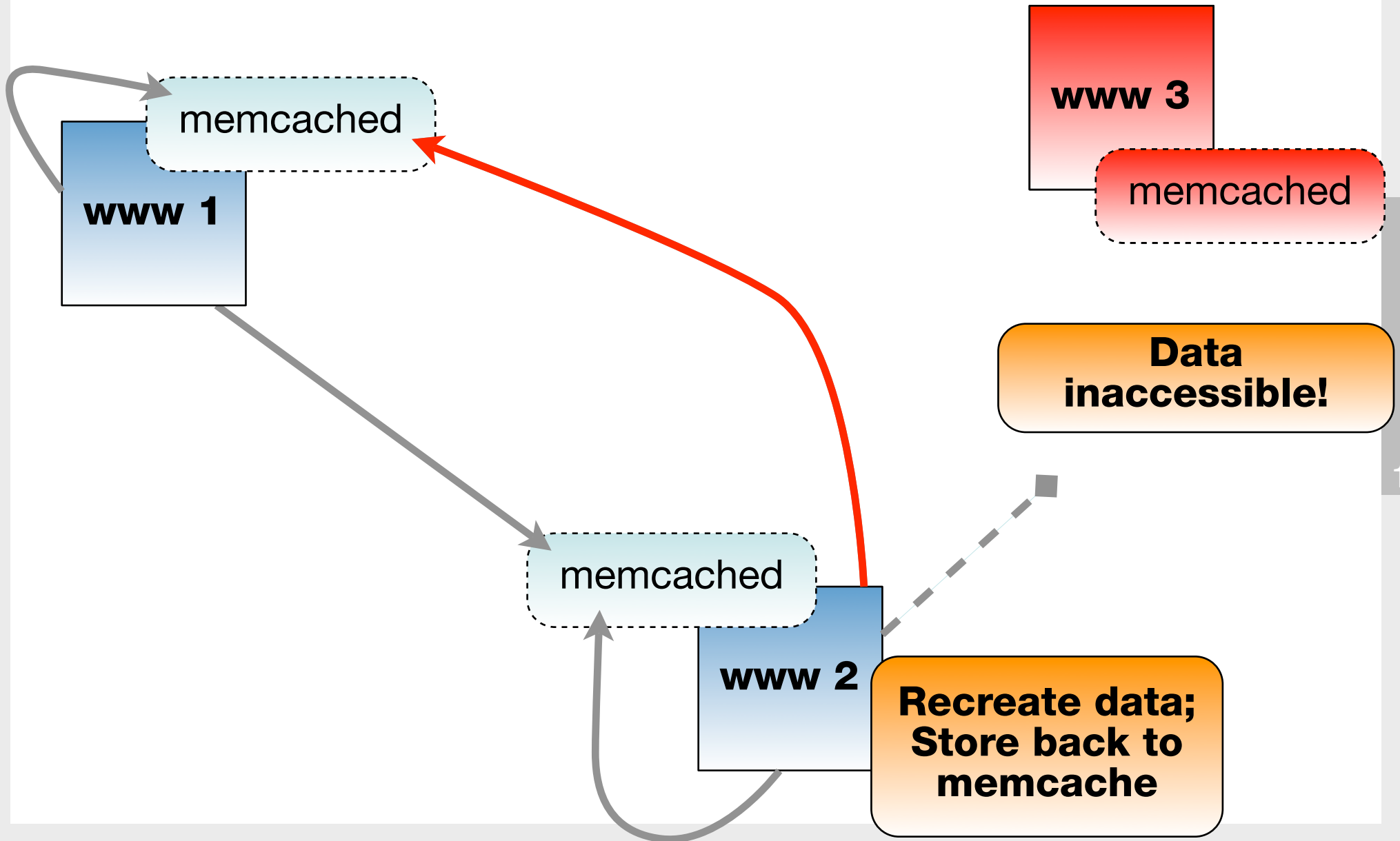
# Storing data in the pool

- Advantage is in scalability
- To fully see the advantage, use a “pool”
- memcached itself doesn't know about the pool
- The pool is created by and managed from the client library



# Deterministic failover

- When one server goes down, the system fails over to another server in the pool
- Memcached does not provide this
- Some memcache clients provide failover
- If you can't find the data in memcache, eat the look-up cost and retrieve from your data source again, storing it back to the cache



# The memcached protocol API

- Storage commands:  
set, add, replace, append, prepend, cas
- Retrieval command: get, gets
- Deletion command: delete
- Increment/decrement: incr, decr
- Other commands:  
stats, flush\_all, version, verbosity,  
quit



```
$> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.

set foobar 0 0 15
This is a test.
STORED

get foobar
VALUE foobar 0 15
This is a test.
END

quit
Connection closed by foreign host.
$>
```

# Setting it up

- <http://danga.com/memcached/>
- `$> ./configure; make; make install`
- `$> memcached -d -m 2048 -p 11211`
- Done!
- Windows port of v1.2.4 at <http://www.splinedancer.com/memcached-win32/>

# Memcached clients

- Perl, Python, Ruby, Java, C#
- C (libmemcached)
- PostgreSQL (access memcached from procs and triggers)
- MySQL (adds memcache\_engine storage engine)
- PHP (pecl/memcache)

# pecl/memcache

- The PHP client for connecting to memcached and managing a pool of memcached servers
- <http://pecl.php.net/package/memcache>
- `$> pecl install memcache`
- Stable: 2.2.3  
Beta: 3.0.1

```
<?php
```

```
$memcache = new MemcachePool();  
$memcache->addServer('192.168.1.10', 11211);  
$memcache->addServer('192.168.1.11', 11211);  
$memcache->addServer('192.168.1.12', 11211);  
  
if (($fooObject = $memcache->get('key')) === false) {  
    $tmpObject = new stdClass;  
    $tmpObject->foo = 'bar';  
    $tmpObject->baz = 'quz';  
  
    $fooObject = $tmpObject;  
    $memcache->set('key', $tmpObject, 0, 300);  
}  
  
// Do stuff with $fooObject
```

# Features of pecl/memcache

- memcache.allow\_failover
- memcache.hash\_strategy
- memcache.hash\_function
- memcache.protocol
- memcache.redundancy
- memcache.session\_redundancy

# pecl/memcache interface

- MemcachePool::connect()
- MemcachePool::addServer()
- MemcachePool::setServerParams()
- MemcachePool::get()
- MemcachePool::add()
- MemcachePool::set()
- MemcachePool::replace()
- MemcachePool::cas()

# pecl/memcache interface

- MemcachePool::append()
- MemcachePool::prepend()
- MemcachePool::delete()
- MemcachePool::increment()
- MemcachePool::decrement()
- MemcachePool::setFailureCallback()



# Key hashing

- Keys longer than 250 characters are truncated without warning
- Good practice to hash your key (with MD5 or SHA) at the userland level to ensure long keys don't get truncated
- Keys are “global”
- Use something to uniquely identify keys, e.g. a method signature or an SQL statement

# Object serialization

- Objects are serialized before being stored to memcache:

```
get key
```

```
VALUE key 1 59
```

```
0:8:"stdClass":2:{s:3:"foo";s:3:"bar";s:  
3:"baz";s:3:"quz";}
```

```
END
```

- Extension unserializes them before returning the object
- Only objects that can be serialized safely can be stored to memcache, i.e. problems with DOM, SimpleXML, etc.

# Redundancy and failover

- memcache.redundancy & memcache.session\_redundancy
- Implement redundancy at the userland level?
- Again, memcache is not a database

# Extending MemcachePool

- Implement global values vs. page-specific values
- Ensure a single instance of the MemcachePool object
- Do complex key hashing, if you so choose
- Set a default expiration for all your data
- Add all of your servers upon object instantiation

# Database techniques

- Create a wrapper for `mysql_query()` that checks the cache first and returns an array of database results
- Extend PDO to store results to the cache and get them when you execute a statement

# Database techniques

- For large datasets, run a scheduled query once an hour and store it to the cache
- Please note: memcached can store arrays, objects, etc., but it cannot store a resource, which some database functions (e.g. `mysql_query()`) return

# Session storage

- As of 2.1.1, you can set the session save handler as “memcache” and all will work automagically

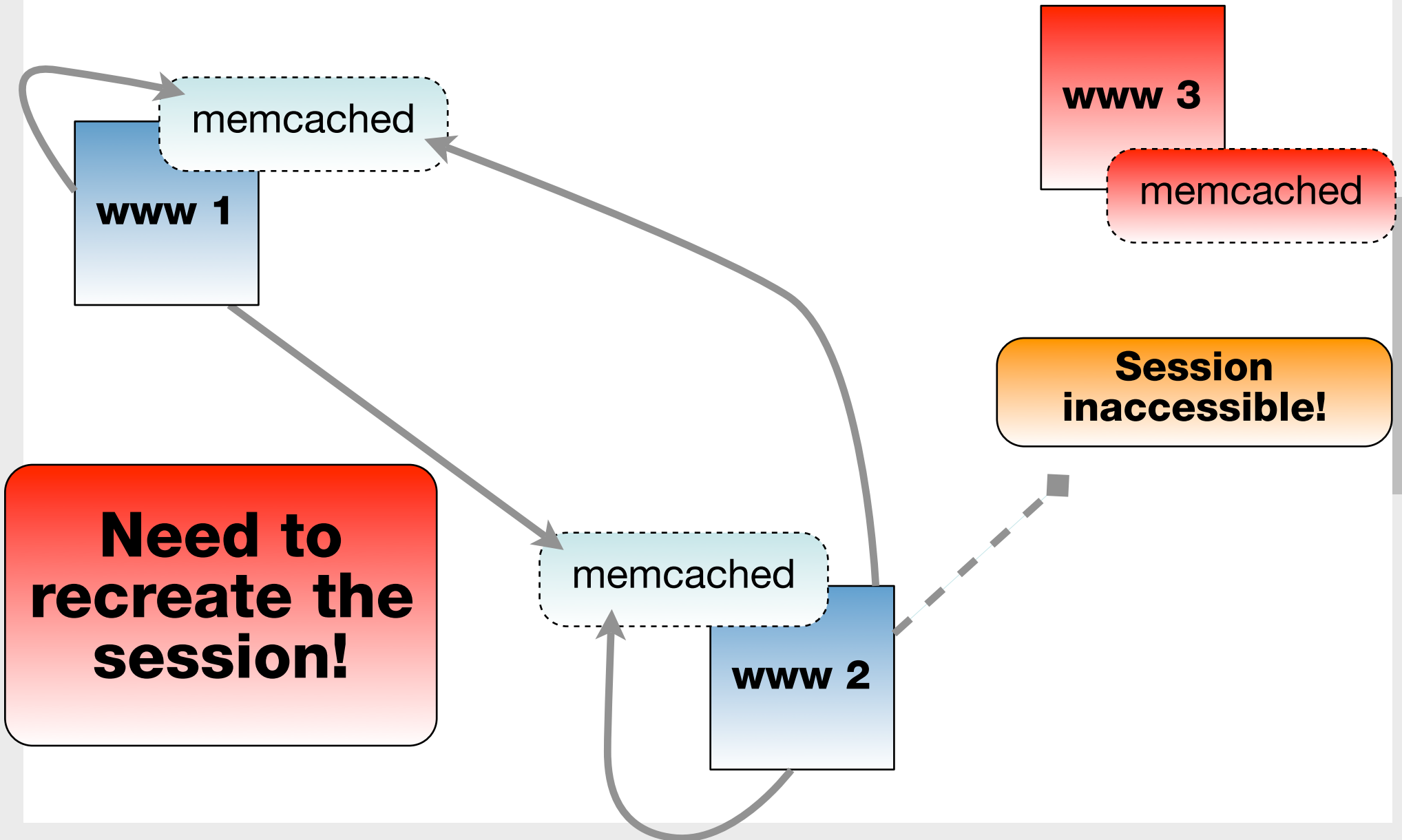
```
session.save_handler = memcache
```

```
session.save_path = "tcp://
```

```
192.168.1.10:11211,tcp://
```

```
192.168.1.11:11211,tcp://192.168.1.12:11211"
```

- Store sessions to both the database and memcache
- Write your own session handler that stores to the database and memcache





A demonstration...

## For more information...

<http://danga.com/memcached/>

<http://pecl.php.net/package/memcache>

<http://www.socialtext.net/memcached/>

# Thank You

Slides available for download at [benramsey.com](http://benramsey.com).

Ben Ramsey

Software Architect

Schematic

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

# We're Hiring!

It goes without saying: Schematic is only as good as the people who work here. That's why we're so particular about recruiting, training, nurturing, and retaining the very best people in our field.

If you have digital expertise (technical, creative, managerial—or something else entirely), enthusiasm, curiosity, and the ability to collaborate with others, we'd love to hear from you.

Please visit <http://www.schematic.com/> for more information.