international PHP 2007 conference - spring edition -

Give Your Site a Boost With memcached

Ben Ramsey

About Me

- Proud father of 3-month-old Sean
- Organizer of Atlanta PHP user group
- Founder of PHP Groups
- Founding principal of PHP Security Consortium
- Original member of PHPCommunity.org
- Author, Speaker, & Blogger
- Software Architect at Schematic

memcached:
Distributed Memory Object Caching System

- Caching daemon
- Developed by Danga Interactive for LiveJournal.com
- Uses memory for storage
- Accessed via a simple API: get, set, add, and replace
- Acts as a dictionary of stored data/objects with key/value pairs

Why Not Use...

- Database...
    - ACID-compliant databases have blocking queries
    - For non-ACID DBs, reading threads block on writing threads
- Shared memory...
    - Cache is duplicated multiple times, once for each thread

Why Not Use...

- MySQL 4.x query caching...
  - Cache is destroyed on every change, 32-bit servers limited to 4GB of virtual memory, not an object cache
- Database replication...
  - You can spread reads, but not writes; you must keep adding slaves to make up for the resource consumption for writes

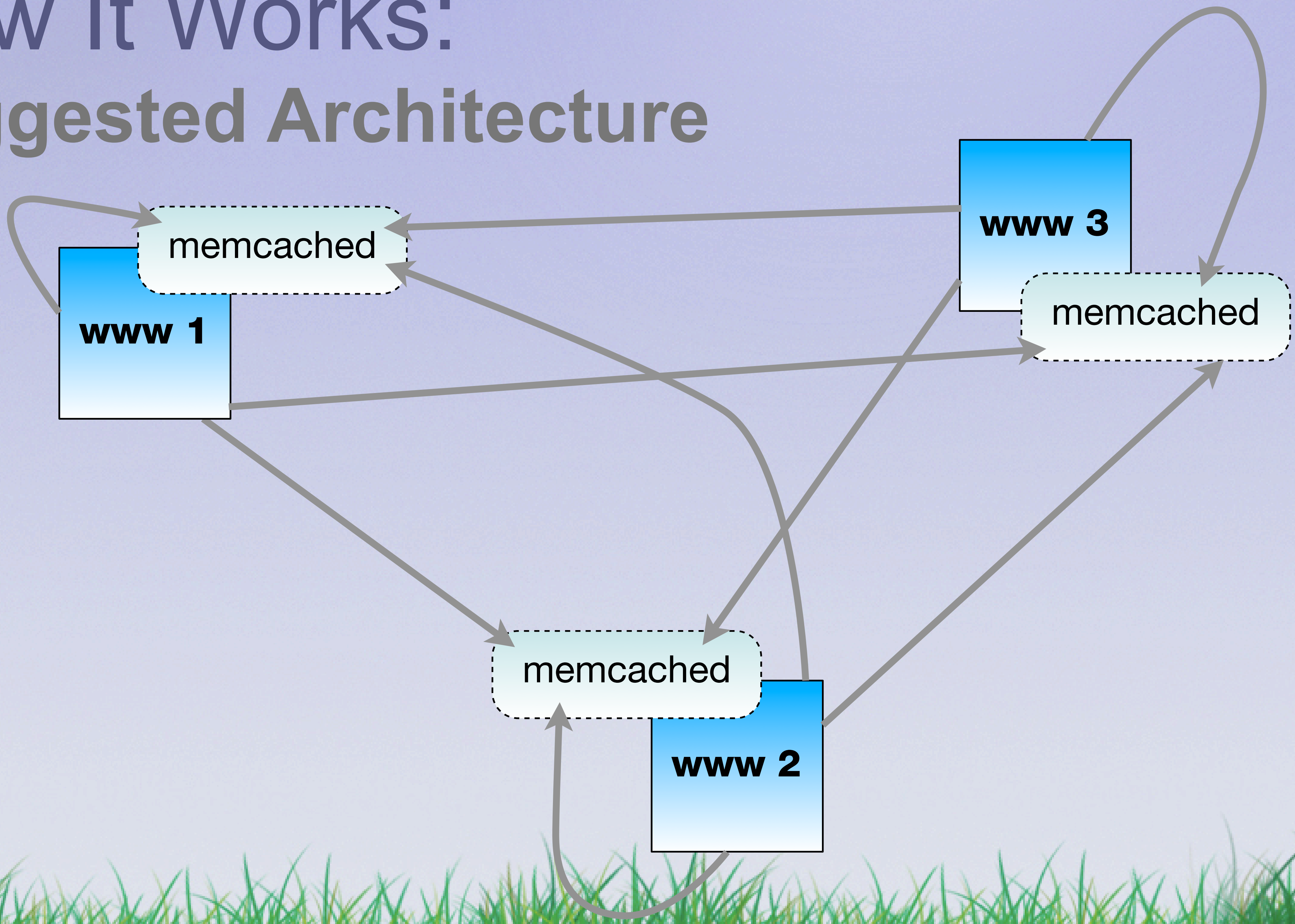
Is memcached Fast?

Short Answer: Very

- Stored in memory, not on disk
- Uses non-blocking network I/O
- Uses libevent to scale to any number of open connections
- refcounts internal objects (so objects can be in multiple states to multiple clients)
- Uses its own slab allocator and hash table so virtual memory never gets externally fragmented and allocations are guaranteed O(1)

How It Works:

Suggested Architecture



How It Works:

Writing To the memcached Cluster

1. Set up a pool/cluster of memcached servers
2. Assign values to keys that are stored in the cluster
3. The memcached API 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

How It Works:
Things To Keep In Mind

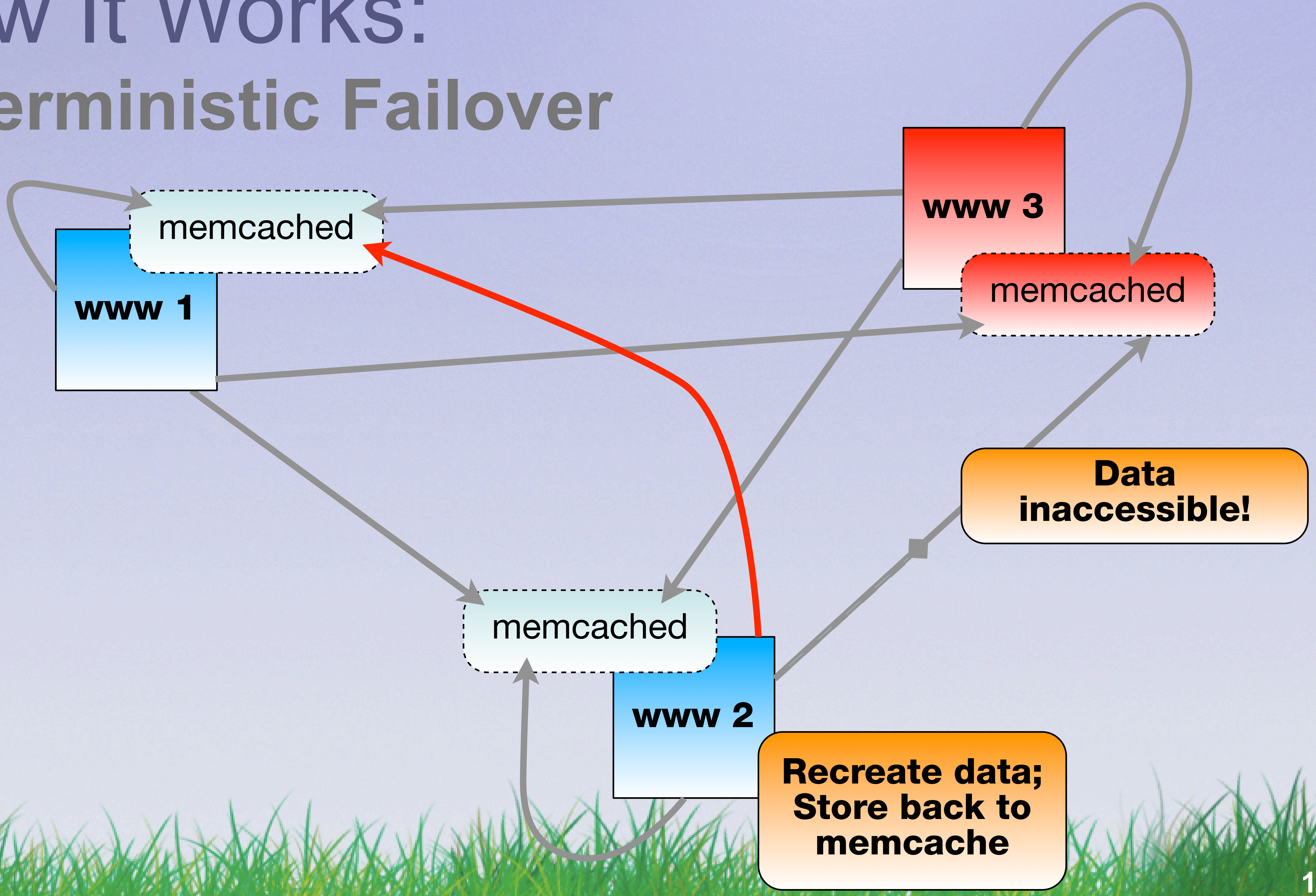
- Data is not replicated across the cluster
  - If N machines participate, there is 1/N chance the cached item resides on the local machine
  - Larger the cluster, more probability of cache lookup visiting a remote machine
  - If a machine goes down, the cached data is lost (recreate it and store again)

How It Works:
Things To Keep In Mind

- Works great on a small and local-area network
  - Connections between machines are fast
  - Can artificially segment memcached clusters if the network connection is too expensive
- A single value cannot contain more than 1MB of data
- Keys are strings limited to 256 characters

How It Works:

Deterministic Failover



Sounds Complicated...

Is It Difficult To Use?

∏ **Not at all!**

Setting Up

memcached

- Download from http://www.danga.com/memcached/
  - `./configure; make; make install`
  - `memcached -d -m 2048 -p 11211`
  - Done!
    - Running in the background as a daemon
    - Uses up to 2GB of memory for storage
    - Connect to memcached on port 11211 of the machine
    - Specify -l to tell it what IP address to listen on (use 192.168.x.x to accept only local connections)

What About Windows?

memcached Is There, Too!

- Download from http://jehiah.cz/projects/memcached-win32/
- Unzip to `c:\memcached`
- Install with `c:\memcached\memcached.exe -d install`
- Start from Microsoft Management Console or with `c:\memcached\memcached.exe -d start`
- By default, it runs on port 11211

pecl/memcache

Using memcached With PHP

- ***NIX:** Download from http://pecl.php.net/package/memcache
  - Read manual for installing PECL extensions
- **Windows:** Download from http://pecl4win.php.net/ext.php/php_memcache.dll
- Or... install from the command line with:
  `pecl install memcache`
- Enable in `php.ini`

Basic Usage of
pecl/memcache

```
$memcache = new Memcache();
$memcache->addServer('192.168.0.10', 11211); // www 1
$memcache->addServer('192.168.0.11', 11211); // www 2
$memcache->addServer('192.168.0.12', 11211); // www 3

if (($object = $memcache->get('key')) === FALSE)
{
    $tmp_object = new stdClass;
    $tmp_object->str_attr = 'test';
    $tmp_object->int_attr = 123;

    $object = $tmp_object;
    $memcache->set('key', $tmp_object, FALSE, 300);
}

// Do stuff with $object
```

Techniques:
Avoiding Key Collision

- memcached is a global cache; keys are global
  - ‘foo’ set on page X has the same data when you retrieve it from memcache on page Y
- Keys cannot be longer than 256 characters
  - If a key is longer than this limit (i.e. using an SQL statement, it will be truncated)
  - Use an MD5 hash of your key string as the key to avoid collisions from truncated keys

Techniques:
Avoiding Key Collision

- Differentiate between “global” and “page” data by hashing the file name into the key for page-specific data:
  - `md5(__FILE__ . ‘keyString’)`
- For data stored from a particular method, create a key that is a hash of the method name and args:
  - `md5(__METHOD__ . $args)`
- Use a hash of an SQL statement for DB results

Techniques:

Extending pecl/memcache

- Implement global values vs. page-specific values
- Ensure a single instance of the Memcache 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
- What else?

```php
class MyMemcache extends Memcache
{
   protected $expire = 1440;

   // Singleton instance
   private static $instance;

   private function __construct() {}

   // Returns singleton instance
   public static function getInstance($server, $port = '11211', $persist = TRUE)
   {
      if (!isset(self::$instance))
      {
         $c = __CLASS__;
         self::$instance = new $c;
         self::$instance->addServer($server, $port, $persist);
      }
      return self::$instance;
   }
```

```
private function makeKey($key, $global = FALSE)
{
    if ($global)
    {
        return md5($key);
    }
    return md5($key . $_SERVER['SCRIPT_FILENAME']);
}

public function set($key, $var, $global = FALSE, $flag = MEMCACHE_COMPRESSED)
{
    return parent::set($this->makeKey($key, $global), $var, $flag, $this->expire);
}
```

```
    public function get($key, $global = FALSE)
    {
        if (is_array($key))
        {
            foreach ($key as &$k)
            {
                $k = $this->makeKey($k, $global);
            }
        }
        else
        {
            $key = $this->makeKey($key, $global);
        }

        return parent::get($key);
    }
}
```

Techniques:

pecl/memcache & Database Queries

- Create a wrapper for `mysql_query()` that checks the cache first and returns an array of database results (storing those results to the cache if it queries the database)
- For large data sets that don't update often, run a scheduled query once an hour and store it to the cache (i.e. Top 10/Top 100 lists, Most Popular, etc.)

Techniques:
pecl/memcache & Database Queries

- Extend PDO to store results to the cache and get them when you execute a statement
- Use a DAO instead of extending PDO and create a hashed key from the `__METHOD__` and parameters
- Please note: memcached can store arrays, objects, etc. without the need to serialize, but it cannot store a resource, which some database functions (e.g. `mysql_query()`) return

Techniques:
Using memcached As the Session Store

- As of pecl/memcache 2.1.1, you can set the session save handler in `php.ini` as “memcache” and all will work automagically:

    -
      ```
      session.save_handler = memcache
      session.save_path = "tcp://192.168.0.10:11211,
      tcp://192.168.0.11:11211,tcp://192.168.0.12:11211"
      ```
- Or you can override the session handler with a session manager class and do the same...

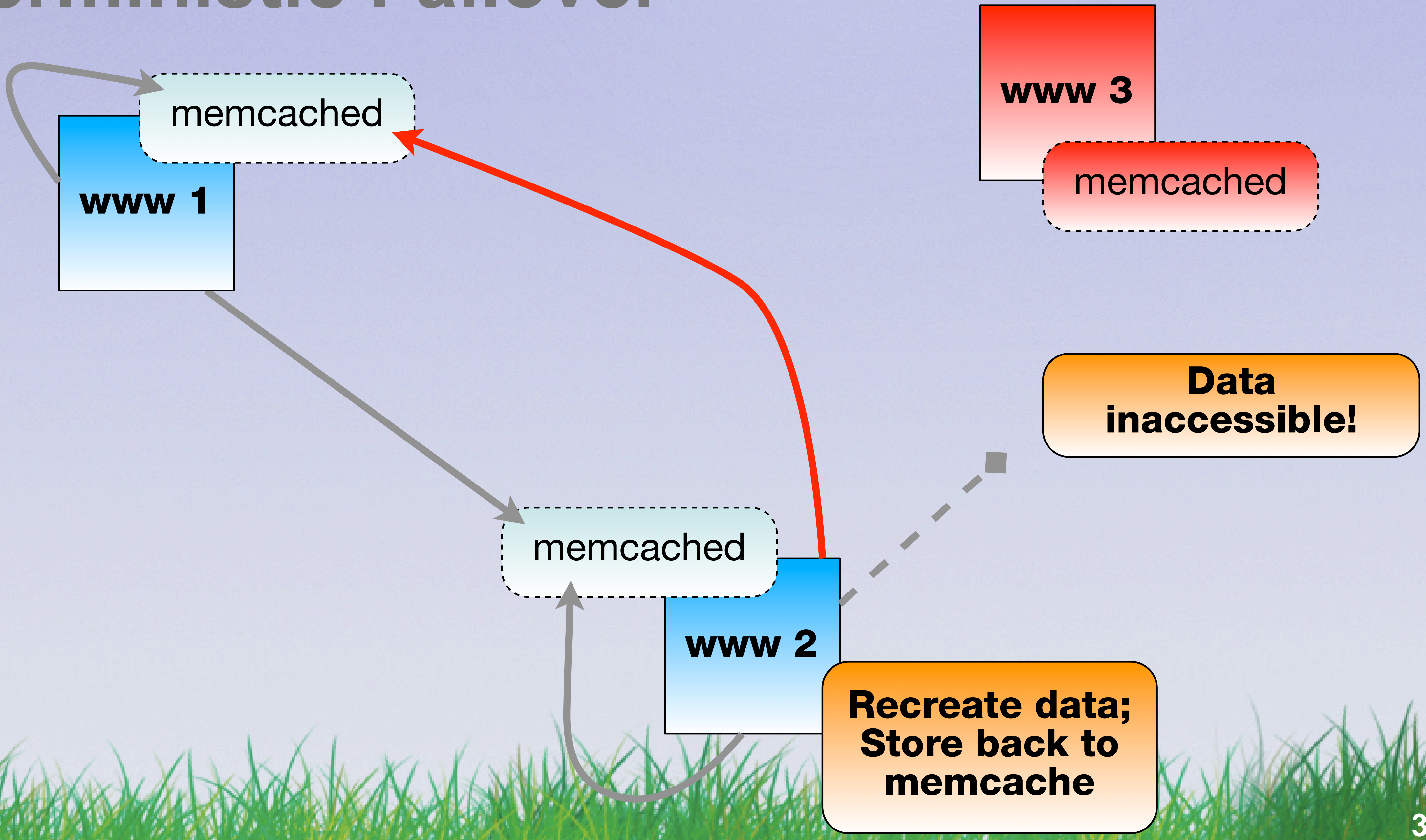
```
class MySessionManager
{
   // Prefix added to session identifier for memcache key
   private $session_prefix = 'MY_SESSION';

   // Life time of session data
   private $life_time;

   // Memcache object for the session store
   private $session_store;
```

```php
public function __construct()
{
    // Read the maxlifetime setting from PHP
    $this->life_time = ini_get('session.gc_maxlifetime');

    // Set up session store object
    $this->session_store = MyMemcache::getInstance();

    $this->session_store->addServer('192.168.0.10', '11211');
    $this->session_store->setExpire($this->life_time);

    // Register this object as the session handler
    session_set_save_handler(
        array(&$this, 'open'),
        array(&$this, 'close'),
        array(&$this, 'read'),
        array(&$this, 'write'),
        array(&$this, 'destroy'),
        array(&$this, 'gc')
    );
}
```

```
//  Opens session (we do nothing with it)
public function open($save_path, $session_name)
{
   return TRUE;
}

// Closes session (we do nothing with it)
public function close()
{
   return TRUE;
}

// Returns session data from memcached server
public function read($id)
{
   $key = $session_prefix . $id;
   return $this->session_store->get($key, TRUE);
}

// Saves session data to memcached server
public function write($id, $data)
{
   $key = $session_prefix . $id;
   return $this->session_store->set($key, $data, TRUE);
}
```

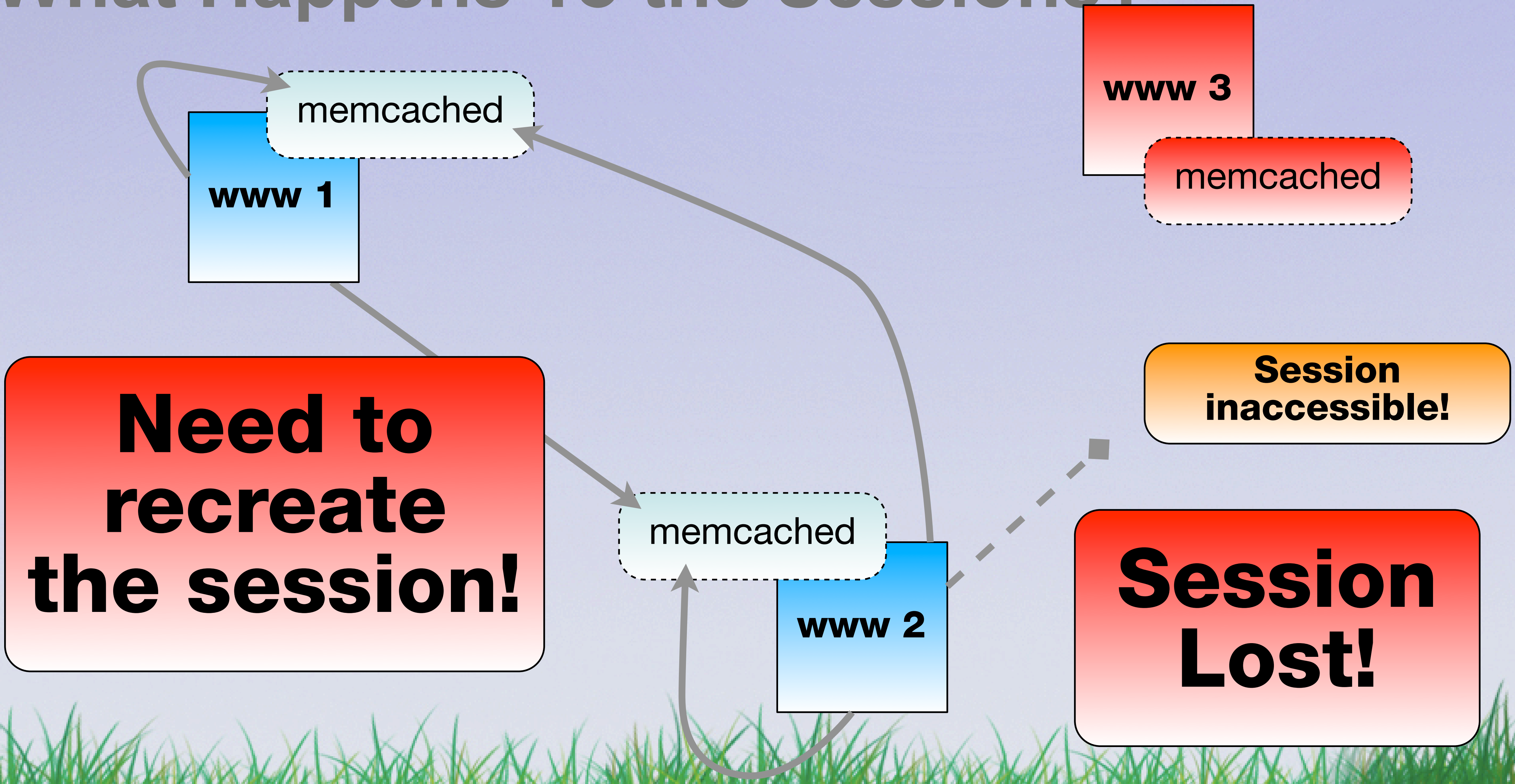
```
    // Remove data from the session
    public function destroy($id)
    {
        $key = $session_prefix . $id;
        return $this->session_store->delete($key, TRUE);
    }

    // Garbage collection (we do nothing with it)
    public function gc($maxlifetime)
    {
        return TRUE;
    }
}
```

Recall Server Failure:
Deterministic Failover



Failover and Sessions:

What Happens To the Sessions?



memcached

Usage Suggestions

- Application configuration
    - Load run-time configuration, store it to memcached with an expire time of 24 hours (or something like that)
    - You don't have to read the config file on every page load, which has a lot of overhead

memcached
Usage Suggestions

- DB access object
    - This object rarely changes and is used for accessing the database
    - Store this object to memcached with an expire time of 24 hours
    - Your DB class file doesn't need to be loaded on every request

memcached
Usage Suggestions

- Temporary DB tables
  - Instead of using the DB to store temporary information, use memcached
  - Reduces the need for queries to the DB

memcached

Usage Suggestions

- XML files
  - If you have XML files that need to be read often, store the content of the file to memcached when you first read it
  - If the files do not change often, set your expire time high (1 - 24 hours)
  - If the files are over 1MB, split them up so that you can store the entire file in memcached across keys
  - Better yet, don't store the XML files and, instead, store the DOM object representation of the XML file in memcache

memcached
Usage Suggestions

- Templates
  - Store your templates/views to the cache so that you don't have to read them on every page request
  - Templates do not change often, so you can set the TTL high
  - Store Smarty templates or straight PHP templates and eval() the PHP template at runtime

Resources

For More Information

- Memcached: http://www.danga.com/memcached/
- For Windows: http://jehiah.cz/projects/memcached-win32/
- PHP Extension: http://pecl.php.net/package/memcache
- PHP Manual: http://www.php.net/memcache

- Slides: http://benramsey.com/archives/ipcse07-slides/
- My company: http://www.schematic.com/