

all the little pieces

distributed systems with PHP

Andrei Zmievski @ Digg
Dutch PHP Conference @ Amsterdam

Who is this guy?

- Open Source Fellow @ Digg
- PHP Core Developer since 1999
- Architect of the Unicode/18n support
- Release Manager for PHP 6
- Twitter: @a
- Beer lover (and brewer)

Why distributed?

- Because Moore's Law will not save you
- Despite what DHH says

Share nothing

- Your Mom was wrong
- No shared data on application servers
- Distribute it to shared systems

distribute...

- memory (memcached)
- storage (mogilefs)
- work (gearman)

Building blocks

- GLAMMP - have you heard of it?
- Gearman + LAMP + Memcached
- Throw in Mogile too



memcached

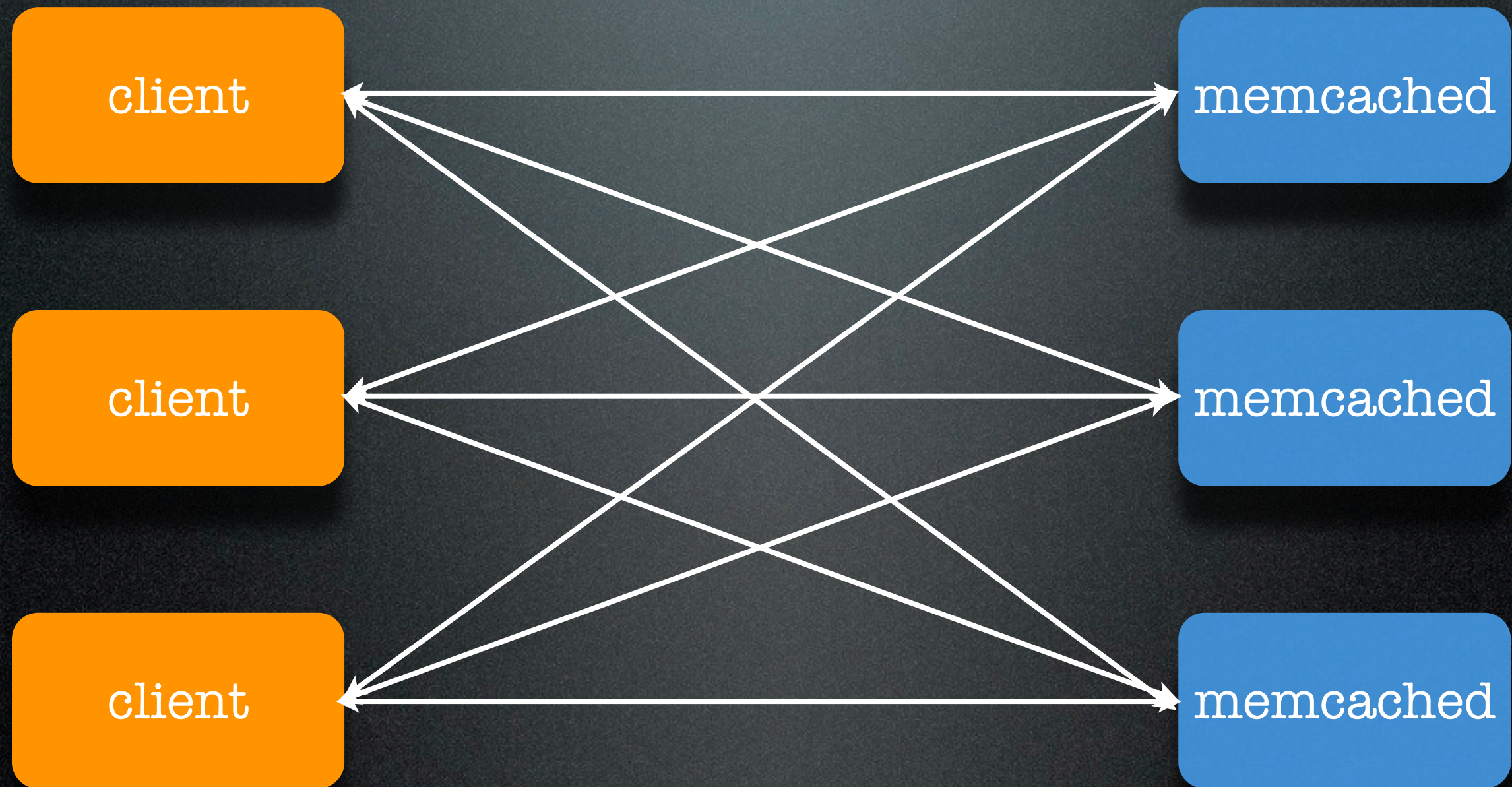
background

- created by Danga Interactive
- high-performance, distributed memory object caching system
- sustains Digg, Facebook, LiveJournal, Yahoo!, and many others
- if you aren't using it, you are crazy

background

- Very fast over the network and very easy to set up
- Designed to be transient
- You still need a database

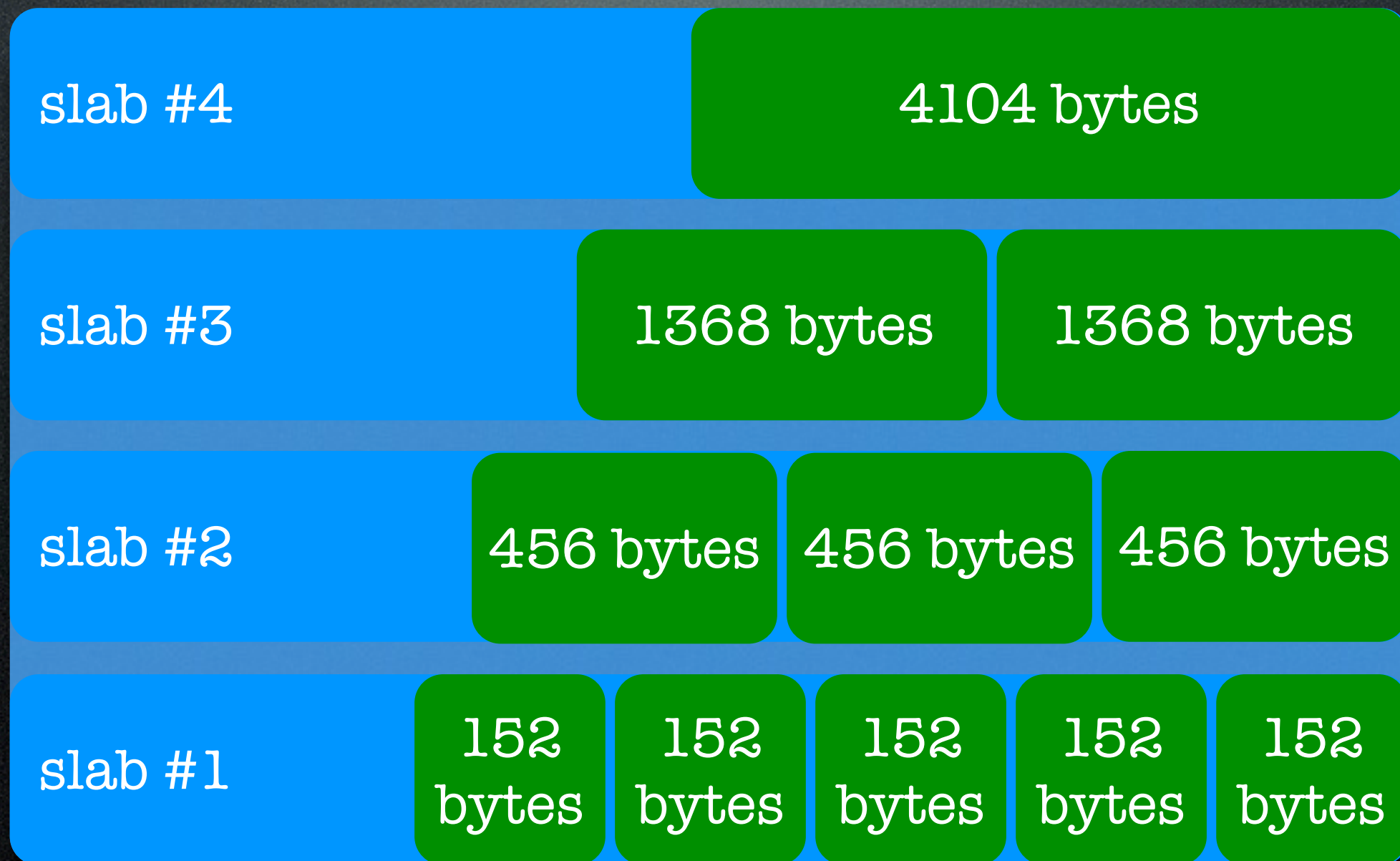
architecture



architecture



memcached



memory architecture

- memory allocated on startup, released on shutdown
- variable sized slabs (30+ by default)
- each object is stored in the slab most fitting its size
- fragmentation can be problematic

memory architecture

- items are deleted:
 - on set
 - on get, if it's expired
 - if slab is full, then use LRU

applications

- object cache
- output cache
- action flood control / rate limiting
- simple queue
- and much more

PHP clients

- a few private ones (Facebook, Yahoo!, etc)
- `pecl/memcache`
- `pecl/memcached`

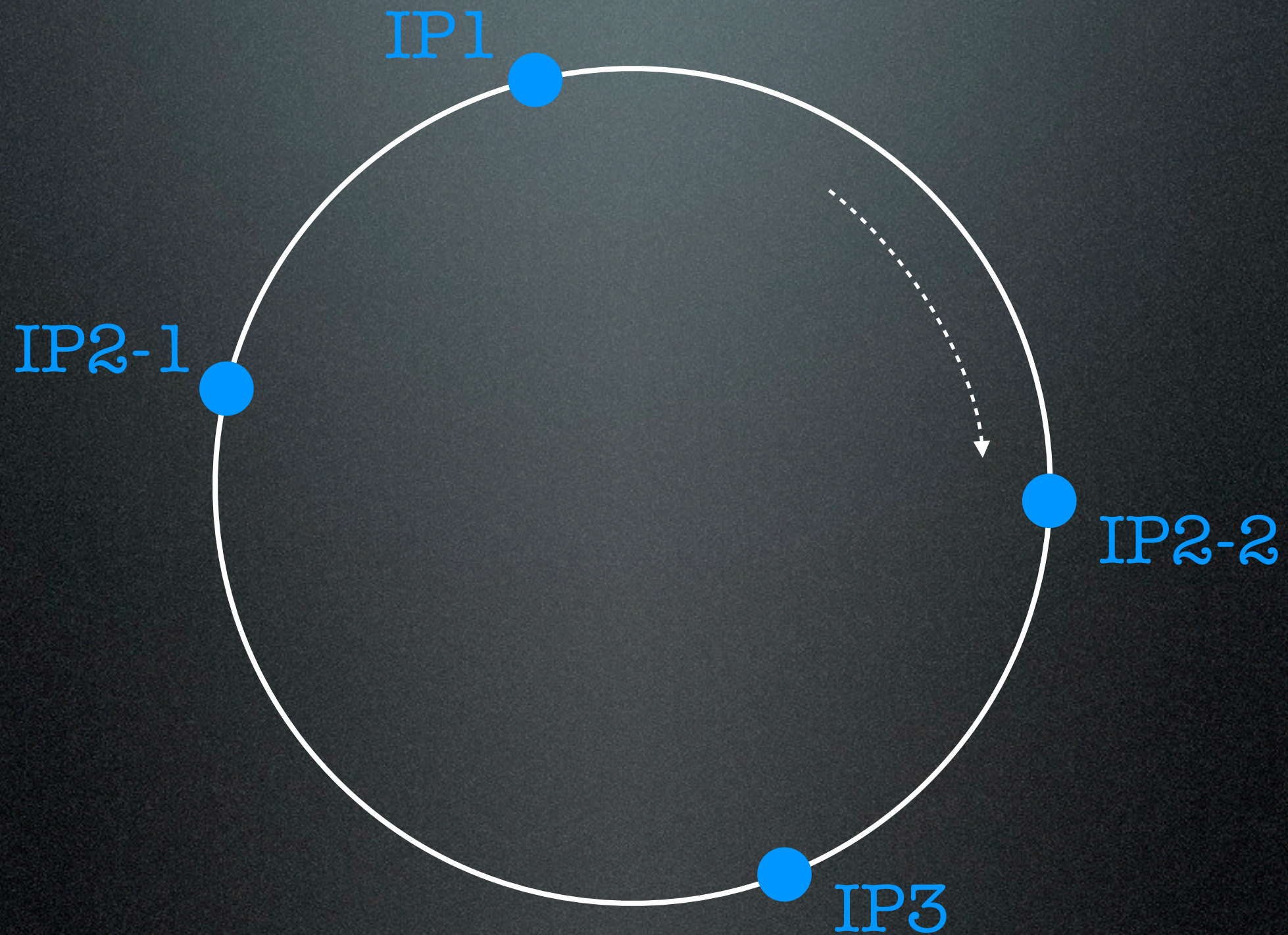
pecl/memcached

- based on libmemcached
- released in January 2009
- surface API similarity to pecl/memcache
- parity with other languages

API

- get
- set
- add
- replace
- delete
- append
- prepend
- cas
- *_by_key
- getMulti
- setMulti
- getDelayed / fetch*
- callbacks

consistent hashing



compare-and-swap (cas)

- “check and set”
- no update if object changed
- relies on CAS token

compare-and-swap (cas)

```
$m = new Memcached();  
$m->addServer('localhost', 11211);  
  
do {  
    $ips = $m->get('ip_block', null, $cas);  
  
    if ($m->getResultCode() == Memcached::RES_NOTFOUND) {  
        $ips = array($_SERVER['REMOTE_ADDR']);  
        $m->add('ip_block', $ips);  
    } else {  
        $ips[] = $_SERVER['REMOTE_ADDR'];  
        $m->cas($cas, 'ip_block', $ips);  
    }  
}  
  
} while ($m->getResultCode() != Memcached::RES_SUCCESS);
```


delayed “lazy” fetching

- issue request with `getDelayed()`
- do other work
- fetch results with `fetch()` or `fetchAll()`

binary protocol

- performance
 - every request is parsed
 - can happen thousands times a second
- extensibility
 - support more data in the protocol

callbacks

- read-through cache callback
- if key is not found, invoke callback, save value to memcache and return it

callbacks

- result callback
- invoked by `getDelayed()` for every found item
- should not call `fetch()` in this case

buffered writes

- queue up write requests
- send when a threshold is exceeded or a 'get' command is issued

key prefixing

- optional prefix prepended to all the keys automatically
- allows for namespacing, versioning, etc.

key locality

- allows mapping a set of keys to a specific server

multiple serializers

- PHP
- igbinary
- JSON (soon)

future

- UDP support
- replication
- server management (ejection, status callback)

tips & tricks

- 32-bit systems with > 4GB memory:

```
memcached -m4096 -p11211
```

```
memcached -m4096 -p11212
```

```
memcached -m4096 -p11213
```


tips & tricks

- write-through or write-back cache
- Warm up the cache on code push
- Version the keys (if necessary)

tips & tricks

- Don't think row-level DB-style caching; think complex objects
- Don't run memcached on your DB server — your DBAs might send you threatening notes
- Use multi-get — run things in parallel

delete by namespace

```
1  $ns_key = $memcache->get("foo_namespace_key");
2  // if not set, initialize it
3  if ($ns_key === false)
4      $memcache->set("foo_namespace_key",
5          rand(1, 10000));
6  // cleverly use the ns_key
7  $my_key = "foo_". $ns_key . "_12345";
8  $my_val = $memcache->get($my_key);
9  // to clear the namespace:
10 $memcache->increment("foo_namespace_key");
```


storing lists of data

- Store items under indexed keys: `comment.12`, `comment.23`, etc
- Then store the list of item IDs in another key: `comments`
- To retrieve, fetch `comments` and then multi-get the comment IDs

preventing stampeding

- embedded probabilistic timeout
- gearman unique task trick

optimization

- watch stats (eviction rate, fill, etc)
 - `getStats()`
 - telnet + “stats” commands
 - peep (heap inspector)

slabs

- Tune slab sizes to your needs:
 - **-f** chunk size growth factor (default 1.25)
 - **-n** minimum space allocated for key+value+flags (default 48)

slabs

```
slab class    1: chunk size      104 perslab 10082
slab class    2: chunk size      136 perslab  7710
slab class    3: chunk size      176 perslab  5957
slab class    4: chunk size      224 perslab  4681
...
slab class   38: chunk size 394840 perslab     2
slab class   39: chunk size 493552 perslab     2
```

Default: 38 slabs

slabs

Most objects: ~1-2KB, some larger

```
slab class    1: chunk size    1048 perslab    1000
slab class    2: chunk size    1064 perslab     985
slab class    3: chunk size    1080 perslab     970
slab class    4: chunk size    1096 perslab     956
...
slab class 198: chunk size    9224 perslab     113
slab class 199: chunk size    9320 perslab     112
```

memcached -n 1000 -f 1.01

memcached @ digg

ops

- memcached on each app server (2GB)
- the process is niced to a lower level
- separate pool for sessions
- 2 servers keep track of cluster health

key prefixes

- global key prefix for apc, memcached, etc
- each pool has additional, versioned prefix: **.sess.2**
- the key version is incremented on each release
- global prefix can invalidate all caches

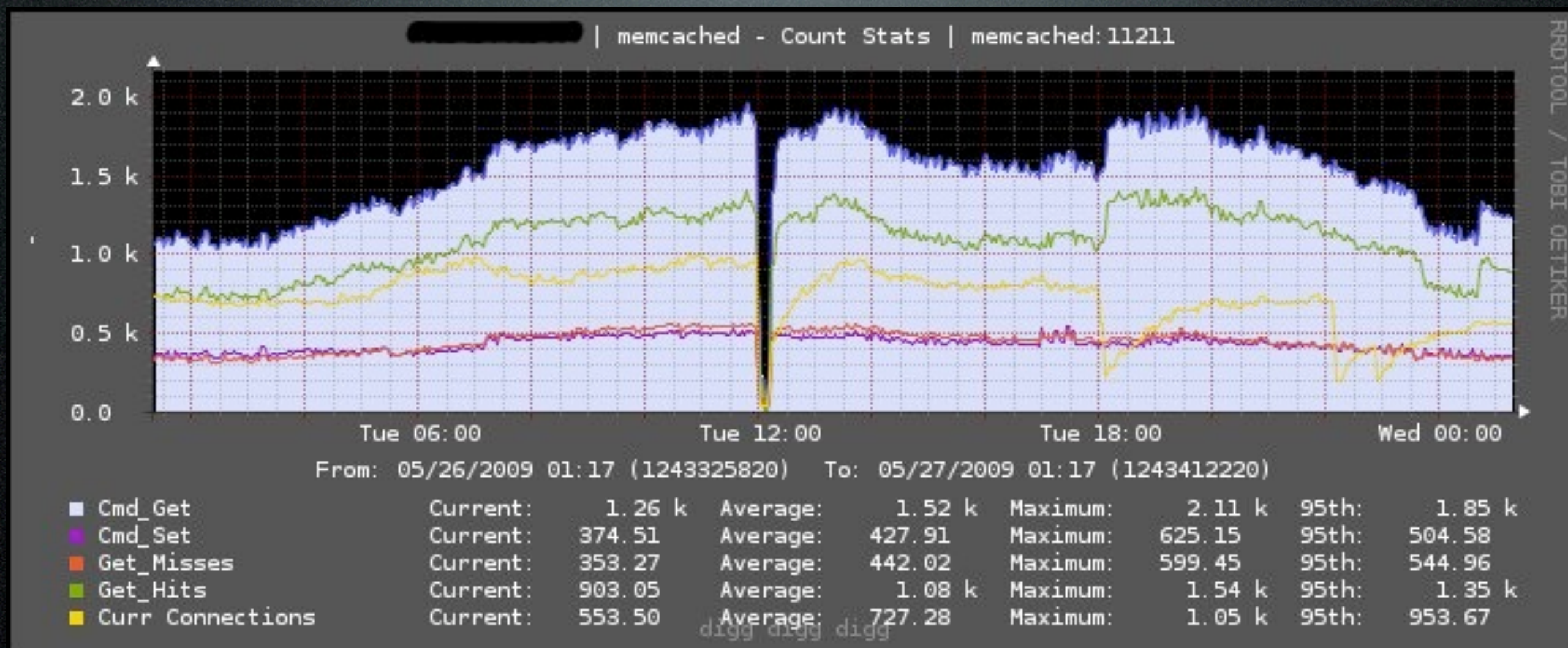
cache chain

- multi-level caching: globals, APC, memcached, etc.
- all cache access is through Cache_Chain class
- various configurations:
 - APC → memcached
 - \$GLOBALS → APC

other

- large objects ($> 1\text{MB}$)
- split on the client side
- save the partial keys in a master one

stats



alternatives

- in-memory: Tokyo Tyrant, Scalaris
- persistent: Hypertable, Cassandra, MemcacheDB
- document-oriented: CouchDB



mogile

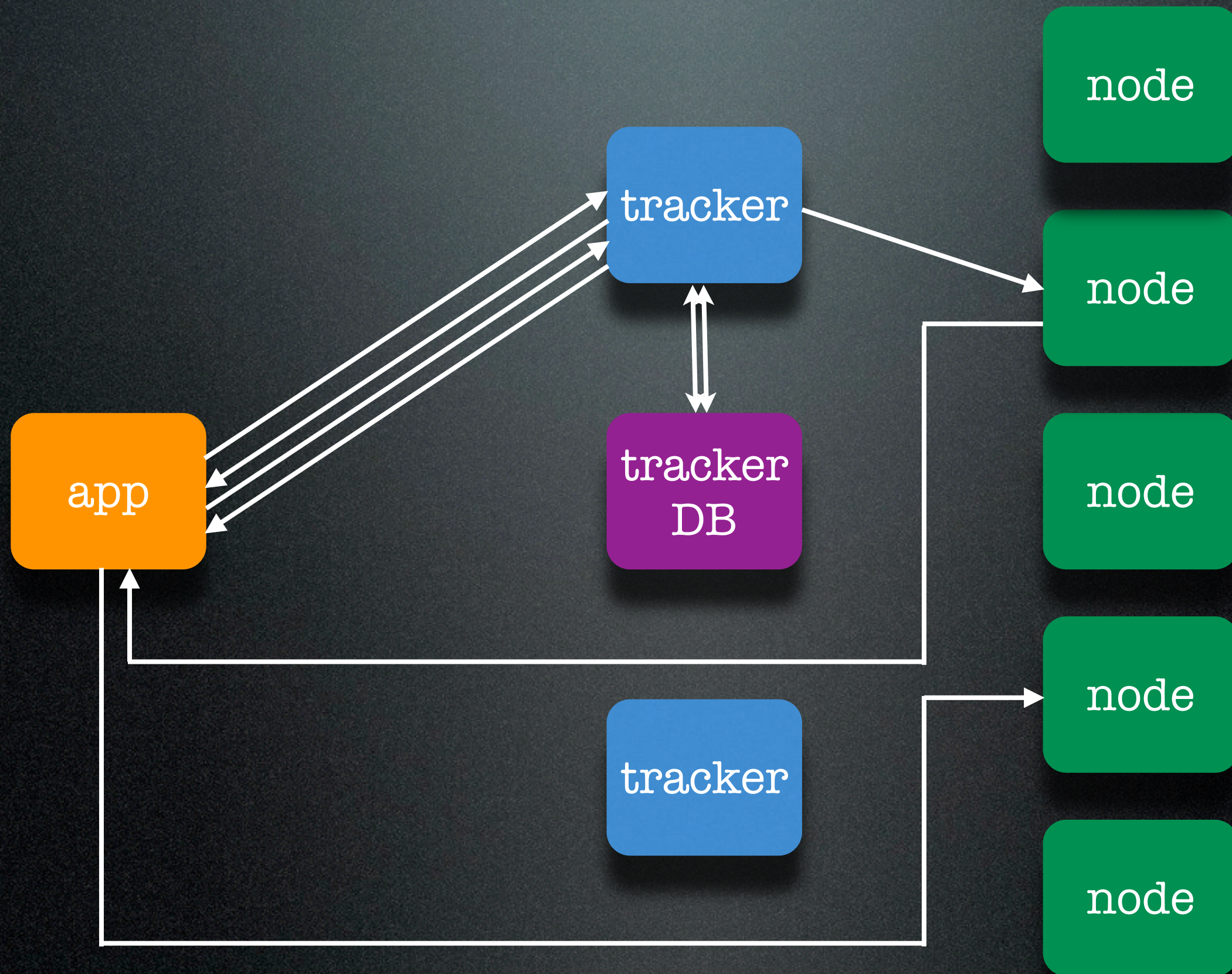
background

- created by Danga Interactive
- application-level distributed filesystem
- used at Digg, LiveJournal, etc
- a form of “cloud caching”
- scales very well

background

- automatic file replication with custom policies
- no single point of failure
- flat namespace
- local filesystem agnostic
- not meant for speed

architecture



applications

- images
- document storage
- backing store for certain caches

PHP client

- File_Mogile in PEAR
- MediaWiki one (not maintained)

Example

```
$hosts = array('172.10.1.1', '172.10.1.2');  
$m = new File_Mogile($hosts, 'profiles');  
$m->storeFile('user1234', 'image',  
              '/tmp/image1234.jpg');
```

...

```
$paths = $m->getPaths('user1234');
```


mogile @ digg

mogile @ digg

- Wrapper around File_Mogile to cache entries in memcache
- fairly standard set-up
- trackers run on storage nodes

mogile @ digg

- not huge (about 3.5 TB of data)
- files are replicated 3x
- the user profile images are cached on Netscaler (1.5 GB cache)
- mogile cluster load is light



gearman

background

- created by Danga Interactive
- anagram of “manager”
- a system for distributing work
- a form of RPC mechanism

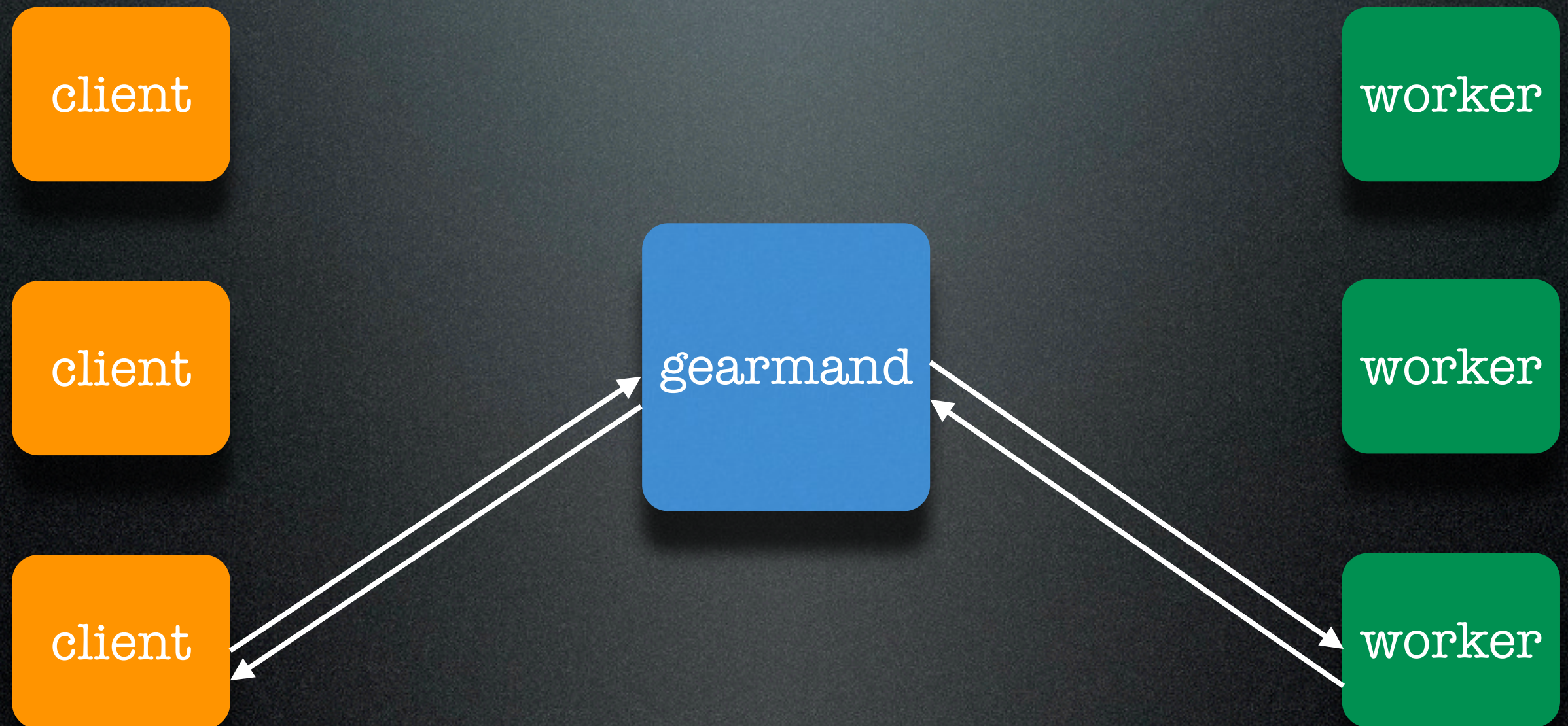
background

- parallel, asynchronous, scales well
- fire and forget, decentralized
- avoid tying up Apache processes

background

- dispatch function calls to machines that are better suited to do work
- do work in parallel
- load balance lots of function calls
- invoke functions in other languages

architecture



applications

- thumbnail generation
- asynchronous logging
- cache warm-up
- DB jobs, data migration
- sending email

servers

- Gearman-Server (Perl)
- gearmand (C)

clients

- Net_Gearman
 - simplified, pretty stable
- pecl/gearman
 - more powerful, complex, somewhat unstable (under development)

Concepts

- Job
- Worker
- Task
- Client

Net_Gearman

- Net_Gearman_Job
- Net_Gearman_Worker
- Net_Gearman_Task
- Net_Gearman_Set
- Net_Gearman_Client

Echo Job

```
class Net_Gearman_Job_Echo extends Net_Gearman_Job_Common
{
    public function run($arg)
    {
        var_export($arg);
        echo "\n";
    }
}
```

Echo.php

Reverse Job

```
class Net_Gearman_Job_Reverse extends Net_Gearman_Job_Common
{
    public function run($arg)
    {
        $result = array();
        $n = count($arg);
        $i = 0;
        while ($value = array_pop($arg)) {
            $result[] = $value;
            $i++;
            $this->status($i, $n);
        }

        return $result;
    }
}
```

Reverse.php

Worker

```
define('NET_GEARMAN_JOB_PATH', './');

require 'Net/Gearman/Worker.php';

try {
    $worker = new Net_Gearman_Worker(array('localhost:4730'));
    $worker->addAbility('Reverse');
    $worker->addAbility('Echo');
    $worker->beginWork();
} catch (Net_Gearman_Exception $e) {
    echo $e->getMessage() . "\n";
    exit;
}
```


Client

```
require_once 'Net/Gearman/Client.php';

function complete($job, $handle, $result) {
    echo "$job complete, result: ".var_export($result,
true)."\\n";
}

function status($job, $handle, $n, $d)
{
    echo "$n/$d\\n";
}
```

continued..

Client

```
$client = new Net_Gearman_Client(array('lager:4730'));  
  
$task = new Net_Gearman_Task('Reverse', range(1,5));  
$task->attachCallback("complete",Net_Gearman_Task::TASK_COMPLETE);  
$task->attachCallback("status",Net_Gearman_Task::TASK_STATUS);  
  
continued..
```


Client

```
$set = new Net_Gearman_Set();  
$set->addTask($task);  
  
$client->runSet($set);  
  
$client->Echo('Mmm... beer');
```


pecl/gearman

- More complex API
- Jobs aren't separated into files

Worker

```
$gmworker= new gearman_worker();
$gmworker->add_server();
$gmworker->add_function("reverse", "reverse_fn");

while (1)
{
    $ret= $gmworker->work();
    if ($ret != GEARMAN_SUCCESS)
        break;
}

function reverse_fn($job)
{
    $workload= $job->workload();
    echo "Received job: " . $job->handle() . "\n";
    echo "Workload: $workload\n";
    $result= strrev($workload);
    echo "Result: $result\n";
    return $result;
}
```


Client

```
$gmclient= new gearman_client();  
$gmclient->add_server('lager');  
echo "Sending job\n";  
  
list($ret, $result) = $gmclient->do("reverse", "Hello!");  
  
if ($ret == GEARMAN_SUCCESS)  
    echo "Success: $result\n";
```


gearman @ digg

gearman @ digg

- 400,000 jobs a day
- Jobs: crawling, DB job, FB sync, memcache manipulation, Twitter post, IDDB migration, etc.
- Each application server has its own Gearman daemon + workers

tips and tricks

- you can daemonize the workers easily with daemon or supervisord
- run workers in different groups, don't block on job A waiting on job B
- Make workers exit after N jobs to free up memory (supervisord will restart them)



Thrift

background

- NOT developed by Danga (Facebook)
- cross-language services
- RPC-based

background

- interface description language
- bindings: C++, C#, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, Smalltalk
- data types: base, structs, constants, services, exceptions

IDL

Demo

Thank You

<http://gravitonic.com/talks>