

A stylized gear icon with a central circle and four smaller circles around it, all in white.

Gearman

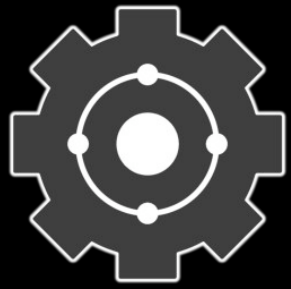
OSCON 2009

Eric Day – Sun Microsystems

<http://www.oddmments.org/>

Brian Aker – Sun Microsystems

<http://krow.net/>

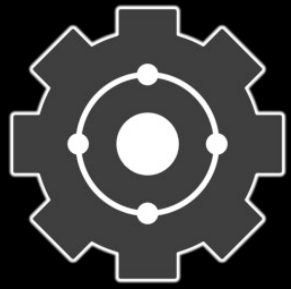


Tutorial Software

- Gearman C Server and Library
 - Packages available (Ubuntu, Debian/sid)
 - <https://launchpad.net/gearmand>
 - `./configure; make; make install`
- Gearman PHP Extension
 - <http://pecl.php.net/package/gearman>
 - Follow README
- PHP command line interface
 - `php5-cli` package (Ubuntu, Debian)

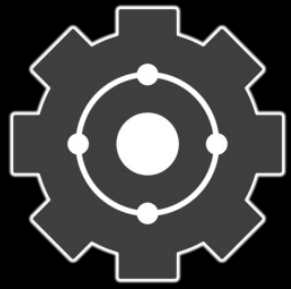
!

!



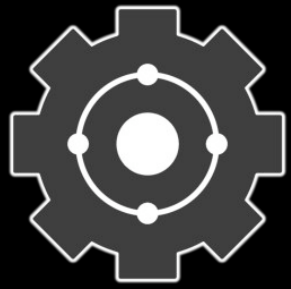
Gearman Overview

- History
- Basics
- Example
- API
 - Client
 - Worker
- Job Server



Gearman Overview

- Applications
 - Map/Reduce
 - Log Analysis
 - Asynchronous Queues
 - Narada
- Roadmap

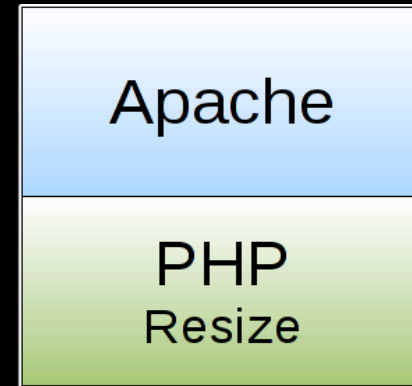
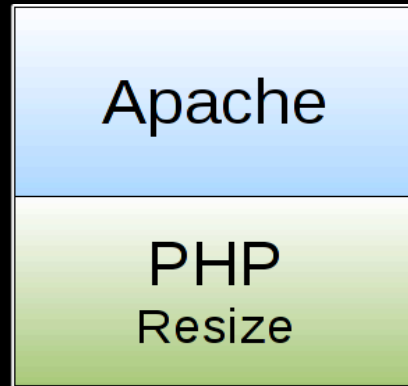
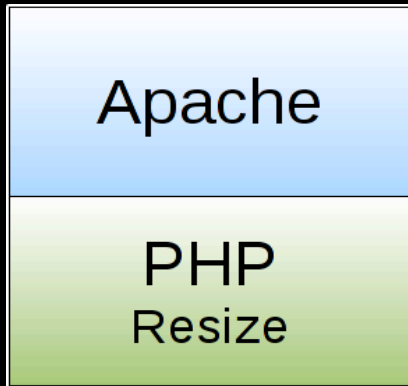


Kittens!

(LiveJournal.com Image Processing)

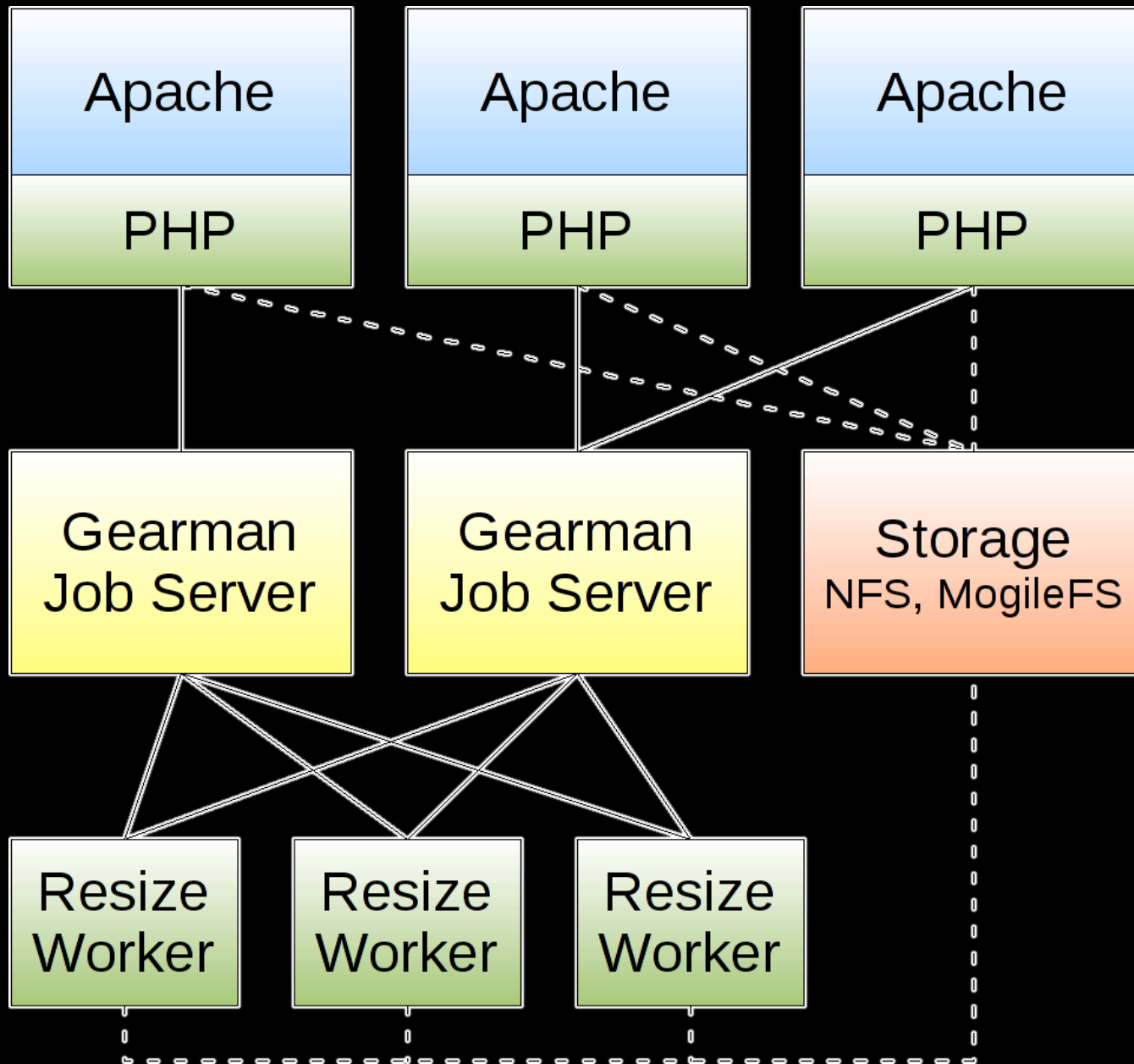
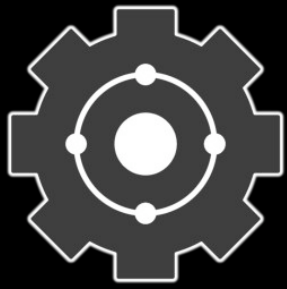
!

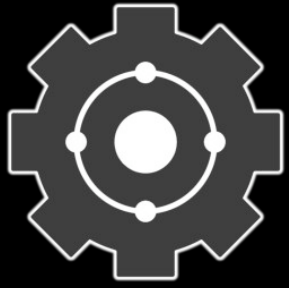
!



!

!





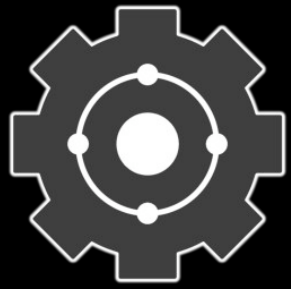
“The way I like to think of Gearman is as a massively distributed, massively fault tolerant fork mechanism.”

- Joe Stump, Digg



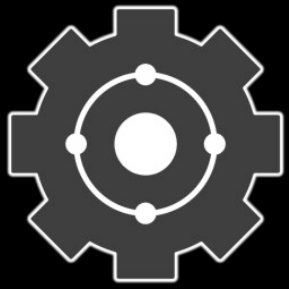
History

- Danga – Brad Fitzpatrick & company
 - Related to memcached, MogileFS, ...
- Anagram for “manager”
 - Gearman, like managers, assign the tasks but do none of the real work themselves
- Digg: 45+ servers, 400K jobs/day
- Yahoo: 60+ servers, 6M jobs/day
- LiveJournal, SixApart, DealNews, ...



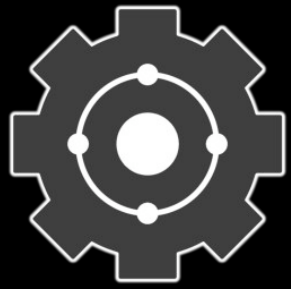
Recent Development

- Rewrite in C
- New language APIs
 - PHP ext, Perl XS, Drizzle, MySQL, PostgreSQL
- Command line tool
- Protocol additions
- Multi-threaded (50k jobs/second)
- Persistent queues
- Pluggable protocol



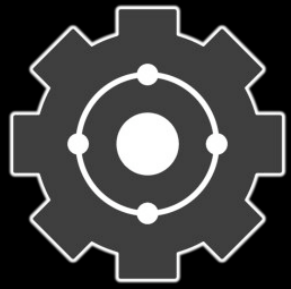
Features

- Open Source (mostly BSD)
- Multi-language
 - Mix clients and workers from different APIs
- Flexible Application Design
 - Not restricted to a single distributed model
- Simple & Fast
- Embeddable
 - Small & lightweight for applications of all sizes
- No Single Point of Failure

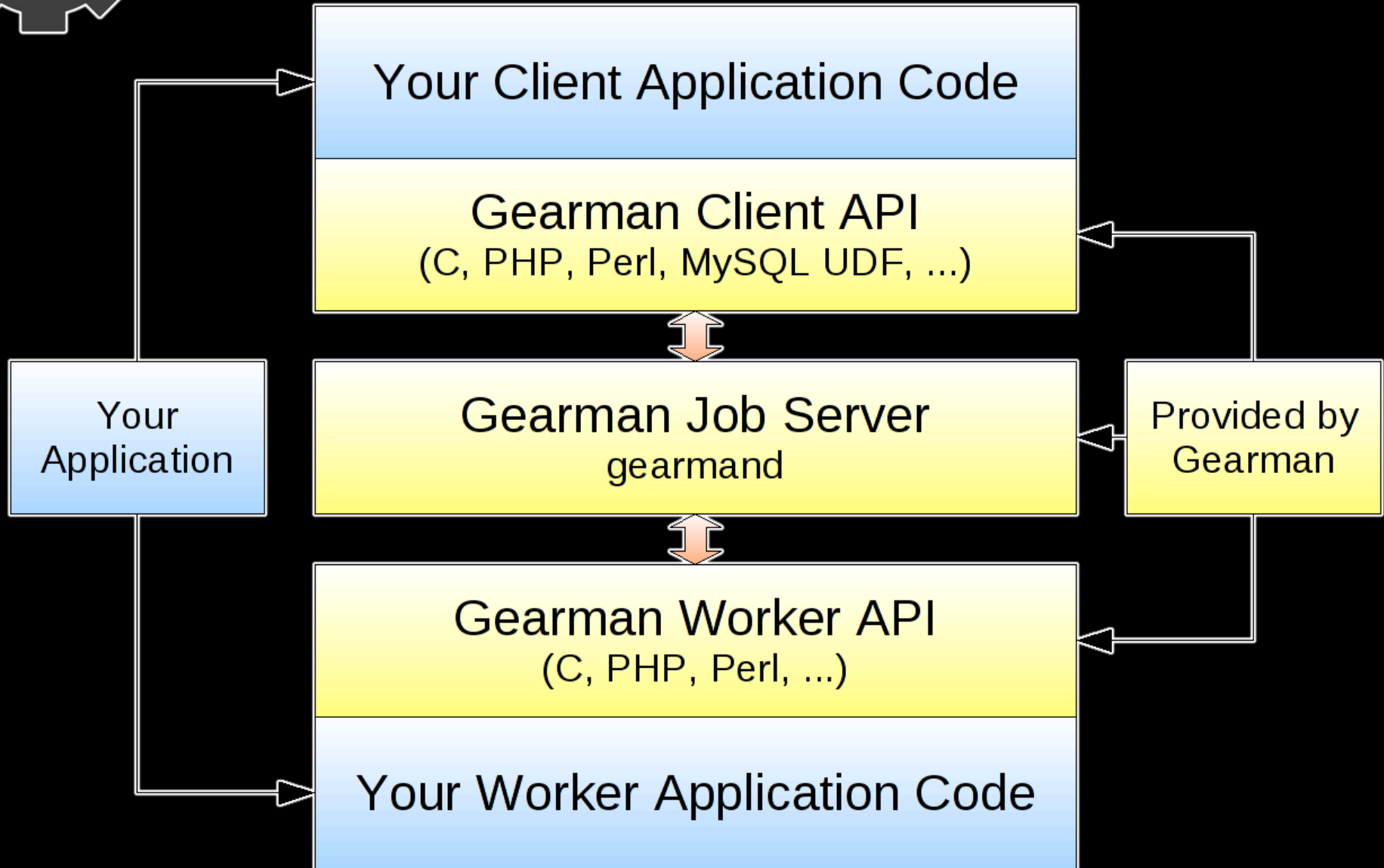


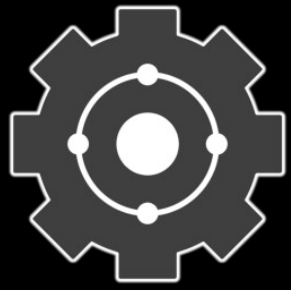
Basics

- Gearman provides a distributed application framework
- Uses TCP port 4730 (was port 7003)
- **Client** – Create jobs to be run and send them to a job server
- **Worker** – Register with a job server and grab jobs to run
- **Job Server** – Coordinate the assignment from clients to workers, handle restarts

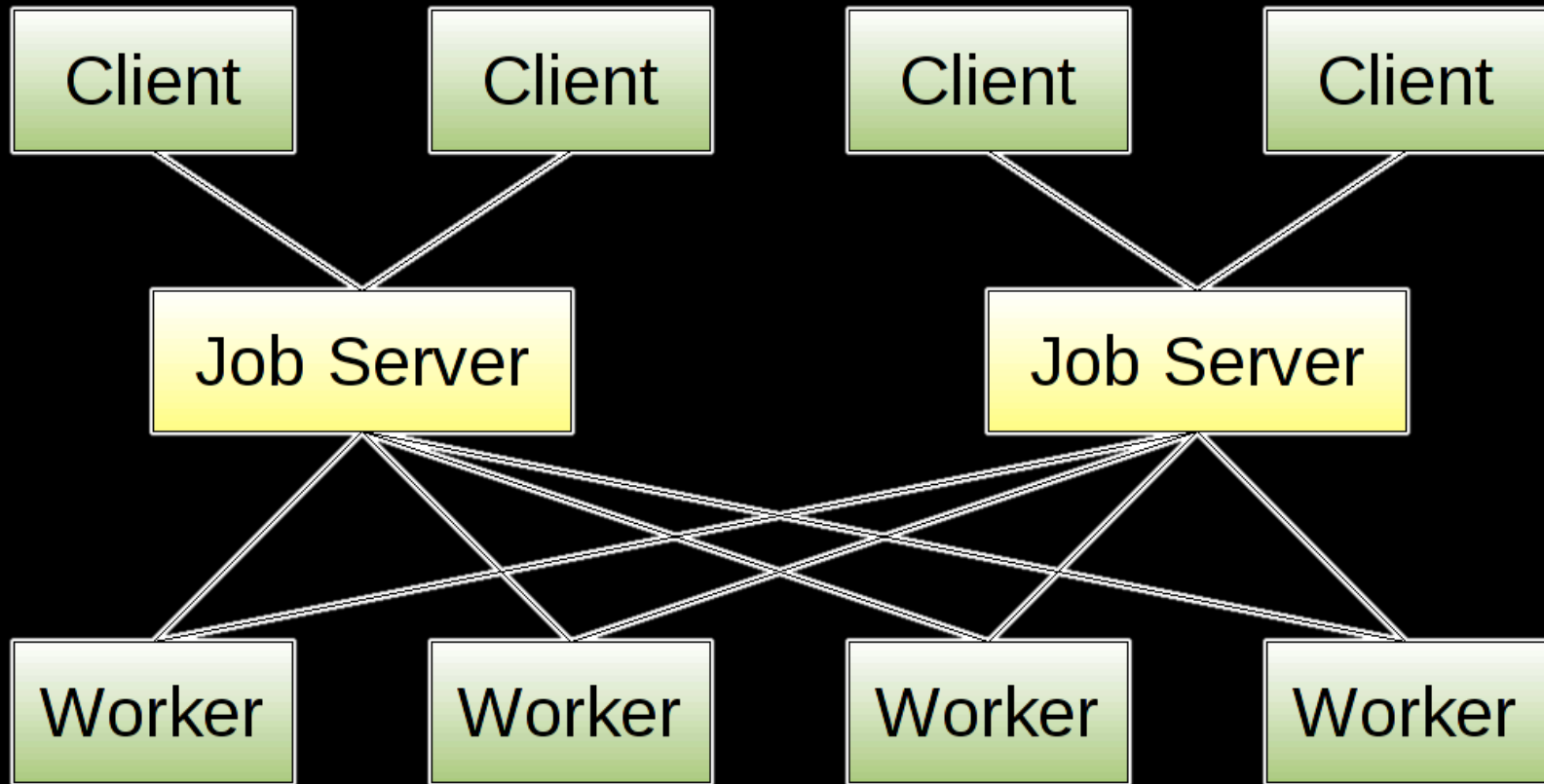


Gearman Stack





No Single Point of Failure





Hello World

```
$client= new GearmanClient();  
$client->addServer();  
print $client->do("reverse", "Hello World!");
```

```
$worker= new GearmanWorker();  
$worker->addServer();  
$worker->addFunction("reverse", "my_reverse_function");  
while ($worker->work());  
  
function my_reverse_function($job)  
{  
    return strrev($job->workload());  
}
```

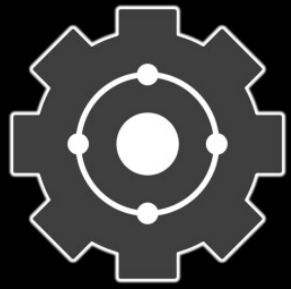


Hello World

```
shell$ gearmand -d
```

```
shell$ php worker.php &  
[1] 17510
```

```
shell$ php client.php  
!dlrow olleH
```



How Is This Useful?

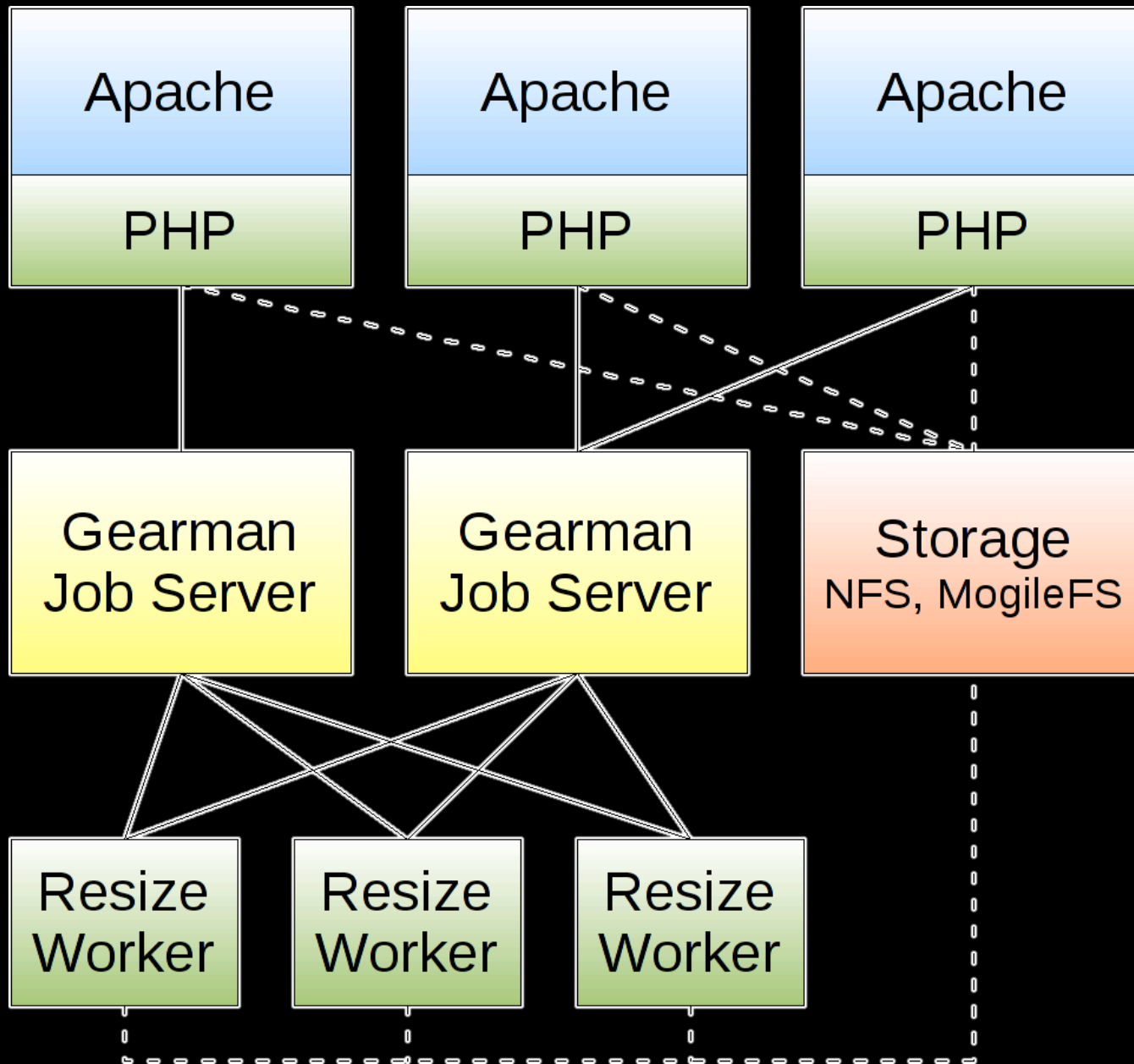
- Provides a distributed nervous system
- Natural load balancing
 - Workers are notified and ask for work, not forced
- Multi-language integration
- Distribute processing
 - Possibly closer to data
- Synchronous and Asynchronous queues



Back to the Kittens

!

!



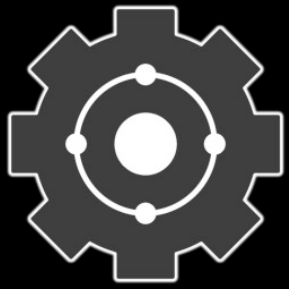


Image Resize Worker

```
$worker= new GearmanWorker();
$worker->addServer();
$worker->addFunction("resize", "my_resize_function");
while ($worker->work());

function my_resize_function($job)
{
    $thumb = new Imagick();
    $thumb->readImageBlob($job->workload());
    $thumb->scaleImage(200, 150);
    return $thumb->getImageBlob();
}
```

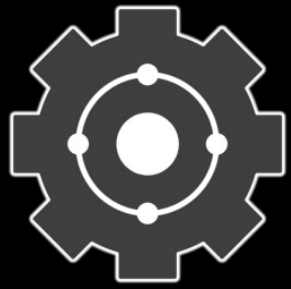


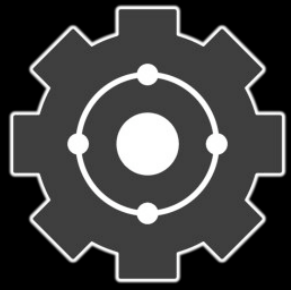
Image Resize Worker

```
shell$ gearmand -d
```

```
shell$ php resize.php &  
[1] 17524
```

```
shell$ gearman -f resize < large.jpg > thumb.jpg
```

```
shell$ ls -sh large.jpg thumb.jpg  
3.0M large.jpg    32K thumb.jpg
```



APIs

!

!



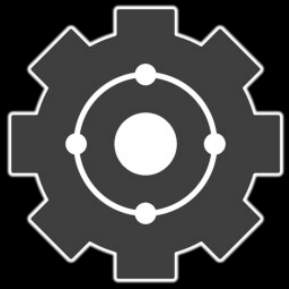
APIs

- Focus on OO PHP extension API
 - PHP extension also has procedural
 - `$client->do(...)` == `gearman_client_do($client, ...)`
- Other implementations similar
 - Same concepts
 - Similar function names
- C (libgearman), Native Perl, Perl::XS, Native PHP, Java, JMS, Native Python, Drizzle, MySQL, PostgreSQL, ...
- New C-based Python, Ruby, and others soon!



Command Line Tool

- gearman
 - Included in C server and library package
 - Command line and shell script interface
- Client mode
 - `ls | gearman -f function`
 - `gearman -f function < file`
 - `gearman -f function "some data"`
- Worker mode
 - `gearman -w -f function -- wc -l`
 - `gearman -w -f function !./script.sh`



Command Line Tool

```
shell$ gearmand -d
```

```
shell$ gearman -w -f test -- grep lib &  
[1] 7622
```

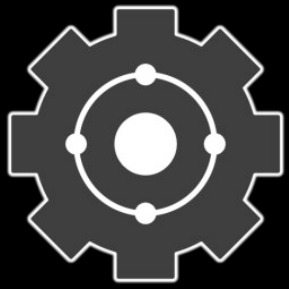
```
shell$ ls / | gearman -f test  
lib  
lib32  
lib64
```



Client API

!

!



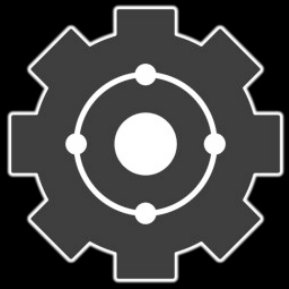
Client API

- Create client object
 - Can have multiple objects in an application
 - Server lists, options, ...

```
$client= new GearmanClient();  
$client->addServer();
```

```
$client= new GearmanClient();  
$client->addServer("10.0.0.1");  
$client->addServer("10.0.0.2", 7003);
```

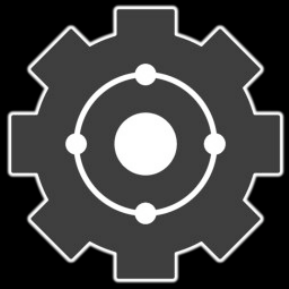
```
$client= new GearmanClient();  
$client->addServers("10.0.0.1,10.0.0.2:7003");
```



Client API

- One job at a time
 - **do** functions and methods

```
$client= new GearmanClient();  
$client->addServer();  
  
print $client->do("reverse", "Hello World!") . "\n";  
print $client->doHigh("reverse", "Hello High!") . "\n";  
print $client->doLow("reverse", "Hello Low!") . "\n";
```



Client API

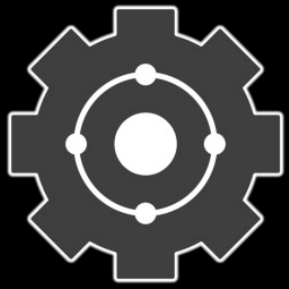
- Multiple jobs at a time
 - **task** functions and methods

```
$client= new GearmanClient();
$client->addServer();

$client->addTask("reverse", "Hello World!");
$client->addTaskHigh("reverse", "Hello High!");
$client->addTaskLow("reverse", "Hello Low!");

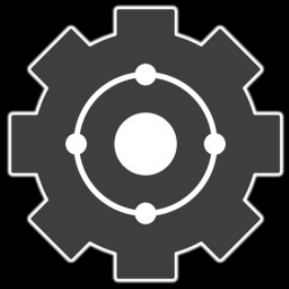
$client->setCompleteCallback("complete");
$client->runTasks();

function complete($task)
{
    print $task->data() . "\n";
}
```



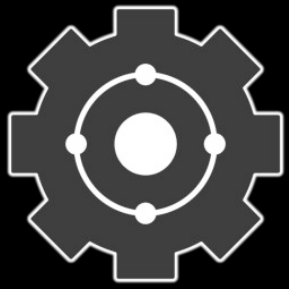
Tasks vs Jobs

- A job is a task
- A task is not always a job
- Check status of background job
 - Not a job, but is a task
 - Send job handle, receive status
 - `$client->jobStatus()`
 - `$client->addTaskStatus()`
- Clients deal with tasks
- Workers deal with jobs



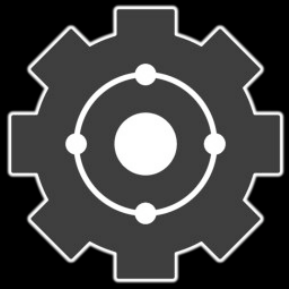
Job Attributes

- Foreground (Synchronous)
 - `$client->do()`
- Background (Asynchronous)
 - `$client->doBackground()`
- High and Low priority
 - `$client->doHigh()`
 - `$client->doLow()`
- Mix and match
 - `$client->doHighBackground(), ...`



Job Attributes

- Application data/state for tasks
- Optional Unique Key
 - UUID generated if none given
 - Coalesce jobs in job server
 - Only one worker per key at any time
 - All clients get the same response
 - Resizing an image
 - Multiple clients submit job at the same time
 - Worker runs once, all clients get thumbnail



Concurrent Task API

- Queue jobs to run
- Register callbacks for interested events
- Run tasks
 - `$client->runTasks();`
 - Returns when all tasks are complete
- No guarantee on response order



Concurrent Task API

- Available Callbacks
 - `setWorkloadCallback()` - Streaming job workload
 - `setCreatedCallback()` - Job created in server
 - `setDataCallback()` - Partial data response
 - `setWarningCallback()` - Partial warning response
 - `setStatusCallback()` - Status (X/Y done)
 - `setCompleteCallback()` - Job complete
 - `setExceptionCallback()` - Exception caught (Perl)
 - `setFailCallback()` - Job failed
 - `clearCallbacks()` - Clear all set callbacks



Concurrent Task API

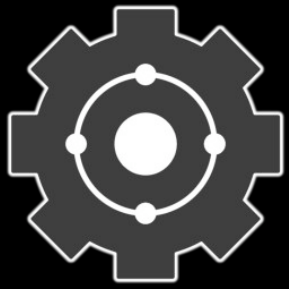
```
$client= new GearmanClient();
$client->addServer();

$client->addTask("reverse", "Hello World 1!");
$client->addTask("reverse", "Hello World 2!");
$client->addTask("reverse", "Hello World 3!");

$client->setCreatedCallback("created");
$client->setCompleteCallback("complete");
$client->runTasks();

function created($task)
{
    print "Created: " . $task->jobHandle() . "\n";
}

function complete($task)
{
    print "Complete: " . $task->data() . "\n";
}
```



Concurrent Task API

- Start gearmand
- Start multiple reverse workers

```
shell$ php task.php
Created: H:lap:108
Created: H:lap:109
Created: H:lap:110
Complete: !2 dlrow olleH
Complete: !3 dlrow olleH
Complete: !1 dlrow olleH
```



Task Application Data

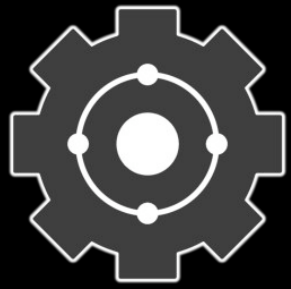
- Pass application data and state variables

```
$client= new GearmanClient();
$client->addServer();

$count= 0;
$client->addTask("reverse", "Hello World 1!", &$count);
$client->addTask("reverse", "Hello World 2!", &$count);
$client->addTask("reverse", "Hello World 3!", &$count);

$client->setCompleteCallback("complete");
$client->runTasks();
print "Completed $count tasks\n";

function complete($task, $count)
{
    print "$count: " . $task->data() . "\n";
    $count++;
}
```



Task Application Data

- Pass variables by reference
- \$count incremented for each job

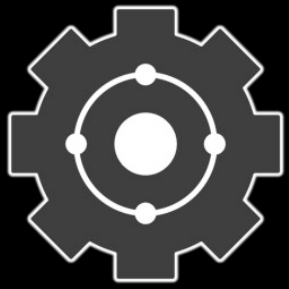
```
shell$ php app_data.php
0: !2 dlrow olleH
1: !3 dlrow olleH
2: !1 dlrow olleH
Completed 3 tasks
```



Worker API

!

!



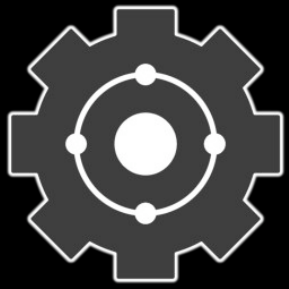
Worker API

- Create worker object
 - Can have multiple objects in an application
 - Server lists, options, ...

```
$worker= new GearmanWorker();  
$worker->addServer();
```

```
$worker= new GearmanWorker();  
$worker->addServer("10.0.0.1");  
$worker->addServer("10.0.0.2", 7003);
```

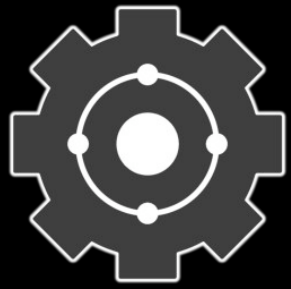
```
$worker= new GearmanWorker();  
$worker->addServers("10.0.0.1,10.0.0.2:7003");
```



Worker API

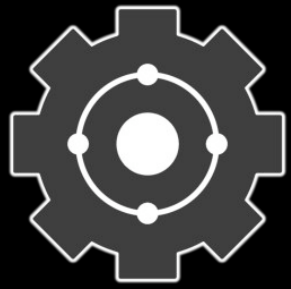
- Register function abilities
 - Function name
 - Local callback function
- Wait for jobs on those functions
- Will only run one job at a time

```
$worker= new GearmanWorker();  
$worker->addServer();  
$worker->addFunction("function1", "first_callback");  
$worker->addFunction("function2", "common_callback");  
$worker->addFunction("function3", "common_callback");  
while ($worker->work());
```



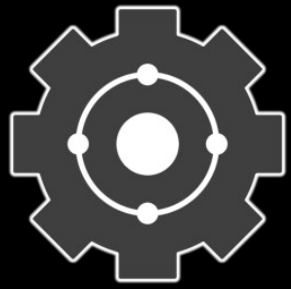
Worker Callback Function

- Have access to \$job object
- Get job information
 - \$job->workload() - Get job workload
 - \$job->functionName() - Get function name
 - \$job->unique() - Get unique key
 - \$job->handle() - Get job handle



Worker Callback Function

- Send intermediate job responses
 - `$job->data(...)` - Send partial result
 - `$job->warning(...)` - Send warning
 - `$job->status(X, Y)` – Send X/Y status
- Return value sent back to caller (if any)
- Don't bother if it's a background job
 - Except `$job->status(X, Y)`

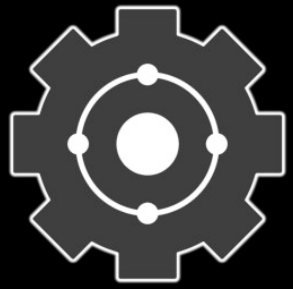


Worker Callback Function

- Can also return failure

```
$worker= new GearmanWorker();
$worker->addServer();
$worker->addFunction("reverse", "always_fail");
while ($worker->work());

function always_fail($job)
{
    $job->setReturn(GEARMAN_WORK_FAIL);
}
```



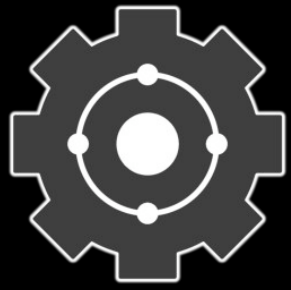
Worker Application Data

- Pass data or state variable into callback

```
$worker= new GearmanWorker();
$worker->addServer();
$count= 0;
$worker->addFunction("reverse", "reverse_cb", &$count);
while ($worker->work());

function reverse_cb($job, $count)
{
    $count++;
    return "$count: " . strrev($job->workload());
}
```

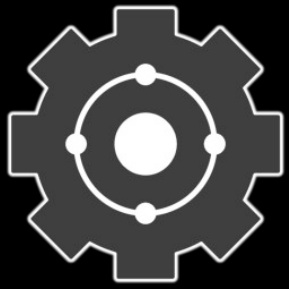
```
shell$ gearman -f reverse hello
1: olleh
shell$ gearman -f reverse hello
2: olleh
...
```



Job Server

!

!



Job Server

- Listens on port 4730
- Clients and workers connect
- Handle job assignment
- Restart jobs on worker failure
- Advanced features
 - Pluggable persistent queue
 - Pluggable protocols



Job Server

- gearmand --help
- Common options

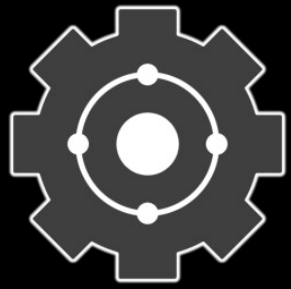
```
-d, --daemon
-h, --help
-l, --log-file=FILE
-p, --port=PORT
-P, --pid-file=FILE
-t, --threads=THREADS
-u, --user=USER
-v, --verbose
```



Verbose Option

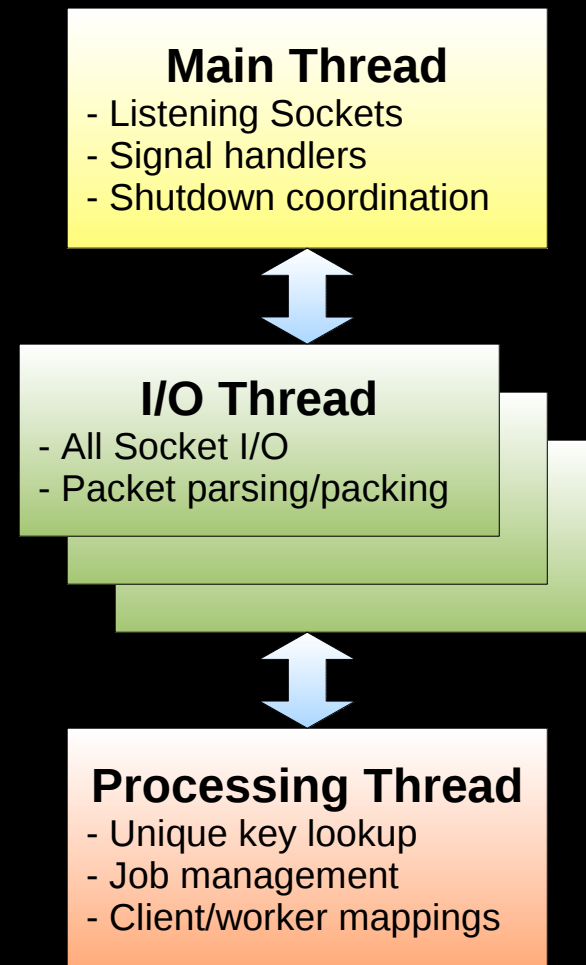
- Can specify multiple -v
- Very useful for debugging

```
shell$ gearmand -vvvv
  INFO Starting up
DEBUG Initializing libevent for main thread
DEBUG Method for libevent: epoll
DEBUG Trying to listen on :::4730
  INFO Listening on :::4730 (6)
DEBUG Trying to listen on 0.0.0.0:4730
  INFO Creating wakeup pipe
DEBUG Creating 0 threads
  INFO Creating IO thread wakeup pipe
  INFO Adding event for listening socket (6)
  INFO Adding event for wakeup pipe
  INFO Entering main event loop
...
!
!
```



Threading Model

- Can run single threaded
- The -t option sets threads
 - I/O thread count
 - Default is -t 0
 - One thread/core
- Processing thread
 - Hashes and other state
 - No systems calls
 - Non-blocking

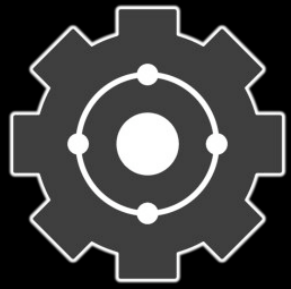




Persistent Queues

- Only for background jobs
- Specify `-q <queue>` option
- `libdrizzle` module for Drizzle and MySQL

```
shell$ gearmand -vvv -q libdrizzle
INFO Initializing libdrizzle module
INFO libdrizzle module using table 'test.queue'
...
INFO libdrizzle replay start
...
DEBUG libdrizzle add: 3ec068d9-293c-4af8-943f-d265138e67f8
DEBUG libdrizzle flush
...
DEBUG libdrizzle done: 3ec068d9-293c-4af8-943f-d265138e67f8
```



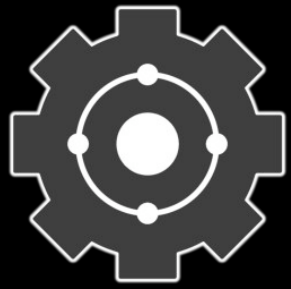
Persistent Queues

- libdrizzle queue is as robust as your DB
- Can create your own table

```
libdrizzle Options:
```

```
--libdrizzle-host=HOST  
--libdrizzle-port=PORT  
--libdrizzle-uds=UDS  
--libdrizzle-user=USER  
--libdrizzle-password=PASSWORD  
--libdrizzle-db=DB  
--libdrizzle-table=TABLE  
--libdrizzle-mysql
```

```
shell$ gearmand -q libdrizzle ...
```



Persistent Queues

- libmemcached
- PostgreSQL
- sqlite3
- Flat file
- Easy to add your own!



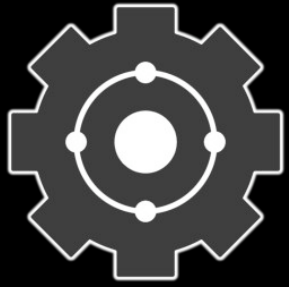
Pluggable Protocol

- Handle packet parsing and packing
- Optionally handle raw socket I/O
- HTTP protocol
- Others coming soon
 - memcached
 - XMPP
 - ?



HTTP Protocol

- GET and POST requests
 - Send workload with POST, Content-Length
- Function name from URL
- Optional headers
 - X-Gearman-Unique: <unique key>
 - X-Gearman-Background: true
 - X-Gearman-Priority: <high|low>
- Other headers ignored for now



HTTP Protocol

```
shell$ gearmand -r http &
```

```
<start reverse worker>
```

```
shell$ nc localhost 8080
```

```
POST /reverse HTTP/1.1
```

```
Content-Length: 12
```

```
Hello World!
```

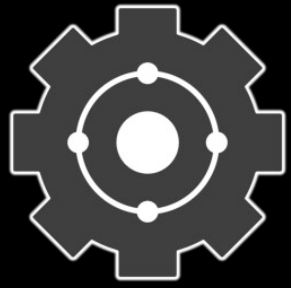
```
HTTP/1.0 200 OK
```

```
X-Gearman-Job-Handle: H:lap:1
```

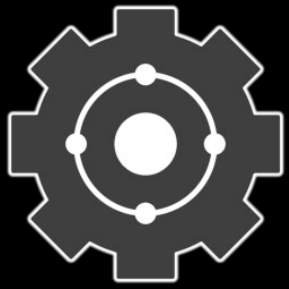
```
Content-Length: 12
```

```
Server: Gearman/0.9
```

```
!dlrow olleH
```

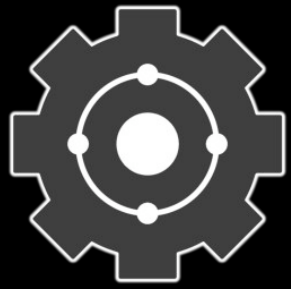


Applications

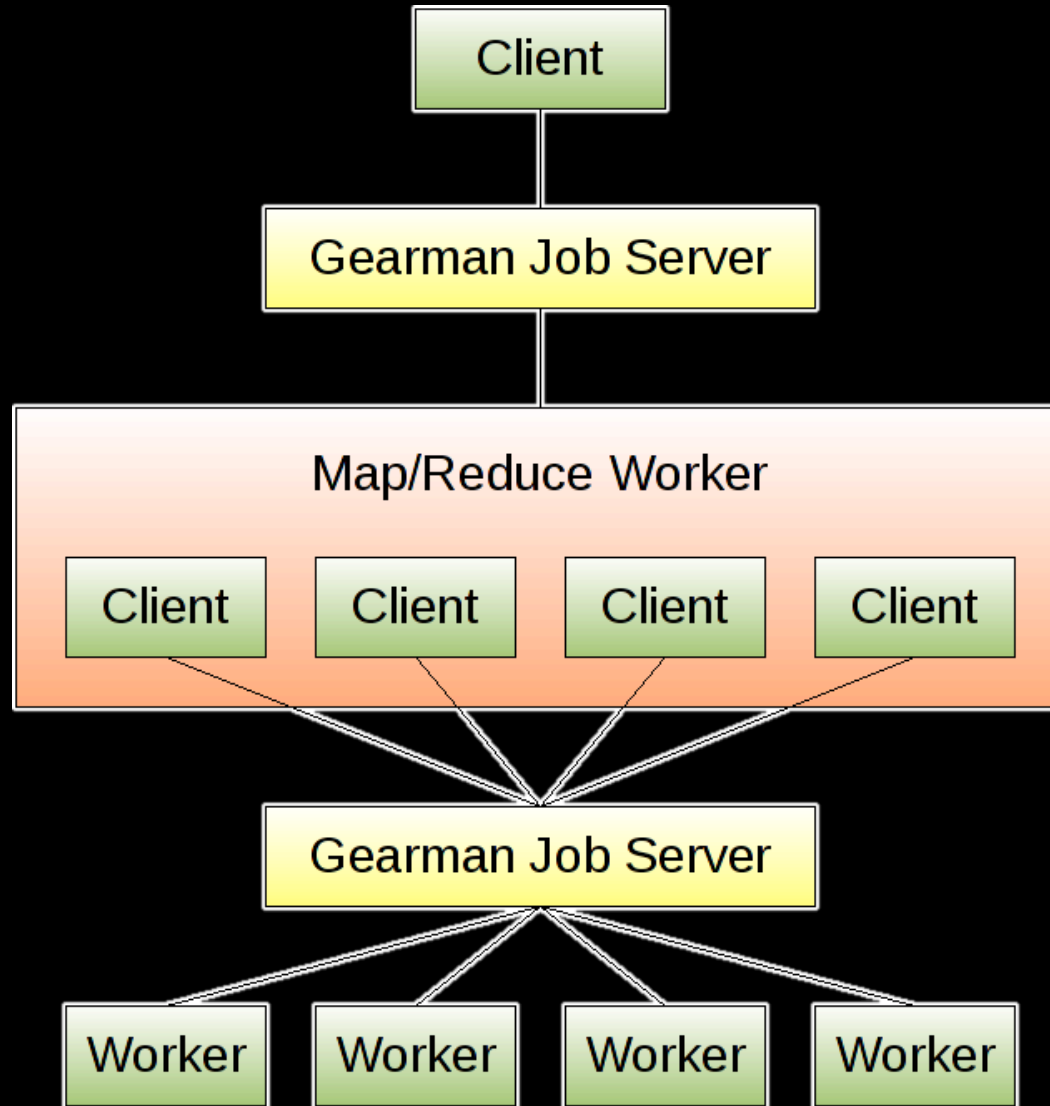


Prototype!

- Command line tool is your friend
- Or use simple scripting language
- Rewrite if needed
 - Faster
 - Security
 - More features
 - More robust



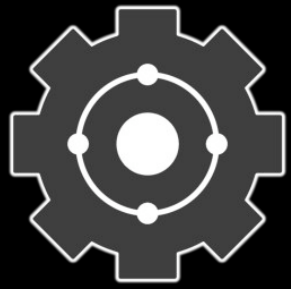
Map/Reduce





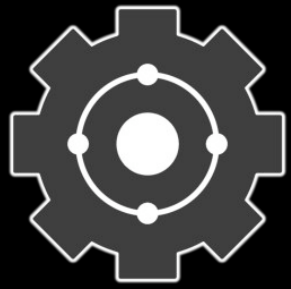
Map/Reduce

- Can be multi-tier
- Fan-out depends on application
- Don't need to follow strict MR model
 - Not Google API (paper) or Hadoop
 - Can be ad-hoc
- Use any Gearman API (mix/match)



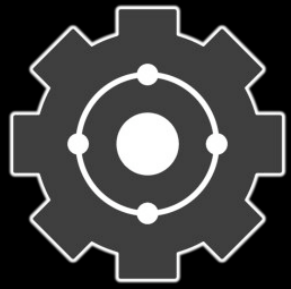
Log Processing

- Bring Map/Reduce to Apache logs
- Get log storage off Apache nodes
- Push processing to log storage nodes
- Combine data in some meaningful way
 - Summary
 - Distributed merge-sort algorithms

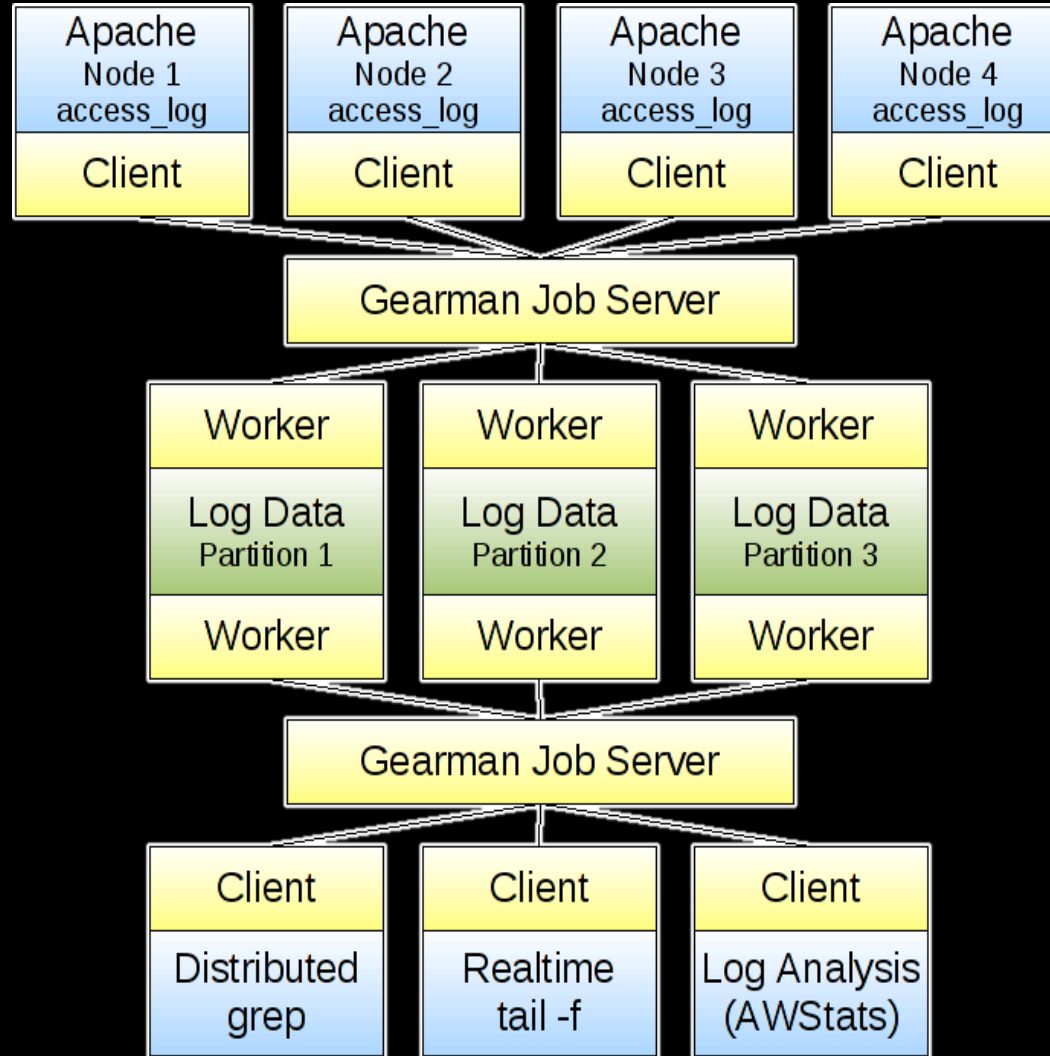


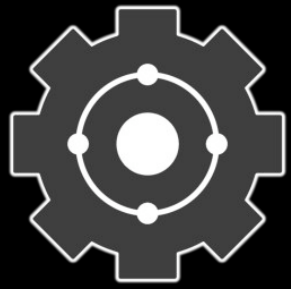
Log Processing

- Collection
 - `tail -f access_log | gearman -n -f logger`
 - `CustomLog "| gearman -n -f logger" common`
 - Write a Gearman Apache logging module
- Processing
 - Distributed/parallel grep
 - Log Analysis (AWStats, Webalizer, ...)
 - Custom data mining & click analysis



Log Processing

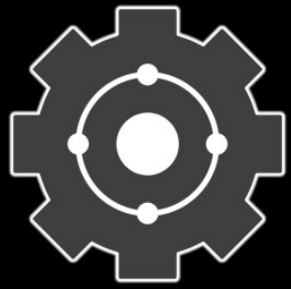




Log Processing

- Simple shell version
- Provide remote storage and distributed grep
- Send logs as mentioned before
 - `tail -f access_log | gearman -h host -f logger`
- On each log collection machine
 - `gearman -w -h host -f logger > log_file`
 - `gearman -w -h host -f logger1 ./dgrep.sh`

```
#!/bin/sh
read pattern
grep $pattern log_file
```



Log Processing

- Query logging machines
 - gearman -h host -f logger1 -f logger2 ... pattern

```
shell$ gearman -h host -f logger1 -f logger2 -f logger3 \  
news_archive | cut -f 4,5,7 -d' '
```

```
[06/Jul/2009:18:22:57 -0700] /bzt/narada/eday-dev/php/in  
dex.php?q=&u=http%3A%2F%2Flocalhost%2Fsrc%2Fnwveg%2Fnews  
_archive.php&s=Submit
```

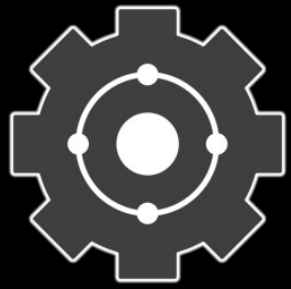
```
[06/Jul/2009:18:23:03 -0700] /src/nwveg/news_archive.php
```

```
[06/Jul/2009:18:23:54 -0700] /src/nwveg/news_archive.php
```

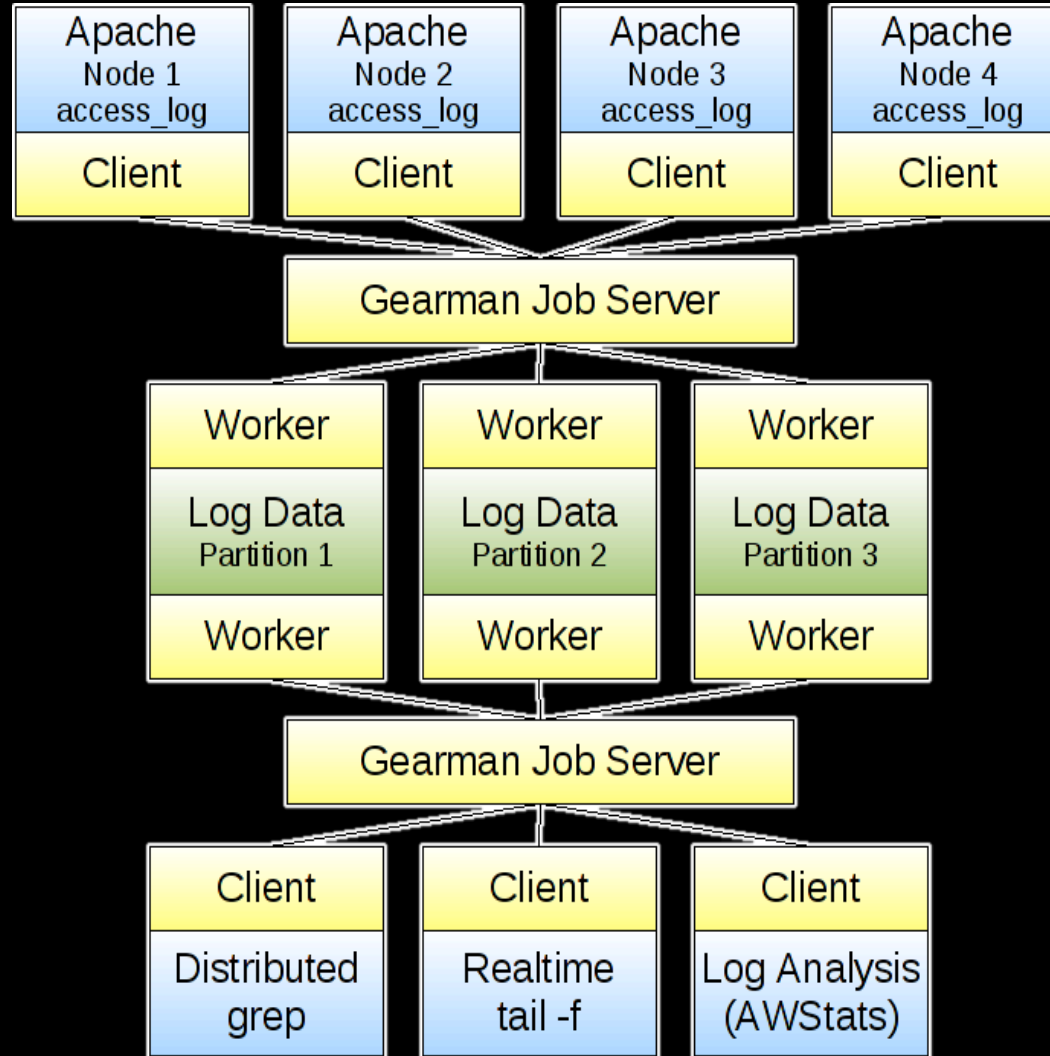
```
[06/Jul/2009:18:24:12 -0700] /src/nwveg/news_archive.php
```

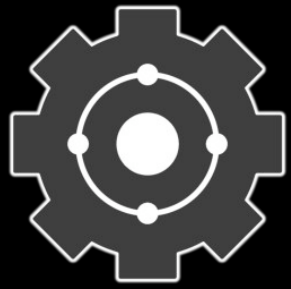
```
[06/Jul/2009:18:31:37 -0700] /bzt/narada/eday-dev/php/in  
dex.php?q=news&s=Search&u=
```

```
...
```



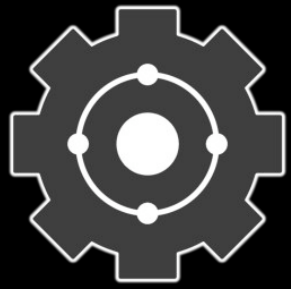
Log Processing





Asynchronous Queues

- Background Tasks
- They help you scale
- Distributed data storage
 - Eventually consistent data models
 - Choose “AP” in “CAP”
 - Consistency
 - Availability
 - Partitions (tolerance to network partitions)
 - Make eventual consistency work
 - Conflict resolution if needed



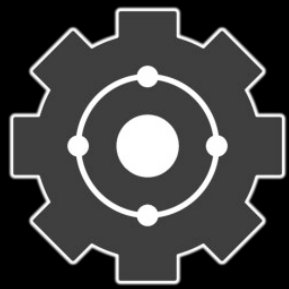
Asynchronous Queues

- Not everything needs immediate action
 - E-Mail notifications
 - Tweets
 - Certain types of database updates
 - RSS aggregation
 - Search indexing
- Allows for batch operations

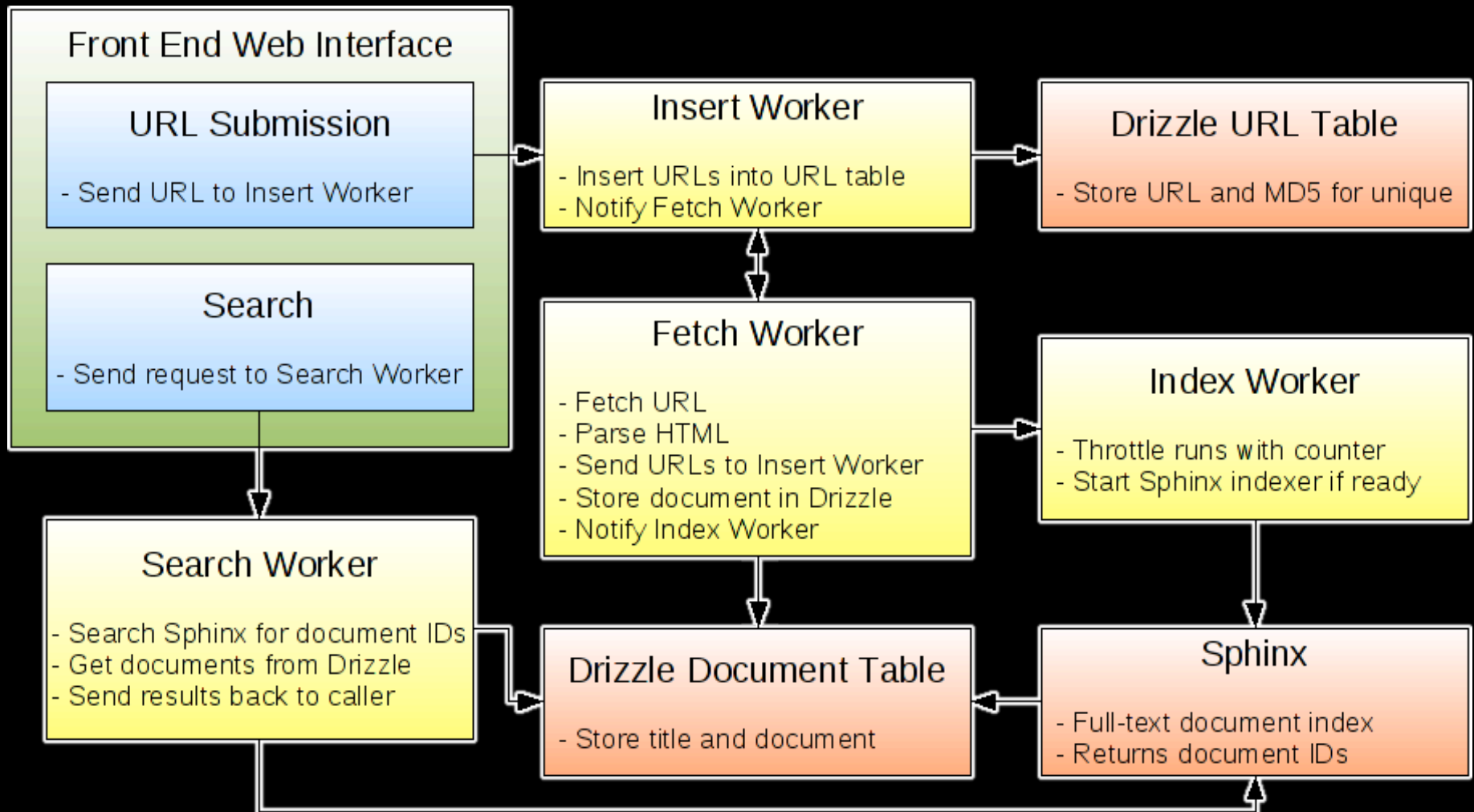


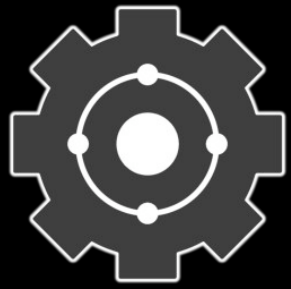
Narada

- Example in Patrick Galbraith's book
- Custom search engine
- Perl, PHP, and Java implementations
- Asynchronous queues
- Drizzle or MySQL
- Optionally use memcached
- Easy to integrate into existing projects
- <https://launchpad.net/narada>



Narada





URL Submission

Narada Search

NARADA SEARCH

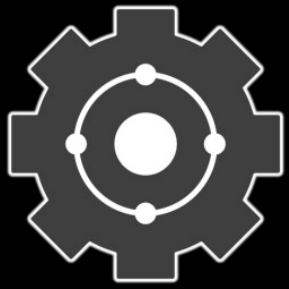
Search:

Submit URL:

Indexing URL: <http://localhost/src/nwveg/>

!

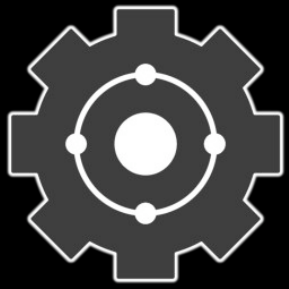
!



Insert Worker

- Insert URL into table
- Notify Fetch Worker

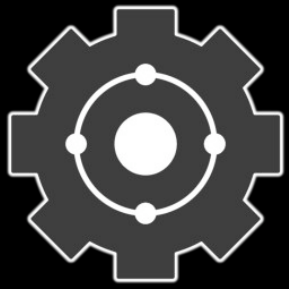
```
shell$ php NaradaInsert.php  
Insert URL: http://localhost/src/nwveg/
```



Fetch Worker

- Fetch URL, process HTML, find more URLs
- Store document and notify Index Worker

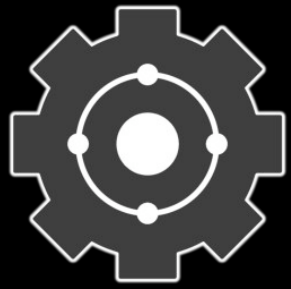
```
shell$ php NaradaFetch.php
Fetching: http://localhost/src/nwveg/
Protocol: http://
Domain: localhost
Path: /src/nwveg/
File:
Local URL: http://localhost/src/nwveg/index.php
Local URL: http://localhost/src/nwveg/news.php
Local URL: http://localhost/src/nwveg/whyveg.php
Local URL: http://localhost/src/nwveg/board.php
Local URL: http://localhost/src/nwveg/subscribe.php
Local URL: http://localhost/src/nwveg/discounts.php
...
Skipping Remote URL: http://twitter.com/northwestveg
Title: Northwest VEG - Home
```



Insert Worker

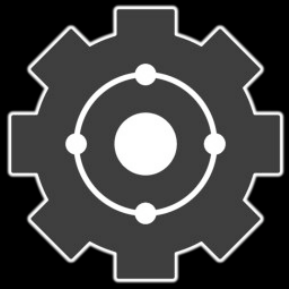
- More URLs from Fetch Worker

```
shell$ php NaradaInsert.php
Insert URL: http://localhost/src/nwveg/
Insert URL: http://localhost/src/nwveg/index.php
Insert URL: http://localhost/src/nwveg/news.php
Insert URL: http://localhost/src/nwveg/whyveg.php
Insert URL: http://localhost/src/nwveg/mvp.php
Insert URL: http://localhost/src/nwveg/veg101.php
Insert URL: http://localhost/src/nwveg/join.php
Insert URL: http://localhost/src/nwveg/volunteer.php
Insert URL: http://localhost/src/nwveg/events.php
...
```



Insert/Fetch Workers

- URL Feedback Loop
- Filter at some point
 - Recursion level
 - Restrict to given domain
- Now ready to index
- ...then search!



Index Worker

- Don't run indexer for every document

```
shell$ php NaradaIndex.php
Index Count: 1
...
Index Count: 10
Running Index
Sphinx 0.9.9-rc2 (r1785)
Copyright (c) 2001-2009, Andrew Aksyonoff

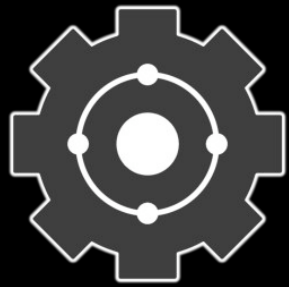
using config file '/home/eday/other/sphinx/etc/sphinx.conf'...
indexing index 'url'...
collected 25 docs, 0.1 MB
sorted 0.0 Mhits, 100.0% done
total 25 docs, 61524 bytes
total 0.085 sec, 718134 bytes/sec, 291.81 docs/sec
indexing index 'url_delta'...
collected 0 docs, 0.0 MB
total 0 docs, 0 bytes
total 0.045 sec, 0 bytes/sec, 0.00 docs/sec
distributed index 'dist_url' can not be directly indexed; skipping.
total 3 reads, 0.000 sec, 30.4 kb/call avg, 0.0 msec/call avg
total 9 writes, 0.000 sec, 8.3 kb/call avg, 0.0 msec/call avg
rotating indices: succesfully sent SIGHUP to searchd (pid=17010).
```



Search Worker

- Query Sphinx for document IDs
- Get documents from Drizzle
 - Soon will get from memcached too

```
shell$ php NaradaSearch.php
1 Documents matched 'nonprofit'
2 Documents matched 'board'
5 Documents matched 'oregon'
3 Documents matched 'vegfest'
7 Documents matched 'volunteer'
```



Search

Narada Search

NARADA SEARCH

Search:

Submit URL:

Northwest VEG - Master Veg

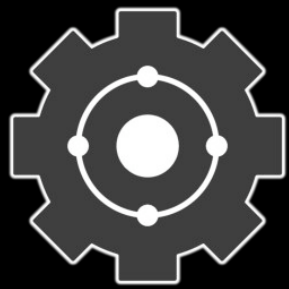
... contribute at least 16 *volunteer* hours to a nonprofit... contribute at least 16 *volunteer* hours to sharing the... completion of training and *volunteer* work. © 2008 Northwest...

LAST UPDATED: 2009-07-14 00:01:48

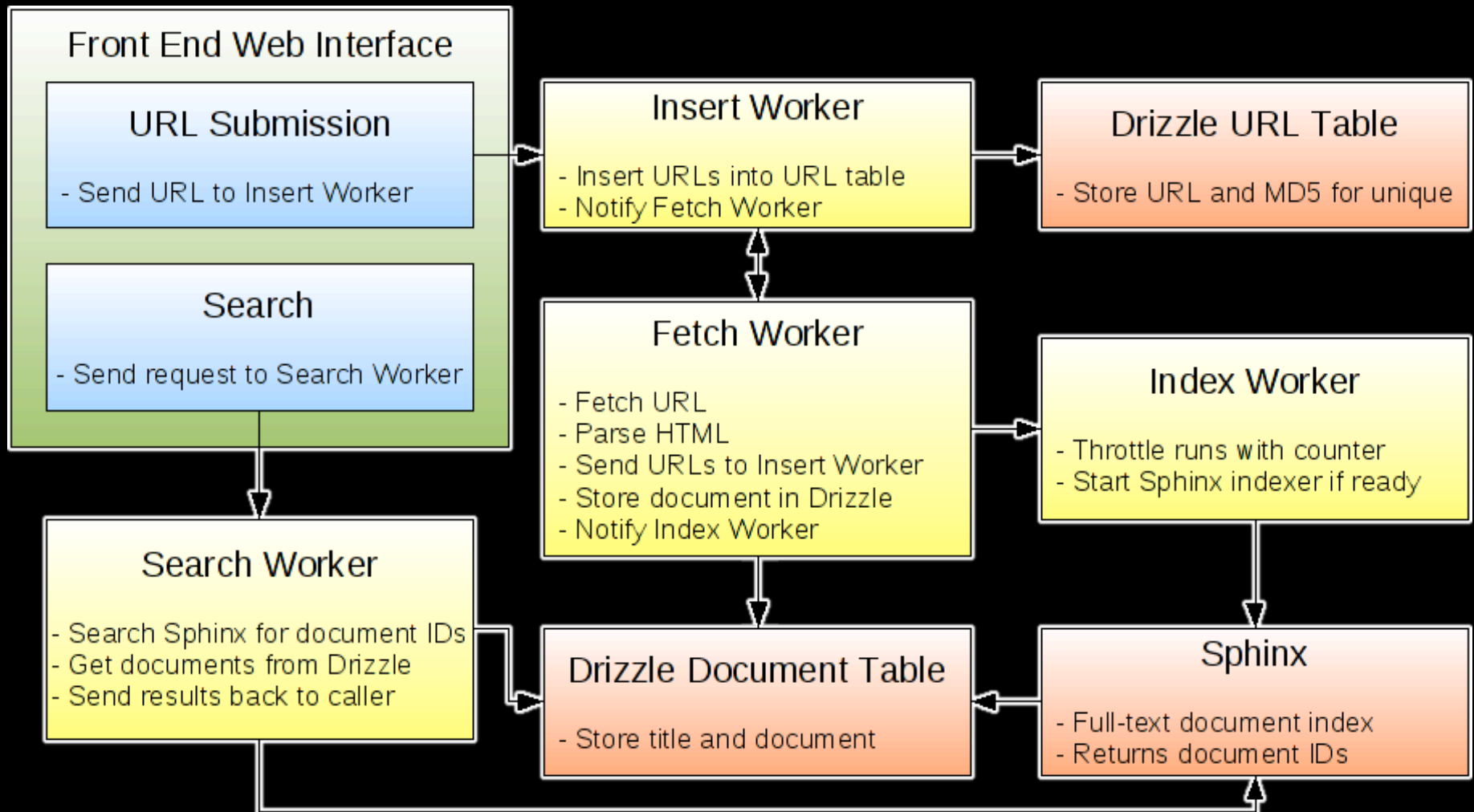
Northwest VEG - Join Us

... membership, it is possible to contribute *volunteer* hours in exchange for membership. For... information, contact Wendy Gabbe Day at *volunteer@nwveg.org*. © 2008 Northwest ...

LAST UPDATED: 2009-07-14 00:01:48



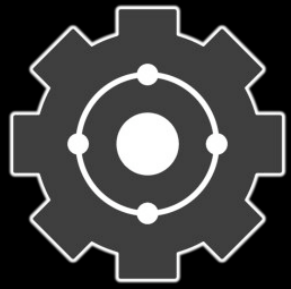
Narada





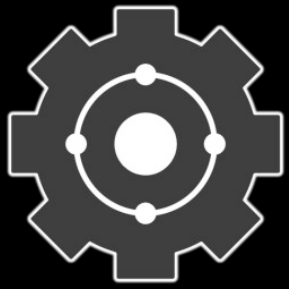
Other Applications

- MogileFS
- Distributed e-mail storage
- Gearman Monitor Project
 - Configuration management (elastic)
 - Statistics gathering
 - Monitoring
 - Modular (integrate existing tools)
- What will you build?



What's Next?

- More protocol and queue modules
- TLS, SASL, multi-tenancy
- Replication (job relay)
- Job result cache (think memcached)
- Improved statistics gathering and reporting
- Event notification hooks
- Monitor service



Get involved

- <http://gearman.org/>
- #gearman on irc.freenode.net
- <http://groups.google.com/group/gearman>
- Gearman @ OSCON
 - 45 Minute Session (similar material)
 - Birds of a Feather (BoF)
 - Expo Hall Booth