Batch vs. Queue: an API Smackdown

John VanDyk
drupal.org: jvandyk irc: clouseau
DrupalCon San Francisco 2010

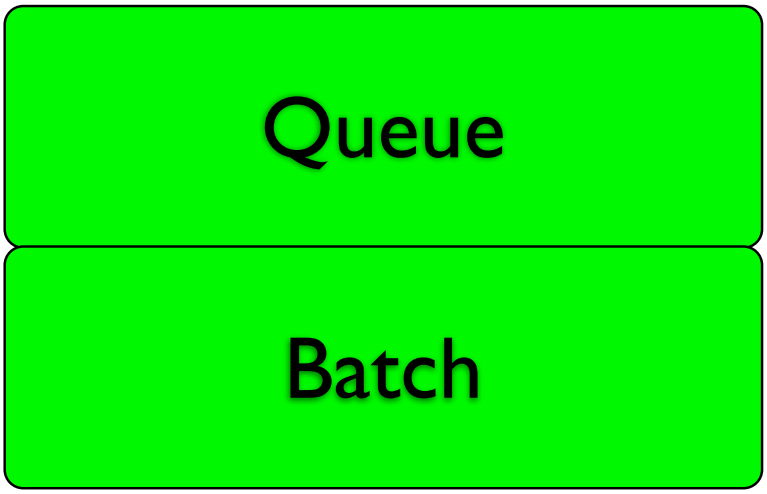
History

- December 6, 2005: drumm and Unconed create progress indicator for update.php
- December 22, 2006: yched notes that long-running CCK operations may hit the max execution time for PHP
- December 23, 2006: KarenS sees that the install code has defense against timeout

History

- March 4, 2007: chx notices that rebuilding node access needs a solution for long-running queries
- March 5, 2007: Gabor asks if it can be abstracted out of update.php
- *magic happens here*
- May 4, 2007: batch.inc added to core

This presentation

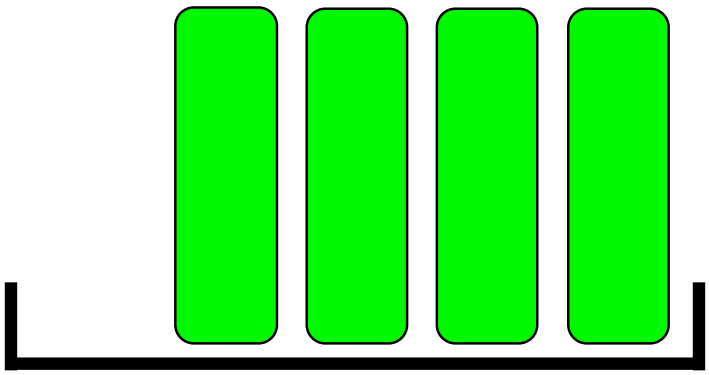


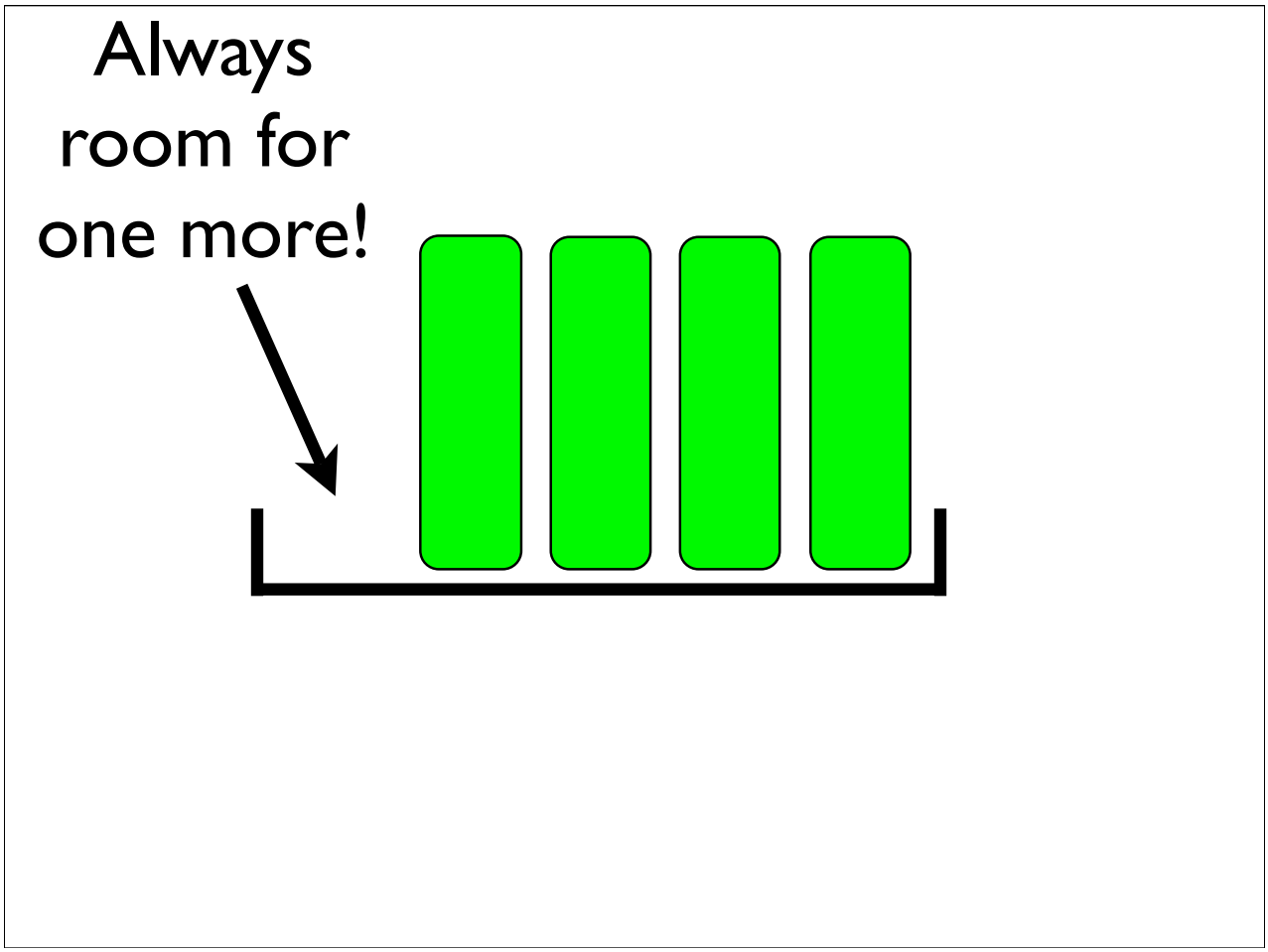
What is a Queue?

Requirement One:
an Apple store...

http://www.flickr.com/photos/gubatron/4486192745/sizes/o/

Requirement two: a new Apple product





Unless you run out of resources, e.g. memory for an in-memory queue or drive space for disk drive-based queue.

Quick Terminology Break

- Object-oriented programming
- Classes of objects
- Interfaces are clearly defined methods of interacting with objects

Drupal's Queue Interface

createQueue()
deleteQueue()

createItem($item)
deleteItem($item)

claimItem()
releaseItem($item)

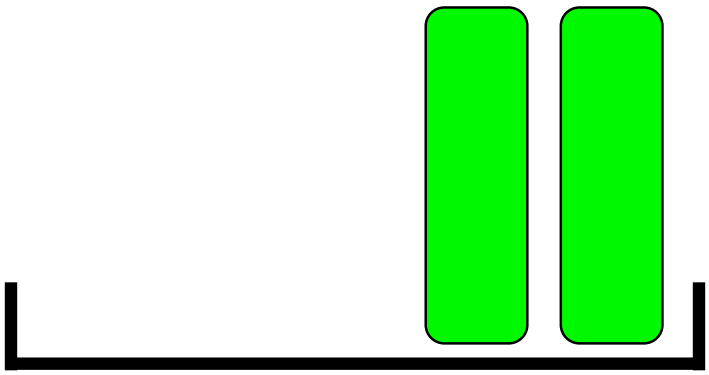
numberOfItems()

Any queue that implements (that is, uses) Drupal's Queue interface will have these methods. See modules/system/system.queue.inc.

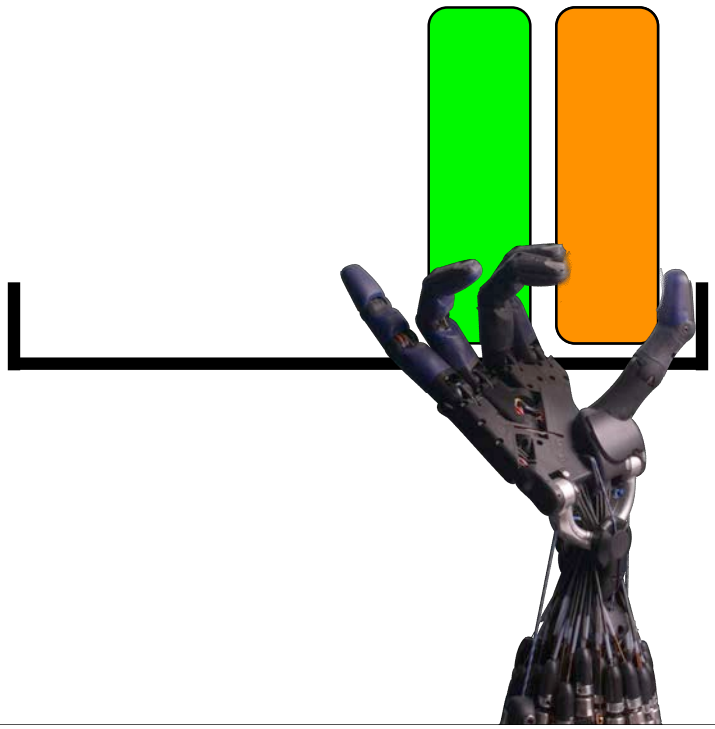
`Queue in core by chx, dww, neclimdul, Crell, alex_b, et al`

createQueue()

createItem($data)

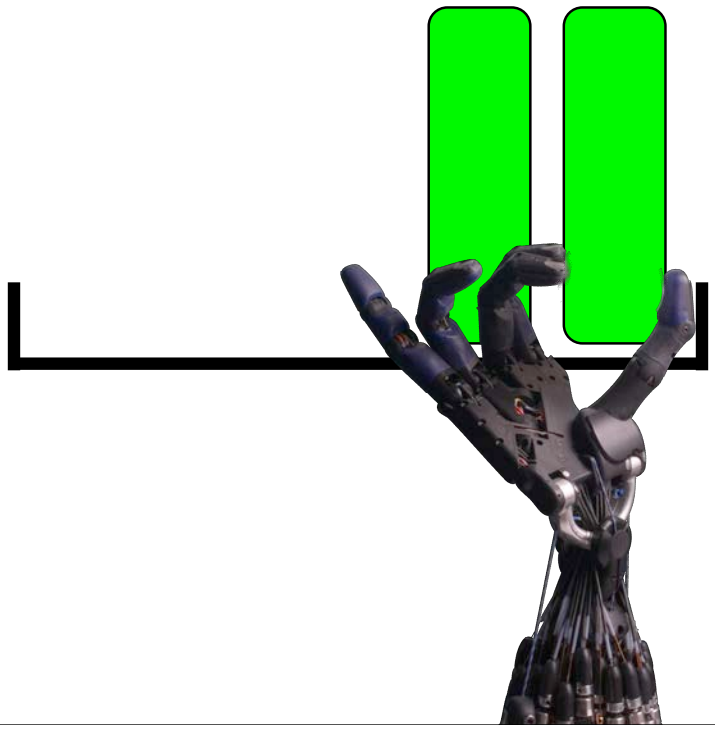


claimItem($seconds)

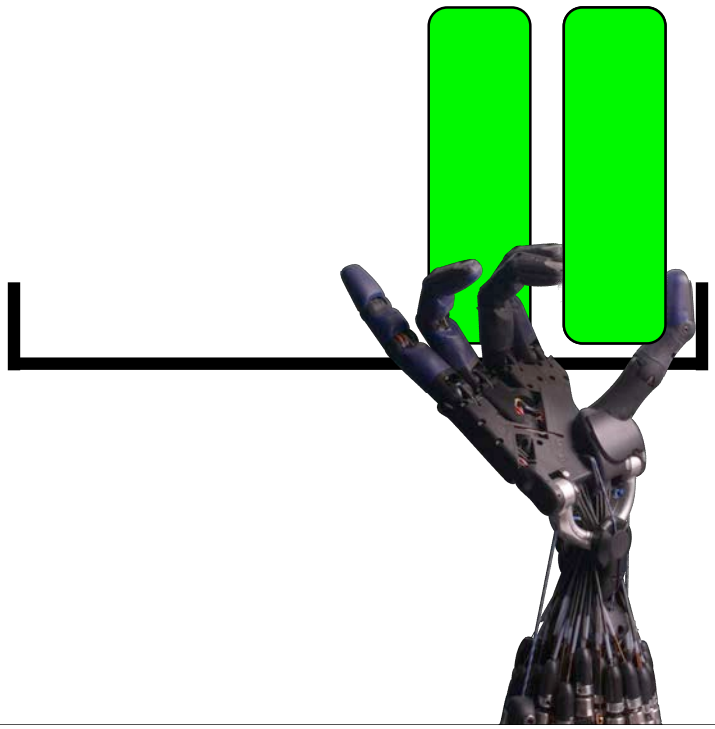


claimItem() gives you an exclusive claim on a queue item for a period of time
Robotic hand from http://commons.wikimedia.org/wiki/File:Shadow_Hand_Bulb_large_Alpha.png

deleteItem()



releaseItem()



Queue Classes in Core

- SystemQueue
- MemoryQueue
- BatchQueue extends SystemQueue
- BatchMemoryQueue extends MemoryQueue

Class	Storage	FIFO	Lease Time	getAllItems()
SystemQueue	Database	Yes*	Yes	No
BatchQueue	Database	Guar-anteed	No	Yes
MemoryQueue	Memory	Yes*	Yes	No
BatchMemoryQueue	Memory	Guar-anteed	No	Yes

*FIFO in practice but pluggable backends or multiple consumers of the queue may change this

```php
<?php

  $queue = DrupalQueue::get('tofu_sandwich');

  $queue->createQueue(); // no-op.

  $things = array('bread', 'tofu', 'provolone',
'sprouts');

  foreach ($things as $item) {
    $queue->createItem($item);
  }
```

createQueue() does nothing in Drupal core's queues, but in your own it might do something like create the directory to hold queue contents, requisition a new EC2 server to handle queues, or text message you to let you know that a new queue was created.

queue table

item_id Primary Key: Unique item ID.	name The queue name.	data The arbitrary data for the item.	expire Timestamp when the claim lease expires on the item.	created Timestamp when the item was created.
1	tofu_sandwich	s:5:"bread";	0	1271722100
2	tofu_sandwich	s:4:"tofu";	0	1271722100
3	tofu_sandwich	s:9:"provolone";	0	1271722100
4	tofu_sandwich	s:7:"sprouts";	0	1271722100

How SystemQueue items are stored

```
$count = $queue->numberOfItems();

drupal_set_message(
  t('Queued up %count items.',
  array('%count' => $count))
);
```

numberOfItems() is a best guess if there are multiple queue consumers/populaters

```
  $items = array();
  while ($item = $queue->claimItem()) {
    $message .= $item->item_id . ':' .
                $item->data    . '; ';
    $items[] = $item;
  }

  drupal_set_message('Queue contains: ' .
check_plain($message));

  // Release claims on items in queue.
  foreach ($items as $item) {
    $queue->releaseItem($item);
  }
```

We can just claim and then release to see what's in the queue.

Inspecting an $item

```
stdClass Object (
  [item_id] => 3
      [data] => sprouts
  [created] => 1271721970
    [expire] => 1271722000
)
```

Here's what you get when you ask for an item from the queue. What you want is ->data.

Live Demo!

```
$retrieved_items = array();
while ($item = $queue->claimItem()) {
  $retrieved_items[] = array(
    'data' => array($item->data, $item->item_id)
  );
  $queue->deleteItem($item);
}

$variables = array(
  'header'     => array(t('ID'), t('Item')),
  'rows'       => $retrieved_items,
  'attributes' => array(),
  'caption'    => '',
  'colgroups'  => array(),
  'sticky'     => TRUE,
  'empty'      => t('No items.'),
);
return theme_table($variables);
```

Here is the queue dumper code. Each item is claimed, then deleted.

Pluggable Queue Backends

The default queue class is SystemQueue.

```
$queue = DrupalQueue::get('tofu_sandwich');


$queue = new SystemQueue('tofu_sandwich');
```

Use the first version. It goes through a factory class which allow swappability.

Changing default queue class

```
$class = variable_get('queue_default_class', 'SystemQueue')

variable_set('queue_default_class', 'MyOwnQueue')
```

Changing class for one queue only

Set variable for 'queue_class_' . $name

variable_set('queue_class_tofu_sandwich', 'OtherQueue')

See http://drupal.org/project/beanstalkd

Debugging: a Logging Queue

- We can subclass core queue classes
- We can set default queue classes per queue
- Therefore, a logging queue for debugging is easy easy!

Remember these?

createQueue()
deleteQueue()

createItem($item)
deleteItem($item)

claimItem()
releaseItem($item)

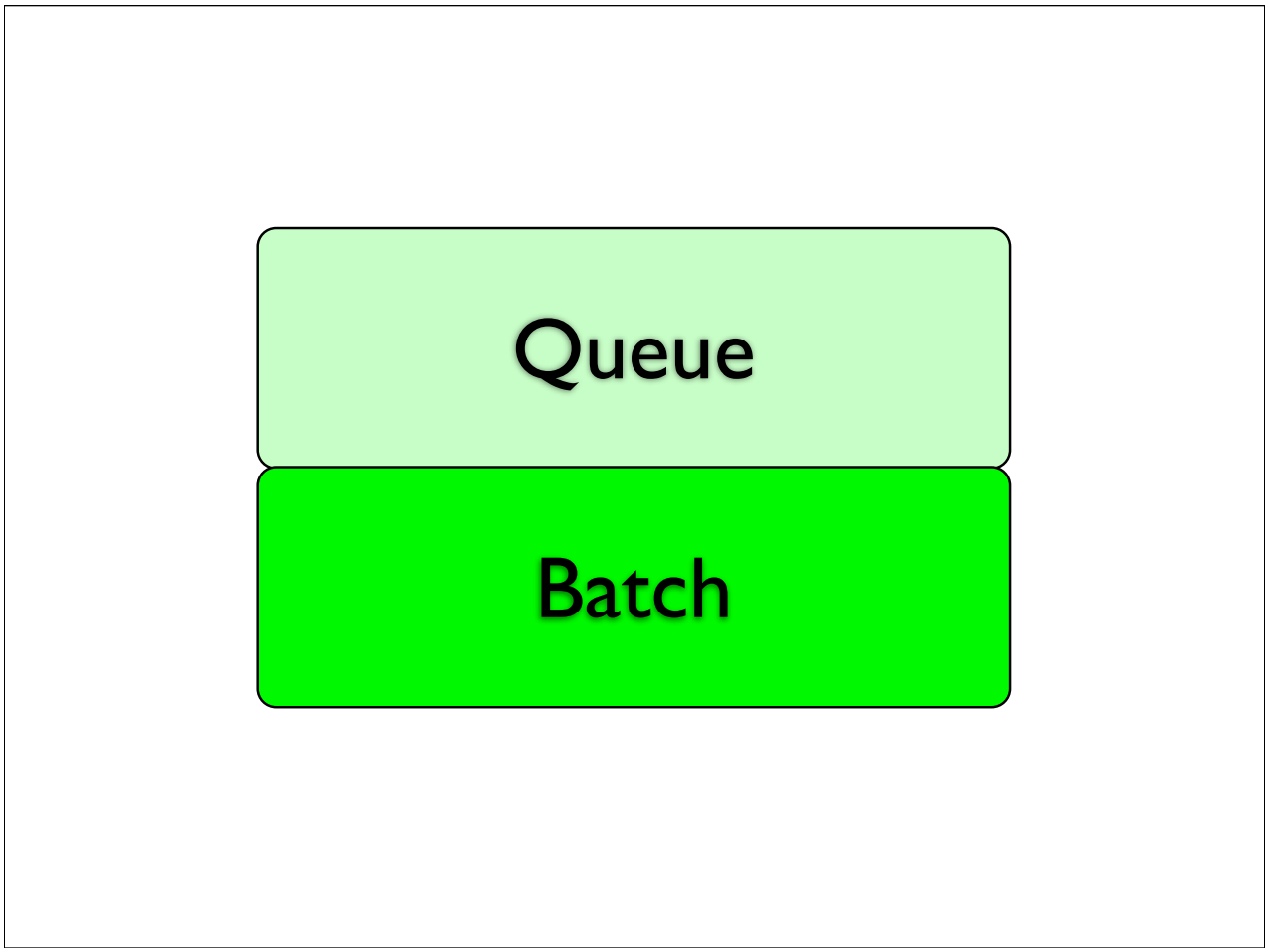
numberOfItems()

Our new subclass will just wrap all the methods!

```
class LoggingMemoryQueue extends MemoryQueue  {

  public function __construct($name) {
    watchdog('queue', 'Constructed queue: %name', array('%name' =>
$name));
    parent::__construct($name);
  }

  public function createItem($data) {
    watchdog('queue', 'Created item: %data', array('%data' =>
print_r($data, TRUE)));
    parent::createItem($data);
  }

  public function numberOfItems() {
    $count = parent::numberOfItems();
    watchdog('queue', 'Asked for numberOfItems: %num', array
('%num' => $count));
    return $count;
  }
...
```

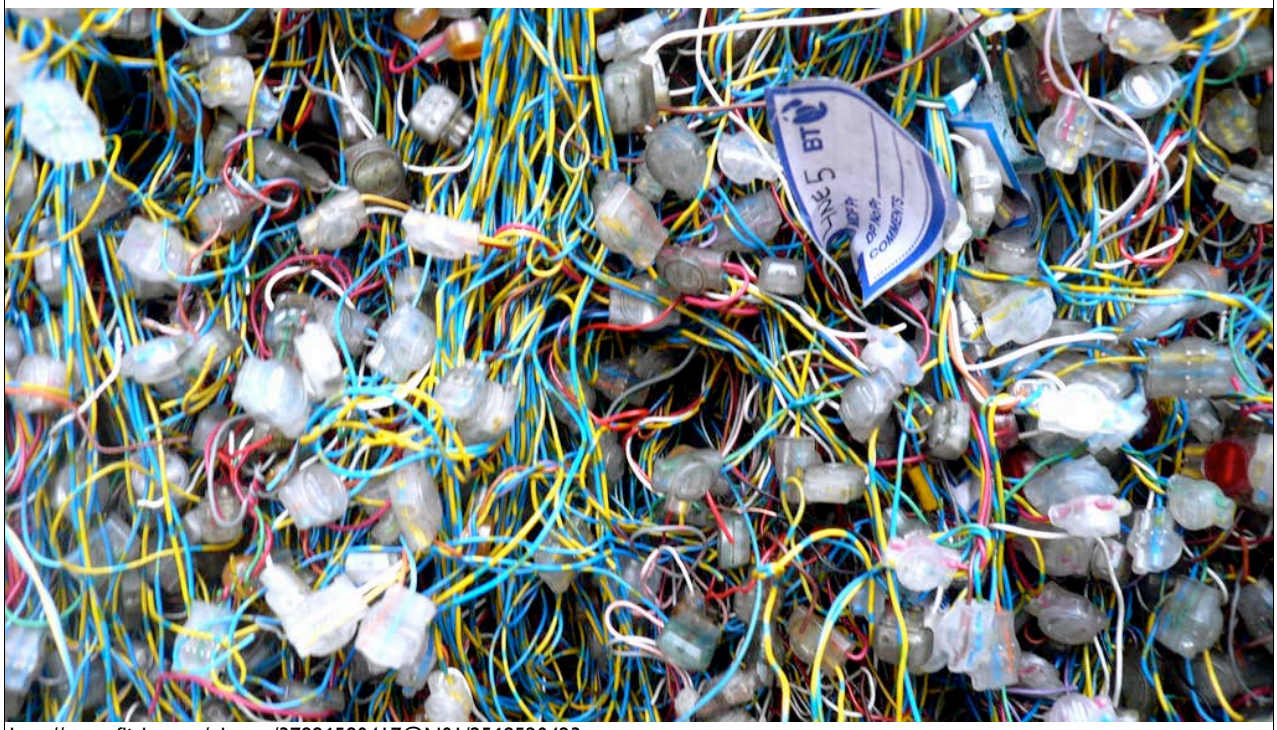
```
variable_set('queue_class_tofu_sandwich',
  'LoggingMemoryQueue');
```

04/19/2010 - 15:47	Released item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Released item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Released item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Released item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	All items have been claimed	jvandyk
04/19/2010 - 15:47	Claimed item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Claimed item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Claimed item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Claimed item: stdClass Object ([item_id] => ...	jvandyk
04/19/2010 - 15:47	Asked for numberOfItems: 4	jvandyk
04/19/2010 - 15:47	Created item: sprouts	jvandyk
04/19/2010 - 15:47	Created item: provolone	jvandyk
04/19/2010 - 15:47	Created item: tofu	jvandyk
04/19/2010 - 15:47	Created item: bread	jvandyk
04/19/2010 - 15:47	Queue created	jvandyk
04/19/2010 - 15:47	Constructed queue: queuer_sandwich	jvandyk



On to batches!

Batch API



http://www.flickr.com/photos/37996580417@N01/2569520423

chx: "No one understands it. But we treat it with reverence."
(I suspect some people do understand it judging by http://drupalcode.org/viewvc/drupal/drupal/includes/batch.inc?view=log)

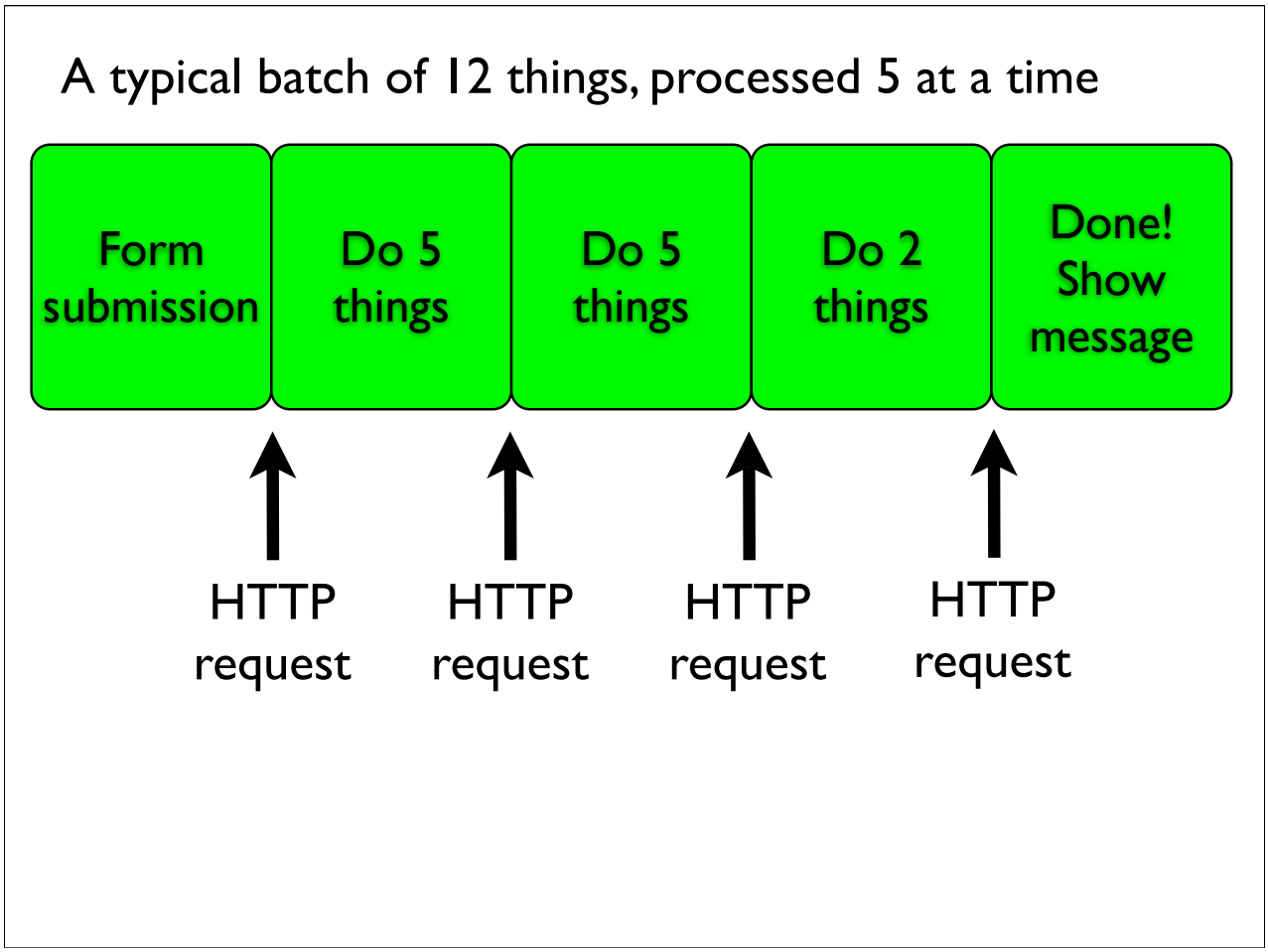
What's a batch?

- A bunch of things that need to be done

The Enemies

1) max_execution_time

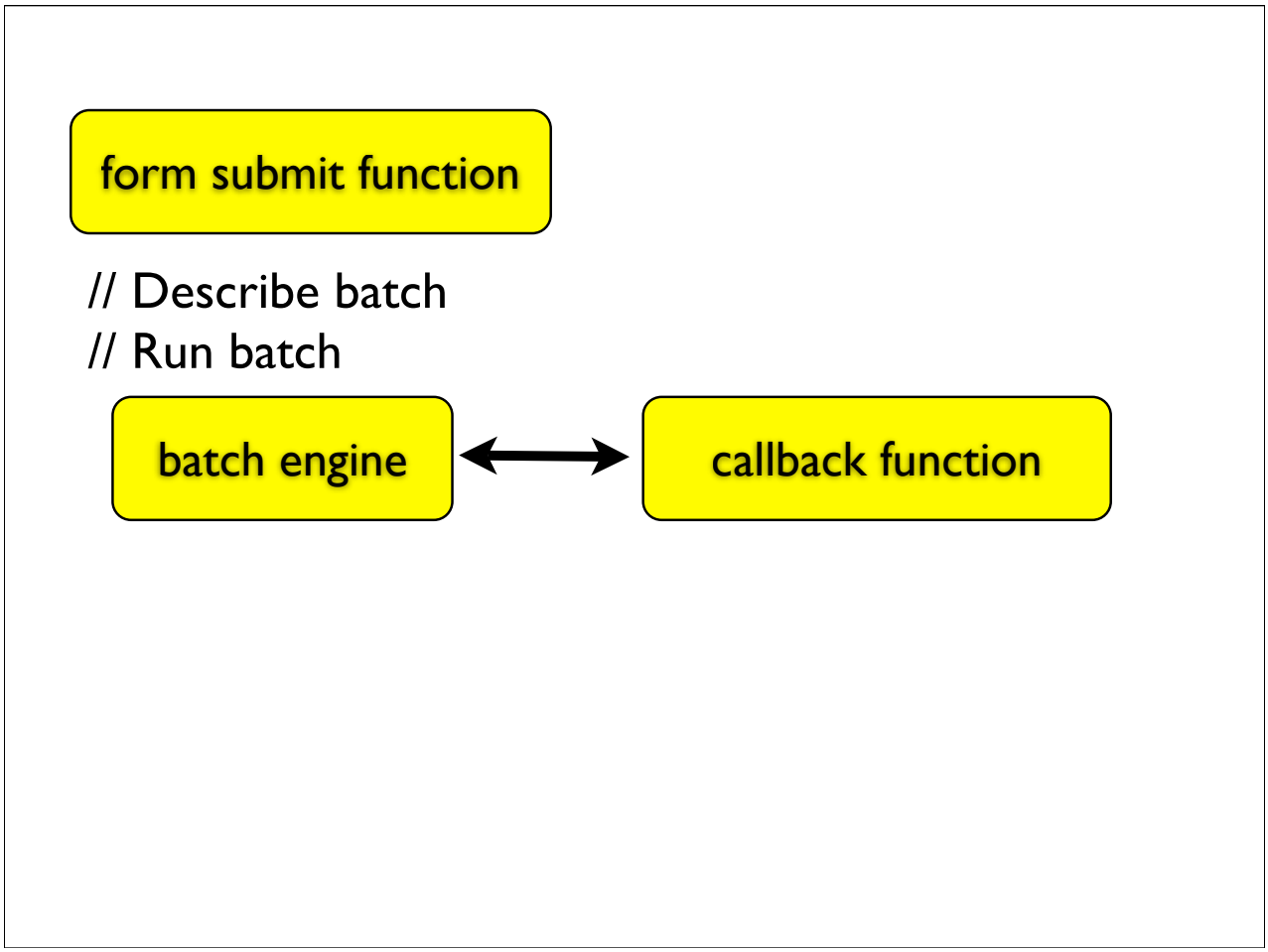
2) impatient users



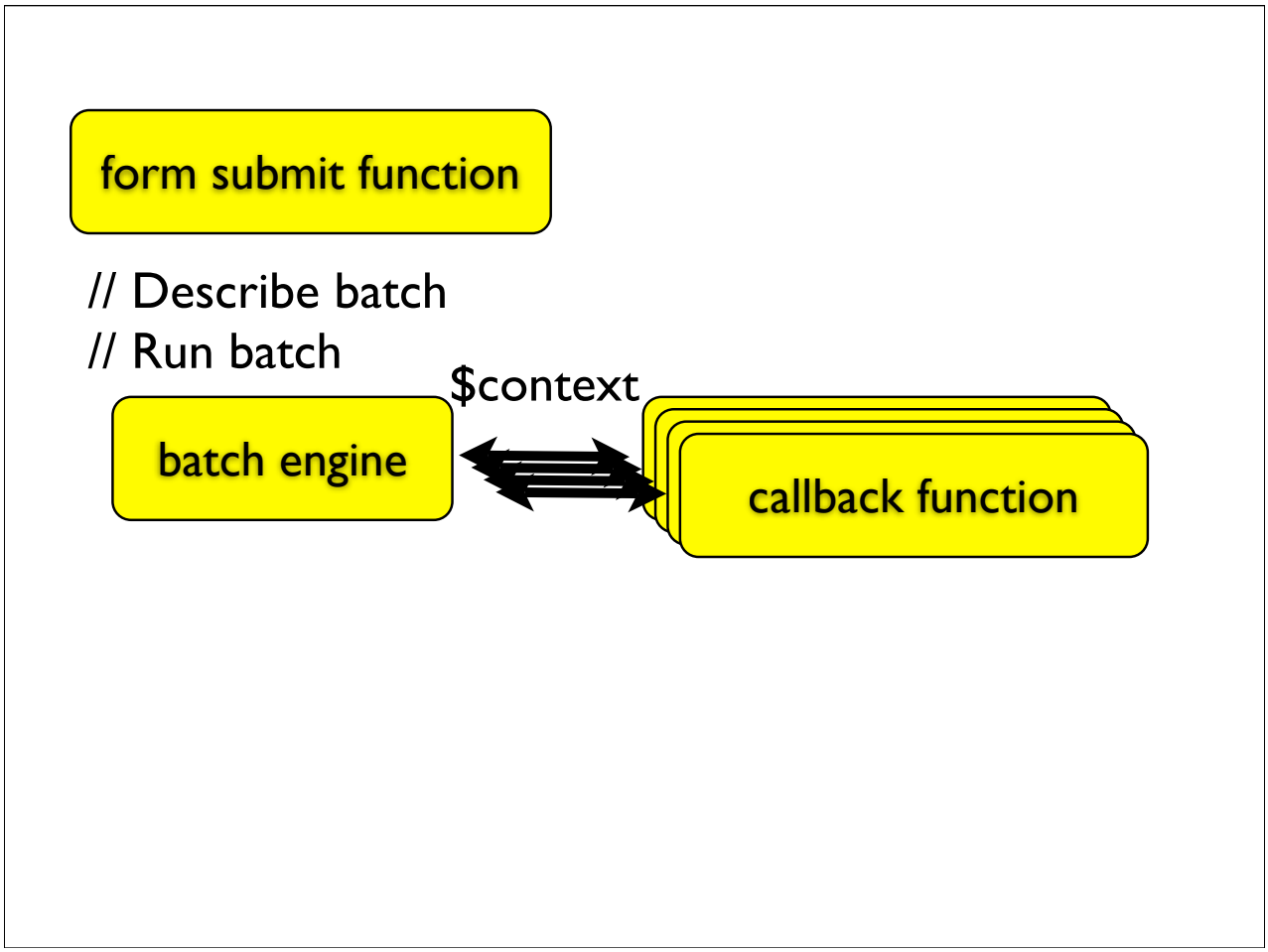
You've seen this when installing Drupal.

Batch basics

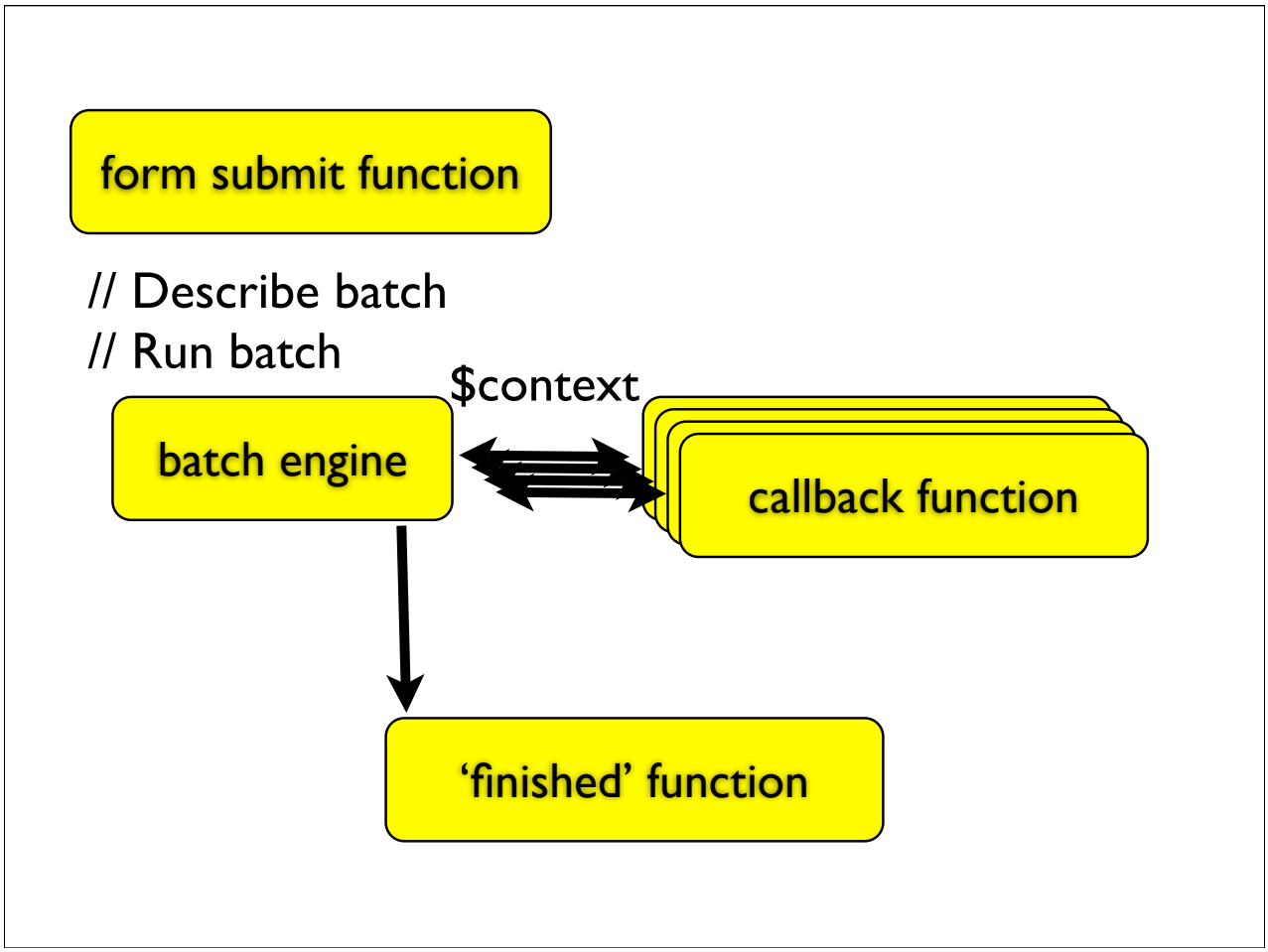
- What are the things that need to be done?
- How do we tell when we are done?
- What happens if PHP times out?



The main parts of a form-submit-initiated batch run

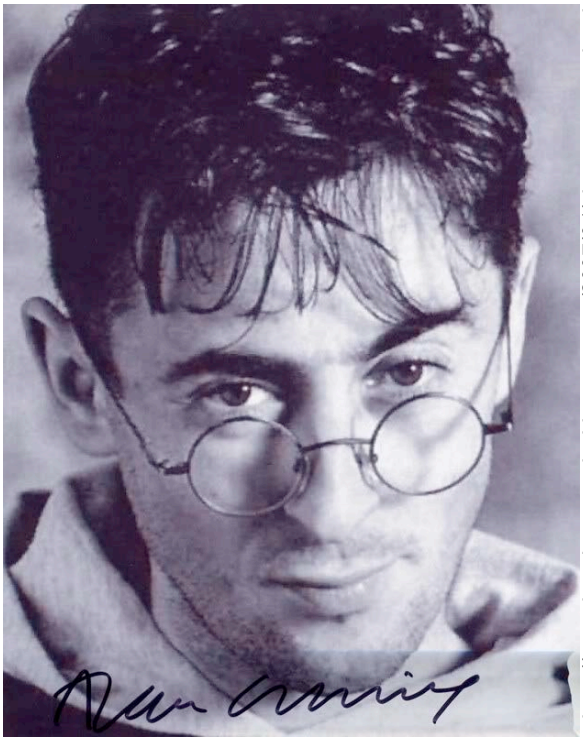


You can have as many callback functions as you want.
A $context array is passed each time so the callback function can keep track of what is going on.



You may also specify a function that runs when all the callbacks are done.

Give me the code!



http://jamesbondautographs.blogspot.com/2007/04/alan-cummings.html

Alan Cumming as Boris Grischenko. Goldeneye.

```php
function batchler_form($form, &$form_state) {
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Begin'),
  );
  return $form;
}
```

Introducing the batchler module. Can't get much simpler than this.

```php
function batchler_form_submit($form, &$form_state) {
  $batch = array(
    'operations' => array(
      array('batchler_callback1', array()),
    ),
  );

  batch_set($batch);
}
```

The minimum required batch description. Calling one callback with no parameters.
Passing the batch to batch_set() sets the batch engine running.

```
function batchler_form_submit($form, &$form_state) {
  $batch = array(
    'operations' => array(
      array(
        'batchler_callback1',
        array(),
      ),
      array(
        'batchler_callback2',
        array('parameter1', 'parameter2'),
      ),
    ),
  );

  batch_set($batch);
}
```

Several different operations can be defined along with parameters that will be passed to them when they are called.

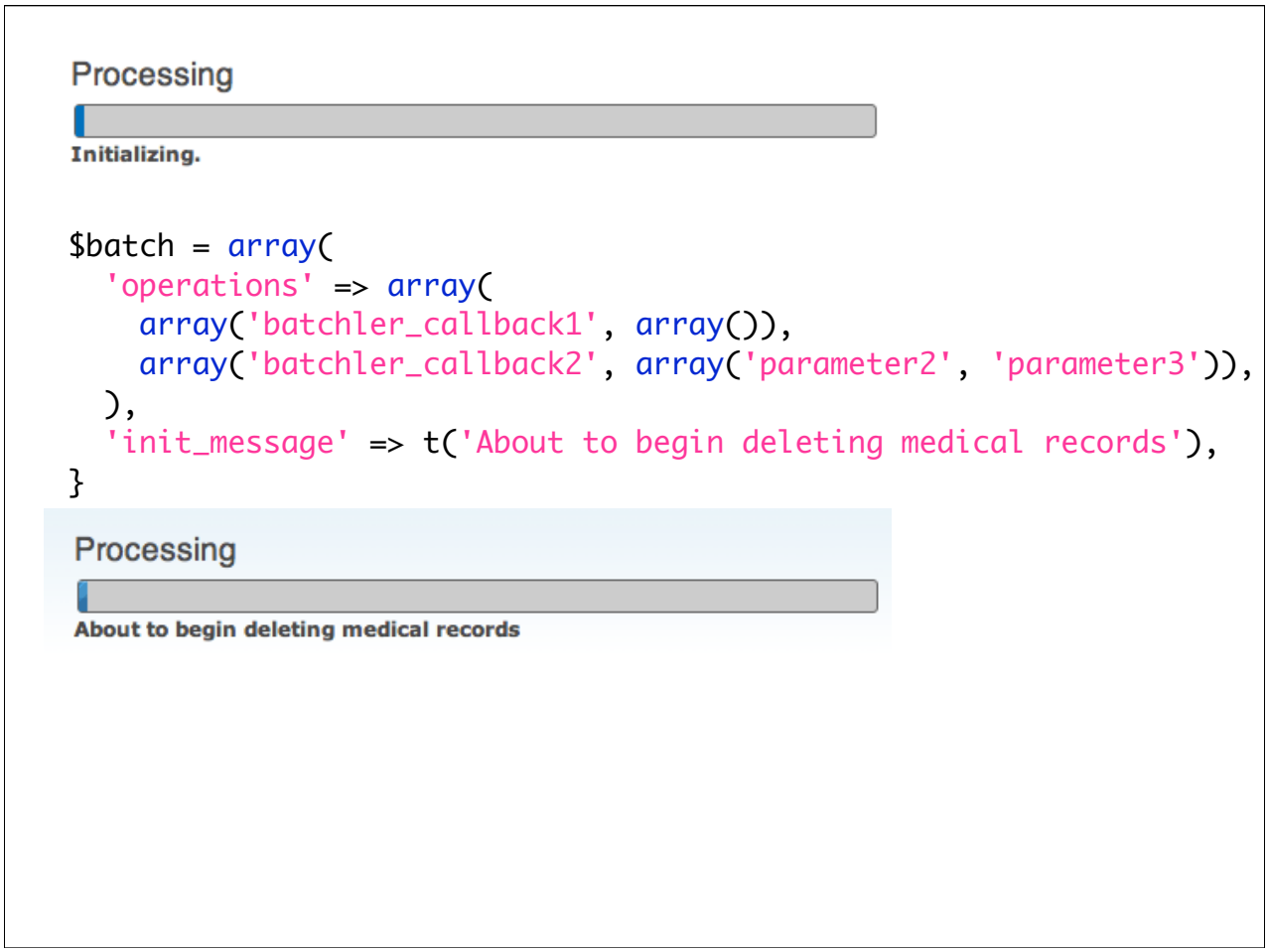
```
function batchler_callback1(&$context) {
  drupal_set_message('batchler_callback1 got
called');
}
```

This is all you need for your callback. No magic. Just a function that gets called!

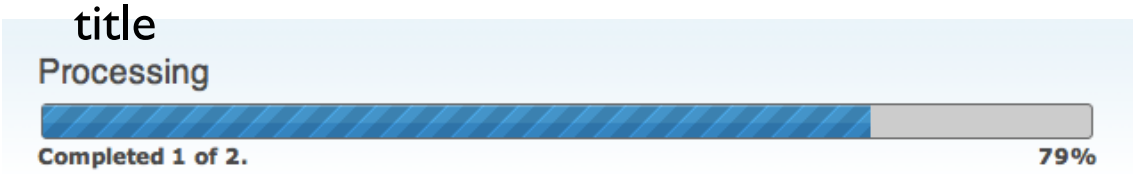
```php
function batchler_callback1($p1, $p2, &$context) {
  if (!isset($context['sandbox']['iteration'])) {
    $context['sandbox']['iteration'] = 0;
  }
  $context['sandbox']['iteration']++;

  $context['finished'] =
            $context['sandbox']['iteration']/10000;
}
```

But you can do a lot more. This callback will be called many times.
The sandbox is your area for persistent storage. 'finished' is set to 1 the first time the callback is called. Dial it back from 1 to <1 and you will keep getting called until you set it to 1.

Live Demo!



We have both a finished key and a finished callback.




```
$batch = array(
  'operations' => array(
    array('batchler_callback', array()),
    array('batchler_callback3', array('parameter2',
'parameter3')),
  ),
  'init_message' => t('About to begin deleting medical records'),
  'title' => t('Processing'),
  'progress_message' => t('Completed @current of @total'),
}
```

Other useful keys are title and progress_message.


```
$batch = array(
  'operations' => array(
    array('batchler_callback', array()),
    array('batchler_callback3', array('parameter2',
'parameter3')),
  ),
  'init_message' => t('About to begin deleting medical records'),
  'title' => t('Processing'),
  'progress_message' => t('Current: @current | Remaining:
@remaining | Total: @total | Percentage: @percentage | Estimate:
@estimate | Elapsed: @elapsed'),
}
```

There are other placeholders you can use in your progress message.

```
$batch = array(
  'operations' => array(
    array('batchler_callback', array()),
    array('batchler_callback3', array('parameter2', 'parameter3')),
  ),
  'init_message' => t('About to begin deleting medical records'),
  'title' => t('Processing'),
  'progress_message' => t('Completed @current of @total'),
  'error_message' => t('Something has gone horribly wrong'),
  'file' => drupal_get_path('module', 'batchler') .
    '/batchler.admin.inc',
}
```

You can use 'file' to bring callback functions into scope.

Batching Without a Form

```
function batchler_nofapi() {
  $batch = array(
    'operations' => array(
      array('batchler_callback1', array()),
      array('batchler_callback2', array('parameter2', 'parameter3')),
    ),
    'progress_message' => t('Current: @current | Remaining:
@remaining | Total: @total | Percentage: @percentage | Estimate:
@estimate | Elapsed: @elapsed')
  );
  batch_set($batch);
  batch_process('');
}
```

Use batch_process(). Demo of running batch directly from a menu callback.

When to use Batch

- When you're doing something that will exceed PHP's timeout
- When you're writing something that could get large
- When you want to give users lots of feedback on what is happening

Similar to pager support. Put it in, because someone will try to use your module with 1.5 million nodes.

When to use Queue

- When you want to stash things for later processing
- When you want to distribute processing
- When you want a queue with a twist: subclass one of the core classes

You can do both!

```
/**
 * Process a step in the batch for fetching
 * available update data.
 */
function update_fetch_data_batch(&$context) {
  $queue = DrupalQueue::get('update_fetch_tasks');
  ...
}
```

See update.module in Drupal 7 for an example.

Further resources

Batch

- http://api.drupal.org/api/search/7/batch
- http://drupal.org/project/examples
- *Pro Drupal Development* pp. 560-570

Queue

- http://api.drupal.org/api/group/queue/7
- http://drupal.org/project/queue_ui
- modules/system/system.queue.inc