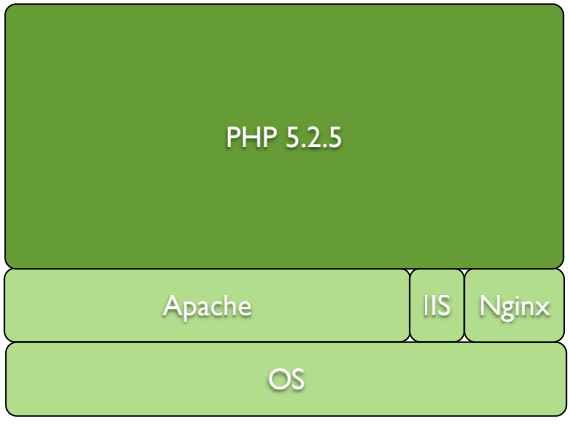
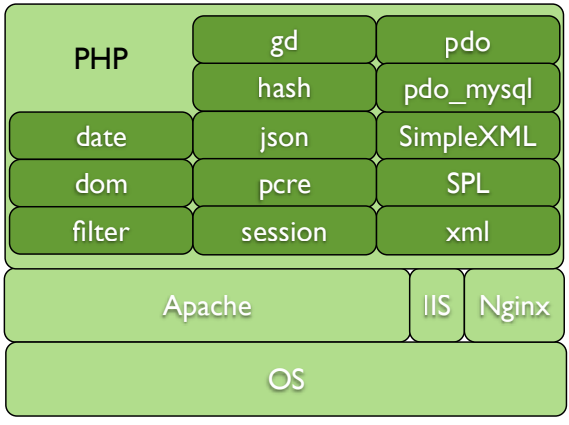
An Introduction to Drupal Architecture

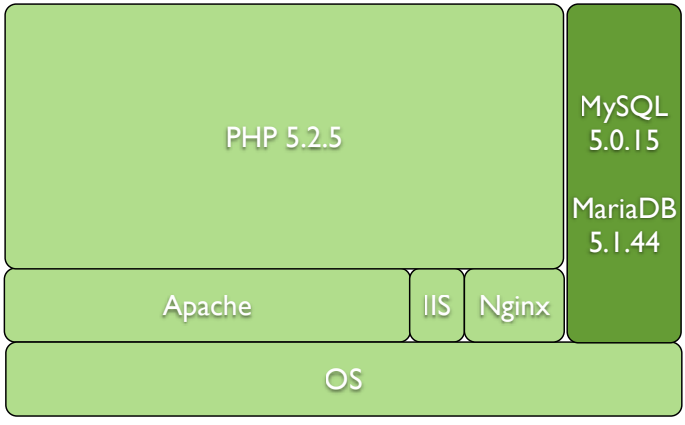
John VanDyk
DrupalCamp Des Moines, Iowa
September 17, 2011



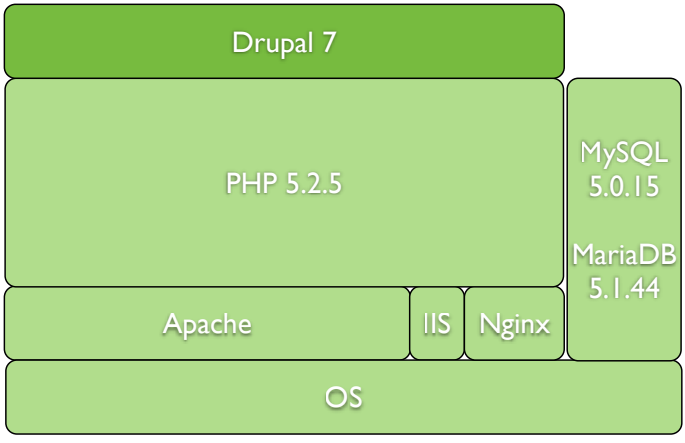
Stack with OS, webserver and PHP. Most people use mod_php but deployments with engineers at the helm sometimes use FastCGI. PHP 5.2.4 with the 5.2.5 security backport is fine too (Ubuntu 8.0.4).



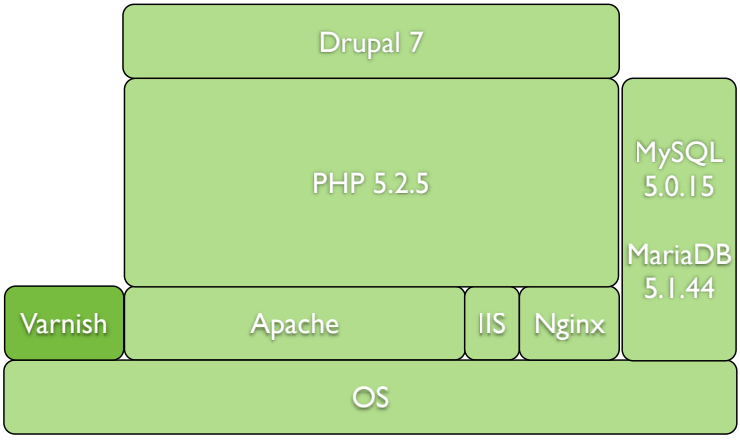
Some PHP extensions are required.



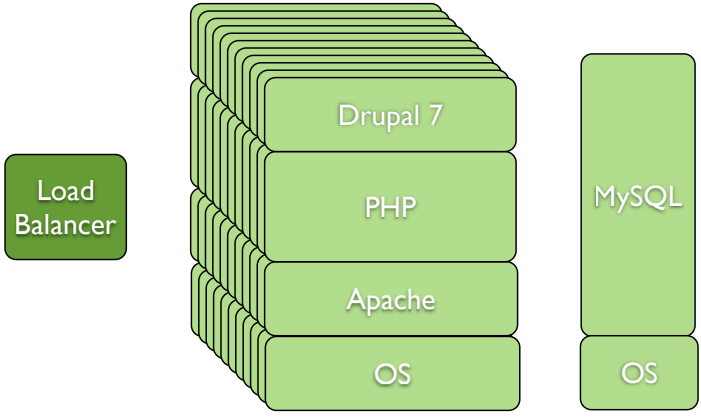
Database.



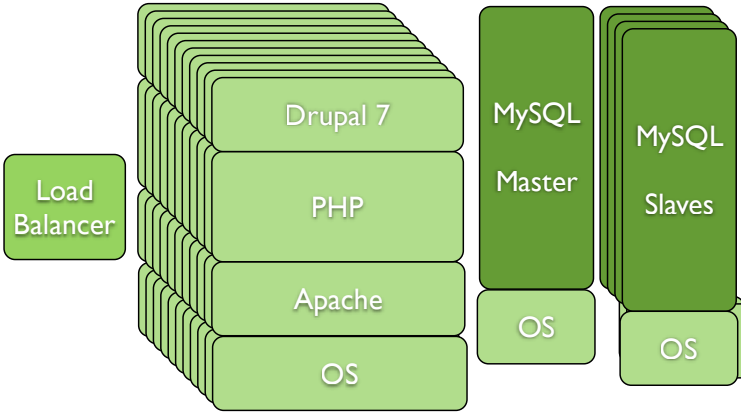
Drupal sits atop the stack.



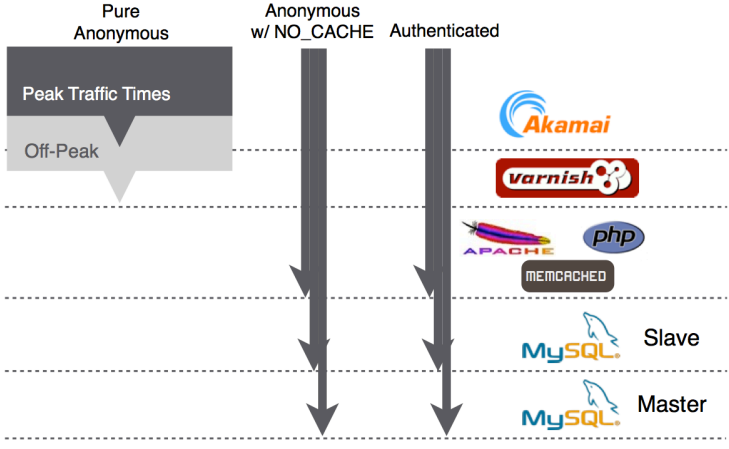
Varnish can be added to the front end for fast RAM-based response to anonymous users.



Webheads can be scaled out.

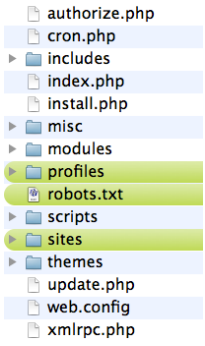


Databases can be scaled out. Show master-slave but master-master is also a possibility. See http://www.jochus.be/site/2011-01-13/drupal/drupal-master-master-replication-architecture for an example.



Source: Nate Haug, Lullabot *Scaling the Grammys*. DrupalCamp Denver 2010.

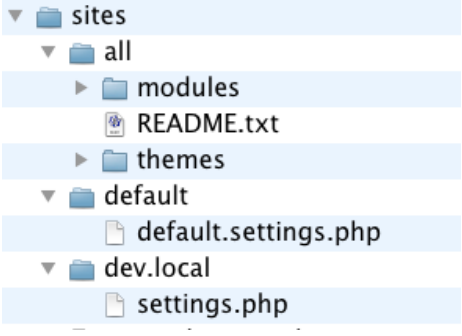
214 million hits in six hours at peak. Database load: 5%. Apache load: 1%.



You will not hack core.

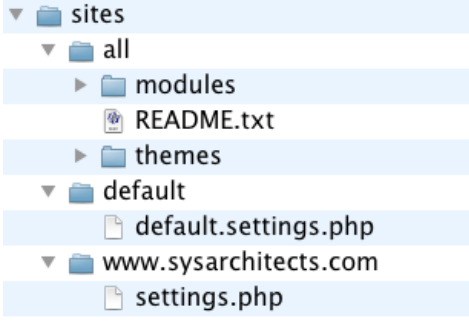
You should only modify the locations shown in green. Modifying the other files will inevitably lead to grief.

Development Site



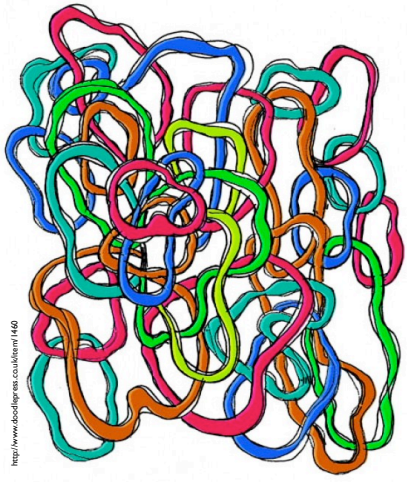
Local settings, including database connection info, are in sites/all/dev.local.

Live Site



On the live site settings are in a site-specific directory.

Drupal 7 Architecture



Drupal architecture. Each include file has a different color and their interrelationships are shown. Uh...just kidding.

Browser request: GET /node/1

Bootstrap (includes/bootstrap.inc)
- brings session, database, variables online

Menu System (router)
- maps URL to a function
- checks access control
- may bring functions into scope
- calls function to build page data structure
- calls function to theme page (look and feel)
- page returned to client

It is important for a developer to understand the sequence of events that occur when a request is processed by Drupal.

/node/1

mod_rewrite in .htaccess

/index.php?q=node/1

$_GET['q'] will contain 'node/1'

A typical request starts with mod_rewrite.

index.php

menu system

Which function should this path be mapped to?

```
$items['node/%node'] = array(
  'page callback' => 'node_page_view',
  'page arguments' => array(1),
  'access callback' => 'node_access',
  'access arguments' => array('view', 1),
);
```


index.php invokes the menu system, which handles much more than menus. The menu item that maps to node/1 can be found in node.module.

index.php

menu system

Does this user have permission to access this path?

```
$items['node/%node'] = array(
  'page callback' => 'node_page_view',
  'page arguments' => array(1),
  'access callback' => 'node_access',
  'access arguments' => array('view', 1),
);
```


node.module tells the menu system to invoke a function (node_access()) to determine whether the current user has permission.

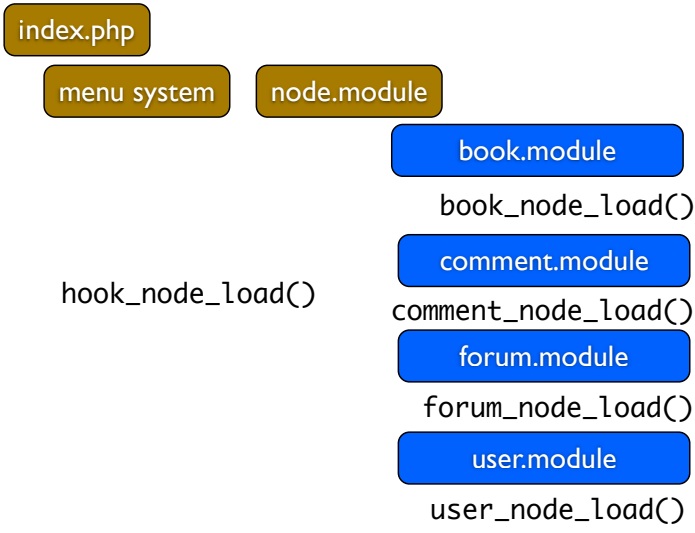
index.php

menu system

node.module

```
/**
 * Menu callback; view a single node.
 */
function node_page_view($node) {
...
}
```


The page callback for the path node/1 is 'node_page_view'. That's the name of the function that is called if the user has permission to view the page.



Other modules have a say in the loading of the node object. Many of these hooks (better thought of as "events") happen during the building of a page.

index.php

menu system

```
nodes = Array [2]
  #sorted = (boolean) true
  1 = Array [11]
    #bundle = (string:4) page
    #entity_type = (string:4) node
    #language = (string:2) en
    #node = Object of: stdClass
    #pre_render = Array [1]
    #theme = (string:4) node
    #view_mode = (string:4) full
    #weight = (int) 0
    body = Array [16]
      #access = (boolean) true
      #bundle = (string:4) page
      #entity_type = (string:4) node
      #field_name = (string:4) body
      #field_translatable = (string:1) 0
      #field_type = (string:17) text_with_summary
      #formatter = (string:12) text_default
      #items = Array [1]
      #label_display = (string:6) hidden
      #language = (string:3) und
      #object = Object of: stdClass
      #theme = (string:5) field
      #title = (string:4) Body
      #view_mode = (string:4) full
      #weight = (int) 0
      0 = Array [1]
        #markup = (string:11) <p>Hi.</p>
    comments = Array [0]
    links = Array [5]
```


A large array with the node's data structure is returned to the menu system.

index.php

menu system

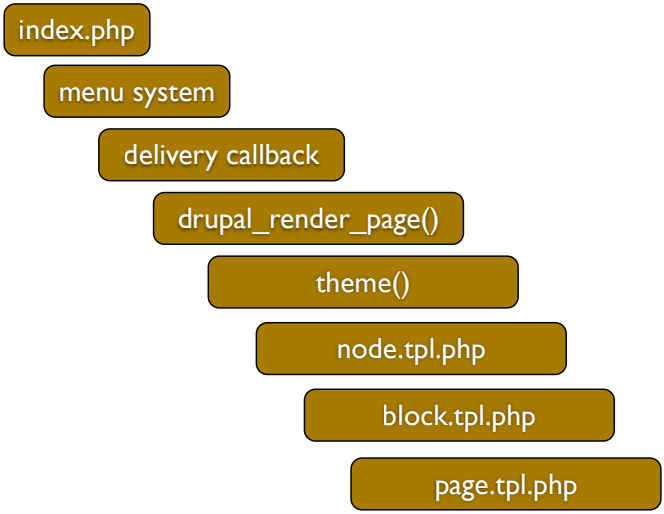
node.module

hook_node_view_alter()

drupalcorn.module

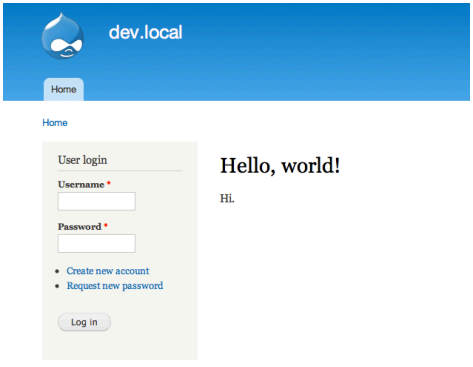
drupalcorn_node_view_alter()

Other modules may alter the final data structure. We've seen three common patterns that pervade Drupal: (1) hooks allow other modules to respond to events; (2) Drupal uses large structured arrays, and (3) other modules may alter these arrays once built.

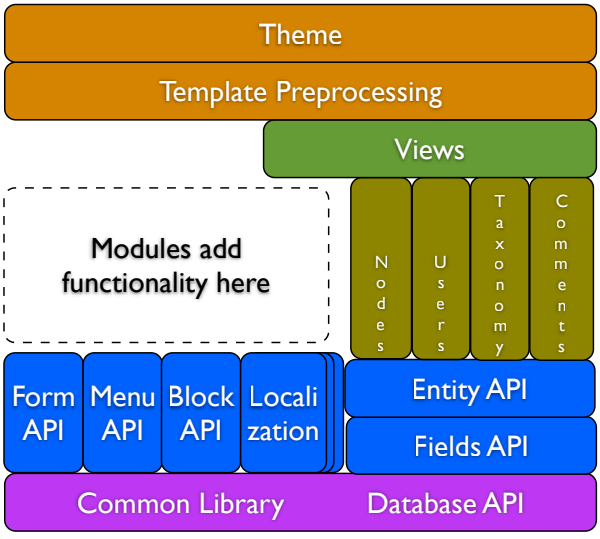


The menu system passes the structure array to a delivery callback which turns the array into HTML by rendering it; that is, passing it through Drupal's template engine. Template files (.tpl.php, pronounced "tipple-fipp") in the current theme are used to create the HTML.

index.php



The HTML is then returned to the browser.



Block diagram if you need a mental model. This doesn't have nearly enough dimensions to describe what actually happens. If you asked 5 developers to make Drupal block diagrams, you'd get 5 different diagrams. Hopefully this one helps someone.

"Entities"

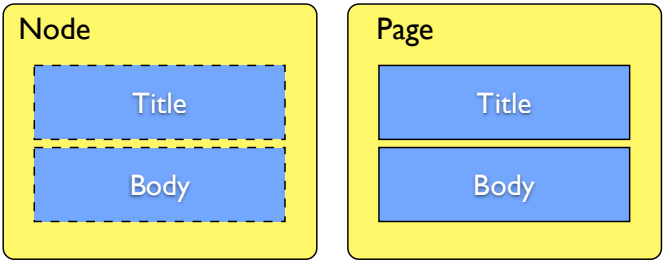
Node: a piece of content

User: a user account

Taxonomy: categories for tagging/classification

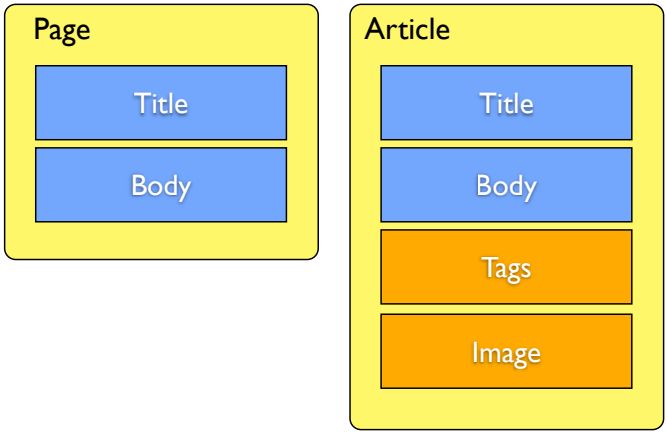
Comment: content with a parent

An Entity has Bundles



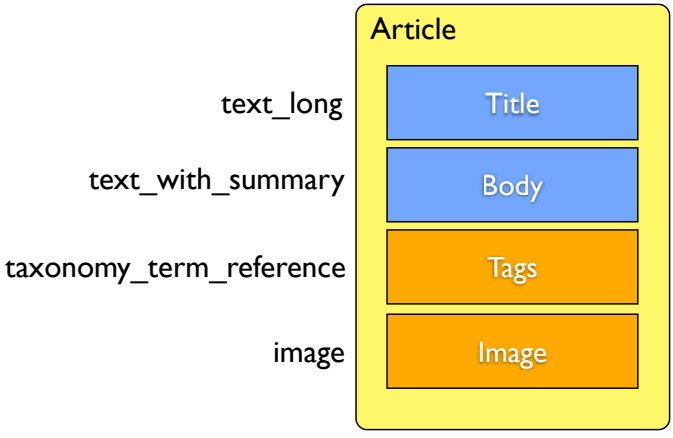
An entity can be thought of as a base object, and bundles are subclasses. For example, a node is an entity while the bundle of fields that make up a page are a type of node.

An Entity has Bundles



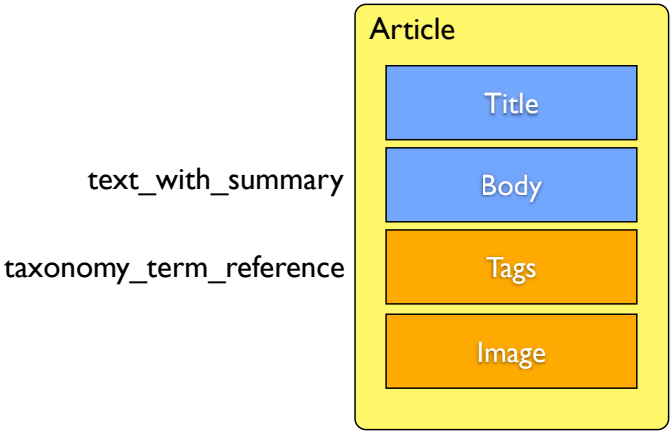
These are types of nodes. Each has a bundle of fields.

A Bundle has Fields



And each field has a type.

Not SQL Fields



These are more than SQL fields. text_with_summary autogenerates a summary. taxonomy_term_reference splits multiple tags into separate records. These are Drupal fields.

In the Real World

"Entity" = node | user | comment | taxonomy

"Bundle" = "Content Type"

You assemble content types from fields using the UI.

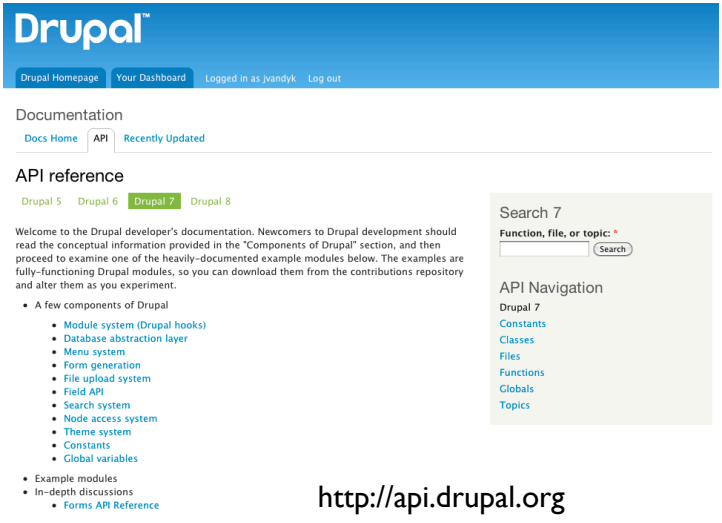
Part of the conversation with your client!

Nobody says "let's create a new Node Bundle and call it Bug Report".

Before Building a Site

- What content types and fields?
- How will the data get in? (Forms)
- How will the data get out? (Views)
- Who may create/view/delete data? (Permissions)
- What additional functionality? (Modules)
- How can we make it pretty? (Theming)

Questions to map client needs to Drupal concepts.



http://api.drupal.org

Your home for API-related searches.

http://drupal.org/project/examples

Simple implementations of APIs. You can base your own work on these examples.

Bonus!

Two tools I use all the time.

Defeating the White Screen of Death

Define PHP error log in php.ini:

```
error_log = /var/log/php/error.log
```


I close with two essential tools. First, don't be frightened by the WSOD.

Watch the error log to find out what's going on:

```
tail -f /var/log/php/error.log

[07-Sep-2011 16:25:02] PHP Fatal error:
Call to undefined function int() in
bgstat.module on line 41
```


PHP will always tell you what went wrong in its log. If you've defined a PHP error log in php.ini, you can watch it in real time with tail -f.

Where is the code?

Quickly search the codebase for a function:

```
egrep -rn searchstring .
```


Second, don't be overwhelmed with the codebase. egrep and pipes can help you find where functions live and where things happen. Example: where are mail-related variables set?

```
egrep -rn mail . | grep variable_set

./includes/install.core.inc:1802:
variable_set('site_mail', $form_state['values']
['site_mail']);

./includes/install.core.inc:1813:
variable_set('update_notify_emails', array($form_state
['values']['account']['mail']));

./modules/openid/openid.test:323:
variable_set('user_email_verification', TRUE);
...
```


There we go. The codebase is like a rich porridge, waiting for you to sift through and find the tasty lumps.

Thanks!

Hopefully these help create a big picture
of how Drupal works. This only scratches the surface.

Now get your debugger
working and explore Drupal core on your own!