# Search Module Demystified

Steven Wittens

October 2005
DrupalCon, Amsterdam

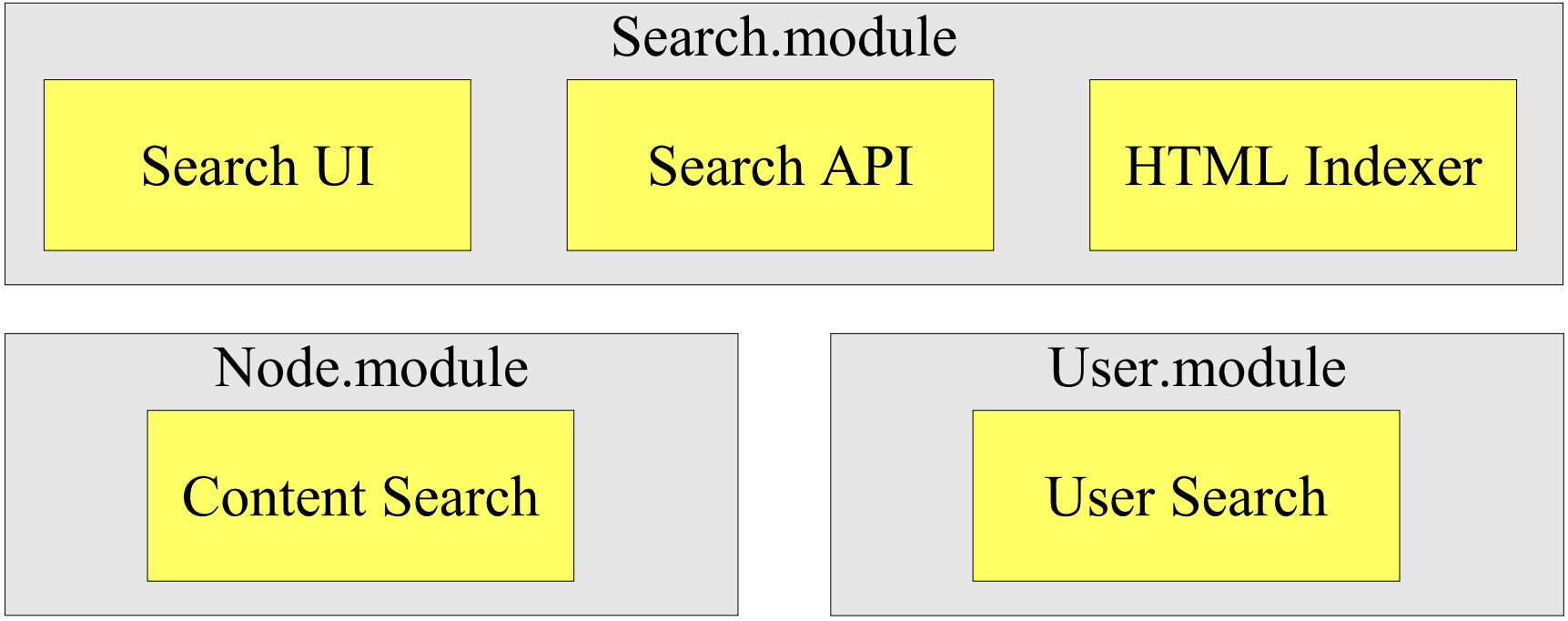
# What I'll be talking about

- Search API/Hooks structure (4.7)
- HTML Indexer
- Searching Drupal.org

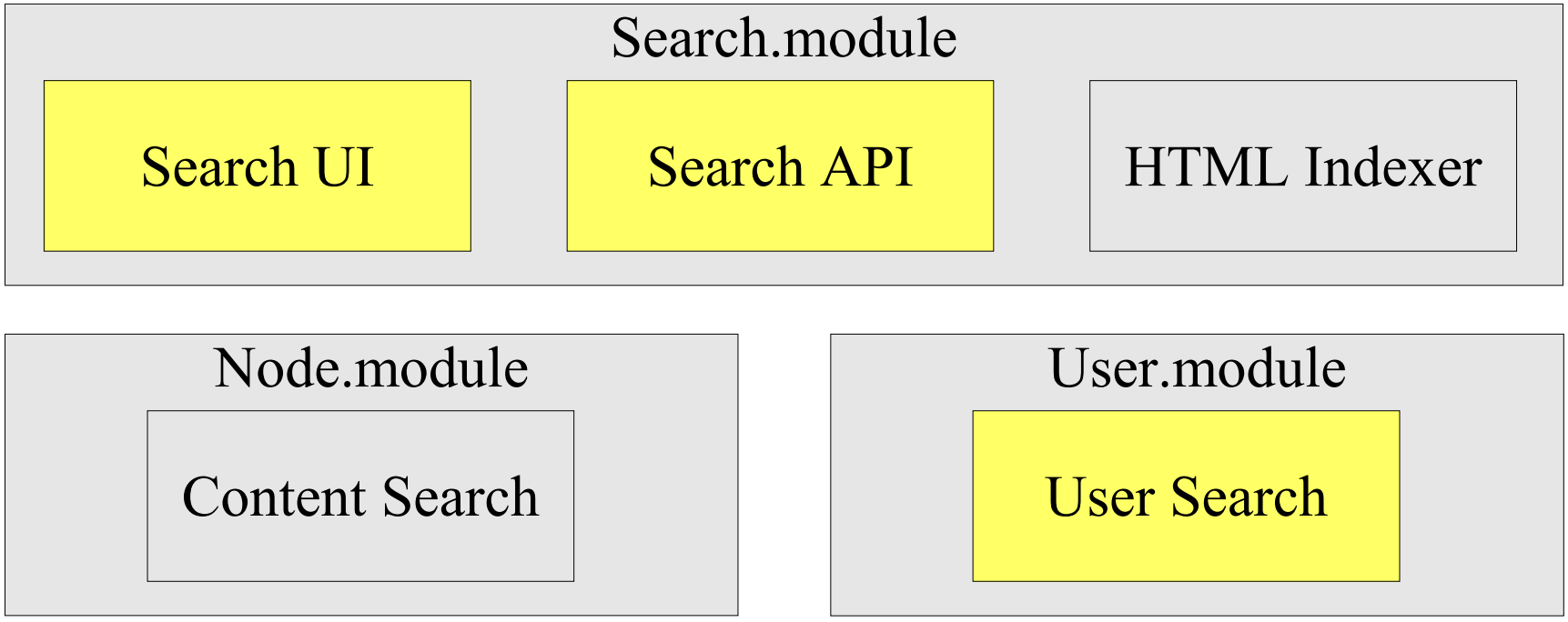
*Break open the big scary module with the gigantic tables*

# Overview

- Several layers of hooks
- Evolved out of pre-4.6 search

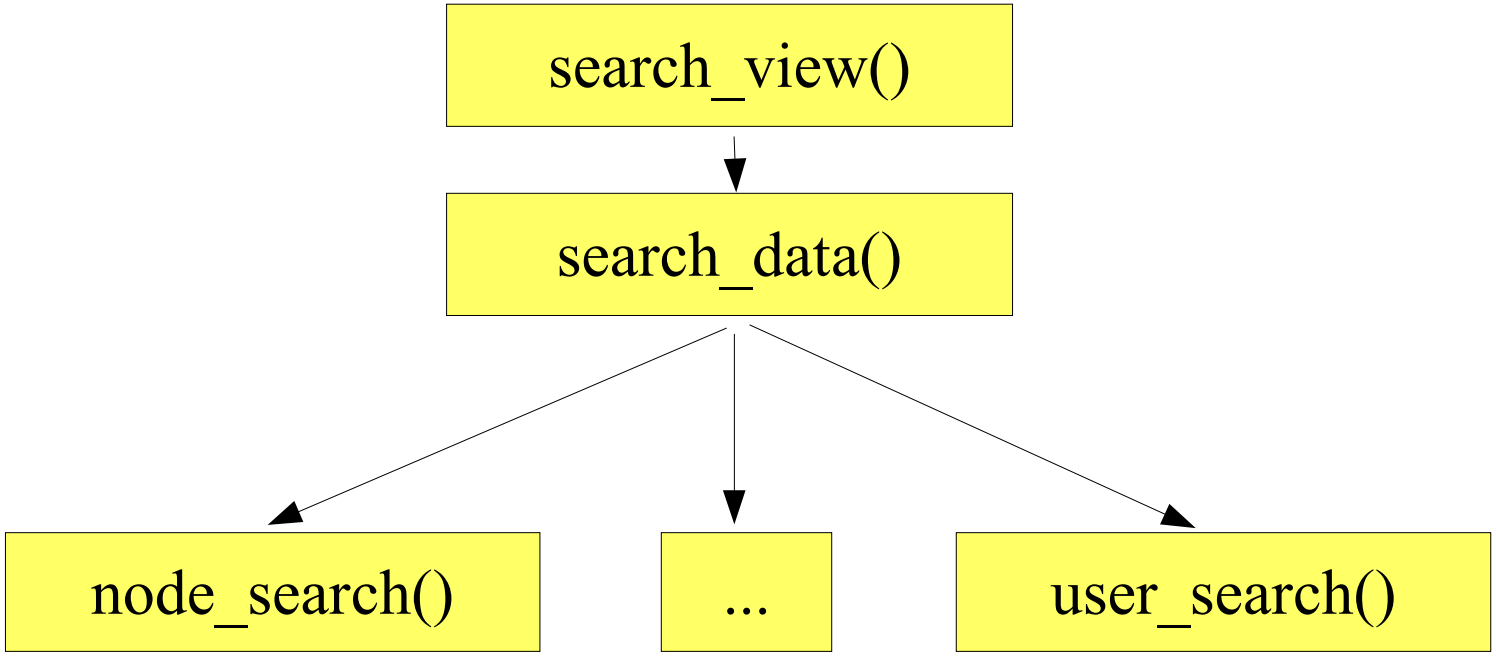


# Overview


Search.module
Search UI
Search API
HTML Indexer
Node.module
Content Search
User.module
User Search

# Search.module

- Gateway to searching, invokes `hook_search()`



# Hook_search()

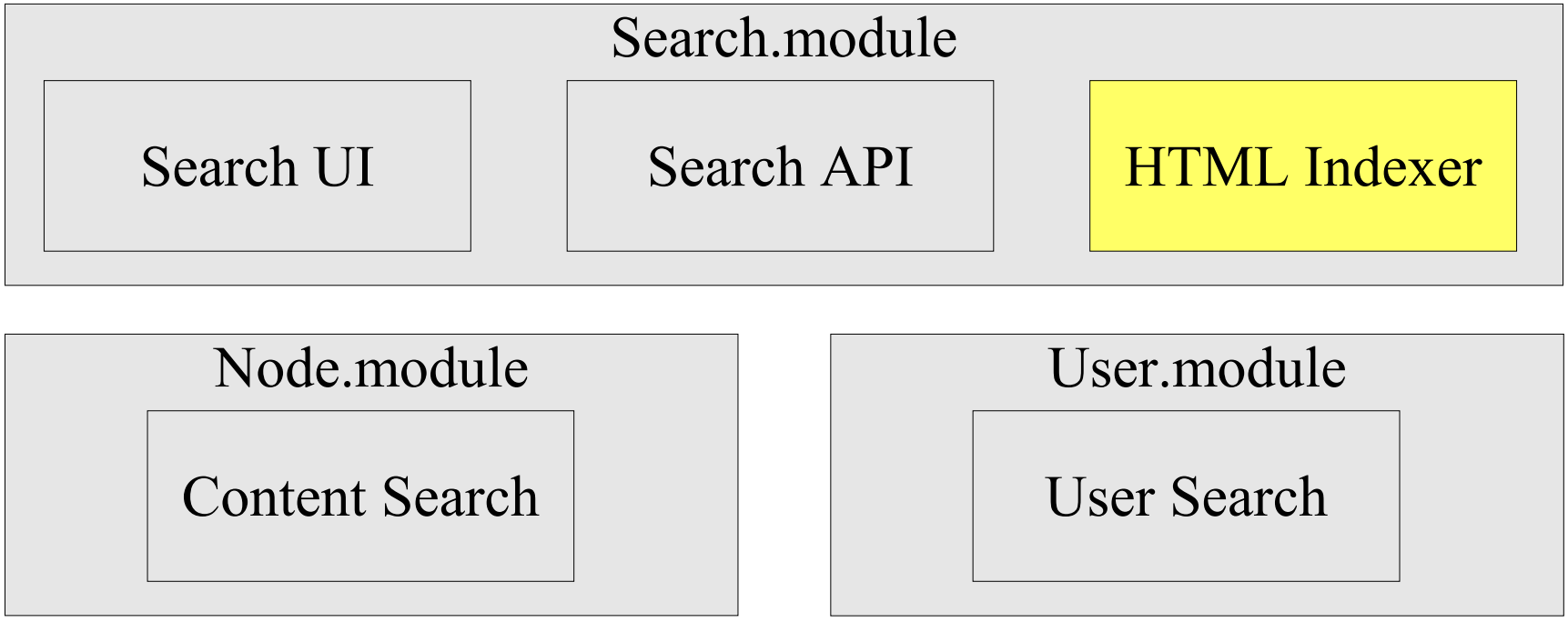
**Multifunctional hook with an operation ($op) parameter.**

- Tab on `/search/`*`modulename`*
- `search_form()` extensible through `$op='form'`
- Clean permalink for each query (HTTP GET): `/search/`*`modulename/keywords`*
- Returns array of results, with various named fields (title, snippet, date, type, ...)
- Results themed with `theme_search_item()`

# Advantages

- Consistent look and theming of results
- `search_data()` can be invoked by anyone (e.g. Do a content search on 404, based on URL)
- Lets you focus on fetching the data itself: `user_search()` is 17 lines long

# Overview


Search.module
Search UI
Search API
HTML Indexer
Node.module
Content Search
User.module
User Search

# HTML Indexer

- System for efficiently searching chunks of HTML (items) with an id and type (e.g. node 42)
- Indexed with `hook_update_index()` on cron
- Can run complicated queries (and/or, phrases, negatives), like popular search engines

> Zaphod Ford OR Arthur "Paranoid android" -radio

- Returns results ranked by *relevancy*
- Two-pass extensible query in SQL using temporary tables

# Preprocessing

- Goal: split text into words (tokenization)
- Applied to index data and search keywords
- Rules for dealing with acronyms, URLs, numerical data (Unicode-aware)
- Language-specific preprocessing through `hook_search_preprocess(`*`$text`*`)`:

| | | | |
|---|---|---|---|
| resumé | → | resume | (accent removal) |
| blogging | → | blog | (stemming) |
| blogs | → | blog | (stemming) |
| 青い猫 | → | 青い　猫 | (word splitting) |

# Inverted Index (1st pass)

- Use HTML tags to find important words
- Sum scores for multiple keywords after dividing by their total count across the site
- Automatically separates meaningful words from noise words
- Results in a relevancy score, fractional number 0..1 (more is better)

# 1st pass: Inverted index

- `search_index` table stores all the unique words for each item, along with a score per word
- Score based on frequency and HTML tags

**Drupal is a content management system. Drupal is coded in PHP.**

Drupal = 2

# 1st pass: Inverted index

- `search_index` table stores all the unique words for each item, along with a score per word
- Score based on frequency and HTML tags

**Drupal is a content management system. Drupal is coded in PHP.**

Content = 5 * 1

# 1st pass: Inverted index

- `search_index` table stores all the unique words for each item, along with a score
- Score based on frequency and HTML tags

**Drupal is a content management system. Drupal is coded in PHP.**

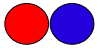
- Scores summed per word and saved in `search_total`. Higher total = more noisy

# Searching & Ranking: TF/IDF

- First pass: searching the inverted index = simple AND query on the positive keywords + relevancy ranking.
- Per keyword: score in an item / sum of all scores across site = relevancy for a keyword
  e.g. Drupal = 7, total(Drupal) = 1000
  Installation = 3, total(Installation) = 10
  → Relevancy = 7/1000 + 3/10 = 0.307
- Rare words score better than common words

# HTML Links

- Recognizes links to nodes on the current site, both relative and absolute
- Can resolve URL aliases
- Adds the link's caption to the target node rather than the current item being indexed
- If the caption is just the URL, use the target's title instead



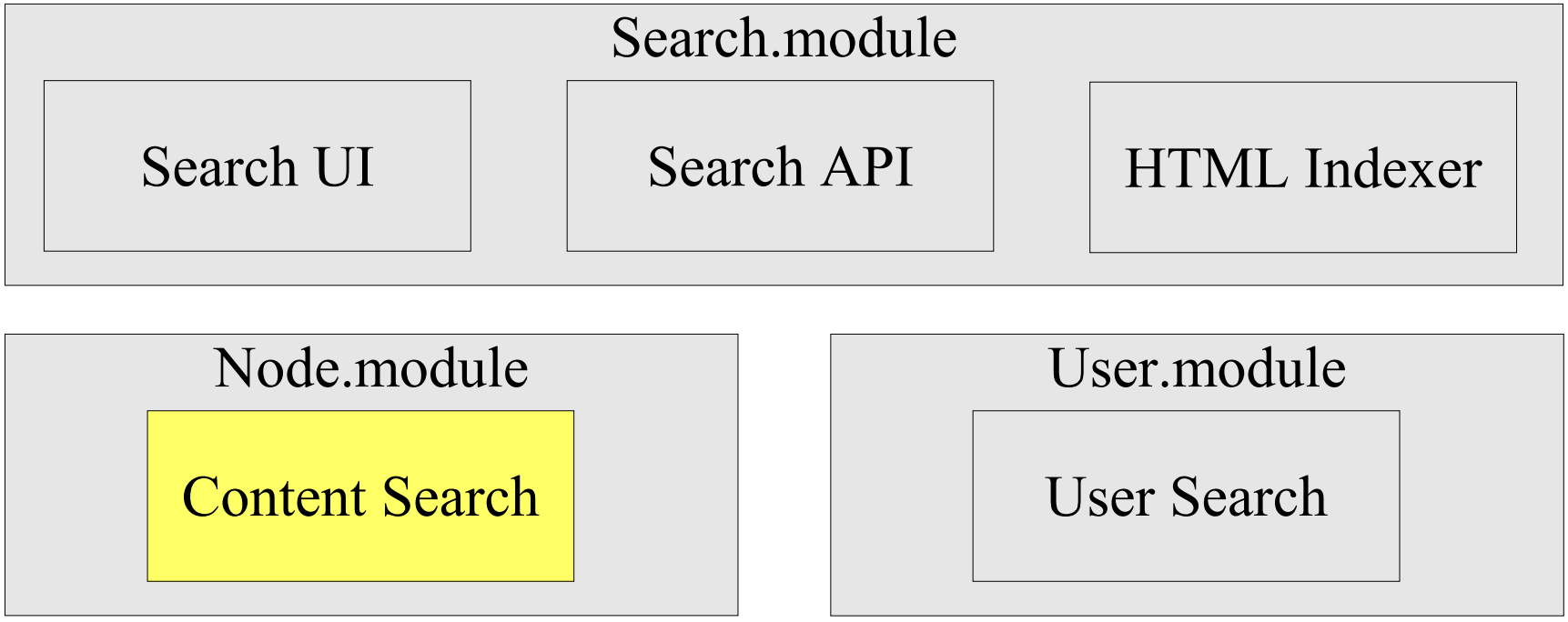
# 2nd pass: Full Dataset

- `search_dataset` table stores the literal (preprocessed) data
- Do literal string matching to satisfy phrases, AND/OR mixing, negatives
- Without the first pass, this operation would be very expensive (full table scan)

# Why not use MySQL FULLTEXT()?

- DB-specific (PgSQL tsearch2 is not standard)
- Fulltext is a special type of database index on one or more columns of a table
- Nodes, comments, ... would need to be aggregated into a single table anyway
- Possible for the future, but would not get rid of cron-based indexing
- Would not understand HTML nor track links

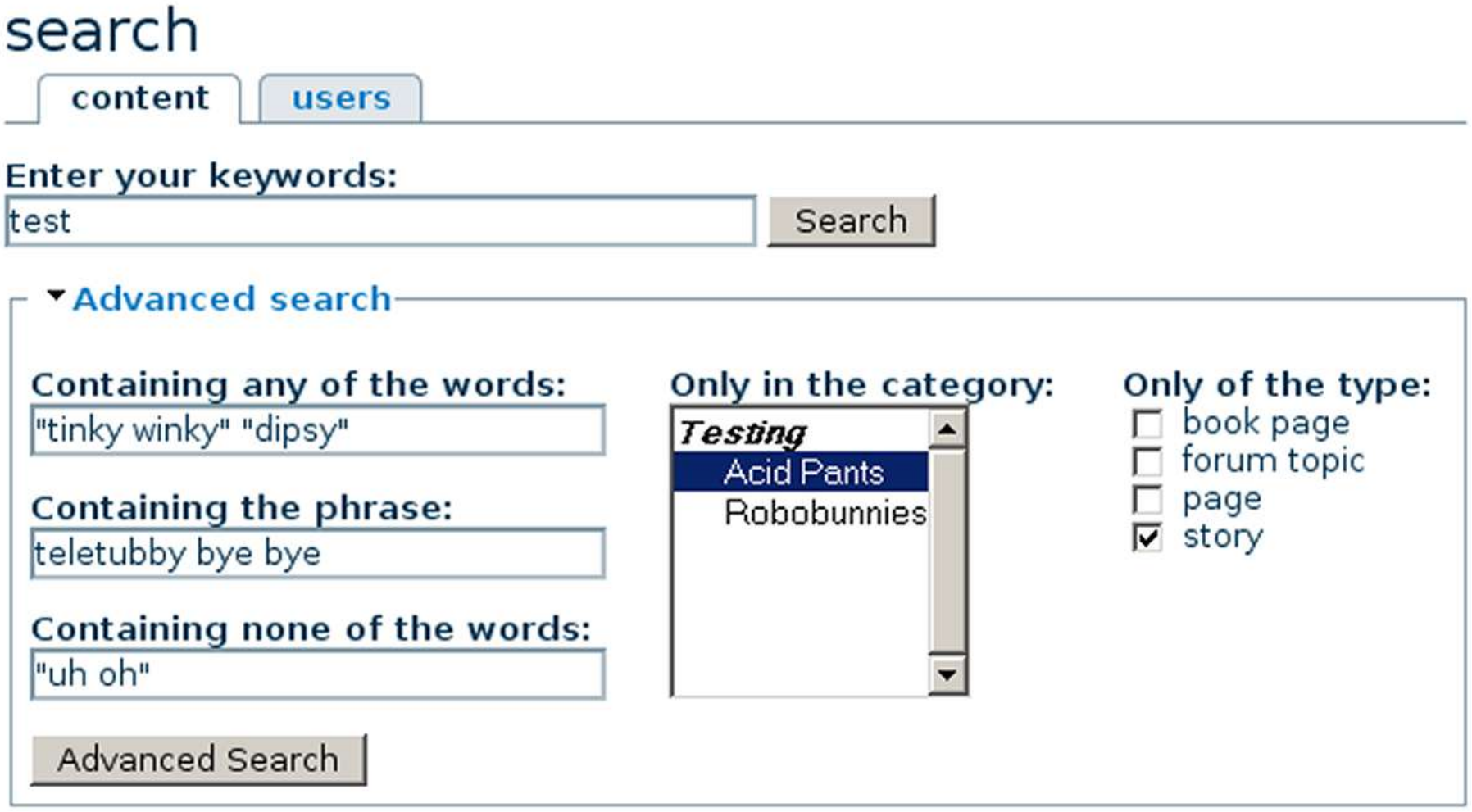
# Overview


Search.module
Search UI
Search API
HTML Indexer
Node.module
Content Search
User.module
User Search

# Content: node_search()

- Uses the HTML indexer to index entire nodes (with comments)
- Provides extra conditions (node type, taxonomy term, ...) with a Google-like syntax (type:blog)
- Extends search ranking with extra factors which can be weighted by the admin:
  relevancy * 5 + freshness * 3 + comment count
- Indexed data can be further extended through `nodeapi('update index')` (e.g. File attachment contents)

# Content Search




test type:forum,story category:1 "tinky winky" OR "dipsy" -"uh oh" "teletubby bye bye"

# Index entire node

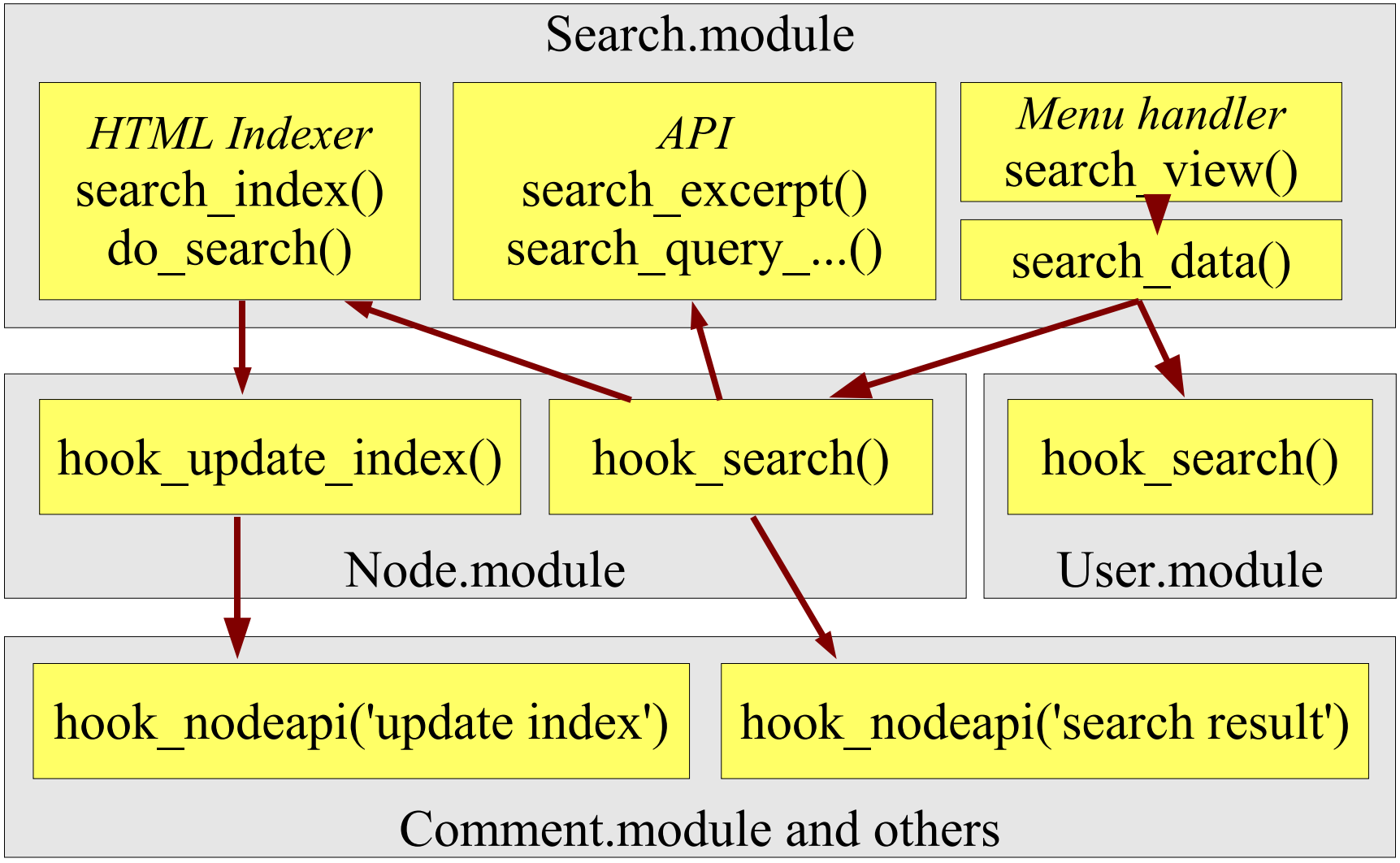
- Nodeapi('update index') used to add extra HTML content, using tags as well

```
function comment_nodeapi(&$node, $op, $arg = 0) {
  switch ($op) {
    case 'update index':
      $text = '';
      $comments = db_query('SELECT subject, comment, format
                            FROM {comments}
                            WHERE nid = %d AND status = %d'
                            $node->nid, COMMENT_PUBLISHED);
      while ($comment = db_fetch_object($comments)) {
        $text .= '<h2>'. check_plain($comment->subject) .'</h2>'.
                  check_markup($comment->comment, $comment->format, FALSE);
      }
      return $text;
  }
}
```

# Content Search Results

- Highlighted snippet with `search_excerpt()`
- Nodeapi('search result') used to add extra information (e.g. Comment count)
- Node type, author information, creation date, ...

# Big picture


Search.module
*HTML Indexer*
search_index()
do_search()
*API*
search_excerpt()
search_query_...()
*Menu handler*
search_view()
search_data()
hook_update_index()
hook_search()
Node.module
hook_search()
User.module
hook_nodeapi('update index')
hook_nodeapi('search result')
Comment.module and others

# Drupal.org search

- Large database (30000 nodes, 60000 comments)
- Low signal-to-noise ratio, lots of repeat
- Means: even with AND search, too many results
- Almost no-one goes to 2nd page of results

→ *Ranking*, not matching, is most essential factor

- Stemming reduces index size by 30%

# What was wrong with 4.6 search?

- HTML tag recognition got confused with unclosed HTML tags:

Foo bar <**b**>foo bar<**b**> foo bar

- Wildcards destroyed performance of database indices (use stemming instead)
- No advanced matching
- Coefficients not as optimized

# Why not trip_search.module?

- Queries original tables directly, does not aggregate entire nodes
- Sorts by date, only good if there is high signal-to-noise
- Does full table scans every time

*Good for small sites with lots of relevant content*

# Why not Google?

- Google only sees public content
- Google does not understand Drupal node structure (e.g. Taxonomy)
- Google's free API is limited in # of queries

# Issue: search as a module?

- Search is becoming more essential, but is still an optional module
- Useful API mixed with front-end
- But, API (indexer) needs to be a module (cron), like taxonomy.module
- Node search is located in node.module, adds a large chunk of non-essential code to a required module

# UI improvements

- Examine search patterns for end users
- Determine requirements for module developers
- What is needed?

*4.7 update is mostly algorithmic*

# Conclusion

- If you remember 50% of all that, great
- Search is very extensible, so get in there and play around
- Slides / more info (neato 404 search)
  http://acko.net/amsterdam
- Pre-patched HEAD
  http://acko.net/dumpx/searchpatched.zip