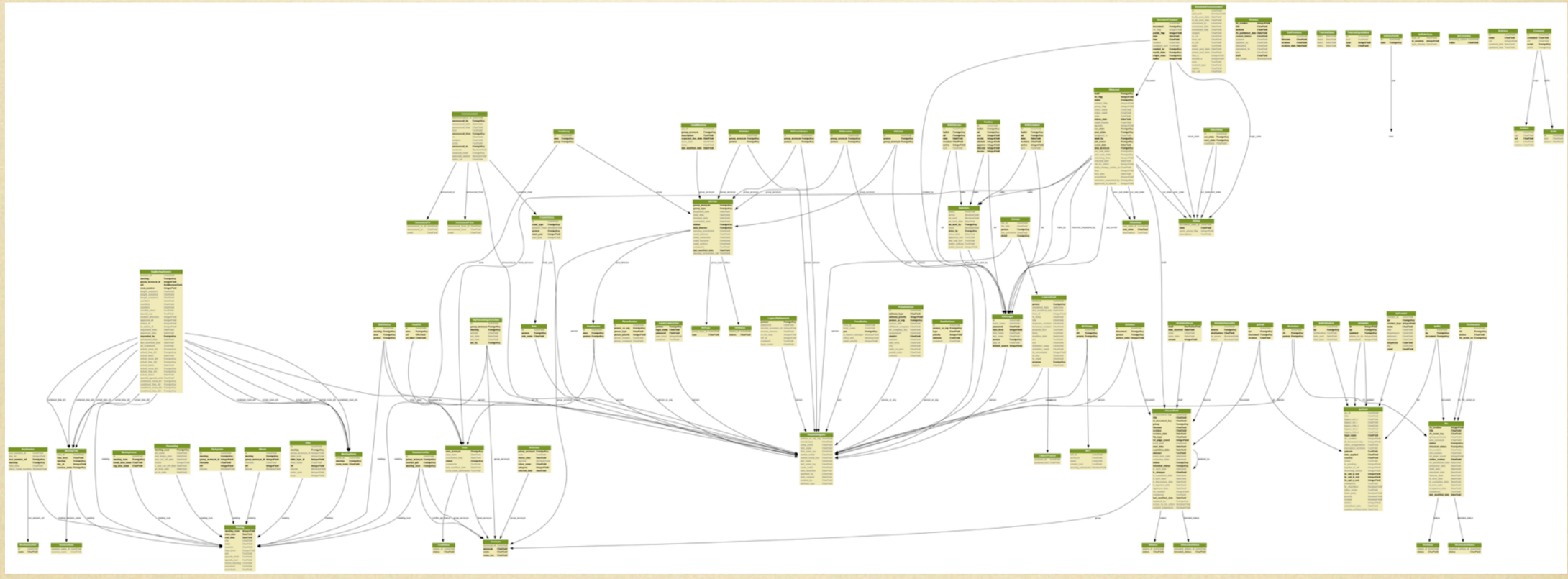
The IETF Database

Once there was a young and inexperienced database ...

- 12 years ago, it was felt that there was a major problem with lack of transparency with regards to the status of documents, IESG comments and discusses, and too many dropped balls.
- To do something about this, a database was set up to track documents, last calls, discusses, and more. This helped, this was good.
- The database grew, and expanded, and grew and expanded, to cover more and more ...

… and eventually things became rather complex.



5 years ago, we discovered serious sql injection attack holes, and decided to to a re-take.

- March 2007: Bill Fenner discovers SQL injection attack holes in the datatracker. Efforts to get the secretariat to understand the seriousness of this fail. Bill and I decide to do a full rewrite of the public tracker as a skunkworks project.
- First svn commit: 19 April 2007.
- Russ and Ray informed 2 May 2007.
- Release of rewritten public tracker 28 June 2007

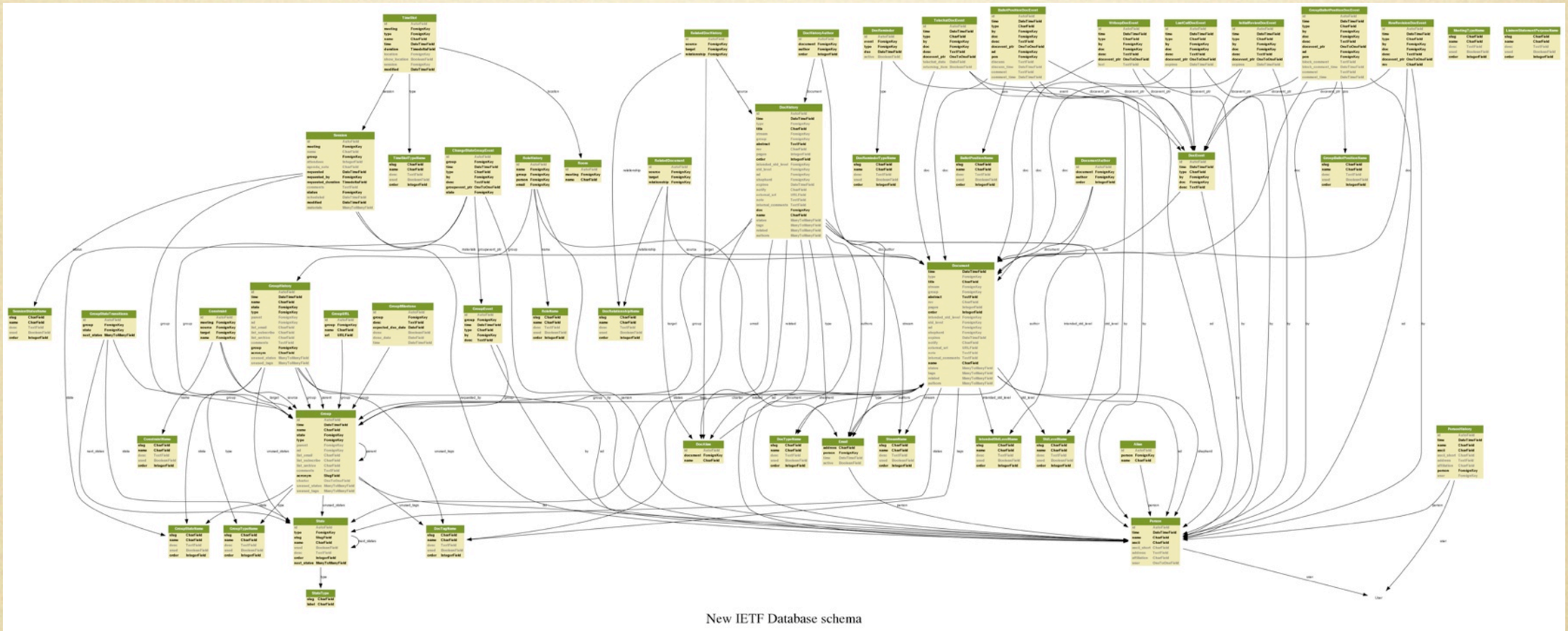
For IETF 70, Russ initiated the code sprints as a way to get more people involved in coding.

- The first code sprints got a few things done, but it took some time to settle into a stable form that worked well.
- Eventually, as we continued to hack on the code, it became more and more apparent that the biggest obstacle to refactoring and good new code was the database schema.
- The schema had ‘just grown’ during 7 years...

A new IETF database schema

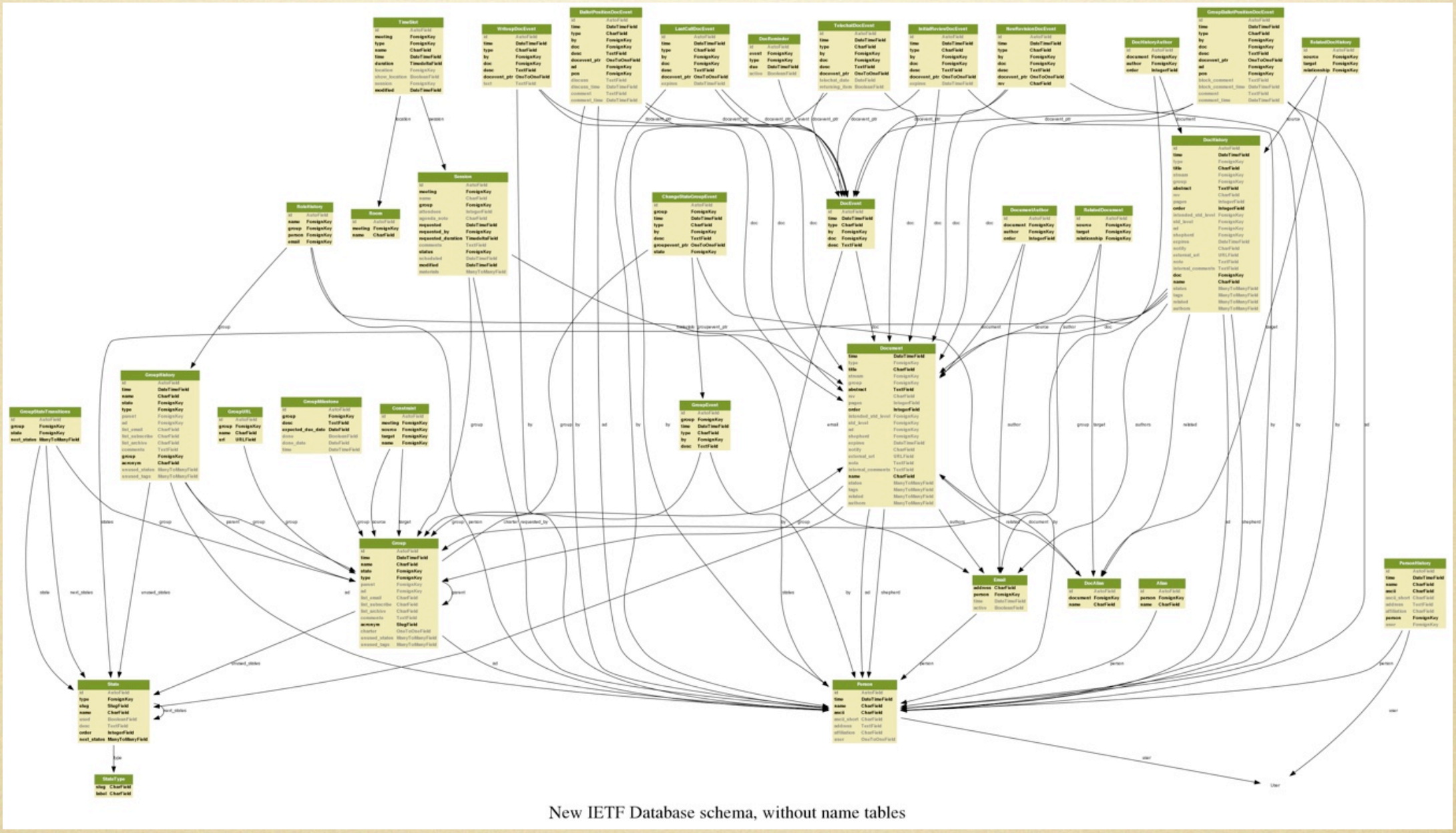
- During the spring of 2008 I started thinking about a new schema.
- During summer, I visited the secretariat in CA for a week, working intensively on the design, and coordinating with them to make sure the design would work also from their viewpoint.
- During autumn and winter 2008, I got review feedback and tweaked the schema as needed.

The new database schema (1/3)



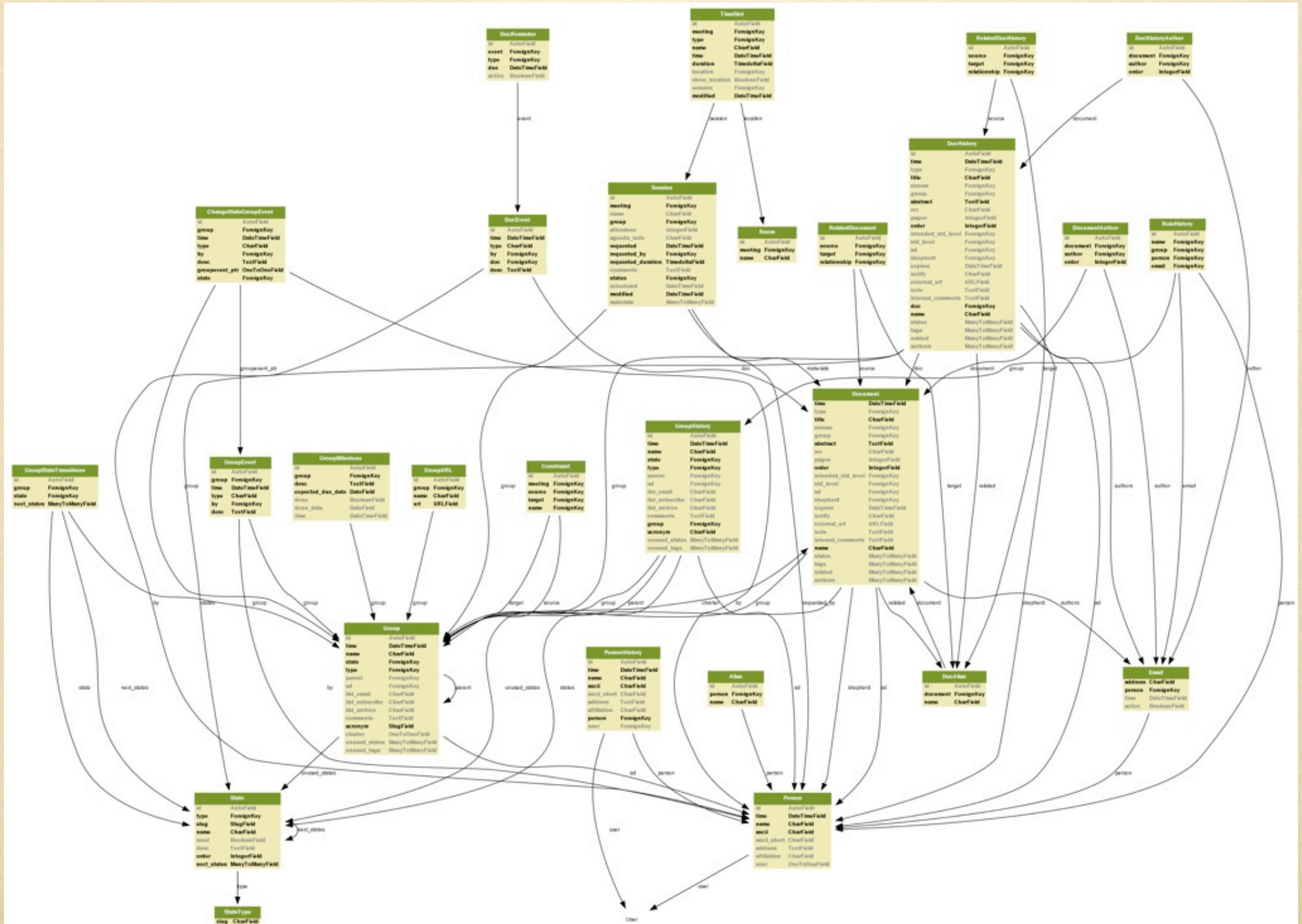
New IETF Database schema

The new database schema (2/3)



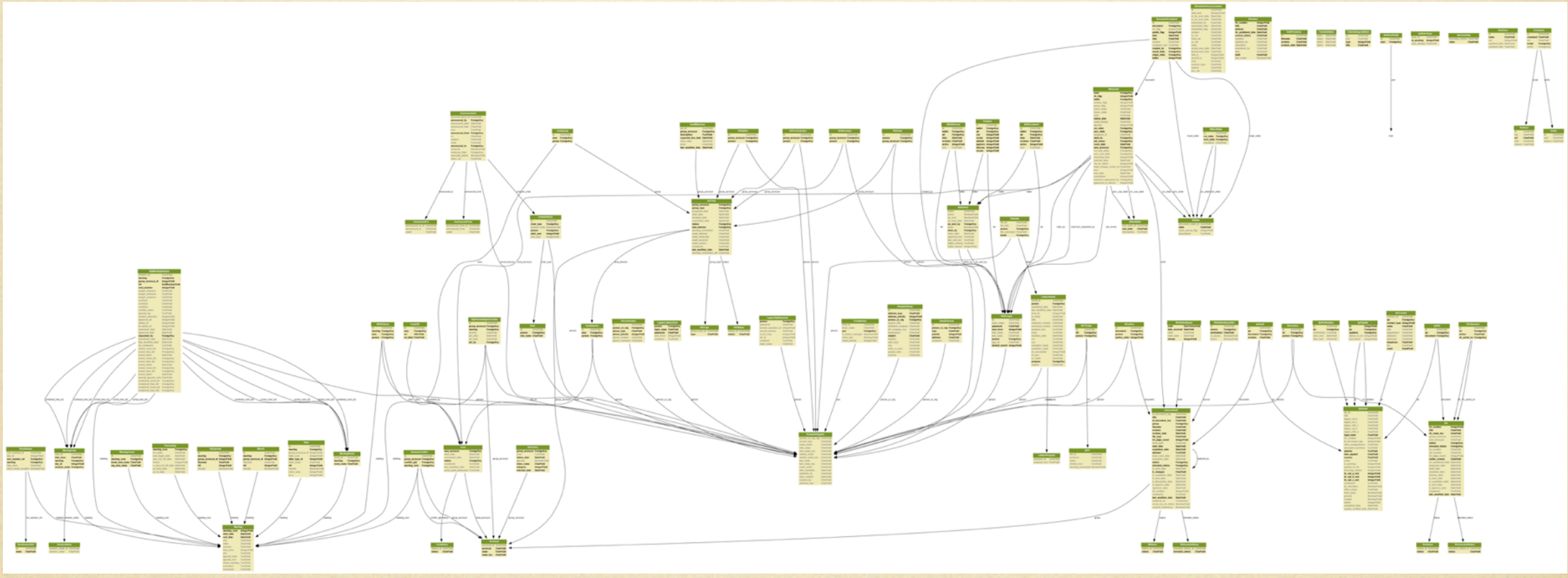
New IETF Database schema, without name tables

The new database schema (3/3)



New IETF Database schema, without name tables and subevents

For comparison, here's the old schema again.



New features (1/2)

- History tables for central objects: Documents, Groups, Persons.
- Generalized documents, which lets us treat charters, agendas, minutes, and other as documents, with revisions, diffs, states, etc.
- Name tables, indexed by slugs, to make code more readable and less fragile

New features (2/2)

- Generalized states -- multiple states per document is possible (for instance, draft states for wg-state, iesg-state, iana-state, rfc-ed-state, etc.), and different document types can have different state sets.
- Unified permission handling through Roles

Comparison

- Old schema: 86 tables
- New schema: 27 tables (not counting name lookup tables and subclasses)

So, how do we go from here to there?

Build a completely new system, independent of the old one, and deploy when done?

This runs into 'Second system effect[1]' problems: Feature creep, bloat, and loosing all the small but important exceptions encapsulated in the current system's code. It would also take longer before we could deploy and start to get benefits.

[1]) Fred Brooks: *The Mythical Man-Month*

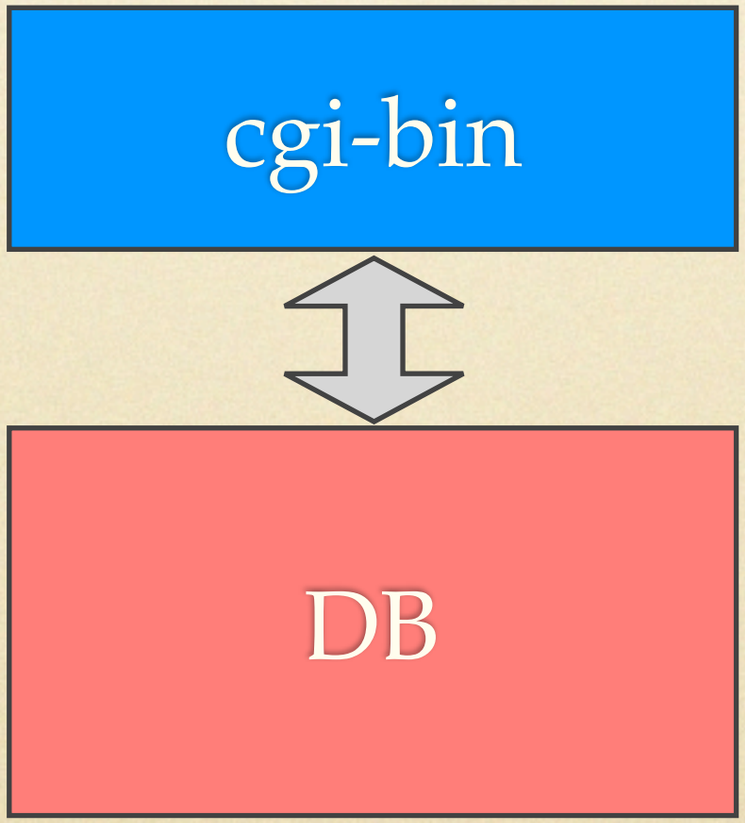
So, how do we go from here to there?

Divide and conquer? Keep as much as we can of the current code, only change the database?

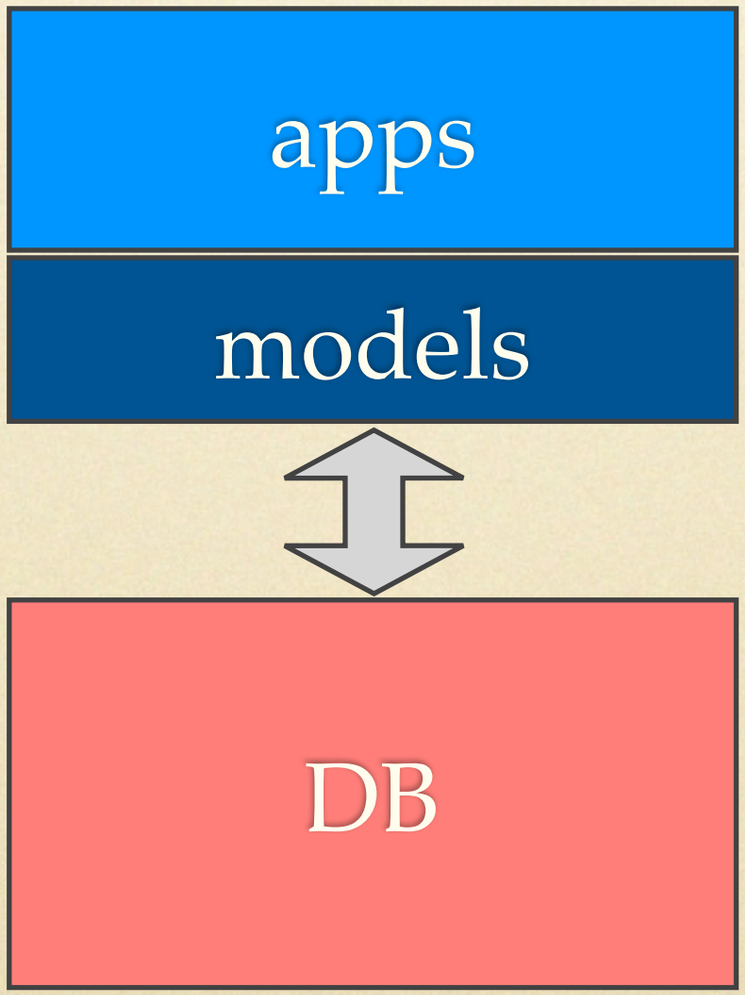
This is attractive, if it can be done, but can it?

It turns out that the framework we used when we did the rewrite of the public datatracker, 'Django', has developed greatly since we chose it, and now provides model class inheritance in 3 flavours, one of which is proxy models.

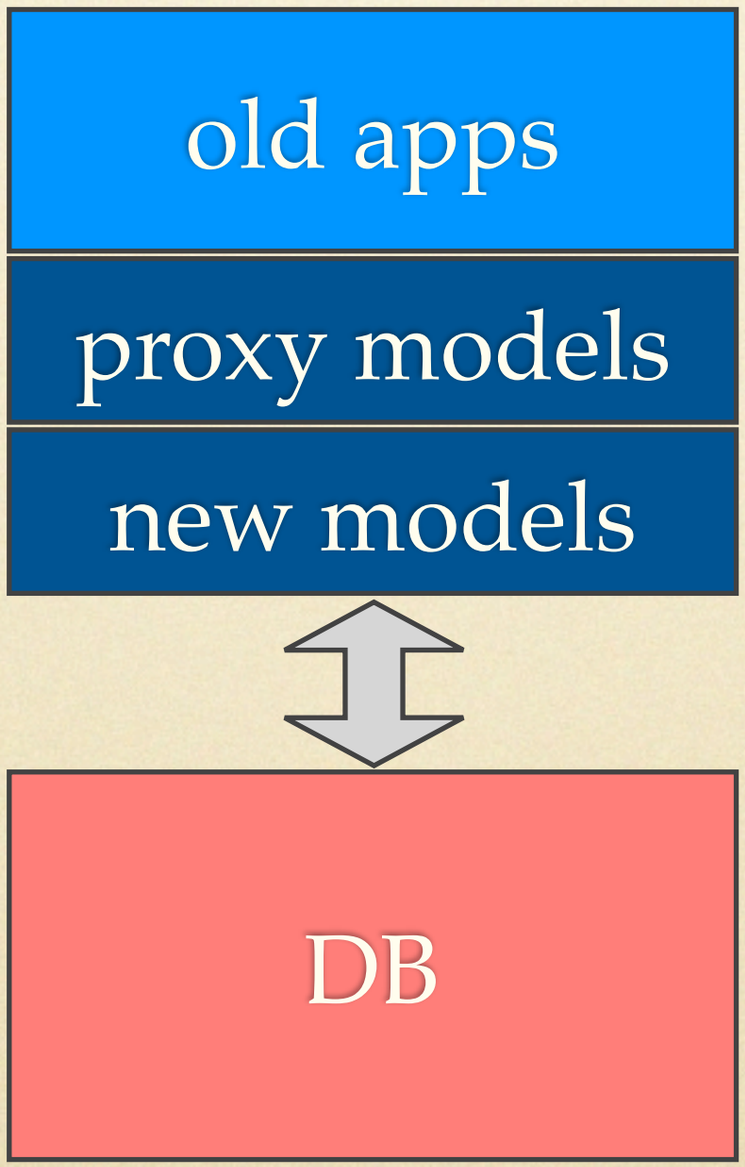
Old datatracker



Django datatracker



Django datatracker
New schema, old apps



Testing the waters

- This idea requires two parts:
  - Proxy models, showing the old interface code a semblance of the old models
  - Conversion scripts, to convert the data in the old tables to the new schema
- During the first part of 2010, I worked on both, but after some time, it became very clear that the job was too big for a part-time effort, if it was to be completed within reasonable time.

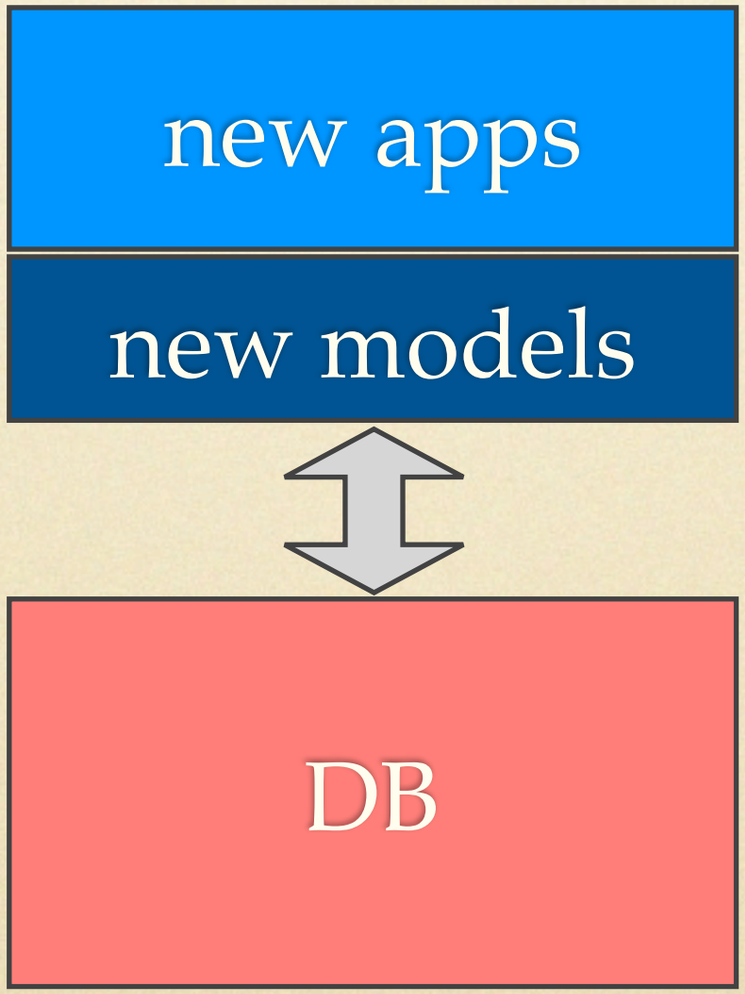
- During the second half of 2010, we put out an RFP for the work of fleshing out the proxy models and writing conversion scripts to convert the data in the old tables to the new schema, and clean it up in the process.
- A small Danish firm of software consultants, IOLA, won the contract, and started the work during autumn 2010.

Looking forwards

- There's one more step we need to take to get to where we want to be; which can be done a piece at a time: Rewriting the old UI code to use the new models directly. That work needs to be done in order to achieve a clean codebase.

- New apps should require much less interface and data massaging code, as the it won't have to patch together data as many disparate and inconsistent tables as with the old system.

Django datatracker
New schema, new apps



New apps already in the pipeline, written for the new schema

- Document tracking and notification
- WG Charter management (charters as documents with diffs and states, discusses, IESG balloting, process support
- Automated document state updates from IANA and the RFC-Editor

Other things we can do now: JSON Export

- JSON export of filtered database content:
  `dt.ietf.org/json/doc/document/?<filter_args>`


```
{
   "doc.document": {
      "draft-ietf-idr-bgp-issues": {
         "time": "2012-03-27 10:04:44",
         "type": "draft",
         "title": "Issues in Revising BGP-4 (RFC1771 to RFC4271)",
         "stream": "ietf",
         "group": 1041,
         "abstract": "   This document records the issues ...",
         "revision": "06",
         "pages": 170,
         ...

      },
      "draft-ietf-idr-bgp-nh-cost": {
         ...
      }
      ...
   }
}
```

JSON Export

The great thing with this particular idea is that it will make it possible for all you tool writers to pull exactly the data you need for your new brilliant IETF-related app directly from the datatracker in a machine-readable format.

This makes screen-scraping datatracker pages obsolete :-)

It also makes your new apps use the new schema, which means easily integrable if wanted

Question and Answer Session

Thank You