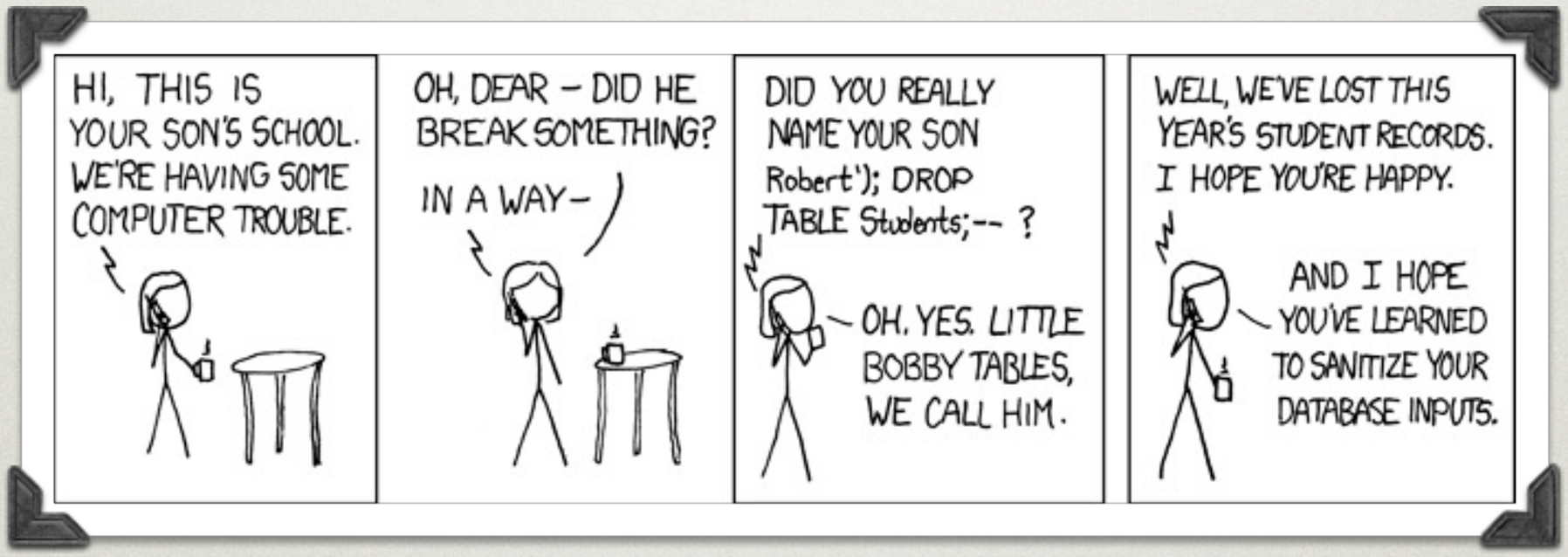
Internet Programming in Python - Week 9

Databases



http://xkcd.com/327/

Brian Dorsey
brian@dorseys.org

ANNOUNCEMENTS

System Development with Python

http://staff.washington.edu/jon/python-course/course3.html

Questions and review (20)

0:10

some thoughts from the assignments

Assignment

turn in here: `http://is.gd/uwipip_week8`

- port your week 7 or week 8 application to a different database backend

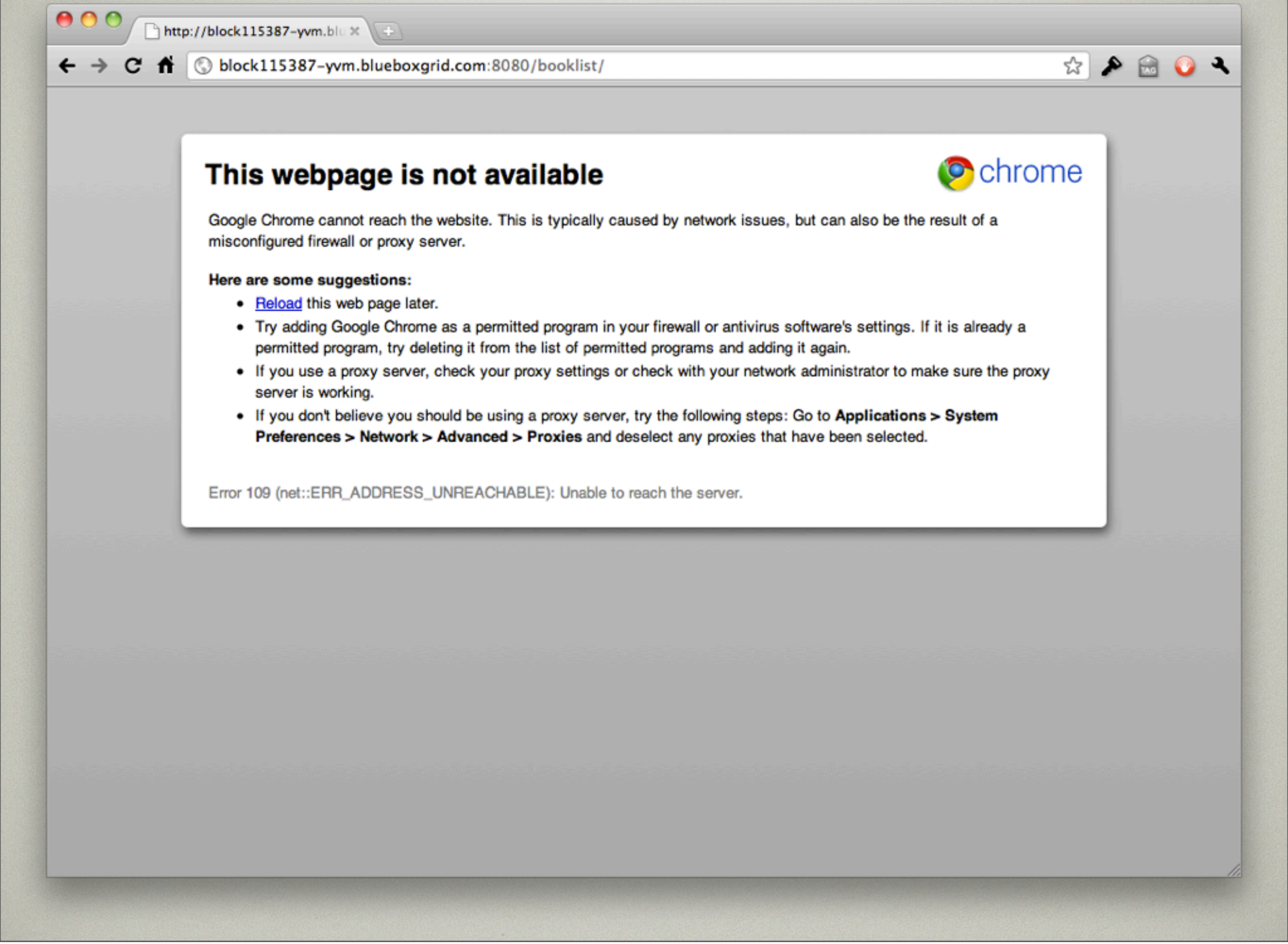
  next:

- A: Continue hacking on your week 8 application, get it setup on semi-permanent hosting, add polish, etc.

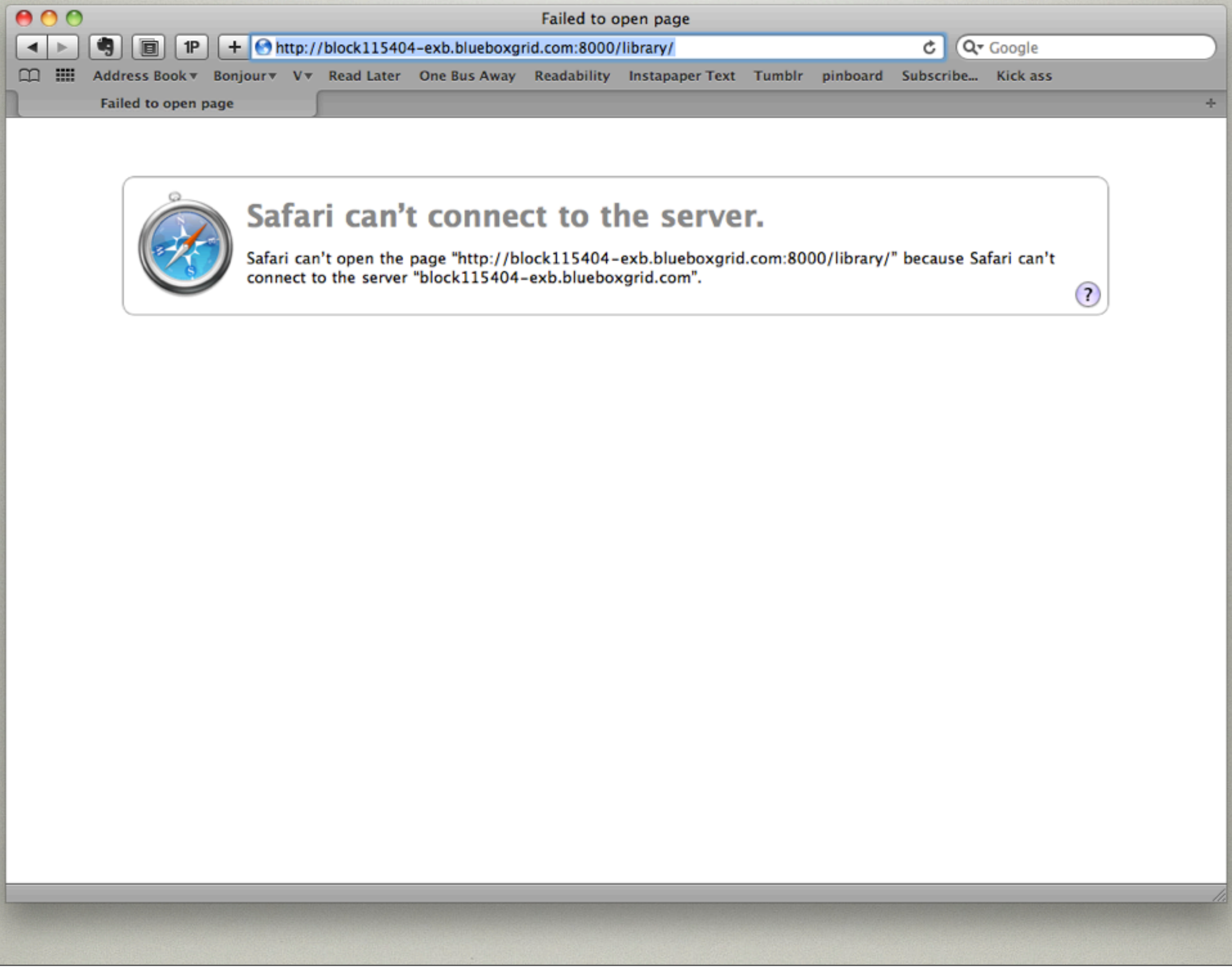
- B: Install redis and redis-py.
  Experiment with the list, set, sorted set and hash data types. How are they like Python datatypes? JSON? How are they different?
  Make two scripts which cooperate via the DB.
  (see week 9 extra reading for ideas)
  (the server is linux/mac only - use a VM if needed)

how’d it go?

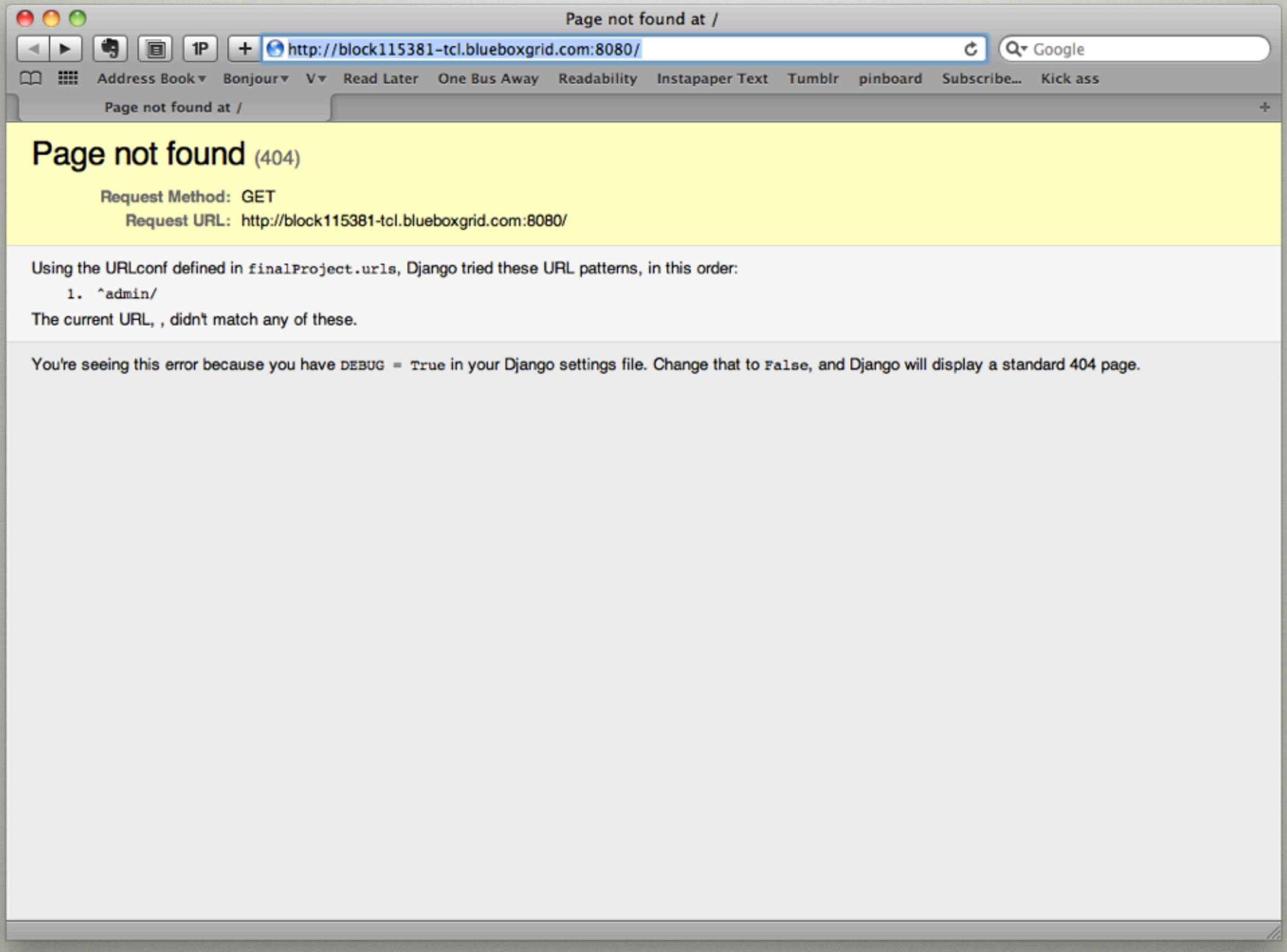
some of the assignments



this page lies, lies, lies.



this one seems to be accurate, though


Page not found at /
http://block115381-tcl.blueboxgrid.com:8080/
Google
Address Book
Bonjour
V
Read Later
One Bus Away
Readability
Instapaper Text
Tumblr
pinboard
Subscribe...
Kick ass
Page not found at /
Page not found (404)
Request Method: GET
Request URL: http://block115381-tcl.blueboxgrid.com:8080/
Using the URLconf defined in finalProject.urls, Django tried these URL patterns, in this order:
1. ^admin/
The current URL, , didn't match any of these.
You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard 404 page.

BuildNet

djingodjango.com/buildnet/

BuildNet

Parts

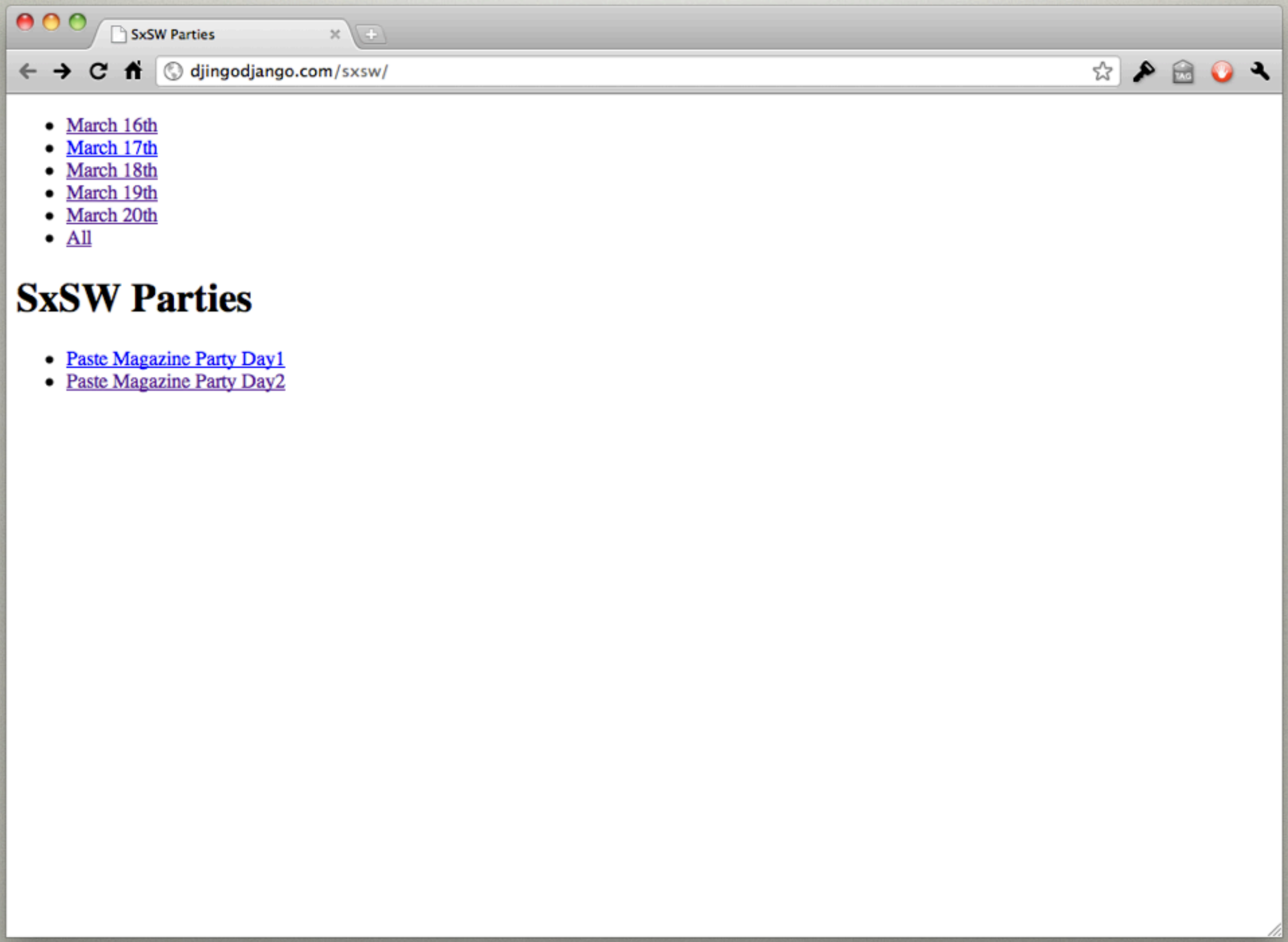
- [α] Bar [β]
- [α] Baz [β]
- [α] Foo [β]

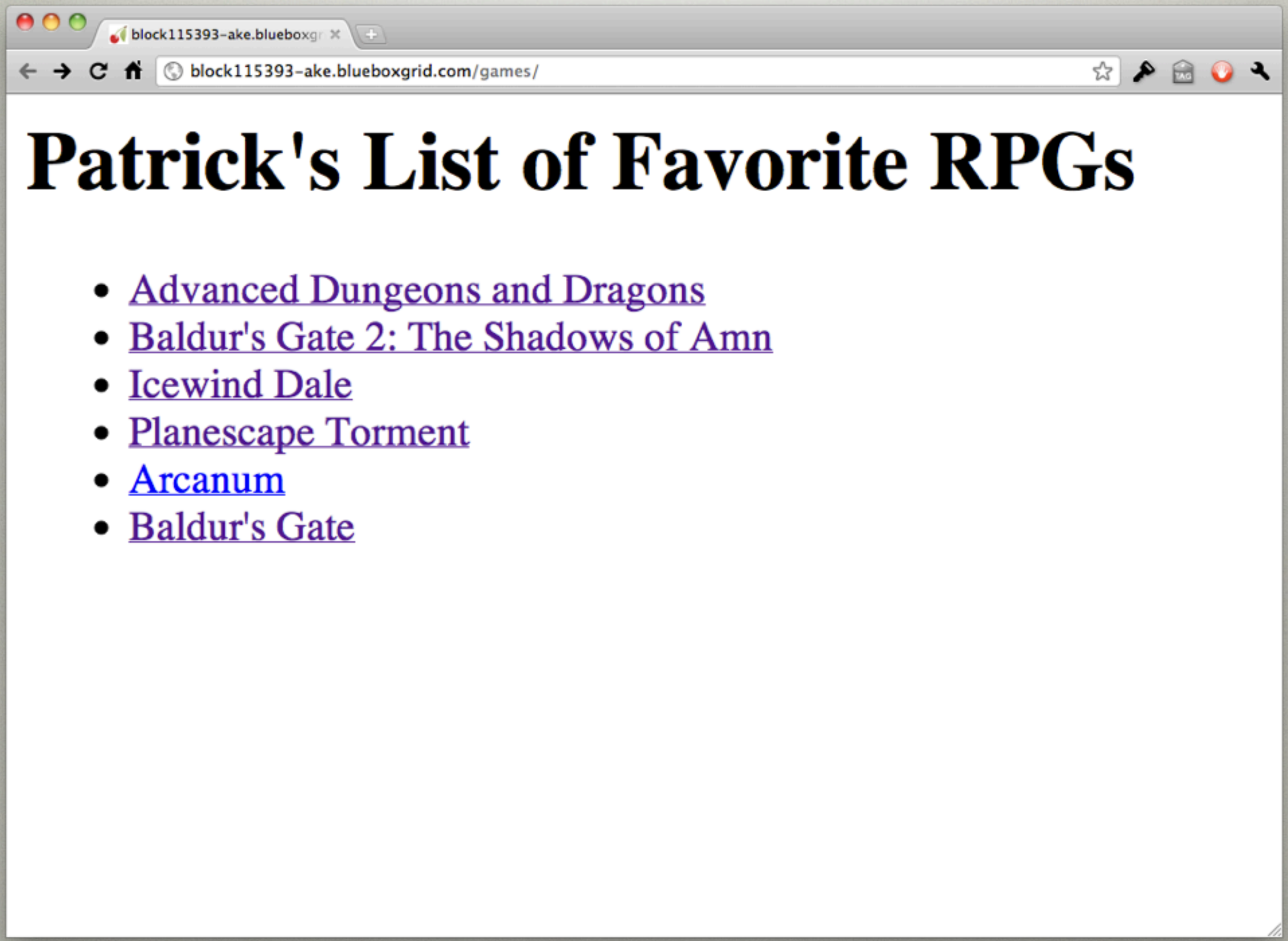
Station Alpha

- Foo
- Bar
- Baz
- Bar
- Bar
- Bar
- Foo
- Bar
- Foo
- Bar

Station Beta

- Bar
- Foo
- Baz
- Foo
- Foo
- Baz
- Foo
- Bar


SxSW Parties
djingodjango.com/sxsw/
March 16th
March 17th
March 18th
March 19th
March 20th
All
SxSW Parties
Paste Magazine Party Day1
Paste Magazine Party Day2


block115393-ake.blueboxgr
block115393-ake.blueboxgrid.com/games/
Patrick's List of Favorite RPGs
Advanced Dungeons and Dragons
Baldur's Gate 2: The Shadows of Amn
Icewind Dale
Planescape Torment
Arcanum
Baldur's Gate

My Food Blog

test
March 1, 2011, 4:29 p.m.
tets

My fifth post
March 1, 2011, 4:24 p.m.
Coca Cola can - 330 calories

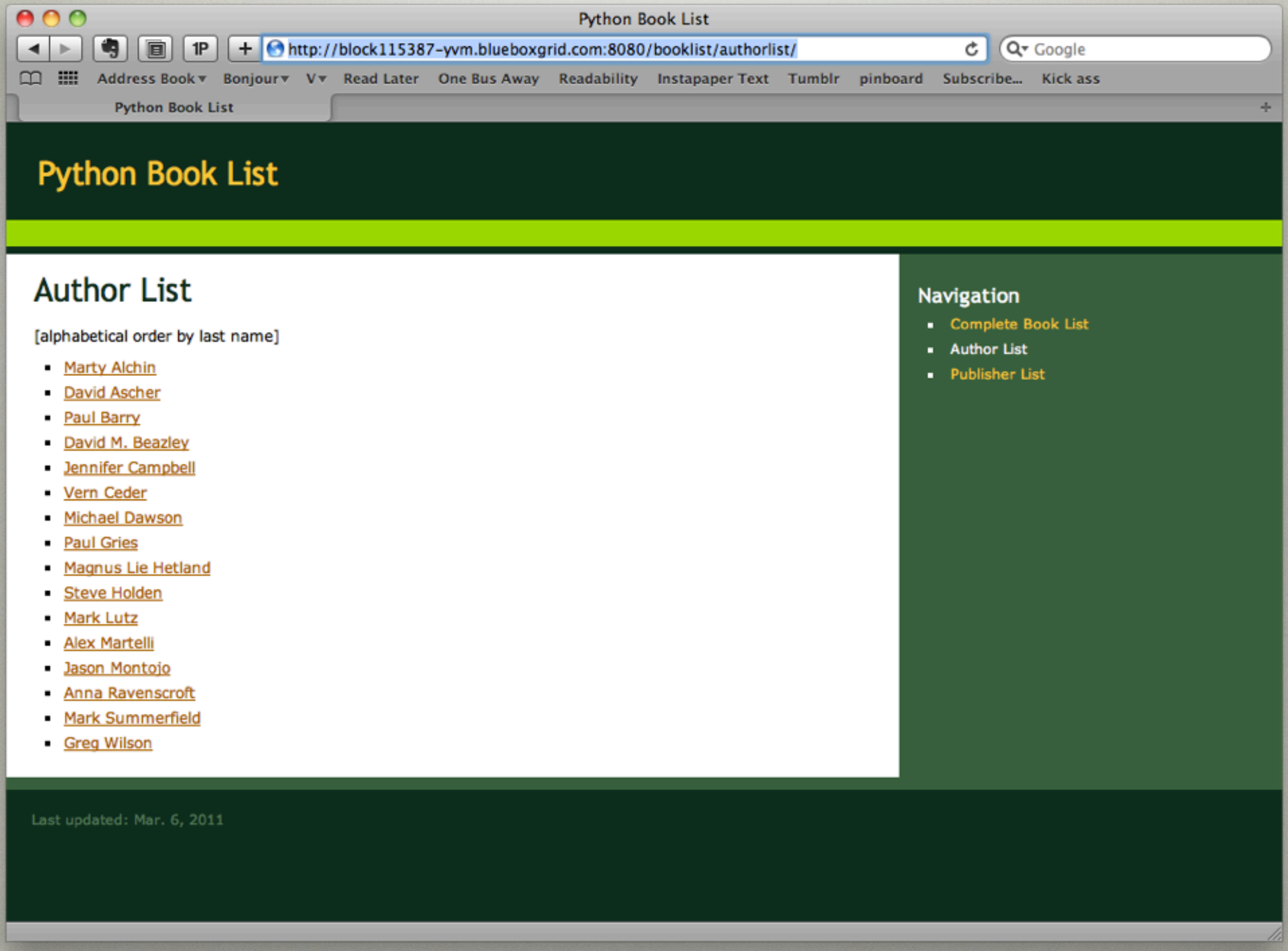
My fourth post
March 1, 2011, 4:19 p.m.
breakfast I had a piece of bread for breakfest - 100 calories lunch Odwalla chocolate bar - 170 calories bean soup - 220 calories 2 glass of water - 0 calories

---

My third post
Feb. 28, 2011, 6:15 p.m.
breakfest 2 scrabble eggs - 200 calories orange juice - 112 calories lunch steak fries - 300 calories

---

My second post
Feb. 28, 2011, 4:45 p.m.
My diet today was breakfast 1 glass of water - 0 calories 1 Sun Chips, multi grain - 210 calories 1 breakfast sandwich - 350 calories Lunch 1 bowl of Chili - 400 calories 1 glass of water - 0 calories


Python Book List
http://block115387-yvm.blueboxgrid.com:8080/booklist/authorlist/
Google
Address Book
Bonjour
V
Read Later
One Bus Away
Readability
Instapaper Text
Tumblr
pinboard
Subscribe...
Kick ass
Python Book List
Python Book List
Author List
[alphabetical order by last name]
Marty Alchin
David Ascher
Paul Barry
David M. Beazley
Jennifer Campbell
Vern Ceder
Michael Dawson
Paul Gries
Magnus Lie Hetland
Steve Holden
Mark Lutz
Alex Martelli
Jason Montojo
Anna Ravenscroft
Mark Summerfield
Greg Wilson
Navigation
Complete Book List
Author List
Publisher List
Last updated: Mar. 6, 2011

Families

block115394-lrq.blueboxgrid.com/uwdjango/families/

Families

- Julie's Family
- Jacob's Family

Main

Family Member Contact Info

block115394-lrq.blueboxgrid.com/uwdjango/families/contact/3/

Jason Newfield

- Birthdate: Jan. 12, 1972
- Phone: 616-552-2424
- Cell: 616-525-1166
- Email: jason@domain.com
- Address: 4450 Hollywood Blvd Hollywood, CA 98595

Families

Events Calendar

darkpits.com/events/

Logout

View Your Events

view by MONTH	view by YEAR	Search by title
year (4 digits):	Year (4 digits):	Enter Title:
month: January	Rows (per column): 3	Search
View	View	

March 2011						
Mon	Tue	Wed	Thu	Fri	Sat	Sun
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Hide »

Versions
Django 1.2.5

Time
CPU: 160.00ms (177.57ms)

Settings

HTTP Headers

Request Vars

SQL
33 queries in 11.54ms

Templates

Signals

Logging
0 messages

Recipe List

208.85.148.174/recipes/

Recipe List "Main Index" ::.

Search

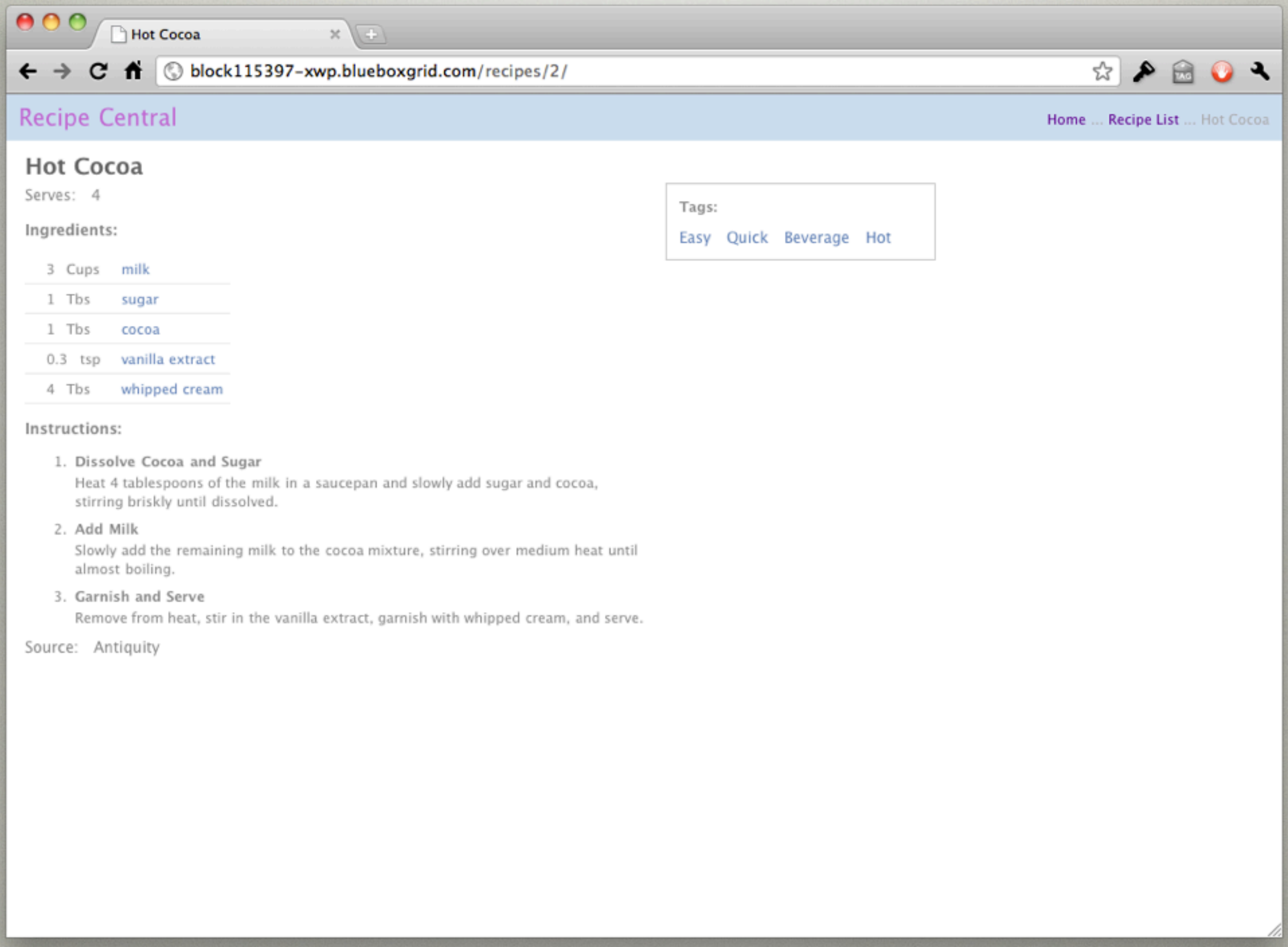
[Home] [Link 1] [Link 2] [Link 3]

Recipe List

Recipe Name	Category	Net Carbs	Protein
Beef Stuff	Lunch	22	3

Contact ::[jeff at mindcloud dot com]

Heading 5


Hot Cocoa
block115397-xwp.blueboxgrid.com/recipes/2/
Recipe Central
Home ... Recipe List ... Hot Cocoa
Hot Cocoa
Serves: 4
Ingredients:
3 Cups milk
1 Tbs sugar
1 Tbs cocoa
0.3 tsp vanilla extract
4 Tbs whipped cream
Tags:
Easy Quick Beverage Hot
Instructions:
1. Dissolve Cocoa and Sugar
Heat 4 tablespoons of the milk in a saucepan and slowly add sugar and cocoa, stirring briskly until dissolved.
2. Add Milk
Slowly add the remaining milk to the cocoa mixture, stirring over medium heat until almost boiling.
3. Garnish and Serve
Remove from heat, stir in the vanilla extract, garnish with whipped cream, and serve.
Source: Antiquity

boardgames

block115384-uav.blueboxgrid.com/games/

Board Games!

Everyone loves board games!

Welcome to my Board Game Collection!

Select a game for more information

- 7 Wonders
- Acquire
- Agricola
- Apples to Apples
- Bananagrams
- Battle Line
- Battlestar Galactica
- Bohnanza
- Brass
- Ca$h 'n Gun$
- Carcassonne
- Catan: Traders & Barbarians
- Citadels
- Coloretto
- Cosmic Encounter
- Cranium
- David & Goliath
- Dominion
- Dominion: Intrigue
- Dominion: Prosperity
- Dominion: Seaside
- El Grande
- Galaxy Trucker
- Hive
- Le Havre
- Memoir 44
- No Thanks!
- Pandemic
- Pandemic: On the Brink
- Power Grid
- Power Grid: Benelux/Central Europe
- Project Kells: Sacred Hill, High Kings of Tara & Poisoned Chalice
- Puerto Rico
- Ra
- Race for the Galaxy

Main menu

- Home
- Top 10 games
- Top 10 games by user
- About

- Team Cowboy API

boardgames

block115384-uav.blueboxgrid.com/games/10/

Board Games!

Everyone loves board games!

Board game details for Carcassonne

Title: Carcassonne

Description: A great entry level tile laying game.

Publisher: N/A
Year: 2000
Min number players: 2
Max number players: 5
Genre: tile laying

Average rating - 7.62

Ben rates this game a 8.0
Patti rates this game a 7.0 - Fun, easy to learn
Megan rates this game a 6.5 - Not my favorite.
Brian rates this game a 9.0 - Love, love, love this game. Need at least three players to allow for coopetition. With just two players, it's just kinda mean.

Rate this game

Name:

Rating:

Main menu

- Home
- Top 10 games
- Top 10 games by user
- About

- Team Cowboy API

random stuff

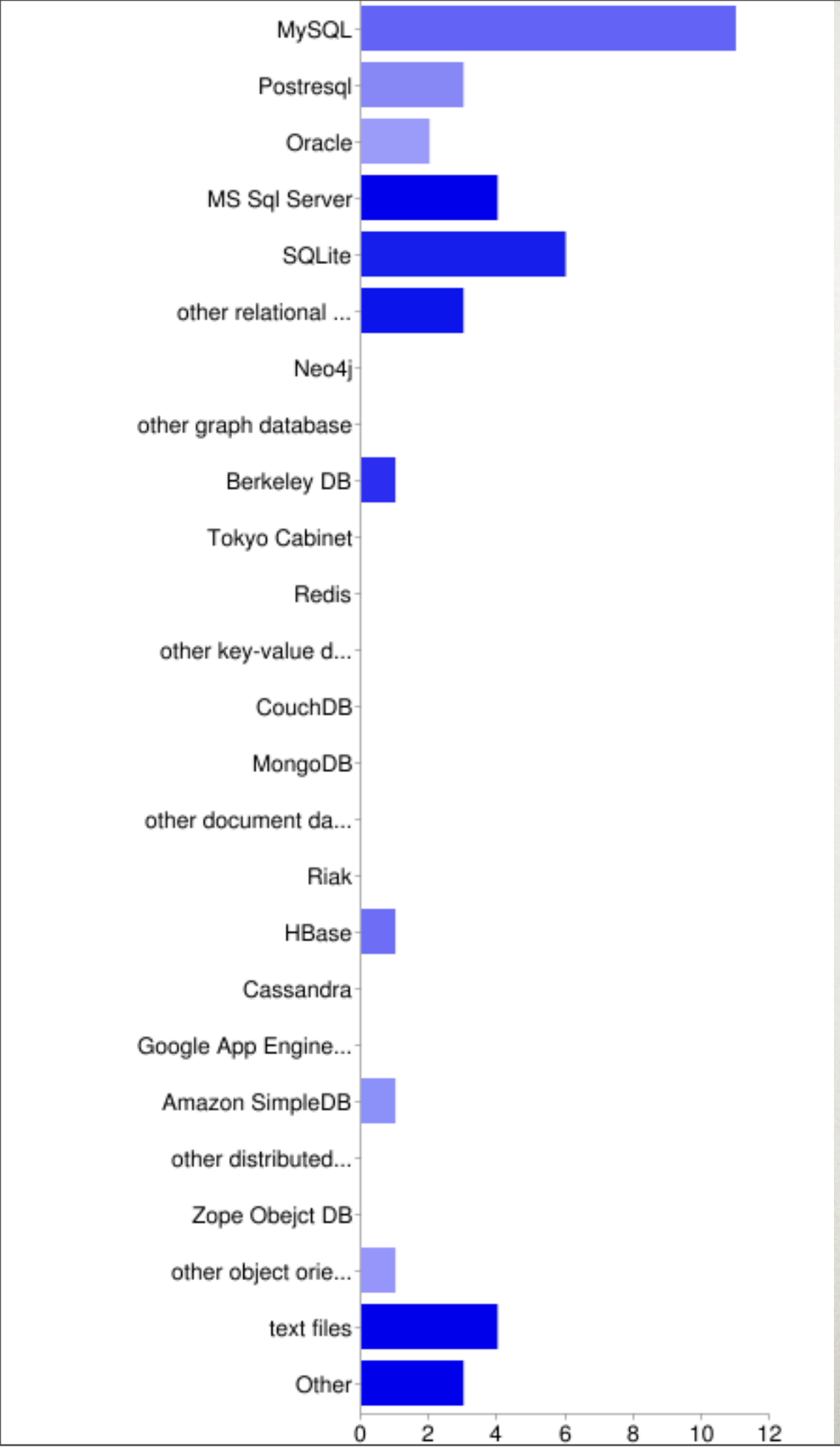
```
$ cat .gitignore
.DS_Store
*.pyc
*.swp
```

A moment to reflect

0:30

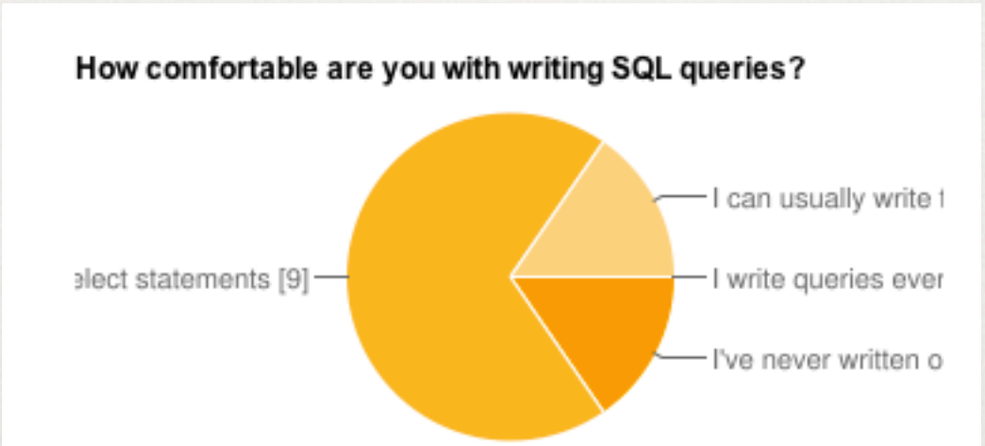
from the DB survey

(thanks!!)



Which of the following could you describe to a friend?

How comfortable are you with writing SQL queries?



I've never written one before	**2**	13%
I've written some select statements	**9**	56%
I can usually write the queries I want without looking at documentation	**2**	13%
I write queries everyday	**0**	0%

Lecture A (25)

overview and relational DBs

my bias

statically typed databases

dynamically typed languages

I learned relational databases before I learned programming.
Will I still think this in a few years?

STARTING AT THE END

unless you have special needs *and* enough time to evaluate databases,

just use postgresql and build your app

nearly everything you read about databases online is irrelevant for small and medium projects

by the time any of it matters,
you'll have real users with
real usage patterns to
optimize

nonetheless

in choosing databases, as in programming:

“premature optimization is the root of all evil”

-- Donald Knuth

http://shreevatsa.wordpress.com/2008/05/16/premature-optimization-is-the-root-of-all-evil/

"A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified."

what *are* the database options these days?

in that spirit, let's get the lay of the land

big picture:

relational

everything else

some definitions first

data

facts and statistics collected together for reference or analysis. See also datum.

1. **Computing** - the quantities, characters, or symbols on which operations are performed by a computer, being stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

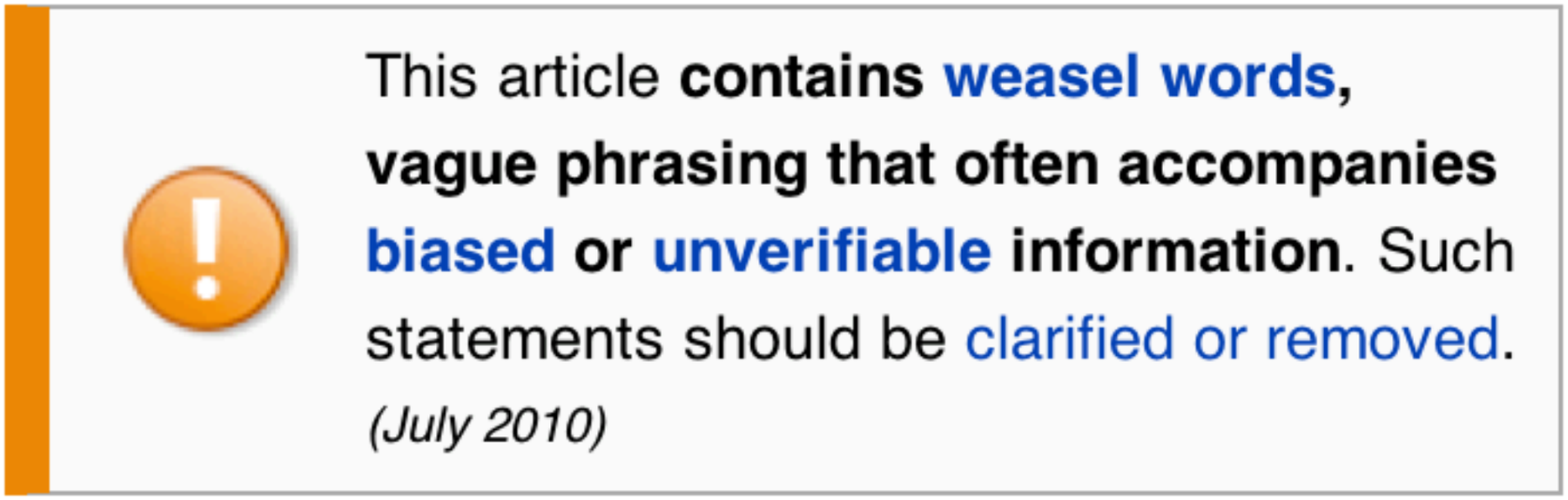
2. **Philosophy** - things known or assumed as facts, making the basis of reasoning or calculation.

   ORIGIN mid 17th cent. (as a term in philosophy): from Latin, plural of datum .

Oxford American Dictionary

but first, some definitions

Usage in English [edit]



This article **contains weasel words, vague phrasing that often accompanies biased or unverifiable information**. Such statements should be clarified or removed.

(July 2010)

We still can't agree wether data is singular or plural.

http://en.wikipedia.org/wiki/Data#Usage_in_English

“In computer science, data is information in a form suitable for use with a computer. Data is often distinguished from programs. A program is a set of instructions that detail a task for the computer to perform.

In this sense, data is thus everything that is not program code.”

http://en.wikipedia.org/wiki/Data_(computing)

this distinction is extended to the file system (rwxrwxrwx) and all the way down to the memory level in the OS and to the hardware level in some processors (http://en.wikipedia.org/wiki/Data_Execution_Prevention)

database

1. a structured set of data held in a computer, esp. one that is accessible in various ways.

Oxford American Dictionary

DBMS

DataBase Management System

1. software that handles the storage, retrieval, and updating of data in a computer system.

    - Interface drivers — These drivers are code libraries that provide methods to prepare statements, execute statements, fetch results, etc.

    - SQL engine — This component interprets and executes the DDL, DCL, and DML statements. It includes three major components (compiler, optimizer, and executor).

    - Transaction engine — Ensures that multiple SQL statements either succeed or fail as a group, according to application dictates.

    - Relational engine — Relational objects such as Table, Index, and Referential integrity constraints are implemented in this component.

    - Storage engine — This component stores and retrieves data from secondary storage, as well as managing transaction commit and rollback, backup and recovery, etc.

`http://en.wikipedia.org/wiki/Database`

ACID - properties of *transactions in a database*

1. Atomicity - “all or nothing”
2. Consistency - visible DB is always in a valid state at the beginning and end of transactions
3. Isolation - concurrent transactions cannot see other transactions uncommitted data
4. Durability - once the client is told the transaction succeeded, even hardware failure won’t loose the data.

http://en.wikipedia.org/wiki/ACID

ACID is defined in terms of transactions. If a database or storage engine doesn’t have transactions, it isn’t ACID.

once there is enough data,
or enough users,
the system likely isn't ACID

(thus, NOSQL)

NOSQL

1. No SQL

2. Not Only SQL

3. Next Generation Databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontal scalable**. The original intention has been modern **web-scale databases**. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: **schema-free**, **easy replication** support, **simple API**, **eventually consistent**, a **huge data** amount, and more. So the misleading term "nosql" should be seen as an alias to something like the definition above.

`http://nosql-database.org/`

`CAP theorem` - distributed system

1. Consistency - all nodes see the same data

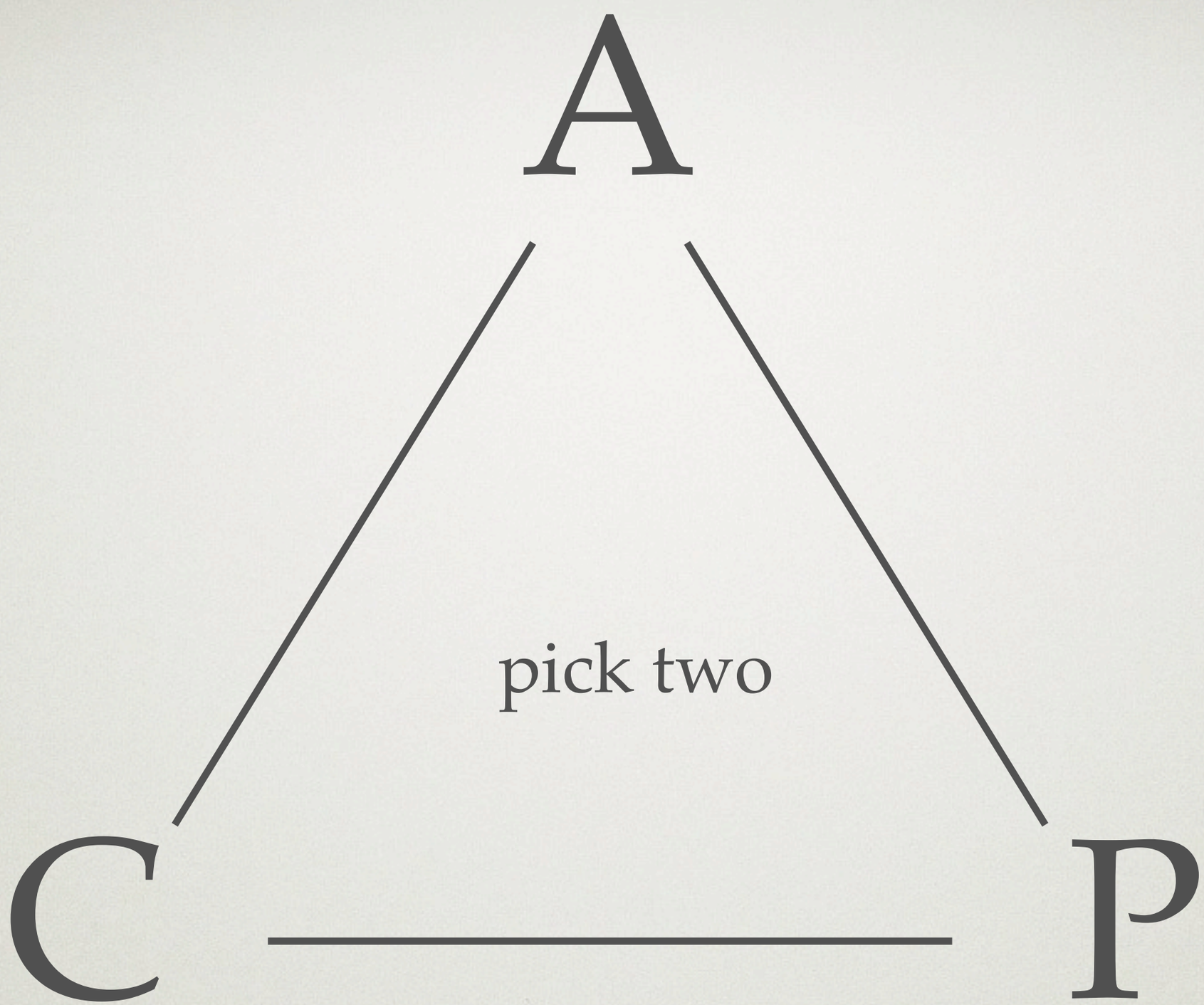
2. Availability - the system responds even with some failed nodes

3. Partition tolerance - the system keeps functioning even if parts cannot see each other

Pick any two.

`This is a great article --> http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

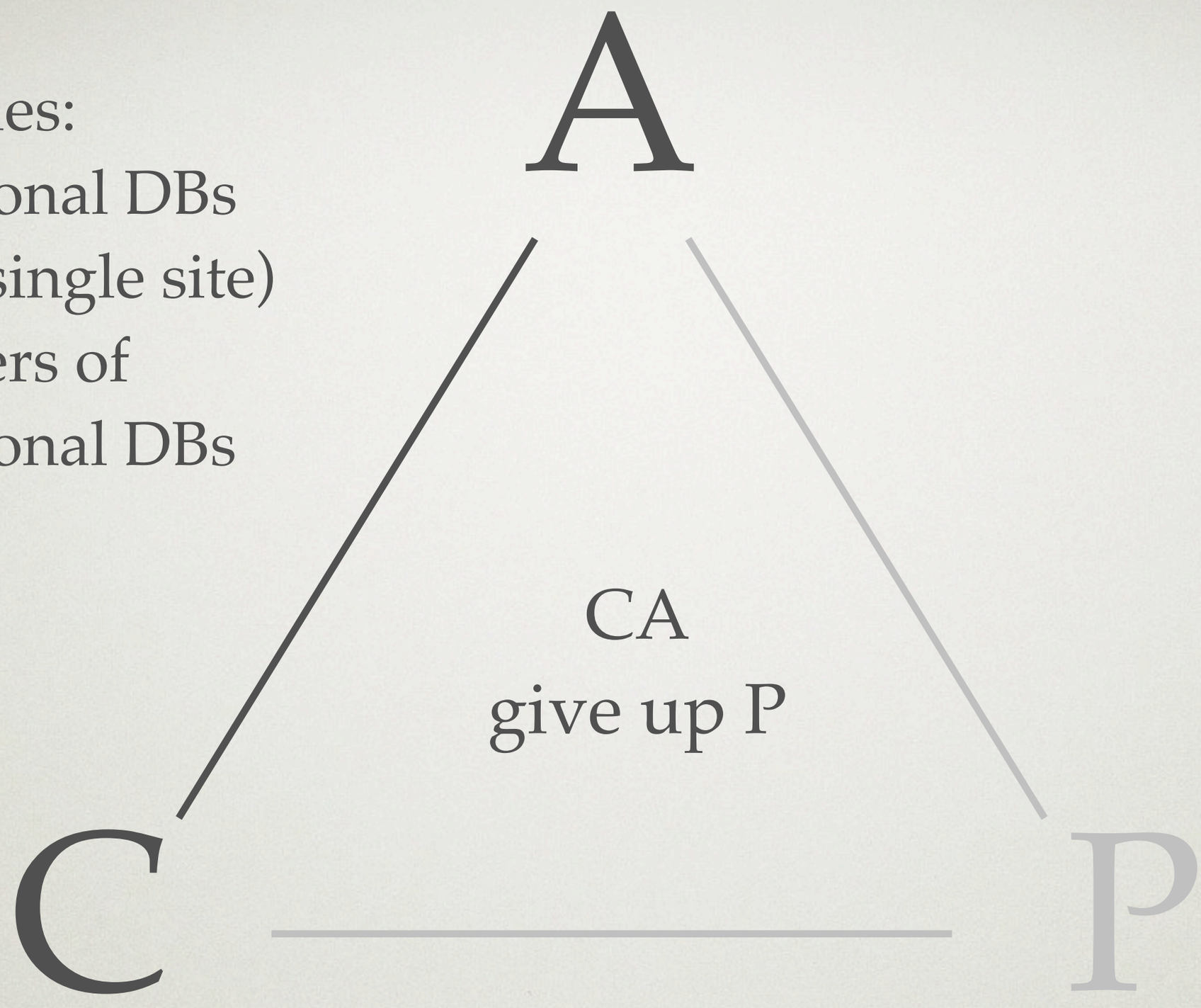
Once your system is larger than two machines with data on them, you only get two.

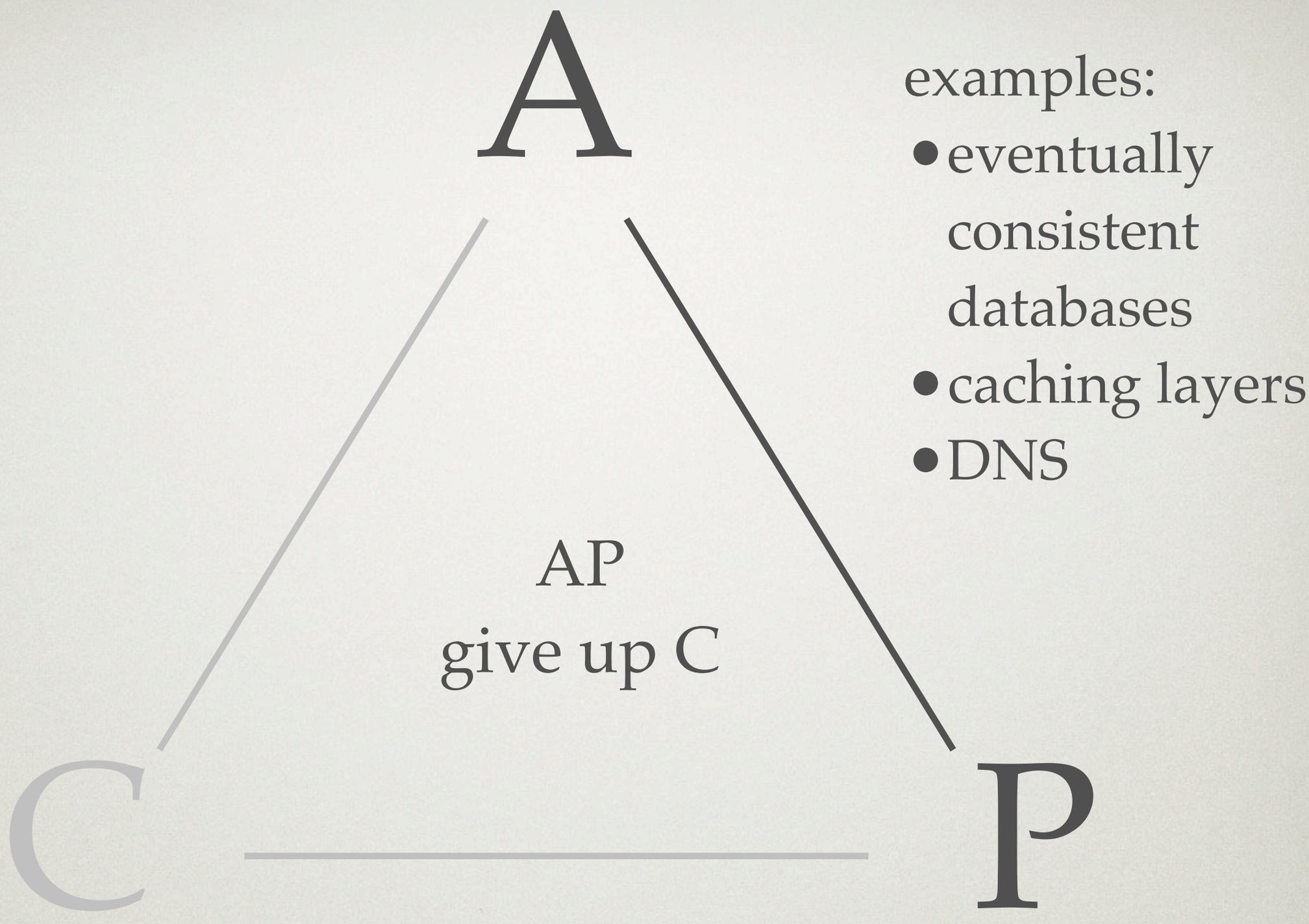


inspired by: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
http://blog.nahurst.com/visual-guide-to-nosql-systems

examples:

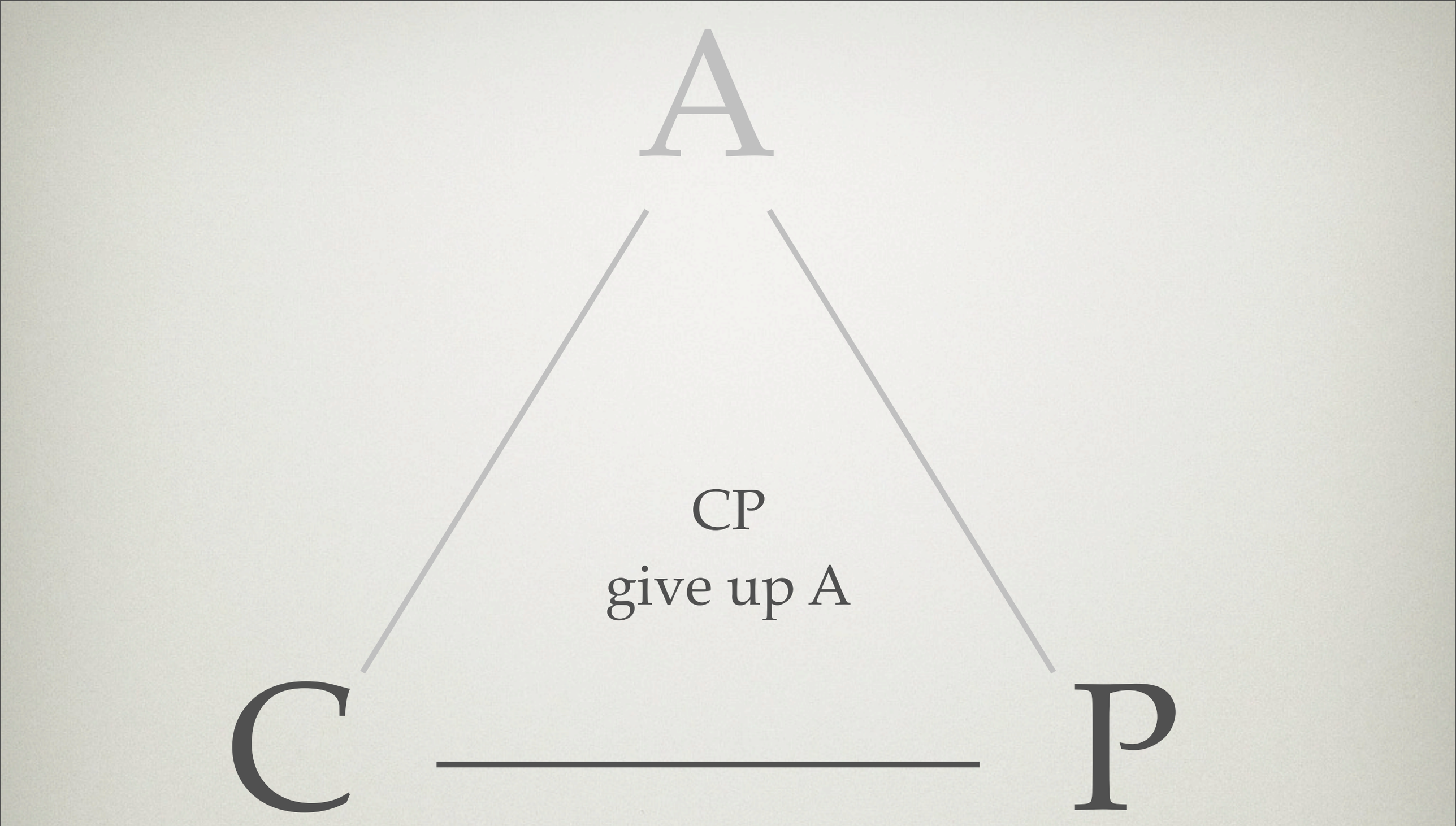
- relational DBs (at a single site)
- clusters of relational DBs





examples:

- eventually consistent databases
- caching layers
- DNS



examples:

- distributed locking
- distributed databases (in some configurations)

these are unavoidable tradeoffs

all of them are useful in some situations

If you can figure out how to get all three, your career is set and your name will be remembered.

exactly where a database fits depends upon configuration in many cases

think of CAP as a set of dials to tweak for your system

also, different parts of the system will choose different trade offs

ex:

single MySQL - CA

caching layer - AP

a very common deployment pattern is to have a single database with a distributed cache running on each webserver

Relational Databases (RDBMS)

a database which
implements
the relational model

(thank you, tautology)

what is the relational model?

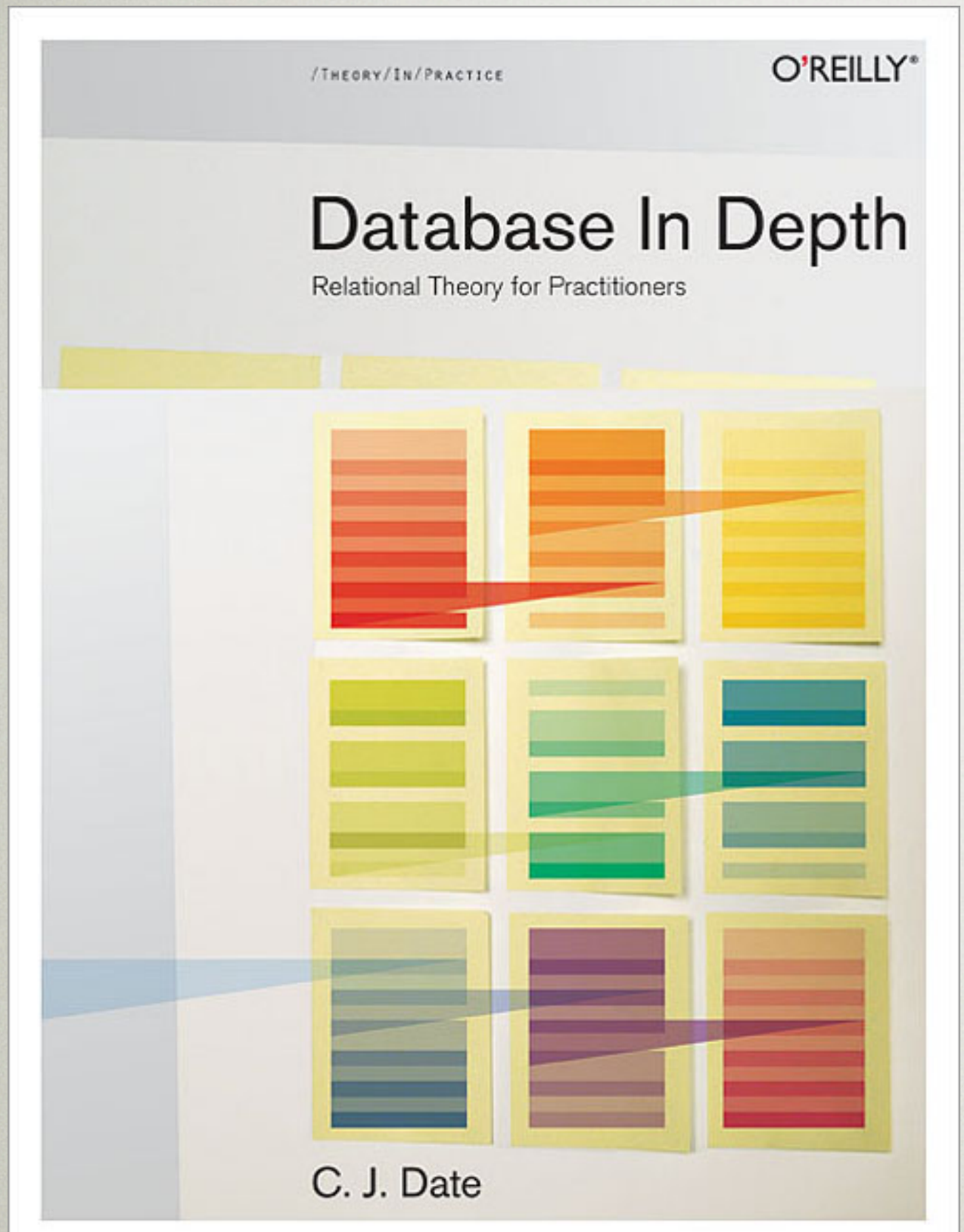
Relational Database

(wrong, but useful definition)

1. A database system which uses tables, views, constraints, joins and SQL queries.

2. A database created by MySQL, PostgreSQL, SQLite, Oracle, MS SQL Server, etc.

(tautology again)



read this for the real answer

http://oreilly.com/catalog/9780596100124

when people say:
relational database

they usually mean:
Relational DataBase
Management System

DBMS

DataBase Management System

1. software that handles the storage, retrieval, and updating of data in a computer system.

    - Interface drivers — These drivers are code libraries that provide methods to prepare statements, execute statements, fetch results, etc.

    - SQL engine — This component interprets and executes the DDL, DCL, and DML statements. It includes three major components (compiler, optimizer, and executor).

    - Transaction engine — Ensures that multiple SQL statements either succeed or fail as a group, according to application dictates.

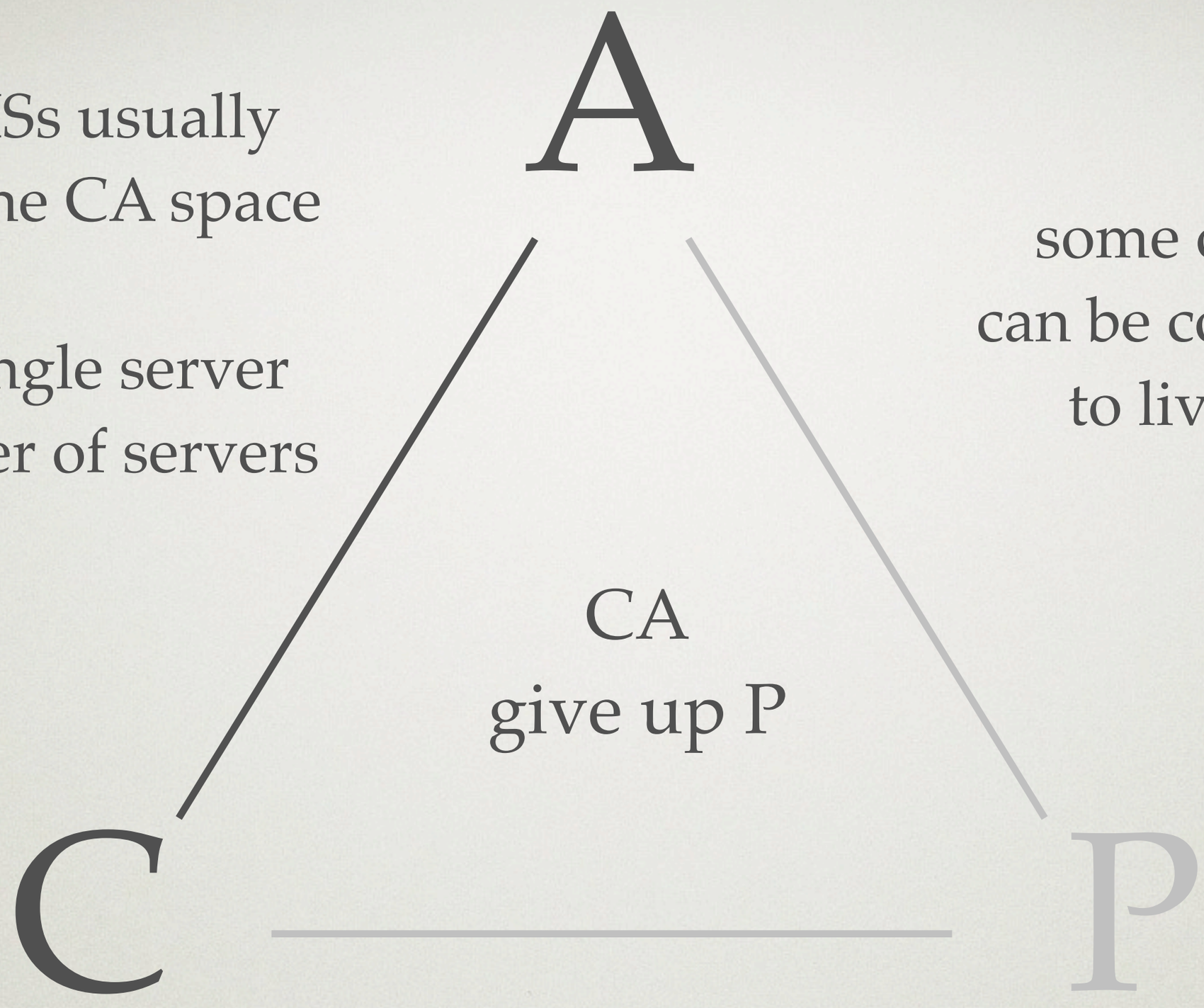
    - Relational engine — Relational objects such as Table, Index, and Referential integrity constraints are implemented in this component.

    - Storage engine — This component stores and retrieves data from secondary storage, as well as managing transaction commit and rollback, backup and recovery, etc.

`http://en.wikipedia.org/wiki/Database`

RDBMSs usually
live in the CA space

on a single server
or cluster of servers



some of them
can be configured
to live here

CA
give up P

... or here

some common terms

I'm working at the pragmatic, user-of-an-existing-product level here. See Database in Depth for the principles of the relational model itself.

tables

grid, columns have types

views

constraints

primary key

tables need a primary key or they’re just a bag - need a way to refer to specific rows.
surrogate keys are very common

foreign keys

indexes

stored procedures

other terms?

relational implementations

- SQLite
- MySQL
- PostgreSQL
- many, many others

There are *tons* of database systems – for the purposes of this talk, I'm choosing a few popular open source implementations of each type. Please assume that there are commercial implementations and many other open source implementations as well.

common features

- ad hoc queries (SQL)
- maintained indexes
- enforce constraints
- transactions
- mature
- documentation, books, and training are all easily available

SQLite

"SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. "

- superpowers - small, serverless library
- kryptonite - concurrent writes
- interesting - SQLite is the most widely deployed SQL database engine in the world.
  Also: `con = sqlite3.connect(":memory:")`

http://www.sqlite.org/

seriously, SQLite is *everywhere*
my phone, each browser, built into OSX, Ubuntu, Python

MySQL

"The world's most popular open source database."

- superpowers - widely used
- kryptonite - Oracle
- interesting - HandlerSocket - talks *directly* to InnoDB storage layer for key-value (row) queries. Much faster for PK queries. You get NOSQL and SQL access to the same data. (link)

http://www.mysql.com/

PostgreSQL

"The world's most advanced open source database."

- superpowers - mature, flexible, solid
  also: datatypes: GIS, array, hstore, graphs, etc.
- kryptonite - uncool?
- interesting - hstore - key-value hashes in a column.
  It's like having a Python dictionary stored as the value of one column.
  (they have an array (list) datatype, too)

http://www.postgresql.org/

the future

Drizzle

"A Lightweight SQL Database for Cloud and Web."

Drizzle is a community-driven open source project that is forked from the popular MySQL database.

- superpowers - modular architecture, cloud & ACID
- kryptonite - young?
- interesting - most everything is pluggable

http://www.postgresql.org/

Python APIs

there is a python module for pretty much every database

- SQLite - sqlite3 (built-in), or pysqlite
- MySQL - mysql-python
- PostgreSQL - Psycopg and others
- Drizzle - drizzle-python
- ODBC - pyodbc

for example

ODBC?

think:

printer drivers for databases

language implementors
make an ODBC library for
the language

database implementors
make an ODBC driver for
the database

in theory, any ODBC library can then use any database which has an ODBC driver

in practice, it mostly works

especially on windows

but... you miss out on some database specific special features and optimizations

there are python wrappers for most database libraries

they get used instead of ODBC most of the time

(SQL Server on windows is an exception - pyodbc seems to have the best support.)

OK, but...

"Do I have to learn a new module for *every* database?"

dbapi 2

http://www.python.org/dev/peps/pep-0249/

"This API has been defined to encourage similarity between the Python modules that are used to access databases. By doing this, we hope to achieve a consistency leading to more easily understood modules, code that is generally more portable across databases, and a broader reach of database connectivity from Python."

http://www.python.org/dev/peps/pep-0249/

```
conn = sqlite3.connect('/tmp/example')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''create table stocks
(date text, trans text, symbol text,
 qty real, price real)''')

# Insert a row of data
c.execute("""insert into stocks
          values ('2006-01-05','BUY','RHAT',100,35.14)""")

# Save (commit) the changes
conn.commit()

# We can also close the cursor if we are done with it
c.close()
```

http://docs.python.org/release/2.6.5/library/sqlite3.html

```python
# Never do this -- insecure!
symbol = 'IBM'
c.execute("... where symbol = '%s'" % symbol)

# Do this instead
t = (symbol,)
c.execute('select * from stocks where symbol=?', t)

# Larger example
for t in [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
          ('2006-04-05', 'BUY', 'MSOFT', 1000, 72.00),
          ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
         ]:
    c.execute('insert into stocks values (?,?,?,?,?)', t)
```

http://docs.python.org/release/2.6.5/library/sqlite3.html

```
>>> conn.row_factory = sqlite3.Row
>>> c = conn.cursor()
>>> c.execute('select * from stocks')
<sqlite3.Cursor object at 0x7f4e7dd8fa80>
>>> r = c.fetchone()
>>> type(r)
<type 'sqlite3.Row'>
>>> r
(u'2006-01-05', u'BUY', u'RHAT', 100.0, 35.14)
>>> len(r)
5
>>> r[2]
u'RHAT'
>>> r.keys()
['date', 'trans', 'symbol', 'qty', 'price']
>>> r['qty']
100.0
>>> for member in r: print member
...
2006-01-05
BUY
RHAT
100.0
35.14
```

http://docs.python.org/release/2.6.5/library/sqlite3.html

ORM

Object Relational Mapping

program data is often naturally modeled as trees or graphs of objects

relational databases have tables, keys, joins, etc

ORMs try to translate...

but it's always a leaky abstraction

there is a ton of controversy around ORMs

(and, honestly, relational databases as a whole)

"...; **there is no good solution to the object/relational mapping problem.** There are solutions, sure, but they all involve serious, painful tradeoffs. And the worst part is that you can't usually see the consequences of these tradeoffs until much later in the development cycle."

http://www.codinghorror.com/blog/2006/06/object-relational-mapping-is-the-vietnam-of-computer-science.html

a couple of ORMs of note

Django's ORM (models)

you guys are familiar with this.

it works pretty well

especially if the
ORM owns the DB

when you reach the limits, there is a clear, supported escape hatch:

This is best illustrated with an example. Suppose you've got the following model:

```
class Person(models.Model):
    first_name = models.CharField(...)
    last_name = models.CharField(...)
    birth_date = models.DateField(...)
```

You could then execute custom SQL like so:

```
>>> for p in Person.objects.raw('SELECT * FROM myapp_person'):
...     print p
John Smith
Jane Jones
```

or completely custom SQL:

```
def my_custom_sql():
    from django.db import connection, transaction
    cursor = connection.cursor()

    # Data modifying operation - commit required
    cursor.execute("UPDATE bar SET foo = 1 WHERE baz = %s", [self.baz])
    transaction.commit_unless_managed()

    # Data retrieval operation - no commit required
    cursor.execute("SELECT foo FROM bar WHERE baz = %s", [self.baz])
    row = cursor.fetchone()

    return row
```

if you're using Django,
don't let anyone talk you out
of using it's models

if you can't use the models,
maybe Django isn't the right
tool for the job

SQLAlchemy

http://www.sqlalchemy.org/

if you need to connect to an existing database, look into SQLAlchemy

SQLAlchemy

home | features | news | documentation | wiki | community

The Python SQL Toolkit and Object Relational Mapper

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

http://www.sqlalchemy.org/

SQLALCHEMY'S PHILOSOPHY

SQL databases behave less like object collections the more size and performance start to matter; object collections behave less like tables and rows the more abstraction starts to matter. SQLAlchemy aims to accommodate both of these principles.

SQLAlchemy doesn't view databases as just collections of tables; it sees them as relational algebra engines. Its object relational mapper enables classes to be mapped against the database in more than one way. SQL constructs don't just select from just tables—you can also select from joins, subqueries, and unions. Thus database relationships and domain object models can be cleanly decoupled from the beginning, allowing both sides to develop to their full potential.

The main goal of SQLAlchemy is to change the way you think about databases and SQL!

Most importantly, **SQLAlchemy is not just an ORM.** Its data abstraction layer allows construction and manipulation of SQL expressions in a platform agnostic way, and offers easy to use and superfast result objects, as well as table creation and schema reflection utilities. No object relational mapping whatsoever is involved until you import the `orm` package. Or use SQLAlchemy to write your own !

in conclusion

http://bjclark.me/2009/08/nosql-if-only-it-was-that-easy/

killer feature of
relational databases:

application independence

(also: programming language independence)

data often lives far longer than applications, especially inside organizations. Also: reporting, reporting, reporting. Again: ad hoc reporting. Rich tool availability.

Lab A

(20)

Lab A

- no lab A today

Something in the schedule had to give...

Break

(10)

Guest Speaker

(30)

Dave Peck

http://davepeck.org/

API Talks (20)

talks

Break

(10)

RANDOM

Lecture B (20)

Non-relational databases, observations and recommendataions

in order to shard a relational database, you give up a bunch of the relational engine's features

(shard == data partition)

and add operational overhead

“if you’re deploying memcache on top of your database, you’re inventing your own ad-hoc, difficult to maintain NoSQL database”

-- Ian Eure from Digg

http://www.rackspace.com/cloud/blog/2010/02/25/should-you-switch-to-nosql-too/

I’m going to run through a bunch of popular NOSQL databases

a few notes before we start

the categorizations aren't cleanly separated

people mean a lot of different things when they say “NOSQL”

key-value?

schemaless?

durable?

scalable? (automatically)

fast?

big data?

magic pixie dust?

whenever you read
“NOSQL”, you have to infer
the sub-set of properties the
author has assumed

this makes it basically impossible to tell what is going on without knowing at least a bit about a lot of the popular implementations

which brings us to this talk

- pre-‘nosql’
- key-value stores (on disk)
- distributed (automatically)
- document
- other

again these are best effort groupings... there is some overlap
I’m going to go through the groups and give an example or two of notable databases in each group. I’ve probably missed something.

pre-‘nosql’

- fixed width records
- text files
- csv - comma separated values
- excel
- bdb - Berkley db

key-value (on disk)

- usually both key and value are limited to strings
- usually read and write directly to disk

many of the other databases I'll mention also have a key-value interface - however, they either have richer data distribution, or richer values (more than just strings on disk)

key-value (on disk)

- dbm (1979) and **many** derivitives
  there is an **`anydbm`** module!
- berkely db (also owned by Oracle now)
  bdb will be removed from stdlib soon
- Tokyo Cabinet / Kyoto Cabinet

`http://en.wikipedia.org/wiki/Dbm`

Tokyo/Kyoto Cabinet

"A modern implementation of DBM."

- superpowers - focus
- kryptonite - focus
- interesting - fastest way to be sure data is on disk?
  Each has a suite of tools, including an HTTP API:
  Tokyo Tyrant
  Kyoto is newer, but GPL. Tokyo is LGPL
  Naming craziness (see next slide)

`http://fallabs.com/tokyocabinet/`

```
var now = new Date();
    if((now.getFullYear() + now.getMonth() + now.getDate() + now.getHours()) % 4 == 0){
      var label;
      switch((now.getMonth() + now.getDay() + now.getHours() + now.getMinutes()) % 24){
      default: label = "Tokyo Cabinet"; break;
      case 1: label = "Shibuya Cabinet"; break;
      case 2: label = "Harajuku Cabinet"; break;
      case 4: label = "Aoyama Cabinet"; break;
      case 5: label = "Gakudai Cabinet"; break;
      case 7: label = "Shinjuku Cabinet"; break;
      case 8: label = "Ikebukuro Cabinet"; break;
      case 10: label = "Akihabara Cabinet"; break;
      case 11: label = "Ueno Cabinet"; break;
      case 13: label = "Sugamo Cabinet"; break;
      case 14: label = "Akasaka Cabinet"; break;
      case 16: label = "Roppongi Cabinet"; break;
      case 17: label = "Yokohama Cabinet"; break;
      case 19: label = "Saitama Cabinet"; break;
      case 20: label = "Tokorozawa Cabinet"; break;
      case 22: label = "Kyoto Cabinet"; break;
      case 23: label = "Nagoya Cabinet"; break;
      }
      var text;
      switch((now.getMonth() + now.getDate() + now.getMinutes()) % 12){
      default: text = "super hyper ultra database manager"; break;
      case 1: text = "much quicker database manager"; break;
      case 3: text = "the ultimate database manager"; break;
      case 5: text = "the supreme database manager"; break;
      case 7: text = "the lightning database manager"; break;
      case 9: text = "the mighty unbeatable invincible database manager"; break;
      case 11: text = "the dinosaur wing of database managers"; break;
      }
      elem.firstChild.nodeValue = label + ": " + text;
```

is he just messing with us?

distributed databases

- data is stored on multiple machines
- transparent to applications
- add capacity without downtime
- survives failure of some machines
- seminal papers: Big Table, Dynamo

`http://labs.google.com/papers/bigtable.html`

`http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf`

distributed databases

- Google App Engine datastore
- Amazon Simple DB
- HBase (Hadoop)
- Riak
- Voldemort
- Cassandra

these guys scale

Cassandra

"... a highly scalable second-generation distributed database, bringing together Dynamo's fully distributed design and Bigtable's ColumnFamily-based data model."

- superpowers - writes are faster than reads
- kryptonite - getting started? scaling down?
- interesting - fully automated data distribution,
  perf vs. durability tradeoff at client
  no single point of failure

http://cassandra.apache.org/

document databases

- basically key-value databases with very rich values
- values logically JSON-like
- engine understands structure, so you can index, query into, and across docs
- schemaless - store any shape data

CouchDB

"Apache CouchDB is a document-oriented database that can be queried and indexed in a MapReduce fashion using JavaScript. CouchDB also offers incremental replication with bi-directional conflict detection and resolution."

- superpowers - bi-directional replication
- kryptonite - ad hoc queries? compaction?
- interesting - native API is HTTP/REST
  can store HTML/JavaScript apps *in* the database
  queries are persisted map/reduce, calculated on *write*

`http://couchdb.apache.org/`

Cassandra and CouchDB are *both* apache projects

MongoDB

"MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database."

- superpowers - ad hoc queries, in place updates
- kryptonite - durability needs multiple servers, scaling complexity
- interesting - more like a DBMS than just a storage engine

http://www.mongodb.org/

feels the most like a 'database'

Redis

> “Redis is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.”

- superpowers - FAST, rich data types, caching, messaging
- kryptonite - in memory - delayed persistence & size limits
- interesting - collections map well to dynamic languages powerful, fundamental building blocks
  https://github.com/amix/redis_wrap
  https://github.com/peta/redis-natives-py

http://redis.io/

not a good choice for a primary datastore for most applications

other

- object databases
  ZODB - stores linked Python objects
- graph databases
  Neo4j - nodes & edges, graph queries
- RDF databases
- full text search databases
  Solr, Sphinx, ElasticSearch
- caching - memcache

SOME OBSERVATIONS

overwhelming, right?

amazing that all of
this is available for free

everything has a Python interface available

yea!

common to find a short description!

every project had a one sentence description near the top of the front page!
there are a lot of DBs, but most of them are trying to be clear about what they are

RDBMS people assume DB lives longer than apps, and multiple apps will r/w data

NOSQL people assume app and DB grow together

WARNING! Generalization of unspoken assumptions.

If you have reporting, you always have at least two apps.

related:

where is the responsibility
for data integrity?

in DB? in app?

ON DATA MODELING

ignore anyone who claims you don't have to model your data in NOSQL DBs

"The document DB sort of swaps the complexities: SQL has inflexible data and flexible queries, document DBs are the other way around."

http://stackoverflow.com/questions/1189911/non-relational-database-design

this is true for almost all NOSQL, MongoDB is the closest to an exception here

in many NOSQL DBs, you need to know your queries up front, and arrange your data to make fast queries

this is true for SQL DBs, too

sorry, no free lunch

http://nosql.mypopescu.com/post/451094148/nosql-data-modeling

RECOMMENDATIONS

“Picking good technologies for your project is hard work”

-- Salvatore Sanfilippo (Redis)

http://news.ycombinator.com/item?id=2052852

if you're working on a project which has already chosen DB tech, just use that

if your team has experience with a DB, just use that

first, if you aren't already,
get comfortable with a
relational database

then play with some other options on the side

many of these only work on Linux - setup an Ubuntu VM for experimenting

when comparing DBs

- loading and exporting data
- queries - what is easy vs. hard?
- when does it ***actually*** write to disk?
- recovery process after a crash?
- ad hoc queries? reporting?
- schema changes - when does the work happen? DB available while changing schema? Where: server? client?

example: Mongo doesn't crash much, but it can corrupt data and require a full DB repair step before serving data again. It sounds like CouchDb crashes more, but it's an append only DB, no delay to restart at all.

a biased, short list

start at the top, work down

- PostgreSQL
- App Engine datastore
- Redis
- MongoDB
- Cassandra

this is just my opinion... but if you work through these, you'll cover a lot of ground

ENDING AT THE START

unless you have special needs *and* enough time to evaluate databases,

just use postgresql and build your app

Lab B

(20)

2:35

Lab B

- Which are you most interested in experimenting with? Why?
- Quickly describe a project you're considering building. What does the group think is the right DB?

- PostgreSQL
- App Engine datastore
- Redis
- MongoDB
- Cassandra
- other?

make groups of 3-5

WRAPUP

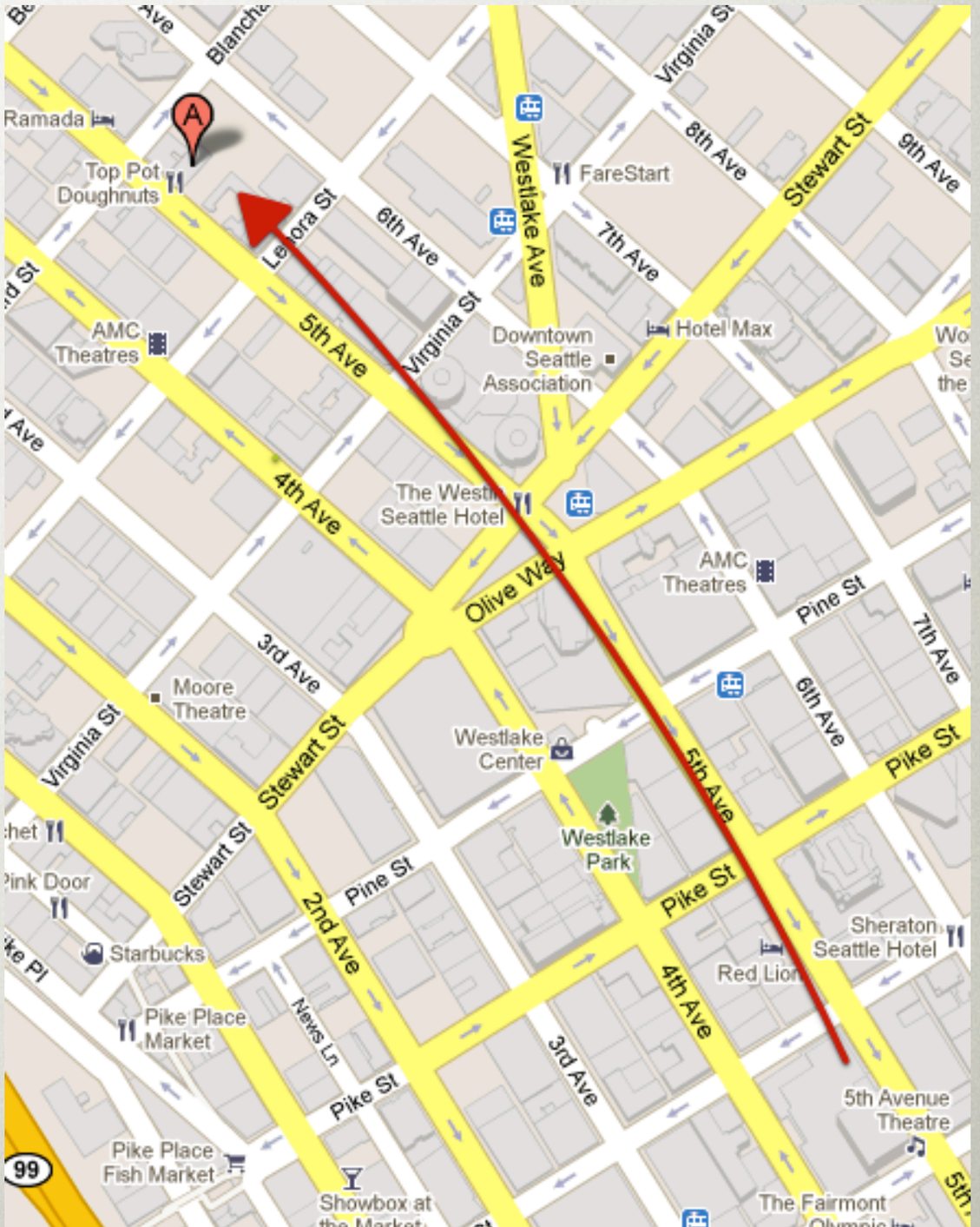
2:55

no Brian this weekend --> PyCon!

Info

Office hours:
Sunday 2-5pm

Top Pot Donuts
2124 5th Ave
Seattle, WA 98121
206-728-1966



Assignment

turn in here: `http://is.gd/uwipip_week9`

- Sign up for Google App Engine:
  `https://appengine.google.com/` (phone with SMS needed)
- List all of the queries in your week 7/8 app
- Import your week 7/8 data into a NOSQL database. Maybe App Engine or MongoDB?
- Attempt to write equivalent queries against your NOSQL DB.
- Check in your import script and query script.
- stretch - port your week 7/8 app to use the NOSQL DB.
- super stretch - port your week 7/8 app to App Engine using django-nonrel:
  `http://www.allbuttonspressed.com/projects/django-nonrel`

Would you rather do something different with NOSQL? Ask me, it's probably OK.