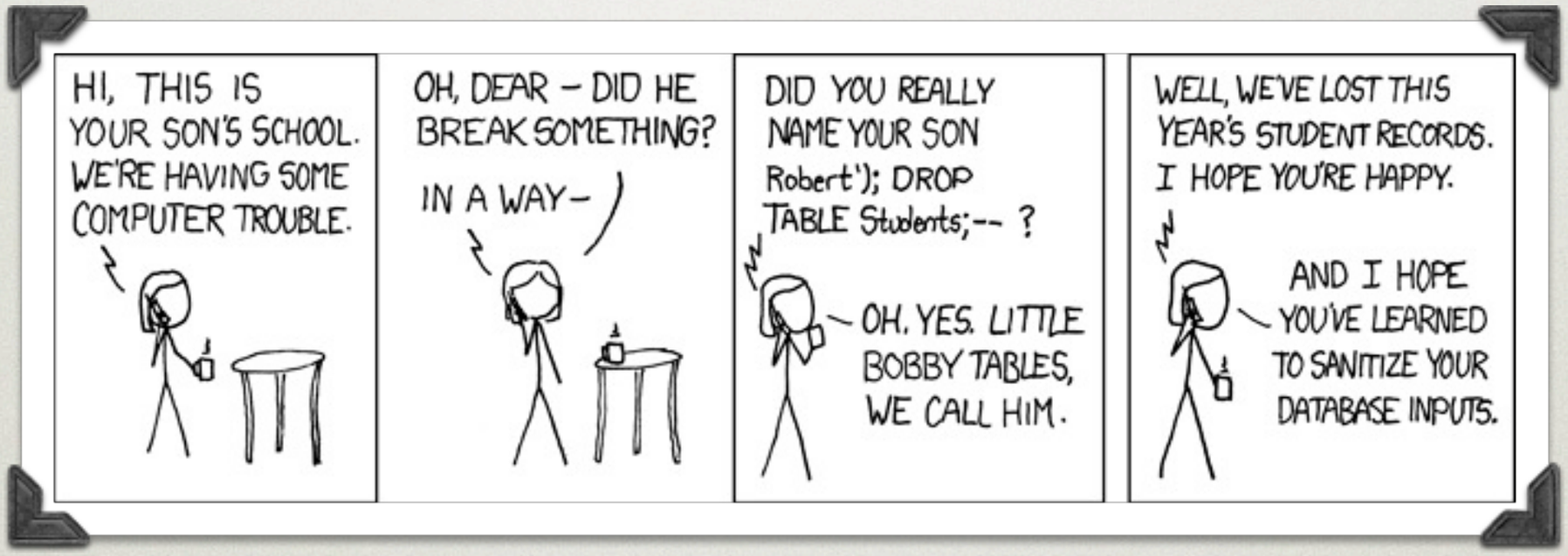
Internet Programming in Python - Week 9

Databases



http://xkcd.com/327/

Brian Dorsey
brian@dorseys.org

ANNOUNCEMENTS

-

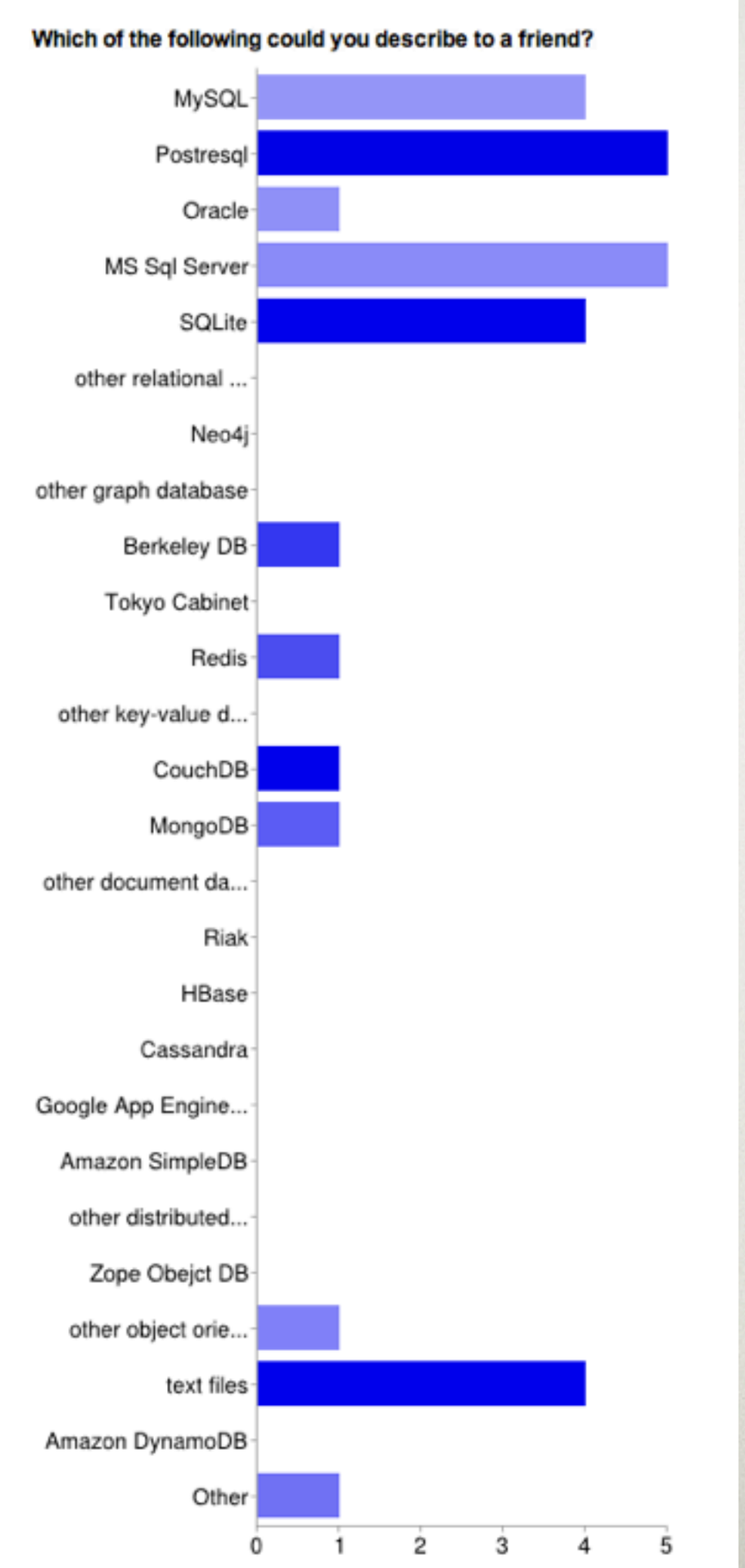
Questions and Review

some thoughts from the assignments

A moment to reflect

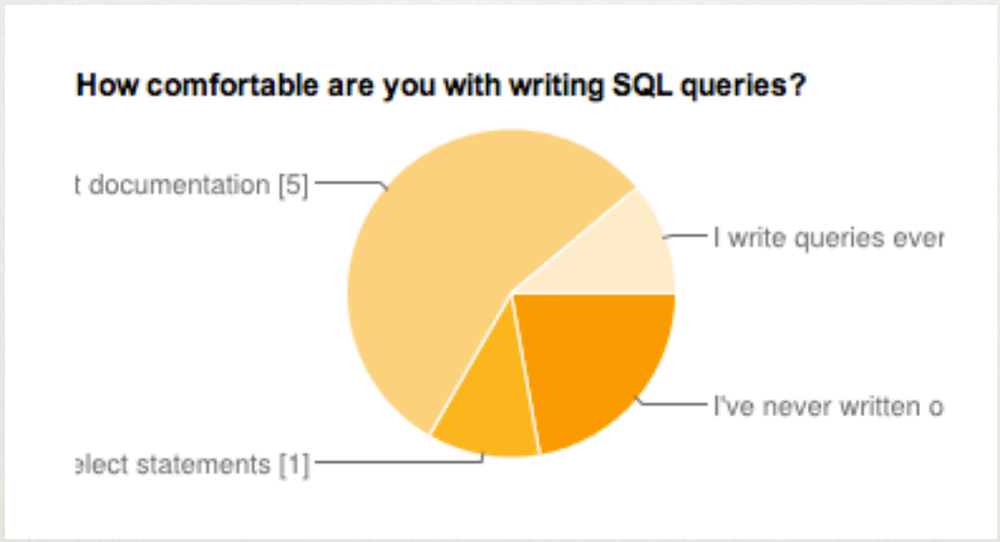
from the DB survey

(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**	22%
I've written some select statements	**1**	11%
I can usually write the queries I want without looking at documentation	**5**	56%
I write queries everyday	**1**	11%

Lecture A

overview and relational DBs

my bias

statically typed databases

dynamically typed languages

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?

too many to count

big picture:

relational

everything else

some definitions first

data

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

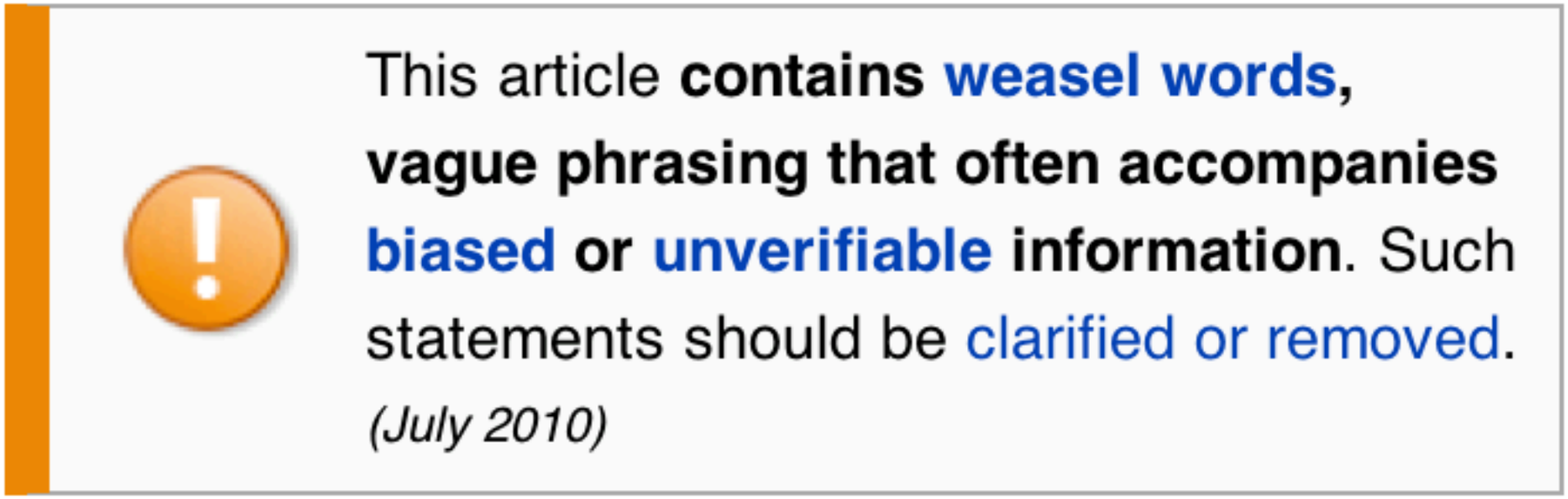
1. **Computing** - the quantities, characters, or symbols on which operations are performed by a computer, being stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

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

Usage in English [edit]



We still can't agree whether 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."

rwxrwxrwx

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

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 lose the data.

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

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

(thus, NOSQL)

(because it's too bit for one computer)

"enough" has become *very* large recently

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/

Scale

1. with many users, does the system still respond the same for all of them?

Speed (latency)

1. for a given user, is the site fast?

`CAP theorem` - distributed system

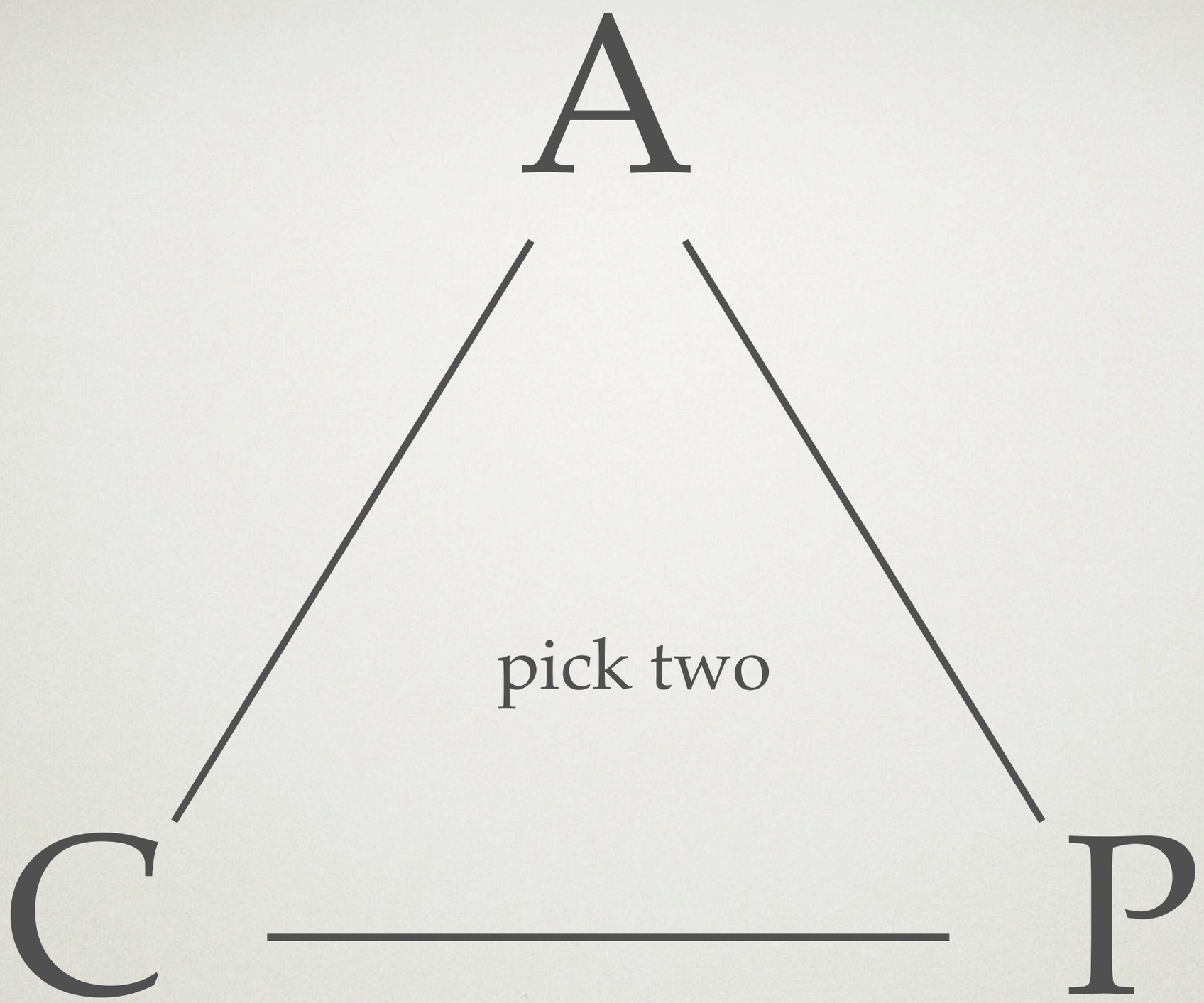
1. Consistency - all nodes see the same data

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.

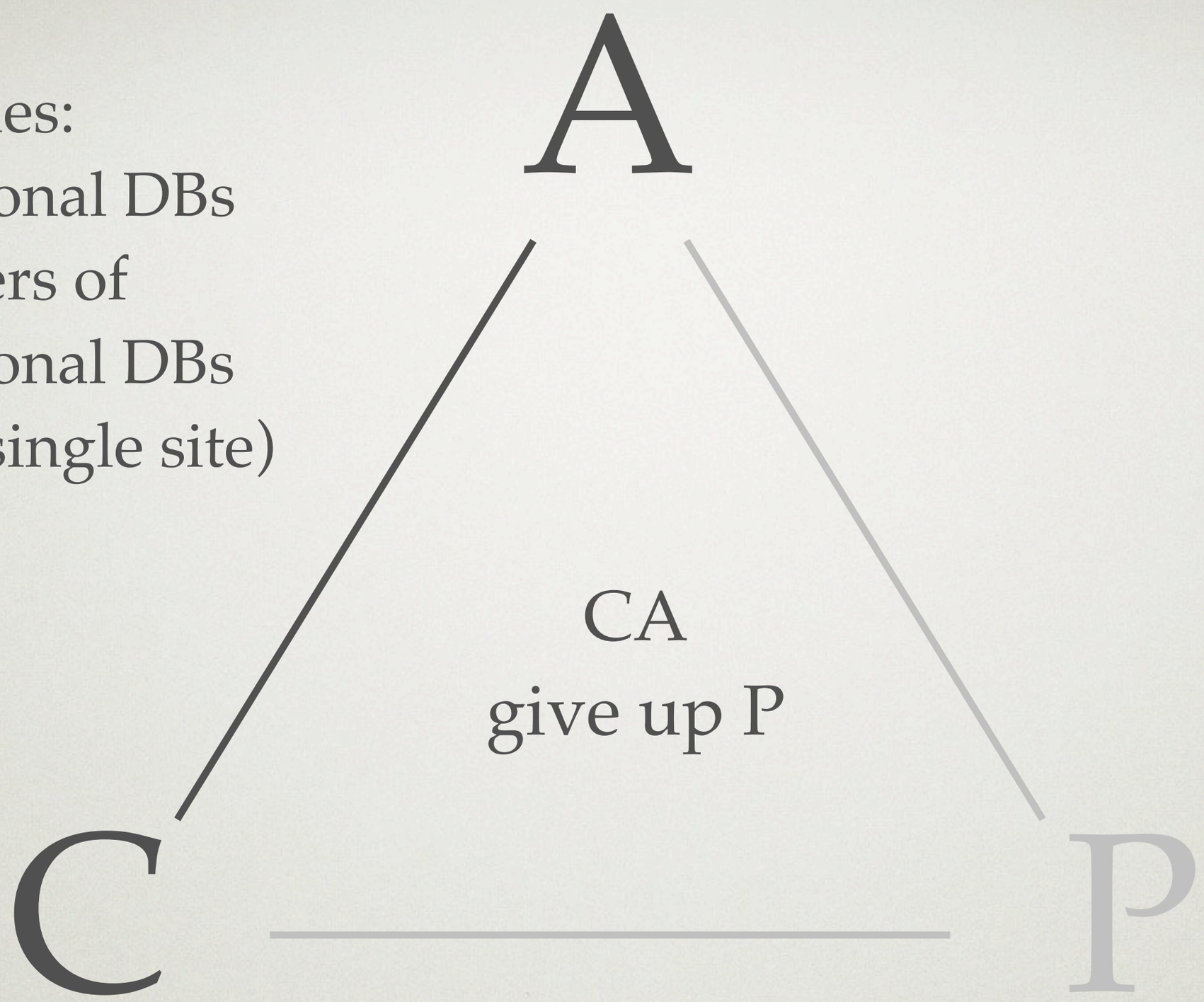
`This is a great article --> http://www.julianbrowne.com/article/viewer/brewers-cap-theorem`

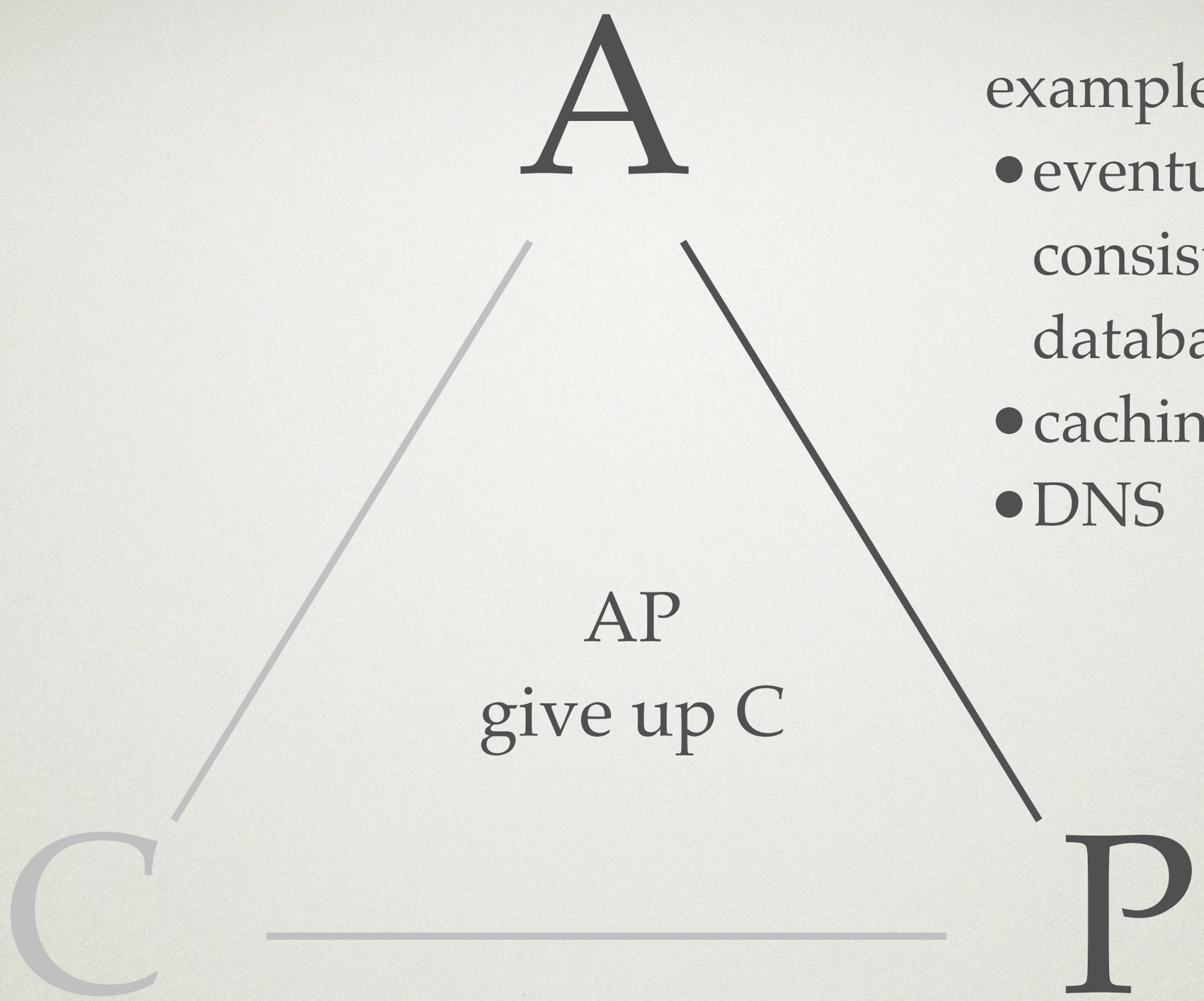


inspired by: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
http://blog.nahurst.com/visual-guide-to-nosql-systems

examples:

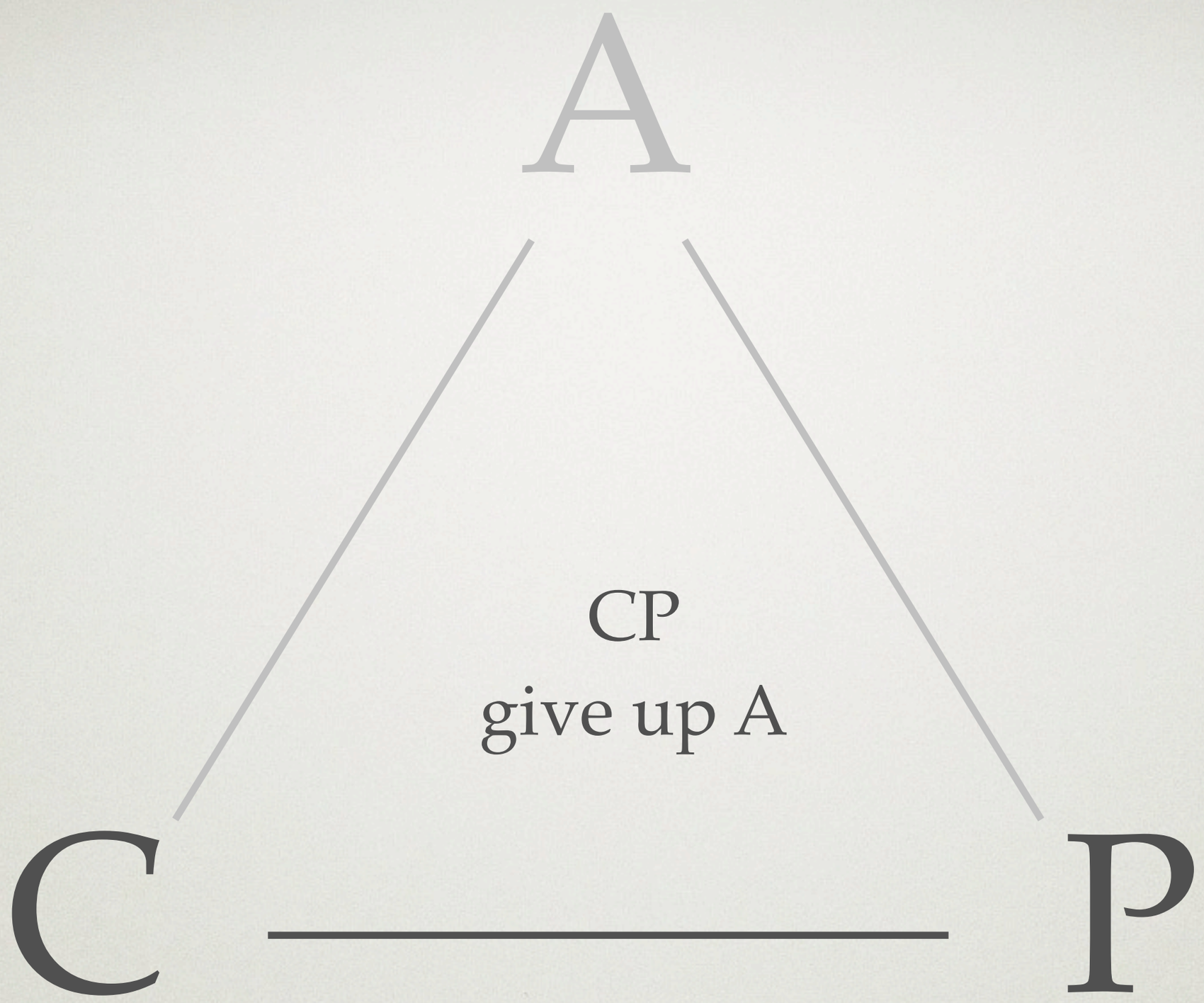
- relational DBs
- clusters of relational DBs (at a single site)





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 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

RELATIONAL DATABASES (RDBMS)

a database which implements the relational model

(thank you, tautology)

what is the relational model?

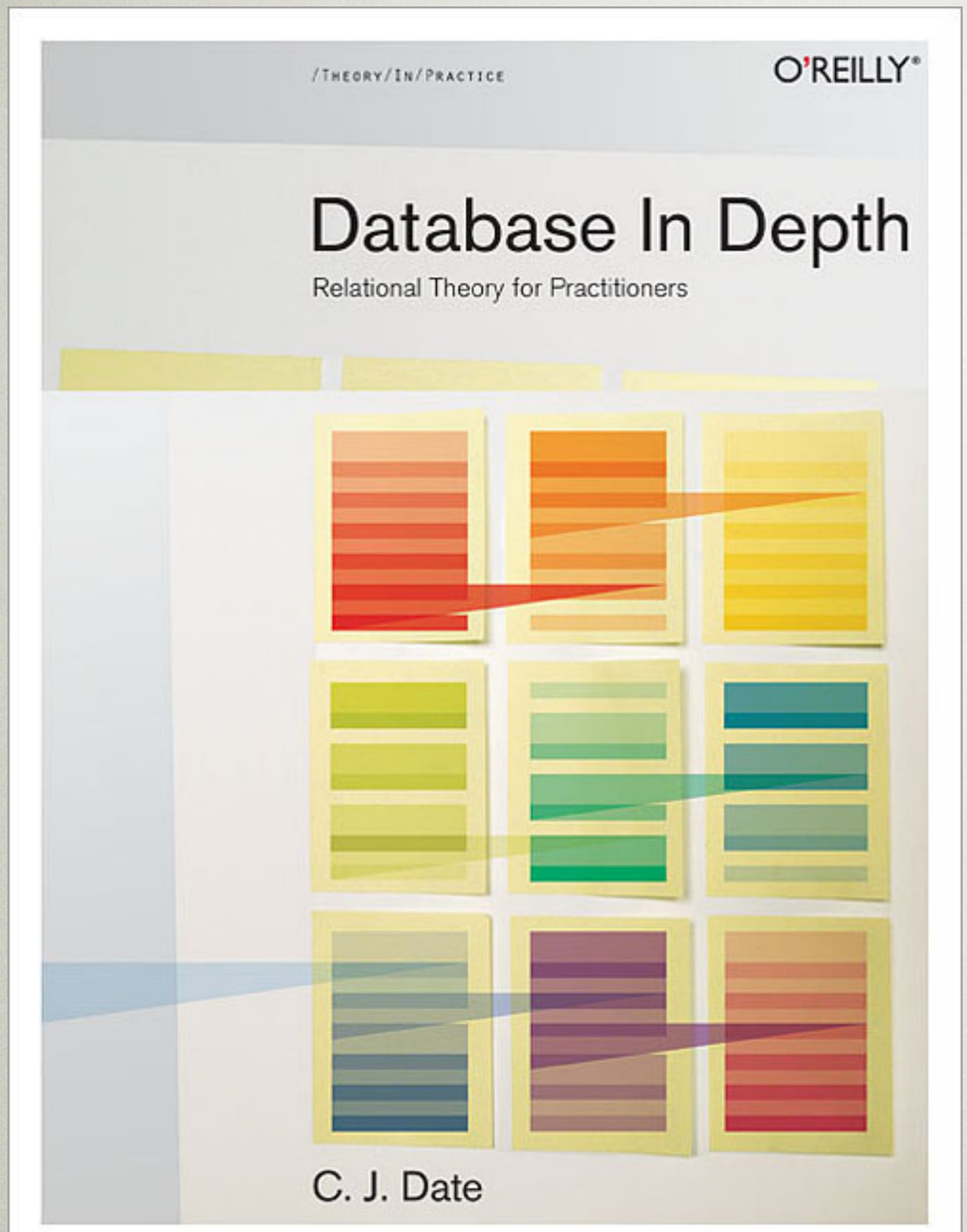
Relational Database

(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.

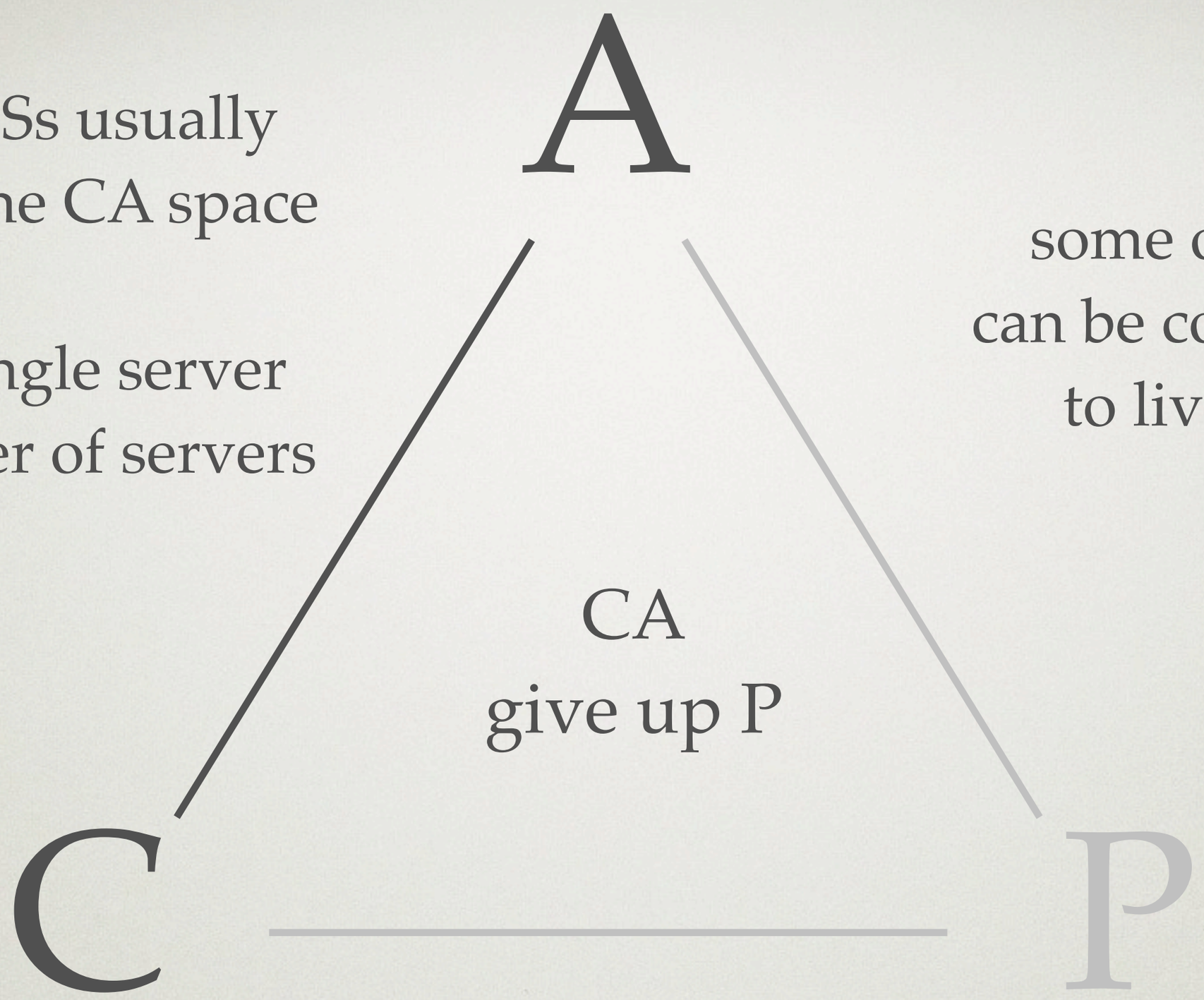
    - Relational engine — Relational objects such as Table, Index, and Referential integrity constraints are implemented in this component.

    - 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

tables

views

constraints

primary key

foreign keys

indexes

stored procedures

other terms?

relational implementations

- SQLite
- MySQL
- PostgreSQL
- many, many others

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/

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 - weak CLI, 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) and JSON datatypes, 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/

Holy Grail?

- various in-memory distributed key-value DBs with SQL query layers
- Google F1 paper released *today*.
- others?

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

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, databases in general)

"...; **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)

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 its' models

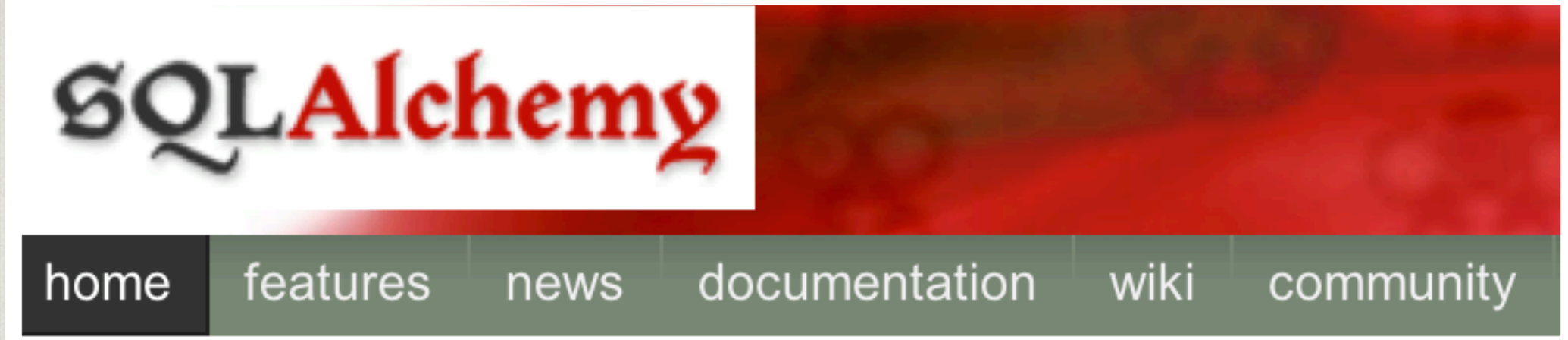
if you can't use the models,
maybe Django isn't the right
tool for the job

SQLAlchemy

http://www.sqlalchemy.org/

if you need to connect to an existing database,

look into SQLAlchemy



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)

Lightning Talks (10)

Break

(5)

Lab A

(20)

Lab A

- no lab A today

Something in the schedule had to give...

Lecture B

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)

“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

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

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/`

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/

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/

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!

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.

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

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

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?

a biased, short list

start at the top, work down

- PostgreSQL
- Redis
- Amazon DynamoDB
- MongoDB
- Cassandra

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)

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?

Redis interlude

These slides inspired by:

Redis Tutorial
by Simon Willison
(2009)

http://simonwillison.net/static/2010/redis-tutorial/

What is Redis?

in-memory
non-blocking (evented) I/O
key / value server
data structures as values
very, very fast

it's like a dictionary
in another process

remote access
multiple clients

persistence

changelog

or

background snapshots

string (int)

hash

list

set

sorted set

publish / subscribe

all keys:

EXISTS

DEL

TYPE

EXPIRE

EXPIRE

allows use as a cache server (similar to memcache)

strings

GET

SET

MGET

MSET

INCR

DECR

bit operations

use for?

key / value cache
key / value storage
counters & stats
(high writes, OK)
web server sessions, tokens, etc
persistent state for interpreter
shared state for interpreter

https://github.com/peta/redis-natives-py

*NX

atomic primitives

distributed systems, etc

lists

doubly linked list

O(1) insertion anywhere

fast push/pop at either end

lists

RPUSH / LPUSH

RPOP / LPOP

LLEN

LRANGE

(range start/end inclusive!)

lists

blocking operators!
atomic!

BLPOP
BRPOP

uses

capped logs (trim range)

queues

worker task distribution

hashes

another level of key / value

all values are strings

(with INCR commands)

sets

SCARD

SISMEMBER

SUNION

SDIFF

SINTER

uses

keep sets of other redis keys for querying values, etc

sorted sets

sets, but each entry also
has a numeric score

whole set is always
sorted by this score

range commands

uses

high score tables

activity lists

timeseries (date is score)

actually, this is still
just the basics

replication
scripting
bit operations
clever data structure uses

redis-benchmark -q

demo

Redis Lab (20)

Redis Lab

WRAPUP

Assignment

- Last week, no assignment!