Database and python

David Cournapeau (cournape@gmail.com)

Database for the scientist

3rd March 2012

What is this talk about

- An introduction to database for scientists:
  - Why would one use database ? Why not ?
  - What does ACID, NoSQL, relational means, and why should I care ? How can I make a choice ?
  - Have an idea of what's available in python to deal with databases
- This talk is NOT about scaling architectures, advanced usage of database, etc...

My Background

- PhD in Kyoto University in machine learning (speech recognition)
- Numpy/scipy contributor since 2007
- Silveregg: recommendation engine for 3rd party websites in Japan
- Enthought since last year

Databases: a few possible definitions

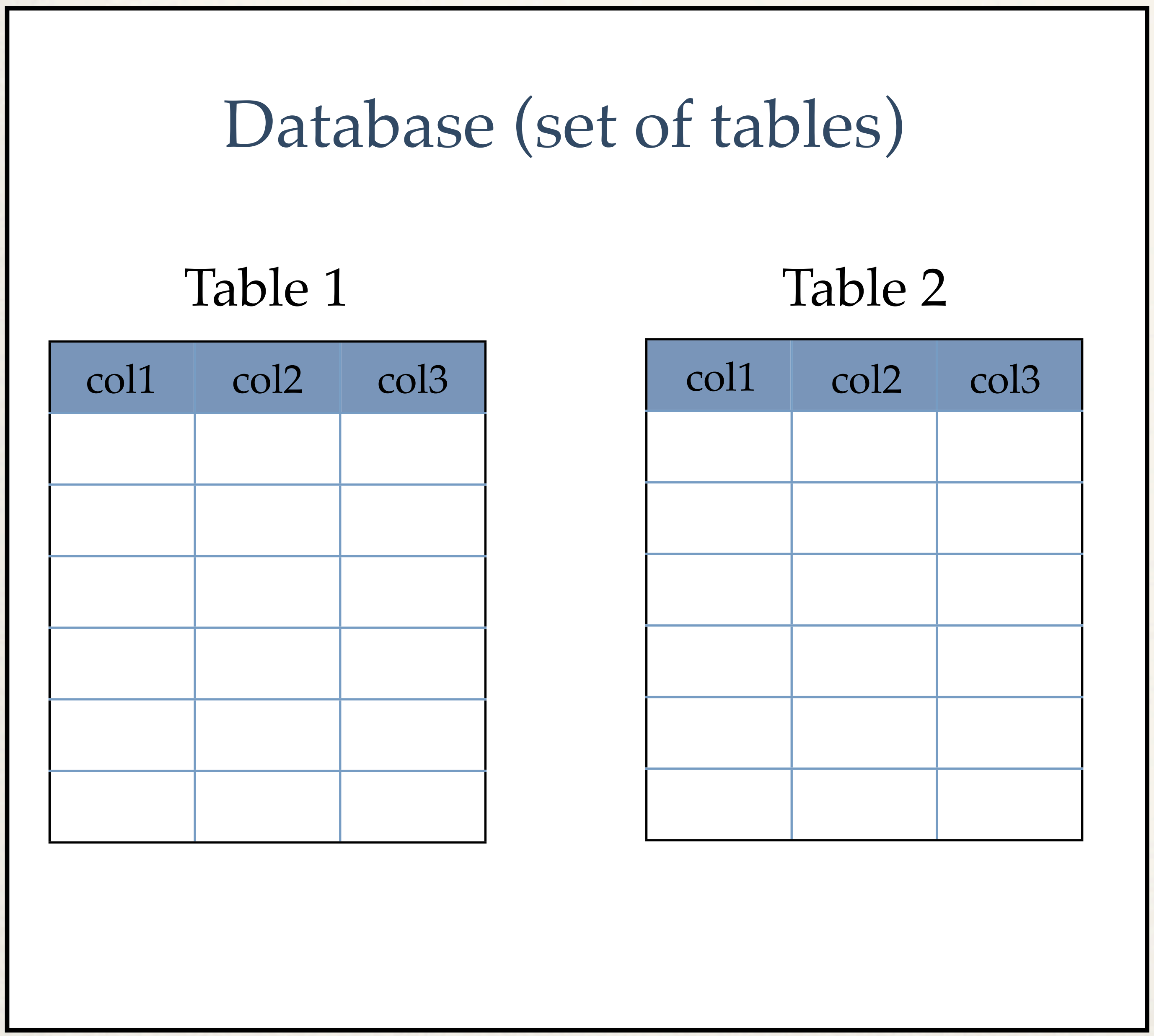
- Wikipedia: “A database is an organised collection of data for one or more purposes, usually in digital form.”
- Numpy arrays are a database ?
- CSV files are a database ?
- MySQL/PostgreSQL/Oracles are databases:
  - because of querying language ?
  - because of storage ?

SQL Databases

SQL Databases

- Structured Query Language (SQL) databases:
  - open source: MySQL, PostgreSQL
  - proprietary: Oracle, MS SQL Server
  - embedded: sqlite (open source, included in python >= 2.5)
- SQL is a programming language:
  - Set-like querying of the data (relational algebra)
  - Data Definition Language (DDL): creating data description (table, etc...)

Hierarchical organisation



- A database is a container of tables
- Each table has a set of columns ("types")
- Each table has a set of rows: every row has the same columns

Creating table and data: example

- IPython notebook: example 1
- Exercises:
  - add new cities
  - select cities which population is above 1 million
  - Change state abbreviations (e.g. "Tex." to TX)

A few comments about raw SQL

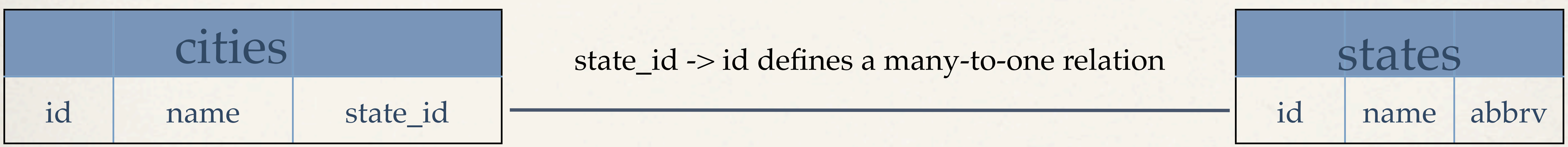
- We used “raw sql”: bad practice (SQL injection risk, maintenance)
- One should use bound parameters instead:
  - we will sometimes continue using raw SQL to save screen space, though

Relationship

- SQL Database are also called relational databases:
  - tables have relations that link data together
  - relations are defined by a mapping between a set of columns from table 1 to a set of columns of table 2
    - mapping can be implicit
    - or explicit (foreign key)
  - different types of relations: one-to-one, one-to-many, etc...

Example

- A city has exactly one state, a state has multiple cities



- Querying cities and states together requires joining the tables:

```sql
SELECT cities.name, states.name
FROM cities, states
WHERE cities.state_id = states.id
```

Exercises

- IPython notebook: example 2
- Exercises:
    - Add the code to import cities and states
    - Change state abbreviation, e.g. 'Tex.' -> 'TX'
    - Compare to changing abbreviation in example 1 ?

Data normalisation

- Relational algebra is a key theory behind relational databases (Cobb, 1970).
- Requires modelling data in a normalised form to be used effectively
  - multiple levels of normal forms (level 1, level 2, etc...)
  - formalise “good practices”
  - Applied correctly, allows for unforeseen queries with reasonable performances, avoids inconsistencies, etc...
- First exercise: table not normalised (repetition of state abbreviation)

Referential constraints

- SQL databases can enforce constraints on data:
  - Encoding relationship through foreign key
  - Enforcing non-null value, uniqueness of column, etc...

Example

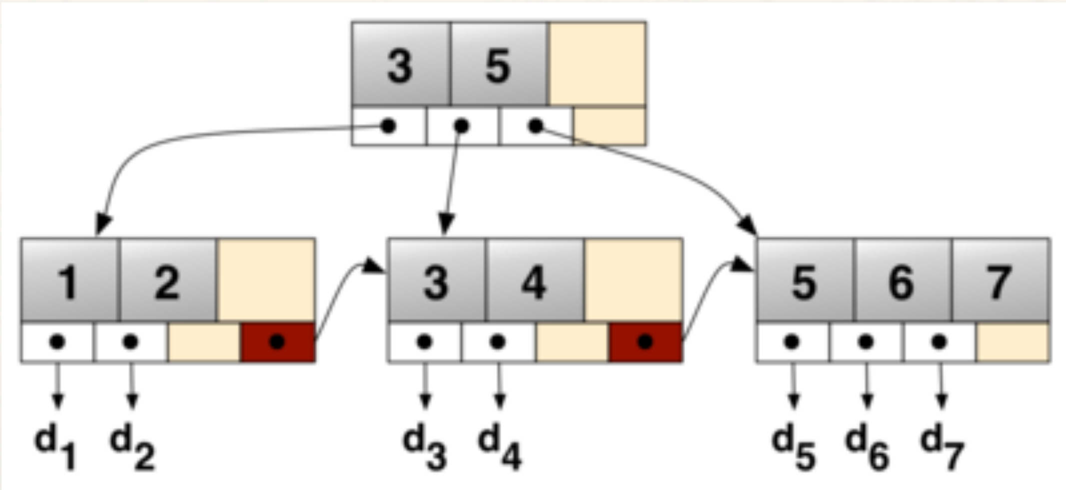
- IPython notebook: “sql - ex 3”

Databases indexing

- Relational model abstracts querying from on-disk format
  - must maintain reasonable performance for insertion, selection, deletion, etc...
- Without index: every search would require full table scan
- Index takes size, and reduces writer performance !
- Extremely simplified: based on B+-Trees (“binary search trees with large fanout”)

B+-tree

- Each node contains N “pointers”
- Only leaves have values
- Insertion guaranteed to be O(log(k)) where k is the tree’s height
- N values consecutive on disk (minimises random I/O)



Thinking back a bit

- Compared to e.g. numpy arrays:
  - multiple datasets can be related to each other
  - minimise chances of inconsistencies (constraints)
  - set-based querying language on multiple attributes
  - low-level data layout separated from access patterns
- The cons should be obvious to scientists !

SQLAlchemy: a python DB toolkit

What is SQLAlchemy

- SQLAlchemy (SQLA) is a python library to deal with databases:
    - Enables to reduce impedance mismatch between python and SQL
    - Allows to deal with multiple databases in a portable way
- The main SQLA library is divided in two parts:
    - SQLA-core (expression language): reduces python/SQL mismatch
    - SQLA-orm: higher level abstraction

Revisiting sql through SQLA

- SQLA Exercise 1:
  - copy/execute the given notebook
  - Rewrite our first exercises using SQLAlchemy
- Those exercises use only SQLA core

SQLA - ORM

- Object Relational Mapping: “mapping rows to objects”
- Declare object mapping / table in one shot (since 0.7)

Given by SQLA

```
class City(Base):
    __tablename__ = "city"

    id = Column("id", Integer, primary_key=True)
    name = Column("name", String(30))
    population = Column("population", Integer)

    def __repr__(self):
        return "City(name={0}, population={1})".format(
                                                self.name, self.population)
```

Exercise

- ipython notebook, sqla exercise 2/3
- Few tips:
  - try with echo=True to see the generated sql
  - think about the implication of lazy loading in terms of performance (tip: look at generated SQL)

Thinking back a bit

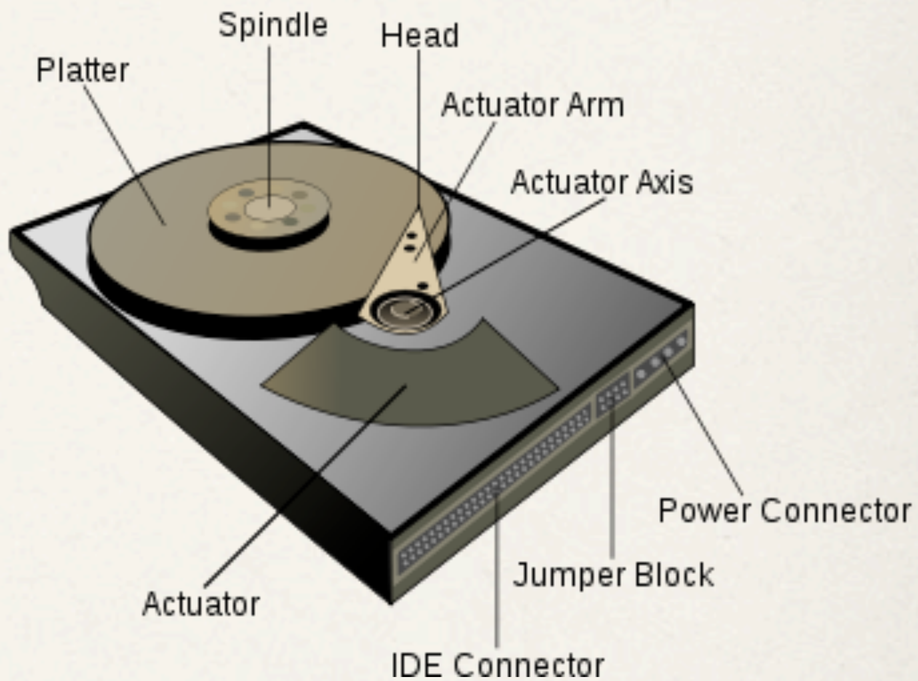
- SQLAlchemy is more than an ORM
- Used properly, SQLA core does not have impact on SQL performance
  - "SQL databases behave less like object collections the more size and performance start to matter; object collections behave less like tables and rows the more abstraction starts to matter"

SQL Databases

aka writing to disk is hard

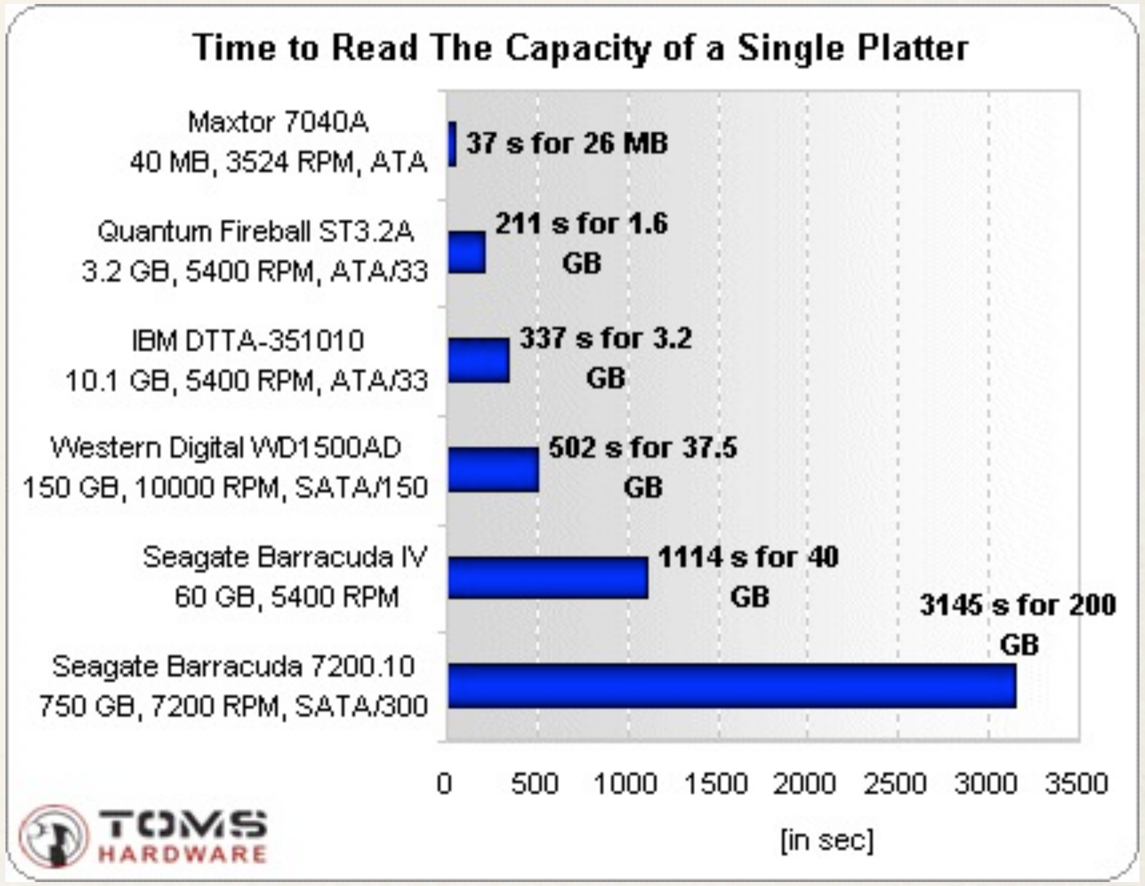
What's a hard drive

- Hard drives are mechanical:
  - hard drive has multiple platters
  - writing means a head has to move
  - throughput not consistent (angular speed is)
  - disks are bigger and bigger, heads are not getting (much) faster



A few numbers to keep in mind

- Areal density (~capacity) increases by several orders of magnitude (1990: ~100 Mb, 2010: ~1Tb)
- Speed increases (~1 Mb/sec, ~50 Mb / sec) not as significant
- Reading a typical, full HD from 2012 takes ~half of a day
- Access time is still ±few ms



Source: Toms hardware

- Hard drives are slow
- Hard drives are a PITA

Writing to hard drive is hard

- Hard drives are slow
    - Operating Systems use a lot of tricks to hide this (buffer, write cache)
    - Hard drives controllers use tricks as well
    - What happens when hardware crashes ?
- Database's job is to "untrick" the trickster

How your DB sees your OS

Transactions

- Transaction is a significant feature of SQL databases:
  - every commit == transaction == has to really write to the disk (fsync)
  - Writing 1000 rows in one transaction is different than writing 1000 rows one row at a time !
  - Transaction Per Second (TPS): this is not your throughput ! (beware about numbers)

SQL databases and ACID

- Atomicity, Consistency, Isolation, Durability
  - Atomicity: operation are done or not done (not halfway)
  - Consistency: read the same value after write is acknowledged
  - Isolation: transactions don’t interfere with each other
  - Durability: once the data is written, it is there “forever”

ACID (2)

- ACID helps reasoning about a system with multiple readers/writers
- This is where numpy/pandas/flat files “fail”
- But ACID is not absolute !

Durability you said ?

‘NOSQL’ databases

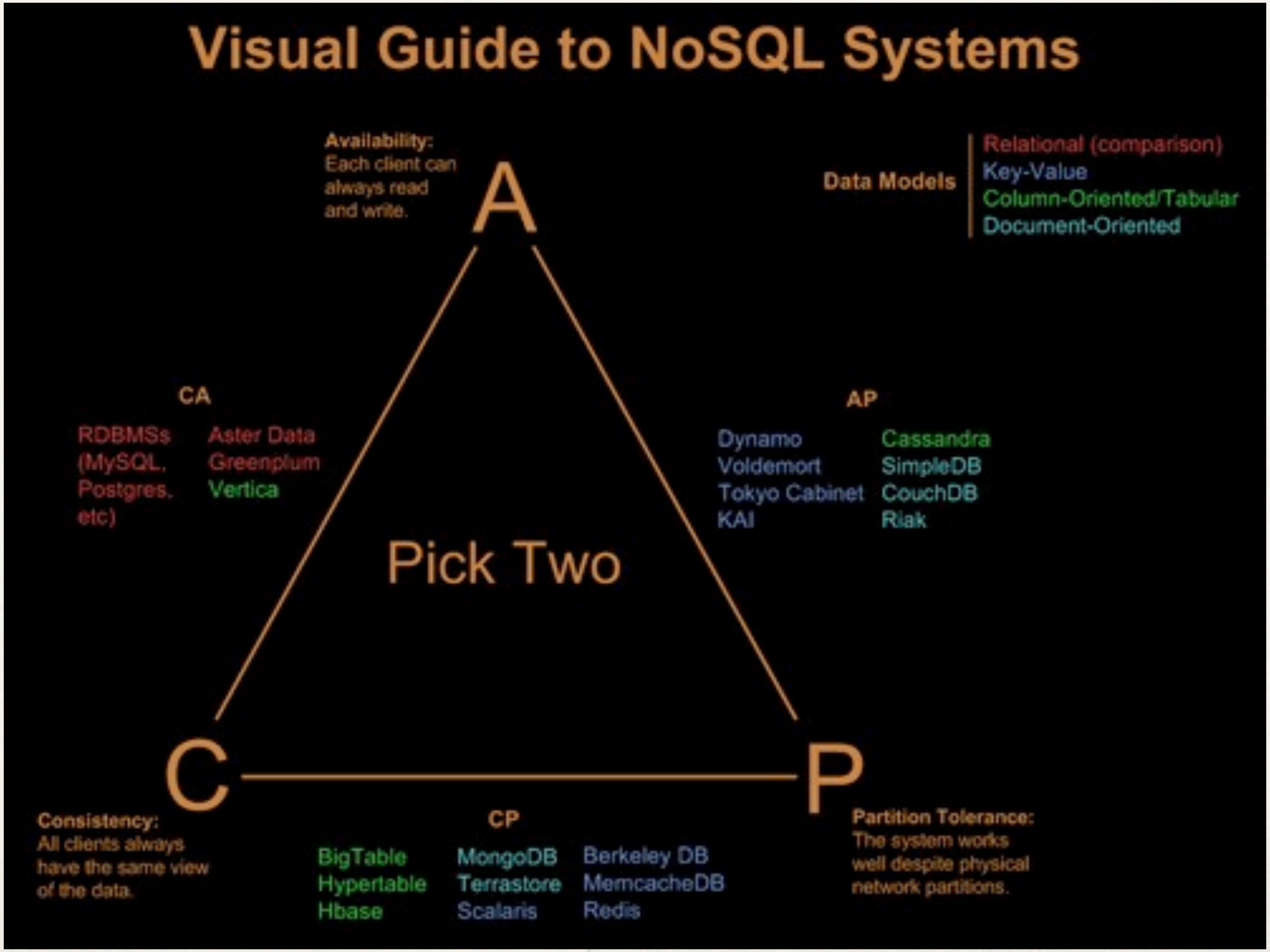
A bit of history

- First time: around 1998
- NoSQL may mean:
  - non-relational (no join) ?
  - non-acid ?
  - schema-less (each "row" has a different set of attributes) ?
- Often advertised scalability, better performance, easier to use, etc...

CAP Theorem

- Distributed data system ("Brewer theorem", early 2000)
  - Consistent: read back what you wrote
  - Available: always get a response from the system
  - Partition tolerance: system can work when system is partitioned
- You can't have three, you have to pick 2

Different usages, different tradeoffs



Redis

- Redis is a persistent, “mostly in memory”, key-value store:
    - keys are (binary-safe) strings
    - values can be strings, list, sorted sets
- Commonly referred as a data-structure server
- Relatively fast: ±10k operations/sec on decent machine

Simple example

- Strings, increment, list, set: redis ex 1 notebook

Good use cases for redis

- Keeping a list of statistics from multiple processes/machines
  - cpu % per server per day
  - keep track of cpu distribution
- Handling time series (logs, etc...)
- Recent versions of redis: pub/sub, key TTL, etc...

Mongodb

- Document databases ('schemaless')
  - each "row" is a document
  - document ~ a json files (~ recursive python dict)
- Many "advanced" features: indexes, automatic sharding, performance, etc...
- Loose on data integrity (writes not acknowledged, no single-server durability by default, etc...)

Example

- Ipython notebook “mongodb - ex 1”
  - schema-less: pros/cons
  - simple select/where-like queries

Thinking back a bit

- Inserting documents can be done in any key-value store:
    - e.g. pickling your object into any SQL db
    - Many SQL DB supports XML, some json
- Real power of a document store: create indexes on those attributes

NoSQL is the new cool kid ?

A few cautionary words

NoSQL technologies are not new

- NoSQL is not new:
  - BekerkelyDB (late 70ies) is key-value store
  - Lotus Note is a document store (has it ever be cool ?)
- Typical advertised advantages of NoSQL:
  - scalability
  - speed
  - easy of use

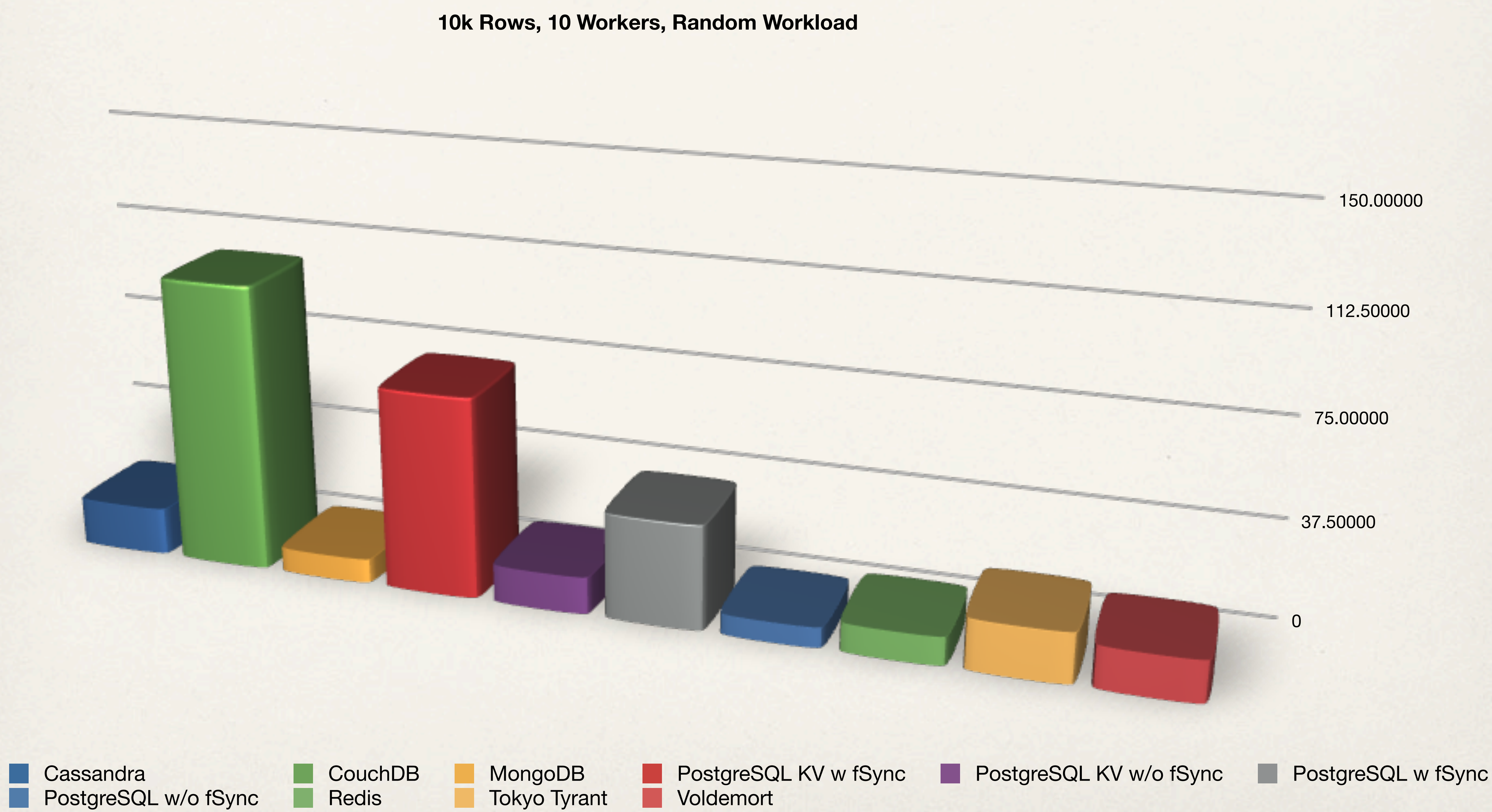
A few words about speed, scalability

- Speed != scalability:
  - speed: how fast you are with the current environment
  - scalability: how much faster you can be by adding resources
  - Example: Map/Reduce is not fast, but is (can be) scalable
- Beware of benchmarks !

A few words about speed, scalability (2)

- Example: insertion speed, > 10000 insertions /sec ?
  - "normal" hard drives cannot possibly write 10000 / sec
  - what happens when data is bigger than RAM ?
    - what happens when indexes are bigger than RAM ?
  - SQL databases can also have their durability tweaked (e.g. MySQL 'innodb_flush_log_at_trx_commit')
- Adwords was using MySQL in 2002 (still is ?)
- Facebook is still massively using MySQL

A few words about speed, scalability (3)



A few words about speed, scalability (4)

... I believe the relational data model is the "right" way to structure most of the data for an application like Quora ... Schemas allow the data to persist in a typed manner across lots of new versions of the application as it's developed, they serve as documentation, and prevent a lot of bugs. And SQL lets you move the computation to the data as necessary rather than having to fetch a ton of data and post-process it in the application everywhere...

Adam D'Angelo (Quora founder, former CTO of facebook)

http://www.quora.com/Quora-Infrastructure/Why-does-Quora-use-MySQL-as-the-data-store-instead-of-NoSQLs-such-as-Cassandra-MongoDB-CouchDB-etc

A few words about profiling

- Performance issues with DB (sql or nosql): profile !
- Linux has good IO profiling capabilities:
  - iotop (pre process IO)
  - htop
  - think in terms of statistics, not just average (scores, etc...)

Conclusion

- Databases can be your friend:
  - use them for well defined data with constraints
  - use them for multiple readers/multiple writers systems
  - You can access most of them from python
- Different systems have different tradeoffs:
  - be careful about benchmarks
  - CPU is cheap, IO is expensive: tradeoffs are different from 20 years ago !