Relational

vs.

Non-Relational

Josh Berkus
PostgreSQL Experts Inc.
Open Source Bridge 2010

PGX
POSTGRESQL
EXPERTS, INC.

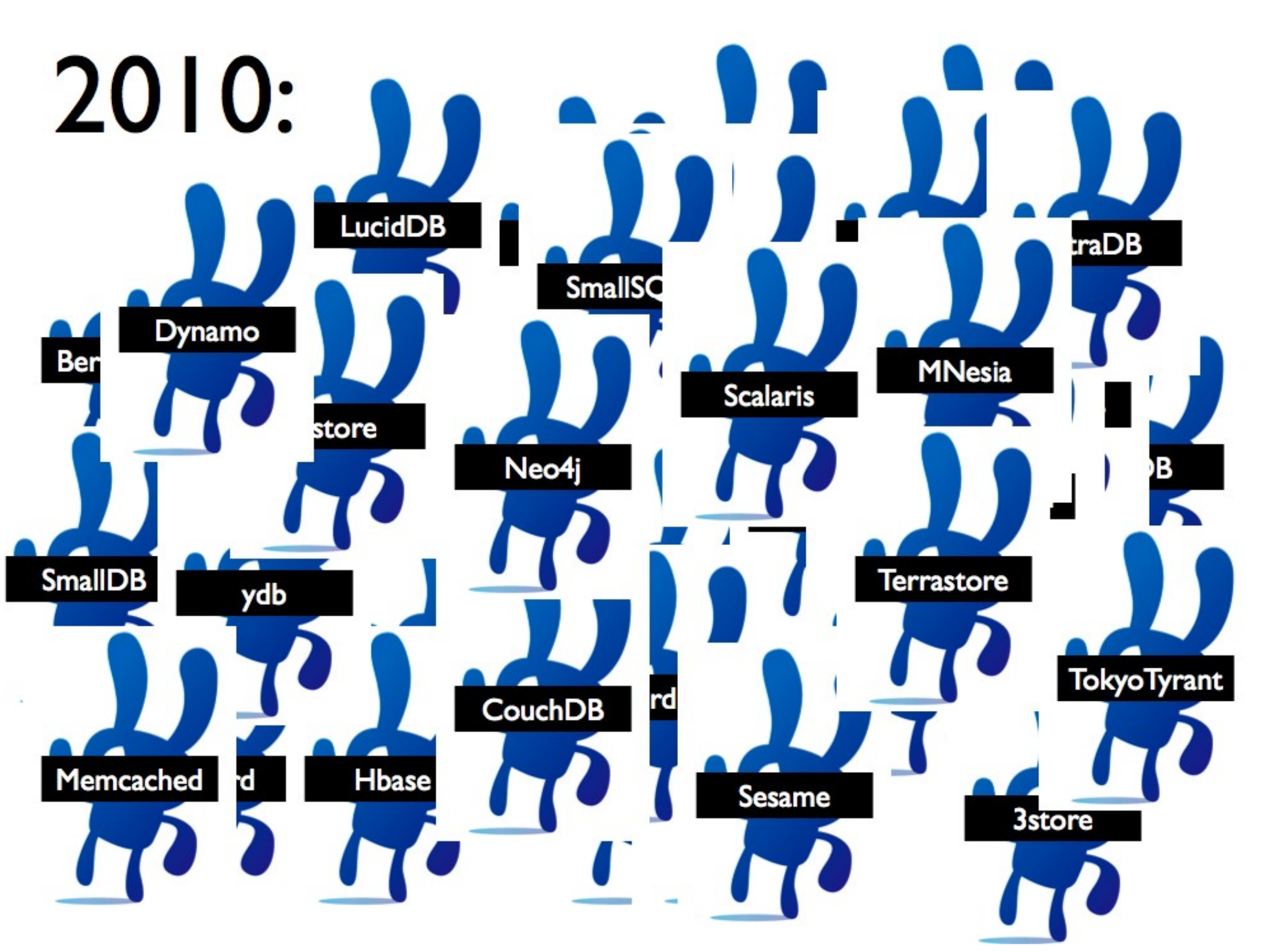
2003:

- MySQL
- PostgreSQL
- FireBird
- BerkeleyDB
- Derby
- HSQLDB
- SQLite

2003:

- **MySQL**
- **PostgreSQL**
- **FireBird**
- BerkeleyDB
- **Derby**
- **HSQLDB**
- **SQLite**

Postgres vs. MySQL


2010:
LucidDB
Dynamo
store
Neo4j
Scalaris
MNesia
SmallDB
ydb
Terrastore
CouchDB
TokyoTyrant
Memcached
Hbase
Sesame
3store

“NoSQL movement”

All non-relational databases

Are *not*
the same

Graph

Neo4J
HyperGraphDB
Jena

Document

CouchDB
BerkeleyDB-XML
Solr

Key-value

Memcached
Tokyo Cabinet
db4o
RIAK

Distributed

Cassandra
Hypertable
MySQL NDB

Hierarchical

MongoDB

All relational databases

Are *not*
the same

Embedded

SQLite
Firebird
HSQL

OLTP

PostgreSQL
MySQL
Oracle
SQL Server

MPP

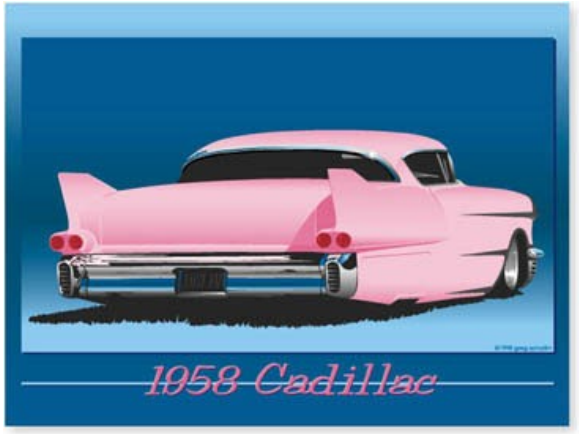
TeraData
Greenplum
Aster

Streaming

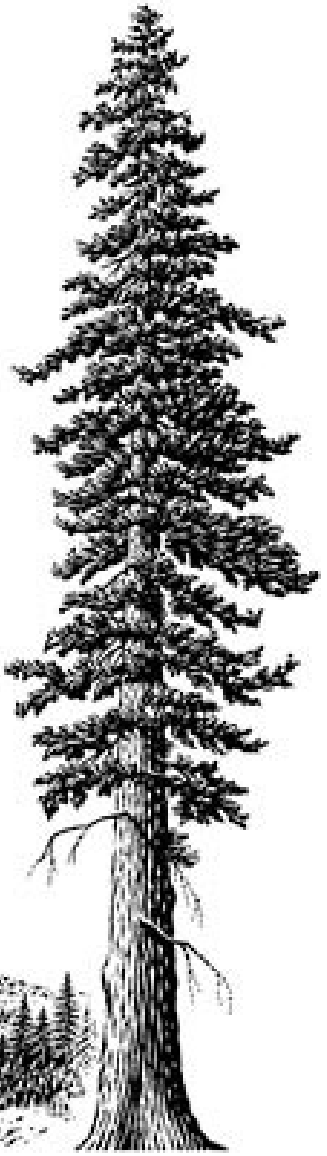
Streambase
Truviso

C-Store

LucidDB
MonetDB



NoFins




MYTHBUSTERS

“NoSQL movement”

Mythbust #2

“revolutionary”

There

are

no

new

database

designs

There are only new
implementations
and
combinations

“A database storing application-friendly formatted objects, each containing collections of attributes which can be searched through a document ID, or the creation of ad-hoc indexes as needed by the application.”

CouchDB, 2007

Pick, 1965

CouchDB, 2007

embeddable Pick
JSON storage
REST API
map/reduce

“revolutionary”

“ evolutionary”

“renaissance
of
non-relational
databases”

Mythbust #3

“non-relational databases are toys”

Google

Bigtable

Amazon

Dynamo

FaceBook

Memcached

US Vetrans' Administration

Pick, Caché

Mythbust #4

“Relational databases will become obsolete”

“Three decades past, the relational empire conquered the hierarchical hegemony. Today, an upstart challenges the relational empire's dominance ...”

XML Databases 2001

--Philip Wadler, Keynote
VLDB, Rome, September 2001

Anyone remember XML databases?

No?

What happened?

established relational
and non-relational
databases
hybridized XML

Oracle XML
PostgreSQL XML2
BerkeleyDB XML
DB2

Mythbust #5

“Relational databases are for when you need ACID transactions.”

Transactions

≠

Relational

Robust Transactions without Relationality:

BerkeleyDB
Amazon Dynamo

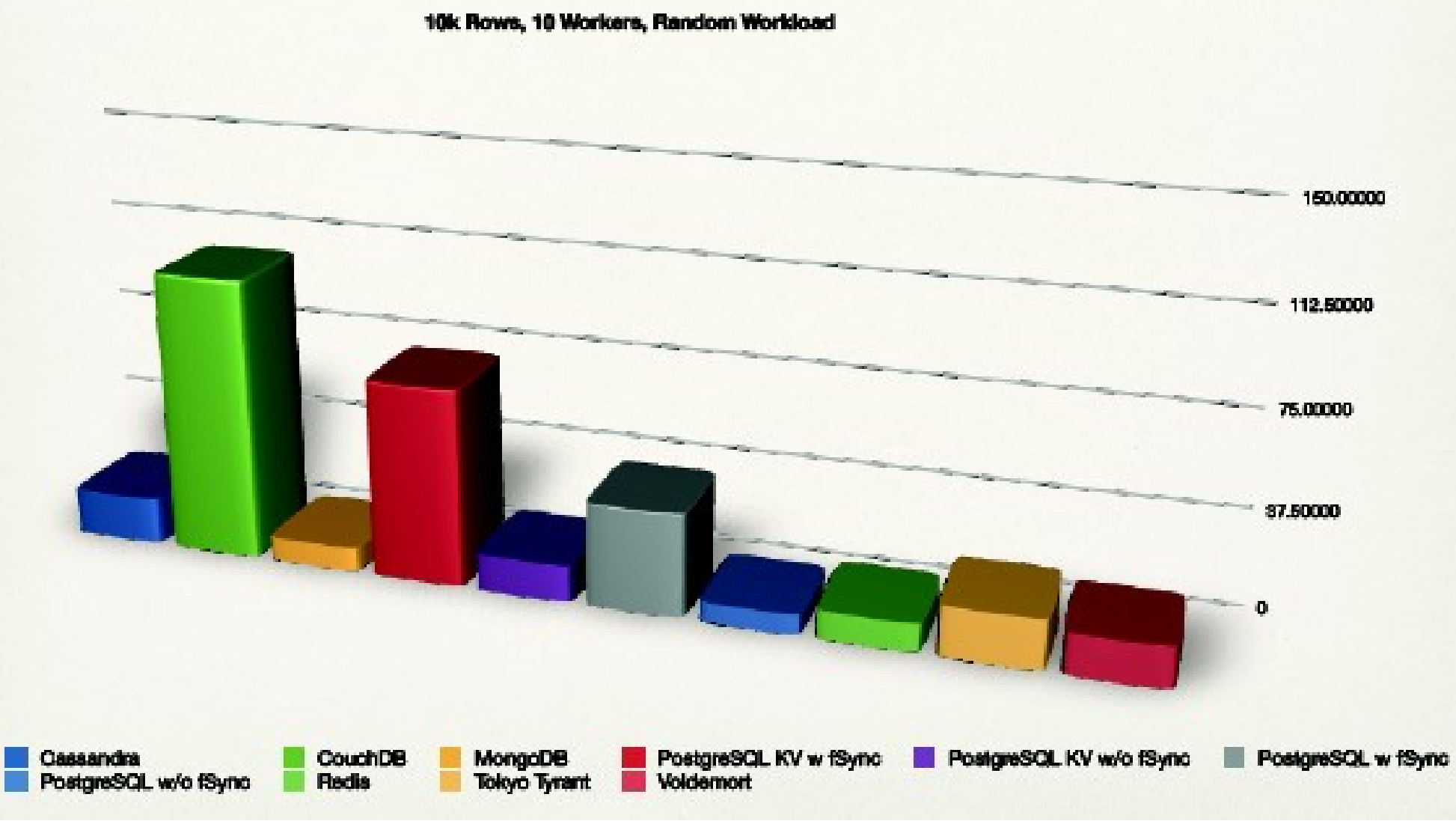
SQL Without Transactions:

MySQL MyISAM
MS Access

Mythbust #6

“Users are adopting noSQL for web-scale performance”

KVPBench Random Workload



Horizontal Scalability

Database	Difficulty	Scalability	Redundancy
Memcached	Very Easy	10 to 50	None
Cassandra	Difficult	100	Sync
MySQL	Easy	12	Async
Posgres+Skytools	Difficult	20-50	Both
MySQL NDB	Difficult	20-50	Sync
CouchDB	Easy	2 to 6	Async
MongoDB	Moderate	2 to 6	Async
Redis	Easy	2 to 6	Async

Note: data in the above chart is extremely dated. Some databases were tested over a year ago.

Mythbust #7

You

do

not

have

to choose

one database.

Choose
the database system
which fits your
current
application goals.
or ...

Use more than one together

MySQL + Memcached

PostgreSQL + CouchDB

or ...

Use a Hybrid

MySQL NDB

PostgreSQL Hstore

HadoopDB

But what about relational vs non-relational?

Relational OLTP Databases*

- Transactions: more mature support
  - including multi-statement
- Constraints: enforce data rules absolutely
- Consistency: enforce structure 100%
- Complex reporting: keep management happy!
- Vertical scaling (but not horizontal)

** mature ones, anyway*

SQL vs. Not SQL

SQL promotes:

- portability
- managed changes over time
- multi-application access
- many mature tools

SQL vs. Not SQL

But …

SQL is a full programming language, and you have to learn it to use it.

SQL vs. Not SQL

No-SQL allows:

- programmers as DBAs
- no impedance
- fast interfaces
- fast development

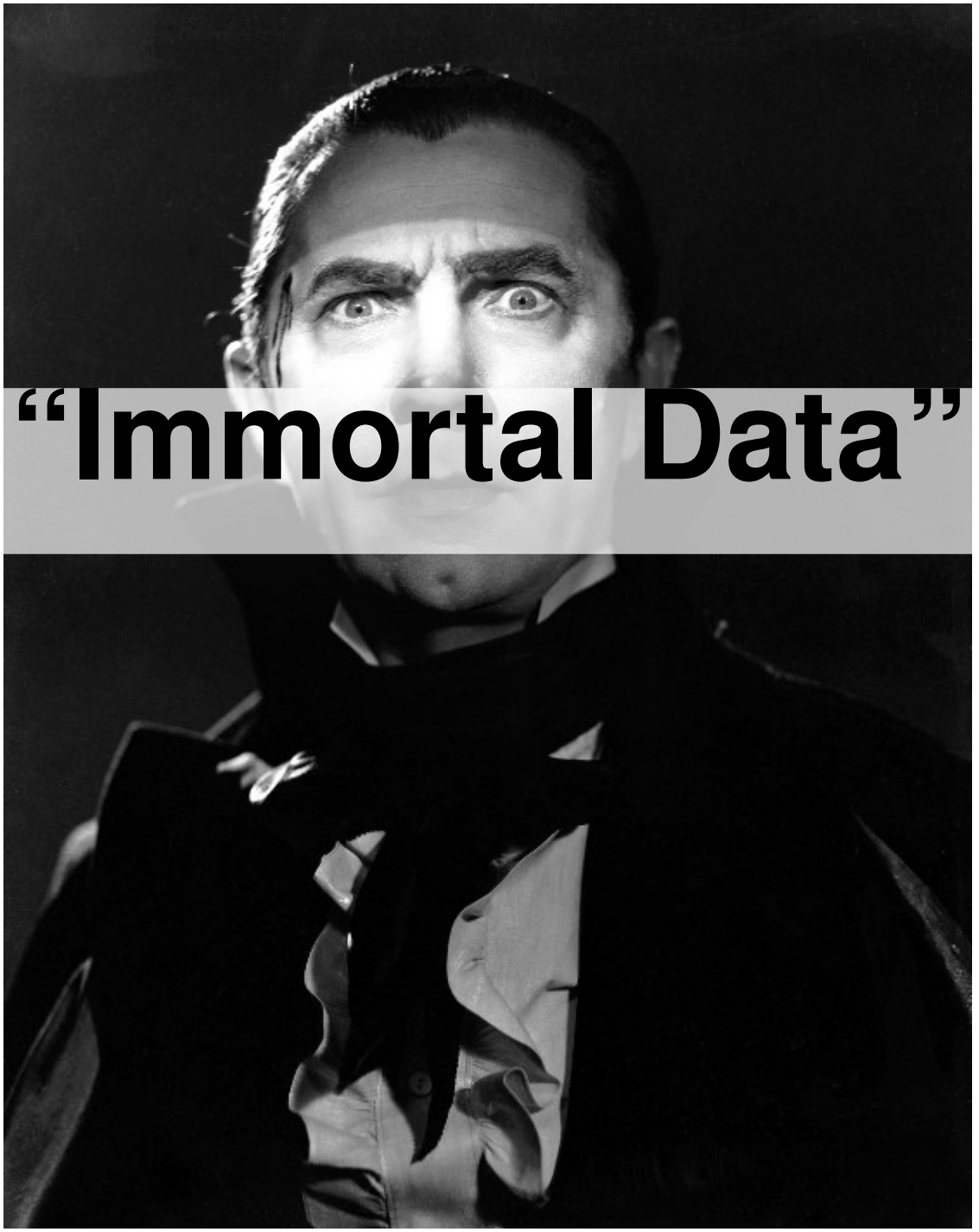
SQL vs. Not SQL

… but:

may involve learning complex proprietary APIs

```
db.things.find({j: {$ne: 3}, k: {$gt: 10}});
```

The main reason to use SQL-RDBMSs


“Immortal Data”

“Immortal Data”

your data has
a life
independent
of this specific
application implementation

How *do* I choose?

Define the problem you are trying to solve

- “I need a database for my blog”
- “I need to add 1000's of objects per second on a low-end device.”
- “I need my database to enforce business rules across several applications.”
- “I want my application to be location-sensitive.”
- “I need to cache data an access it 100K times per second.”
- “I need to produce summary reports across 2TB of data.”
- “I have a few hundred government documents I need to serve on the web and mine”
- “I need to know who-knows-who-knows-who.”
- “I need to data mine 1000's of 30K bug reports per minute.”

Define the features you *actually* need

- many connections
- multi-server scalability
- complex query logic
- APIs
- redundancy
- data integrity
- schema/schemaless
- data mining

fit the database
to the task

“I need a database for my blog”

Use anything!

- MySQL
- PostgreSQL
- MongoDB
- SQLite
- CouchDB
- Flatfiles
- DBaseIII
- Something you wrote yourself

“I need my database to unify several applications and keep them consistent.”

PostgreSQL

“OLTP
SQL-Relational Database”

“It's not just a database: it's a development platform”


Postgres 9
beta out now!

“I need my application to be location-aware.”

PostGIS

“Geographic Relational Database”

PostGIS

- Queries across “contains” “near” “closest”
- Complex geometric map objects
  - polygons
  - lines (roads, etc)

or now … CouchDB Spatial

and Spatialite!

“I need to store 1000's of event objects per second on embedded hardware.”

db4object

“Embedded Key-Value Store”

db4object

“Embedded Key-Value Store”

BerkeleyDB, Redis, TokyoCabinet, MongoDB

db4object

- German Train System
- Insert 1000's of objects per second
- Low-end embedded console computer
- Simple access in native programming language (Java, .NET)

  - *compromise: embedded SQL database: SQLite*

“I need to access 100K objects per second over 1000's of connections.”

memcached

"Distributed In-Memory Key-Value Store"

memcached

- Use: public website
- Used for caching 1000's of serialized objects per second
- Available for 100000's of requests per second across 1000's of connections
- Cache each object only once per site
- *Supplements* a relational database

Alternatives: Redis, KyotoTyrant, etc.

“I need to produce complex summary reports over 2TB of data.”

LucidDB

“Relational Column-Store”

LucidDB

- For reporting and analysis
- Large quantities of data (TB)
- Complex OLAP & analytics queries
- Business intelligence
- *compliments* a transactional database

“I have 100's of
government documents
I need to
serve on the web
and mine for data.”

CouchDB

“Document Store”

CouchDB

1. CividDB Project
2. Storing lots and lots of government documents
3. Don't know what's in them
4. Don't know how they are structured
5. Store them, figure out structure later by data mining.

It's also good for mobile applications!

“I have a social application and I need to know who-knows-who-knows-who-knows-who-knows-who.”

Neo4j

“Graph Database”

Neo4j

- Social Network Website
- 6 degrees of separation
- “you may also like”
- type and degrees of relationship

“I get 1000's of
30K bug reports
per minute
and I need
to mine them for trends.”

Hadoop

"Massively Parallel Data Mine"

Hadoop + HBase

- Massive bug report data feed
- 1000's of bug reports per minute
- Each bug report 2-45K of data
- Need to extract trends and correlate inexact data
- Summarize in daily & weekly reports

Conclusion

- Different database systems do better at different tasks.
  - every database feature is a tradeoff
  - no database can do all things well
- Relational vs. non-relational doesn't matter
  - pick the database(s) for the project or the task

Questions?

- PostgreSQL Project
  - www.postgresql.org
  - josh@postgresql.org
- PostgreSQL Experts
  - www.pgexperts.com
    - www.pgexperts.com/documents.html
- Open Source Database Survey
  - Selena Deckleman
  - Open Source Database Survey: www.ossdbsurvey.org

- **PostgreSQL BOF,** Tonight 7pm

Special thanks to Selena Deckelman for the bunnies image and Gavin Roy for NoSQL performance tests.

PGX
POSTGRESQL EXPERTS, INC.

Relational

vs.

Non-Relational

Josh Berkus
PostgreSQL Experts Inc.
Open Source Bridge 2010

PGX
POSTGRESQL
EXPERTS, INC.

This talk is aimed at helping people to decide what kind of databases they need and to answer a lot of questions around the noSQL, non-relational options.

2003:

- MySQL
- PostgreSQL
- FireBird
- BerkeleyDB
- Derby
- HSQLDB
- SQLite

Back in 2003 the open source database scene was less exciting and had a lot less options.

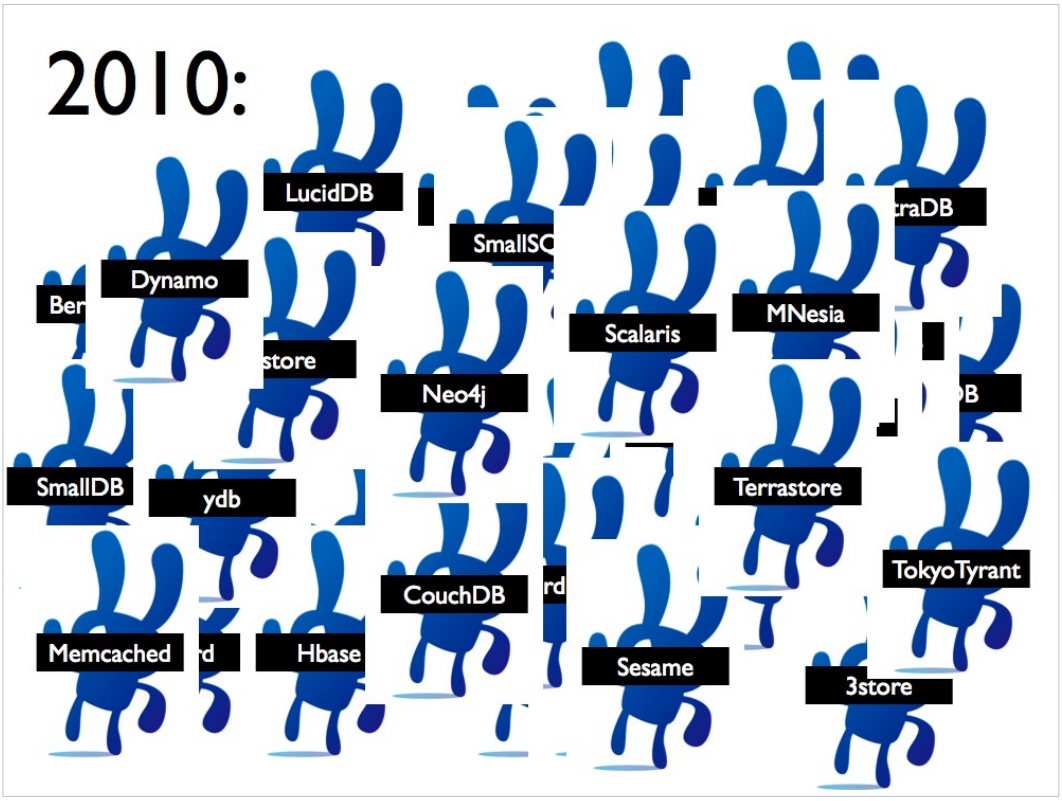
2003:

- **MySQL**
- **PostgreSQL**
- **FireBird**
- BerkeleyDB
- **Derby**
- **HSQLDB**
- **SQLite**

Most of they used SQL interfaces and were relational databases.

Postgres vs. MySQL

The biggest question was postgres vs. mysql.



Today there are more open source databases than anyone has yet tracked. They're popping up like bunnies in the spring.

“NoSQL movement”

So what about the so-called “nosql movement”?

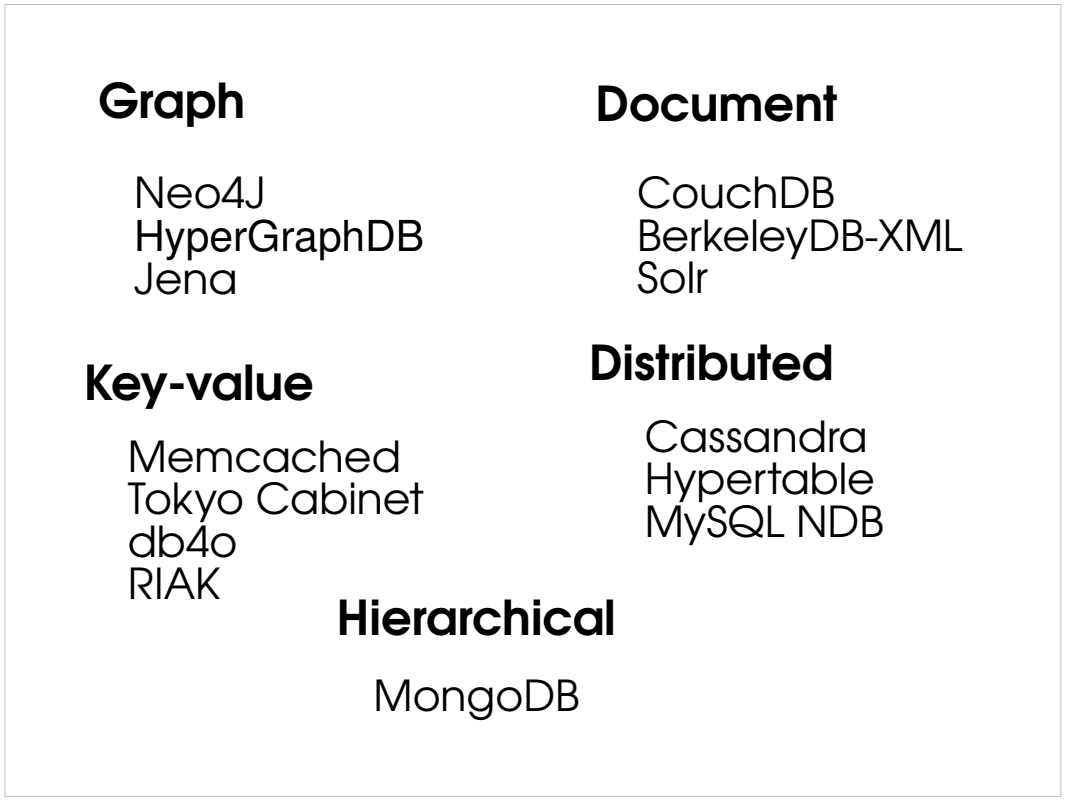
Well, that's a rather misleading bit of marketing.

All non-relational databases

(next several slides)

Just because a database is non-relational doesn't mean it's automatically identical.

Are *not* the same

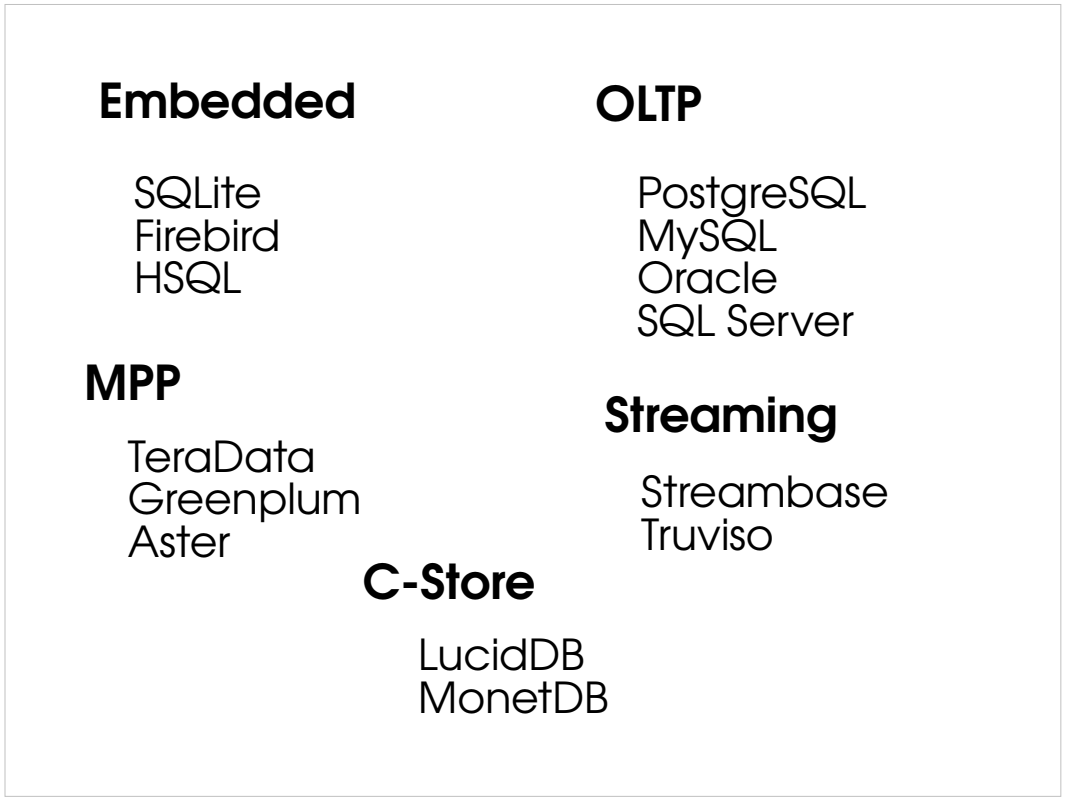


Non-relational databases span a large range of radically different functionalities and architectures. Many of these have very little in common.

All relational databases

Don't assume that all relational databases are identical either.

Are *not*
the same



Relational databases also have a wide range of features, implementations, levels of maturity, and use cases.

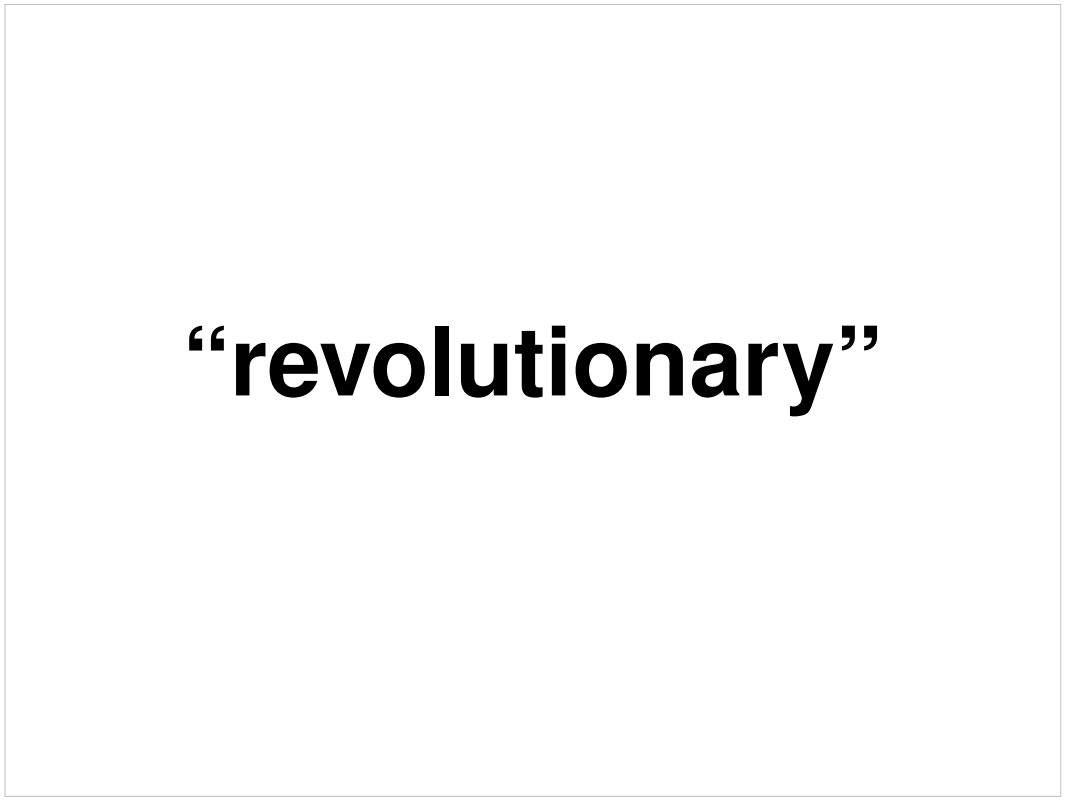
So the whole NoSQL concept is a fallacy; it's like taking a dolphin, Nemo, and a 1958 cadillac on one side, and an octopus, a Prius and a redwood tree on the other, drawing a line between them, and calling the ones on the right “nofins”.

It's not even accurate; several of the new databases already have, or are implementing SQL interfaces.

So as mythbusters would say ...

Myth - Busted!

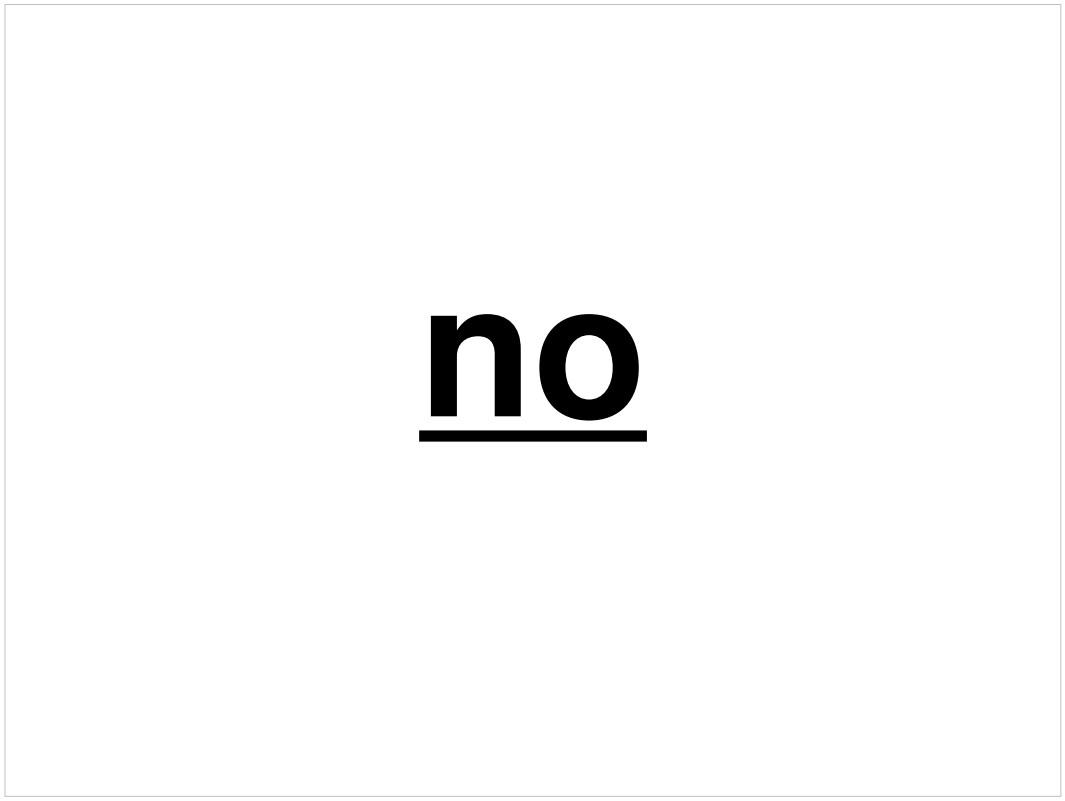
Mythbust #2

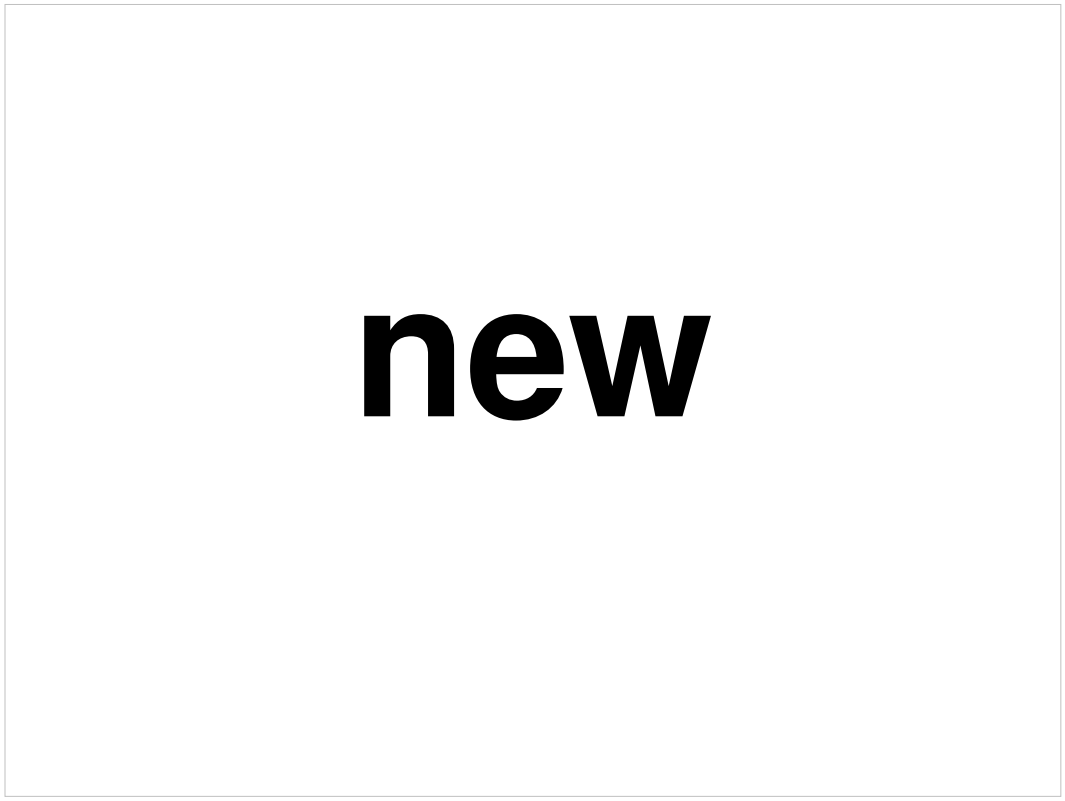


The other myth is the idea that non-relational databases are revolutionary, and entirely new thing under the sun.

There

are


no


new

database

designs

There are only new
implementations
and
combinations

The newest database technoligies are from 2002. The new databases we see today are new implementations of older database concepts. Let me give you an example.

“A database storing application-friendly formatted objects, each containing collections of attributes which can be searched through a document ID, or the creation of ad-hoc indexes as needed by the application.”

This is a description of a non-relational database design. Can you guess which one it is?

CouchDB, 2007

Pick, 1965

This describes both CouchDB today and Pick from 1965.

CouchDB, 2007

embeddable Pick
JSON storage
REST API
map/reduce

Really if you look at couchdb, one of my favorite non-relational databases, it's a good new re-implementation of multiple exisitng concepts, and a new combination of things which had not been comb ined before, at least not usefully.

“revolutionary”

“ evolutionary”

So the new non-relational databases are really not revolutionary, they are evolutionary.

“renaissance of non-relational databases”

What we have today is a renaissance. And that's not a bad thing, it's a good thing. We just need to understand where we are.

Mythbust #3

This is the one you'll see from the relational database camp. It's a bit of FUD from people who don't want to learn new things.

Google

Bigtable

Many of our conference sponsors would disagree.
they use non-relational databases for core business applications.

Amazon

Dynamo

FaceBook

Memcached

US Vetrans' Administration

Pick, Caché

Even the old ones are still in use … Pick and its decendant cache are in use in vetrans' hospitals across the USA.

Mythbust #4

“Relational databases will become obsolete”

This is the hype from a few members of the “noSQL” camp. It's FUD which is popular with them because they are trying to raise money for companies.

“Three decades past, the relational empire conquered the hierarchical hegemony. Today, an upstart challenges the relational empire's dominance ...”

XML Databases 2001

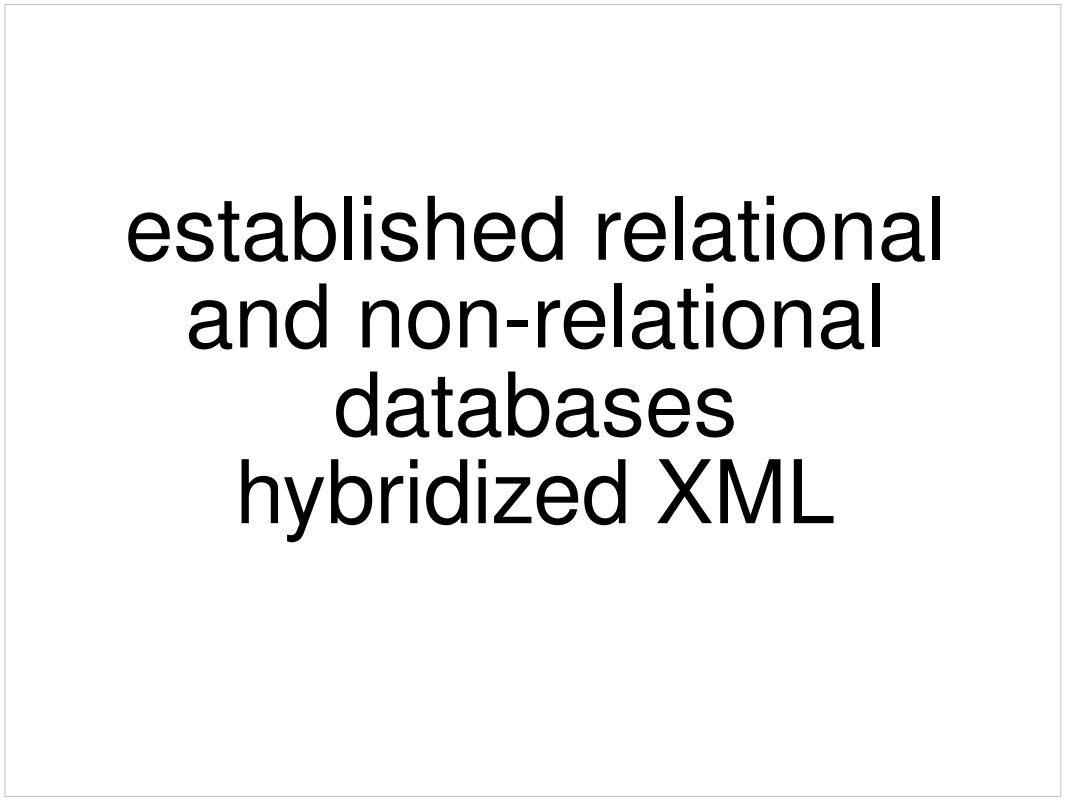
--Philip Wadler, Keynote
VLDB, Rome, September 2001

There was a lot of hype around xmldb a few years back. People were making grand claims about the end of relational databases.

Anyone remember XML databases?

No?

What happened?



mature database platforms have an easy time adding new features. They just absorbed the XML customer base. And dedicated XML dbs went away.

So I think we can exepect to see some hybridization in the future with the new functionality.

Mythbust #5

“Relational databases are for when you need ACID transactions.”

“I don't need a relational database, I'm not a bank”. This is an honest bit of confusion.

Transactions
≠
Relational

Transaction support and relationality are two different, orthagonal features.

Robust Transactions without Relationality:

BerkeleyDB
Amazon Dynamo

SQL Without Transactions:

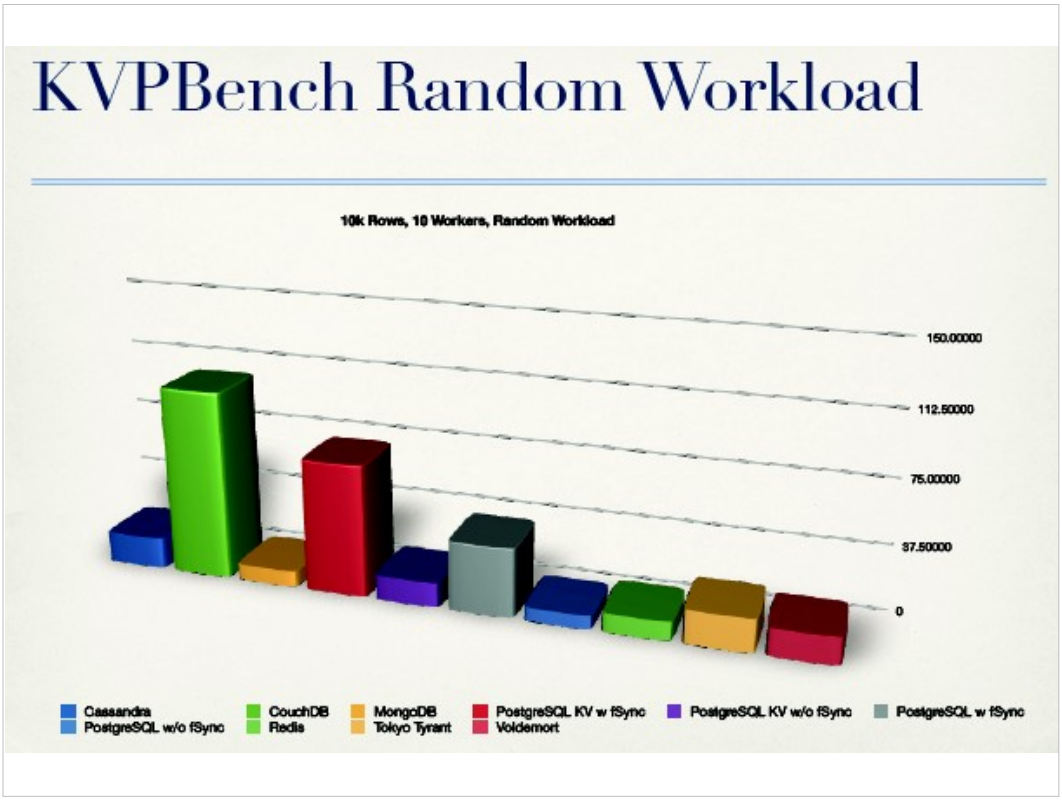
MySQL MyISAM
MS Access

You can have full transaction support without being relational. You can also be a relational database without having full support for transactions.

Mythbust #6

“Users are adopting noSQL for web-scale performance”

This is a popular bit of nonsense which the press has picked up, which ignores the differences befween non-relational databases.



First let's look at single-node performance. Even on a key-value storage workload, the main differentiating factor is whether or not the database persists data to disk (CouchDB, PostgreSQL w/ Fsync). Databases which don't persist to disk (“running with scissors”) are all similarly fast, relational and non-relational.

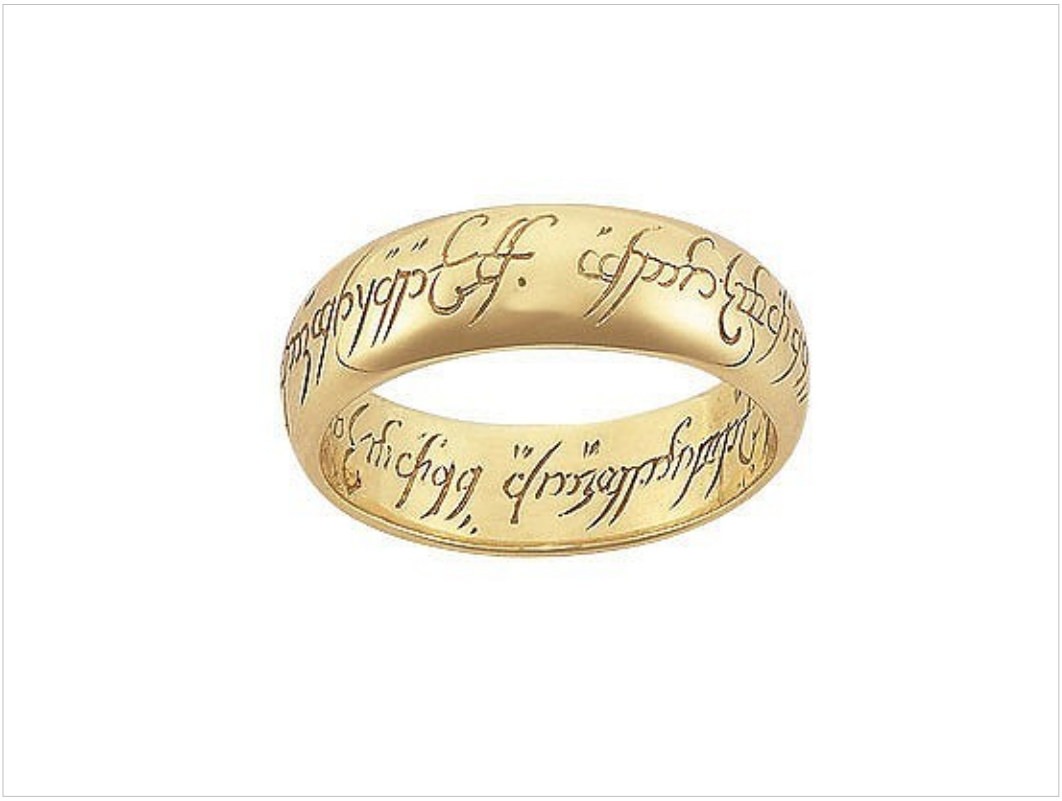
Horizontal Scalability

Database	Difficulty	Scalability	Redundancy
Memcached	Very Easy	10 to 50	None
Cassandra	Difficult	100	Sync
MySQL	Easy	12	Async
Posgres+Skytools	Difficult	20-50	Both
MySQL NDB	Difficult	20-50	Sync
CouchDB	Easy	2 to 6	Async
MongoDB	Moderate	2 to 6	Async
Redis	Easy	2 to 6	Async

Note: data in the above chart is extremely dated. Some databases were tested over a year ago.

If you're more concerned about multi-server scalability, again we have a mixed bag. It's certainly true that a few non-relational databases scale beyond anything else, but most don't. Master-slave replication scales the same regardless of the underlying database.

Mythbust #7



The next myth is something I call “the one ring” school of database selection. This is where developers say “I must find the One True Database” and use it for all things thereafter. Which database is the best at everything?

You

do

not

have

to choose

one database.

In a modern world of fast servers, virtualization, free open source databases and management tools, there is no reason why you need to use one database for every task.

Choose
the database system
which fits your
current
application goals.
or ...

You can pick the database with fits the applicaiton, or

Use more than one together

MySQL + Memcached

PostgreSQL + CouchDB

or ...

you can use more than one database for an application, and probably should, or

Use a Hybrid

MySQL NDB

PostgreSQL Hstore

HadoopDB

You can use a hybrid solution involving more than one database technology.

But what about
relational
vs
non-relational?

Relational OLTP Databases*

- Transactions: more mature support
  - including multi-statement
- Constraints: enforce data rules absolutely
- Consistency: enforce structure 100%
- Complex reporting: keep management happy!
- Vertical scaling (but not horizontal)

** mature ones, anyway*

OLTP-SQL Relational databases are the kind which developers are most faminiar with. Here's a few reasons why you'd want to use one rather than another kind of database.

SQL vs. Not SQL

SQL promotes:

- portability
- managed changes over time
- multi-application access
- many mature tools

Theres alsto the question of SQL or not. SQL has some advantages for the developer.

SQL vs. Not SQL

But …

SQL is a full programming language, and you have to learn it to use it.

However, it's also a full programming language, and thus another language to learn in order to use properly.

SQL vs. Not SQL

No-SQL allows:

- programmers as DBAs
- no impedance
- fast interfaces
- fast development

Not using SQL has some advantages in terms of having a developer-centric shop.

SQL vs. Not SQL

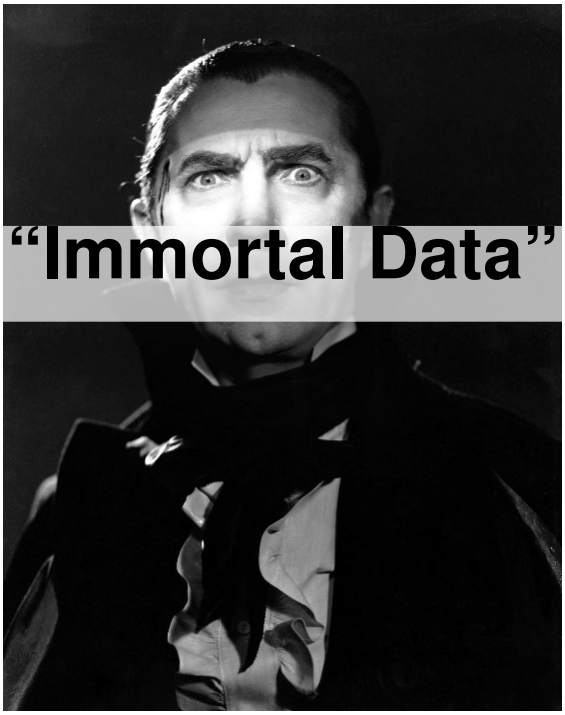
… but:

may involve learning complex proprietary APIs

```
db.things.find({j: {$ne: 3}, k: {$gt: 10}});
```

However, some non-SQL databases have proprietary interfaces every bit as difficult as SQL.

The main reason to use SQL-RDBMSs


“Immortal Data”

“Immortal Data”

your data has
a life
independent
of this specific
application implementation

Important data … business data, personal data … often outlives the current application. SQL-relational databases are designed to help you maintain your data over long periods of time, and through changes of interface, programming language, and even main application purpose.

How *do* I choose?

OK, that still doesn't really tell you how to choose, though, does it?

Define the problem you are trying to solve

Step one is define the data problem or problems you need to solve. If there are several problems, prioritize them.

- “I need a database for my blog”
- “I need to add 1000's of objects per second on a low-end device.”
- “I need my database to enforce business rules across several applications.”
- “I want my application to be location-sensitive.”
- “I need to cache data an access it 100K times per second.”
- “I need to produce summary reports across 2TB of data.”
- “I have a few hundred government documents I need to serve on the web and mine”
- “I need to know who-knows-who-knows-who.”
- “I need to data mine 1000's of 30K bug reports per minute.”

Here's examples of several simple data problems which suggest specific databases.

Define the features you *actually* need

- many connections
- multi-server scalability
- complex query logic
- APIs
- redundancy
- data integrity
- schema/schemaless
- data mining

Once you have a problem definition, then you can settle on a list of features you need, and maybe a priority assigned to each feature. Make sure you diffferentiate between the features you NEED and the ones it would be nice to have … it can be hard to make your coworkers do this.

fit the database to the task

Once we have our feature list, with the dozens of open source databases available, it's time to go shopping!

Blogs seem to be overused as examples of “new database X”. It's like everyone first ports wordpress to a new DB before they do anything else.

Use anything!

- MySQL
- PostgreSQL
- MongoDB
- SQLite
- CouchDB
- Flatfiles
- DBaseIII
- Something you wrote yourself

If it's your personal blog, use anything you want. MySQL, Couch. Flatfiles. Whatever it is, it will work.

“I need my database to unify several applications and keep them consistent.”

Imagine you have 4 or 5 applications for various parts of your business and they all need to share the same data even though they work significantly differently. Not only that, the data needs to be consistent between different applications, some of which are written in different programming languages.

PostgreSQL

“OLTP
SQL-Relational Database”

“It's not just a database: it's a development platform”

This is what mainstream relational databases like PostgreSQL excel at.

Shameless plug … Postgres 9 is in beta, please test!

“I need my application to be location-aware.”

Geo applications are increasingly popular thanks to these pocket computers we all have.

PostGIS

“Geographic Relational Database”

For this you want a geographic relational or non-relational database like PostGIS

PostGIS

- Queries across “contains” “near” “closest”
- Complex geometric map objects
  - polygons
  - lines (roads, etc)

or now … CouchDB Spatial
and Spatialite!

Spatial databases let you do queries on near, within, overlapping, adjacent to, and points and lines.

“I need to store
1000's of event objects
per second on
embedded hardware.”

This is a common datqabase task for embedded hardware these days. Not only does the database have to be very fast, it has to be very small … but doesn't need many features.

db4object

“Embedded Key-Value Store”

For this you need an embedded key-value store like DB4Object, BerkeleyDB, Redis, etc.

db4object

“Embedded Key-Value Store”

BerkeleyDB, Redis, TokyoCabinet, MongoDB

db4object

- German Train System
- Insert 1000's of objects per second
- Low-end embedded console computer
- Simple access in native programming language (Java, .NET)

- *compromise: embedded SQL database: SQLite*

Example: the German train system runs on db4o. You can also use an embedded SQL database if you need SQL features like multiple tables.

“I need to access 100K objects per second over 1000's of connections.”

On the other hand, what you may need is really massive read access, without necessarily needing persistance.

memcached

“Distributed In-Memory Key-Value Store”

This is what caching databases like memcached are for.

memcached

- Use: public website
- Used for caching 1000's of serialized objects per second
- Available for 100000's of requests per second across 1000's of connections
- Cache each object only once per site
- *Supplements* a relational database

Alternatives: Redis, KyotoTyrant, etc.

Most people are familiar with memcached, which is the most mature and stable. But there are increasing numbers of alternatives.

“I need to produce complex summary reports over 2TB of data.”

Other times you have truly massize amounts of data on disk and need to summarize it, even producing fancy reports and graphs.

LucidDB

“Relational Column-Store”

This is what BI databases like LucidDB are for.

LucidDB

- For reporting and analysis
- Large quantities of data (TB)
- Complex OLAP & analytics queries
- Business intelligence
- *compliments* a transactional database

They generally go alongside some other kind of operational database, and can't be beat for large scale data reporting.

“I have 100's of government documents I need to serve on the web and mine for data.”

Say you have a few hundred documents whose structure is not known, and the most important thing to you is to get them on the web or a network before you start analysing them.

CouchDB

“Document Store”

For this you want a web-enabled document store like CouchDB.

CouchDB

1. CividDB Project
2. Storing lots and lots of government documents
3. Don't know what's in them
4. Don't know how they are structured
5. Store them, figure out structure later by data mining.

It's also good for mobile applications!

We used CouchDB for a government open data project because it was very easy to make documents indexed and available vial the web.

“I have a social application and I need to know who-knows-who-knows-who-knows-who-knows-who.”

This is a more common siituation than you'd think … “network queries” like in social network sites, but also for thinks like “you may also like”. Most kinds of databases suck at this kind of query.

Neo4j

“Graph Database”

So you need a graph database.

Neo4j

- Social Network Website
- 6 degrees of separation
- “you may also like”
- type and degrees of relationship

Graph databases do “6 degrees of separation” queries orders of magnitude faster than other kinds of databases. Generally that's all they do, though.

This was the mission at Mozilla, where they wanted to process all of the incoming bug reports instead of just a sample.

Hadoop

"Massively Parallel Data Mine"

Hadoop+HBase is made for this; it distributes cpu-intensive data processing across a large network of machines or virtual machines.

Hadoop + HBase

- Massive bug report data feed
- 1000's of bug reports per minute
- Each bug report 2-45K of data
- Need to extract trends and correlate inexact data
- Summarize in daily & weekly reports

Then the data can be loaded into another kind of database and displayed on the web.

Conclusion

- Different database systems do better at different tasks.
  - every database feature is a tradeoff
  - no database can do all things well
- Relational vs. non-relational doesn't matter
  - pick the database(s) for the project or the task

To wrap up: choose a database based on its actual features matching the features you need, not based on coolness or hype.

Questions?

- PostgreSQL Project
  - www.postgresql.org
  - josh@postgresql.org
- PostgreSQL Experts
  - www.pgexperts.com
    - www.pgexperts.com/documents.html
- Open Source Database Survey
  - Selena Deckleman
  - Open Source Database Survey: www.ossdbsurvey.org

- **PostgreSQL BOF,** Tonight 7pm

Special thanks to Selena Deckelman for the bunnies image and Gavin Roy for NoSQL performance tests.

PGX POSTGRESQL EXPERTS, INC.