IEEE Postgres Keynote: Battling the NoSQL Hype Cycle and Obtaining Academic Assistance

BRUCE MOMJIAN

This talk explores how new NoSQL technologies are unique, and how existing relational database systems like Postgres are adapting to handle NoSQL workloads.

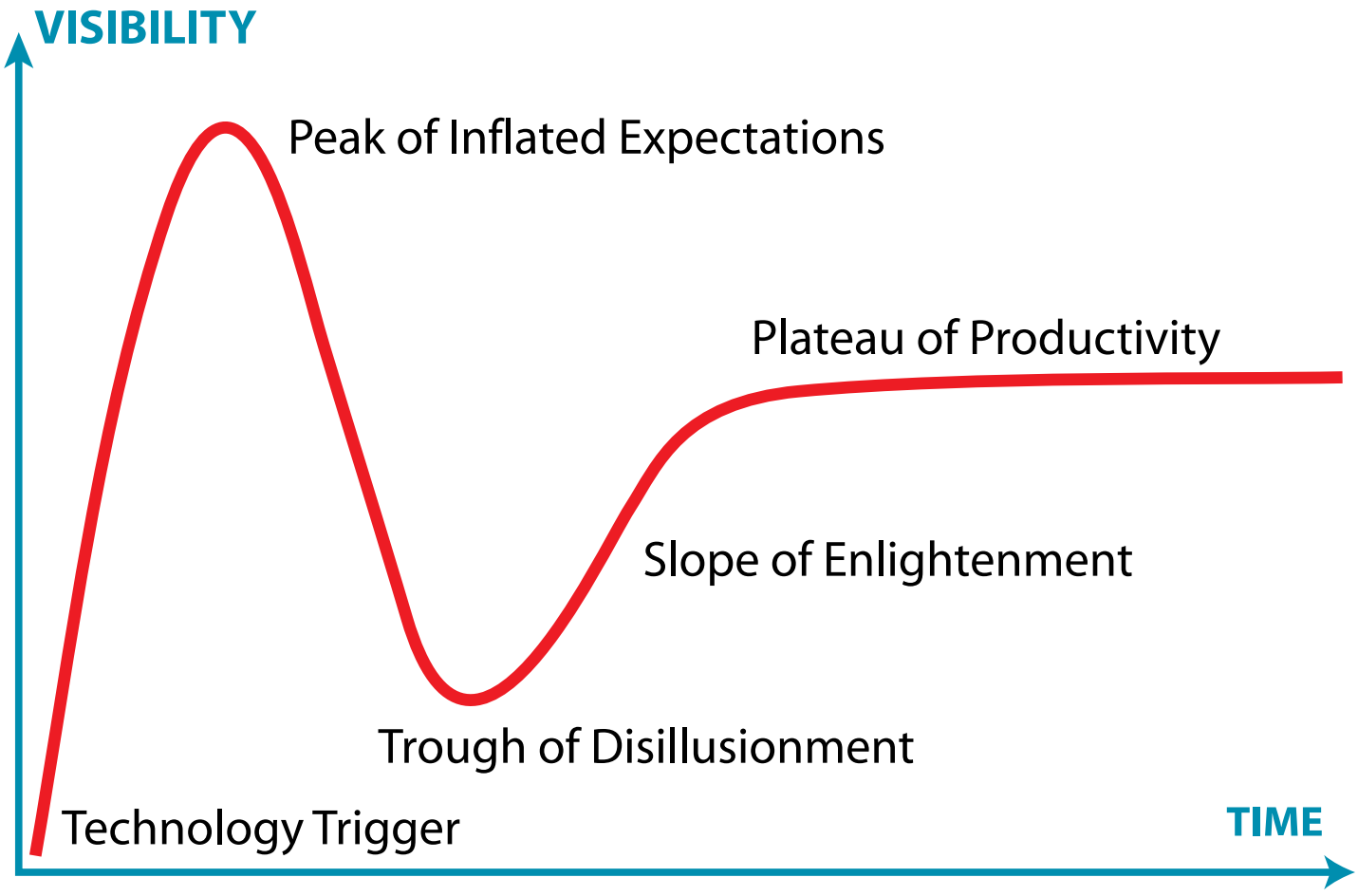
 http://momjian.us/presentations

Last updated: May, 2016

Outline

1. Hype Cycle
2. History of relational challenges
3. NoSQL goals
4. Postgres adaptations
5. Postgres history
6. How you can help

1. Hype Cycle



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

2. History of Relational Challenges

- Object databases
- XML databases

3. NoSQL Goals

There is no single NoSQL technology. They all take different approaches and have different features and drawbacks:

- Key-value stores, e.g. Redis
- Document databases, e.g. MongoDB (JSON)
- Columnar stores: Cassandra
- Graph databases: Neo4j

These are mostly aggregate-oriented — see Martin Fowler's video at https://www.youtube.com/watch?v=qI_g07C_Q5I.

Why NoSQL Exists

Generally, NoSQL is optimized for:

- Fast simple queries
- Auto-sharding
- Flexible schemas

NoSQL Sacrifices

- A powerful query language
- A sophisticated query optimizer
- Data normalization
- Joins
- Referential integrity
- Durability

Are These Drawbacks Worth the Cost?

- **Difficult Reporting** Data must be brought to the client for analysis, e.g. no aggregates or data analysis functions. Schema-less data requires complex client-side knowledge for processing
- **Complex Application Design** Without powerful query language and query optimizer, the client software is responsible for efficiently accessing data and for data consistency
- **Durability** Administrators are responsible for data retention

When Should NoSQL Be Used?

- Massive write scaling is required, more than a single server can provide
- Only simple data access pattern is required
- Additional resource allocation for development is acceptable
- Strong data retention or transactional guarantees are not required
- Unstructured duplicate data that greatly benefits from column compression

When Should Relational Storage Be Used?

- Easy administration
- Variable workloads and reporting
- Simplified application development
- Strong data retention

Postgres Adaptations

Postgres has many NoSQL features without the drawbacks:

- Schema-less data types, with sophisticated indexing support
- Transactional schema changes with rapid addition and removal of columns
- Durability by default, but controllable per-table or per-transaction

Schema-Less Data: JSON and JSONB

```
CREATE TABLE customer (id SERIAL, data JSONB);

INSERT INTO customer VALUES (DEFAULT, '{"name" : "Bill", "age" : 21}');

SELECT data->>'name' FROM customer
WHERE data->>'age' = '21';
 ?column?
----------
 Bill

-- this lookup is indexable
SELECT data->>'name' FROM customer
WHERE data @> '{"age" : 21}'::jsonb;
 ?column?
----------
 Bill
```

Incremental JSON Improvements

- 9.2 (2012): JSON data type (syntax checking)
- 9.3 (2013): JSON extraction and conversion functions
- 9.4 (2014): JSONB (binary JSON) and GIN index improvements
- 9.5 (2016): JSONB generation and manipulation functions

JSONB matches or beats MongoDB in performance and storage size, except for update operations, which are slower.

Easy Relational Schema Changes

```
ALTER TABLE customer ADD COLUMN status CHAR(1);
BEGIN WORK;
ALTER TABLE customer ADD COLUMN debt_limit NUMERIC(10,2);
ALTER TABLE customer ADD COLUMN creation_date TIMESTAMP WITH TIME ZONE;
ALTER TABLE customer RENAME TO cust;
COMMIT;
```

NoSQL Access via Foreign Data Wrappers

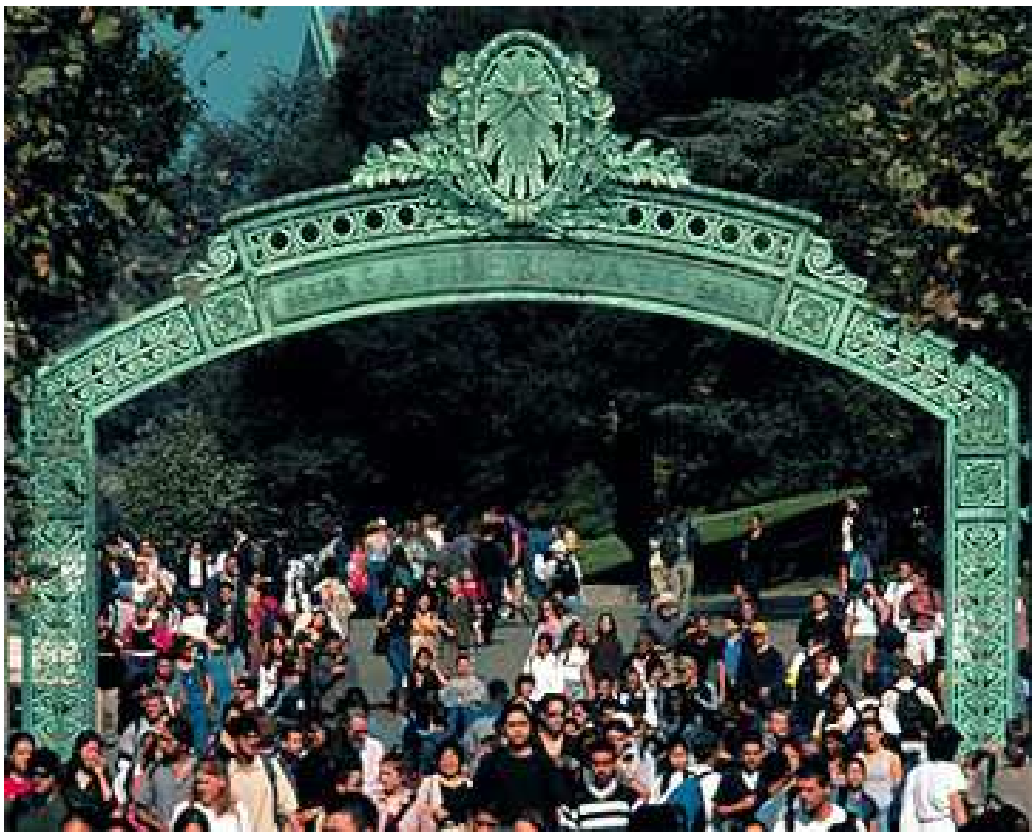
Foreign data wrappers (SQL MED) allow queries to read and write data to foreign data sources. Foreign database support includes:

- Cassandra (columnar)
- CouchDB (document)
- MongoDB (document)
- Neo4j (graph)
- Redis (key-value)

The transfer of aggregates and sorts to foreign servers is not yet implemented. Join transfer is implemented in Postgres 9.6.

http://www.postgresql.org/docs/current/static/ddl-foreign-data.html
http://wiki.postgresql.org/wiki/Foreign_data_wrappers

5. Postgres History
The University of California at Berkeley



Postgres

Michael Stonebraker

Postgres95

Jolly Chen and Andrew Yu

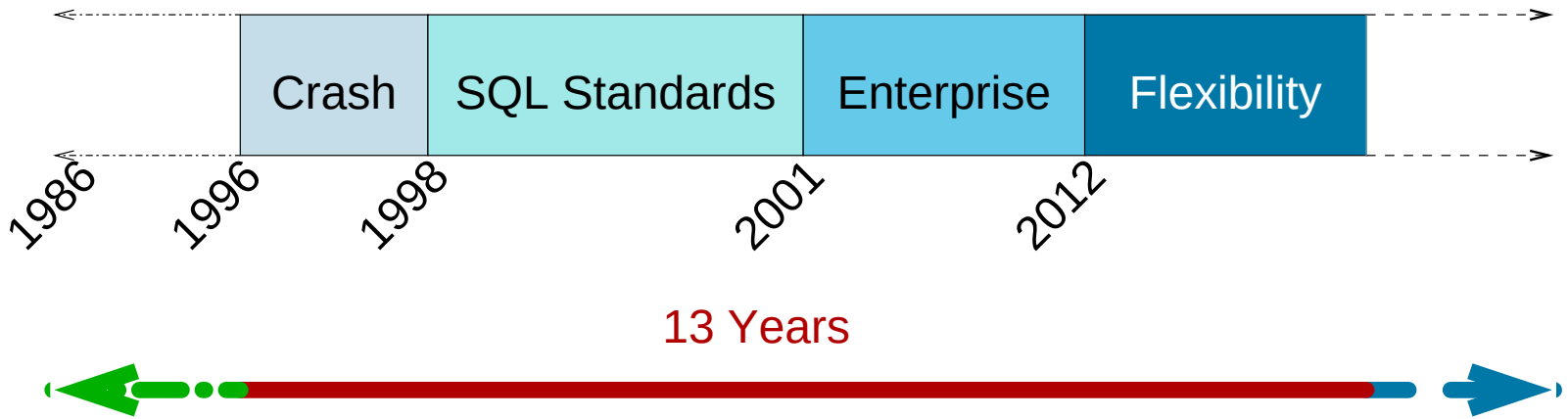
PostgreSQL Code Base History

- Ingres — research prototype, spawned Relational Technologies, purchased by Computer Associates
- Postgres — research prototype, spawned Illustra, purchased by Informix
- Postgres95 — added SQL, spawned PostgreSQL

PostgreSQL Through the Years

1977-1985	Ingres
1986-1994	Postgres
1994-1995	Postgres95
1996-	PostgreSQL

PostgreSQL Evolution



6. How You Can Help

- Query optimization
- Optimizer statistics
- Indexing structures
- Sorting
- Parallelism
- Sharding
- Distributed transactions and snapshots
- Machine learning
- Incremental maintenance of materialized views
- Automated substitution of materialized views in queries
- Performance under very heavy loads
- Non-relational/Json/NoSQL performance

Hardware Issues

- CPU locality
- Reducing function call overhead
- High CPU core performance
- NUMA
- Large RAM performance
- Persistent memory (NVRAM,PCM)
- SSD performance

Tips to Being Effective

- Use the most recent version of PostgreSQL
- Publish source code in a permanent location
- Supply a commit log showing all your changes
- Publish the data used for testing, or data generator code
- Copyright your changes under a Postgres-compatible license

Conclusion

http://momjian.us/presentations

http://flickr.com/photos/vpickering/3617513255