NoSQL way in PostgreSQL

Vibhor Kumar (Principal System Engineer)

Agenda

- Intro to JSON, HSTORE and PL/V8
- JSON History in Postgres
- JSON Data Types, Operators and Functions
- JSON, JSONB– when to use which one?
- JSONB and Node.JS – easy as pie
- NoSQL Performance in Postgres – fast as greased lightning
- Say ‘Yes’ to ‘Not only SQL’
- Useful resources

Let's Ask Ourselves, Why NoSQL?

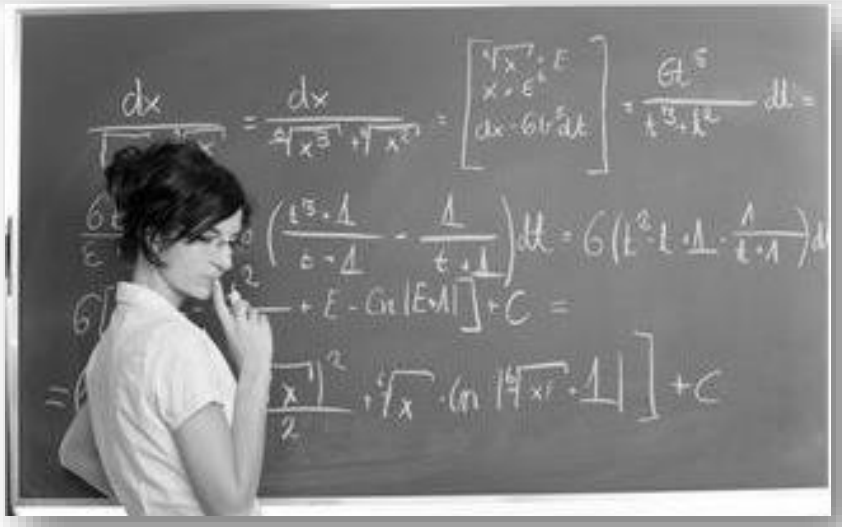
- Where did NoSQL come from?
  - Where all cool tech stuff comes from – Internet companies
- Why did they make NoSQL?
  - To support huge data volumes and evolving demands for ways to work with new data types
- What does NoSQL accomplish?
  - Enables you to work with new data types: email, mobile interactions, machine data, social connections
  - Enables you to work in new ways: incremental development and continuous release
- Why did they have to build something new?
  - There were limitations to most relational databases

NoSQL: Real-world Applications

- Emergency Management System
  - High variability among data sources required high schema flexibility
- Massively Open Online Course
  - Massive read scalability, content integration, low latency
- Patient Data and Prescription Records
  - Efficient write scalability
- Social Marketing Analytics
  - Map reduce analytical approaches

Source: Gartner, A Tour of NoSQL in 8 Use Cases,
by Nick Heudecker and Merv Adrian, February 28, 2014

Postgres' Response



- HSTORE
  - Key-value pair
  - Simple, fast and easy
  - Postgres v 8.2 – pre-dates many NoSQL-only solutions
  - Ideal for flat data structures that are sparsely populated
- JSON
  - Hierarchical document model
  - Introduced in Postgres 9.2, perfected in 9.3
- JSONB
  - Binary version of JSON
  - Faster, more operators and even more robust
  - Postgres 9.4

Postgres: Key-value Store

- Supported since 2006, the HStore contrib module enables storing key/value pairs within a single column
- Allows you to create a schema-less ACID compliant data store within Postgres
- Create single HStore column and include, for each row, only those keys which pertain to the record
- Add attributes to a table and query without advance planning
- Combines flexibility with ACID compliance

HSTORE Examples

- Create a table with HSTORE field

```
CREATE TABLE hstore_data (data HSTORE);
```

- Insert a record into hstore_data

```
INSERT INTO hstore_data (data) VALUES ('
          "cost"=>"500",
          "product"=>"iphone",
          "provider"=>"apple"');
```

- Select data from hstore_data

```
SELECT data FROM hstore_data ;
-----------------------------------------
"cost"=>"500”,"product"=>"iphone”,"provider"=>"Apple"
(1 row)
```

Postgres: Document Store

- JSON is the most popular data-interchange format on the web
- Derived from the ECMAScript Programming Language Standard (European Computer Manufacturers Association).
- Supported by virtually every programming language
- New supporting technologies continue to expand JSON's utility
  - PL/V8 JavaScript extension
  - Node.js
- Postgres has a native JSON data type (v9.2) and a JSON parser and a variety of JSON functions (v9.3)
- Postgres will have a JSONB data type with binary storage and indexing (coming – v9.4)

JSON Examples

- Creating a table with a JSONB field

```
CREATE TABLE json_data (data JSONB);
```

- Simple JSON data element:

```
{"name": "Apple Phone", "type": "phone", "brand":
"ACME", "price": 200, "available": true,
"warranty_years": 1}
```

- Inserting this data element into the table json_data

```
INSERT INTO json_data (data)  VALUES
    (' {   "name": "Apple Phone",
           "type": "phone",
           "brand": "ACME",
           "price": 200,
           "available": true,
           "warranty_years": 1
    } ')
```

JSON Examples

- JSON data element with nesting:

```
{“full name”: “John Joseph Carl Salinger”,
“names”:
    [
    {"type": "firstname", “value”: ”John”},
    {“type”: “middlename”, “value”: “Joseph”},
    {“type”: “middlename”, “value”: “Carl”},
    {“type”: “lastname”, “value”: “Salinger”}
    ]
}
```

A simple query for JSON data

```
SELECT DISTINCT
    data->>'name' as products
FROM json_data;

            products
-------------------------------
 Cable TV Basic Service Package
 AC3 Case Black
 Phone Service Basic Plan
 AC3 Phone
 AC3 Case Green
 Phone Service Family Plan
 AC3 Case Red
 AC7 Phone
```

This query does not return JSON data – it returns text values associated with the key ‘name’

A query that returns JSON data

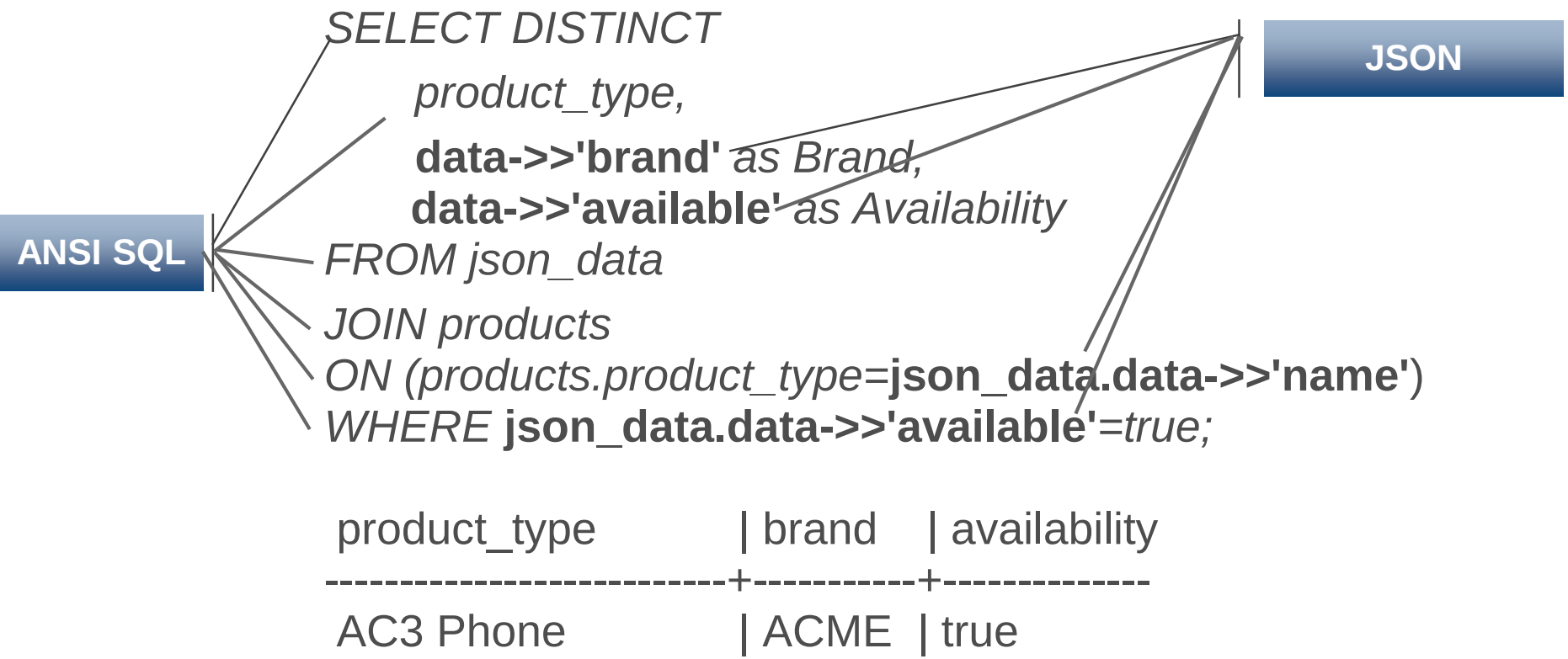
```
SELECT data FROM json_data;

data

----------------------------------------

 {"name": "Apple Phone", "type": "phone",
"brand": "ACME", "price": 200,
"available": true, "warranty_years": 1}
```

This query returns the JSON data in its original format

JSON and ANSI SQL - PB&J for the DBA

- JSON is naturally integrated with ANSI SQL in Postgres
- JSON and SQL queries use the same language, the same planner, and the same ACID compliant transaction framework
- JSON and HSTORE are elegant and easy to use extensions of the underlying object-relational model

JSON and ANSI SQL Example



No need for programmatic logic to combine SQL and NoSQL in the application – Postgres does it all

Bridging between SQL and JSON

Simple ANSI SQL Table Definition

```
CREATE TABLE products (id integer, product_name text );
```

Select query returning standard data set

```
SELECT * FROM products;
```

```
 id | product_name
----+--------------
  1 | iPhone
  2 | Samsung
  3 | Nokia
```

Select query returning the same result as a JSON data set

```
SELECT ROW_TO_JSON(products) FROM products;
```

```
{"id":1,"product_name":"iPhone"}
{"id":2,"product_name":"Samsung"}
{"id":3,"product_name":"Nokia”}
```

JSON Data Types

JSON is defined per RFC – 7159
For more detail please refer
http://tools.ietf.org/html/rfc7159

- 1. Number:
  - Signed decimal number that may contain a fractional part and may use exponential notation.
  - No distinction between integer and floating-point
- 2. String
  - A sequence of zero or more Unicode characters.
  - Strings are delimited with double-quotation mark
  - Supports a backslash escaping syntax.
- 3. Boolean
  - Either of the values true or false.
- 4. Array
  - An ordered list of zero or more values,
  - Each values may be of any type.
  - Arrays use square bracket notation with elements being comma-separated.
- 5. Object
  - An unordered associative array (name/value pairs).
  - Objects are delimited with curly brackets
  - Commas to separate each pair
  - Each pair the colon ':' character separates the key or name from its value.
  - All keys must be strings and should be distinct from each other within that object.
- 6. null
  - An empty value, using the word null

JSON Data Type Example

```
{
  "firstName": "John",           -- String Type
  "lastName": "Smith",           -- String Type
  "isAlive": true,               -- Boolean Type
  "age": 25,                     -- Number Type
  "height_cm": 167.6,            -- Number Type
  "address": {                   -- Object Type
    "streetAddress": "21 2nd Street”,
    "city": "New York”,
    "state": "NY”,
    "postalCode": "10021-3100”
  },
  "phoneNumbers": [         // Object Array
    {                      // Object
      "type": "home”,
      "number": "212 555-1234”
    },
    {
      "type": "office”,
      "number": "646 555-4567”
    }
  ],
  "children": [],
  "spouse": null          // Null
}
```

JSON 9.4 – New Operators and Functions

- JSON
  - New JSON creation functions (json_build_object, json_build_array)
  - json_typeof – returns text data type (‘number’, ‘boolean’, …)
- JSONB data type
  - Canonical representation
    - Whitespace and punctuation dissolved away
    - Only one value per object key is kept
    - Last insert wins
    - Key order determined by length, then bytewise comparison
  - Equality, containment and key/element presence tests
  - New JSONB creation functions
  - Smaller, faster GIN indexes
  - jsonb subdocument indexes
    - Use “get” operators to construct expression indexes on subdocument:
    - `CREATE INDEX author_index ON books USING GIN ((jsondata -> 'authors'));`
    - `SELECT * FROM books WHERE jsondata -> 'authors' ? 'Carl Bernstein'`

JSON and BSON

- BSON – stands for ‘Binary JSON’
- BSON != JSONB
  - BSON cannot represent an integer or floating-point number with more than 64 bits of precision.
  - JSONB can represent arbitrary JSON values.
- Caveat Emptor!
  - This limitation will not be obvious during early stages of a project!

JSON, JSONB or HSTORE?

- JSON/JSONB is more versatile than HSTORE
- HSTORE provides more structure
- JSON or JSONB?
  - if you need any of the following, use JSON
    - Storage of validated json, without processing or indexing it
    - Preservation of white space in json text
    - Preservation of object key order Preservation of duplicate object keys
    - Maximum input/output speed
- For any other case, use JSONB

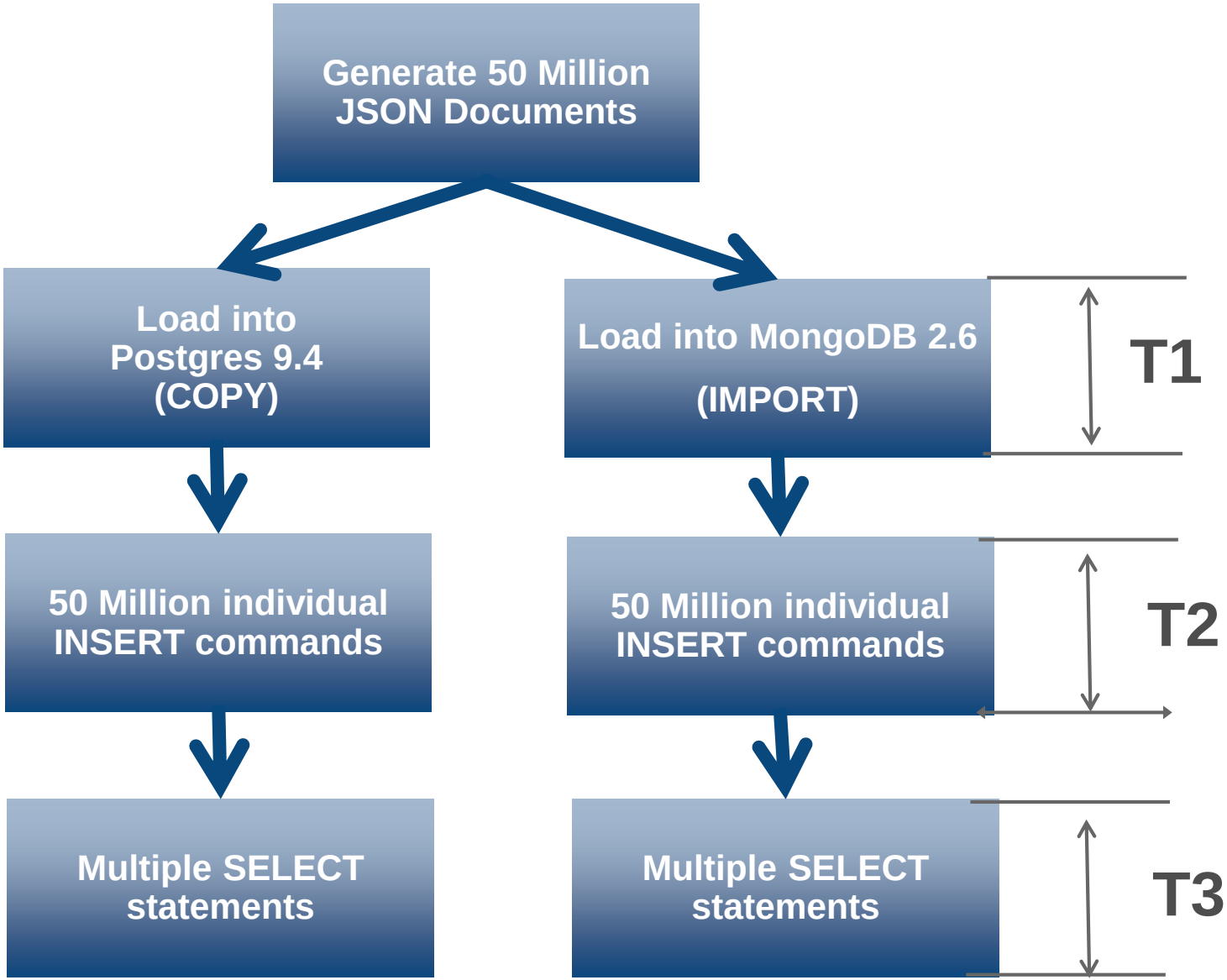
JSONB and Node.js - Easy as TT

```
// require the Postgres connector
var pg = require("pg");

// connection to local database
var conString = "pg://postgres:password@localhost:5432/nodetraining";

var client = new pg.Client(conString);
client.connect();

// initiate the sample database
 client.query("CREATE TABLE IF NOT EXISTS emps(data jsonb)");
 client.query("TRUNCATE TABLE emps;");
 client.query('INSERT INTO emps VALUES($JSON$ {"firstname": "Ronald" , "lastname":"McDonald" }$JSON$)')
 client.query('INSERT  INTO emps values($JSON$ {"firstname": "Mayor", "lastname": "McCheese"}$JSON$)')

// run SELECT query
 client.query("SELECT * FROM emps",function(err,result){
     console.log("Test Output of JSON Result Object");
     console.log(result);
     console.log("Parsed rows");

// parse the result set
    for (var i = 0; i< result.rows.length ; i++ ){
        var data = JSON.parse(result.rows[i].data);
        console.log("First Name => "+ data.firstname + "\t| Last Name => " + data.lastname);
    }
 client.end();
})
```

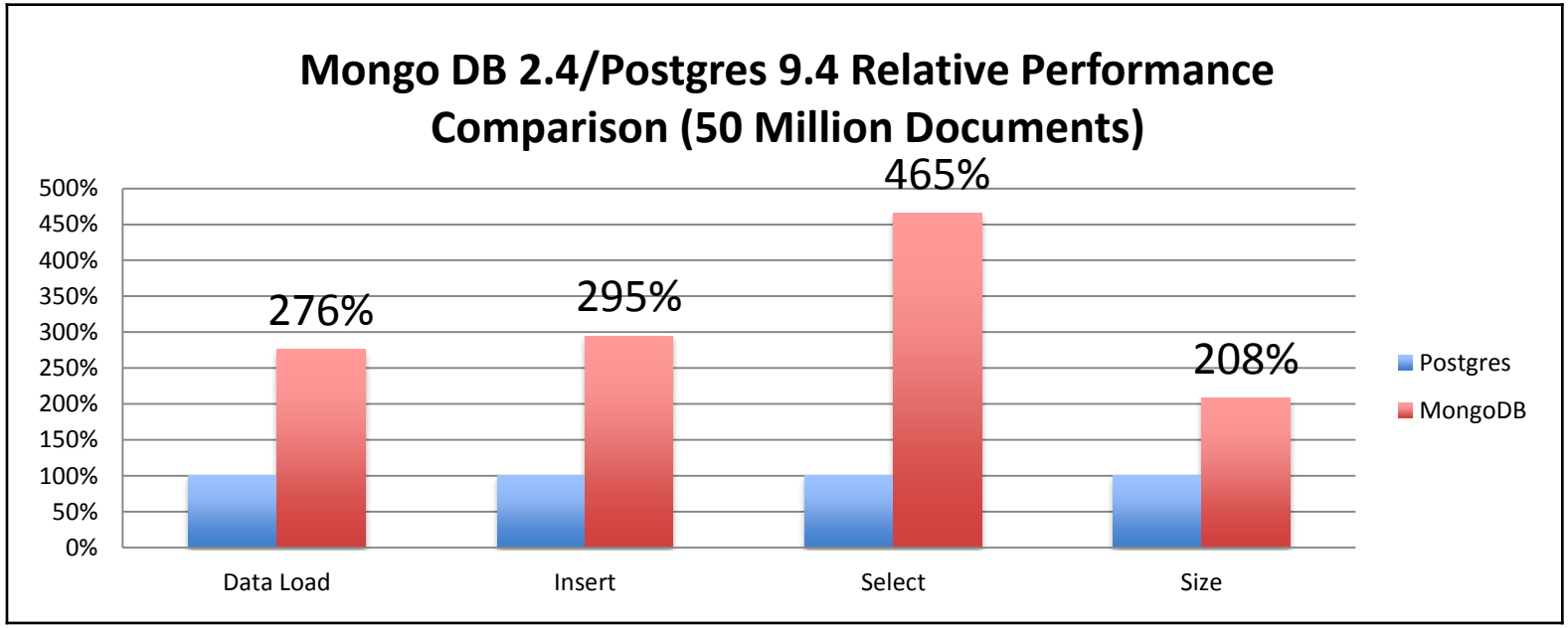
JSON Performance Evaluation

- Goal
  - Help our customers understand when to chose Postgres and when to chose a specialty solution
  - Help us understand where the NoSQL limits of Postgres are
- Setup
  - Compare Postgres 9.4 to Mongo 2.6
  - Single instance setup on AWS M3.2XLARGE (32GB)
- Test Focus
  - Data ingestion (bulk and individual)
  - Data retrieval

Performance Evaluation



NoSQL Performance Evaluation



	Postgres	MongoDB
Data Load (s)	4,732	13,046
Insert (s)	29,236	86,253
Select (s)	594	2,763
Size (GB)	69	145

Correction to earlier versions:

MongoDB console does not allow for INSERT of documents > 4K. This lead to truncation of the MongoDB size by approx. 25% of all records in the benchmark.

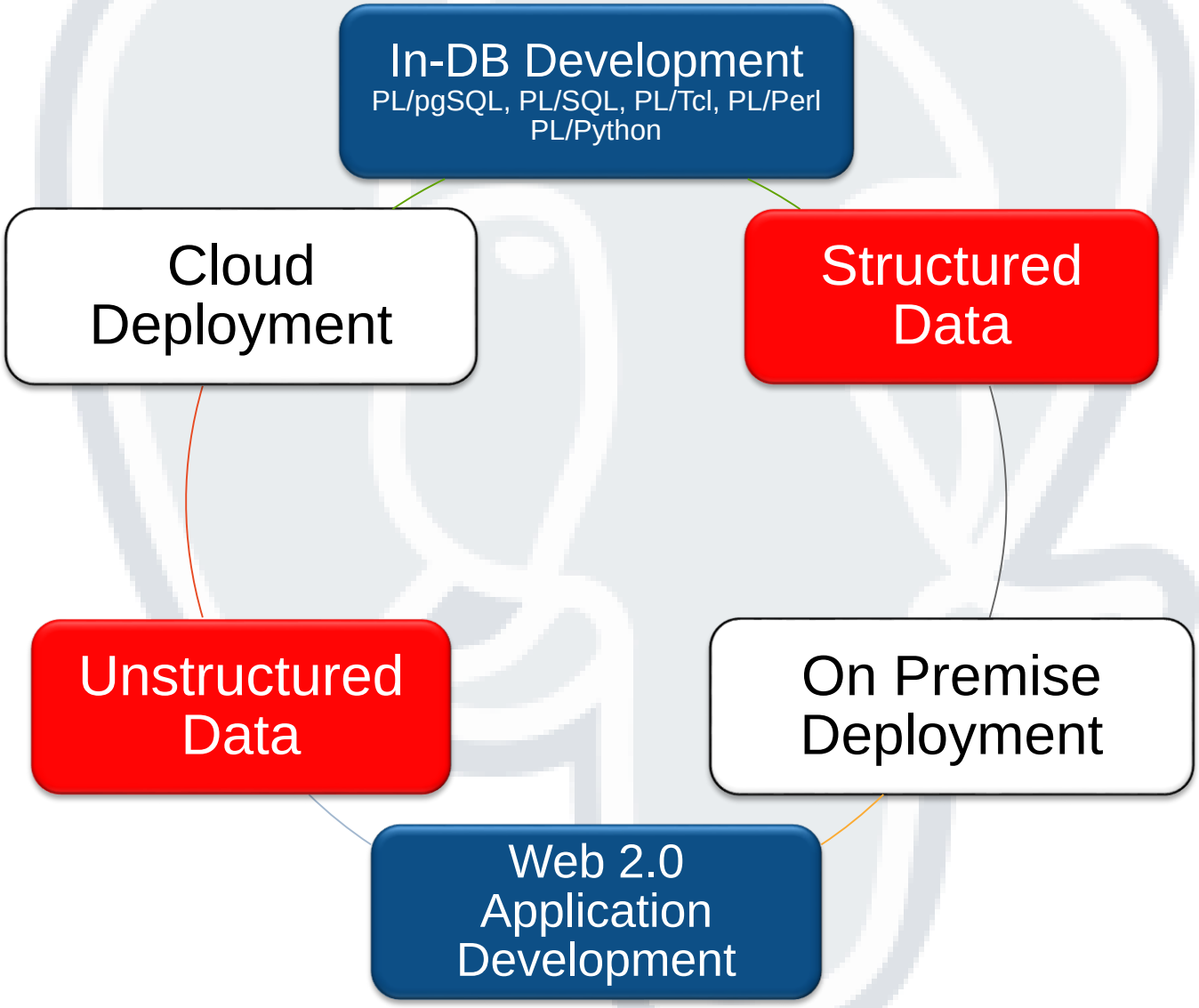
Performance Evaluations – Next Steps

- Initial tests confirm that Postgres' can handle many NoSQL workloads
- EDB is making the test scripts publically available
- EDB encourages community participation to better define where Postgres should be used and where specialty solutions are appropriate
- Download the source at https://github.com/EnterpriseDB/pg_nosql_benchmark
- Join us to discuss the findings at http://bit.ly/EDB-NoSQL-Postgres-Benchmark

Structured or Unstructured?
“No SQL Only” or “Not Only SQL”?

- Structures and standards emerge!
- Data has references (products link to catalogues; products have bills of material; components appear in multiple products; storage locations link to ISO country tables)
- When the database has duplicate data entries, then the application has to manage updates in multiple places – what happens when there is no ACID transactional model?

Ultimate Flexibility with Postgres



Say yes to ‘Not only SQL’

- Postgres overcomes many of the standard objections “It can’t be done with a conventional database system”
- Postgres
  - Combines structured data and unstructured data (ANSI SQL and JSON/HSTORE)
  - Is faster (for many workloads) than than the leading NoSQL-only solution
  - Integrates easily with Web 2.0 application development environments
  - Can be deployed on-premise or in the cloud

Do more with Postgres – the Enterprise NoSQL Solution

Useful Resources

- Postgres NoSQL Training Events
  - Bruce Momjian & Vibhor Kumar @ pgEurope
    - Madrid (Oct 21): Maximizing Results with JSONB and PostgreSQL
- Whitepapers @ http://www.enterprisedb.com/nosql-for-enterprise
  - PostgreSQL Advances to Meet NoSQL Challenges (business oriented)
  - Using the NoSQL Capabilities in Postgres (full of code examples)
- Run the NoSQL benchmark
  - https://github.com/EnterpriseDB/pg_nosql_benchmark

Do More With Postgres!