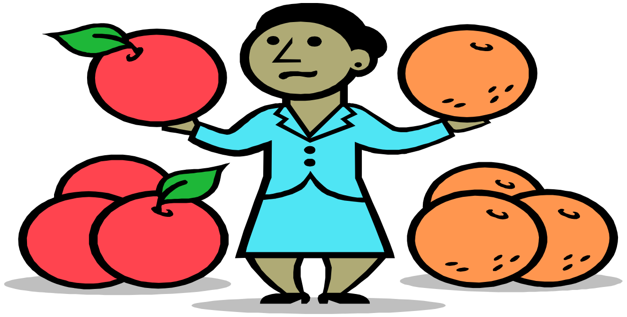
Informix NoSQL

Apples and Oranges

Relational systems and non-relational systems solve different problems and have different philosophies on server responsibility.

Informix – Relational Database	MongoDB - Document Store
Scales within node and by adding nodes	**Scales by adding nodes**
Suite of data protection capabilities	Minimal security
Transactional	No multi-statement transactions
Guaranteed writes	Write concern levels
Consistency of data	Eventual consistency
DB schema defines app structures	**App structures define DB data**



NoSQL Requirements driven by Use Cases

Description	Informix
Consistent Low Latency, even under high load • Ability to handle thousands of users • Typically millisecond response time	Yes
Schema Flexibility & Development Agility • Application not constrained by fixed pre-defined schema • Application drives the schema • Ability to develop a minimal application rapidly, and iterate quickly in response to customer feedback • Ability to quickly add, change or delete "fields" or data-elements • Ability to handle a mix of structured and unstructured data • Easier, faster programming -> Faster time to market, quick to adapt	Yes
Continuous Availability • 24x7x365 availability • (Today) Requires data distribution and replication • Online Maintenance Operations • Ability to upgrade hardware or software without any down time	Yes
Dynamic Elasticity • Rapid horizontal scalability • Ability to add or delete nodes dynamically • Application transparent elasticity (e.g. automatic (redistribution of data, if needed) • Cloud compatibility	Yes
Low cost infrastructure • Commonly available hardware (Windows & Linux,…) • Lower cost software (open source or pay-per-use in cloud)	Yes
Low/No Admin Reduced need for database administration, and maintenance	Yes

Full ACID (Atomicity, Consistency, Isolation, Durability) NOT a requirement

High Level Solution – Why is it Important?

- **Modern Interface providing JSON and BSON native support**
  - Flexible Schema support allows rapid delivery of applications
  - Compatible with all MongoDB programming interfaces
  - Connect the same application developed for MongoDB to Informix with minimal/no application changes
- **Simplify the "up and running" experience and automatically tune the data store with only 3 questions:**
  - Where to place the ***Product***?
  - Where to place the ***Data***?
  - How many ***users*** do you anticipate?
- **Super scale out**
  - Simplify the ability to scale out to multiple nodes, multiple versions, multiple copies
  - Provided diskless and disk based scale out at the individual node with automatic failover
  - Provided Sharded Insert, Update, Delete and Query operations
  - Cloud and Virtualized environment supportability

AP:Invoice Entry
ACME Airlines Mobile
Search for Flights
Checkin for Flights
Update Profile
Login/Logout
Settings
IBM
Informix
NoSQL Cluster

JSON: JavaScript Object Notation

- **What is JSON?**
  - JSON is lightweight text-data interchange format
  - JSON is language independent
  - JSON is "self-describing" and easy to understand

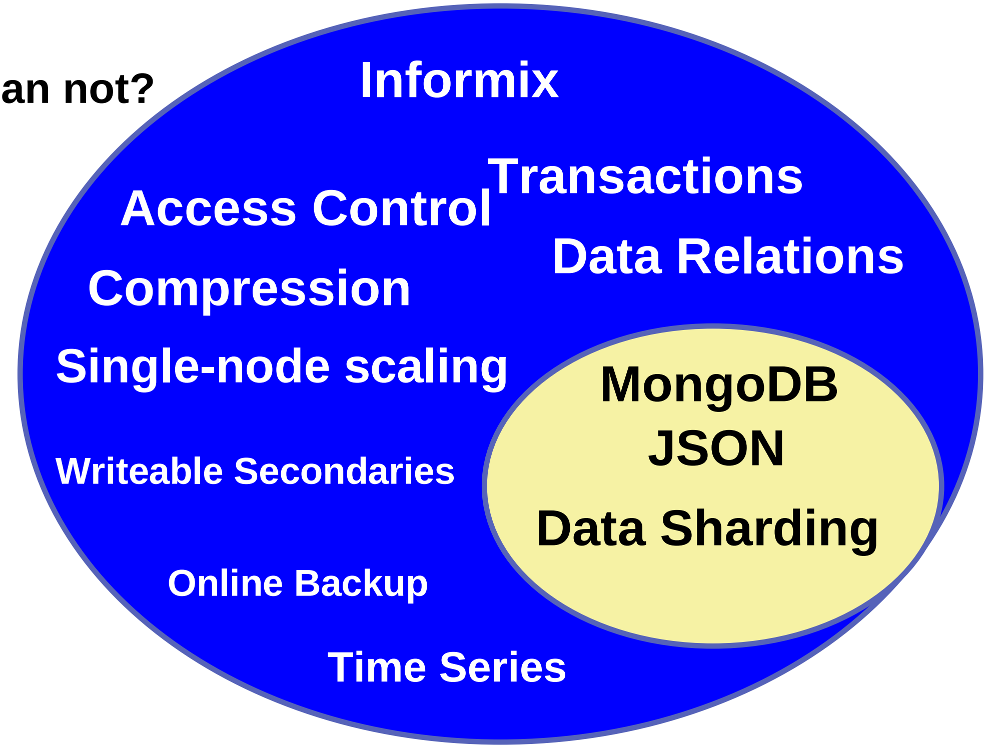
- **JSON is syntax for storing and exchanging text information much like XML. However, JSON is smaller than XML, and faster and easier to parse.**

```
{
"name":"John Miller",
"age":21,
"count":27,
"employees": [
    { "firstName":"John" , "lastName":"Doe" },
    { "firstName":"Anna" , Middle”:”Marie”,"lastName":"Smith" },
    { "firstName":"Peter" , "lastName":"Jones" }
                  ]
}
```

BSON is a binary form of JSON.

Major Capability Differences

- **What can MongoDB and Informix both do?**
  - – Handle structured data in JSON format
  - – Distribute(shard) query execution between server nodes
- **What can Informix do that MongoDB can not?**
  - – Relationships between entities
  - – Transactions
  - – Access Control
  - – …a great many things



Hybrid Solution – Best of Both Worlds

- **Relational and non-relational data in one system**
  - JSON (BSON) as first-class citizen data type
- **Distributed Queries**
- Multi-statement Transactions
- Enterprise Proven Reliability
- Enterprise Ready Security
- Enterprise Level Performance

Informix provides the ability to leverage

the abilities of both relational DBMS and document store systems.

MongoDB does not. It is a document store system lacking key abilities like transaction durability.

Scalability

- **Better performance on multi-core, multi-session scenarios**
  - Architecture has finer grain locking – not just entire database as with MongoDB
  - Better concurrency because less resources locked
- **Document Compression**
  - 60% to 90% observed
- **Bigger documents – 2GB maximum size**
  - MongoDB caps at 16MB
- **Informix has decades of optimization on single node solution**
- **Better utilization of enterprise system resources means less need to shard**
- **MongoDB has higher space requirements for same data**

Security

- **Encryption**
  - – Protects data from access in transit and on disk
- **Auditing**
  - – Records who has accessed data
- **Discretionary Access Control**
  - – Verifies that a user is authorized to do what they are trying to do – roles, etc

- **Decades of solving customer security requirements**

- **With MongoDB**
  - – Security mostly responsibility of the application
  - – Every application has to code for security
  - – Consistent implementation of policies?

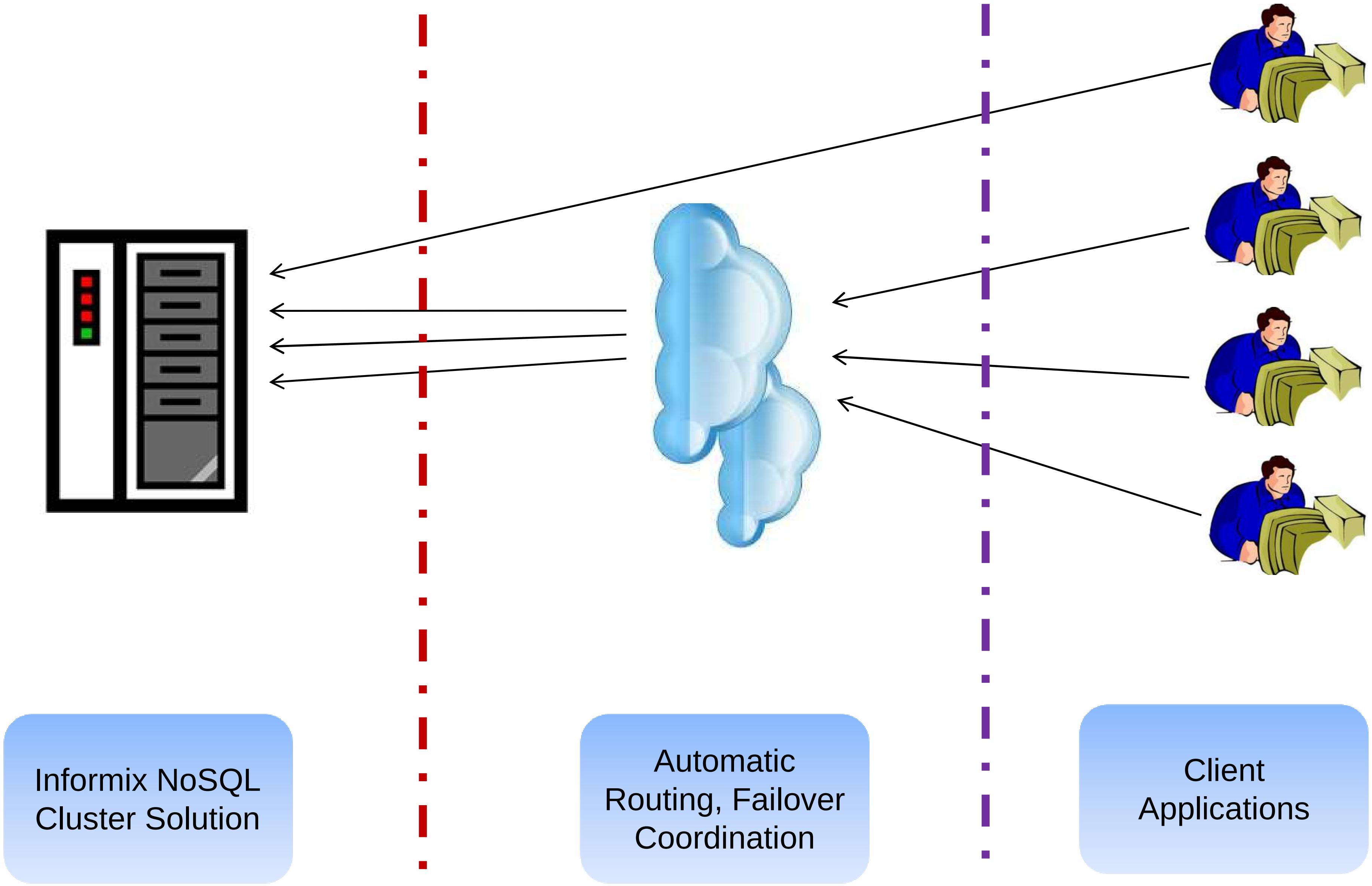
Support and Maintenance

- **IBM Informix Support**
  - Consistently highly rated (#1 at VendorRate 2009)
  - Simple offering
  - Severity and level of response determined by impact to customer

- **Informix reliability second to none**
  - Greater than five 9s uptime
  - Possible to manage 1000s of seats per DBA

- **MongoDB Support**
  - Various support offerings
  - Level of response determined by subscription

High Level Architecture



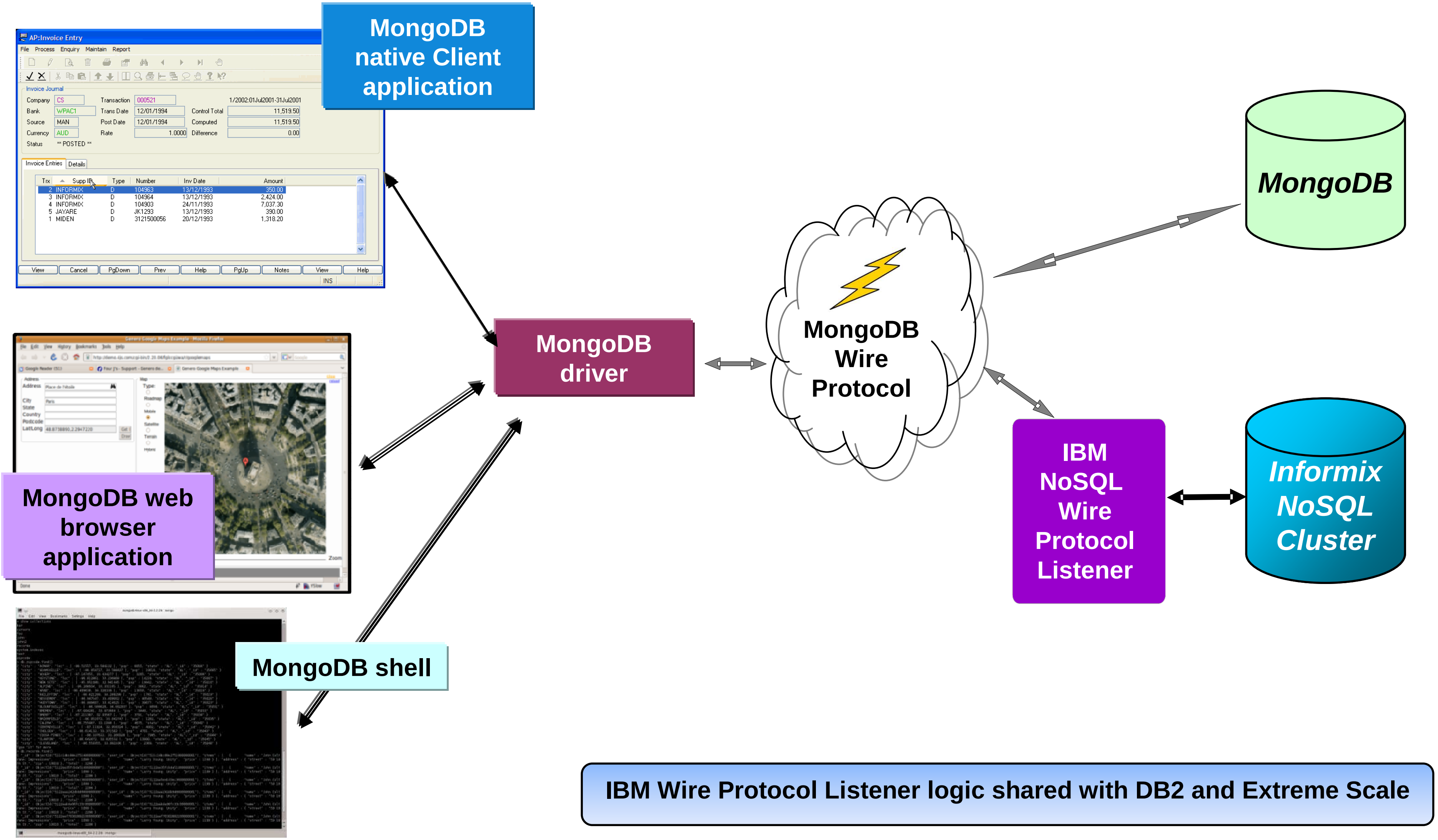
Informix NoSQL Cluster Solution

Automatic Routing, Failover Coordination

Client Applications

Client Applications

- New Wire Protocol Listener supports existing MongoDB drivers
- Connect to MongoDB or Informix with same application!



MongoDB native Client application

AP:Invoice Entry

File Process Enquiry Maintain Report

Invoice Journal

Company	CS	Transaction	000521	1/2002:01Jul2001-31Jul2001	
Bank	WPAC1	Trans Date	12/01/1994	Control Total	11,519.50
Source	MAN	Post Date	12/01/1994	Computed	11,519.50
Currency	AUD	Rate	1.0000	Difference	0.00
Status	** POSTED **				

Invoice Entries | Details

Trx	Supp ID	Type	Number	Inv Date	Amount
2	INFORMIX	D	104963	13/12/1993	350.00
3	INFORMIX	D	104964	13/12/1993	2,424.00
4	INFORMIX	D	104903	24/11/1993	7,037.30
5	JAYARE	D	JK1293	13/12/1993	390.00
1	MIDEN	D	3121500056	20/12/1993	1,318.20

MongoDB web browser application

MongoDB shell

MongoDB driver

MongoDB Wire Protocol

MongoDB

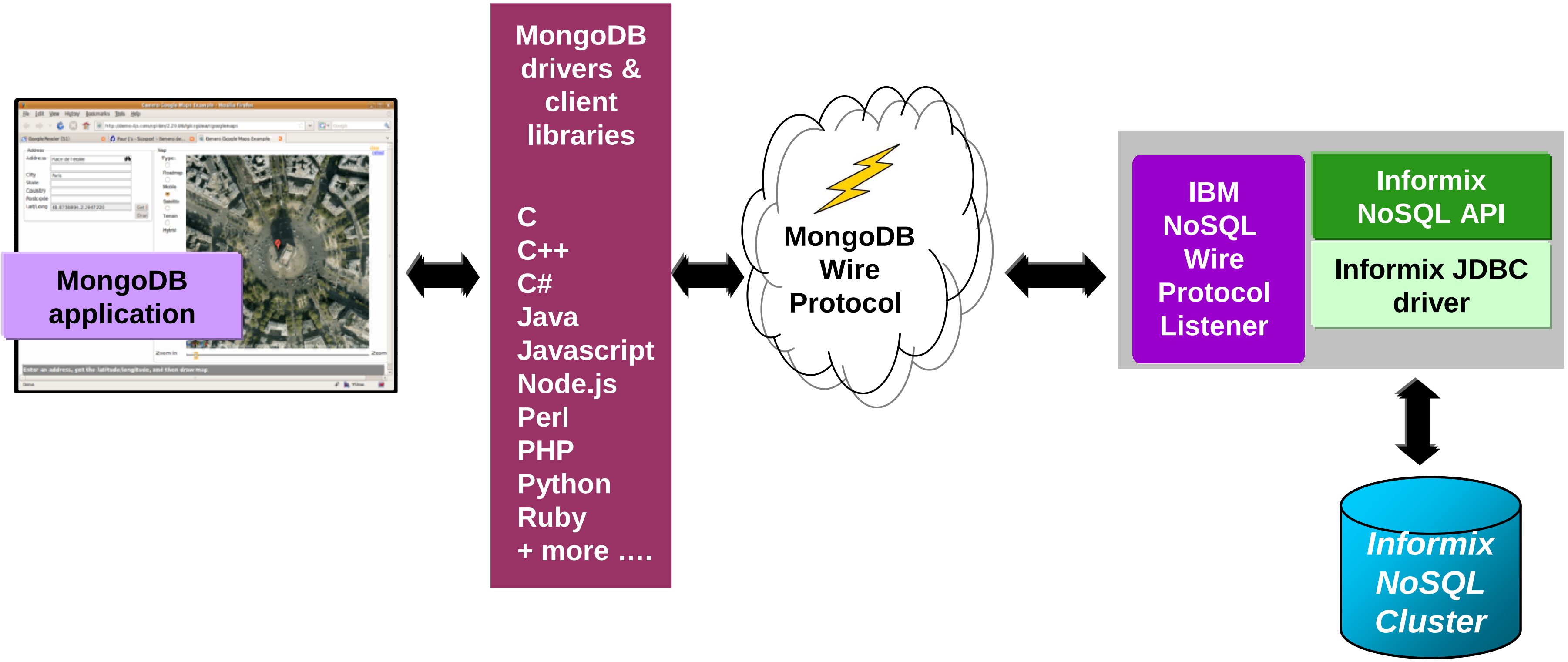
IBM NoSQL Wire Protocol Listener

Informix NoSQL Cluster

IBM Wire Protocol Listener logic shared with DB2 and Extreme Scale

Client Applications - Details

- **NoSQL Wire Protocol Listener works with existing drivers using standard MongoDB client-server protocol**
  - Java, PHP, Python, Javascript, etc.
  - MongoDB supported or community supported drivers
- **Uses new Informix NoSQL API functions**
- **Connectivity to Informix via JDBC**



MongoDB application

MongoDB drivers & client libraries

C
C++
C#
Java
Javascript
Node.js
Perl
PHP
Python
Ruby
+ more ….

MongoDB Wire Protocol

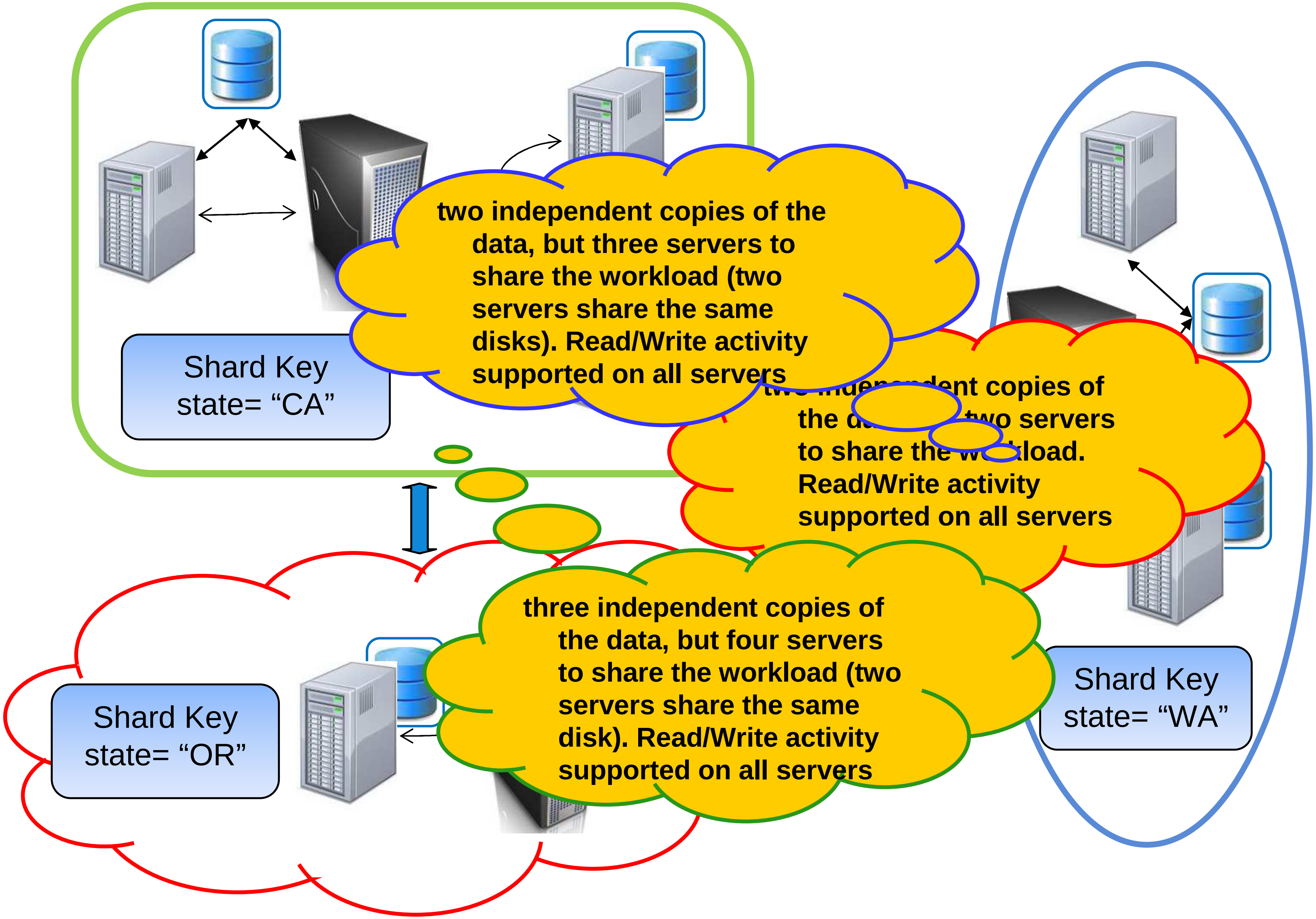
IBM NoSQL Wire Protocol Listener

Informix NoSQL API

Informix JDBC driver

Informix NoSQL Cluster

Informix NoSQL Cluster Architecture Overview

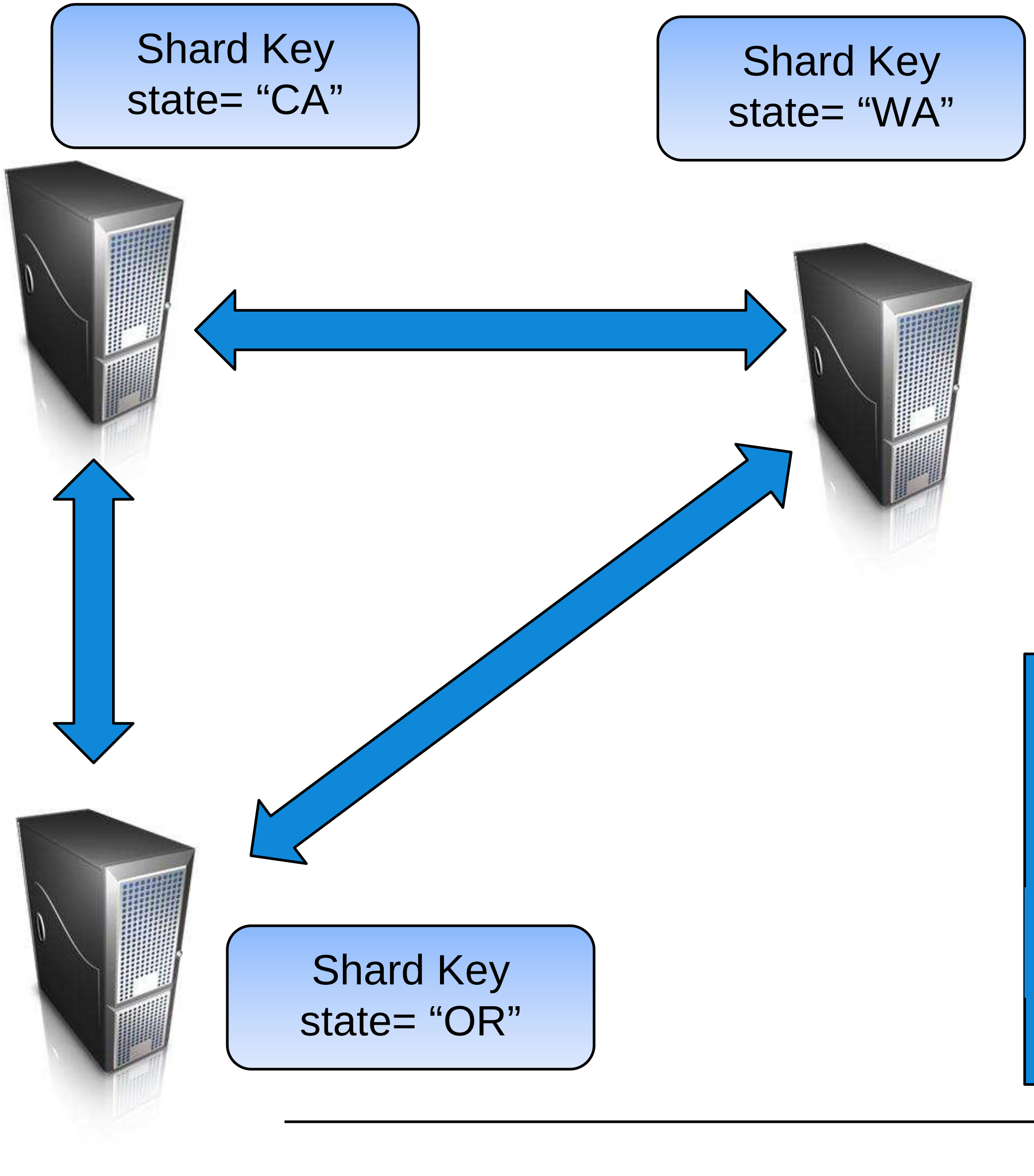


two independent copies of the data, but three servers to share the workload (two servers share the same disks). Read/Write activity supported on all servers

two independent copies of the data, but two servers to share the workload. Read/Write activity supported on all servers

three independent copies of the data, but four servers to share the workload (two servers share the same disk). Read/Write activity supported on all servers

Scaling Out - Sharded Query



Shard Key
state= “CA”

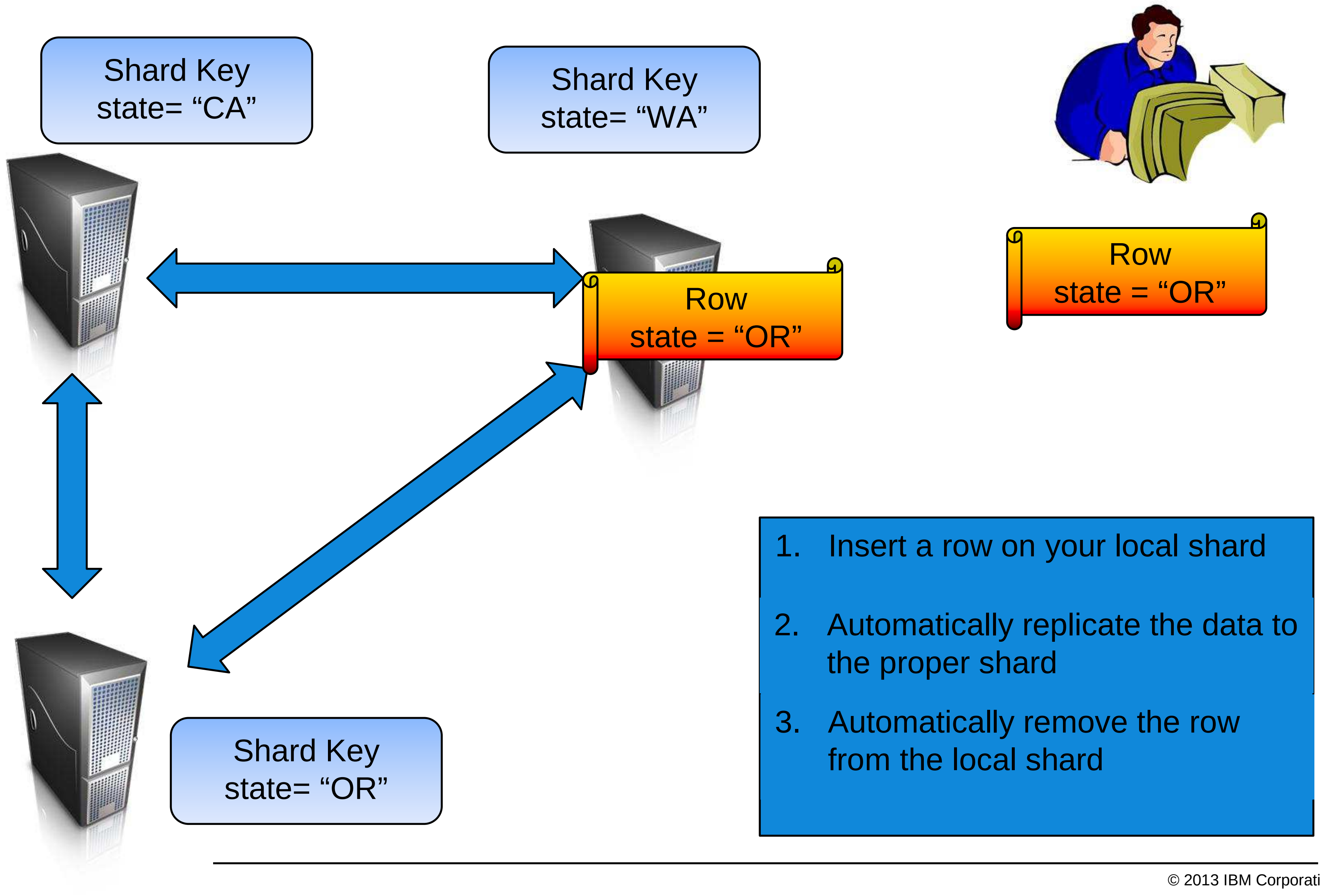
Shard Key
state= “WA”

Shard Key
state= “OR”

Find Total sold for
all states

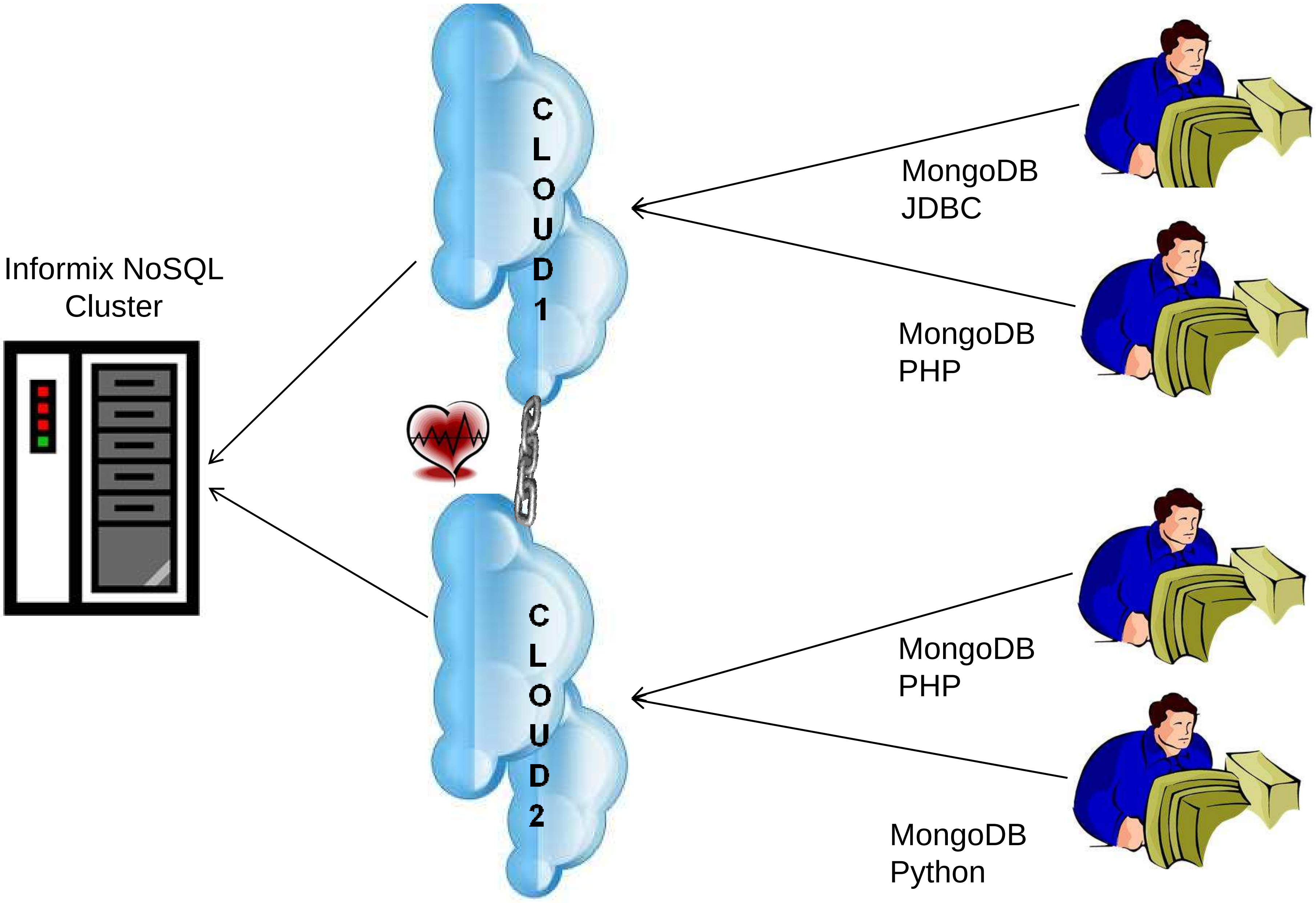
1. Request data from local shard
2. Automatically sends request to other shards requesting data
3. Returns results to client

Scaling Out - Sharded Insert



1. Insert a row on your local shard
2. Automatically replicate the data to the proper shard
3. Automatically remove the row from the local shard

Automatic Routing and Failover Coordination



Reference and Details

NoSQL/JSON Overview

Informix Core Themes to a NoSQL Solution

- **Invisible and Easy to Install and Administer**

- **Dynamic Elasticity**
  - Simple to Scale Up
  - Easy to Scale-out
  - Adding and removing nodes is simple

- **Informix Value Add Propositions**
  - Hybrid functionality (combined NoSQL and Relational)
    - Relational tables and NoSQL collections co-existing in the same database
    - Join between NoSQL and Relational tables
    - Joins utilize indexes on both Relational and NoSQL
  - Enterprise level functionality

New Functionality

Simple, Simple, Simple

Description
Auto tuning of CPU VPS
Auto Table Placement
Auto Buffer pool tuning
Auto Physical Log extension
Auto Logical Log Add
Asynchronous Sharded Deletes
Asynchronous Sharded Updates
Asynchronous Sharded Inserts
Easy Install

Informix Answers to Mobile Requirements

- **Consistent low latency, even under high load**
  - Informix has a history of handling thousands of users
    - Recent customer driven feature increased the user limit from 32,000 users to 128,000 users on a single node
    - Provide latency-consistency tradeoff knobs available
    - Ability to insert data while buffering the database transaction logging
      - Provides transactional semantics, but does not require storage persistence
      - Session/User can change the knob to/from buffered transaction logging
    - Read the data without acquiring lock
  - Early tests on old hardware shows millisecond response time

- **Schema Flexibility and Development Agility**
  - Provides JSON & BSON functionality by default
  - Adopted core MongoDB API functionality
  - Leverages Informix's history of "keeping it simple" for JSON and BSON support
  - Provides the ability to integrate relational and NoSQL data
    - Allow indexed joins between relational and NoSQL data

Informix Answers to Mobile Requirements

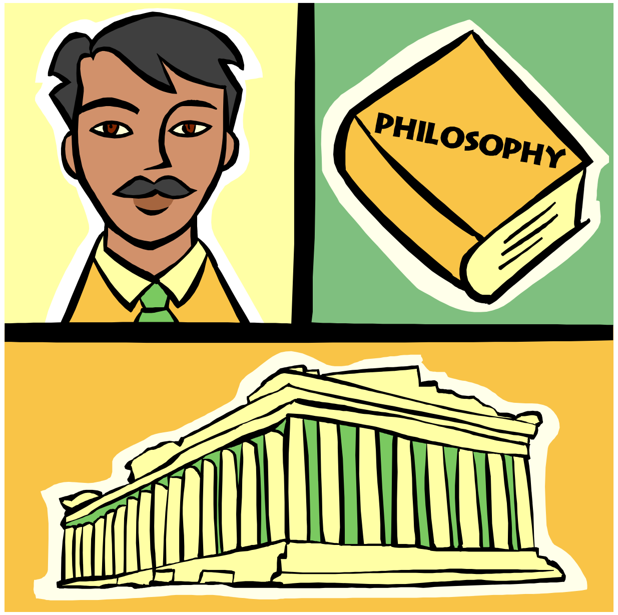
- **Continuous availability**
  - Informix Grid Replication
    - Supports servers running of different
      - Database server version
      - Operating system version
      - Machine architecture
    - Automatic resynchronization for troubled nodes
    - All functionality exists commodity hardware and software
  - Connection Manager provides
    - Connections based on policy or workload
    - Automatic re-direction for down servers

- **Dynamic Elasticity**
    - Provides a one setup for new nodes
    - When the MACH component is integrated within a Grid's node
      - Provides ondemand diskless horizontal scaling
      - Failover redundancy

- **Low cost infrastructure**
  - History of many customer running thousands of systems which exceedingly high up time and little to no DBAs

NoSQL Database Philosophy Differences

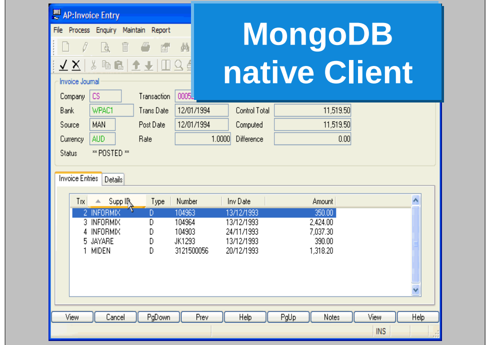
- **No ACID**
  - No ACID (Atomicity, Consistency, Isolation, Durability)
  - An eventual consistence model
- **No Joins**
  - Generally single row/document lookups
- **Flexible Schema**
  - Rigid format



High Level Architecture

- **New Wire Protocol Listener supports existing MongoDB drivers**
  - Simple port change allows applications written for MongoDB to be intercepted by wire listener
  - Compatible with all MongoDB programming interfaces
    - Java, PHP, Python, Javascript, etc.
- **The wire listener combines MongoDB messages and BSON documents to perform actions against a distributed data store**

Applications



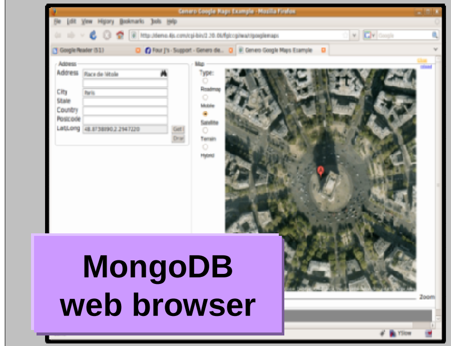
AP:Invoice Entry

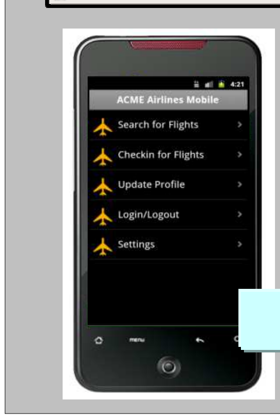
Invoice Journal

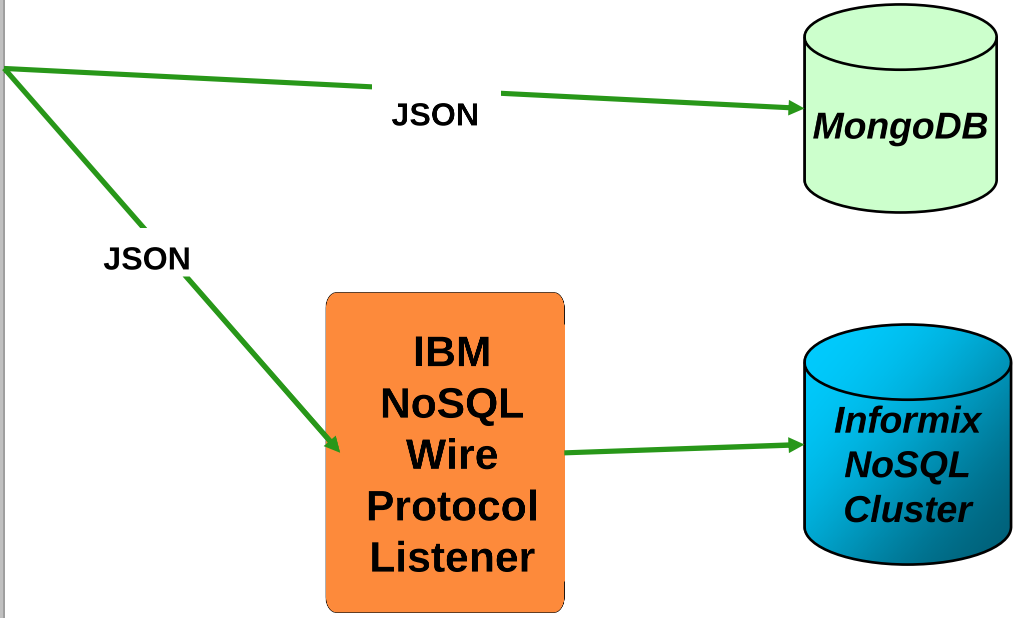
Company	CS	Transaction	0005		
Bank	WPAC1	Trans Date	12/01/1994	Control Total	11,519.50
Source	MAN	Post Date	12/01/1994	Computed	11,519.50
Currency	AUD	Rate	1.0000	Difference	0.00
Status	** POSTED **				

Invoice Entries | Details

Trx	Supp ID	Type	Number	Inv Date	Amount
2	INFORMIX	D	104963	13/12/1993	350.00
3	INFORMIX	D	104964	13/12/1993	2,424.00
4	INFORMIX	D	104903	24/11/1993	7,037.30
5	JAYARE	D	JK1293	13/12/1993	390.00
1	MIDEN	D	3121500056	20/12/1993	1,318.20







New Functionality

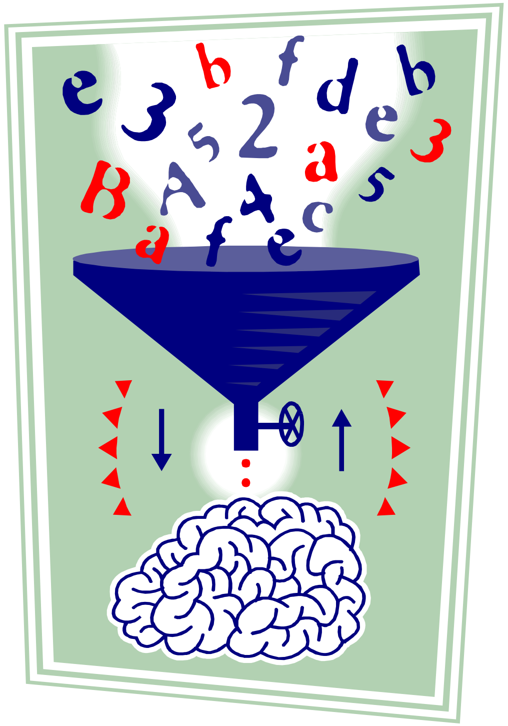
- **Add three new built-in data-types**
  - Longlvarchar
  - JSON
  - BSON

- **New data types are native to all databases**
  - Automatically convert JSON to BSON document
  - Automatically converts BSON to JSON

- **Add new Built-in BSON Functions**

- **Complete the Sharded Operations**
  - Query in 12.10.UC1
  - Insert, Delete, Update

- **Add Simplification**
  - Installation
  - Resource Allocation



New Built-in BSON Expressions

```
bson_value_double(lvarchar doc, lvarchar key) returns float
bson_value_lvarchar(lvarchar doc, lvarchar key) returns lvarchar as string
bson_value_document(lvarchar doc, lvarchar key) returns lvarchar as BSON object
bson_value_array(lvarchar doc, lvarchar key) returns lvarchar as BSON array
bson_value_binary(lvarchar doc, lvarchar key) returns lvarchar as BSON binary
bson_value_objectid(lvarchar doc, lvarchar key) returns lvarchar as string
bson_value_boolean(lvarchar doc, lvarchar key) returns boolean
bson_value_date(lvarchar doc, lvarchar key) returns datetime
bson_value_code(lvarchar doc, lvarchar key) returns lvarchar as string
bson_value_int(lvarchar doc, lvarchar key) returns bigint
bson_value_bigint(lvarchar doc, lvarchar key) returns bigint
bson_value_timestamp(lvarchar doc, lvarchar key) returns datetime
bson_key_exists(lvarchar doc, lvarchar key) returns boolean
```

JSON: JavaScript Object Notation

- **What is JSON?**
  - JSON is lightweight text-data interchange format
  - JSON is language independent
  - JSON is "self-describing" and easy to understand

- **JSON is syntax for storing and exchanging text information much like XML. However, JSON is smaller than XML, and faster and easier to parse.**

```
{
"name":"John Miller",
"age":21,
"count":27,
"employees": [
    { "firstName":"John" , "lastName":"Doe" },
    { "firstName":"Anna" , Middle”:”Marie”,"lastName":"Smith" },
    { "firstName":"Peter" , "lastName":"Jones" }
                  ]
}
```

BSON is a binary form of JSON.

Understanding Informix BSON Indexes

- **Indexes are created on BSON data and support**
  - Arrays
  - Composite Indexes
  - Unique Indexes (enforced at a single node level)
  - Primmary Key (enforced across all nodes)

```
{
"fname":"Sadler",
"lname":"Sadler",
"company":"Friends LLC",
"age":21,
"count":27,
“phone": [ “408-789-1234”, “408-111-4779” ],
}
```

```
create index fnameix1 on customer(bson_value(bson,"fname")) using bson;

create index lnameix2 on customer(bson_value(bson,"lname")) using bson;

create index phoneix3 on customer(bson_value(bson,"phone")) using bson;
```

Understanding Informix BSON Indexes

```
create index fnameix1 on customer(bson_value(bson,"fname")) using bson;

create index lnameix2 on customer(bson_value(bson,"lname")) using bson;

create index phoneix3 on customer(bson_value(bson,"phone")) using bson;
```

```
select  * from customer where  bson_value(bson,"fname") = "Ludwig";
        -- use fnameix1
select  * from customer where  bson_value(bson,"lname") = "Sadler";
        -- use lnameix2
select  * from customer where  bson_value(bson,"phone") = "408-789-8091";
        -- use phoneix3
select  * from customer where  bson_value(bson,"phone") = "415-822-1289" OR
                             bson_value(bson,"phone") = "408-789-8091";
        -- use phoneix3
select  * from customer where  bson_value(bson,"company") = "Los Altos Sports";
        -- no index use sequential scan
```

What is a NoSQL Database?

- **Not Only SQL or NOt allowing SQL**
- **A non-relational database management systems**
  - Does not require a fixed schema
  - Avoids join operations
  - Scales horizontally
  - No ACID (eventually consistent)
- **Good with distributing data and prototype project**
- **Big with web developers**

Provides a mechanism for storage and retrieval of data while providing horizontal scaling.

Basic NoSQL Terms

Term	Description
NoSQL	A class of database management systems that use some API other than SQL as the primary language. Two common features in such databases are a flexible schema, and automatic sharding and query routing across distributed nodes.
JSON	Acronym for JavaScript Object Notation – It is a text-based standard for data representation and interchange. The JSON format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application, serving as an alternative to XML.
BSON	A standardized binary representation format (see bsonspec.org) for serializing JSON documents. It allows for faster traversal of the document than when using the textual representation.

Basic Terms Translation

Mongo/NoSQL Term	Informix Term
Database	Database
Collection	Table
Document or BSON document	Row
Field	Column
Embedded documents and links	Table joins
Aggregation framework	Group by with aggregation functions

Basic Data Distribution/Replication Terms

Term	Description	Informix Term
Shard	A single node or a group of nodes holding the same data (replica set)	Instance
Replica Set	A collection of nodes contain the same data	MACH Cluster
Shard Keys	The field that dictates the distribution of the documents. Must always exist in a document.	???
Sharded Cluster	A group shards were each shard contains a portion of the data.	Grid/ER
Slave	A server which contains a second copy of the data for read only processing.	Secondary Server Remote Secondary

Basic MongoDB Operations Conceptual Operations

Mongo Action	Informix Action
db.customer.insert({ name: “John", age: 21 })	INSERT INTO customer (name, age) VALUES (“John”,21)
db.customer.find()	SELECT * FROM customer
db.customer.find({age: { $gt:21 } })	SELECT * FROM customer WHERE age > 21
db.customer.drop()	DROP TABLE customer
db.customer.ensureIndex({ name : 1, age : -1 })	CREATE INDEX idx_1 on customer(name , age DESC)
db.customer.remove({age: { $gt:21 } })	DELETE FROM customer where age > 21
db.customer.update({ age: { $gt: 20 } }, { $set: { status: “Drink" } }, { multi: true })	UPDATE customer SET status = “Drink" WHERE age > 20

JSON Details

- **JSON Syntax Rules**
  - JSON syntax is a subset of the JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

- **JSON Name/Value Pairs**
  - JSON data is written as name/value pairs.
  - A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

"name":"John Miller"

- **JSON Values can be**
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An array (in square brackets)
  - An object (in curly brackets)
  - Null

Some Typical NoSQL Use Cases

Mostly Interactive Web/Mobile

- **Online/Mobile Gaming**
  - Leaderboard (high score table) management
  - Dynamic placement of visual elements
  - Game object management
  - Persisting game/user state information
  - Persisting user generated data (e.g. drawings)
- **Display Advertising on Web Sites**
  - Ad Serving: match content with profile and present
  - Real-time bidding: match cookie profile with ad inventory, obtain bids, and present ad
- **Dynamic Content Management and Publishing (News & Media)**
  - Store content from distributed authors, with fast retrieval and placement
  - Manage changing layouts and user generated content
- **E-commerce/Social Commerce**
  - Storing frequently changing product catalogs
- **Social Networking/Online Communities**
- **Communications**
  - Device provisioning
- **Logging/message passing**
  - Drop Copy service in Financial Services (streaming copies of trade execution messages into (for example) a risk or back office system)

IBM Use Case Characteristics

- **Consistent low latency, even under high loads**
  - Ability to handle thousands of users
  - Typically millisecond response time
- **Schema flexibility and development agility**
  - Application not constrained by fixed pre-defined schema
  - Ability to handle a mix of structured and unstructured data
- **Continuous availability**
  - 24x7x365 availability
  - Online maintenance operations
  - Ability to upgrade hardware or software without down time
- **Dynamic Elasticity**
  - Rapid horizontal scalability
  - Ability to add or delete nodes dynamically in the grid
  - Application transparent elasticity
- **Low cost infrastructure**
  - Commonly available hardware (Windows & Linux,…)
- **Reduced need for database administration and maintenance**

Why Most Commercial Relational Databases cannot meet these Requirements

- **Consistent Low Latency, even under high load**
  - ACID requirements inherently introduce write latency
  - There is no latency-consistency tradeoff knobs available
  - Requirement can be met, but at a much higher cost (hardware, software or complexity)
- **Schema Flexibility & Development Agility**
  - Relational schemas are inherently rigid
  - Database design needs to be done upfront
  - Different rows cannot have a different structuree
  - Database design needs to be done before application is developed
  - Data modeling based on domain objects, which may not be well understood upfront
- **High Availability**
  - Requirement can be met, but at a significant cost
  - Typically hardware and software upgrades require some downtime
  - Typically rolling version upgrades are complex in clustered RDBMS
- **Dynamic Elasticity**
  - Not a natural fit for RDBMS, due to requirement for strong consistency
  - Scale-out requires partition tolerance, that increases latency
- **Low Cost**
  - Distributed RDBMS typically require specialized hardware to achieve performance
  - Popular relational databases typically require several DBAs for maintenance and tuning