# NoSQL Application Development with JSON and MapR-DB

Dale Kim, Sr. Director, Industry Solutions

Tug Grall, Technical Evangelist

August 24, 2016

# Our Speakers

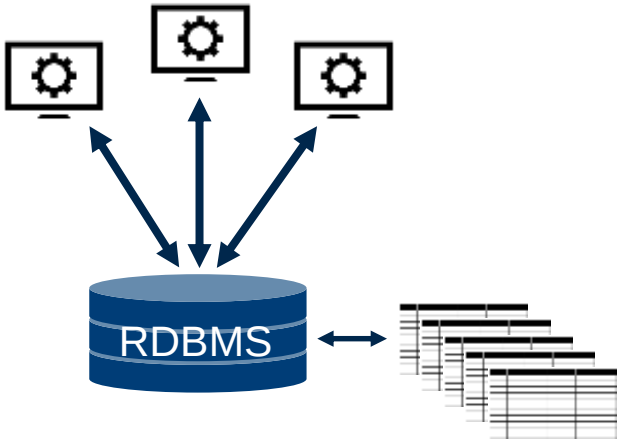
Dale Kim
Sr. Director, Industry Solutions


Tugdual Grall
Technical Evangelist

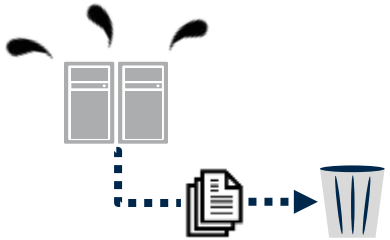
# Quick NoSQL Overview

# Relational Databases Were Not Designed for Big Data

- RDBMSs are the default choice for applications
  - But large, rapidly changing, and/or diverse data sets add cost/time pressures

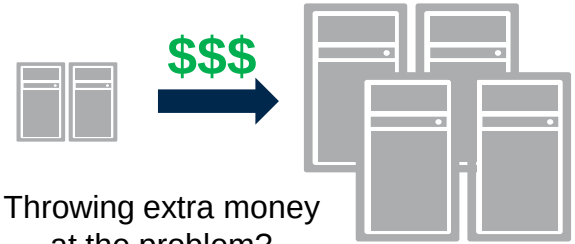



- This forces trade-offs with your data



Throwing away data to preserve performance?

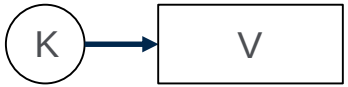
- Or significant costs



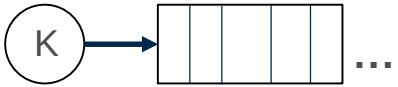

Throwing extra money at the problem?

# The Common NoSQL Data Models

Key-Value



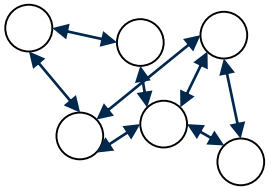

Wide Column




Document (JSON)

```
{
  “fname”: “John”,
  “lname”: “Doe”
}
```

Graph



JSON documents are popular because they easily model:

- Hierarchical/nested data
- Evolving data
- Varying data

# Let’s Dive into JSON

# JSON is a Powerful Construct

```
function Thing()
{
// public
  this.get_name = function() {
    return name;
  }

  this.set_name = function(tname) {
    name = tname;
  }

  this.increment = (function () {
    var count = 0;
    return function () {return ++count;}
  })();


// private
  var name = null;
}
```

```
var thing = new Thing();

thing.set_name("John");

// alert(thing.name); //error, private
variable

var counter = thing.counter;
```

# JSON Allows Easy Variation across Records

```
{
  "_id" : "rp-prod132546",
  "name" : "Marvel T2 Athena”,
  "brand" : "Pinarello",
  "category" : "bike",
  "type" : "Road Bike”,
  "price" : 2949.99,

  "size" : "55cm",
  "wheel_size" : "700c",
  "frameset" : {
            "frame" : "Carbon Toryaca",
            "fork" : "Onda 2V C"
  },
  "groupset" : {
            "chainset" : "Camp. Athena 50/34",
            "brake" : "Camp."
  },
  "wheelset" : {
            "wheels" : "Camp. Zonda",
            "tyres" : "Vittoria Pro"
  }
}
```

bike

```
{
  "_id" : "rp-prod106702",
  "name" : " Ultegra SPD-SL 6800”,
  "brand" : "Shimano",
  "category" : "pedals",
  "type" : "Components,
  "price" : 112.99,

  "features" : [
            "Low profile design increases ...",
            "Supplied with floating SH11 cleats",
            "Weight: 260g (pair)"
  ]
}
```

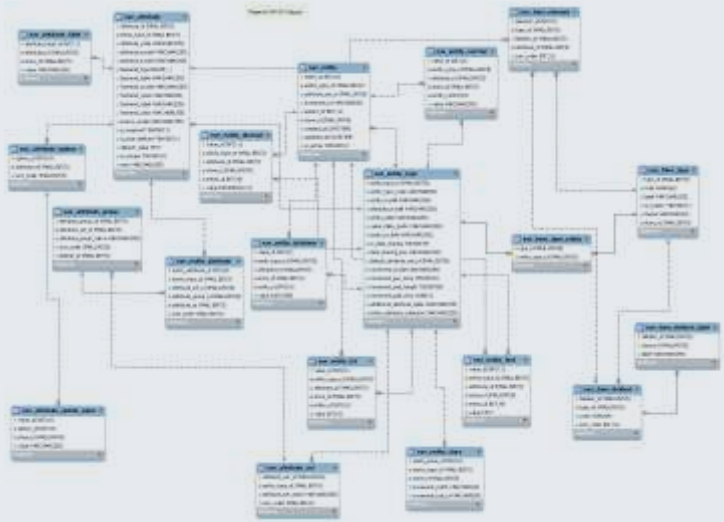
pedal

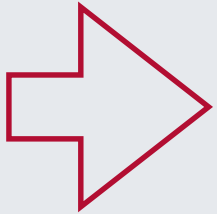
```
{
  "_id" : "rp-prod113104",
  "name" : "Bianchi Pride Jersey SS15”,
  "brand" : "Nalini",
  "category" : "Jersey",
  "type" : "Clothing,
  "price" : 76.99,

  "features" : [
            "100% Polyester",
            "3/4 hidden zip",
            "3 rear pocket"
  ],
  "color" : "black"
}
```


jersey

# How Variability Is Handled in an RDBMS





```
SELECT * FROM (
    SELECT
        ce.sku,
        ea.attribute_id,
        ea.attribute_code,
        CASE ea.backend_type
           WHEN 'varchar' THEN ce_varchar.value
           WHEN 'int' THEN ce_int.value
           WHEN 'text' THEN ce_text.value
           WHEN 'decimal' THEN ce_decimal.value
           WHEN 'datetime' THEN ce_datetime.value
           ELSE ea.backend_type
        END AS value,
        ea.is_required AS required
    FROM catalog_product_entity AS ce
    LEFT JOIN eav_attribute AS ea
        ON ce.entity_type_id = ea.entity_type_id
    LEFT JOIN catalog_product_entity_varchar AS ce_varchar
        ON ce.entity_id = ce_varchar.entity_id
        AND ea.attribute_id = ce_varchar.attribute_id
        AND ea.backend_type = 'varchar'
   LEFT JOIN catalog_product_entity_text AS ce_text
        ON ce.entity_id = ce_text.entity_id
        AND ea.attribute_id = ce_text.attribute_id
        AND ea.backend_type = 'text'
    LEFT JOIN catalog_product_entity_decimal AS ce_decimal
        ON ce.entity_id = ce_decimal.entity_id
        AND ea.attribute_id = ce_decimal.attribute_id
        AND ea.backend_type = 'decimal'
    LEFT JOIN catalog_product_entity_datetime AS ce_datetime
        ON ce.entity_id = ce_datetime.entity_id
        AND ea.attribute_id = ce_datetime.attribute_id
        AND ea.backend_type = 'datetime'
    WHERE ce.sku = 'rp-prod132546'
  ) AS tab
  WHERE tab.value != '';
```

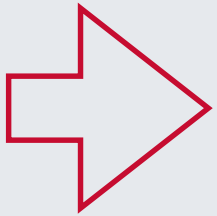
"Entity Value Attribute" pattern

To get a single product

# Product Catalog - NoSQL/Document

```
{
  "_id" : "rp-prod132546",
  "name" : "Marvel T2 Athena”,
  "brand" : "Pinarello",
  "category" : "bike",
  "type" : "Road Bike”,
  "price" : 2949.99,
```

```
  "size" : "55cm",
  "wheel_size" : "700c",
  "frameset" : {
        "frame" : "Carbon Toryaca",
        "fork" : "Onda 2V C"
  },
  "groupset" : {
        "chainset" : "Camp. Athena 50/34",
        "brake" : "Camp."
  },
  "wheelset" : {
        "wheels" : "Camp. Zonda",
        "tyres" : "Vittoria Pro"
  }
}
```



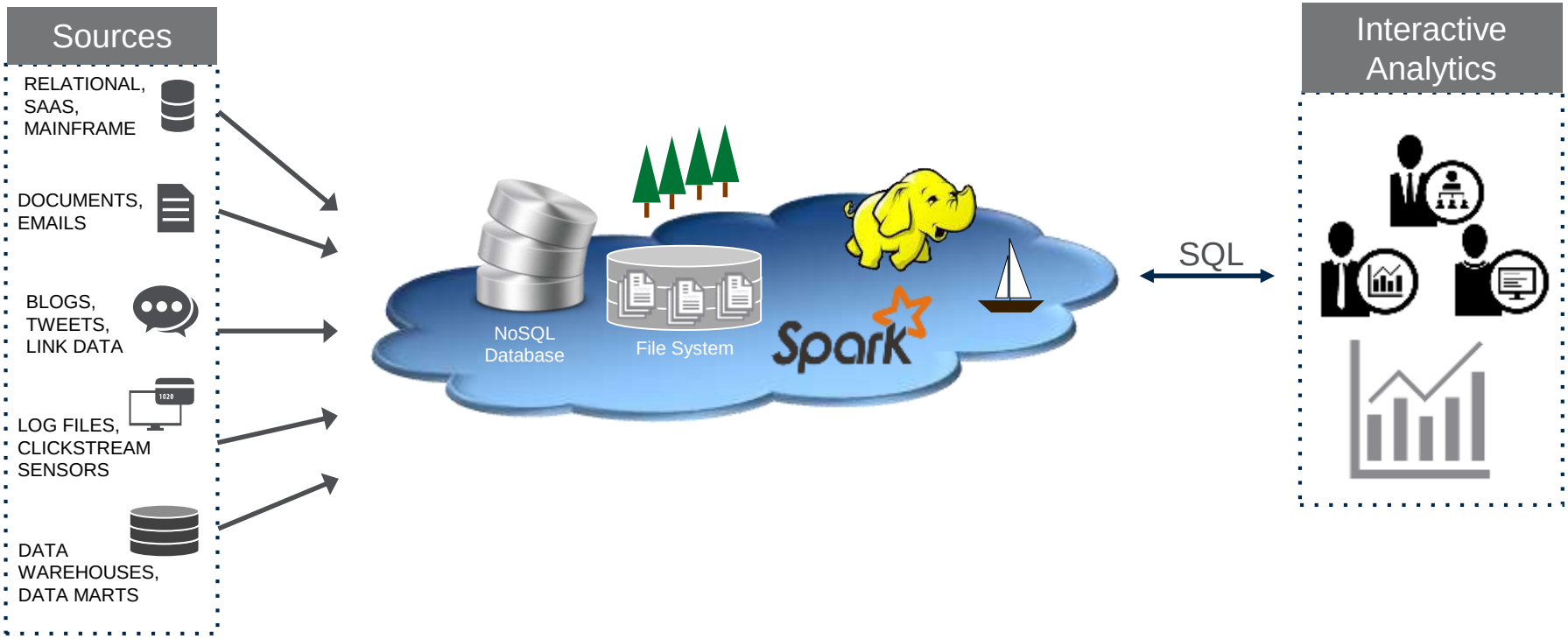
```
products
  .findById(“rp-prod132546”)
```

Store the product “as a business object”

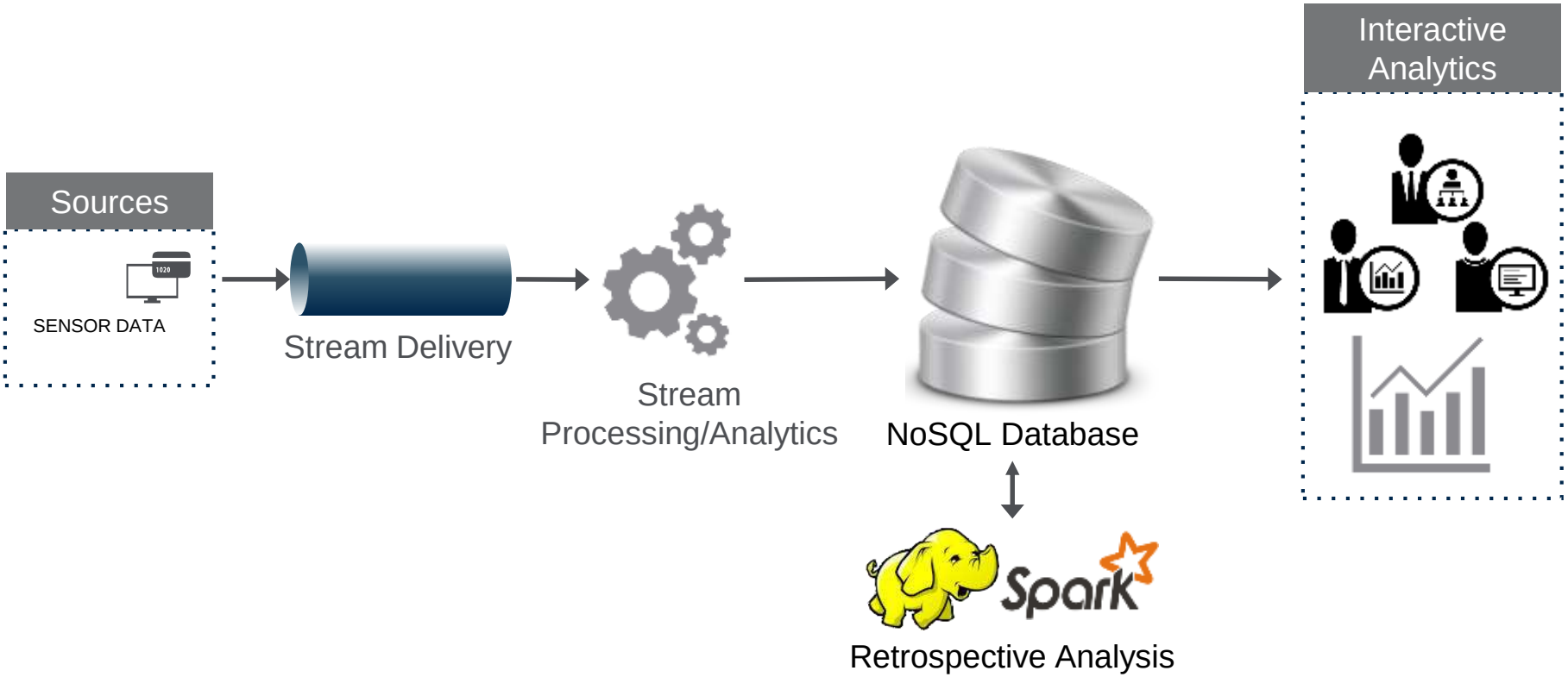
To get a single product

# Example Use Cases for JSON/NoSQL

# Example Use Case 1: Data Lake



# Example Use Case 2: IoT and Predictive Analytics



# What Should Native JSON Support Include?

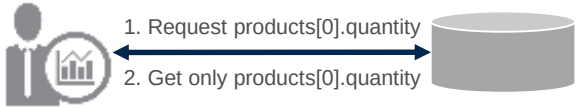
# Recognize JSON Elements at Server-Side

```
{
    order_num: 5555,
    products: [
       {   product_id: 348752,
           quantity: 1,
           unit_price: 149.99,
           total_price: 149.99
       },
       {   product_id: 439322,
           quantity: 1,
           unit_price: 99.99,
           total_price: 99.99
       },
       {   product_id: 953923,
           quantity: 1,
           unit_price: 49.99,
           total_price: 49.99
       },
    ]
}
```

Reads/writes at element level

- Granular disk reads/writes
- Less network traffic
- Higher concurrency


versus




Any new elements added on demand

- No predefined schemas
- Easy to store evolving data

# Organize Elements for Different Policies

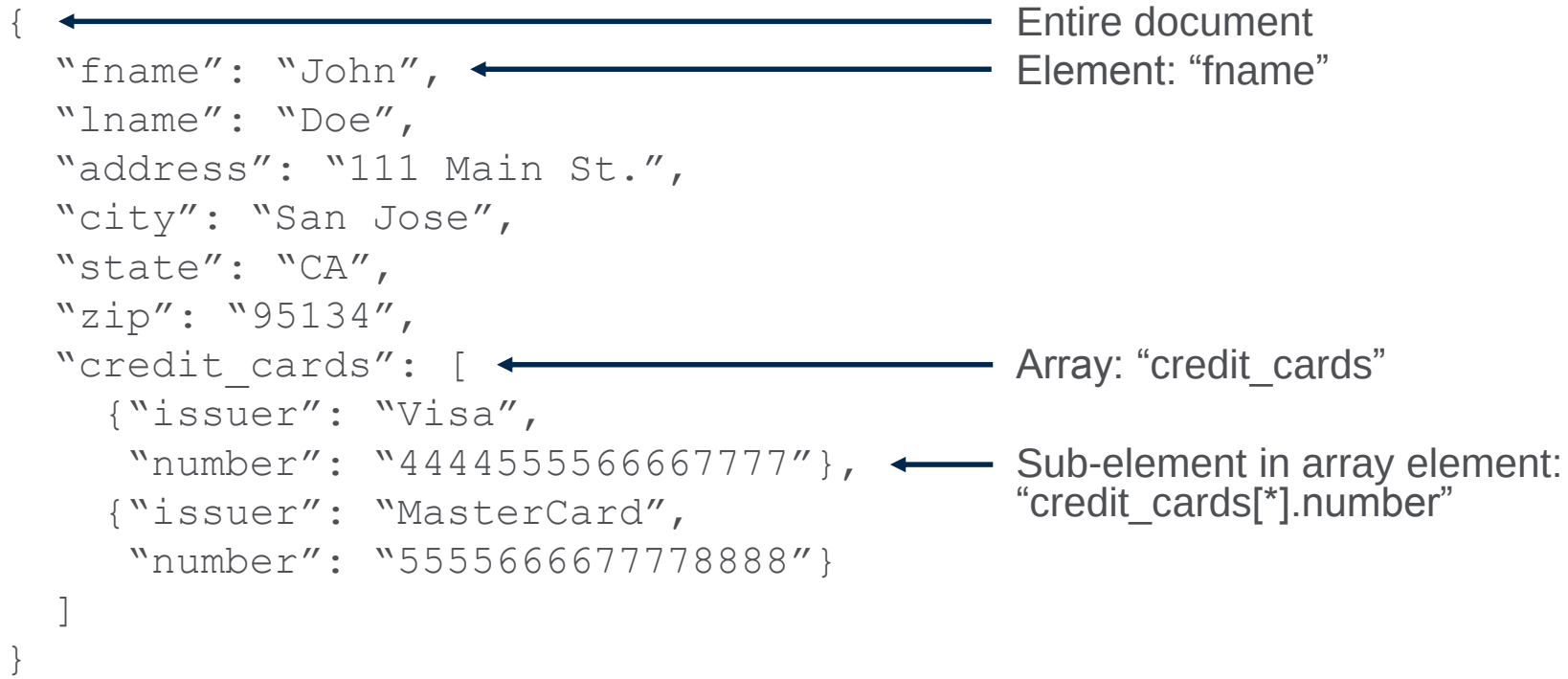
```
/
{a:
  {a1:
    {b1: "v1",
     b2: [
       {c1: "v1",
        c2: "v2"}
     ]
    },
   a2:
    {
      e1: "v1",
      e2: <inline jpg>
    }
  }
}
```

- Control layout for faster data access
- Different TTL requirements
- Separate replication settings
- Efficient access controls

Column Family 1

Column Family 2

# Fine Grained Security within JSON Document




Specify different permissions levels within the document.

# Extensions for Comprehensive Data Type Support

- NULL
- Boolean
- String
- Map
- Array
- Float, Double
- Binary
- Byte, Short, Int, Long
- Date
- Decimal
- Interval
- Time
- Timestamp

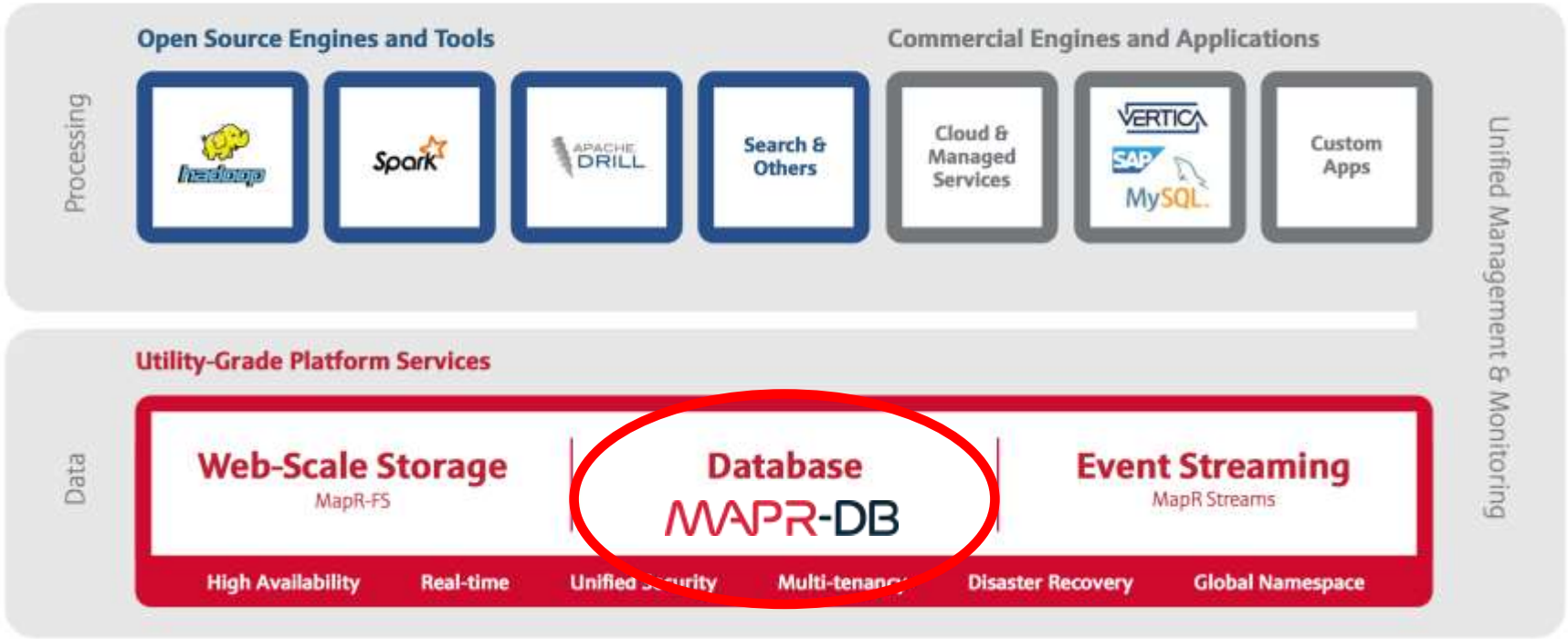
Examples:

```
{
  “sample_int”: {"$numberLong”: 2147483647},
  “sample_date”: {“$dateDay”: “2016-02-22”},
  “sample_decimal”:{“$decimal”: “1234567890.23456789”},
  “sample_time”: {“$time”: “10:26:12.487”},
  “sample_timestamp”: {“$date”: “2016-02-22T10:26:12.487+Z”}
}
```

# How Does This Relate to MapR?

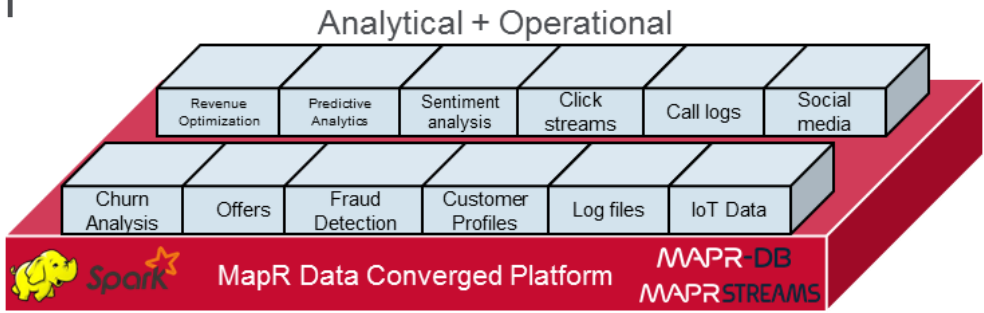


# MapR Converged Data Platform



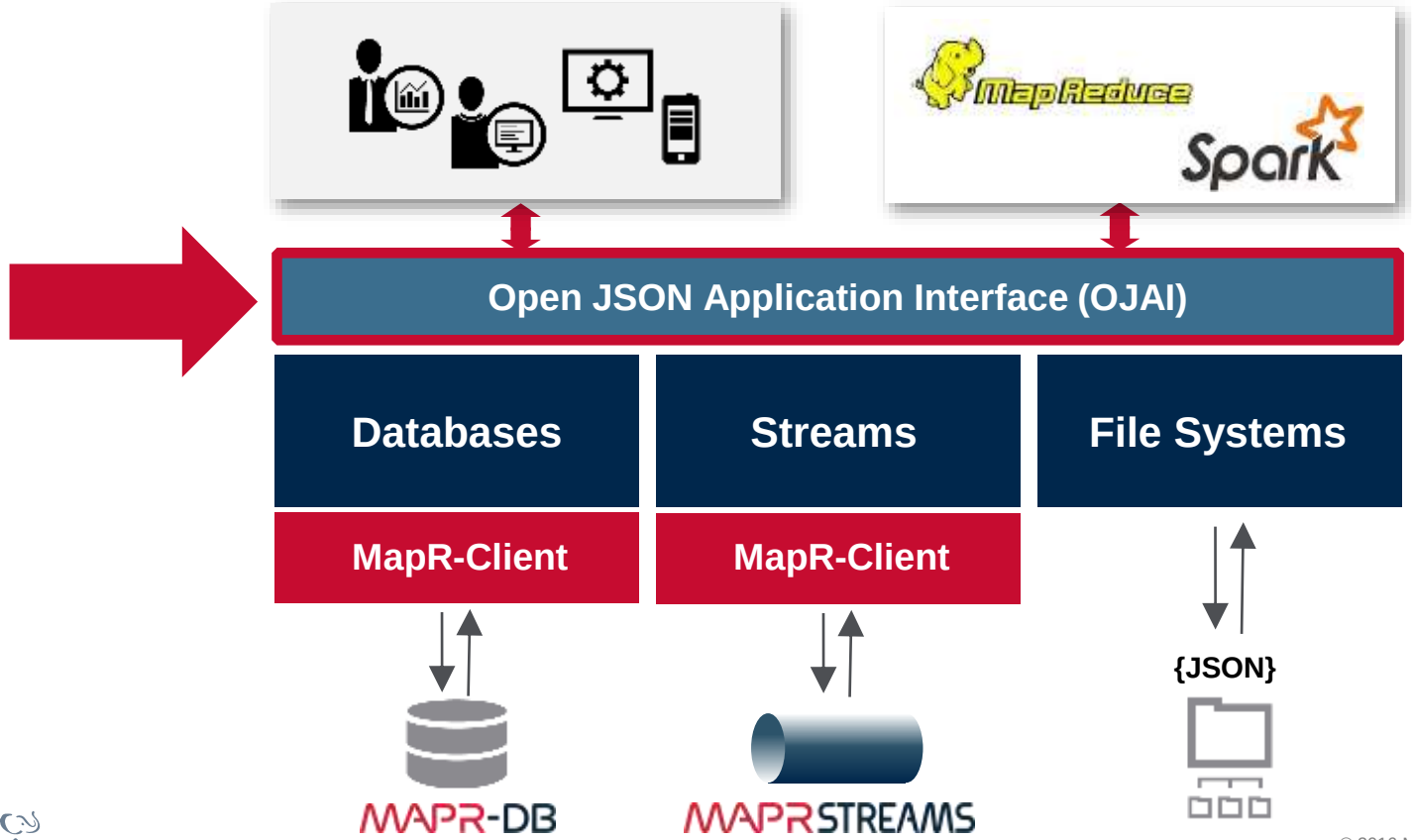
# A Single Application Development Platform

- Business applications and analytics on the same platform
  - No data movement
  - Real-time access to data
- Architectural simplicity
- Optimized stack for greater efficiency
- “Hadoop-scale” for multi-petabyte deployments




Analytics as it happens, no cross-cluster copying

# Open Source OJAI API for JSON-Based Applications



# Familiar JSON Paradigm – Similar API Constructs

## MapR-DB

```
Document record = Json.newDocument()
    .set("firstName", "John")
    .set("lastName", "Doe")
    .set("age", 50);

table.insert("jdoe", record);
```

## Other Document Database

```
BasicDBObject doc = new BasicDBObject
  ("firstName", "John")
  .append("lastName", "Doe")
  .append("age", 50);

coll.insert(doc);
```

# Multi-Master Global Deployments


Real-Time Replication

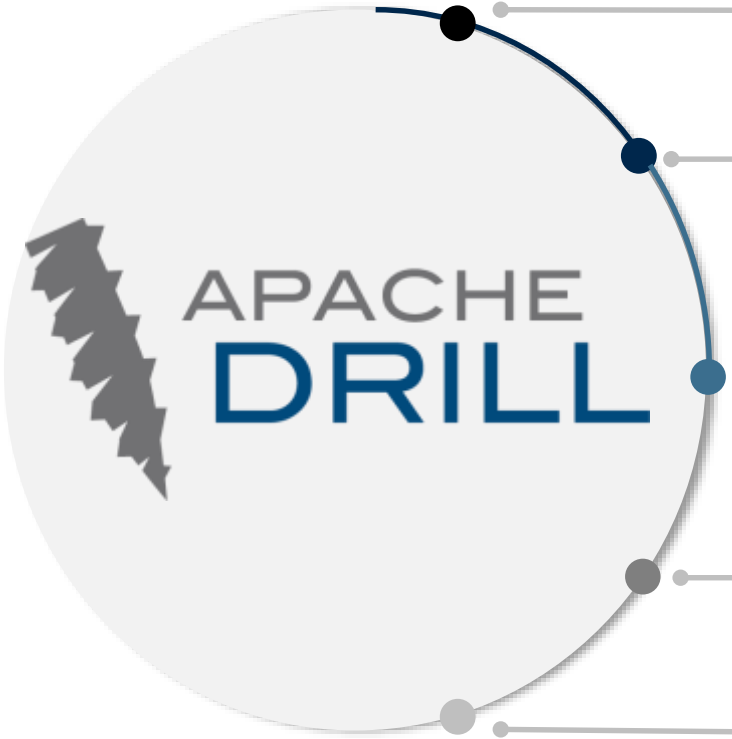
Active Read/Write

End Users

## Active-active replication

- **Faster data access** – minimize network latency on global data with local clusters
- **Reduced risk of data loss** – real-time, bi-directional replication for synchronized data across active clusters
- **Application failover** – upon any cluster failure, applications continue via redirection to another cluster

# Apache Drill for SQL Querying on MapR-DB




**Access to *any* data type, *any* data source**

- Relational
- Nested data
- Schema-less

**Rapid time to insights**

- Query data in-situ
- *No Schemas* required
- Easy to get started

**Integration with existing tools**

- *ANSI* SQL
- BI tool integration

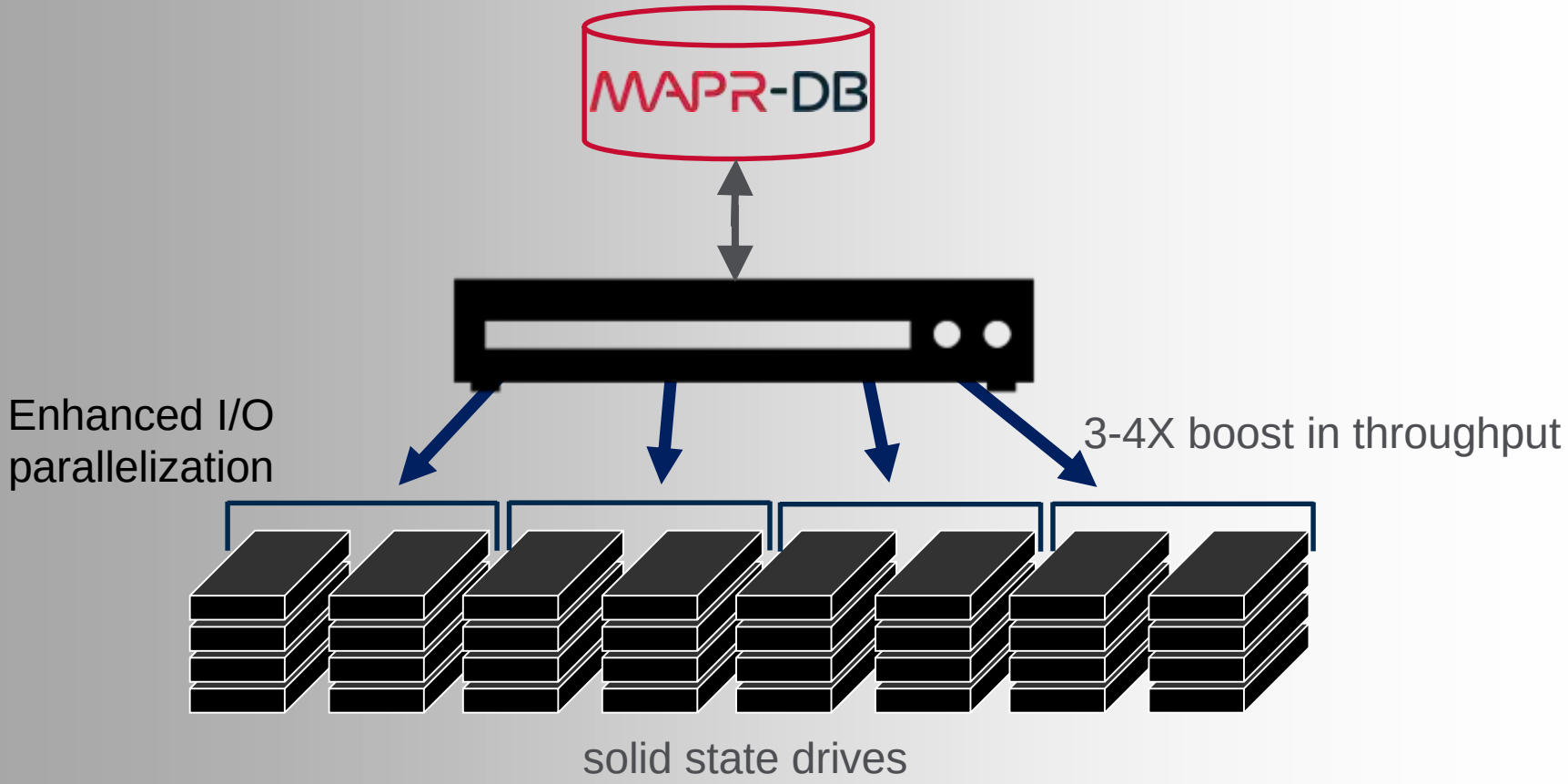
**Scale in all dimensions**

- TB-PB of scale
- 1000's of users
- 1000's of nodes

**Granular security**

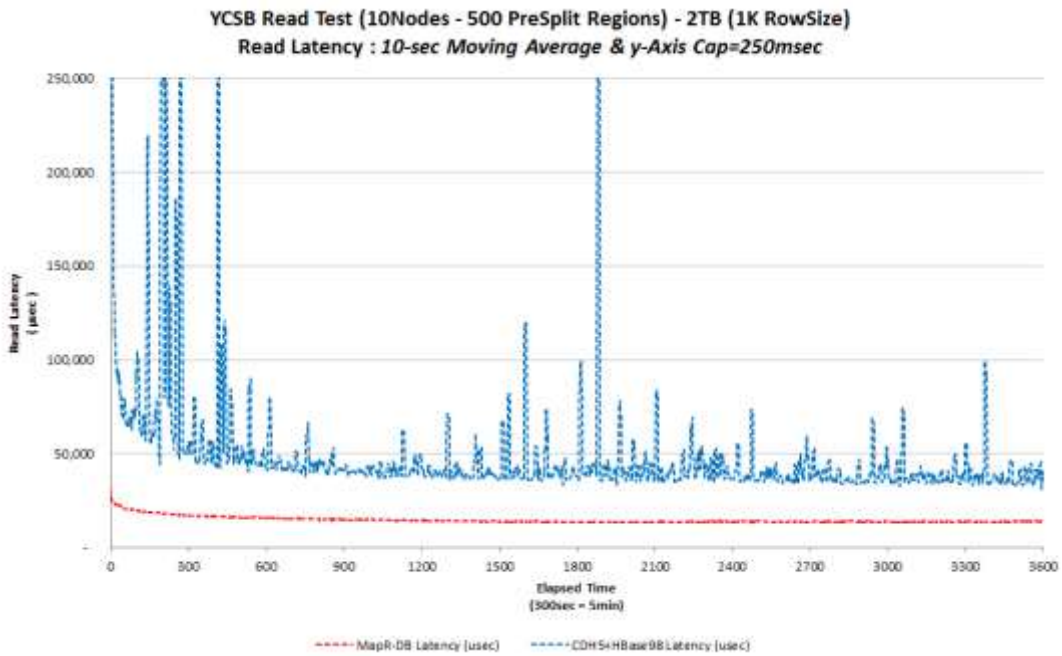
- Authentication
- Row/column level controls
- De-centralized

# Flash/SSD/NVMe Optimization



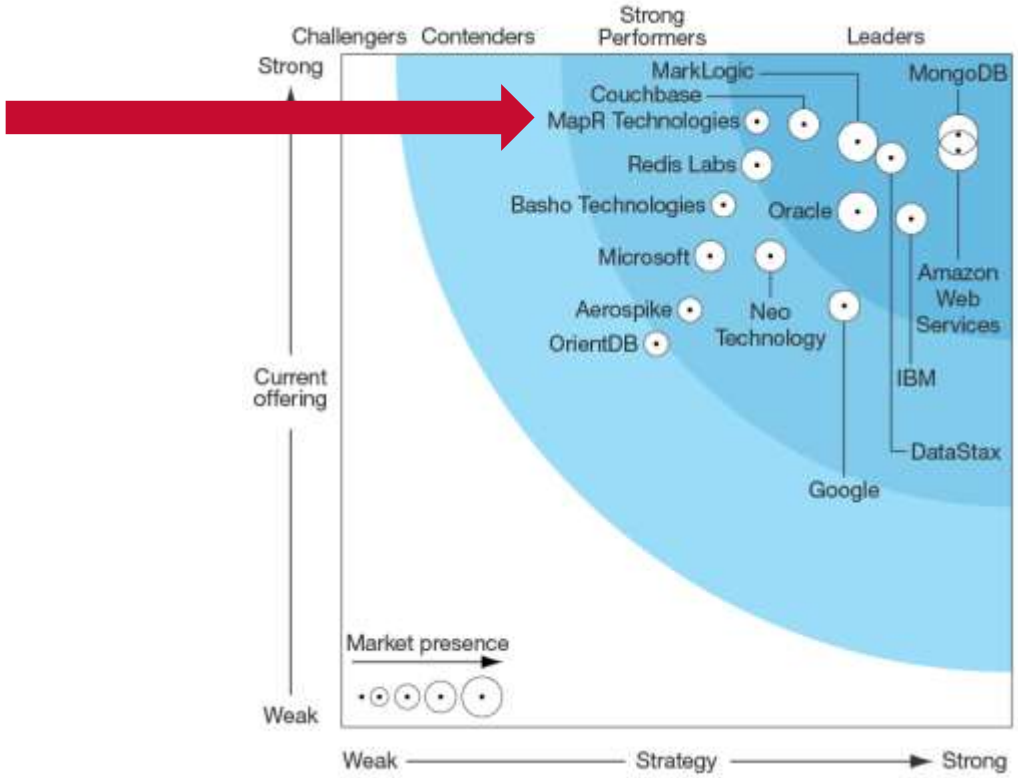
# Fast, Automatic Optimizations

- No manual cleanup tasks
- Automatic sharding
- No compaction delays
- No anti-entropy task needed to sync replicas (strong consistency)




Red plots show MapR-DB response times.
Blue plots show other NoSQL response times.

# Independent Third-Party Evaluation Results




The Forrester Wave™: Big Data NoSQL, Q3 2016

# Demo

# Q&A

## Engage with us!

1. MapR-DB
   Learn more and get started using MapR-DB
   https://www.mapr.com/products/mapr-db-in-hadoop-nosql

   Free on-demand training:
   https://www.mapr.com/ODT

2. Ask Questions:
   - **Ask Us Anything about MapR-DB in the MapR Community now through Fri (Aug 26)**
   - **https://community.mapr.com/**

CONVERGE community
Powered by MapR