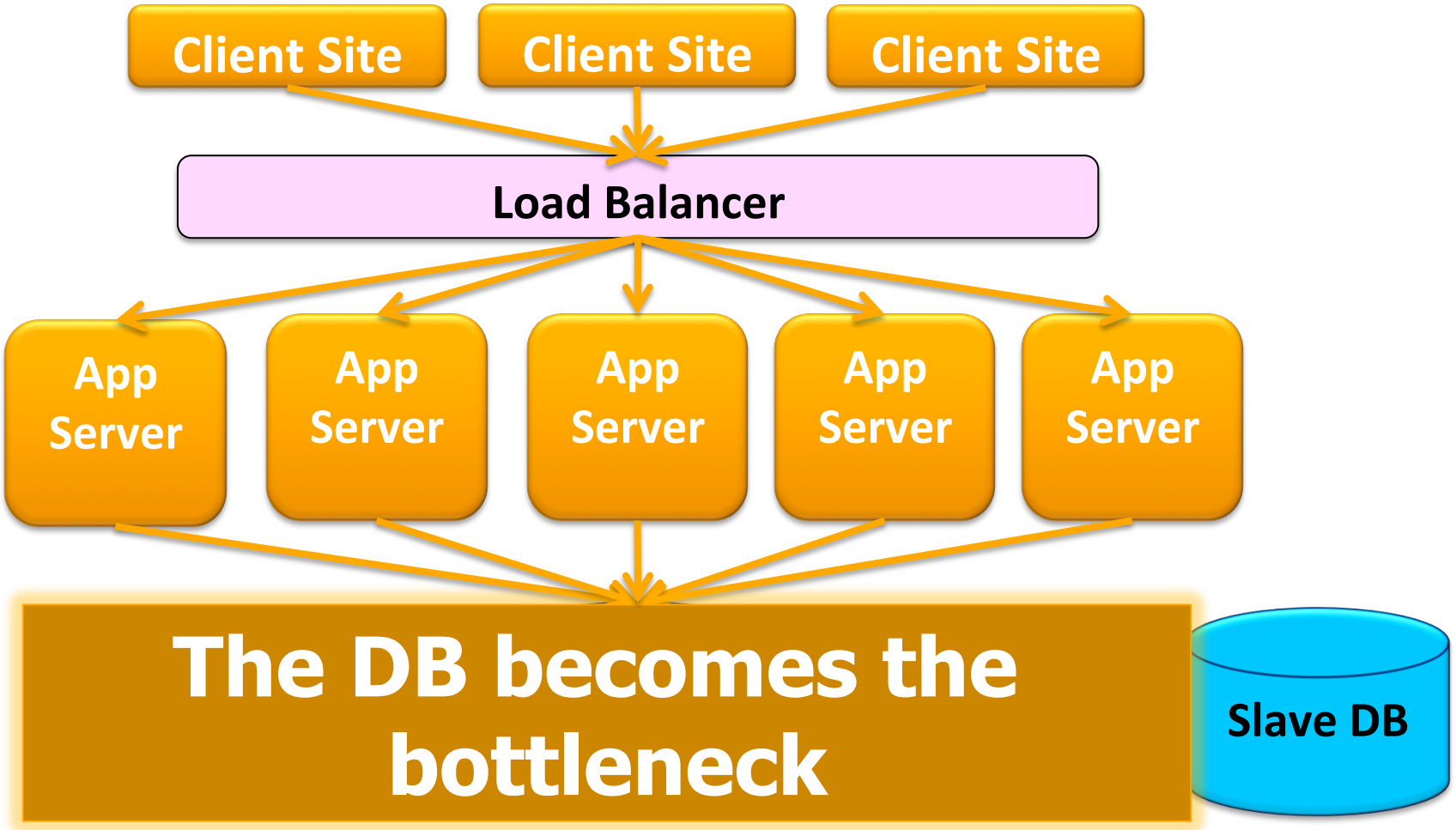
NoSQL and NewSQL

1. Motivations
2. The CAP theorem
3. NoSQL
4. NewSQL
5. Which DBMS for which requirements?

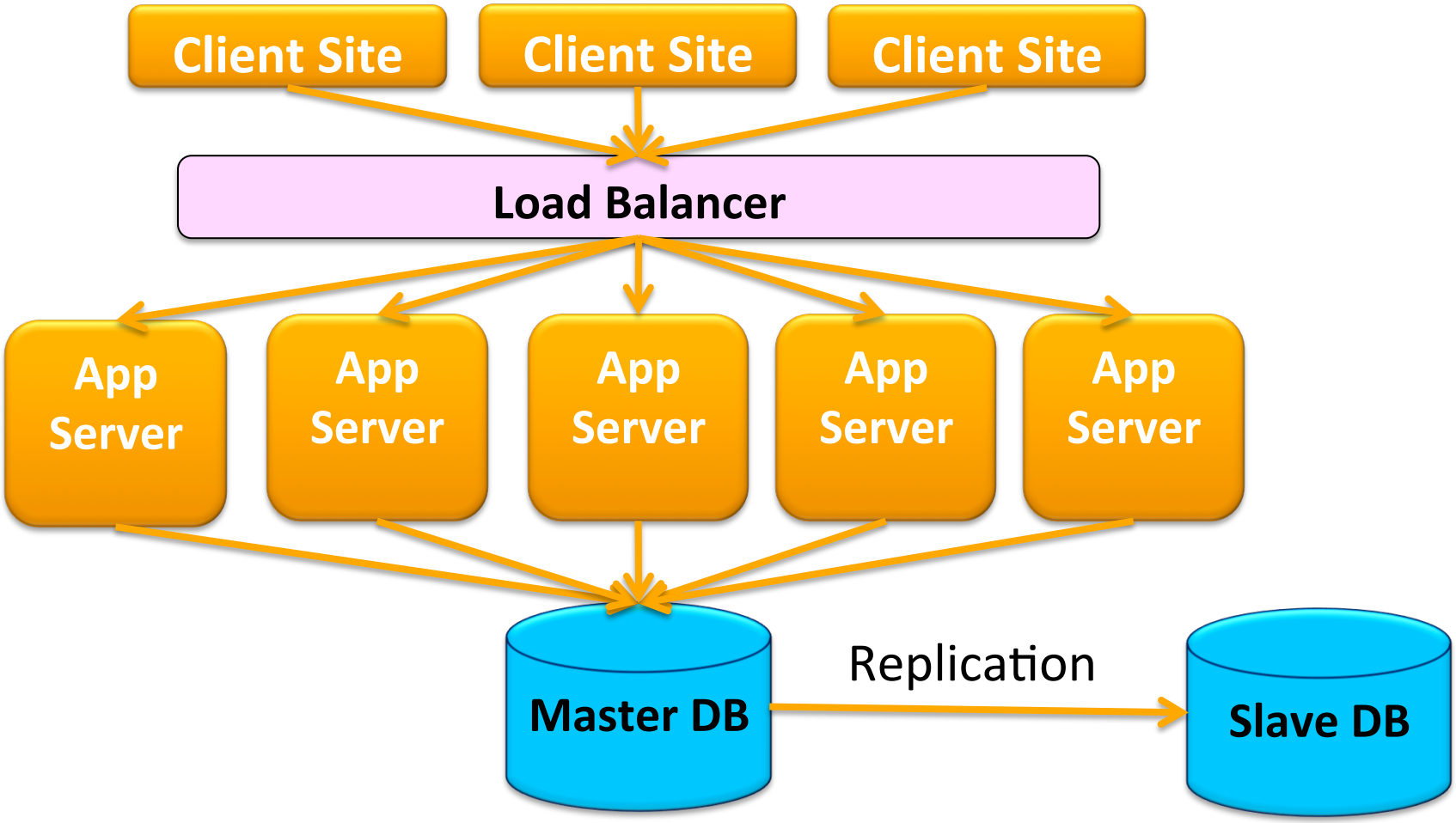
1. Motivations

- Trends
  - Big data
    - Unstructured data
  - Data interconnection
    - Hyperlinks, tags, blogs, etc.
  - Very high scalability
    - Data size, data rates, numbers of concurrent users,
- Limits of RDBMSs (SQL)
  - Need for skilled DBA and well-defined schemas
  - SQL and complex tuning
  - Hard to make updates scalable
    - Parallel RDBMS use a shared-disk for OLTP
      - Scale logarithmically and shared-disk has to be scaled up

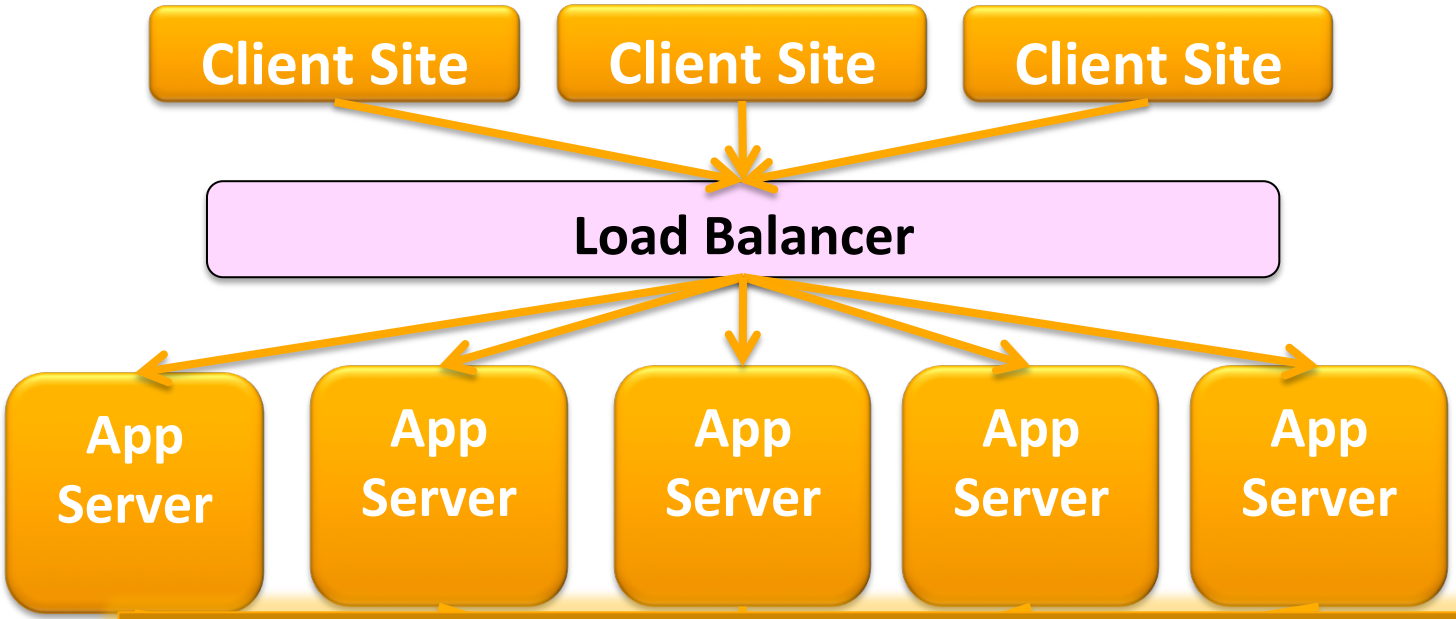
Database in the Cloud



Database in the Cloud



Scalability in the Cloud



Scalability and extensibility
But limited consistency

2. The CAP Theorem

- Polemical topic
  - "A database can't provide consistency AND availability during a network partition"
  - Wrong argument used by NoSQL to excuse their lack of ACID properties
  - Nothing to do with scalability
- Two different points of view
  - Relational databases
    - Consistency is essential
      - ACID transactions
  - Distributed systems
    - Service availability is essential
      - Inconsistency tolerated by the user, e.g. web cache

What is the CAP Theorem?

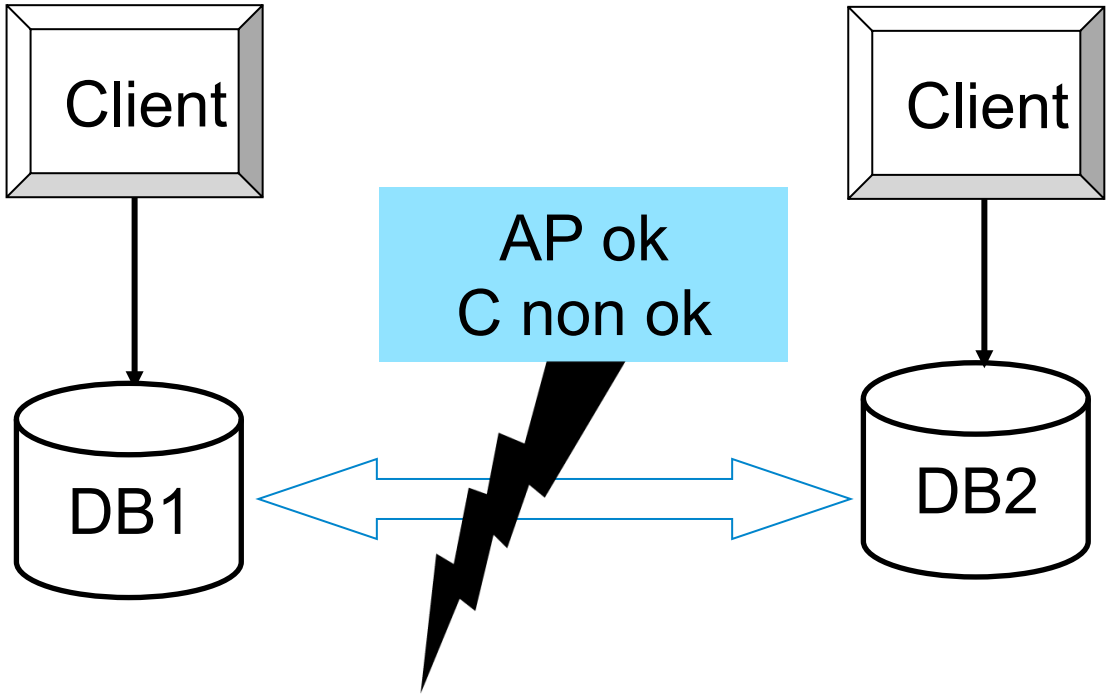
- The desirable properties of a distributed system
  - ***C****onsistency*: all nodes see the same data values at the same time
  - ***A****vailability*: all requests get an answer
  - ***P****artition tolerance*: the system keeps functioning in case of network failure
- History
  - At the PODC 2000 conference, Brewer (UC Berkeley) conjectures that one can have only two properties at the same time
  - In 2002, Gilbert and Lynch (MIT) prove the conjecture, which becomes a theorem

Strong vs Eventual Consistency

- Strong consistency (ACID)
  - All nodes see the same data values at the same time
- Eventual consistency
  - Some nodes may see different data values at the same time
  - But if we stop injecting updates, the system reaches strong consistency

- Illustration with symmetric, asynchronous replication in databases

Symmetric, Asynchronous Replication



But we have eventual consistency

- After reconnection (and resolution of update conflicts), consistency can be obtained

3. NoSQL (Not Only SQL): definition

- Specific DBMS: for web-based data
  - Specialized data model
    - Key-value, table, document, graph
  - Trade relational DBMS properties
    - Full SQL, ACID transactions, data independence
  - For
    - Simplicity (schema, basic API)
    - Scalability and performance
    - Flexibility for the programmer (integration with programming language)

- NB: SQL is just a language and has nothing to do with the story

NoSQL Approaches

- Characterized by the data model, in increasing order of complexity:
  1. Key-value: DynamoDB
  2. Tabular: Hbase
  3. Document: MongoDB
  4. Graph: Neo4J
  5. Multimodel: OrientDB

- What about object DBMS or XML DBMS?
  - Were there much before NoSQL
  - Sometimes presented as NoSQL
  - But not really scalable

3.1. Key-value Stores

- Simple (key, value) data model
  - Key = unique id
  - Value = a text, a binary data, structured data, etc.
- Simple queries
  - Put (key, value)
    - Inserts a (key, value) pair
  - Value = get (key)
    - Returns the value associated with key
  - {(key, value)} = get_range (key1, key2)
    - Returns the data whose key is in interval [key1, key2]

Amazon DynamoDB

- Major service of AWS for data storage
  - E.g. product lists, shopping carts, user preferences
- Data model (key, structured value)
  - Partitioning on the key and secondary indices on attributes
  - Simple queries on key and attributes
  - Flexible: no schema to be defined (but automatically inferred)
- Consistency
  - Eventual consistent reads (default)
  - Strong consistent reads
  - Atomic updates with atomic counters
- High availability and fault-tolerance
  - Synchronous replication between data centers
- Integration with other AWS services
  - Identity control and access
  - MapReduce
  - Redshift data warehouse

DynamoDB – data model

- Table (items)
- Item (key, attributes)
  - 2 types of primary (unique) keys
    - Hash (1 attribute)
    - Hash & range (2 attributes)
  - Attributes of the form "name":"value"
    - Type of value: scalar, set, or JSON
- Java API with methods
  - Add, update, delete item
  - GetItem: returns an item by primary key in a table
  - BatchGetItem: returns the items of same primary key in multiple tables
  - Scan : returns all items
  - Query
    - Range on hash & range key
    - Access on indexed attribute

Table: Forum_Thread

Forum	Subject	Date of last post	Tags
"S3"	"abc"	"2017 ..."	"a" "b"
"S3"	"acd"	"2017 ..."	"c"
"S3"	"cbd"	"2017 ..."	"d" "e"
"RDS"	"xyz"	"2017 ..."	"f"
"EC2"	"abc"	"2017 ..."	"a" "e"
"EC2"	"xyz"	"2017 ..."	"f"

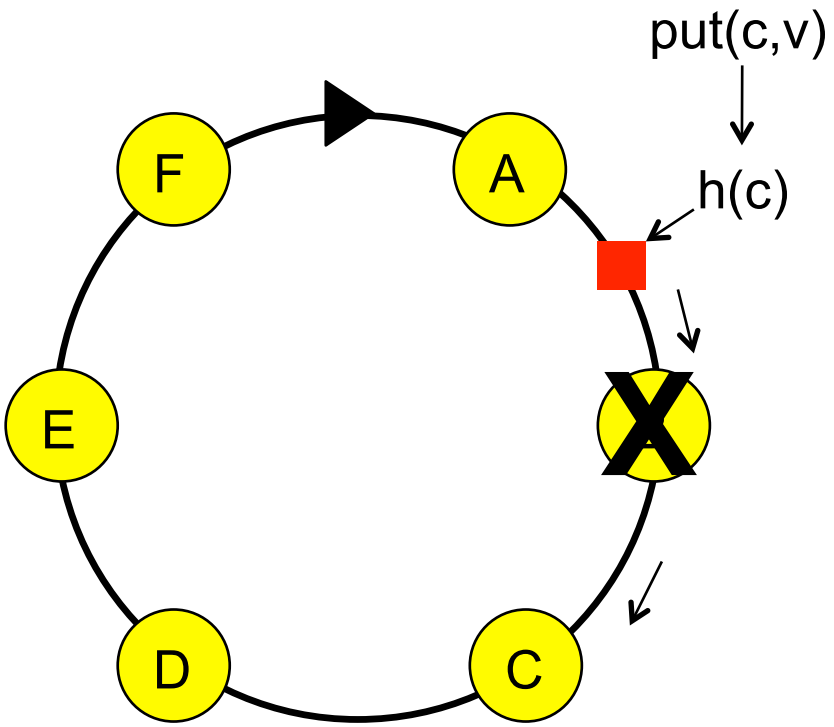
Hash key — Range key

GetItem (Forum="EC2", Subject="xyz")

Query (Forum="S3", Subject > "ac")

DynamoDB - data partitioning

- Consistent hashing: the interval of hash values is treated as a ring
- Advantage: if a node fails, its successor takes over its data
  - No impact on other nodes
- Data is replicated on next nodes



Node B is responsible for the hash value interval (A,B]. Thus, item (c,v) is assigned to node B.

3.2. Document Stores

- Main applications
  - Document systems
  - Content Management Systems
  - Catalogs
  - Personalization
  - Analysis of messages (tweets, etc.) in real time
  - Database for web servers that handle JSON natively
  - Etc.

Data Models for Documents

- Documents
  - Hierarchical structure, with nesting of elements
  - Weak structuring, with "similar" elements
  - Base types: text, but also integer, real, date, etc.
- Two main data models
  - XML (eXtensible Markup Language): W3C standard (1998) for exchanging data on the Web
    - Complex and heavy
  - JSON (JavaScript Object Notation) by Douglas Crockford (2005) for exchanging data JavaScript
    - Simple and light

JSON Example

```
{"bib": [
  {"book":
        {"title": "Principles of Distributed Database Systems",
         "authors": [{"name": "Patrick Valduriez"}, {"name": "Tamer Özsu"}],
         "publisher": "Springer",
         "year": "2011"},
  {"book":
        {"title": "XML: des bases de données aux services Web ",
         "authors": [{"name": "Georges Gardarin"}],
         "publisher": "Dunod",
         "year": "2002"}
]}
```

On peut définir ses propres balises (comme pour XML)

JSON Data Model

- Subset of ECMAScript object notation
  - ECMA standard for scripting languages de script such as JavaScript and ActionScript
- Independent of programming language
  - JavaScript, C, C++, C#, Java, Perl, Python, etc.
- Simple
  - Syntaxe familiar to programmers of languages like C, C++, Java, JavaScript
- Literals
  - String, number, true, false, null
- Two structures
  - Object, e.g. { "a":1,"b":2,"c":3 }
  - Array, e.g. [1, 2, 3, "html", ”xml", "json"]

JSON Query Languages

- Objective
  - Similar to that of XQuery, but for JSON
- No standard yet but some proposals
  - JSONpath (Stephan Goessner)
    - Equivalent of Xpath for JSON
  - JSONiq (jsoniq.org)
  - JAQL (Google)
  - MongoDB
  - N1QL (pronounce "niquel") of CouchBase

MongoDB

- Objective: performance and scalability
  - A document is a collection of (key, typed value) with a unique key (generated by MongoDB)
- Data model and query language based on JSON
  - Binary JSON (BSON): more compact
- No schema, no join, no complex transaction
- Scalability through explicit sharding (complex)
- SN cluster architecture
- Secondary indexes
- Integration with MapReduce & Spark

MongoDB – data model

- A MongoDB (posts) collection

_id: ObjectId("abc")	author: "alex", title: "No Free Lunch", text: "This is ....", tags : ["business", "ramblings"], comments: [{who: "jane", what: "I agree."}, {who: "joe", what: "No…"}]
_id: ObjectId("abd")	A post by X
_id: ObjectId("acd")	A post by Y

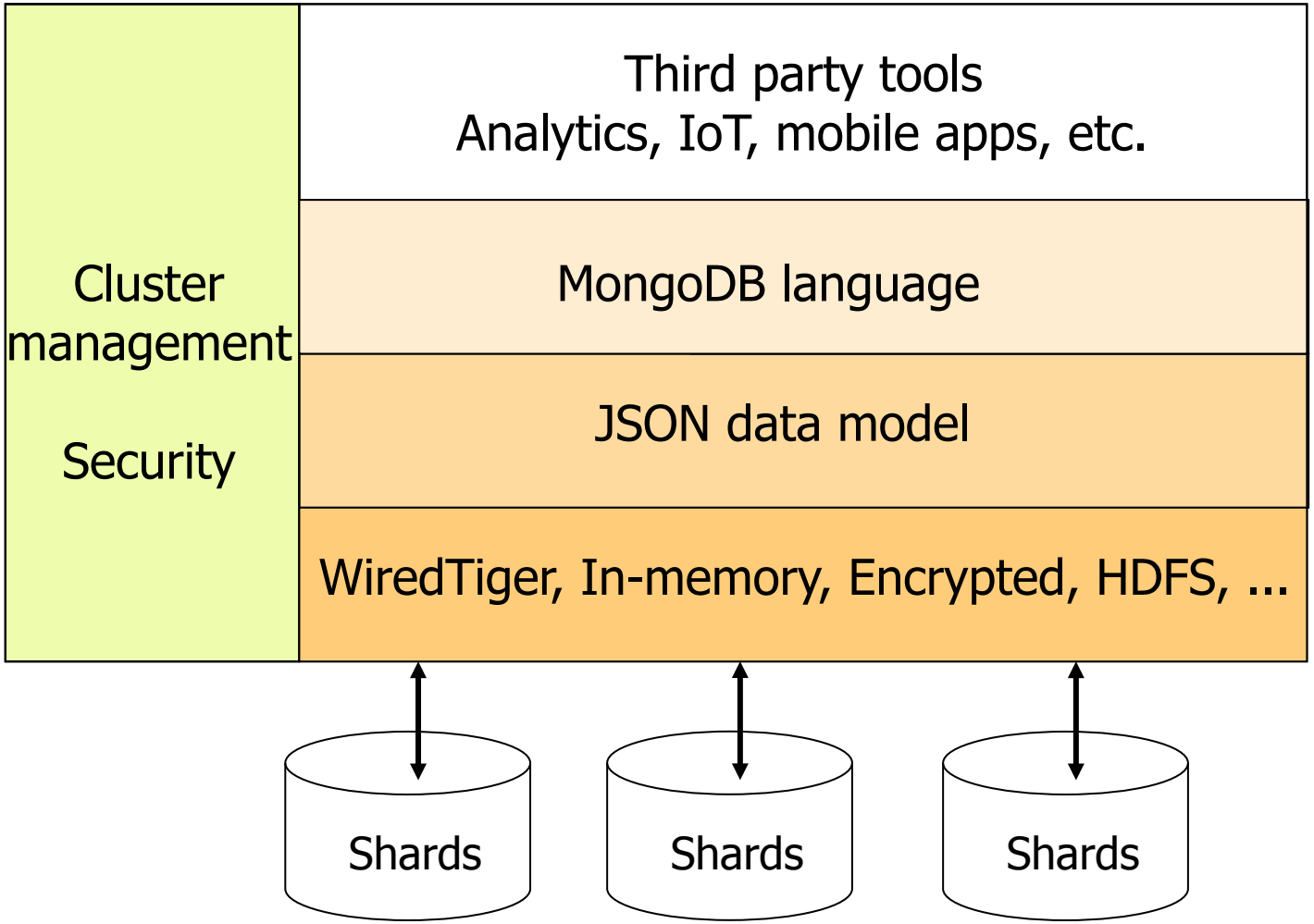
Unique key generated by MongoDB

Value = JSON object with nested arrays

MongoDB – query language

- Expression of the form
  - db.nomBD.function (JSON expression)
- Update examples
  - db.posts.insert({author:'alex', title:'No Free Lunch'})
  - db.posts.update({author:'alex', {$set:{age:30}})
  - db.posts.update({author:'alex', {$push:{tags:'music'}})
- Select examples
  - db.posts.find({author:"alex"})
    - All posts from Alex
  - db.posts.find({comments.who:"jane"})
    - All posts commented by Jane

MongoDB - architecture



Main NoSQL JSON DBMSs

Vendor	Product	Languages	Comments
Apache	CouchDB	JavaScript	Open source (Apache)
Couchbase Inc.	Couchbase	N1QL	Open source (Apache)
djondb.com	djondb	JSON	
ejdb.org	EJDB	Mongo-DB like	Open source (LGPL)
linkedin	Espresso	JSON	
MarkLogic	Marklogic server	JSON	Integration with Hadoop
Mongodb.com	MongoDB	Ext. JSON	
zorba.io	Zorba	JSONiq	Open source (Apache)

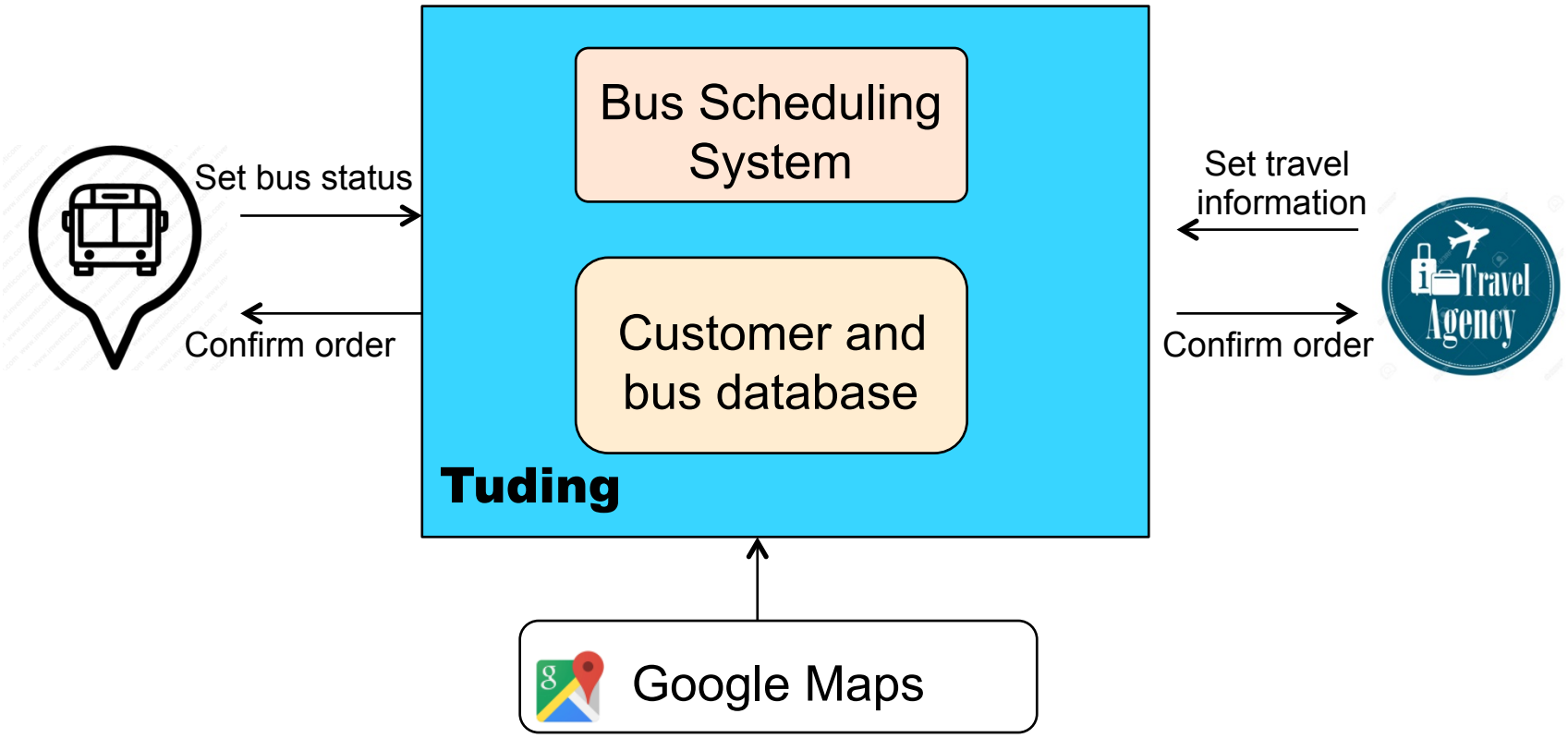
Case study: MetLife

- MetLife
  - Large health insurance company in USA
- Goal of MetLife Wall (client call center)
  - 360° view of policyholders for quick response to calls
- Problem
  - Integration of a wide variety of data from 70+ silos with:
    - 50 million insurance policies and 120 million clients
    - Transaction history (payments, compensation claims): 200 millions documents
  - RDBMS does not work
    - Problem: data schema evolution
- MongoDB solution
  - No data schema to define, easy to add new fields
  - Scalability in a SN cluster
  - MetLife Wall adopted by all call center operators

Case Study: TudingBus

- Tuding: Mobility as a Service
  - B2B platform (Uber-like) for bus sharing, group trips,...
- Data
  - Requests from customers, e. g. travel agencies, companies,...
  - Availability and location of buses
  - Google Maps data
- Problems
  - Storage of heterogeneous data
  - Dynamic bus scheduling

Architecture



MongoDB Solution

- Data
  - Travel agencies: text, unstructured
  - Bus rental companies: JSON
  - Google Maps: unstructured, very heterogenous among countries
- MongoDB choice
  - DBaaS with Microsoft Azure CosmosDB
  - Scalability
  - Storage of text and unstructured data
  - Indexes on different fields

3.3. Tabular Stores: Hbase

- Origin: Google CBigtable
  - Google Cloud, internal applications
- Objectives
  - Scalable database management in a SN cluster, with a simple API (no SQL)
  - Rely on HDFS to store data, with fault tolerance and availability
- Specific data model, combining row storage and column storage
  - Rows with multi-valued columns
  - Partial schema (columns)
- Dynamic partitioning of tables by interval in regions

HBase – data model

Row id Column Column family

Row key	Name	Email
100	Prefix : "Dr" Last : "Dobb"	email : gmail.com : " dobb@gmail.com"
101	First : "Alice" Last : "Martin"	email : gmail.com : " amartin@gmail.com" email : free.fr : " amartin@free.fr"

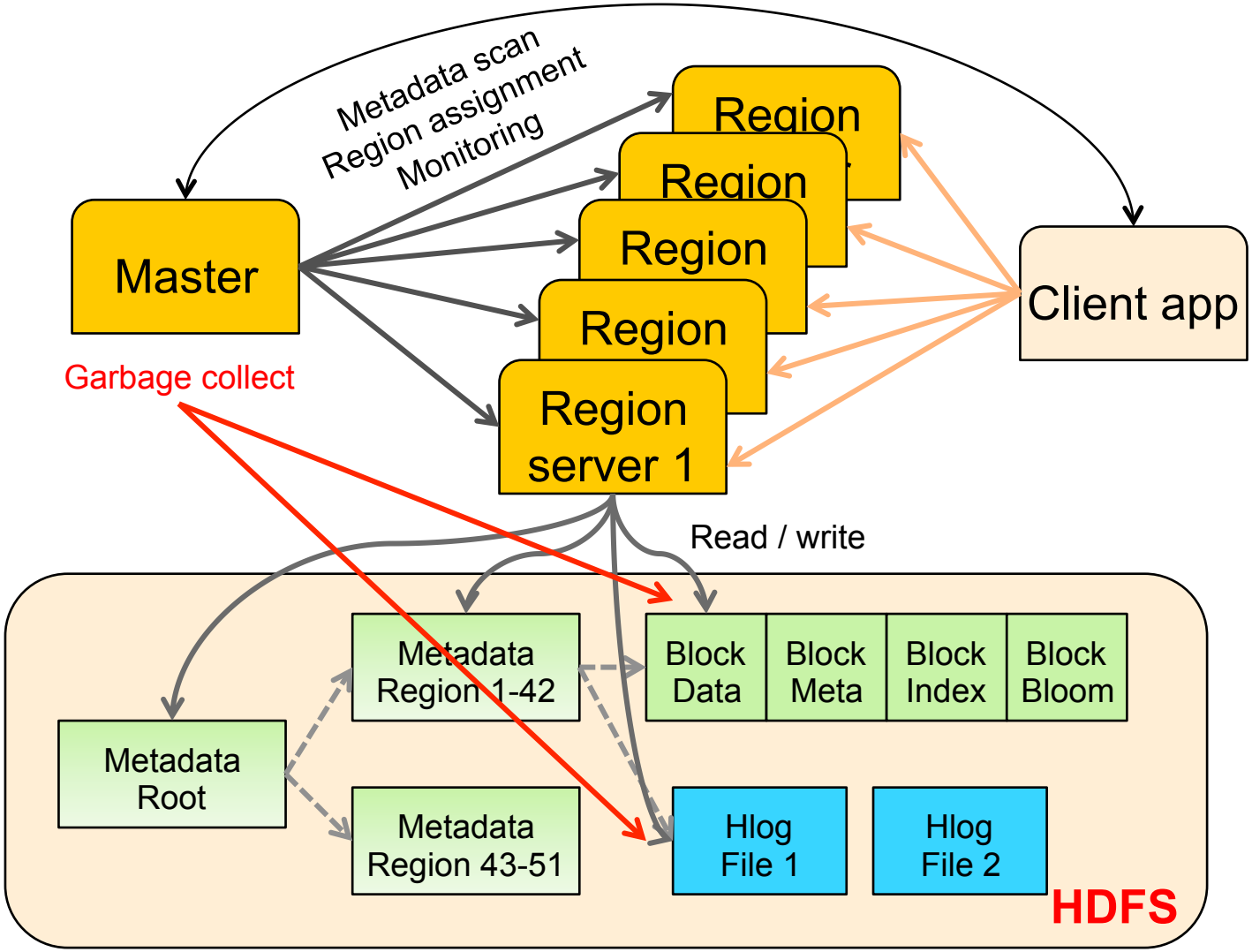
Region

- Column family = a kind of multi-valued attribute
  - Set of columns (of the same type), each identified by a key
    - Column key = attribute value, but used as a name e.g. gmail.com, free.fr
- Unit of access control and compression

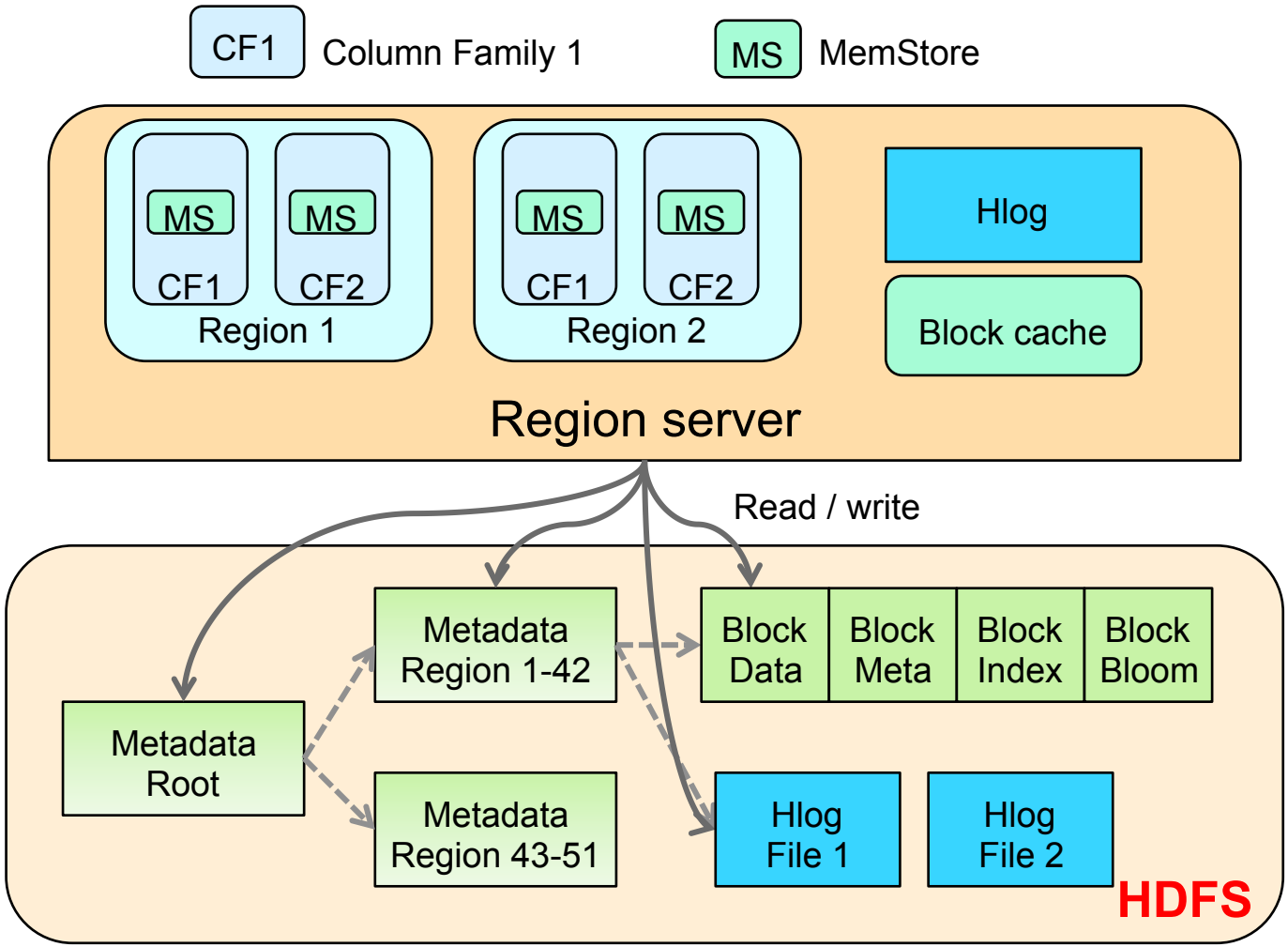
HBase - language

- No such thing as SQL
- Basic API for defining and manipulating tables, within a programming language such as Java
  - No impedance mismatch
  - Various operators to write and update values, and to iterate over subsets of data, produced by a scan operator
  - Simple selection operator on a table
  - Selection of rows and columns
  - No join or union
  - Atomicity for individual rows
    - In a region server

HBase - architecture



Region Server



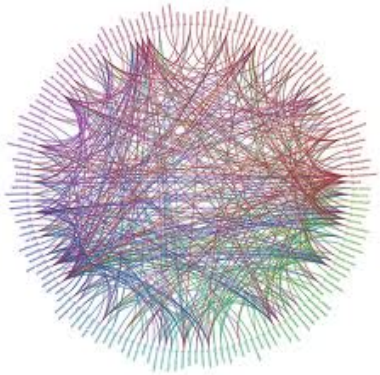
Case Study: Scaled Risk

- Scaled Risk: French startup in finance
  - Software solutions: risk management, market surveillance and fraud detection
- Problem
  - Continuous increase in data and need for real-time dashboards
  - E.g. stock market: 200,000 orders/s, 1 million during peak periods
  - Structured but very heterogeneous and evolving financial data
- HBase based solution
  - Flexible model for financial data
  - Performance on big data
  - Using timestamps to manage write data consistency

Case Study: NetFlix

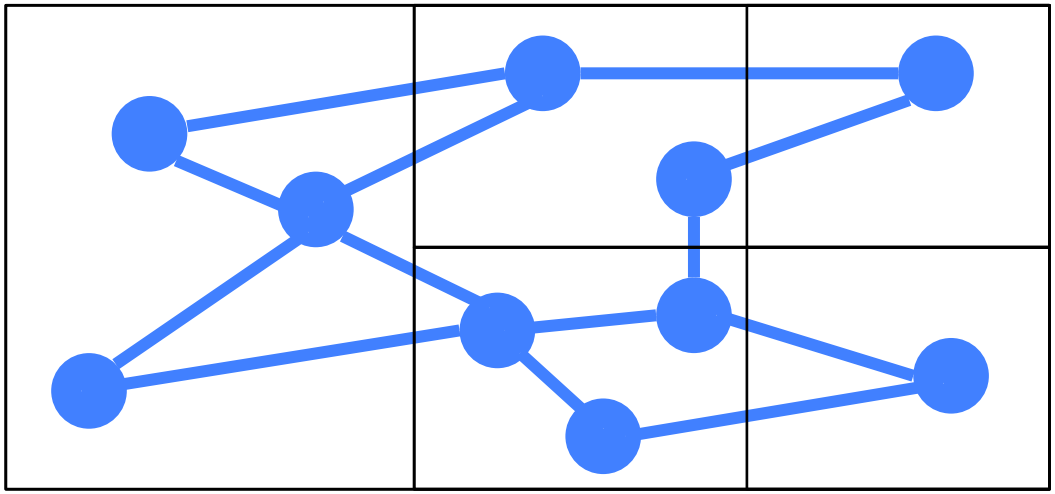
- The Netflix database in Amazon cloud
  - 500 terabytes
  - 1 billion requests/day
  - 3000 nodes
- Problem
  - Periodic maintainance by Amazon of its Xen hypervisers, which implies rebooting nodes (8%)
- Solution: Cassandra
  - NoSQL by Facebook inspired by BigTable and DynamoDB
  - Column families, consistent hashing, regions, ...
  - Cassandra Query Language
- Advantages
  - Automatic routing of requests to active nodes
  - No service interruption

3.4. Graph DBMS



- Database graphs
  - Very big: billions of nodes and links
  - Many: millions of graphs
- Main applications
  - Social networks
    - Recommendation, sharing, sentiment analysis
  - Master data management
    - Reference business objects, data governance
  - Fraud detection in real time
    - E-commerce, insurance, etc.
  - Enterprise networks
    - Impact analysis, QoS
  - Identity management
    - Group management, provenance

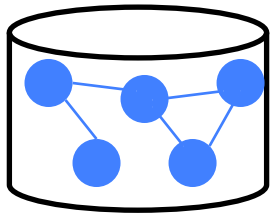
Graph Partitioning (Sharding)



- Objective: get balanced shards
  - NP-hard problem: no optimal algorithm
  - Solutions: approximate, heuristics, based on the graph topology

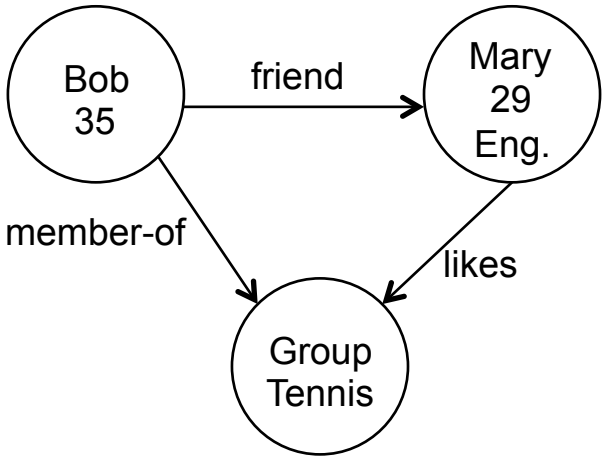
Neo4J

- Direct support of graphs
  - Data model, API, query language
  - Implemented by linked lists on disk
  - Optimized for graph processing
  - Transactions
- Implemented on SN cluster
  - Asymmetric replication
  - But no graph partitioning (sharding)
    - Planned for a future version



Neo4J – data model

- Nodes
- Links between nodes
- Properties on nodes and links



A Neo transaction

```
NeoService neo = … // factory
Transaction tx = neo.beginTx();
Node n1 = neo.CreateNode();
n1.setProperty("name", "Bob");
n1.setProperty("age", 35);
Node n2 = neo.createNode();
n2.setProperty("name", "Mary");
n1.setProperty("age", 29);
n1.setProperty("job", "engineer");
n1.createRelationshipTo(n2,
   RelTypes.friend);
tx.Commit();
```

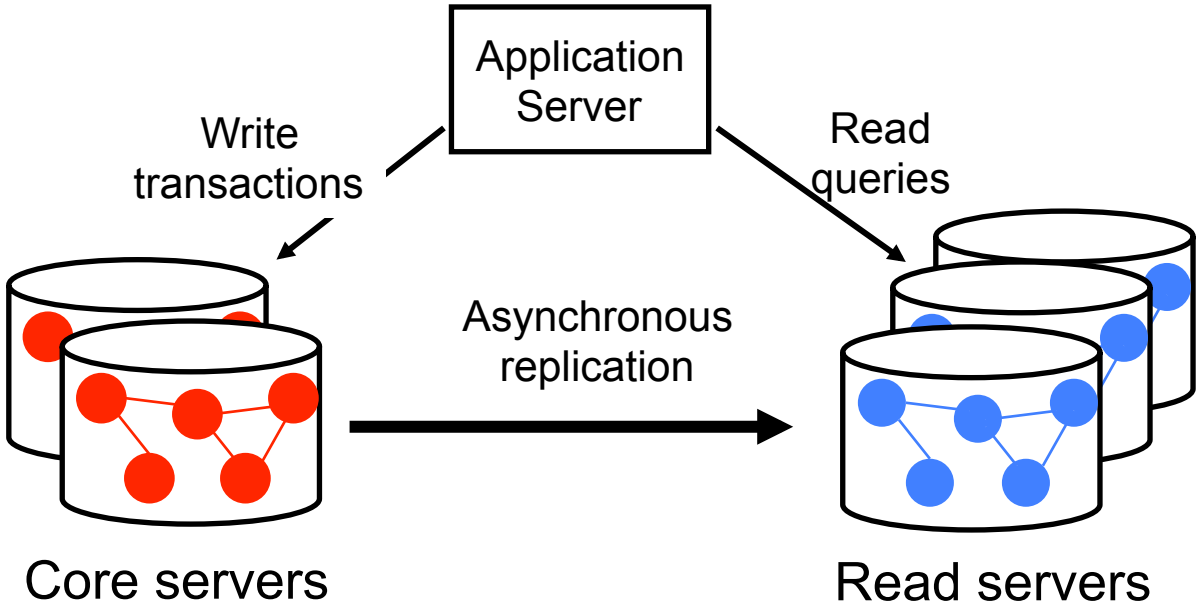
Neo4J - languages

- Java API (navigational)
- Cypher query language
  - Queries and updates with graph traversals
- Support of SparQL for RDF data

Ex. of Cypher query that returns the (indirect) friends of Bob whose name starts with "M"

```
START
   bob=node:node_auto_index
   (name = 'Bob')
MATCH bob-[:friend]->()-
   [:friend]->follower
WHERE follower.name =~ 'M.*’
RETURN bob, follower.name
```

Neo4J – architecture



Case Study : GameSys

- GameSys (UK)
  - One of the leaders in online gambling (4+ million players)
- Objective
  - Integration of the online gaming platform with Facebook and adaptation of a reference data management system to offer new social services
- Problem
  - Big interconnected data
- Solution with Neo4J
  - Naturally supports social network graph, with scalability and flexibility
  - Real-time analytical queries on data, allowing to discover new indicators and new measures without delay

3.5. Multimodel Stores: OrientDB

- Integration of key-value, document and graph
  - Extension of an objet model, with direct connections between objects/nodes
  - SQL extended with graph traversals
- Distributed architecture
  - Graph partitioning in a cluster
  - Symmetric replication between data centers
  - ACID transactions
  - Web technologies: JSON, REST

Other NoSQL Systems

Vendor	Product	Category	Comment
Apache	Cassandra Accumulo	KV Tabular	Open source, Orig. Facebook Open source, Orig. NSA
ArangoDB	ArangoDB	Multimodel	Open source
Couchbase	Couchbase	KV, document	Origin: MemBase
Google	Bigtable	Tabular	Proprietary, patents
FaceBook	RocksDB	KV	Open source
LinkedIn	Voldemort Expresso	KV Document	Open source ACID transactions
MapR	MapR DB	Multimodel	Proprietary
Oracle	NoSQL	KV	Based on BerkeleyDB
Neo4J.org	Neo4J	Graph	Open source, ACID transactions
Ubuntu	CouchDB	Document	Open source
Redis	Redis	KV	Open source, in-memory

Returns of Experience on NoSQL

- Many success stories on the software vendors's sites
- But also negative feedback
  - Example: "Why You Should Never Use MongoDB", Sarah Mei's blog
    - Application: Diaspora's P2P social network
    - Problem: with hierarchical documents, there is data redundancy for representing graph data
    - But this is a poor design
- Conclusion: make the difference between good and bad use of the technology !

4. NewSQL

- Pros NoSQL
  - Scalability
    - Often by relaxing strong consistency
  - Performance
  - Practical APIs for programming
- Pros Relational
  - Strong consistency
  - Transactions
  - Standard SQL
    - Makes it easy for tool vendors (BI, analytics, ...)
- NewSQL = NoSQL/relational hybrid

Google F1

- F1 inspired by the term *hybrid filial 1* in genetics
  - For the AdWords killer app
    - More than 100 Terabytes, 100K requests per sec
    - Scalability problem with the MySQL / cluster solution
- Objective
  - "F1 combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and usability of traditional SQL databases."
- Geographic distribution of the data centers
  - Synchronous replication between data centers for high availability
  - Partitioning and parallel processing within data center

Google F1

- Based on Spanner, a large scale distributed system
  Synchronous replication between data centers with Paxos
  - Load balancing between F1 servers
    - Favor the geographical zone of the client
- Different levels of consistency
  - ACID transactions
  - *Snapshot* (read only) transactions
    - Based on data versioning
  - Optimistic transactions (read without locking, then write)
    - Validation phase to detect conflicts and abort conflicting transactions
- Two interfaces
  - SQL
  - NoSQL key-value interface
- Hierarchical relational storage
  - Precomputed joins

Case Study: Google AdWords

- Application to produce sponsored links as results of search engine
  - Revenue: $50 billion/year
- Use of an auction system
  - Pure competition between suppliers to gain access to consumers, or consumer models (the probability of responding to the ad), and determine the right price offer (maximum cost-per-click (CPC) bid)
- The AdWords database with F1
  - 30 billion search queries per month
  - 1 billion historical search events
  - Hundreds of Terabytes

HTAP

- Hybrid Transaction and Analytics Processing
  - Gartner 2015
- OLAP+OLTP on the *same* data
  - Realtime analysis of operational data
  - No separation between operational DB and DW
    - No more ETLs
- Techniques
  - NoSQL: key-value storage, partitioning, fault-tolerance, replication, …
  - SQL: queries, in-memory
  - NewSQL: transactions

LEANXCALE

- SQL DBMS
  - Access from a JDBC driver
- Key-value store (KiWi)
  - Multistore access: HDFS, MongoDB, Hbase, …
- OLAP parallel processing (Query Engine)
- Ultra-scalable transactionprocessing (patents)
  - SQL isolation level: snapshot
    - Reads not blocked by writes, using multi-version concurrency control (MVCC)
  - Timestamp-based ordering and conflict detection jsut before commit
  - Parallel commits of transactions
    - No bottleneck

Case Study: IKEA

- Objective: proximity marketing
  - Real-time analysis of customer behaviour in stores (in a given region) in order to provide targeted offers
- Problem
  - OLTP and OLAP on a very large scale in real time
- Solution with LeanXcale
  - Ingestion of real-time information on customer itineraries in store (transactions)
    - Use of beacons: sensors to identify and locate customers from their smartphone
  - Analysis and segmentation of customers by similar behaviour in other stores

Other NewSQL Systems

Vendor	Product	Objective	Comment
CockroachDB Labs, NY	CockroachDB	OLTP	Open source (ex-googlers) inspired by Spanner, based on RocksDB KV store
SAP	HANA	HTAP	The HTAP pioneer. In-memory, column store
MemSQL Inc.	MemSQL	HTAP	In-memory, column and row store, MySQL compatible
Esgyn	Esgyn	HTAP	Apache Trafodion for OLTP, Hadoop for OLAP
NuoDB	NuoDB	HTAP	Cloud solution (Amazon)
Splice Machine	Splice Machine	HTAP	HBase as storage engine, Derby as OLTP query engine and SparkQL as OLAP query engine. Custom centralized transactional manager.
VoltDB Inc.	VoltDB	HTAP	Open source and proprietary In-memory

5. Which Data Store for What?

Category	Systems	Requirements
Key-value	DynamoDB, SimpleDB, Cassandra	Access by key Flexibility (no schema) Very high scalability and performance
Document	MongoDB, CouchDB, Expresso	Web content management Flexibility (no schema) Limited transactions
Tabular	BigTable, Hbase, Accumulo	Very big collections Scalability and high availability
Graph	Neo4J, Titan, AllegroDB	Efficient storage and management of large graphs
Multimodel	OrientDB, ArangoDB	Integrated key-value, document and graph management
NewSQL	Google F1, VoltDB LeanXcale, Splice Machine	ACID transactions , flexibility and scalability SQL and key-value access