Relational VS NoSQL

Recap

- Relational DBMS software has worked very well for many decades
- Companies have invested lots of money in software built upon relational DBMS infrastructure
- Companies have invested in staff/talent skilled in RDBMS technologies

BUT

- RDBMS systems struggle to adequately scale to support Big Data's volume, variety, and velocity demands
- Big Data is exploding faster than RDBMS technologies can handle

NOSQL Data Models

- **NoSQL** (“Not Only SQL”)
  - Uses clusters:
    - Distribute the Data
    - Distribute the Processing
  - Uses Replication to provide
    - Redundancy
    - High availability
    - Parallel Processing
  - Horizontally scalable

NOSQL Data Models

- **NoSQL** (“Not Only SQL”)
  - Less Structure Required
    - Does not store data in tables with rigid row/column structure
    - Uses Aggregate model
  - Restricts join capabilities
  - Relaxes ACID transaction compliance
  - May use a *non-SQL* query language
  - Typically open source, very low-cost software acquisition

NOSQL Data Models

Relational

- Schema defines rigid structure
  - Tables, Rows, Columns
- Foreign Key relationships
  - Which support joins
- Uses SQL
- Maintains ACID compliance
- Normalized: store a value only once
- Clustering available, but challenging
- Leading DBMS solutions are quite expensive

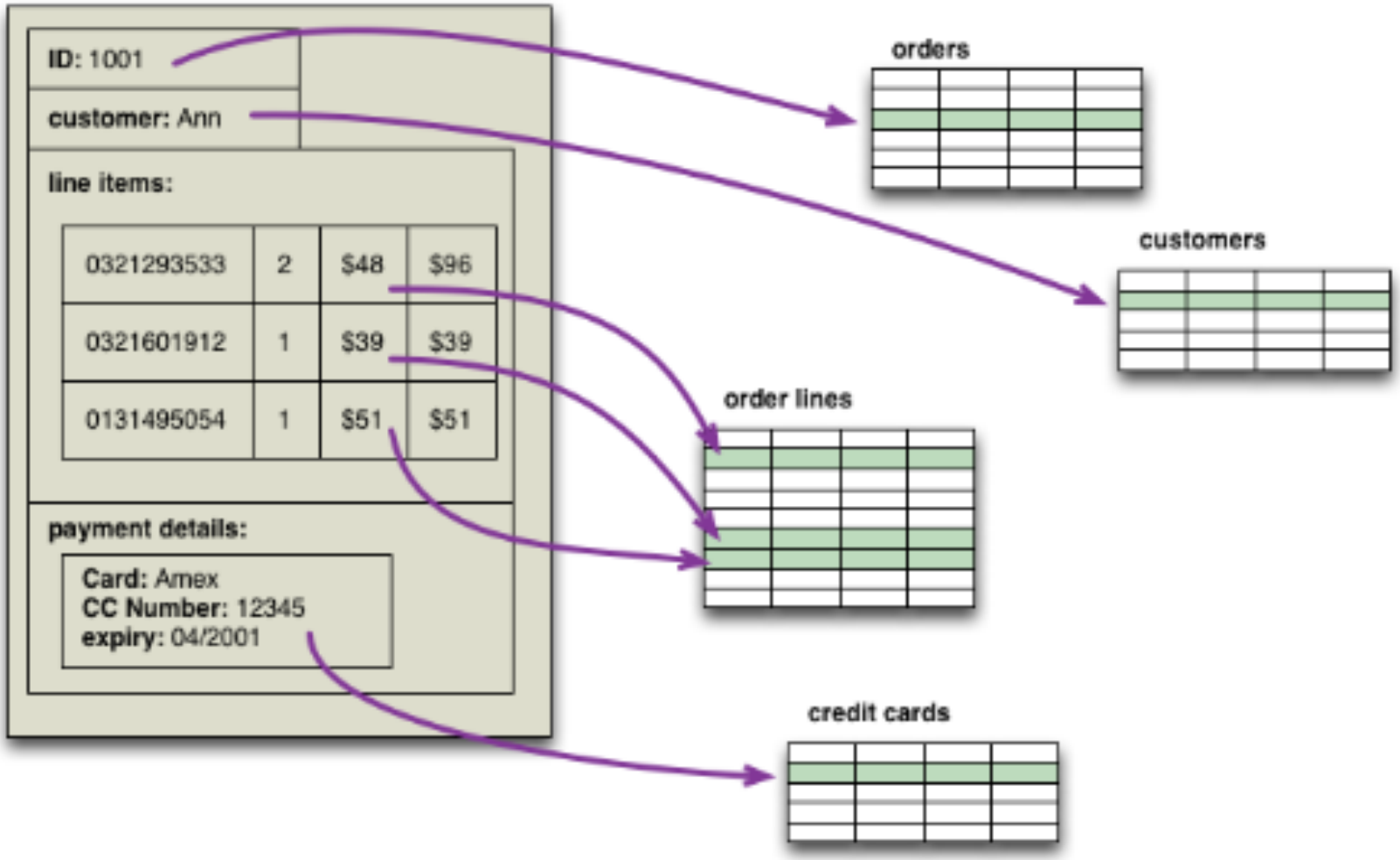
NoSQL

- Stores related values in aggregates
- Flexible structure:
  - Ranges from none to some
- No joins
- Relaxed ACID compliance
- De-normalized
- Uses alternate query language
- Designed to support clustering
- Almost all players are open source and low-cost

NOSQL Data Models

- **NoSQL "aggregate" Data Model**
  - RDBMS requires Tables, Rows, Columns as data stores
    - Columns have "domains"
    - Third normal form – no multi-values, "store it once"
  - NoSQL sytems use AGGREGATES as data stores
    - Data values are grouped together as users need them
    - Think of an unnormalized document, or an "object"
    - Contains related values that are retrieved and manipulated together

NOSQL Data Models



NOSQL Data Models

- **Aggregates are conceptually the opposite of 3rd Normal Form**

- **Why aggregates?**
  - It is difficult to spread a relational model across nodes in a cluster
    - Replication and sharding introduce big challenges in consistency
  - Each query should minimize the number of nodes being accessed across the cluster
  - Data values that are accessed together should live on the same node

NOSQL Data Models

- **NoSQL Data Schema/Models**
  - Document Store (using XML or JSON format)
  - Graph (using node/edge structures with properties)
  - Key-Value pairs
  - Wide Columnar store (rows with dynamic columns holding key-value pairs)

NOSQL Data Models

- Document
  - MongoDB
- Graph
  - Neo4j
- Key-Value
  - Amazon Dynamo
- Wide Column
  - Google BigTable, Apache Cassandra

NOSQL Data Models

- **Document Database**
  - Organized around a “document” containing text
  - Can handle very large data volumes
  - Provides Speed and Scalability
  - Document format is easily understood by humans
  - No “schema”, but JSON/XML provides internal structure within a document
  - Documents are indexed and stored within “collections”
  - Supports full text search
- **Popular Implementations** (open source)
  - MongoDB
  - CouchDB

JSON

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

XML

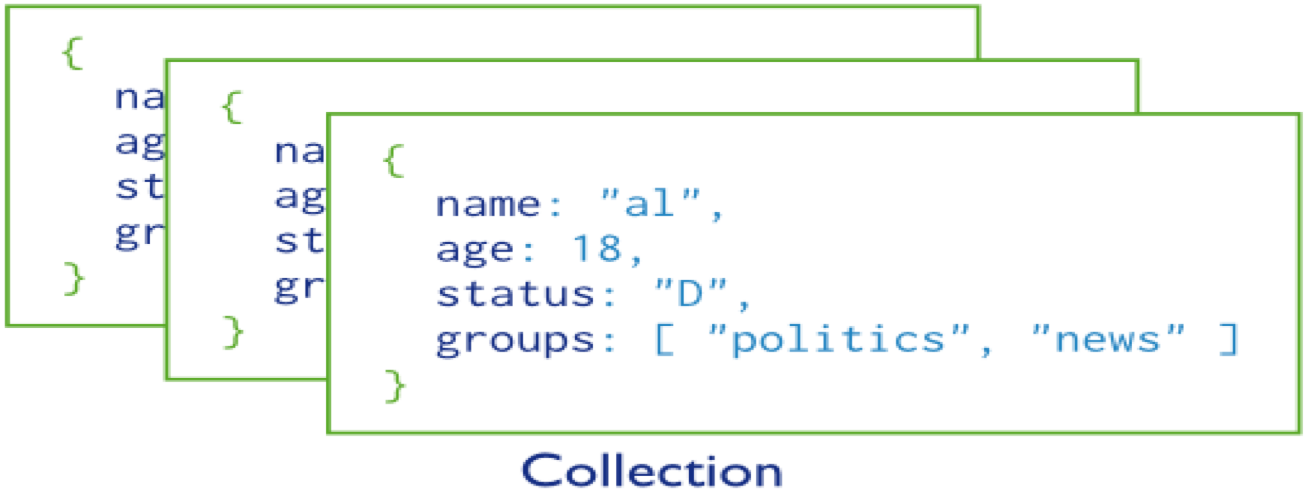
```
<contact>
        <firstname>Bob</firstname>
        <lastname>Smith</lastname>
        <phone type="Cell">(123)555-0178</phone>
        <phone type="Work">(890)555-0133</phone>
        <address>
                <type>Home</type>
                <street>123 Black St.</street>
                <city>Big Rock</city>
                <state>AR</state>
                <zip>23225</zip>
                <country>USA</country>
        </address>
</contact>
```

NOSQL Data Models

- **Document Database**
  - Keeps related information together (not normalized into tables)
  - Access to a document is fast (index/key/URL)
  - ACID compliance is maintained only within a document
  - Cannot easily “join” across documents
  - Documents are kept in “collections”

NOSQL Data Models

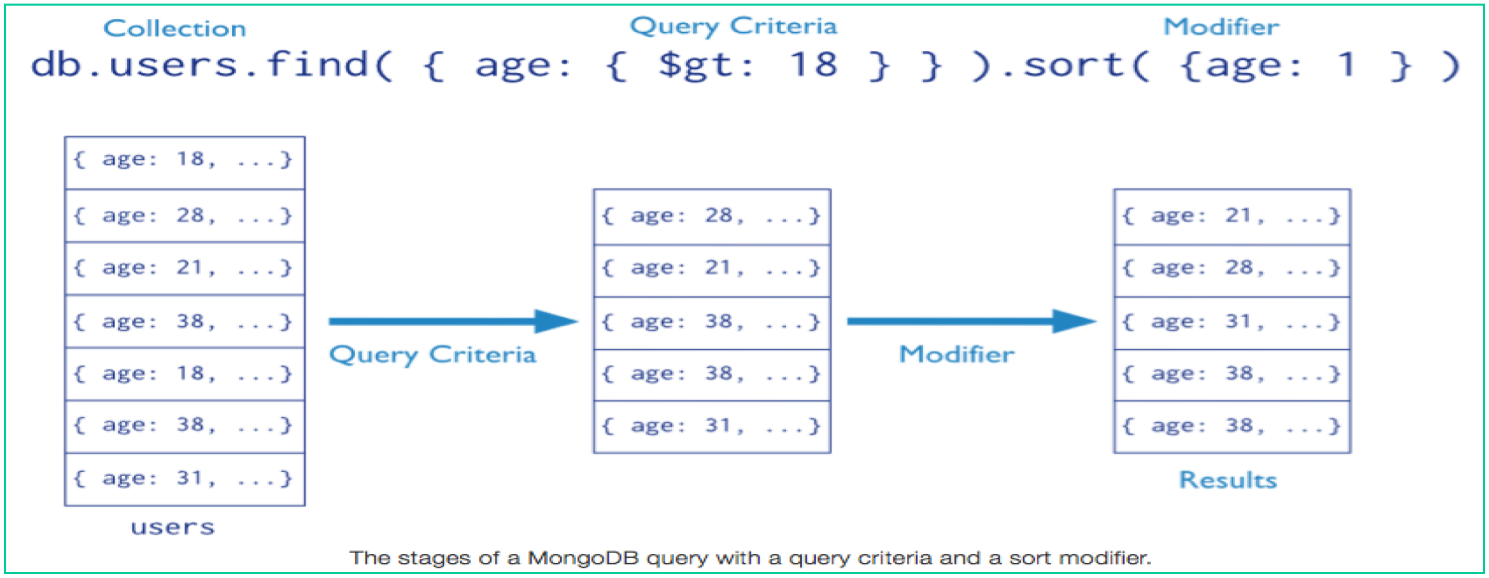
A MongoDB document.



A collection of MongoDB documents.

In MongoDB a query targets a specific collection of documents. Queries specify criteria, or conditions, that identify the documents that MongoDB returns to the clients.

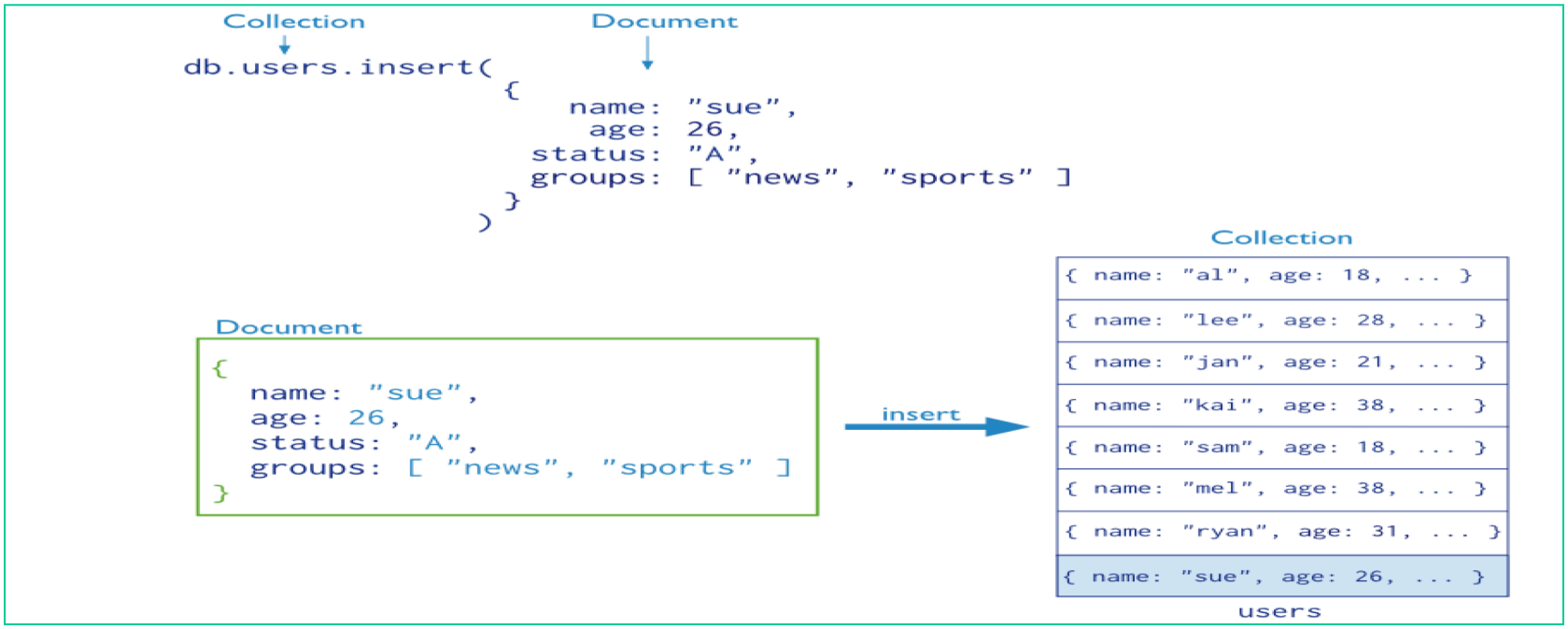
A query may include a *projection* that specifies the fields from the matching documents to return. You can optionally modify queries to impose limits, skips, and sort orders.



The stages of a MongoDB query with a query criteria and a sort modifier.

NOSQL Data Models

Data modification refers to operations that create, update, or delete data. In MongoDB, these operations modify the data of a single collection. For the update and delete operations, you can specify the criteria to select the documents to update or remove.

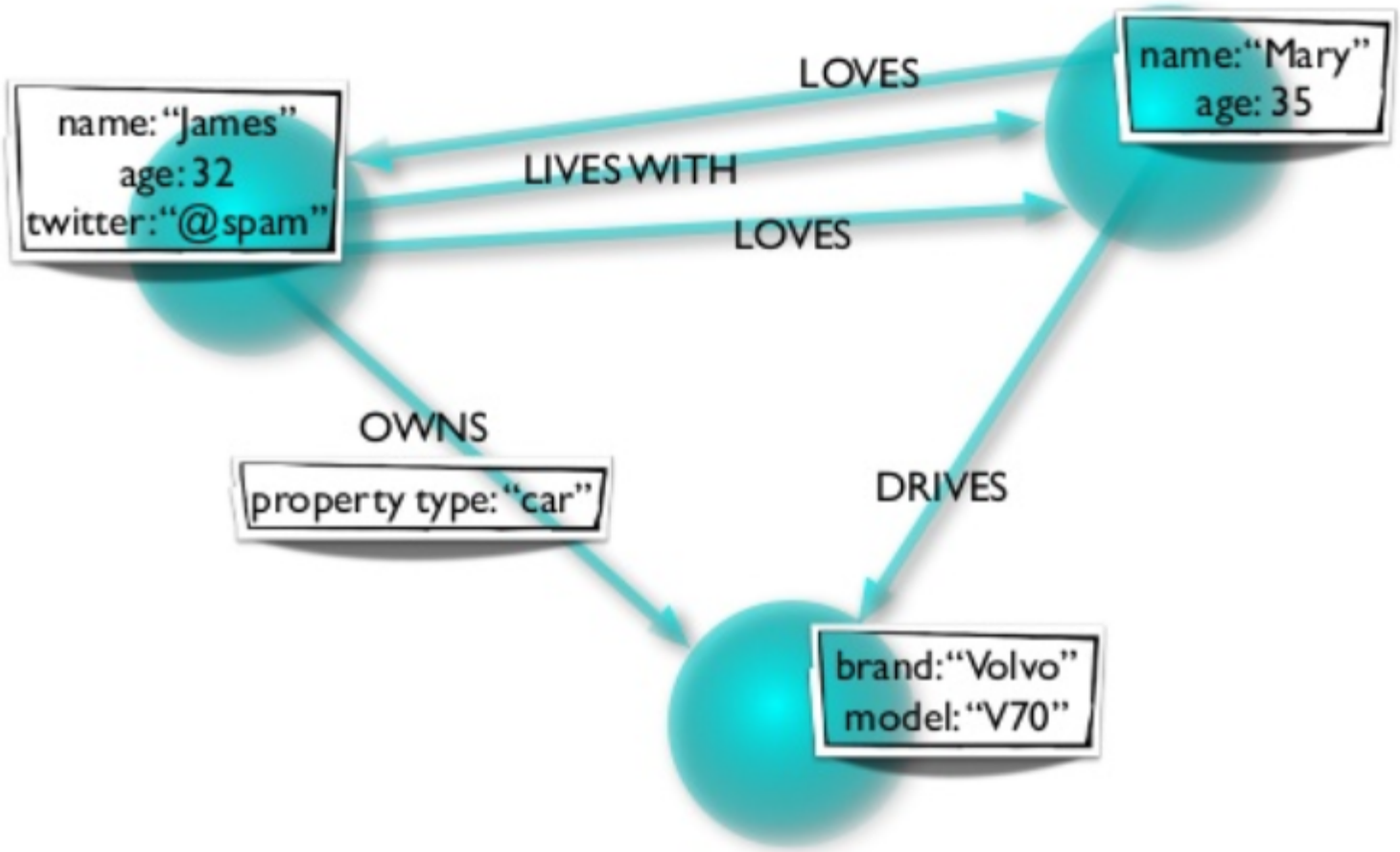


- **More on MongoDB next week**

NOSQL Data Models

- **Graph Database**
  - Uses a graph structure consisting of
    - Nodes – Represents an entity (like a person)
    - Edges – Represents a relationship between entities
    - Properties – Attributes associated with Nodes and Edges
  - Supports navigation along edges from a starting point node
  - Designed for applications tracking the inter-connections among entities
    - Who is friends with whom? (In a social network application)
    - Who is following me, who am I following?
  - Uses a pattern matching query language to navigate nodes & edges
- **Popular Implementations** (open source)
  - Neo4j

NOSQL Data Models



- **Stopped here on Friday April 3**

NOSQL Data Models

- **Key-Value Pairs Database**
  - "Schemaless"
  - Maps a key to an opaque value
    - That is, the database doesn't understand anything within the value
  - Simple operations (put, get, remove, modify)
  - Keys are unique in a collection
  - Most useful when access is by the primary key
  - May be a building block for other data models (such as key-value pairs within a wide-columnar store like Cassandra)
- **Popular Implementations**
  - Amazon Dynamo (available via AWS in the cloud)
  - Riak, Redis (open source)

Product Catalog Example

```
{
    Id: 206,
    Title: "20-Bicycle 206",
    Description: "206 description",
    BicycleType: "Hybrid",
    Brand: "Brand-Company C",
    Price: 500,
    Color: ["Red", "Black"],
    ProductCategory: "Bike",
    InStock: true,
    QuantityOnHand: null,
    RelatedItems: [
        341,
        472,
        649
    ],
    Pictures: {
        FrontView: "http://example.com/products/206_front.jpg",
        RearView: "http://example.com/products/206_rear.jpg",
        SideView: "http://example.com/products/206_left_side.jpg"
    },
    ProductReviews: {
        FiveStar: [
                "Excellent! Can't recommend it highly enough!  Buy it!",
                "Do yourself a favor and buy this."
        ],
        OneStar: [
                "Terrible product!  Do not buy this."
        ]
    }
}
```

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.CaseStudy.html

Key-Value Pairs Database Example (Amazon DynamoDB)

- The key value (Id) is 206.
- Most of the attributes have simple data types, such as String, Number, Boolean and Null.
- One attribute (Color) is a String Set.
- The following attributes are document data types:
  - A List of Related Items. Each element is an Id for a related product.
  - A Map of Pictures. Each element is a short description of a picture, along with a URL for the corresponding image file.
  - A Map of ProductReviews. Each element represents a rating and a list of reviews corresponding to that rating. Initially, this map will be populated with five-star and one-star reviews.

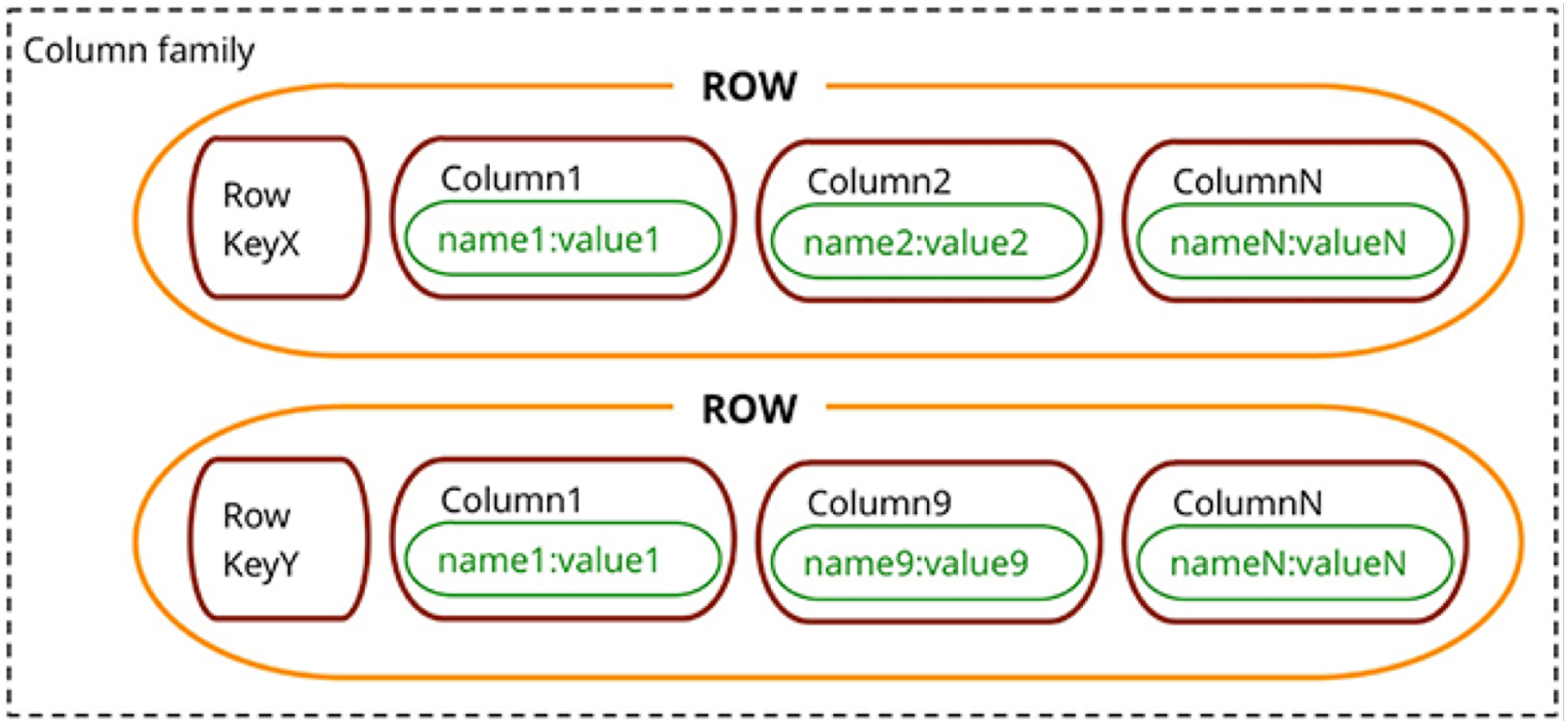
NOSQL Data Models

- **Wide-Column (Column Family) Store Database**
  - A TWO-LEVEL aggregate
  - Data is stored within “collections” of dynamic related columns
  - Similar to key/value with the pairs having columnar structure
- **Based on Google’s “Big Table”**
- **Popular Implementations** (open source)
  - Cassandra
  - HBase

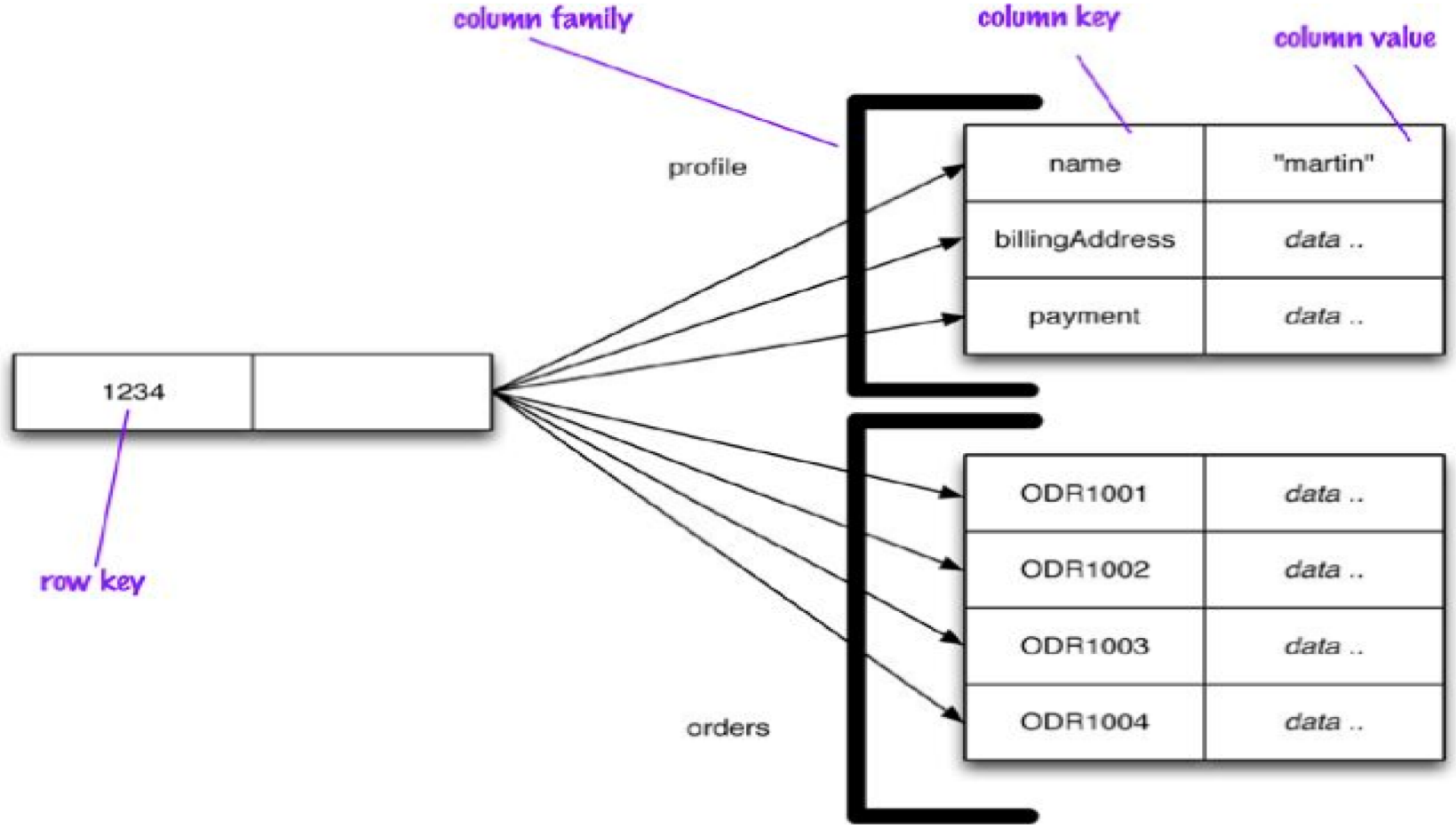
NOSQL Data Models

- **The first key is the row-key**
  - The entire row is an aggregate of related data
  - The row consists of many key-value pairs (“columns”)
- **The second key is the column family**
  - Each column family consists of sparse key-value pairs
    - “Sparse” means the column value isn’t stored if it isn’t needed

NOSQL Data Models



NOSQL Data Models



- **Wide-Column Store Example**
- Consider an application that needs to store stock trades and retrieve them by stock symbol
- Some data that might be stored includes the following:
  - symb: The stock symbol
  - id: A unique trade id
  - date: Date of the trade
  - price: Trade price
  - quant: Number of shares traded
  - notes: General comment field.

NOSQL Data Models

- Below, we illustrate trade data in a wide column store
  - The stock symbol (symb) is the row key
  - Generally the complete row is stored on a single node
  - The trades are stored as columns in the row
  - If there were many trades, the row could get very wide
  - Note also, that Trade 2 has no notes column
  - Contrast this to a normalized relational model with a trade table which would generally store trades in separate rows

symb	id	date	price	quant	notes	id	…	quant	id	…	notes
GM	1c3f	2014-09-15	32.96	100	blah	2a5f		400	2e99		bar

Benefits of a Wide Column Store

- Lookup of rows by row key can be **very fast**
- In a distributed cluster system, the complete row is stored on one node
- May be stored redundantly across the cluster
- Can be retrieved with one access
- Good fit for **scale-out** systems
- Can scale to large capacity and high availability
- The columns are "**sparse**"
  - That is, columns with no values are absent, saving space
  - As shown in Trade 2 having no Notes column
- Flexible access to column data in a row
  - You can flexibly query on them
  - e.g. - get all trades, get the last 10 trades, or 1st ten trades, etc.

Issues with a Wide Column Store

- The data model is **complex**, and **not very intuitive**
- Generally, you create your data model **based on your queries**
  - Possibly taking into account how data is distributed
- For example, querying for trades by stock symbol is very fast for the trade table shown earlier -- it's just one query to one node
- Consider finding all trades (for all stocks) on a particular date
  - The previous data model is not very good for this
  - You'd need to query every node, get the trades from that node, then combine them all for the result
- In practice, that may not scale
- You could, however, store data optimized for retrieval by date
- This would require another table / column layout

Issues with a Wide Column Store

- Client code **may be extensive**
  - For example, to keep track of trades by both stock symbol and trade date, you may need to maintain two sets of data
  - Which needs to be done via application code on a client
- Different from relational model where you maintain one set of data, and just write queries as needed
- However, wide column stores tend to have much more scale out capability -- Which is why we use them in the first place

Linear Scalability as in Cassandra

- Capacity may be easily added simply by adding new nodes
- For example,
  - Suppose 2 nodes can handle 100,000 transactions per second,
  - 4 nodes will support 200,000 transactions/sec, and
  - 8 nodes will tackle 400,000 transactions/sec: