Introduction to NoSQL

Instructor: Ekpe Okorafor

1. Big Data Academy - Accenture
2. Computer Science - African University of Science & Technology

Agenda

- Introduction
- Technical Overview
- Use Cases
- Under The Hood: Compare & Contrast

Agenda

- Introduction
- Technical Overview
- Use Cases
- Under The Hood: Compare & Contrast

What Is NoSQL?

NoSQL is a bit like Cloud Computing - An umbrella term

NoSQL:

- Data stores that avoid the **RELATIONAL** model
- Use other data models

NoSQL == Not Relational

Typical NoSQL characteristics …..

- No schema
- No joins
- Usually distributed
- Usually replicated
- Usually not ACID
- No SQL

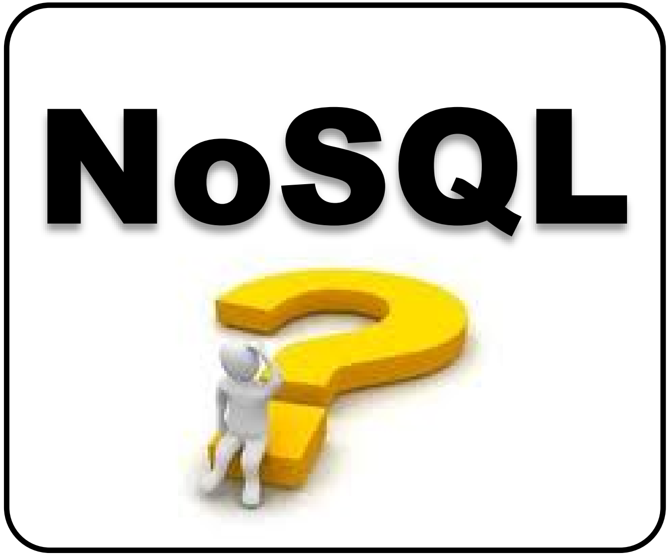
Relational databases have been a successful technology for twenty years, providing persistence, concurrency control, and an integration mechanism

Why NoSQL?

Definitely consider NoSQL if you have …..

- Need to scale horizontally without having to invest in EXPENSIVE large servers and storage area networks (SAN)
- Requirement to control 99 %ile latency
- Requirement for rapid development
- in a coder friendly environment

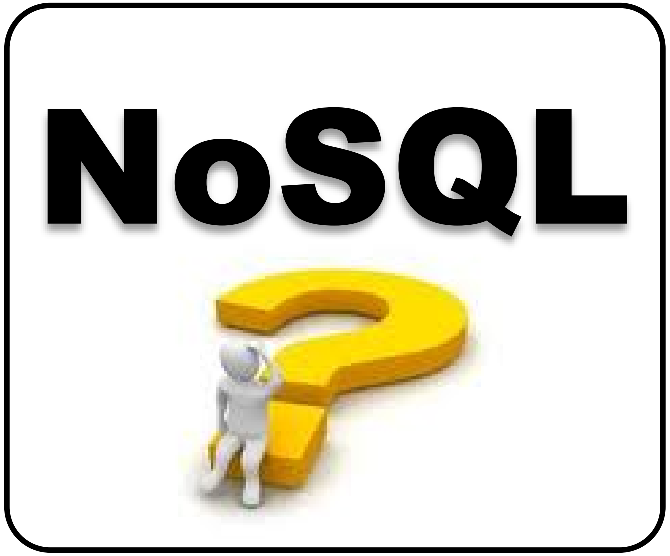
NoSQL seems to be a better match for some companies than to others. For many industry needs, traditional RDBMS will work adequately.



…Other Reasons

Problems that don’t require RDBMS

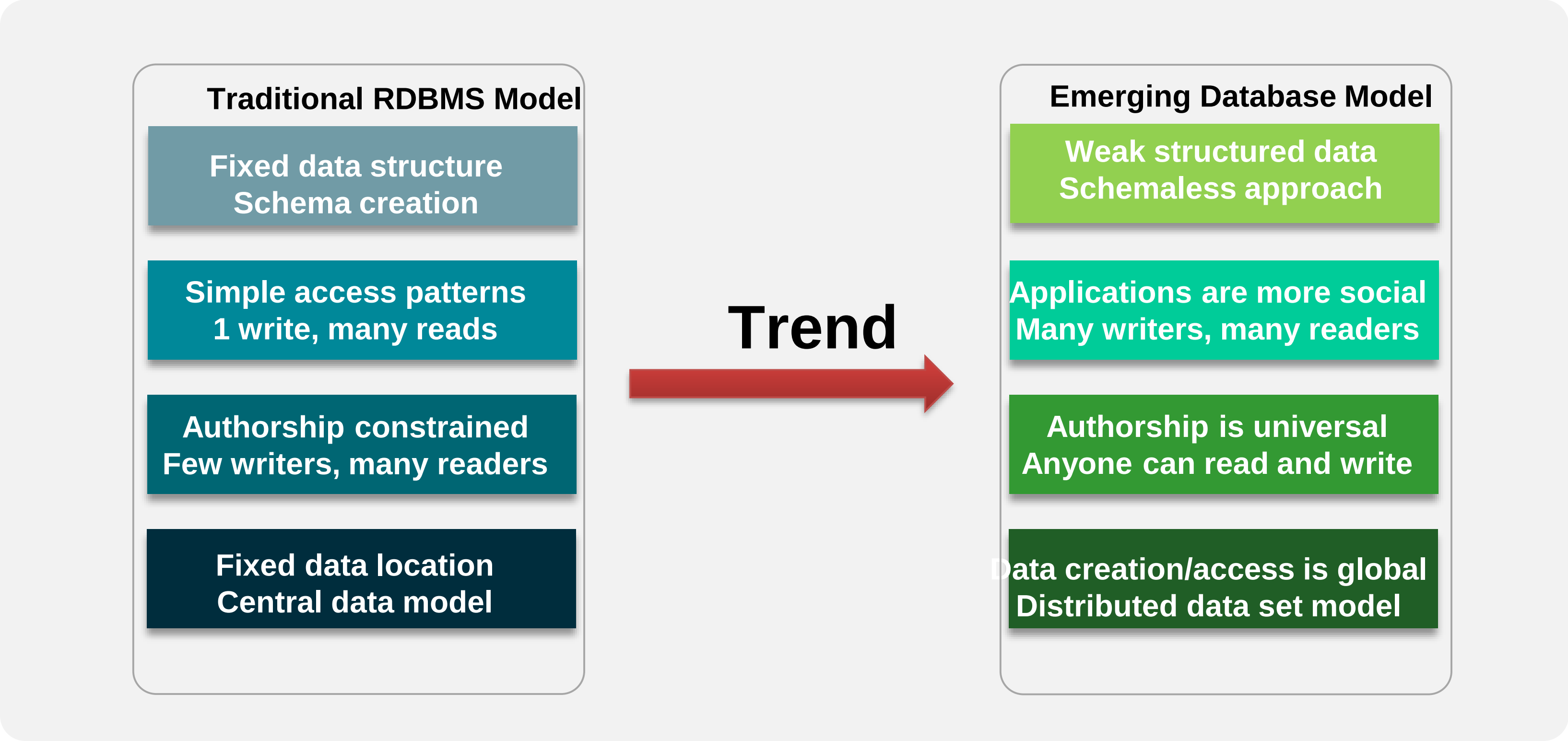
- Data access by primary key only
- Data join not needed
- Write-intensive and continuously
- Data model is a single set of items



These problems don’t necessarily require a relational database and other data models and solutions can be considered.

Look At The Trends

The enterprise data landscape is changing



Traditional RDBMS Model

Fixed data structure
Schema creation

Simple access patterns
1 write, many reads

Authorship constrained
Few writers, many readers

Fixed data location
Central data model

Trend

Emerging Database Model

Weak structured data
Schemaless approach

Applications are more social
Many writers, many readers

Authorship is universal
Anyone can read and write

Data creation/access is global
Distributed data set model

Traditional "relational" databases are not designed to manage emerging data types

What It All Means

Enterprises have a cost effective option to …….

- Undertake data problems previously thought to be too difficult or impossible to solve using traditional legacy relational databases
- Tap into huge unstructured data sources from emerging platforms for data analysis and business intelligence
- Derive connected intelligence using graph database methods as data becomes increasingly more complex and highly connected

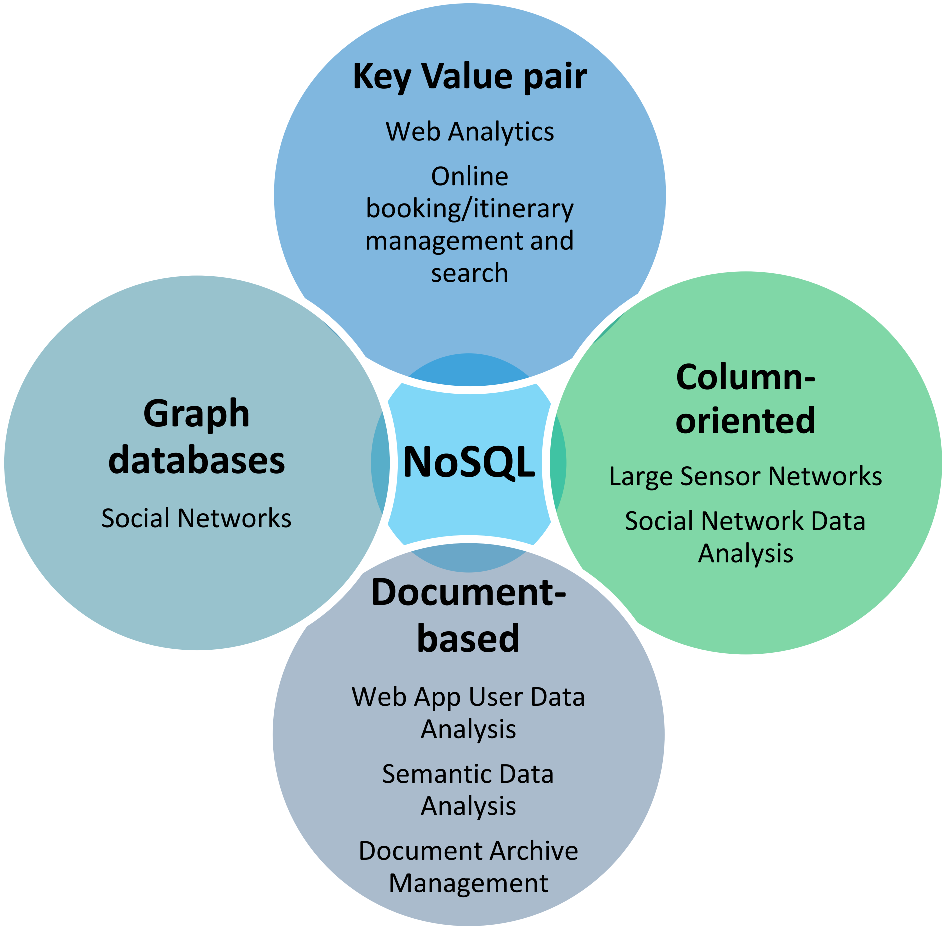
```
                    VECTOR/SIGNATURE

                    ACCOUNT MAINTENANCE

                    BANK NUMBER :
                    APPLICATION :
                    ACCT NUMBER :

                    FUNCTION    :

  1 - ADD NEW ACCOUNT                  5 - DELETE / REACTIVATE ACCOUNT
  2 - ADD SIGNATURE/INFORMATION        6 - SIGNATURE SELECTION
  3 - CHANGE ACCOUNT INFORMATION       7 - COPY ACCOUNT
  4 - CHANGE SIGNATURE/INFORMATION

  KEY BANK, APPLICATION, ACCOUNT, AND FUNCTION; THEN <ENTER>.
  <PF1> SEARCH NAME FUNCTION
  <CLR> TERMINATE MAINTENANCE FUNCTION.
```

Legacy!!!

Emerging

What Should Be Done

- NoSQL business enterprise data model analysis



- Key-Value pair databases are frequently found in caching and fast-lookup apps
- Column-oriented databases power sensor networks, such as with SETI and NASA
- Document-based databases are often used in place of Key-Value Pair databases when richer querying is required
- Graph databases can match social graphs, and simplify relationship navigation

Making The Right choice

Consider the key MOTIVATION & business need

- Just as transactional & analytical processing needs lead to technologies optimized for OLTP and OLAP
- Align the critical motivation and business needs to desired NoSQL solution

Convenience

- Simple to set up , ease of use and schema-less data
- Knowledge about the individual
- **key-value** and **document stores**) help solve problems related to atomic intelligence

Connectedness

- Complex and connected data.
- Knowledge about the networks and relationships
- **Graph databases** can markedly improve one's ability to leverage connected intelligence

Big Data

- Large volume of data
- Storage and processing requirements
- **Column oriented** and **key-value stores** are well suited to big data environments providing big data intelligence

Agenda

- Introduction
- Technical Overview
- Use Cases
- Under The Hood: Compare & Contrast

NoSQL Systems

Are alternative to traditional RDBMS, providing …

- Flexible schema
- Quicker/cheaper to set up
- Massive scalability
- Relaxed consistency → higher performance & availability

✓No declarative query language → more programming

✓Relaxed consistency → fewer guarantees

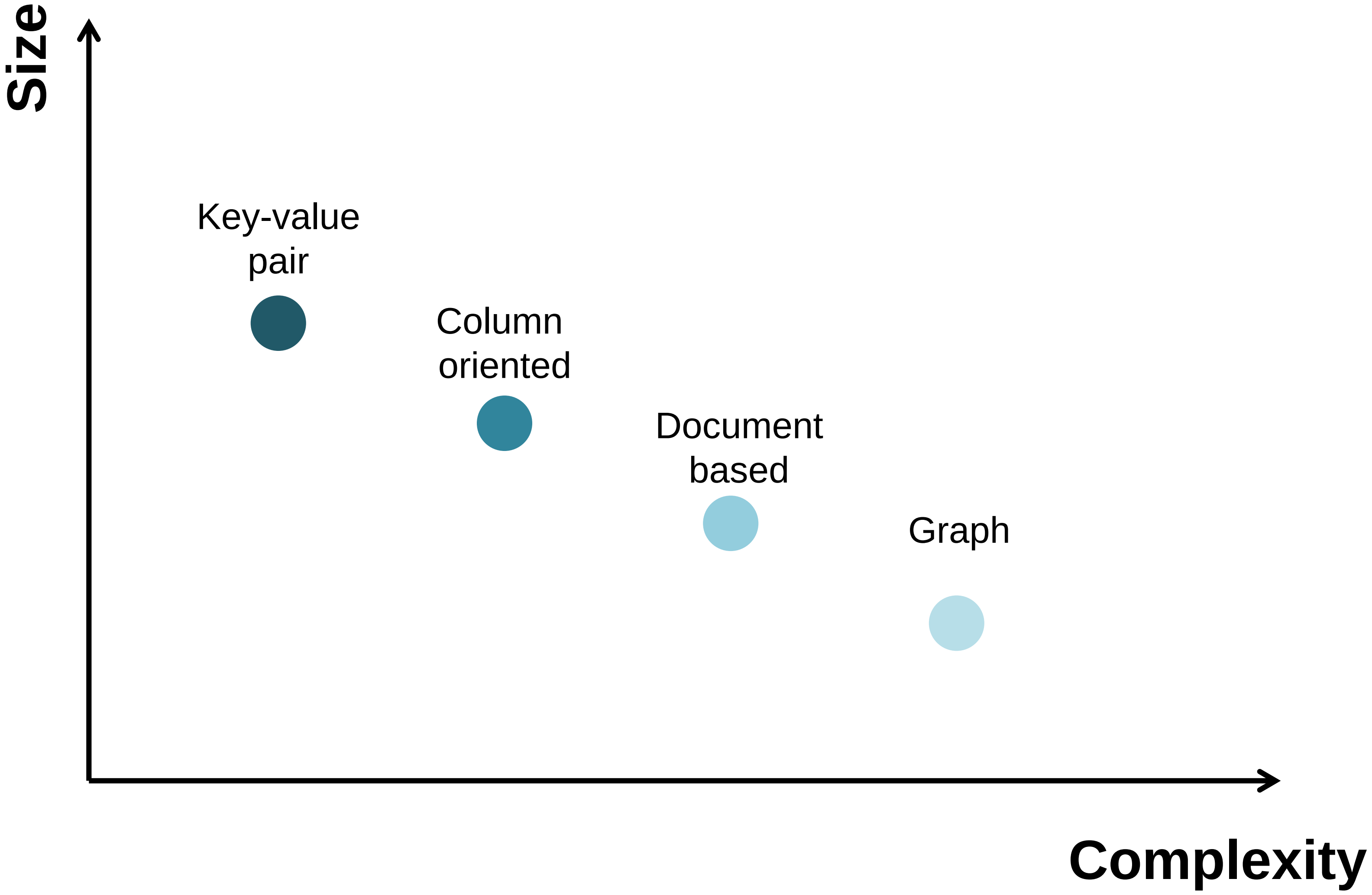
NoSQL Systems

Data Models

- “NoSQL” = “Not Only SQL’

  Not every data management/analysis problem is best solved exclusively using traditional RDBMS
- Current NoSQL based on data model types include:
  - Key-value pair
  - Document-based
  - Column oriented
  - Graph database

Complexity


Size
Key-value pair
Column oriented
Document based
Graph
Complexity

Key-Value Pair

Frequently found in caching and fast-lookup apps

- Extremely simple interface
  - Data model: (key, value) pairs
  - Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)
- Implementation: efficiency, scalability, fault-tolerance
  - Records distributed to nodes based on keys
  - Replication
  - Single-record transactions, “eventual consistency”
- Example systems
  - Redis, Riak

Document-Based

Used when richer key-value querying is required

- Like key-value store except value is document
  - Data model: (key, document) pairs
  - Document: JSON, XML, other semi-structured formats
  - Basic operations:
  - Insert(key,document), Fetch(key), Update(key), Delete(key)
- Example systems
  - CouchDB, MongoDB, Riak, ...

Column Oriented

Used when richer key-value querying is required

- Like key-value store except value is document
  - Data model: columnar stores
  - Document: structured data designed to scale to large size
  - Basic operations:

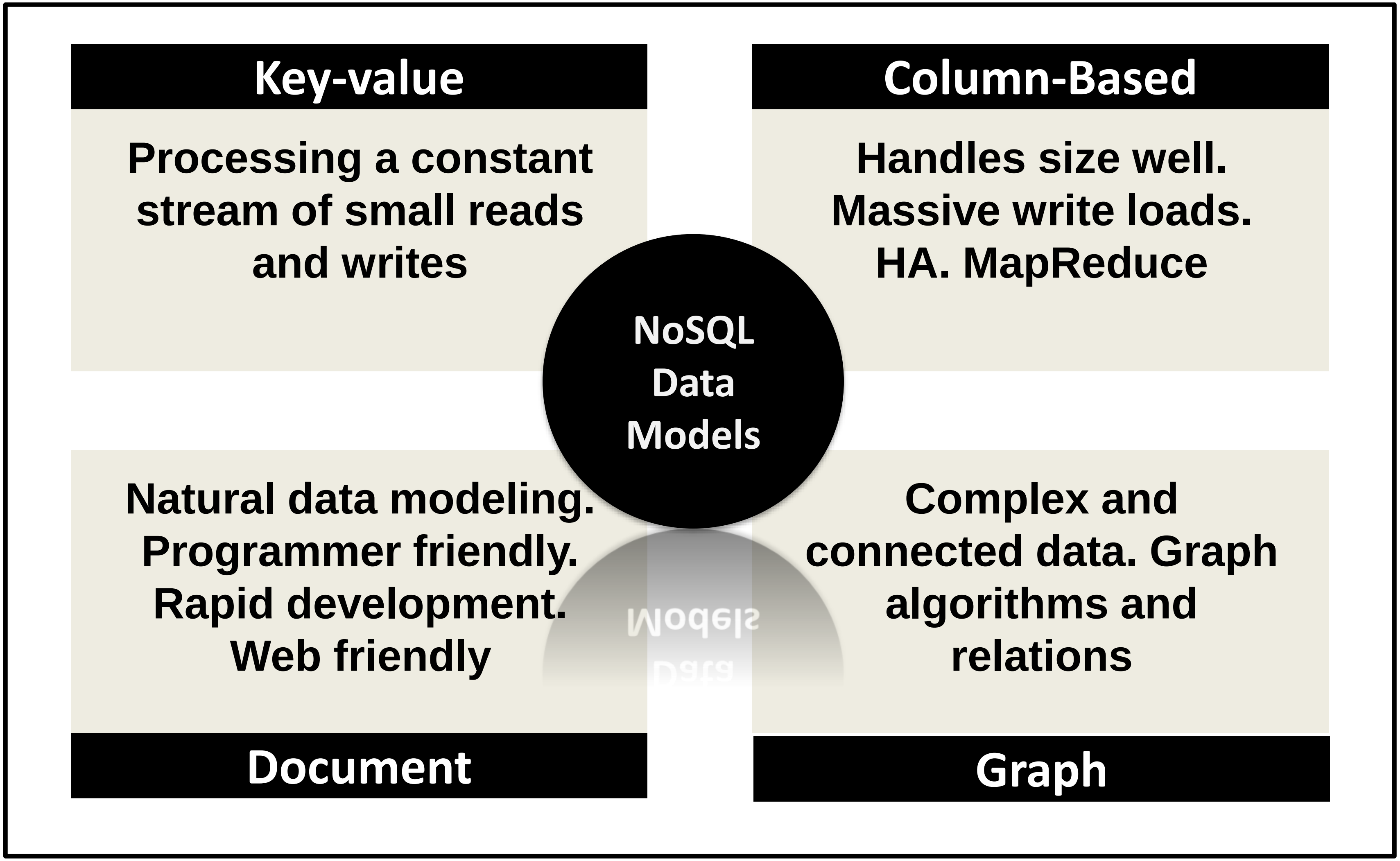
- Example systems
  - Hbase, Cassandra

Graph Database

Used to simplify relationship navigation

- Graph database systems
  - Data model: nodes and edges
  - Nodes may have properties (including ID)
  - Edges may have labels or roles
  - Interfaces and query languages vary
- Example systems
  - Neo4J, DSE Graph, GraphDB, …….

Which One To Use?



Key-value

Processing a constant stream of small reads and writes

Column-Based

Handles size well. Massive write loads. HA. MapReduce

NoSQL Data Models

Natural data modeling. Programmer friendly. Rapid development. Web friendly

Document

Complex and connected data. Graph algorithms and relations

Graph

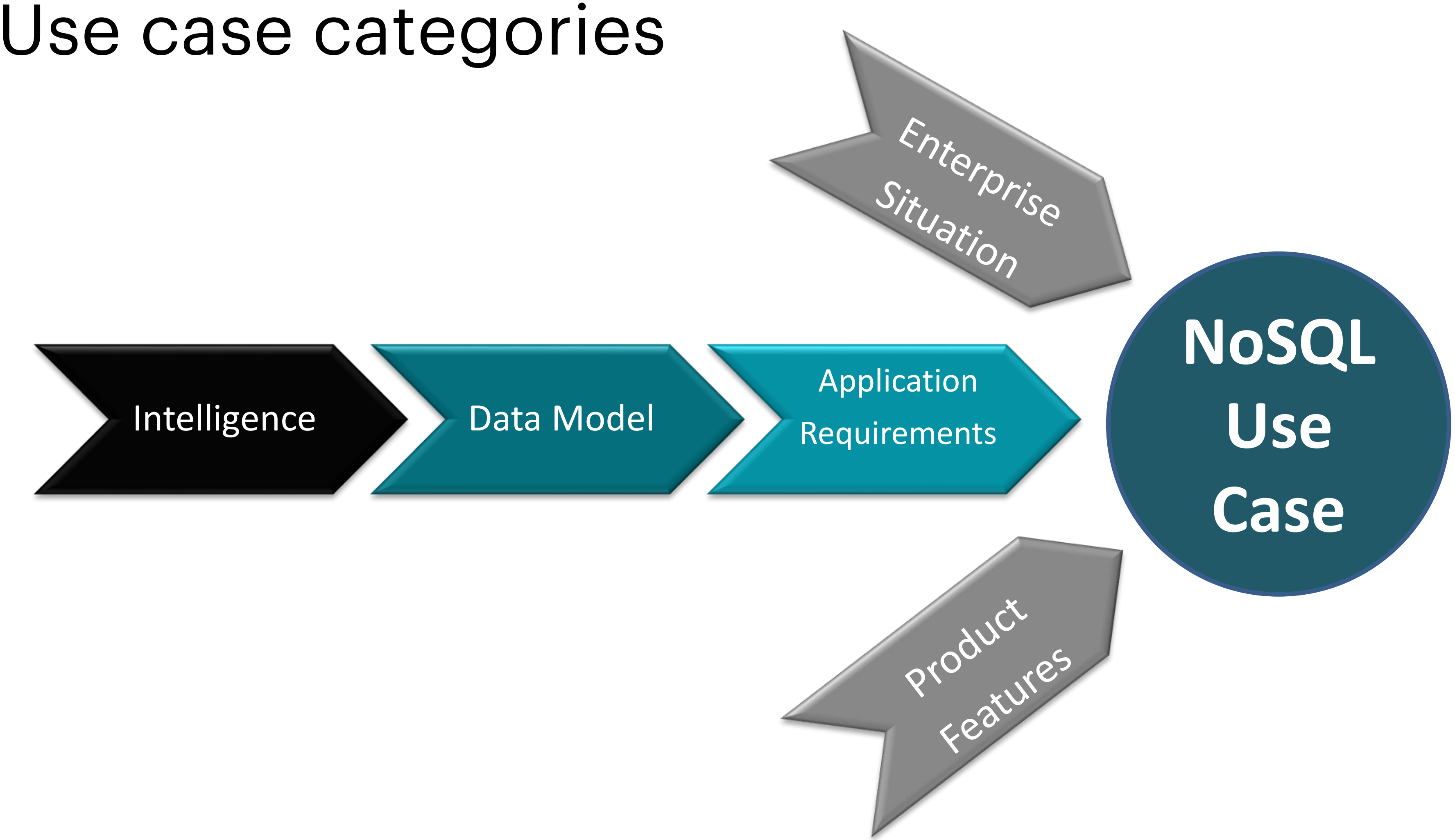
Beyond Data Models

Choosing a solution by data model alone is not enough

Need a classification that would actually allow an observer to determine whether or not the solution category is appropriate for a given use case?

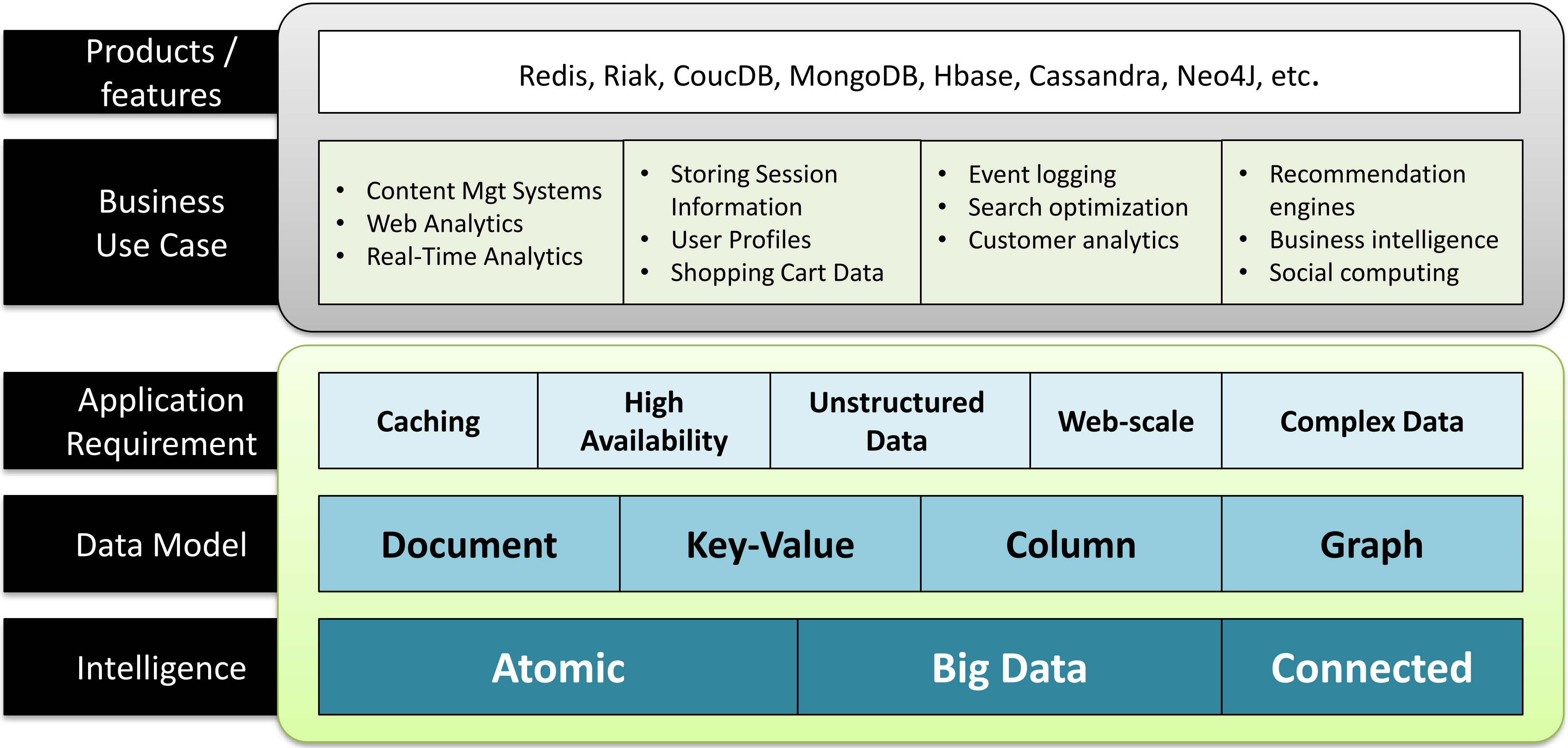
NoSQL Solutions

Use case categories



Use Case Categories

Non-exhaustive list of use case categories



Products / features

Redis, Riak, CoucDB, MongoDB, Hbase, Cassandra, Neo4J, etc.

Business Use Case

- Content Mgt Systems
- Web Analytics
- Real-Time Analytics

- Storing Session Information
- User Profiles
- Shopping Cart Data

- Event logging
- Search optimization
- Customer analytics

- Recommendation engines
- Business intelligence
- Social computing

Application Requirement

Caching | **High Availability** | **Unstructured Data** | **Web-scale** | **Complex Data**

Data Model

Document | **Key-Value** | **Column** | **Graph**

Intelligence

Atomic | **Big Data** | **Connected**

Agenda

- Introduction
- Technical Overview
- Use Cases
- Under The Hood: Compare & Contrast

1. Social Media

Atomic + Key-Value + High Availability

Background

- Yammer is an enterprise social network
- Huge data to manage from its rapidly growing user base
- Data is always updated
- **Needed to build a new notifications feature**
- Gives the user a sorted set of notifications
- Call to action based on the nature of the notification

yammer

Challenge

- Data size = 2+ Terabytes
- Duplicate data and stability concerns due to difficulty with replication and database crashes
  - Data is stored in a Postgres data store
  - Postgres provides consistency of data guarantees at the expense of availability
  - Need for high availability (HA)

NoSQL Approach

- Employ a reliable, scalable NoSQL solution
- High availability is paramount
- Amazon – Dynamo model fits use case
- Dynamo-inspired projects – (Riak & Voldemort)
- Riak chosen because of stability and very low latency

Results

- Yammer now has a robust Notifications module in its social collaboration tool
- No increase its data footprint on its single point of failure
- Very low latency
- Highly available data powering the notifications

2. Data Management

Atomic + Document-Based + Web Scale

Background

- The Compact Muon Solenoid Experiment (CMS) at CERN
- Data Management and Workflow Management (DMWM)
- Provides all offline processing infrastructure to CMS
  - Data cataloging
  - Data transfer
  - Creating simulated data

Compact
Muon
Solenoid
experiment at
CERN's LHC
CMS

Challenge

- CMS will collect roughly 10PB of data per year.
- Problems that don't fit well with standard relational databases or file systems
- Small number of users, but an amount of data similar to Facebook's
- Needed a solution that could handle large amounts of data, often without metadata, quickly in a distributed environment in which incoming database connections are frequently impossible

NoSQL Approach

- Employ a NoSQL solution designed for distributed environments
- Capable of handling large number numbers of transactions
- No Need to Manage a Complex Replication Infrastructure
- MongoDB and CouchDB have these features
- CouchDB chosen – speed of development

Results

- DMWM team don't have to write and maintain large pieces of code
- Rapid application development / deployment

3. Search Optimization

Atomic + Document-Based + Caching

Background

- ebay – large BASE environments based on Oracle DB
- Every database is shared and partitioned
- Logical hosts are mapped to physical based on static mapping tables which are controlled by the DBAs
- Common ORM framework (DAL) provides powerful and consistent patterns for data scalability

Challenge

- ORM is not the fastest way to develop
- Search suggestion
- Need to use RAM more aggressively and seamlessly to speed up queries
- Must have <60 – 70 msec round trip end to end

NoSQL Approach

- Employ a reliable, document based NoSQL solution
- Caching is important
- Data sets to fit in RAM
- Single replica set, no shards
- MongoDB chosen
    - Multiple indexes allow flexible lookups
    - In-memory data placement ensures lookup speed
    - Large data set is durable and replicated

Results

- Search suggest list is a MongoDB document indexed by work prefix as well as by some metadata; product category, search domain, etc.
- MongoDB query < 1.4 msec

4. Online Streaming

Big Data + Column-Based + Web Scale + HA

Background

- Netflix is a provider of on-demand Internet streaming media
- In addition to streaming more titles to more devices in both the US and Canada, Netflix has moved its infrastructure, data, and applications to the AWS cloud.
- Goal is infinite scale

Challenge

- Pick a data store suitable for the Cloud
- Translate RDBMS concepts to key-value store
- Work around issues specific to the chosen KV store
- Create a bi-directional DC-Cloud data replication pipeline

NoSQL Approach

- Employ a highly durable cloud data store with writes automatically replicated across availability zones with a region – Amazon SimpleDB
- High performance column oriented distributed database solution, good for managing ever growing data volumes – HBase
- Cassandra at Netflix is used to hold both the member data set (aka Subscriber) and the A/B test data sets. It is also used to hold the streaming viewing history.

Results

- Netflix is the leading global content streaming platform
- Re-distribute load across nodes at runtime
- A single global Cassandra cluster can simultaneously service applications and asynchronously replicate data across multiple geographical locations

5. Content Management

Big Data + Column-Based + Web Scale

Background

- Nextbio is a life sciences research firm that helps pharmaceutical companies conduct genomic research
- 100-node Hadoop cluster – 100s of terabytes of data
- 3.2 billion base pairs behind each of the 100s of genomes studied

NEXTBIO

Challenge

- Big data – over 30 billion rows of information
- How to scale effectively across distributed system while spreading the storage and compute load across more servers
- Deliver optimal write and read performance

NoSQL Approach

- Because of need to scale, MySQL reached its limits
- Employ a highly scalable data store that integrates well with Hadoop
- Transactional platform for running high-scale, real-time applications
- HBase & Cassandra are possibilities
- Hbase chosen because it provides consistency, while Cassandra is known for availability.

Results

- Nextbio is able to scale effectively to handle the write-heavy workloads
- Tabular access to data with big data scale

6. Logistics

Connected + Graph + Complex + HA

Background

- One of the world's largest logistics carriers
- Projected to outgrow capacity of old system
- New parcel routing system
- Single source of truth for entire network
- B2C & B2B parcel tracking
- Real-time routing: up to 5M parcels per day

Deutsche Post

Challenge

- 24x7 availability, year round
- Peak loads of 2500+ parcels per second
- Complex and diverse software stack
- Need predictable performance & linear scalability
- Daily changes to logistics network: route from any point, to any point

NoSQL Approach

- Neo4j provides the ideal domain fit:
- a logistics network is a graph
- Extreme availability & performance with Neo4j clustering
- "Whiteboard friendly" model easy to understand

Results

- Hugely simplified queries, vs. relational for complex routing
- Flexible data model can reflect real-world data variance much better than relational

7. Workforce Management

Connected + Graph + Complex

Background

- Largest provider of contingent workforce management solutions in the health care industry
- Full set of SaaS solutions allowing hospitals and agencies to manage internal & external staffing
- Connects 1700+ health care facilities to 1000+ staffing vendors, w/130K+ health care professionals.

ShiftWise™

Challenge

- Recommending the right person for the right shift
- Matching profiles to staffing orders based on skills, location, schedule, and other qualifying criteria
- Managing the flow of jobs between critical care hospitals, staffing agencies, and staff
- Scaling beyond skilled nursing and allied care, to physicians, ambulatory care, and IT workers

NoSQL Approach

- Enable a new architecture which will address long-standing issues in the core application
- Enable scaling required by the business
- Schema flexibility: overcome struggles with the inflexibility of the relational DBMS
- New system of record, using Neo4j & PostgreSQL

Results

- Gradual retirement of legacy Microsoft SQL Server architecture, which is less flexible and less scalable
- Performance: timely execution of complex recommendations

8. Recommendation

Connected + Graph + Complex + HA

Background

- Cisco.com serves customer and business customers with Support Services
- Needed real-time recommendations, to encourage use of online knowledge base

Challenge

- Call center volumes needed to be lowered by improving the efficacy of online self service
- Leverage large amounts of knowledge stored in service cases, solutions, articles, forums, etc.
- Problem resolution times, as well as support costs, needed to be lowered

NoSQL Approach

- Cases, solutions, articles, etc. continuously scraped for cross-reference links, and represented in Neo4j
- Real-time reading recommendations via Neo4j
- Neo4j Enterprise with HA cluster

Results

- The result: customers obtain help faster, with decreased reliance on customer support

9. Social, Access Control

Connected + Graph + Complex + HA

Background

- One of the ten largest software companies globally
- $4B+ in revenue. Over 11,000 employees.
- Launched Creative Cloud in 2012, allowing its Creative Suite users to collaborate via the Cloud

Adobe

Challenge

- Needed highly robust and available, 24x7 distributed global system - collaboration for users of its highest revenue product line
- Storing creative artifacts in the cloud meant managing access rights for (eventually) millions of users, groups, collections, and pieces of content
- Complex access control rules controlling who was connected to whom, and who could see or edit what, proved a significant technical challenge

NoSQL Approach

- Selected Neo4j to meet very aggressive project deadlines. The flexibility of the graph model, and performance, were the two major selection factors.
- Easily evolve the system to meet tomorrow's needs
- Extremely high availability and transactional performance requirements. 24x7 with no downtime.

Results

- Neo4j allows consistently fast response times with complex queries, even as the system grows
- First (and possibly still only) database cluster to run across three Amazon EC2 regions: U.S., Europe, Asia

10. Resource Management

Connected + Graph + Complex + HA

Background

- 10th largest Telco provider in the world, leading in the Nordics
- Online self-serve system where large business admins manage employee subscriptions and plans
- Mission-critical system whose availability and responsiveness is critical to customer satisfaction

telenor

Challenge

- Degrading relational performance. User login taking minutes while system retrieved access rights
- Millions of plans, customers, admins, groups. Highly interconnected data set w/massive joins
- Nightly batch workaround solved the performance problem, but meant data was no longer current

NoSQL Approach

- Moved authorization functionality from Sybase to Neo4j
- Modeling the resource graph in Neo4j was straightforward, as the domain is inherently a graph

Results

- Able to retire the batch process, and move to real-time responses: measured in milliseconds
- Users able to see fresh data, not yesterday’s snapshot
- Customer retention risks fully mitigated

Agenda

- Introduction
- Technical Overview
- Use Cases
- Under The Hood: Compare & Contrast

RDBMS Vs Graph

Consider the following entities

Users

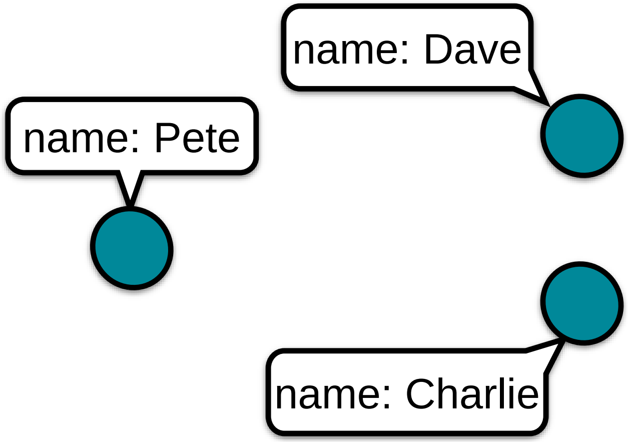
Dave Charlie Pete

RDBMS

User

id	name
1	Dave
2	Charlie
3	Pete

Graph



Compare & Contrast (1)

Finding Entities

Cypher

```
START user = node:users(id = ’2’)
RETURN user.name
```

SQL

```
SELECT name
FROM User
WHERE id = 2
```

RDBMS

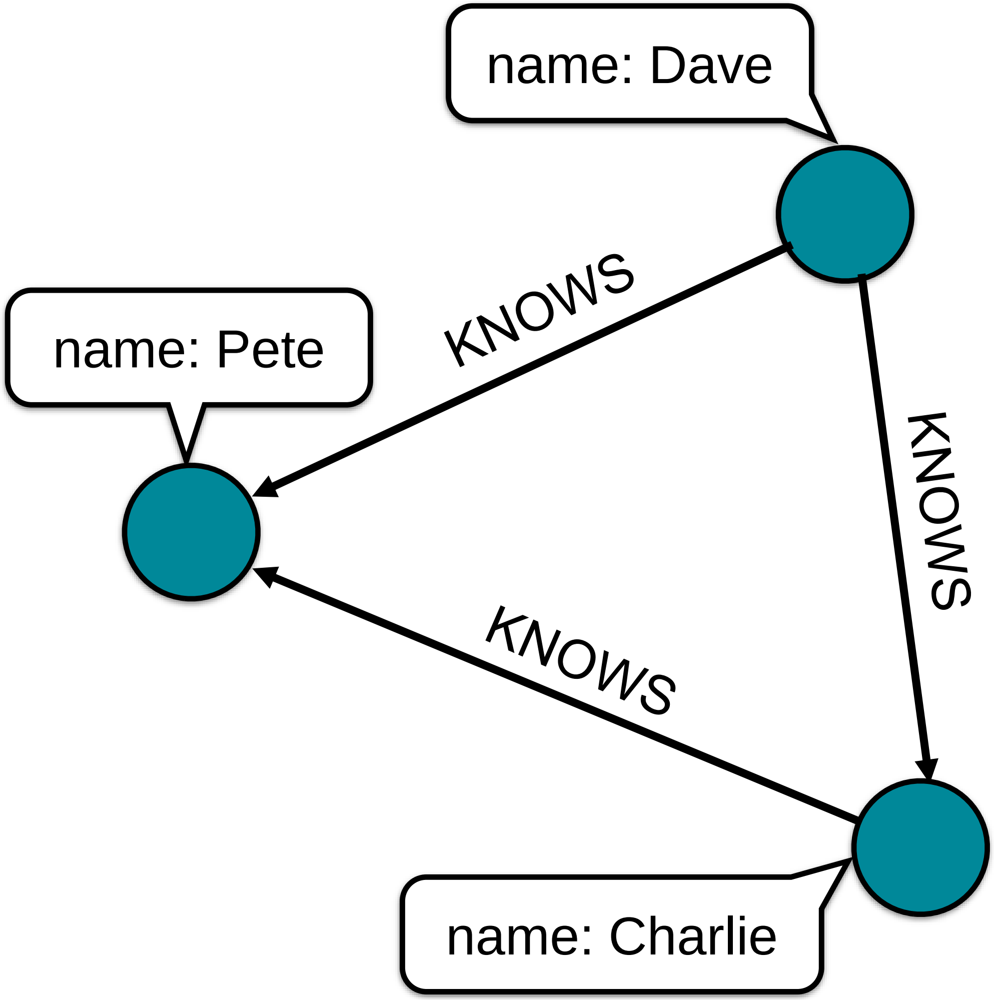
User

id	name
1	Dave
2	Charlie
3	Pete

Knows

src	dst
1	2
1	3
2	3

Graph



Compare & Contrast (2)

Finding Friends

```
START user = node:users(id = ’2’)
MATCH user-[:KNOWS]-friend
RETURN friend.name
```

```
SELECT name
FROM User
WHERE id IN (  SELECT dst FROM Knows
WHERE src = 2  UNION ALL  SELECT src
FROM Knows WHERE dst = 2);
```

Entities

Users

Dave Charlie Pete

Products

Socks Couch

RDBMS

User

id	name
1	Dave
2	Charlie
3	Pete

Knows

src	dst
1	2
1	3
2	3

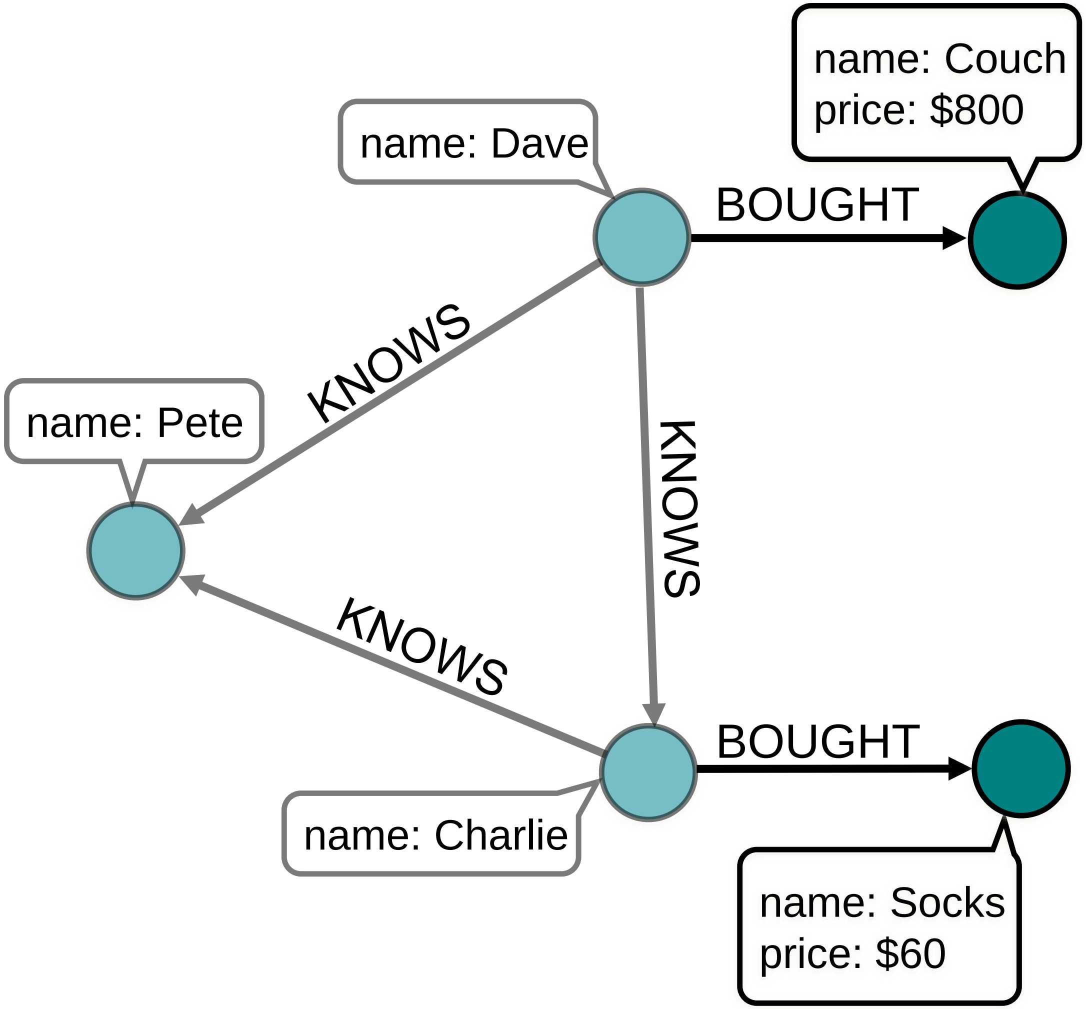
Product

id	name	price
10	Socks	$60
30	Couch	$800

Bought

user	prod
1	30
2	10

Graph


name: Couch
price: $800
name: Dave
BOUGHT
KNOWS
name: Pete
KNOWS
KNOWS
BOUGHT
name: Charlie
name: Socks
price: $60

Compare & Contrast (3)

What did your friends buy?

```
START user = node:users(id = ’2’)
MATCH user-[:KNOWS]-friend-[:BOUGHT]-product
RETURN friend.name, product.name
```

```
SELECT User.name as Friend, Product.nameFROM
User
JOIN Bought ON User.id = Bought.user
JOIN Product ON Bought.prod = Product.id
WHERE id IN (SELECT dst FROM Knows WHERE src =
2 UNION ALL  SELECT src FROM Knows WHERE dst =
2)
```

Entities

Users

Dave Charlie Pete

Products

Socks Couch

Categories

Clothing Furniture

RDBMS

Category

id	name
100	Clothing
200	Furniture

User

id	name
1	Dave
2	Charlie
3	Pete

Knows

src	dst
1	2
1	3
2	3

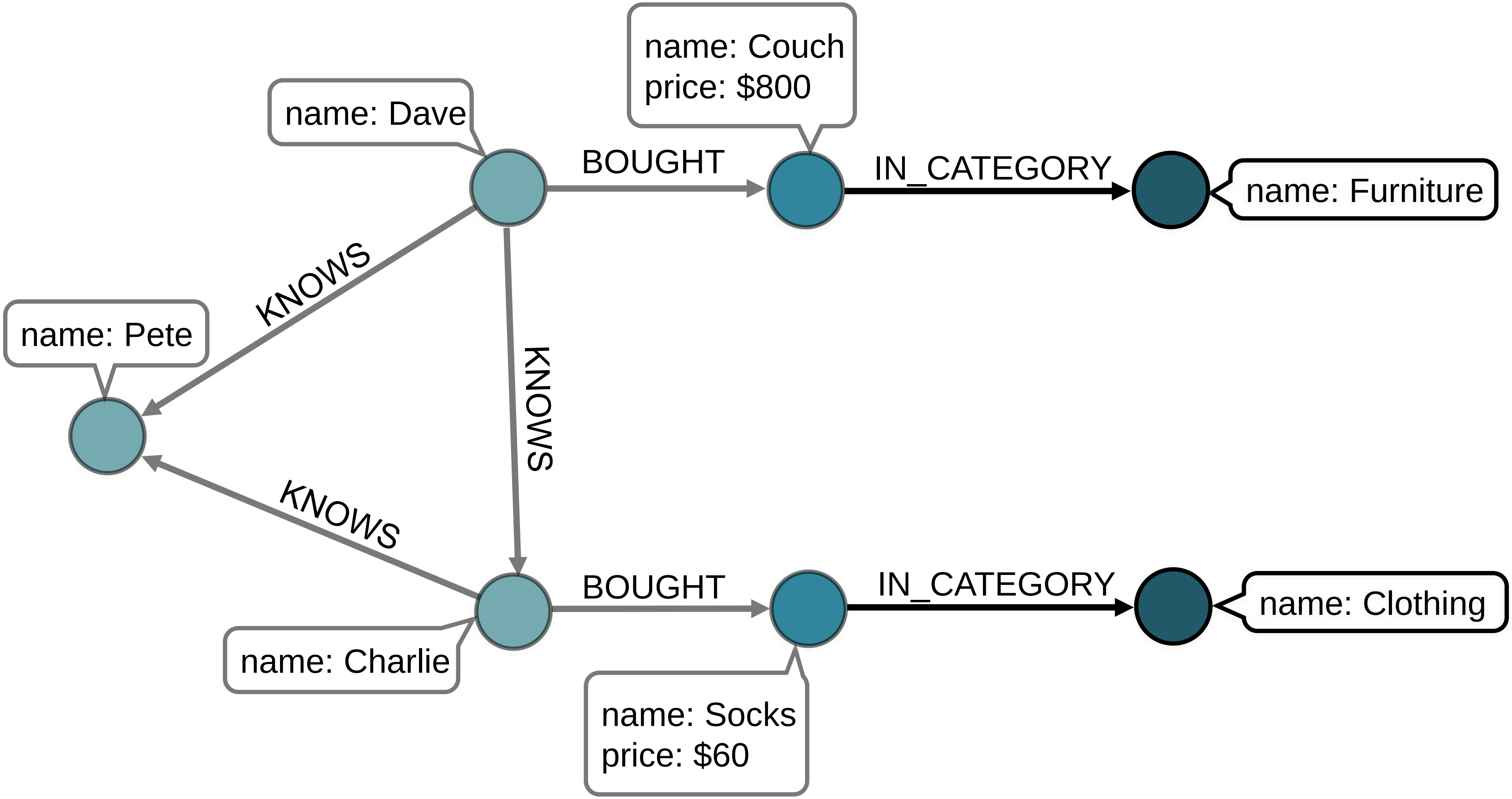
Product

id	name	price	*ctgry*
10	Socks	$60	100
30	Couch	$800	200

Bought

user	prod
1	30
2	10

Graph


name: Couch
price: $800
name: Dave
BOUGHT
IN_CATEGORY
name: Furniture
KNOWS
name: Pete
KNOWS
KNOWS
BOUGHT
IN_CATEGORY
name: Clothing
name: Charlie
name: Socks
price: $60

Compare & Contrast (4)

What categories do you shop in?

```
START user = node:users(id = ’2’)
MATCH user-[:BOUGHT]-product-[:IN_CATEGORY]-category
RETURN category, COUNT(category)
```

```
SELECT Category.name
FROM UserJOIN Bought ON User.id = Bought.user
JOIN Product ON Bought.prod = Product.id
JOIN Category ON Product.ctgry = Category.id
WHERE User.id = 2;
```

RDBMS

Category

id	name
100	Clothing
200	Furniture
300	Men’s

User

id	name
1	Dave
2	Charlie
3	Pete

Knows

src	dst
1	2
1	3
2	3

Product

id	name	color	price
10	Socks		$60
20	Blouse	red	$80
30	Couch		$800

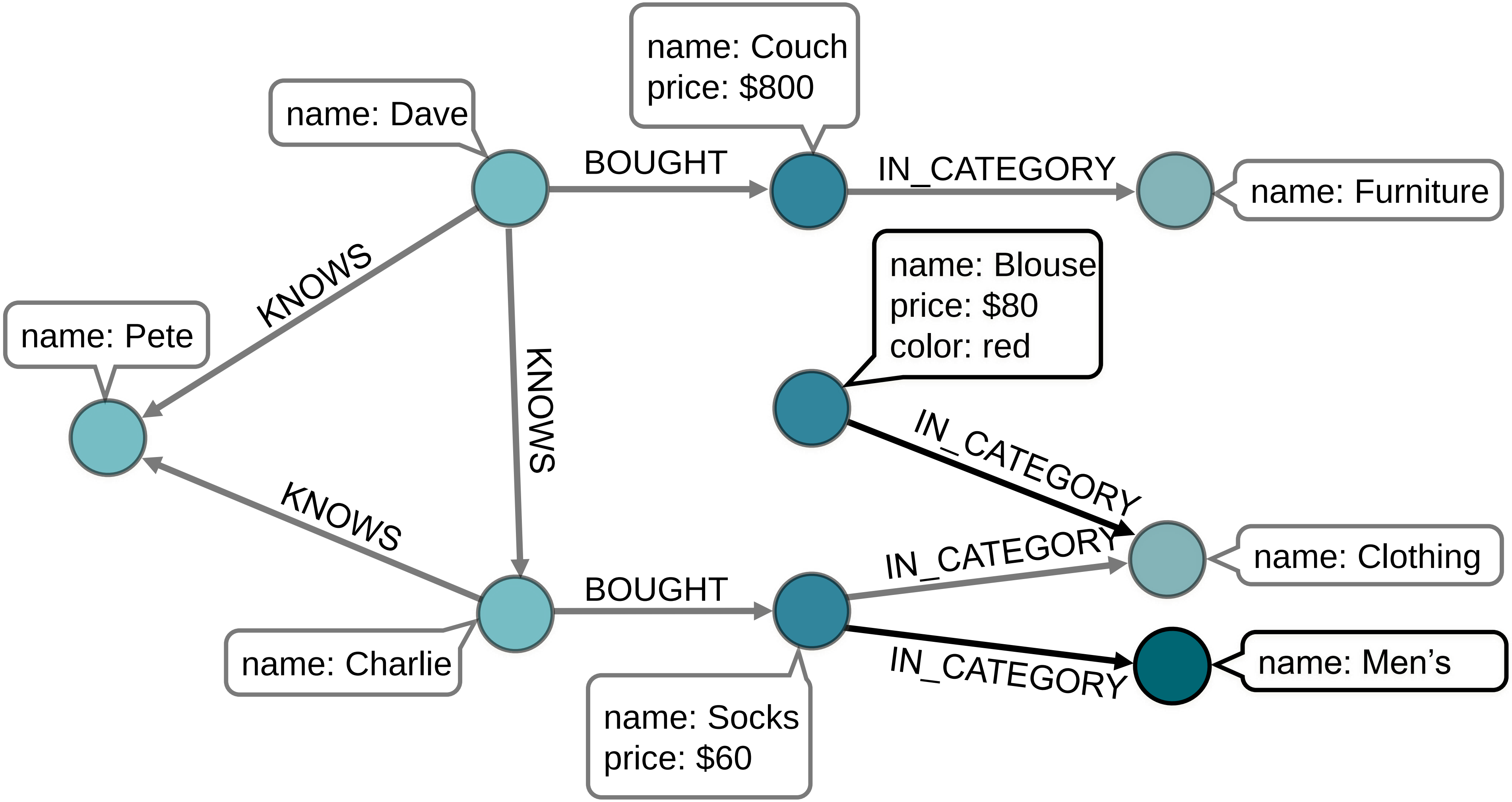
Prod_Ctgry

prod	ctgry
10	100
10	300
20	100
30	200

Bought

user	prod
1	30
2	10

Graph


name: Couch
price: $800
name: Dave
BOUGHT
IN_CATEGORY
name: Furniture
name: Blouse
price: $80
color: red
KNOWS
name: Pete
KNOWS
IN_CATEGORY
KNOWS
IN_CATEGORY
name: Clothing
BOUGHT
IN_CATEGORY
name: Men’s
name: Charlie
name: Socks
price: $60

Compare & Contrast (5)

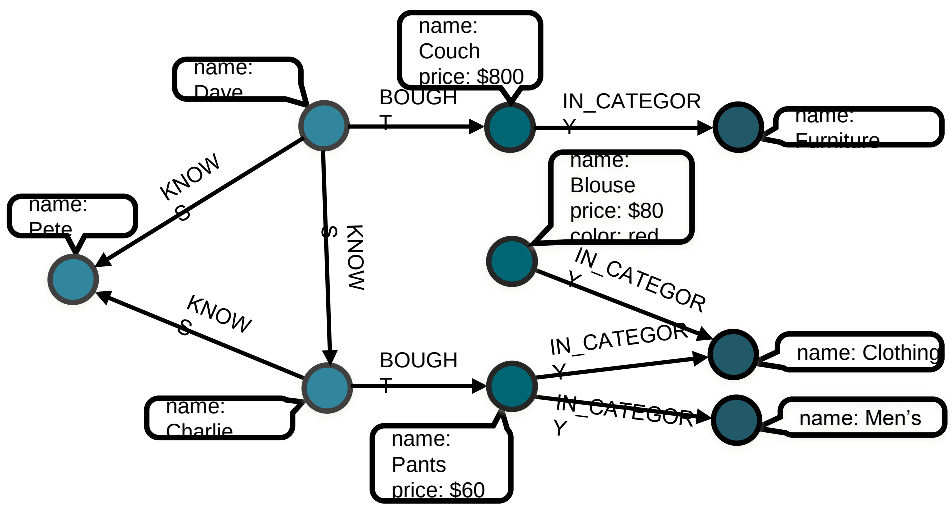
What categories do you shop in?

```
START user = node:users(id = ’2’)
MATCH user-[:BOUGHT]-product-[:IN_CATEGORY]-category
RETURN category, COUNT(category)
```

```
ALTER TABLE Product
ADD color varchar(255);

SELECT Category.name
FROM UserJOIN Bought ON User.id = Bought.user
JOIN Product ON Bought.prod = Product.id
JOIN Prod_Ctgry ON Product.id = Prod_Ctgry.prod
JOIN Category ON Prod_Ctgry.ctgry =
Category.idWHERE User.id = 2;
```

Result

Graph



RDBMS

Category

id	name
100	Clothing
200	Furniture
300	Men's

User

id	name
1	Dave
2	Charlie
3	Pete

Knows

src	dst
1	2
1	3
2	3

Product

id	name	color	price
10	Pants		$60
20	Blouse	red	$80
30	Couch		$800

Prod_Ctgry

prod	ctgry
10	100
10	300
20	100
30	200

Bought

user	prod
1	30
2	10

ANY
QUESTIONS
?