L24: NoSQL (continued)

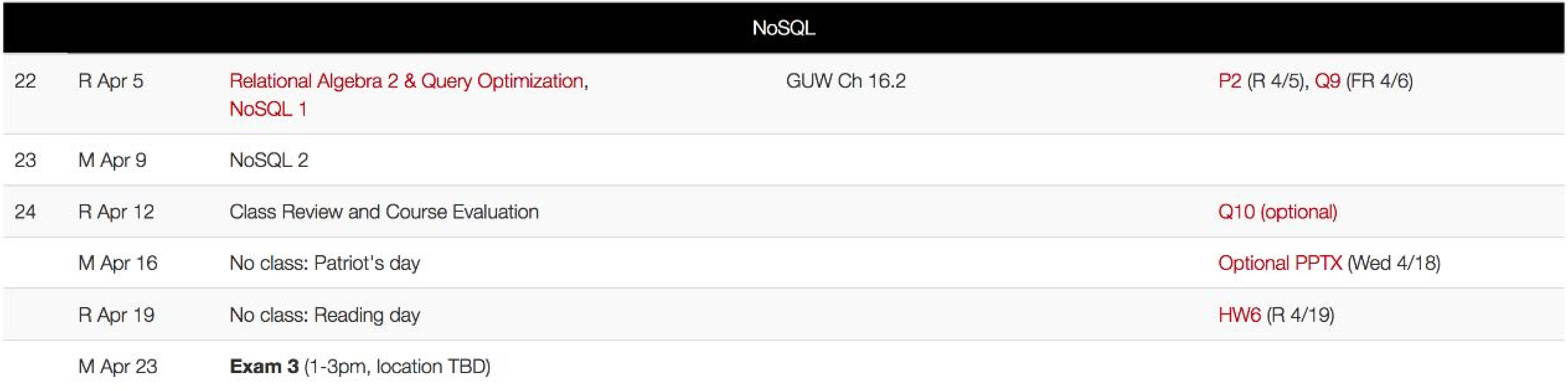
CS3200 Database design (sp18 s2)

https://course.ccs.neu.edu/cs3200sp18s2/

4/12/2018

Last Class today

- NoSQL (15min): Graph DBs
- Course Evaluation (15min)
- Course review

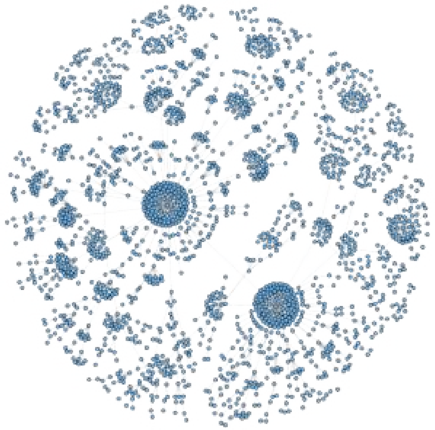


NoSQL				
22	R Apr 5	Relational Algebra 2 & Query Optimization, NoSQL 1	GUW Ch 16.2	P2 (R 4/5), Q9 (FR 4/6)
23	M Apr 9	NoSQL 2		
24	R Apr 12	Class Review and Course Evaluation		Q10 (optional)
	M Apr 16	No class: Patriot's day		Optional PPTX (Wed 4/18)
	R Apr 19	No class: Reading day		HW6 (R 4/19)
	M Apr 23	**Exam 3** (1-3pm, location TBD)		

Outline

- Introduction
- Transaction Consistency
- 4 main data models
  - Key-Value Stores (e.g., Redis)
  - Column-Family Stores (e.g., Cassandra)
  - Document Stores (e.g., MongoDB)
  - Graph Databases (e.g., Neo4j)
- Concluding Remarks

Graph Databases



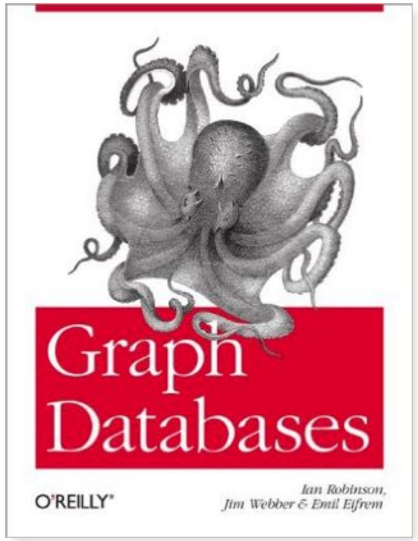
- Restricted case of a relational schema:
  - Nodes (+labels/properties)
  - Edges (+labels/properties)
- Motivated by the popularity of network/communication oriented applications
- Efficient support for graph-oriented queries
  - *Reachability*, *graph patterns*, *path patterns*
  - Ordinary RDBs either not support or inefficient for such queries
    - Path of length k is a k-wise self join; yet a very special one...
- Specialized languages for graph queries
  - For example, pattern language for paths
- Plus distributed, 2-of-CAP, etc.
  - Depending on the design choices of the vendor

Example Databases

- Graph with nodes/edges marked with labels and properties (labeled property graph)
  - Sparksee (DEX) (Java, 1st release 2008)
  - neo4j (Java, 1st release 2010)
  - InfiniteGraph (Java/C++, 1st release 2010)
  - OrientDB (Java, 1st release 2010)
- Triple stores: Support W3C RDF and SPARQL, also viewed as graph databases
  - MarkLogic, AllegroGraph, Blazegraph, IBM SystemG, Oracle Spatial & Graph, OpenLink Virtuoso, ontotext

neo4j

- Open source, written in Java
  - First version released 2010
- Supports the Cypher query language (declarative graph QL)
- Clustering support
  - Replication and sharding through master-slave architectures
- Used by ebay, Walmart, Cisco, National Geographic, TomTom, Lufthansa, ...

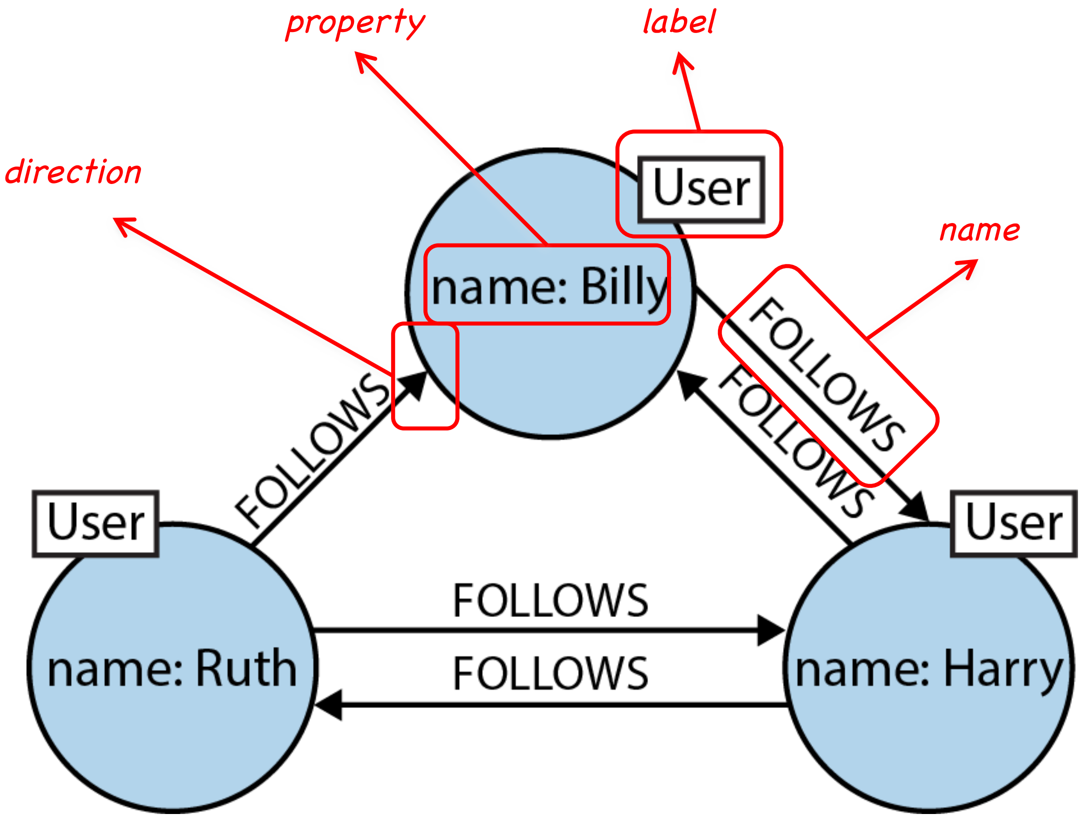


Examples taken from *Graph Databases* by Robinson, Webber, and Eifrem (O’Reilly) – free eBook

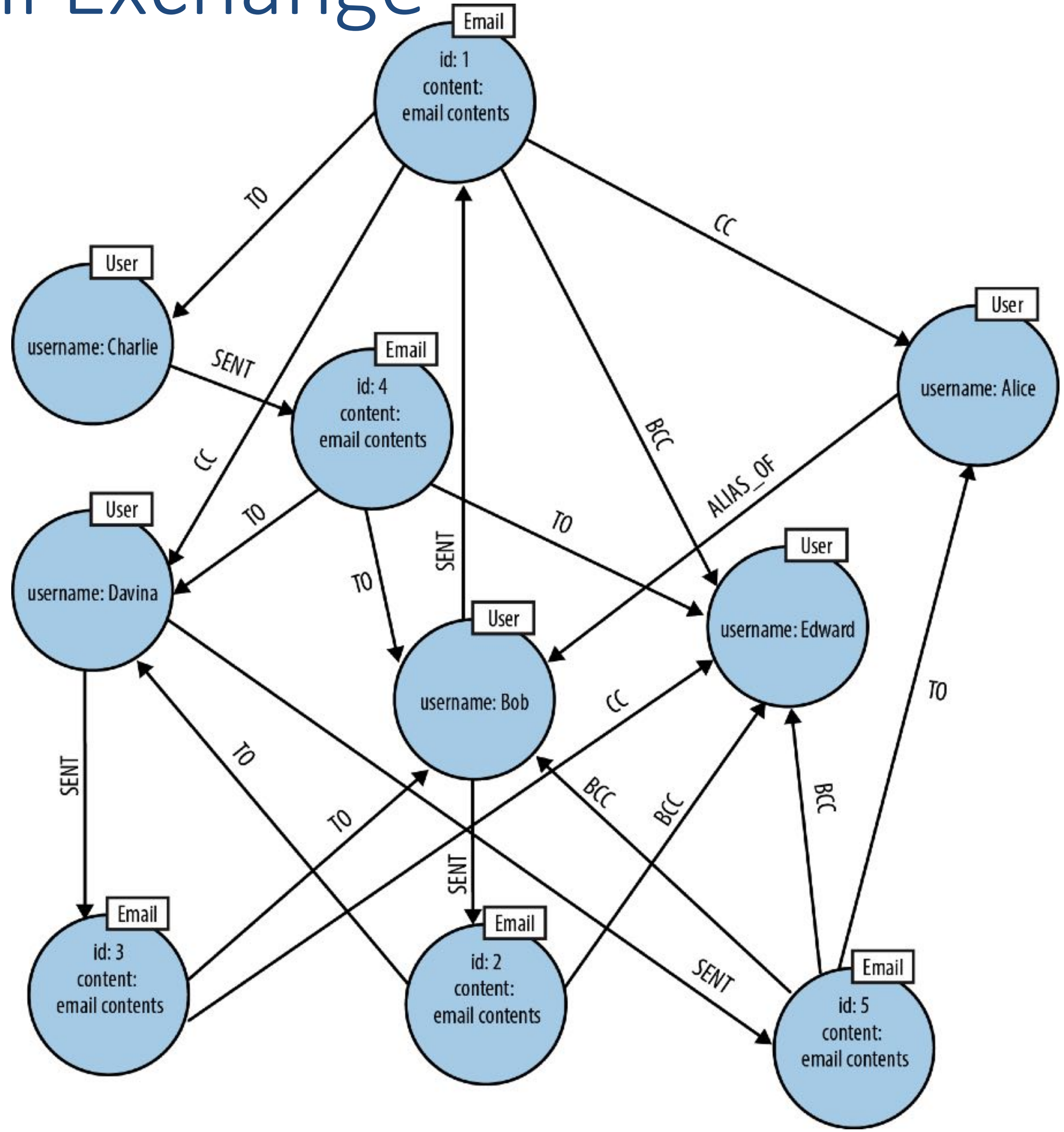
The Graph Data Model in Cypher

- Labeled property graph model
- Node
  - Has a set of *labels* (typically one label)
  - Has a set of *properties* key:value (where value is of a primitive type or an array of primitives)
- Edge (relationship)
  - Directed: node→node
  - Has a *name*
  - Has a set of *properties* (like nodes)

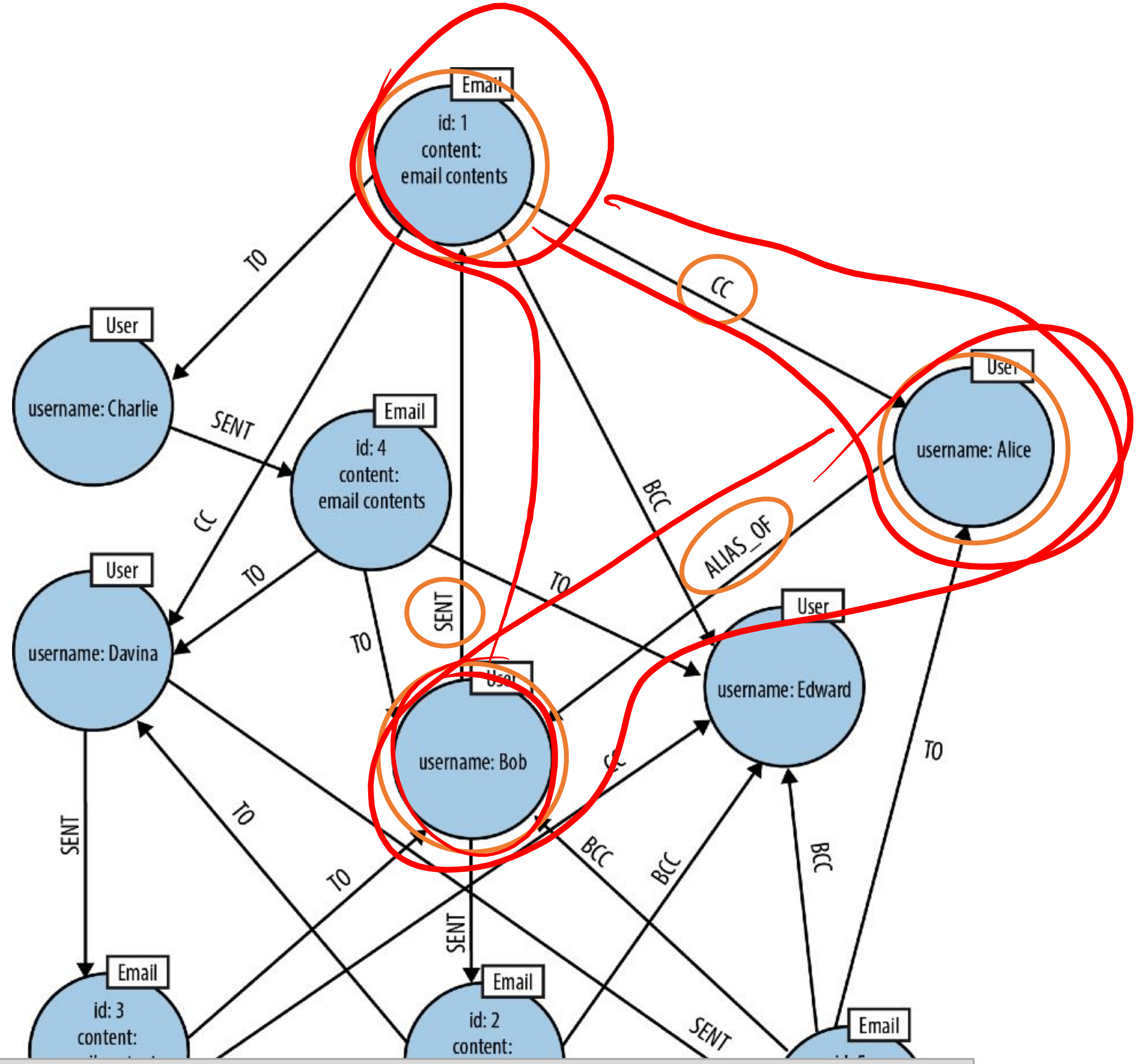
Example: Cypher Graph for Social Networks



Another Example: Email Exchange



Query Example

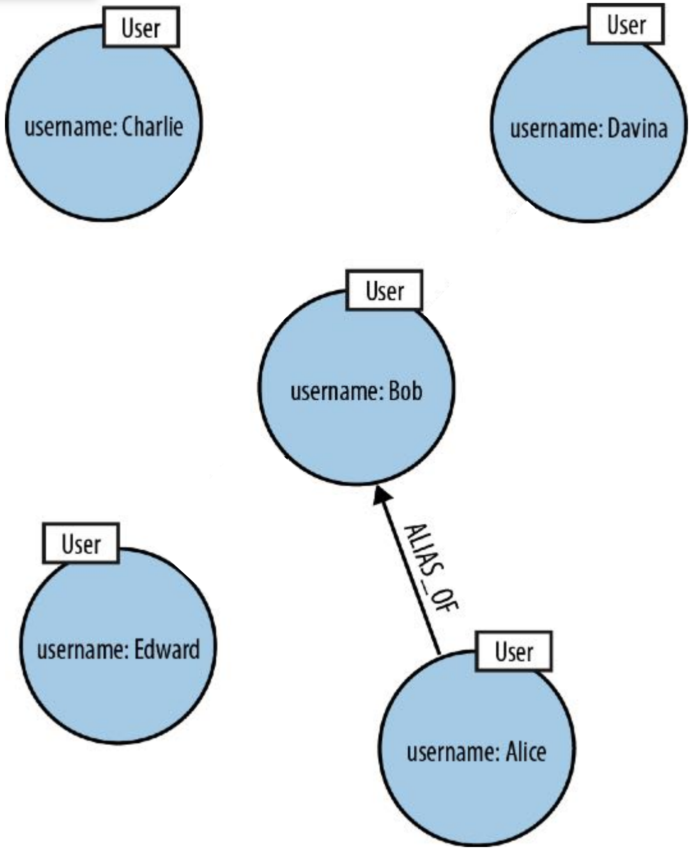


email
Node{id:"1",content:"..."}

```
MATCH (bob:User{username:'Bob'})-[:SENT]->(email)-[:CC]->(alias),
      (alias)-[:ALIAS_OF]->(bob)
RETURN email
```

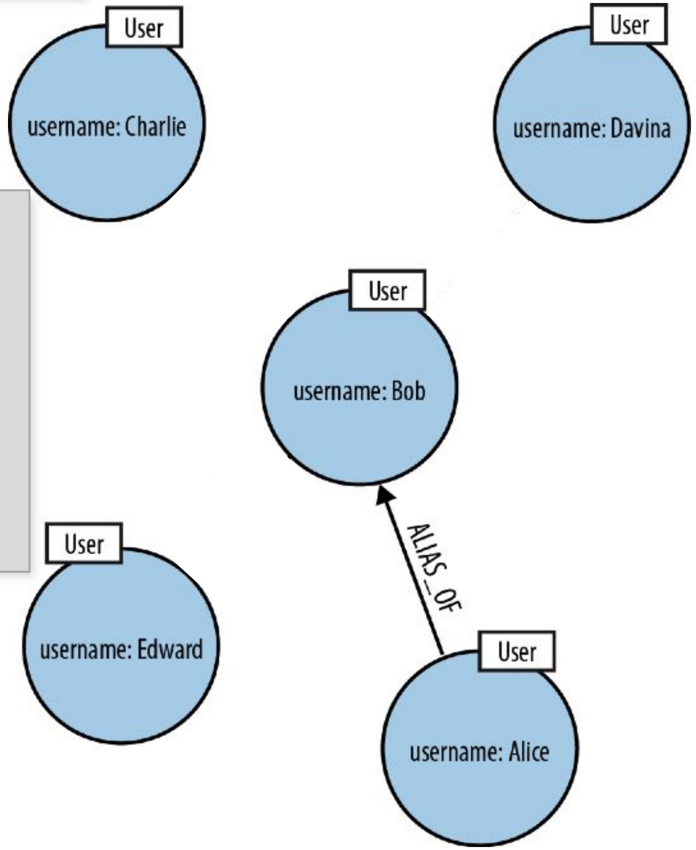
Creating Graph Data

```
CREATE (alice:User {username:'Alice'}),
       (bob:User {username:'Bob'}),
       (charlie:User {username:'Charlie'}),
       (davina:User {username:'Davina'}),
       (edward:User {username:'Edward'}),
       (alice)-[:ALIAS_OF]->(bob)
```



Creating Graph Data

```
CREATE (alice:User {username:'Alice'}),
       (bob:User {username:'Bob'}),
       (charlie:User {username:'Charlie'}),
       (davina:User {username:'Davina'}),
       (edward:User {username:'Edward'}),
       (alice)-[:ALIAS_OF]->(bob)
```



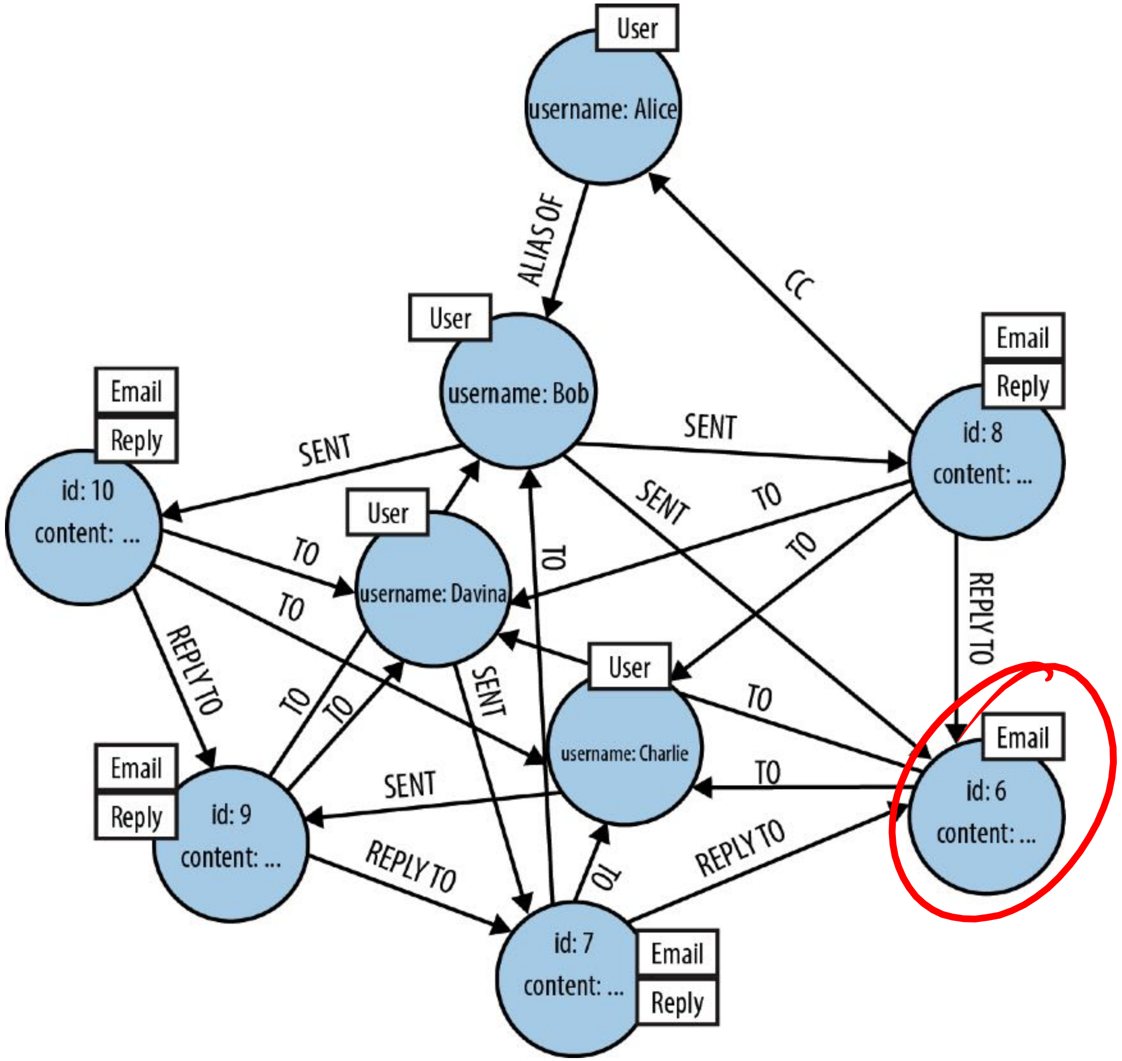
```
MATCH (bob:User {username:'Bob'}),
      (charlie:User {username:'Charlie'}),
      (davina:User {username:'Davina'}),
      (edward:User {username:'Edward'})
CREATE (bob)-[:EMAILED]->(charlie),
       (bob)-[:CC]->(davina),
       (bob)-[:BCC]->(edward)
```

Another Example

Path assignment

replier	depth
Davina	1
Bob	1
Charlie	2
Bob	3

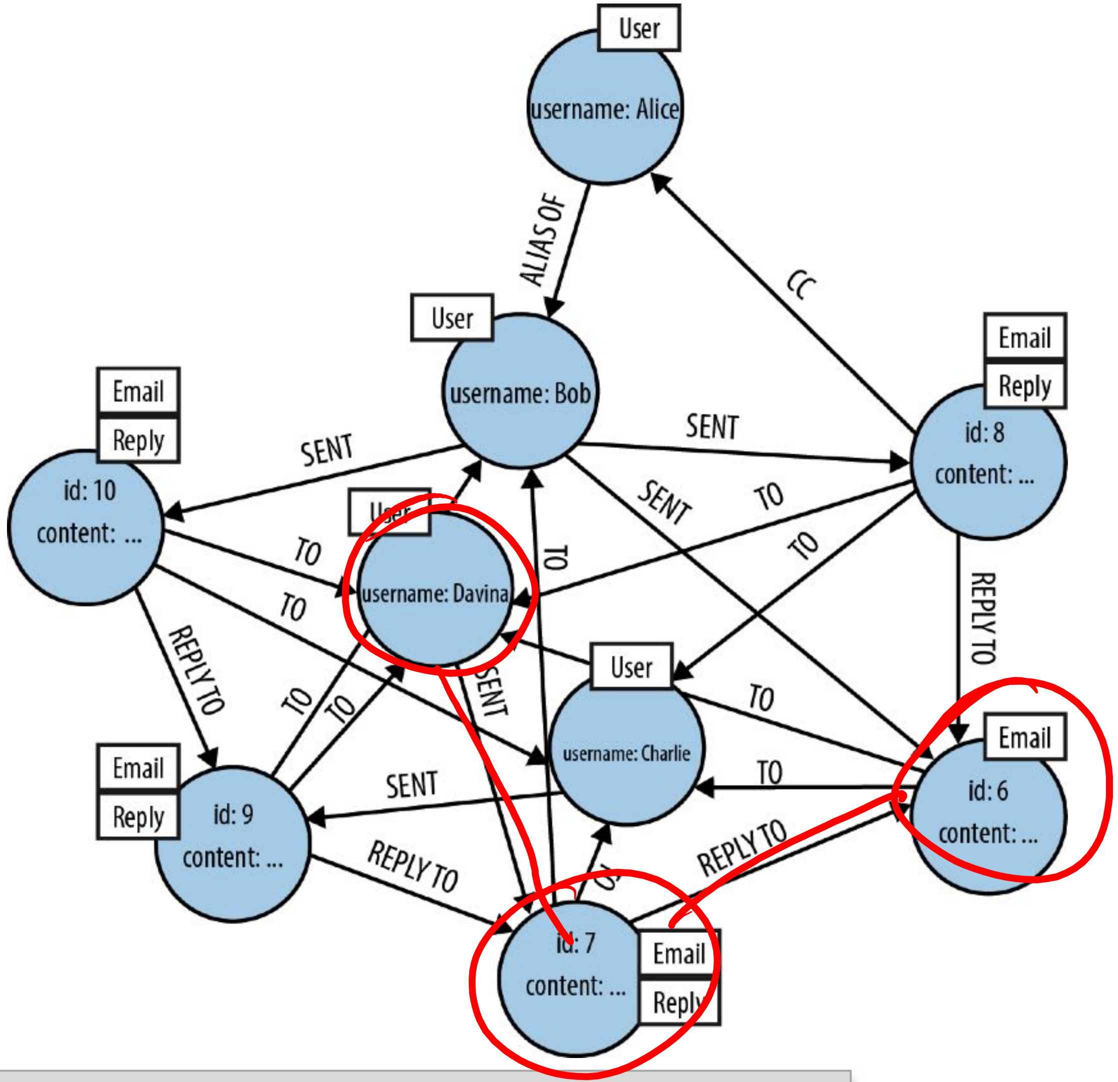
```
MATCH p = (email:Email {id:'6'})
      <-[:REPLY_TO*1..4]-(:Reply)<-[:SENT]-(replier)
RETURN replier.username AS replier, length(p) - 1 AS depth
ORDER BY depth
```



Another Example

Path assignment

replier	depth
Davina	1
Bob	1
Charlie	2
Bob	3

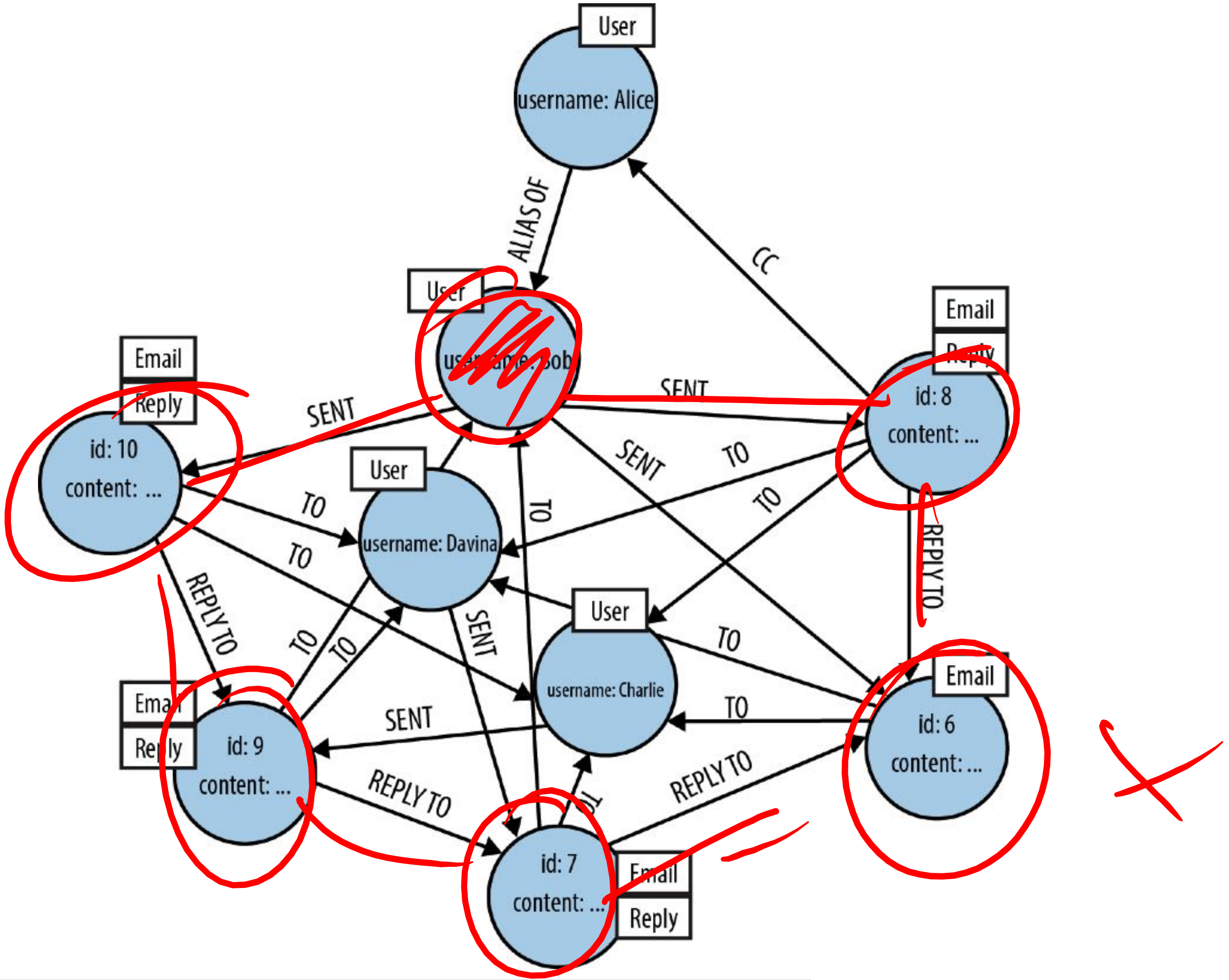


```
MATCH p = (email:Email {id:'6'})
      <-[:REPLY_TO*1..4]-(:Reply)<-[:SENT]-(replier)
RETURN replier.username AS replier, length(p) - 1 AS depth
ORDER BY depth
```

Another Example

Path assignment

replier	depth
Davina	1
Bob	1
Charlie	2
Bob	3



```
MATCH p = (email:Email {id:'6'})
      <-[:REPLY_TO*1..4]-(:Reply)<-[:SENT]-(replier)
RETURN replier.username AS replier, length(p) - 1 AS depth
ORDER BY depth
```

When to use it

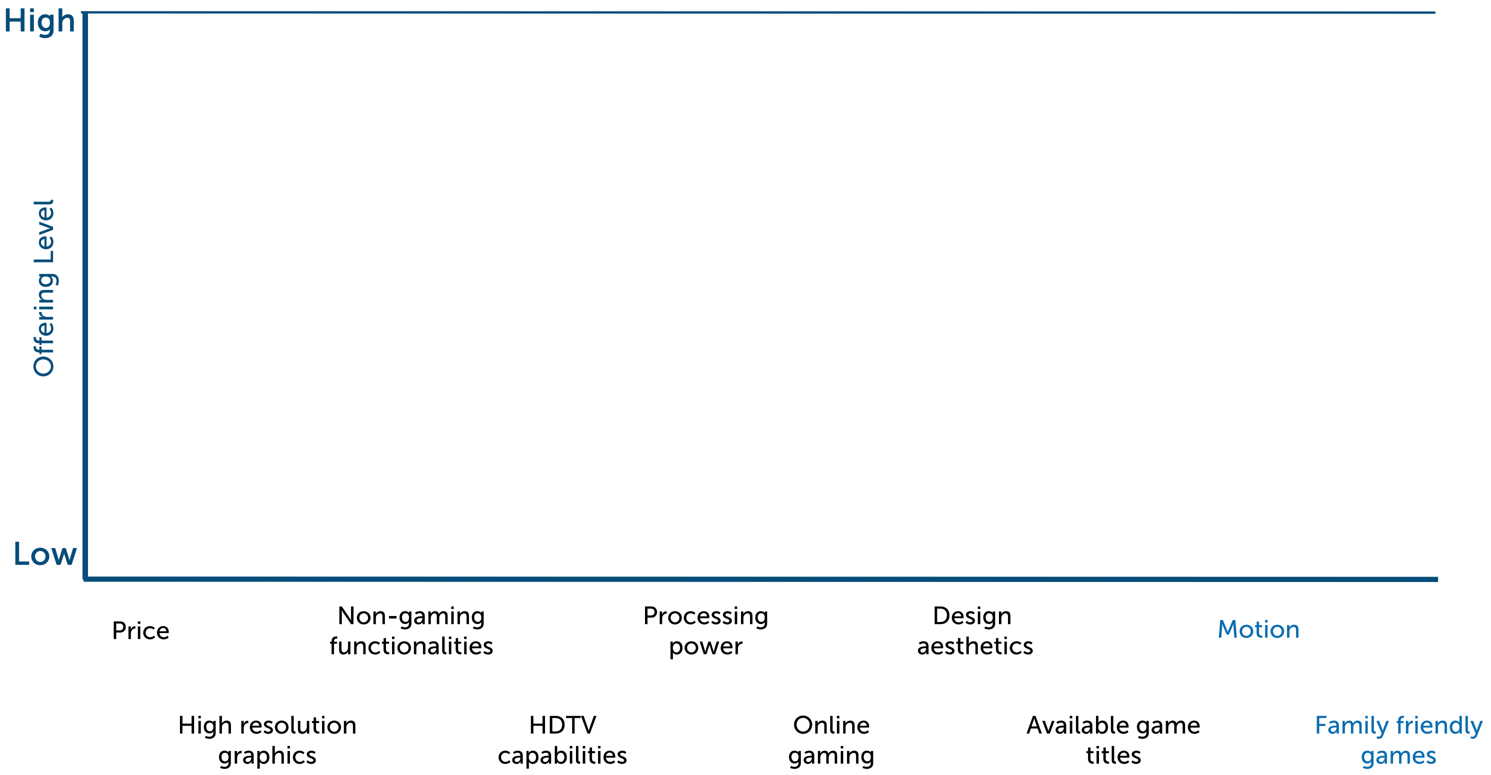
- Use it:
  - Connected data, e.g. social graphs, employees where they worked
  - Location-based services
  - Recommendation engines
- Don't use it:
  - Change properties on many entities

Outline

- Introduction
- Transaction Consistency
- 4 main data models
  - Key-Value Stores (e.g., Redis)
  - Column-Family Stores (e.g., Cassandra)
  - Document Stores (e.g., MongoDB)
  - Graph Databases (e.g., Neo4j)
- Concluding Remarks

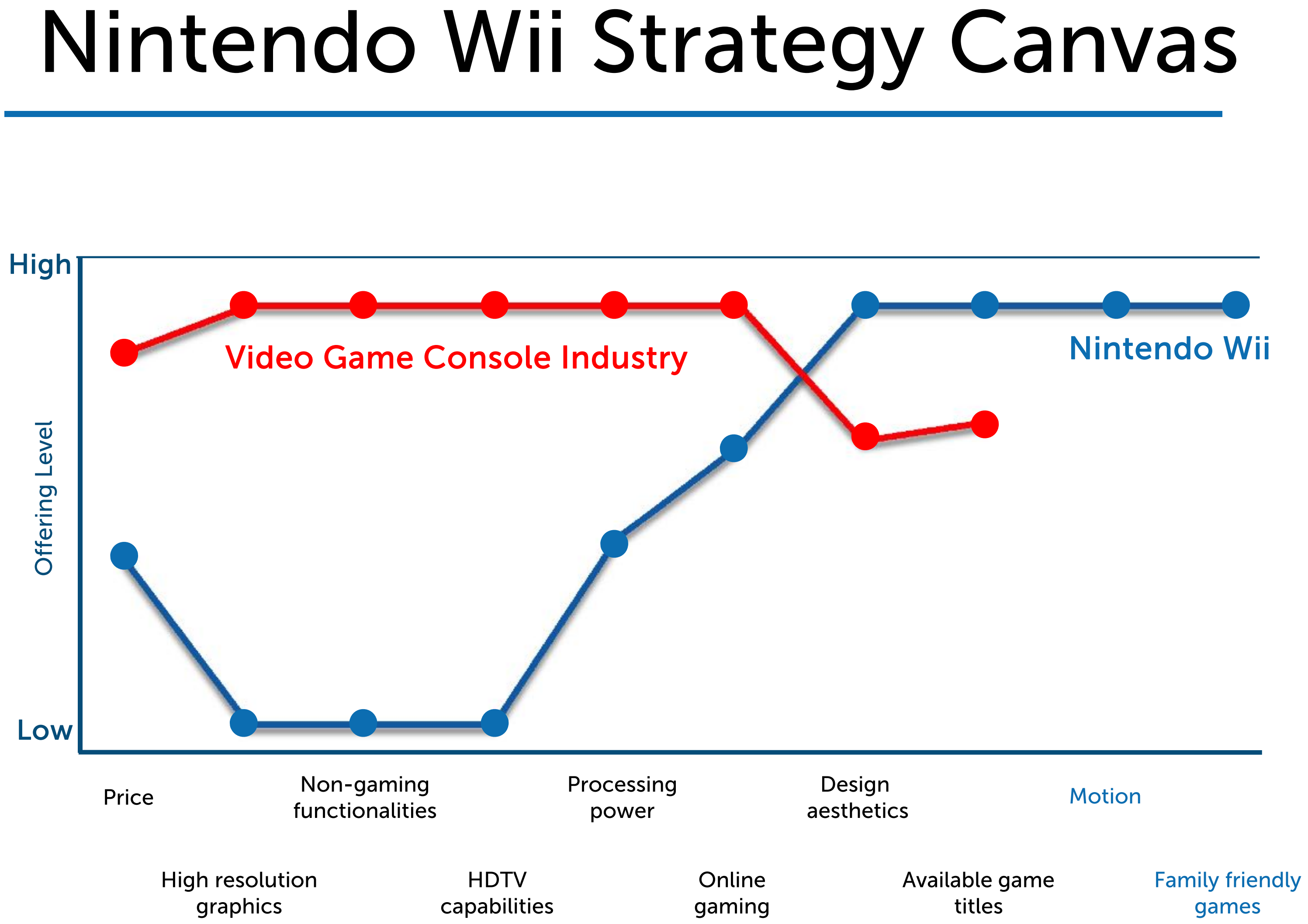
Strategy Canvas: Example Nintendo Wii (1/3)

Nintendo Wii Strategy Canvas



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

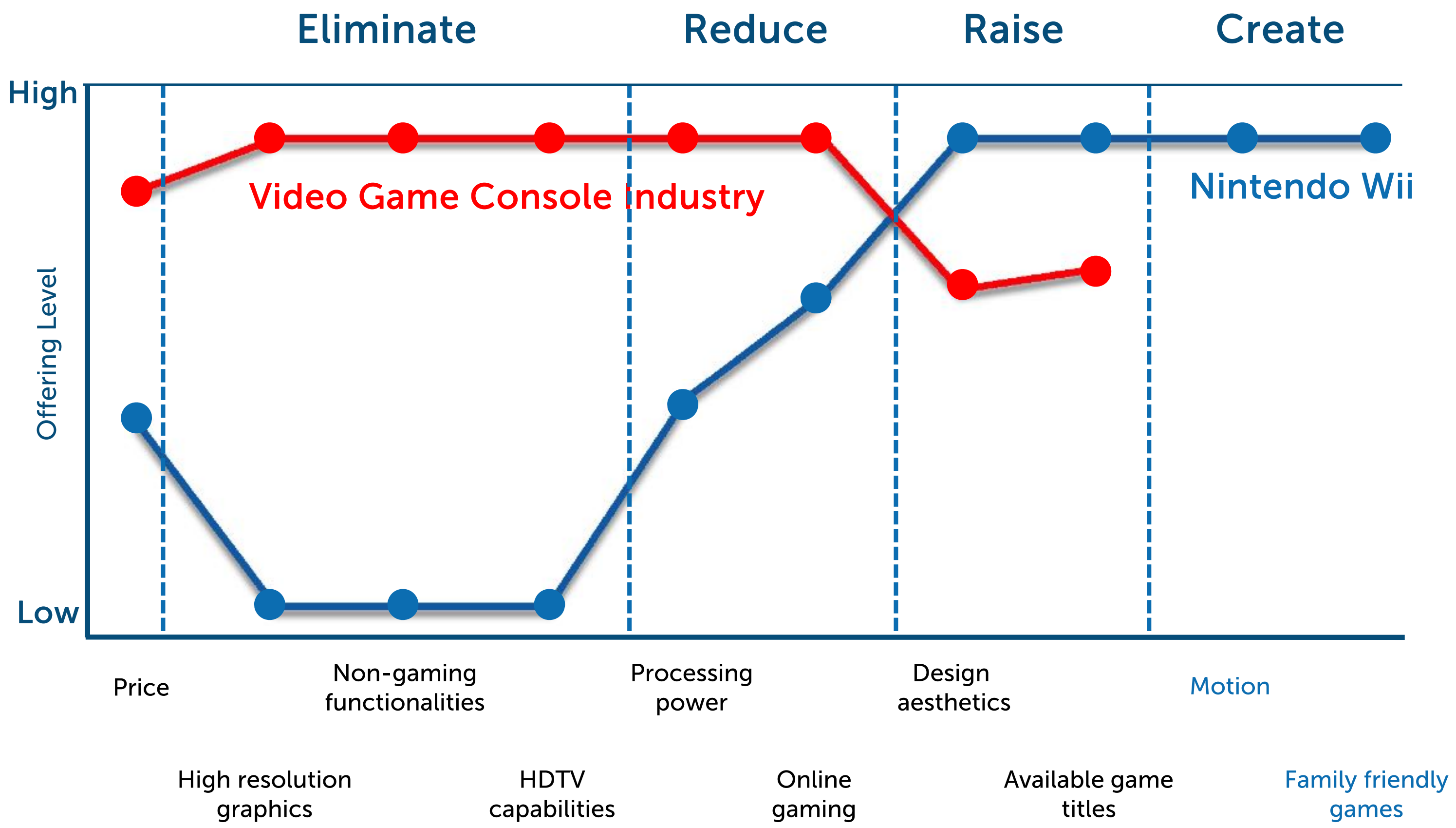
Strategy Canvas: Example Nintendo Wii (2/3)



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

Strategy Canvas: Example Nintendo Wii (3/3)

Nintendo Wii Strategy Canvas



Source: INSEAD, Blue Ocean Strategy Institute, 2013.

Redefine the Market

Concluding Remarks on Common NoSQL

- Aim to avoid join & ACID overhead
  - Joined within, correctness compromised for quick answers; believe in best effort
- Avoid the idea of a schema
- Query languages are more imperative
  - And less declarative
  - Developer better knows what’s going on; less reliance on smart optimization plans
  - More responsibility on developers
- No standard well studied languages (yet)