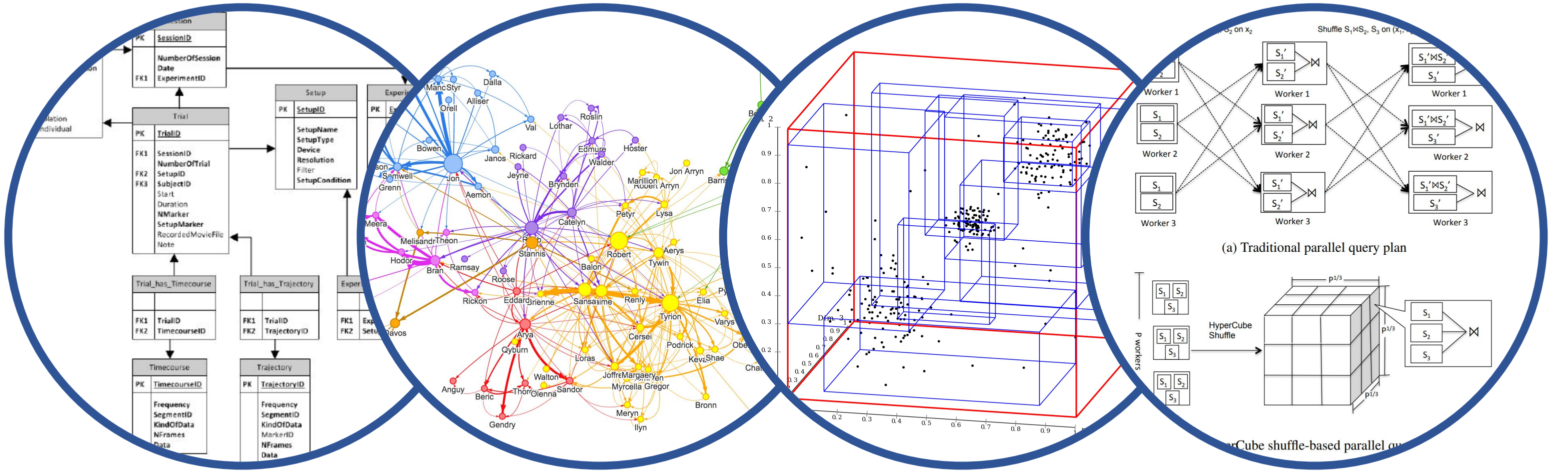


Introduction to Data Management

Key-Value vs Semi-Structured Data

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

Do sign up for AWS Educate early! Some people have had to verify their student info before receiving their accounts.

Recap: NoSQL in a Nutshell

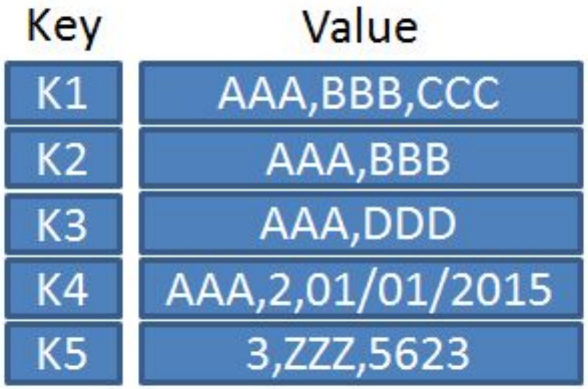
- NoSQL □ Looser data model
  - Give up built-in OLAP/analysis functionality
  - Give up built-in ACID consistency

Outline

- KV Store
  - Hash Table (Key □ Blob)
- Document Store
  - Hash Table + Parsable Documents

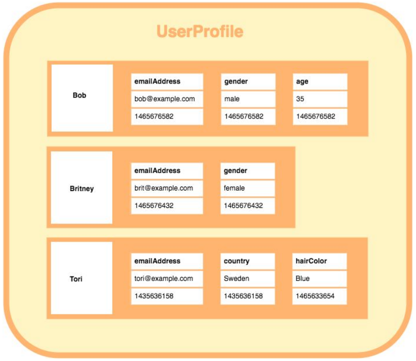
NoSQL Data Models

Key-Value Database



Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Wide-Column Store (Extensible Record Store)



UserProfile

Bob

emailAddress	gender	age
bob@example.com	male	35
1465676582	1465676582	1465676582

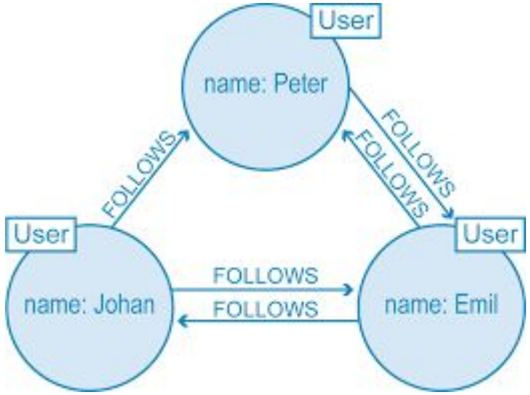
Britney

emailAddress	gender
brit@example.com	female
1465676432	1465676432

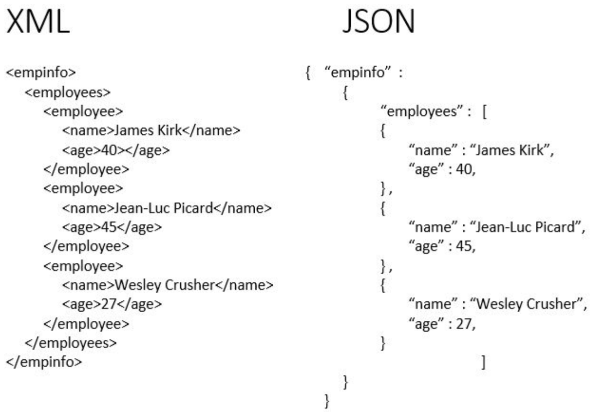
Tori

emailAddress	country	hairColor
tori@example.com	Sweden	Blue
1435636158	1435636158	1465633654

Graph Database



Document Store



XML

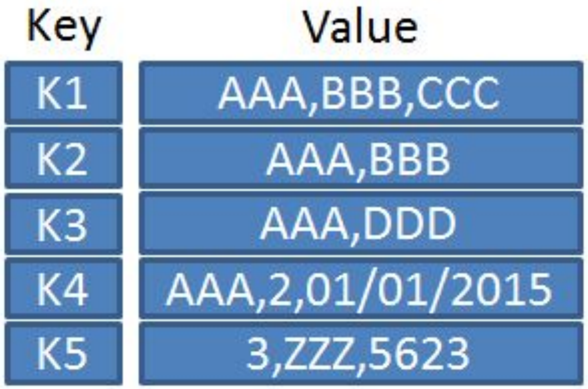
```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
      {
          “employees” :  [
          {
              “name” : “James Kirk”,
              “age” : 40,
          },
          {
              “name” : “Jean-Luc Picard”,
              “age” : 45,
          },
          {
              “name” : “Wesley Crusher”,
              “age” : 27,
          }
                        ]
      }
   }
```

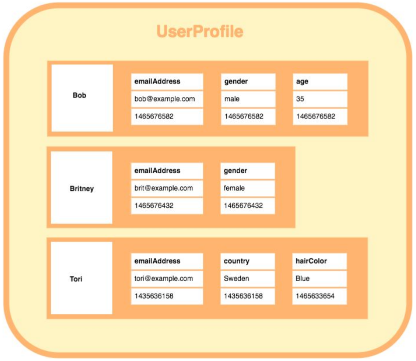
NoSQL Data Models

Key-Value Database

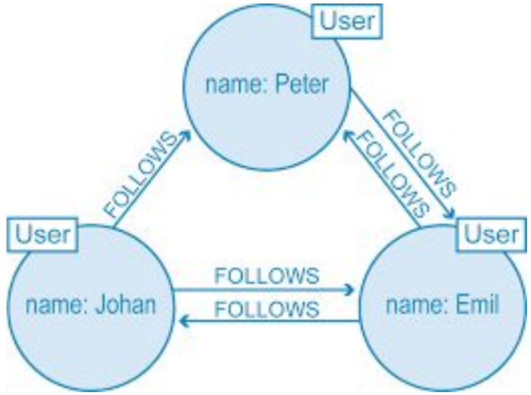


Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

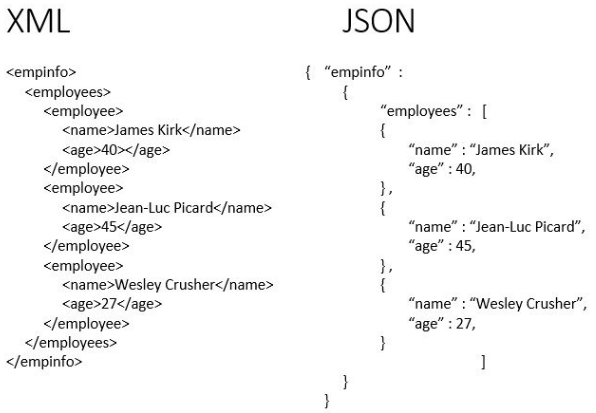
Wide-Column Store (Extensible Record Store)



Graph Database



Document Store



XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

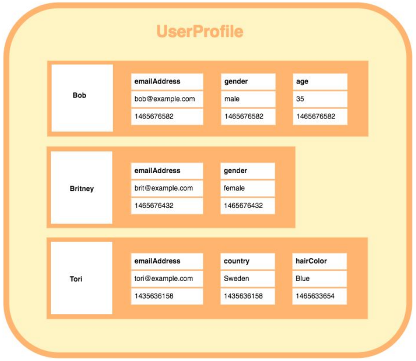
```
{  "empinfo" :
      {
          "employees" :  [
          {
              "name" : "James Kirk",
              "age" : 40,
          },
          {
              "name" : "Jean-Luc Picard",
              "age" : 45,
          },
          {
              "name" : "Wesley Crusher",
              "age" : 27,
          }
                             ]
      }
   }
```

NoSQL Data Models

Key-Value Database

- Key to value pairs
- “A hash table”

Wide-Column Store (Extensible Record Store)



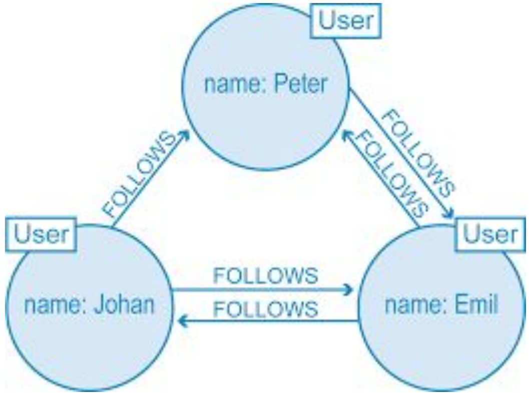
UserProfile

Bob	emailAddress	gender	age
	bob@example.com	male	35
	1465676582	1465676582	1465676582

Britney	emailAddress	gender
	brit@example.com	female
	1465676432	1465676432

Tori	emailAddress	country	hairColor
	tori@example.com	Sweden	Blue
	1435636158	1435636158	1465633654

Graph Database



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  “empinfo” :
      {
            “employees” :  [
            {
                “name” : “James Kirk”,
                “age” : 40,
            },
            {
                “name” : “Jean-Luc Picard”,
                “age” : 45,
            },
            {
                “name” : “Wesley Crusher”,
                “age” : 27,
            }
                              ]
      }
   }
```

NoSQL Data Models

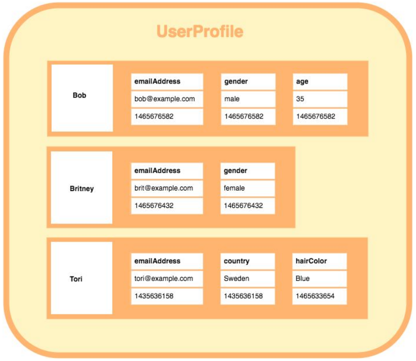
Key-Value Database

amazon DynamoDB

RocksDB

redis

Wide-Column Store (Extensible Record Store)



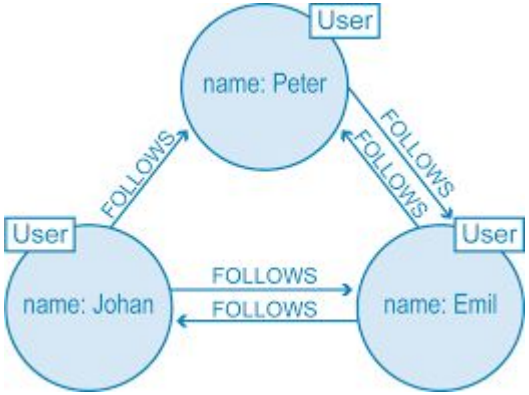
UserProfile

Bob — emailAddress: bob@example.com, 1465676582 | gender: male, 1465676582 | age: 35, 1465676582

Britney — emailAddress: brit@example.com, 1465676432 | gender: female, 1465676432

Tori — emailAddress: tori@example.com, 1435636158 | country: Sweden, 1435636158 | hairColor: Blue, 1465633654

Graph Database



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
         "employees" :  [
         {
              "name" : "James Kirk",
              "age" : 40,
         },
         {
              "name" : "Jean-Luc Picard",
              "age" : 45,
         },
         {
              "name" : "Wesley Crusher",
              "age" : 27,
         }
                        ]
      }
   }
```

NoSQL Data Models

Key-Value Database

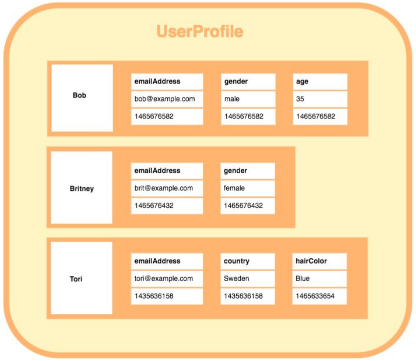
amazon DynamoDB

RocksDB

redis

Persistent KV store

Wide-Column Store (Extensible Record Store)



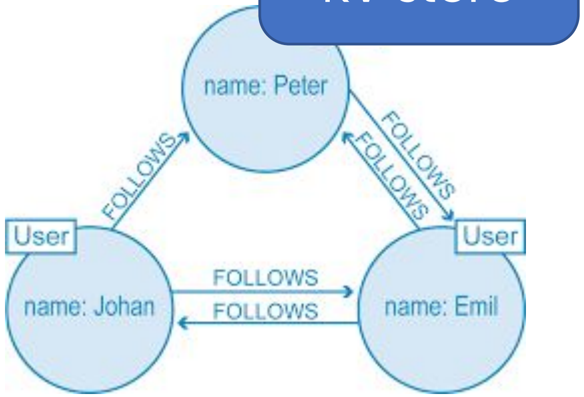
UserProfile

Bob
emailAddress | bob@example.com | 1465676582
gender | male | 1465676582
age | 35 | 1465676582

Britney
emailAddress | brit@example.com | 1465676432
gender | female | 1465676432

Tori
emailAddress | tori@example.com | 1435636158
country | Sweden | 1435636158
hairColor | Blue | 1465633654

Graph



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
        {
            "employees" :  [
            {
                "name" : "James Kirk",
                "age" : 40,
            },
            {
                "name" : "Jean-Luc Picard",
                "age" : 45,
            },
            {
                "name" : "Wesley Crusher",
                "age" : 27,
            }
                              ]
        }
   }
```

NoSQL Data Models

Key-Value Database

amazon DynamoDB

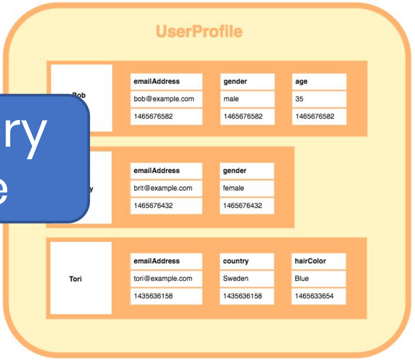
RocksDB

Persistent KV store

redis

In-memory KV store

Wide-Column Store (Extensible Record Store)



UserProfile

Bob

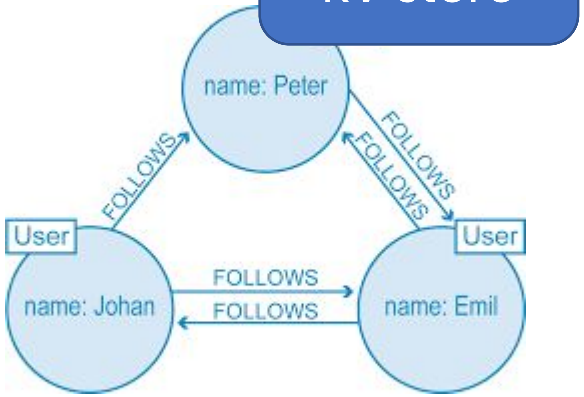
emailAddress	gender	age
bob@example.com	male	35
1465676582	1465676582	1465676582

emailAddress	gender
brit@example.com	female
1465676432	1465676432

Tori

emailAddress	country	hairColor
tori@example.com	Sweden	Blue
1435636158	1435636158	1465633654

Graph



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
         "employees" :  [
         {
            "name" : "James Kirk",
            "age" : 40,
         },
         {
            "name" : "Jean-Luc Picard",
            "age" : 45,
         },
         {
            "name" : "Wesley Crusher",
            "age" : 27,
         }
                              ]
      }
   }
```

NoSQL Data Models

Key-Value Database

amazon DynamoDB

Extended to also be a Document Store

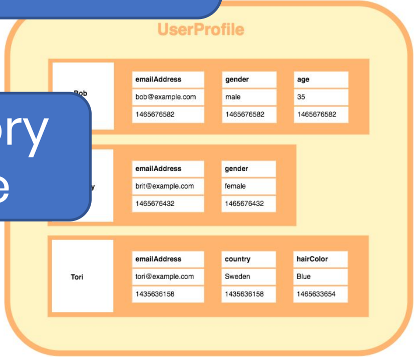
RocksDB

Persistent KV store

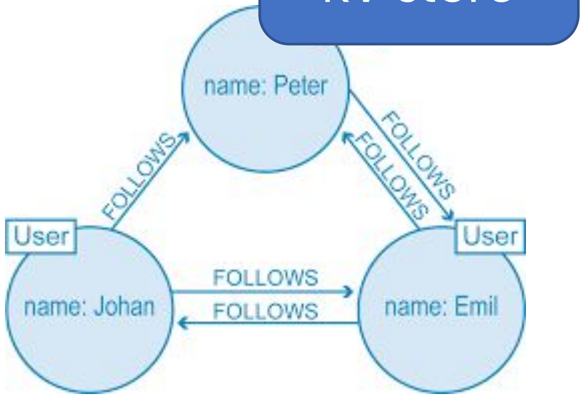
redis

In-memory KV store

mn Store cord Store)



Graph



Document Store

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
         "employees" :  [
         {
            "name" : "James Kirk",
            "age" : 40,
         },
         {
            "name" : "Jean-Luc Picard",
            "age" : 45,
         },
         {
            "name" : "Wesley Crusher",
            "age" : 27,
         }
                        ]
      }
   }
```

Key-Value Store

- Data model:
  - (key, value) pairs
  - Key □ string/integer/…, unique for the entire data
  - Value □ anything

Key-Value Store

- Data model:
  - (key, value) pairs
  - Key □ string/integer/…, unique for the entire data
  - Value □ anything
- Basic Operations:
  - get(key)
  - put(key, value)

Key-Value Store

- Data model:
  - (key, value) pairs
  - Key □ string/integer/…, unique for the entire data
  - Value □ anything
- Basic Operations:
  - get(key)
  - put(key, value)
- Distribution/Partitioning:
  - Access via hash function
  - No replication: Key k stored at server $h(k)\%N$
  - 3-way replication: Key k stored at servers $h_1(k)\%N$, $h_2(k)\%N$, $h_3(k)\%N$

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value
FID	Single flight record

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

Key	Value
FID	Single flight record
Date	All flight records on that day

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

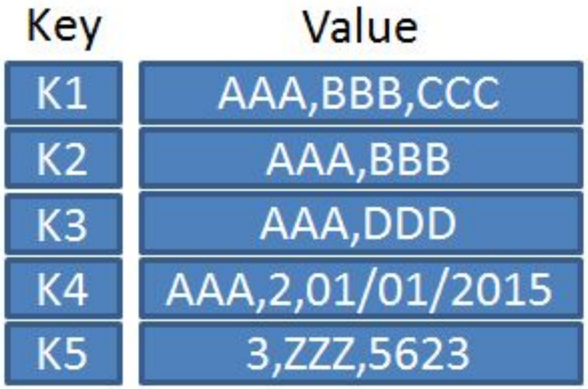
Key	**Value**
FID	Single flight record
Date	All flight records on that day
(origin, destination)	All flight records between the cities

DynamoDB API

- Create, Read, Update, Delete (CRUD) actions
  - Create □ **PutItem**
  - Read □ **GetItem**
  - Update □ UpdateItem (Document store functionality)
  - Delete □ DeleteItem
- Read consistency
  - Eventually consistent (default, may be stale data)
  - Strongly consistent (gets most recent written data)
- As of December 2018, ACID is "supported"
  - **TransactWriteItems**
  - **TransactGetItems**

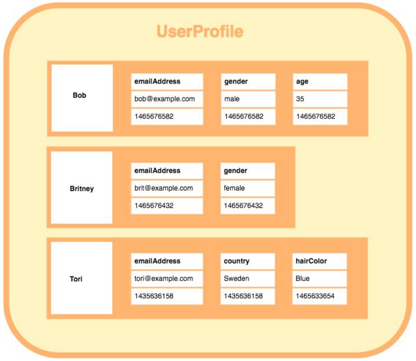
NoSQL Data Models

Key-Value Database



Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Wide-Column Store (Extensible Record Store)



UserProfile

Bob

emailAddress	gender	age
bob@example.com	male	35
1465676582	1465676582	1465676582

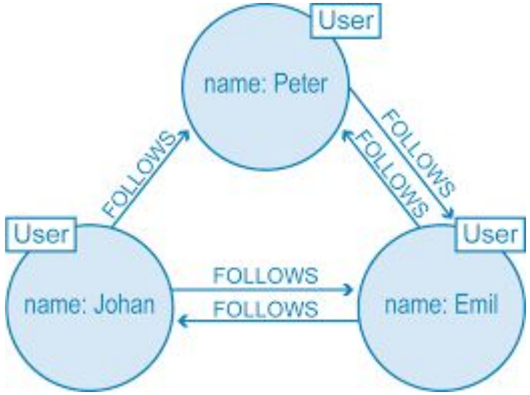
Britney

emailAddress	gender
brit@example.com	female
1465676432	1465676432

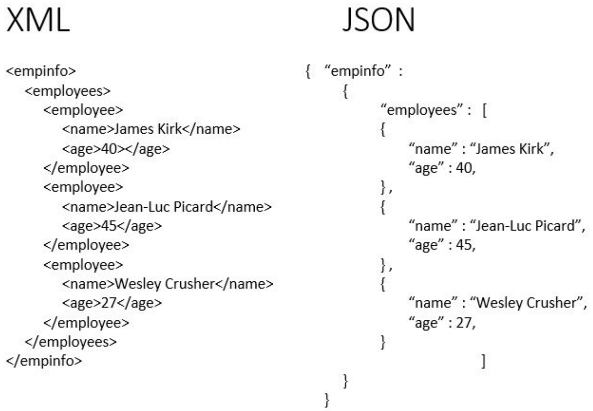
Tori

emailAddress	country	hairColor
tori@example.com	Sweden	Blue
1435636158	1435636158	1465633654

Graph Database



Document Store



XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

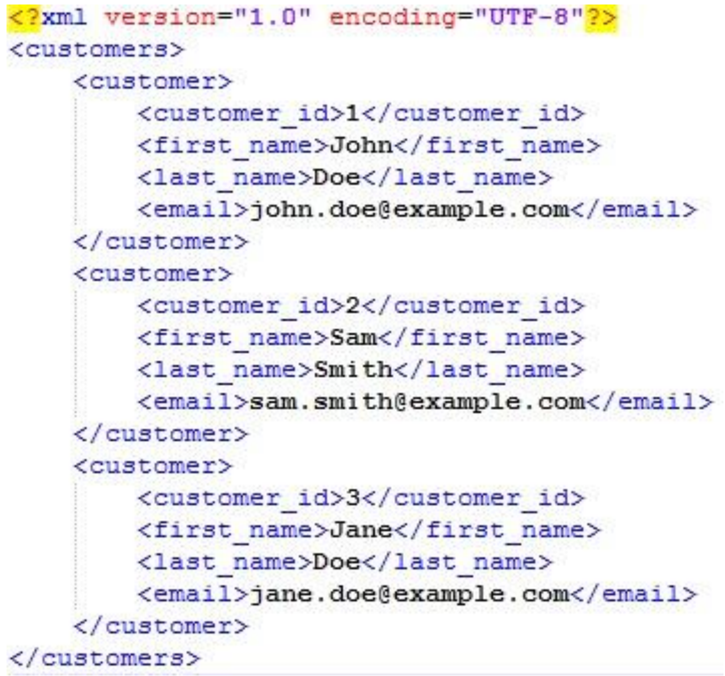
```
{  "empinfo" :
      {
            "employees" :  [
            {
                 "name" : "James Kirk",
                 "age" : 40,
            },
            {
                 "name" : "Jean-Luc Picard",
                 "age" : 45,
            },
            {
                 "name" : "Wesley Crusher",
                 "age" : 27,
            }
                                  ]
      }
   }
```

What is a "document" anyways?

- Loose terminology
- Any "parsable" file qualifies
  - Ex: MongoDB can handle CSV files

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - **XML**
  - Protobuf
  - JSON

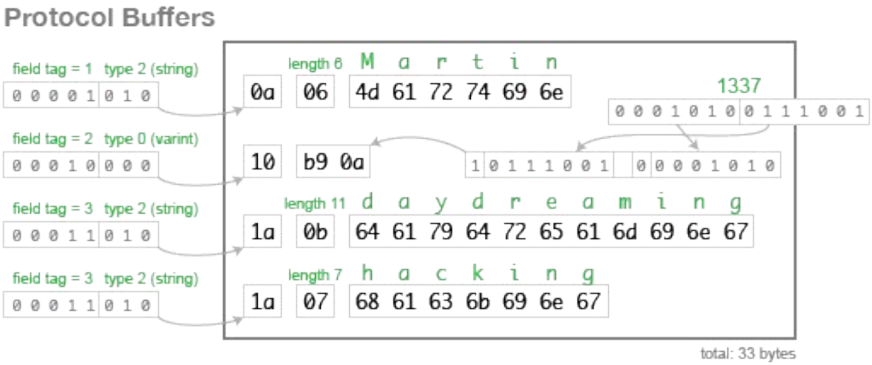


```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer>
        <customer_id>1</customer_id>
        <first_name>John</first_name>
        <last_name>Doe</last_name>
        <email>john.doe@example.com</email>
    </customer>
    <customer>
        <customer_id>2</customer_id>
        <first_name>Sam</first_name>
        <last_name>Smith</last_name>
        <email>sam.smith@example.com</email>
    </customer>
    <customer>
        <customer_id>3</customer_id>
        <first_name>Jane</first_name>
        <last_name>Doe</last_name>
        <email>jane.doe@example.com</email>
    </customer>
</customers>
```

Tags surround the respective data

Semi-Structured Documents

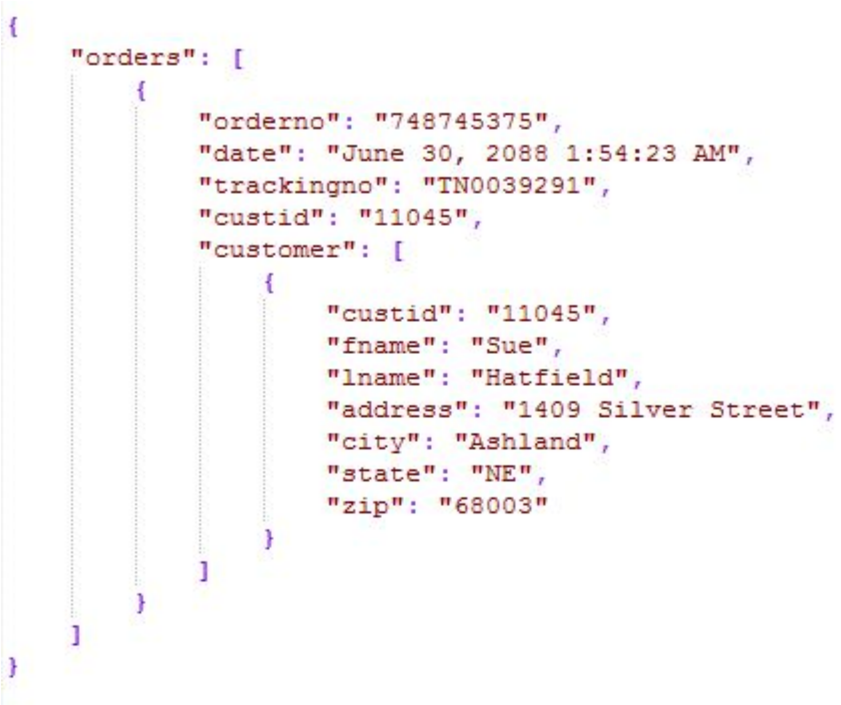
- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - **Protobuf**
  - JSON



Not human readable in serialized format

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - **JSON**



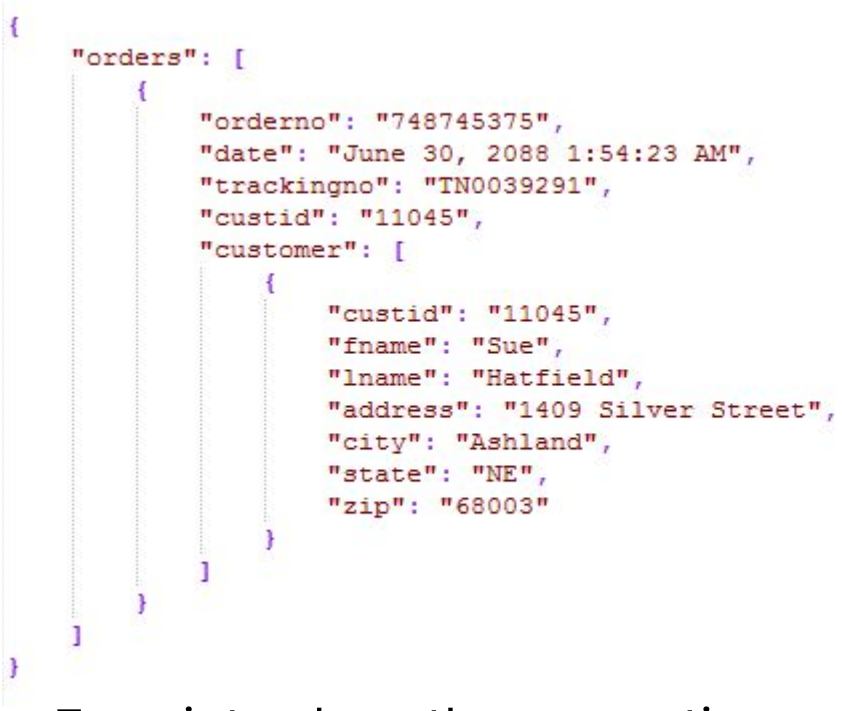
```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Tags introduce the respective data

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - **JSON**

Many applications have phased out XML in favor of JSON



```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Tags introduce the respective data

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

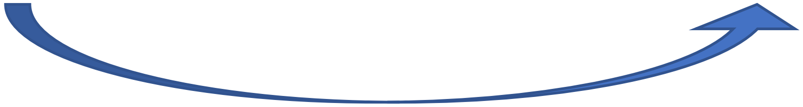
- Semi-Structured
  - Self-described schema
  - Tree-structured data

Less well-defined/More flexible

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data



Less well-defined/More flexible

- Basic retrieval process:
  1. Retrieve table
  2. Run through rows
  3. Return data

- Basic retrieval process:
  1. Retrieve document
  2. Parse document tree
  3. Return data

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data

Less well-defined/More flexible

- Basic retrieval process:
  1. Retrieve table
  2. Run through rows
  3. Return data

- Basic retrieval process:
  1. Retrieve document
  2. Parse document tree
  3. Return data

Inefficient encoding/Easy exchange of data

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

Types

Primitives include:

- String (in quotes)
- Numeric (unquoted number)
- Boolean (unquoted true/false)
- Null (literally just null)

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

Types

Objects are an *unordered* collection of name-value pairs:

- "name": <value>
- Values can be primitives, objects, or arrays
- Enclosed by { }

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

Types

Objects are an *unordered* collection of name-value pairs:

- "name": <value>
- Values can be primitives, objects, or arrays
- Enclosed by { }

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

Types

Arrays are an *ordered* list of values:

- Order is preserved in interpretation
- May contain any mix of types
- Enclosed by [ ]

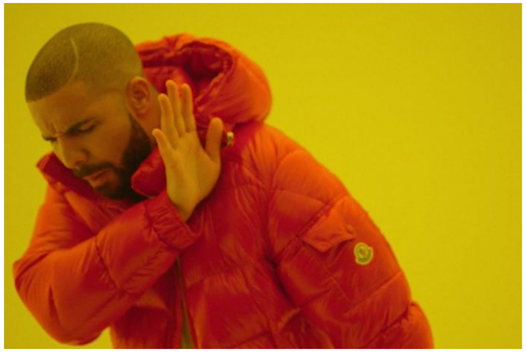
JSON Standard – Rules of the Game

- JSON Standard too expressive
  - Implementations **restrict syntax**
  - Ex: Duplicate fields

```
{
   "id": "01",
   "language": "Java",
   "author": "H. Javeson",
   "author": "D. Suciu",
   "author": "A. Cheung",
   "year": 2015
}
```

JSON Standard – Rules of the Game

- JSON Standard too expressive
  - Implementations **restrict syntax**
  - Ex: Duplicate fields



```
{
   "id": "01",
   "language": "Java",
   "author": "H. Javeson",
   "author": "D. Suciu",
   "author": "A. Cheung",
   "year": 2015
}
```

```
{
   "id": "01",
   "language": "Java",
   "author": ["H. Javeson",
              "D. Suciu",
              "A. Cheung"],
   "year": 2015
}
```

Thinking About Semi-Structured Data

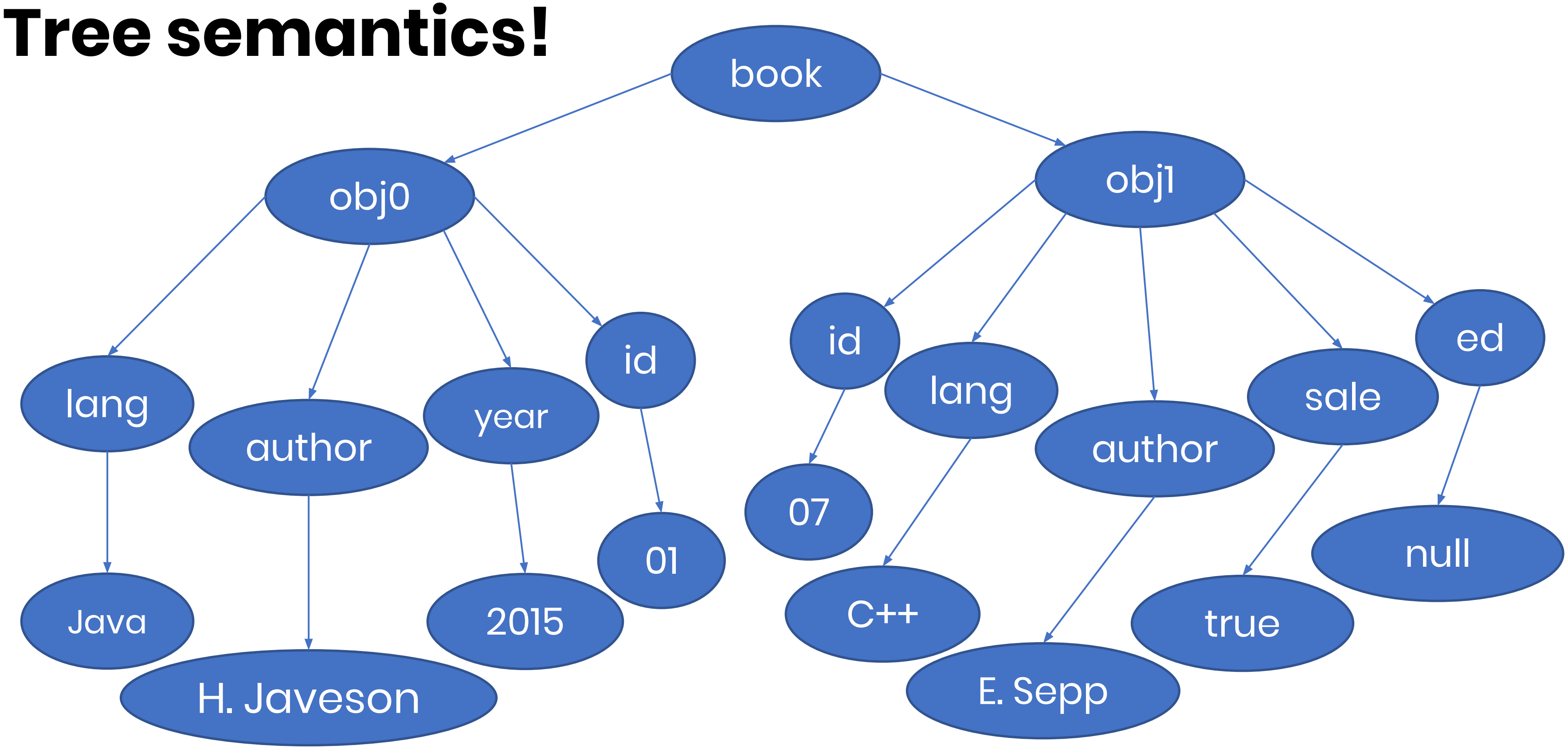
What does semi-structured data structure encode?

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": null,
         "sale": true
      }
   ]
}
```

Thinking About Semi-Structured Data

What does semi-structured data structure encode?

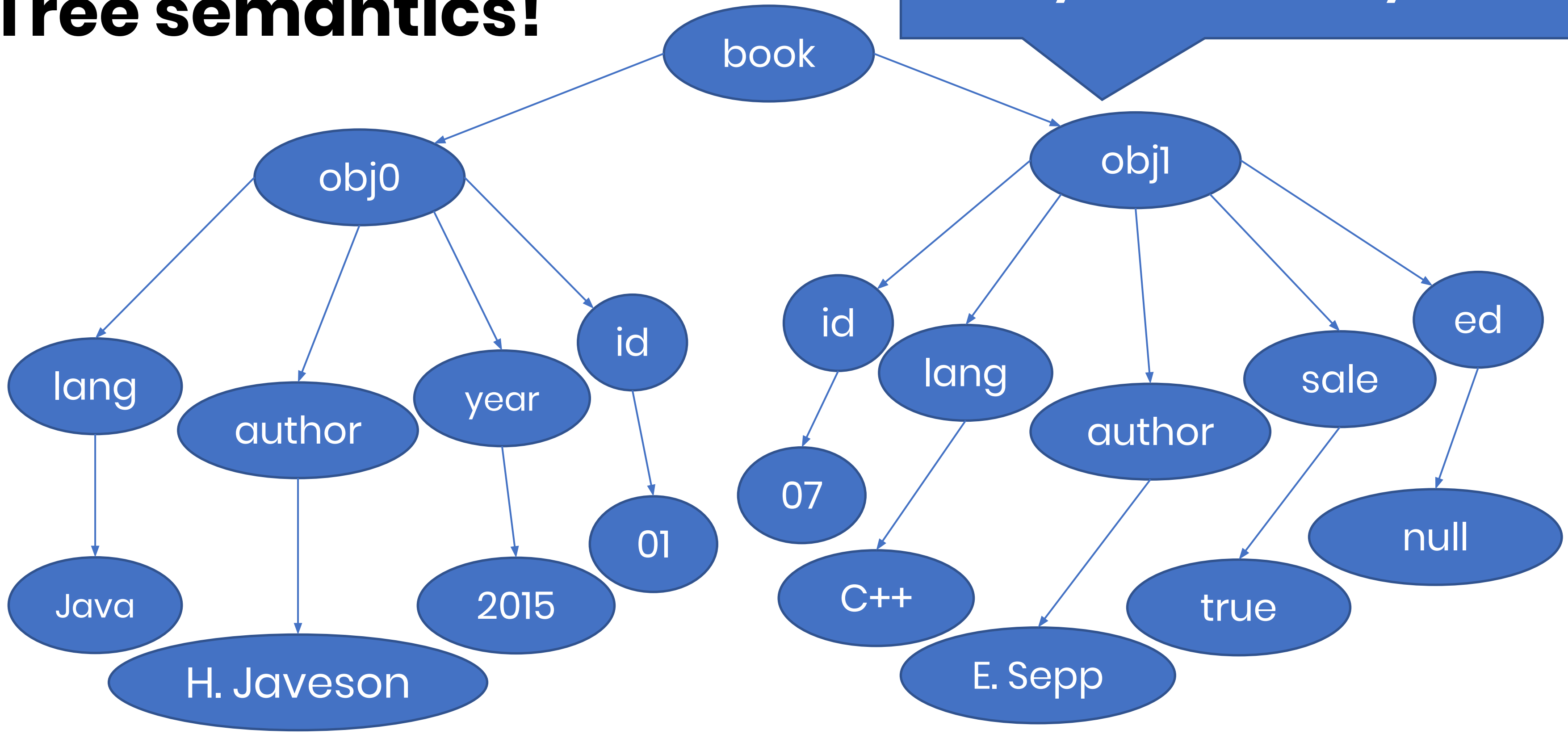
Tree semantics!



Thinking About Semi-Structured Data

What does semi-structured data structure encode?

Tree semantics!



These object don't have labels, as they are in an array

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in
semi-structured land?

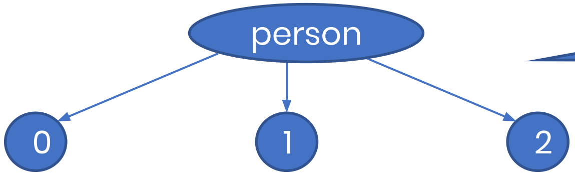
person

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in semi-structured land?



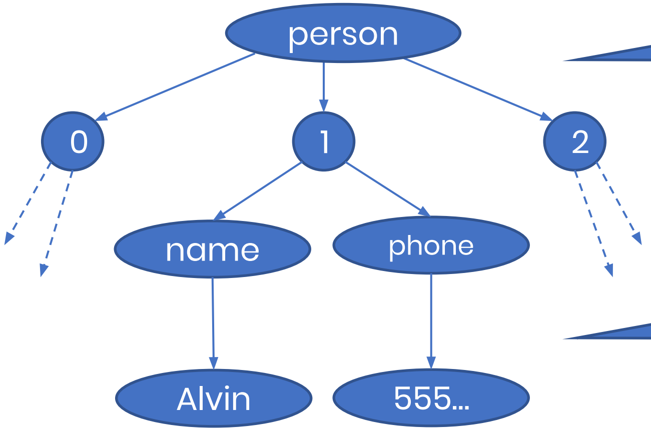
Tables are just an array of elements (rows)

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in semi-structured land?



Tables are just an array of elements (rows)

Rows are just simple (unnested) objects

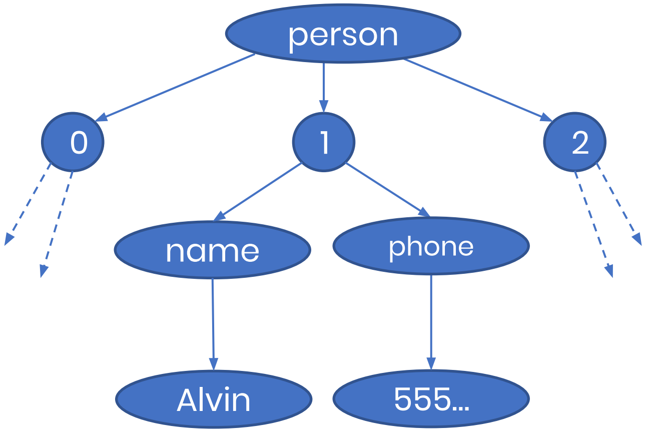
From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

What is a table in semi-structured land?



From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

How can NULL be represented?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL be represented?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL be represented?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": null
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL be represented?

```
{
    "person":[
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda"
        }
    ]
}
```

OK for field to be missing!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can’t represent?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can't represent?

Non-flat data!

- Array data
- Multi-part data
- Heterogeneous collections

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	???
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can't represent?

Non-flat data!

- **Array data**
- Multi-part data
- Heterogeneous collections

```
{
   "person":[
      {
         "name": "Dan",
         "phone": [
            "555-123-4567",
            "555-987-6543"
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
???	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

```
{
   "person":[
      {
         "name": {
             "fname": "Dan",
             "lname": "Suciu"
          },
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      }
   ]
}
```

Are there things that the Relational Model can't represent?

Non-flat data!

- Array data
- **Multi-part data**
- Heterogeneous collections

From Relational to Semi-Structured

Person

Name	Phone
???	555-123-4567
Alvin	555-234-5678
Magda	???

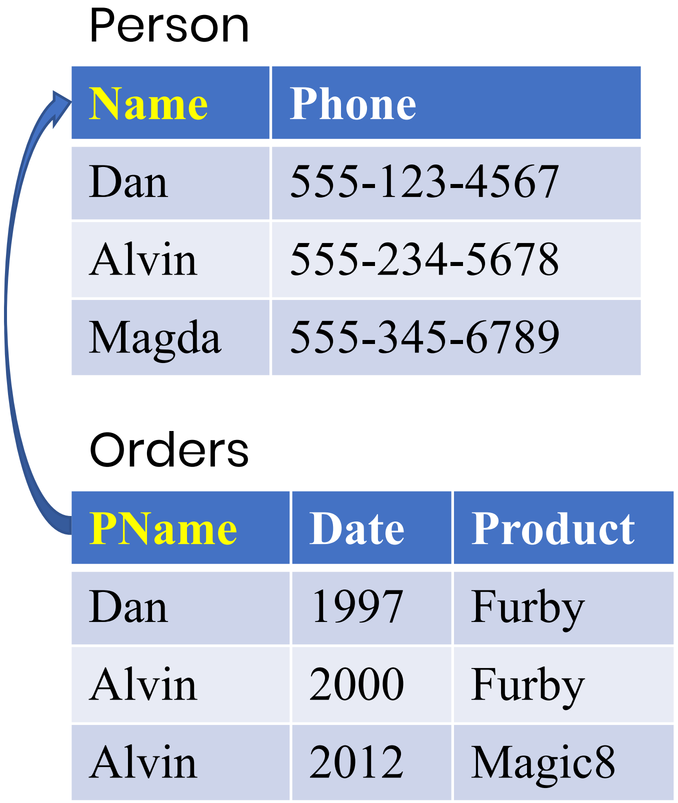
Are there things that the Relational Model can’t represent?

Non-flat data!

- Array data
- Multi-part data
- **Heterogeneous collections**

```
{
   "person":[
      {
         "name": {
             "fname": "Dan",
             "lname": "Suciu"
          },
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda"
      }
   ]
}
```

From Relational to Semi-Structured



Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

How do we represent foreign keys?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567",
         "orders": [
            {
               "date": 1997,
               "product": "Furby"
            }
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678",
         "orders": [
            {
               "date": 2000,
               "product": "Furby"
            },
            {
               "date": 2012,
               "product": "Magic8"
            }
         ]
      },
      {
         "name": "Magda",
         "phone": "555-345-6789",
         "orders": []
      }
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567",
         "orders": [
            {
               "date": 1997,
               "product": "Furby"
            }
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678",
         "orders": [
            {
               "date": 2000,
               "product": "Furby"
            },
            {
               "date": 2012,
               "product": "Magic8"
            }
         ]
      },
      {
         "name": "Magda",
         "phone": "555-345-6789",
         "orders": []
      }
   ]
}
```

Precomputed equijoin!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this
many-to-many
relationship easily
convertible to JSON?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this
many-to-many
relationship easily
convertible to JSON?

Nest the data?
Person □ Orders □ Product

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this
many-to-many
relationship easily
convertible to JSON?

Nest the data?
Person □ Orders □ Product

We might miss some
products!
&
Product data will be
duplicated!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this
many-to-many
relationship easily
convertible to JSON?

Nest the data?
Product □ Orders □ Person

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this
many-to-many
relationship easily
convertible to JSON?

Nest the data?
Product □ Orders □ Person

We might miss some people!
&
People data will be
duplicated!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Convert each table to a separate array/document?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Convert each table to a separate array/document?

We wanted to avoid joining in the first place!

From Relational to Semi-Structured

Big ideas:

- Semi-structured data is **parsed**
  - Data model flexibility
  - Potentially lots of redundancy
- Semi-structured data expresses **unique patterns**
  - Collection/multi-part data
  - Precompute joins
- Semi-structured data **has limits**
  - Relies on relational-like patterns in some situations

Next time

- AsterixDB as a case study of Document Store
  - Introducing AsterixDB and SQL++