NOSQL

EGCO321 DATABASE SYSTEMS

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

WHAT IS NOSQL?

- Stands for No-SQL or Not Only SQL.
- Class of non-relational data storage systems
  - E.g. MongoDB, Neo4j, …
- Usually do not require a fixed table schema nor do they use the concept of joins
  - Distributed data storage systems
- All NoSQL offerings relax one or more of the ACID properties.

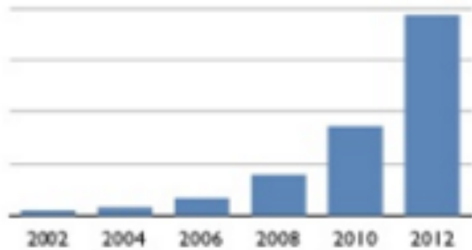
WHY NOSQL? (1)

- Relational database were not built for distributed applications. Because …
  - Joins are expensive
  - Hard to scale horizontally
  - Impedance mismatch occurs
  - Expensive (Product Cost, Hardware, Maintenance, etc.)
- It's weak in
  - Speed (Performance)
  - High availability
  - Partition tolerance

WHY NOSQL? (2)

New Trends

Clip slide



Big data

Connectivity

P2P Knowledge

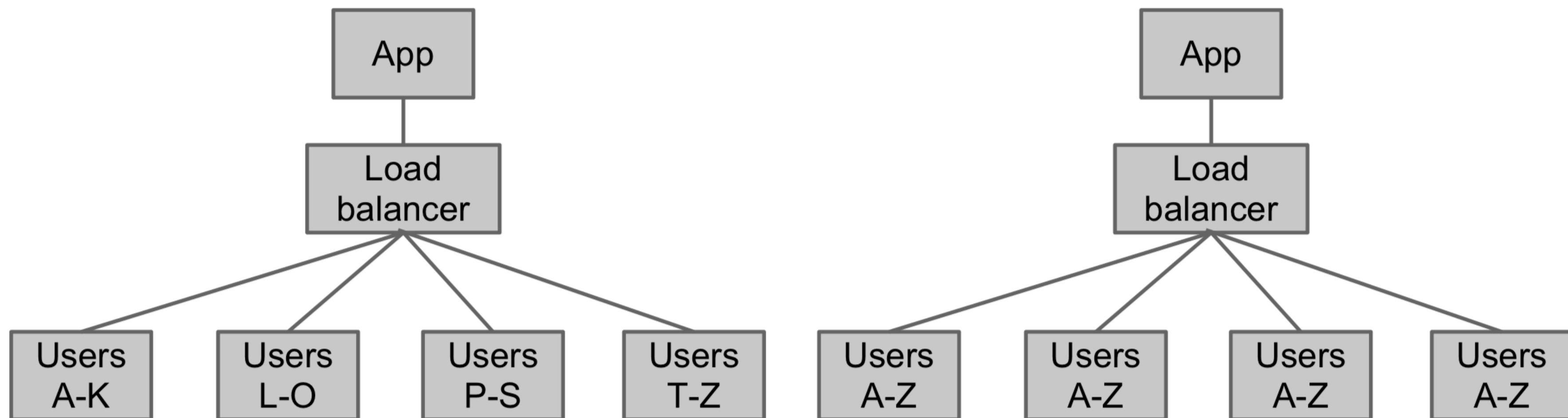
Concurrency

Diversity

Cloud-Grid

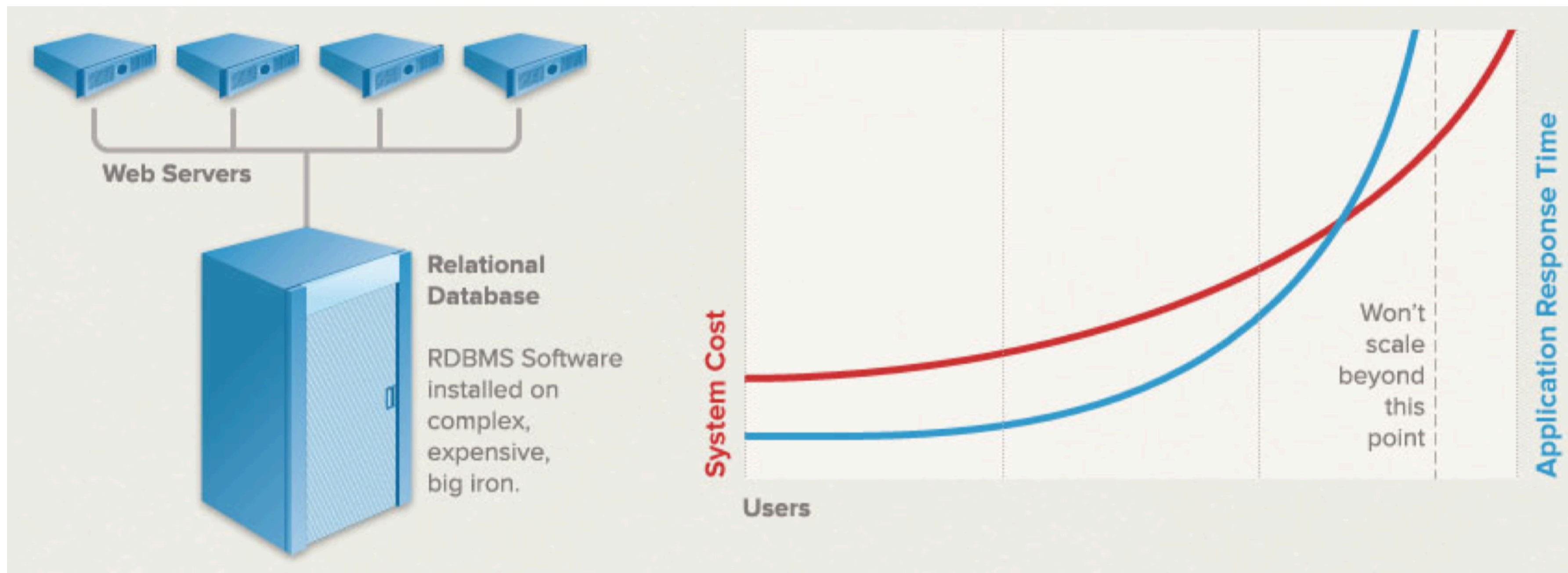
DATABASE SCALING

- Relational Database is "**scaled up**" by adding hardware processing power
- NoSQL is "**scaled out**" by spreading the load



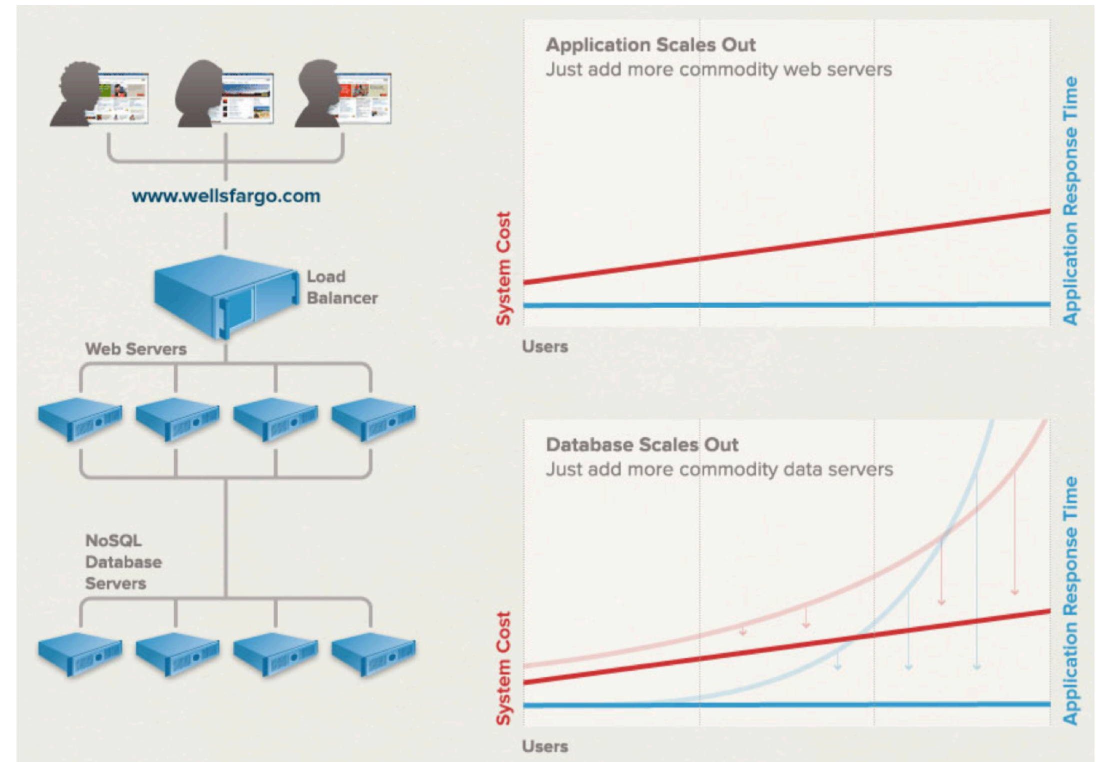
RELATIONAL DATABASE SCALING

- At certain point relational database won't scale



NOSQL DATABASE SCALING

- Scaling horizontally is possible with NoSQL
- Scaling up / down is easy
    - Supports rapid production-ready prototyping
- Better handling of traffic spikes



WHERE NOSQL IS USED?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)

CAP THEOREM

- The CAP theorem applies a similar type of logic to distributed systems — namely, that a distributed system can deliver only two of three desired characteristics: consistency, availability, and partition tolerance.

CAP THEOREM - CONSISTENCY

- Consistency means that all clients see the same data at the same time, no matter which node they connect to.
- For this to happen, whenever data is written to one node, it must be instantly forwarded or replicated to all the other nodes in the system before the write is deemed 'successful.'

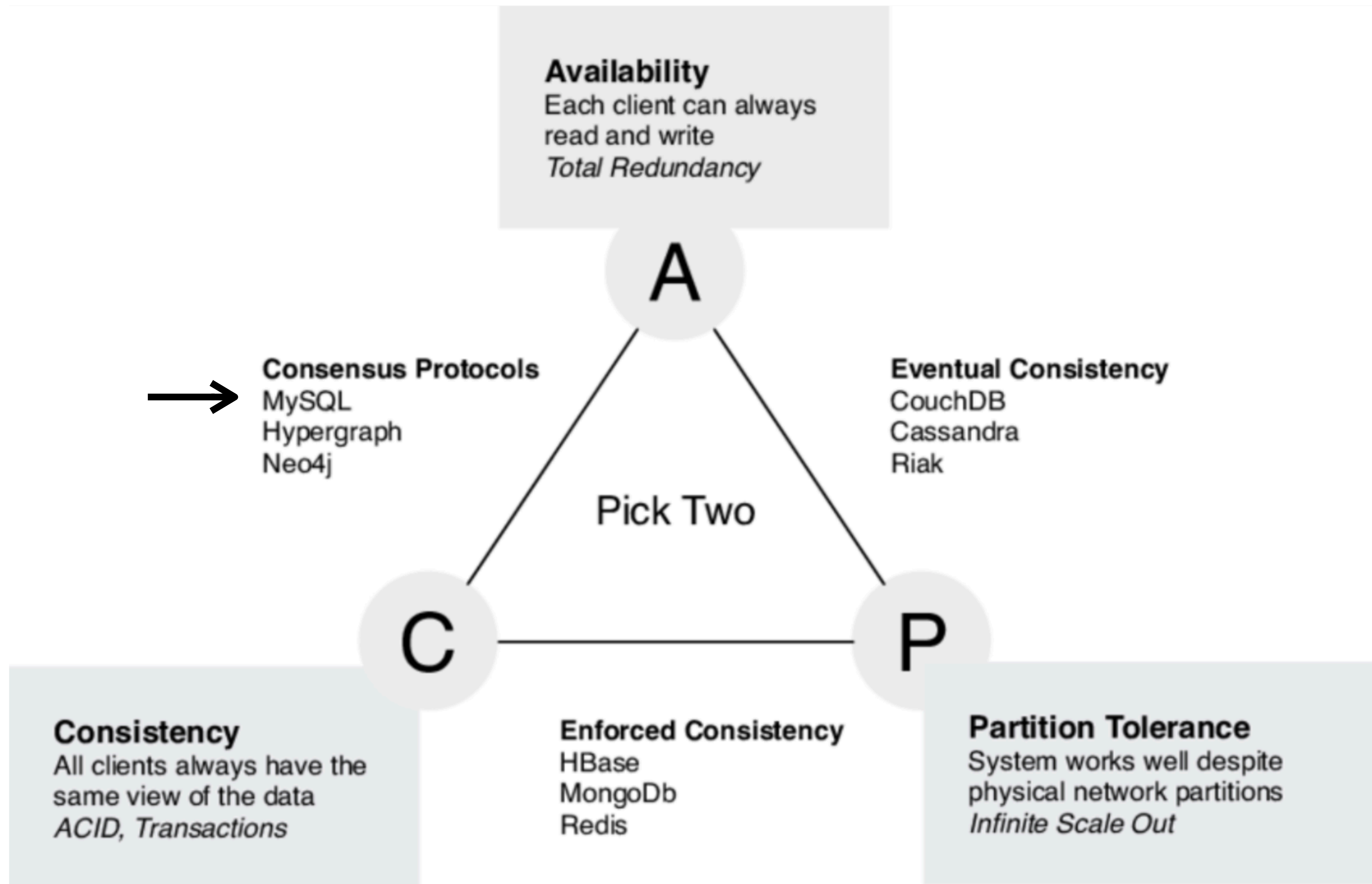
CAP THEOREM - AVAILABILITY

- Availability means that any client making a request for data gets a response, even if one or more nodes are down.
- Another way to state this — all working nodes in the distributed system return a valid response for any request, without exception.

CAP THEOREM - PARTITION TOLERANCE

- A partition is a communications break within a distributed system — a lost or temporarily delayed connection between two nodes.
- Partition tolerance means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

NOSQL AND CAP (1)



NOSQL AND CAP (2)

- In other words, CAP can be expressed as "If the network is broken, your database won't work"
- In Relational DBMS we do not have P (network partitions)
  - Consistency and Availability are achieved
- In NoSQL we want to have P
  - Need to select either C or A
  - Drop A -> Accept waiting until data is consistent
  - Drop C -> Accept getting inconsistent data sometimes

ACID VS BASE

- Scalability and better performance of NoSQL is achieved by sacrificing ACID compatibility.
  - Atomic, Consistent, Isolated, Durable
- NoSQL is having BASE compatibility instead.
  - Basically Available, Soft state, Eventual consistency

BASE — BASICALLY AVAILABLE

- Use replication and sharding to reduce the likelihood of data unavailability and use sharding, or partitioning the data among many different storage servers, to make any remaining failures partial.
- The result is a system that is always available, even if subsets of the data become unavailable for short periods of time.

BASE AND AVAILABILITY

- The availability of BASE is achieved through supporting partial failures without total system failure.
- Example. If users are partitioned across five database servers, BASE design encourages crafting operations in such a way that a user database failure impacts only the 20 percent of the users on that particular host.
  - This leads to higher perceived availability of the system. Even though a single node is failing, the interface is still operational.

BASE — SOFT STATE

- While ACID systems assume that data consistency is a hard requirement, NoSQL systems allow data to be inconsistent and relegate designing around such inconsistencies to application developers.
- In other words, soft state indicates that the state of the system may change over time, even without input.
  - This is because of the eventual consistency model (the acronym is a bit contrived).

BASE — EVENTUALLY CONSISTENT

- Although applications must deal with instantaneous consistency, NoSQL systems ensure that at some future point in time the data assumes a consistent state.
- In contrast to ACID systems that enforce consistency at transaction commit, NoSQL guarantees consistency only at some undefined future time.
  - Where ACID is pessimistic and forces consistency at the end of every operation, BASE is optimistic and accepts that the database consistency will be in a state of flux.

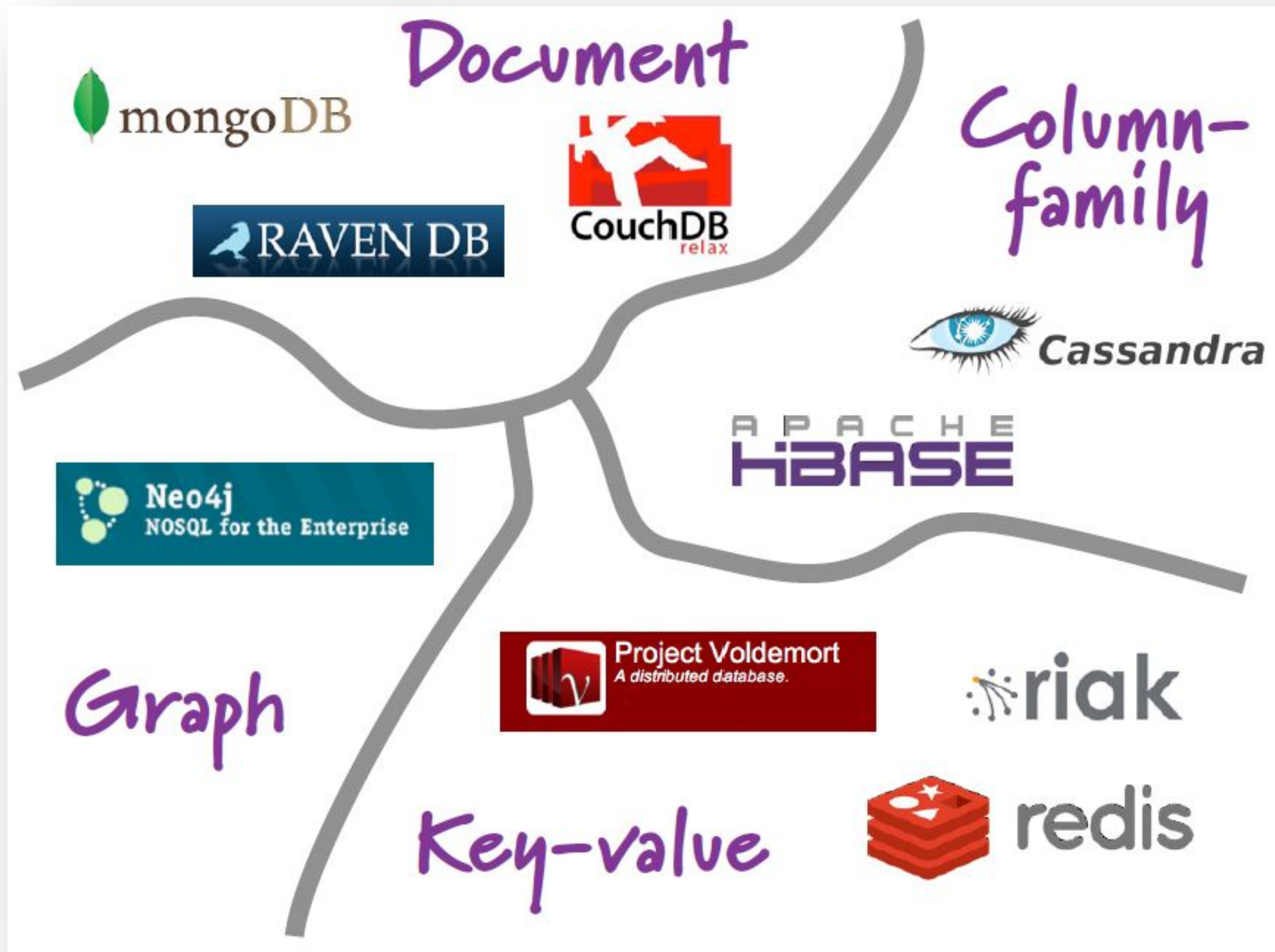
BASE AND CONSISTENCY

- As DB nodes are added while scaling up, need for synchronization arises
- If absolute consistency is required, nodes need to communicate when read/write operations are performed on a node
  - Consistency over availability -> bottleneck
- As a trade-off, "eventual consistency" is used
- Consistency is maintained later
  - Numerous approaches for keeping up "distributed consistency" are available
    - Amazon Dynamo - consistent hashing
    - CouchDB - asynchronous master-master replication
    - MongoDB - auto-sharding+replication cluster with a master server

SOME BREEDS OF NOSQL SOLUTIONS

- Key-Value Stores
- Column Family Stores
- Document Databases
- Graph Databases
- In addition: Object and RDF databases as well as Tuple stores

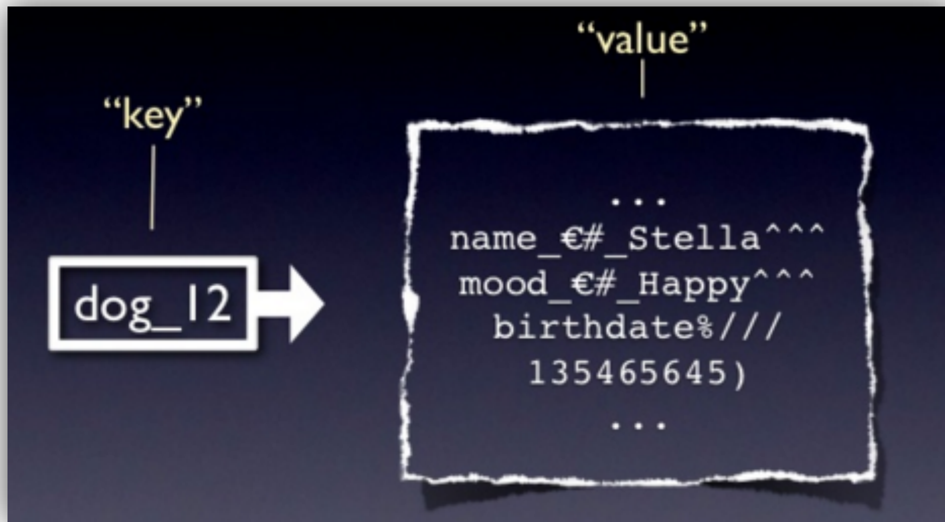
TYPE OF NOSQL SOLUTIONS



KEY-VALUE STORES

- In NoSQL databases, a Key-Value Store is the simplest and most fundamental type of NoSQL database. It stores data as a collection of key-value pairs, where:
  - A key is a unique identifier (like an ID or a name).
  - A value is the data associated with that key — it can be a string, number, JSON object, or even a binary file.
- This structure is similar to a dictionary, hash table, or map in programming.

EXAMPLE OF KEY-VALUE STORES



Car	
Key	**Attributes**
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Example: Redis Commands

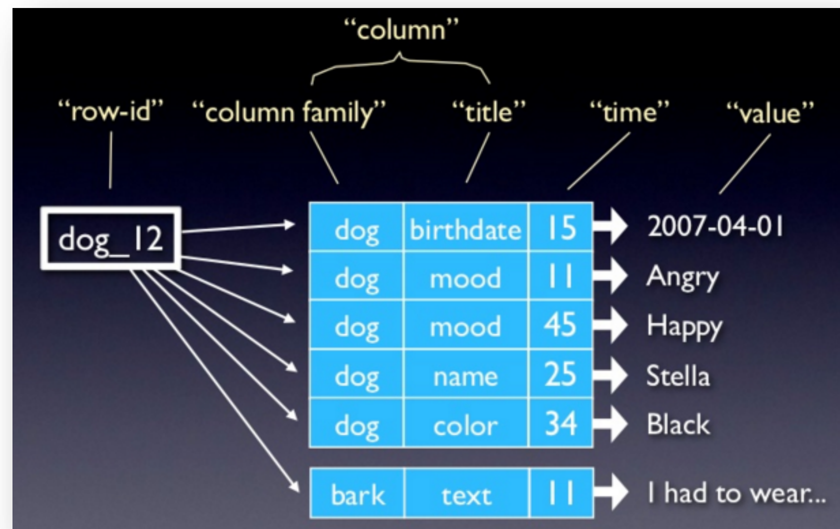
```
SET product:12345 '{"name": "Coffee Mug", "price": 299, "stock": 120}'
GET product:12345
```

COLUMN FAMILY STORES (1)

- In NoSQL databases, a Column Family Store (also called a Wide-Column Store) organizes data into rows and columns, similar to a relational database — but much more flexible and scalable.
- Instead of having a fixed schema like in SQL, each row can have a different number of columns, and columns are grouped into column families for better data organization and performance.

COLUMN FAMILY STORES (2)

- The column is lowest/ smallest instance of data.
- It is a tuple that contains a name, a value and a timestamp



<table>
<tr><td colspan="3">ColumnFamily: Authors</td></tr>
<tr><td>Key</td><td colspan="2">Value</td></tr>
<tr><td rowspan="5">"Eric Long"</td><td colspan="2">Columns</td></tr>
<tr><td>Name</td><td>Value</td></tr>
<tr><td>"email"</td><td>"eric (at) long.com"</td></tr>
<tr><td>"country"</td><td>"United Kingdom"</td></tr>
<tr><td>"registeredSince"</td><td>"01/01/2002"</td></tr>
<tr><td rowspan="5">"John Steward"</td><td colspan="2">Columns</td></tr>
<tr><td>Name</td><td>Value</td></tr>
<tr><td>"email"</td><td>"john.steward (at) somedomain.com"</td></tr>
<tr><td>"country"</td><td>"Australia"</td></tr>
<tr><td>"registeredSince"</td><td>"01/01/2009"</td></tr>
<tr><td rowspan="5">"Ronald Mathies"</td><td colspan="2">Columns</td></tr>
<tr><td>Name</td><td>Value</td></tr>
<tr><td>"email"</td><td>"ronald (at) sodeso.nl"</td></tr>
<tr><td>"country"</td><td>"Netherlands, The"</td></tr>
<tr><td>"registeredSince"</td><td>"01/01/2010"</td></tr>
</table>

Example: Cassandra, Hadoop

CASE STUDY: FACEBOOK

- Some statistics about Facebook Search (using Cassandra)

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms

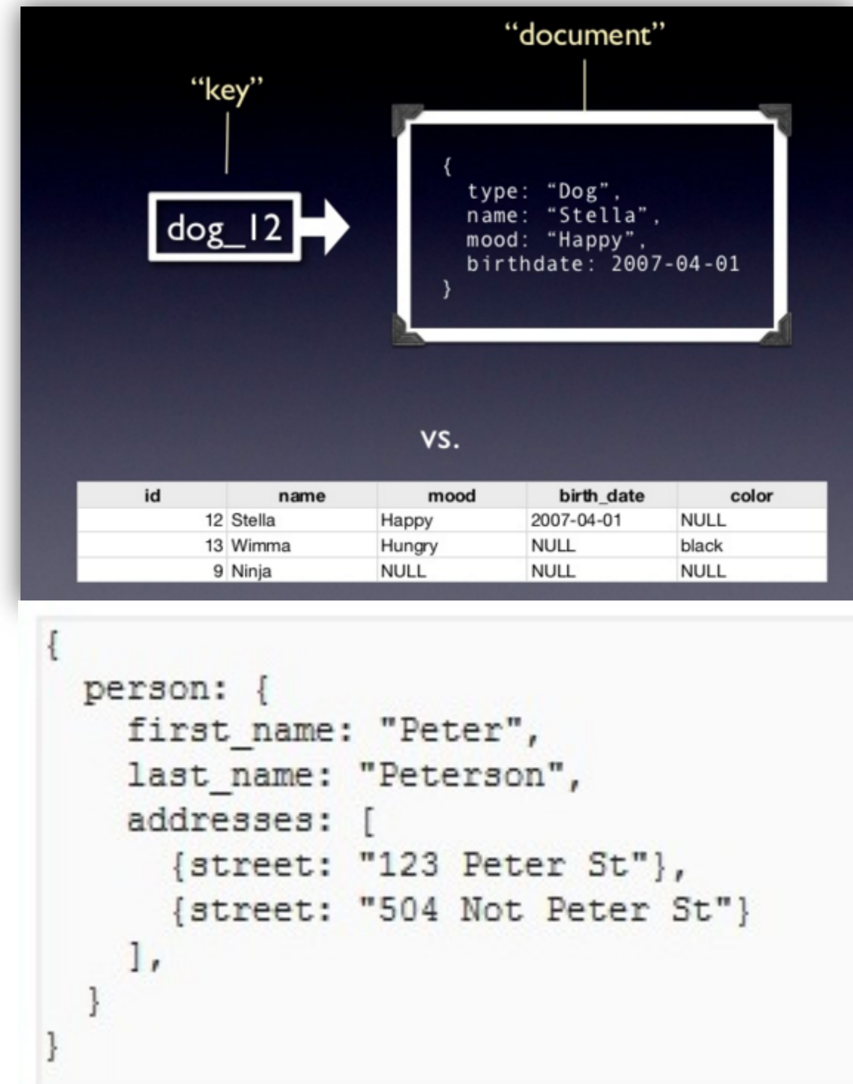
- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

DOCUMENT DATABASES (1)

- In NoSQL databases, a Document Database (or Document Store) stores data in the form of documents, usually using JSON, BSON, or XML formats.
- Each document is a self-contained unit that can store complex and nested data — similar to an object in programming.
- A Document Database stores data as:
  - Collection: A group of related documents (similar to a table in SQL).
  - Document: A single record (similar to a row), but more flexible.
  - Field: A key-value pair inside a document.

DOCUMENT DATABASES (2)

- Pair each key with complex data structure known as data structure.
- Indexes are done via B-Trees.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- Example: CouchDB, MongoDB, Lotus Notes, Redis, etc.

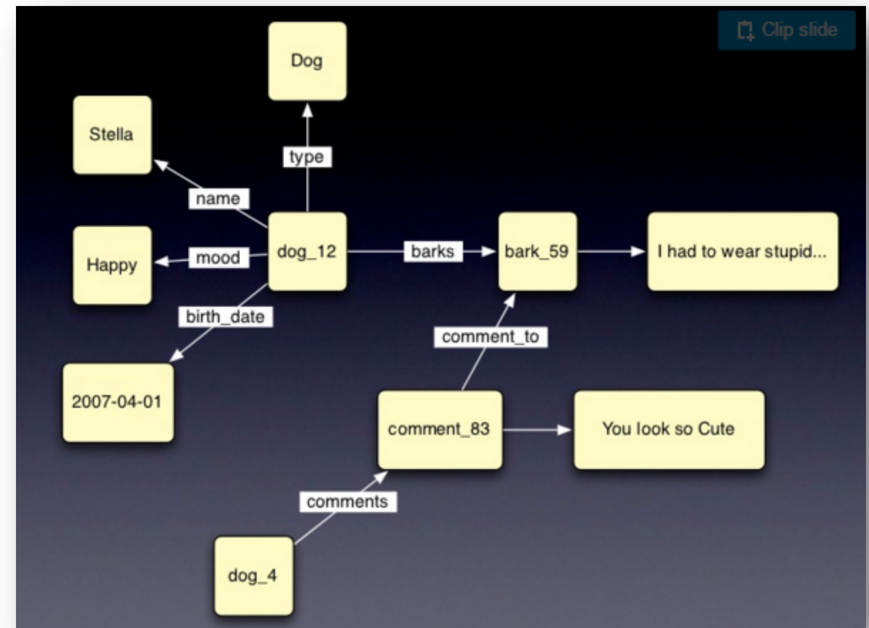


id	name	mood	birth_date	color
12	Stella	Happy	2007-04-01	NULL
13	Wimma	Hungry	NULL	black
9	Ninja	NULL	NULL	NULL

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

GRAPH DATABASES

- Graph Databases are built with nodes, relationships between nodes (edges) and the properties of nodes.
    - Based on Graph Theory.
    - Scale vertically, no clustering.
    - You can use graph algorithms easily
    - Transactions (ACID)
- Example: Neo4J, GraphBase, InfoGrip, etc.



SOME NOSQL CHALLENGES

- Lack of maturity — numerous solutions still in their beta stages
- Lack of commercial support for enterprise users
- Lack of support for data analysis
- Maintenance efforts and skills are required — experts are hard to find

XML FORMAT

- XML stands for Extensible Markup Language
- There are three important characteristics of XML that make it useful in a variety of systems and solutions:
  - XML is extensible
  - XML carries the data, does not present it
  - XML is a public standard

XML SYNTAX (1)

- XML Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Tags and Elements
  - Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element> or <element/>
```

- Root Element

```
<root>
  <x>...</x>
  <y>...</y>
</root>
```

XML SYNTAX (2)

- Case Sensitivity: The names of XML-elements are case-sensitive.
- XML Attributes
  - An attribute specifies a single property for the element, using a name/value pair.
  - An XML- element can have one or more attributes.

```
<a b="x" c="y" b="z">....</a>
```

- XML References
  - Entity References: for example ` ` for spacebar
  - Character References: for example `A` for letter “A”.

XML DOCUMENT EXAMPLE (1)

- **Example 1:**

```
<?xml version="1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

- **Example 2:**

```
<?xml version=“1.0"?>
<pet>
  <items>Cat</items>
  <items>Dog</items>
  <items>Rabbit</items>
</pet>
```

XML DOCUMENT EXAMPLE (2)

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <Zip>10999</Zip>
  </Address>
  <Notes>Please leave packages in shed by driveway.</Notes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

XML: WELL-FORMED

- XML documents must have a root element
- XML elements must have a closing tag
- XML tags are case sensitive
- XML elements must be properly nested
- XML attribute values must always be quoted

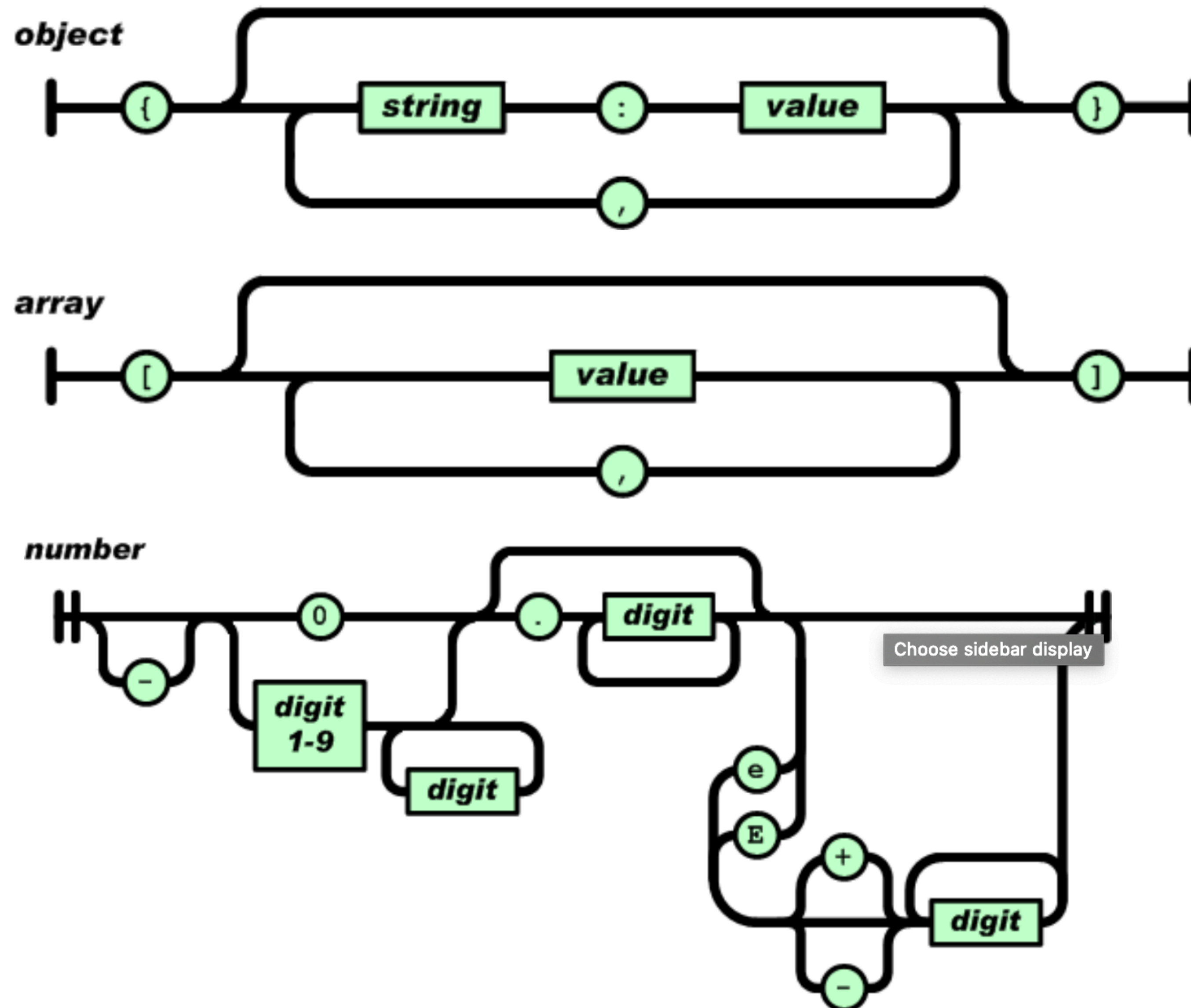
XML: VALIDITY

- Valid XML is An XML document that has an associated document type declaration and complies with the constraints expressed in it.
  - Document Type Definition (DTD)
  - XML Schema
  - Relax NG
  - ISO DSDL
  - etc.

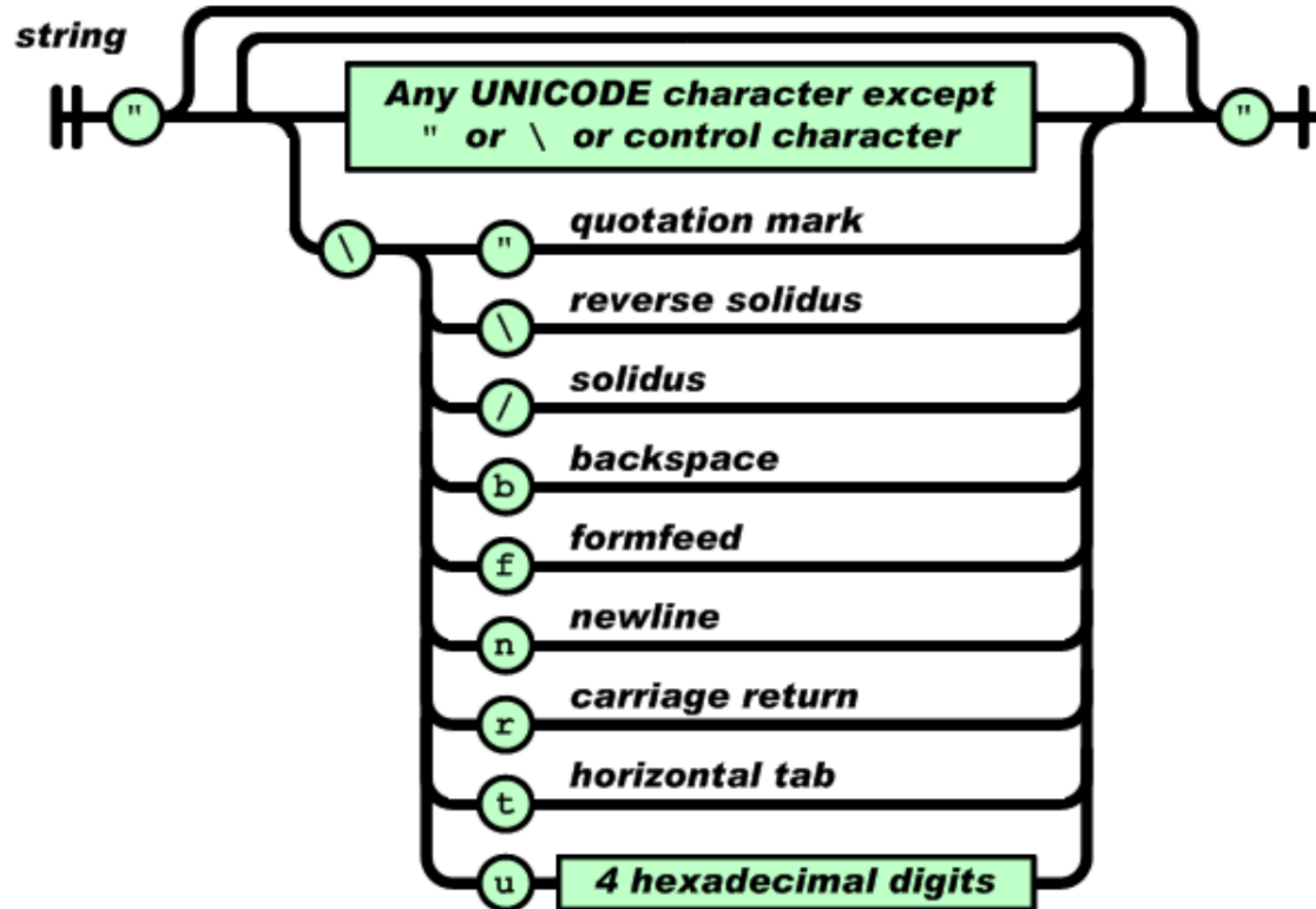
JSON FORMAT

- JSON (JavaScript Object Notation) is a lightweight data-interchange format
- Simple format
  - Easy for humans to read and write
  - Easy for machines to parse and generate
- JSON is a text format
  - Programming language independent
  - Uses conventions that are familiar to programmers of the C- family of languages, including C, C++, C#, Java, JavaScript, Perl, Python

JSON SYNTAX (1)



JSON SYNTAX (2)



JSON: EXAMPLE

```
{
  “firstname”:“John”,
  “lastname”:“Doe”,
  “address”: {
    “street”:“25/25 Putthamonton 4 Road Salaya”,
    “city”: “Nakorn Pathom”,
    “zip”: “73170”
  },
  “phone”:[
    “02-889-2138”,
    “02-889-2225”
  ]
}
```

WHY USE JSON OVER XML

- Lighter and faster than XML as on-the-wire data format
- JSON objects are typed while XML data is typeless
  - JSON types: string, number, array, boolean
  - XML data are all string
- Native data form for JavaScript code
  - Data is readily accessible as JSON objects in your JavaScript code vs. XML data needed to be parsed and assigned to variables through tedious DOM APIs
  - Retrieving values is as easy as reading from an object property in your JavaScript code

JSON STRUCTURES

- A collection of name/value pairs
  - In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array
- An ordered list of values
  - In most languages, this is realised as an array, vector, list, or sequence
- These are universal data structures supported by most modern programming languages

JSON FORMAT

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- Example:

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

JAVASCRIPT JSON EXAMPLE

```
<html>
    <script>

        let myJSON = '{"employees"[
{ "firstName":"John", "lastName":"Doe" },
{ "firstName":"Anna", "lastName":"Smith" },
{ "firstName":"Peter", "lastName":"Jones" } ]}';

        const obj = JSON.parse(myJSON);
        let text = "";
        for (const x in obj.employees){
            text += obj.employees[x].firstName + " "
+ obj.employees[x].lastName + "<br/>";
        }
        document.write(text);

    </script>
</html>
```

JSON VS. XML (1)

- JSON Example:

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

- XML Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON VS. XML (2)

- JSON Example:

```
{
  "student": [
      {
         "id":"01",
         "name": "Tom",
         "lastname": "Price"
      },
      {
         "id":"02",
         "name": "Nick",
         "lastname": "Thameson"
      }
  ]
}
```

- XML Example

```
<?xml version="1.0"
encoding="UTF-8" ?>
<root>
 <student>
    <id>01</id>
    <name>Tom</name>
    <lastname>Price</lastname>
 </student>
 <student>
    <id>02</id>
    <name>Nick</name>
    <lastname>Thameson</lastname>
 </student>
</root>
```

JSON DATA TYPES

- In JSON, values must be one of the following data types:
  - String
  - Number
  - Object (JSON Object)
  - Array
  - Boolean

- JSON Objects Example:

```
{
  "employee":{ "name":"John", "age":30, "city":"New York" }
}
```

JSON ARRAYS

- JSON Example:

```
{
  "name":"John",
  "age":30,
  "cars":[ "Ford", "BMW", "Fiat" ]
}
```

JSON BOOLEAN & NULL

- JSON Example:

```
{
  "flag":true,
  "middlename":null
}
```

JSON/XML (PRACTICE)

- Convert the data in the ***customer*** table of the ***company*** database into JSON and XML format (try writing two rows).
- Try embedding the data from the ***order*** and ***salesman*** tables into the ***customer*** document (try writing one row).

DIFFERENCE BETWEEN JSON AND XML (1)

JSON	**XML**
JSON object has a type	XML data is typeless
JSON types: string, number, array, Boolean	All XML data should be string
Data is readily accessible as JSON objects	XML data needs to be parsed
JSON is supported by most browsers	Cross-browser XML parsing can be tricky
JSON has no display capabilities.	XML offers the capability to display data because it is a markup language.
JSON supports only text and number data type.	XML support various data types such as number, text, images, charts, graphs, etc.

DIFFERENCE BETWEEN JSON AND XML (2)

JSON	**XML**
Supported by many AJAX toolkit	Not fully supported by AJAX toolkit
It supports only UTF-8 encoding.	It supports various encoding.
JSON files are easy to read as compared to XML.	XML documents are relatively more difficult to read and interpret.
It doesn't support comments.	It supports comments.
It is less secured.	It is more secure than JSON.