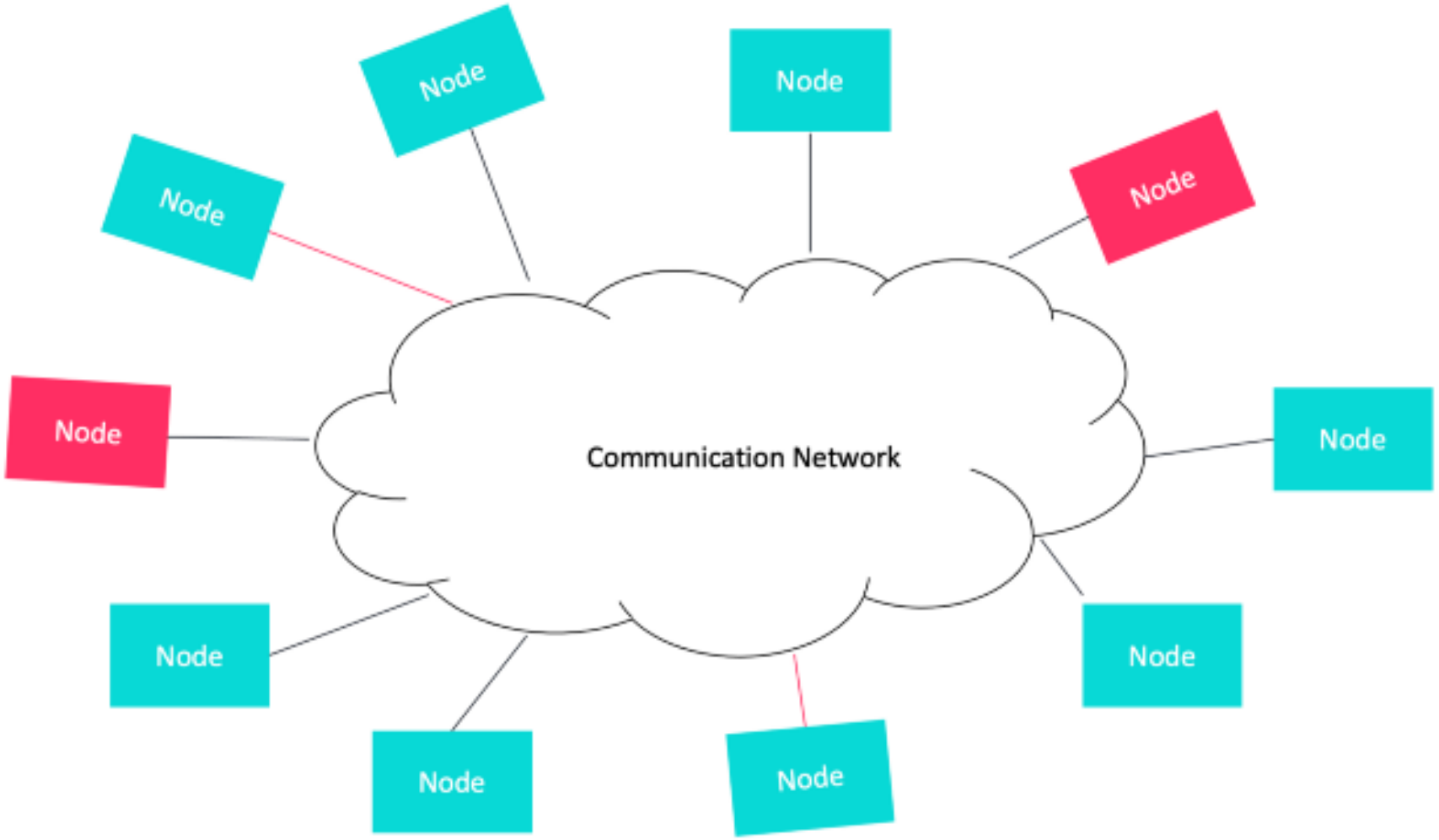
NoSQL Databases

政治大學
資訊科學系
沈錳坤

Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based


Node
Node
Node
Node
Node
Node
Communication Network
Node
Node
Node
Node
Node

Distributed Database

- Distributed computing system: consists of a number of processing sites or nodes that are interconnected by a computer network
- Distributed database: a collection of multiple logically interrelated databases distributed over a computer network
- Distributed database management system: software system that manages a distributed database while making the distribution transparent to the user.

EMPLOYEE

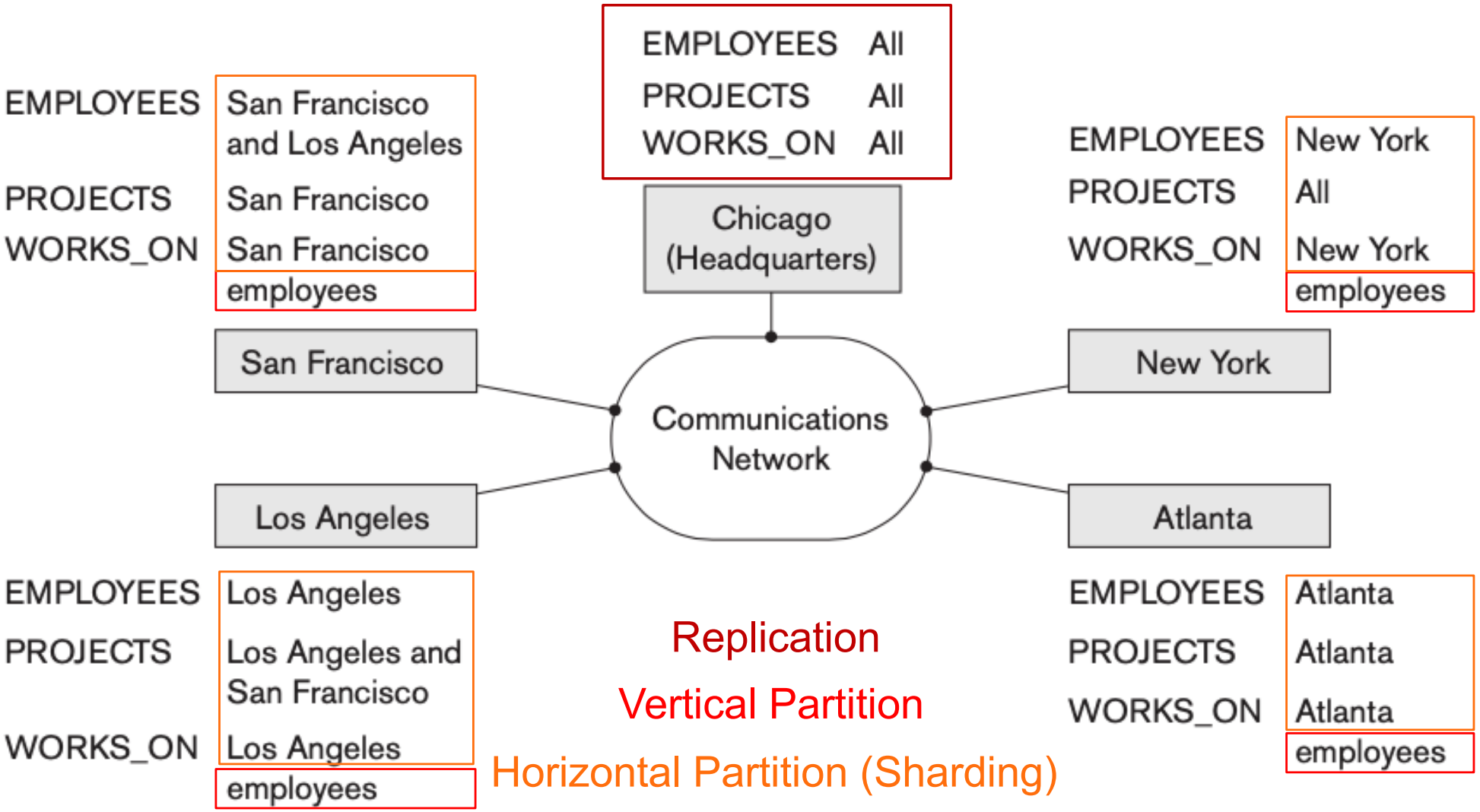
Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4


EMPLOYEES All
PROJECTS All
WORKS_ON All
Chicago (Headquarters)
EMPLOYEES San Francisco and Los Angeles
PROJECTS San Francisco
WORKS_ON San Francisco
employees
San Francisco
Los Angeles
Communications Network
New York
Atlanta
EMPLOYEES New York
PROJECTS All
WORKS_ON New York
employees
EMPLOYEES Los Angeles
PROJECTS Los Angeles and San Francisco
WORKS_ON Los Angeles
employees
EMPLOYEES Atlanta
PROJECTS Atlanta
WORKS_ON Atlanta
employees
Replication
Vertical Partition
Horizontal Partition (Sharding)

Transparency

- Transparency
  - hiding implementation details from end users/ application developer
  - Data organization transparency (distribution or network transparency)
    - Location transparency
    - Naming transparency
  - Replication transparency: unaware of existence of copies
  - Fragmentation transparency: unaware of existence of fragments
    - Horizontal fragmentation (partition)
    - Vertical fragmentation (partition)
  - Design transparency: unaware of how the distributed database is designed
  - Execution transparency: unaware of where a transaction executes

Concepts of Distributed DB

- Availability & Reliability
  - Fault Tolerance
    - Availability: probability that the system is continuously available during a time interval
    - Reliability: probability that a system is running (not down) at a certain time point
- Scalability
  - determines the extent to which the system can expand its capacity while continuing to operate without interruption
  - Horizontal scalability: expanding the number of nodes
  - Vertical scalability: expanding the capacity of the individual nodes
  - Partition Tolerance: system should have the capacity to continue operating while the network is partitioned.
- Autonomy
  - determines the extent to which individual nodes can operate independently.

Concurrency Control & Recovery in Distributed DB

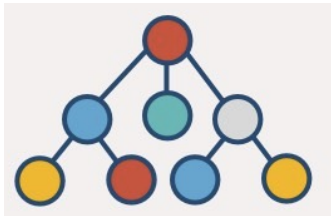
- Problems arise in a distributed DBMS
  - Dealing with consistency of multiple copies of the data items
  - Failure of individual sites
  - Failure of communication links
  - Distributed commit
  - Distributed deadlock

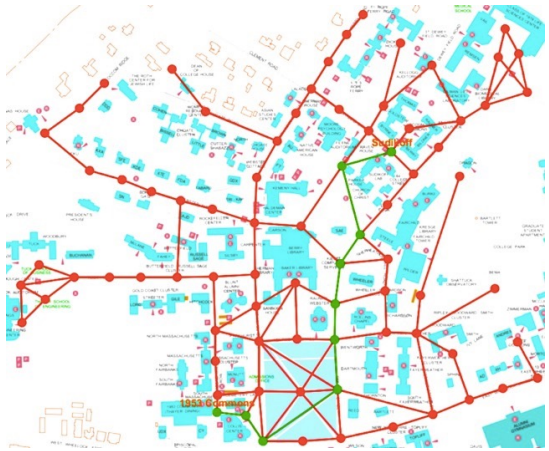
Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based

Data

- Structured data
- Text ?
- Image, Audio, Video content ?
- Tree ?
- Graph ?
- Geographical data ?





NoSQL

- Not Only SQL, rather than No to SQL
- Most NoSQL systems
  - distributed databases or distributed storage systems
  - focus on
    - semi-structured data storage
    - high performance
    - availability
    - data replication
    - scalability
  - as opposed to
    - structured data storage
    - Immediate data consistency
    - powerful query language

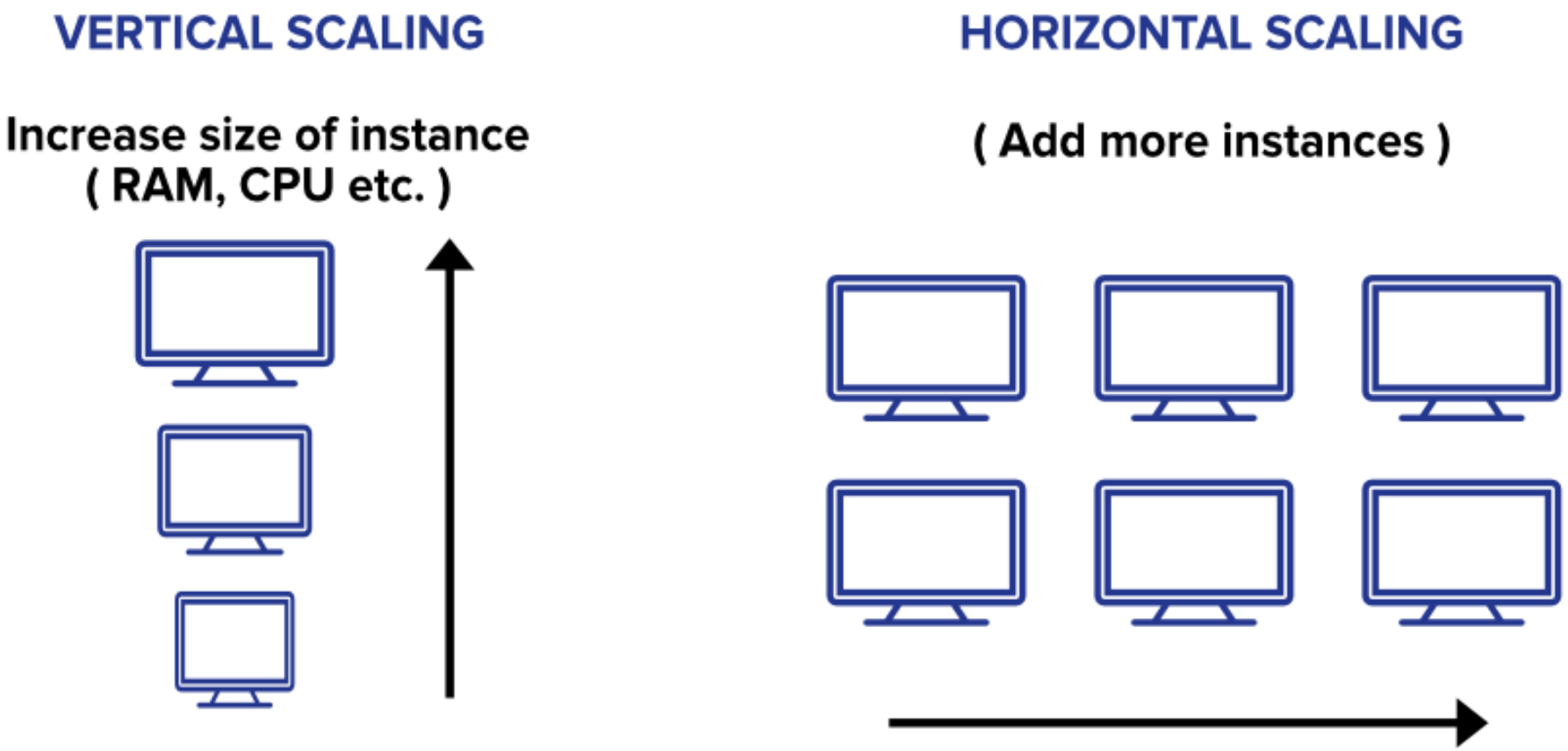
Applications

- e-mails
- social media
  - posts (text, image, video)
  - user profile
  - social relationships
- …

Characteristics of NoSQL

- Scalability
  - Horizontal scalability: distributed system is expanded by adding more nodes as the volume of data grows (NoSQL)
  - Vertical scalability: distributed system is expanded by adding the storage & computing power of existing nodes
- Availability, Replication & Eventual Consistency
  - Data is replicated over two or more nodes
  - Update must be applied to every copy
  - Slow down write performance if serializable consistency is required
  - Eventual consistency: relaxed form of consistency

Guarantees that the outcome of executing a set of transactions is the same as if the transactions were executed one after another.



(ref: https://www.geeksforgeeks.org/system-design-horizontal-and-vertical-scaling/)

Characteristics of NoSQL (cont.)

- Replication Models
  - Master-slave replication: requires one master copy, all write operations applied to master and propagated to slave copies
  - Master-Master replication: allows read & write at any copies but will be temporarily inconsistency.
- Sharding of Files
  - Horizontal partitioning
- High performance Data Access
  - Hashing
  - Range partitioning

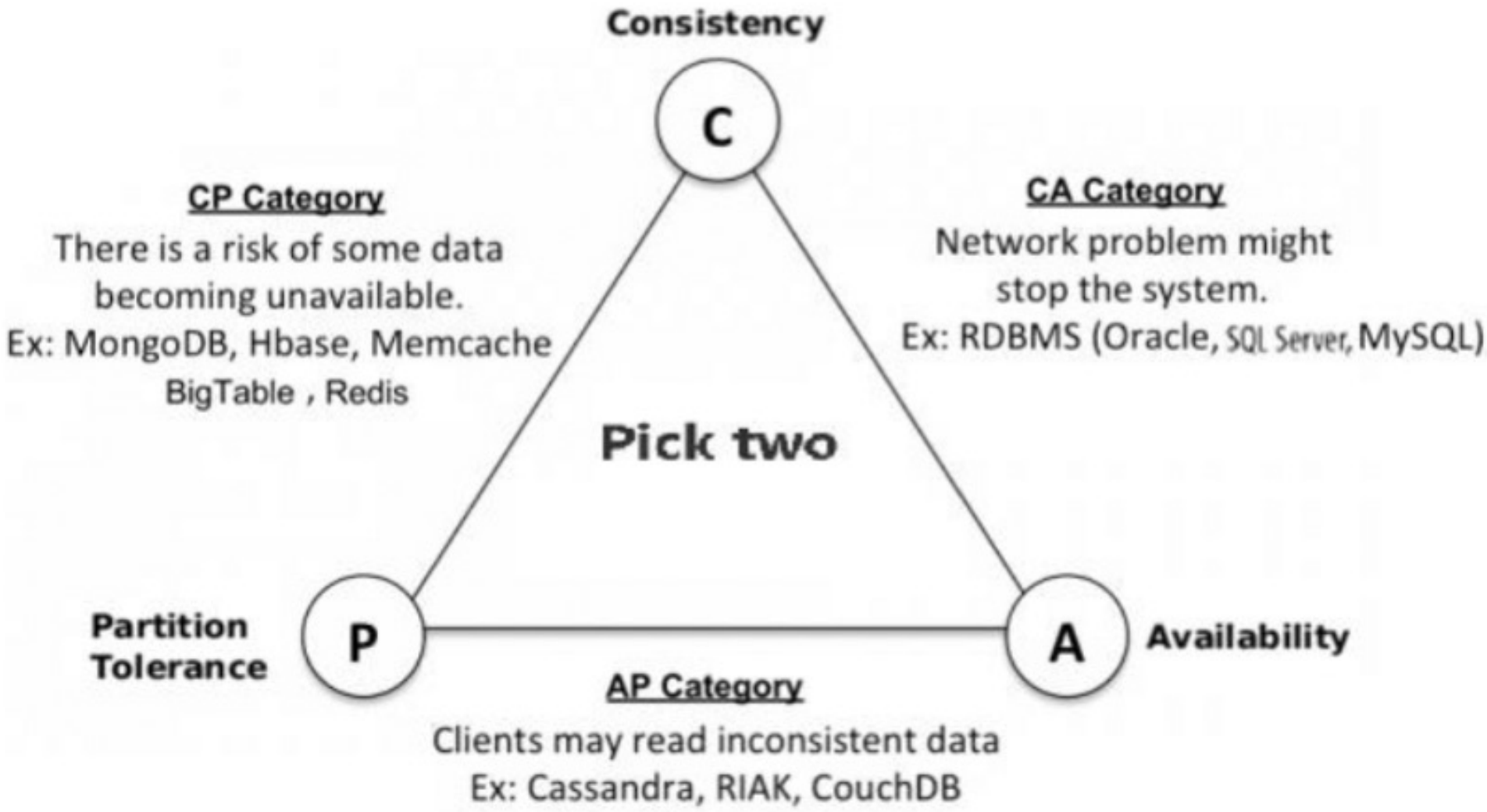
Characteristics of NoSQL (cont.)

- Not requiring a schema
  - Semi-structured, self-describing data
  - JSON, XML
- Less powerful query language
  - NoSQL provide a set of functions & operations as API
  - CRUD (Create, Read, Update, Delete)
  - Do not provide JOIN
- Versioning
  - Some NoSQL systems provide storage of multiple versions of data items with timestamp

CAP Theorem

- Serialization is hard to enforce in NoSQL databases
- It is hard to implement ACID properties in NoSQL
- CAP
  - Consistency (among replicated copies)
    - all clients see the same data at the same time
  - Availability (of the systems for read & write operations)
    - any client making a request for data gets a response, even if one or more nodes are down.
  - Partition tolerance
    - the system must continue to work despite any number of communication breakdowns between nodes in the system.

(ref: https://www.ibm.com/topics/cap-theorem)


Consistency
C
CP Category
There is a risk of some data becoming unavailable.
Ex: MongoDB, Hbase, Memcache
BigTable , Redis
CA Category
Network problem might stop the system.
Ex: RDBMS (Oracle, SQL Server, MySQL)
Pick two
Partition Tolerance
P
A
Availability
AP Category
Clients may read inconsistent data
Ex: Cassandra, RIAK, CouchDB

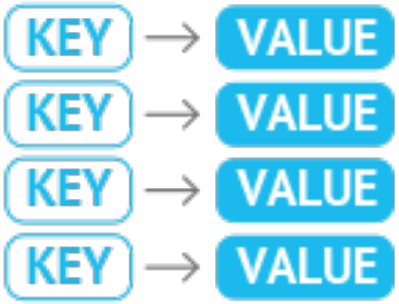
CAP Theorem (cont.)

- It is not possible to guarantee CAP at the same time while having replication.
- A weaker consistency in NoSQL, guaranteeing AP, is acceptable for some applications.
- Eventual consistency is used some times to satisfy CAP.
- Consistency
  - CAP: refers to the consistency of values in different copy of the same data item in a replicated distributed system
  - ACID: refer to the fact that a transaction will not violate the integrity constraints specified on the DB schema

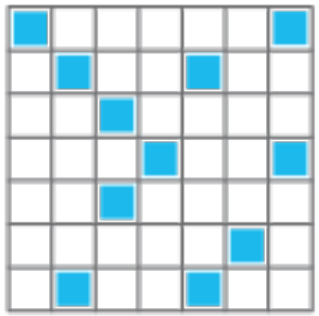
Categories of NoSQL Syetems

- Document-based NoSQL Systems
- Key-value NoSQL Stores
- Column-based NoSQL Systems
- Graph-based NoSQL Systems

Document



Key-Value



Column-Family

Graph

Commercial NoSQL

- Document-based
  - MongoDB, CouchDB
- Key-value
  - Amazon DynamoDB, Redis
- Key-value + Column-based
  - Facebook (Apache) Cassandra
- Column-based
  - Google BigTable
  - Apache Hbase
- Graph-based
  - Neo4j, GraphBase

Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based

Document-based NoSQL

- Store data as collections of similar documents
- Resemble complex objects or XML documents
- No schema required, self-describing data
- Documents in a collection should be similar, they can have different data elements (attributes)
- Users can request to create indexes on some of the data elements
- Documents can be specified in JSON, XML, ...
- MongoDB, CouchDB, …

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

(a) project document with an array of embedded workers:

```
{
    _id:        "P1",
    Pname:      "ProductX",
    Plocation:  "Bellaire",
    Workers: [
        { Ename: "John Smith",
          Hours: 32.5
        },
        { Ename: "Joyce English",
          Hours: 20.0
        }
    ]
);
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

(b) project document with an embedded array of worker ids:

```
{
    _id:          "P1",
    Pname:        "ProductX",
    Plocation:    "Bellaire",
    WorkerIds:    [ "W1", "W2" ]
}
    { _id:        "W1",
    Ename:        "John Smith",
    Hours:        32.5
}
    { _id:        "W2",
    Ename:        "Joyce English",
    Hours:        20.0
}
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

(c) normalized project and worker documents (not a fully normalized design for M:N relationships):

```
{
    _id:        "P1",
    Pname:      "ProductX",
    Plocation:  "Bellaire"
}
{   _id:        "W1",
    Ename:      "John Smith",
    ProjectId:  "P1",
    Hours:      32.5
}
{   _id:        "W2",
    Ename:      "Joyce English",
    ProjectId:  "P1",
    Hours:      20.0
}
```

Document Database: MongoDB

- MongoDB
  - database: database
  - collection: table
  - document
    - tuple, record
    - composed of field and value pairs
    - similar to JSON objects
    - values of fields may include other documents, arrays, and arrays of documents
  - BSON: binary representation of JSON

document ∈ collection ∈ database

record ∈ table ∈ database

Document Database: MongoDB

- MongoDB CRUD

C

```
db.users.insertOne(                    ← collection
   {
      name: "sue",                     ← field: value
      age: 26,                         ← field: value      } document
      status: "pending"                ← field: value
   }
)
```

R

```
db.users.find(                         ← collection
   { age: { $gt: 18 } },               ← query criteria
   { name: 1, address: 1 }             ← projection
).limit(5)                             ← cursor modifier
```

U

```
db.users.updateMany(                   ← collection
   { age: { $lt: 18 } },               ← update filter
   { $set: { status: "reject" } }      ← update action
)
```

D

```
db.users.deleteMany(                   ← collection
   { status: "reject" }                ← delete filter
)
```

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	`$lookup`, embedded documents
primary key	primary key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline See the SQL to Aggregation Mapping Chart.
SELECT INTO NEW_TABLE	`$out` See the SQL to Aggregation Mapping Chart.
MERGE INTO TABLE	`$merge` (Available starting in MongoDB 4.2) See the SQL to Aggregation Mapping Chart.
UNION ALL	`$unionWith` (Available starting in MongoDB 4.4)

Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based

Key-value Stores

- Key
  - an unique id associated with a data item
  - is used to locate data item rapidly
- Value
  - is the data item itself
  - can have very different formats
  - unstructured, semi-structured, or structured
  - the application using the key-value store has to interpret the structure of the data value
- Every value (data item) must be associated with a unique key
- Retrieving the value by supplying the key must be very fast

Key-value Stores: DynamoDB

- DynamoDB
  - basic model uses the concepts of tables, items, attributes
  - table: hold a collection of items
  - item
    - is a self-describing record
    - composed of one or more (attribute, value) pairs
  - Attribute: fundamental data element, something that does not need to be broken down any further
  - When a table is created, it is required to specify a table name & a primary key.
  - primary key is the key & item is the value

People

```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Music

```
{
    "Artist": "No One You Know",
    "SongTitle": "My Dog Spot",
    "AlbumTitle": "Hey Now",
    "Price": 1.98,
    "Genre": "Country",
    "CriticRating": 8.4
}
```

```
{
    "Artist": "No One You Know",
    "SongTitle": "Somewhere Down The Road",
    "AlbumTitle": "Somewhat Famous",
    "Genre": "Country",
    "CriticRating": 8.4,
    "Year": 1984
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Still in Love",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 2.47,
    "Genre": "Rock",
    "PromotionInfo": {
        "RadioStationsPlaying": [
            "KHCR",
            "KQBX",
            "WTNR",
            "WJJH"
        ],
        "TourDates": {
            "Seattle": "20150625",
            "Cleveland": "20150630"
        },
        "Rotation": "Heavy"
    }
}
```

```
{
    "Artist": "The Acme Band",
    "SongTitle": "Look Out, World",
    "AlbumTitle": "The Buck Starts Here",
    "Price": 0.99,
    "Genre": "Rock"
}
```

Key-value Stores: DynamoDB (cont.)

- 2 kinds of primary keys
  - Partition key
    - A simple primary key, composed of one attribute
    - DynamoDB uses the partition key's value as input to an internal hash function
    - output from the hash function determines the partition (physical storage) in which the item will be stored.
    - In a table that has only a partition key, no two items can have the same partition key value.
  - Partition key + Sort key (composite primary key)
    - is composed of two attributes = partition key + sort key
    - all items with the same partition key value are stored together, in sorted order by sort key value

Key-value Stores: DynamoDB (cont.)

- Secondary Indexes
  - query the data in the table using an alternate key, in addition to queries against the primary key.
  - 2 kinds of indexes
    - global secondary index – An index with a partition key and sort key that can be different from those on the table.
    - local secondary index – An index that has the same partition key as the table, but a different sort key.

```
// Return a single song, by primary key
{
    TableName: "Music",
    KeyConditionExpression: "Artist = :a and SongTitle = :t",
    ExpressionAttributeValues: {
        ":a": "No One You Know",
        ":t": "Call Me Today"
    }
}
```

```
{
    TableName: "Music",
    Key: {
        "Artist":"No One You Know",
        "SongTitle":"Call Me Today"
    },
    UpdateExpression: "SET RecordLabel = :label",
    ConditionExpression: "Price >= :p",
    ExpressionAttributeValues: {
        ":label": "Global Records",
        ":p": 2.00
    }
}
```

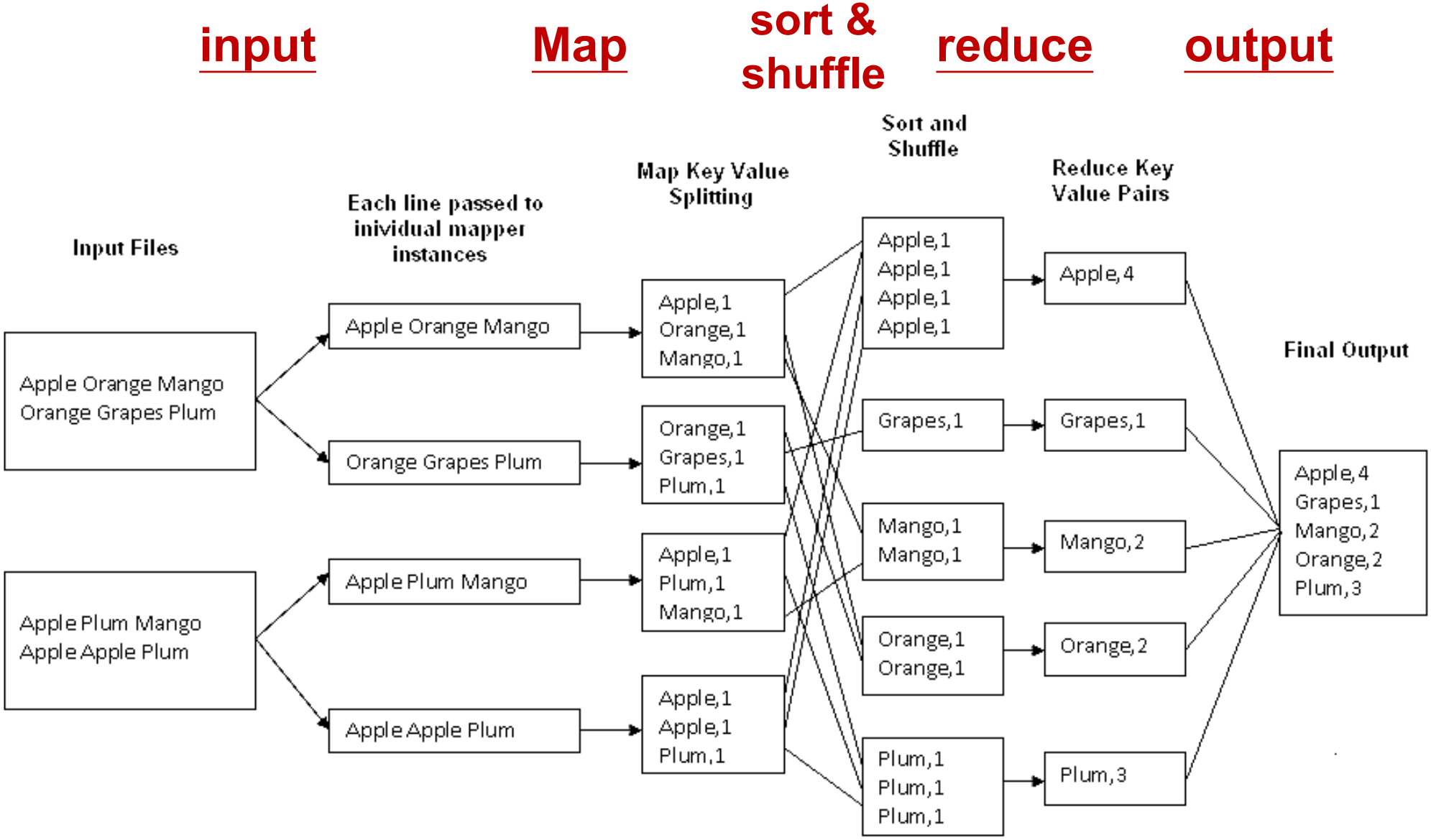
MapReduce

- A Programming model & associated implementation for processing and generating large datasets
- User specify the computation in terms of

  map & *reduce* functions
- The underlying runtime system automatically
  - parallelizes the computation across large scale clusters of machines,
  - handles machine failures and
  - schedules inter-machine communications

MapReduce (cont.)

1. Programming model
2. Execution framework that coordinates the execution of programs
3. Software implementation of the programming model and the execution framework (e.g. Hadoop)


input
Map
sort & shuffle
reduce
output
Input Files
Each line passed to inividual mapper instances
Map Key Value Splitting
Sort and Shuffle
Reduce Key Value Pairs
Final Output
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple Orange Mango
Orange Grapes Plum
Apple Plum Mango
Apple Apple Plum
Apple,1
Orange,1
Mango,1
Orange,1
Grapes,1
Plum,1
Apple,1
Plum,1
Mango,1
Apple,1
Apple,1
Plum,1
Apple,1
Apple,1
Apple,1
Apple,1
Grapes,1
Mango,1
Mango,1
Orange,1
Orange,1
Plum,1
Plum,1
Plum,1
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3
Apple,4
Grapes,1
Mango,2
Orange,2
Plum,3

Mappers & Reducers

- Map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$
- Reduce: $(k_2, [v_2]) \rightarrow (k_3, [v_3])$
  - keys & values may be primitives or complex structures
  - […]: a list
  - key: url, value: HTML content
  - key: node id, value: adjacent lists of nodes

Mappers & Reducers

- Map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$
- Reduce: $(k_2, [v_2]) \rightarrow (k_3, [v_3])$
- Mapper
  - is applied to every input key-value pair (split across an arbitrary number of files)
  - generate an arbitrary number of intermediate key-value pairs
- Reducer
  - is applied to all values associated with the same intermediate key
  - generate output key-value pairs.

Mappers & Reducers

- Map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$
- Reduce: $(k_2, [v_2]) \rightarrow (k_3, [v_3])$

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)

class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        Emit(term t, count sum)
```

Join in Map Reduce

page_view

pageid	**userid**	time
A	**111**	9:08:01
B	**111**	9:08:13
A	**222**	9:08:14

X

user

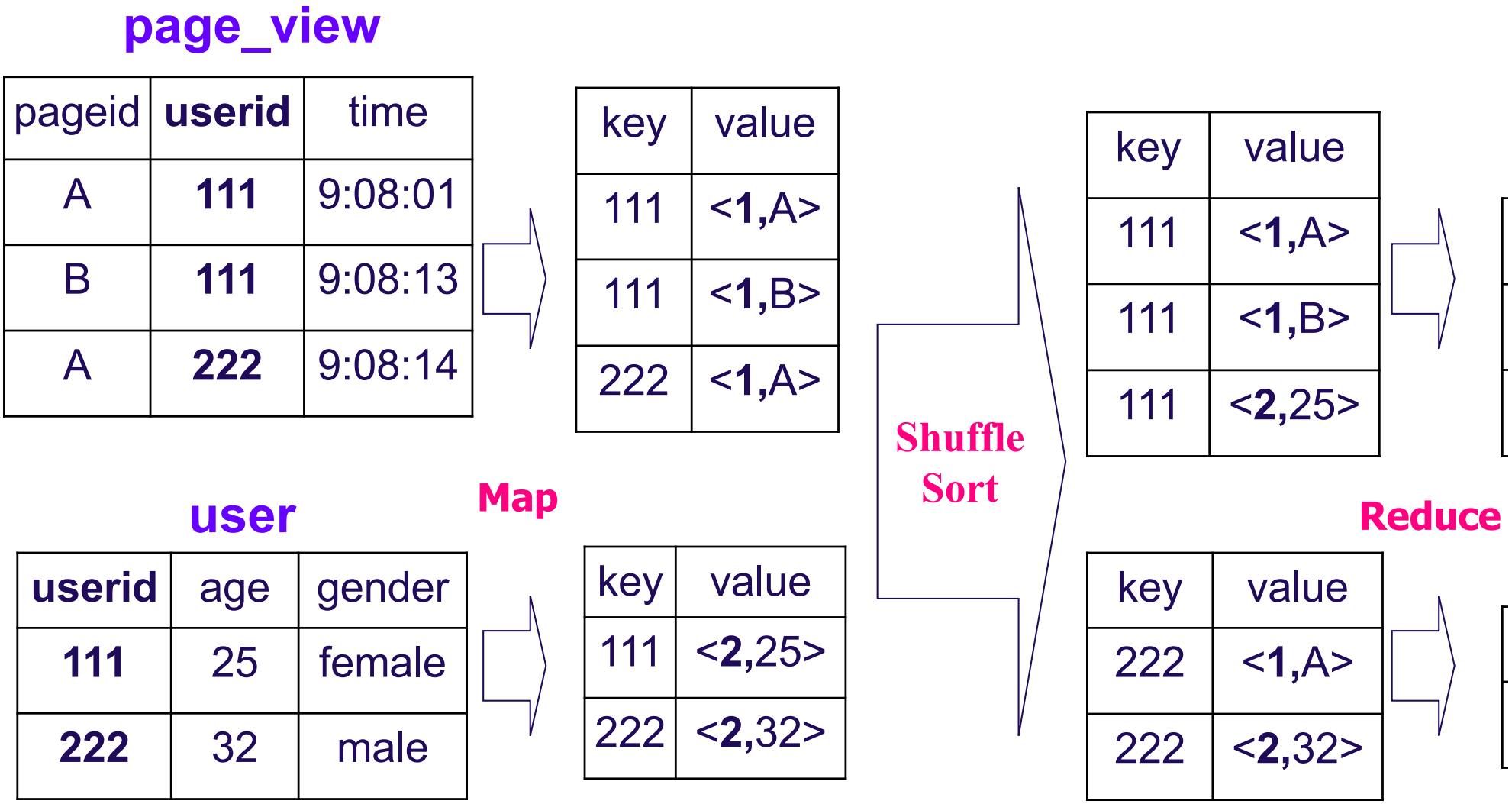
userid	age	gender
111	25	female
222	32	male

=

pv_users

pageid	age
A	25
B	25
A	32

Join in Map Reduce (cont.)



Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based

Column-based NoSQL

- Main difference between column-based & key-value
  - Nature of the key
  - Column-based: key is multi-dimensional
  - E.g. Hbase: key is a combination of

    (table name, row key, column, timestamp).
- Google BigTable (using GFS, Google File System)
- Apache Hbase (using HDFS, Hadoop Distributed File System)

Column-based NoSQL: Hbase

- Hbase
  - namespace: database, a collection of tables
  - table: data in a table is stored as self-describing rows
  - row: has a unique row key (lexicographically ordered strings)
  - column family
    - a table is associated with one or more column families
    - each column family have a name
    - column families must be specified when the table is created.
  - column qualifier
    - each column family can be associated with many column qualifier when the data is loaded into a table.
    - column qualifiers make the model a self-describing data model because the qualifiers can be dynamically specified as new rows are created and inserted into the table
  - cell: basic data item, key=(table name, row key, column, timestamp)

Column-based NoSQL: Hbase (cont.)

- application programmers know which column qualifiers belong to each column family, even though they have the flexibility to create new column qualifiers on the fly when new data rows are created.
- concept of column family is somewhat similar to vertical partitioning columns (attributes) that are accessed together
- Columns qualifiers belong to the same column family are stored in the same files
- Each column family of a table is stored in its own files using the HDFS file system.

(a) creating a table:

create 'EMPLOYEE', 'Name', 'Address', 'Details'

(b) inserting some row data in the EMPLOYEE table:

put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'

(c) Some Hbase basic CRUD operations:

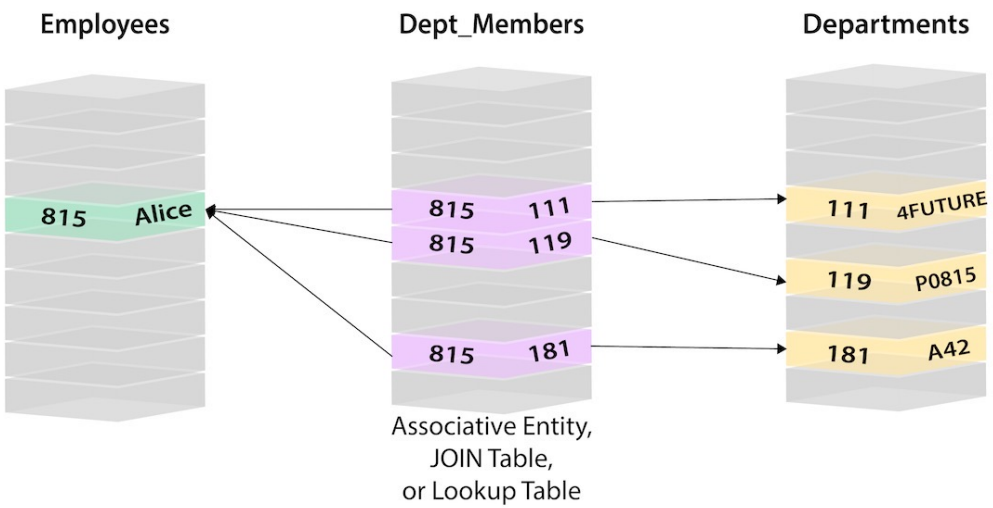
Creating a table: create <tablename>, <column family>, <column family>, …
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>

Contents

- Distributed Database
- NoSQL
  - Document-based
  - Key-value
  - Column-based
  - Graph-based

Graph Databases

- In relational data model
  - references to other rows and tables are indicated by referring to primary key attributes via foreign key columns
  - Joins are computed at query time by matching primary and foreign keys of all rows in the connected tables
  - Joins are compute-heavy and memory-intensive and have an exponential cost.
  - With many-to-many relationships, JOIN table that holds foreign keys of both the participating tables, further increases join operation costs.



Graph Databases (cont.)

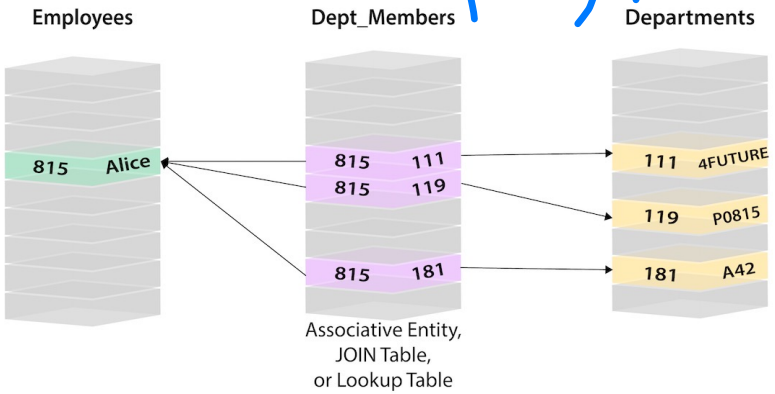
- Data is represented as a graph
- A graph is collection of vertices and edges
- Both nodes & edges can be labeled to indicate types of entities & relationships
- Pre-materialized relationships

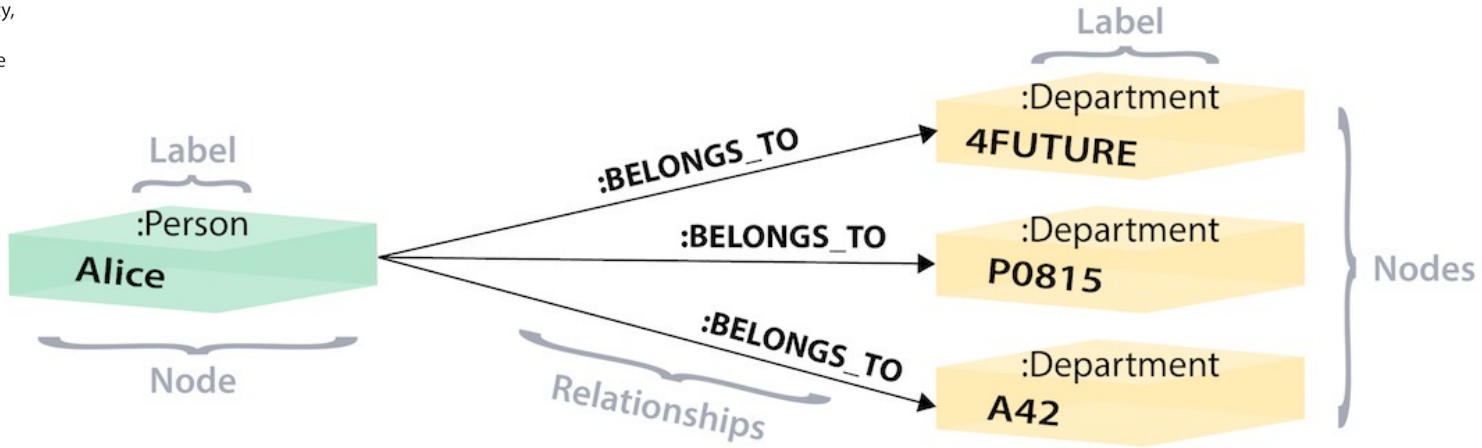
↳ Stored explicitly, allowing for efficient traversal

↳ Not entity type!

Entity: node

Relationship: edge

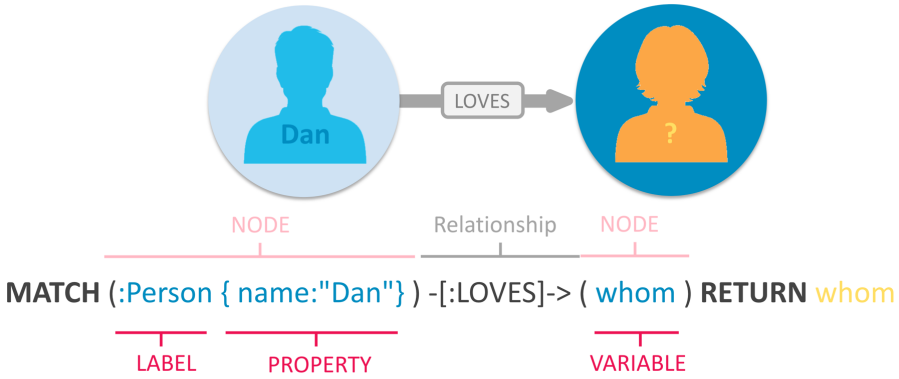




Graph Databases (cont.)

enhanced (extended) E-R model

- Neo4j
  - resembles how data is represented in ER & EER model
  - data is organized as nodes, relationships, and properties
    - nodes: entities
    - node labels: entity types & subclass
    - relationship: relationship instance
    - relationship label: relationship types
    - properties: attributes
  - query language: Cypher



  - 3 Differences
    - relationship is directed in Neo4j
    - node may have no label in Neo4j
    - Neo4j is for high performance distributed DB system

creating some nodes for the COMPANY data (from Figure 5.6):

```
CREATE (e1: EMPLOYEE, {Empid: '1', Lname: 'Smith', Fname: 'John', Minit: 'B'})
CREATE (e2: EMPLOYEE, {Empid: '2', Lname: 'Wong', Fname: 'Franklin'})
CREATE (e3: EMPLOYEE, {Empid: '3', Lname: 'Zelaya', Fname: 'Alicia'})
CREATE (e4: EMPLOYEE, {Empid: '4', Lname: 'Wallace', Fname: 'Jennifer', Minit: 'S'})
...
CREATE (d1: DEPARTMENT, {Dno: '5', Dname: 'Research'})
CREATE (d2: DEPARTMENT, {Dno: '4', Dname: 'Administration'})
...
CREATE (p1: PROJECT, {Pno: '1', Pname: 'ProductX'})
CREATE (p2: PROJECT, {Pno: '2', Pname: 'ProductY'})
CREATE (p3: PROJECT, {Pno: '10', Pname: 'Computerization'})
CREATE (p4: PROJECT, {Pno: '20', Pname: 'Reorganization'})
...
CREATE (loc1: LOCATION, {Lname: 'Houston'})
CREATE (loc2: LOCATION, {Lname: 'Stafford'})
CREATE (loc3: LOCATION, {Lname: 'Bellaire'})
CREATE (loc4: LOCATION, {Lname: 'Sugarland'})
```

creating some relationships for the COMPANY data (from Figure 5.6):

```
CREATE (e1) – [ : WorksFor ] –> (d1)
CREATE (e3) – [ : WorksFor ] –> (d2)
...
CREATE (d1) – [ : Manager ] –> (e2)
CREATE (d2) – [ : Manager ] –> (e4)
...
CREATE (d1) – [ : LocatedIn ] –> (loc1)
CREATE (d1) – [ : LocatedIn ] –> (loc3)
CREATE (d1) – [ : LocatedIn ] –> (loc4)
CREATE (d2) – [ : LocatedIn ] –> (loc2)
...
CREATE (e1) – [ : WorksOn, {Hours: '32.5'} ] –> (p1)
CREATE (e1) – [ : WorksOn, {Hours: '7.5'} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p1)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p3)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p4)
...
```

A relationship can have properties

(c) **Basic simplified syntax of some common Cypher clauses:**

Finding nodes and relationships that match a pattern: MATCH <pattern>
Specifying aggregates and other query variables: WITH <specifications>
Specifying conditions on the data to be retrieved: WHERE <condition>
Specifying the data to be returned: RETURN <data>
Ordering the data to be returned: ORDER BY <data>
Limiting the number of returned data items: LIMIT <max number>
Creating nodes: CREATE <node, optional labels and properties>
Creating relationships: CREATE <relationship, relationship type and optional properties>
Deletion: DELETE <nodes or relationships>
Specifying property values and labels: SET <property values and labels>
Removing property values and labels: REMOVE <property values and labels>

(d) Examples of simple Cypher queries:

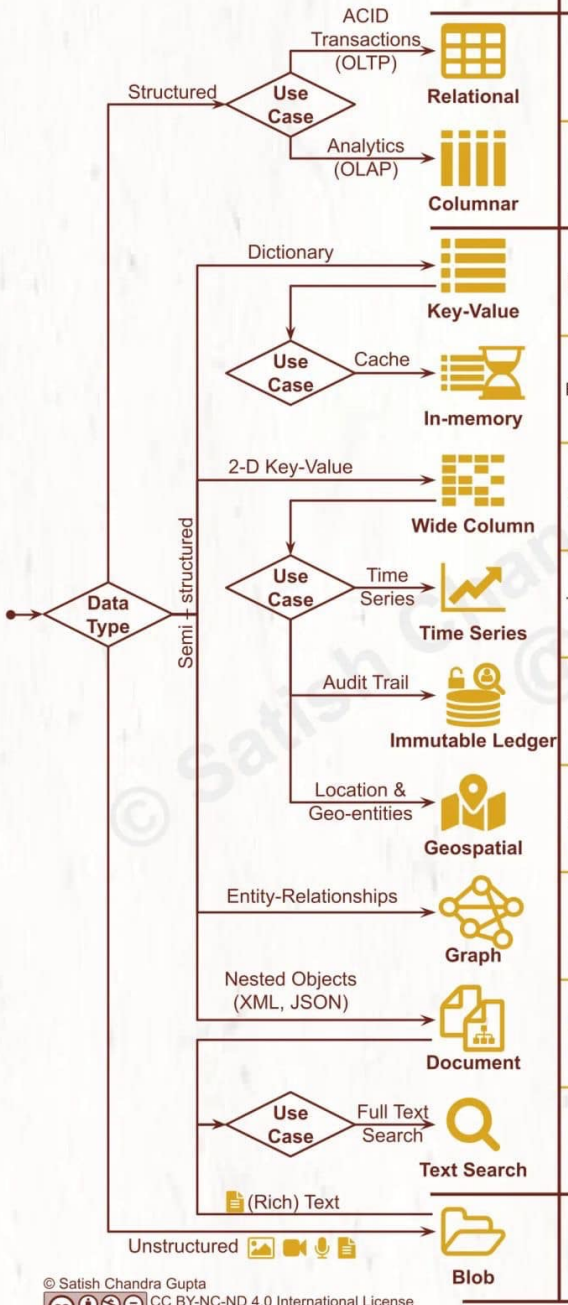
1. MATCH (d : DEPARTMENT {Dno: '5'}) – [: LocatedIn] → (loc)
   RETURN d.Dname , loc.Lname
2. MATCH (e: EMPLOYEE {Empid: '2'}) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
3. MATCH (e) – [w: WorksOn] → (p: PROJECT {Pno: 2})
   RETURN p.Pname, e.Ename , w.Hours
4. MATCH (e) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
5. MATCH (e) – [w: WorksOn] → (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
   LIMIT 10
6. MATCH (e) – [w: WorksOn] → (p)
   WITH e, COUNT(p) AS numOfprojs
   WHERE numOfprojs > 2
   RETURN e.Ename , numOfprojs
   ORDER BY numOfprojs
7. MATCH (e) – [w: WorksOn] → (p)
   RETURN e , w, p
   ORDER BY e.Ename
   LIMIT 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Engineer'

Graph Databases (cont.)

- Data problems involve many-to-many relationships with heterogeneous data
  - navigate deep hierarchies
  - discover inter-relationships between items
  - find hidden connections between distant items
- Focus on the relationships more than about the individual data elements
  - social network
  - payment networks
  - road network
  - …

SQL vs. NoSQL: Cheatsheet for AWS, Azure, and Google Cloud

ml4devs.com/datastores

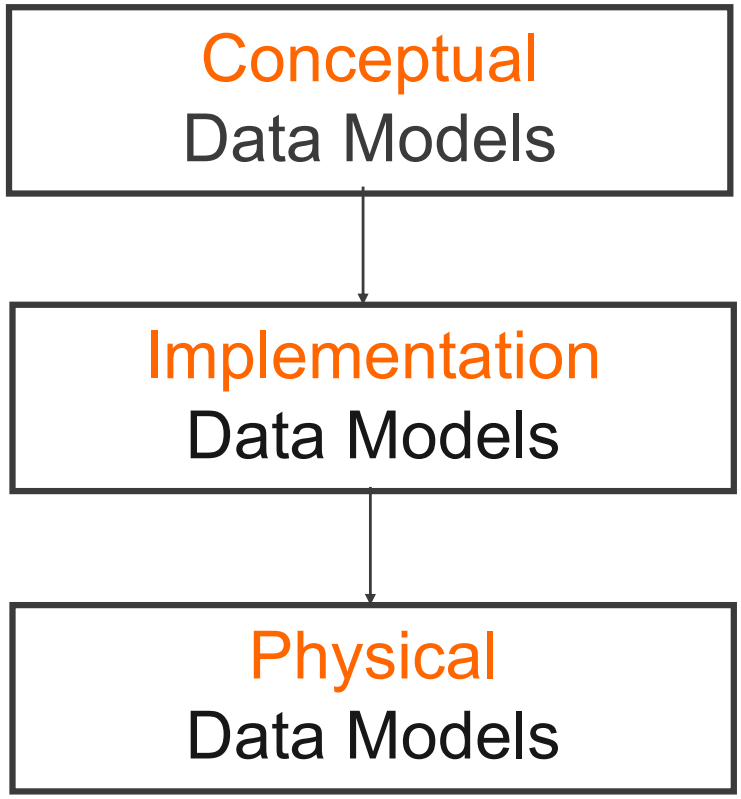


	aws	Azure	Google Cloud	Cloud Agnostic
Relational	RDS, Aurora	Azure SQL Database	Cloud SQL, Cloud Spanner	SQL Server, Oracle, DB2, MySQL, PostgreSQL
Columnar	RedShift	Azure Synapse	BigQuery	Snowflake, ClickHouse, Druid, Pinot, Databricks
Key-Value	DynamoDB	Cosmos DB	BigTable	Redis, ScyllaDB, Ignite
In-memory	ElastiCache	Azure Cache for Redis	Memory-store	Redis, Memcached, Hazelcast, Ignite
Wide Column	Keyspaces	Cosmos DB	BigTable	HBase, Cassandra, ScyllaDB
Time Series	Timestream	Cosmos DB	BigTable, BigQuery	TimescaleDB, OpenTSDB, InfluxDB, ScyllaDB
Immutable Ledger	Quantum Ledger Database (QLDB)	Azure SQL Database Ledger	✖	Hyperledger Fabric
Geospatial	Keyspaces	Cosmos DB	BigTable, BigQuery	Solr, PostGIS, MongoDB (GeoJSON)
Graph	Neptune	Cosmos DB	JanusGraph + BigTable	OrientDB, Neo4J, Giraph
Document	Document DB	Cosmos DB	Firestore	MongoDB, Couchbase, Solr
Text Search	Open-Search, Cloud-Search	Cognitive Search	Search APIs on Datastores	Elastic-Search, Solr, Elassandra
Blob	S3	Blob Storage	Cloud Storage	HDFS, MinIO

NoSQL

scgupta.me
twitter.com/scgupta
linkedin.com/in/scgupta

Data Models



Entity-Relationship Model

ER to Relational Mapping

Relational Data Model
SQL, Relational Algebra
Normalization

DBMS
Record Structure, Indexing
Query Processing
Query Optimization
Transaction Processing

NoSQL

Hadoop Distributed File System,
Google File System,
Simple Storage System

Rank			DBMS	Database Model	Score		
Oct 2021	Sep 2021	Oct 2020			Oct 2021	Sep 2021	Oct 2020
1.	1.	1.	Oracle	Relational, Multi-model	1270.35	-1.19	-98.42
2.	2.	2.	MySQL	Relational, Multi-model	1219.77	+7.24	-36.61
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	970.61	-0.24	-72.51
4.	4.	4.	PostgreSQL	Relational, Multi-model	586.97	+9.47	+44.57
5.	5.	5.	MongoDB	Document, Multi-model	493.55	-2.95	+45.53
6.	6.	↑ 8.	Redis	Key-value, Multi-model	171.35	-0.59	+18.07
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.96	-0.60	+4.06
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	158.25	-1.98	+4.41
9.	9.	9.	SQLite	Relational	129.37	+0.72	+3.95
10.	10.	10.	Cassandra	Wide column	119.28	+0.29	+0.18
11.	11.	11.	Microsoft Access	Relational	116.38	-0.56	-1.87
12.	12.	12.	MariaDB	Relational, Multi-model	102.59	+1.90	+10.82
13.	13.	13.	Splunk	Search engine	90.61	-0.99	+1.21
14.	14.	↑ 15.	Hive	Relational	84.74	-0.83	+15.19
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	79.72	+1.46	+15.32
16.	16.	16.	Amazon DynamoDB	Multi-model	76.55	-0.38	+8.14
17.	17.	↓ 14.	Teradata	Relational, Multi-model	69.83	+0.15	-5.96
18.	↑ 21.	↑ 64.	Snowflake	Relational	58.26	+6.19	+52.32
19.	↓ 18.	↑ 21.	Neo4j	Graph	57.87	+0.24	+6.53
20.	↓ 19.	↓ 19.	SAP HANA	Relational, Multi-model	55.28	-0.96	+1.04
21.	↓ 20.	↑ 23.	FileMaker	Relational	52.84	+0.52	+5.46
22.	22.	↓ 20.	Solr	Search engine, Multi-model	51.17	+1.36	-1.31
23.	23.	↓ 18.	SAP Adaptive Server	Relational, Multi-model	48.59	+1.57	-6.58
24.	24.	↓ 22.	HBase	Wide column	45.20	+0.14	-3.16
25.	25.	↓ 24.	Google BigQuery	Relational	43.79	-0.13	+9.38

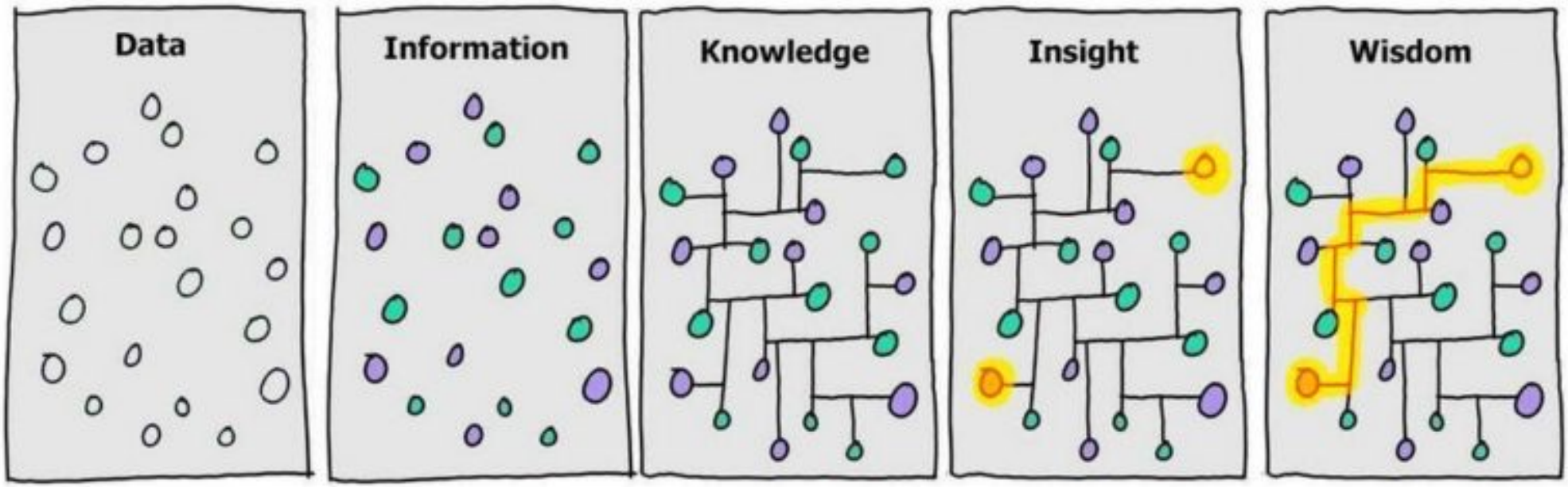
Topics

- ER Model
- Relational Model & Relational Constraint
- ER-Relational Mapping
- Relational Algebra & SQL
- Functional Dependency & Normalization

Midterm

- Primary File Organization
- Secondary File Organization (Indexing)
- Query Processing
- Query Optimization
- Performance Tuning
- Transaction Processing
- NoSQL (Distributed Database)

Data Management



Google核心使命

整理來自世界各地的資料，方便所有人擷取

Subject	Why study?	Best book	Best videos
Programming	Don't be the person who "never quite understood" something like recursion.	*Structure and Interpretation of Computer Programs*	Brian Harvey's Berkeley CS 61A
Computer Architecture	If you don't have a solid mental model of how a computer actually works, all of your higher-level abstractions will be brittle.	*Computer Organization and Design*	Berkeley CS 61C
Algorithms and Data Structures	If you don't know how to use ubiquitous data structures like stacks, queues, trees, and graphs, you won't be able to solve hard problems.	*The Algorithm Design Manual*	Steven Skiena's lectures
Math for CS	CS is basically a runaway branch of applied math, so learning math will give you a competitive advantage.	*Mathematics for Computer Science*	Tom Leighton's MIT 6.042J
Operating Systems	Most of the code you write is run by an operating system, so you should know how those interact.	*Operating Systems: Three Easy Pieces*	Berkeley CS 162
Computer Networking	The Internet turned out to be a big deal: understand how it works to unlock its full potential.	*Computer Networking: A Top-Down Approach*	Stanford CS 144
Databases	Data is at the heart of most significant programs, but few understand how database systems actually work.	*Readings in Database Systems*	Joe Hellerstein's Berkeley CS 186
Languages and Compilers	If you understand how languages and compilers actually work, you'll write better code and learn new languages more easily.	*Compilers: Principles, Techniques and Tools*	Alex Aiken's course on Lagunita
Distributed Systems	These days, *most* systems are distributed systems.	*Distributed Systems, 3rd Edition* by Maarten van Steen	□