Introduction to Data Structure (Data Management) Lecture 10

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  - University ID Num Name (no “(   )“)
  - Ex: 202054321 Juan Dela Cruz
  -
  - Not changing your name to this format
    - you might be marked Absent

    * → absent?

- NoSQL

- JSon and Semi-tructured Data

INTRO TO DATA STRUCTURE

NOSQL
(CH 11.1)

Motivation for NoSQL

Hello
hello 2!

- Motivated by Web 2.0 Applications
  - Web 2.0 allow anyone to create and share online information or material
  - Key element is allow people to create, share, collaborate & communicate
  - Hosted services (Google Maps), Web Apps (Google Docs, Flickr), vid sharing sites(YouTube), wikis, blogs, SNS(FB,IG), microblogging(Twitter)

Motivation for NoSQL

- Goal is to scale simple OLTP-style applications to millions or even billions of users
- OLTP (OnLine Transaction Processing)
  - capture, store, process data from transactions in real-time
  - typical size range from 100MB to 10GB
  - Ex: online banking, purchasing book online, booking ticket, send text message, call center staff view/update customer info

Motivation for NoSQL

- Facebook has 1.79B active users daily (Q2 2020)
  - use often correlated in time in each region
    - *correlated : one thing affects or depends on another*
  - more than 10M requests/sec if 25% users arrive w/in hour
  - SQL Server would crash under this workload
- Users doing both reads and updates

What is Problem?

- Single server DBMS too small for Web data

What is Problem?

- Single server DBMS too small for Web data
  - Solution→ scale out to multiple servers
    - *scale: resize a device, object or system*
    - *"scale up" or "scale vertically: expanding capability of a machine*
    - *"scale out" or ""scale horizontally": add more machines*

What is Problem?

- Single server DBMS too small for Web data
  - Solution→ scale out to multiple servers
    - *scale: resize a device, object or system*
    - *"scale up" or "scale vertically: expanding capability of a machine*
    - *"scale out" or ""scale horizontally": add more machines*
- This is hard for ***entire*** functionality of DBMS

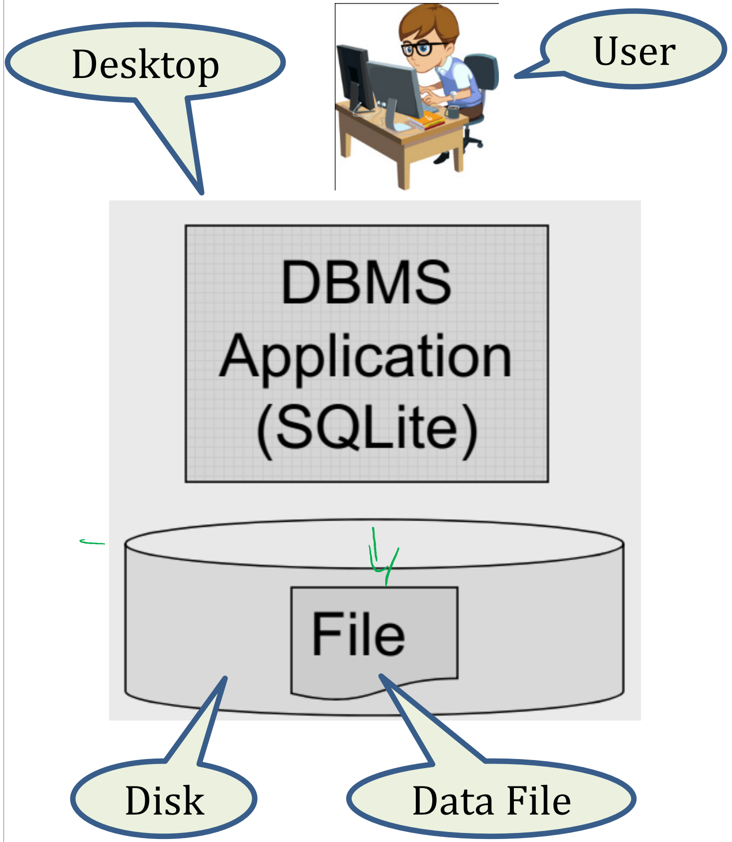
What is Problem?

- Single server DBMS too small for Web data
  - Solution→ scale out to multiple servers
    - *scale: resize a device, object or system*
    - *"scale up" or "scale vertically: expanding capability of a machine*
    - *"scale out" or ""scale horizontally": add more machines*
- This is hard for ***entire*** functionality of DBMS
- NoSQL: reduce functionality for easier scaling

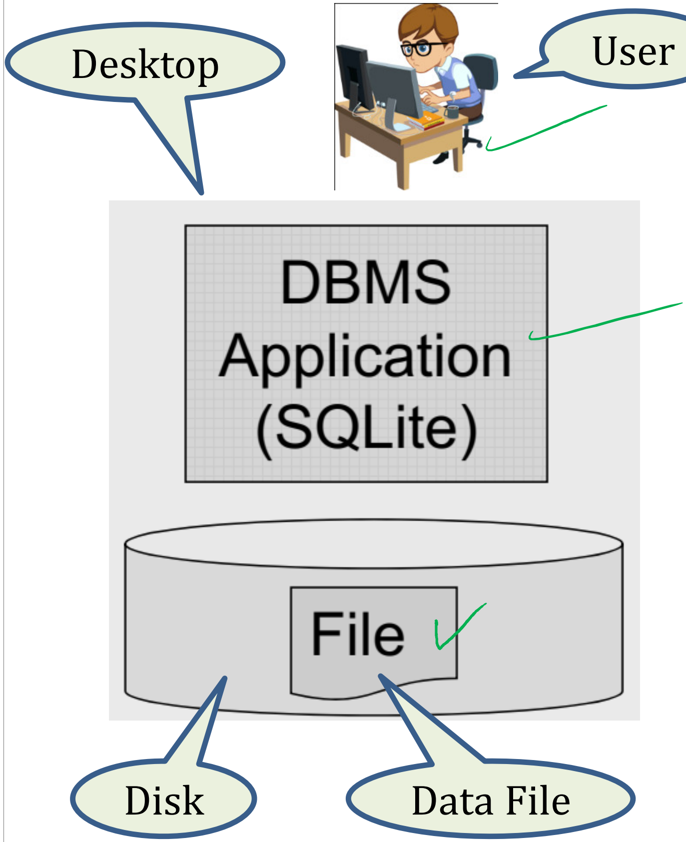
What is Problem?

- Single server DBMS too small for Web data
  - Solution→ scale out to multiple servers
    - *scale: resize a device, object or system*
    - *“scale up” or “scale vertically: expanding capability of a machine*
    - *“scale out” or ”“scale horizontally”: add more machines*
- This is hard for ***entire*** functionality of DBMS
- NoSQL: reduce functionality for easier scaling
  - simpler data model
  - fewer guarantees

Serverless Architecture



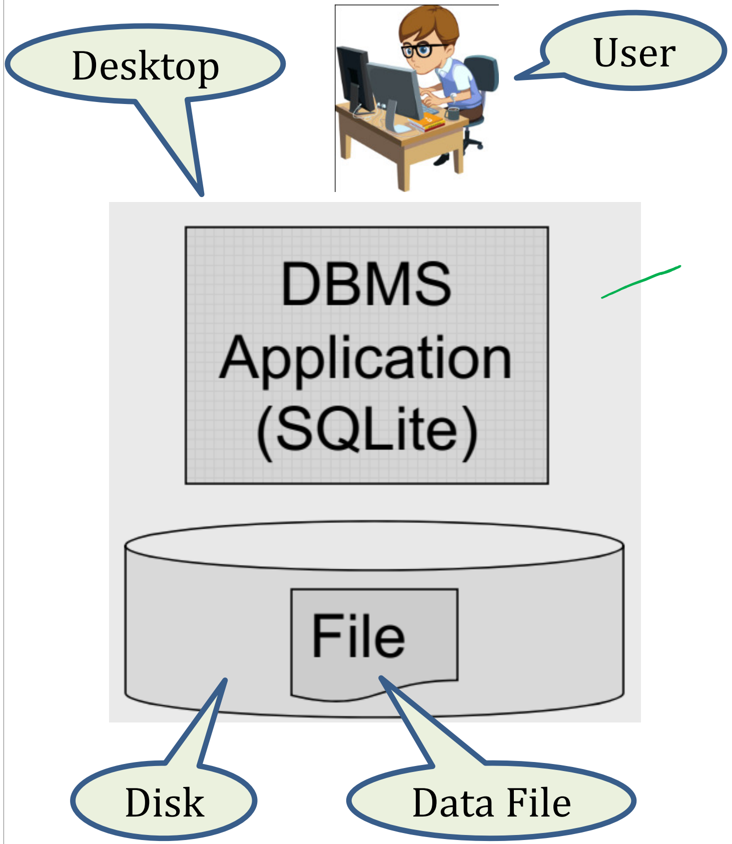
Serverless Architecture



SQLite

- One data file
- One user
- One DBMS application

Serverless Architecture

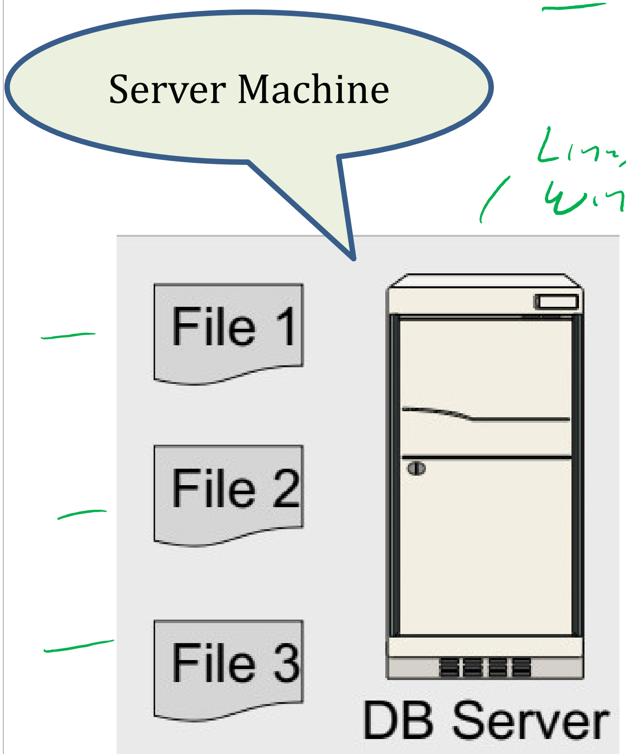


SQLite

- One data file
- One user
- One DBMS application

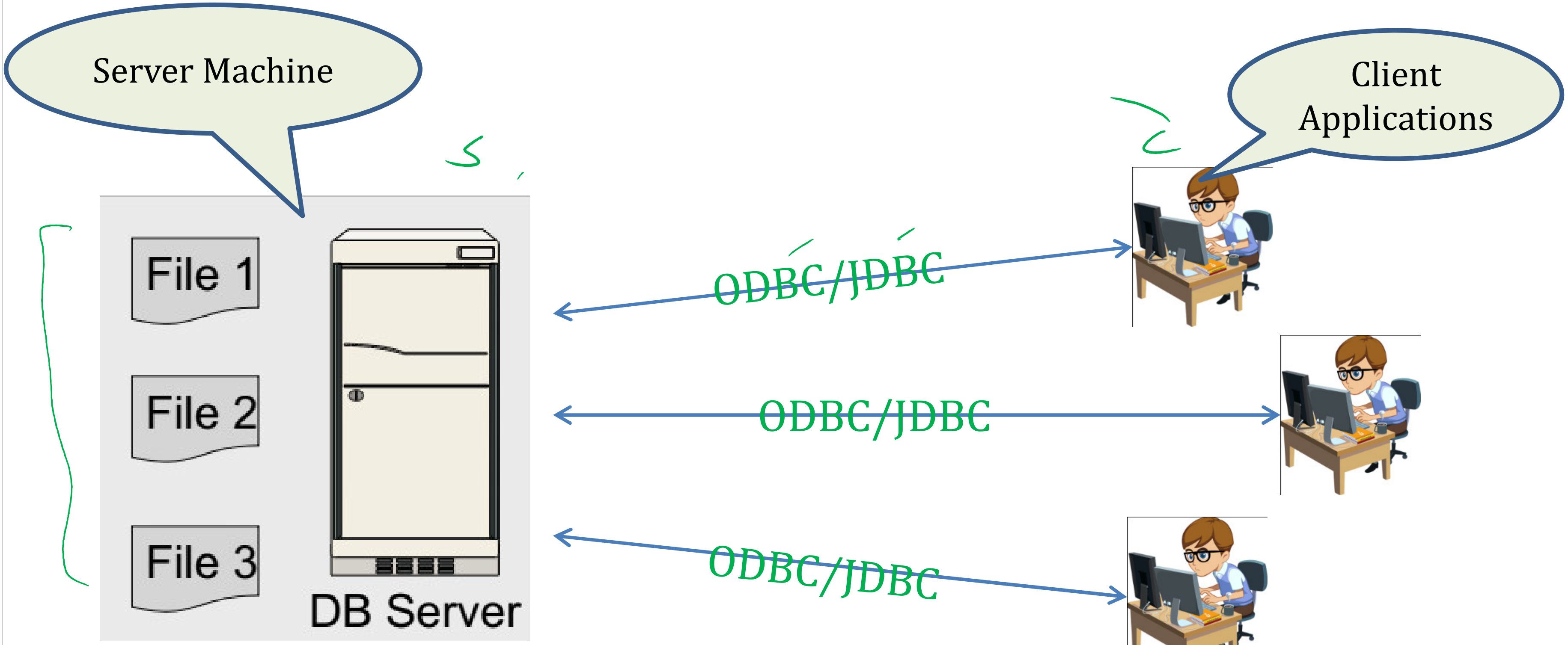
- Scales well
- But only a limited number of scenarios work with such model
- Can be in browser/ phone

Client-Server Architecture



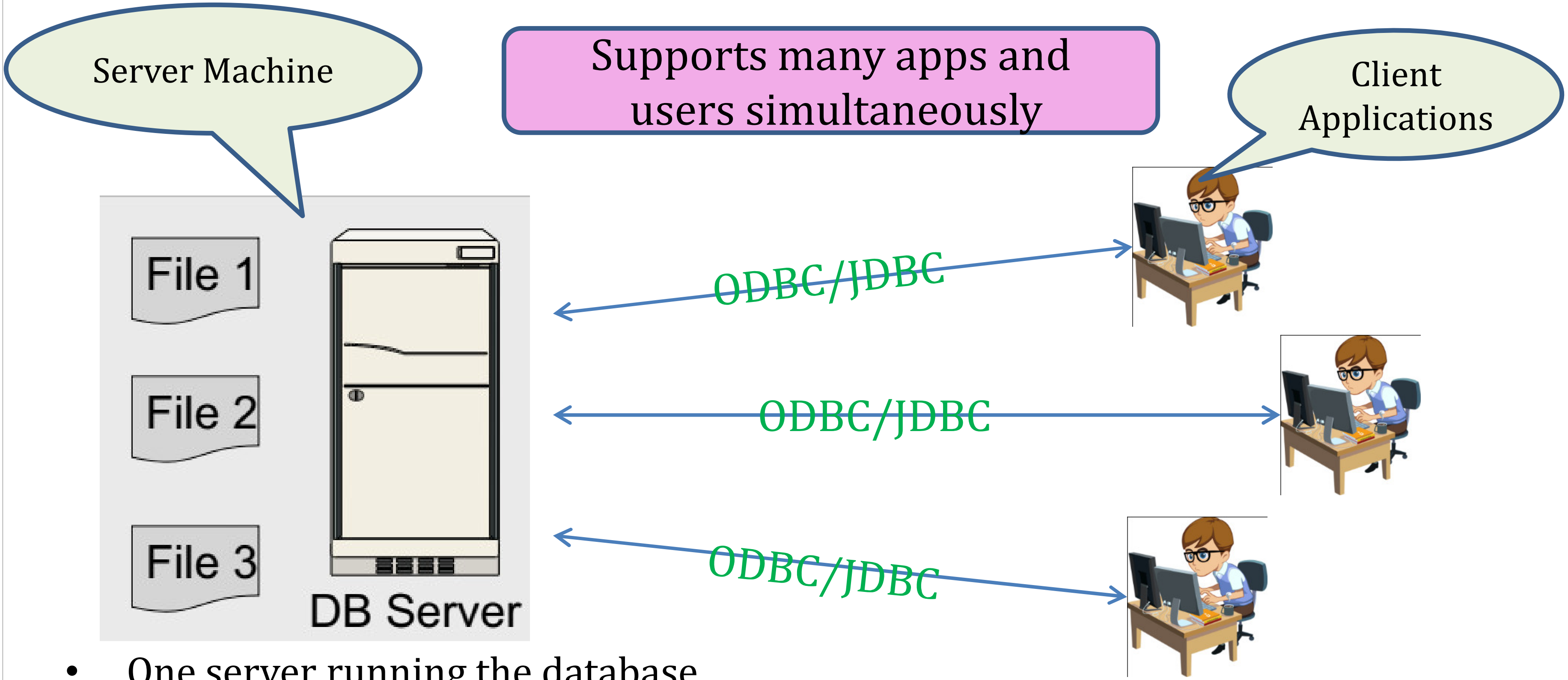
Linux
Win
Unix

Client-Server Architecture



- One server running the database
- Many clients, connecting via ODBC (Open DB Connectivity / JDBC (Java DB Connectivity

Client-Server Architecture



- One server running the database
- Many clients, connecting via ODBC (Open DB Connectivity / JDBC (Java DB Connectivity

Client-Server

- One *server* runs DBMS (or RDBMS)
  - on own desktop
  - some beefy/powerful system
  - cloud service

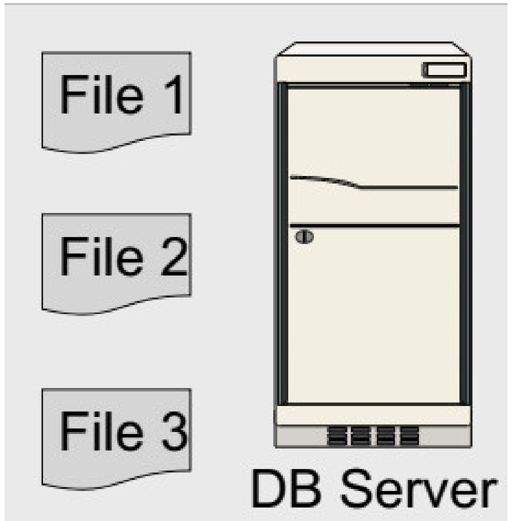
Client-Server

- One *server* runs DBMS (or RDBMS)
  - on own desktop
  - some beefy/powerful system
  - cloud service
- Many *clients* run apps and connect to DBMS
  - MS Management Studio (for SQL Server) or
  - pSQL (for postgres)
  - some Java program or C++ program

Client-Server

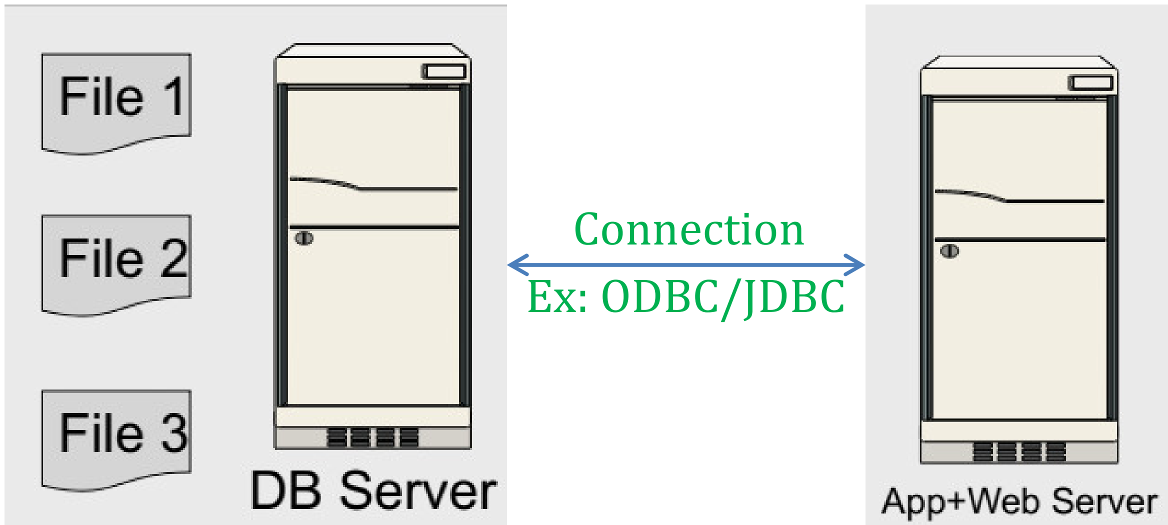
- One *server* runs DBMS (or RDBMS)
  - on own desktop
  - some beefy/powerful system
  - cloud service
- Many *clients* run apps and connect to DBMS
  - MS Management Studio (for SQL Server) or
  - pSQL (for postgres)
  - some Java program or C++ program
- Clients "talk" to server using ODBC/JDBC protocol

3-Tiered Architecture



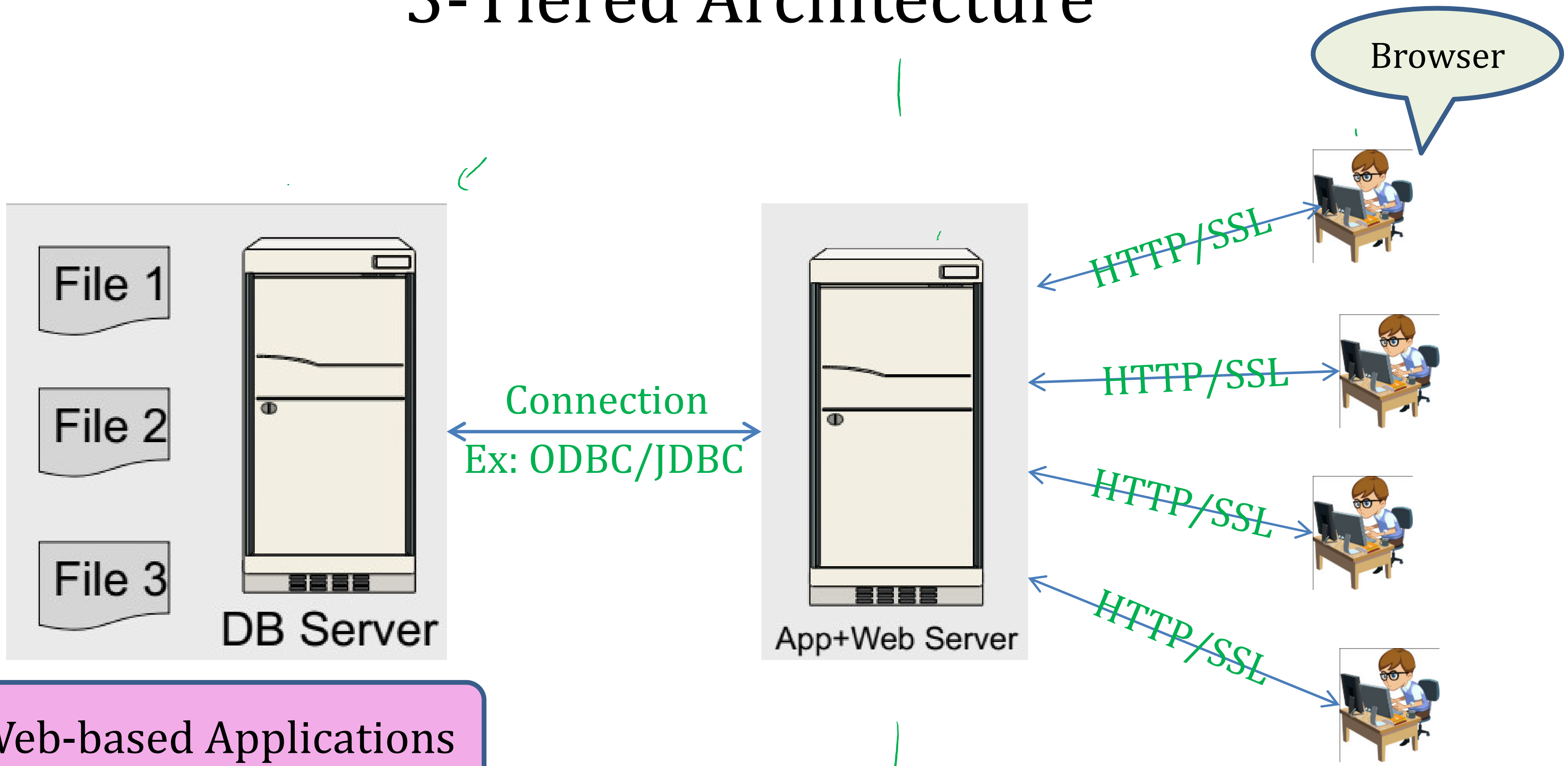
Web-based Applications

3-Tiered Architecture



Web-based Applications

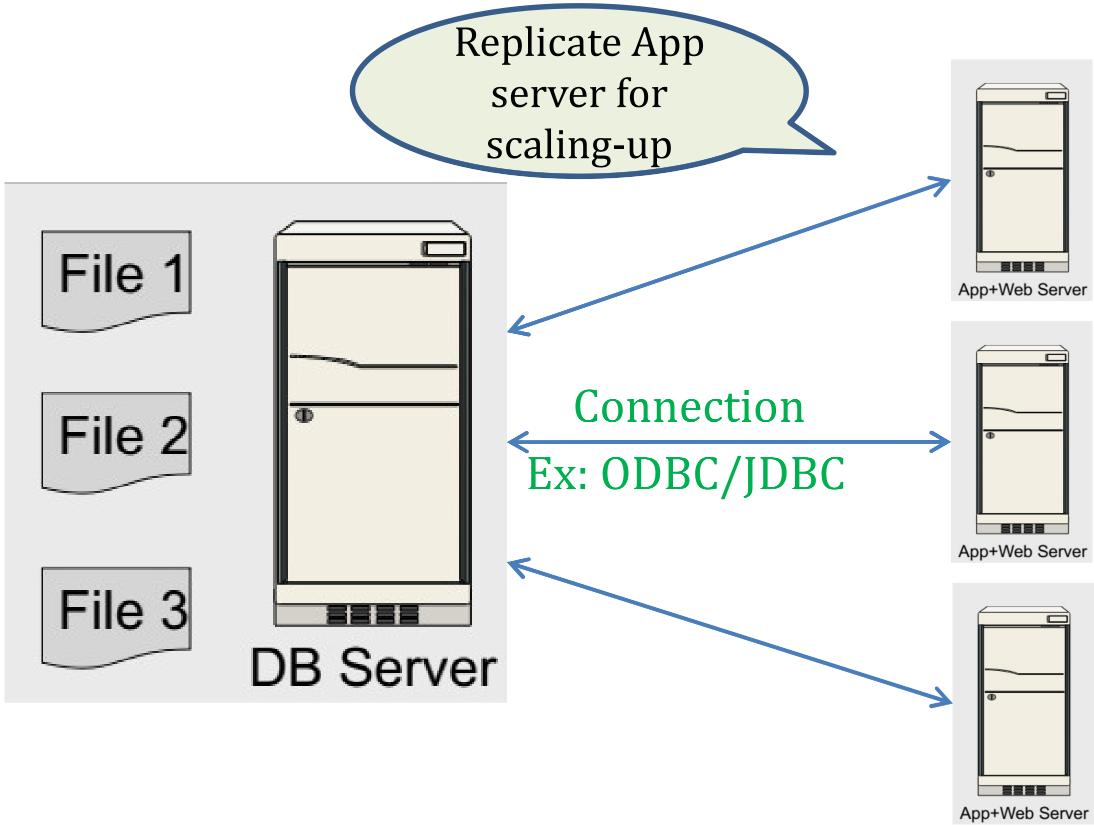
3-Tiered Architecture



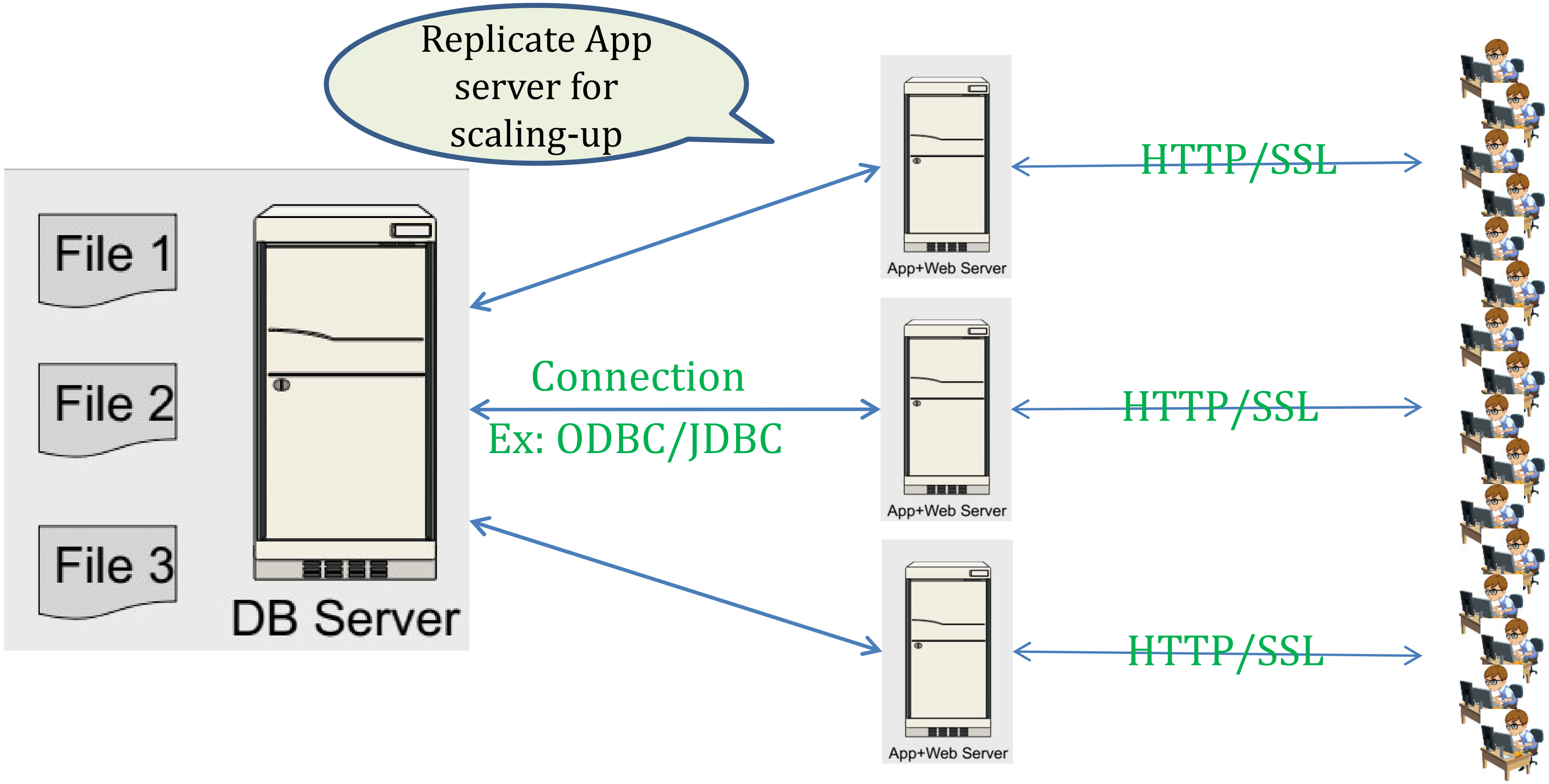
Web-based Applications

* HTTP s = Hyper Text Transfer Protocol Secure
* SSL = Secure Socket Layer

3-Tiered Architecture

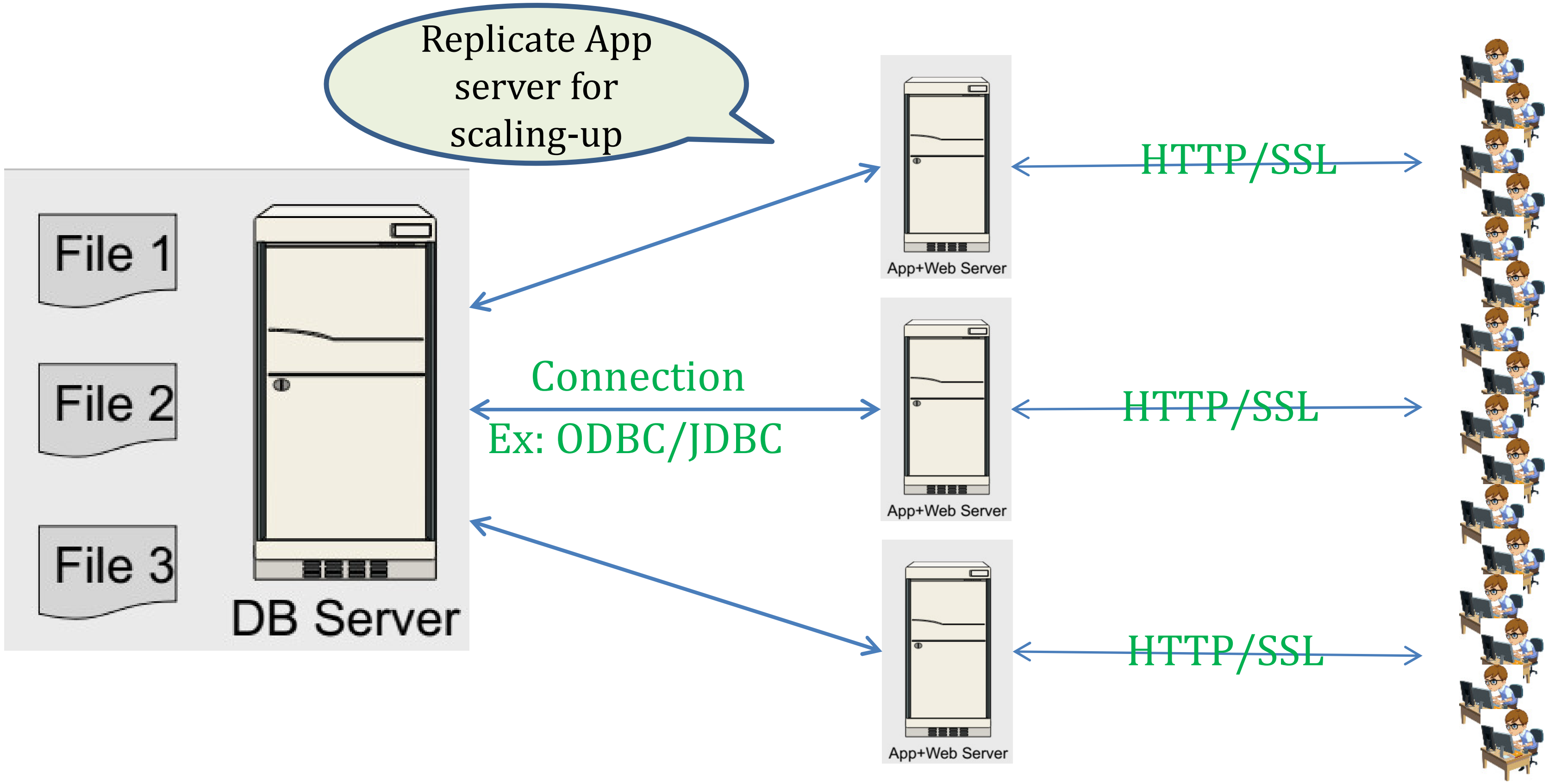


3-Tiered Architecture



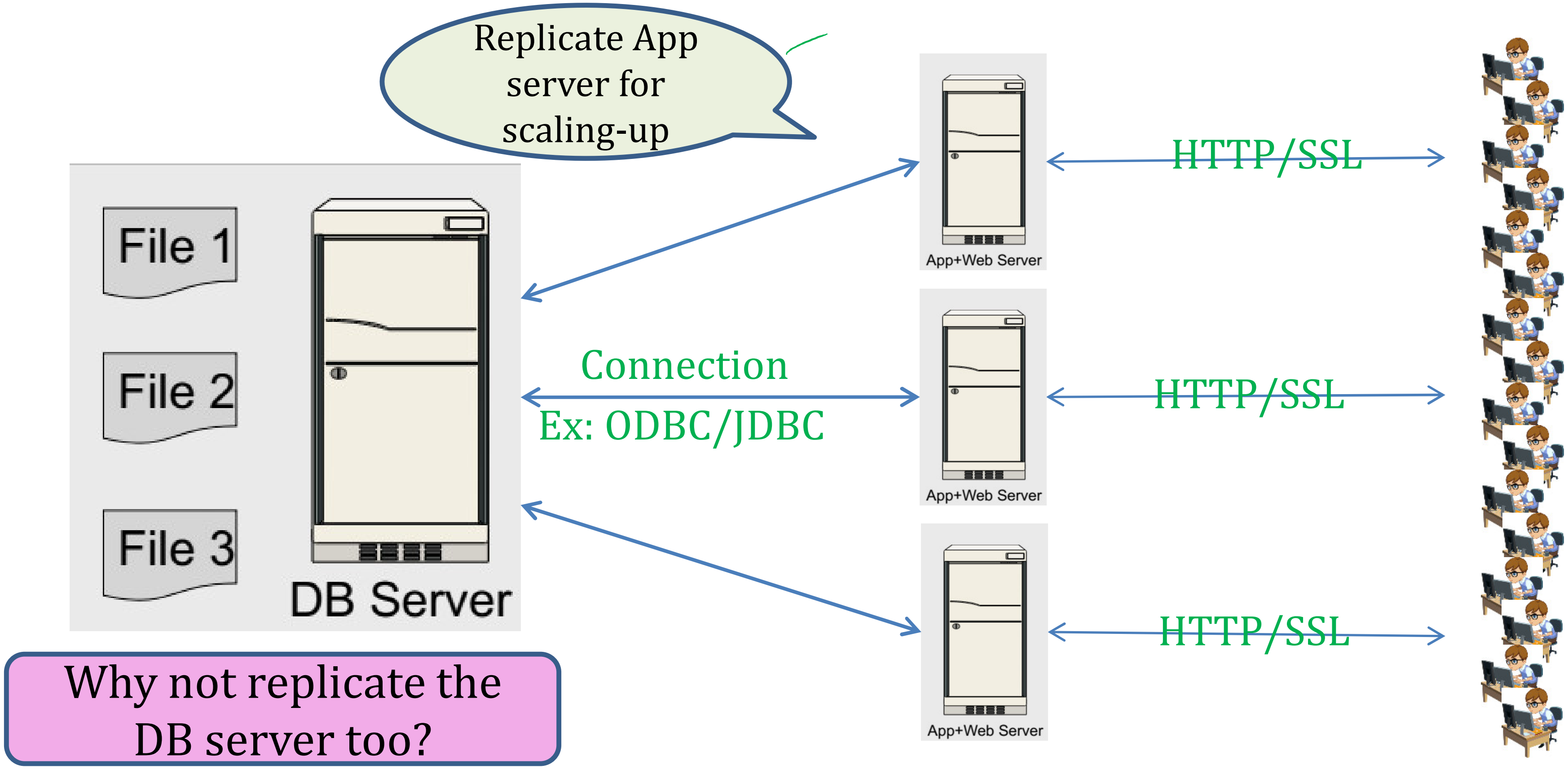
* HTTP = Hyper Text Transfer Protocol
* SSL = Secure Socket Layer

3-Tiered Architecture



* HTTP = Hyper Text Transfer Protocol
* SSL = Secure Socket Layer

3-Tiered Architecture



* HTTP = Hyper Text Transfer Protocol
* SSL = Secure Socket Layer

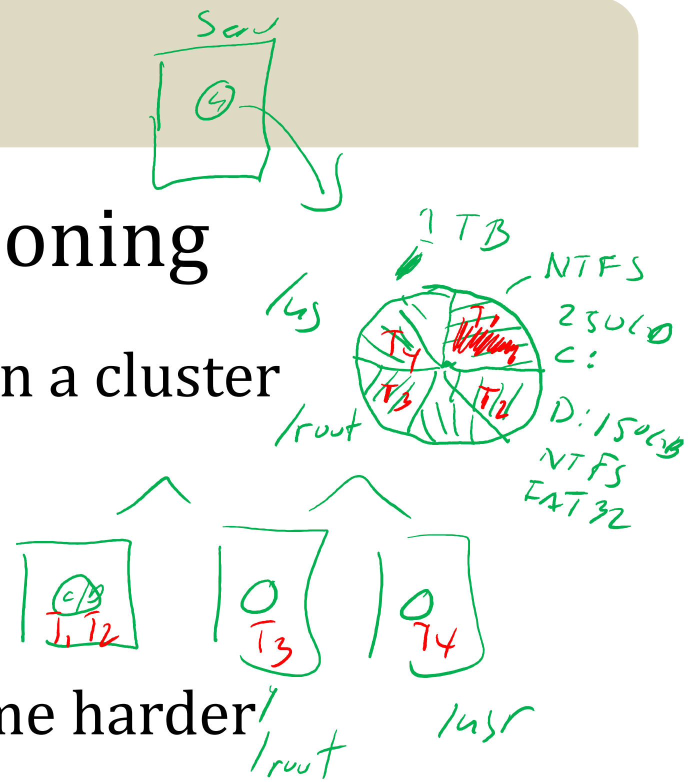
Replicating the Database

- Much harder because the state must be unique. In other words, database must act as a whole
  - Current DB instance must always be ***consistent***
    - Ex: Foreign keys must exist
    - as a result, some **updates** must occur simultaneously

Replicating the Database

- Much harder because the state must be unique. In other words, database must act as a whole
  - Current DB instance must always be ***consistent***
    - Ex: Foreign keys must exist
    - as a result, some **updates** must occur simultaneously
- Two basic approach:
  - Scale up by partitioning
  - Scale up by replication

Scale Through Partitioning



- Partition the DB across many machines in a cluster
  - Database could fit in main memory
  - Queries spread across these machines
- Can increase throughput
- Easy for (simple) writes but reads become harder

* throughput = amount of material/items passing through a system or process

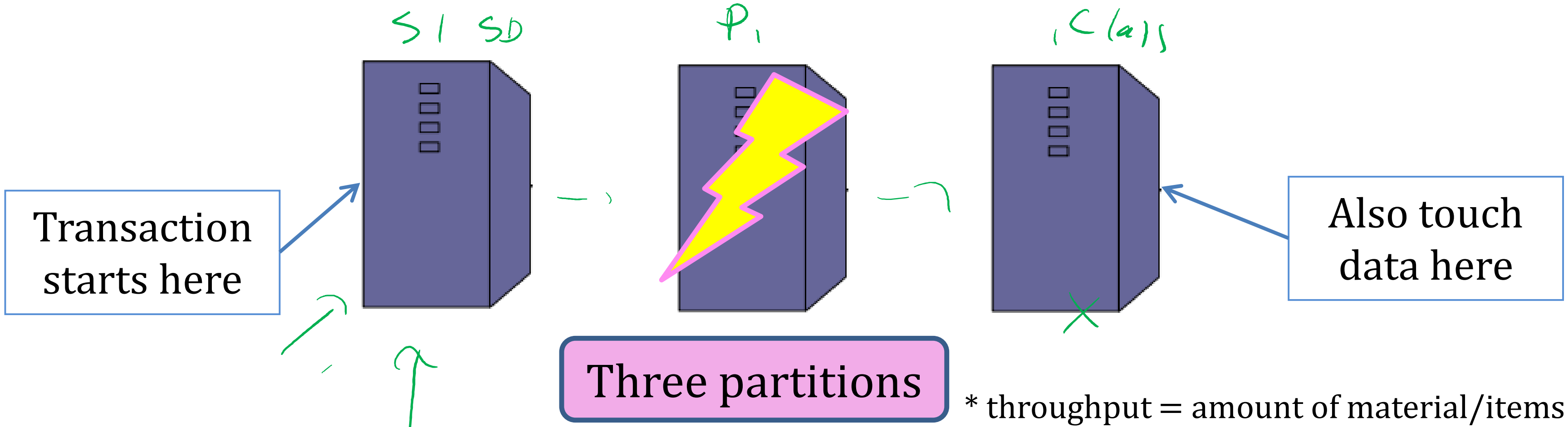
Scale Through Partitioning

- Partition the DB across many machines in a cluster
  - Database could fit in main memory
  - Queries spread across these machines
- Can increase throughput
- Easy for (simple) writes but reads become harder

* throughput = amount of material/items passing through a system or process

Scale Through Partitioning

- Partition the DB across many machines in a cluster
  - Database could fit in main memory
  - Queries spread across these machines
- Can increase throughput
- Easy for (simple) writes but reads become harder



* throughput = amount of material/items passing through a system or process

Scale Through Replication

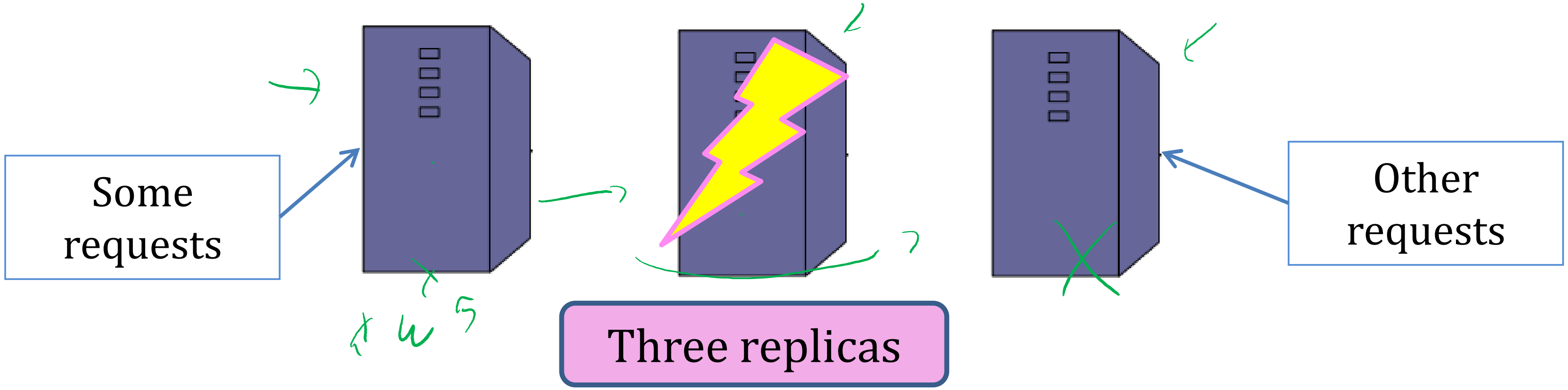
- Create multiple copies of each database partition
- Spread queries across these replicas
- Can increase throughput and lower latency
- Can also improve fault-tolerance
- Easy for reads but writes become harder

* latency = delay before transfer of data starts after instruction for its transfer
* fault-tolerance = ability of system to continue operating w/o interruption when one or more components fail

Scale Through Replication

- Create multiple copies of each database partition
- Spread queries across these replicas
- Can increase throughput and lower latency
- Can also improve fault-tolerance
- Easy for reads but writes become harder

* latency = delay before transfer of data starts after instruction for its transfer
* fault-tolerance = ability of system to continue operating w/o interruption when one or more components fail



NoSQL Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort(LinkedIn), Memcached

* taxonomy = science of categorization or classification of things based on a predetermined system

NoSQL Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort(LinkedIn), Memcached
- Document stores
  - Ex.: SimpleDB, CouchDB, MongoDB

* taxonomy = science of categorization or classification of things based on a predetermined system

NoSQL Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort(LinkedIn), Memcached
- Document stores
  - Ex.: SimpleDB, CouchDB, MongoDB
- Extensible Record stores
  - Ex.: Hbase, Cassandra, PNUTS

* taxonomy = science of categorization or classification of things based on a predetermined system

NoSQL Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort(LinkedIn), Memcached
- Document stores
  - Ex.: SimpleDB, CouchDB, MongoDB
- Extensible Record stores
  - Ex.: Hbase, Cassandra, PNUTS

* taxonomy = science of categorization or classification of things based on a predetermined system

Key-Value Stores Features

- Data Model: (key, value) pairs
  - Key = string/integer, unique for entire data
  - Value = can be anything (very complex object)

Key-Value Stores Features

- Data Model: (key, value) pairs
  - Key = string/integer, unique for entire data
  - Value = can be anything (very complex object)
- Operations
  - Get(key), Put(key, value)
  - Operations on value not supported

Key-Value Stores Features

- Data Model: (key, value) pairs
  - Key = string/integer, unique for entire data
  - Value = can be anything (very complex object)
- Operations
  - Get(key), Put(key, value)
  - Operations on value not supported
- Distribution/ Partitioning
  - No replication: key k is stored at server h(k)
  - 3-way replication: key is stored at h1(k), h2(k), h3(k)

How does get(k) work? How does put(k,v) work?

- `Flights(fid, date, carrier,`
  `flight_num, origin, dest, …)`
- `Carriers(cid, name)`

Example

How would you represent the Flights data as (key, value) pairs

```
Flights(fid, date, carrier,
        flight_num, origin, dest, …)
Carriers(cid, name)
```

Example

How would you represent the Flights data as (key, value) pairs

date#c ..a # f num#...

- Option 1: key=fid, value=entire flight record

```
Flights(fid, date, carrier,
        flight_num, origin, dest, …)
Carriers(cid, name)
```

Example

How would you represent the Flights data as (key, value) pairs

- Option 1: key=fid, value=entire flight record

datetime [ddmmyyhhmmss]

- Option 2: key=date, value=all flights that day

```
Flights(fid, date, carrier,
        flight_num, origin, dest, …)
Carriers(cid, name)
```

Example

How would you represent the Flights data as (key, value) pairs

- Option 1: key=fid, value=entire flight record
- Option 2: key=date, value=all flights that day
- Option 3: key=(origin, dest), value=all flights between

inc hwi

How does query processing work?

Key-Value Stores Internals

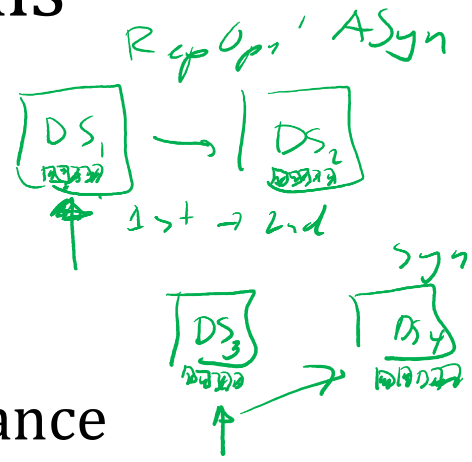
- Data remains in main memory
  - One implementation: distributed hash table

Key-Value Stores Internals

- Data remains in main memory
  - One implementation: distributed hash table
- Most systems also offer a persistence option

* persistence = data survives after process it was created has ended

Key-Value Stores Internals



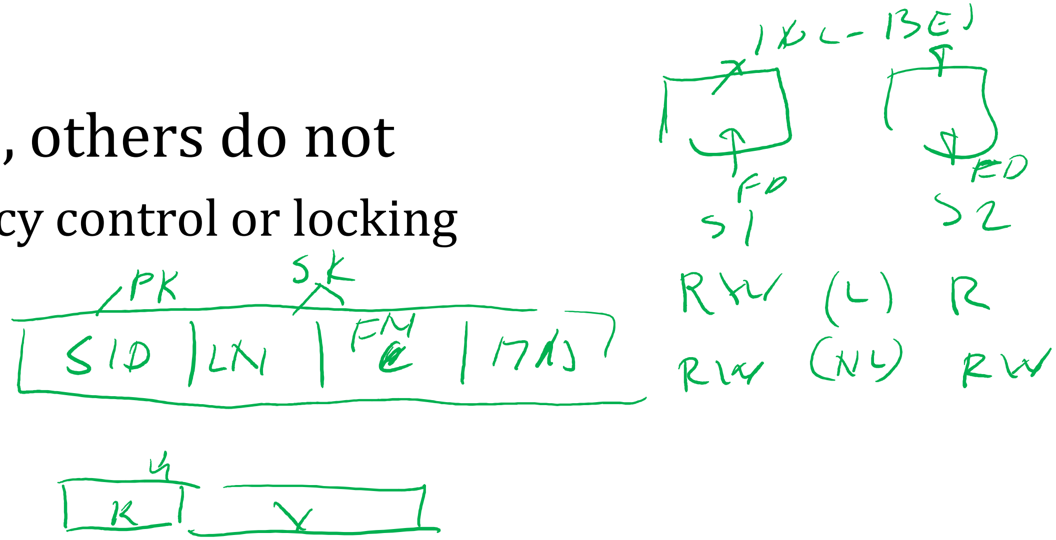
- Data remains in main memory
  - One implementation: distributed hash table
- Most systems also offer a persistence option
- Others use replication to provide fault-tolerance
  - Asynchronous replication: copy data to the replica after the data is already written to the primary storage
  - Synchronous replication: write data to primary storage and the replica simultaneously
  - Tunable consistency: read/write one replica or majority

Tune

* persistence = data survives after process it was created has ended
* replica = exact copy of a database or other data store

Key-Value Stores Internals

- Some offer transactions, others do not
  - Multi-version concurrency control or locking
- No secondary indices



Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort(LinkedIn), Memcached
- Document stores
  - Ex.: SimpleDB, CouchDB, MongoDB
- Extensible Record stores
  - Ex.: Hbase, Cassandra, PNUTS

* taxonomy = science of categorization or classification of things based on a predetermined system

Document Stores Features

- Data Model: (key, document) pairs
  - Key = string/integer, unique for entire data
  - Document = JSON or XML

Document Stores Features

- Data Model: (key, document) pairs
  - Key = string/integer, unique for entire data
  - Document = JSON or XML

K V
Get(k), P(k,v)

- Operations
  - Get/Put document by key → Get(k,d) P(k,d)
  - Limited, non-standard query language on JSON

Document Stores Features

- Data Model: (key, document) pairs
  - Key = string/integer, unique for entire data
  - Document = JSON or XML
- Operations
  - Get/Put document by key
  - Limited, non-standard query language on JSON
- Distribution/ Partitioning
  - Entire documents, as for key/value pairs

Will discuss JSon next time

Data Models

Taxonomy based on data models

- Key-value stores
  - Ex.: Project Voldemort, Memcached
- Document stores
  - Ex.: SimpleDB, CouchDB, MongoDB
- Extensible Record stores
  - Ex.: Hbase, Cassandra, PNUTS

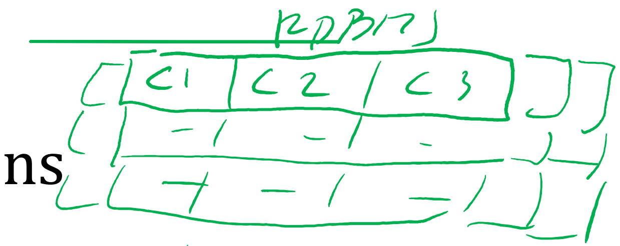
* taxonomy = science of categorization or classification of things based on a predetermined system

Extensible Record Stores

- Based on Google's BigTable (data storage system)
  - HBase is an open source implementation of BigTable

Extensible Record Stores

- Based on Google's BigTable (data storage system)
  - HBase is an open source implementation of BigTable
- Data model is rows and columns
  - Can add both new rows and new columns

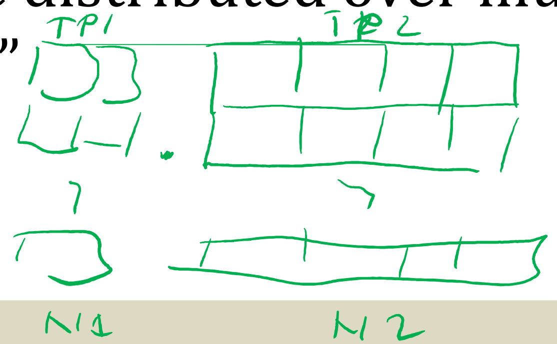


Long Schema

C1 : INT PK
C2 : vchar(2)

Extensible Record Stores

- Based on Google's BigTable (data storage system)
  - HBase is an open source implementation of BigTable
- Data model is rows and columns
  - Can add both new rows and new columns
- Scalability by splitting rows & columns over nodes
  - Rows partitioned through hashing on primary key
  - Columns of a table are distributed over multiple nodes using "column groups"



NoSQL Summary

- Simple data model with weaker guarantees
- But they scale as far as needed

NoSQL Summary

- Simple data model with weaker guarantees
- But they scale as far as needed

- Meanwhile…

  SQL systems continue to improve

Recent SQL Progress

- Modern systems need to store data across the globe
  - Individual data centers go offline
  - Need servers close to users to be efficient

Recent SQL Progress

- Modern systems need to store data across the globe
  - Individual data centers go offline
  - Need servers close to users to be efficient
- Speed of light is fundamental limit
  - 200+ms latency (across US) visible to users
  - 40ms latency (South Korea 4G networks)

Recent SQL Progress

- Modern systems need to store data across the globe
  - Individual data centers go offline
  - Need servers close to users to be efficient
- Speed of light is fundamental limit
  - 200+ms latency (across US) visible to users
  - 40ms latency (South Korea 4G networks)
- Systems must weaken guarantees

Recent SQL Progress

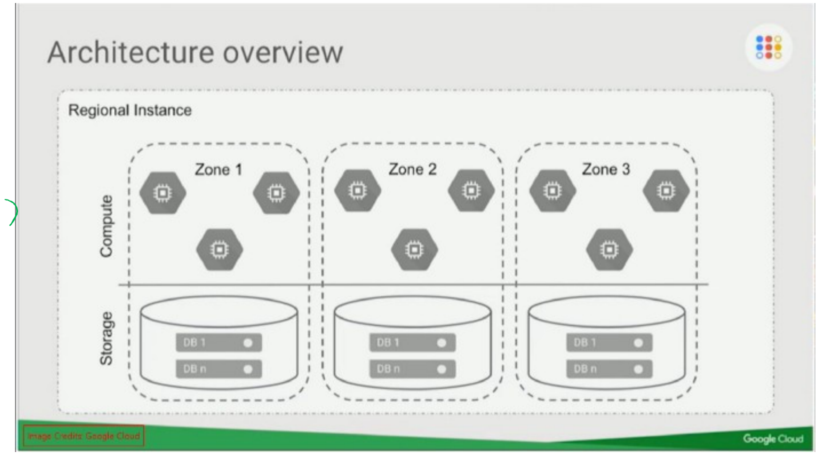
- Modern systems need to store data across the globe
  - Individual data centers go offline
  - Need servers close to users to be efficient
- Speed of light is fundamental limit
  - 200+ms latency (across US) visible to users
  - 40ms latency (South Korea 4G networks)
- Systems must weaken guarantees
- Google Spanner (support SQL)
  - Write data over whole globe (a bit slowly)
  - Reads occur slightly in the past

Prediction

- Best guess is SQL will win

Prediction

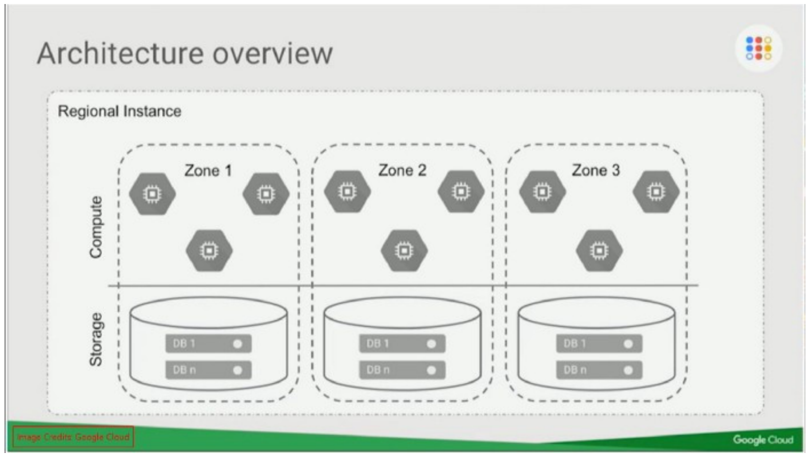
- Best guess is SQL will win
- Pieces are out there already
  - Spanner: multi-node transactions



  - AsterixDB: multi-node query optimization

Prediction

- Best guess is SQL will win
- Pieces are out there already
  - Spanner: multi-node transactions



  - AsterixDB: multi-node query optimization
- For now, NoSQL still offers key benefits

INTRO TO DATA STRUCTURE

JSON

Where Are We

- So far, we have studied relational data model
  - Data are stored in tables (relations)
  - Queries are expressions in the SQL/ Datalog/ Relational Algebra

Where Are We

- So far, we have studied relational data model
  - Data are stored in tables (relations)
  - Queries are expressions in the SQL/ Datalog/ Relational Algebra

- Today: Semi-structured data model
  - Popular formats: XML, JSon, protobuf

Semi-structured Data

- Database of semi-structured data
  - Collection of nodes(leaf, interior)

Semi-structured Data

- Database of semi-structured data
  - Collection of nodes(leaf, interior)
- Leaf node
  - Associated data; any atomic type: integer, string, etc

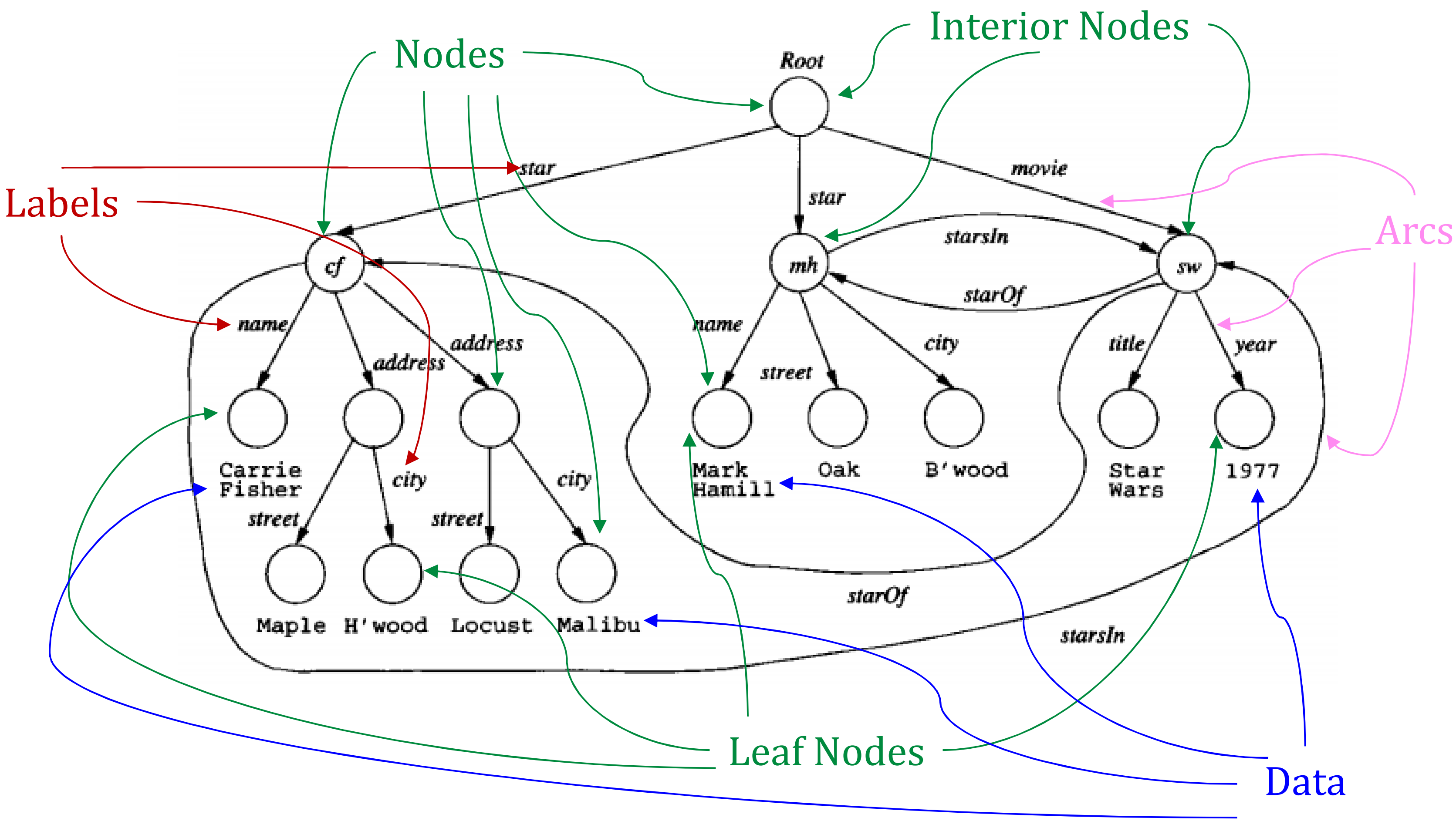
Semi-structured Data

- Database of semi-structured data
  - Collection of nodes(leaf, interior)
- Leaf node
  - Associated data; any atomic type: integer, string, etc
- Interior node
  - One or more arcs going out, each arc has label

Semi-structured Data

- Database of semi-structured data
  - Collection of nodes(leaf, interior)
- Leaf node
  - Associated data; any atomic type: integer, string, etc
- Interior node
  - One or more arcs going out, each arc has label
- Root node (an interior node)
  - No arc entering, represent entire DB
  - Every node must be reachable from root node

Semi-structured Data



JSON

- 13 years ago
  - Java interpreters were very slow
  - Native browser function parsed JSON 100x faster

* parse = break up a sentence or group of words into separate components, including the definition of each part's function or form

JSON

- 13 years ago
  - Java interpreters were very slow
  - Native browser function parsed JSON 100x faster

- XML was also an option, but
  - IE had memory leak in its XML parser

* parse = break up a sentence or group of words into separate components, including the definition of each part's function or form

JSON

- 13 years ago
  - Java interpreters were very slow
  - Native browser function parsed JSON 100x faster

- XML was also an option, but
  - IE had memory leak in its XML parser

- JSON used in Gmail etc. for this reason
- Spread organically to server-side systems

* parse = break up a sentence or group of words into separate components, including the definition of each part's function or form

Overview of JSON

- JavaScript Object Notation
  - Lightweight text-based open standard designed for human-readable data interchange.
  - Interfaces in C, C++, Java, Python, Perl, etc.

Overview of JSON

- JavaScript Object Notation
  – Lightweight text-based open standard designed for human-readable data interchange.
  – Interfaces in C, C++, Java, Python, Perl, etc.

- The filename extension is .json

We will emphasize JSon as semi-structured data

JSon vs Relational

- Relational data model
  - Rigid flat structure (tables)
  - Schema must be fixed in advanced
  - Binary representation
    - *good for performance, bad for exchange*

JSon vs Relational

- Relational data model
  - Rigid flat structure (tables)
  - Schema must be fixed in advanced
  - Binary representation
    - *good for performance, bad for exchange*
- Semi-structured data model/JSon
  - Flexible, nested structure (trees)
  - Does not require predefined schema ("self-describing")
  - Text representation
    - *bad for performance, good for exchange*
  - Most common use
    - *language API; query languages emerging*

JSon Syntax

```
{ "book": [
    {"id": "01",
     "language": "Java",
     "author": "H. Javeson",
     "year": 2015
    },
    {"id": "07",
     "language": "C++",
     "edition": "second",
     "author": "E. Sepp",
     "price": 22.25
    }
  ]
}
```

JSon Terminology

- Curly braces “{ }”hold objects
  - Each object is a list of name/value pairs separated by a comma “,”
  - Each pair is a name followed by a colon “:”, and followed by the value

JSon Terminology

- Curly braces “{ }”hold objects
  - Each object is a list of name/value pairs separated by a comma “,”
  - Each pair is a name followed by a colon “:”, and followed by the value
- Square brackets “[ ]” holds array and values are separated by comma “,”

JSon Terminology

- Curly braces "{ }"hold objects
  - Each object is a list of name/value pairs separated by a comma ","
  - Each pair is a name followed by a colon ":", and followed by the value
- Square brackets "[ ]" holds array and values are separated by comma ","
- Data made up of objects, lists, and atomic values (integers, floats, strings, booleans)

JSon Data Structures

- Collections of name-value pairs:
  - {“name1”:value1, “name2”:value2, …}
  - The “name” is also called a “key”

JSon Data Structures

- Collections of name-value pairs:
  - {"name1":value1, "name2":value2, ...}
  - The "name" is also called a "key"
- Ordered lists of values:
  - [obj1, obj2, obj3, ...]

Avoid Using Duplicate Keys

The standard allows them, but many implementations doesn't

```
{"id": "07",
 "title": "Databases",
 "author": "Garcia-Molina",
 "author": "Ullman",
 "author": "Widom",
}
```

Avoid Using Duplicate Keys

The standard allows them, but many implementations doesn't

```
{"id": "07",
 "title": "Databases",
 "author": "Garcia-Molina",
 "author": "Ullman",
 "author": "Widom",
}
```

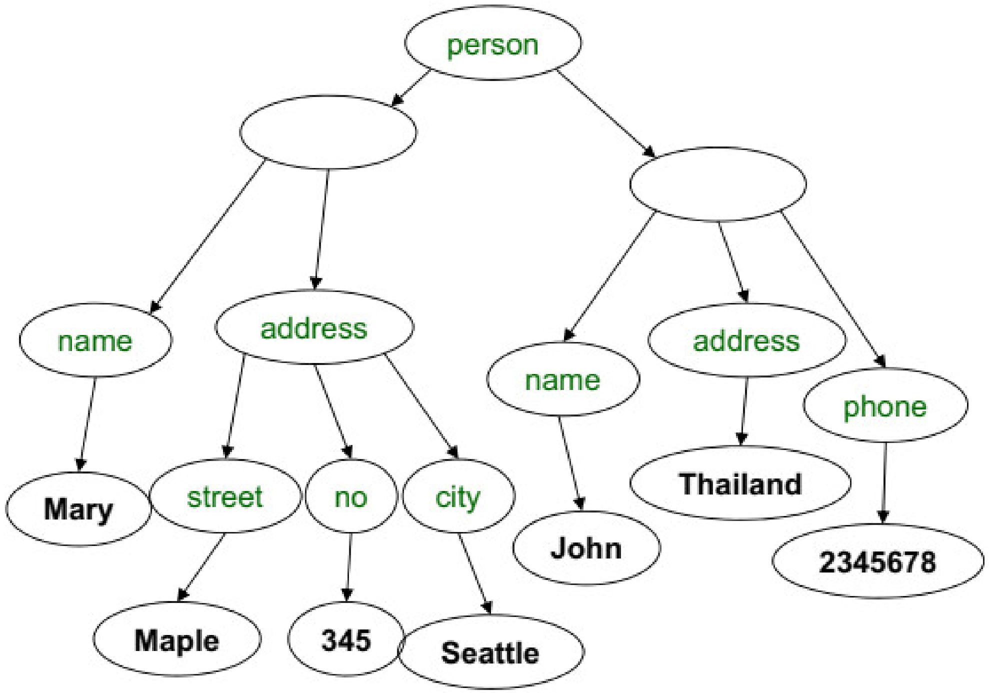
```
{"id": "07",
 "title": "Databases",
 "author": ["Garcia-Molina",
            "Ullman",
            "Widom"]
}
```

JSon Data Types

- Number
- String = double-quoted
- Boolean = True or False
- Null/empty

JSon Semantics

```
{“person”:
  [{“name”:”Mary”,
    “address”:
    {“street”:”Maple”,
     “no”:345,
     “city”:”Seattle”}},
   {“name”:”John”,
    “address”:”Thailand”,
    “phone”: 2345678}
  ]
}
```



JSon Data

- JSon is self-describing

JSon Data

- JSon is self-describing
- Schema elements become part of the data
  - Relational schema: person(name, phone)
  - In Json: “person”, “name”, “phone” are part of the data, are are repeated many times

JSon Data

- JSon is self-describing
- Schema elements become part of the data
  - Relational schema: person(name, phone)
  - In Json: “person”, “name”, “phone” are part of the data, are are repeated many times
- Consequence: JSon is much more flexible
  - also uses more space (but can be compressed)

JSon Data

- JSon is self-describing
- Schema elements become part of the data
  - Relational schema: person(name, phone)
  - In Json: "person", "name", "phone" are part of the data, are are repeated many times
- Consequence: JSon is much more flexible
  - also uses more space (but can be compressed)
- JSon is an example of semi-structured data

Thank you.