Cloud Computing
CS 15-319

Apache Pig, Hive and Zookeeper

Lecture 16, Mar 14, 2012

Majd F. Sakr, Mohammad Hammoud and
Suhail Rehman

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Today…

- Last session
  - BigTable Video Lecture and Discussion
- Today's session
  - Apache Pig, Hive and Zookeeper
- Announcement:
  - Project update is due today

Going beyond MapReduce…

- MapReduce provides a simple abstraction to write distributed programs running on large-scale systems on large amounts of data

- MapReduce is not suitable for everyone
  - MapReduce abstraction is low-level and developers need to write custom programs which are hard to maintain and reuse

- Sometimes user requirements may differ:
  - Interactive processing of large log files
  - Process big data using SQL syntax rather than Java programs
  - Warehouse large amounts of data while enabling transactions and queries
  - Write a custom distributed application but don't want manage distributed synchronization and co-ordination

Unstructured vs. Structured Data

- Structured Data
  - Data with a corresponding data model, such as a schema
  - Fits well in relational tables
  - E.g. Data in an RDBMS
- Unstructured Data
  - No data model, schema
  - Textual or bit-mapped (pictures, audio, video etc.)
  - E.g. Log Files, E-mails etc.

schema →

Email ID	First Name	Class	Major
johndoe@cmu.edu	“John”	2014	CS
janedoe@cmu.edu	“Jane”	2013	IS

Relational Database Table

```
123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET /pics/wpaper.gif HTTP/1.0" 200 6248 "http://www.jafsoft.com/asctortf/" "Mozilla/4.05 (Macintosh; I; PPC)"

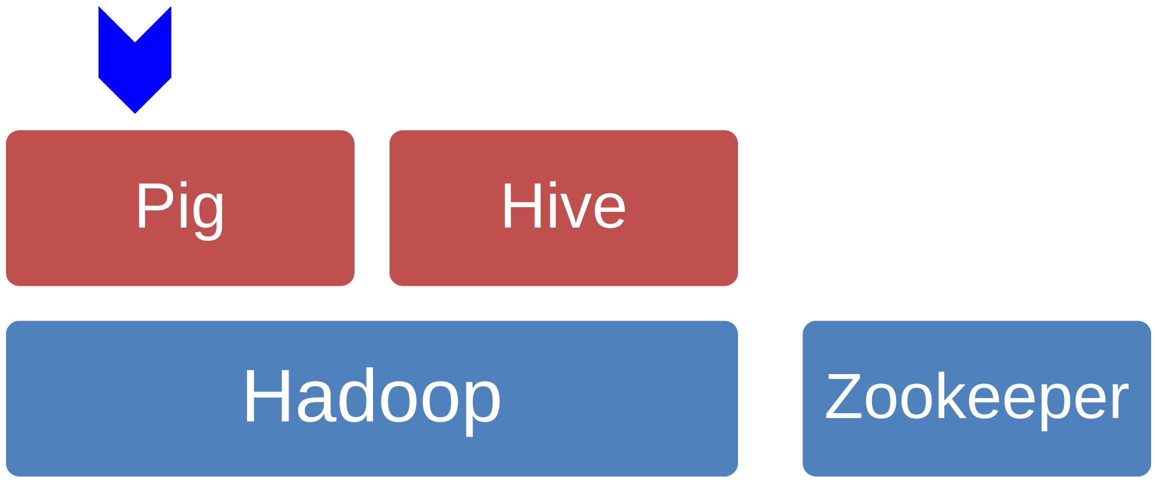
123.123.123.123 - - [26/Apr/2000:00:23:47 -0400] "GET /asctortf/ HTTP/1.0" 200 8130 "http://search.netscape.com/Computers/Data_Formats/Document/Text/RTF" "Mozilla/4.05 (Macintosh; I; PPC)"

123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET /pics/5star2000.gif HTTP/1.0" 200 4005 "http://www.jafsoft.com/asctortf/" "Mozilla/4.05 (Macintosh; I; PPC)"
```

Apache Web Server Log

From: http://www.jafsoft.com/searchengines/log_sample.html

Hadoop Spin-offs



Why Pig?

- Many ways of dealing with small amounts of data:
  - Unstructured Logs on single machine: awk, sed, grep etc.
  - Structured Data: SQL queries through an RDBMS
- How to process giga/tera/peta-bytes of unstructured data?
  - Web crawls, log files, click streams
  - Converting log files into database entries is tedious
- SQL syntax may not be ideal
  - Strict syntax, not suited for scripting–centric programmers
- MapReduce is tedious!
  - Rigid data flow – Map and Reduce
  - Custom code for common operations such as joins – and difficult!
  - Reuse is difficult

Apache Pig

- Pig latin language
  - High-level language to express operations on data
  - User specifies the operations on the data as a *query execution plan* in *Pig Latin*
- Apache Pig framework
  - Interprets and executes pig latin programs into MapReduce jobs
  - Grunt – a command line interface to pig
  - Pig Pen – debugging environment

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Pig Use Cases

- Ad-hoc analysis of unstructured data
  - Web crawls, log files, click streams
- Pig is an excellent ETL tool
  - "Extract, Transform, Load" for pre-processing data before loading it into a data warehouse
- Rapid Prototyping for Analytics
  - You can experiment with large data sets before you write custom applications

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Design Goals of Pig Latin

- *Dataflow language*
  - Operations are expressed as a sequence of steps, where each step performs only a single high-level data transformation
  - Unlike SQL where the query should encapsulate most of the operation required
- *Quick start and interoperability*
  - Quickly load flat files and text files, output can also be tailored to user needs
  - Schemas are optional, i.e., fields can be referred to by position ($1, $4 etc.)
- *Fully nested data model*
  - A field can be of any data type, a data type can encapsulate any other data type
- *UDFs as first-class citizens*
  - User defined functions can take in any data type and return any data type
  - Unlike SQL which restricts function parameters and return types

Pig Latin – Data Types

- Data types
  - *Atom*: Simple atomic value
  - *Tuple*: A tuple is a sequence of fields, each can be any of the data types
  - *Bag*: A bag is a collection of tuples
  - *Map*: A collection of data items that is associated with a dedicated atom

'alice'	('alice', 'lakers')	$\left\{ \begin{array}{c} (\text{'alice'}, \text{'lakers'}) \\ (\text{'alice'}, (\text{'iPod'}, \text{'apple'})) \end{array} \right\}$	$\left[\begin{array}{c} \text{'fan of'} \rightarrow \left\{ \begin{array}{c} \text{'lakers'} \\ \text{'iPod'} \end{array} \right\} \\ \text{'age'} \rightarrow 20 \end{array} \right]$
Atom	Tuple	Bag	Map

Pig Latin – Expressions

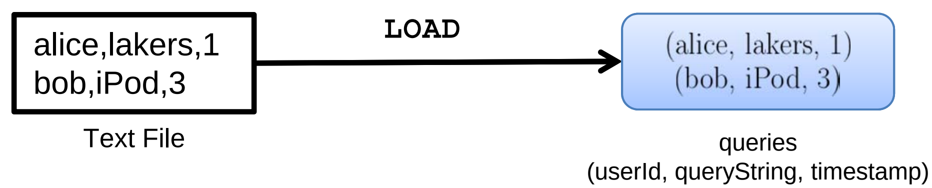
$$t = \left(\overset{f1}{\text{'alice'}}, \overset{f2}{\left\{ \begin{matrix} (\text{'lakers'}, 1) \\ (\text{'iPod'}, 2) \end{matrix} \right\}}, \overset{f3}{[\text{'age'} \rightarrow 20]} \right)$$

Expression Type	Example	Value for tuple *t*
Constant	`'bob'`	Independent of *t*
Field by position	`$0`	'alice'
Field by name	`f3`	$[\text{'age'} \rightarrow 20]$
Projection	`f2,$0`	$\left\{ \begin{matrix} (\text{'lakers'}) \\ (\text{'iPod'}) \end{matrix} \right\}$
Map Lookup	`f3#'age'`	20
Function Evaluation	`SUM(f2.$1)`	$1 + 2 = 3$
Conditional Expression	`F3#'age'>18? 'adult':'minor'`	'adult'
Flattening	`FLATTEN(f2)`	'lakers', 1 'ipod', 2

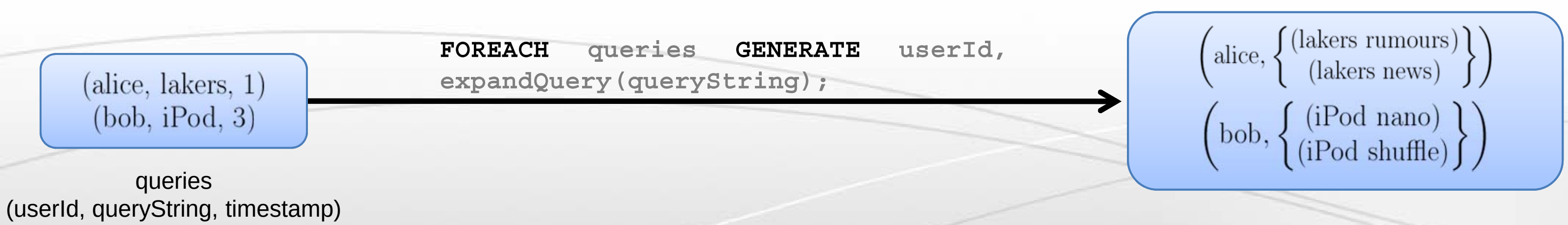
جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Pig Latin – Commands / Operators (1)

- `LOAD` – Specify input data
  - `queries = LOAD 'query_log.txt' USING myLoad() AS (userId, querystring, timestamp);`
    - `myLoad()` is a user defined function (UDF)



- `FOREACH` – Per-tuple processing
  - `expanded_queries = FOREACH queries GENERATE userId, expandQuery(queryString);`



Pig Latin – Commands / Operators (2)

- **FLATTEN** – Remove nested data in tuples

(alice, {(lakers rumours), (lakers news)})
(bob, {(iPod nano), (iPod shuffle)})

```
FLATTEN(expandedQueries);
```

(alice, lakers rumors)
(alice, lakers news)
(bob, iPod nano)
(bob, iPod shuffle)

- **FILTER** – Discarding unwanted data

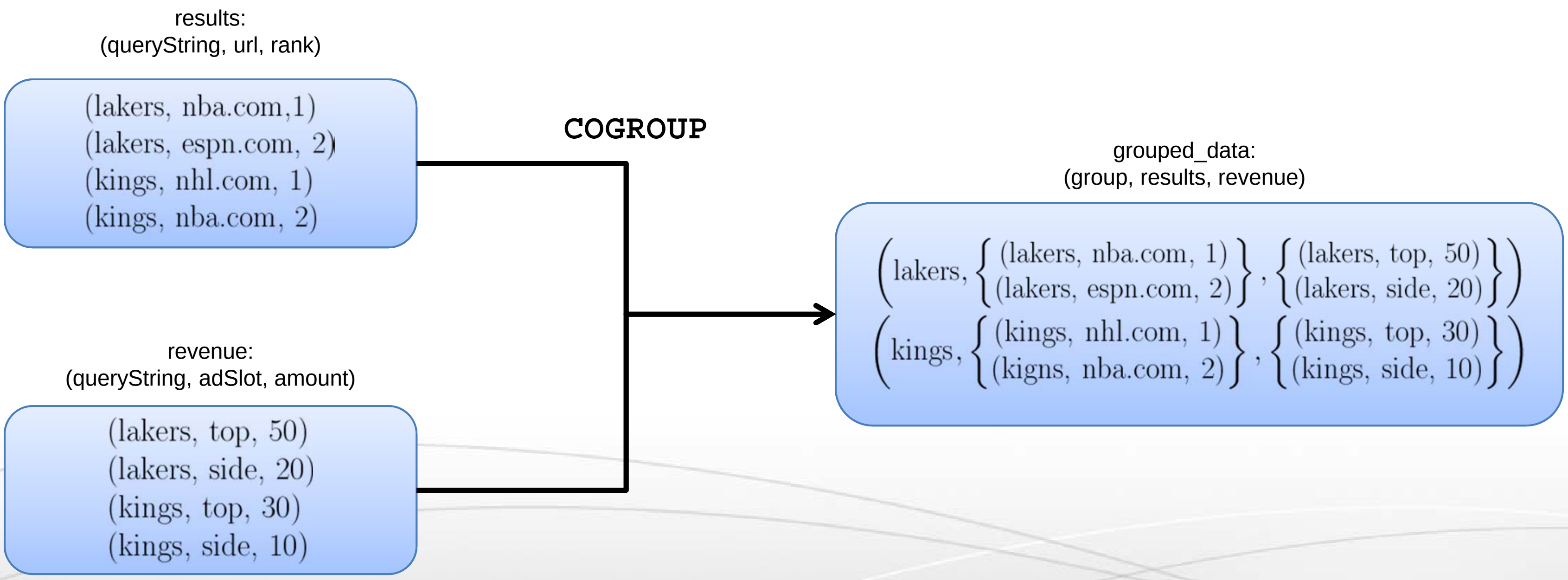
(alice, lakers rumors)
(alice, lakers news)
(bob, iPod nano)
(bob, iPod shuffle)

```
FILTER expandedQueries BY
userId == 'alice'
```

(alice, lakers rumors)
(alice, lakers news)

Pig Latin – Commands / Operators (3)

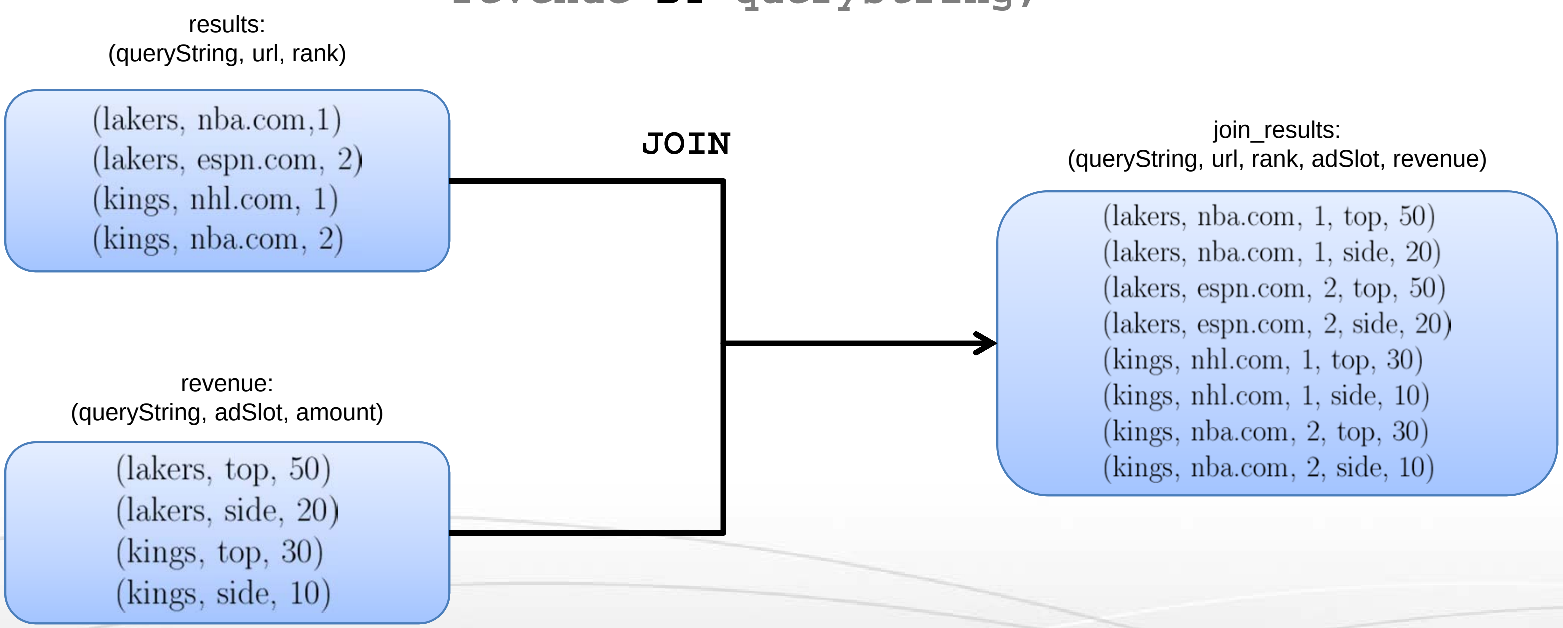
- `COGROUP` – Getting related data together
  - `grouped_data = COGROUP results BY queryString,`
    `revenue BY queryString;`



GROUP is a special case of **COGROUP**

Pig Latin – Commands / Operators (4)

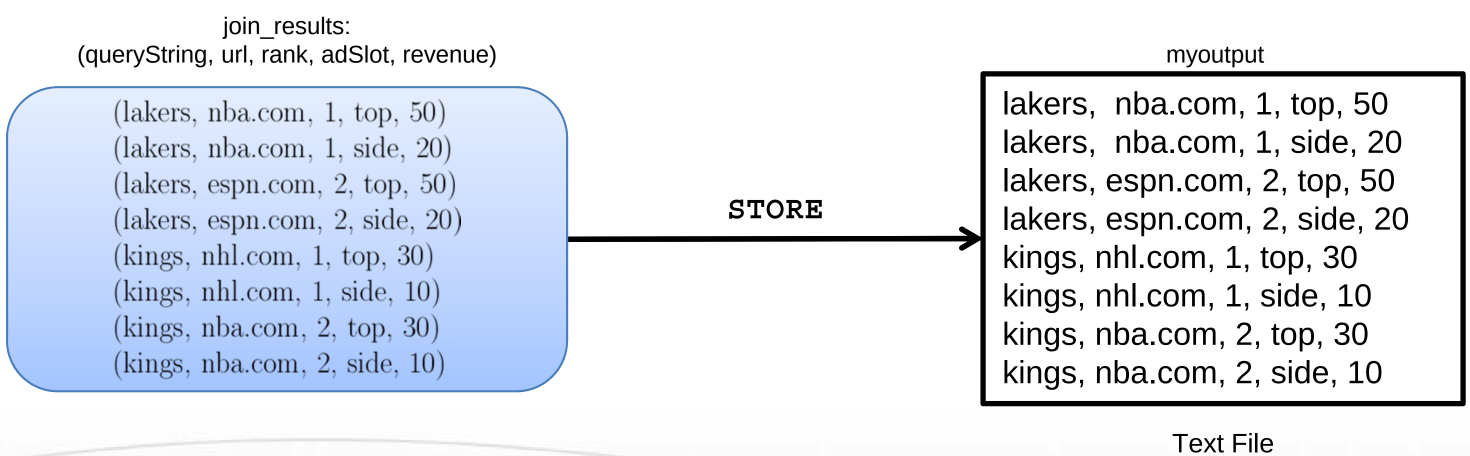
- **JOIN** – Cross product of two tables
  - ```
    join_result = JOIN results BY queryString,
                       revenue BY queryString;
    ```



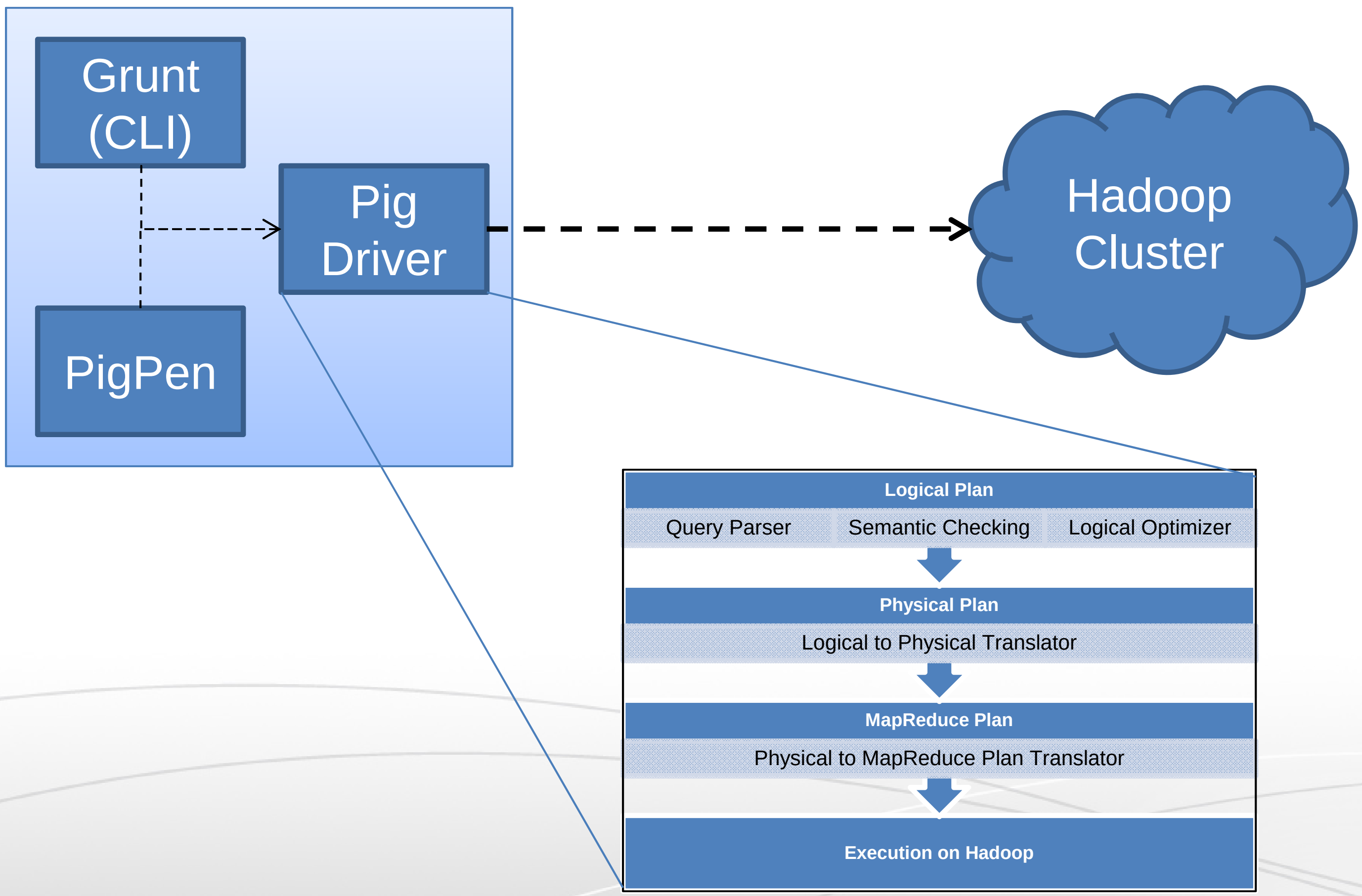
JOIN is the same as **COGROUP** + **FLATTEN**
    ```

Pig Latin – Commands / Operators (5)

- **STORE** – Create output
  - `final_result = STORE join_results INTO 'myoutput',`
    `USING myStore();`

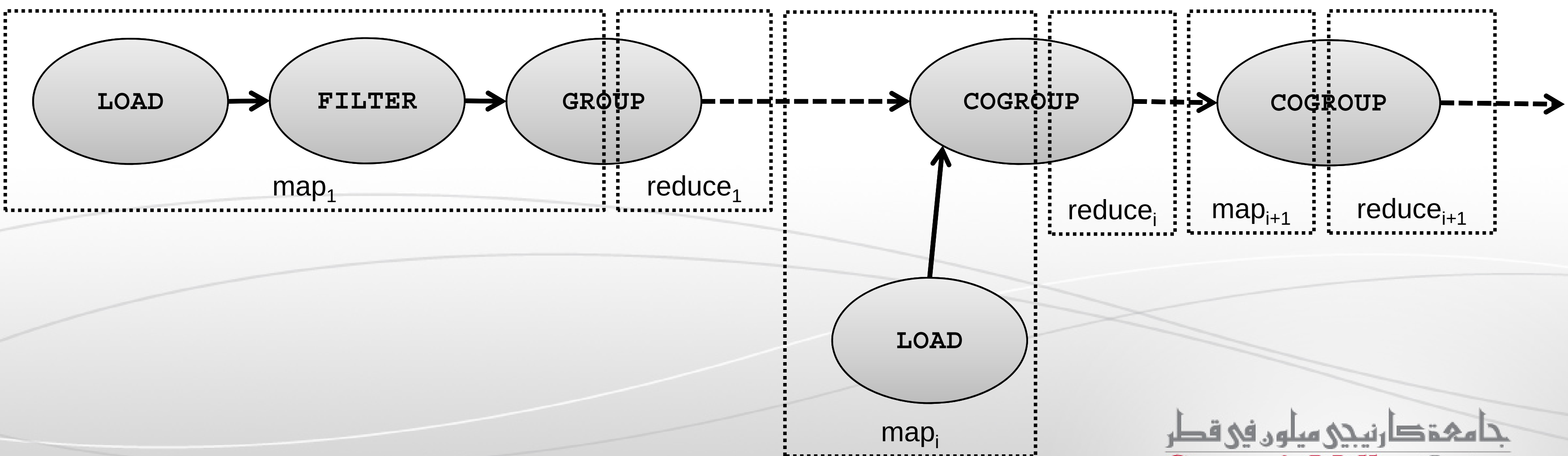


Architecture of Pig

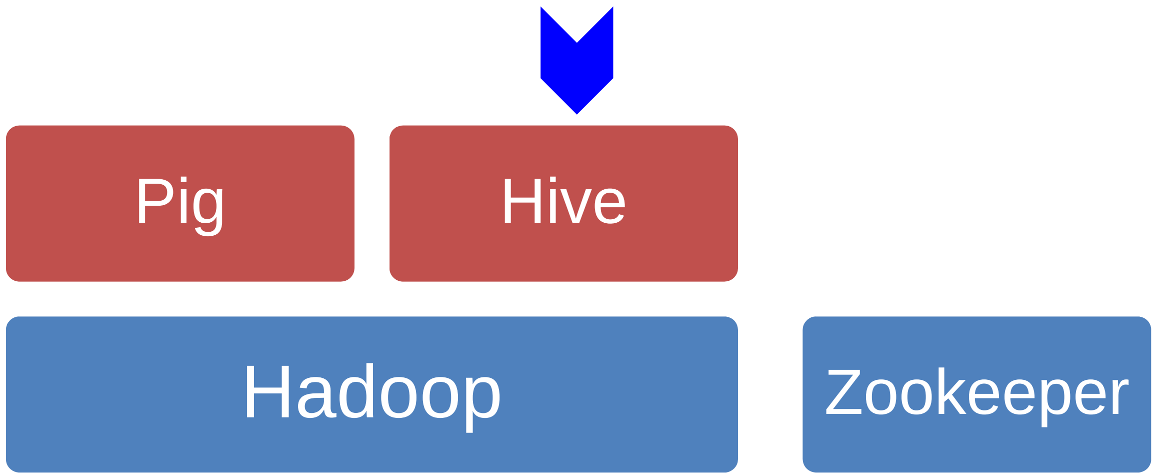


Interpretation of a Pig Program

- The Pig interpreter parses each command and builds a *logical plan* for each bag created by the user.
- The logical plan is converted to a *physical plan*
- Pig then creates an *execution plan* of the physical plan with maps and reduces
- Execution starts only after output is requested– *lazy compilation*



Hadoop Spin-offs



Motivation for Hive

- Organizations that have been using SQL-based RDBMS for storage
  - Oracle, MSSQL, MySQL etc.
- The RDBMS has grown beyond what one server can handle
  - Storage can be expanded to a limit
  - Processing of Queries is limited by the computational power of a single server
- Traditional business analysts with SQL experience
  - May not be proficient at writing Java programs for MapReduce
  - Require SQL interface to run queries on TBs of data

Apache Hive

- Hive is a data warehouse infrastructure built on top of Hadoop that can compile SQL-style queries into MapReduce jobs and run these jobs on a Hadoop cluster
  - MapReduce for execution
  - HDFS for storage

- Key principles of Hive's design:
  - SQL Syntax familiar to data analysts
  - Data that does not fit traditional RDBMS systems
  - To process terabytes and petabytes of data
  - Scalability and Performance

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Hive Use Cases

- Large-scale data processing with SQL-style syntax:

Predictive Modeling & Hypothesis Testing

Google Analytics

Customer Facing Business Intelligence

Document Indexing

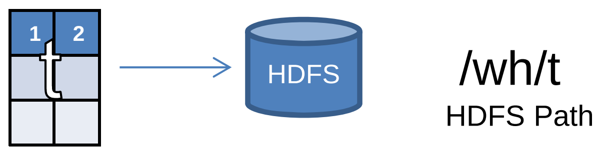
Text Mining & Data Analysis

Hive Components

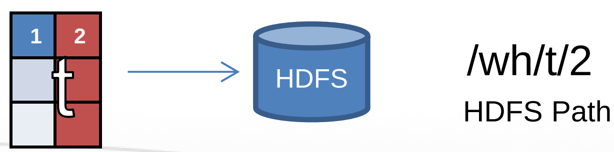
- HiveQL
  - Subset of SQL with extensions for loading and storing
- Hive Services
  - The Hive Driver – compiler, executor engine
  - Web Interface to Hive
  - Hive Hadoop Interface to the JobTracker and NameNode
- Hive Client Connectors
  - For existing Thrift, JDBC and ODBC applications

Hive Data Model

- Tables
  - Similar to Tables in RDBMS
  - Each Table is a unique directory in HDFS



- Partitions
  - Partitions determine the distribution of data within a table.
  - Each partition is a sub-directory of the main directory in HDFS



- Buckets
  - Partitions can be further divided into buckets.
  - Each bucket is stored as a file in the directory

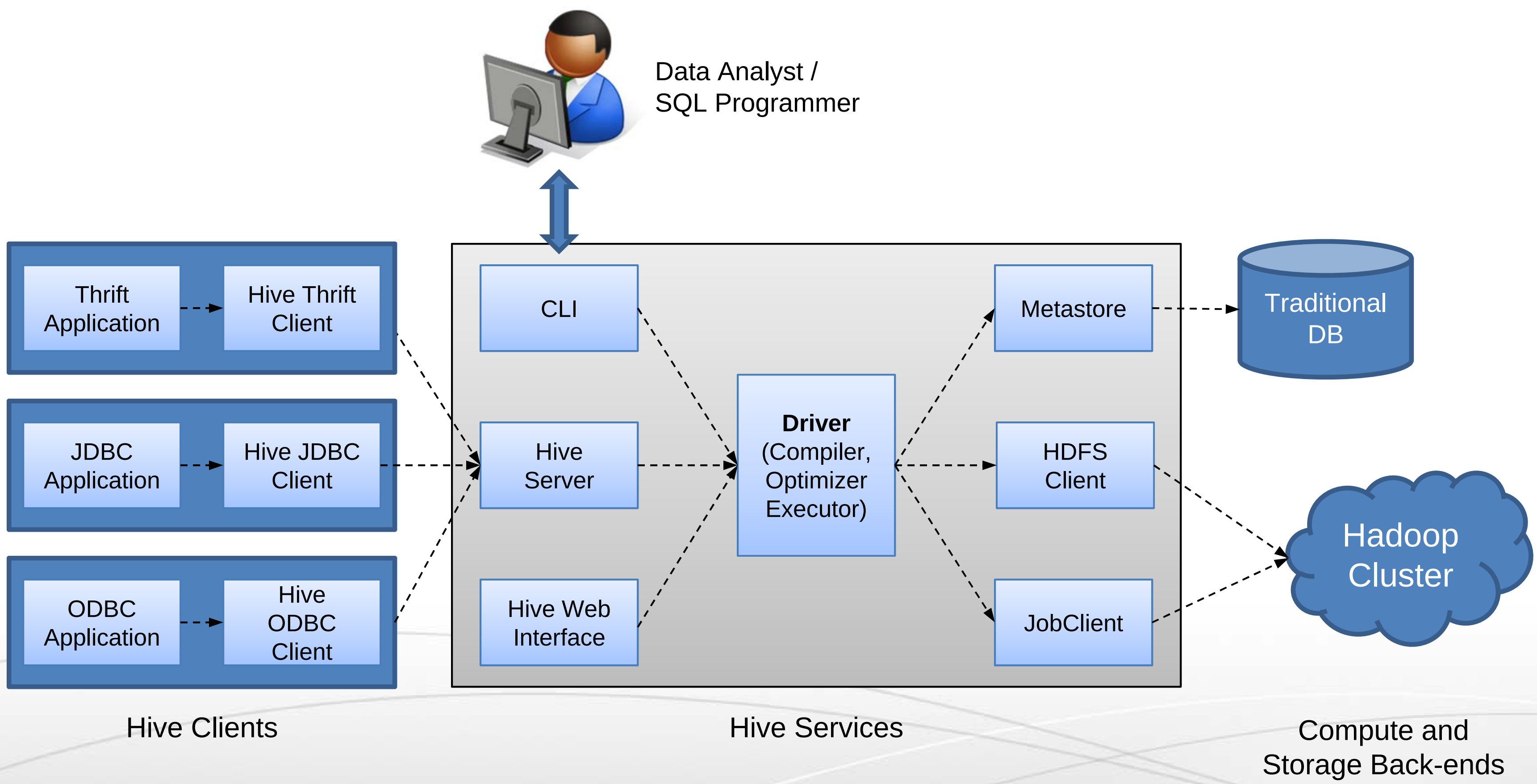
HiveQL Commands

- Data Definition Language
  - Used to describe, view and alter tables.
  - For E.g. **`CREATE TABLE`** and **`DROP TABLE`** commands with extensions to define file formats, partitioning and bucketing information
- Data Manipulation Language
  - Used to load data from external tables and insert rows using the **`LOAD`** and **`INSERT`** commands
- Query Statements
  - SELECT
  - JOIN
  - UNION
  - etc.

User-Defined Functions in Hive

- Four Types
- User Defined Functions (UDF)
  - Perform tasks such as Substr, Trim etc. on data elements
- User Defined Aggregation Functions (UDAF)
  - Performed on Columns
  - Sum, Average, Max, Min… etc.
- User Defined Table-Generating Functions (UDTF)
  - Outputs a new table
  - Explode is an example – similar to FLATTEN() in Pig.
- Custom MapReduce scripts
  - The MR scripts must read rows from standard output
  - Write rows to standard input.

Architecture of Hive



جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Compilation of Hive Programs

Parser

Parses the query string into a parse tree representation

↓

Semantic Analyzer

Retrieves the schema and verifies the validity of the query.

Transforms the query into an internal representation

↓

Logical Plan Generator

Converts the internal query representation into a logical execution plan

↓

Optimizer

Multiple passes over the logical plan and rewrites it

Combines Multiple joins, reduces the number of MR jobs, etc.

↓

Physical Plan Generator

Logical plan is converted into a physical plan, which is a DAG of Map-Reduce jobs.

↓

Execution in Hadoop

Hadoop Spin-offs

Why ZooKeeper?

- Writing distributed applications is hard
  - Need to deal with synchronization, concurrency , naming, consensus, configuration etc.
  - Well known algorithms exist for each of these problems
  - But programmers have to re-implement them for each distributed application they write.

- Master-slave architecture is popular for distributed applications
  - But how do you deal with master failures?
  - Single master can quickly become the performance bottleneck for many distributed applications.

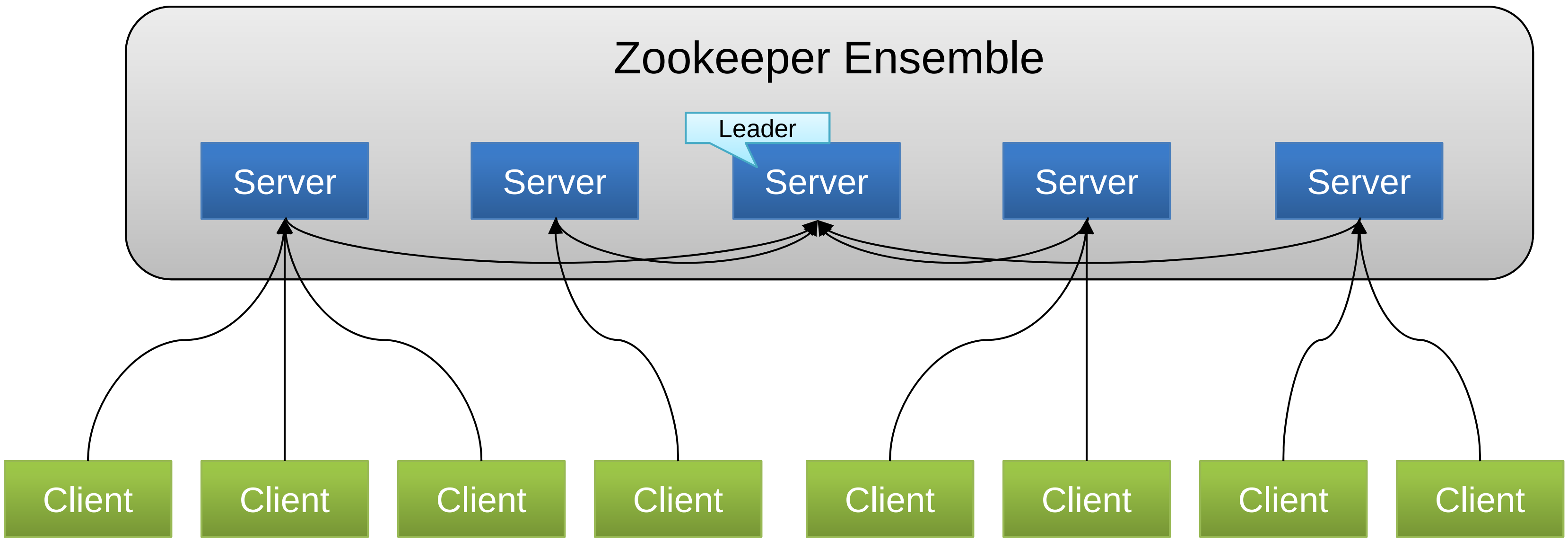
What is Apache ZooKeeper?

- ZooKeeper is a distributed co-ordination service for large-scale distributed systems

- ZooKeeper allows application developers to build the following systems for their distributed application:
  - Naming
  - Configuration
  - Synchronization
  - Organization
  - Heartbeat systems
  - Democracy / Leader election

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

ZooKeeper Architecture



Client Interactions with Zookeeper

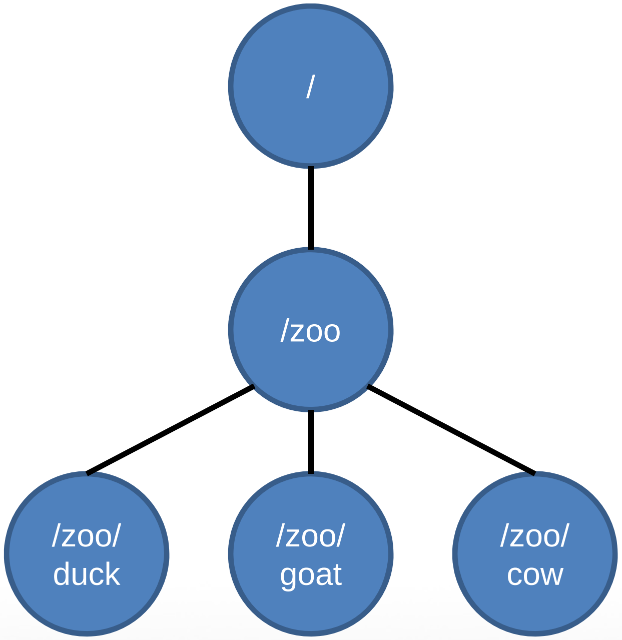
- Clients must have the list of all the zookeeper servers in the ensemble
  - Clients will attempt to connect to the next server in the ensemble if one fails

- Once a client connects to a server, it creates a new *session*
  - The application can set the session timeout value
  - Session is kept alive through the heartbeat mechanism.
  - Failure events are automatically handled and watch events are delivered to the client on reconnection.

Zookeeper Data Model

- Similar to a filesystem
  - Hierarchical layout to denote a membership list.
- Each node is known as a *znode*
  - znodes can be *ephemeral* or *persistent*
  - An ephemeral znode exists as long as the session of the client who created it.
  - Ephemeral znodes cannot have children.
  - *Sequential* znodes are persistent and have a sequence number attached.
  - For e.g. if a second goat znode is declared under /zoo, it will be /zoo/goat2 etc.

- Znodes can store data and have an associated ACL
  - Size limit of 1 MB per znode
  - Sanity check as its more than enough to store configuration/state information



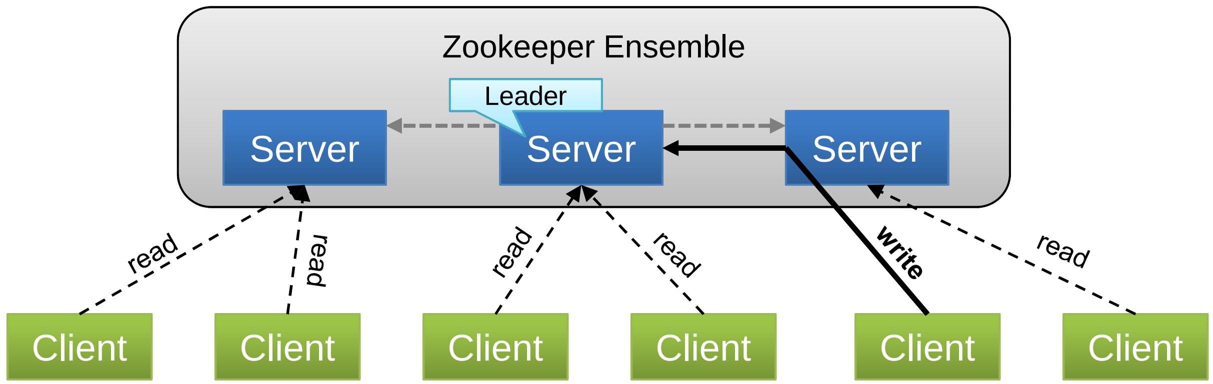
جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

ZooKeeper API

Operation	Description
`create`	Creates a znode
`delete`	Deletes a znode (znode should not have any children)
`exists`	Tests if a znode exists and retrieves its metadata
`getACL, setACL`	Gets/sets ACL for a znode
`getChildren`	Gets a list of children for a znode
`getData, setData`	Gets and sets data for a znode
`sync`	Synchronizes a client's view of a znode with ZooKeeper

Reads, Writes and Watches

- Reads can be collected from any server.
- Write requests are always forwarded to the leader which commits the write to a majority of servers *atomically*



- A *watch* can be optionally set on a znode after a read operation to monitor if it has been deleted or changed.
  - A watch is triggered when there is an update to a specific znode and it can be used to notify clients that have read the znode.

Zookeeper Protocol : Zab

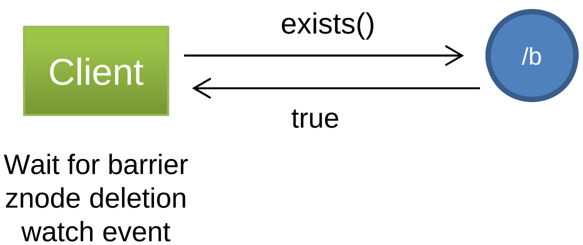
- *Zab* ensures zookeeper can keep its promises to clients. It is a two phase protocol
- *Phase 1: Leader Election*
  - All the members of the ensemble elect a distinguished member, called the leader and other members are designated as followers.
  - The election is declared complete when a majority (quorum) of followers have synchronized the state with the leader
- *Phase 2: Atomic Broadcast*
  - Write requests are always forwarded to the leader
  - The update is broadcast to all the followers.
  - The leader then commits the update when a majority of followers have persisted the change
  - The writes thus happen atomically in accordance with a two-phase commit (2PC) protocol

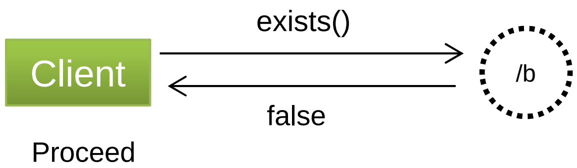
Zookeeper guarantees…

- That every modification to the znode tree is *replicated to a majority* of the ensemble
- That *fault tolerance* is achieved
  - As long as a majority of the nodes in the ensemble are active.
  - Ensembles are typically configured to be an odd number.
- That every update is *sequentially consistent*
- That all updates to the znode state are *atomic*
- That every client sees only a *single system image*
- That updates are *durable* and persist, in spite of server failures.
- That client's view is *timely* and is not out-of-date

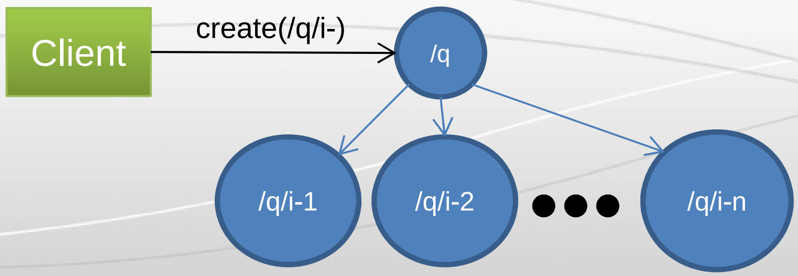
Creating Higher-level Constructs with Zookeeper

- Barrier
  - Creating a barrier for distributed clients is easy.
  - Designate a barrier node, and clients check if it exists.





- Queue
  - `create()` sequential znodes under a parent to designate queue items.
  - Queue can be processed using a `getchildren()` call on the /q item. A watch can notify client of new items on the queue



جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Next Class

Virtualization