

# Module V- Programming Models and Advances

Dr. Madhukrishna Priyadarsini

Assistant Professor

NIT Trichy

# Outline:

## Programming Models:

- MapReduce
- Apache Spark
- Tensor Flow
- **Intercloud Architecture**

## Advances:

- Mobile Cloud Computing
- Edge Computing
- Fog Computing

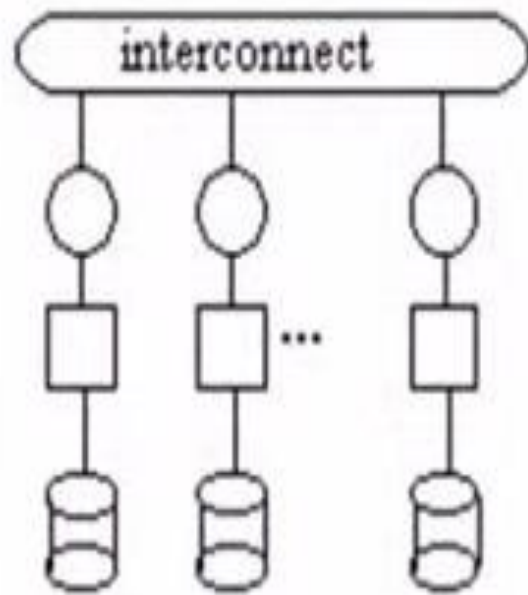
# MapReduce

- MapReduce is a programming model developed by Google
- Objective:
  - Implement large scale search
  - Text processing on massively scalable web data stored using big table and GFS distributed file systems
- Designed for processing and generating large volumes of data via massively parallel computations, utilizing tens of thousands of processors at a time.
- Fault tolerant: ensure progress of computation even if processors and networks fail
- Example:
  - Hadoop: An open source implementation of MapReduce (developed at Yahoo!)
  - Available on pre-packaged AMIs on Amazon EC2 cloud platform

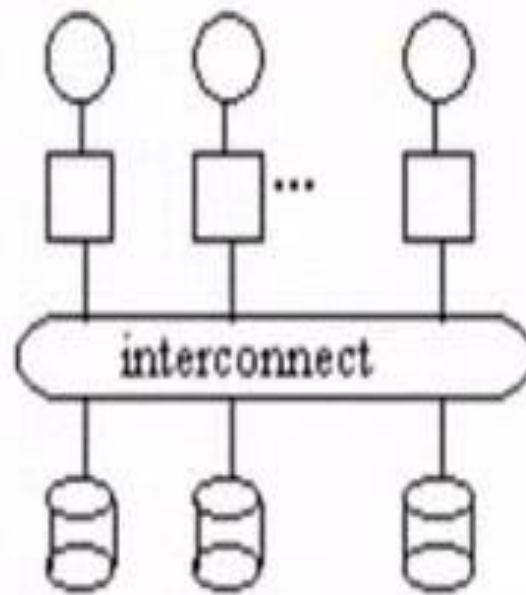
# Parallel computing

- Different models of parallel computing
  - Nature and evolution of multi-processor computer architecture
  - Shared-memory model
    - Assumes that any processor can access any memory location
    - Unequal latency
  - Distributed memory model
    - Each processor can access only its own memory and communicates with other processors using message passing
- Parallel computing
  - Developed for computing intensive scientific tasks
  - Later found application in the database arena
    - Shared memory
    - Shared disk
    - Shared nothing

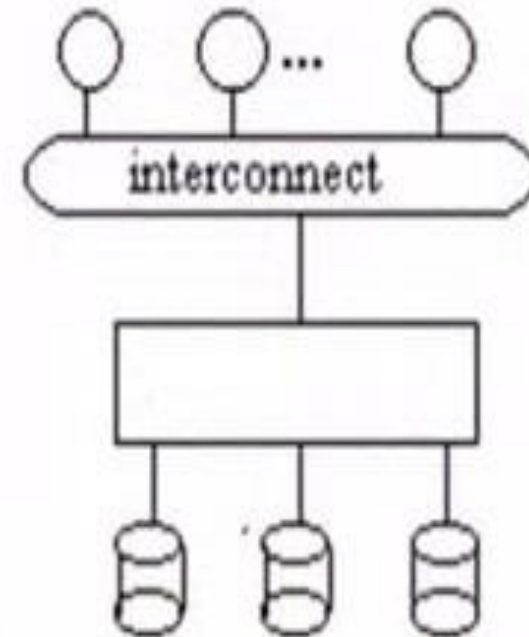
# Parallel database architecture



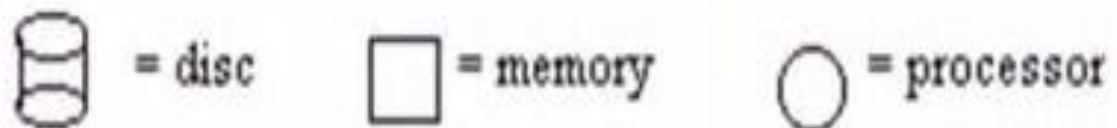
a) shared nothing



b) shared disc



c) shared memory



# Parallel database architecture contd...

- Shared memory
  - Suitable for servers with multiple CPUs
  - Memory address space is shared and managed by a symmetric multi-processing (SMP) operating system
  - SMP:
    - Schedules processes in parallel exploiting all the processors
- Shared nothing
  - Cluster of independent servers each with its own disk space
  - Connected by a network
- Shared disk
  - Hybrid architecture
  - Independent server cluster share storage through high speed network storage viz, NAS or SAN
  - Clusters are connected to storage via: standard ethernet or faster Fiber channel or infiband connections

# Parallel Efficiency

- If a task takes time  $T$  in uniprocessor system, it should take  $T/p$  if executed on  $p$  processors.
- Inefficiencies introduced in distributed computation due to:
  - Need for synchronization among processors
  - Overheads of message communication between processors
  - Imbalance in the distribution of work to the processors
- Parallel efficiency of an algorithm is defined as:  $\epsilon = \frac{T}{p \cdot T/p}$
- Scalable parallel implementation
  - Parallel efficiency remains constant as the size of data is increased along with a corresponding increase in processors
  - Parallel efficiency increases with the size of data for a fixed number of processors

# MapReduce Model

- It is a parallel programming abstraction
- Used by many different parallel applications which carry out large scale computation involving thousands of processors
- Leverages a common underlying fault-tolerant implementation
- MapReduce has two phases:
  - Map operation
  - Reduce operation
- A configurable number of M 'mapper' processors and R 'reducer' processors are assigned to work on the problem
- Computation is coordinated by a single master process



- Map Phase:

- Each mapper reads approximately  $1/M$  of the input from the global file system, using locations given by the master
- Map operation consists of transforming one set of key-value pairs to another:

$$Map: (K1, V1) \rightarrow [(K2, V2)]$$

- Each mapper writes computation results in one file per reducer
- Files are sorted by a key and stored to the local file system
- The master keeps track of the location of these files

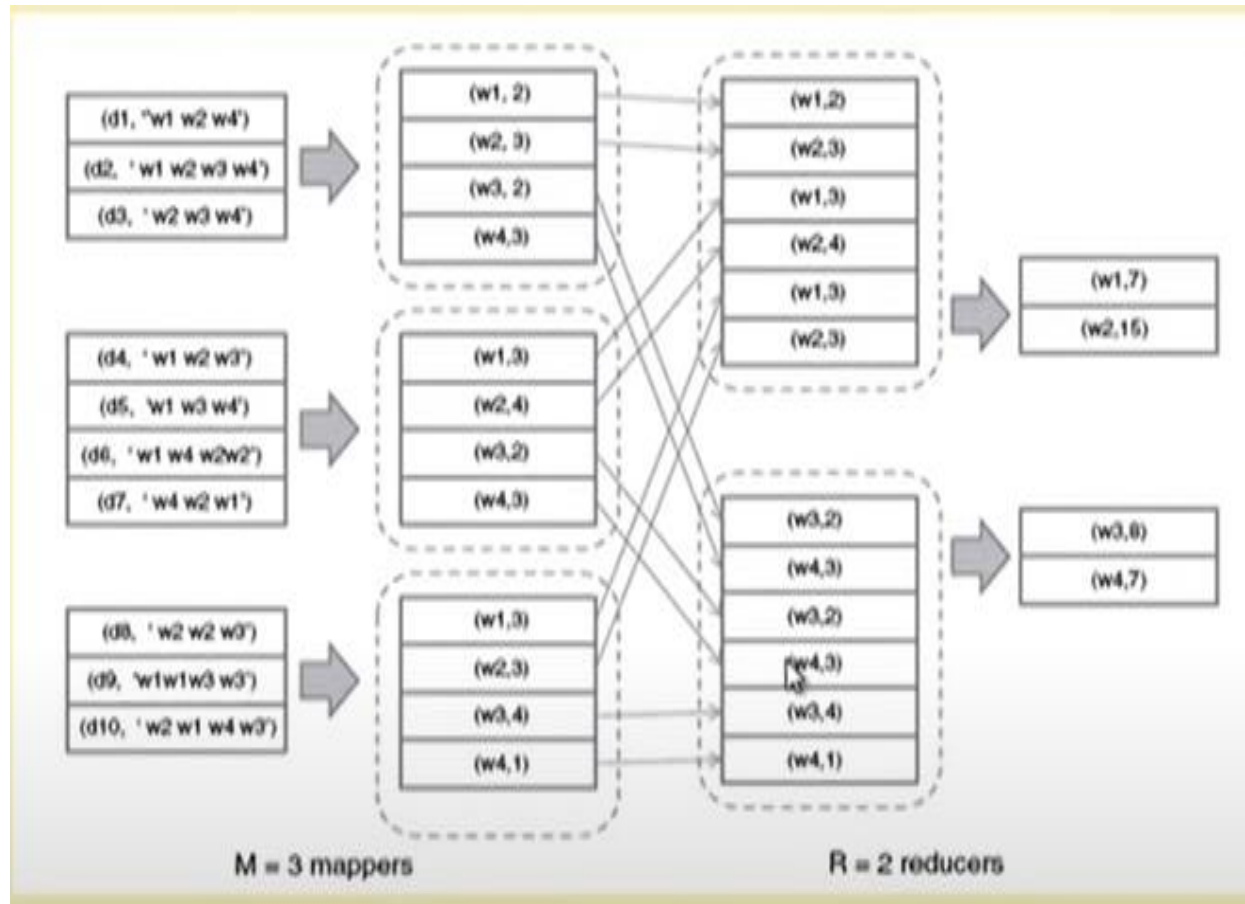
- Reduce Phase:

- The master informs the reducers where the partial computation have been stored on local files of respective mappers
- Reducers make remote procedure call to the mappers to fetch the files
- Each reducer groups the results of the map step using the same key and performs a function  $f$  on the list of values that correspond to these key values:

$$reduce: (K2, [V2]) \rightarrow (K2, f[V2])$$

- Final results are written back to the GFS file system

# MapReduce Example



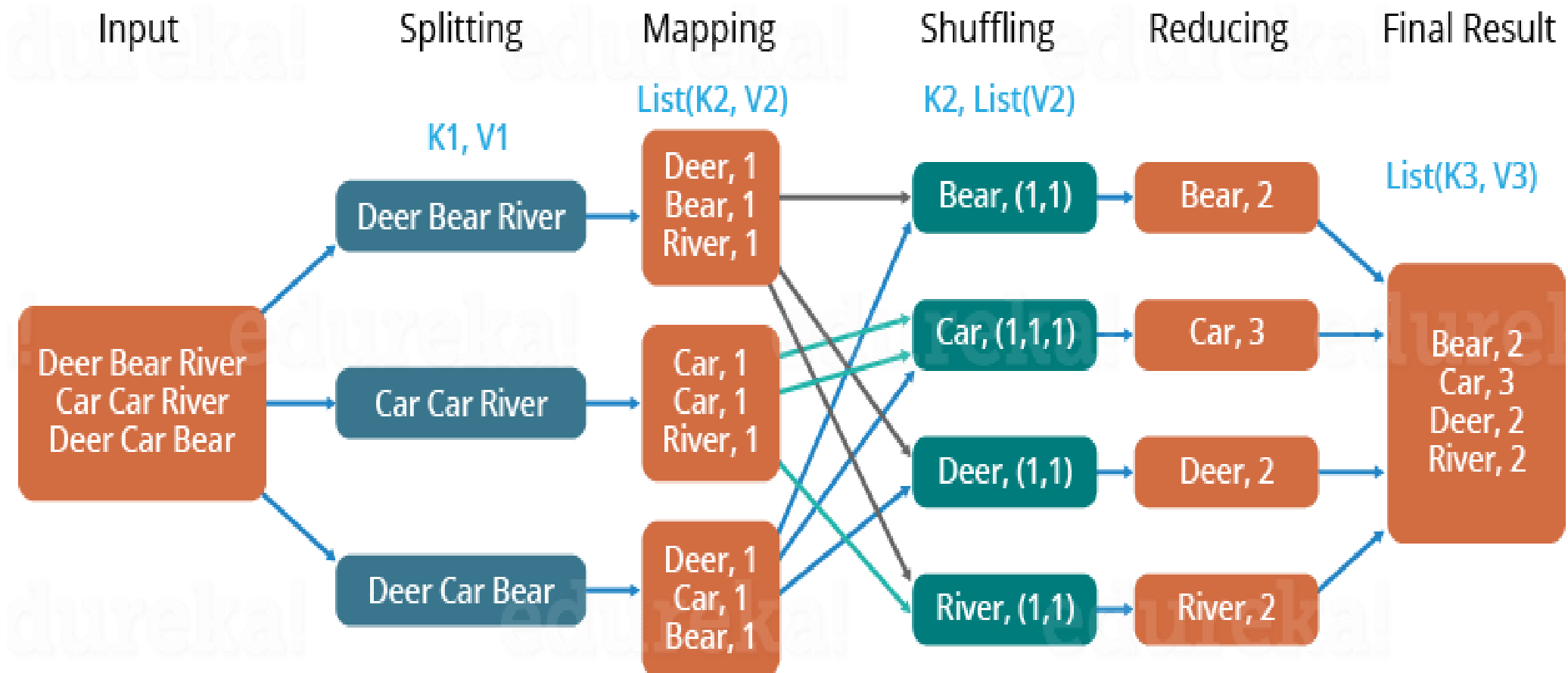
- 3 mappers; 2 reducers
- Map function:

$$(d_k, [w_1 \dots w_n]) \rightarrow [(w_i, c_i)].$$

- Reduce function:

$$(w_i, [c_i]) \rightarrow \left( w_i, \sum_i c_i \right)$$

## The Overall MapReduce Word Count Process



# MapReduce: Fault Tolerance

- Heartbeat communication:
  - Updates are exchanged regarding the status of the task assigned to workers
  - Communication exists, but no progress: master duplicates those tasks and assigns to processors who have already completed
- If a mapper fails, the master reassigns the key-range designated to it to another working node for re-execution
  - Re-execution is required as the partial computations are written into local files, rather than GFS file system
- If a reducer fails, only the remaining tasks are reassigned to another node, since the completed tasks are already written back into GFS

# MapReduce Efficiency

- General computation task on a volume of data  $D$
- Takes  $wD$  time on an uniprocessor (time to read data from disk + performing computation + time to write back to disk)
- Time to read/write one word from/to disk =  $c$
- Now, the computational task is decomposed into map and reduce stages as follows:
  - ❑ Map stage:
    - ✓ Mapping time =  $c_m D$
    - ✓ Data produced as output =  $\sigma D$
  - ❑ Reduce stage:
    - ✓ Reducing time =  $c_r \sigma D$
    - ✓ Data produced as output =  $\sigma \mu D$

- Considering no overheads in decomposing a task into a map and a reduce stages, we have the following relation:

$$wD = cD + cmD + cr\sigma D + c\sigma\mu D$$

- Now, we use  $p$  processors that server as both mapper and reducer in respective phases to solve the problem
- Additional overhead:
  - Each mapper writes to its local disk followed by each reducer remotely reading from the local disk of each mapper
- For analysis purpose; time to read a word locally or remotely is same
- Time to read data from disk by each mapper =  $\frac{wD}{p}$
- Data produced by each mapper =  $\frac{\sigma D}{p}$

- Time required to write into local disk  $= \frac{c\sigma D}{p}$
- Data read by each reducer from its partition in each of  $p$  mappers  $= \frac{\sigma D}{p^2}$
- The entire exchange can be executed by  $p$  steps, with each reducer  $r$  reading from mapper  $r + i \bmod p$  in step  $i$
- Transfer time from mapper local disk to GFS for each reducer  $= \frac{c\sigma D}{p^2} \times p = \frac{c\sigma D}{p}$
- Total overhead in parallel implementation due to intermediate disk reads and writes  $(\frac{wD}{p} + 2c \frac{\sigma D}{p})$
- Parallel efficiency of the MapReduce implementation:

$$\epsilon_{MR} = \frac{wD}{p(\frac{wD}{p} + 2c \frac{\sigma D}{p})} = \frac{1}{1 + \frac{2c}{w}\sigma}$$



# MapReduce: Applications

- Indexing a large collection of documents
  - Important aspect in web search as well as handling structured data
  - The map task consists of emitting a word-document/record-id pair for each word:  
 $(d_k, [w_1, \dots, w_n]) \rightarrow [(w_i, d_k)]$
  - The reduce step groups the pairs by word and creates an index entry for each word:  
 $[(w_i, d_k)] \rightarrow (w_i, [d_{i1}, \dots, d_{im}])$
- Relational operations using MapReduce
  - Execute SQL statements (relational joins/group by) on large datasets
  - Advantages over parallel database
    - Large scale
    - Fault-tolerance

**Problem-1:** In a MapReduce framework consider the HDFS block size is 64MB. We have 3 files of size 64Kb, 65Mb, and 127Mb. How many blocks will be created by Hadoop framework?

**Problem-2:** Write the pseudo codes (for map and reduce functions) for calculating the average of a set of integers in MapReduce.

Suppose  $A = (10, 20, 30, 40, 50)$  is a set of integers. Show the map and reduce outputs?

**Problem-3:** Compute total and average salary of organization XYZ and group by gender (male or female) using MapReduce. The input is as follows:

**Name, Gender, Salary**

John, M, 10k

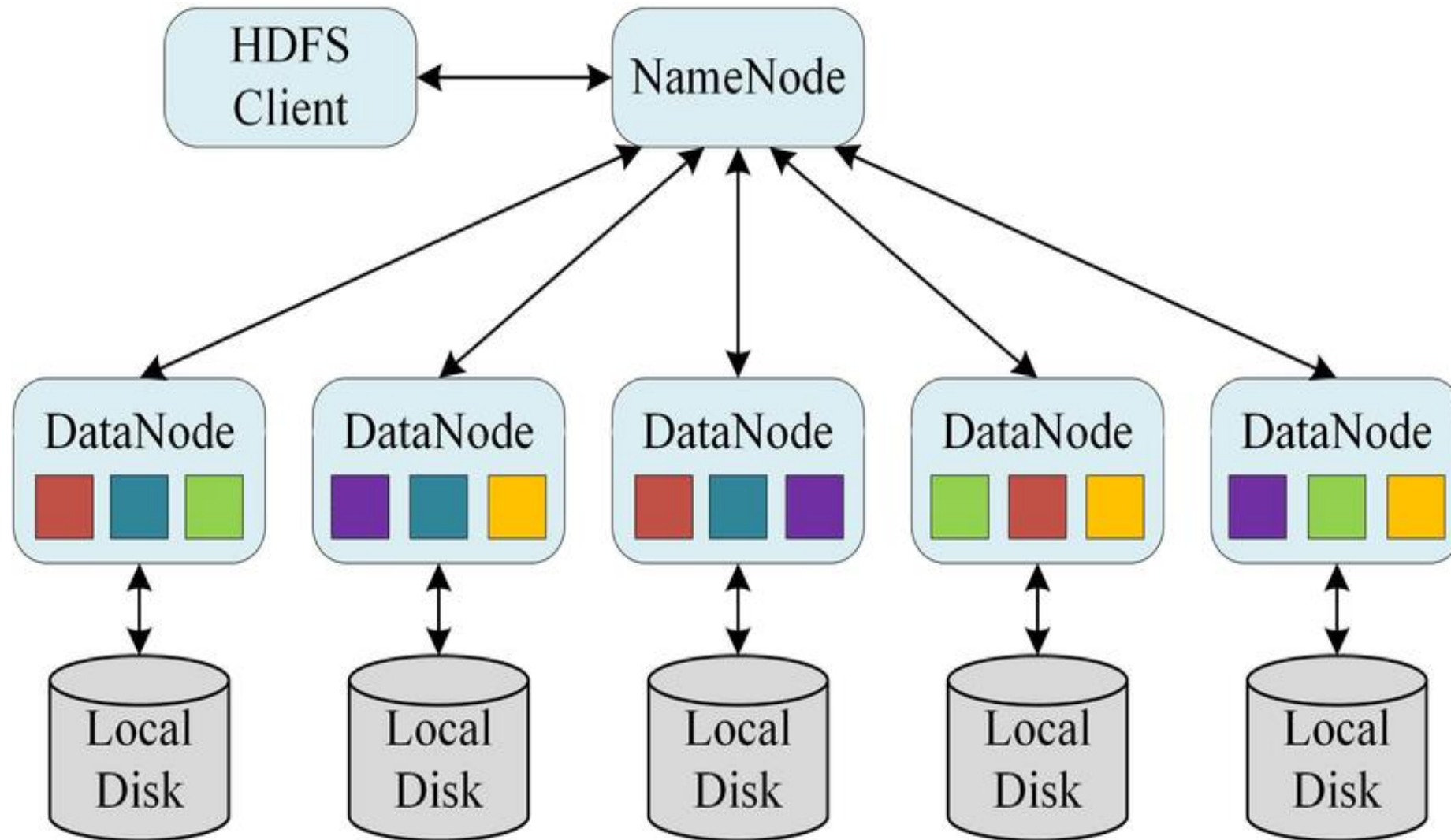
Martha, F, 15k

.....

# Hadoop

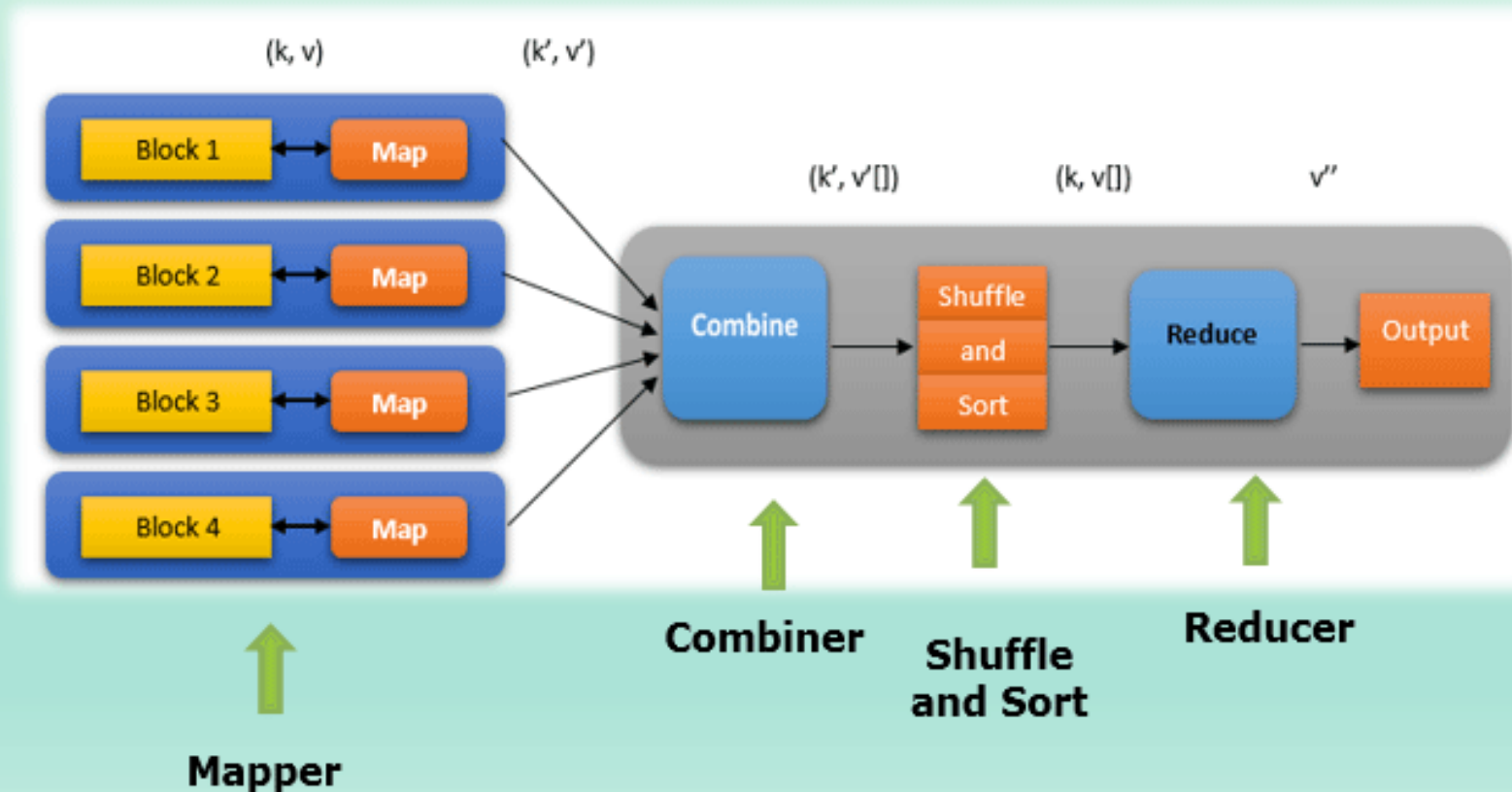
- Hadoop is an open source framework that allows us to store and process large datasets in parallel and distributed manner.
- Two main components: HDFS and MapReduce.
- Hadoop Distributed file system (HDFS) is the primary data storage system used by Hadoop applications.
- MapReduce is the processing unit of Hadoop.

# Hadoop Distributed File System (HDFS)



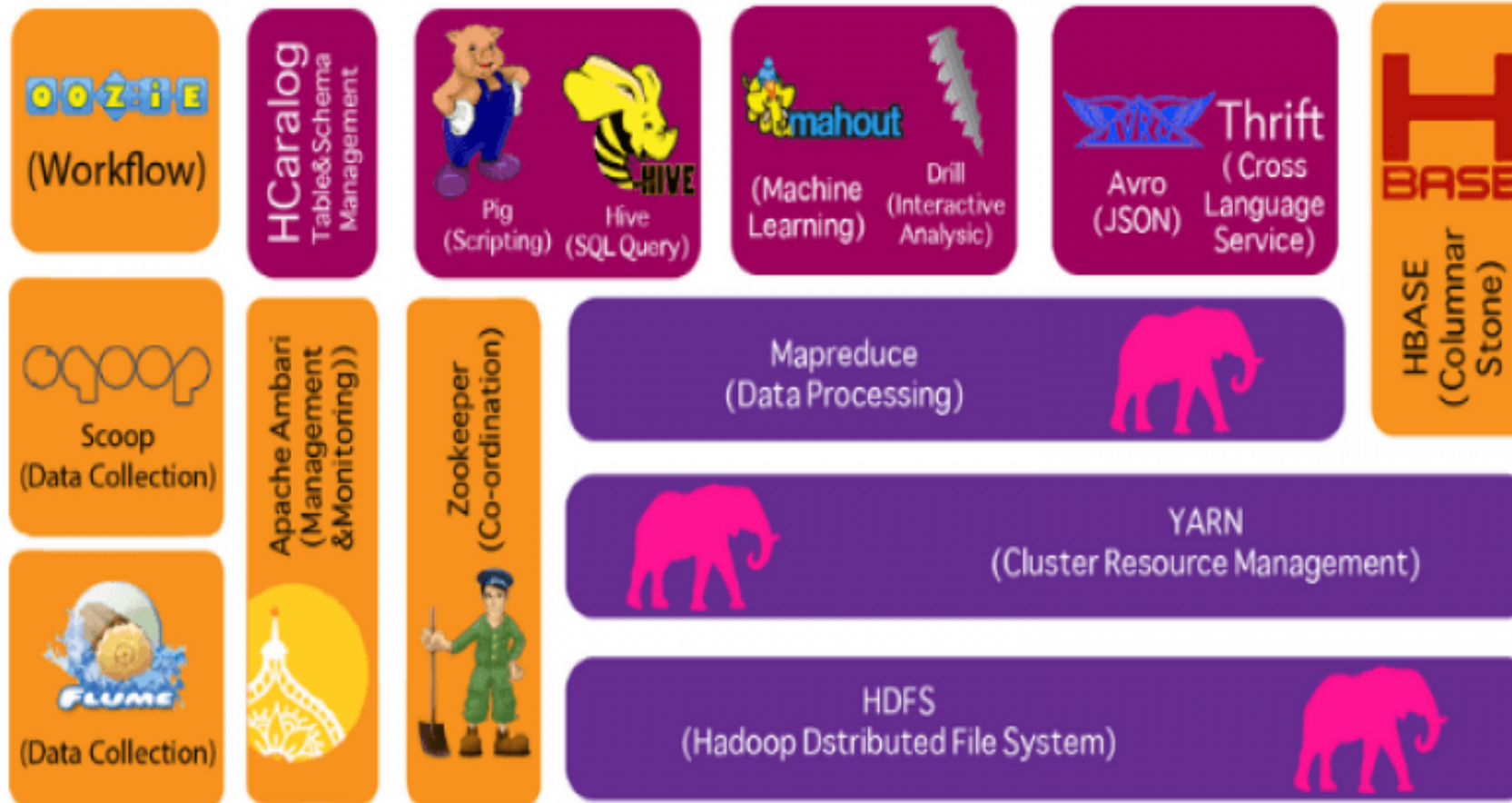
# MapReduce

## How MapReduce Works



# Hadoop Eco System

## Top Hadoop Ecosystem Components





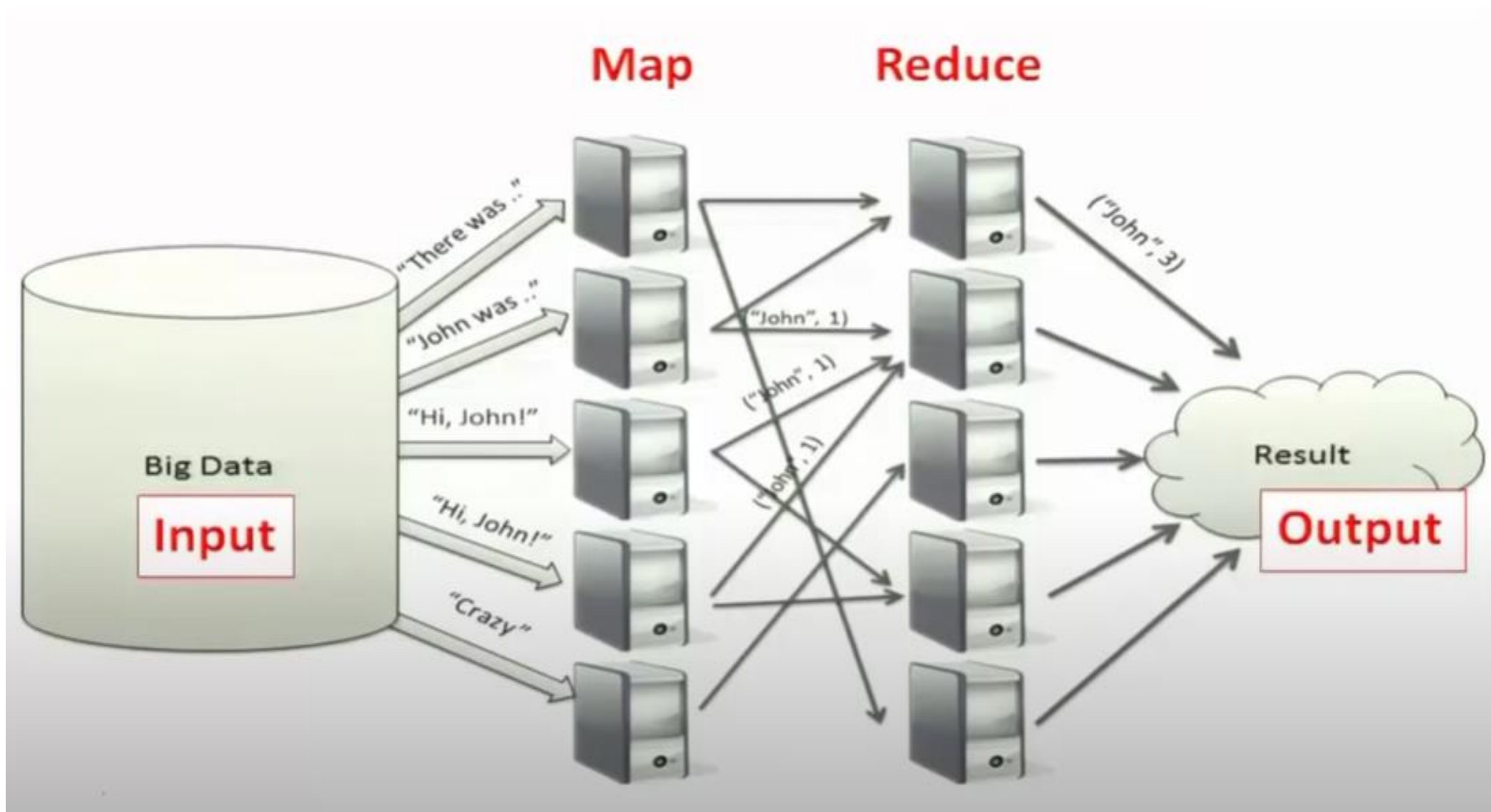
# Apache Spark

- **Framework of Spark**
- **Resilient Distributed Datasets (RDDs)**
- **Applications: PageRank, GraphX**

- ❑ Apache Spark is a big data analytics framework that was designed by University of California in 2012. Since, then it has gained a lot of attraction both in academia and industry.
- ❑ It is an another system for big data analytics.

# Need of Spark

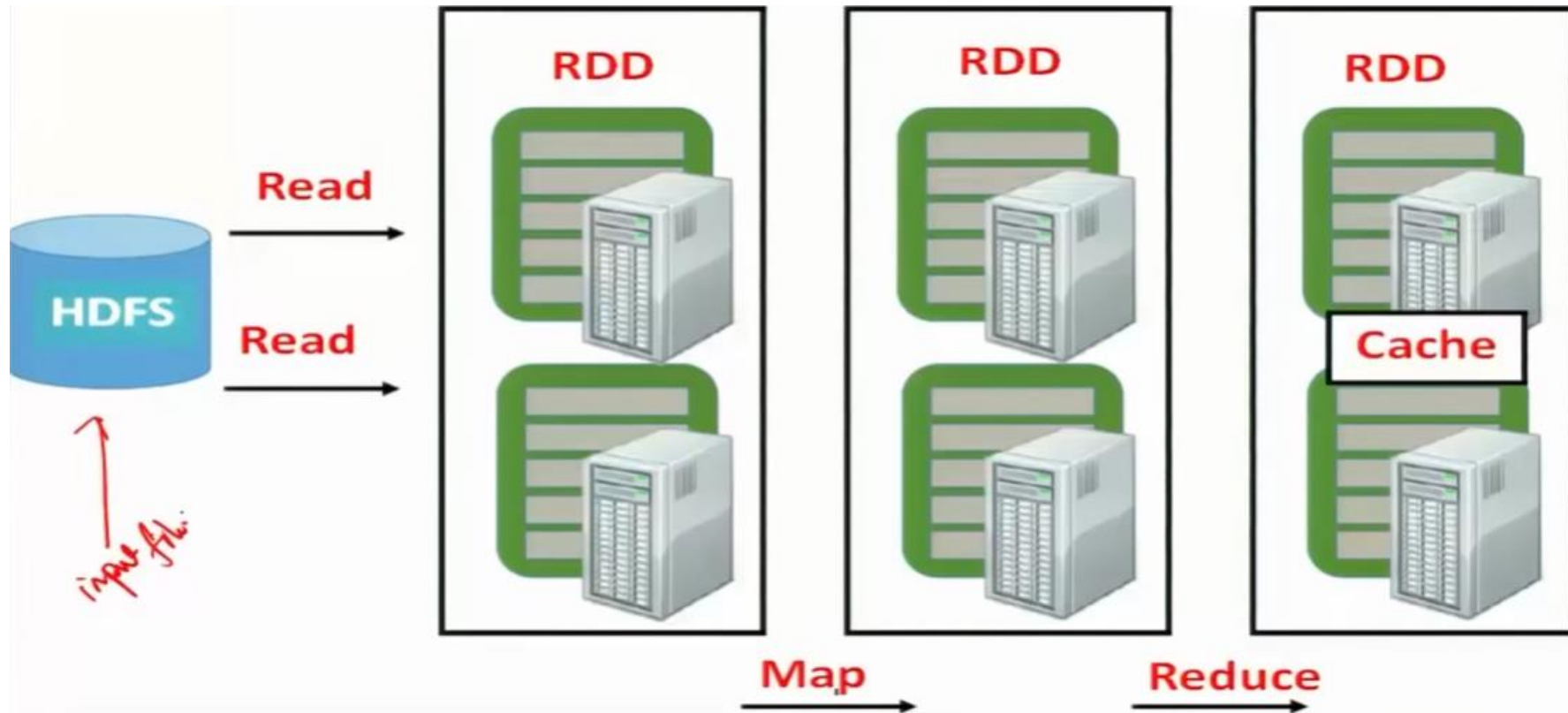
- ❑ Is not MapReduce good enough?
  - ❑ Simplifies batch processing on large commodity clusters.
- ❑ MapReduce can be expensive for some applications such as;
  - ❑ Iterative
  - ❑ Interactive
- ❑ Lacks efficient data sharing
- ❑ Specialized frameworks did evolve for different programming models
  - ❑ Bulk synchronous processing (Pregel)
  - ❑ Iterative MapReduce (Hadoop)



# Solution: Resilient Distributed Datasets (RDDs)

## Resilient Distributed Data Sets (RDDs):

- Immutable, partitioned collection of records
- Built through coarse grained transformations (map, join,.....)
- Can be cached for efficient reuse.



- Fault Recovery?

- Lineage!
- Log the coarse grained operation applied to a partitioned dataset
- Simply recompute the lost partition if failure occurs
- No cost if no failure

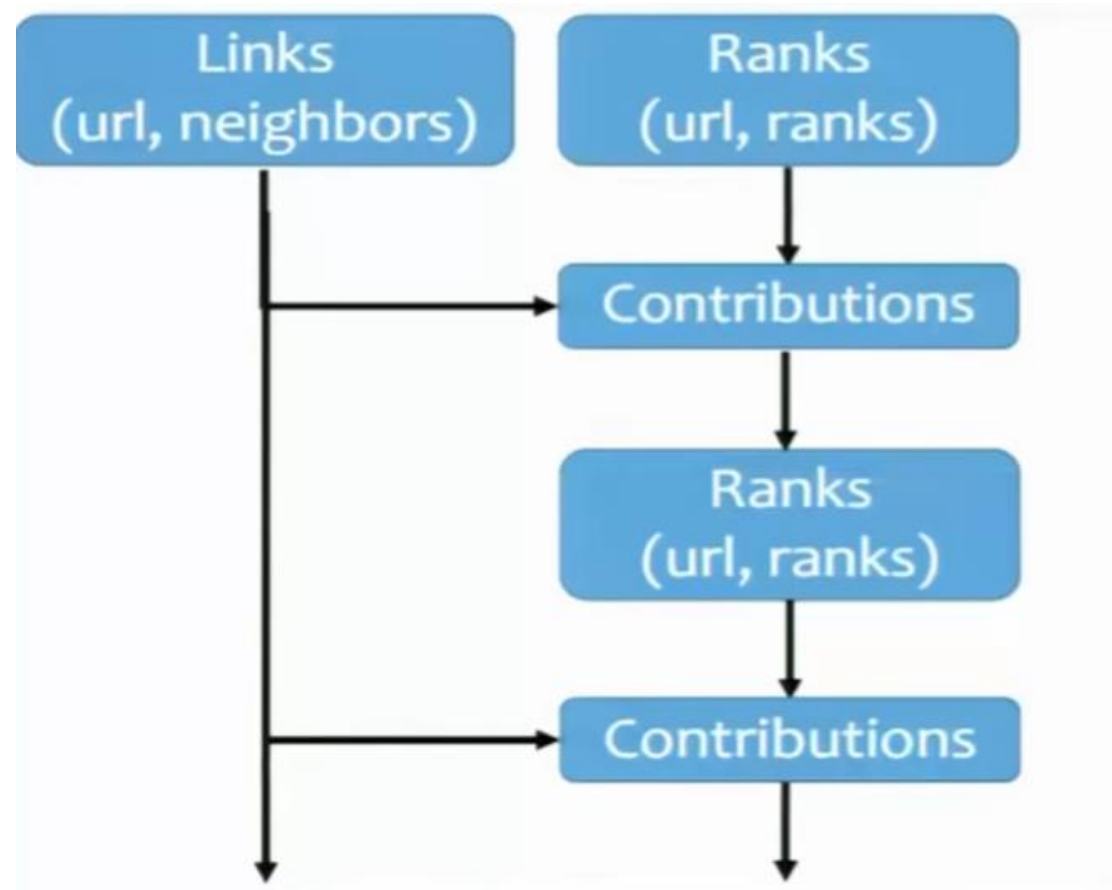


- The RDDs track the graph of transformations that built them (their lineage) to rebuild lost data

# What we can do with Spark?



- RDD Operations:
  - Transformations e.g., filter, join, map, group by,.....
  - Actions e.g., count, print,.....
- Control:
  - Partition: Spark also gives you control over how you can partition your RDDs
  - Persistence: Allows you to choose whether you want to persist RDD on to disk or not

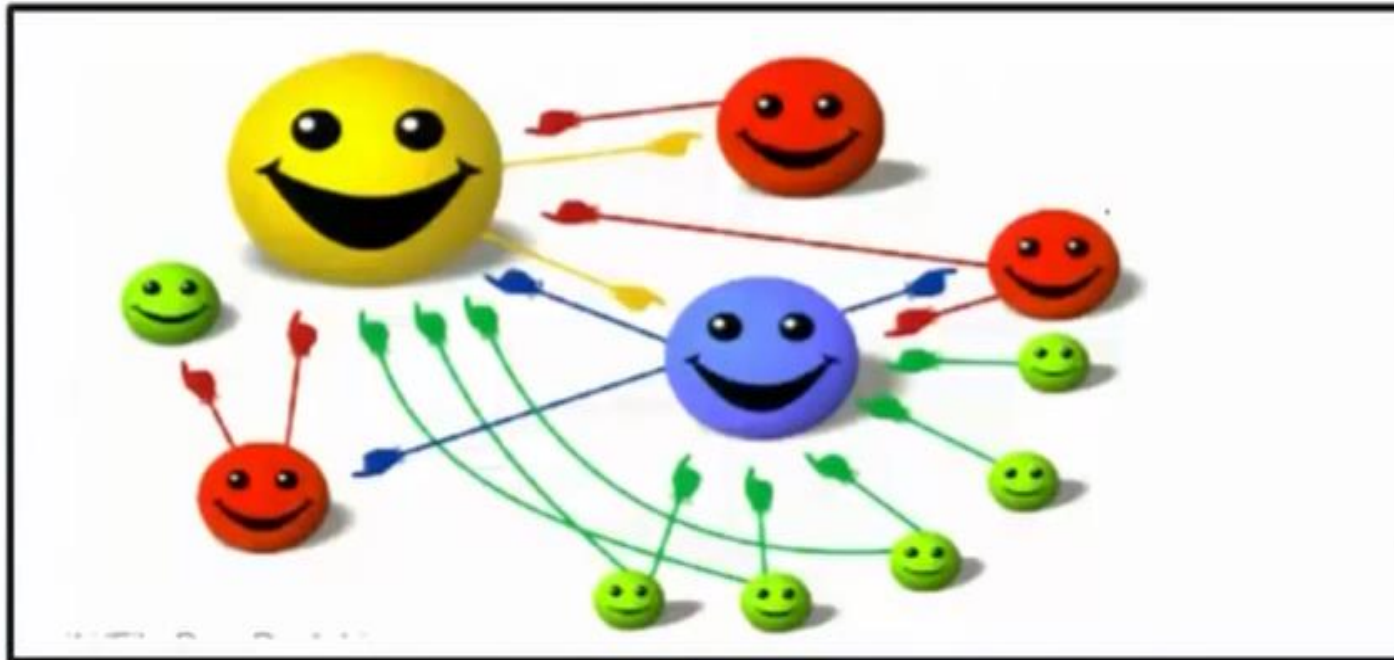
# Partitioning: PageRank



- Join takes place repeatedly
- Good partitioning reduces shuffles

# Example: PageRank

- Give page ranks (scores) based on links to them
- Links from many pages  High rank
- Links from high rank pages  High rank



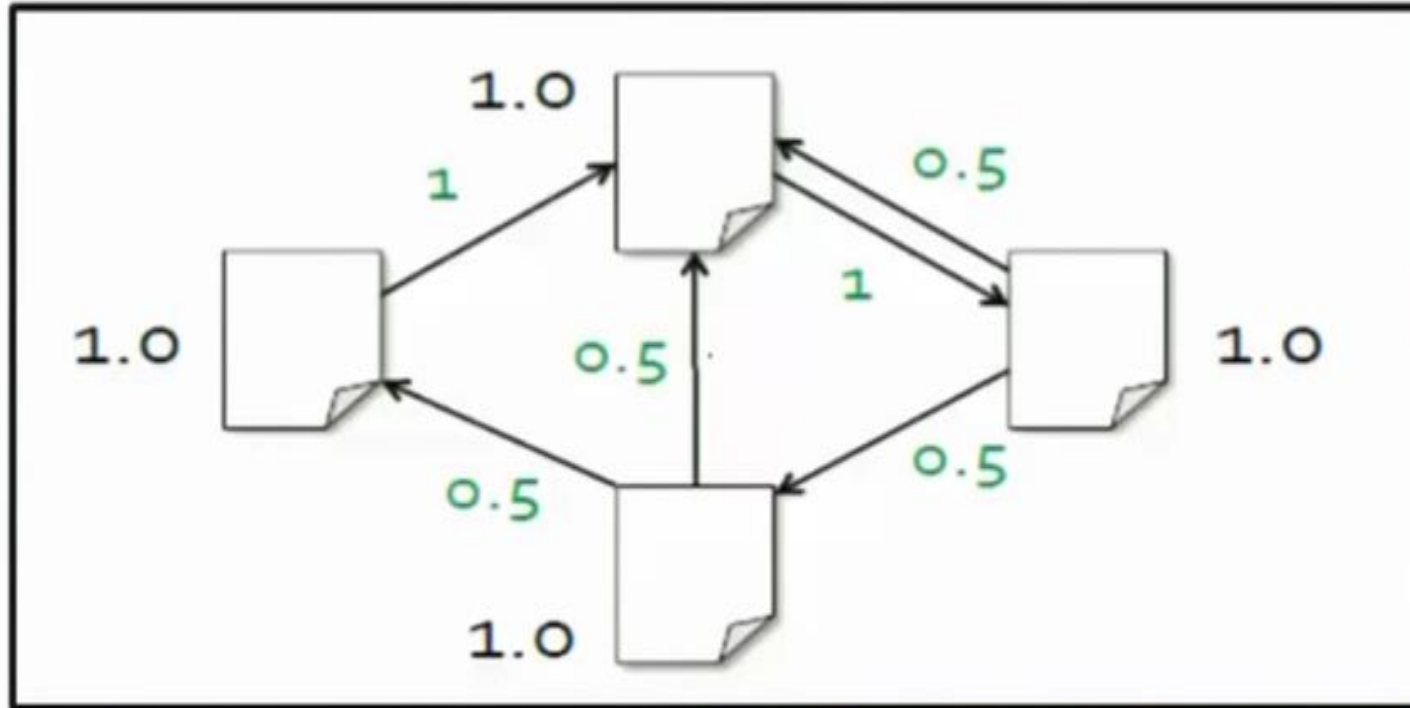


# Algorithm

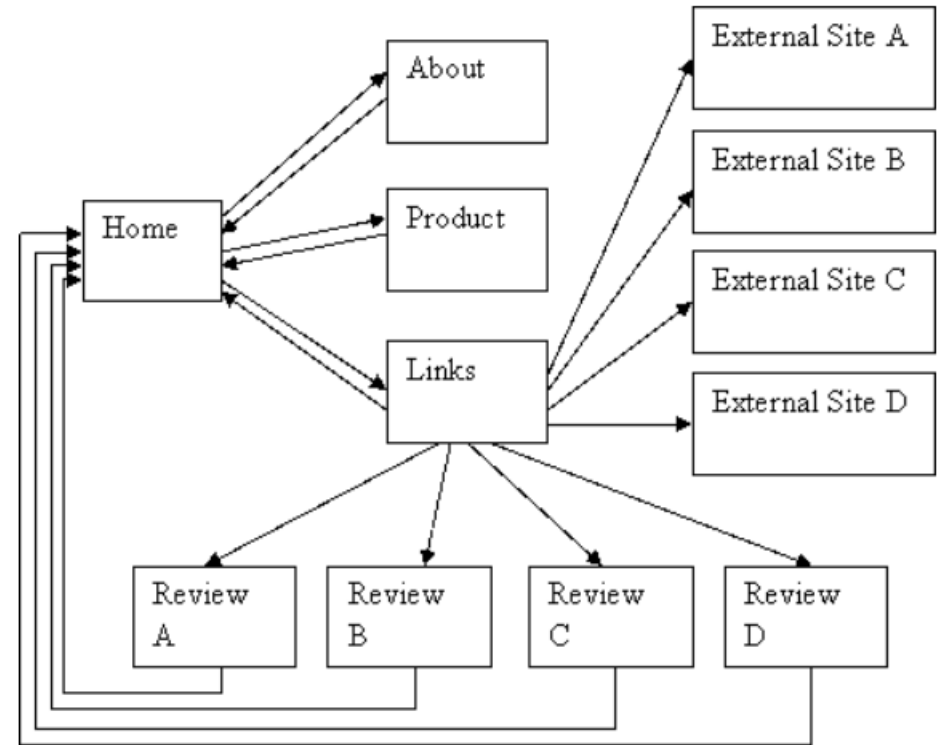
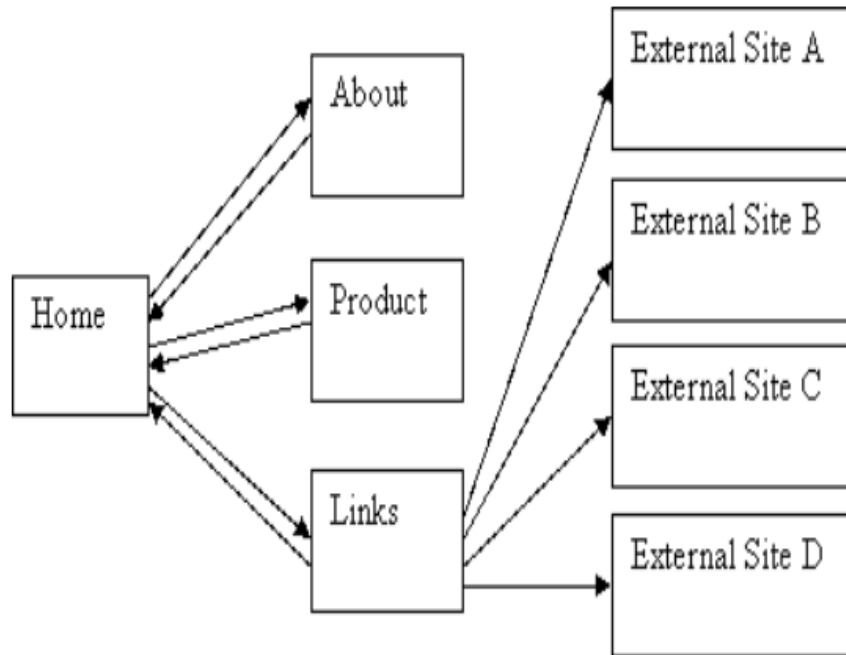
**Step-1:** Start each page at a rank of 1

**Step-2:** On each iteration, have page  $p$  contribute  $\text{rank}_p / |\text{neighbors}_p|$  to its neighbors

**Step-3:** Set each page's rank to  $0.15 + 0.85 * \text{contributions}$

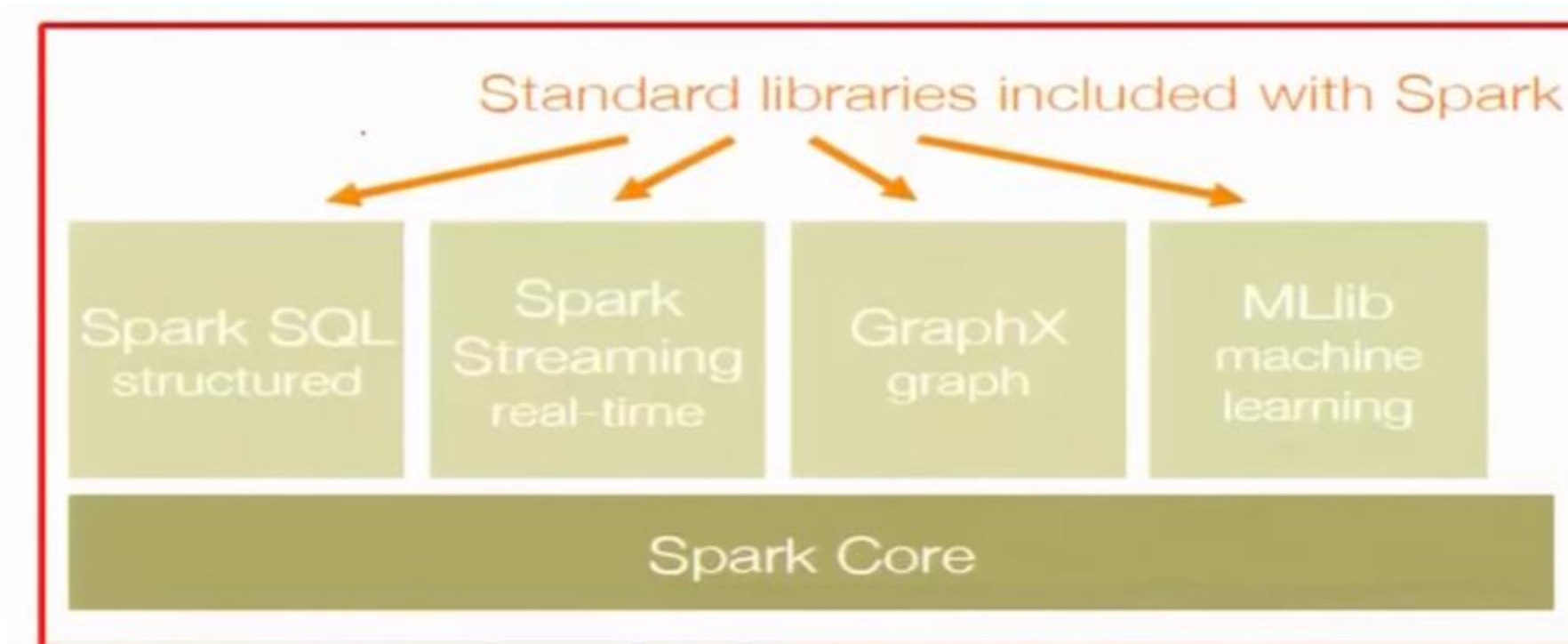


Q- Calculate the page ranks of each pages shown below?



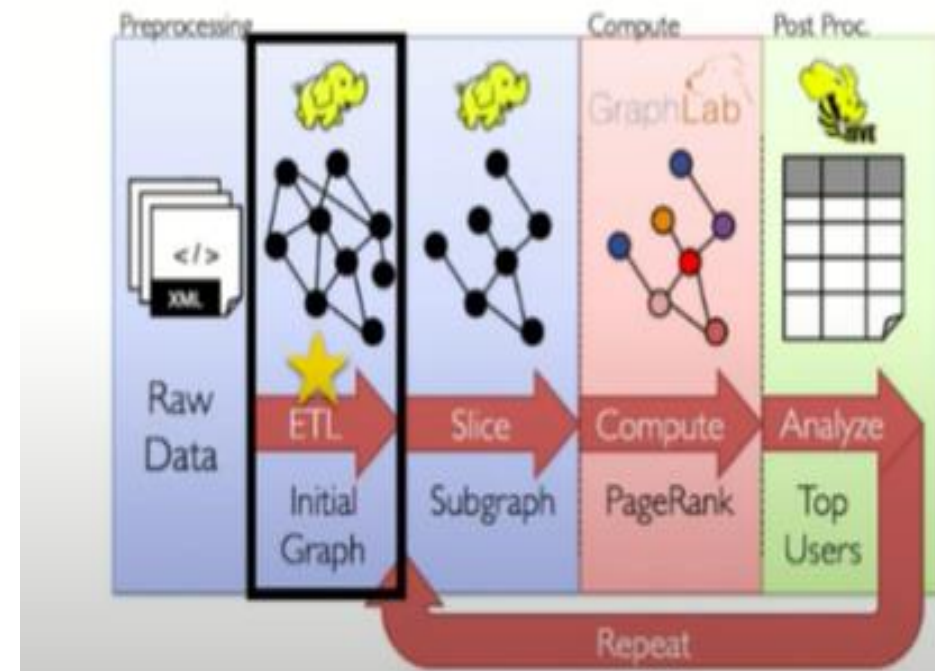
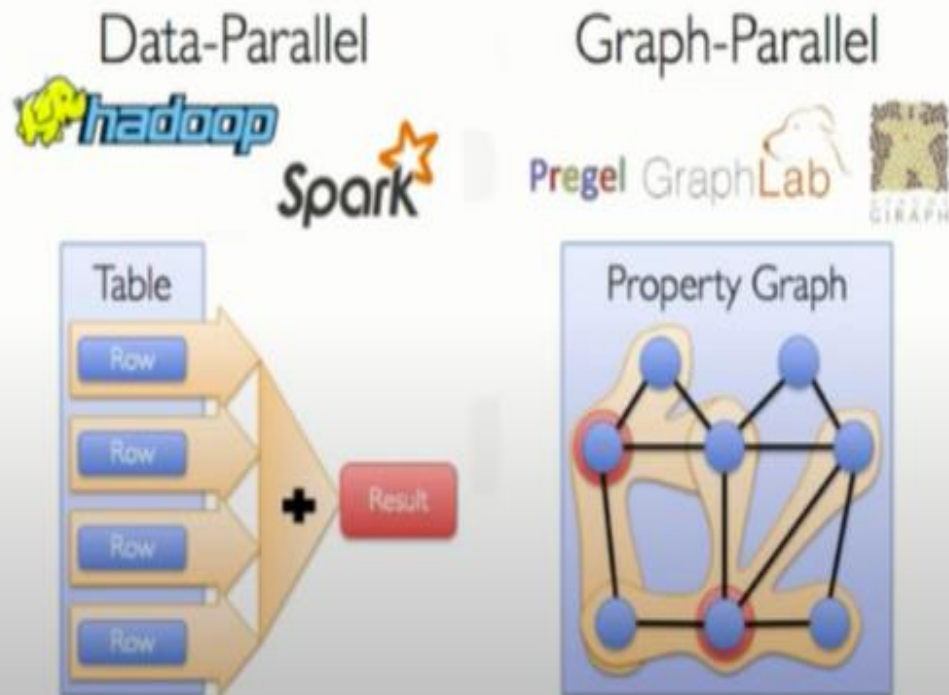
# Generality

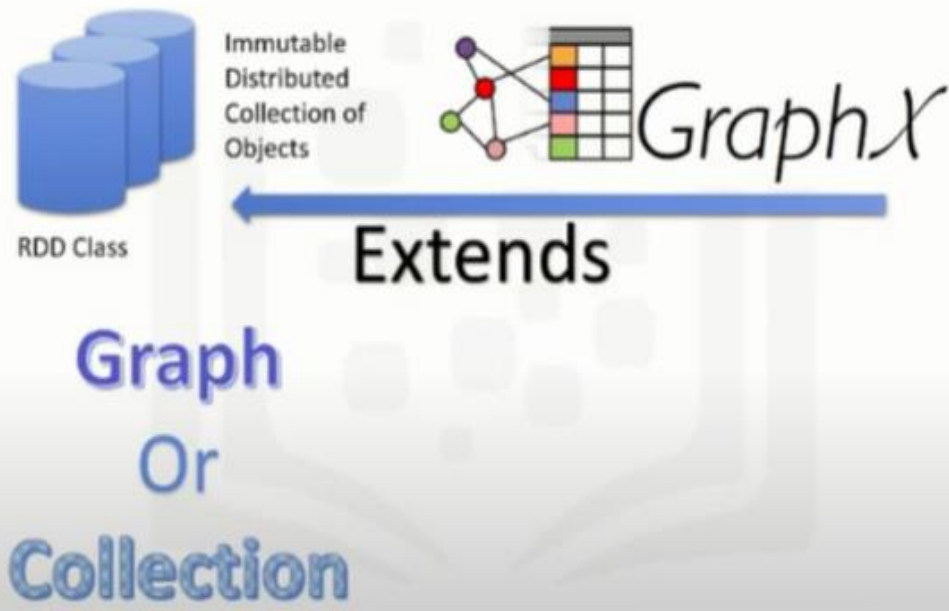
- RDDs allow unification of different programming models
  - Stream processing
  - Graph processing
  - Machine learning



# GraphX

- GraphX is the Apache Spark component for graph parallel computations, built upon a branch of mathematics called graph theory. It is a distributed graph processing framework that sits on top of spark core.



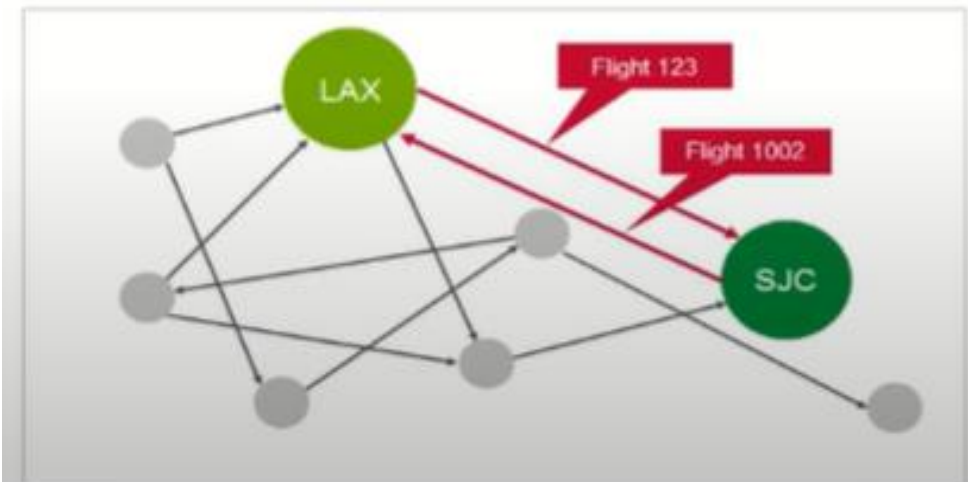


## GraphX Property graph:

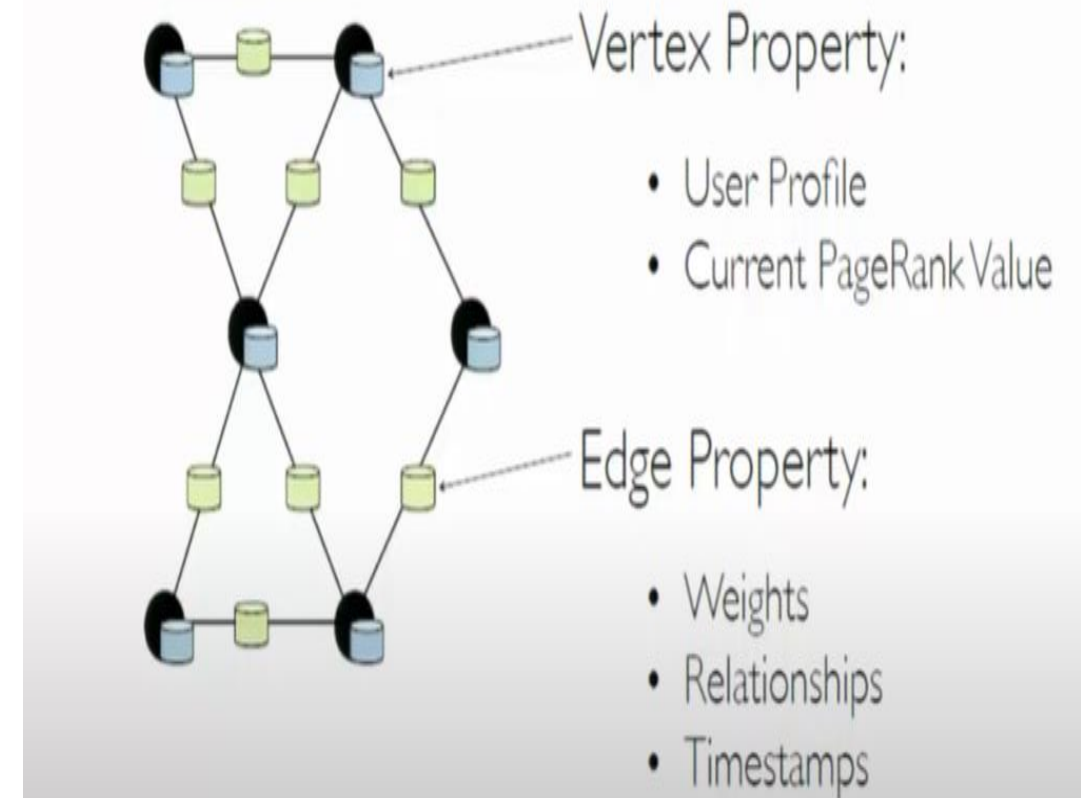
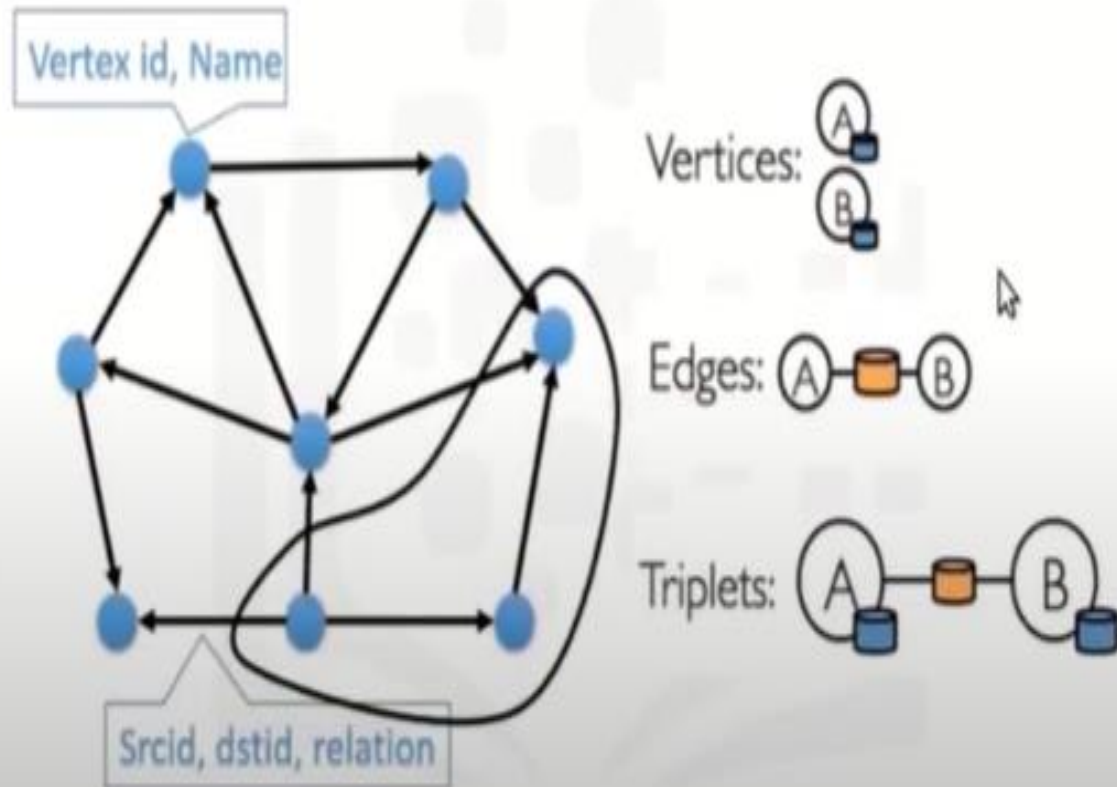
- GraphX extends the Spark RDD with a Resilient Distributed Property Graph.
- The property graph is a directed multigraph which can have multiple edges in parallel. Every edge and vertex has user-defined properties associated with it. The parallel edges allow multiple relationships between the same vertices.

## Basic Graph Properties:

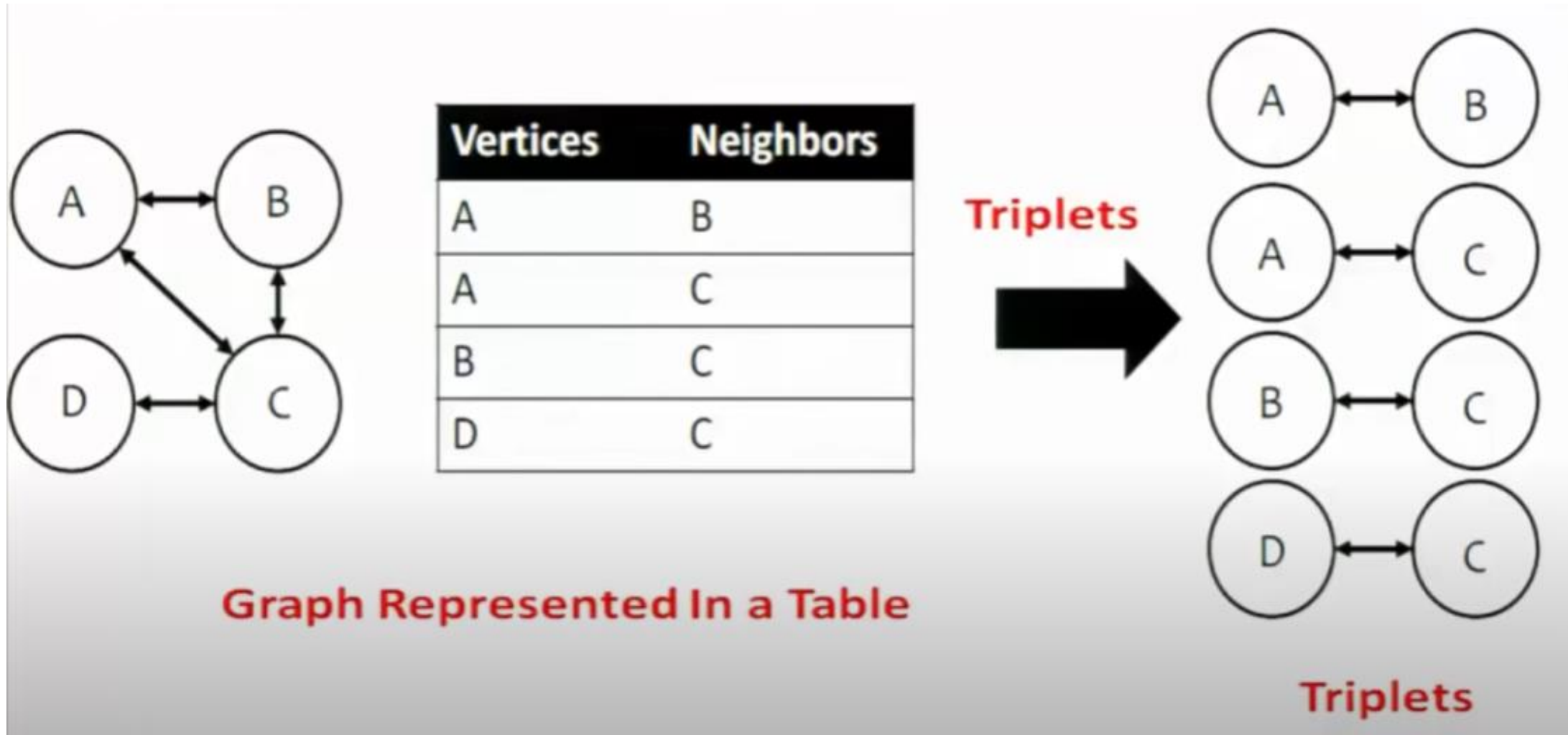
- Relationship between objects
- Vertices and edges
- Directed graph
- Regular graph



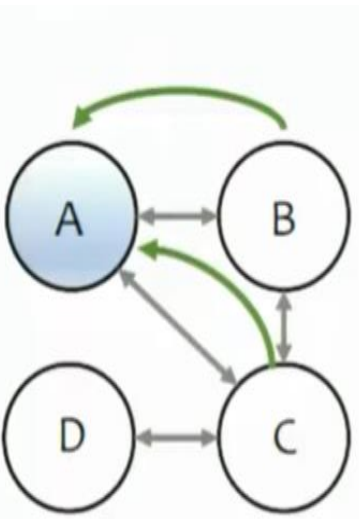
# Components of GraphX (Property Graph)



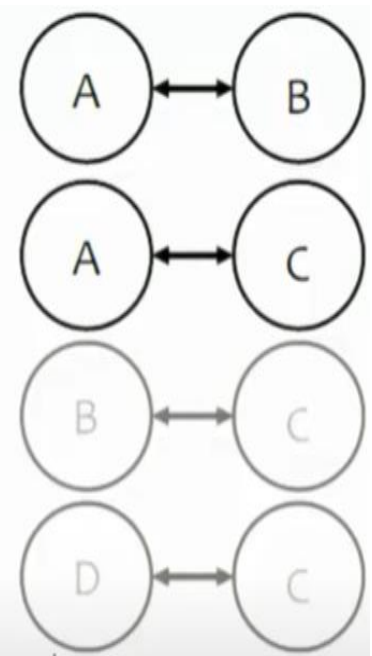
# Gather-Apply-Scatter-on GraphX



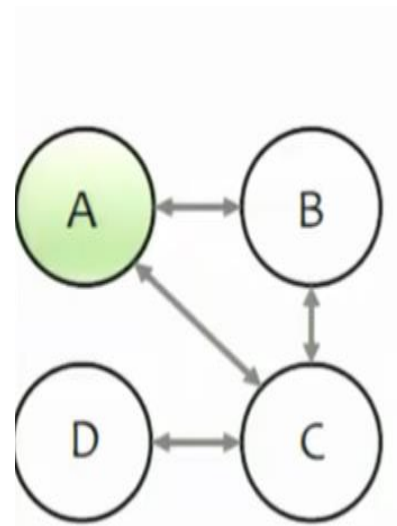




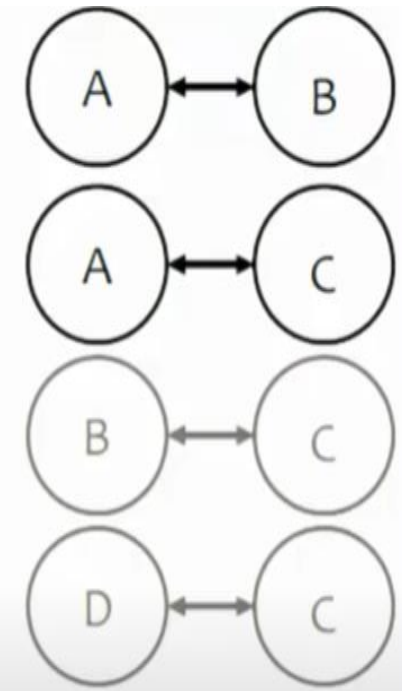
Gather at A



Group-By A

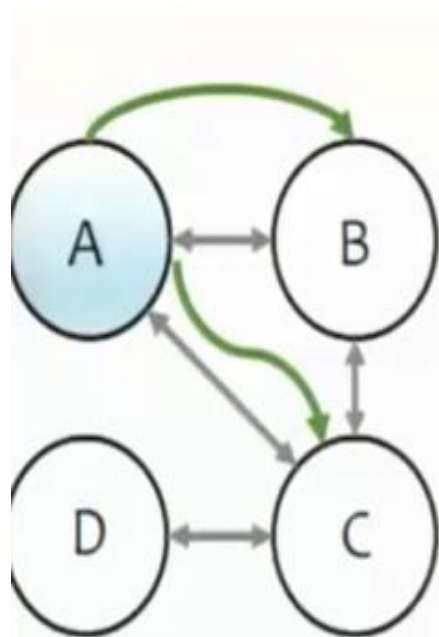


Apply

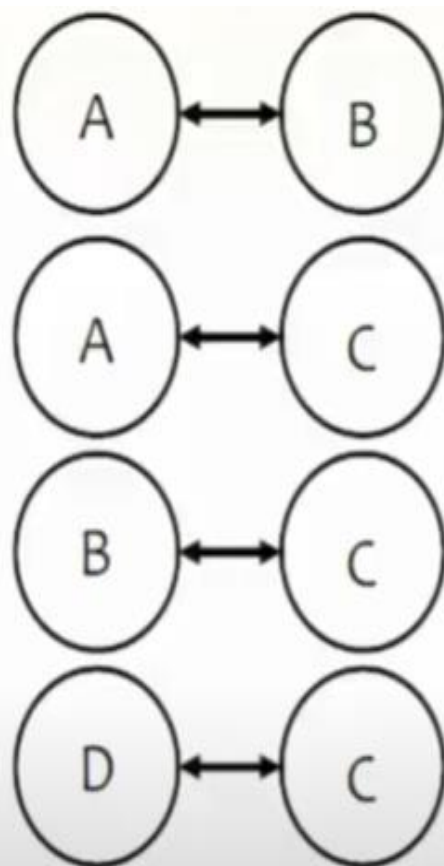


Map

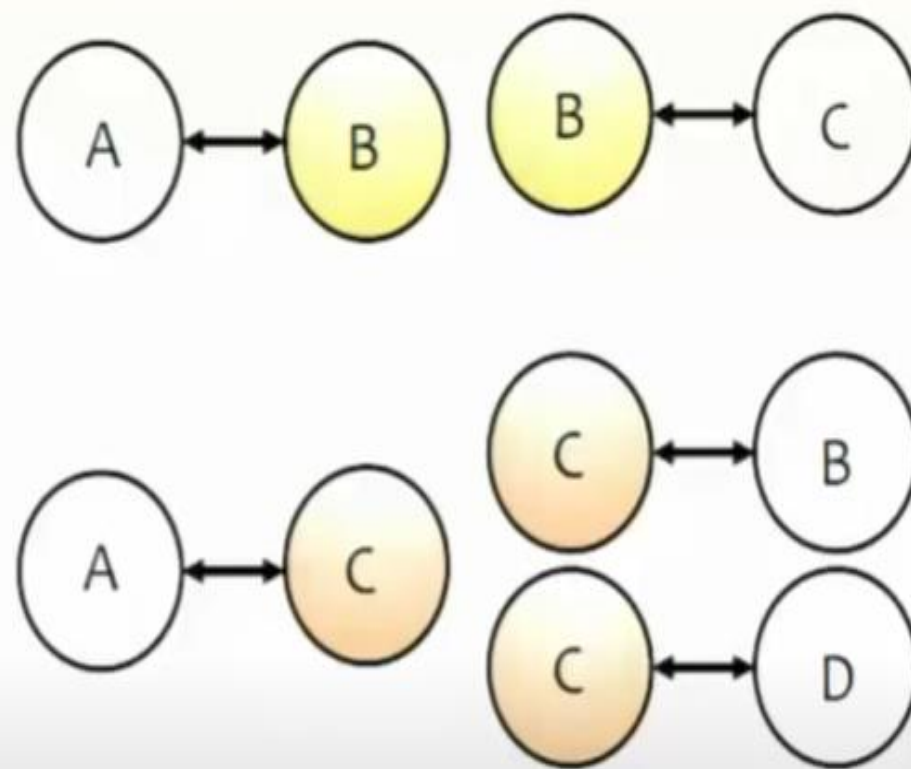




**Scatter**



**Triplets**



**Join**

# Creating a Graph (Scala)

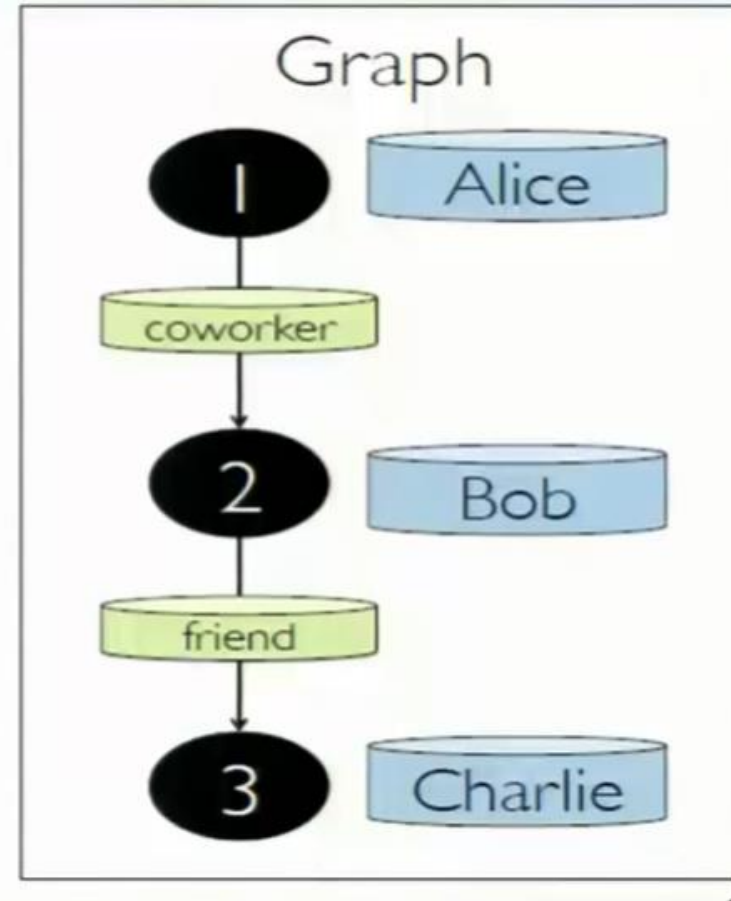
```
type VertexId = Long

val vertices: RDD[(VertexId, String)] =
  sc.parallelize(List(
    (1L, "Alice"),
    (2L, "Bob"),
    (3L, "Charlie")))

class Edge[ED](
  val srcId: VertexId,
  val dstId: VertexId,
  val attr: ED)

val edges: RDD[Edge[String]] =
  sc.parallelize(List(
    Edge(1L, 2L, "coworker"),
    Edge(2L, 3L, "friend")))

val graph = Graph(vertices, edges)
```



- Installation and getting started with Scala: <https://docs.scala-lang.org/getting-started/index.html>

```

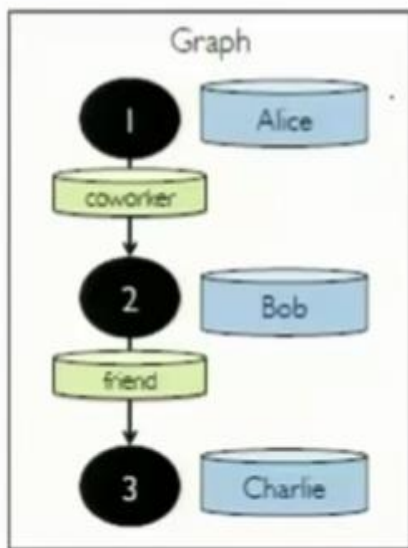
class Graph[VD, ED] {
  // Table Views -----
  def vertices: RDD[(VertexId, VD)]
  def edges: RDD[Edge[ED]]
  def triplets: RDD[EdgeTriplet[VD, ED]]
  // Transformations -----
  def mapVertices[VD2](f: (VertexId, VD) => VD2): Graph[VD2, ED]
  def mapEdges[ED2](f: Edge[ED] => ED2): Graph[VD2, ED]
  def reverse: Graph[VD, ED]
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,
               vpred: (VertexId, VD) => Boolean): Graph[VD, ED]
  // Joins -----
  def outerJoinVertices[U, VD2]
    (tbl: RDD[(VertexId, U)])
    (f: (VertexId, VD, Option[U]) => VD2): Graph[VD2, ED]
  // Computation -----
  def aggregateMessages[A](
    sendMsg: EdgeContext[VD, ED, A] => Unit,
    mergeMsg: (A, A) => A): RDD[(VertexId, A)]
}

```

# The Triplet View

```
class Graph[VD, ED] {  
  def triplets: RDD[EdgeTriplet[VD, ED]]  
}
```

```
class EdgeTriplet[VD, ED](  
  val srcId: VertexId, val dstId: VertexId, val attr: ED,  
  val srcAttr: VD, val dstAttr: VD)
```



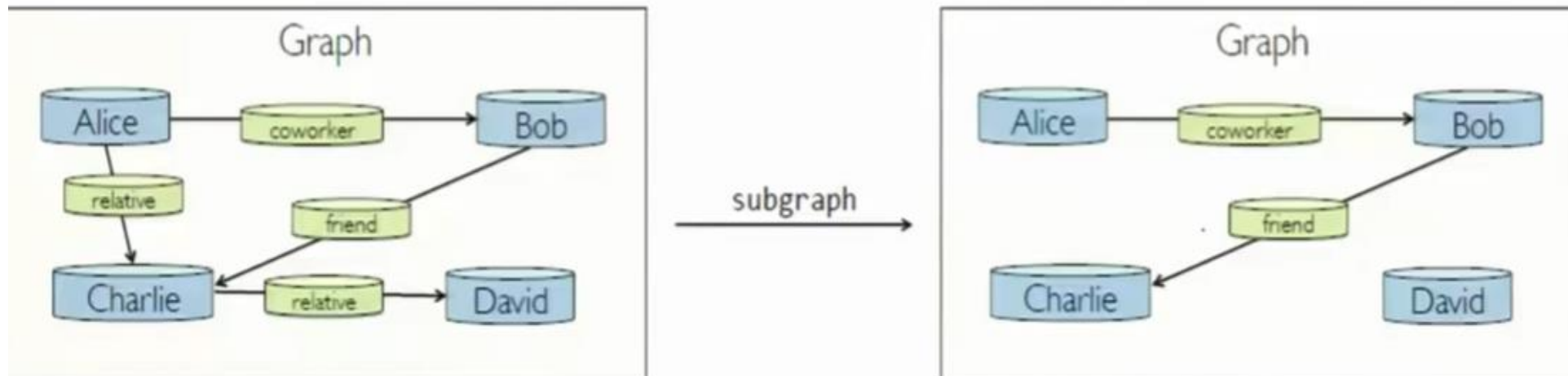
triplets →

RDD		
srcAttr	dstAttr	attr
Alice	coworker	Bob
Bob	friend	Charlie

# The Subgraph Transformation

```
class Graph[VD, ED] {  
  def subgraph(epred: EdgeTriplet[VD, ED] => Boolean,  
              vpred: (VertexId, VD) => Boolean): Graph[VD, ED]  
}
```

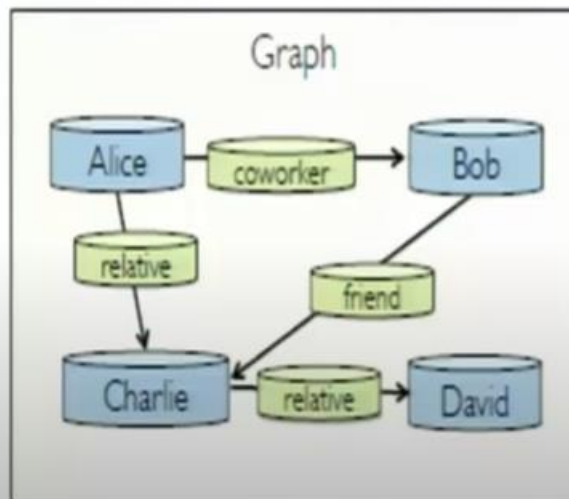
```
graph.subgraph(epred = (edge) => edge.attr != "relative")
```





# Computation with aggregated messages

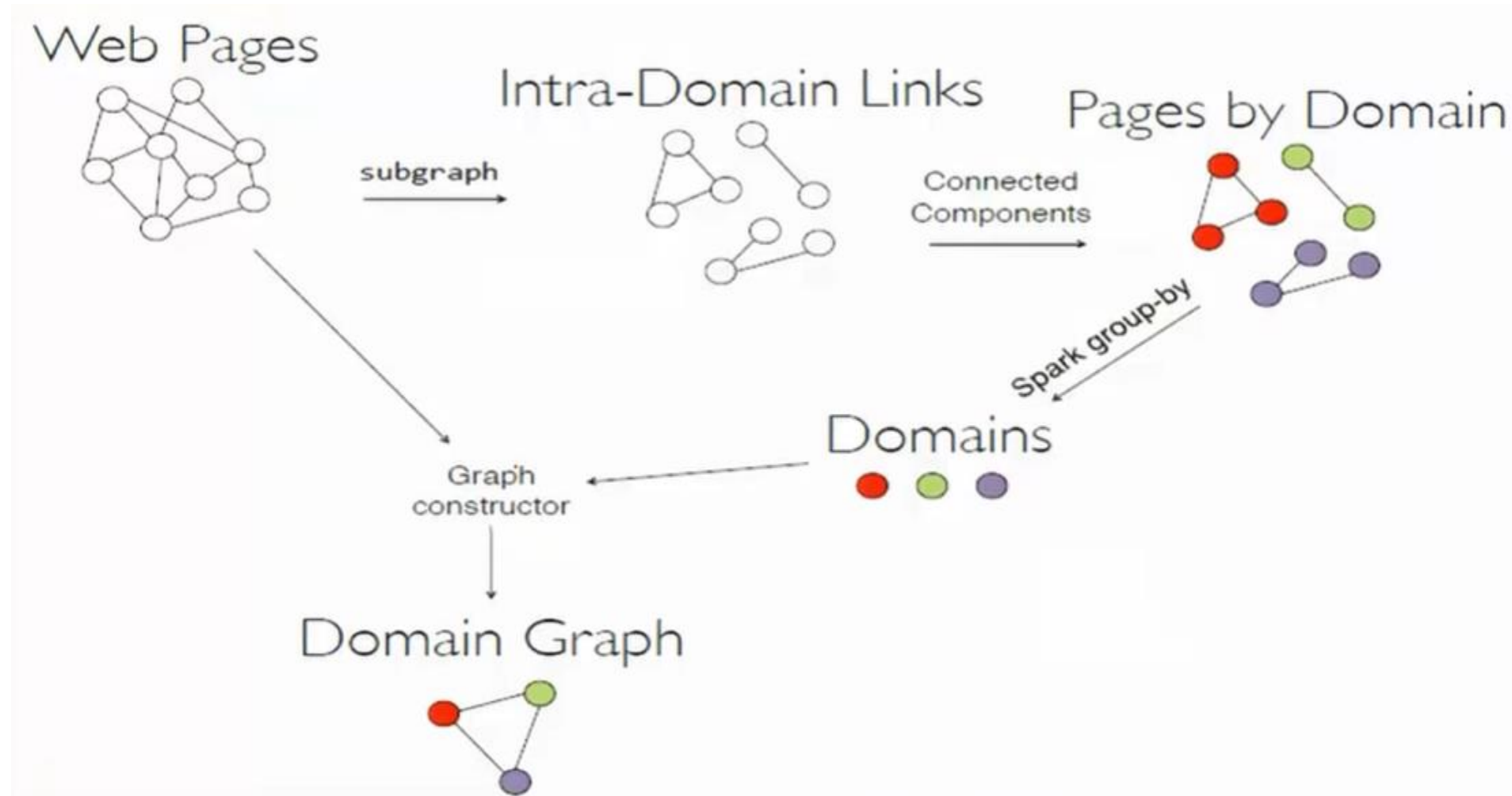
```
class Graph[VD, ED] {  
  def aggregateMessages[A](  
    sendMsg: EdgeContext[VD, ED, A] => Unit,  
    mergeMsg: (A, A) => A): RDD[(VertexId, A)]  
  }  
  
class EdgeContext[VD, ED, A](  
  val srcId: VertexId, val dstId: VertexId, val attr: ED,  
  val srcAttr: VD, val dstAttr: VD) {  
  def sendToSrc(msg: A)  
  def sendToDst(msg: A)  
}  
  
graph.aggregateMessage  
  ctx => {  
    ctx.sendToSrc(1)  
    ctx.sendToDst(1)  
  },  
  _ + _)
```



aggregateMessages

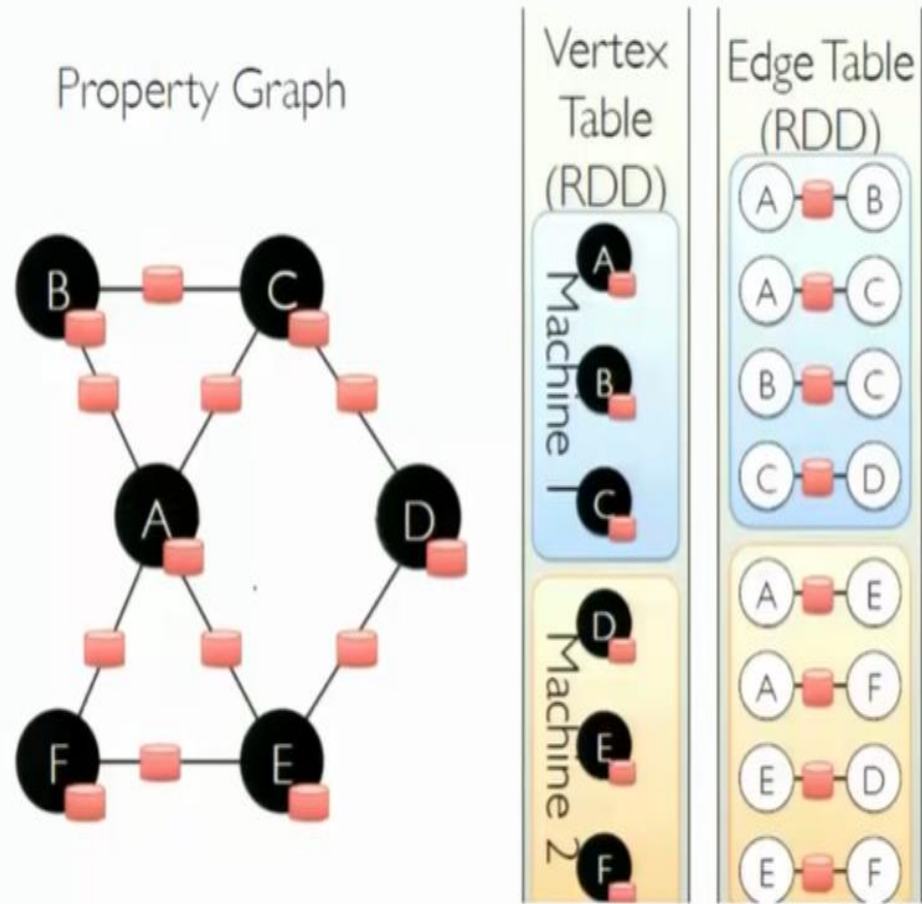
RDD	
vertex id	degree
Alice	2
Bob	2
Charlie	3
David	1

# Example: Graph Coarsening



# How GraphX works?

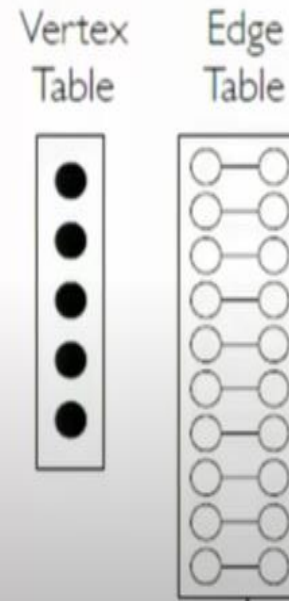
- Storing graphs as tables



- Simple operations

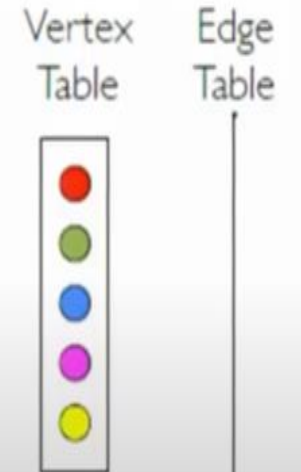
## Reuse vertices or edges across multiple graphs

Input Graph



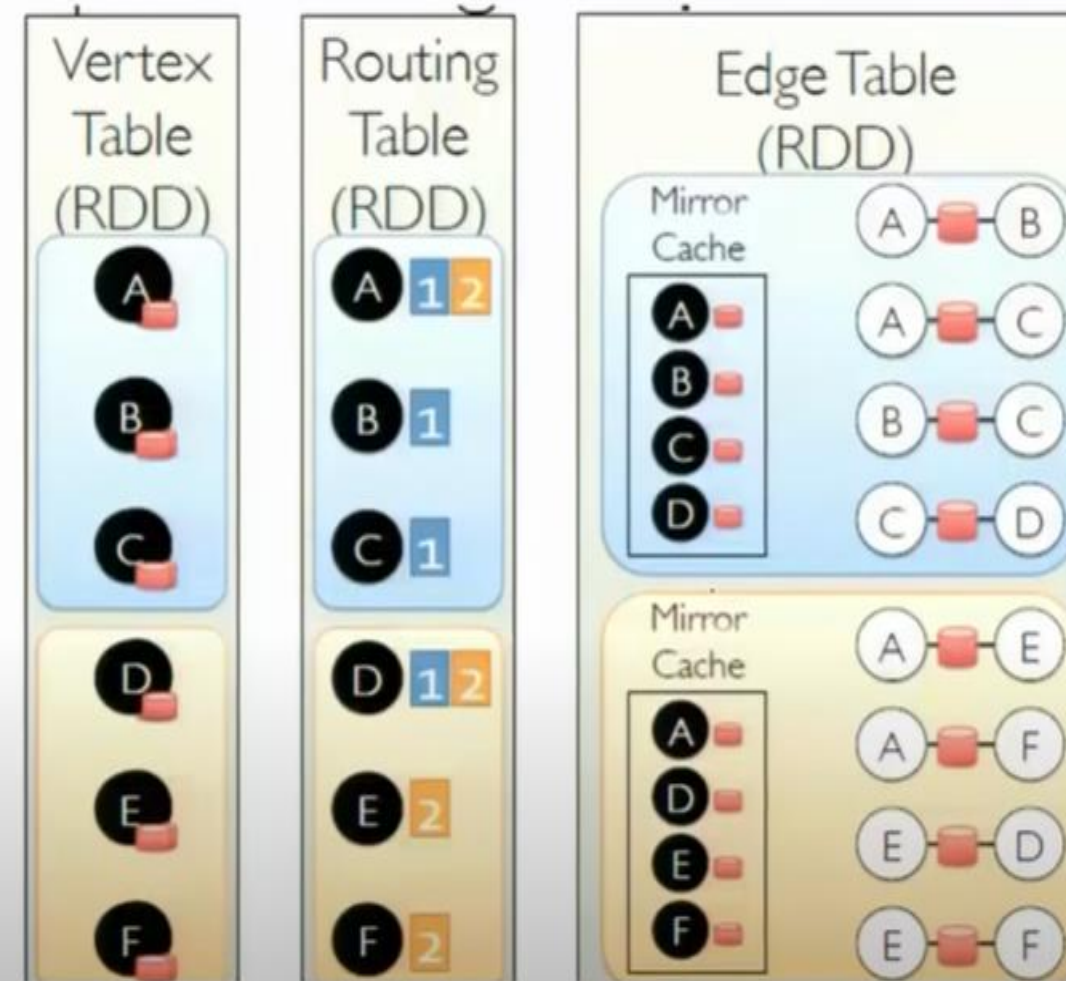
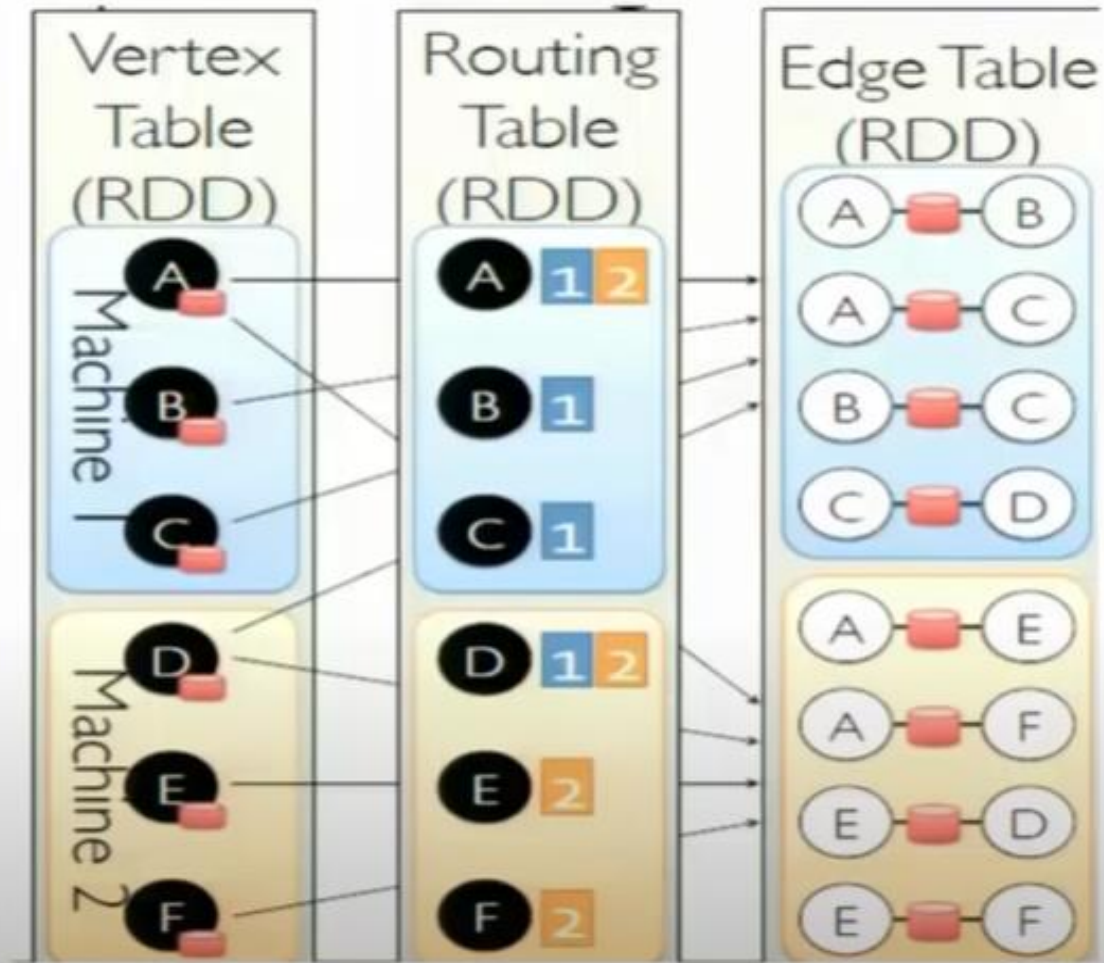
Transform Vertex Properties

Transformed Graph

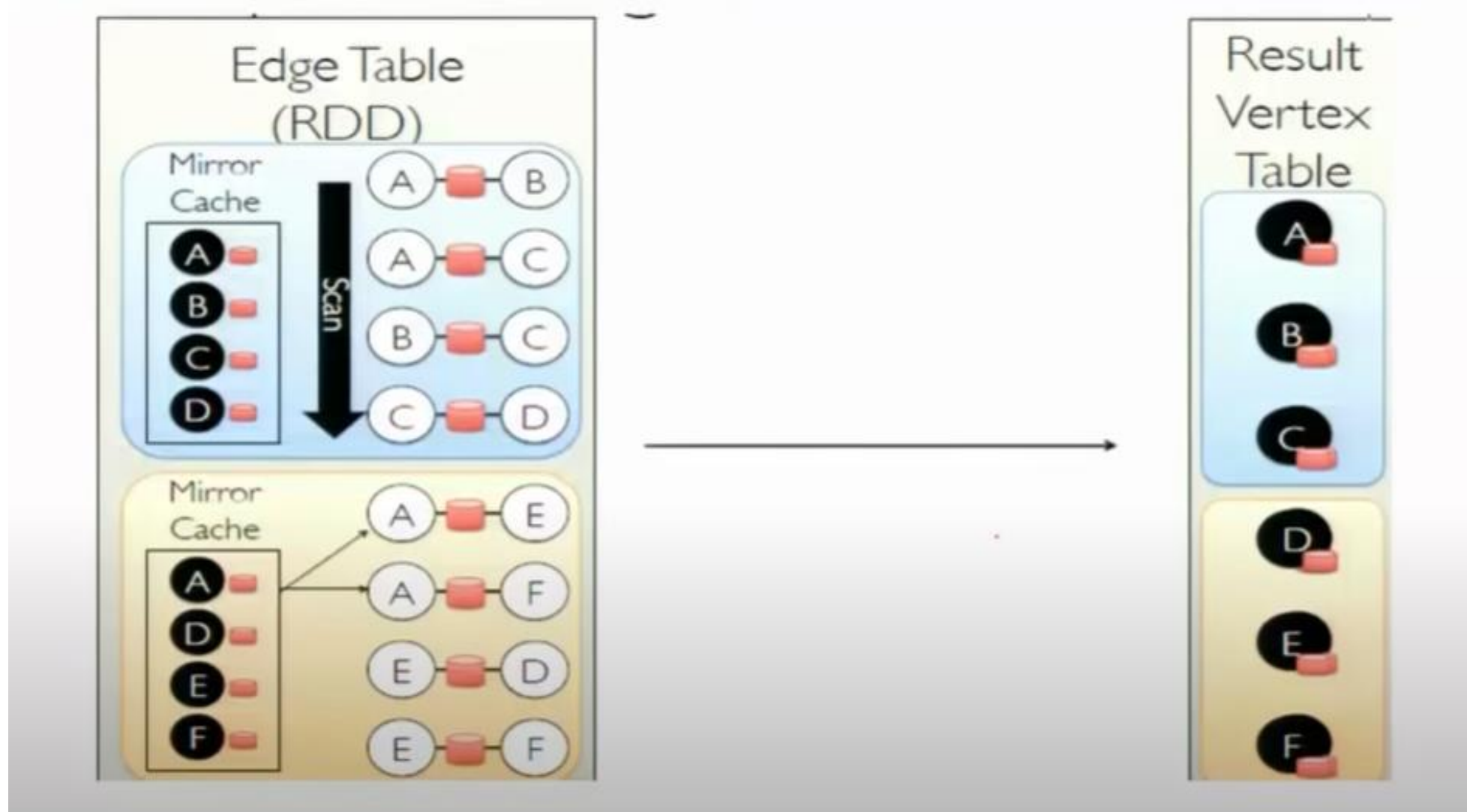




# Implementing the Triplets



# Implementing Aggregate messages



Example: Analyze 3 flights, for each flight we have the following information:

Originating Airport	Destination Airport	Distance
A	B	1800 miles
B	C	800 miles
C	A	1400 miles

Use GraphX to find the routes having > 1000 miles distance?

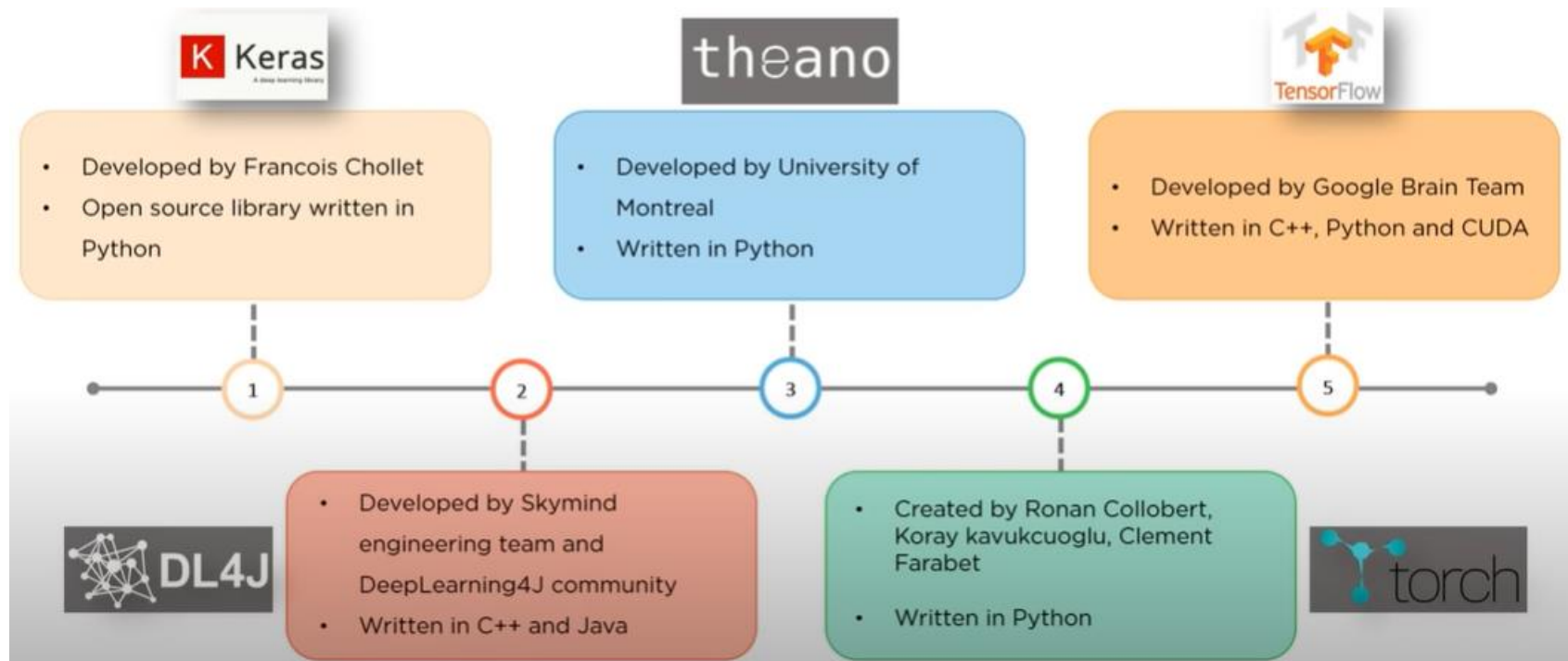
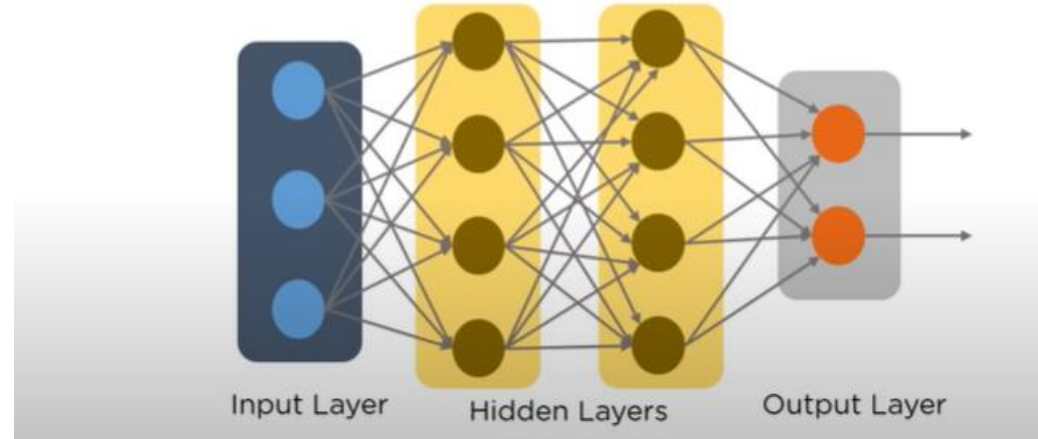
# Other Spark Applications

- Twitter Spam Classification
- EM algorithm for traffic prediction
- K-means clustering
- Alternating Last-squares Matrix factorization
- In Memory OLAP Aggregation on Hive Data
- SQL on Spark

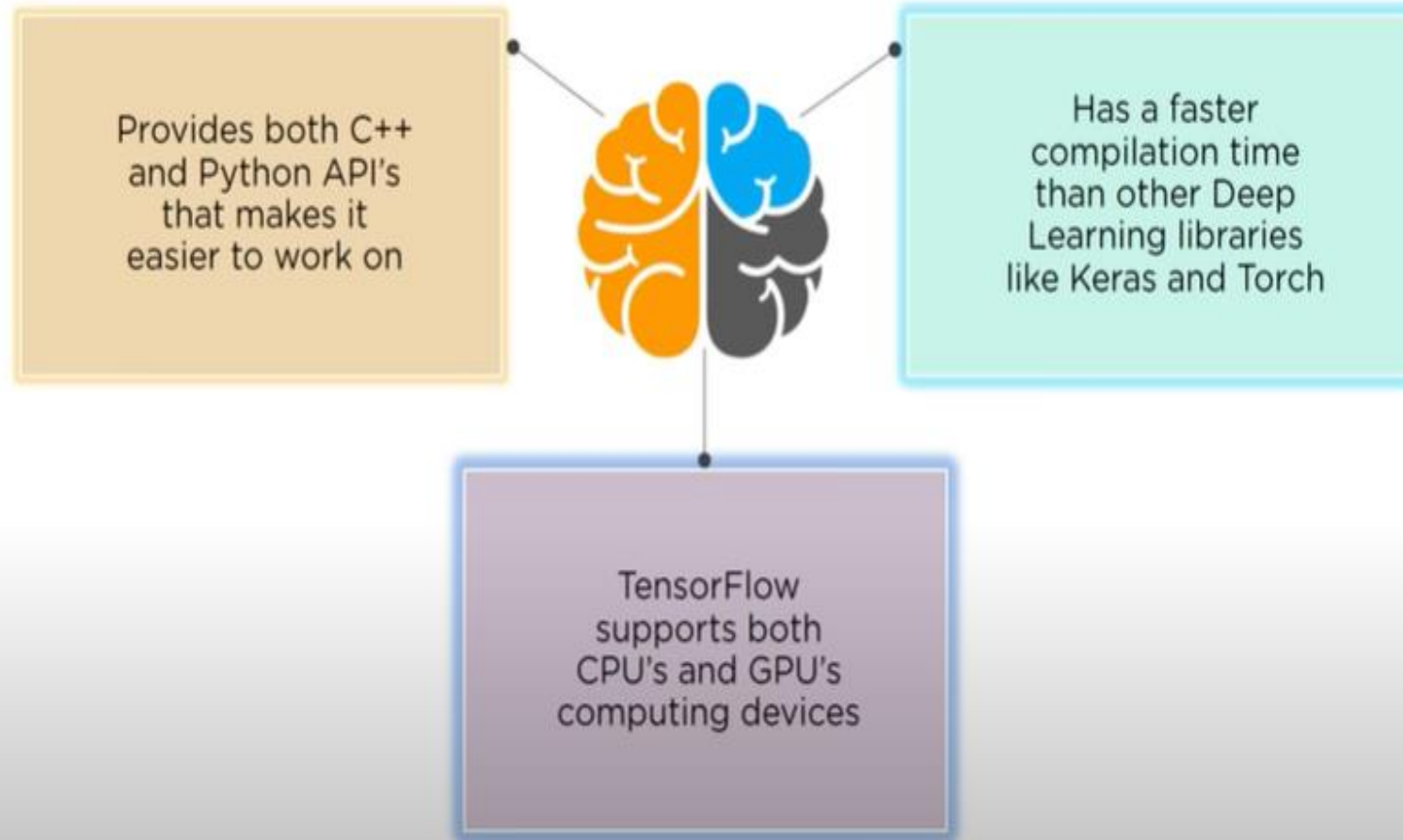
# Tensor Flow



- Used as a programming model in the deep learning applications



# Why Tensor Flow?



# What is Tensor Flow?

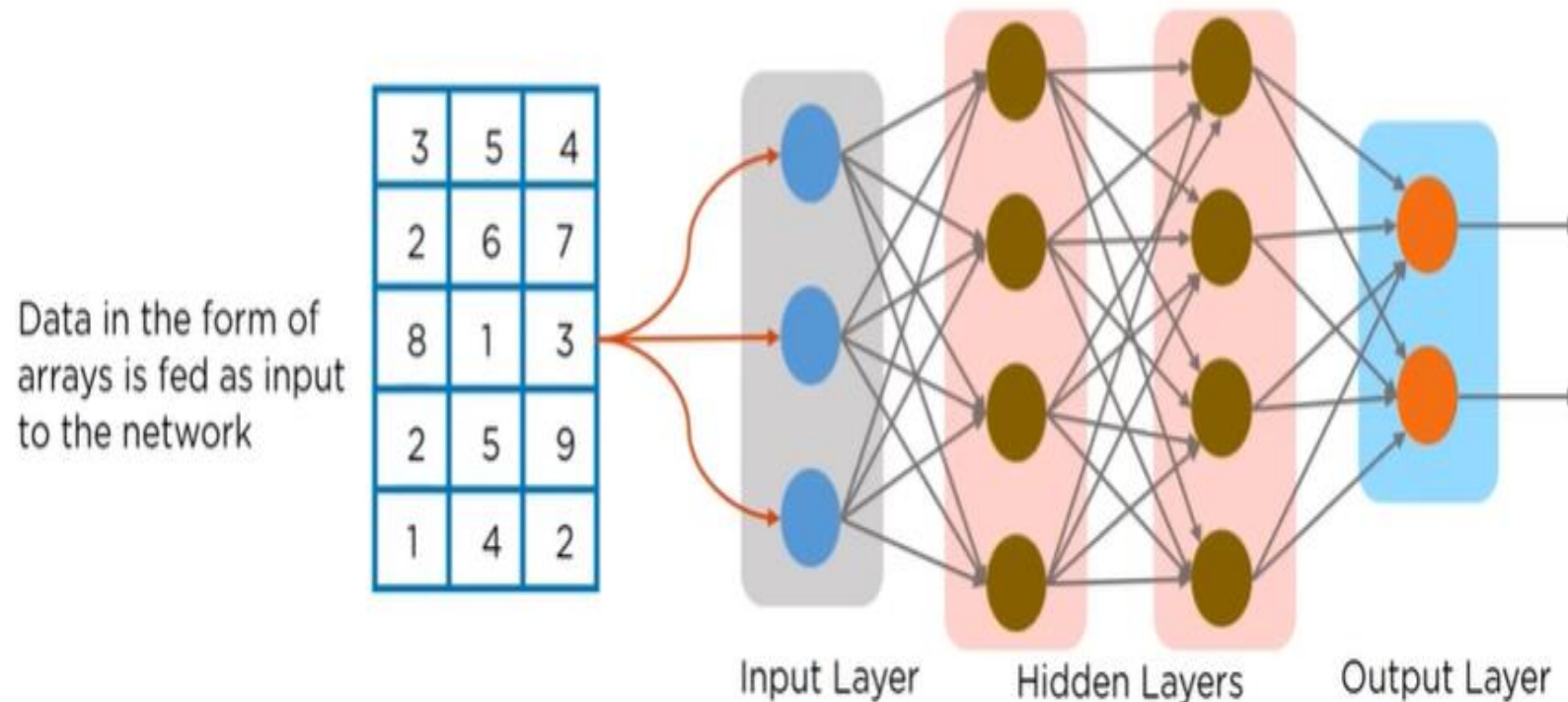
- Tensor Flow is an open source library developed by Google.
- Developed originally to run large numerical computations.
- Accepts data in the form of multi-dimensional arrays of higher dimensions called Tensors.
- Works on the basis of Data Flow graphs that have nodes and edges





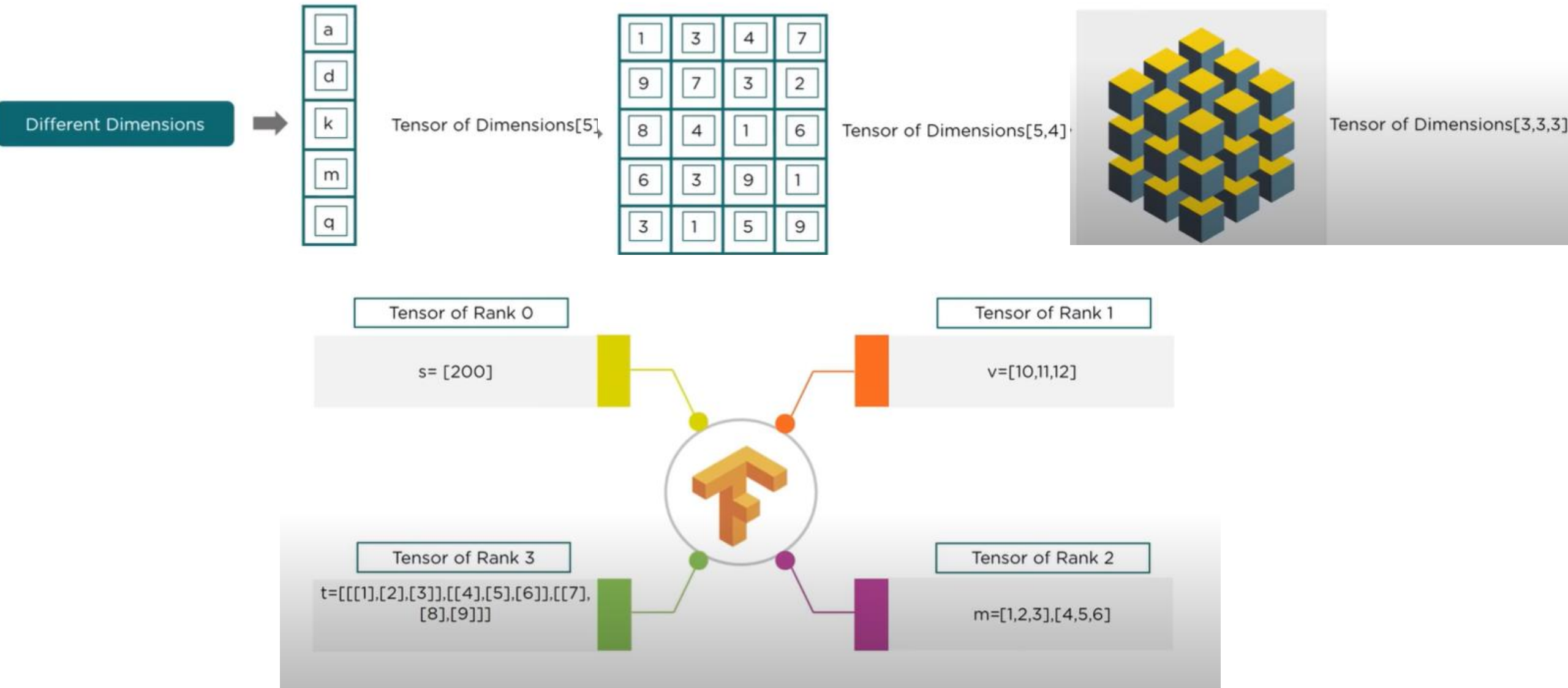
# What are Tensors?

- Tensor is a generalization of vectors and matrices of potentially higher dimensions.
- Arrays of data of different dimensions and ranks that are fed as input to the neural network are called Tensors.

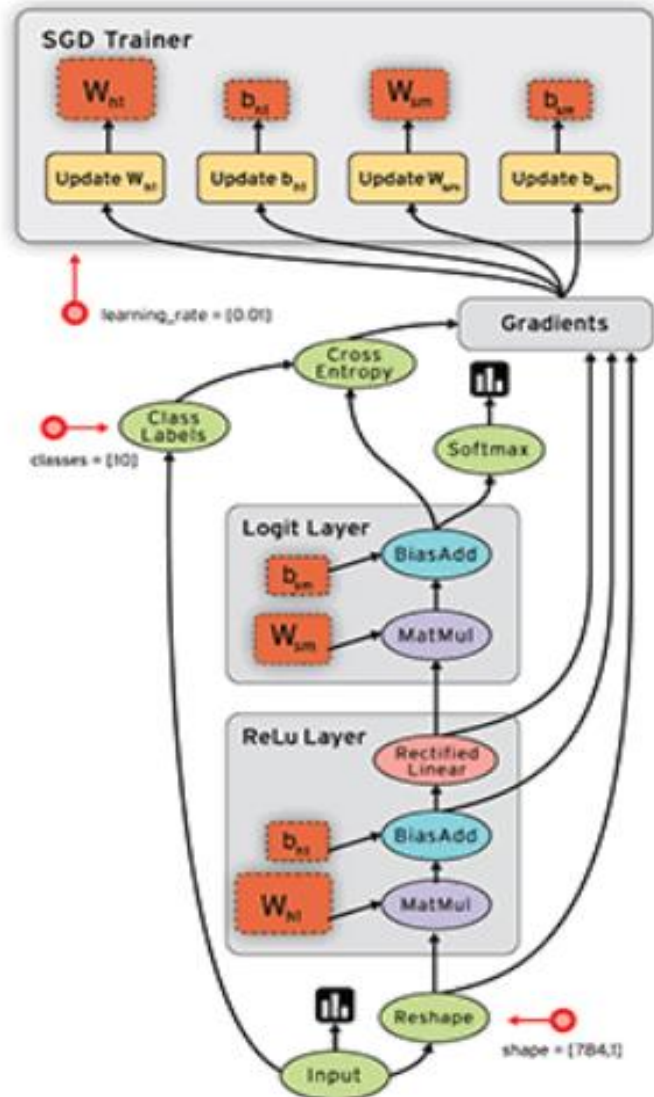




# Tensor Dimensions and Ranks



# Data Flow Graph



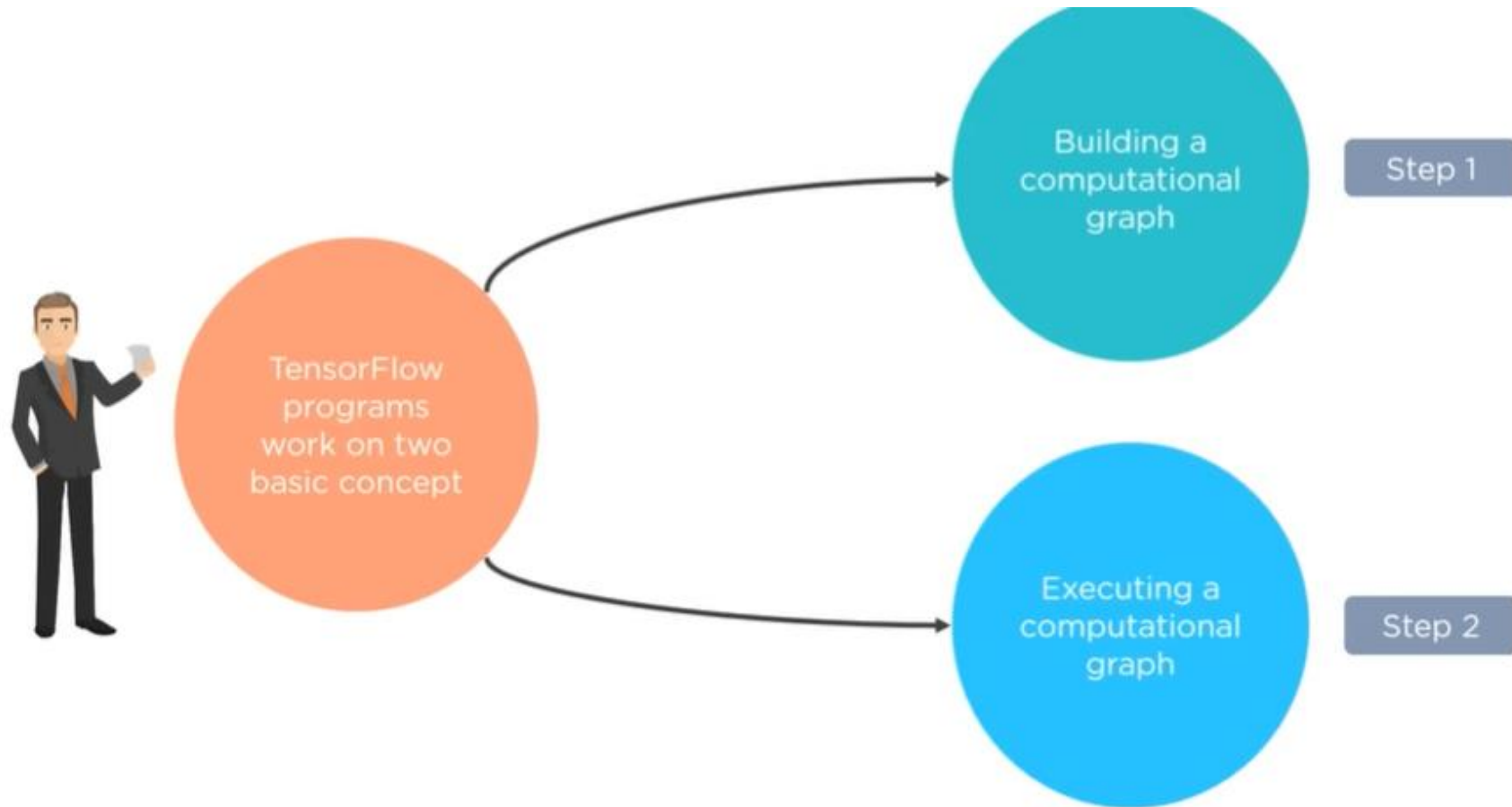
Each computation in tensor Flow is represented as a data flow graph

Each node in the graph represents a mathematical operation (add, subtract, multiply etc) and each edge represents multi-dimensional arrays (Tensors)

Computational graph is the graph of programming logic which Tensor Flow builds in the memory

Enables creating large scale neural networks as computing can be distributed across several CPU's or GPU's

# Program Elements in Tensor Flow



# Program Elements in Tensor Flow

## Constants

*Constants* are parameters whose value does not change. To define a constant we use `tf.constant()` command.

Example:

```
a = tf.constant(2.0, tf.float32)
b = tf.constant(3.0)
Print(a, b)
```

## Variable

*Variables* allow us to add new trainable parameters to graph. To define a variable we use `tf.Variable()` command and initialize them before running the graph in a session.

Example:

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W*x+b
```

## Placeholder

*Placeholders* allow us to feed data to a tensorflow model from outside a model. It permits a value to be assigned later. To define a placeholder we use `tf.placeholder()` command.

Example:

```
a = tf.placeholder(tf.float32)
b = a*2
with tf.Session() as sess:
    result = sess.run(b, feed_dict={a:3.0})

print result
```

*feed\_dict* specifies tensors that provide concrete values to the placeholders

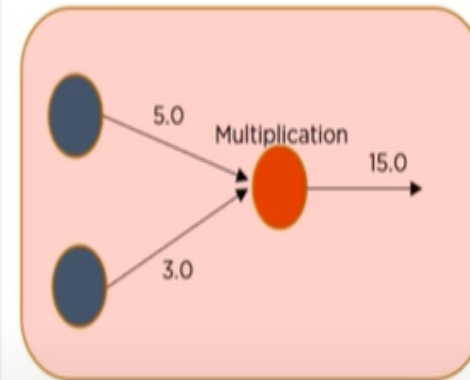
# Program Elements in Tensor Flow

## Session

A *session* is run to evaluate the nodes. This is called as the *TensorFlow runtime*.

Example:

```
a = tf.constant(5.0)
b = tf.constant(3.0)
c = a*b
# Launch Session
sess = tf.Session()
# Evaluate the tensor c
print(sess.run(c))
```



Running a Computation Graph

# Tensor Flow basic programs

## Variables in TensorFlow

```
my_tensor = tf.random_uniform((4,4),0,1)

my_var = tf.Variable(initial_value=my_tensor)

print(my_var)

<tf.Variable 'Variable:0' shape=(4, 4) dtype=float32_ref>
```

You must initialize all global variables

```
init = tf.global_variables_initializer()

init.run()

my_var.eval()

array([[ 0.18764639,  0.76903498,  0.88519645,  0.89911747],
       [ 0.18354201,  0.63433743,  0.42470503,  0.27359927],
       [ 0.45305872,  0.65249109,  0.74132109,  0.19152677],
       [ 0.60576665,  0.71895587,  0.69150388,  0.33336747]], dtype=float32)

sess.run(my_var)

array([[ 0.18764639,  0.76903498,  0.88519645,  0.89911747],
       [ 0.18354201,  0.63433743,  0.42470503,  0.27359927],
       [ 0.45305872,  0.65249109,  0.74132109,  0.19152677],
       [ 0.60576665,  0.71895587,  0.69150388,  0.33336747]], dtype=float32)
```

## Placeholders in TensorFlow

```
ph = tf.placeholder(tf.float64)

ph = tf.placeholder(tf.int32)

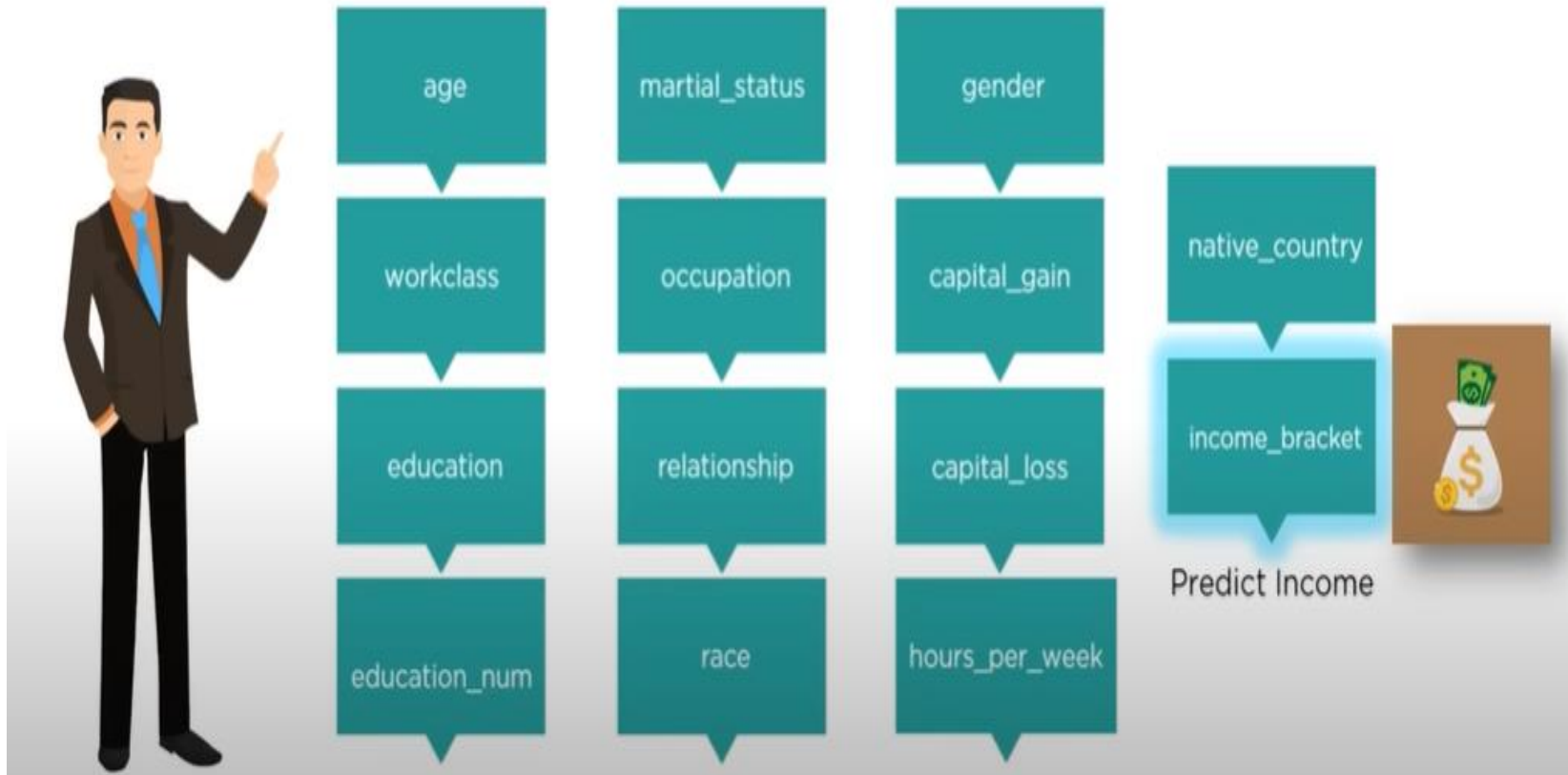
# Its common to use (None,# of Features) for shape as None can be filled by number of samples in the data
ph = tf.placeholder(tf.float32,shape=(None,5))
```

```
b = ph*2

with tf.Session as sess
    result = sess.run(b,feed_dict={ph:4.5})
    print result
```

# Use case implementation using Tensor Flow

Lets use various features of an individual to predict what *class of income* they belong to (>50k or <=50k) using a *Census Data*.



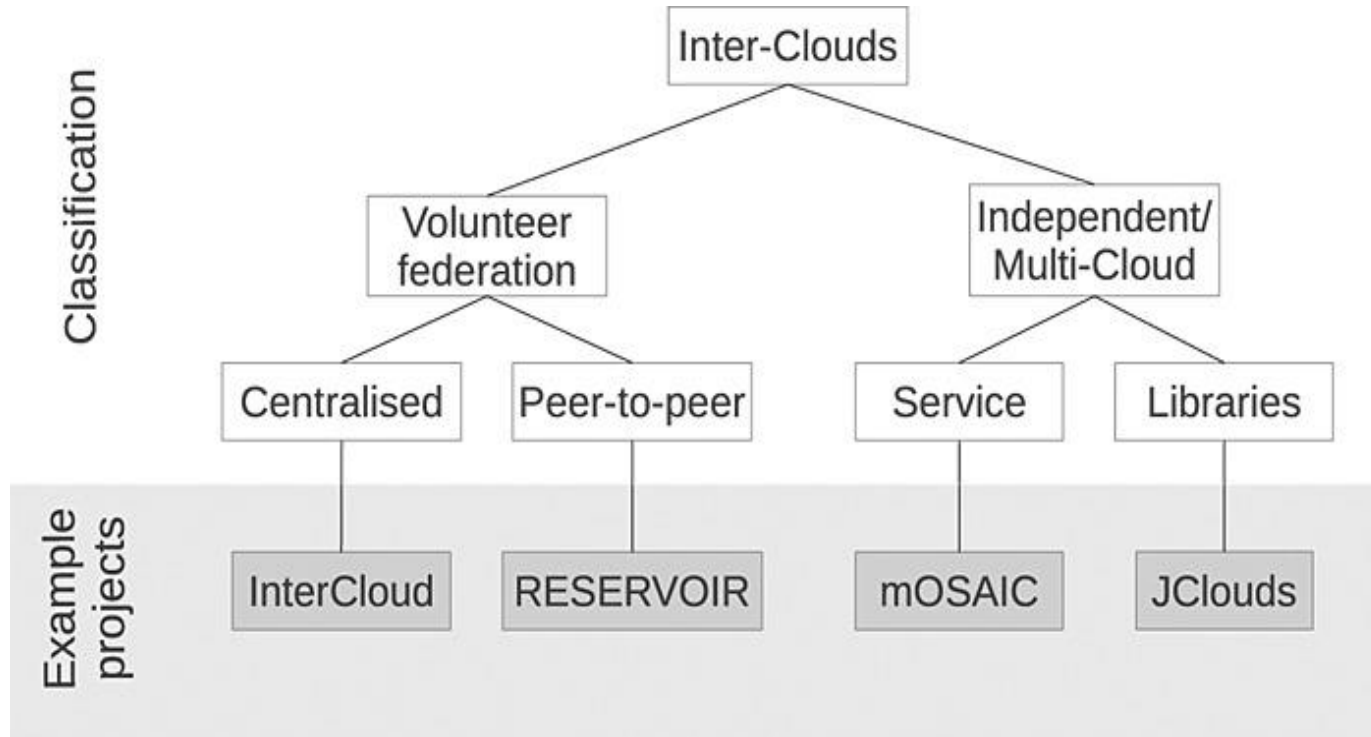
<https://github.com/nlintz/TensorFlow-tutorials/>

# Intercloud

- The concept of connected cloud networks including public, private, and hybrid clouds.
- It improves the interoperability and portability among the cloud networks.
- It is used to connect different cloud computing platforms and allows the data and application to be ported between data centers as cloud services.
- The main focus is an direct interoperability between public cloud service providers.



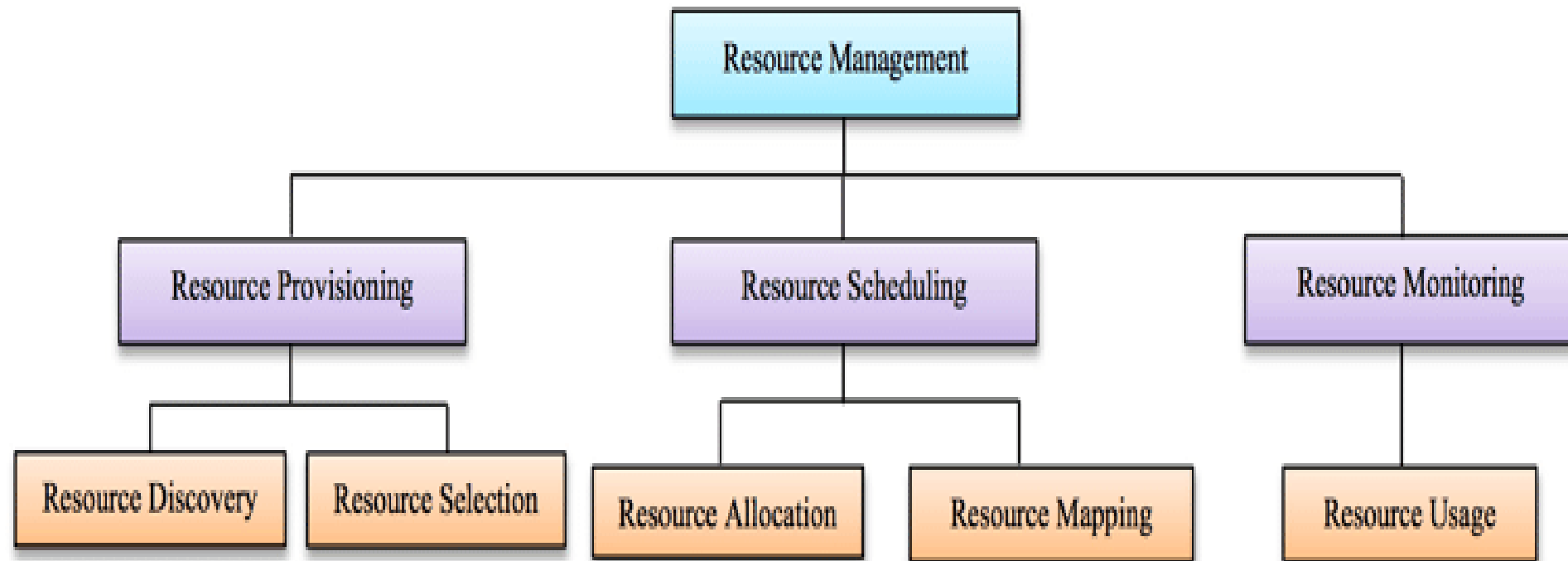
# Intercloud Architecture



- Centralised intercloud federation
- Peer-to-peer intercloud federation
- Multi-cloud services

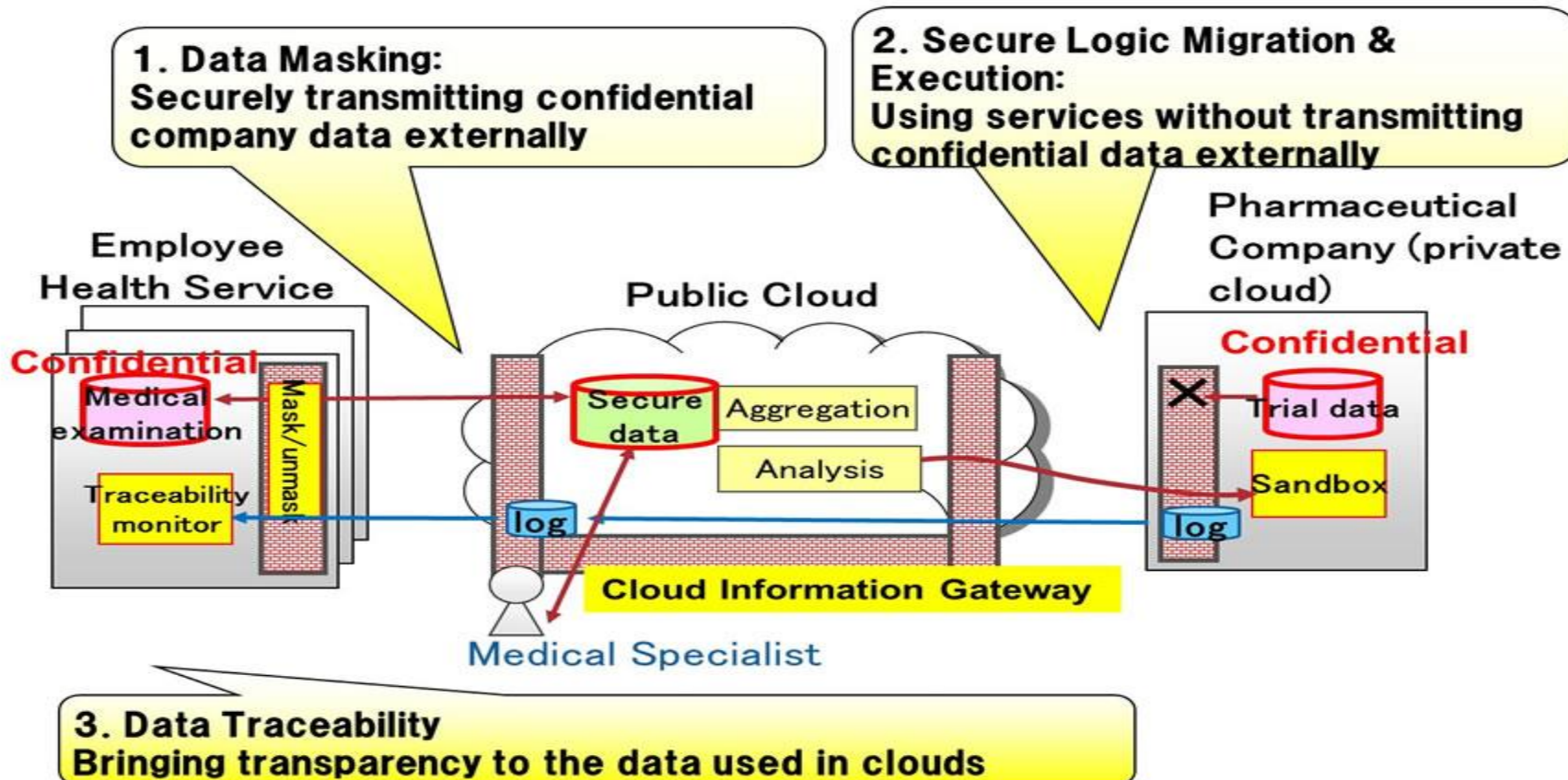
# Intercloud Resource Management

- These services are built to perform resource-discovery, match, schedule, select, negotiate, and monitor operations.



# Intercloud Security

- Intercloud security involves protecting data and applications that are distributed across multiple cloud environments.
- It ensures the confidentiality, integrity, and availability of information as it moves between different cloud platforms



# Intercloud Billing

- Intercloud billing can be complex due to the diverse nature of cloud services and providers involved.

Parameters involved in intercloud billing :

1. Resource usage measurement
2. Metering
3. Pricing models
4. Cross cloud transactions
5. Aggregation and conversion
6. Billing statements
7. Challenges and considerations
8. Third-party tools

# Mobile Cloud Computing (MCC)

## Motivation:

- Growth in use of smart phone, apps
  - Increased capability of mobile devices
  - Access of Internet using mobile devices rather than PCs!
- 
- Resource challenges (battery life, storage, bandwidth) in mobile devices?
  - Cloud computing offers advantages to users by allowing them to use infrastructure, platforms and software by cloud providers at low cost and elastically in an on-demand fashion.

# Mobile backend-as-a-service

What	<ul style="list-style-type: none"><li>• Provides mobile application developer a way to connect their application to backend cloud storage and processing</li></ul>
Why	<ul style="list-style-type: none"><li>• Abstract away complexities of launching and managing own infrastructure</li><li>• Focus more on frontend development instead of backend functions</li></ul>
When	<ul style="list-style-type: none"><li>• Multiple Apps, multiple backends, multiple developers</li><li>• Multiple mobile platforms, multiple mobile integration, multiple third-party systems and tools</li></ul>
How	<ul style="list-style-type: none"><li>• Meaningful resources for app development accelerations- 3<sup>rd</sup> party API, Device SDK's, enterprise connectors, social integration, cloud storage</li></ul>

# Augmenting Mobile with Cloud computing

- Amazon Silk browser
  - Split browser
- Apple Siri
  - Speech recognition in cloud
- Apple iCloud
  - Unlimited storage and sync capabilities
- Image recognition apps on smart phones useful in developing augmented reality apps on mobile devices
  - Augmented reality apps using Google Glass

# What is Mobile Cloud Computing?

- Mobile Cloud Computing (MCC) is the combination of cloud computing, mobile computing, and wireless networks to bring rich computational resources to mobile users.
- MCC provides mobile users with data storage and processing services in cloud
  - ✓ Obviating the need to have a powerful device configuration(e.g., CPU speed, memory capacity etc.)
  - ✓ All resource-intensive computing can be performed in the cloud
- Moving computing power and data storage away from the mobile devices
  - ✓ Powerful and centralised computing platforms located in cloud
  - ✓ Accessed over the wireless connection based on a thin native client



# Why MCC?

- **Speed and Flexibility**

Mobile cloud applications can be built or revised quickly using cloud services. They can be delivered to many different devices with different operating systems.

- **Shared Resources**

Mobile apps that run on the cloud are not constrained by a device's storage and processing resources. Data intensive processes can run in the cloud. User engagement can continue seamlessly from one device to another.

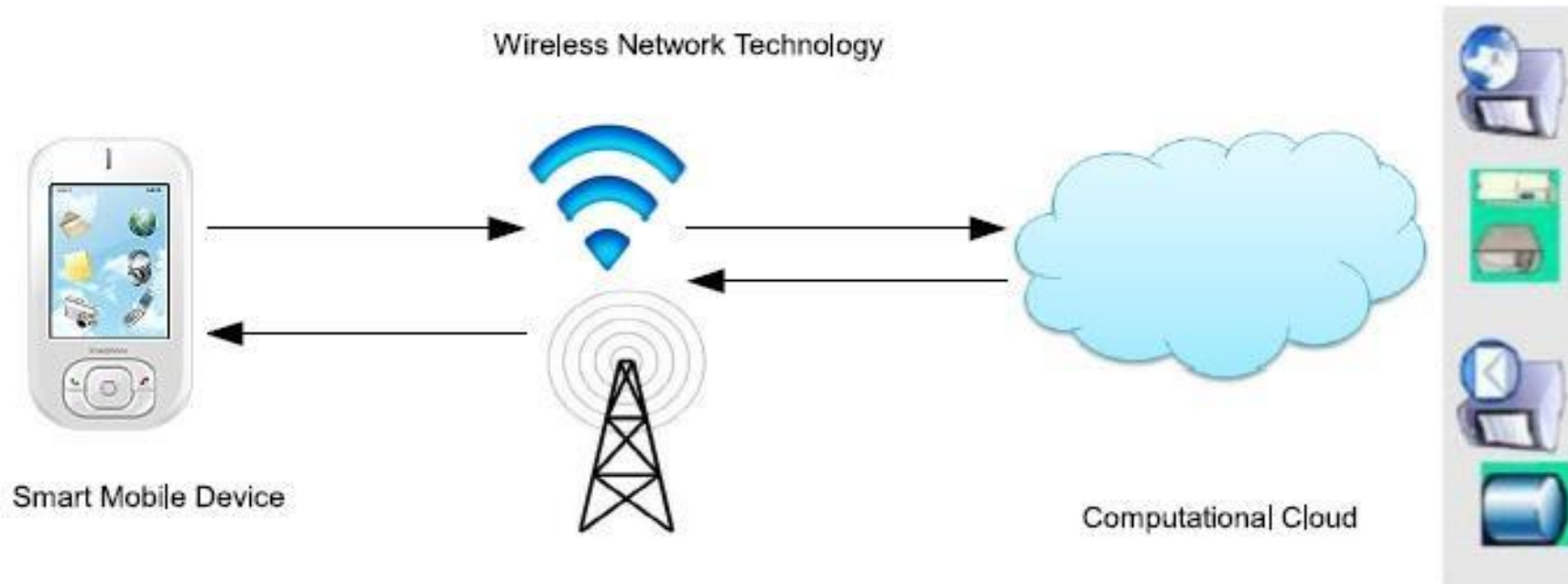
- **Integrated Data**

Mobile cloud computing enables users to quickly and securely collect and integrate data from various sources, regardless of where it resides.

# Key features of MCC

- MCC delivers applications to mobile devices quickly and securely, with capabilities beyond those of local resources.
- Facilitates the quick development, delivery, and management of mobile apps.
- Uses fewer device resources because applications are cloud supported.
- Supports a variety of development approaches and devices.
- Mobile devices connect to services delivered through an API architecture.
- Improves reliability with information backed up and stored in the cloud.

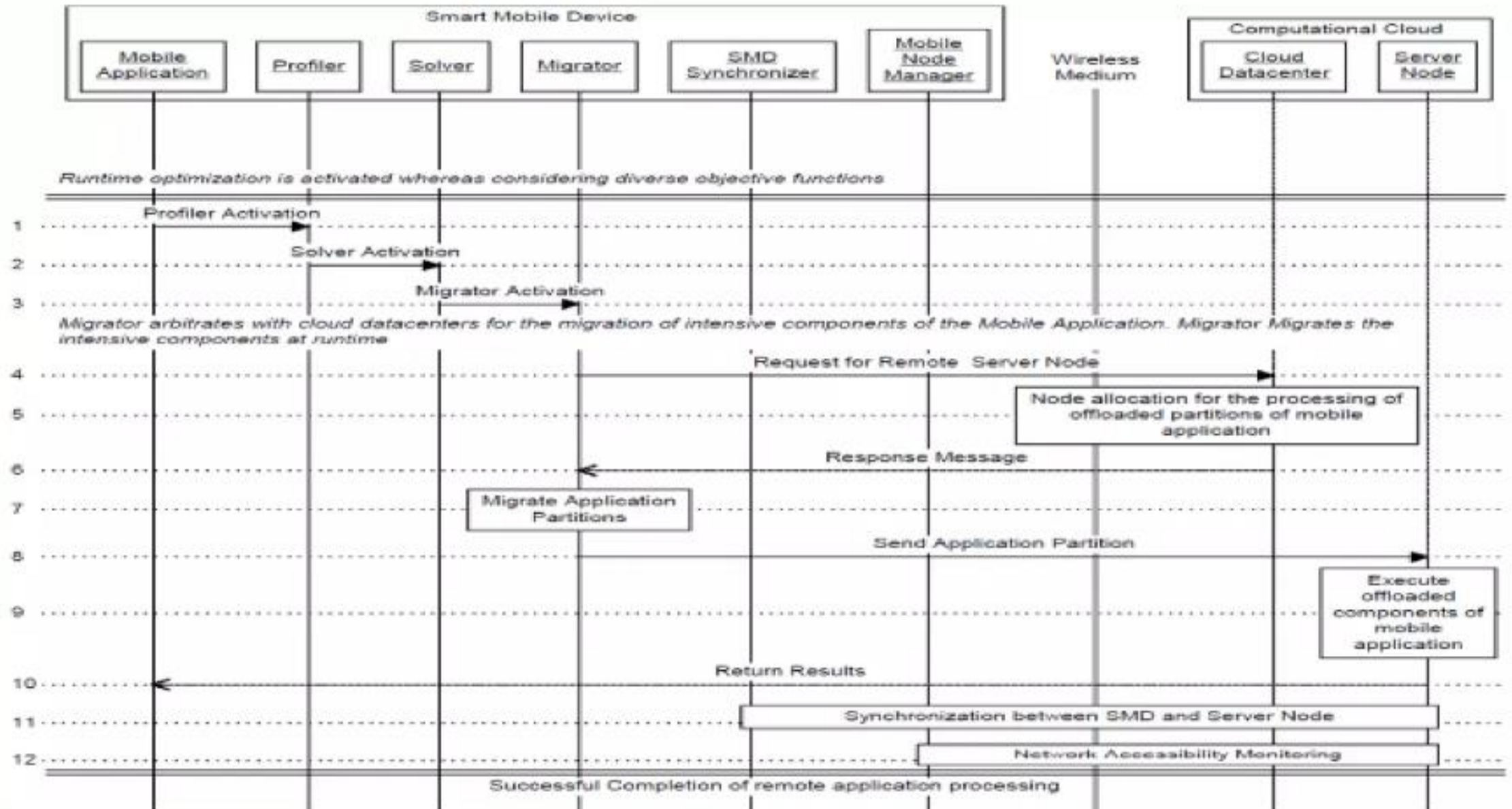
# Mobile Cloud Computing



Mobile cloud computing is a framework to augment a resource constrained mobile device to execute parts of the program on cloud based servers

Pros	Cons
Saves battery power	Must send the program states (data) to the cloud servers, hence consumes battery
Makes execution faster	Network latency can lead to execution delay

# Typical MCC Workflow



# MCC key components

## ❑ Profiler

- Monitors application execution to collect data about the time to execute, power consumption, network traffic

## ❑ Solver

- Solver has the task of selecting which parts of an app runs on mobile and cloud

## ❑ Synchronizer

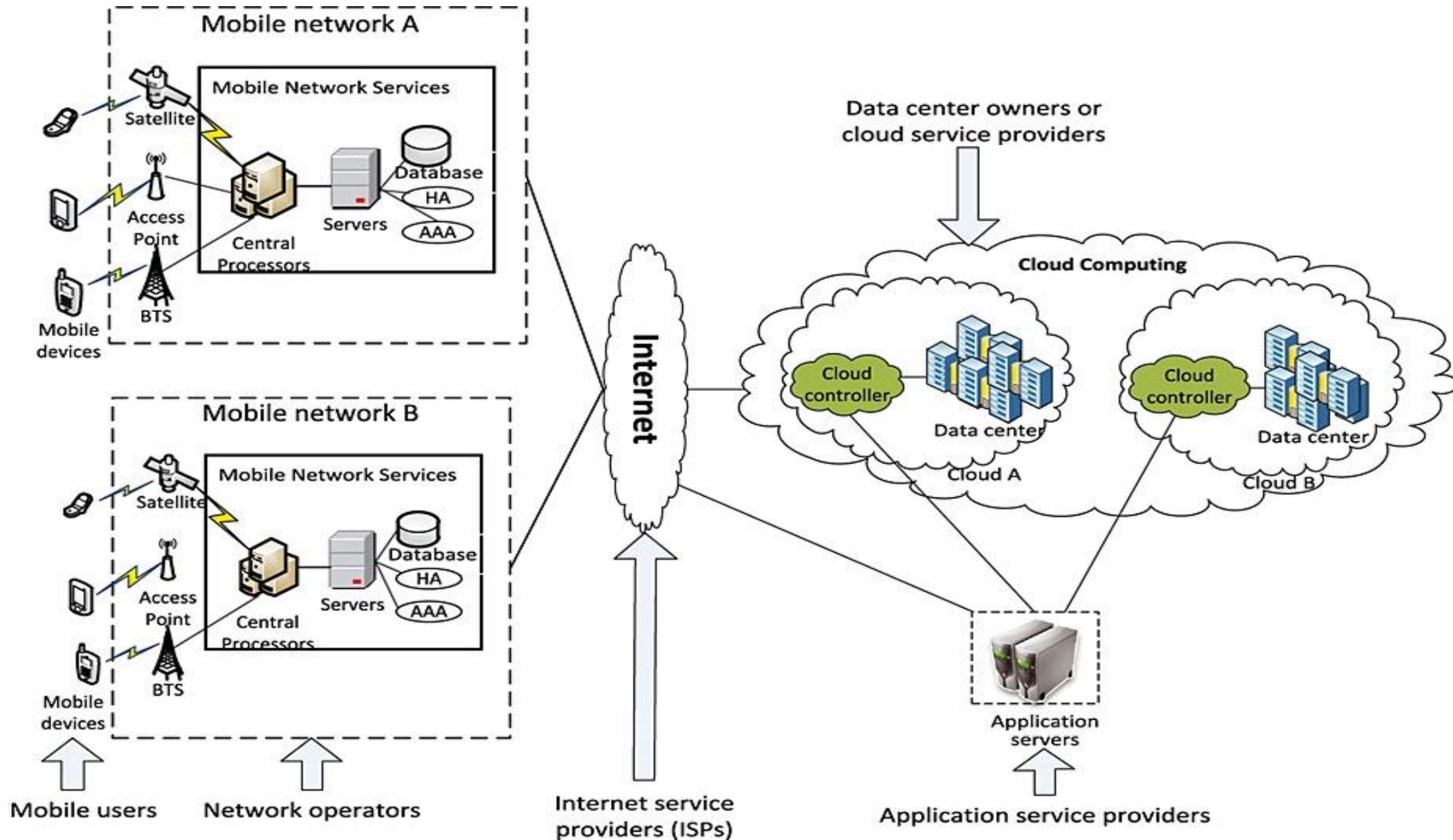
- Tasks of synchronizer modules is to collect results of split execution and combine, and make the execution details transparent to the user

- Simple APIs offering access to mobile services and requiring no specific knowledge of underlying network technologies

- Web Interface

- Internet access to remotely stored applications in the cloud

# MCC- Typical Architecture



# Types of Cloud Resources in MCC

- Distant mobile cloud
- Distant immobile cloud
- Proximate mobile entity
- Hybrid

# Advantages of MCC

- Extending battery lifetime
- Improving data storage capacity and processing power
- Improve reliability and availability
- Dynamic provisioning
- Scalability
- Multi-tenancy
- Ease of integration



# MCC Challenges

- **Security Issues**

Protecting user privacy and data/application secrecy from adversaries is the key to establish and maintain consumer's trust in the mobile platform, especially in MCC.

MCC security issues have two main categories

- ✓ Security for mobile users
- ✓ Securing data on clouds

# MCC Challenges

- **Security and Privacy for Mobile Users**

- Mobile devices are exposed to numerous security threats like malicious codes and their vulnerability
- GPS can cause privacy issues for subscribers
- Security for mobile applications:
  - Installing and running security software are the simplest ways to detect security threats
  - Mobile devices are resource constrained, protecting them from the threats is more difficult than that for resourceful devices.
- Location based services (LBS) faces a privacy issue on mobile users' provide privacy information such as their current location
- Problem becomes even worse, if an adversary knows user's important information

# MCC Challenges

- **Security for Mobile Users**

- Approaches to move the threat detection capabilities to the cloud
- Host agent runs on mobile devices to inspect the file activity on a system. If an identified file is not available in a cache of previously analyzed files, the file will be sent to the cloud network service for verification.
- Attack detection for a smart phone is performed on a remote server in the cloud.
- The smart phone records only a minimal execution trace, and transmits it to the security server in the cloud.

# MCC Challenges

- Network Access Management
- Quality of Service
- Pricing
- Context-aware Mobile Cloud services
- Standard Interface
- Service Convergence

# MCC Applications

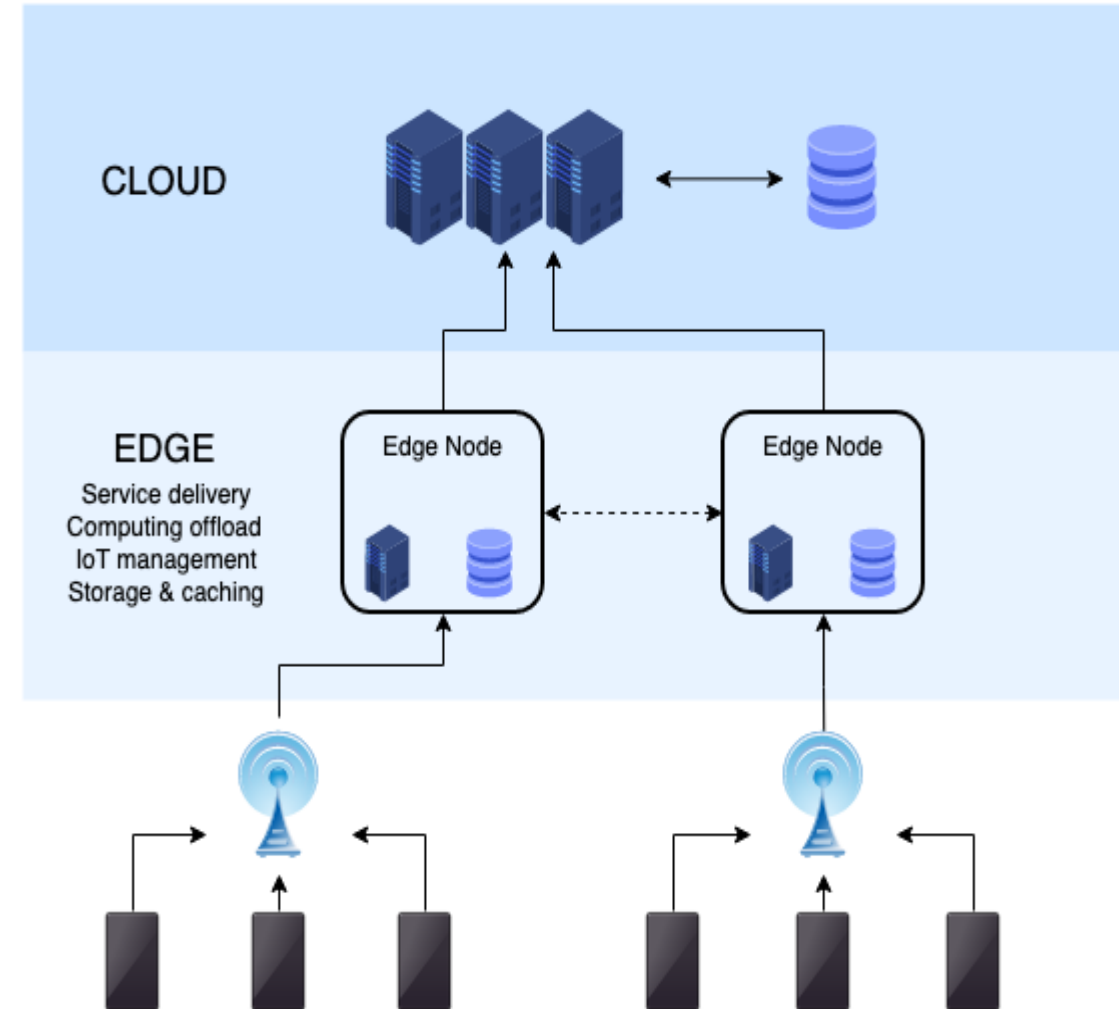
- M-Commerce (B2B, B2C, C2C, D2C, D2D)
- Mobile learning
- Mobile healthcare
- Mobile gaming
- And a lot more.....

# Edge Computing

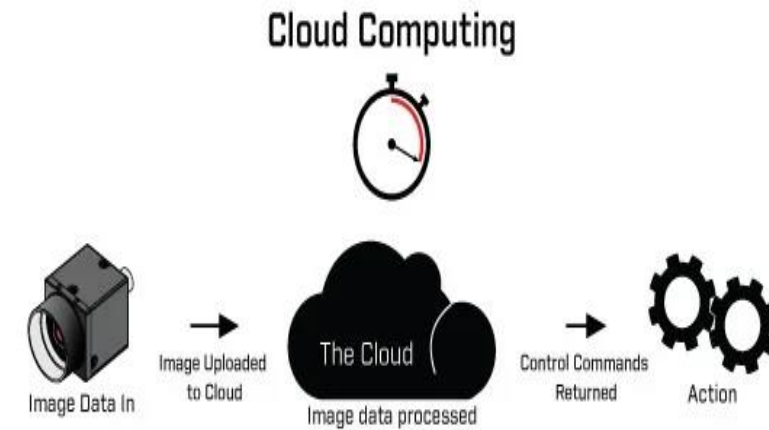
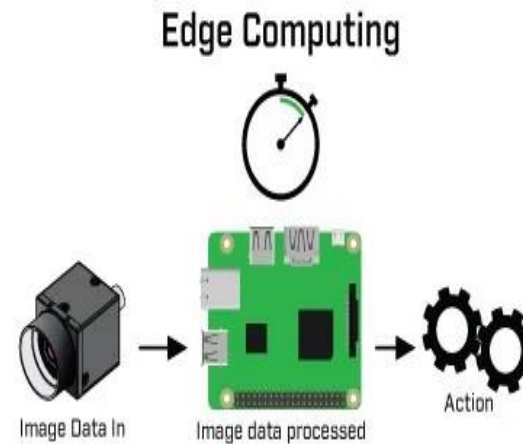
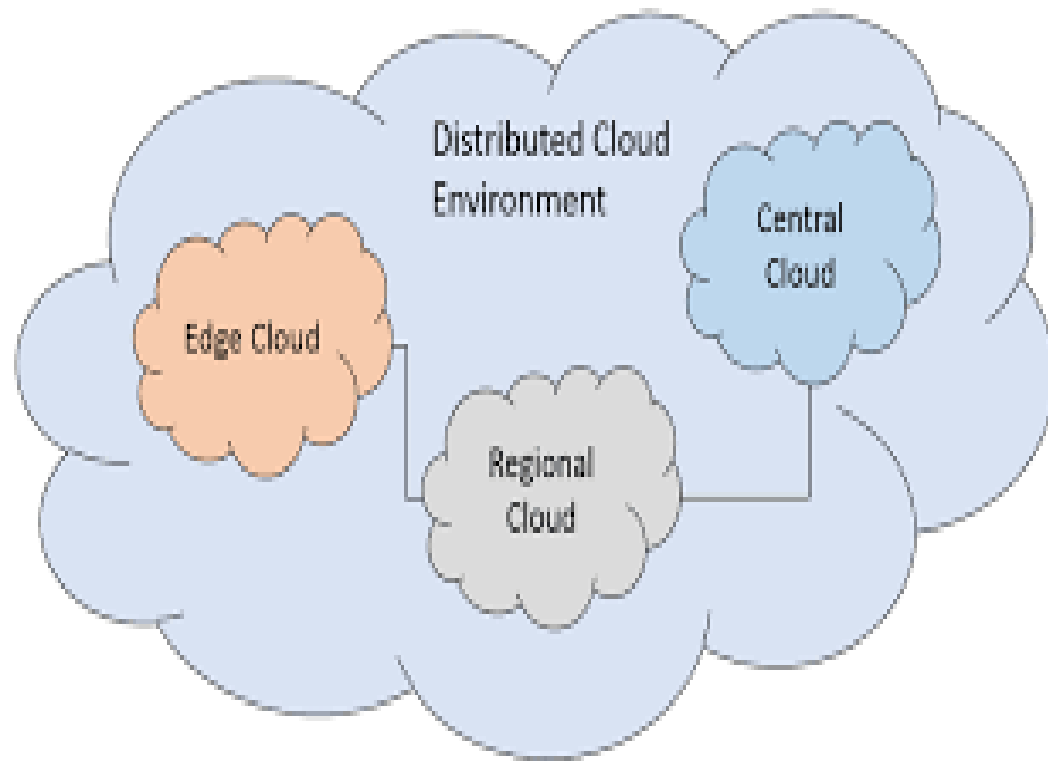
- Edge computing allows the cloud to be genuinely distributed.
- Don't need to rely on the cloud for all the processing and data aggregation collection processing and querying.
- Mimics the public cloud platform capabilities.
- Reduces the latency by avoiding the round-trip and brings in the data sovereignty by keeping data where it actually belongs.
- Delivers local storage, compute and network services.

# Edge computing: makes distributed cloud

- The previous generation of cloud was a client-server architecture where a very little processing was done in the client side but all the heavy lifting was done in the cloud.
- With all the innovations done in the hardware chips, it brings more sense to bring the compute down to the devices.



Edge computing: mimics the public cloud platform capabilities, reduces the latency



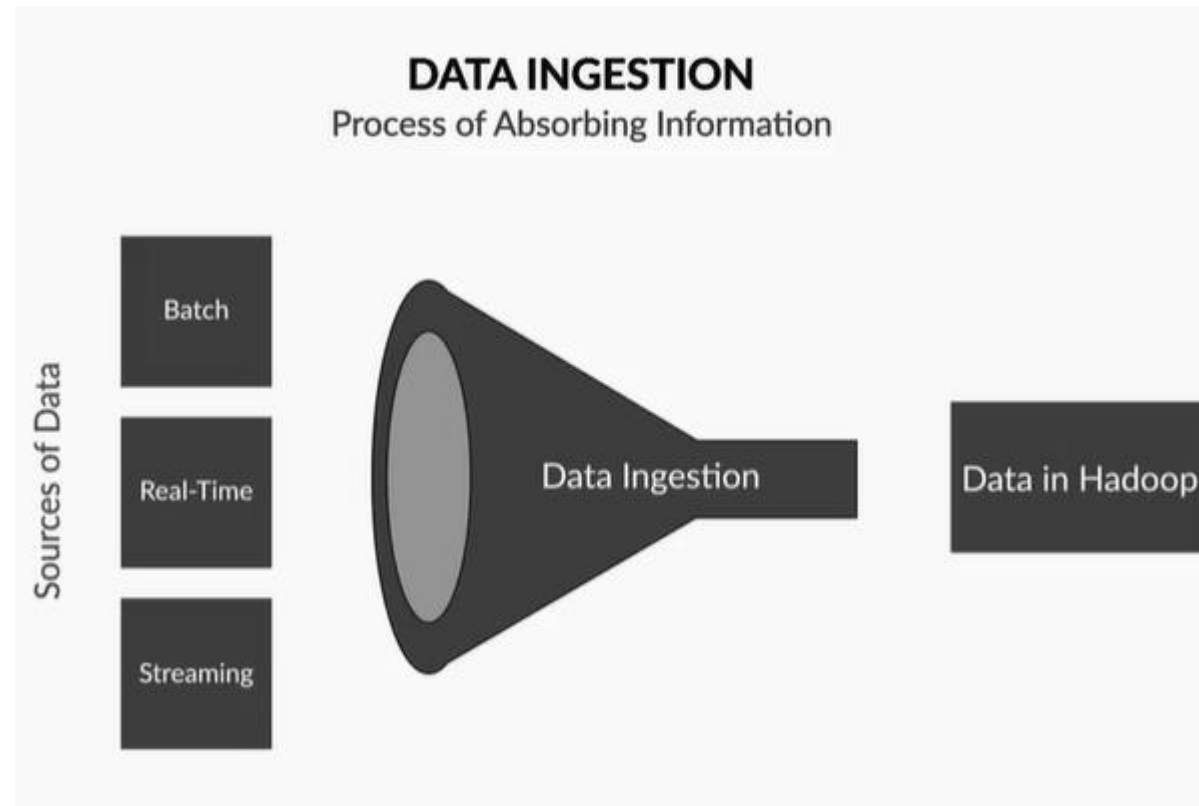


# Edge computing building blocks

- Data ingestion
- M2M brokers
- Object storage
- Function as a service
- NoSQL/Time-series database
- Stream processing
- ML models

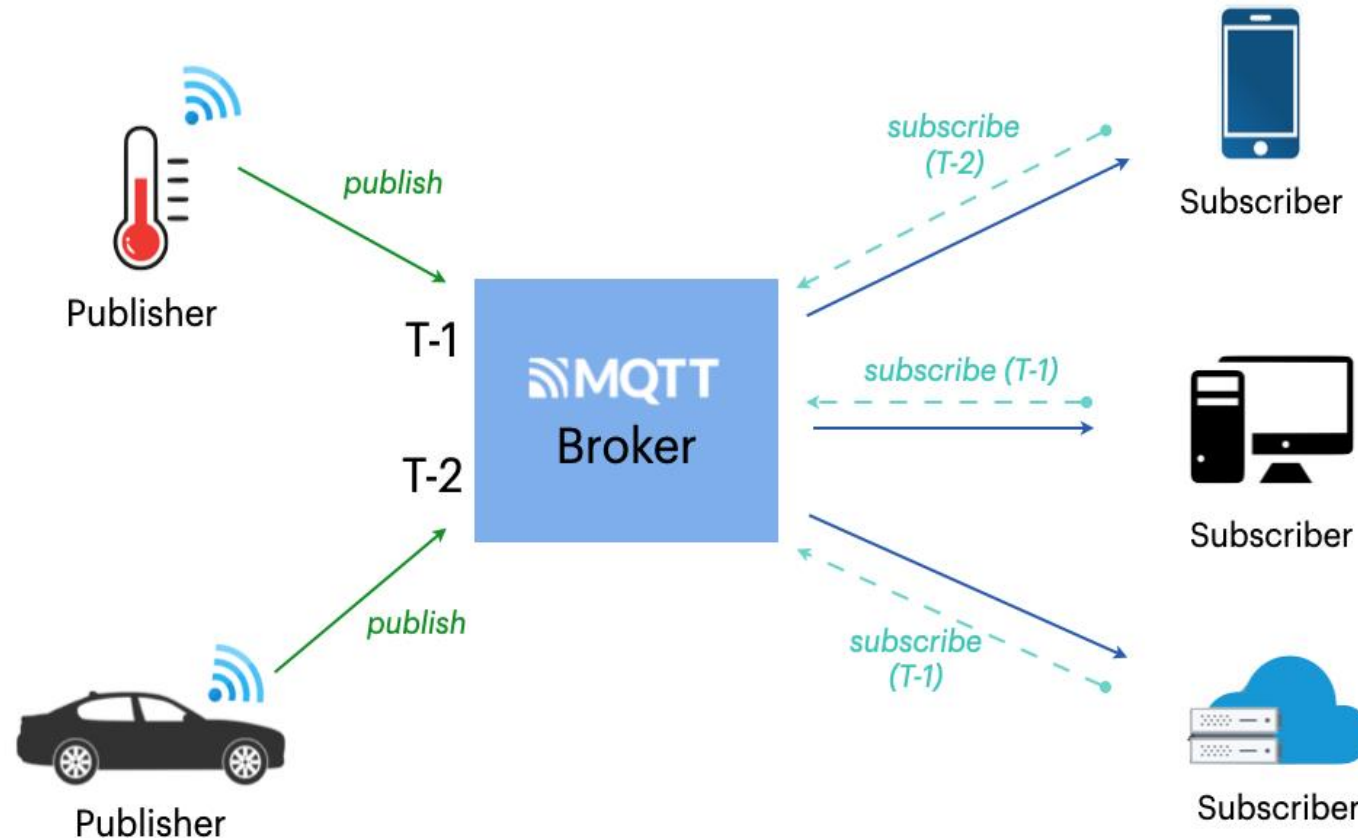
# Data Ingestion

- It is the process of obtaining and importing data for the immediate use or storage in a database. Data can be streamed in real-time or ingested in batches. In real-time data ingestion, each data item is imported as the source emits it.



# Machine to Machine (M2M) Brokers

- Edge will also run message brokers that will orchestrate machine to machine communications. Ex: device one talks to device two via the M2M brokers.



# Storage

- Object Storage-> anything that is unstructured will go into object storage, particularly to store the feed from video cameras and mics.
- NoSQL/Time-series Database-> More structured data go into NoSQL and time-series database.



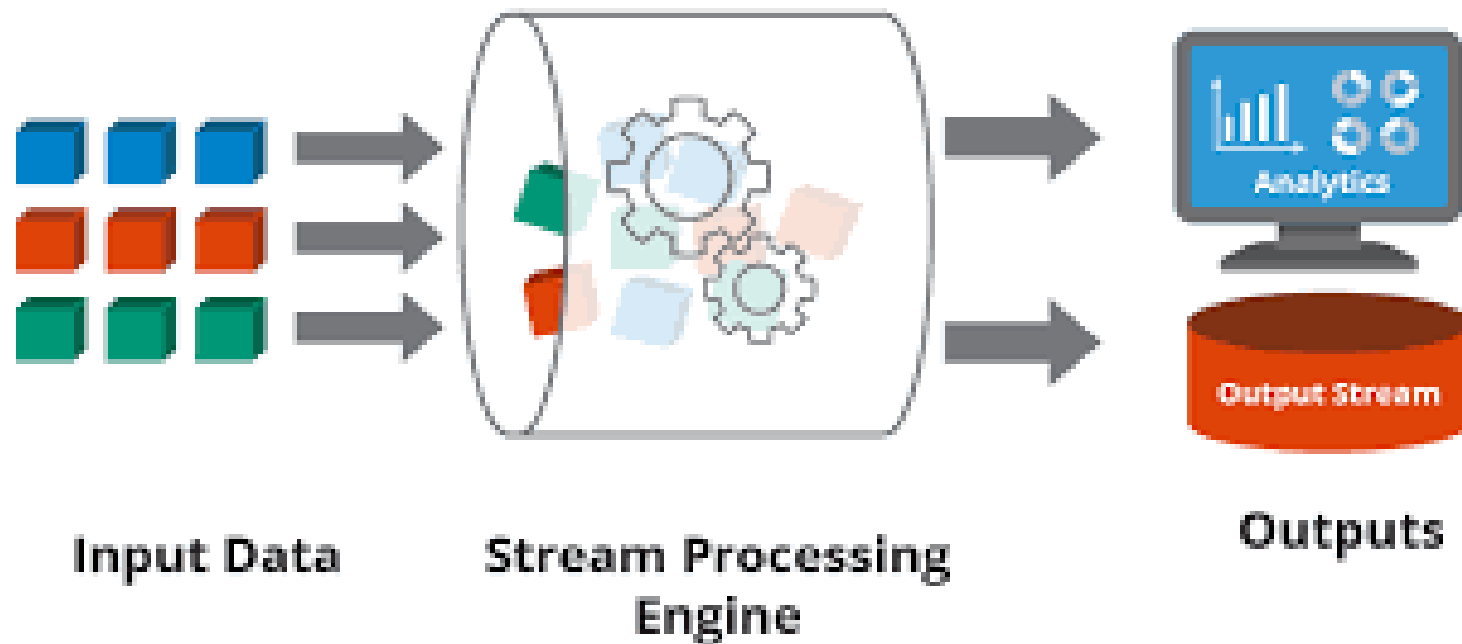
UNSTRUCTURED DATA



STRUCTURED DATA

# Stream Processing

- It is a complex event processing engine that is enabling you to perform real-time queries and process the data as it comes. Ex: convert Farenheit to Celsius or convert one time stamp to another.



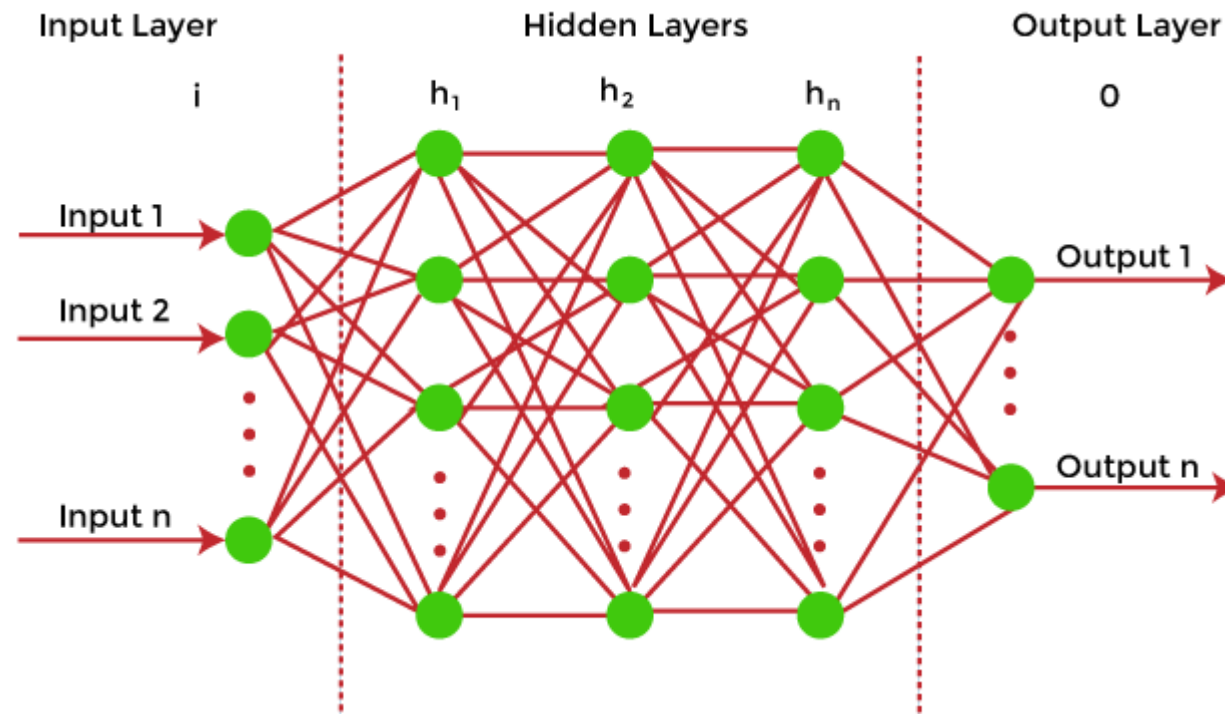
# Function as a Service

- Function as a service is actually responsible for running light-weight computation. All the sophisticated coding is done in the function as a service paradigm.

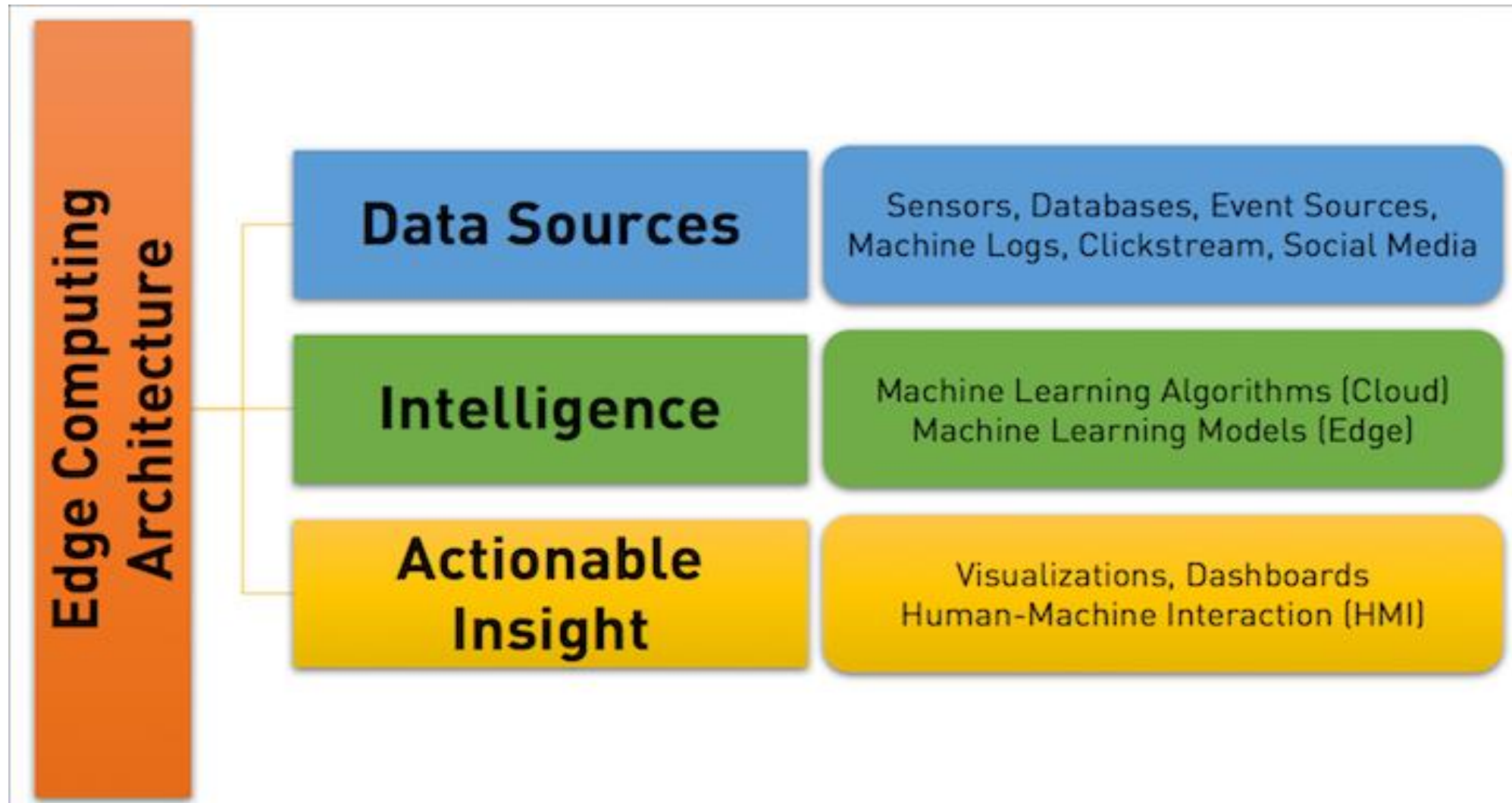


# ML Models

- There is an ML run time, for example; most of the computing platforms are capable of running TensorFlow light, pitorch models, Cafe models, so you can actually process the data that comes in more intelligently and take preventive measures and perform predictive analytics.



# Edge Computing Architecture





# Data Source Tier

- These are nothing but original end point from where the data is acquired or the origin of the data.
- In industrial IoT environment, this could be a set of devices that are generating the data.

# Intelligence Tier

- Responsible for running the machine running models.
- This intelligent tier cuts across the cloud and the edge, so there is a very well-defined boundary between edge and cloud where the training takes place on the cloud and the inferencing is run on the edge. But, collectively this overlap between the cloud and the edge is the intelligence layer.

# Actionable Inside Tier

- Responsible for sending an alert to the relevant stake holders or populating the dashboards and showing some visualizations or even the edge taking an action to immediately shutdown a faulty machine or controlling an actuator and again the actionable insight takes place on the edge so this is not a physical boundary.

# Fog Computing

Challenges of cloud computing:

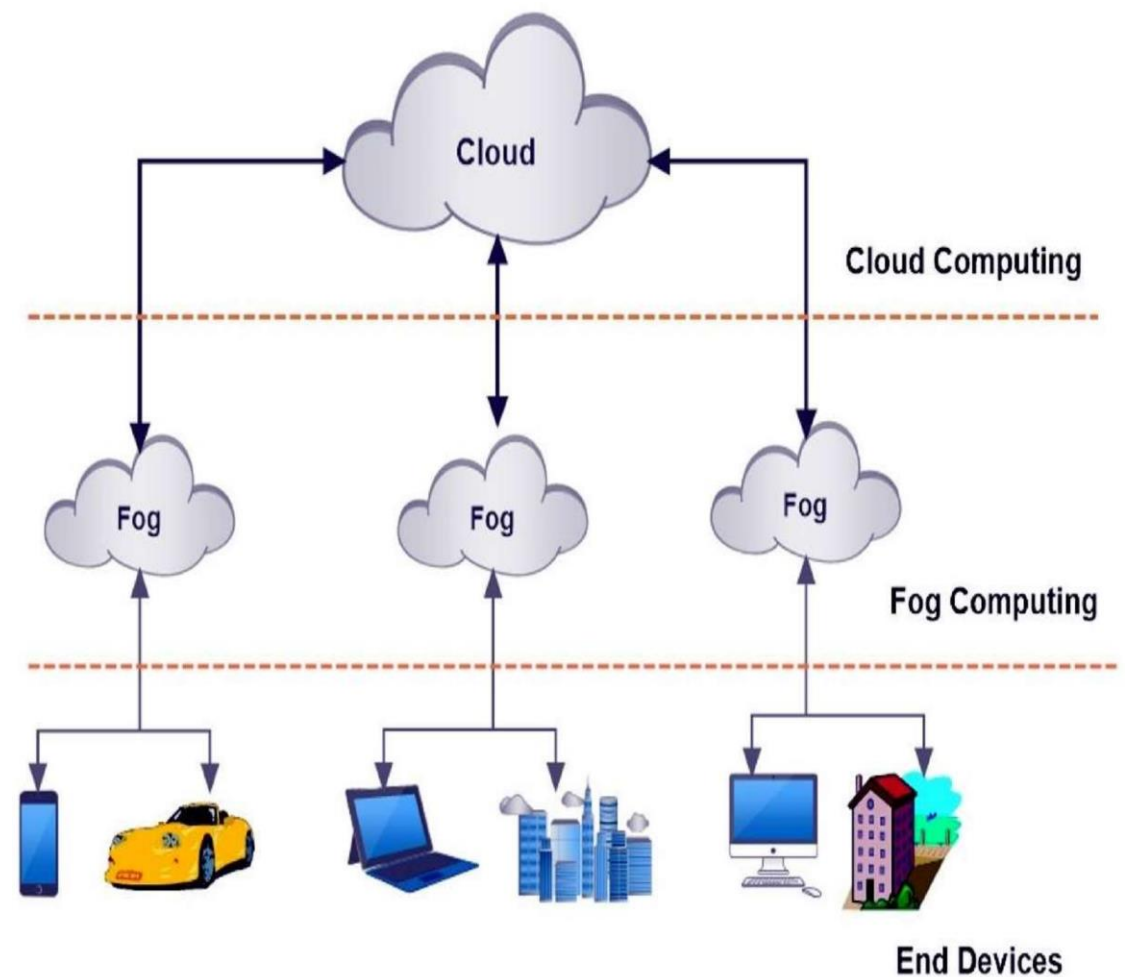
- Processing of huge data in a datacenter.
- Datacenter may be privately hosted by the organization (private cloud set up) or publicly available by paying rent (public cloud).
- All the necessary information has to be uploaded to the cloud for processing and extracting knowledge from it.

Issues with cloud-only computing:

- Communication takes a long time due to human-smart phone interaction.
- Datacenters are centralized, so all the data from different regions can cause congestion in the core network.
- Such a task requires very low response time, to prevent further crashes or traffic jam.

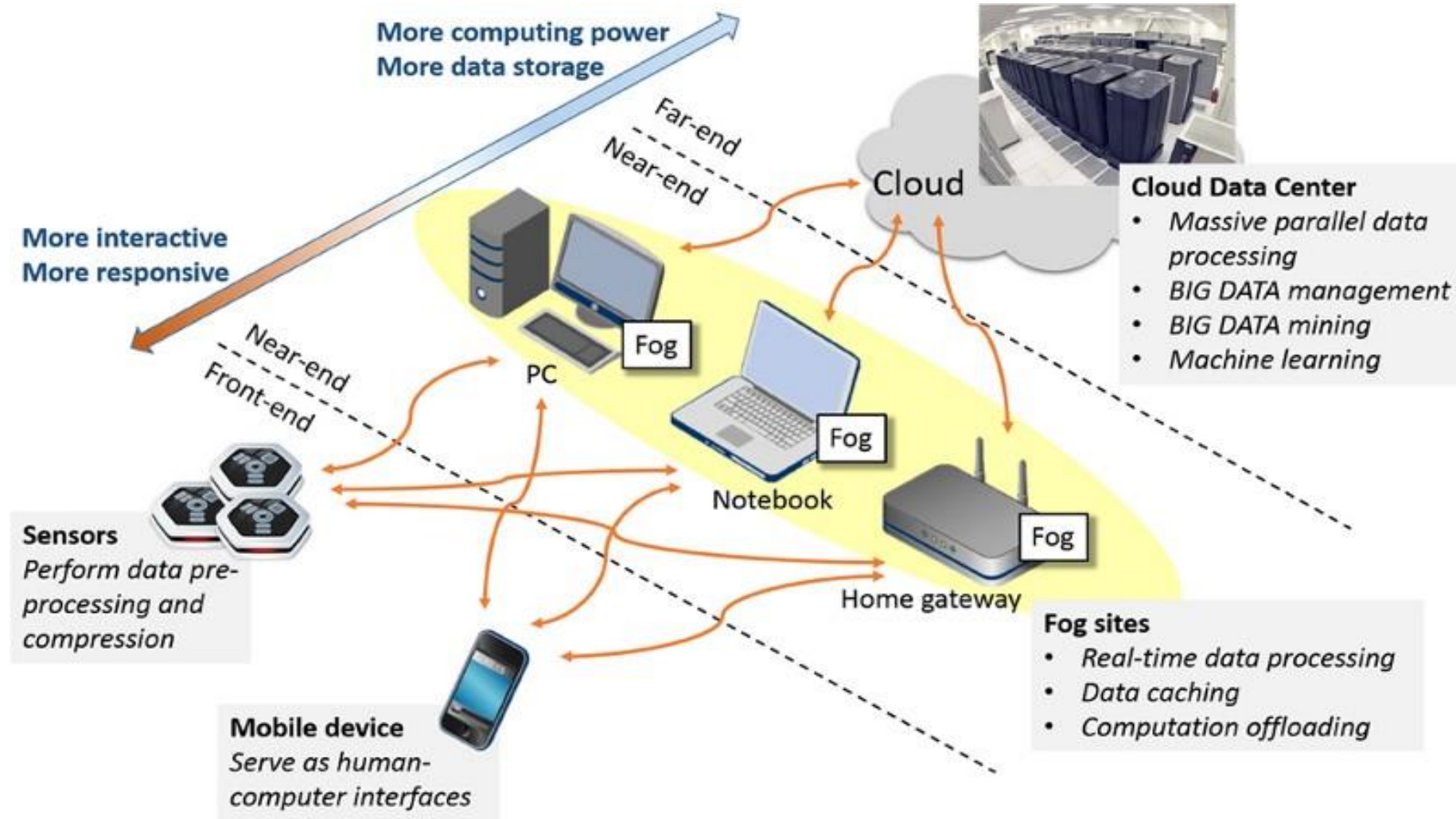
- Fog computing is known as fogging computing, it is a model in which data processing and applications are concentrated in devices at the network edge rather than existing almost entirely in the cloud.
- The term “Fog computing” was introduced by the cisco systems as new model to ease wireless data transfer to distributed services in the Internet of things (IoT) paradigm.
- Cisco’s vision of fog computing is to enable applications on billion of connected devices to run directly at the network edge.
  - Users can develop, run, manage network applications on cisco framework of networked devices, including hardened routers and switches.
  - Cisco brings the open source Linux and network operating system together in a single networked device.

- Bringing intelligence down from the cloud close to the ground/end-user.
- Cellular base stations, network routers, Wi-fi gateways will be capable of running applications.
- End devices like sensors are able to perform basic data processing.
- Processing close to devices lowers response time, enabling real-time applications.
- It enables some of transactions and resources at the edge of the cloud, rather than establishing channels for cloud storage and utilization.
- It reduces the need for bandwidth by not sending every bit of information over cloud channels, and instead aggregating it at certain access points.
- These kind of distributed strategy, may help in lowering cost and improve efficiencies.



# Motivation

- Fog computing is a paradigm that extends cloud and its services to the edge of the network.
- Fog provides data, compute, storage, and application services to the end users.
- Recent advances: smart grid, smart traffic lights, connected vehicles, software-defined network.



# Fog Computing Enablers

- **Virtualization:** Virtual machines can be used in edge devices.
- **Containers:** Reduces the overhead of resource management by using light-weight virtualizations. Example: Docker containers.
- **Service Oriented Architecture:** Service-oriented architecture (SOA) is a style of software design where services are provided to the other components like application components, through a communication protocol over a network.
- **Software-defined Networking:** Software-defined Networking (SDN) is an approach to use open protocols such as OpenFlow, to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware.



# Fog Computing- not a replacement of Cloud Computing

- Fog/edge devices are there to help the cloud data center to better response time for real-time applications. Handshaking among fog and cloud computing are needed.
- Broadly, benefits of fog computing are:
  - Low latency and location awareness
  - Widespread geographical distribution
  - Mobility
  - Very large number of nodes
  - Predominant role of wireless access
  - Strong presence of streaming and real-time applications
  - Heterogeneity

# Fog advantages

- Fog can be distinguished from cloud by its proximity to end users.
- Dense geographical distribution and its support for mobility
- It provides low latency, location-awareness, and improves quality-of-service (QoS) and real-time applications.

# Fog Computing and Cloud Computing

Requirement	Cloud Computing	Fog Computing
Latency	High	Low
Delay jitter	High	Very low
Location of server nodes	Within Internet	At the edge of local network
Distance between the client and server	Multiple hops	One hop
Security	Undefined	Can be defined
Attack on data encounter	High probability	Very less probability
Location awareness	No	Yes
No. of server nodes	Few	Very large
Geographical distribution	Centralized	Distributed
Support of mobility	Limited	Supported
Real-time interactions	Supported	Supported
Type of last mile connectivity	Leased line	Wireless

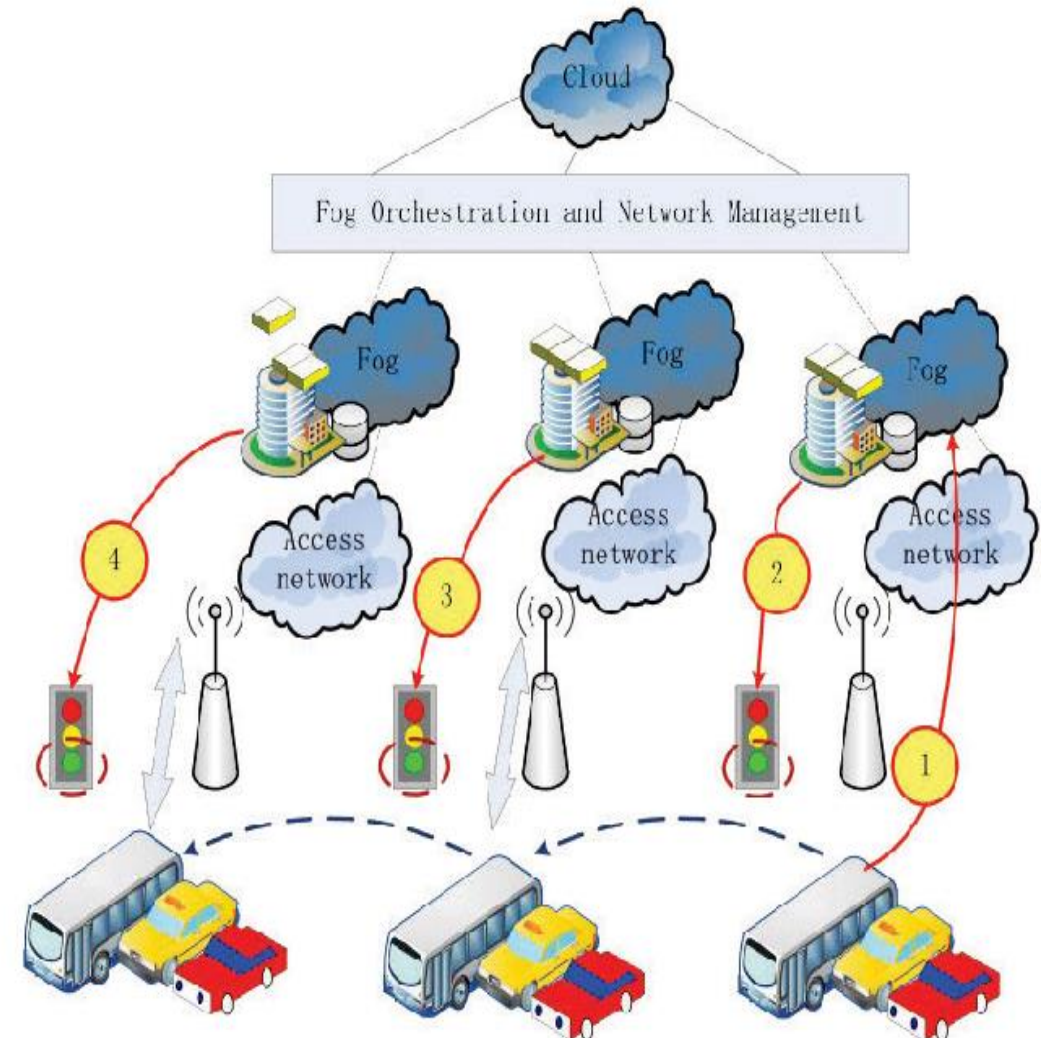
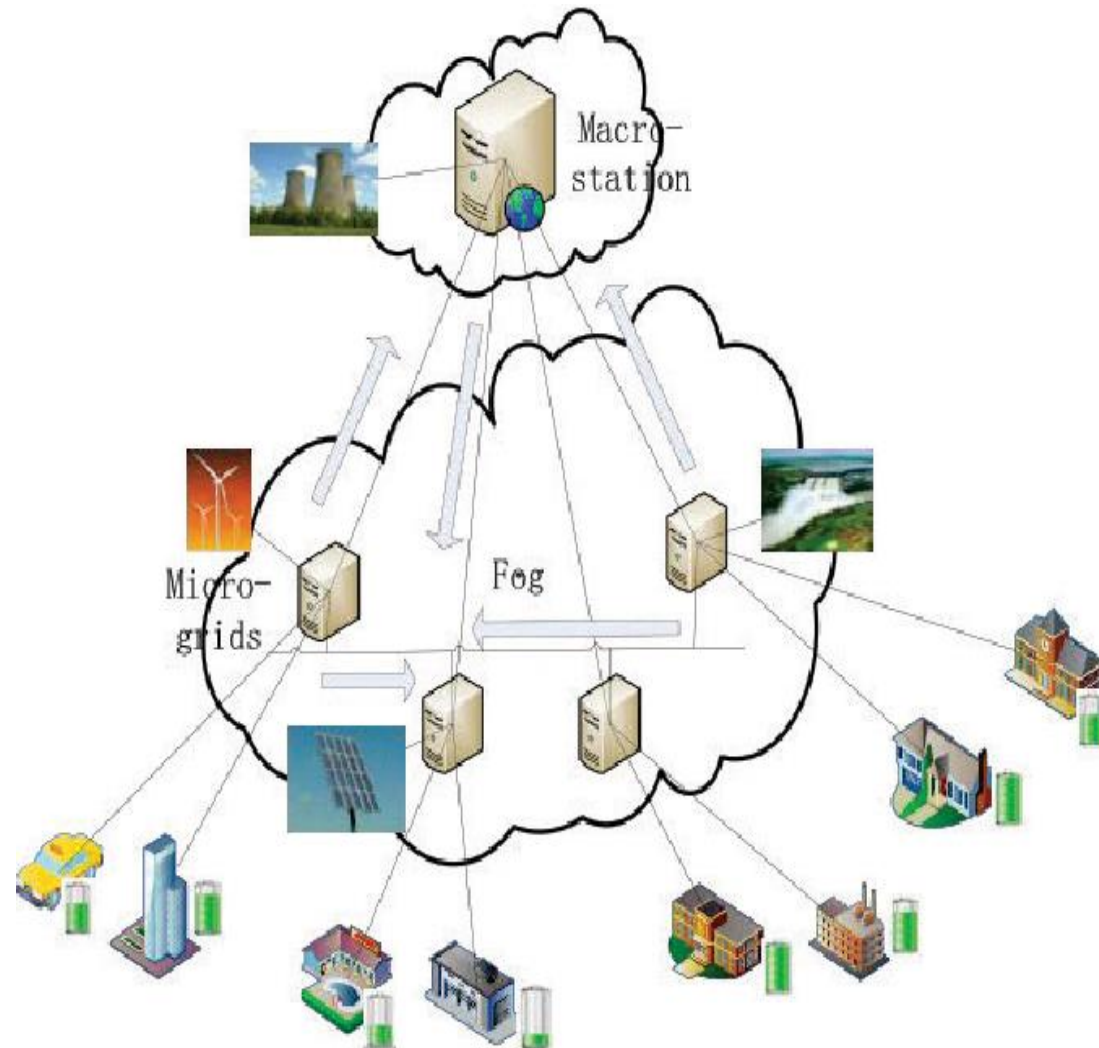
# Fog Computing Use cases

- **Emergency Evacuation Systems:** Real-time information about currently affected areas of building and exit route planning.
- **Natural Disaster Management:** Real-time notification about landslides, flash floods to potentially affected areas.
- Large sensor deployments generate a lot of data, which can be pre-processed, summarized, and then sent to the cloud to reduce congestion in the network.
- **Internet of Things (IoT)** based big data applications: Connected vehicles, smart cities, wireless sensor and actuator networks (WSAN) etc.

# Applicability

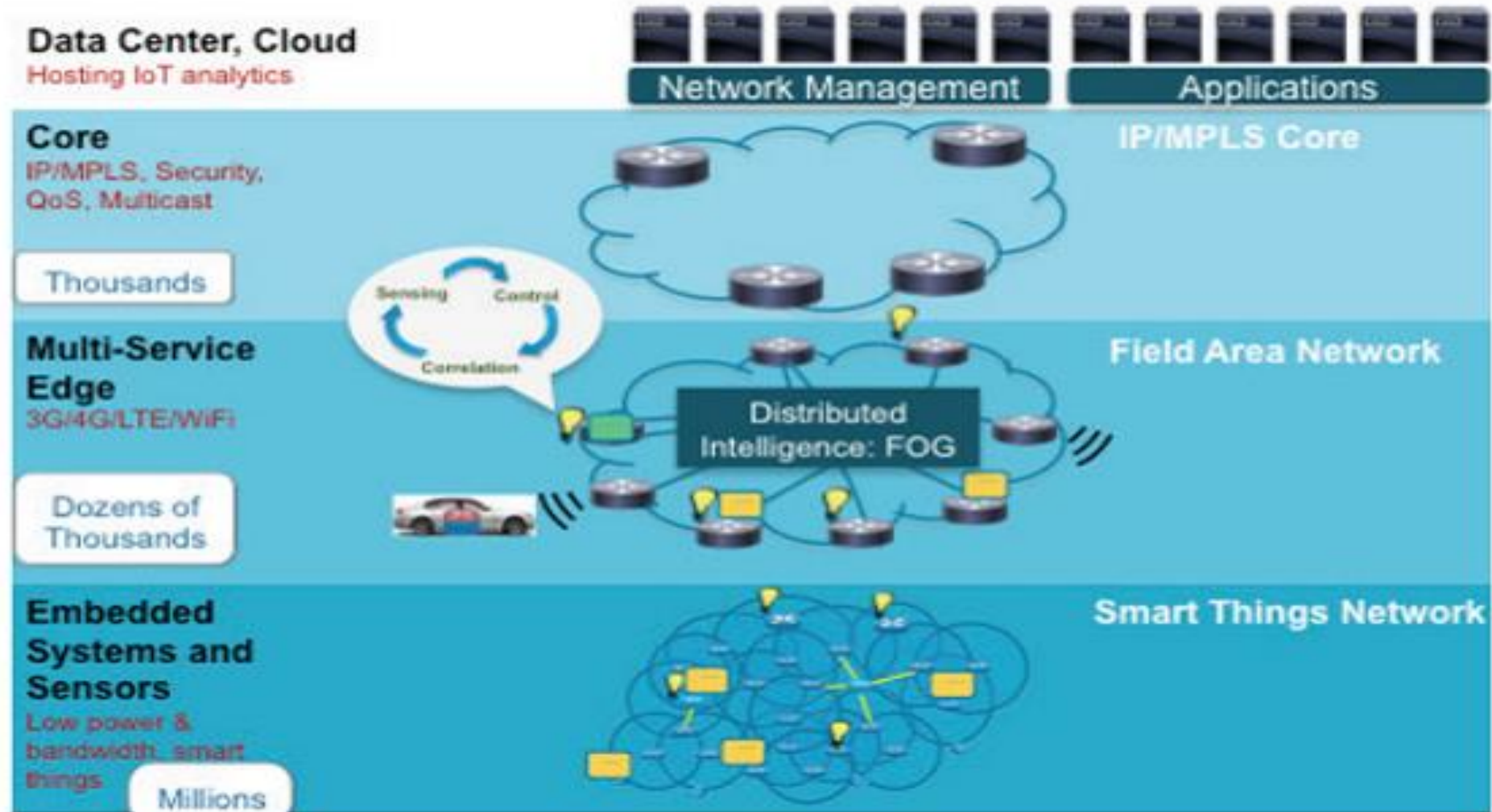
- Smart traffic lights
- Connected vehicles
- Smart grids
- Wireless sensors
- Internet of Things
- Software-defined network

# Fog computing in smart traffic lights and connected vehicles

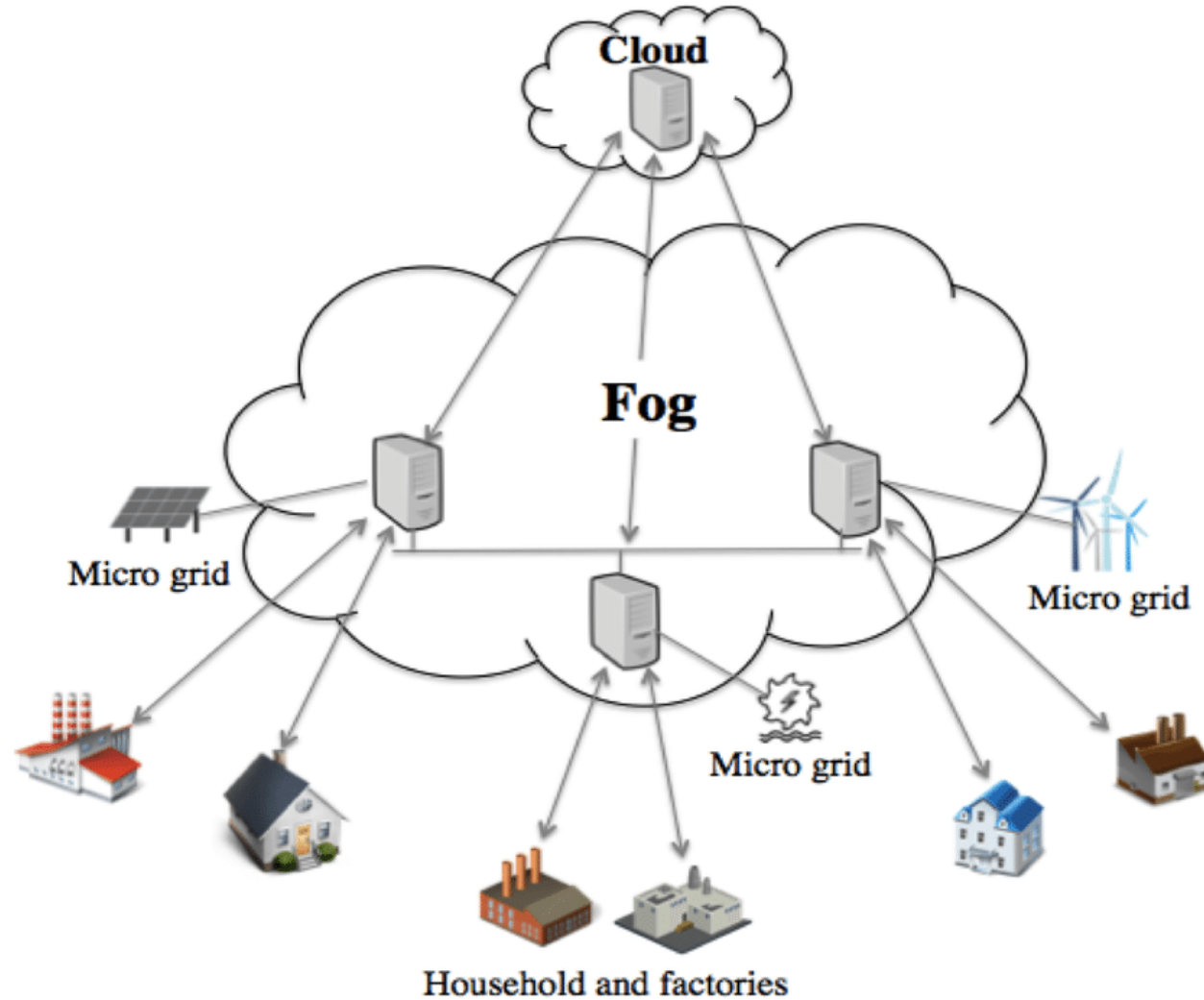


# Fog computing and IoT

## The Internet of Thing Architecture and Fog Computing



# Fog computing and Smart grids





# Fog Challenges

- Fog computing systems suffer from the issue of proper resource allocation among the applications while ensuring the end-to-end latency of the services.
- Resource management of the fog computing network has to be addressed so that the system throughput increases ensuring high availability as well as scalability.
- Security of applications/services/data.

# Resource Management of Fog Network

- Utilization of idle fog nodes for better throughput.
- More parallel operations.
- Handling load balancing.
- Meeting the delay requirements of real-time applications.
- Provisioning crash fault tolerance.
- More scalable system.

# Resource Management- Challenges

- Data may not be available at the executing fog node. Therefore, data fetching is needed from the required sensor or data source.
- The executive node might become unresponsive due to heavy workload, which comprises the latency.
- Choosing a new node in case of micro service execution migration, so that the response time gets reduced.
- Due to unavailability of an executing node, there is a need to migrate the partially processed persistent data to a new node (state migration).
- Final result has to be transferred to the client or actuator within very less amount of time.
- Deploying application components in different fog computing nodes ensuring latency requirement of the components.
- Multiple applications may collocate in the same fog node. Therefore, the data of one application may get compromised by another application. Data security and integrity of individual applications by resource isolation has to be ensured.

# Resource Management- Approaches

- Execution migration to the nearest node from the mobile client.
- Minimizing the carbon footprint for video printing service in fog computing.
- Emphasis on resource prediction, resource estimation and reservation, advance reservation as well as pricing for new and existing IoT customers.
- Docker as an edge computing platform. Docker may facilitate fast deployment, elasticity, and good performance over virtual machine based edge computing platform.
- Resource management based on the fluctuating relinquish probability of the customers, service price, service type, and variance of the relinquish probability.

# Security issues

- Major security issues are authentication at different levels of gateways as well as in fog nodes.
- Man-in-the-middle attack
- Privacy issues
- In case of the smart grids, the smart meters installed in the consumer's homes. Each smart meter and smart appliance has an IP address. A malicious user can either tamper with its own smart meter, report false readings, or spoof IP addresses.