Big Data Analytics →

Big Data (huge Data)
↳ quantity is very high.

* It is collection of data that is huge in volume → growing exponentially with time
↳ eg., Instagram

* **Big Data Analytics** →

process of examining large data sets containing a variety of data types to uncover hidden patterns, customer preferences.
↓
amazon

~~* Challenging~~

* **Challenges** → (Amazon story)

① Insufficient understanding & acceptance of bigdat
② Confusion while selecting Big Data tools.
③ paying lots of money.
④ Data integration. (Data ko alag alag jagah se collect karana)
⑤ Data Security

* Problems with Traditional large ~~scale~~ Scale Systems →

→ Majority of data comes in the form of semi structured or unstructed data.
↳ eg., image/vedio

→ Big data bahut hi teji ke sath grow hota hai jisse traditional system nahi handle kar sakte. Vo Keval steady data ko hi handle kar sakte hai.

→ Data is so expensive to store in traditional systems. Data is filtered & aggregated and large volume of data is thrown out. aur data ko minimize karane se accuracy reduce hoti hai.

* **Sources of Big Data** →

→ Social Networking sites (Instagram)
→ E-commerce sites (Amazon)
→ Weather station (Satellite)
→ Telecom company (Jio/Airtel)
→ Share Market (daily transaction)

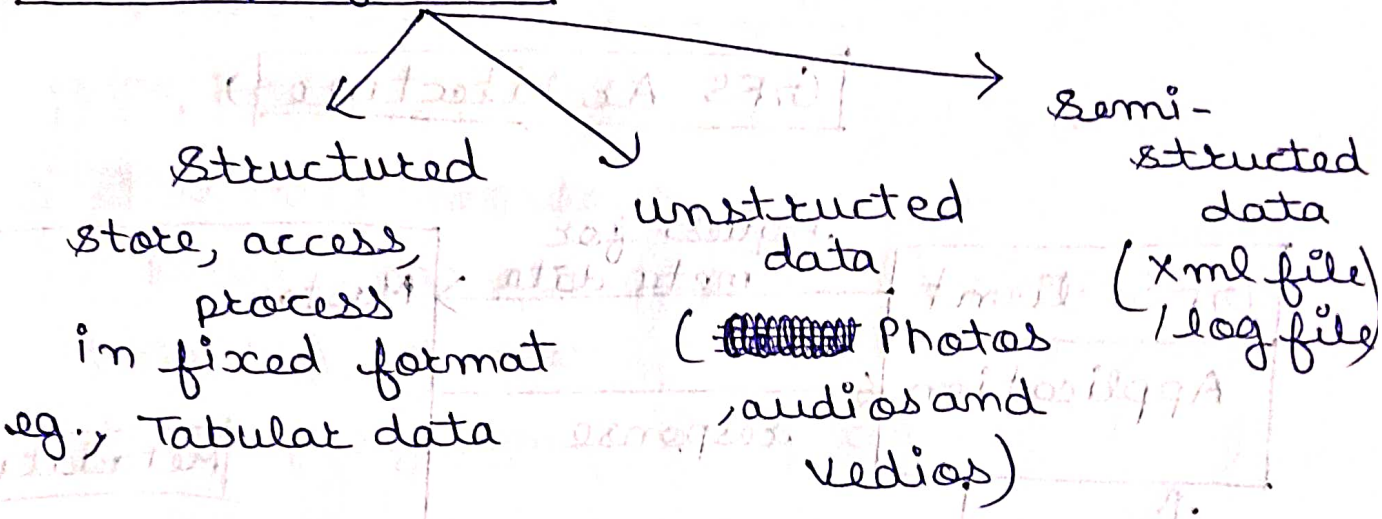
* **3V's of Big Data**

Velocity

Variety (photos, videos, gif, pdfs etc)

Volume

* **Types of Big Data**



Examples

Working with Big Data

* Google file system (GFS)

→ GFS is a scalable distributed file system.

→ GFS provide fault tolerance.

→ Runs on inexpensive hardwares.

→ Delivers high performance to the large number of clients.

→ files are organised heirarchically in directories and identified by path name.

→ Supports CRUD

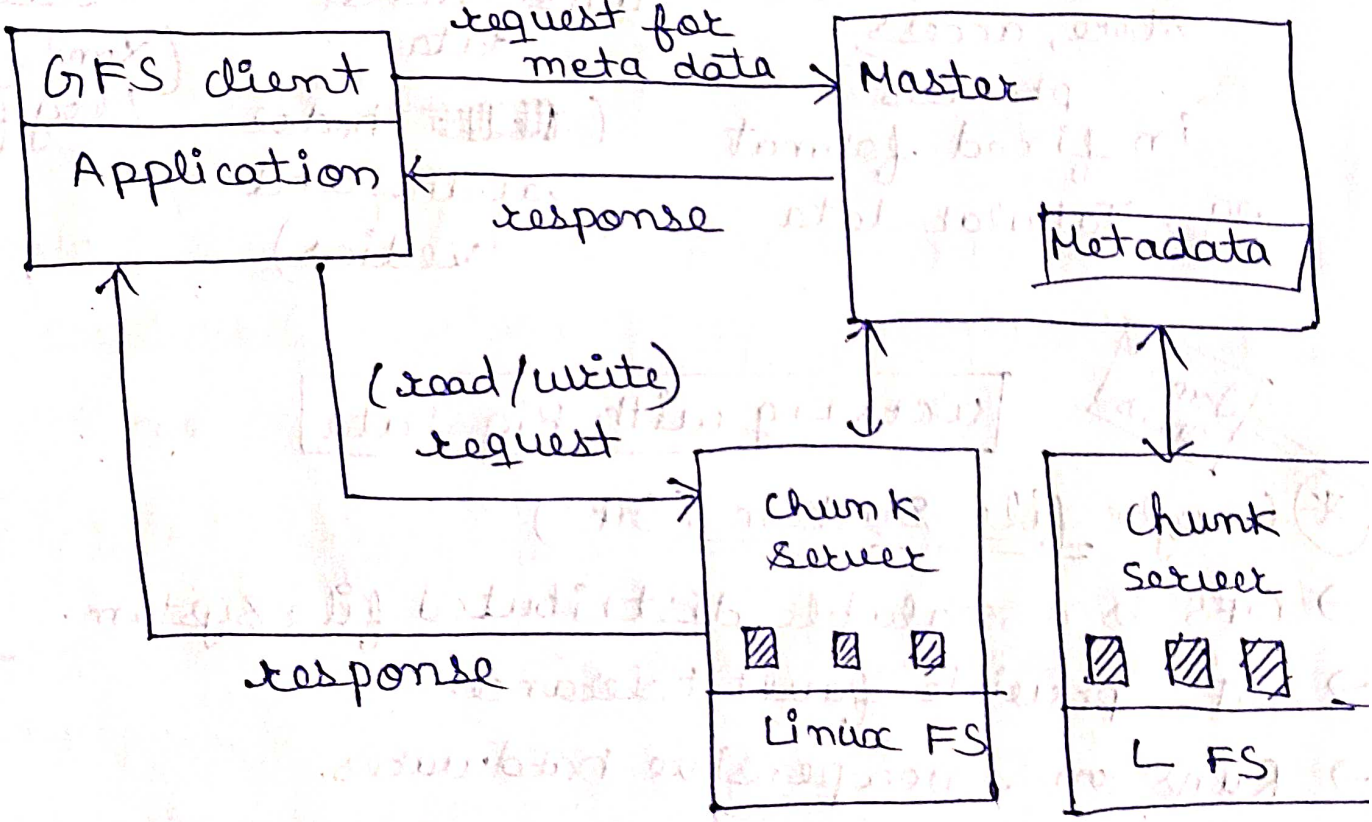
* Snapshot operation

file / folder ki copy banata hai low cost mai.

* Record operation

ek time par ek se jayada person ek file mai data append kar sakte hai

GFS Architecture →



* Hadoop Distributed file System →

→ based on GFS.

→ designed to run on large clusters (thousand of computers)

* Master / Slave architecture

↓ Name Node (contains meta Data)

↘ Data Node (actuall) data)

→ files split into several blocks and stored in a set of data nodes.

* **Building Blocks of Hadoop →**

① Name Node (Master)

→ stores metadata

→ kyuki hamare pass ek hi Name Node hota hai to agar ye fail hua to pura hadoop system fail ho jata hai.

② Data Nodes →

→ client can communicate directly with these nodes.

③ Secondary Name Node

→ assistant that monitors the state of the cluster

→ har cluster ke pass ek secondary Name Node hota hai.

→ ye Name Node se communicate kar sakta hai HDFS metadata ka Snapshot lene ke liye.

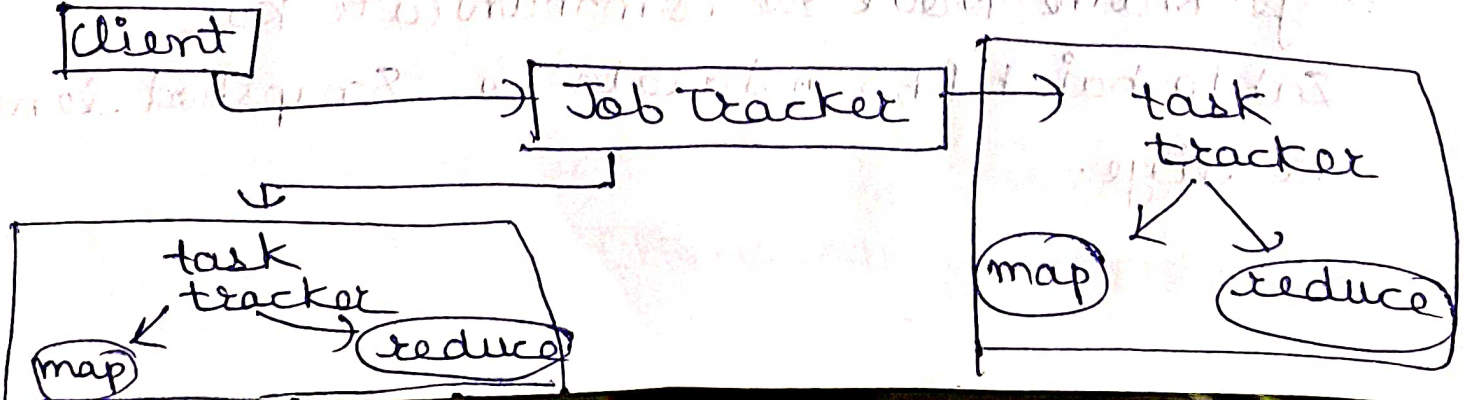
④ Job Tracker

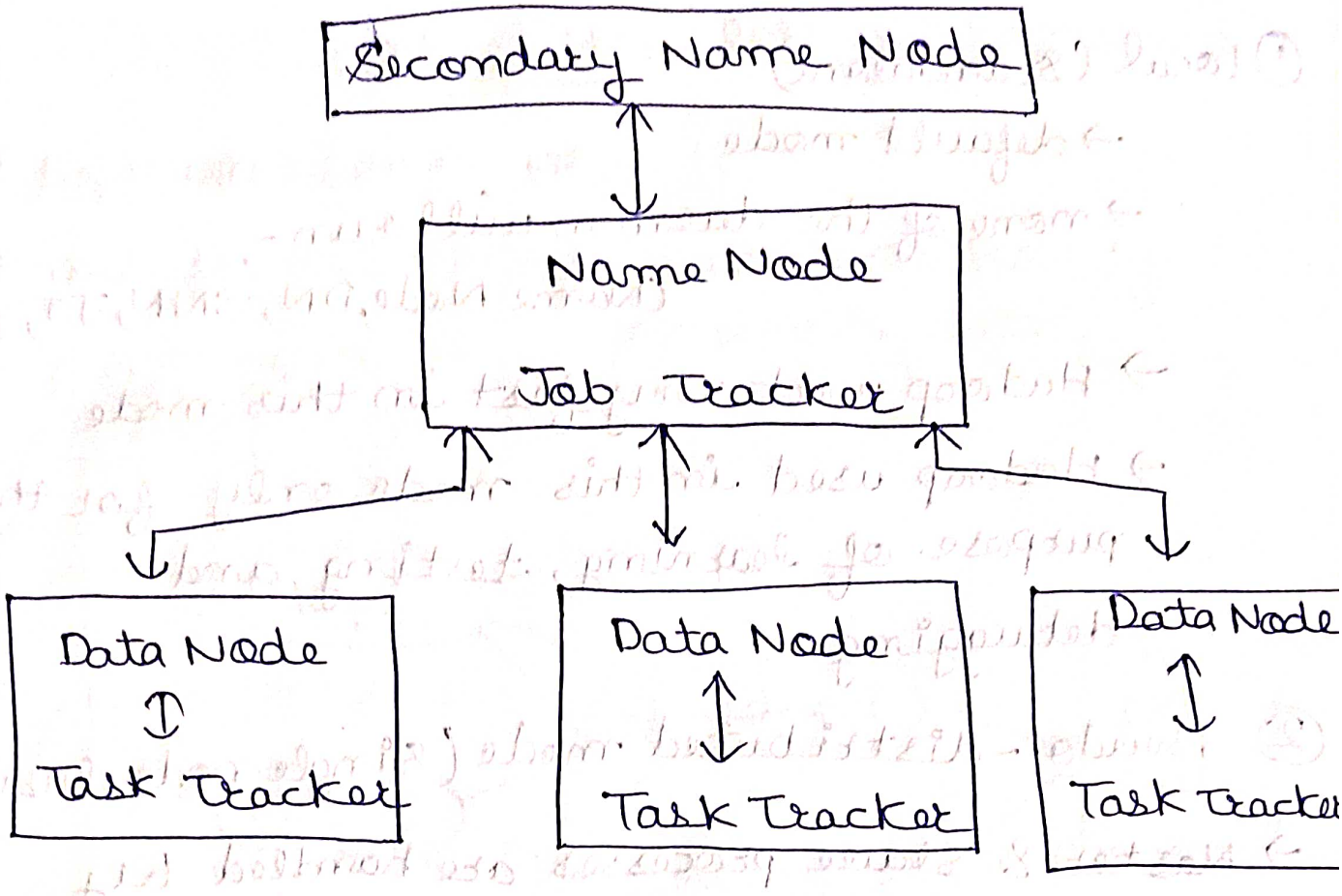
→ Link between your application and hadoop
→ har cluster ke pass ek Job Tracker hota hai
→ ye basically dekhta hai ki
  ① Kaunsi file pehle process hogi
  ② kaunsi file ko kis data node ko dena hai
  ③ aur task ko monitor karata hai aur agar koi task fail ho jata hai to usse re-launch karata hai.

⑤ Task Tracker

→ ye task karata hai aur constantly Job Tracker se communicate karata hai in the form of heart beat. aur agar ek time interval ke baad Job Tracker ko heart beat nahi milti to vo uss task ko fail man kar re launch kar deta hai.

Architecture





(*) Introducing and configuring Hadoop cluster →

→ we need to configure several xml files → configuration directory of Hadoop-Hom

Hadoop settings

hadoop-default.xml

→ Hadoop cluster can be configured by one of the three modes:

① Local (Stand alone) (default mode)

② Pseudo-distributed

③ fully-distributed

① Local (Standalone)

→ default mode

→ none of the daemon will run -
(Name Node, DN, SNN, JT, TT)

→ Hadoop work very fast in this mode

→ Hadoop used in this mode only for the purpose of learning, testing and debugging.

② Pseudo-distributed mode (Single node cluster)

→ Master & slave processes are handled by Single System.

→ All the processes inside cluster will run independently to each other.

→ All daemon will run seperately on various JVM (Java Virtual Machine)

→ used for development & debugging purpose

③ fully distributed mode (Multi Node cluster)

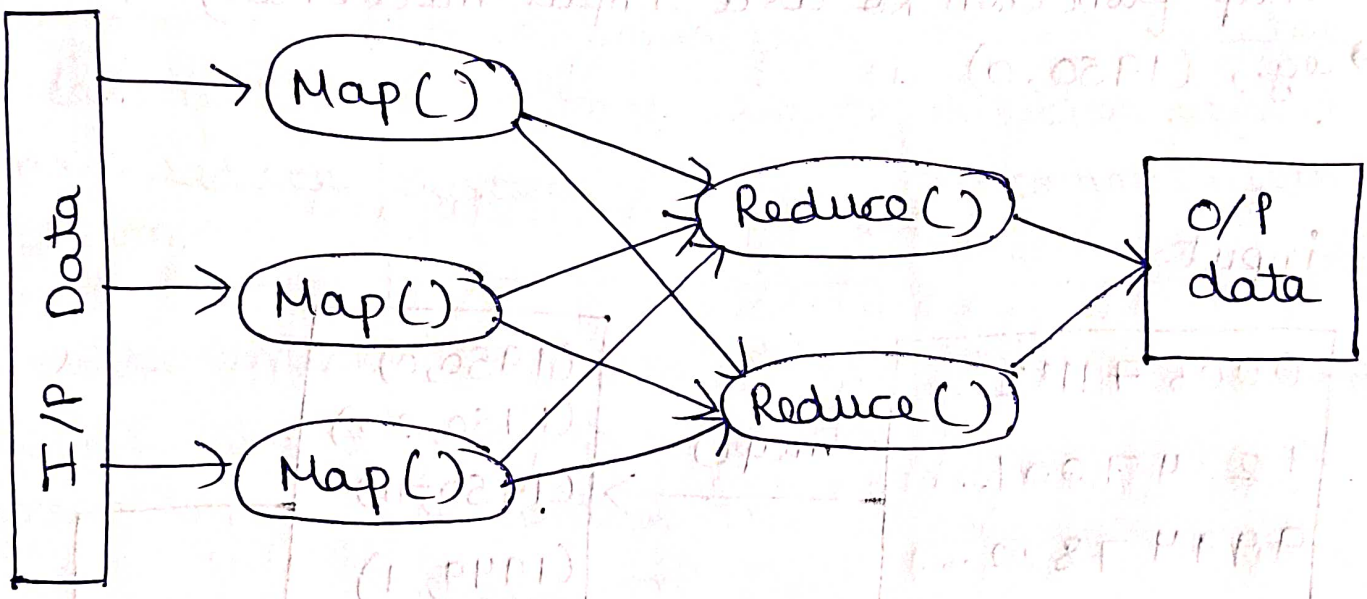
→ Isme bahut saare nodes hoti hai kuch Master Daemon ko chalati hai aur kuch slave Daemon ko chalati hai

→ ye production mode mai use hota hai

writing Map-Reduce program →

① Software framework

→ used for writing app which process big amount of data



→ Single Master (Job Tracker) and one Slave (Task Tracker) per cluster.

* **Weather Dataset** →

↳ Data Format → isme jo data use karate hai usse NCDC (National climatic data centre) se collect kiya Jata hai

→ line oriented ASCII format

→ Each line is a record.

→ Data files are organized by Date & weather station

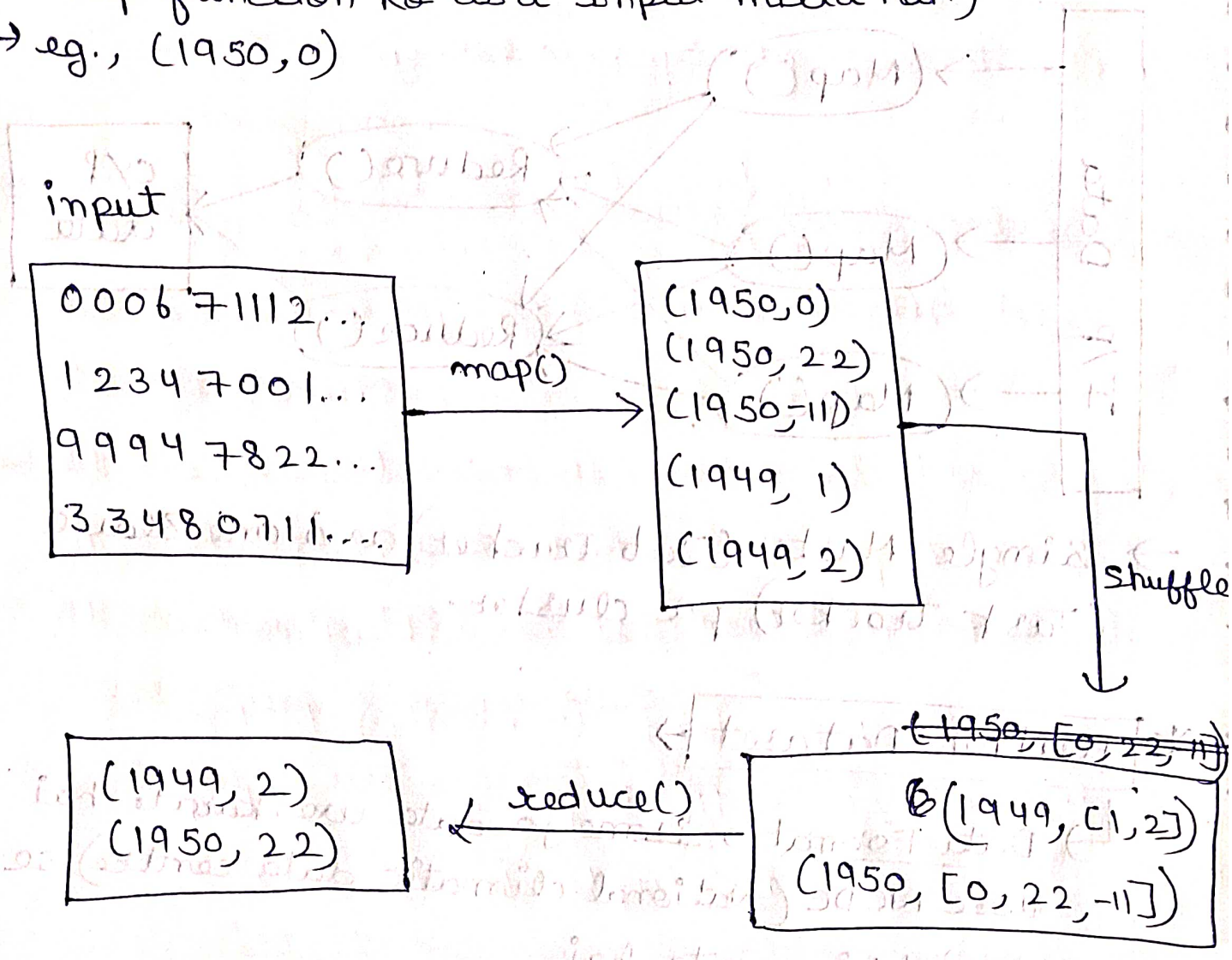
→ Analyzing the Data with Hadoop →

→ weather Data set → Map-Reduce Program → Hadoop

→ I/P → Raw data from NCDC.

→ Pull out year and the air temperature in map function. (matlab har year ka air temperature map function ko as a input milta hai.)

→ eg., (1950, 0)



* Old and New Java MapReduce API →

Difference	New API	Old API
① Mapper & Reducer	It uses Mapper & Reducer as class	as interface

② Package	New API is in org.apache.hadoop.mapreduce package	old API is in org.apache.hadoop.mapred package
③ user code to communicate with map-reduce System	use context object to communicate with map-reduce System	uses JobConf, OutputCollector, Reporter object to comm. with map-reduce System
④ ~~cotto~~ control Mapper & Reducer execution	It allows both mappers & reducers to control the execution flow by overriding the run() method.	Controlling mappers by writing a MapRunnable but no equivalent exists for reducers
⑤ Job control	Job control is done through the Job class	~~Job control was~~ done through JobClient
⑥ O/P file name	Map → part-m-nnnnn reduce → part-r-nnnnn	for both part-nnnnn
⑦ reduce() method pass value	Java.lang.Iterable	Java.lang.Iterator

Basic Program of Hadoop Map-Reduce →

→ Driver code → (initialize the Job) and (control ki kaha par O/P file store hongi)

```
// code

Configuration confg = new Configuration();

Job job = new Job(confg, "My word count Prog.");

Job.setJarByClass( Word Count.class);
Job.setMapperByClass (Map.class);
Job.setReducerByClass ( Reduce.class);
Job.setOutputKeyClass (Text.class);
Job.setOutputValueClass (IntWritable.class);
Job.setInputFormatClass (Text Input.class);
Job.setOutputFomatClass (Text OutputFormat.class);

Path output-path = new Path (args[1]);
FileInputFormat.addInputPath (Job, new path (args[0]));
FileOutputFormat.setOutputPath (Job, new path (args[1]));
```

for setting up the path (matlab O/P file kaha store hogi)

Mapper Code

```
public static class Map extends Mapper
                <longwritable, Text, Text, IntWritable>
{
  public void map (Longwritable key, Text value,
                                          (context)
  throws ~~Exception~~ IO Exception, Interrupted
                                          Exception
  {
    String line = value.toString();
    StringTokenizer token = new StringTokenizer
                                          (line);
    while (token.hasmoreToken())
    {
      value.set (token.nextToken());
      Context.write (value, new IntWritable(1));
    }
  } // void map close
} // class close
```

---

*Reducer Code →

```
public static class Reduce extends Reducer
                <Text, IntWritable, Text, IntWritable>
{
  public void reduce (Text key, IntWritable value
    , context) throws IOException, ~~Interr~~ Interrupted
    Exception

  {
```

```
int sum = 0
for ( IntWritable x : values)
 {
   sum += x.get();
 }
 Context.write(key, new IntWritable(sum));
 }
}
```

⊛ **Record Reader** → (This comes before mapper)

↓

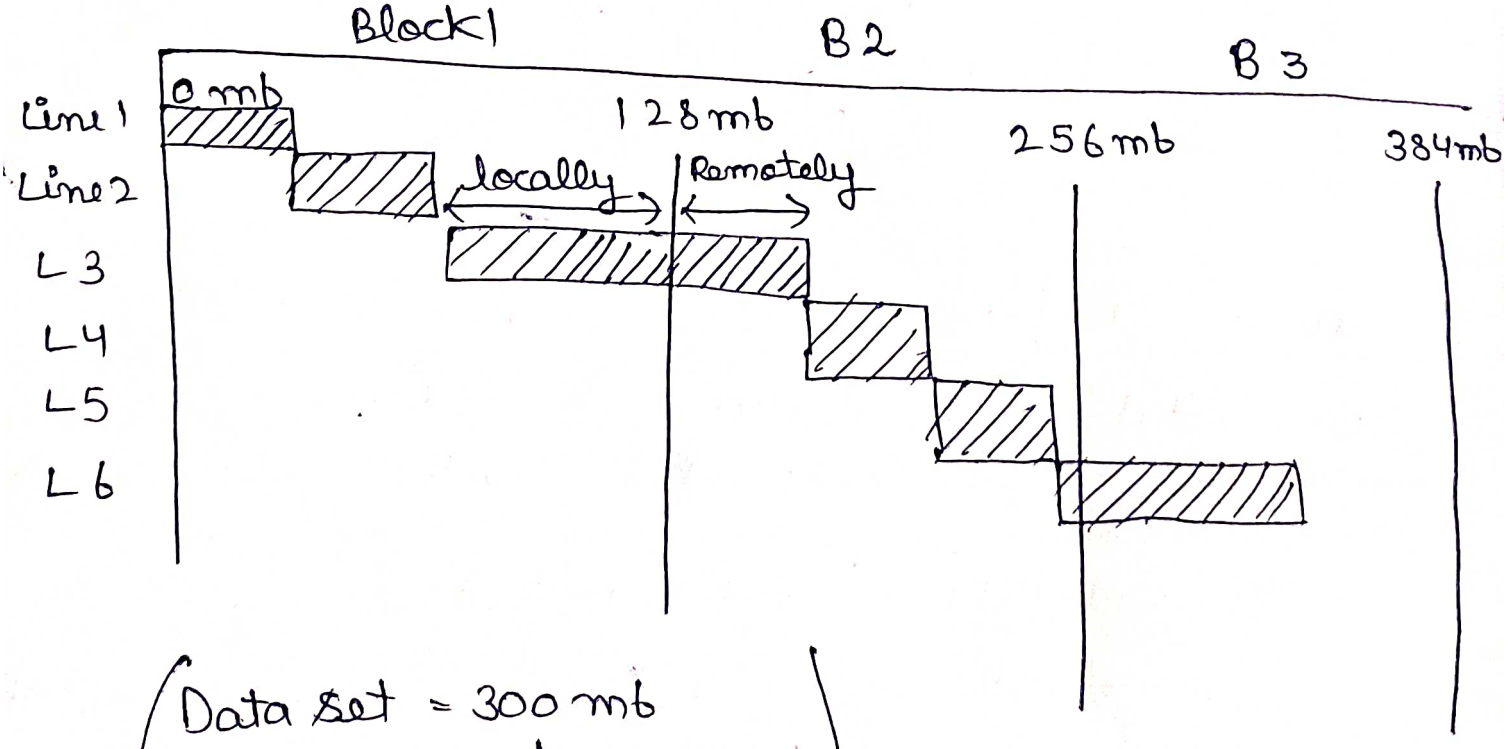
Data/Input files

→ Loads data & convert into key value pair

→ Its Instance is defined by the InputFormat

→ Text InputFormat – Default

→ Key associated with each line is its byte offset.



(Data Set = 300 mb
6 lines each of 50 mb)

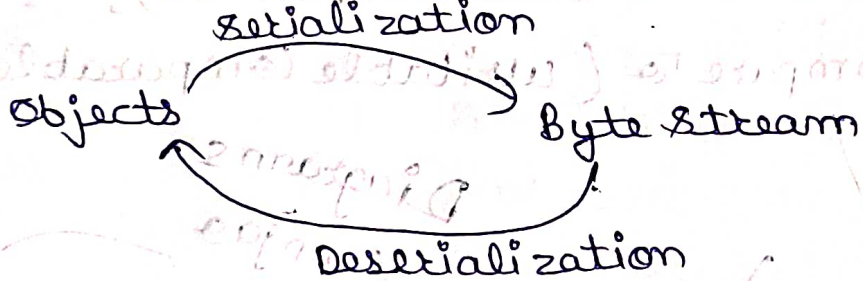
* Combiner → It is used to summarize the map output records with the same key.
(Semi Reducer)

* Partitioner → See (Map-Reduce Basic diag) → Controls the partitioning of the keys of the Intermediate map O/P.

→ Configure() and getPartition()

Chapter → 3

* Writable Interface →



→ Hadoop uses its own ~~Seri~~ Serialization format known as Writables

→ Two Method → Data output (write Byte Stream)
Data Input (read → Byte stream ko object mai convert karke read karata hai)

Program →

```
Package org.apache.hadoop.io.
import java.io.*;
public interface Writable
{
    void write (DataOutput out);
    void readFields (DataInput in);
}
```

Note → If writable is not present in Hadoop. than it uses the serialization of Java which increases the data over head in network.

* **Writable comparable and comparator** →

* Sub interface of Writables

```
public interface WritableComparable extends
                    Writable, Comparable
{
    void write (DataOutput out);
    void readFields (DataInput in);
    int compareTo (WritableComparable O);
}
```

- returns (-1 0 1)

Diagram se samjao

$(K_1, V_1) \rightarrow$ Map $\rightarrow (K_2, \text{List}[V_2])$

$(K_2, \text{List}[V_2]) \rightarrow$ Reduce $\rightarrow (K_3, V_3)$

WritableComparable is used here

Program

```
package org.apache.hadoop.io
import java.util.*;
public interface RawComparator
                    extends Comparator
{
    public int compare (byte[] b1, int s1
        , int l1, byte[] b2, int s2, int l2);
}
```

*Writable class →

→ Hadoop provides classes that wrap the Java primitive types and implement the Writable (Comparable & Writable interface.

→ provided in org.apache.hadoop.io package

- → set() → ~~store~~ set wrapped value
- → get() → retrieve " "

Java primitive	Writable Implementation	bytes
byte	~~Writ~~ Byte Writable	1
short	Short Writable	2
int	Int Writable	4
	V Int Writable	1-5
long	Long Writable	8
	V Long Writable	1-9
float	Float Writable	4
double	Double Writable	8
boolean	Boolean Writable	1

~~Text~~
Some Writable class

Text → Writable for UTf-8 sequence

feature → Indexing charAt()
Iteration
Mutability (change ho sakti hai string)

Byte Writable →

↳ Byte writable ki value ko change kar sakte hai by calling its set() function

Syntax

```
ByteWritable b = new ByteWritable
                 (new byte[] {3, 5});
```

* Null writable →

↳ used as placeholder & store empty value

* Object and Generic Writable →

object

→ Java Primitive
→ string
→ enum
→ null

* useful when a field can be of more than one type

Generic

better than object writable because in object writable we append the class declaration as a string into the output file in key value pair.

* **Writable Collections** →

Array writable classes
- → Array Writable (1 D)
- → 2D Array writable

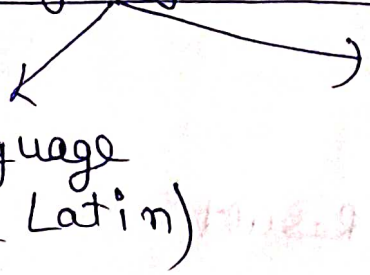
Map writable class
- → Abstract Map Writable
- → Map Writable
- → Sorted M.W

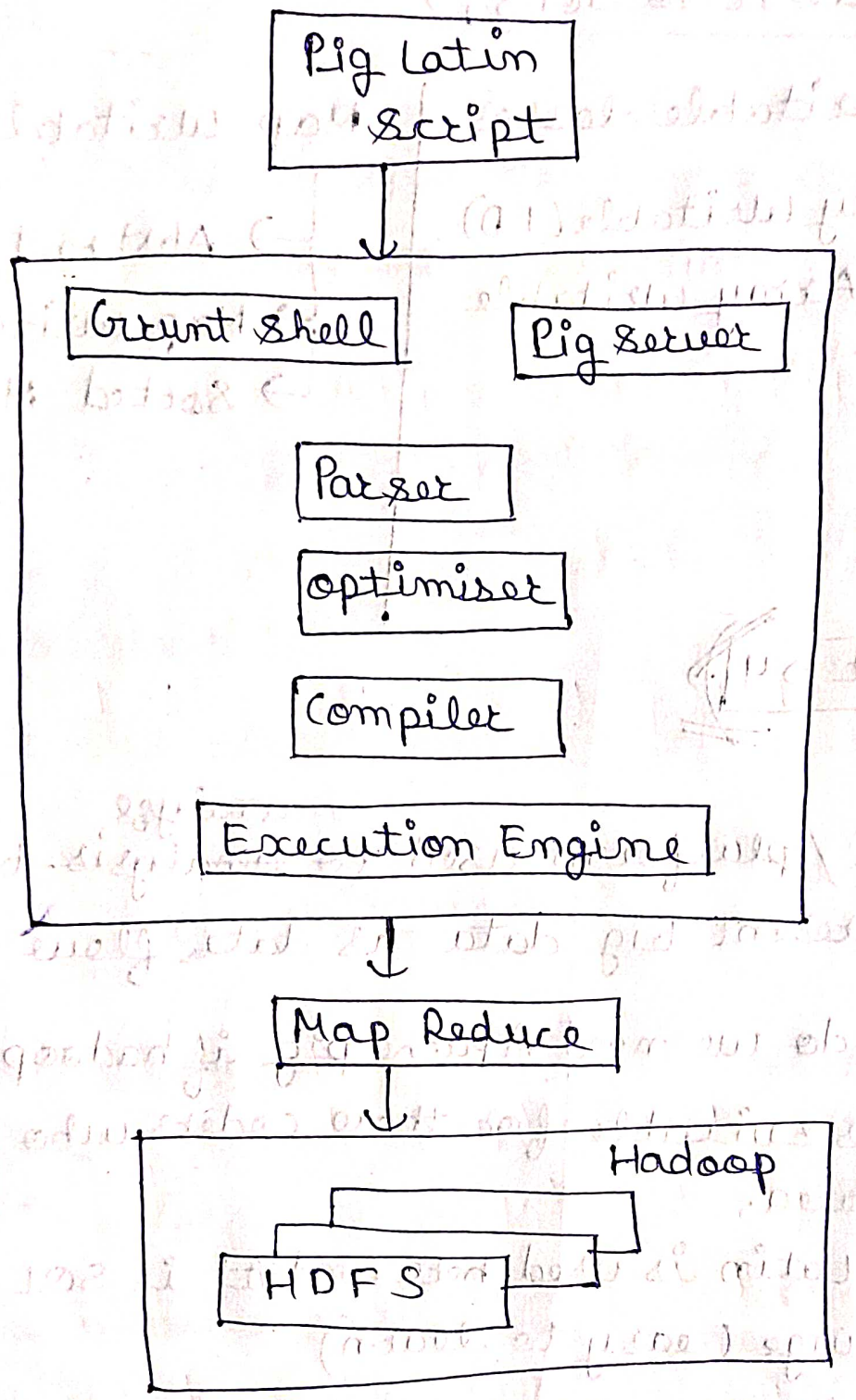
Chapter → 4

Pig
→ tool/platform used to ~~analysis~~ analyze big data.
→ Represent big data as data flow

* why do we need Apache pig if hadoop exist?
→ It is suitable for those coders who are not good in Java.
→ Pig Latin is used here and it is SQL like language (easy to learn)
→ provides many built in operators (eg., joins, filters)
→ provides nested data types (tuples, map, bags)

* **Admiring Pig Architecture** →





→ Going with Pig Latin Application flow →

```
A = LOAD 'data_file.txt'
B = GROUP.....;
C = FILTER.....;   } transformation functions

DUMP B;

STORE C INTO 'Result'
```

Working through the ABC's of Pig latin →

- Keep it simple

- Make it smart (coders ko jayada dimaag na lagana pade development mai)

- Don't limit Development

kyunki iska main point hi yahi tha ki jisse jaisa nahi aata vo isse use kare.

* **Looking at Pig data types & Syntax**

Two data Types

Scalar (contains single value)

Complex (contains other type)

for eg., **Atom** - for eg., "Laveesh", 1, 1.2

tuple → Sequence of field

Bag → Collection of non unique tuple

map → Collection of key value pairs

* Evaluating Local & distributed Modes of Running Pig script →

Local Mode →

→ useful when we have small data set
→ useful for developing & testing Pig Latin
→ Hadoop is not Required
→ Pig Program runs in Context of local JVM

Distributed / MapReduce / Hadoop Mode →

Pig Script → MapReduce Jobs → runs → Hadoop Cluster

* Checking out the Pig script Interface →

Pig script ko likhne ke tareeke

- **Script** .pig — Eg., Flight Data.pig

- ~~Script~~ commands are Interpreted by PL compiler & executed in the order determined by Pig optimizer.

- **Grunt**
  ↳ It is a shell in which we can write small pig codes.

- **Embedded** → we can use other languages also to execute Pig Latin Statements

* Scripting with pig latin →

→ ~~We can~~

Pig is designed to be extensible via UDFs (User defined function)

→ Some of the Pig UDFs → LOAD/ STORE function

→ Pig can be embedded in most languages such as Java, Python.

→ Pig doesn't support control flow statement

→ but If you want to use control flow than use embedded in other language.

Chapter → 5

Hive

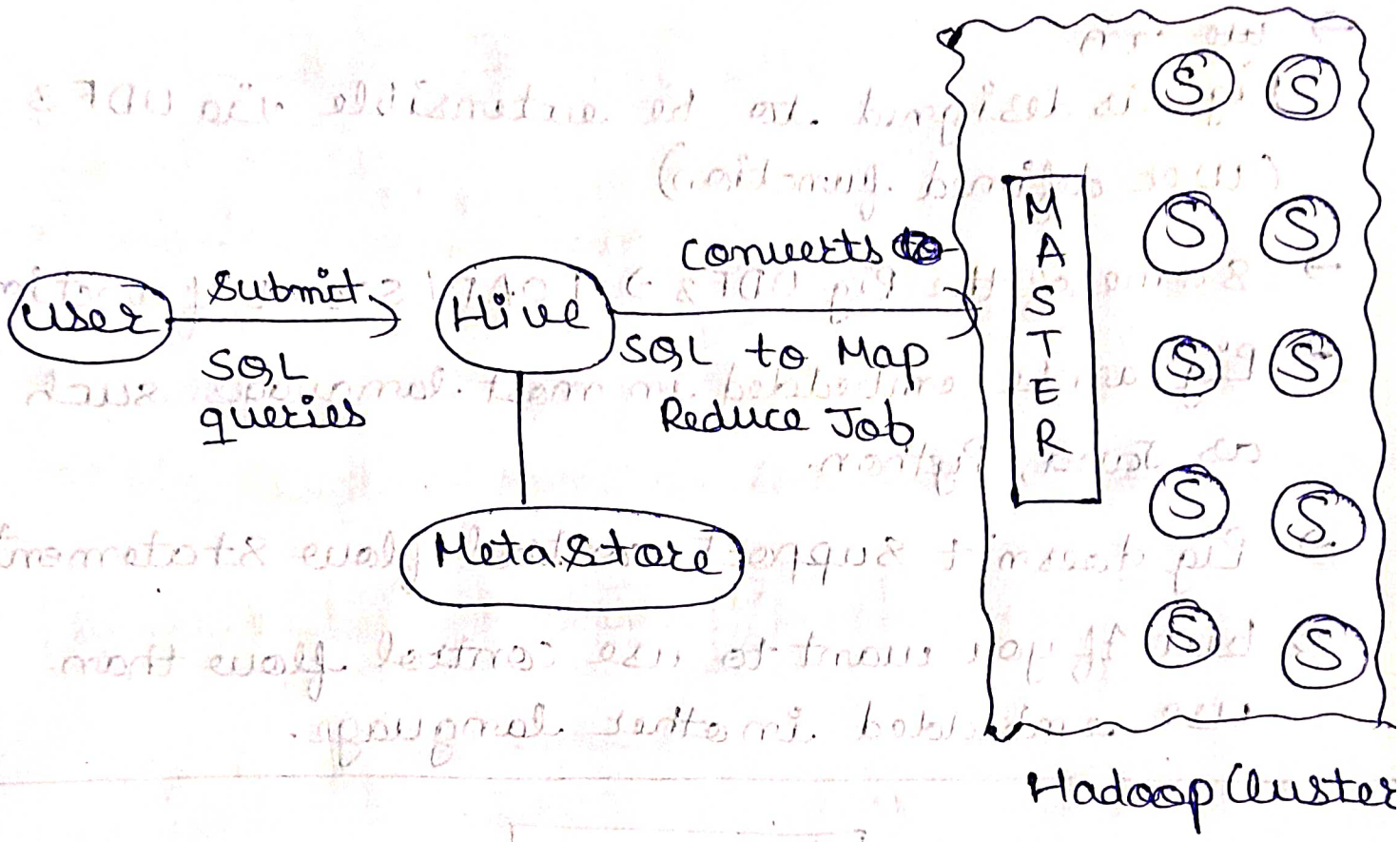
→ It is a open source data warehouse system for querying, analyzing large data set stored in Hadoop file

→ { Analyze, querying, data summarize }

→ It uses HiveQL

SQL + Hive = HiveQL

→ Highly ~~Scal~~ Scalable



Master bigger task ko sub task mai divide karke slaves ko de deta hai

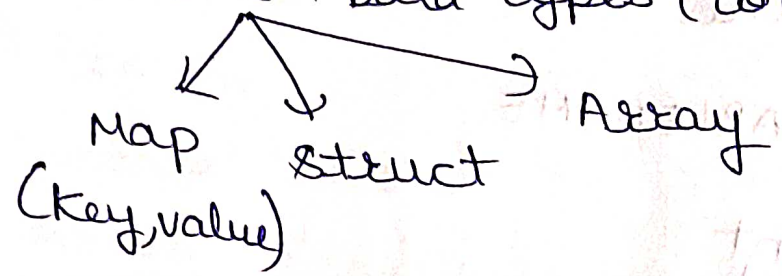
Topic → 2

Hive Data Types → simple, complex

*Simple

Data types	Size
Tiny int	1 byte
Small int	2 byte
int	4 byte
large int	8 byte
float	4 byte
double	8 byte
String	"John"

* Collection Data Types (complex Data types)



Topic → 3

* Different types of Hive Table

① Managed Table

```
CREATE TABLE TABLENAME (col1 datatype,
col2 datatype, col3 datatype) row format
delimited fields terminated by '' STORED as
textfile;
```

② External Table

```
CREATE EXTERNAL TABLE - - - -
- - - - - - - - - - - - - -
- - - - - - - - - - - - - -
```

* Difference

Managed Table	External Table
① Jab hum isse drop karenge tab data + Meta data will be deleted	① only metadata is deleted.

Some Hive Commands

① DESCRIBE TABLENAME

O/P

employeeid int

employee name string

② show tables → show all the tables

~~Insert Statement~~

③ Display

↳ SELECT * FROM TABLENAME

④ Insert

↓

INSERT INTO TABLE TABLENAME VALUES (108, 'Lavesh'), (109, 'Shubham');

⑤ drop table (delete the table)

drop table tablename;

⑥ WHERE clause

SELECT * FROM TABLENAME WHERE condition;

⑦ ALTER STATEMENT

→ Rename Table, add column, remove col

* Rename Table

ALTER TABLE TABLENAME RENAME TO NEWTABLENAME

* add columns

ALTER TABLE TABLENAME ADD COLUMNS (newCol1 datatype, newCol2 datatype);

* Replace / Remove columns

ALTER TABLE TABLENAME REPLACE COLUMNS (col1 datatype, col2 datatype);

for eg.,

employeeid int

employeename ~~string~~ int

→ ALTER TABLE TABLENAME REPLACE COLUMNS (employeename string)

→ after that table will be

employeename string

* Truncate vs delete.

↓	↓
deletes rows not schema	deletes rows as well as schema

* Delete

DELETE from TABLENAME WHERE cond^{n}

* Truncate

↓

delete rows not schema

TRUNCATE TABLE TABLENAME

* DROP (Deletes entire table)

DROP TABLE TABLENAME