CSC 555 and DSC 333
Mining Big Data
Lecture 3

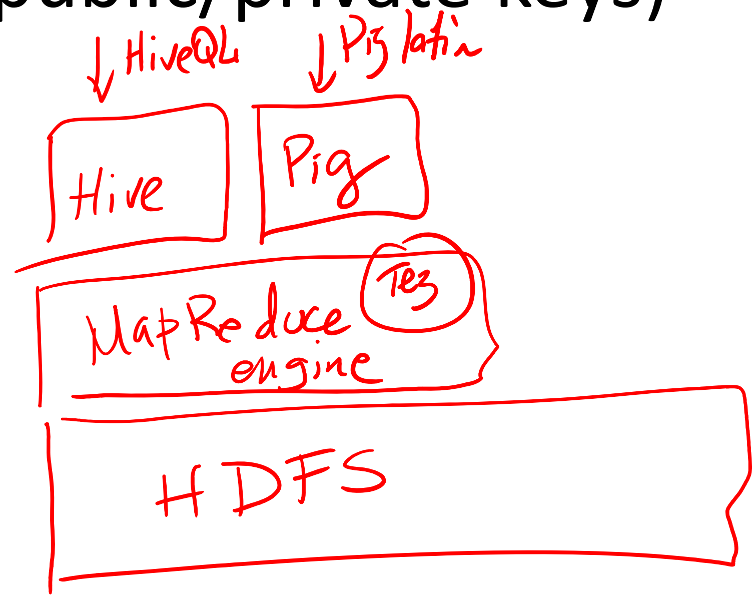
Alexander Rasin

College of CDM, DePaul University

September 28th, 2021

Tonight

- Virtual instances
- MapReduce
- Cryptography (public/private keys)
- Hive
- Pig



Cloud Service

- A virtual machine
  - Simulated environment
  - Guaranteed performance
  - On a shared (larger) machine
- <u>Independent</u> from your computer

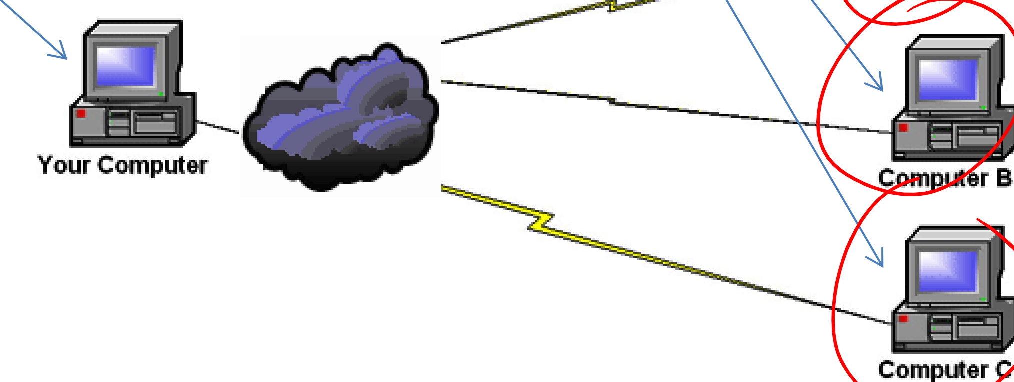
Hadoop/MapReduce

- Many loosely connected computers
- Limited centralized control

- Therefore
  - Mapper only sees a <u>block at-a-time</u>
  - Reducer only sees its keys
  - Reducer emits <u>one "result" per key</u>

Remote Access

- Remote connection (such as remote desktop)
- Work is done at the destination
- Your computer is the terminal



Host Names/IP

- Nodes addressed by
  - Internet Protocol (IP) Address
    - 140.192.5.61
  - Domain Name
    - www.depaul.edu
- IP and Domain name are interchangeable

Conventions

- localhost = “This machine”
- Hosts
  - ec2-54-187-35-9.us-west-2.compute.amazonaws.com
  - 54.187.35.9

54-187-35-9

Student / Faculty

- Student
  - ID, Name, Age, Year, Advisor
  - Table RDBMS / Raw file in HDFS
    - 1 | Jane Doe | 37 | 2021 | 3
    - 2 | Mike Smith | 33 | 2023 | 2
- Faculty
  - ID, Name, Area, Year
    - 2 , Eric S. , DS , 2021 , 3
    - 3 , Tanu M. , CSC , 2023 , 2

Set Difference (MINUS)

if line.count('|') == 4 :
    execute Mapper1S code
else:
    execute Mapper2F code

- MapReduce has to parse both files

Mapper1S ID S_4
Mapper2F ID F_R

Reducer ID

- SQL:

  SELECT ID FROM Student

  MINUS (or EXCEPT)

  SELECT ID FROM Faculty;

X 5 {S_4, S_4, F_R}
✓ 6 {S_4, S_4, S_4}
X 7 {F_R, F_R}

Difference

- Process two input files
- Need to identify <u>which key</u> is <u>from where</u>
- Reduce
  - Iterate through all keys
  - Check that only left side key appears
    - Emit output
- Does order matter?

SQL Join

Mapper1S
Mapper2F
Reducer

Key | Advisor | ID | ID/Advisor

Name_S | Area_F | Name + Area

Alex_S, CSC_F

5 {Alex, CSC}
6 {DSC, Math, CSC}
DSC_F, Math_S, CSC_F
DSC_F, '_S', 'NULL_F'

- SQL:

    SELECT Student.Name, Faculty.Area

    FROM Student, Faculty

    WHERE Student.Advisor = Faculty.ID;

- Or:

    SELECT Student.Name, Faculty.Area

    FROM Student JOIN Faculty ON Advisor = Faculty.ID;

Join by MapReduce

- Mapper
  - Process both files
  - Identify source of each row
- Reducer (for same key)
  - Collect all values from left file
  - Collect all values from right file
  - Return all (matching) combinations

Generalizing Join

S.First = F.First AND
S.Last = F.Last

First_Last

- Join by 2 attributes (A, B)
  - Rewrite both Mappers to use (A+B) as key
  - Leave Mappers as is and compare/join by the 2nd attribute in the Reduce function
- Join without equality? (non-equi-join)
  - E.g., (Student.Age > Faculty.Age)

Grouping/Aggregation

- Key-based aggregation
  - MR join is also similar to an aggregate
- Mapper
  - Same as for join
- Reduce
  - Collect all values
  - Perform aggregation or join

SQL Join + Aggregation

- Join + Aggregate:

    SELECT Faculty.Name, COUNT(*)
    FROM Student, Faculty
    WHERE Student.Advisor = Faculty.ID; MR#1
    GROUP BY Faculty.Name; MR#2

Mapper1S Advisor -S
Mapper2F ID Name
Reducer1 Name

Tanu
Tanu
Tanu
Jay
Jay

key value
Mapper3 Name -
Reducer Name Count(*)

Tanu 3
Jay 2

Complex Queries

- N-way joins
- ~~Non-equi join~~
- Complex predicates
- Complex aggregation

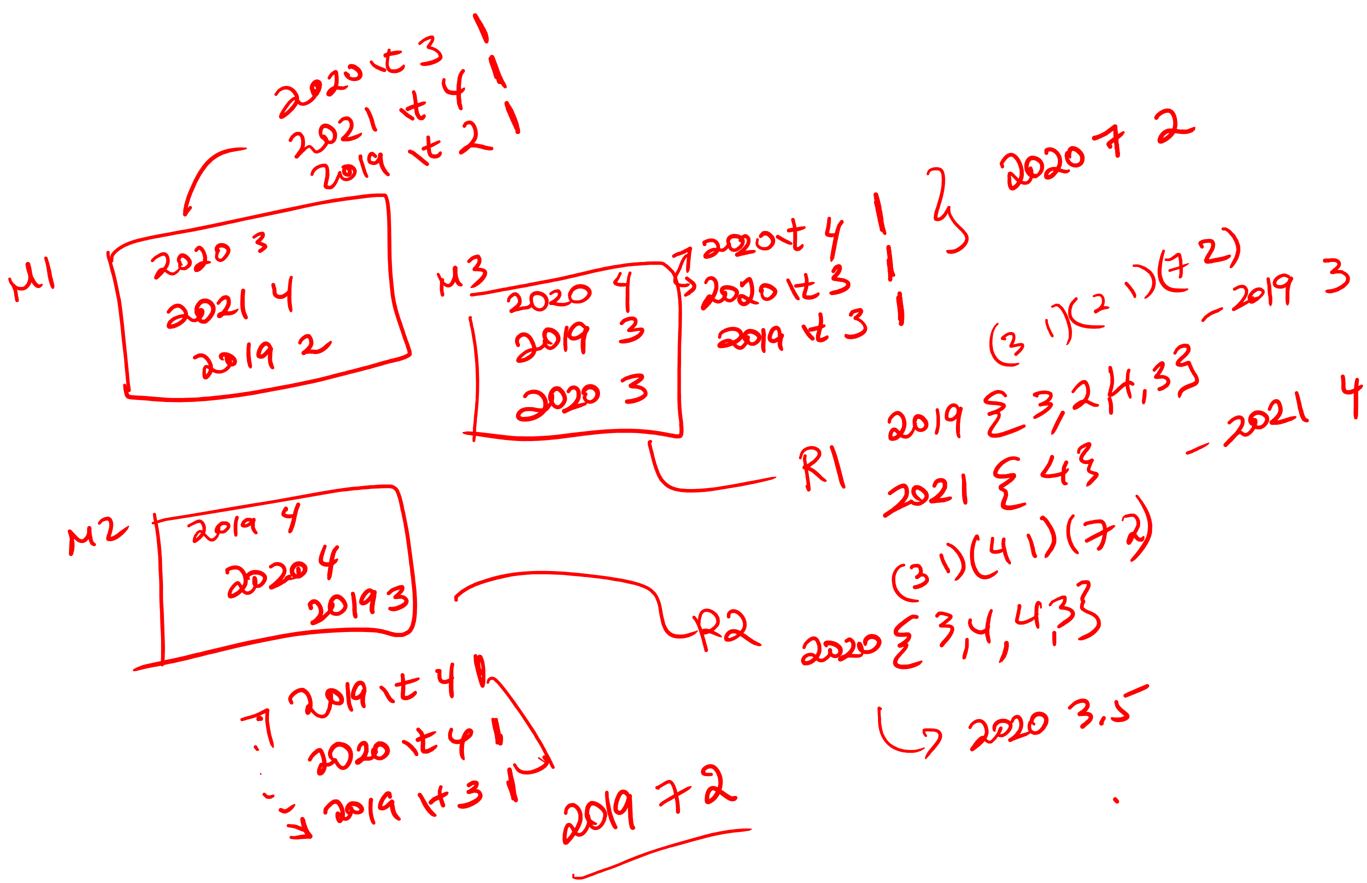
Computing GPA (by year)

Select Year, AVG(Grade)
FROM Records
GROUP BY Year

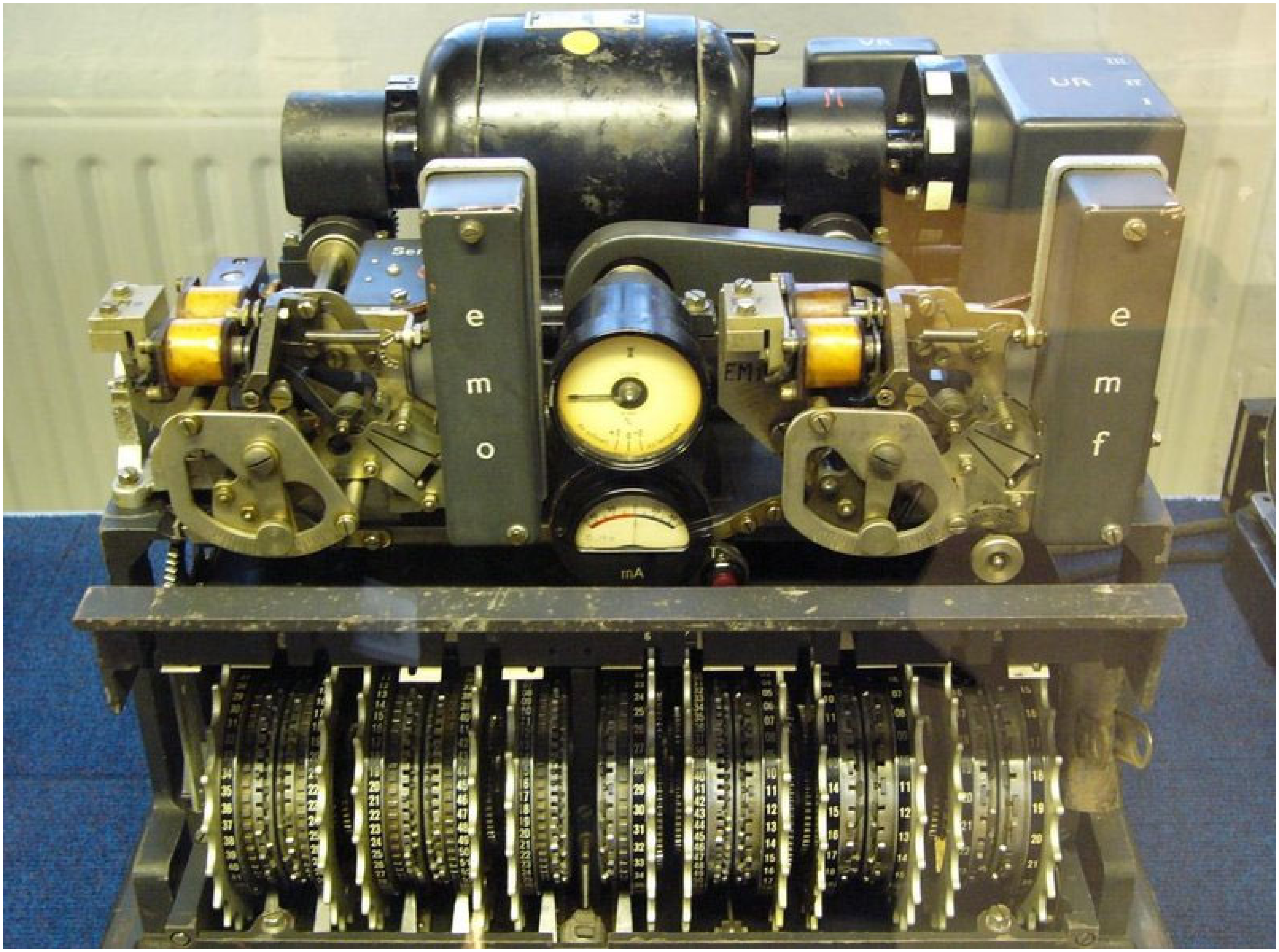
- Map
  - Read/parse the file
  - Emit pairs (year, grade)
- Reduce
  - Sum up all the enrollment values
  - Divide by the count of values
  - Produce (year, avg_grade) output
- Combiner?

	key	value
Mapper	Year	Grade
Reducer	Year	AVG(Grade)

	Key	Value
Mapper	Year	Grade 1
Combiner	Year	SUM(Grade) SUM(1)
Reducer	Year	$\frac{\text{SUM(sums)}}{\text{SUM(counts)}}$



Cryptography



Distributing the Password

- Password-protected access
- How do you share the password?
  – (without the risk of compromising it)

Password → Hash('Password')

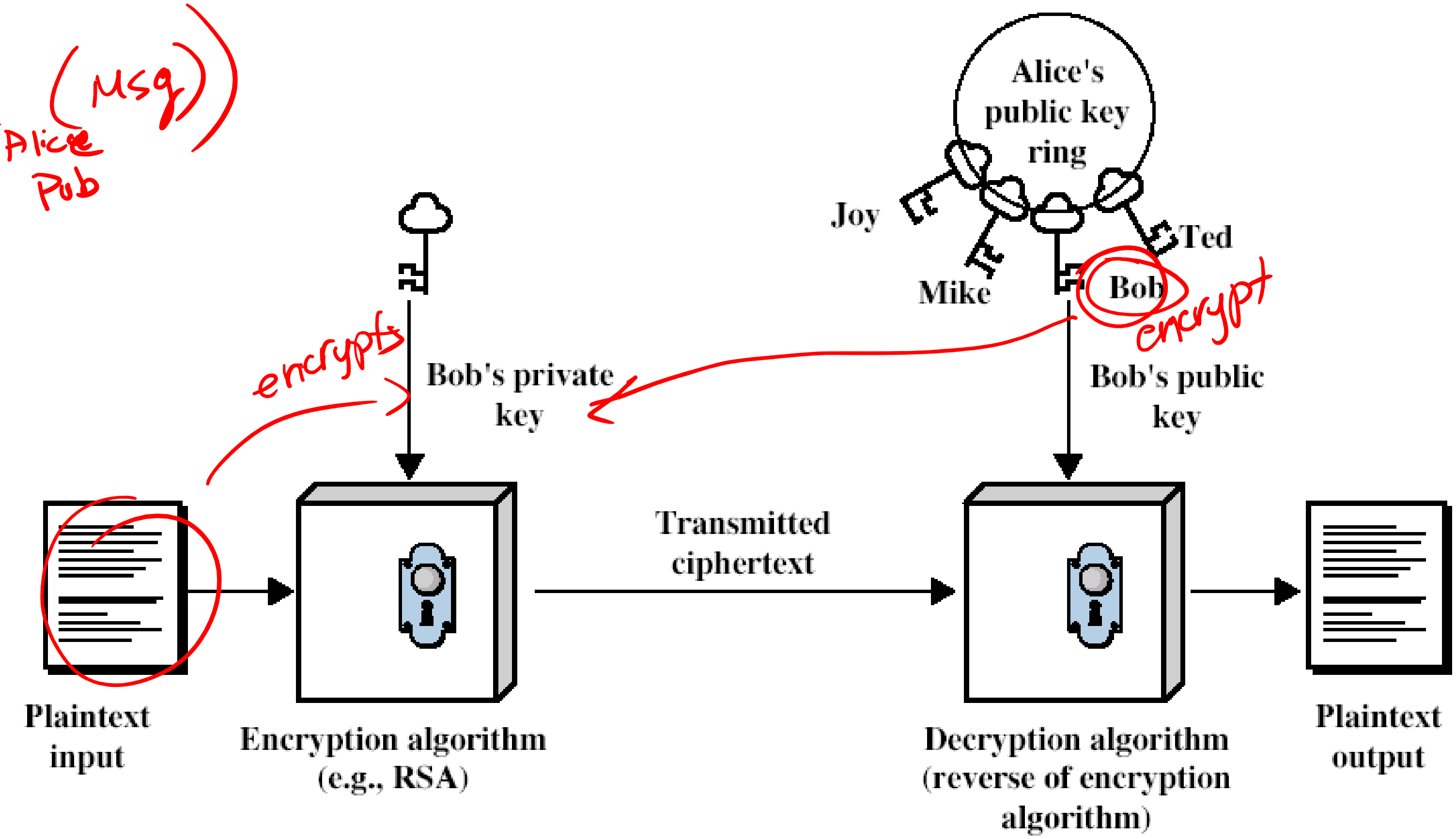
- Asymmetric encryption!
  – Use a joint pair of keys
  – Public key + Private key

Public/Private Key

- Private key
  - Known only to the owner
  - Secret code
- Public key
  - Known to everyone and shared freely
    - Need a reliable way to publish
  - Can be used to verify private key authenticity

Asymmetric Encryption for Authentication

$E_{Bob\ priv}(E_{Alice\ Pub}(Msg))$

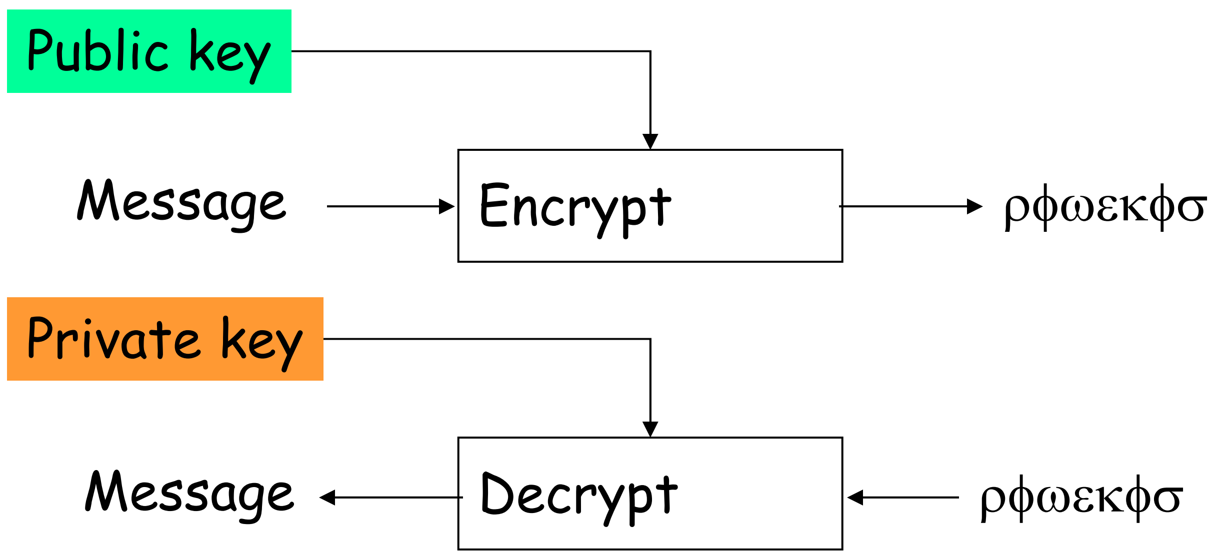


Bob

Alice

Public key Encryption

- Alice has a key pair: public and private
  - **publish** the public key such that the key is publicly known
  - Alice keeps the private key secret
- Other people use Alice's public key to encrypt messages for Alice
- Alice uses her private key to decrypt
- Only Alice can decrypt since only Alice has the private key



- Trick: To compute the private key from the public key is a difficult problem.


7:40

RSA

- Invented by Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key scheme
- Use large integers (e.g. 1024 bits) 2^{1024}
- Security due to cost of factoring large numbers
  - factorization is expensive

RSA Key Setup

- Each user generates a public/private key pair by
  - select two large primes at random: p, q
  - compute their system modulus $n=p \cdot q$
    - note $\varnothing(n)=(p-1)(q-1)$
  - select at random the encryption key e
    - where $1<e<\varnothing(n)$, $\gcd(e,\varnothing(n))=1$
  - solve following equation to find decryption key d
    - $e \cdot d=1 \bmod \varnothing(n)$ and $0\le d\le n$
  - publish their public encryption key: KU= $\{e,n\}$
  - keep secret private decryption key: KR= $\{d,n\}$

RSA Usage

- To encrypt a message M:
  - sender obtains public key of receiver $KU=\{e,n\}$
  - computes: $C=\boxed{M}^e \bmod n$, where $0 \le M < n$
- To decrypt the ciphertext C:
  - receiver uses its private key $KR=\{d,n\}$
  - computes: $M=\boxed{C}^d \bmod n$
- Message M must be smaller than the modulus n (cut into blocks if needed)

$M^{e^d} = M^{d^e} = 1 \bmod n$

RSA Example: Computing Keys

1. Select primes: $p=17$, $q=11$
2. Compute $n=pq=17\times 11=187$
3. Compute $\varnothing(n)=(p-1)(q-1)=16\times 10=160$
4. Select e: $\gcd(e,160)=1$ and $e<160$
   - choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d<160$
   - $d=23$ since $23\times 7=161=10\times 160+1$
6. Publish public key $KU=\{7,187\}$
7. Keep secret private key $KR=\{23,187\}$

RSA: Encryption and Decryption

- Given message $M = 88$ ($88<187$)
- Encryption $KU=\{7,187\}$:

  $C = 88^7 \bmod 187 = 11$

  ~~PE~~ Encrypt Bob_public

- Decryption $KR=\{23,187\}$:

  $M = 11^{23} \bmod 187 = 88$

- $e*d \equiv 1 \pmod{p-1}$ and
- $e*d \equiv 1 \pmod{q-1}$
- Therefore $m^{ed} \equiv 1 \pmod{pq}$

Matrix Multiplication

- Matrix times Vector

$$\mathbf{AB} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \times x + b \times y \\ c \times x + d \times y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}$$

- Unit of operation
  – Value-by-value
  – Vector-by-vector

Example Queries:

```
SELECT StoreID, MIN(TransactionAmt), MAX(TransactionAmt)
FROM StoreTransactions
GROUP BY StoreID;

SELECT ZipCode, MEDIAN(Age)
FROM People
GROUP BY ZipCode

SELECT d.category, COUNT(*)
FROM Employee e, Department d
WHERE e.dept = d.id
GROUP BY d.category;
```

Key #1

Key #2

	key	value	
Mapper	Trans ID	TransAmt	TransAmt
Combiner	TransID	min(TransAmt)	max(TransAmt)
Reducer	TransID	min(TransAmt)	max(TransAmt)
Mapper	ZipCode	Age	
Reducer	ZipCode	Median(Age)	
Mapper 1E	Dept	_E	
Mapper 1D	ID	Category_D	
Reducer	ID/Dept	Category	
Mapper 2	Category	1	
Reducer 2	Category	Sum(1s)	

Hive Execution Flow

- Parse the query
- Get metadata from MetaStore
- Create a logical plan
- Optimize the plan
- Create a physical plan
  - DAG of MR jobs

Hive Types

- TINYINT – 1 byte integer
- SMALLINT – 2 byte integer
- INT – 4 byte integer
- BIGINT – 8 byte integer
- BOOLEAN – True/False
- FLOAT/DOUBLE
- STRING

Hive Examples

CREATE TABLE u_data (userid INT,

movieid INT,

rating INT,

unixtime STRING)

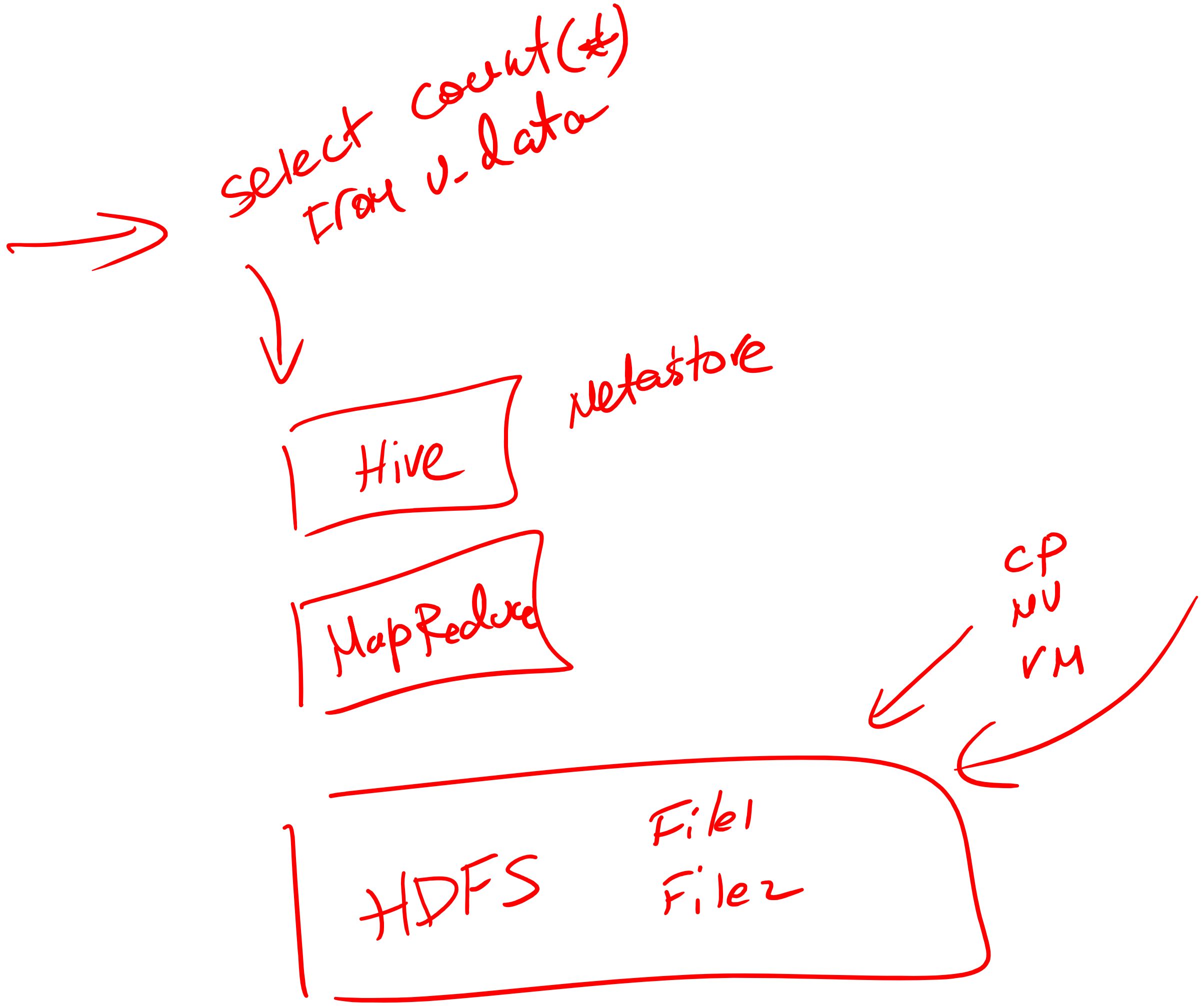
ROW FORMAT DELIMITED FIELDS

TERMINATED BY '\t' STORED AS TEXTFILE; (not compressed)

- show tables; describe u_data;
- wget http://www.grouplens.org/system/files/ml-100k.zip

LOAD DATA LOCAL INPATH 'ml-100k/u.data'

OVERWRITE INTO TABLE u_data;


select count(*)
from u_data
Hive
Metastore
MapReduce
HDFS
File1
File2
cp
mv
rm

Using Hive

- SELECT COUNT(*) FROM u_data;
- SELECT * FROM u_data WHERE userid = 449;
- SELECT userid, AVG(rating) from u_data GROUP BY userid;
- SELECT userid, AVG(rating) as AR from u_data GROUP BY userid ORDER BY AR;

Custom MapReduce Plug

```
CREATE TABLE u_data_new ( userid INT,
movieid INT, rating INT, weekday String)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY '\t';
add FILE weekday_mapper.py;
INSERT OVERWRITE TABLE u_data_new
SELECT TRANSFORM (userid, movieid, rating, unixtime)
USING 'python weekday_mapper.py'
AS (userid, movieid, rating, weekday) FROM u_data;
```

Using Hive

```
SELECT weekday, COUNT(*)
FROM u_data_new GROUP BY weekday;

SELECT weekday, COUNT(*)  as Total
FROM u_data_new GROUP BY weekday
ORDER BY Total;
```

Sampling in Hive

```
SELECT COUNT(*) FROM u_data
TABLESAMPLE(BUCKET 1 OUT OF 100 ON rand());

SELECT COUNT(*) FROM movie_ratings2
TABLESAMPLE(BUCKET 4 OUT OF 50 ON movieid);

CREATE VIEW MovieSample AS
SELECT * FROM movie_ratings2
TABLESAMPLE(0.1 PERCENT);
```

Joins in Hive

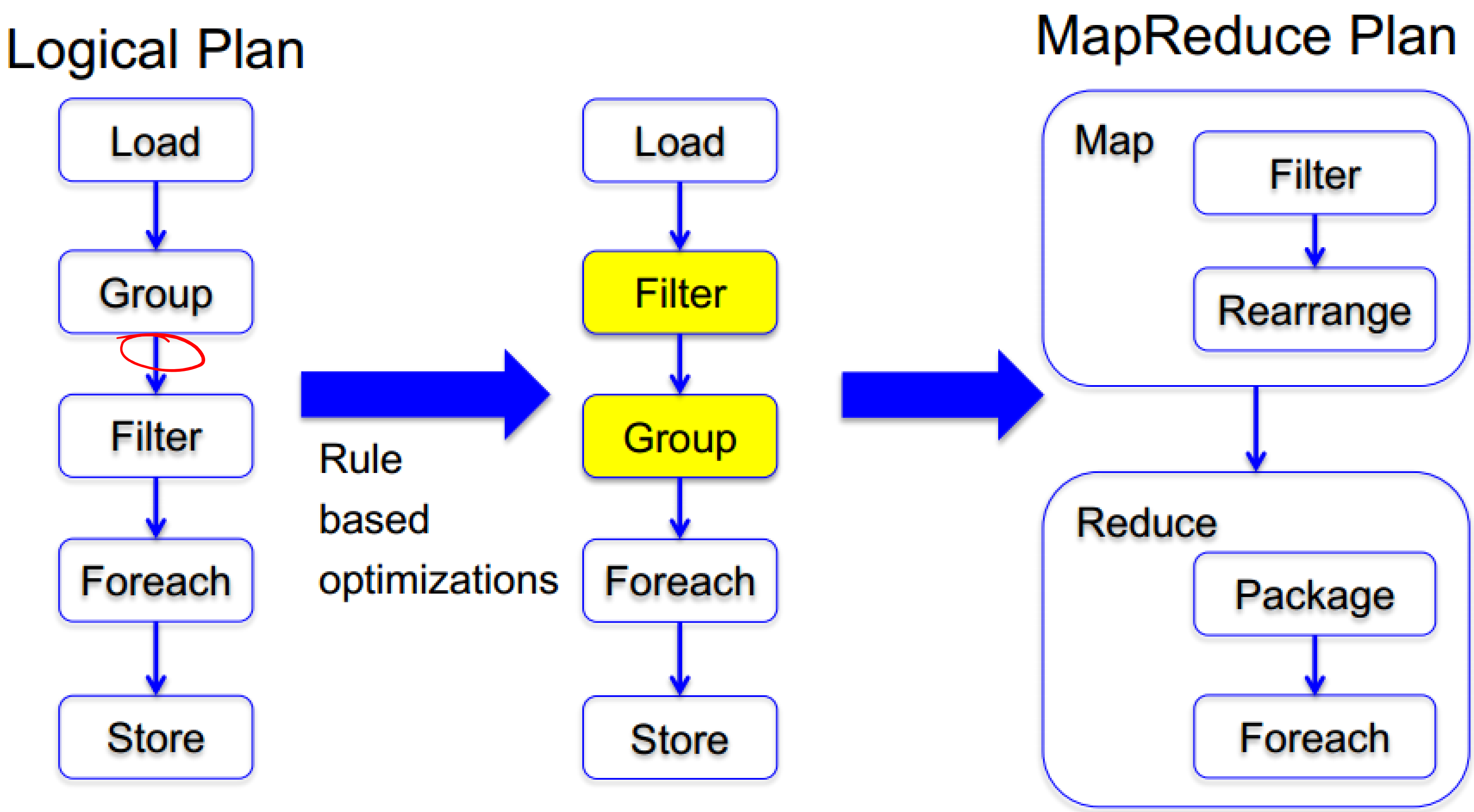
```
SELECT COUNT(*)
FROM u_data JOIN MovieSample ON
(u_data.movieid = MovieSample.movieid);

CREATE VIEW JoinView1 AS
SELECT *
FROM u_data JOIN MovieSample ON
(u_data.movieid = u_data_new.movieid)
WHERE Rating > 3;
```

Outer Join / External Table

- External Tables
  - Data not owned by Hive
- Describe u_data;
- Describe u_data_new;
- SELECT * FROM u_data
  FULL OUTER JOIN u_data_new ON
  (u_data.rating = u_data_new.weekday)
  WHERE u_data.rating > 3;

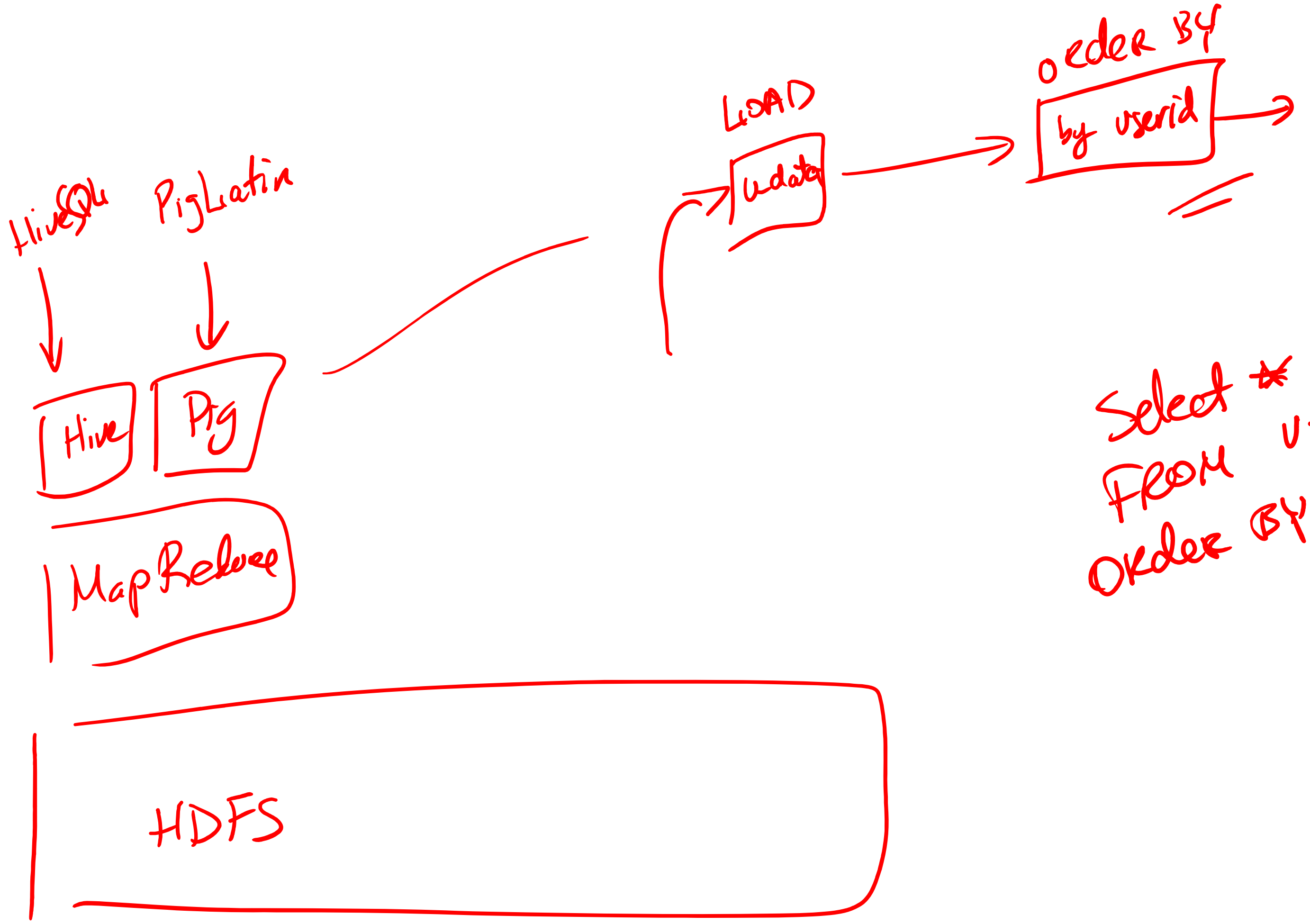
Pig Architecture



Pig Use

```
UData = LOAD 'u.data' USING PigStorage('\t') AS
(userid:int, movieid:int, rating:int,
unixtime:chararray);
DESCRIBE UData;
DUMP UData;

UDataS = ORDER UData BY userid;
DUMP UDataS;
```



Select *
FROM u-data
ORDER BY userid

We Have Stopped Here

Pig Use

```
UDataSample = SAMPLE UData 0.01;
DUMP UDataSample;
STORE UDataSample INTO 'UDataSample'
USING PigStorage ('_');

ONE_USER = FILTER UData BY userid == 251;
DUMP ONE_USER;
```

Pig Use

```
GoodRatings = FILTER UData BY rating > 2;
UserSet = GROUP GoodRatings BY userid;
DUMP UserSet;
UserRatings = FOREACH UserSet GENERATE
COUNT(GoodRatings);
DUMP UserRatings;
ILLUSTRATE UserRatings;
```

Pig Use

UserSet2 = GROUP UData BY userid;

UserRatings2 = FOREACH UserSet2 GENERATE UData.userid, AVG(UData.rating);

DUMP UserRatings2;

Next Time:

- Hadoop ecosystem
  – Hadoop config, Hadoop Streaming
  – More Hive and Pig
- Read:
  – Mining of Massive Datasets
    - Sections 2.5
  – Hadoop: The Definitive Guide
    - Pp162-167, "Launching a Job" to "Retrieving the Results"
    - Pp204-205, "Speculative Execution"