CSC 555 and DSC 333
Mining Big Data
Lecture 6

Alexander Rasin

College of CDM, DePaul University

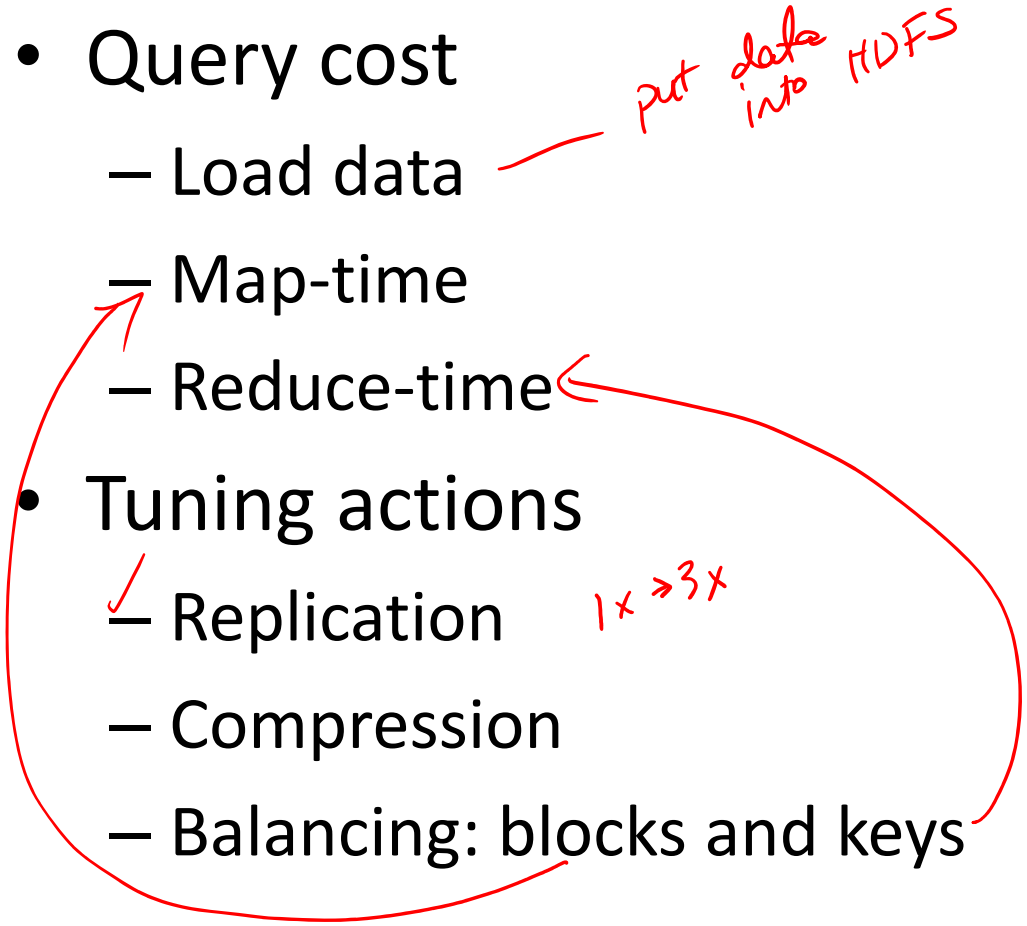
October 19th, 2021

Tonight

- Performance considerations and compression
- NoSQL
- Multi-node cluster setup
- Hadoop streaming join

Performance

- Query cost
  - Load data
  - Map-time
  - Reduce-time
- Tuning actions
  - Replication
  - Compression
  - Balancing: blocks and keys



Replication

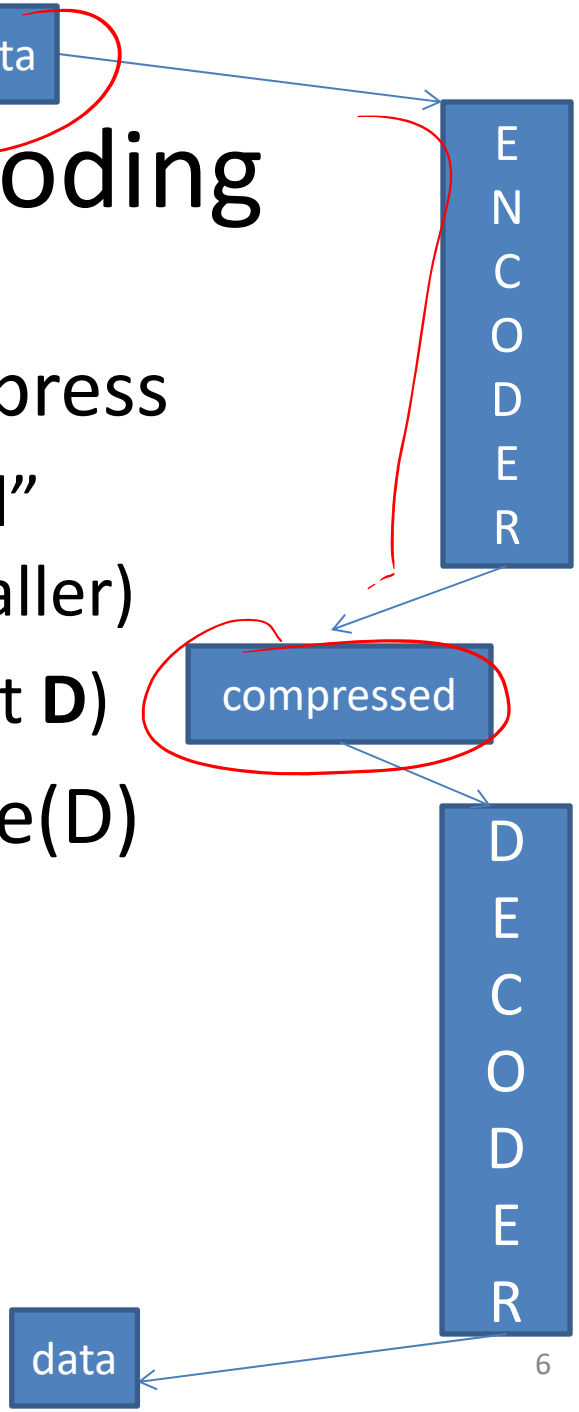
- Number of times a block is copied in HDFS
- Expected performance change?
  - Load
  - Map-Runtime
  - Reduce-Runtime
- Configuration (JobConf)
  - setMapSpeculativeExecution
  - setReduceSpeculativeExecution

Data Compression

- Reduce the data set (file)
  - Save space when storing the file
  - Save time when reading/sending the file
  - Pay overhead for compression/decompression
- Applications
  - Gzip, Zip, etc.
  - MP3, JPEG, MPEG, …
  - Built-in (databases, file systems, etc.)

Encoding and Decoding

- Data: binary data set **D** to compress
  - Encode: generate a "compressed" representation D_c (hopefully smaller)
  - Decode: reconstruct **D** (or almost **D**)
- Compression ratio: Size(D_c)/Size(D)
- Lossless/Lossy
  - Text, files
  - Images, MP3, Video, …

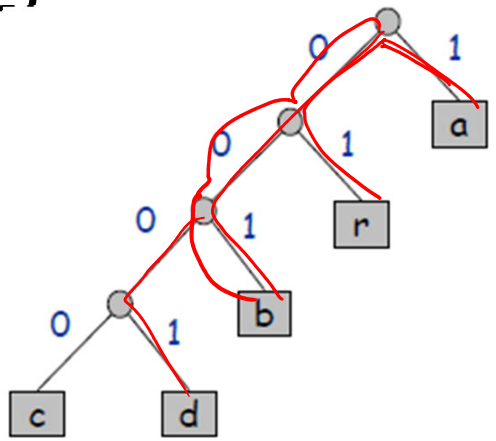


Simple Ideas

- Run Length Encoding (RLE)
  - Convert (AAAAACCCCDDDDDDD)
  - Into (A,5;C,4;D,7)
- Fixed length (dictionary) coding
  - Assign codes to every letter
    - E.g., a = 00001, b = 00010, … z = 11000
  - Encode character => code
  - Have to store the mapping

Variable Length Encoding

- Fewer bits for frequent letters (E, T, A)
- More bits for rare letters (J, Q, Z)
- Potential ambiguity
  - (E=0, T=01, Z=0001101)?
- Use a binary tree



char	encoding
a	1
b	001
c	0000
d	0001
r	01

- 11001000010100011

Compression Quality

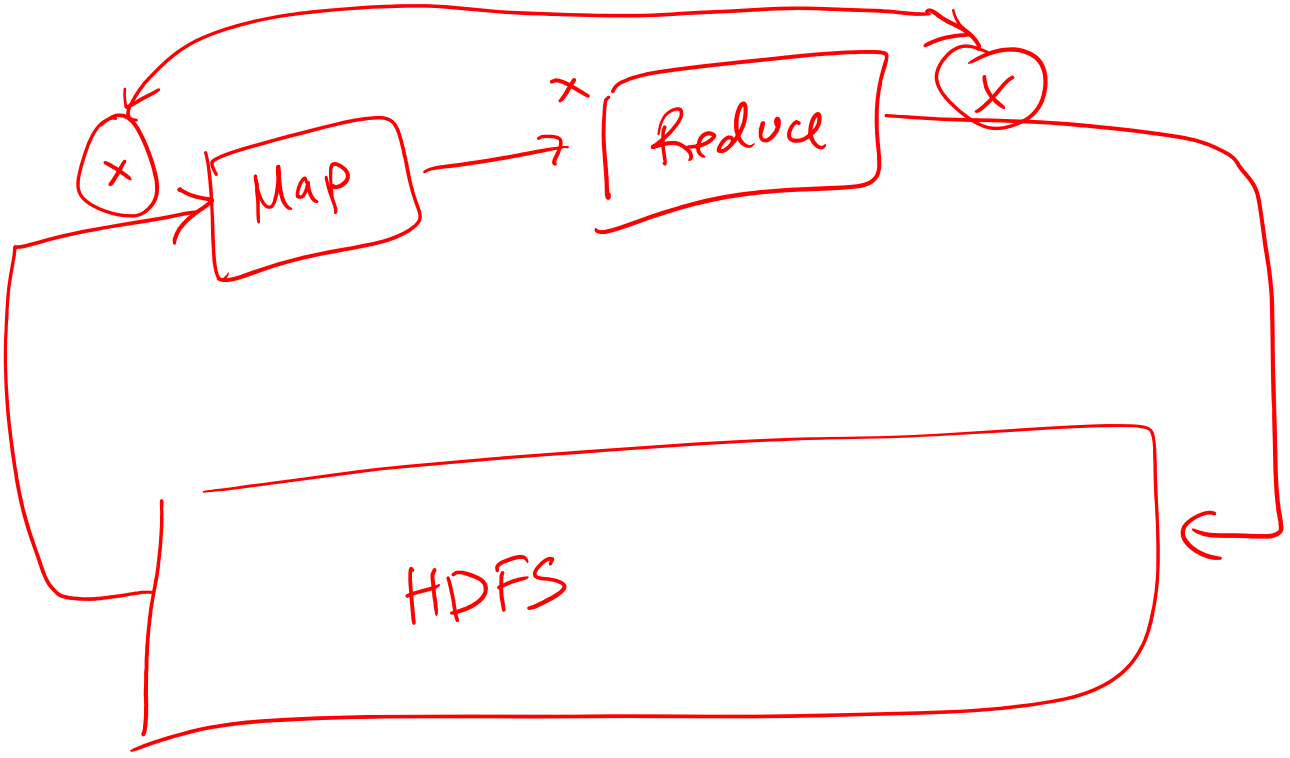
- Huffman coding/Huffman tree
  - Build a tree based on character frequency
- Difficult to compress
  - Random data (or compressed data)
- All compression algorithms exploit bias in data
  - Images have white patches
  - Letters E/T/A occur more frequently

Entropy. (Shannon 1948) $$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$

2021 Hadoop 1 → 2021 Hadoop 2

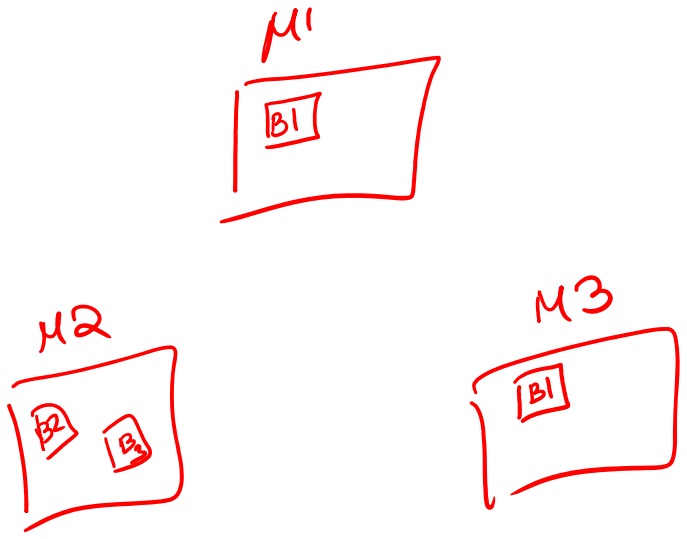
Compression in Hadoop

- Multiple (independent) opportunities
  - HDFS => Mapper => Reducer => HDFS



Assigning Data to Mappers

- HDFS-based
- Mapper executed locally for best performance
- Everything is managed in blocks
  - Replication
  - Balancing



Block Rebalancing

- How the data is distributed on HDFS nodes
- Expected performance change?
  - Load
  - Map-Runtime
  - Reduce-Runtime

Running the Rebalancer

- Runs in background
- Specify the threshold
  - hdfs balancer -threshold X
  - Percent of node usage
  - (default 10%)
- Configuration
  - dfs.balance.bandwidthPerSec limits the amount of data each node spends on rebalancing (1MB default)

Assigning Data to Reducers

- Partitioner
  - JobConf.setPartitionerClass()
- Uses Java hash function + MOD by default
- Sub any function (based on # of reducers)

Partitioning Function

- Function assigning Key-value pairs to Reducers
- Expected performance change?
  - Load
  - Map-Runtime
  - Reduce-Runtime
- Sorting

- No SQL / Not Only SQL
- Non-relational databases (data-store) systems
- Relax some of the relational database rules
  - Integrity constraints
  - Rigid schema (some)
  - Update/synchronization guarantees
  - Flexible
  - Highly-scalable
- Always a trade-off

Key Value Stores

- For a (key, value) combination
  - Key is a key
  - Value is a “blob”
  - Insert
  - Update
  - Delete
  - Get
- Many implementations
  - DynamoDB, Riak, Redis, BerkeleyDB, …

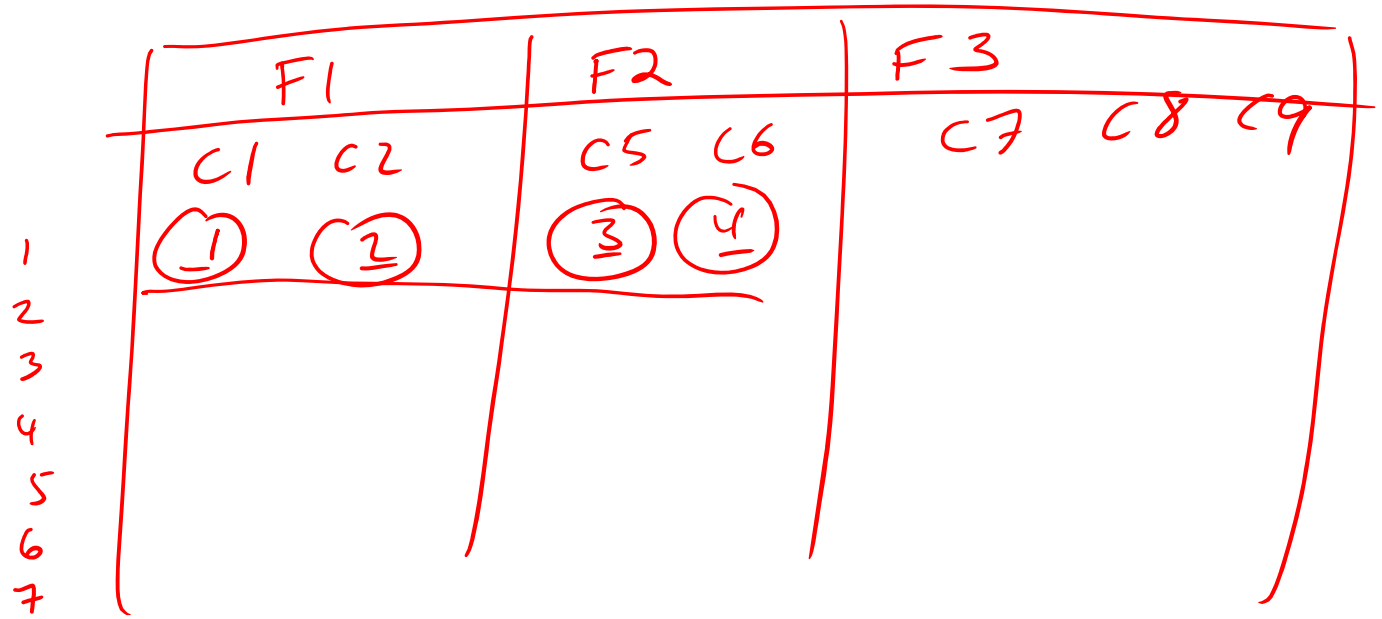
Document Stores

{ "Author" : "Rusty" }

- Similar to key value stores
- Value is not a blob, but a document, e.g.
  - {"Subject": "I like Plankton“
    "Author": "Rusty"
    "PostedDate": "5/23/2006"
    "Tags": ["plankton", "baseball", "decisions"]
    "Body": "I decided today that I don't like baseball. I like plankton." }
- You may have heard of
  - MongoDB, CouchDB, Terrastore, RavenDB

Column Family Stores

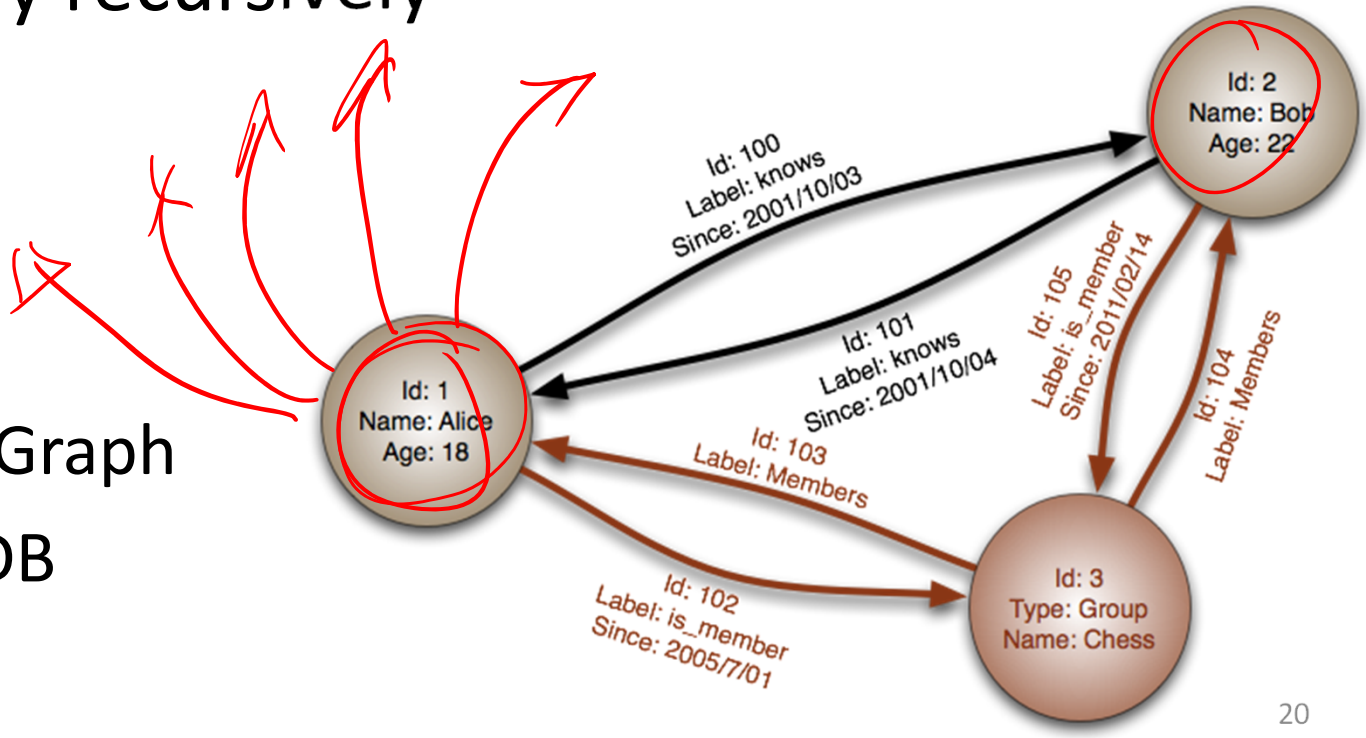
- Similar to document stores
- Add the concept of “column family”
- Helps optimize read access and updates



Graph Databases

- Database stores the set of items
- Items are connected to each other
- Can query recursively

- DBs
  - Neo4j
  - InfiniteGraph
  - OrientDB

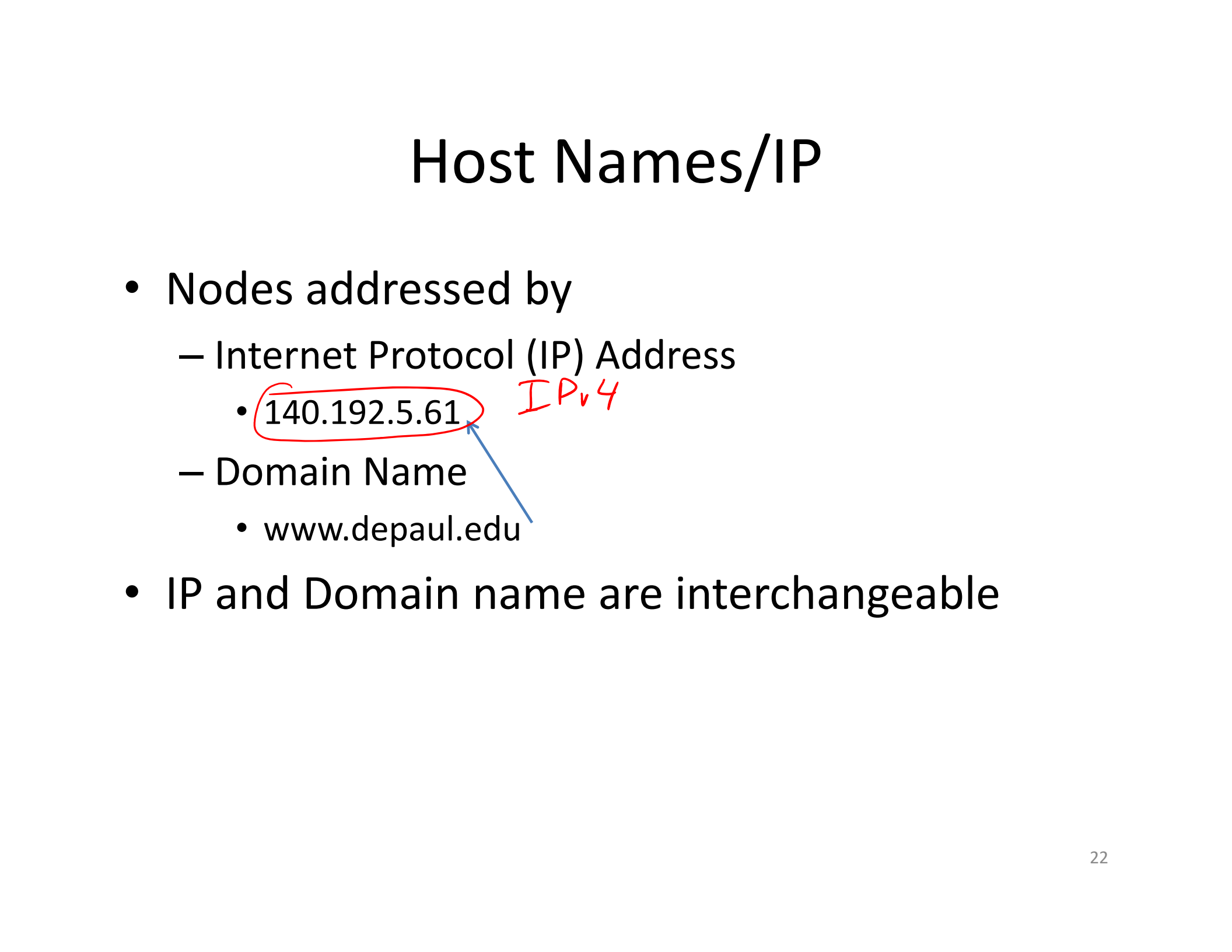


7:45

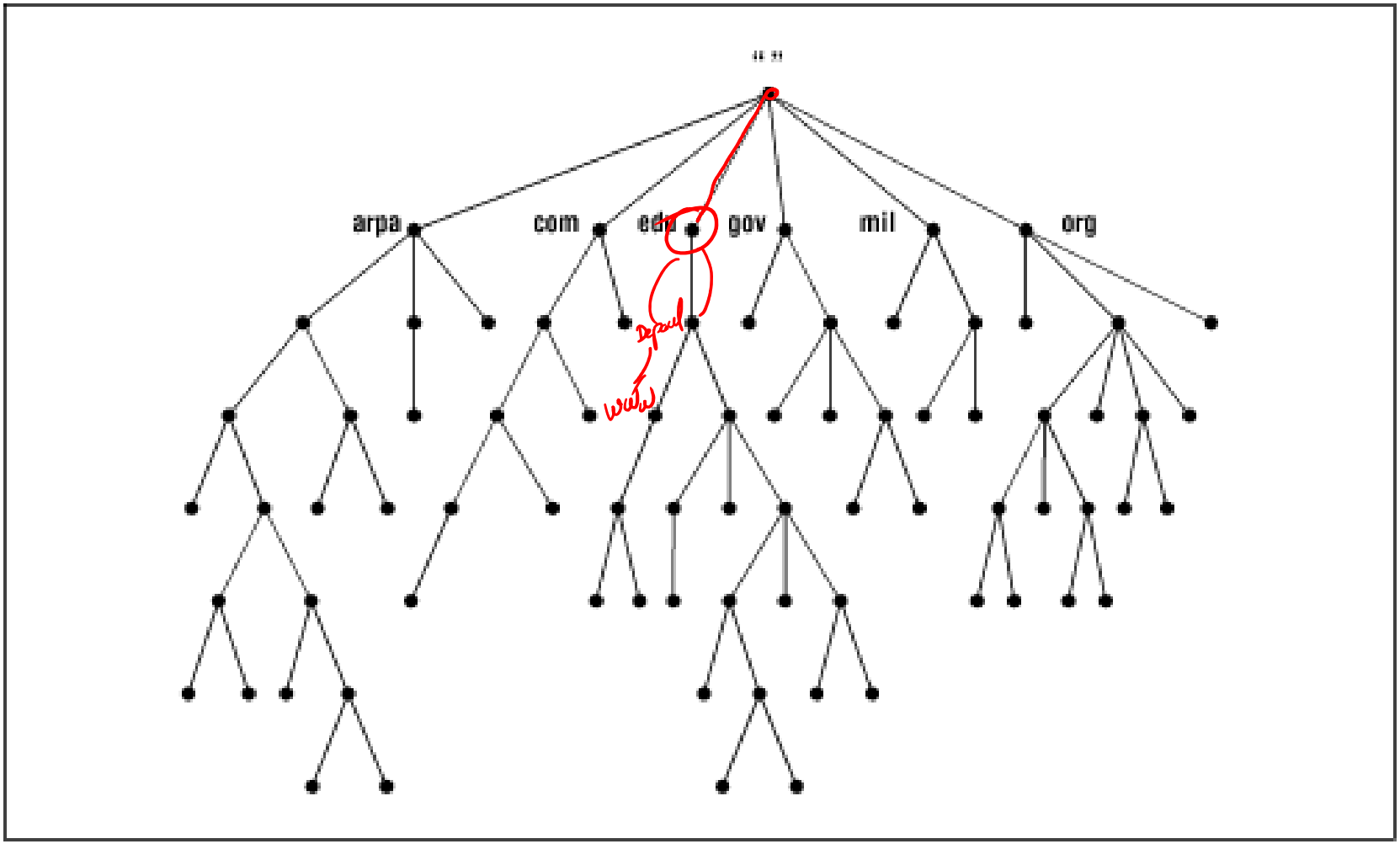
A Break

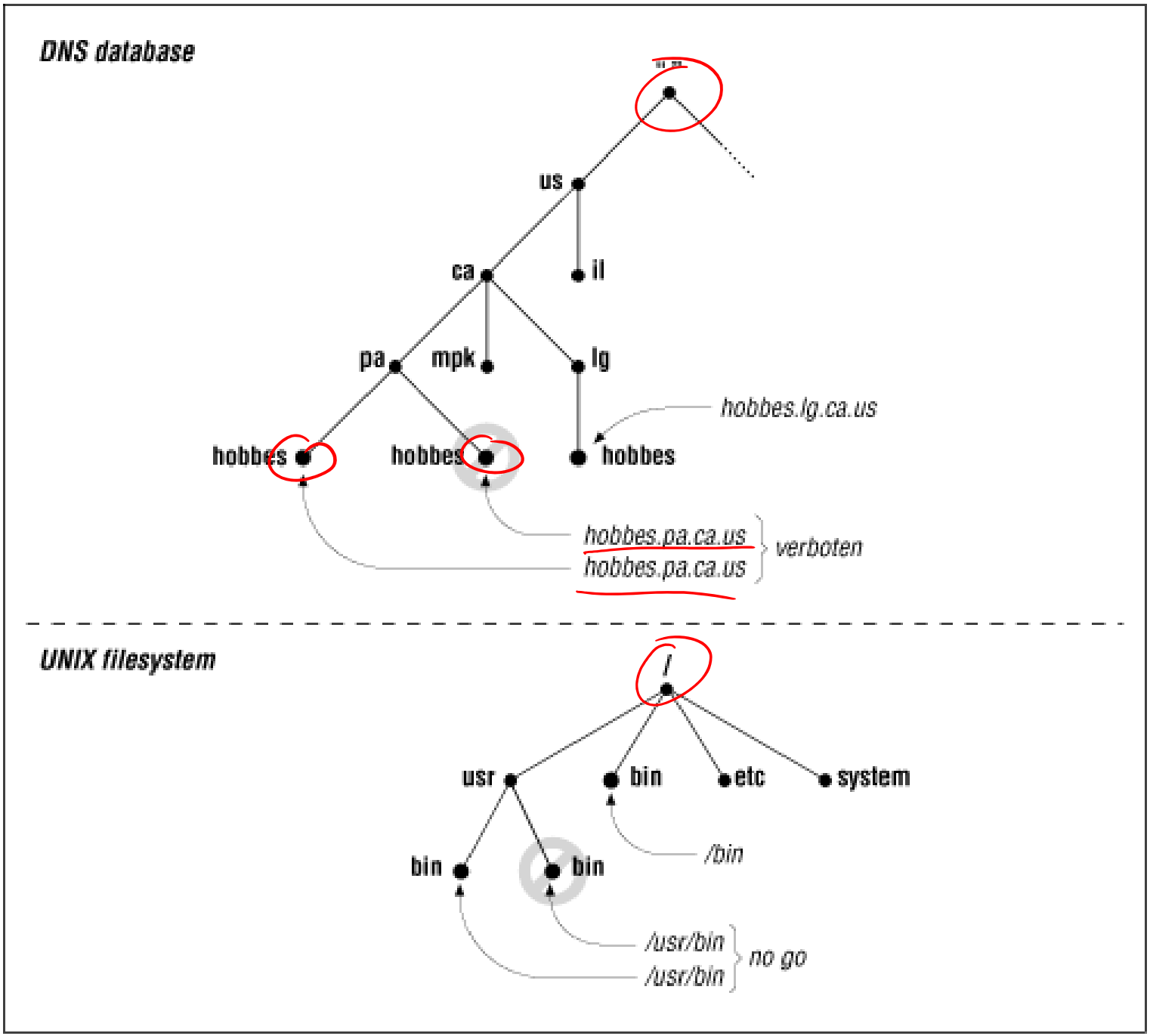
Host Names/IP

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



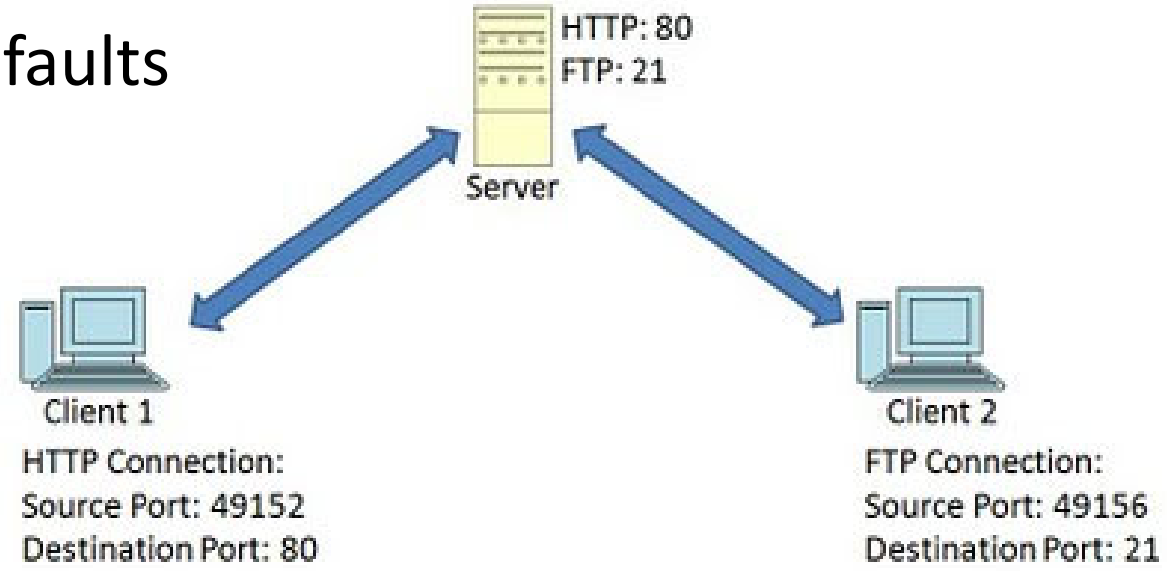
Domain Reverse Hierarchy




DNS database
us
ca
il
pa
mpk
lg
hobbes.lg.ca.us
hobbes
hobbes
hobbes
hobbes.pa.ca.us
hobbes.pa.ca.us
verboten
UNIX filesystem
/
usr
bin
etc
system
bin
bin
/bin
/usr/bin
/usr/bin
no go

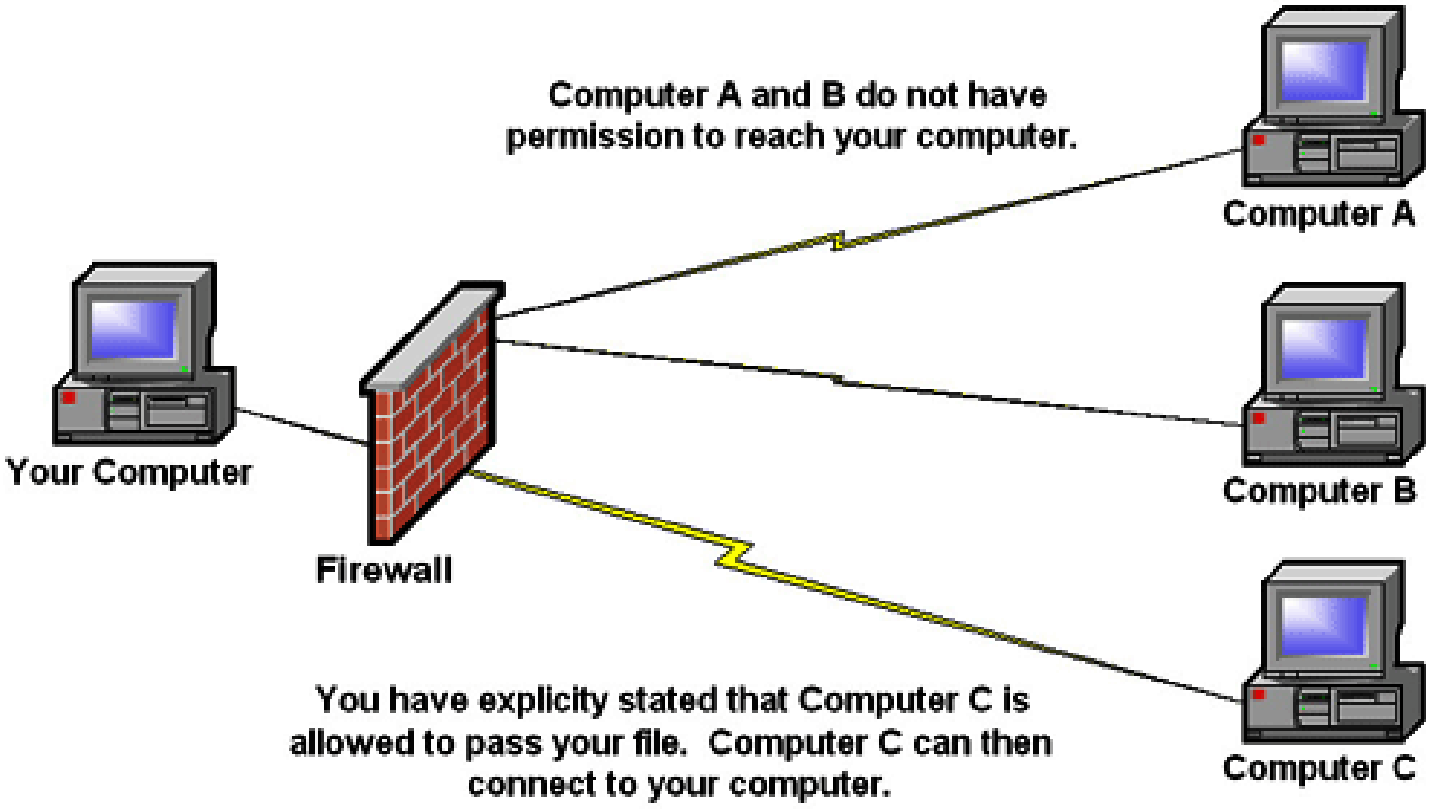
Machine Address + Port

- Address always includes a port
  - Sometimes not explicitly specified
  - Well known defaults
    - ssh = 22
    - www = 80
    - ftp = 21
    - mail = 25
  - Lesser known
    - Hadoop status report = 50070



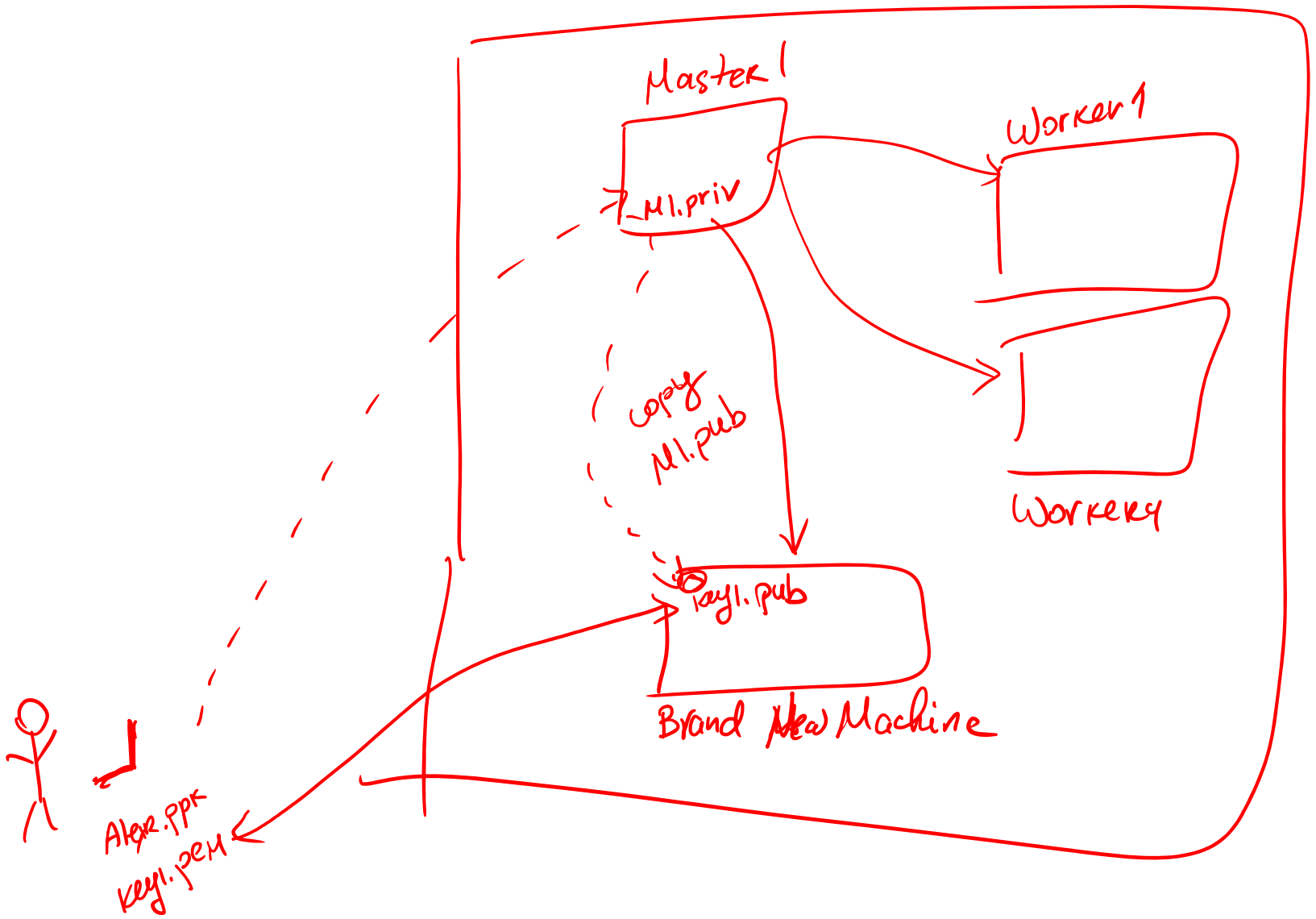
Firewall

- Controls the individual port access



Cluster Setup

- Passwordless login
  - Public/private key
  - Localhost to Localhost
  - Master to Worker
- Firewall / Security groups
- Hadoop logs


Master 1
Worker 1
M1.priv
Worker 2
copy
M1.pub
key1.pub
Brand New Machine
Alex.ppk
key1.pem

Hadoop Streaming: A Join!

```
SELECT SUBSTRING(tv.Plate, 0, 3), COUNT(*)
FROM TowedVehicles AS tv,
        RelocatedVehicles AS rv
WHERE
SUBSTRING(tv.Plate, 0, 3) = SUBSTRING(rv.Plate, 0, 3)
GROUP BY SUBSTRING(tv.Plate, 0, 3)
```

Hive Transform

CREATE TABLE u_data (userid INT, movieid INT, rating INT, unixtime STRING) ROW FORMAT DELIMITED FIELDS

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

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

OVERWRITE INTO TABLE u_data;

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;

Next Time:

- More on Hadoop Ecosystem
- Mahout
- Link analysis