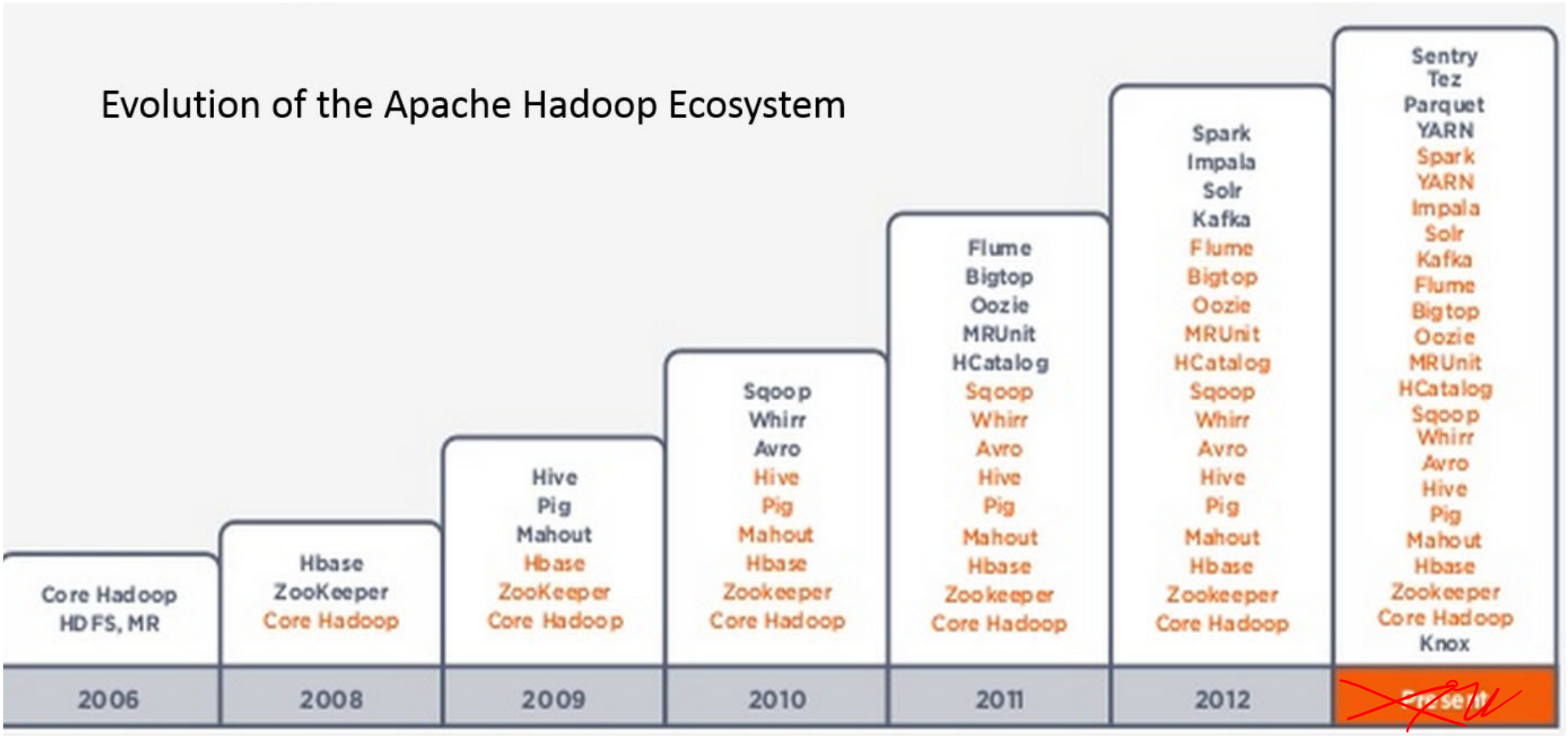
CSC 555 and DSC 333
Mining Big Data
Lecture 7

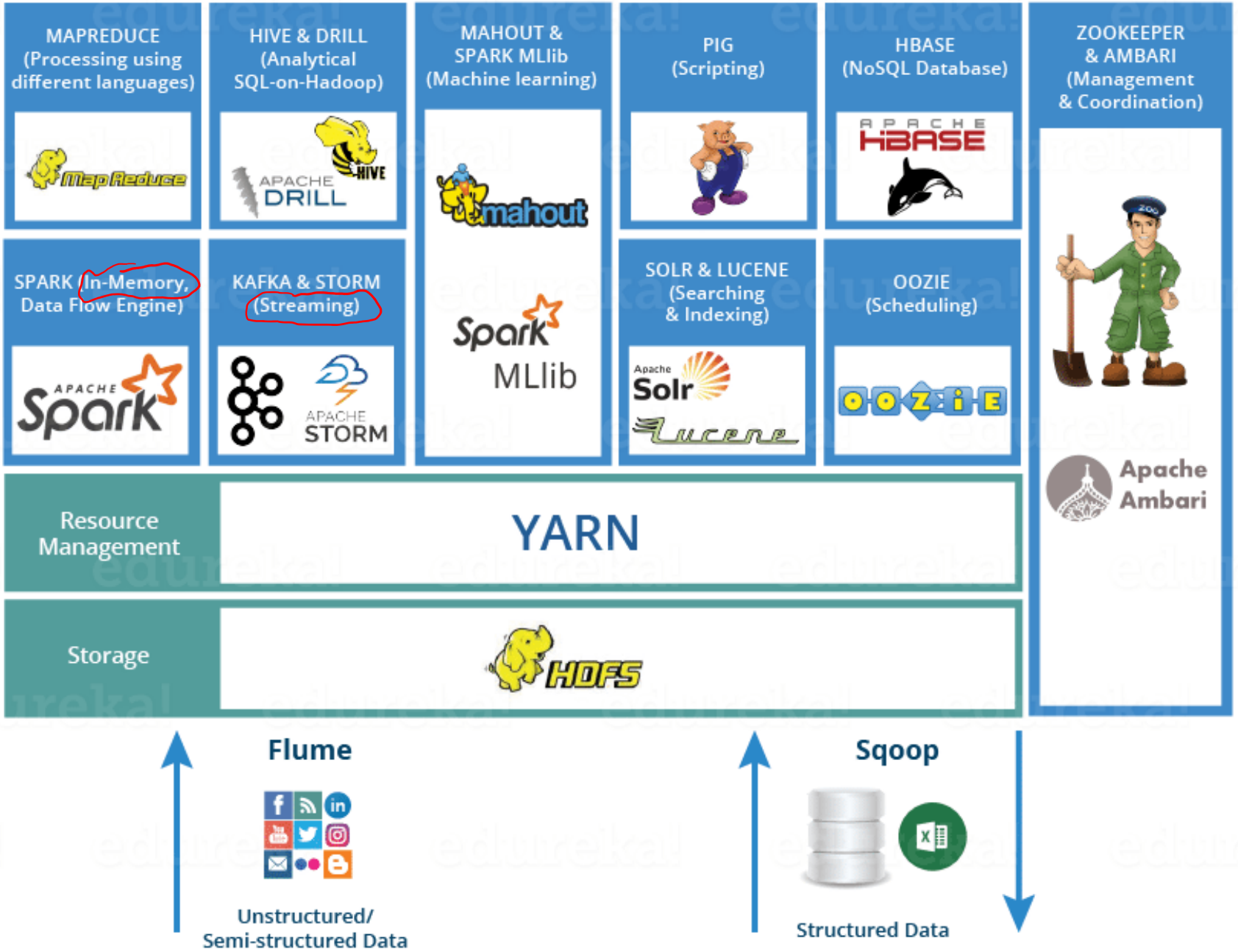
Alexander Rasin
College of CDM, DePaul University
October 26th, 2021

Tonight

- Hadoop ecosystem
- Cluster setup
- Hive Transform
- Link analysis

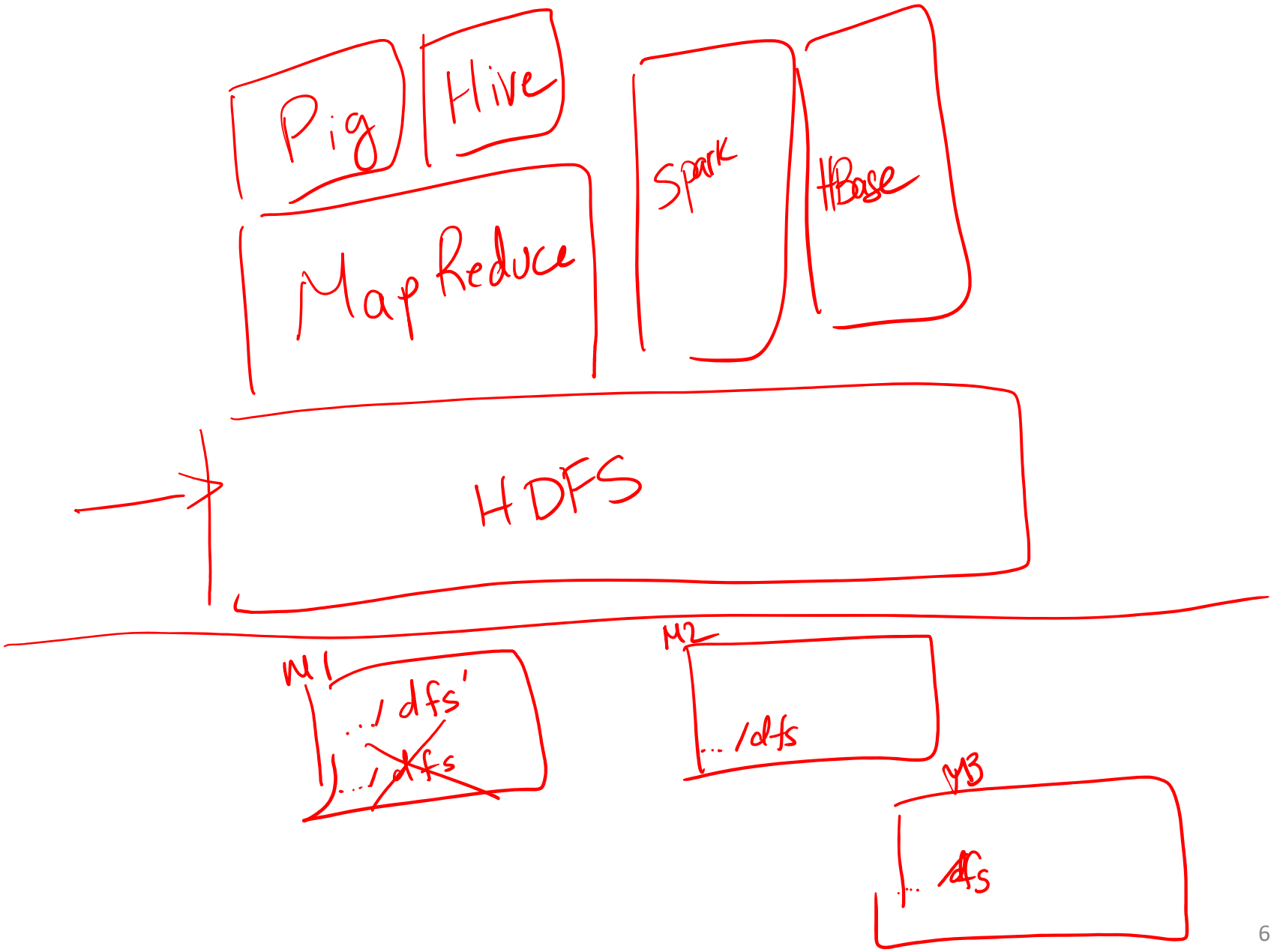
Apache Hadoop Ecosystem




MAPREDUCE
(Processing using different languages)
HIVE & DRILL
(Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib
(Machine learning)
PIG
(Scripting)
HBASE
(NoSQL Database)
ZOOKEEPER & AMBARI
(Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM
(Streaming)
SOLR & LUCENE
(Searching & Indexing)
OOZIE
(Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/
Semi-structured Data
Sqoop
Structured Data

Cluster Reformatting

- Stop Hadoop
  - Run stop scripts (if stop fails, killall -9 java)
- Delete the old DFS storage **on every node**
  - **rm -rf /tmp/hadoop-ec2-user/dfs/**
    - (For master and every worker)
- Reformat HDFS
  - hdfs namenode -format
    - (on master)


Pig
Hive
Map Reduce
Spark
HBase
HDFS
M1
.../dfs'
.../dfs
M2
.../dfs
M3
.../dfs

HDFS Shell Commands

hadoop distcp <hdfs_src> <hdfs_dst>

-i ignore failure

-log <logdir> (logging)

-m <num_maps> (max simultaneous copies)

-update (overwrite if src size different from dst)

hdfs dfsadmin [-safemode enter / leave /get / wait]

hdfs fsck <dir>

hadoop fs -du /

hdfs dfsadmin -report

Timing Pig Commands

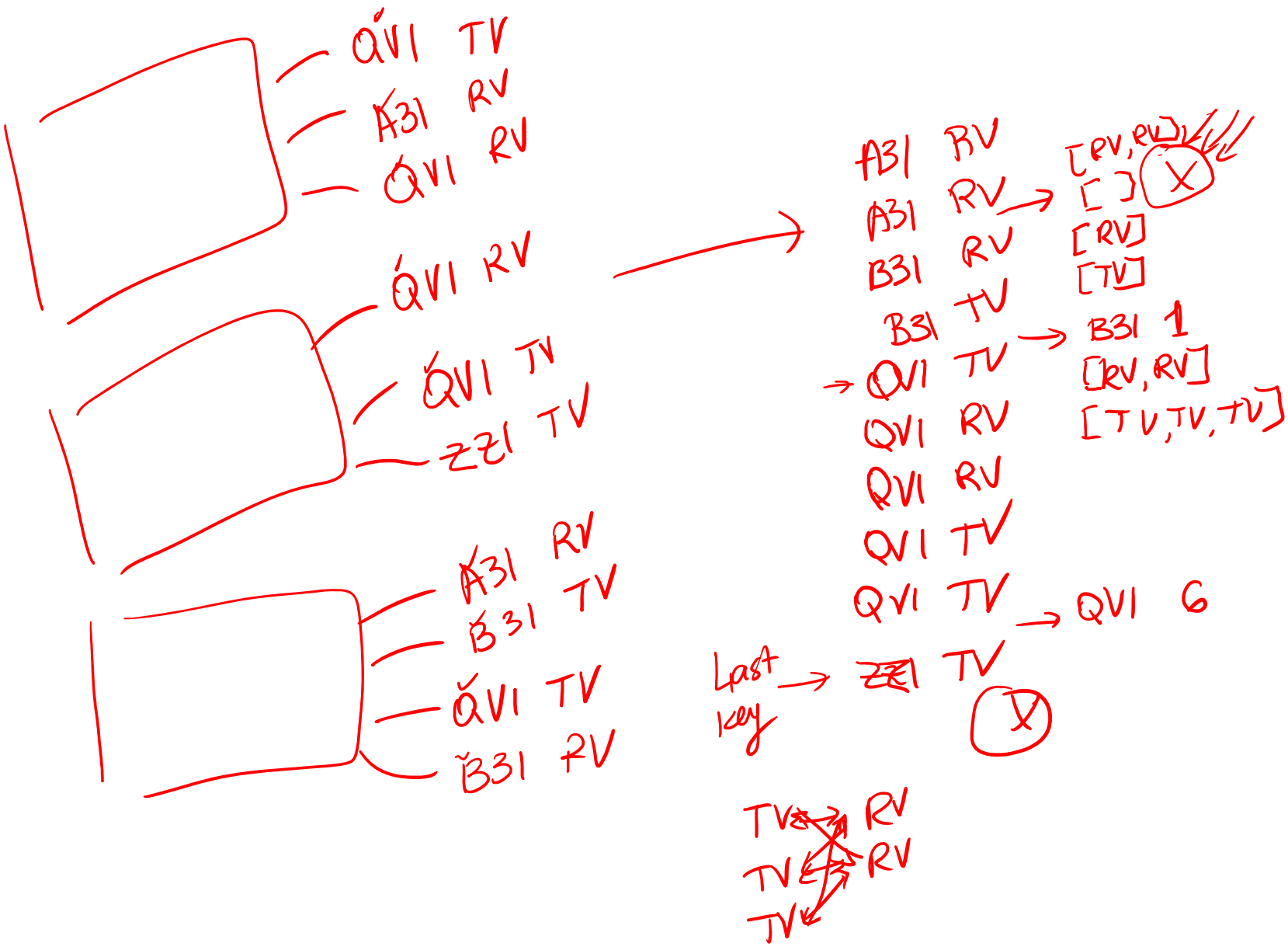
```
UData = LOAD 'u.data' USING PigStorage('\t') AS
(userid:int, movieid:int, rating:int,
unixtime:chararray);

GoodRatings = FILTER UData BY rating > 2;

UserSet = GROUP GoodRatings BY userid;

UserRatings = FOREACH UserSet GENERATE
COUNT(GoodRatings);
```

Chaining Hadoop Jobs

- hadoop jar hadoop-streaming-2.6.4.jar -D mapred.reduce.tasks=3 -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator -D mapred.text.key.comparator.options=-nr -input /user/ec2-user/u.data -output /data/output3 -mapper /bin/cat -reducer myReducer.py -file myReducer.py

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)
```


QV1 TV
A31 RV
QV1 RV
QV1 RV
QV1 TV
ZZ1 TV
A31 RV
B31 TV
QV1 TV
B31 RV
A31 RV
A31 RV
B31 RV
B31 TV
QV1 TV
QV1 RV
QV1 RV
QV1 TV
QV1 TV
ZZ1 TV
[RV,RV]
[ ]
X
[RV]
[TV]
B31 1
[RV,RV]
[TV,TV,TV]
QV1 6
Last key
TV
TV
TV
RV
RV

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, rating, unixtime) USING 'python weekday_mapper.py'

AS (userid, weekday) FROM u_data;

Link Analysis

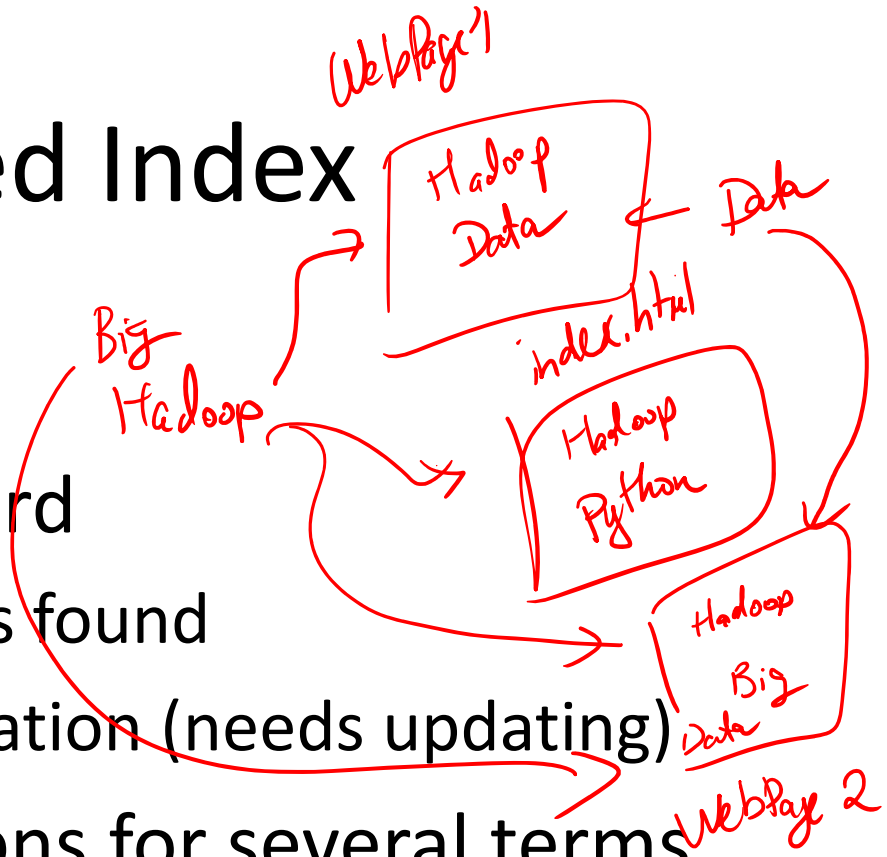
- Web search
- Link spamming
  - Search engine optimization (SEO)

Early Web Searches

- Crawl the web
- Find the search *terms*
  - Scan collected web-page snapshots
  - Inverted index
- Weight the terms accordingly
  - Header = more important
  - Frequently used = more important

Inverted Index

- Search term location
- For popular term record
  - Locations where it was found
  - One-time pre-computation (needs updating)
- Intersect these locations for several terms
- Rank the selected results



Gaming the System

- Term spamming
  - Add Hadoop to header
  - Add Hadoop 1000 times
  - Hide words in background
- Add related terms
  - Search for Hadoop
  - Copy the pages that come up first

The Google Solution

- Find the “important” pages
  - Random walk the web
    - Start anywhere
    - Follow random links
    - See where you end up
  - Cannot count the # of links instead…
- Do not trust the (destination) page
  - Rely on terms in the linking page
  - Source page outside of destination control

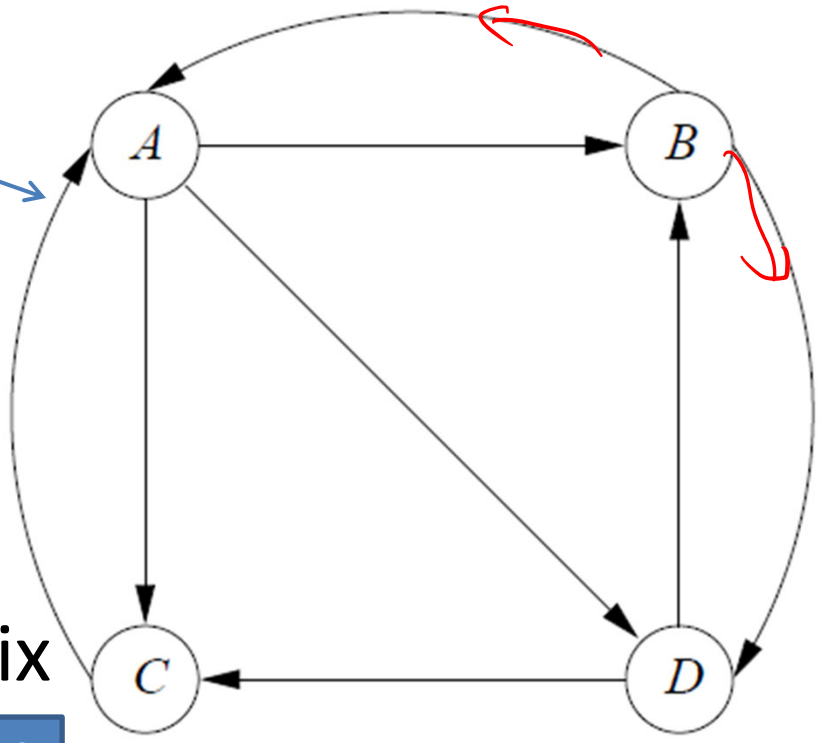
Page Rank

- Ignore the terms on the page
- Only trust others to define your page
- Ignore link farms
  - Link farms are a closed system

- Web traffic indicative of page popularity
- Links indicative of page relevance

Definiton of PageRank

- Imagine this is the web
  - Assume random walk
  - From A -> 3 places
  - From B -> 2 places
  - …
- Define a transition matrix



$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

A B C D

A B C D

Surfer Probability Distribution

- Randomly placed surfer [1/n, 1/n, … 1/n]
  - [¼, ¼, ¼, ¼] for our example
- To compute probability distribution
  - Multiply the transition matrix by vector

$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} \rightarrow \begin{bmatrix} 3/8 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

$0 + \frac{1}{8} + \frac{1}{4} + 0$

Probability Distribution

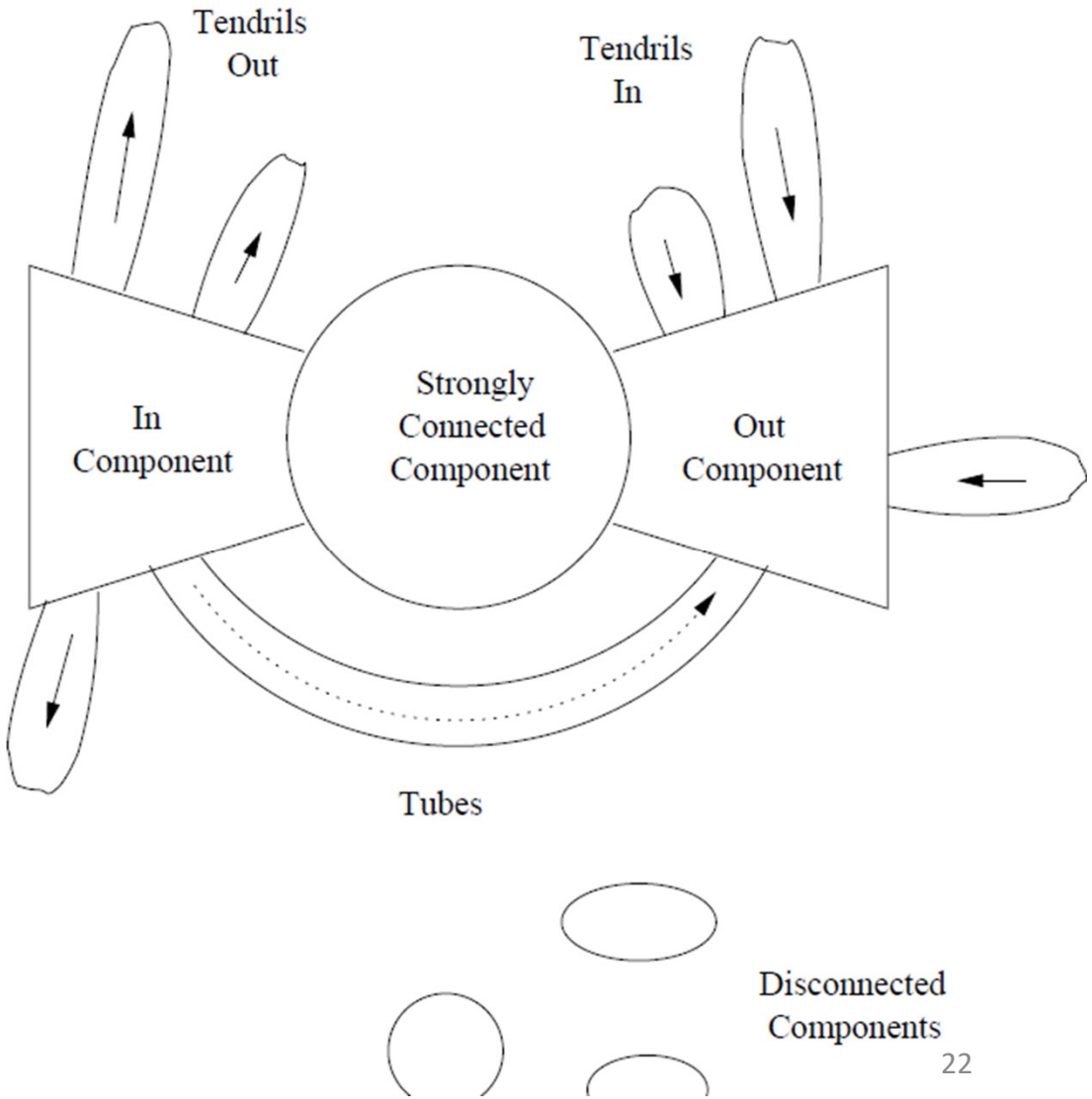
- Probability distribution after 2 steps
  - $M * M * v = M^2 * v$
- Eventually converge
  - Strongly connected graph
  - No dead ends
- In practice – 50 to 75 iterations
- In reality starting probability not equal

$M * V_x = V_x$

Structure of the Web

- InC => SCC/OutC
- SCC => OutC
- Dead ends
  - 0% to leave

- Spider traps



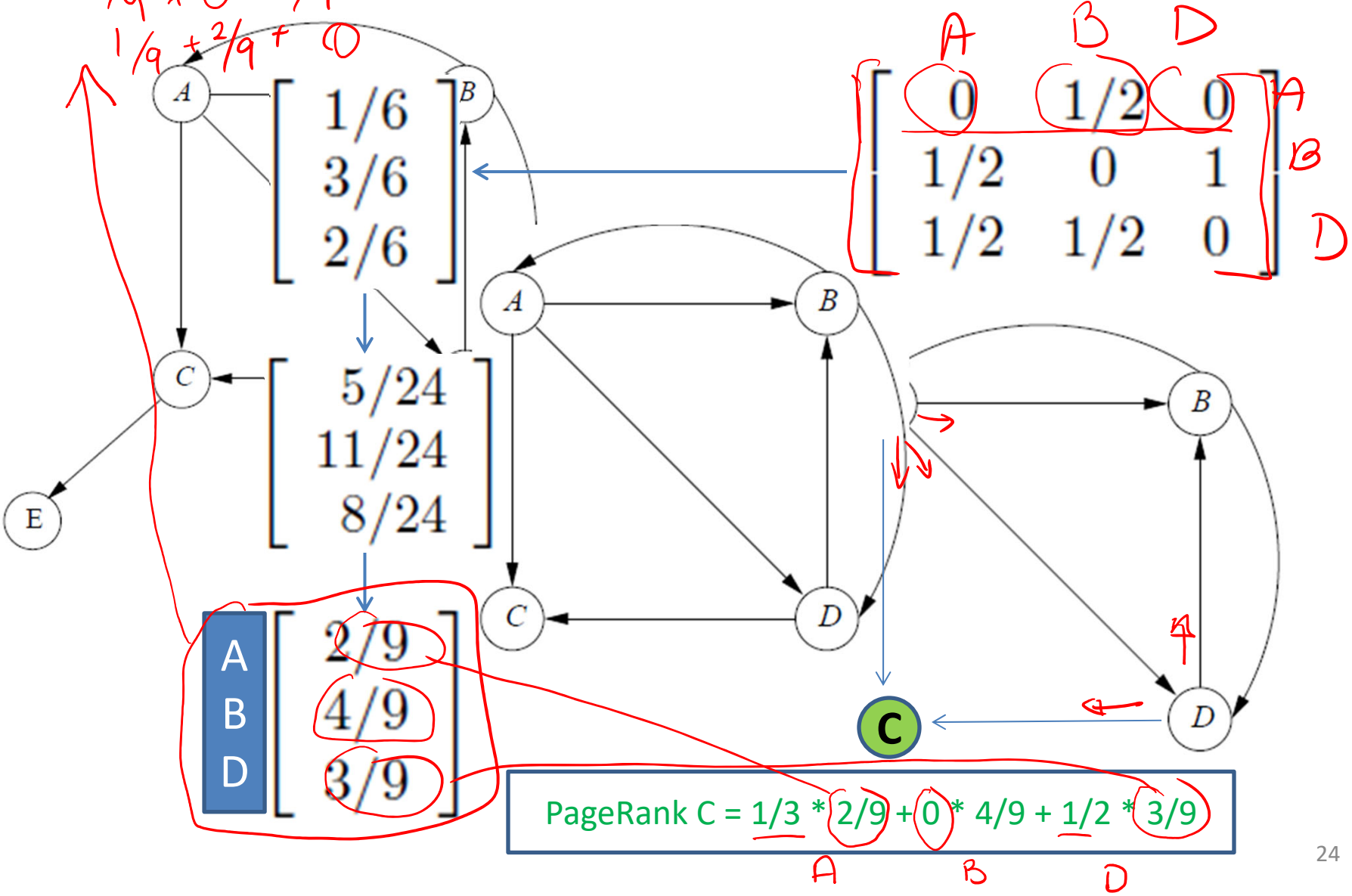
Avoiding Dead Ends

- Node C = dead end

$$\begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

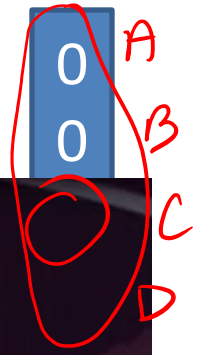
- Matrix multiplication
  - Converges to [0, 0, 0, 0]
  - C linked to itself? (spider trap)
- Solution?
  - Drop dead end nodes

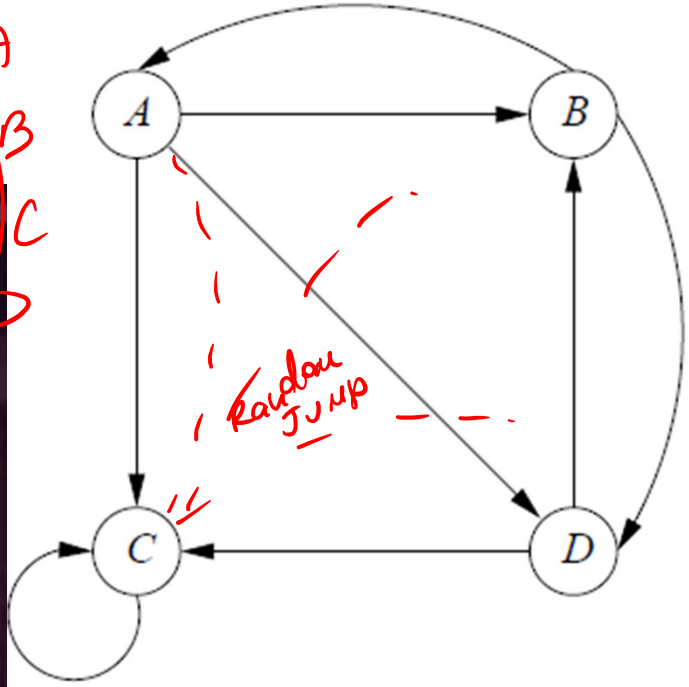
Dropping Dead Ends

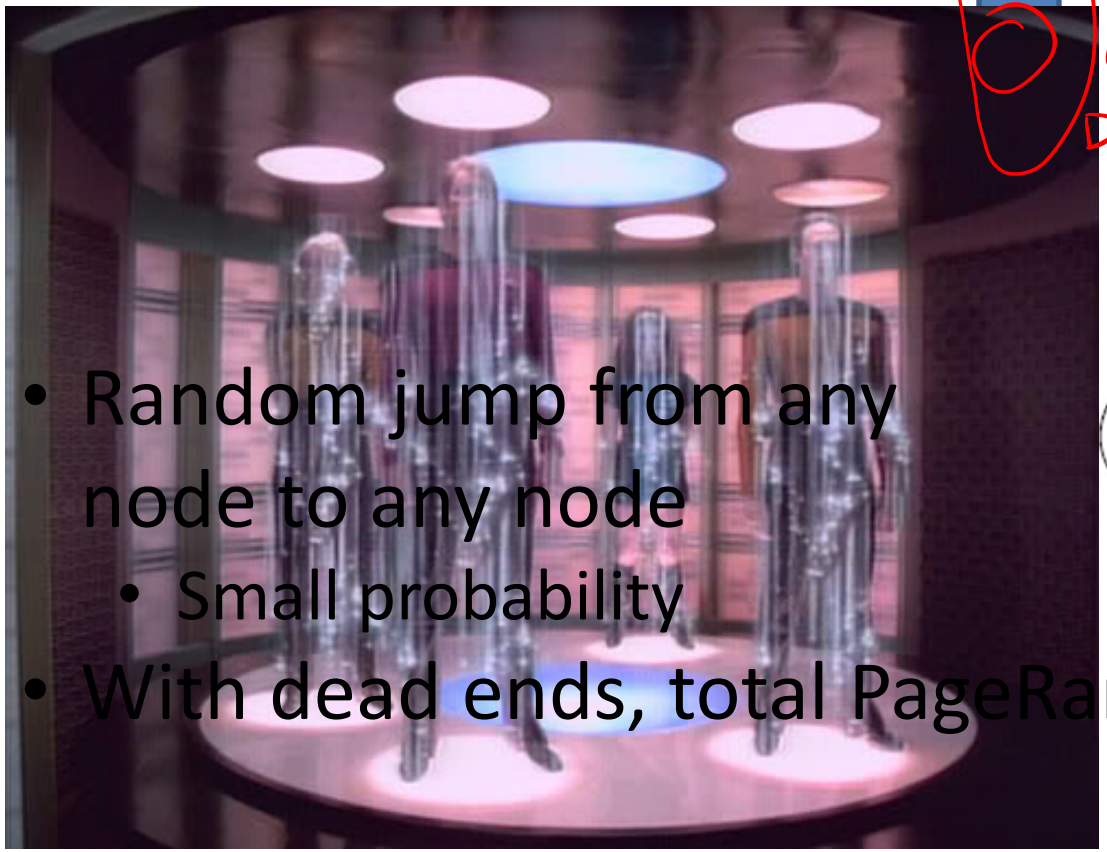


Spider Traps

- PageRank converges to





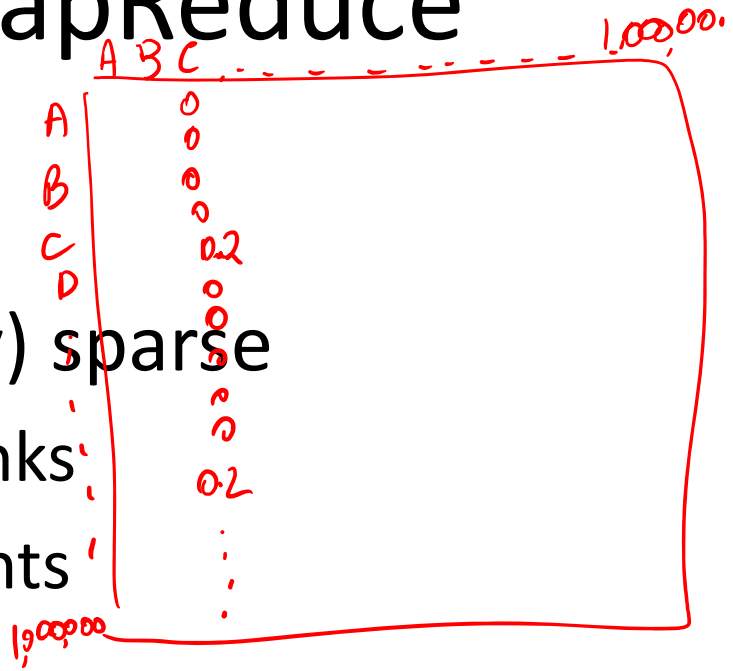


- Random jump from any node to any node
  - Small probability
- With dead ends, total PageRank < 1

PageRank Computation

- Google's secret formula
- Select pages with search terms
- Apply PageRank
  - Filter out unrelated/spam results
  - Apply your preferences (Google+, gmail)
  - Weight by topic?
- PageRank is just one of the components

PageRank and MapReduce



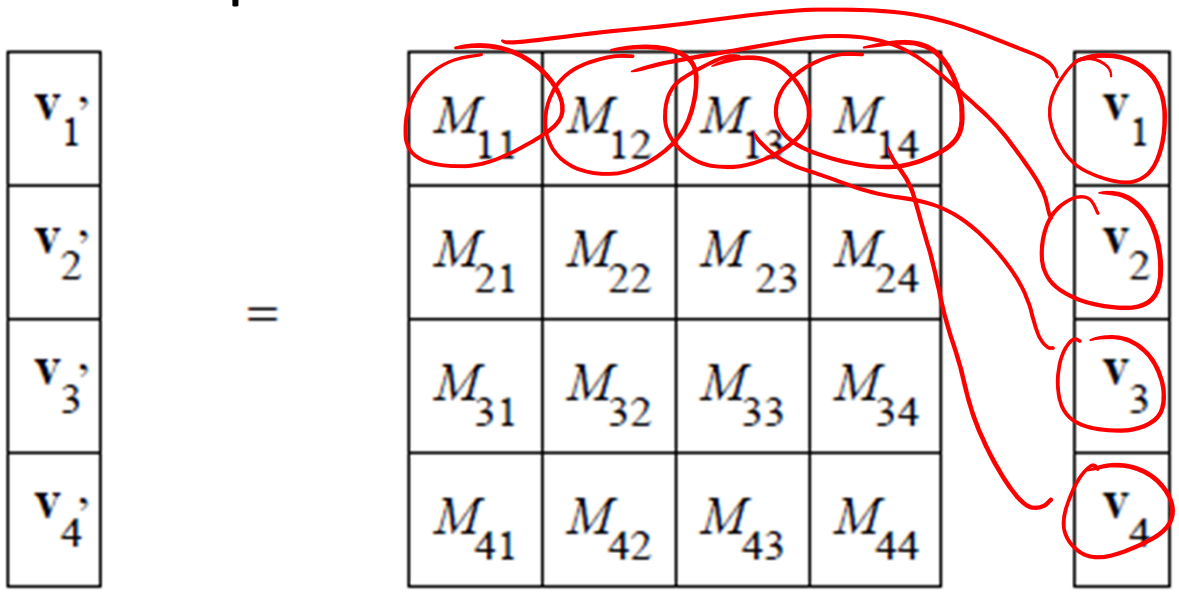
- Matrix multiplication
- Transition matrix is (usually) sparse
  - Average page has ~10 out links
  - Operate on non-zero elements
- Graph representation

$$\begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

Source	Degree	Destinations
A	3	B, C, D
B	2	A, D
C	1	A
D	2	B, C

PageRank and MapReduce

- Stripe the vector ***v*** (to fit in memory)
- Partition the matrix into blocks
  - k^2 map tasks
  - Vector stripes sent across the network



Topic-Sensitive PageRank

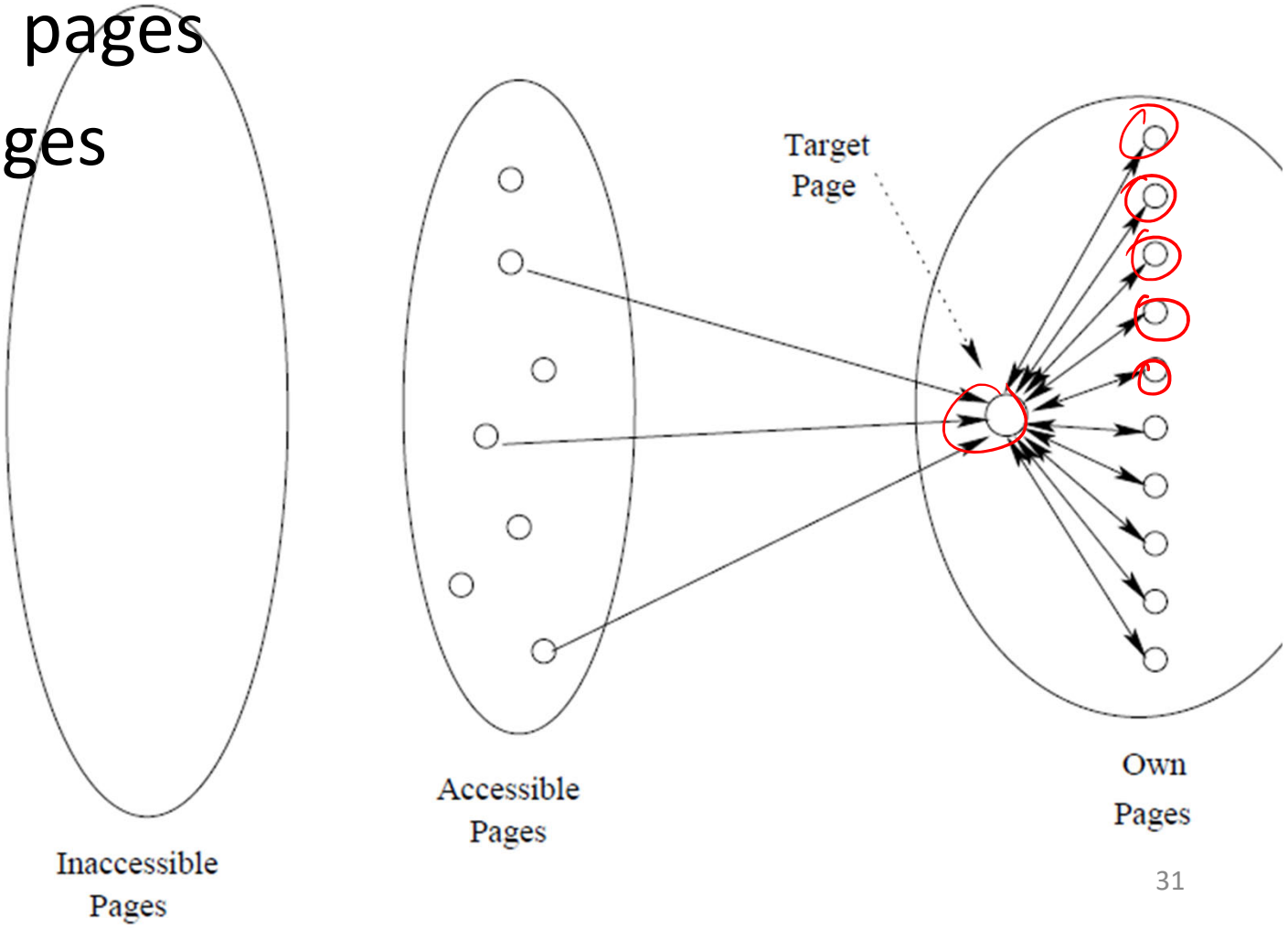
- Different user interest for pages
  - Ideally, every user has individualized PageRank
- Topic-sensitive PageRank
  - Biased starting rank by subject
  - E.g. by category (DMOZ)
- Bias towards particular category
  - Add a teleport factor
  - Topic-specific pages (mostly) lead to related pages

Topic-Sensitive PageRank

- Choose which topics to bias for
- Select pages that match that topic (+teleport)
- Determine topic(s) relevant to the query
  - Determining context – very difficult
  - Manual selection/key words/user info
- Using keywords
  - Indicative terms (words specific to topic)
  - Classify pages by computing Jaccard similarity between the page and the topic word set

LinkSpam

- Inaccessible pages
- Accessible pages
- Owned pages



Dealing with LinkFarms

- TrustRank
  - Topic-sensitive set of pages that is “trustworthy”
  - Cannot be spam-able (blogs, etc.)
  - Human selected
    - Pages
    - Domains
- Spam Mass
  - (PageRank – TrustRank)/PageRank
  - Negative/small values => likely ok