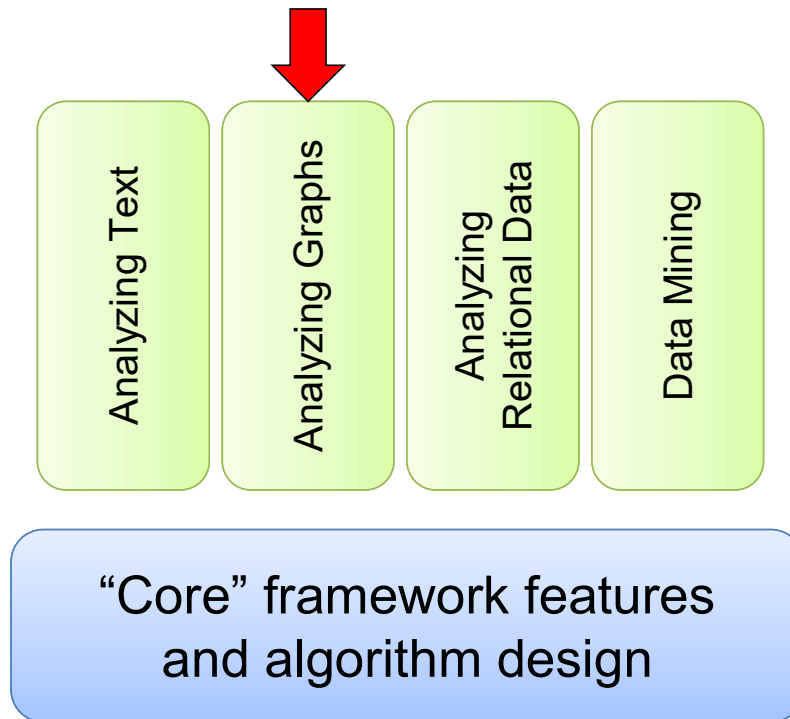


Data-Intensive  
Distributed  
Computing  
CS431/451/631/651

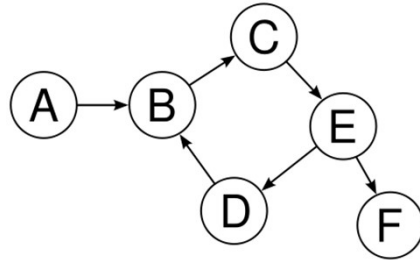
Module 5 – Analysing  
Graphs



# Structure of the Course



# What's a Graph



- $G = (V, E)$  (set of vertices and set of edges)
- Direction:
  - Undirected – edge  $(u, v)$  implies edge  $(v, u)$
  - Directed – edge  $(u, v)$  does not imply edge  $(v, u)$
- Edges may or may not be labelled
  - (Numerical labels are usually called “weights”)
- This should not be news to anybody!

# Vocab Reminders

- Vertex (or “node”) – The circle thingies
- Edge (or “link”) – connects one Vertex to another
  - (Or to itself, perhaps)
- If there is edge  $(u, v)$  then:
  - $v$  is an “out-neighbour” of  $u$
  - $u$  is an “in-neighbour” of  $v$
- In-Degree( $v$ ) – how many edges lead into  $v$
- Out-Degree( $v$ ) – how many edges lead out from  $v$

# Example Graphs

- Roadways, Water mains, power lines, other SimCity things.
- Social Networks (a person is a vertex, edges are “friends”)
- Actual computer networks
- The internet (just a big network, innit?)

Most graphs are sparse: number of edges is closer to  $|V|$  than it is to  $|V|^2$



# Representation

In the past you've probably seen three representations

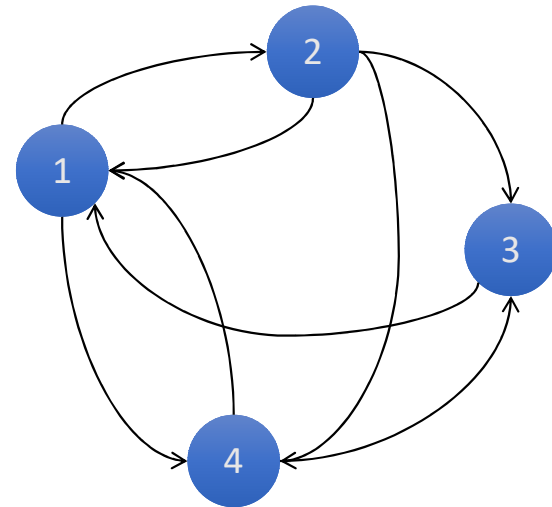
- Adjacency Matrix
- Adjacency List
- Edge List

# Adjacency Matrix

An  $n \times n$  matrix  $M$ .

$M_{ij} = 1$  iff there's an edge between  $v_i$  and  $v_j$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>1</b>	0	1	0	1
<b>2</b>	1	0	1	1
<b>3</b>	1	0	0	0
<b>4</b>	1	0	1	0



# Adjacency Matrix

## PRO:

- Who doesn't love a matrix?
- Useful for mathematical operations
- CUDA loves matrix operations
- Fast neighbourhood check (both in and out)

## CON:


- Mostly 0s in a sparse graph (wasted space)
- $O(n)$  to find neighbourhood of  $v$  (in and out)



# Adjacency List

Row  $i$  is a list of the out-neighbours of vertex  $v_i$

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	
<b>1</b>	0	1	0	1	1: 2, 4
<b>2</b>	1	0	1	1	2: 1, 3, 4
<b>3</b>	1	0	0	0	3: 1
<b>4</b>	1	0	1	0	4: 1, 3



# Adjacency List

## PRO

- Smaller than the matrix (especially for small graphs)
- Even easier to find out-neighbours of  $v$  (directly stored)

## CON

- Difficult to find in-neighbours of  $v$ 
  - have to search all other adjacency lists

# Edge List

- Just a list of all the edges

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>1</b>	0	1	0	1
<b>2</b>	1	0	1	1
<b>3</b>	1	0	0	0
<b>4</b>	1	0	1	0



(1, 2)  
(1, 4)  
(2, 1)  
(2, 3)  
(2, 4)  
(3, 1)  
(4, 1)  
(4, 3)

# Edge List

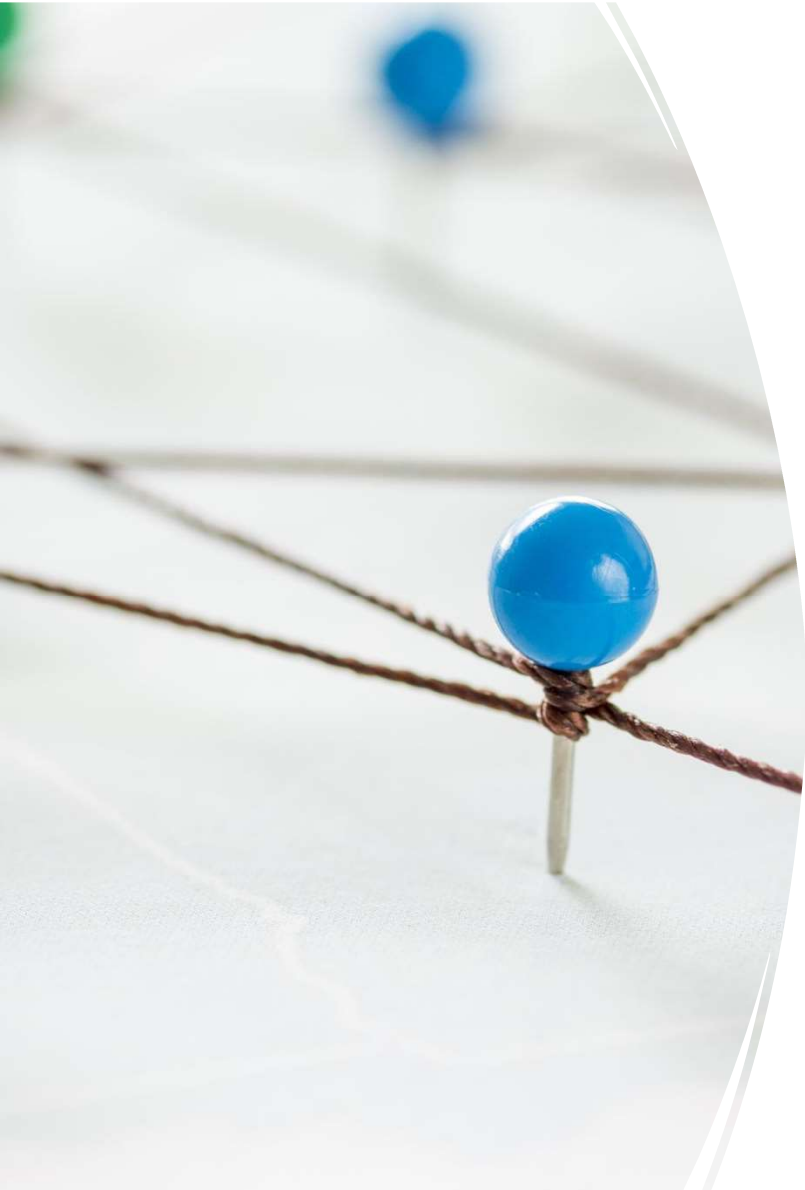
## PRO

- Easy to add an edge (just append)
- Simple

## CON

- Hard to find neighbours
- Hard to find anything, really





# Graph Problems

- Shortest Path – Google Maps, Delivery Planning
- Minimum Spanning Tree – Utility Lines
- Min-Cut – Utility Lines, Disease Spread
- Max-Flow – Airline scheduling
- Graph Colouring – Planning Final Exams
- Bi-Partite Matching – Dating Sites
- PageRank – Google

# What does the web look like?

- 4.77 billion pages – 50 billion (vertices)
  - It changes by a lot every day
- 100 billion – 1 trillion links (directed edges)

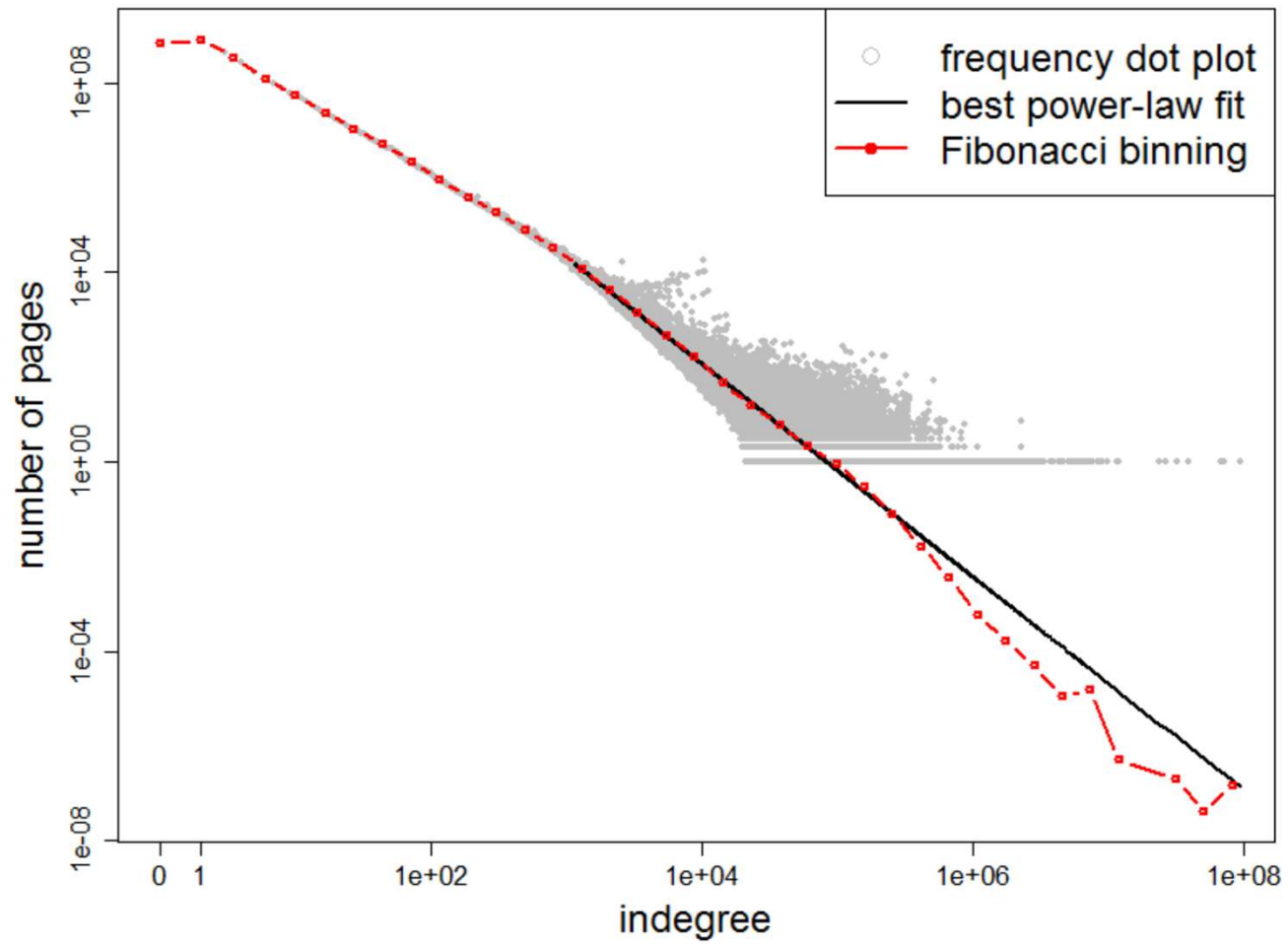
Sources: worldwidewebsite.com, Tilburg University Research

That's big

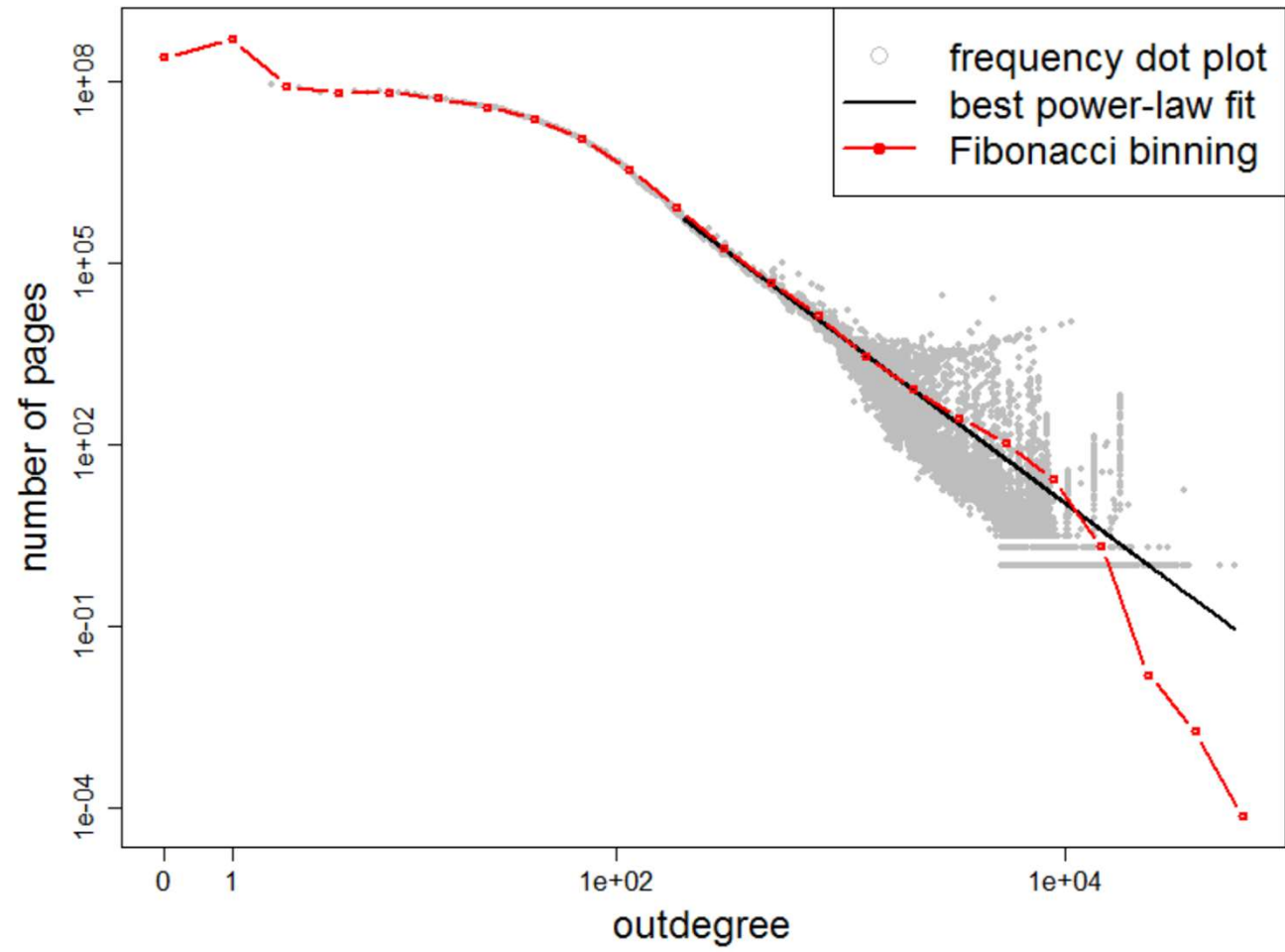
Power laws again:

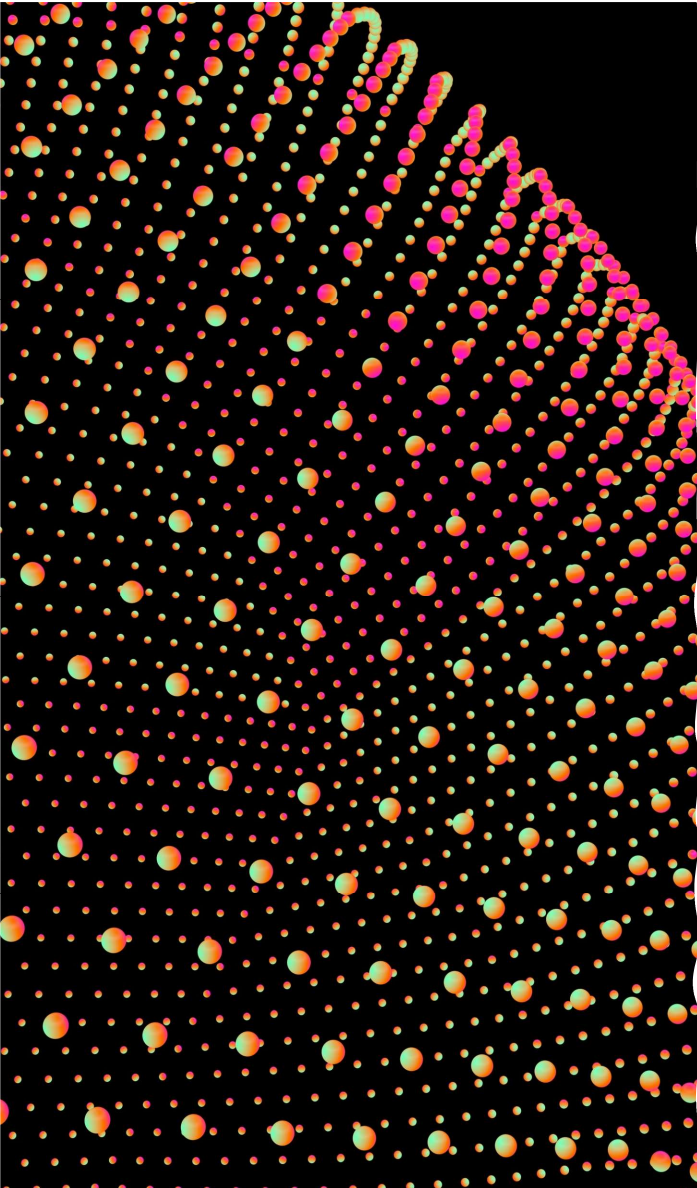
$$P(k) \sim k^{-\gamma}$$

Fraction of pages that have  $k$  links:









So when you say big,  
you mean...

60+ GB

The web graph is Big Data™

Clearly, we need Hadoop!

How do we do this on MapReduce (or Spark)?



# Graphs and Clusters

Many graph algorithms involve:

- Local Computations for each node
- Propagating these results to neighbours (graph traversal)

Questions:

1. Which representation fits the best?
2. How do the local computations work?
3. How does the traversal work?

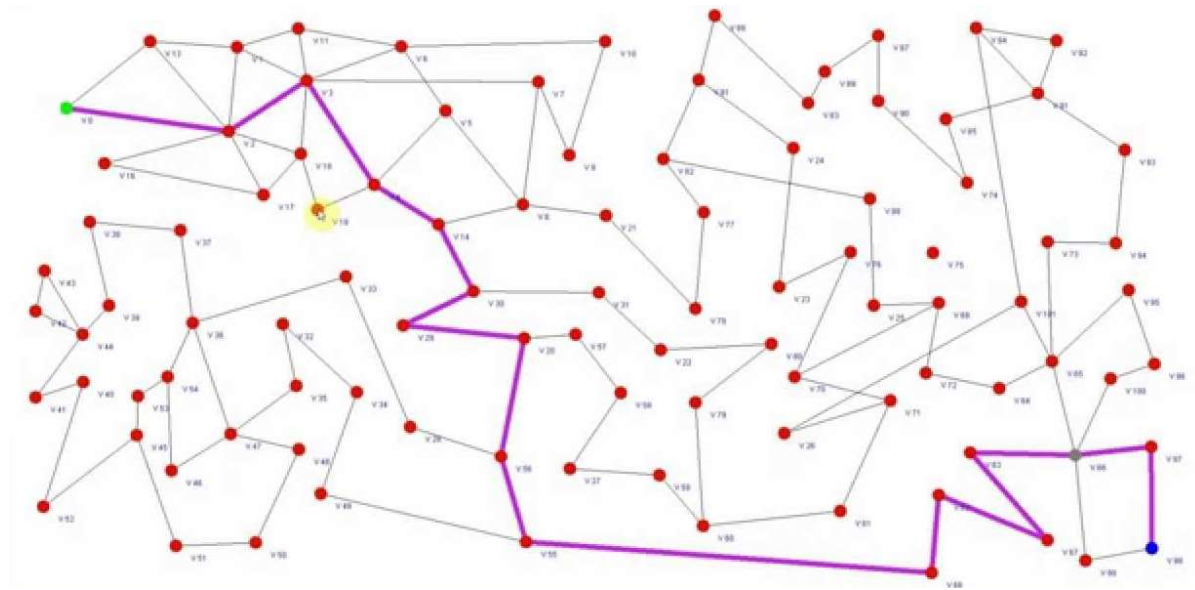
## Answer Key

- (Dan hopes there was some discussion there)
- Local computation means **INDEPENDENT**
  - Sounds very much like a map-like task
- Adjacency List makes the most sense
  - KVP – key is a vertex, value is its adjacency list
- Propagation
  - Shuffle – Collecting the propagated messages is reduce-like

# Single-Source Shortest Path

Problem: Find the shortest path from a single node (source) to all other nodes

(shortest might mean fewest links, or lowest total weight)



# Dijkstra's Algorithm

- You 100% have seen this in CS240 or CS234, don't even pretend

## Step 1

Set all nodes as unvisited, with  $D = \text{infinity}$

Set source node's  $D$  to 0

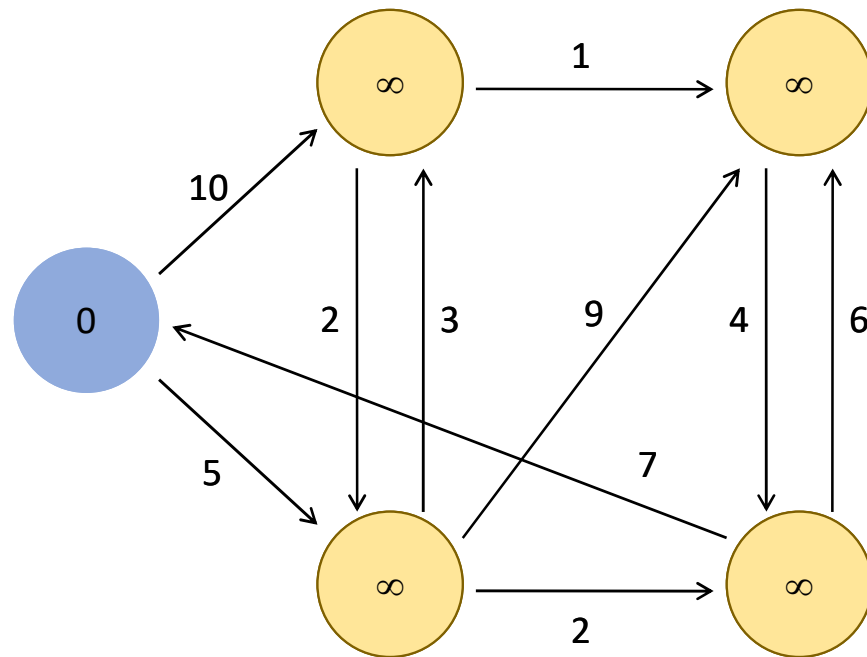
## Step 2

Let  $v$  be the unvisited node with lowest distance

For all out-neighbours  $u$  of  $v$ :

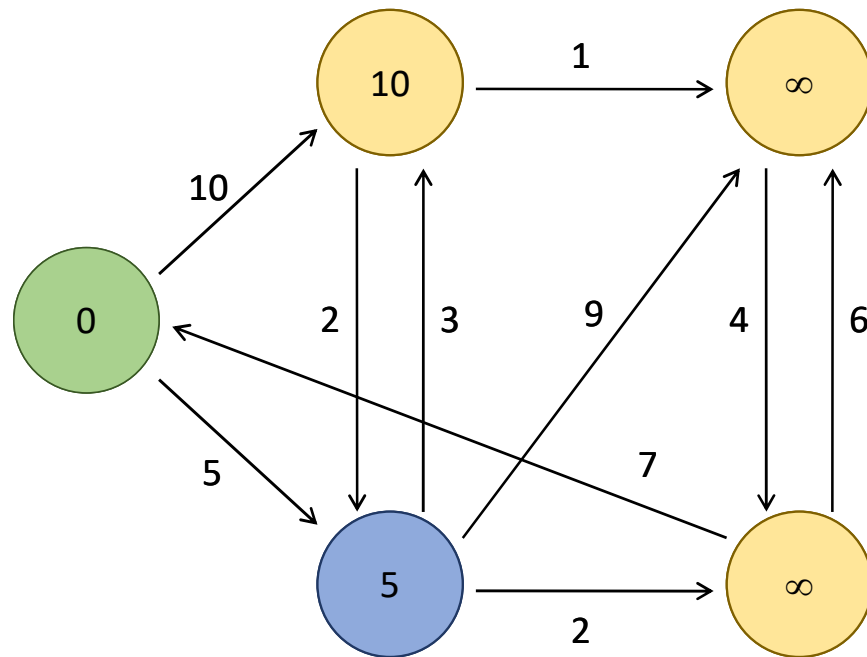
$$D[u] = \min(D[u], D[v] + \text{edge}(v, u).\text{weight})$$

# Dijkstra's Algorithm Example



Example from CLR

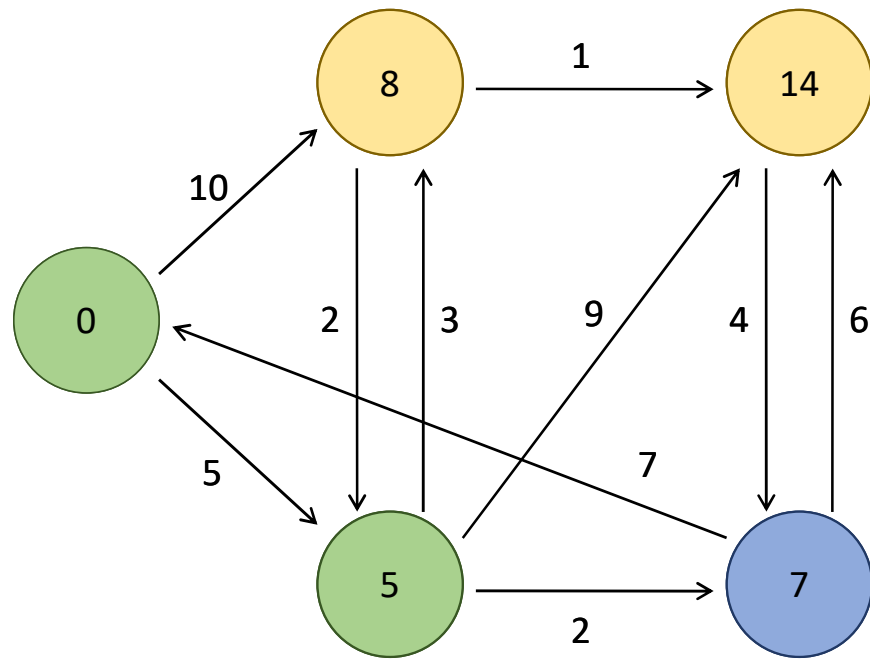
# Dijkstra's Algorithm Example



Example from CLR

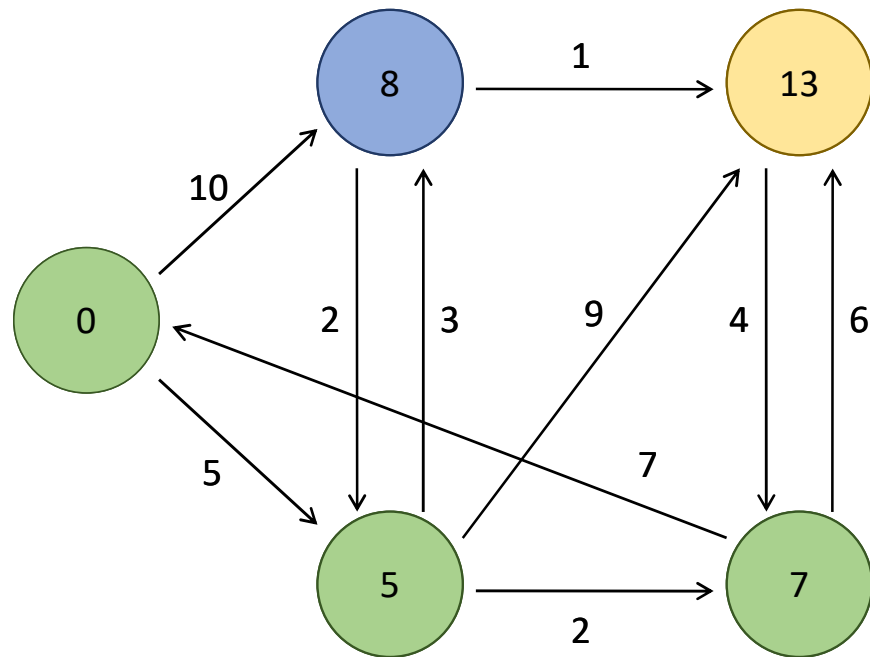


# Dijkstra's Algorithm Example



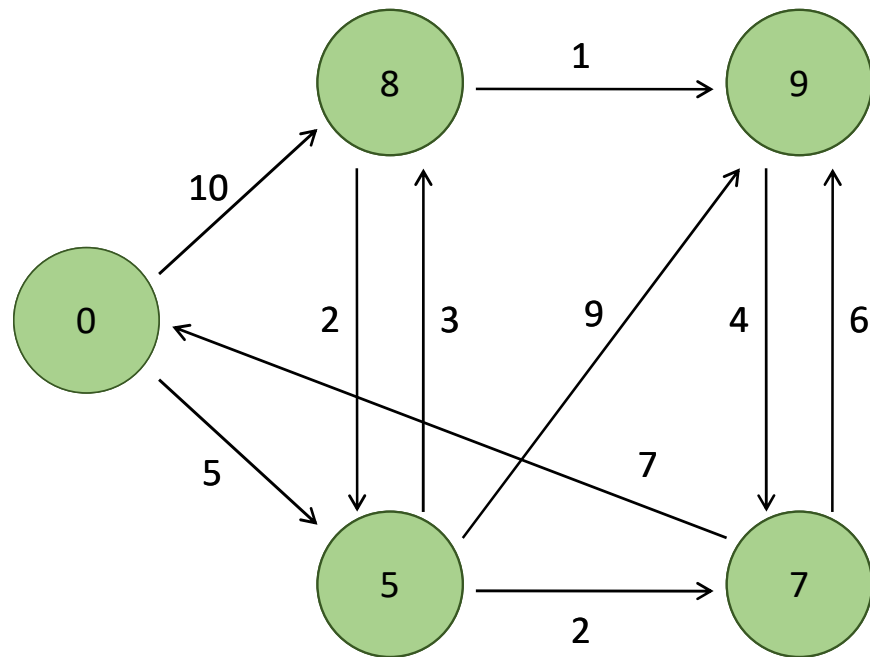
Example from CLR

# Dijkstra's Algorithm Example



Example from CLR

# Dijkstra's Algorithm Example



Example from CLR

# Dijkstra on MapReduce

Not Parallel 😞

Why?

- “Minimum D” – not local

So instead, we use:

- Parallel Breadth-First Search (pBFS)

# Simple Case – Unweighted Graph

Want the fewest “hops” from source to destination

Inductive Definition

- $\text{dist}(s) = 0$
- $\text{dist}(v) = 1$  if there is an edge from  $s$  to  $v$
- $\text{dist}(v) = 1 + \min_u (\text{dist}(u))$  (for all  $u$  s.t. there is an edge  $u \rightarrow v$ )

# Simple Case – Unweighted Graph

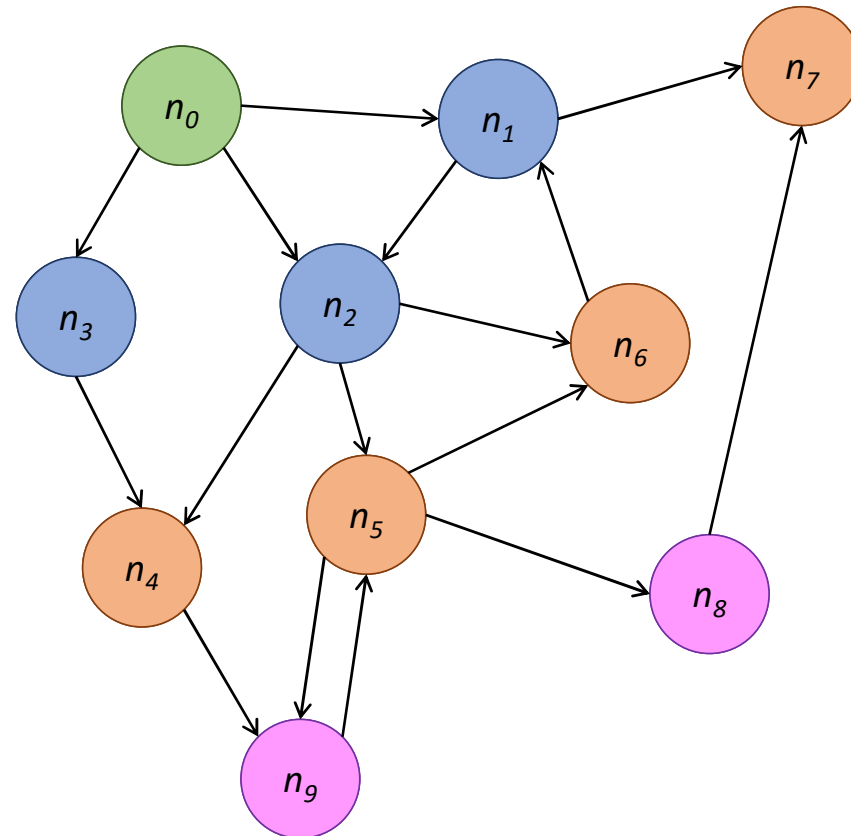
- Iteration

- 0

- 1

- 2

- 3



# Implementation on Hadoop

Keys – node  $n$

Values – ( $d$  -- distance to  $n$ , adjacency list of  $n$ )

Mapper:

for  $m$  in adjacency list, emit ( $m : d + 1$ )

also emit ( $n : d$ )

Reducer:

Update distance to node  $n$  based on ( $m : dist$ ) messages from mapper

# Iteration

To iterate, the output of the reducer gets passed to another (identical) job as the input.

How? Isn't the adjacency list gone?

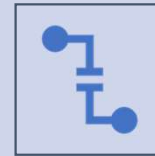
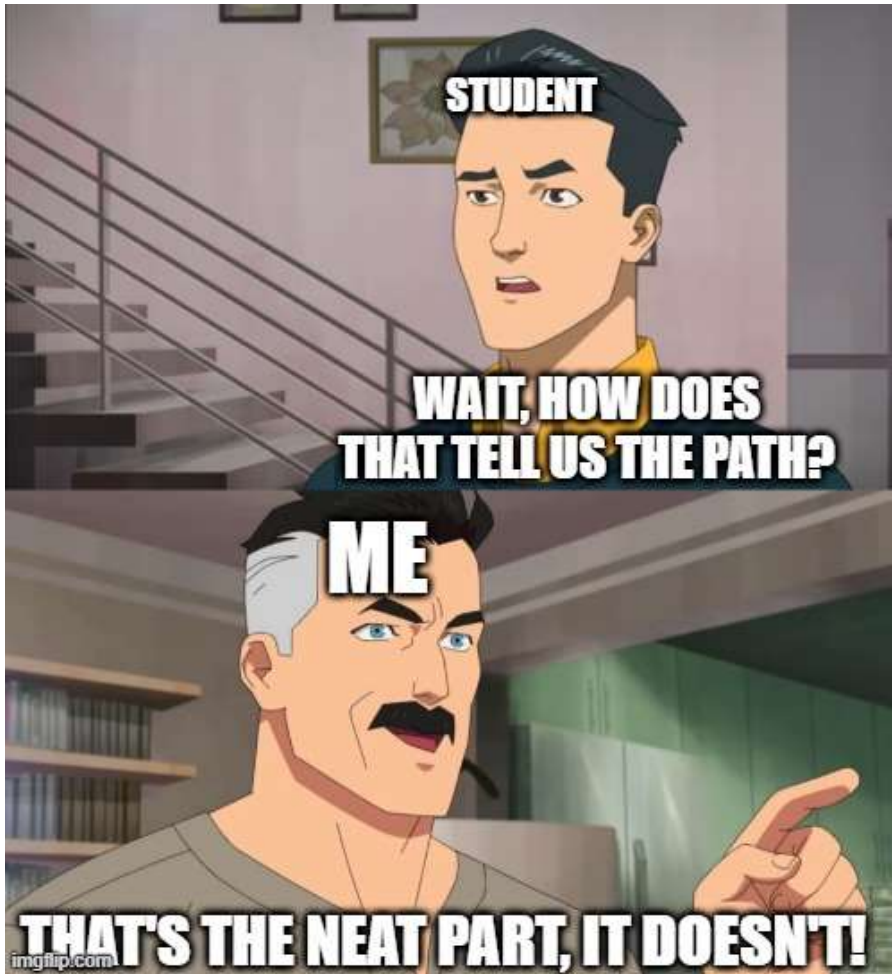
UGH. Yes. OK, so...also emit the adjacency list...



# Pseudocode

```
def map (id, node):  
    emit(id, node)  
    for m in node.adjList:  
        emit(m, node.d + 1))
```

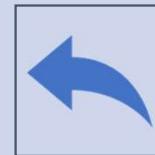
```
def reduce (id, values):  
    d = infinity  
    node = None  
    for o in values:  
        if isNode(o):  
            node = o  
        else:  
            d = min(d, o)  
    node.d = min(node.d, d)  
    emit(id, node)
```



It's not actually hard to modify this to include the path.



Option 1: Node has a field "path that results in given d"

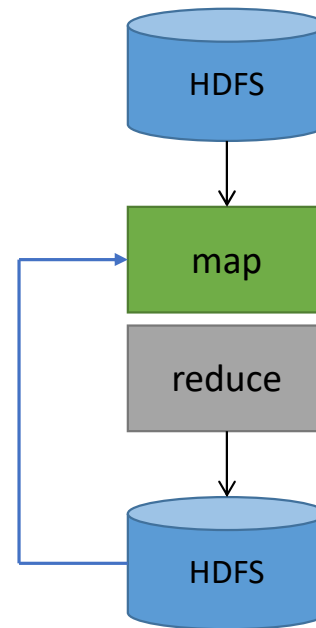


Option 2: Node has a field "previous"

# MapReduce Iteration

1. Do the job
2. If we need to keep going, make a new job that reads the output from the last job
3. See step 1

What's the issue here?



# Quitting Time

- The most important part of recursion is knowing when to stop
- The second most important part of recursion is knowing when to stop
- The third most important part of recursion is knowing when to stop



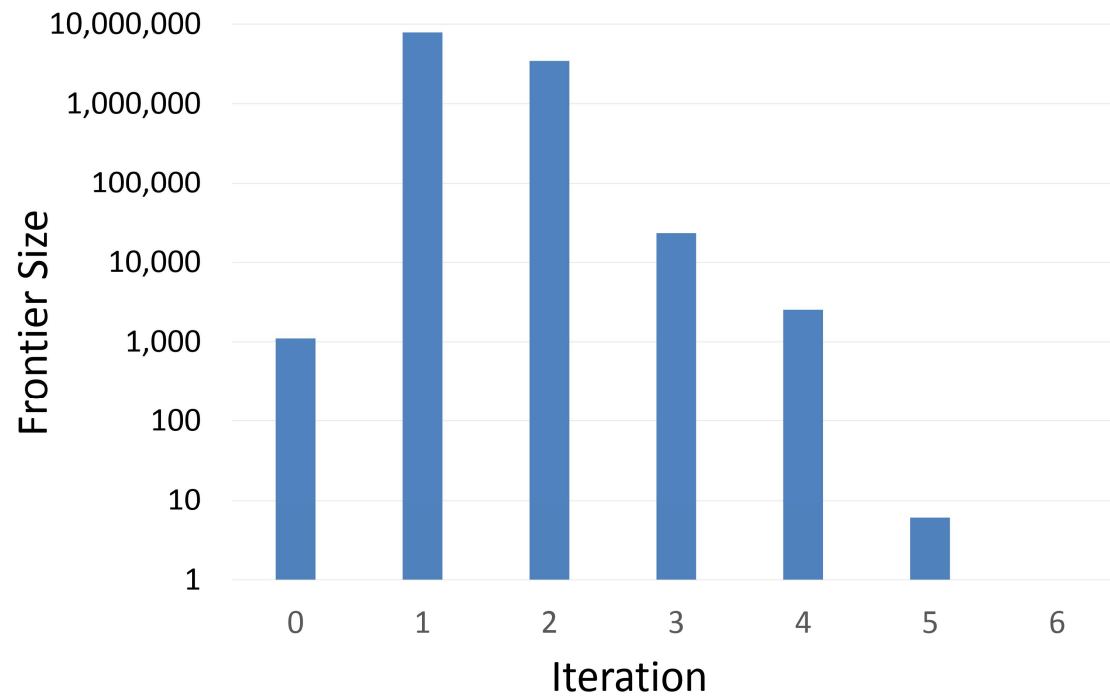
# Kevin Bacon

---

- All nodes with min distance of  $k$  are “visited” in iteration  $k$
- Why? Did you not see the colourful diagram???
- So how many to search the entire graph?
  - 6?



# Frontier size during BFS traversal



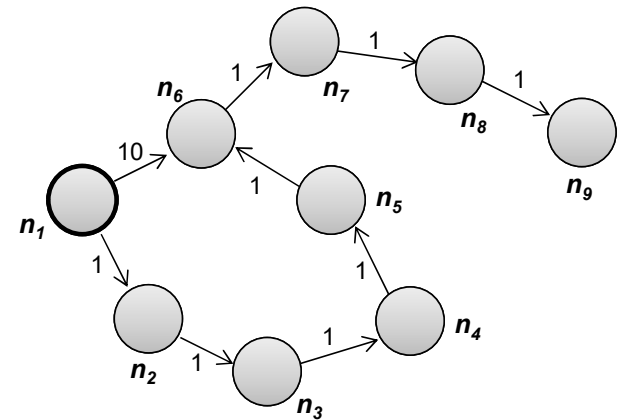
## Pseudocode (Weighted Edges)

```
def map (id, node):  
    emit(id, node)  
    for m in node.adjList:  
        emit(m.id, node.d + m.w))
```

```
def reduce (id, values):  
    d = infinity  
    node = None  
    for o in values:  
        if isNode(o):  
            node = o  
        else:  
            d = min(d, o)  
    node.d = min(node.d, d)  
    emit(id, node)
```

# Weighted Edges, Termination

- You can still update a node after you first “discover” it
  - Fuzzy Frontier
- Stopping condition unchanged
  - Stop when no changes
- More iterations needed
  - Could be a lot more





# BFS vs Dijkstra

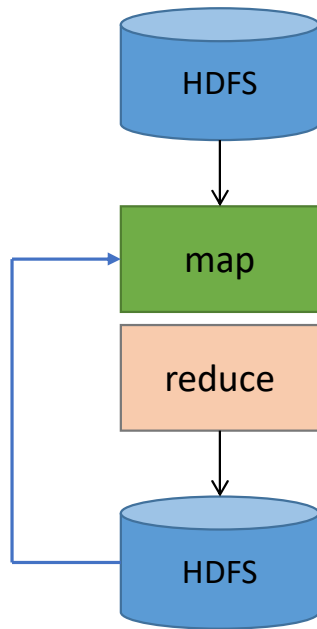
- Dijkstra only investigates the lowest-cost node on the frontier
- Parallel BFS investigates all paths in parallel
  - It's a simple optimization to restrict to the frontier
  - But you're still sending the adjacency list back and forth
- Can we do better on MapReduce?

## Issues with MapReduce Iteration

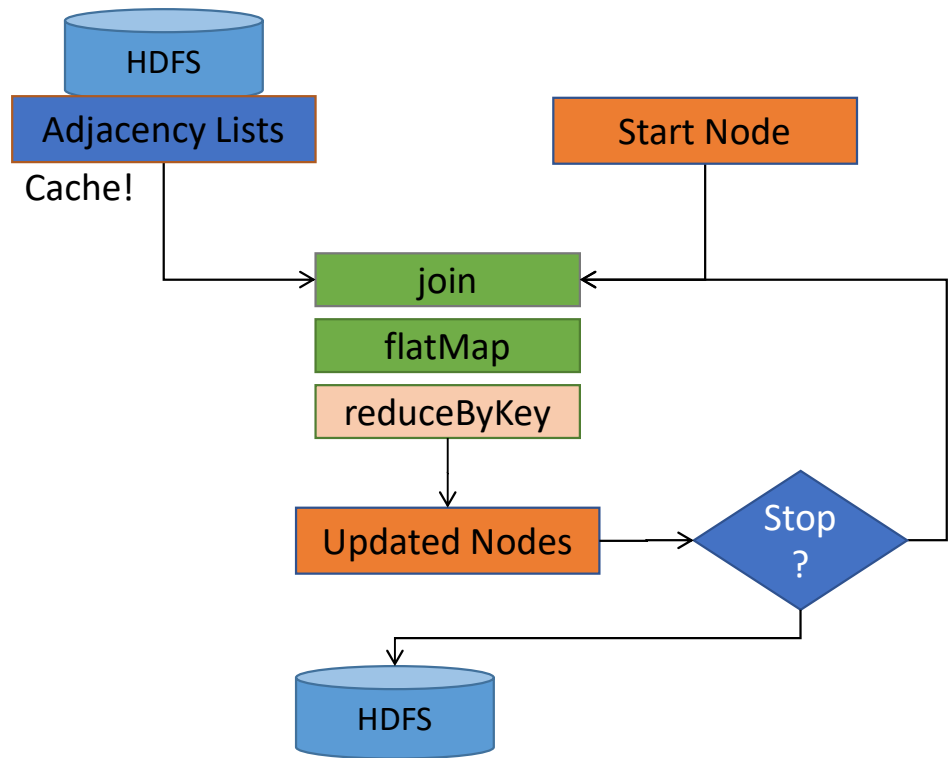
Everything is written to HDFS, then loaded again

We must send the entire graph structure to the reducers each iteration, only to have them send it back again

# MapReduce



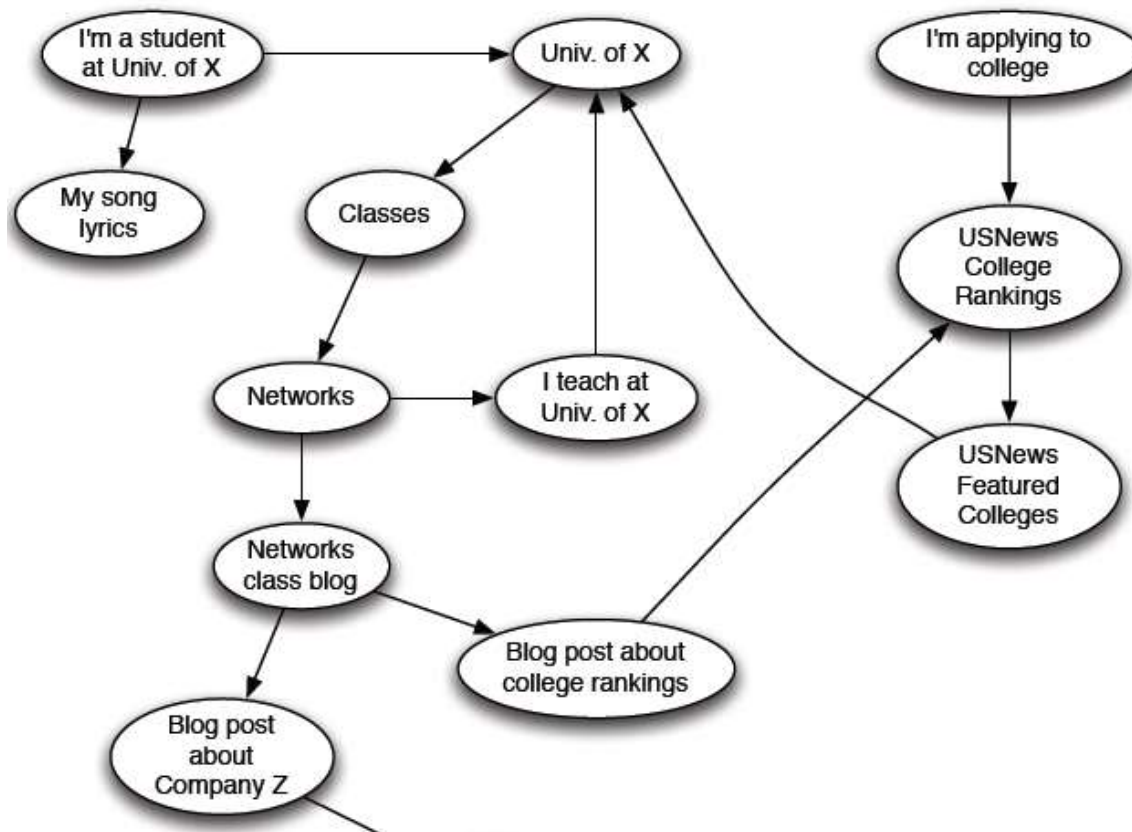
# Spark



## Something Borrowed

- Next Slides are thanks to Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University)

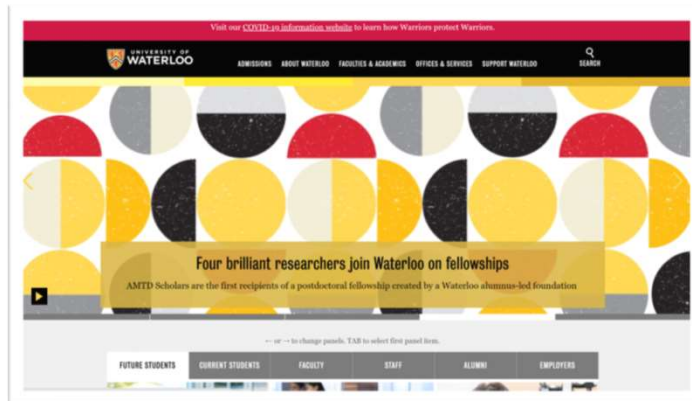
# Web as a Directed Graph



# Who to trust?

Query: University of Waterloo

uwaterloo.ca



fakeuw.ca



Ranked retrieval fails!

# Web Search Challenge

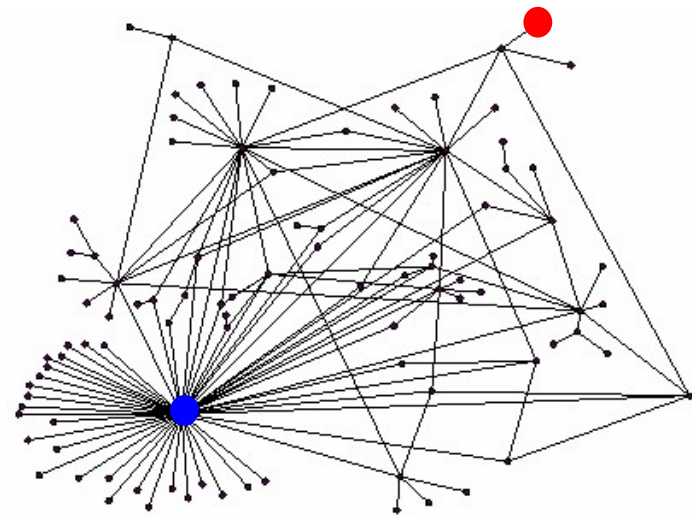
- **Web contains many sources of information**  
**Who to “trust”?**
  - **Trick:** Trustworthy pages may point to each other!

# Ranking Nodes on the Graph


- **All web pages are not equally “important”**

www.joeschmoe.com vs. www.stanford.edu

- There is large diversity in the web-graph node connectivity.  
**Let's rank the pages by the link structure!**





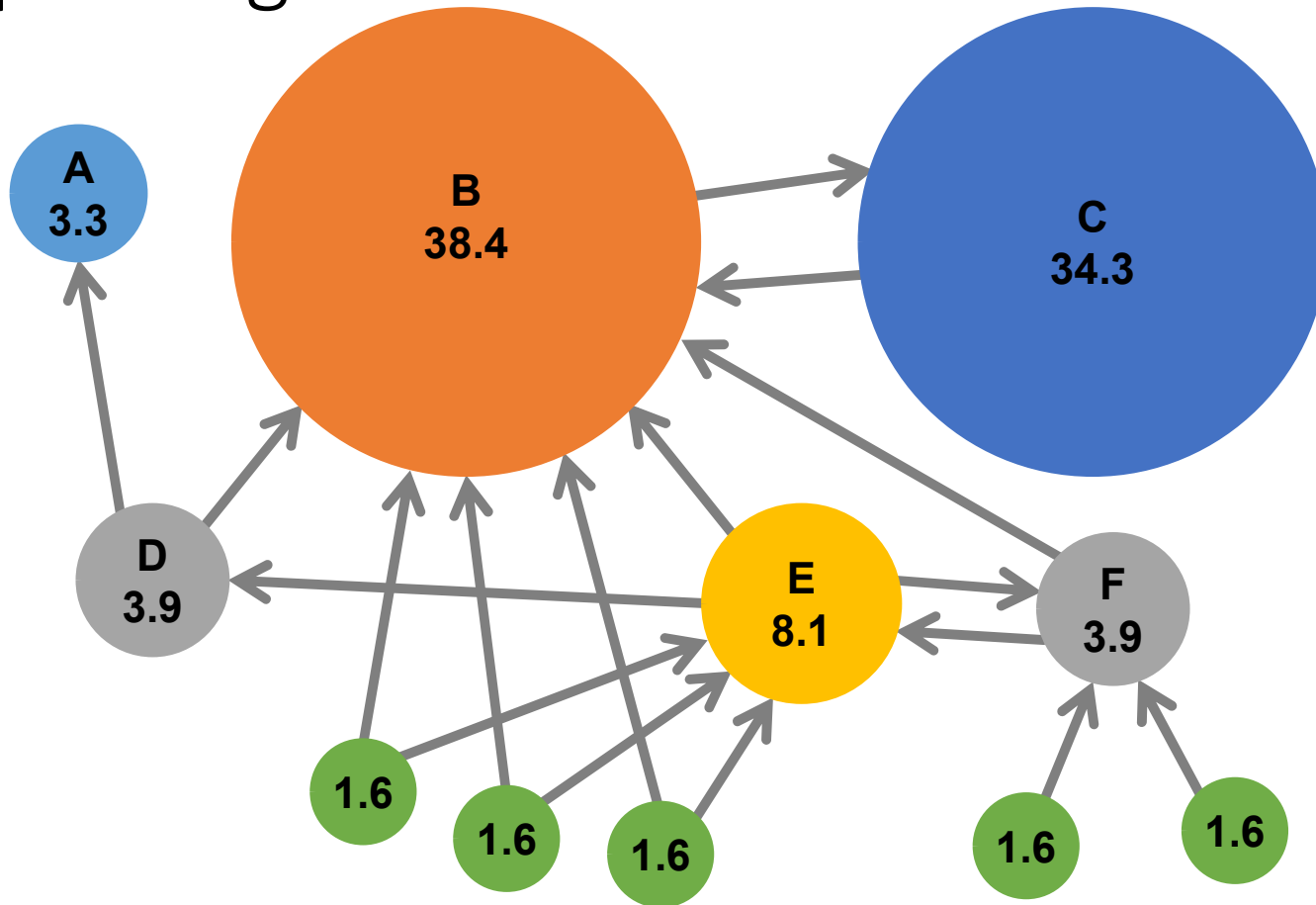


# PageRank: The “Flow” Formulation

# Links as Votes

- **Idea: Links as votes**
  - **Page is more important if it has more links**
    - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - [www.stanford.edu](http://www.stanford.edu) has 23,400 in-links
  - [www.joeschmoe.com](http://www.joeschmoe.com) has 1 in-link
- **Are all in-links equal?**
  - **Links from important pages count more**
  - Recursive question!

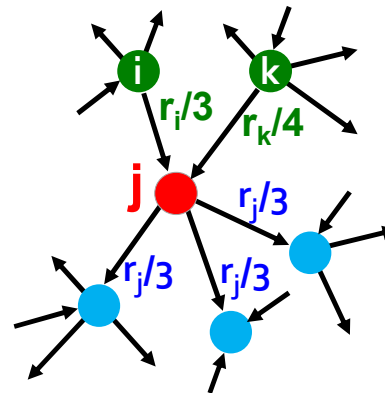
# Example: PageRank Scores



## Simple Recursive Formulation

- Each link's vote is proportional to the **importance** of its source page
- If page **j** with importance  $r_j$  has **n** out-links, each link gets  $r_j / n$  votes
- Page **j**'s own importance is the sum of the votes on its in-links

$$r_j = r_i/3 + r_k/4$$

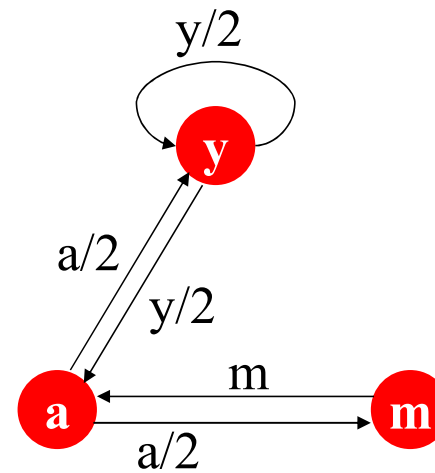


# PageRank: The “Flow” Model

- Define a “rank”  $r_j$  for page  $j$

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

$d_i$  ... out-degree of node  $i$



“Flow” equations:

$$r_y = r_y/2 + r_a/2$$

$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

# Solving the Flow Equations

- **3 equations, 3 unknowns, no constants**
  - No unique solution
  - All solutions equivalent modulo the scale factor
- **Additional constraint forces uniqueness:**
  - $r_y + r_a + r_m = 1$
  - **Solution:**  $r_y = \frac{2}{5}$ ,  $r_a = \frac{2}{5}$ ,  $r_m = \frac{1}{5}$
- **Gaussian elimination method works for small examples, but we need a better method for large web-size graphs**
- **We need a new formulation!**

Flow equations:

$$r_y = r_y/2 + r_a/2$$

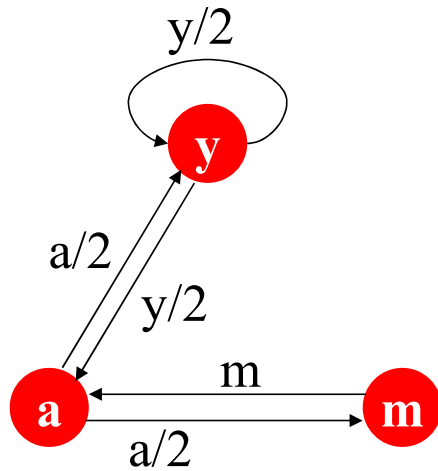
$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

# PageRank: Matrix Formulation

- **Stochastic adjacency matrix  $M$**

- Let page  $i$  has  $d_i$  out-links
- If  $i \rightarrow j$ , then  $M_{ji} = \frac{1}{d_i}$  else  $M_{ji} = 0$ 
  - $M$  is a **column stochastic matrix**
    - Columns sum to 1



	<b>y</b>	<b>a</b>	<b>m</b>
<b>y</b>	$\frac{1}{2}$	$\frac{1}{2}$	0
<b>a</b>	$\frac{1}{2}$	0	1
<b>m</b>	0	$\frac{1}{2}$	0

# PageRank: How to solve?

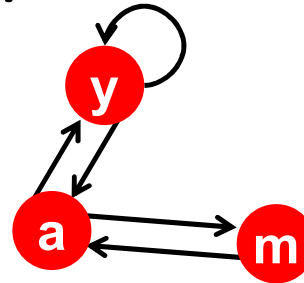
- **Power Iteration:**

- Set  $r_j = 1/N$
- **1:**  $r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$
- **2:**  $r = r'$
- Goto **1**

- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$$

Iteration 0, 1, 2, ...



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

$$r_y = r_y/2 + r_a/2$$

$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$



# PageRank: How to solve?

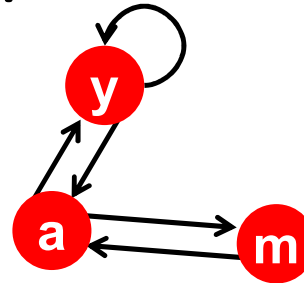
- **Power Iteration:**

- Set  $r_j = 1/N$
- **1:**  $r'_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$
- **2:**  $r = r'$
- Goto **1**

- **Example:**

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \dots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, ...



	y	a	m
y	1/2	1/2	0
a	1/2	0	1
m	0	1/2	0

$$r_y = r_y/2 + r_a/2$$

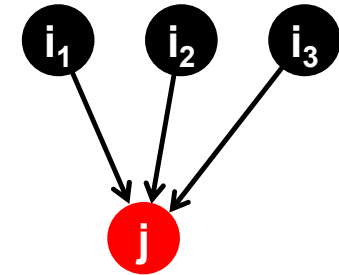
$$r_a = r_y/2 + r_m$$

$$r_m = r_a/2$$

# Random Walk Interpretation

## ■ Imagine a random web surfer:

- At any time  $t$ , surfer is on some page  $i$
- At time  $t + 1$ , the surfer follows an out-link from  $i$  uniformly at random
- Ends up on some page  $j$  linked from  $i$
- Process repeats indefinitely



$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_{\text{out}}(i)}$$



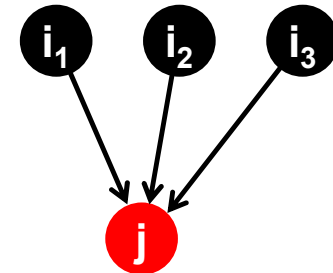
# Random Walk Interpretation

## ■ Imagine a random web surfer:

- At any time  $t$ , surfer is on some page  $i$
- At time  $t + 1$ , the surfer follows an out-link from  $i$  uniformly at random
- Ends up on some page  $j$  linked from  $i$
- Process repeats indefinitely

## ■ Let:

- $\mathbf{p}(t)$  ... vector whose  $i^{\text{th}}$  coordinate is the prob. that the surfer is at page  $i$  at time  $t$
- So,  $\mathbf{p}(t)$  is a probability distribution over pages



$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_{\text{out}}(i)}$$

# The Stationary Distribution

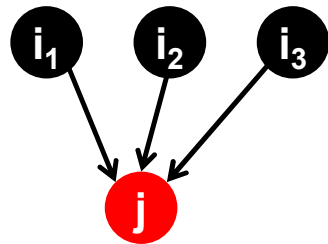
- **Where is the surfer at time  $t+1$ ?**

- Follows a link uniformly at random

$$\mathbf{p}(t + 1) = \mathbf{M} \cdot \mathbf{p}(t)$$

- Suppose the random walk reaches a state  $\mathbf{p}(t + 1) = \mathbf{M} \cdot \mathbf{p}(t) = \mathbf{p}(t)$

then  $\mathbf{p}(t)$  is **stationary distribution** of a random walk



$$\mathbf{p}(t + 1) = \mathbf{M} \cdot \mathbf{p}(t)$$

# Existence and Uniqueness

- **A central result from the theory of random walks (a.k.a. Markov processes):**

For graphs that satisfy **certain conditions**, the **stationary distribution is unique** and eventually will be reached no matter what the initial probability distribution at time  **$t = 0$**

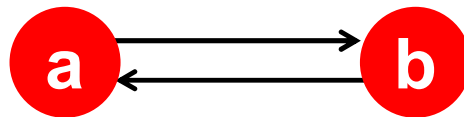
# PageRank: The Google Formulation

# PageRank: Three Questions

$$r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}$$

- Does this converge?
- Does it converge to what we want?
- Are results reasonable?

Does this converge?



$$r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}$$

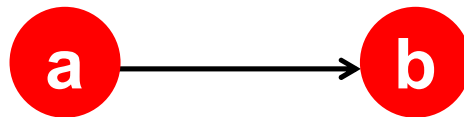
• **Example:**

	$\begin{matrix} = \\ \hline \end{matrix}$			
$r_a$	1	0	1	0
$r_b$	0	1	0	1

Iteration 0, 1, 2, ...



Does it converge to what we want?



$$r_j^{(t+1)} = \sum_{i \rightarrow j} \frac{r_i^{(t)}}{d_i}$$

• **Example:**

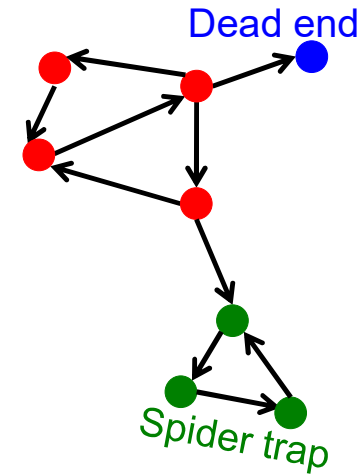
	$\begin{matrix} = \\ \hline \end{matrix}$			
$r_a$	1	0	0	0
$r_b$	0	1	0	0

Iteration 0, 1, 2, ...

# PageRank: Problems

## 2 problems:

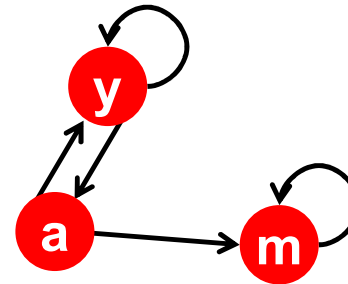
- **(1)** Some pages are **dead ends** (have no out-links)
  - Random walk has “nowhere” to go to
  - Such pages cause importance to “leak out”
- **(2) Spider traps:**  
(all out-links are within the group)
  - Random walker gets “stuck” in a trap
  - And eventually spider traps absorb all importance



# Problem: Spider Traps

- Power Iteration:**

- Set  $r_j = 1$
- $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$ 
  - And iterate



m is a spider trap

	y	a	m
y	1/2	1/2	0
a	1/2	0	0
m	0	1/2	1

$$r_y = r_y/2 + r_a/2$$

$$r_a = r_y/2$$

$$r_m = r_a/2 + r_m$$

- Example:**

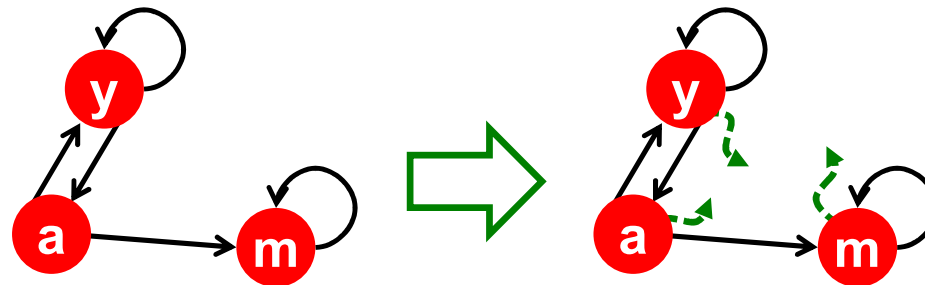
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{pmatrix}$$

Iteration 0, 1, 2, ...

All the PageRank score gets “trapped” in node m.

# Solution: Teleports!

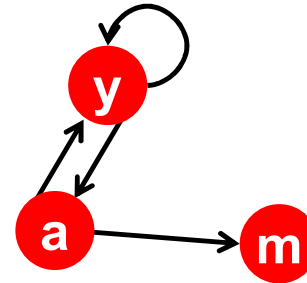
- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob.  $\beta$ , follow a link at random
  - With prob.  $1-\beta$ , jump to some random page
  - Common values for  $\beta$  are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



# Problem: Dead Ends

- **Power Iteration:**

- Set  $r_j = 1$
- $r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$ 
  - And iterate



	y	a	m
y	1/2	1/2	0
a	1/2	0	0
m	0	1/2	0

$$r_y = r_y/2 + r_a/2$$

$$r_a = r_y/2$$

$$r_m = r_a/2$$

- **Example:**

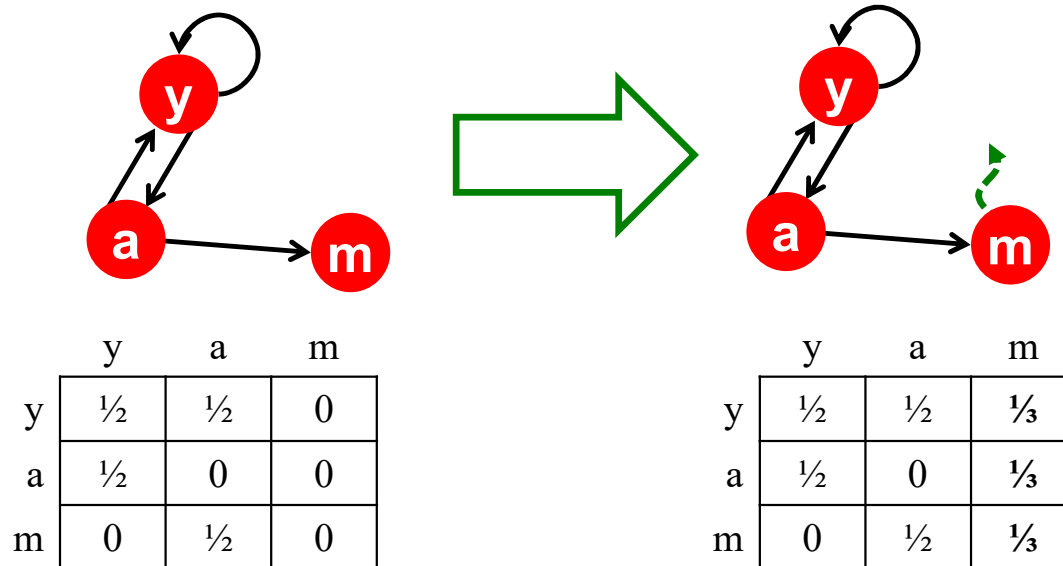
$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{pmatrix}$$

Iteration 0, 1, 2, ...

Here the PageRank “leaks” out since the matrix is not stochastic.

# Solution: Always Teleport!

- **Teleports:** Follow random teleport links with probability 1.0 from dead-ends
  - Adjust matrix accordingly



# Why Teleports Solve the Problem?

**Why are dead-ends and spider traps a problem and why do teleports solve the problem?**

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic, so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

# Solution: Random Teleports

- **Google's solution that does it all:**

At each step, random surfer has two options:

- With probability  $\beta$ , follow a link at random
- With probability  $1-\beta$ , jump to some random page

- **PageRank equation** [Brin-Page, 98]

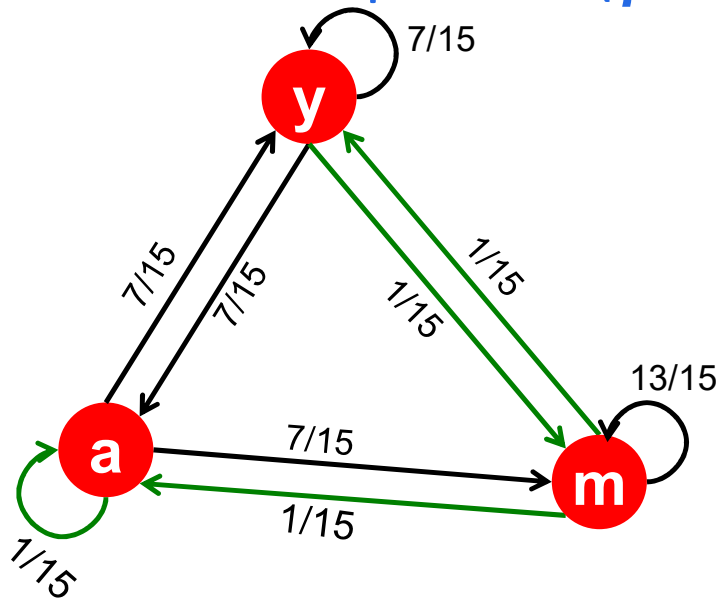
$$r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

$d_i$  ... out-degree of node  $i$

This formulation assumes that  $M$  has no dead ends. We can either preprocess matrix  $M$  to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.



# Random Teleports ( $\beta = 0.8$ )



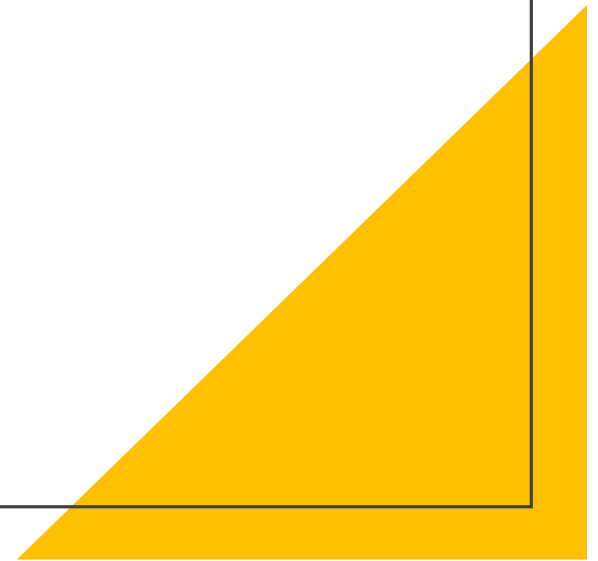
$$0.8 \begin{matrix} & \mathbf{M} \\ \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} & + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} \mathbf{A} \\ \begin{matrix} y \\ a \\ m \end{matrix} \end{matrix} \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

y	=	1/3	0.33	0.24	0.26	...	7/33
a		1/3	0.20	0.20	0.18	...	5/33
m		1/3	0.46	0.52	0.56		21/33

# PageRank with MapReduce

We now return to Dan's slides



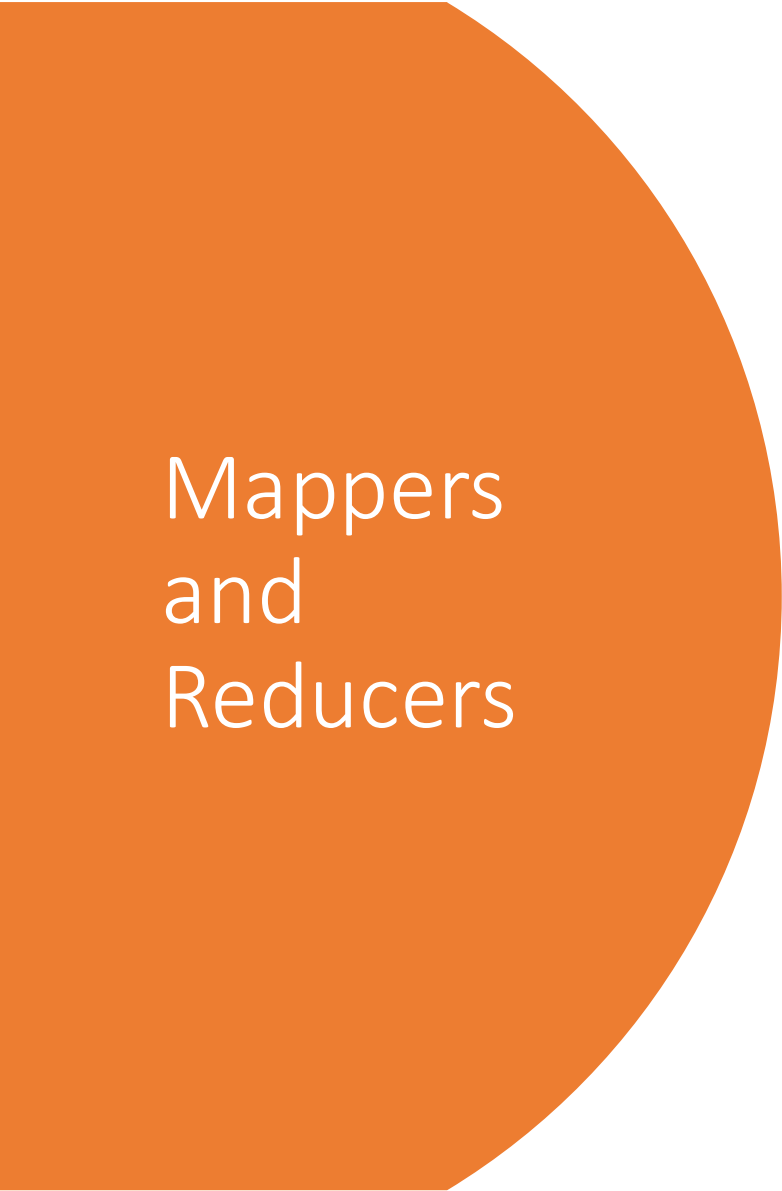
Keep it  
Simple (At  
First)

---

No random  
jumps

---

No dead-ends

A large orange shape on the left side of the slide, consisting of a vertical rectangle on the left and a quarter-circle on the right.

Mappers  
and  
Reducers

Map Phase – Each node  
“sends” its importance to its  
out-links

Reduce Phase – Each node  
sets its new importance to  
the sum of the received  
values

# Pseudocode

```
def map(id, n):  
    emit(id, n)  
    p = n.rank / len(n.adj)  
    for m in n.adj:  
        emit(m, p)
```

```
def reduce(id, msgs):  
    n = None  
    sum = 0  
    for o in msgs:  
        if o is Node:  
            n = o  
        else:  
            s += o  
    n.rank = s  
    emit(id, n)
```

## PageRank vs. BFS

	PageRank	BFS
Map	PR/N	d+1
Reduce	sum	min

A large class of graph algorithms involve:

Local computations at each node

Propagating results: “traversing” the graph



## Complete PageRank

Missing two things:

- Random Jumps ( $1-\beta$ )
- Dead End Jumps  
(always)

How can we add these?

# Random Jumps

Every node has the same chance of jumping:  $1 - \beta$

If it jumps, it jumps to ANY random page ( $1/N$  for a particular page)

Only need to change reducer side



# New Reducer

```
def reduce(id, msgs):  
    n = None  
    sum = 0  
    for o in msgs:  
        if o is Node:  
            n = o  
        else:  
            s += o  
    n.rank = s *  $\beta$  + (1 -  $\beta$ ) / N  
    emit(id, n)
```

# What about dead-ends?

Dead-Ends send all their weight everywhere, instead of only some of it

- Option 1: Replace dead-ends with “links to everyone, everywhere”
  - No changes to the code
  - That’s a lot of messages
- Option 2: Post-process to redistribute “missing mass”
  - Avoids making N-degree nodes
  - Aww man, I have to think?



## Post-Processing

$R = \text{sum of all ranks}$

$1 - R = \text{“missing mass” (sum of ranks of dead ends)}$

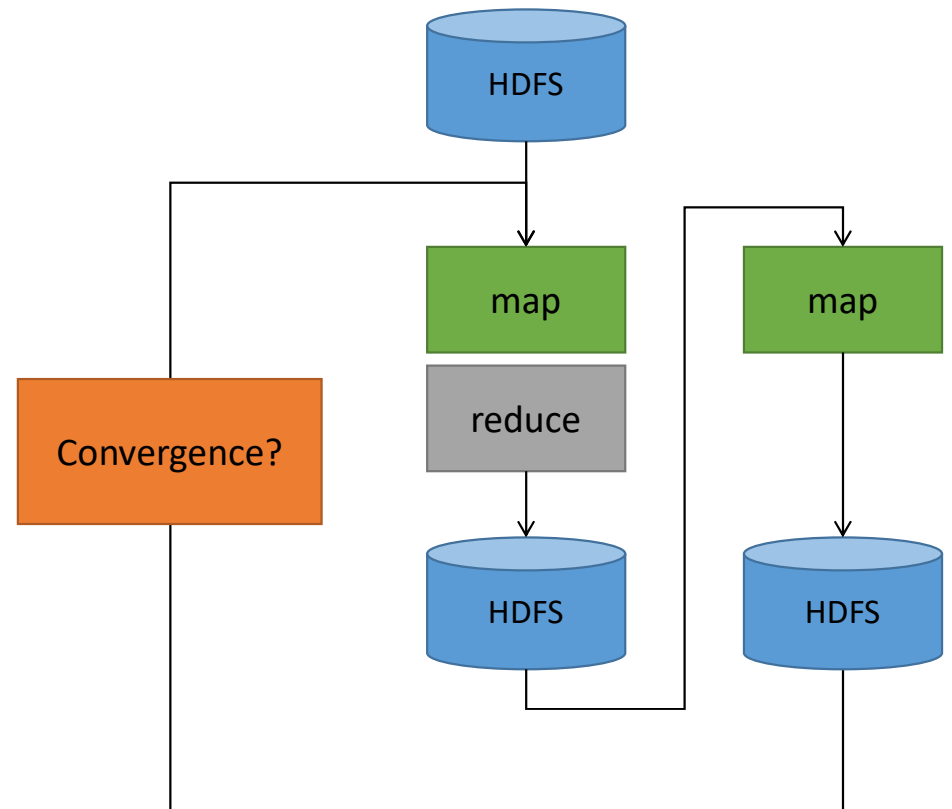
Add  $(1 - R)/N$  to all nodes

---

# Post Processing

Note: You can move the “post processing” map into the “next map”

(Just remember to adjust the convergence test)





## Alternative

Mappers send:

- $\beta(\text{node.rank})$  divided evenly amount out-links
- $(1 - \beta)(\text{node.rank})$  to “everyone”
- special case: dead-ends send entire rank to “everyone”

Reducer adds  $(1/N) \times$  “everyone” rank to each sum

---



## Small Problem

$N = 1$  billion. The individual masses will be small.

Solution: store the logs

---



## Log Masses

$$a = \log m, b = \log n$$

$$m \times n \rightarrow a + b$$

$$m + n \rightarrow \begin{cases} b + \log(1 + e^{a-b}) & a < b \\ a + \log(1 + e^{b-a}) & a \geq b \end{cases}$$

---

# Why Log Masses Work

---

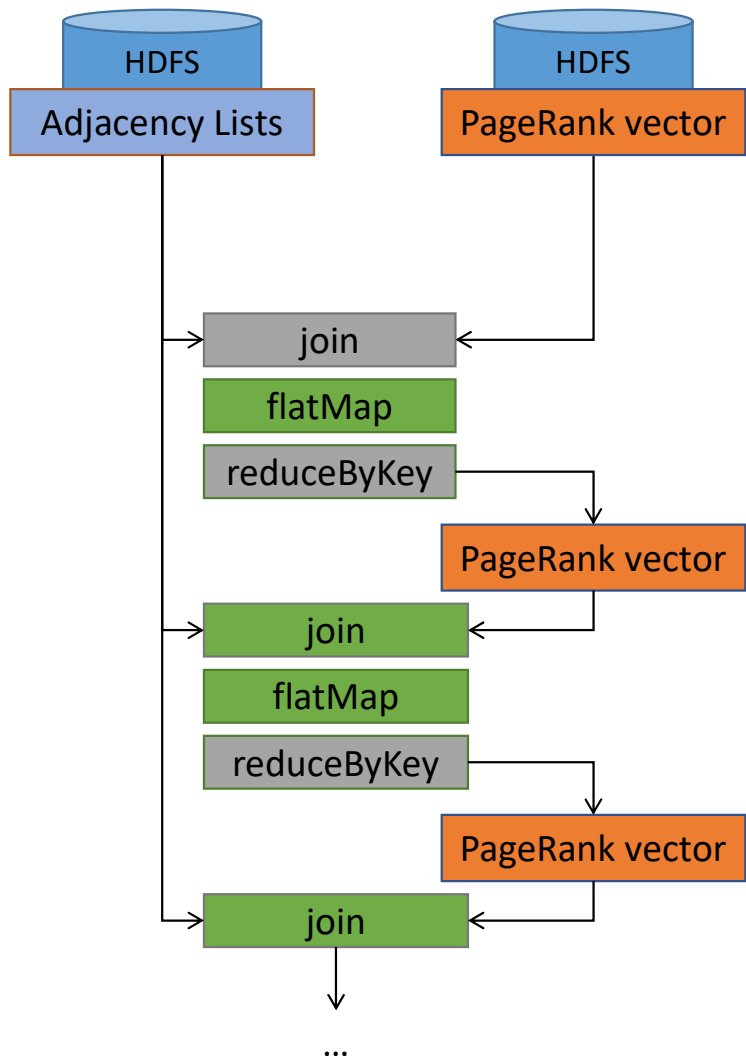
- Ranks are in  $[0,1]$  -- or, really,  $(0,1)$
- Mathematically,  $\sim \frac{1}{4}$  of all float values are also in this range
  - Sign + =  $\frac{1}{2}$  , exponent negative =  $\frac{1}{2}$
- $\text{Log}(x) : [0,1] \Rightarrow [-\infty, 0]$ 
  - Now we're using  $\frac{1}{2}$  of the values instead of  $\frac{1}{4}$
- It also lets us store MUCH smaller numbers without underflow
  - Most pages will have a very small but non-zero PageRank so this is important



# Let's Spark

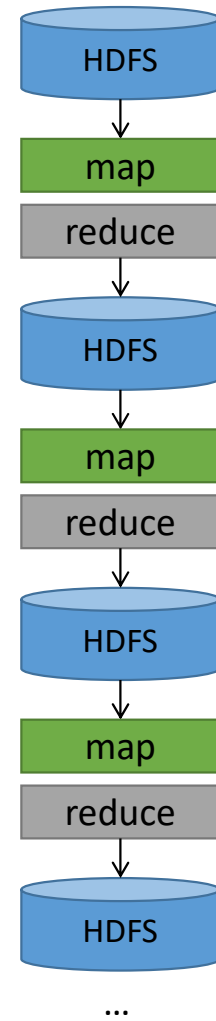
## Problems with MapReduce

- Sending Adjacency lists with each iteration (never changes)
- Needless Shuffling
- Needless Filesystem Access
- Verbose programs
- Each iteration is a new job
  - Hadoop has a fairly long start-up time per job

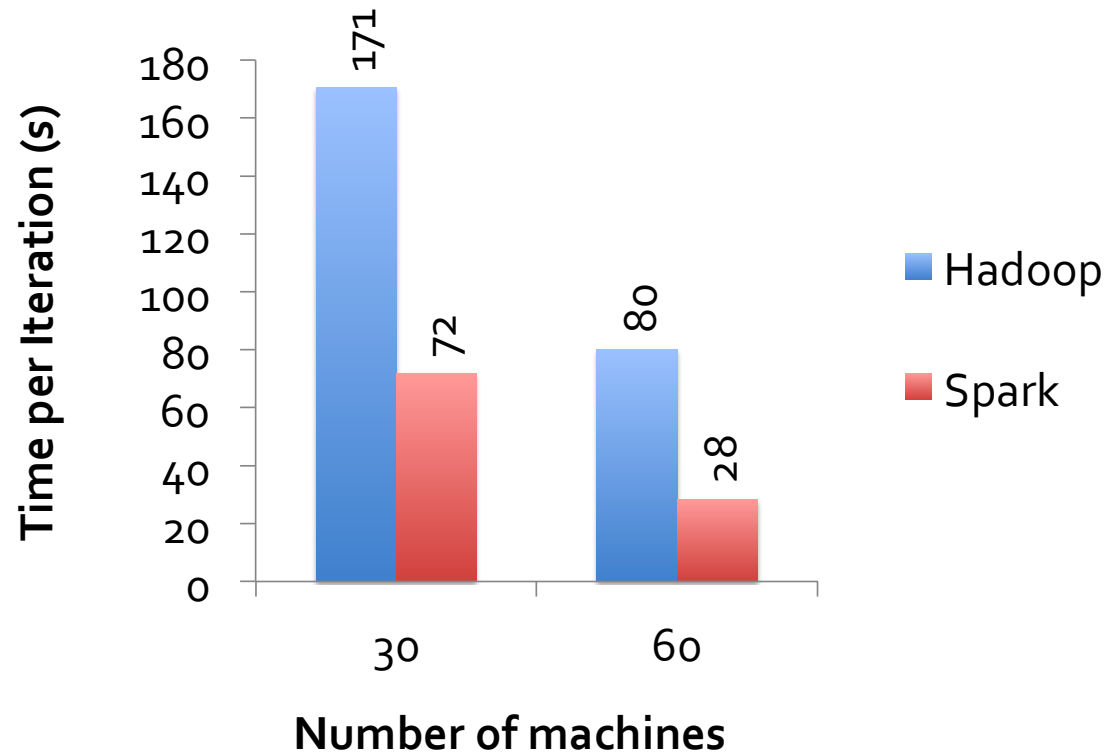


### Colour Legend

- **Cached RDD** (Blue)
- **Uncached RDD** (Orange)
- **No Shuffle** (Green)
- **Shuffle** (Grey)



# MapReduce vs. Spark



# PageRank in Spark

```
val lines = sc.textFile("the-internet.txt")
val damp = 0.85
val links = lines.map{ s =>
    val parts = s.split("\\s+")
    (parts(0), parts(1))
}.distinct().groupByKey().cache()

var ranks = links.mapValues(v => 1.0)

for (i <- 1 to iters) {
    val contribs = links.join(ranks).values.flatMap{ case (urls, rank) =>
        val size = urls.size
        urls.map(url => (url, rank / size))
    }
    ranks = contribs.reduceByKey(_ + _).mapValues((1 - damp) + damp * _)
}
```

# PageRank in PySpark

```
lines = sc.textFile("the-internet.txt").map(lambda r: r[0])
# Loads all URLs from input file and initialize their neighbors.
links = lines.map(lambda urls: parseNeighbors(urls)).distinct().groupByKey().cache()

# Loads all URLs with other URL(s) link to from input file and initialize ranks of them to one.
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

# Calculates and updates URL ranks continuously using PageRank algorithm.
for iteration in range(iterations):
    # Calculates URL contributions to the rank of other URLs.
    contribs = links.join(ranks).flatMap(lambda url_urls_rank: computeContribs(
        url_urls_rank[1][0], url_urls_rank[1][1] # type: ignore[arg-type]
    ))
    # Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * 0.85 + 0.15)
```

# PageRank in MapReduce



See Bespin Implementation



I can't paste it here, it's 600 lines long

The background of the slide is a dense field of stars. Most are dark grey or black with a subtle metallic sheen. Several stars are a bright, reflective gold color, standing out prominently. The stars are scattered across the frame, creating a textured, starry effect.

# Page Rank Improvements

# Remember Search?

Old Search Ranking – TF and DF and logarithms

Flaw: Term Spam.

Set div to not render: Spam spam spam spam spam spam spam spam ...

New Search Ranking – Page Rank

New Term Spam:

A SEO walks into a bar pub inn roadhouse saloon tavern alehouse beer house  
beer garden public house drinkery beer ale draught wine...



# Solution?

Trust what others say about you, not what you say about yourself:

Use link text (and surrounding text) as terms, instead of contents of page

Remember “tragic love story” vs “star-crossed romance”? Solved.

# It has its own problems, though



The screenshot shows a Google search interface with the search term "miserable failure" entered. The search results are displayed under the "Web" tab, showing the first 10 results. The first result is a biography of President George W. Bush from the official White House website. The second result is the official site of Michael Moore. The third result is a BBC News article titled "Americas | 'Miserable failure' links to Bush". The fourth result is a search engine watch article about the search for "miserable failure" on Google.

Google Web Images Groups News Froogle Local more »  
miserable failure Search Advanced Search Preferences

**Web** Results 1 - 10 of about 969,000 for **miserable failure**. (0.06 seconds)

[Biography of President George W. Bush](#)  
Biography of the president from the official White House web site.  
[www.whitehouse.gov/president/gwbbio.html](http://www.whitehouse.gov/president/gwbbio.html) - 29k - [Cached](#) - [Similar pages](#)  
[Past Presidents](#) - [Kids Only](#) - [Current News](#) - [President](#)  
[More results from www.whitehouse.gov »](#)

[Welcome to MichaelMoore.com!](#)  
Official site of the gadfly of corporations, creator of the film Roger and Me and the television show The Awful Truth. Includes mailing list, message board, ...  
[www.michaelmoore.com/](http://www.michaelmoore.com/) - 35k - [Sep 1, 2005](#) - [Cached](#) - [Similar pages](#)

[BBC NEWS | Americas | 'Miserable failure' links to Bush](#)  
Web users manipulate a popular search engine so an unflattering description leads to the president's page.  
[news.bbc.co.uk/2/hi/americas/3298443.stm](http://news.bbc.co.uk/2/hi/americas/3298443.stm) - 31k - [Cached](#) - [Similar pages](#)

[Google's \(and Inktomi's\) Miserable Failure](#)  
A search for **miserable failure** on Google brings up the official George W. Bush biography from the US White House web site. Dismissed by Google as not a ...  
[searchenginewatch.com/sereport/article.php/3296101](http://searchenginewatch.com/sereport/article.php/3296101) - 45k - [Sep 1, 2005](#) - [Cached](#) - [Similar pages](#)

# Forum Spam / Comment Spam

What if you go to every page that allows posting, and link to your webpage?

Now YouTube, Facebook, CBC News, etc. all link to you.

They have high rank => You have high rank

You also chose the link text, so you're picking your own terms again



# Spam Farming

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets,  
<http://www.mmds.org>

# Spam Farming Techniques

“Spider traps” accumulate rank.

- Random jumps prevent them from accumulating ALL rank, but it’s still boosted by the topology

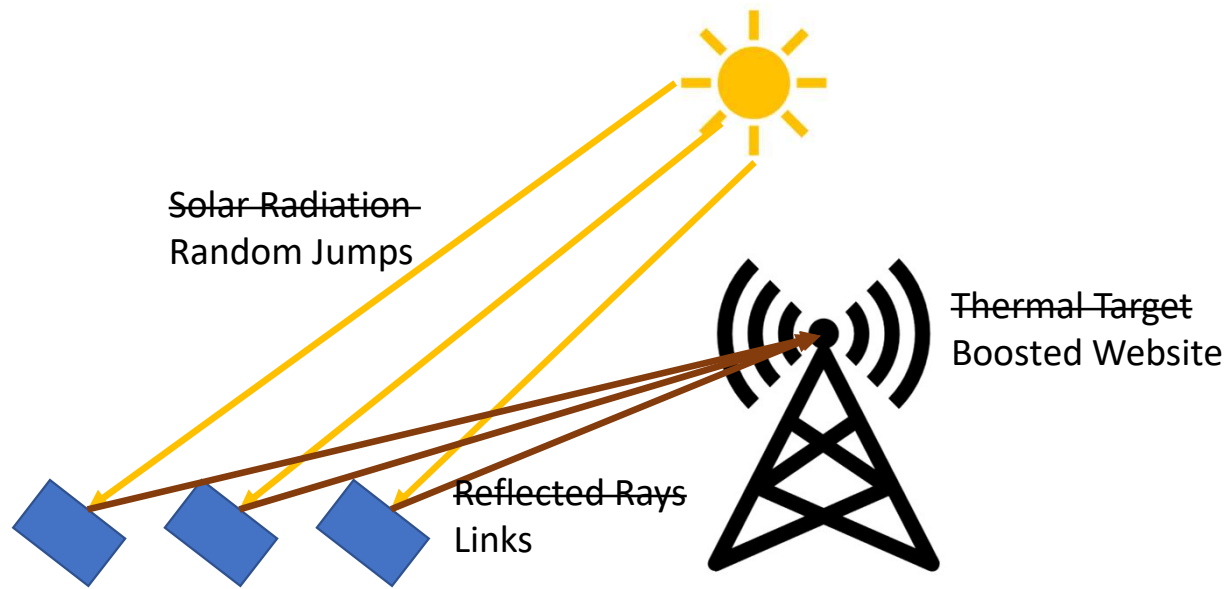
Technique:

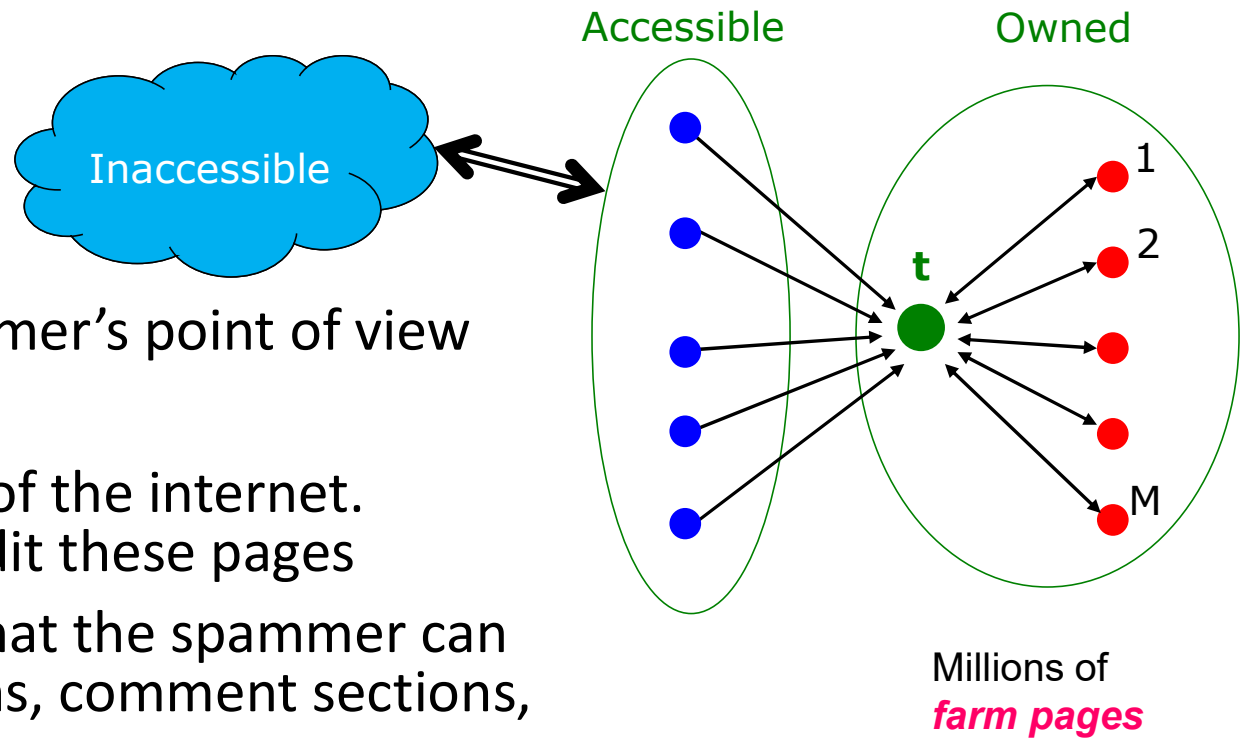
- Page you want to promote has millions of hidden links to farm pages
  - They all accumulate the random-jump weight
- Farm pages all link back to the page you want to promote
  - They send all their rank back to the page being boosted

An aerial photograph of a solar power plant. In the foreground, a tall, white central receiver tower stands on a concrete base, surrounded by various industrial structures and piping. The tower is labeled 'Bogus Website'. The middle ground is filled with a vast field of heliostats (mirrors) that reflect sunlight onto the tower. The heliostats are arranged in a grid pattern and are labeled 'Link Farm'. The background shows a flat, arid landscape under a clear sky.

Bogus  
Website

Link Farm



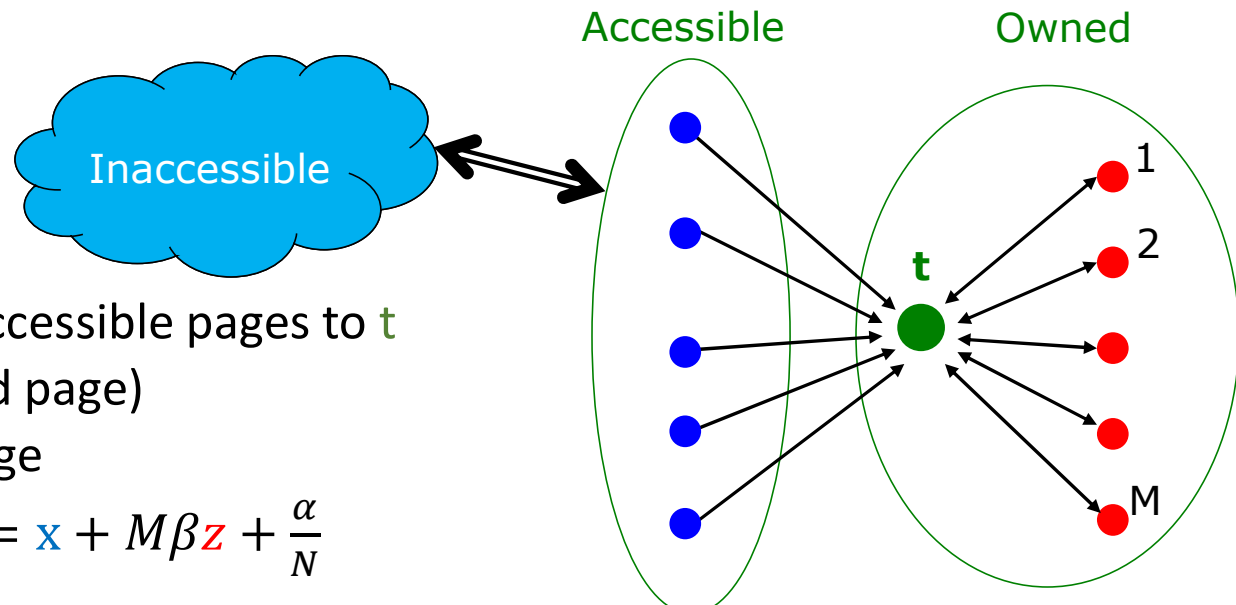


Pages from the spammer's point of view

- **Inaccessible:** Most of the internet. Spammer cannot edit these pages
- **Accessible:** Pages that the spammer can post links on (forums, comment sections, etc.)
- **Owned:** Pages the spammer directly controls



# MATH!



$x$ : total contribution of accessible pages to  $t$

$y$ : page rank of  $t$  (boosted page)

$z$ : page rank of a farm page

$$z = \frac{\beta y}{M} + \frac{\alpha}{N} \qquad y = x + M\beta z + \frac{\alpha}{N}$$

$$y = x + M\beta \left[ \frac{\beta y}{M} + \frac{\alpha}{N} \right] + \frac{\alpha}{N}$$

This term is very small, so ignored

Millions of *farm pages*

3.6x for  $\beta = 0.85$   
Spider trap amplifies incoming links

$$y = \frac{x}{1 - \beta^2} + \frac{\beta M}{(1 + \beta)N}$$

For  $\beta = 0.85$ ,  $0.45M/N$   
Grows linearly with  $M$

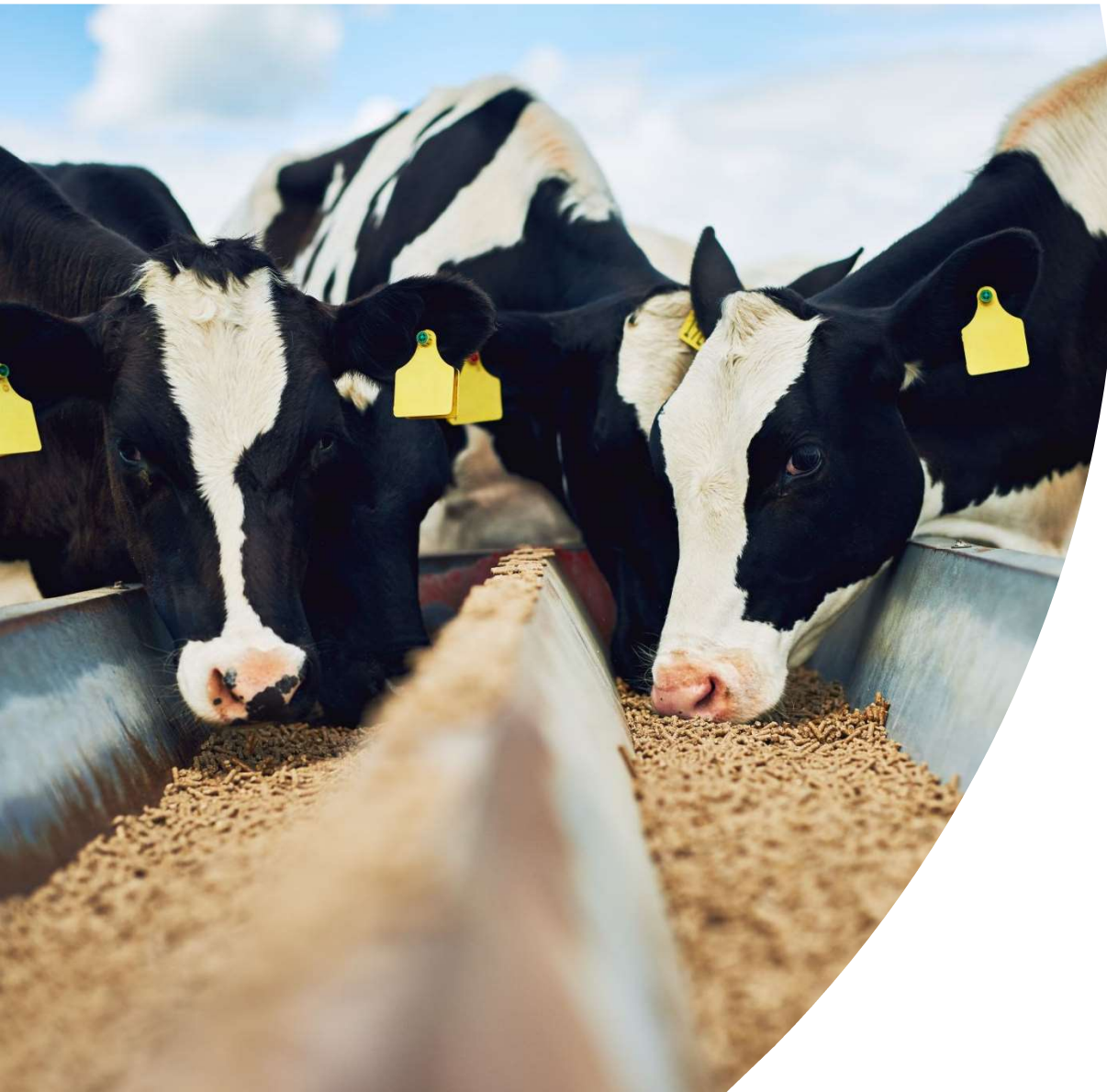
# Solution to Link Spam

- Ignore links tagged as “nofollow”
- Convince forums, news sites, etc. to insert “nofollow” to all links posted in comments

Added Benefit: A researcher (university website, high rank) can link to a page (to use as an example of term spam) and not boost its ranking

Makes this 0.  
Solved?

$$y = \frac{x}{1 - \beta^2} + \frac{\beta M}{(1 + \beta)N}$$



# How to Solve a Problem like Spam Farms

# Solution to Link Farms



What's the solution?



It's the topic of the Graph assignment!



In Personalized Page Rank, spam farms don't work. Why?

# What to use for “Source Nodes”

We should identify “trustworthy” pages

Easy to say...

What’s trustworthy?

Domains with strict entry requirements?

.edu, .gov, etc.

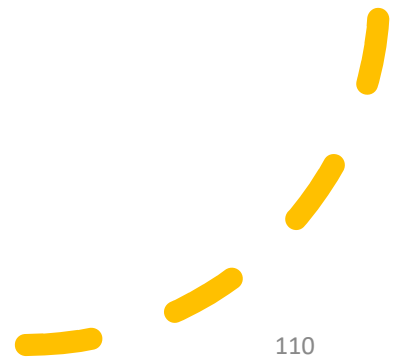
(UW doesn’t make the cut...)

# Idea

Collect a sampling of webpages (seed pages)

Oracle (Human) sorts the trustworthy from the spam.

- Expensive
- Keep the set as small as possible





Idea

Use the “good” pages as the source nodes for personalized page rank

Small change: Each page in trust set is initialized to 1:

Trust sums to  $M$  instead of to 1

After iteration, all pages have a trust factor of between 0 and 1

Pick a threshold and mark all pages below that as spam



# Justification

- Trustworthy pages mostly only link to other trustworthy pages
- Spam pages mostly only link to other spam pages
- By only teleporting to known good pages, only the “good” partition accumulates significant trust



# Conflicting Interests

- The more seed pages there are, the most time and effort needed to curate
- The fewer seed pages there are, the less trust there is in the system
  - Threshold will need to be lower, more spam pages slip through
- Need to pick seed pages that are highly likely to point to “most” of the other good pages
  - Your Trust is roughly proportional to your link distance from a “good” seed page.

# Picking Good Seeds

## Pick the top k pages by Page Rank

- Assumption: even with link farms, bad pages won't be in the top k

## Use Trusted Domains

- Can't get a .edu, .mil, .gov domain just by buying one!
- But, in fact, you can be trustworthy without being the US Government or a US University

Scandalous claim!

# Bootstrapping Trust

- If your seed set is small, will miss a lot of trustworthy sites
    - Alternate View: You will only catch a small number of spam farmers
  - BUT: Anything that you find is probably pretty trustworthy
    - Candidates to be added to the trusted set.
1. Run PageRank
  2. Select top pages as seed (and verify trustworthiness)
  3. Run TrustRank
  4. Set threshold low enough to avoid false positives
  5. Remove spam pages from graph
  6. Goto Step 1

## Alternative – Spam Mass

$r_p$  = PageRank of Page  $p$

$r_p^+$  = PageRank of Page  $p$ , but random jumps only lead to **Trusted** pages

$r_p^- = r_p - r_p^+$  = Contribution of “low trust” pages to  $p$ 's rank

$S_p = \frac{r_p^-}{r_p}$  = Spam Mass (Fraction of  $p$ 's rank that's from “low trust” pages)

The higher your Spam Mass, the more likely you are to be spam



# More Variations on a Theme

Next Week – “Topic Identification”

A page might have high rank because it's important to a particular topic

Is it important to all topics?

ESPN might be popular for sports news. Should it show up on a search about Greek history?

# Topic-Specific Page Rank

Instead of the teleport set being all pages, it's all pages on a given topic T

Where do find this set?

DMOZ is dead, long live  
Curie

- DMOZ?
- High Page-Rank pages, classified using ML (Topic Classification) ?

In other words, the same thing as Multi-Source Personalized Page Rank

# Further Tweaks

---

The random-teleport set can be weighted without breaking anything

If a total of  $X$  mass teleports then

- Unweighted: Every Source Node gets  $X / M$
- Weighted: Source Node  $S_i$  gets  $w_i X / W$  (Where  $W$  is the sum of all  $w_i$ )

You can use regular Page Rank as the weights.