Northeastern

EECE5698
Parallel Processing for Data Analytics

Lecture 5: Lazy Evaluation, Resilience, & Persistence

Outline

- Lazy Evaluation & Resilience
- Persistence
- Example: PageRank Algorithm

Outline

- ❑ Lazy Evaluation & Resilience
- ❑ Persistence
- ❑ Example: PageRank Algorithm

Run This On Interpreter

```
rdd=sc.parallelize(range(1000))

rddWithTallies=rdd.map(lambda x: (x,1))

total,count = rddWithTallies.reduce(lambda x,y:
                                    (x[0]+y[0],x[1]+y[1]) )

print (1.*total/count)
```

Lazy

Evaluation!

Reduce triggers execution!

Lazy Evaluation

- Postpones evaluations until action (`reduce, collect, count`) requires their computation

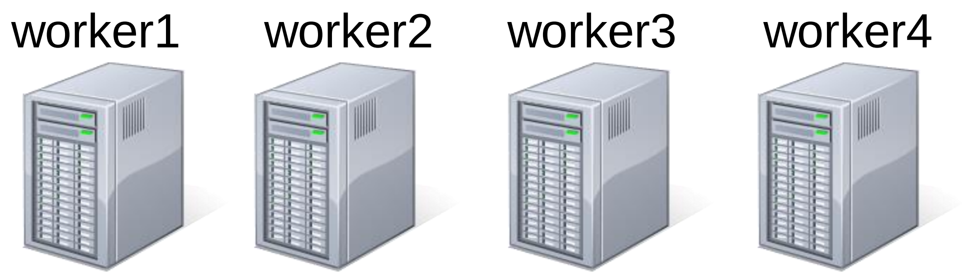
- Instead, spark builds a DAG (directed acyclic graph of rdd dependencies

DAG for this example

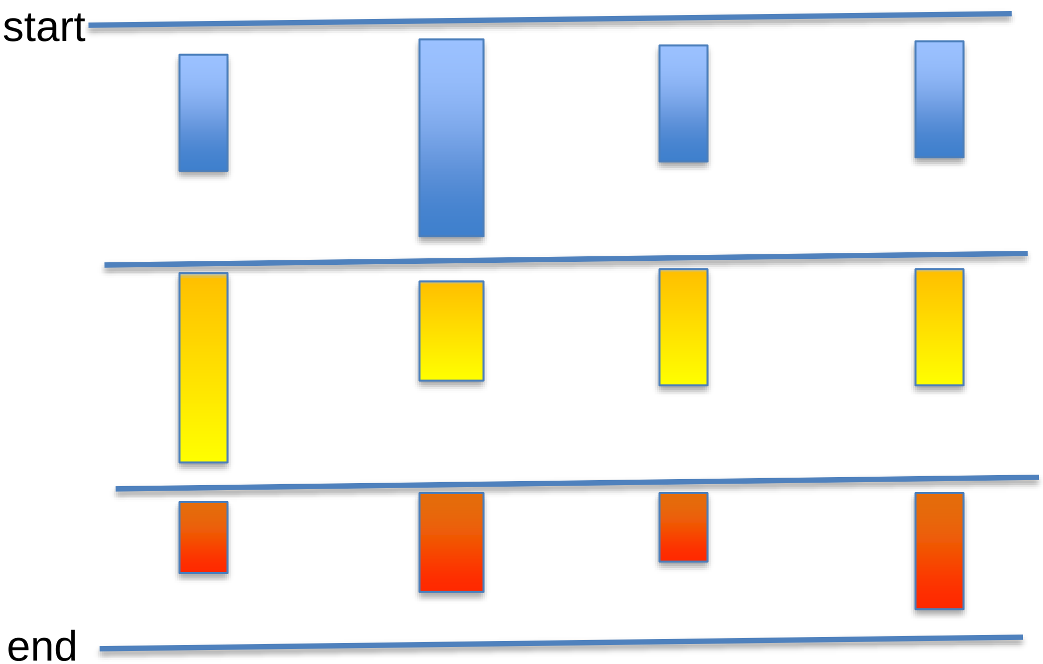
```
rdd=sc.parallelize(range(1000))
rddWithTallies=rdd.map(lambda x: (x,1))
total,count = rddWithTallies.reduce(lambda x,y:
                                  (x[0]+y[0],x[1]+y[1]) )
print(1.*total/count)
```

sc $\xrightarrow{\text{parallelize}}$ rdd $\xrightarrow{\text{map}}$ rddWithTallies $\xrightarrow{\text{reduce}}$ output returned to driver variables

Pipelining



Non-Pipelined Execution: Workers can be idle



sc

parallelize

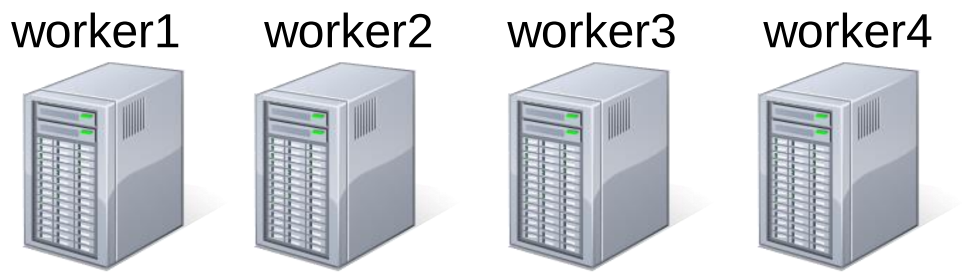
rdd

map

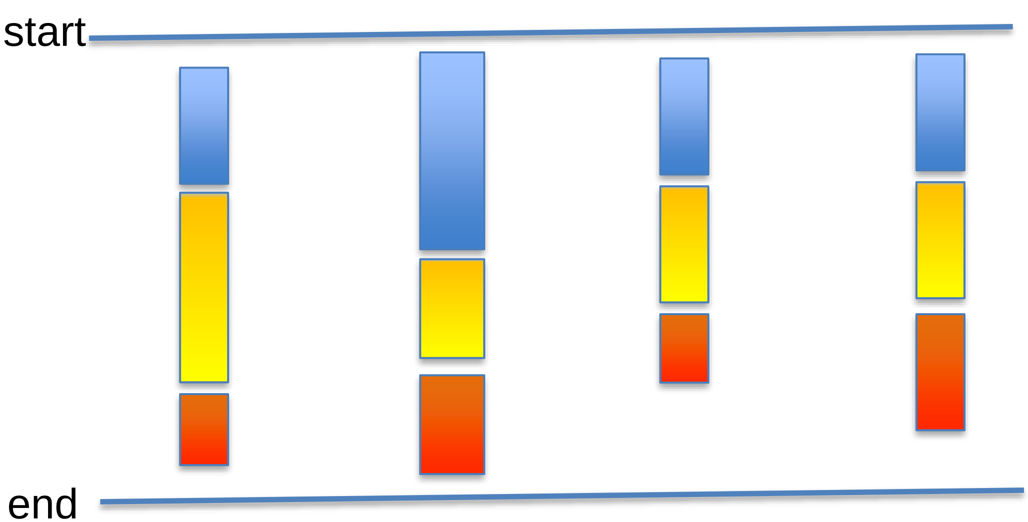
rddWithTallies

reduce

Pipelining



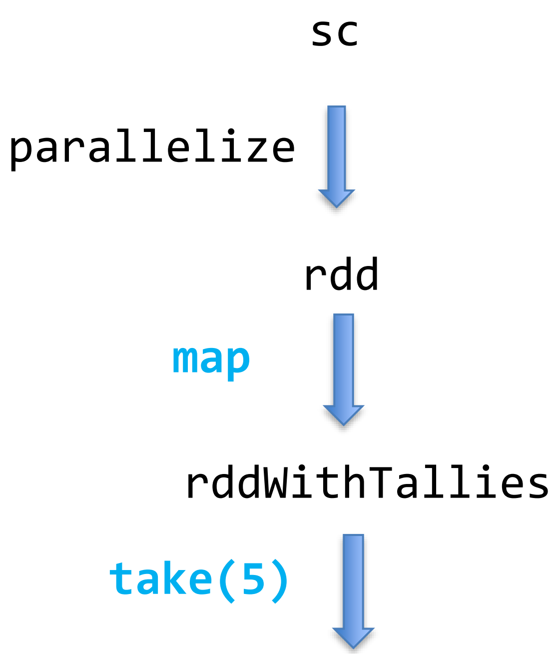
Pipelined Execution: A worker can start next op before other workers finish




```
sc
  parallelize ↓
rdd
  map ↓
rddWithTallies
  reduce ↓
```

Optimization

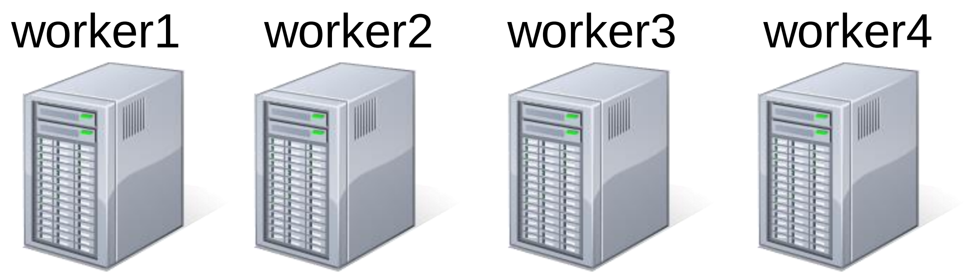
Pipelined Execution: A worker can start next op before other workers finish



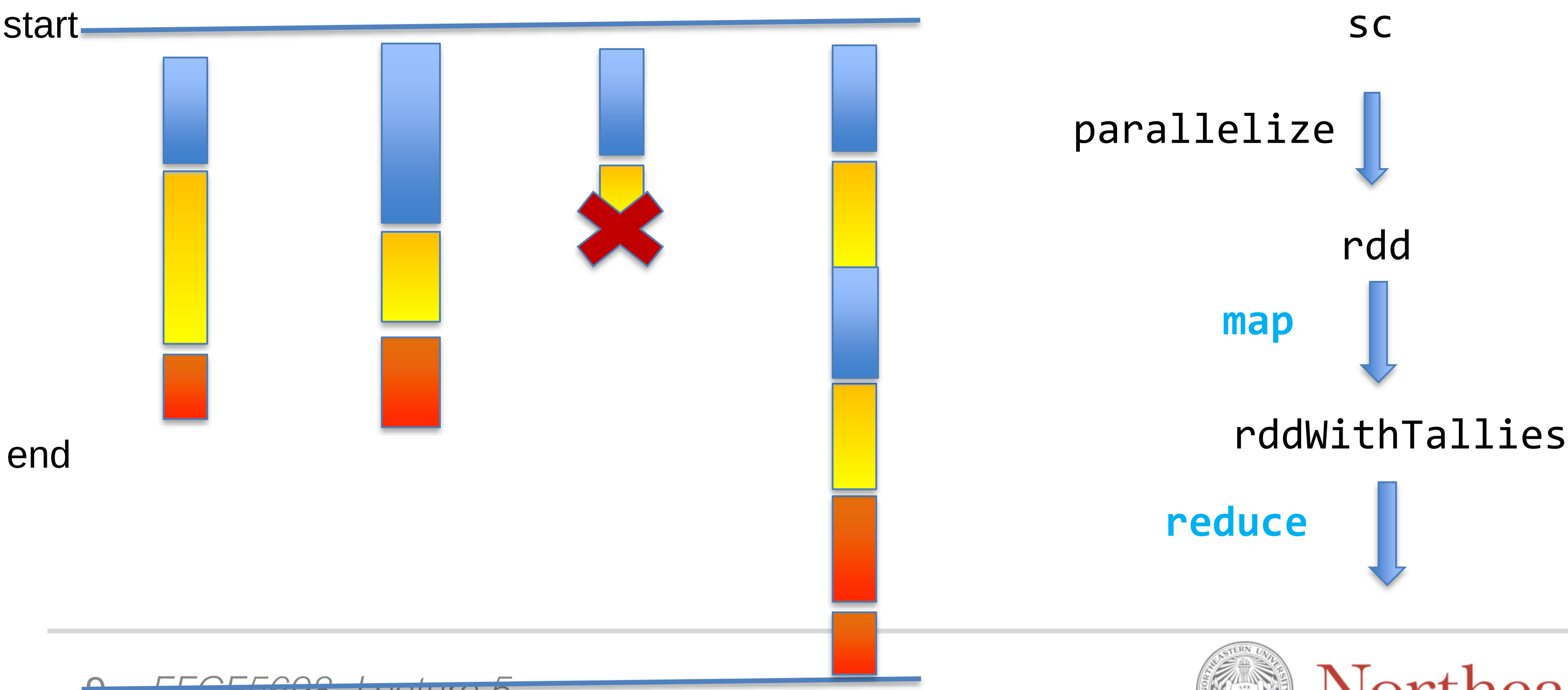
take(5)

Northeastern

Recovery From Failure



Pipelined Execution: A worker can start next op before other workers finish



RDDs are **NOT STORED, but DAG IS**

sc $\xrightarrow{\text{parallelize}}$ rdd $\xrightarrow{\text{map}}$ rddWithTallies $\xrightarrow{\text{reduce}}$ output returned to driver variables

```
rdd=sc.parallelize(range(1000))
rddWithTallies=rdd.map(lambda x: (x,1))
total,count = rddWithTallies.reduce(lambda x,y:
                                     (x[0]+y[0],x[1]+y[1]) )
print(1.*total/count)
(…)
rddWithTallies.collect()
```

Triggers recomputation of rdd, rddWithTallies **from scratch !!!**

What's The Point of Lazy Evaluation?

`sc` —parallelize→ `rdd` —**map**→ `rddWithTallies` —**reduce**→ output returned to driver variables

Pipelining

- ❑ Executions postponed can be **grouped together** and parallelized more efficiently

Optimization

- ❑ Only parts of RDD needed are computed

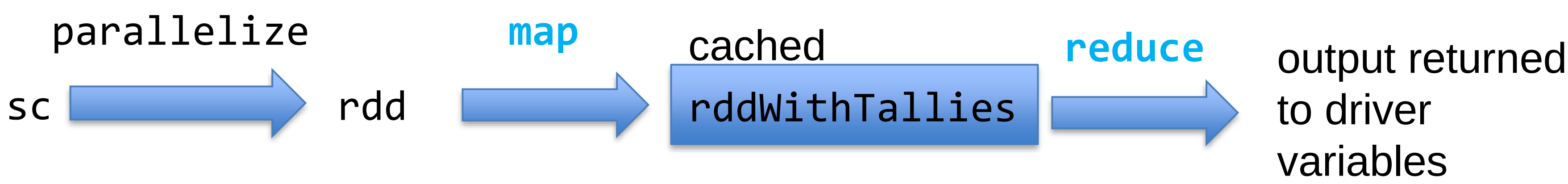
Resilience (the R in RDD)

- ❑ DAG allows **recovery** from crashed nodes/failed executions

Outline

- Lazy Evaluation & Resilience
- Persistence
- Example: PageRank Algorithm

Persistence



- RDDs that are reused can be stored in memory, by explicitly calling `cache()`

- If you expect to be using an RDD again later down the road, then cache it! This avoids recomputation

Persistence Example.

sc $\xrightarrow{\text{parallelize}}$ rdd $\xrightarrow{\text{map}}$ rddWithTallies $\xrightarrow{\text{reduce}}$ output returned to driver variables

```
rdd=sc.parallelize(range(1000))
rddWithTallies=rdd.map(lambda x: (x,1)).cache()
total,count = rddWithTallies.reduce(lambda x,y:
                                 (x[0]+y[0],x[1]+y[1]) )
print(1.*total/count)
(…)
rddWithTallies.collect()
```

⇐ rddWithTallies is cached, so no recomputation necessary

Caching Extremely Important for **Iterative Algorithms**

```
for i in range(50):
    #update rdd
    rdd=rdd.map(…).cache()
    #do some computation
```

If rdd is not cached, k-th iteration may repeat iterations k-1,k-2,…,1!

What If you Run Out of Memory?

```
for i in range(5000000000):
    #update rdd
    rdd=rdd.map(…).cache()
    #do some computation
```

- If cache is full, RDDs are evicted using **LRU policy**

- What if you try to access an RDD that has been evicted?
  - Resilience! DAG used to recompute
  - Option to spill excess RDDs in hard disk (see **`persist()`** instead of **`cache()`**)

What If you Run Out of Memory?

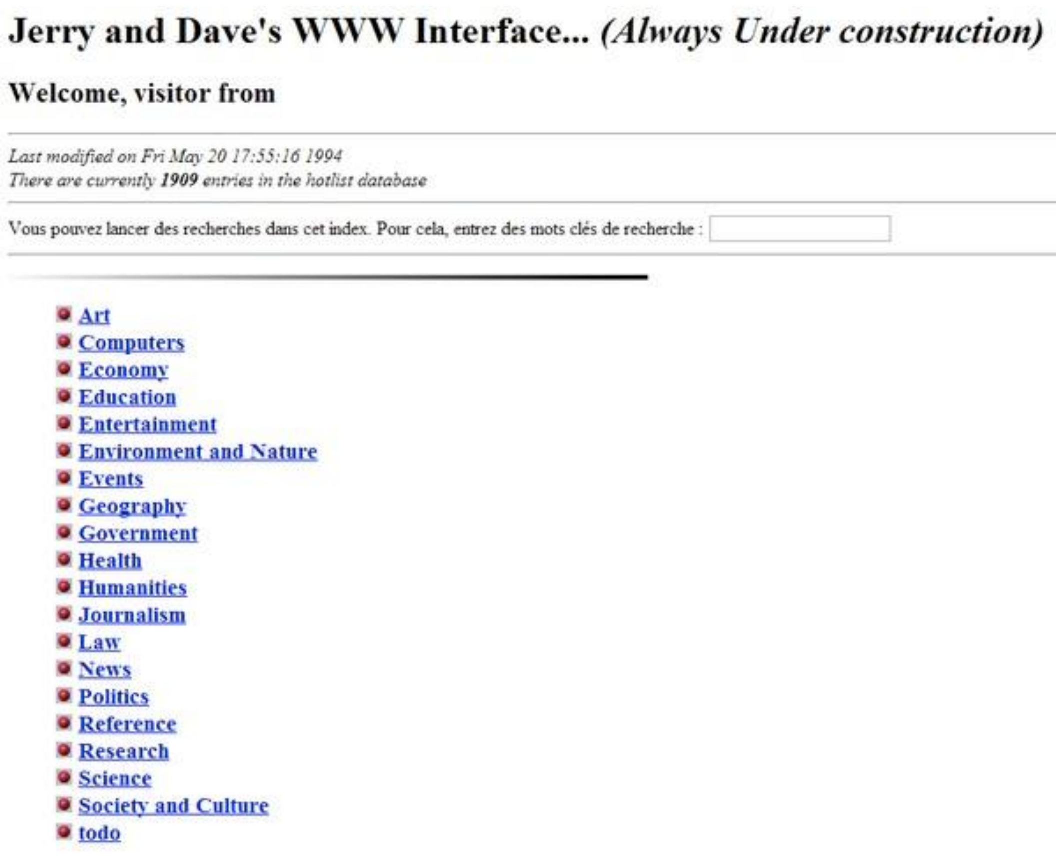
```
for i in range(1000000):
    oldrdd=rdd
    #update rdd
    rdd=rdd.map(…).cache()
    oldrdd.unpersist()
     #do some computation
```

- If you know that do you don't need rdd anymore, use **unpersist()**.

Outline

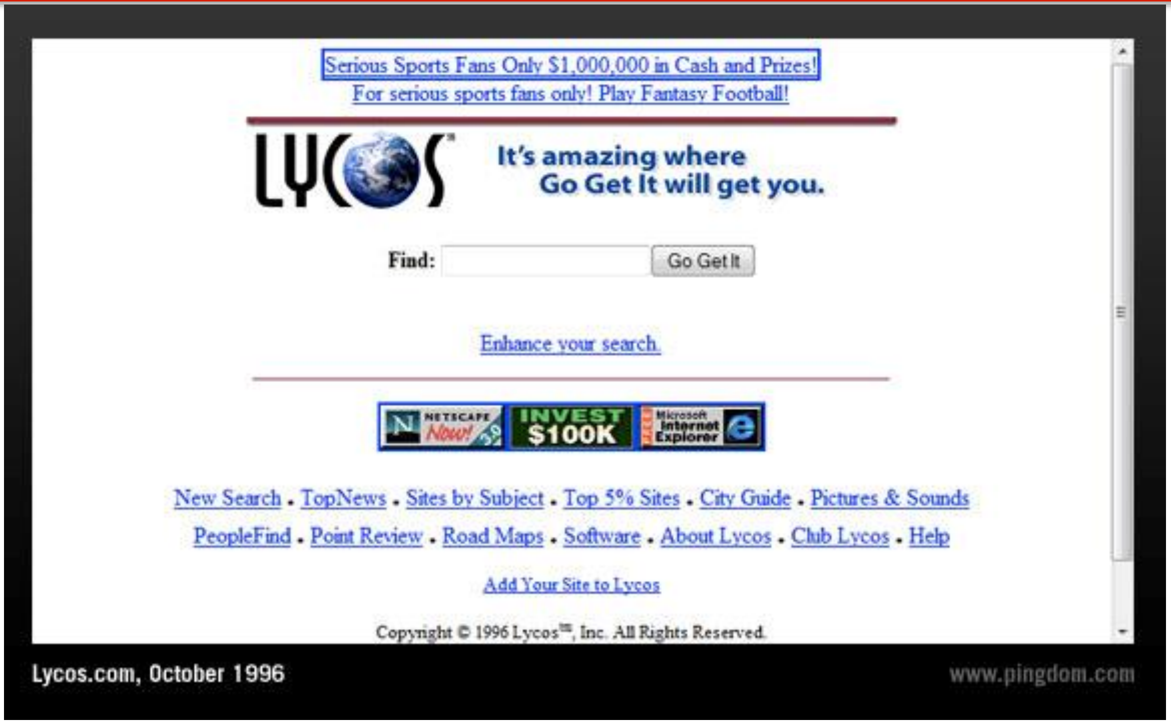
- ❑Lazy Evaluation & Resilience
- ❑Persistence
- ❑Example: PageRank Algorithm

Early Days of The Web: Portals



Northeastern

Early Search Engines



Main Idea:

- ❑ Crawl web and store html files
- ❑ Match query text to html file text

Challenge: Identify Important Websites!!!!

Northeastern

PageRank

[The **PageRank** citation ranking: Bringing order to the web.]
L Page, S Brin, R Motwani, T Winograd – 1999 Cited by 10184

The PageRank Citation Ranking: Bringing Order to the Web

January 29, 1998

Abstract

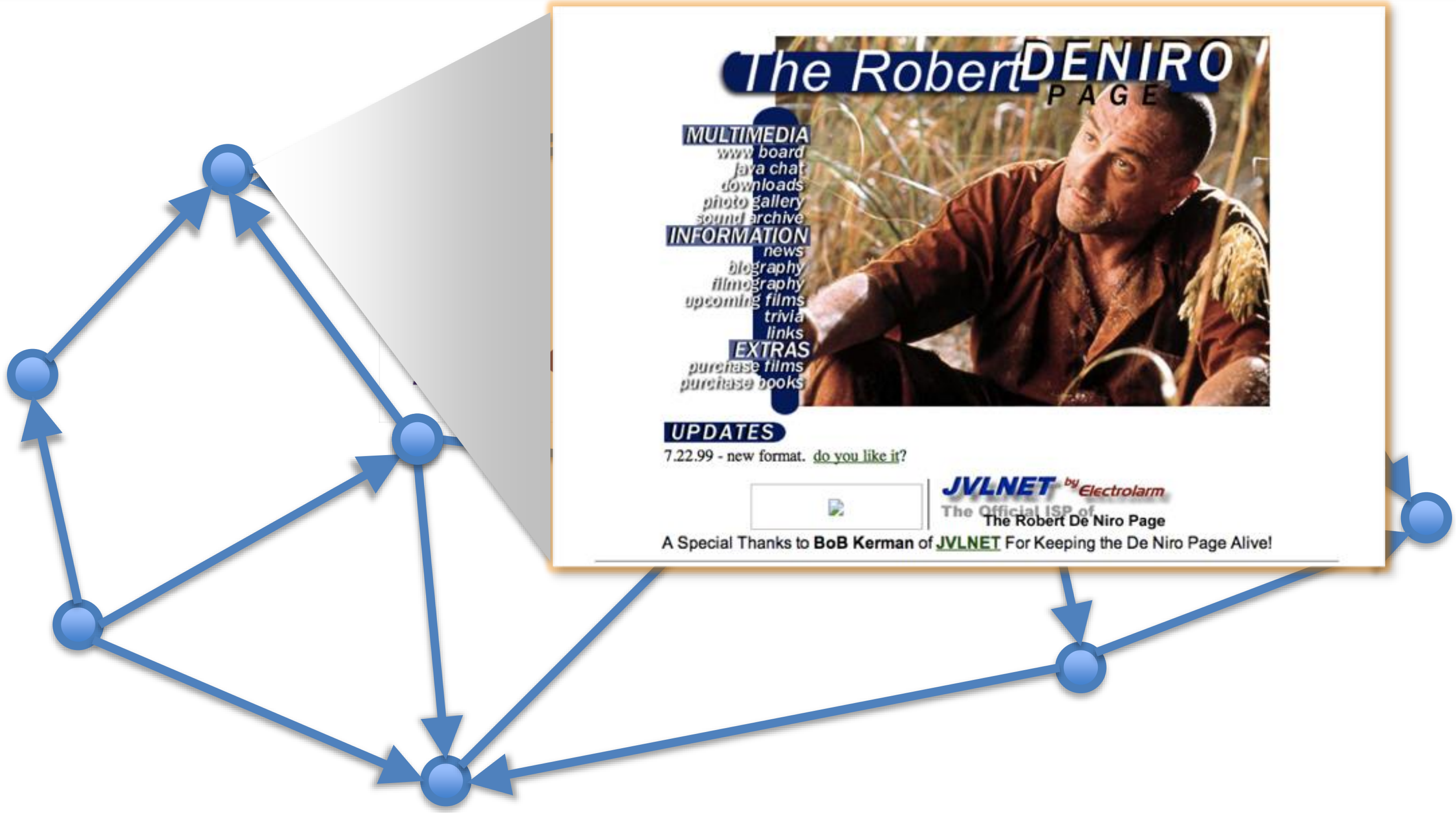
The importance of a Web page is an inherently subjective matter, which depends on the readers interests, knowledge and attitudes. But there is still much that can be said objectively about the relative importance of Web pages. This paper describes PageRank, a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them.

We compare PageRank to an idealized random Web surfer. We show how to efficiently compute PageRank for large numbers of pages. And, we show how to apply PageRank to search and to user navigation.

1 Introduction and Motivation

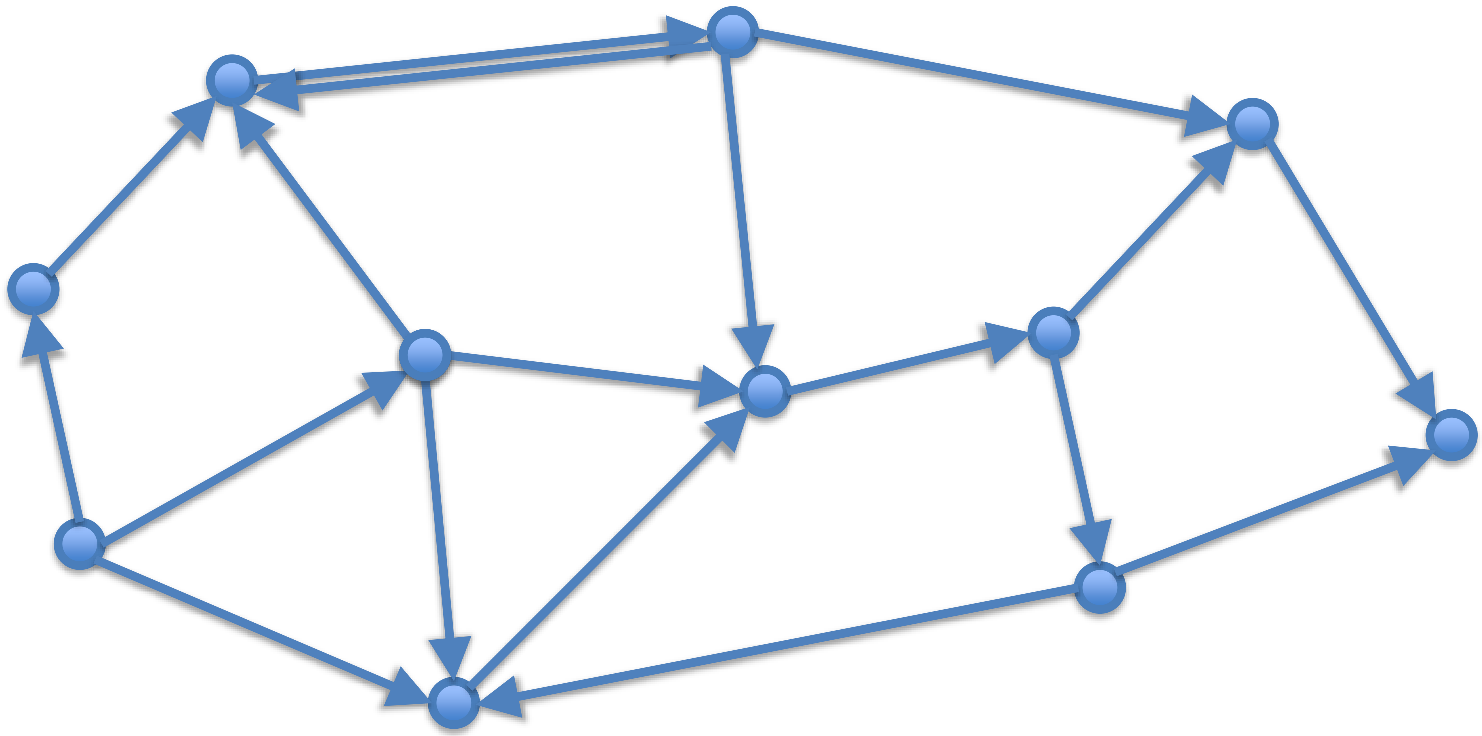
The World Wide Web creates many new challenges for information retrieval. It is very large and heterogeneous. Current estimates are that there are over 150 million web pages with a doubling life of less than one year. More importantly, the web pages are extremely diverse, ranging from "What is Joe having for lunch today?" to journals about information retrieval. In addition to these major challenges, search engines on the Web must also contend with inexperienced users and pages

A Graph of the World Wide Web (WWW)



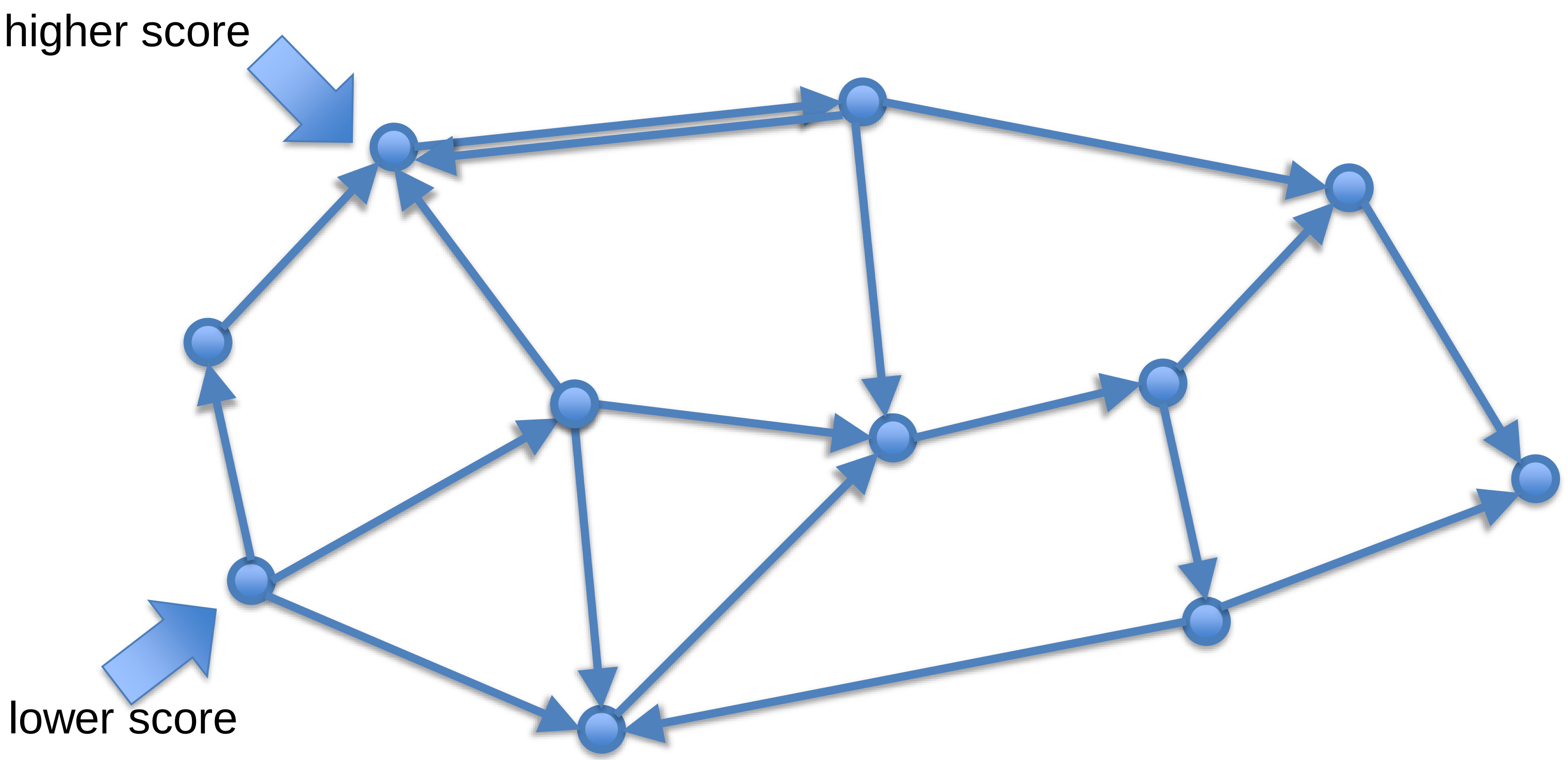
Nodes: Websites (N in total)
Edges: An edge from A to B exists if A contains a hyperlink to B

PageRank Score



Each webpage has an **importance score** between 0 and 1

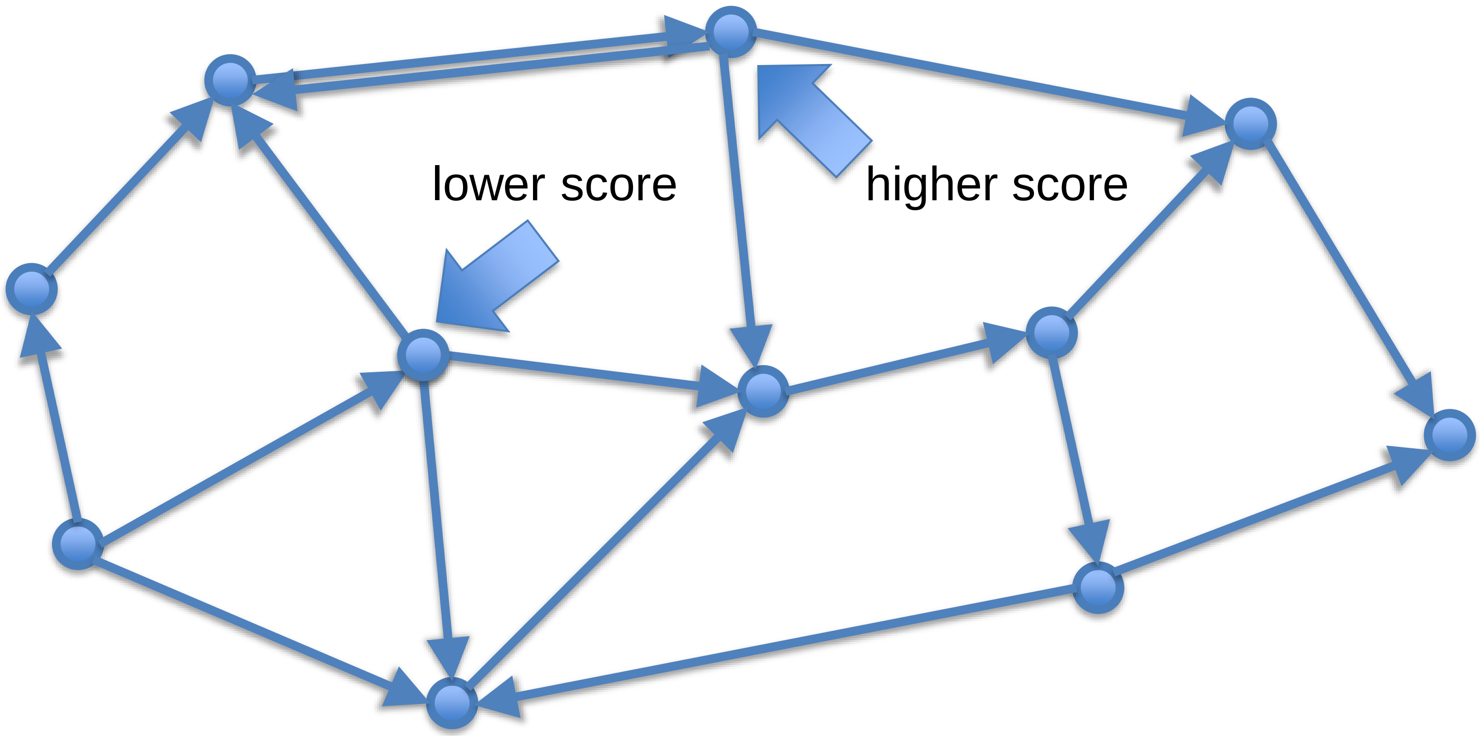
PageRank Score



Property 1:
The importance score of a website is **higher** if a lot of websites **link to it.**

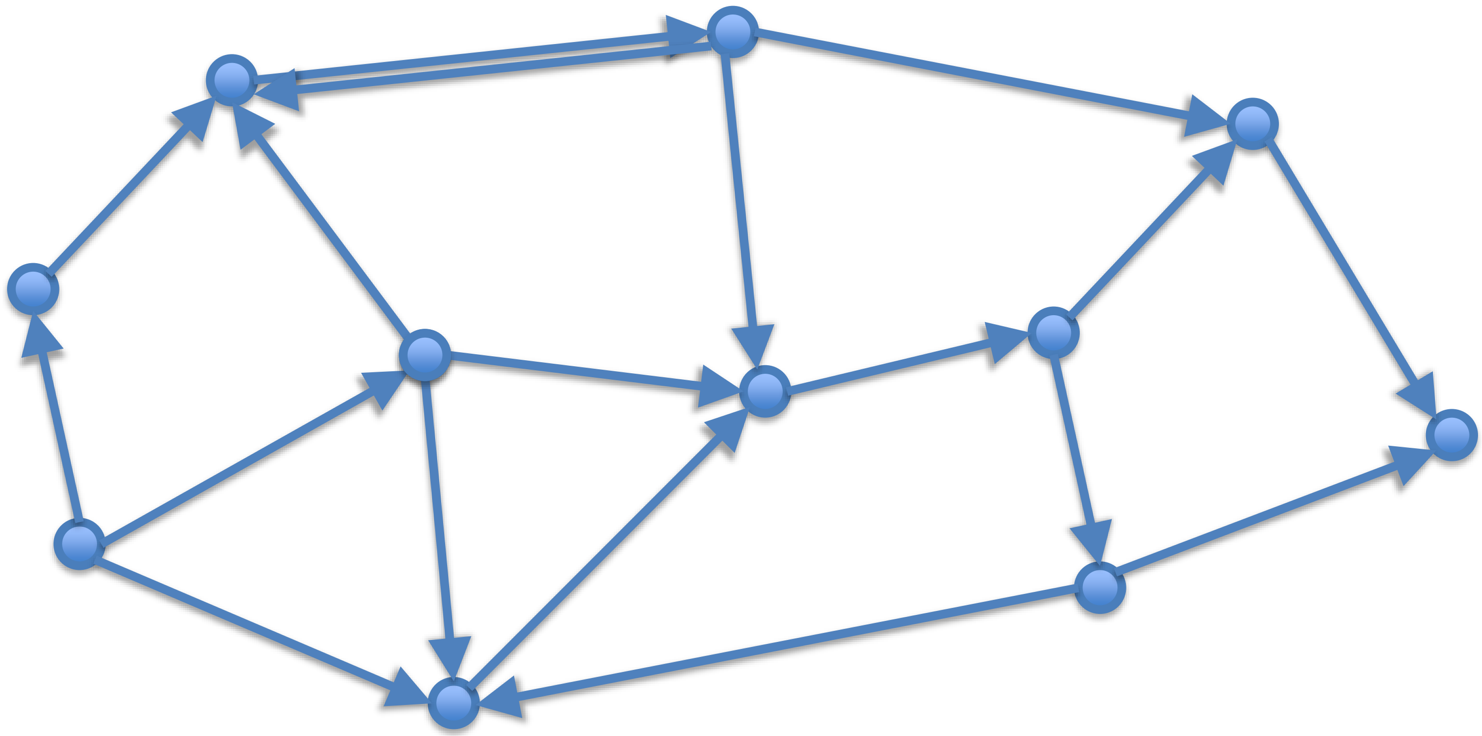
Northeastern

PageRank Score



Property 2:
The importance score of a website is higher if **important websites link to it**

PageRank Algorithm

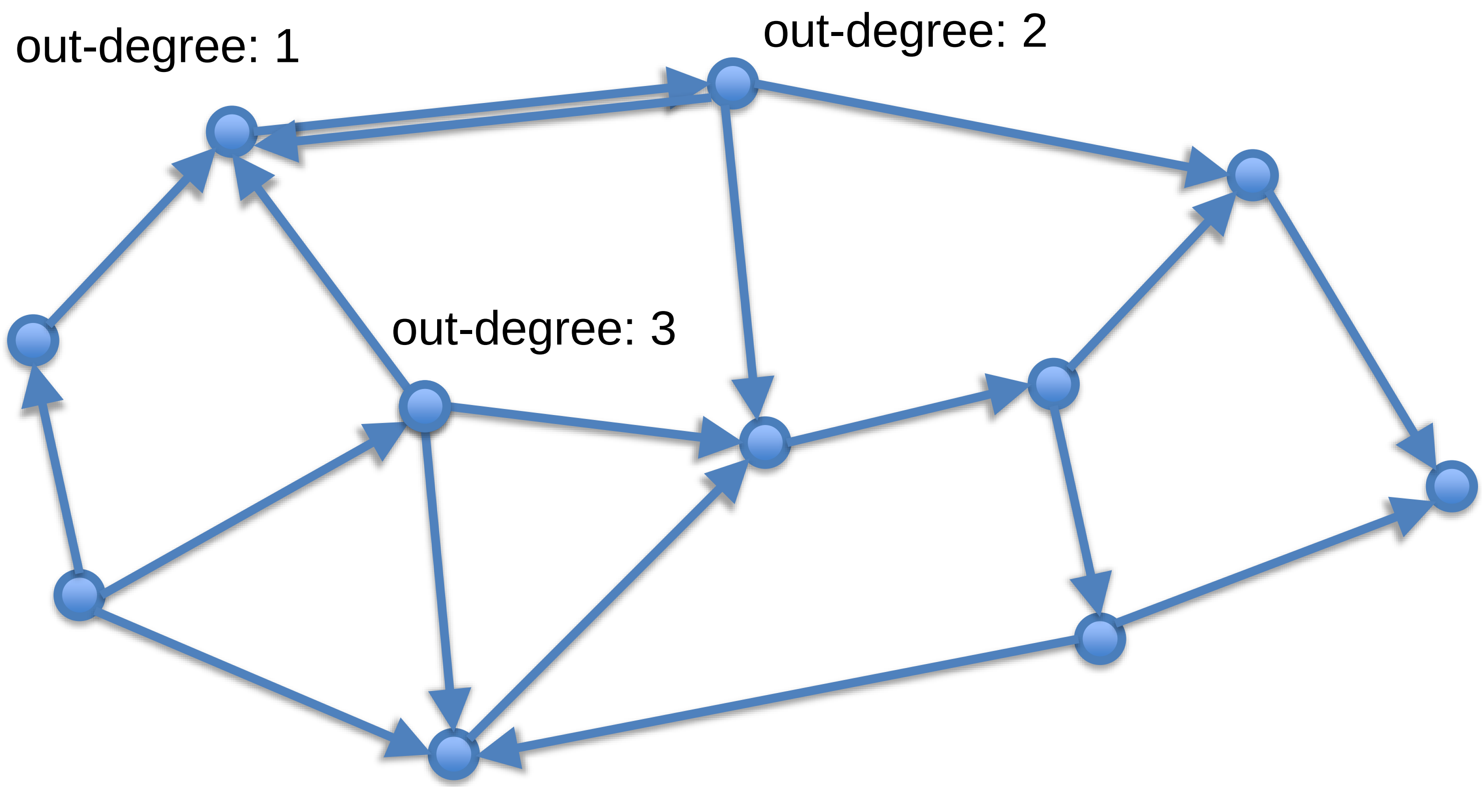


All websites start with equal score. If N websites in total, then:

$$s_w = \frac{1}{N} \qquad \text{for every website } w$$

Northeastern

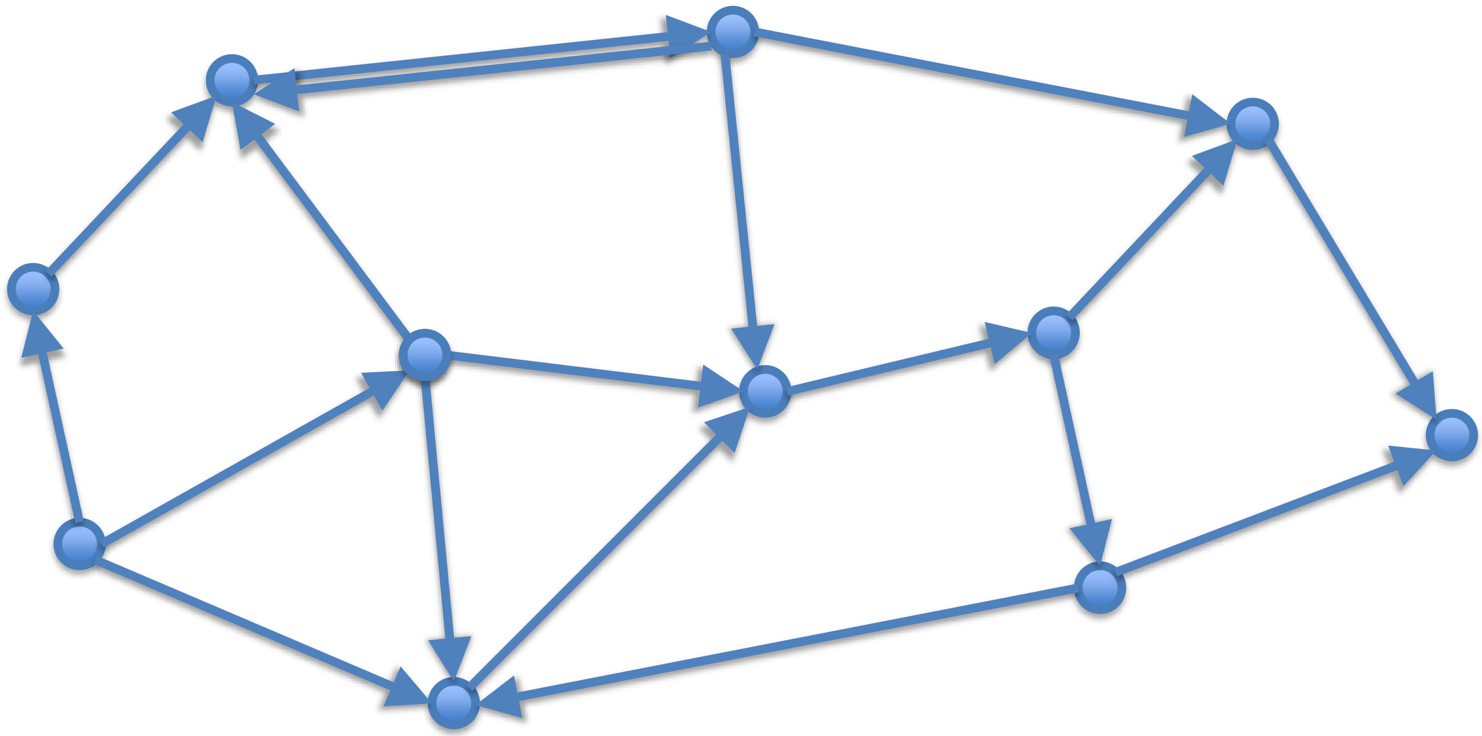
PageRank Algorithm



Let d_w be the **out-degree** of a website (i.e., # of links in website)

$$d_w = \#\ \text{websites}\ w'\ \text{s.t.}\ w \rightarrow w'$$

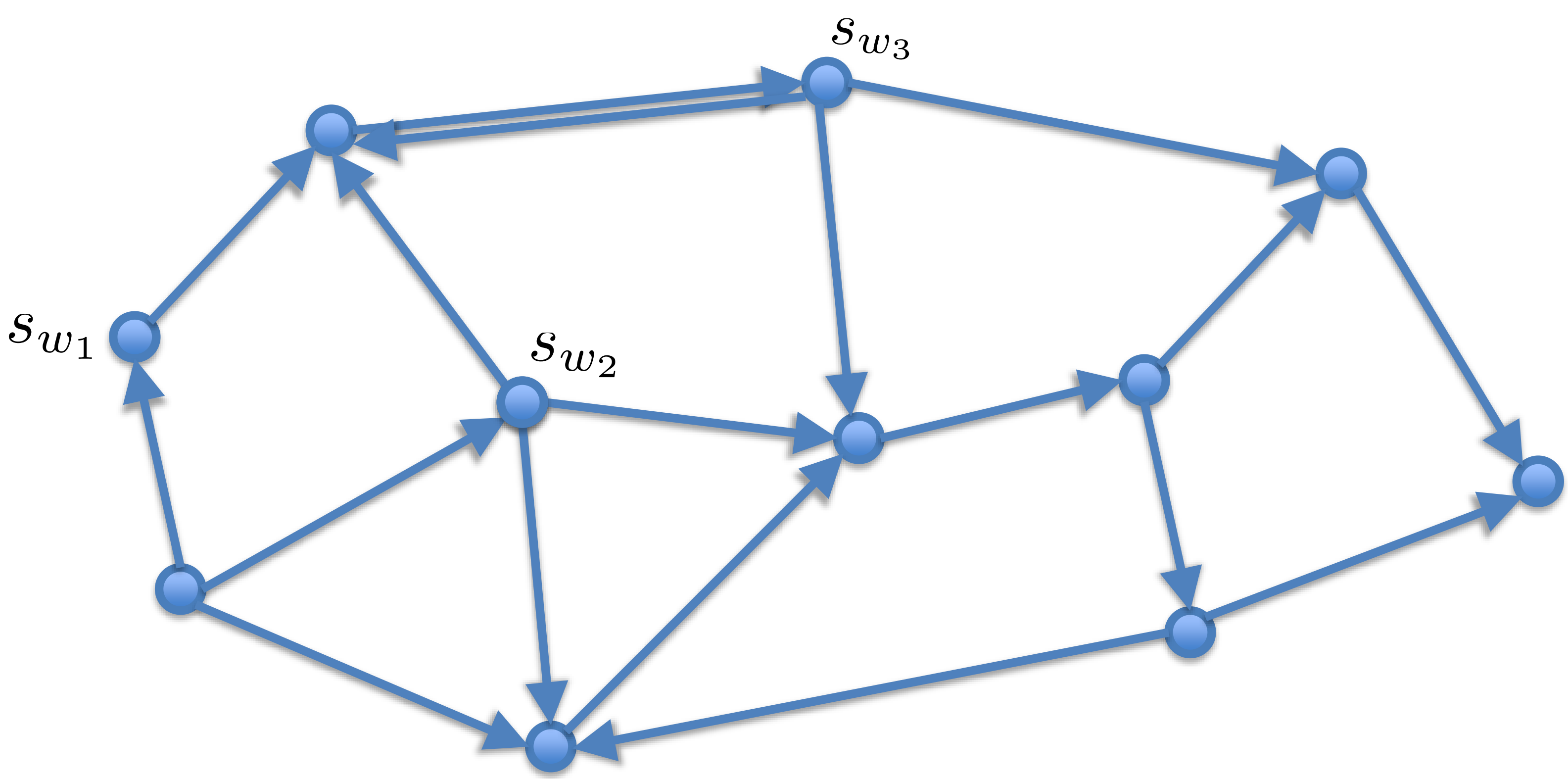
PageRank Algorithm



The score of every website w gets updated as follows:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w':w' \to w} \frac{s_{w'}}{d_{w'}}, \text{ where } \gamma = 0.15$$

PageRank Algorithm

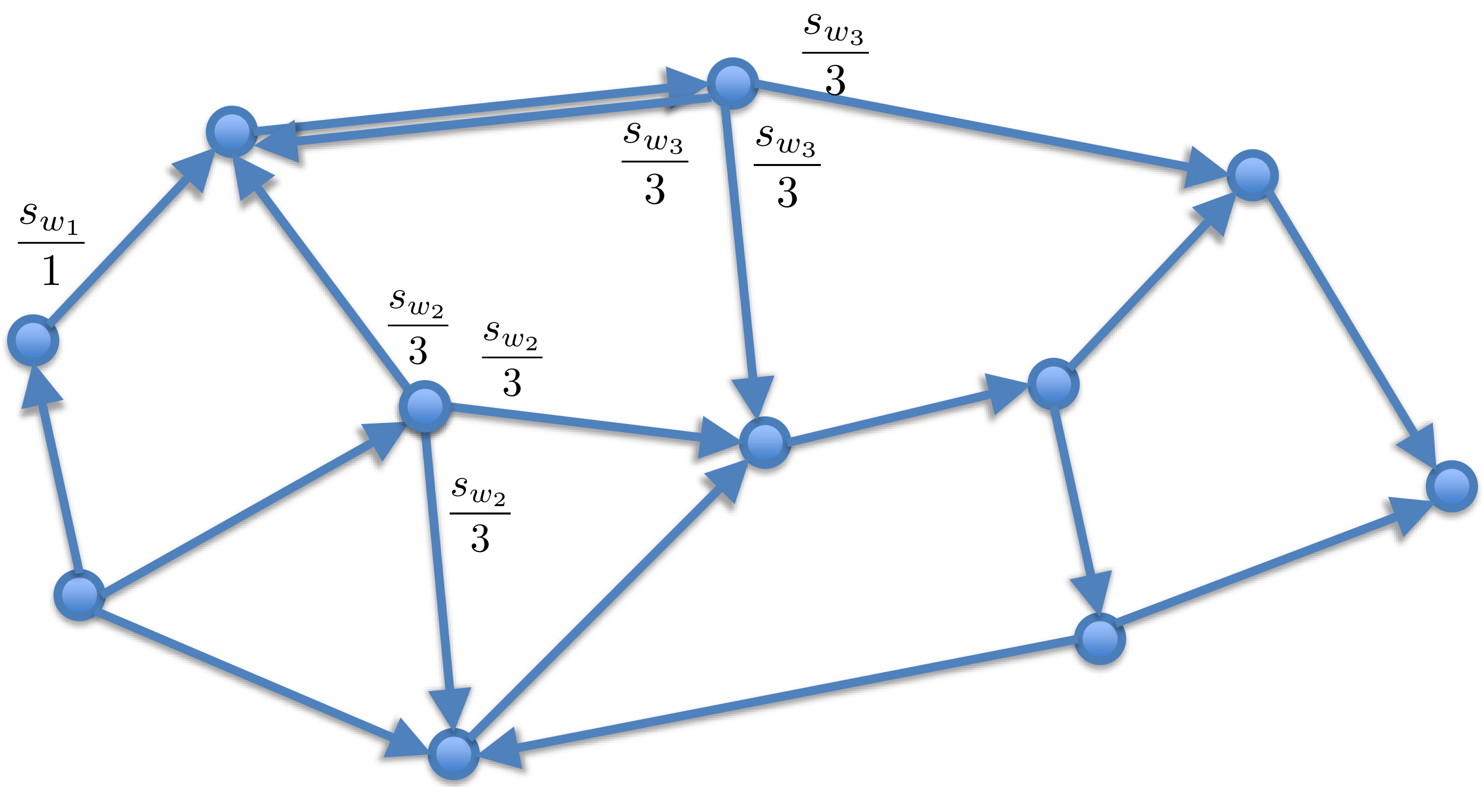


The score of every website w gets updated as follows:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w' : w' \to w} \frac{s_{w'}}{d_{w'}}$$, where $\gamma = 0.15$

Northeastern

PageRank Algorithm

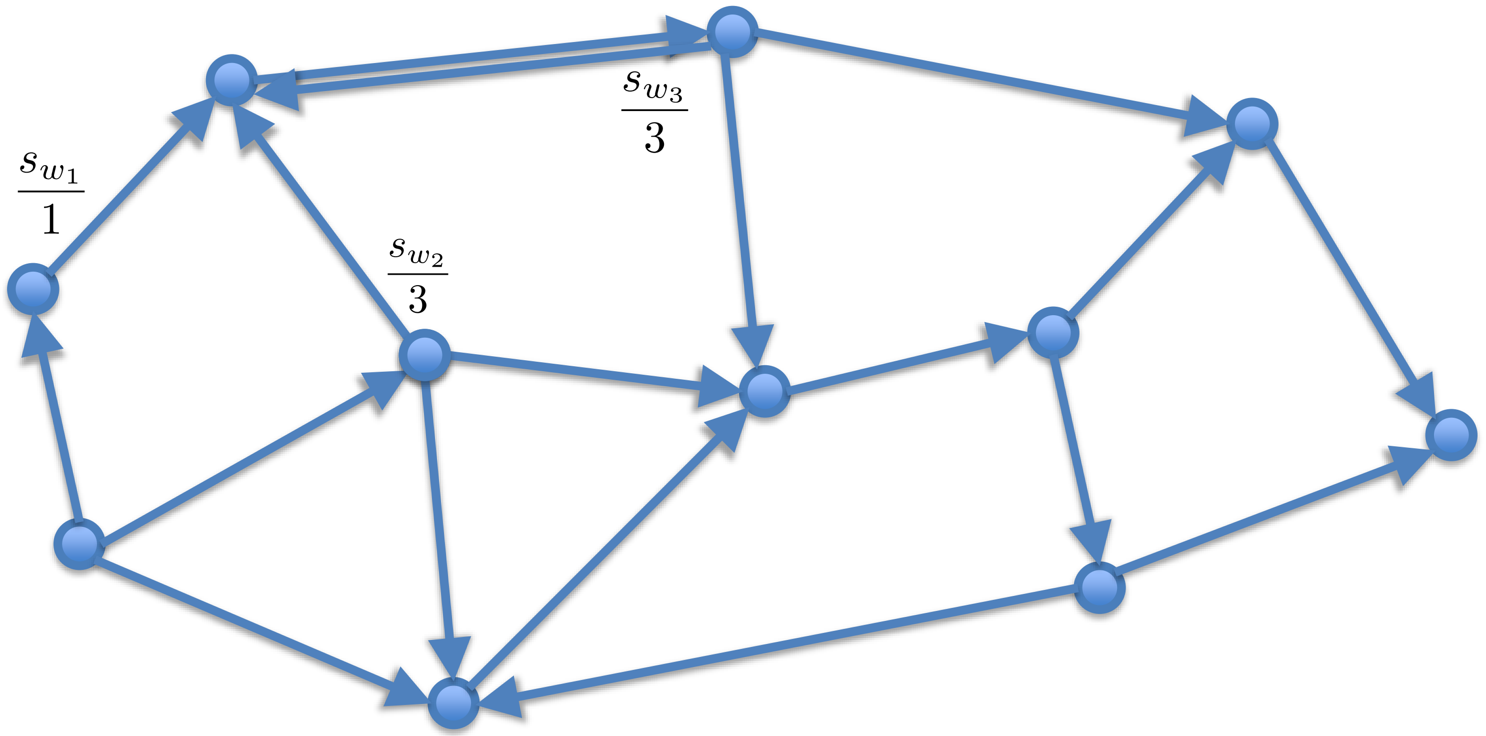


The score of every website w gets updated as follows:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w' : w' \to w} \frac{s_{w'}}{d_{w'}}$$, where $\gamma = 0.15$

Northeastern

PageRank Algorithm

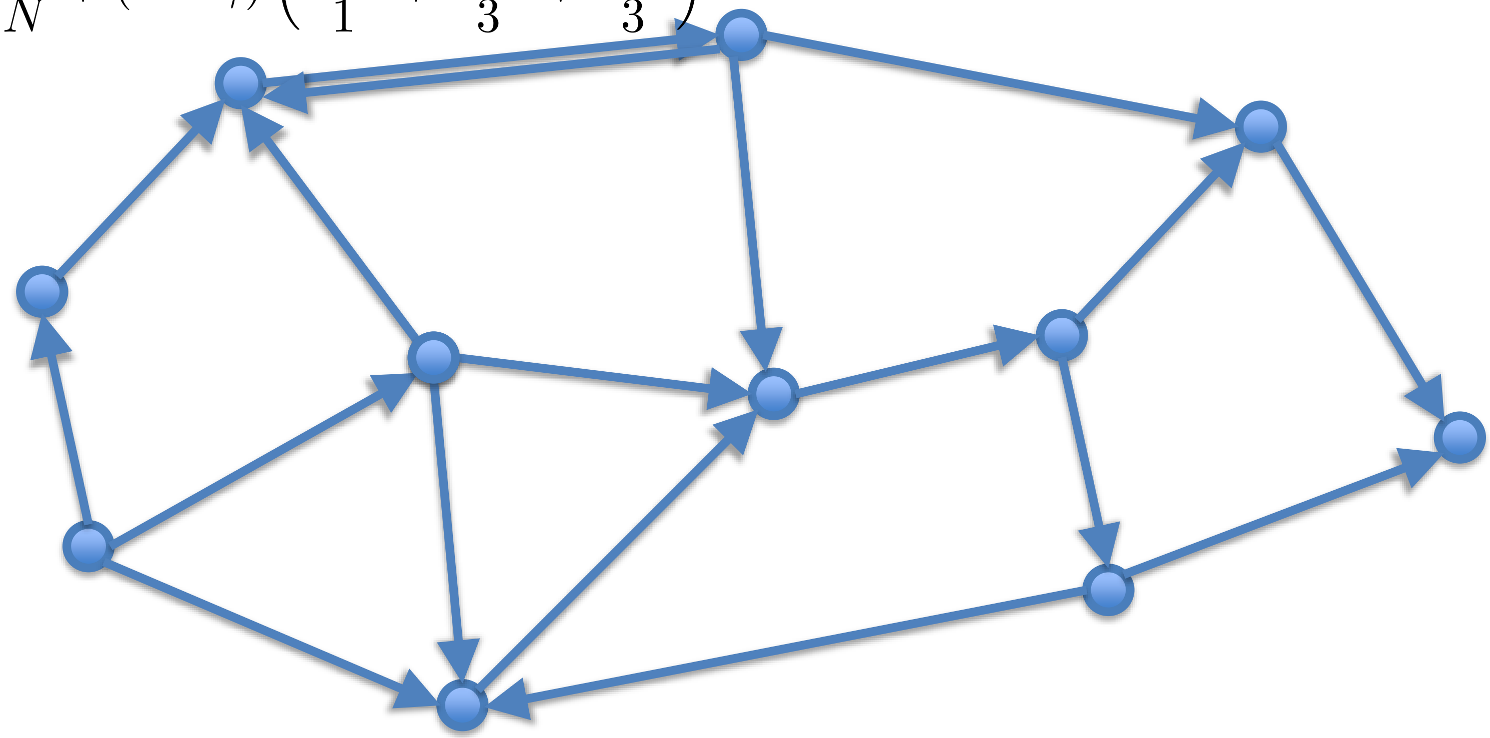


The score of every website w gets updated as follows:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w':w' \to w} \frac{s_{w'}}{d_{w'}}$$, where $\gamma = 0.15$

PageRank Algorithm

$$s_w = \gamma \frac{1}{N} + (1-\gamma)\left(\frac{s_{w_1}}{1} + \frac{s_{w_2}}{3} + \frac{s_{w_3}}{3}\right)$$

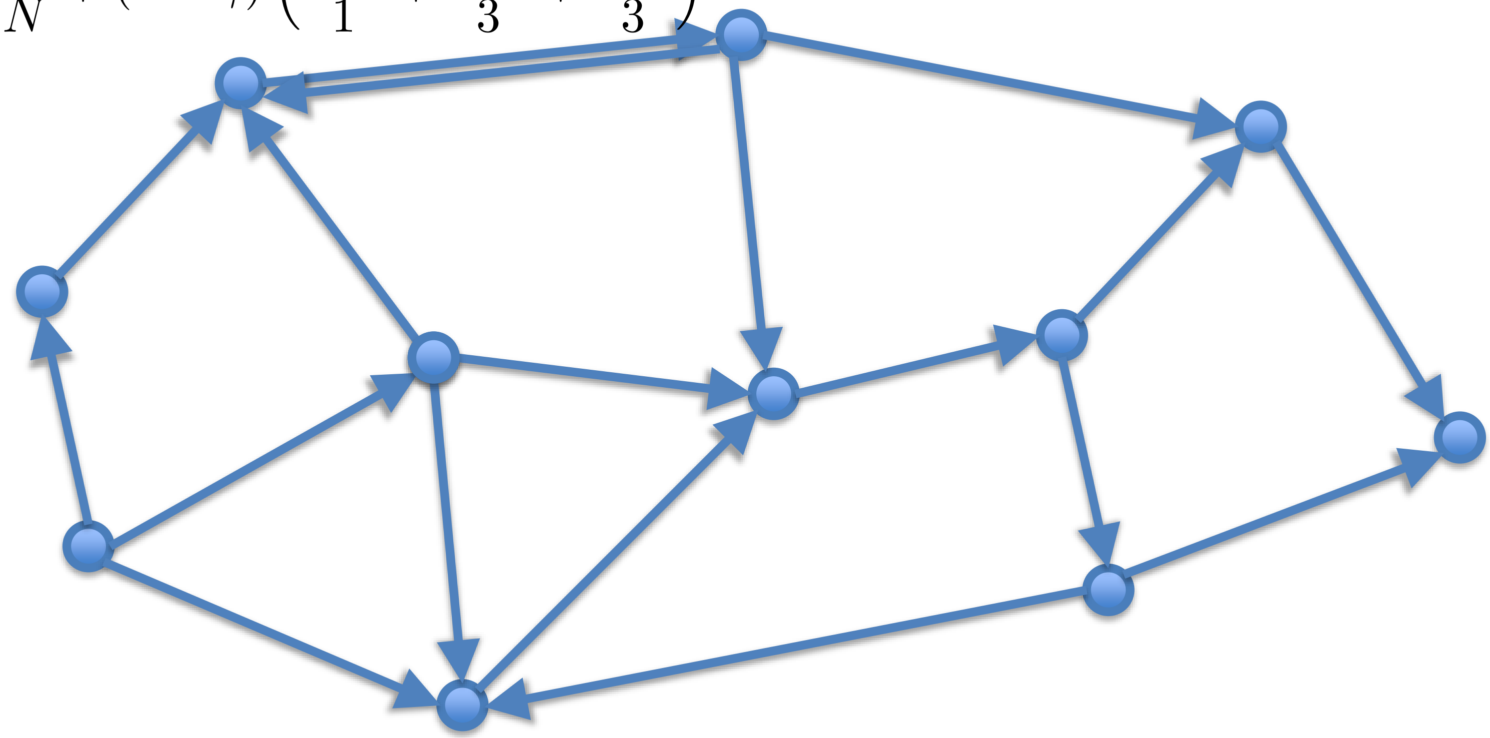


The score of every website w gets updated as follows:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w' : w' \to w} \frac{s_{w'}}{d_{w'}}$$, where $\gamma = 0.15$

PageRank Algorithm

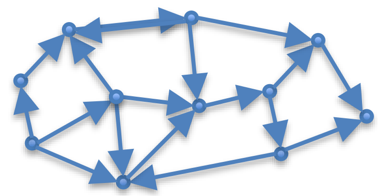
$$s_w = \gamma \frac{1}{N} + (1-\gamma)\left(\frac{s_{w_1}}{1} + \frac{s_{w_2}}{3} + \frac{s_{w_3}}{3}\right)$$



The algorithm is repeated over multiple iterations, until convergence.

PageRank Algorithm: Pseudo-Code

Input: WWW graph G



Initialization: Set $s_w = \frac{1}{N}$ for every website w

Main Loop: For $\gamma = 0.15$ repeat:

$$s_w = \gamma \cdot \frac{1}{N} + (1-\gamma) \cdot \sum_{w' : w' \to w} \frac{s_{w'}}{d_{w'}}$$

until convergence

Northeastern