Northeastern University

Google's PageRank Computation

How to build a framework capable of executing elastic computations across tons of computers

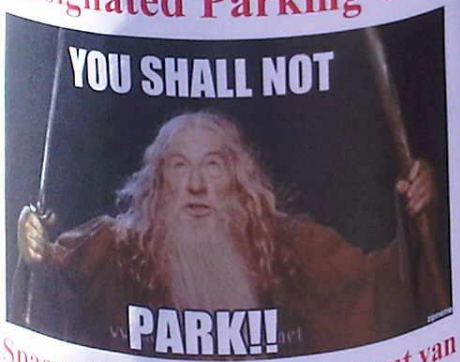
Google's PageRank

- Initially developed at Stanford University by Google founders, Larry Page and Sergey Brin, in 1995 to rank any type of recursive "documents" using MapReduce
- Sergey Brin and Lawrence Page (1998). *"The anatomy of a large-scale hypertextual Web search engine"* Proceedings of the seventh international conference on World Wide Web 7: 107-117
- Led to a functional prototype named.. you know... Google, in 1998
- Still provides the basis for all of Google's web search tools


Google


EXPECTING
MOTHER

Designated Parking ☺



Spaces Reserved for Haircut van

06/11

Thank you

Google

Security:650-253-5353

$$

- The name "PageRank" is a trademark of Google, and the PageRank process has been patented (*U.S. Patent 6,285,999*).
- However, the patent was assigned to Stanford University and not to Google..
  - Google has exclusive license rights on the patent from Stanford University
  - The university received 1.8 million shares of Google in exchange for use of the patent
  - The shares were sold in 2005 for $336 million

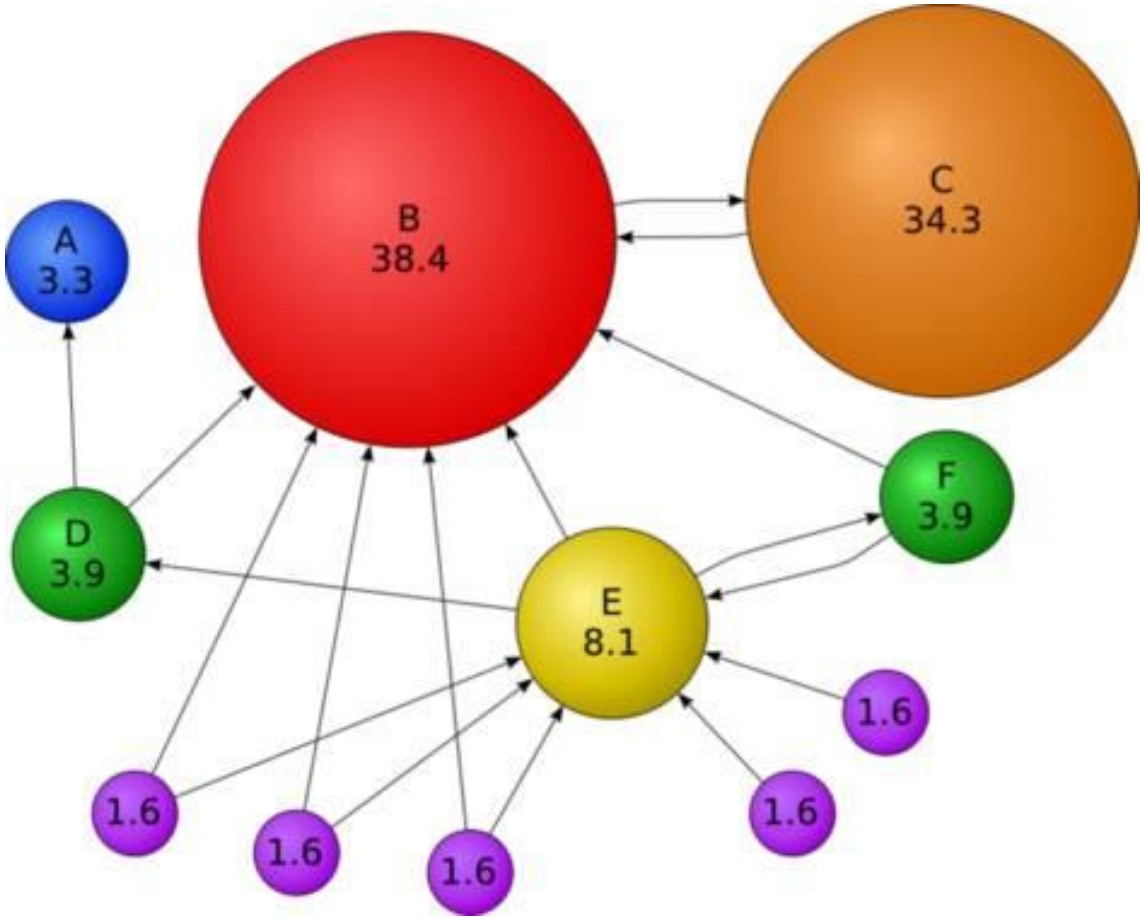
*Larry Page*Rank

- PageRank was developed at Stanford University by Larry Page (hence the name *Page*-Rank) and later Sergey Brin as part of a research project about a new kind of search engine
- Google interprets a link from page A to page B as a *vote*, by page A, for page B
- The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links")
- A page that is linked to by many pages with high PageRank receives a high rank itself

Votes are a bit “skewed”

- PageRank relies on the uniquely democratic nature of the web by using its vast link structure as an indicator of an individual page's value
- But Google looks at more than the sheer volume of votes, or links a page receives
  - It also analyzes the page that casts the vote
  - Votes cast by pages that are themselves "important" weigh more heavily and help to make other pages "important"

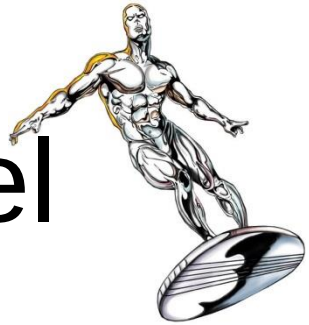
PageRanks for a simple network



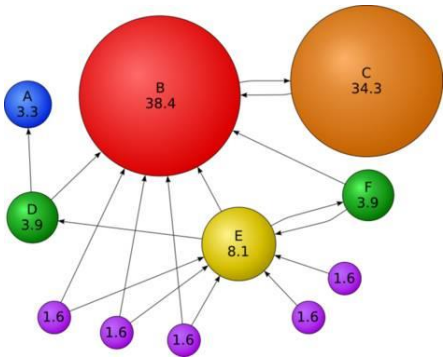
- Page C has a higher PageRank than Page E, *even though it has fewer links to it*; The link it has is of higher value
- A web surfer who chooses a random link on every page is going to be on Page E for 8.1% of the time

(PageRanks reported by Google are rescaled logarithmically)

Surfer Model



- There is a 15% likelihood that the surfer jumps to a random page on the whole web
  - This corresponds to a damping factor of 85%
- Without damping, all Web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero
- This corresponds roughly to how we browse the Web: by following URLs or by typing at the address bar



JCPenney

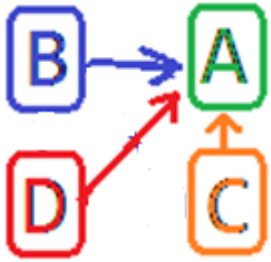
- Various strategies to manipulate PageRank have been employed in concerted efforts to improve search results rankings and monetize advertising links
- These strategies have severely impacted the reliability of the PageRank concept, which seeks to determine which documents are actually highly valued by the Web community
  - Google runs big "spam" team to uphold PageRank reliability
- NYT, Sunday February 13 2011: *"The Dirty Little Secrets of Search"*

Influences

- PageRank has been influenced by
  - Citation analysis, early developed by Eugene Garfield in the 1950s at the University of Pennsylvania
    - The first example of automated citation indexing was CiteSeer, later to be followed by Google Scholar
  - Hyper Search, developed by Massimo Marchiori at the University of Padua
  - In the same year PageRank was introduced (1998), Jon Kleinberg published his important work on HITS.
- Google's founders cite *Garfield*, *Marchiori*, and *Kleinberg* in their original paper

WWW = A + B + C + D

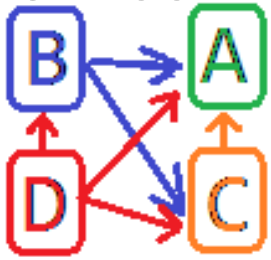
- Assume a small universe of four web pages: **A**, **B**, **C** and **D**. The initial approximation of PageRank would be evenly divided between these four documents
  - Hence, each document would begin with an estimated PageRank of 0.25



- *If pages **B**, **C**, and **D** each only link to **A***, they would each confer 0.25 PageRank to **A**
- All PageRank **PR( )** in this simplistic system would thus gather to **A** because all links would point to **A**
  - This is 0.75
  - Then B, C, and D share the remaining PageRank of 0.25

WWW = A + B + C + D

- Suppose that page **B** has a link to pages **A** and **C** **but not D**, while page **D** has links to **all three** pages, and C **only** to A
  - The *value of each link/vote is divided among all the outbound links on a page*
  - Thus, page **B** (starting worth=0.25) gives a vote worth 0.125 to page **A** and a vote worth 0.125 to page **C**
  - While only one third of **D**'s PageRank is counted for A's PageRank (approximately 0.083)



$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1} + \frac{PR(D)}{3}.$$

WWW

- In the general case, the PageRank value for any page **u** can be expressed as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

- The PageRank value for a page **u** is dependent on the PageRank values for each page **v** out of the set $\mathbf{B_u}$ (this set contains all pages linking to page **u**), divided by the number $L(v)$ of links from page **v**
- The sum of all PageRanks is always 1

Wait.. Damping Factor

- PageRank theory holds that even an imaginary surfer who is randomly following links will eventually stop clicking, and instead will enter a new completely unrelated URL on his browser's address bar
- The probability, at any step, that the person will continue following hyperlinks is a damping factor *d*
- Various studies have tested different damping factors, but it is generally assumed the damping factor hovers around 0.85
- So, we re-evalute: If *N* is the number of documents in the collection, then:

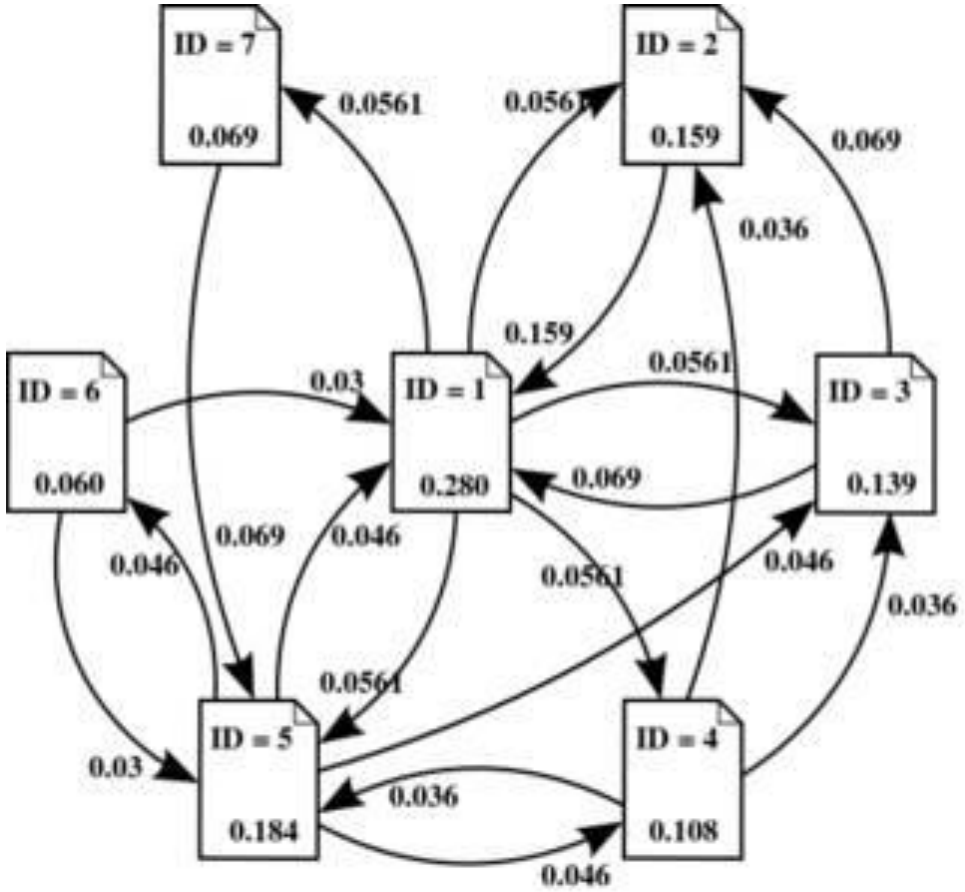
$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right)$$

User starts at page A, and stays there

User starts at a page other than A, and surfs to A by following hyperlinks

Easier to verify than to calculate.

BTW, that's the characteristic of NP-Complete..



Ok, but how to calculate?

- Google recalculates PageRank scores each time it crawls the Web and rebuilds its planetary index
  - Google *Hummingbird* was one version (https://en.wikipedia.org/wiki/Google_Hummingbird)
  - PageRank is only one of over 200 major "ingredients" that go into recipe http://searchengineland.com/seotable
  - More info here https://moz.com/google-algorithm-change
- PageRank formula uses a model of a *random surfer* who additionally gets bored after several clicks and switches to a random page (damping factor)
  - If a page has no links to other pages, it becomes a sink and therefore terminates the random surfing process. If the random surfer arrives at a sink page, it picks another URL at random and continues surfing
  - The PageRank value of a page reflects the probability that the surfer will land on that page by clicking on a link
  - Search engines reward pages with the right combination of ranking factors or "signals."
- It can be understood as a ***Markov chain*** process in which the states are pages and the transitions are all probable (with different probabilities) and are the links between pages

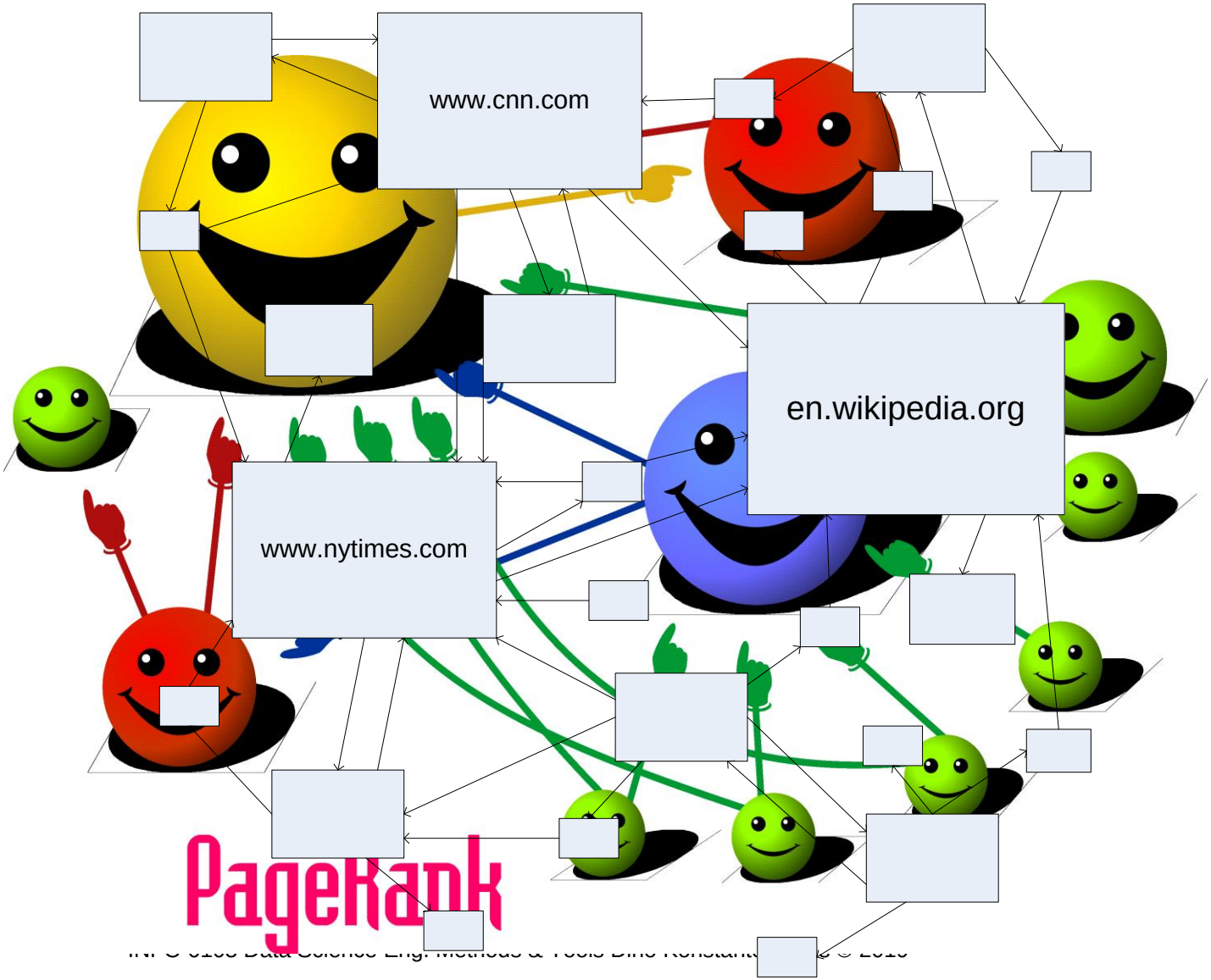
SEO


On-The-Page Factors
These elements are in the direct control of the publisher
Content
Cq QUALITY Are pages well written & have substantial quality content?
Cr RESEARCH Have you researched the keywords people may use to find your content?
Cw WORDS Do pages use words & phrases you hope they'll be found for?
Cf FRESH Are pages fresh & about "hot" topics?
Cv VERTICAL Do you have image, local, news, video or other vertical content?
Ca ANSWERS Is your content turned into direct answers within search results?
Vt THIN Is content "thin" or "shallow" & lacking substance?
Architecture
Ac CRAWL Can search engines easily "crawl" pages on site?
Am MOBILE Does your site work well for mobile devices?
Ad DUPLICATE Does site manage duplicate content issues well?
As SPEED Does site load quickly?
Au URLS Do URLs contain meaningful keywords to page topics?
Ah HTTPS Does site use HTTPS to provide secure connection for visitors?
Vc CLOAKING Do you show search engines different pages than humans?
HTML
Ht TITLES Do HTML title tags contain keywords relevant to page topics?
Hd DESCRIPTION Do meta description tags describe what pages are about?
Hs STRUCTURE Do pages use structured data to enhance listings?
Hh HEADERS Do headlines & subheads use header tags with relevant keywords?
Vs STUFFING Do you excessively use words you want pages to be found for?
Vh HIDDEN Do colors or design "hide" words you want pages to be found for?


The Periodic Table of SEO Success Factors
Search engine optimization (SEO) seems like alchemy to the uninitiated. But there's a science to it. Below are some important "ranking factors" and best practices that can lead to success with both search engines and searchers.
On-The-Page SEO
Content
Architecture
HTML
Off-The-Page SEO
Trust
Links
Personal
Social
Cq Quality
Ac Crawl
Ht Titles
Ta Authority
Lq Quality
Pc Country
Sr Reputation
Cr Research
Am Mobile
Hd Description
Te Engage
Lt Text
Pl Locality
Ss Shares
Cw Words
Ad Duplicate
Hs Structure
Th History
Ln Numbers
Ph History
Cf Fresh
As Speed
Hh Headers
Vd Piracy
Vp Paid
Cv Vertical
Au URLs
Vs Stuffing
Va Ads
Vl Spam
Ca Answers
Ah HTTPS
Vh Hidden
Vt Thin
Vc Cloaking
Factors Work Together
All factors on the table are important, but those marked 3 carry more weight than 1 or 2. No single factor guarantees top rankings or success, but having several favorable ones increases the odds. Negative "violation" factors shown in red harm your chances.


Off-The-Page Factors
Elements influenced by readers, visitors & other publishers
Trust
Ta AUTHORITY Do links, shares & other factors make pages trusted authorities?
Te ENGAGE Do visitors spend time reading or "bounce" away quickly?
Th HISTORY Has site or its domain been around a long time, operating in same way?
Vd PIRACY Has site been flagged for hosting pirated content?
Va ADS Is content ad-heavy? Do you make use of intrusive interstitials?
Links
Lq QUALITY Are links from trusted, quality or respected web sites?
Lt TEXT Do links pointing at pages use words you hope they'll be found for?
Ln NUMBER Do many links point at your web pages?
Vp PAID Have you purchased links in hopes of better rankings?
Vl SPAM Have you created links by spamming blogs, forums or other places?
Personal
Pc COUNTRY What country is someone located in?
Pl LOCALITY What city or local area is someone located in?
Ph HISTORY Has someone regularly visited your site?
Social
Sr REPUTATION Do those respected on social networks share your content?
Ss SHARES Do many share your content on social networks?
WRITTEN BY: Search Engine Land
CREATED BY: COLUMN FIVE
LEARN MORE: http://salnd.com/seotable
© 2017 Third Door Media

PageRank First Pass



PageRank example

- Simulates a “random-surfer”
- Begins with pair (URL, *list-of-URLs*)
- Preprocess to (URL, (PR, *list-of-URLs*))
- Maps again taking above data, and for each *u in list-of-URLs* returns two items:
  - *(u*, PR/|*list-of-URLs*|, *list-of-URLs*),
  - (*u, URL1)*, (*u, URL2)*, (*u, URL3), …*
- Reduce receives (URL, list-of-URLs), and many (URL, value) pairs, and calculates:
  - (URL, new-PR, *new-list-of-URLs*)

PageRank

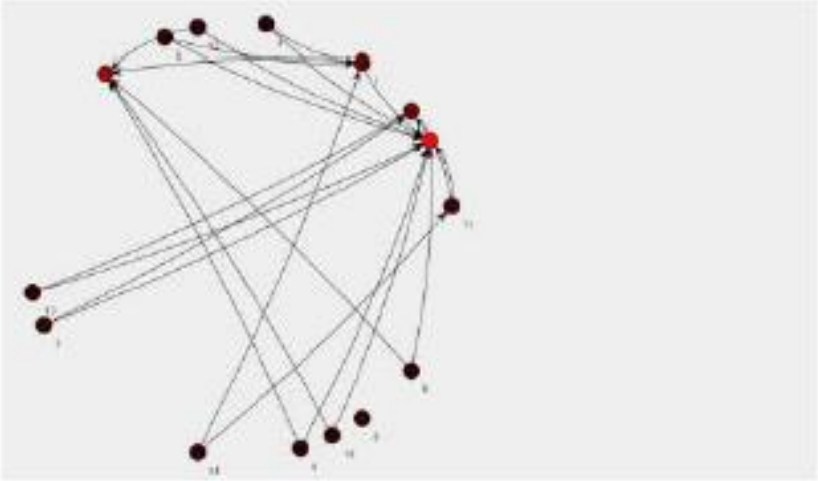
- Start with a number of Web pages
- Each Web page points to another
- Each “pointing” on a Web page (a URL!) is a “vote” of the Web page for another Web page
- Elect the most “popular” Web page, and rank all the others in terms of popularity

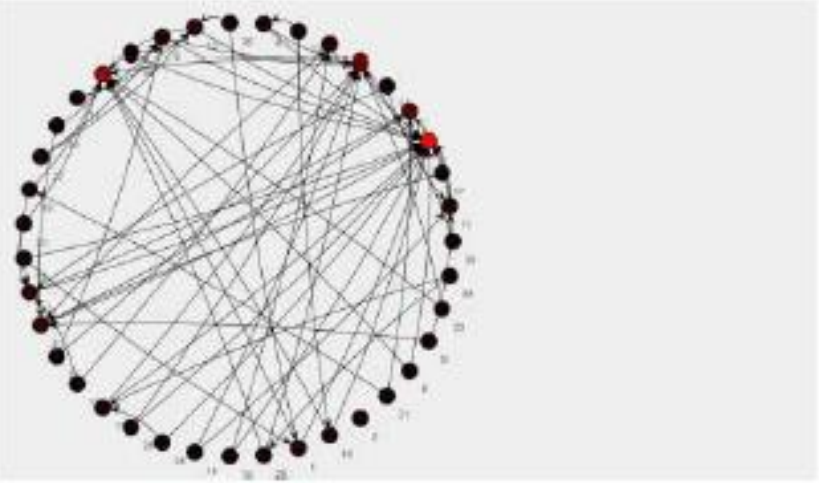
How to compute?

- The internet is huge: Google has found over 1 trillion unique URLs!
- Assume each URL takes 0.5K, then we need over 400TB just to store URLs (no content caching)!
- Need an efficient algorithm (Google's *PageRank*), and need an efficient computation framework (*MapReduce*)

Step 0

- Start with a number of interlinked web pages





Step 1: Parsing URLs

- ***Map:***
- foo.html:
  <html><body><***a href***=a><**a href**=b>…</body></html>
- bar.html:
  <html><body><***a href***=a><**a href**=y>…</body></html>
- Other web pages…
- Output: ∀outlink:
  (key=foo, value=a)
  (key=foo, value=b)
  …
  (key=bar, value=a)
  …

- ***Reduce:***
  (key=foo, value=a, b,..)
  (key=bar, value=a, y,..)
  …
- Assign an initial PageRank to foo, bar, etc:
  - Assume each web page has 1/total-of-web-pages "voting power"
    (key=foo, value=PR(foo),a, b,..)
    (key=bar, value=PR(bar),a, y,..)

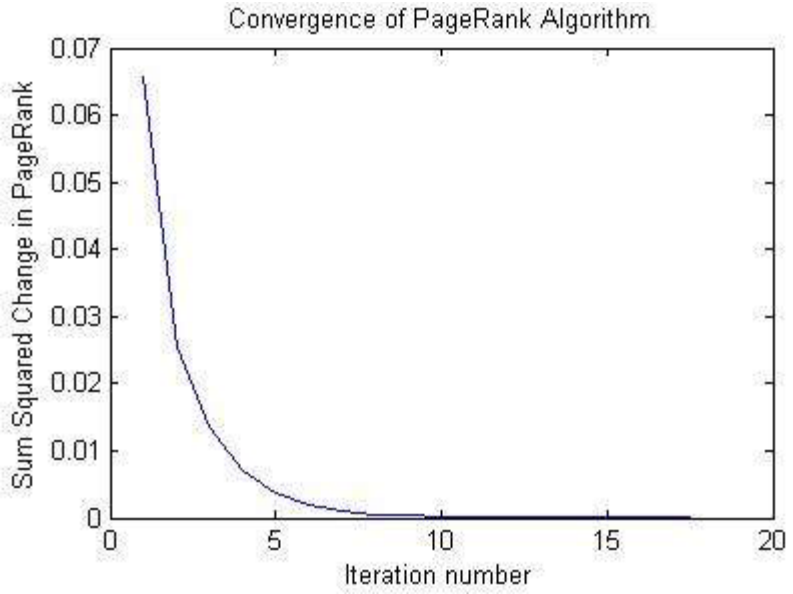
Step 2: Recurse

- ***Map:***
- Input:
  (key=foo, value=PR(foo),a, b,..)
  (key=bar, value=PR(bar),a, y,..)
  …
- Output: ∀outlink:
  (key=a, value=foo, PR(foo),#-of-outlinks(foo))
  (key=b, value=foo, PR(foo),#-of-outlinks(foo))
  (key=a, value=bar, PR(bar),#-of-outlinks(bar))
  …

- ***Reduce:***
- Input:
  (key=a, value=foo, PR(foo),#-of-outlinks(foo),bar, PR(bar),#-of-outlinks(bar),..)
  …
- Calculation: $PR(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j;t)}{L(p_j)}$
  (key=a, value=new PR(a))
  (key=b, value=new PR(b))
  (key=y, value=new PR(y))
  (key=foo, value=new PR(foo))
  (key=bar, value=new PR(bar))
  …
- Output:
  (key=foo, value=PR_{new}(foo),a, b,..)
  (key=bar, value=PR_{new}(bar),a, y,..)

Finally

- Sort the list of all Web pages by PageRank, and present to the user!
- The PageRank algorithm converges rapidly for any sized web-graph. Figure below shows the convergence for a graph of 1000 vertices (with an average of two links per vertex).
- Brin and Page report that PageRank computation for a webgraph of 322 million links converges in only 52 iterations. (Page, Lawrence; Brin, Sergey; Motwani, Rajeev and Winograd, Terry *Bringing order to the Web* (1999))



Bit complicated?

- Let’s do it again ☺

Step 1: Graph Builder

- foo.html:
  <html><body><***a href***=a><a href=b>…</body></html>
- bar.html:
  <html><body><***a href***=a><a href=c>…</body></html>
- Other web pages…
- Output:
  (key=foo, value=a, b,..)
  (key=bar, value=a, c,..)
  …

Assign an initial PageRank to foo, bar, etc.:

- Assume each web page has 1/total-of-web-pages "voting power"

(key=foo, value=PR_1(foo),a, b,..)
(key=bar, value=PR_1(bar), a, c,..)

Most often, $PR_1(x)$ = 0.5 or 0.1

Possible to help this process by starting with a flat text file, where each page is embedded on a single line

Step 2: Iterate

Outlinks or votes (foo points to)

- ***Map:***
- Input:
  (key=foo, value=PR_i(foo),a, b,..)
  (key=bar, value=PR_i(bar),a, c,..)
  …
  (key=a, value=PR_i(a),p, q,..)
- Output: ∀outlink:
  (key=a, value=foo, PR(foo),#-of-outlinks(foo))
  (key=b, value=foo, PR(foo),#-of-outlinks(foo))
  …

Inlinks or voters (a pointed to by)

- ***Reduce:***
- Input:
  (key=a, value=foo, PR(foo),#-of-outlinks(foo))
  (key=a, value=bar, PR(bar),#-of-outlinks(bar))
  …
  (key=a, value=PR_i(a),p, q,..)
- Output:
  (key= a, value=Pr_{i+1}(a), p, q,..)

$$PR(p_i;t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j;t)}{L(p_j)}$$

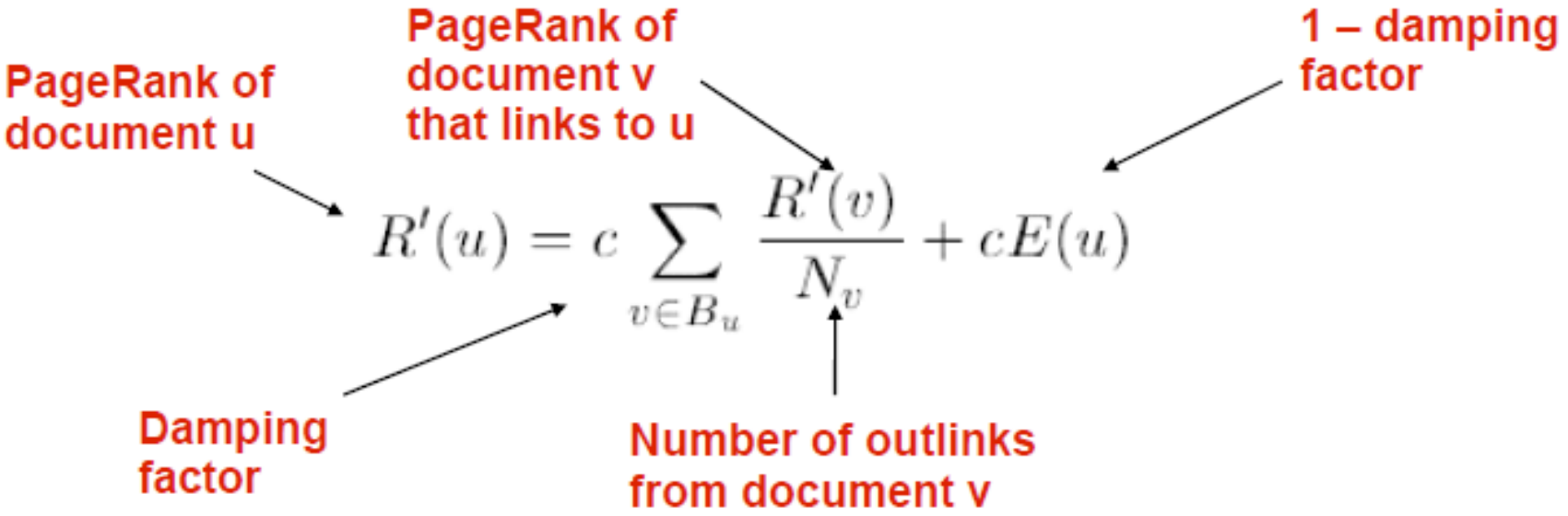
1. Start with an initial PR and outlinks for each document

2. For each outlink, output the source document, and that document's PR and outlink count

3. The reducer has many identical document ids, each with a different inlink to it with the inlink's PR and inlink's number of outlinks

4. Reducer computes new PR, and returns page and its outlinks

Refining PageRank

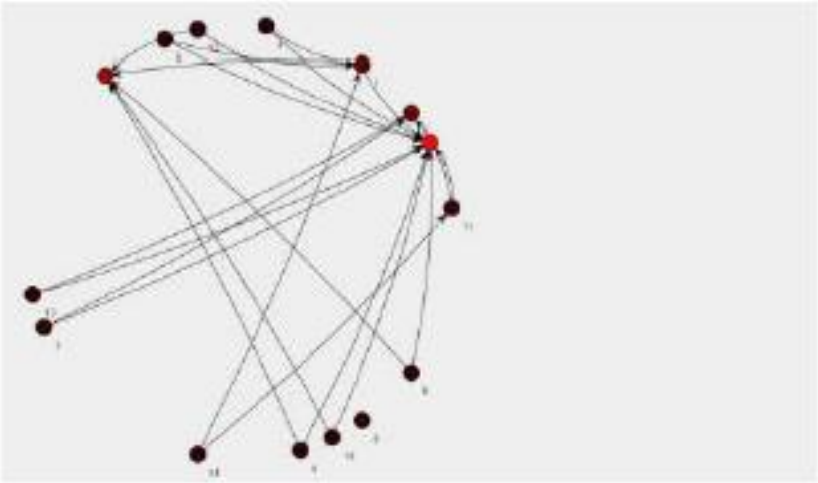


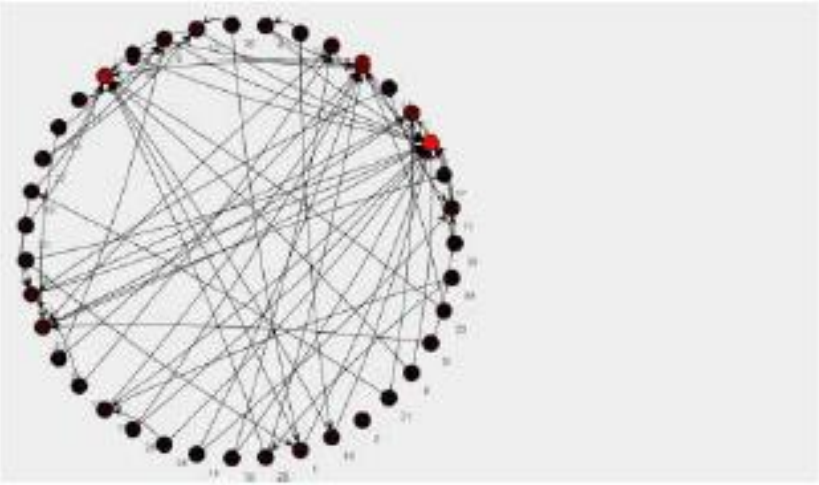
Allows random-surfer to "teleport" out of the sub-graph, essentially getting him unstuck


KEEP
CALM
AND
LET'S DO
IT AGAIN
KeepCalmAndPosters.com

Step 0

- Start with a number of interlinked objects





Step 1

- PageA -> Page1, Page2, Page3...
- PageB -> Page1', Page2', Page3'...
- Before the first iteration, we need to assign the initial rank for each webpage
  - 0.5 is a good number for small website or network
- <PageA, 0.5> -> Page1, Page2, Page3...
- <PageB, 0.5> -> Page1', Page2', Page3'...
  - This data structure indicates the *outlinks* (votes) by each Page for each other Page
  - But in PageRank Silver Surfer Model, we are more concerned with *inlinks* (voters)!
  - So the first thing we need to do in our Mapper function is to invert/pivot the votes/voters relationship..

Map (1st pivot/inversion)

```
function Mapper
input <PageN, RankN>, PageA, PageB, PageC...
begin
    Nn := the number of outlinks for PageN;
    for each outlink PageK
        output <PageK, <PageN, RankN/Nn>>
    // output the outlinks for PageN.
    output <PageN,<PageA, PageB, PageC…>>
end
```

Outlinks = votes,

Inlinks = voters

Now repeat this process for many pages N

So after Mapper function, we have the following data structure:

$<PageK_1, <PageN, RankN/Nn >>$

$<PageK_2, <PageN, RankN/Nn >>$

…

<PageN,<PageA, PageB, PageC …>>

where PageN is the inlink of $PageK_i$, and PageA, PageB … are the outlinks of PageN.

These outlinks are saved because we need them to rebuild the input format for the next iteration.

Reduce

- Update PageRank for PageK, using the PageRanks of inlinks:

```
function Reducer
                    // inlinks of page K one by one, followed by list of its outlinks
input <PageK, <PageN1, RankN1/Nn1>>,
                <PageN2, RankN2/Nn2>>,
                ...
      PageK, <PageAk, PageBk, PageCk...>>
begin
      RankK := 0;
      for each inlink PageNi
            RankK += RankNi/Nni * beta
      // output the PageK and its new Rank for the next iteration
      output <<PageK, RankK>, <PageAk, PageBk, PageCk...>>
end
```

$$PR(p_i; t+1) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)}$$

Repeat..

- We start with outlink pages and all their inlinks
- We MapReduce..
- We end up with inlink pages and their outlinks, exactly what we had at the very beginning!

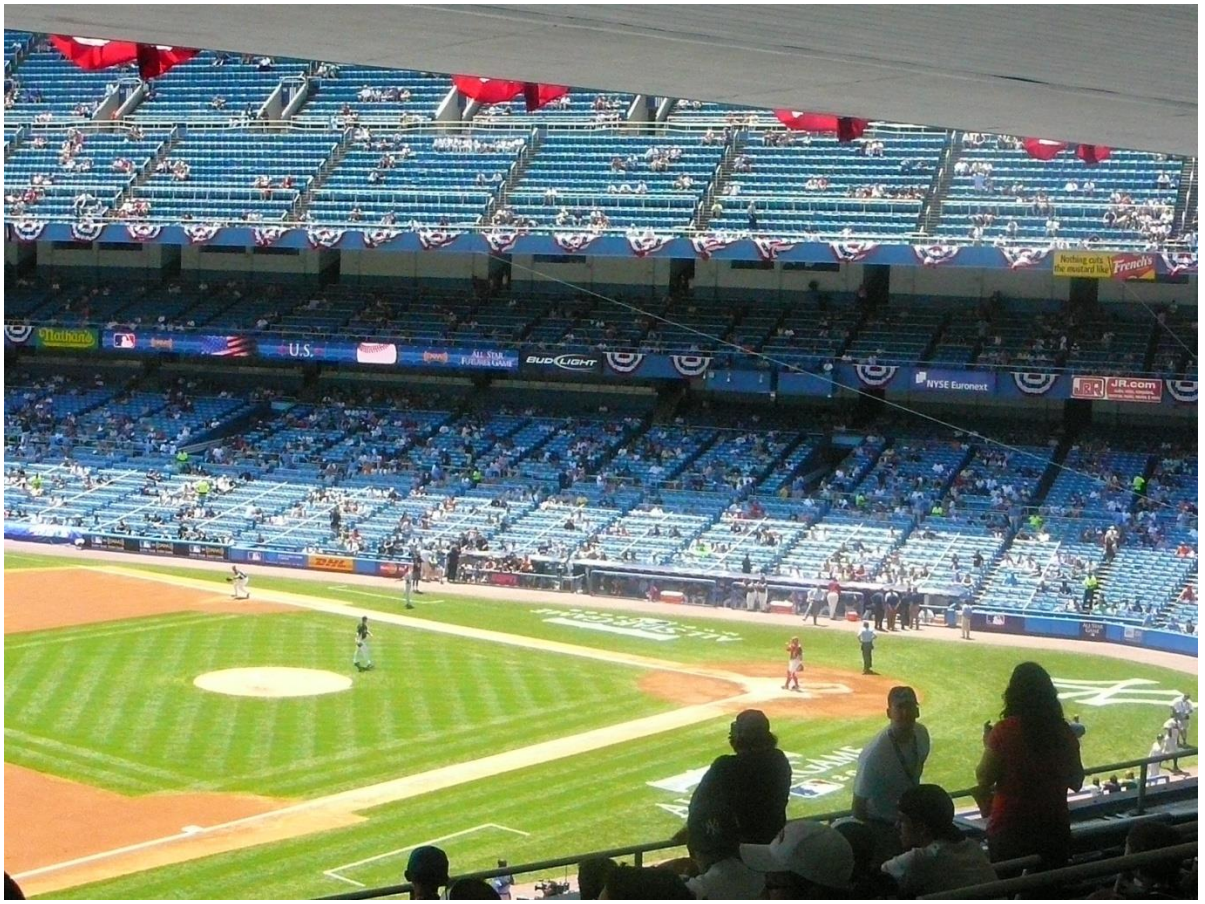
Fast convergence of iteration

- Repeat and repeat and repeat..

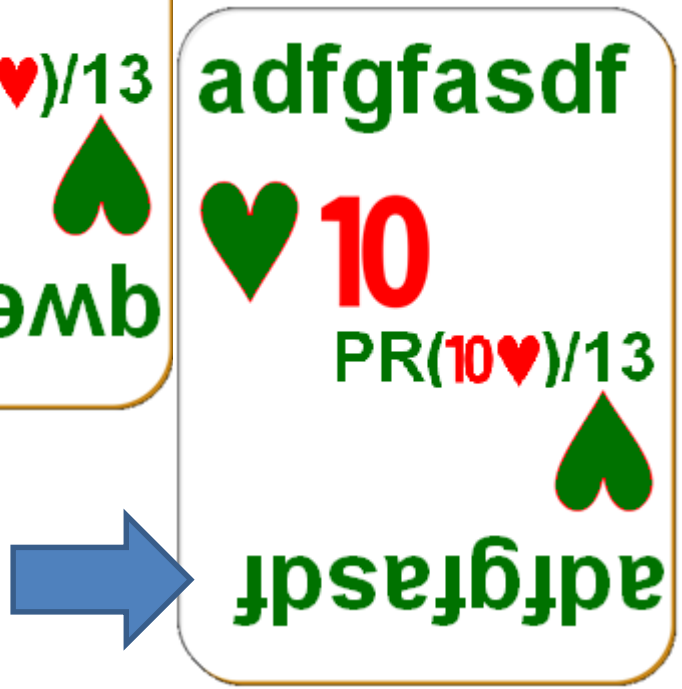
  All you need is to run this Map-Reduce procedure iteratively till convergence. 15 or so iterations are enough in small cases, larger case may take a few dozen more iterations..

- Because the WWW is sparse..
  - (Horsey-farm to horsey-farm!)
  - You will converge pretty fast!

Sparse Graph

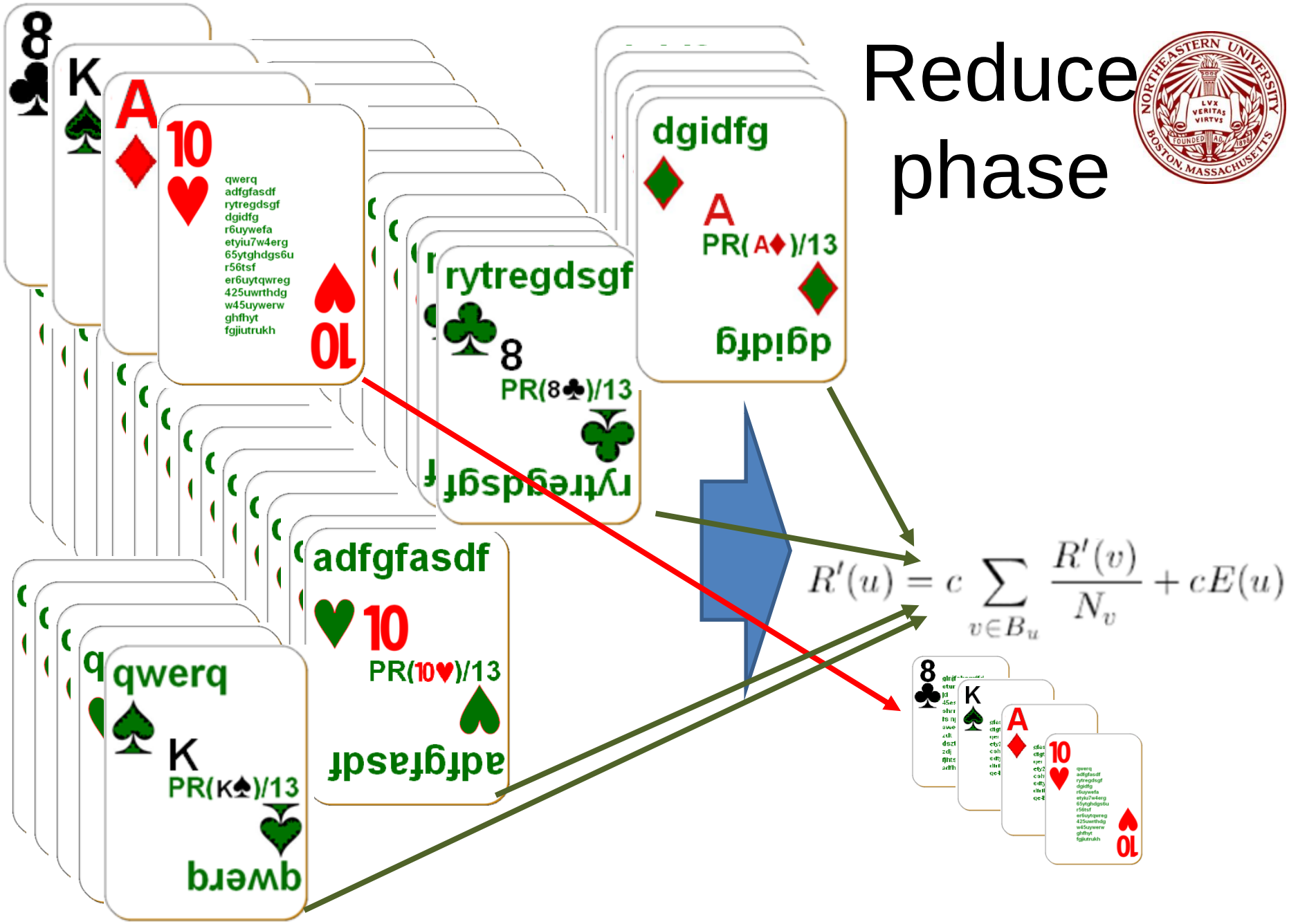


Card Analogy: Map phase



• • •

Reduce phase



Matrix Algebra

- Any state vector can be written as a linear combination of the eigenvectors of A (if A is not degenerate)
- The product of this state vector with the transition Matrix A represents the next state vector
- Since multiplying A by the eigenvector of A is equal to the eigenvalue λ times the eigenvector, and A^n times the eigenvector of A is λ^n times the eigenvector, and when n is large, the only contribution remaining is the **dominant eigenvector**
- So all we have to do is multiply a state vector by A^n where n is a large number, and we will eventually converge to the dominant eigenvector. That's the *Power method*!

Matrix Multiplication

- Matrix Multiplication and inversion can be accomplished via mapreduce methods, for very large Matrices (dimensions above thousands)