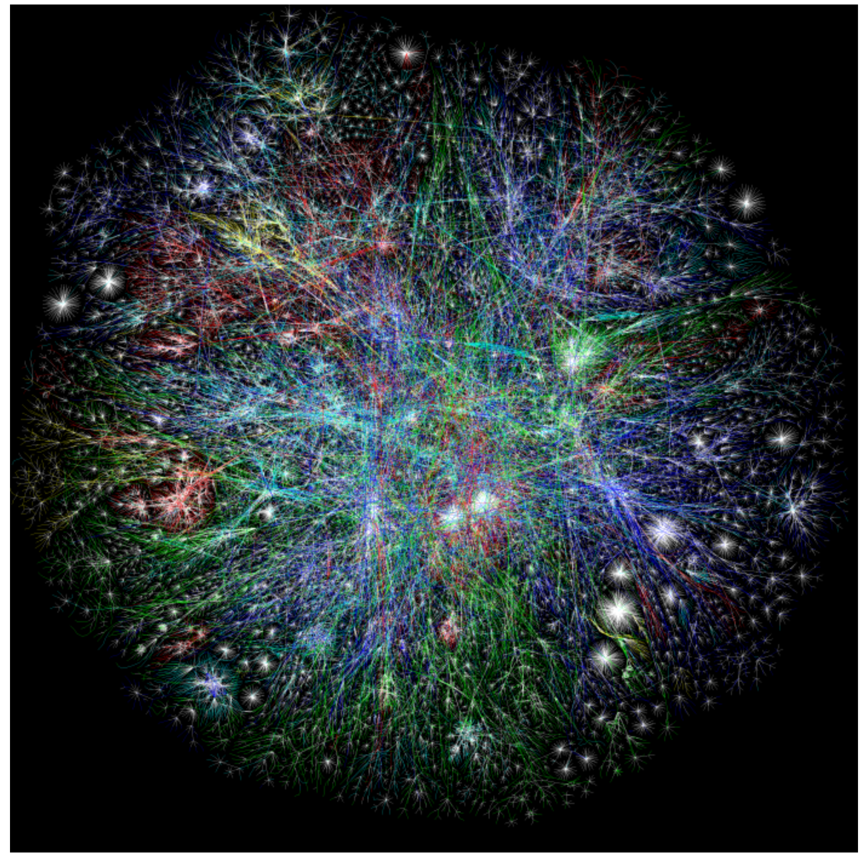
The Technology Behind Google

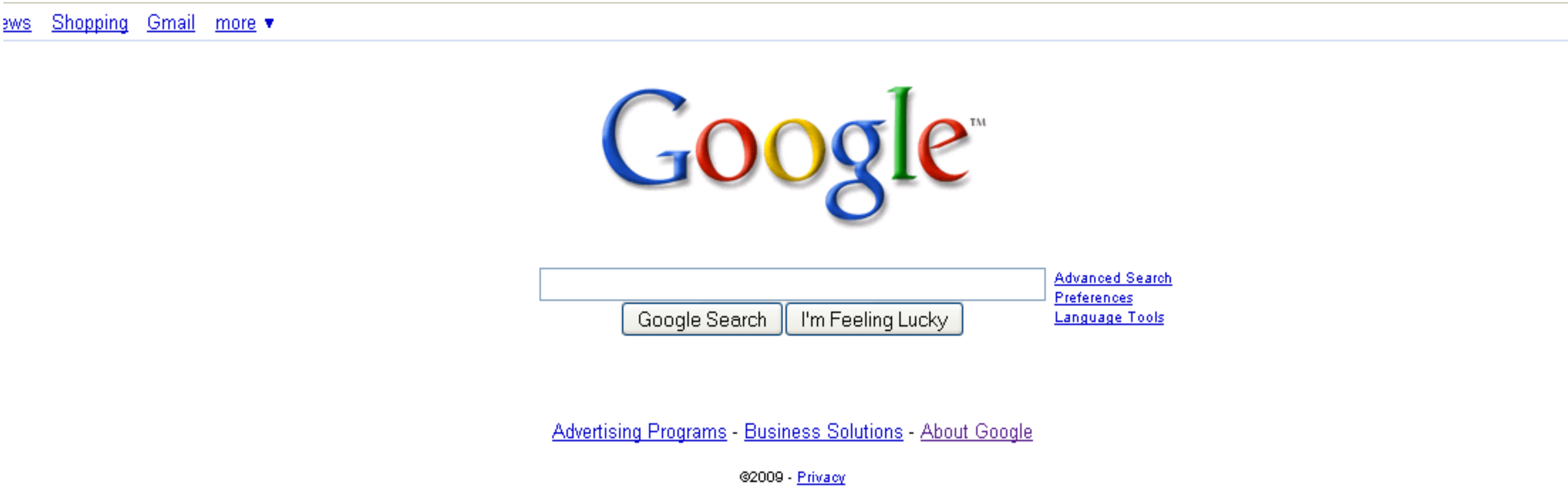
Slides organized by:
Sudhanva Gurumurthi

The World Wide Web

- In July 2008, Google announced that they found **1 trillion** unique webpages!
- **Billions** of new web pages appear each day!
- About **1 billion** Internet users (and growing)!!



The World Wide Web in 2003.
Image source: http://www.opte.org/



Use a huge number of computers – Data Centers
An ordinary Google Search uses 700-1000 machines!

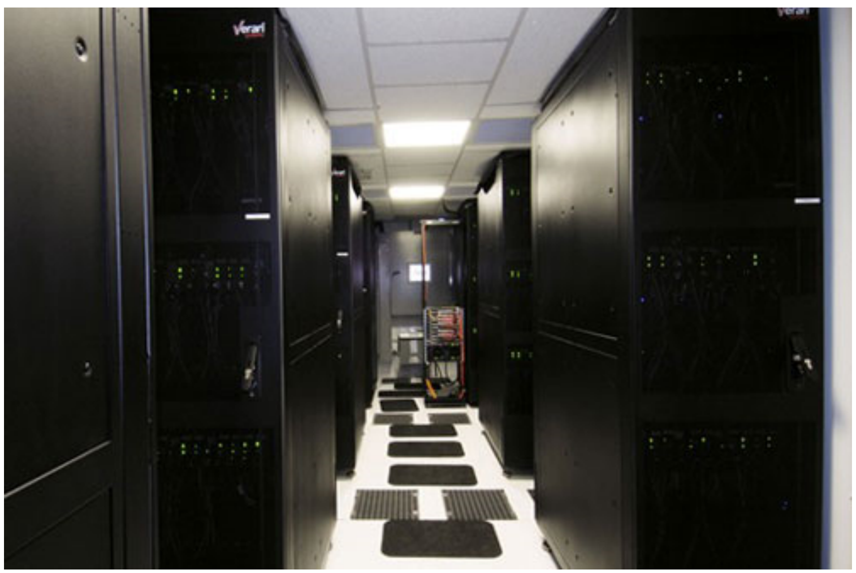
Search through a massive number of webpages – MapReduce

Find which webpages match your query - PageRank

Data Centers

- Buildings that house computing equipment

- Contain 1000s of computers



Inside a Microsoft Data Center

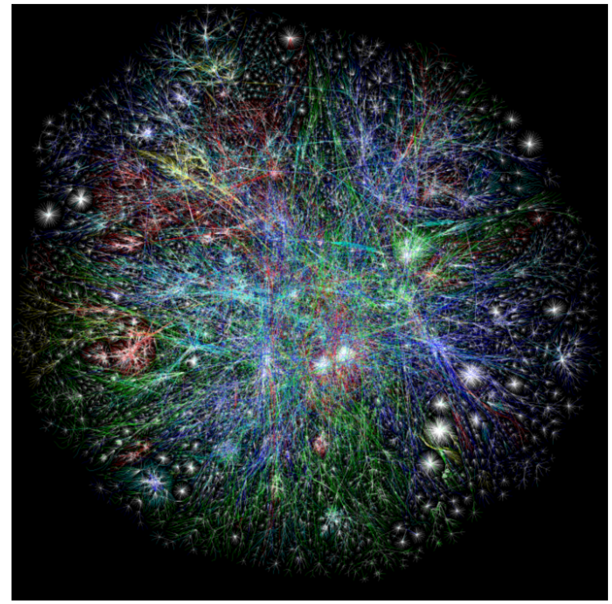
Google's Data Center in The Dalles, Oregon

Google's 36 Data Centers

Why do we need so many computers?

- Searching the Internet is like looking for a needle in a haystack!
  - There are a **trillion** webpages
  - There are **millions** of users
  - Imagine writing a for or while-loop to search the contents of each webpage!

☞ Use the 1000s of computers **in parallel** to speed up the search



Map/Reduce

- Adapted from the Lisp programming language
- Easy to distribute across many computers

MapReduce slides adapted from Dan Weld's slides at U. Washington: http://rakaposhi.eas.asu.edu/cse494/notes/s07-map-reduce.ppt

Map/Reduce in Lisp

- (`map` ***f*** ***list [list$_2$ list$_3$ ...]***)

Unary operator

- (`map` square ‘(1 2 3 4))
  - (1 4 9 16)

Binary operator

- (`reduce` + ‘(1 4 9 16))
  -
  - 30

Map/Reduce ala Google

- map(key, val) is run on each item in set
  – emits new-key / new-val pairs

- reduce(key, vals) is run for each unique key emitted by map()
  – emits final output

Example: Counting words in webpages

– Input consists of (url, contents) pairs

– map(key=url, val=contents):

- For each word *w* in contents, emit (w, "1")

– reduce(key=word, values=uniq_counts):

- Sum all "1"s in values list
- Emit result "(word, sum)"

Count, Illustrated

map(key=url, val=contents):

  For each word *w* in contents, emit (w, “1”)

reduce(key=word, values=uniq_counts):

  Sum all “1”s in values list

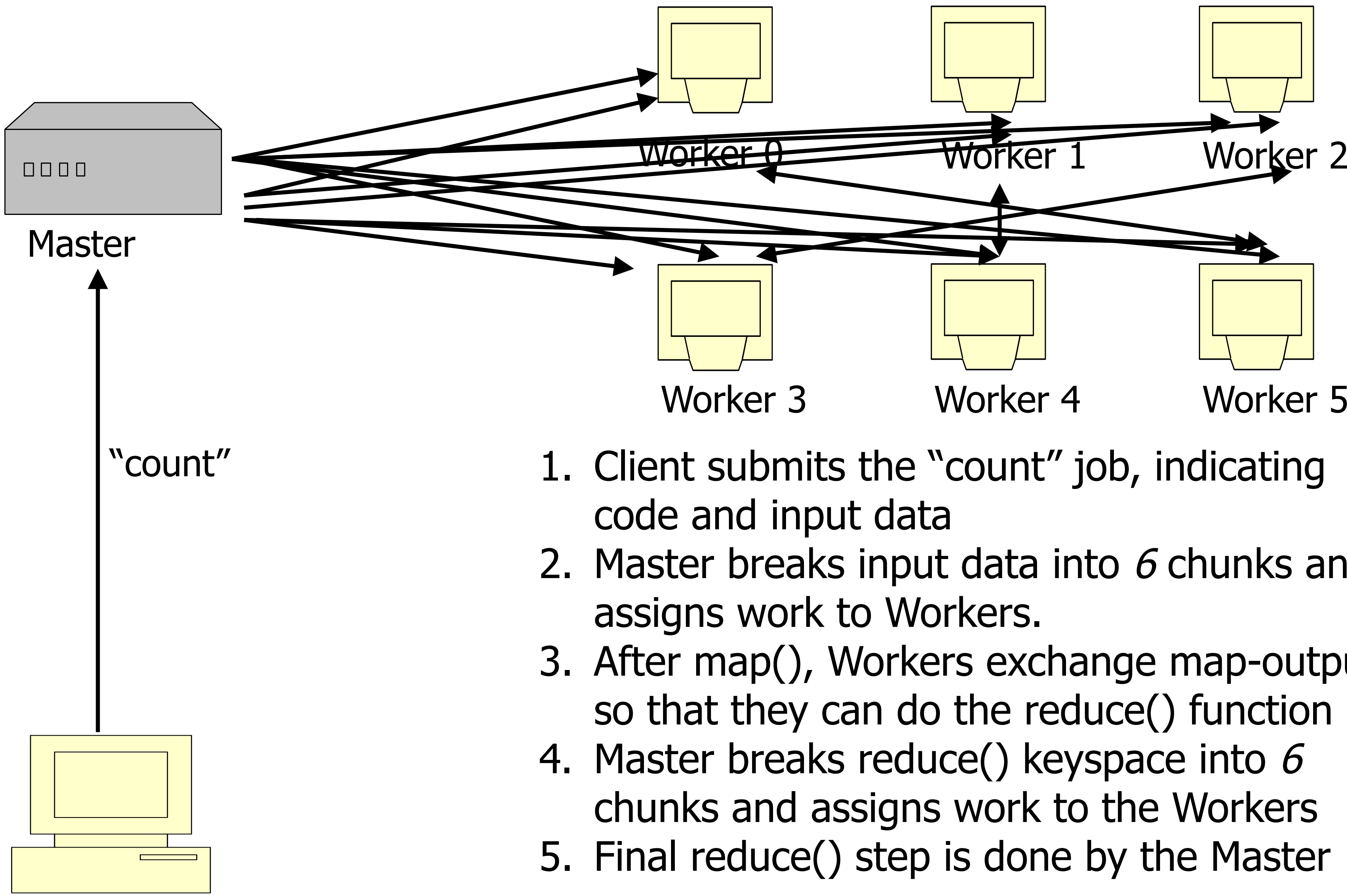
  Emit result “(word, sum)”

see bob throw
see spot run

see	1
bob	1
run	1
see	1
spot	1
throw	1

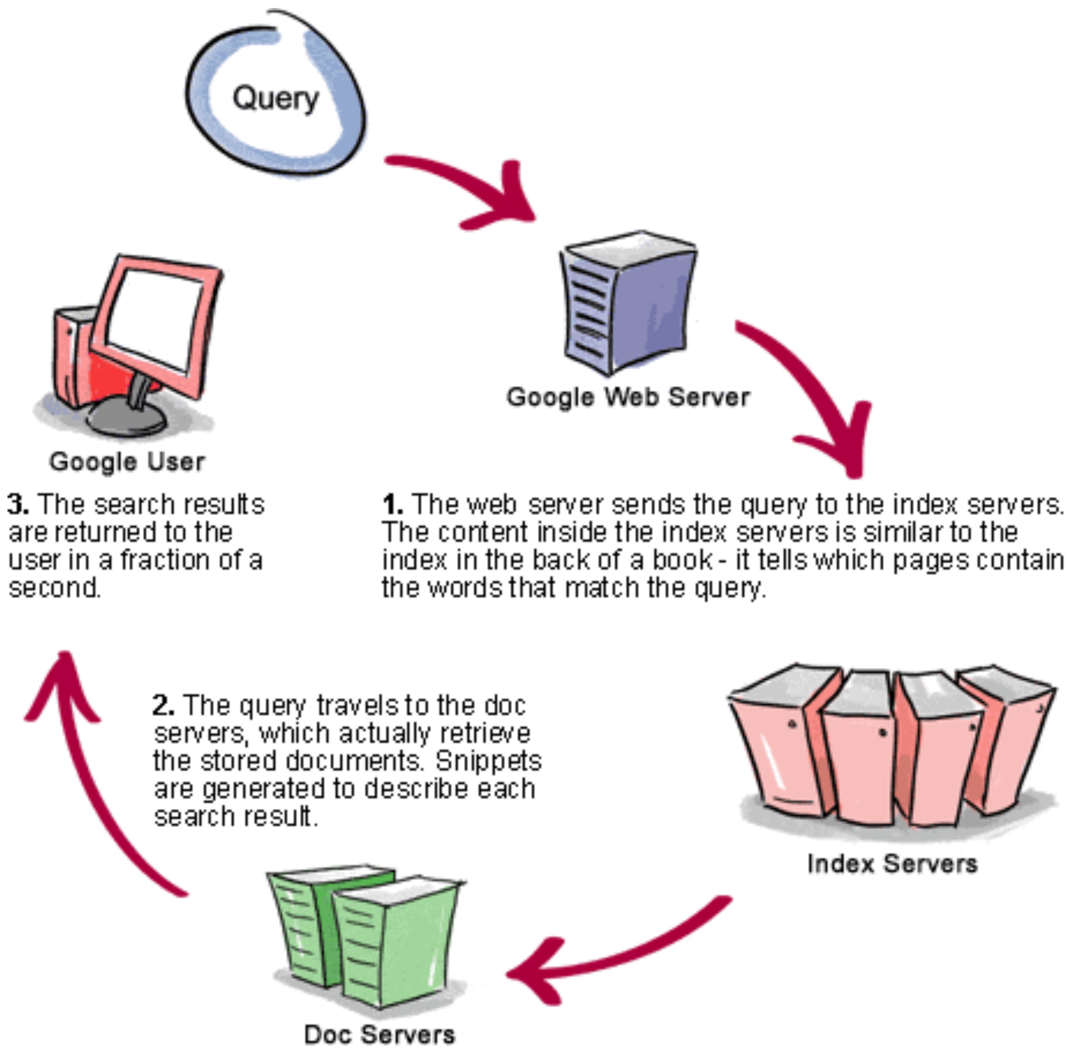
bob	1
run	1
see	2
spot	1
throw	1

Map/Reduce Job Processing



1. Client submits the "count" job, indicating code and input data
2. Master breaks input data into *6* chunks and assigns work to Workers.
3. After map(), Workers exchange map-output so that they can do the reduce() function
4. Master breaks reduce() keyspace into *6* chunks and assigns work to the Workers
5. Final reduce() step is done by the Master

The Life of a Google Query



MapReduce
+
PageRank

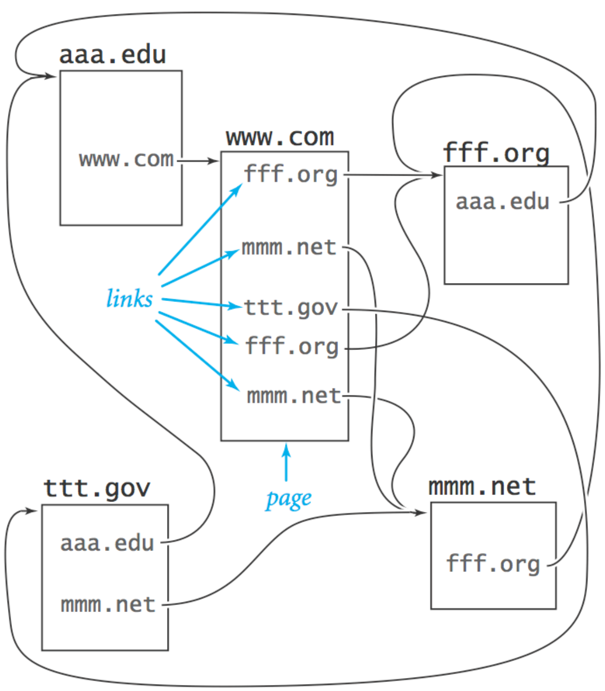
Image Source: www.google.com/corporate/tech.html

Finding the Right Websites for a Query

- Relevance - Is the document similar to the query term?
- Importance - Is the document useful to a variety of users?
- Search engine approaches
  - Paid advertisers
  - Manually created classification
  - Feature detection, based on title, text, anchors, …
  - "Popularity"

Google's PageRank™ Algorithm

- Measure popularity of pages based on hyperlink structure of Web.



Pages and links


Google Founders – Larry Page and Sergei Brin

90-10 Rule

- Model. Web surfer chooses next page:
  - 90% of the time surfer clicks a link on current page.
  - 10% of the time surfer types a random page.

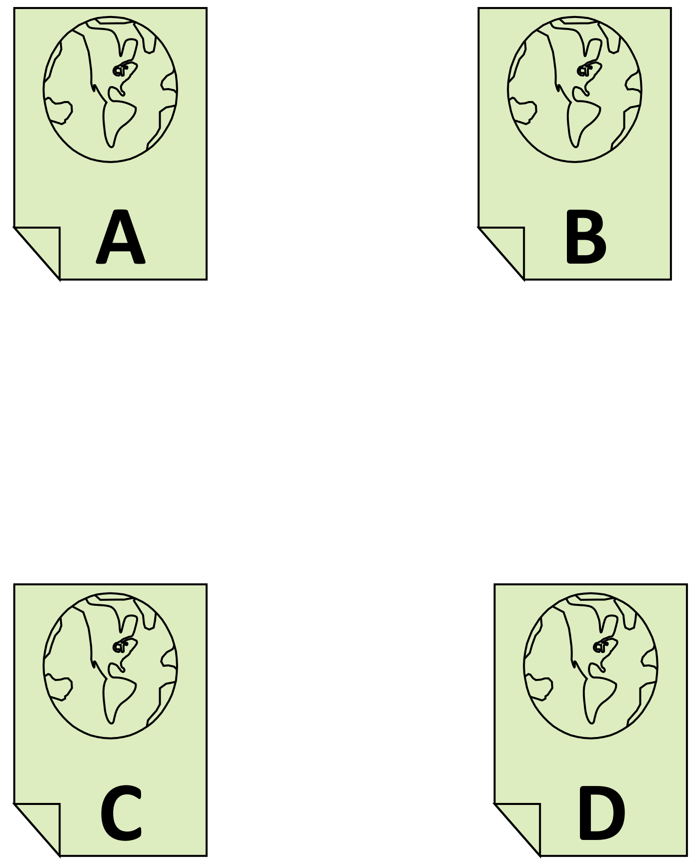
- Crude, but useful, web surfing model.
  - No one chooses links on a page with equal probability.
  - The 90-10 breakdown is just a guess.
  - It does not take the back button or bookmarks into account.

Basic Ideas Behind PageRank

- PageRank is a probability distribution that denotes the likelihood that the "random surfer" will arrive at a particular webpage.
- Links coming from important pages convey more importance to a page.
  - If a web page has a link off the CNN home page, it may be just one link but it is a very important one.
- A page has high rank if the sum of the ranks of its inbound links is high.
  - Covers the cases where a page has many inbound links and also when a page has a few highly ranked inbound links.

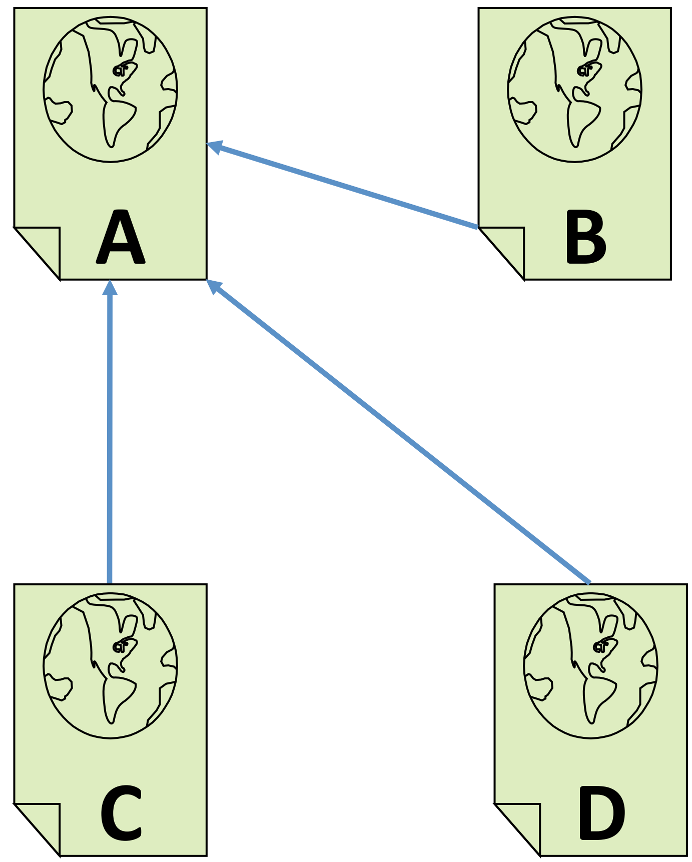
The PageRank Algorithm

- Assume that there are only 4 pages – A, B, C, D and that the distribution is evenly divided among the pages
  – PR(A) = PR(B) = PR(C) = PR(D) = 0.25



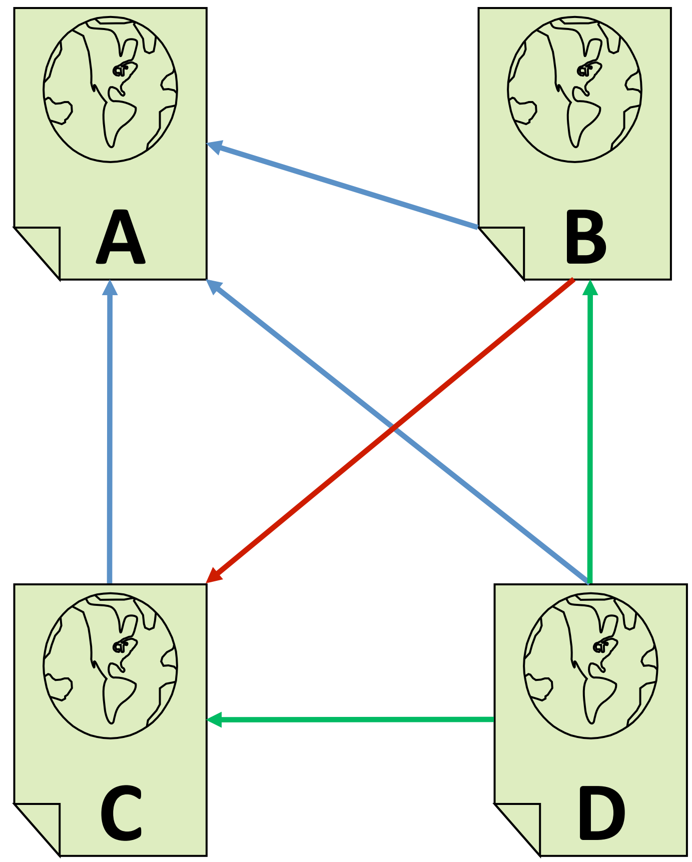
If B, C, D each only link to A

- B, C, and D each confer their 0.25 PageRank to A
- PR(A) = PR(B) + PR(C) + PR(D) = 0.75



Assume B links to C and D links to B and C

- Value of link-votes divided amongst the outbound links on a page
  - B gives vote worth 0.125 to A and C
  - D gives vote worth 0.083 to A, B, C
- PR(A) = (PR(B)/2) + (PR (C)/1) + (PR(D)/3)



PageRank

- The PageRank for any page *u*:

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

PR(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**

References

- The paper by Larry Page and Sergei Brin that describes their Google prototype:

http://infolab.stanford.edu/~backrub/google.html

- The paper by Jeffrey Dean and Sanjay Ghemawat that describes MapReduce:

http://labs.google.com/papers/mapreduce.html

- Wikipedia article on PageRank: http://en.wikipedia.org/wiki/PageRank