# Some Interesting Applications of Theory

PageRank

Minhashing

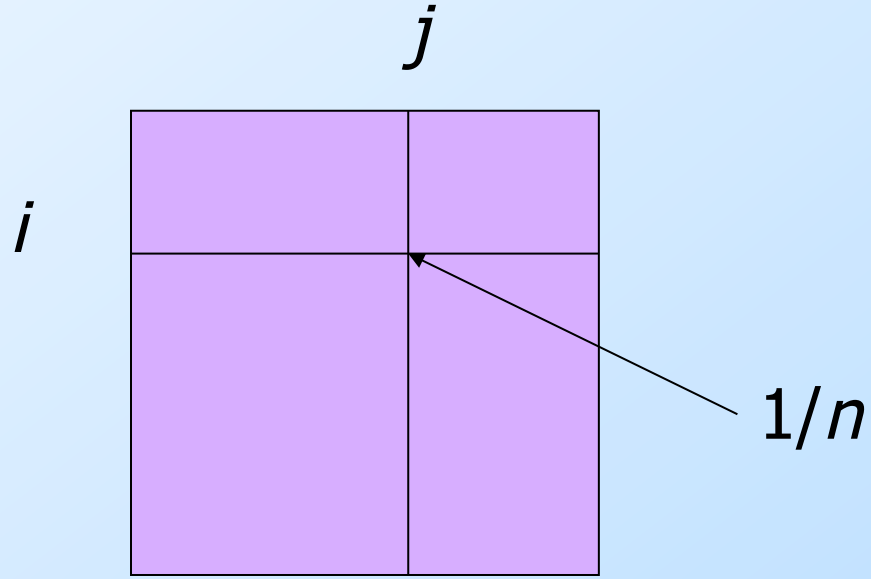
Locality-Sensitive Hashing

# PageRank

- The thing that makes Google work.
- Intuition: solve the recursive equation: "a page is important if important pages link to it."
- In high-falutin' terms: *importance* = the principal eigenvector of the stochastic matrix of the Web.
  - A few fixups needed.

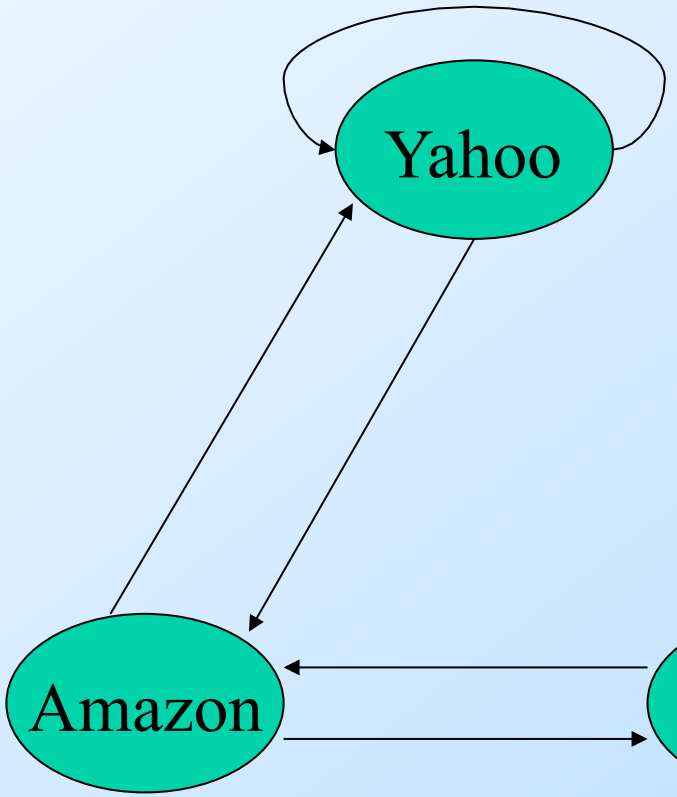
# Stochastic Matrix *M* of the Web

Suppose page *j* links to *n* pages, including *i*




Expresses how "importance" flows around the Web. Equivalent to following "random walkers."

# Example: The Web in 1839



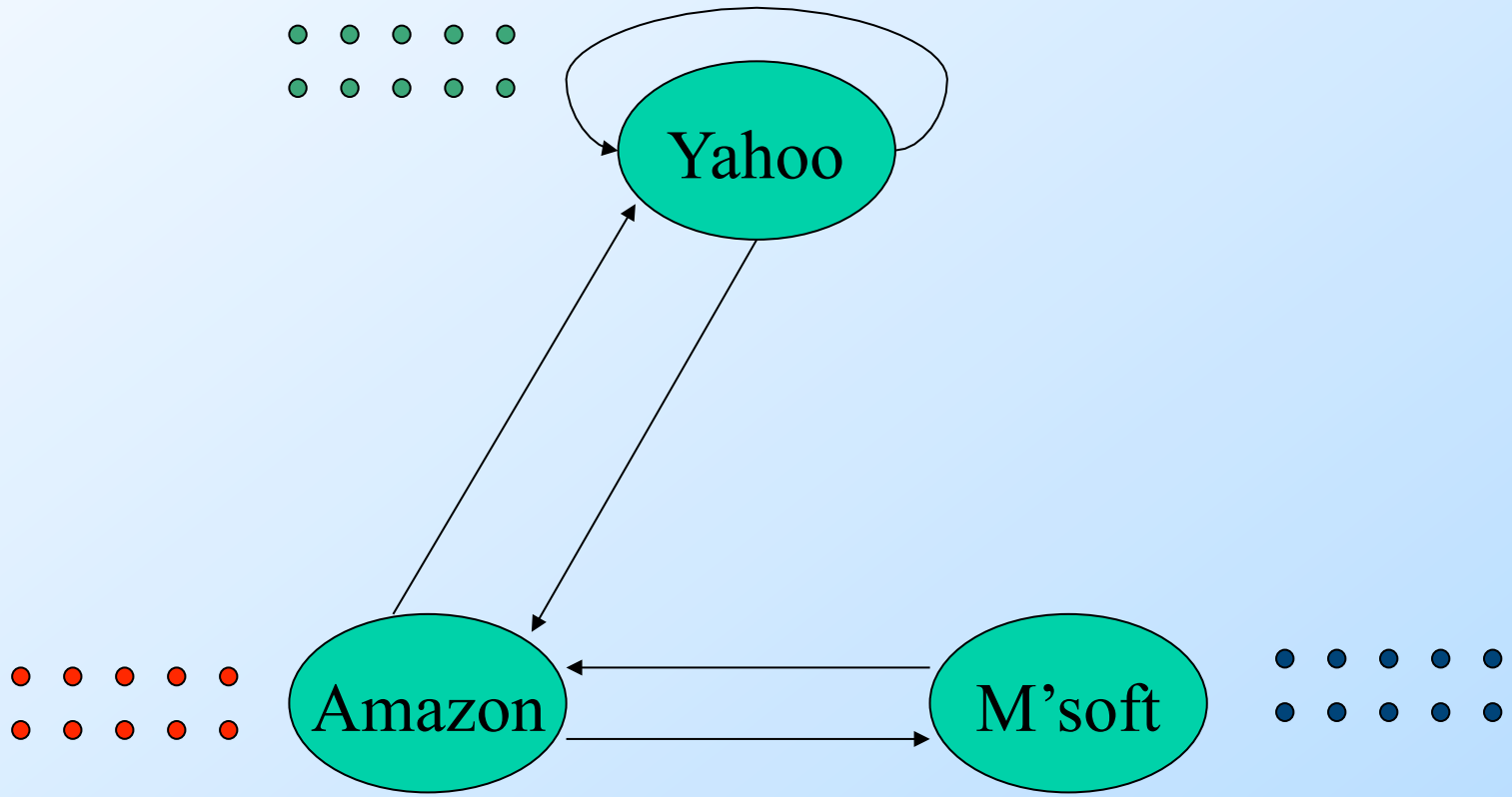

| | y | a | m |
|---|---|---|---|
| y | 1/2 | 1/2 | 0 |
| a | 1/2 | 0 | 1 |
| m | 0 | 1/2 | 0 |

*M*

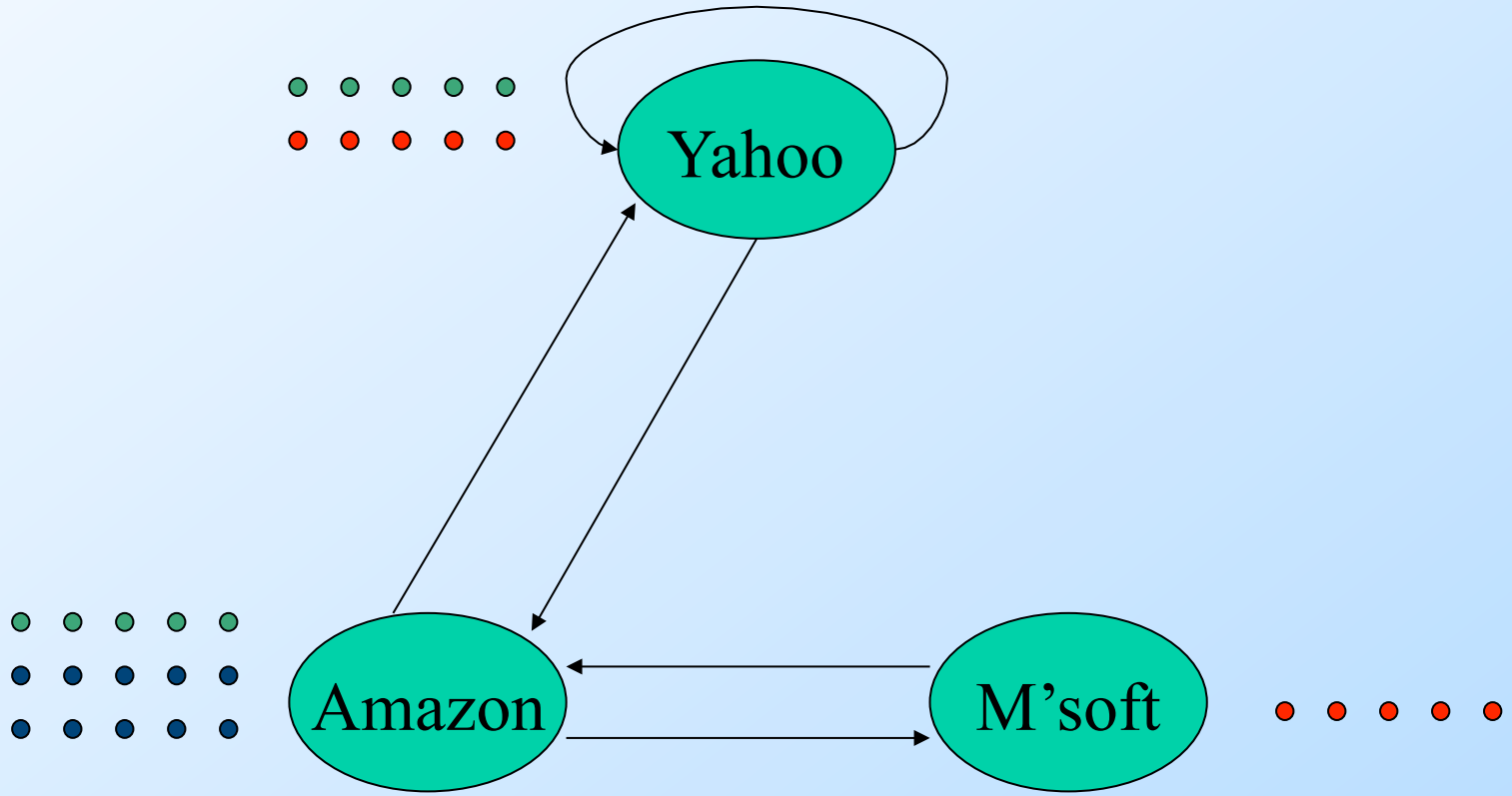
# The Google Idea

- Imagine many random walkers on the Web.
- At each "tick," each walker picks an out-link at random and follows it.
- Distribution of walkers **v** becomes $M\,\mathbf{v}$ after one tick.
- Compute $M^{50}\mathbf{v}$ (approximately 50).

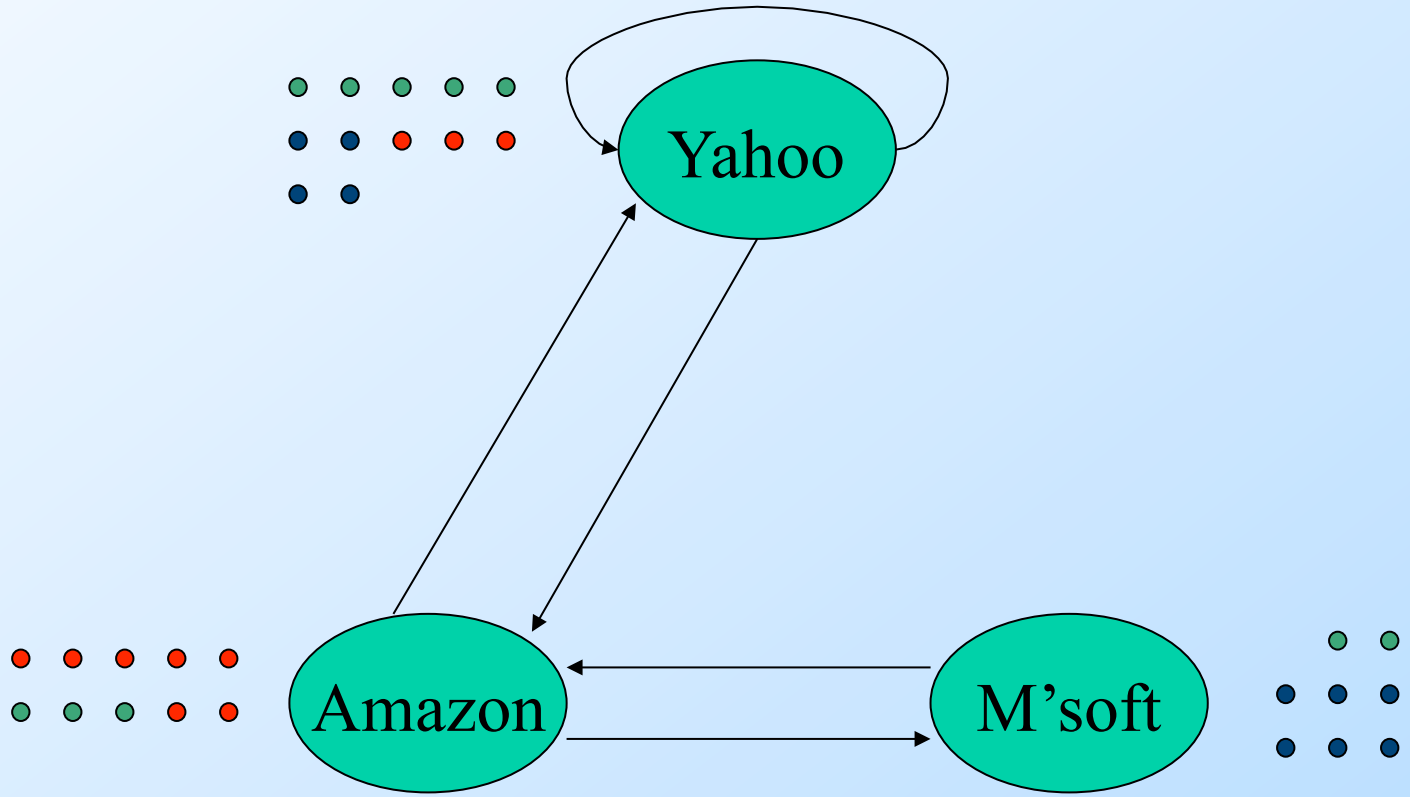
# The Walkers



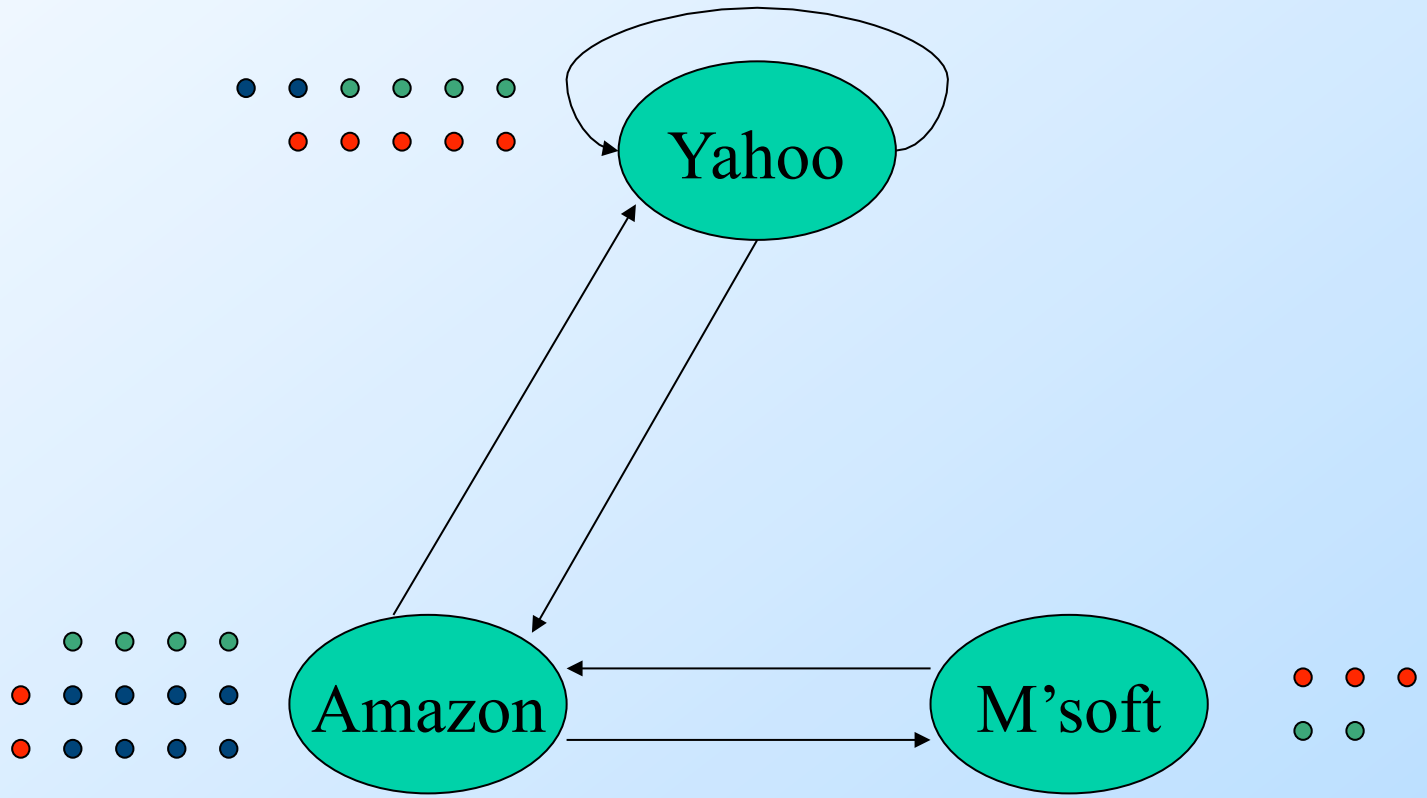
# The Walkers



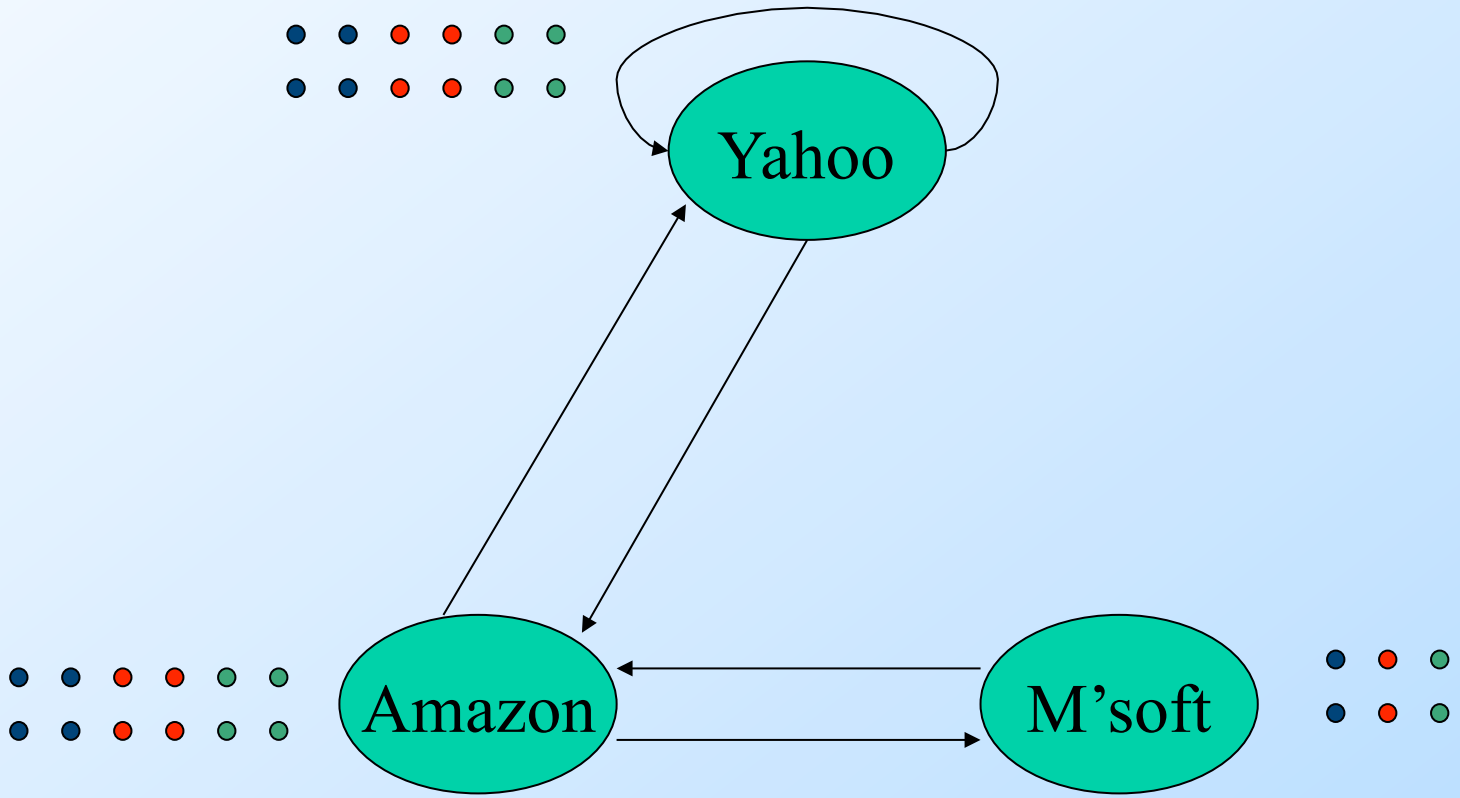
# The Walkers



# The Walkers



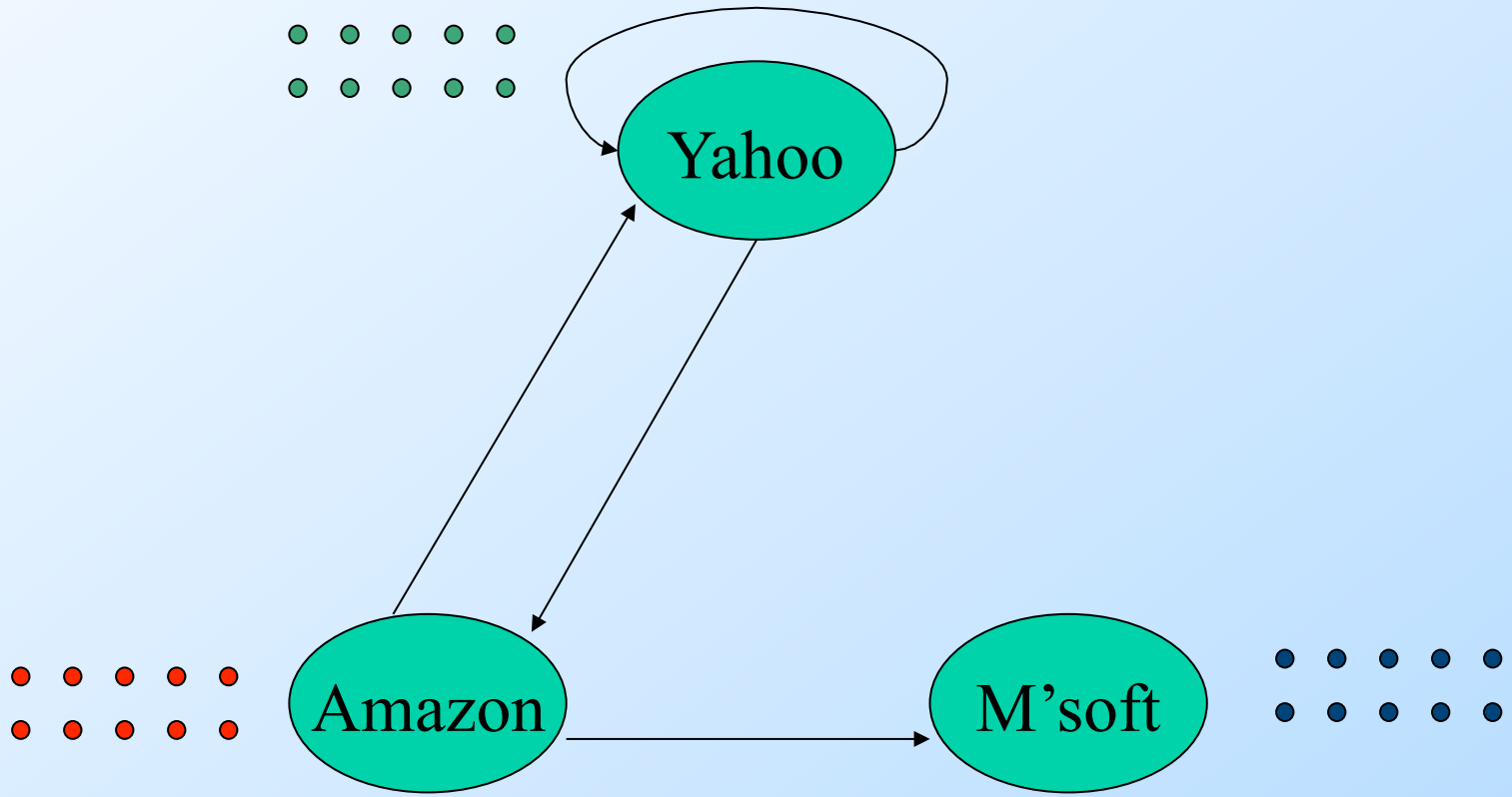
# In the Limit …



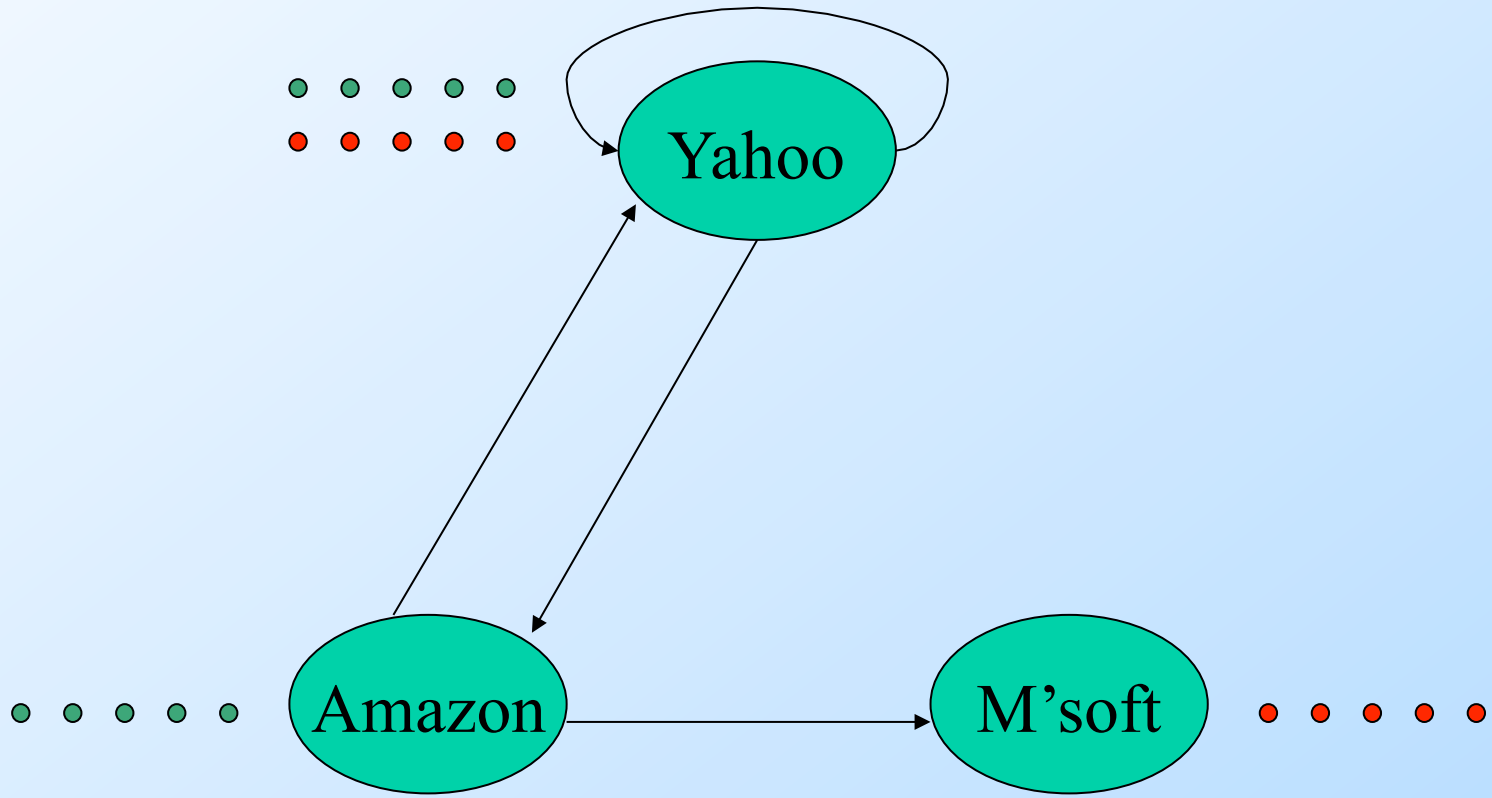
# Real-World Problems

- Some pages are "dead ends" (have no links out).
  - Such a page causes importance to leak out.
- Other (groups of) pages are *spider traps* (all out-links are within the group).
  - Eventually spider traps absorb all importance.

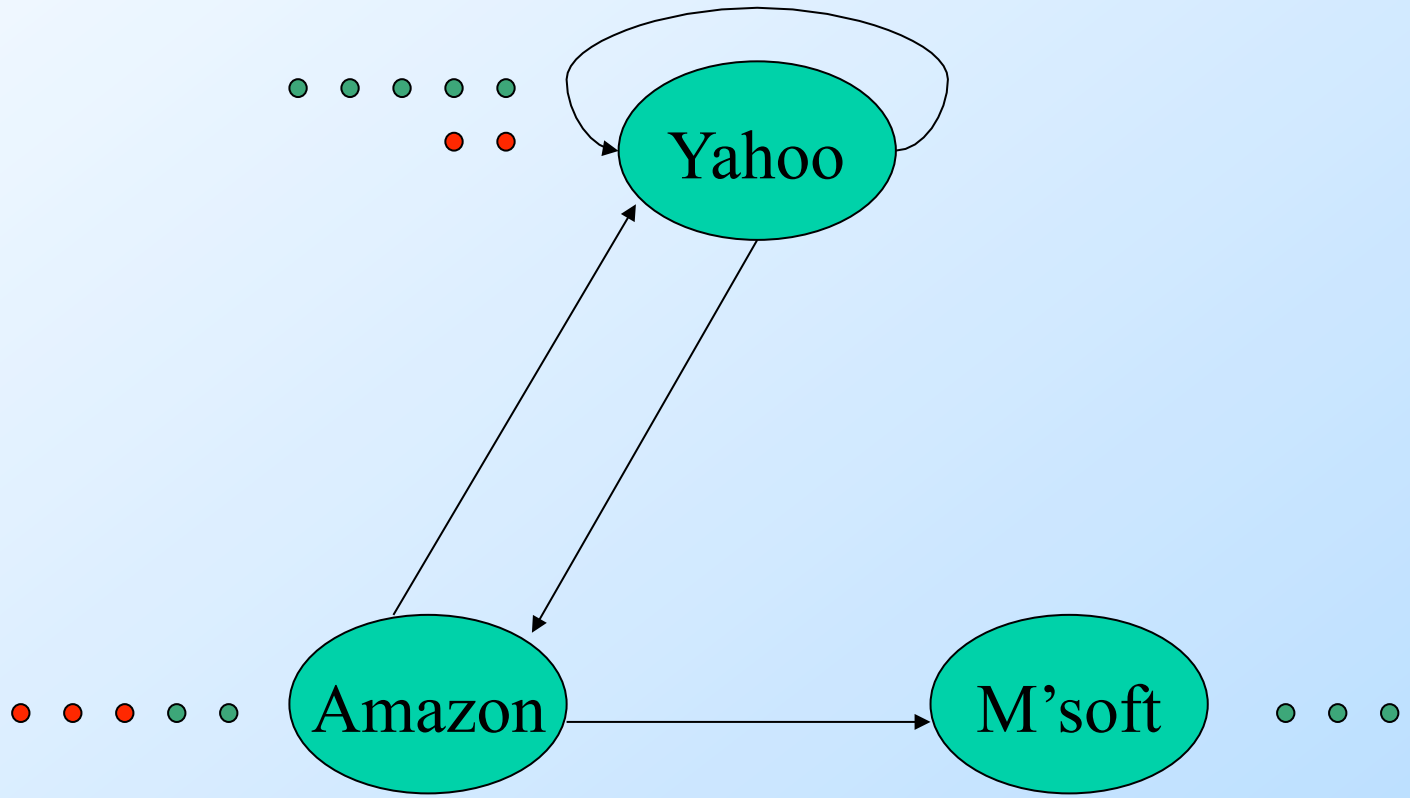
# Microsoft Becomes a Dead End



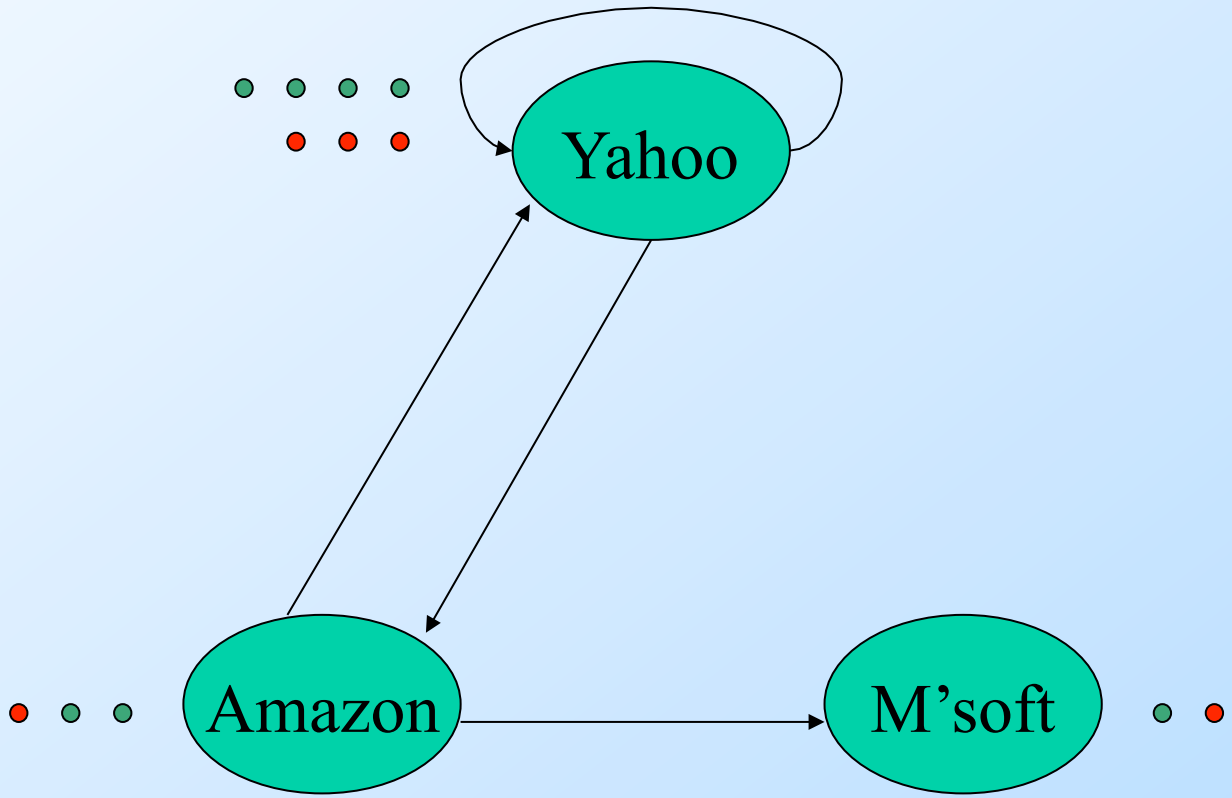
# Microsoft Becomes a Dead End



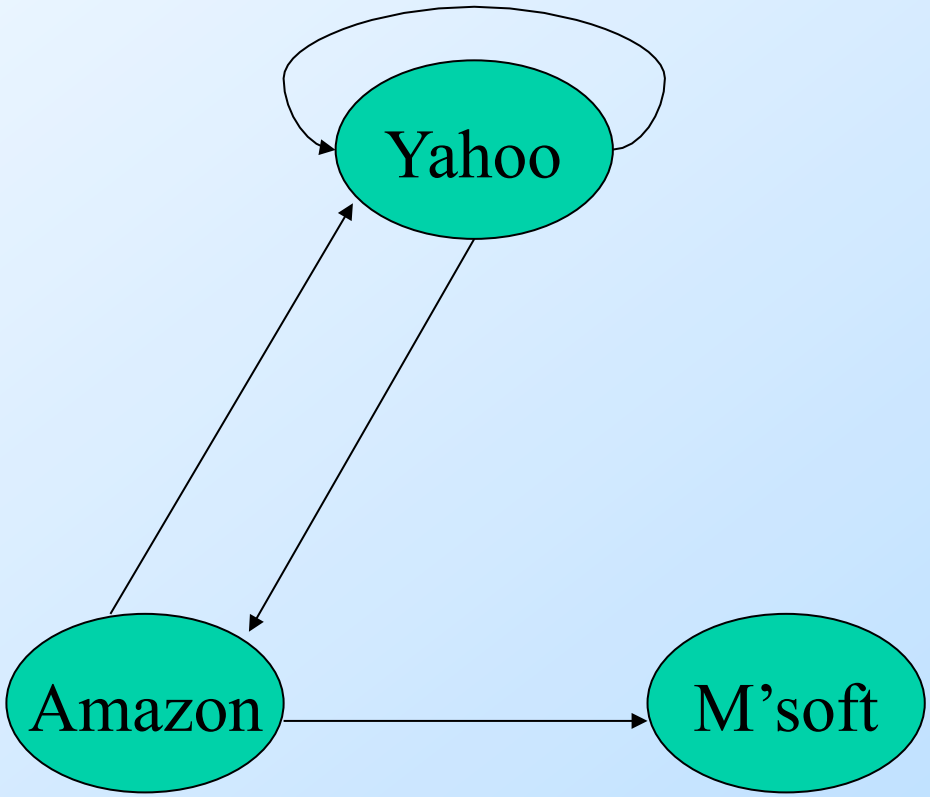
# Microsoft Becomes a Dead End



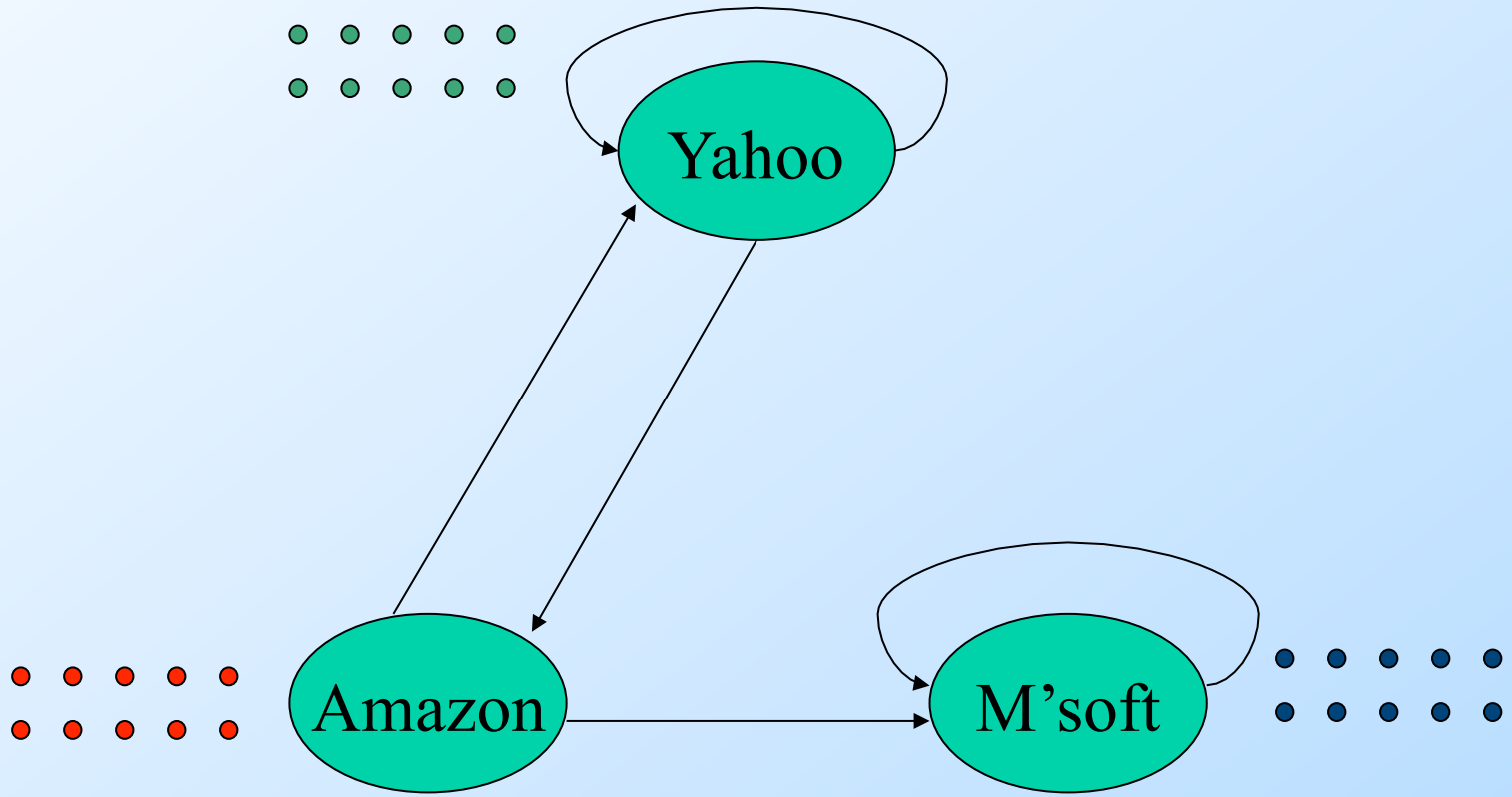
# Microsoft Becomes a Dead End



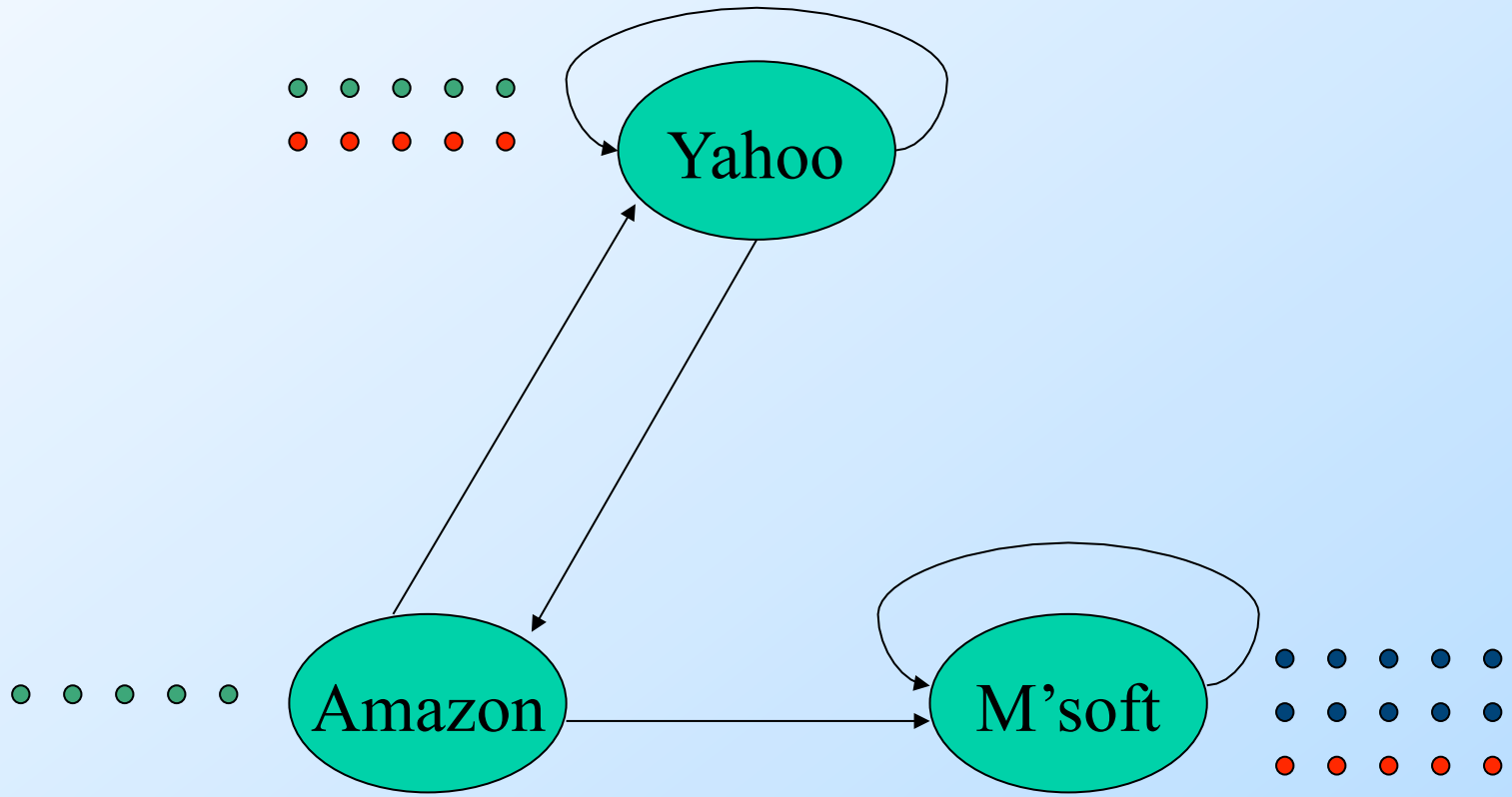
# In the Limit …



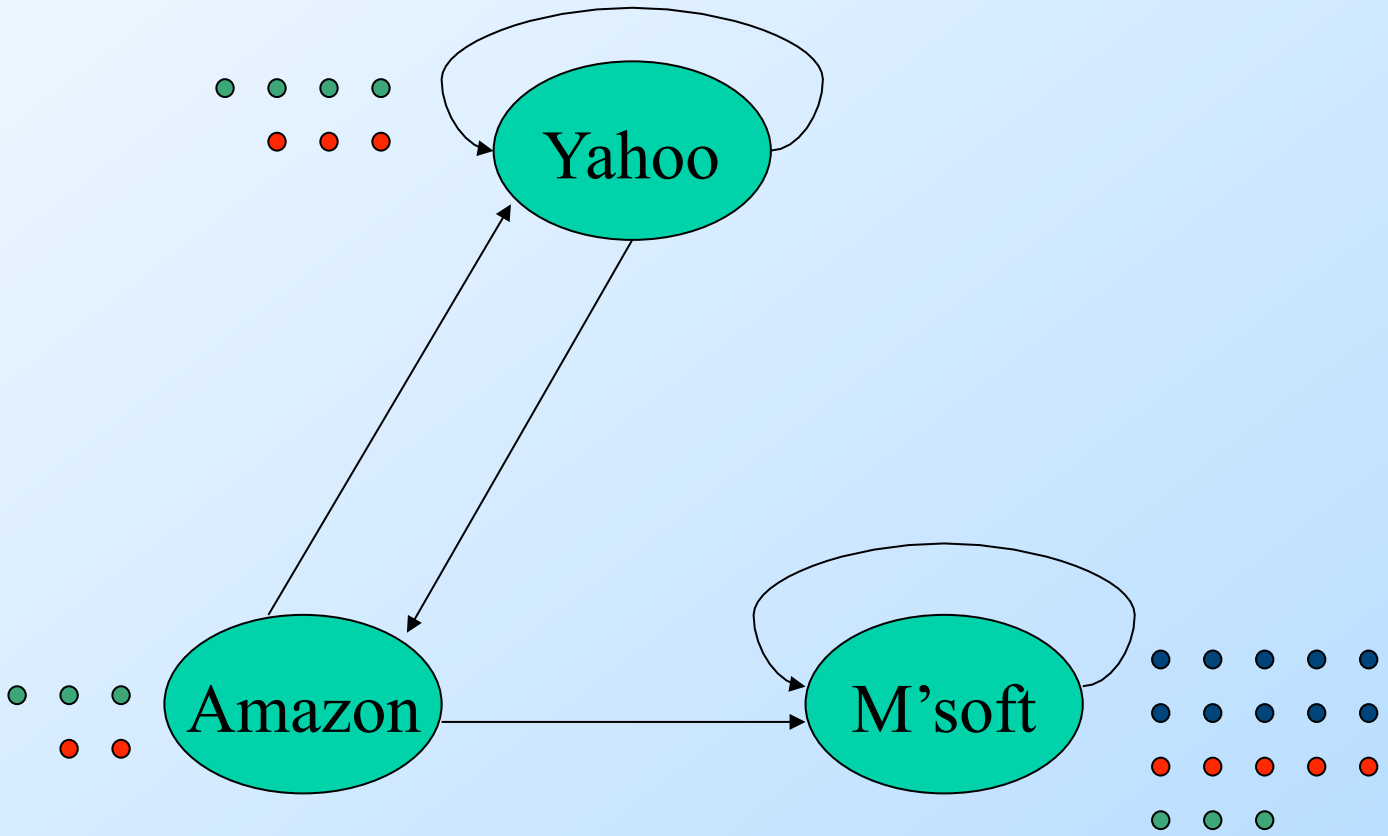
# Microsoft Becomes a Spider Trap



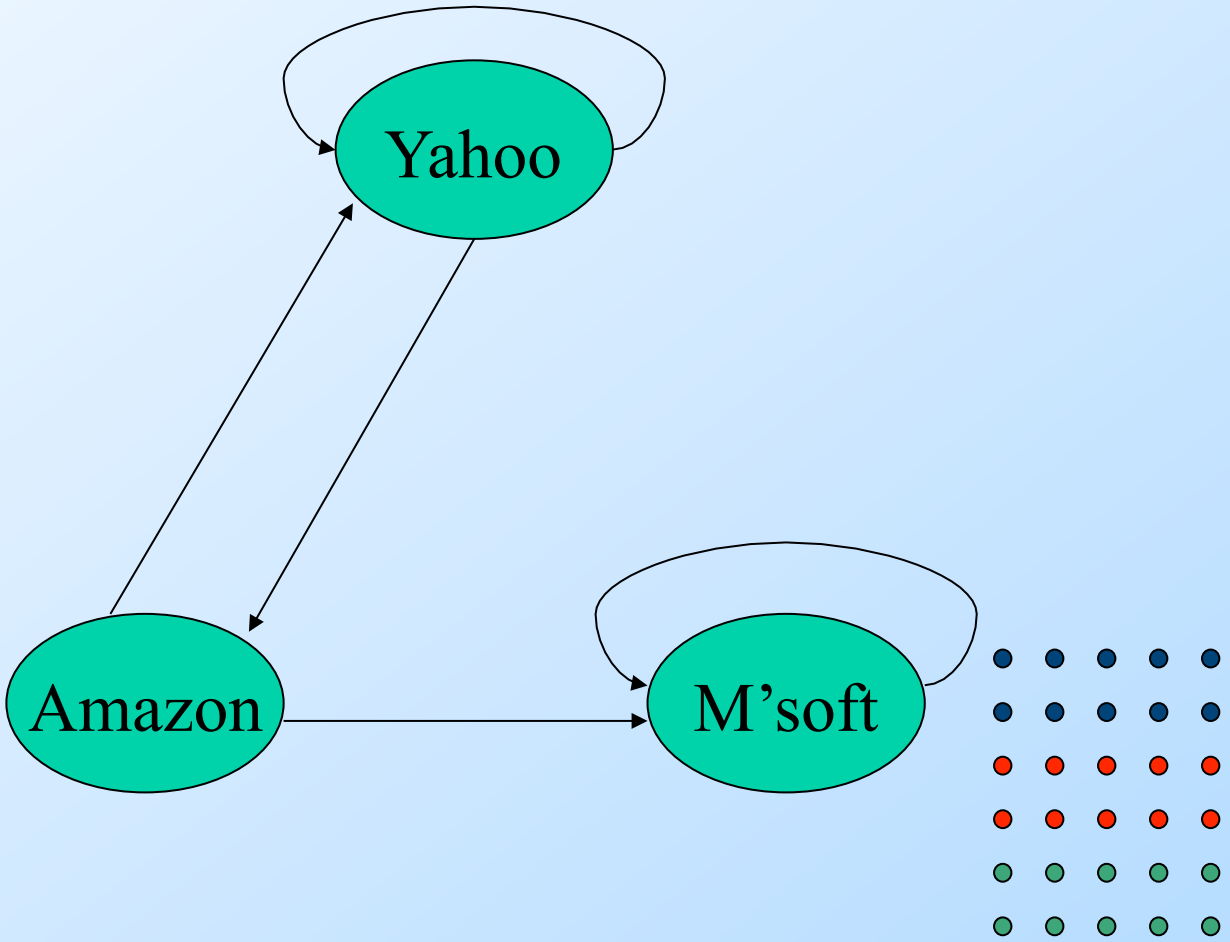
# Microsoft Becomes a Spider Trap



# Microsoft Becomes a Spider Trap



# In the Limit …



# Topic-Specific Page Rank

- Goal: Evaluate Web pages not just according to their popularity, but by how close they are to a particular topic, e.g. "sports" or "cooking."
- Allows search queries to be answered based on interests of the user.
  - Example: Query `batter` wants different pages depending on whether you are interested in sports or cooking.

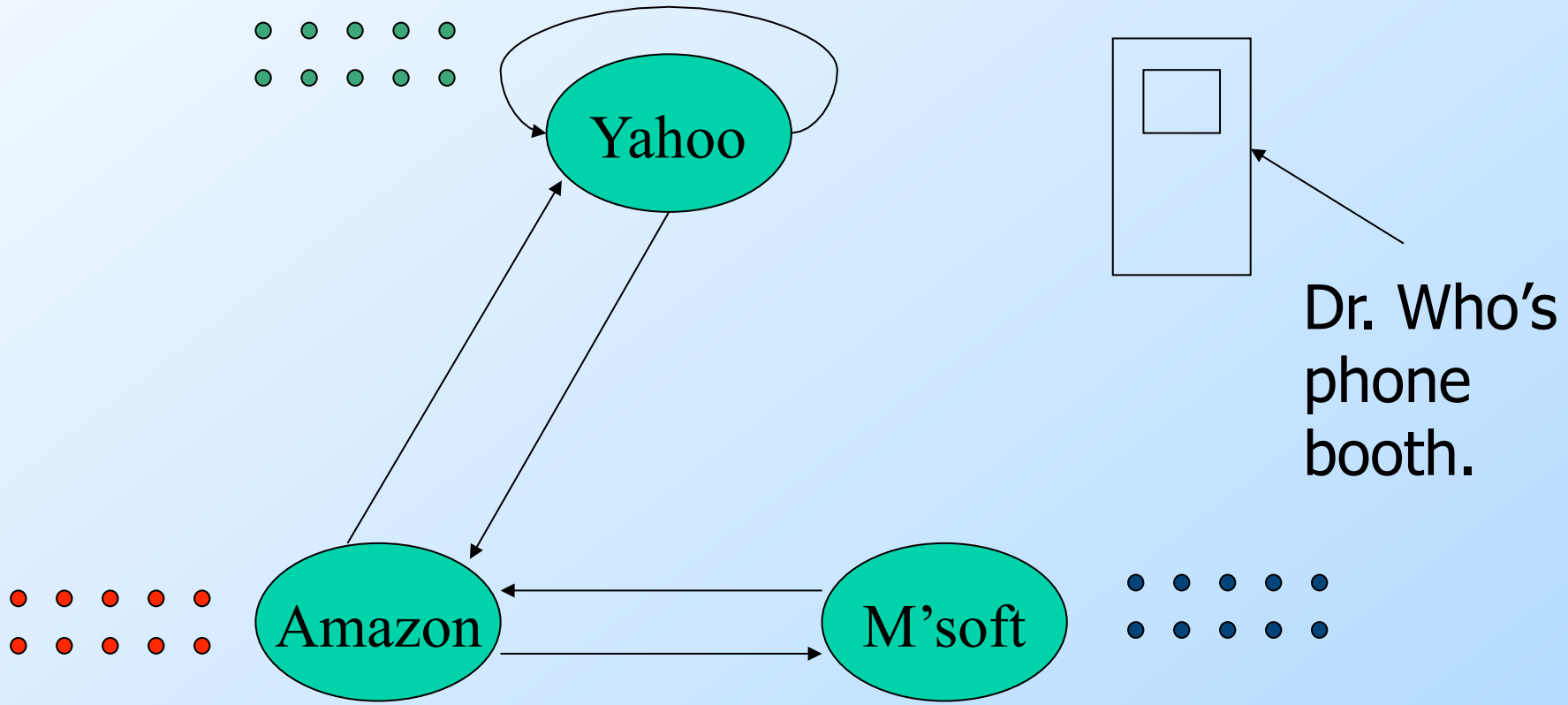
# Teleport Sets

- Assume each walker has a small probability of "teleporting" at any tick.
- Teleport can go to:
  1. Any page with equal probability.
     - To avoid dead-end and spider-trap problems.
  2. A topic-specific set of "relevant" pages (*teleport set* ).
     - For *topic-specific* PageRank.

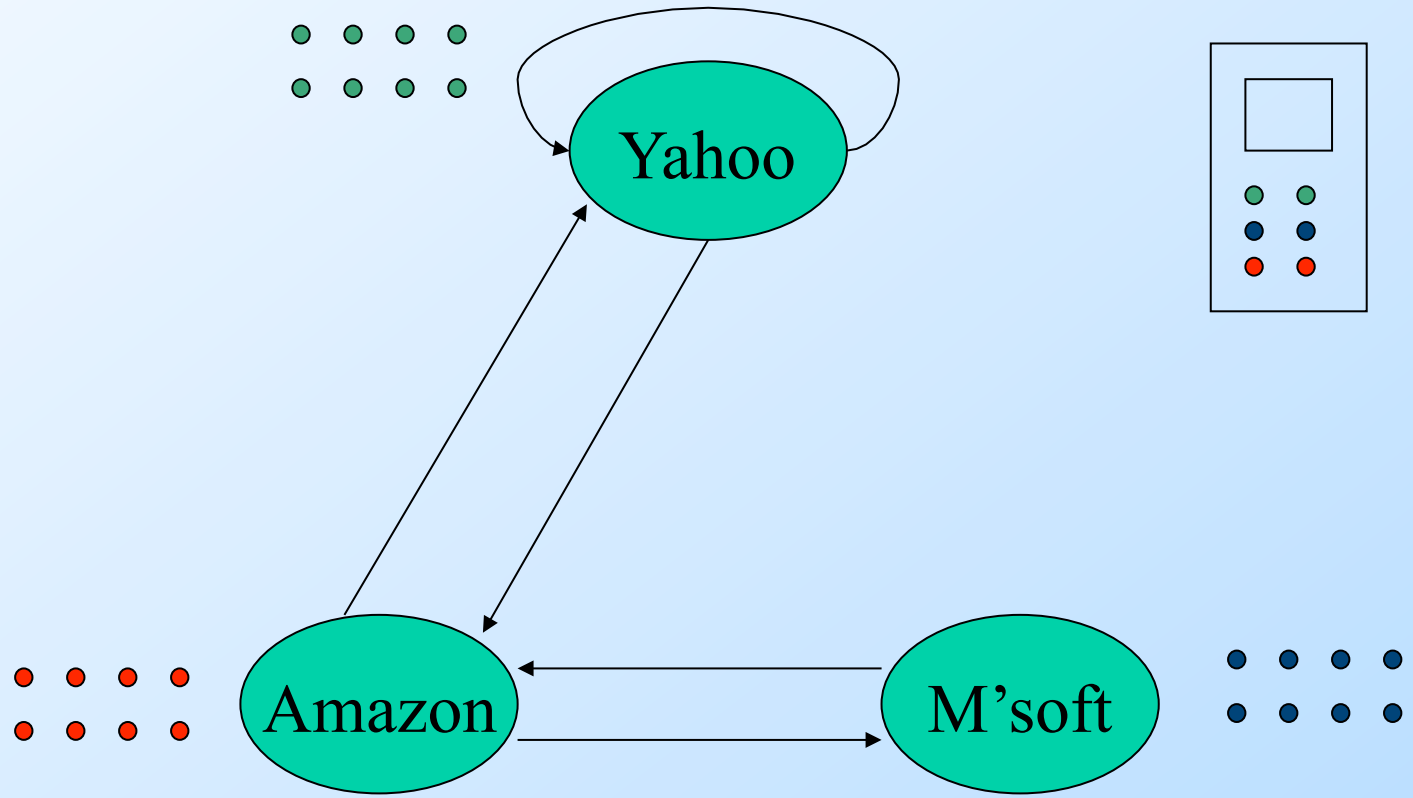
# Example: Topic = Software

- Only Microsoft is in the teleport set.
- Assume 20% "tax."

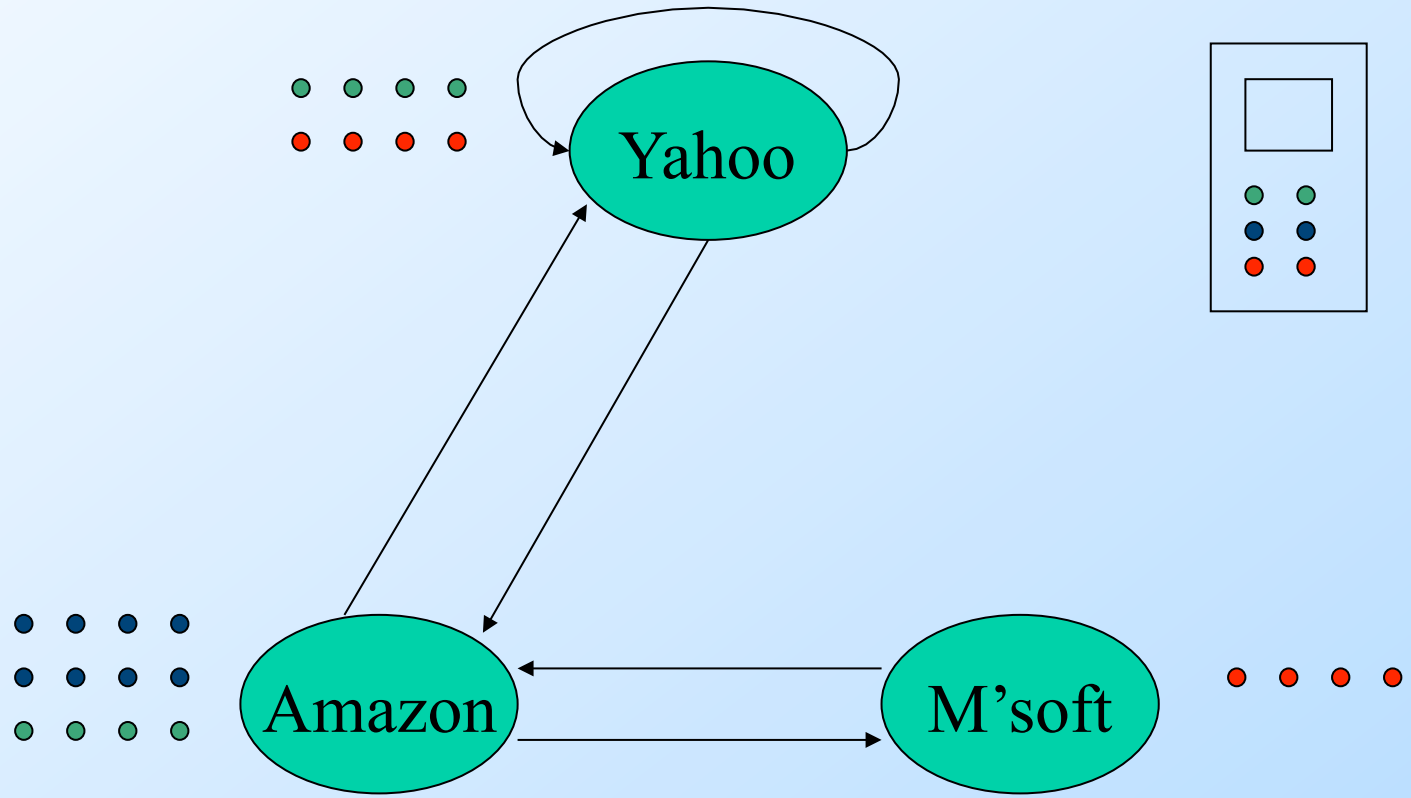
# Only Microsoft in Teleport Set



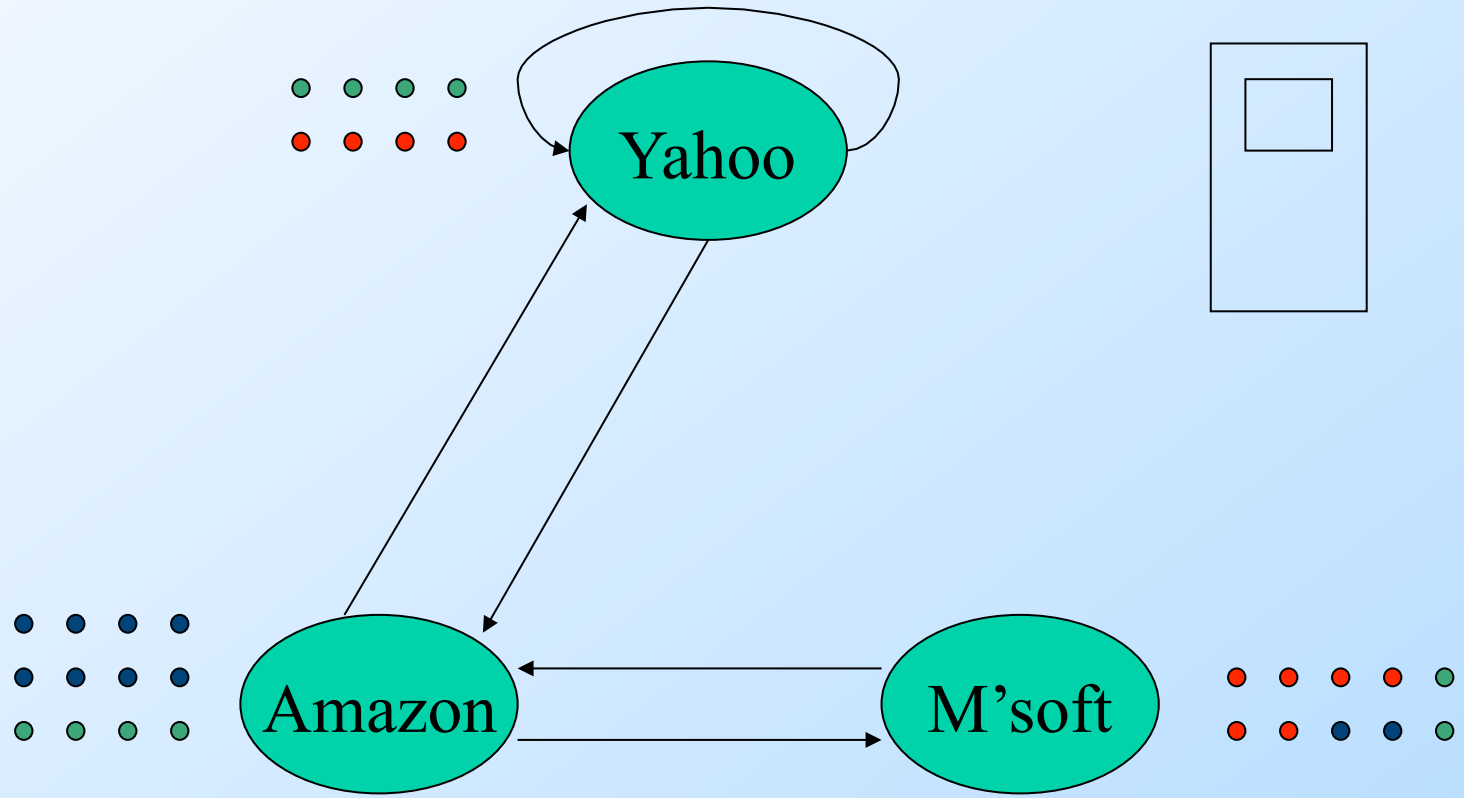
# Only Microsoft in Teleport Set



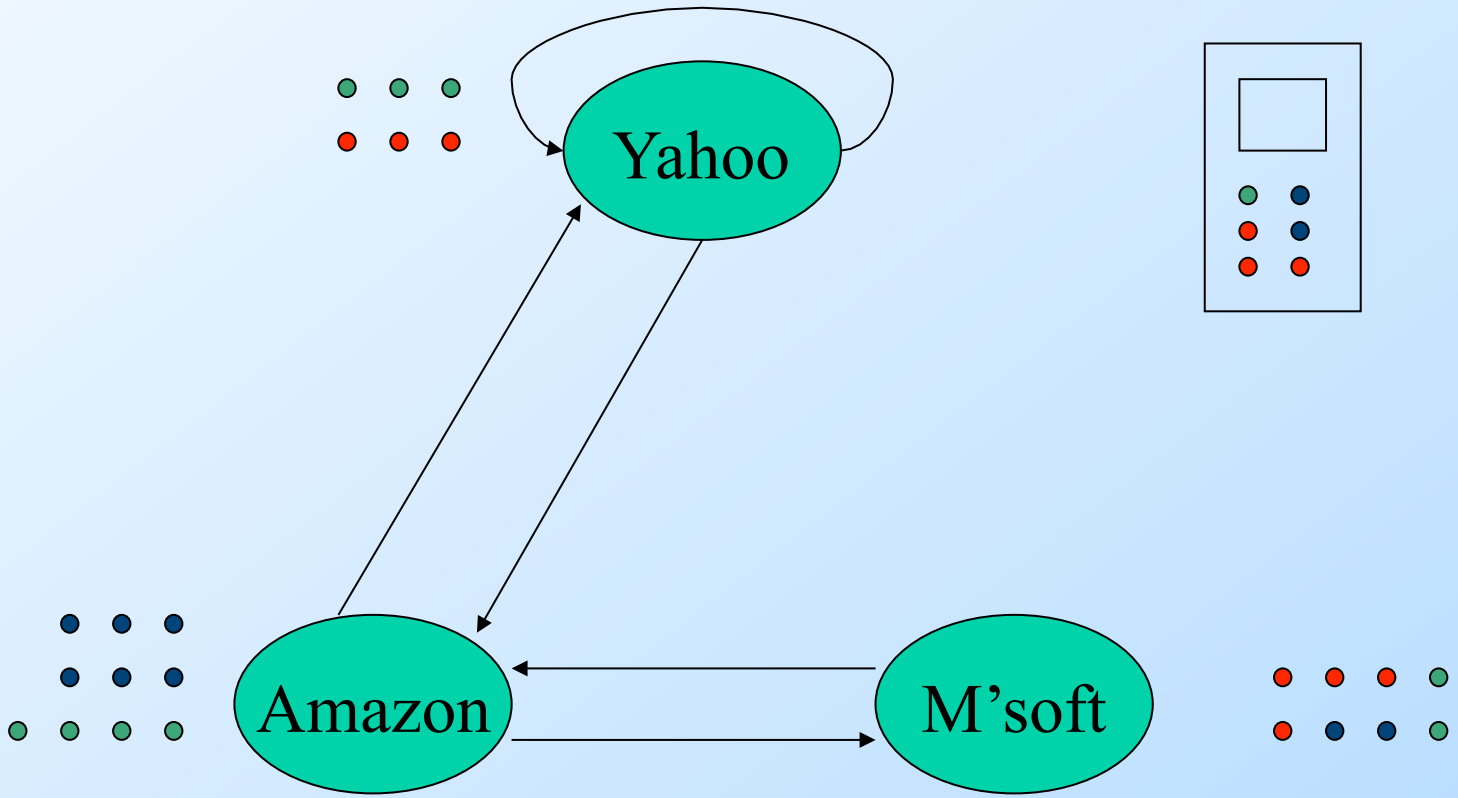
# Only Microsoft in Teleport Set



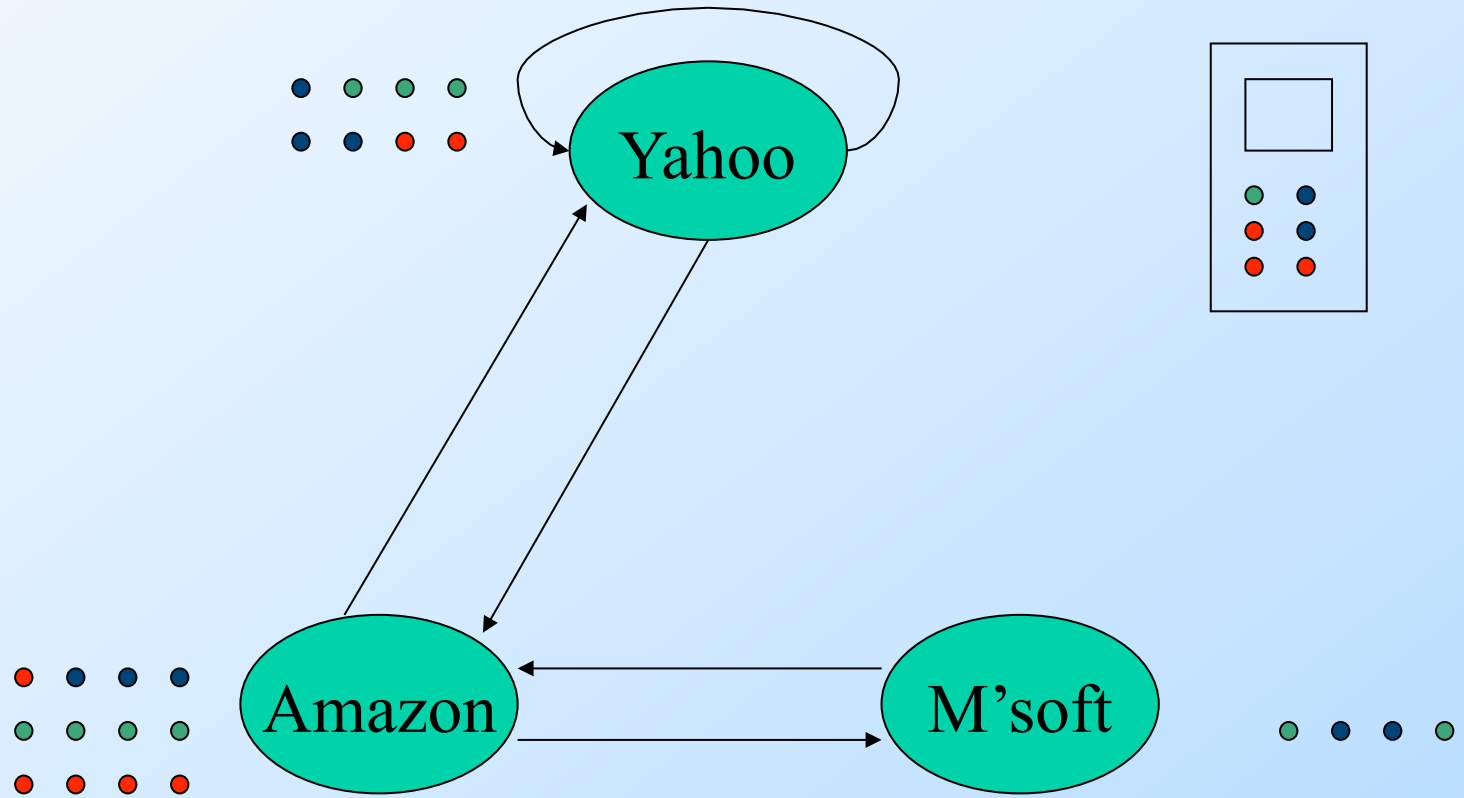
# Only Microsoft in Teleport Set



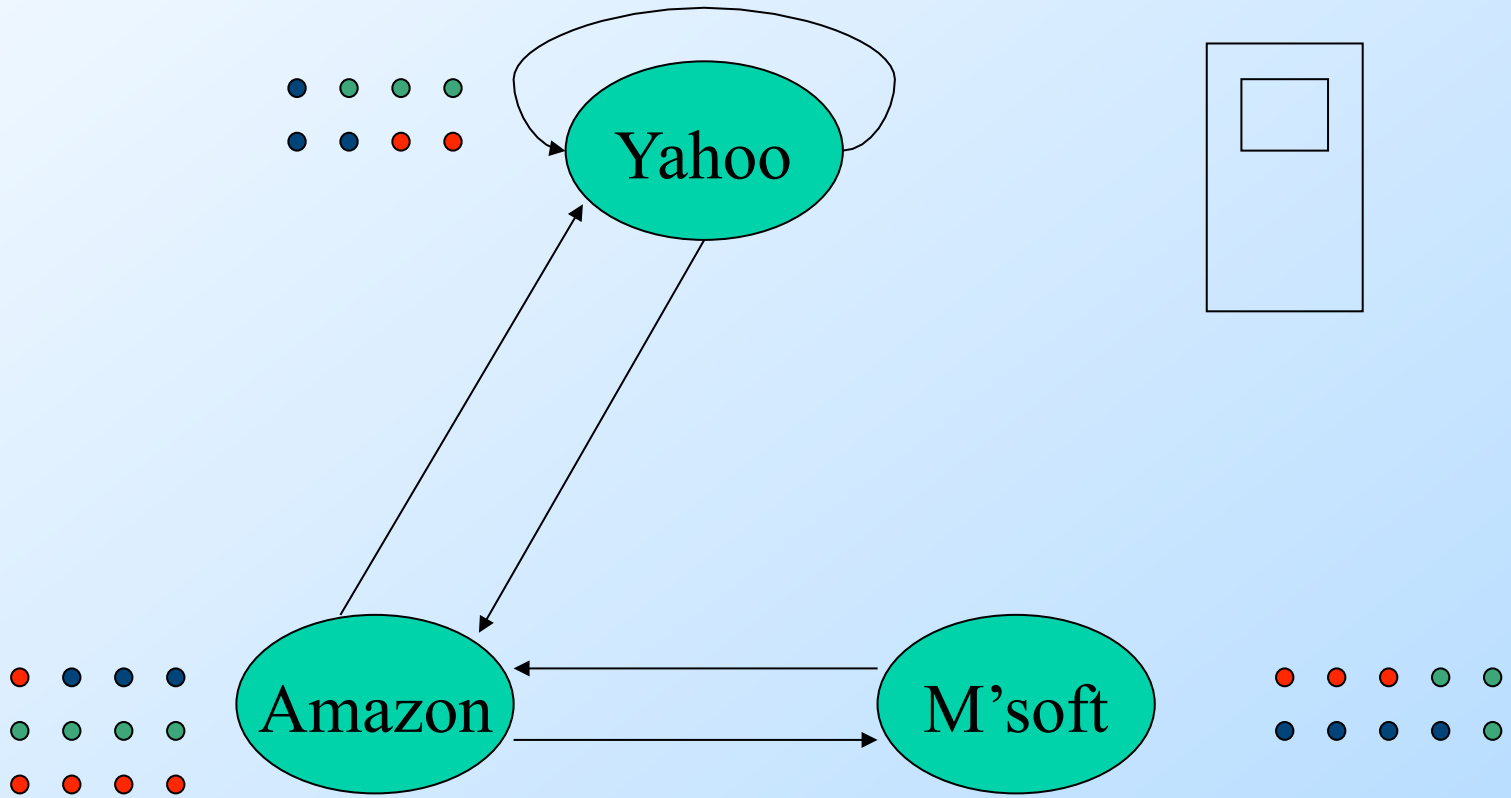
# Only Microsoft in Teleport Set



# Only Microsoft in Teleport Set



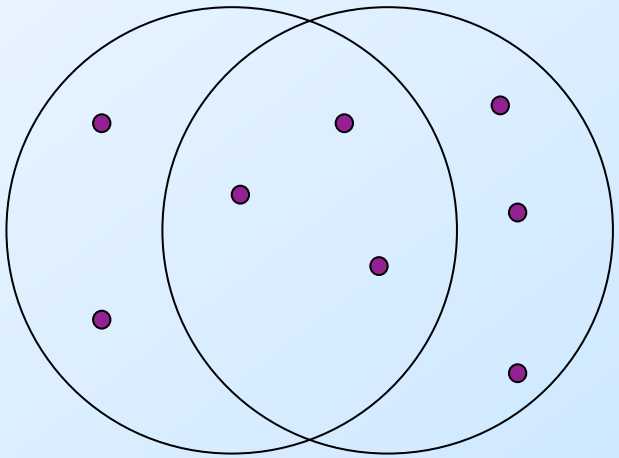
# Only Microsoft in Teleport Set



# New Topic: Similarity Search

- Many "objects" that populate overlapping sets.
- Find the pairs of sets that are "similar."
  - *Jaccard similarity* of sets = size of intersection divided by size of union.

# Example: Jaccard Similarity



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

# Applications

1. *Collaborative Filtering* : Represent Amazon customers by the sets of products they buy.
   - Recommend what similar customers bought.
2. *Similar Documents* : Represent pages by their sets of *k-shingles* = strings of *k* consecutive characters.
   - Similar pages could be plagiarism.

# When Is the Problem Interesting?

1. When the sets are so large or so many that they cannot fit in main memory.
2. When there are so many sets that comparing all pairs of sets takes too much time.

# Key Ideas

1. *Minhashing* : (Edith Cohen, Andrei Broder) Construct small *signatures* for sets so that the Jaccard similarity of sets can be determined from the signatures.
2. Locality-Sensitive Hashing : (Rajeev Motwani, Piotr Indyk) Focus on pairs of (likely) similar sets without looking at all pairs.

# Minhashing as a Matrix Problem

- Think of sets represented by a matrix of 0's and 1's.
- Row = element.
- Column = set.
- 1 means that element is in that set.

# Example

| | $C_1$ | $C_2$ | |
|---|---|---|---|
| a | 0 | 1 | * |
| b | 1 | 0 | * |
| c | 1 | 1 | * * |
| d | 0 | 0 | |
| e | 1 | 1 | * * |
| f | 0 | 1 | * |

$C_1 = \{b, c, e\}$
$C_2 = \{a, c, e, f\}$

*Sim* $(C_1, C_2) =$
$2/5 = 0.4$

# Four Types of Rows

- Given columns $C_1$ and $C_2$, rows may be classified as:

| | $C_1$ | $C_2$ |
|---|---|---|
| *a* | 1 | 1 |
| *b* | 1 | 0 |
| *c* | 0 | 1 |
| *d* | 0 | 0 |

- Also, *a* = # rows of type *a* , etc.
- Note *Sim* ($C_1$, $C_2$) = $a/(a+b+c)$.

# *Minhashing*

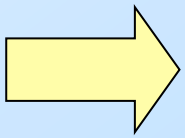
- Imagine the rows permuted randomly.
- Define "hash" function $h(C)$ = the number of the first (in the permuted order) row in which column $C$ has 1.
- Use several (100?) independent hash functions to create a signature.

# Minhashing Example

Input matrix

| | | |
|---|---|---|
| 1 | 4 | 3 |
| 3 | 2 | 4 |
| 7 | 1 | 7 |
| 6 | 3 | 6 |
| 2 | 6 | 1 |
| 5 | 7 | 2 |
| 4 | 5 | 5 |

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 |



Signature matrix *M*

| | | | |
|---|---|---|---|
| 2 | 1 | 2 | 1 |
| 2 | 1 | 4 | 1 |
| 1 | 2 | 1 | 2 |

# Surprising Property

- The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.
- Both are $a/(a+b+c)$! Why?
  - Look down columns $C_1$ and $C_2$ (in permuted order) until we see a 1.
  - If it's a type-*a* row, then $h(C_1) = h(C_2)$. If a type-*b* or type-*c* row, then not.

# Finding Similar Sets

- We can use minhashing to replace sets (columns of the matrix) by short lists of integers.
- But we still need to compare each pair of signatures.
- Example: 20 million Amazon customers; $2*10^{14}$ pairs of customers to evaluate.

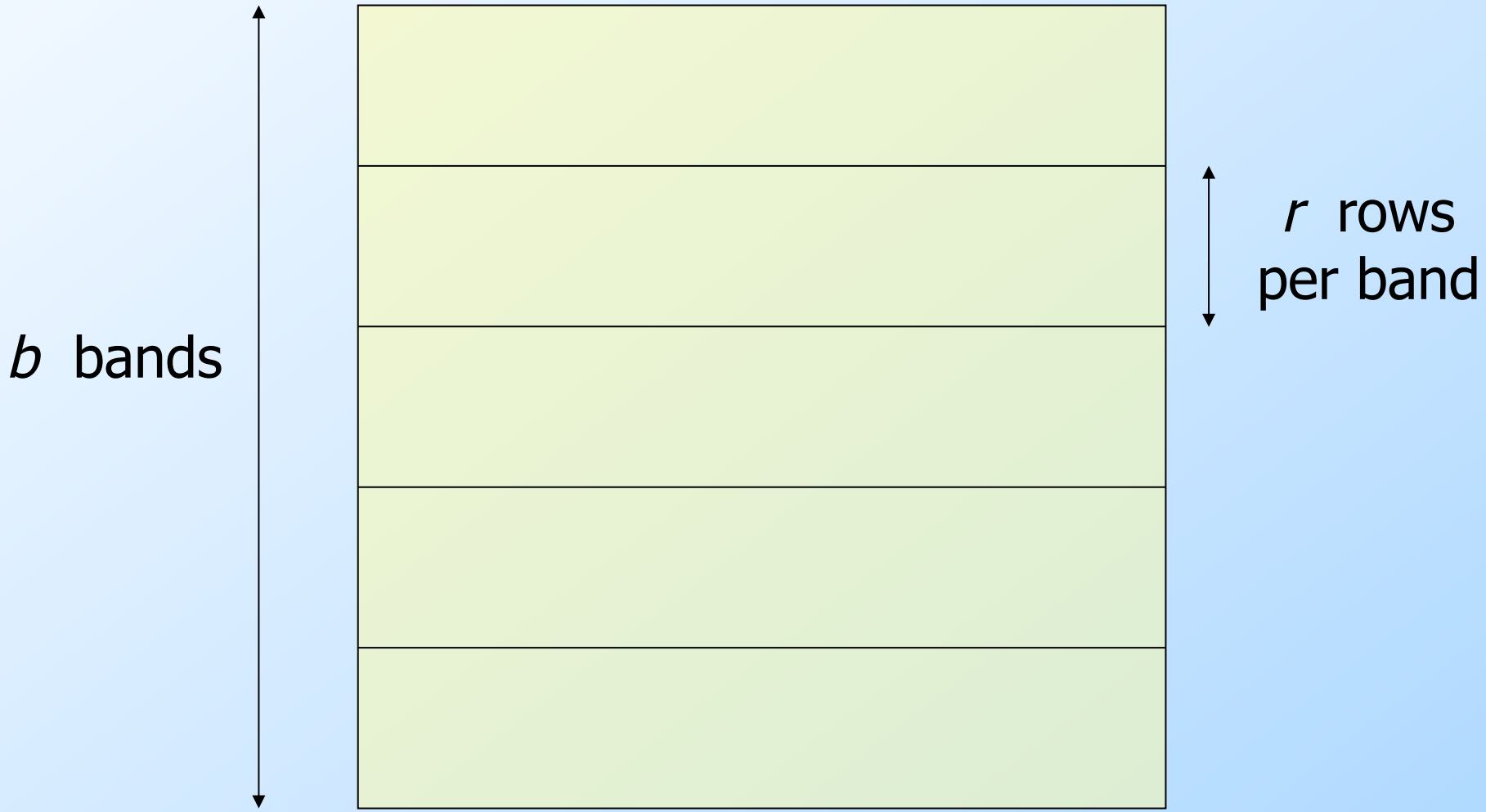
# Locality-Sensitive Hashing

- What we want seems impossible: map signatures to buckets so that
  1. Two similar signatures have a very good chance of appearing in the same bucket.
  2. If two signatures are not very similar, they probably don't appear in one bucket.
- Then, we only have to compare bucket-mates (*candidate pairs* ).

# The LSH Trick

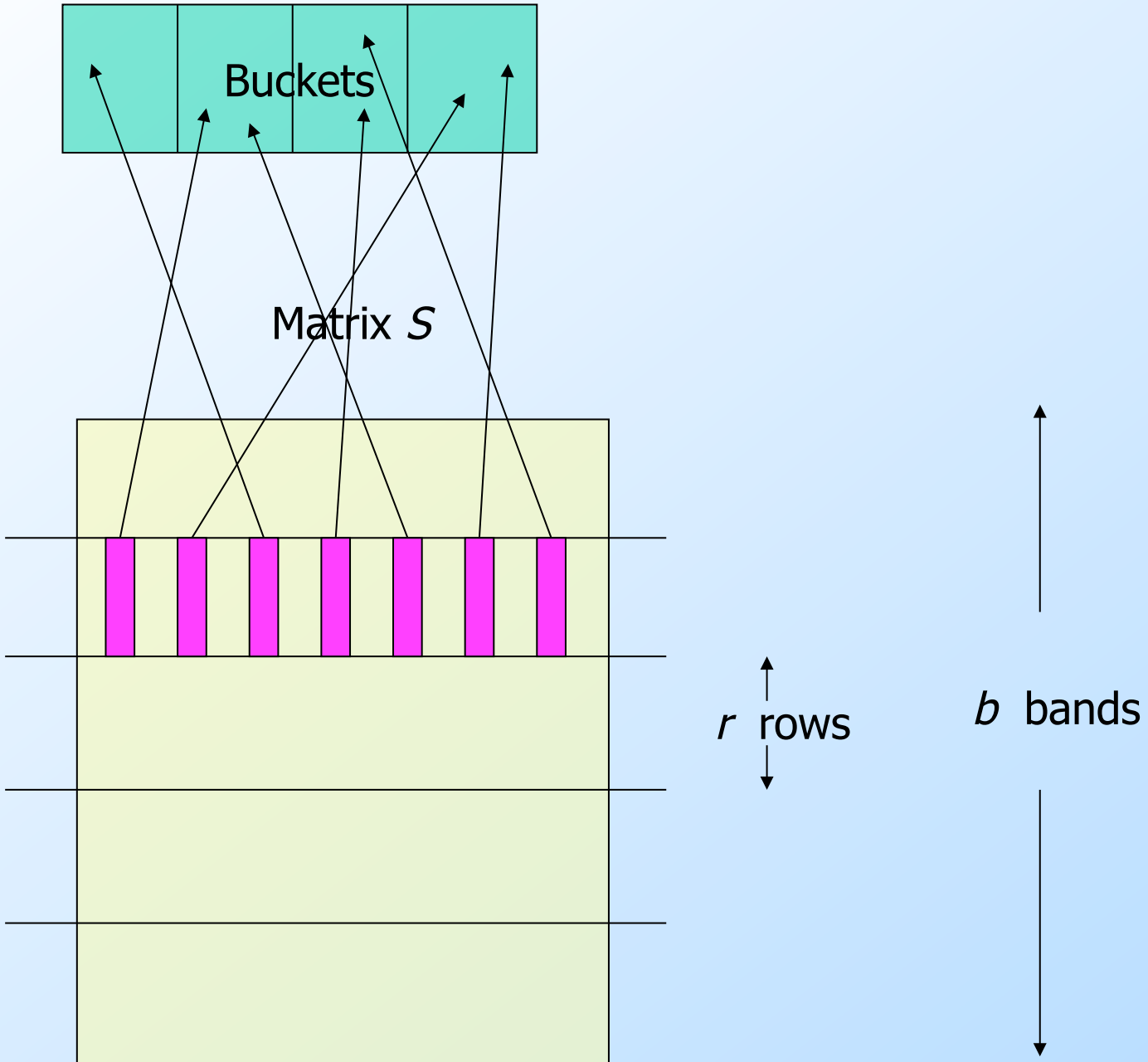
- Think of the signature for each column as a column of the *signature matrix* *S*.
- Divide the rows of *S* into *b bands* of *r* rows each.

# Partition Into Bands



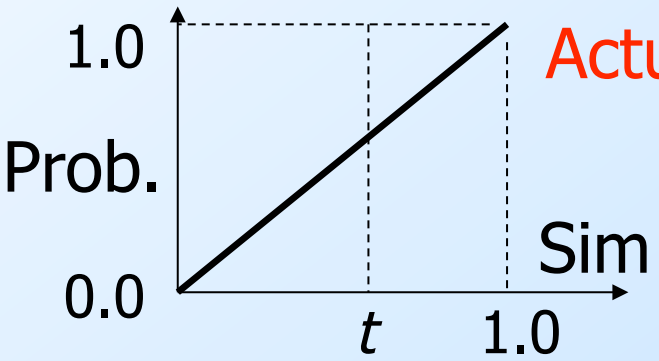
# Partition into Bands --- (2)

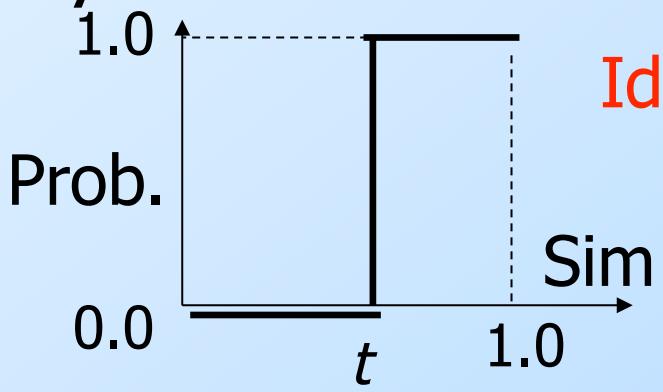
- For each band, hash its portion of each column to a hash table with many buckets.
- *Candidate* column pairs are those that hash to the same bucket for $\geq$ 1 band.
- Tune *b* and *r* to catch most similar pairs, but few nonsimilar pairs.


Buckets
Matrix S
r rows
b bands

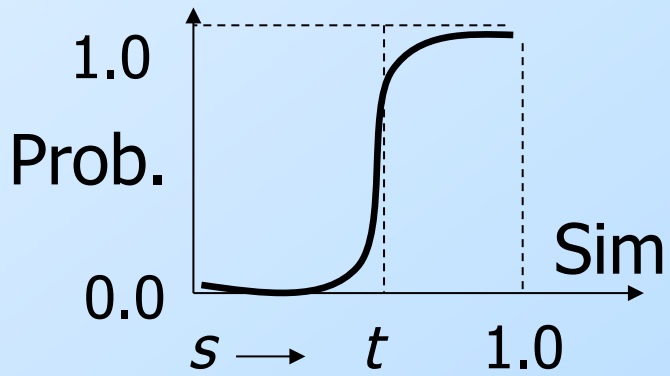
# LSH --- Graphically

- Example Target: All pairs with *Sim* > *t*.
- Suppose we use only one hash function:




- Partition into bands gives us:




$$1 - (1 - s^r)^b$$

$$t \sim (1/b)^{1/r}$$

# Summary of Minhash/LSH

1. Represent the objects you are comparing by sets (ad-hoc method).
2. Represent the sets by signatures (Minhashing).
3. Use LSH to create buckets; candidate pairs are those in the same bucket.
4. Evaluate only the candidate pairs.

# Experience

1. Finding news articles with the same source.
2. *Entity resolution* : finding customers shared by two businesses.

# News Sources

- Two members of the database group at Stanford were asked by the PoliSci Dept. to examine 1.5 million news articles and identify those that were really the same synticated article published by different newspapers.
  - Each newspaper "decorates" the article with its own material, e.g. masthead.

# News Sources – (2)

- They developed their own algorithm and reported it to the group.
- I suggested "minhashing + LSH."
- They reimplemented and found that minhash+LSH was faster and more accurate for all but very high degrees of similarity.

# Matching Customer Records

- I once took a consulting job solving the following problem:
  - Company A agreed to solicit customers for Company B, for a fee.
  - They then had a parting of the ways, and argued over how many customers.
  - Neither recorded exactly which customers were involved.

# Customer Records – (2)

- Company B had about 1 million records of all its customers.
- Company A had about 1 million records describing customers, some of which it had signed up for B.
- Records had name, address, and phone, but for various reasons, they could be different for the same person.

# Customer Records – (3)

- Step 1: design a measure of how similar records are:
  - E.g., deduct points for small misspellings ("Jeffrey" vs. "Geoffery"), same phone, different area code.
- Step 2: score all pairs of records; report very similar records as matches.

# Customer Records – (4)

- Problem: (1 million)$^2$/2 is too many pairs of records to score.
- Solution: A simple LSH.
  - Three hash functions: exact values of name, address, phone.
    - Compare iff records are identical in at least one.
  - Misses similar records with a small difference in all three fields.

# Customer Records – Aside

- We were able to tell what values of the scoring function were reliable in an interesting way.
  - Identical records had a creation date difference of 10 days.
  - We only looked for records created within 90 days, so bogus matches had a 45-day average.

# Aside – (2)

- By looking at the pool of matches with a fixed score, we could compute the average time-difference, say x, and deduce that fraction (45-x)/35 of them were valid matches.
- Alas, the lawyers didn't think the jury would understand.