Search Engine Architectures from Conventional to P2P

Torsten Suel
Yahoo! Research
Sunnyvale, CA 94089

Y!Research

Topic of this Talk

- large-scale search engines: conventional vs. P2P
- a "conventional" perspective

- different communities
  - IR, algorithms, databases, systems, networks, distrib. computing

- what can we learn from each other?
- what are the main challenges?
- what to do, what to avoid?

Note: the following are personal opinions by the author, and do not reflect those of Yahoo!. All statements are based on personal experience and the open literature, and nothing should be inferred from them about Yahoo!'s actual search services and internal system architecture.

Topic of this Talk

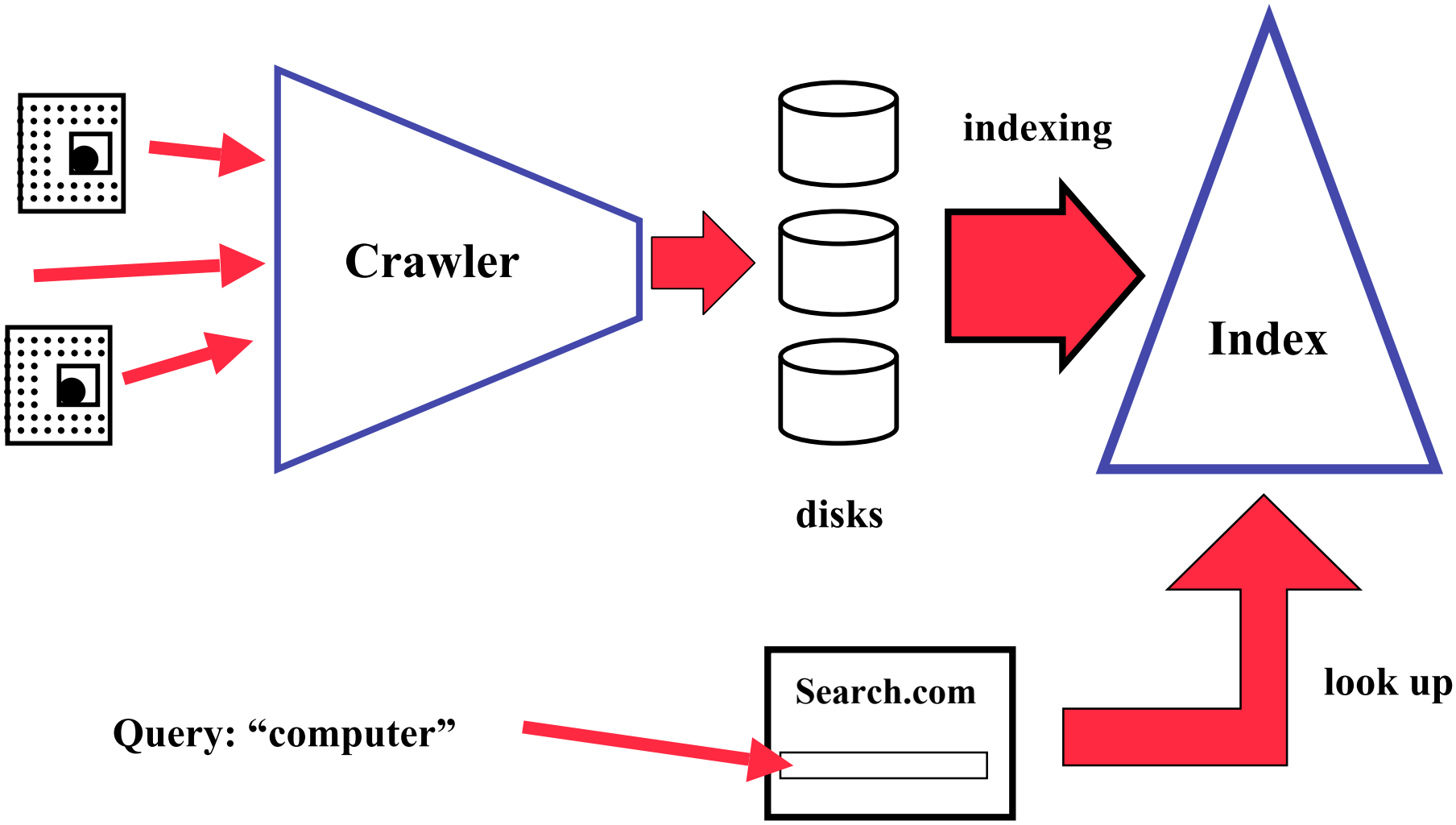
- focus: large scale (web) search
  - billions and terabytes of documents
  - textual -- not (just) terabytes of multi-media
  - noisy/open/unstructured collection (spam, informal content)
  - (usually) hosted outside the P2P network
- large scale P2P web search as a straw man
  - may not be realistic now (or ever?)
  - but interesting to study!
- many other applications for IR in P2P

Talk Outline

- conventional search architecture: a (mis)guided tour
- discussion of P2P approaches and perspectives
- current challenges:
  - P2P data mining: platforms and algorithms
  - query processing: index optimizations, early termination, etc.
- conclusions

Conventional Search Engine Architecture: A (Mis)Guided Tour

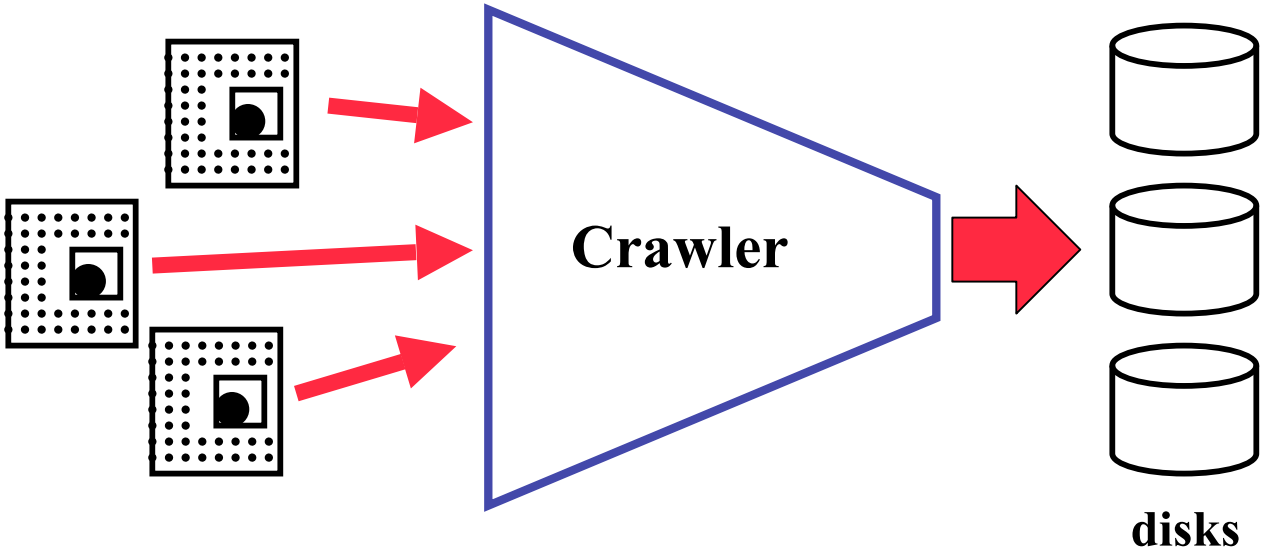
Search Engine Architecture Overview



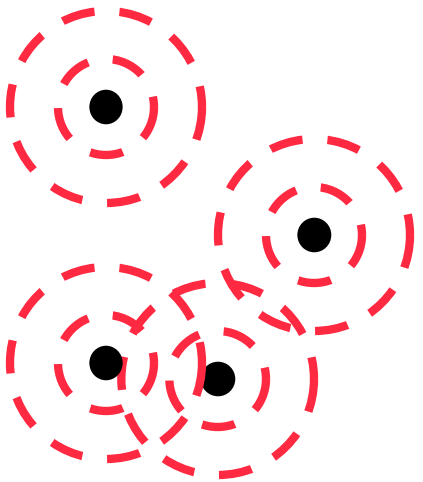
Search Engine Architecture Overview

- data acquisition -- web crawling
- data analysis -- text, link, and log mining
- inverted index building and updates
- query processing

Data Acquisition -- Web Crawling



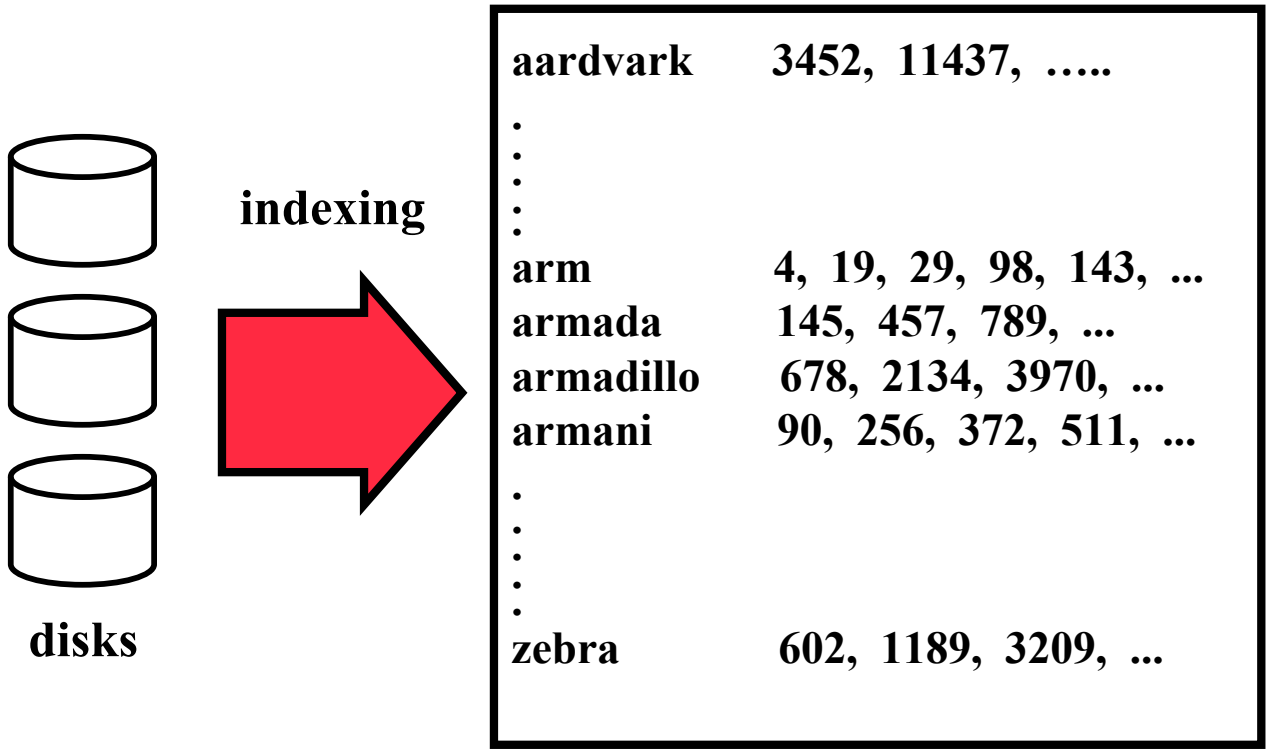
- follow hyperlinks to download pages
- BFS (not), refresh, focus on quality
- performance, policy, management
- IRLBot, Atrax/Mercator, PolyBot



Data Analysis -- Web and Log Mining

- spam detection
- (near) duplicate detection
- site structure analysis
- link analysis -- e.g., Pagerank
- query log mining -- e.g., click-through analysis
- ... and many other things ...
- prepares input for crawling and query processing
- typically based on mapReduce type computing

Index Construction and Updates



inverted index

- build inverted index structure
- in bulk, similar to mining
- but updates trickier

Query Processing

Boolean queries:

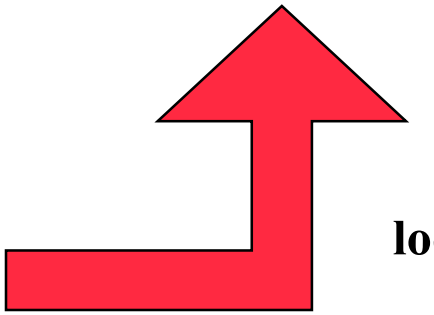
(zebra AND armadillo) OR armani

➡ **compute unions/intersections of lists**

Ranked queries: *zebra, armadillo*

➡ **give scores to all docs in union**

aardvark	3452, 11437, …..
⋮	
arm	4, 19, 29, 98, 143, ...
armada	145, 457, 789, ...
armadillo	678, 2134, 3970, ...
armani	90, 256, 372, 511, ...
⋮	
zebra	602, 1189, 3209, ...



look up

Search Engine Architecture -- Discussion

Search Engine Architecture -- 1995

(handful of people)

- data acquisition
- data analysis
- index building
- query processing

Search Engine Architecture -- 2000

(dozens of people)

- data acquisition
- data analysis
- index building
- query processing

Search Engine Architecture -- 2007

(hundreds of people)

- data acquisition

- **data analysis**

- index building

- **query processing**

Three Different System Setups

- main tasks: crawling, mining, query execution
- all based on large clusters of low-cost machines
- but different systems:
  - different workloads
  - different programming environments (e.g., mapReduce)
  - different scaling behavior (collection size, query load, etc.)
  - different optimal hardware balance
- no reason to treat as one unit in P2P! (?)

Data Mining

- steadily increasing in importance (e.g., insecure sites)
- many new tasks (spam etc.)
- plus moving some tasks from crawling and processing
- to support smarter crawling strategies and ranking
- bulk-computing via mapReduce (Hadoop)
- giant data mining platform at heart of engines
- now most of the effort ....
- ... and how about P2P?

Crawling

- three aspects:
  - performance
  - strategy/policy: what to crawl when & how often
  - management: robots, politeness, resilience, control
- most published work focused on first two
- do not neglect the last! (social capital on the web)
- do not put too much of the second *inside* the crawler
- dealing with millions of web sites (and their owners!)
- is P2P crawling really a good idea?

Search Engine Ranking

- Cosine, BM25?

$$BM25(q,d) = \sum_{t \in q} \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \times \frac{(k_1 + 1) f_{d,t}}{K + f_{d,t}}$$

- add Pagerank! (but how?)
- add distances
- add ...

- real engines: machine-learned ranking (MLR) with 100s of features
- search engine "feature zoo"
- ... are simple functions irrelevant?

Search Engine Ranking

- are simple functions irrelevant? NO!!
- but how to efficiently evaluate scoring with 100s of features?
- one approach: query processing in phases
  - use a simple function as filter
  - fully score only small subset
- e.g., top-100 BM25 give top-10 BM25+distance with p=0.98
- simple functions are great to approximate full MLR scoring
- but then it's not enough to get top-4 or top-10 on simple

- but explicit "sort of complicated" functions not so useful?

Search Engine Numbers

- know thy numbers ...
- search engines are big: many billions of docs, thousands of machines, thousands of queries per sec
- but search engines are freaking fast ..
- among the most optimized systems out there
- TREC GOV2: 25.2 M docs, a few ms/query on 1 core
  (look at recent years' published TREC numbers)
- beat the cr*p out of a database (except monet ...)
- P2P: tough to compete with 16 quad-core systems

Search Engine Queries

- queries are awesome ...

- hard to get -- SEs take protecting these very serious
- everyone should have looked at some *public* traces
- ... not for entertainment, but to learn
- so much variety: search strategies, interests, ...
- head, torso, tail queries
- don’t make assumptions about how people search ...

Query Processing Performance Challenge

- query processing challenge: how to answer 1000s of queries per second on billions of documents at acceptable hardware cost
- naive solution has cost proportional to

  numQueries * collectionSize
- how can we do better?
- how to scale out to hundreds of billions or trillions of pages, and to ever increasing query loads

Query Processing Performance Challenge

Optimization Techniques

- index compression
- index and result caching
- parallel computing
- early termination and related techniques
  - early termination
  - index tiering
  - index pruning
  - index specialization (?)

Early Termination in a Nutshell

"**early termination techniques** speed up query processing by avoiding a complete traversal of the inverted index structures for all query terms"

"**index tiering** is a special case of early termination that organizes the index into a discrete number of fixed tiers and then accesses a subset of these tiers"

"**index pruning** reduces index size by throwing away some postings, thus thinning out some documents"

"**index specialization** means building a much larger index that has small specialized structures that are highly efficient on certain types of queries" (?)

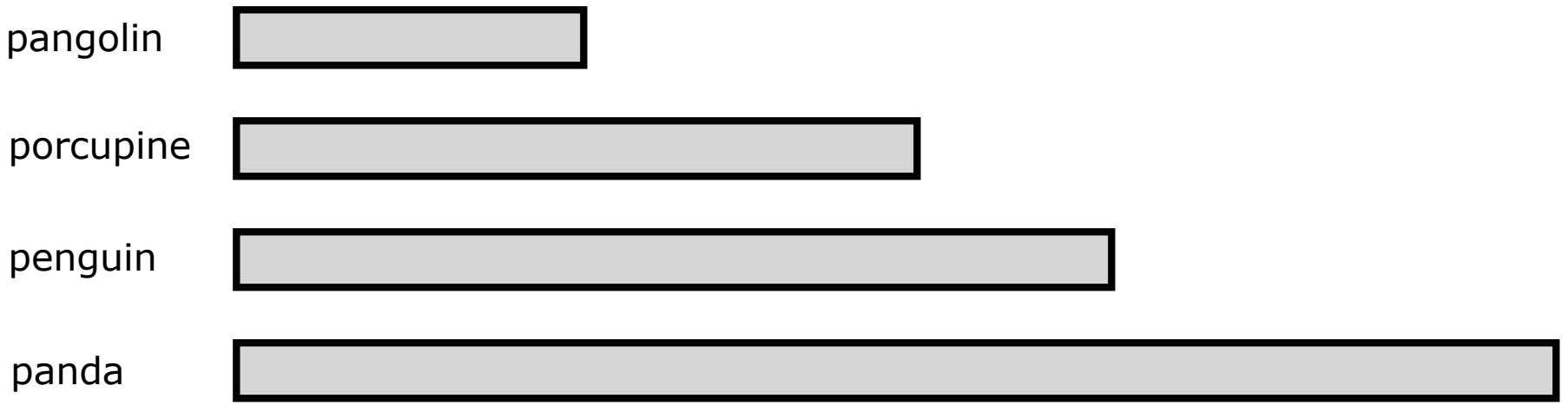
Assumptions/Perspective

Given a ranking function f() for our document collection, our goal is to **approximate** this function without traversing all the index structures

- goal: get **same** quality at **lower** cost
- not: improve quality
- we assume others **fix** the ranking function
- ranking function could be explicit formula, machine learned score, or black box ranking
- approximate could mean **guarantee same results**, or **mostly same results**, or **same precision** etc.

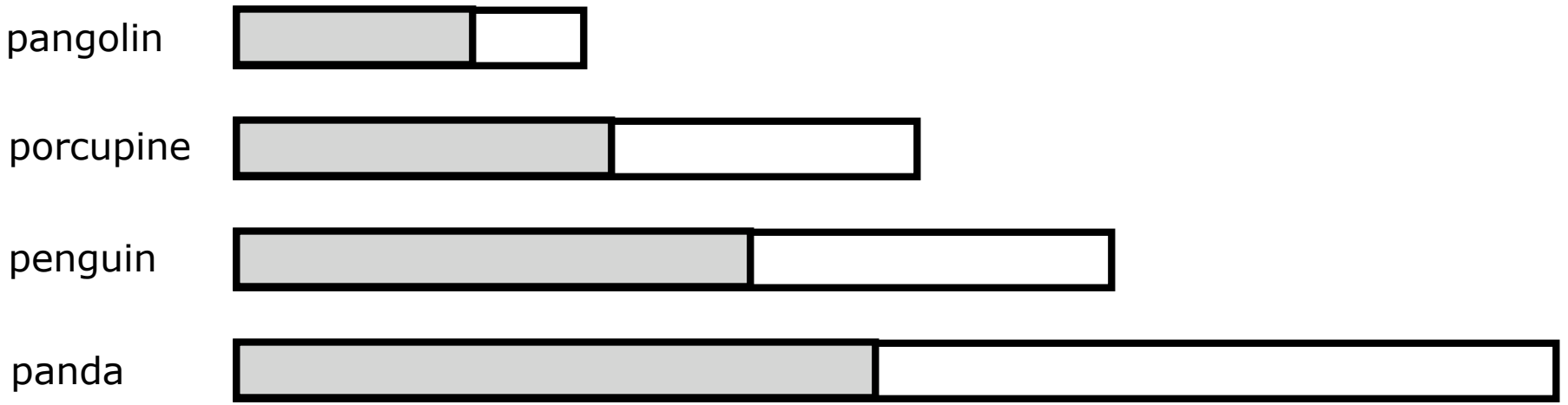
Early Termination: Basic Setup

- Given a query q and the inverted lists associated with that query:



Early Termination: Basic Setup

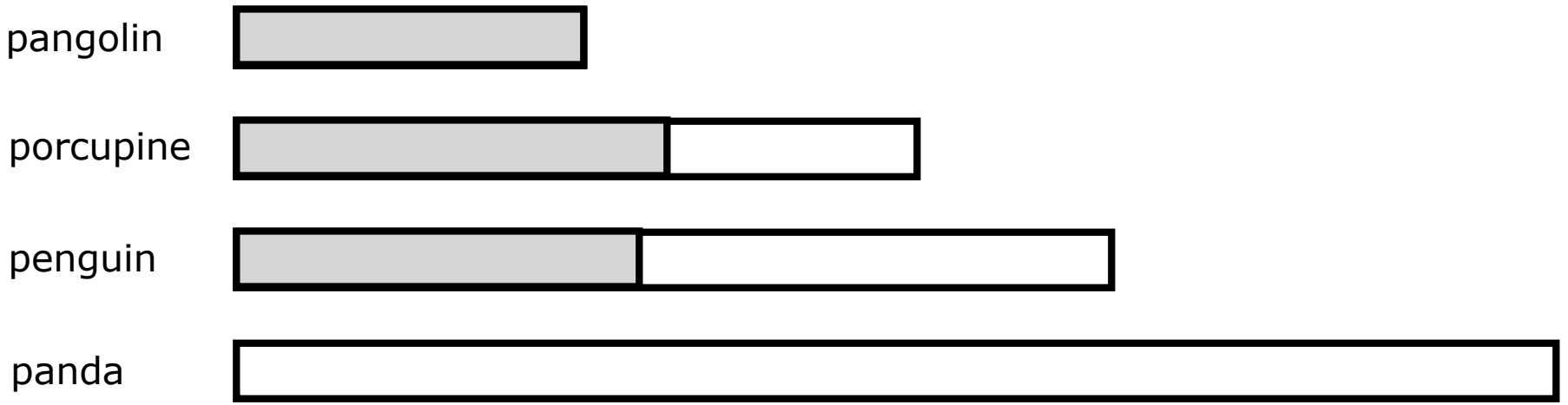
- Basic Idea #1 -- only look at prefix of lists:



- how can we put good stuff at start of list?
- maybe allow random lookups or not
- other issues: index representation and pagerank

Early Termination: Basic Setup

- Basic Idea #2 -- look at only some of the lists:

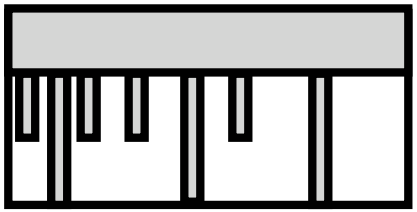


- seems to require OR query semantics
- best for traditional IR query loads: dozens or hundreds of query terms
- can be combined with previous idea

Early Termination: Basic Setup

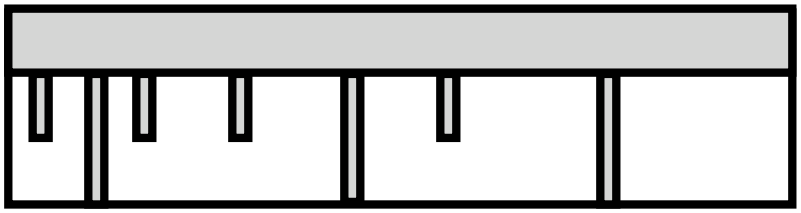
- Basic Idea #3 -- look at only some of the features ?

pangolin



docIDs plus term scores
positions
other features of docs, query, context

porcupine

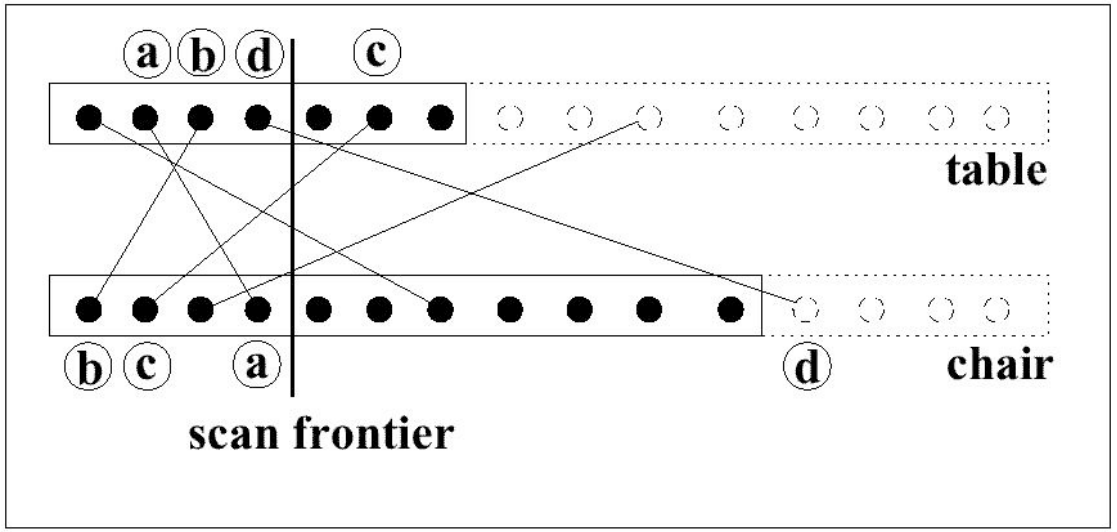


docIDs plus term scores
positions
other features of docs, query, context

- fetch advanced features only for promising docs
- after initial intersection
- literature: not much done, usually simple functions
- ... but remember the two-phase approach

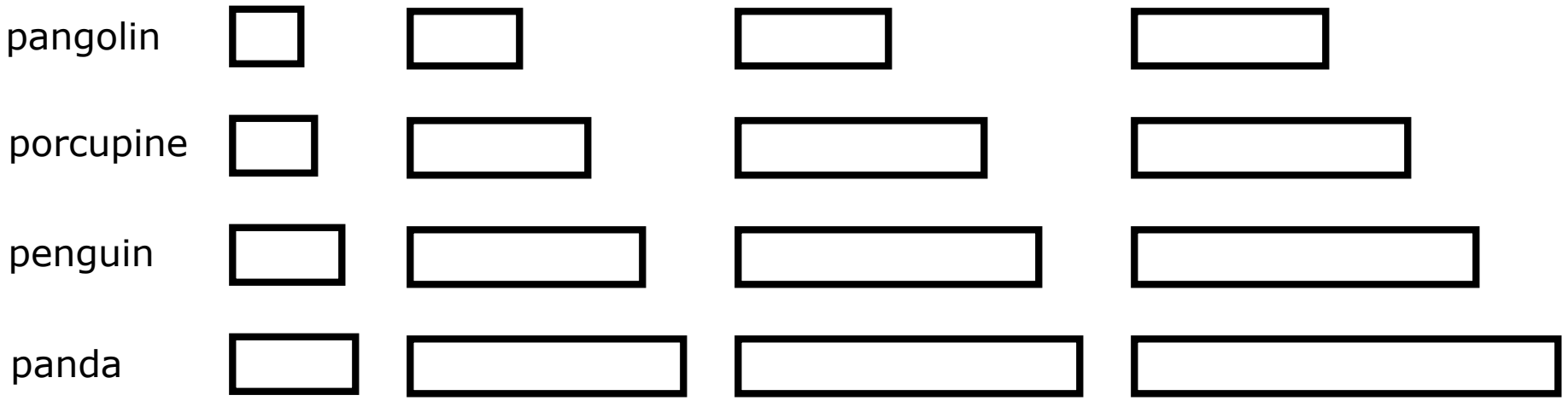
Basic Approach #1

- Fagin PODS 96/01, and subsequent work
  - motivation: searching multimedia objects by several criteria
  - typical assumptions: few attributes, OR semantics, random access
  - FA (Fagin's algorithm), TA (Threshold algorithm), NRA
  - formal bounds: $N^{m-1/m} \cdot k^{1/m}$ for k lists if lists independent
  - term-based ranking: presort each list by contribution to cosine



Basic Approach #2

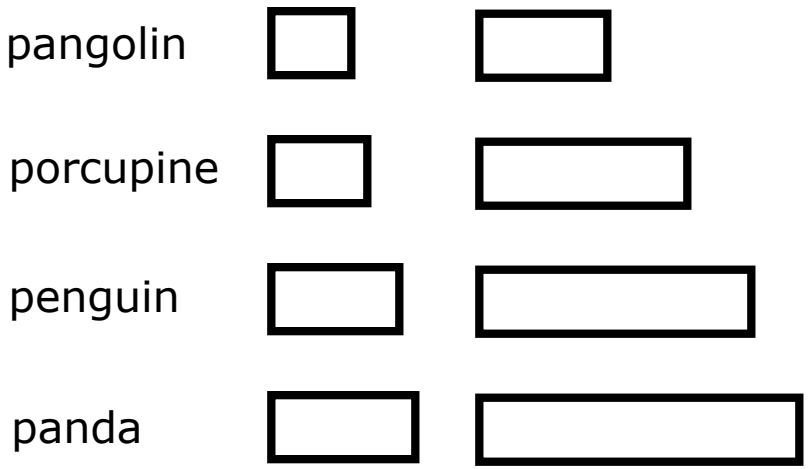
- Persin/Zobel/Sacks-Davis 93/96



- split each inverted list in m chunks
- first chunk: highest term values, best postings
- or by frequency value
- in each chunk, sort by docID
- this gives good compression and easy intersection
- read most chunks of short lists, maybe few or none of large

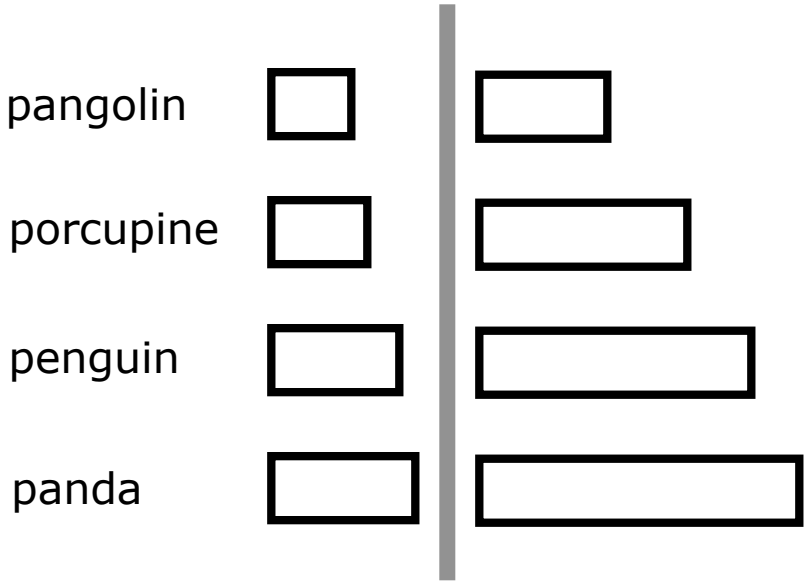
Basic Approach #2

- Brin/Page 1998: "Fancy Lists"



- split each list into only two chunks: fancy postings (URL, title, anchortext) and basic postings
- each chunk is sorted via pagerank
- if you find k results in fancy lists, just take those
- otherwise, no details given

Tiering



- split each list in two parts, put first part on one set of machines, and second part on another
- usually based on document scores [Risvik/Aasheim/Lidal 2003]
- then route most queries only to first set of machines

Query Processing Performance Summary

- a lot of interesting research
- many techniques and models
- but it is not enough to just get top-10

Research on P2P Search Architectures: What We Have, What We Should Do

What's Hot and What Not

+ interesting system designs [minerva, alvis, etc.]
+ query processing work
  [local/global/hybrid index, query routing, Bloom, top-k]
+ index updates and P2P assumptions
+ other applications: pubsub, mm search, ...

- the mining part -- general approaches needed
- really convincing numbers (big data, fast queries)

What are the Obstacles?

- communities:
  - conv. web search: IR, algorithms, systems, DB, ML
  - vs. P2P: networking, distributed computing
  - networking and IR do not mix well?
  - no "opinionated plumbing", please ...
- success of conventional engines
  - there's gold in them hills (computational advertising)
  - value of a query greatly exceeds current system costs
  - increasing barriers to entry (includes data mining)
- no real killer app?

What Should We Do?

- fast query processing
- index optimizations, tiering, early termination
- exploiting query logs for improved efficiency

- active area in conventional search engines
- potential for contributions by P2P community there
- potential for deployment in hybrid systems

What Should We Do?

- data mining platforms
- P2P mapReduce/hadoop? [Marozzo/Talia/Trunfio]
- not just individual problems/computations
  (pagerank, spam detection, near duplicate detection)

- or better to do this in a centralized cluster?

What Should We Do?

- hybrid architectures
- piggyback on existing engine [Felber et al, today]
- or use some centralized components
- e.g., crawler or data mining
- or central + p2p query processing

- and, use and share BIG data and software
- more open environment for search (beyond P2P)

Thank You!

… and remember, Yahoo! is hiring …