

# Applications of Indexing

## Lecture 14



# Outline

- Background & Motivation
  - Full-Text Search
  - Big-O Review
  - Indexing
- The Inverted Index
  - An Example
  - Design a relational index
  - Advanced Issues
  - Example in Cognitive Modeling
- The R-tree
  - Overview
  - Application in Optimization



# Problem: Full-Text Search

- Given: set of “documents” containing “words”
  - General problem in the field of *Information Retrieval*
- Task: find “best” document(s) that contain a set of words
- Requirements
  - Fast & scalable
  - Relevant results (precision, recall, f-score)
  - Expressive queries
  - Up-to-date



# Example: Web Search

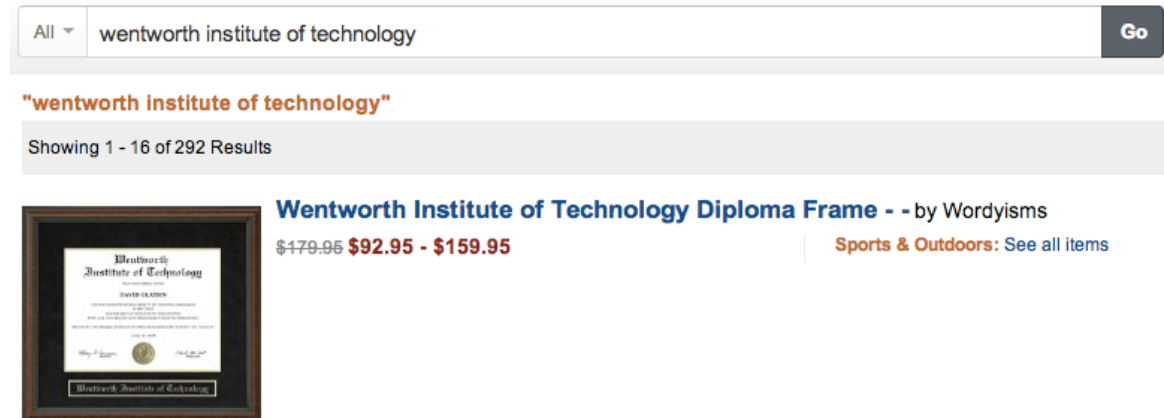
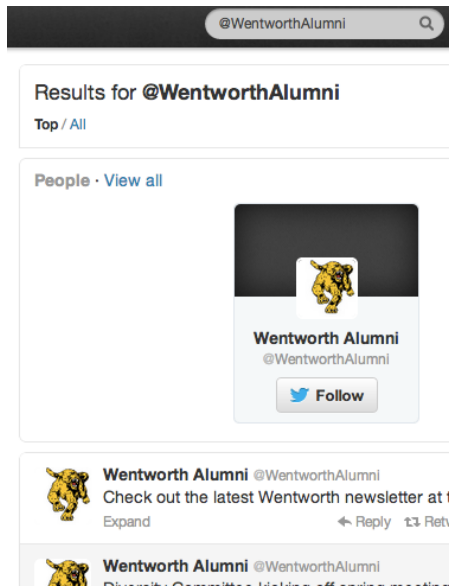
The screenshot shows a Google search for "wentworth institute of technology". The search bar contains the text "wentworth institute of technology" and a search button. Below the search bar are tabs for "Web", "Images", "Maps", "Shopping", "More", and "Search tools". A blue callout box points to the search results with the text "document = web page/map listing".

The search results include:


- Search Summary:** About 670,000 results (0.30 seconds)
- Advertisement:** Ad related to wentworth institute of technology. [Continuing Education - WIT.edu](http://www.wit.edu/continuinged). [www.wit.edu/continuinged](http://www.wit.edu/continuinged). Wentworth Institute of Technology. Become a leader in your industry!
- Search Result:** [Wentworth Institute of Technology: Boston, MA : Wentworth Instit...](http://www.wit.edu/). [www.wit.edu/](http://www.wit.edu/). Offers bachelor's degrees in architecture, design, engineering, **technology**, and management of **technology**. Nearly 3000 students attend this coeducational ... 4.6 ★★★★★ 13 Google reviews · Write a review
- Location:** 550 Huntington Ave, Boston, MA 02115 (617) 989-4590
- Links:**
  - [Admissions](#): Undergraduate - Visit - Apply Now - Financial Aid - Transfer - ...
  - [Academics](#): At Wentworth you will test yourself in extraordinary and ...
  - [Architecture](#): Faculty & Staff - Architecture - Contact Us - ...
  - [WIT Email Landing Page](#): Learn more about your Wentworth email account, including how to ...
  - [About Wentworth](#): Colleges and Departments - Campus Map - Accreditation
  - [Continuing Education](#): Contact Us - Programs - Certificate Programs - Schedules - ...
- Map Listing:** A map showing the location of the Wentworth Institute of Technology in Boston, MA. The map includes a street view and a list of nearby locations: Museum of Fine Arts, Longwood Medical Area, and others. A blue arrow points from the map listing to the search results.



# Other Examples

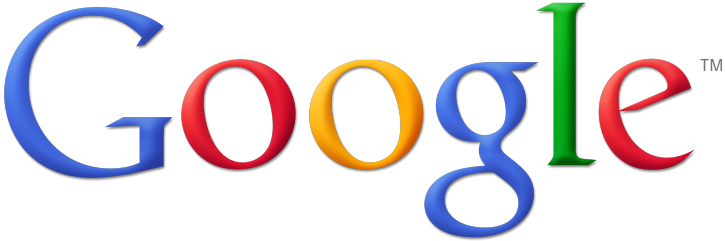



# Scaling Up: # of “Documents”



# Scaling Up: Search Frequency



# Big-O Review

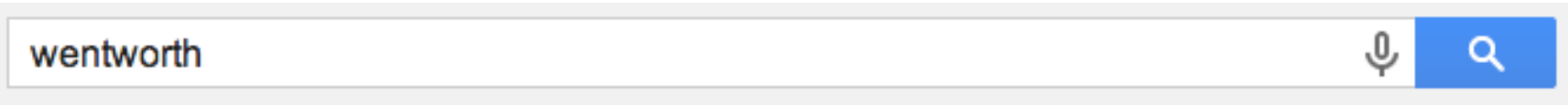
- What does  $O(n)$  mean?
- A linear algorithm for full-text search?
  - Find 5 documents that contain “WIT”
- What is the complexity in terms of documents ( $d$ ) and average-words-per-document ( $w$ )?





# Is Linear-Time Google Possible?

- Assume simple query:



- Single listing of all web pages + words

– 60T pages \* 250 w/page \* 8 bytes/w ~ 106PB

- Require 1s response

– 7.5M \* 2GHz 64-bit CPU (assume 1 cycle/w)

– (7.5M \* 68K) CPUs \* 85W/CPU \* \$.15/kWh ~ **\$1.8M/s**

~4M Blu-ray

~70 CPU/  
person

3X US GDP



# Indexing

- Improve search speed at the cost of extra...
  - **Memory** for data structure(s)
  - **Time** to update the data structure(s)
- Backbone of databases (physical design)
  - Search engines
  - Graphics/game engines
  - Simulation software
  - ...



# Inverted Index by Example

Given documents  $\{ D_1, D_2, D_3 \}$ :

- $D_1$  = “it is what it is”
- $D_2$  = “what is it”
- $D_3$  = “it is a banana”

Inverted Index:

Distinct Word List  
(sorted)

- “a”: [  $D_3$  ]
- “banana”: [  $D_3$  ]
- “is”: [  $D_1, D_2, D_3$  ]
- “it”: [  $D_1, D_2, D_3$  ]
- “what”: [  $D_1, D_2$  ]

Document Lists

Let’s try some queries:  
“what”, “a”, “banana”, “apple”

Describe an algorithm to query  
this data structure

Describe an algorithm to  
populate these lists

Time & Memory:  $O(?)$   
*Construction & Query*



# Design a **Relational** Inverted Index

Develop a set of table(s) and index(es) that support efficient construction and querying of an inverted index

## Assume

- Documents have a unique id and a path
- A document is a sequence of words
  - Document  $d = [w_1, w_2, \dots, w_n]$
- Search for a single, exact-match word
  - Does document  $D$  have word  $w$ ?
  - The list of documents  $\mathbf{D}$  that have word  $w$ ?



# Advanced Issues

- More expressive query semantics
  - Multi-word
  - Locality: [“what is it”] vs. [ “what it is” ] vs. [“what”, “is”, “it”]
- Ranked results
  - Document-ranking algorithm (e.g. PageRank)
  - Efficient ranked retrieval
- Dynamics
  - Document addition/removal/modification
  - Rank
    - Document changes
    - Integration of real-time variables (e.g. location)



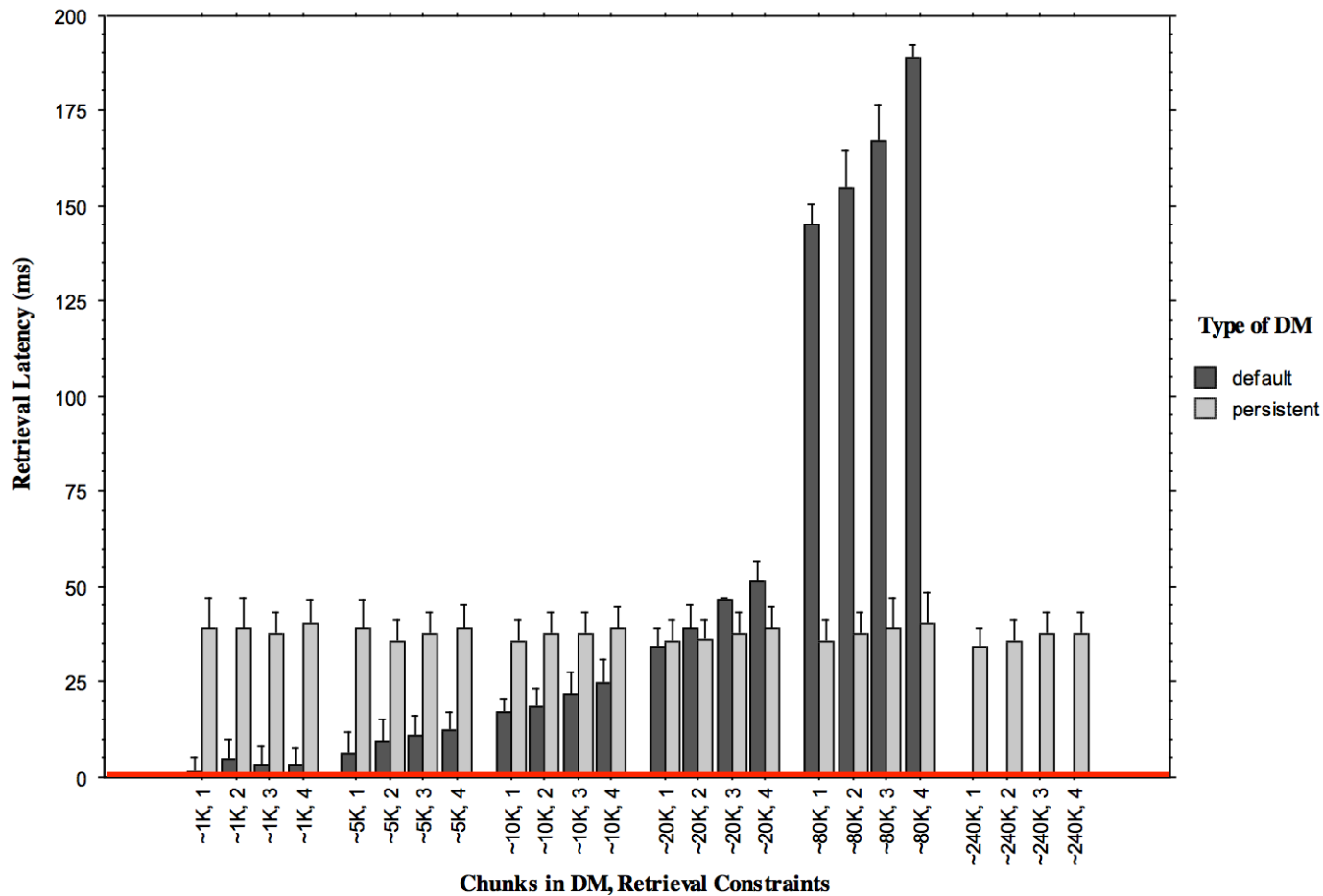
# Modeling Semantic Memory

- Semantic memory is a human's long-term store of facts about the world, independent of the context in which they were originally learned
- The ACT-R (<http://act-r.psy.cmu.edu>) model of semantic memory has been successful at explaining a variety of psychological phenomena (e.g. retrieval bias, forgetting)
- The model does *not* scale to large memory sizes, which hampers complex experiments



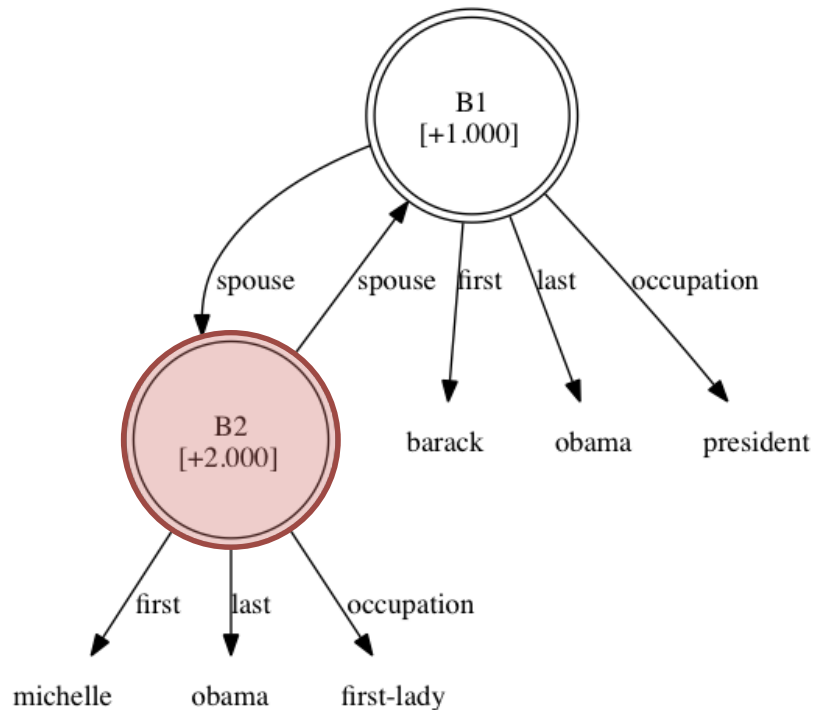
# Scale Fail [AFRL '09]

Retrieval Latency: Chunks in DM x Retrieval Constraints x Type of DM  
(Error Bars: 95% Confidence Interval)



# Memory Representation

- Document = Node
- Word = edge



Example cue:

**last (obama) , spouse (X)**

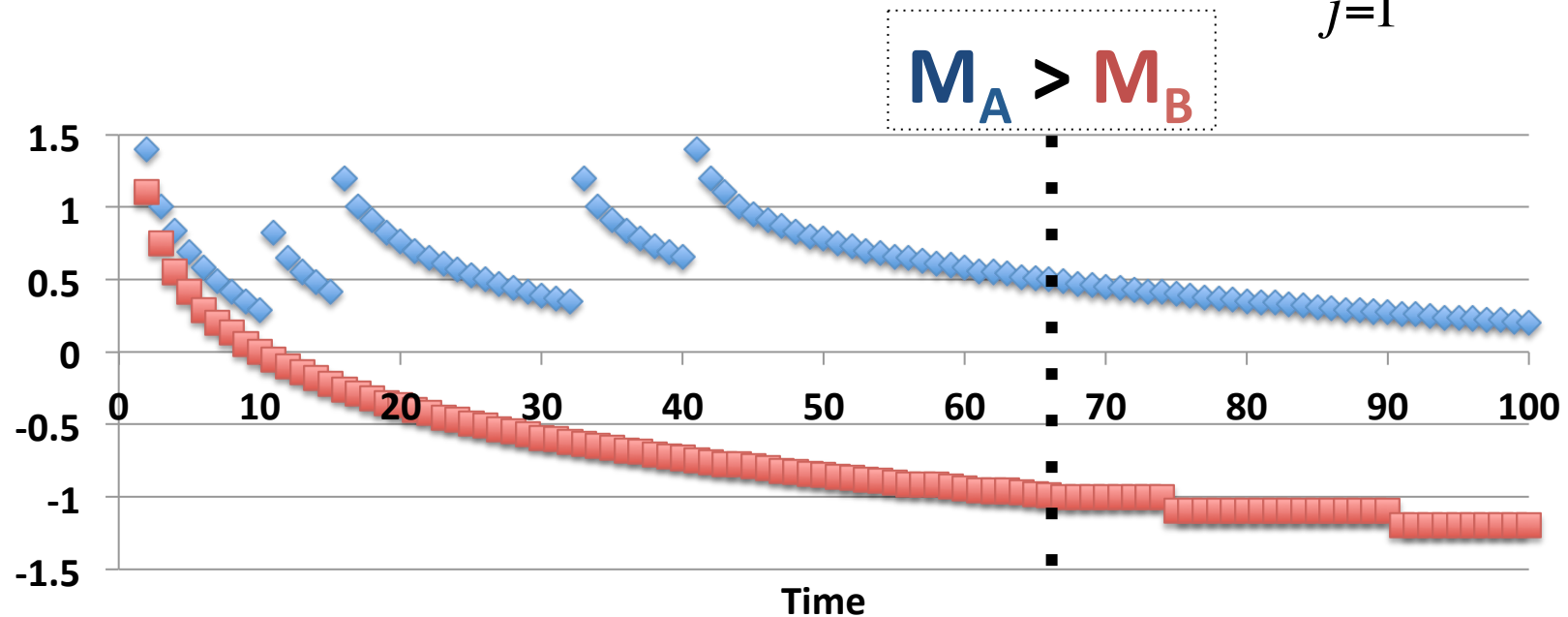




# Ranking [Anderson et al. '04]

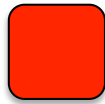
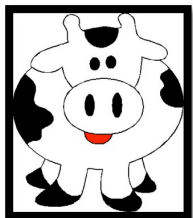
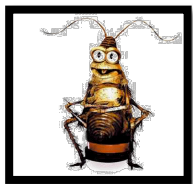
Predict future usage via history

$$\ln\left(\sum_{j=1}^n t_j^{-d}\right)$$



# Example

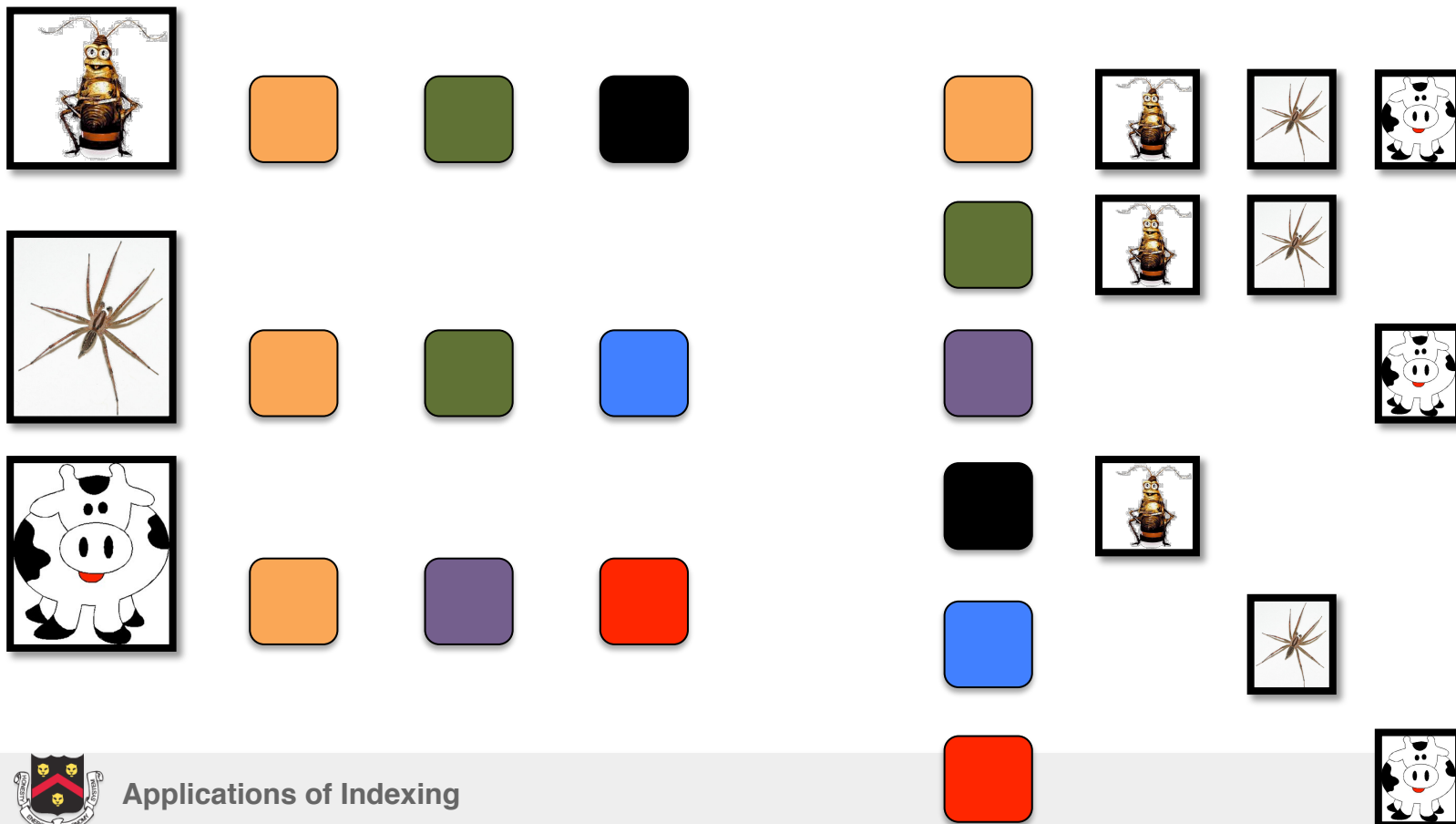
## Semantic Objects: Features



# Inverted Index

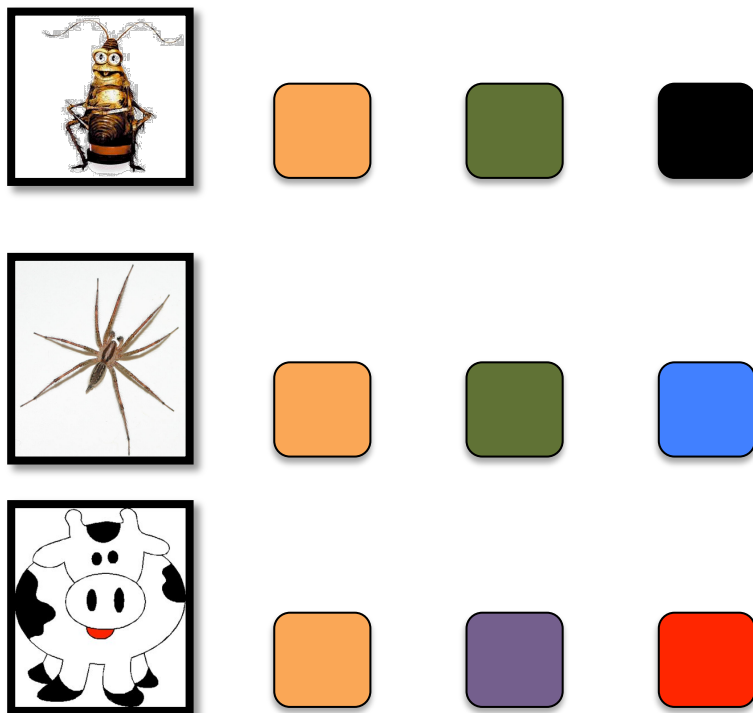
## Semantic Objects: Features

## Inverted Index

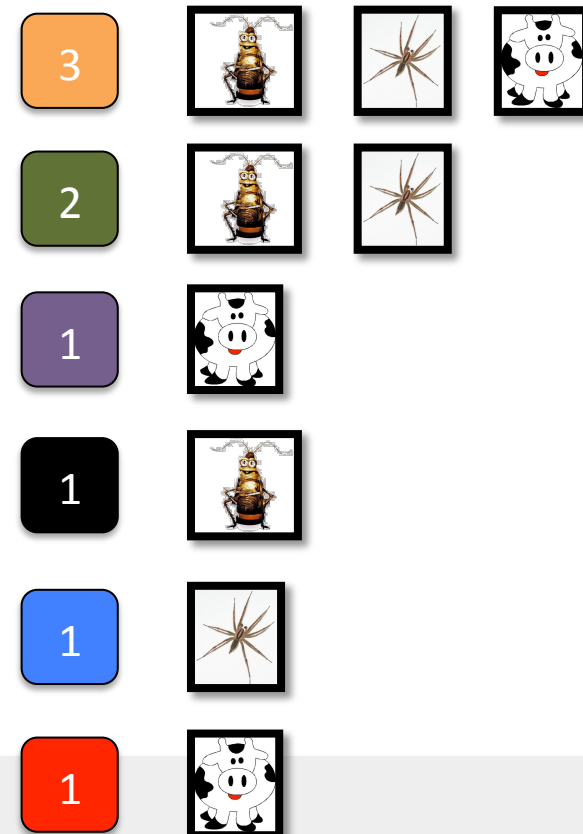


# Index Statistics

## Semantic Objects: Features

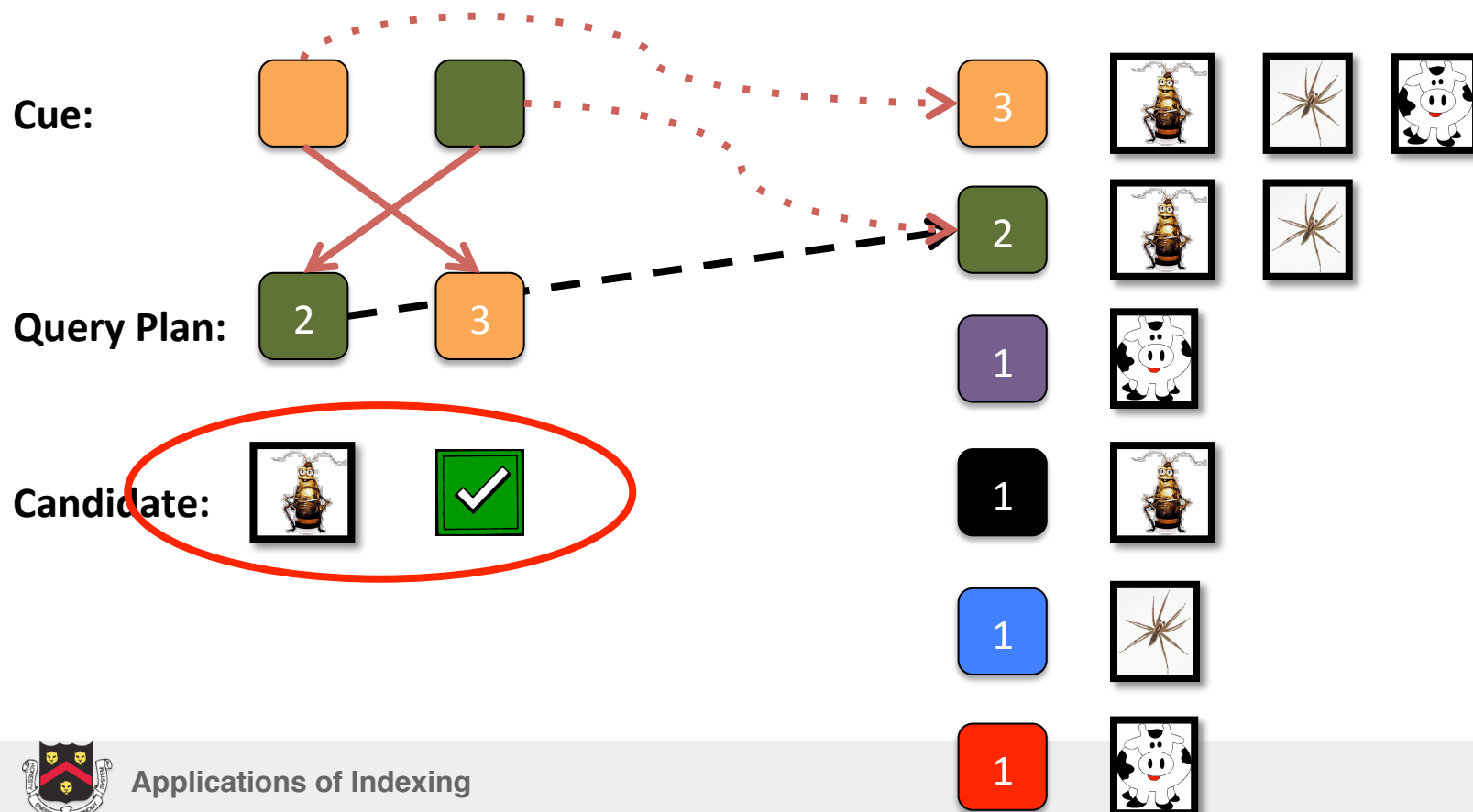


## Inverted Index



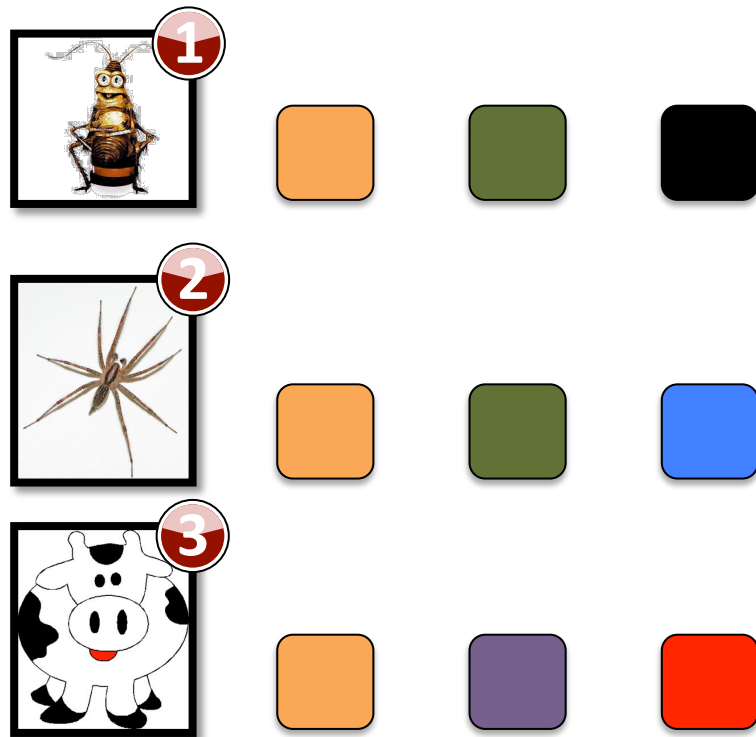
# Top-1 Non-Ranked Retrieval

## Inverted Index

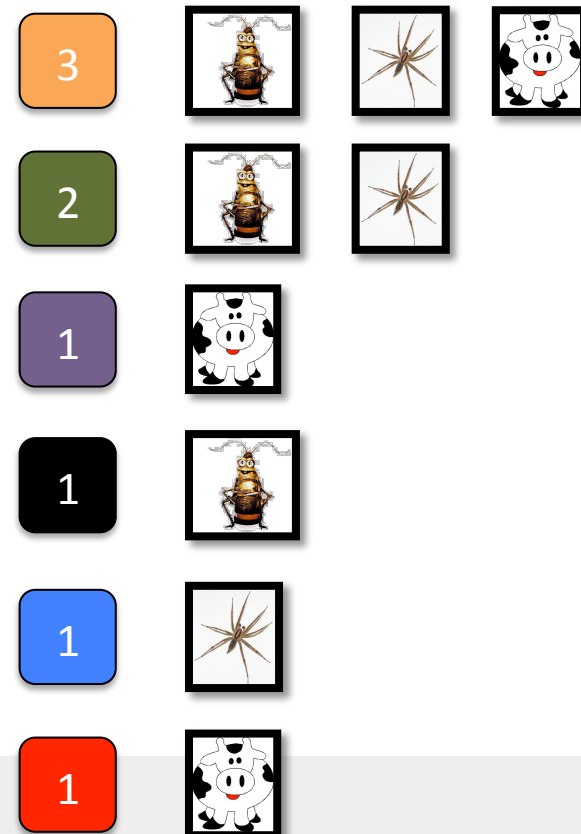


# Introducing Rank

## Semantic Objects: Features



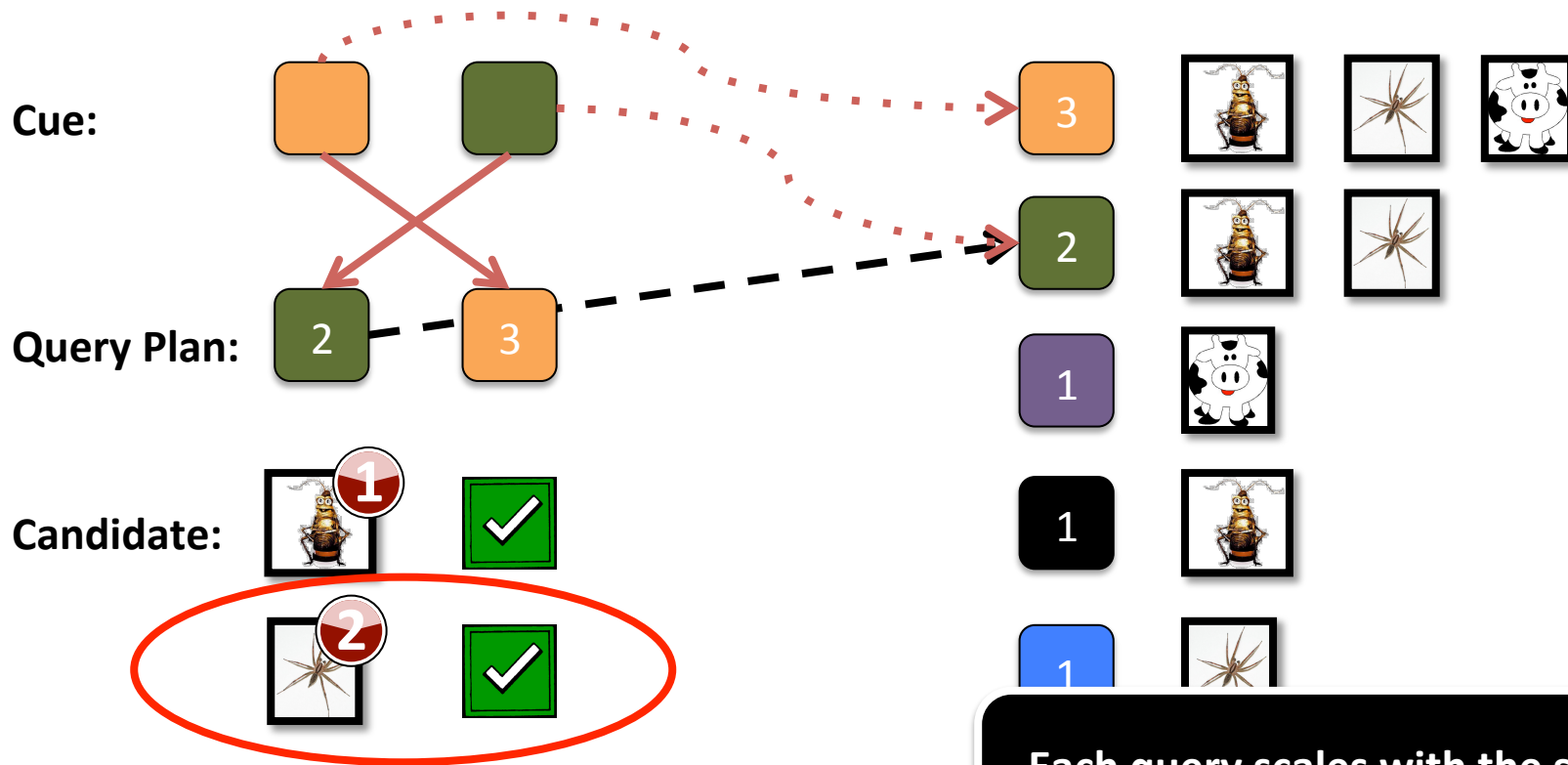
## Inverted Index



# Ranked Retrieval Algorithm #1

## Sort on Query

### Inverted Index



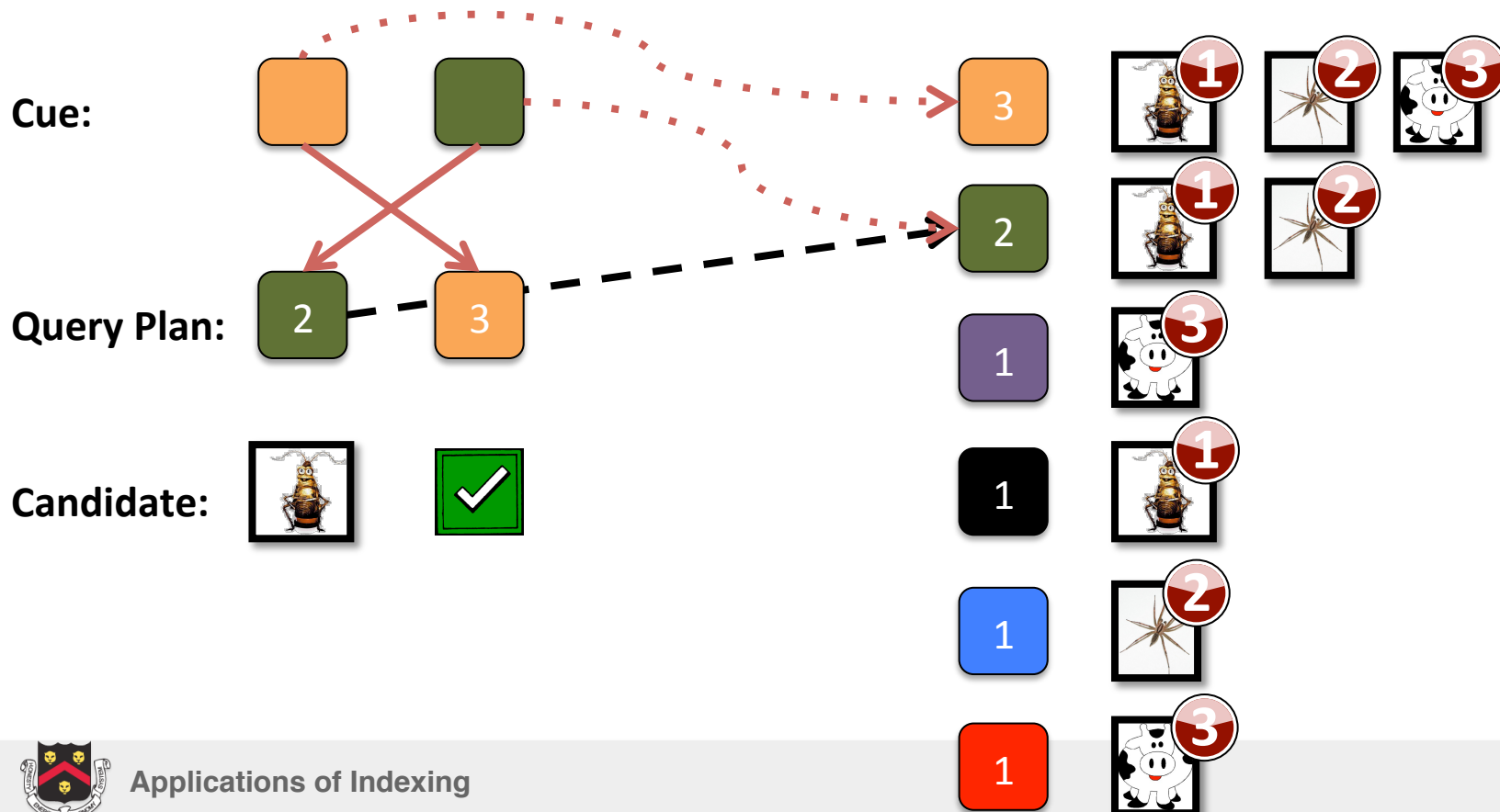
Each query scales with the size of the candidate list!



# Ranked Retrieval Algorithm #3

## Static Sort

### Inverted Index

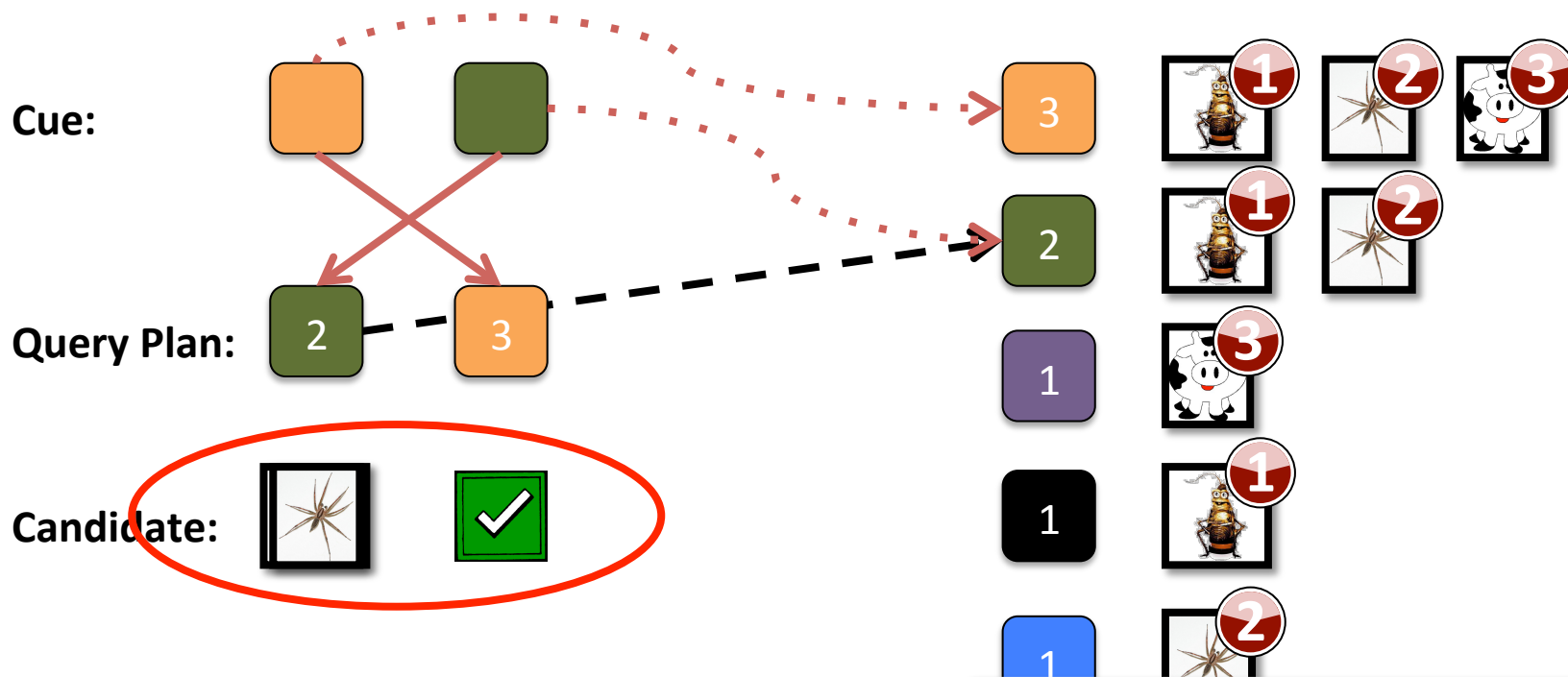




# Ranked Retrieval Algorithm #2

## Static Sort

### Inverted Index



Each rank update scales with feature cardinality!



# Hybrid Approach

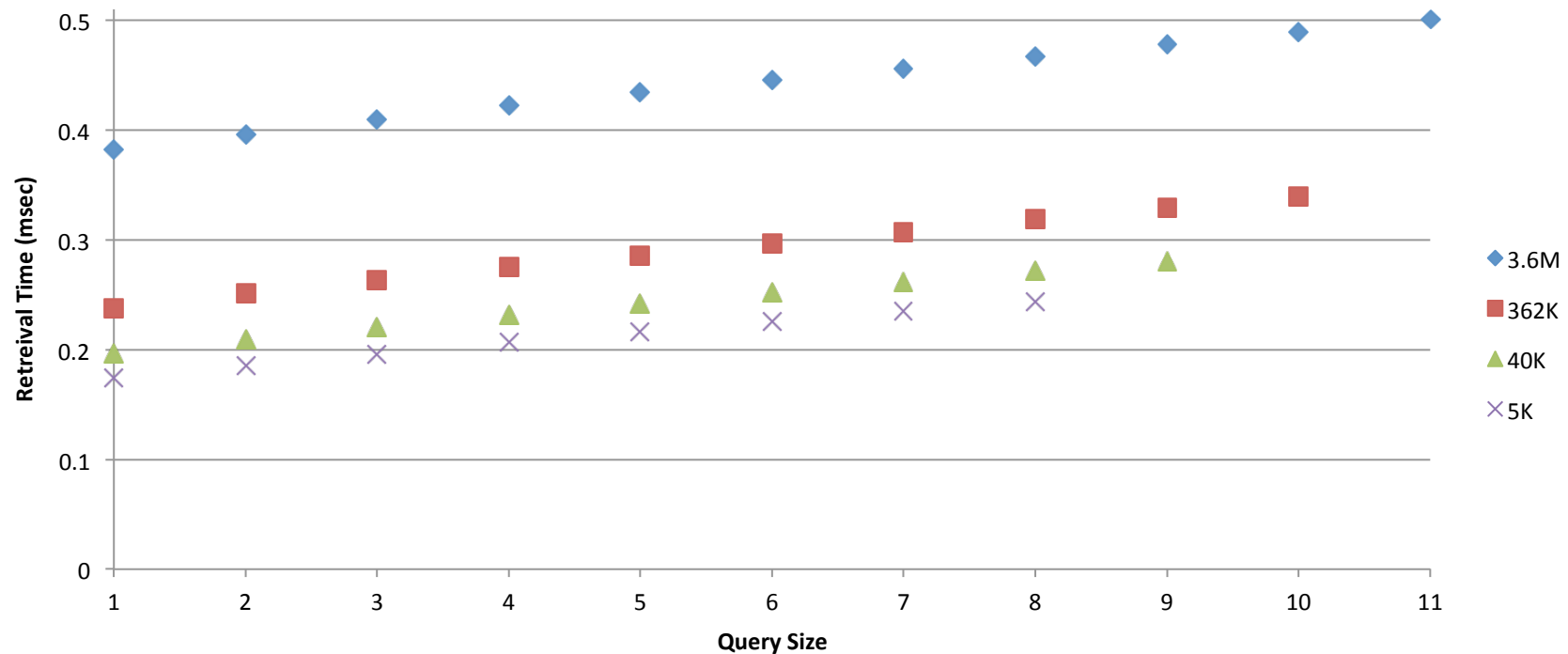
- Empirically supported cardinality threshold,  $\theta$
- If (cardinality  $> \theta$ ): Sort on Query [#1]
  - Candidate enumeration scales with # of objects with large cardinality (empirically rare)
- If (cardinality  $\leq \theta$ ): Static Sort [#2]
  - Bias updates must be locally efficient
    - Objects affected:  $O(1)$
    - Computation:  $O(1)$



# Some Results

Inverted index (via SQLite) + new approach was **fast** and **scaled!**

>30x faster than off-the-shelf database (on >3x data)!

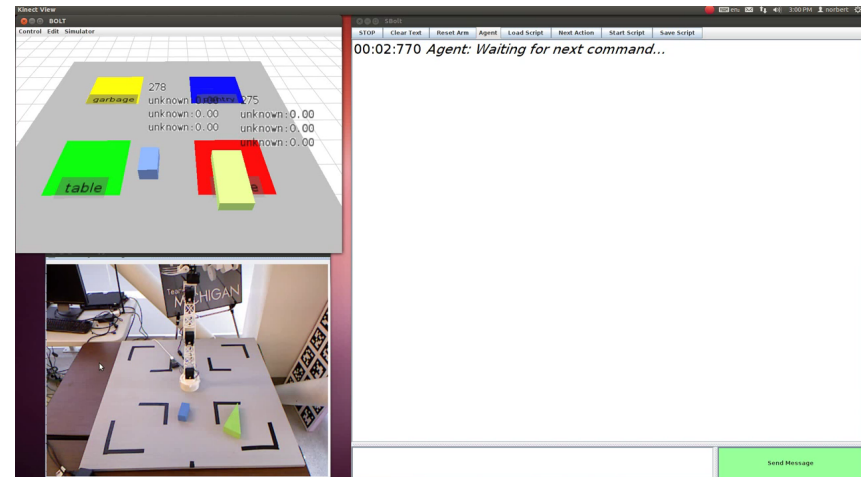


# Applications: AI + Inverted Index

## Learned Navigation

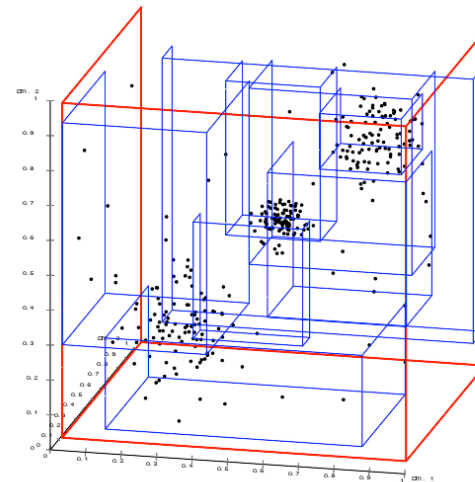
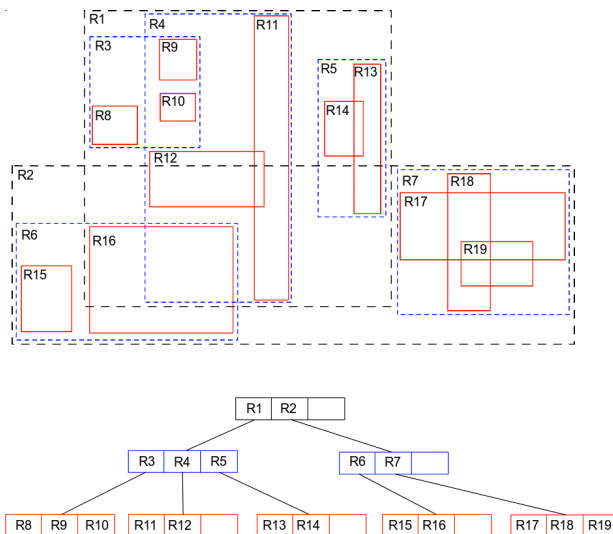


## Task Learning



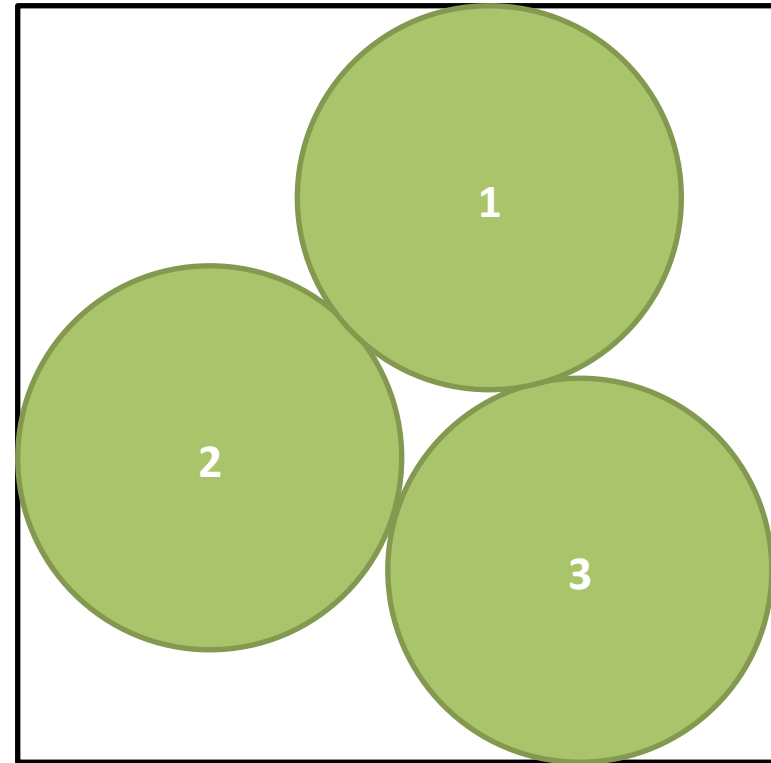
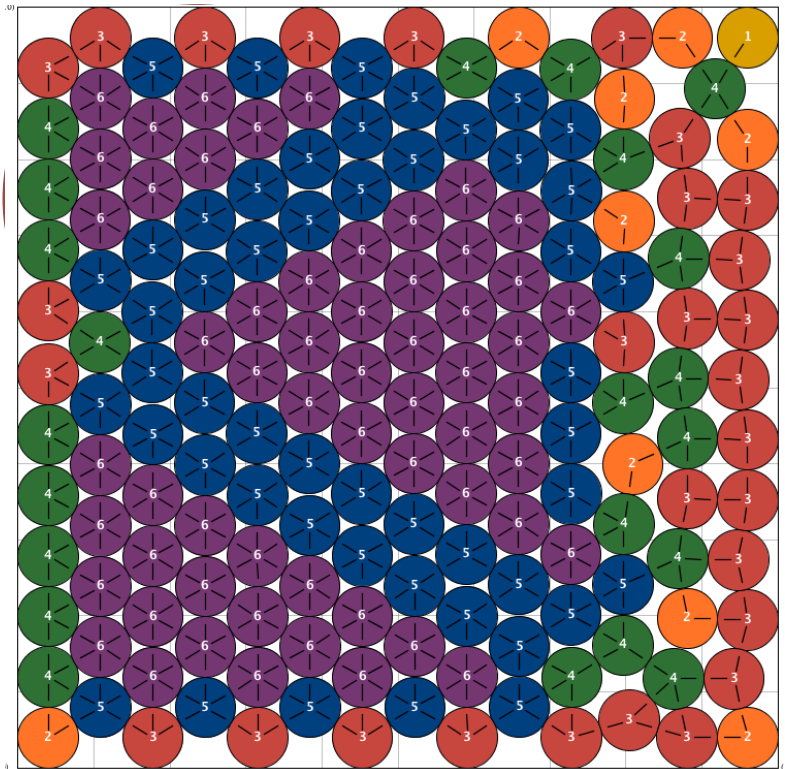
# Another Index: R-tree

“Group nearby objects and represent them with their **minimum bounding rectangle** in the next higher level of the tree... Since all objects lie within this bounding rectangle, a query that does not intersect the bounding rectangle also cannot intersect any of the contained objects. At the leaf level, each rectangle describes a single object; at higher levels the aggregation of an increasing number of objects.” -- Wikipedia

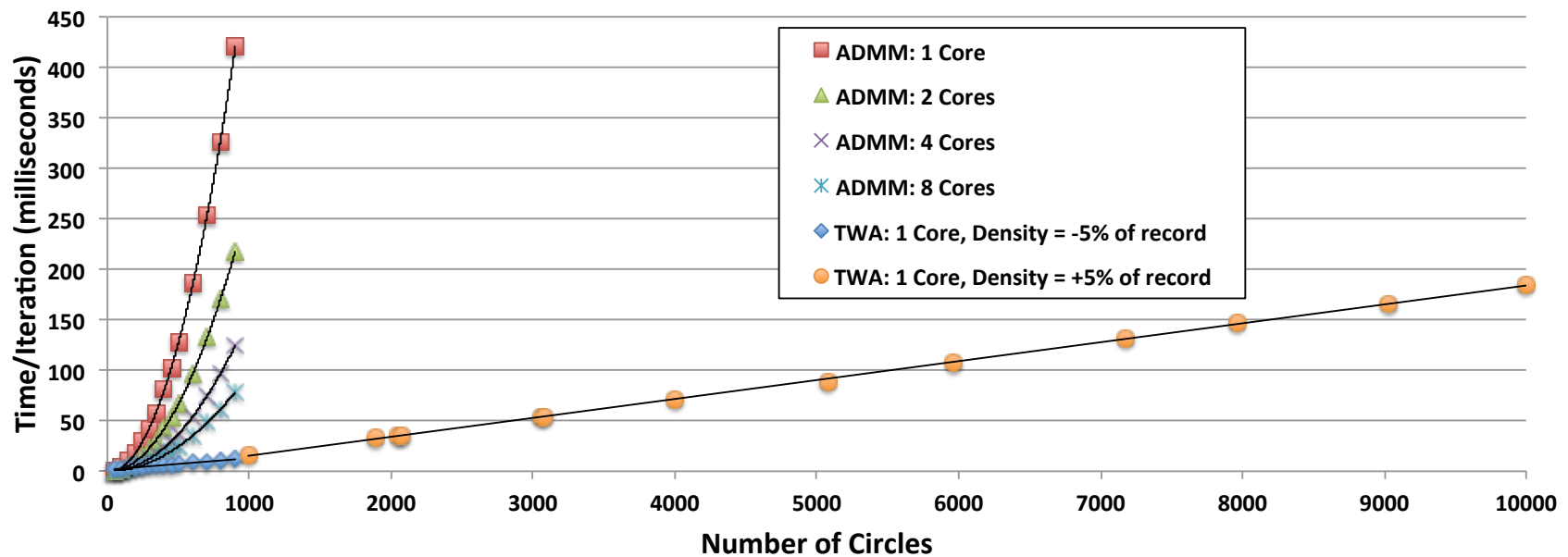


# Application: Optimization + R-tree

**Packing.** Fit  $n$  circles of radius  $r$  in a square of side-length  $s$  without overlap (non-convex, NP-hard,  $\infty$  solutions). Used in making codes, physical packing, computer-assisted origami.



# Large-Scale Evaluation



# Takeaways

- A common approach to large-scale search is indexing: using data structure(s) to improve access speed
- An inverted index is commonly used for full-text search (even in situations that might not look like it)
  - Inverted indexes are fast, scalable, and straight-forward to implement
- An R-tree is commonly used for spatial queries over objects in 2/3D space (e.g. what is within X miles of Y? are A and B colliding?)
- Know your indexes/data structures! Careful problem analysis and algorithm development can often beat generic approaches
  - Even if you don't use a DBMS, DBMS methods can be very useful in a variety of applications!

