Data Structures with Randomness:
Skip Lists

Flashback to Data Structures…

Recall the `List` interface

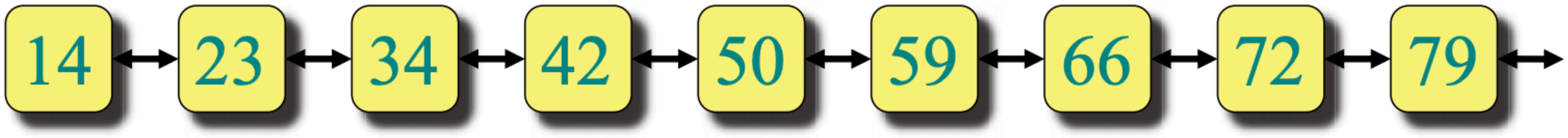
- What are the `List` operations?
- What concrete `List` implementations did we study?

- What are the tradeoffs between arrays and linked lists?
- Do those tradeoffs change when our lists are sorted?
- How does this compare to a binary search tree?

Let’s develop a data structure with the strengths of a Binary Search Tree but the (relative) simplicity of a `List`

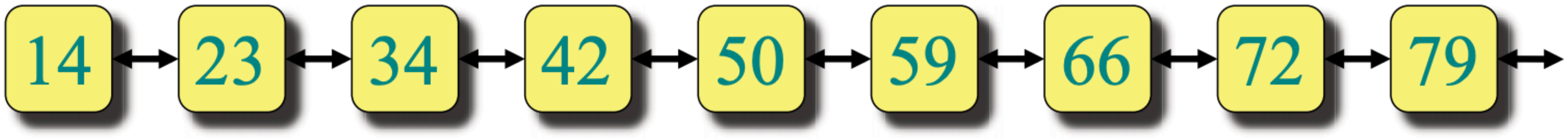
One Linked List

- Start from simplest data structure: (sorted) linked list
- Search cost?
    - $\Theta(n)$
- How can we improve it?



Two Linked Lists

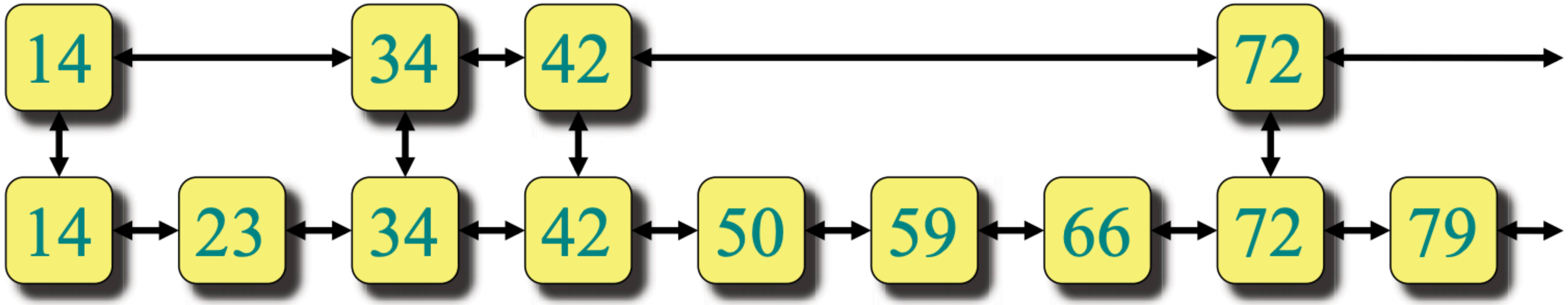
- Suppose you instead had *two* sorted linked lists
  - Each list can contain a subset of the total elements
  - Elements can appear in one or both lists
- **Class exercise.** How can you use two lists to speed up searches?



NYC Subway System

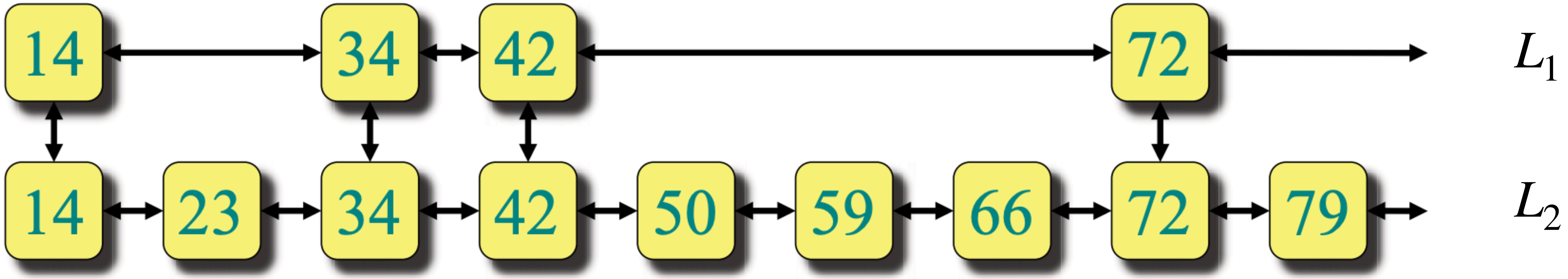
Two Linked Lists

- **Idea**: we have both express and local subways
- Express lines connect a few main stations (and skip a bunch)
- Local lines connect all stations but are slow
- All express stops are also local stops so you can switch



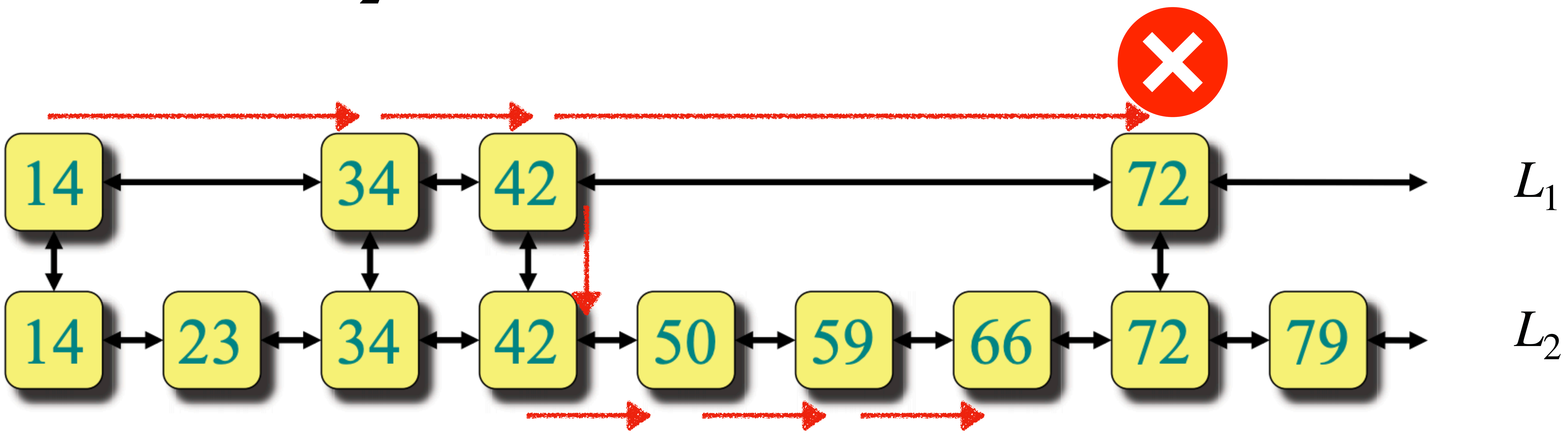
Two Linked Lists

- **Search**(x):
  - Walk right in top linked list L_1 until going right would be too far
  - Walk down to bottom linked list L_2
  - Walk right in L_2 until x is found or reach end (report not found)



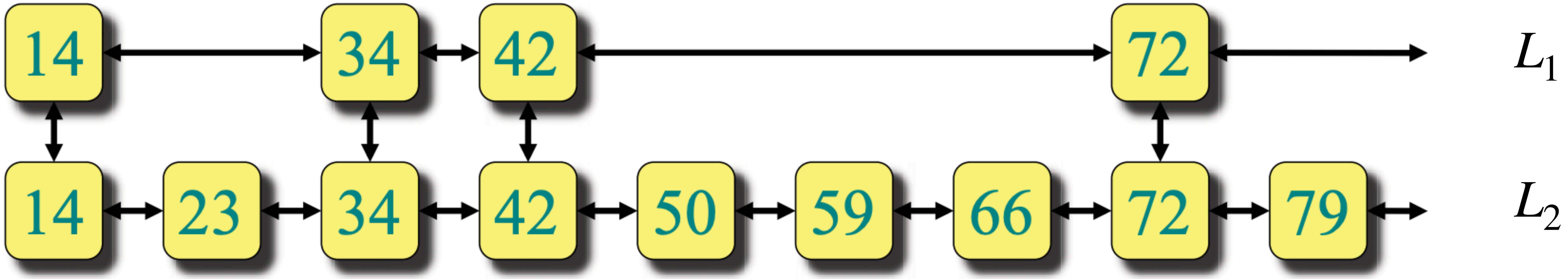
Two Linked Lists

- **Search**(66):
  - Walk right in top linked list L_1 until going right would be too far
  - Walk down to bottom linked list L_2
  - Walk right in L_2 until x is found or reach end (report not found)



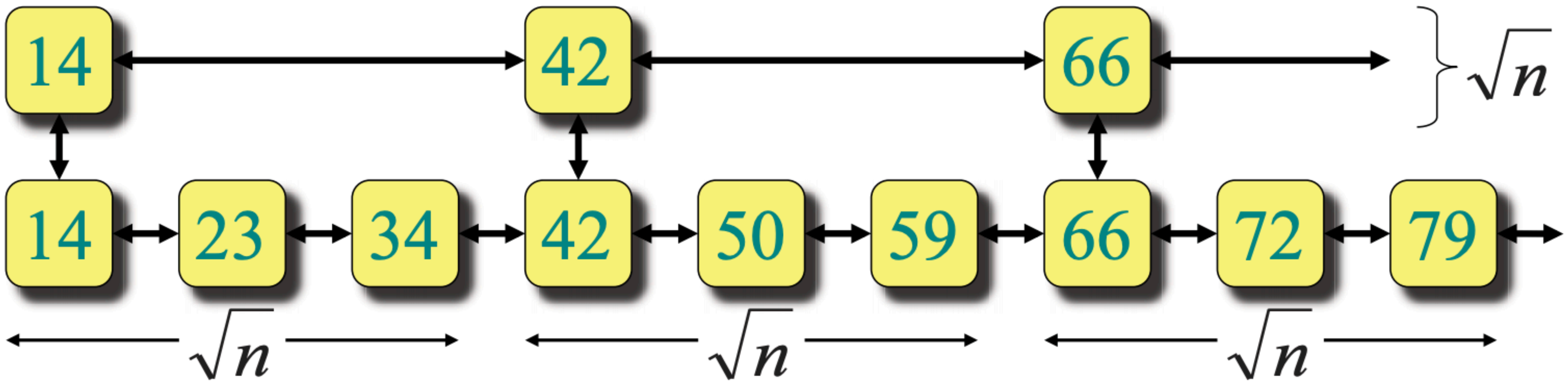
Two Linked Lists

- How should we organize the two lists?
  - Which nodes go in L_1?
  - How much of gap to leave between L_1 elements?
  - **Best approach**: evenly space and promote elements



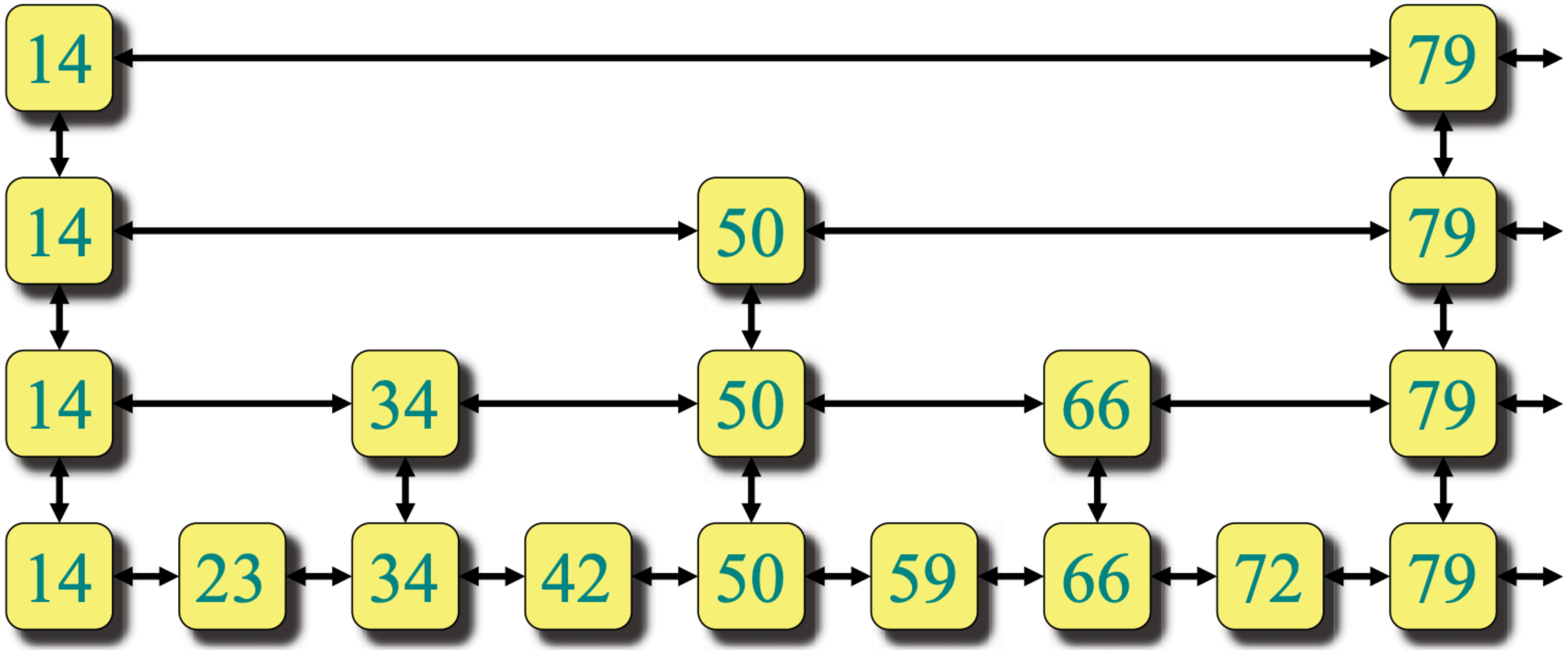
Two Linked Lists

- If gap between elements in top list is g, then the number of elements traversed (search cost) is at most $g + n/g$
- Optimized by setting $g = \sqrt{n}$
- So the search cost is at most $2\sqrt{n}$



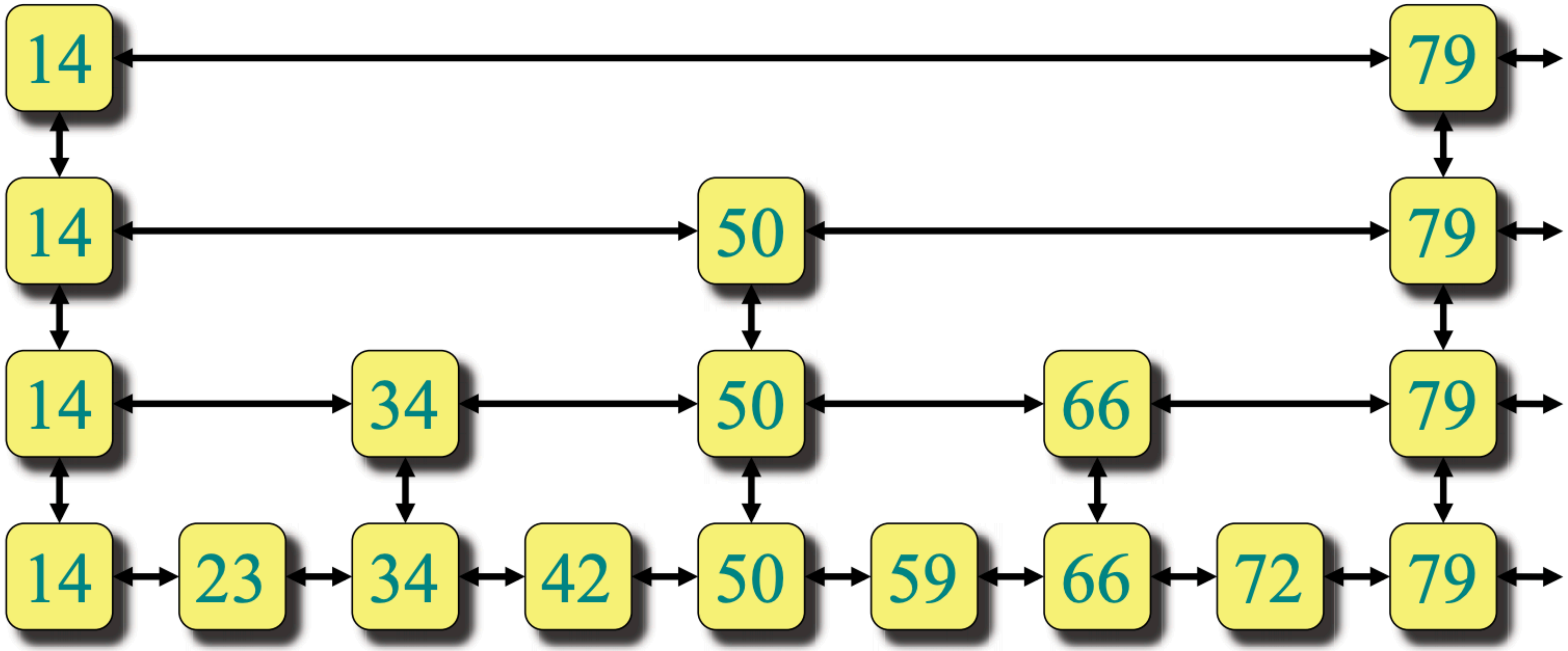
More Linked Lists

- Search cost with two linked list: $2\sqrt{n}$
- Search cost with three linked list: $3n^{1/3}$



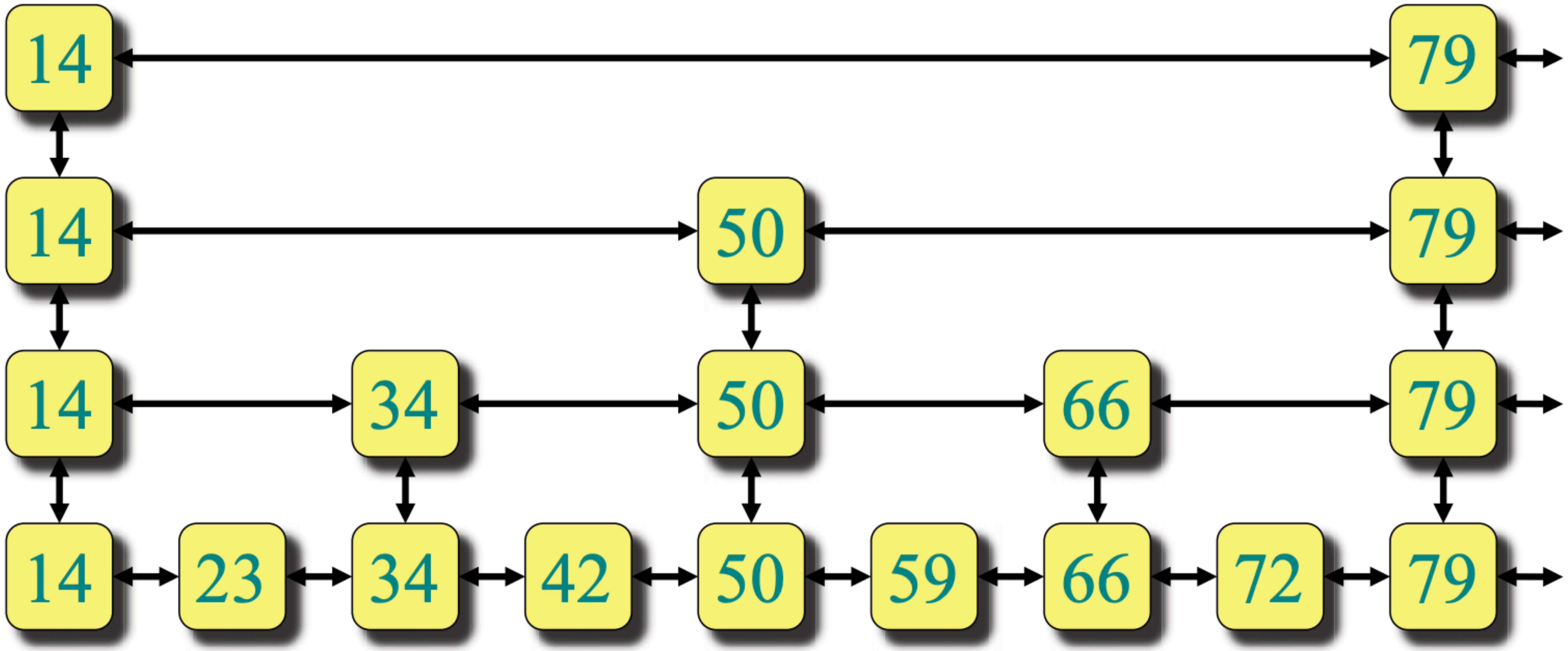
k Linked Lists

- Search cost with k linked lists: $kn^{1/k}$
- Search cost with $\log n$ linked lists: $\log n \cdot n^{1/\log n}$
  - $\log n \cdot n^{1/\log n} = 2 \log n$



Insertion Cost

- This is good, but how can we insert?
- Every new element disrupts our spacing
- Idea: use randomness!



Acknowledgments

- Some of the material in these slides are taken from
  - Shikha Singh
  - MIT slides: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-12-skip-lists/lec12.pdf
  - Eric Demaine handout: https://courses.csail.mit.edu/6.046/spring04/handouts/skiplists.pdf