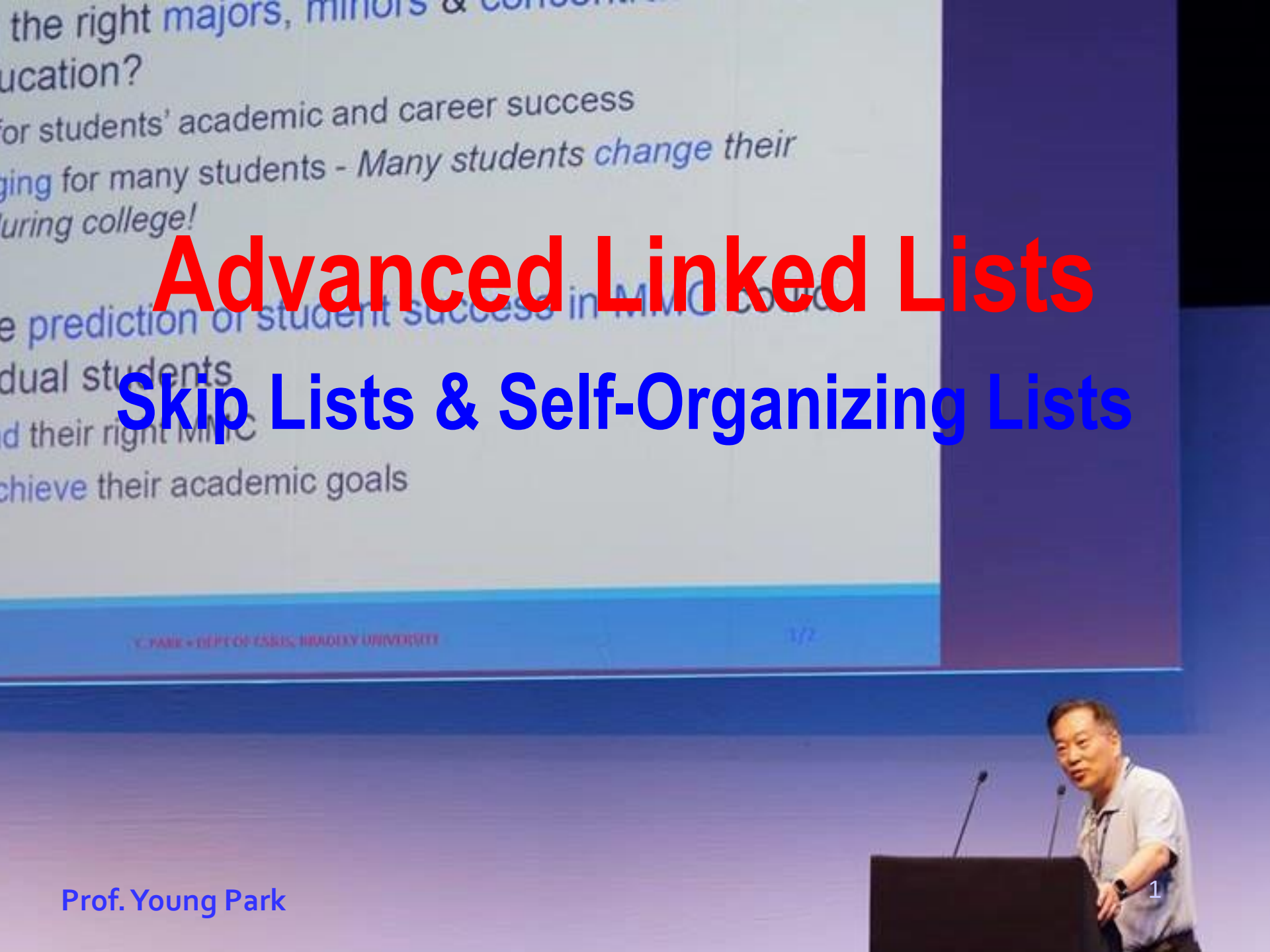
Advanced Linked Lists

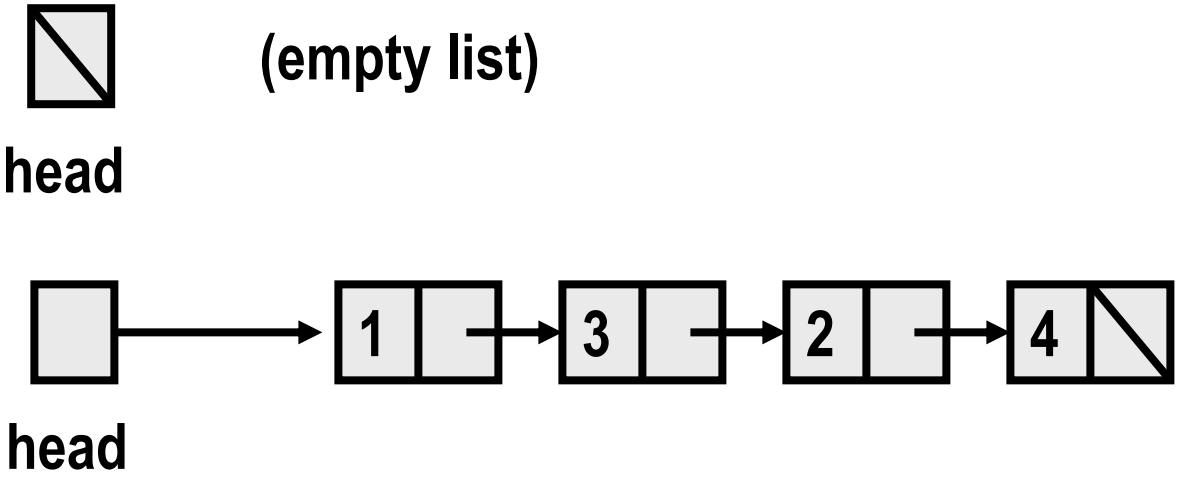
Skip Lists & Self-Organizing Lists



Prof. Young Park

Singly Linked Lists

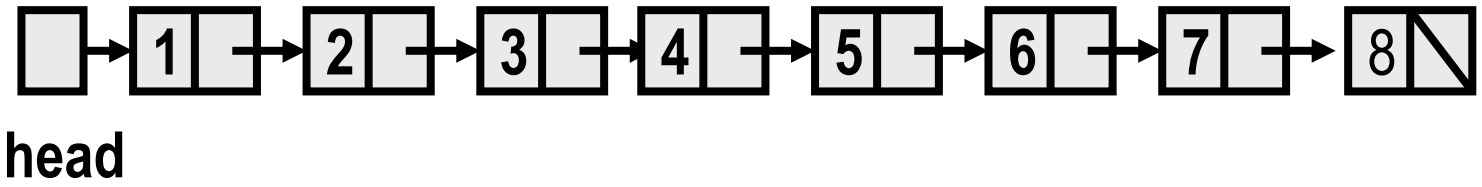
Singly Linked List



- **Search?**
  - ➔ **O(n) time** (Sequential search) at worst case & average case
- Can we do better?

Sorted (Ordered) Singly Linked List

- How about **sorted** singly linked list?
  - All nodes are sorted.



- **Search?**
  - Sequential search? **O(n) time** at worst case & average case
  - **Binary search? O(n) time!** at worst case & average case
    - **WHY?**

►QUIZ?

- Can Binary Search be used for sorted linked lists?
  - ➔Yes
- What is the worst-case time complexity of the binary search algorithm on a sorted linked list?
  - ➔O(n)

Motivations?

- Can we **search** in a (sorted) linked list in better than O(n) time at **worst-case**?
- Can we **search** in a (sorted) linked list in better than O(n) time at **average-case**?

- Can we **search** in a (sorted) linked list in better than O(n) time at **amortized-case**?

Advanced Linked Lists

Why Advanced Linked Lists?

- **Why?**
  - ➔ The **search** operation on **linked lists** – whether **unsorted** or **sorted**:
    - **O(n)** time at worst-case & average-case
  - ➔ A better **search** operation?
  - ➔ **To improve the time complexity of the search operation - worst-case, average-case & amortized-case!**

What Advanced Linked Lists?

- **What?**
  - For **O(log n) worst-case Search** & **O(log n) average-case Search, Insert/Delete**?
  - **Skip lists**
    - **Perfect (Ideal)** skip lists
    - **Good (Practical)** skip lists
  - For **Better than O(n) amortized Search**?
  - **Self-organizing lists – Self-Adjusting Data Structure!**

Skip (Linked) Lists

Skip Lists

- Problem:
  - Linked lists – The search operation requires O(n) time!
- Goal:
  - **To improve the worst-case & average-case time complexity of the search operation!**
- Idea:
  - Use **a sorted list with additional (skipping) links**!
  - Called a **skip** list!

Skip Lists

- **Perfect (Ideal)** skip lists
- **Good (Practical)** skip lists - **Randomized/Probabilistic Data Structure!**

Good average-case time complexity!

Perfect Skip List

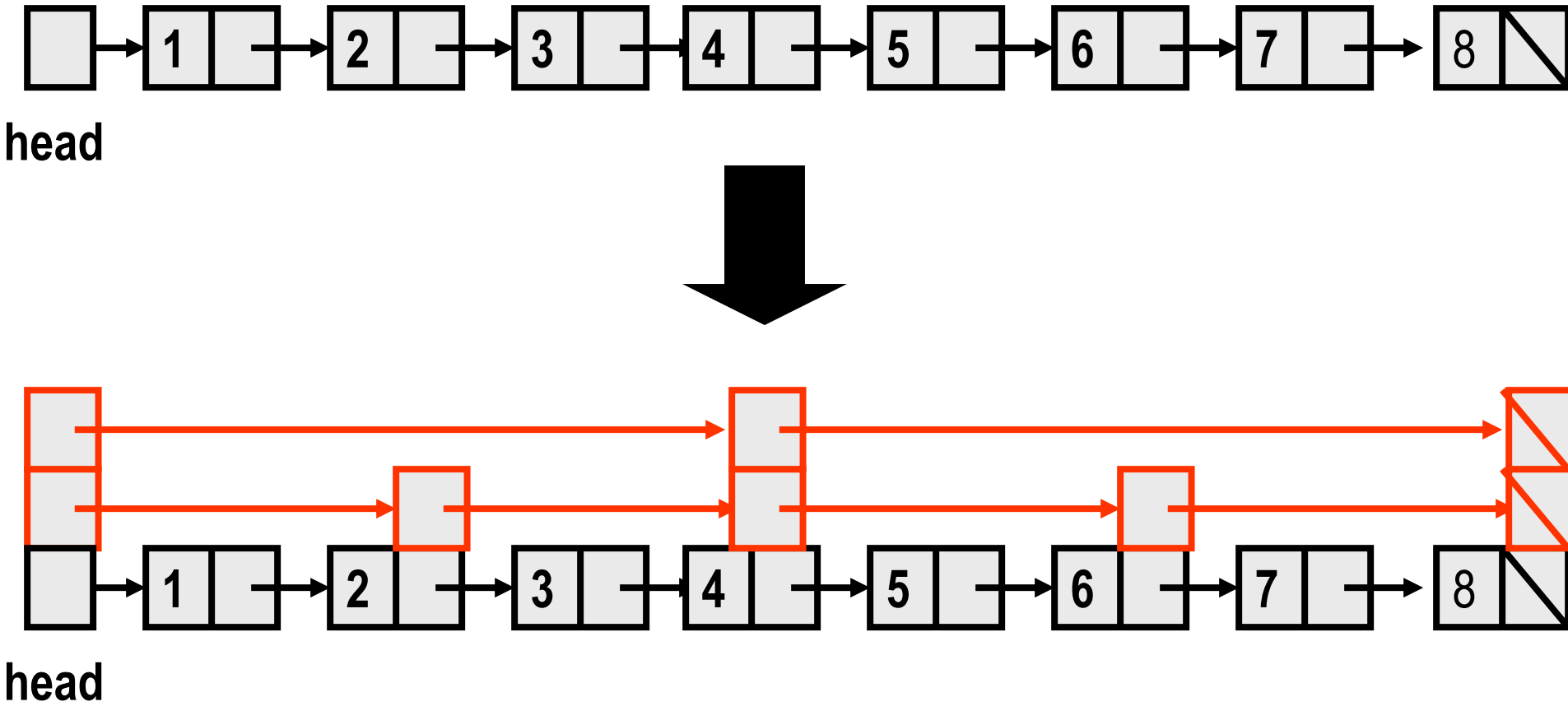
Perfect Skip List

- A **sorted** linked list + **Additional links**
- To improve the worst-case time complexity of the search operation!

Additional Skipping Links

- **Skipping Links**:
  - Every 2nd node connected.
    - Every 2nd element has 2 pointers.
  - Every 4th node connected
    - Every 4th element has 3 pointers.
  - Every 8th node connected
    - Every 8th element has 4 pointers.
  - And so on …

Example: A Perfect Skip List



Pointers in A Perfect Skip List

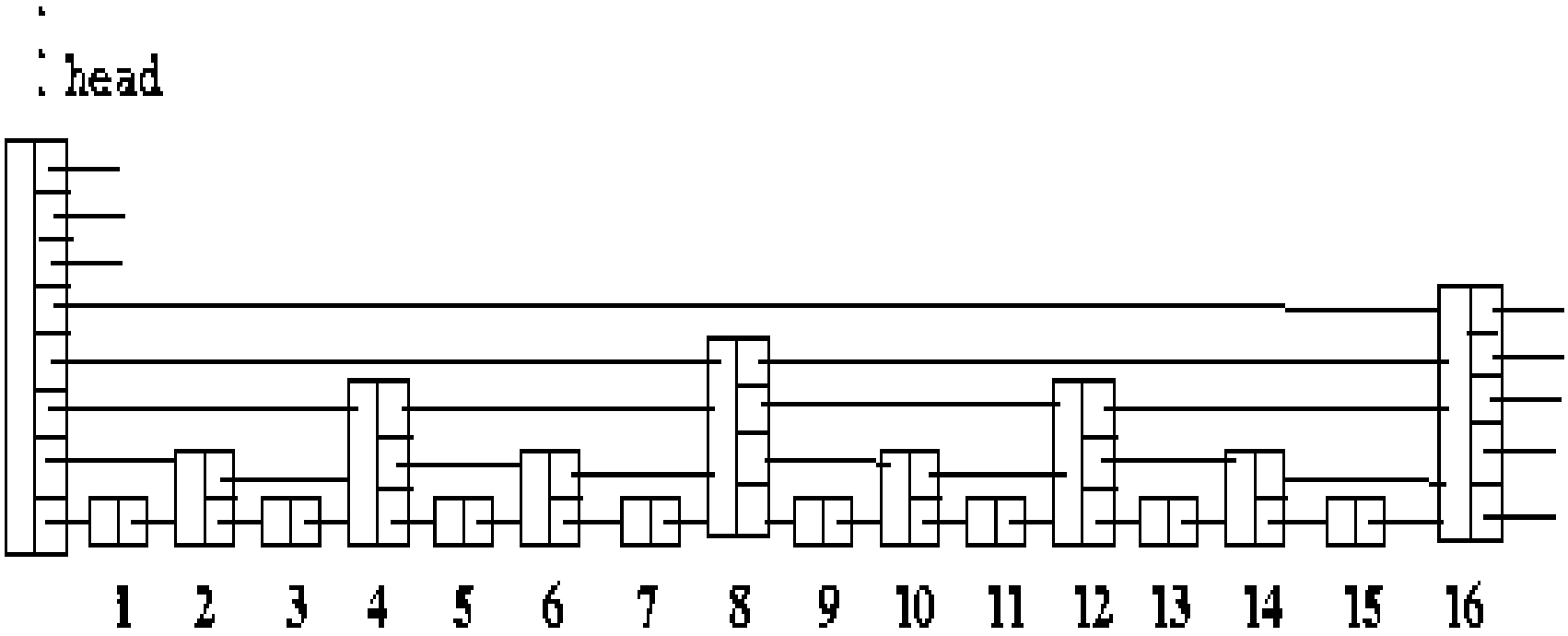
- Level-0 pointers:
  - ➔Original pointers
- Level-1 pointers:
  - ➔Pointers that connect every 2nd nodes.
- Level-2 pointers:
  - ➔Pointers that connect every 2^2-th nodes.
- Level-3 pointers:
  - ➔Pointers that connect every 2^3-th nodes.
- **Level-i pointers:**
  - ➔Pointers that connect **every 2^i-th nodes**.

Levels in A Perfect Skip List

- The level of a node is the number of pointers in the node.
  - ➔ **A level-i node has (i+1) pointers!**
- **The header node points to the first node of each level!**
- What is the highest (maximum) level of a skip list of N nodes?

  - ➔ **O(log N) levels!**

Example: A Perfect Skip List



Perfect Skip List

- In a perfect skip list, there are
  - 1/2 nodes at the level-0
  - 1/4 nodes at the level-1
  - 1/8 nodes at the level-2
  - 1/16 nodes at the level-3
  - and so on …

►QUIZ?

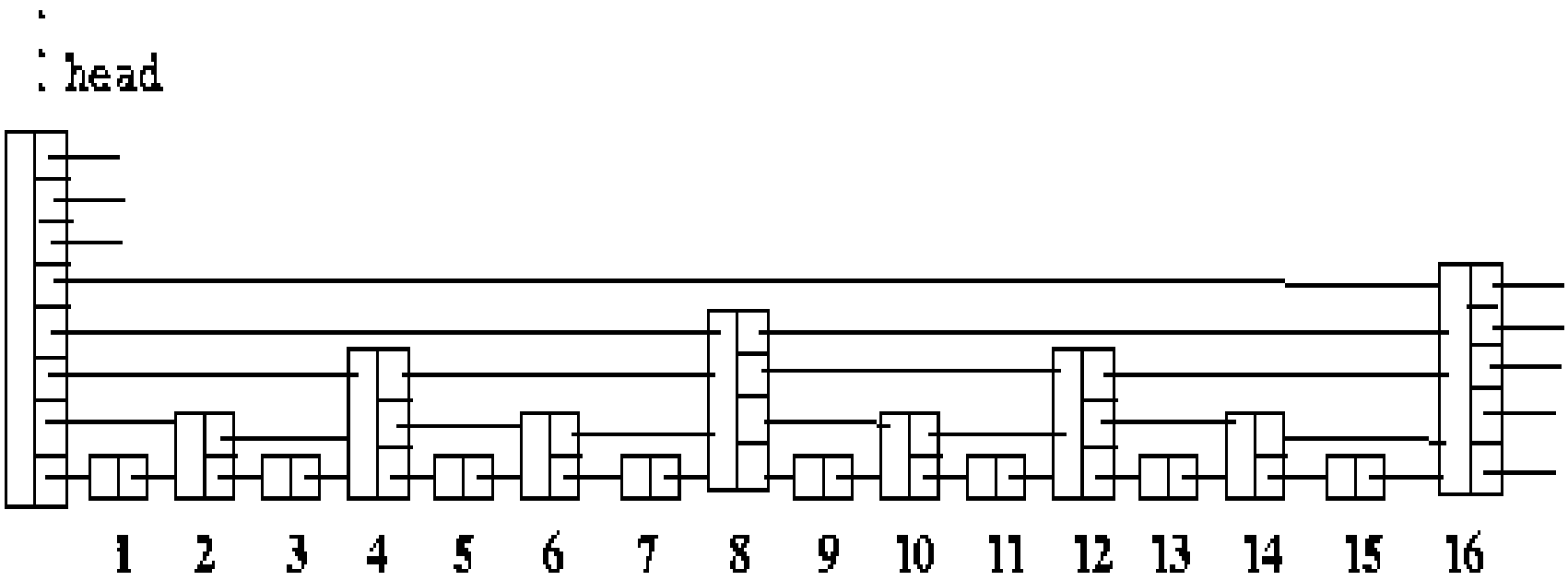
- Construct by inserting 19, 8, 12, 5, 17, 9 and 7 into an empty perfect skip list.

Search in A Perfect Skip List

- A non-sequential **search** like a **binary search** is possible!!!
  - ➔Start at the highest link (level) at the header;
  - ➔Traverse along this level;
  - ➔ If the next node is larger or NULL then go to the next lower level and continue the strategy;
  - ➔Stop at level 0;
- The worst-case time for **search** is
  - ➔**O(log n) time**

Example: A Perfect Skip List Search

- Search (9)?



Insert & Delete with A Perfect Skip List

- However, the **insertion** and **deletion** operations can be very inefficient!
  - ➔**Need restructuring!**
  - ➔**O(n) time**

- This expense is too great, so maintaining a perfectly balanced condition may not be not worth it!

Perfect Skip List – Time Complexity

- The worst-case time for **search**:
  - ➔ **O(log n) time**
- The worst-case time for **insertion** and **deletion** operations:
  - ➔ **O(n) time**

Randomized Skip List

Practically Good Skip List

- We want a not-perfect-but-practically-good skip list?
  - ➔**Relax the structuring condition!**
  - ➔To improve the average-case time complexity of the search operation!
- **Randomized (Probabilistic) Data Structure**
  - Makes some of its decision **at random**!
  - **Probabilistic DS vs Deterministic DS**

Good average-case time complexity!

Observation: Perfect Skip List

- In a perfect skip list, there are
  - 1/2 nodes at the level-0
  - 1/4 nodes at the level-1
  - 1/8 nodes at the level-2
  - 1/16 nodes at the level-3
  - and so on …

Idea: Randomized Skip List

- **Idea?**
  - Reduce the cost of insertion by selecting a random level for the new node such that **the proportion of level-0, level-1, level-2, etc. nodes is roughly preserved.**
  - Try to keep
    - 1/2 total nodes at the level-0
    - 1/4 total nodes at the level-1
    - 1/8 total nodes at the level-2
    - 1/16 total nodes at the level-3
    - …

Idea: Insert at Random Level with Probability

- **How?**

Good average-case time complexity!

  - **Determine a level randomly with probability.**
  - The level of a new element is chosen by flipping a coin (or generating a random integer):

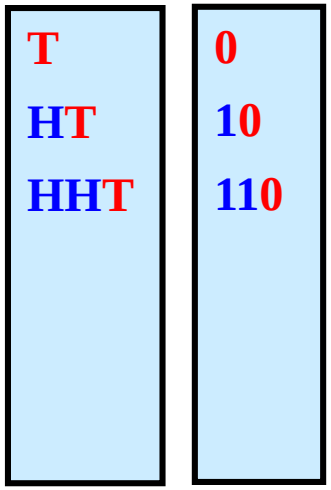
1. Flip a coin until we get a "tail" (even 0) or until the maximum number of levels is reached.
2. **The level of the new element** is **the number of times the coin came up "head"** (odd 1) **before it comes up tail** (even 0).

Random Level with Probability

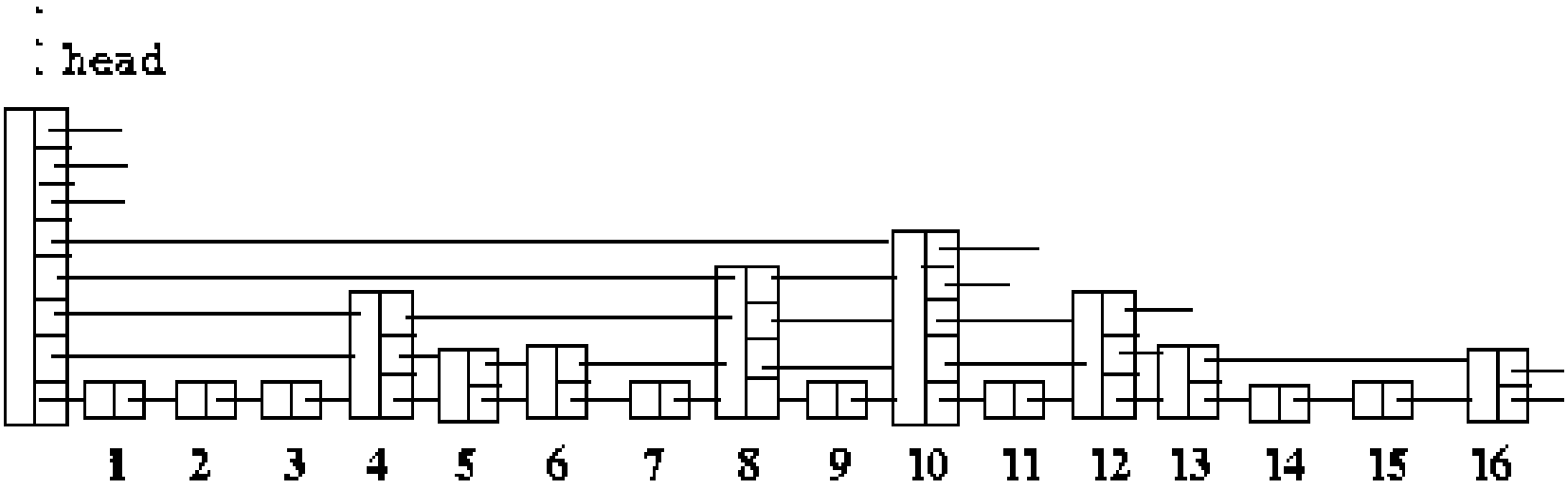
1. Flip a coin until we get a "tail" (even 0) or until the maximum number of levels is reached.
2. **The level of the new element** is **the number of times the coin came up "head"** (odd 1) **before it comes up tail** (even 0).

➔ **Try to keep**

- 1/2 total nodes at the level-0
- 1/4 total nodes at the level-1
- 1/8 total nodes at the level-2
- 1/16 total nodes at the level-3
- …



Example: A Randomized Skip List



Insert in A Good Skip List

- Insert:
  - Find the insertion point using search;
  - Keep track of each pointer where we switch to a lower level;
  - **Determine the level randomly with probability for the new node!**
  - Splice the new node into the list;

  - **Randomized Algorithm!**
  - **Randomized Data Structure!**

►QUIZ?

- Construct by inserting 19, 8, 12, 5, 17, 9 and 7 into an empty randomized probabilistic skip list.

►QUIZ?

- **Insert** to a randomized skip list?

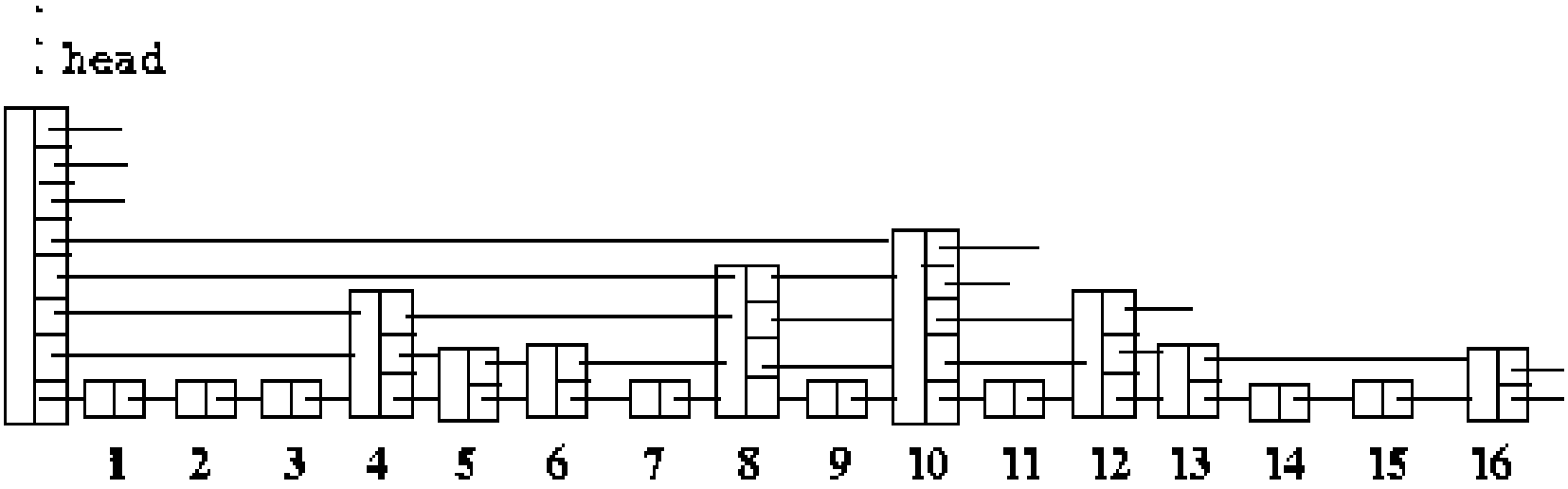
Insert in A Good Skip List

- **Worst case: A sorted linked list! (All Tails!)**
  - ➔ **O(n) time**
- **Average (Expected) case:**
  - ➔ **O(log n) time**

Good average-case time complexity!

Example: A Randomized Probabilistic Skip List Search

- Search (9)?



Search in A Good Skip List

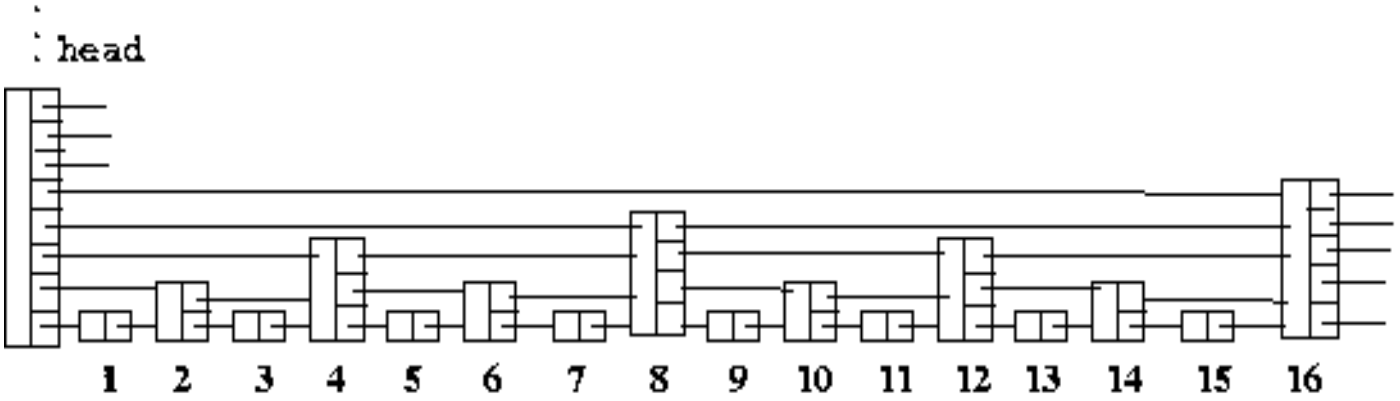
- Not perfect anymore, but we can still skip large distances and so the performance is still good.
- **Worst case: A sorted linked list!**
  - ➔ **O(n) time**
- **Average (Expected) case:**
  - ➔ **O(log n) time**

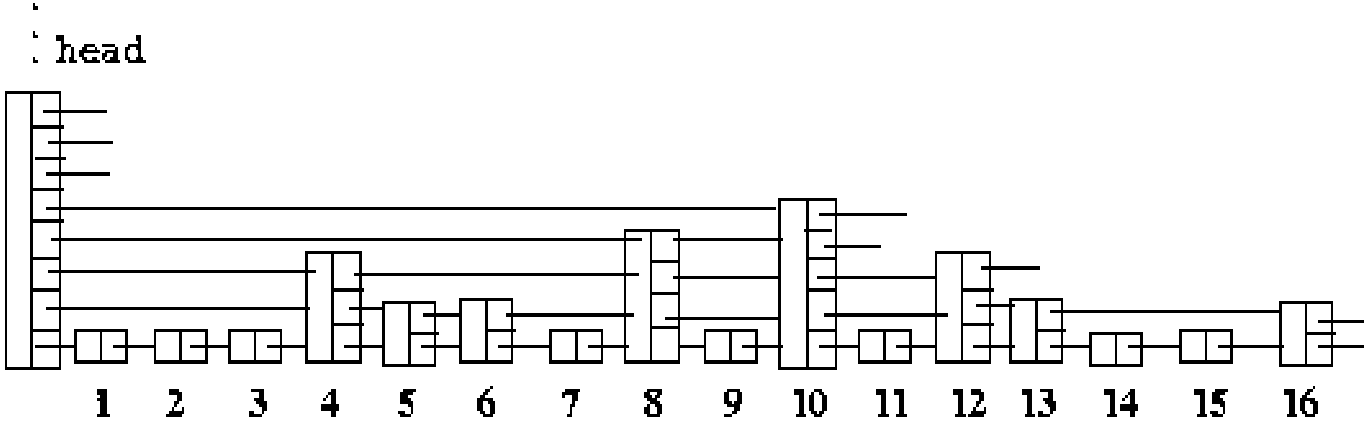
Good average-case time complexity!

A Good Skip List – Time Complexity

- The time complexity for **search**:
  - **Worst case: O(n) time**
  - **Average (Expected) case: O(log n) time**
- The time complexity for **insertion** and **deletion** operations:
  - **Worst case: O(n) time**
  - **Average (Expected) case: O(log n) time**

Example: Perfect vs Randomized Skip Lists





►QUIZ?

- Compare the **perfect skip lists** with the **randomized skip lists**?

►QUIZ?

- Compare the **Randomized Skip Lists** with the **Binary Search Trees**?

Skip List History

- *Skip lists: a probabilistic alternative to balanced trees*, Communications of the ACM, June 1990, 33(6), pp. 668-676.

  Skip lists are a data structure that can be used in place of balanced trees. Skip lists use probabilistic balancing rather than strictly enforced balancing and as a result the algorithms for insertion and deletion in skip lists are

  much simpler and significantly faster than equivalent algorithms for balanced trees.

Skip List Visualization

- *Skip list Visualization*

Self-Organizing (Linked) Lists

(Self-Adjusting Data Structures)

Self-Organizing Lists

- Why?
  - Linked lists – Linear search O(n) worst-case & average-case
- Goal:
  - To improve the **amortized (average time over operations) time** complexity of the linear **search** operation.
  - Not worst-case time!

Self-Organizing Lists

- Idea?
  - Locality of reference
  - (80/20) 80% of the access are to 20% of the items!

Self-Organizing Lists

- How?
  - By dynamically organizing (restructuring or adjusting) the list in a certain manner.
  - By moving more frequently accessed items towards the head of the list for future operations!
  - **Self-Adjusting Linked Lists**
  - **Self-Adjusting Data Structures!**

Self-Organizing Lists

- Approach?
  - ***Place more frequently accessed items closer to head!***
- **Search using Heuristics!**
  - Move to front (MTF)
  - Transpose
  - Frequency Count
  - …

Self-Organizing Lists

- A self-organizing list is a list that reorders its elements based on some self-organizing heuristic to improve amortized time.

- Amortized Analysis:
  - ➔ The cost of the MTF self-organizing list heuristic is **never more than twice** the cost for the *optimal* static ordering of the list.

Self-Organizing Methods

- **Move to front (MTF)**: After the desired element is located, put it at the beginning of the list.
- **Transpose (TR)**: After the desired element is located, swap it with its predecessor unless it is at the head of the list.
- **Frequency Count (FC)**: Order the list by the number of times elements are being accessed.

Move-to-front Method

- Move-to-front method
  - ➔Move it to the beginning of the list.
    - A B C D
    - Search D
    - D A B C

    - A B D
    - Insert C
    - A B D C

Transpose Method

- Transpose method
  - ➔Swap it with its predecessor.
    - A B C D
    - Search D
    - A B D C

    - A B D
    - Insert C
    - A B D C

Frequency Count Method

- Frequency Count method
  - ➔Order the list by the number of times accessed.
    - A3 B1 C1 D1
    - Search D
    - A3 D2 B1 C1
    - A3 B1 D1
    - Insert C
    - A3 B1 D1 C1

►QUIZ?

- Access A, C, B, **C**, D, **A**, **D**, **A**, **C**, **A**, **C**, **C**, E and **E**. (NOTE: Black= Insert, Blue= Search)

(Insert **A**, Insert **C**, Insert **B**, Search **C**, Insert **D**, Search **A**, Search **D**, Search **A**, Search **C**, Search **A**, Search **C**, Search **C**, Insert **E** and Search **E**.)

- ➔Plain (search_plain)
- ➔Move-to-front method (search_mtf)
- ➔Transpose method (search_t)
- ➔Frequency Count method (search_fc)

►QUIZ:

- Access A, C, B, **C**, D, **A**, **D**, **A**, **C**, **A**, **C**, **C**, E and **E**.

  - ➔Plain ACBDE
  - ➔Move-to-front method ECADB
  - ➔Transpose method CADEB
  - ➔Frequency Count method CAEDB

►QUIZ?

- Compare the **perfect skip lists** vs the **randomized skip lists** vs the **self-organizing lists**?

Self-Organizing Singly Linked List Implementation

```
template <class DT>
class SOLListNode
{
public:
        SOLListNode() { }
        SOLListNode(const DT& theData, SOLListNode<DT>* theLink)
                : data(theData), link(theLink) { }

        DT data;
        SOLListNode<DT>* link;
};
```

Self-Organizing Singly Linked List Implementation

```
template <class DT>
class SOLList
{
public:
 // Constructor & Copy constructor
 SOLList ();
 SOLList ( const SOLList<DT> &srcList );

 // Destructor
 ~SOLList ();

 // List manipulation operations
 void insert ( const DT &newData );
 bool search_mtf ( const DT &key);
 bool search_t ( const DT &key);

...

private:
        SOLListNode<DT> *head;
};
```

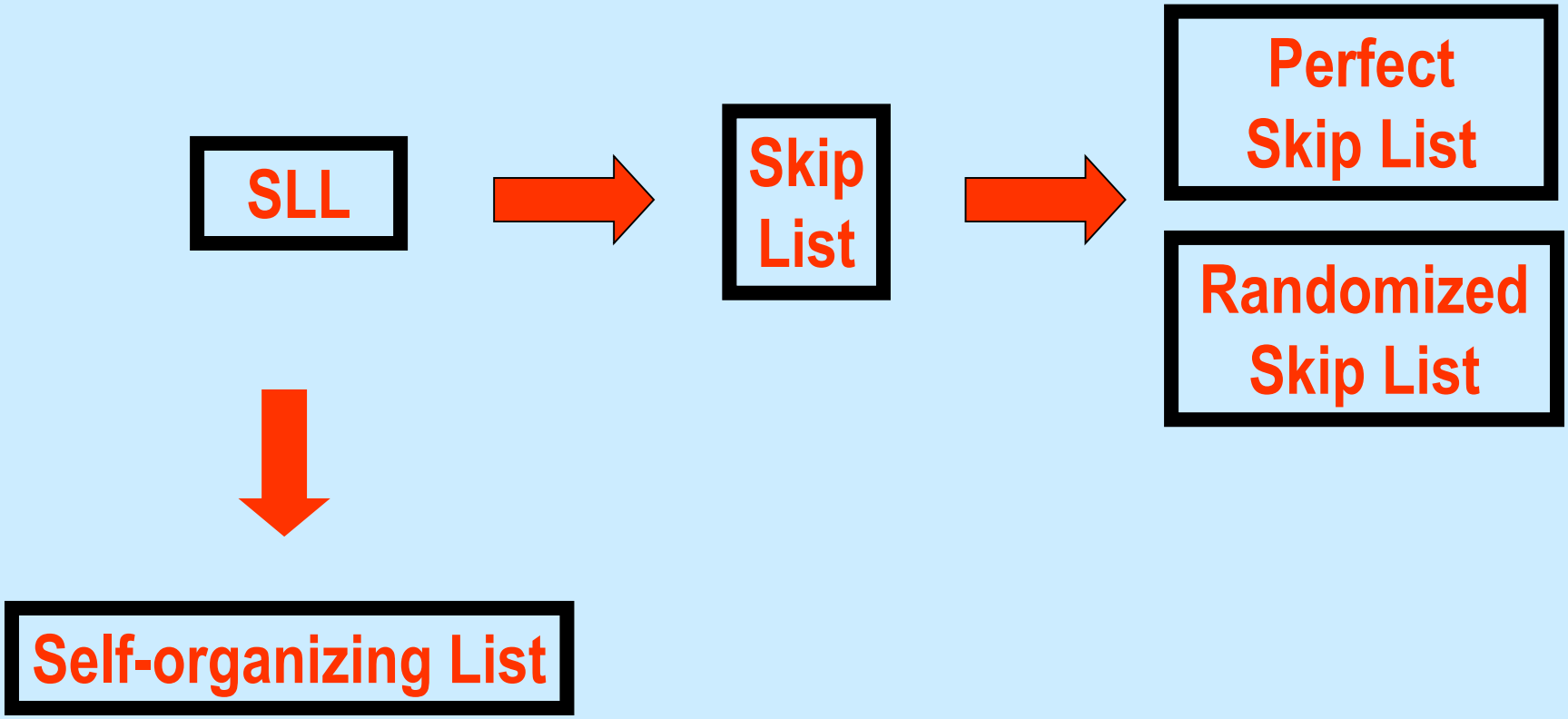
Self-Organizing Singly Linked List Implementation

```
template <class DT>
SOLList<DT>::SOLList()
  : head(NULL) { }


...

SOLList<int> l1, l2;
l1.insert(1); l2.insert(1);

...
l1.search_mtf(1);

...
l2.search_t(1);

...
```

Advanced Linked Lists Summary

- Skip lists – Perfect vs. Randomized
- Self-organizing lists - Heuristics

SLLs, Skip Lists & Self-organizing Lists



Homework Assignment

►Homework Assignment?

- Draw the **perfect skip list** that results when you insert items with the keys 19, 6, 26, 9, 2, 12, 25, 7, 21 and 17 in that order into an initially empty perfect skip list.
- Draw the **randomized skip list** that results when you insert items with the keys 19, 6, 26, 9, 2, 12, 25, 7, 21 and 17 in that order into an initially empty randomized skip list.

►Homework Assignment?

- Compare the **binary search tree** with the perfect skip list and with the **randomized skip list.**

►Homework Assignment

- ***Advanced Linked List:* A Self-Organizing Linked List with Move-To-Front & Transpose**
- Design and implement the Self-Organizing (Unsorted) Singly-Linked List with Move-To-Front & Transpose ADT.

  - You should include ***search*** *(linear search using no self-organizing heuristic)*, ***search_mtf*** *(linear search using the* ***move-to-front*** *self-organizing heuristic)*, ***search_t*** *(linear search using the* ***transpose*** *self-organizing heuristic)*, ***insert*** and ***showSOLL***.
  - The ***insert*** operation scans the entire list to verify that the item is not already present and then inserts the item *at the end of the list*.
  - The operation ***showSOLL*** prints the values in the self-organizing linked list in a linked list form.

Test Cases

- Insert **A**, Insert **C**, Insert **B**, Search **C**, Insert **D**, Search **A**, Search **D**, Search **A**, Search **C**, Search **A**, Search **C**, Search **C**, Insert **E** and Search **E**.

  - Plain ACBDE
  - Move-to-front method ECADB
  - Transpose method CADEB

the right majors, minors &

ucation?

or students' academic and career success

ging for many students - *Many students change their*
uring college!

e prediction of student success in MMC could
dual students
d their right MMC
chieve their academic goals

END

Prof. Young Park