Introduction to Algorithms and Data Structures

5.5. Data Structure (0): Array and Linked List

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.
uehara@jaist.ac.jp
http://www.jaist.ac.jp/~uehara

http://www.jaist.ac.jp/~uehara/course/2020/myanmar/

Algorithms and Data Structure

- Algorithm: Method for solving a problem
- Data Structure:
  - Way for storing data and intermediate values
  - Influence on efficiency of a computation
  - Depending on algorithms
  - Example: array, linked list, stack, queue, heap, tree

ARRAY AND LINKED LIST

[First data structure!]

Array: Easy to access

- Put data on consecutive area in memory
- We can access any index in a constant time : Random Accessibility

  ⇔ (Linked list does not have this property)

- Easy to access in order of indices : Sequential accessibility

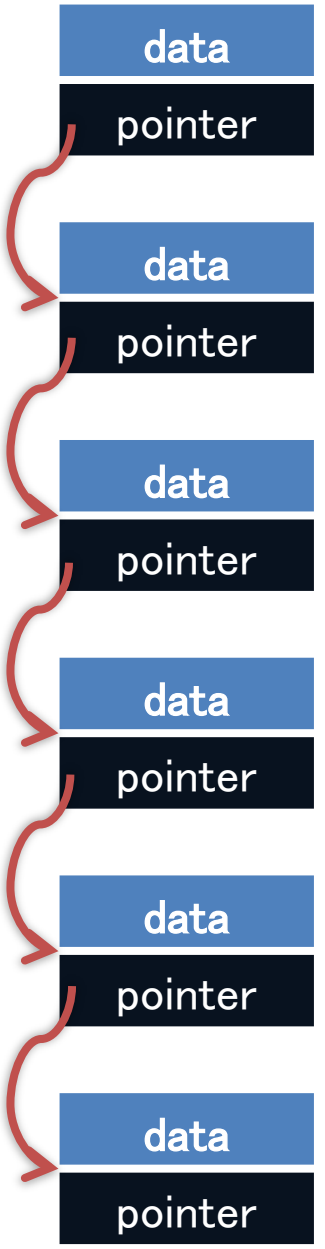
  ⇔ (Some data may have no ordering)
  (E.g. some tree structure)

[Second data structure]

Linked list

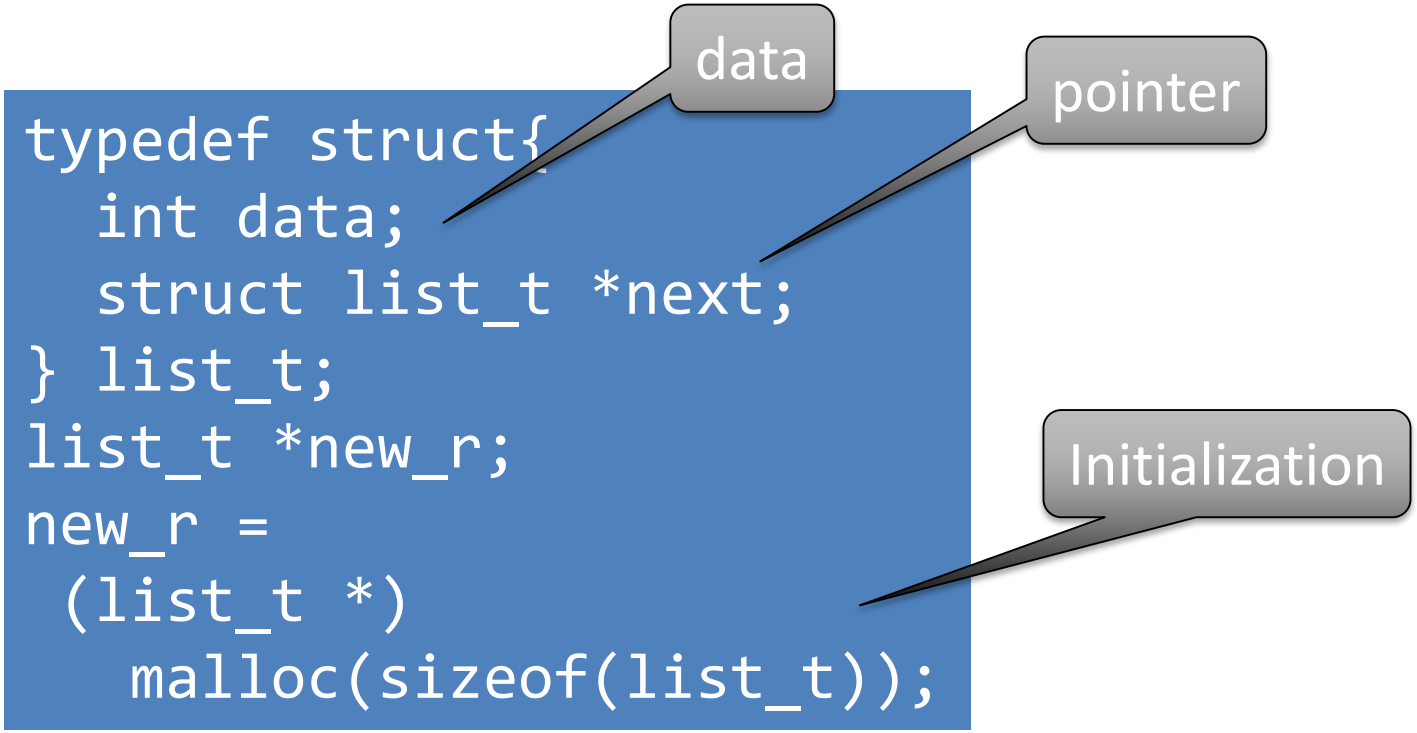
- Sequence of records
- They can be located in any places in memory
- Each item stores the place(s) of its “next” (and “previous”) data
  - Data: storing data
  - Pointer: indicates its “next” data (by address)

- Some variations
  - One-way linked list: Store “next”
  - Two-way linked list: Store “next”/“previous”
  - Tree can be: Store “neighbors”



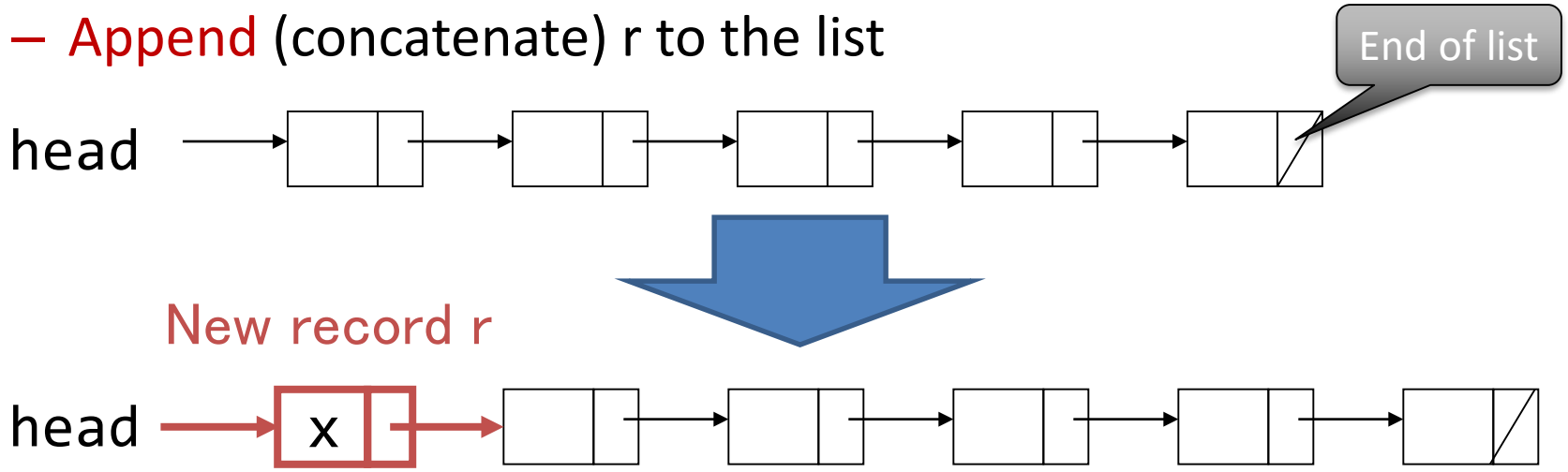
One-way linked list

- Sequence of records
  - Data: store data
  - Pointer: indicates its “next” data (by address)



Example: Store many data into one-way linked list

- Basic:
  - Initialize list. head indicates the top data
  - For each data x, make a record r that has x in data area
  - Append (concatenate) r to the list

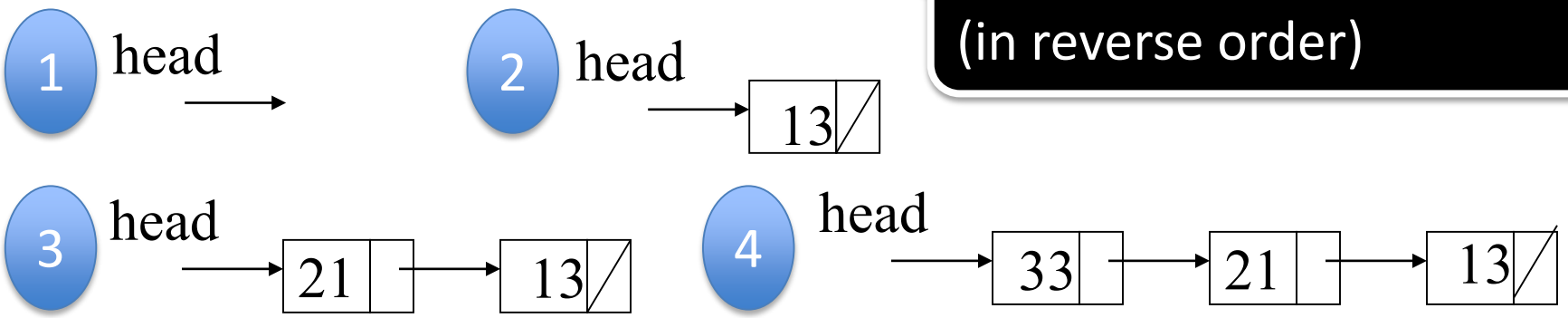


- We append at the top or last of the list

A program for adding new records at the top of the linked list

```
list_t *head, *new_r;
head = NULL;    (It is empty at first)
while(/* we have data */){
    int x = input data;
    new_r = (list_t *)
        malloc(sizeof(list_t));
    new_r->data = x;
    new_r->next = head;
    head = new_r;
}
```

New record is added at the top (in reverse order)



[Important point!]

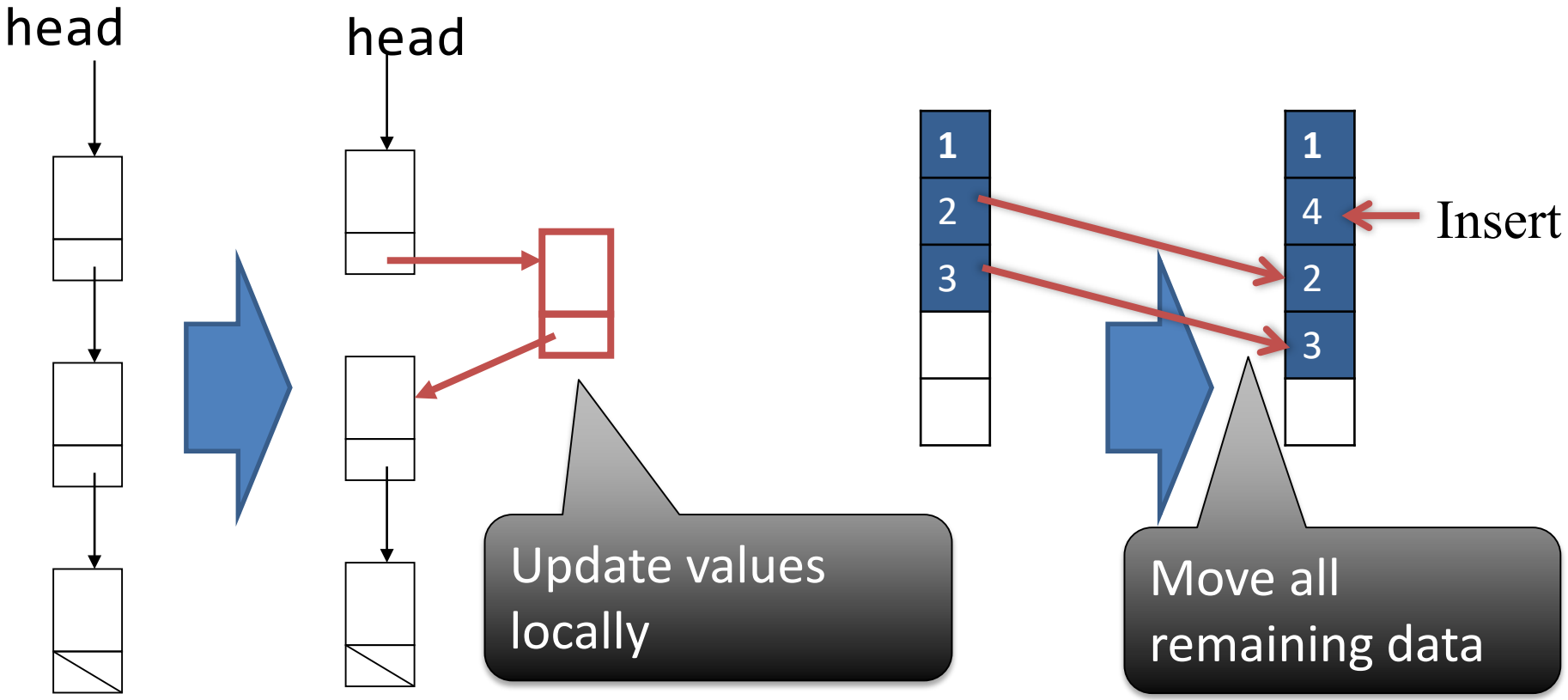
Advantage of linked list (comparing to array): Easy to add/remove data

Linked list

- No move of data

Array

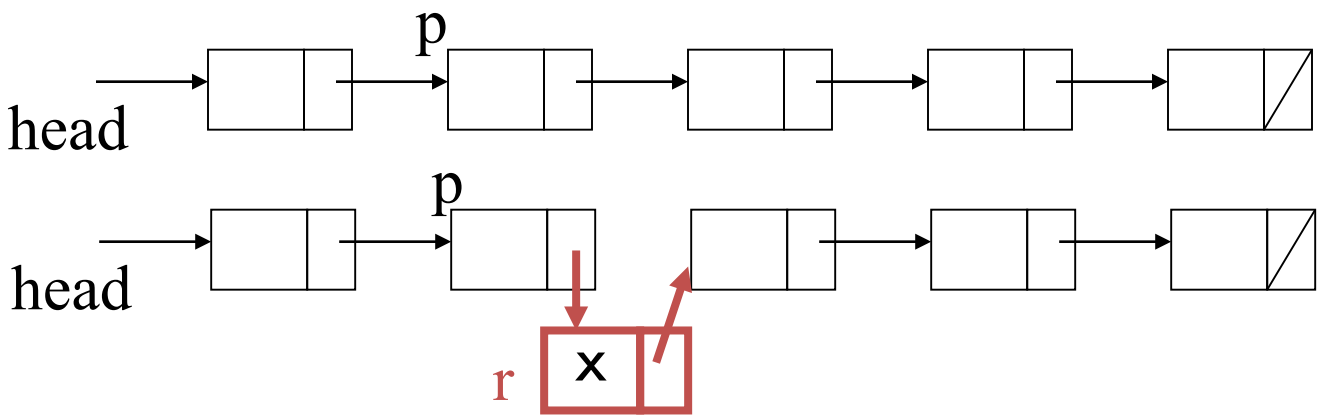
- (Many) data should be shifted



One-way linked list: ＜Insertion of data＞

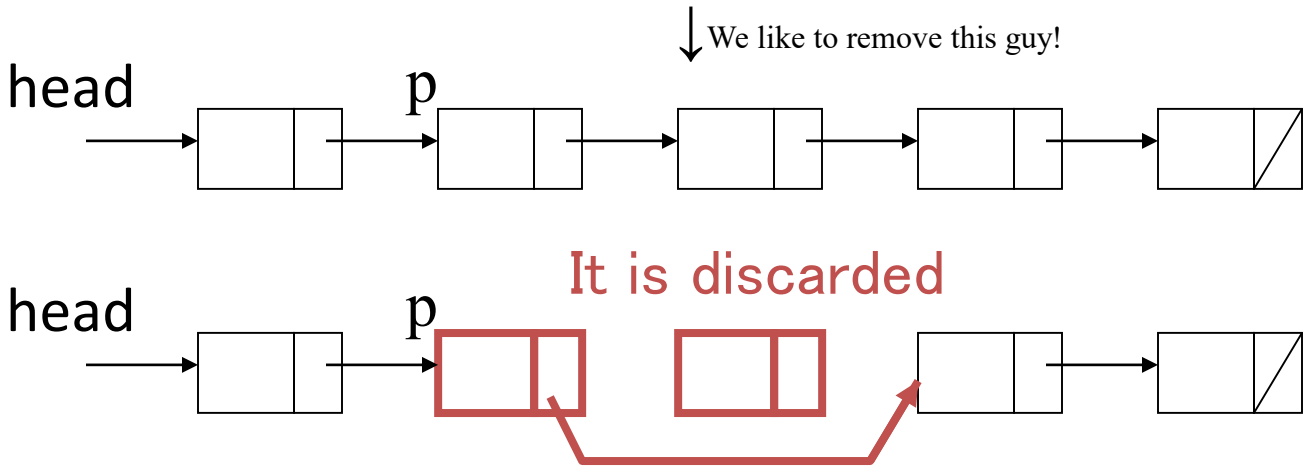
- Insert data x after the node p
  - Make a record r
  - “Next node of r” was “next node of p”
  - “next node of p” becomes r

```
void insert (Node p, data x) {
    Node r = new Node(x, p.next);
    p.next = r;
}
```



One-way linked list:
＜Remove data (1)＞

- Remove the next node of node p
  → We can skip it (without removal)




```
void deleteNextNode (Node p) {
    p.next = p.next.next;
}
```

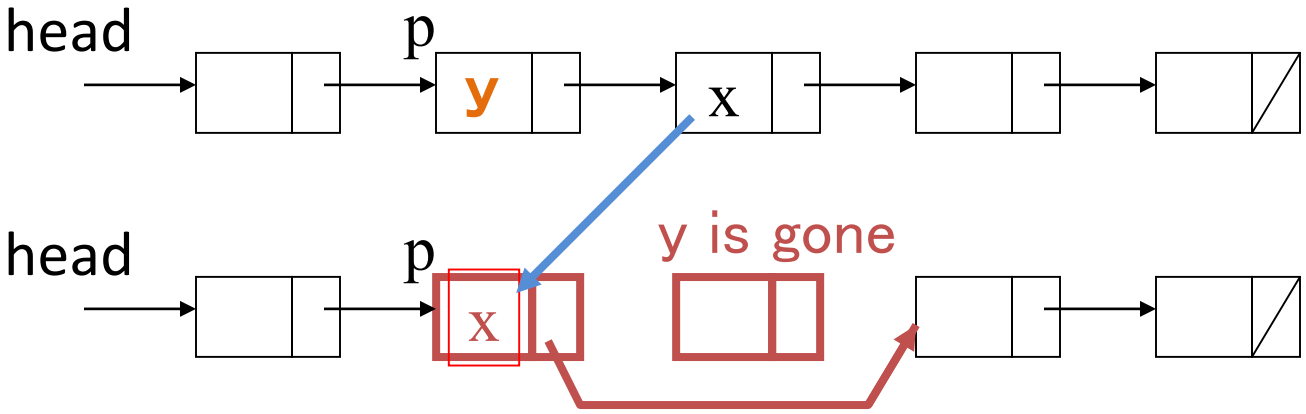
[bit tricky!]

One-way linked list:
＜Remove data (2)＞

- Remove the current node p (or its data y)

  → We do not remove p itself, but we remove the next node q of p.

  Before removing, copy the data x from q to p.




```
void delete (Node p) {
    p.data = p.next.data;
    p.next = p.next.next;
}
```

Properties of one-way linked list

- Advantage: Easy to insert/remove

  O(1) time ⇔ array requires O(n) time

- Disadvantage: “Taking the i-th data” is slow…

  O(n) time ⇔ array requires O(1) time

Consider:

  - In practical, when should you use array, and when should you use one-way linked list?
  - How about advantage/disadvantage of one-way and two-way linked lists?