Linked Lists

Saikrishna Arcot
M. Hudachek-Buswell

July 19, 2020

Linked List

- A linked list is a data structure consisting of a sequence of nodes.

Linked List

- A linked list is a data structure consisting of a sequence of nodes.
- Each node in the linked list stores the data and a pointer to the next/previous node(s).

Linked List

- A linked list is a data structure consisting of a sequence of nodes.
- Each node in the linked list stores the data and a pointer to the next/previous node(s).
- Nodes are created/destroyed as necessary. When adding a new element, a new node is created, and linked into the linked list. When removing an element, the node containing the element is destroyed.

Linked List

- A head pointer (pointer to the first node in the list) is necessary, while a tail pointer is optional.

Linked List

- A head pointer (pointer to the first node in the list) is necessary, while a tail pointer is optional.
- Linked lists are better for manipulating data, while array lists are better for storing/accessing data.

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:
  - the element

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:
  - the element
  - pointer to the next node

Singly Linked List

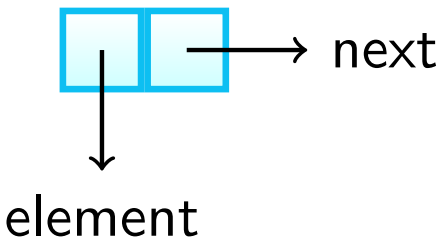
- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:
  - the element
  - pointer to the next node
- Example of a node

Singly Linked List

- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:
  - the element
  - pointer to the next node
- Example of a node

Singly Linked List

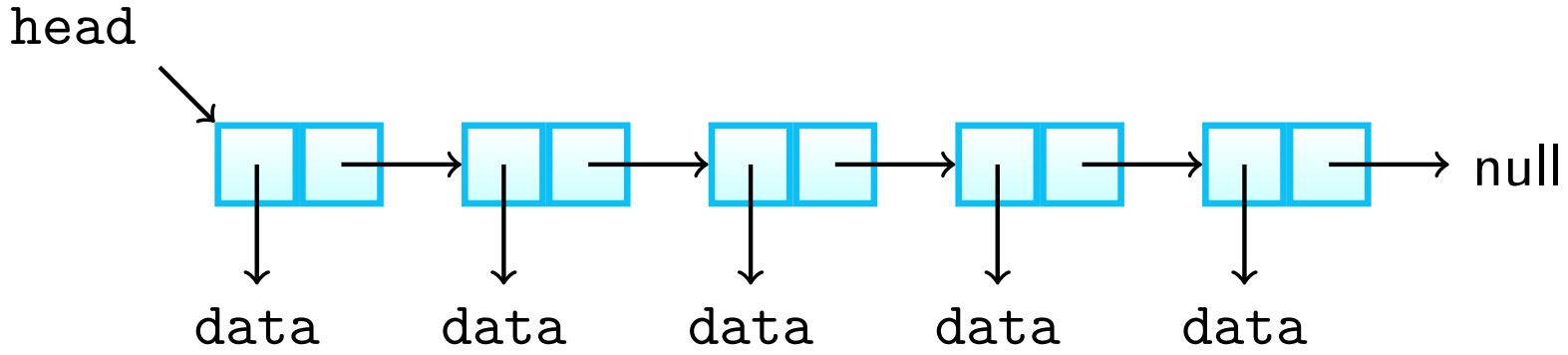
- A singly linked list is a version of a linked list where each node *only* has a pointer to the next node (i.e. it is not possible to go back to the previous node). The singly linked list starts from the head pointer.
- Each node stores:
  - the element
  - pointer to the next node
- Example of a node



Singly Linked List

In this course, we refrain from creating "dummy" nodes that lead to null references at the end of a singly linked list. We just have the next reference of the last node point to null.

Example of a singly linked list:



Inserting at the Head

- Allocate a new node with data and next reference.

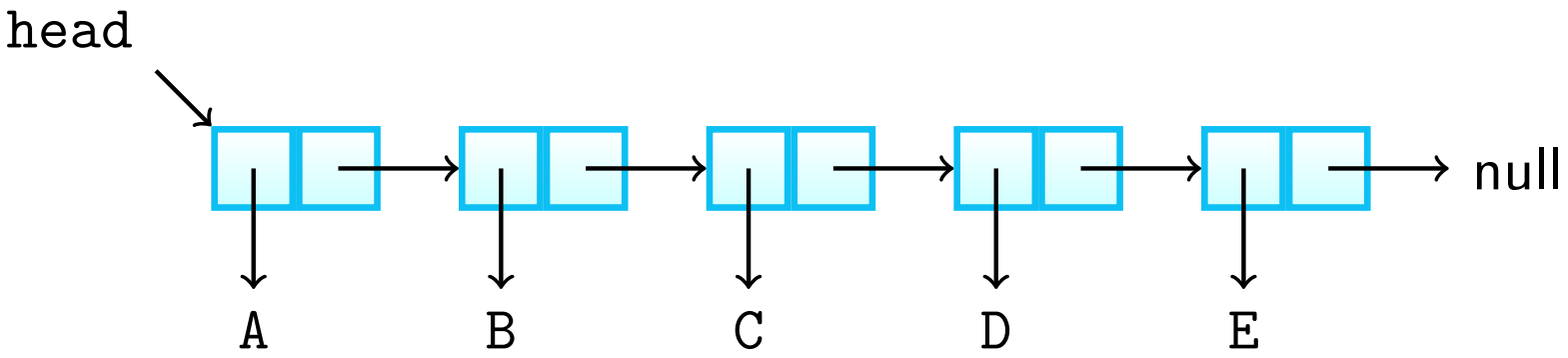
Inserting at the Head

- Allocate a new node with data and next reference.
- Have the new node's next reference point to old head.

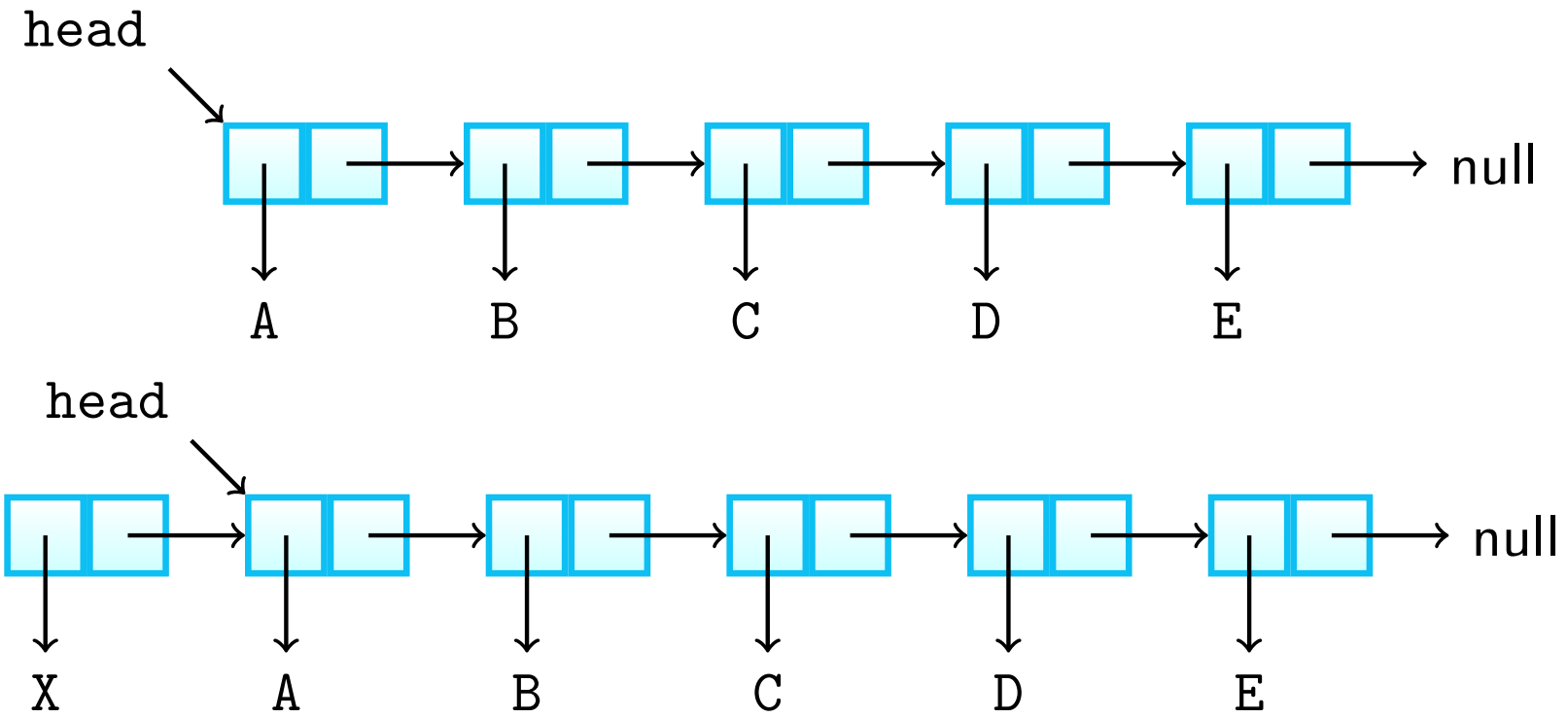
Inserting at the Head

- Allocate a new node with data and next reference.
- Have the new node's next reference point to old head.
- Update the head reference to point to new node.

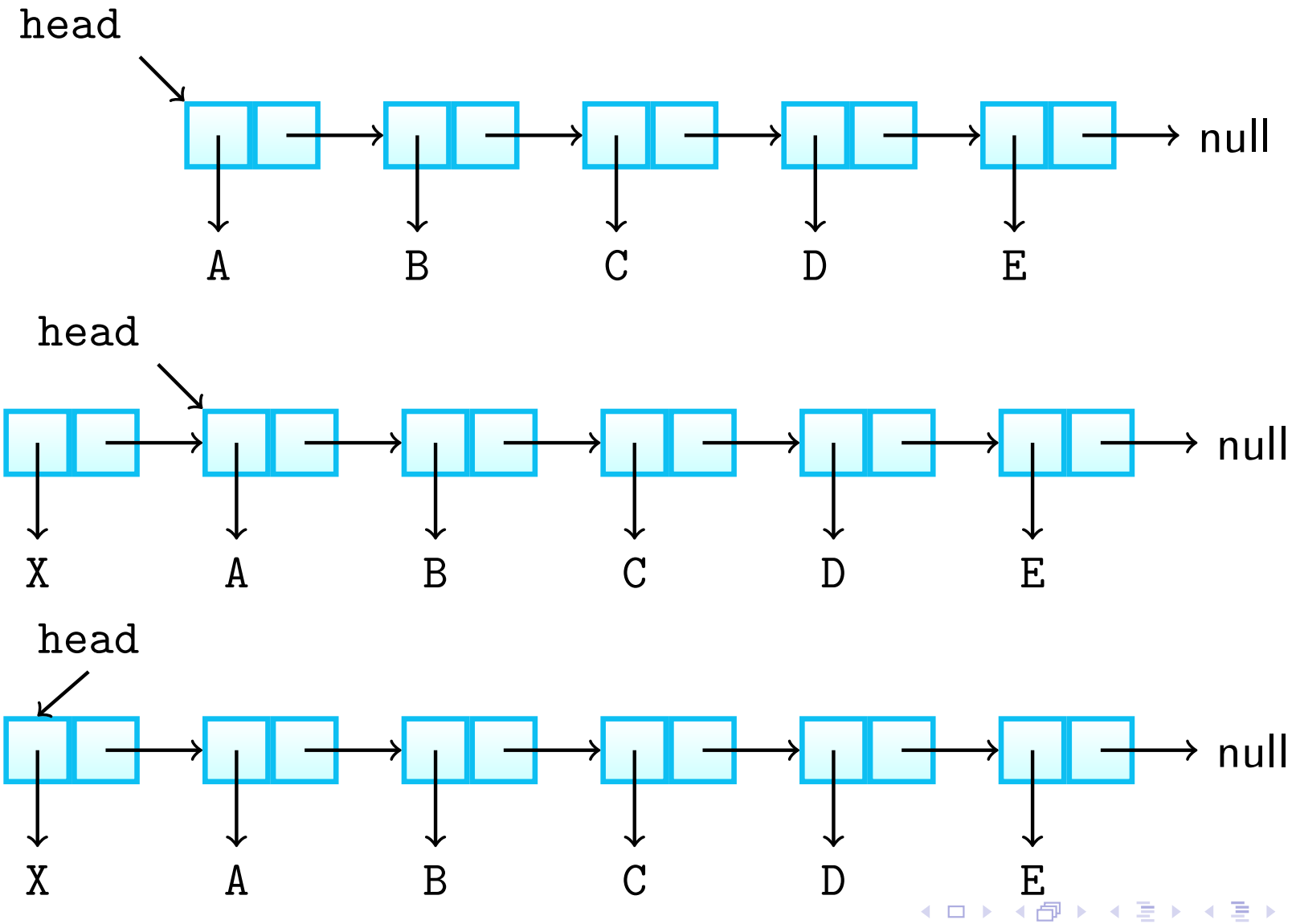
Inserting at the Head



Inserting at the Head



Inserting at the Head



Inserting at the Tail

- Allocate a new node with data and next reference.

Inserting at the Tail

- Allocate a new node with data and next reference.
- Have new node's next reference point to `null`.

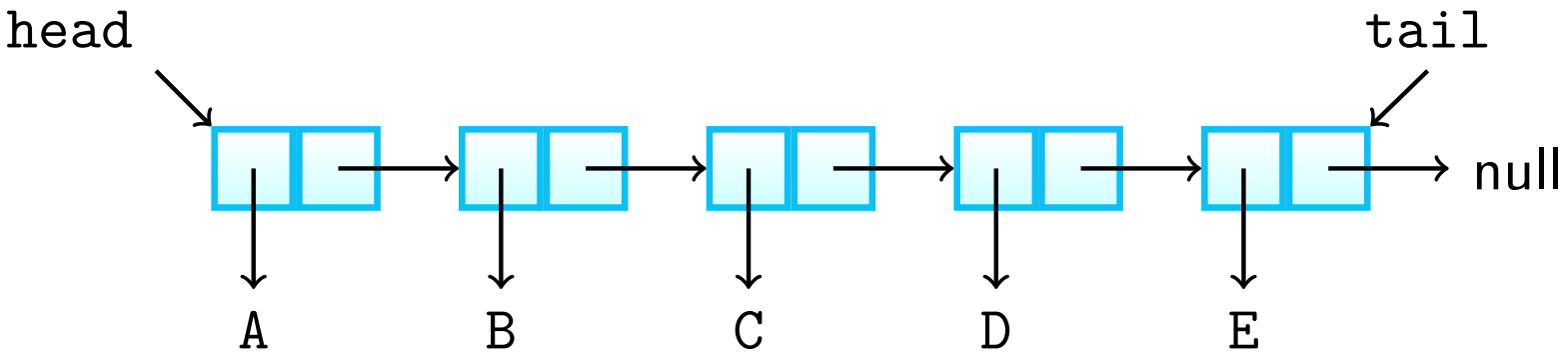
Inserting at the Tail

- Allocate a new node with data and next reference.
- Have new node's next reference point to `null`.
- Have the last node's next reference point to new node.

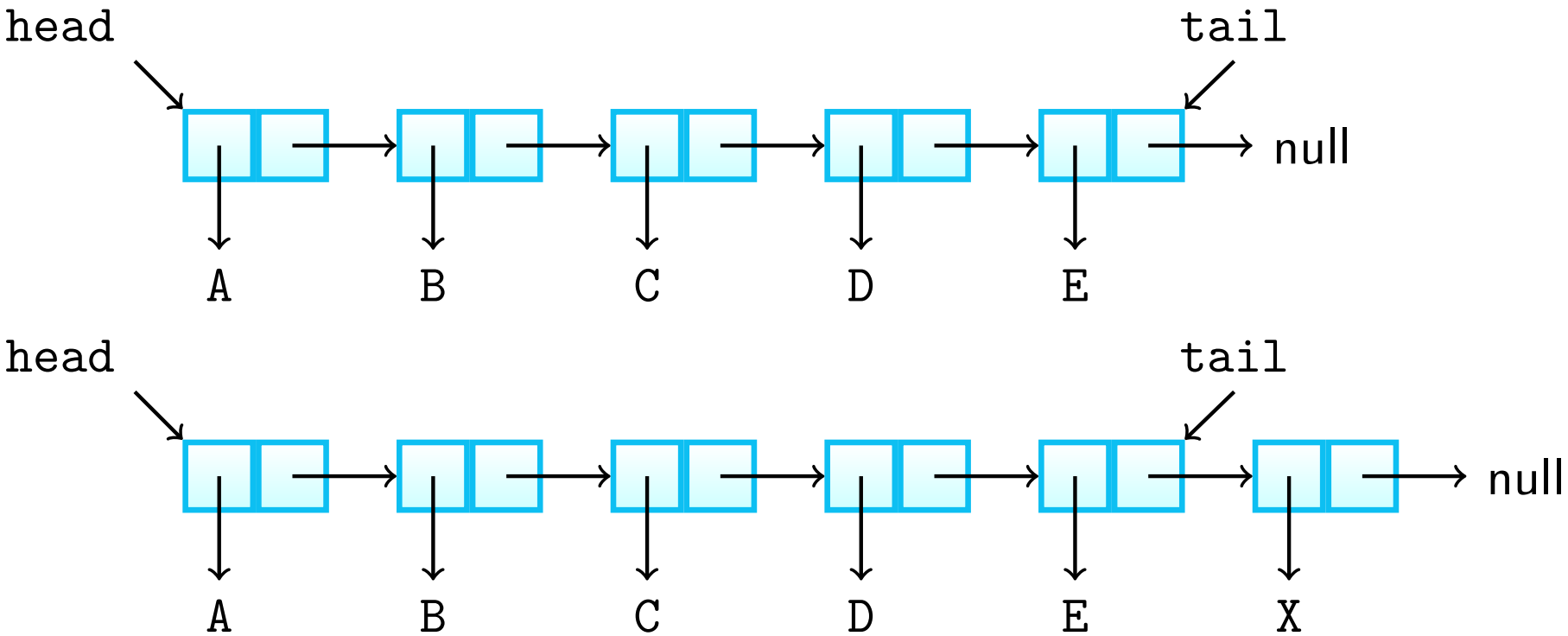
Inserting at the Tail

- Allocate a new node with data and next reference.
- Have new node's next reference point to `null`.
- Have the last node's next reference point to new node.
- Update tail (if there is a tail pointer) to point to new node.

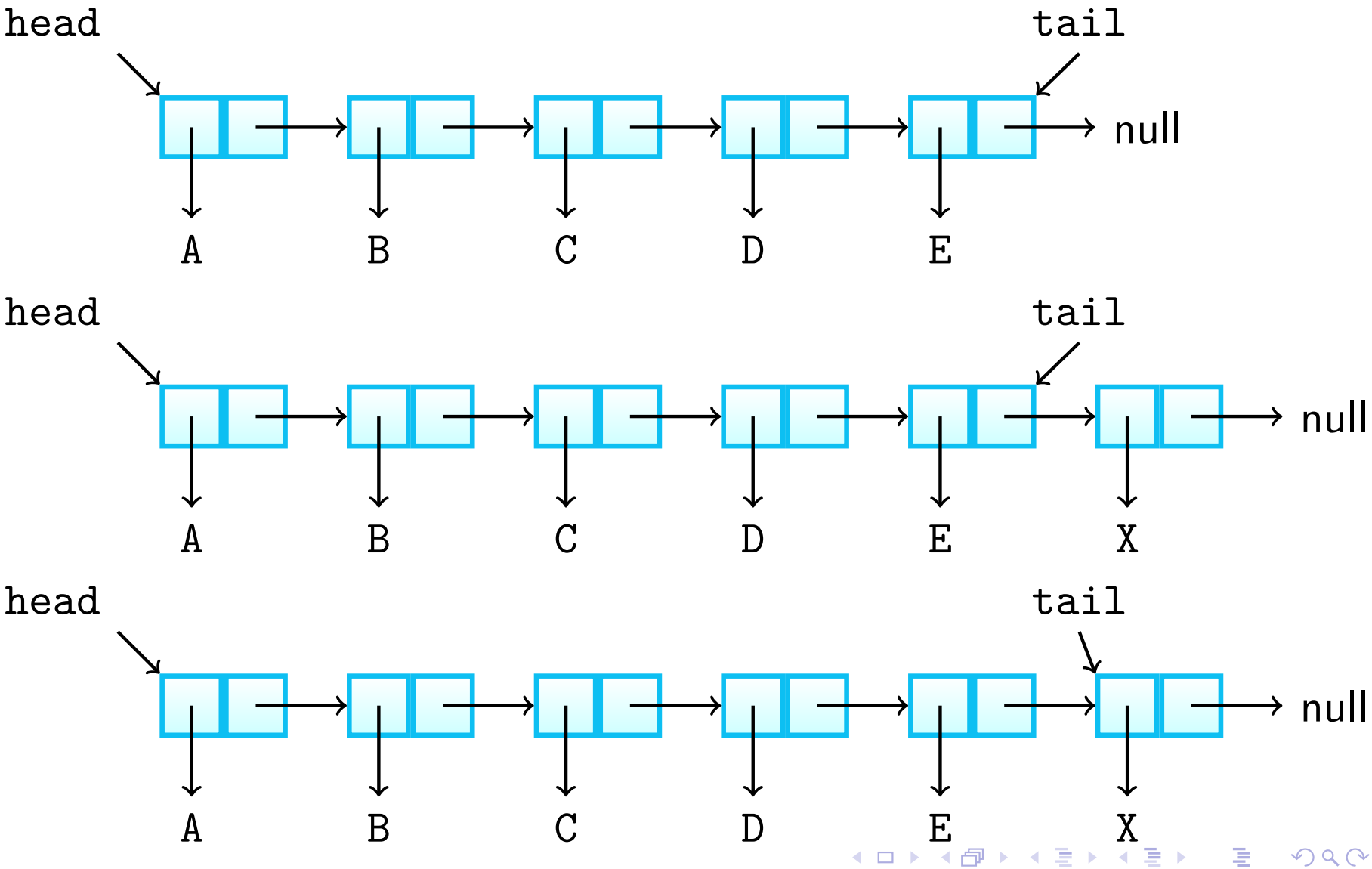
Inserting at the Tail



Inserting at the Tail



Inserting at the Tail



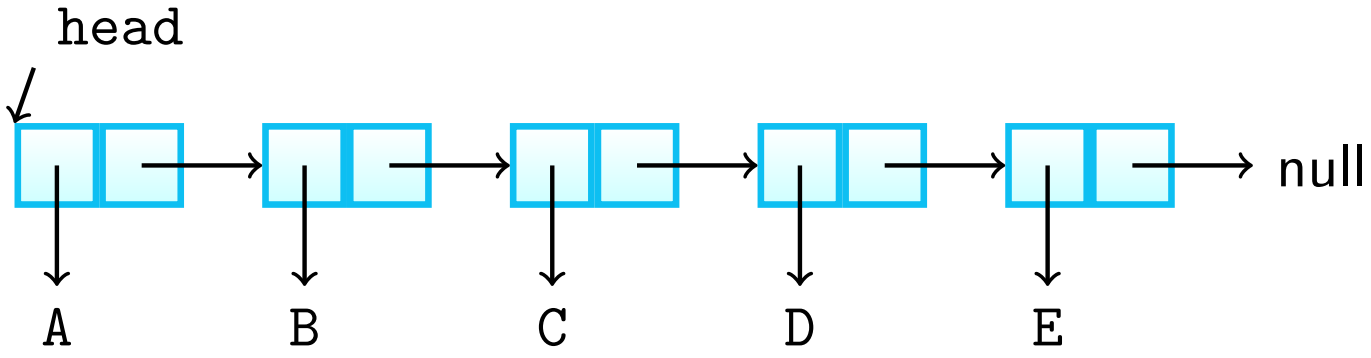
Removing at the Head

- Update head to point to the next node in the list.

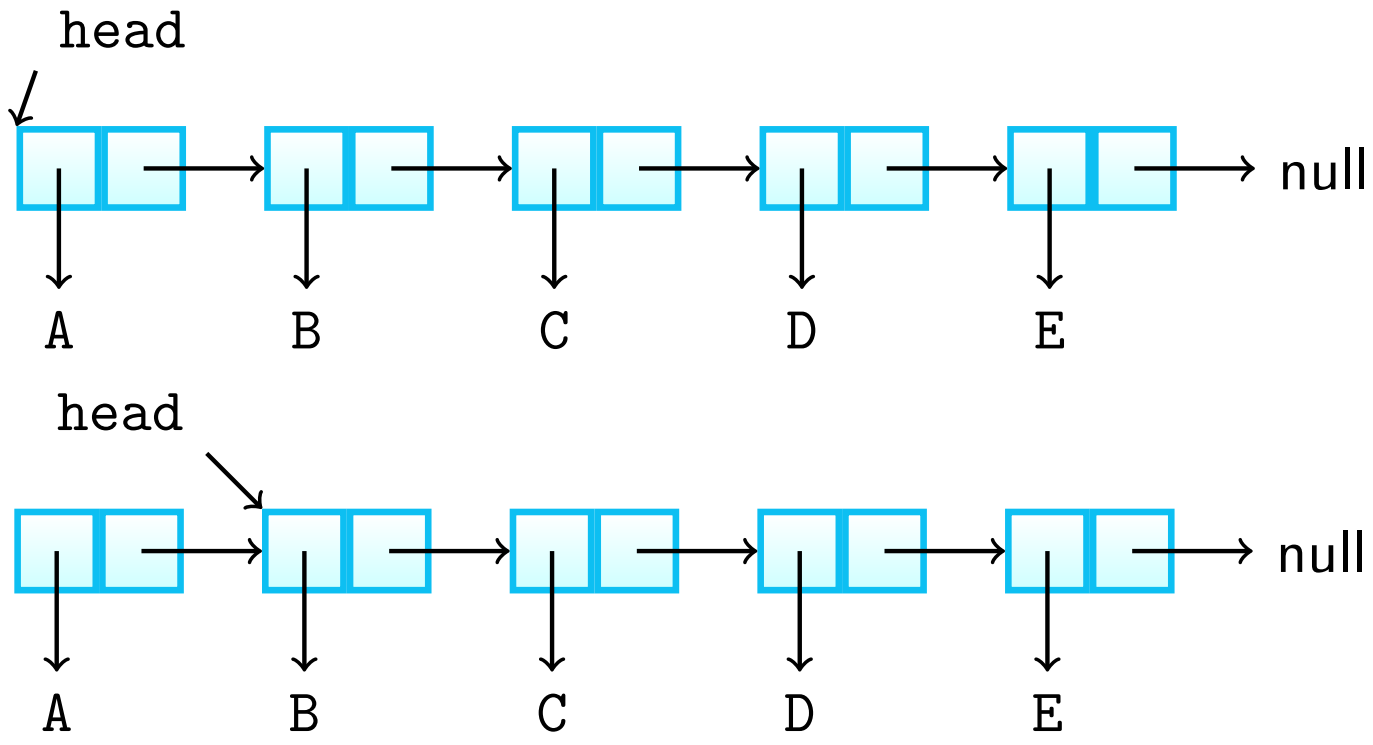
Removing at the Head

- Update head to point to the next node in the list.
- Allow garbage collector to reclaim the former first node.

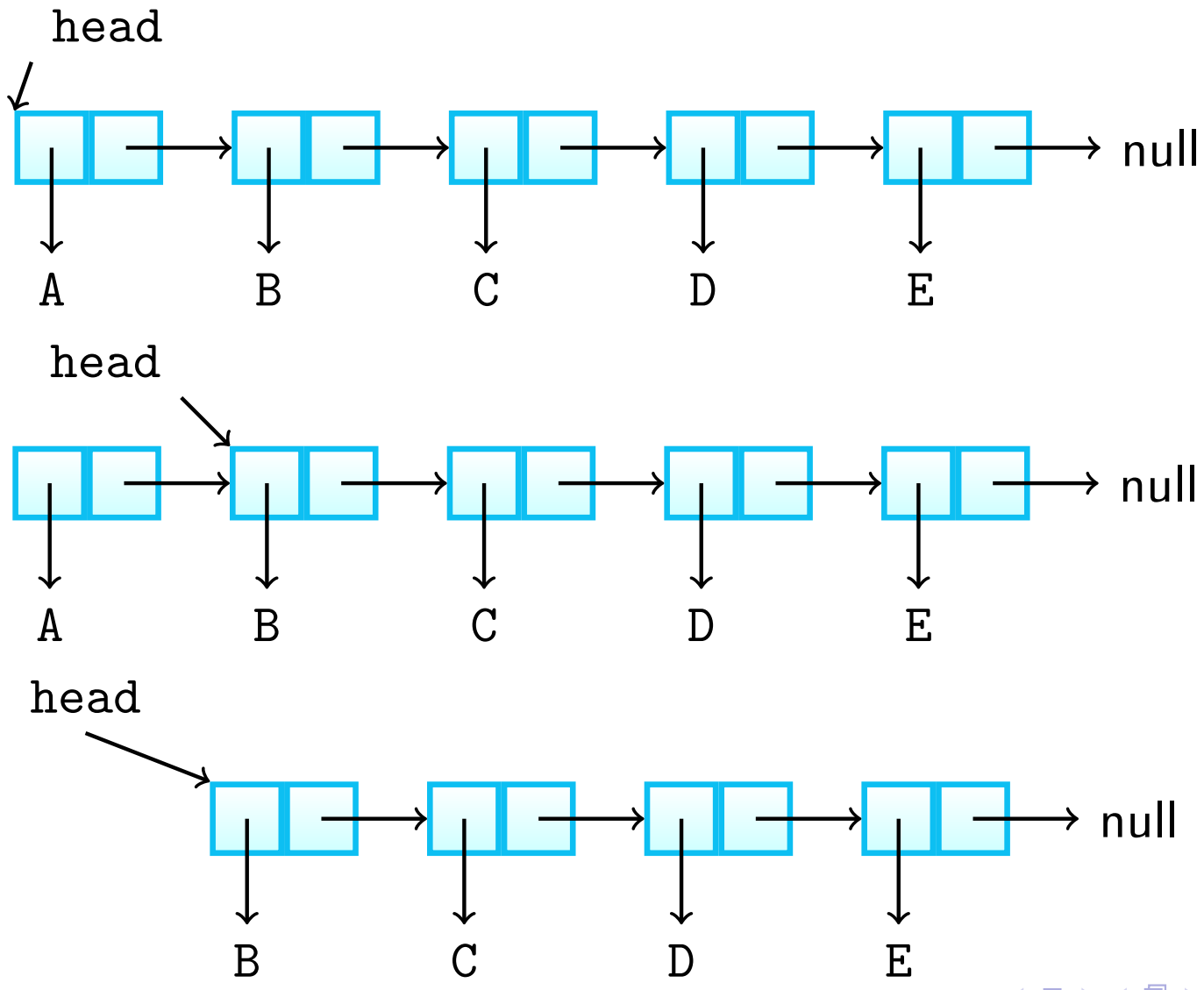
Removing at the Head



Removing at the Head



Removing at the Head



Removing at the Tail

- Removing at the tail of a singly linked list is NOT efficient! There is no constant-time way to update the tail to point to the previous node.

Removing at the Tail

- Removing at the tail of a singly linked list is NOT efficient! There is no constant-time way to update the tail to point to the previous node.
- Iterate to the node before the last node.

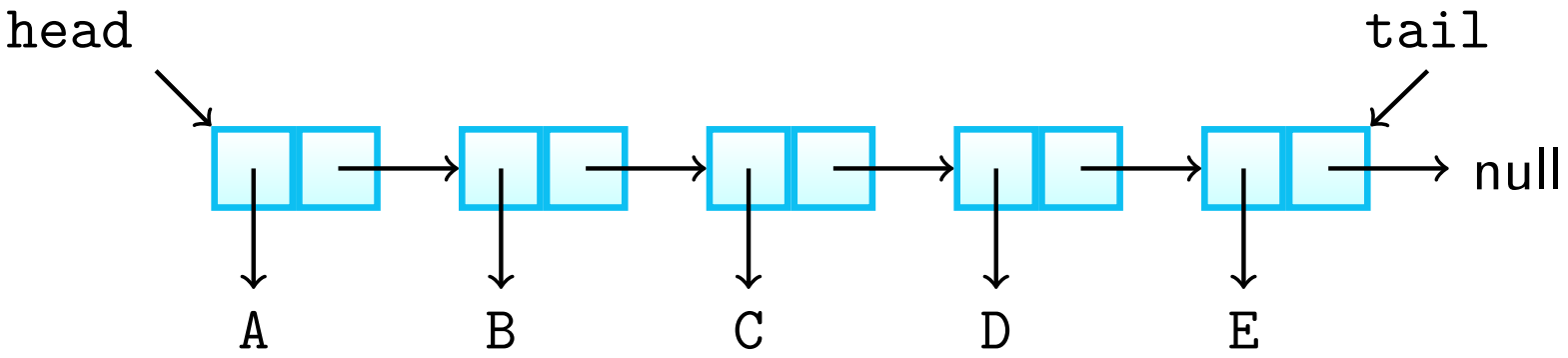
Removing at the Tail

- Removing at the tail of a singly linked list is NOT efficient! There is no constant-time way to update the tail to point to the previous node.
- Iterate to the node before the last node.
- Update tail (if there is a tail pointer).

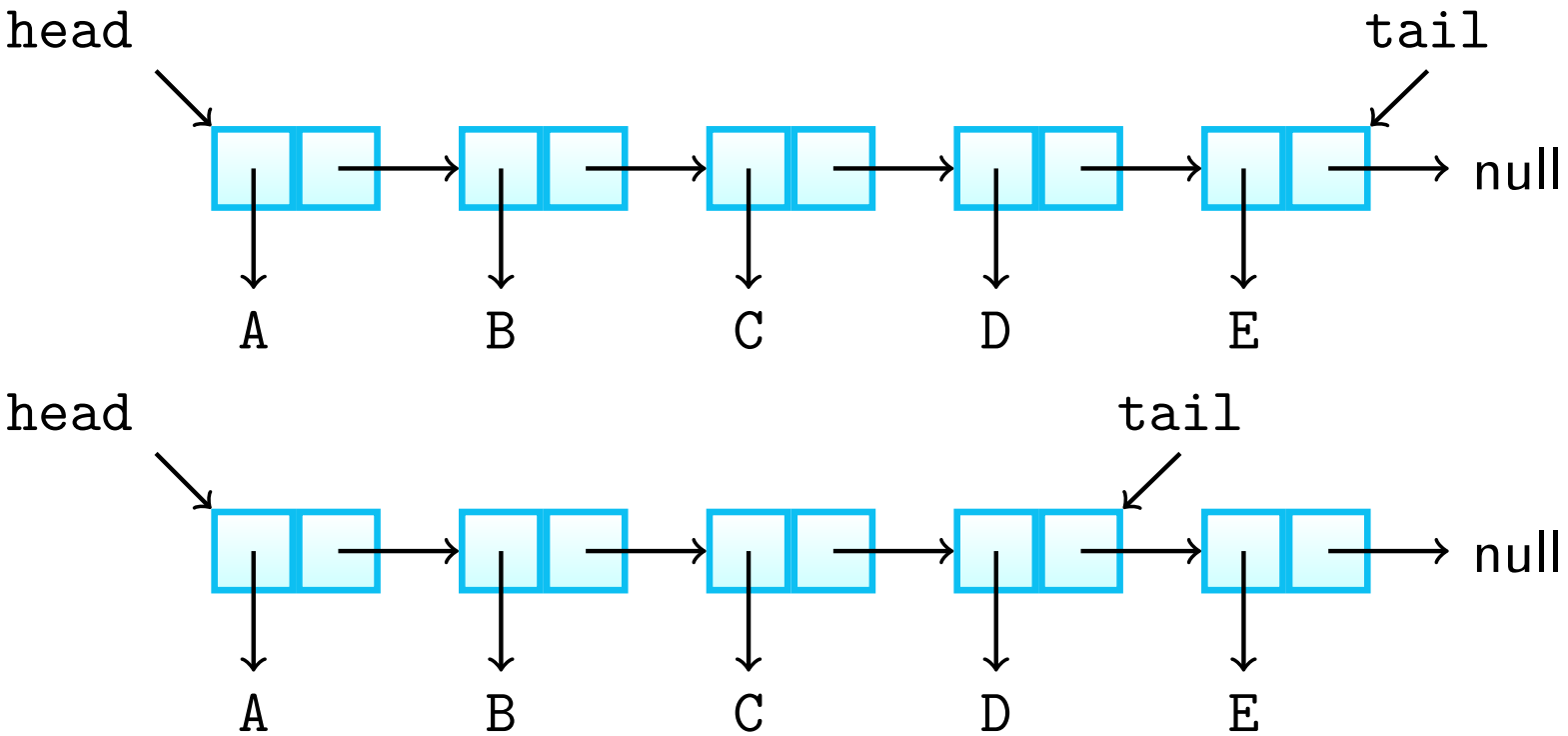
Removing at the Tail

- Removing at the tail of a singly linked list is NOT efficient! There is no constant-time way to update the tail to point to the previous node.
- Iterate to the node before the last node.
- Update tail (if there is a tail pointer).
- Set the current node's next reference to `null`

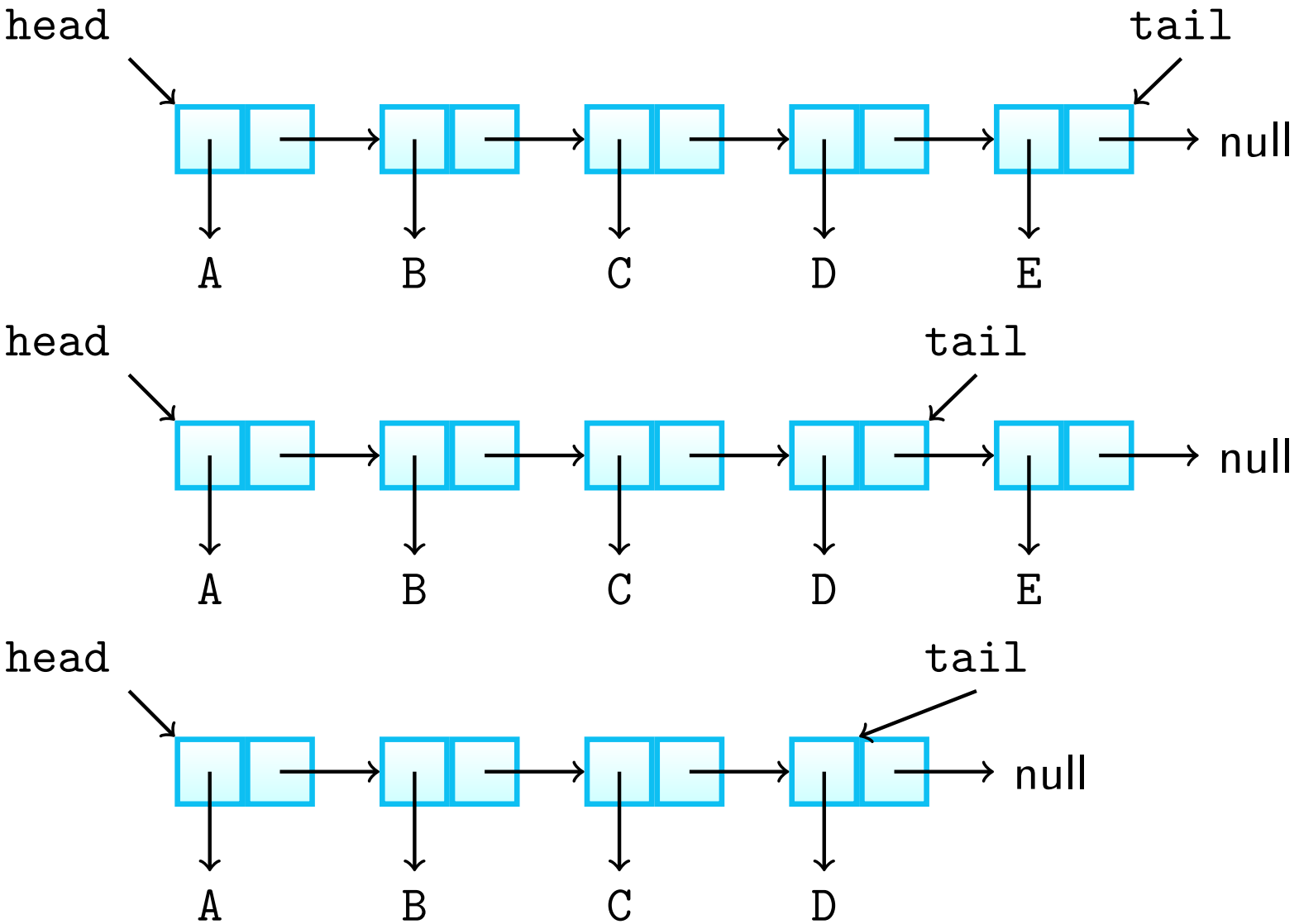
Removing at the Tail



Removing at the Tail



Removing at the Tail



Inserting/Removing at Arbitrary Locations

- In addition to inserting/removing at the head/tail, you can add items at any location in the linked list.

Inserting/Removing at Arbitrary Locations

- In addition to inserting/removing at the head/tail, you can add items at any location in the linked list.
- Note that you need to update the next pointer (and the previous pointer, if it is present) so that the new node is linked in correctly, and that the list can be traversed correctly.

Inserting/Removing at Arbitrary Locations

- In addition to inserting/removing at the head/tail, you can add items at any location in the linked list.
- Note that you need to update the next pointer (and the previous pointer, if it is present) so that the new node is linked in correctly, and that the list can be traversed correctly.
- Examples of these are shown for doubly-linked lists.

Pseudocode For Inserting/Removing

procedure AddFirst(e)
    node $\leftarrow$ Create a new `Node` object, and have its data be *e*, and the next node be *head*
    head $\leftarrow$ *node*
    if list is empty **then**
        tail $\leftarrow$ *node*
    end if
    size $\leftarrow$ *size* $+ 1$
end procedure

Pseudocode For Inserting/Removing

procedure AddLast(e)
    node ← Create a new `Node` object, and have its data be *e*
    if list is empty **then**
        head ← *node*
    else
        tail.*next* ← *node*
    end if
    tail ← *node*
    size ← *size* + 1
end procedure

Pseudocode For Inserting/Removing

procedure REMOVEFIRST(e)
    $node \leftarrow head$
    $head \leftarrow head.next$
    $size \leftarrow size - 1$
    return *node*
end procedure