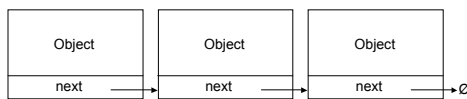


## Linked Lists

## Example

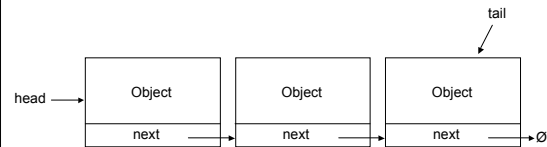
- We would like to keep a list of inventory records – but only as many as we need
- An array is a fixed size
- Instead – use a linked list
- What are the disadvantages of using a linked list?

## Linked List



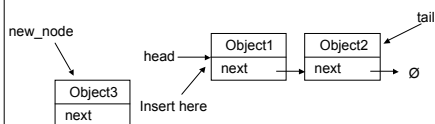
- **Node** – one element of the linked list
  - *Object* – data stored in the node – examples?
  - *next* – a reference to the next node in the list
    - last node points to NULL

## Linked List

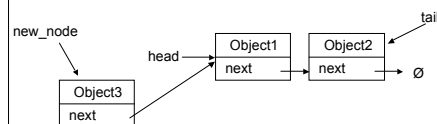


- *head* keeps track of the head of the list
- *tail* keeps track of the last node in the list
  - tail not always used

## Insertion at Head

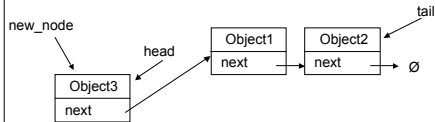


## Insertion at Head



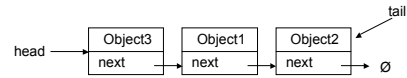
- Create *new\_node*
  - store object in *new\_node*
- Point *new\_node* next to the node head points to

## Insertion at Head

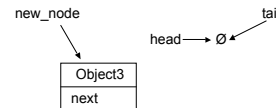


- Create new\_node
  - store object in new\_node
- Point new\_node next to the node head points to
- Point head to new\_node

## Insertion at Head



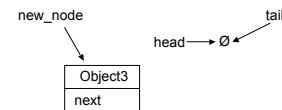
- Does this algorithm work for the list below?



## Insertion at Head

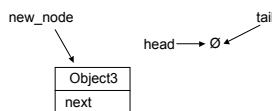
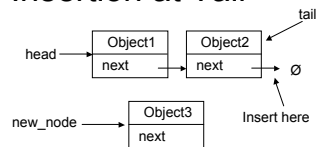
- Create new\_node
  - store object in new\_node
- Point new\_node next to the node head points to
- Point head to new\_node
- If tail points to NULL
  - point tail to new\_node

## Insertion at Head

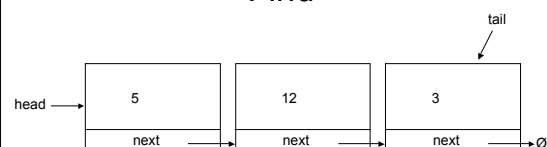


- Create new\_node
  - store object in new\_node
- Point new\_node next to the node head points to
- Point head to new\_node
- If tail points to NULL
  - point tail to new\_node

## Insertion at Tail

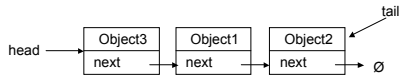


## Find



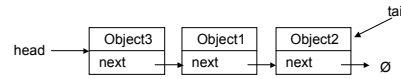
- find(3)
- find(16) - *always remember to deal with special cases*

## Deletion



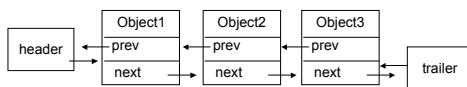
- Deletion of head
  - Complexity?
- Deletion of tail
  - Complexity?

## Insertion/Deletion in Middle



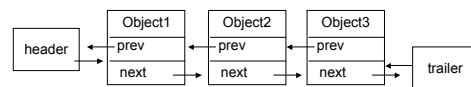
- Insert between Object1 and Object2
- Delete Object1

## Doubly Linked Lists

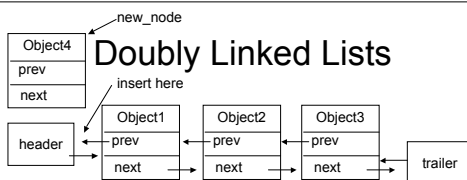
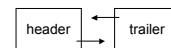


- Each node keeps a pointer to the next node and to the previous node
  - Makes some operations (such as insertion at end) more efficient
  - Costs?
- At the beginning and end of the list are *sentinel* nodes
  - Simplify insertion/deletion algorithm

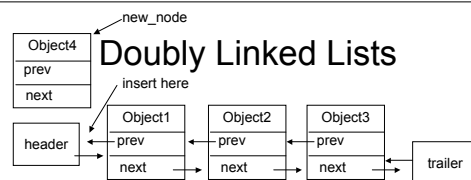
## Doubly Linked Lists



- Insertion and deletion at beginning/end
- Insertion and deletion in middle



- Insertion



- Insertion at head
  1. Set next of new\_node to point to what header's next points to
  2. Set prev of node that header's next points to to point to new\_node
  3. Set prev of new\_node to point to header
  4. Set header's next to point to new\_node
- Number 1 must come before number 4
- Insertion at trailer?
- Deletion?