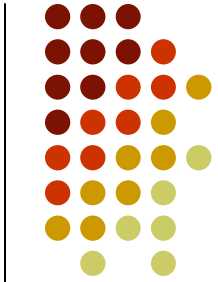
Comparison summary



- Array based (dynamic)
  - Keeps place for up to 4N elements
  - Each element takes 1 memory places
  - Fast accession time
  - Slow removals and insertion (due to need of copying data and resizing array)
- Reference based (LL)
  - Keeps place for exactly N elements
  - Each element takes 2 memory places
  - Slow accession time
  - Slow removals and insertion (due to accession)

Passing a Linked List to a Method

- A method with access to a linked list's head reference has access to the entire list
- When head is an actual argument to a method, its value is copied into the corresponding formal parameter

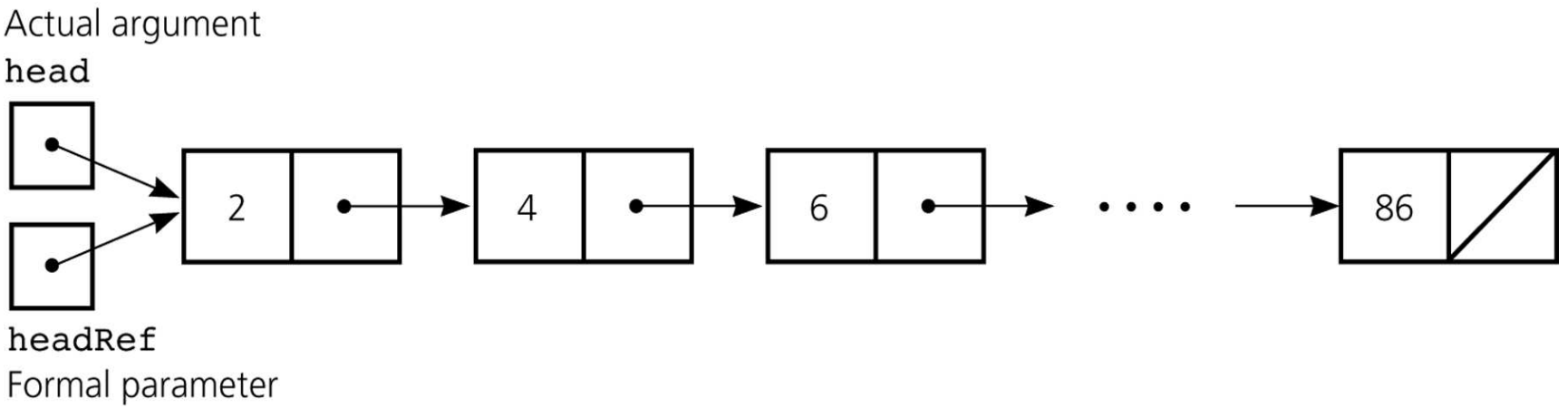
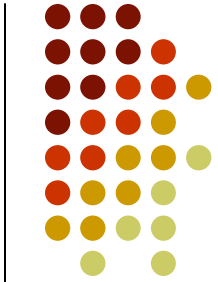


Figure 5-19

A head reference as an argument

Examples



- Merge() method, takes 2 sorted lists and returns one sorted list containing all elements (the original lists can be destroyed)
- Fast insertions and deletions at the beginning of the list using array-based implementation
- Fast insertion at the end and fast deletion at the beginning using linked-list (alternatively, array-based) implementation [tail reference]

Variations of the Linked List: Tail References

- `tail` references
  - Remembers where the end of the linked list is
  - To add a node to the end of a linked list

```
tail.setNext(new Node(request, null));
```

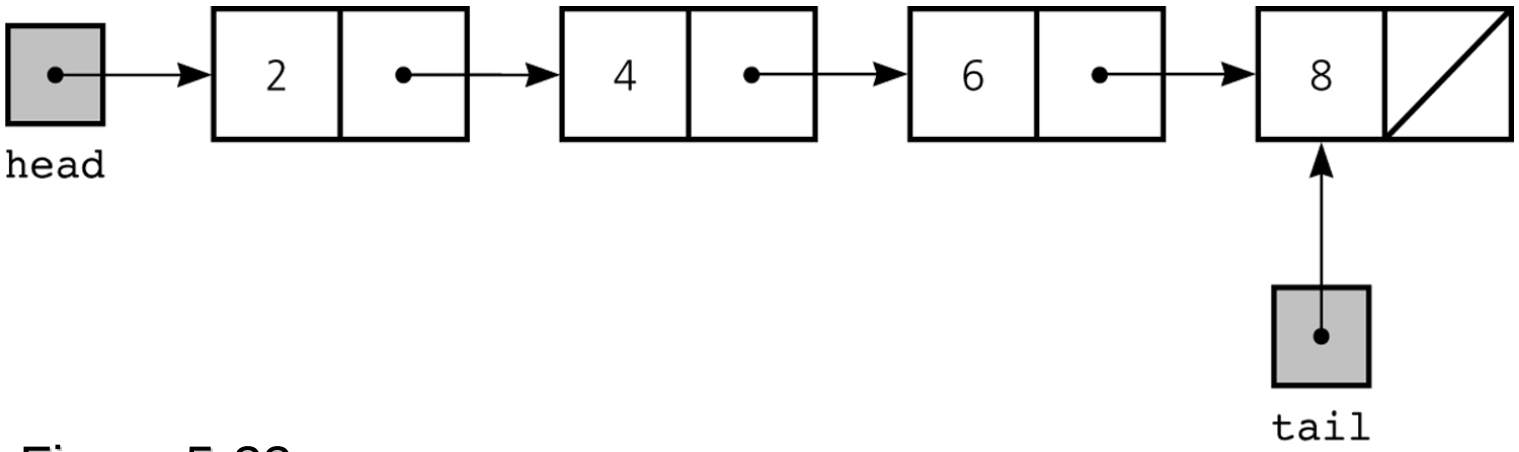


Figure 5-22

A linked list with ***head*** and ***tail*** references

Doubly Linked List

- Each node references both its predecessor and its successor
- Dummy head nodes are useful in doubly linked lists

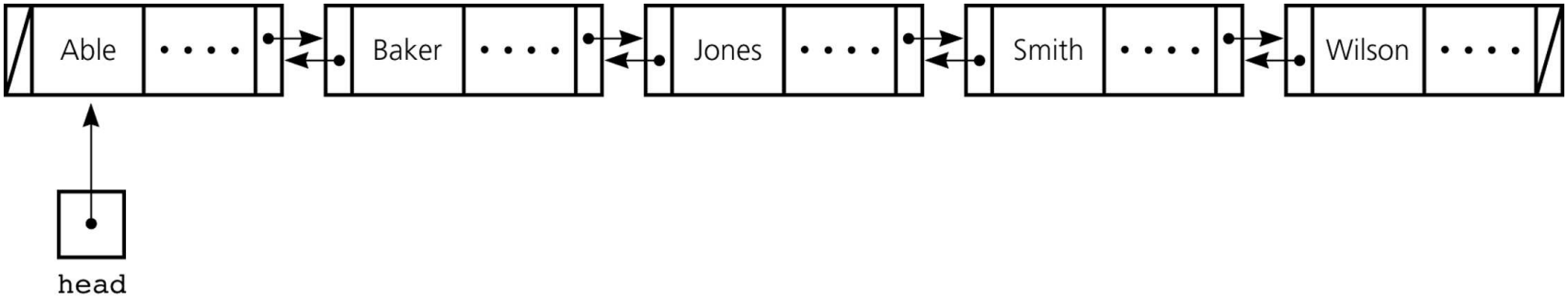


Figure 5-26

A doubly linked list

Doubly Linked List

- To delete the node that `curr` references

```
curr.getPrecede().setNext(curr.getNext());
curr.getNext().setPrecede(curr.getPrecede());
```

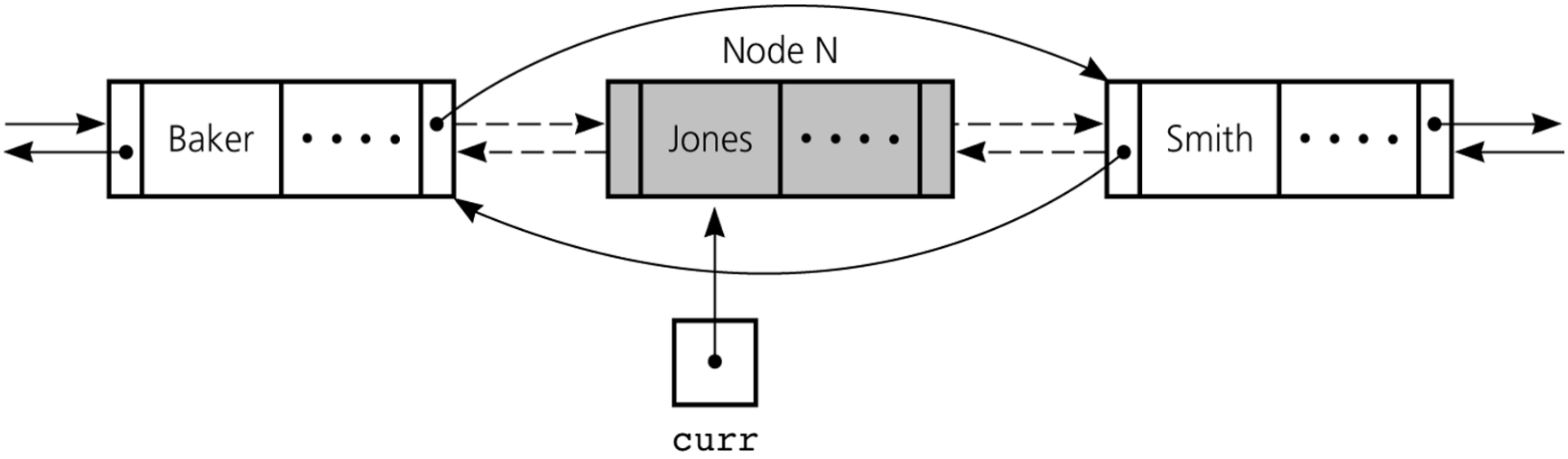


Figure 5-28

Reference changes for deletion

Doubly Linked List

- To insert a new node that newNode references before the node referenced by curr

```
newNode.setNext(curr);
newNode.setPrecede(curr.getPrecede());
curr.setPrecede(newNode);
newNode.getPrecede().setNext(newNode);
```

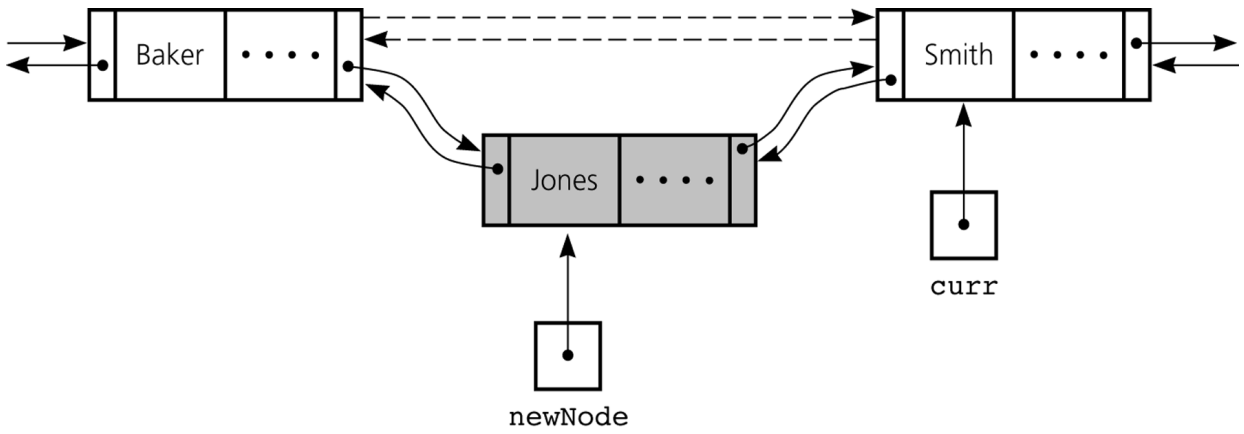


Figure 5-29

Reference changes for insertion

Eliminating special cases in Doubly Linked List

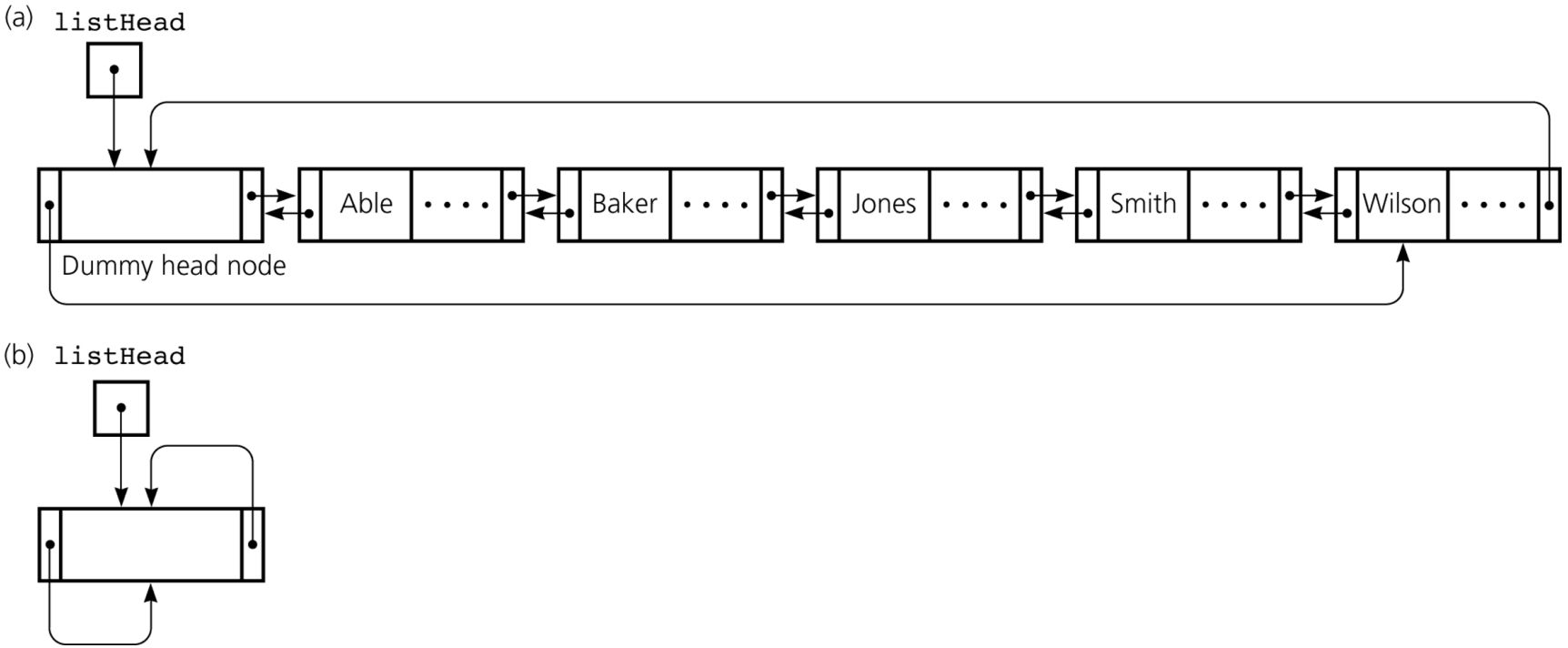
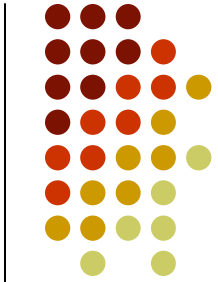


Figure 5-27

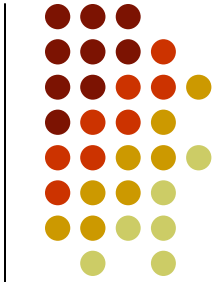
a) A circular doubly linked list with a dummy head node; b) an empty list with a dummy head node

Doubly Linked List



- Circular doubly linked list
  - `precede` reference of the dummy head node references the last node
  - `next` reference of the last node references the dummy head node
  - Eliminates special cases for insertions and deletions

Doubly Linked List - benefits



- Fast insertions and deletions at both ends of the list
- To perform deletion and insertion, we need reference to the current node (to node after place where we are inserting).

(Usual implementation of LinkedLists)