


ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

# Inserción y borrado con iteradores

Manuel Montenegro Montes  
Departamento de Sistemas Informáticos y Computación  
Facultad de Informática – Universidad Complutense de Madrid



# Operaciones a implementar

## Para listas

- Insertar un elemento en la posición apuntada por un iterador (***insert***)
- Eliminar el elemento apuntado por el iterador (***erase***)

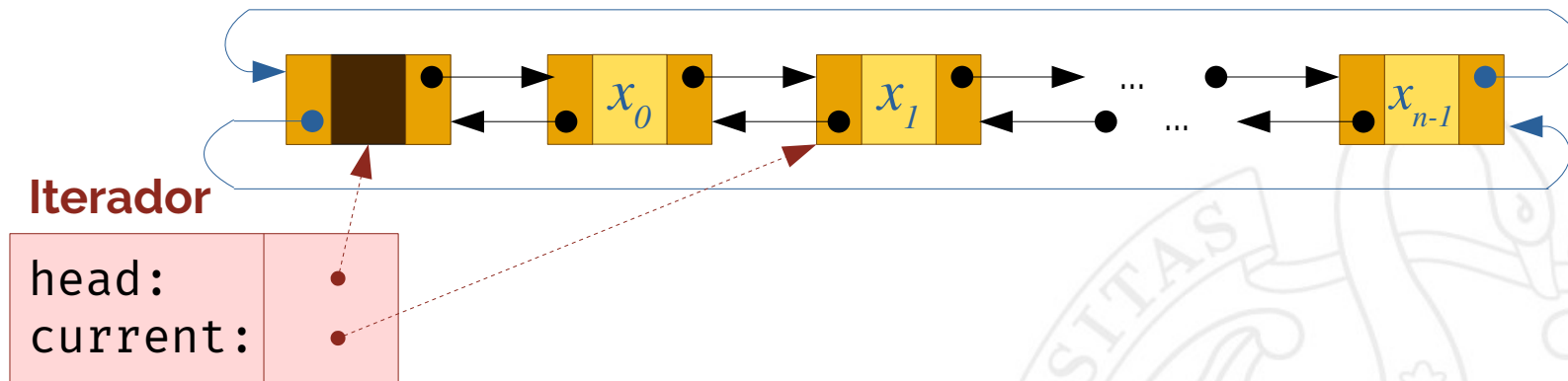


# Interfaz de ListLinkedList<T>

```
template <typename T>
class ListLinkedList {
public:
    ...
    iterator insert(iterator it, const T &elem);
    iterator erase(iterator it);
    ...
};
```

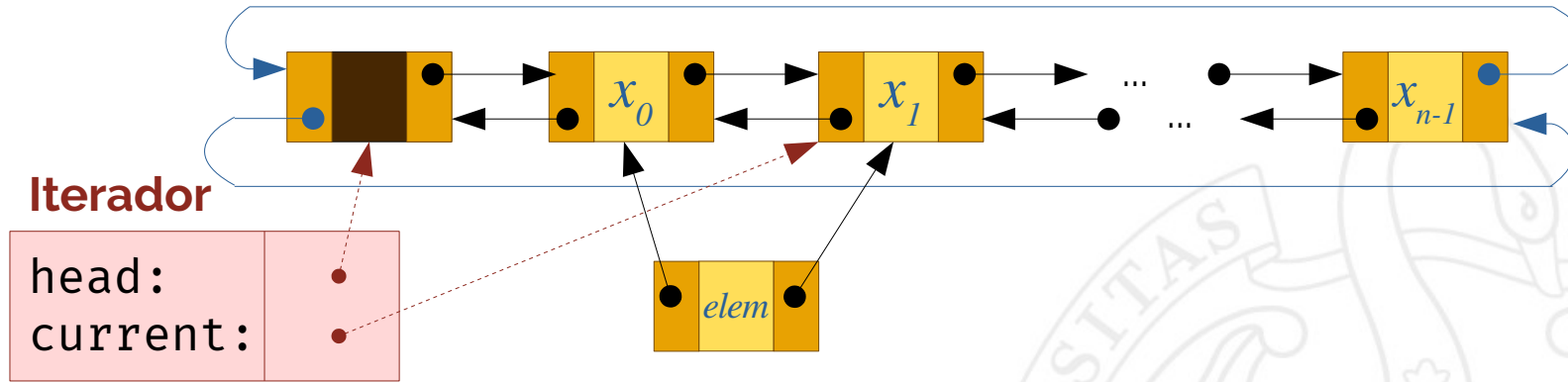


# Insertar un elemento



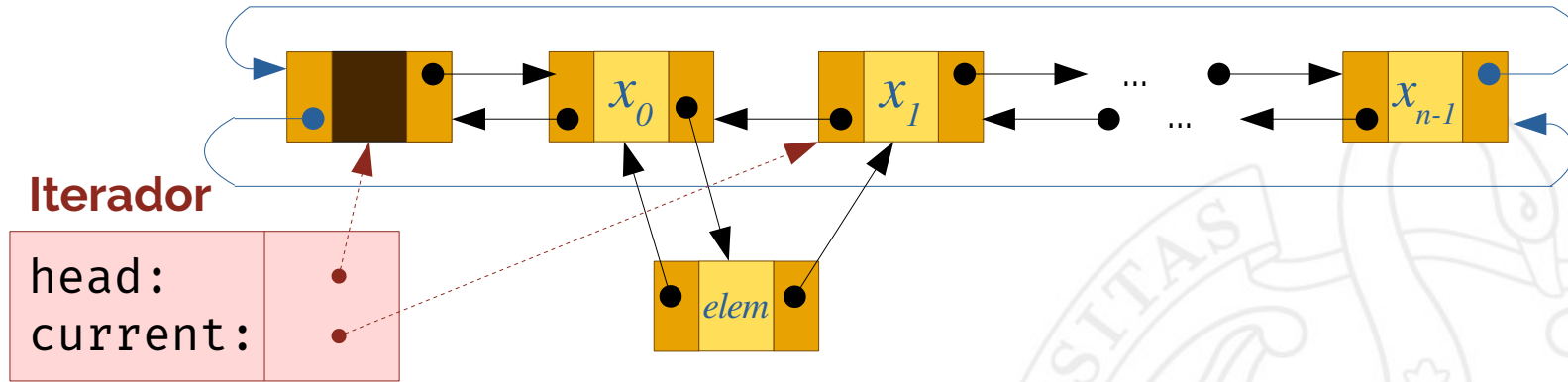
```
iterator insert(iterator it, const T &elem) {  
    assert(it.head == head);  
    Node *new_node = new Node { elem, it.current, it.current->prev };  
    it.current->prev->next = new_node;  
    it.current->prev = new_node;  
    num_elems++;  
    return iterator(head, new_node);  
}
```

# Insertar un elemento



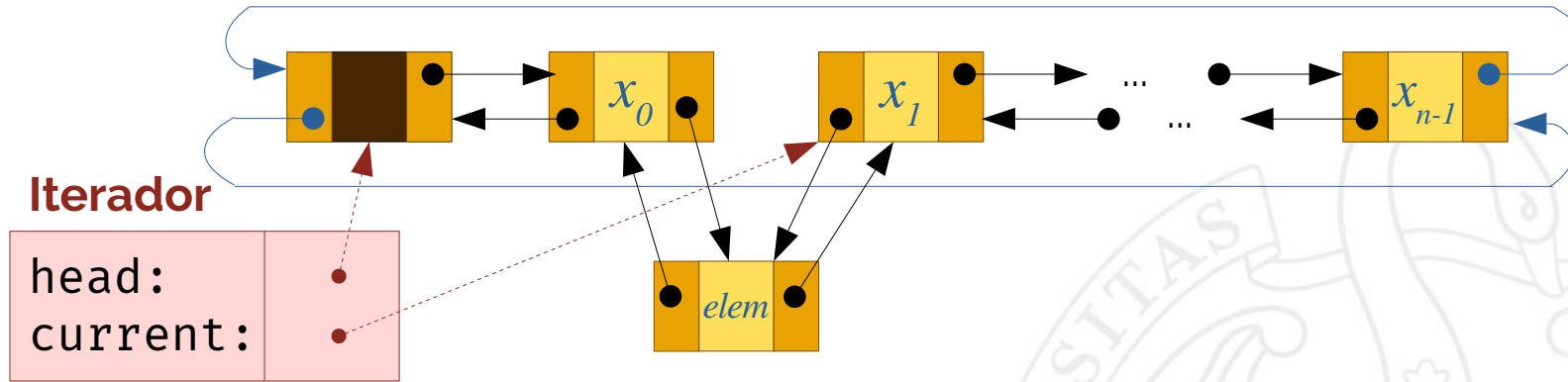
```
iterator insert(iterator it, const T &elem) {  
    assert(it.head == head);  
    Node *new_node = new Node { elem, it.current, it.current->prev };  
    it.current->prev->next = new_node;  
    it.current->prev = new_node;  
    num_elems++;  
    return iterator(head, new_node);  
}
```

# Insertar un elemento



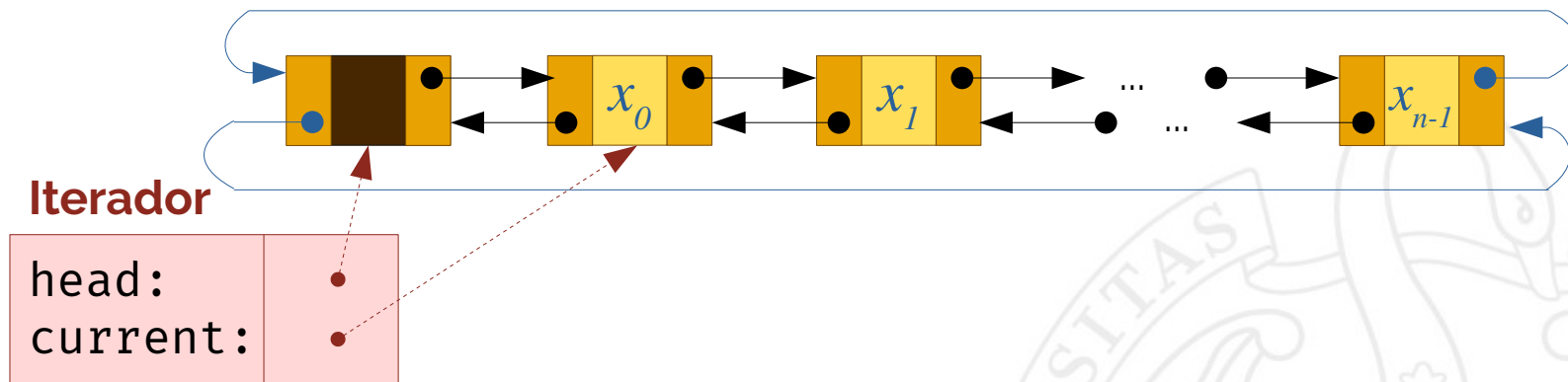
```
iterator insert(iterator it, const T &elem) {  
    assert(it.head == head);  
    Node *new_node = new Node { elem, it.current, it.current->prev };  
    it.current->prev->next = new_node;  
    it.current->prev = new_node;  
    num_elems++;  
    return iterator(head, new_node);  
}
```

# Insertar un elemento



```
iterator insert(iterator it, const T &elem) {  
    assert(it.head == head);  
    Node *new_node = new Node { elem, it.current, it.current->prev };  
    it.current->prev->next = new_node;  
    it.current->prev = new_node;  
    num_elems++;  
    return iterator(head, new_node);  
}
```

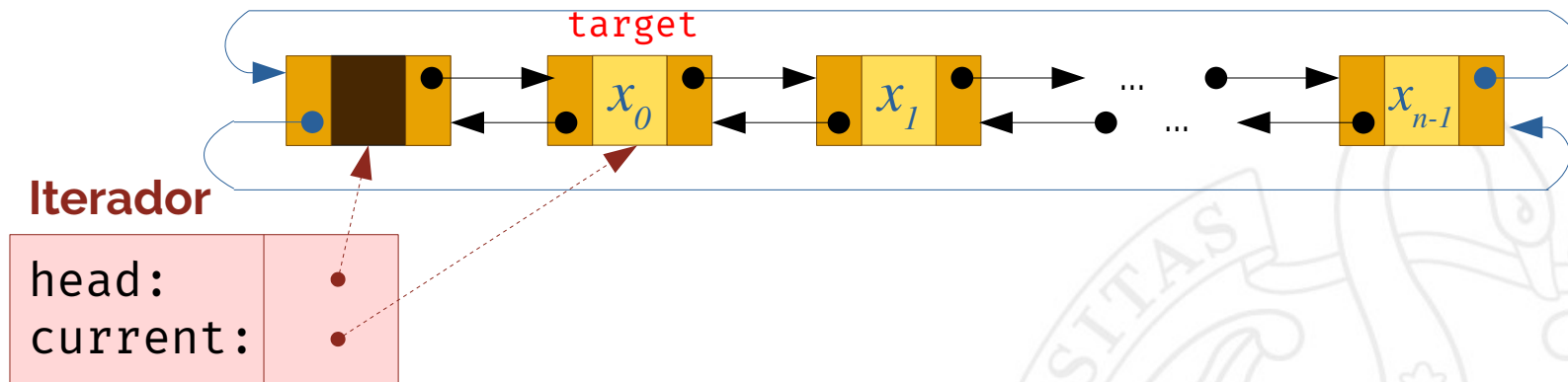
# Borrar un elemento



```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

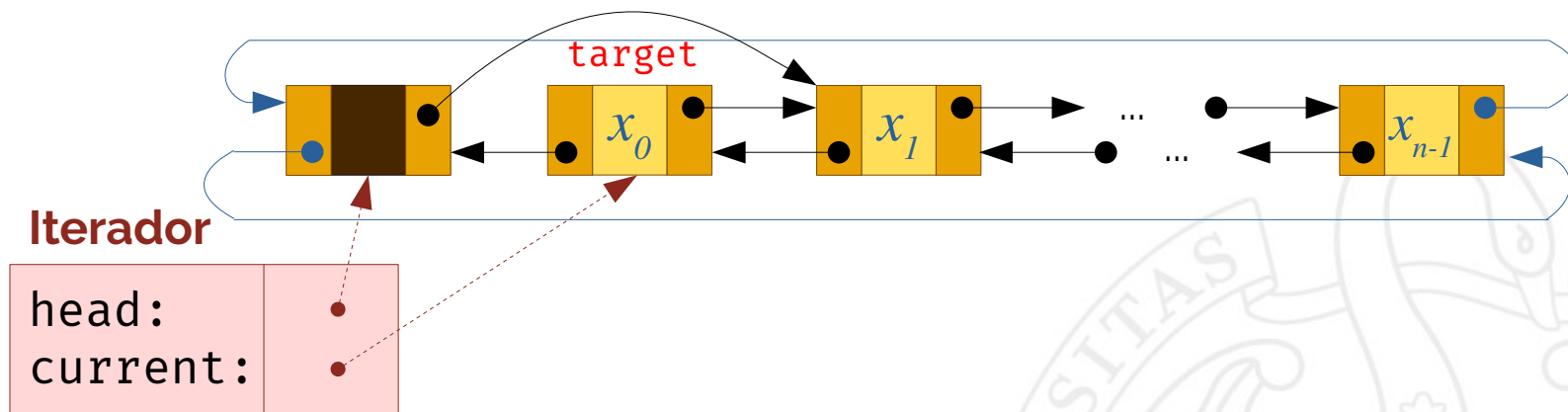


# Borrar un elemento



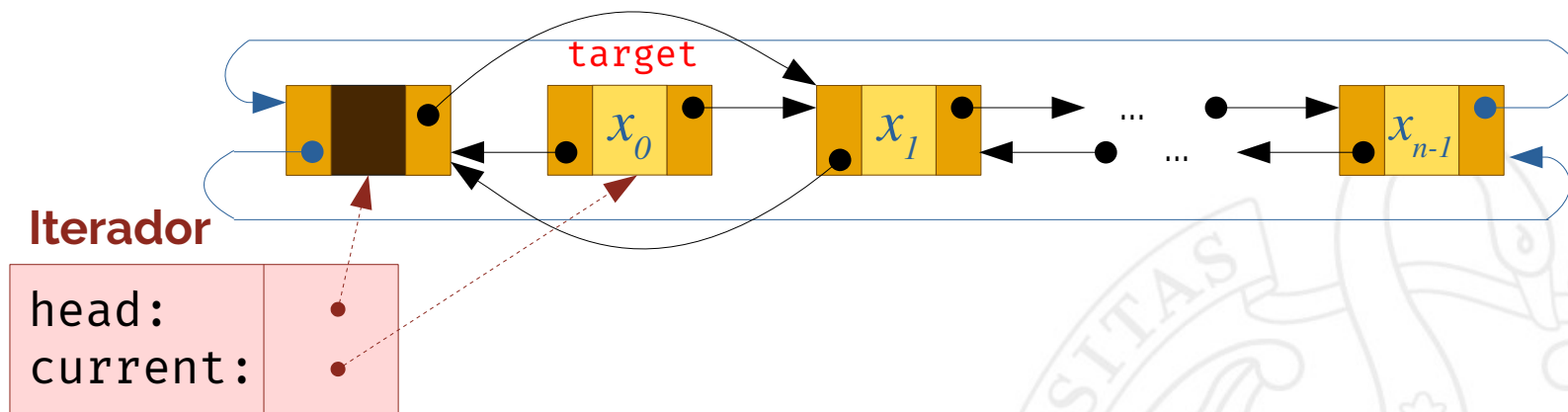
```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

# Borrar un elemento



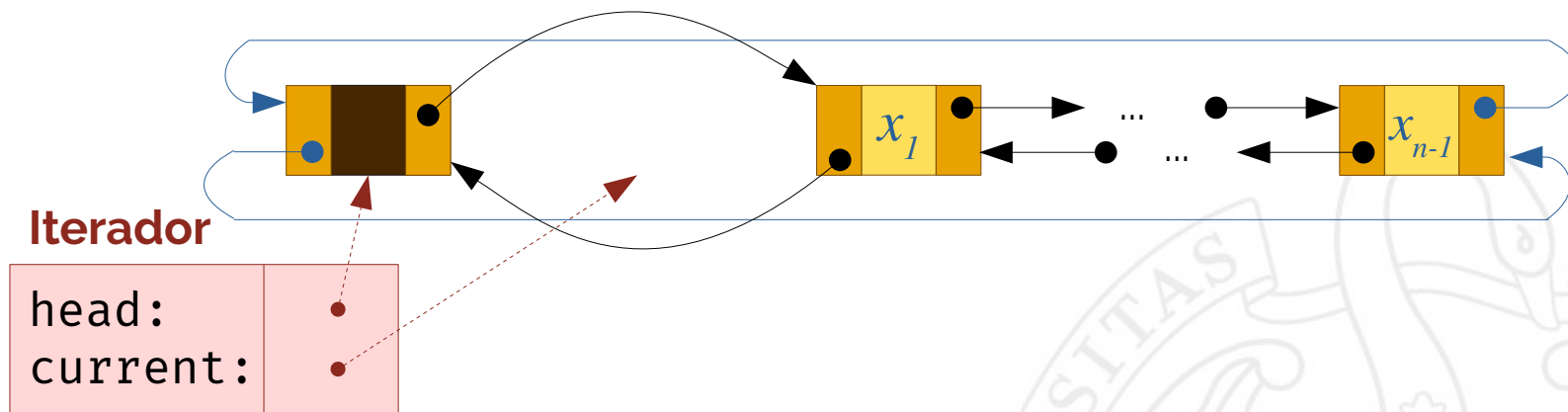
```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

# Borrar un elemento



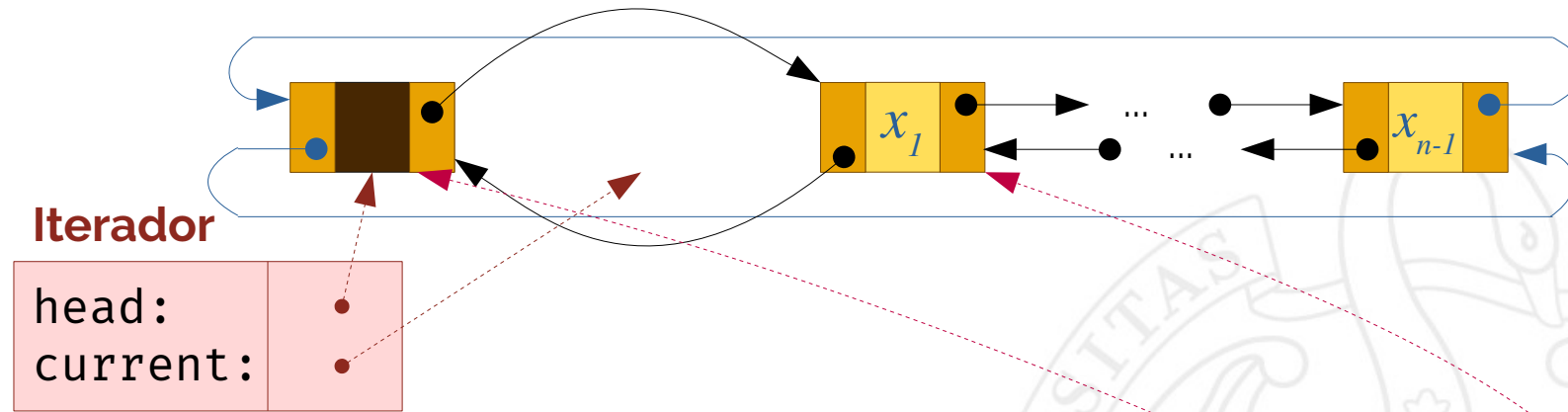
```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

# Borrar un elemento

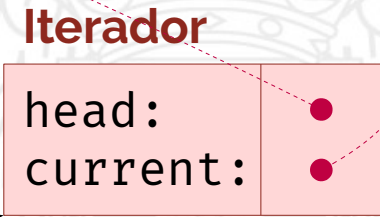


```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

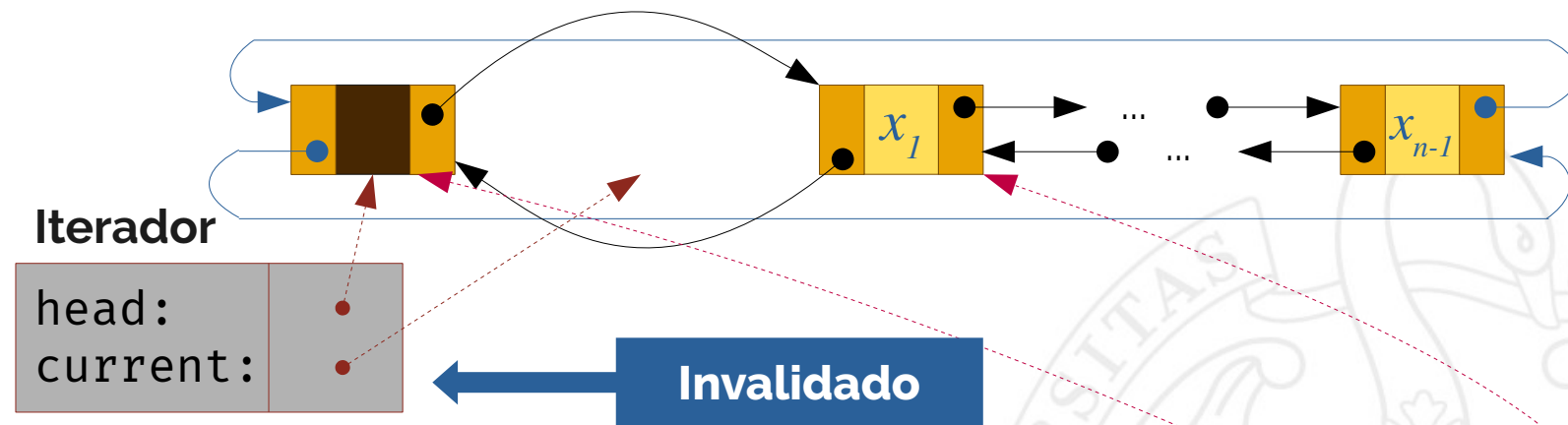
# Borrar un elemento



```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```



# Borrar un elemento



```
iterator erase(iterator it) {  
    assert(it.head == head && it.current != head);  
    Node *target = it.current;  
    target->prev->next = it.current->next;  
    target->next->prev = it.current->prev;  
    iterator result(head, target->next);  
    delete target;  
    num_elems--;  
    return result;  
}
```

**Iterator**

head:  
current: