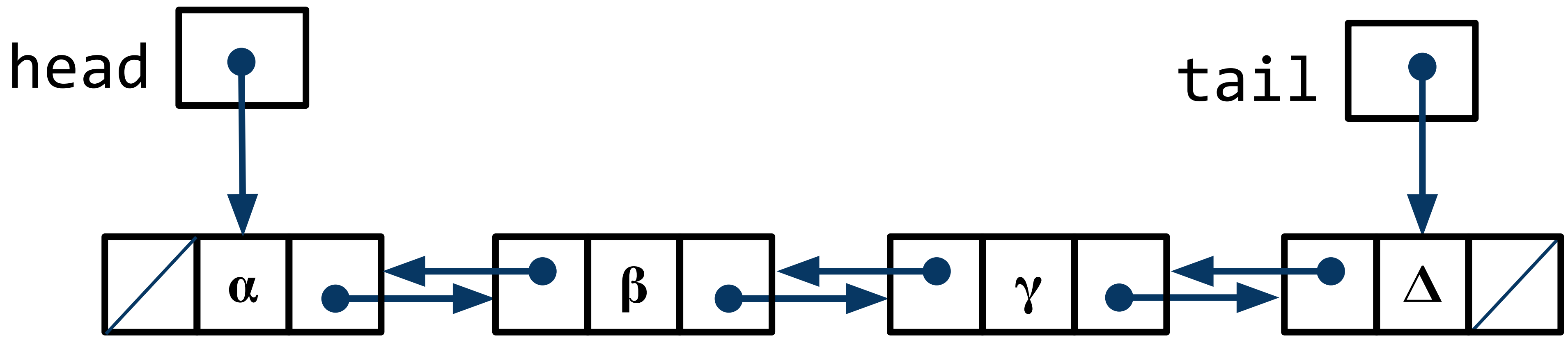
SYSC 2100

Algorithms and Data Structures

Lab 7: Doubly-Linked Lists

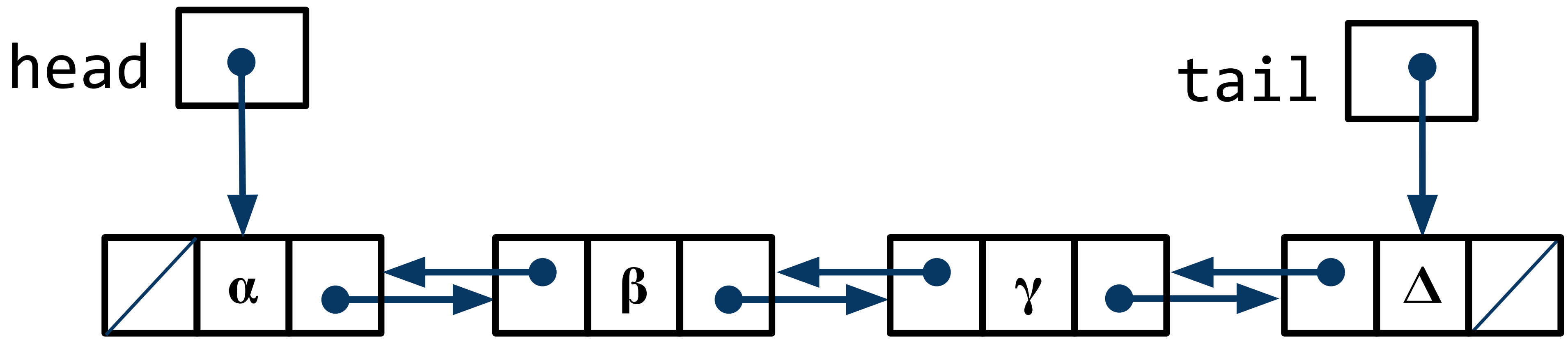
Edited: Mar. 4, 2023

Doubly-Linked Lists



- `head` stores the link to the head (first) node
- `tail` stores the link to the tail (last) node
- Every node *u* in a doubly-linked list stores two links (see next slide)

Doubly-Linked Lists



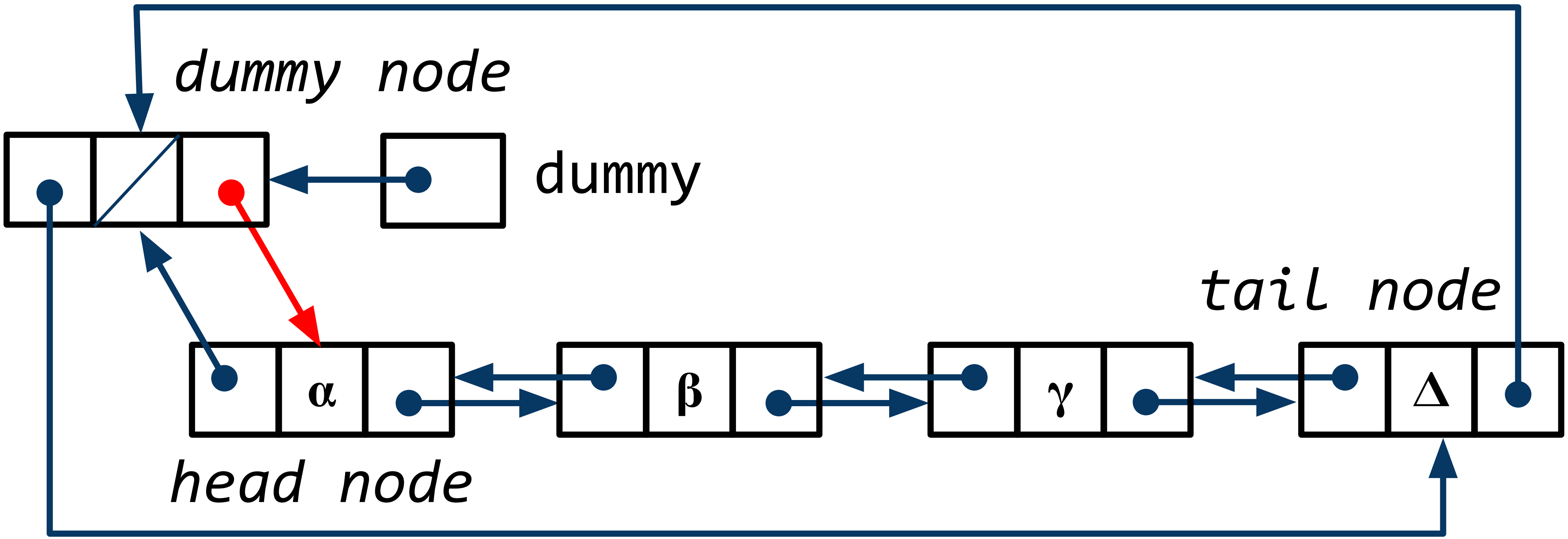
- *u.next* stores the link to the node that follows *u*
  - *u.next* in the tail node is the end-of-list marker
- *u.prev* stores the link to the node that precedes *u*
  - *u.prev* in the head node is the end-of-list marker

Carleton UNIVERSITY
Canada's Capital University

Special Cases

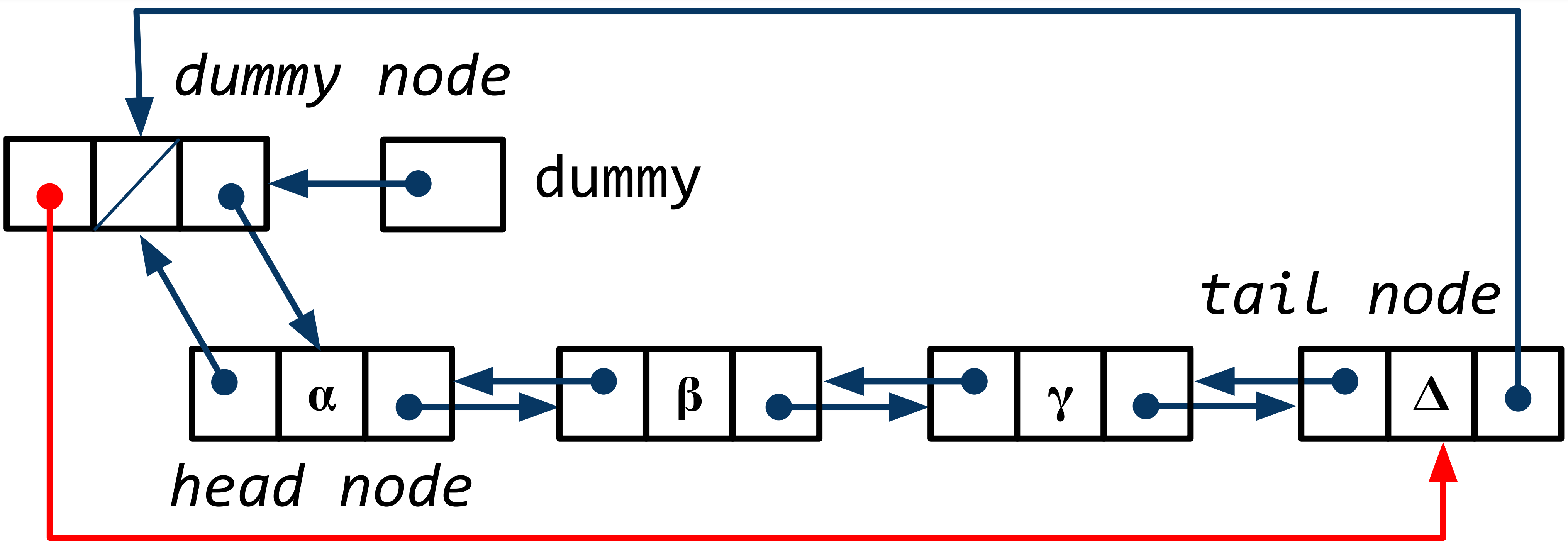
- Recall that, with a singly-linked list, making the first node a "dummy node" reduces the number of special cases that have to be handled separately by the list operations that insert and remove nodes
- For the same reason, we make the first node in a doubly-linked list a dummy node

Doubly-Linked List with Dummy Node



- *dummy.next* replaces the head variable; i.e., it points to the head node

Doubly-Linked List with Dummy Node

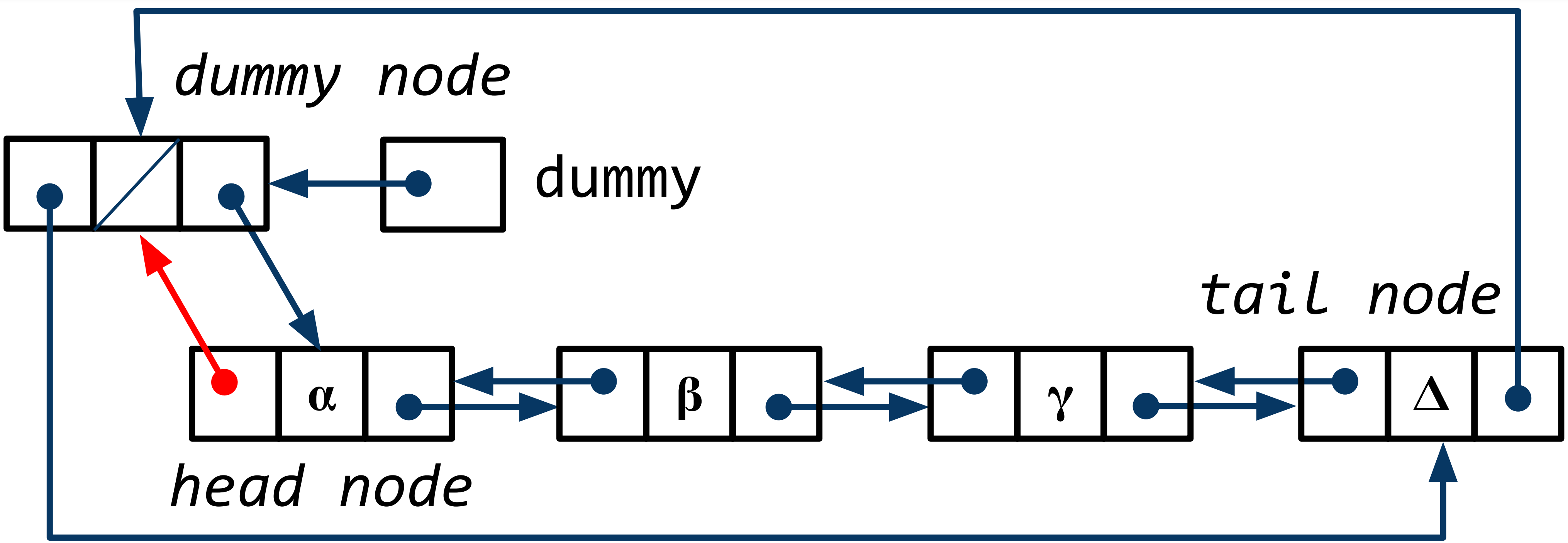


- *dummy.prev* replaces the `tail` variable; i.e., it points to the tail node

Doubly-Linked List with Dummy Node

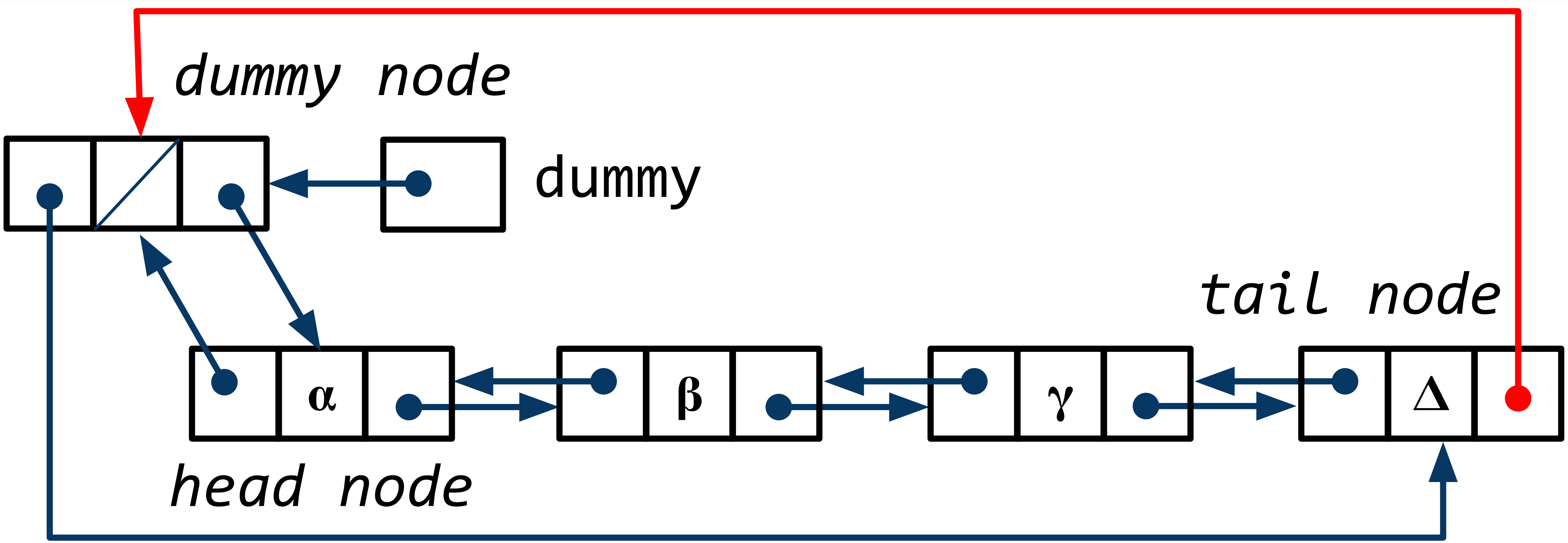
- The dummy node doesn't store any of the items that are stored in the linked list
- The dummy node ensures that every node has a node that precedes it and a node that follows it
    - End-of-list markers aren't required

Doubly-Linked List with Dummy Node



- The *prev* link in the head node points to the dummy node
- So, *dummy.next.prev* points to the dummy node

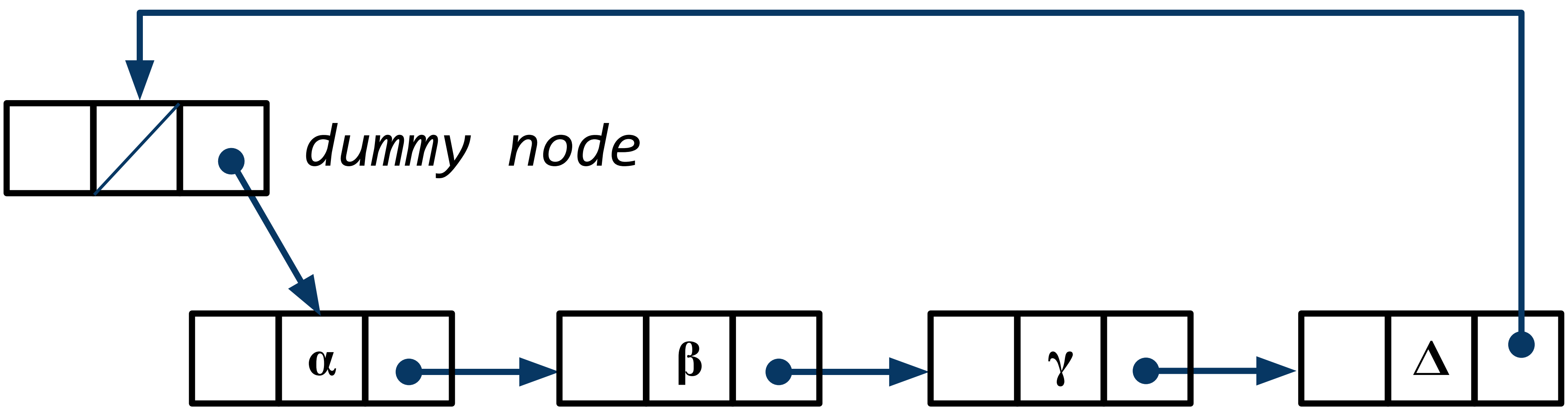
Doubly-Linked List with Dummy Node



- The *next* link in the tail node points to the dummy node
- So, *dummy.prev.nex*t points to the dummy node

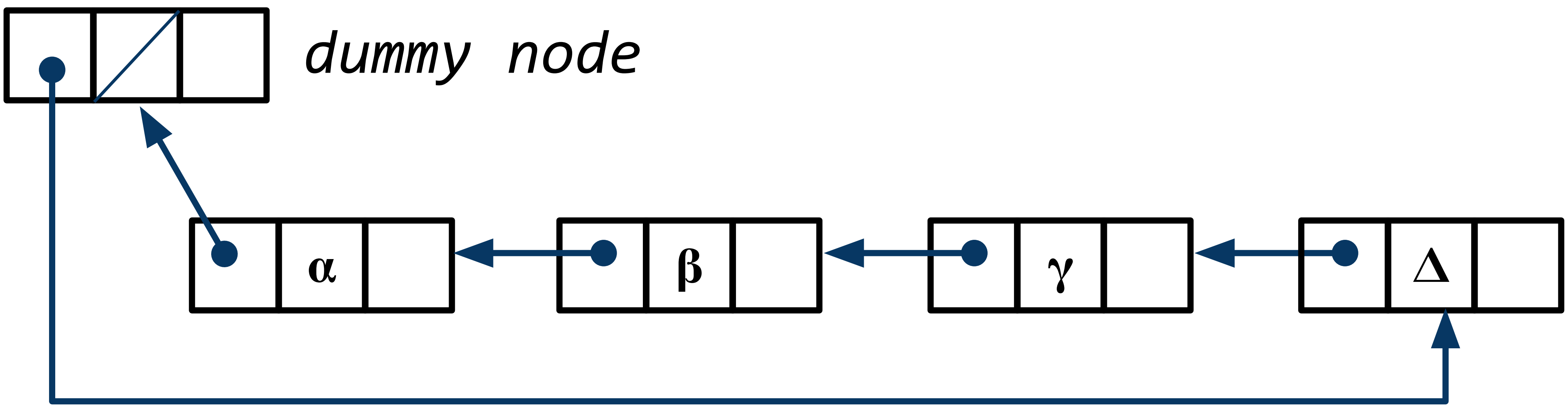
Linked List Cycles

- The linked list is circular: the *next* links form a front-to-back cycle through the nodes



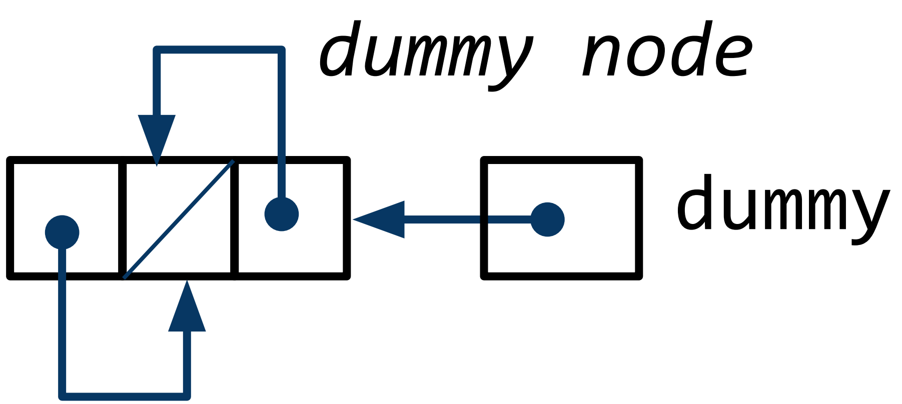
Linked List Cycles

- The *prev* links form another cycle (back to front) through the nodes



Empty Linked List

- An empty doubly-linked list has one node: the dummy node
- *dummy.next* and *dummy.prev* point to the the dummy node



Initialization Algorithm

- Initialize a new, empty doubly-linked list

new_node() creates a new node containing nil as the payload

$$
\begin{aligned}
&dummy \leftarrow \mathtt{new\_node}(nil)\\
&dummy.prev \leftarrow dummy\\
&dummy.next \leftarrow dummy\\
&num\_items \leftarrow 0
\end{aligned}
$$

Initialization: Python

- Class `LinkedDeque` uses a doubly-linked list as the underlying data structure

```
class LinkedDeque:
    class _Node(self, item: any) -> None:
        self.item = item
        self.prev = None
        self.next = None

    def __init__(self):
        self._dummy = LinkedDeque._Node(None)
        self._dummy.prev = self._dummy
        self._dummy.next = self._dummy
        self._num_items = 0
```

Carleton UNIVERSITY
Canada's Capital University

insert_before(): Algorithm

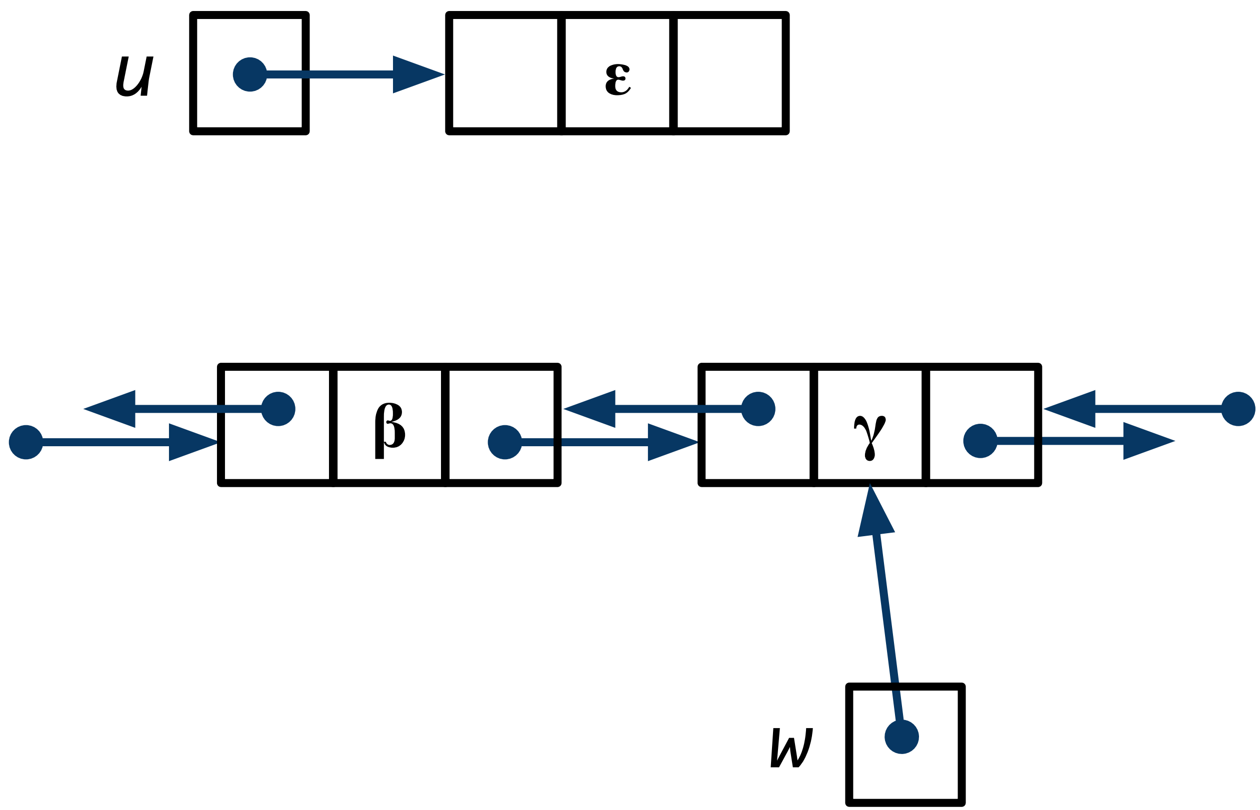
- insert_before(w, x) inserts a new node, u, containing x, before the node pointed to by w

```
insert_before(w, x)
    u ← new_node(x)
    u.prev ← w.prev
    u.next ← w
    u.next.prev ← u
    u.prev.next ← u
    num_items ← num_items + 1
```

Complexity of insert_before() is $O(1)$

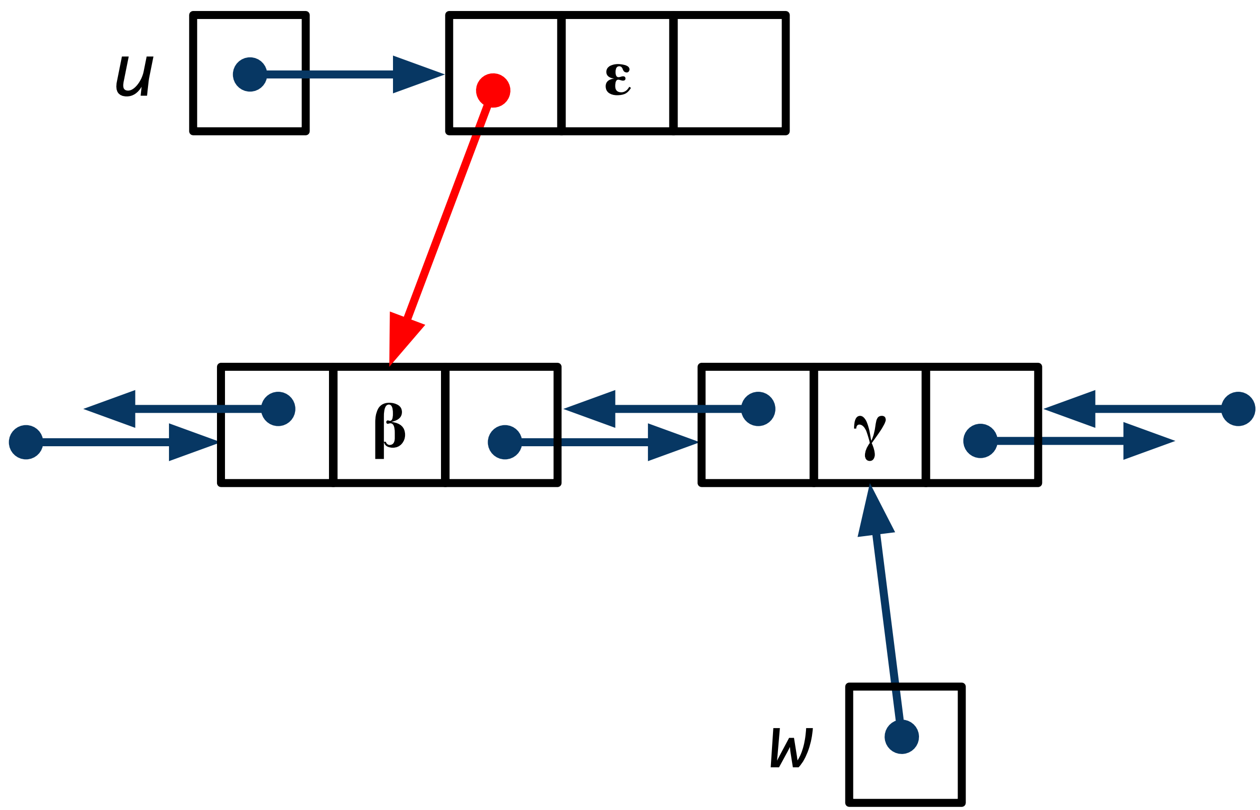
insert_before(), Step-by-Step

- After u ← new_node(ε)



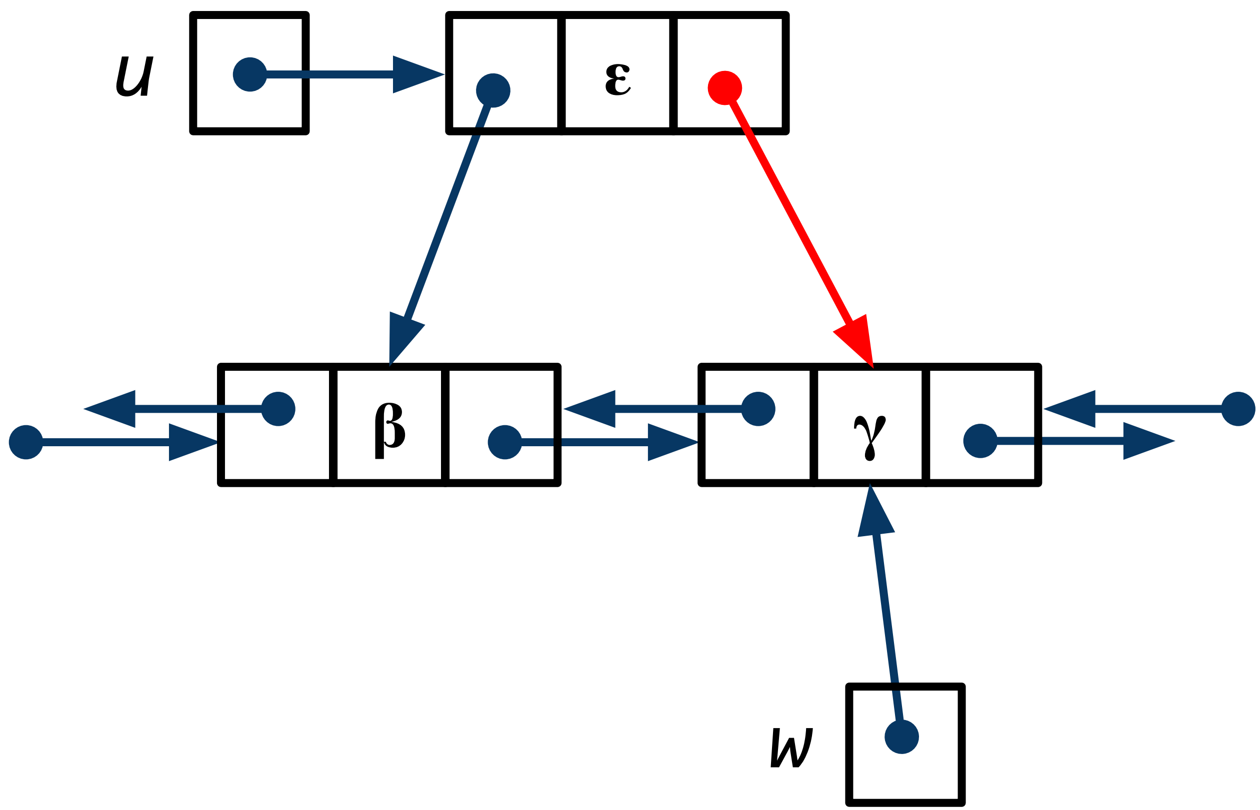
insert_before(), Step-by-Step

- After $u.prev \leftarrow w.prev$



insert_before(), Step-by-Step

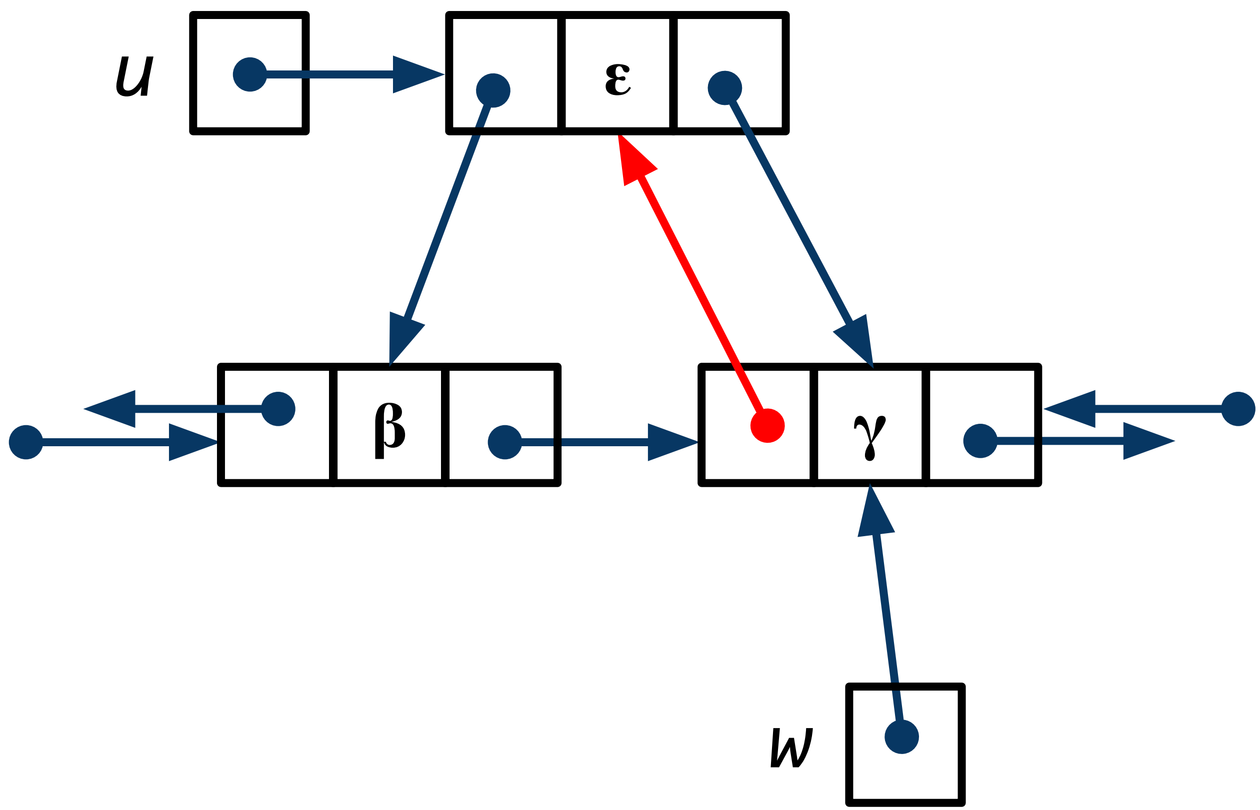
- After $u.next \leftarrow w$



Carleton UNIVERSITY
Canada's Capital University

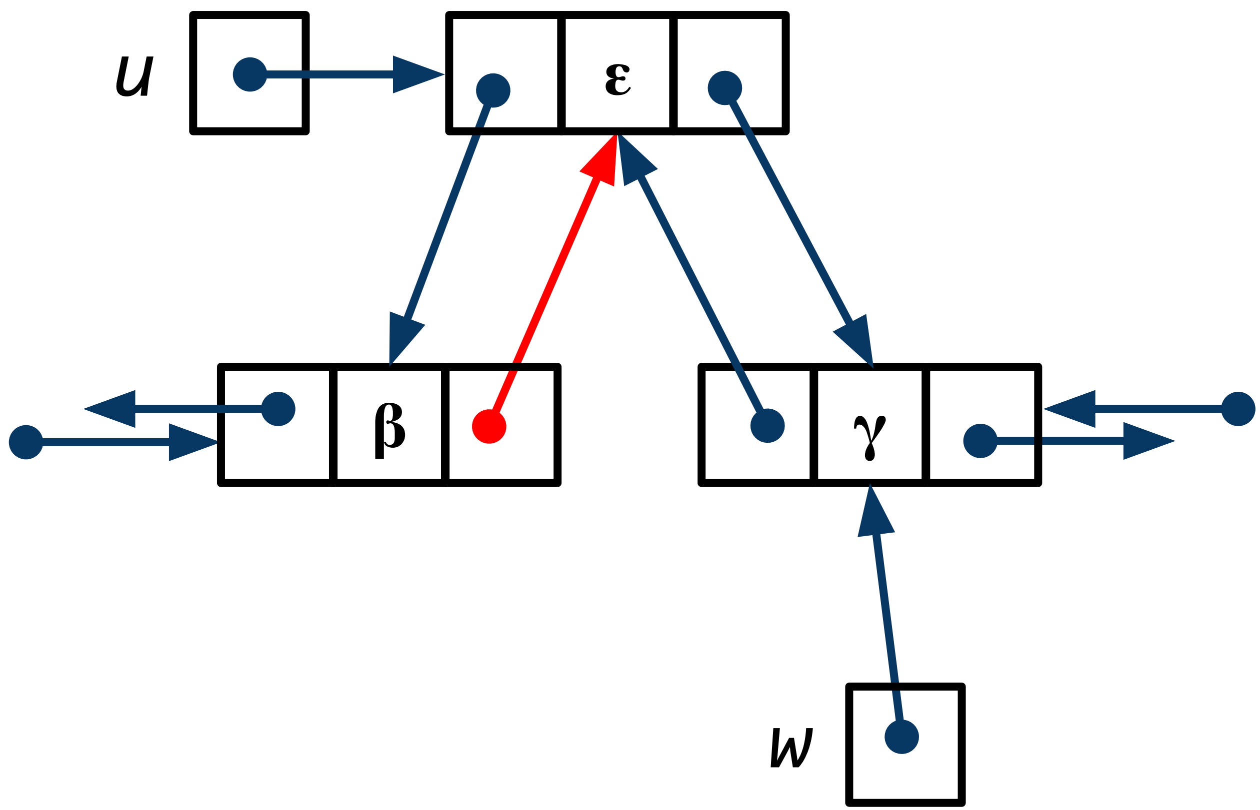
insert_before(), Step-by-Step

- After $u.next.prev \leftarrow u$



insert_before(), Step-by-Step

- After $u.prev.next \leftarrow u$



remove(): Algorithm

- remove(*w*), unlinks the node the pointed to by *w*
  - Updates links so that the node before *w* points to the node after *w*, and the node after *w* points to the node before *w*

```
remove(w)
    w.prev.next ← w.next
    w.next.prev ← w.prev
    num_items ← num_items - 1
```