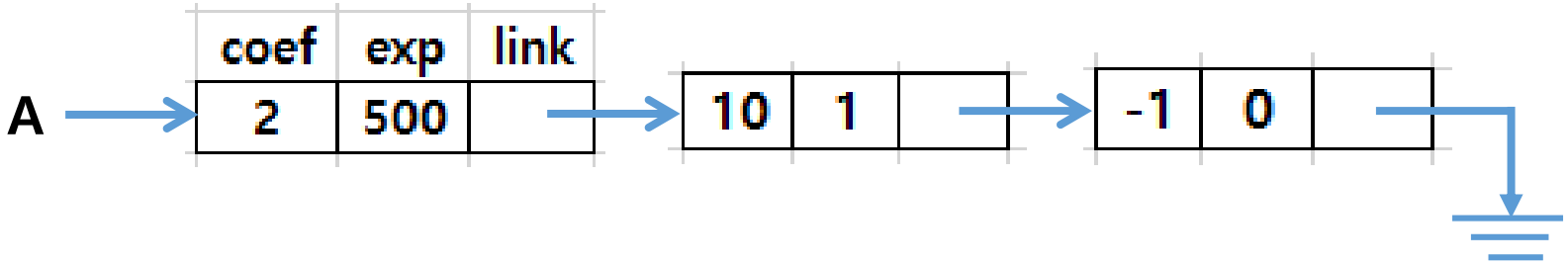
Chap 4. Linked Lists

Linked list



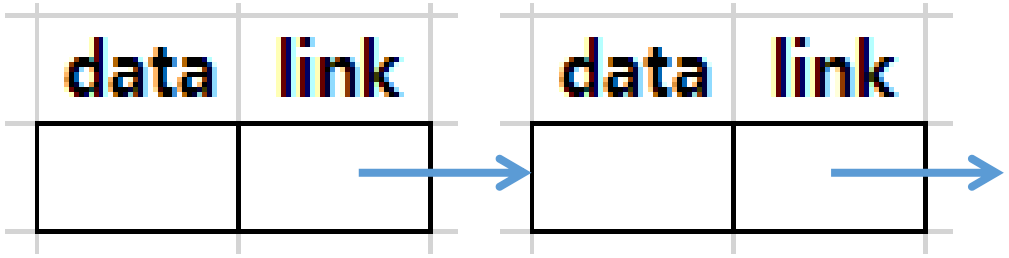
- Alternative to array
- Example
  - Polynomial $A(x) = 2x^{500} + 10x - 1$
- Node: consists of following fields
  - Data
  - Link (to next node)
- Array
  - Static allocation of memory
  - Sequential representation of data in A[0] … A[n-1]
- Linked list
  - dynamic allocation of memory (nodes)
  - Linked representation of data in the nodes of the list

Node

- Type declaration in C

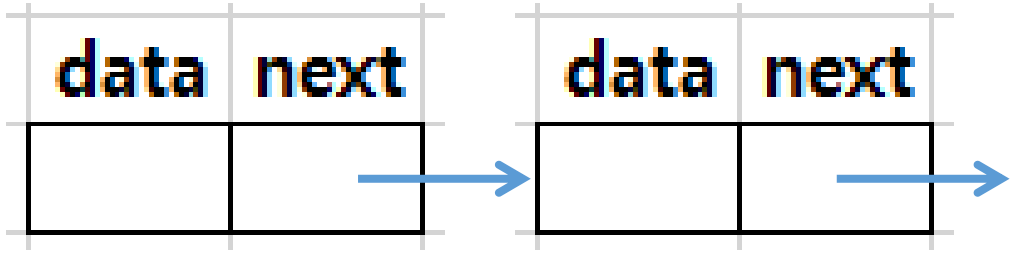
```
typedef struct listnode *nodePtr;
typedef struct listnode {
    //declaration of data fields
    nodePtr link; //declaration of link field
};
```

- *Self-referential structure*

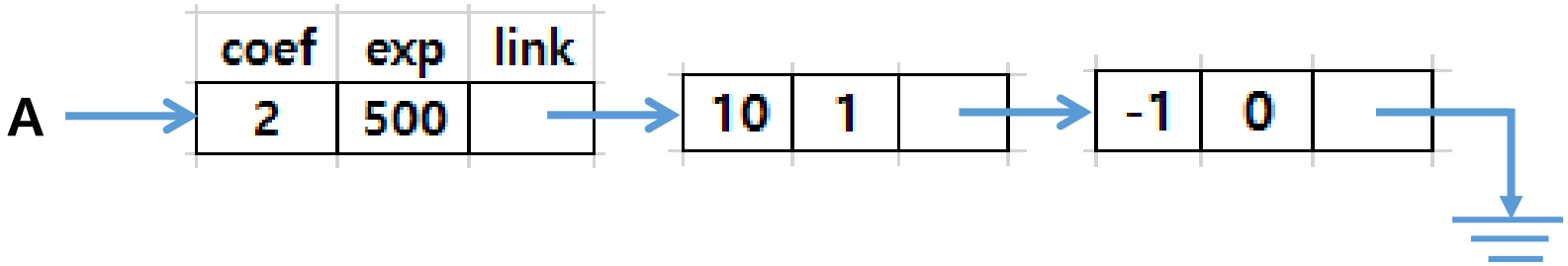


- Other naming of types

```
typedef struct node *listPtr;
typedef struct node {
    //declaration of data fields
    listPtr next; //declaration of link field
};
```



Example



- Polynomial: each term is represented in a node
- Declaration 1

```
typedef struct term *ptr2term;
typedef struct term {
    int coef;
    int exp;
    ptr2term link;
};
ptr2term A;
```

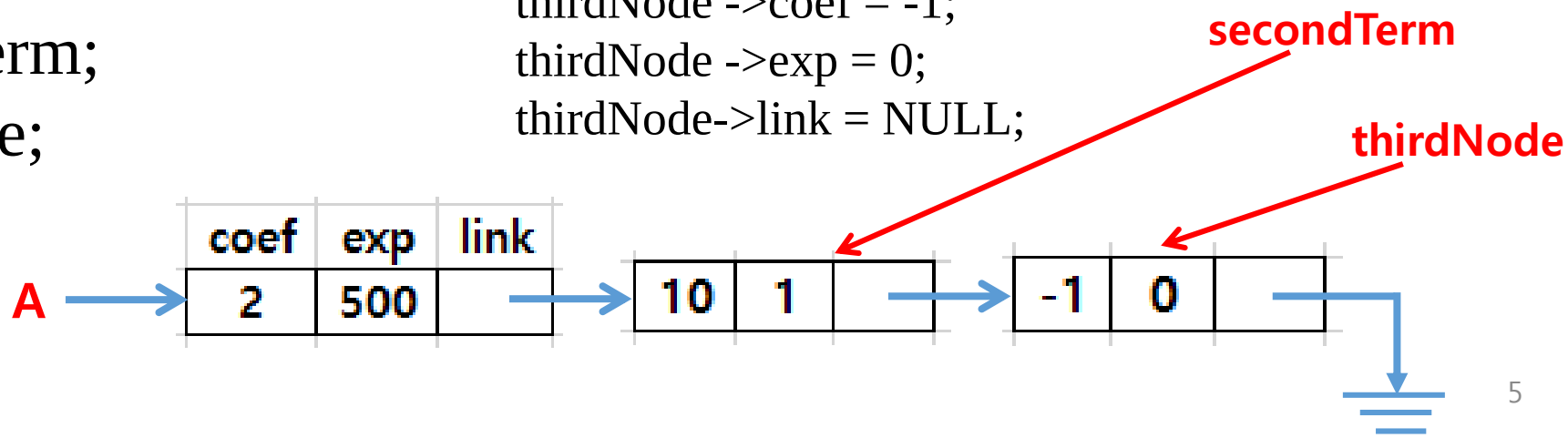
Declaration 2

```
typedef struct term *polynomial;
typedef struct term {
    int coef;
    int exp;
    polynomial link;
};
polynomial A;
```

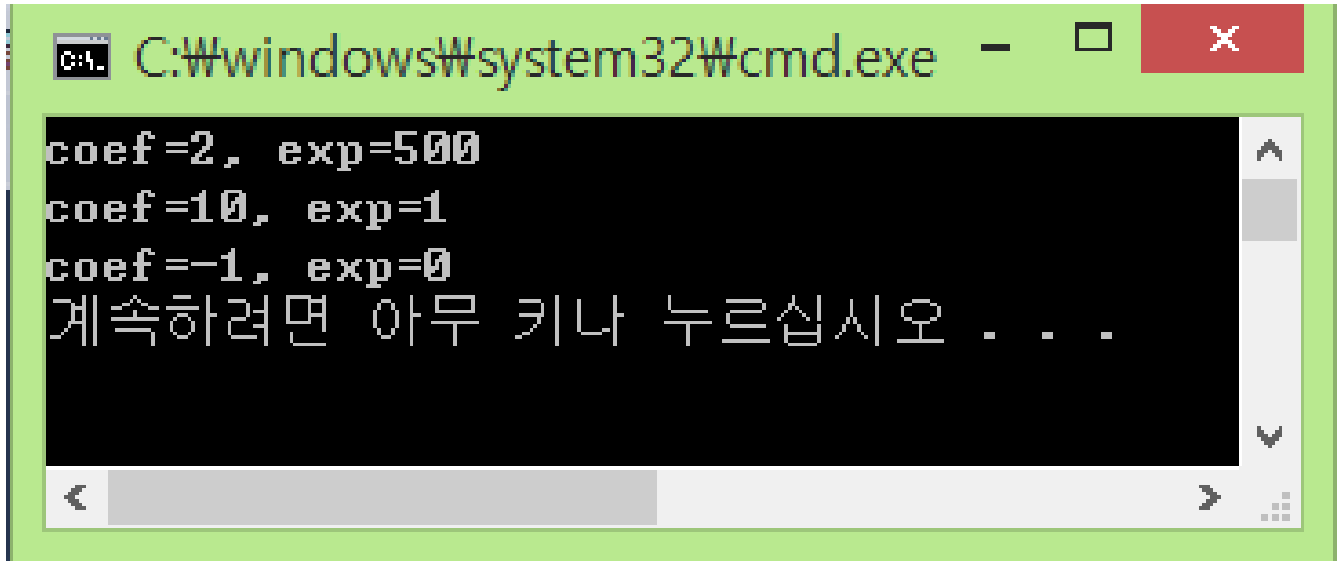
Dynamic allocation of a node

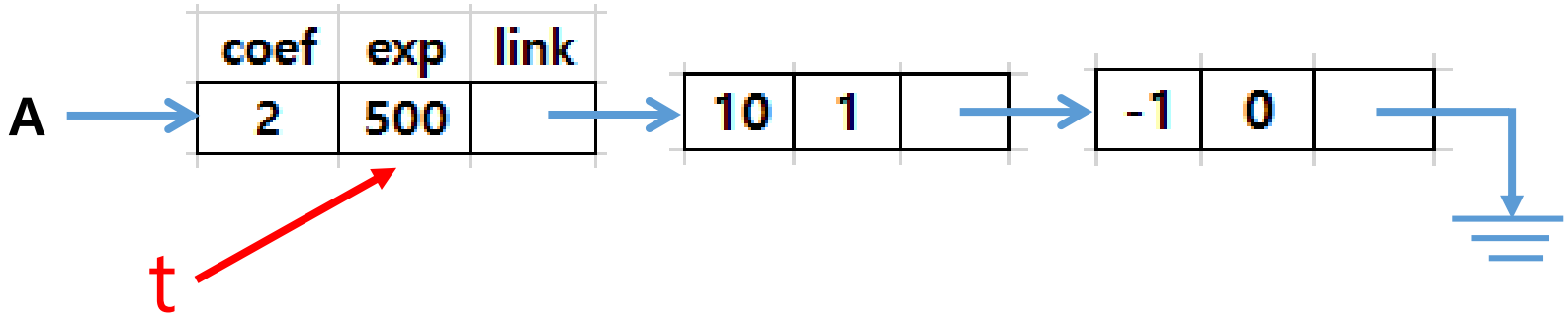
```
typedef struct term *listPtr;
typedef struct term {
    int coef;
    int exp;
    listPtr link;
};
listPtr A;
listPtr secondTerm;
listPtr thirdNode;
```

```
A = (listPtr) malloc(sizeof(*A));
A->coef = 2;
A->exp = 500;
secondTerm = (listPtr) malloc(sizeof(*secondTerm));
A->link = secondTerm;
secondTerm ->coef = 10;
secondTerm ->exp = 1;
thirdNode = (listPtr) malloc(sizeof(struct term));
secondTerm ->link = thirdNode;
thirdNode ->coef = -1;
thirdNode ->exp = 0;
thirdNode->link = NULL;
```

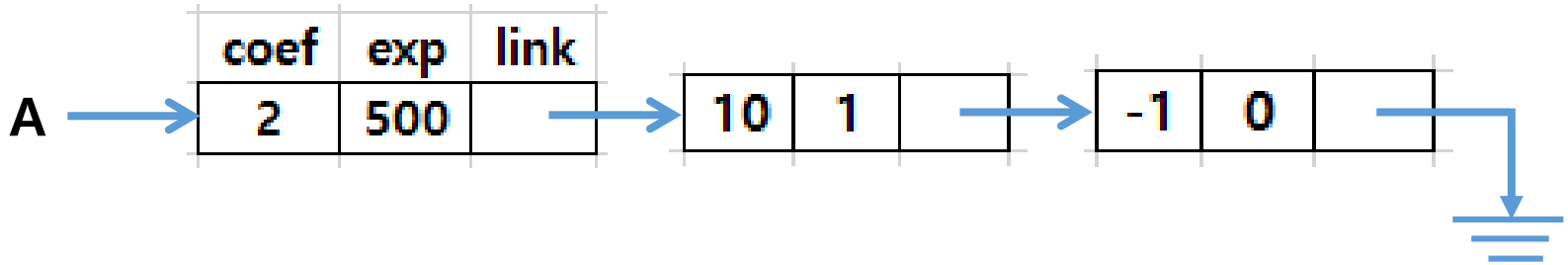


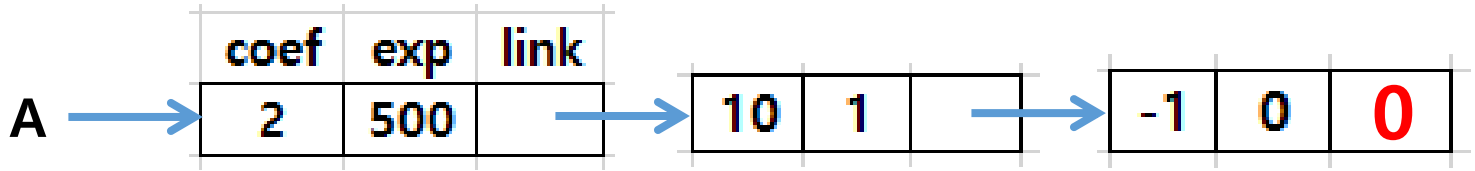
```
#include "stdafx.h"
#include "stdlib.h"
void main() {
    //********
    // C code in the previous slide
    //********
    listPtr t = A;
    while(t != NULL) {
        printf("coef=%d, exp=%d₩n", t->coef, t->exp);
        t = t->link;
    }
}
```

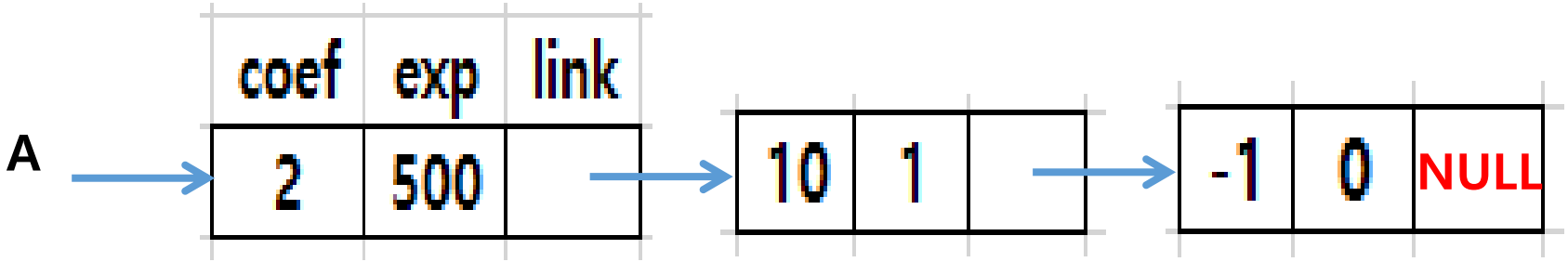




Link of the last node: NULL pointer





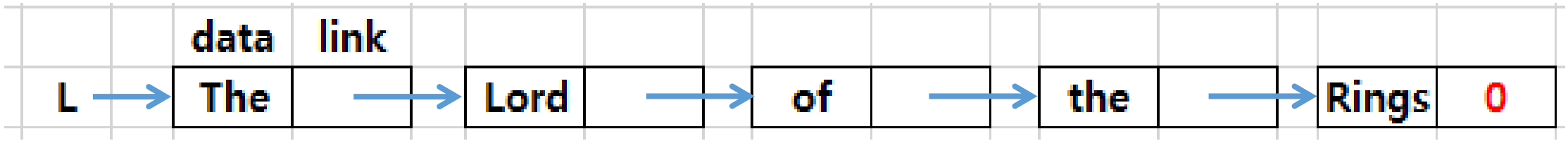


Search a node

- Traverse the list nodes
- Looping through links

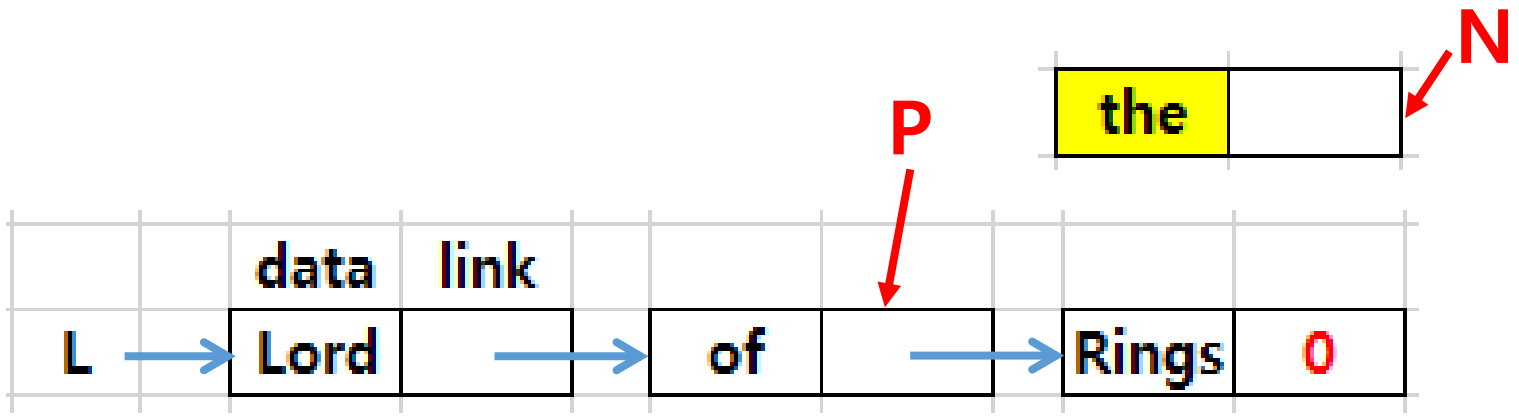
```
p ← L
while(p != NULL) {
    if(p->data=“Rings”) ……
    p ← p->link
}
```

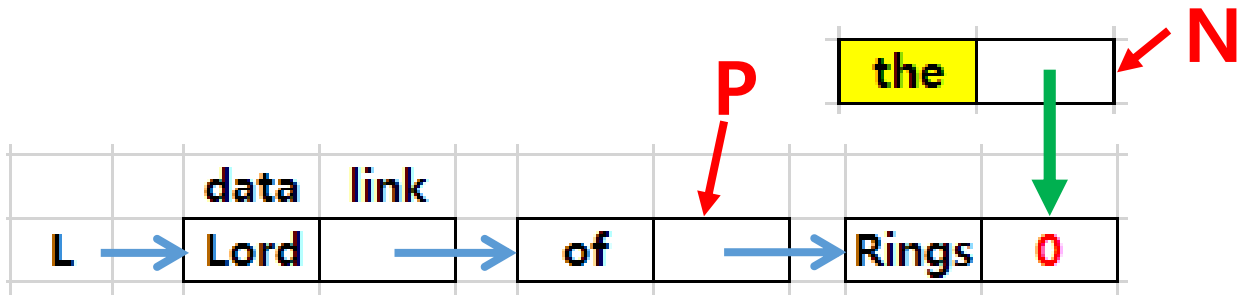
```
for(p ← L; p != NULL; p ← p->link) {
        if(p->data=“Rings”) …..
}
```

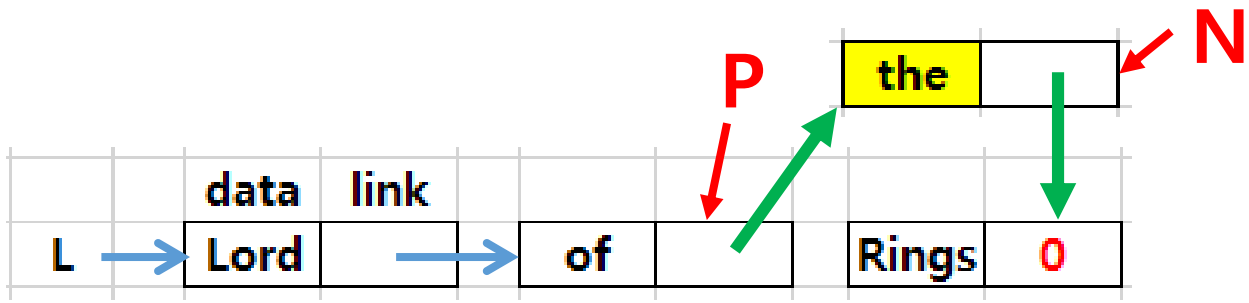


Node insertion

- Insert a node
  - Where?: after *P
  - Data = “the”
- Memory allocation for a node
  - alloc_node()
- Order of link updates





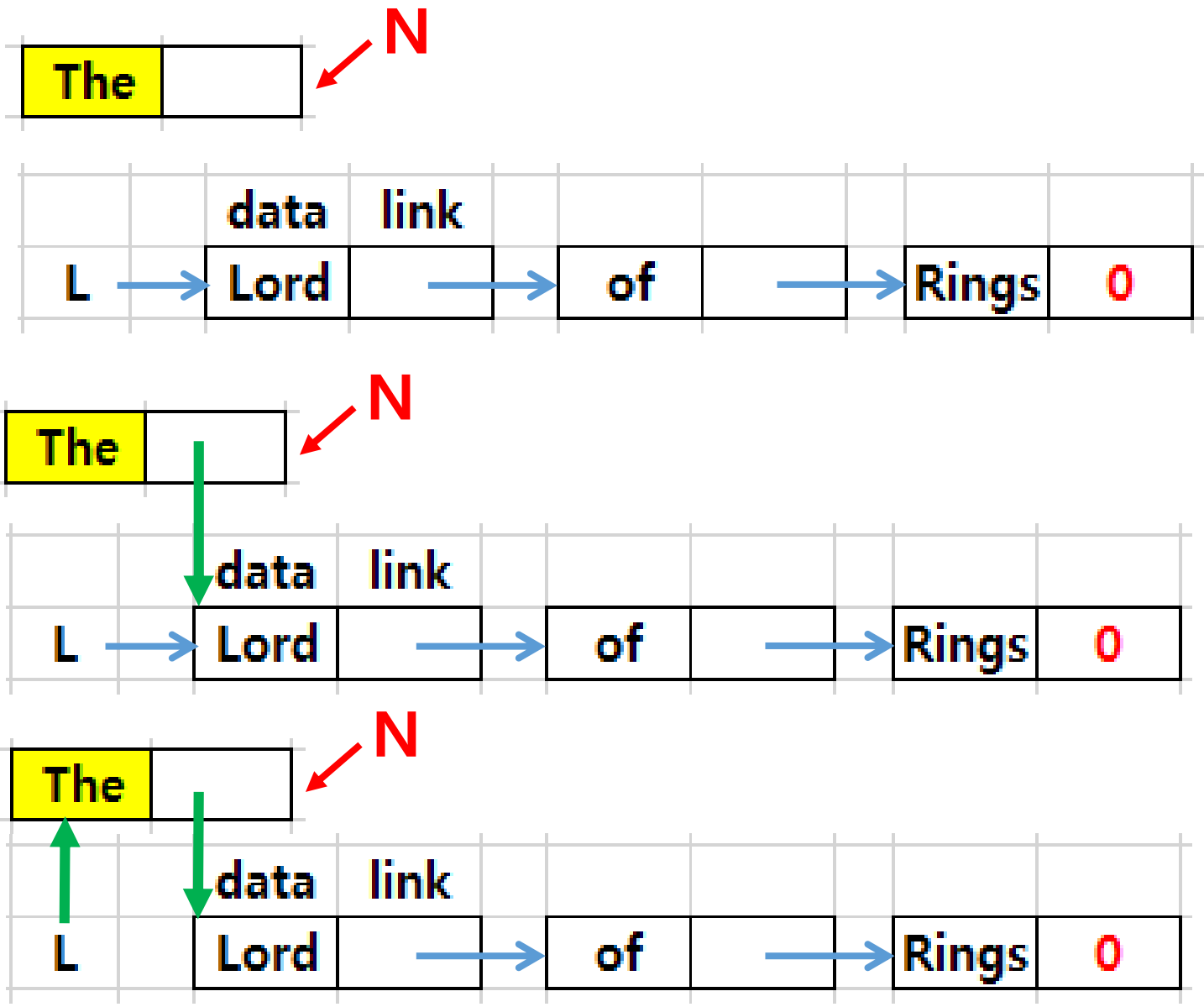



```
N ← alloc_node()
N->data ← “the”
N->link ← p->link
P->link ← N
```

Node insertion(2)

- Insert a node
  - Where?: at the front
  - Data = "The"

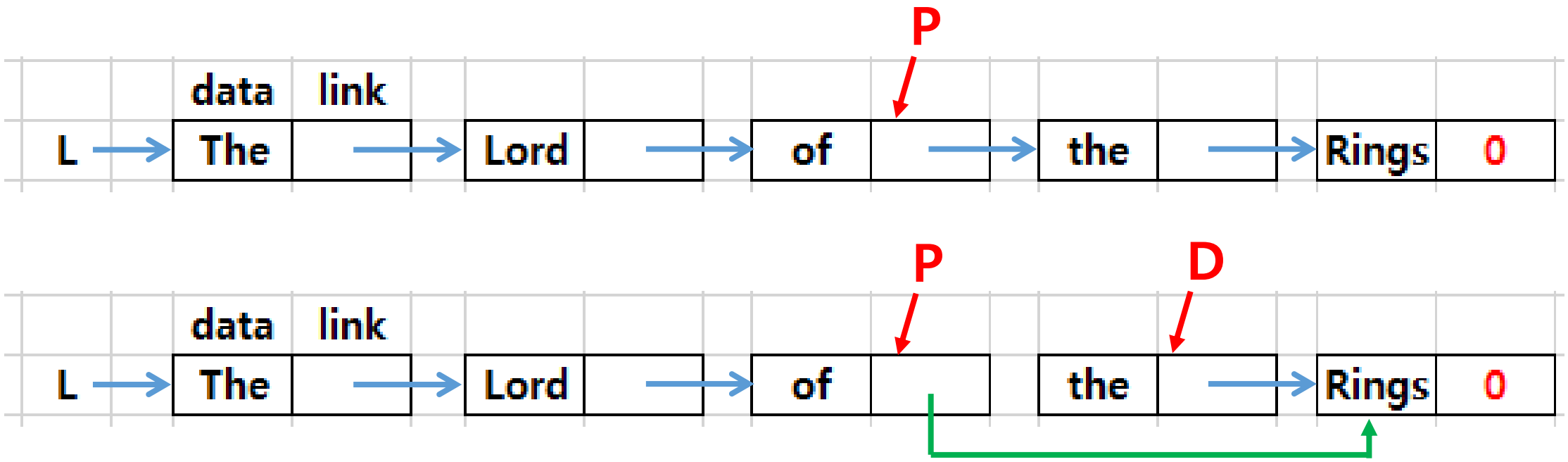
```
N ← alloc_node()
N->data ← "The"
N->link ← L
L ← N
```



Node deletion

- delete a node
  - Which node?: the one after *P
  - Return space of deleted node
    - free_node()

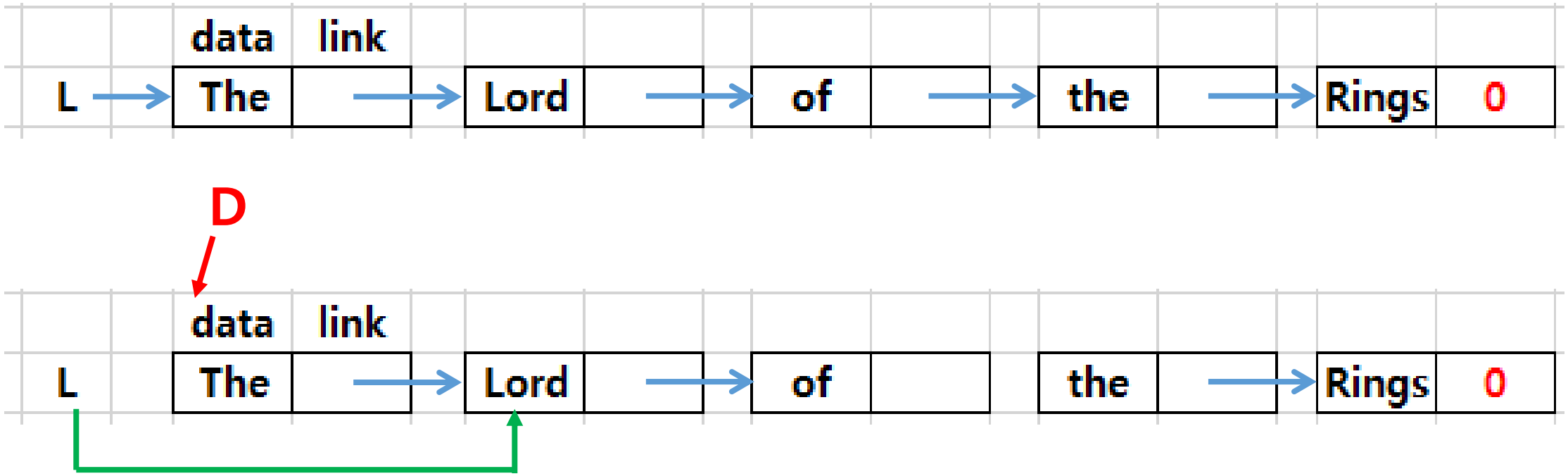
```
D ← P->link
P->link ← D->link //D != NULL
free_node(D)
```



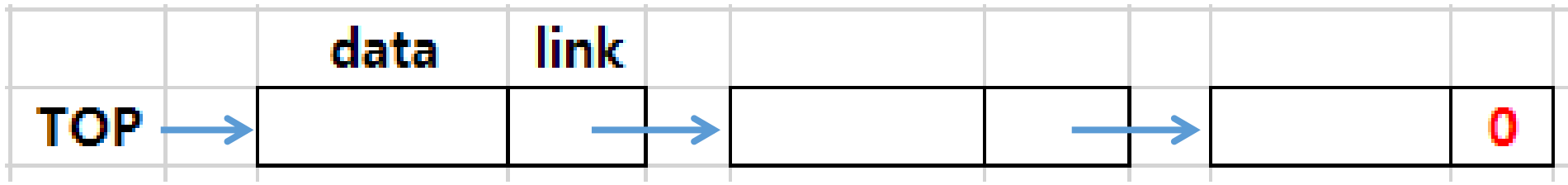
Node deletion(2)

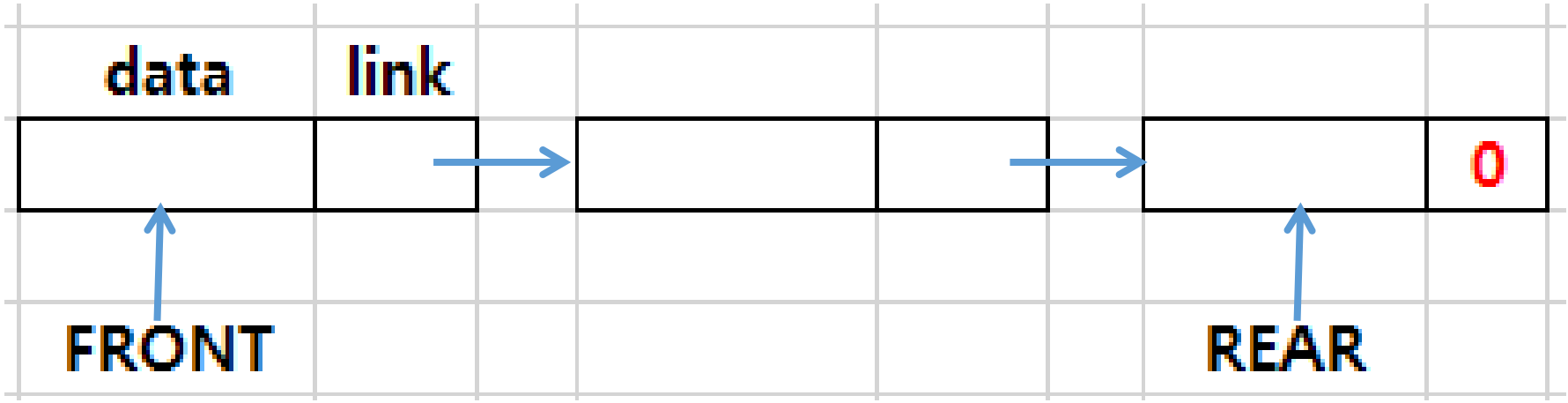
- delete a node
  - Which node?: the first one

```
D ← L
L ← D->link //D != NULL
free_node(D)
```



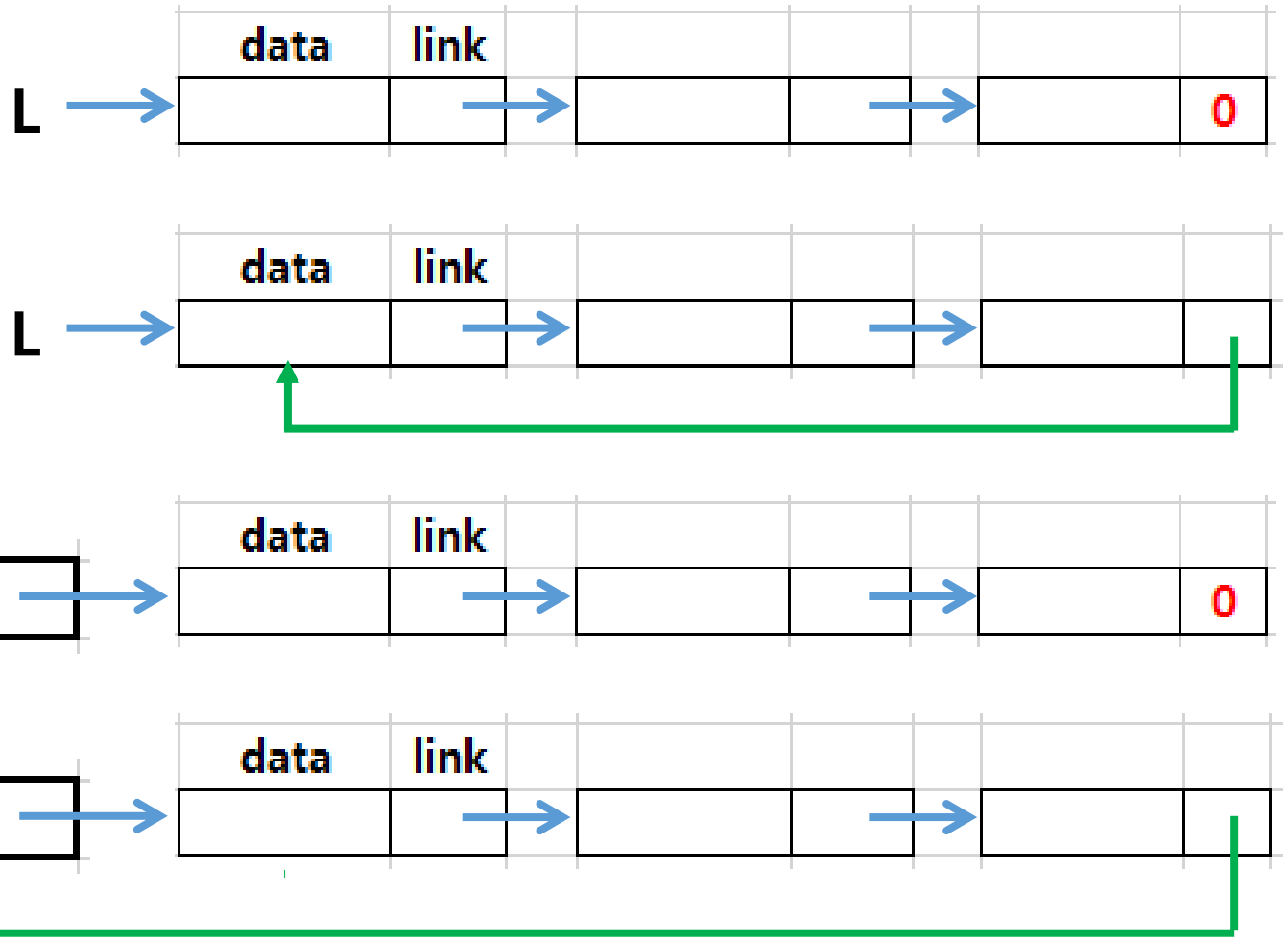
Linked representation of stacks and queues



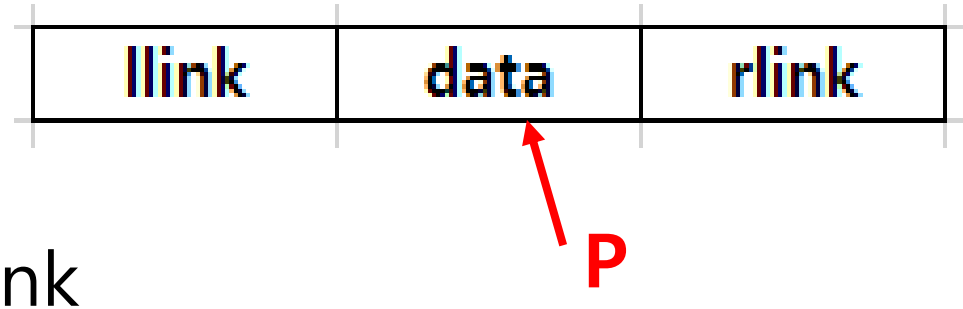


Variations of linked lists

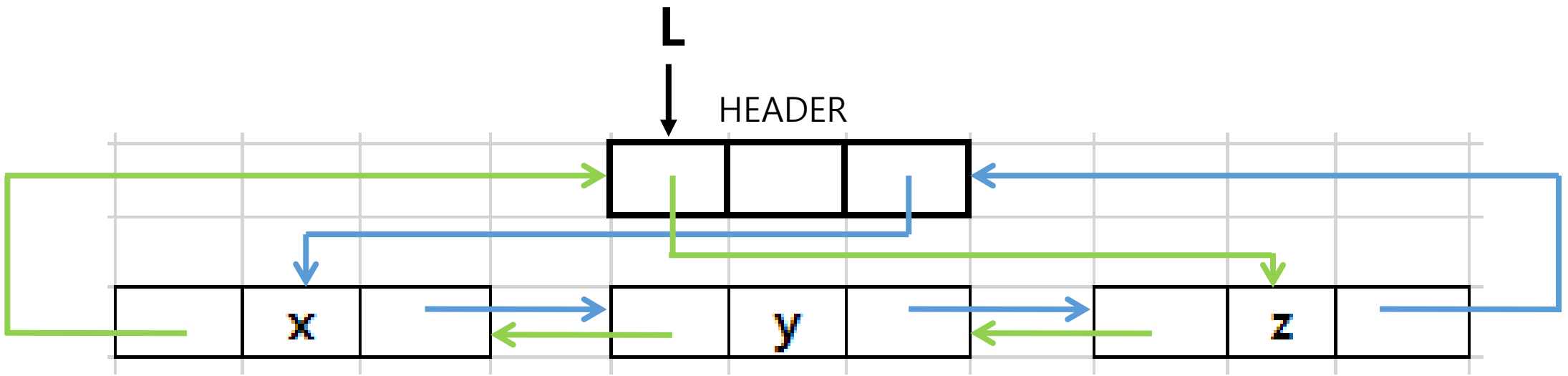
- Singly linked list (Chain)
- Circularly linked list
- List with a header node
- Doubly linked list



Doubly linked list

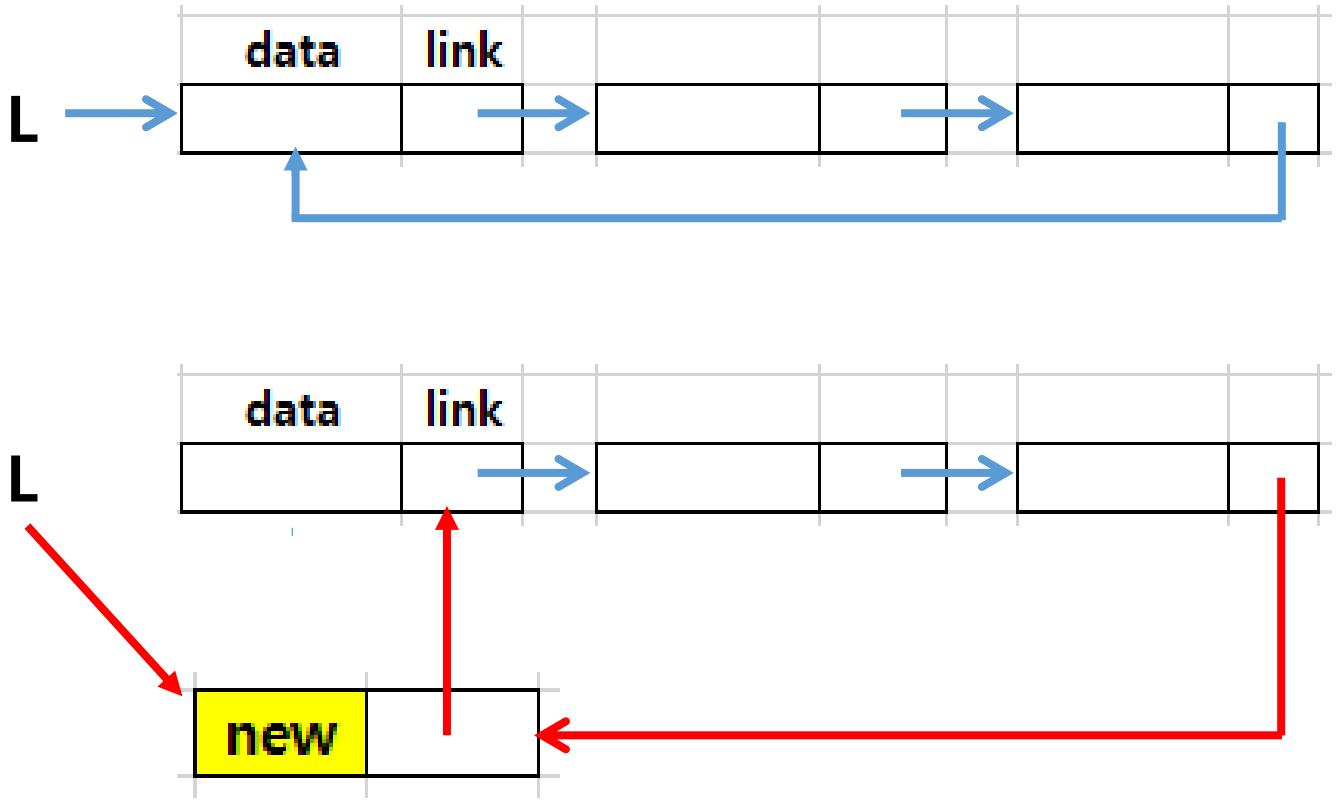


- Two links: left & right
- P = P->llink->rlink = P->rlink->llink
- Circularly & doubly linked list with a header node



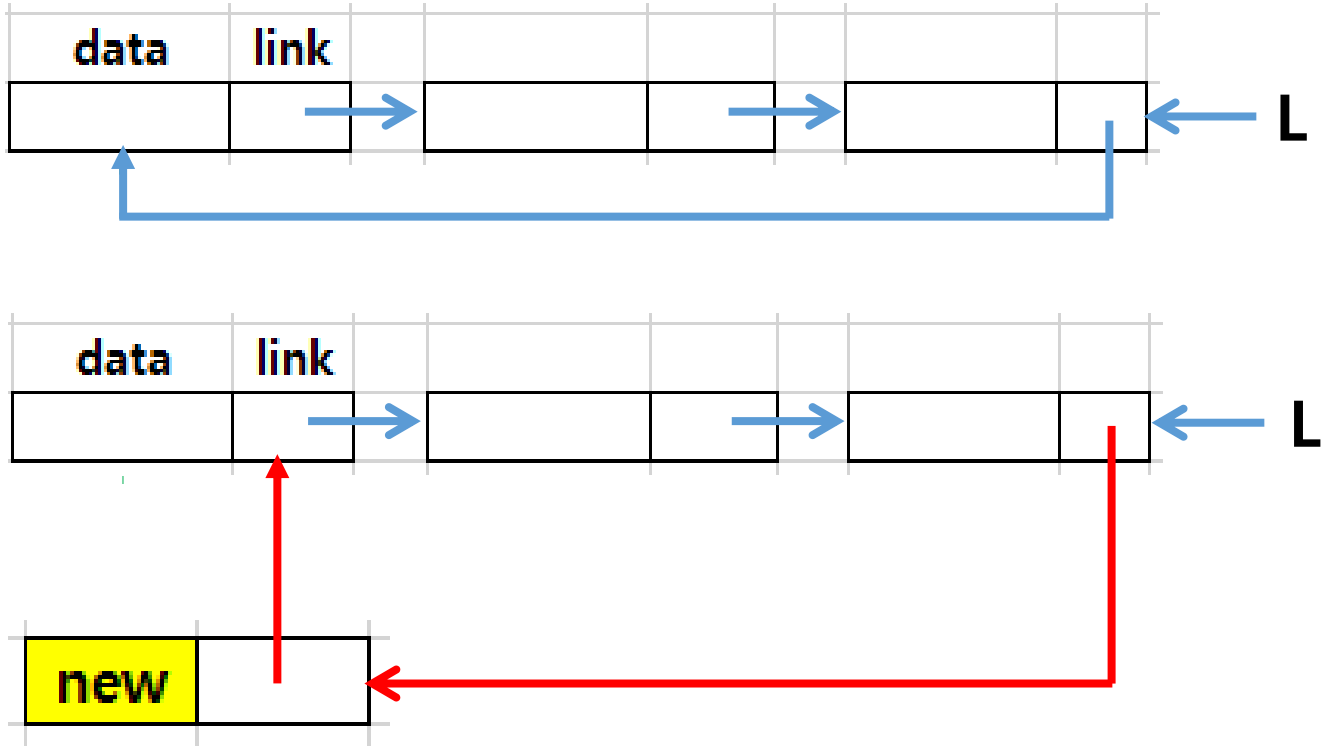
Insert a node at the front of a circular list

- Update of the last node's link: O(n)



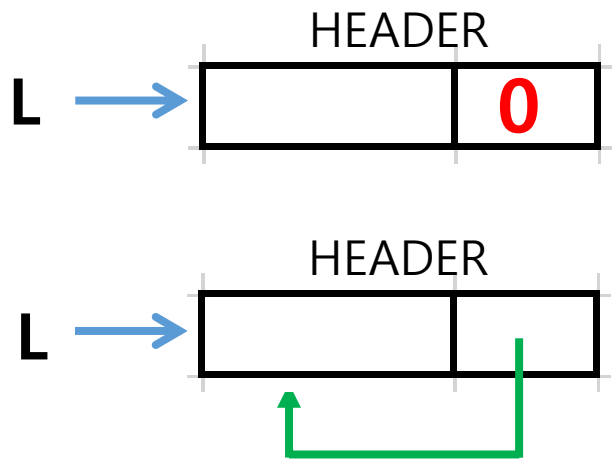
Insert a node at the front of a circular list(2)

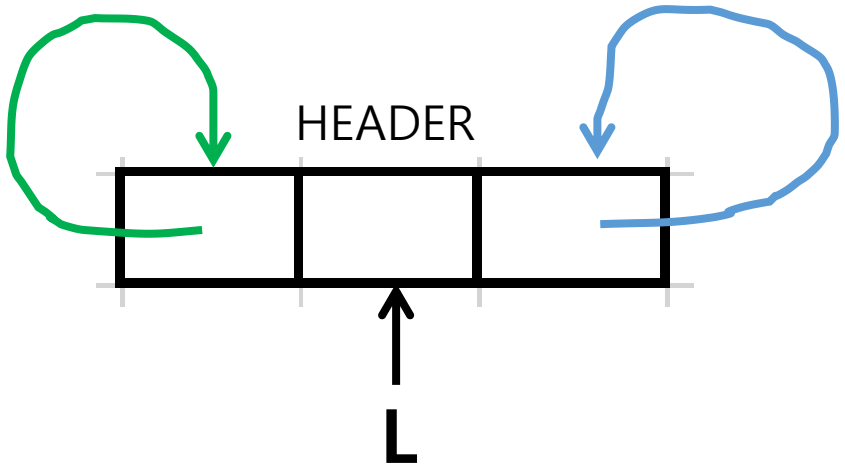
- Solution: let the list pointer point to the last node
  - Access to the first node: O(1)
- Update of the last node's link: O(1)



Header node

- Could store some info. about the list
- Empty list: still with the header node
  - Makes it easier to write codes for operations





Polynomial addition

- Polynomial representations
  - Chain
  - Circular list with a header node
    - Assumption: exponents >=0
    - Exponent in the header = -1
- Example
  - $A(x) = 2x^{500} + 10x - 1$