Lists (a first look)

CSE 373
Data Structures

Readings

- Reading
  - Chapter 3
  - You can start peeking at Chapter 6

We will cover

- List ADT (a first look)
- List implementation
  - Array
  - Linked list
  - Doubly linked list
- An example application (long integers)
- Circular list

List ADT

- What is a List?
  - Ordered sequence of elements A_1, A_2, …, A_N
- Elements may be of arbitrary type, but all are of the same type
- Elements have values
- Elements have positions (first, kth, last etc..)

Common operations on lists

- Constructor for an empty list
- Queries: size(); isEmpty();
- Insert and delete
  - Must indicate where: first, last, kth, after some element etc…
- Find, set, replace
  - With a given value, find previous etc…
- Will look at a “list interface” in the Java sense later

Simple Examples of List Use

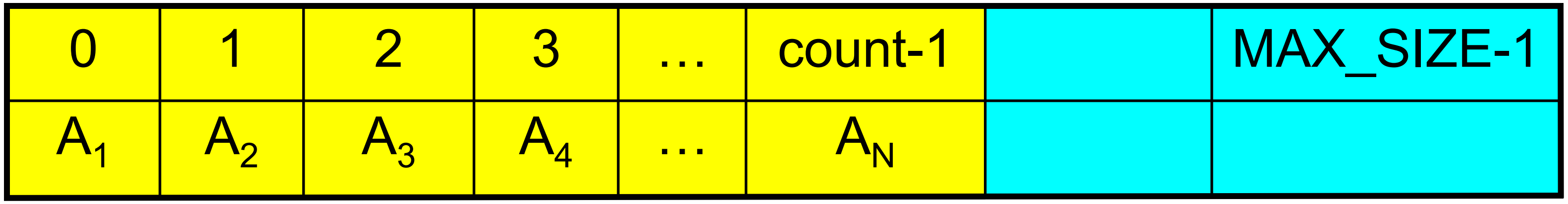
- Polynomials
  - $25 + 4x^2 + 75x^{85}$
  - An element is a term whose value must indicate the power and the coefficient
- Unbounded Integers
  - 4576809099383658390187457649494578
  - Do not fit within a single computer word
  - An element has for value a single digit

List Implementations

- Two types of implementation:
  - Array-Based
  - Linked list (pointer based)

List: Array Implementation

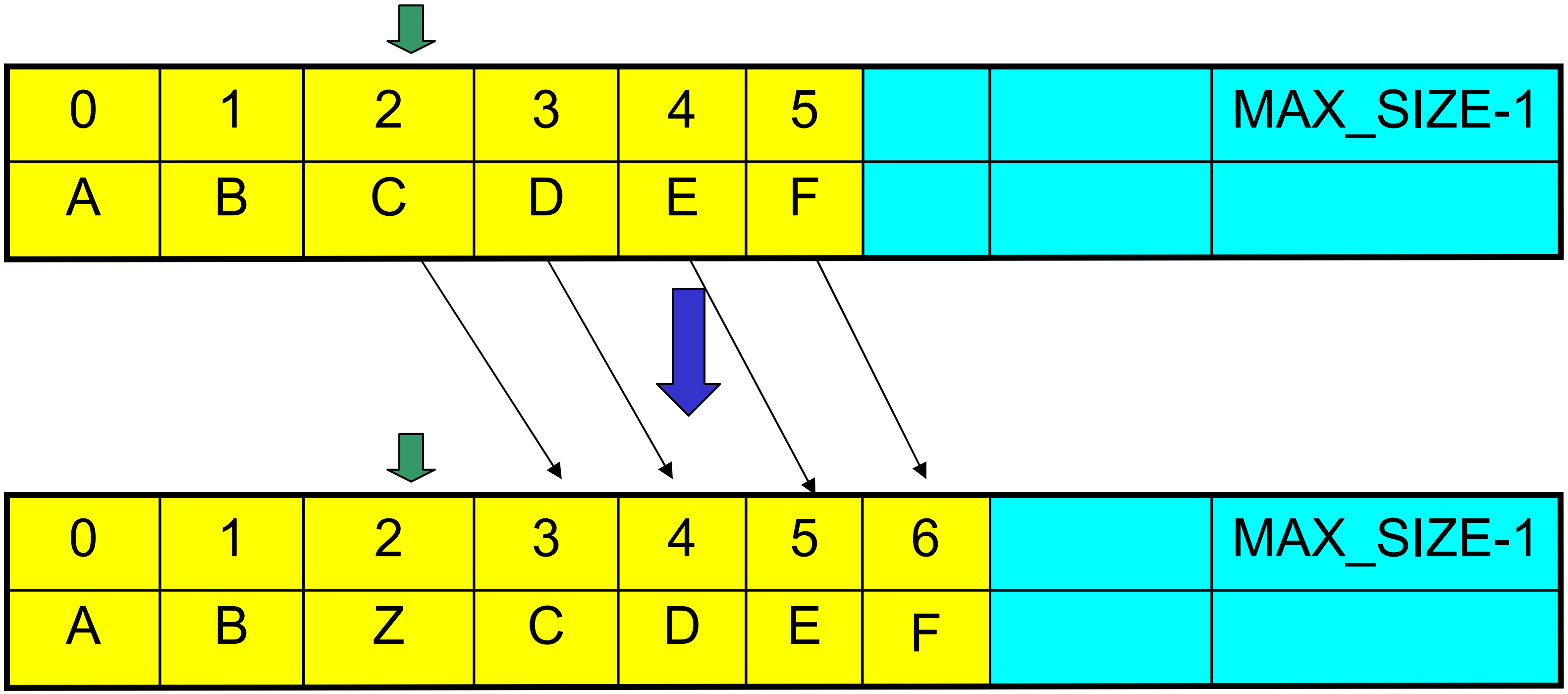
- Basic Idea:
    - Pre-allocate a big array of size MAX_SIZE
    - Keep track of current size using a variable `count`
    - Shift elements when you have to insert or delete (except of course for insertlast and deletelast)



0	1	2	3	…	count-1		MAX_SIZE-1
A_1	A_2	A_3	A_4	…	A_N		

List: Array Implementation

Insert Z in kth position



0	1	2	3	4	5			MAX_SIZE-1
A	B	C	D	E	F			

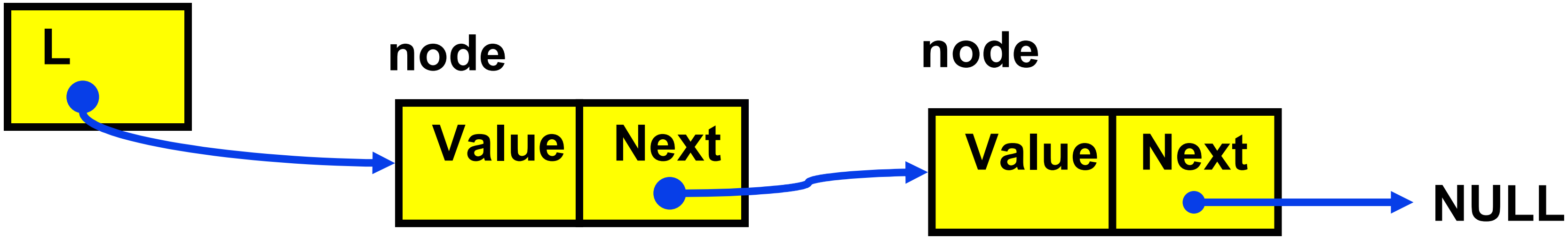
0	1	2	3	4	5	6		MAX_SIZE-1
A	B	Z	C	D	E	F		

Array Insert_kth Running Time

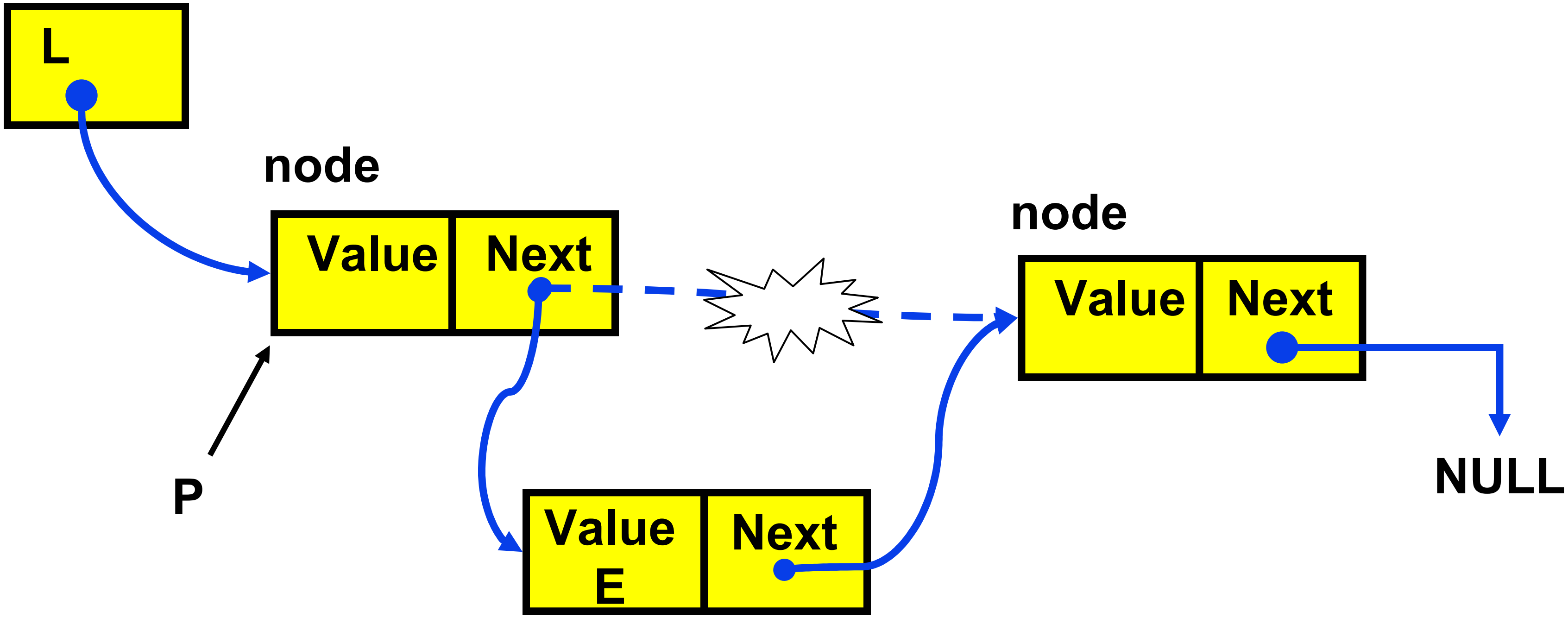
- Running time for N elements?
- On average, must move half the elements to make room – assuming insertions at positions are equally likely
- This is O(N) running time.
- Worst case is insert at position 0. Must move all N items one position before the insert. Still O(N)

Linked Implementation

- Basic Idea:
  - Allocate little blocks of memory (nodes) as elements are added to the list
  - Keep track of list by linking the nodes together
  - Change links (pointers) when you want to insert or delete



Linked list: Insert_after

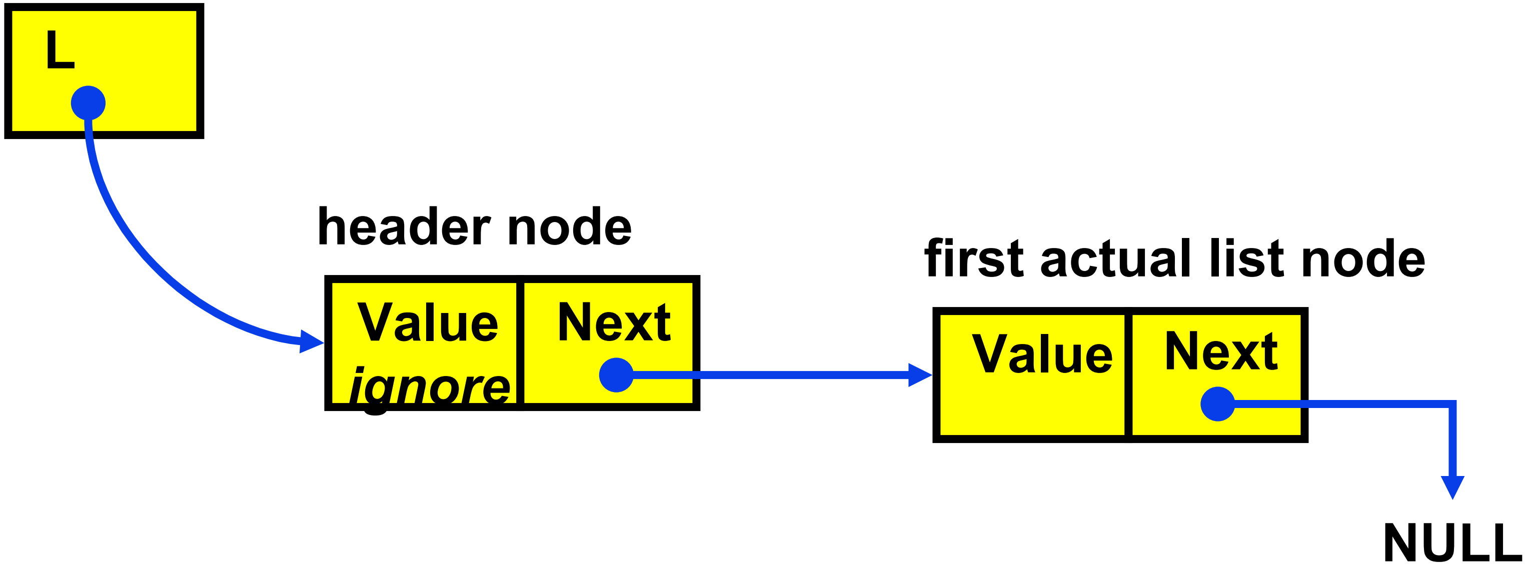


Insert the value E after P

Insertion After

```
InsertAfter(p : node, e : thing): {
x : node; //declares the type of x
x := new node;
x.value := v;
x.next := p.next;//be sure to do in right order
p.next := x;
}
```

Linked List with Header Node



Advantage: “insert after” and “delete after” can be easily done at the beginning of the list (insert_first and delete_first)

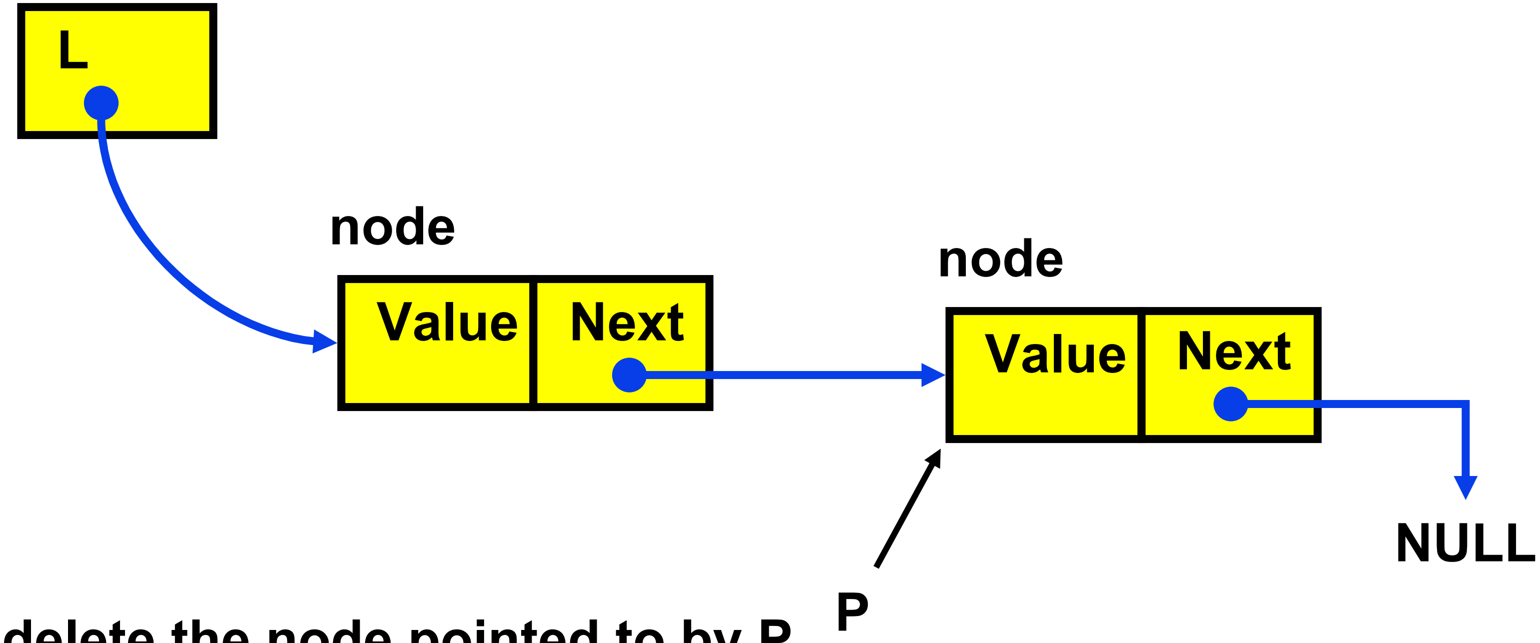
Linked list Implementation Caveats

- Whenever you break a list, your code should fix the list up as soon as possible
  - Draw pictures of the list to visualize what needs to be done
- Pay special attention to boundary conditions:
  - Empty list
  - Single item – same item is both first and last
  - Two items – first, last, but no middle items

Linked List Insert Running Time

- Running time for N elements?
- “Insert_after” takes constant time (O(1))
- Does not depend on input size
- Compare to array based list Insert_kth which is O(N)
- However, how about Insert_last?

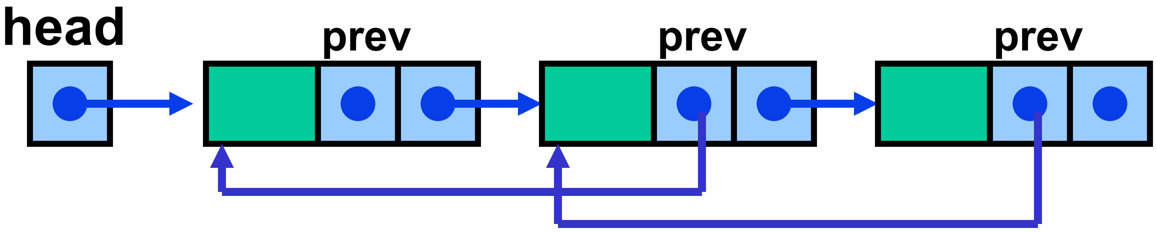
Linked List Delete



To delete the node pointed to by P,
need a pointer to the previous node.
Thus we need to traverse the list to find the previous node.
So we might want to use …

Doubly Linked Lists

- In singly linked lists, findPrevious (and hence Delete) is slow [O(N)] because we cannot go directly to previous node
- Solution: Keep a "previous" pointer at each node

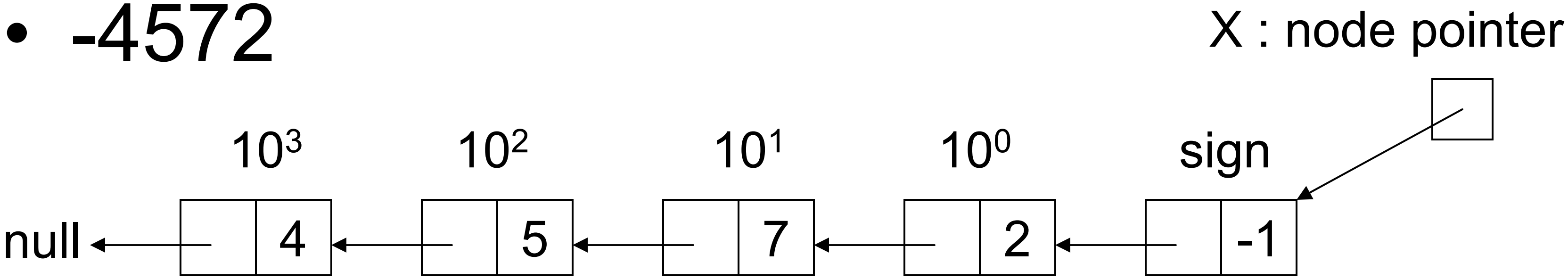


Double Link Pros and Cons

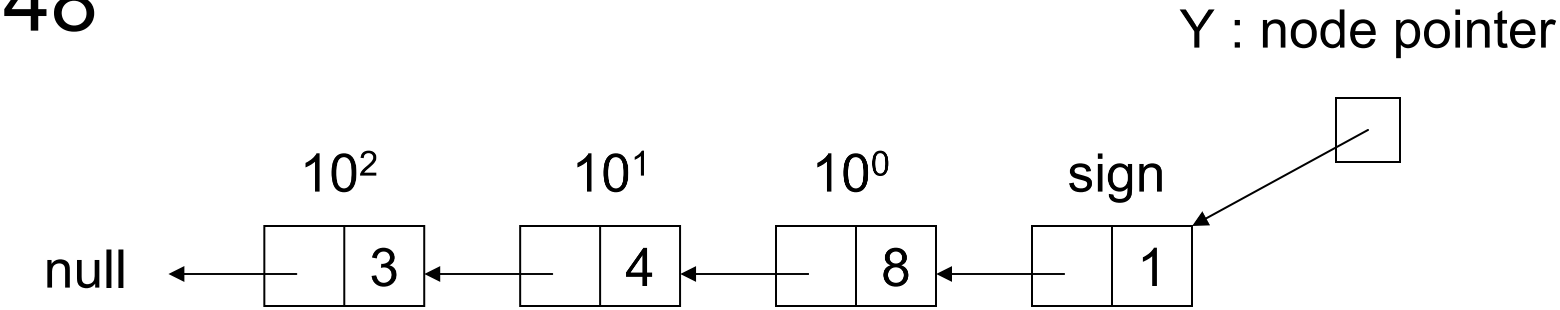
- Advantage
  - › Delete (not DeleteAfter) and FindPrev are faster
- Disadvantages:
  - › More space used up (double the number of pointers at each node)
  - › More book-keeping for updating the two pointers at each node (pretty negligible overhead)

Unbounded Integers Base 10

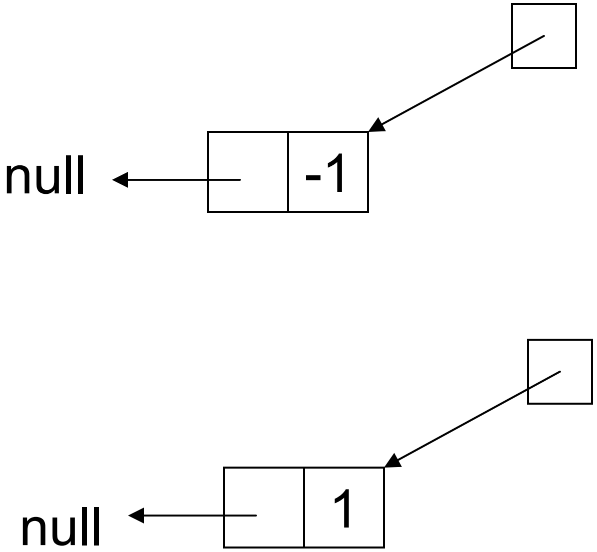
- -4572



- 348

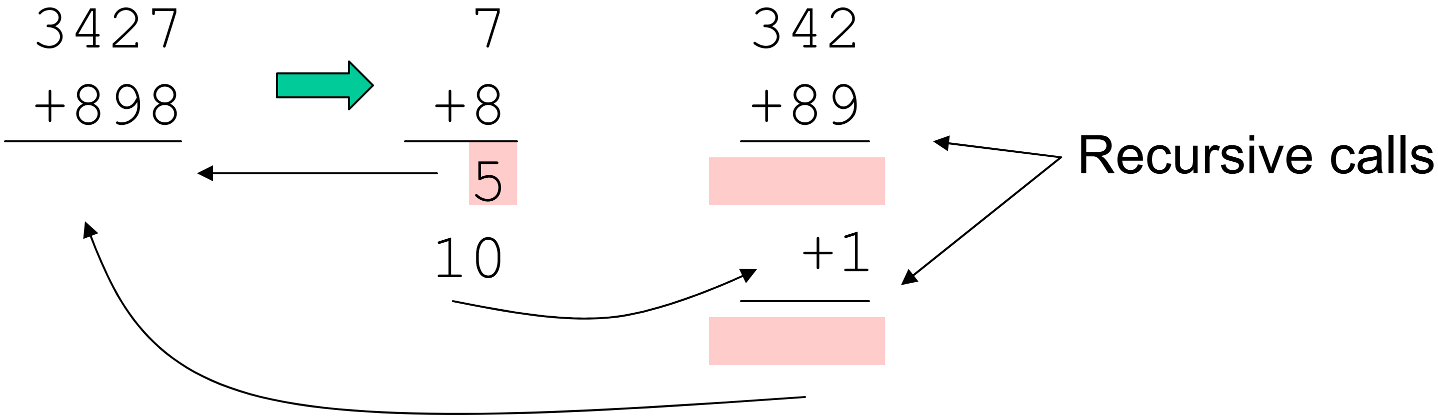


Zero



Recursive Addition

- Positive numbers (or negative numbers)



3427
+898

7
+8
5
10

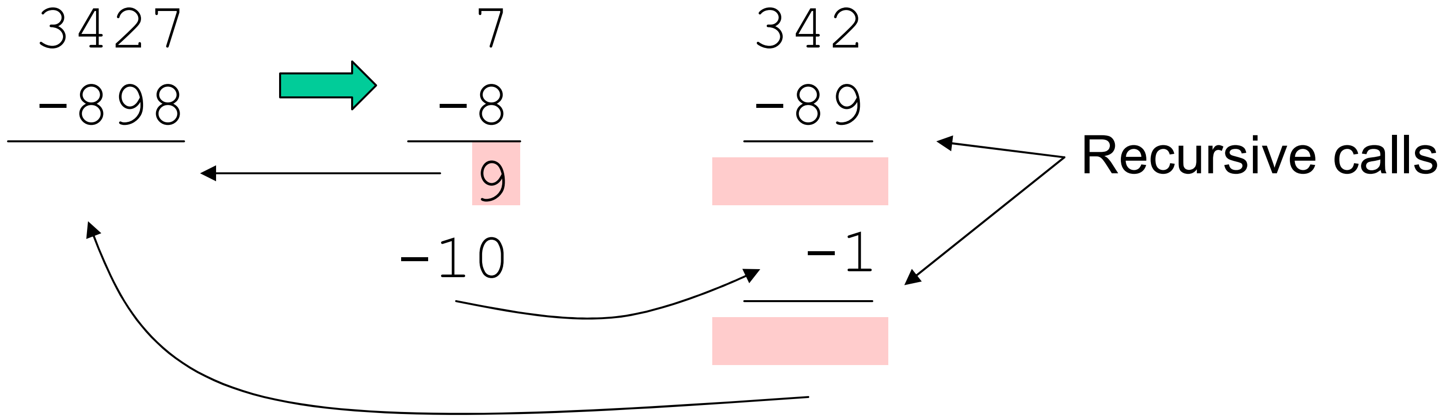
342
+89

+1

Recursive calls

Recursive Addition

- Mixed numbers



```
 3427        7      342
 -898       -8      -89
            9
          -10        -1
```

Recursive calls