Linked Stacks and Queues

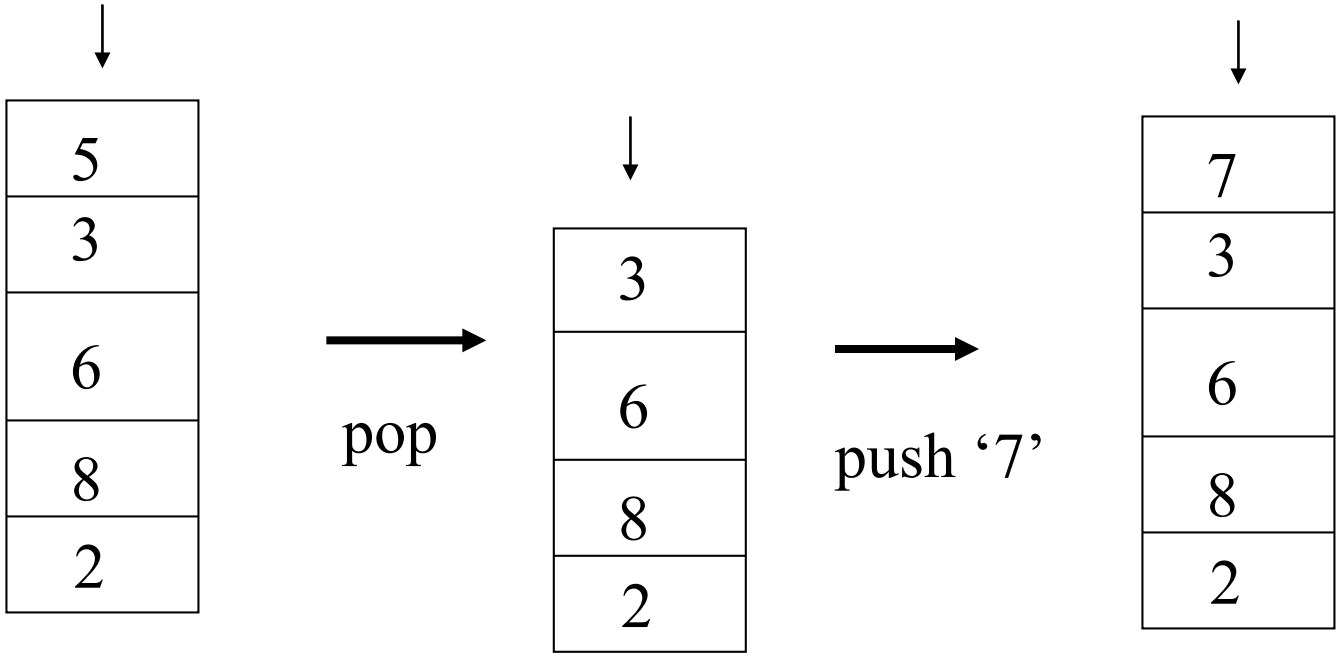
Lecture 15

Menu

- A Stack using a Linked List with a header
- A Queue using a Linked List with a header

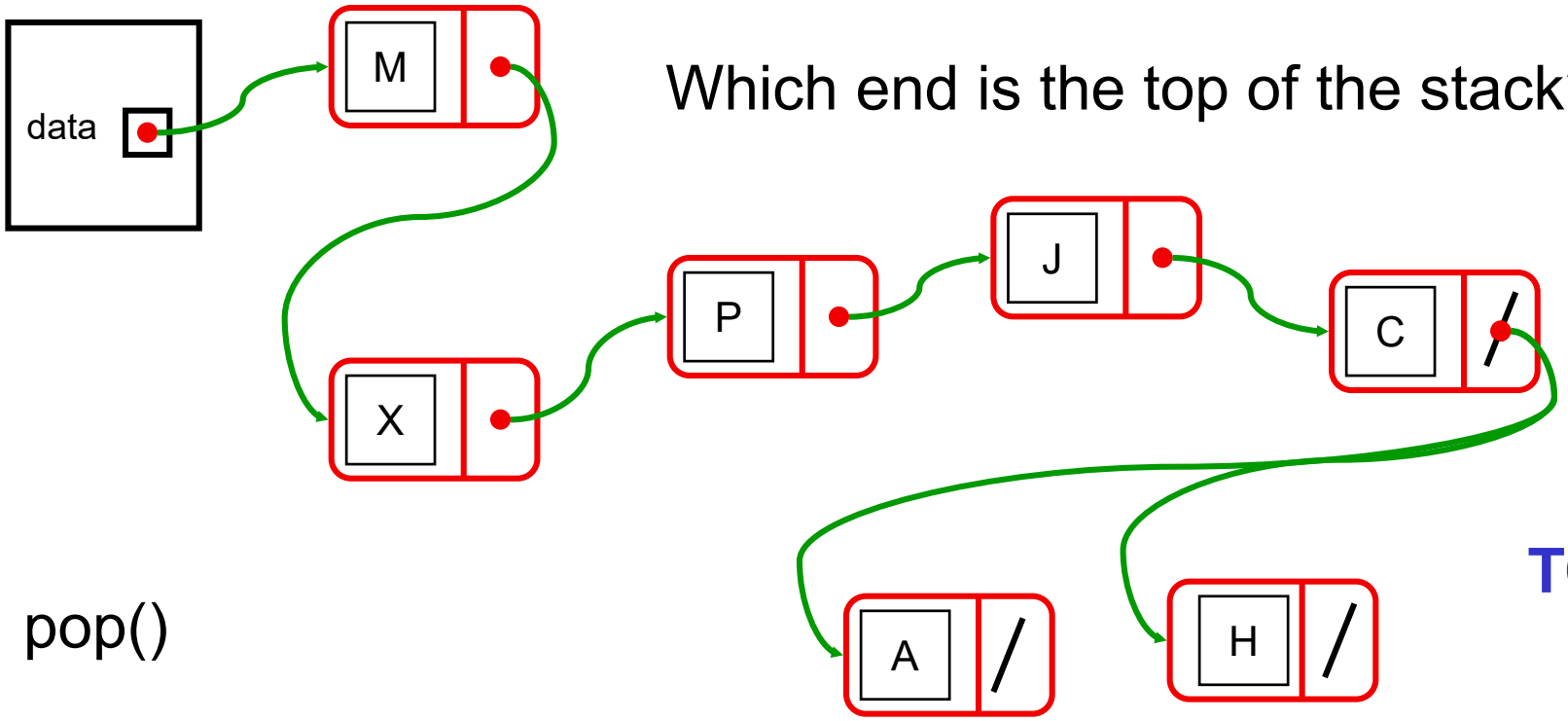
Stacks (LIFO)

- insertions (push) & deletions (pop) only at the end (top)



A Linked Stack

- Implement a Stack using a linked list.



Which end is the top of the stack?

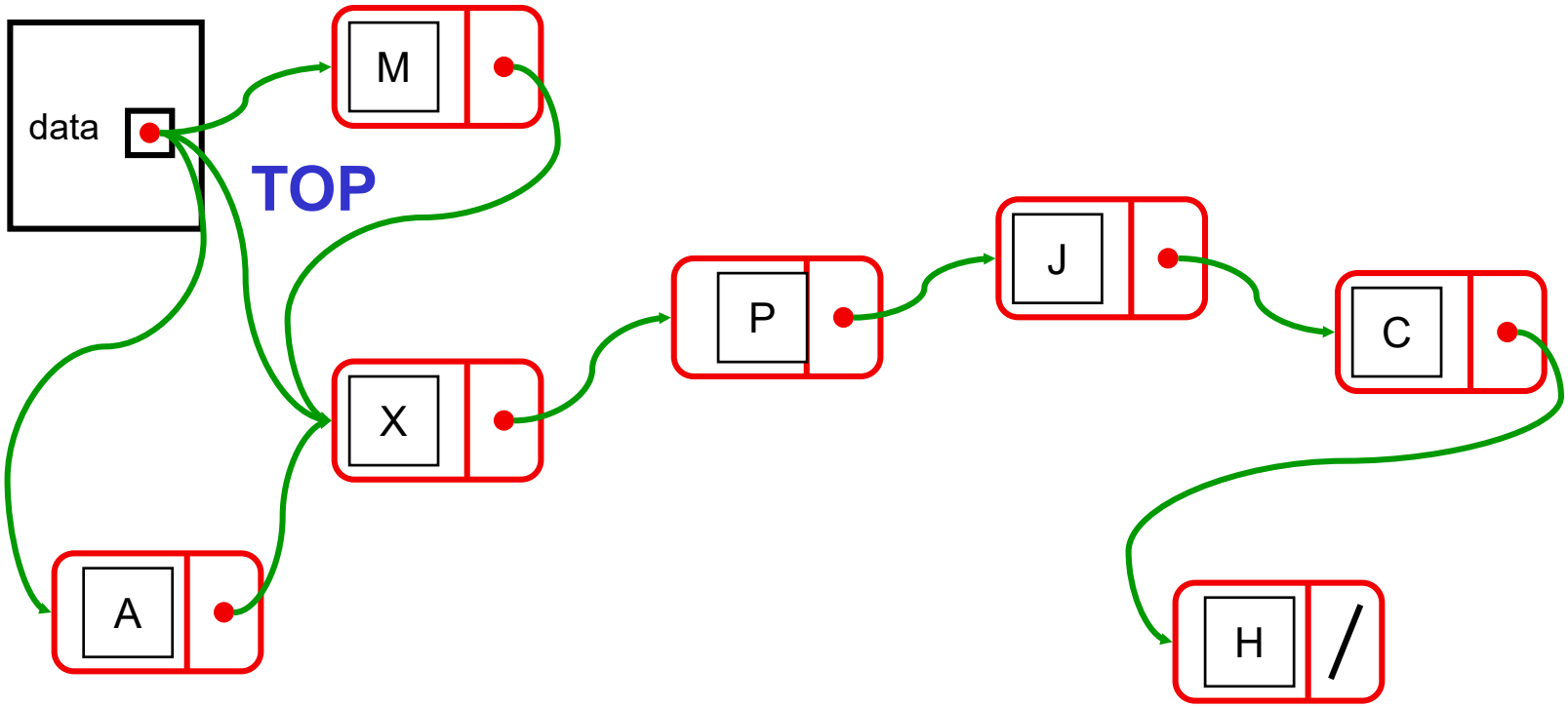
- pop()

- push("A")

Why is this a bad idea?

A Linked Stack

- Make the top of the Stack be the front of the list.



- pop()
- push(“A”)

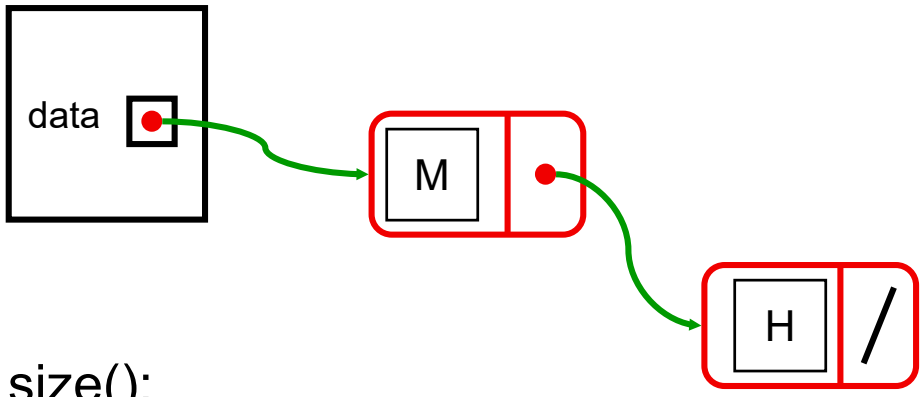
Implementing LinkedStack

- Use the LinkedNode class:

```
public class LinkedStack <E> extends AbstractCollection <E> {
    private Node<E> data = null;
    public LinkedStack(){  }
    public int size(){…
    public boolean isEmpty(){…
    public E peek(){…
    public E pop(){…
    public void push(E item){…
    public Iterator <E> iterator(){
```

LinkedStack

```
public boolean isEmpty(){
    return data==null;
}
public int size () {
    if (data == null)   return 0;
    else                return data.size();
}
```

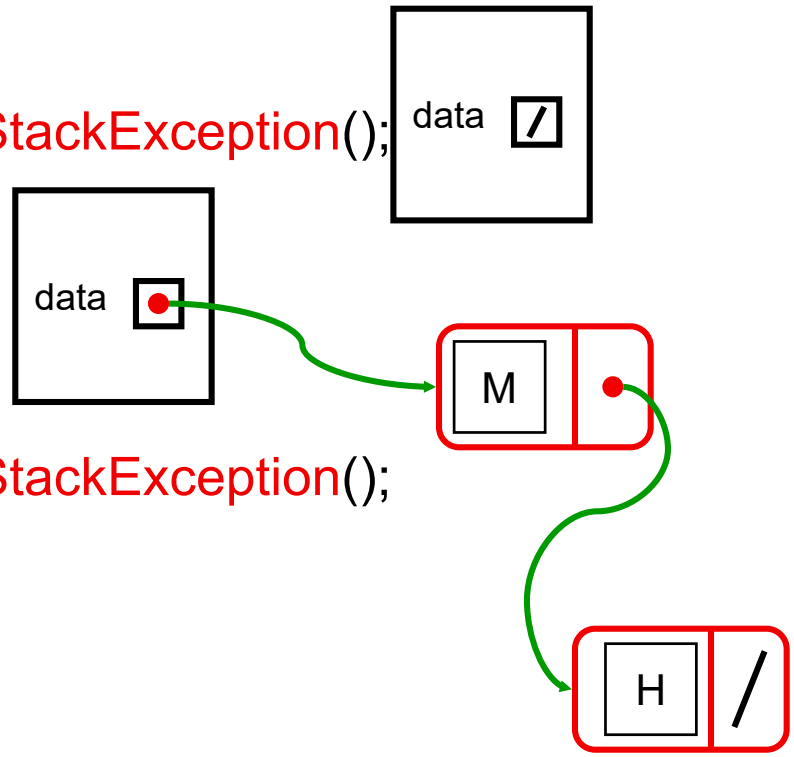


- Need size() method in Node class:

```
public int size (){
    int ans = 0;
    for (Node<E> rest = data; rest!=null; rest=rest.next)
        ans++;
    return ans;
}
```

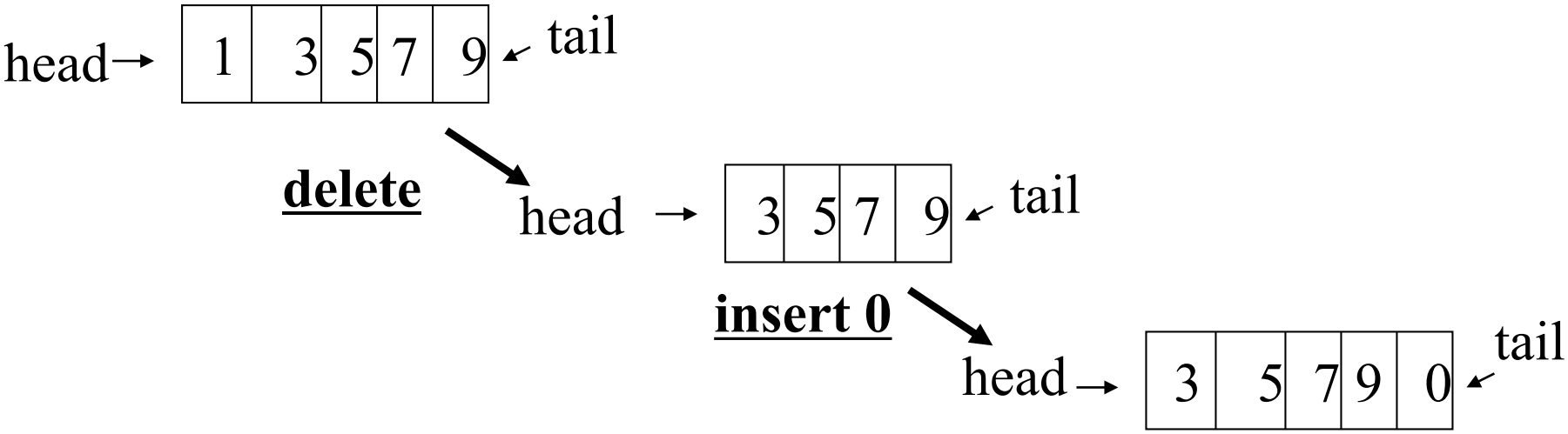
LinkedStack

```
public E peek(){
   if (data==null) throw new EmptyStackException();
   return data.value;
}

public E pop(){
   if (data==null) throw new EmptyStackException();
   E ans = data.value;
   data = data.next;
   return ans;
}
public void push(E item){
   if (item == null) throw new IllegalArgumentException();
   data = new Node(item, data);
}
public Iterator <E> iterator(){
      return new NodeIterator(data);
}
```



Queues (FIFO)

- Example: waiting lines
- Insertion at the end (tail), deletion from the front (head)

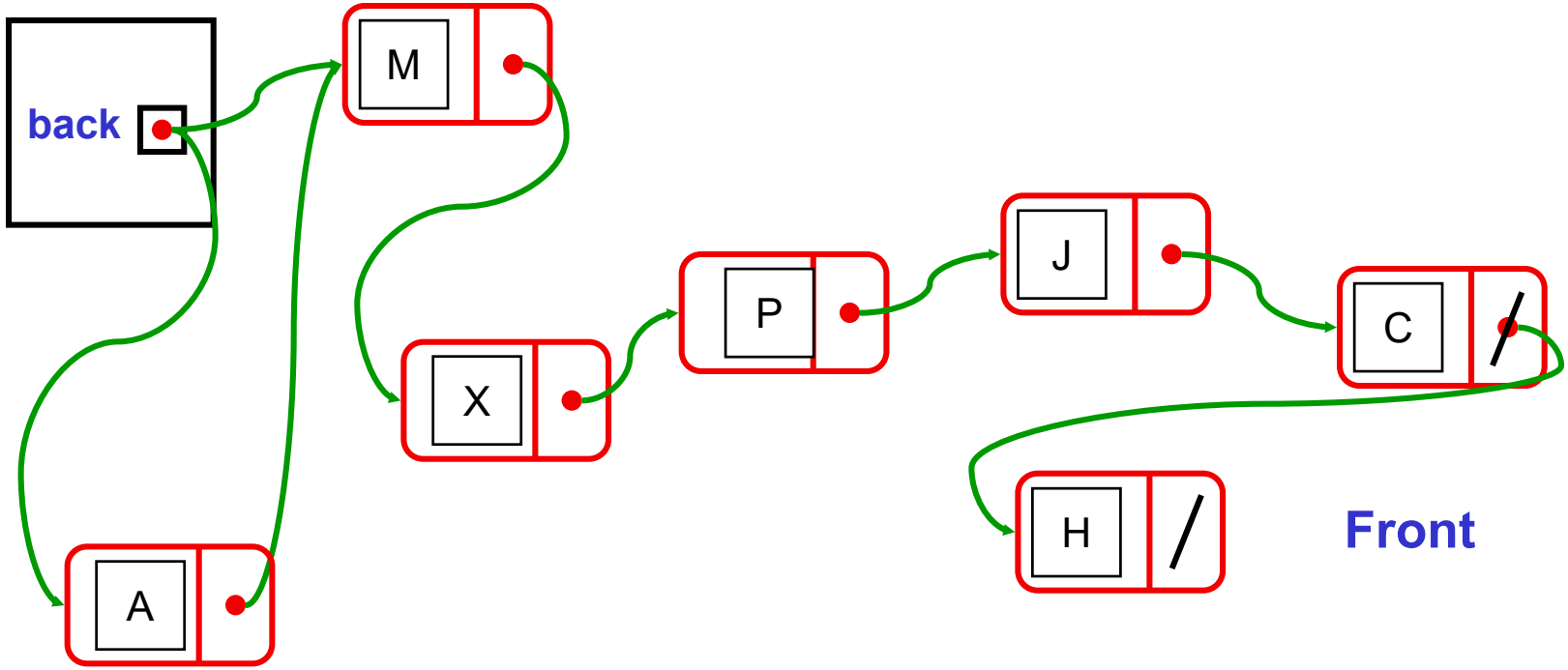


Application of Queues

- user job queue
- print spooling queue
- I/O event queue
- incoming packet queue
- outgoing packet queue

A Linked Queue #1

- Put the front of the queue at the end of the list

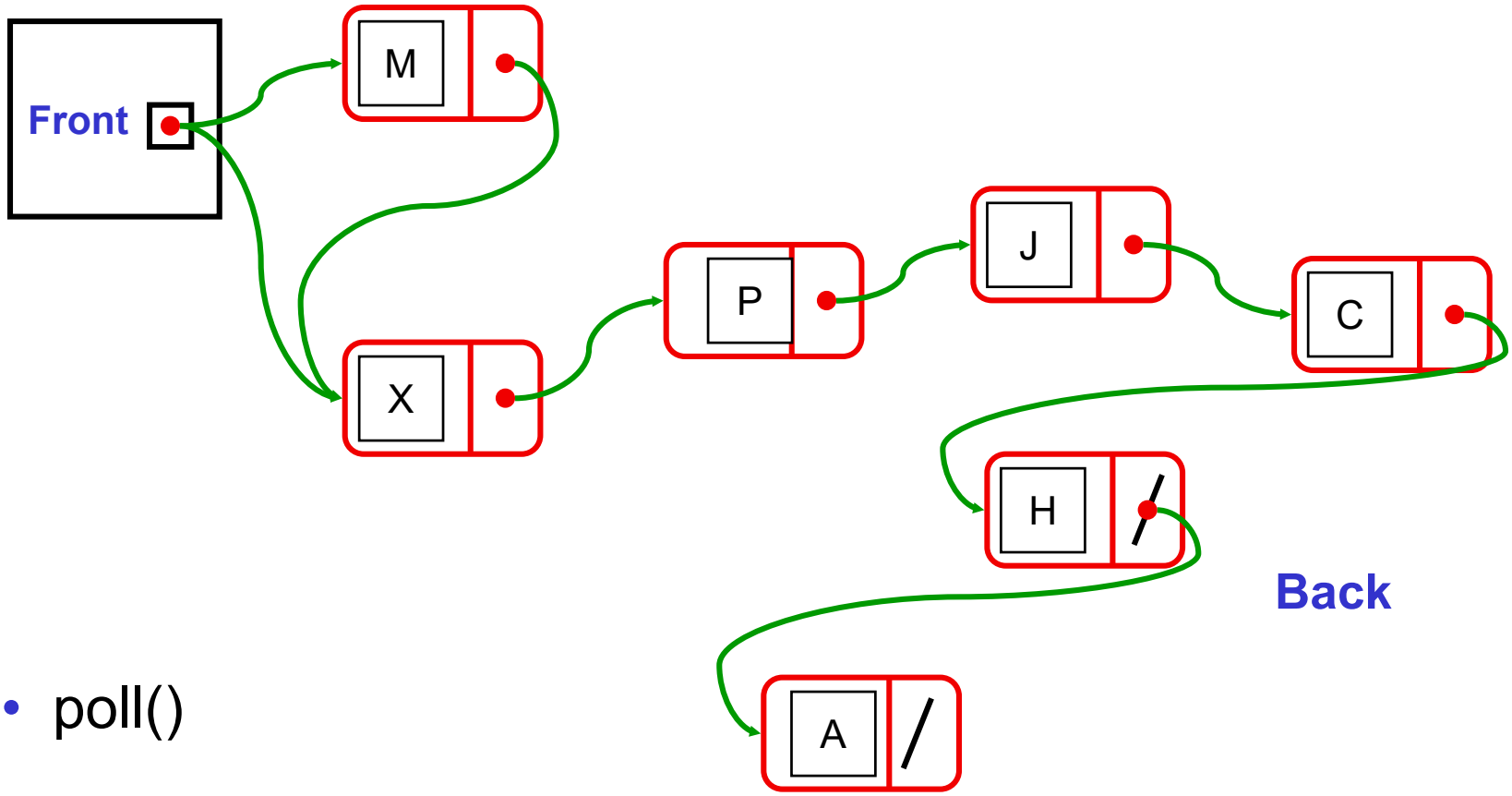


- offer("A")

- poll()

A Linked Queue #2

- Put the front of the Queue at the head of the list.

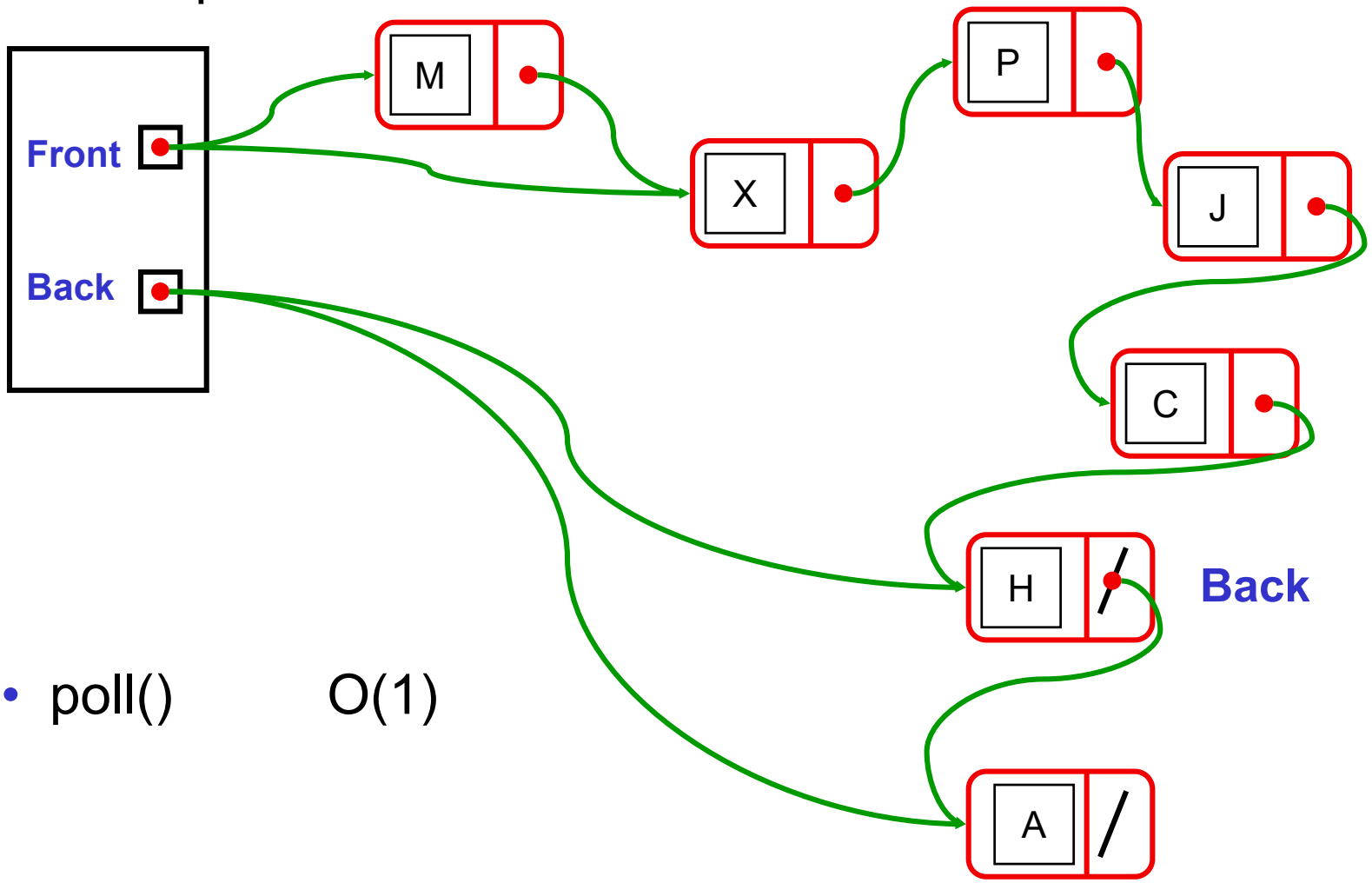


- poll()

- offer("A")

A Better Linked Queue

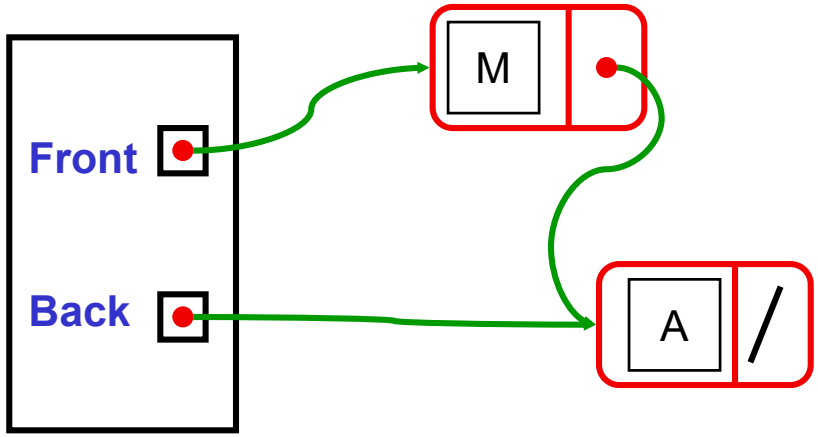
- Have pointers to both ends!



- poll() O(1)
- offer("A") O(1)

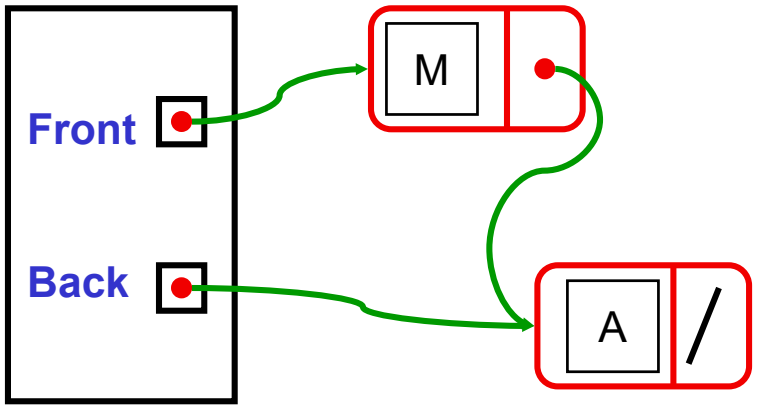
Implementing LinkedQueue

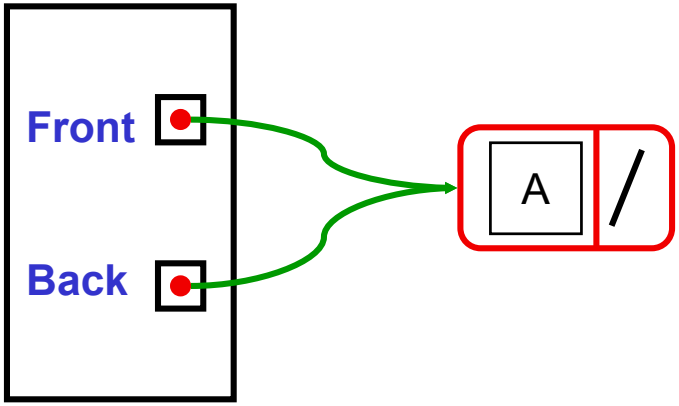
```
public class LinkedQueue <E> implements AbstractQueue <E> {

    private Node<E> front = null;
    private Node<E> back = null;

    public LinkedQueue(){  }

    public int size(){…

    public boolean isEmpty(){…

    public E peek(){…

    public E poll(){…

    public void offer(E item){…

    public Iterator <E> iterator(){
```

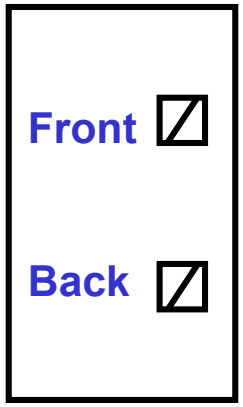


LinkedQueue

```
public boolean isEmpty(){
    return front==null;
}
public int size () {
   if (front == null)   return 0;
   else                 return front.size();
}
```



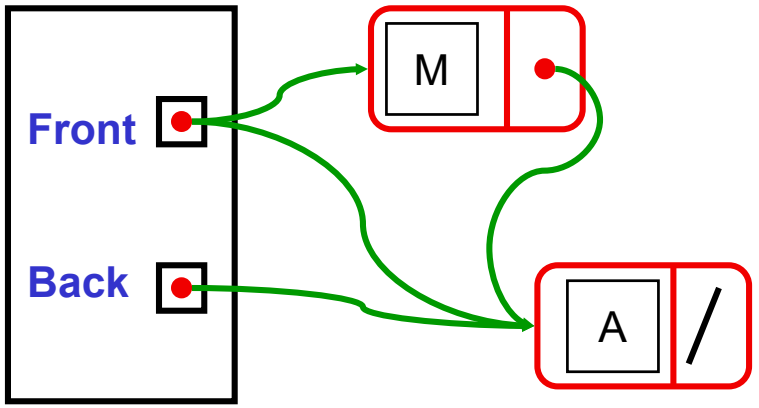




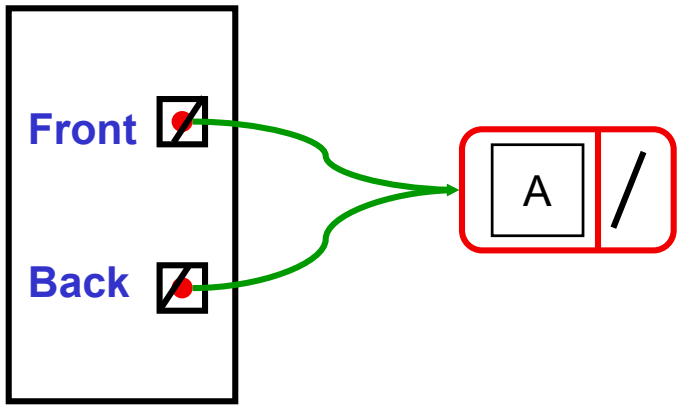
- Always three cases: 0 items, 1 item, >1 item

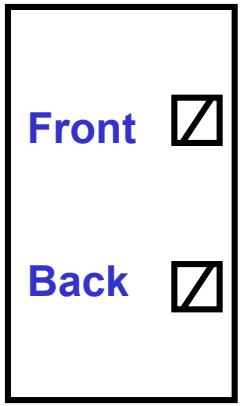
LinkedQueue

```
public E peek(){
   if (front==null)   return null;
   else               return front.value;
}
```



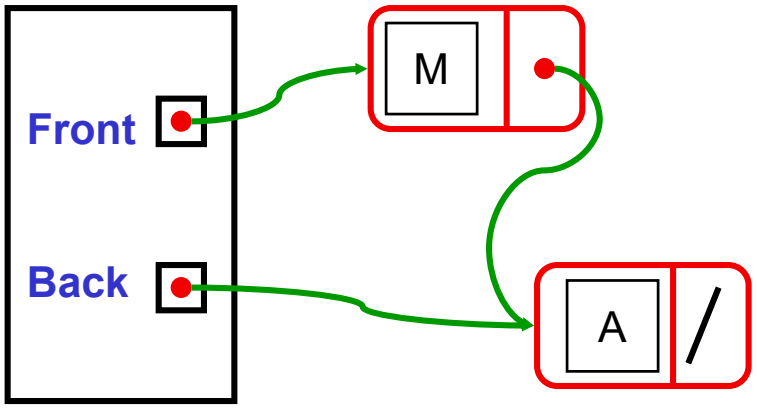

```
public E poll(){
   if (front==null) return null;
   E ans = front.value;
   front = front.next;
   if (front==null)  back = null;
   return ans;
}
```



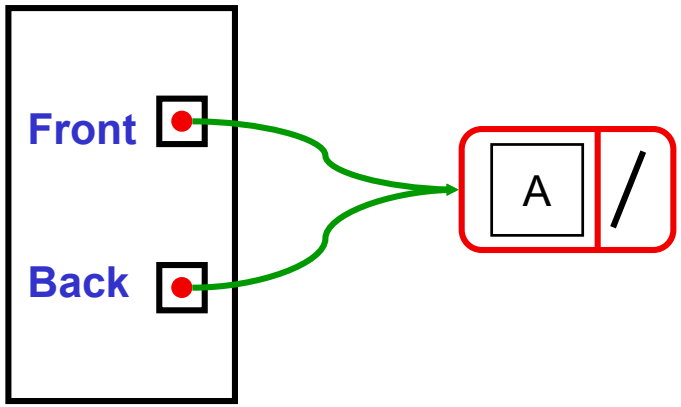


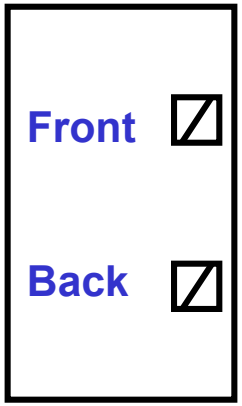
Exercise: work out the method body of 'offer'

```
public boolean offer(E item){
```



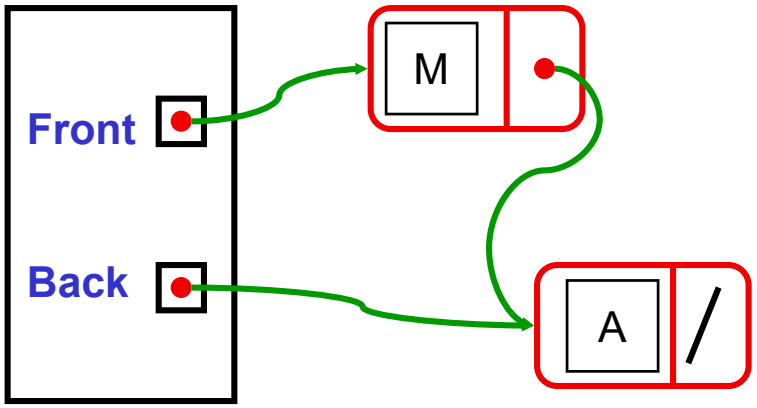

```
}
```

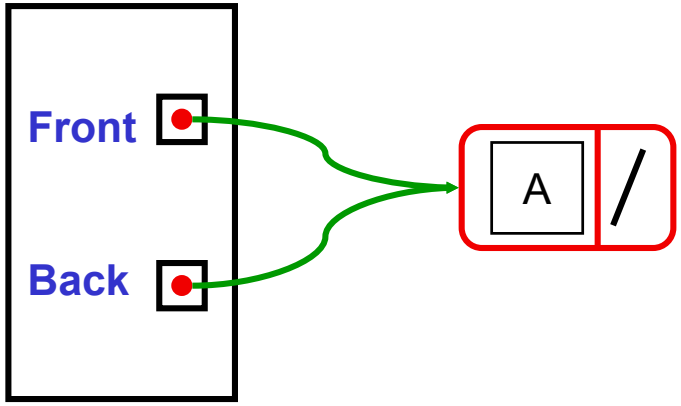


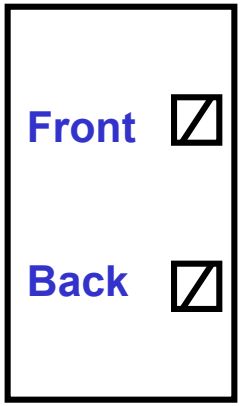


LinkedQueue

```
public boolean offer(E item){
    if (item == null) return false;
    if (front == null){
        back = new Node(item, null);
        front = back;
    }
    else {
        back.next = (new Node(item, null));
        back= back.next;
    }
    return true;
}
```







Linked Stack and Queue

- Uses a “header node”
  - contains link to head node, and maybe last node of linked list
- Important to choose the right end.
  - easy to add or remove from head of a linked list
  - hard to add or remove from the last node of a linked list
  - easy to add to last node of linked list if have pointer to tail
- Linked Stack and Queue:
  - all main operations are O(1)
- Can combine Stack and Queue
  - addFirst, addLast, removeFirst
  - also need removeLast to make a “Deque” (double-ended queue)
    ⇒ need doubly linked list (why?)
  - See the java “LinkedList” class.

Summary

- A Stack using a Linked List with a header
- A Queue using a Linked List with a header

Readings

- [Mar07] Read 3.6, 3.7, 6.2
- [Mar13] Read 3.6, 3.7, 6.2