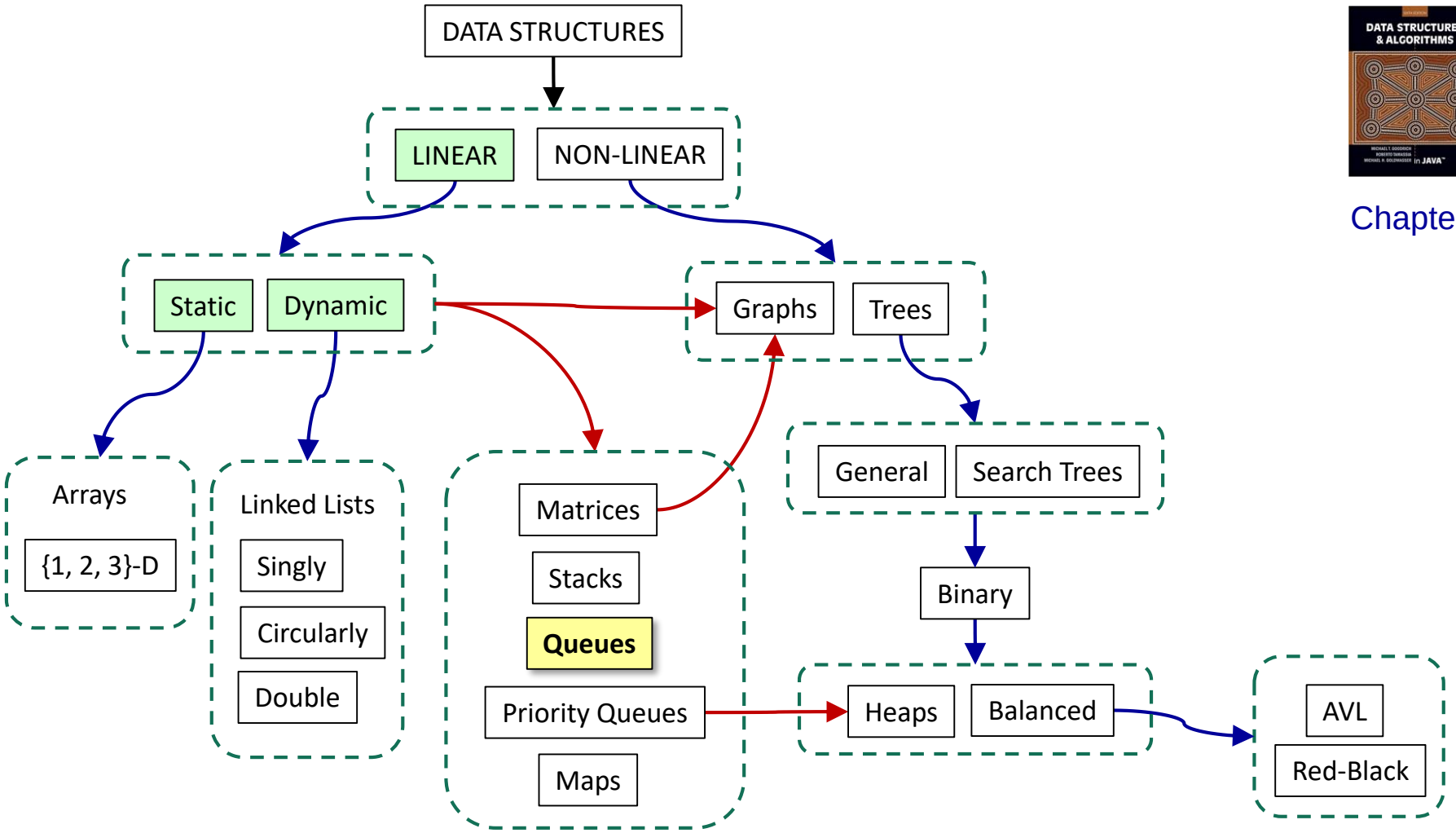
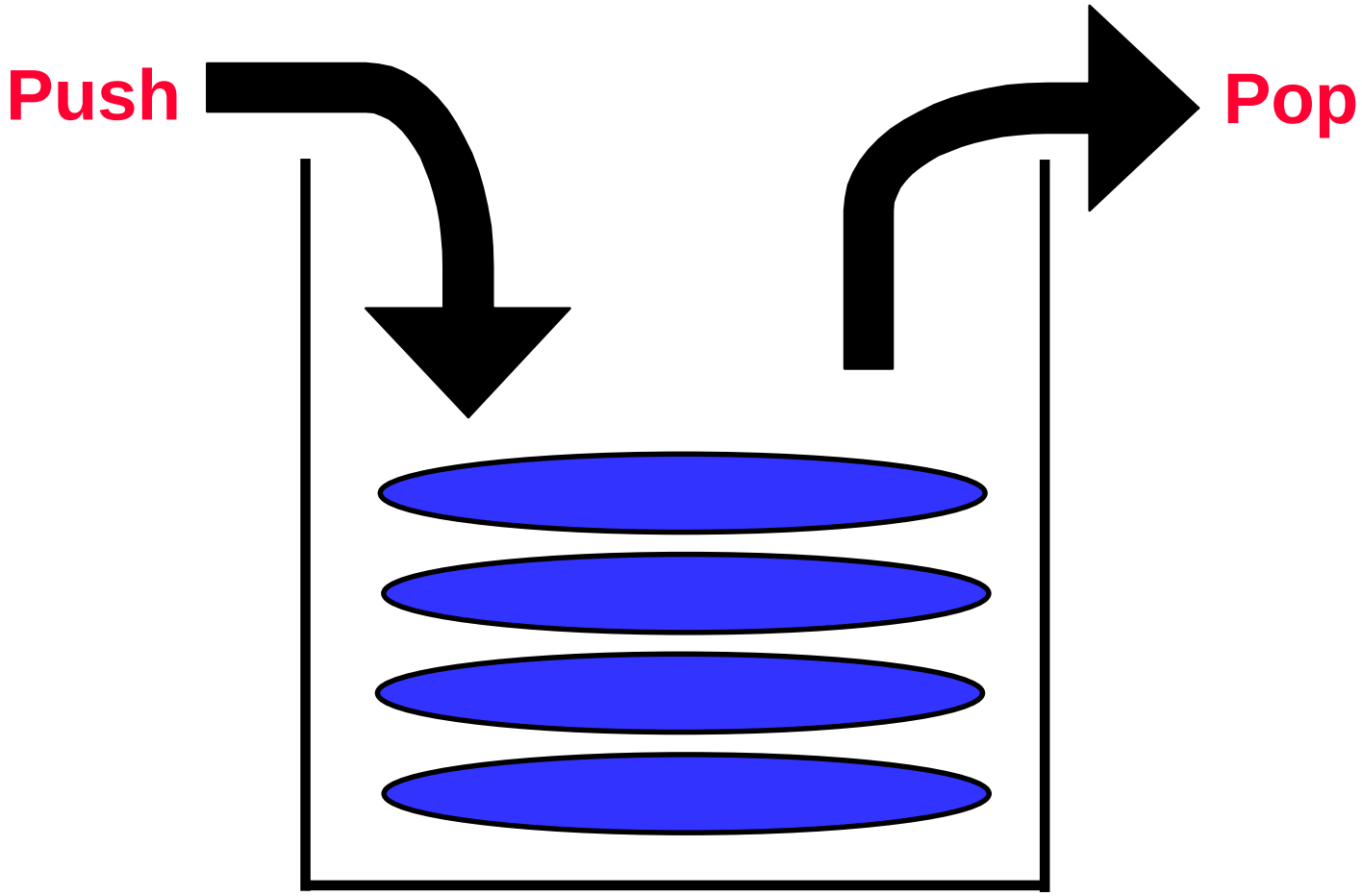

DATA STRUCTURES
LINEAR
NON-LINEAR
Static
Dynamic
Graphs
Trees
Arrays
{1, 2, 3}-D
Linked Lists
Singly
Circularly
Double
Matrices
Stacks
Queues
Priority Queues
Maps
General
Search Trees
Binary
Heaps
Balanced
AVL
Red-Black

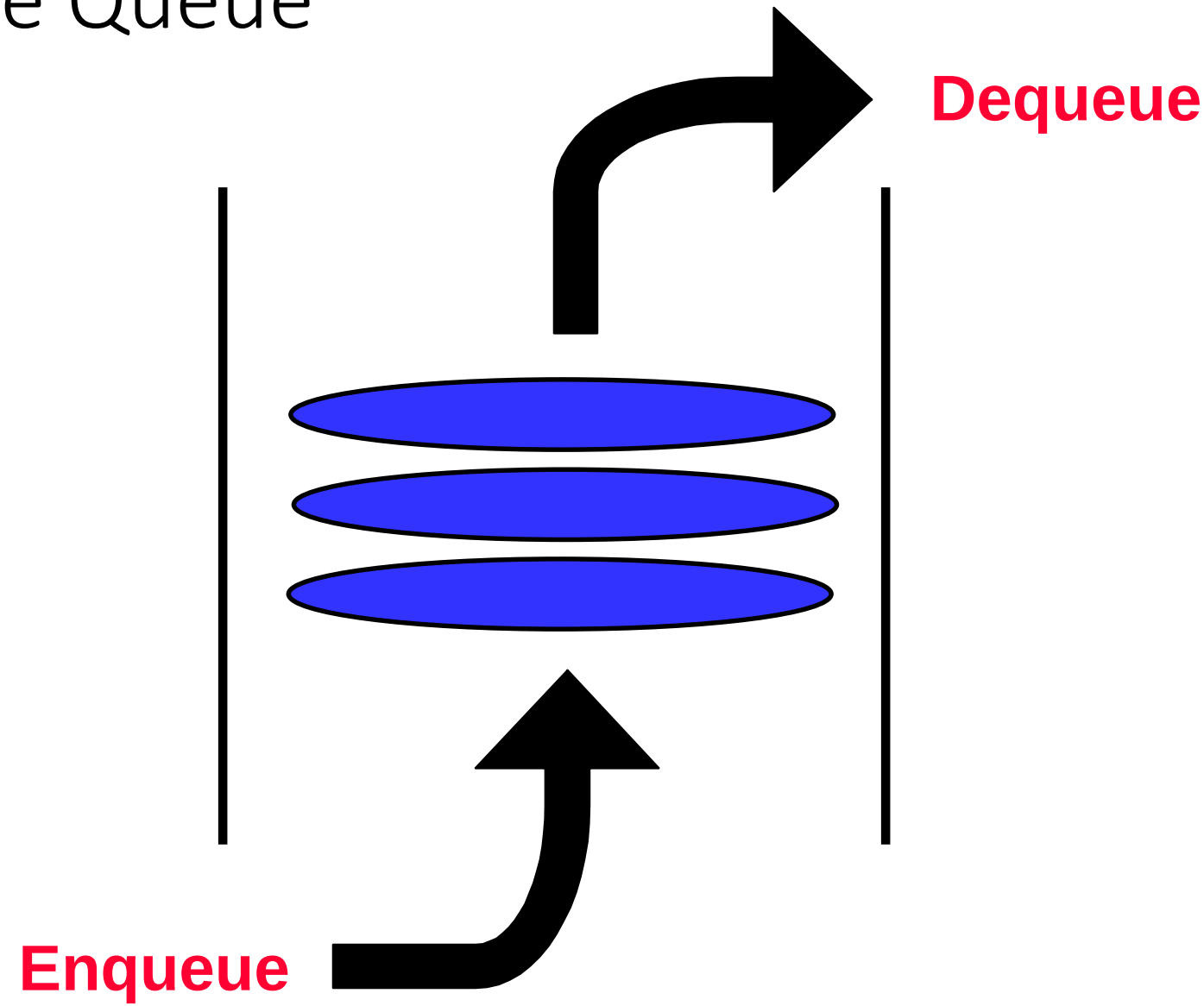
DATA STRUCTURES
& ALGORITHMS
in JAVA

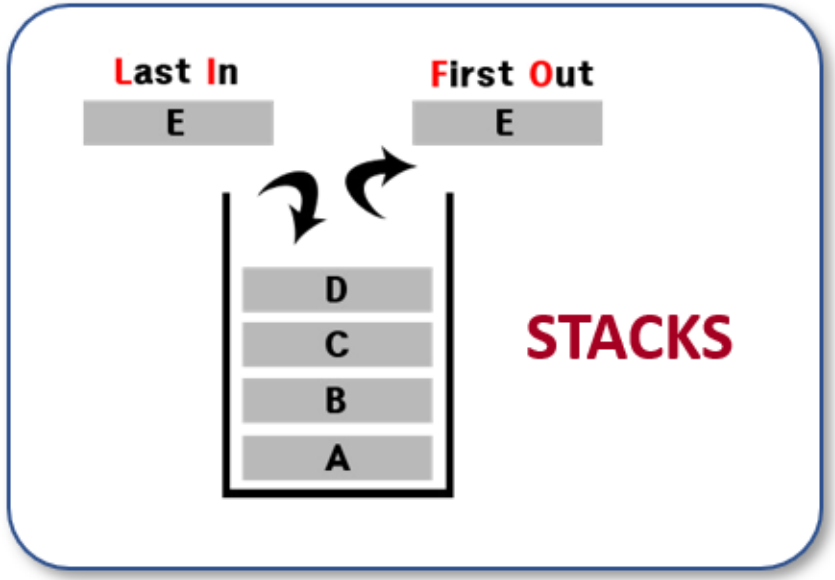
Chapter 6

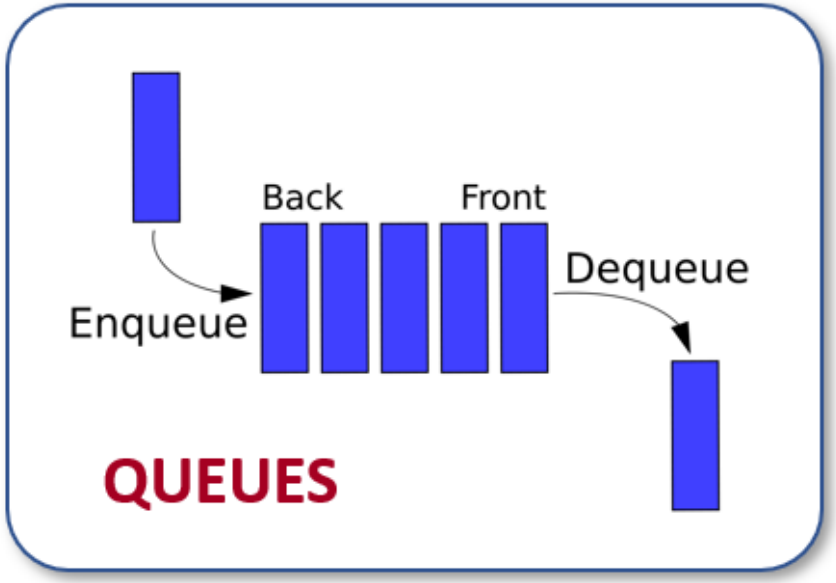
The Stack



The Queue





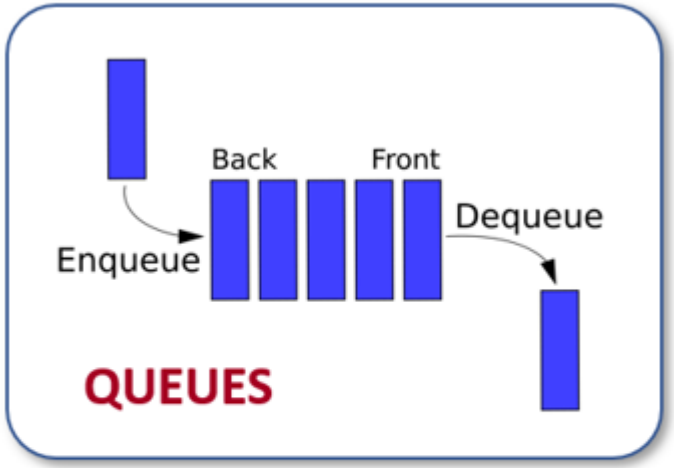


Last-In First-Out
LIFO

First-In First-Out
FIFO

Applications of Queues

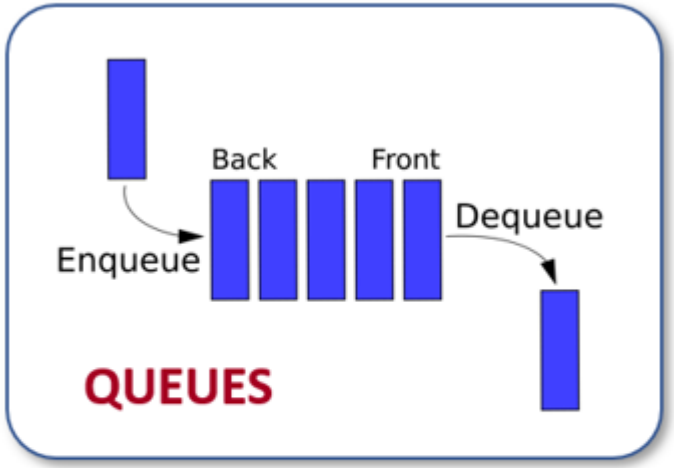
- **Waiting in line**
  - At the grocery store
  - At the movies
  - Printer queue
- **Ordering items**
  - Bills to pay
  - Making pizzas …



First-In First-Out
FIFO

Applications of Queues

- **Operating Systems**
  - jobs
  - print files
- **Simulation**
  - gas station
  - banks
  - reservation systems
  - elevator simulation ...

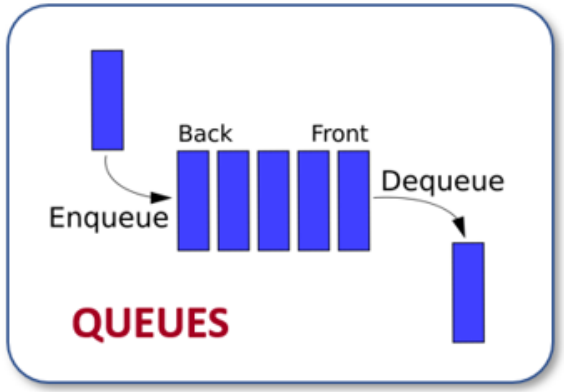


First-In First-Out
FIFO

The Queue ADT

- Insertions and deletions follow the **first-in first-out** scheme

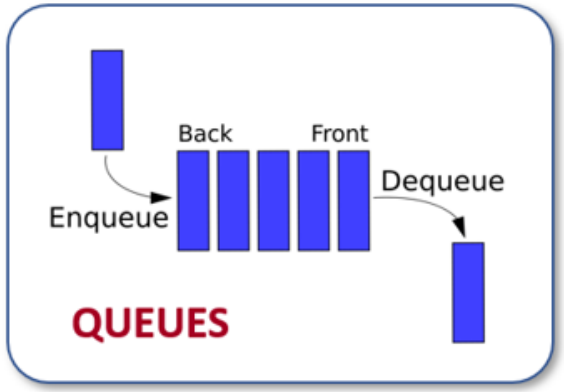
- Main queue operations:
  - enqueue(object): inserts an element at the end of the queue
  - object dequeue(): removes and returns the element at the front of the queue



First-In First-Out
FIFO

The Queue ADT

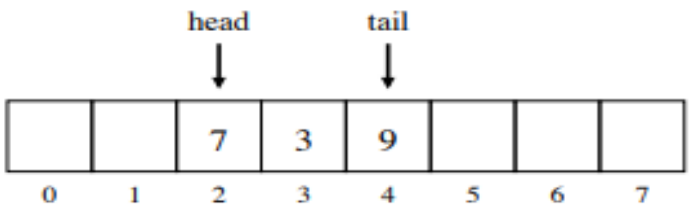
- Auxiliary queue operations:
  - object front(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean isEmpty(): indicates whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyQueueException



First-In First-Out
FIFO

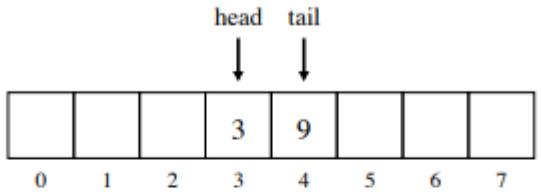
Array-based Queue

- May be implemented using an array
  - Use two variables to point to the beginning and end of the list
    - The "head" index is incremented after dequeuing, the "tail" index when enqueuing
    - E.g.

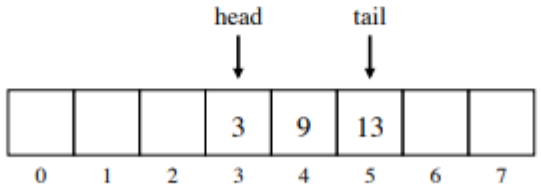


Array-based Queue

Dequeue



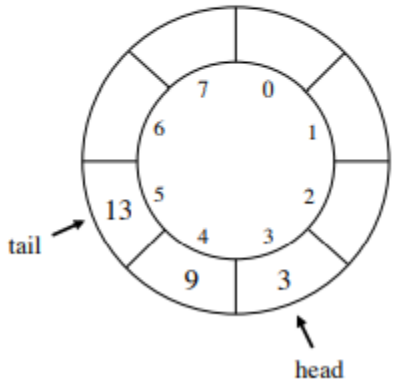
Enqueue 13



Array-based Queue

– Since the indices will eventually run off the end, the array is "wrapped around" to form a *circular array* (*ring buffer*)

- E.g.



Array-based Queue

- Modulus arithmetic must be used when incrementing the indices
  - i.e. Keep them in the range 0 to N-1, where N is the size of the array
- Head and tail are set to -1 to indicate an empty queue

Array-based Queue

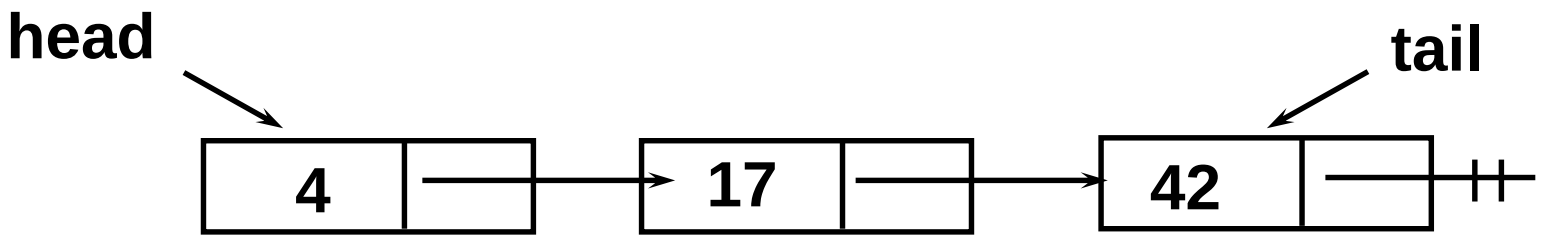
- To enqueue:
  - If the queue is empty
    - Set head and tail to 0
  - Else
    - Increment tail mod N
  - Set array[tail] to element value

Array-based Queue

- To dequeue:
  - Store array[head] in a temporary variable
  - If only one element in the queue (head == tail)
    - Set head and tail to -1 (indicates empty queue)
  - Else
    - Increment head mod N
  - Return the value in the temporary variable

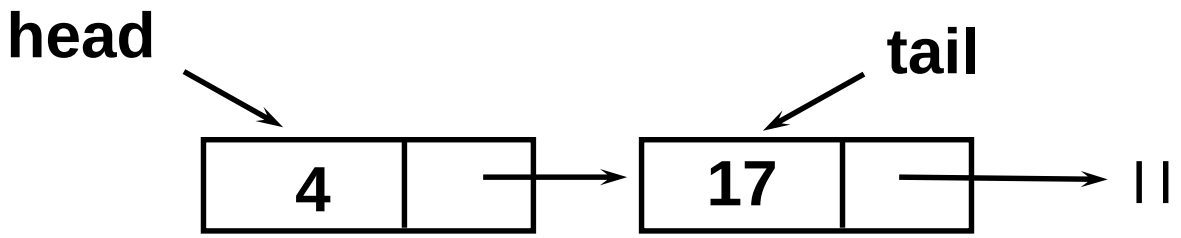
Queues:
Dynamic Implementation

- **A linked list with restricted set of operations to change its state: only modified by adding to one end and deleting from the other.**



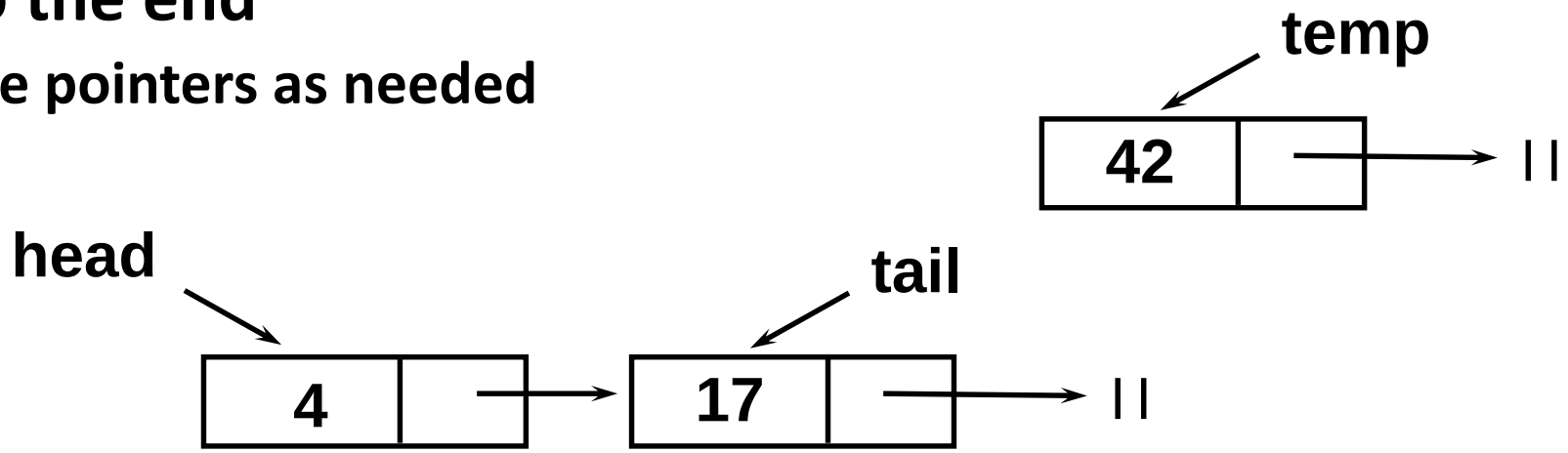
Enqueue

- **Create a new node with data**
- **Add it to the end**
  - **Update pointers as needed**



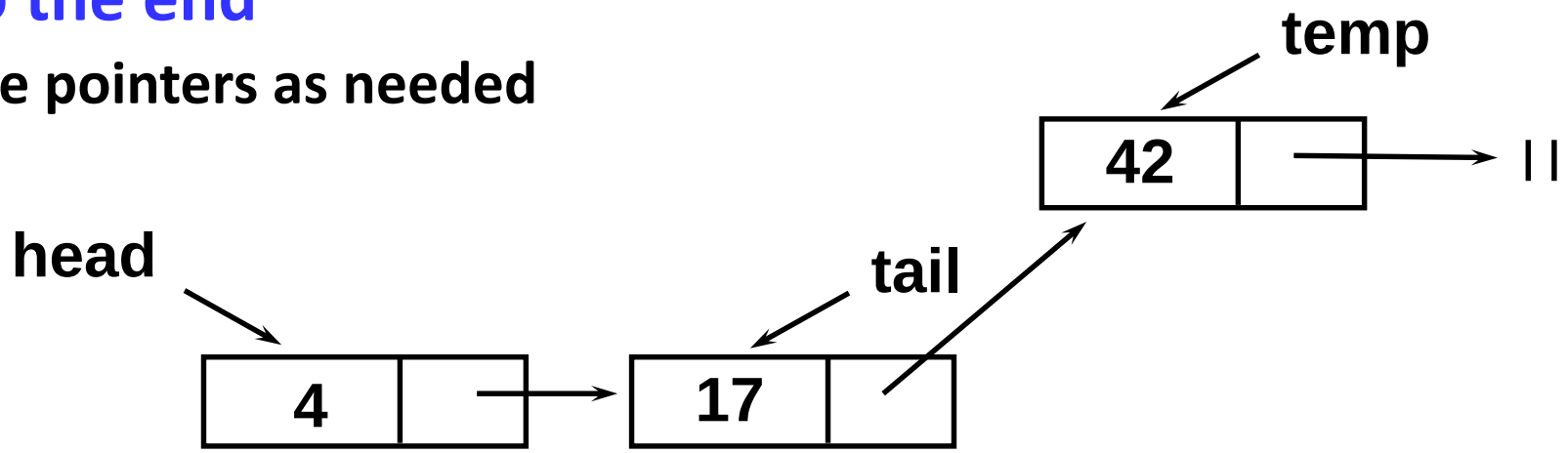
Enqueue

- **Create a new node with data**
- **Add it to the end**
  - **Update pointers as needed**



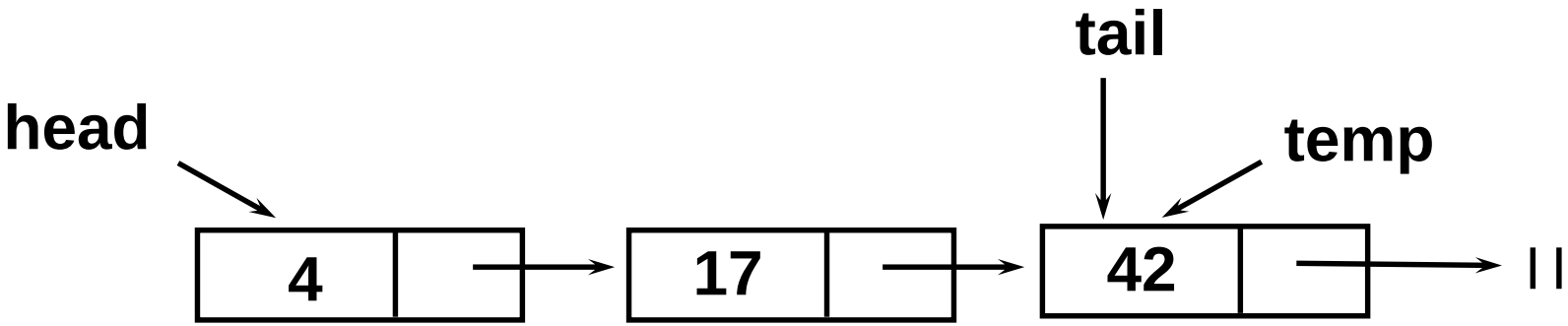
Enqueue

- **Create a new node with data**
- **Add it to the end**
  - **Update pointers as needed**



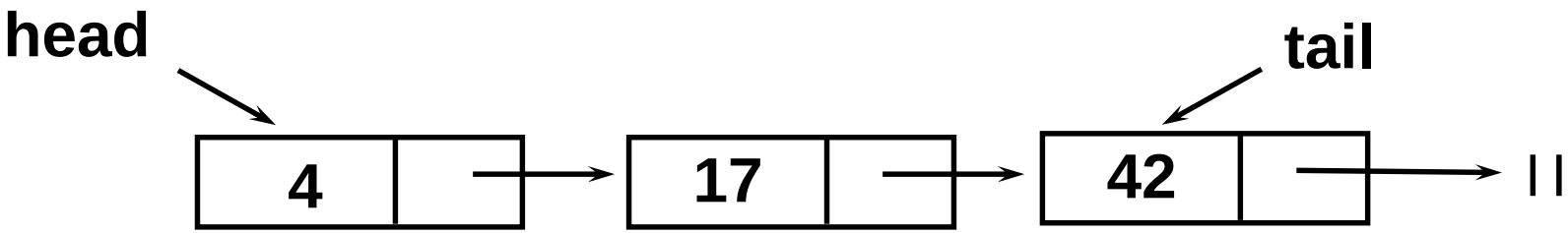
Enqueue

- **Create a new node with data**
- **Add it to the end**
  - **Update pointers as needed**



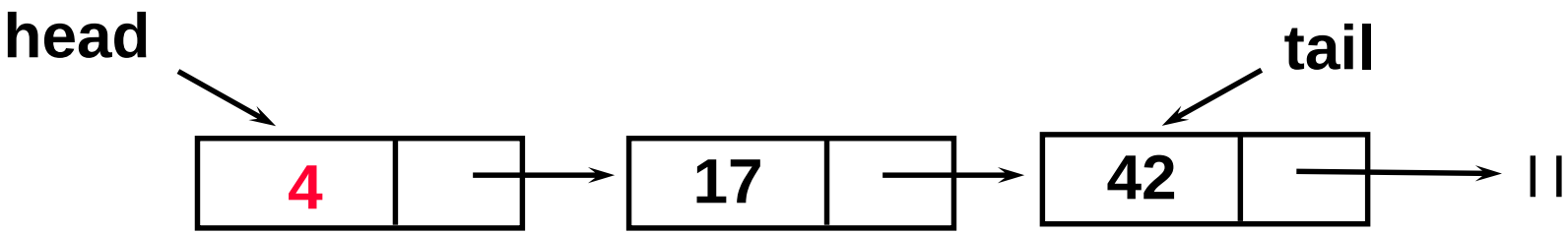
Dequeue

- **Capture the first value (to return)**
- **Remove the first node (move head to next)**



Dequeue

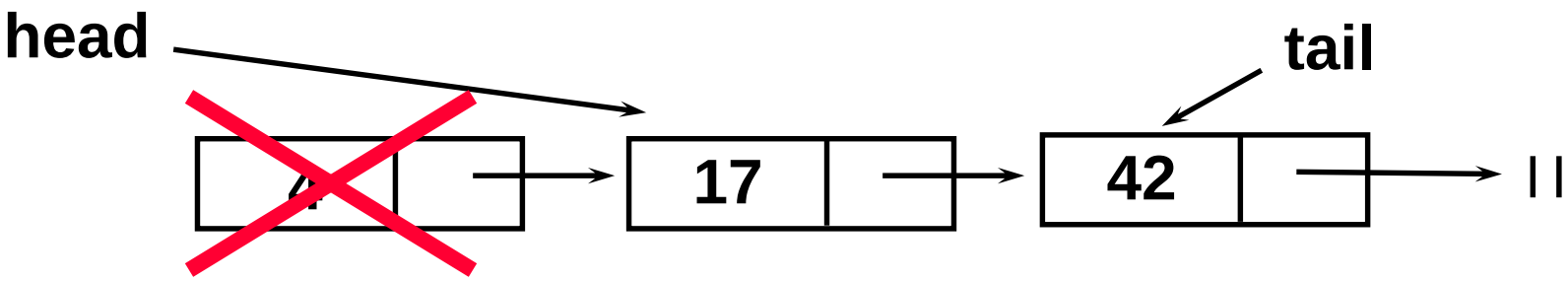
- **Capture the first value (to return)**
- **Remove the first node (move head to next)**




```
value = 4
```

Dequeue

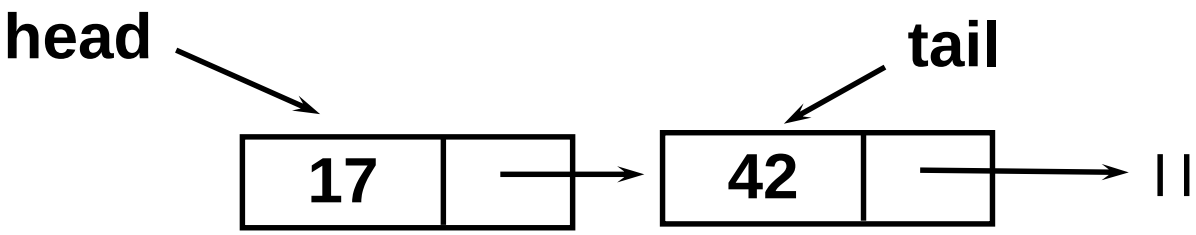
- **Capture the first value (to return)**
- **Remove the first node (move head to next)**




```
value = 4
```

Dequeue

- **Capture the first value (to return)**
- **Remove the first node (move head to next)**




```
value (4) is returned
```

Priority Queue

- Are linear data structures that store prioritized elements
- Each element has an associated priority
  - Usually a numeric value, where the smallest value means the highest priority
  - Stored as a key in the node for an element
- When dequeuing, one always removes the element with the highest priority (lowest key) from the list

Priority Queue

- May be implemented using an unsorted linked list
  - New elements are always added to the tail
    - i.e. Do the standard enqueue operation
    - Is O(1)
  - To dequeue the highest priority element, one must search the entire list for the lowest key
  - Is O(n) in the best and worst cases

Priority Queue

- May be implemented using a sorted linked list
  - New elements are inserted into the list in their proper position using the key
    - Is O(n) in the worst case
  - To dequeue the highest priority element, simply remove the first element
    - Is O(1)

Summary: Queues

- **Allow us to model "first-in, first-out" (FIFO) behavior**
- **Can be implemented using different data types**
- **Behavior is important (and defines a Queue)**
  - **Enqueue to end**
  - **Dequeue from front**
  - **(or vice-versa – just do at opposite ends)**

- **Priority Queue:**
  - Are linear data structures that store prioritized elements