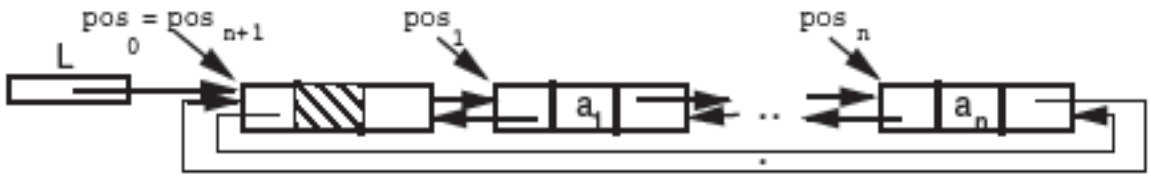
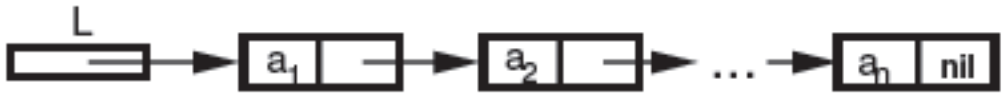
Liste

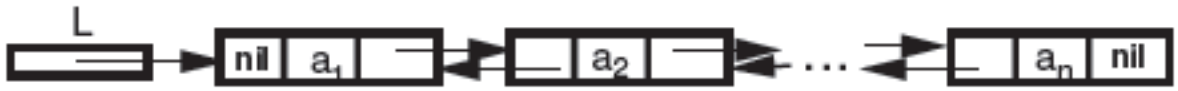
- **Liste (List, Linked List)**
  - **Una sequenza di nodi, contenenti dati arbitrari e 1-2 puntatori all'elemento successivo e/o precedente**
  - **Contiguità nella lista ⇏ contiguità nella memoria**
- **Costo operazioni**
  - **Tutte le operazioni hanno costo $O(1)$**
- **Realizzazione possibili**
  - ***Bidirezionale*** **/ monodirezionale**
  - ***Con sentinella*** **/ senza sentinella**
  - ***Circolare*** **/ non circolare**



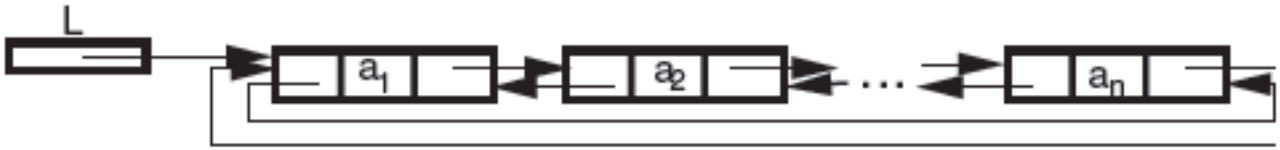
Linked List - Altri esempi



monodirezionale



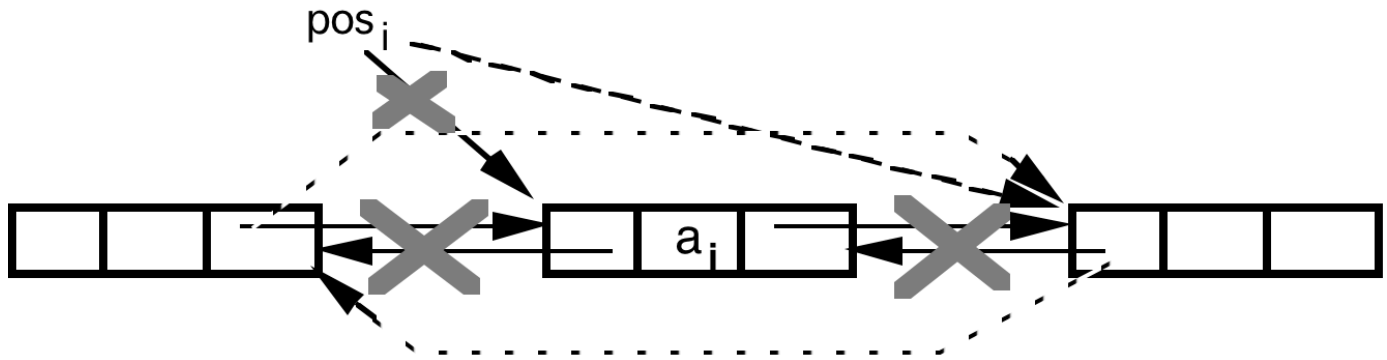
bidirezionale



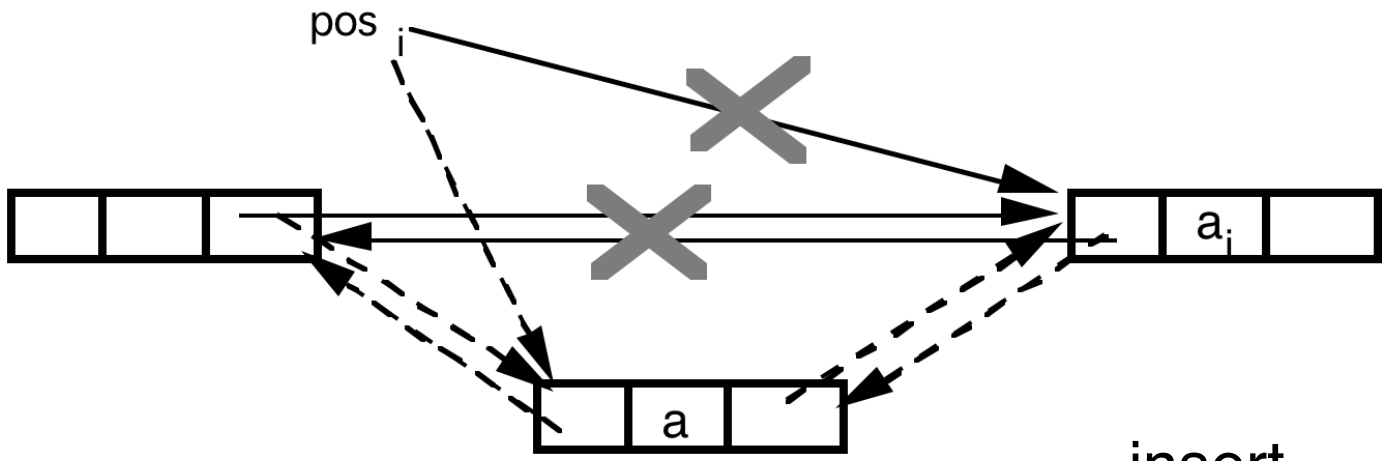
bidirezionale circolare

monodirezionale con sentinella

List – cancellazione e inserimento



remove



insert

List (Java) - bidirezionale senza sentinella

```
class Pos
{
Pos succ;
Pos pred;
Object v;

Pos(Object v) {
   succ = pred = null;
   this.v = v;
}
```

List (Java) - bidirezionale senza sentinella

```
public class List {

/** First element of the list */
private Pos head;

/** Last element of the list */
private Pos tail;

public List() {
  head = tail = null;
}

public Pos head()                    { return head; }
public Pos tail()                    { return tail; }
public Pos next(Pos pos)             { return (pos != null ? pos.succ : null); }
public Pos prev(Pos pos)             { return (pos != null ? pos.pred : null); }
public boolean finished(Pos pos)     { return pos == null; }
public boolean isEmpty()             { return head == null; }
public Object read(Pos p)            { return p.v; }
public void write(Pos p, Object v)   { p.v = v; }
```

List (Java) - bidirezionale senza sentinella

```
  public Pos insert(Pos pos, Object v) {
Pos t = new Pos(v);
if (head == null) {
  // Insert in a emtpy list
   head = tail = t;
} else if (pos == null) {
  // Insert at the end
  t.pred = tail;
  tail.succ = t;
  tail = t;
} else {
  // Insert in front of an existing position
   t.pred = pos.pred;
   if (t.pred != null)
     t.pred.succ = t;
   else
      head = t;
   t.succ = pos;
   pos.pred = t;
}
return t;
}
```

List (Java) - bidirezionale senza sentinella

```
  public void remove(Pos pos) {
  if (pos.pred == null)
    head = pos.succ;
  else
    pos.pred.succ = pos.succ;
  if (pos.succ == null)
    tail = pos.pred;
  else
    pos.succ.pred = pos.pred;
}
```

Liste - Realizzazione con vettori

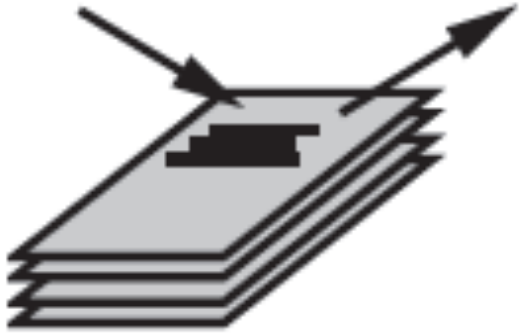
- E' possibile realizzare una lista con vettori
  - Posizione ≡ indice nel vettore
  - Le operazioni **insert**() e **remove**() hanno costo *lineare in* (*n*)
  - Tutte le altre operazioni hanno costo *O*(1)
- Problema
  - Spesso non si conosce a priori quanta memoria serve per memorizzare la lista
  - Se ne alloca una certa quantità, per poi accorgersi che non è sufficiente.
- Soluzione
  - Si alloca un vettore di dimensione maggiore, si ricopia il contenuto del vecchio vettore nel nuovo e si rilascia il vecchio vettore
  - Esempi: `java.util.Vector`, `java.util.ArrayList`

Vettori dinamici: espansione

```
private Object[] buffer = new Object[INITSIZE];
// Utilizzato in ArrayList()
private void doubleStorage() {
  Object[] newb = new Object[2*buffer.length];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}
// Utilizzabile in Vector()
private Object[] buffer = new Object[INITSIZE];
private void incrementStorage() {
  Object[] newb = new Object[buffer.length+INCREMENT];
  System.arraycopy(buffer,0, newb,0, buffer.length);
  buffer = newb;
}
```

Stack

- **Una pila (stack)**
  - **è un insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: "*quello che per meno tempo è rimasto nell'insieme*"**
  - **politica "*last in, first out*" (LIFO)**



pratiche burocratiche

- **Operazioni previste (tutte realizzabili in *tempo costante*)**

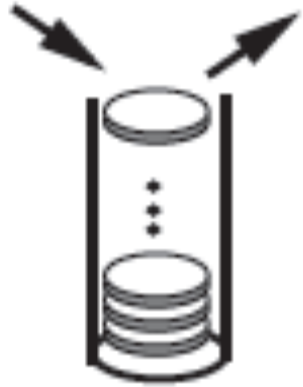
STACK

% Restituisce **true** se la pila è vuota
integer isEmpty()

% Inserisce v in cima alla pila
push(ITEM v)

% Estrae l'elemento in cima alla pila e lo restituisce al chiamante
ITEM pop()

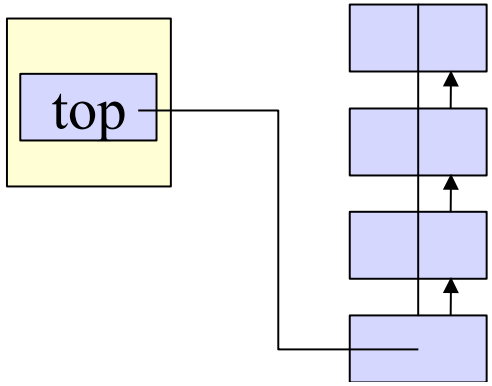
% Legge l'elemento in cima alla pila
ITEM top()

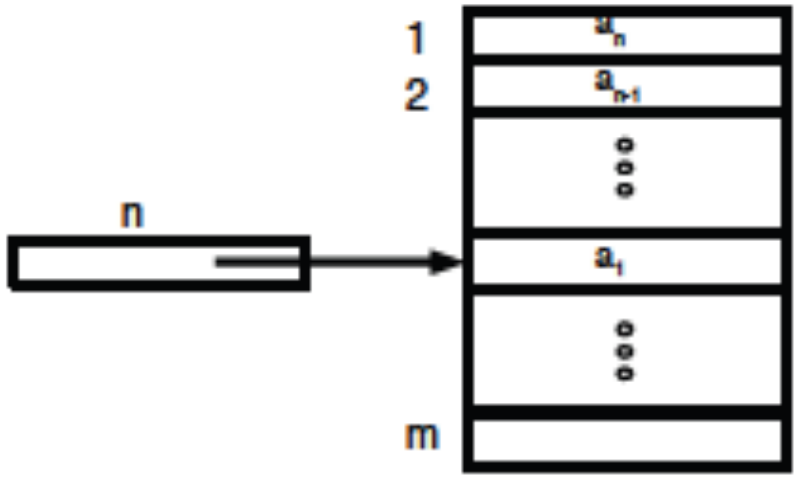


tubetto di pastiglie

Stack

- **Possibili implementazioni**
  - **Tramite *liste bidirezionali***
    - **puntatore all'elemento top**
  - **Tramite *vettore***
    - **dimensione limitata, overhead più basso**





Stack - pseudocodice

```
STACK
ITEM[] A                    % Elementi
integer n                   % Cursore
integer m                   % Dim. max

STACK Stack(integer dim)
    STACK t ← new STACK
    t.A ← new integer[1 ... dim]
    t.m ← dim
    t.n ← 0
    return t

ITEM top()
    precondition: n > 0
    return A[n]

boolean isEmpty()
    return n = 0

ITEM pop()
    precondition: n > 0
    ITEM t ← A[n]
    n ← n − 1
    return t

push(ITEM v)
    precondition: n < m
    n ← n + 1
    A[n] ← v
```

Stack: implementazione tramite vettori (Java)

```
public class VectorStack implements Stack
{

/** Vector containing the elements */
private Object[] A;

/** Number of elements in the stack */
private int n;

public VectorStack(int dim) {
   n = 0;
   A = new Object[dim];
}

public boolean isEmpty() {
   return n==0;
}
```

Stack: implementazione tramite vettori (Java)

```
public Object top() {
  if (n == 0)
    throw new IllegalStateException("Stack is empty");
  return A[n-1];
}

public Object pop() {
  if (n == 0)
    throw new IllegalStateException("Stack is empty");
  return A[--n];
}

public void push(Object o) {
  if (n == A.length)
    throw new IllegalStateException("Stack is full");
  A[n++] = o;
}

}
```

Queue

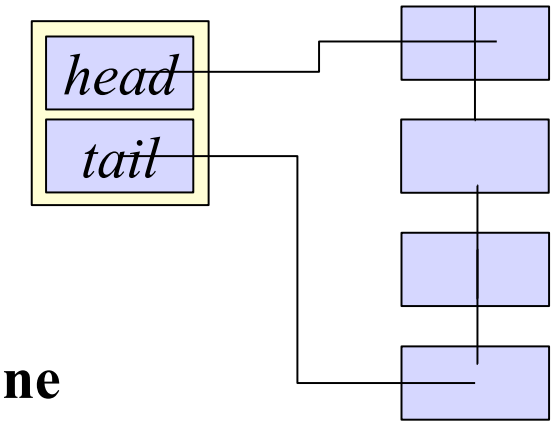
- **Una coda (queue)**
  - **è un insieme dinamico in cui l'elemento rimosso dall'operazione di cancellazione è predeterminato: "*quello che per più tempo è rimasto nell'insieme*"**
  - **politica "*first in, first out*" (*FIFO*)**

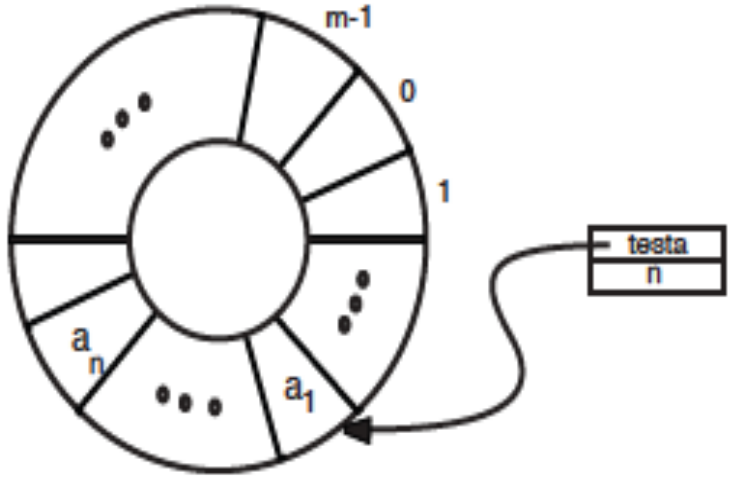
- **Operazioni previste (tutte realizzabili in $O(1)$)**

QUEUE
% Restituisce **true** se la coda è vuota **integer** isEmpty()
% Inserisce v in fondo alla coda enqueue(ITEM v)
% Estrae l'elemento in testa alla coda e lo restituisce al chiamante ITEM dequeue()
% Legge l'elemento in testa alla coda ITEM top()

Queue

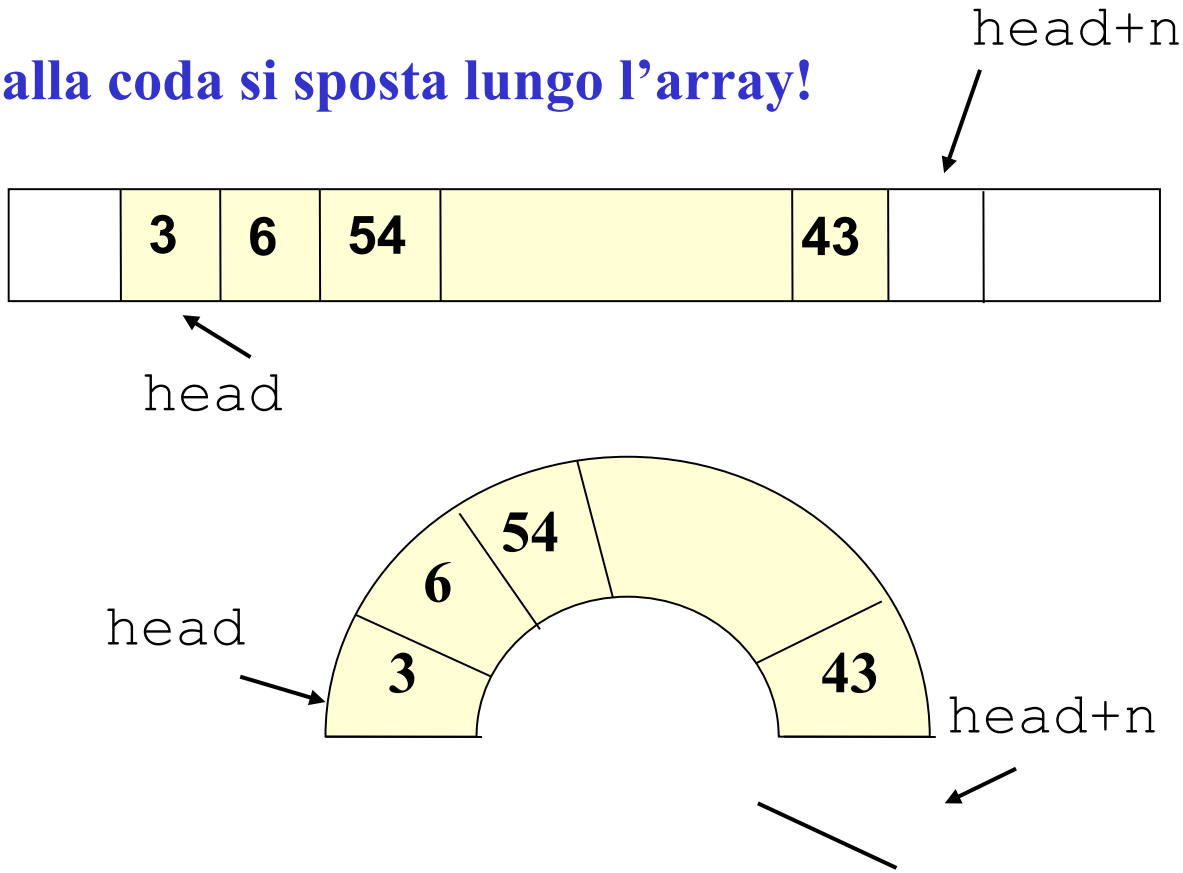
- **Possibili utilizzi**
  - **Nei sistemi operativi, i processi in attesa di utilizzare una risorsa vengono gestiti tramite una coda**
  - **La politica FIFO è fair**
- **Possibili implementazioni**
  - **Tramite liste monodirezionali**
    - **puntatore *head* (inizio della coda), per estrazione**
    - **puntatore *tail* (fine della coda), per inserimento**
  - **Tramite array circolari**
    - **dimensione limitata, overhead più basso**



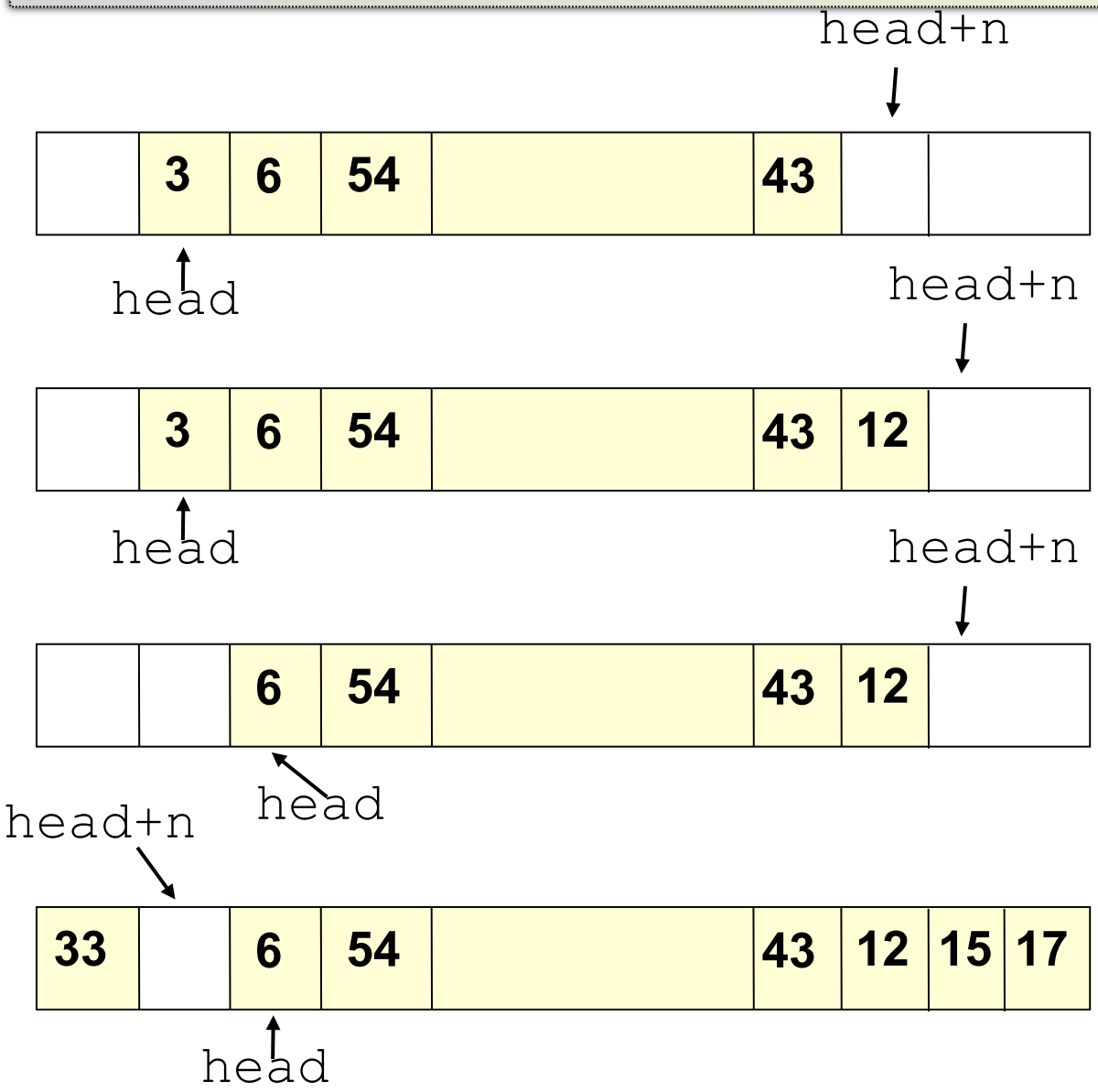


Coda: Realizzazione tramite vettore circolare

- La “finestra” dell’array occupata dalla coda si sposta lungo l’array!
- Dettagli implementativi
  - L'array circolare può essere implementato con un'operazione di modulo
  - Bisogna prestare attenzione ai problemi di overflow (buffer pieno)



Coda: Realizzazione tramite vettore circolare



Coda - pseudocodice

QUEUE

```
ITEM[] A                          % Elementi
integer n                         % Dim. attuale
integer testa                     % Testa
integer m                         % Dim. max

QUEUE Queue(integer dim)
    QUEUE t ← new QUEUE
    t.A ← new integer[0 ... dim − 1]
    t.m ← dim
    t.testa ← 0
    t.n ← 0
    return t

ITEM top()
    precondition: n > 0
    return A[testa]

boolean isEmpty()
    return n = 0

ITEM dequeue()
    precondition: n > 0
    ITEM t ← A[testa]
    testa ← (testa + 1) mod m
    n ← n − 1
    return t

enqueue(ITEM v)
    precondition: n < m
    A[(testa + n) mod m] ← v
    n ← n + 1
```

Queue: Implementazione tramite array circolari (Java)

```
public class VectorQueue implements Queue
{

/** Vector containing the elements */
private Object[] A;

/** Number of elements in the queue */
private int n;

/** Top element of the queue */
private int head;

public VectorQueue(int dim) {
  n = 0;
  head = 0;
  A = new Object[dim];
}

public boolean isEmpty()
{
   return n==0;
}
```

Queue: Implementazione tramite array circolari (Java)

```
public Object top() {
  if (n == 0)
   throw new IllegalStateException("Queue is empty");
  return A[head];
}

public Object dequeue()
{
  if (n == 0)
   throw new IllegalStateException("Queue is empty");
  Object t = A[head];
  head = (head+1) % A.length;
  n = n-1;
  return t;
}

public void enqueue(Object v)
{
  if (n == A.length)
    throw new IllegalStateException("Queue is full");
   A[(head+n) % A.length] = v;
   n = n+1;
}
```