Review

- Data-structure
- Dynamic data-structures
- Abstract Data Types (ADTs)
- Collections (adding, removing, getting, ...): Dynamic ADTs
- Linked-lists

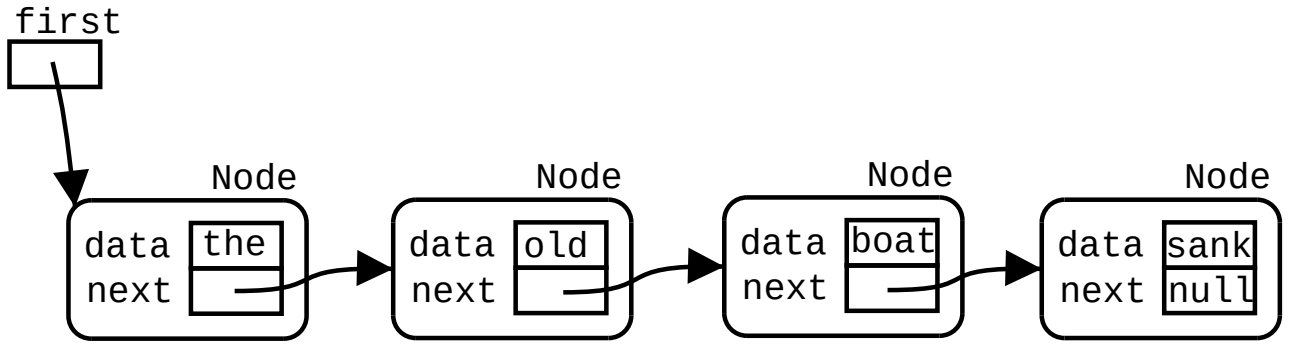
Review

```
class Node {
  Object data;
  Node next;
}

// Heterogeneous
class LinkedList {
  Node first;

  LinkedList() { first = null; }
  void add(Object o) { ... }
  int lenght() { ... }
  Object element_at(int index) { ... }
}
```

Review


first
Node
data
next
the
Node
data
next
old
Node
data
next
boat
Node
data
next
sank
null

Linked Lists

```
class Movie {
  private String title, director;
  // ...
}

class MovieNode {
  Movie     data;
  MovieNode next;
  public MovieNode(Movie m, MovieNode n)
  {
    data = m;
    next = n;
  }
  public Movie    get_movie() { return data; }
  public MoviNode get_next()  { return next; }
  public void set_movie(Movie m)    { data = m; }
  public void set_next(MovieNode n) { next = n; }
}
```

Linked Lists

```
class MovieList {
  private MovieNode first;

  public MovieList() { first = null; }

  public void add(Movie m)
  {
    MovieNode new_node = new MovieNode(m, first);
    first = new_node;
  }
}
```

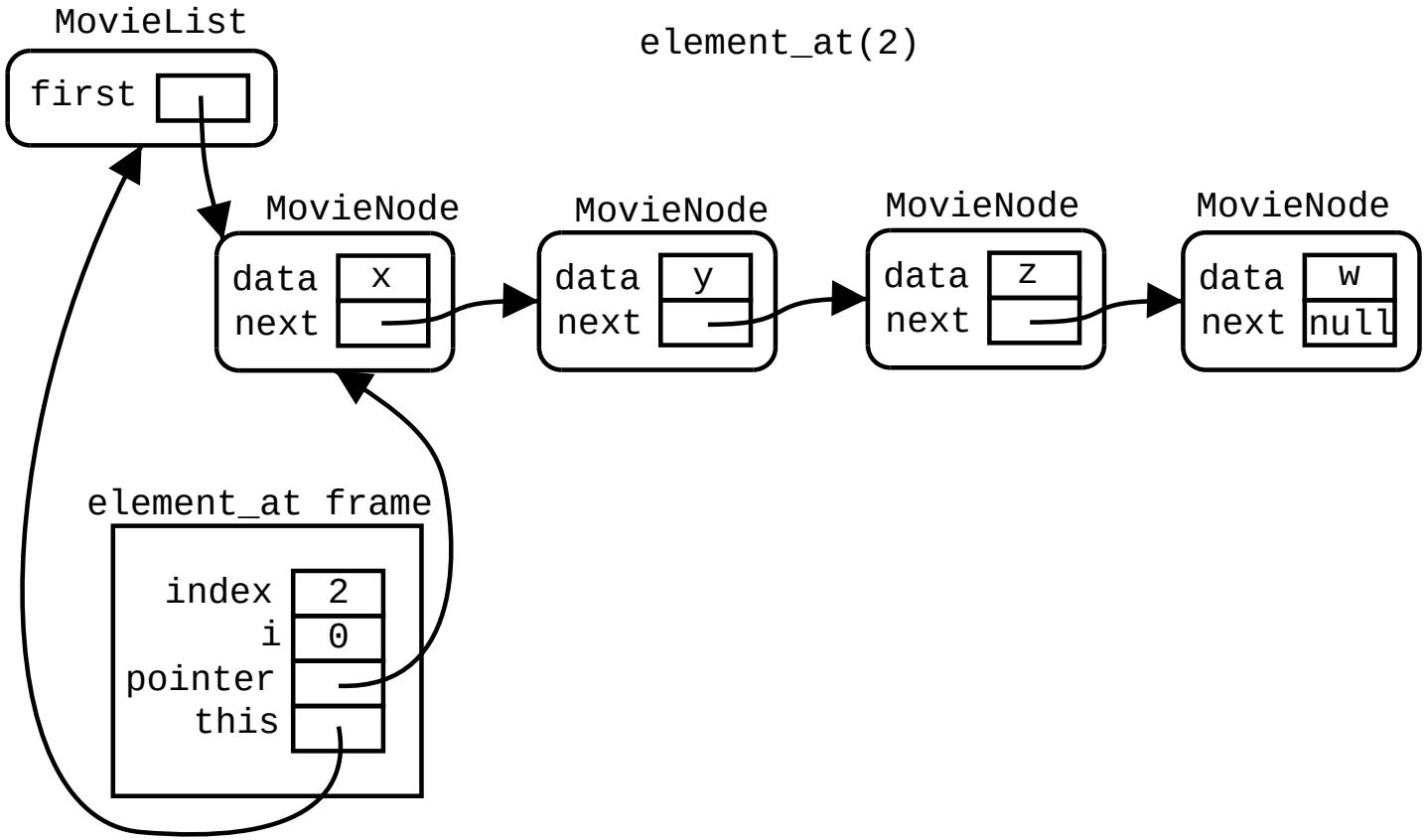
Linked Lists

```
class MovieList {
  private MovieNode first;
  //...
  public Movie element_at(int index)
  throws IndexOutOfBoundsException
  {
    if (index < 0)
      throw new IndexOutOfBoundsException();
    int i = 0;
    MovieNode pointer = first;
    while (pointer != null && i < index) {
      pointer = pointer.get_next();
      i++;
    }
    if (pointer == null)
      throw new IndexOutOfBoundsException();
    return pointer.get_movie();
  }
}
```

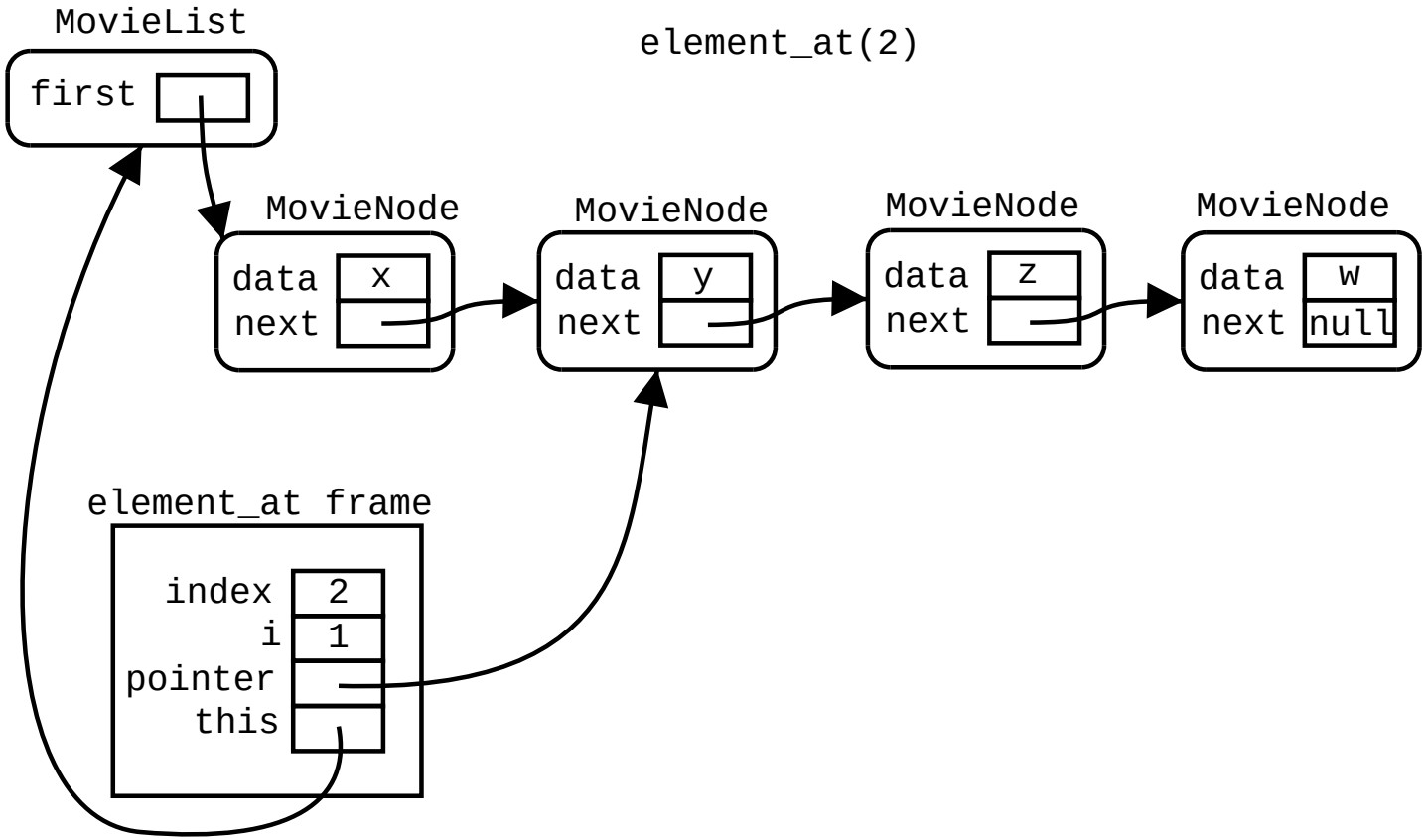
Linked Lists

```
class Test {
  public static void main(String[] args)
  {
    MovieList l = new MovieList();
    Movie w = new Movie(“abc”,“def”);
    Movie x = new Movie(“bca”,“efd”);
    Movie z = new Movie(“cba”,“fef”);
    Movie y = new Movie(“xxx”,“yyy”);
    l.add(w);
    l.add(z);
    l.add(y);
    l.add(x);
    Movie m = l.element_at(2);
  }
}
```

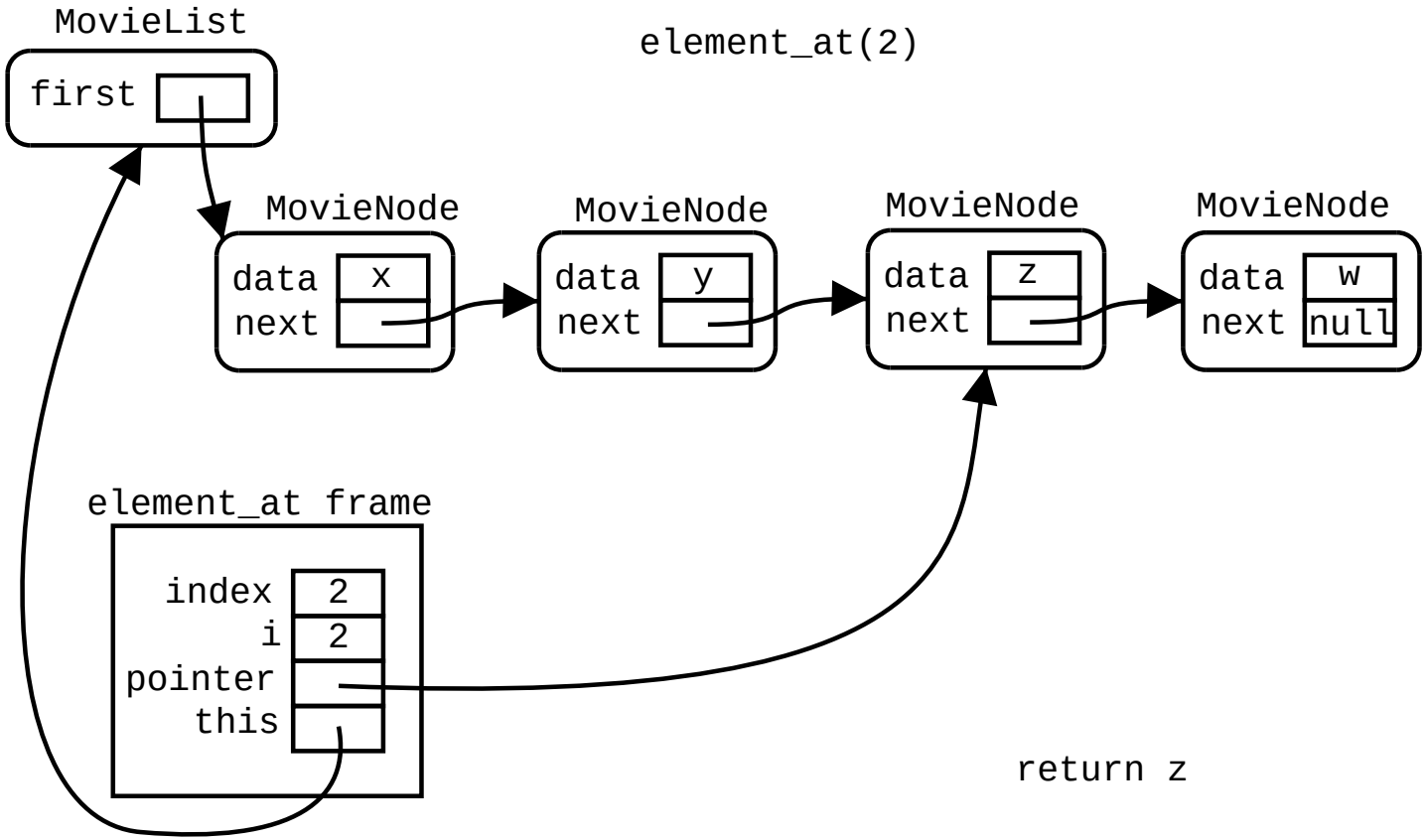
Linked Lists



Linked Lists



Linked Lists



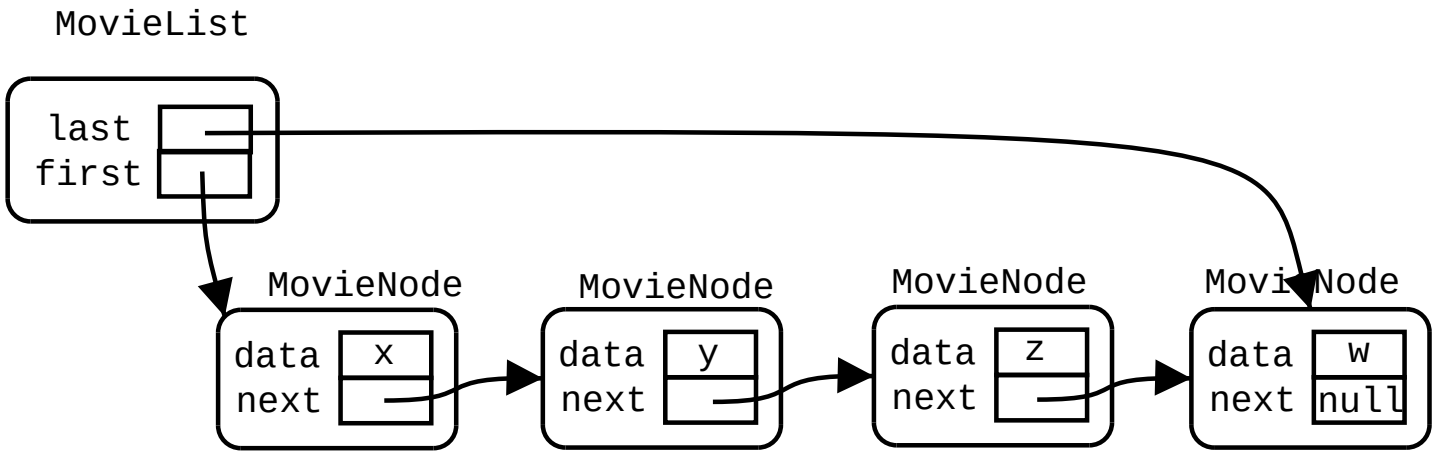
Linked Lists

```
class MovieList {
  private MovieNode first;
  //...
  public void add_at_end(Movie m)
  {
    MovieNode new_node = new MovieNode(m, null);
    MovieNode pointer;
    if (first == null) {
      first = new_node;
    }
    else {
      pointer = first;
      while (pointer.get_next() != null) {
        pointer = pointer.get_next();
      }
      pointer.set_next(new_node);
    }
  }
}
```

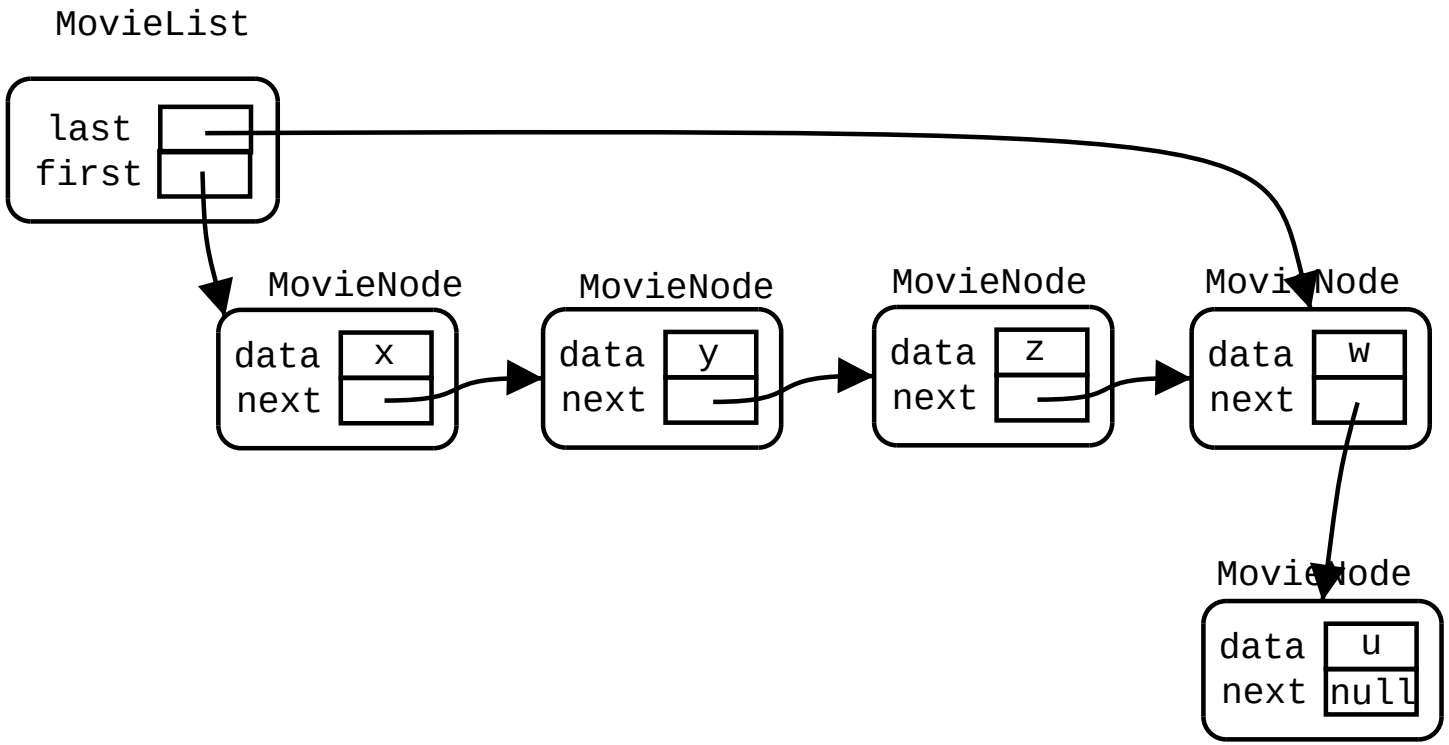
Linked Lists

```
class MovieList {
  private MovieNode first, last;
  //...
  public void add_at_end(Movie m)
  {
    MovieNode new_node = new MovieNode(m, null);
    if (first == null) {
      first = new_node;
      last = new_node;
    }
    else {
      last.set_next(new_node);
      last = new_node;
    }
  }
}
```

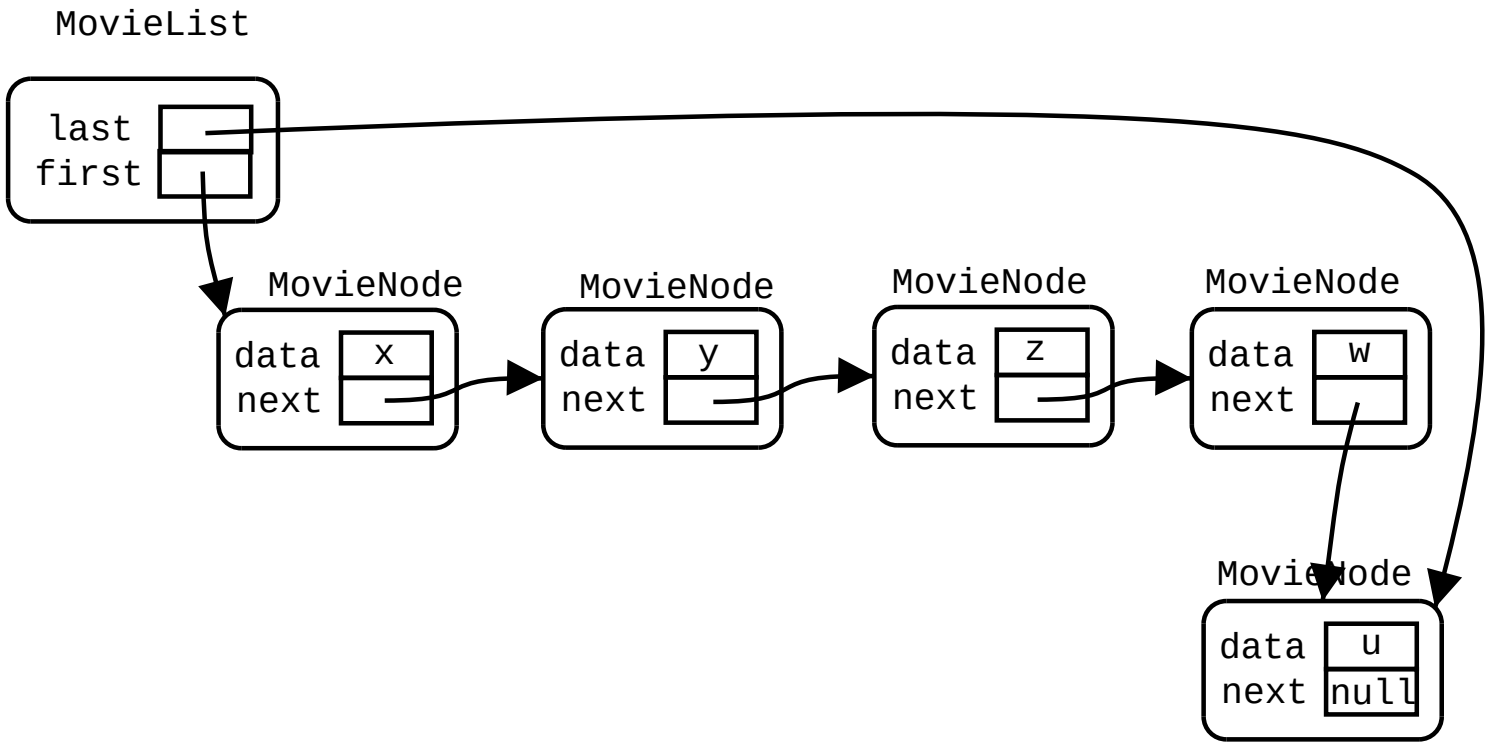
Linked-lists



Linked-lists



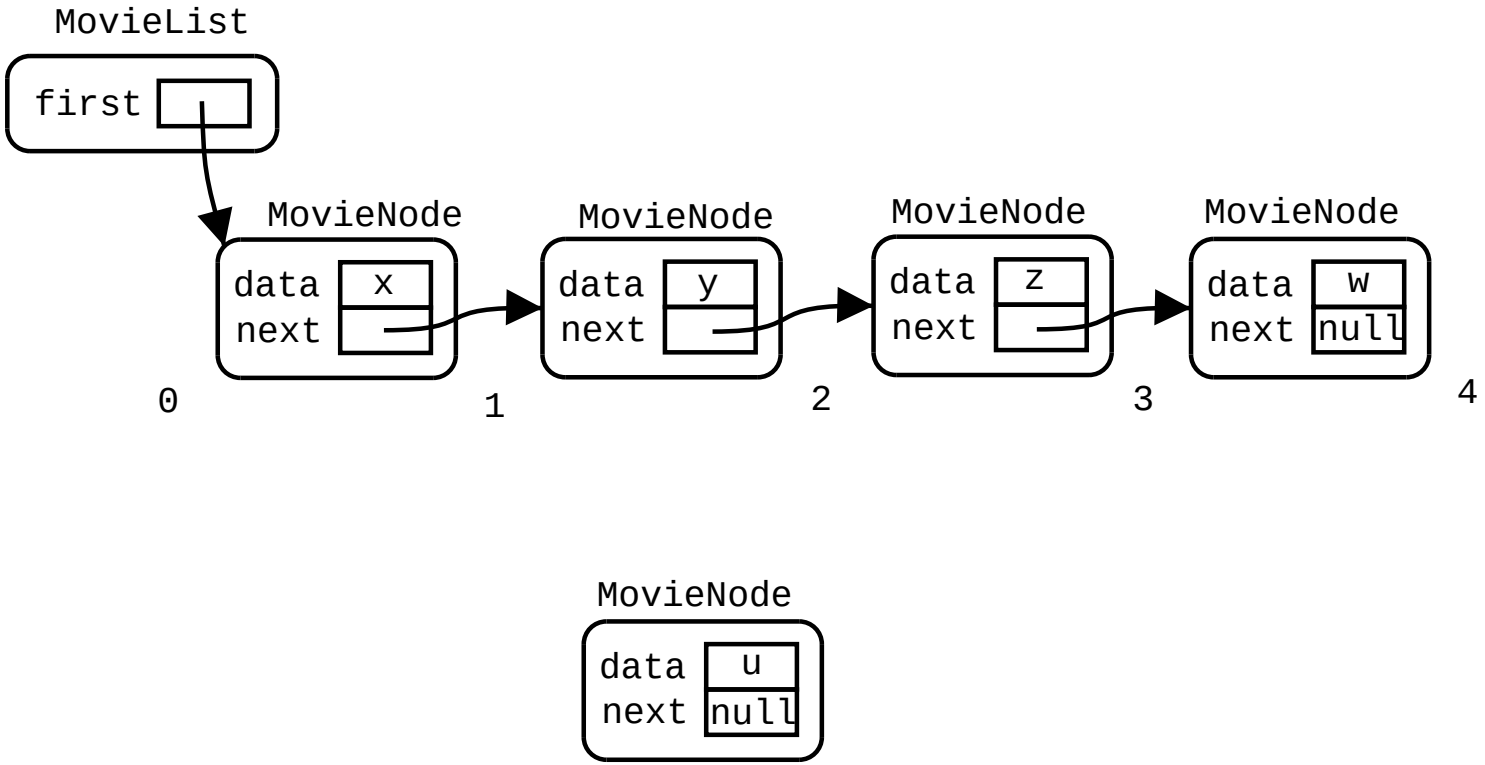
Linked-lists



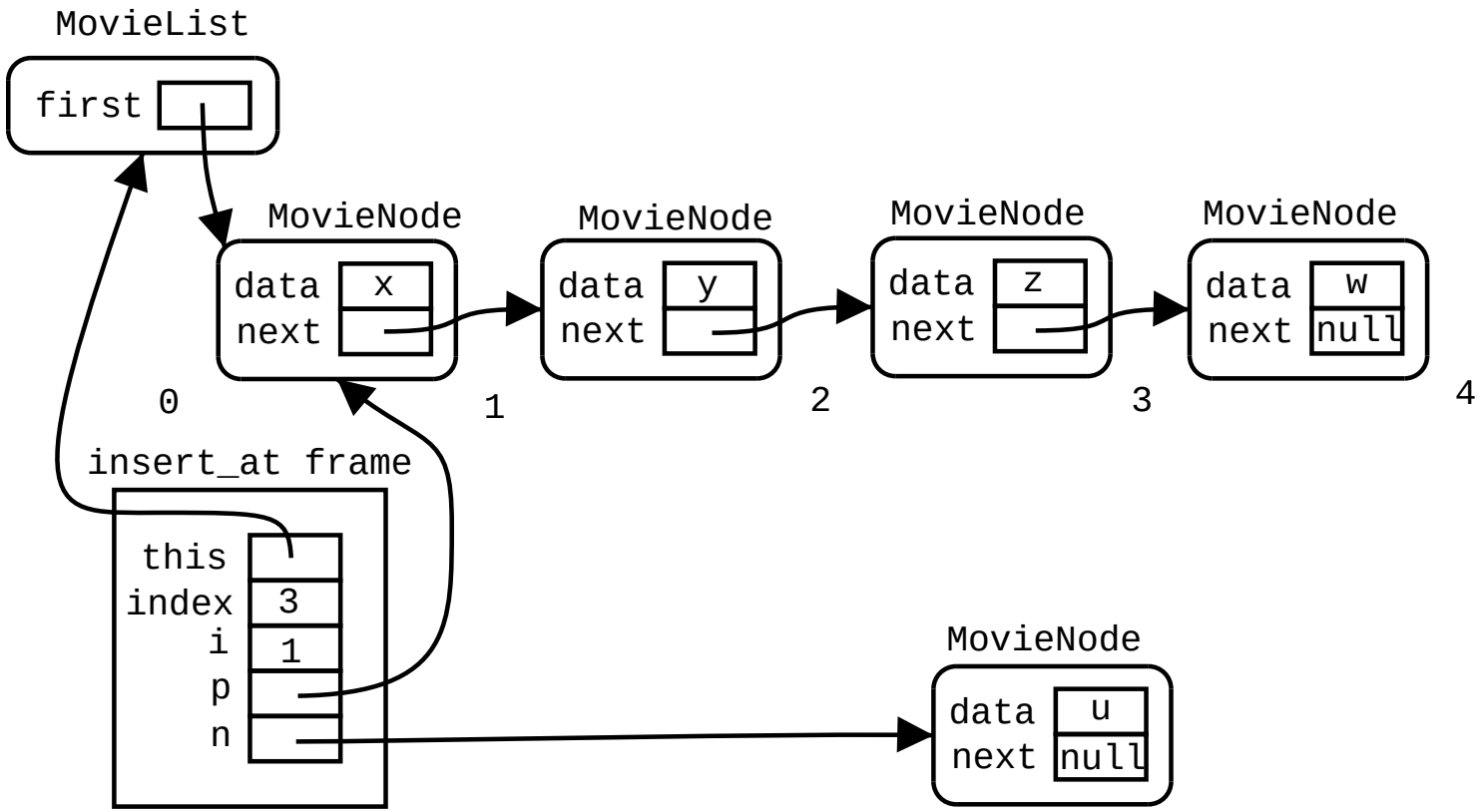
Linked-lists

```
public void insert_at(Movie m, int index)
throws IndexOutOfBoundsException {
  if (index < 0)
    throw new IndexOutOfBoundsException();
  MovieNode n = new MovieNode(m, null);
  if (index == 0) {
    n.set_next(first);
    first = n;
  }
  else {
    MovieNode p = first;
    int i = 1;
    while (i < index && p != null) {
      p = p.get_next();
      i++;
    }
    if (p == null)
      throw new IndexOutOfBoundsException();
    n.set_next(p.get_next());
    p.set_next(n);
  }
}
```

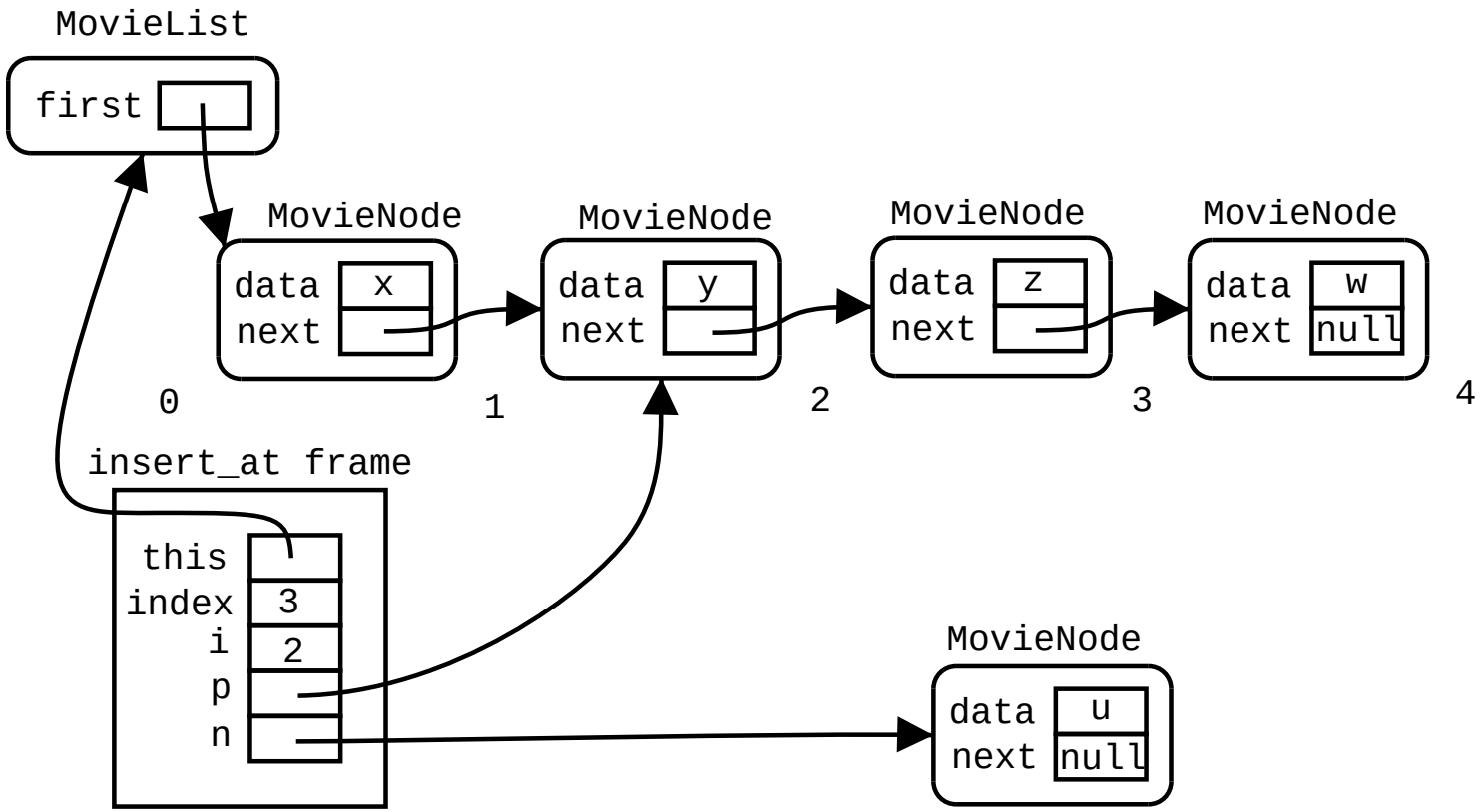
Linked-lists



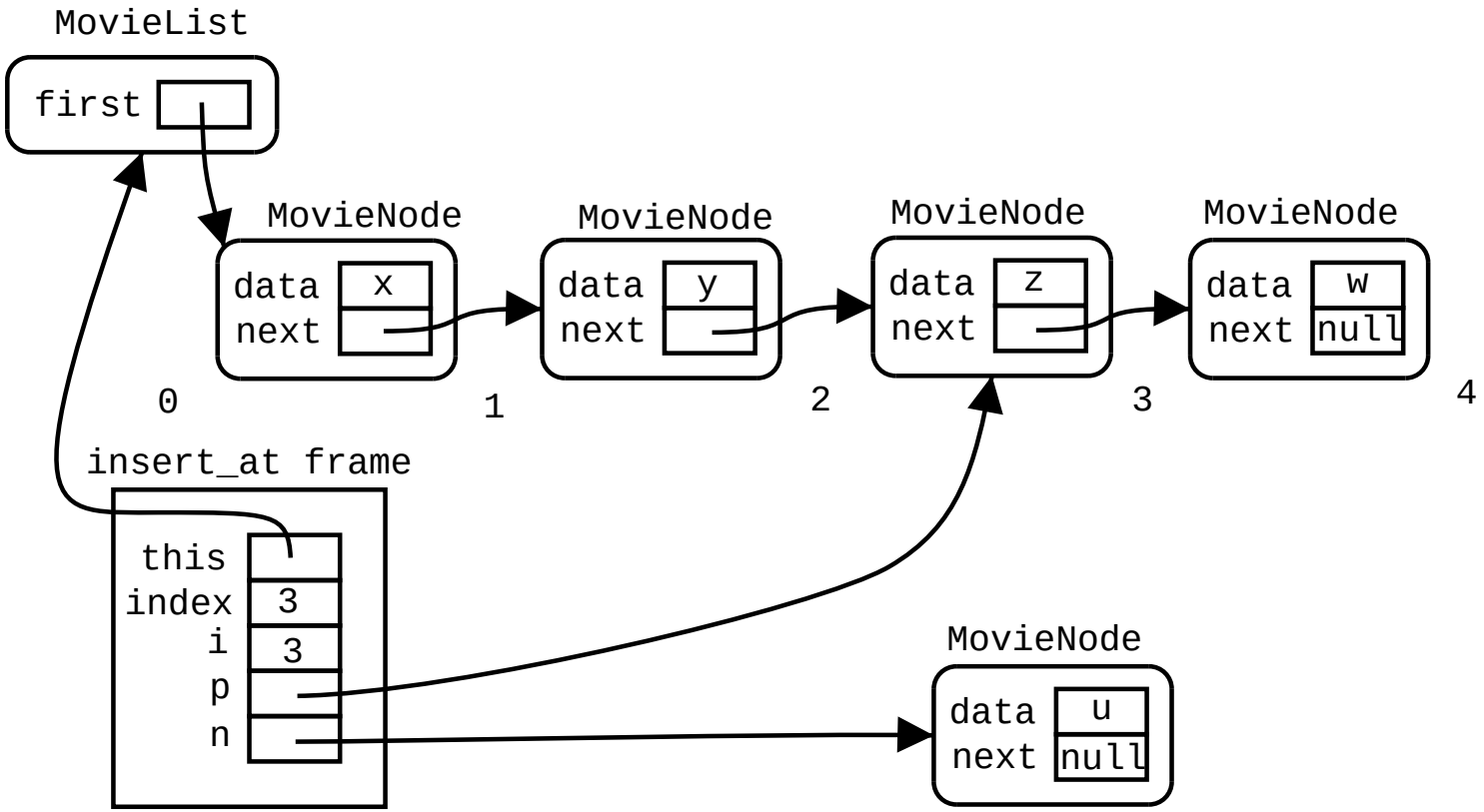
Linked-lists



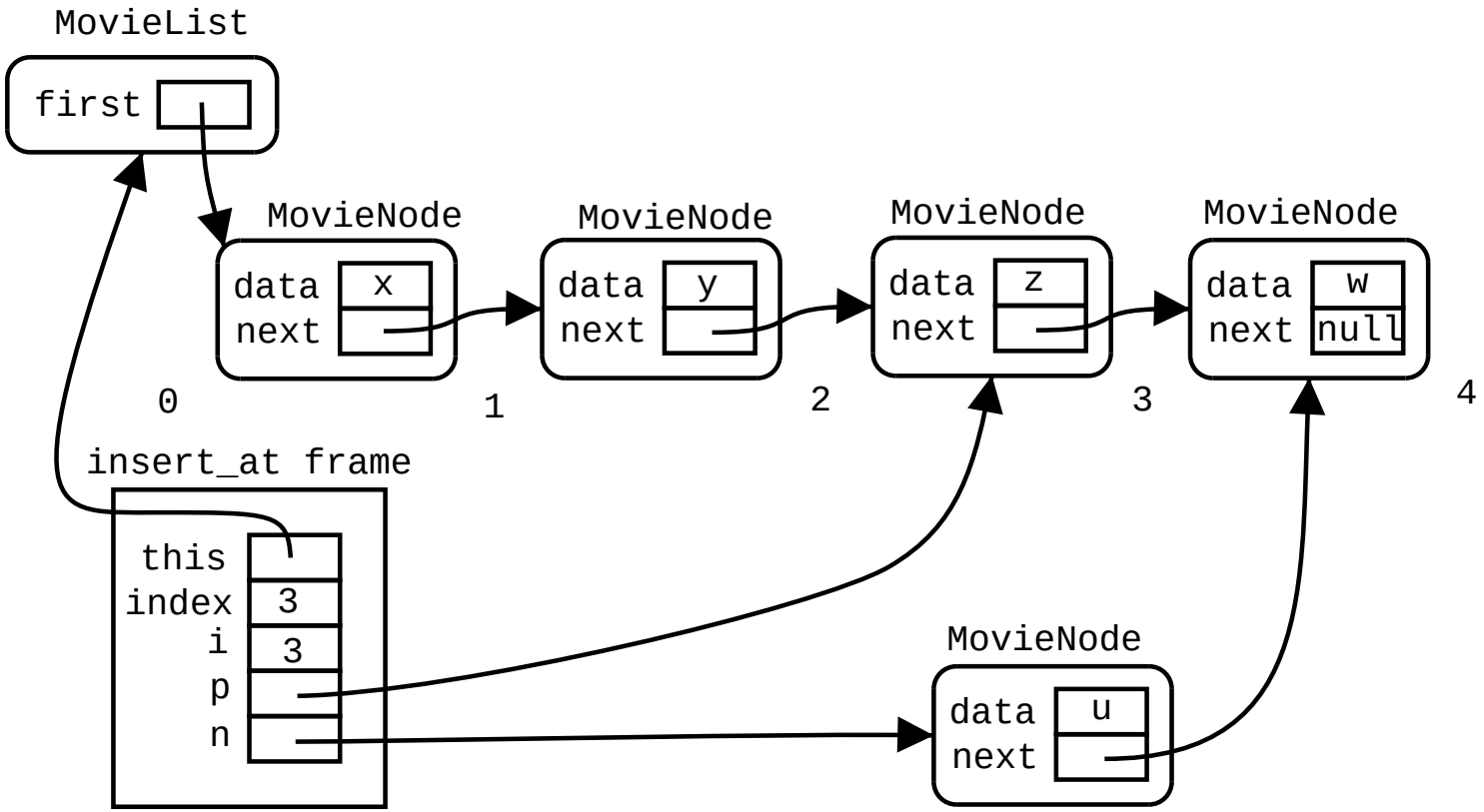
Linked-lists



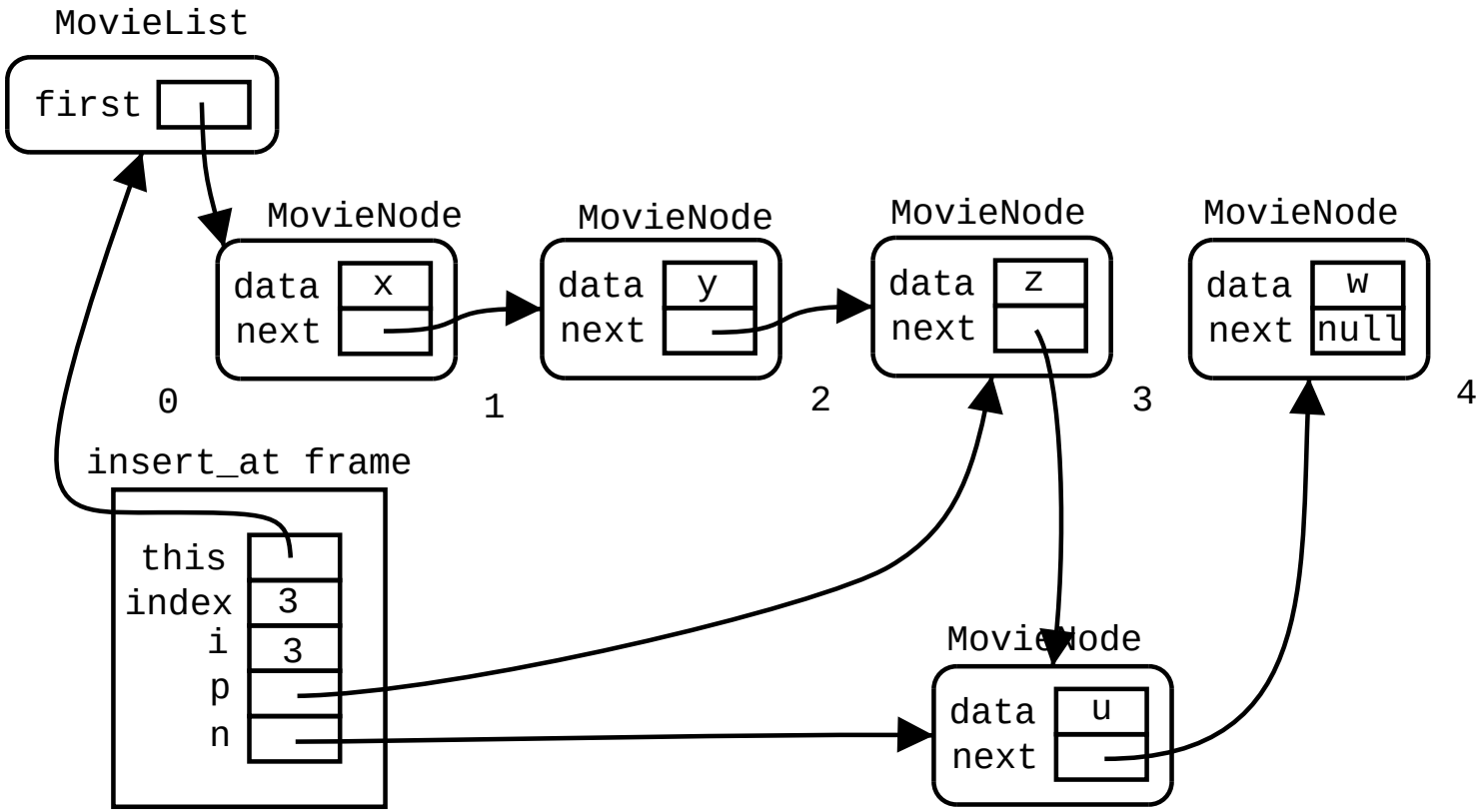
Linked-lists



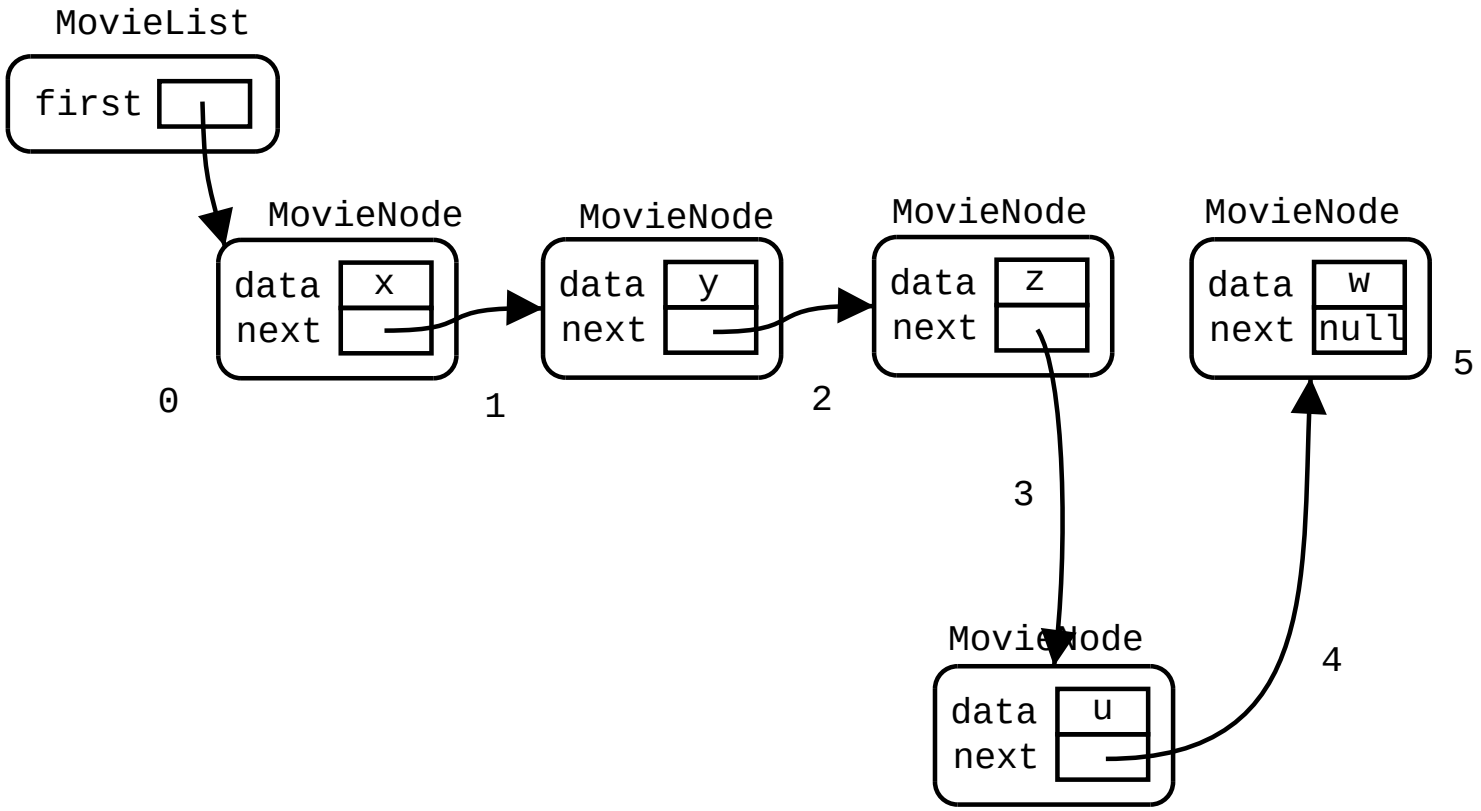
Linked-lists



Linked-lists



Linked-lists



Linked-lists

```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  public void add(Movie m)
  throws IndexOutOfBoundsException
  {
    insert_at(m, 0);
  }
  public void add_at_end(Movie m)
  throws IndexOutOfBoundsException
  {
    insert_at(m, length());
  }
}
```

Linked-lists

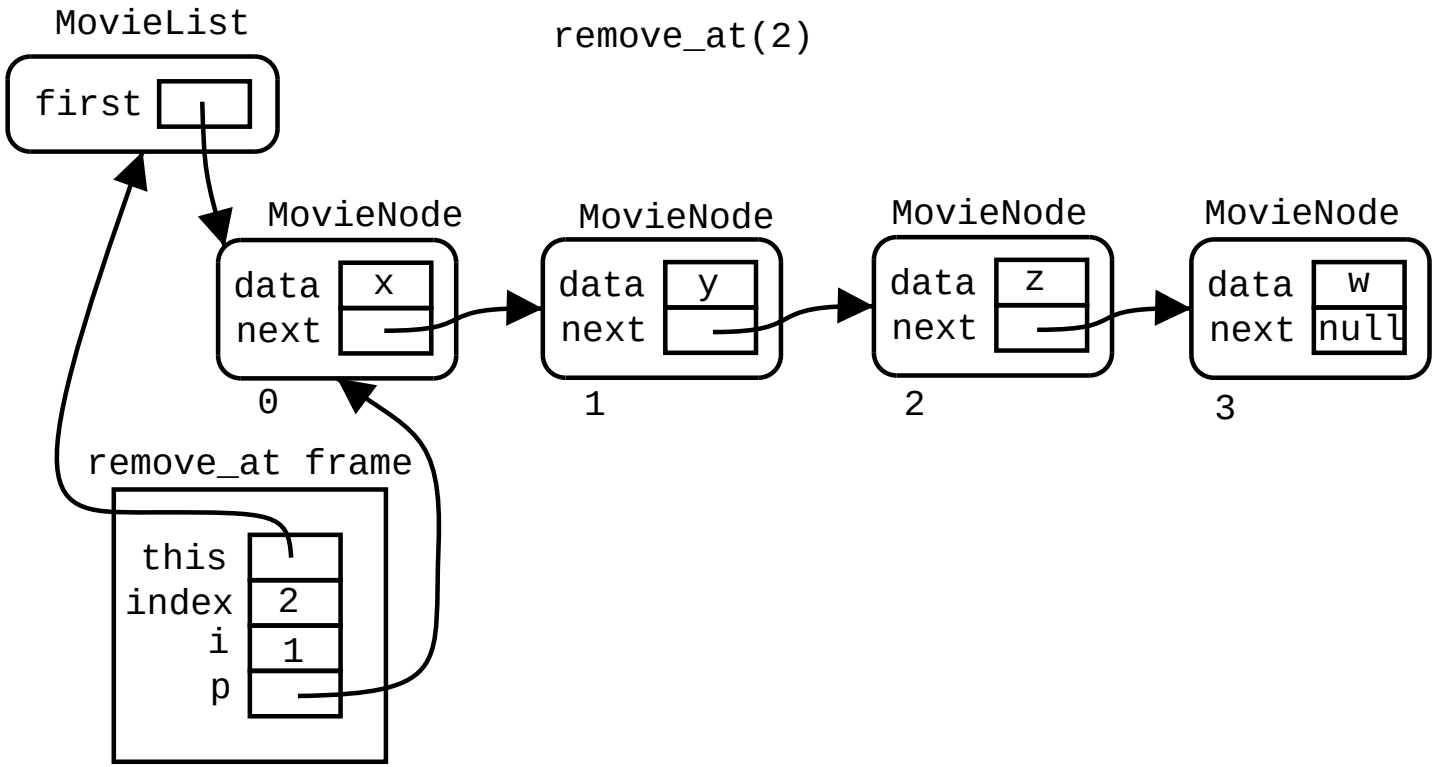
```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  public void remove_first()
  throws IndexOutOfBoundsException
  {
    if (first == null)
      throw new IndexOutOfBoundsException();
    first = first.get_next();
  }
}
```

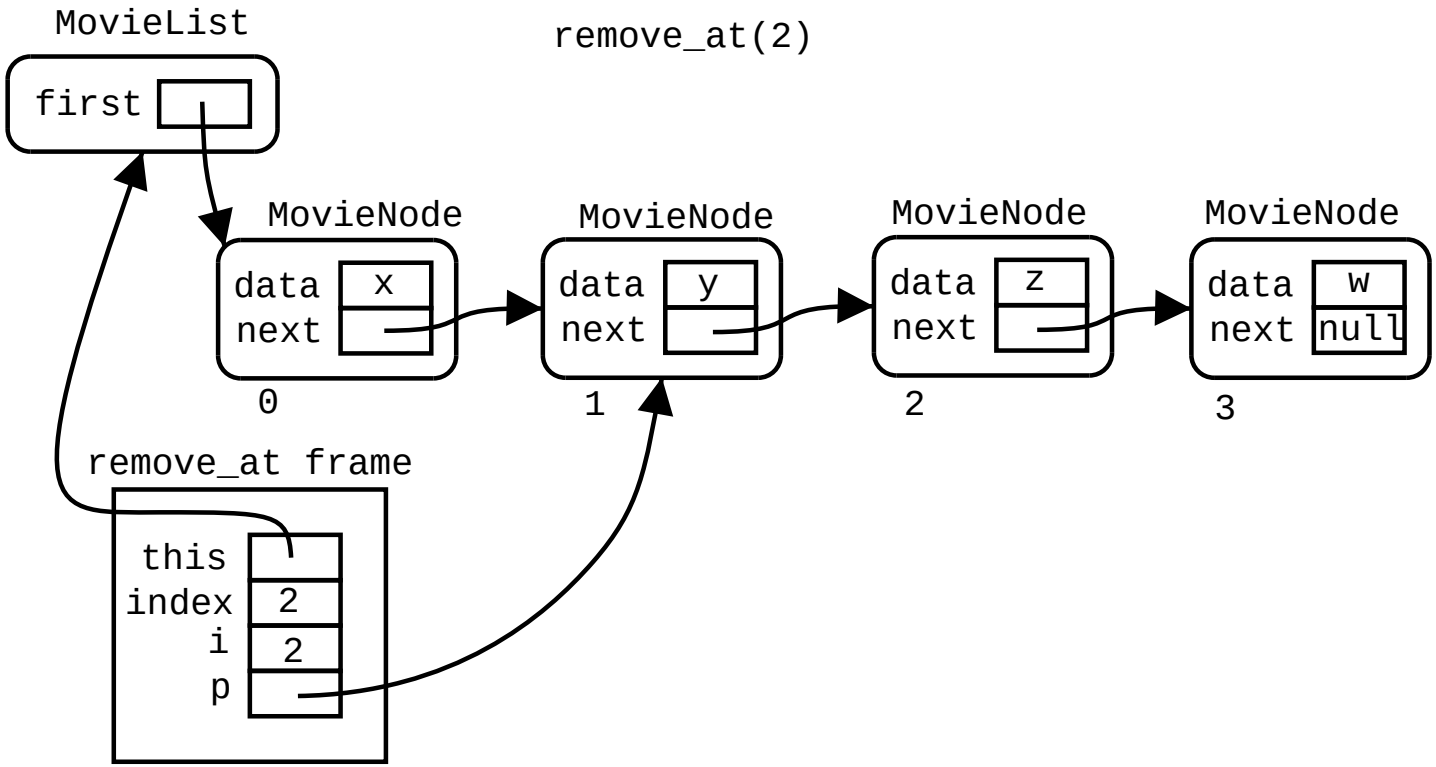
Linked-lists

```
public void remove_at(int index)
throws IndexOutOfBoundsException
{
  if (index < 0)
    throw new IndexOutOfBoundsException();
  if (index == 0) {
    first = first.get_next();
  }
  else {
    MovieNode p = first;
    int i = 1;
    while (i < index && p.get_next() != null) {
      p = p.get_next();
      i++;
    }
    if (p.get_next() == null)
      throw new IndexOutOfBoundsException();
    p.set_next(p.get_next().get_next());
  }
}
```

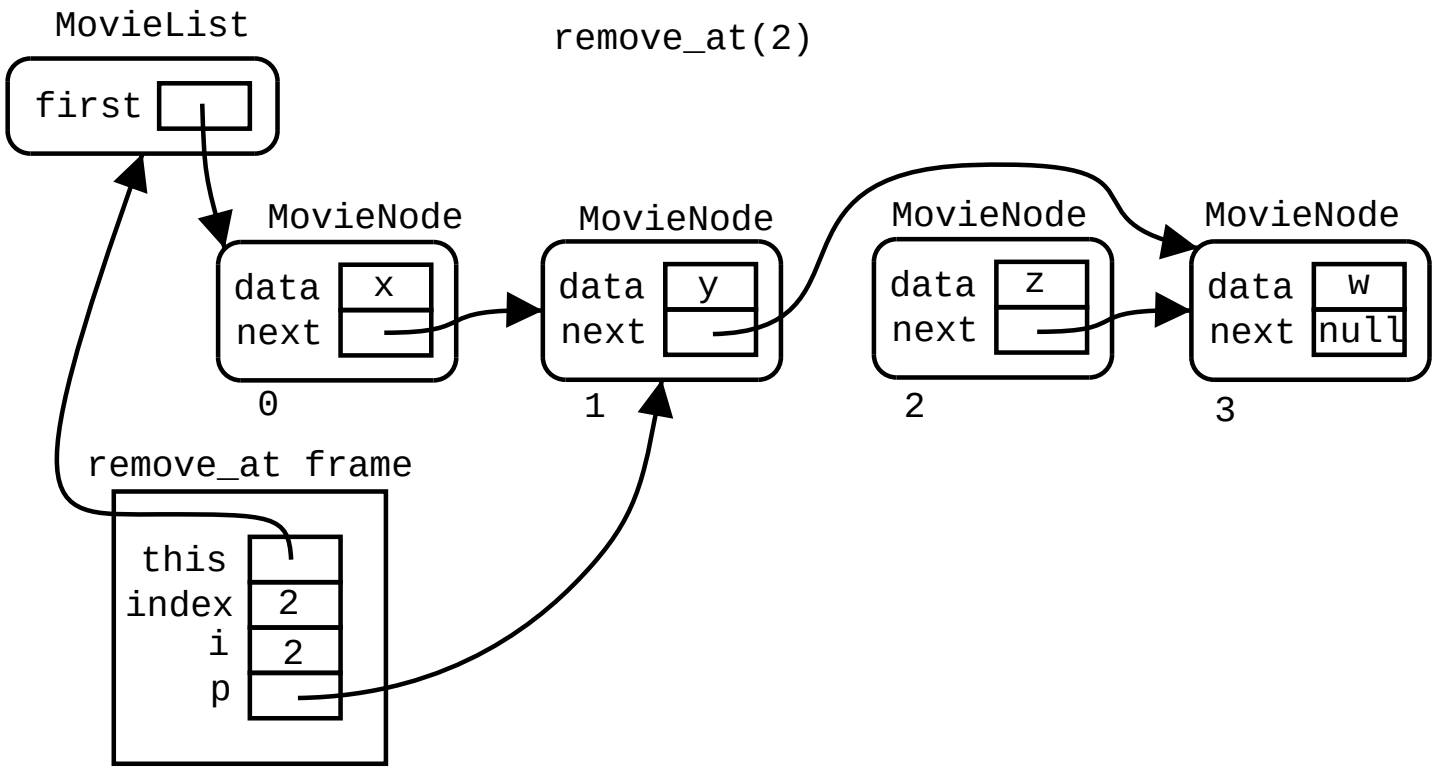
Linked-lists



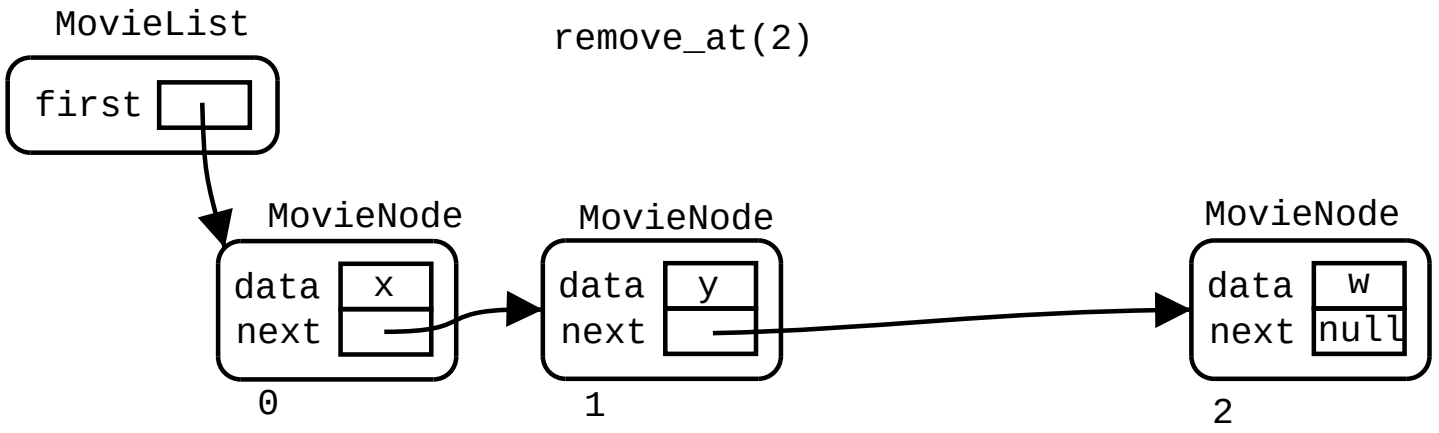
Linked-lists



Linked-lists



Linked-lists



Linked-lists

```
class MovieList {
  MovieNode first;

  MovieList() { first = null; }
  boolean equals(MovieList l)
  {
    if (l == null) return false;
    if (first == null) return l.first == null;
    return first.equals(l.first);
  }
}
```

Linked-lists

```
class Movie {
  // ...
  public boolean equals(Movie m) { ... }
}
class MovieNode {
  Movie data;
  MovieNode next;
  //...
  public boolean equals(MovieNode n) {
    if (n == null) return false;
    boolean equal_data = data.equals(n.data);
    if (next == null && n.next == null)
      return equal_data;
    return equal_data && next.equals(n.next);
  }
}
```

Linked-lists

- Structural equality of two lists:

- Given two lists A and B

1. If A is non-empty and B is empty, then they are different

2. If A and B have only one element, they are equal if the elements are equal

3. Otherwise, A and B are equal if their first elements are equal and the rest of A is equal to the rest of B

... where "the rest of L" means the list L without its first element.