An essential data structure: **Lists**

Lists

- A list is a finite sequence of elements
- Examples of lists in Prolog:

[mia, vincent, jules, yolanda]

[mia, robber(honeybunny), X, 2, mia]

[ ]

[mia, [vincent, jules], [butch, friend(butch)]]

[[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Important things about lists

- List elements are enclosed in square brackets
- The length of a list is the number of elements it has
- All sorts of Prolog terms can be elements of a list
- There is a special list:
  the empty list [ ]

Head and Tail

- A non-empty list can be thought of as consisting of two parts
  - The head
  - The tail
- The head is the first item in the list
- The tail is everything else
  - The tail is the list that remains when we take the first element away
  - The tail of a list is always a list

Head and Tail example 1

- [mia, vincent, jules, yolanda]

  Head:
  Tail:

Head and Tail example 1

- [mia, vincent, jules, yolanda]

  Head: mia
  Tail:

Head and Tail example 1

- [mia, vincent, jules, yolanda]

  Head: mia
  Tail: [vincent, jules, yolanda]

Head and Tail example 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head:
  Tail:

Head and Tail example 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head: [ ]
  Tail:

Head and Tail example 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head: [ ]
  Tail: [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Head and Tail example 3

- [dead(z)]

  Head:
  Tail:

Head and Tail example 3

- [dead(z)]

  Head: dead(z)
  Tail:

Head and Tail example 3

- [dead(z)]

  Head: dead(z)
  Tail: [ ]

Head and tail of empty list

- The empty list has neither a head nor a tail
- For Prolog, [ ] is a special simple list without any internal structure

- The empty list plays an important role in recursive predicates for list processing in Prolog

The built-in operator |

- Prolog has a special built-in operator | which can be used to decompose a list into its head and tail
- The | operator is a key tool for writing Prolog list manipulation predicates

The built-in operator |

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].

Head = mia
Tail = [vincent,jules,yolanda]
yes

?-
```

The built-in operator |

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent,jules,yolanda]
yes

?-
```

The built-in operator |

```
?- [X|Y] = [ ].

no

?-
```

The built-in operator |

```
?- [X,Y|Tail] = [[ ], dead(z), [2, [b,c]], [], Z, [2, [b,c]]] .

X = [ ]
Y = dead(z)
Z = _4543
Tail = [[2, [b,c]], [ ], Z, [2, [b,c]]]
yes

?-
```

Anonymous variable

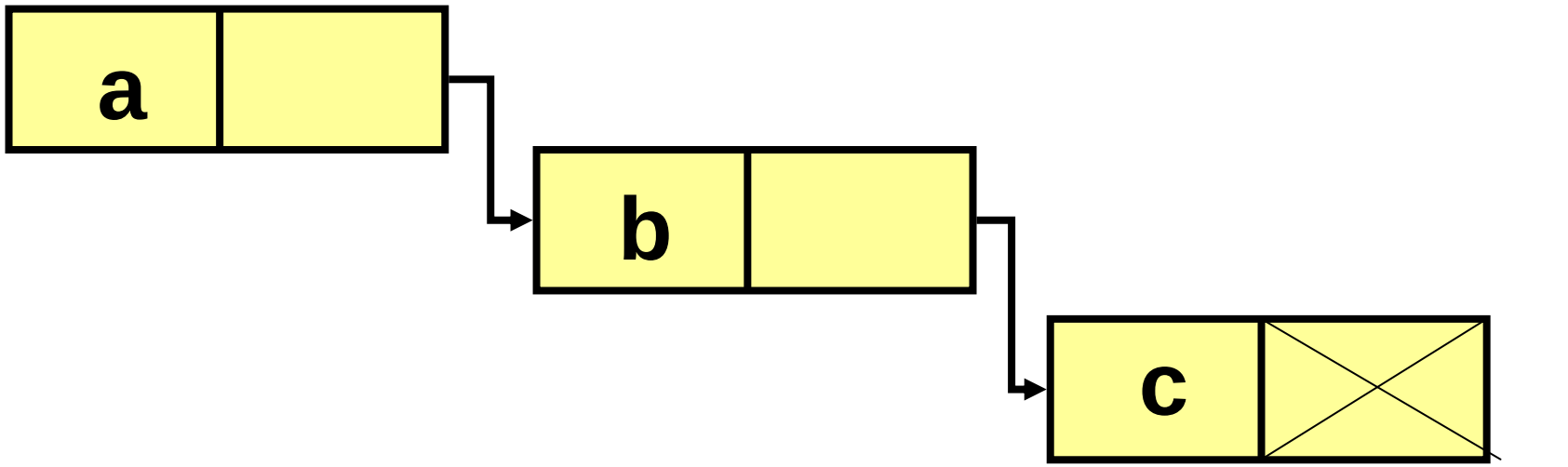
- Suppose we are interested in the second and fourth element of a list

```
?- [X1,X2,X3,X4|Tail] = [mia, vincent, marsellus, jody, yolanda].
X1 = mia
X2 = vincent
X3 = marsellus
X4 = jody
Tail = [yolanda]
yes

?-
```

Another view of lists

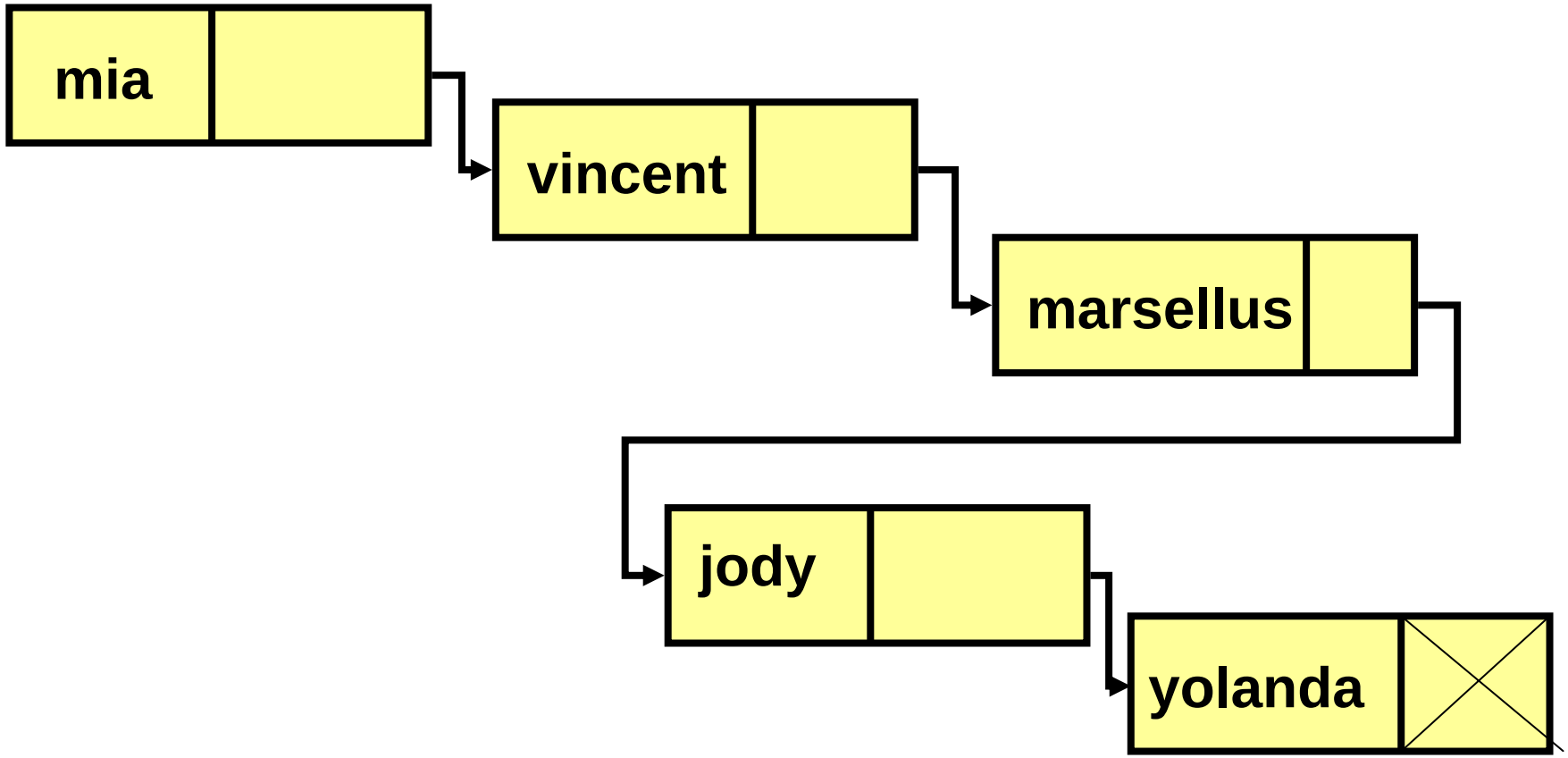
- A list [a, b, c] can be seen as the following structure:



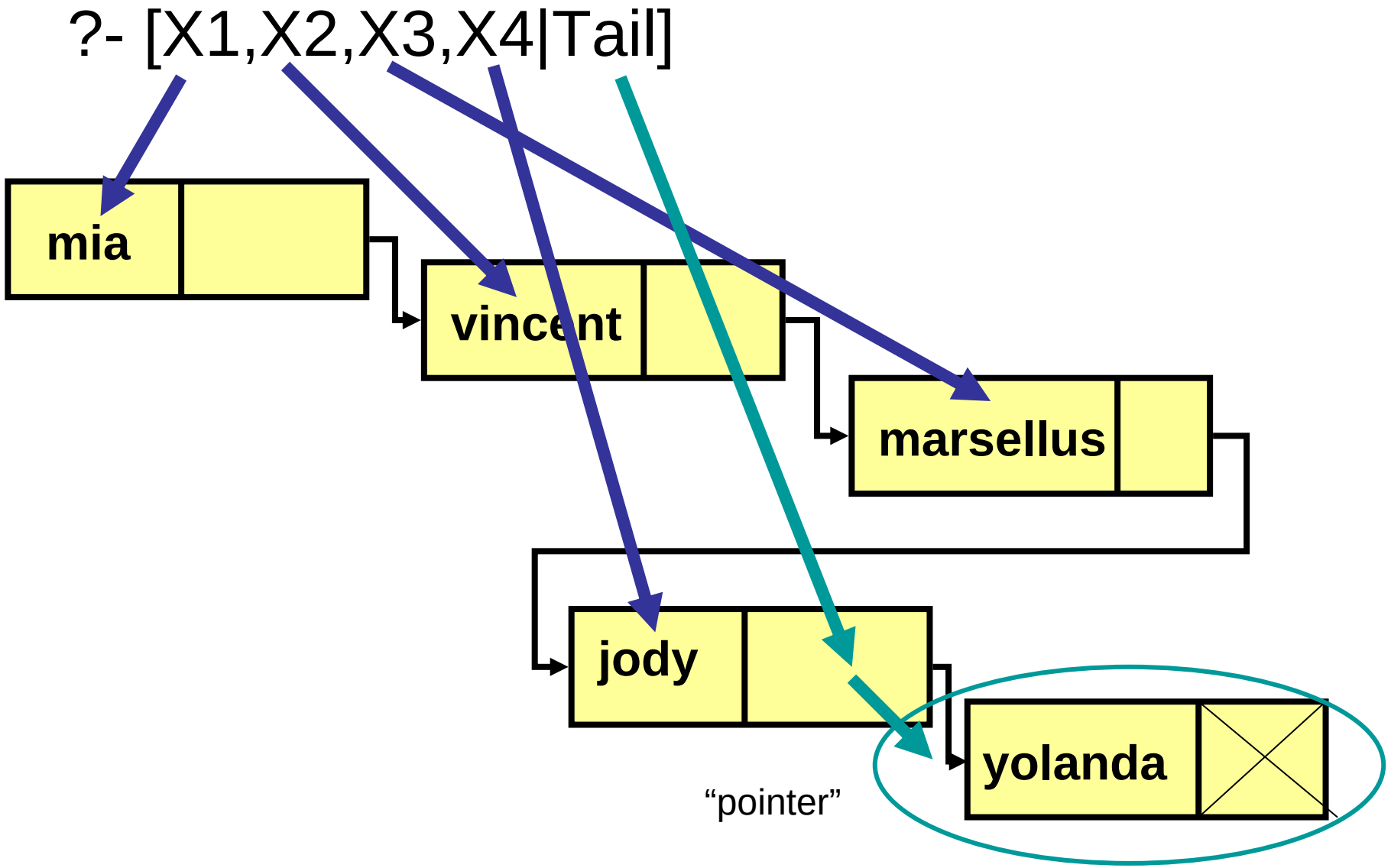
[ ] *or* **nil**

Another view of lists

- [mia, vincent, marsellus, jody, yolanda].

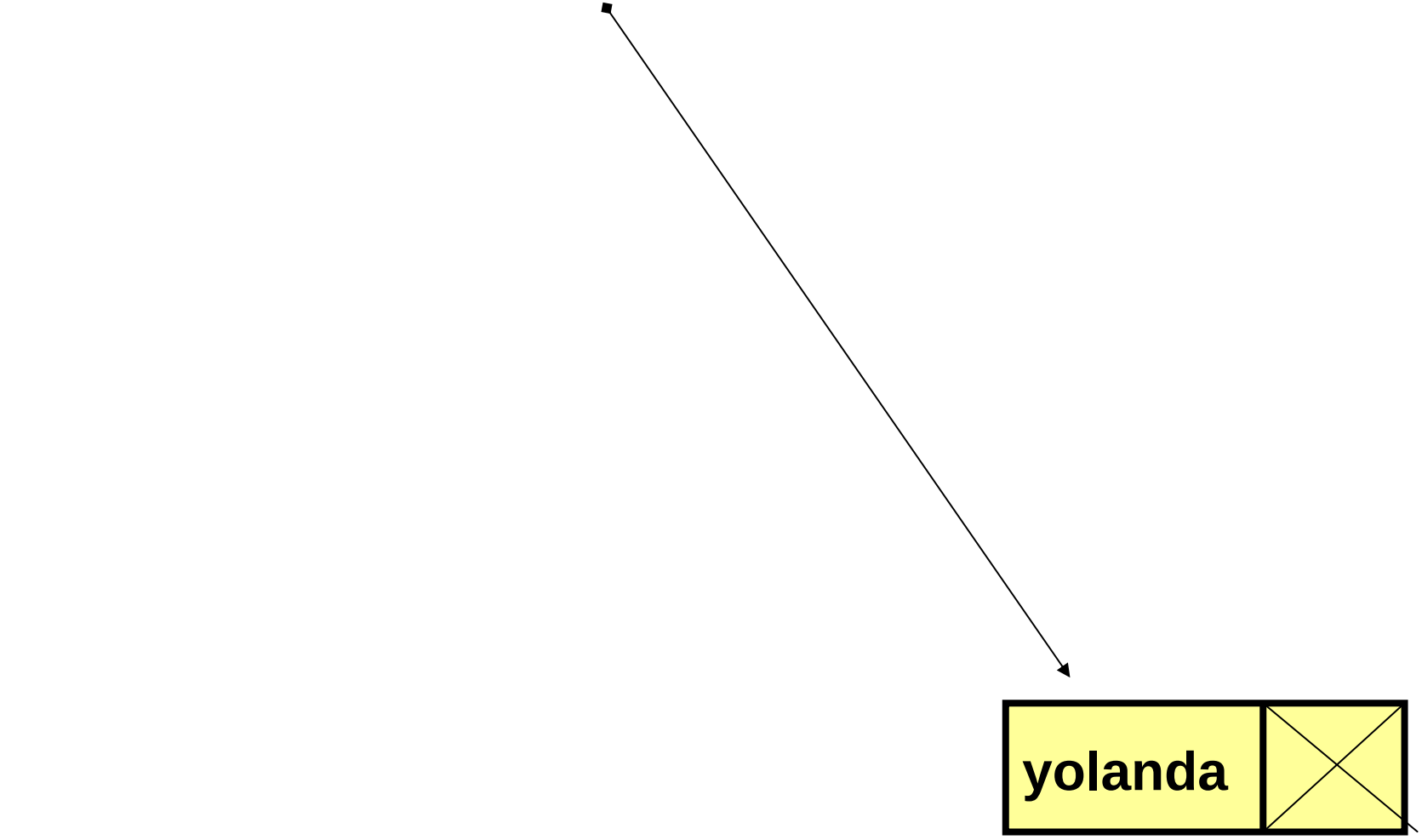


Another view of lists



Another view of lists

?- [X1,X2,X3,X4|Tail]



Exercise

- Work out what Prolog will tell us about the following **before** looking at what it actually does!!!

```
?- [X,Y,Z|Tail] = [[ ], dead(z), [2, [b,c]], [], Z, [2, [b,c]]]
```

Anonymous variables

- There is a simpler way of obtaining only the information we want:

```
?- [ _,X2, _,X4|_ ] = [mia, vincent, marsellus, jody, yolanda].
X2 = vincent
X4 = jody
yes

?-
```

  - The underscore is the anonymous variable

Practical Work and Exercises

More examples and exercises

write a predicate to work out
whether something is an element
of a list of not...

```
memberof (Element, List)
```

should return yes, iff Element is one
of the members of the list.

CLUE: THINK RECURSIVELY!!!

More examples and exercises

write a predicate to stick two lists together, end to end, i.e., to ***append*** one list to another...

```
app (List1, List2, Result)
```

e.g., if List1 is [a,b]
and List2 is [1,2,3]
then Result should be [a,b,1,2,3].

Instantiations

- When Prolog unifies two terms it performs all the necessary instantiations, so that the terms are equal afterwards
- This makes unification a powerful programming mechanism

More examples and exercises

write a predicate to ‘look up’ the corresponding value for a given key in an association list consisting of pairs of keys and values

```
lookup (Key, Alist, Result)
```

e.g., if Key is fred

and Alist is

[[mary,judy],[george,mary],[peter,june],
[fred,judy],[john,james]]

then Result should be judy.

More examples and exercises

write a predicate to reverse the elements
of a list!

```
rev (List, Result)
```

e.g., if List is [a,b,c,d]
then Result should be [d,c,b,a].