A05 Sets: TreeSet

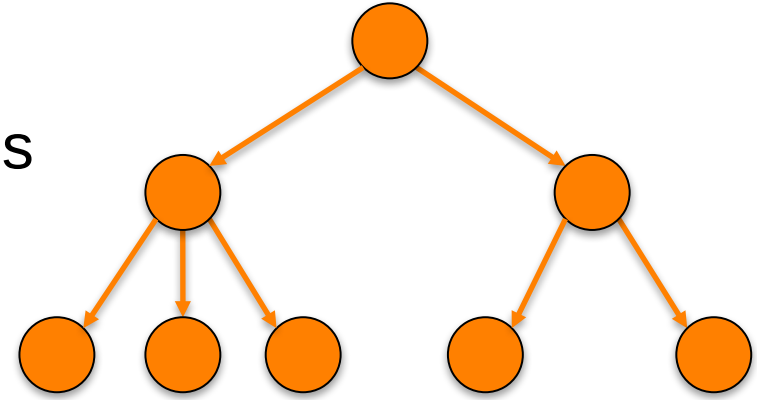
Tree as an ADT
A tree-based Set implementation

Tree as an ADT

The **tree** ADT corresponds to an ordered tree in mathematics.

A tree is defined recursively in terms of nodes:

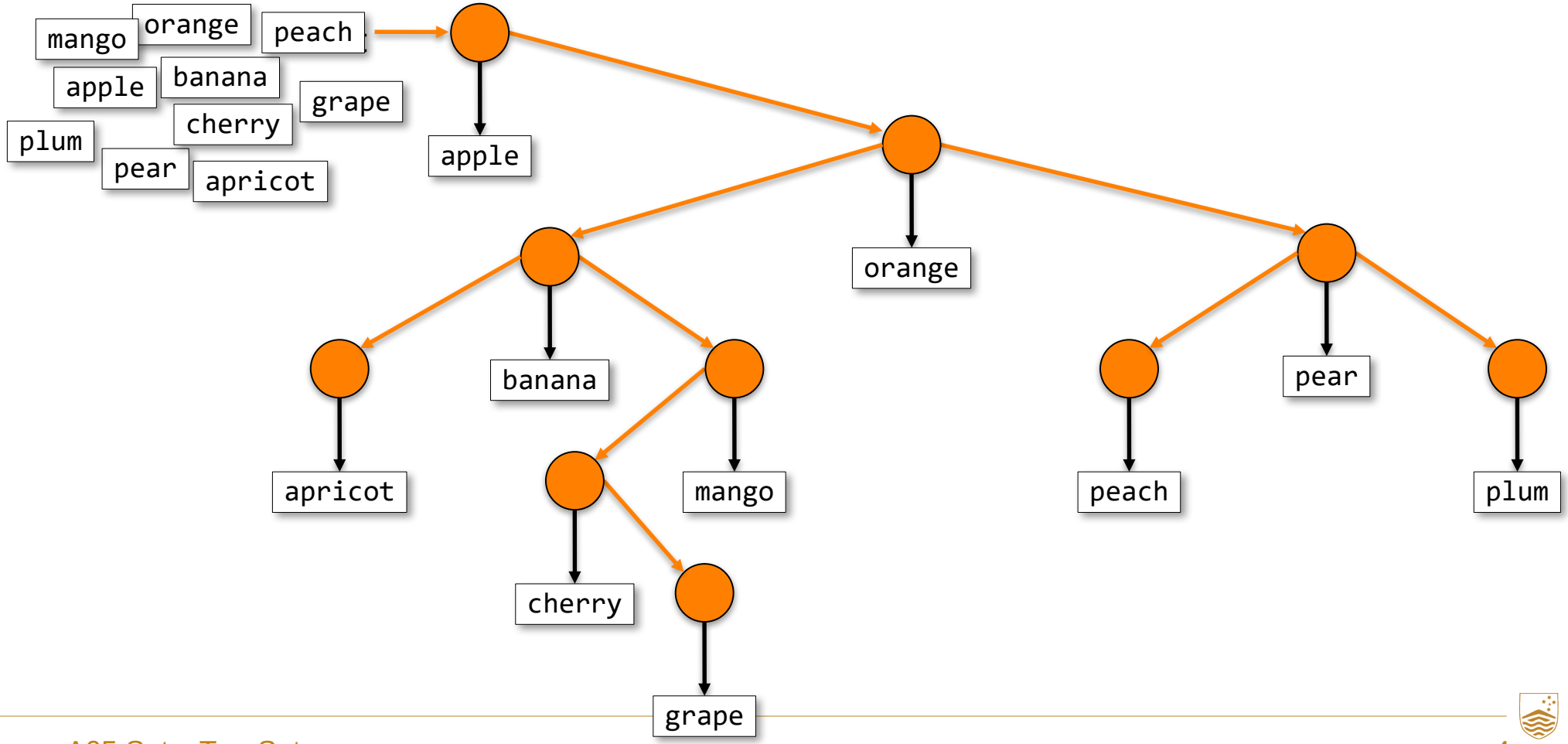
- A tree is a node
- A node contains a value and a list of trees
- No node is duplicated

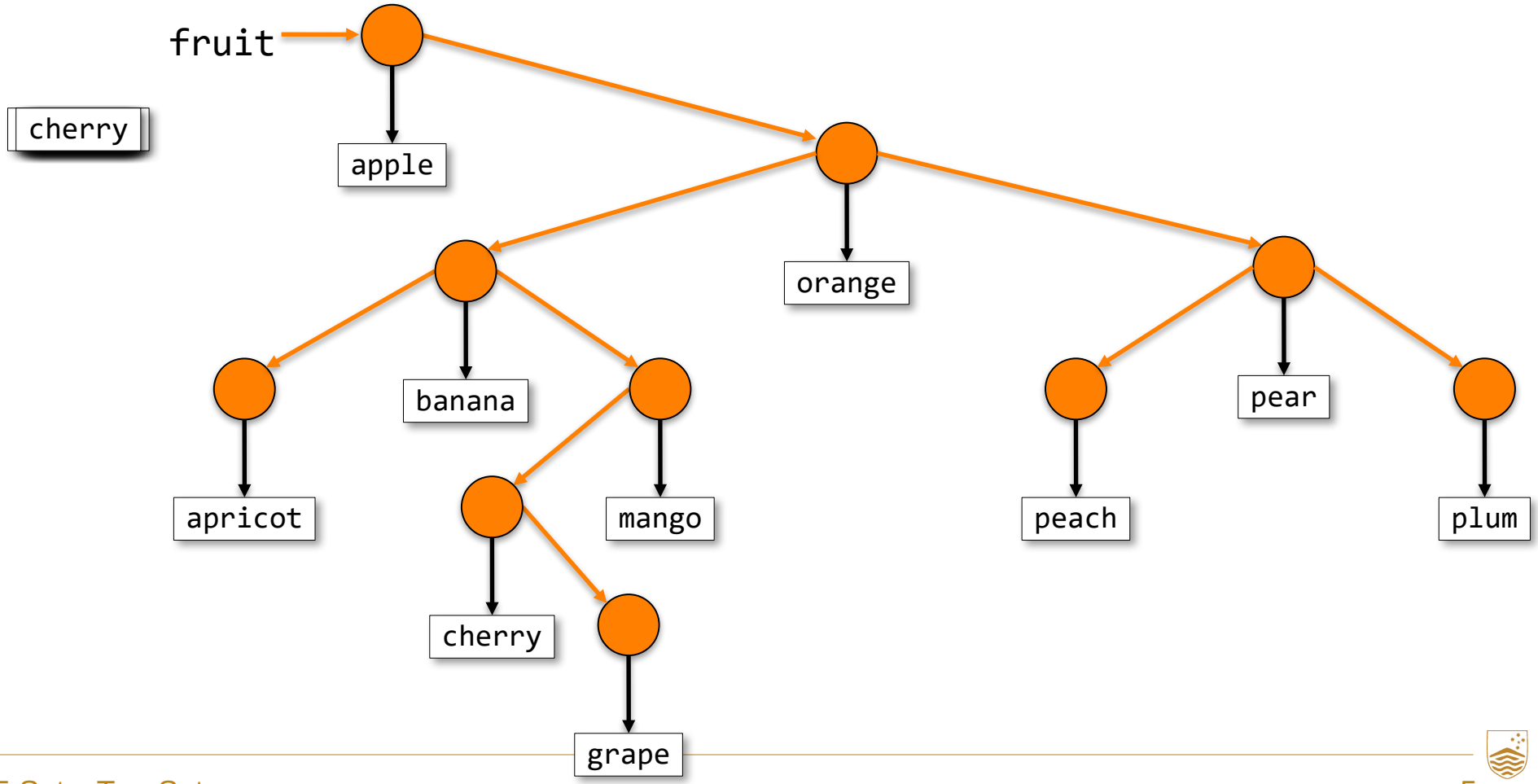


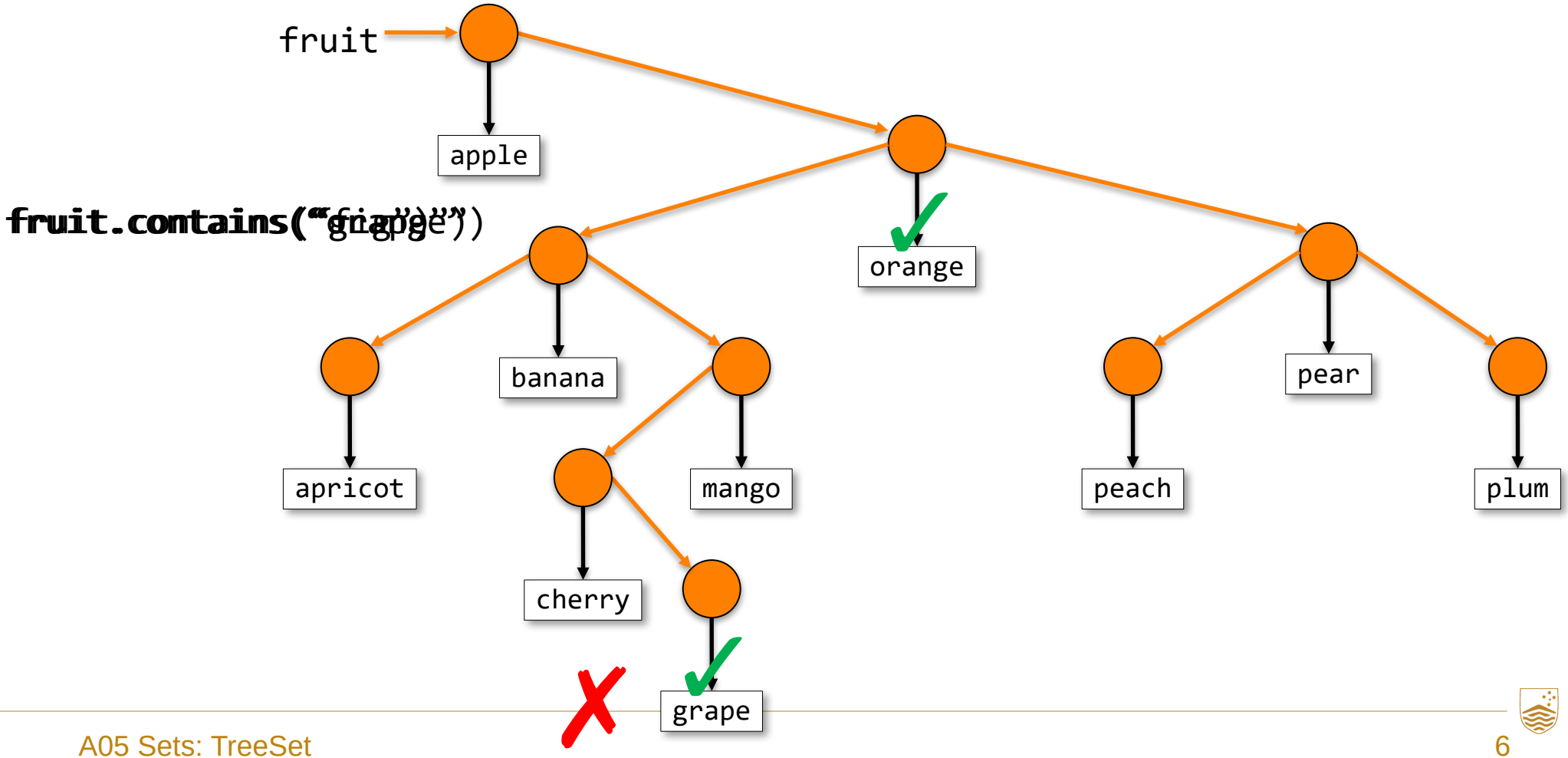
Binary Search Tree to Implement Set

A **binary** search tree is a tree with the following additional properties:

- Each node has at most **two** sub-trees
- Nodes may contain (key, value) pairs, or just keys
- Keys are ordered within the tree:
  - The left sub-tree only contains keys less than the node's key
  - The right sub-tree only contains keys greater than the node's key


mango
orange
peach
apple
banana
grape
cherry
plum
pear
apricot
apple
orange
banana
apricot
mango
peach
pear
plum
cherry
grape


fruit
cherry
apple
orange
banana
apricot
mango
cherry
grape
pear
peach
plum


fruit
apple
orange
banana
apricot
mango
cherry
grape
peach
pear
plum
fruit.contains("grape")

Ordering in Java (Recall J14)

Objects of any class that implements the `Comparable` interface can be ordered:

`a.compareTo(b)`

- < 0 iff a is ordered before b
- > 0 iff a is ordered after b
- == 0 if a.equals(b) (but also if a and b are not ordered)

Our `Set` interface does not bound our contained type parameter to be `Comparable`, what to do?

- Bound `T` in the `TreeSet` class declaration:
  - `class TreeSet<T extends Comparable<T>> implements Set<T>`
- Throw runtime exception on use of non-comparable types (the approach in `java.utils.TreeSet`).
- Force users to provide `Comparator` (e.g., as lambda expression).

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Comparable.html

Complexity

```
boolean add(T value);
boolean contains(T value);
int size();
boolean remove(T value);
```

- `add`,`contains`,`remove` – **Time O(log(n)) amortized, O(n) worst**
  - self-balancing trees (e.g., B-Trees) have O(log(n)) worst case
- `size` – **Time O(1)**
  - explicitly tracked

Space O(n)