Australian National University

Review

R1

Australian National University

"80 percent of success is showing up"

Woody Allen

Australian National University

"You can know the name of a bird in all the languages of the world, but when you're finished, you'll know absolutely nothing whatever about the bird... So let's look at the bird and see what it's doing -- that's what counts. I learned very early the difference between knowing the name of something and knowing something."

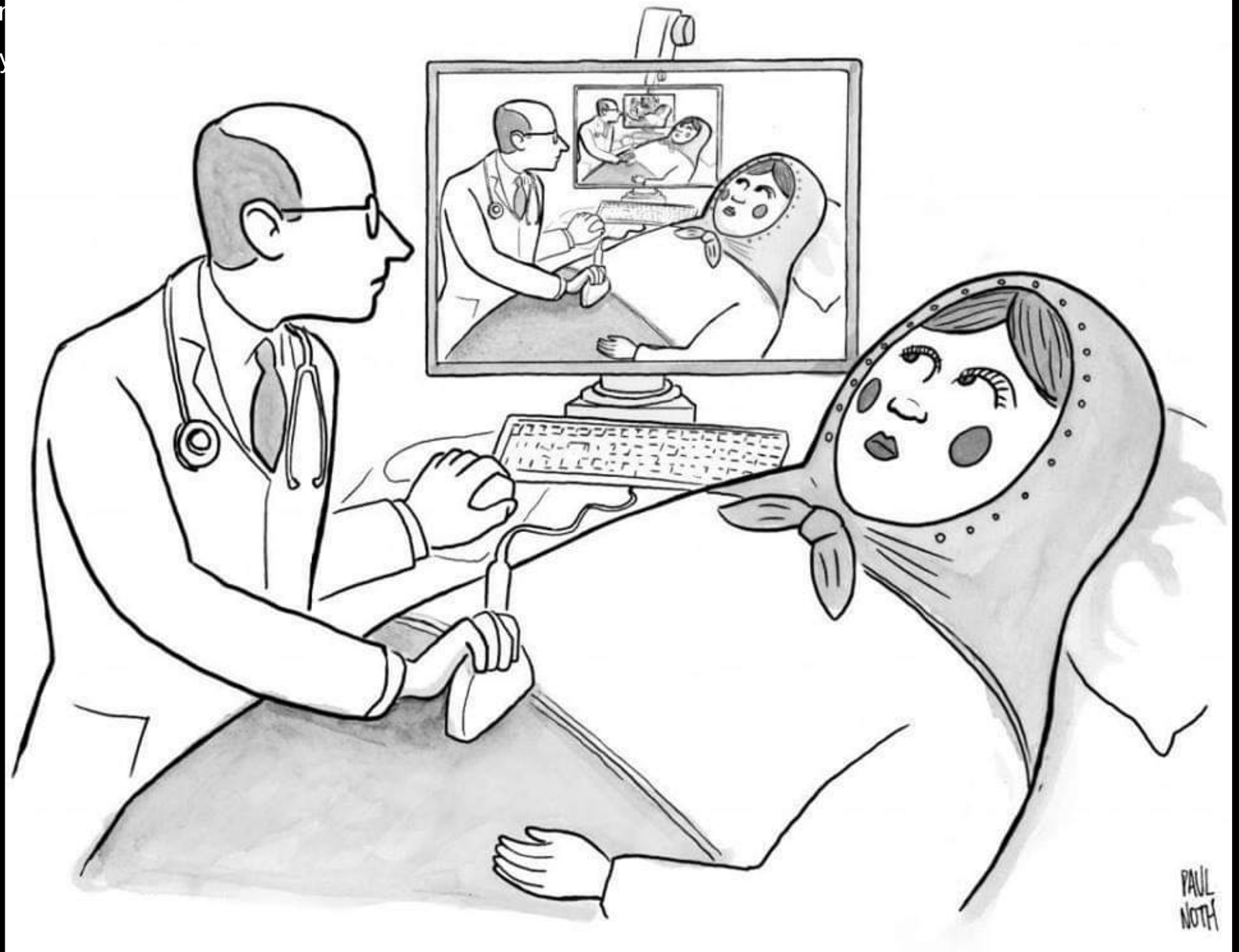
Richard Feynman

Australian National University

(C01) Recursion

Clarify key ideas in recursion


Australian
National
University
C1
Recursion
Recursive Algorithms
Structured Programming 1110/1140/6710


PAUL
NOTH

Recursive Algorithms

A recursive algorithm references itself.

A recursive algorithm is comprised of:

- one or more base cases
- a remainder that reduces to the base case/s

Example: Fibonacci sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377…

fib(0) = 1 *(base case)*

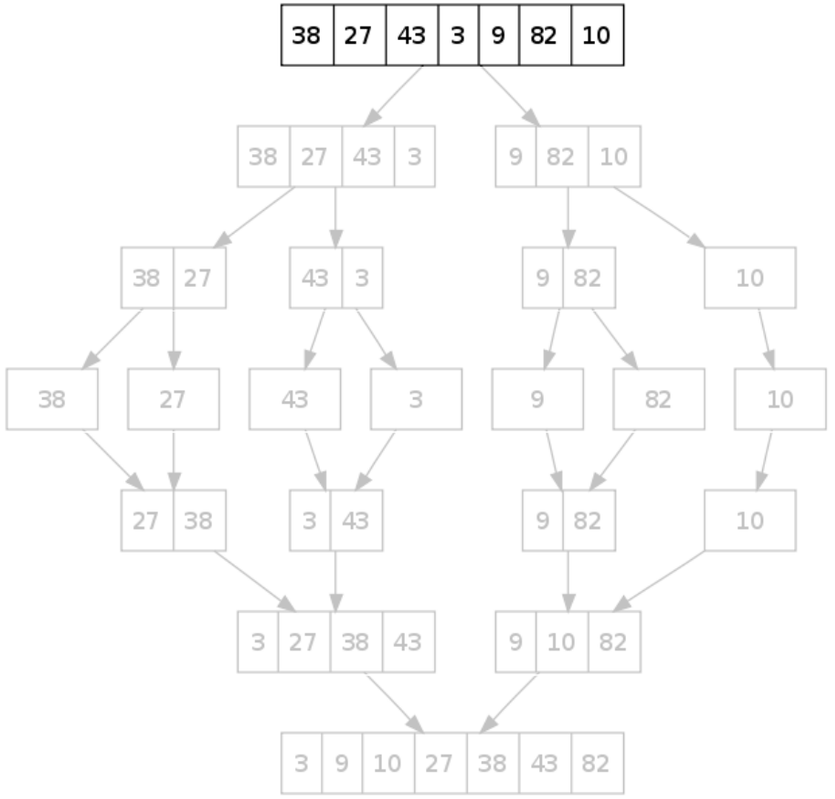
fib(1) = 1 *(base case)*

fib(n) = fib(n-1) + fib(n-2) *(for n ≥ 2)*

Example: Mergesort (von Neumann, 1945)

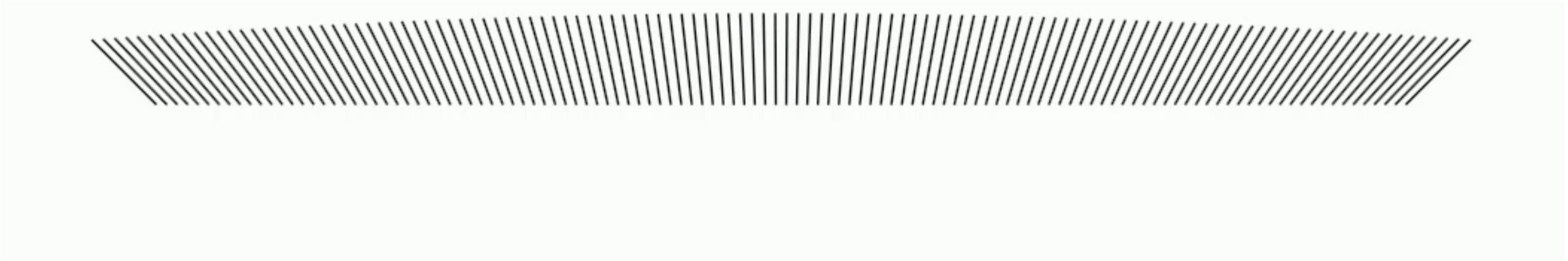
Sort a list

- List of size 1 *(base case)*
  - Already sorted
- List of size > 1
  - Split into two sub lists
  - Sort each sub list *(recursion)*
  - Merge the two sorted sub lists into one sorted list (by iteratively picking the lower of the two least elements)



Animation: Visualizing Algorithms, Mike Bostock, bost.ocks.org/mike/algorithms

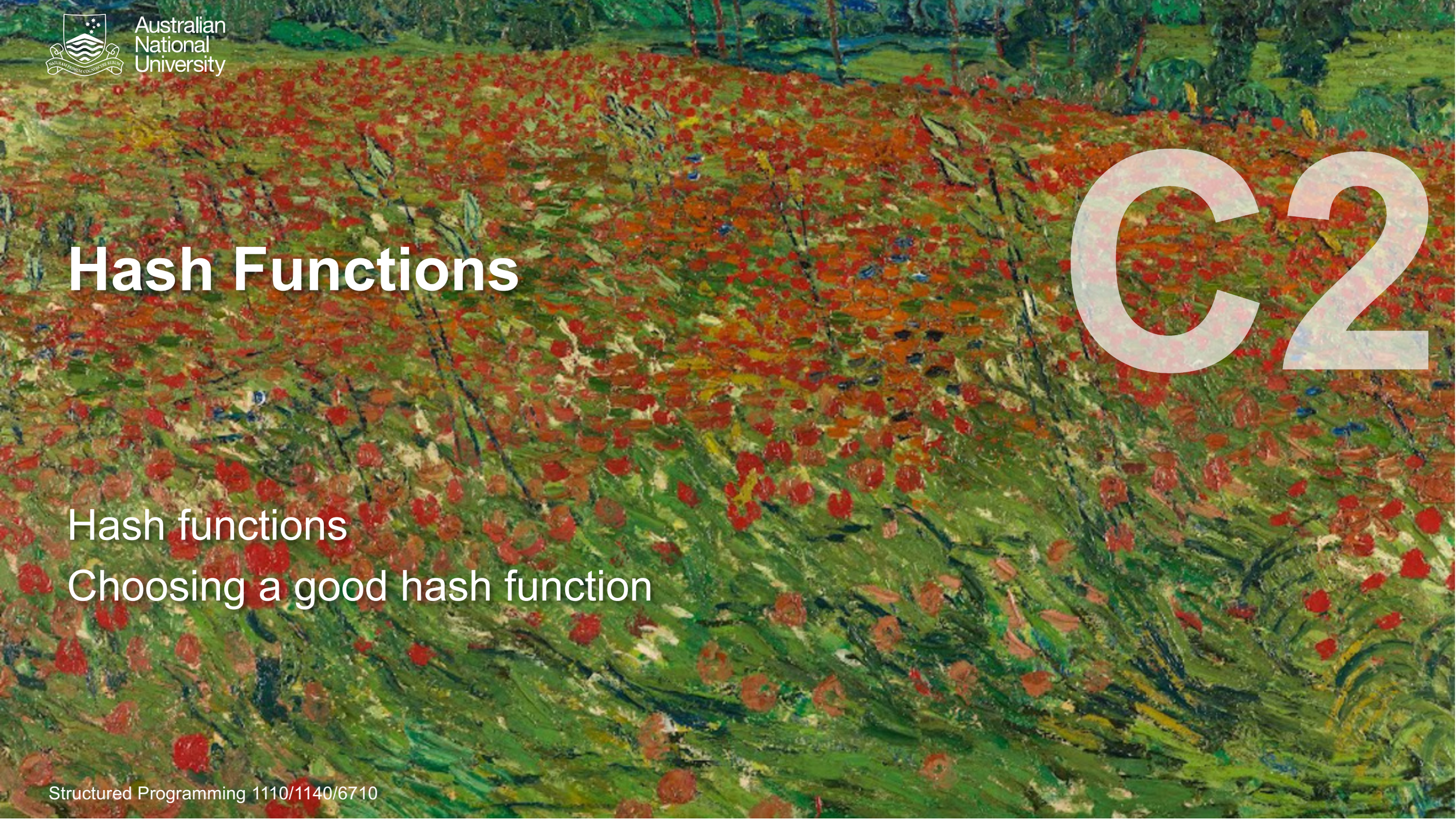
Example: Mergesort (von Neumann, 1945)



Australian National University

(C02) Hash Functions

I would like a recap of hashing



Hash Functions

C2

Hash functions

Choosing a good hash function

Hash Functions

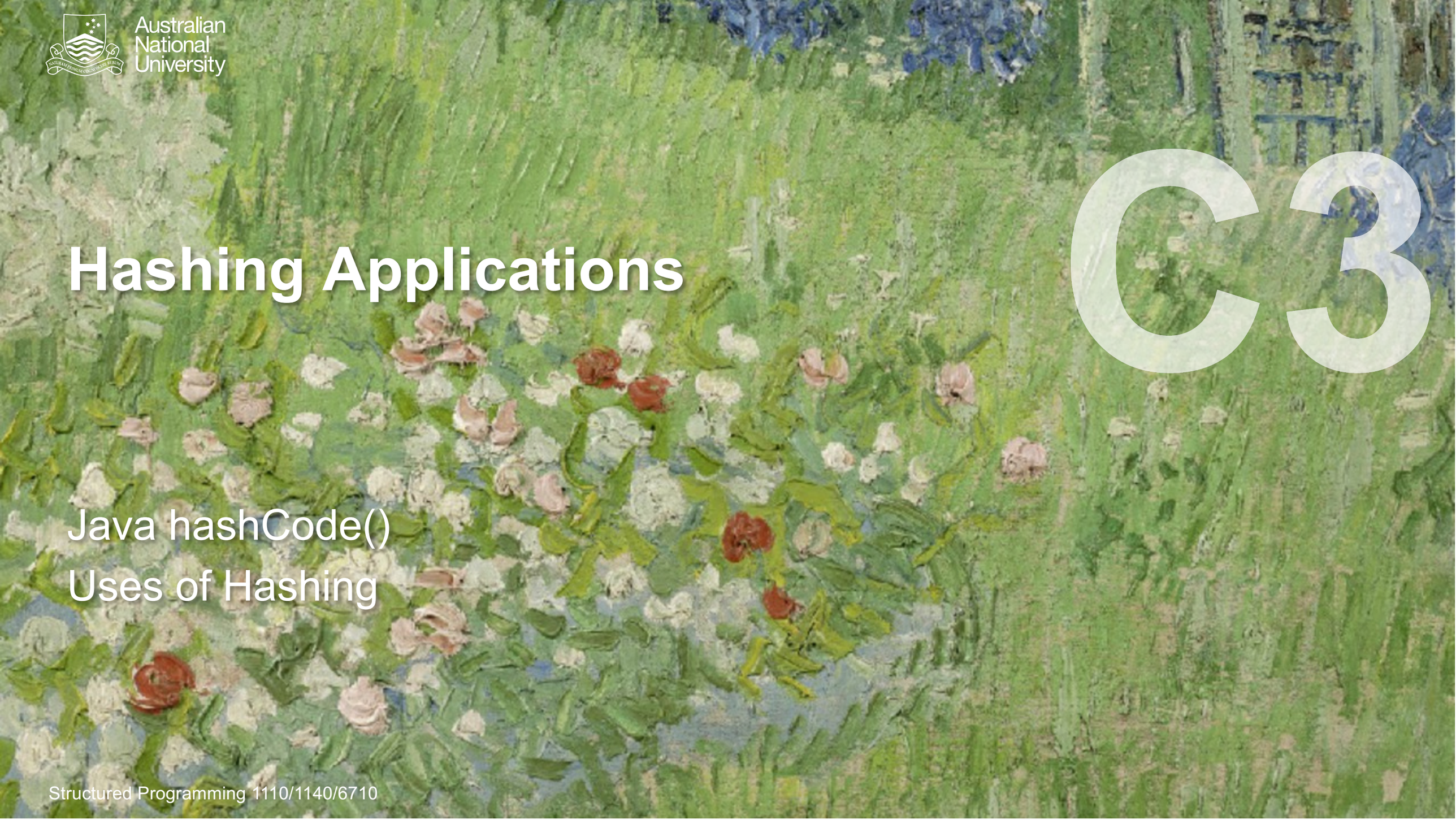
A hash function is a function $f(k)$ that maps a key, k, to a value, $f(k)$, within a prescribed range.

A hash is deterministic. (For a given key, k, $f(k)$ will always be the same).

Choosing a Good Hash Function

A good hash for a given population, P, of keys, $k \in P$, will distribute $f(k)$ evenly within the prescribed range for the hash.

A *perfect hash* will give a unique $f(k)$ for each $k \in P$



Hashing Applications

C3

Java hashCode()

Uses of Hashing

Java `hashCode()`

Java provides a hash code for *every* object

- 32-bit signed integer
- Inherited from `Object`, but may be overwritten
- Objects for which `equals()` is `true` must also have the same `hashCode()`.
- The hash need not be perfect (i.e. two different objects may share the same hash).

Uses of Hashing

- Hash table (a map from key to value)
- Pruning a search
  - Looking for duplicates
  - Looking for similar values
- Compression
  - A hash is typically much more compact that the key
- Correctness
  - Checksums can confirm inequality

Practical Examples…

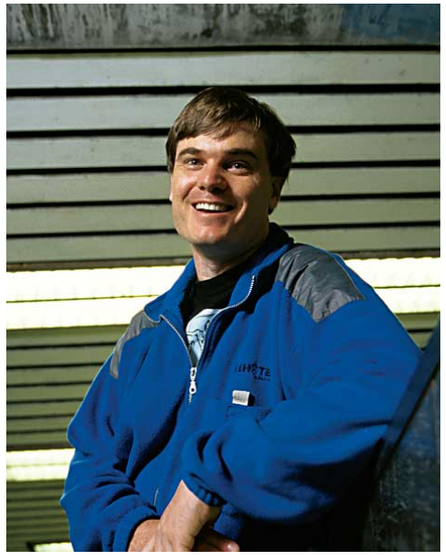
Luhn Algorithm

Used to check for transcription errors in credit cards (last digit checksum).

Hamming Codes

Error correcting codes (as used in EEC memory)

Practical Examples…



rsync (Tridgell)

Synchronize files by (almost) only moving the parts that are different.

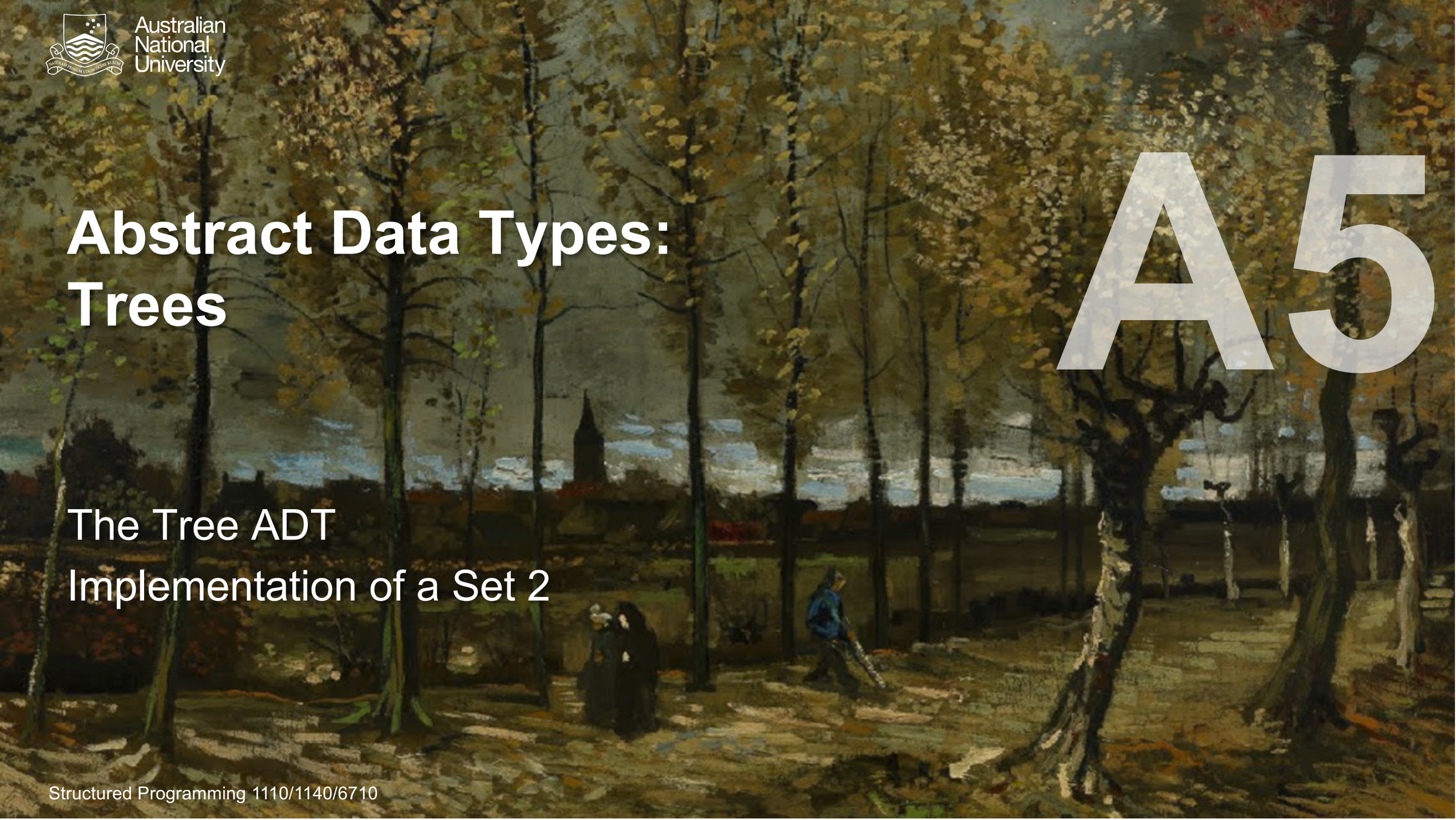
MD5 (Rivest)

Previously used to encode passwords (but no longer).

Australian National University

(A05) Trees (A06) Maps

I would like a recap of Tree and Map ADT



Abstract Data Types: Trees

A5

The Tree ADT

Implementation of a Set 2

The Tree ADT

The **tree** ADT corresponds to a mathematical *tree*. A tree is defined recursively in terms of nodes:

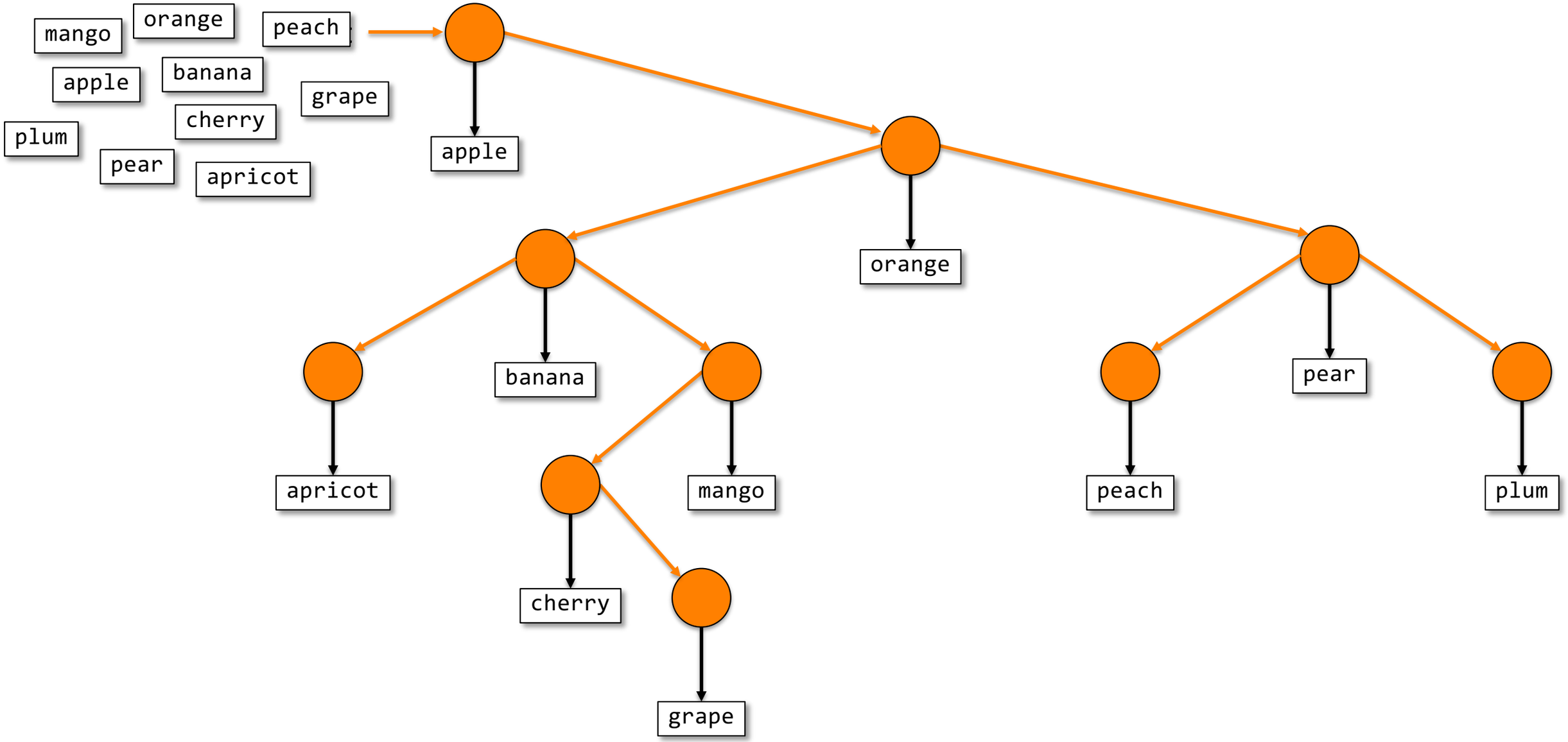
- A tree is a node
- A node contains a *value* and a list of *trees.*
- No node is duplicated.

Binary Search Tree

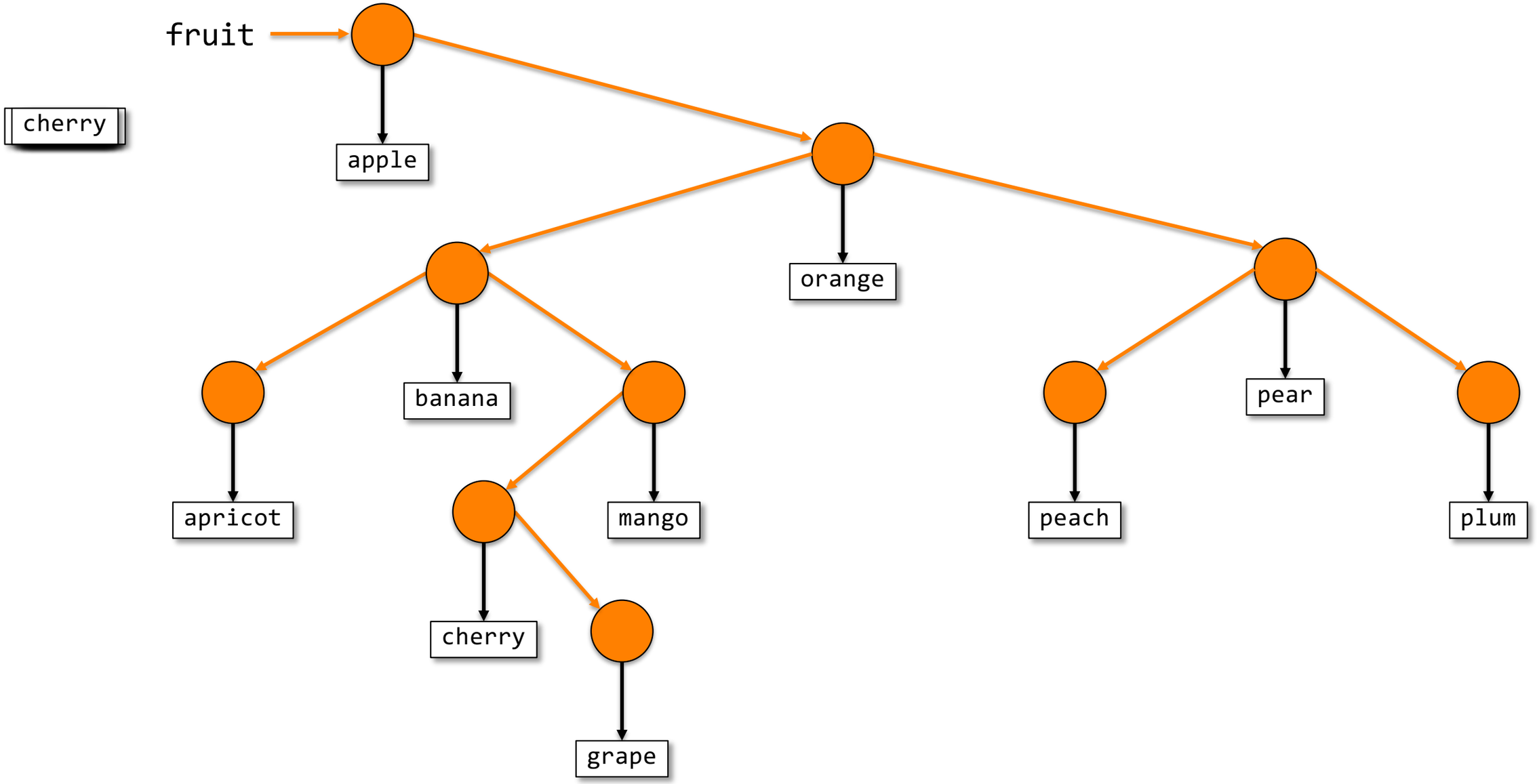
A **binary** search tree is a tree with the following additional properties:

- 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

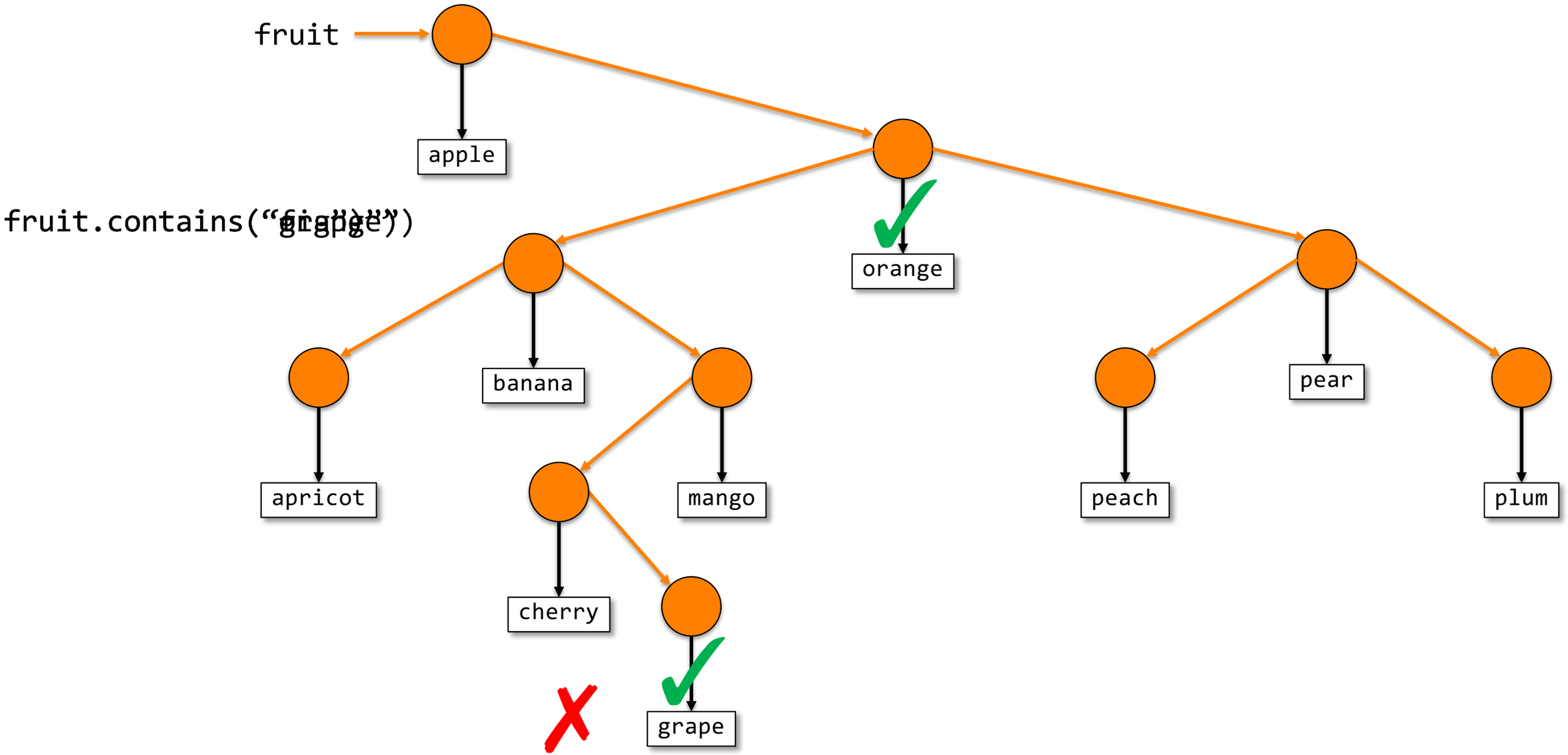
Abstract Data Types: Trees A5



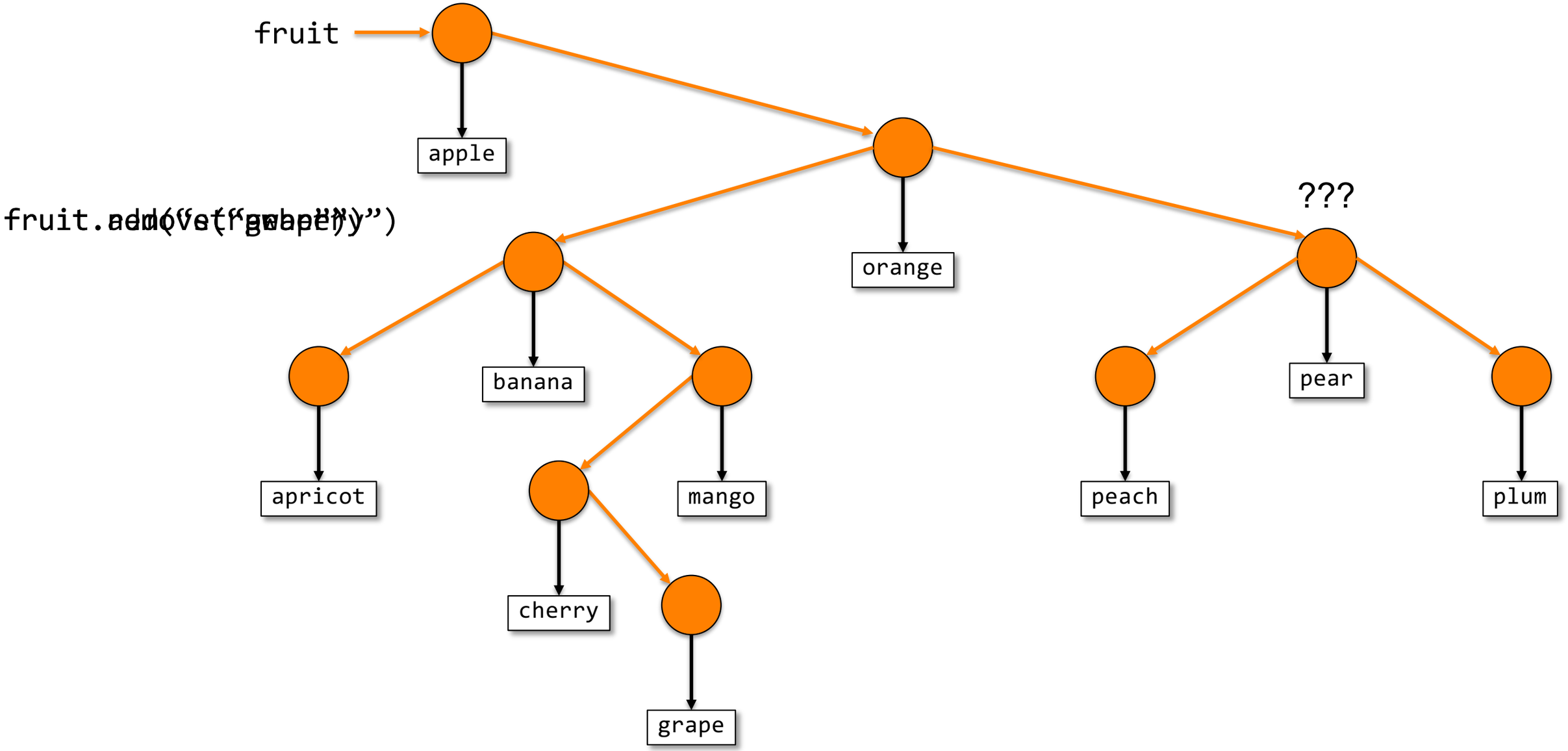
Abstract Data Types: Trees A5



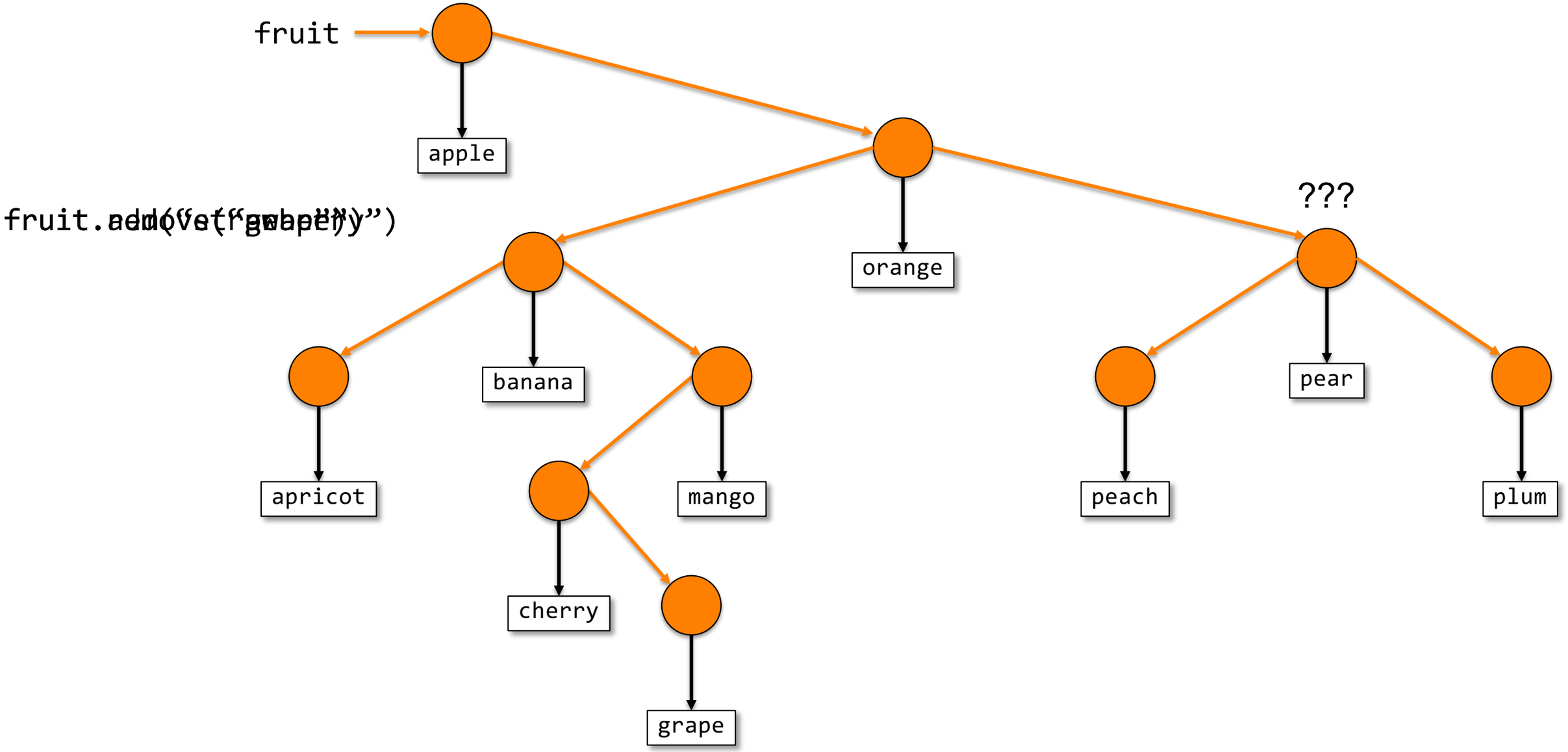
Abstract Data Types: Trees A5



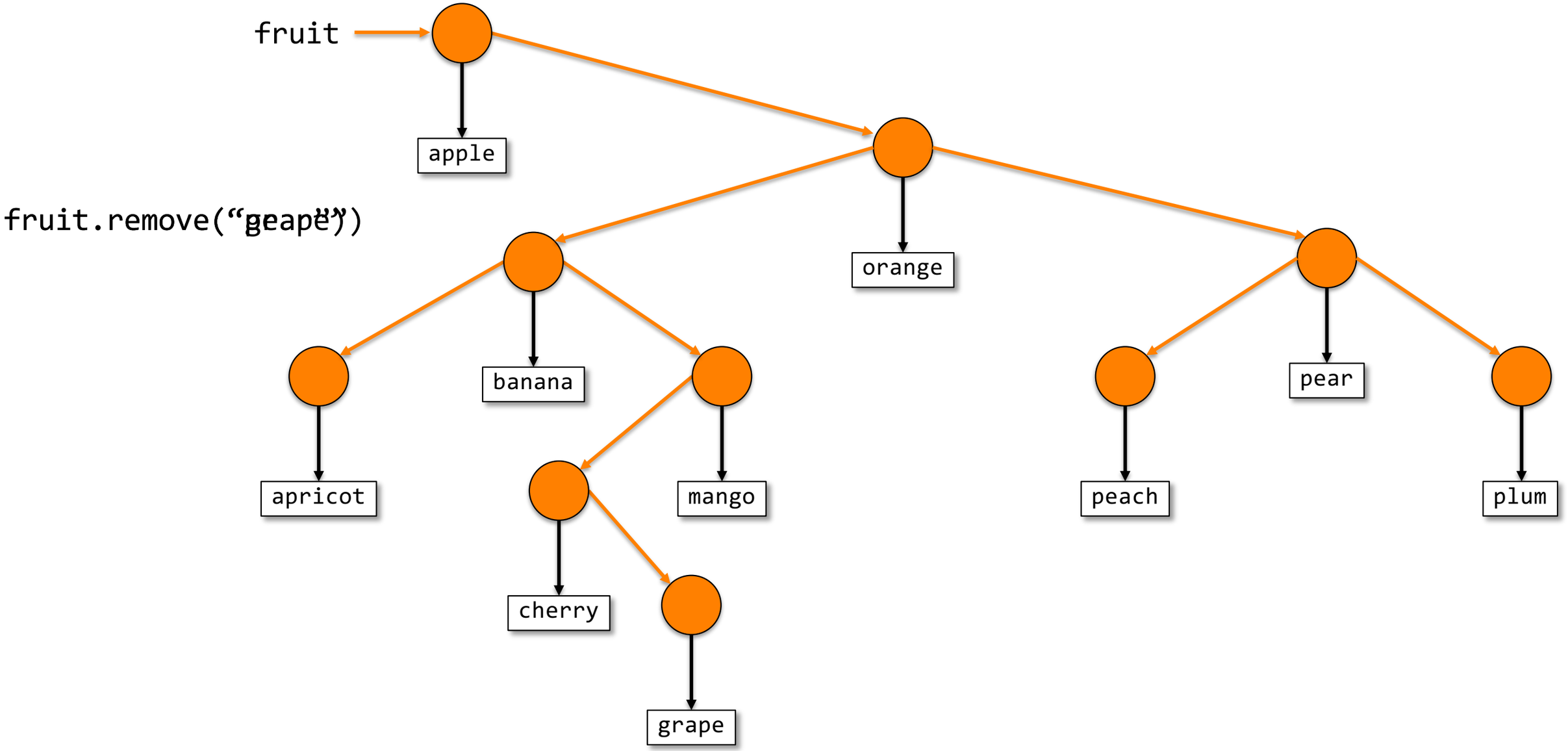
Abstract Data Types: Trees A5

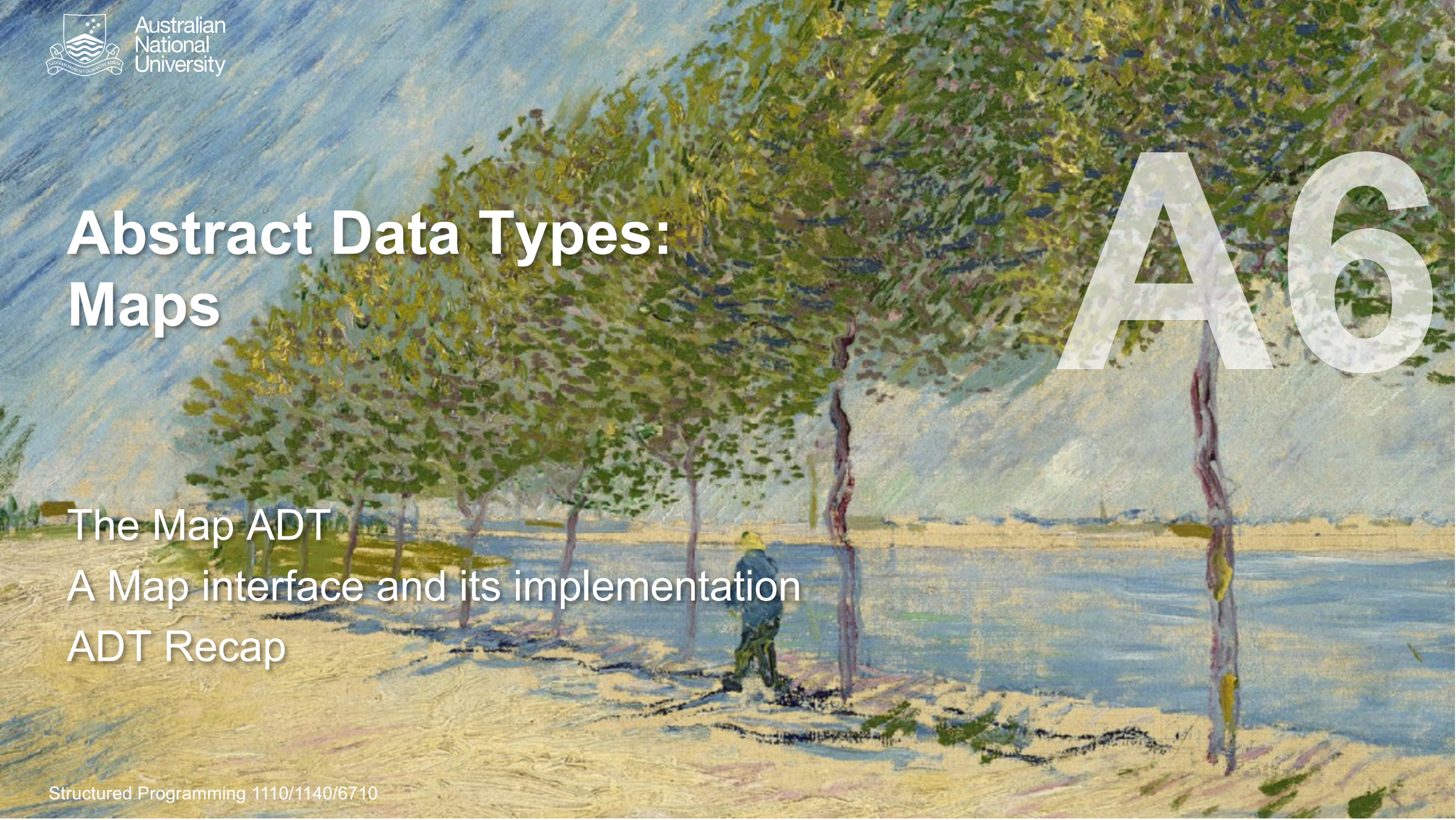


Abstract Data Types: Trees A5



Abstract Data Types: Trees A5





Abstract Data Types: Maps

A6

The Map ADT

A Map interface and its implementation

ADT Recap

ADT Recap

First-principles implementation of three Java container types:

- List
  - `ArrayList`, `LinkedList` implementations *(A1, A2)*
- Set
  - `HashSet`, `BSTSet` implementations *(A3, A4, A5)*
- Map
  - `HashMap`, `BSTMap` implementations *(A6)*

Introduced hash tables, trees *(A4, A5)*

The Map ADT (A.K.A. Associative Array)

A map consists of (key, value) pairs

- Each key may occur only once in the map
- Values are retrieved from the map via the key
- Values may be modified
- Key, value pairs may be removed

Our Map Interface

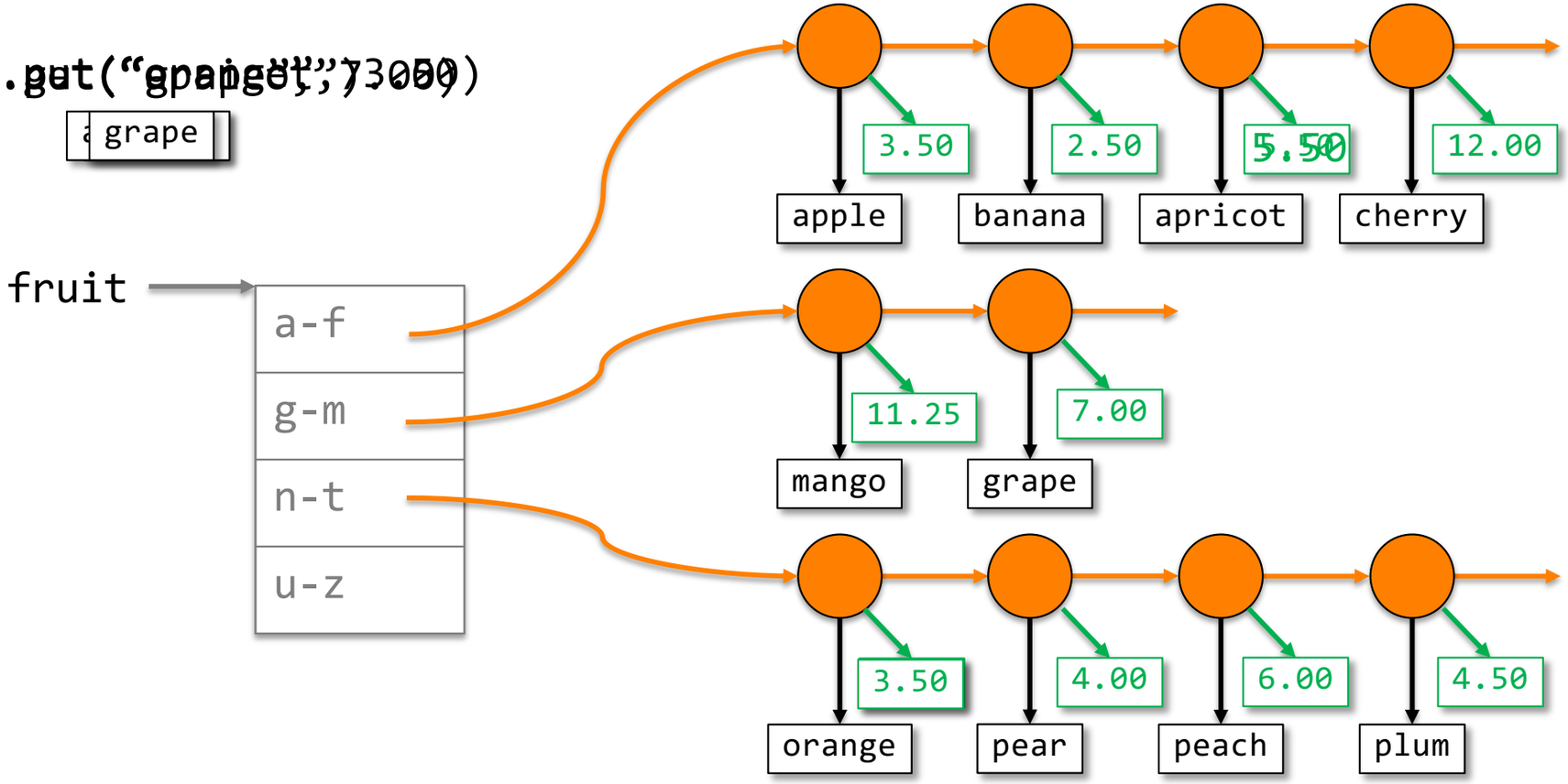
We will explore maps using an interface with the following methods:

```
public void put(K key, V value);
public V get(K key);
public void remove(K key);
public int size();
public String toString();
```

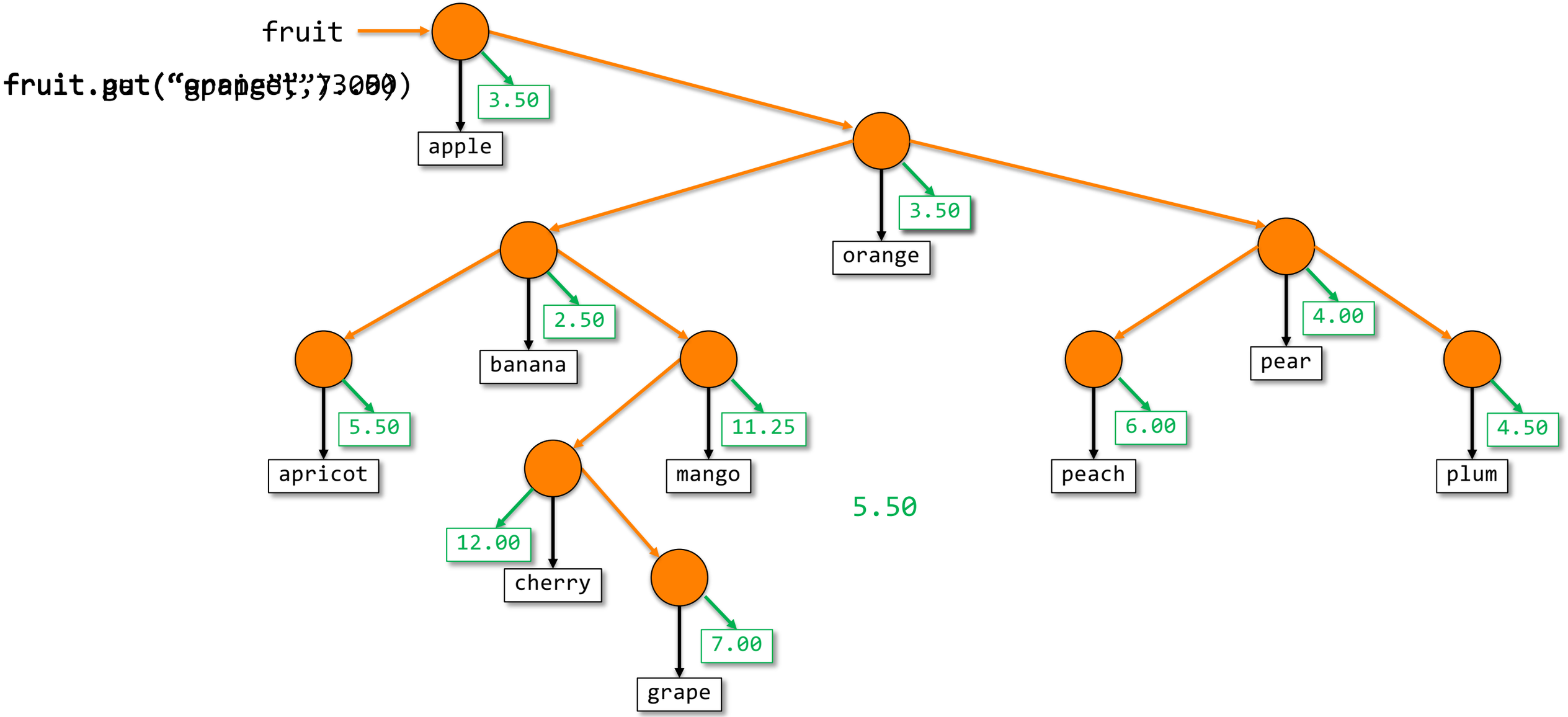
Abstract Data Types: Maps A6

fruit.get("grape")

grape



Abstract Data Types: Maps A6



Australian National University

(001) Object Orientation

What is the advantage of object orientation?

(J01) Imperative Programming Languages

Imperative v functional.

Can they solve the same problems?

Australian National University

Introductory Java 1

J1

Imperative programming languages
Java Standard Library
Types
Hello World


No.7
GEDORE
MADE IN GERMANY

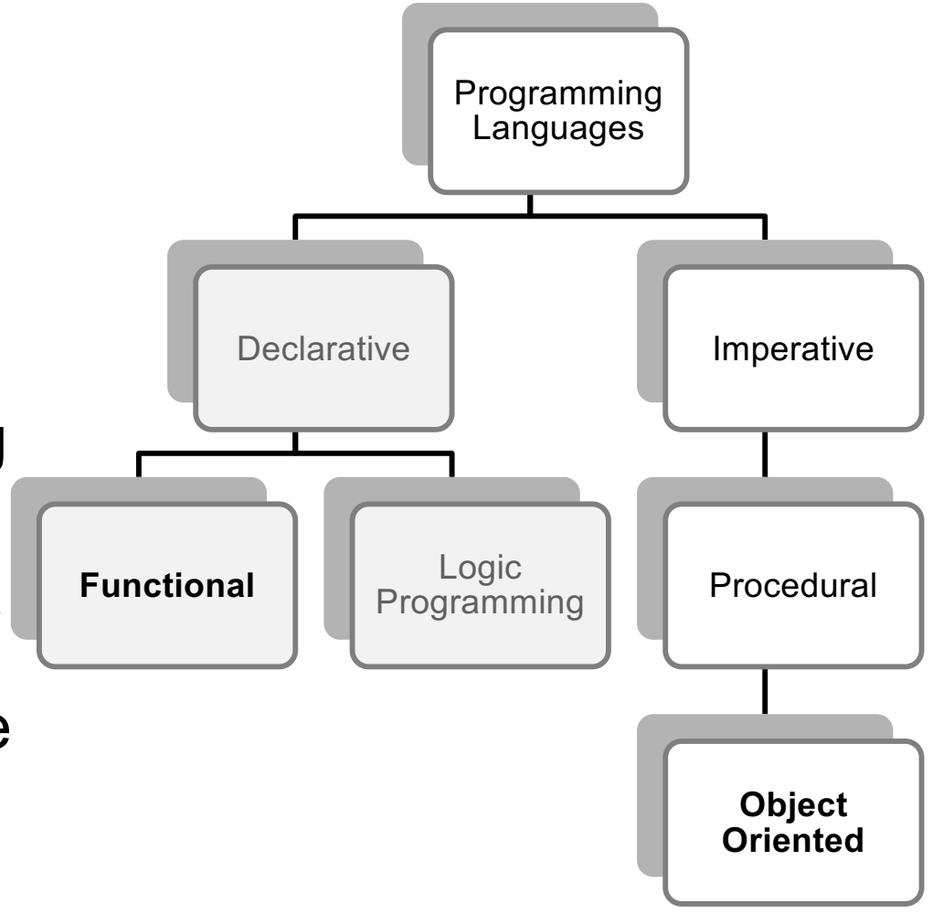
Why Java?

- Learn multiple programming paradigms
- Important example of:
  - Object-oriented programming
  - Large scale programming
  - Programming with a rich standard library

Imperative Programming Languages

Declarative languages describe the desired result without explicitly listing steps required to achieve that goal.

Pure functional languages, like Haskell, will only transform state by using



Imperative languages describe computation in terms of a series of statements that transform state.

Object-oriented languages use structured (procedural) code, tightly

Imperative Programming Languages

- Sequence
- Selection
- Iteration

Object Oriented Programming Languages

- Structured code
- Code (*behavior*) tightly coupled with data (*state*) that it manipulates

Australian National University

(003) Interfaces (005) Abstract Classes

Please explain the "power" and "goodness" of Java interfaces and abstract classes.
w.r.t. software architecture, programming, and utility.

Australian National University

Interfaces

03

Interfaces

An interface can be thought of as a contract.
A class which implements an interface *must* provide the specified functionality. Compared to a class, an interface:

- Uses `interface` keyword rather than `class`
- Cannot be instantiated (can't be created with `new`)
- Can *only* contain constants, method signatures (not the bodies), nested types
  - (Java 8 allows `default` and `static` methods)
- Classes implement interfaces via `implements` keyword

Interfaces as Types

An interface can be used as a type

- A variable declared with an interface type can hold a reference to a object of any class that implements that interface.

Australian National University

Inheritance 2

05

java.lang.Object
Final classes, methods and fields
Abstract classes and methods

Object as superclass

In Java all classes ultimately inherit from **one** root class: `java.lang.Object`. Implemented methods:

- `clone()` *returns copy of object*
- `equals(Object obj)` *establishes equivalence*
- `finalize()` *called by GC before reclaiming*
- `getClass()` *returns runtime class of the object*
- `hashCode()` *returns a hash code for the object*
- `toString()` *returns string representation of object*

Final Classes and Methods

The `final` keyword in a class declaration states that the class *may not* be subclassed.

The `final` keyword in a method declaration states that the method *may not* be overridden.

Abstract Classes and Methods

The `abstract` keyword in a class declaration states that the class is abstract, and therefore cannot be instantiated (its subclasses may be, if they are not abstract).

The `abstract` keyword in a method declaration states that the method declaration is abstract; the implementation must be provided by a subclass.

Australian National University

(C4) Files

What is the important syntax for Files?

Australian National University

(C7) Threads

Can you explain why synchronized was necessary in the in-lecture example?