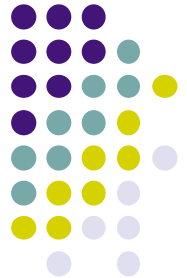
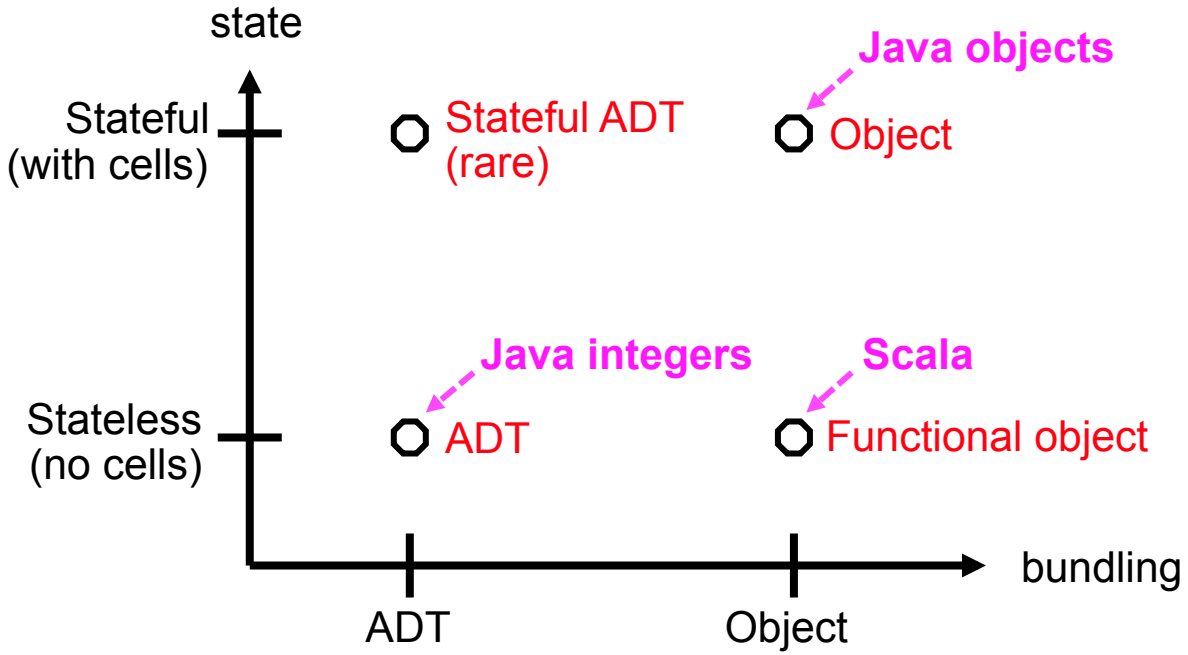
Four ways to do data abstraction



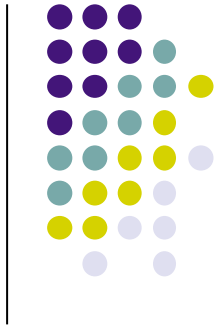
- We have seen two ways to make data abstractions:
  - Abstract data types (without state)
  - Objects (with state)
- There are two more ways to build data abstractions
  - Abstract data types with state (stateful ADTs)
  - Objects without state (functional objects)
- This gives four ways in all
  - Let's take a look at the two additional ways
  - And then we'll conclude this lesson on data abstraction

Four ways to do data abstraction



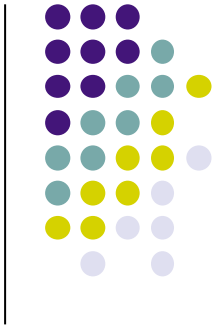
- Objects (with state) and ADTs (stateless) are popular
- Functional objects are less popular (except in Scala)
- Stateful ADTs are rarely used

The two less-used data abstractions



- A functional object is possible
  - Functional objects are immutable; invoking an object returns another object with a new value
  - Functional objects are becoming more popular because of Scala
- A stateful ADT is possible
  - Stateful ADTs were much used in the C language (although without enforced encapsulation, since it is impossible in C)
  - They are also used in other languages (e.g., classes with static attributes in Java)
- Let's take a closer look at how to build them

A functional object

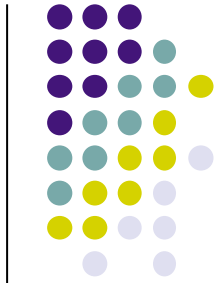


- We can implement the stack as a functional object:

```
local
   fun {StackObject S}
      fun {Push E} {StackObject E|S} end
      fun {Pop S1}
         case S of X|T then S1={StackObject T} X end end
      fun {IsEmpty} S==nil end
   in stack(push:Push pop:Pop isEmpty:IsEmpty) end
in
   fun {NewStack} {StackObject nil} end
end
```

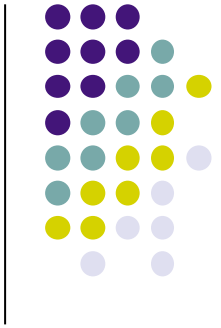
- This uses no cells and no secure wrappers. It's the simplest of all our data abstractions since it only needs higher-order programming.

Functional objects in Scala



- Scala is a hybrid functional-object language: it supports both the functional and object-oriented paradigms
- In Scala we can define an immutable object that returns another immutable object
  - For example, a RationalNumber class whose instances are rational numbers (and therefore immutable)
  - Adding two rational numbers returns another rational number
- Immutable objects are functional objects
  - The advantage is that they cannot be changed (the same advantage of any functional data structure)

A stateful ADT

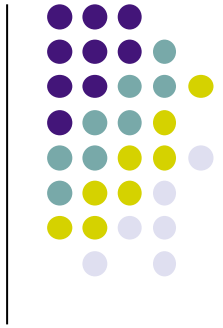


- Finally, let us implement our trusty stack as a stateful ADT:

```
local Wrap Unwrap
   {NewWrapper Wrap Unwrap}
   fun {NewStack} {Wrap {NewCell nil}} end
   proc {Push S E} C={Unwrap S} in C:=E|@C end
   fun {Pop S} C={Unwrap S} in
      case @C of X|S1 then C:=S1 X end
   end
   fun {IsEmpty S} @{Unwrap S}==nil end
in
   Stack=stack(new:NewStack push:Push pop:Pop isEmpty:IsEmpty)
end
```

- This uses both a cell and a secure wrapper. Note that Push, Pop, and IsEmpty do not need Wrap! They modify the stack state by updating the cell *inside* the secure wrapper.

Conclusion



- Data abstractions are a key concept needed for building large programs with confidence
  - Data abstractions are built on top of higher-order programming, static scoping, explicit state, records, and secret keys
  - Data abstractions are defined precisely in terms of these concepts; our definitions give the semantics of data abstractions
- There are four kinds of data abstraction, along two axes: objects versus ADTs on one axis and stateful versus stateless on the other
  - Two kinds are more visible than the others, but the others also have their uses (for example, functional objects are used in Scala)
- Modern programming languages strongly support data abstractions
  - They support much more than just objects; it is more correct to consider them data abstraction languages and not just object-oriented languages